WebAssembly Micro Runtime and use cases for IoT and PC

https://github.com/intel/wasm-micro-runtime

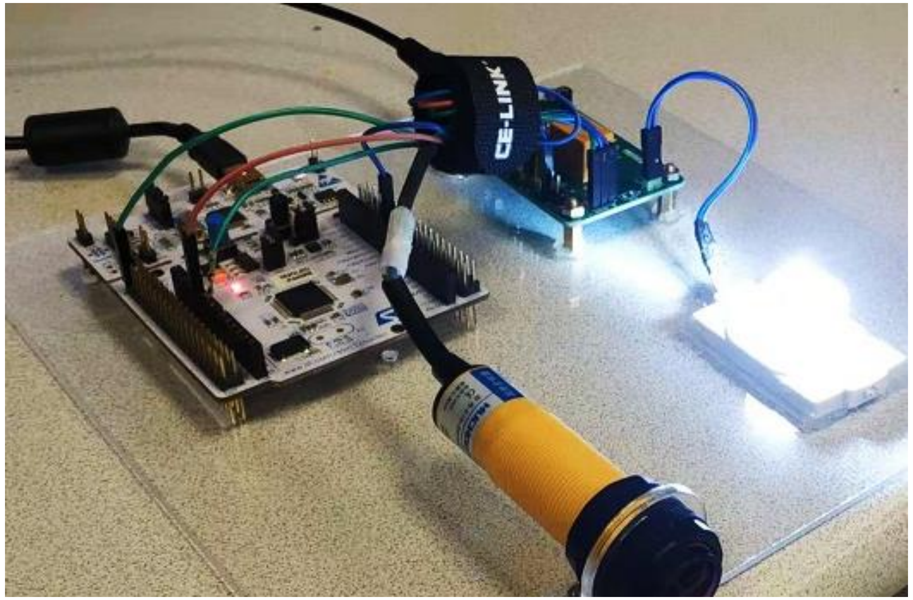
Xin Wang (xin.wang@intel.com)

Intel Corporation

8/9/2019

Webassembly micro runtime (WAMR) overview

- Open sourced standalone WASM runtime for device
- Interpreter currently and AoT in the plan
- Designed for small footprint
  - 84K for VMCore
  - 130K for the whole runtime (+API, remote app mgt)
- Support Linux, Zephyr, VxWorks, AliOS Things already



STM32 board with 128K SRAM and 512K FLASH

WAMR runs on embedded OS

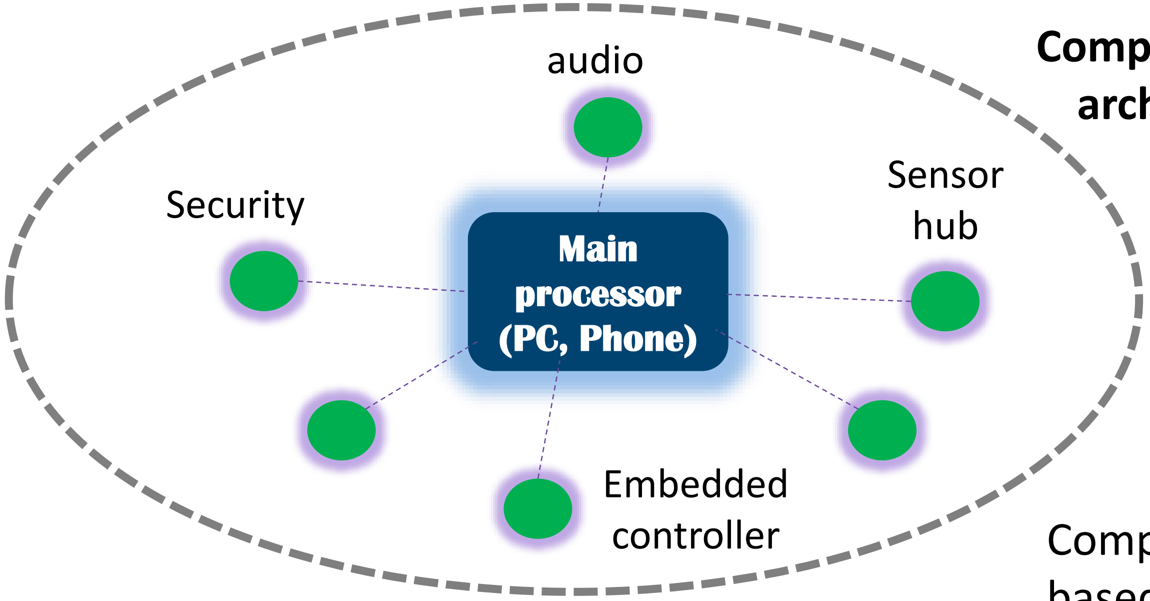
Wasm Application runs on WAMR:

- Senses presence of human and turns light on/off accordingly

*Other names and brands may be claimed as the property of others.

Project motivations

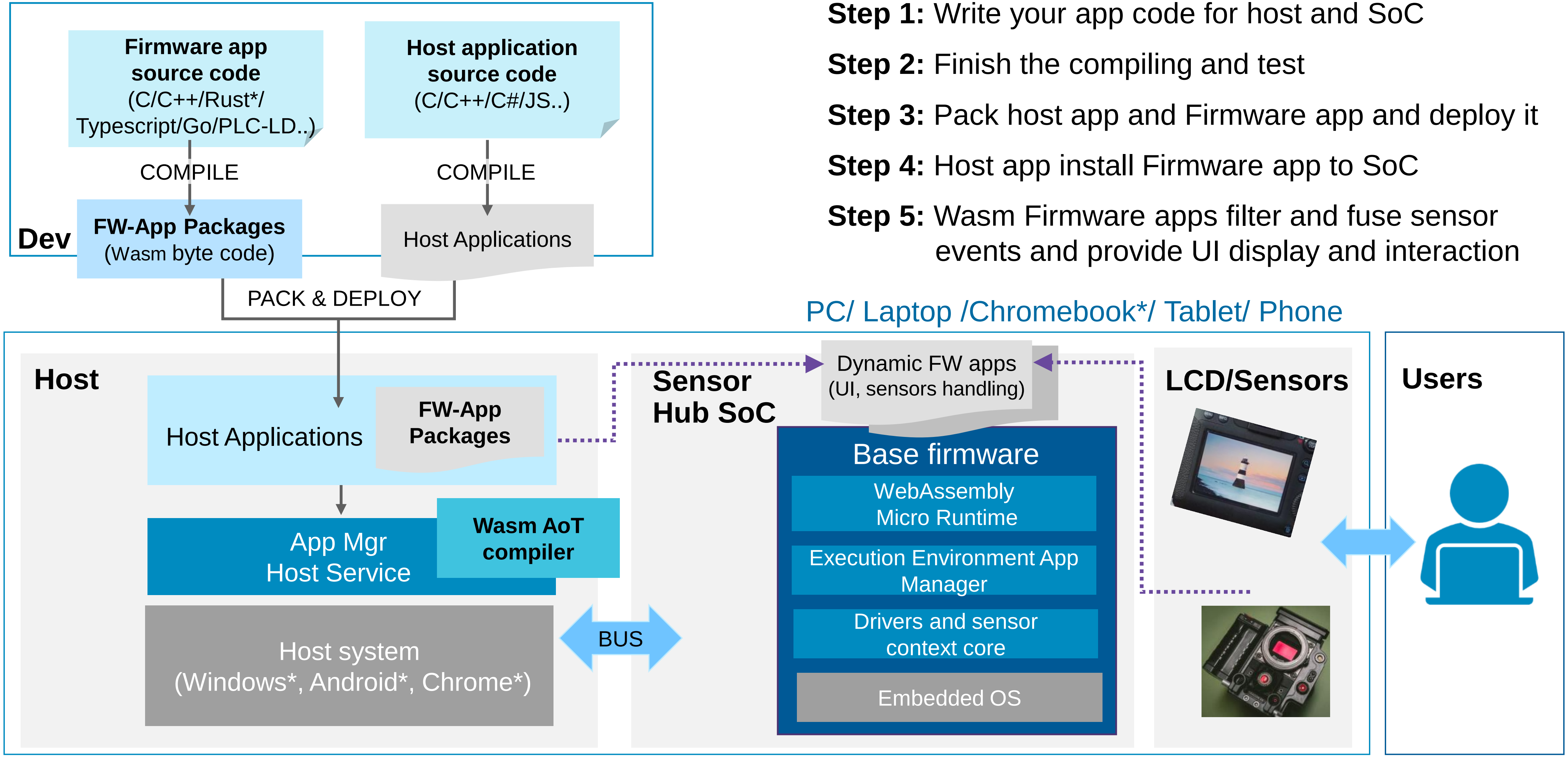
- Enable dynamic firmware applications for companion chip architectures and IoT devices
- Support open development environments for OEM/ISV to continuously innovate use cases on the hardware capabilities cross the product lifecycle



Companion chip architecture

Companion chips usually are MCU based and run embedded OS

Companion chip use case working flow



Firmware app source code (C/C++/Rust*/ Typescript/Go/PLC-LD..)
Host application source code (C/C++/C#/JS..)
COMPILE
COMPILE
Dev
FW-App Packages (Wasm byte code)
Host Applications
PACK & DEPLOY
PC/ Laptop /Chromebook*/ Tablet/ Phone
Host
Host Applications
FW-App Packages
App Mgr Host Service
Wasm AoT compiler
Host system (Windows*, Android*, Chrome*)
BUS
Sensor Hub SoC
Dynamic FW apps (UI, sensors handling)
Base firmware
WebAssembly Micro Runtime
Execution Environment App Manager
Drivers and sensor context core
Embedded OS
LCD/Sensors
Users

Step 1: Write your app code for host and SoC

Step 2: Finish the compiling and test

Step 3: Pack host app and Firmware app and deploy it

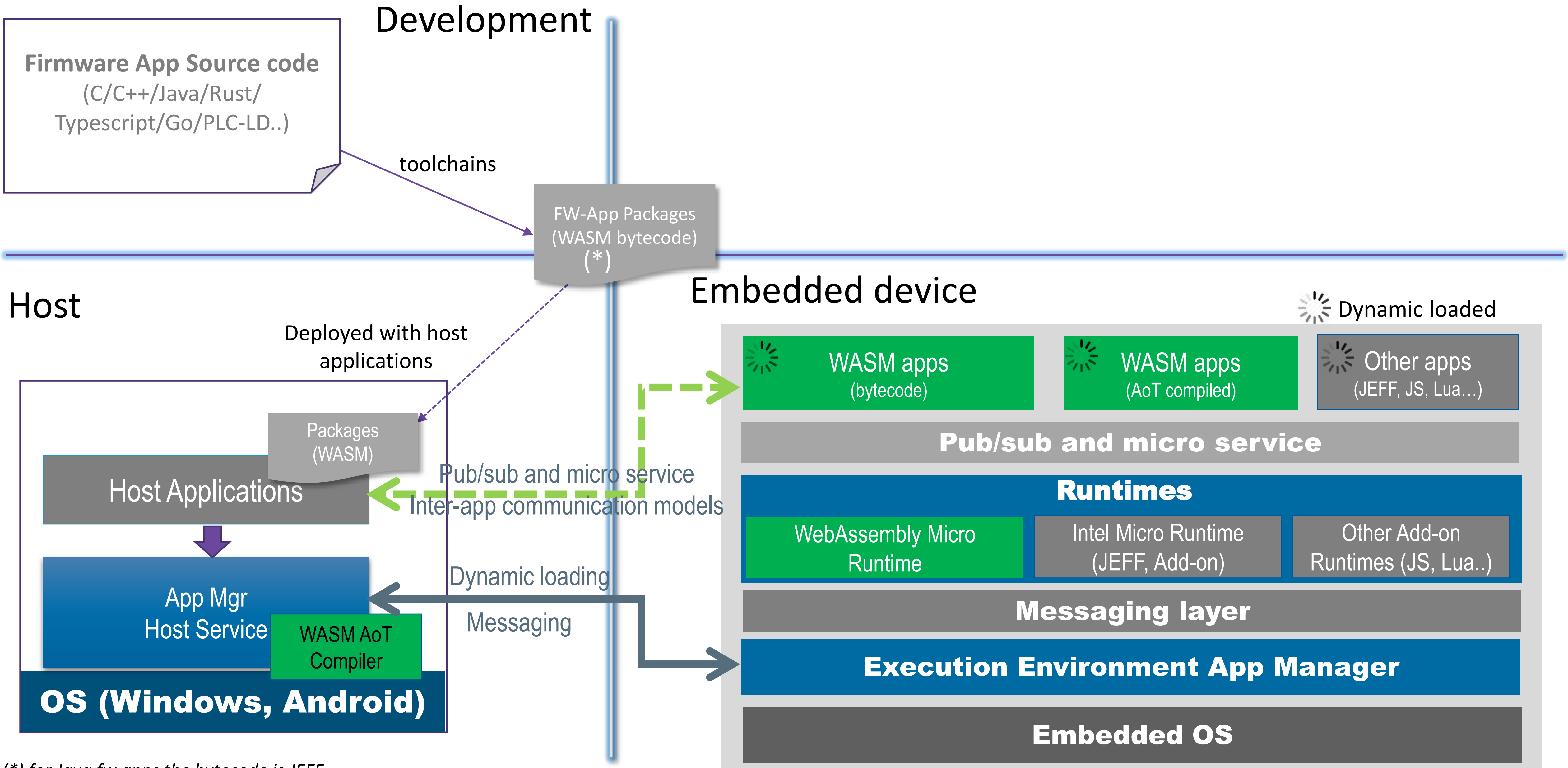
Step 4: Host app install Firmware app to SoC

Step 5: Wasm Firmware apps filter and fuse sensor events and provide UI display and interaction

*Other names and brands may be claimed as the property of others.

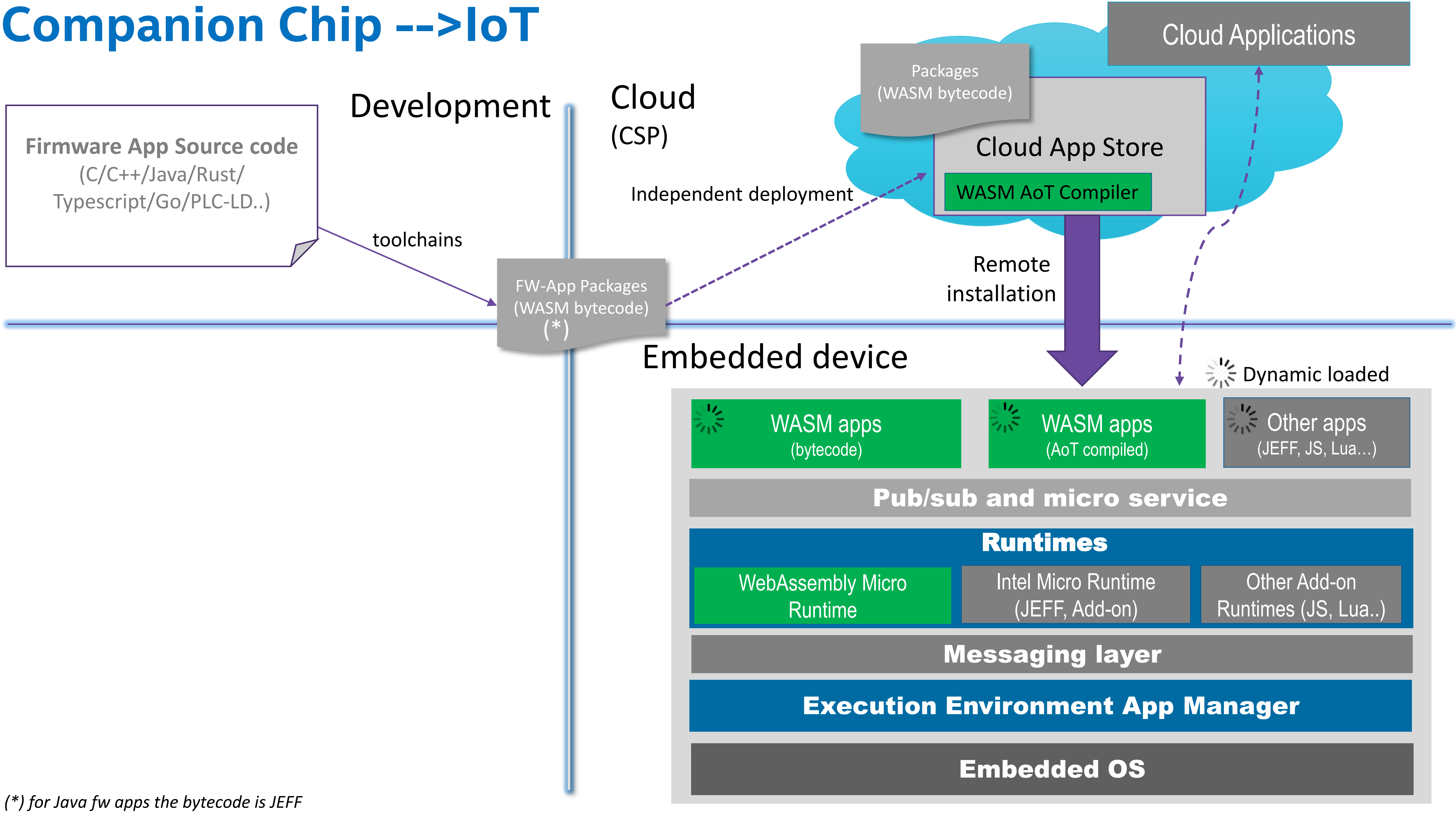
Top photo by Neil Thomas on Unsplash.
Bottom photo by Jakob Owens on Unsplash

Companion Chip -->IoT



(*) for Java fw apps the bytecode is JEFF

Companion Chip -->IoT



Development

Cloud (CSP)

Embedded device

Firmware App Source code
(C/C++/Java/Rust/
Typescript/Go/PLC-LD..)

toolchains

FW-App Packages
(WASM bytecode)
(*)

Independent deployment

Packages
(WASM bytecode)

Cloud App Store

WASM AoT Compiler

Cloud Applications

Remote installation

Dynamic loaded

WASM apps
(bytecode)

WASM apps
(AoT compiled)

Other apps
(JEFF, JS, Lua...)

Pub/sub and micro service

Runtimes

WebAssembly Micro Runtime

Intel Micro Runtime (JEFF, Add-on)

Other Add-on Runtimes (JS, Lua..)

Messaging layer

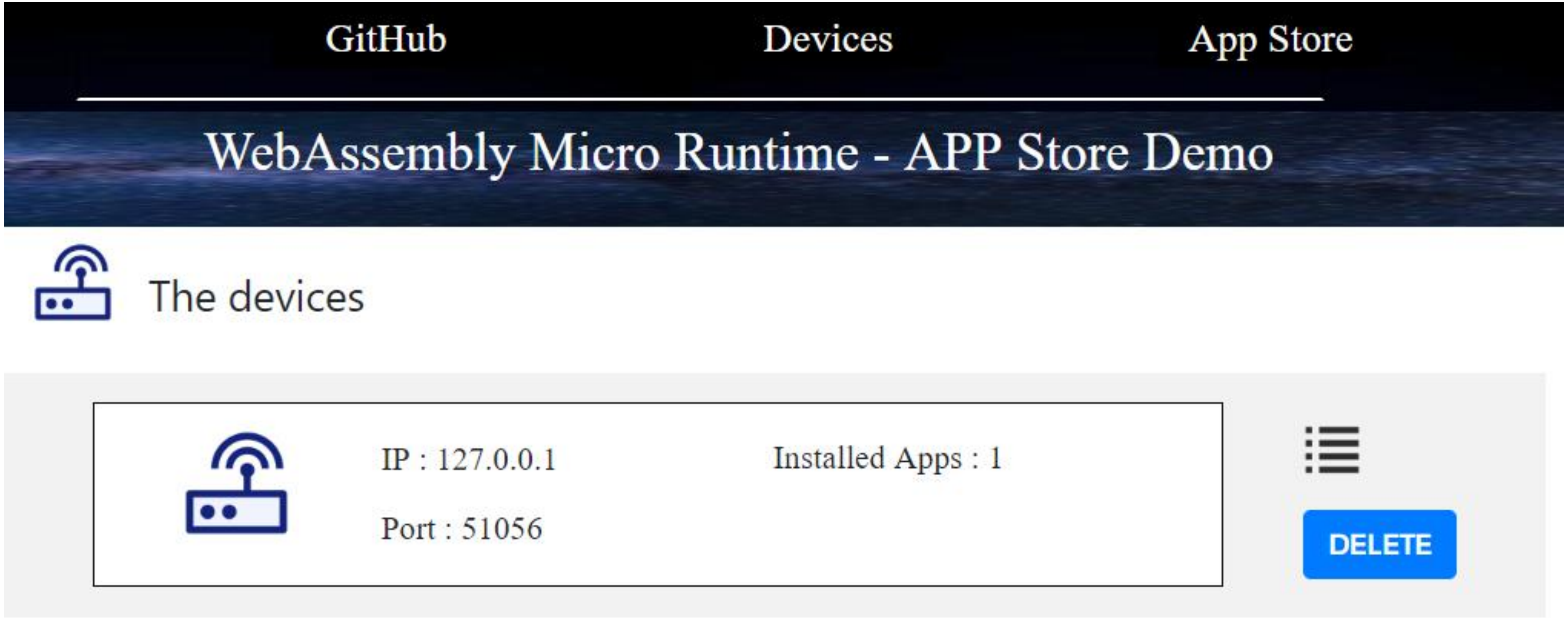
Execution Environment App Manager

Embedded OS

(*) for Java fw apps the bytecode is JEFF

IoT App Store Demo

http://39.106.110.7/



WebAssembly Micro Runtime - APP Store Demo

The devices

IP : 127.0.0.1

Port : 51056

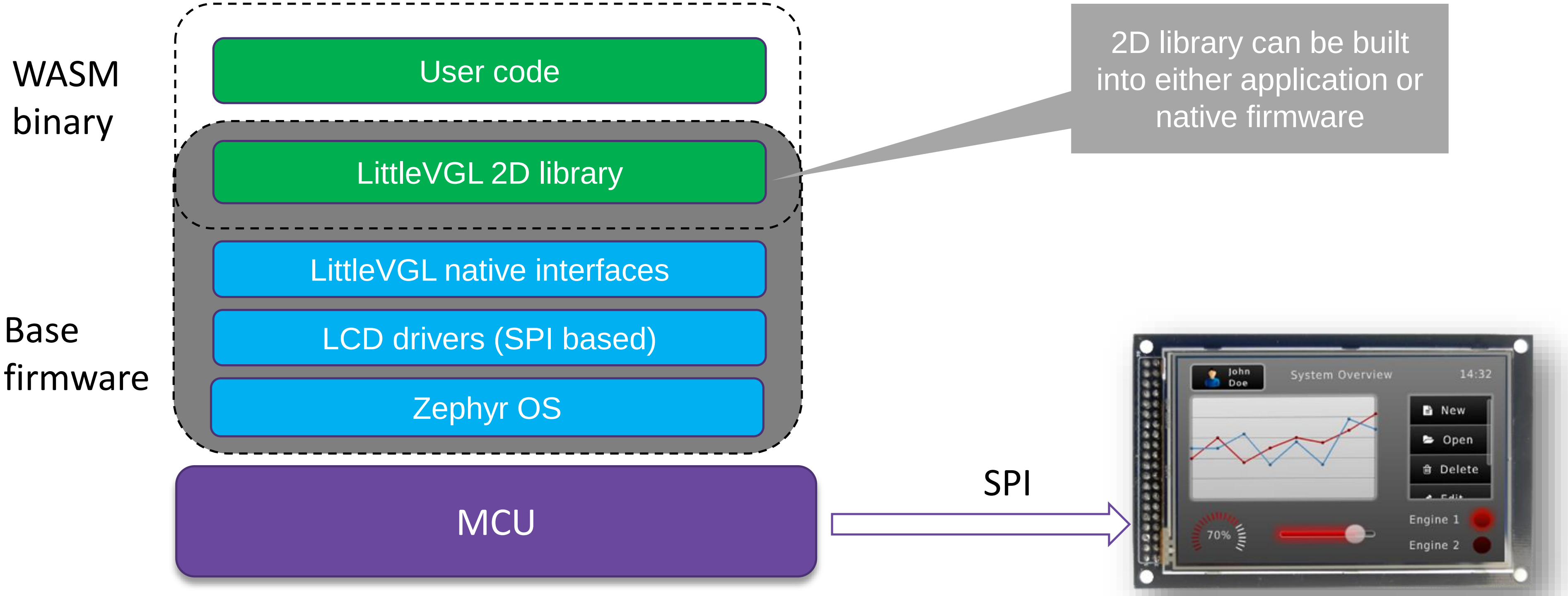
Installed Apps : 1

Demo

WAMR UI sample with 2D library “LittleVGL”

LITTLEVGL - Open-source Embedded GUI Library

- Powerful building blocks buttons, charts, lists, sliders, images etc
- Advanced graphics with animations, anti-aliasing, opacity, smooth scrolling
- Various input devices touch pad, mouse, keyboard, encoder etc
- Hardware independent to use with any microcontroller or display
- Scalable to operate with little memory (80 kB Flash, 10 kB RAM)
- Single frame buffer operation even with advanced graphical effects



2D library can be built into either application or native firmware

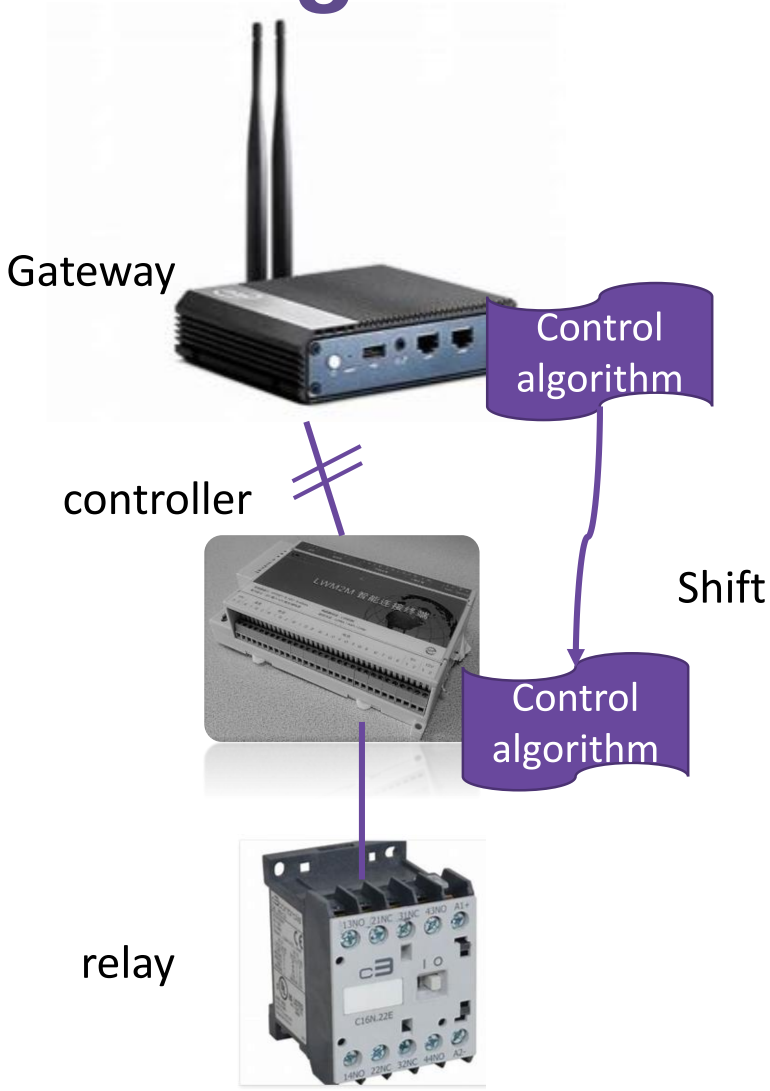
New Opportunities for IoT

IoT use case discussion – control algorithms

How to design a reliable control function is a very difficult part of IoT system

Too many possible failure points to be covered if we develop a gateway application sending control commands to the controller over the cable

WAMR enables control algorithms running from the controller. That reduces the complexity greatly comparing to achieving the same reliability level from gateway

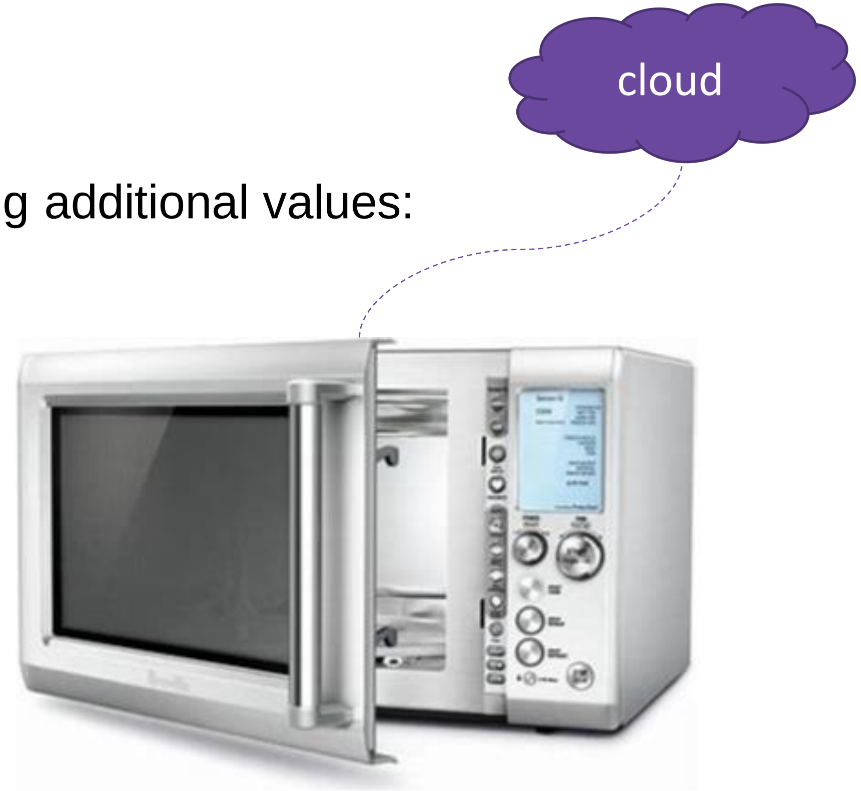


IoT use case discussion – smart small appliance

Small appliances are becoming connected and smart. And many of them are based on embedded system due to the cost requirements

Taking smart microwave oven as example, WAMR could bring following additional values:

- Personalized user interface applications
- Updating the control models from cloud
- Control models installed on demand
- Display the new product promotion news
- User feedback or remote diagnosis
- …



New development model

Before：

Product development team finish all the works below:

- Develop and integrate the OS and drivers
- Develop the applications
- Build image and test

Now：

Product system development team finish ：

- Develop and integrate the OS and drivers
- Integrate WASM runtime and design WASM APP API
- Build image and test
- Deliver the WASM APP SDK

Open development supported for the applications:

- Independent service development team
- System integrators
- 3rd party developers

WAMR Application Programming Tutorial

WAMR WASM App programming model

Single thread per WASM app instance

Event driven model

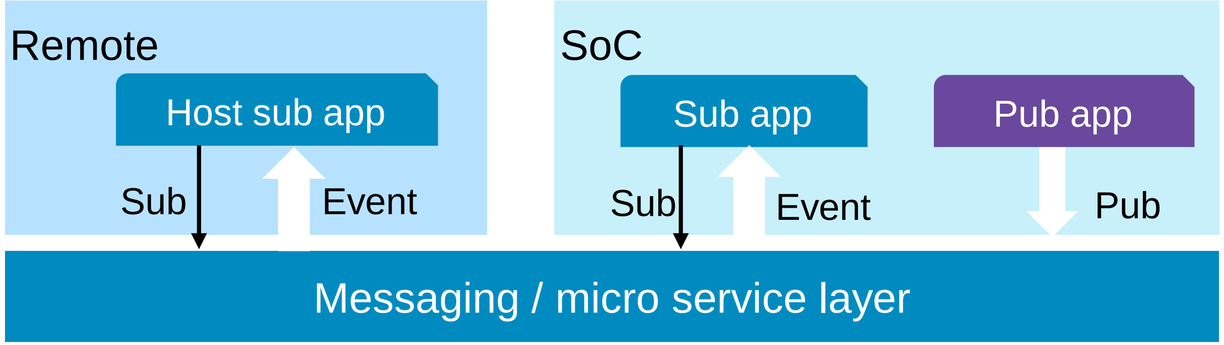
App must implement system callbacks:

  - on_init
  - on_destrory

Application API support:

- Timer
- Micro service (Request/Response)
- Pub/Sub
- Sensor
- Connection and data transmission

pub/sub sample



Firmware App as Subscriber

```
void on_init()
{
  api_subscribe_event (" alert/overheat", event1_handler);
}

void on_destroy()
{
}

void event1_handler(request_t *request)
{
}
```

Firmware App as Publisher

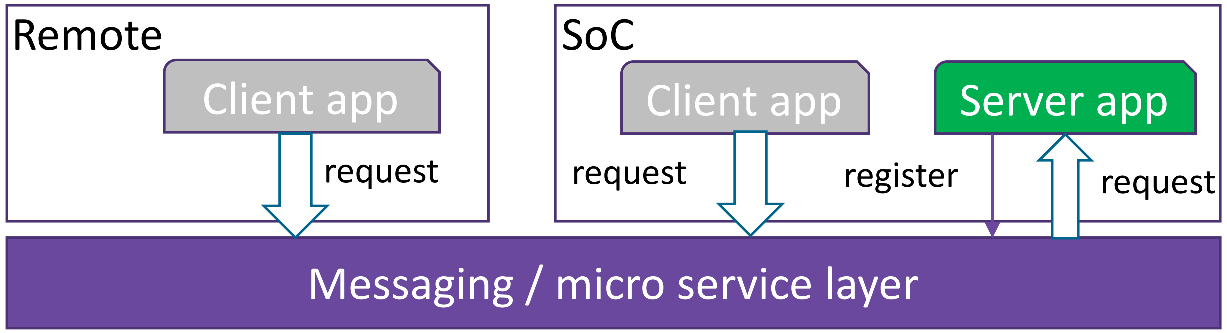
```
/* Timer callback */
void timer1_update(user_timer_t timer)
{
  attr_container_t *event;
  printf("Timer update %d\n", num++);

  event = attr_container_create("event");
  attr_container_set_string(&event,
          "warning",
          "temperature is over high");

  api_publish_event("alert/overheat",
          FMT_ATTR_CONTAINER,
          event,
          attr_container_get_serialize_length(event));

  attr_container_destroy(event);
}
void on_init()
{
    user_timer_t timer;
    timer = api_timer_create(1000, true, true, timer1_update);
}
```

micro service sample



```
void on_init()
{
  /* register resource uri */
  api_register_resource_handler("/url1", res1_handler);
  api_register_resource_handler("/url2", res2_handler);
}

void on_destroy()
{
}
```

```
void res1_handler(request_t *request)
{
  response_t response[1];
  attr_container_t *payload;
  payload = attr_container_create("wasm app response payload");
  if (payload == NULL)
    return;

  attr_container_set_string(&payload, "key1", "value1");
  attr_container_set_string(&payload, "key2", "value2");

  make_response_for_request(request, response);
  set_response(response,
          CONTENT_2_05,
          FMT_ATTR_CONTAINER,
          payload,
          attr_container_get_serialize_length(payload));

  api_response_send(response);
  attr_container_destroy(payload);
}
```

Connection and data transfer sample

```
#include "wasm_app.h"

/* User global variable */
static int num = 0;
static connection_t *g_conn = NULL;

void send(connection_t *conn)
{
    char message[64] = {0};
    snprintf(message, sizeof(message), "Hello %d", num++);
    api_send_on_connection(conn, message, strlen(message));
}

void on_data1(connection_t *conn,
              conn_event_type_t type,
              const char *data,
              uint32 len,
              void *user_data)
{
    if (type == CONN_EVENT_TYPE_DATA) {
        char message[64] = {0};
        memcpy(message, data, len);
        printf("Client got a message from server -> %s\n", message);
        send(conn);
    } else if (type == CONN_EVENT_TYPE_DISCONNECT) {
          g_conn = NULL;
          printf("connection is close by server!\n");
    } else {
        printf("error: got unknown event type!!!\n");
    }
}
```

```
void on_init()
{
    user_timer_t timer;
    attr_container_t *args;
    char *str = "this is client!";

    args = attr_container_create("");
    attr_container_set_string(&args, "address", "127.0.0.1");
    attr_container_set_uint16(&args, "port", 7777);

    g_conn = api_open_connection("TCP", args, on_data1, NULL);
    if (g_conn == NULL) {
        printf("connect to server fail!\n");
        return;
    }

    send(g_conn );
}

void on_destroy()
{
    if (g_conn != NULL) {
           api_close_connection(g_conn);
    }
}
```

New physical communication will be expanded through the parameter values

Programming remote app (Android*)

Example: Host app installs Wasm Firmware apps

```
//read fw app binary from sdcard to buffer
String path = "/sdcard/Demo1" + ".wasm";
byte []bpk_buf = read_file_to_buffer(path);

// Setup the payload for restful request
AttributeObject payload= new AttributeObject("Install Applet");
payload.set("name", "Demo1");                 // set applet name
payload.set("bpk", bpk_buf);                  // set applet package buf

// Request the app-mgr service for installing wasm fw app
Request req = new Request(“/applet”);
request.setAction(Request.PUT);
request.setPayload(payload);
request.send(response_handler, request_label_local);
```

*Other names and brands may be claimed as the property of others.

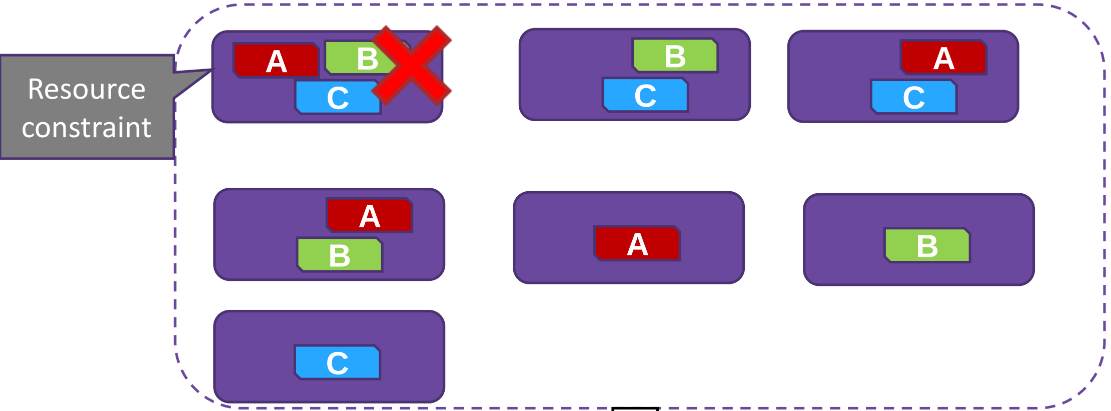
Summary

Unlimited usages on limited resources

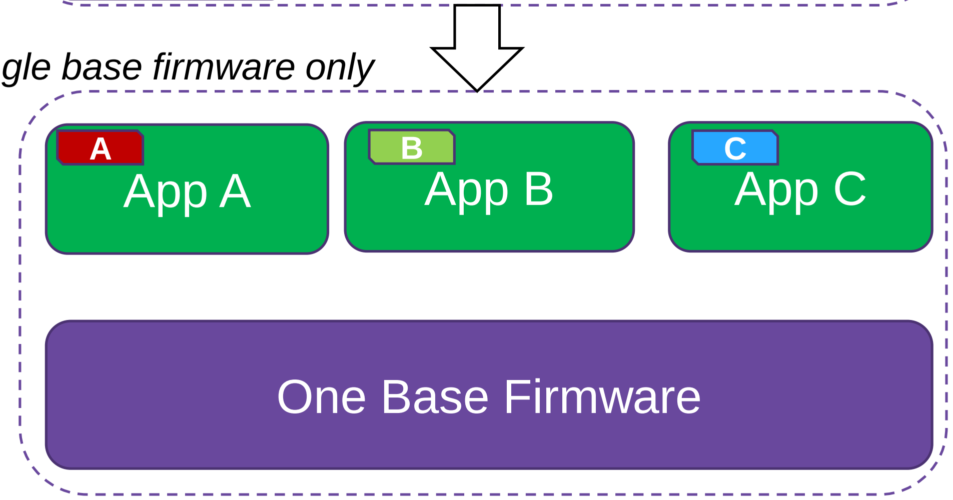
Reduce the base firmware variations

Use case A
Use case B
Use case C

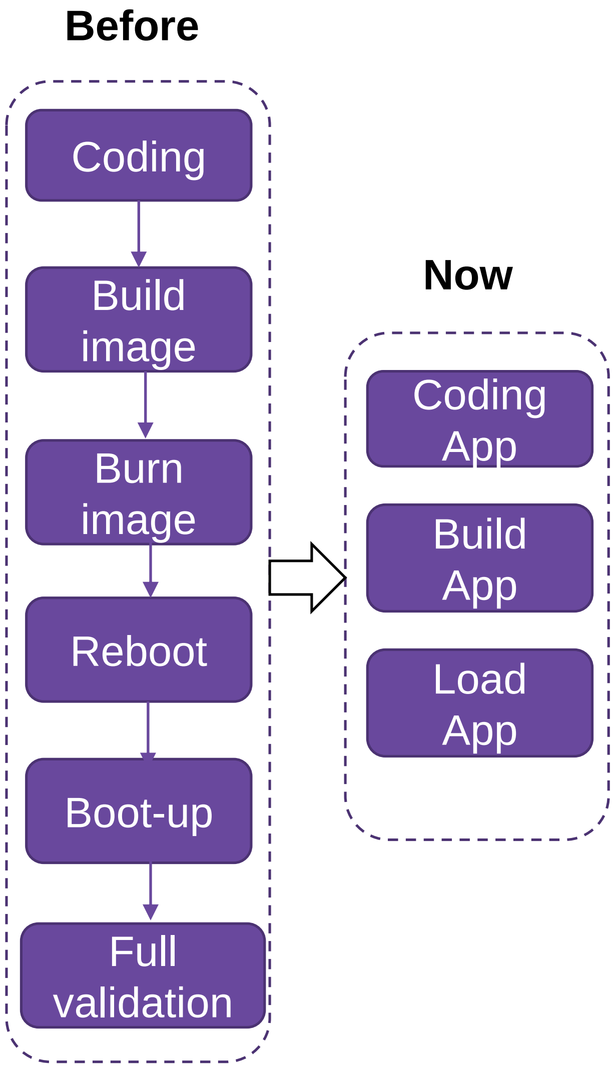
Before: *May maintain many FW versions for different functionalities*
Can't build all possible use cases into one image



Now: *single base firmware only*



Accelerate innovations



Enable 3rd party applications on commercial products

Before: *fixed user experience on the FW by device vendor*

(Like this)

(Like this)

Now: *3rd parties post launch development and FW apps on user demand*

Disclaimers

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request. No product or component can be absolutely secure.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm

Intel, and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others