IoTivity: Cloud Native Architecture and the Internet of Open Source Things

Peter Moonki Hong, Ph.D.

Samsung Electronics

Messages in our talk today…

Keys in Internet of Things (IoT) are **Interoperability** and **Contribution**

- That can be achieved by **Open Source** and **Open Standard**

IoTivity and the **Open Interconnect Consortium** (OIC) provide

- The **Communication Protocol** with all types of data exchanges among **Devices**, **Things**, and the **Cloud**

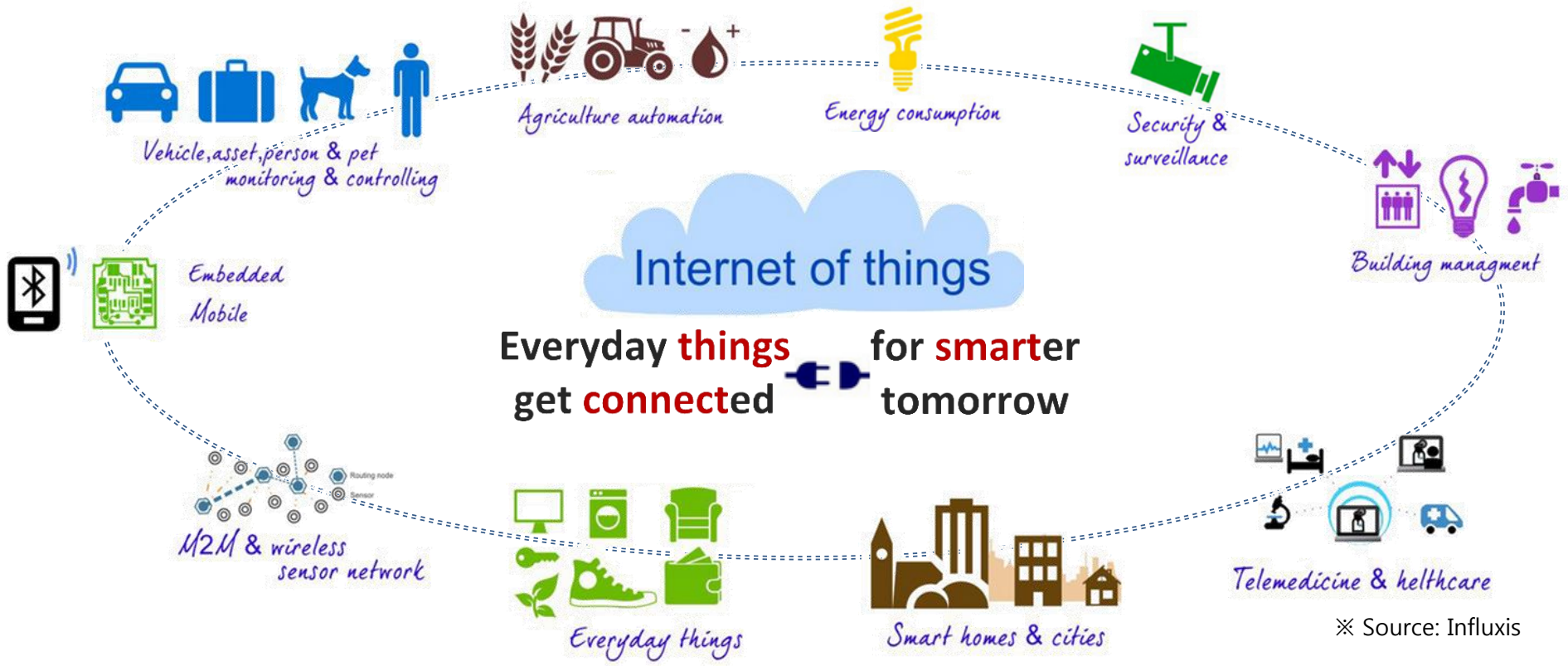
Features provided by IoTivity

- Reference implementation of the **OIC Specification Plus Alpha** that are purely encouraged in **open source manner**

And **TECHNICAL DETAILs** that IoTivity is providing and will provide!

Definition of IoT?

Network of networks? More data?
Operation with better efficiency? Things re-invention? …



※ Source: Influxis

INTERNET collaborating with **BILLIONS of VARIOUS THINGS** !

The Prerequisite is…

INTER + NETWORKING without limitation, providing

- Connectivity agnostic, platform transparent, scalable communication protocol with low memory footprint compatible to **ALL TYPES of THINGS**
- Of course, technical perfection is important, but…

Open for Contribution

Best Efforts

Code Base Interoperability

Which is the same approach as IETF RFCs

Our Approach Is To…

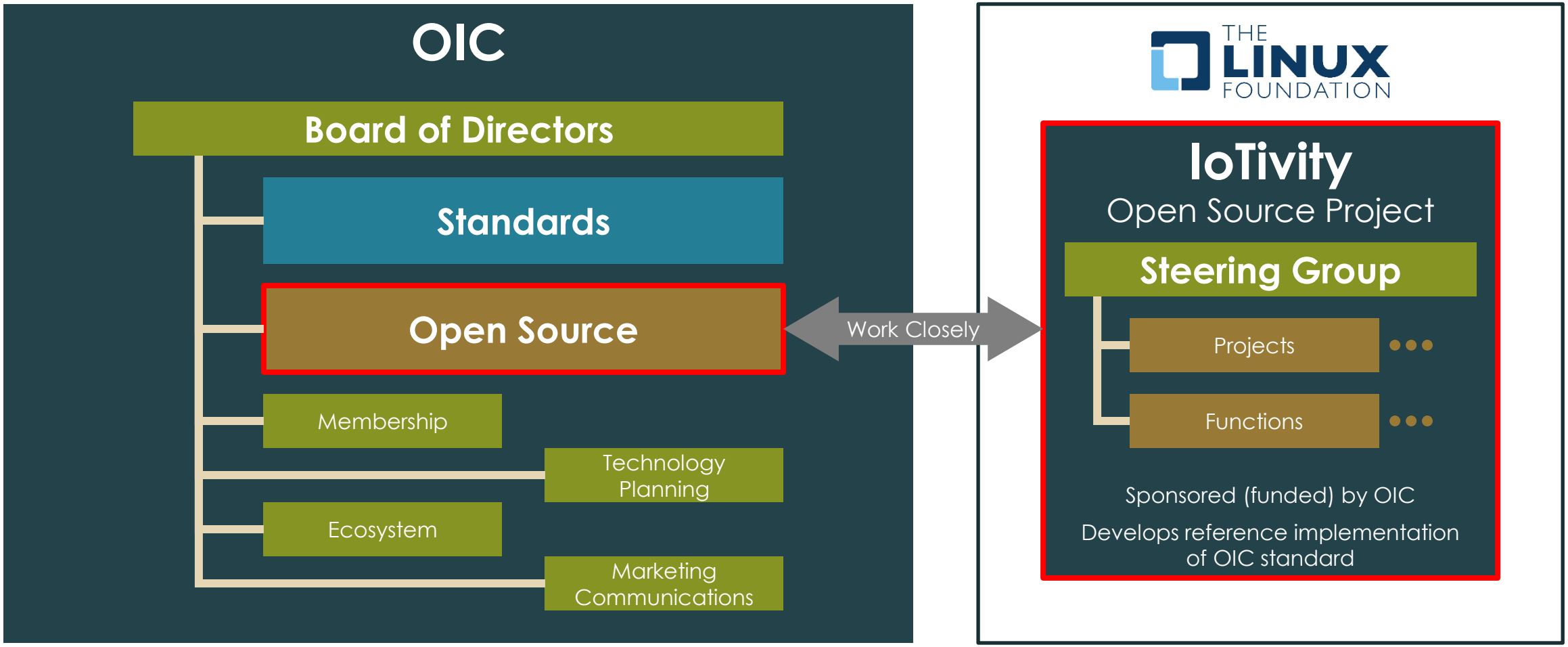
Launch **OPEN SOURCE** and **OPEN STANDARD**

- Achieving Open Contribution and Code-Based Interoperability at the same time

IoTivity and the **Open Interconnect Consortium** (OIC) have been launched in 2014!

- Specify and Implement the Communication Protocol that all types of data exchanges can be supported among Devices, Things, and the Cloud
- Provide **Specification**, **Open Source code**, and **Certification Program** with **developers-friendly IPR Policy** accelerating deployment and contribution

High Level OIC/IoTivity Governance



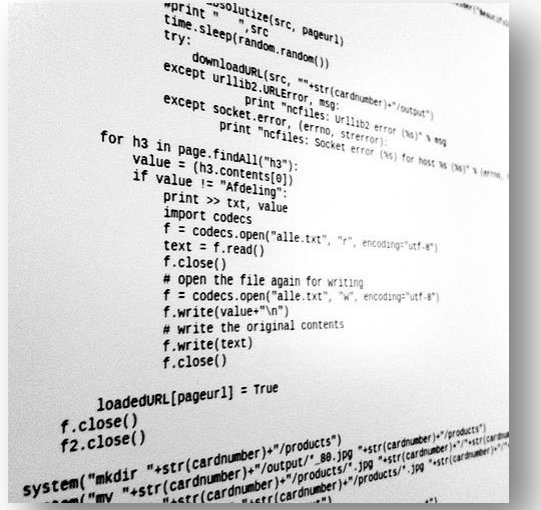
Scope of Availability

IoTivity code that is **COMPLIANT** to the OIC Spec is **OPEN to EVERYBODY**!

- Source code: **Apache 2.0** license
- OIC Spec compliant reference implementation + ALPHA

*"As a guaranteed S/W, I want to **EMPLOY** IoTivity for my product!"*

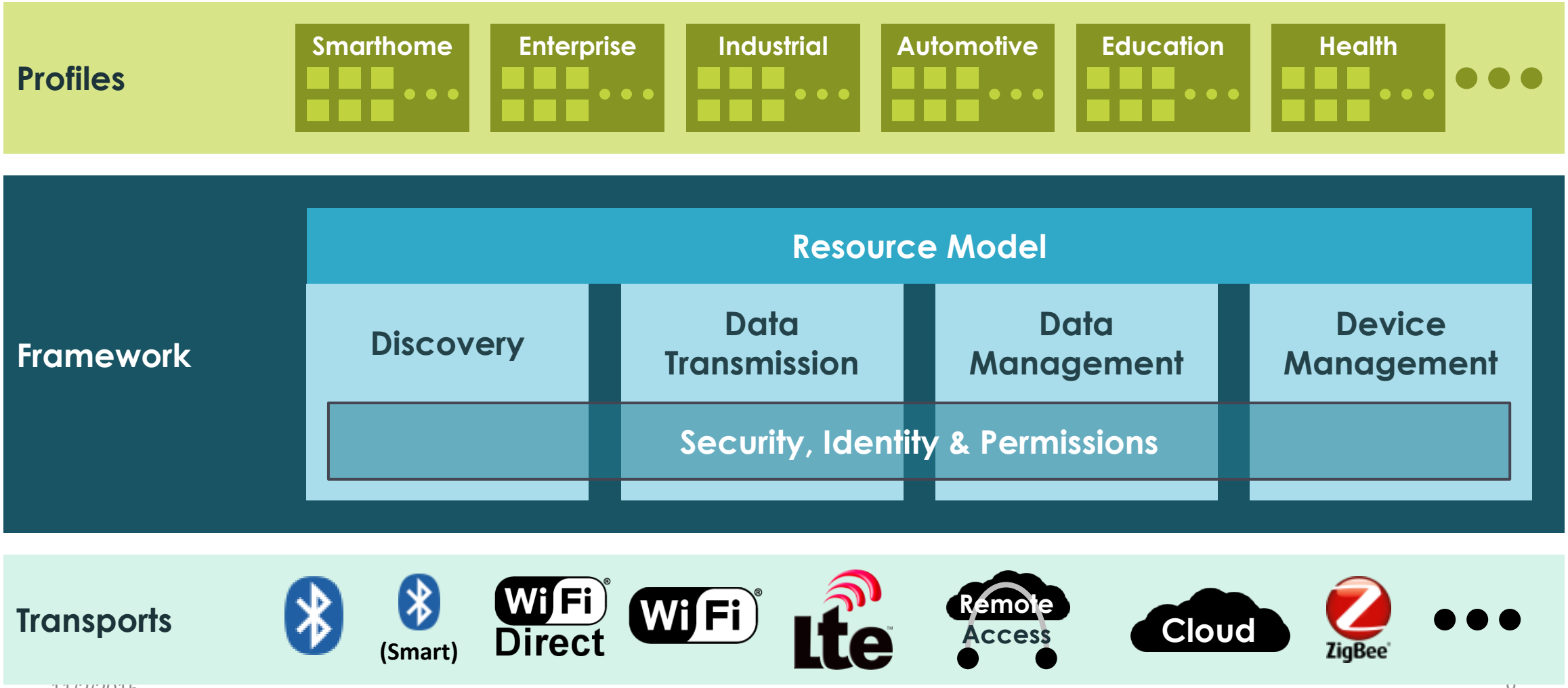
"What does IoTivity look like?"



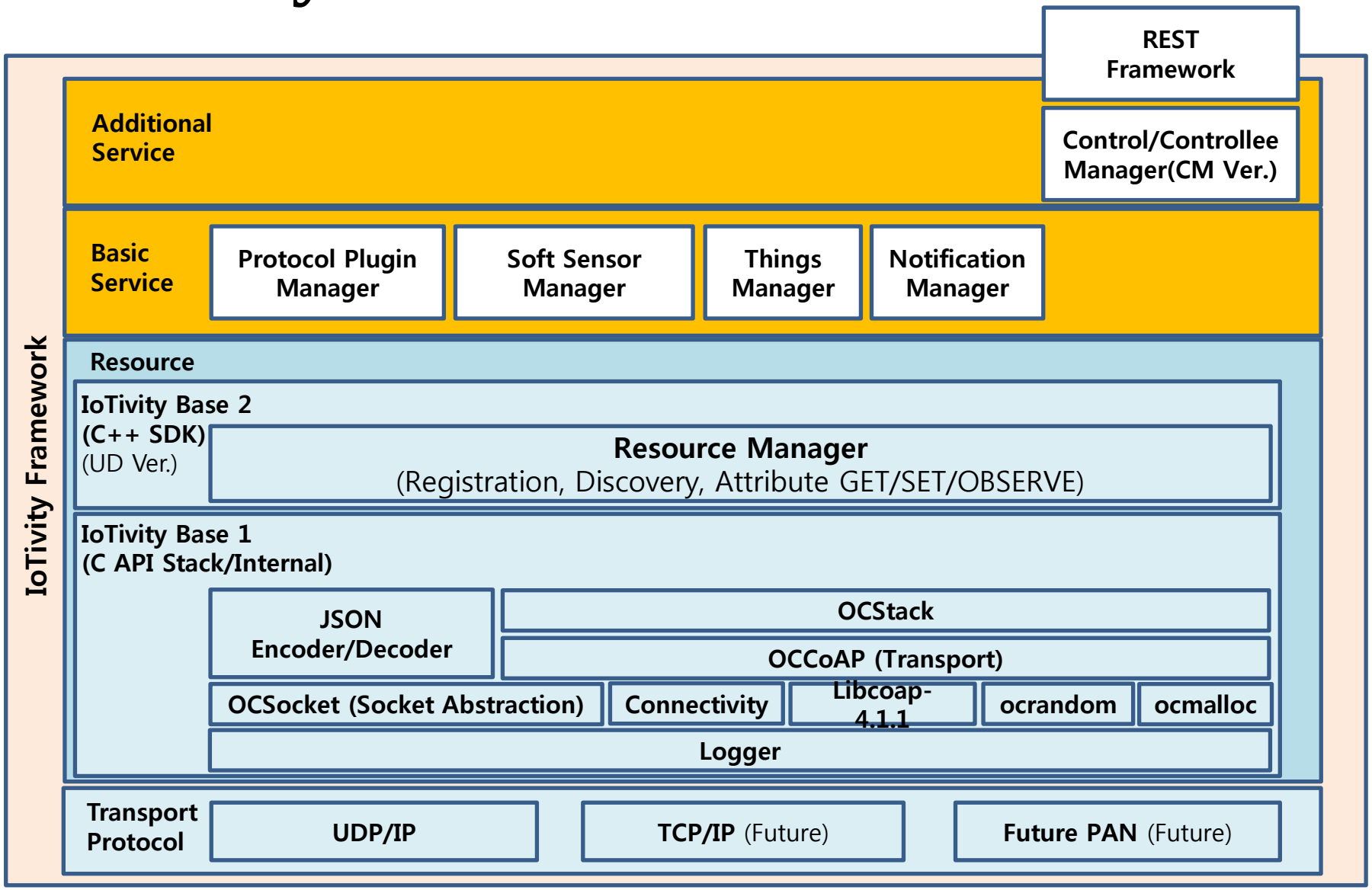
*"I want to **CONTRIBUTE** to the IoTivity implementation!"*

Then, What Does IoTivity Look Like?

Conceptual Framework:



IoTivity S/W Stack in Detail



IoTivity Base (Resource)

RESTful architecture representing all stuff as **RESOURCE**

- Primary message protocol:
  CoAP (Constrained Application Protocol) in IETF
- **CRUD&N** interaction between Client and Server
- **Application Layer** protocol with Connectivity agnostic, Platform transparent
- **IP/UDP-based**, but TCP-based and even non-IP transport are also provided!
  (e.g. Bluetooth, ZigBee, etc.)
- Guaranteeing end-to-end security per packet employing **DTLS, ACL**, etc.

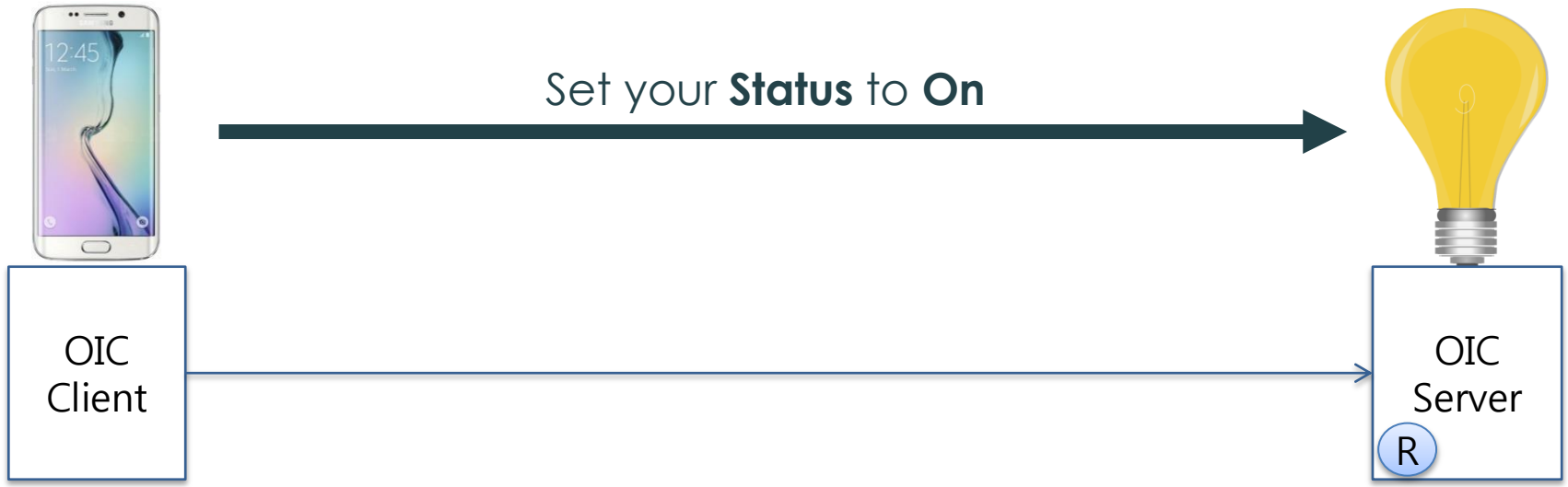
Corresponding IoTivity Project:

- **Discovery & Connectivity** Project
- **Security** Project

Features Provided from IoTivity Base

With respect to the interaction between **CLIENT** and **SERVER**

- Resource registration (Server)
- Resource discovery (Client)
- Device discovery with filtering (Client) – e.g. GET /oc/core?rt=light
- Property manipulation (get/set/observe) (Client/Server)
- Hierarchic resource (resource with sub-resources)



ModeType: Server / Client / Both

Provides API to select the role:

simpleserver

```
// Create PlatformConfig object
PlatformConfig cfg {
    OC::ServiceType::InProc,
    OC::ModeType::Server,
    "0.0.0.0", // By setting to "0.0.0.0", it binds to all avail
    0,         // Uses randomly available port
    OC::QualityOfService::LowQos
};

OCPlatform::Configure(cfg);
try
{
    // Create the instance of the resource class
    // (in this case instance of class 'LightResource').
    LightResource myLight;

    // Invoke createResource function of class light.
    myLight.createResource();
    std::cout << "Created resource." << std::endl;

    myLight.addType(std::string("core.brightlight"));
    myLight.addInterface(std::string("oc.mi.ll"));
    std::cout << "Added Interface and Type" << std::endl

    // A condition variable will free the mutex it is given, ther
    // intensive block until 'notify' is called on it.  In this
    // don't ever call cv.notify, this should be a non-processor
    // of while(true);
    std::mutex blocker;
    std::condition_variable cv;
    std::unique_lock<std::mutex> lock(blocker);
    std::cout <<"Waiting" << std::endl;
    cv.wait(lock);
```

Resource creation

simpleclient

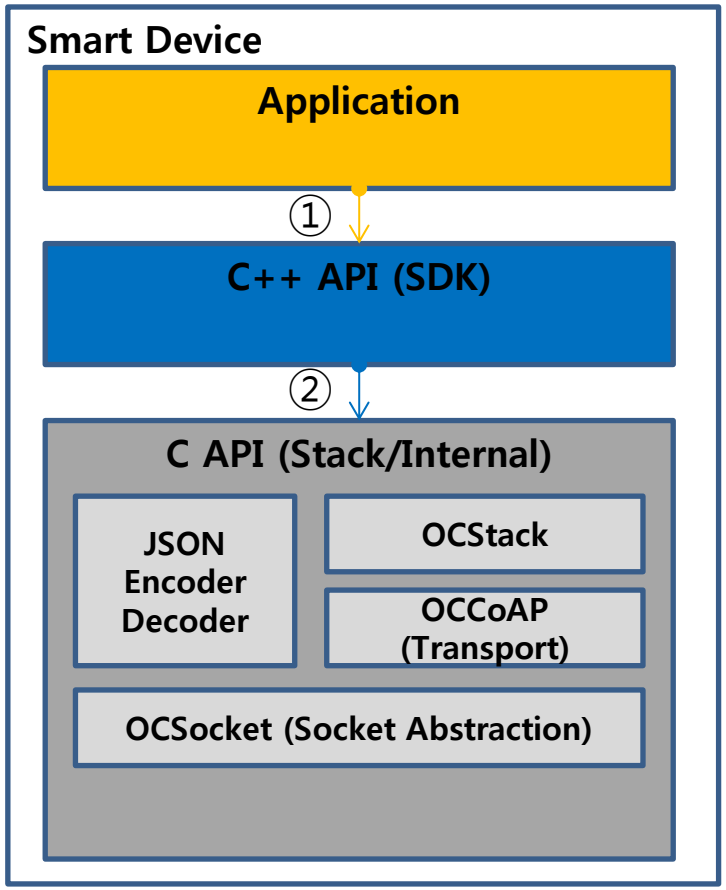
```
// Create PlatformConfig object
PlatformConfig cfg {
    OC::ServiceType::InProc,
    OC::ModeType::Client,
    "0.0.0.0",
    0,
    OC::QualityOfService::LowQos
};

OCPlatform::Configure(cfg);
try
{
    // makes it so that all boolean values are printed as 'true/false' in this stream
    std::cout.setf(std::ios::boolalpha);
    // Find all resources
    OCPlatform::findResource("", "coap://224.0.1.187/oc/core?rt=core.light", &foundResource);
    std::cout<< "Finding Resource... " <<std::endl;

    // A condition variable will free the mutex it is given, then do a non-
    // intensive block until 'notify' is called on it.  In this case, since we
    // don't ever call cv.notify, this should be a non-processor intensive version
    // of while(true);
    std::mutex blocker;
    std::condition_variable cv;
    std::unique_lock<std::mutex> lock(blocker);
    cv.wait(lock);
```

Resource finding

Registering a Resource

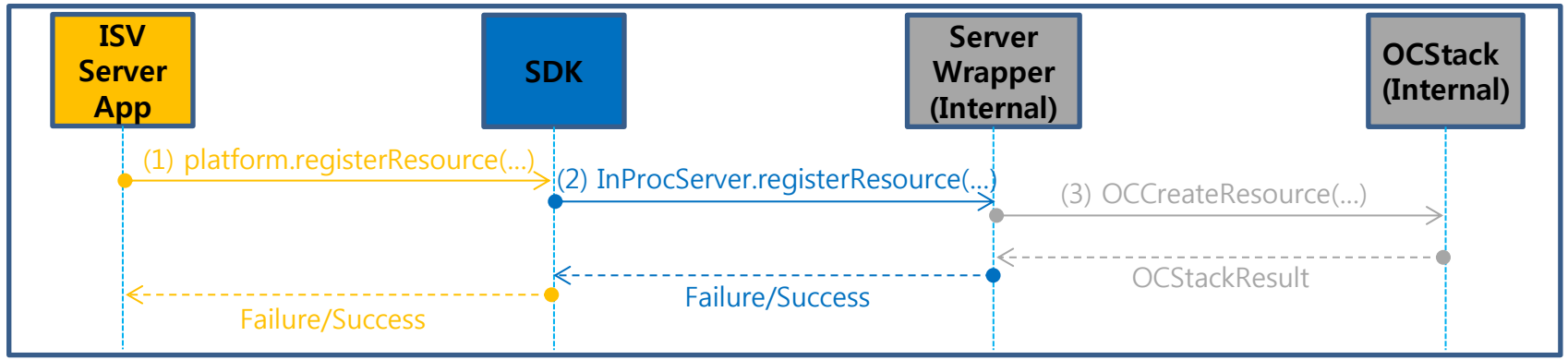


(1) platform.registerResource("/light/1",rt,if,...)

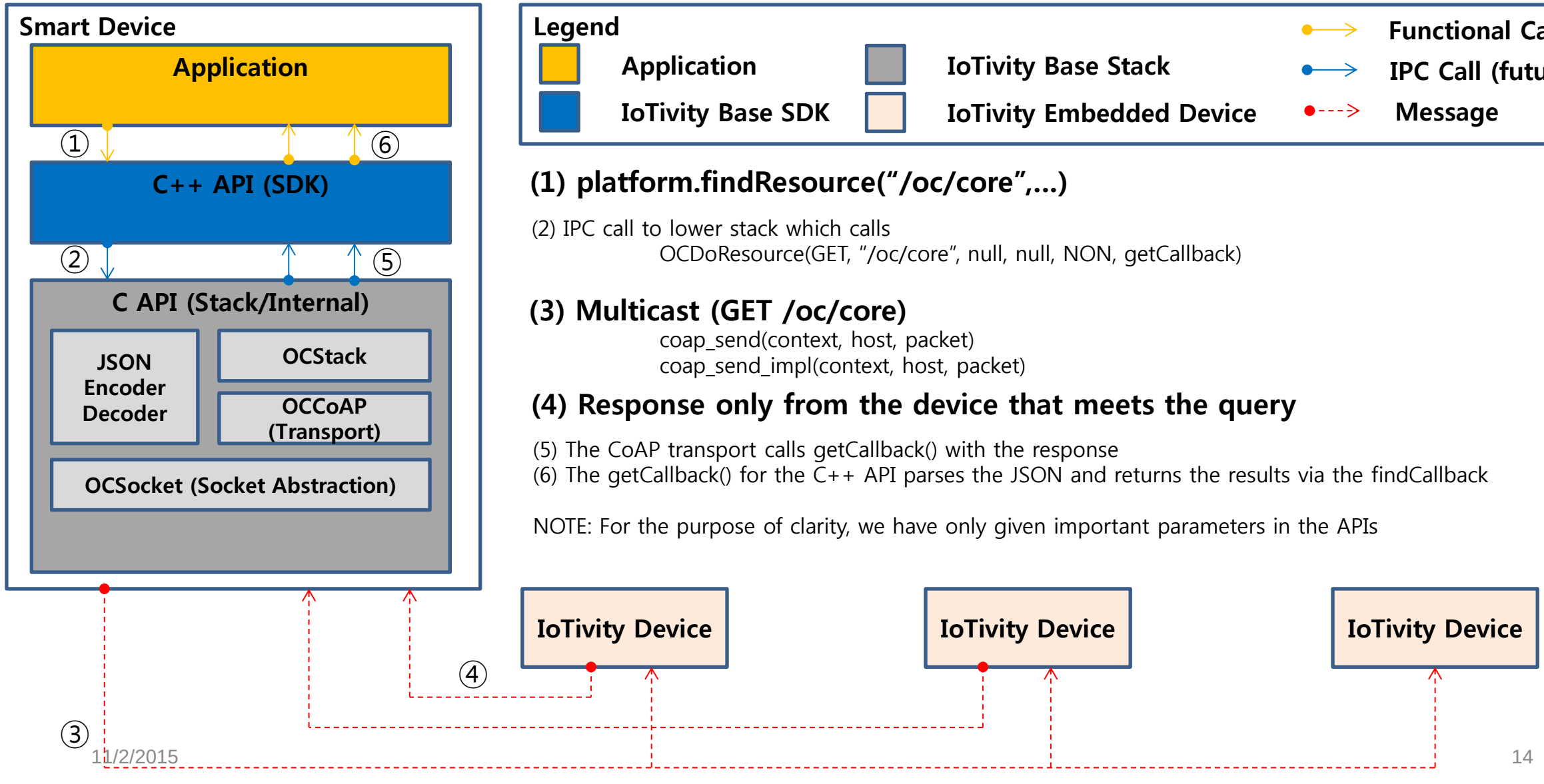
(2) IPC call to lower stack which calls
OCCreateResource(&handle, rt, if, "/light/1", hander, flags)
In OCStack, creates a OCResource record which is added to the resource linked list

NOTE:
- For the purpose of clarity, we have only given important parameters in the APIs
- It is recommended that resources be registered before starting the asynchronous processing (OCProcess) to prevent race conditions such as the entity handler being called before registerResource() returns. This is generally not a problem on bare metal platforms such as Arduino, but could be an issue on multi-threaded platforms.

(Notice) Should register ONE Resource per URL



Finding a Resource – 1/2



(1) platform.findResource("/oc/core",...)

(2) IPC call to lower stack which calls
OCDoResource(GET, "/oc/core", null, null, NON, getCallback)

(3) Multicast (GET /oc/core)

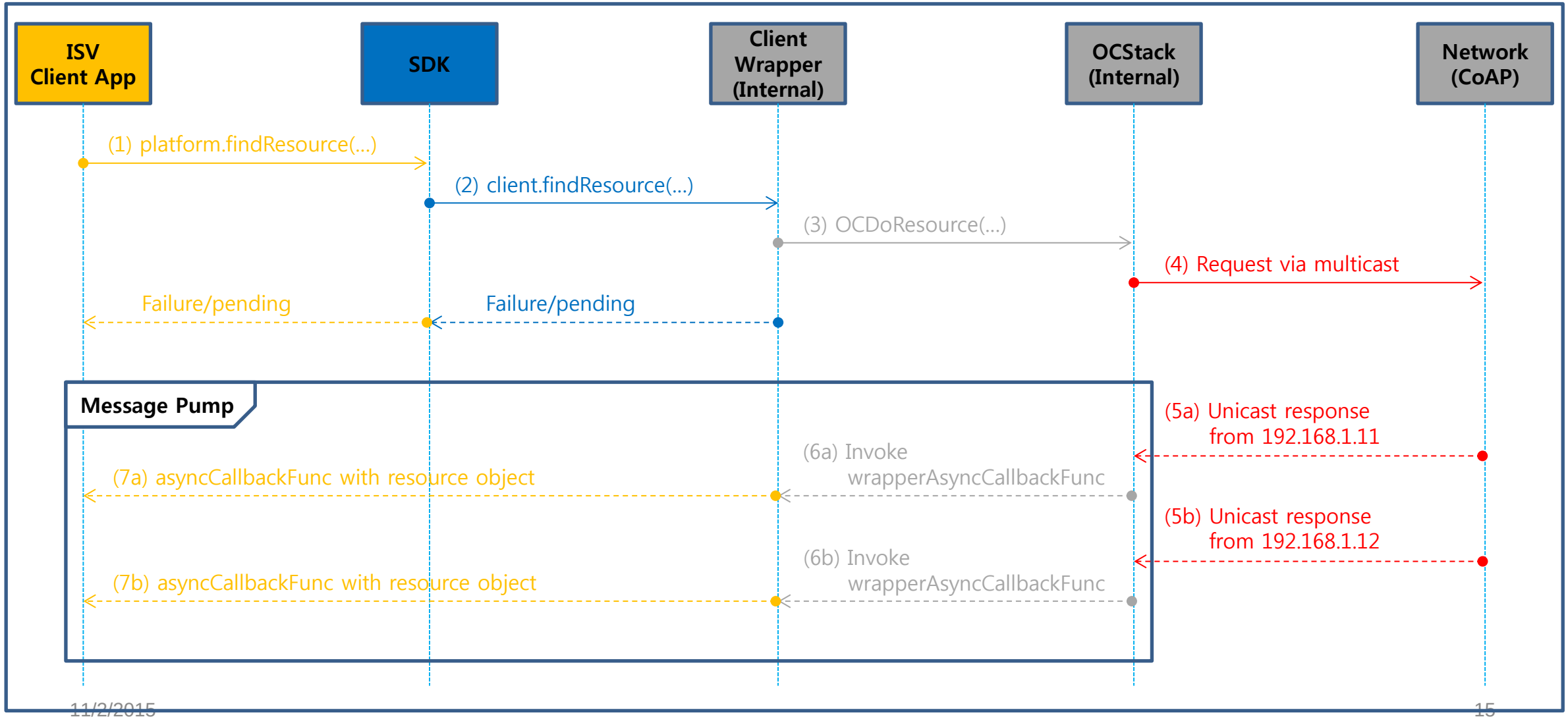
coap_send(context, host, packet)
coap_send_impl(context, host, packet)

(4) Response only from the device that meets the query

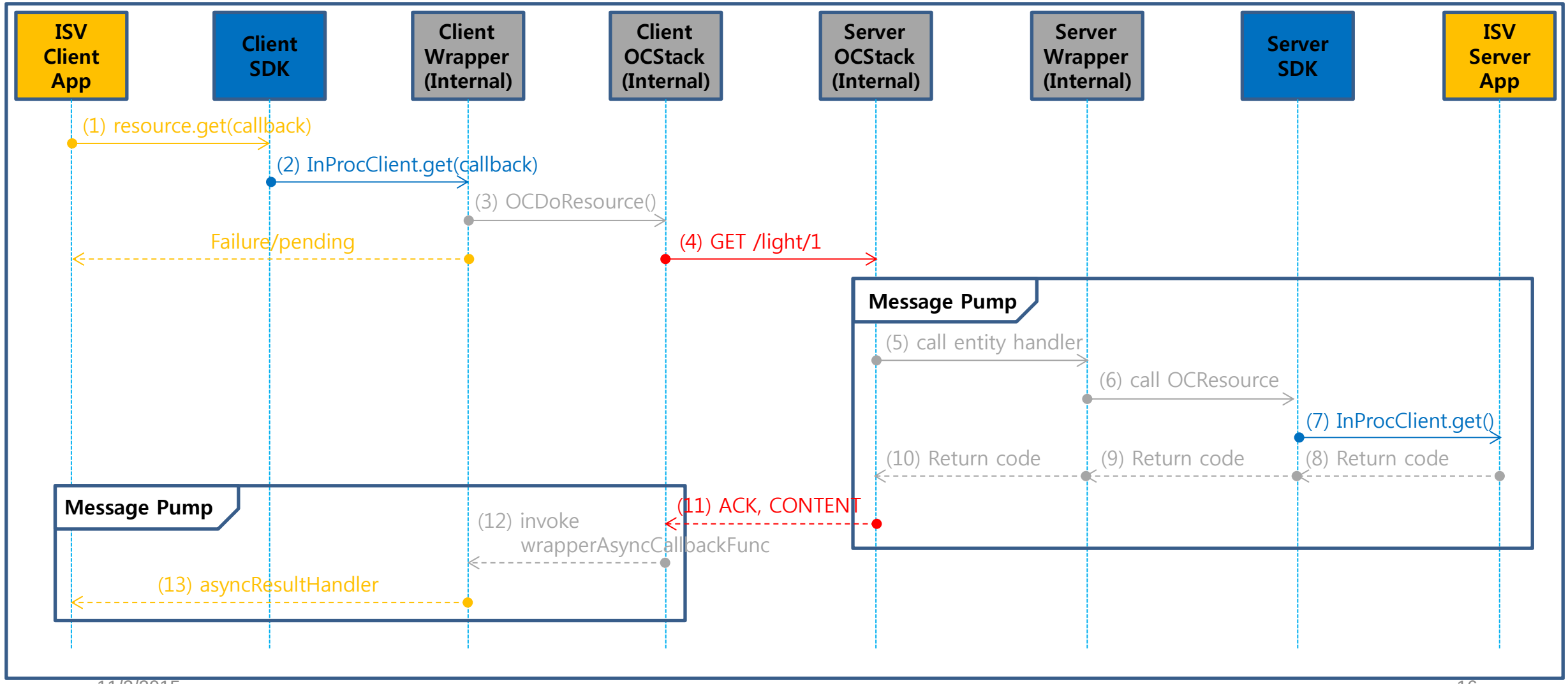
(5) The CoAP transport calls getCallback() with the response
(6) The getCallback() for the C++ API parses the JSON and returns the results via the findCallback

NOTE: For the purpose of clarity, we have only given important parameters in the APIs

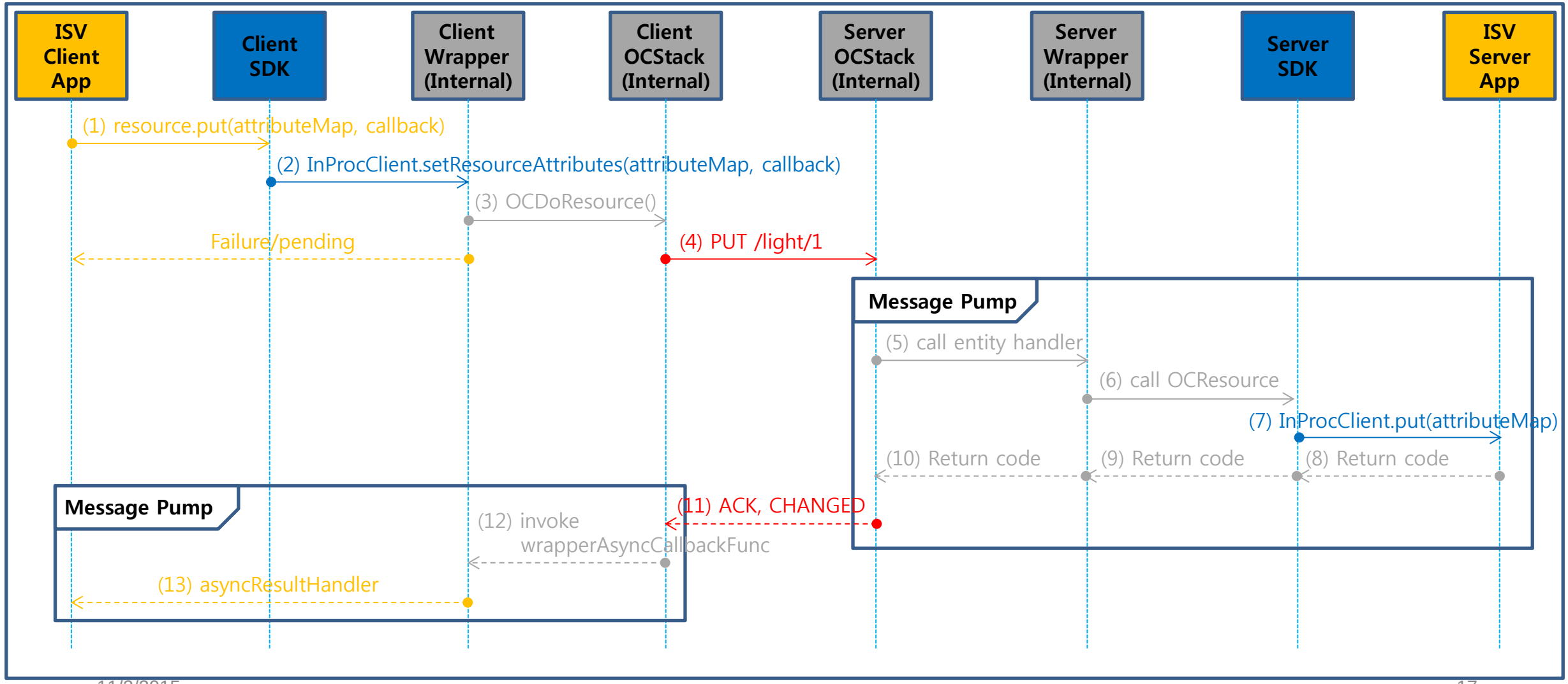
Finding a Resource – 2/2



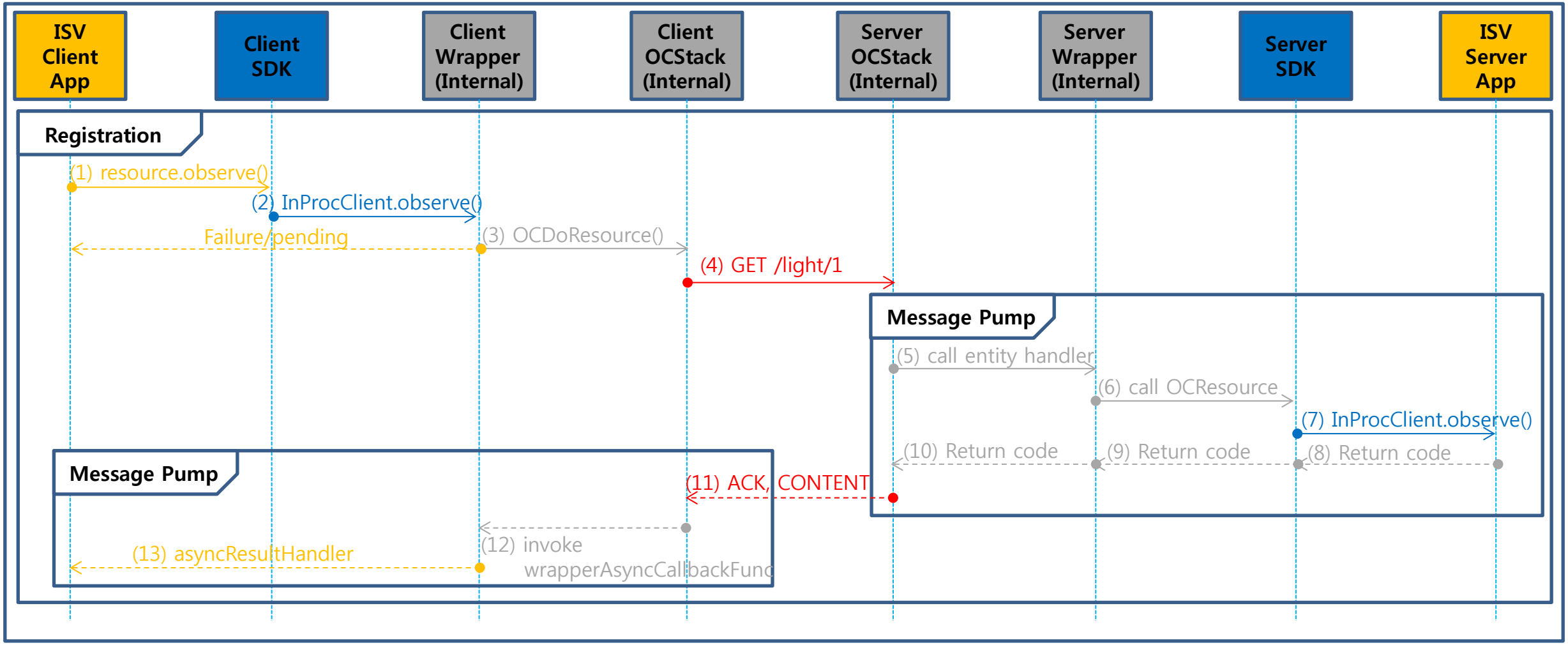
Querying Resource State [GET]



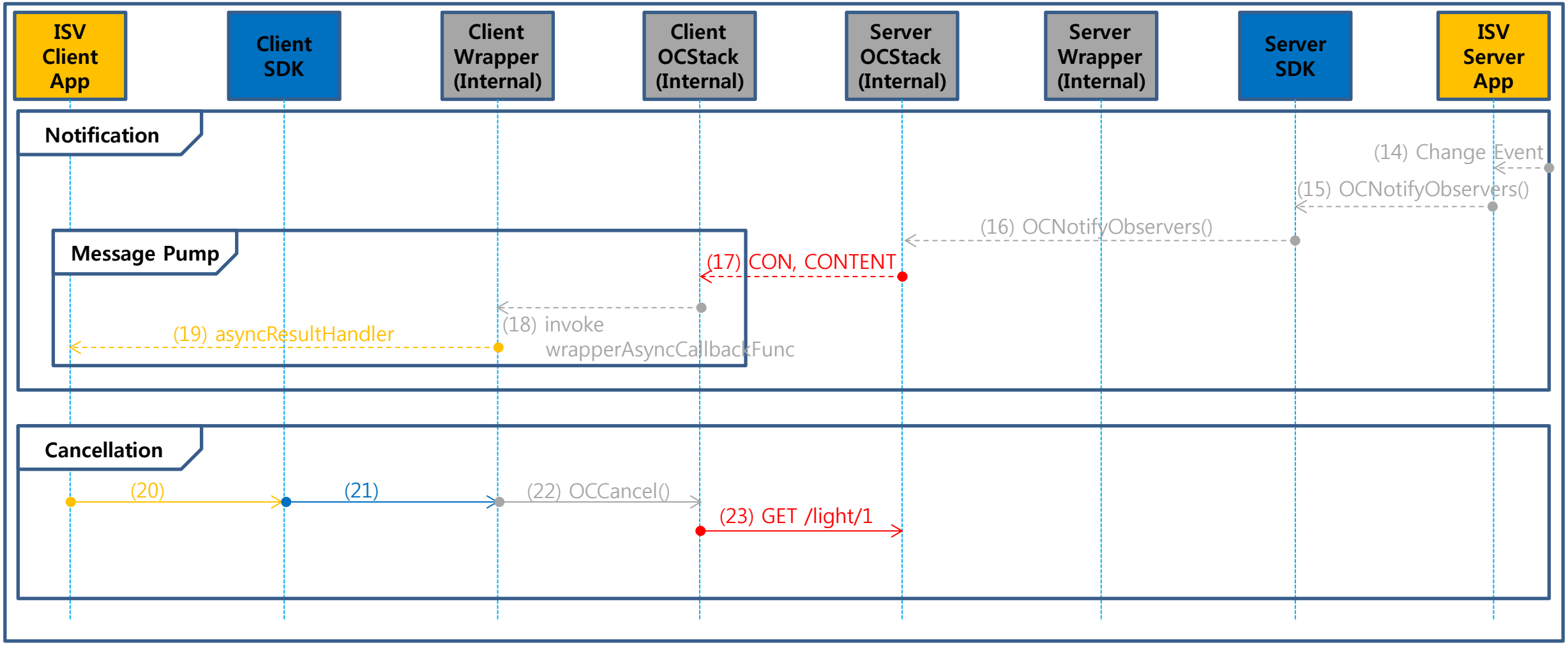
Setting a Resource State [SET]



Observing Resource State [Observe] – 1/2



Observing Resource State [Observe] – 2/2



IoTivity Service Features

Also provides developer-friendly **SERVICE APIs** that can be frequently utilized like:

- **Things Manager** for grouping
- **Soft Sensor Manager** for data aggregation
- **Protocol Plugin Manager** and **Web Service Interface** for extensibility

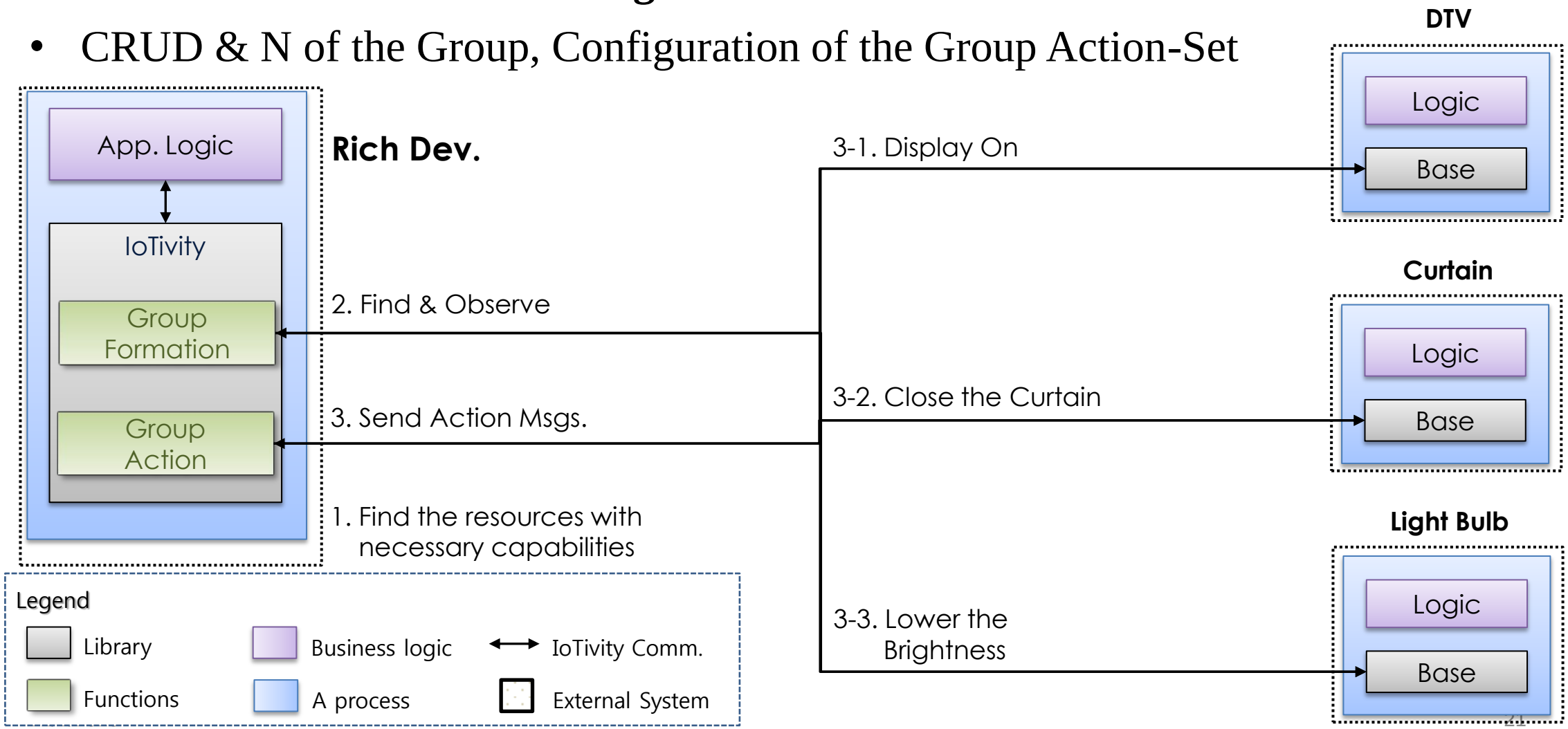
Corresponding IoTivity Project:

- **Primitive Service** Project
- **Web Service Interface** Project

Things Manager

Provides **Activation** and/or **Configuration** of the **GROUP of THINGS**:

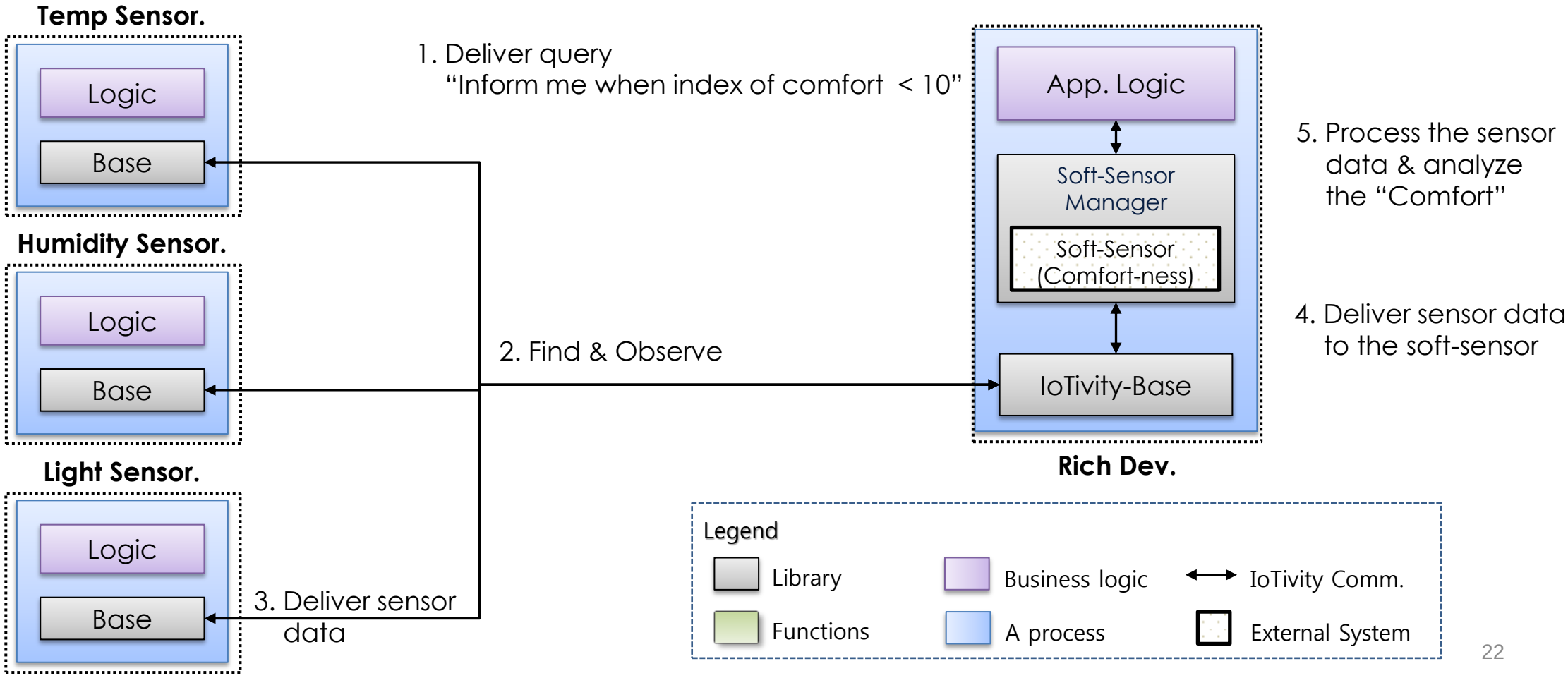
- CRUD & N of the Group, Configuration of the Group Action-Set



Soft Sensor Manager

Provides a service for sensors' **DATA RE-DEFINITION**:

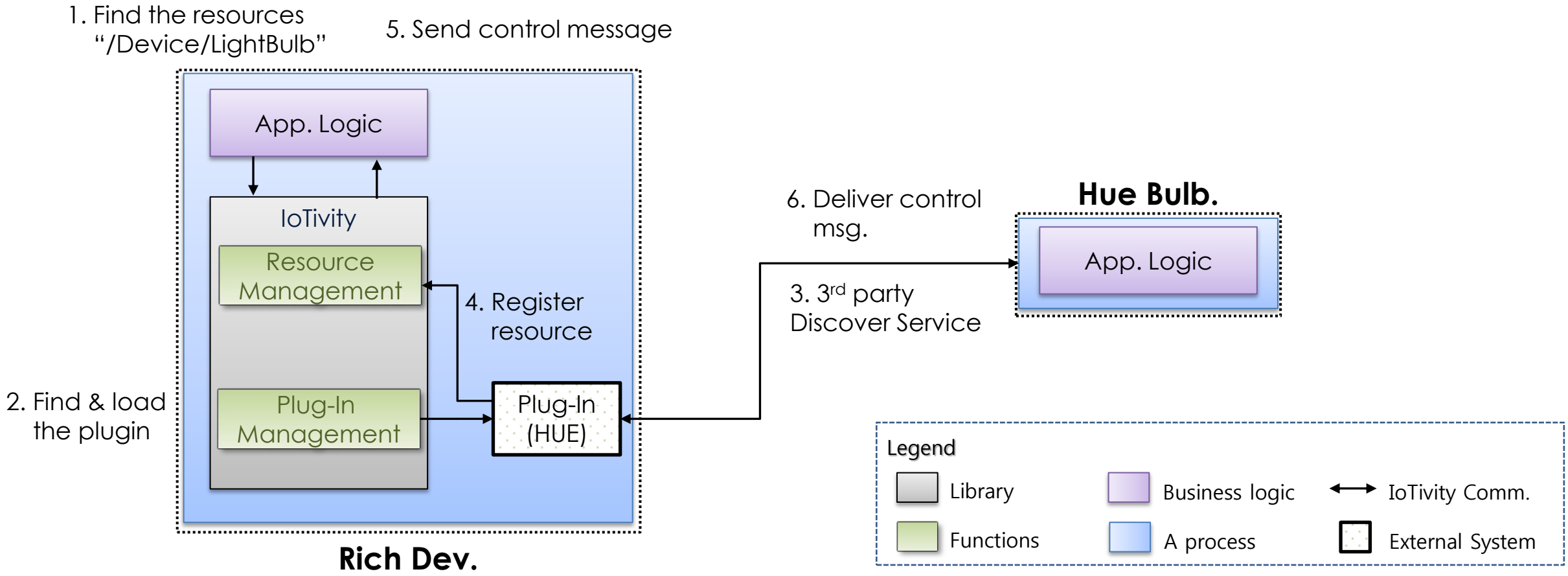
- (Example) "**Discomfort**" sensor – temperature + humidity sensors



Protocol Plugin Manager

Provides **EXTENSIBILITY** for **non IoTivity-based protocols**

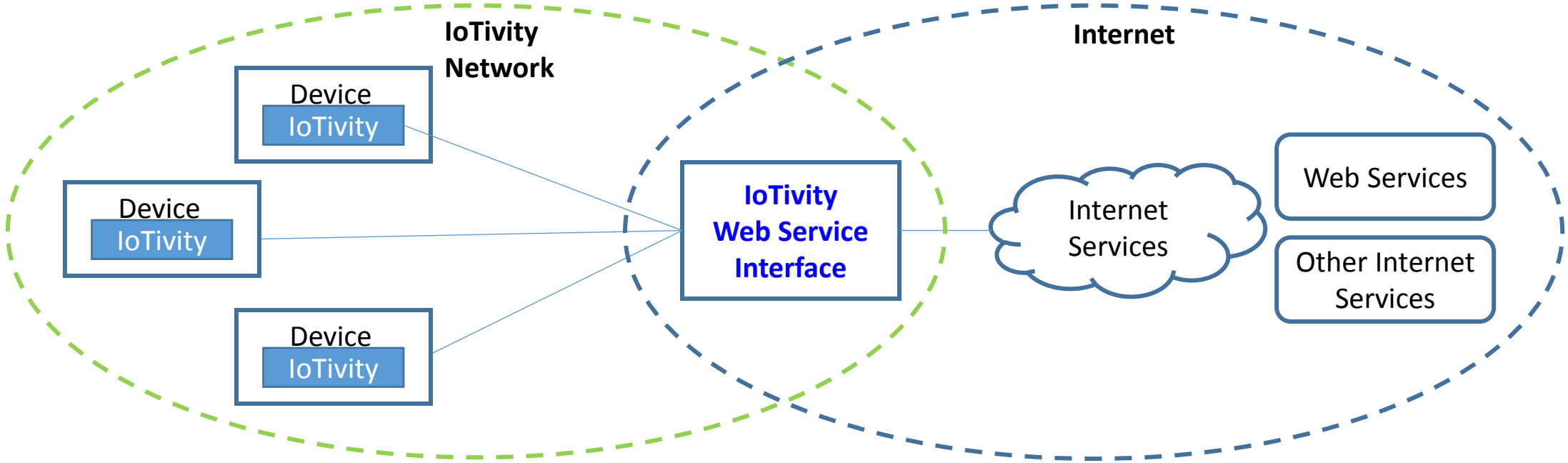
- Loads the correspondent **Protocol Plugin** as **.so** at the given run-time



Web Service Interface

Provides functionalities for enabling connection between IoTivity & Internet

- Mapping OIC resource representation to web services APIs with RESTful manner for CRUD&N methods
- Providing application model for aggregating Internet services and things



IoTivity v1.0 Has Been Just Released!

Features:

- Reference implementation of the full OIC Specification v1.0
- Multi-PHY easy setup (on-Boarding): Wi-Fi OnBoarding with SoftAP
- Remote Access based on XMPP
- CoAP over TCP on Linux
- Resource directory
- Simulator for Java API (Linux): for OIC resource (Server/Client)
- IPv6 and 6LoWPAN
- Bluetooth Serial RFCOMM for Android, BLE GATT for Linux, Android, Arduino
- Block-wise Transfer over IP/Bluetooth
- ZigBee and Z-Wave service profile plugin

How Can I Contribute to IoTivity?

[Reference] https://iotivity.org

- Discuss on mailing list to get general consensus about approach
- Pull latest code
- Build on Your Supported Build Platform
- Develop feature or fix bug following IoTivity Coding Standards; ask questions on mailing list as needed
- Submit to Gerrit for review
- Review and respond to reviewer comments
- Change accepted!

Thank you!

- Stay tuned on **iotivity.org** and **openinterconnect.org** -

moonki1.hong@samsung.com