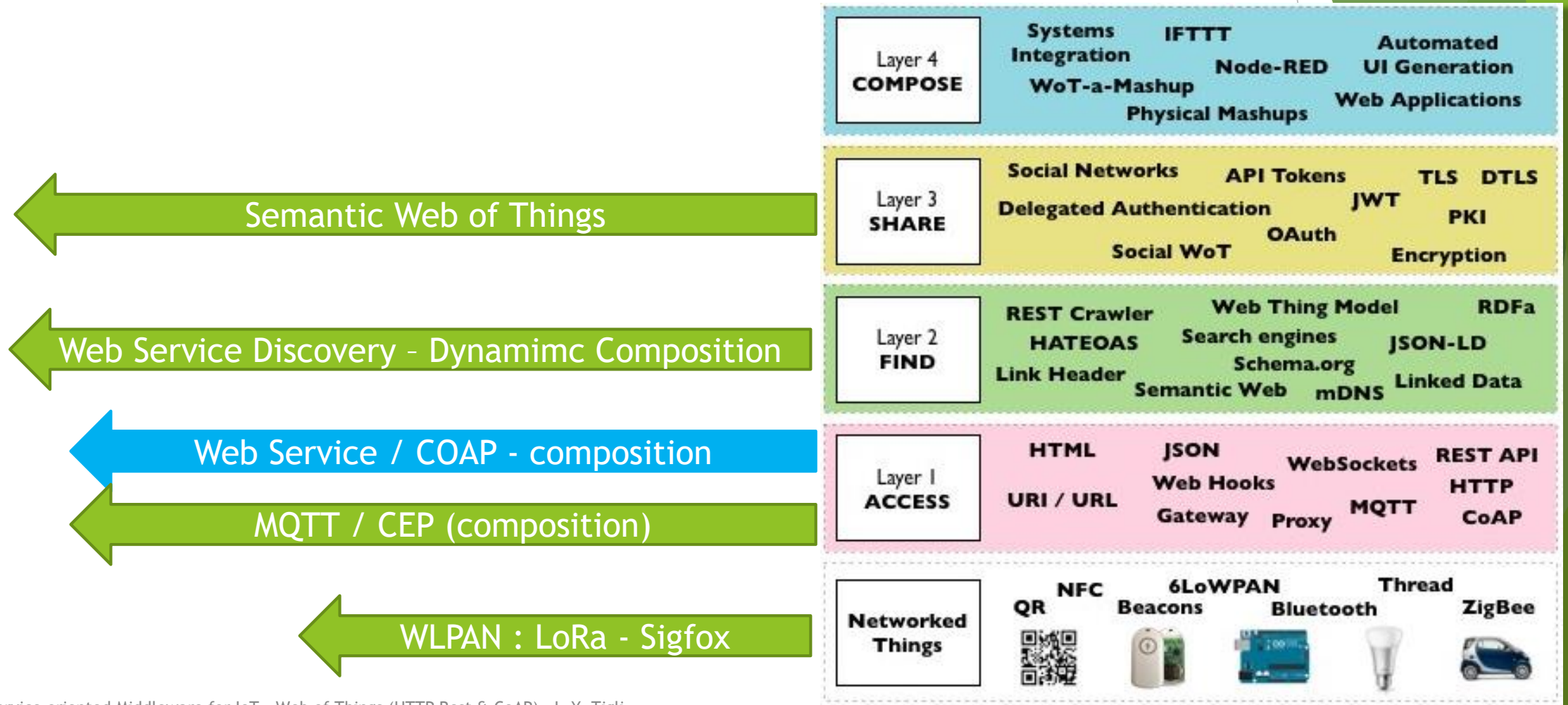


Service Oriented Middleware for IoT

SOM, based on ROA or SOA Approaches

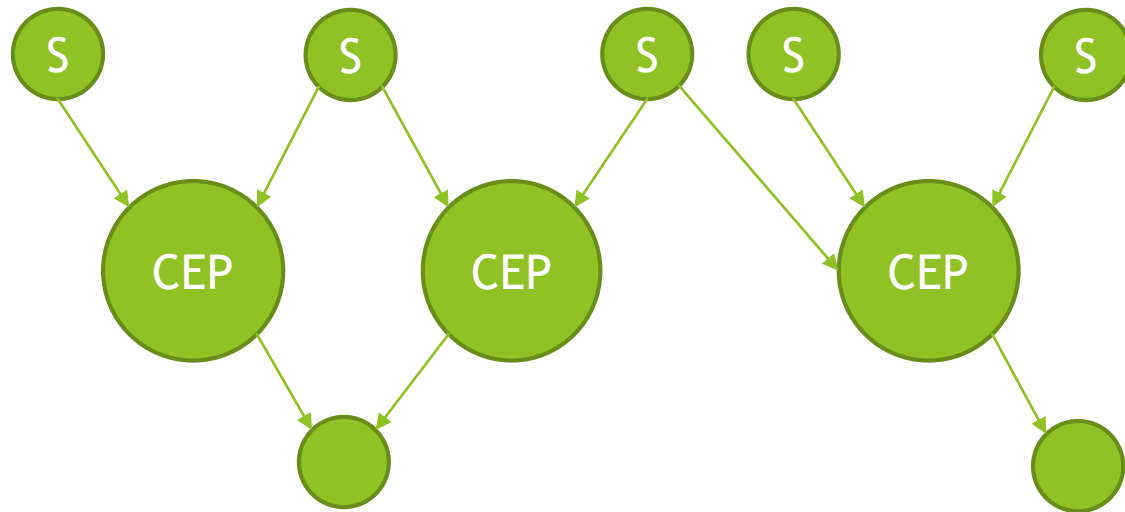
Reference : Service-oriented middleware: A survey Jameela Al-Jaroodi, Nader Mohamed, Journal of Network and Computer Applications, Volume 35, Issue 1, January 2012, Pages 211-220, Collaborative Computing and Applications

MIT Curriculum



MQTT for simples Objects

- ▶ Mostly for Sensors Networks ...



CEP : Complex
Events
Processing

- ▶ Model of composition : message passing and event driven
- ▶ Ex. of Composition : CEP using synchronous language

When Objects are more sophisticated devices than sensors ...

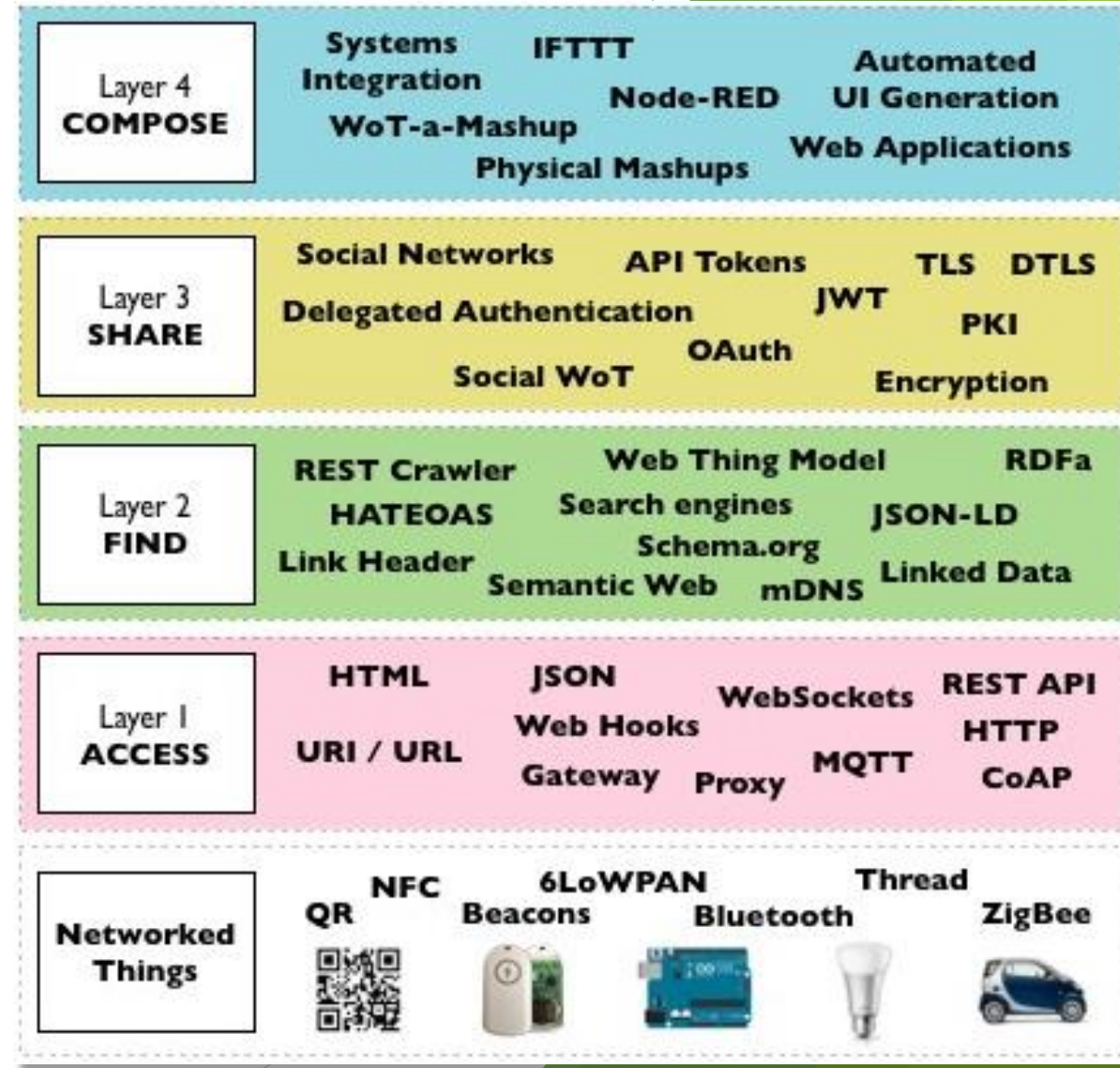
- ▶ What happen when Objects are more sophisticated ?



- ▶ Objects need Address (mostly IP address)
- ▶ Objects need more sophisticated communication protocols (Data/Ressource based or procedure based)
- ▶ Model of composition : Service oriented Programming (Mostly Web services oriented)

Web Service for an “Access” layer for IoT

- ▶ Provides a way to **access** services (devices) through the Web
 - ▶ Using Web standard protocols
 - ▶ Using dedicated protocols specific to IoT
 - ▶ etc...
- ▶ Different kind of **architectures**



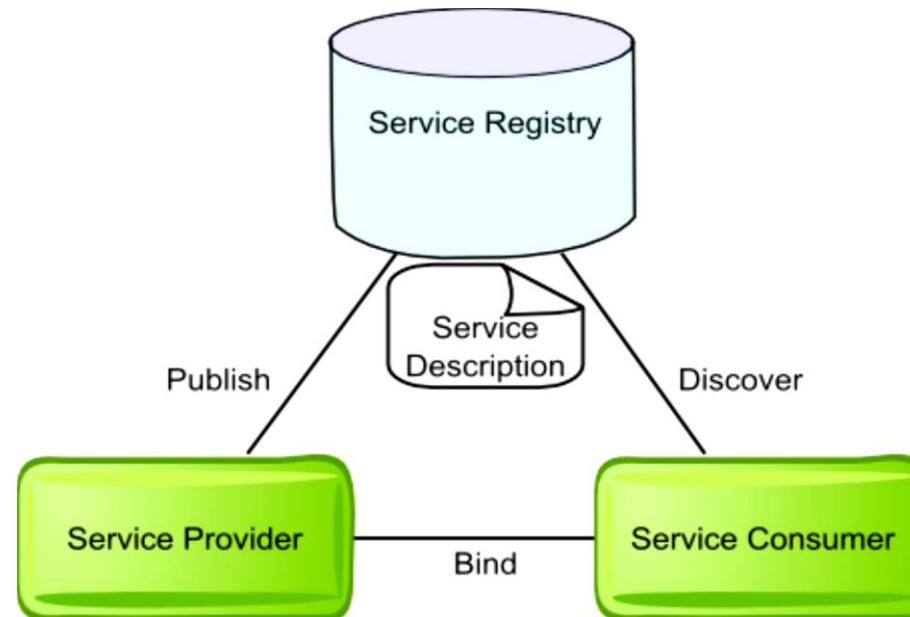
Middleware and Service Oriented Concepts

- ▶ Service-Oriented Middleware* is a kind of middleware based on the Service Oriented Architecture (SOA) paradigm that supports the development of **distributed software systems in terms of loosely coupled networked services**.
- ▶ In SOA, networked resources are made available as autonomous software **services that can be accessed without knowledge of their underlying technologies**.
- ▶ Key feature of SOA is that services are independent entities, with well-defined interfaces, which **can be invoked in a standard way**, without requiring the client to have knowledge about how the service actually performs its tasks.

(*) A Perspective on the Future of Middleware-based Software Engineering, Valérie Issarny, Mauro Caporuscio, Nikolaos Georgantas, Workshop on the Future of Software Engineering : FOSE 2007, 2007, Minneapolis, United States. pp.244-258, 2007, <https://hal.inria.fr/inria-00415919>

Middleware and Service Oriented Concepts

- ▶ The SOA style is structured around three key architectural components: (i) service provider, (ii) service consumer, and (iii) service registry
- ▶ In SOA-based environments, the Service-Oriented Middleware (SOM) is in charge of enabling the **deployment of services** and **coordination among the three key conceptual elements** that characterize the SOA style.
- ▶ **Popularity** of service oriented computing is mainly due to its **Web Service** instantiation.



Trends Web of Things or Web Service for Device

- ▶ Two kinds of approaches based on Service Oriented Architectures :
 - ▶ **ROA (DAO) : Resource or data oriented**
 - ▶ Communication pattern between service consumer and provider is based on **shared URL**
 - ▶ Example : Resources as URL like hyperlinks in a classical Web approach
 - ▶ **SOA : Service oriented (procedure based)**
 - ▶ **Communication pattern** between service consumer and provider is **RPC**
 - ▶ Example : RPC using **SOAP protocol** over HTTP

Resource Oriented Architecture

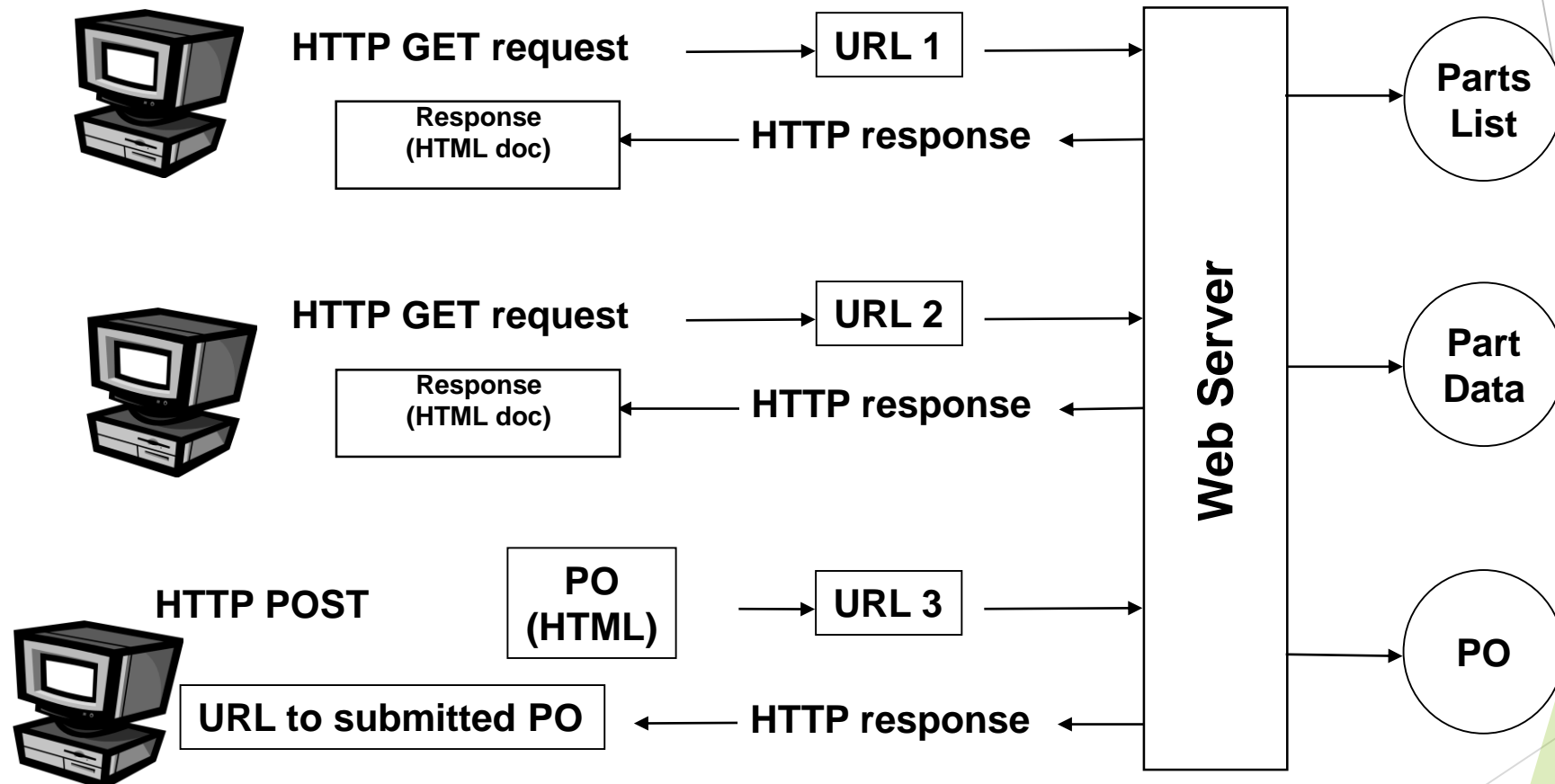
RESTful Web Services

- ▶ REpresentational State Transfer (**REST**)
 - ▶ Architecture inherent in all web based system since 1994, not explicitly described as an architecture until later
 - ▶ **An architecture - not a set of standard**
 - ▶ Web Services is both an architecture and a set of standards
- ▶ Goal: To leverage web based standards to allow inter-application communication as simply as possible
 - ▶ Matches the 'standard' **web interaction model**
 - ▶ **Resources as URL like hyperlinks in a classical Web approach**

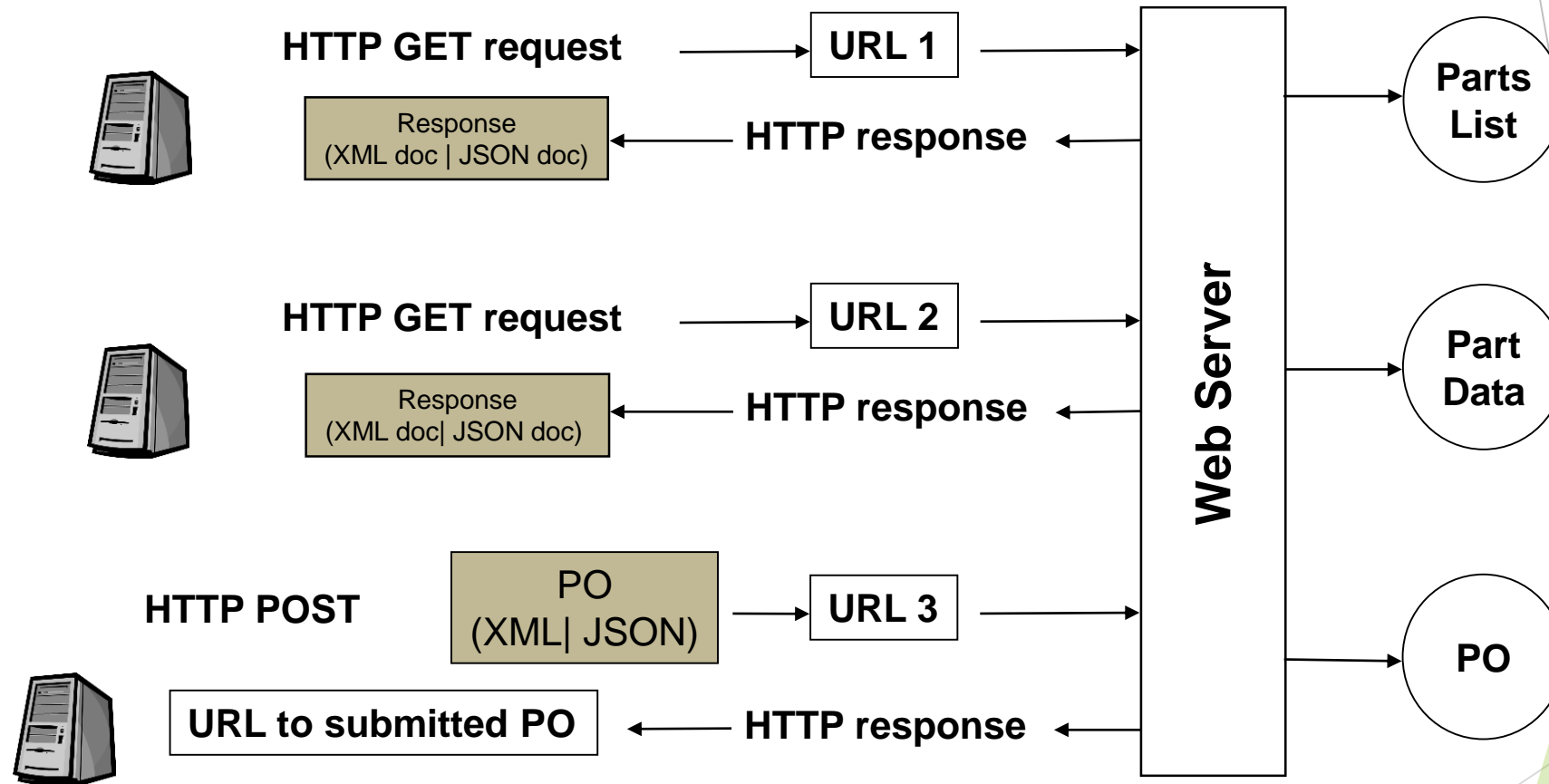
REST architecture

- ▶ Uses **HTTP operations (CRUD equivalence)**, the four basic functions of persistent storage):
 - ▶ POST = "here's some new info" (Create)
 - ▶ GET = "give me some info" (Read/Retrieve)
 - ▶ PUT = "here's some update info" (Update/Modify)
 - ▶ DELETE = "delete some info" (Delete/Destroy)
- ▶ Typically exchanges XML documents but not only ...
 - ▶ But supports a **wide range of other internet media types (JSON, XML, CSV ...)**
- ▶ Example of client side REST request: GET /shoppingcart/5873
 - ▶ Server must be able to correctly interpret the client request as there is no explicitly defined equivalent to an interface definition

The standard Web architecture



The RESTful architecture

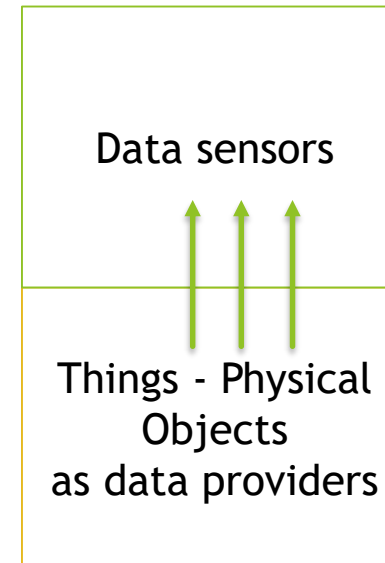


REST Architecture

- ▶ **Servers are stateless** and messages can be interpreted without examining history
 - ▶ Messages are self-contained
- ▶ There is no such thing as a “service” like procedures.
 - ▶ There are **just resources** which **are accessed through URI**
 - ▶ URI = generalization of URL
- ▶ Clients navigate through a series of steps towards a goal by following hypertext links (GET) and submitting representations (POST).
- ▶ *When WS-REST are implemented to access to physical objects, we talk about **Web of Things***

ROA and Mashup

- ▶ Mashups is “A way to create new Web applications by combining existing Web resources utilizing data and Web APIs” [Benslimane et al., 2008]
- ▶ ROA is **Well-adapted for Mashups** (Composite Web Applications)
- ▶ **Well-adapted for Web Sensors Network** (WSN)
- ▶ But **lacks** for non sensor device ... like actuators ...
- ▶ And then **for Sophisticated Devices**



REST - strong versus weak

- ▶ Pure REST should use 'pure' URI only
 - ▶ E.g. GET /shoppingcart/5873
- ▶ **Many REST implementations also allow parameter passing**
 - ▶ E.g. GET /shoppingcart/5873?sessionID=123
- ▶ Allowing parameter passing makes REST a lot more usable but blurs the architectural principle of statelessness
- ▶ Indeed Data can be specific command like instruction code ...
 - ▶ But is it the purpose ?
 - ▶ Is this not another way to rebuild a SOA stack ?

Service Oriented Architecture (SOAP-WS)

SOA : Service oriented Architecture

- ▶ A service provides business functions to its consumer and in ISO 19119 [ISO/TC-211] it is defined as
 - ▶ “ Distinct part of the functionality that is provided by an entity through interfaces ”
- ▶ Also called WS-* (for * recommendations, Cf. <https://www.w3.org/2002/ws/>)
- ▶ SOAP based Web Service, the alternative
- ▶ RPC using SOAP protocol over HTTP

Sample SOAP RPC Message

- ▶ <Envelope> is the root node
- ▶ <Header>, <Body> et <Fault> are children nodes :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
<soap:Header>
```

```
... Header information...
```

```
</soap:Header>
```

```
<soap:Body>
```

```
... Body information...
```

```
<soap:Fault> ...Fault information...
```

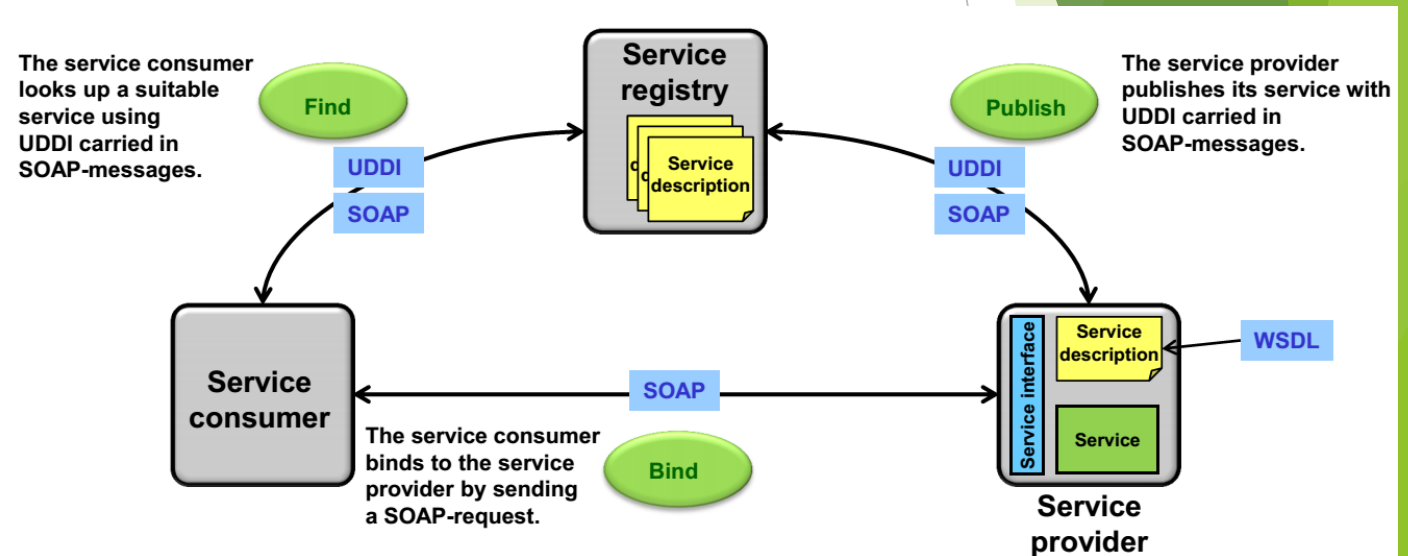
```
</soap:Fault>
```

```
</soap:Body>
```

```
</soap:Envelope>
```

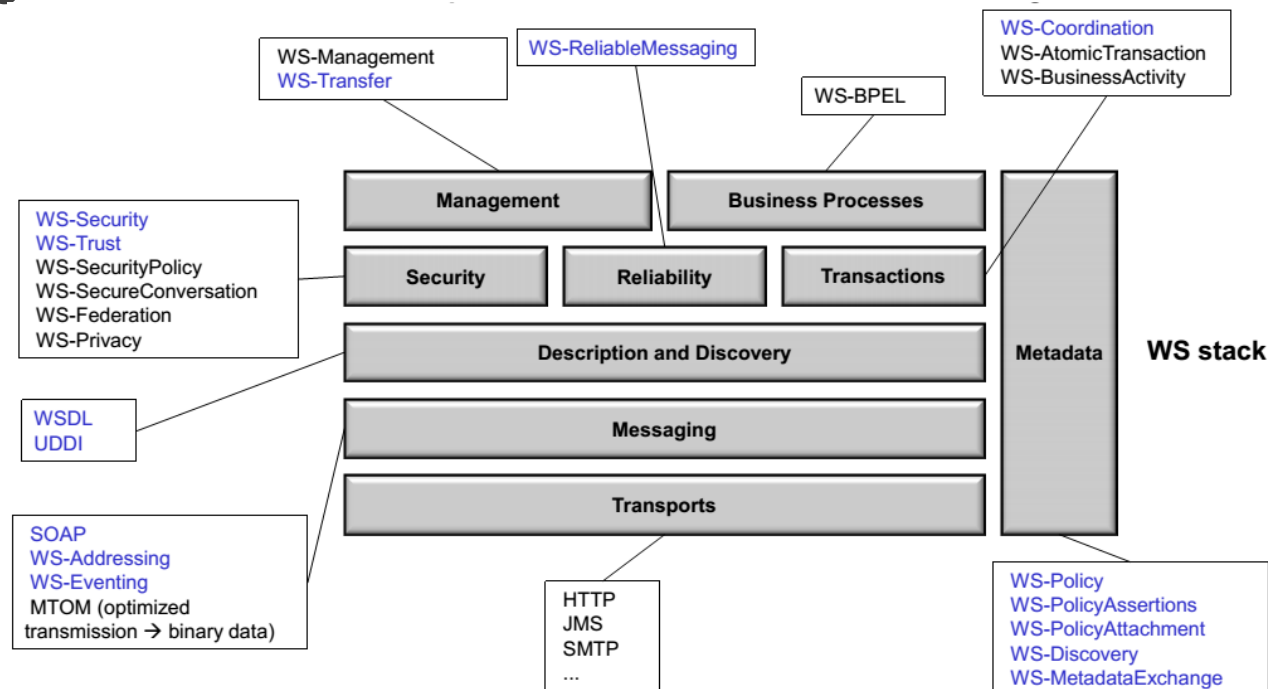
WS-*architecture more than ROA

- ▶ SOAP+WSDL+UDDI defines a general model for a web service architecture.
 - ▶ SOAP: Simple Object Access Protocol
 - ▶ WSDL: Web Service Description Language
 - ▶ UDDI: Universal Description and Discovery Protocol
 - ▶ Service consumer: User of a service
 - ▶ Service provider: Entity that implements a service (=server)
 - ▶ Service registry : Central place where available services are listed and advertised for lookup



WS-* Models

- ▶ Stack of WS-standards
- ▶ The W3C and OASIS WS-stack provide a framework / toolbox for constructing web service architectures



Disadvantages of Web Services

- ▶ Low-level abstraction
 - ▶ leaves a lot to be implemented
- ▶ Interaction patterns have to be built
 - ▶ one-to-one and request-reply provided
 - ▶ one-to-many?
- ▶ No location transparency
- ▶ **HEAVY STACK for tiny devices !**

CoAP : Constrained Application Protocol

LightWeight RESTFUL protocol for IoT and M2M ...

Over UDP

RFC 7252 (IETF 06-2014)

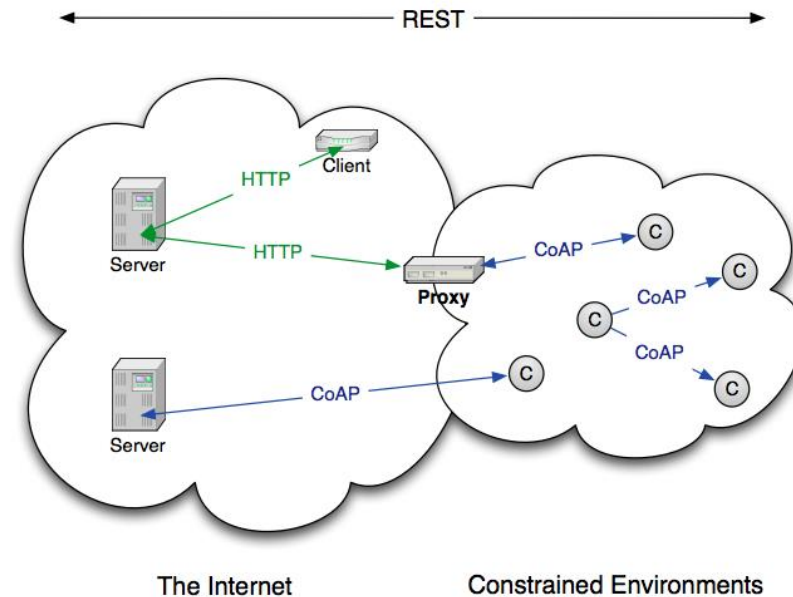
What CoAP is (and is not)

▶ CoAP is

- ▶ A RESTful protocol
- ▶ Both synchronous and asynchronous
- ▶ For constrained devices (small mem, slow proc) and networks
- ▶ Specialized for M2M applications
- ▶ Easy to proxy to/from HTTP

▶ CoAP is not

- ▶ A replacement for HTTP
- ▶ General HTTP compression
- ▶ Separate from the web



Navigateur Web / Applications M2M				
HTTP		CoAP		Application
TCP		UDP		Transport
IPv4 / IPv6			IPv6	Réseau
			6LoWPAN	
UMTS / GPRS	802.3 Ethernet	802.11 Wifi	802.15.4 LoWPAN	Physique et Liaison de Données

CoAP/protocol

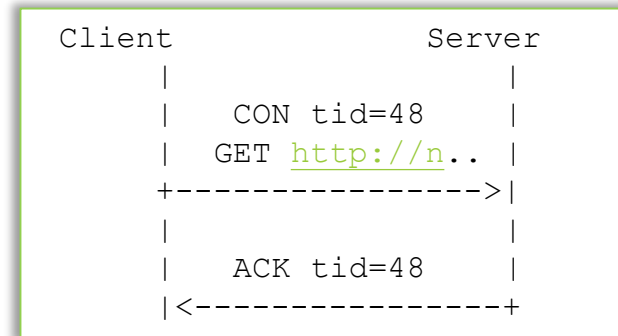
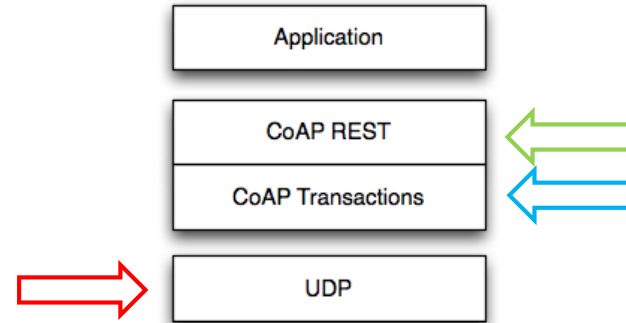
- ▶ Endpoint

- ▶ IP addr, UDP port

- ▶ CoAP Transactions

- ▶ CoAP Message Format

- ▶ 4 bytes header
- ▶ Options
- ▶ Payload
 - ▶ uint (unsigned integer)
 - ▶ string
 - ▶ ...



CoAP/transport and Endpoint

- ▶ **Endpoint**

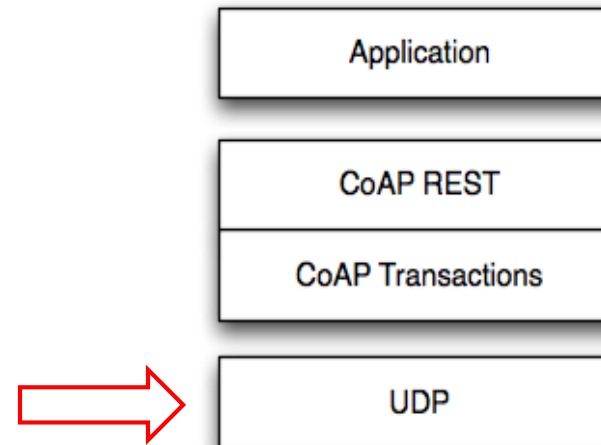
- ▶ IP addr, UDP port

- ▶ **Transport Protocol**

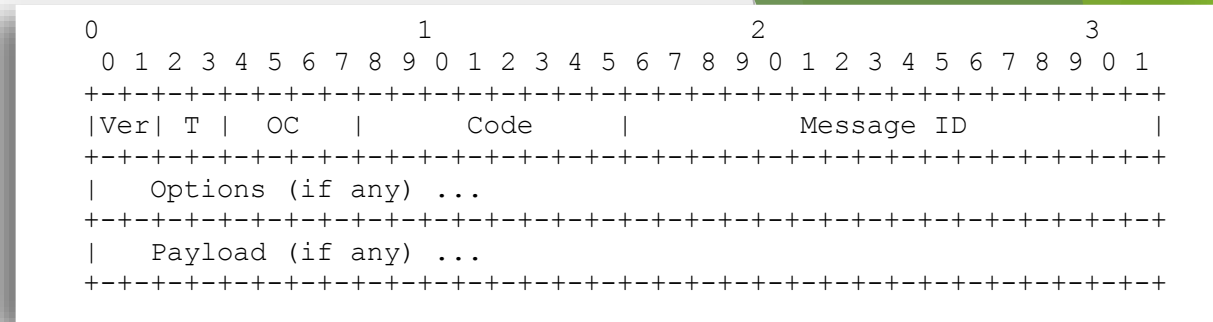
- ▶ **Default UDP but not required**
- ▶ SMS, TCP and SCTP also possible

- ▶ **Ports**

- ▶ UDP Port 5683 (mandatory)
- ▶ UDP Ports 61616-61631 compressed 6lowPAN

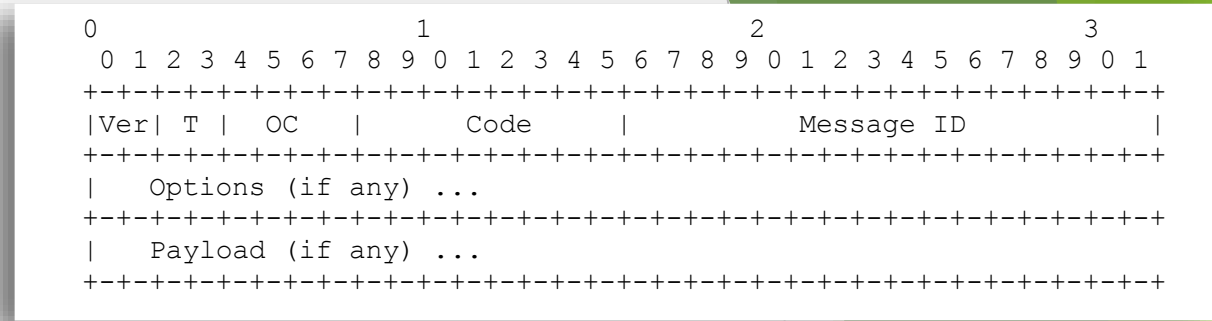


CoAP/protocol



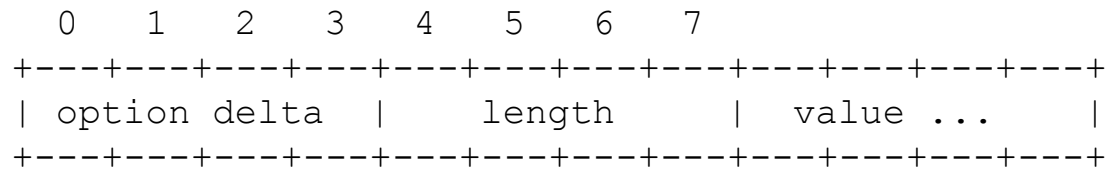
- ▶ The **first 4 bytes** that are mandatory contain the following pieces of information:
 - ▶ A. Byte 0
 - ▶ a. 2-bit version: The first two bits indicate the CoAP **version number**. As of now, only version 1 is supported.
 - ▶ b. 2-bit type code: The next two bits indicate the **message type**. This can take one of 4 values - **CON, NON, ACK, RST**
 - ▶ c. 4-bit token length: The next 4 bits indicate the **length of the token value in bytes**. As explained before, the token is used to correlate messages. The length of token can be between 0-8 bytes. Other values are reserved.
 - ▶ B. Byte 1 - This contains the **message code**.
 - ▶ The message **code values can be GET, PUT, POST, NOT FOUND** etc. I will talk about other possible message codes later.
 - ▶ C. Byte 2,3 - The next two bytes together make up a 16-bit number.
 - ▶ This is where the **message ID** is carried. This is an unsigned number.

CoAP/protocol Options



- ▶ After the first 4 bytes, based on the context, the message may contain **additional bytes**

Typical Option:



CoAP/example

```

Client  Server
|      |
|      |
+----->|   Header: GET (T=CON, Code=1, MID=0x7d34)
| GET   |   Uri-Path: "temperature"
|      |
|      |
|<-----+   Header: 2.05 Content (T=ACK, Code=69, MID=0x7d34)
| 2.05 |   Payload: "22.3 C"
|      |

```

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
| 1 | 0 | 1 |   GET=1   |   MID=0x7d34   |
+-----+-----+-----+-----+
| 11 | 11 |   "temperature" (11 B) ...
+-----+-----+-----+-----+

```

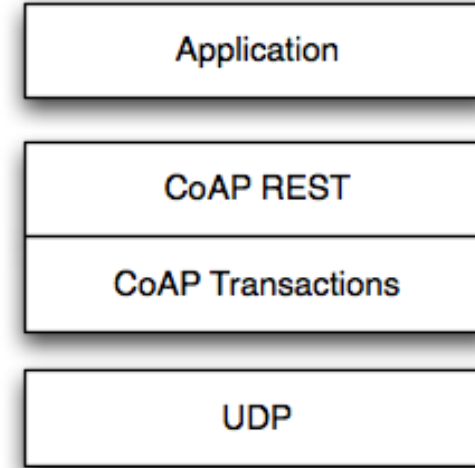
```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
| 1 | 2 | 0 |   2.05=69   |   MID=0x7d34   |
+-----+-----+-----+-----+
|   "22.3 C" (6 B) ...
+-----+-----+-----+-----+

```

The Transaction Model

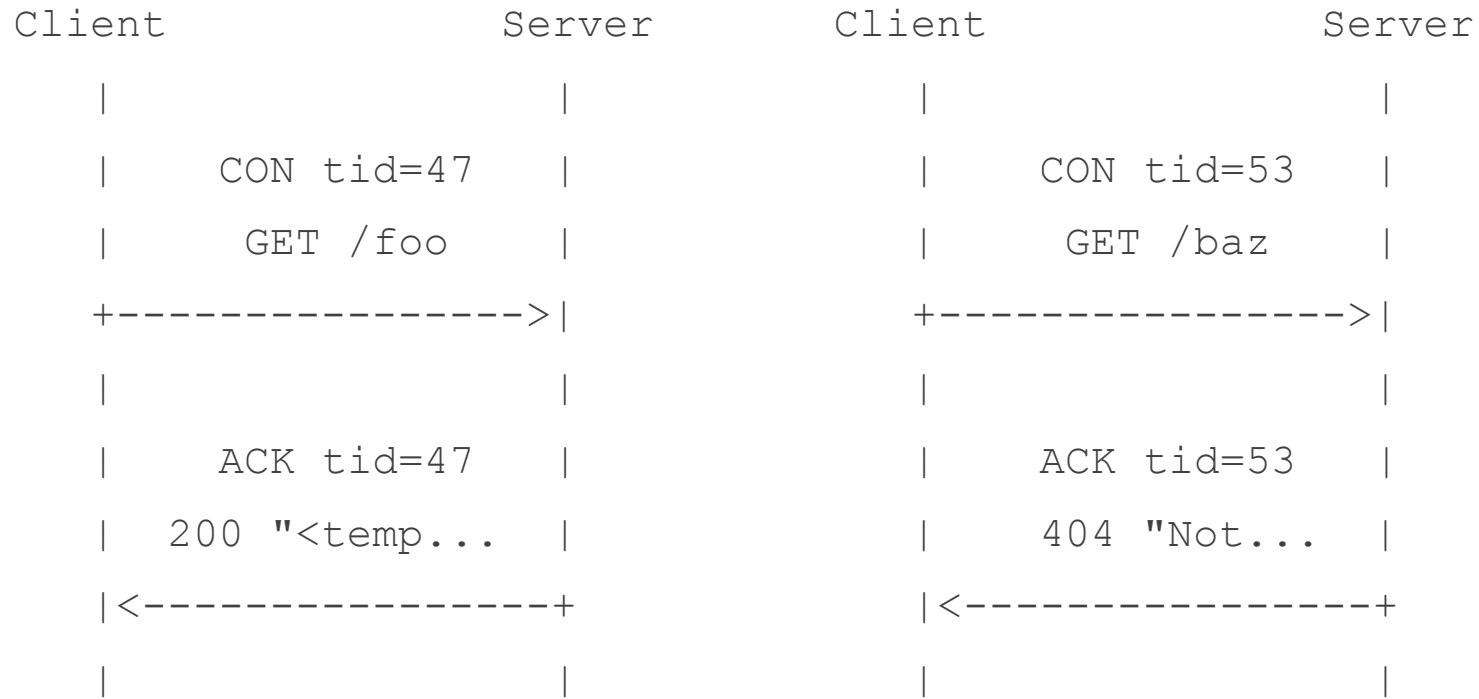
- ▶ Transport
 - ▶ CoAP is defined for **UDP**
- ▶ Transaction
 - ▶ Single message exchange between end-points
 - ▶ **CON, NON, ACK, RST**
- ▶ REST
 - ▶ Piggybacked on transaction messages
 - ▶ Method, **Response Code** and Options (URI, content-type etc.)



CoAP/message types

- ▶ Confirmable message
- ▶ Non-confirmable message
- ▶ Ack message
- ▶ Reset message

Synchronous Transaction

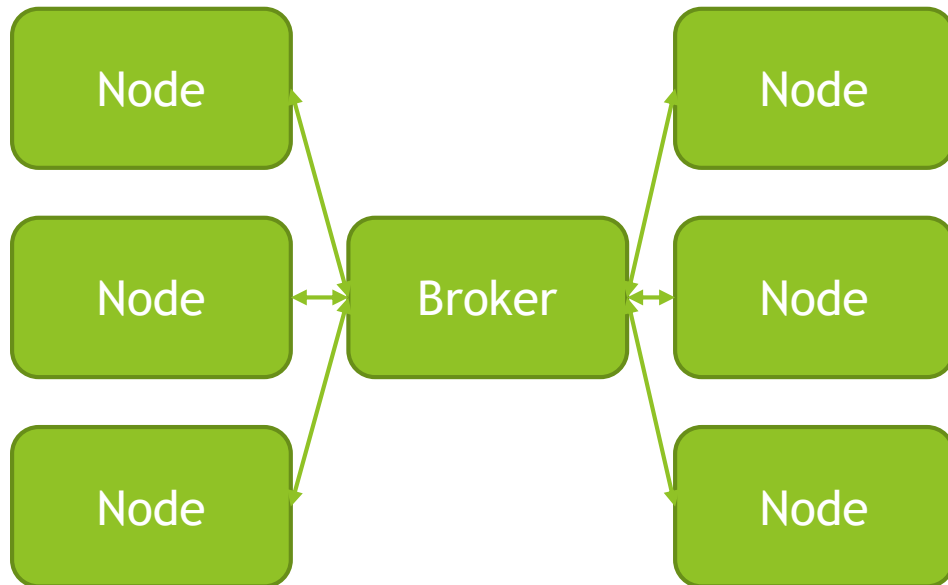


Asynchronous Transaction

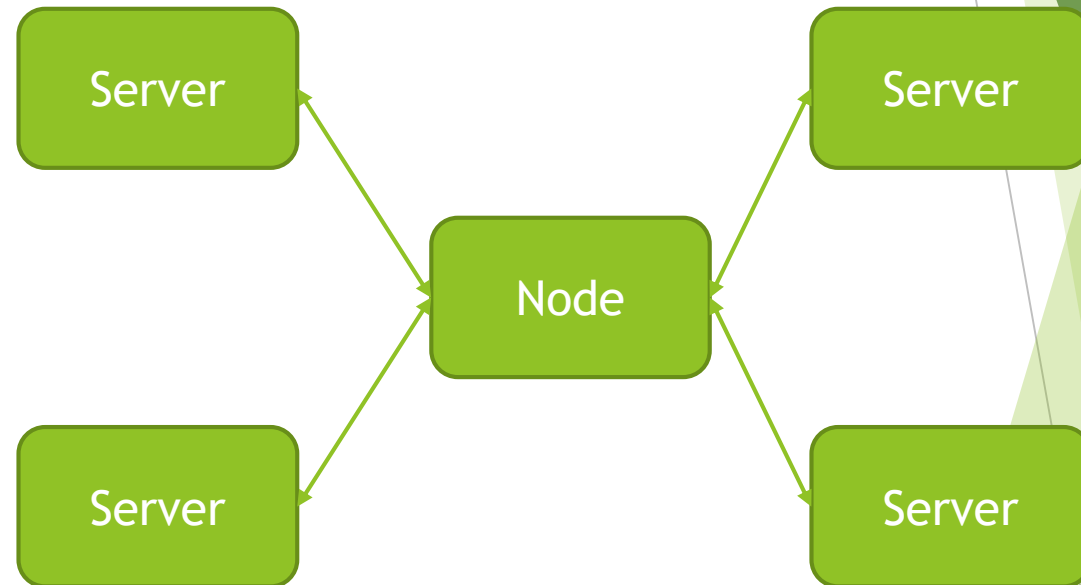
```
Client          Server
|              |
|  CON tid=48   |
|  GET http://n.. |
+----->|
|              |
|  ACK tid=48   |
|<-----+
|              |
... Time Passes ...
|              |
|  CON tid=783  |
|  200 http://n.. |
|  "<html..     |
|<-----+
|              |
|  ACK tid=783  |
+----->|
|              |
```

CoAP vs MQTT

► MQTT



► CoAP



CoAP vs MQTT

▶ MQTT

- ▶ Publish/Subscribe
- ▶ Non RESTful
- ▶ QoS
- ▶ Store and Forward

- ▶ Good for transferring data/commands over unstable connections

▶ Both

- ▶ Well suited for low volumes networks and low power devices (IoT)
- ▶ Can use secure connections (CoAP uses DTLS as preferred cryptographic method)

▶ CoAP

- ▶ Web Service
- ▶ RESTful
- ▶ No QoS in protocol

- ▶ Goof for client/server concepts over stable connections. Nodes only execute “commands”

And...

What do we need now?

Web Service for a “Find” layer for IoT

- ▶ Provides a way to **find** and **locate** relevant services (devices) on the Web
 - ▶ Search engines,
 - ▶ Crawlers,
 - ▶ etc...
- ▶ Some standard provides some protocols for
 - ▶ Dynamic discovery
 - ▶ Availability Management
 - ▶ Ex. UPnP and DPWS
 - ▶ We’ll see that in the next course

