Carnegie Mellon University

95-733 Internet of Things
REST, Integration Patterns, and CoAP

Where are we?

Internet Protocol Suite

HTTP, Websockets, DNS, XMPP, MQTT, CoAp	Application layer
TLS, SSL	Application Layer (Encryption)
TCP, UDP	Transport
IP(V4, V6), 6LowPAN	Internet Layer
Ethernet, 802.11 WiFi, 802.15.4	Link Layer

We are here!

REST

- The term *representational state transfer* (REST) was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.
- The term is intended to evoke an image of how a well-designed Web application behaves. Many web applications are not "RESTful".
- It is a network of Web resources where the user progresses through the application by selecting links, such as /user/tom, and operations such as GET or DELETE, resulting in the next resource (representing the next state of the application) being transferred to the user for their use. (Wikipedia)

REST API Design Principles

Principle	Implementation
Use a constrained user interface. The verbs are polymorphic.	HTTP GET, POST, DELETE, PUT
Use standard status codes. Don't make things up.	HTTP codes
Use URI's for nouns.	Naming. Identification of resources. Well designed URI's pointing to a resource - protocol and location.
Negotiate representations.	JSON or XML messages or Image or …
HATEOAS (Hypertext as the Engine of application state)	Messages returning pointers or links for further discovery.
Statelessness	Simple request/response required. No conversational state. Easy to scale. The request must contain all that is required for the reply to be computed.

A RESTful Request/Response

- GET /basement/water/temperature
  200 OK
  application/text
  40.5 F
- GET /basement/water/volume
  200 OK
  application/text
  200 G

- Still better to use a standard message format (representations) for return values.
- Spend more time designing return value format!
- Use JSON-LD - Javascript Object Notation for Linked Data

IoT Metadata

- Interoperability – How can I interact with this device?
- Make this data available to device users – on the device, on a gateway, or from a digital twin.
- Include metadata in messages to provide meaning – is this temperature in Celsius or Fahrenheit? Is this weight in ounces or grams.
- Include a Thing Description for every Thing. Place it on the device, on a gateway, or in the cloud.
- Follow guidelines from W3C Thing Description standard.

JSON-LD Thing Description

GET http://*mythingserver.com/things/pi*

Accept: application/json

200 OK

```
{
  "id": "https://mywebthingserver.com/things/pi",
  "title": "WoT Pi",
  "description": "A WoT-connected Raspberry Pi",
  "properties": {
        "temperature": {
            "title": "Temperature",
            "type": "number",
            "unit": "degree celsius",
            "readOnly": true,
            "description": "An ambient temperature sensor",
            "links": [{"href": "/things/pi/properties/temperature"}]
         },
```

JSON-LD Thing Description

```
"humidity": {
     "title": "Humidity",
     "type": "number",
     "unit": "percent",
     "readOnly": true,
     "links": [{"href": "/things/pi/properties/humidity"}]
   },
"led": {
     "title": "LED",
     "type": "boolean",
     "description": "A red LED",
     "links": [{"href": "/things/pi/properties/led"}]
   }
 },
```

JSON-LD Thing Description

```
"actions": {
    "reboot": {
      "title": "Reboot",
      "description": "Reboot the device"
    }
  },
  "events": {
    "reboot": {
      "description": "Going down for reboot"
    }
  },
```

JSON-LD Thing Description

```
"links": [
    {
     "rel": "properties",
     "href": "/things/pi/properties"
    },
    {
     "rel": "actions",
     "href": "/things/pi/actions"
    },
    {
     "rel": "events",
     "href": "/things/pi/events"
    },
   {
      "rel": "alternate",
      "href": "wss://mywebthingserver.com/things/pi"
     },
                                        {
                                            "rel": "alternate",
                                            "mediaType": "text/html",
                                            "href": "/things/pi"
                                        }
                                    ]
                                    }
```

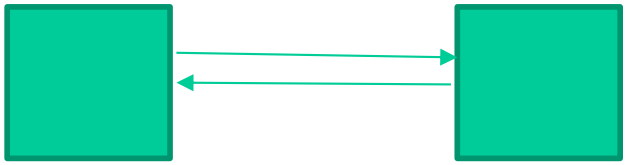
WoT Integration Patterns

- Direct integration Pattern
- Gateway Integration Pattern
- Cloud Integration Pattern

Notes on Integration Patterns from "Building the Web of Things" by Guinard and Trifa

Direct Integration Pattern

- Some Things have full internet access. These Things may provide an HTTP server running over TCP/IP and can directly connect to the internet – using, say, WiFi or Ethernet or cellular. Rasberry-Pis and Photon 2s are examples. These may be used to implement a Direct Integration Pattern – REST on devices.



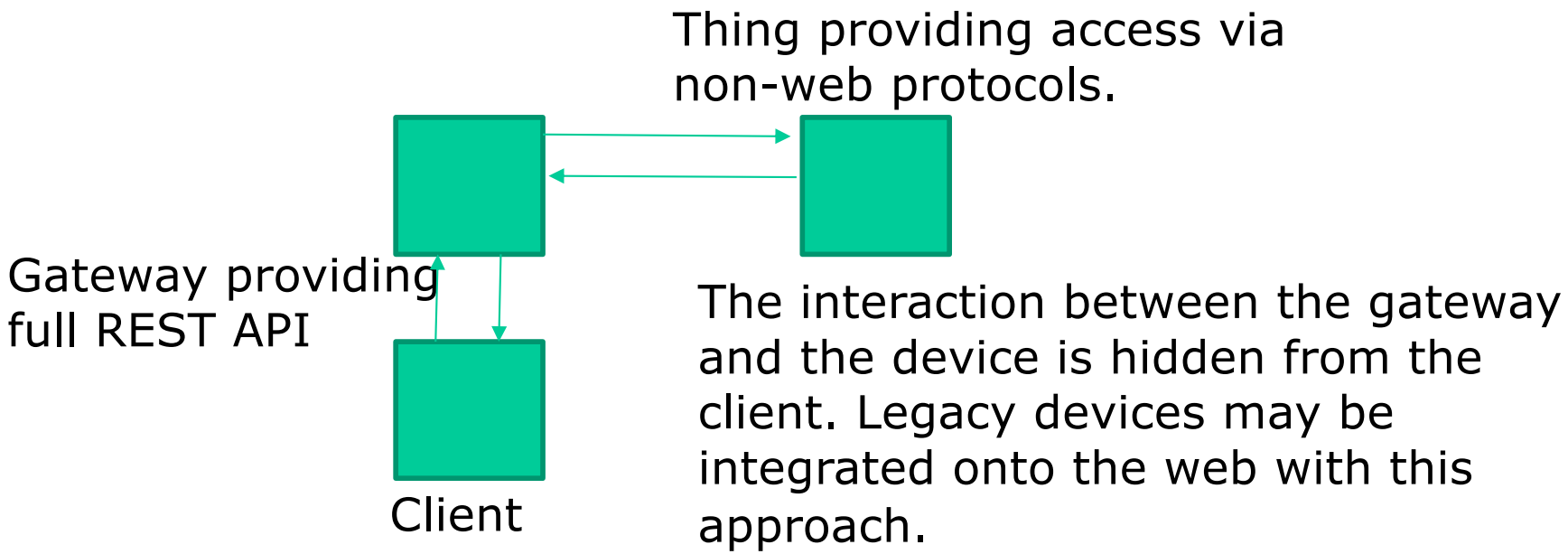
Client

Thing running an HTPP REST server

- Typical use case: The Thing is not battery powered and communicates with low latency to a remote or local device like a phone or browser.
- Example: Use a phone to communicate to an HTTP server on a device.
- Use web sockets for publish/subscribe, e.g., phone listens for doorbell events.

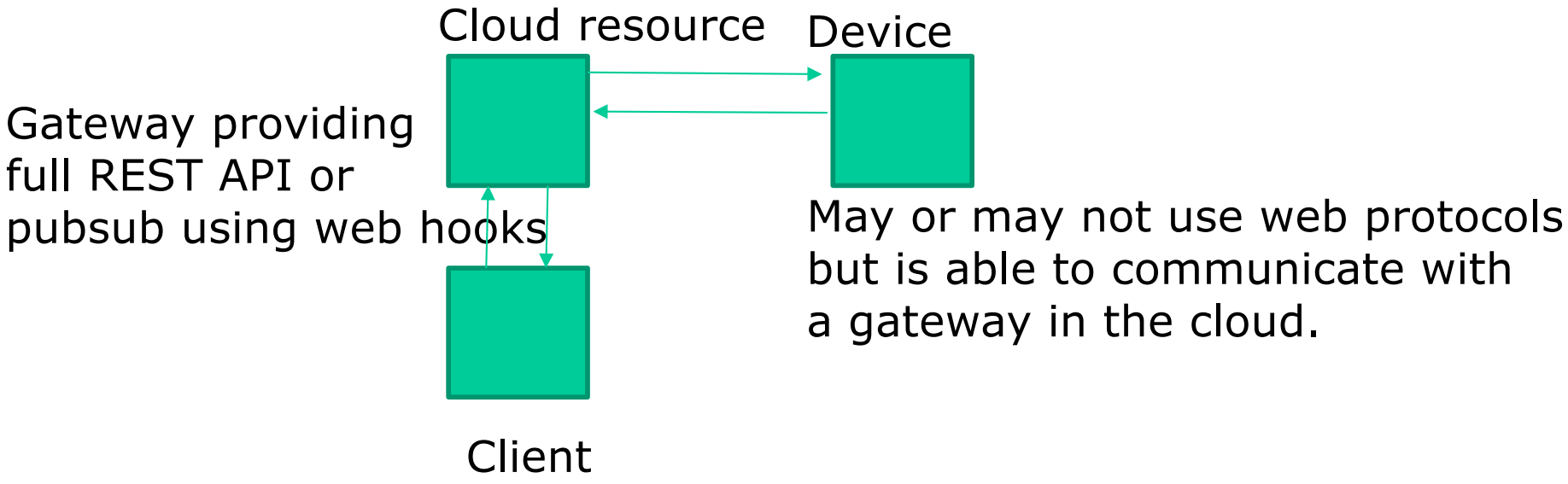
Gateway Integration Pattern

- Some Things may not have full internet access. These Things may support only Zigbee or Bluetooth. Suppose we cannot send IP packets to the device – it is constrained. This is the Gateway Integration Pattern.

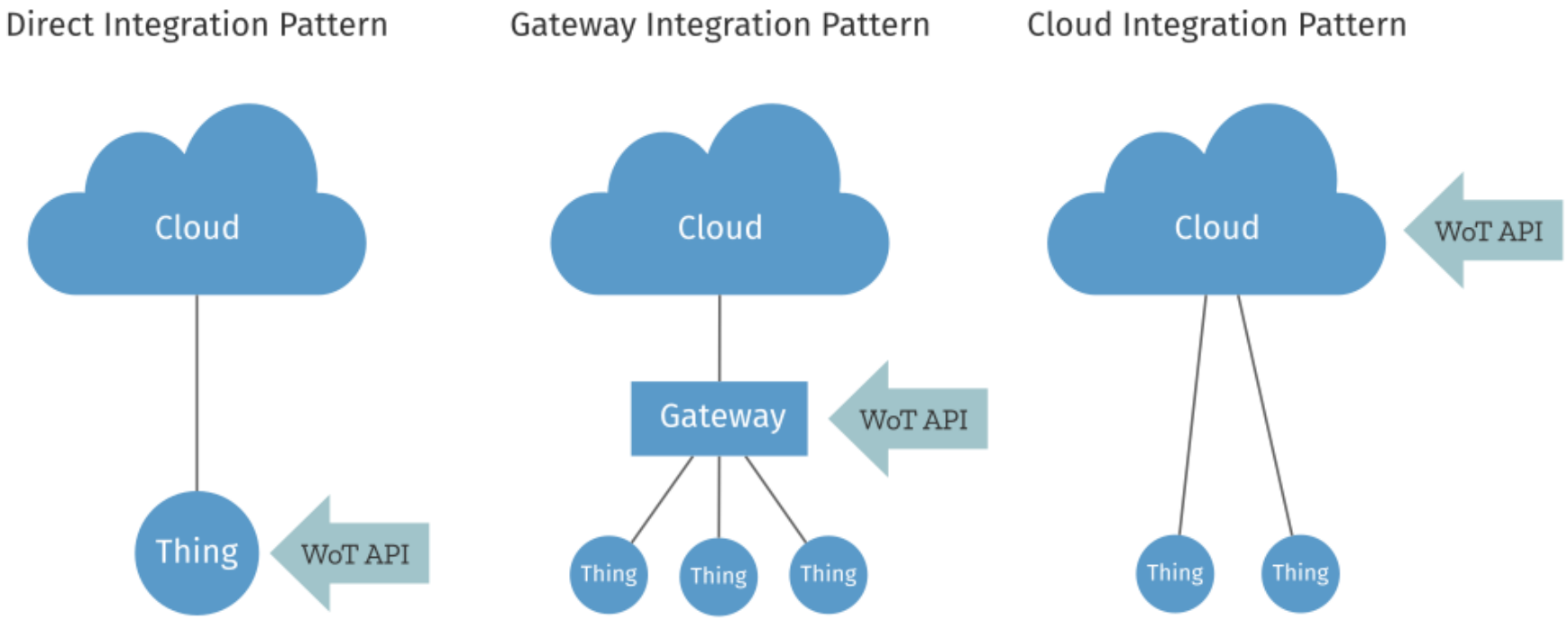


Cloud Integration Pattern

- Some Things have access to the cloud and need powerful and scalable cloud support. This is the Cloud Integration Pattern. The Particle Photon 2, for example, can send event notifications to the Particle cloud. The particle cloud provides publish/subscribe using web hooks.



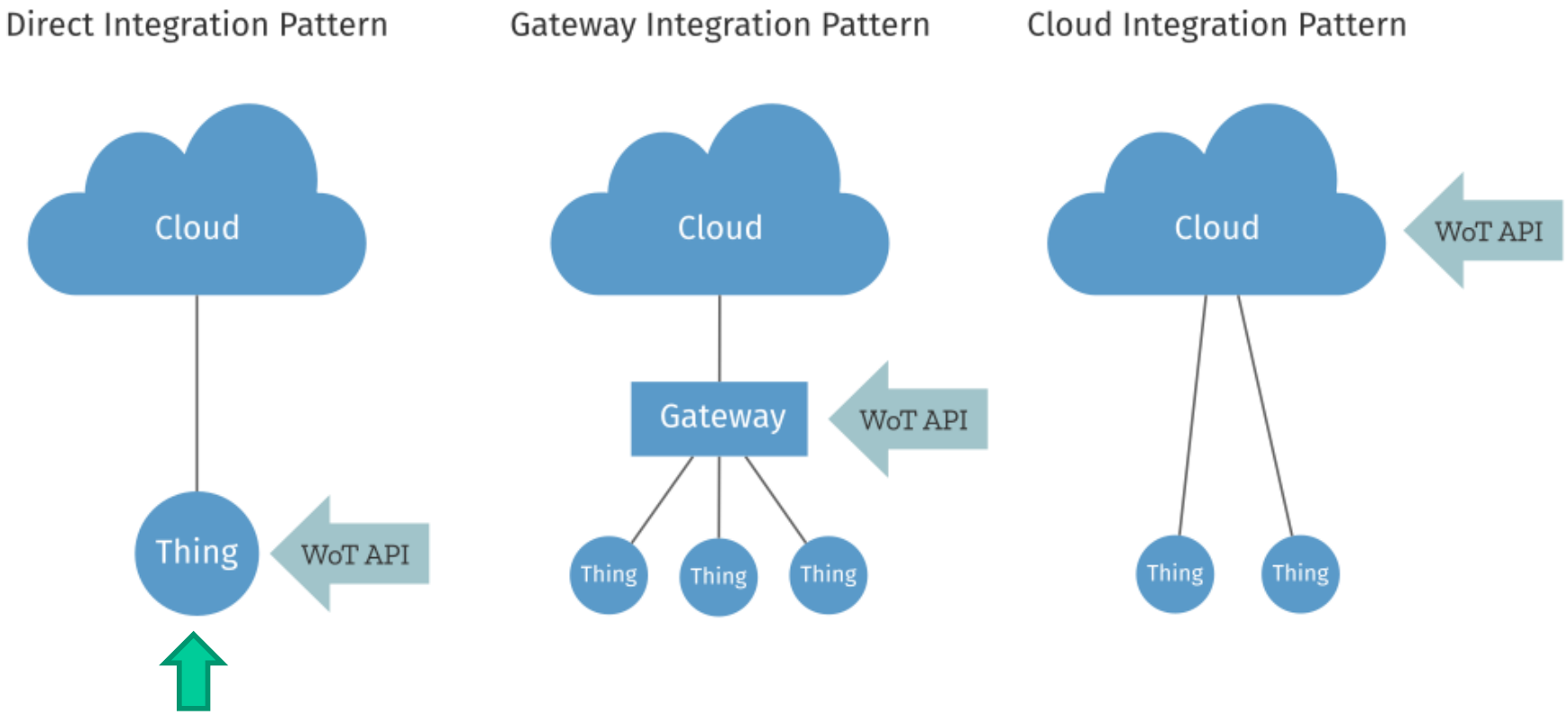
Web of Things Integration Patterns



Patterns from “Building the Web of Things” by Guinard and Trifa

95-733 Internet of Things

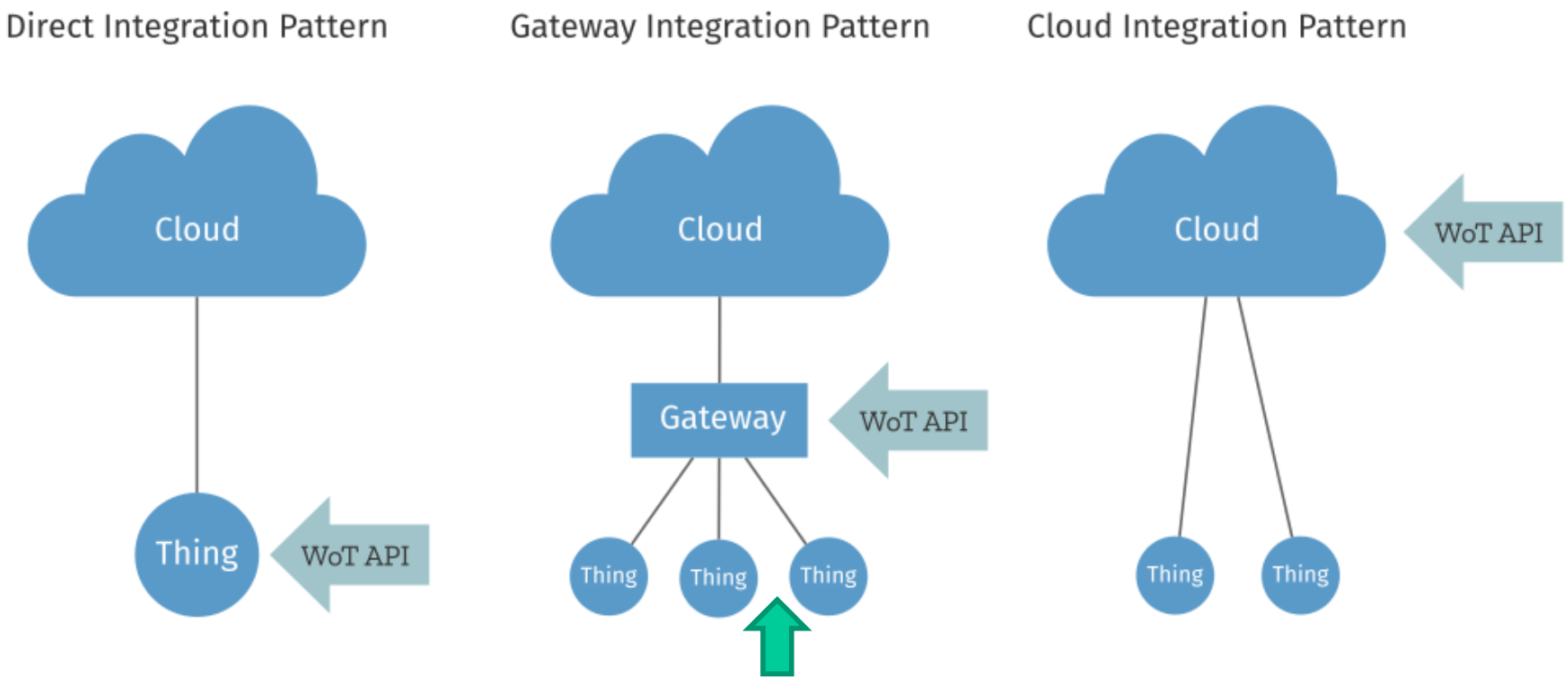
Web of Things Integration Patterns



Use if Thing has enough power to support an HTTP server, e.g., a WiFi camera. Security is of high concern. From Mozilla

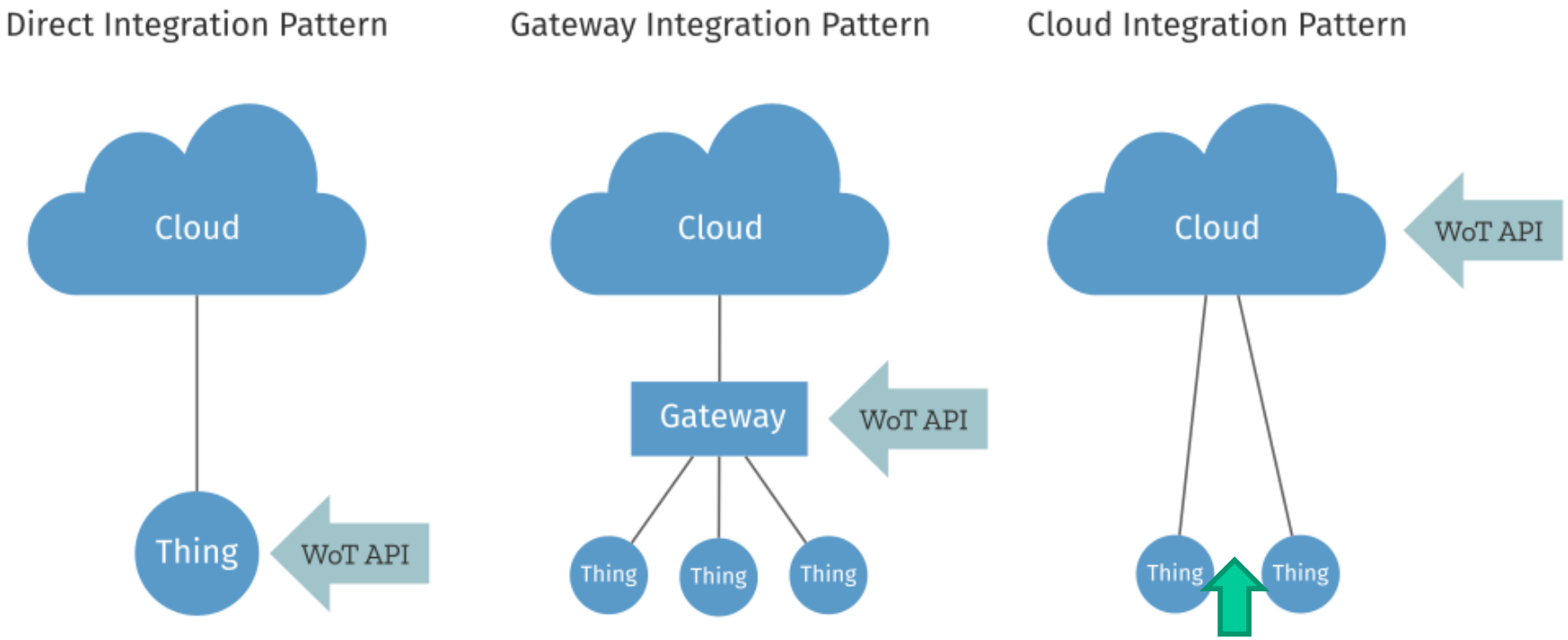
95-733 Internet of Things

Web of Things Integration Patterns



This pattern is particularly useful for devices which have limited power or which use PAN network technologies like Bluetooth or ZigBee that don't directly connect to the Internet (e.g. a battery powered door sensor). From Mozilla

Web of Things Integration Patterns



Suppose the device uses some other protocol to communicate with the server on the back end. This pattern is particularly useful for a large number of devices over a wide geographic area which need to be centrally co-ordinated (e.g. air pollution sensors). LoRaWAN is here. From Mozilla

Constrained Application Protocol

Where are we?

Internet Protocol Suite

HTTP, Websockets, DNS, XMPP, MQTT, CoAp	Application layer
TLS, SSL	Application Layer (Encryption)
TCP, UDP	Transport
IP(V4, V6), 6LowPAN	Internet Layer
Ethernet, 802.11 WiFi, 802.15.4	Link Layer

Constrained Application Protocol (CoAp)

- A key IoT standard. Supported in Java, C, Python, C#, Go, etc.
- Open IETF standard since June 2014.
- Based on web standards, easily integrates with HTPP. Is not simply a compressed version of HTTP.
- Built for small, constrained, imbedded, occasionally sleeping devices. Why sleep?
- Some built-in reliabilty over UDP/IP.
- May also run over UDP/6LoWPan.
- Use on low power, low bandwidth, lossy networks.
- Is not HTTP but is clearly based on REST.

Constrained Application Protocol (CoAp)

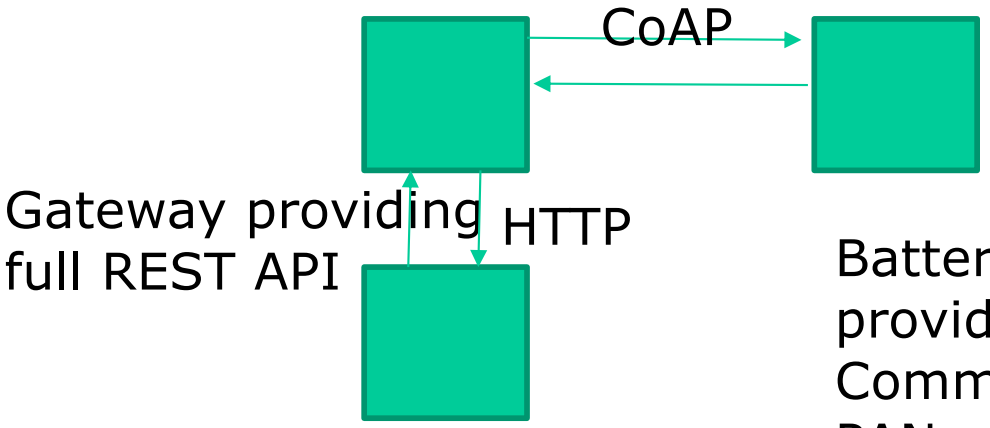
- Over UDP or SMS on cellular networks
- DTLS for security
- Asynchronous subscriptions and notifications over UDP
- Built-in resource discovery
- Peer to peer or client server and multi-cast requests

Recall the typical HTTP Interaction

- Connection oriented and synchronous (blocking)
- TCP 3 way handshake with server
- HTTP GET /kitchen/light
- HTTP response with headers and {“setting” : “dim” }
- TCP 2 way termination
- Too much work for simple IoT applications
- CoAP is not a general replacement for HTTP
- CoAP does not support all features of HTTP

CoAp is based on REST

CoAP provides a request/response RESTful interaction like HTTP.
Smaller messages than HTTP and with very low overhead.
BLE nodes, for example, have limited memory and storage.
Sensors and actuators on BLE nodes are simply CoAP REST resources.
For example, to obtain a current temperature, send a GET request.
To turn on/off or toggle LEDs we use PUT requests.



Battery powered device
providing CoAp.
Communication uses UDP over a
PAN protocol, e.g., 6LoWPAN over
IEEE 802.15.4 or Bluetooth Low E

CoAp

- Has a scheme coap://
- Has a well known port.
- GET, POST, PUT, DELETE encoded in binary (1 == GET)
- Block transfer support.
- Confirmable messages requires an ACK with message ID. The message ID of the ACK matches the message ID of the confirmable message.
- Non-confirmable messages do not require an ACK. Less reliable.
- Responses are matched with requests via the client generated Token.
- Example:

CoAP Client CoAP Server

----> CON {id} GET /basement/light

Confirmable request has an ID

<---- ACK {id} 200 Content {“status” : “on”}

Piggy back response and same ID

CoAP Uses Timeouts over UDP

CoAP Client CoAP Server

---> CON {id} GET /basement/light lost request

timeout

---> CON {id} GET /basement/light finally arrives

<--- ACK {id} 200 Content {“status” : “on”}

The {id} allows us to detect duplicates.

What happens if the ACK is also lost?

CoAP Request/Acknowledge/Callback

CoAP Client CoAP Server

---> CON {id} PUT /basement/cleanFloor Token: 0x22

Needs time

<---- ACK {id} I am on it!

<----- CON {newID} 200 Content /basement/cleanFloor Token: 0x22 Done

----> ACK {newID}

In this example, the same token is used to identify this request and the service response. The id's are used at the message level.

CoAP Publish/Subscribe

The GET includes an “Observe” message to establish a subscription request.
The response includes an “Observe” to say this is a publication.
The value included with Observe response is there for possible re-orderings.
The client should take the most recent sent and not the most recent to arrive.

CoAP Client CoAP Server

```
---->   CON  {id} GET  /basement/light Observe: 0   Token:  0x22
<----   ACK  200 {id}  Observe: 27 Token 0x22
<----   CON 200 Observe: 28 Token: 0x22 {“light” : ”off”}
----->  ACK Token: 0x22
<----   CON 200 Observe: 30 Token: 0x22 {“light” : ”on”}
        :
        :
        etc.
```

Block transfer is similar. We may request a transfer (one block at a time).

CoAP Resource Discovery

- Not the same as service discovery. Service discovery is at a lower level. At low levels, we don't even know if services are available or how they communicate.
- We might register a printer, for example, with a discovery service and find it later on the fly.
- With resource discovery, we know we are looking for web resources.
- Links are returned. HATEOAS.
- Links may include a rel attribute – providing semantics.
- A well known resource is used to discover other resources.
- Perform a GET on the well known resource. Returned content is a list

  of links with REL attributes.
- Resource directories may be used to register resources. Registrations are simply POSTs with links. PUTs are used for updates. GETs for discovery.

CoAP Resource Discovery

CoAp Client CoAp Server

----> CON {id} GET /.well-known/core Token: 0x22
<---- ACK 200 {id} Content "/sensor/temp /sensor/light"
----> CON {id} GET /sensor/light
<---- ACK 205 {id} Content "dim"
----> CON {id} GET /sensor/temp
<---- ACK 205 {id} Content "72"

Notes on CoAP from the CoAp tutorial at
https://www.youtube.com/watch?v=4bSr5x5gKvA

Building the WoT- How to guide

0) A Smart Thing is a digitally enhanced object. A Connected Thing includes a network.

1) Identify resources with Uniform Resource Identifiers. Actuators, sensors, tabletops, rooms, Smart Things, etc. – all get a URI. These are the nouns.

2) Use a constrained interface and exploit polymorphism. Use HTTP or CoAP. PUT, GET, etc. are the verbs.

3) Agree upon resource representation formats. Use JSON or JSON-LD (JSON with semantics). On the web, the wide adoption of HTML allows clients and servers to cooperate without individual agreements.

4) Use well known addresses and links for discovery – perhaps exploiting JSON-LD.

5) Provide stateless interactions. Each request should contain all that the server needs to satisfy the request.

From "A Resource Oriented Architecture for the Web of Things", Guinard, Trifa, and Wilde