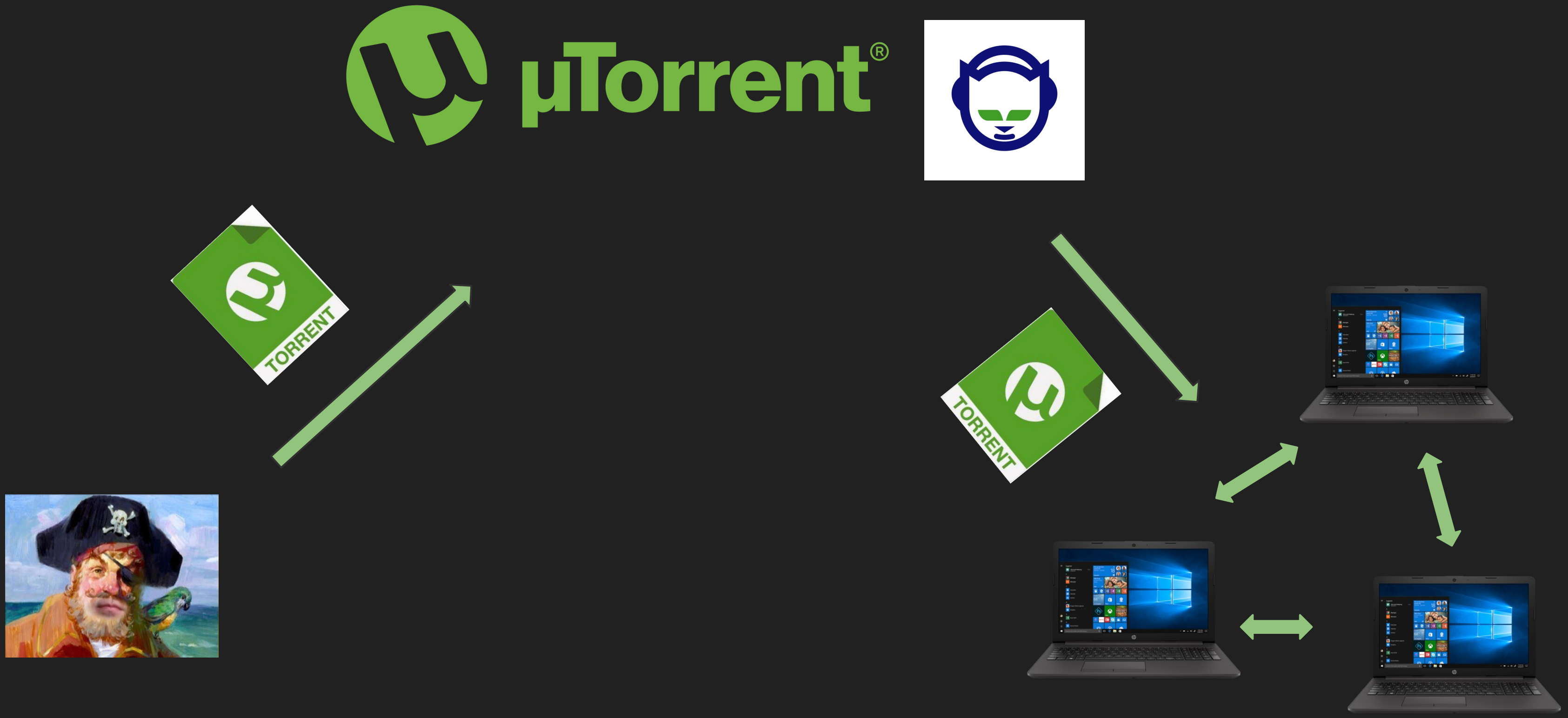
InterPlanetary Learning System

- Pappas Christodoulos
- Chatzopoulos Dimitris
- Lalis Spyros
- Vavalis Manolis

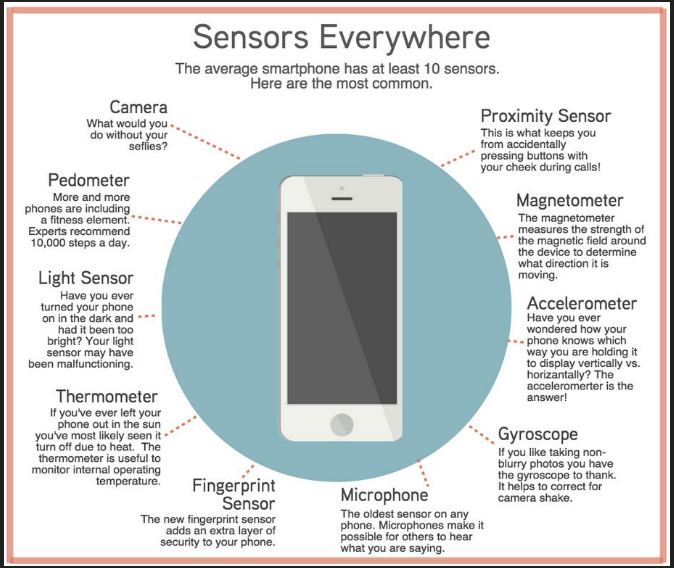
arxiv

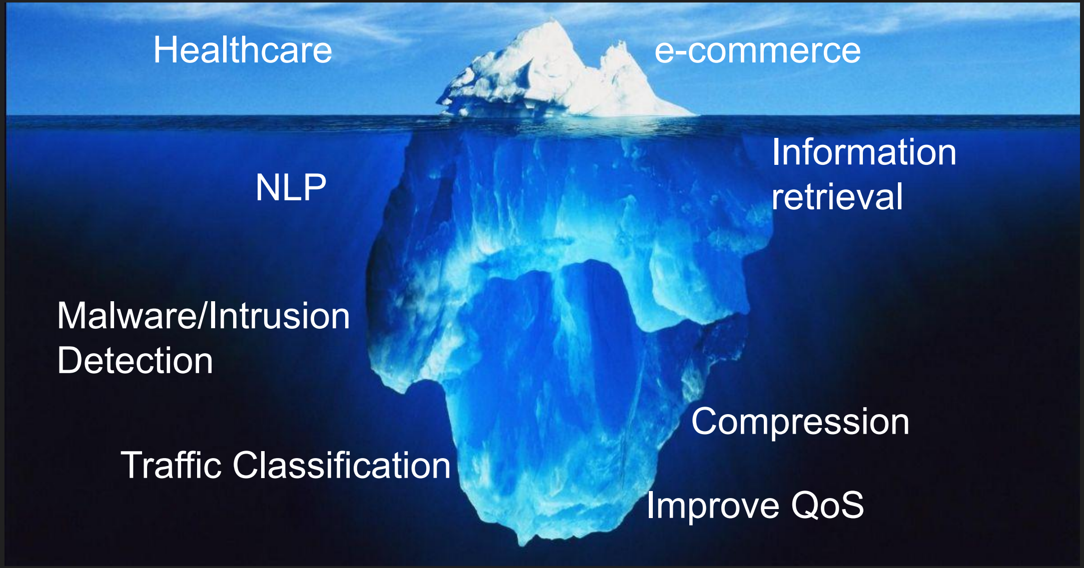
issue

IPLS is for ML models what µTorrent is for files



ML and data availability in our days



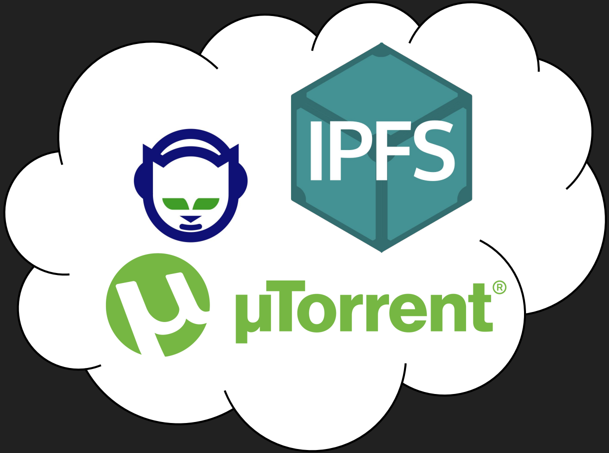


BUT… all these models are trained in a centralized fashion on servers that have access to privacy-sensitive data

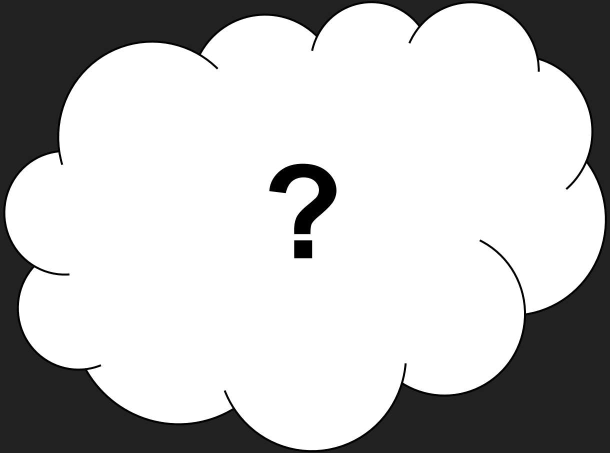
What we want IPLS to be

- Flexible, robust and lightweight middleware that can be deployed on multiple device types
- Energy, network and space efficient
- Easy to participate
- IPLS is already privacy aware (due to FL). Make it privacy preserving!

File Sharing



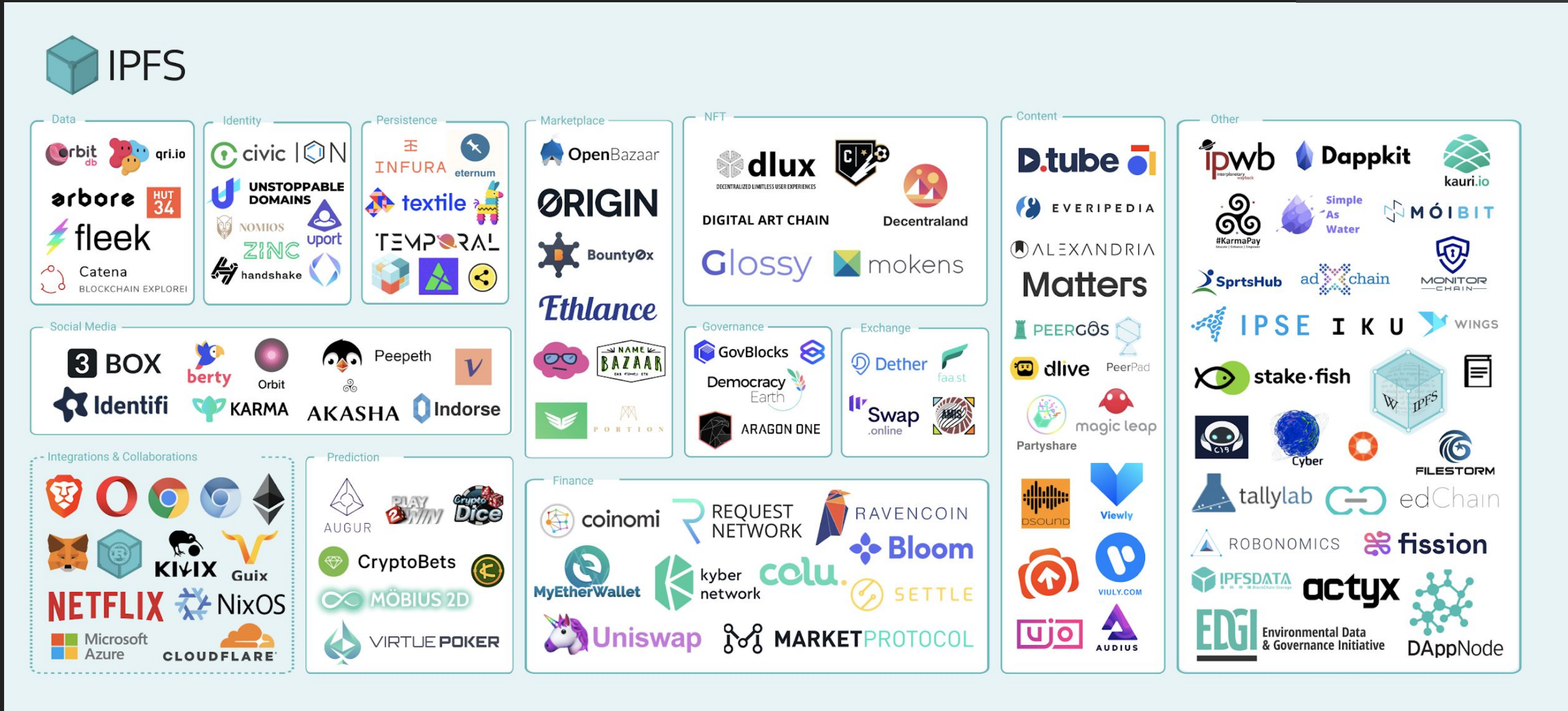
Machine Learning



Why IPFS?

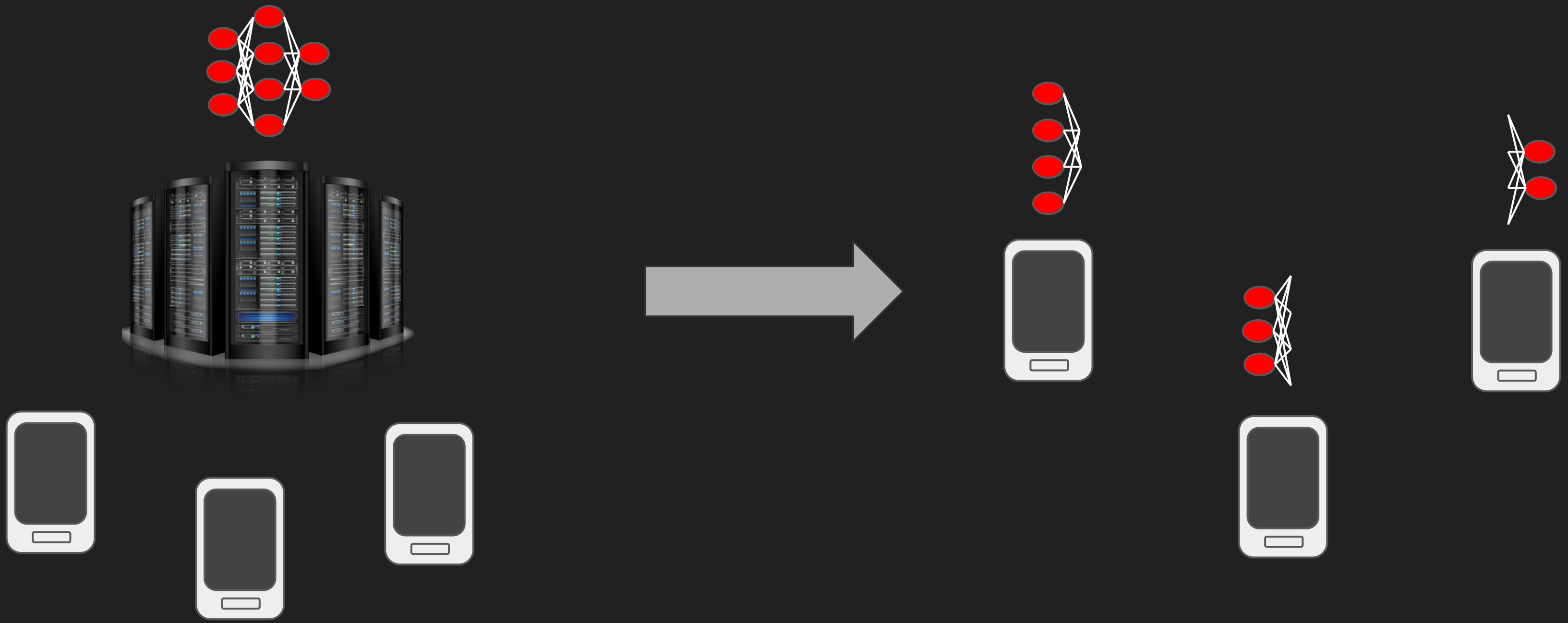
1. The most popular and widely used p2p file sharing system
2. Constantly evolving with a huge active community
3. The new web deserves privacy preserving ML services

libp2p



How IPLS works?

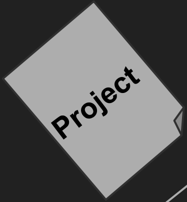
In contrast to centralized FL, where the server is responsible for updating the entirety of the model, in IPLS each peer is responsible for updating a small part of the model.



How IPLS works?

IPFS network

Start-up phase :
Upload project

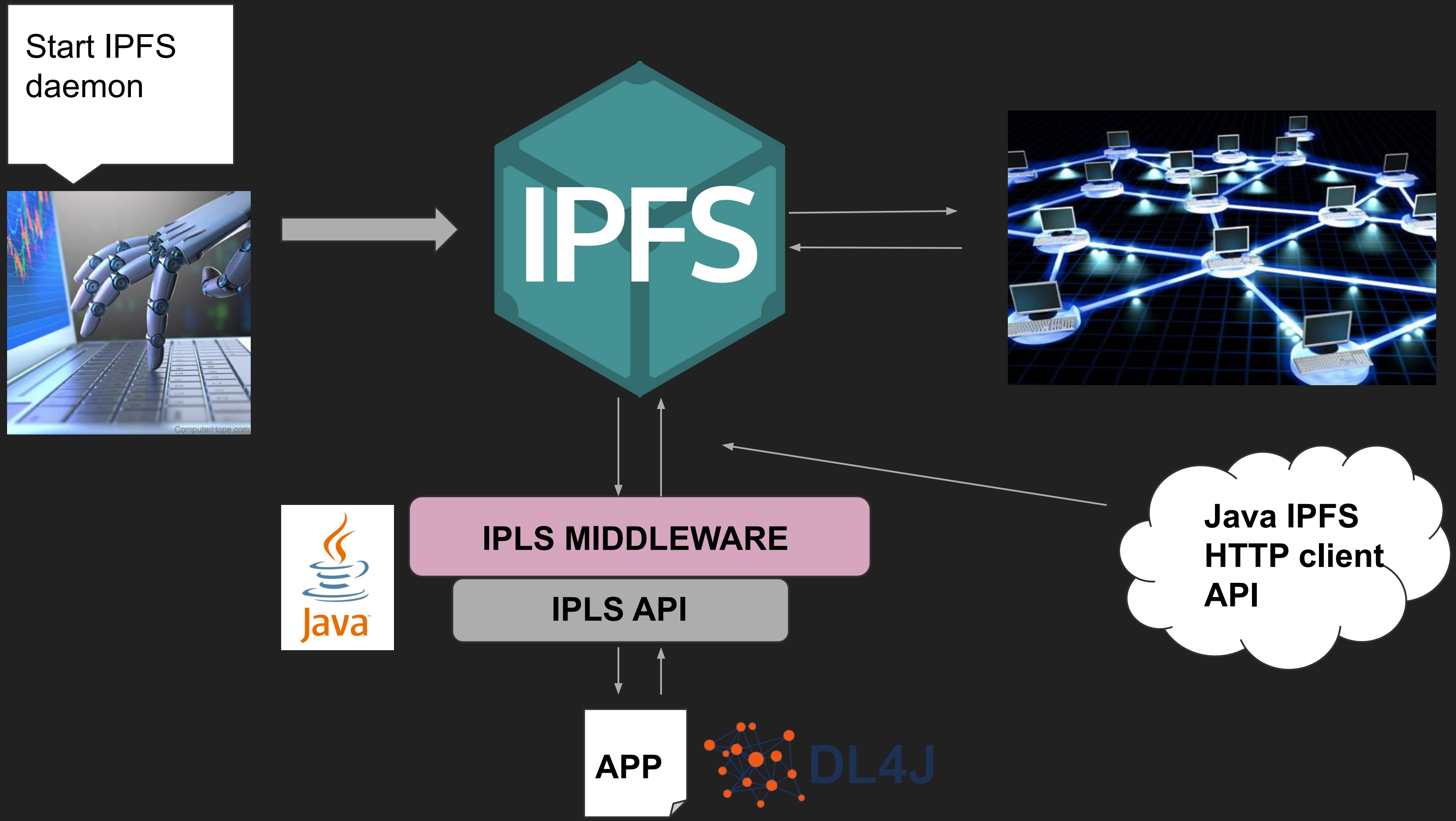


Training phase : Peers collaborate with each other to produce a global model

!

Initialization phase : Interested peers join the project and distribute responsibilities

How to use IPLS?



Concerns and Pitfalls about current Implementation

- Do we take full advantage of the tools given by IPFS?
- Does our implementation overcomes mobility issues?
- Is it network efficient?
- What happens to projects with low participation?
- How will IPLS behave in real world devices?

IPLS PROTOCOL

Important modules

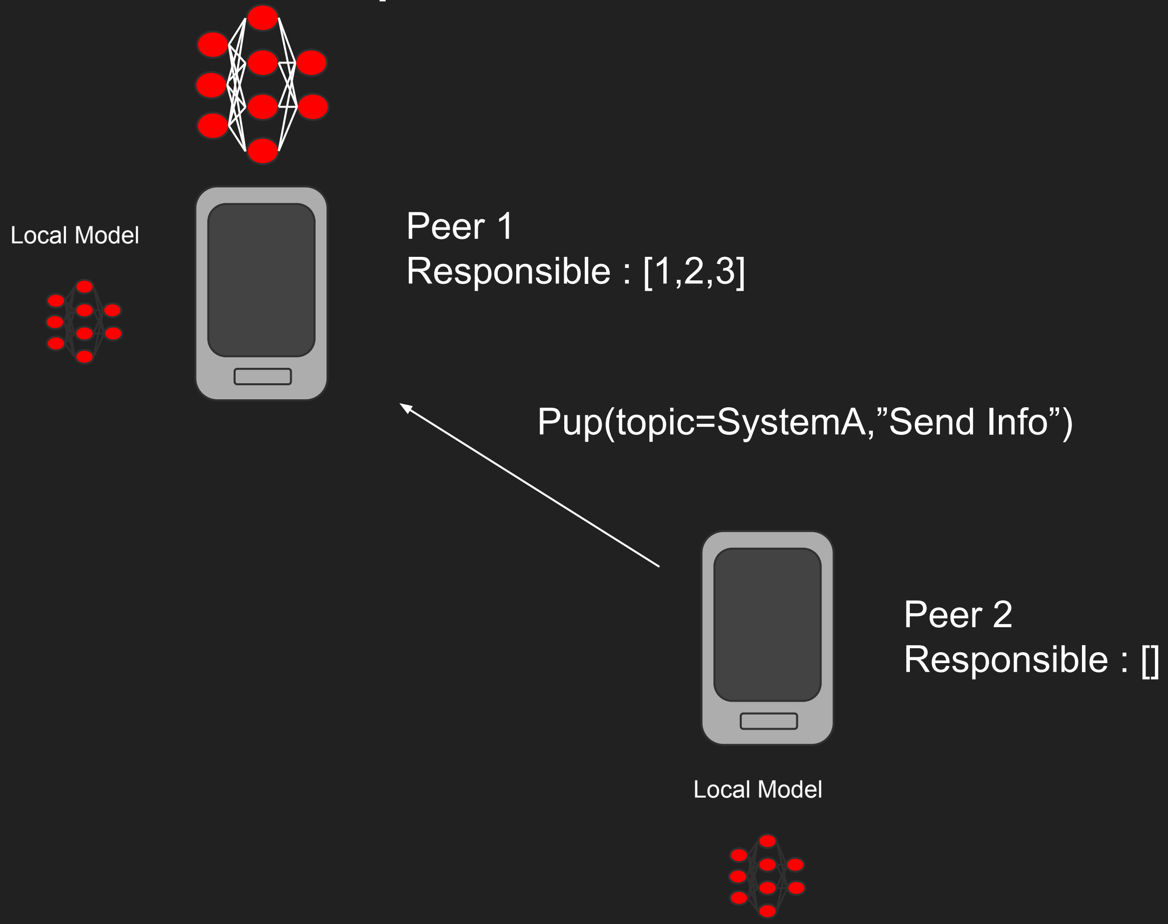
- **IPLS class** : This class contains the API methods of the IPLS described in the paper.
- **Receiver** : Class which processes the incoming messages in the IPLS system (Thread).
- **Swarm Manager** : In this module, the middleware checks for a peer crash and also in the later versions, it will check for indirect messages (Thread).
- **Updater** : In this module, various computational structs are been performed. (Thread)
- **MyIPFSClass** : This module contains wrappers for the IPFS API and also contains methods for serialization and deserialization.

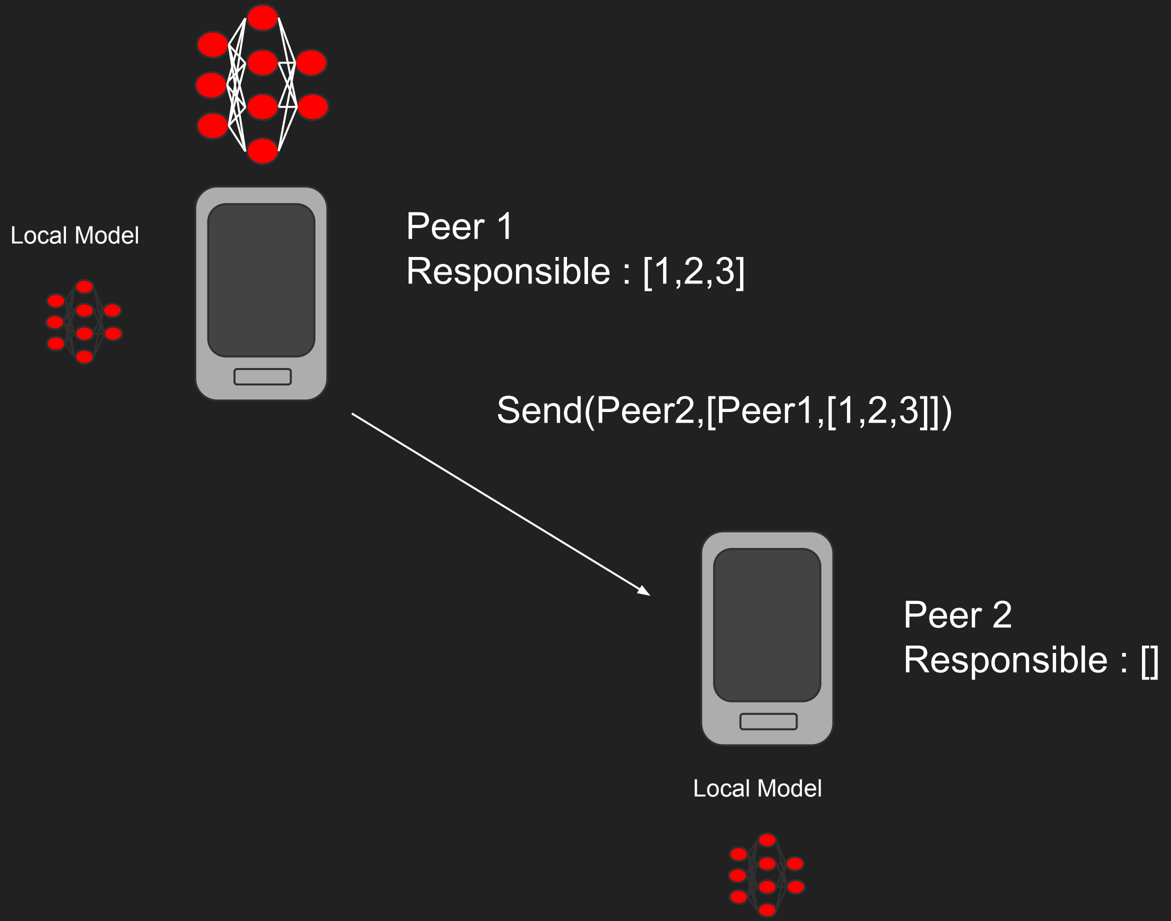
How IPLS works?

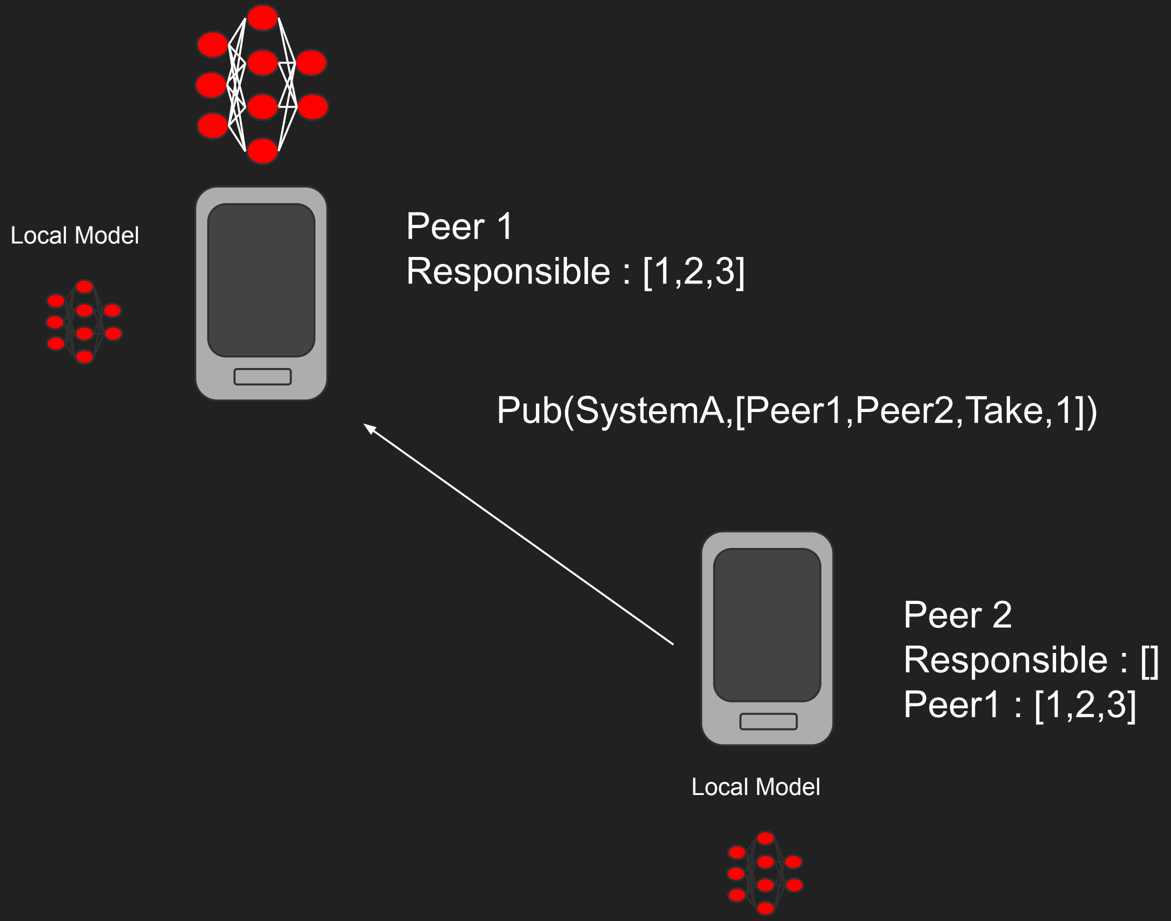
IPLS can be partitioned in three basic phases :

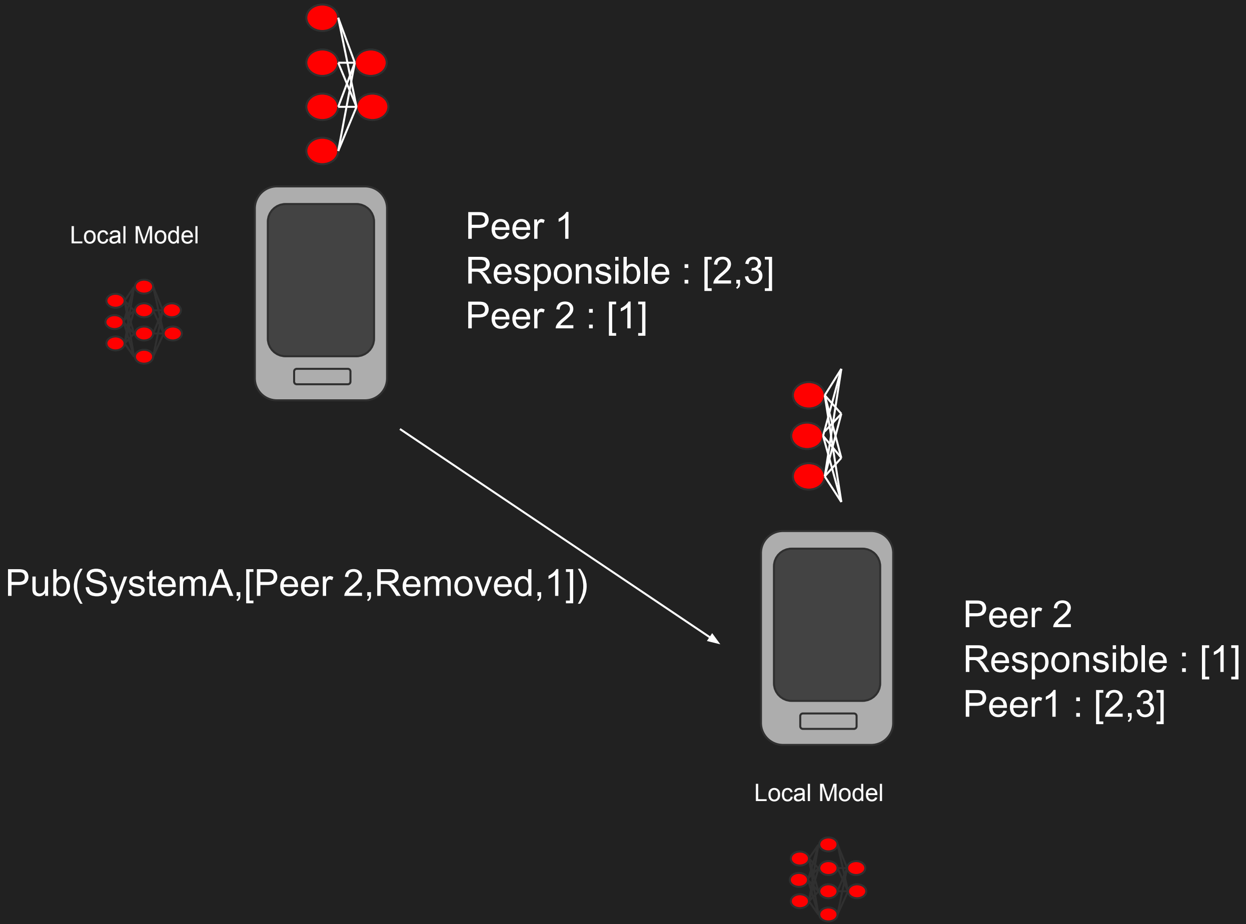
- **Start-up phase** : The application programmer or the data scientist selects the machine learning model, its hyperparameters, the data attributes and then he uploads the project.
- **Initialization phase** : In this phase every peer learns about the responsibilities of the other peers and also selects the partitions of the model that will be responsible for. Moreover if enough peers gathered, then they can proceed to the next phase.
- **Training phase** : The peers using their own data train the model using SGD and then collaborate with each other in order to give a global model.

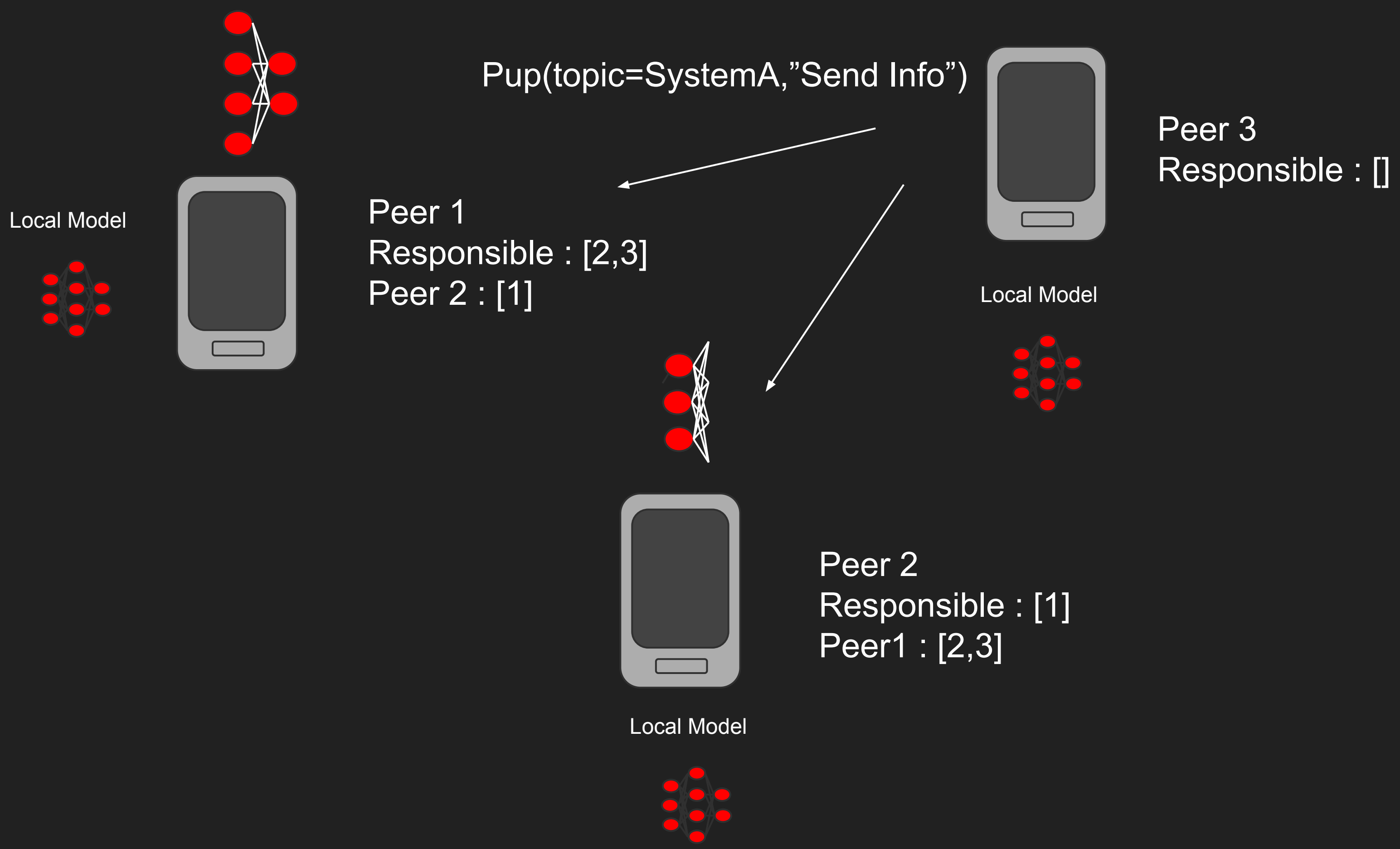
Responsibilities Distribution Protocol

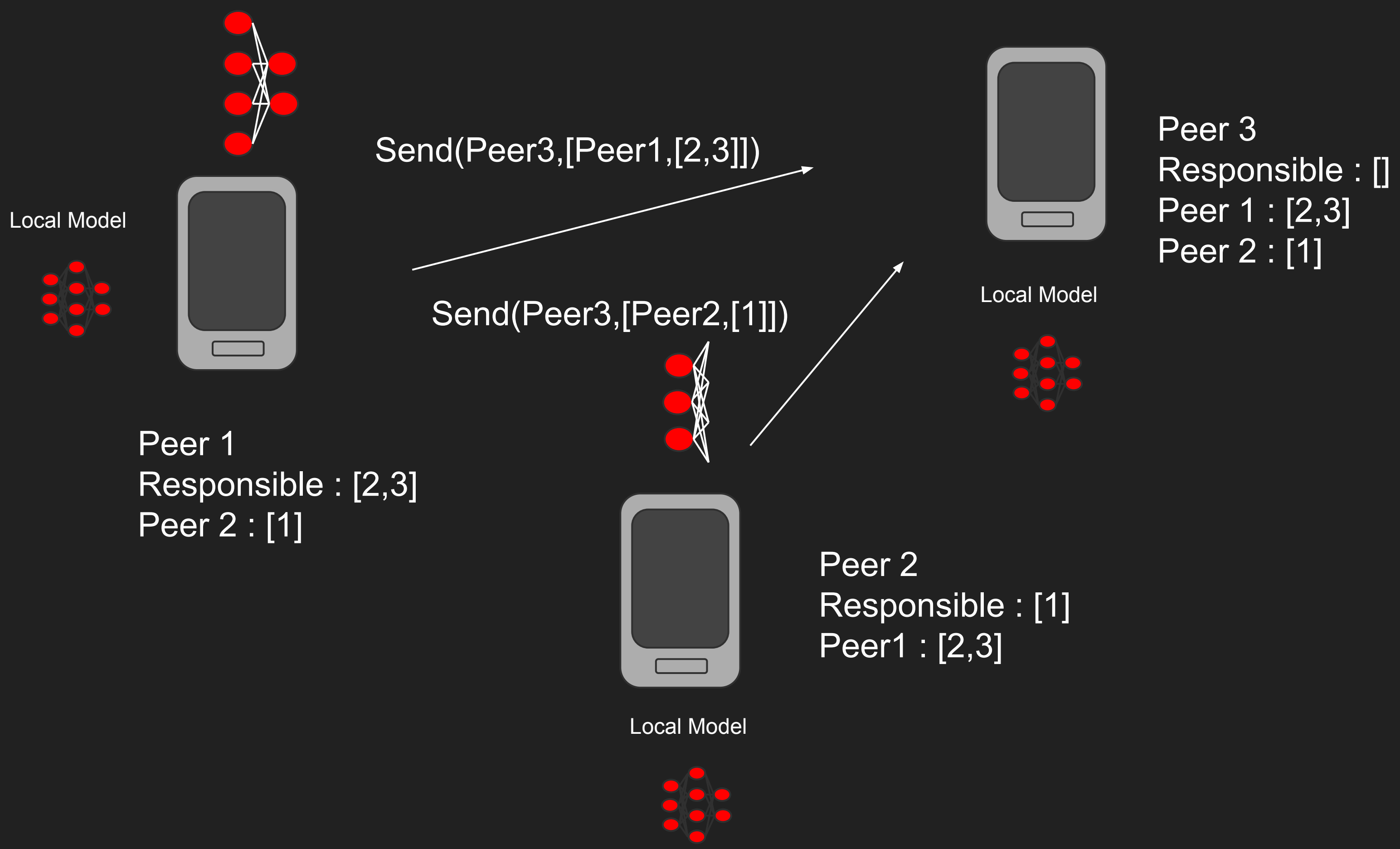


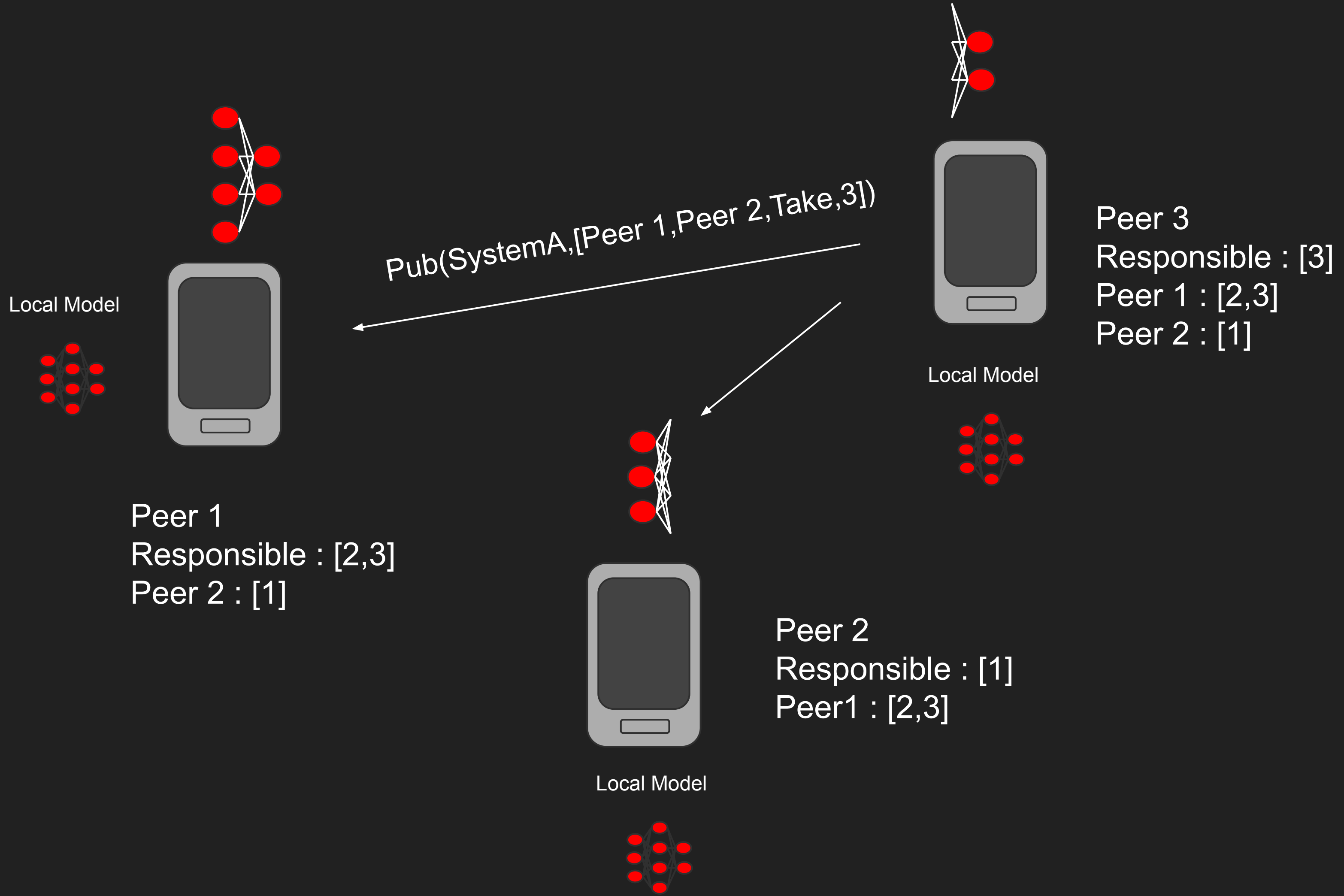

Local Model
Peer 1
Responsible : [1,2,3]
Send(Peer2,[Peer1,[1,2,3]])
Peer 2
Responsible : []
Local Model


Local Model
Peer 1
Responsible : [1,2,3]
Pub(SystemA,[Peer1,Peer2,Take,1])
Peer 2
Responsible : []
Peer1 : [1,2,3]
Local Model

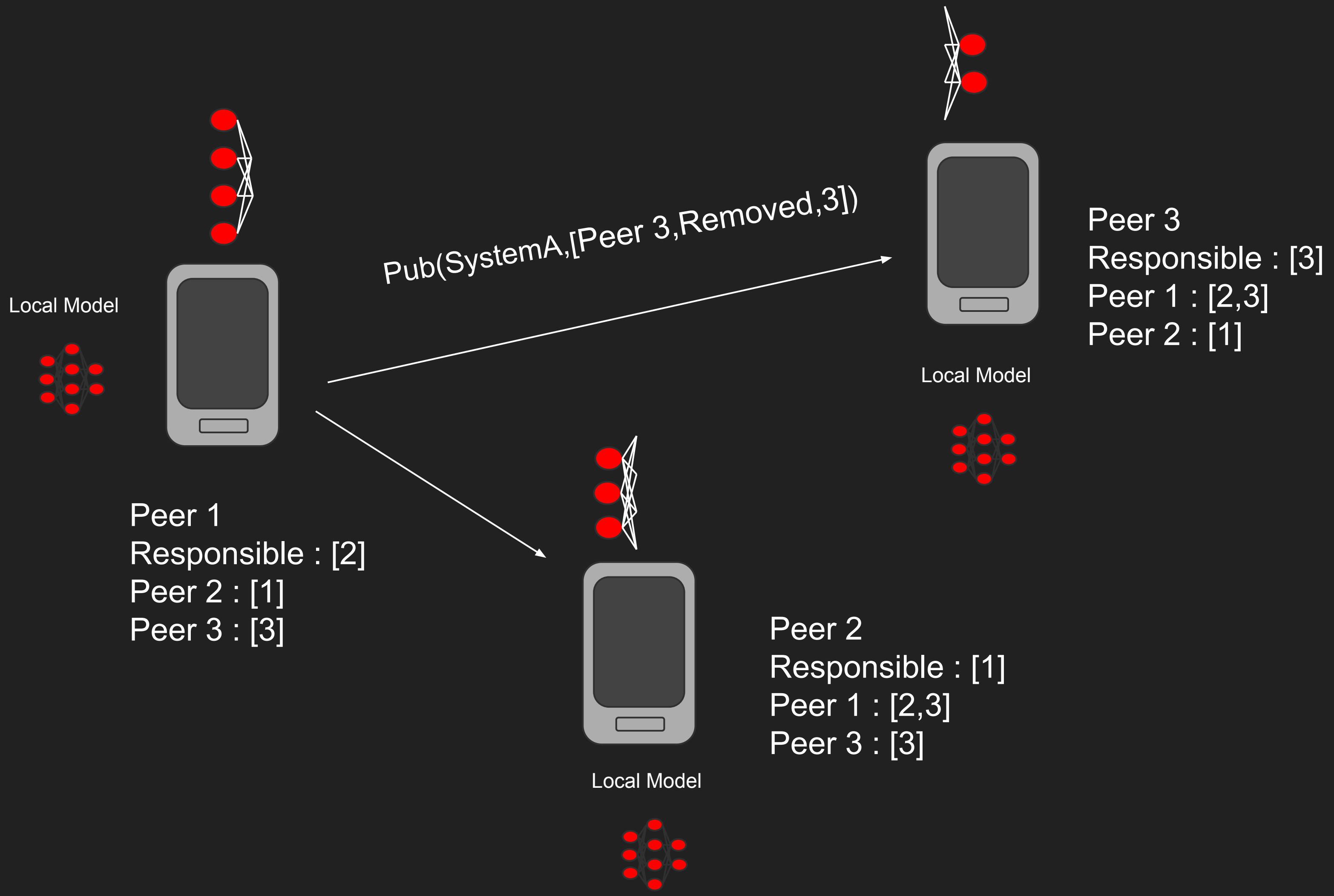

Local Model
Peer 1
Responsible : [2,3]
Peer 2 : [1]
Pub(SystemA,[Peer 2,Removed,1])
Peer 2
Responsible : [1]
Peer1 : [2,3]
Local Model

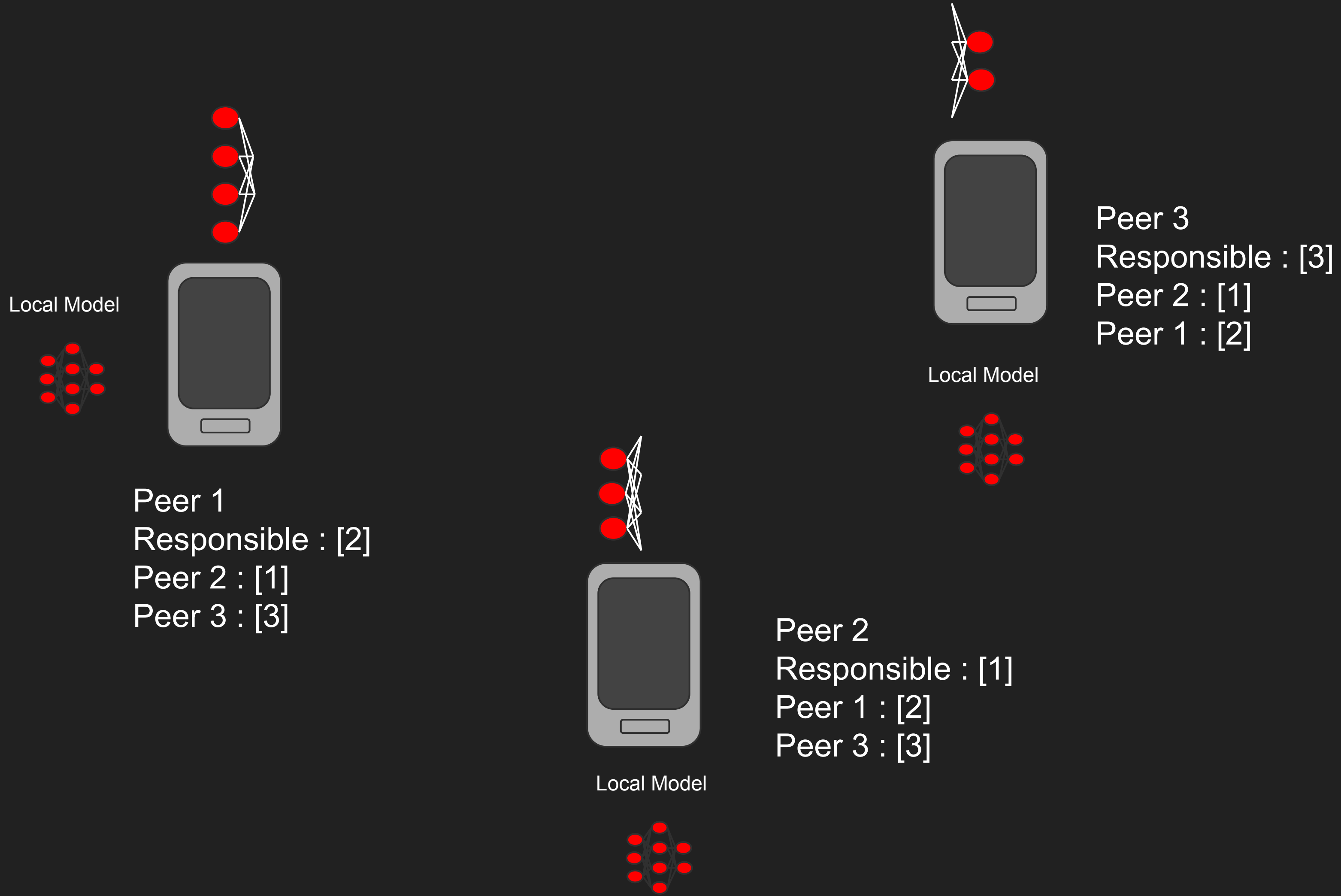

Pup(topic=SystemA,"Send Info")
Peer 3
Responsible : []
Local Model
Peer 1
Responsible : [2,3]
Peer 2 : [1]
Local Model
Peer 2
Responsible : [1]
Peer1 : [2,3]
Local Model


Send(Peer3,[Peer1,[2,3]])
Send(Peer3,[Peer2,[1]])
Local Model
Peer 1
Responsible : [2,3]
Peer 2 : [1]
Peer 3
Responsible : []
Peer 1 : [2,3]
Peer 2 : [1]
Local Model
Peer 2
Responsible : [1]
Peer1 : [2,3]
Local Model



Pub(SystemA,[Peer 1,Peer 2,Take,3])
Local Model
Peer 1
Responsible : [2,3]
Peer 2 : [1]
Peer 3
Responsible : [3]
Peer 1 : [2,3]
Peer 2 : [1]
Local Model
Peer 2
Responsible : [1]
Peer1 : [2,3]
Local Model


Pub(SystemA,[Peer 3,Removed,3])
Local Model
Peer 1
Responsible : [2]
Peer 2 : [1]
Peer 3 : [3]
Peer 3
Responsible : [3]
Peer 1 : [2,3]
Peer 2 : [1]
Local Model
Peer 2
Responsible : [1]
Peer 1 : [2,3]
Peer 3 : [3]
Local Model


Local Model
Peer 1
Responsible : [2]
Peer 2 : [1]
Peer 3 : [3]
Peer 3
Responsible : [3]
Peer 2 : [1]
Peer 1 : [2]
Local Model
Local Model
Peer 2
Responsible : [1]
Peer 1 : [2]
Peer 3 : [3]

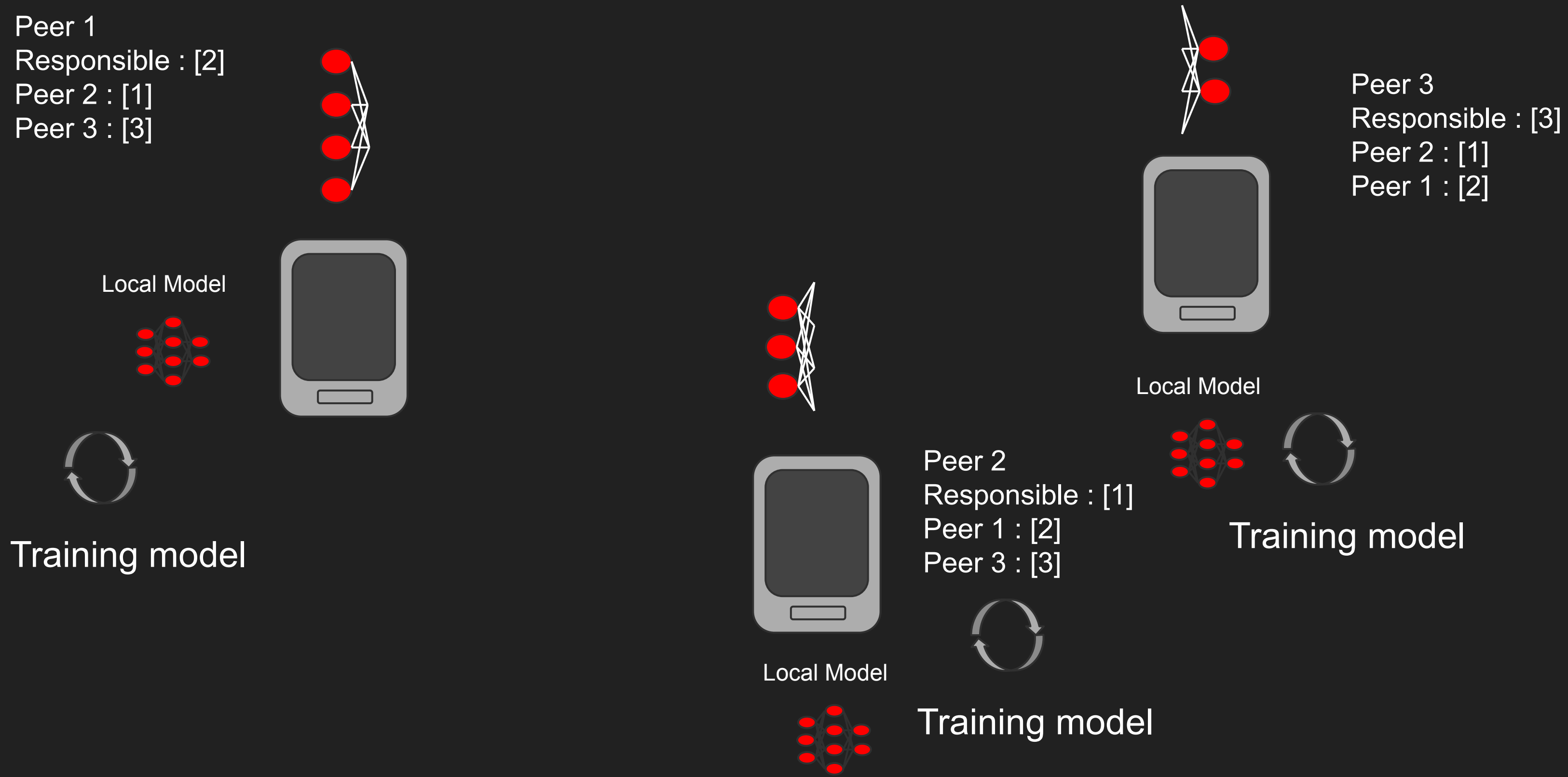
Initialization Phase

- There is only one last step before proceeding to the training. This step is the selection of the **Dealers**, who are peers responsible for the partitions that i am not responsible for. This is done when you call for the first time the method **get_partitions** of the **IPLS** class. (**Check the code**)
- Peers can also discard some of their responsibilities so they inform their “clients” to stop sending them gradients for the partitions that they discarded.
- When a peer receives a discard notification, he deletes his discarded **dealer** and in the next iteration searches for a new dealer.

Training Phase

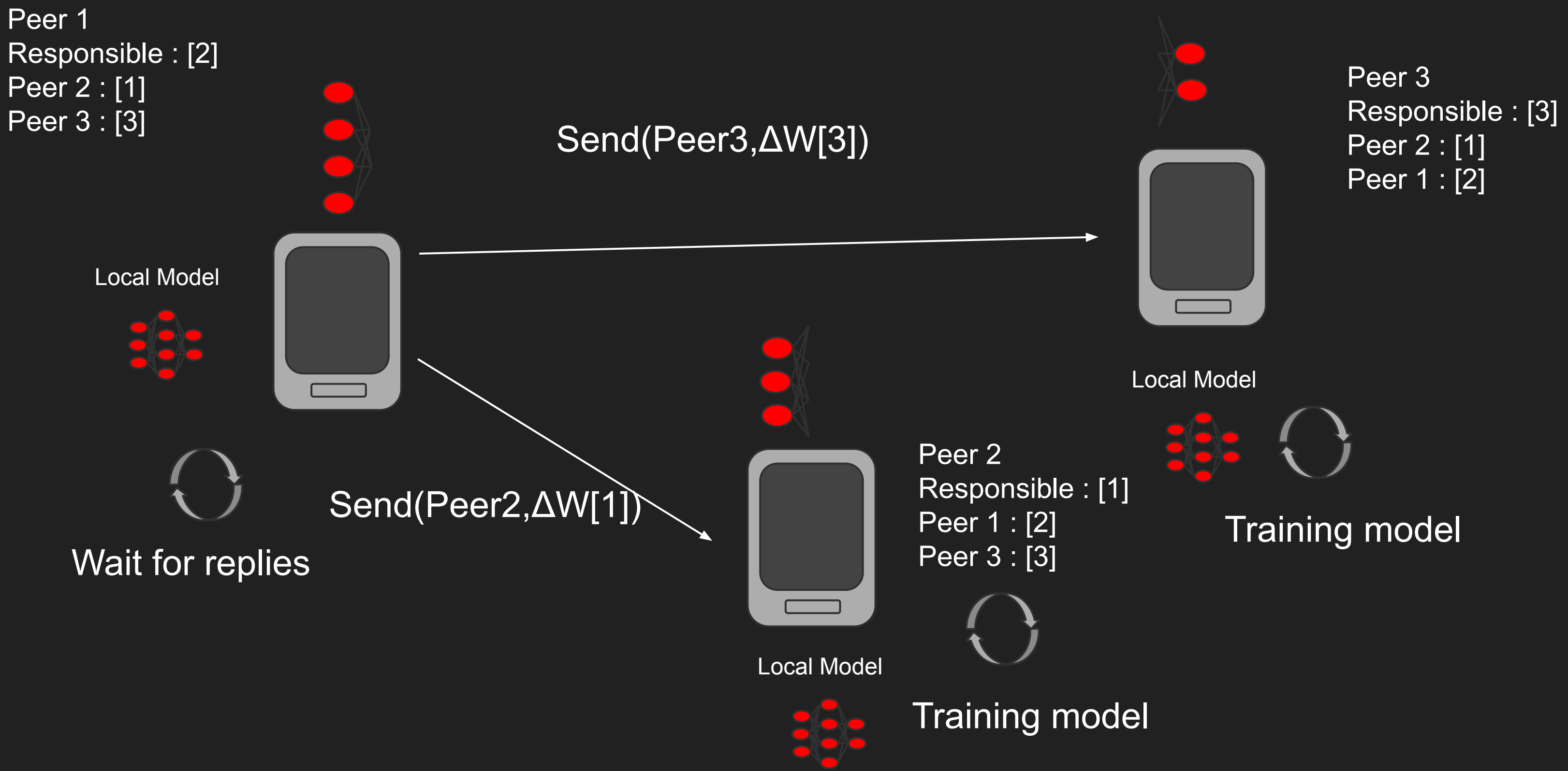
- Each peer trains the model locally using its own private data. Then as we said he computes the difference ΔW.Then he calls the UpdateModel(ΔW) from IPLS API
- He searches peers for partitions that he is not responsible, and sends the corresponding partition of ΔW to that peer
- There is a possibility that there may be many peers with the same partition. The selection policies of the peer depend heavily on the type of training (Asynchronous or Synchronous), peer capabilities and performance etc and also on application
- This is done on the **UpdateGradient** method.

Synchronous SGD

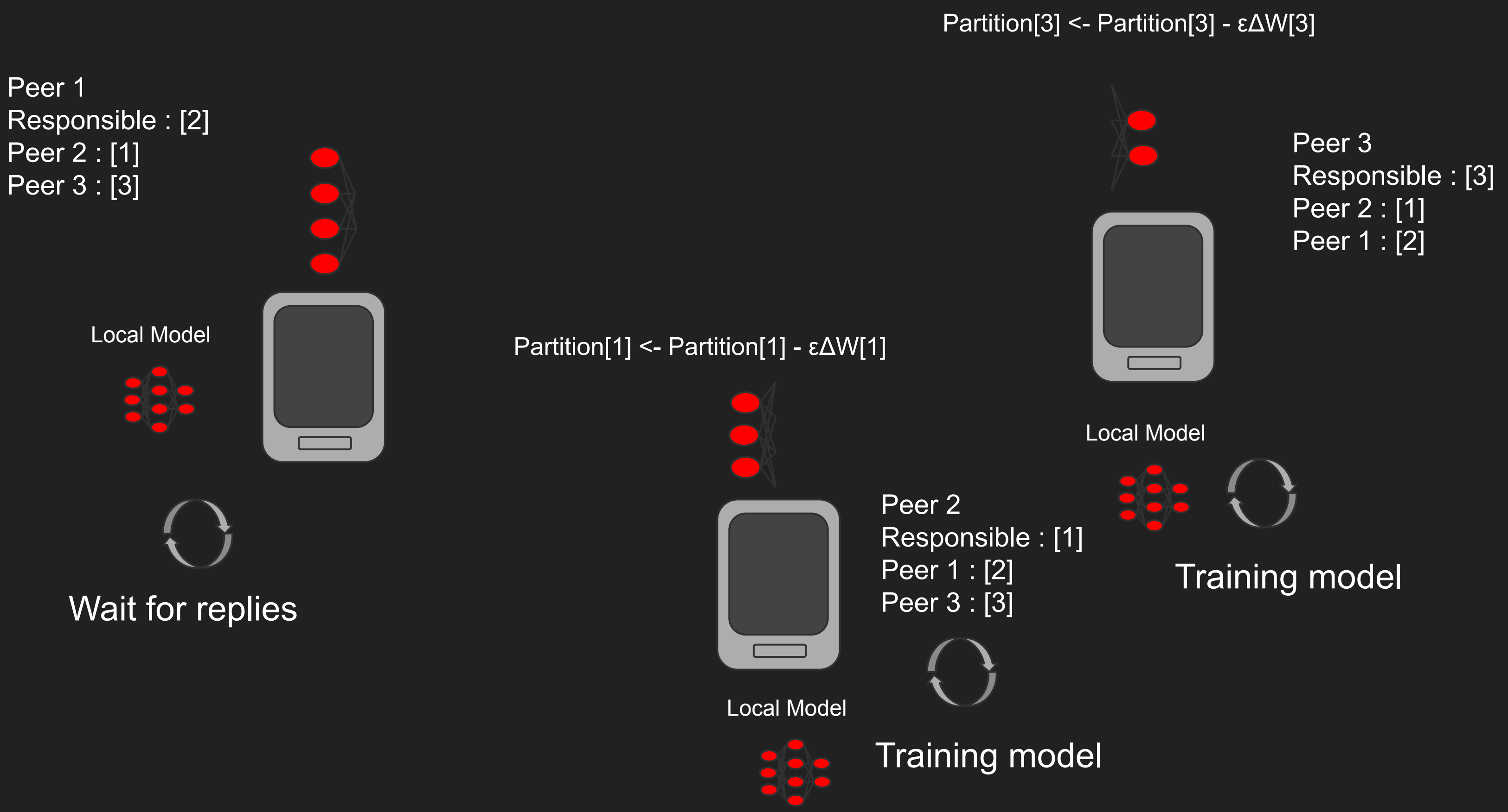


Synchronous SGD

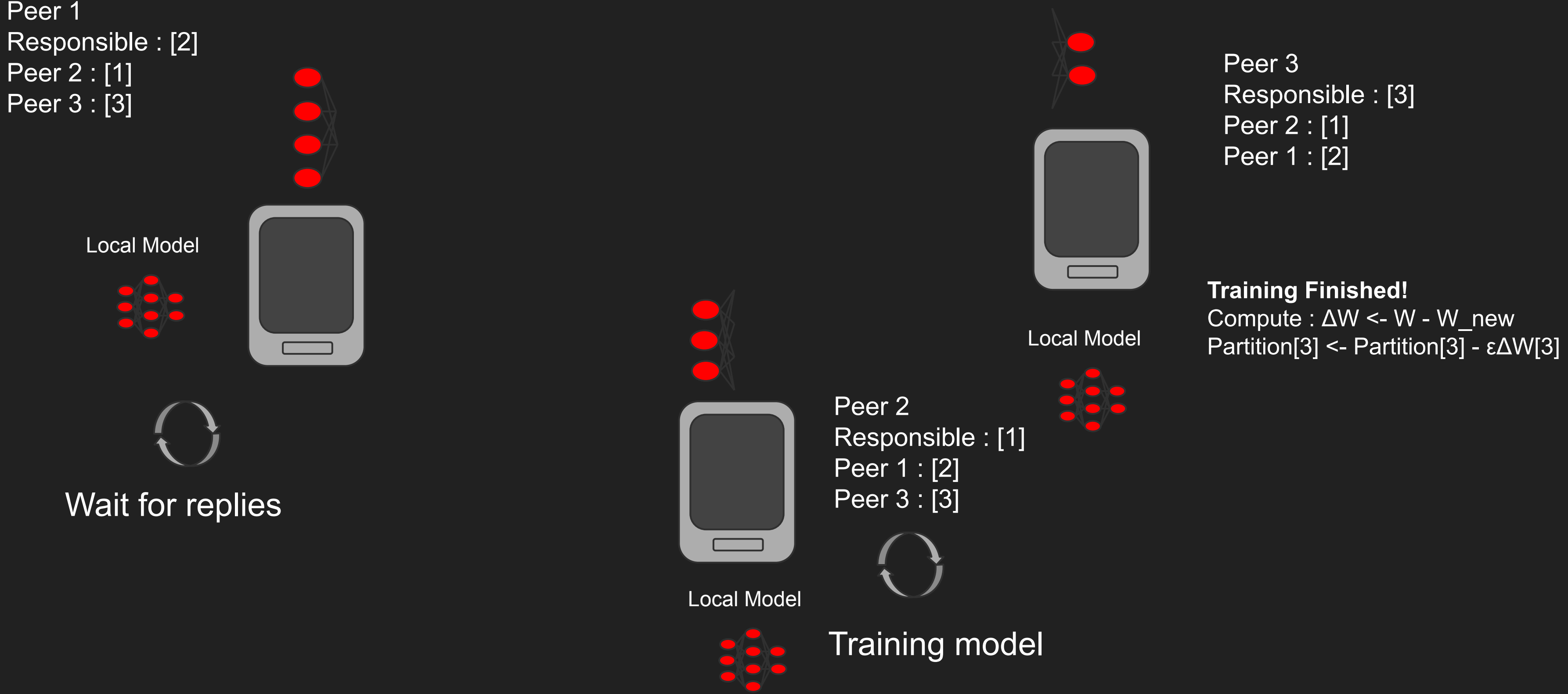
Synchronous SGD



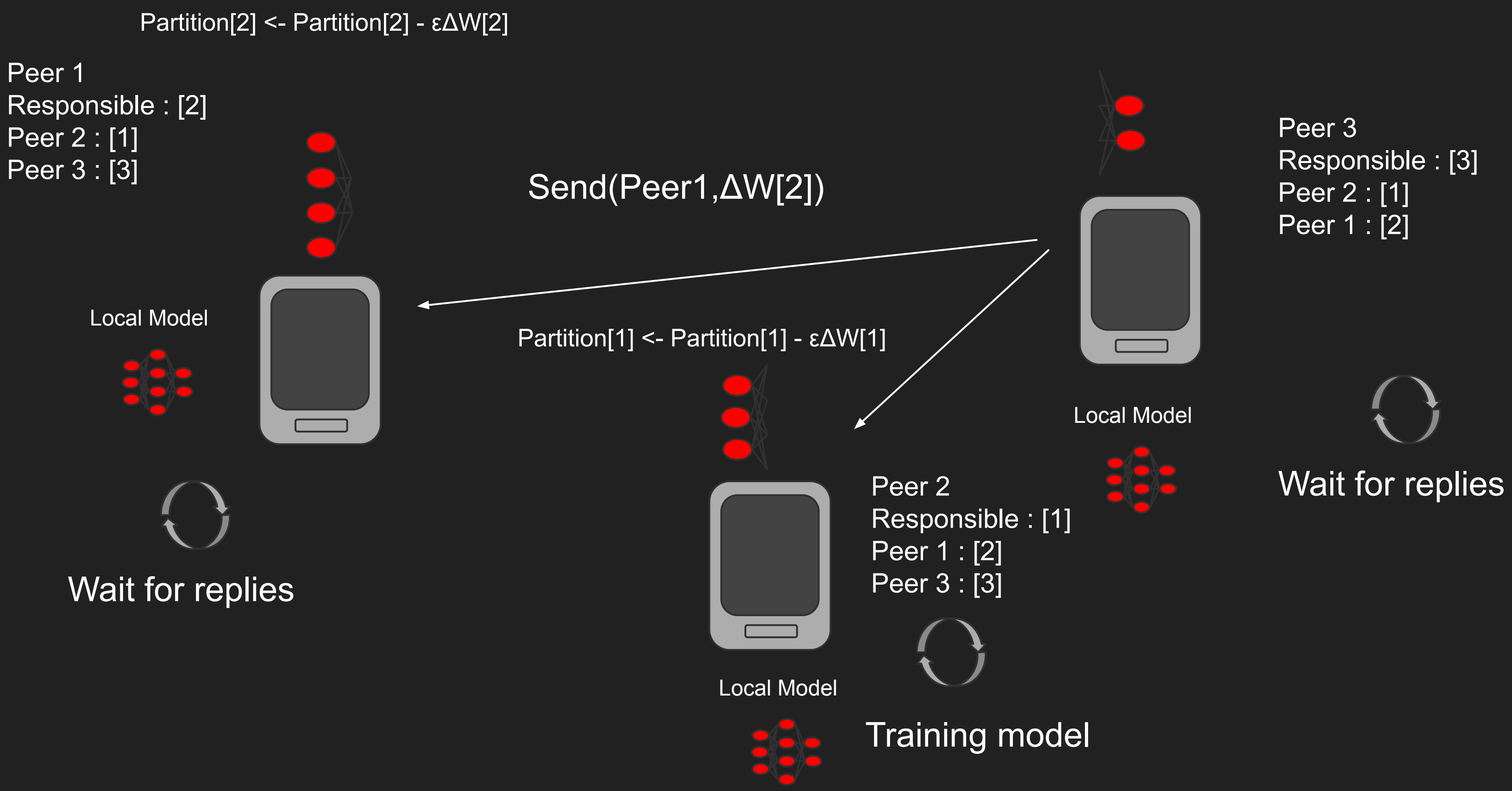
Synchronous SGD



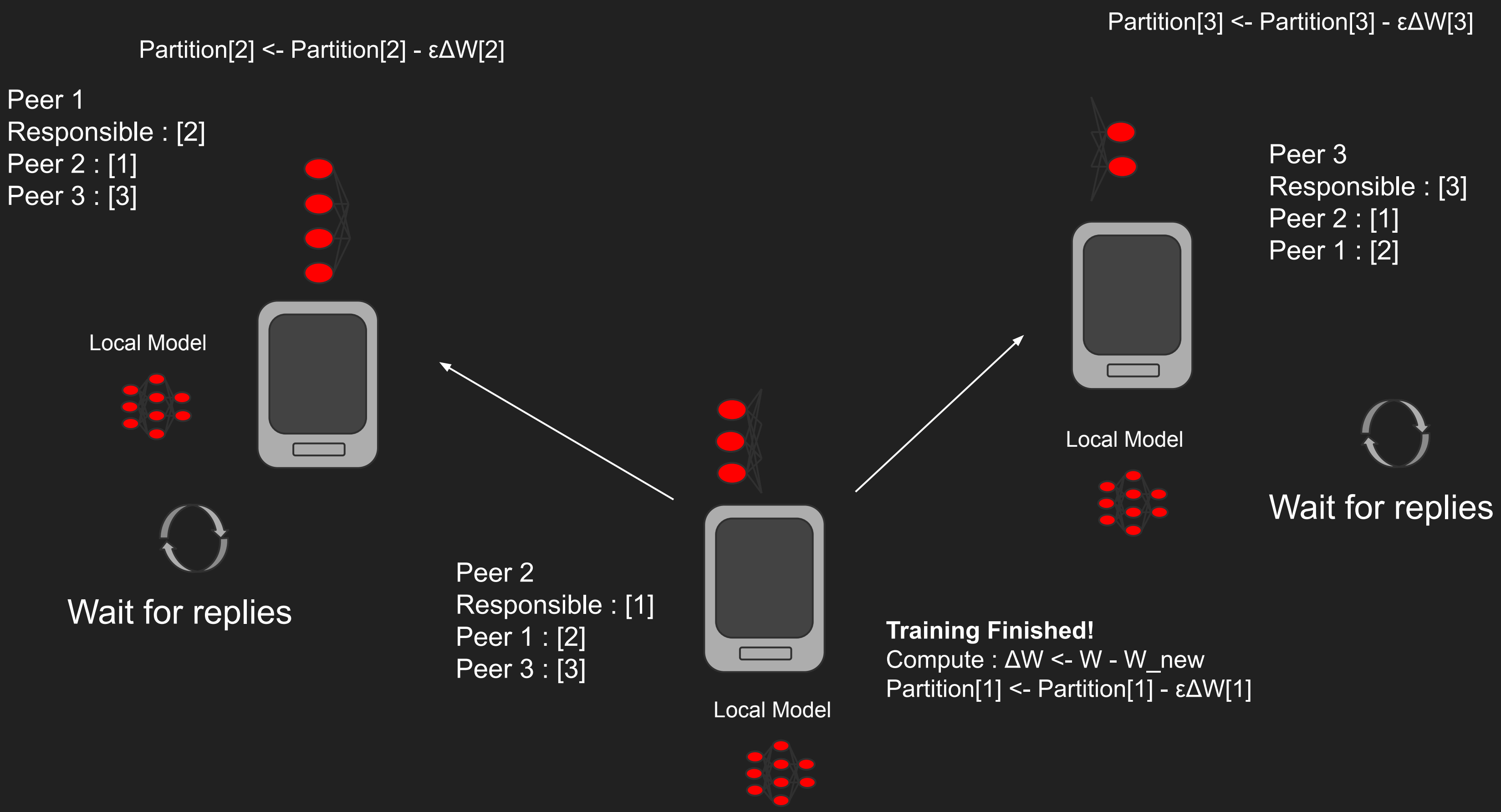
Synchronous SGD



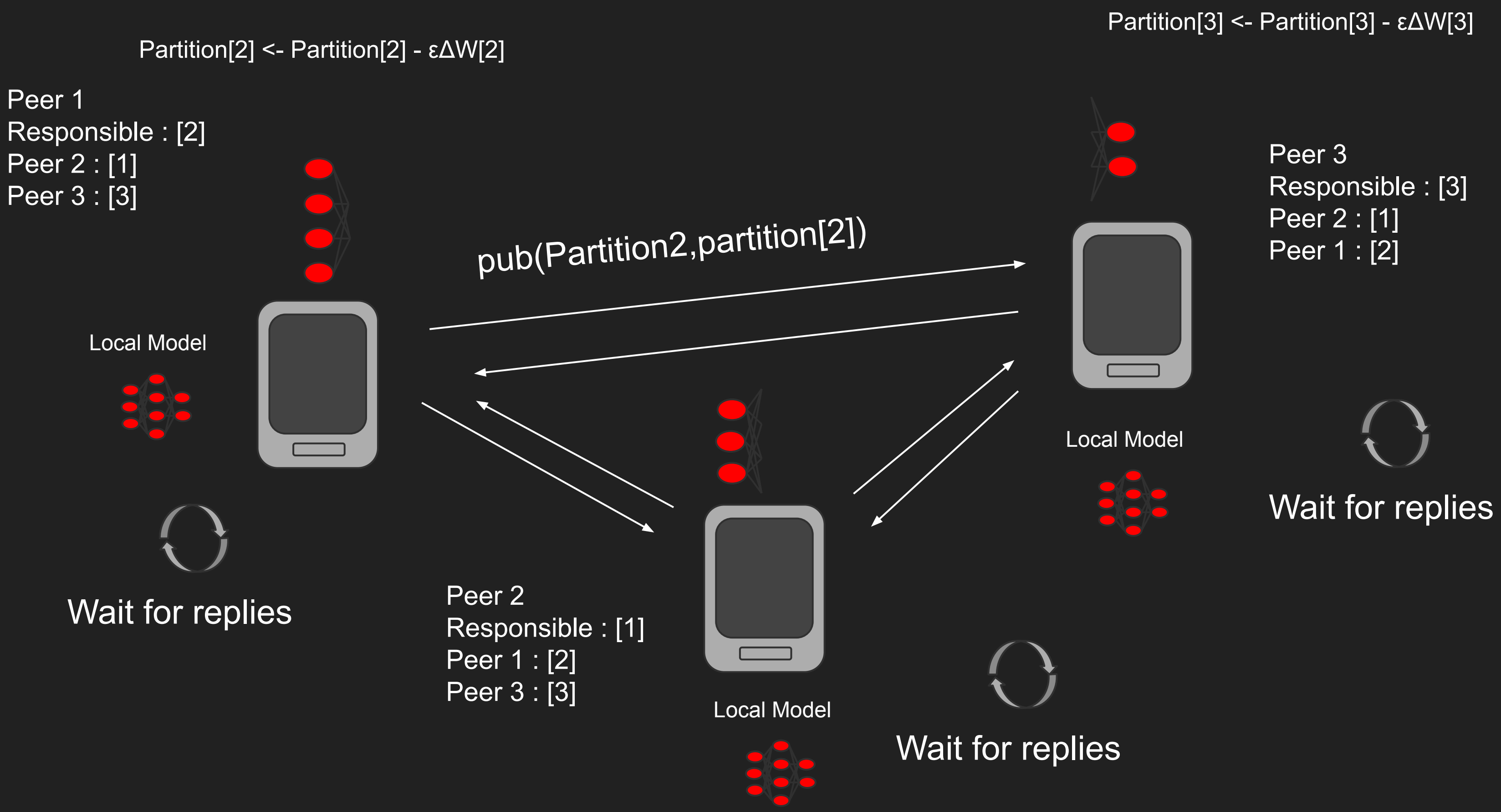
Synchronous SGD



Synchronous SGD

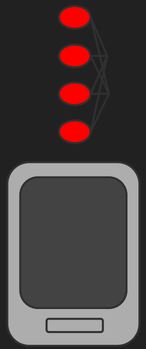


Synchronous SGD



Synchronous SGD

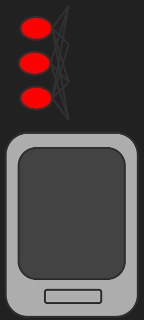
Peer 1
Responsible : [2]
Peer 2 : [1]
Peer 3 : [3]



Local Model

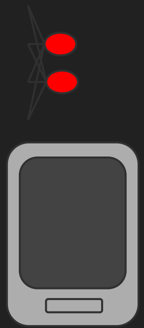
W <- { Partition[1],
Partition[2],
Partition[3] }
W_new <- Fit(W,Data)

Peer 2
Responsible : [1]
Peer 1 : [2]
Peer 3 : [3]



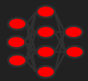
Local Model

W <- { Partition[1],
Partition[2],
Partition[3] }
W_new <- Fit(W,Data)



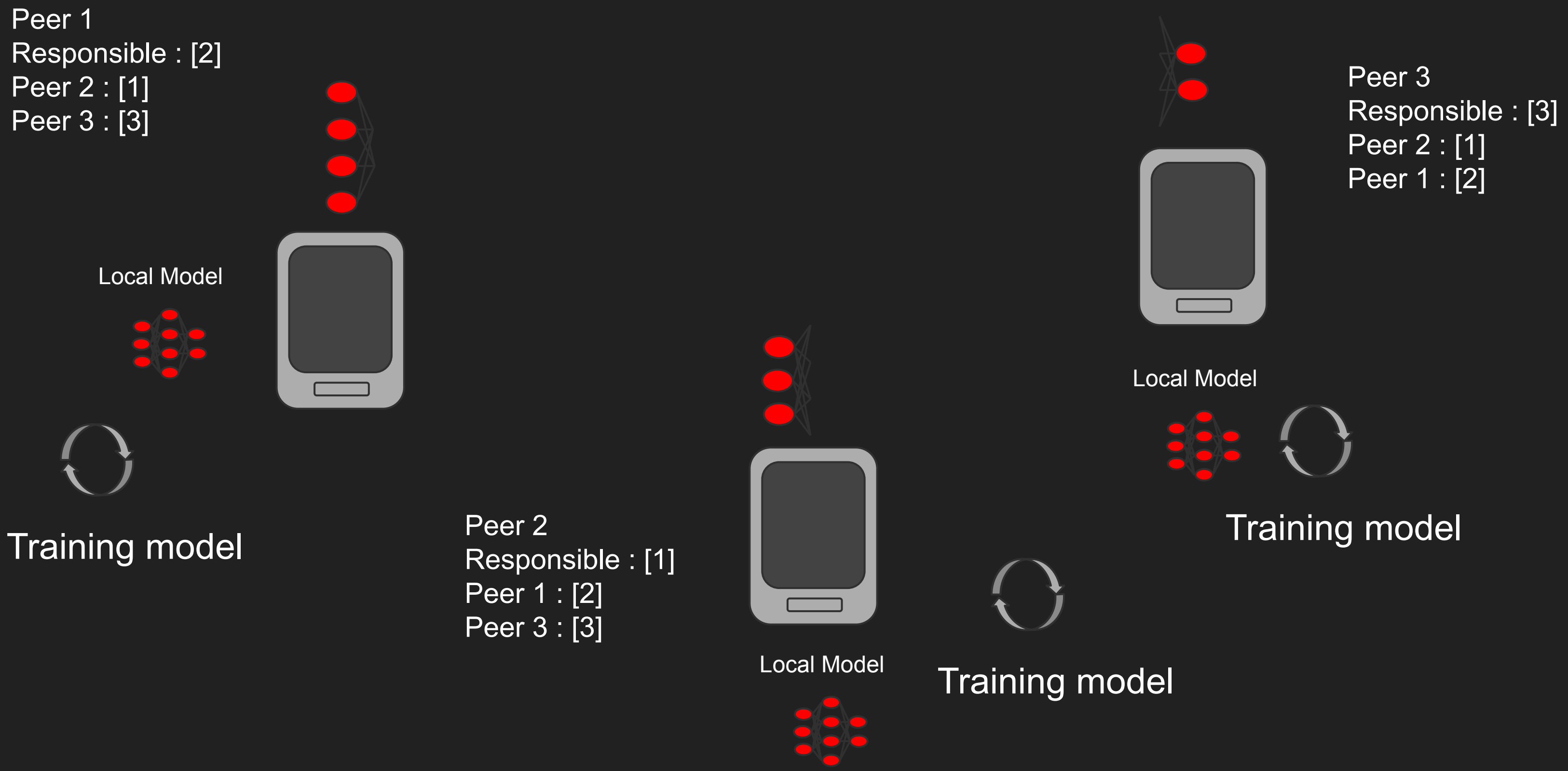
Peer 3
Responsible : [3]
Peer 2 : [1]
Peer 1 : [2]

Local Model

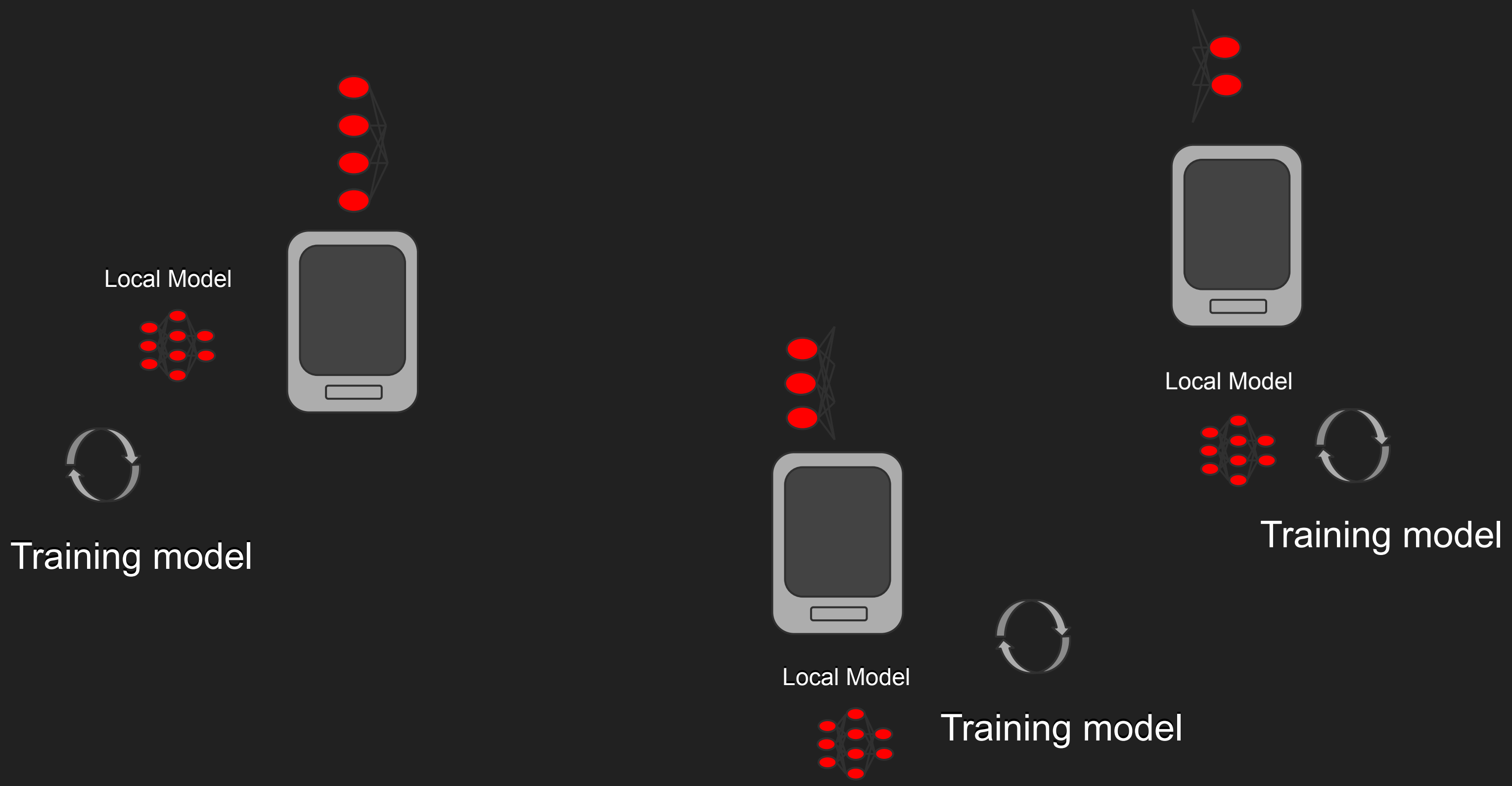


W <- { Partition[1],
Partition[2],
Partition[3] }
W_new <- Fit(W,Data)

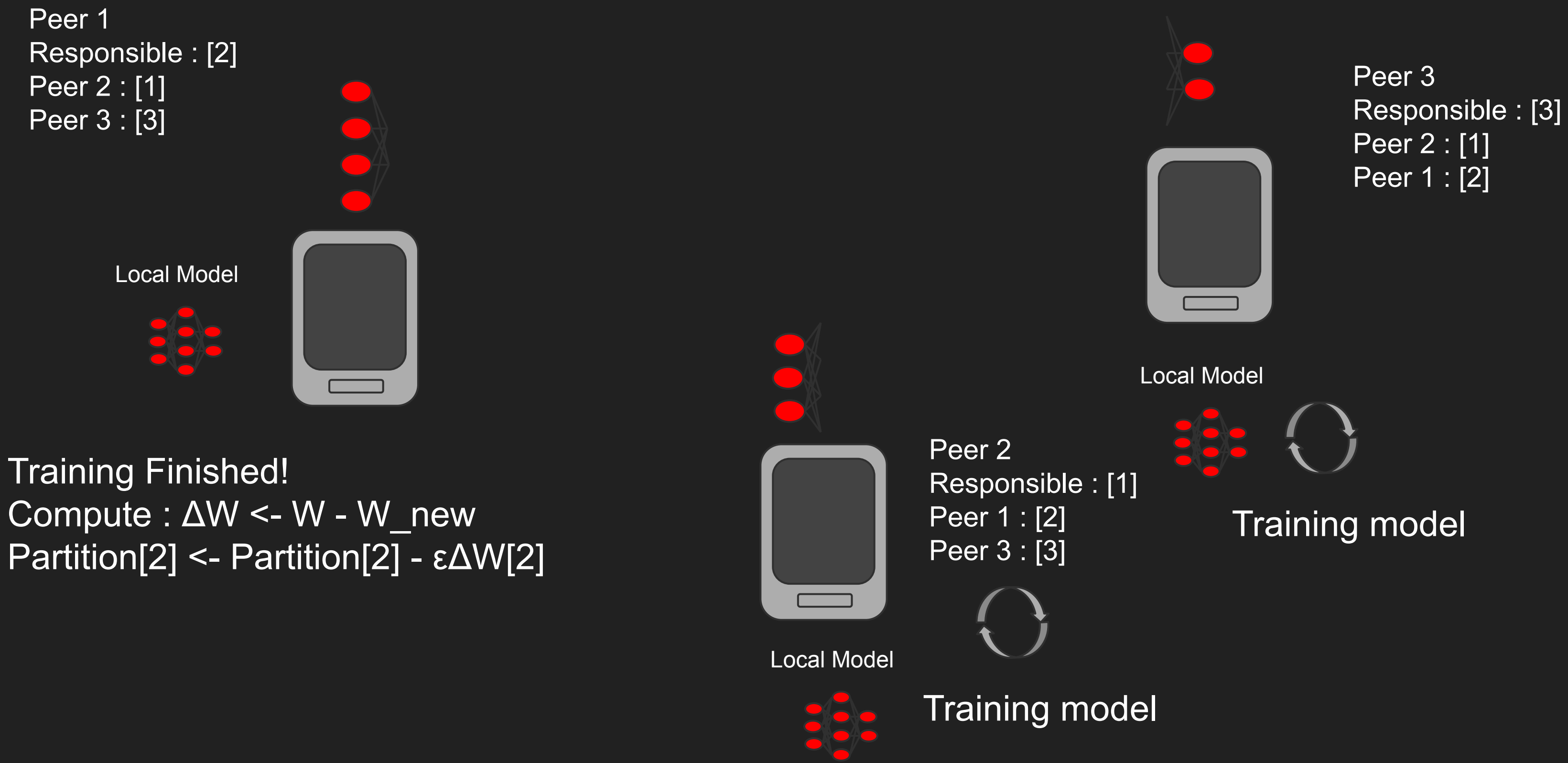
Synchronous SGD



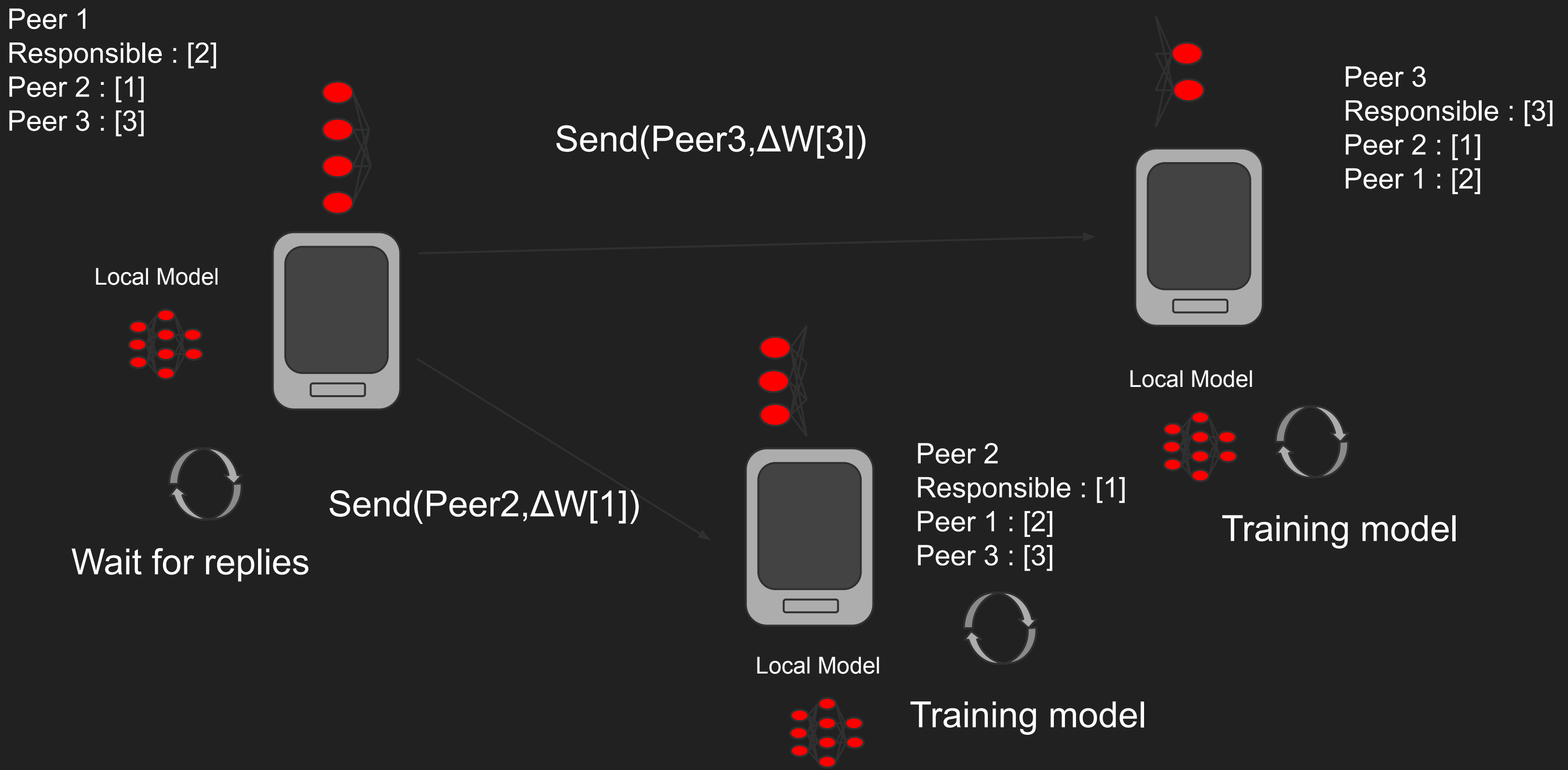
Asynchronous SGD



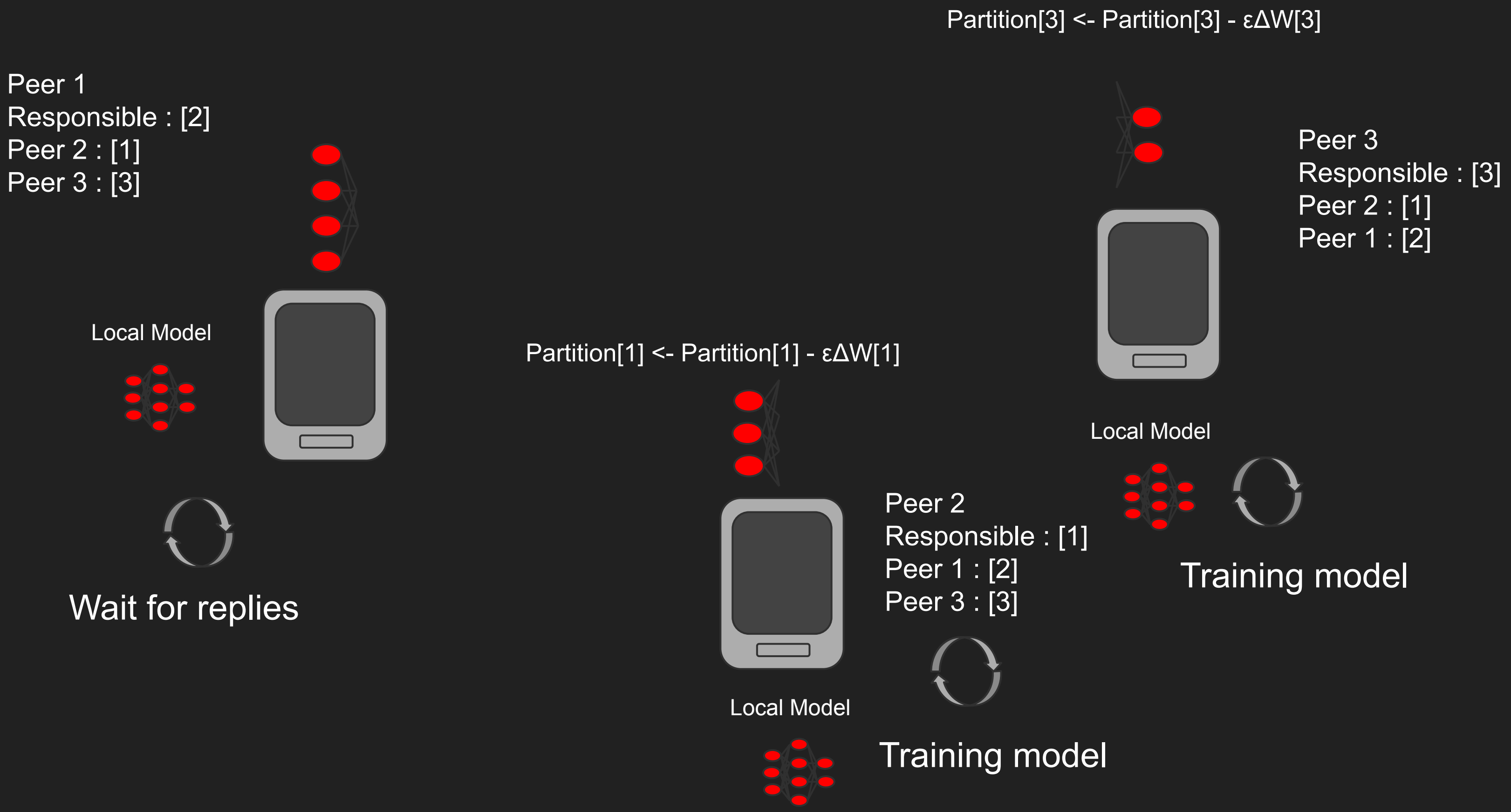
Asynchronous SGD



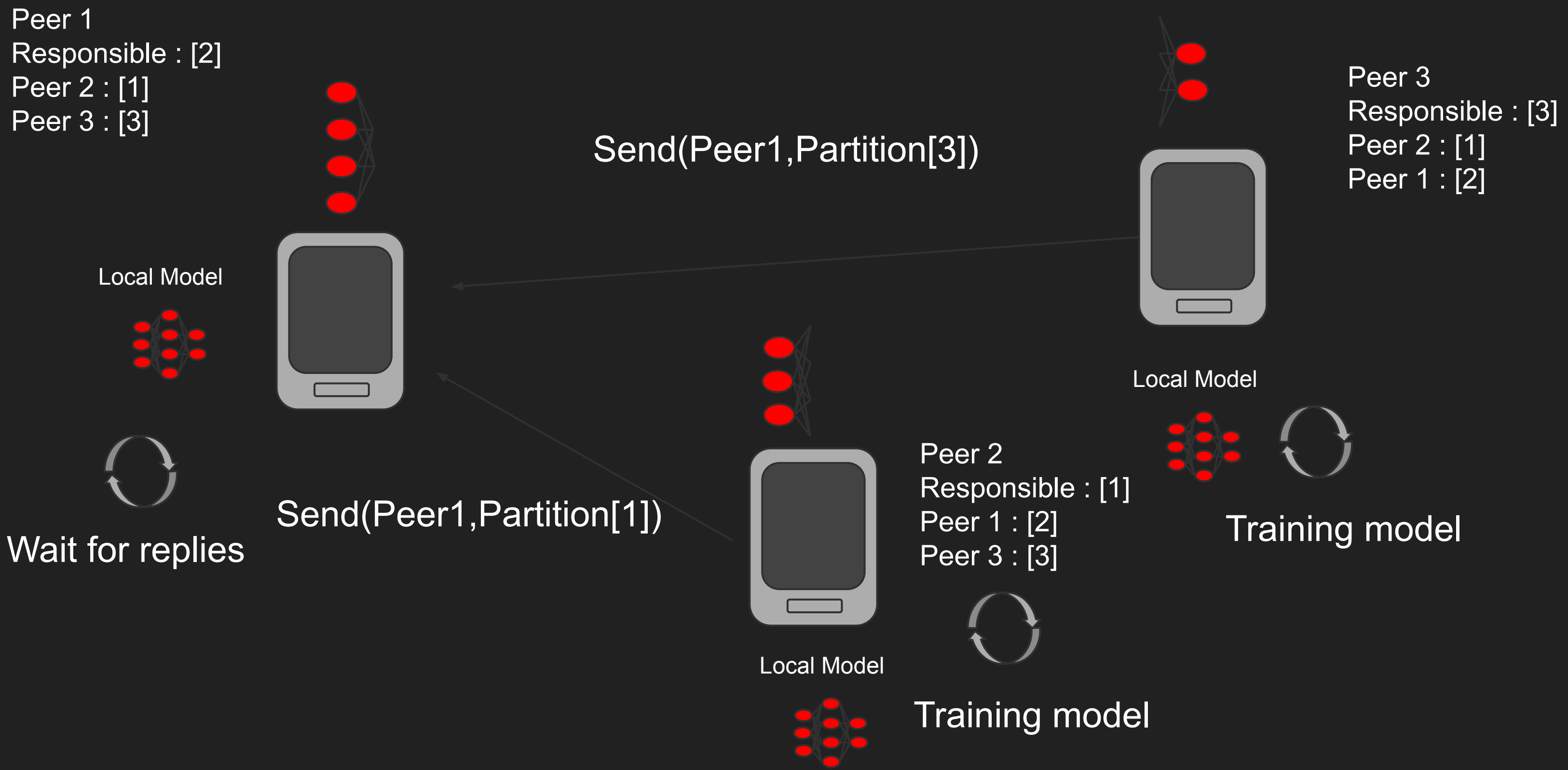
Asynchronous SGD



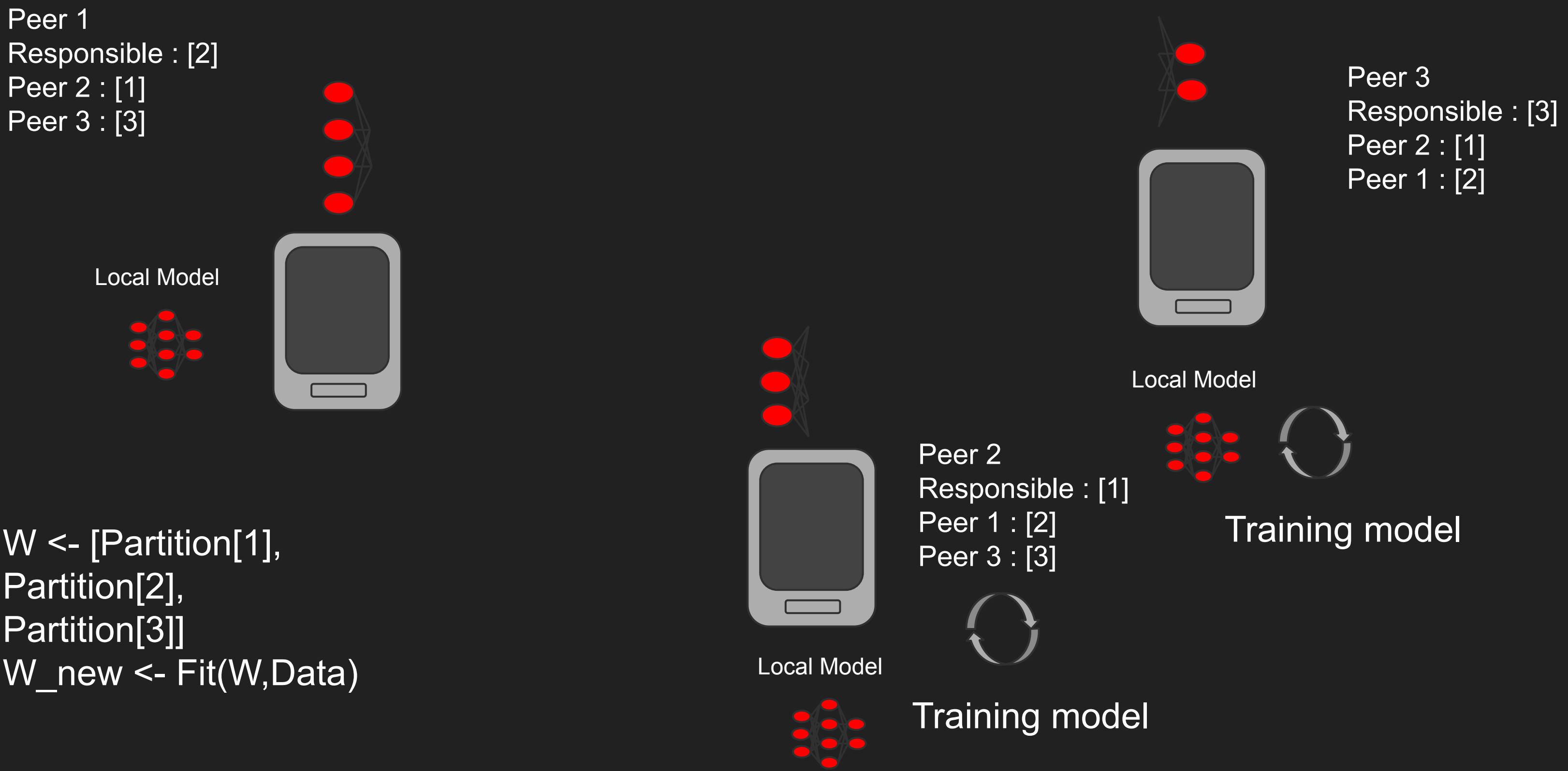
Asynchronous SGD



Asynchronous SGD



Asynchronous SGD



W <- [Partition[1],
Partition[2],
Partition[3]]
W_new <- Fit(W,Data)

Asynchronous SGD