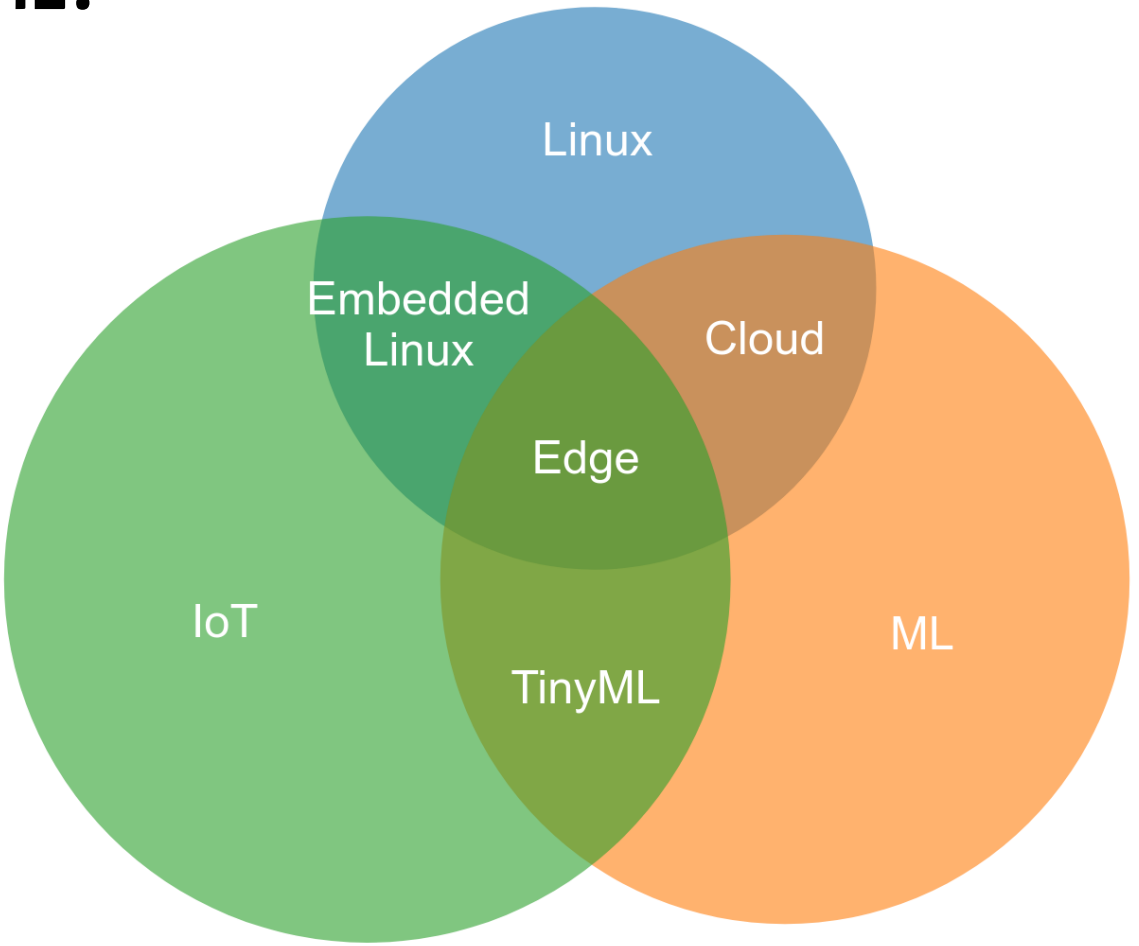
Distributed ML on Unikernel for IoT

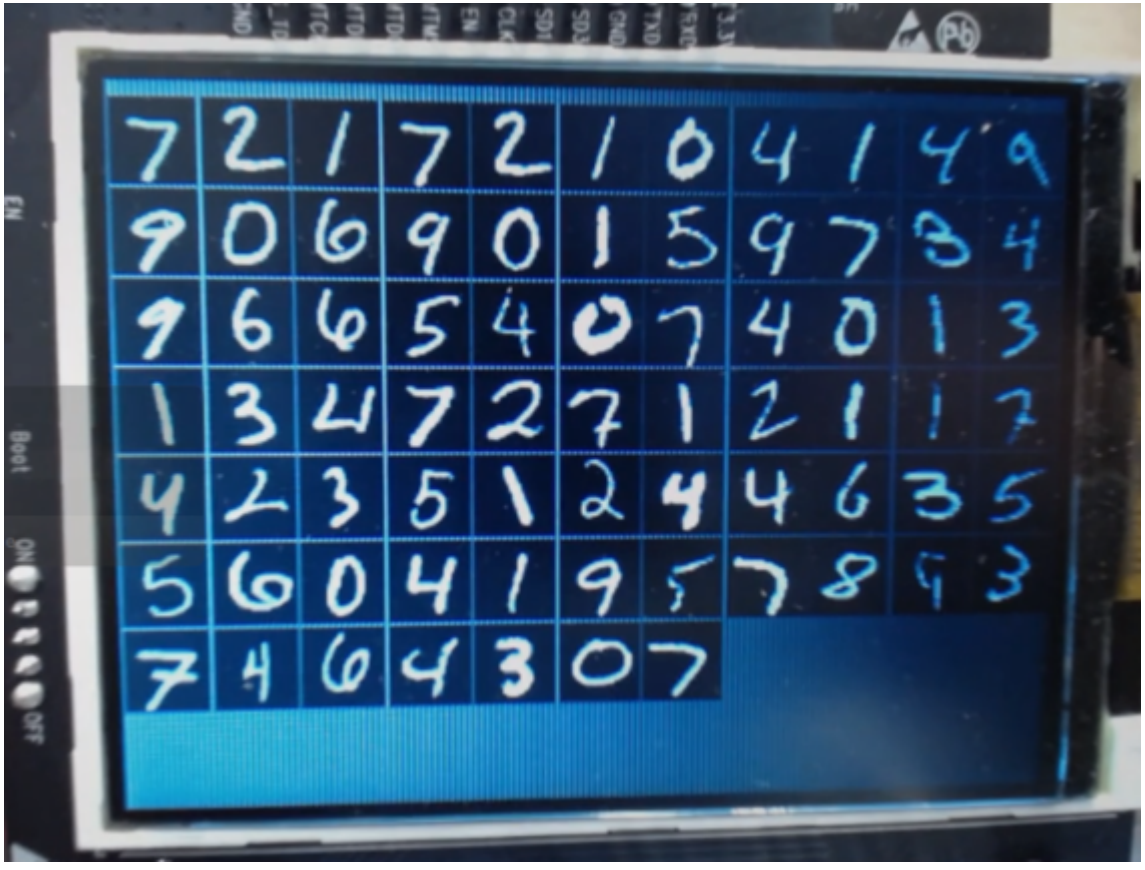
TinyML as-a-Service

Hiroshi Doyu <hiroshi.doyu@ericsson.com>

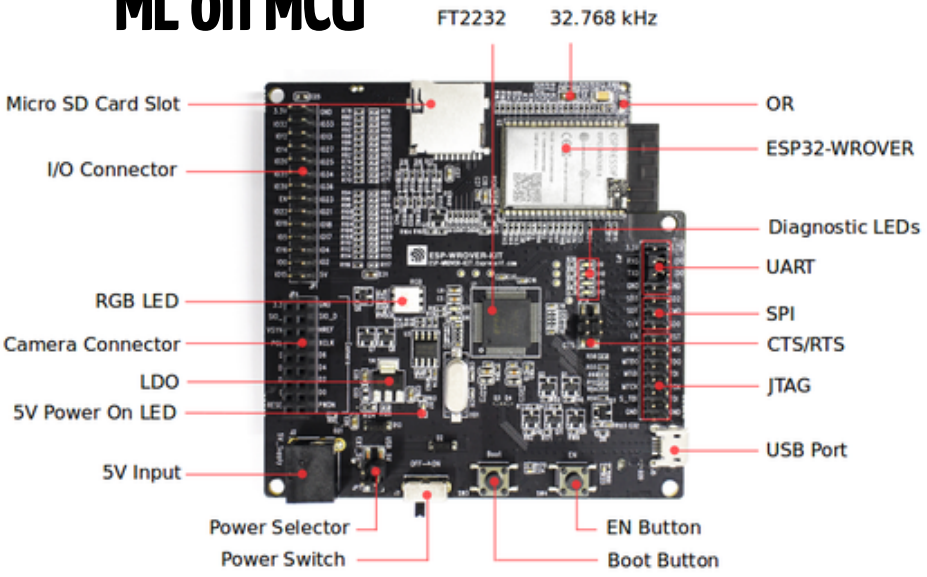
TinyML?

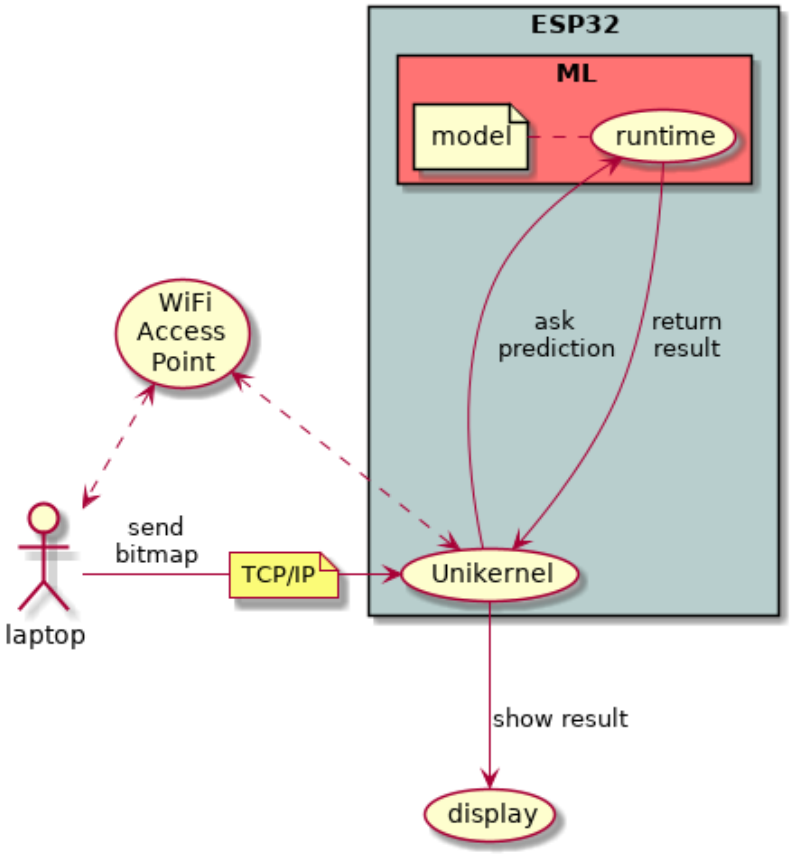


Demo

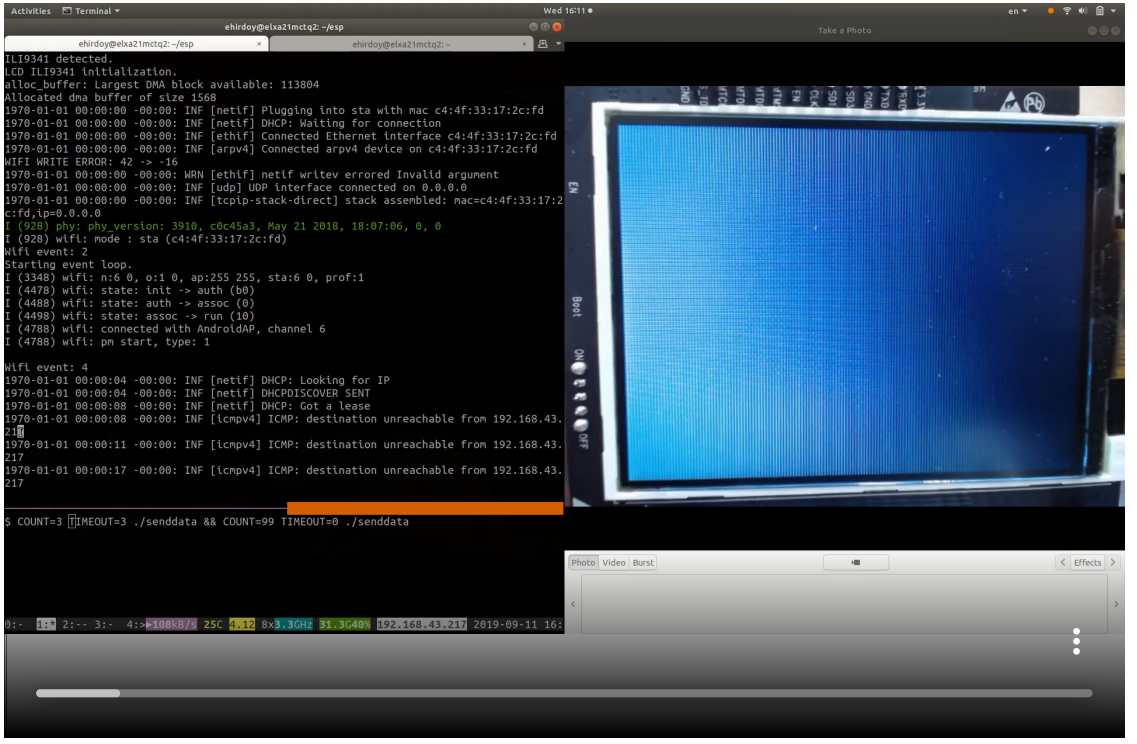


ML on MCU





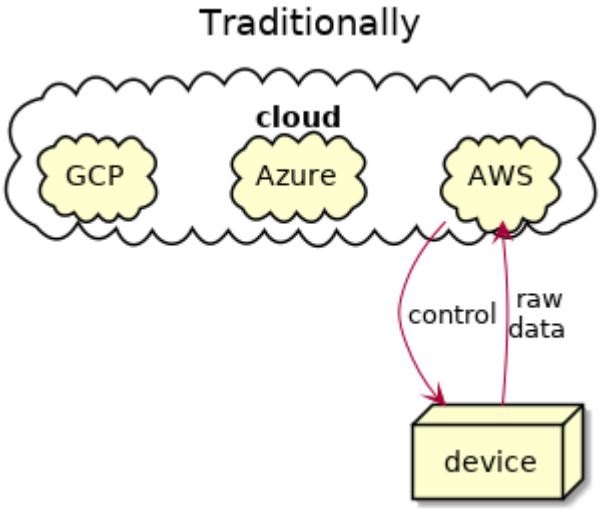
MCU: ESP32
ARCH: Xtensa 32bit
RAM: 520KB SRAM
ROM: 4MB FLASH
160MHz / 600 DMIPS

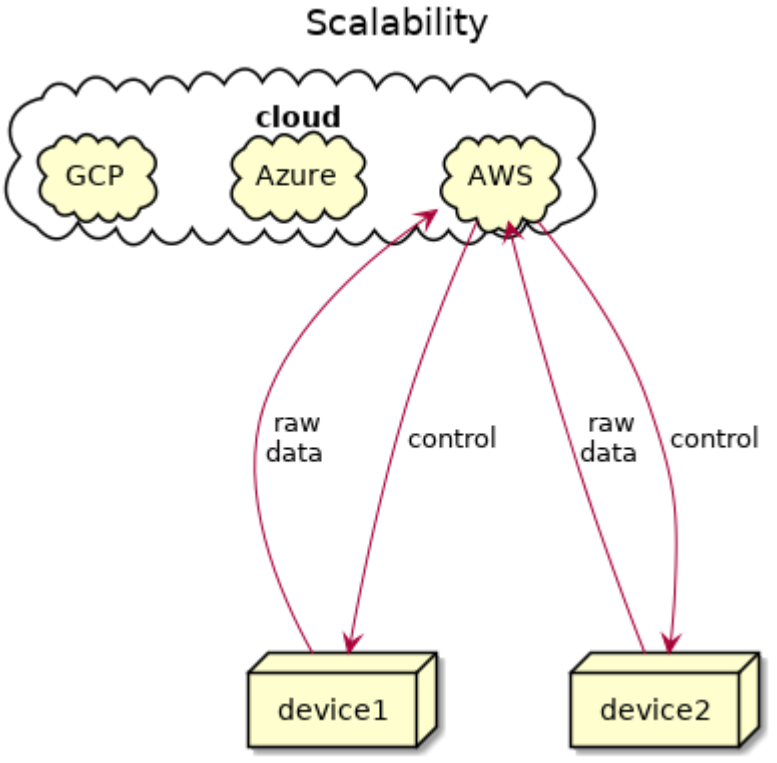


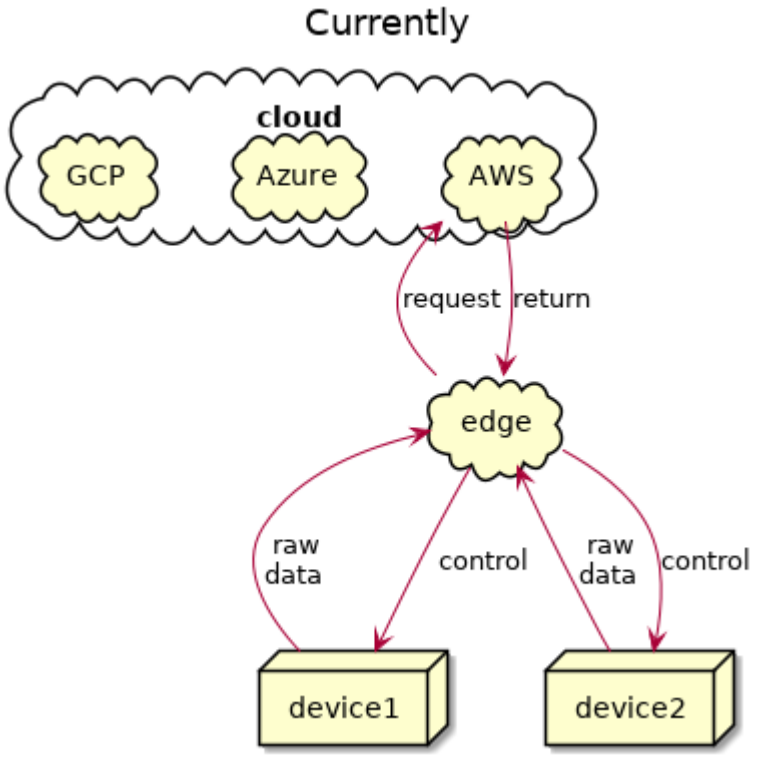
MNIST on ESP32

Outline

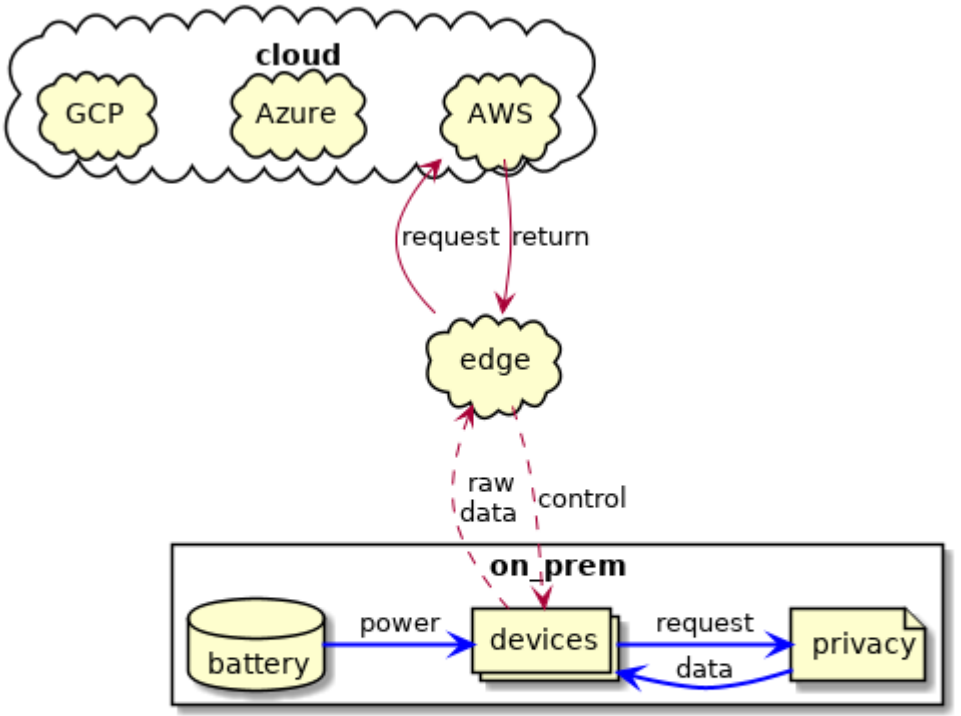
1. ~~Demo~~
2. **Problems**
    - **Edge Computing**
    - Web vs Embedded
    - ML environment
3. Proposal
4. Three Enablers
5. PoC
6. Conclusion


Traditionally
cloud
GCP
Azure
AWS
control
raw
data
device


Scalability
cloud
GCP
Azure
AWS
raw
data
control
raw
data
control
device1
device2

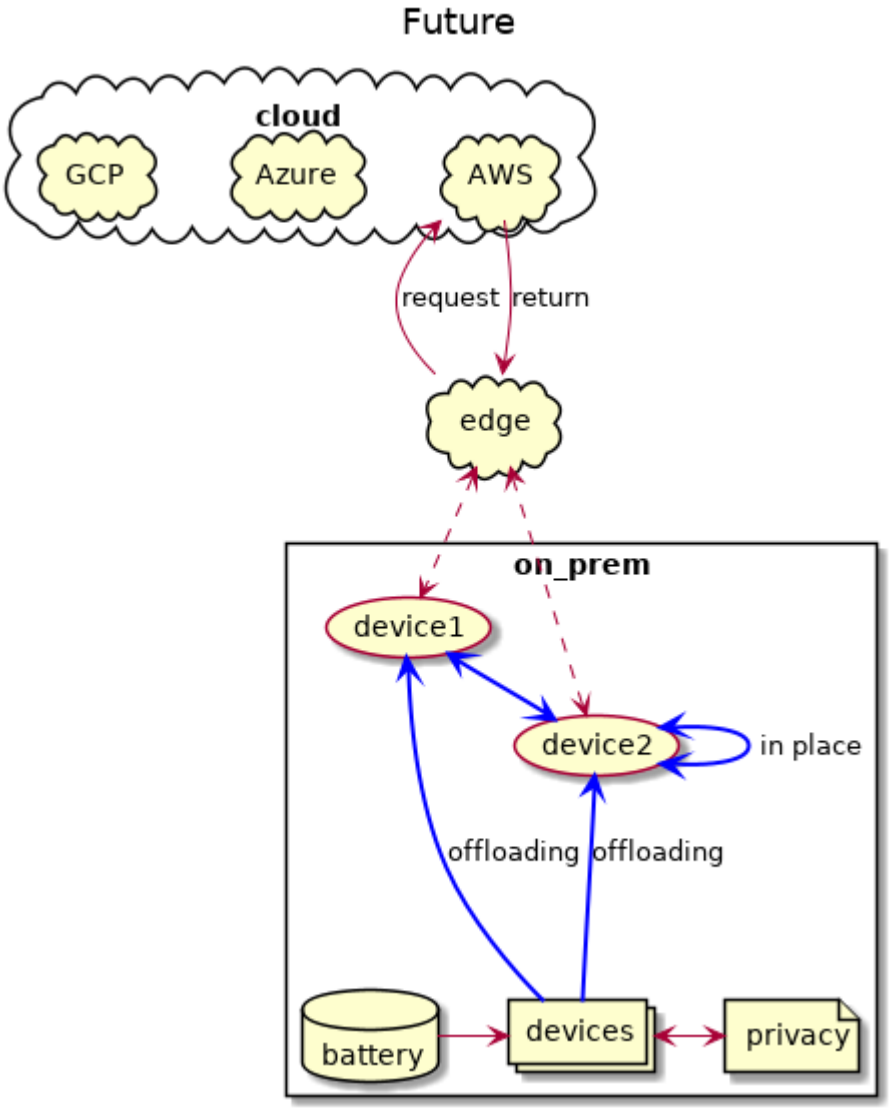

Currently
cloud
GCP
Azure
AWS
request
return
edge
raw
data
control
raw
data
control
device1
device2

Edge computing problems



Privacy
Bandwidth
Latency
Reliability
Energy

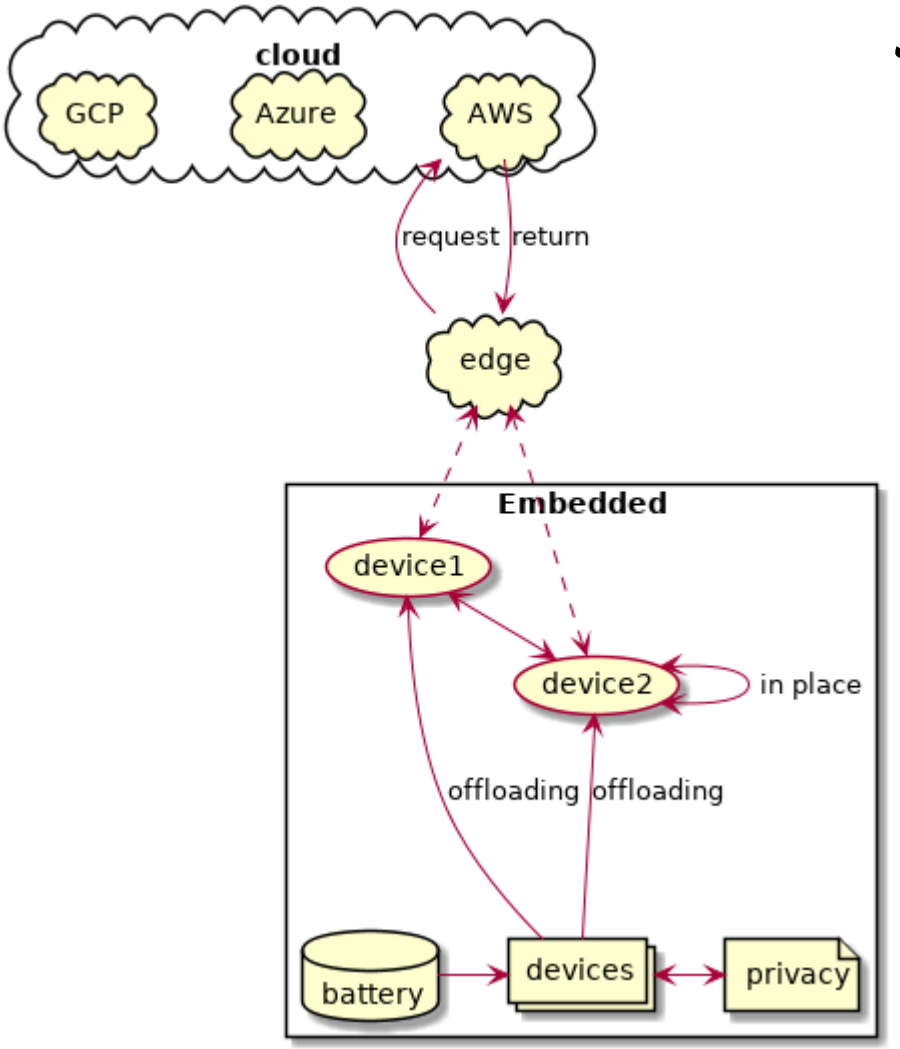
expensive transmission


Future
cloud
GCP
Azure
AWS
request
return
edge
on_prem
device1
device2
in place
offloading
offloading
battery
devices
privacy

Outline

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

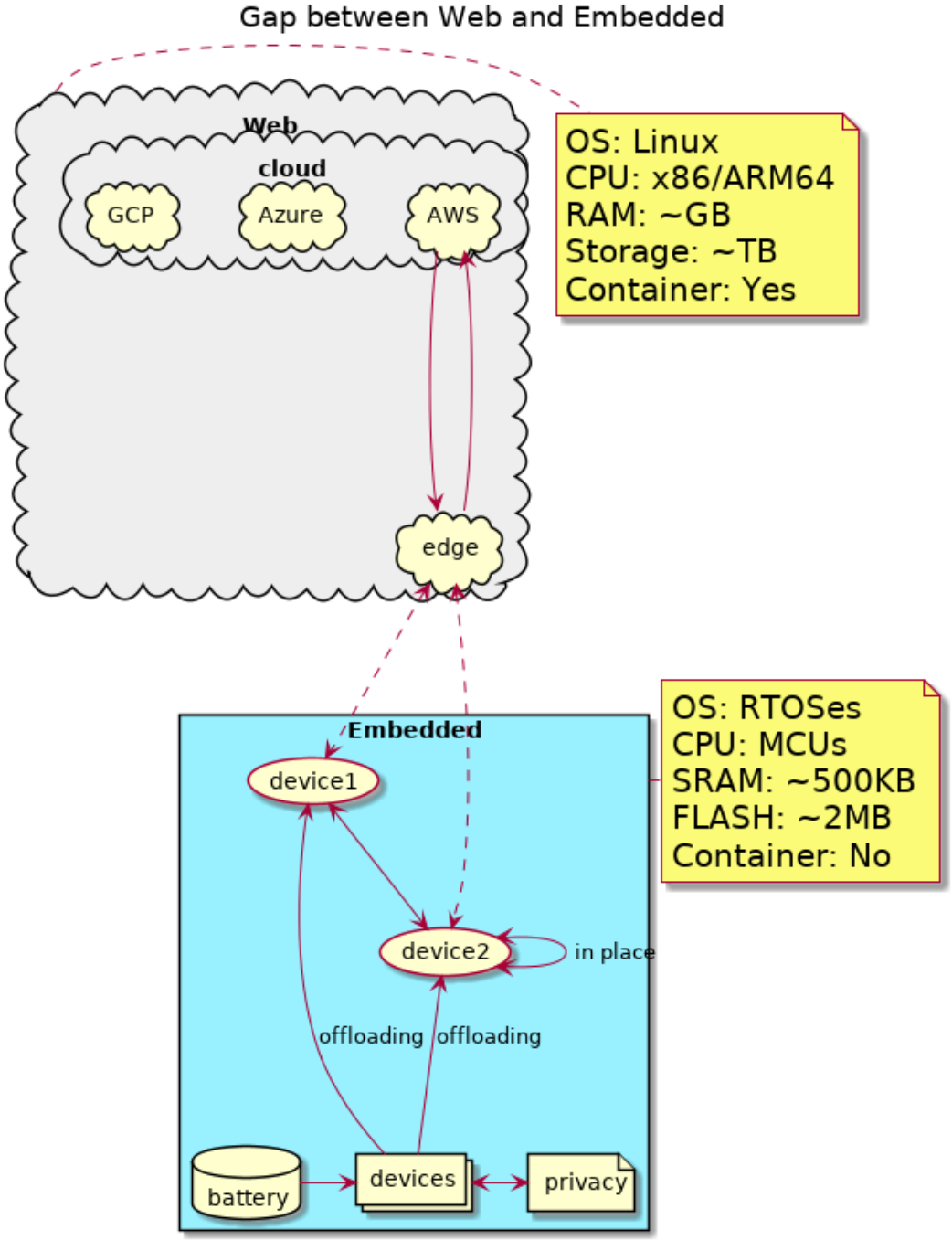
3 areas?



Cloud

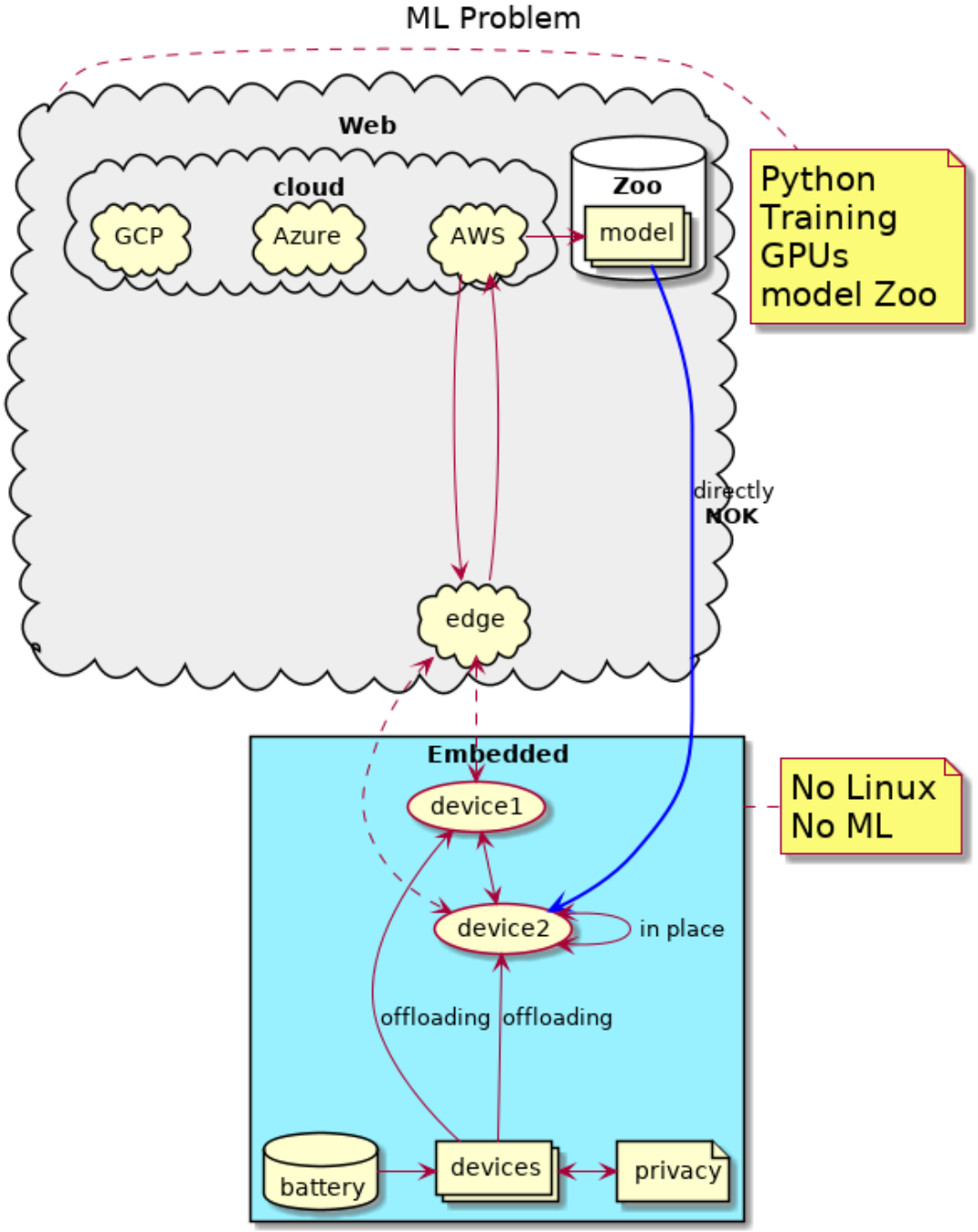
Edge

Embedded

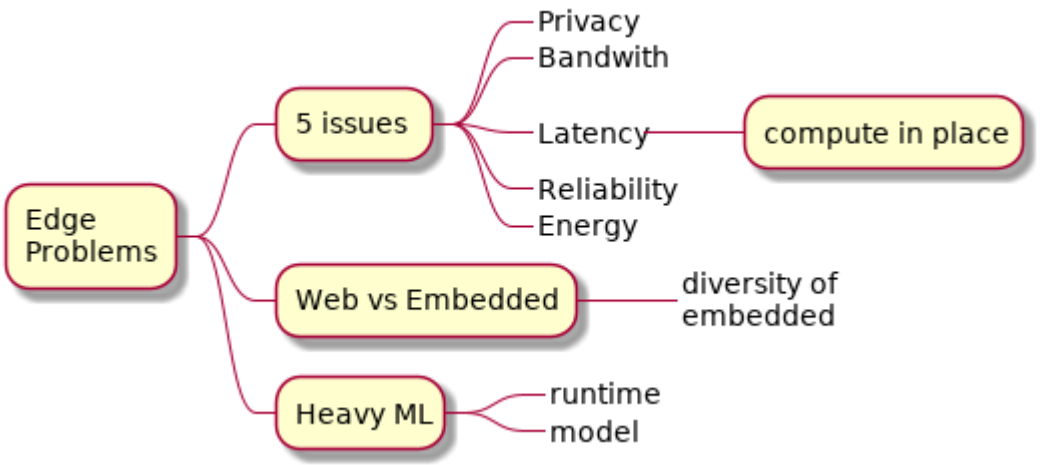

Gap between Web and Embedded
Web
cloud
GCP
Azure
AWS
edge
OS: Linux
CPU: x86/ARM64
RAM: ~GB
Storage: ~TB
Container: Yes
Embedded
device1
device2
in place
offloading
offloading
battery
devices
privacy
OS: RTOSes
CPU: MCUs
SRAM: ~500KB
FLASH: ~2MB
Container: No

Outline

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

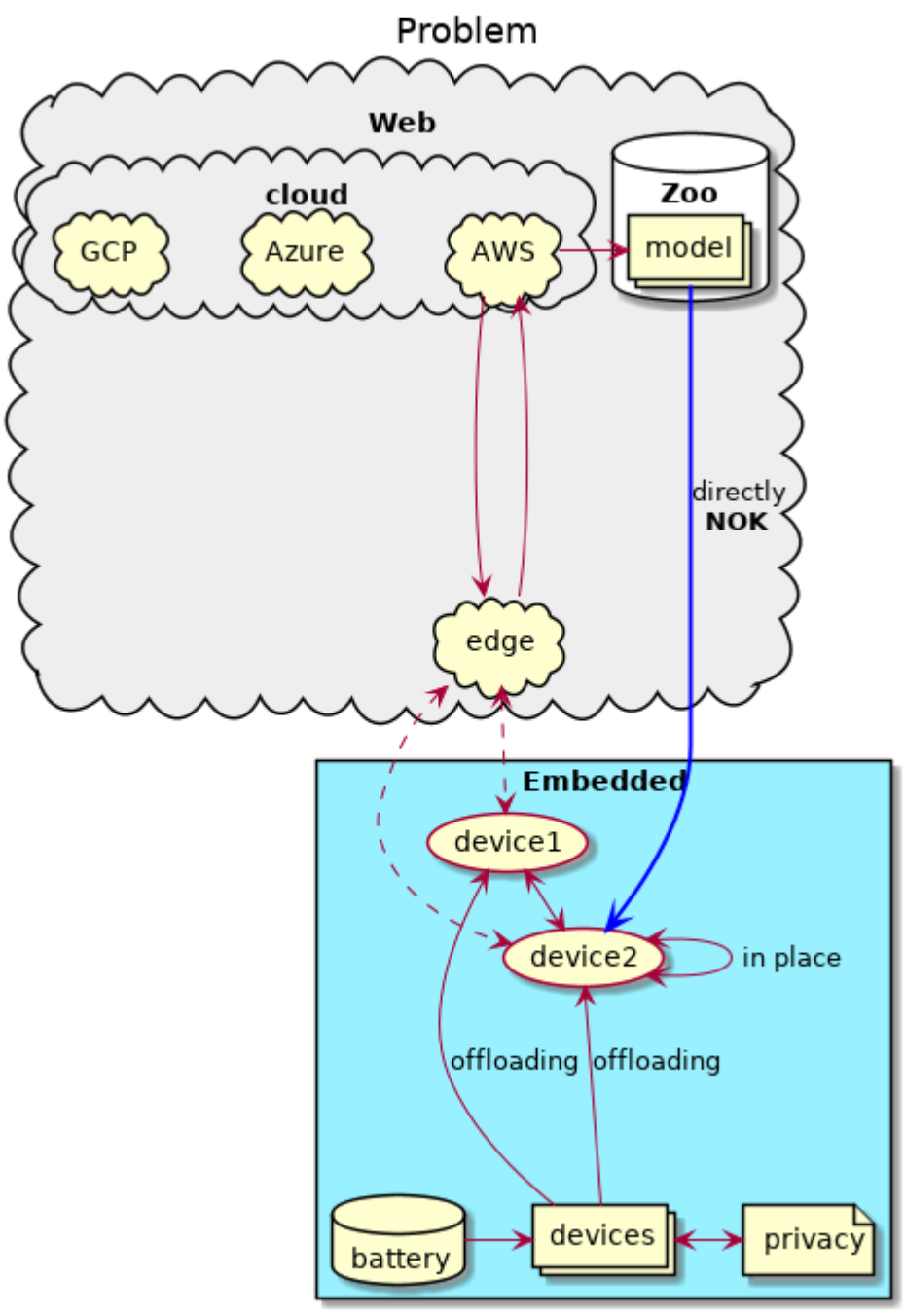

ML Problem
Web
cloud
GCP
Azure
AWS
Zoo
model
Python
Training
GPUs
model Zoo
directly
NOK
edge
Embedded
device1
device2
in place
offloading
offloading
battery
devices
privacy
No Linux
No ML

Summary: Problems



Outline

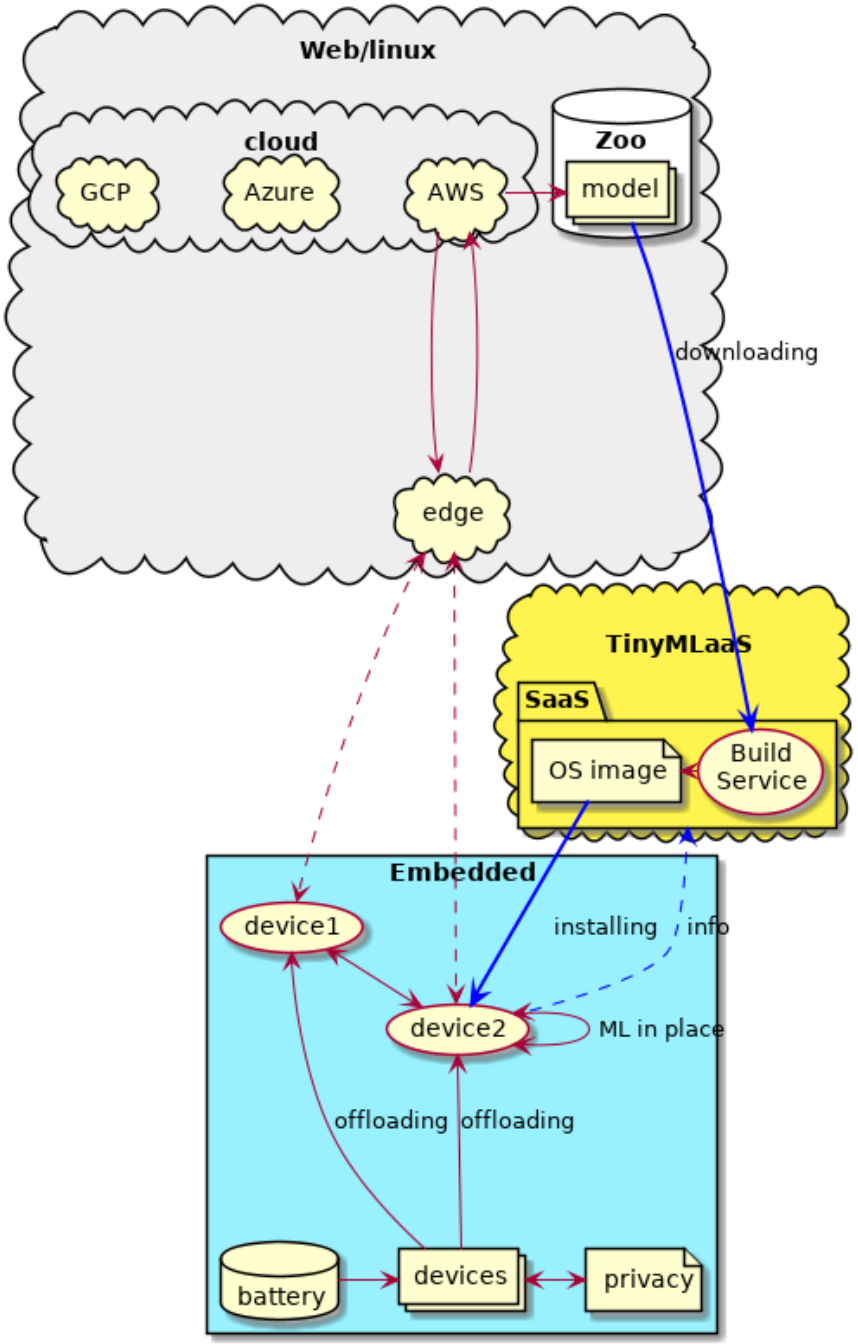
1. ~~Demo~~
2. ~~Problems~~
3. **Proposal**
4. Three Enablers
5. PoC
6. Conclusion

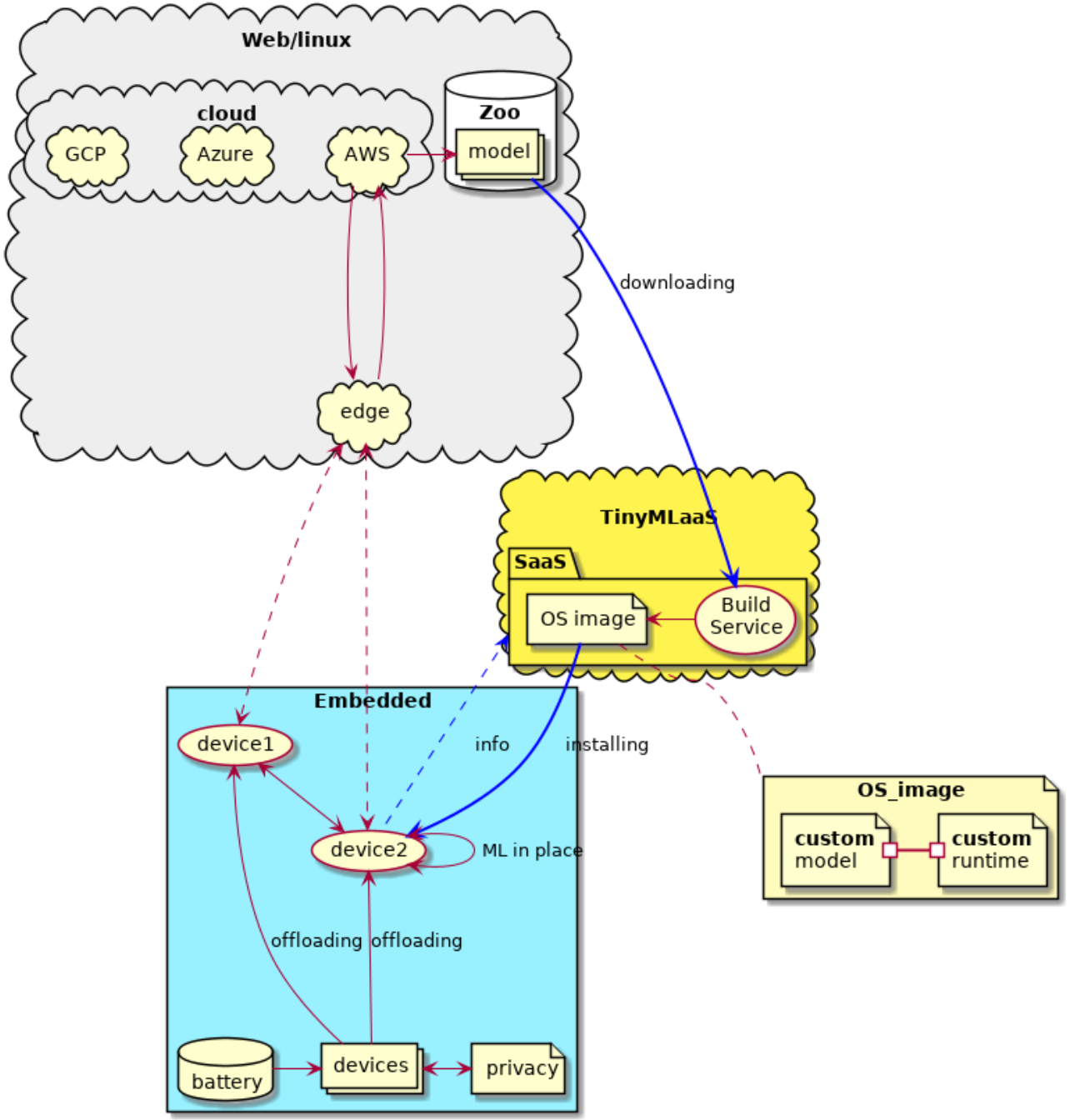

Problem
Web
cloud
GCP
Azure
AWS
Zoo
model
directly
NOK
edge
Embedded
device1
device2
in place
offloading
offloading
battery
devices
privacy

Squeeze ML

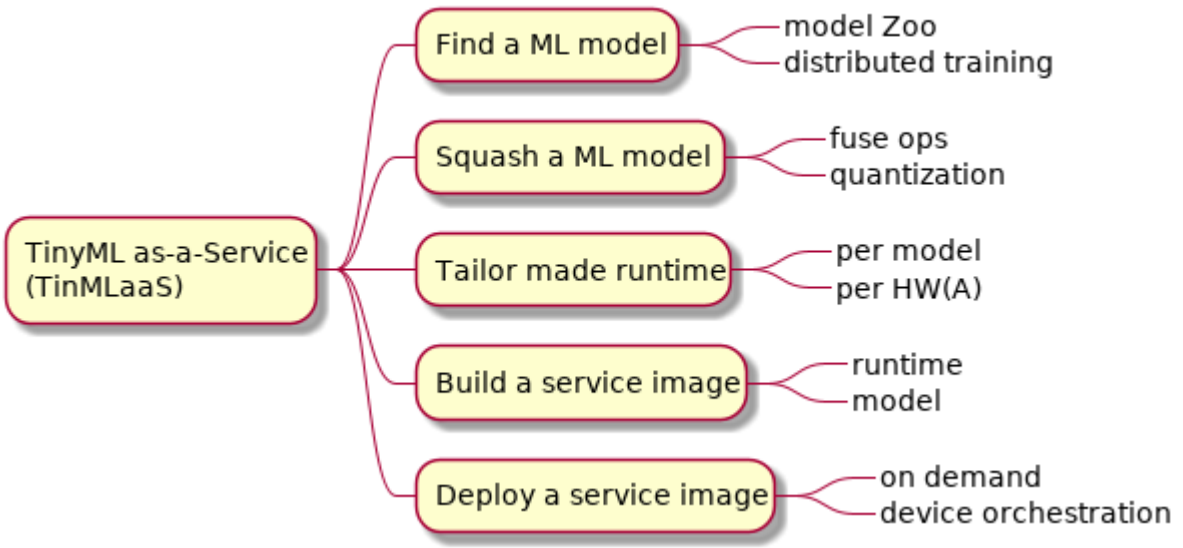
model & runtime

- per RTOS
- per HWA


Web/linux
cloud
GCP
Azure
AWS
Zoo
model
downloading
edge
TinyMLaaS
SaaS
OS image
Build Service
Embedded
device1
installing
info
device2
ML in place
offloading
offloading
battery
devices
privacy


Web/linux
cloud
GCP
Azure
AWS
Zoo
model
edge
downloading
TinyMLaaS
SaaS
OS image
Build Service
Embedded
device1
info
installing
device2
ML in place
offloading
offloading
battery
devices
privacy
OS_image
custom model
custom runtime

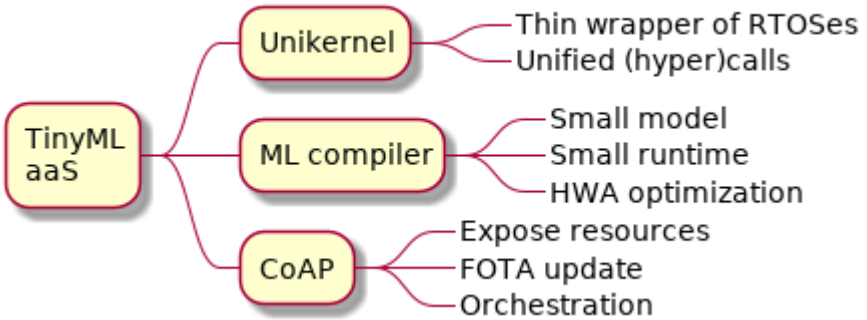
TinyML as-a-Service



Outline

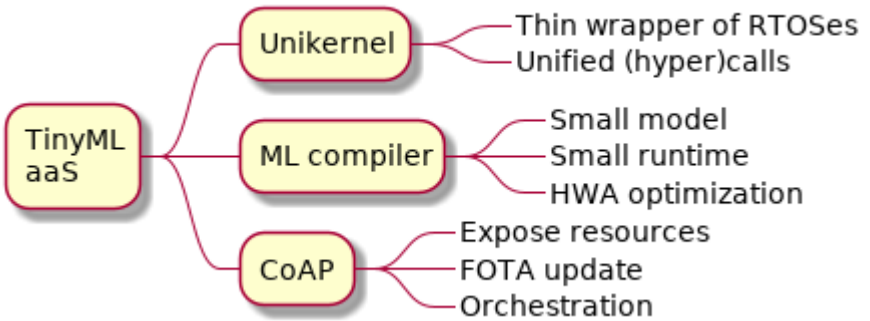
1. ~~Demo~~
2. ~~Problems~~
3. ~~Proposal~~
4. **Three Enablers**
5. PoC
6. Conclusion

Three enblers

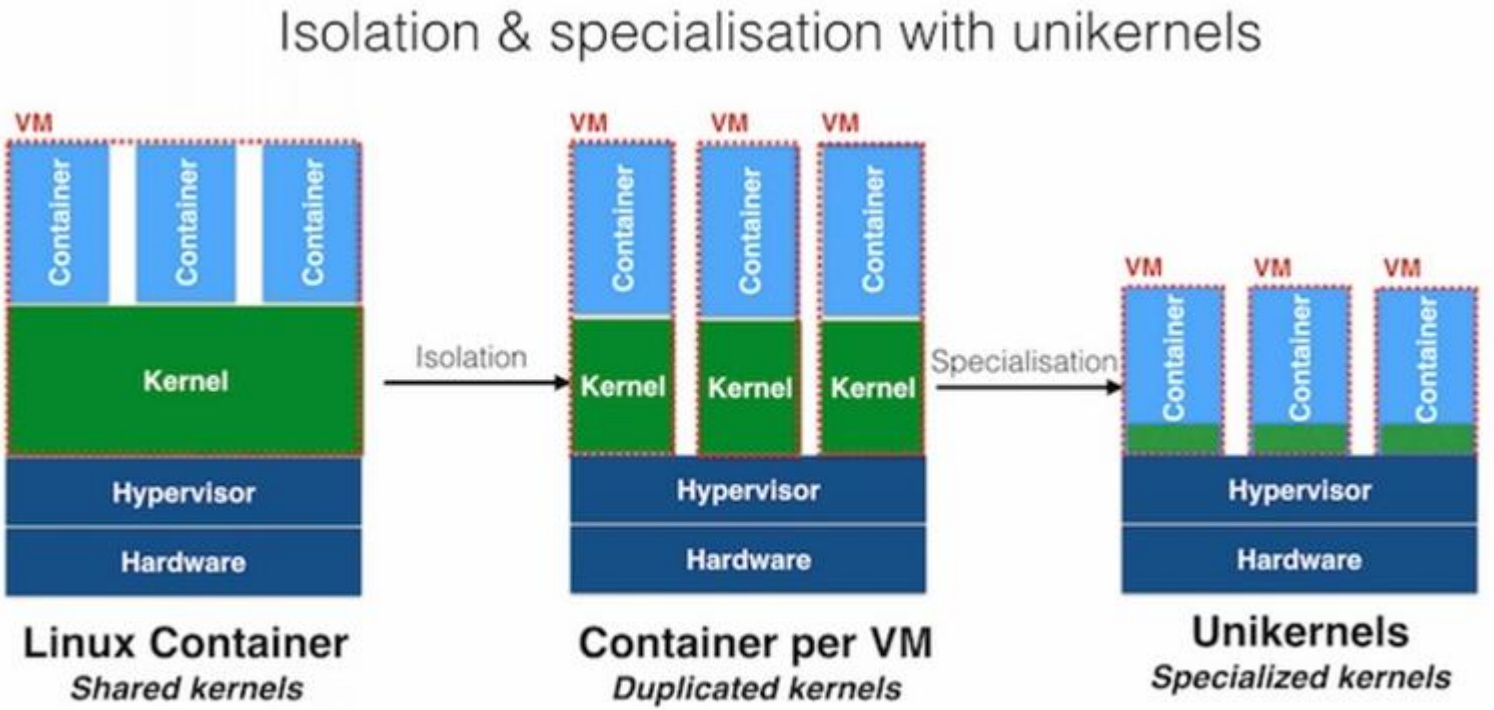


Outline

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

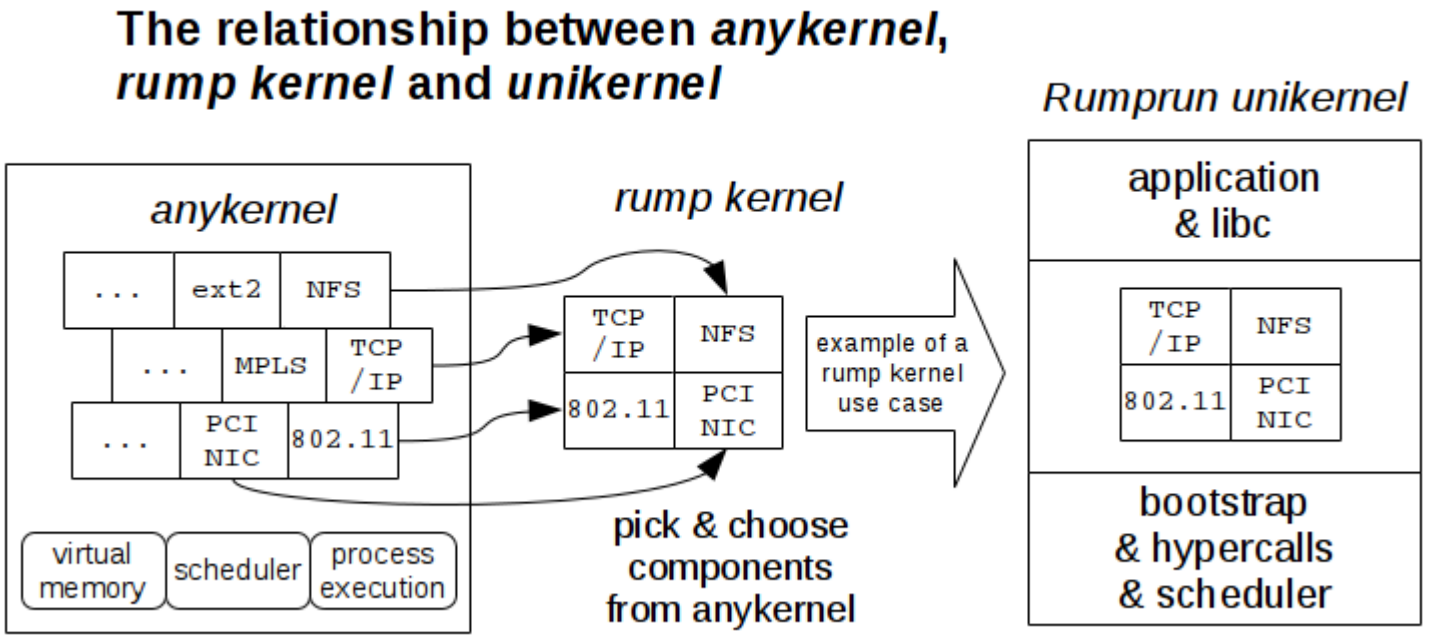


Unikernel



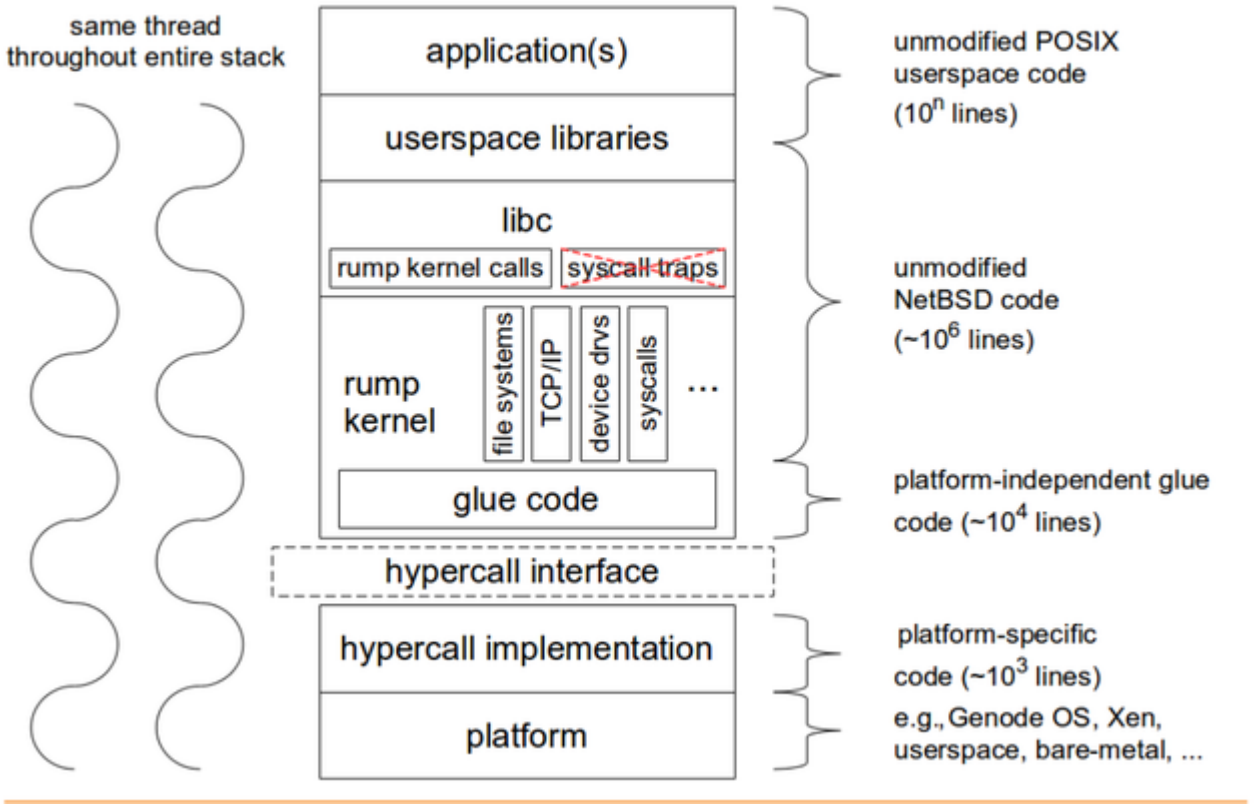
http://unikernel.org/files/2014-cacm-unikernels.pdf

library Operating System (libOS)

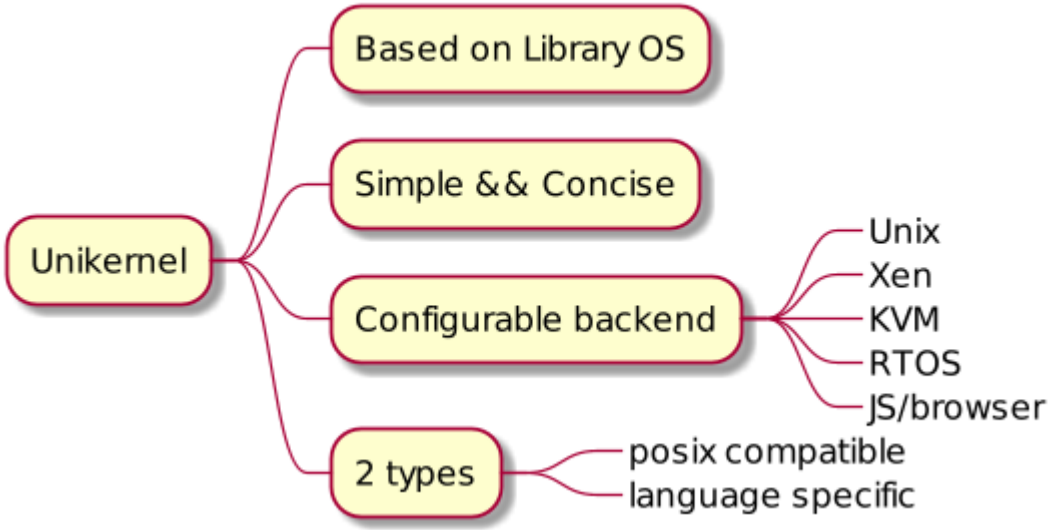


https://xenproject.org/wp-content/uploads/sites/79/2015/08/anyunirumpkernel.png

Internal


same thread
throughout entire stack
application(s)
userspace libraries
libc
rump kernel calls
syscall traps
rump
kernel
file systems
TCP/IP
device drvs
syscalls
...
glue code
hypercall interface
hypercall implementation
platform
unmodified POSIX
userspace code
(10^n lines)
unmodified
NetBSD code
(~10^6 lines)
platform-independent glue
code (~10^4 lines)
platform-specific
code (~10^3 lines)
e.g., Genode OS, Xen,
userspace, bare-metal, ...

Summary: Unikernel

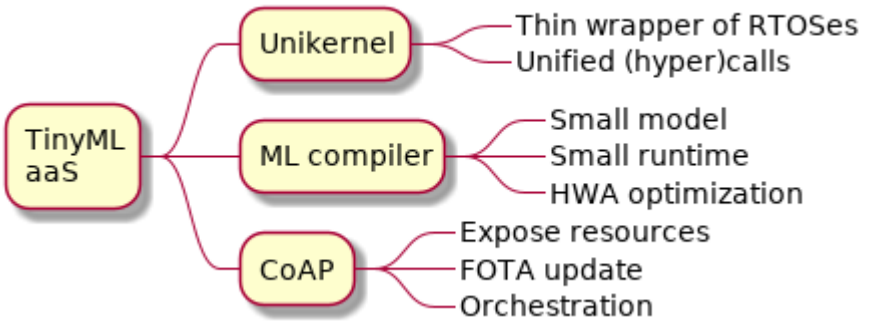


Can be a thin **wrapper** of different RTOSes?

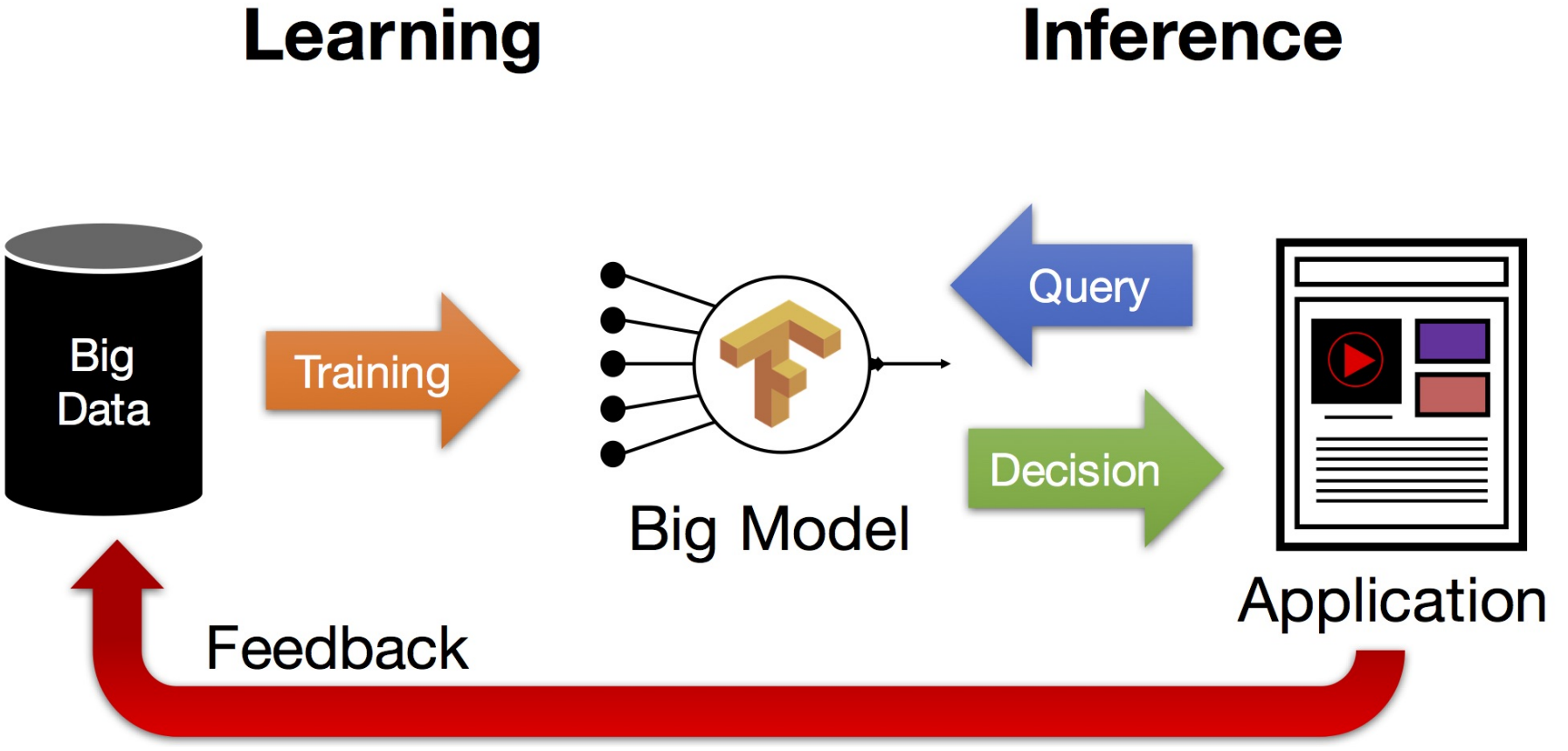
2 types: high level programming languages

Outline

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



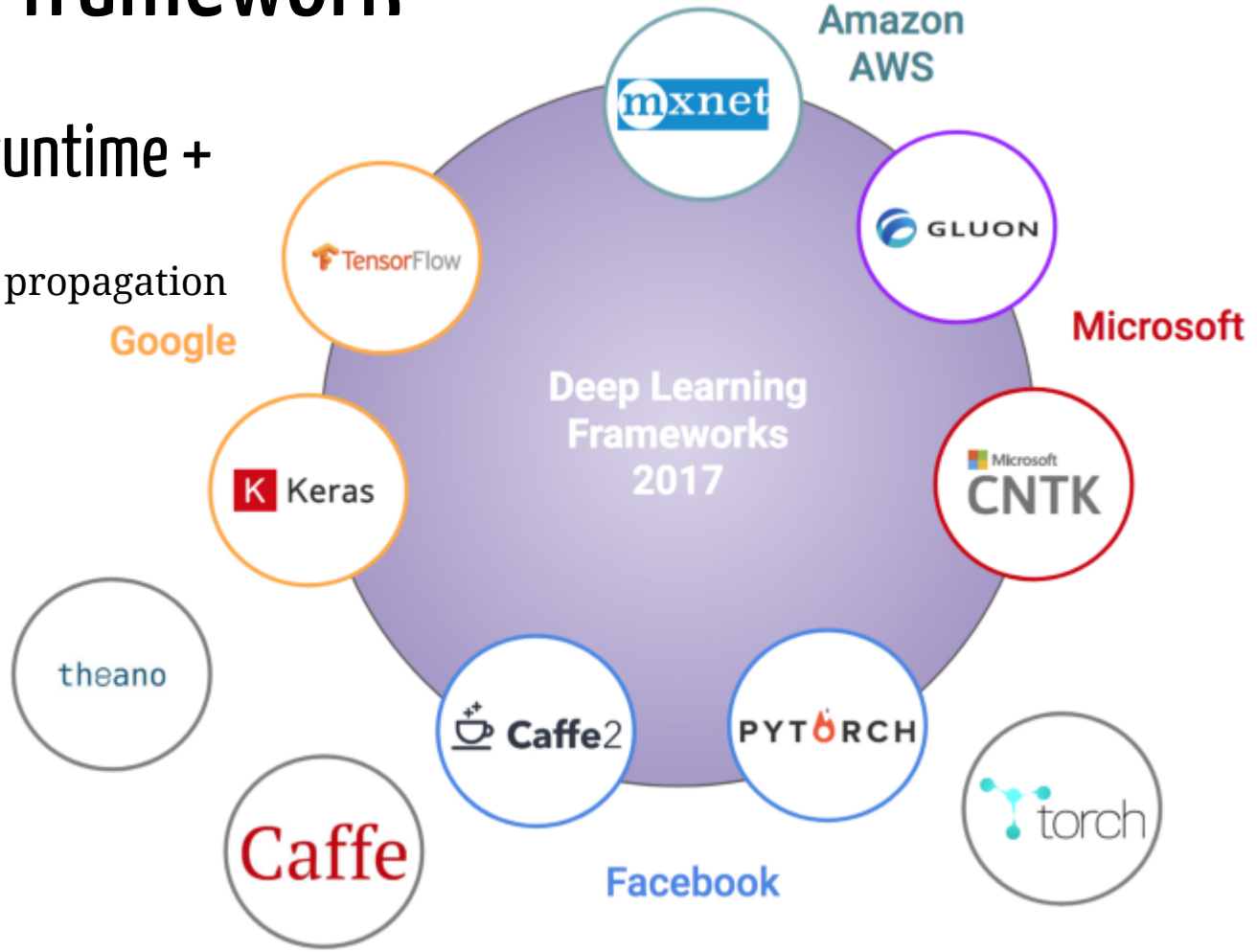
2 ML phases:


Learning
Inference
Big Data
Training
Big Model
Query
Decision
Application
Feedback

ML framework ==

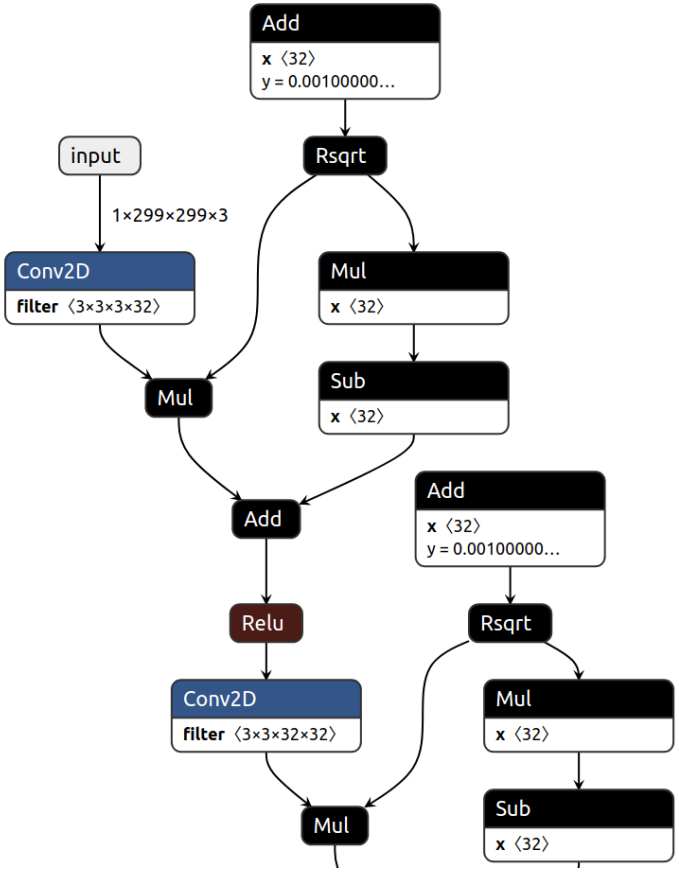
ML runtime +

back propagation



https://devopedia.org/deep-learning-frameworks

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

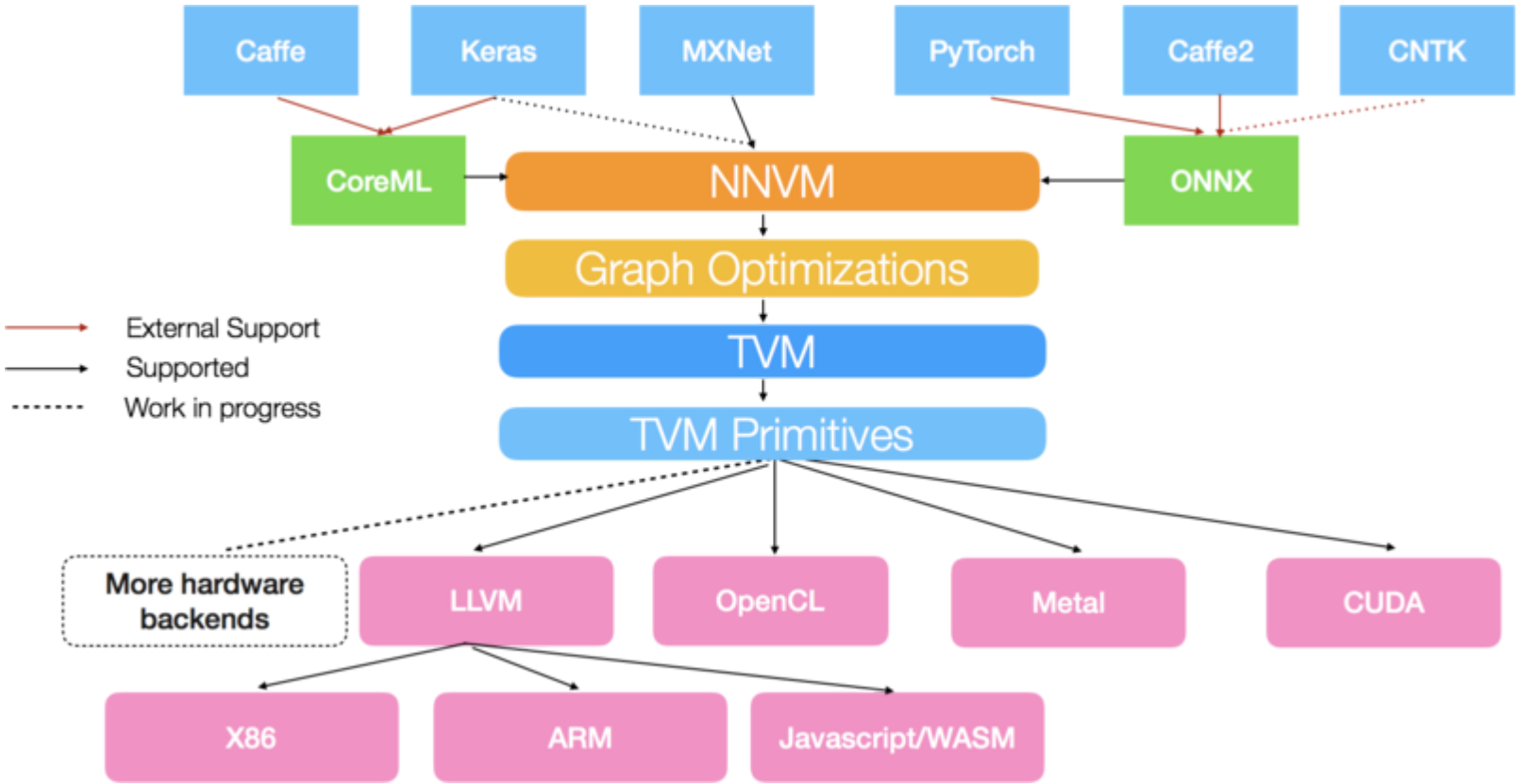
input	id: **InceptionV3/InceptionV3/Conv2d_1a_3x3/Relu**
filter	id: **InceptionV3/Conv2d_2a_3x3/weights/read**

OUTPUTS

output	id: **InceptionV3/InceptionV3/Conv2d_2a_3x3/convolution**

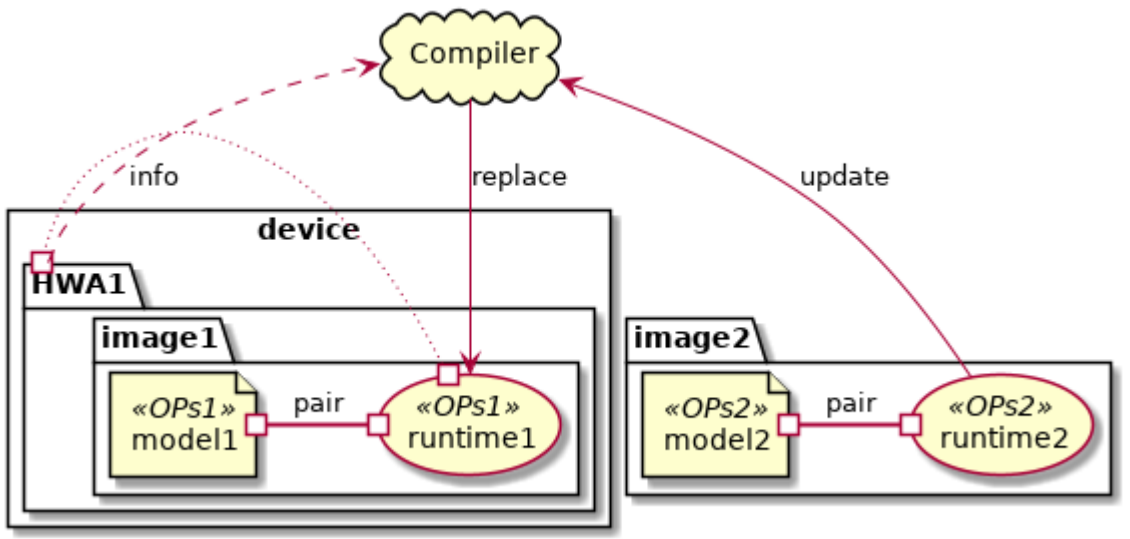
serialization: protobuf, flatbuf

Compiling

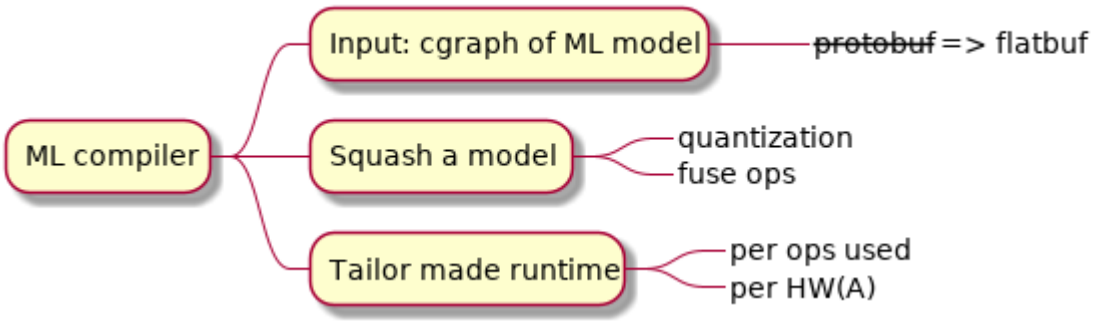

Caffe
Keras
MXNet
PyTorch
Caffe2
CNTK
CoreML
NNVM
ONNX
Graph Optimizations
TVM
TVM Primitives
External Support
Supported
Work in progress
More hardware backends
LLVM
OpenCL
Metal
CUDA
X86
ARM
Javascript/WASM

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.

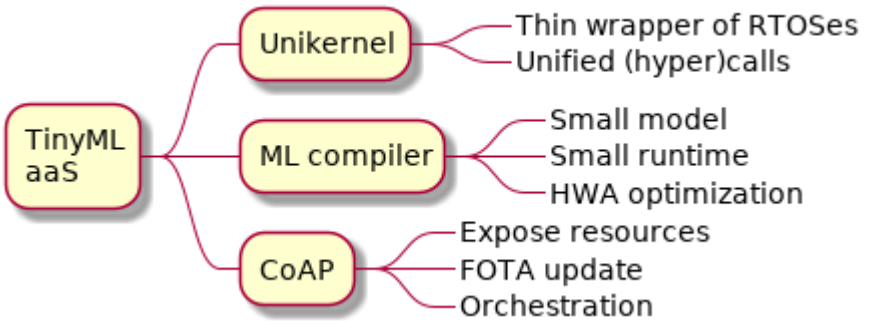


Summary: ML compiler



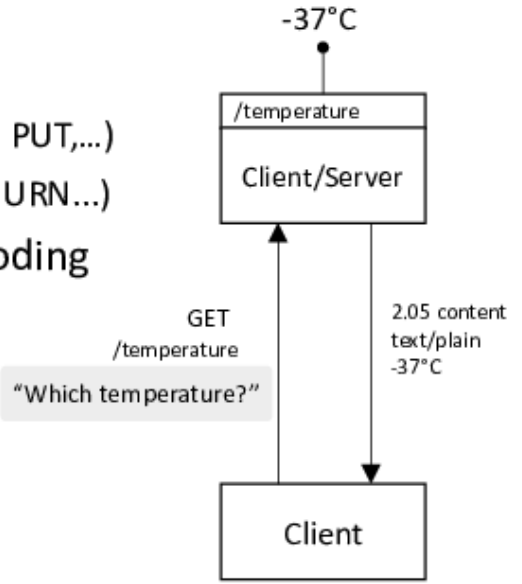
Outline

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

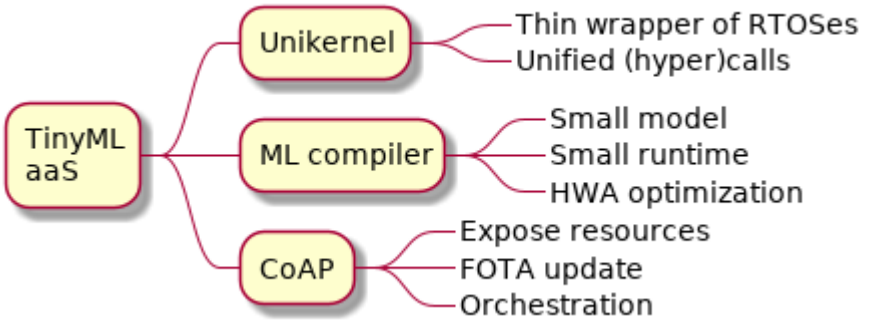


The **Co**nstrained **A**pplication **P**rotocol **(CoAP)**

- 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

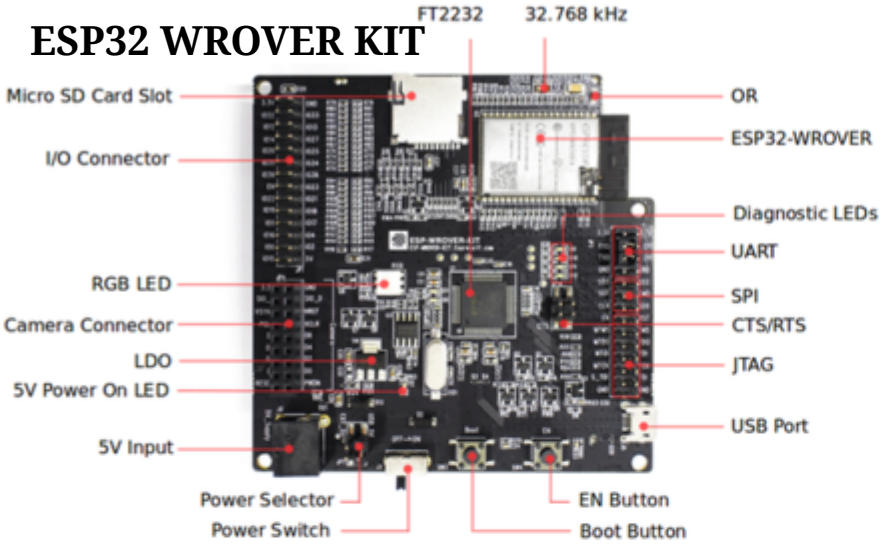
60K images for training, 10K for testing

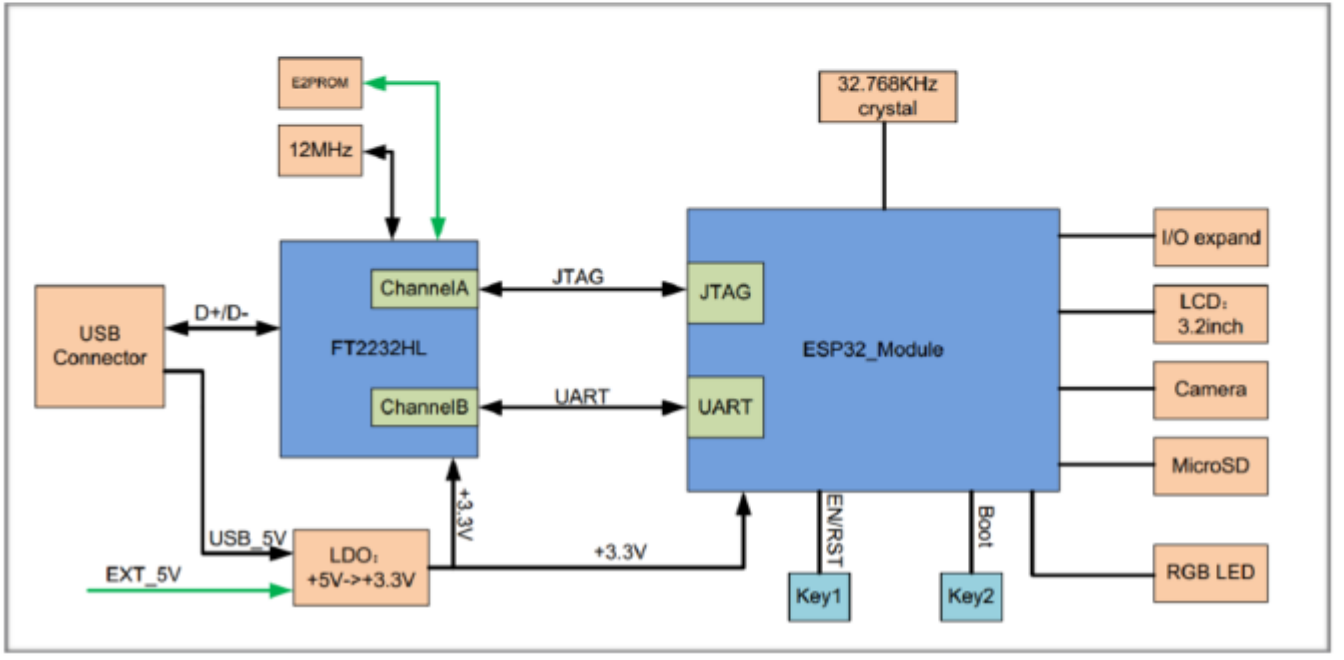
Data & Labels

0
1
2
3
4
5
6
7
8
9

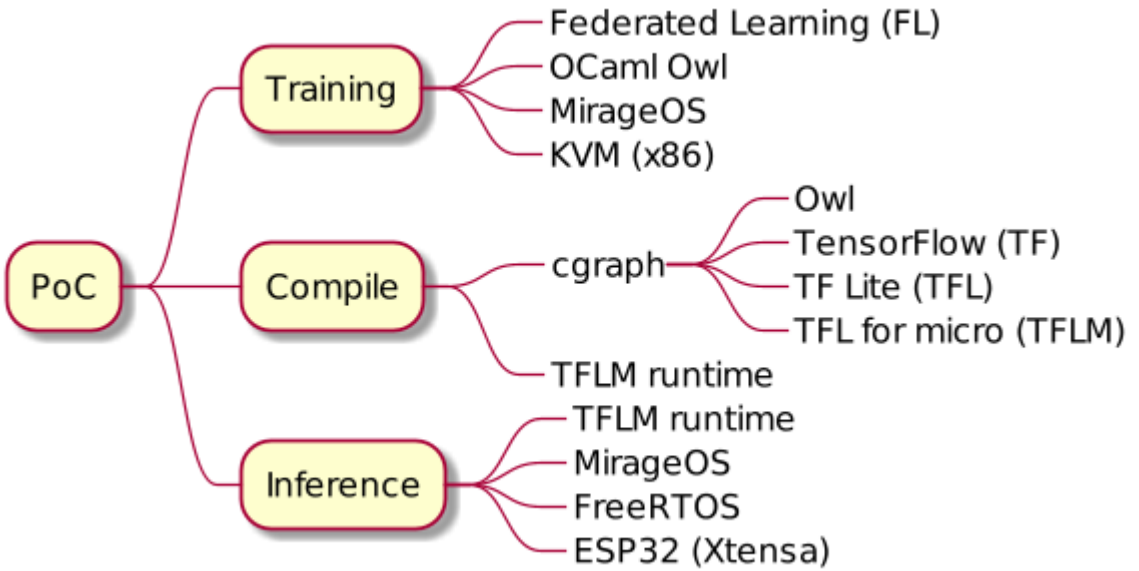
https://www.katacoda.com/basiafusinska/courses/tensorflow-getting-started/tensorflow-mnist-beginner

ESP32 WROVER KIT



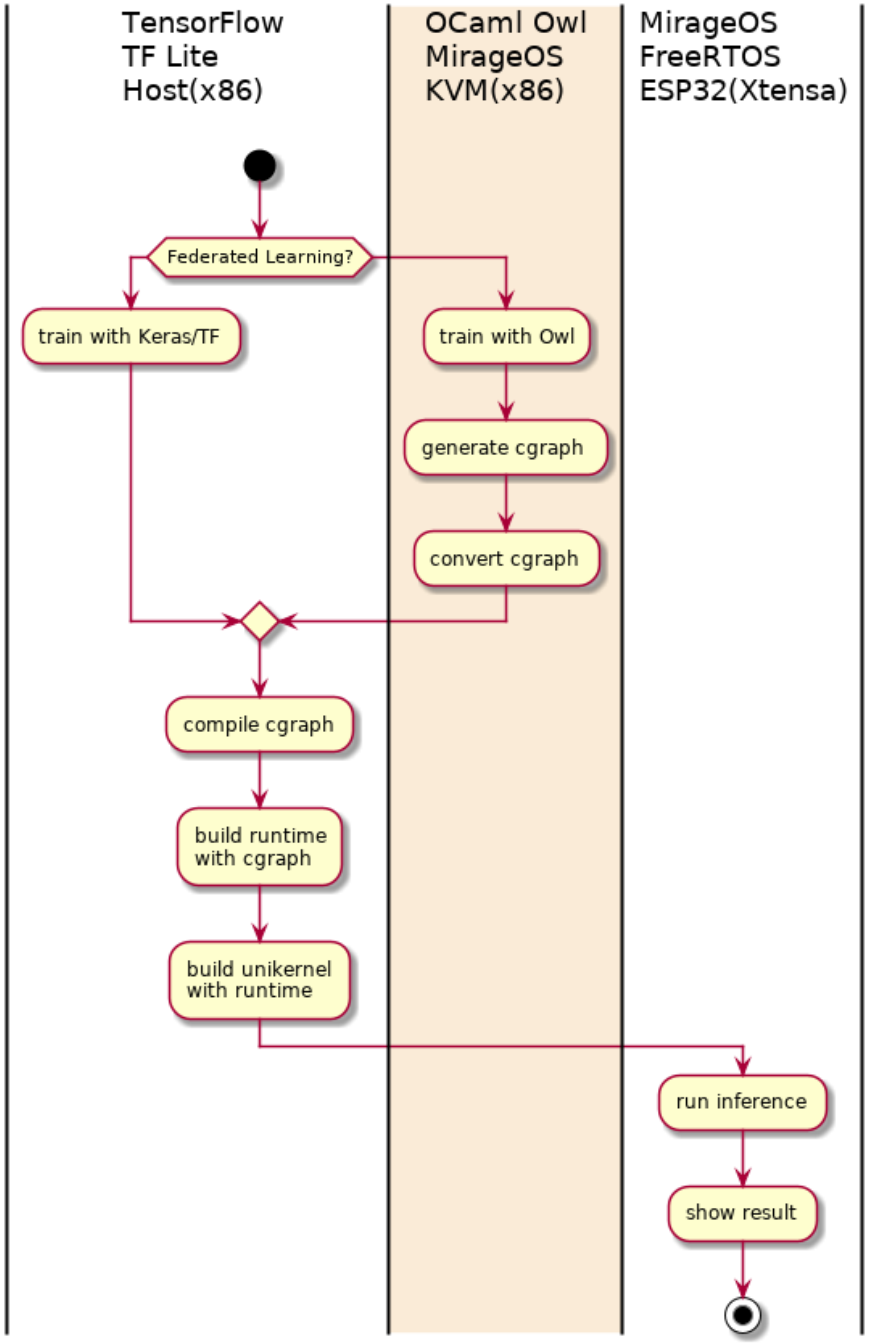


PoC Outline


PoC
Training
Federated Learning (FL)
OCaml Owl
MirageOS
KVM (x86)
Compile
cgraph
Owl
TensorFlow (TF)
TF Lite (TFL)
TFL for micro (TFLM)
TFLM runtime
Inference
TFLM runtime
MirageOS
FreeRTOS
ESP32 (Xtensa)

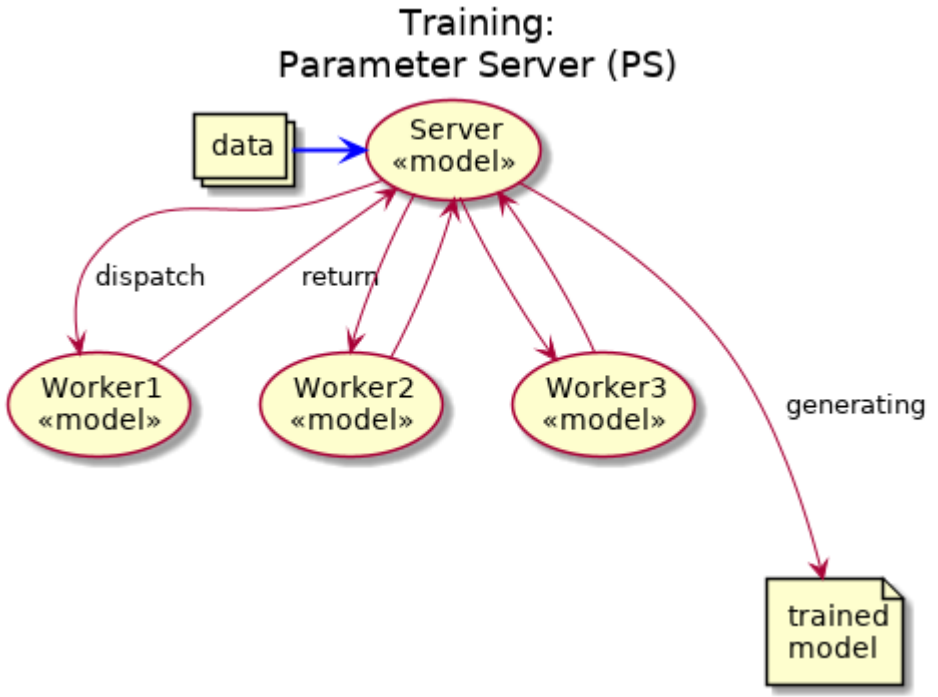
PoC

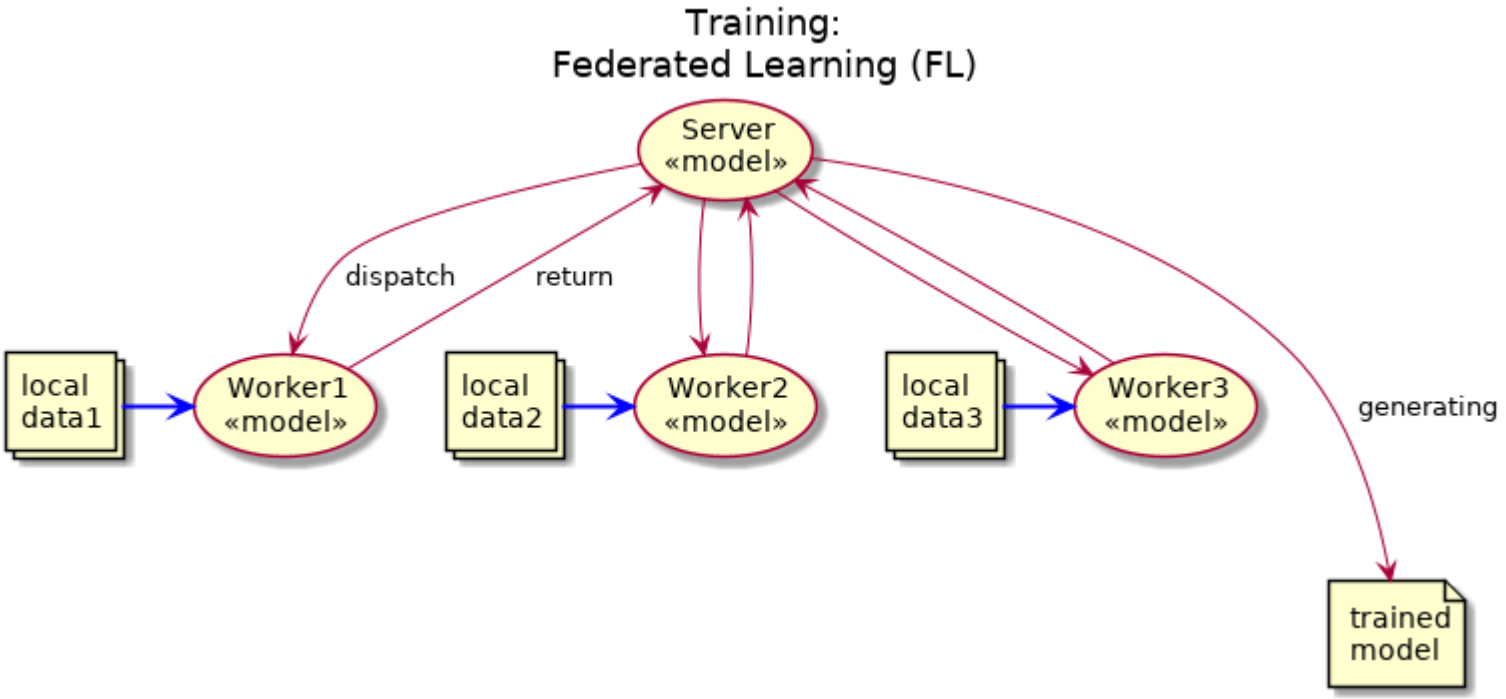
Sequence

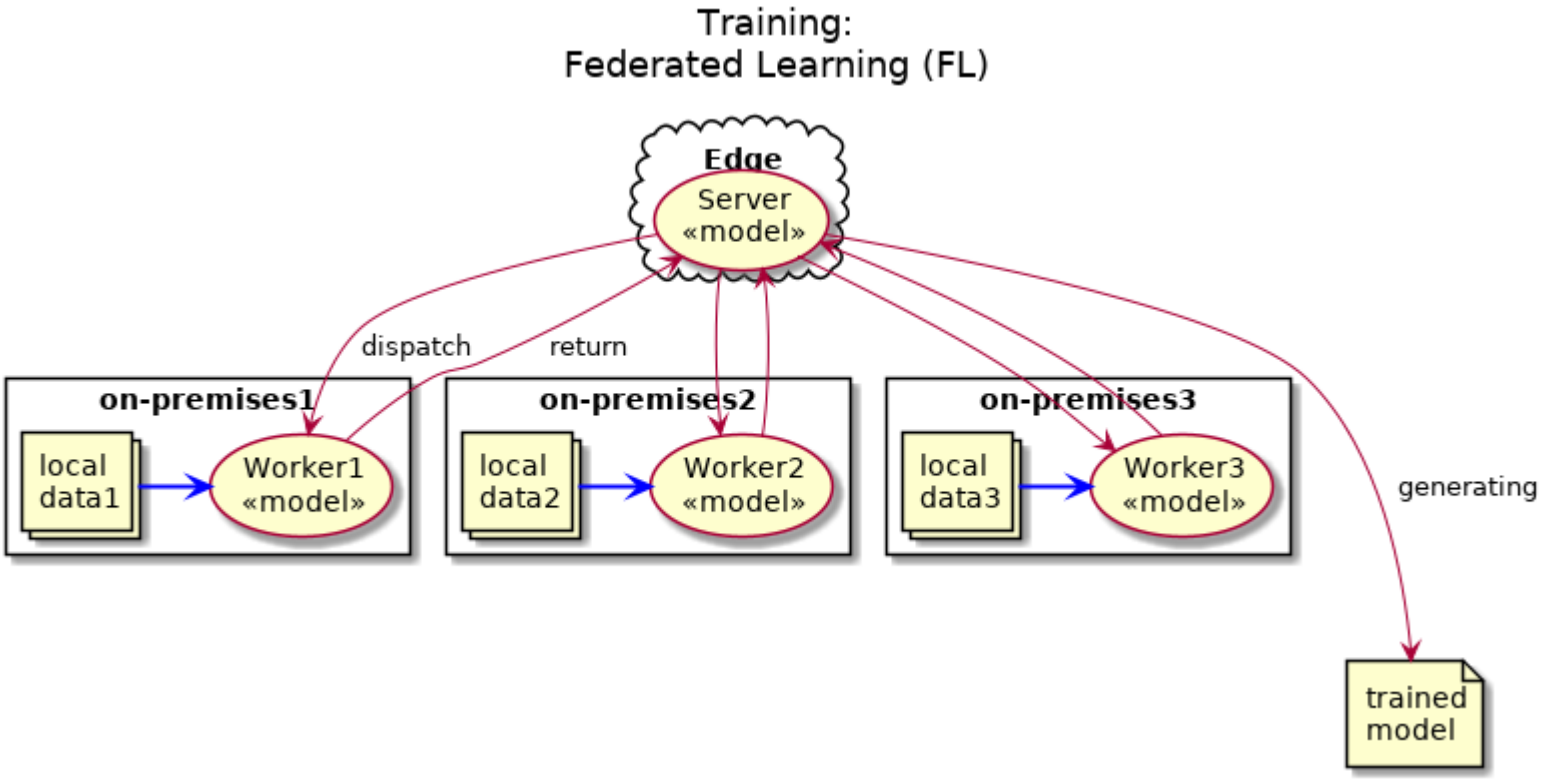


Outline

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

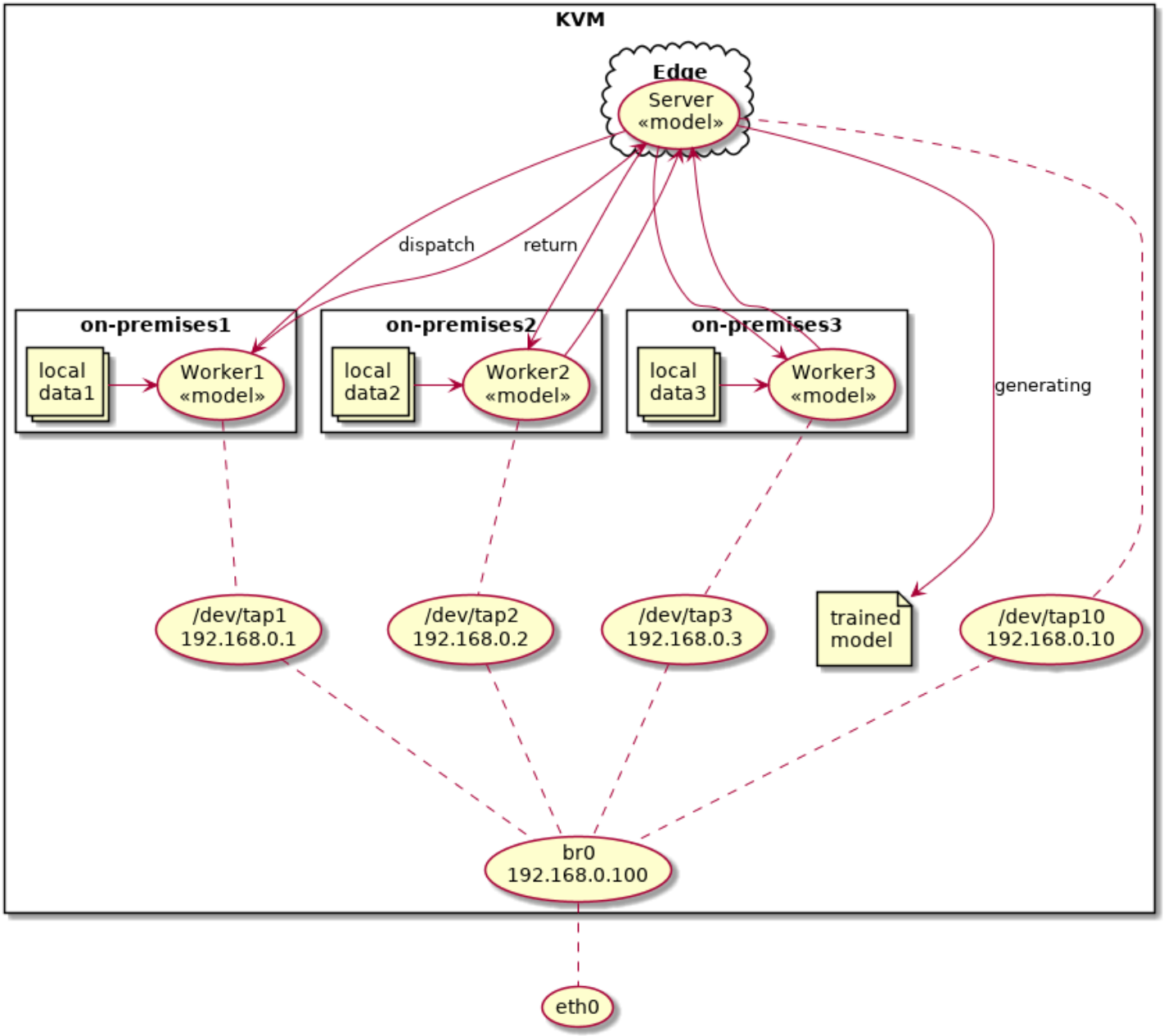

Training:
Parameter Server (PS)
data
Server
«model»
dispatch
return
Worker1
«model»
Worker2
«model»
Worker3
«model»
generating
trained
model


Training:
Federated Learning (FL)
Server
«model»
dispatch
return
local
data1
Worker1
«model»
local
data2
Worker2
«model»
local
data3
Worker3
«model»
generating
trained
model



sequence diagram

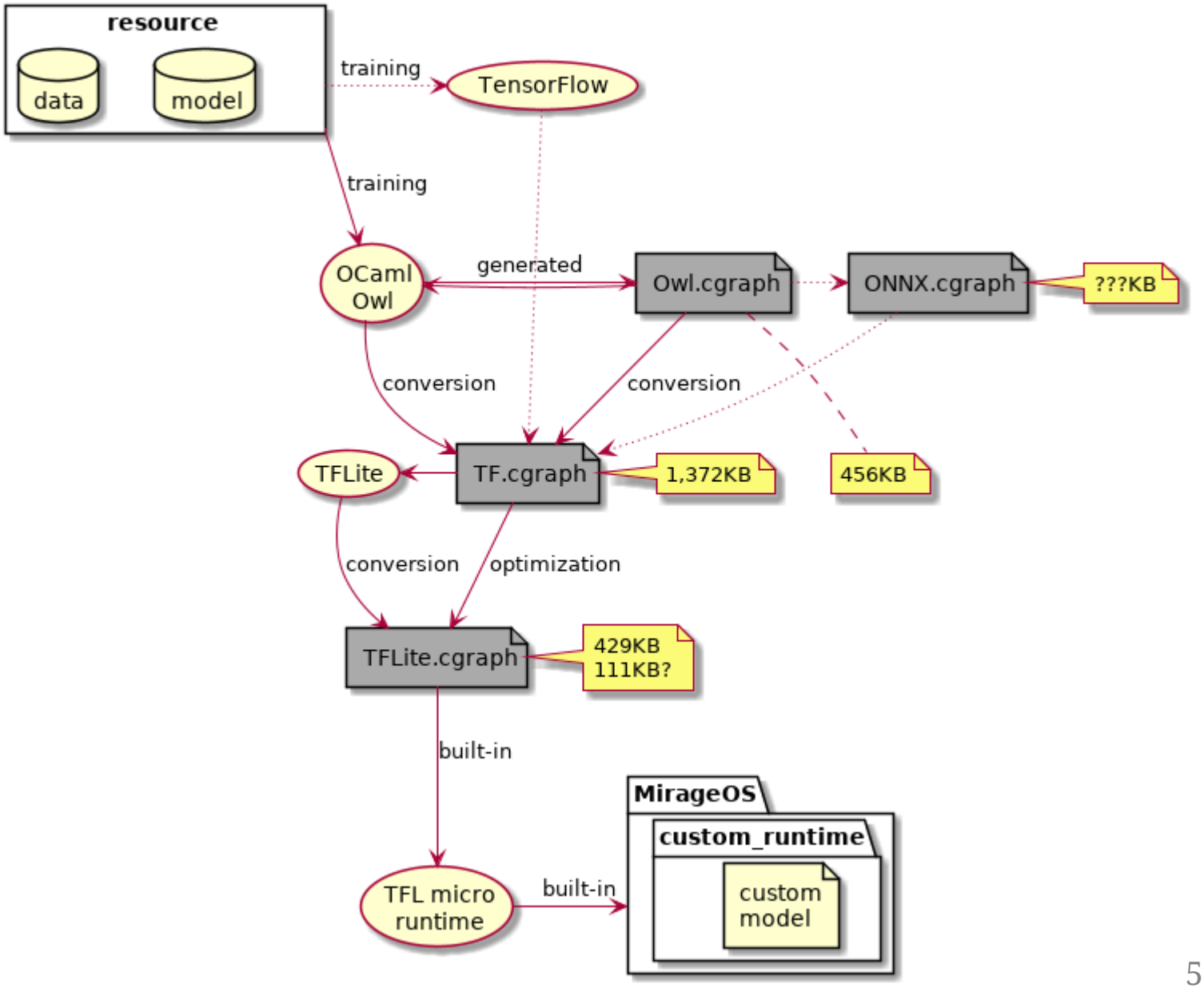
Training:
FL on KVM



Outline

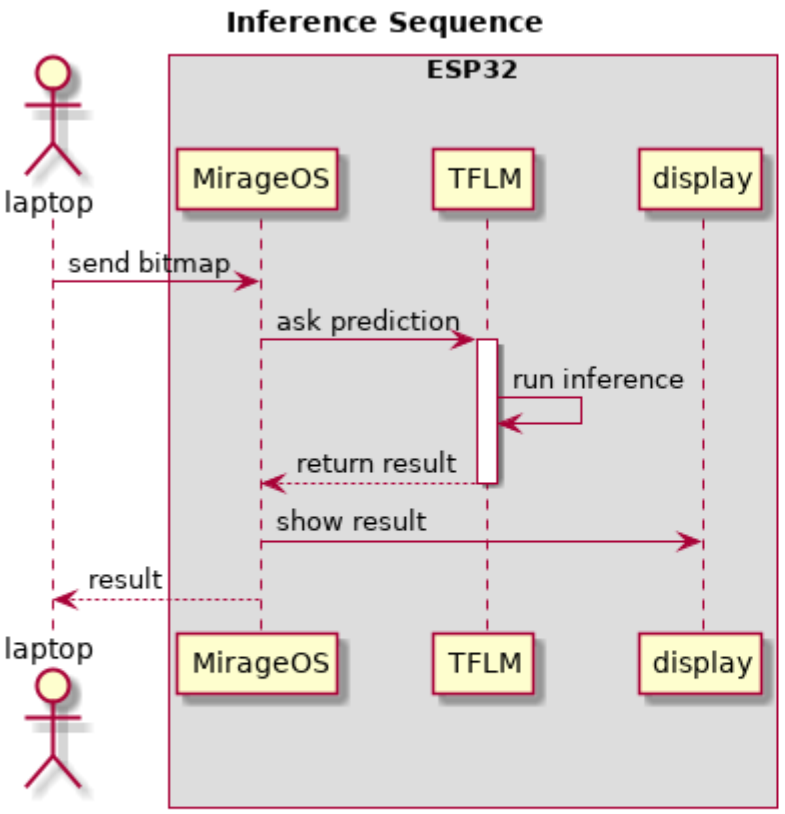
1. ~~Demo~~
2. ~~Problems~~
3. ~~Proposal~~
4. ~~Three Enablers~~
5. **PoC**
    - ~~Training~~
    - **Compile**
    - Inference
6. Conclusion

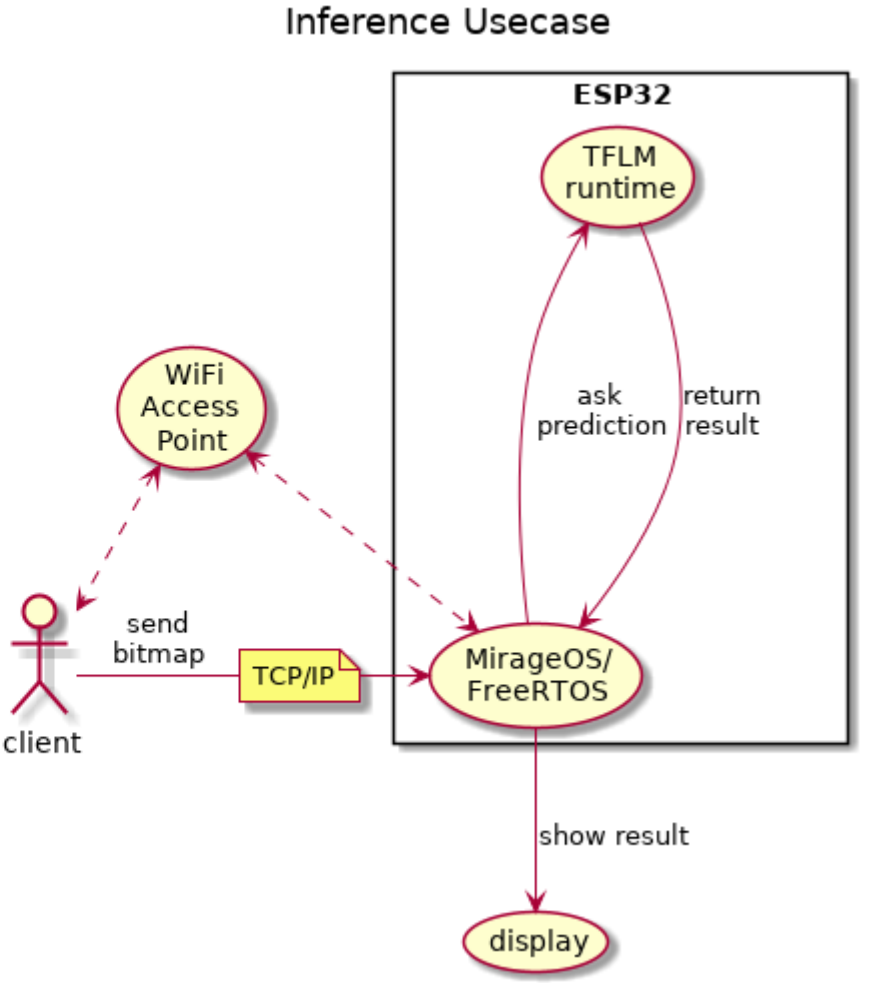
ML compiler: computation graph conversion



Outline

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


Inference Sequence
ESP32
laptop
MirageOS
TFLM
display
send bitmap
ask prediction
run inference
return result
show result
result
laptop
MirageOS
TFLM
display


Inference Usecase
ESP32
TFLM
runtime
ask
prediction
return
result
WiFi
Access
Point
send
bitmap
TCP/IP
MirageOS/
FreeRTOS
client
show result
display

Binary Size

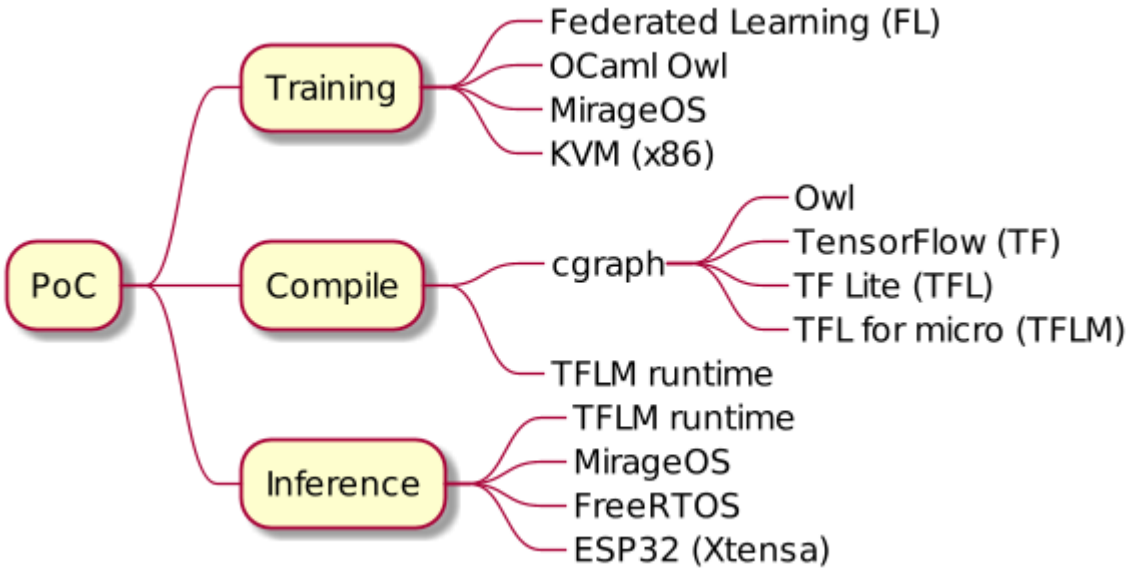
```
Total sizes:                                                          [35/93]
 DRAM .data size:   18784 bytes
 DRAM .bss  size:   62936 bytes
Used static DRAM:   81720 bytes (  99016 available, 45.2% used)
Used static IRAM:   91189 bytes (  39883 available, 69.6% used)
      Flash code: 1062913 bytes
    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**

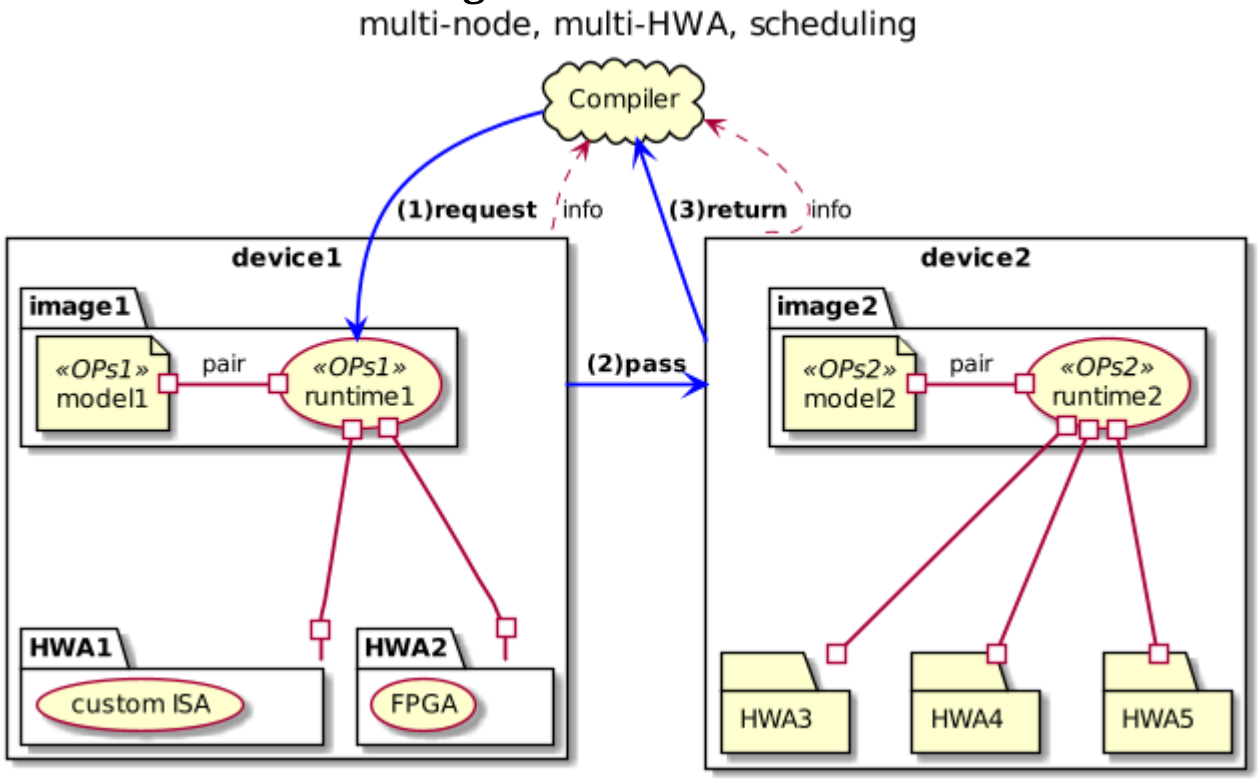
Done

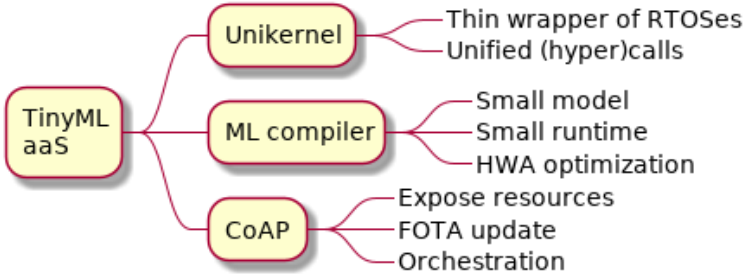
Demonstrated End-to-End **TinyML** (as-a-Service?)

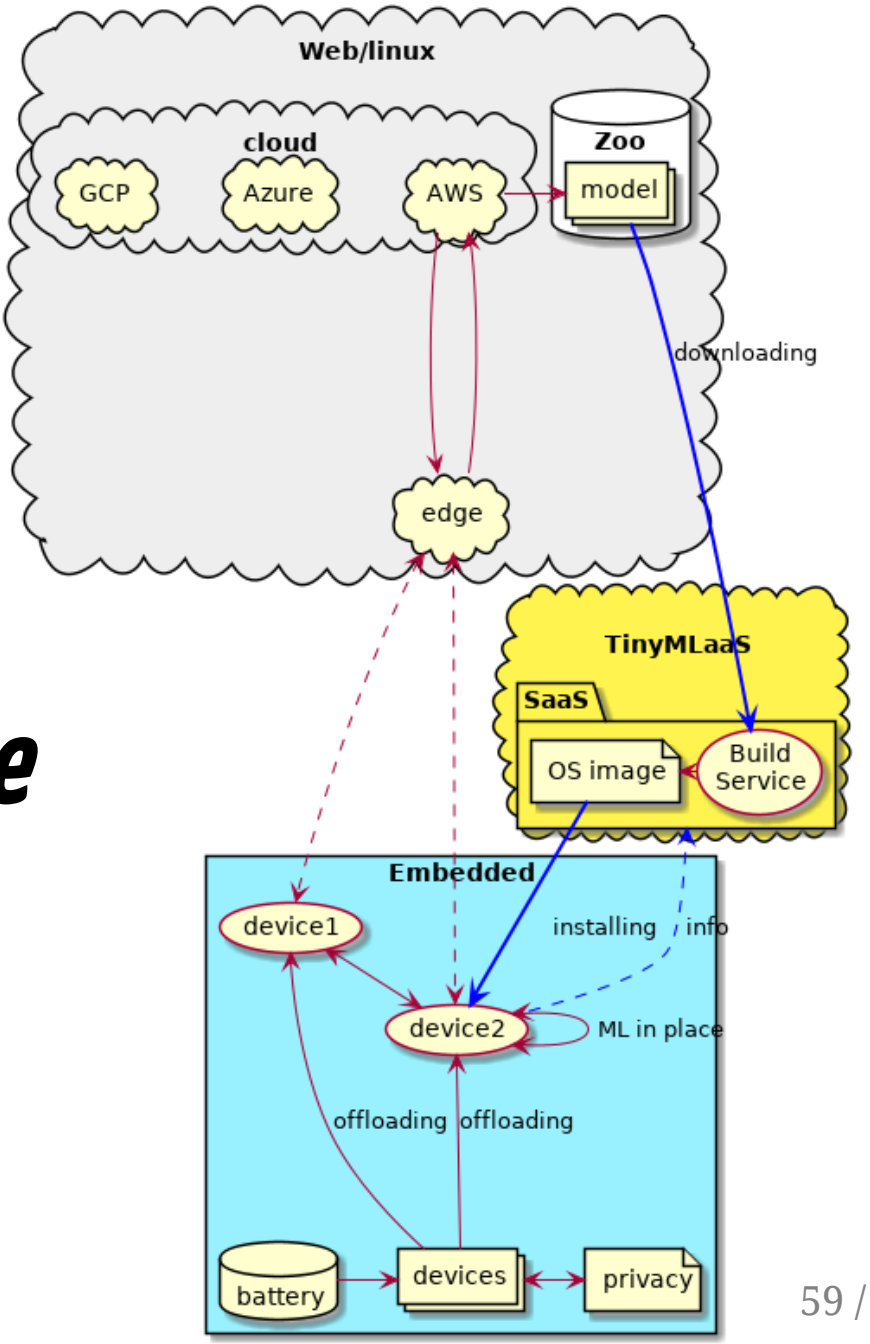


Next

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







TinyML as-a-Service could bring ML onto IoT.

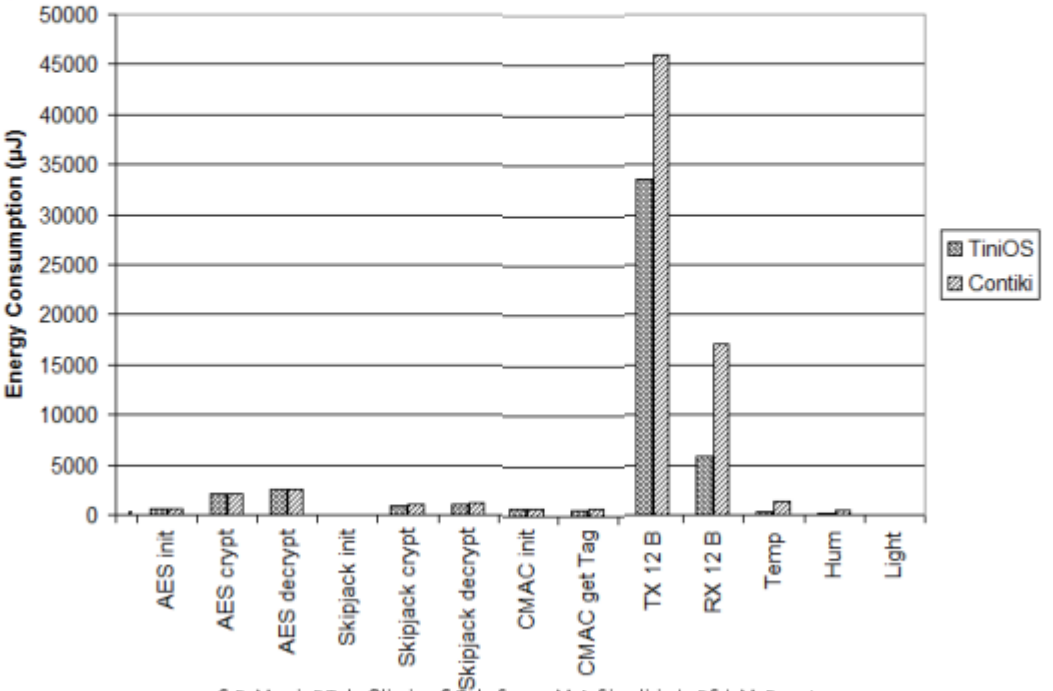
ERICSSON

Appendix

Energy consumption on TelosB

Garrulity

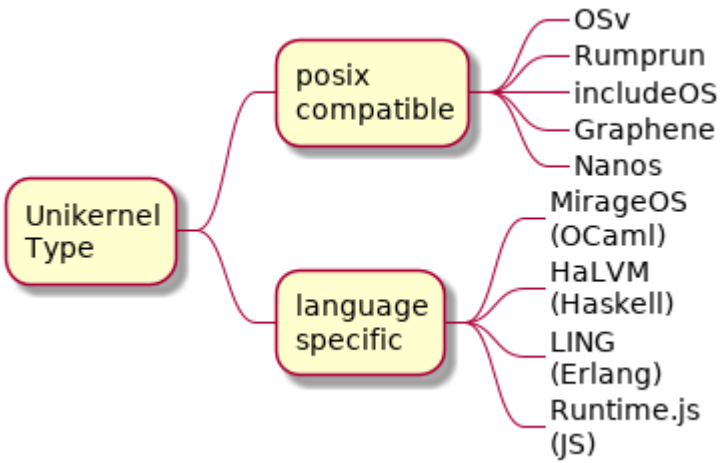
Message exchange cost: orders of magnitude more than processing, symmetric crypto



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

34

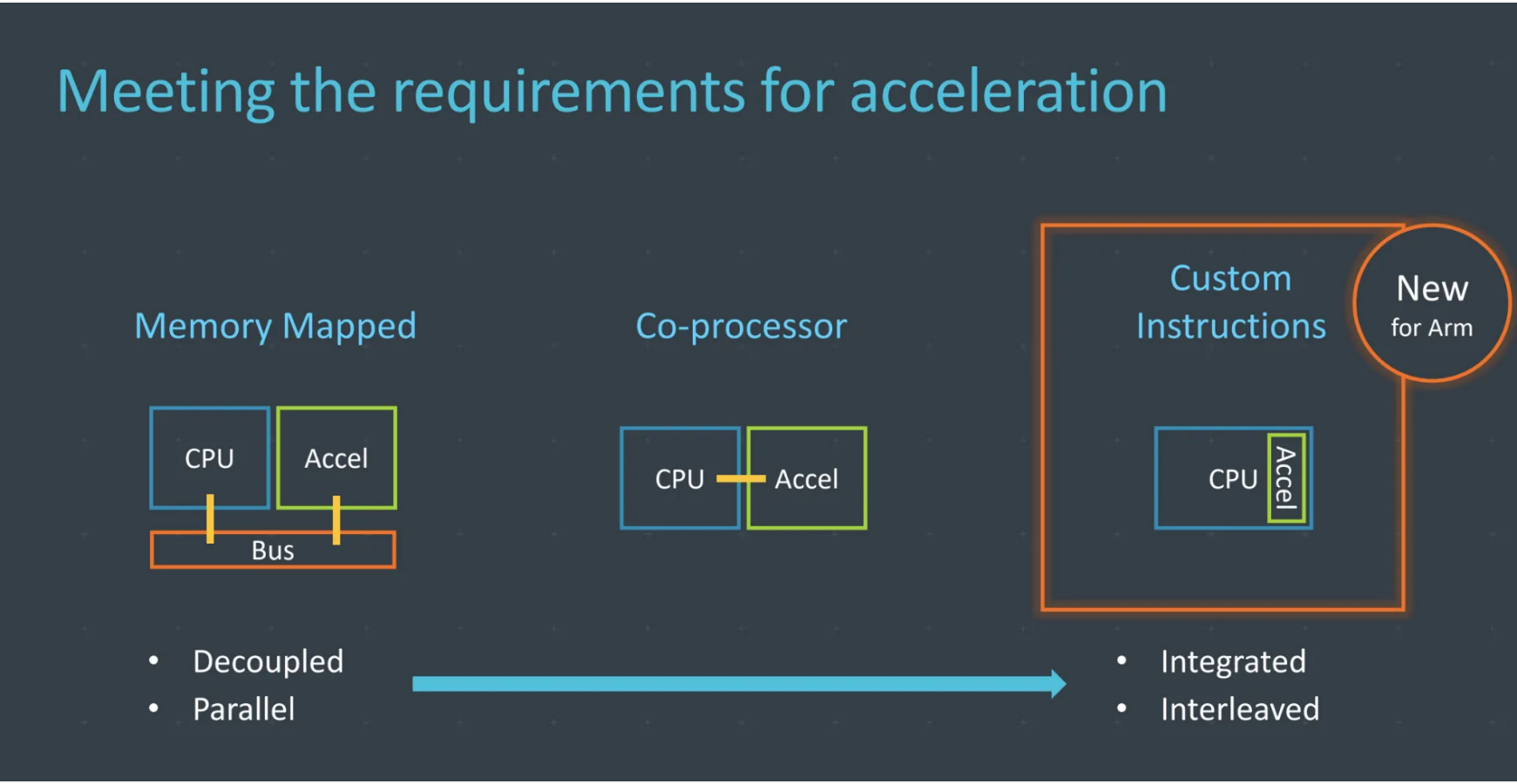
Type of Unikernel



MirageOS in OCaml

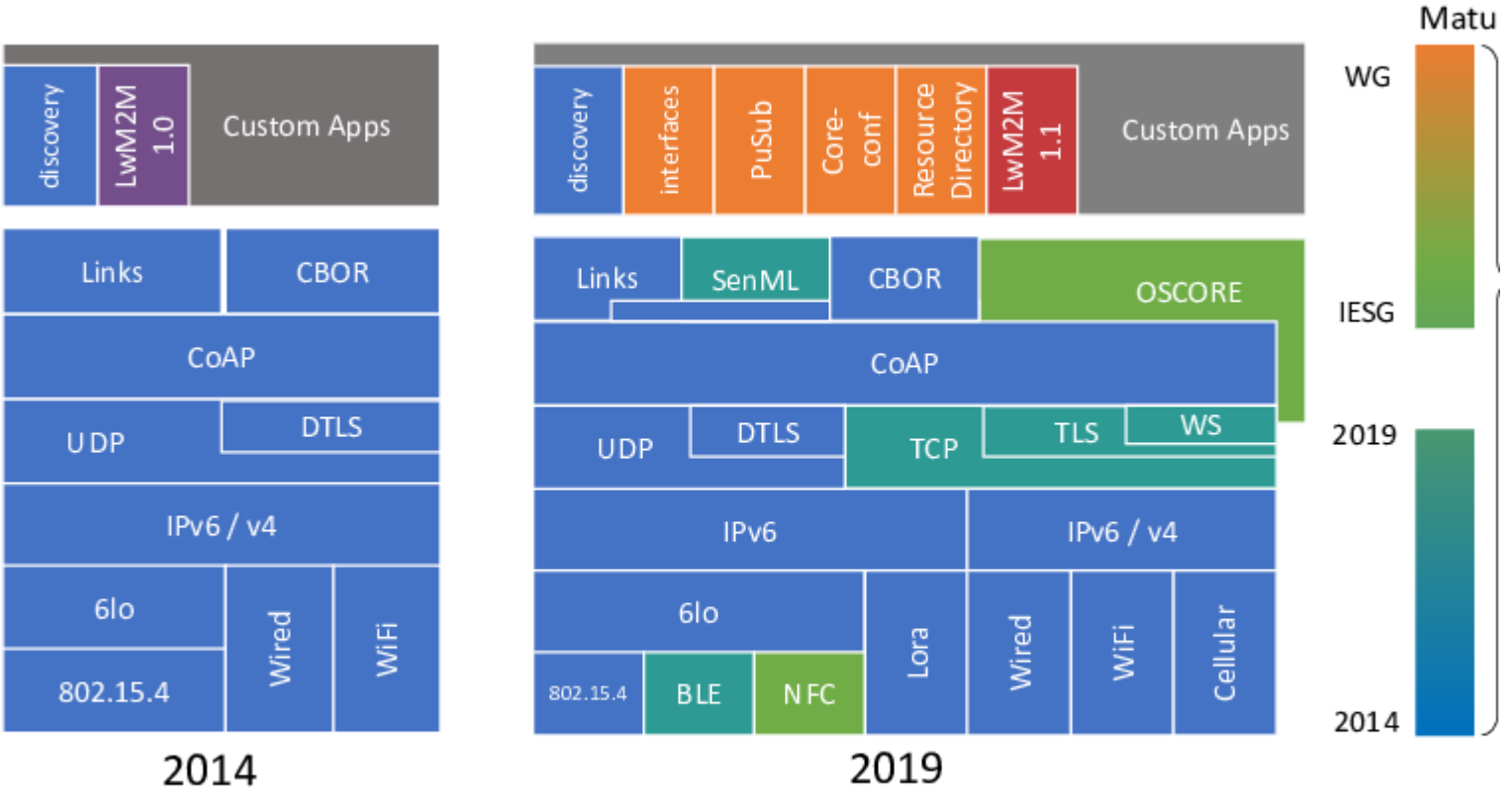
```
open Lwt.Infix

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-cdddd57  (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



https://staceyoniot.com/why-arm-opened-up-its-instruction-set-and-what-it-means-for-iot/

Standards Device Stack



13/2/19

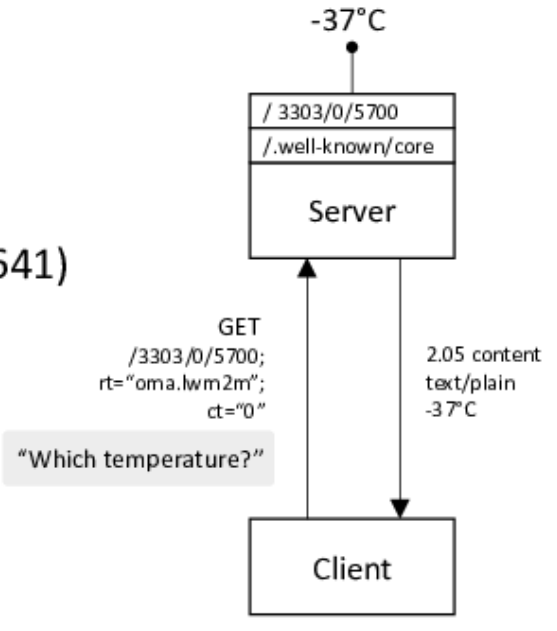
One Data Model Group

The **Co**nstrained **A**pplication **P**rotocol **(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

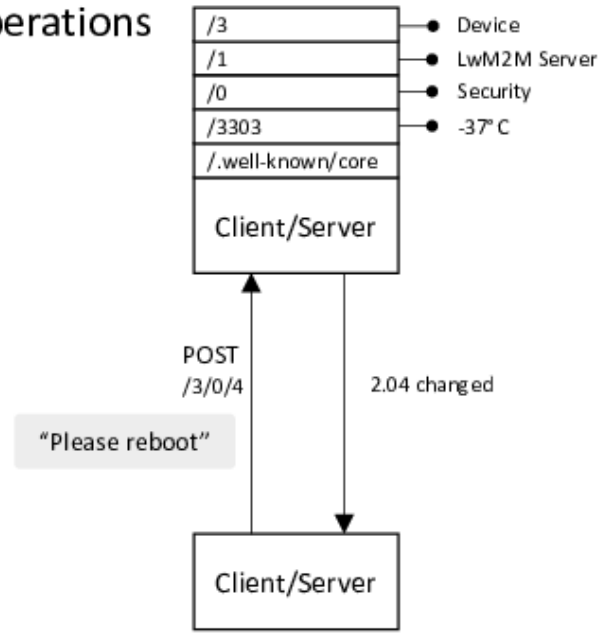


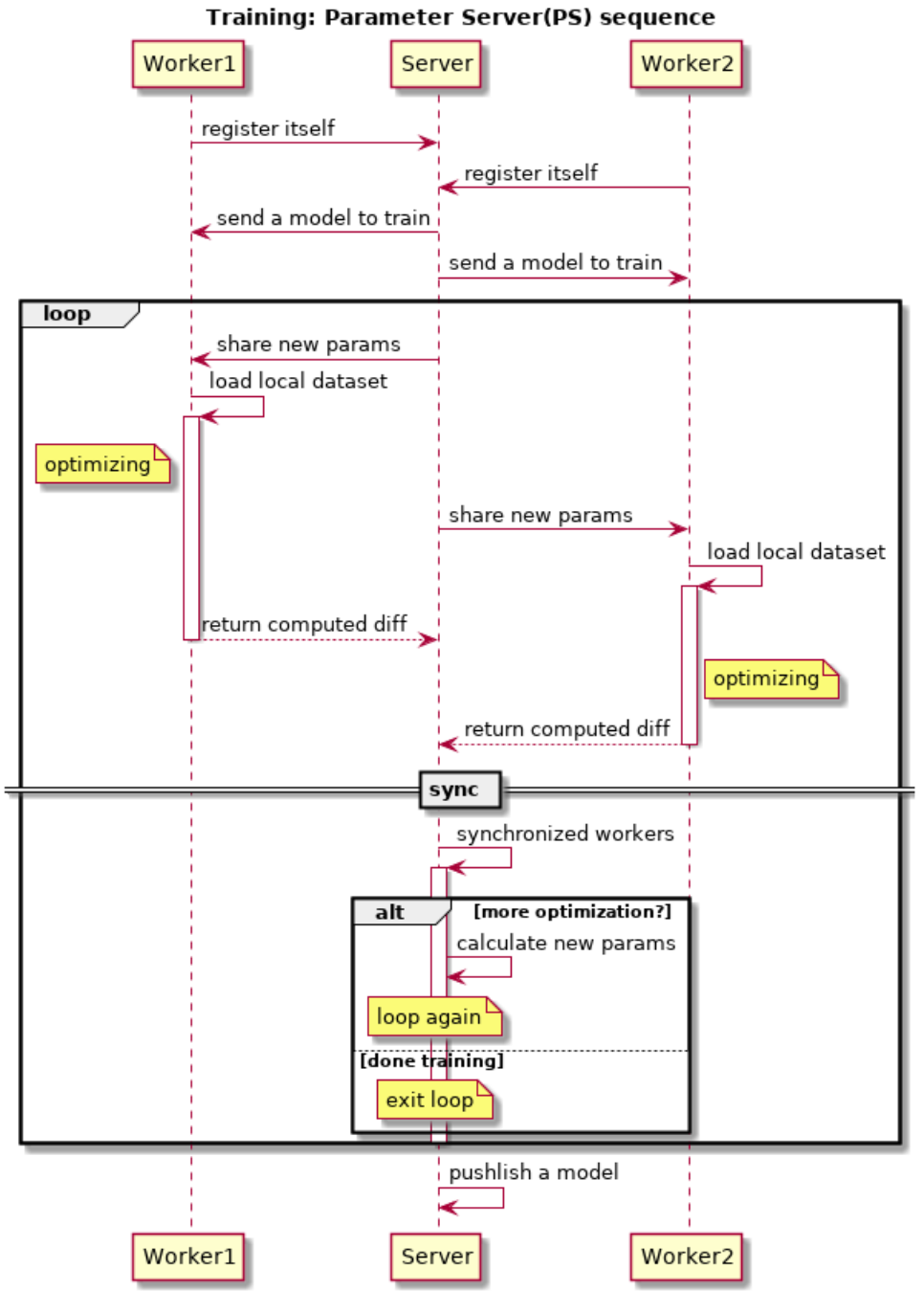
The LightWeight **M2M** Protocol **(LwM2M)**

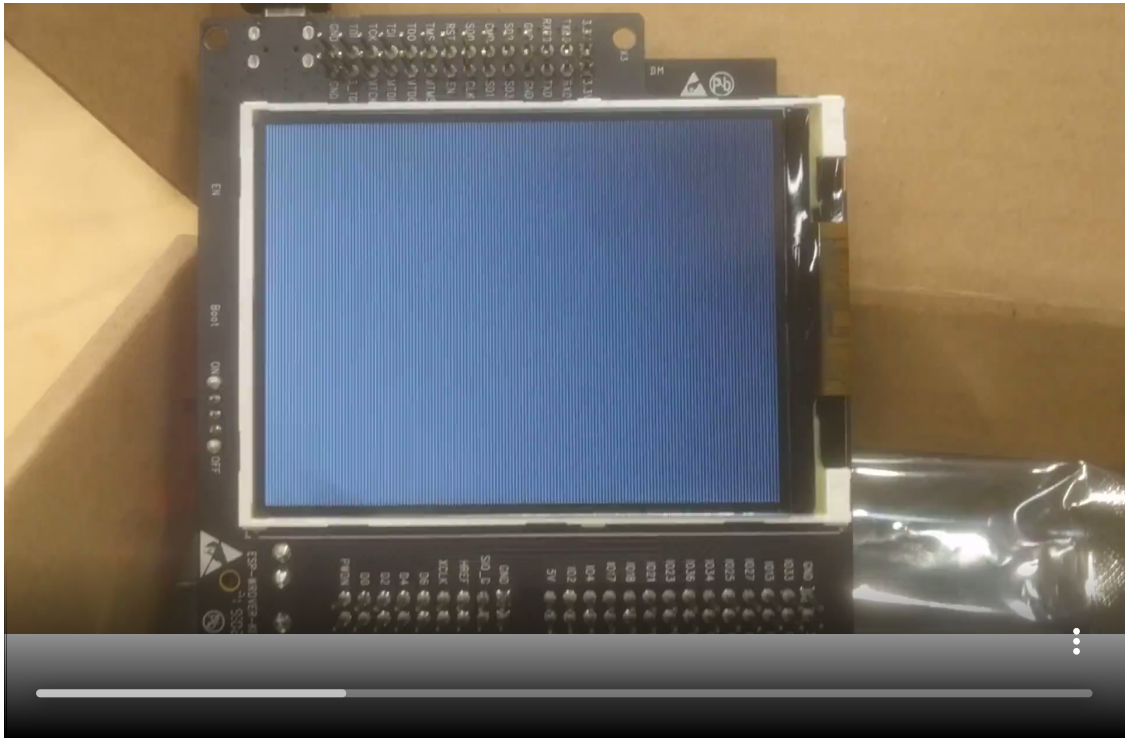
- Mapping of CoAP methods (GET,POST, PUT…) to **CRUD** operations
- Interaction with device through simple "Objects"
  - RWX, Access Control, Observation, Notification
  - Independent from underlying protocol stack (CoAP today)
  - Simple resource structure
  - Objects' resources are accessed with simple 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
Worker1
Server
Worker2
register itself
register itself
send a model to train
send a model to train
loop
share new params
load local dataset
optimizing
share new params
load local dataset
return computed diff
optimizing
return computed diff
sync
synchronized workers
alt
[more optimization?]
calculate new params
loop again
[done training]
exit loop
pushlish a model
Worker1
Server
Worker2



Fashion MNIST from Zalando