

# Distributed Machine Learning

on

## Unikernel for IoT

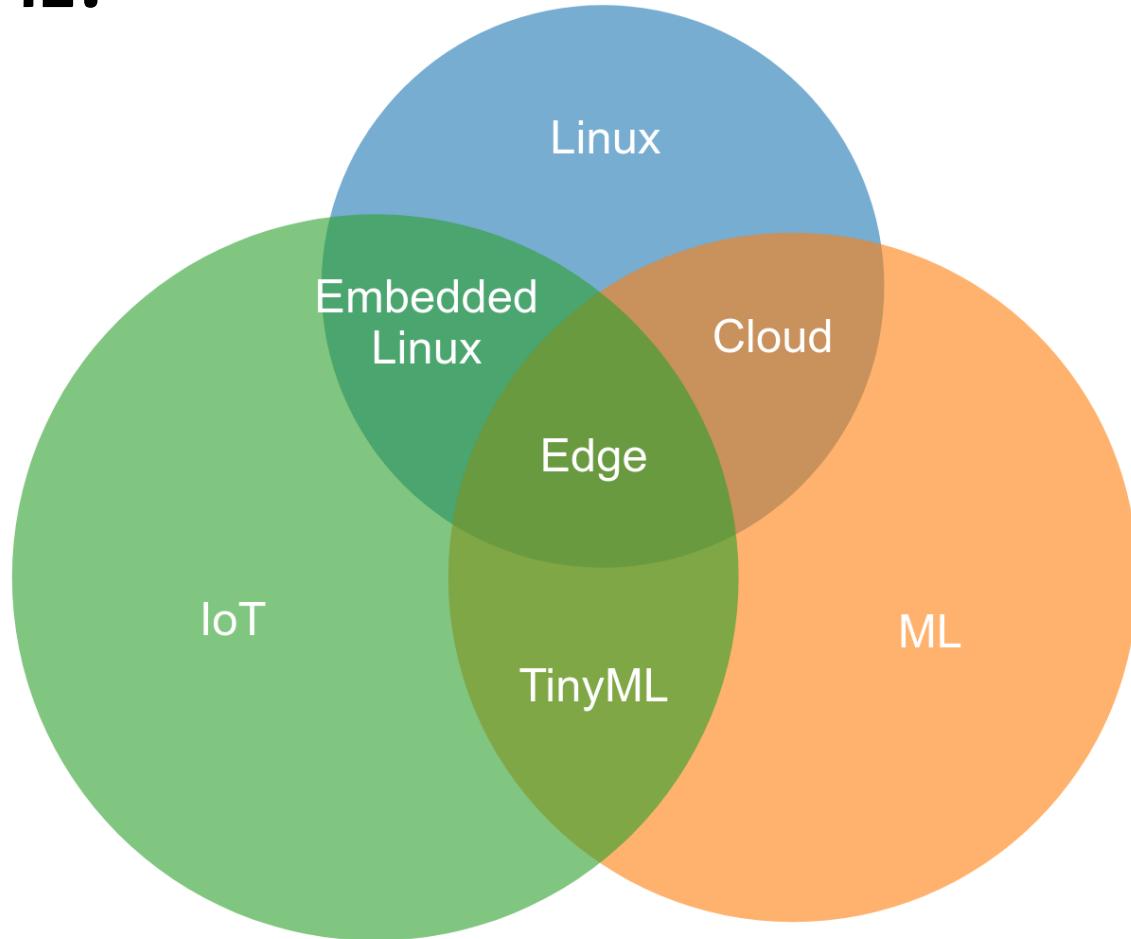
---

*TinyML as-a-Service*

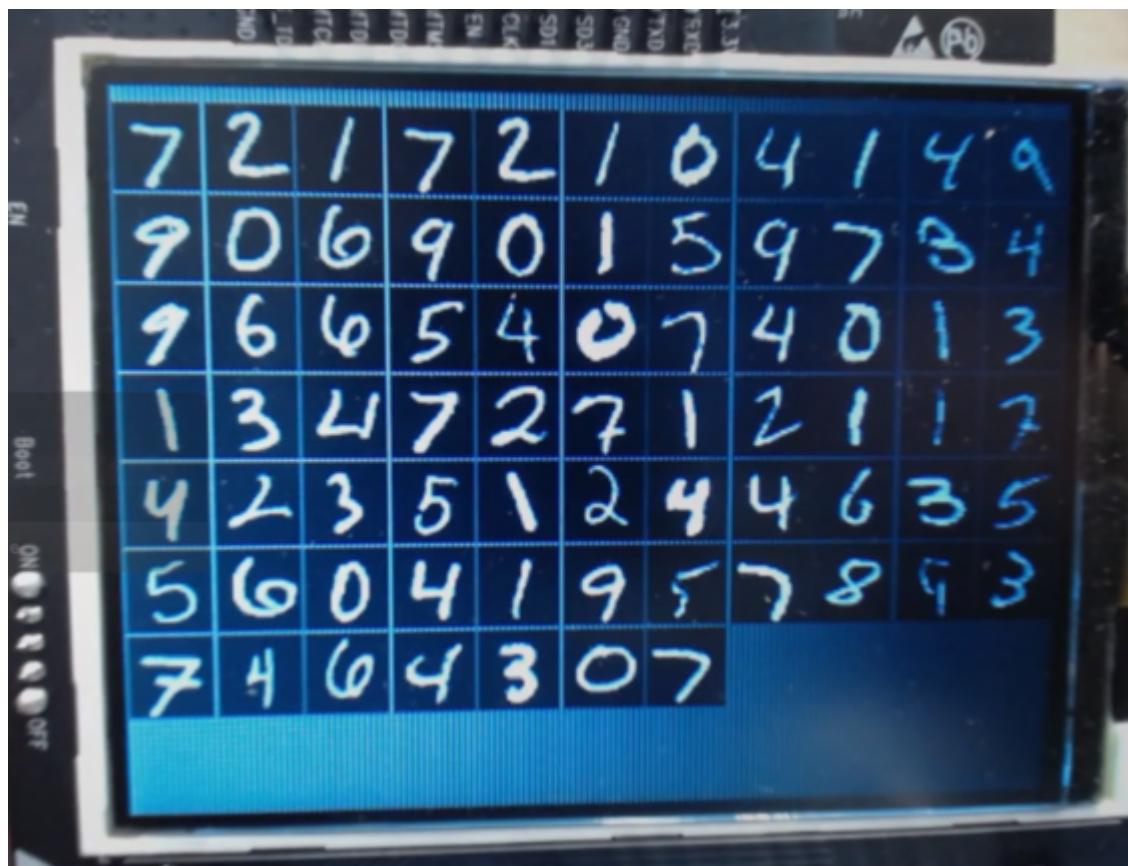
---

Hiroshi Doyu <[hiroshi.doyu@ericsson.com](mailto:hiroshi.doyu@ericsson.com)>

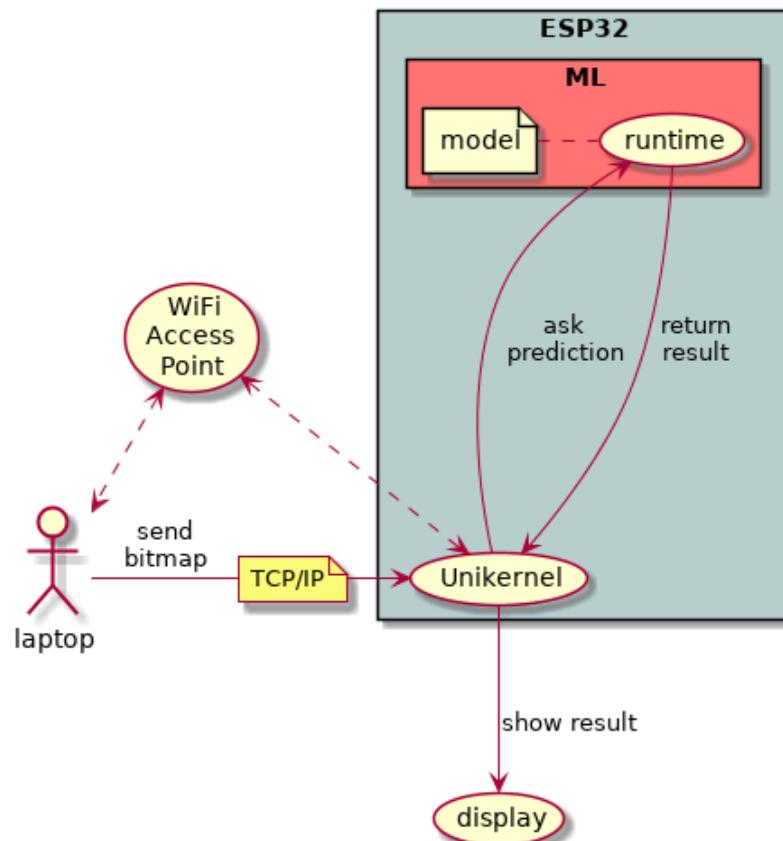
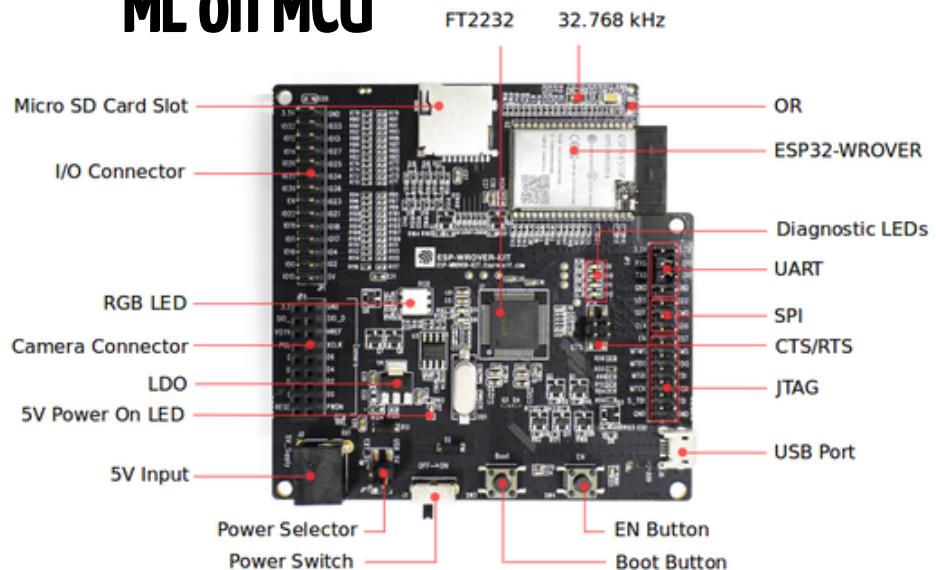
# TinyML?



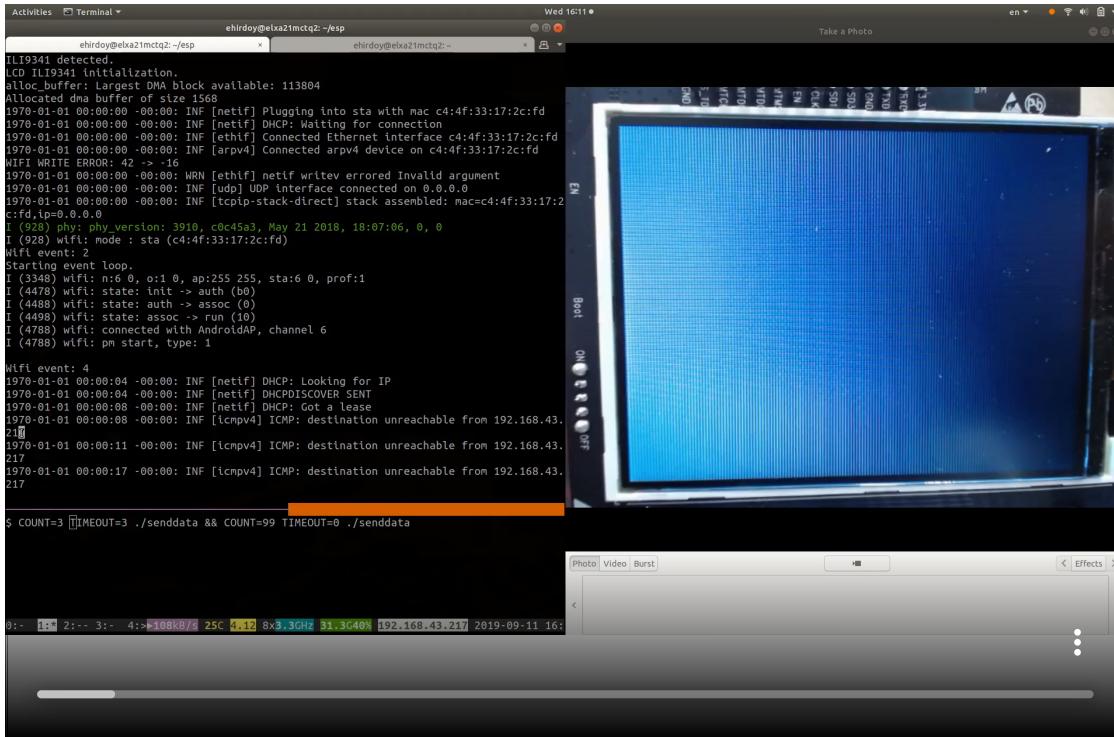
# Demo



# ML on MCU



MCU: ESP32  
ARCH: Xtensa 32bit  
RAM: 520KB SRAM  
ROM: 4MB FLASH  
160MHz / 600 DMIPS

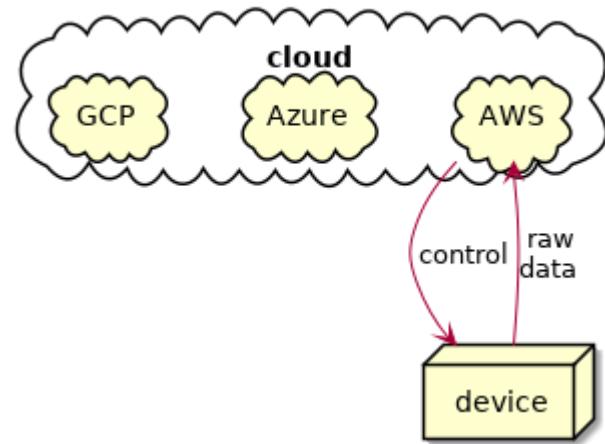


# MNIST on ESP32

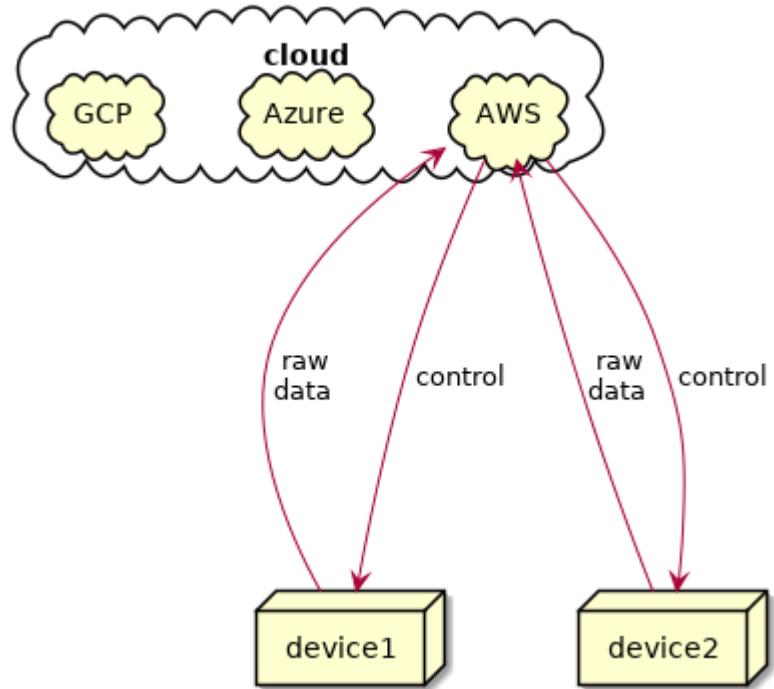
# Outline

1. ~~Demo~~
2. **Problems**
  - Edge Computing
  - Web vs Embedded
  - ML environment
3. Proposal
4. Three Enablers
5. PoC
6. Conclusion

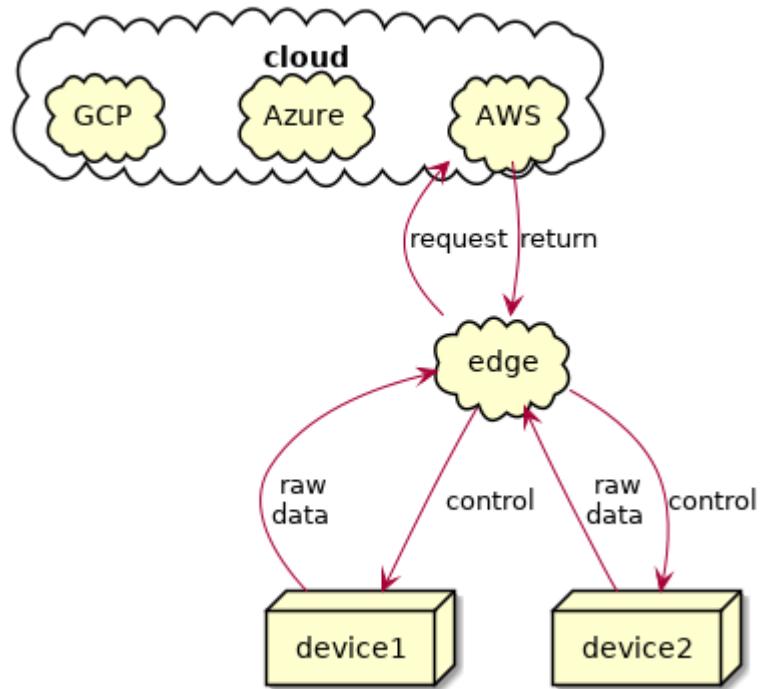
Traditionally



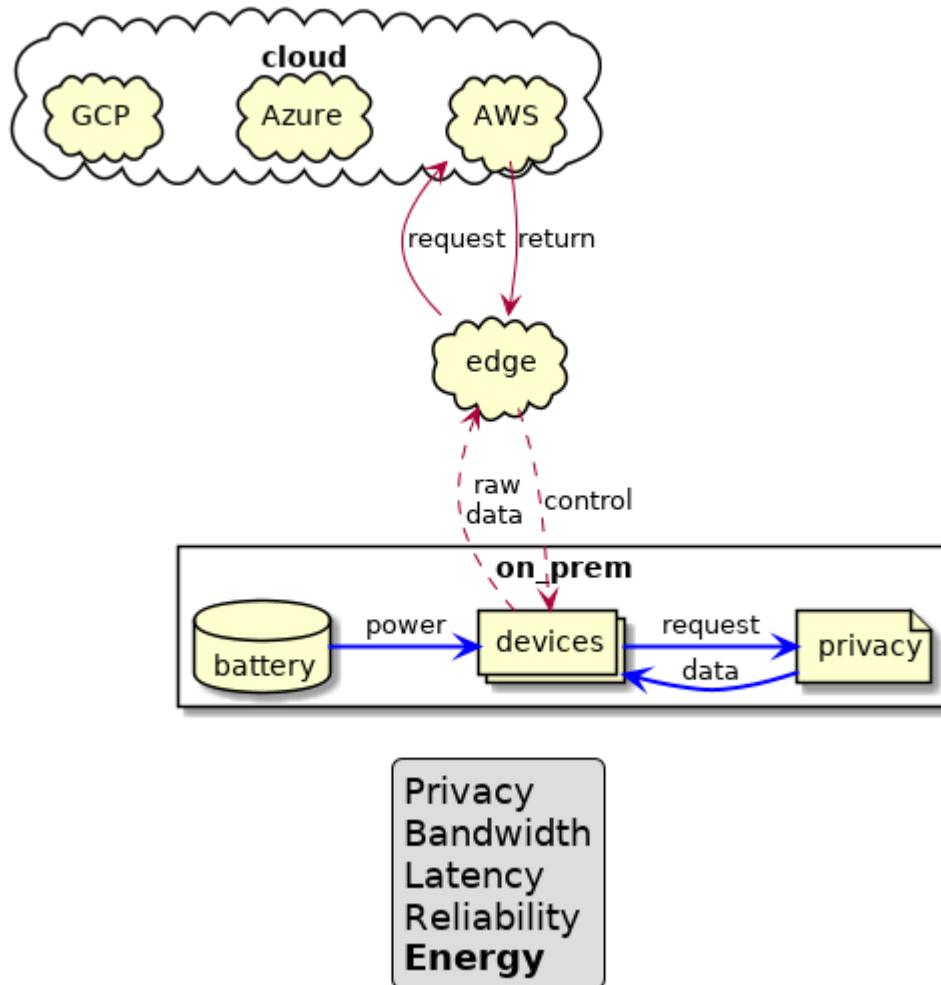
## Scalability

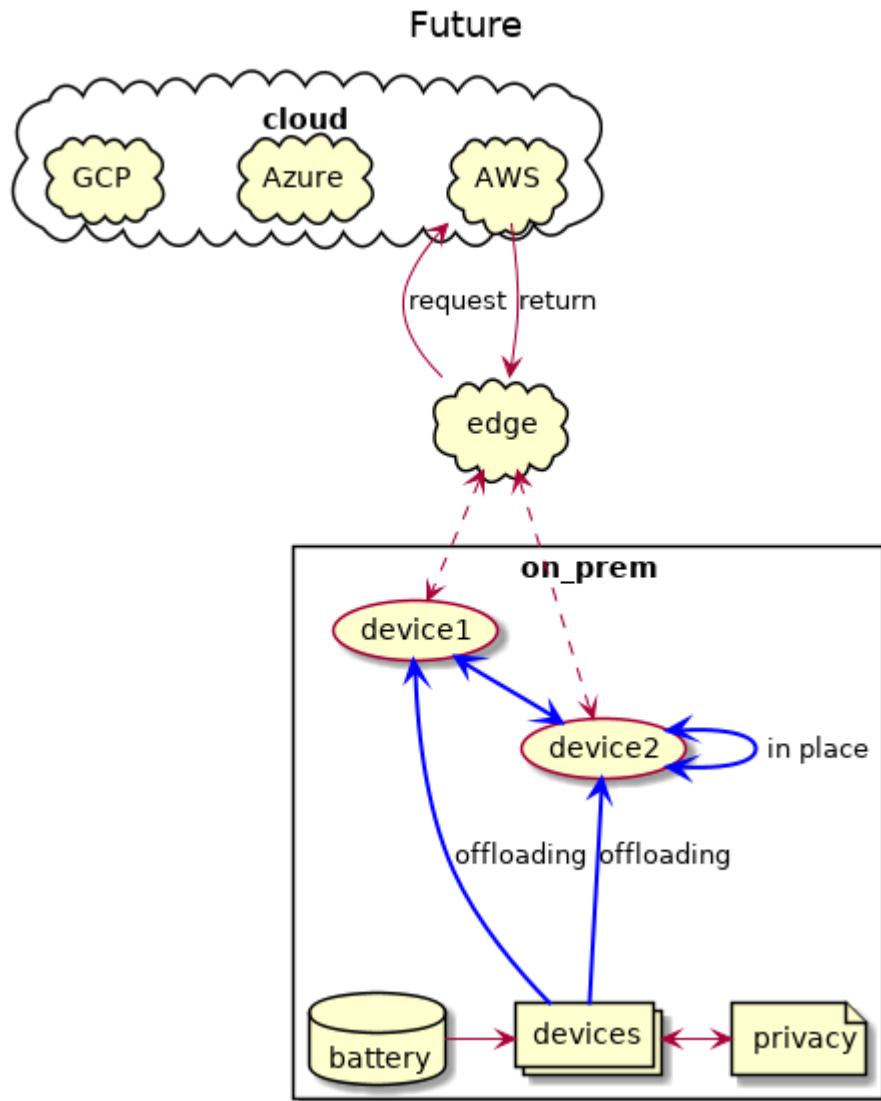


Currently



## Edge computing problems

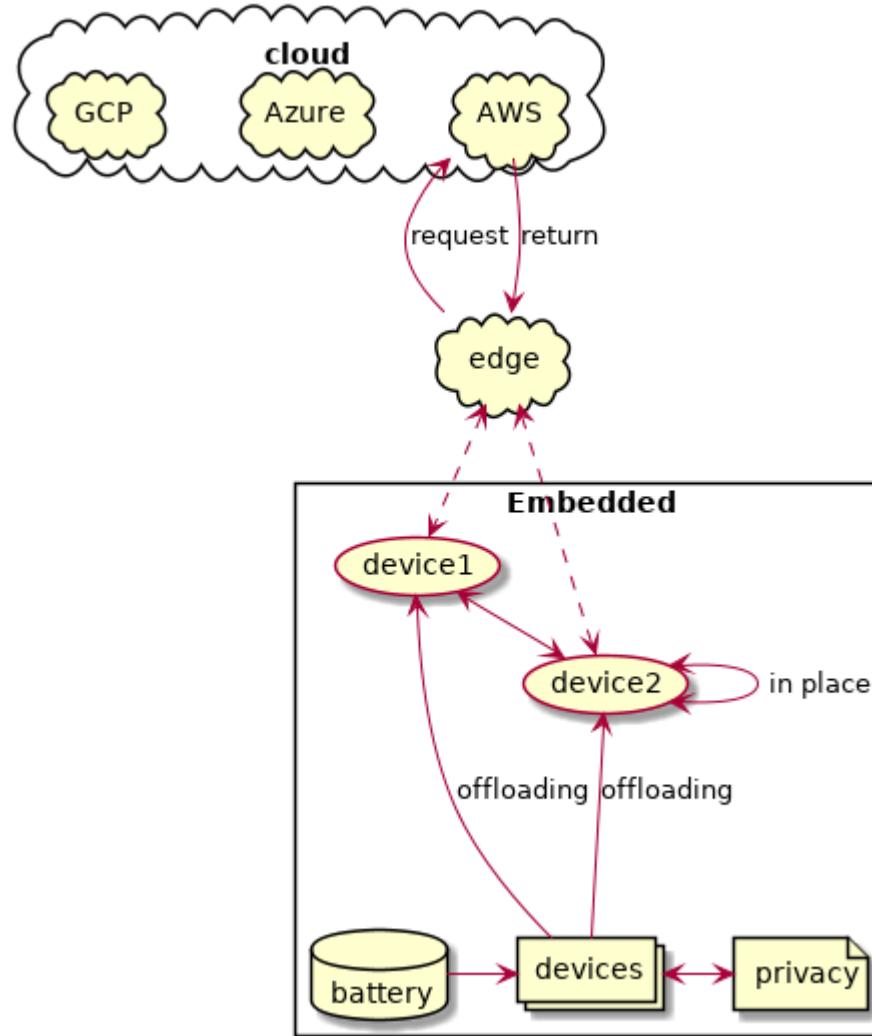




# Outline

1. ~~Demo~~
2. **Problems**
  - ~~Edge Computing~~
  - **Web vs Embedded**
  - ML environment
3. Proposal
4. Three Enablers
5. PoC
6. Conclusion

# 3 areas?

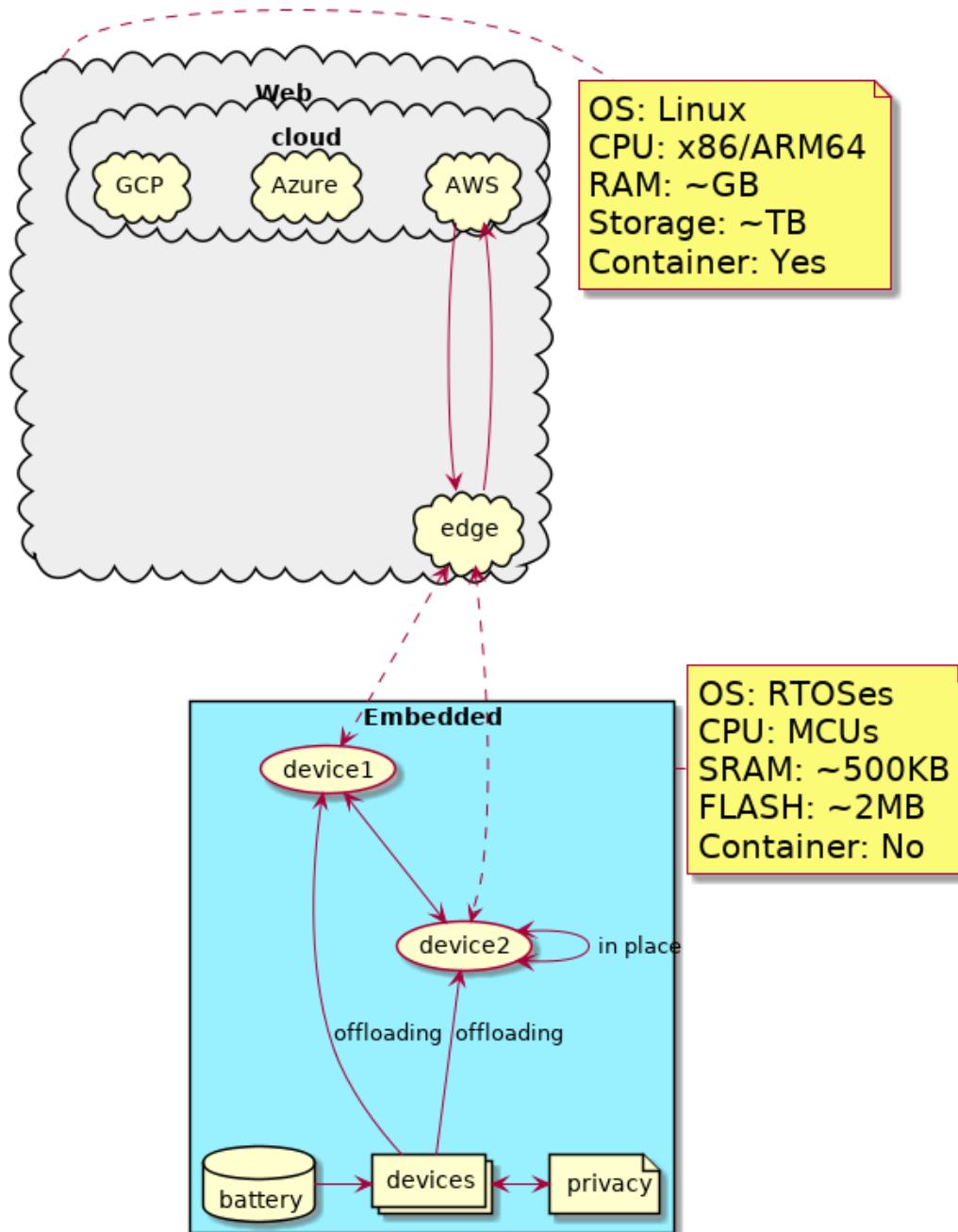


Cloud

Edge

Embedded

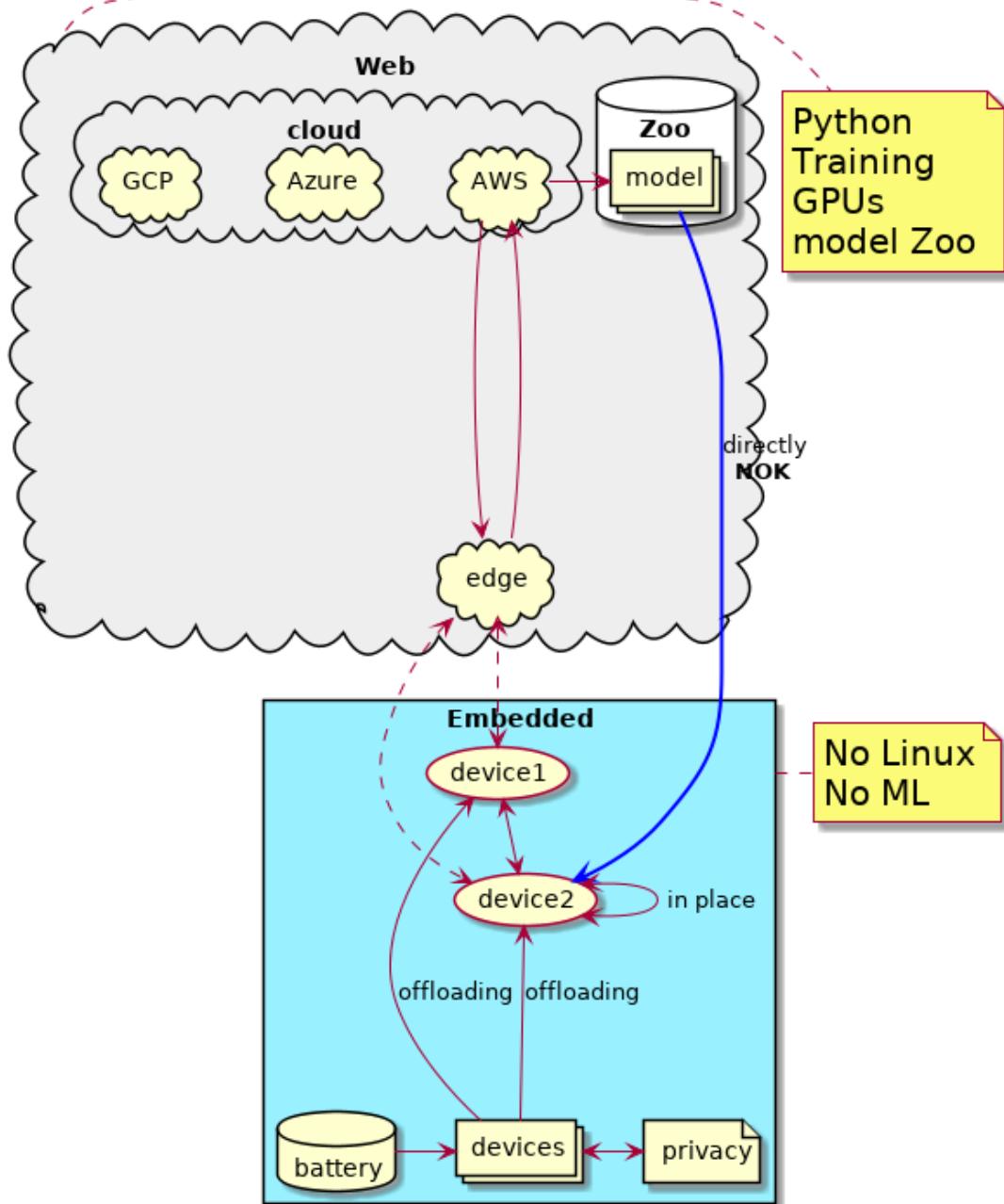
## Gap between Web and Embedded



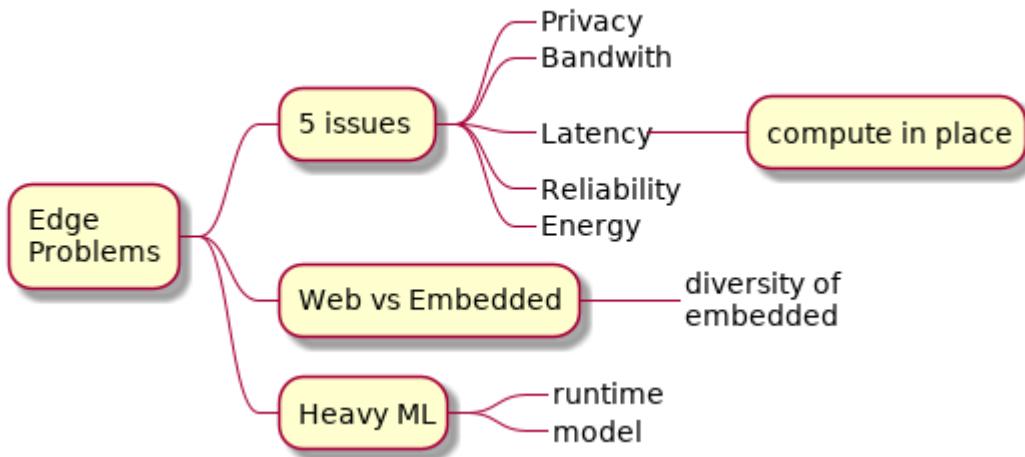
# Outline

1. ~~Demo~~
2. **Problems**
  - ~~Edge Computing~~
  - ~~Web vs Embedded~~
  - **ML environment**
3. Proposal
4. Three Enablers
5. PoC
6. Conclusion

# ML Problem

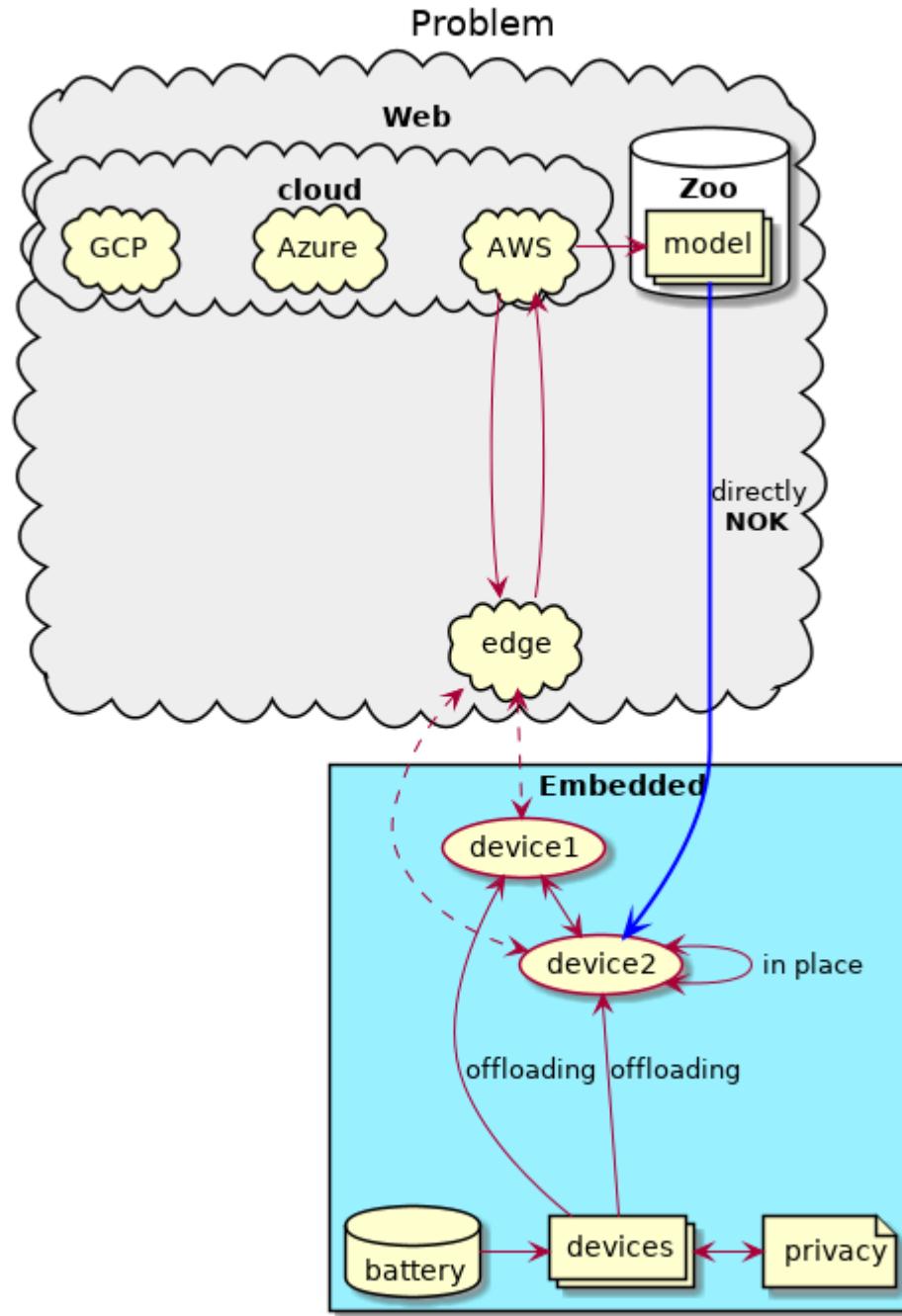


# Summary: Problems



# Outline

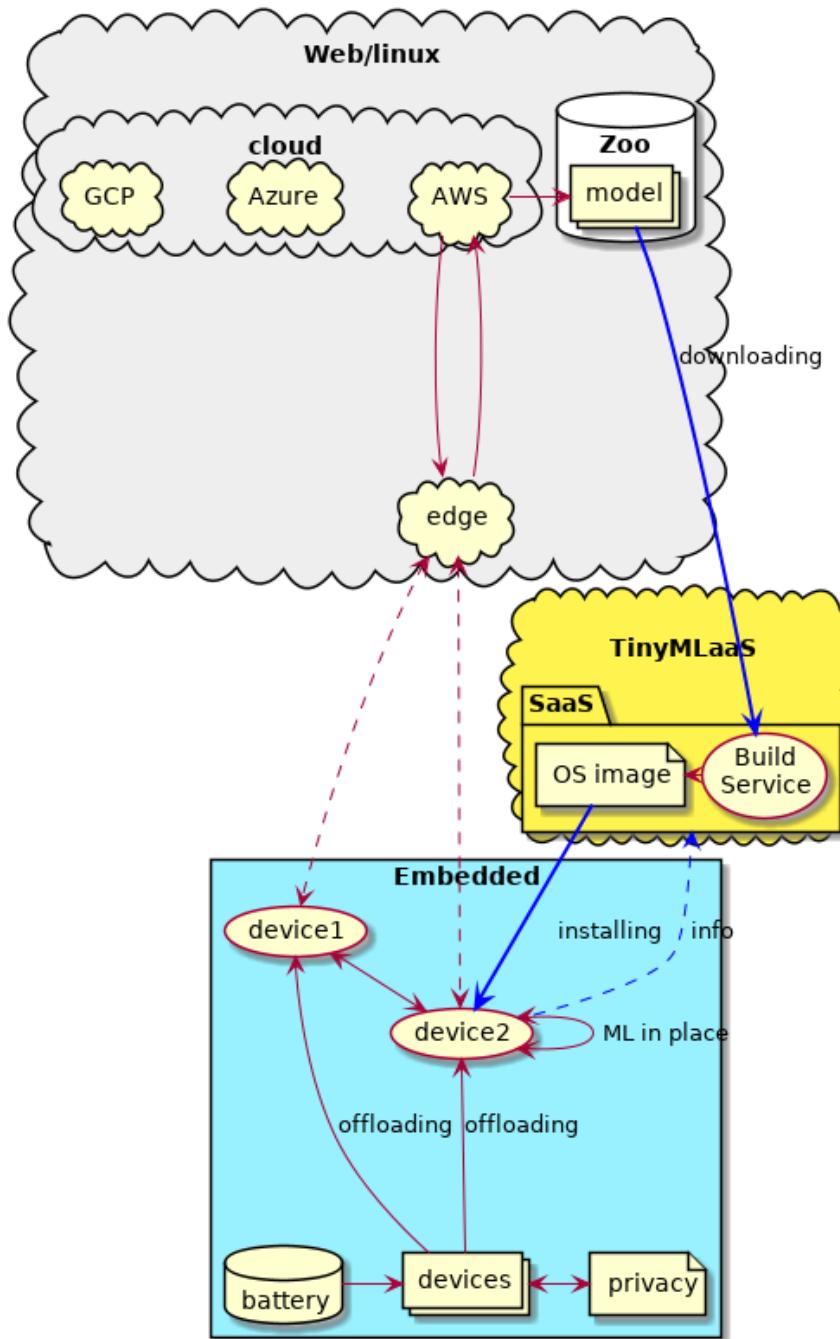
1. ~~Demo~~
2. ~~Problems~~
3. **Proposal**
4. Three Enablers
5. PoC
6. Conclusion

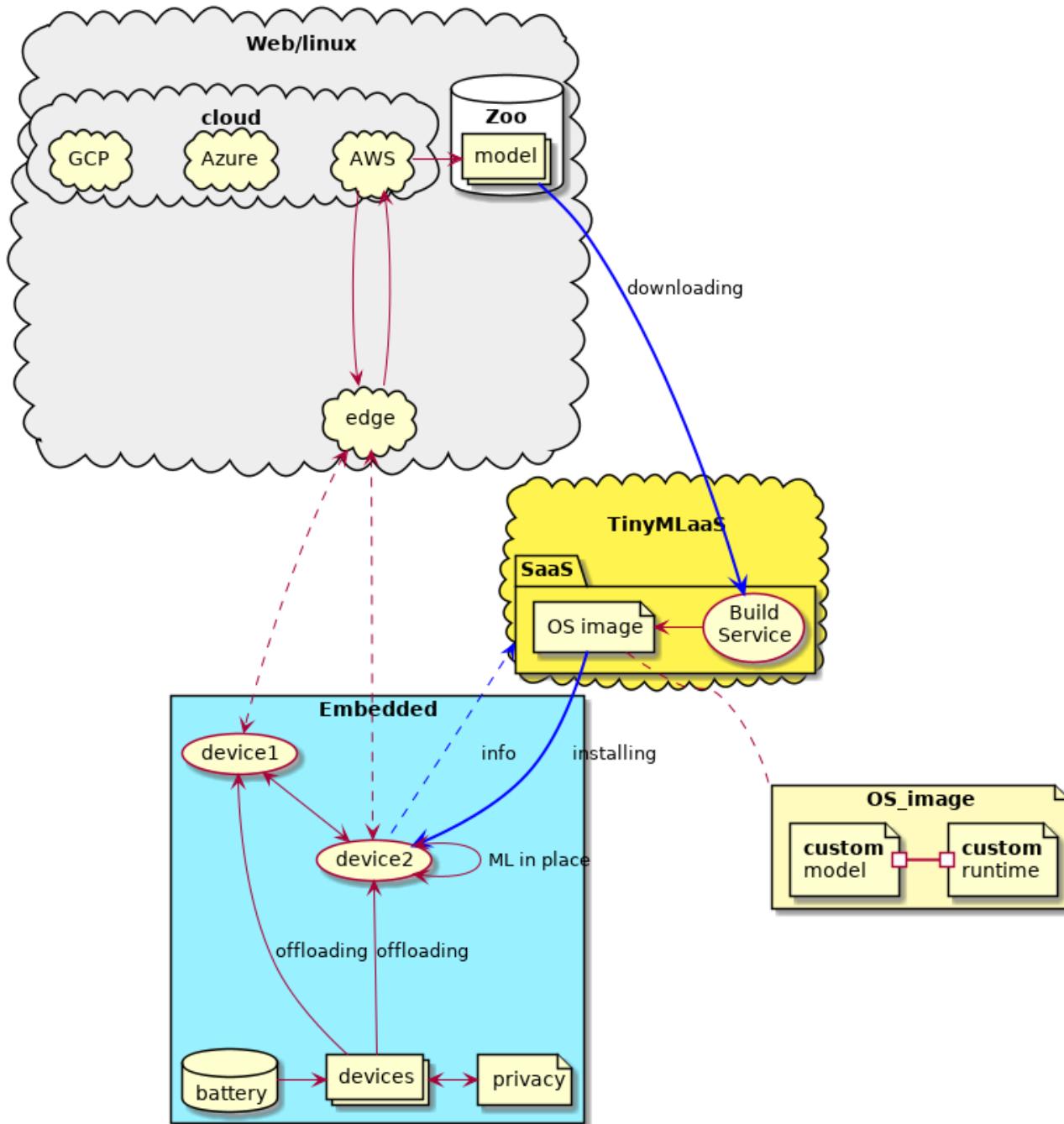


# Squeeze ML

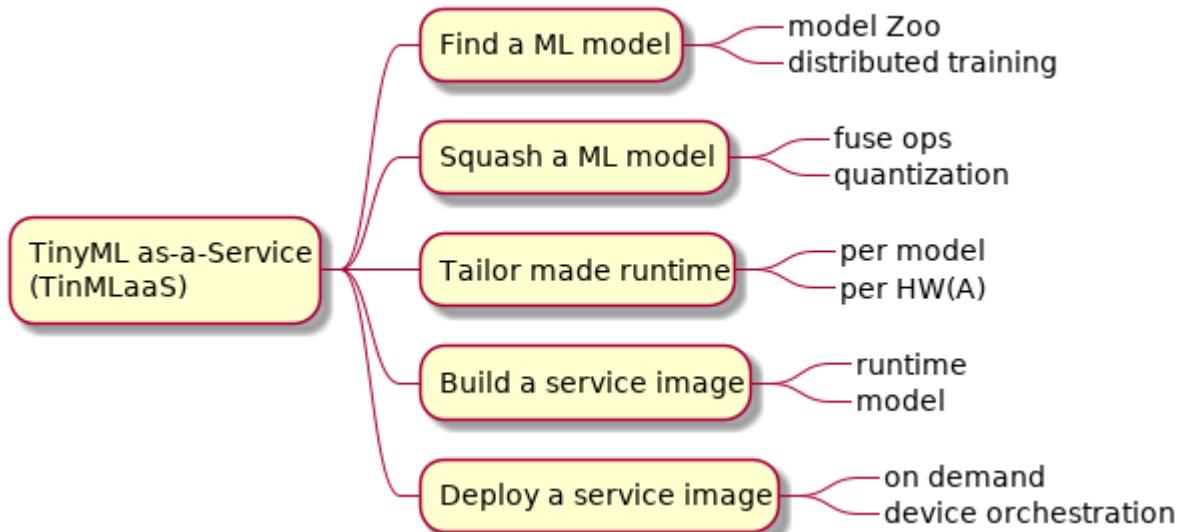
model & runtime

- per RTOS
- per HWA





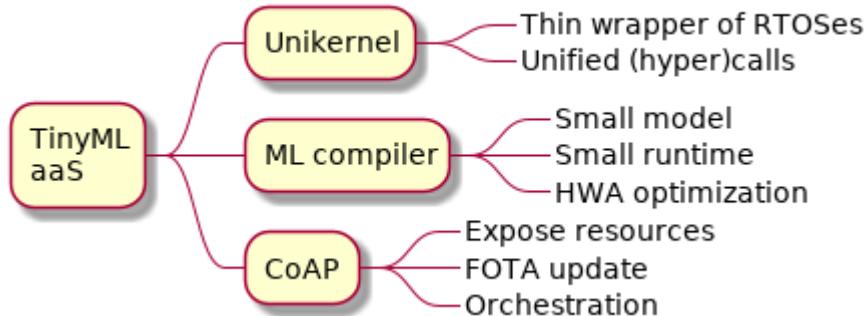
# TinyML as-a-Service



# Outline

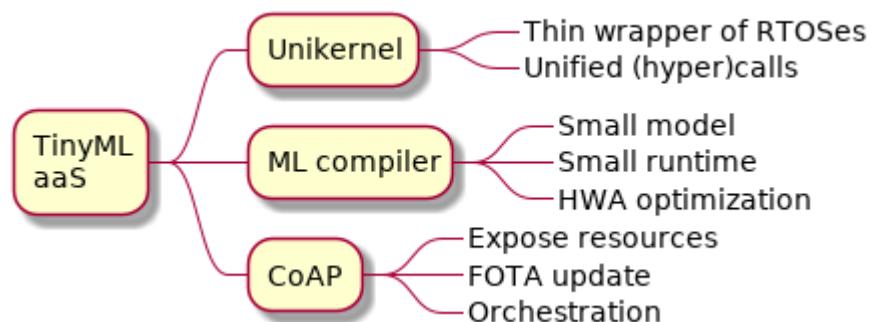
1. ~~Demo~~
2. ~~Problems~~
3. ~~Proposal~~
4. **Three Enablers**
5. PoC
6. Conclusion

# Three enablers



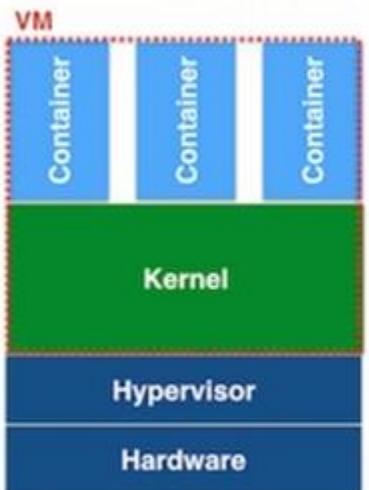
# Outline

1. ~~Demo~~
2. ~~Problems~~
3. ~~Proposal~~
4. **Three Enablers**
  - **Unikernel**
  - ML compiler
  - CoAP
5. PoC
6. Conclusion

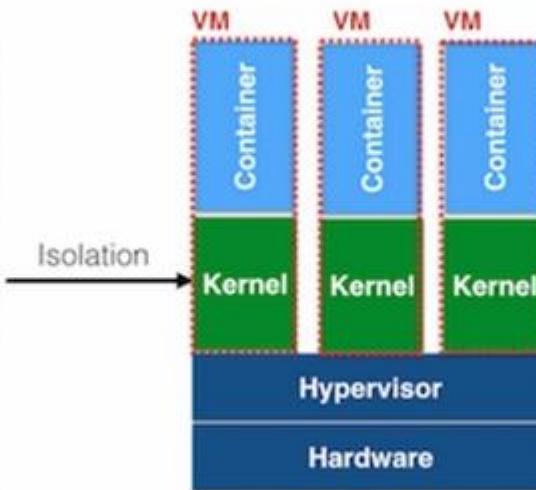


# Unikernel

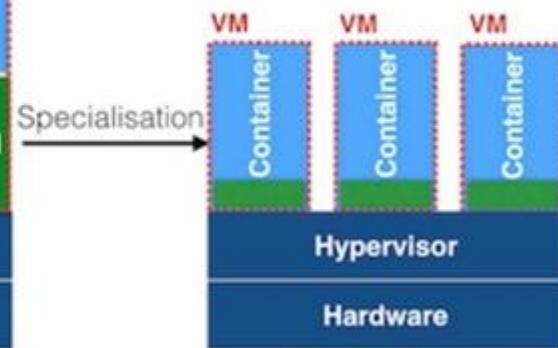
Isolation & specialisation with unikernels



**Linux Container**  
*Shared kernels*



**Container per VM**  
*Duplicated kernels*



**Unikernels**  
*Specialized kernels*

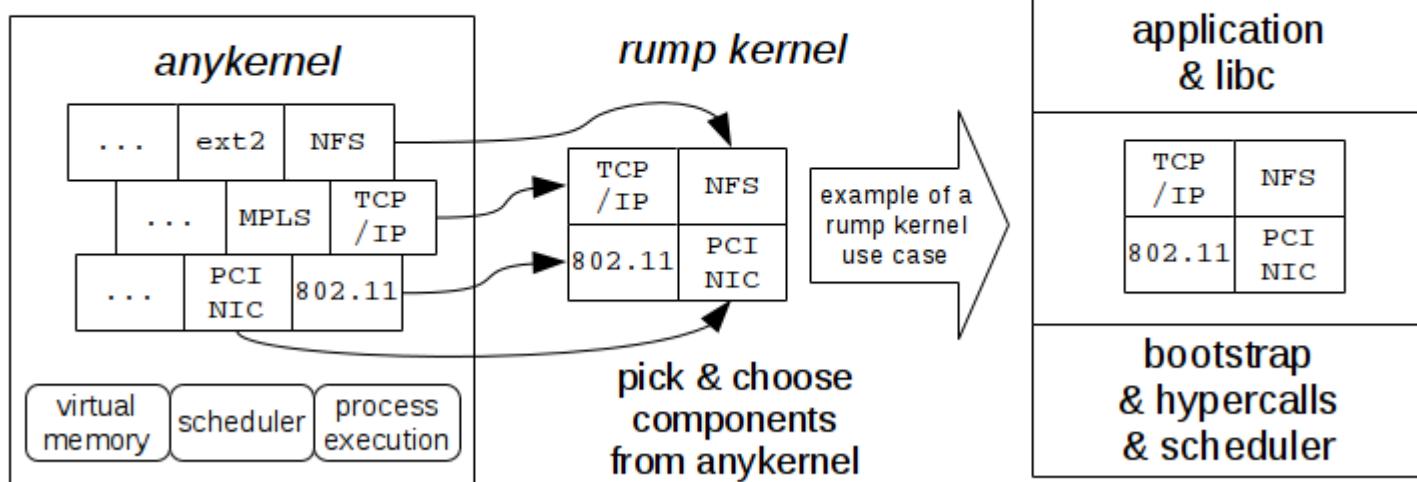
Isolation

Specialisation

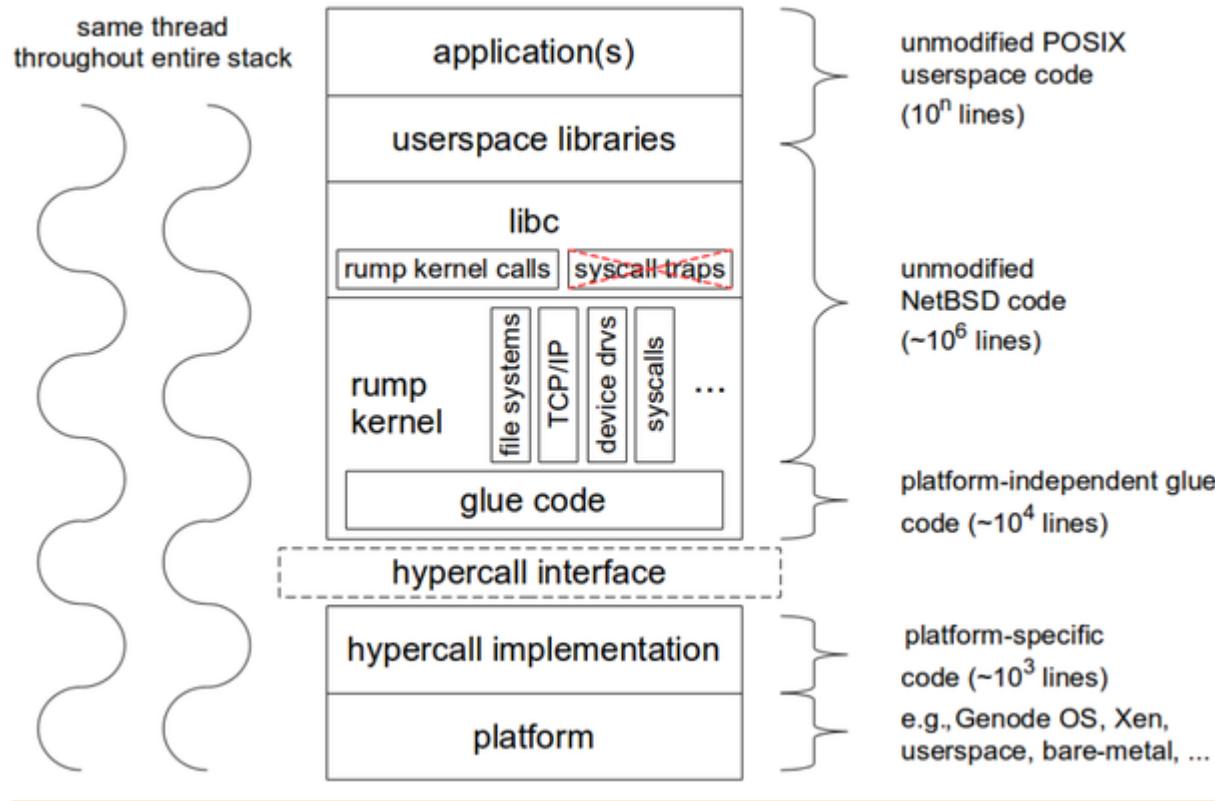
# library Operating System (libOS)

The relationship between *anykernel*,  
*rump kernel* and *unikernel*

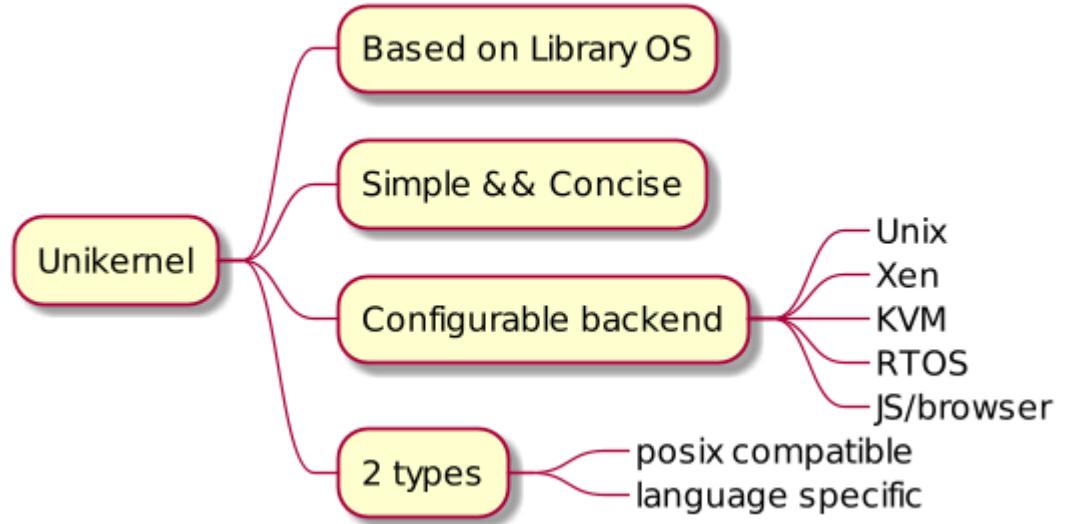
*Rumprun unikernel*



# Internal



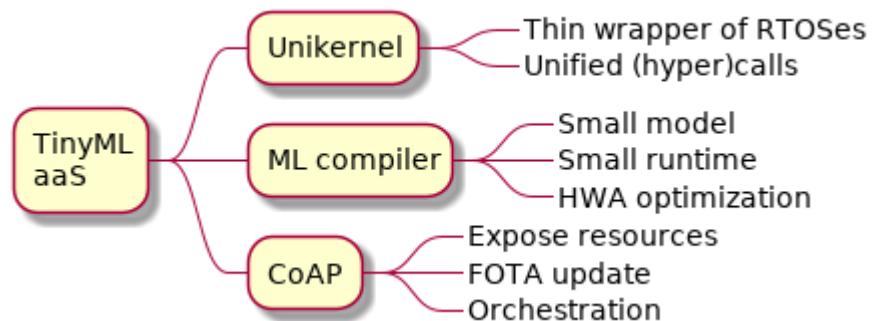
# Summary: Unikernel



Can be a thin **wrapper** of different RTOSes?

# Outline

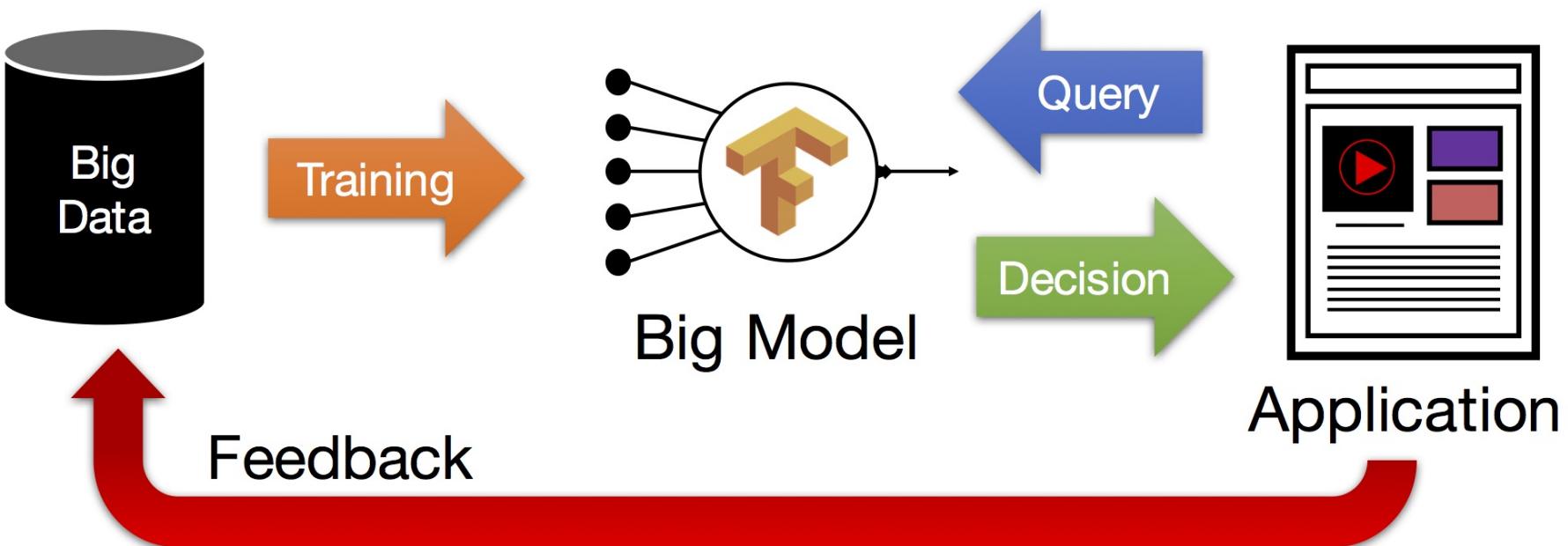
1. ~~Demo~~
2. ~~Problems~~
3. ~~Proposal~~
4. **Three Enablers**
  - **Unikernel**
  - **ML compiler**
  - CoAP
5. PoC
6. Conclusion



# 2 ML phases:

## Learning

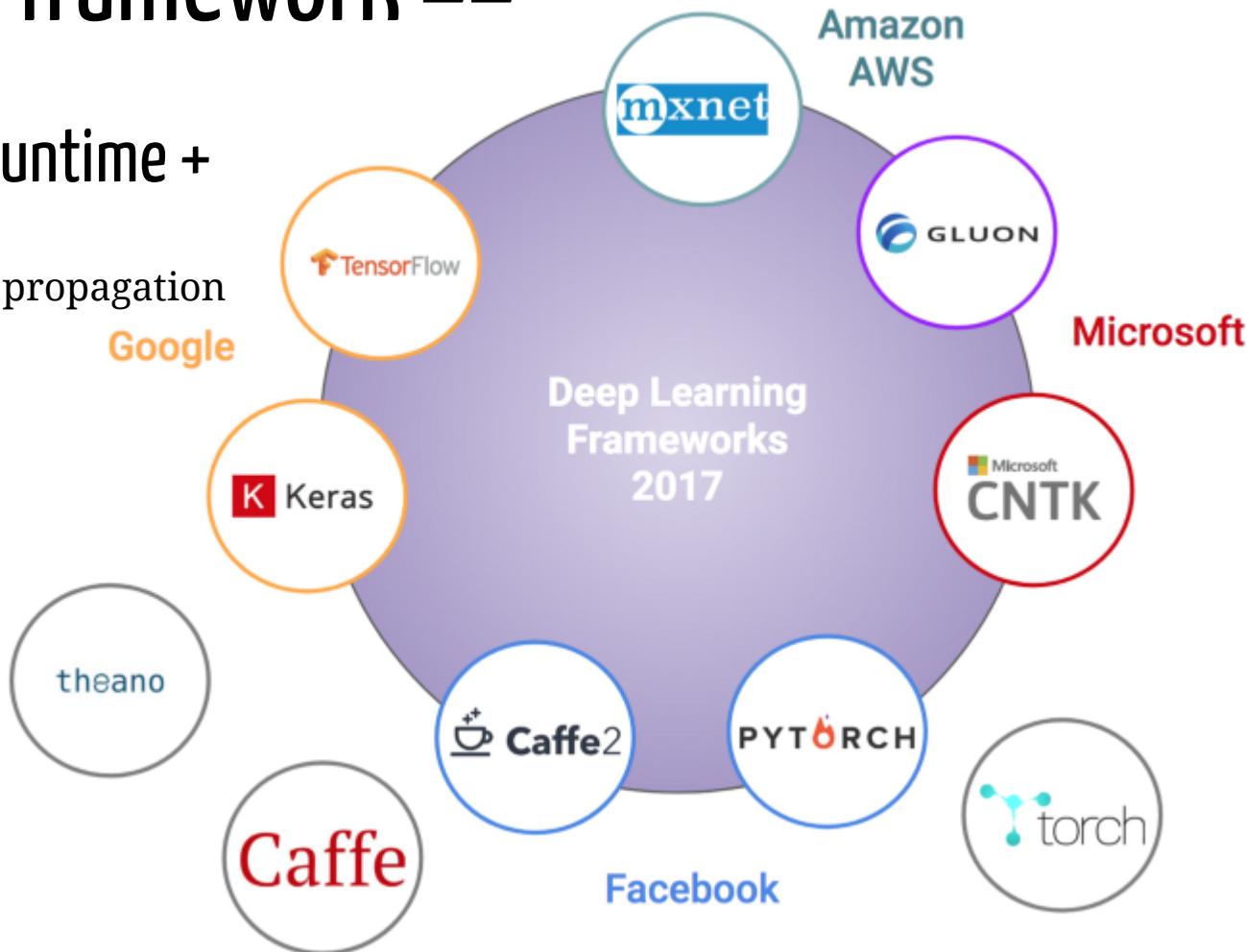
## Inference



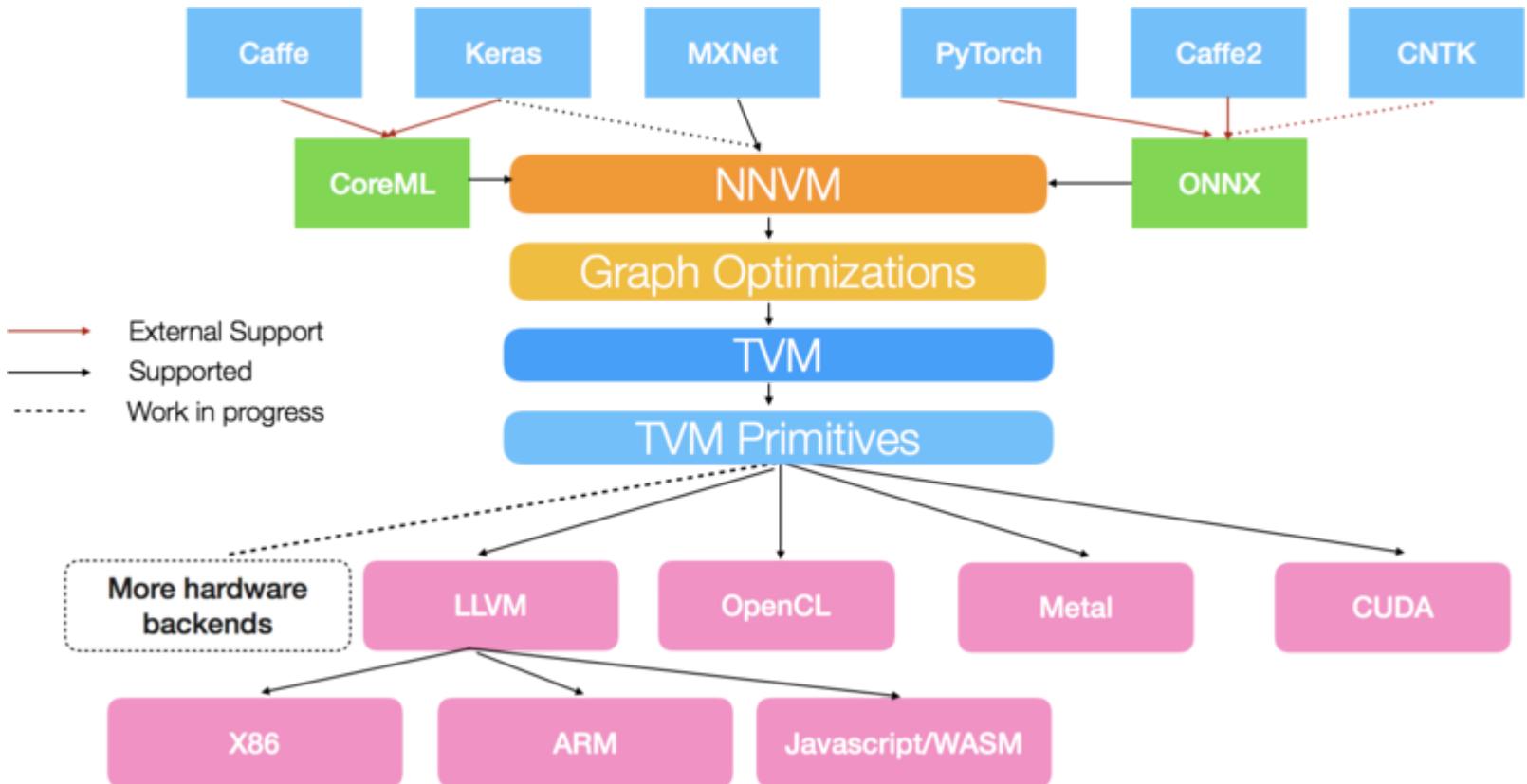
# ML framework ==

ML runtime +

back propagation

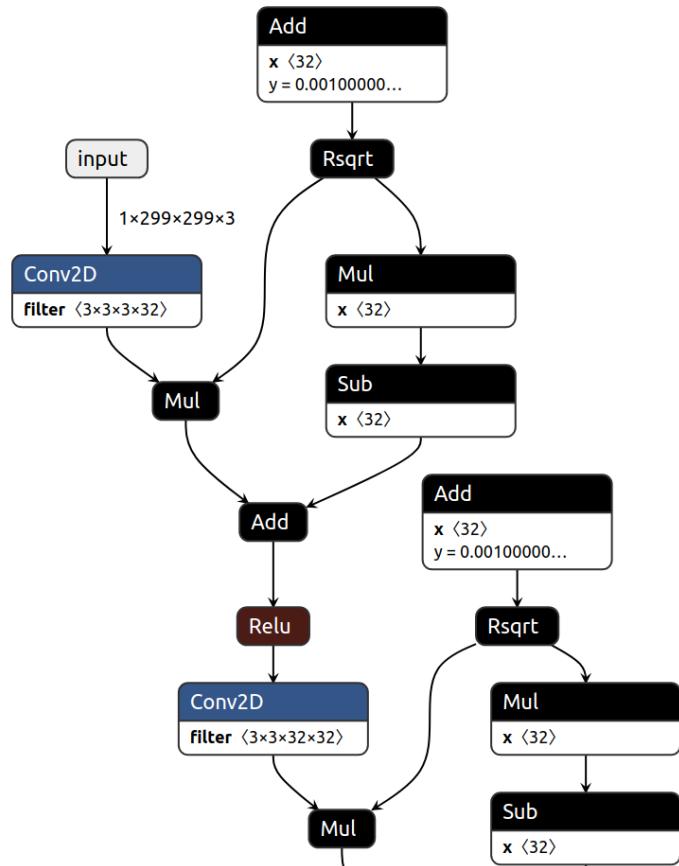


# Compiling



# Computational graph (cgraph)

[≡] [+] [-]



## NODE PROPERTIES

type Conv2D

name InceptionV3/InceptionV3/Conv2d\_2a\_3x3/convolution

## ATTRIBUTES

data\_format NHWC

padding VALID

strides 1, 1, 1, 1

T Float32

use\_cudnn\_on\_g... true

## INPUTS

input id: InceptionV3/InceptionV3/Conv2d\_1a\_3x3/Relu

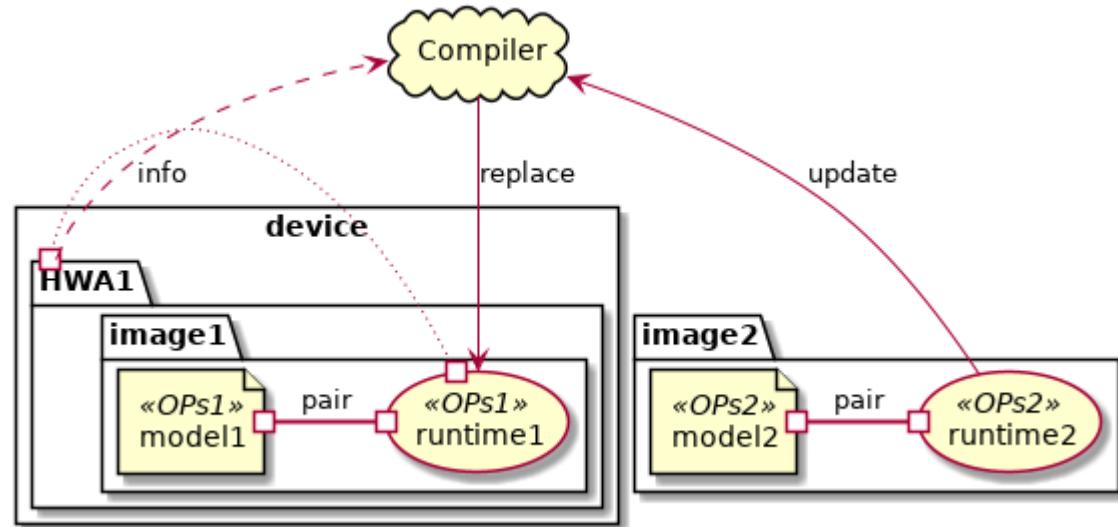
filter id: InceptionV3/Conv2d\_2a\_3x3/weights/read

## OUTPUTS

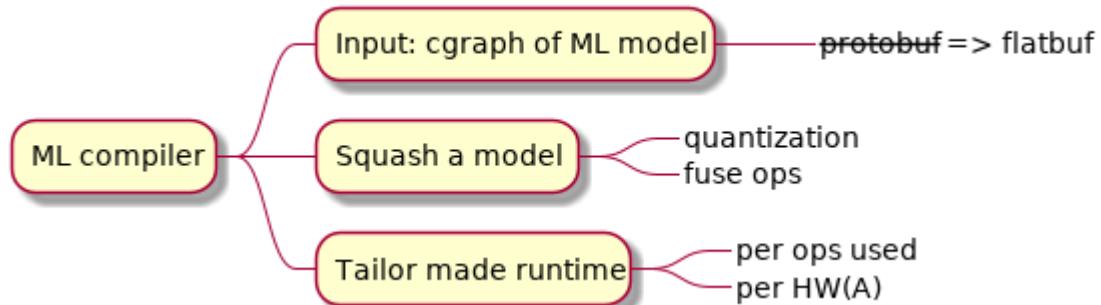
output id: InceptionV3/InceptionV3/Conv2d\_2a\_3x3/convolution

# General vs Special purpose runtime

- General purpose is too big
  - all OPs built-in
- Model could be optimized,
  - per HWAs' OPs,
  - smaller than CPU's OPs(?)
- Runtime should implement **only** OPs,
  - which model uses.

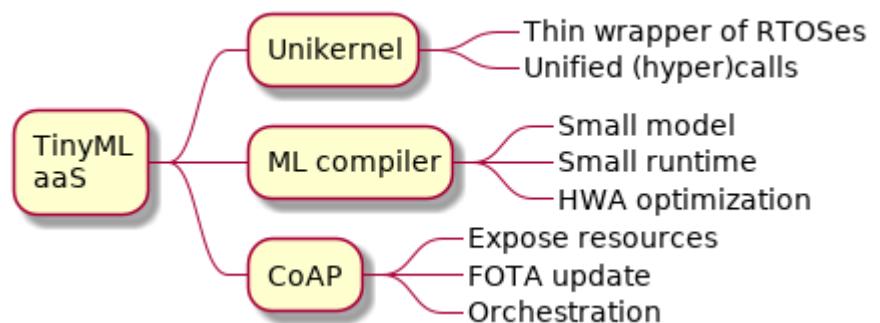


# Summary: ML compiler



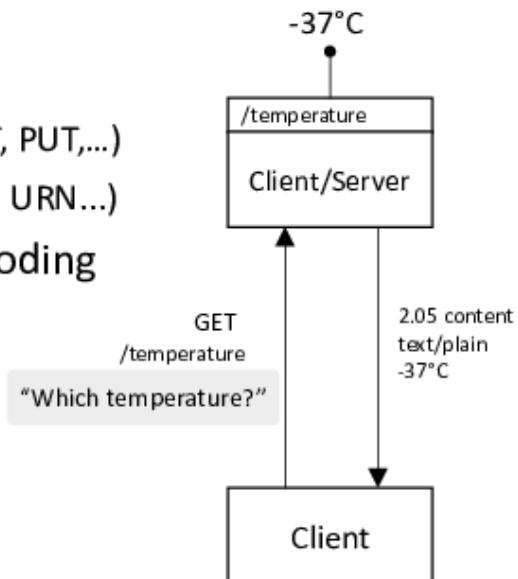
# Outline

1. ~~Demo~~
2. ~~Problems~~
3. ~~Proposal~~
4. **Three Enablers**
  - o ~~Unikernel~~
  - o ~~ML compiler~~
  - o ~~CoAP~~
5. PoC
6. Conclusion

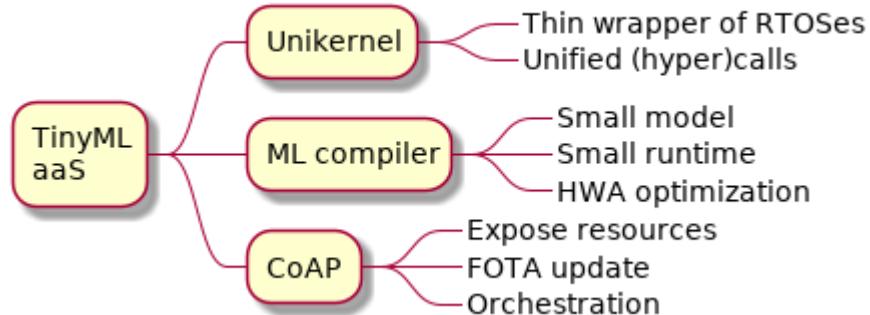


# The Constrained Application Protocol (CoAP)

- CoAP (RFC7252) implements HTTP's **REST** model
  - Simple devices: 100 to 250 KiB code and 10 to 50 KiB RAM
  - Each device can be client and server exposing resources
  - CoAP defines methods to access those resources (GET, POST, PUT,...)
  - Same key concepts borrowed from HTTP (Media types, URL, URN...)
- Has a compact 4-byte header, with simple options encoding
- Simple protocol, datagram (UDP, DTLS)
  - Reliability through header message type "*CON/NON*"
  - With TCP/TLS (RFC8323) support for NAT-ed environments
- The Resource Directory provides a directory service



# Summary: Three Enablers



# Outline

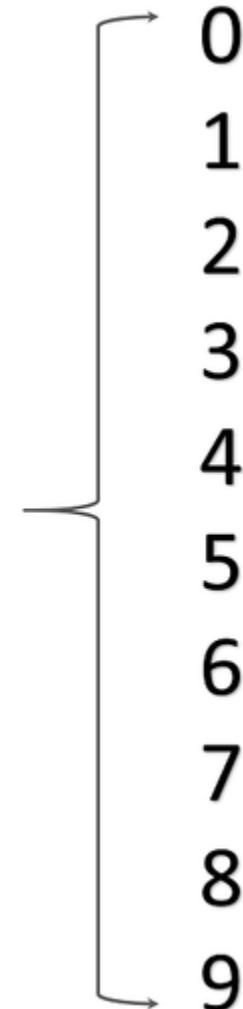
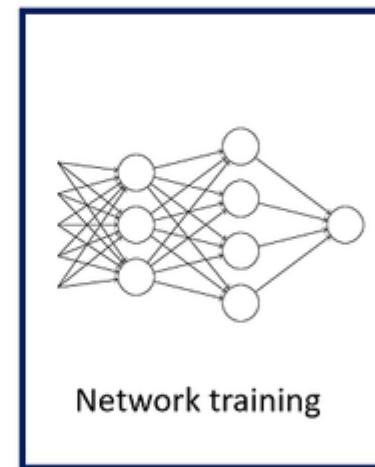
1. ~~Demo~~
2. ~~Problems~~
3. ~~Proposal~~
4. ~~Three Enablers~~
5. PoC
6. Conclusion

# MNIST: Handwriting digits recognition

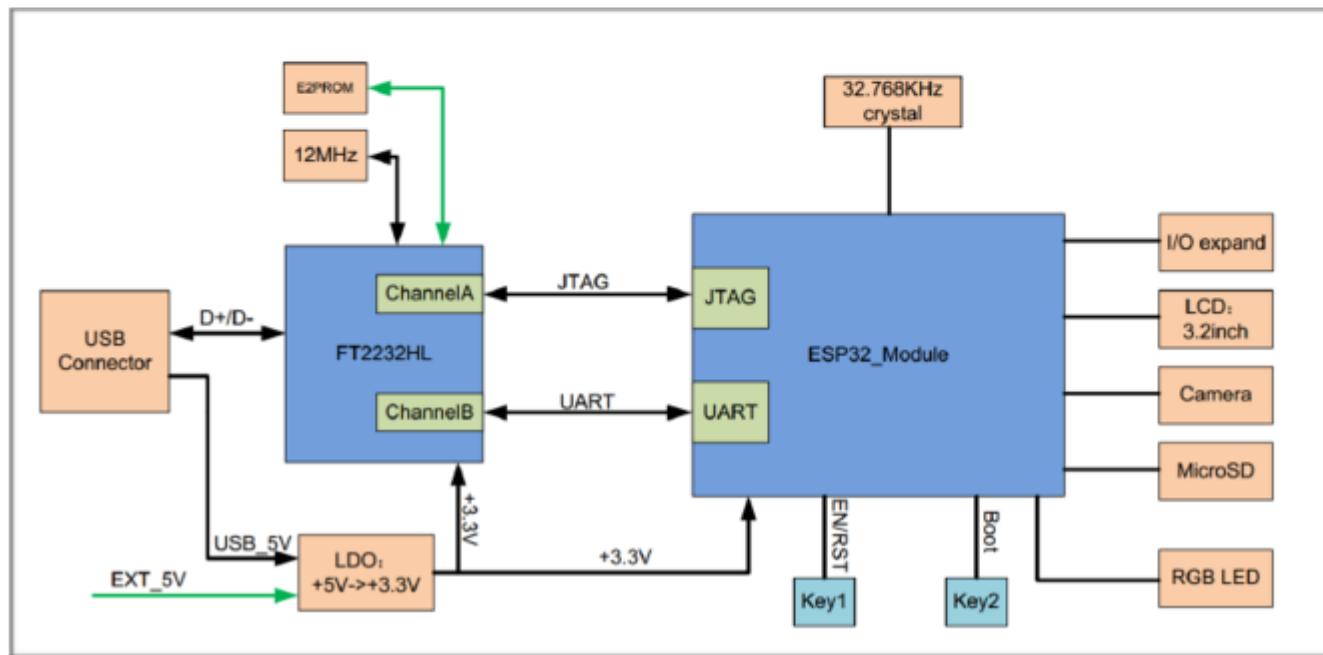
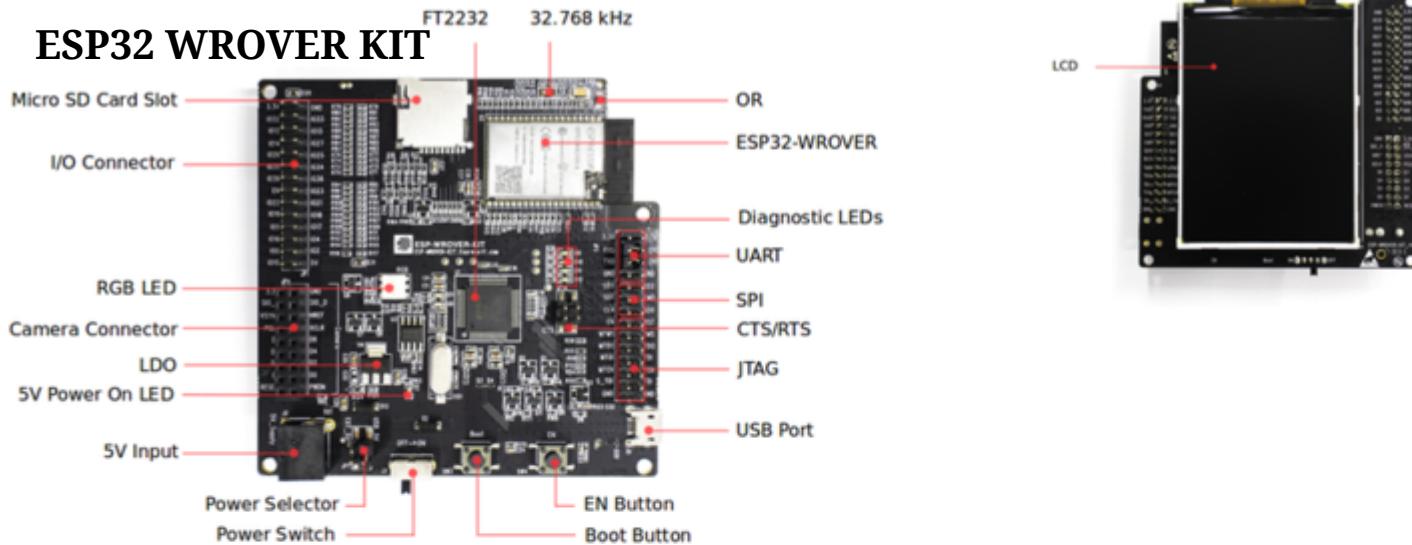
60K images for training, 10K for testing

0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9

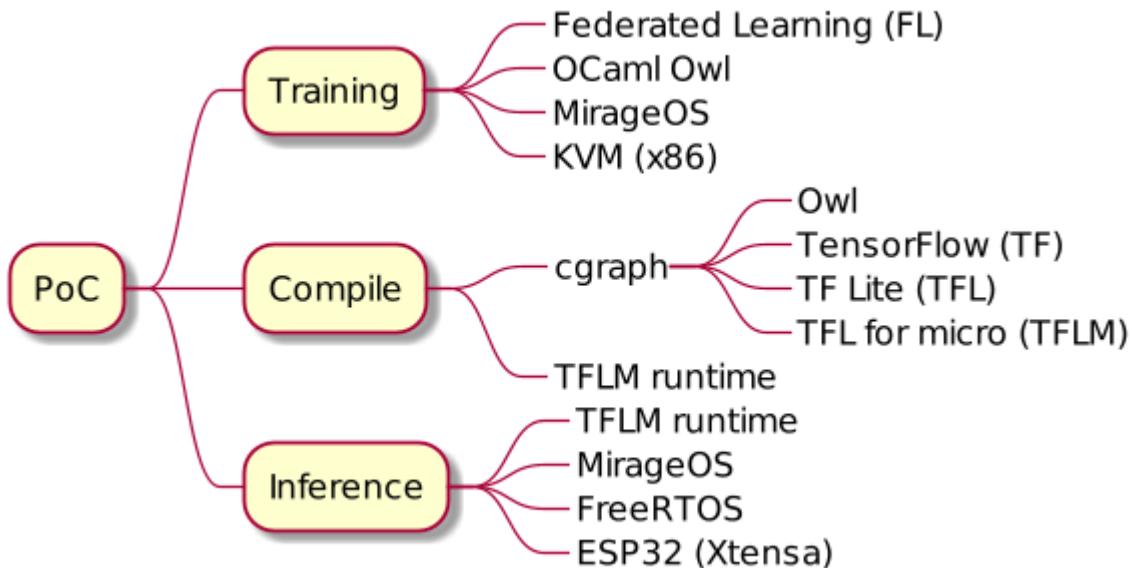
Data & Labels



# ESP32 WROVER KIT

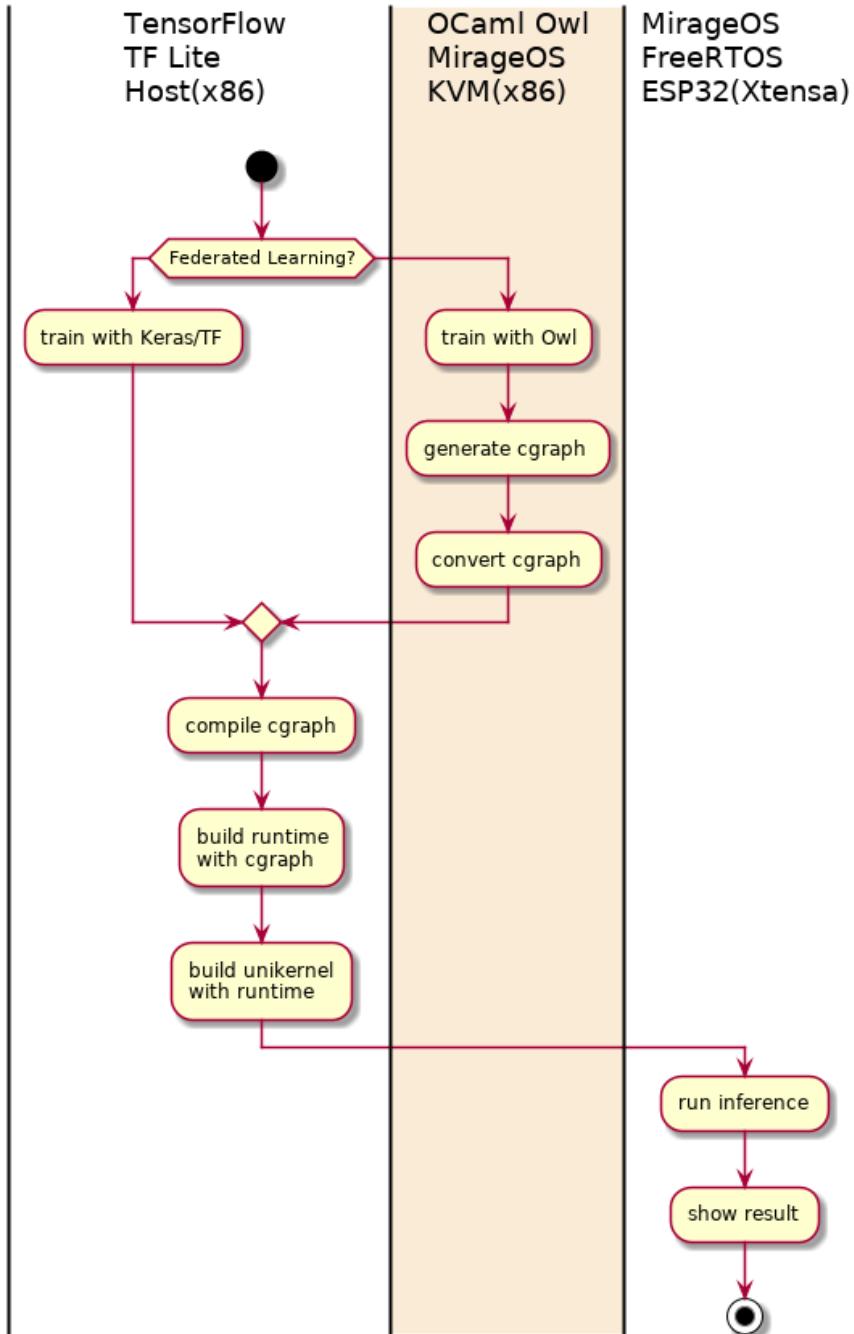


# PoC Outline



# PoC

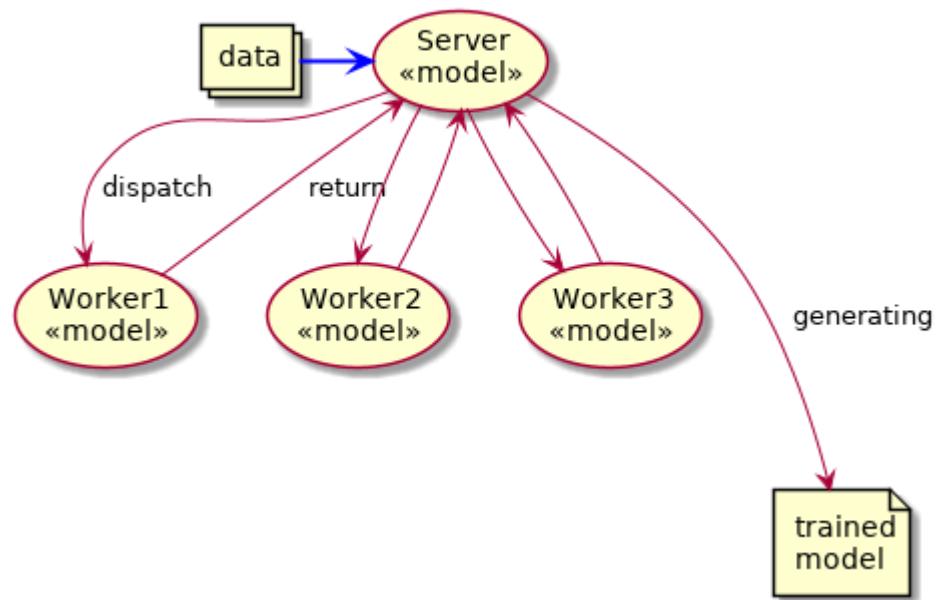
## Sequence



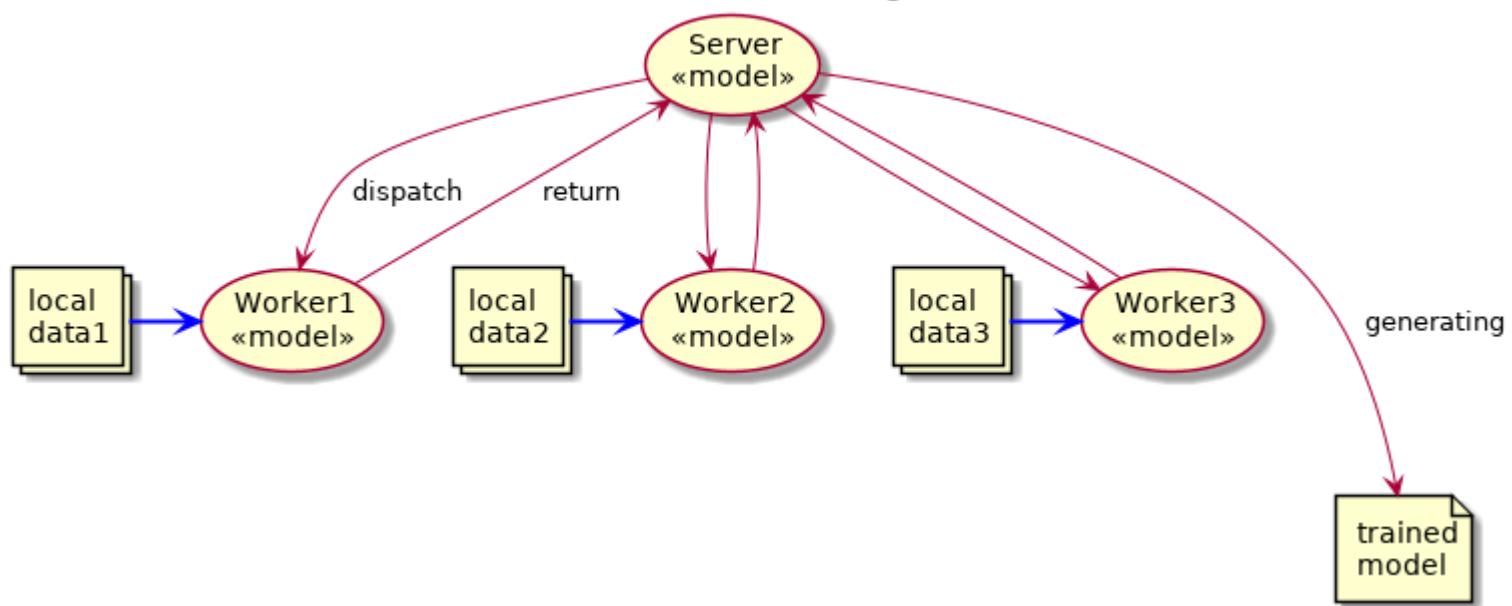
# Outline

1. ~~Demo~~
2. ~~Problems~~
3. ~~Proposal~~
4. ~~Three Enablers~~
5. PoC
  - Training
  - Compile
  - Inference
6. Conclusion

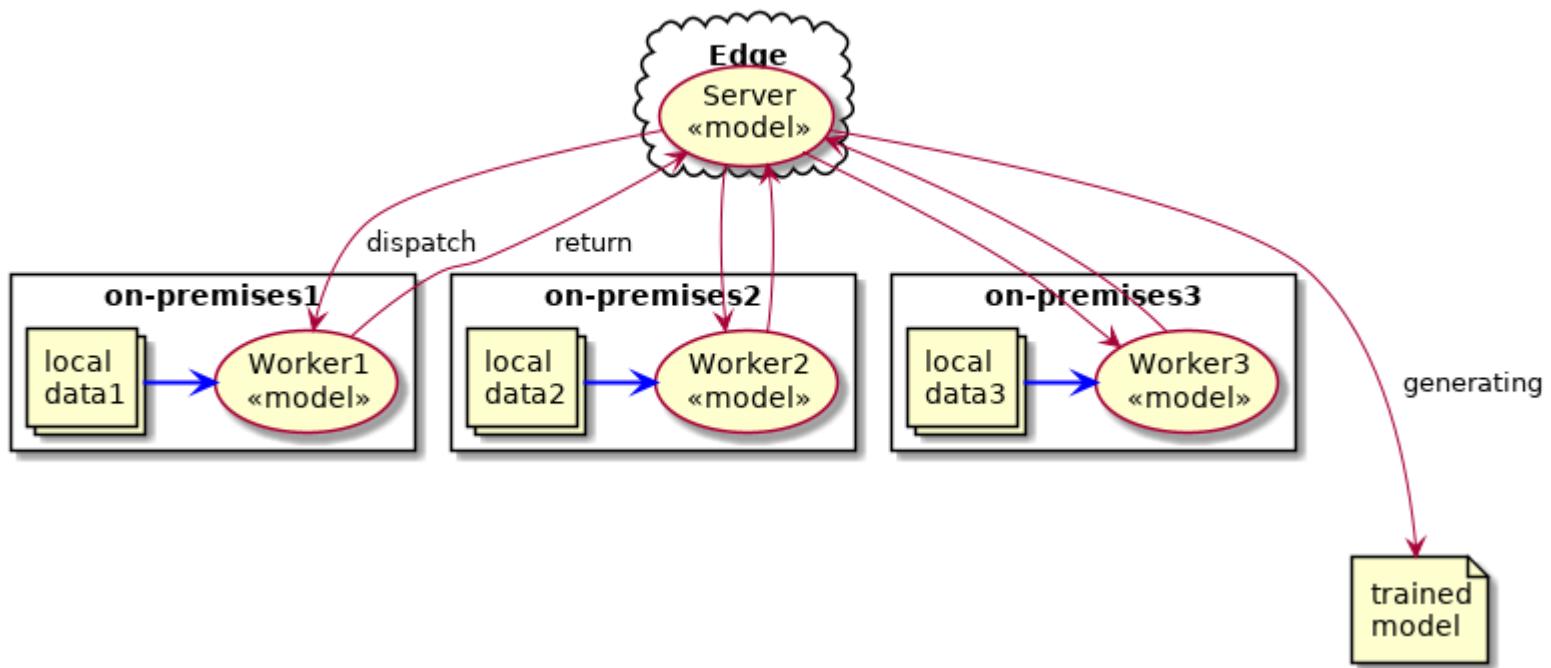
## Training: Parameter Server (PS)



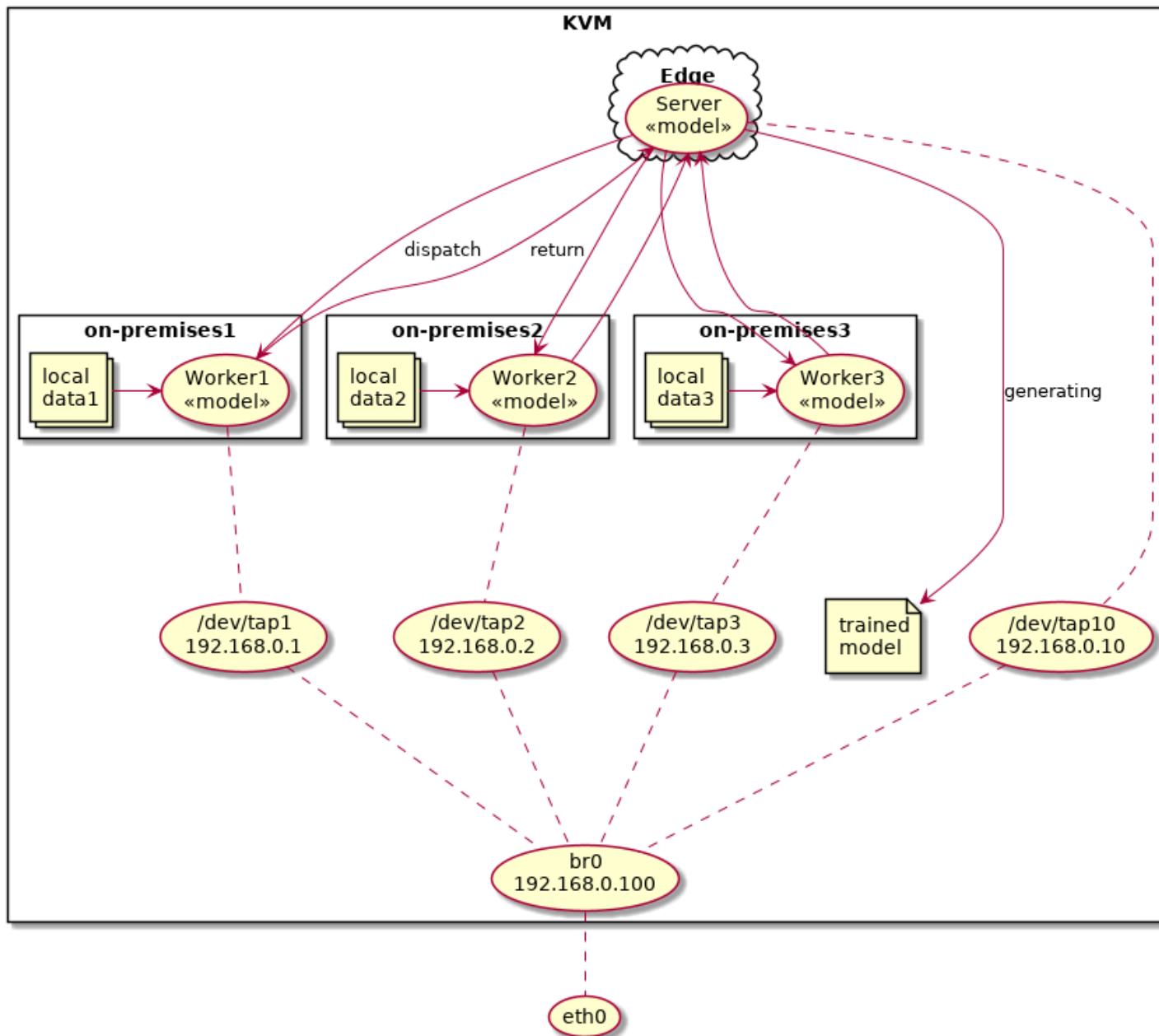
## Training: Federated Learning (FL)



## Training: Federated Learning (FL)



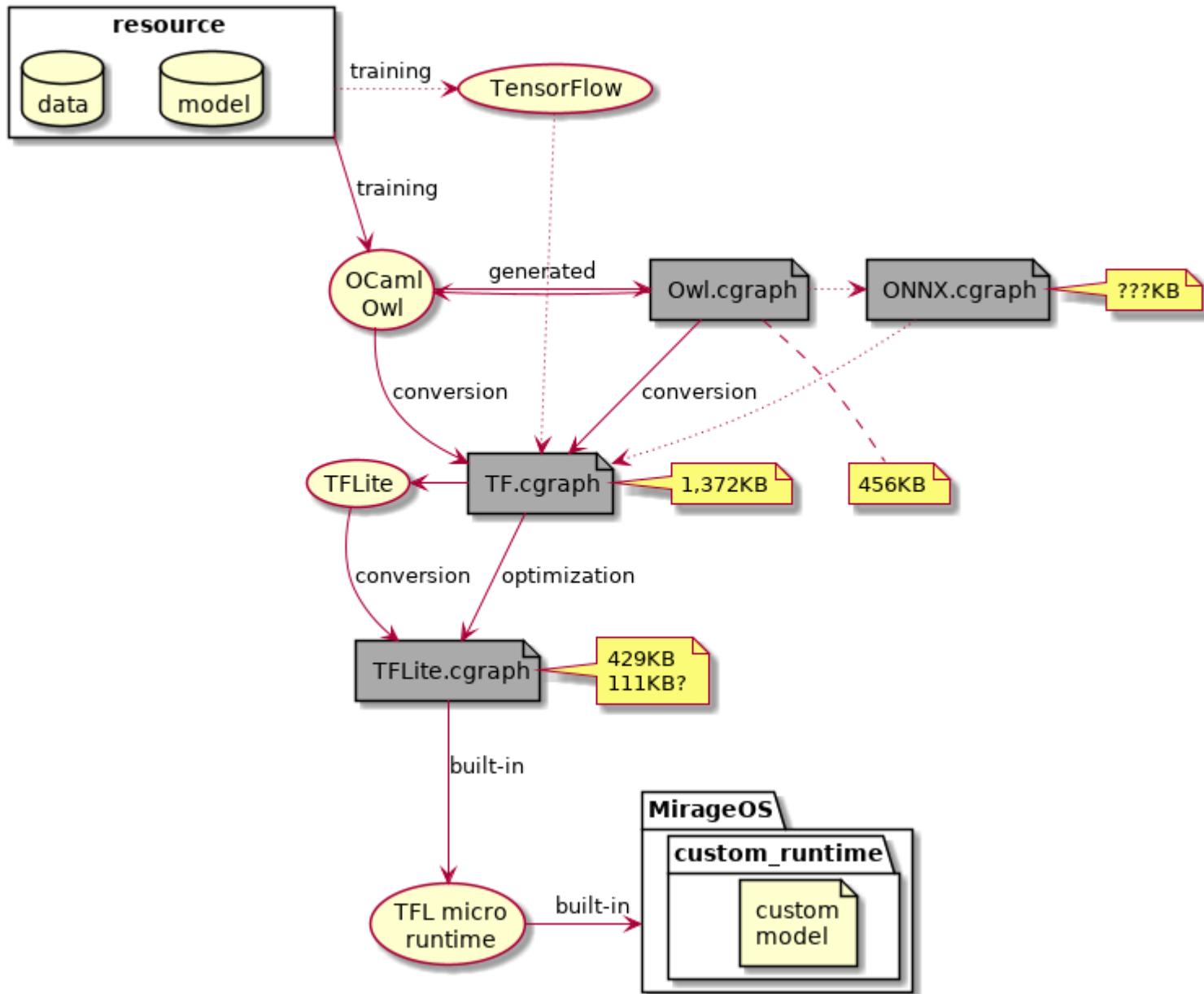
Training:  
FL on KVM



# Outline

1. ~~Demo~~
2. ~~Problems~~
3. ~~Proposal~~
4. ~~Three Enablers~~
5. PoC
  - ~~Training~~
  - ~~Compile~~
  - Inference
6. Conclusion

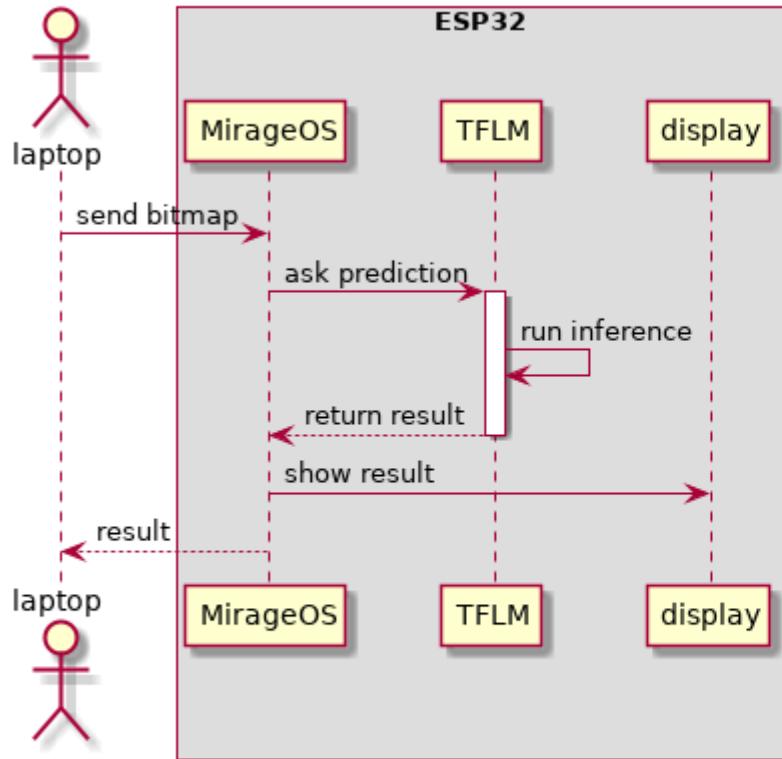
## ML compiler: computation graph conversion



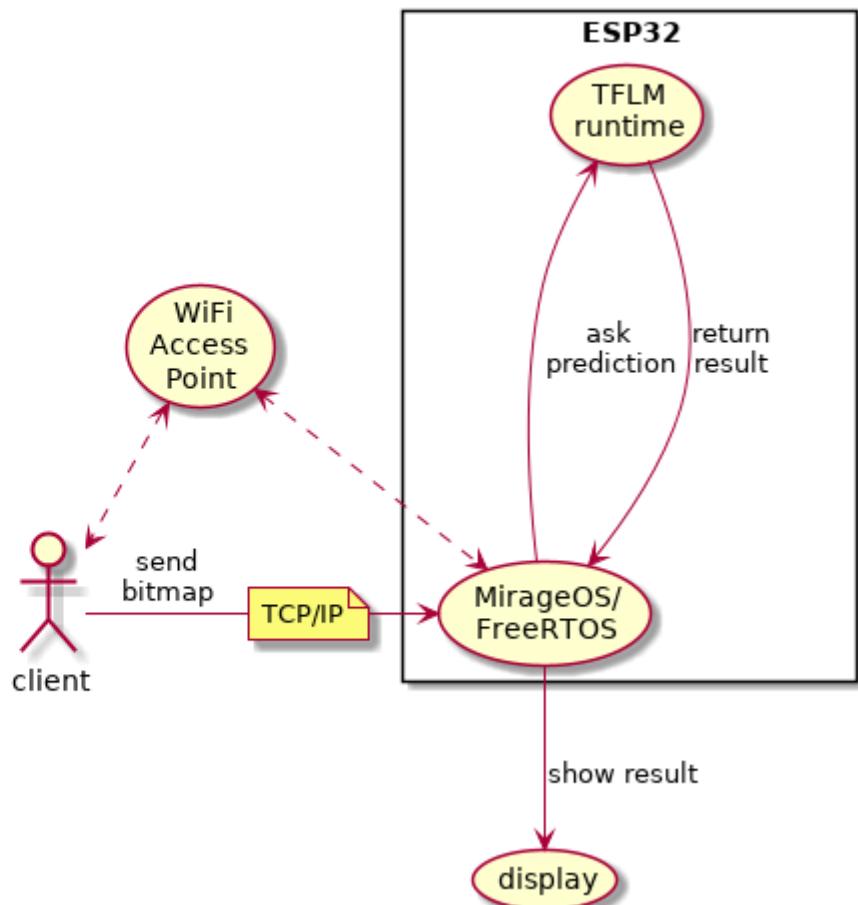
# Outline

1. ~~Demo~~
2. ~~Problems~~
3. ~~Proposal~~
4. ~~Three Enablers~~
5. PoC
  - o ~~Training~~
  - o ~~Compile~~
  - o ~~Inference~~
6. Conclusion

### Inference Sequence



### Inference Usecase



# Binary Size

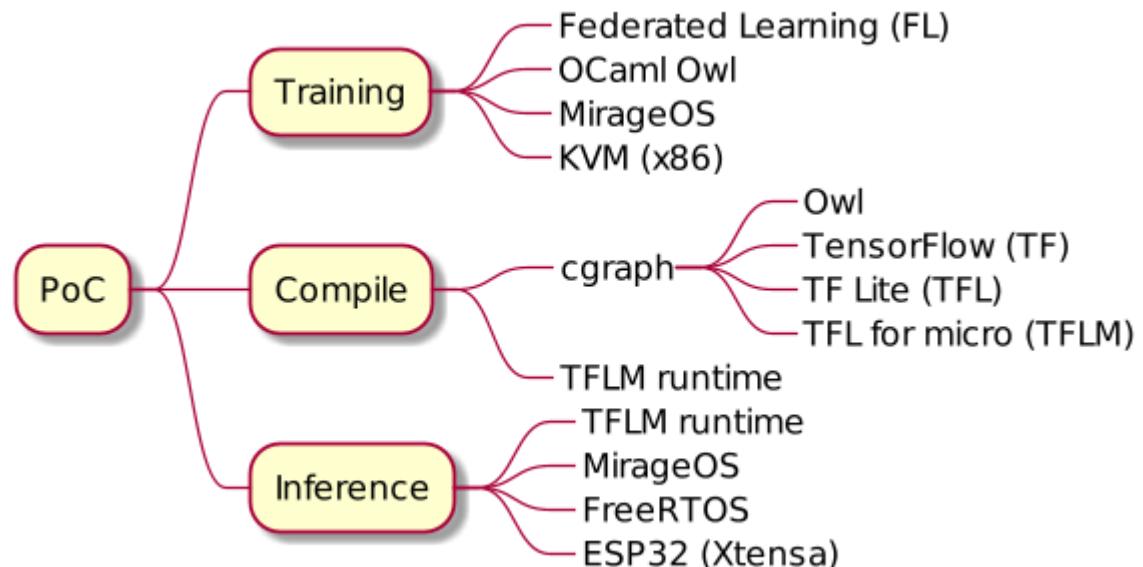
```
Total sizes: [35/93]
DRAM .data size: 18784 bytes
DRAM .bss size: 62936 bytes
Used static DRAM: 81720 bytes ( 99016 available, 45.2% used)
Used static IRAM: 91189 bytes ( 39883 available, 69.6% used)
    Flash code: 1062913 bytes
    Flash rodata: 854548 bytes
Total image size: ~2027434 bytes (.bin may be padded larger)
Per-archive contributions to ELF file:
          Archive File DRAM .data & .bss      IRAM Flash code & rodata   Total
            libmain.a      3410    40192        0     448568     300296   792466
            libtflm.a      1216        0        0     151465     447123   599804
            libnet80211.a    314     9056     3565    107308     12083   132326
  libc-psram-workaround.a    495       24     8070     83797      8017   100403
            liblwip.a       14     1962        0     61917     12227    76120
            libesp32.a      3105    2637    18693     21990     28550    74975
            libpp.a        1229    5274    13275     45188      4280    69246
<:- 3:- 4:-- 5:-*>▶64kB/s 25C 0.92 8x1.8GHz 31.3G18% 131.160.51.146 2019-09-
```

# Outline

1. ~~Demo~~
2. ~~Problems~~
3. ~~Proposal~~
4. ~~Three Enablers~~
5. ~~PoC~~
6. **Conclusion**

# Done

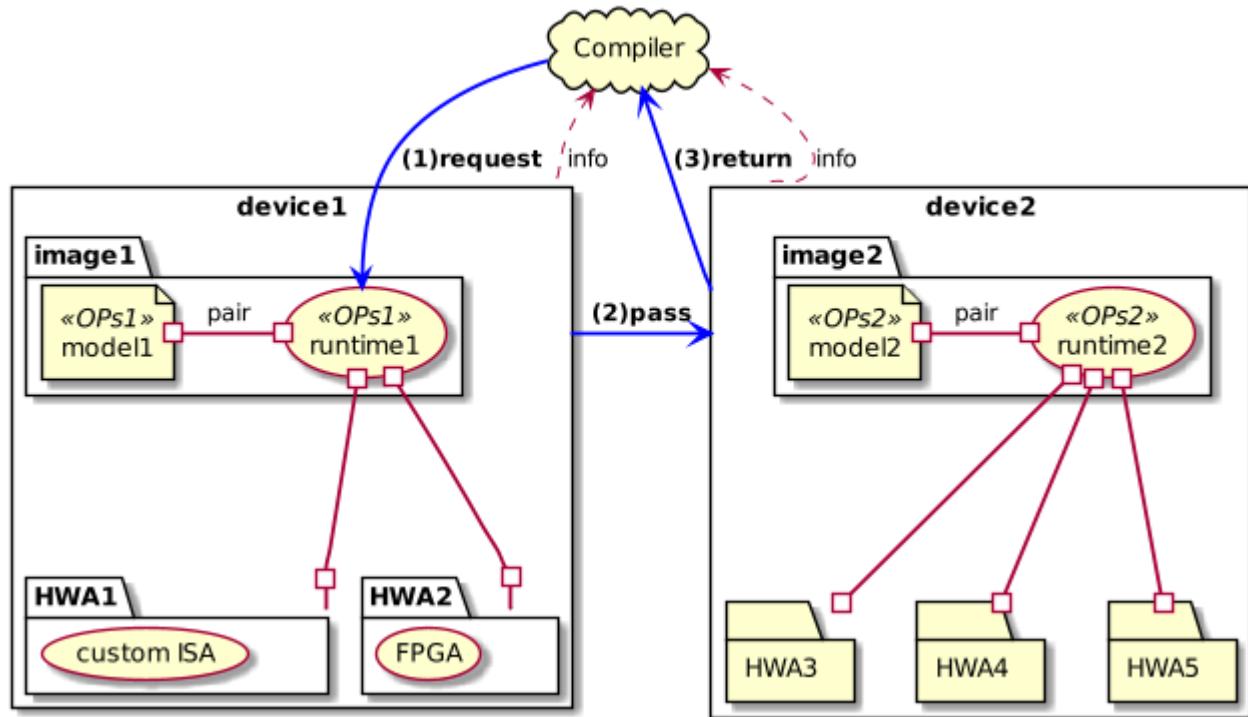
Demonstrated End-to-End TinyML (as-a-Service?)

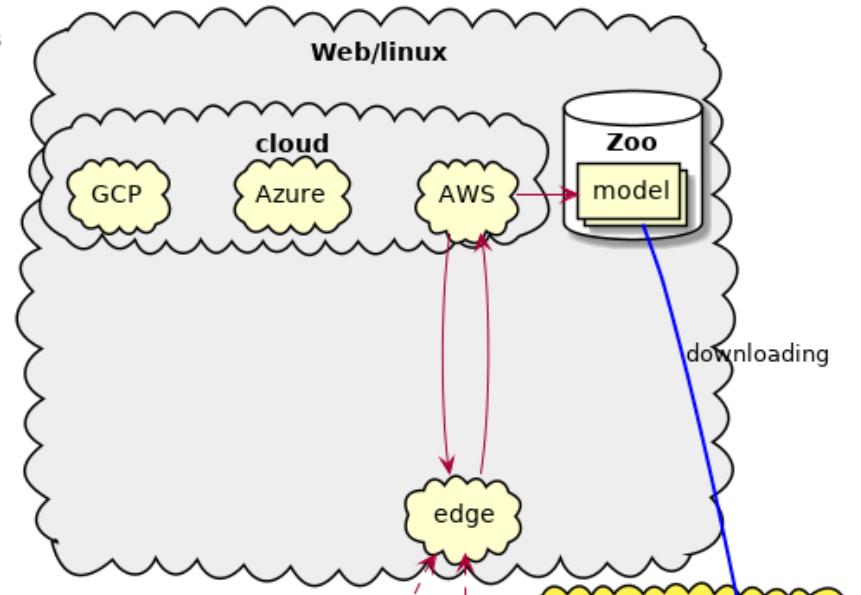
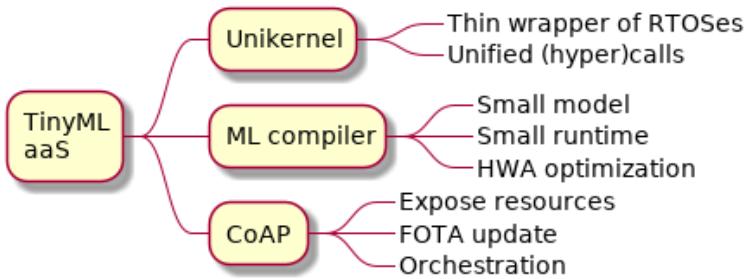


# Next

1. Add **Orchestration** with CoAP?
2. Supprt other MCUs and/or **complicated** models?
3. **Heterogeneous** (distributed) training?
4. Distributed inference on **heterogeneous** HWAs?

multi-node, multi-HWA, scheduling

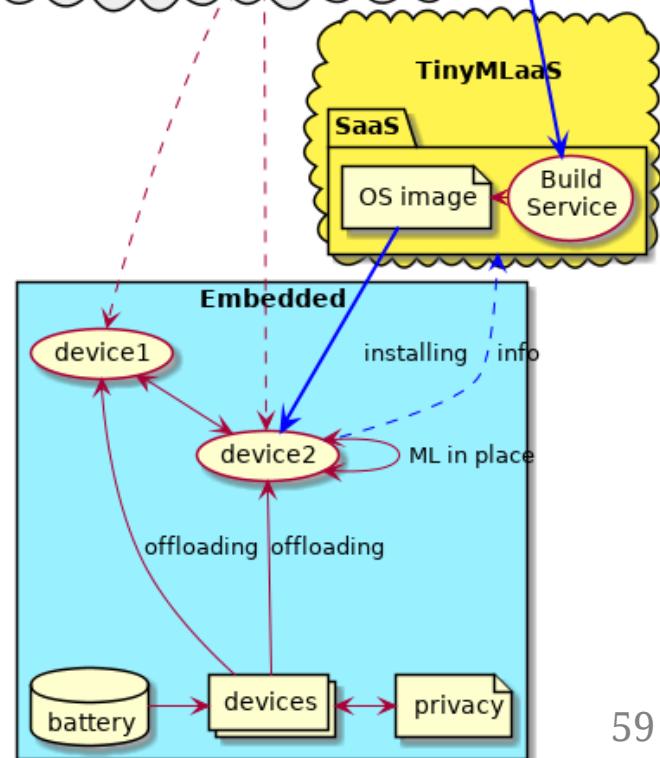




# *TinyML as-a-Service*

*could bring ML*

*onto IoT.*





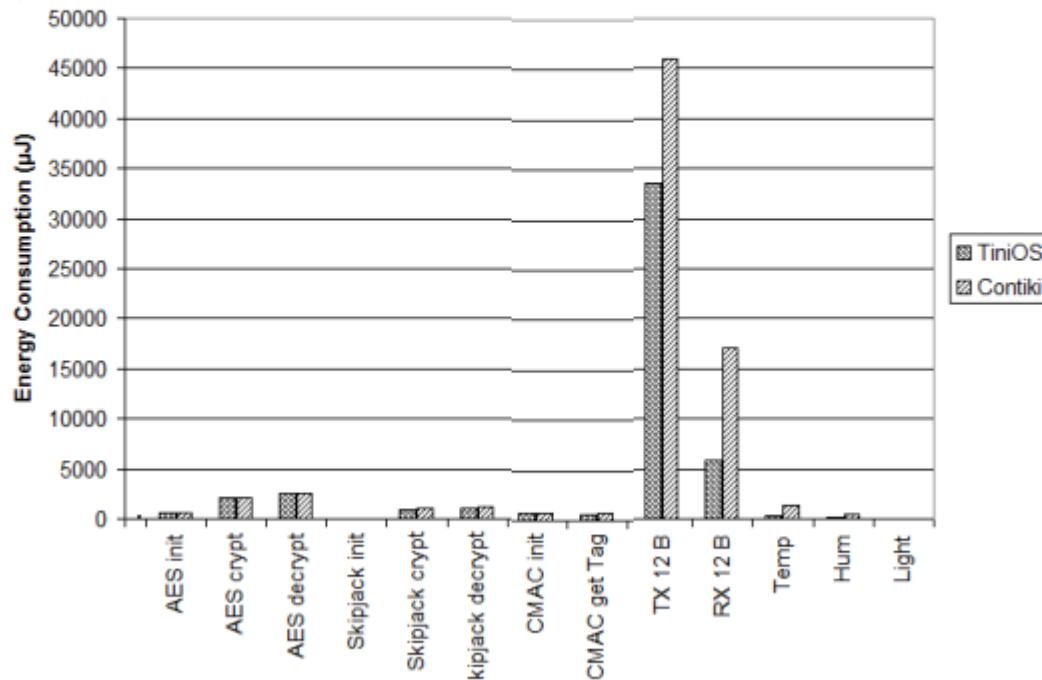
**ERICSSON**

# **Appendix**

# Garrulity

## Energy consumption on TelosB

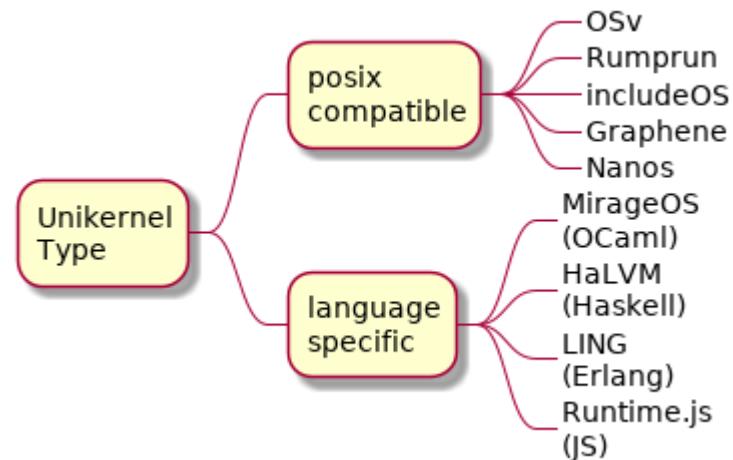
Message exchange cost: orders of magnitude more than processing, symmetric crypto



C.B. Margi, B.T.de Oliveira, G.T.de Souza, M.A. Simplicio Jr, P.S.L.M. Barreto,  
T.C.M.B. Carvalho, M. Näslund, R. Gold, ICCCN'2010 / IEEE WiMAN 2010]

34

# Type of Unikernel



# MirageOS in OCaml

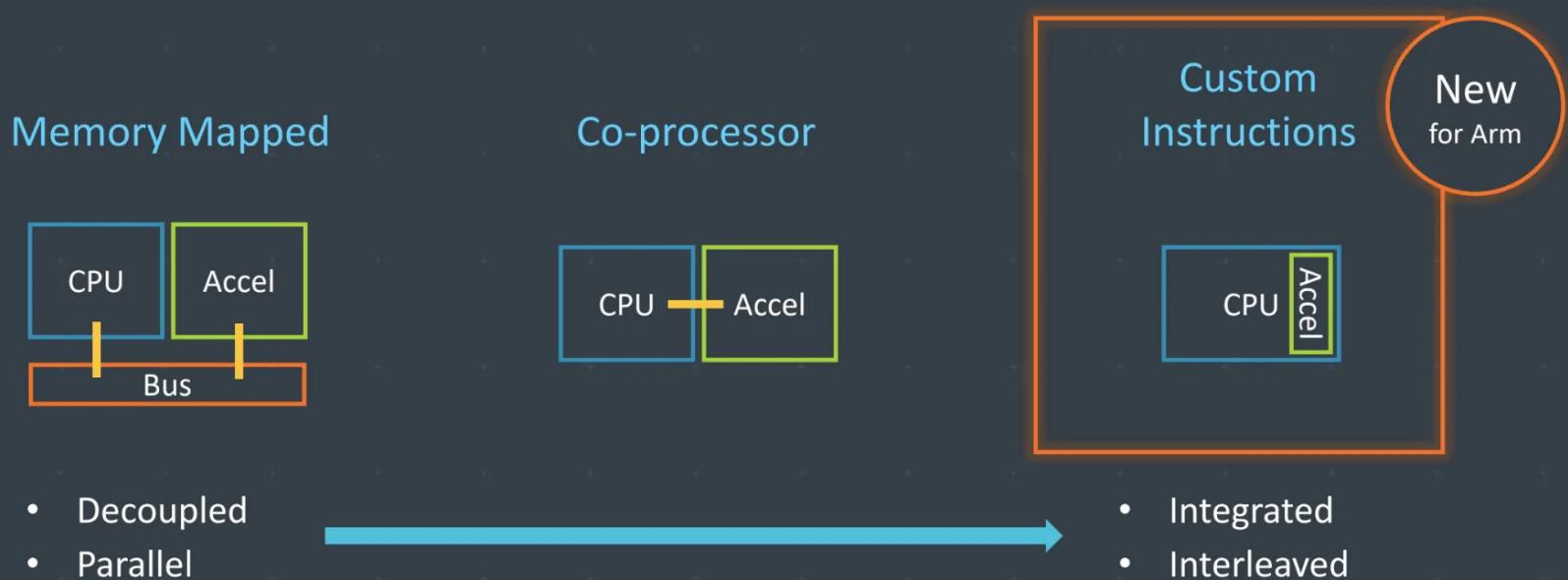
```
open Lwt.Infix

module Main (S: Mirage_stack_lwt.V4) = struct

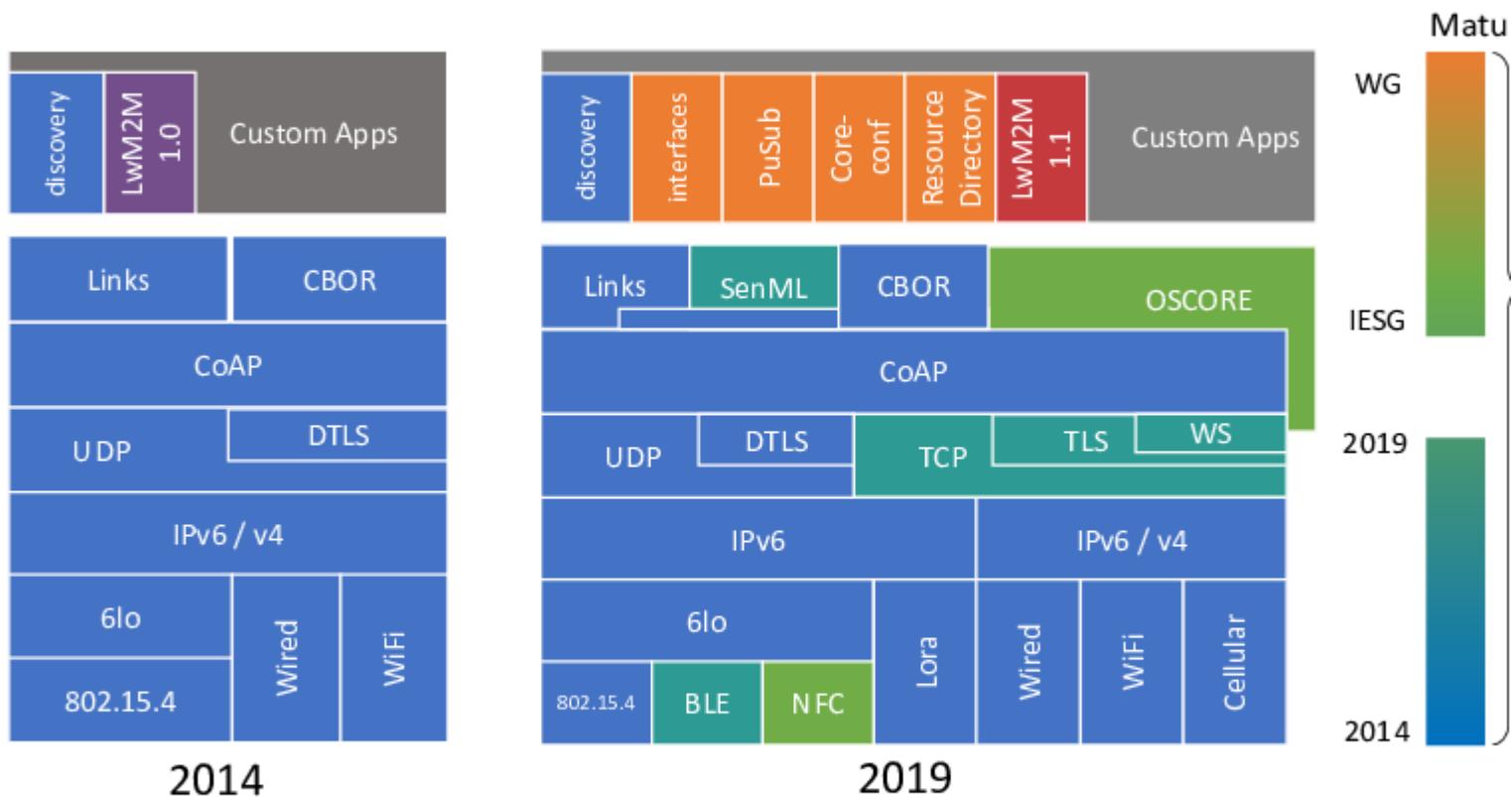
  let start s =
    let port = Key_gen.port () in
    S.listen_tcpv4 s ~port (fun flow ->
      let dst, dst_port = S.TCPV4.dst flow in
      Logs.info (fun f -> f "new tcp connection from IP %s on port %d$"
                  (Ipaddr.V4.to_string dst) dst_port);
      S.TCPV4.read flow >>= function
        | Ok `Eof -> Logs.info (fun f -> f "Closing connection!"); Lwt.$
        | Error e -> Logs.warn (fun f -> f "Error reading data from est$"
        | Ok (`Data b) ->
          Logs.debug (fun f -> f "read: %d bytes:\n%s" (Cstruct.len b) $
          S.TCPV4.close flow
      );
      S.listen s
    end
-UU-:----F1 unikernel.ml All L1 Git-cddddd57 (Tuareg ARev Merlin (
Beginning of buffer
0:emacs#- 1:-* 2:>68kB/s 25C 0.59 8x1.2GHz 31.3G18% 131.160.51.146 2019
```

# Custom ISA, RISC-V & Arm

## Meeting the requirements for acceleration



# Standards Device Stack

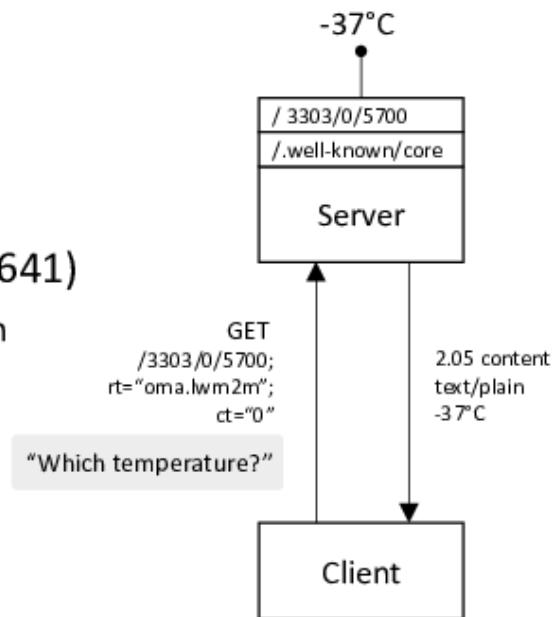


13/2/19

One Data Model Group

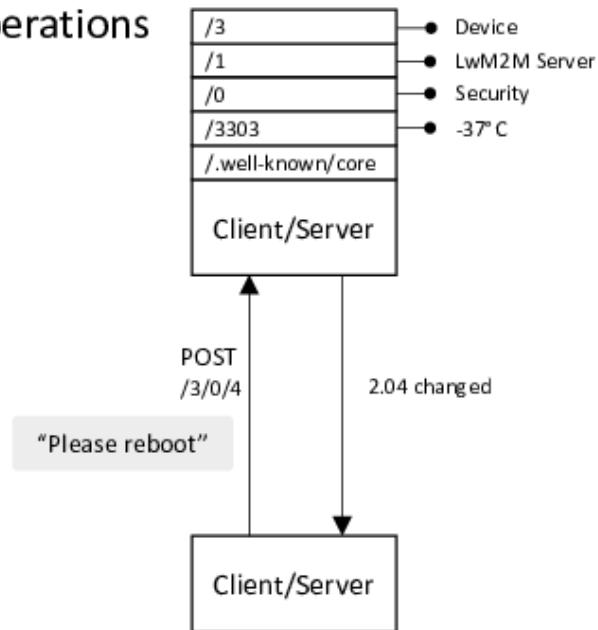
# The Constrained Application Protocol (CoAP)

- CoRELink (RFC6690) provides a link format
  - Reuses Web Linking RFC5988 for IoT.
  - Enables query parameters for discovery (lt, gt...)
  - Enables attribute and relation types (rt, if, sz).  
`<3303/0/5700>;rt="oma:lwm2m:temp";ct="0"`
- Notifications available through *observe* option (RFC7641)
  - Can observe and add query parameters to the observation  
`<3303/0/5700?lt=0>`
- The “*/.well-known/core*” URI provides discovery
- Multiple serialization formats used with CoAP
  - SenML (RFC8428): Minimalistic JSON
  - CBOR (RFC7049): Binary serialization
- Multiple implementations available at [coap.technology](http://coap.technology)

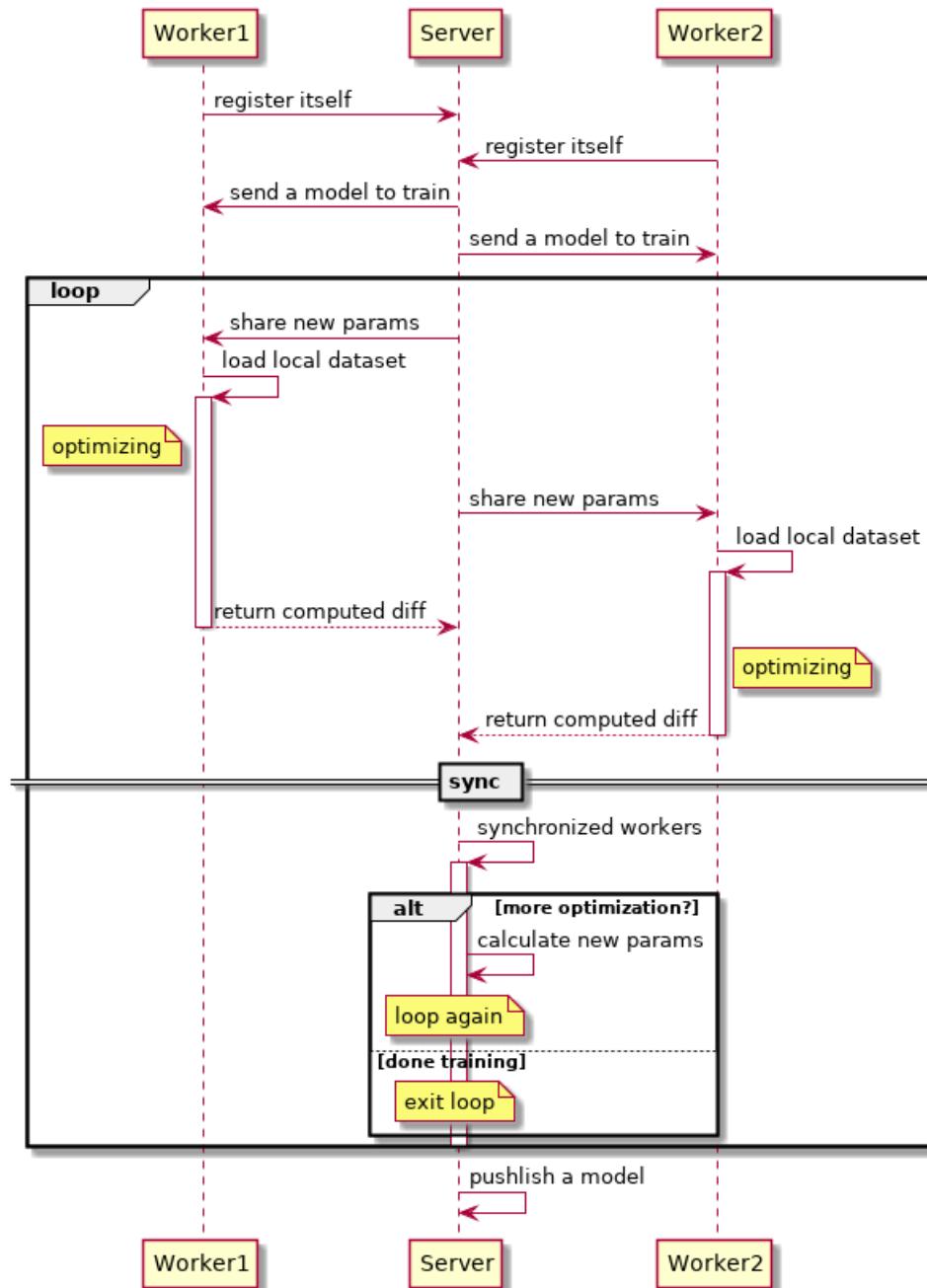


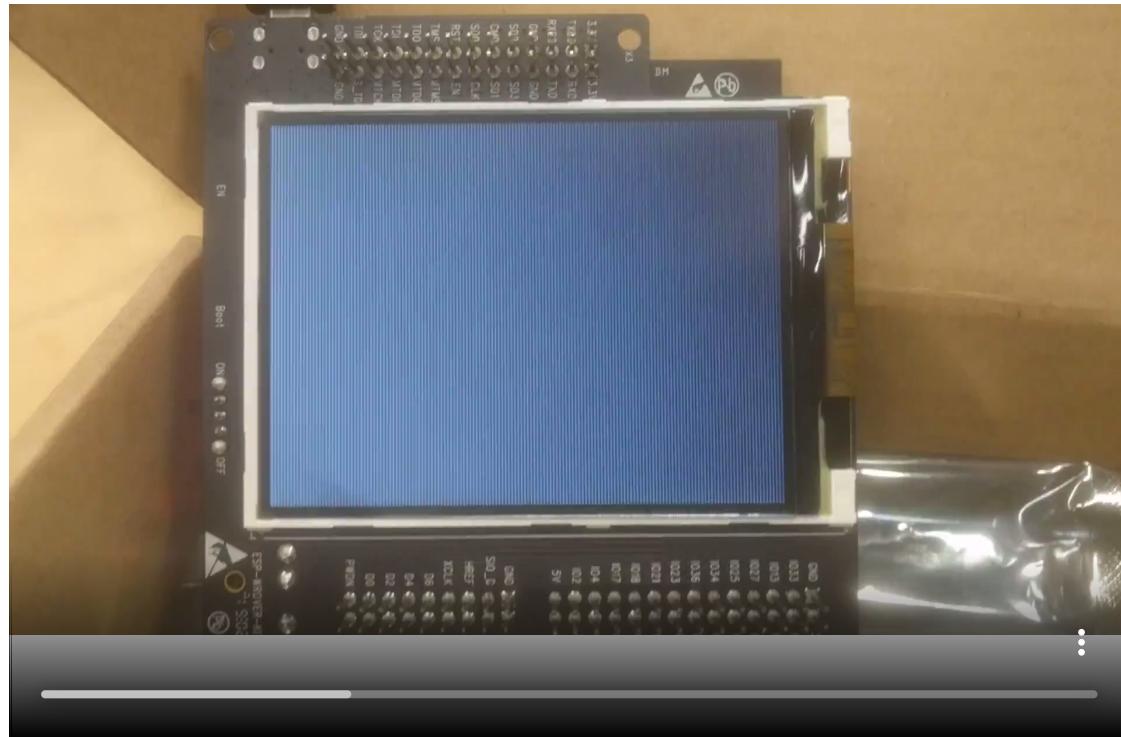
# The LightWeight M2M Protocol (LwM2M)

- Mapping of CoAP methods (GET, POST, PUT...) to **CRUD** operations
- Interaction with device through simple “Objects”
  - RWX, Access Control, Observation, Notification
  - Independent from underlying protocol stack (CoAP today)
  - Simple resource structure
  - Objects’ resources are accessed with simple URIs:  
$$\{/Object\ ID\}/\{Object\ Instance\}/\{Resource\ ID\}$$
  - Multiple serializations:  
*For example JSON, CBOR and raw values.*
- Common repository for all Objects (OMNA)
  - Enables interoperability and reusability



### Training: Parameter Server(PS) sequence





# Fashion MNIST from Zalando