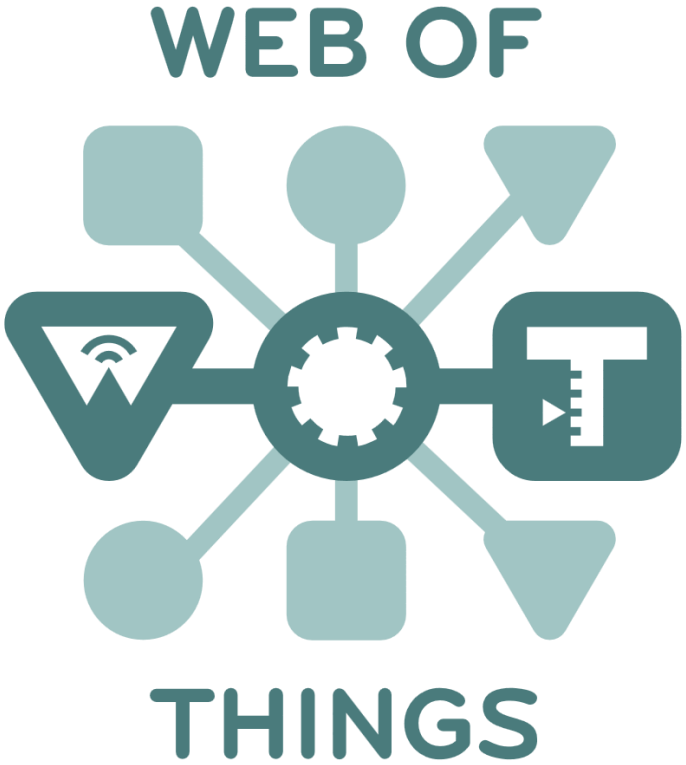


Getting Started with a W3C WoT Project

Eclipse Virtual IoT Meetup, 16 Jun 2016

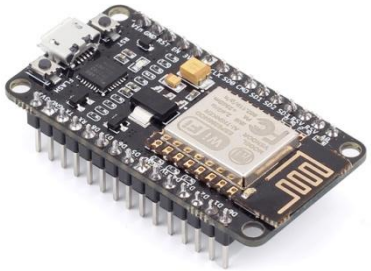
What is the Web of Things?

Application Layer

Internet of Things: **Connectivity**

IEEE 802.15.4

Ethernet



Wi-Fi

Bluetooth

LoRa

…

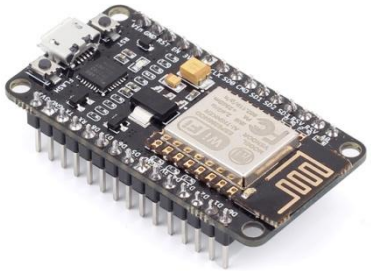
What is the Web of Things?

OCF™ oneM2M openHAB empowering the smart home oma Open Mobile Alliance For a Connected World OPC FOUNDATION

Internet of Things: **Connectivity**

IEEE 802.15.4

Ethernet



Wi-Fi

Bluetooth

LoRa

…

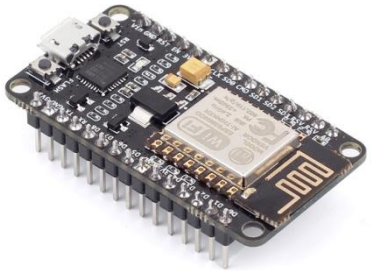
What is the Web of Things?

Web of Things

Internet of Things: **Connectivity**

IEEE 802.15.4

Ethernet



Wi-Fi

Bluetooth

LoRa

…

W3C WoT Mission

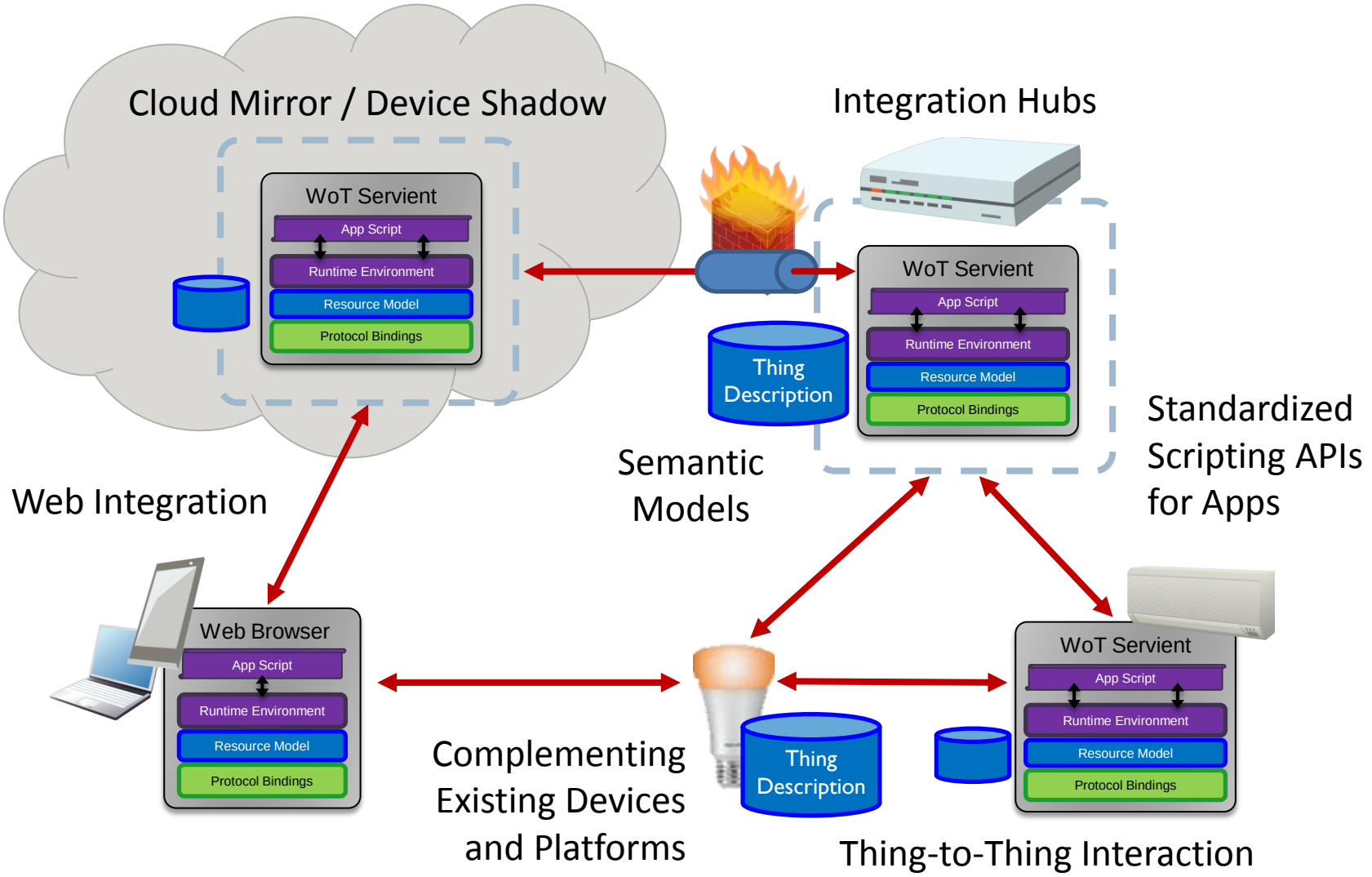
Not to be yet another standard

Web of Things

...

“interconnecting existing Internet of Things platforms and complementing available standards”

Overview of WoT Concepts



Outline

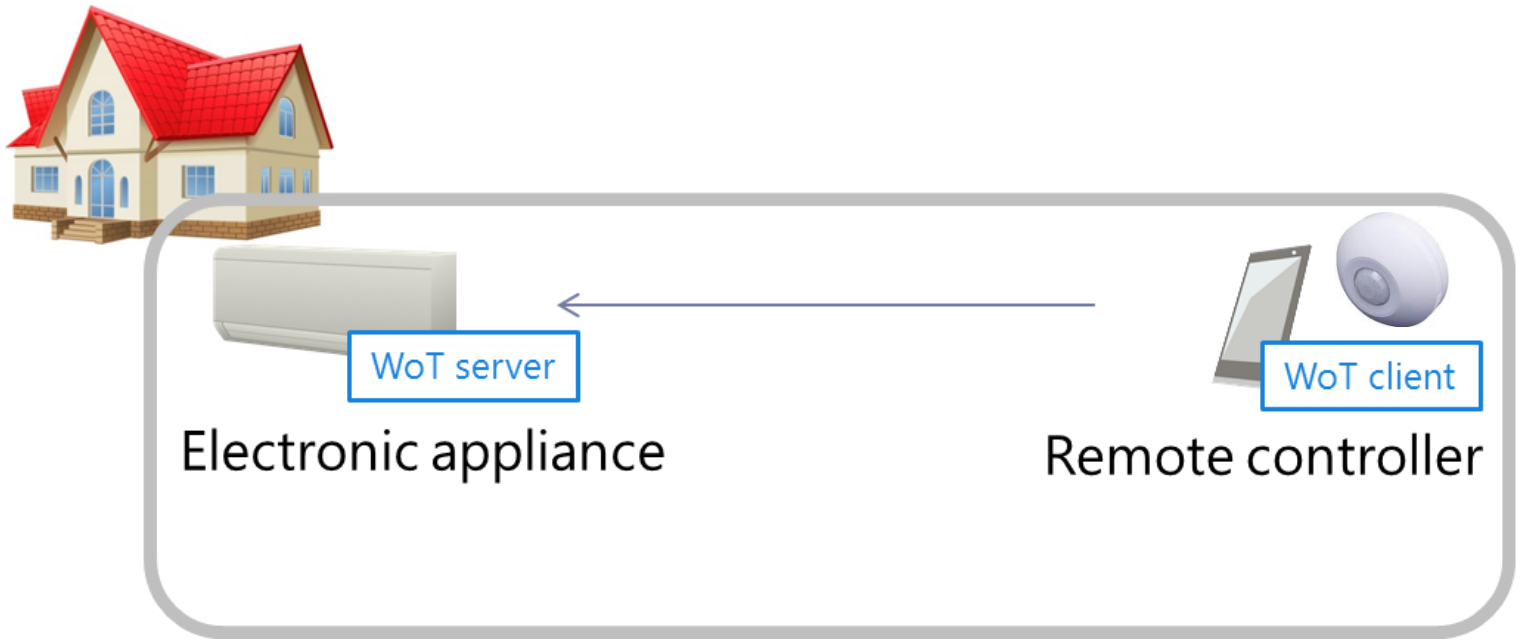
- Architecture
  - Things, Deployment Scenarios, and Servients
- WoT Interface
  - Protocol Bindings and the Web
- Thing Description (TD)
  - Metadata and Interactions
- Scripting API
  - Runtime Environment and Portable Apps

Things, Deployment Scenarios, and Servients

http://w3c.github.io/wot/architecture/wot-architecture.html

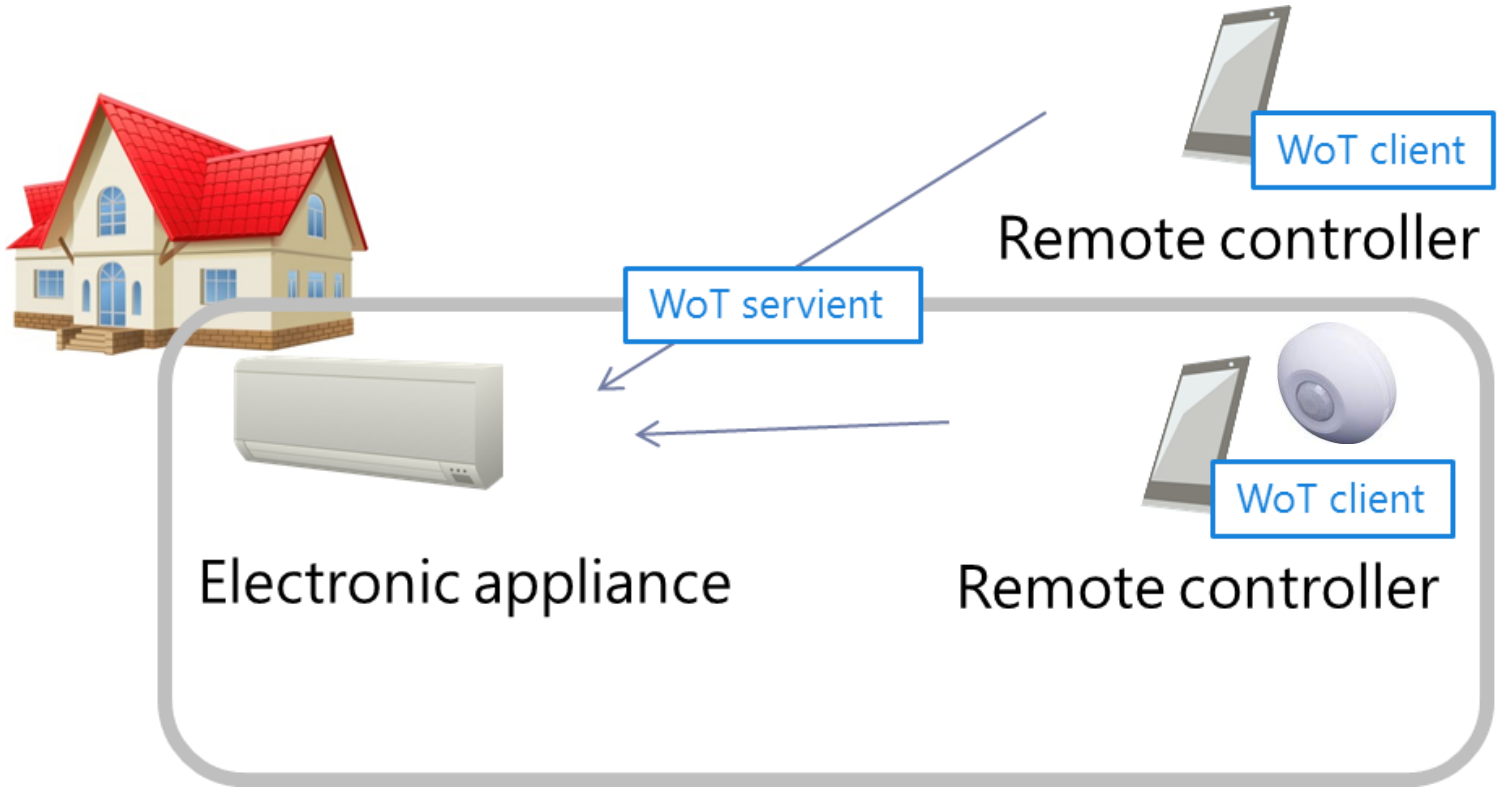
WoT ARCHITECTURE

Local **Thing-to-Thing**



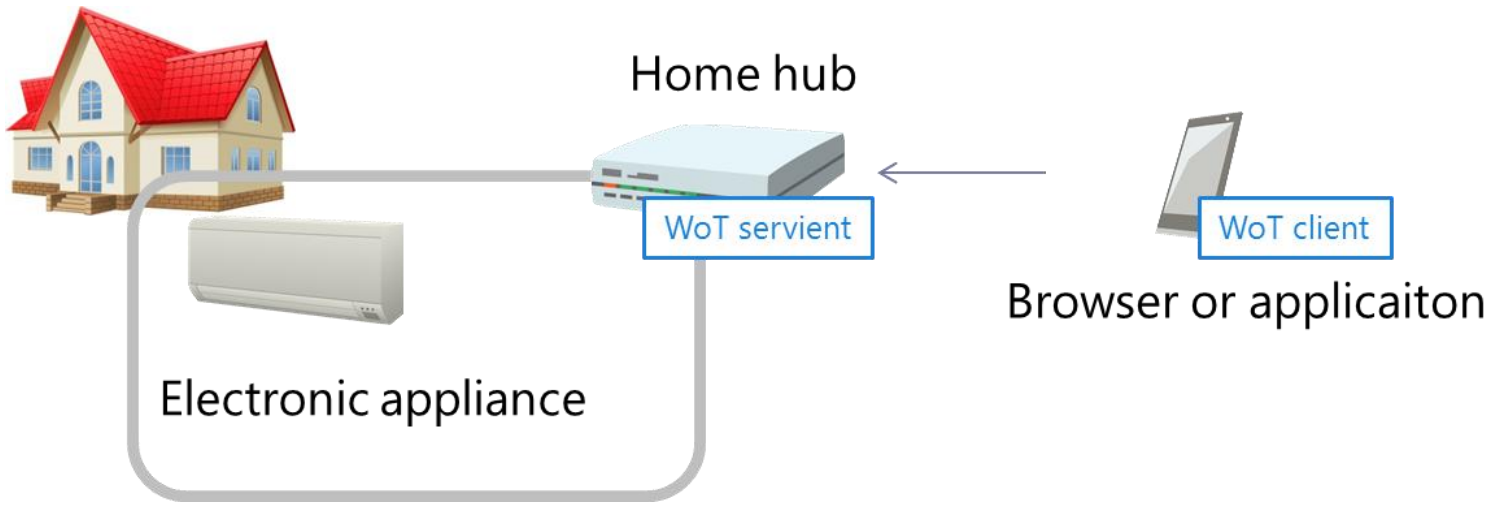
- Local discovery
- WoT server exposes “Interactions” through WoT Interface
- WoT client (UI or other Thing) interacts with WoT Interface

Remote Access



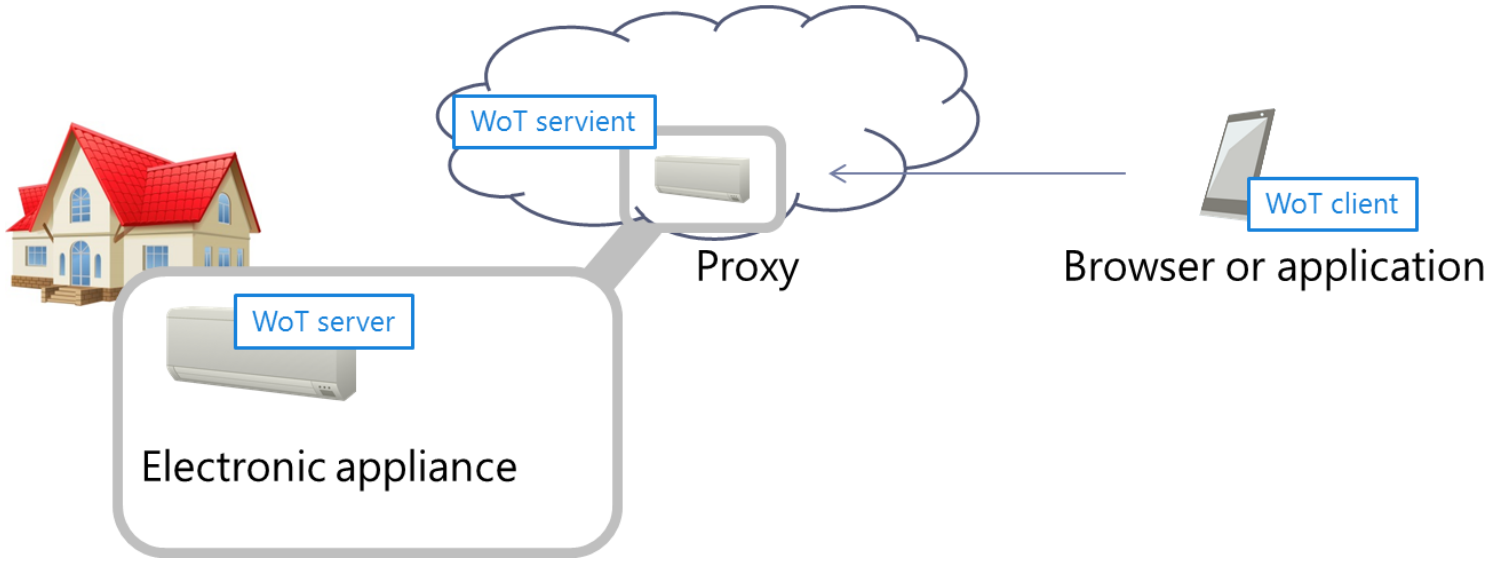
- Local and remote discovery
- Remote client gains access to local network (IPv6, NAT traversal, ...)

Remote Access: Integration Hubs



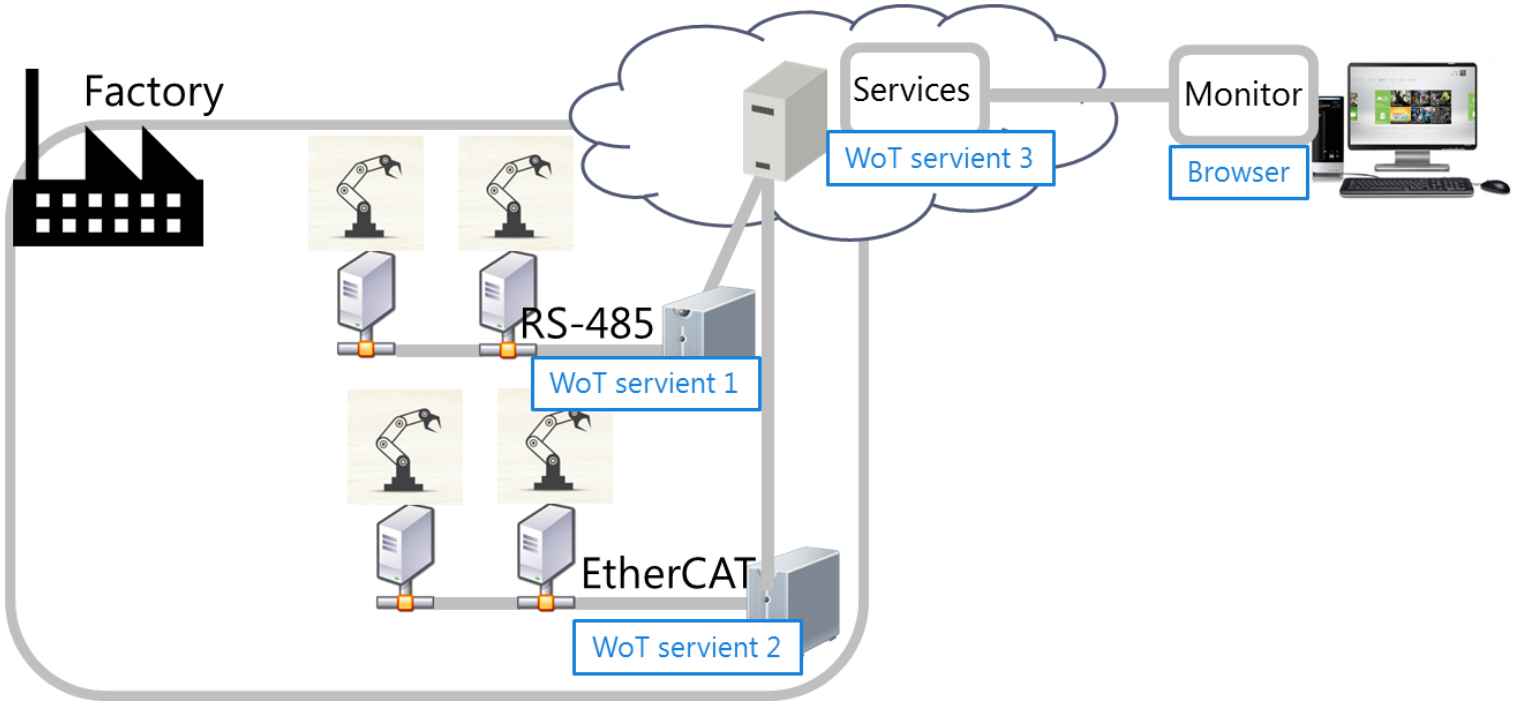
- Hub does local discovery and exposes Things
- Hub can provide virtual Things (e.g., rooms or sensor fusion)
- Hub can integrate and augment legacy devices (gateway)

Remote Access: Cloud Mirror



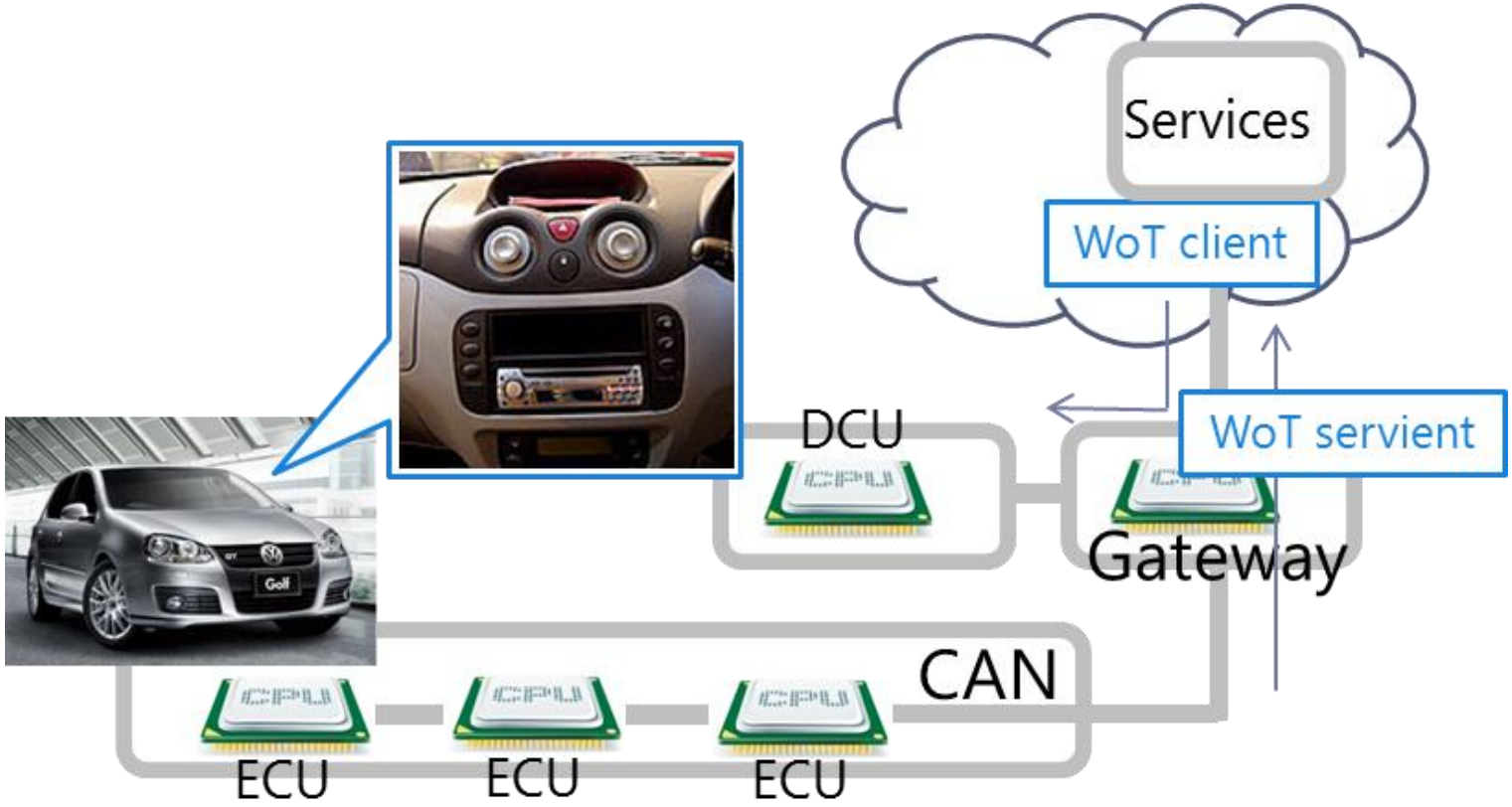
- WoT server is proxied in the cloud, which exposes Things
- Cloud can provide virtual Things (e.g., buildings or sensor fusion)
- Integrates with mobile app world and cloud-based IoT

Smart Factory / Industrial



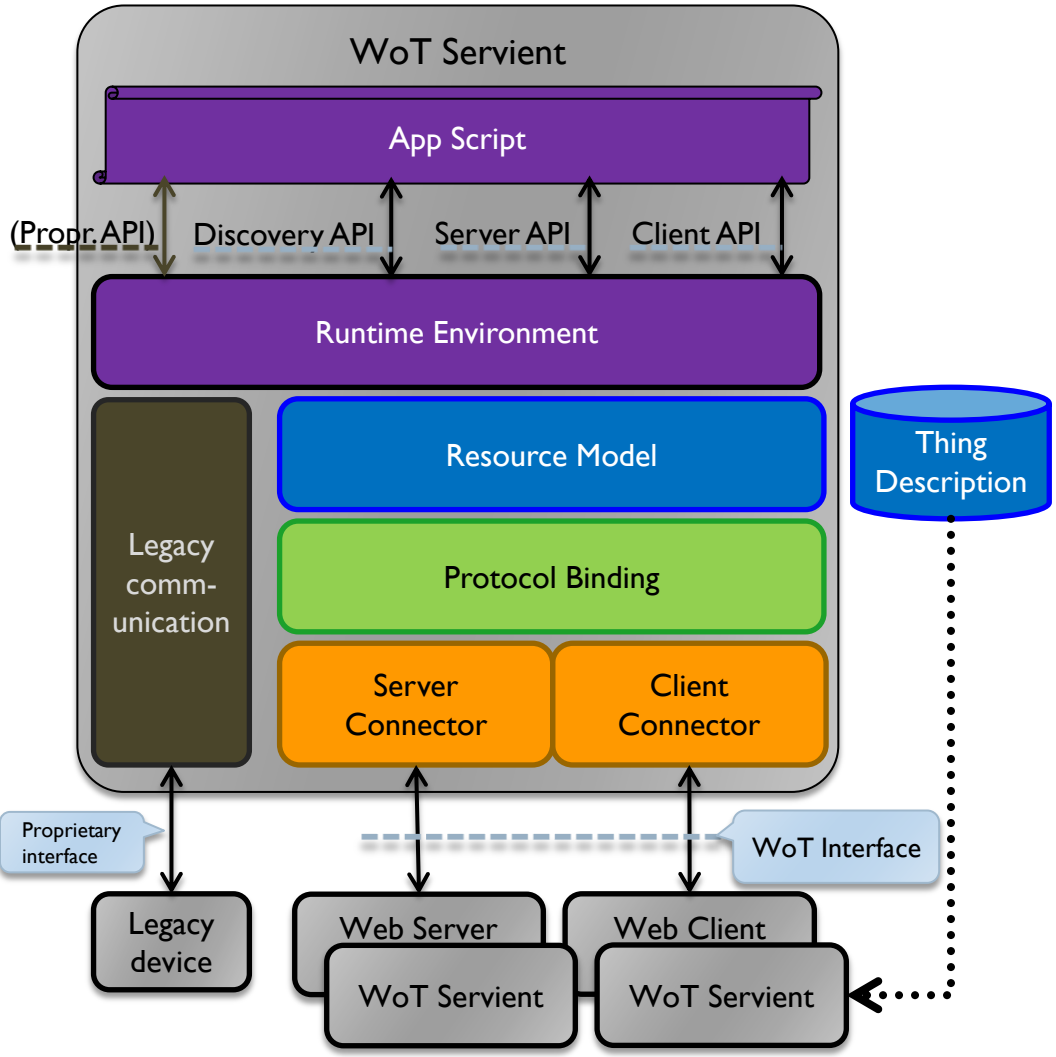
- WoT also targets requirements from industrial applications

Automotive



- WoT activity is in contact with W3C Automotive work

Thing Implementation: **WoT Servient**



Application Logic:

It can access local hardware, locally connected legacy devices, and remote things through the WoT Interface. For this, the runtime environment must provide the Scripting API (Physical, Client, Server).

Thing Description (TD):

Declares WoT Interface for interaction and provides (semantic) metadata for the Thing. TD is used by WoT clients to instantiate local software object of the Thing.

Resource Model:

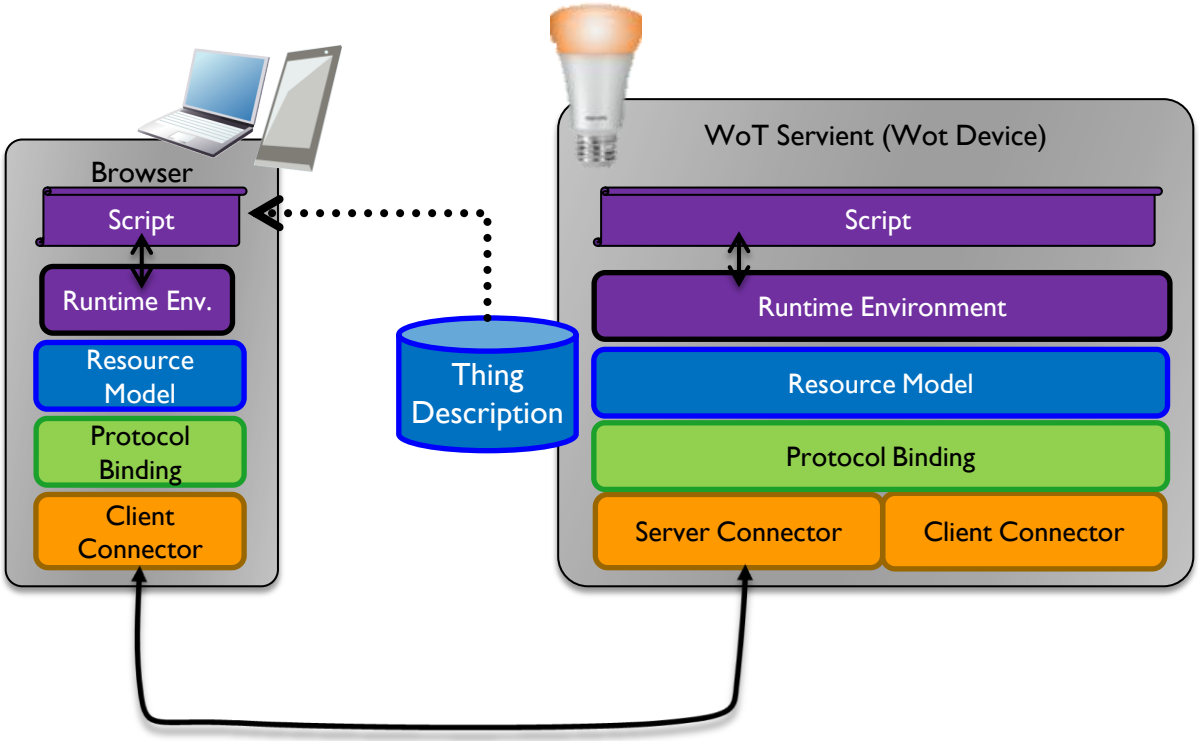
Provides a common abstraction across the different protocols. Just like the Web, it allows to identify and address interaction points with URIs.

Protocol Binding:

Converts interactions with Things using information in TD in accordance with lower-layer protocols to have client and server connectors.

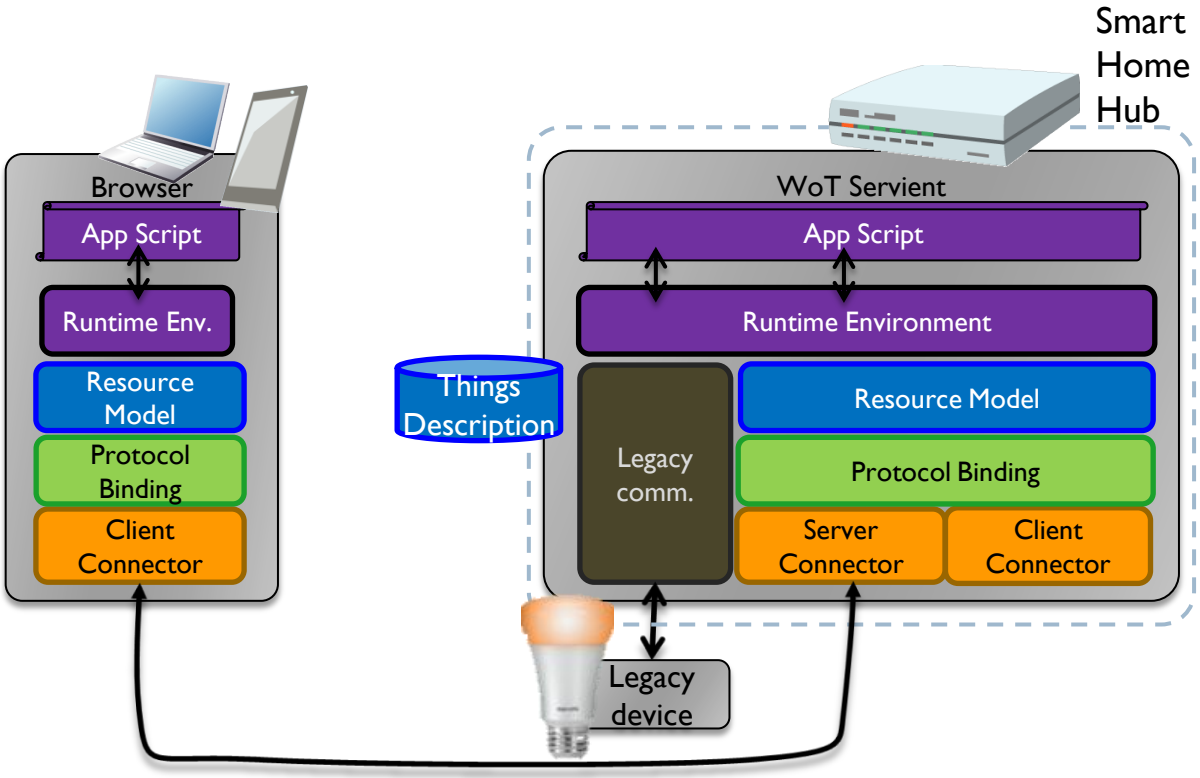
WoT Servient on Thing Itself

- Native WoT Things host a servient directly
- TD is provided by Thing directly



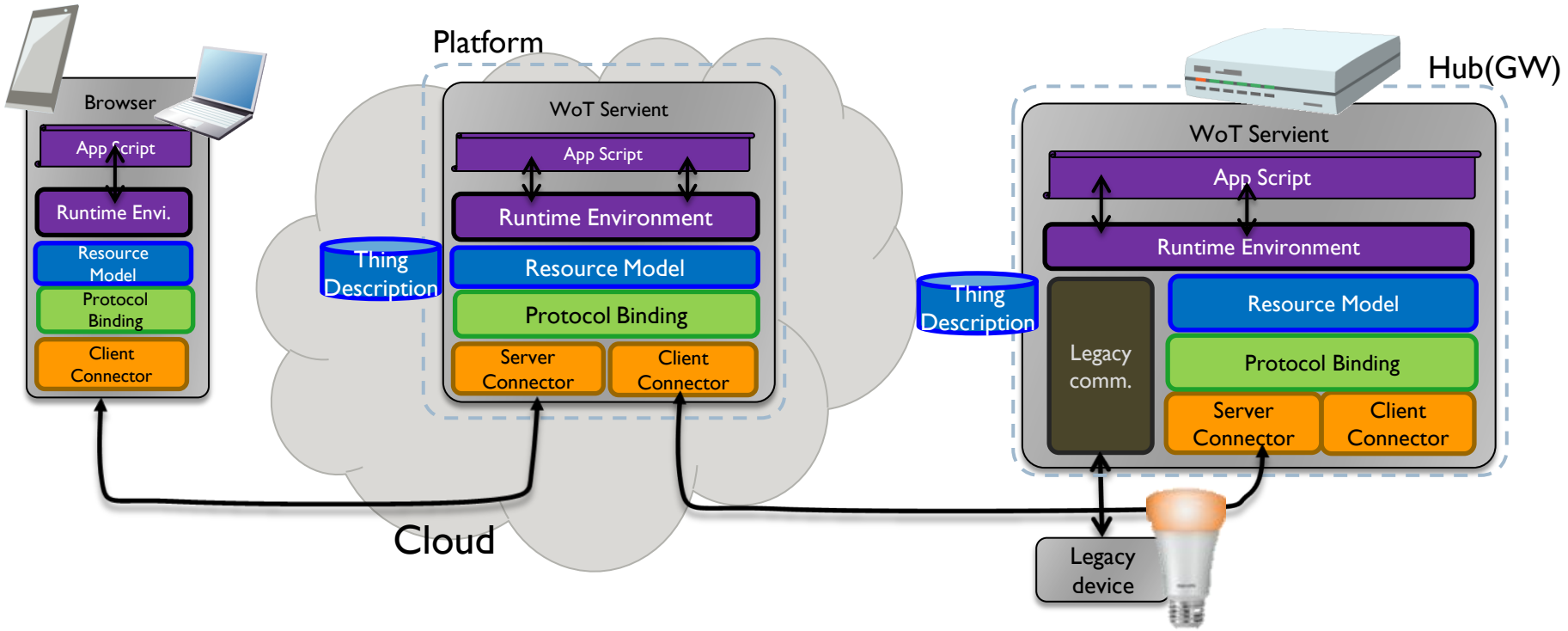
WoT Servient on Integration Hub

- WoT servients can run on hubs (e.g., smartphone, gateway)
- Can act as agent for legacy devices
- Multiple servients can be instantiated through sandboxed apps



WoT Servient in the Cloud

- A cloud mirror / device shadow can forward interactions
- Cloud mirror is synchronized with local servient



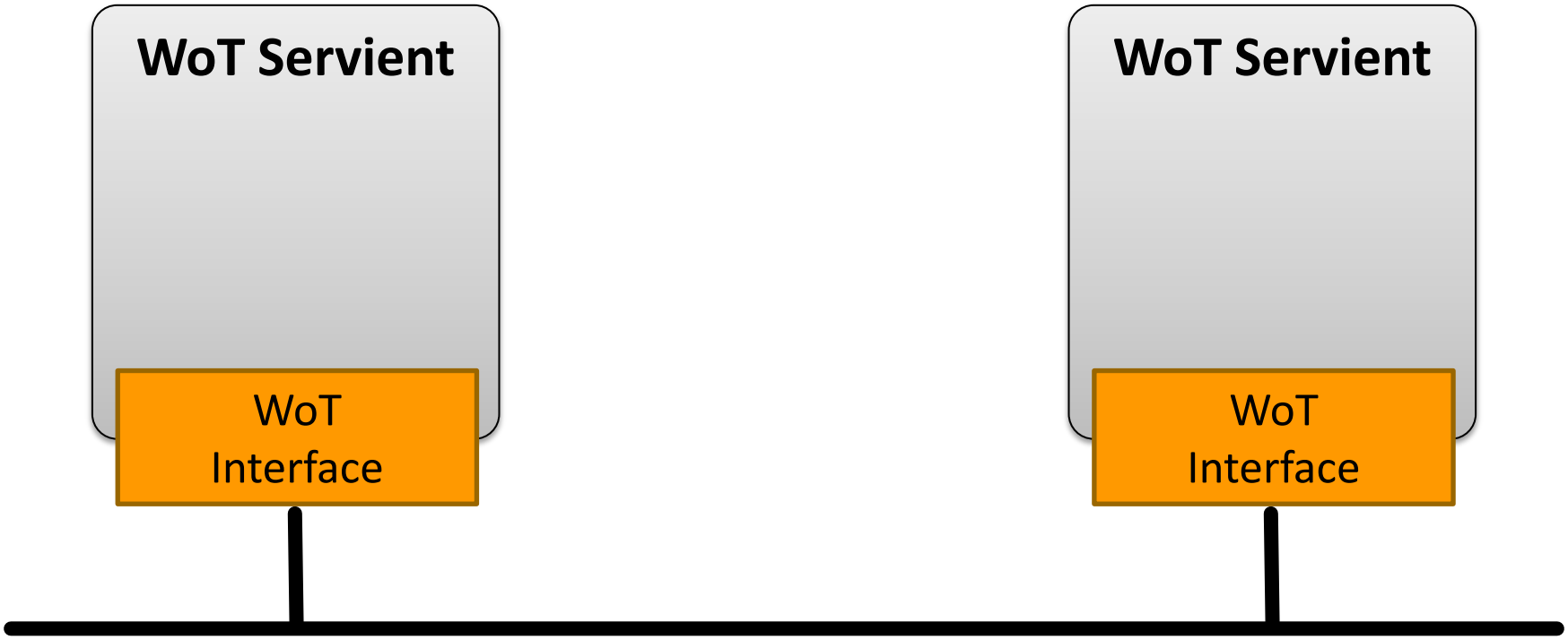
Protocol Bindings and the Web

http://w3c.github.io/wot/current-practices/wot-practices-beijing-2016.html#sec-wot-interface

WoT INTERFACE

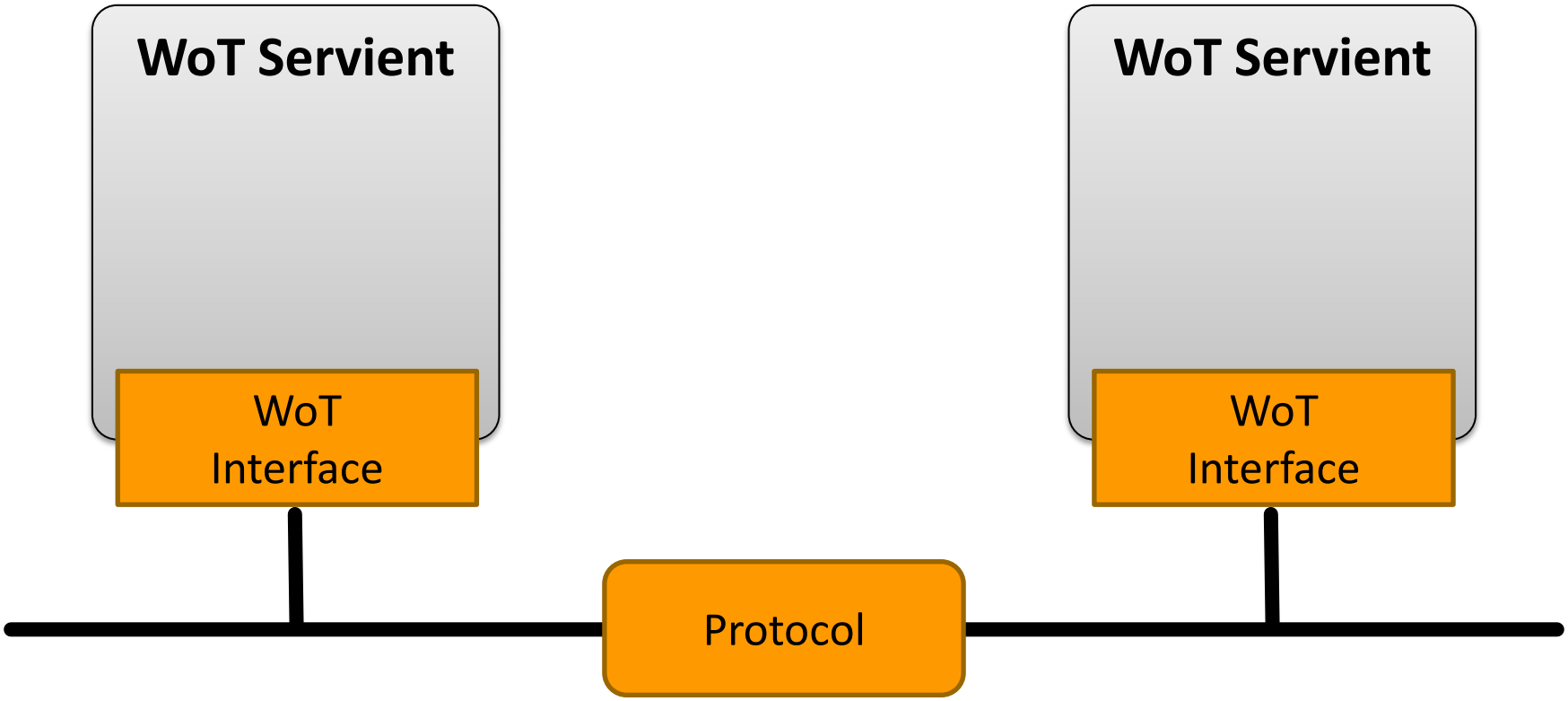
WoT Interface

- Interface exposed by servient to the network



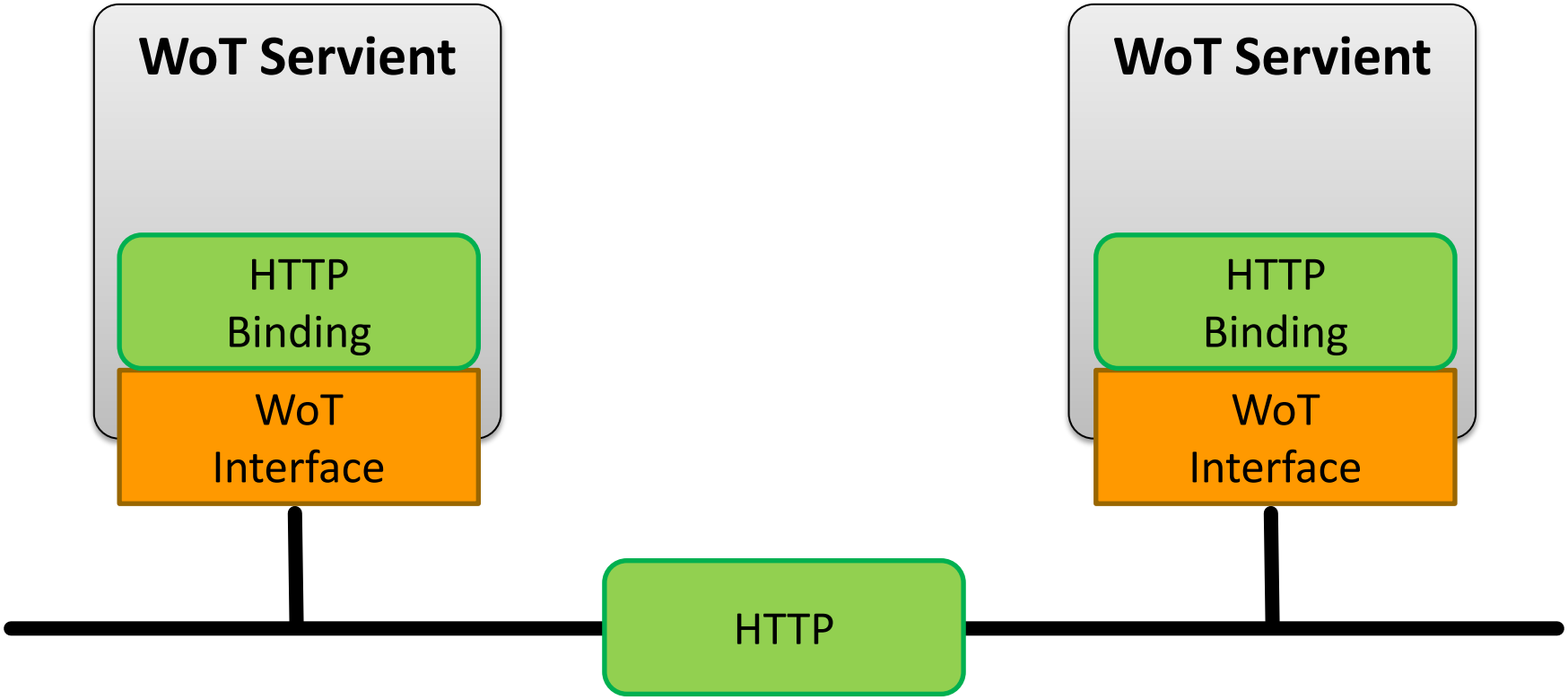
WoT Interface

- Interface exposed by servient to the network



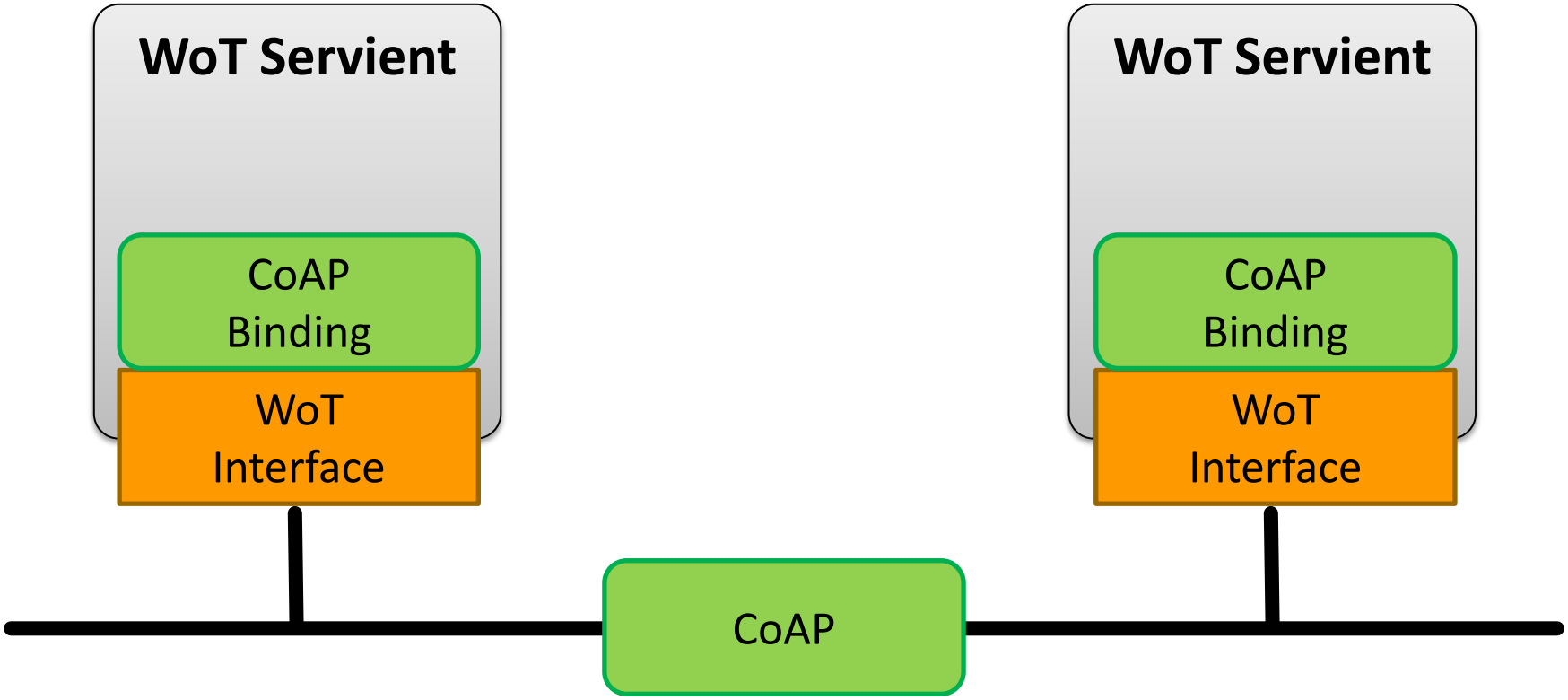
Protocol Bindings

- Interface can be bound to various protocols



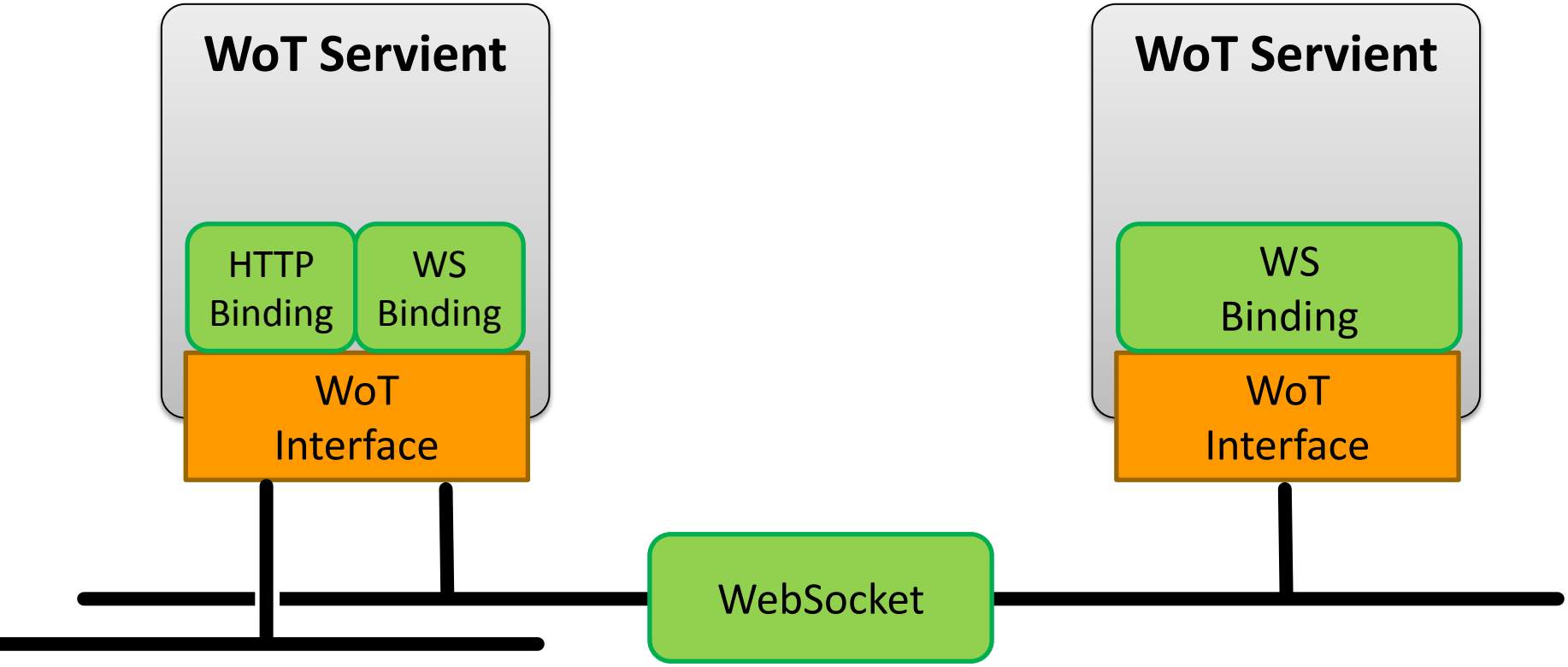
Protocol Bindings

- Interface can be bound to various protocols



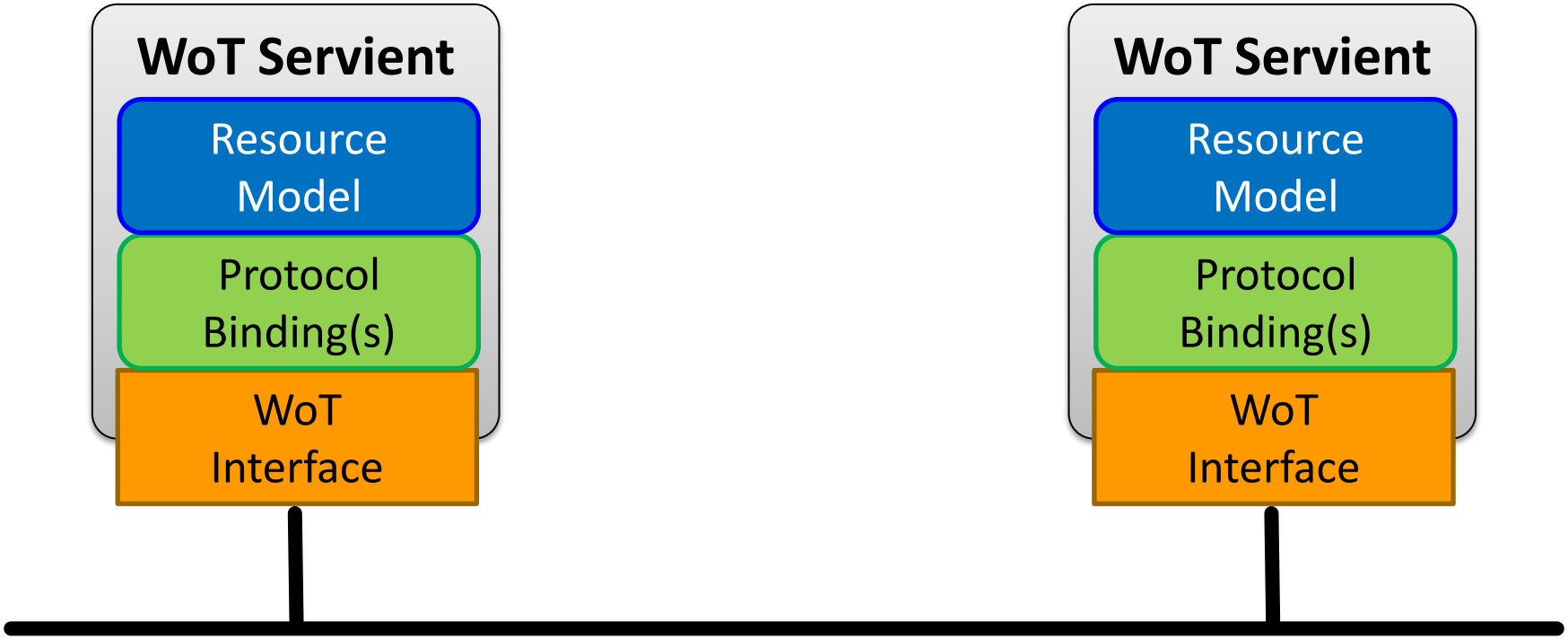
Protocol Bindings

- Multiple bindings possible



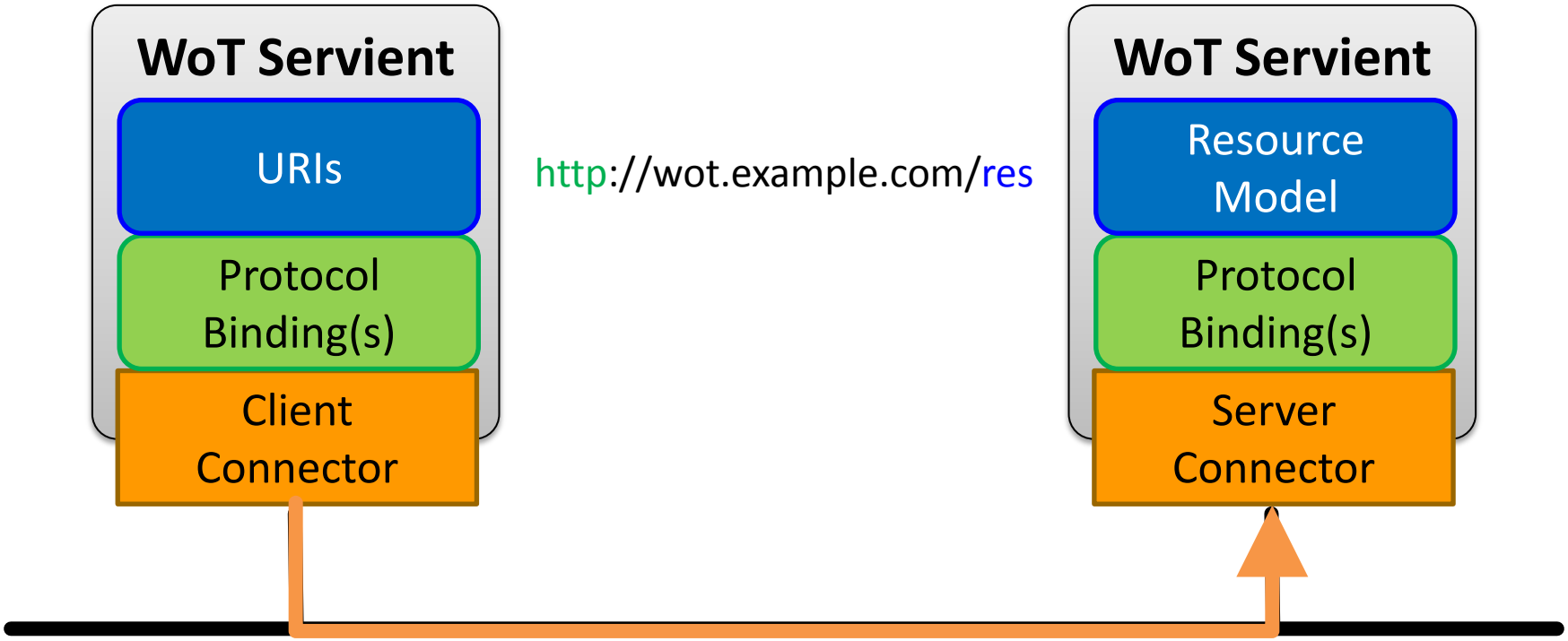
Resource Model

- Interaction points are Web resources



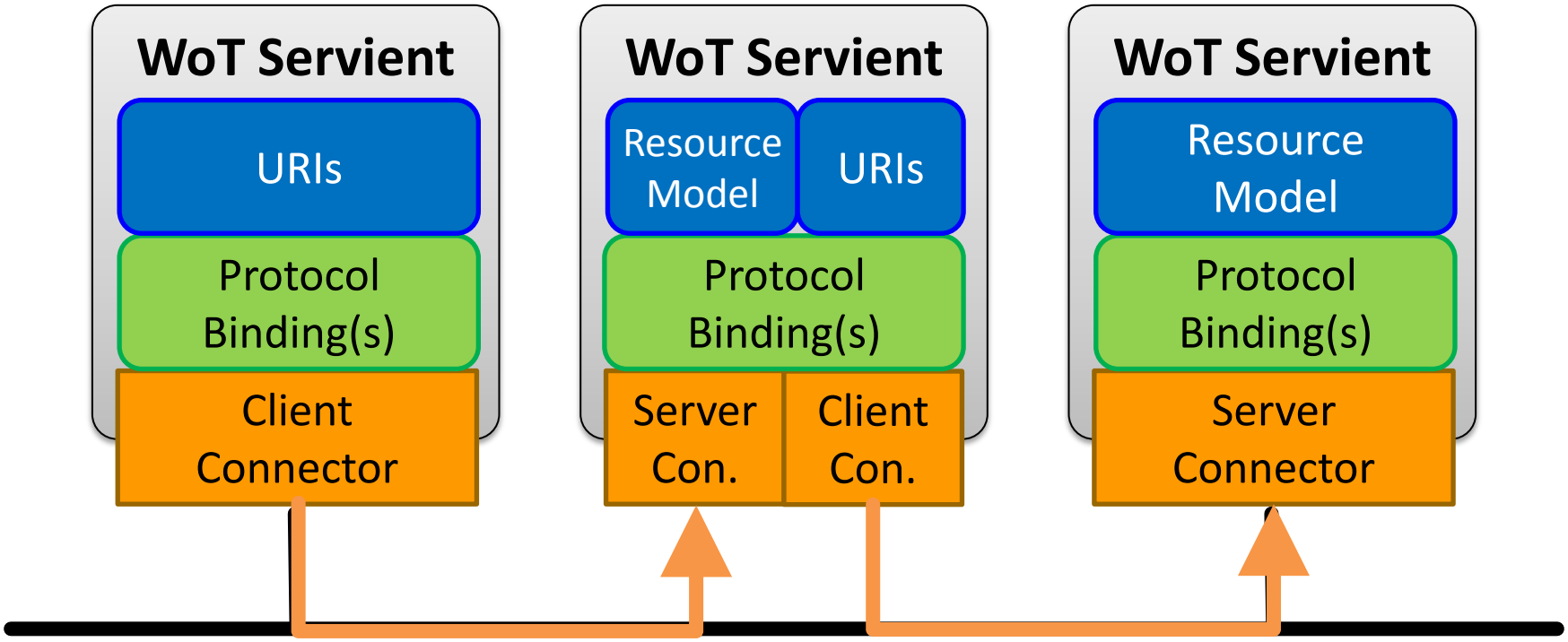
Servient Role

- Servient can act as client or server



Servient Role

- … or both at the same time



Metadata and Interactions

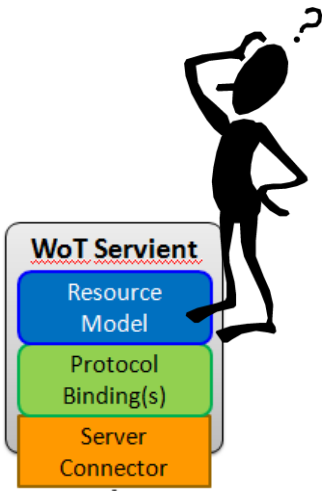
http://w3c.github.io/wot/current-practices/wot-practices-beijing-2016.html#thing-description

THING DESCRIPTION

I Want to Use a WoT Servient

What kind of data do you serve?

Who are you?



How can I access the data/function?

What kind of functions do you have?

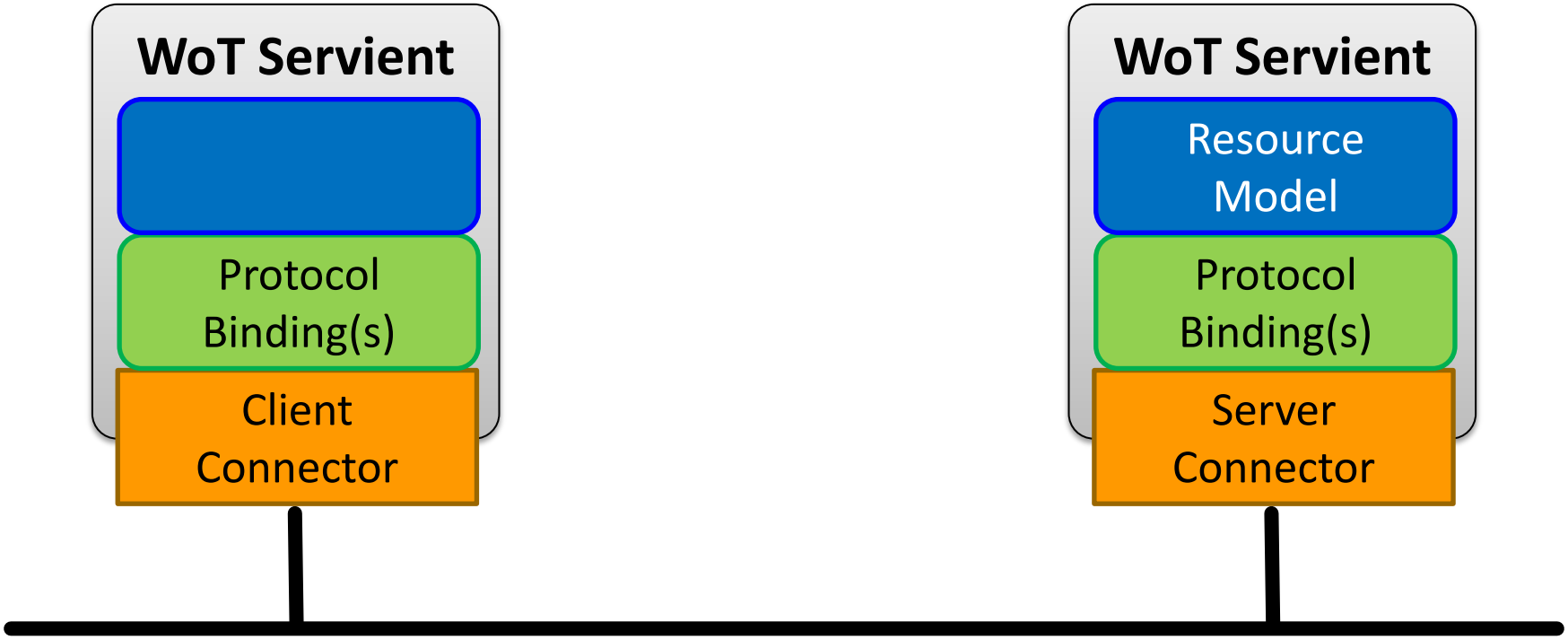
What kind of protocols/encodings do you support?

Are there some security constrains?

→ W3C Thing Description (TD)

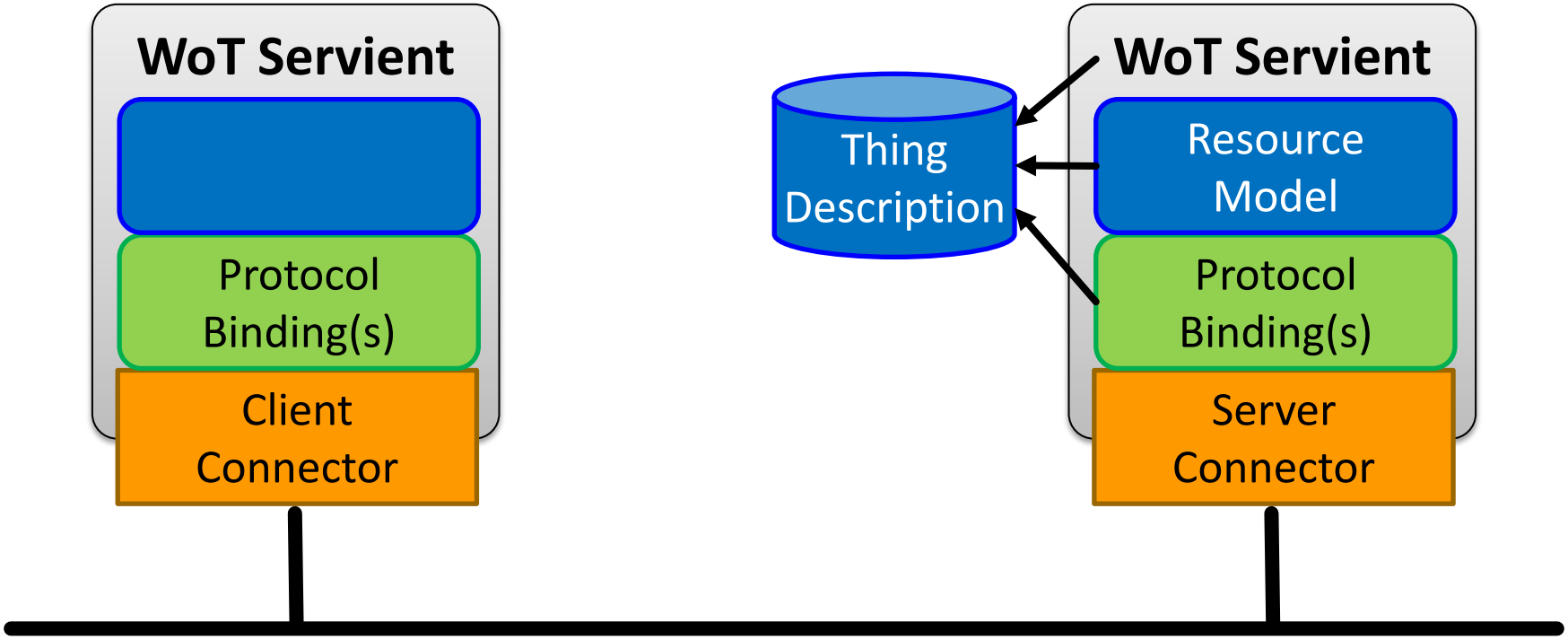
Thing Description

- Describes Thing metadata and interactions



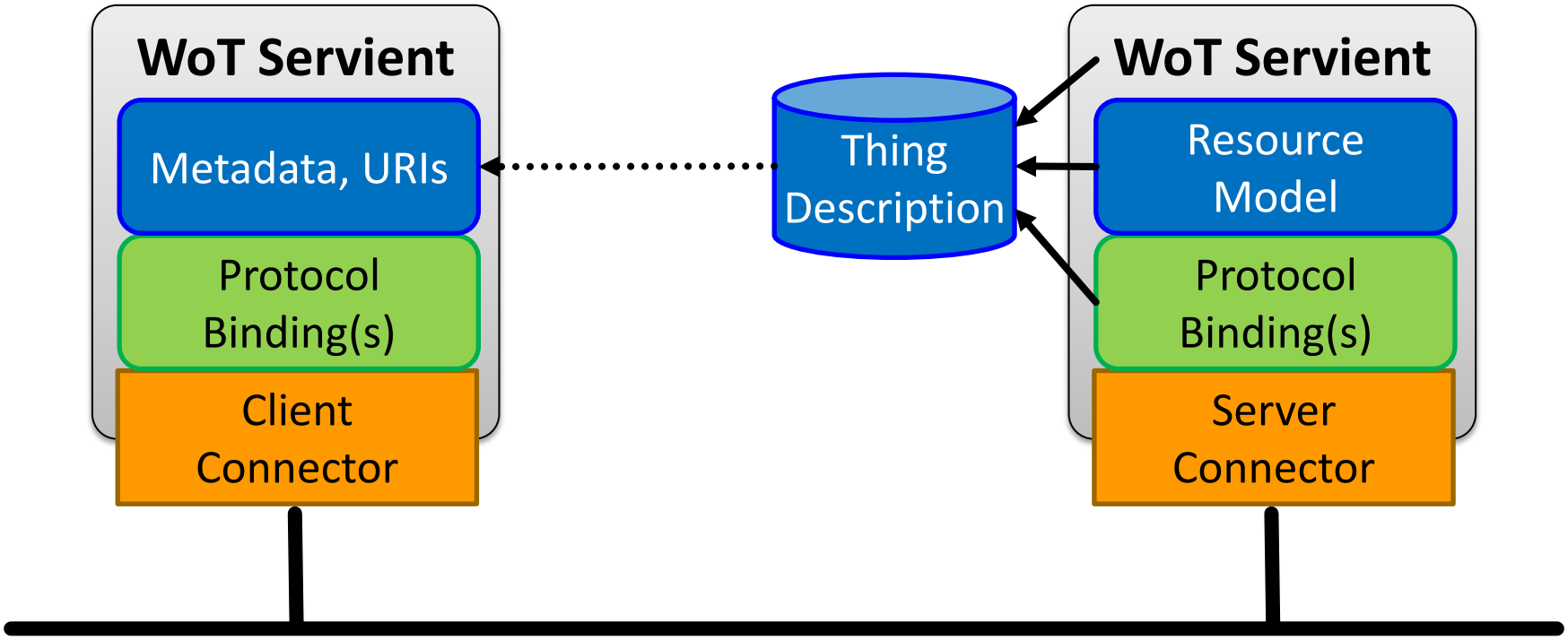
Thing Description

- Describes Thing metadata and interactions



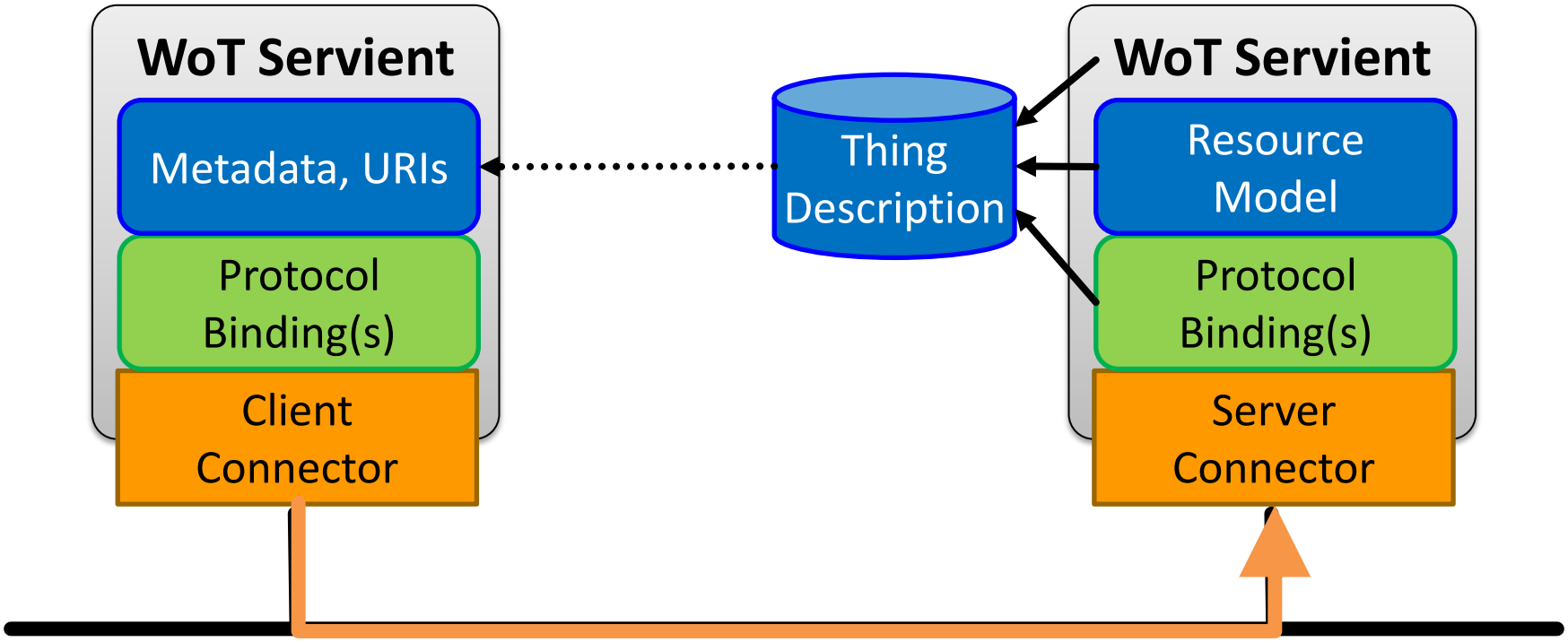
Thing Description

- Machine clients can understand WoT Interface



Thing Description

- Thing-to-thing communication



Describe your Thing based on JSON-LD

- Reach interoperability by a semantic description language
  - based on well established JSON format
  - enables machine interoperability by using (standardized) vocabularies from given *@context*
- JSON-LD is rooted in the RDF model
  - subject, predicate, object triples

TD Example

```
{
  "@context": [
    "http://w3c.github.io/wot/w3c-wot-td-context.jsonld",
    { "actuator": "http://example.org/actuator#" }
  ],

  "@type": "Thing",
  "name": "MyLEDThing",

  "uris": [
    "coap://myled.example.com:5683/",
    "http://mything.example.com:8080/myled/"
  ],

  "encodings": ["JSON", "EXI"],
  "security": {
    "cat": "token:jwt",
    "alg": "HS256",
    "as": "https://authority-issuing.example.org"
  },

  "properties": [
```

```
  "properties": [
    {
      "@type": "actuator:onOffStatus",
      "name": "status",
      "valueType": { "type": "boolean" },
      "writable": true,
      "hrefs": [ "pwr", "status" ]
    }
  ],
  "actions": [
    {
      "@type": "actuator:fadeIn",
      "name": "fadeIn",
      "inputData": {
        "valueType": { "type": "integer" },
        "actuator:unit": "actuator:ms"
      },
      "hrefs": [ "in", "led/in" ]
    },
    {
      "@type": "actuator:fadeOut",
      "name": "fadeOut",
      "inputData": {
        "valueType": { "type": "integer" },
        "actuator:unit": "actuator:ms"
      },
      "hrefs": [ "out", "led/out" ]
    }
```

Type System

- Default currently based on JSON Schema http://w3c.github.io/wot/current-practices/wot-practices-beijing-2016.html#type-system
- Best start with simple types
  - boolean
  - integer
  - number
  - String
- Other systems can be plugged in under "valueType"

How to Create a TD?

- Manually copy, paste, and modify
  - http://w3c.github.io/wot/current-practices/wot-practices-beijing-2016.html#td-examples
  - or look into the TD repository http://vs0.inf.ethz.ch:8080 (might be offline from time to time)
- Generate from development framework
  - Serialization based on the interactions provided

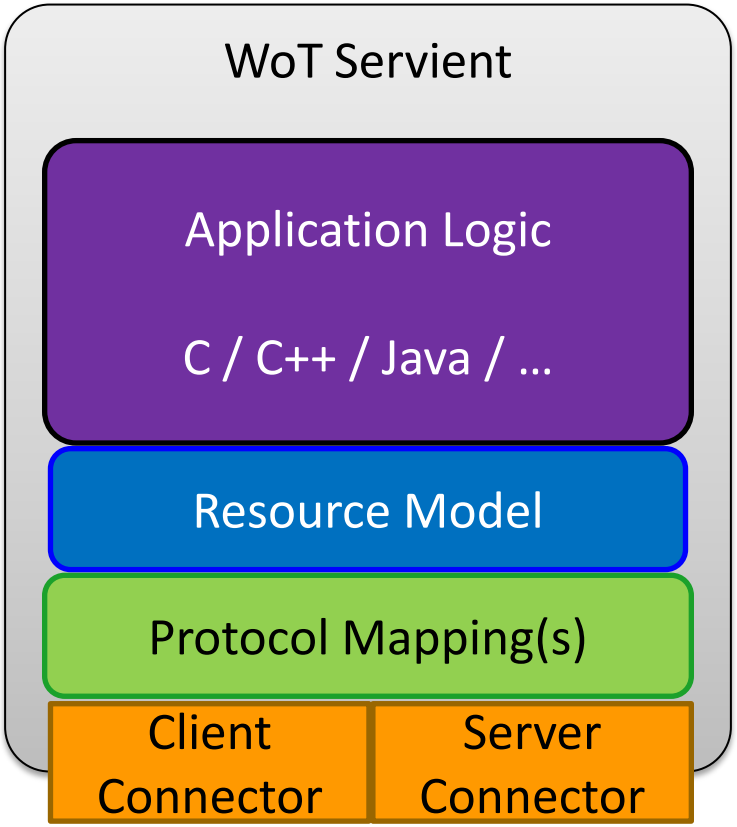
Runtime Environment and Portable Apps

http://w3c.github.io/wot/current-practices/wot-practices-beijing-2016.html#scripting-api

SCRIPTING API

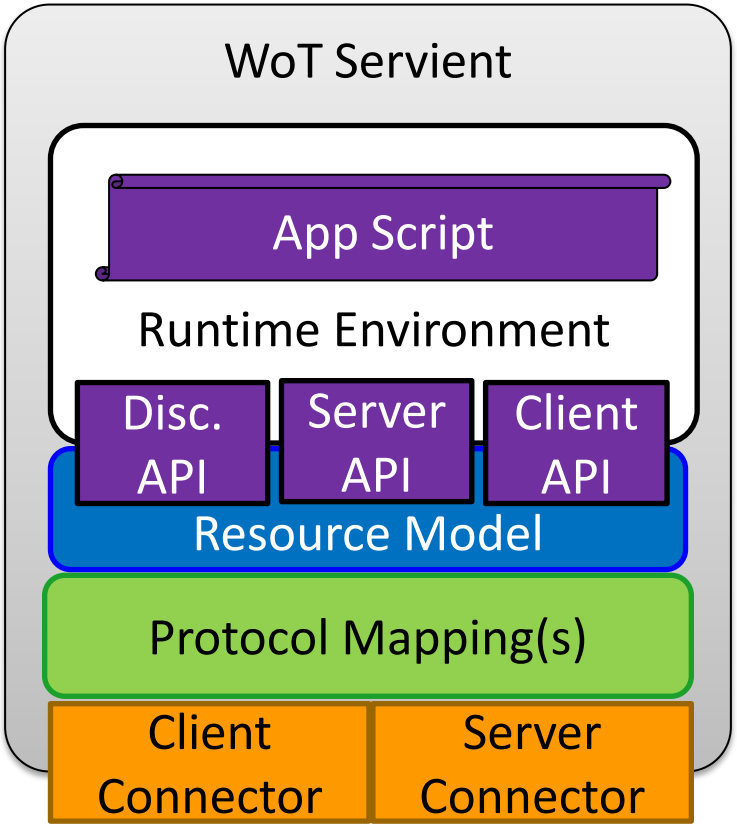
Without Scripting API

- Application logic often implemented natively



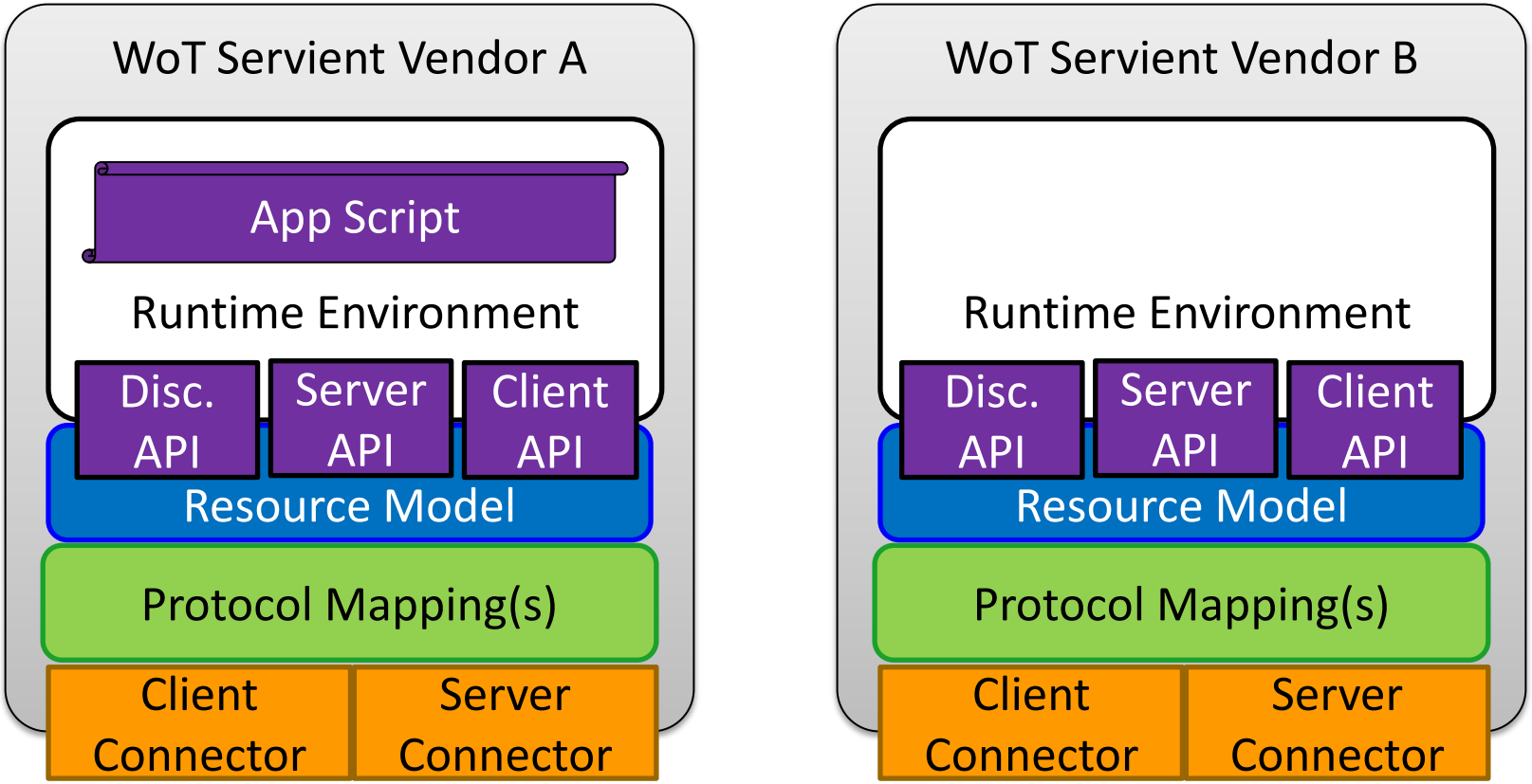
Scripting API

- Common runtime enables portable apps



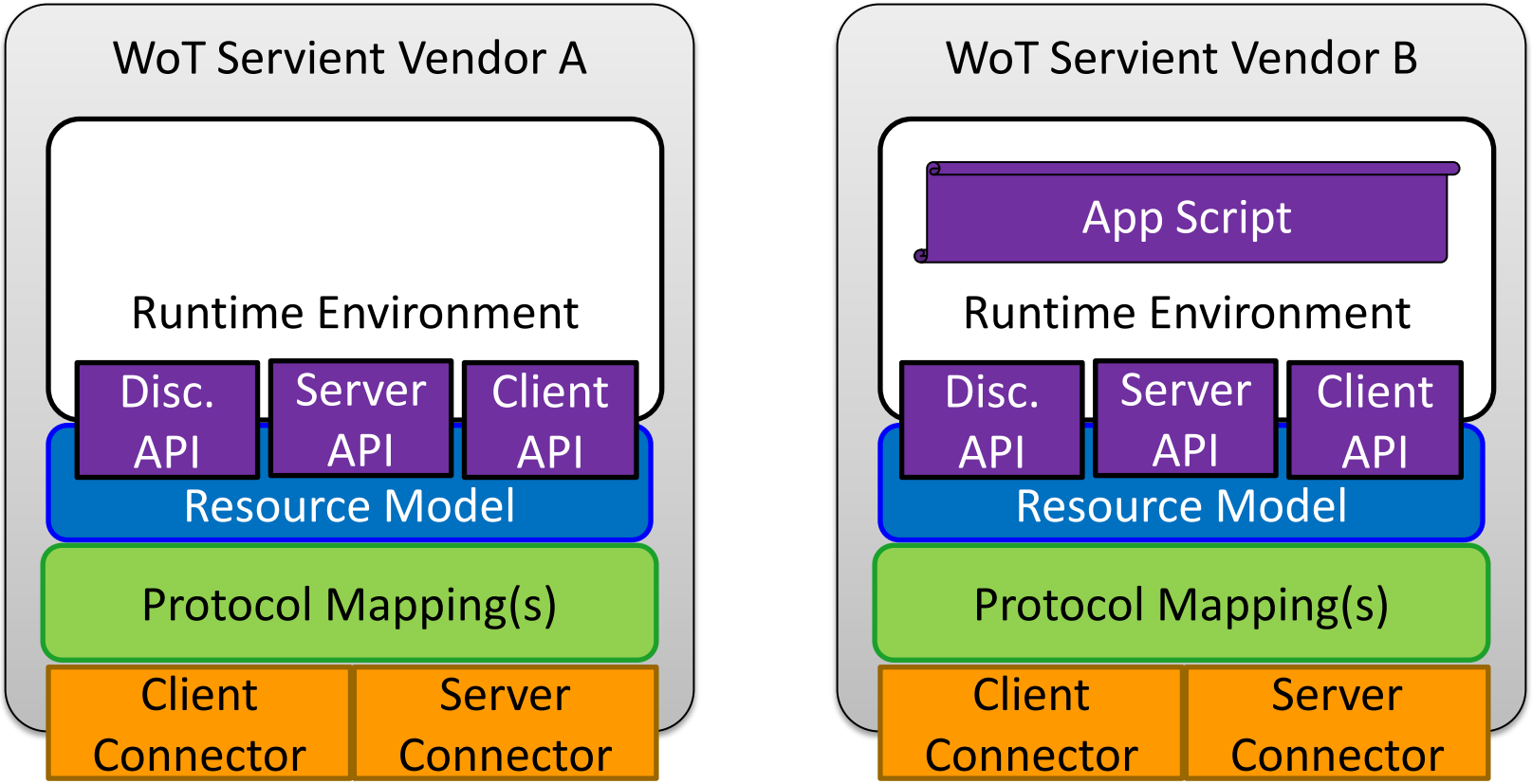
Scripting API

- Common runtime enables portable apps



Scripting API

- Common runtime enables portable apps



How to get started?

http://w3c.github.io/wot/current-practices/wot-practices-beijing-2016.html#participation-howto

PARTICIPATION HOWTO

Pick Your Servient Role

- Client Role
  - User interface
  - Machine agent
- Server Role
  - Sensor/actuator
  - Device simulators
- Both (“servient”)
  - Configurable client
  - Aggregator using other Things

Pick Your Platform

- Client Role
  - User interface — Angular.js and Web browser
  - Machine agent — Python, Ruby, Java, C++, …
- Server Role
  - Sensor/actuator — Arduino, ESP8266, mbed, …
  - Device simulators — Node.js, Java
- Both
  - Hub — Raspberry Pi, smartphone
  - Cloud proxy — Java

Pick Your Protocol(s)

- HTTP
  - Node.js, Jetty, RESTX.io, lighttpd, …
  - Platform-specific (Arduino, Contiki, NodeMCU, …)
- CoAP
  - Californium, node-coap, libcoap
  - Platform-specific (Contiki, mbed, NodeMCU, …)
  - http://coap.technology/
- Others? Design the binding!
  - e.g., MQTT: https://www.eclipse.org/paho/

Pick Your Logic Implementation

- Start with native application logic

- Once familiar, follow the Current Practices document for the Scripting API http://w3c.github.io/wot/current-practices/wot-practices.html

Online Resources

- Interest Group
  - https://www.w3.org/WoT/IG/
  - https://lists.w3.org/Archives/Public/public-wot-ig/ (subscribe to mailing list)
- Documents (for implementers)
  - http://w3c.github.io/wot/architecture/wot-architecture.html
  - http://w3c.github.io/wot/current-practices/wot-practices.html (living document)
    Beijing 2016 Release:
    http://w3c.github.io/wot/current-practices/wot-practices-beijing-2016.html
- GitHub (documents and proposals)
  - https://github.com/w3c/wot
- Wiki (organizational information: WebConf calls, Face-to-Face meetings, …)
  - https://www.w3.org/WoT/IG/wiki/Main_Page
- WoT Projects (implementing WoT Current Practices)
  - https://github.com/thingweb/
  - https://github.com/mkovatsc/wot-demo-devices
  - Please add yours!