Distributed Machine Learning

CPSC 416
Muhammad Shayan
31 March 2022

Introduction

BS Computer Science (2017)

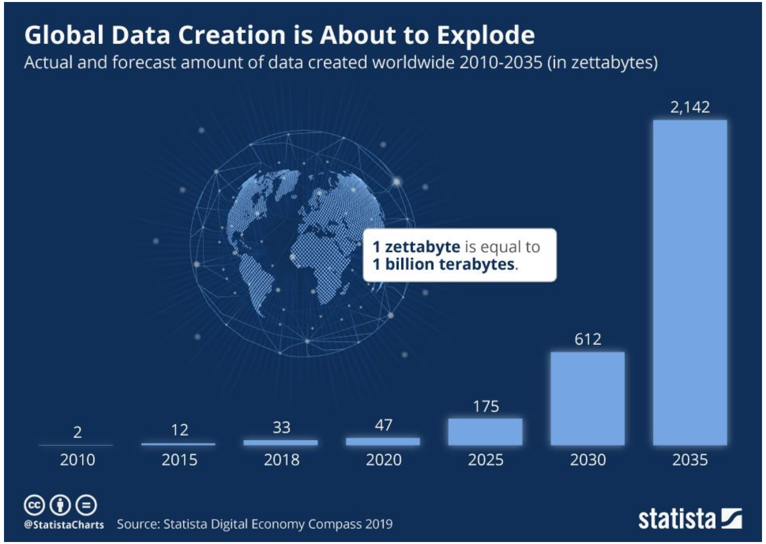
Data Scientist (2017)

MS Computer Science (2018 -2020) => Building a distributed P2P ML system

Machine Learning and Backend Engineer (2020 - Current)

Data is growing at a rapid rate ....

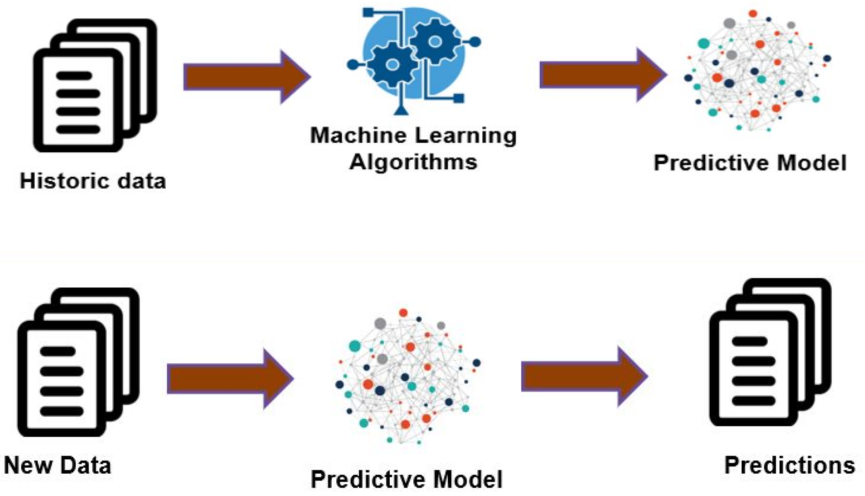
- Data has grown at an unprecedented rate in the last century
- Has a lot of hidden insights
- Machine Learning helps us extract insights and learn patterns from this data



Machine Learning Systems are everywhere

- To process this data, a large number for machine learning systems have emerged
    - These systems track and analyze all the data they can get

What does a machine learning system do?



Training:

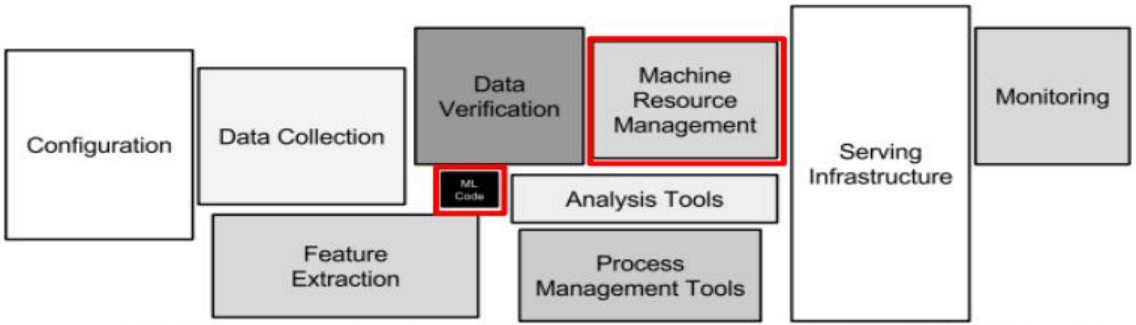
Learning a mathematical model from historical data

Inference:

Using the mathematical model to make predictions on unseen data

Why study ML from a systems perspective?

- The ML code is just a small portion of a complete ML system.
- To be able to build ML models and run them in production, you need a lot of systems knowledge



In this lecture ....

How do distributed systems play a role in building and deploying machine learning models ...?

Outline

ML training

- Background
- Parameter Server
- Federated Learning
- Peer to Peer approaches

ML Inference

- Containers
- Microservices
- Example ML System: Korbit AI

Maintaining ML Models in production

Outline

ML training

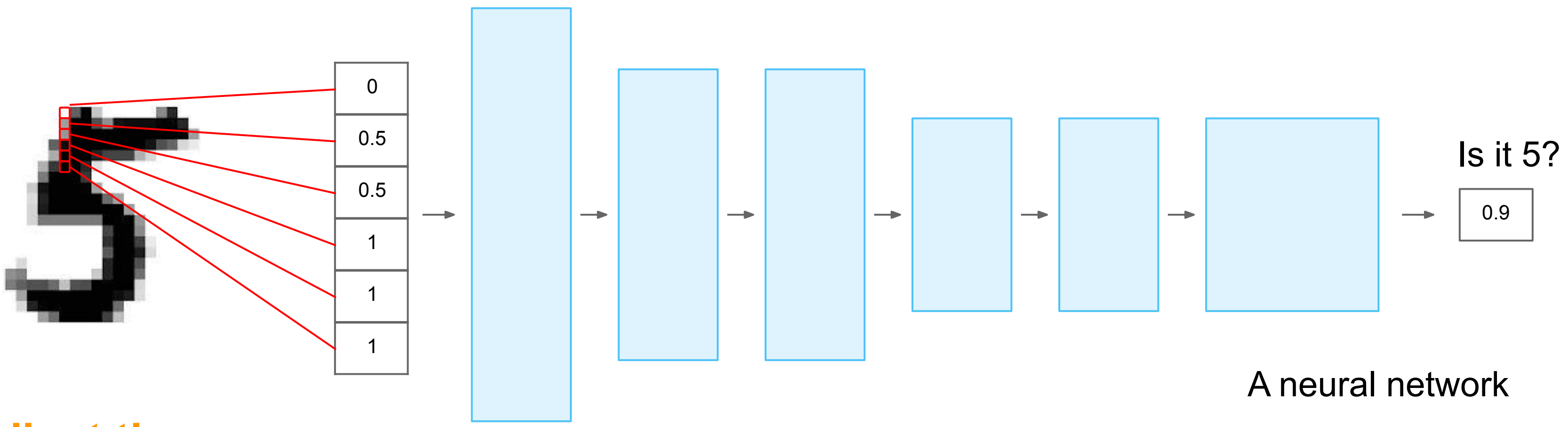
- Background
- Parameter Server
- Federated Learning
- Peer to Peer approaches

ML Inference

- Containers
- Microservices
- Example ML System: Korbit AI

Maintaining ML Models in production

Background - What is an ML model?



Adjust these

$$\mathbf{f}(\text{input}, \text{parameters}) = \text{output}$$

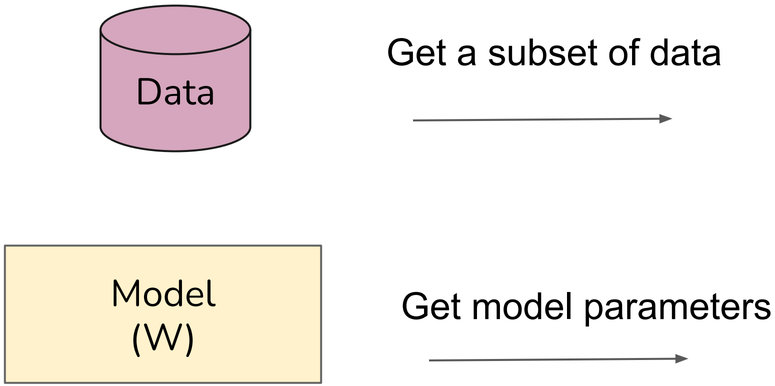
$$\mathbf{loss}(\text{parameters}) = 1/n \sum_i \text{difference}(\mathbf{f}(\text{input}_i, \text{parameters}), \text{desired}_i)$$

to minimize this

How do we train ML models?

Stochastic gradient descent => A general purpose algorithm for training

- Works for many model types (regression, neural networks etc)



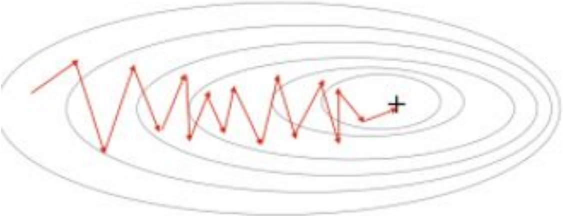
How do we train ML models?

Stochastic gradient descent => A general purpose algorithm for training

- Works for many model types (regression, neural networks etc)

Data

Get a subset of data →



Model
(W)

Get model parameters →

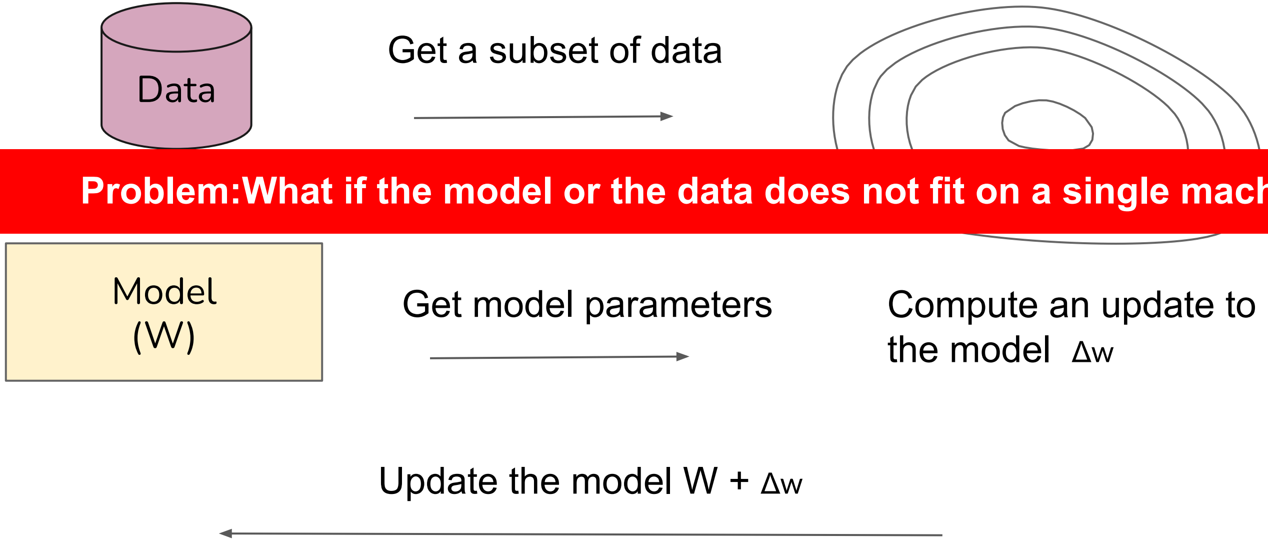
Compute an update
(gradient) to the model
Δw

← Update the model $W + \Delta w$

How do we train ML models?

Stochastic gradient descent => A general purpose algorithm for training

- Works for many model types (regression, neural networks etc)



Problem:What if the model or the data does not fit on a single machine?

Outline

ML training

- Background
- Parameter Server
- Federated Learning
- Peer to Peer approaches

ML Inference

- Containers
- Microservices
- Example ML System: Korbit AI

Maintaining ML Models in production

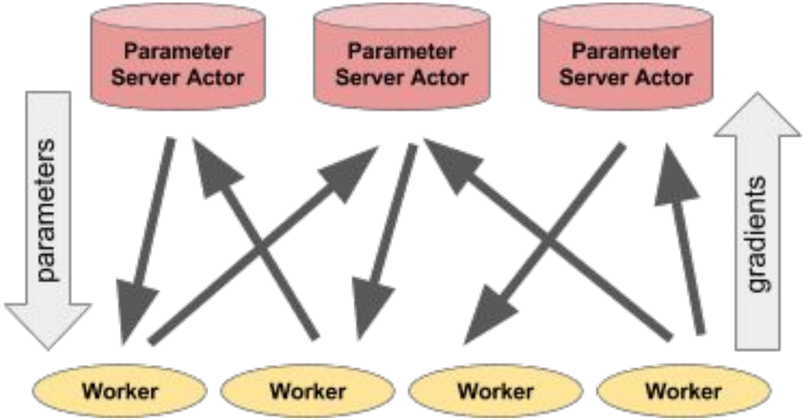
Parameter Server Architecture

- Nodes divided into workers/ servers

- Workers compute updates from the data

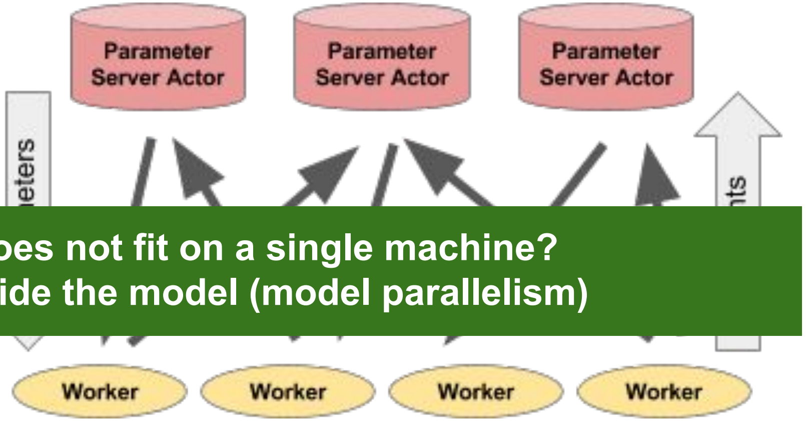
- Server responsible for coordinating the training process.

- Can have multiple servers for migration/replication purposes



Parameter Server Architecture

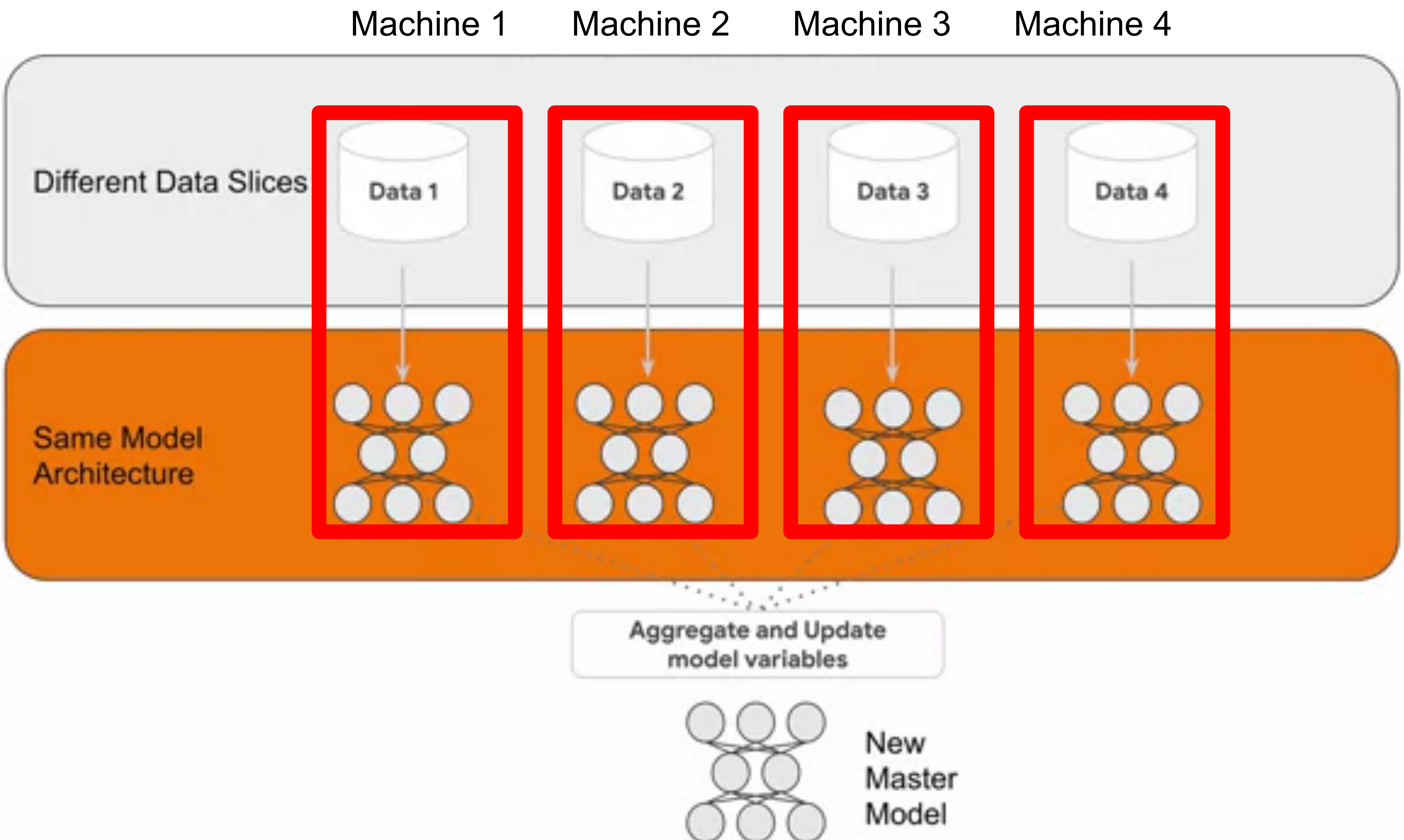
- Nodes divided into workers/ servers
- Workers compute updates from the data



Problem:What if the model or the data does not fit on a single machine?
- Divide the data (data parallelism) or divide the model (model parallelism)

- Can have multiple servers for migration/replication purposes

Data parallelism



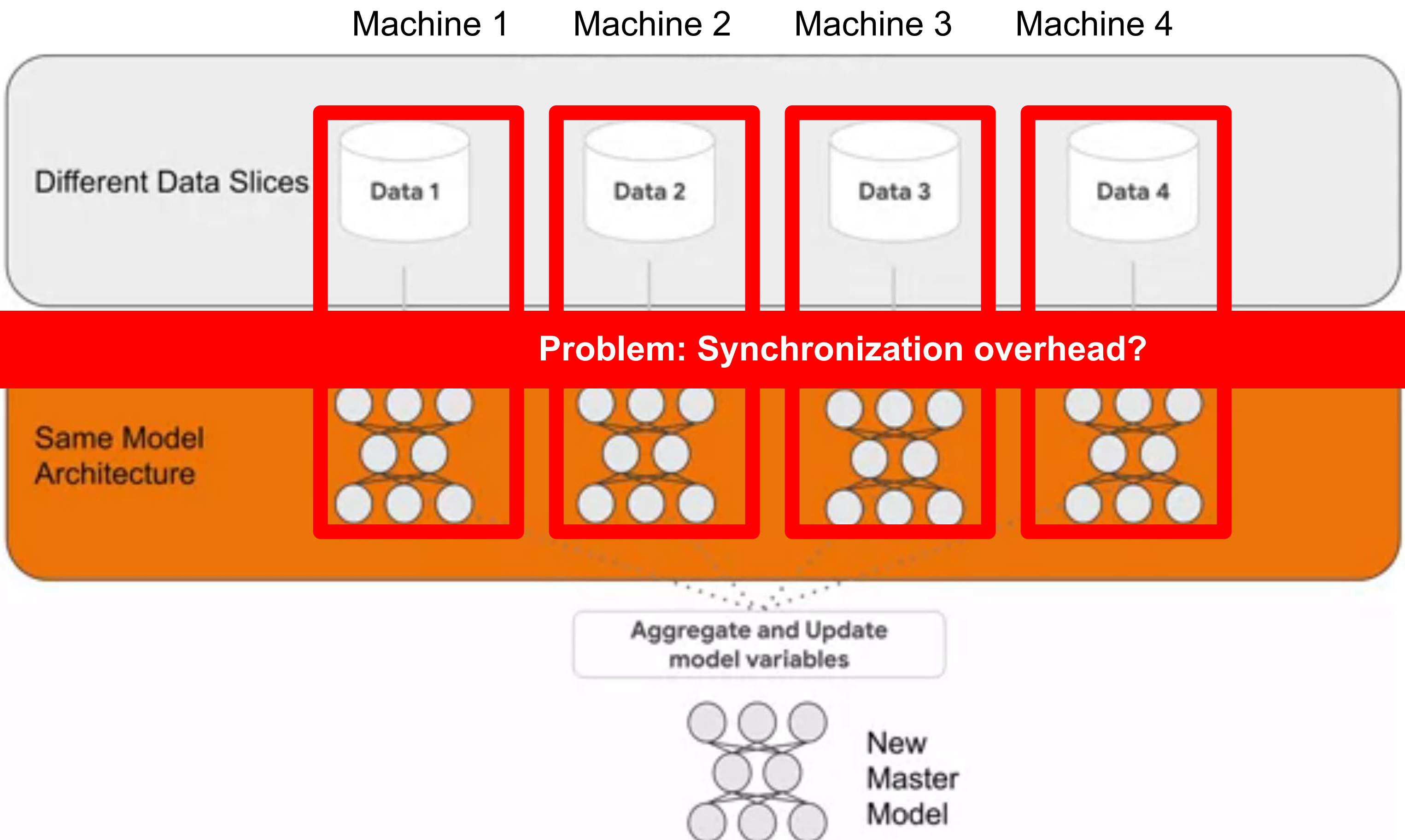
Step 1: Each worker holds a subset of the data and a copy of the model itself

Step 2: Compute an update on the model and send to server

Step 3: Central server aggregates updates at end and shares updated model with workers

Step 4: Repeat

Data parallelism



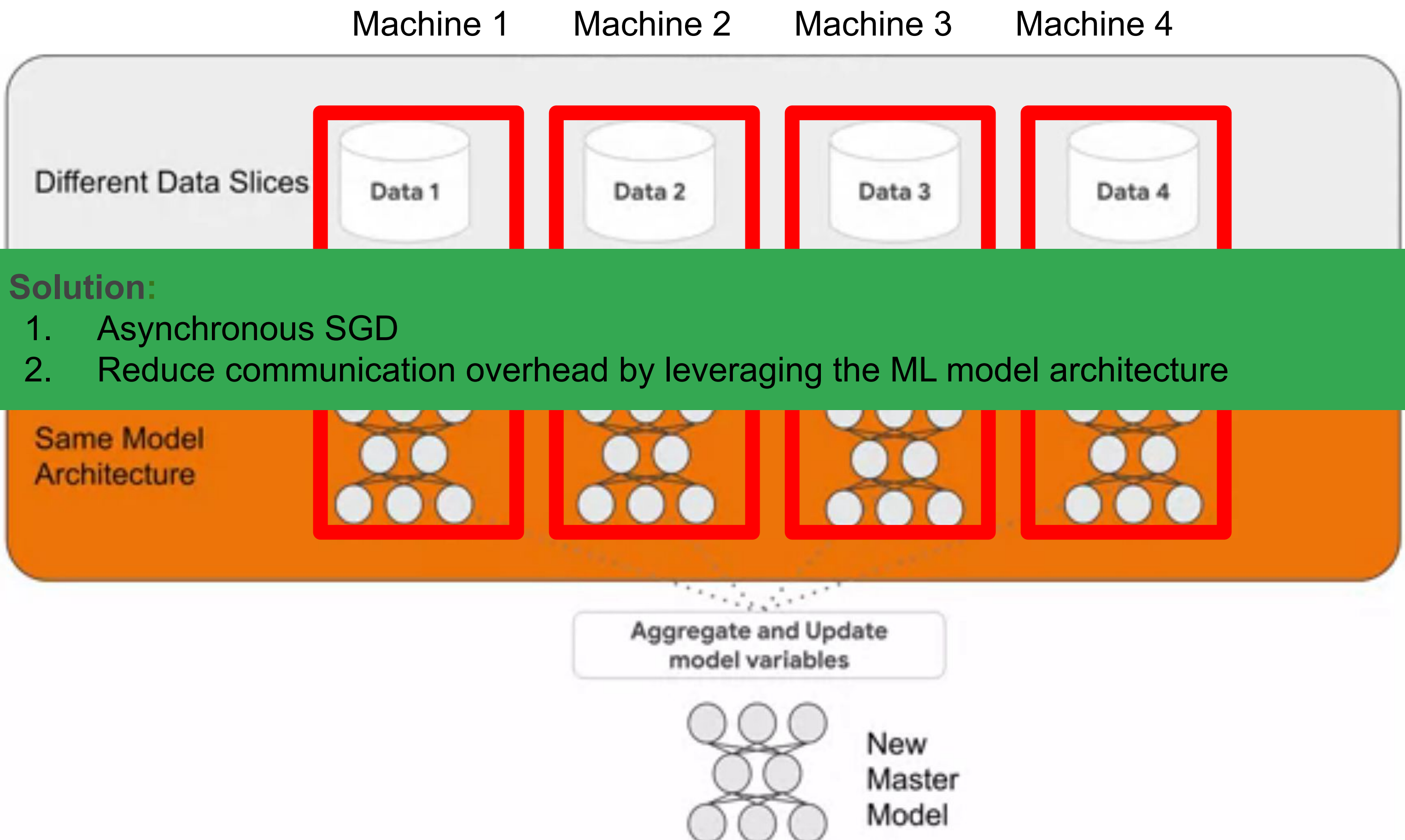
Step 1: Each worker holds a subset of the data and a copy of the model itself

and send to server

Step 3: Central server aggregates updates at end

Step 4: Repeat

Data parallelism



Step 1: Each worker holds a subset of the data and a copy of the model itself

and send to server

Step 3: Central server aggregates updates at end

Step 4: Repeat

Solution:

1. Asynchronous SGD
2. Reduce communication overhead by leveraging the ML model architecture

Data parallelism

Machine 1 Machine 2 Machine 3 Machine 4

Different Da

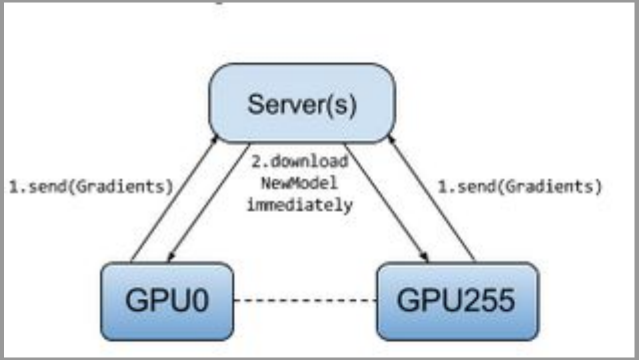
Same Mod
Architectur

Asynchronous SGD/ Hogwild:

Worker does not wait for synchronization at each step. Can only work in some learning settings.

Intuition: Different updates are **sparse** i.e only affects a small subset of the parameters.

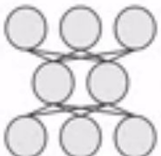
Can be used with sparse SVM's, matrix completion problems etc.



ch worker
set of the
a and a
model

ll the
el from
eter server
te an

ntral
regates
all updated gradients
as the end of each
iteration



New
Master
Model

Data parallelism

Machine 1 Machine 2 Machine 3 Machine 4

Different Da

Same Mod
Architectur

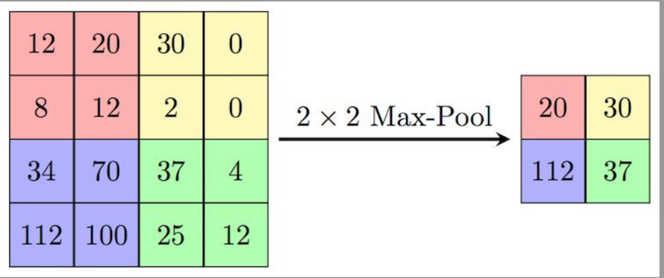
Reduce communication overhead:

- If updates are sparse, only transmit the changed parameters
- Max pooling, Maxout units and convolution layers
  - To encourage sparsity i.e making most parameters updates 0
- Changes might not be feasible given your ML architecture!

$$y = w_1 * x_1 + w_2 * x_2 + .. + w_N * x_N$$

The key-value store could look like

$$\{w_1 : 0.2, w_2 : 0.01, \ldots w_N : 0\}$$

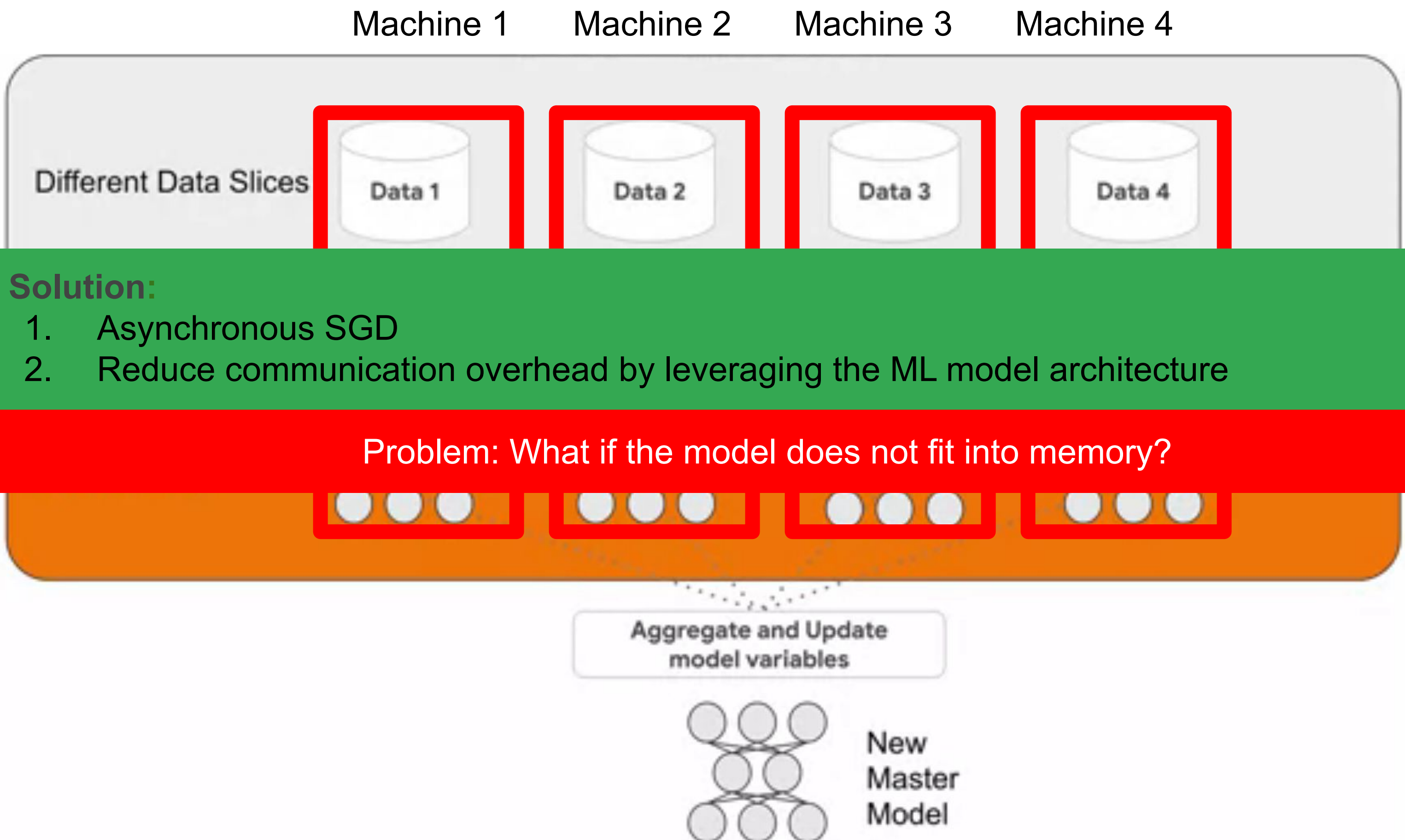


ch worker
set of the
a and a
model

ll the
el from
eter server
te an

ntral
regates
all updated gradients
as the end of each
iteration

Data parallelism



Solution:

1. Asynchronous SGD
2. Reduce communication overhead by leveraging the ML model architecture

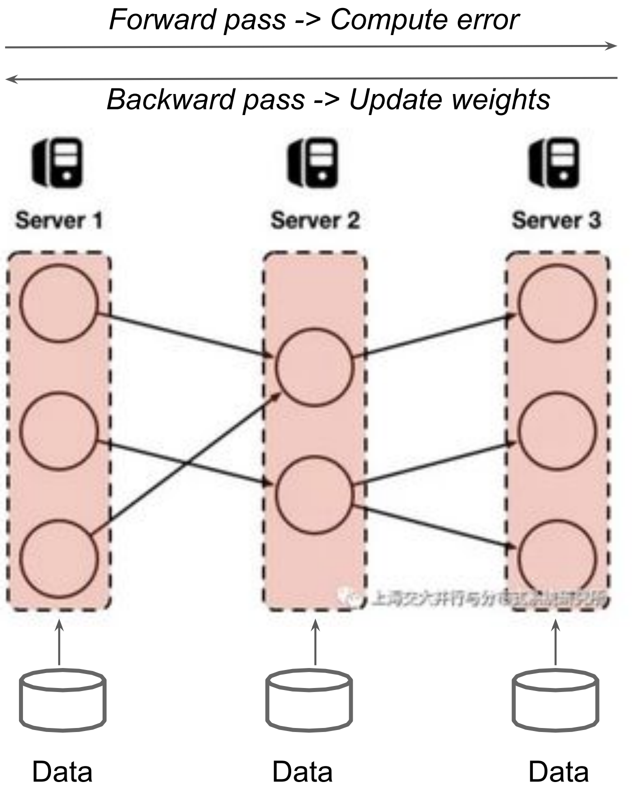
Problem: What if the model does not fit into memory?

Step 1: Each worker holds a subset of the training data and a copy of the model

update

Step 3: Central server aggregates all updated gradients as the end of each iteration

Model parallelism



If model is huge, might not fit into memory/GPU of one machine.

Good idea to split model among multiple machines/GPU’s.

The forward pass and backward pass on the model computes is done in serial across machines/GPUs

Models are generally split in a fashion such that there are least dependencies

When to use data/model parallelism?

- If GPUs are not saturated and have some free capacity (not all cores are running), then model parallelism will be slow. Use data parallelism instead!

- If the model does not fit into memory, model parallelism is the obvious choice.

- Communication overhead can be reduced in data parallelism but not in model parallelism.

When to use data/model parallelism?

- If GPUs are not saturated and have some free capacity (not all cores are running), then model parallelism will be slow. Use data parallelism instead!

Problem:What if data does not fit into memory nor does the model?`

- If the model does not fit into memory, model parallelism is the obvious choice.

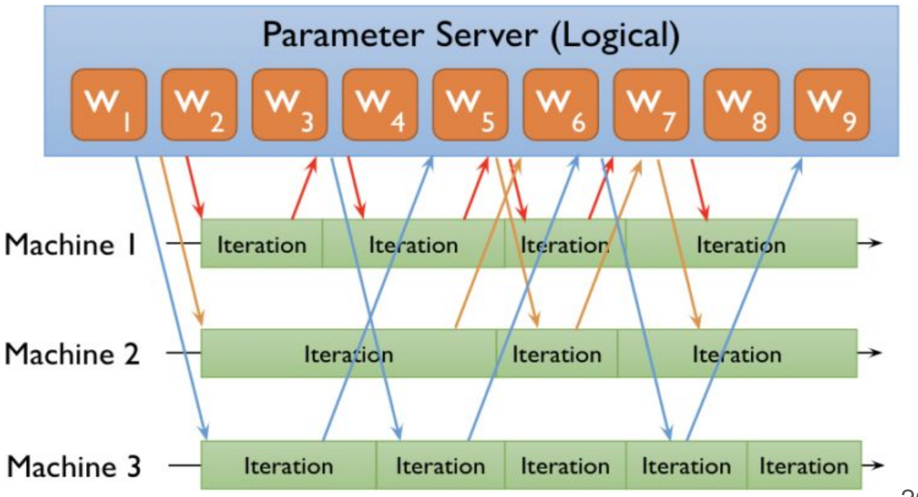
- Communication overhead can be reduced in data parallelism but not in model parallelism.

Google's DistBelief system

Combine data, model parallelism and asynchronous SGD

Workers asynchronously fetch model parameters and push gradients to the parameter server.

The parameters are sharded across multiple parameter servers



A complex system developed for a specific use-case (ImageNet)

Distributed ML training - Key takeaways

- When model/data does not fit into single machine, use data parallelism or model parallelism or a mixture of both.
    - Data parallelism more widely used.

- Parameter server widely used with data parallelism.
    - Though we do have architectures that distribute the aggregation task on all machines. AllReduce?

- Prefer Synchronous execution over Asynchronous execution. This is mostly due to concerns about model stability and convergence.

Distributed ML training - Key takeaways

- When model/data does not fit into single machine, use data parallelism or model parallelism or a mixture of both.
  - Data parallelism more widely used.

What if due to privacy considerations, we cannot collect the data and centralize it?

Example: Gboard

Distributed ML training - Key takeaways

- When model/data does not fit into single machine, use data parallelism or model parallelism or a mixture of both.
  - Data parallelism more widely used.

Solution: Federated Learning

- Prefer Synchronous execution over Asynchronous execution. This is mostly due to concerns about model stability and convergence.

Outline

ML training

- Background
- Parameter Server
- Federated Learning
- Peer to Peer approaches

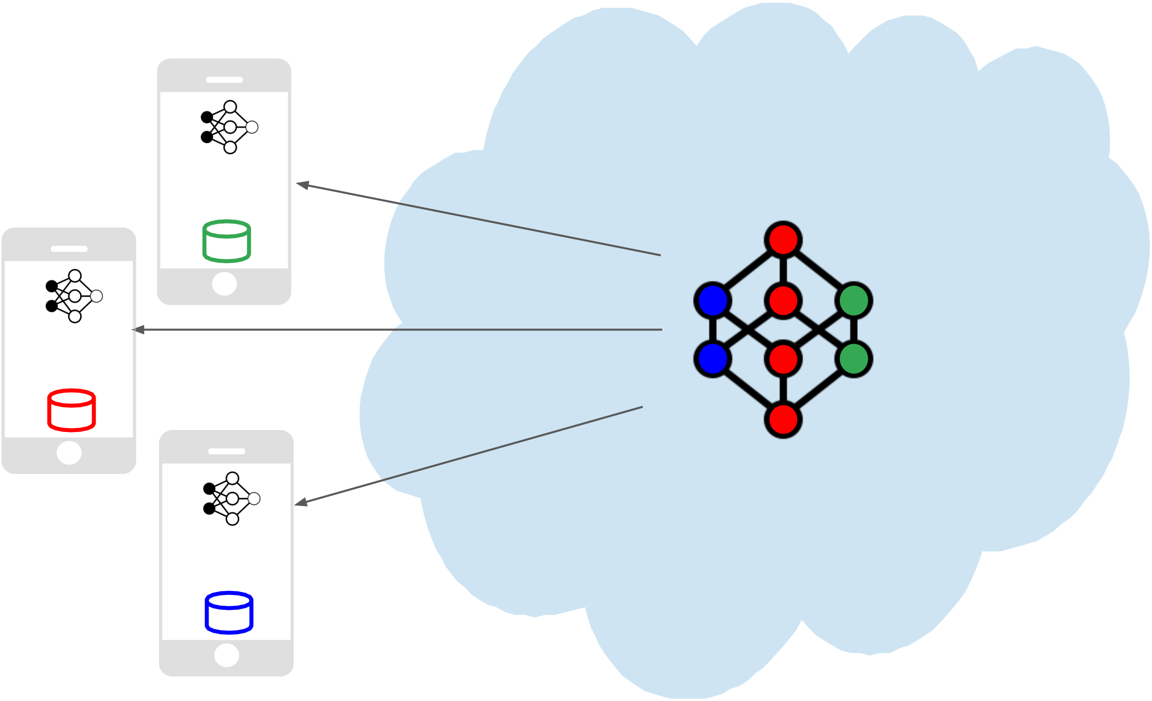
ML Inference

- Containers
- Microservices
- Example ML System: Korbit AI

Maintaining ML Models in production

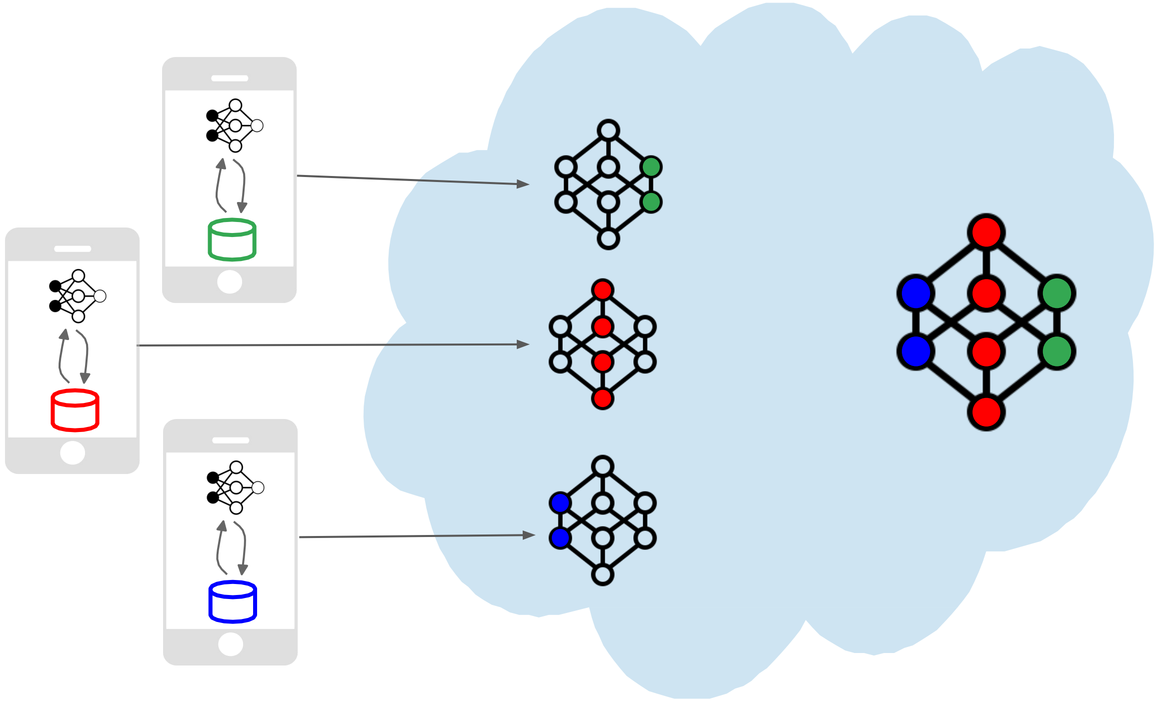
Federated Learning

1. Each client downloads model parameters from central server



Federated Learning

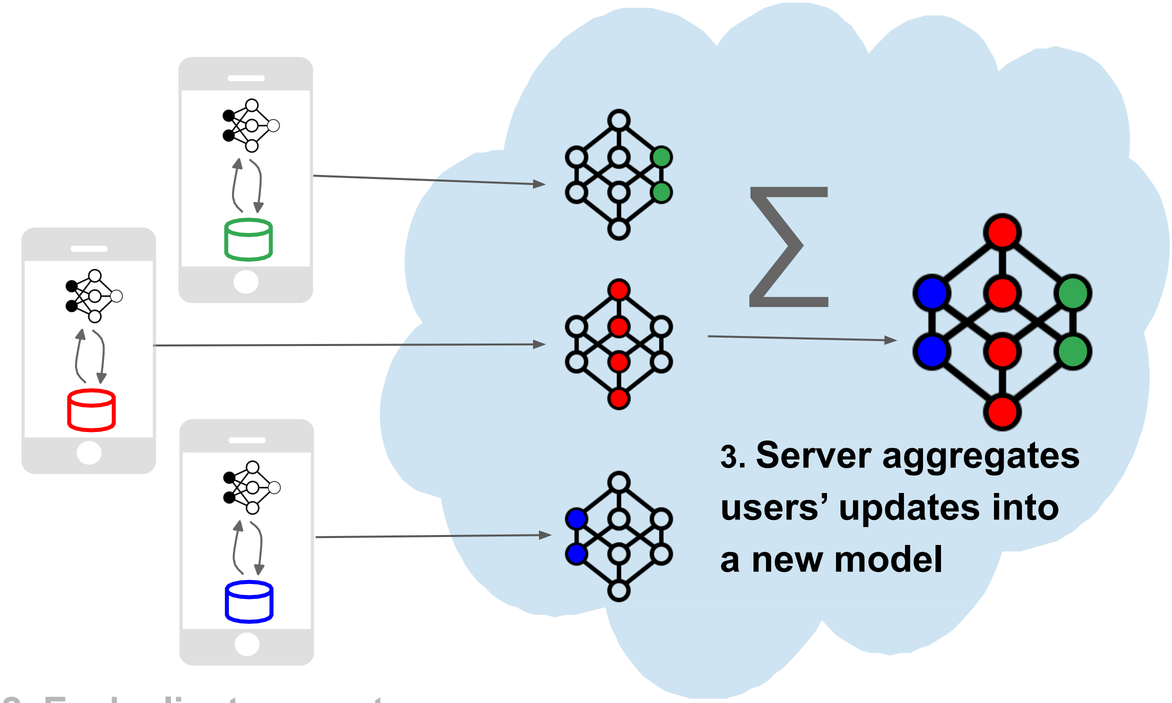
1. Each client downloads model parameters from central server



2. Each client computes updates using their local data and send to server.

Federated Learning

1. Each client downloads model parameters from central server

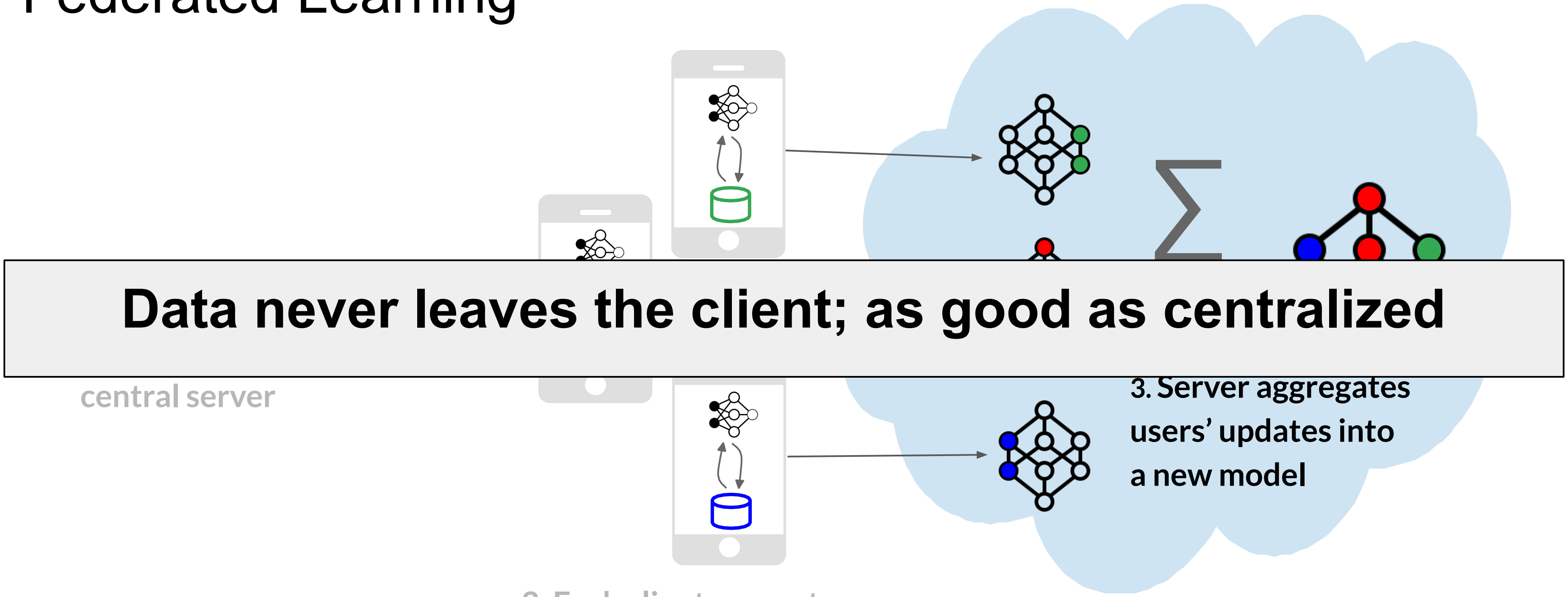


2. Each client computes updates using their local data and send to server.

3. Server aggregates users' updates into a new model

Repeat until convergence.

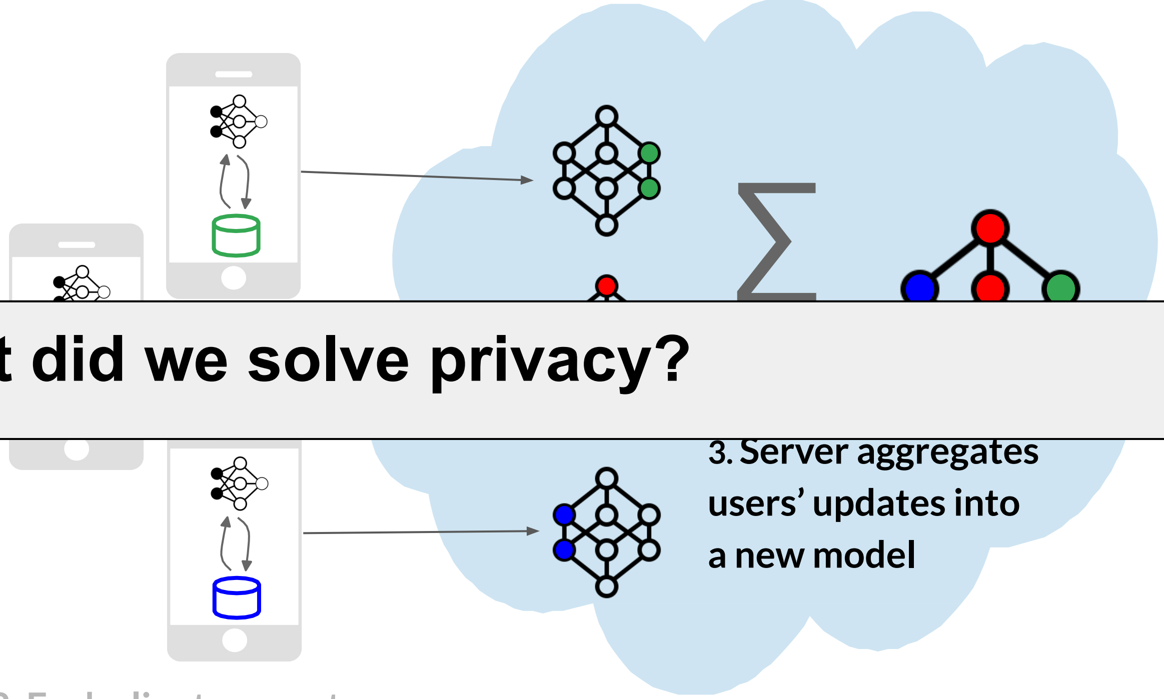
Federated Learning



2. Each client computes updates using their local data and send to server.

Repeat until convergence.

Federated Learning



But did we solve privacy?

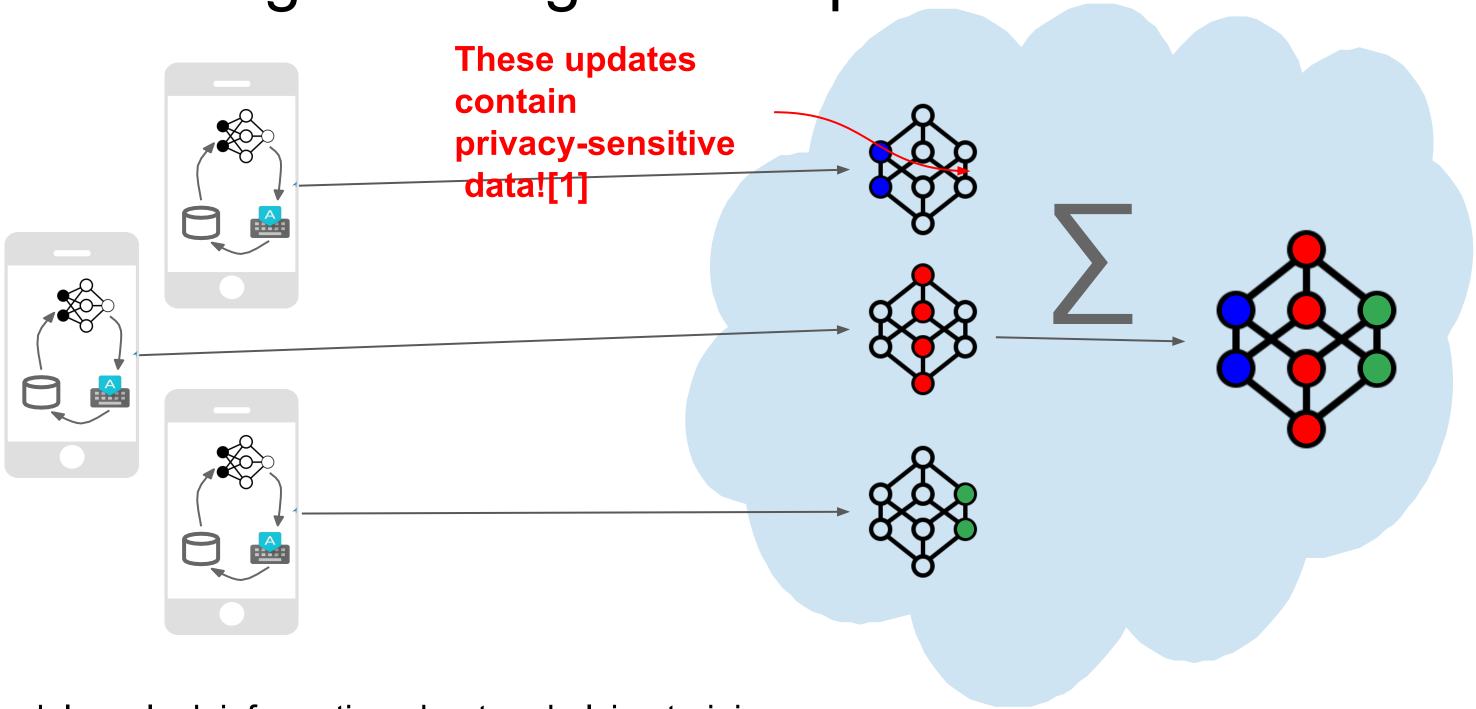
central server

3. Server aggregates users' updates into a new model

2. Each client computes updates using their local data and send to server.

Repeat until convergence.

Federated Learning - Leakage from updates

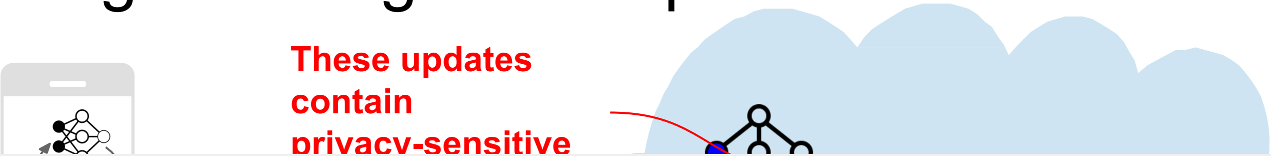


Problem:

- Updates to model can leak information about underlying training data

[1] *Melis et al. “**Exploiting unintended feature leakage in collaborative learning**” IEEE S&P 19*

Federated Learning - Leakage from updates



<u>Leakage from updates:</u>

- Model updates from SGD

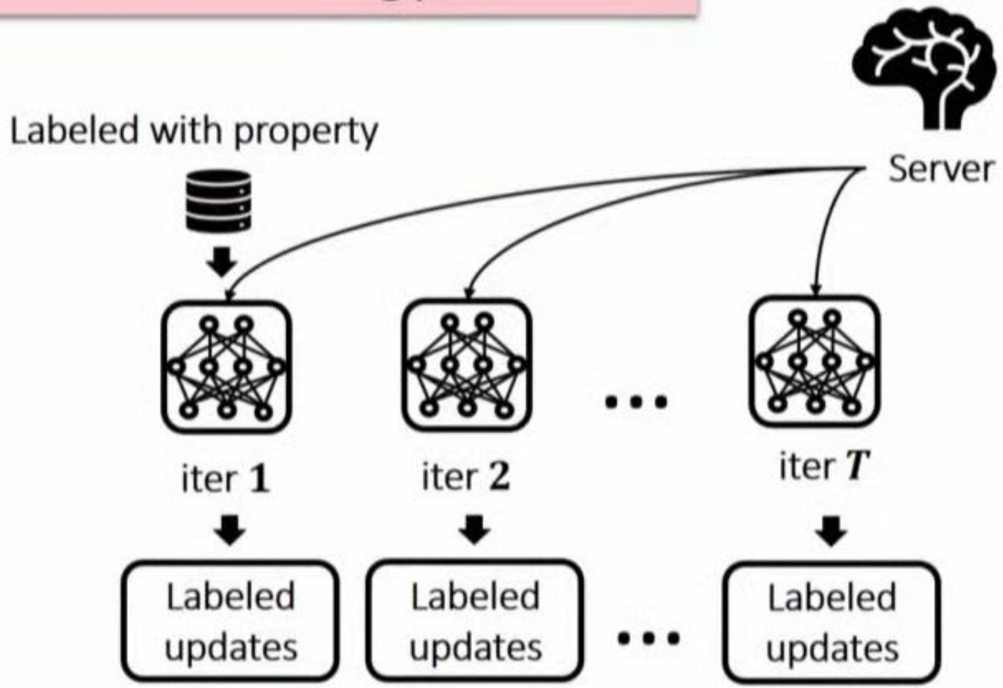
$$y = W \cdot h, \qquad \frac{\partial L}{\partial W} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial W} = \frac{\partial L}{\partial y} \cdot \boldsymbol{h}$$

- $\boldsymbol{h}$ = features of $\boldsymbol{x}$ learned to predict $\boldsymbol{y}$

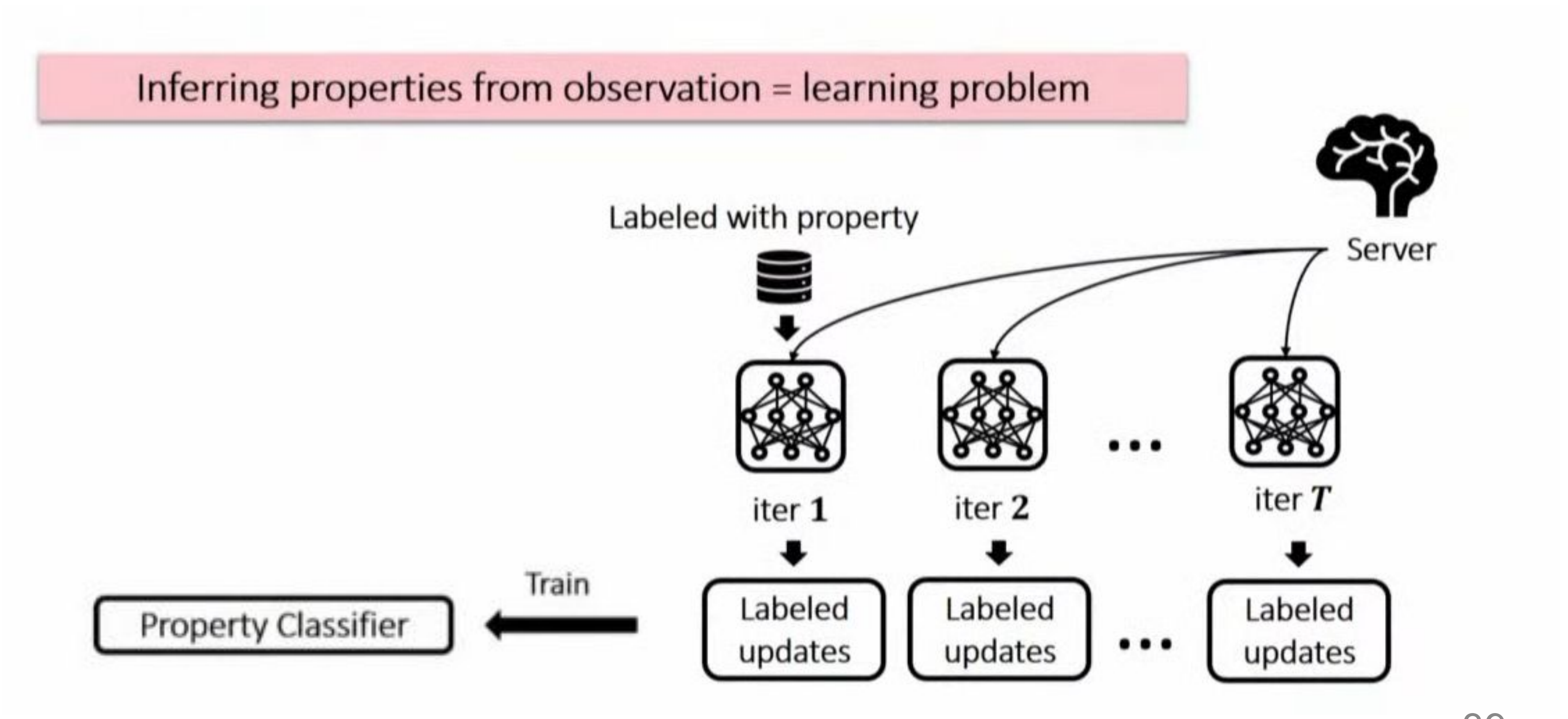
- If adversary has a set of labelled (update, feature) pairs, then it can train a classifier to predict features from updates

Federated Learning - Leakage from updates

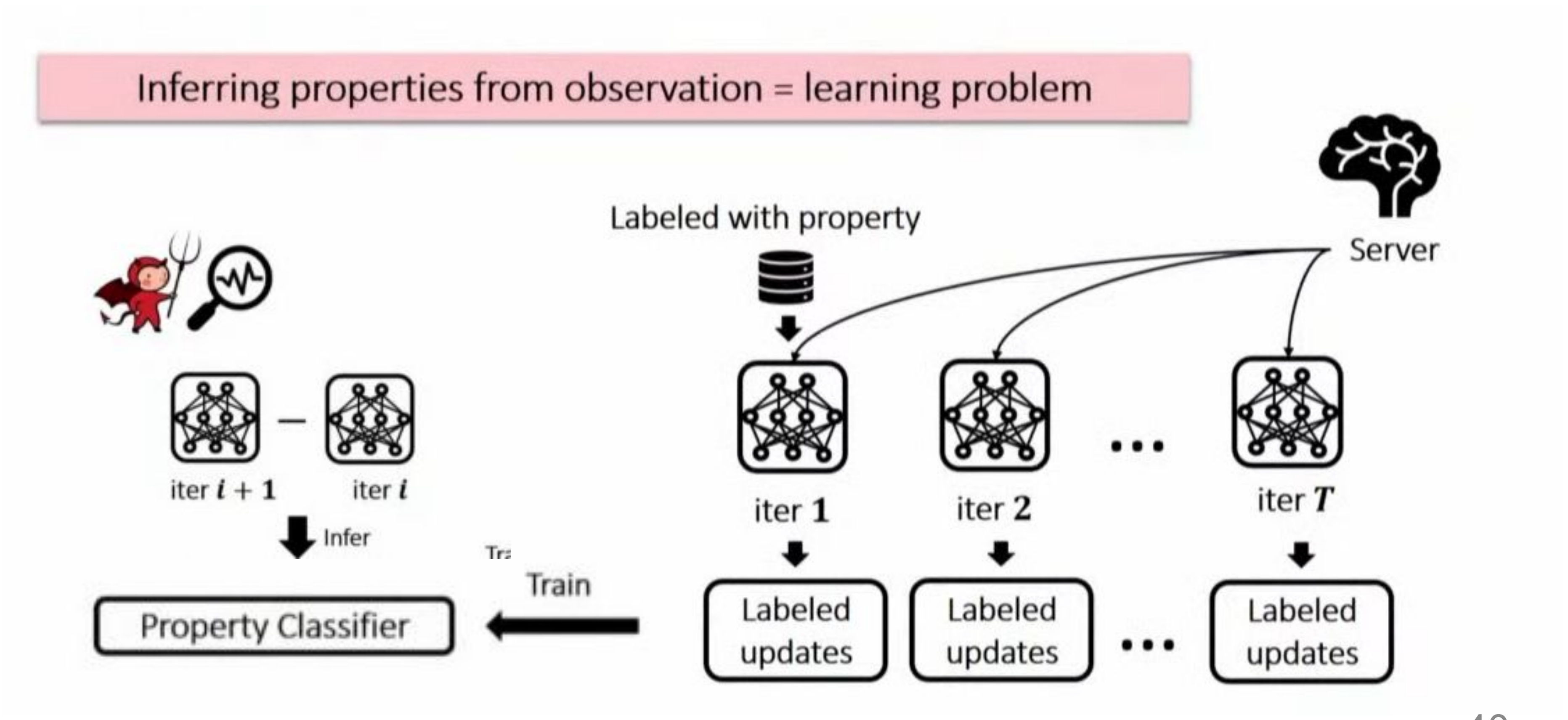
Inferring properties from observation = learning problem



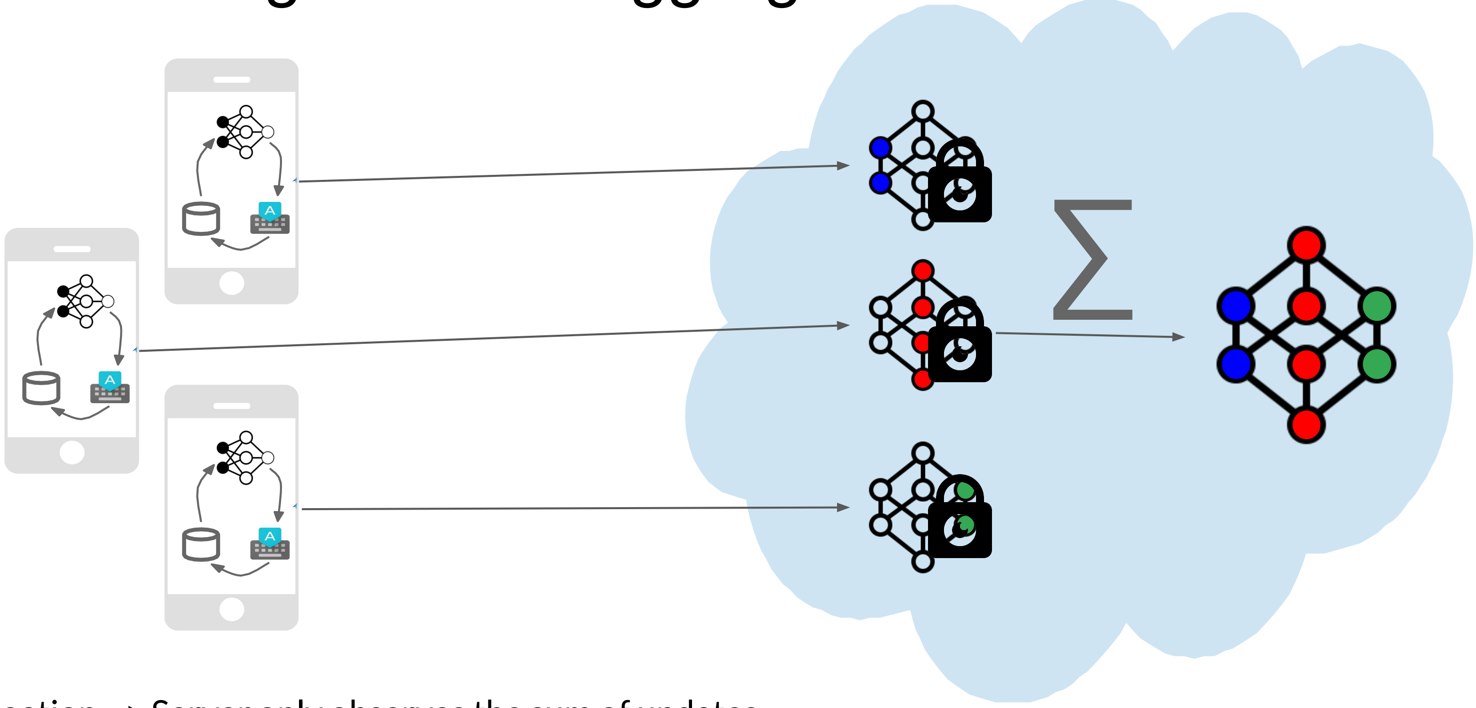
Federated Learning - Leakage from updates



Federated Learning - Leakage from updates



Federated Learning - Secure aggregation

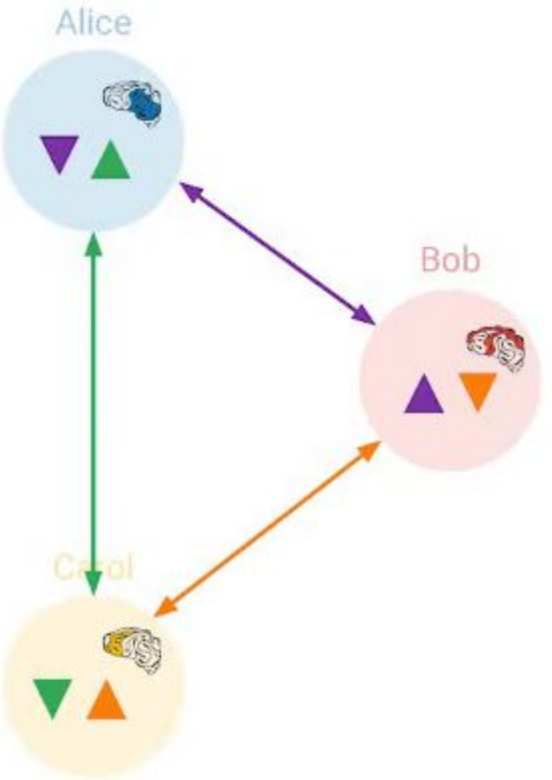


Solution:

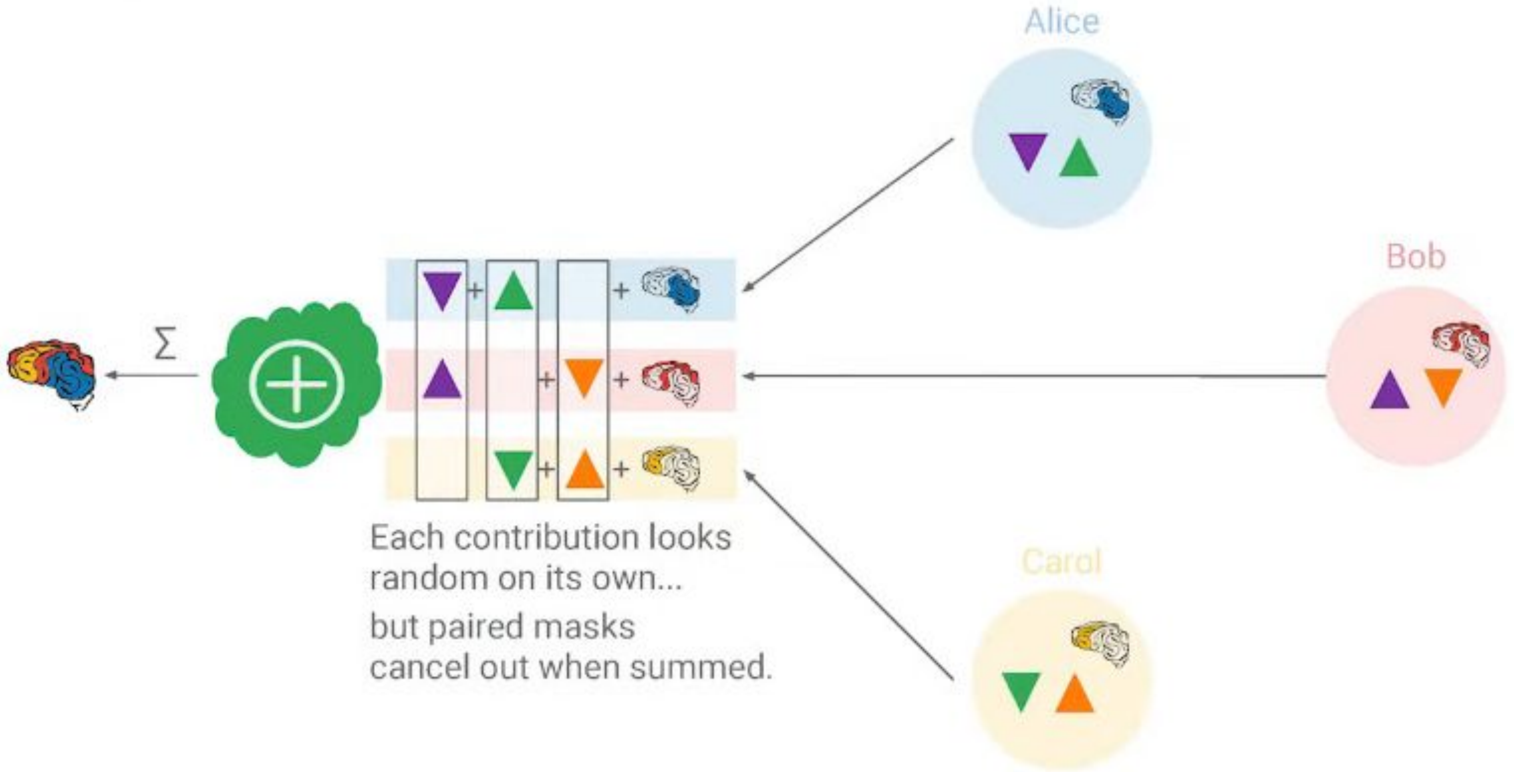
- Secure Aggregation => Server only observes the sum of updates.

Federated Learning - Secure aggregation

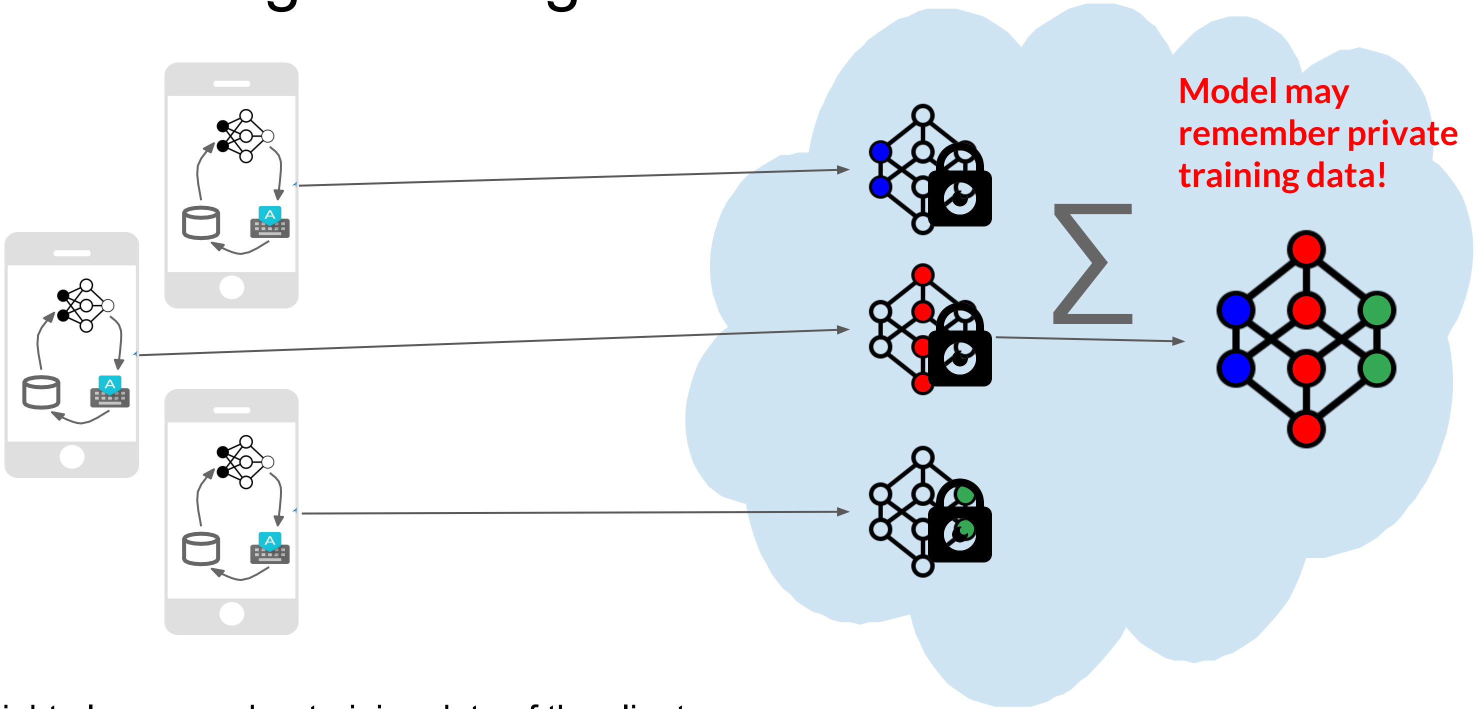
Devices cooperate to sample random pairs of 0-sum masking vectors.



Federated Learning - Secure aggregation



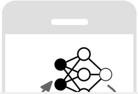
Federated Learning - Leakage from model



Problem:

- The model might also remember training data of the client.

Federated Learning - Leakage from model



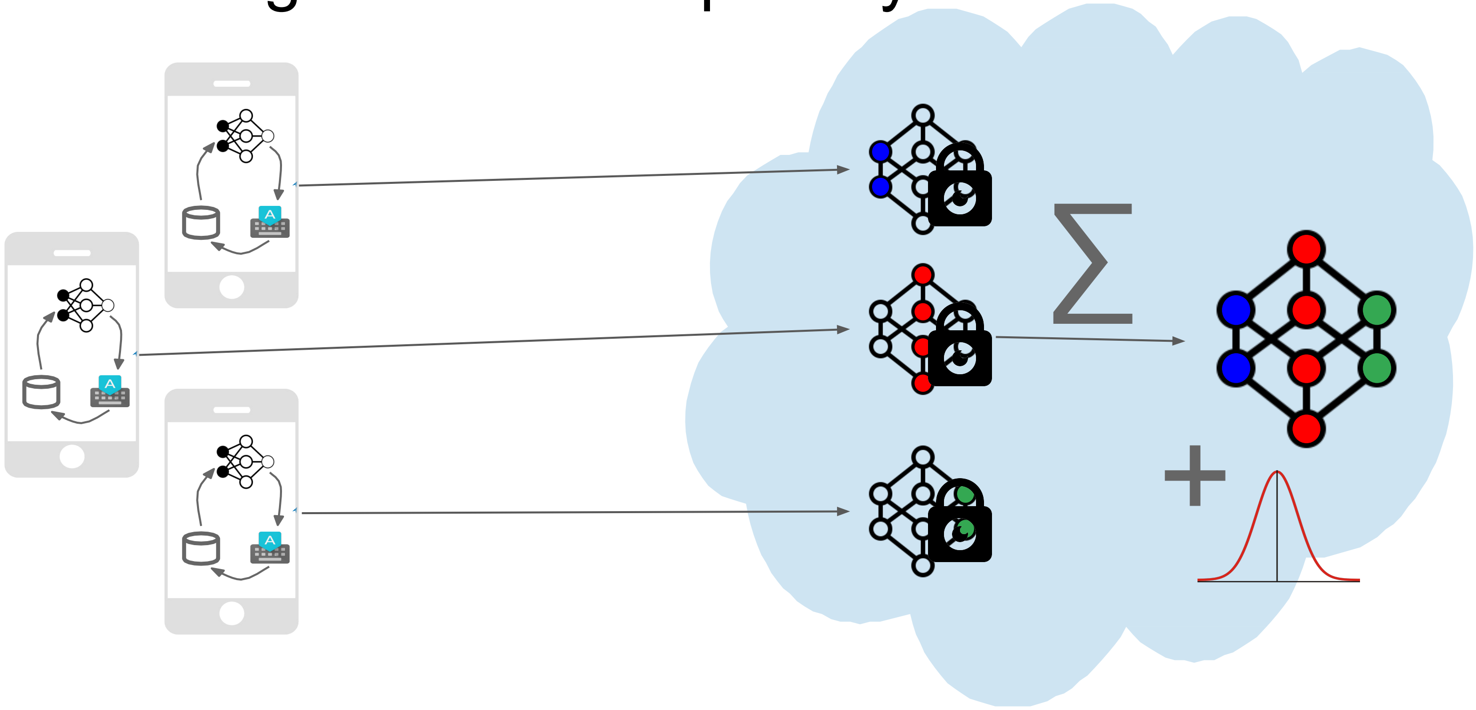
<u>Model inversion attack:</u>

- Solve an optimization problem:

  find the input that maximizes the returned confidence, subject to the classification also matching the target.

Figure 1: An image recovered using a new model inversion attack (left) and a training set image of the victim (right). The attacker is given only the person's name and access to a facial recognition system that returns a class confidence score.

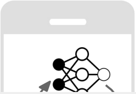
Federated Learning - Differential privacy



Solution:

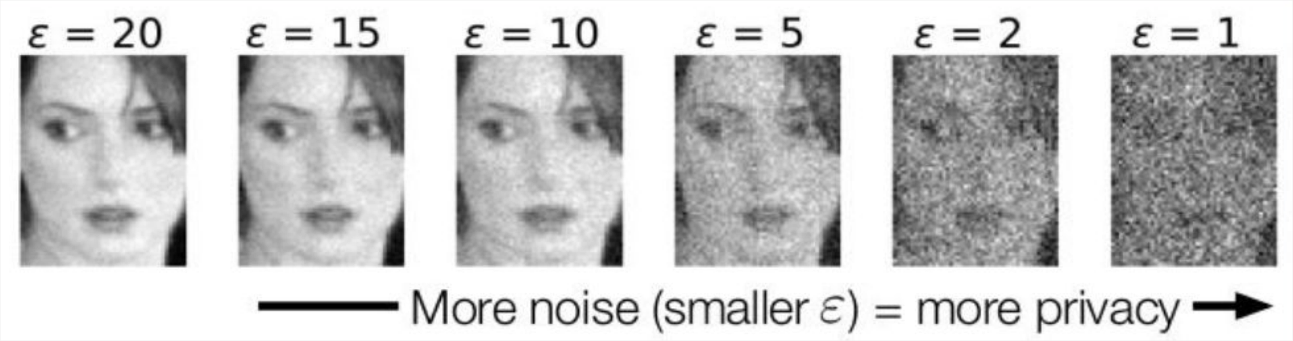
=> Differential Privacy - Add noise to the trained model

Federated Learning - Differential privacy

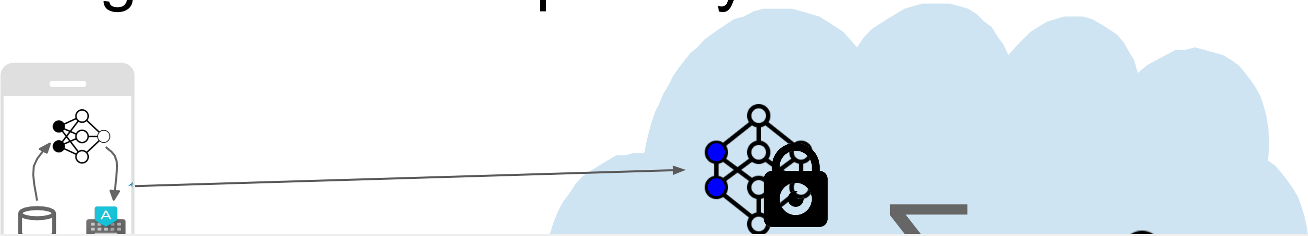


Differential privacy:

- Amount of noise added parametrized by a privacy budget (ϵ)
    - Lower (ϵ) means more noise added so more privacy
    - Lower (ϵ) means more lower utility/performance of model



Federated Learning - Differential privacy



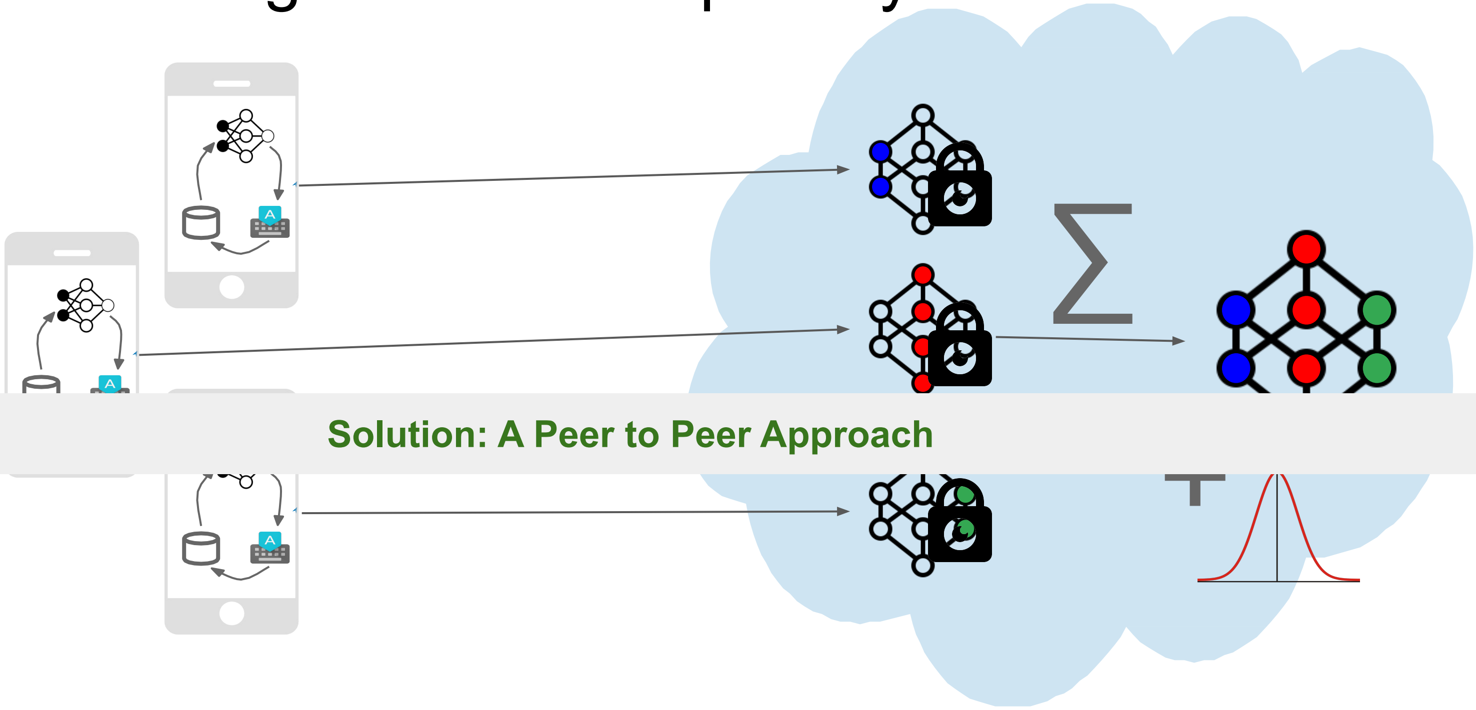
Problems with Federated Learning:

- A centralized coordinator may not always be feasible in all use cases like healthcare, banking
- Clients may be malicious and try to harm the performance of the model.

Solution:

=> Differential Privacy - Add noise to the trained model

Federated Learning - Differential privacy



Solution:
=> Differential Privacy - Add noise to the trained model

Outline

ML training

- Background
- Parameter Server
- Federated Learning
- Peer to Peer approaches

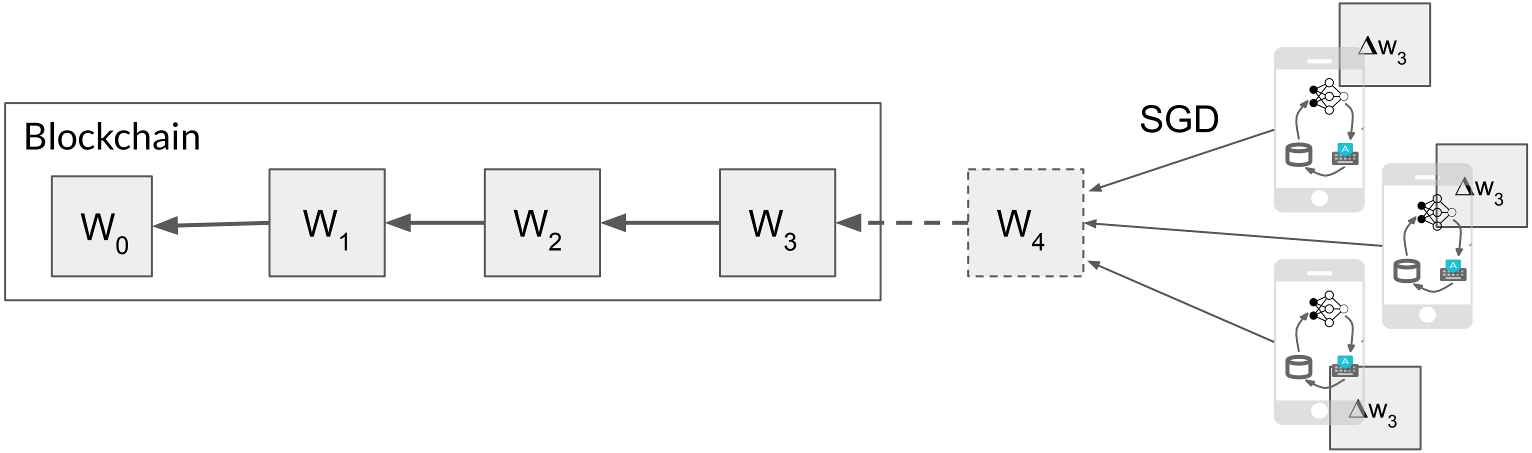
ML Inference

- Containers
- Microservices
- Example ML System: Korbit AI

Maintaining ML Models in production

Peer to Peer ML on the blockchain

- Each block stores a set of updates from multiple peers and the updated model
  - Each peer computes updates using their blockchain state
  - With each block, the set of updates is added, updating the global model



- Much work has been done in this area (OpenMined, OasisLabs, Biscotti) but no widely adopted system yet

Outline

ML training

- Background
- Parameter Server
- Federated Learning
- Peer to Peer approaches

ML Inference

- Containers
- Microservices
- Example ML System: Korbit AI

Maintaining ML Models in production

Putting machine learning models in production

- Once a model has been trained, it’s time that other people start using it.
  - You would want to deploy it, so that other people can start using it.

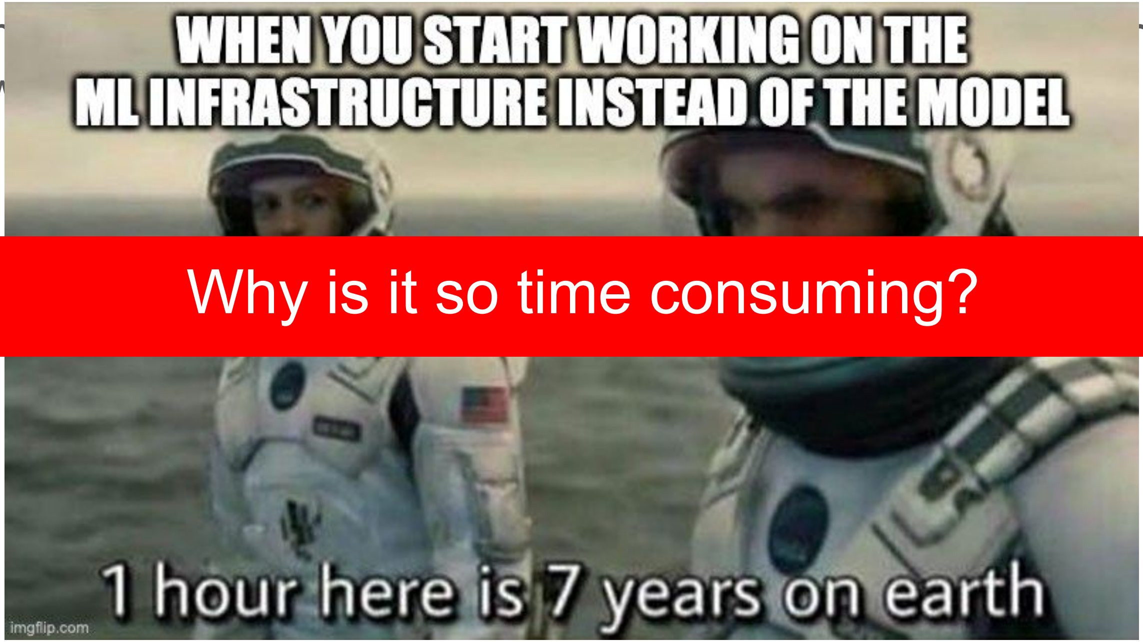
- Deploying machine learning models comes with its own set of challenges!

Putting machine learning models in production

- Once a r... t using it.
    - You w...
- Deployin... challenges!

Putting machine learning models in production

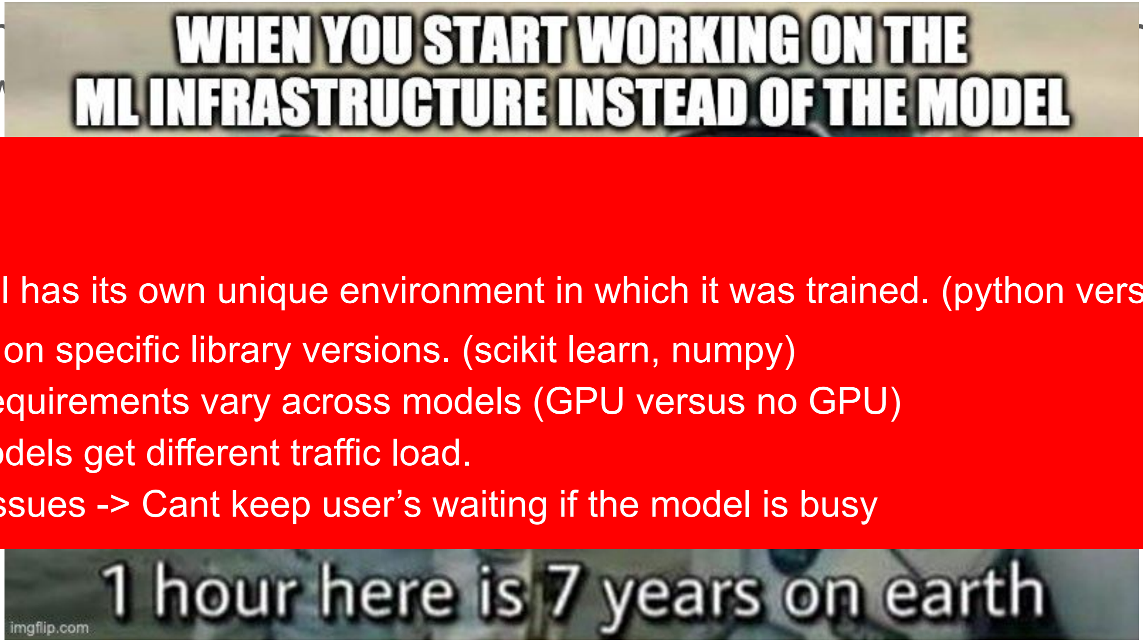
- Once a r ... t using it.
    - You w



Why is it so time consuming?

Putting machine learning models in production

- Once a r … t using it.
    - You w



Problems:

- Every model has its own unique environment in which it was trained. (python versions etc.)
- Models rely on specific library versions. (scikit learn, numpy)
- Resource requirements vary across models (GPU versus no GPU)
- Different models get different traffic load.
- Scalability issues -> Cant keep user’s waiting if the model is busy

Putting machine learning models in production

- Once a r … t using it.
  - You w

Deployin … challenges!

Solution => Docker containers

Sol

Outline

ML training

- Background
- Parameter Server
- Federated Learning
- Peer to Peer approaches

ML Inference

- Containers
- Microservices
- Example ML System: Korbit AI

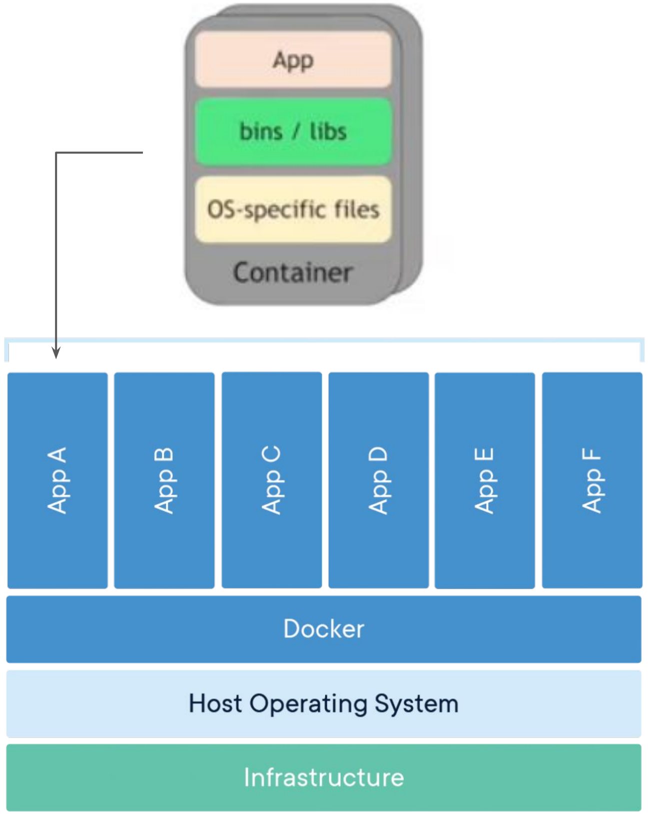
Maintaining ML Models in production

What are docker containers?

- A group of processes that run in isolation on a single machine

- Each container has its own set of:
  - Processes
  - Users
  - Memory

- They share the same base operating system but have their own set of binaries, libraries and os specific files



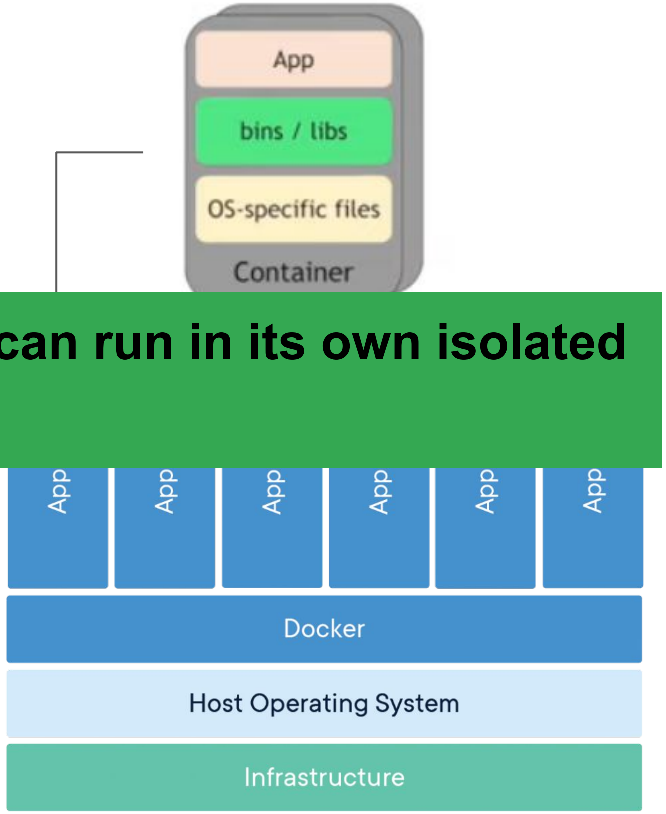
What are docker containers?

- A group of processes that run in isolation on a single machine

Using containers, a machine learning model can run in its own isolated environment

  - Memory

- They share the same base operating system but have their own set of binaries, libraries and os specific files



How to containerize ML models?

- Step 1: Train and save your model
- Step 2: Create an API to send data and make predictions with your model
- Step 3: Create a Dockerfile -> specifying the requirements
- Step 4: Create your container and deploy

How to containerize ML models?

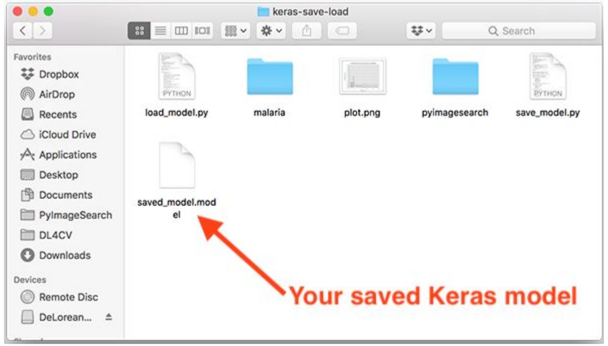
- Step 1: **Train and save your model**
- Step 2: Create an API to send data and make predictions with your model
- Step 3: Create a Dockerfile -> specifying the requirements
- Step 4: Create your container and deploy

How to containerize ML models?

- Step 1: **Train and save your model**
- Step 2: Create an API to send data and make predictions with your model
- Step 3: Create a Dockerfile -> specifying the requirements
- Step 4: Create your container and deploy

- All machine learning libraries have their default save method.

- Or, turn it into an object and save it in a pickle file.



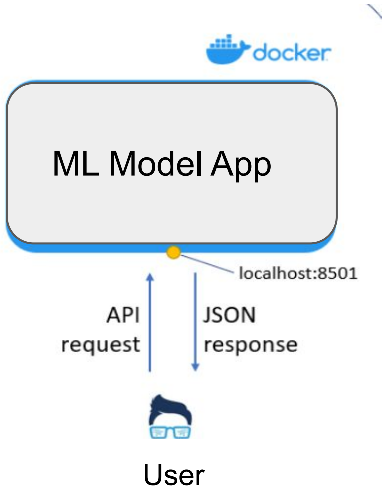
How to containerize ML models?

- Step 1: Train and save your model
- **Step 2: Create a API to receive data and make predictions**
- Step 3: Create a Dockerfile -> specifying the requirements
- Step 4: Create your container and deploy

What is a API?

- A piece of programming code that allows an application to talk to the outside world.

- Allows an application to listens for incoming requests and take action/respond based on the type of request



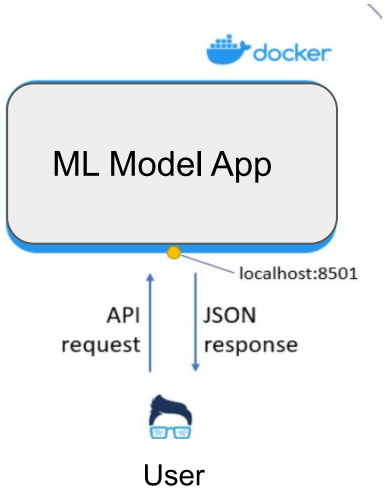
How to contai

- Step 1: Train a
- **Step 2: Create ... ions**
- Step 3: Create ... s
- Step 4: Create

```python
# Import Flask and Machine Learning Dependencies for
# loading your model and preparing the input data
import flask, request
# ...

### MODEL CREATION AND DATA PREPROCESSING ###
# Create a function or functions for preprocessing the input data
# This can be normalizing price data, augmenting images, etc.
def proccess_data(data):
  pass

# Create a function to load your model (based on the load/save your model section)
def load_model(): # can chage "model" to your specific model type or name
  pass

### SETTING UP FLASK APP AND FLAKS ENDPOINTS ###
# Create the flaks App
app = flask.Flask(__name__)

# Define an endpoint for calling the predict function based on your ml library/framework
@app.route("/predict", methods=["GET","POST"])
def predict():
    # Load the Input
    data = request.files['file'] # Image input
    data = requiest.form['form_input_id'] # String input

    # Load the model
    model = load_model()

    # Make predictions on input data
    model.predict(data) # .predict() could change based on libarary/framework


# Start the flask app and allow remote connections
app.run(host='0.0.0.0', port = 80)
```

What is a API?

- A piece of progran outside world.
- Allows an applicat action/respond ba



How to containerize ML models?

- Step 1: Train and save your model
- Step 2: Create a API to receive data and make predictions
- **Step 3: Create a Dockerfile -> specifying the requirements**
- Step 4: Create your container and deploy

What is a Dockerfile?

- Dockerfile is responsible for creating the image that’s used to create the container that hosts the model and API

- Specify all library/model requirements and launch app

How to containerize ML models?

- Step 1
- Step 2
- **Step 3**
- Step 4

What is a Docl

- Dockerfile used to c API
- Specify a

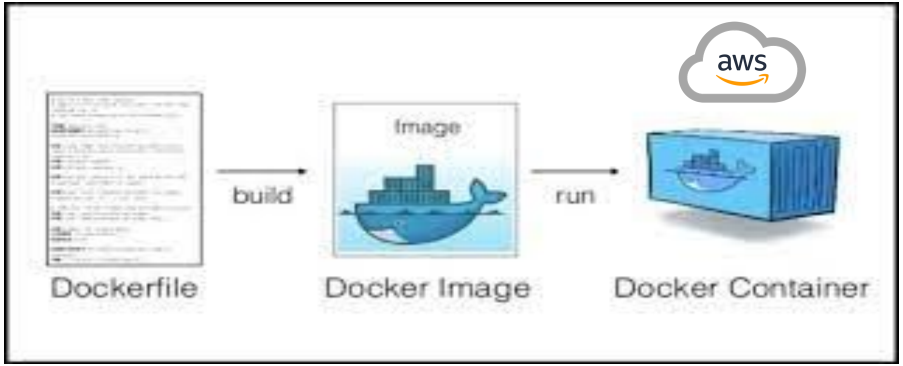
```
# STEP 1: Install base image. Optimized for Python.
FROM python:3.7-slim-buster

# Step 2: Add requirements.txt file
COPY requirements.txt /requirements.txt

# Step 3:  Install required pyhton dependencies from requirements file
RUN pip install -r requirements.txt

# Step 4: Copy source code in the current directory to the container
ADD . /app

# Step 5: Set working directory to previously added app directory
WORKDIR /app

# Step 6: Expose the port Flask is running on
EXPOSE 8000

# Step 9: Run Flask
CMD ["flask", "run", "--host=0.0.0.0", "--port=8000"]
```

How to containerize ML models?

- **Step 1: Train and save your model**
- **Step 2: Create a API to receive data and make predictions**
- Step 3: Create a Dockerfile -> specifying the requirements
- **Step 4: Create your container and deploy**

- Use docker to build an image from the dockerfile
  - A image is a snapshot of the environment that can't change

- Run a container from that image and deploy in the cloud



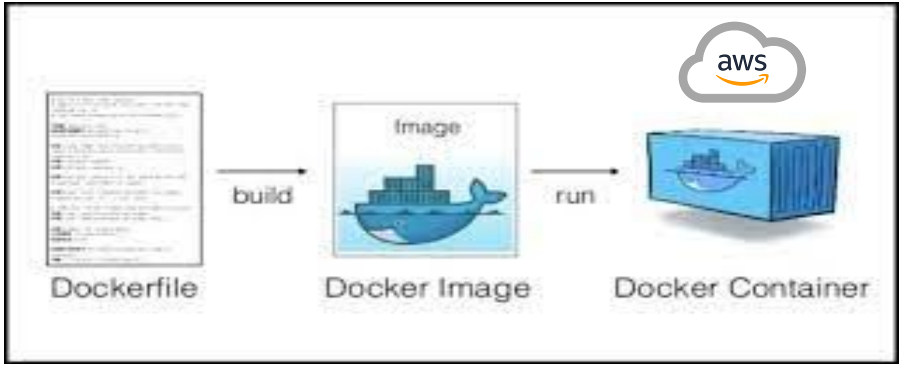
How to containerize ML models?

- **Step 1: Train and save your model**
- **Step 2: Create a API to receive data and make predictions**
- Step 3: Create a Dockerfile -> specifying the requirements
- **Step 4: Create your container and deploy**

Where do these containers fit into when running a complete application?

- Use docker to build an image from the dockerfile
  - A image is a snapshot of the environment that can't change

- Run a container from that image and deploy in the cloud



Outline

ML training

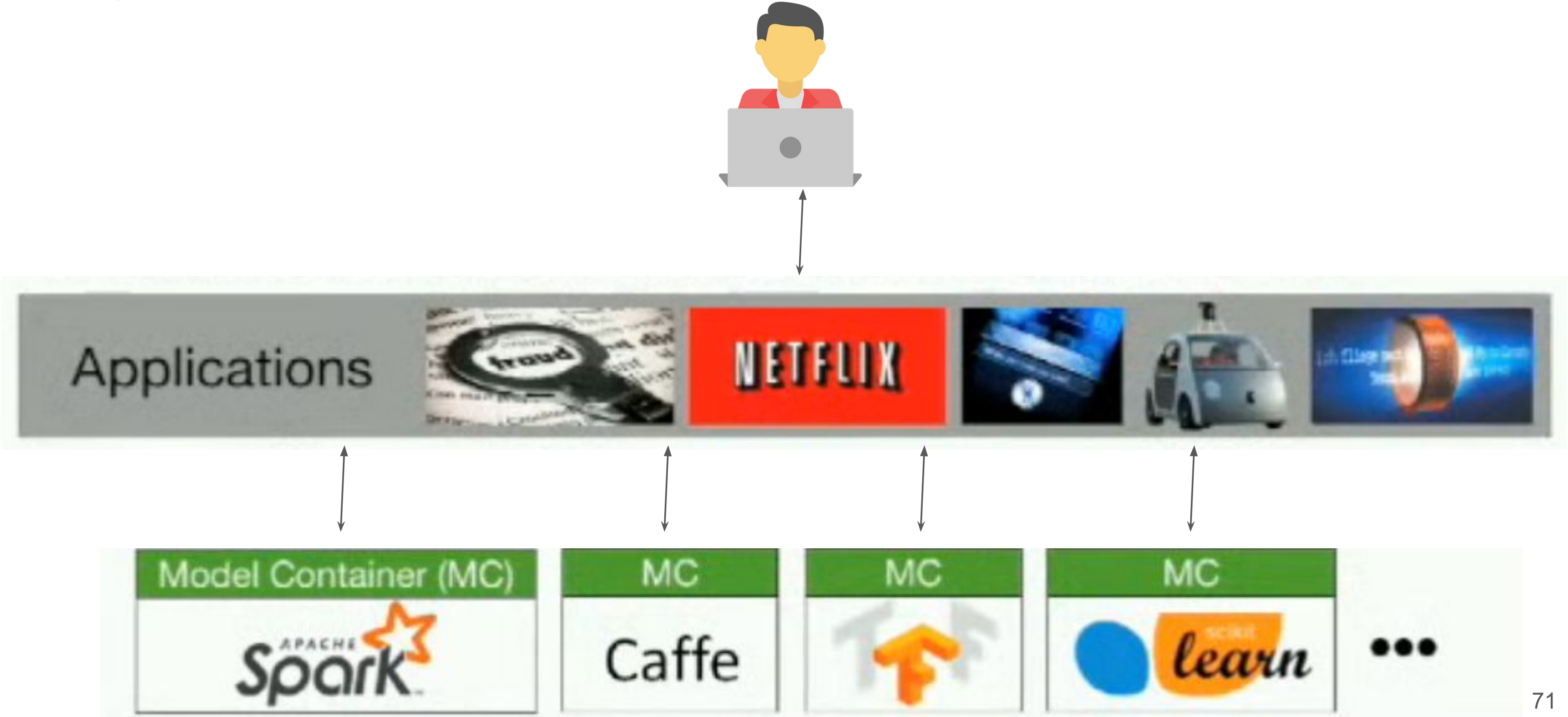
- Background
- Parameter Server
- Federated Learning
- Peer to Peer approaches

ML Inference

- Containers
- Microservices
- Example ML System: Korbit AI

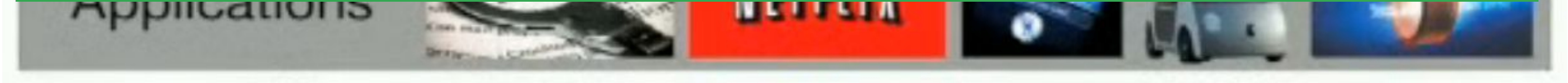
Maintaining ML Models in production

A typical ML application



A typical ML application

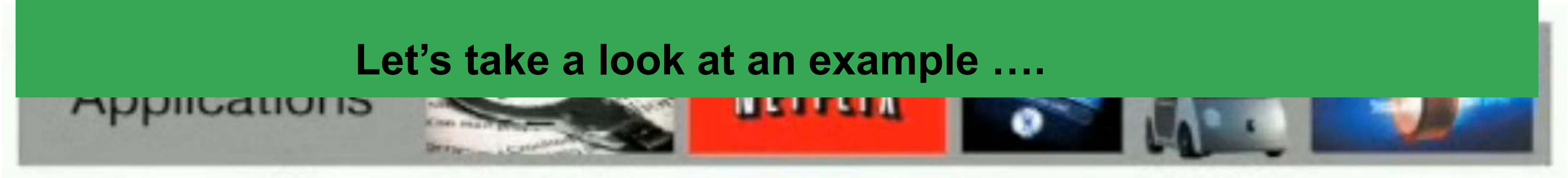
Microservice architecture => Application is a suite of small lightweight independent services



A typical ML application

Microservice architecture => Application is a suite of small lightweight independent services

Let’s take a look at an example ….



Outline

ML training

- Background
- Parameter Server
- Federated Learning
- Peer to Peer approaches

ML Inference

- Containers
- Microservices
- Example ML System: Korbit AI

Maintaining ML Models in production

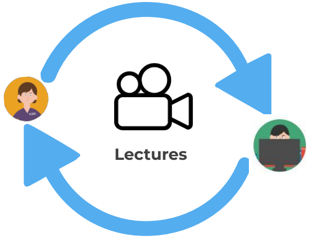
Korbit AI - Demo

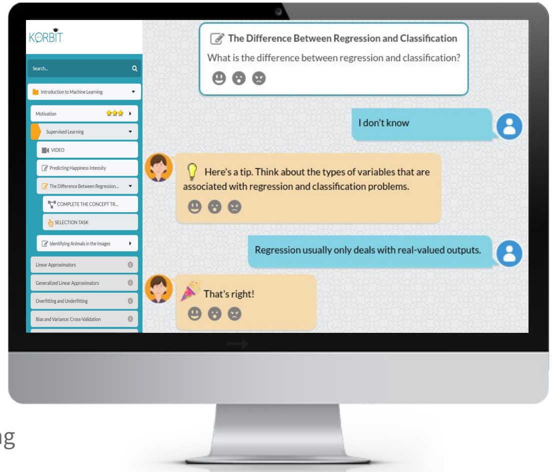
2 MINUTES

KORBIT AI - Learning with an AI tutor

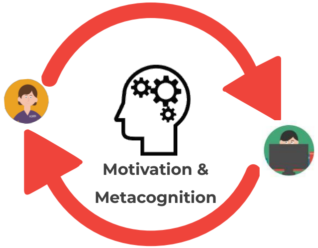
Students learn data science with Korbi through an intuitive, real-time chat interface

Students discuss lectures and text material with Korbi, helping Korbi to diagnose and repair knowledge gaps

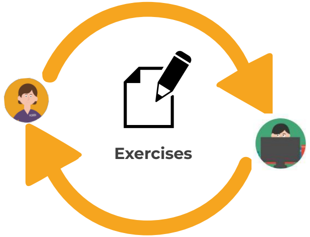




Korbi builds rapport with students, in order to help improve their motivation and metacognitive skills

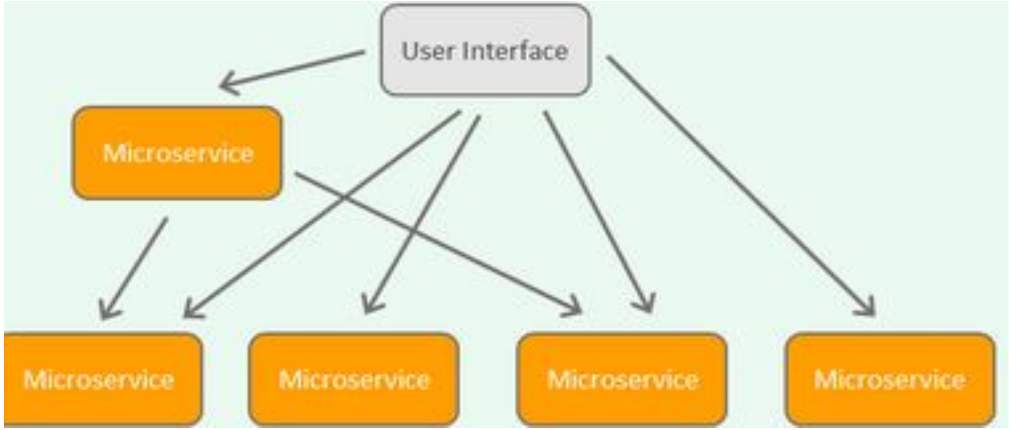


Students do problem-solving exercises with Korbi while receiving instant help and feedback



Students do lab and coding exercises with Korbi, while Korbi analyzes their solutions and provides guidance and feedback

Behind the scenes - A microservice architecture



- All ML models run as their own microservice running in their own isolated docker containers
- Other functionalities like email, database etc also run as stand alone microservices
- Easier to isolate issues and replay bugs

Outline

ML training

- Background
- Parameter Server
- Federated Learning
- Peer to Peer approaches

ML Inference

- Containers
- Microservices
- Example ML System: Korbit AI

Maintaining ML Models in production

Post Production ML Challenges

- When a ML model is put in production, engineers have gone through:
    - Data gathering, Feature Engineering, Model Selection and Error Estimation

- **Production goal: Ensure that the model continues to perform as expected.**

- **Problem: Models faced with a changing environment over time and need to react to incoming data**

Post Production ML Challenges

- When a ML model is put in production, engineers have gone through the following steps:
    - Data gathering, Feature Engineering, Model Selection and Error Estimation

Solution: Ensure that models are retrained periodically with incoming data

expected.

- **Problem: Models faced with a changing environment over time and need to react to incoming data**

When to retrain models in production? - Time based

- A trivial way to do this is to do a ***time-based retraining***

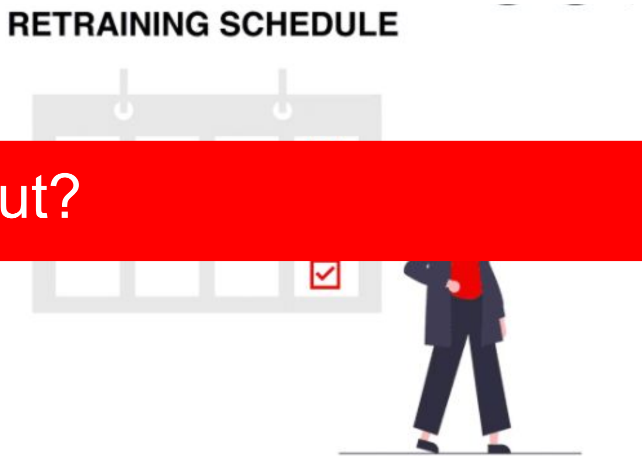
- Retrain at a regular interval, regardless of how it's performing. Needs good understanding of data and how quickly model needs to react

- Can be decided based on how quickly you can collect the data required or how quickly the model needs to react to incoming data.

When to retrain models in production? - Time based

- A trivial way to do this is to do a ***time-based retraining***

- Retrain at a regular interval, regardless of how it's



What if it's not trivial to figure out?

- Can be decided based on how quickly you can collect the data required or how quickly the model needs to react to incoming data.

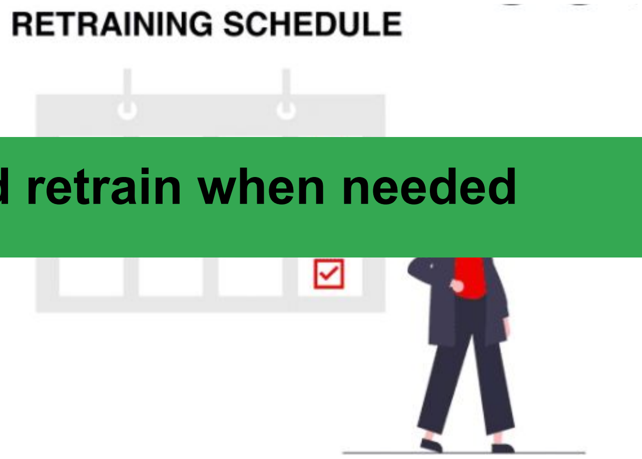
When to retrain models in production? - Time based

- A trivial way to do this is to do a ***time-based retraining***

- Retrain at a regular interval, regardless of how it's

Solution: Periodically measure *model drift* and retrain when needed

- Can be decided based on how quickly you can collect the data required or how quickly the model needs to react to incoming data.



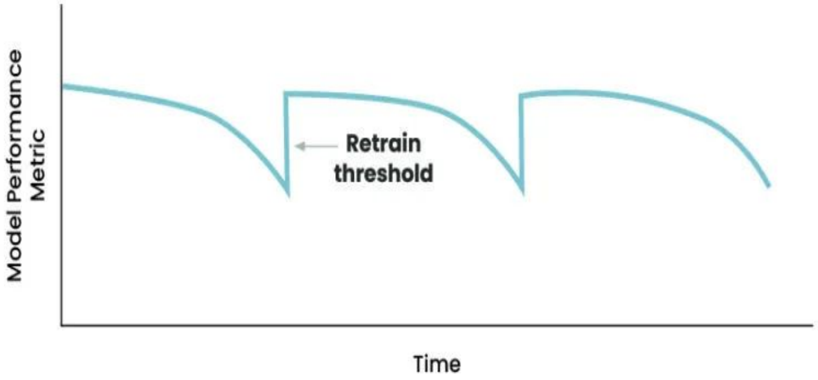
When to retrain models in production? - Continuous

- Model drift is the degradation of a machine learning model over time.

- Multiple ways to measure this decay:
    - Accuracy
    - Change in feature/data distributions
    - Correlation of features
    - Change in data distributions

- Use the metric or combination of metrics that fits your model well.

- If the metric, falls below a threshold then retrain. This is is called *<u>continuous retraining</u>*



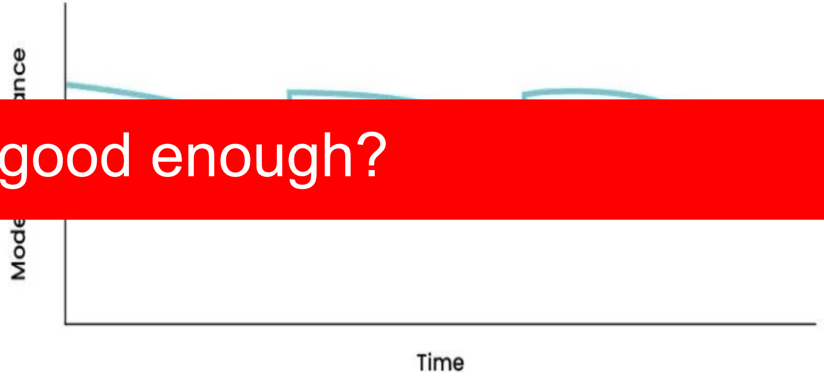
When to retrain models in production? - Continuous

- Model drift is the degradation of a machine learning model over time.

- Multiple ways to measure this decay:

Is the retrained model good enough?

    - Change in data distributions



- Use the metric or combination of metrics that fits your model well.

- If the metric, falls below a threshold then retrain. This is is called <u>*continuous retraining*</u>

After retraining ....

- Once the model is retrained, it's used alongside the old model to see if its performing well.

- Usually done with an A/B test where the performance of the new model is determined and compared with the old one.

- If model improves performance, the cycle continues else we need a new model :(

Conclusion

- Distributed systems play a major role when building a machine learning system.

- When training ML models, we use the following architectures:
    - Parameter server -> When data/model is too large to fit into memory
    - Federated Learning -> When we can't centralize data but trust a centralized controller.
    - P2P approaches -> When users collaborate together to build a ML model

- Once trained, ML models are usually deployed in a microservice based architecture.

- ML models are continuously monitored, periodically retrained until they give good performance