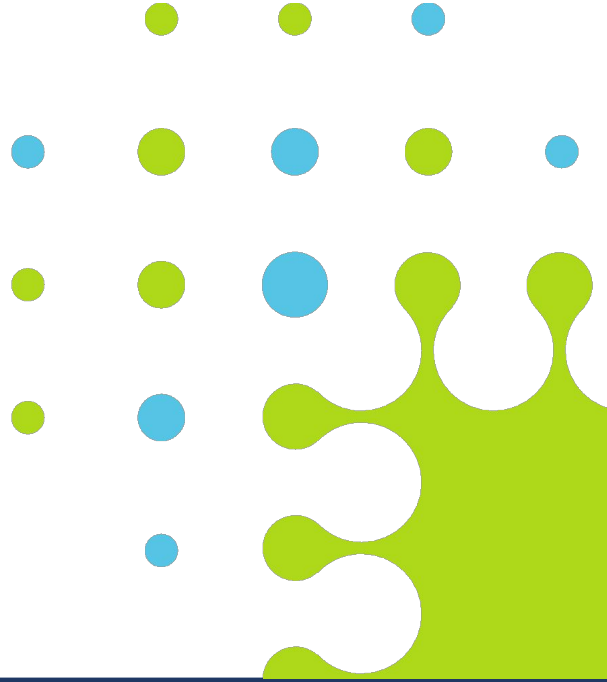
WEdge: Modernizing the Development of AI IoT Devices with WebAssembly-based Edge Virtualization

Midokura (A Sony Group Company)



AGENDA

Introduction
Hands-on Demo: Basic
Hands-on Demo: Advanced
Current and Future Work
Q&A and Feedback

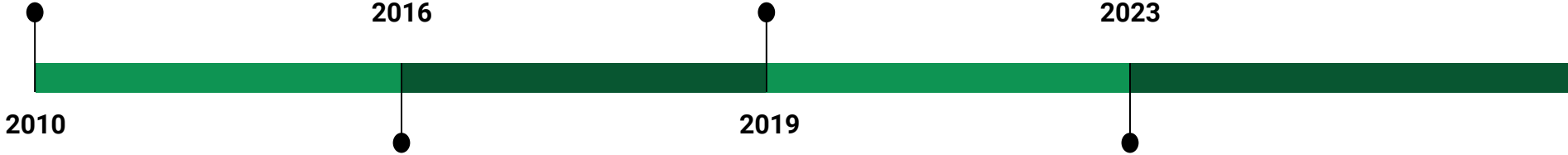
Midokura History

Midokura Founded

Network Virtualization for private clouds with OpenStack

Midokura Acquired by Sony Group

To provide infrastructure management for IoT as part of new Systems Solution BU of Sony Semiconductor Solutions.



IoT Pivot

Applying SDN to IoT security.

WEdge

Embedded device virtualization platform based on WebAssembly

Vision

An accessible platform that powers and enables intelligent solutions based on distributed visual sensing

Targeting solution developers for various vertical applications.

Bring to market a plethora of low cost, easy-to-use sensing devices

High level abstractions lowers the barrier to entry for solution developers

Rapid adoption of our open source device stack

Low operational cost of vision sensing apps

Uncompromised privacy: raw data never sent from sensors

Marketplace to connect AI Developers & Solution Developers

IoT Development Challenges

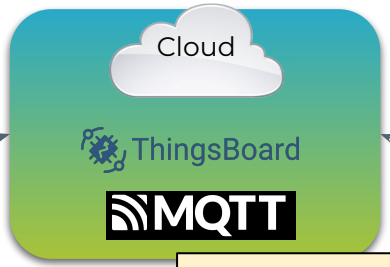
IoT solutions will continue to evolve. Updating device functionality needs to be easy, safe, fast and frequent!

- Most solutions require complete firmware OTA update or device replacement.
- Embedded development is challenging
  - Only low-level languages supported, no reusable libraries, and device specific knowledge may be required
- Isolation lacking at the application level
  - No memory management unit!
  - Changing any functionality requires full firmware update

→ Need lifecycle management of applications on devices (containers are too big!)

WEdge Solution Overview

Develop in your preferred environment with SDK and CLI



Connect to supported IoT services for device and application management.

Devices supported (or will be soon):
MCUs: Esp32, i.mx8, Spresense, etc
Cross-over MCUs: i.MX1170
Others: Raspi, Jetson

WEdge SDK provides useful API for developers to simplify development

- Sensors: Get image frames and configure sensor (frame rate, etc)
- AI/ML: Load model and run inference
- Communication: Application to application, device to device, and device to cloud

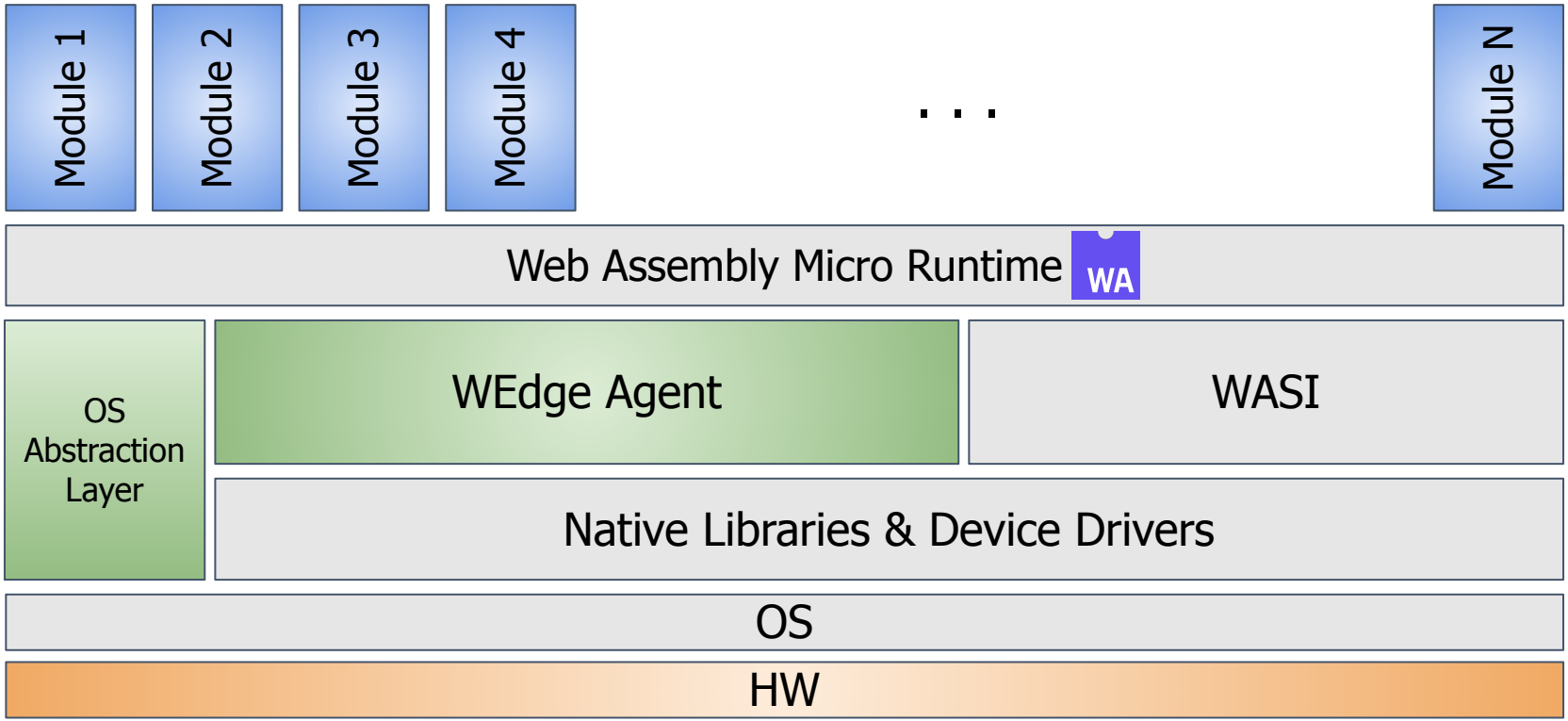
WEdge CLI provides commands to manage applications and agent

- Control WEdge agent locally and remotely
- Manage and deploy applications to a local or remote device
- View application outputs and logs for testing and debugging

WEdge Agent: Like Kubernetes(kubelet) but for tiny IoT devices

- Automated lifecycle management of applications
- Leverages WebAssembly Micro Runtime (WAMR) for strong isolation of applications, with support for high level languages
  - May support different runtimes
- Designed to easily integrate with different IoT platforms
  - Thingsboard currently supported
  - Any MQTT broker can be used for development

WEdge Device Stack



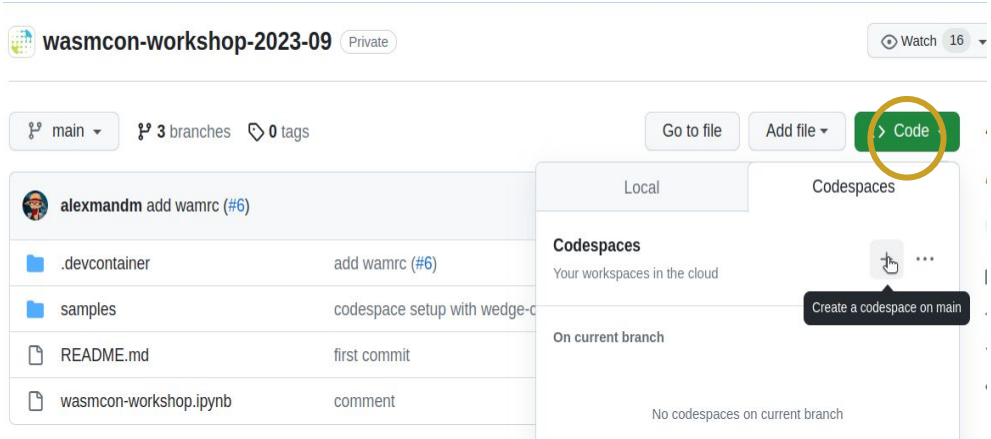
Hands-on Demo: Basic

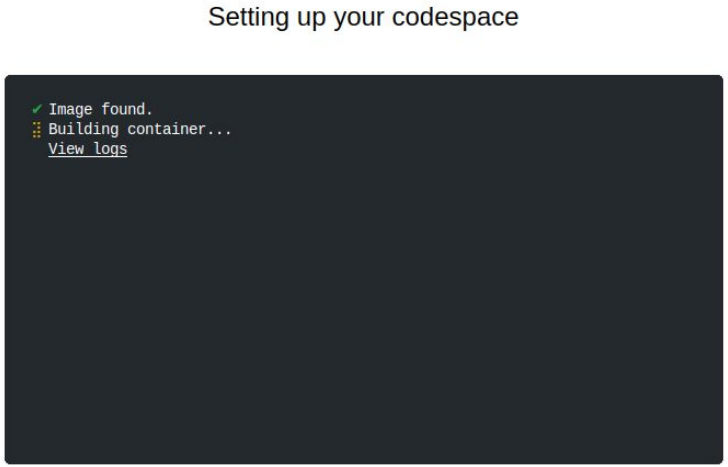
Demo Goals

- Basic Understanding of WEdge
- IoT Application Development Experience
- Deploy and Run Simple App

Open Codespaces from your browser

1. Go to **https://github.com/midokura/wasmcon-workshop-2023-09**
2. Open Codespaces

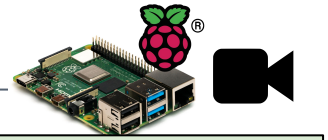




Constraints in the Workshop Environment

Develop and Deploy on Raspi

WEdge Agent is not open yet



WEdge Agent
WEdge CLI

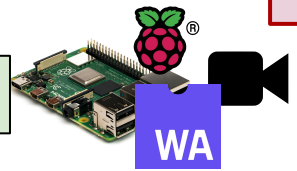
Develop in Codespaces and deploy to Raspi

X

Codespaces does not allow MQTT ports to be open

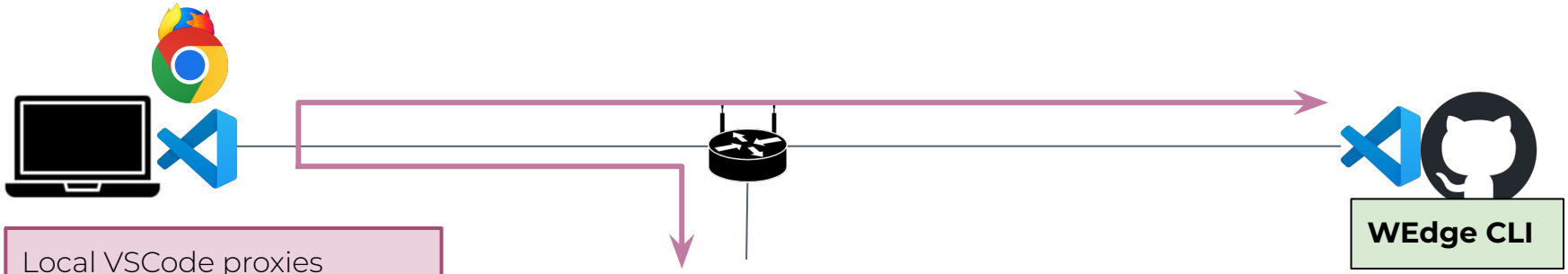
WEdge CLI

WEdge Agent



Workshop Environment - Final Setup

Develop in Codespaces and deploy to Raspi with network traffic going through local VSCode!



Local VSCode proxies network traffic from WEdge agent to Codespaces. This configuration is automatically set up.

WEdge CLI

WEdge Agent

Laptops and Raspberry Pis must be connected to the same WiFi router.

Static IP addresses are assigned to the Raspberry Pi devices.

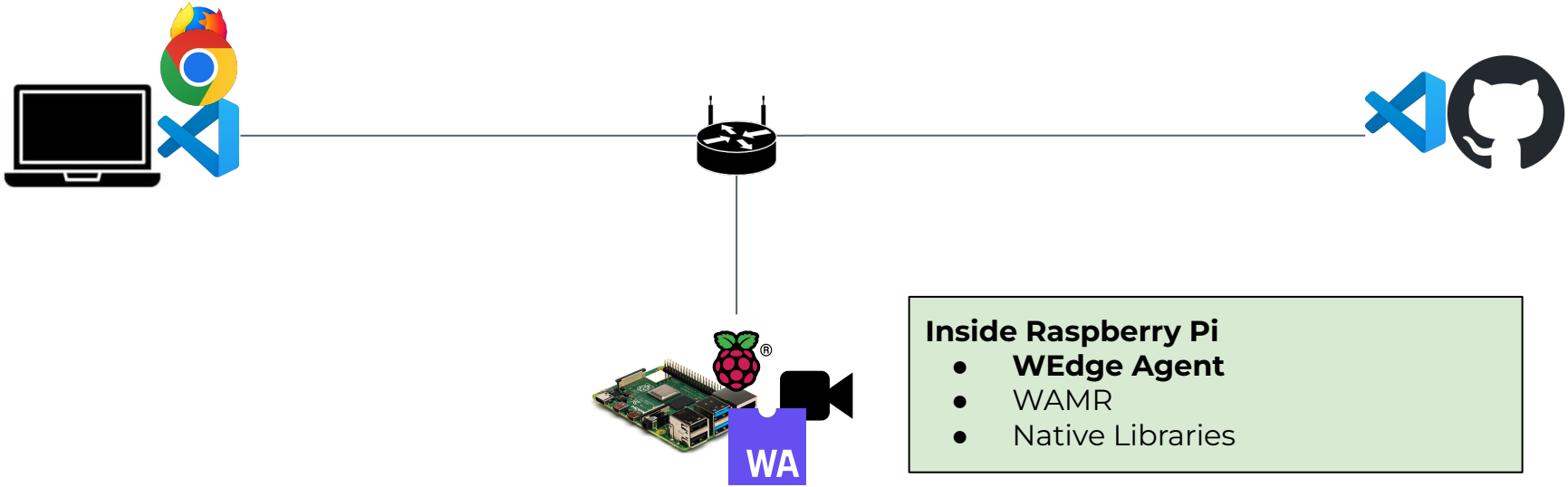
Workshop Environment - Final Setup in Detail

User Requirements

- GitHub account for Codespaces
- VSCode with GitHub Codespaces and Jupyter extensions
- netcat (nc) to send initialization command to the device via TCP
- Browser (Chrome recommended) for viewing visual output

Inside Codespaces

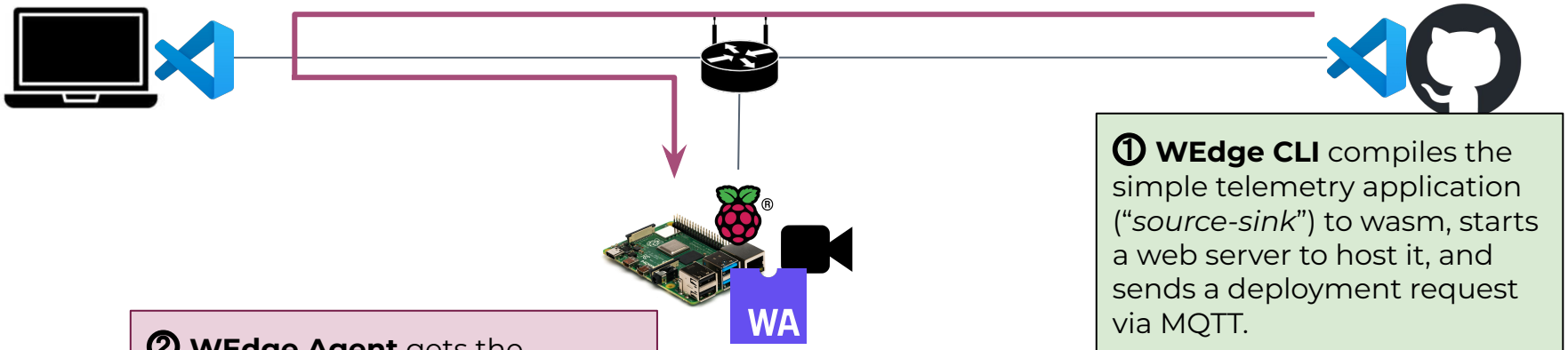
- **WEdge CLI**
- MQTT Broker
- WASM Compiler
- Sample Applications



Inside Raspberry Pi

- **WEdge Agent**
- WAMR
- Native Libraries

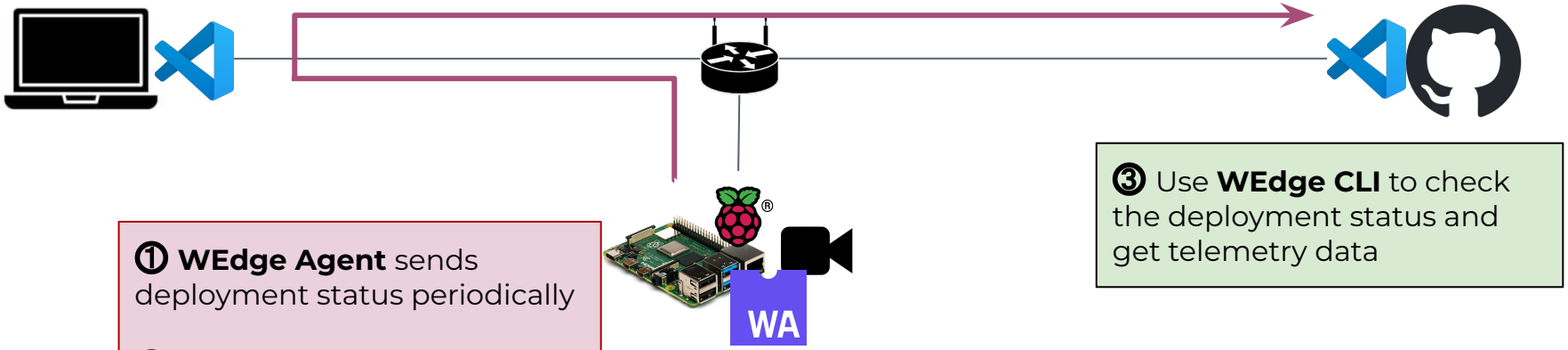
Deployment Workflow



❶ **WEdge CLI** compiles the simple telemetry application (“*source-sink*”) to wasm, starts a web server to host it, and sends a deployment request via MQTT.

❷ **WEdge Agent** gets the deployment request, downloads the application and instantiates it.

Simple Telemetry Application - “source-sink”



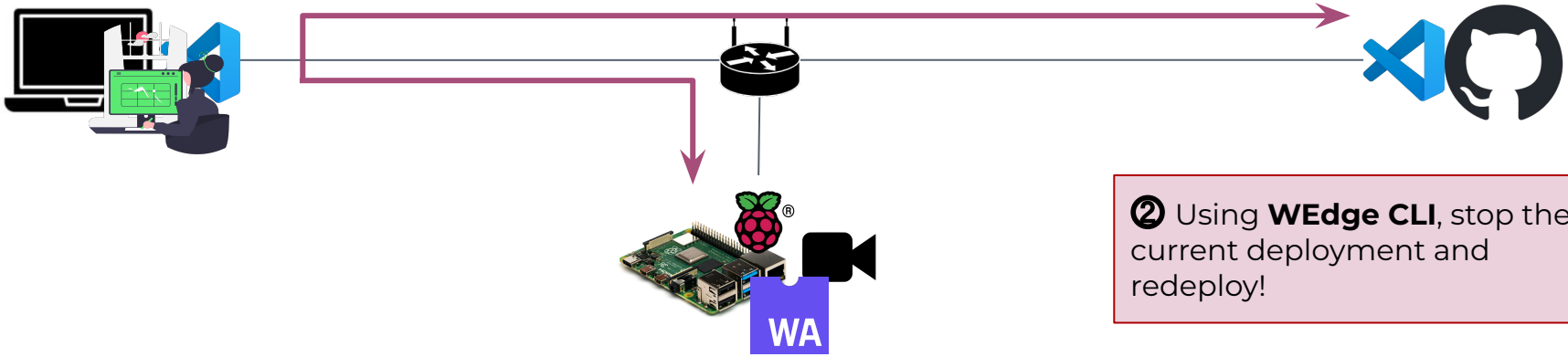
❶ **WEdge Agent** sends deployment status periodically

❷ App sends text telemetry periodically

❸ Use **WEdge CLI** to check the deployment status and get telemetry data

Modify Application and Deploy!

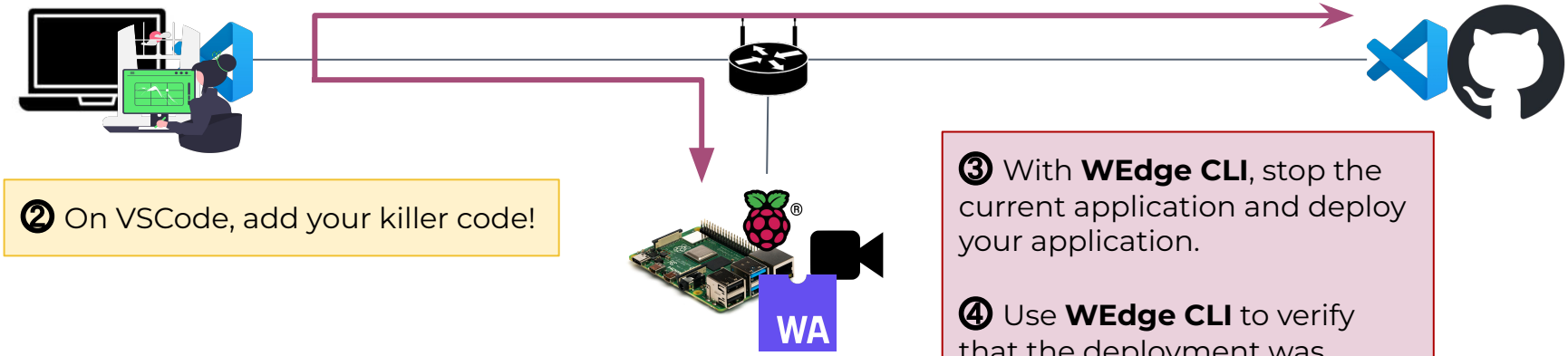
❶ From VSCode, modify “source-sink” application!



❷ Using **WEdge CLI**, stop the current deployment and redeploy!

Create Your Own Application!

❶ Create a new application template with **WEdge CLI**

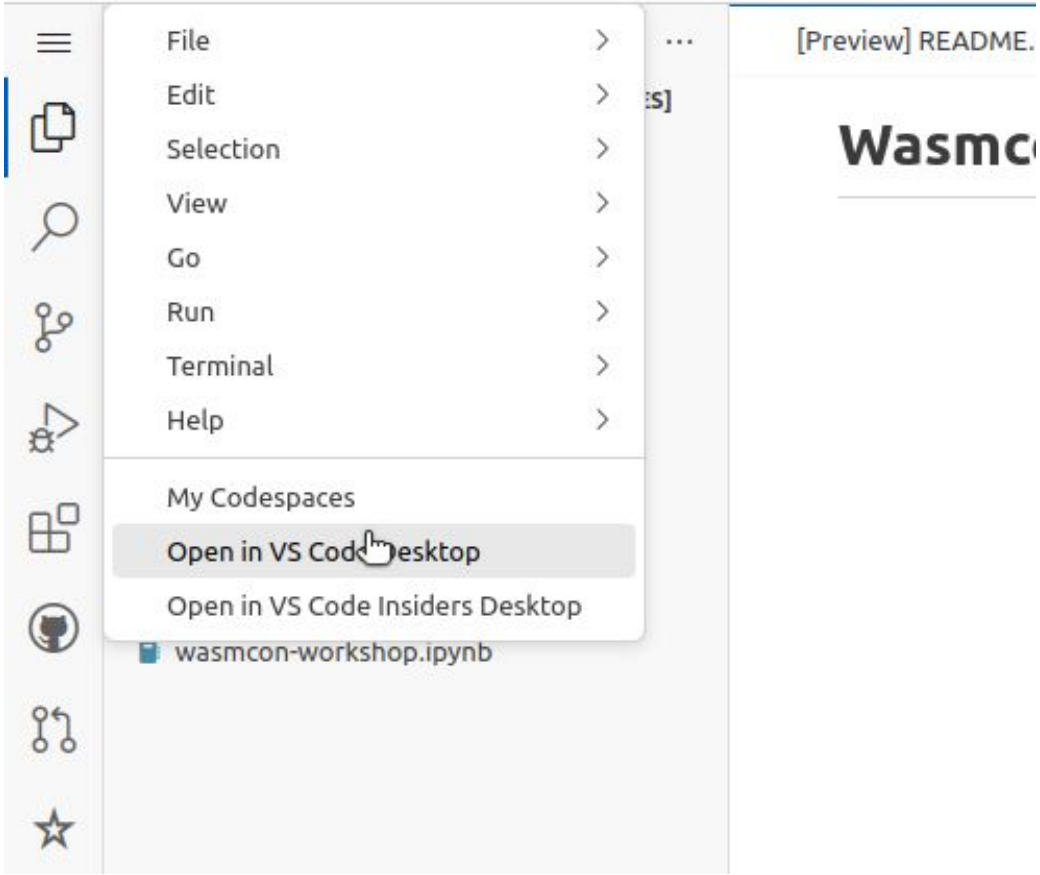


❷ On VSCode, add your killer code!

❸ With **WEdge CLI**, stop the current application and deploy your application.

❹ Use **WEdge CLI** to verify that the deployment was successful.

Open Codespaces with your desktop VSCode



Basic Hands-on Demo Instructions

https://github.com/midokura/wasmcon-workshop-2023-09

For those joining late, follow the instruction steps in README or just call assistants over for help!

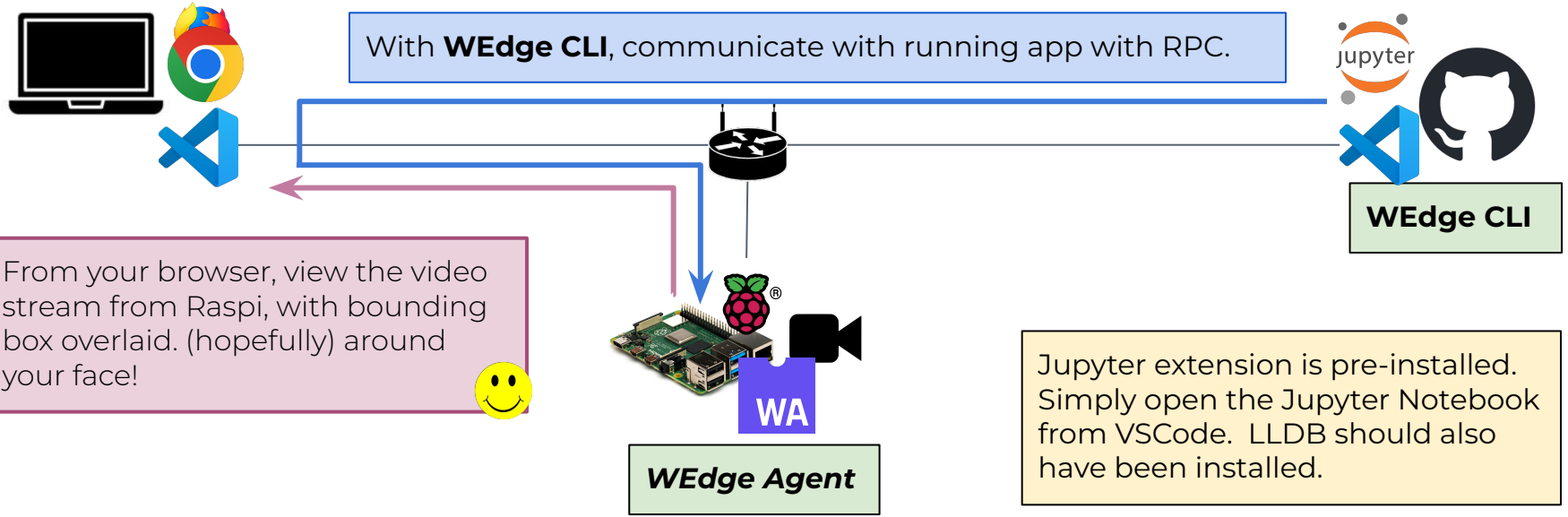
Hands-on Demo: Advanced

Demo Goals

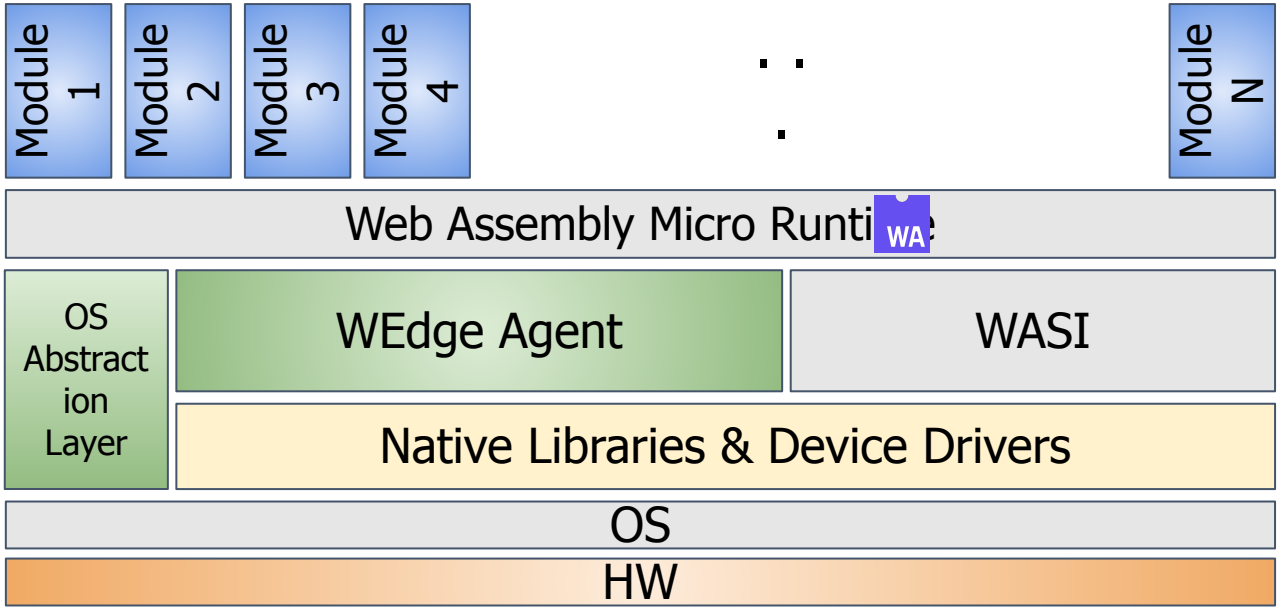
- Running a visual sensing application with sensor native library
- Easy customization of the IoT device functionality

Workshop Environment - Advanced - Face Detection

From Jupyter, see the entire development process of a visual sensing application, including debugging, testing with a sample image, verifying the deployed application visually, and dynamically modify the application's behavior with RPC command.



Workshop Environment - Advanced - Native Libraries



Native libraries are dynamically loaded at the start of the WEdge agent.

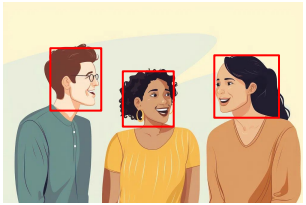
Native LIbraries Used

- **senscord:** Configure image sensors and get frames
- **senscord-ub:** Stream resulting images
- **opencv**: Process images (resize, crop, etc)
- **wasi-nn:** Execute inference

Workflow

This workflow assumes that the user is an AI model developer
There is no dependency between Raspberry Pi.

AI model/image setup <codespaces>	Run Wasm App with Debugger <codespaces>	Deploy to Raspberry Pi4 <raspberry pi>
Prepare the Test environment	Run Wasm application with LLDB Debugger	Deploy to Raspberry Pi4
- Image Pre-processing - Prepare AI Model - Build app with WEdge CLI	- Confirm the inference result - Check what kind of data can be obtained from native libraries - Customize the visualization	- Deploy same Wasm application to Raspberry Pi4 using WEdge-CLI - Confirm the inference result with live camera stream - Change camera setting from codespaces



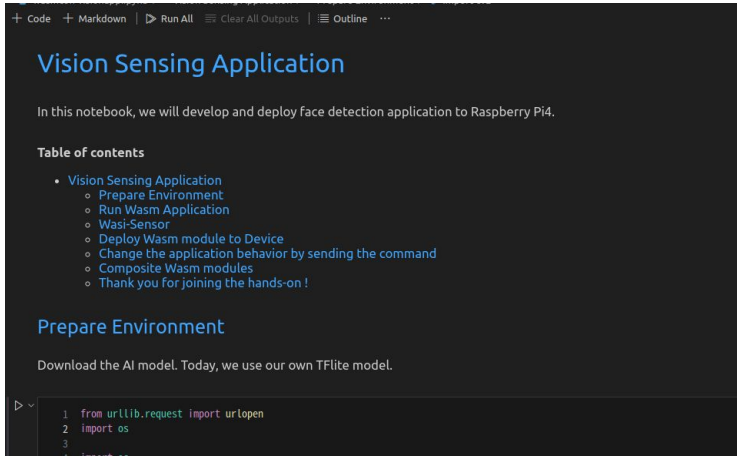
```
105
106     ret = senscord_stream_start(stream);
107     if (ret < 0) {
108         ERR_PRINTF("senscord_stream_start :
109         senscord_get_last_error(&status);
110         PrintError(status);
111         goto close;
112     }
113     s_stream_stopped = false;
```

Is your face detected ?

https://www.rawpixel.com/image/12031004/image-face-paper-person

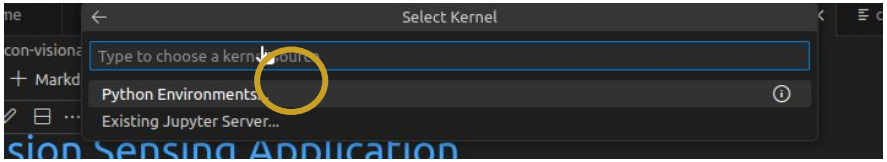
Go to Jupyter Notebook

1. Go to jupyter notebook.

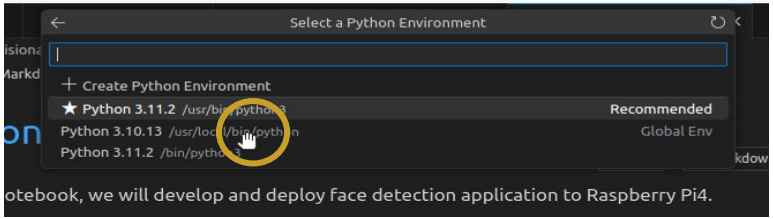


2. Select Kernel

3. Select Python Environments



4. Select Python 3.10.13



Introducing WASI-Sensor

wasi-sensor / wit / **sensor.wit**

ayakoakasaka folder changes

Code | Blame | 58 lines (44 loc) · 1.23 KB

```
1   /// WASI Sensor is an Sensor abstraction API
2
3   // https://github.com/WebAssembly/component-model/blob/main/design/mvp/WIT.md
4
5   interface sensor {
6     use property.{property}
7
8     variant res {
9         none(string),
10        not-found(string),
11        ...
12    }
13
14    type device = u32
15
16    // https://github.com/WebAssembly/component-model/blob/main/design/mvp/WIT.md#item-resource
17    // Sensor as Resource
18    resource device {
19        constructor: func() -> device
20
21        /// add condition to get device list
22        query: func() -> list<string>
23
24        /// power off -> power on/streaming off
25        open: func (
26            device: string
27        )->result<res>
28
```

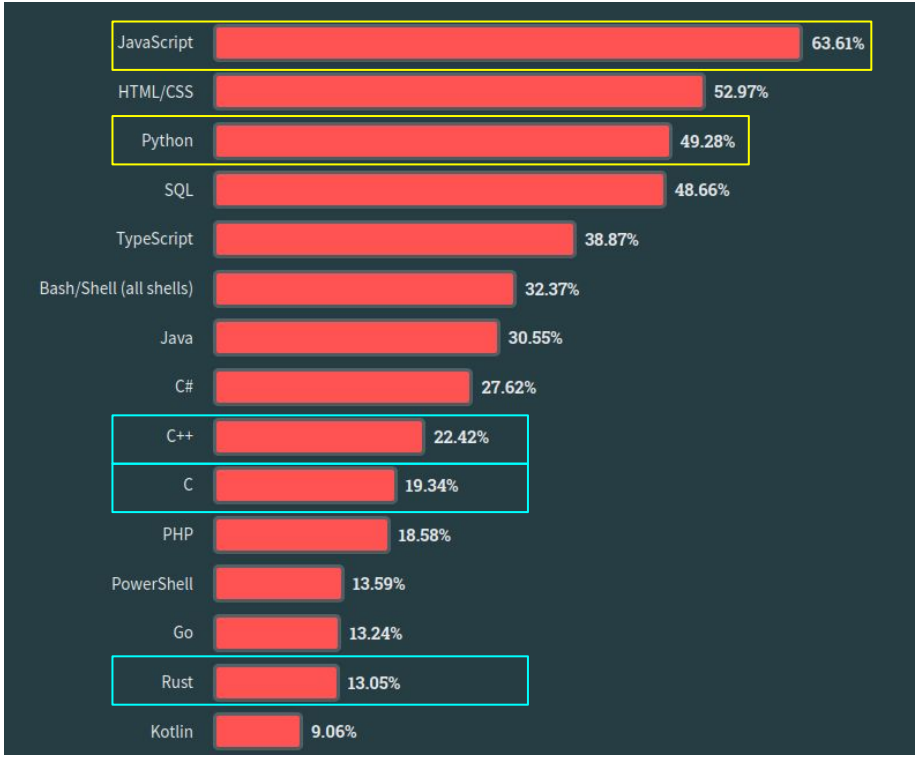
Current and Future Work

Python to Wasm

- Current state of wasm development with python
- Py2wasm: what is it and why we chose to do it this way
- Demo

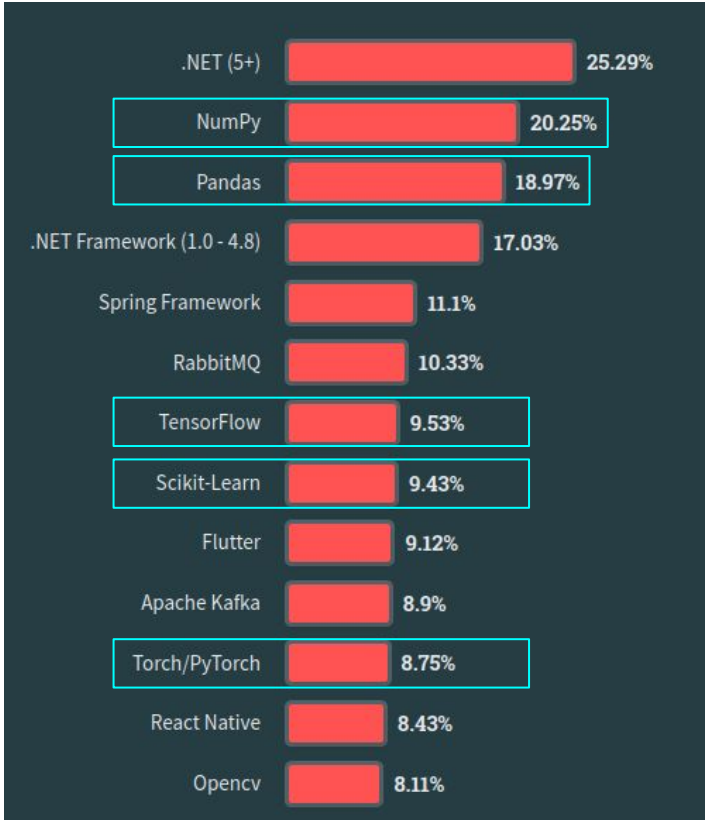
Most popular programming, scripting, and markup languages

Data extracted from https://survey.stackoverflow.co/2023



- Compiled languages compatible with WASM are known by ~25% of respondents
- JavaScript and Python (both interpreted) are known by approx half of the people

Most popular frameworks



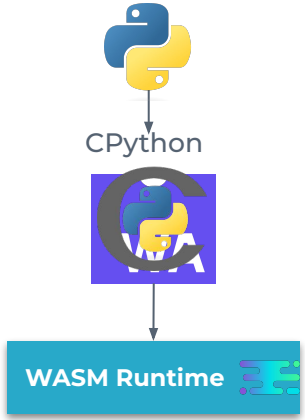
- Most frameworks related to data or AI are Python-specific

3 Ways to Develop Wasm Module using Python

1. Compile python code to WASM

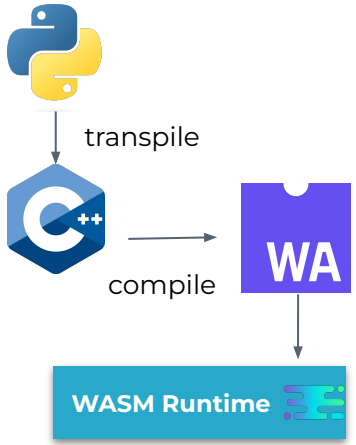
Cannot use full features of Python

2. Make CPython interpreter into wasm



The size of CPython wasm is too large (AOT = ~20MB) for tiny IoT devices

3. Transpile python Code to C++, then compile to wasm



Cannot transpile full features of python (such as Class)

Py2wasm

User's Python Code

CPython.wasm

WASM Runtime

Slow and lacks useful libraries

User's Python Code

Useful Python packages

numpy | pydantic | lap | pillow | flatbuffers

Python to WASM conversion toolchain

py2wasm

Cython

User's Python C Code

+

main.c

+

Frozen modules and objects

+

cpython-312-wasm32-wasi.so

module.wasm

WASM Runtime

Sufficient environment for wasm app development with python

CPython C extensions we currently support

Name	Description
numpy	Scientific computing library
pydantic	Data validation library
pillow	Image processing library
lap	Linear assignment problem solver

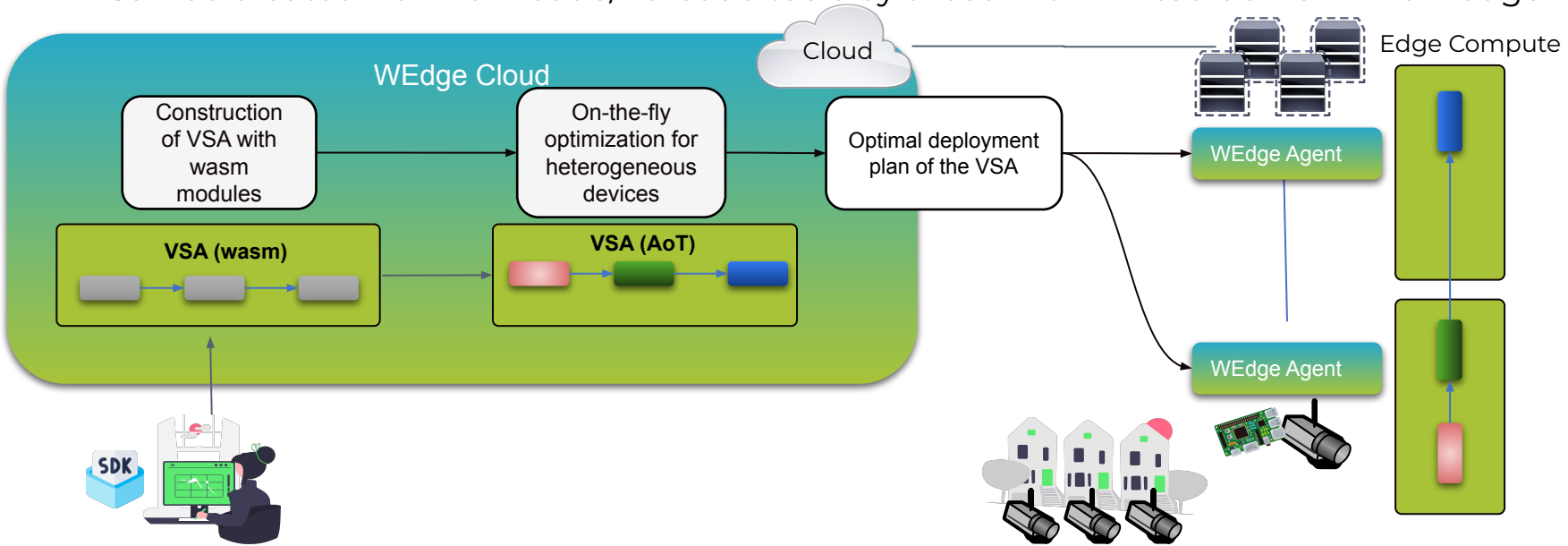
Sensing Pipeline

- Simplify visual sensing application development with better composability and reusability of wasm modules
- Optimal deployment strategy

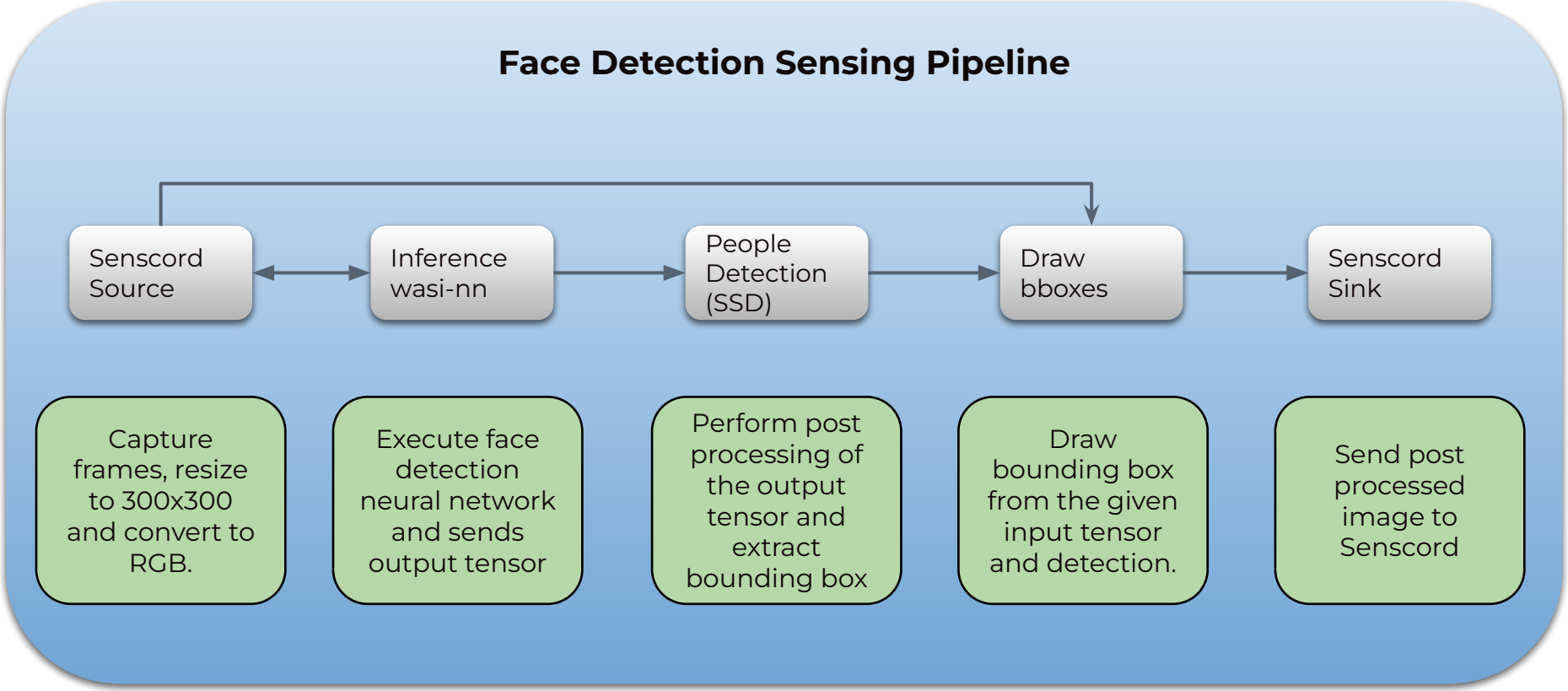
Sensing Pipeline: Vision Sensing Application

Vision Sensing Application (VSA): Combining simple steps to develop a complex and meaningful task

- Composable and reusable WASM modules
- Solution creation is decoupled from the app instantiation logic
- Can be created from low-code/no-code tools by those with limited domain knowledge



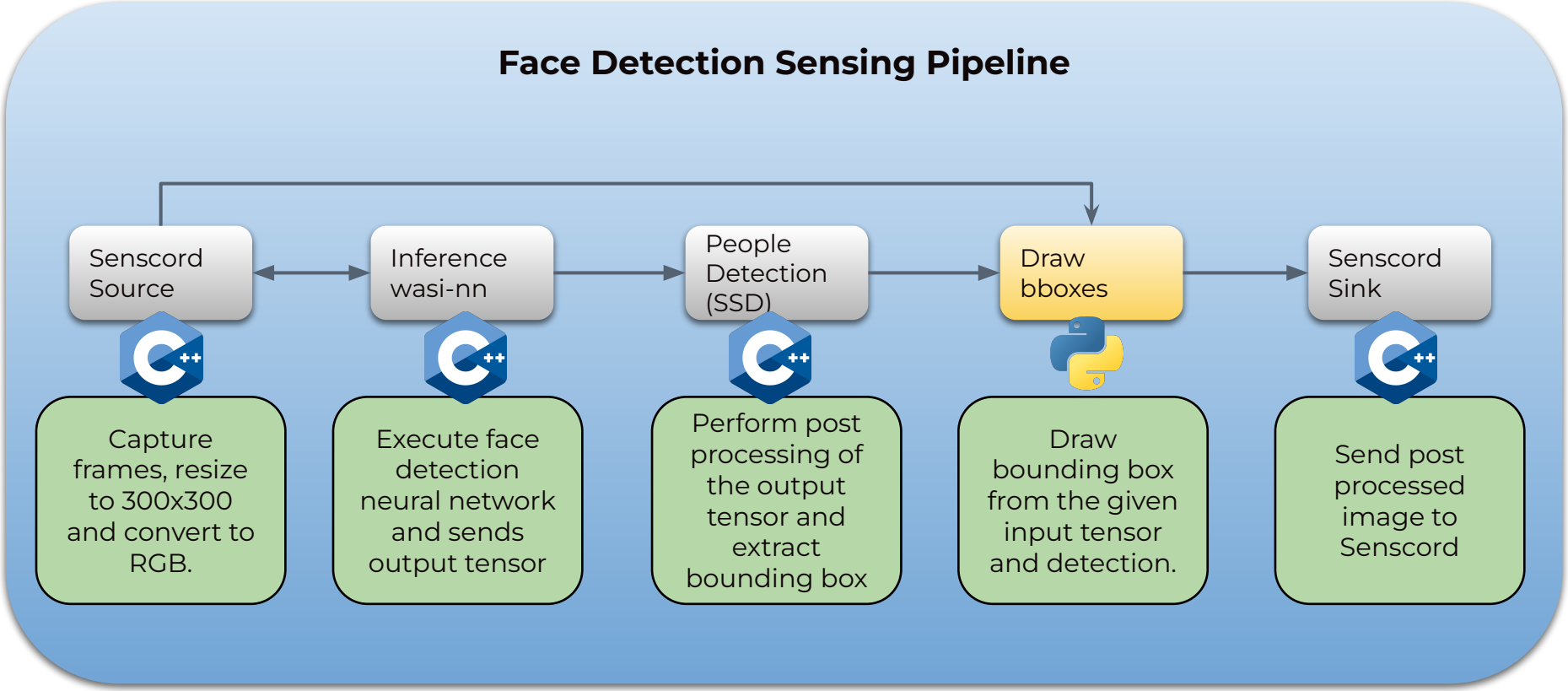
Detection Application as Sensing Pipeline



Putting it all together!

Deploy a sensing pipeline application composed of C and Python nodes!

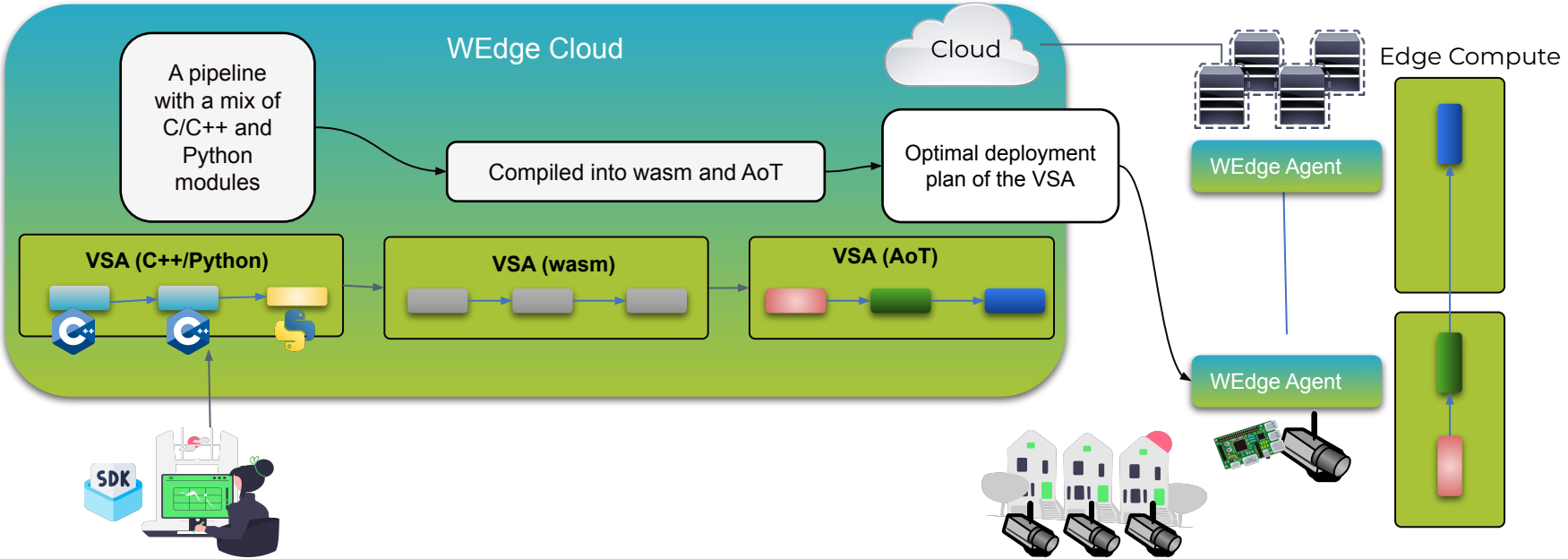
Detection Application as Sensing Pipeline



Deploying Vision Sensing Pipeline Application

Vision Sensing Application (VSA):

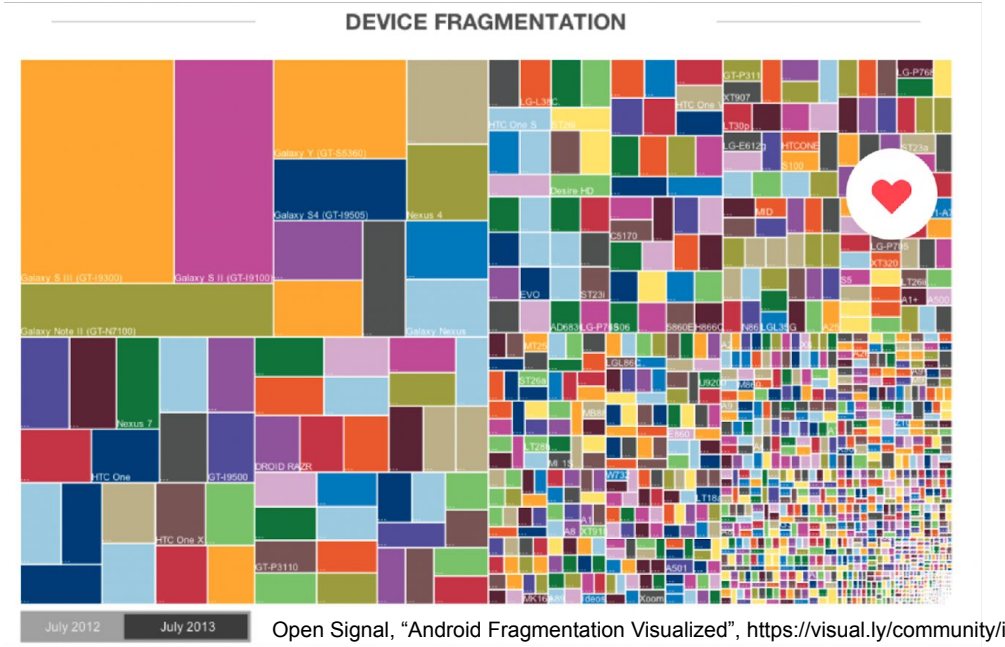
- Compose a VSA with nodes coded in multiple languages
- Include python, which is one of the most popular languages for AI engineers
- Once the pipeline is constructed, the instantiation part is done by the platform



About OSS...

- Feedback!

Open Source to Solve Device Fragmentation in IoT



"We have seen 11,868 distinct devices download our app in the past few months. In our report last year we saw 3,997."

OpenSignal, July 2013

This is the best way of visualizing the sheer number of different Android devices that have downloaded the OpenSignal app in the past few months. From a developer's perspective, comparing fragmentation from this year to the previous year, we see that it has tripled, with even more obscure devices from around the world downloading the app. If you want to understand the challenge of building an app that will work on all devices that want to download it, this image is a good place to start!

Open Signal, "Android Fragmentation Visualized", https://visual.ly/community/interactive-graphic/technology/android-fragmentation-visualized

- IoT devices with a variety of processors, access networks, operating systems, and sensors
- The same thing as Android' fragmentation will happen

→ *WEdge it open!* Feedback greatly appreciated!

Q&A and Feedback

Thank you