LAU
الجامعة اللبنانية الأميركية
Lebanese American University

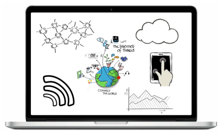
CSC 498R: Internet of Things

Lecture 08: Web of Things (WoT) and Beyond …

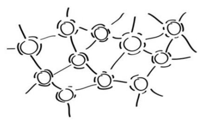
Instructor: Haidar M. Harmanani

Fall 2017

IoT Components



- Things we connect: **Hardware, sensors *and* actuators**



- Connectivity
  – Medium we use to connect things

- Platform
  – Processing and storing collected data
    - Receive and send data via standardized interfaces or API
    - Store the data
    - Process the data.

- Analytics
  – Get insights from gathered data

- User Interface

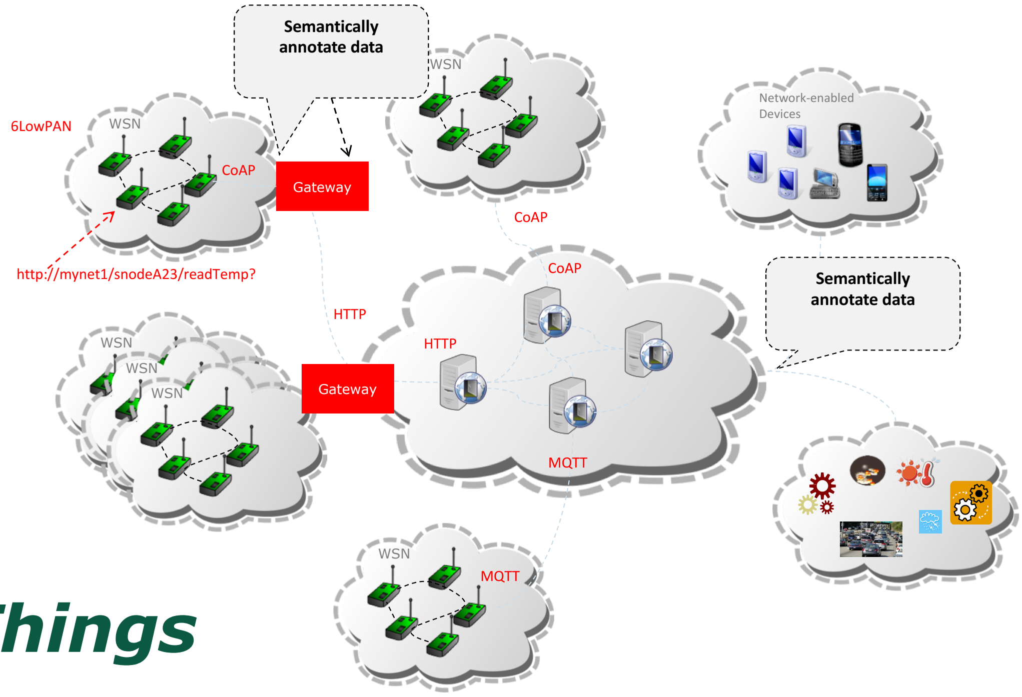
Web of Things

- Integrating the real world data into the Web and providing Web-based interactions with the IoT resources is also often discussed under umbrella term of "Web of Things" (WoT).
- WoT data is not only large in scale and volume, but also continuous, with rich spatiotemporal dependency.

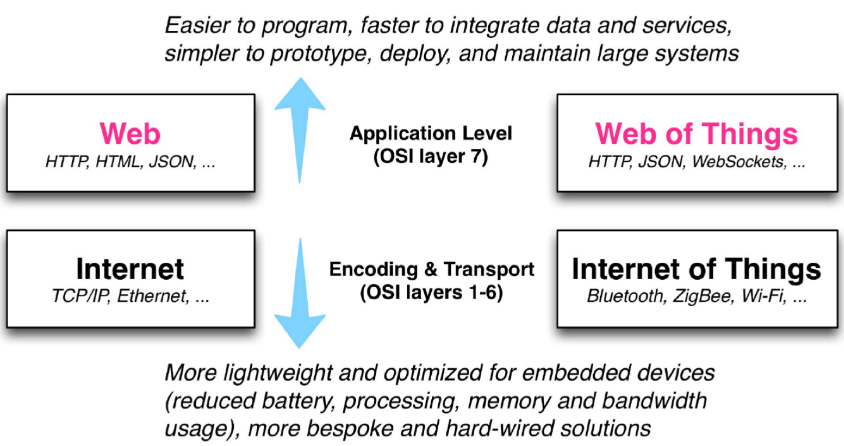
Web of Things

- Connecting sensor, actuator and other devices to the World Wide Web.
  - "Things' data and capabilities are exposed as web data/services.
- Enables an interoperable usage of IoT resources (e.g. sensors, devices, their data and capabilities) by enabling web based discovery, access, tasking, and alerting.

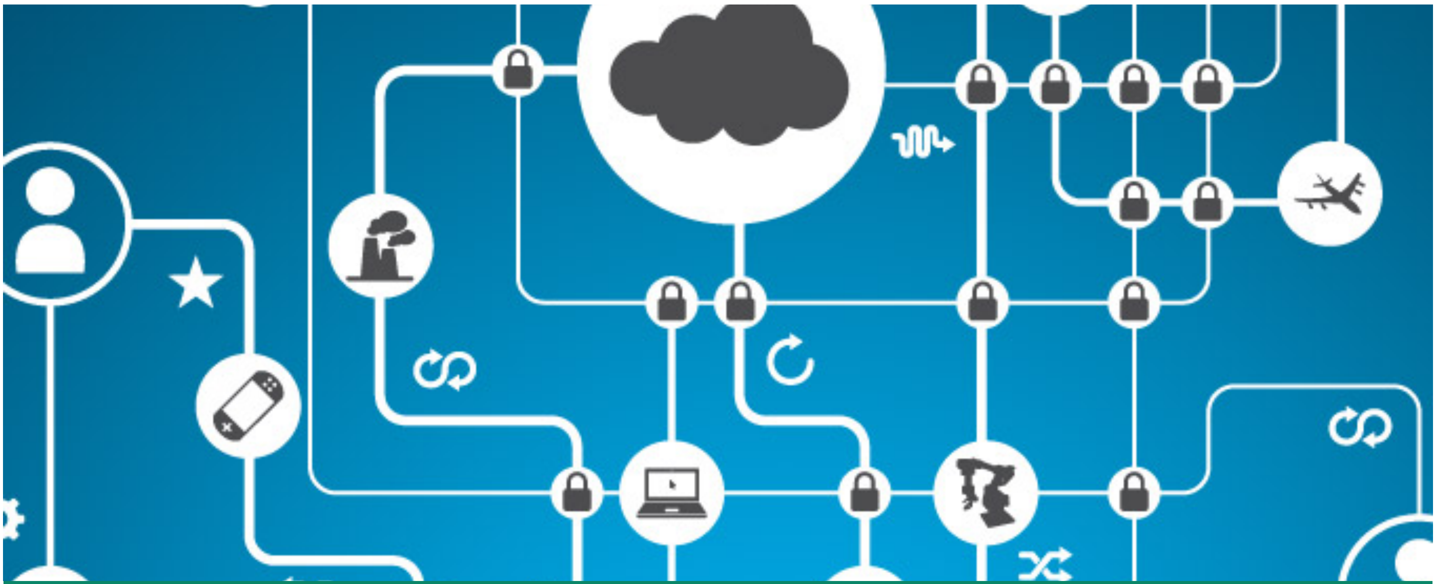
Web of Things



Web of Things



Source: Building the Web of Things: book.webofthings.io

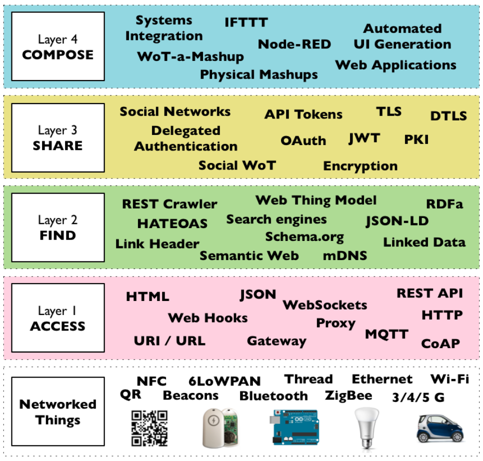


Part I: Programming the Web of Things

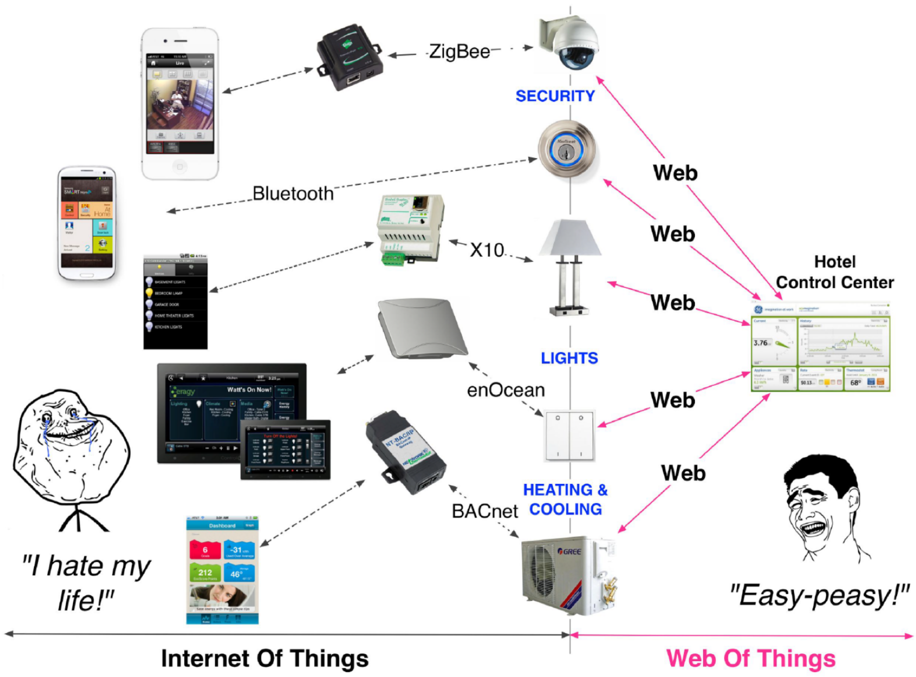
The Web of Things is a refinement of the Internet of Things by integrating smart things not only into the Internet (network), but into the Web Architecture (application)

Dominique Guinard

The Web of Things Architecture



Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0



Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

Example: Hello WoT Using `Node.js`

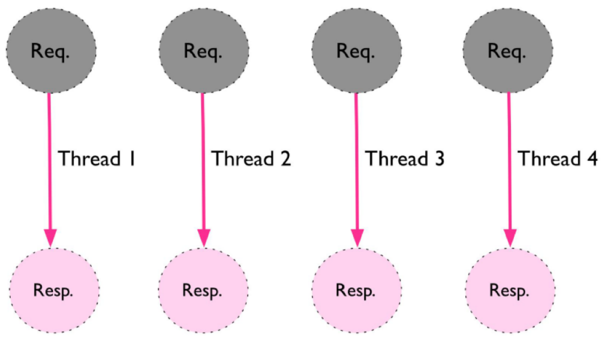
- Raspberry Pi
- Processor
  - With a fast processor, you can process images, sound, data from sensors on the board itself.
  - For example, use the computer vision library OpenCV.
- Network
  - Use the `Node.js` framework to use network capabilities.
  - `Node.js` is built on the Chrome JS runtime.

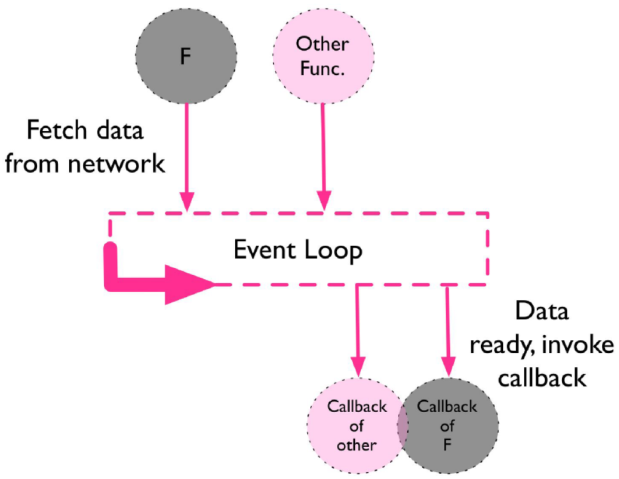
`Node.js`

Why is node.js so good ?

- What is `Node.js` exactly ?
  - `Node.js` is a server-side `Javascript` solution, written in C.
  - It allows programmers to write JavaScript program and to execute them as a standalone application on a server.
- What should I do ?
  - You easily develop server applications, typing less than 20 lines of code !
  - `Node.js` is very fast : non-blocking, asynchronous architecture! It's a way to be able to provide complex tools by using a simple and powerful high level language.

Why Node?





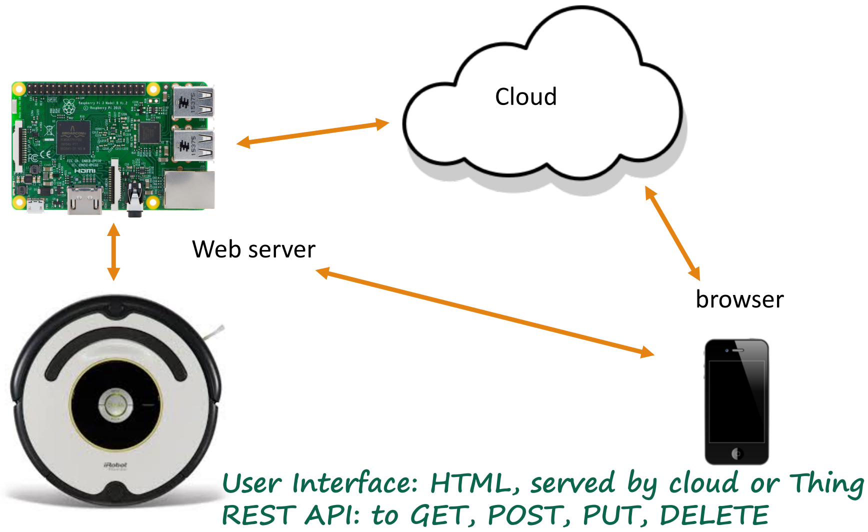
Hello `Node.js`

- Install NVM and **`Node.js`** on your Pi and computer
  - `curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.2/install.sh | bash`
  - `nvm install v4.8.3`
- Build a Node HTTP server

JS Debugging

- Debugging Chrome Dev Tools
  - very powerful html, css, & JS debugger
    - use for **`Node.js`** & browser apps
- safari, IE have similar tools

Back to our Application



User Interface: HTML, served by cloud or Thing
REST API: to GET, POST, PUT, DELETE

`Node.js` Hello World

- Goal
  - `Node.js` is a framework using the Chrome JavaScript runtime and used to communicate over networks.
  - Here we'll create a web server listening on a port and answering `HelloWorld` in your browser
- Steps
  - Create a folder, copy paste the content of the next slide in a file called **`hello.js`**
  - Replace the `X.X.X.X` IP in the file with your `Raspberry Pi` IP.

Node.js Hello World

```
var http = require('http');
http.createServer(
  function (req, res) {
    res.writeHead(200,{'Content-Type': 'text/plain'});
    res.end('Hello World\n');
  }
).listen(1337, 'X.X.X.X');
console.log('Server running at http://X.X.X.X:1337/');
```


Node.js Hello World

- Launch the server
  - Launch the command : `node hello.js`
- See the result
  - From a PC on the same network, launch a browser with : `http://X.X.X.X:1337`
  - You should see `Hello World`

IoT Application Architecture With `Node.js`

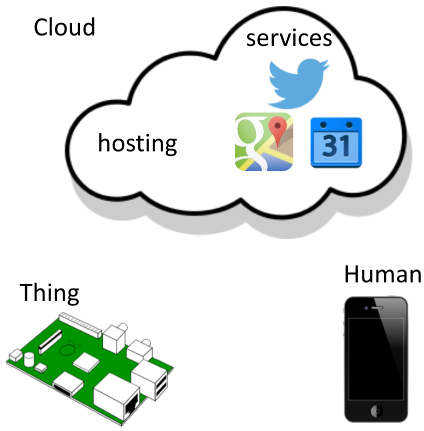
COMPONENTS & FUNCTION

Device: connect to physical world
- Microcontroller: behavior
- Sensors: switch, knob, temp, video, mic
- Actuators: led, speaker, servo, stepper motor

PC/Tablet/phone: user interface
- Observe, control

Server
- Proxy, connectivity
- Aggregation: collect sensor data
- Computation: speech/vision
- Storage: persistent logging
- Security: authentication
- Services: geocoding

TOPOLOGY



JavaScript in One Slide …

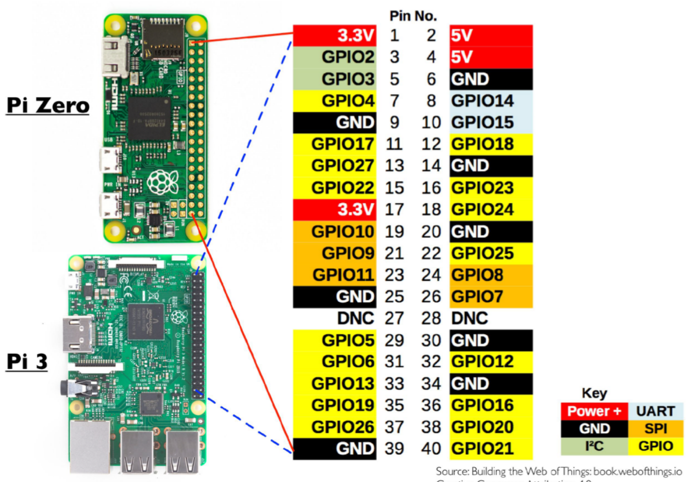
- Started as scripting language for browser
  - Now for server, applications
- Similar syntax to C & Java
  - `if (a > 3) b = 2;`
- Dynamic typing

```
> var a = 1
> a + 1
2
> a = '1'
> a + 1
'11'
```

Environments for Javascript

- Browser: runs on everything with a display
  – PC, phone, tablet, …
- Nodejs: does not require display
  – Laptop, server, beaglebone, raspberry pi, galileo

WoT Sensing and Actuating



Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

GPIO support via Node on Embedded Systems

J5

http://johnny-five.io

https://github.com/intel-iot-devkit/mraa

CYLON.JS

https://cylon.js

fivdi / onoff

https://github.com/fivdi/onoff

heimcontrol.js

http://ni-c.github.io/heimcontrol.js/

Node.js: Raspberry Pi

```
var gpio = require('rpi-gpio');
gpio.setup(7, gpio.DIR_OUT, write);

function write() {
  gpio.write(7, true, function(err) {
    if (err) throw err;
    console.log('Written to pin'); });
}
```

Connecting a PIR Sensor

```
var Gpio = require('onoff').Gpio,
sensor = new Gpio(17,'in','both');//#A
sensor.watch(function(err, value) {//#B
    if(err)
        exit(err);
    console.log(value ?'there is someone!':'not anymore!');
});

function exit(err) {
    if (err)console.log('An error occurred: '+ err);
    sensor.unexport();
    console.log('Bye, bye!’)
    process.exit();
}
process.on('SIGINT',exit);
```


blink.js: the Hello World of the IoT

```
var onoff = require('onoff');

var Gpio = onoff.Gpio,
  led = new Gpio(4, 'out'),
  interval;

interval = setInterval(function () {
  var value = (led.readSync() + 1) % 2;
  led.write(value, function() {
    console.log("Changed LED state to: " + value);
  });
}, 2000);

process.on('SIGINT', function () {
  clearInterval(interval);
  led.writeSync(0);
  led.unexport();
  console.log('Bye, bye!');
  process.exit();
});
```

Import the onoff library.

Initialize GPIO 4 to be an output pin.

This interval will be called every two seconds.

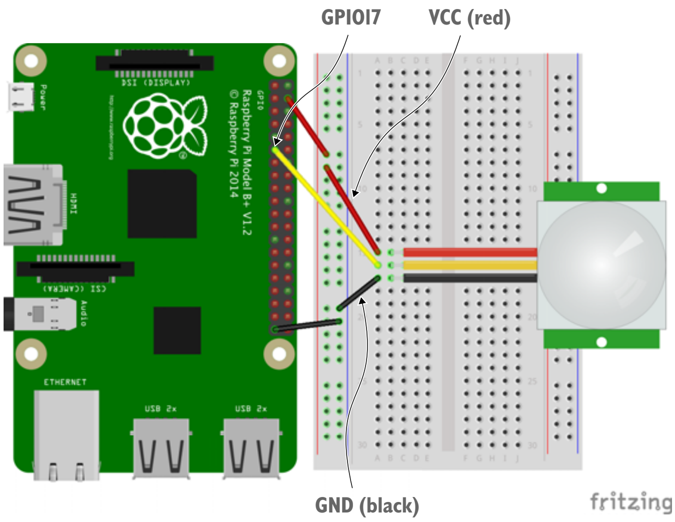
Synchronously read the value of pin 4 and transform 1 to 0 or 0 to 1.

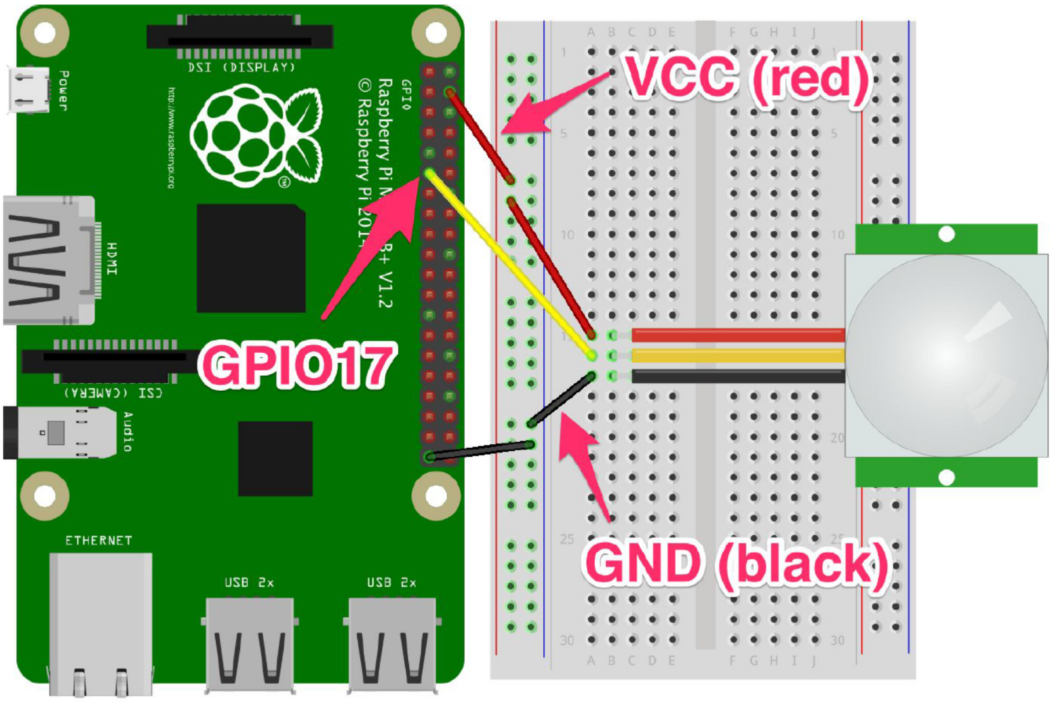
Asynchronously write the new value to pin 4.

Listen to the event triggered by Ctrl-C.

Cleanly close the GPIO pin before exiting.

Connecting a Proximity Sensor





`pir.js`: Reading a PIR Sensor Using the `onoff` Library

```
var Gpio = require('onoff').Gpio,
  sensor = new Gpio(17, 'in', 'both');

sensor.watch(function (err, value) {
  if (err) exit(err);
  console.log(value ? 'there is some one!' : 'not anymore!');
});

function exit(err) {
  if (err) console.log('An error occurred: ' + err);
  sensor.unexport();
  console.log('Bye, bye!')
  process.exit();
}
process.on('SIGINT', exit);
```

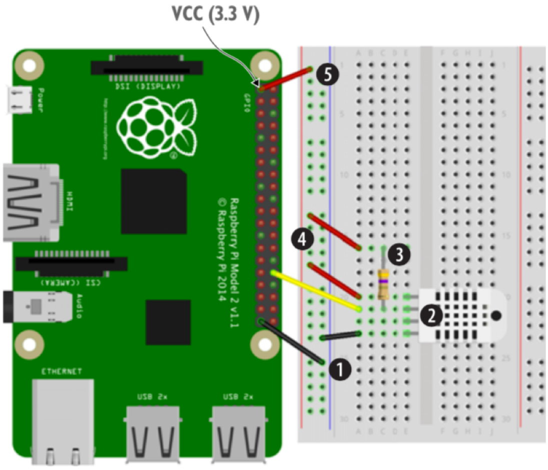
Initialize pin 17 in input mode; 'both' means you want to handle both rising and falling interrupt edges.

Listen for state changes on pin 17; if a change is detected, the anonymous callback function will be called with the new value.

`dht.js`: Connecting a Temperature and a Humidity Sensor

- Use a DHT22 sensor
  - Connect the first DHT22 pin to a ground (GND) pin; for example, pin 39.
  - Don't connect anything to the second pin
  - Connect the third pin to the GPIO 12 of your Pi (pin 32) and place a 4.7K Ohm resistor (yellow, violet, red, gold/silver) 33between the DH22 pin and the connection to the pin of the Pi.
  - Connect this resistor to the VCC line on the breadboard, the red line.
  - Connect the fourth pin to the VCC line on the breadboard.
  - Connect the 3.3-volt power source to the VCC line on the breadboard.

dht.js: Connecting a Temperature and a Humidity Sensor



1. Connect the first DHT22 pin to a ground (GND) pin; for example, pin 39. Don't connect anything to the second pin
2. Connect the third pin to the GPIO 12 of your Pi (pin 32) and place a 4.7K Ohm resistor (yellow, violet, red, gold/silver) 33between the DH22 pin and the connection to the pin of the Pi.
3. Connect this resistor to the VCC line on the breadboard, the red line.
4. Connect the fourth pin to the VCC line on the breadboard.
5. Connect the 3.3-volt power source to the VCC line on the breadboard.

dht.js: Connecting a Temperature and a Humidity Sensor

- Because the DHT22 uses a special protocol, you'll first need to install an additional driver on the Pi called the BCM 2835 C Library

```
$ tar zxvf bcm2835-1.50.tar.gz
$ cd bcm2835-1.50
$ ./configure
$ make
$ sudo make check
$ sudo make install
```

dht.js: Connecting a Temperature and a Humidity Sensor

```
var sensorLib = require('node-dht-sensor');

sensorLib.initialize(22, 12); //#A
var interval = setInterval(function () { //#B
  read();
}, 2000);

function read() {
  var readout = sensorLib.read(); //#C
  console.log('Temperature: ' + readout.temperature.toFixed(2) + 'C, ' + //#D
    'humidity: ' + readout.humidity.toFixed(2) + '%');
};

process.on('SIGINT', function () {
  clearInterval(interval);
  console.log('Bye, bye!');
  process.exit();
});
```


Node.js /Express

- Node.js has the infrastructure to write a web server
  - Receive HTTP requests, send response
- Express: node package, middleware that makes it easy to make a server
  - Serve static assets: HTML, JS, CSS, PNG, …
  - REST API
    - Parse URLs, Route requests, render HTML templates, set MIME types

Getting Started With Express

- Install node
- Install express generator & nodemon

```
npm install -g express-generator nodemon
```

- Make an app

```
express .
```

- Launch it

```
nodemon bin/www
```


Hosting

- Easy to run nodejs on laptop, but networking may make sharing difficult
- PaaS like Heroku make it easy to deploy your service
  - IaaS: machine/OS, amazon
  - PaaS: machine/OS/Application Stack/Scaling, Heroku, azure, elastic beanstalk
  - BaaS: No server programming required: db, Parse, Kinvey, xively
- PaaS
  - Get the program running locally, deploy to their server in the cloud
  - Free, https, managed

```
echo node_modules > .gitignore
git init
git add .
git commit -m 'initial version'
heroku create tpi-iotn
git push heroku master
```

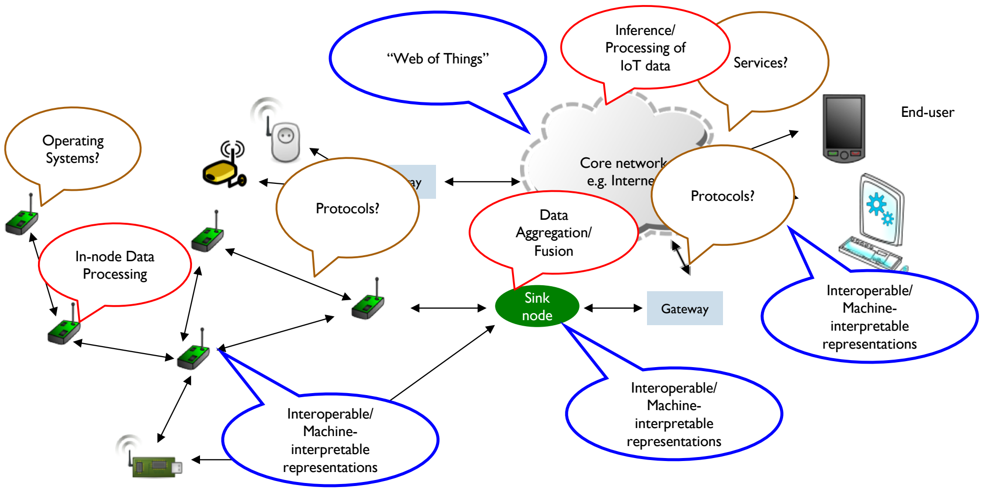
Part II: IoT/WoT Data and Semantic

Wireless Sensor (and Actuator) Networks Revisited

- Sensors (and in general "Things") are increasingly being connected with Web infrastructure.
- This can be supported by embedded devices that directly support IP and web-based connection (e.g. 6LowPAN and CoAp) or devices that are connected via gateway components.
  - Broadening the IoT to the concept of "Web of Things"

Wireless Sensor (and Actuator) Networks Revisited

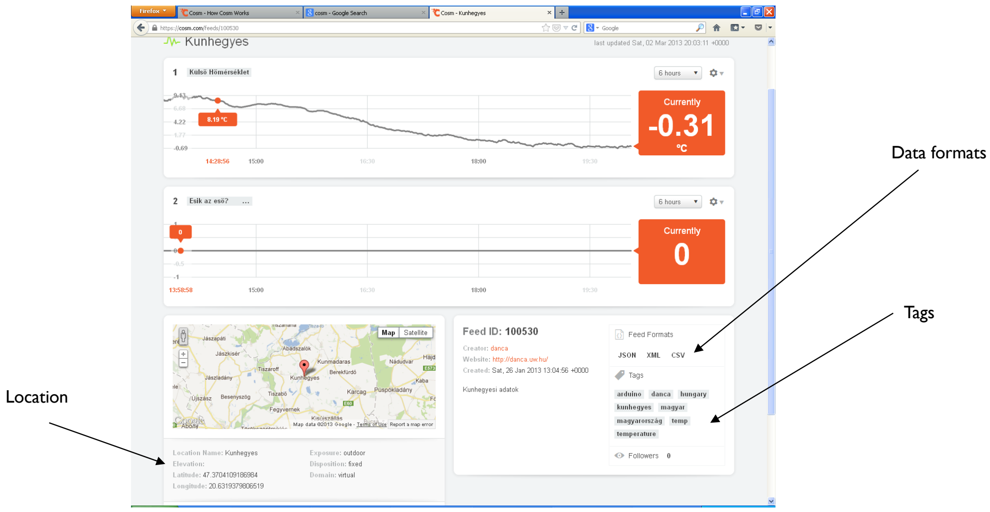
- Standards such as *Sensor Web Enablement (SWE)* are widely being adopted in industry, government and academia.
- While such frameworks provide some interoperability, semantic technologies are increasingly seen as key enabler for integration of IoT data and broader Web information systems.



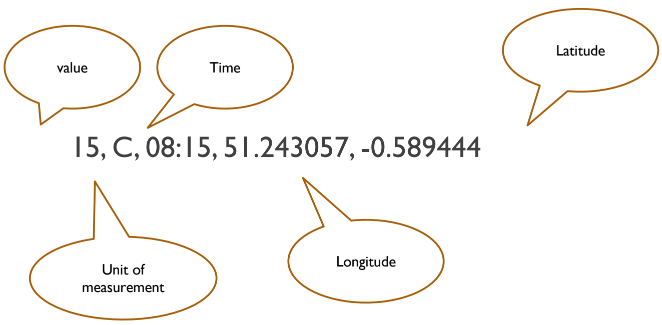
The networks typically run Low Power Devices and cconsist of one or more sensors, could be different type of sensors (or actuators)

Wireless Sensor (and Actuator) Networks

Observation and Measurement Data-Annotation

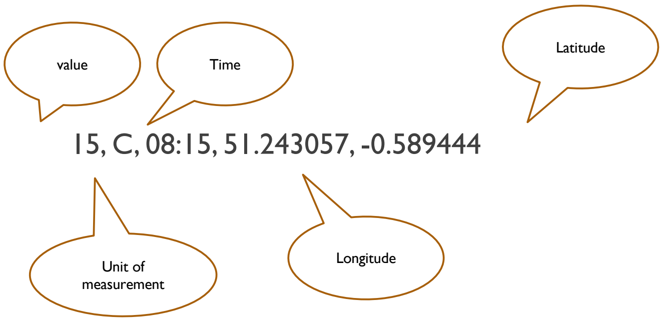


Observation and Measurement Data



How to make the data representations more machine-readable and machine-interpretable;

Observation and Measurement Data



What about this?

```
<value>15</value>
<unit>C</unit>
<time>08:15</time>
<longitude>51.243057</longitude>
<latitude>-0.58944</latitude>
```


Extensible Markup Language (XML)

- XML is a simple, flexible text format that is used for data representation and annotation.
- XML was originally designed for large-scale electronic publishing.
- XML plays a key role in the exchange of a wide variety of data on the Web and elsewhere.
- It is one of the most widely-used formats for sharing structured information.

XML Document Example

```
<?xml version="1.0"?>
<measurement>
    <value>15</value>
    <unit>C</unit>
    <time>08:15</time>
    <longitude>51.243057</longitude>
    <latitude>-0.58944</latitude>
</measurement>
```

XML Prologue- the XML declaration

Root element

XML elements

XML documents MUST be "well formed"

XML Document Example: Attributes

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<measurement>
    <value type="Decimal">15</value>
    <unit>C</unit>
    <time>08:15</time>
    <longitude>51.243057</longitude>
    <latitude>-0.58944</latitude>
</measurement>
```

Well Formed XML Documents

- A "Well Formed" XML document has correct XML syntax.
- XML documents must have a root element
- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must be quoted

Validating XML Documents

- A "Valid" XML document is a "Well Formed" XML document, which conforms to the structure of the document defined in an XML Schema.
- XML Schema defines the structure and a list of defined elements for an XML document.

XML Schema- example

```
<xs:element name="measurement">

<xs:complexType>
  <xs:sequence>
    <xs:element name="value" type="xs:decimal"/>
    <xs:element name="unit" type="xs:string"/>
    <xs:element name="time" type="xs:time"/>
    <xs:element name="longitude" type="xs:double"/>

    <xs:element name="latitude" type="xs:double"/>
  </xs:sequence>
</xs:complexType>

</xs:element>
```

- *XML Schema defines the structure and elements*
- *An XML document then becomes an instantiation of the document defined by the schema;*

XML Documents: Revisiting the Example

"But how about this?"

```
<?xml version="1.0"?>
<sensor_data>
  <reading>15</reading>
  <u>C</u>
  <timestamp>08:15</timestamp>
  <long>51.243057</long>
  <lat>-0.58944</lat>
</sensor_data>
```

XML: limitations for semantic markup

- XML representation makes no commitment on:
  - Domain specific ontological vocabulary
    - Which words shall we use to describe a given set of concepts?
  - Ontological modelling primitives
    - How can we combine these concepts, e.g. "car is a-kind-of (subclass-of) vehicle"

⇨ requires pre-arranged agreement on vocabulary and primitives

Semantic Web technologies

- XML provide a metadata format.
- It defines the elements but does not provide any modelling primitive nor describes the meaningful relations between different elements.
- Using semantic technologies can help to solve some of these issues.

"An extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in co-operation." (Tim Berners-Lee et al, 2001)

Semantic Web

IoT data: semantic related issues

- The current IoT data communications often rely on binary or syntactic data models which lack of providing machine interpretable meanings to the data.
  - Syntactic representation or in some cases XML-based data
  - Often no general agreement on annotating the data;
    - requires a pre-agreement between different parties to be able to process and interpret the data;
  - Limited reasoning based on the content and context data
  - Limited interoperability in data and resource/device description level;
  - Data integration and fusion issues.

Requirements

- Structured representation of concepts
  - Machine-interpretable descriptions
  - Reasoning and interpretation mechanisms
- Access mechanism to heterogeneous resource descriptions with diverse capabilities
- Automated interactions and horizontal integration with existing applications

What are the challenges?

- The models provide the basic description frameworks, but alignment between different models and frameworks are required.
- Semantics are the starting point, reasoning and interpretation of data is required for automated processes.
- Real interoperability happens when data/services from different frameworks and providers can be interchanged and used with minimised intervention.

Possible solutions

- The semantic Web has faced this problem earlier.
  - Proposed solution: using machine-readable and machine-interpretable meta-data
    - Important not: machine-interpretable but not machine-untreatable!
    - Well defined standards and description frameworks: RDF, OWL, SPARQL
    - Variety of open-source, commercial tools for creating/managing/querying and accessing semantic data
      - Jena, Sesame, Protégé, …
- An Ontology defines conceptualisation of a domain.
  - Terms and concepts
  - A common vocabulary
  - Relationships between the concepts
- There are several existing and emerging ontologies in the IoT domain.
- Automated annotation methods, dynamic semantics;

How to adapt the solutions

- Creating ontologies and defining data models are not enough
  - tools to create and annotate data
  - data handling components
- Complex models and ontologies look good, but
  - design lightweight versions for constrained environments
  - think of practical issues
  - make it as much as possible compatible and/or link it to the other existing ontologies
- Domain knowledge and instances
  - Common terms and vocabularies
    - Location, unit of measurement, type, theme, …
- Link it to other resource
- In many cases, semantic annotations and semantic processing should be intermediary not the end products.

Resource Description Framework

A world Wide Web Consortium (W3C) recommendation

Relationships between documents

Consisting of triples or sentences:

- <subject, property, object>
- <"Sensor", hasType, "Temperature">
- <"Node01", hasLocation, "Room_BA_01" >

RDFS extends RDF with standard "ontology vocabulary":

- Class, Property
- Type, subClassOf
- domain, range

RDF is the most mature standard for describing complex data and complex interrelationships for semantic data that do not fit into traditional relational database rows and columns or simple XML. It is also the standard that allows for data exchange of semantic graph databases that support contextual and conceptual meaning.

RDF

RDF for semantic annotation

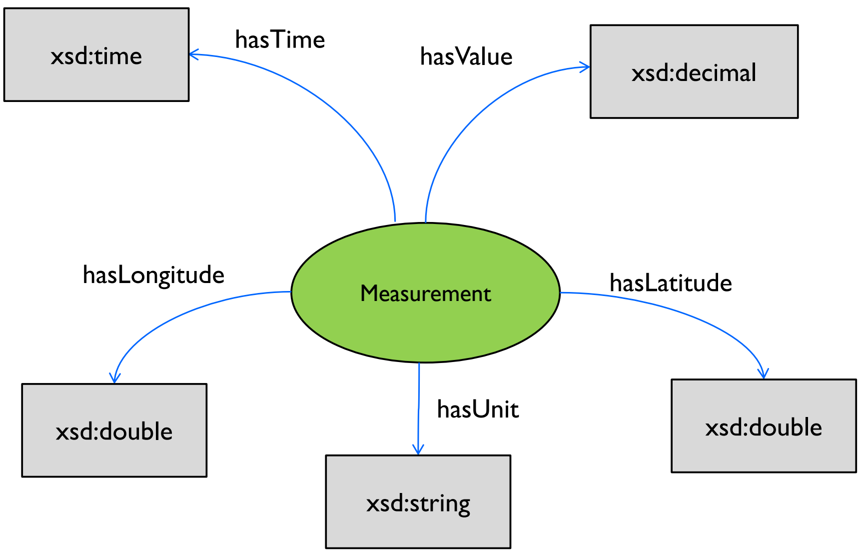
RDF provides metadata about resources

Object -> Attribute-> Value triples or

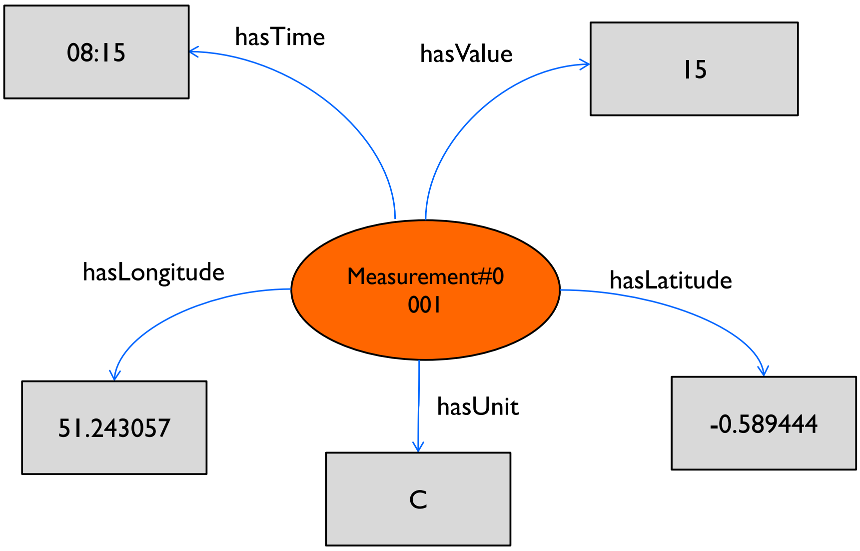
Object -> Property-> Subject

It can be represented in **XML**

The RDF triples form a **graph**



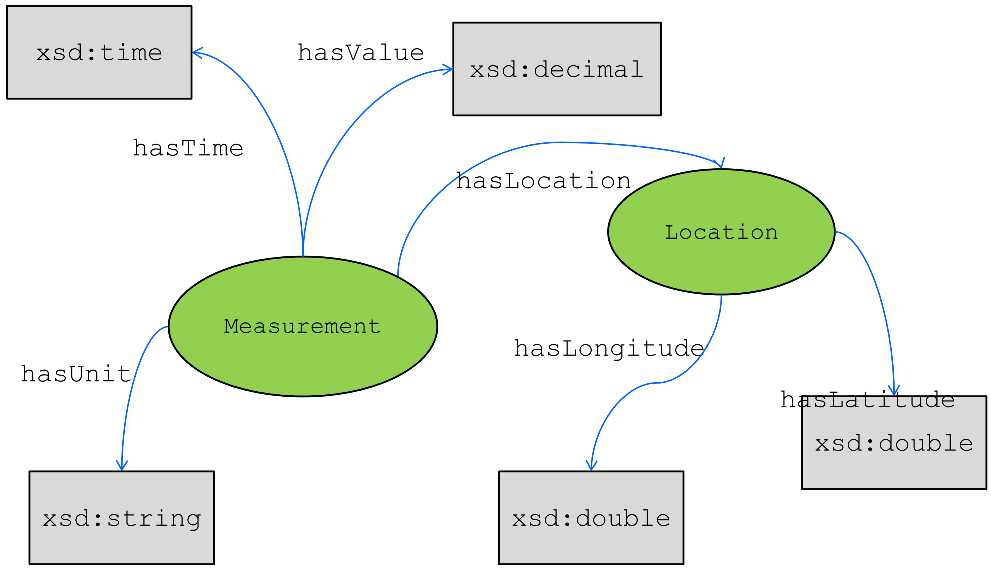
RDF Graph



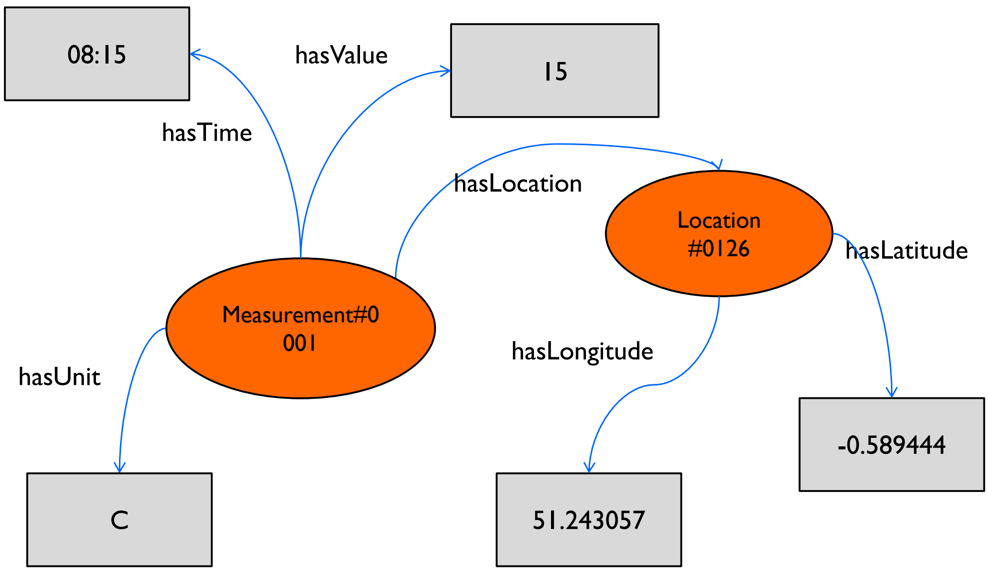
RDF Graph: An instance

RDF/XML

```
<rdf:RDF>
   <rdf:Description rdf:about="Measurment#0001">
   <hasValue>15</hasValue>
   <hasUnit>C</hasUnit>
   <hasTime>08:15</hasTime>
   <hasLongitude>51.243057</hasLongitude>
   <hasLatitude>-0.589444</hasLatitude>
   </rdf:Description>
</rdf:RDF>
```



More RDF



An instance of our model

RDF: Basic Ideas

Resources

- Every resource has a URI (Universal Resource Identifier)
- A URI can be a URL (a web address) or a some other kind of identifier;
- An identifier does not necessarily enable access to a resources
- We can think of a resources as an object that we want to describe it.
  - Car
  - Person
  - Places, etc.

RDF: Basic Ideas

- Properties
  - Properties are special kind of resources;
  - Properties describe relations between resources.
  - For example: "hasLocation", "hasType", "hasID", "sratTime", "deviceID",.
  - Properties in RDF are also identified by URIs.
  - This provides a global, unique naming scheme.
  - For example:
    - "hasLocation" can be defined as:
      - URI: http://www.loanr.it/ontologies/DUL.owl#hasLocation
  - SPARQL is a query language for the RDF data.
    - SPARQL provide capabilities to query RDF graph patterns along with their conjunctions and disjunctions.

JSON

- A subset of JavaScript, using its object literal notation
- A lightweight data-interchange format
  - Can be simply *evaled* in JavaScript, and parsed with little effort in most other languages.
- A popular alternative to XML

Evaluation

- JSON is simpler than XML and more compact
- No closing tags, but if you compress XML and JSON the difference is not so great
- XML parsing is hard because of its complexity
- JSON has a better fit for OO systems than XML, but not as extensible
- Preferred for simple data exchange by many
- MongoDB is a very popular open-source 'NoSQL' database for JSON objects

Example

```
{"firstName": "John",
 "lastName" : "Smith",
 "age"      : 25,
 "address"  :
   {"streetAdr" : "21 2nd Street",
     "city"     : "New York",
     "state"    : "NY",
     "zip"      : "10021"},
 "phoneNumber":
   [{"type"  : "home",
     "number": "212 555-1234"},
    {"type"  : "fax",
     "number" : "646 555-4567"}]
 }
```

- A JSON object with five key-value pairs
- Objects are wrapped by curly braces
- There are no object IDs
- Keys are strings
- Values are numbers, strings, objects or arrays
- Arrays are wrapped by square brackets

A Specification for serializing RDF in JSON

RDF/JSON

RDF in JSON

```
{
"data" : [
{"s" : { "type" : "uri" , "uri" : "http://example.org/about" } ,
 "p" : "http://purl.org/dc/elements/1.1/creator",
 "o" : { "type" : "literal", "val" : "Anna Wilder" }
},
{
"s" : { "type" : "uri" , "uri" : "http://example.org/about" } ,
"p" : "http://purl.org/dc/elements/1.1/title",
"o" : { "type" : "literal", "val" : "Anna's Homepage", "lang" :"en" }
}
]}
```

Flat Triples Approach


```
{
  "@namespaces": {
     "dc":"http://purl.org/dc/elements/1.1/",
     "rss":"http://purl.org/rss/1.0/",
     "georss":http://www.georss.org/georss/
  },
"@type":"rss:channel",
"rss:items": [
     { "@type":"rss:item",
        "rss:title":"A visit to Astoria",
        "rss:description":"sample description",
        "dc:coverage":{
            "@id":"a0",
            "dc:title":"Astoria, Oregon, US",
            "georss:point":"46.18806-123.83“
     }
  },
]
}
```

The '@id' keyword means 'This value is an identifier that is an IRI'

Resource-oriented Approach

JSON-LD: a W3C recommendation for representing RDF data as JSON objects

```
{"@context": {
    "name": "http://xmlns.com/foaf/0.1/name",
    "homepage": {
      "@id": "http://xmlns.com/foaf/0.1/workplaceHomepage",
      "@type": "@id"
    },
    "Person": "http://xmlns.com/foaf/0.1/Person"
  },
  "@id": "http://me.markus-lanthaler.com",
  "@type": "Person",
  "name": "Markus Lanthaler",
  "homepage": "http://www.tugraz.at/"
}
```


Define a context

```
{ "@context":
  {
    "name": "http://schema.org/name",        % [1]
    "image": {
      "@id": "http://schema.org/image",      % [2]
      "@type": "@id"                         % [3]
    },
    "homepage": {
      "@id": "http://schema.org/url",        % [4]
      "@type": "@id"                         % [5]
    } } }
```

[1] This means that 'name' is shorthand for 'http://schema.org/name'

[2] This means that 'image' is shorthand for 'http://schema.org/image'

[3] This means that a string value associated with 'image' should be interpreted as an identifier that is an IRI

[4] This means that 'homepage' is shorthand for 'http://schema.org/url'

[5] This means that a string value associated with 'homepage' should be interpreted as an identifier that is an IRI

Google looks for JSON-LD

- Google looks for and uses some JSON-LD markup (e.g., for organizations)
- Put a JSON-LD object in the head of a web page wrapped with script tags:

```
<script type="application/ld+json">
{...}
</script>
```


Ontologies

- The term ontology is originated from philosophy. In that context it is used as the name of a subfield of philosophy, namely, the study of the nature of existence.
- In the Semantic Web:
  - An ontology is a formal specification of a domain; concepts in a domain and relationships between the concepts (and some logical restrictions).

Ontologies and Semantic Web

- In general, an ontology describes formally a domain of discourse.
- An ontology consists of a finite list of terms and the relationships between the terms.
- The terms denote important concepts (classes of objects) of the domain.
- For example, in a university setting, staff members, students, courses, modules, lecture theatres, and schools are some important concepts.

Web Ontology Language (OWL)

- RDF(S) is useful to describe the concepts and their relationships, but does not solve all possible requirements
- Complex applications may want more possibilities:
  - similarity and/or differences of terms (properties or classes)
  - construct classes, not just name them
  - can a program reason about some terms? e.g.:
    - each «Sensor» resource «A» has at least one «hasLocation»
    - each «Sensor» resource «A» has maximum one ID
- This lead to the development of Web Ontology Language or OWL.

OWL

- OWL provide more concepts to express meaning and semantics than XML and RDF(S)
- OWL provides more constructs for stating logical expressions such as: Equality, Property Characteristics, Property Restrictions, Restricted Cardinality, Class Intersection, Annotation Properties, Versioning, etc.

Source: http://www.w3.org/TR/owl-features/

Ontology engineering

- An ontology: classes and properties (also referred to as schema ontology)
- Knowledge base: a set of individual instances of classes and their relationships
- Steps for developing an ontology:
  - defining classes in the ontology and arranging the classes in a taxonomic (subclass–superclass) hierarchy
  - defining properties and describing allowed values and restriction for these properties
  - Adding instances and individuals

Basic rules for designing ontologies

- There is no one correct way to model a domain; there are always possible alternatives.
  - The best solution almost always depends on the application that you have in mind and the required scope and details.
- Ontology development is an iterative process.
  - The ontologies provide a sharable and extensible form to represent a domain model.
- Concepts that you choose in an ontology should be close to physical or logical objects and relationships in your domain of interest (using meaningful nouns and verbs).

A simple methodology

1. Determine the domain and scope of the model that you want to design your ontology.
2. Consider reusing existing concepts/ontologies; this will help to increase the interoperability of your ontology.
3. Enumerate important terms in the ontology; this will determine what are the key concepts that need to be defined in an ontology.
4. Define the classes and the class hierarchy; decide on the classes and the parent/child relationships.
5. Define the properties of classes; define the properties that relate the classes.
6. Define features of the properties; if you are going to add restriction or other OWL type restrictions/logical expressions.
7. Define/add instances.

Semantic technologies in the IoT

- Applying semantic technologies to the IoT can support:
  - Interoperability
  - effective data access and integration
  - resource discovery
  - reasoning and processing of data
  - information extraction (for automated decision making and management)

Data/Service description frameworks

- There are standards such as Sensor Web Enablement (SWE) set developed by the Open Geospatial Consortium that are widely being adopted in industry, government and academia.
- While such frameworks provide some interoperability, semantic technologies are increasingly seen as key enabler for integration of IoT data and broader Web information systems.

Sensor Markup Language (SensorML)

- The Sensor Model Language Encoding (SensorML) defines models and XML encoding to represent the geometric, dynamic, and observational characteristics of sensors and sensor systems.

Using semantics

- Find all available resources (which can provide data) and data related to "*Room A*" (which is an object in the linked data)?
  - What is "Room A"? What is its location?→ returns "location" data
  - What type of data is available for "Room A" or that "location"? (*sensor types*)
- Predefined Rules can be applied based on available data
  - $(Temp_{Room\_A} > 80°C)$ AND $(SmokeDetected_{Room\_A\ position} == TRUE)$ → $FireEvent_{Room\_A}$

Semantic modelling

- Lightweight: experiences show that a lightweight ontology model that well balances expressiveness and inference complexity is more likely to be widely adopted and reused; also large number of IoT resources and huge amount of data need efficient processing
- Compatibility: an ontology needs to be consistent with those well designed, existing ontologies to ensure compatibility wherever possible.
- Modularity: modular approach to facilitate ontology evolution, extension and integration with external ontologies.

What SSN does not model

- Sensor types and models
- Networks: communication, topology
- Representation of data and units of measurement
- Location, mobility or other dynamic behaviours
- Control and actuation
- ….

WebRTC

WebRTC

- Media stack: audio & video
- Real time communication
  - Audio, video, data
- Peer-to-peer
- Accessible from browser: easy & available

Demo

http://simplewebrtc.com/demo.html?cs144r

WebRTC & IoT

- How could it be used in your project?
  - IoT connects to physical world through sensors
  - Audio/Video
  - Peer to Peer

Where Did WebRTC Come From?

- 5/2011: Google open sourced WebRTC, using audio/video streaming technology from $70M acquisition of Global IP Solutions
- ‘Make the browser the home for innovation in real time communications’
- Real-time audio video relied on proprietary technology and plugins
  - Skype (acquired by Msft for $8.5B 5/11)

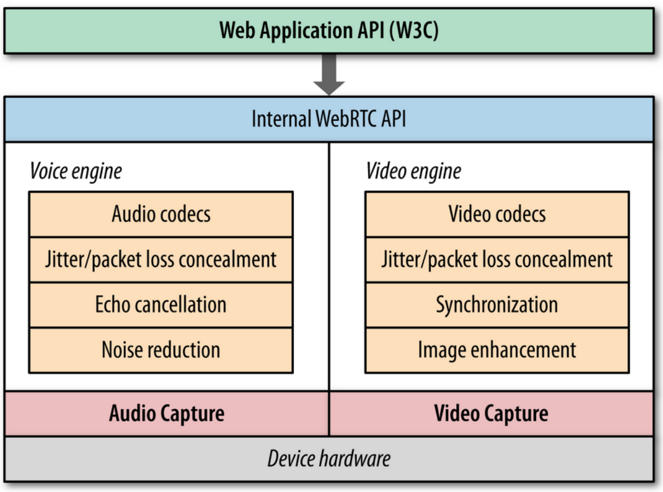
Reach

- Browser
  - Chrome, Firefox, Opera
  - Not IE, Safari
- App
  - SDK for iOS & Android native apps
  - Windows/linux/mac in JS with Node-webkit

API

- Set up peer connection
  – RTCPeerConnection
- Access local camera/audio
  – getUserMedia
- Add data channel
  – RTCDataChannel

Media Stack

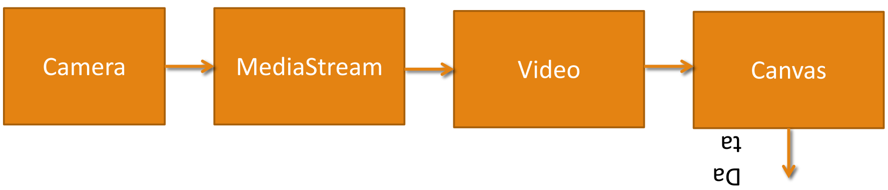


Demo: Connect Camera to Local Browser Video Display

- http://www.simpl.info/getusermedia/

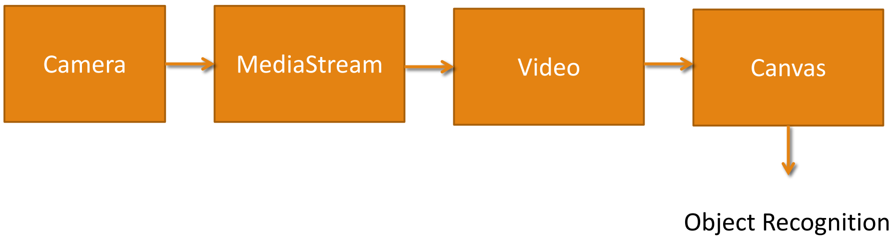
Demo: Process Camera Data

- http://idevelop.ro/ascii-camera/

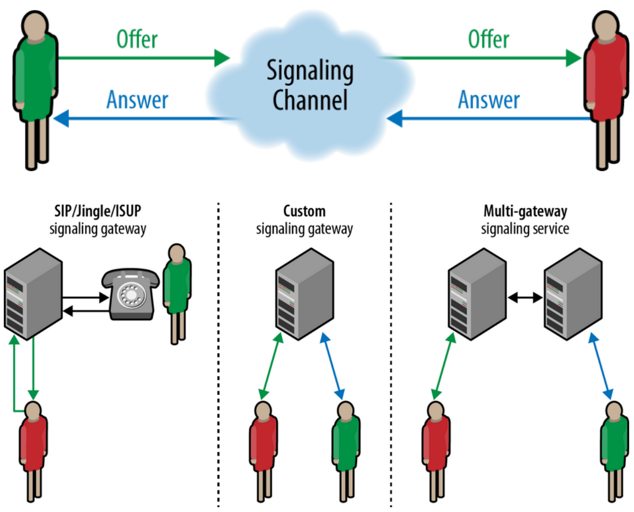


Demo: Process Camera Data2

- http://shinydemos.com/facekat/



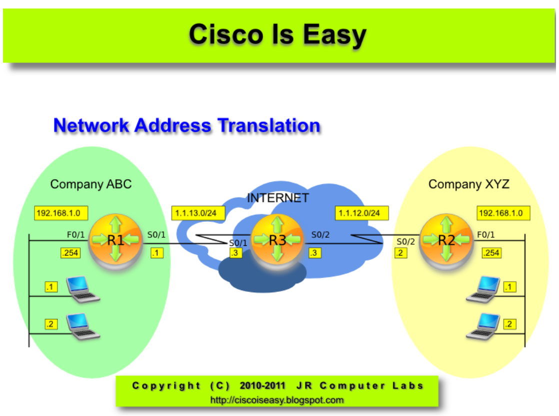
Connecting Clients



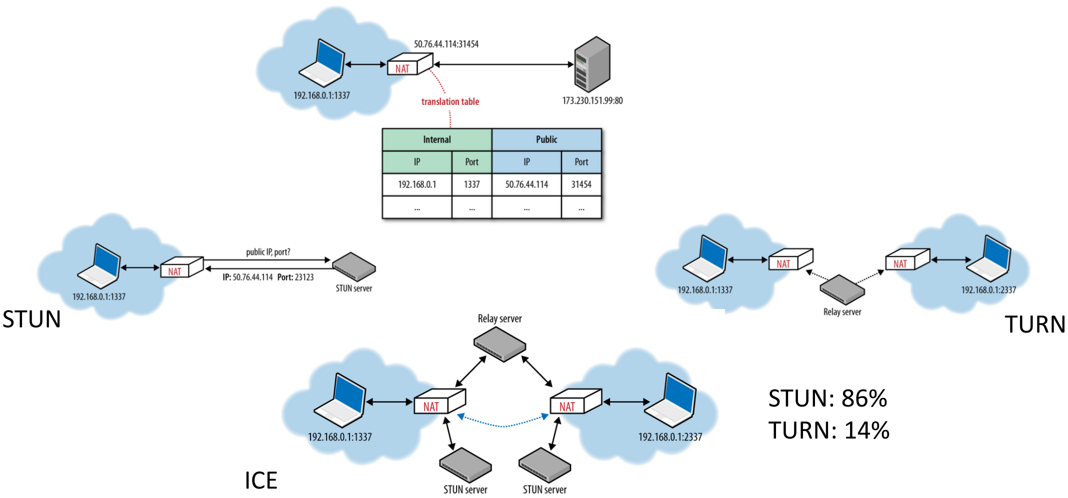
Signaling

- Session control messages: to initialize or close communication and report errors.
- Network configuration: to the outside world, what's my computer's IP address and port?
- Media capabilities: what codecs and resolutions can be handled by my browser and the browser it wants to communicate with?

Peer to Peer

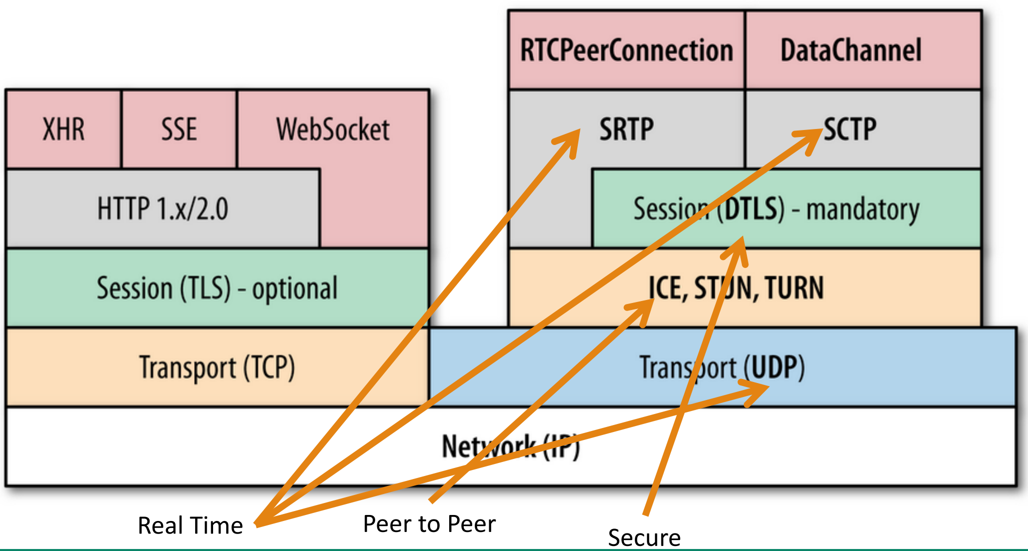


NAT: STUN/TURN/ICE



STUN: 86%
TURN: 14%

WebRTC Protocol Stack



Video Example

- http://www.simpl.info/rtcpeerconnection/index.html

Data

- Demo: http://www.simpl.info/rtcdatachannel/

Summary

- Webrtc
  - Real time, secure, peer to peer communication
  - Audio, video, data
  - Open source. Browser. iOS & Android native. Desktop

References

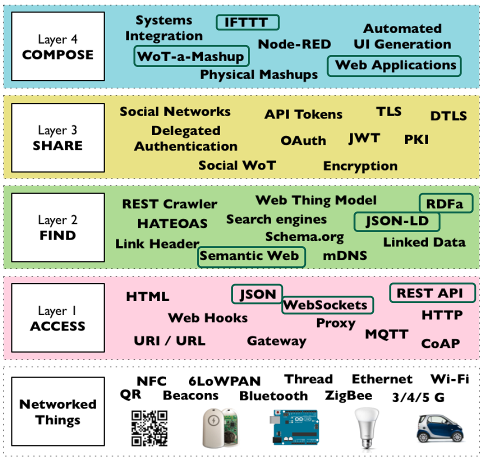
- WebRTC
  - Examples:
    - http://simpl.info/
    - http://simplewebrtc.com/
  - Overview: http://www.html5rocks.com/en/tutorials/webrtc/basics/
  - Free book: High Performance Browser Networking, Ilya Grigorik

Discussion

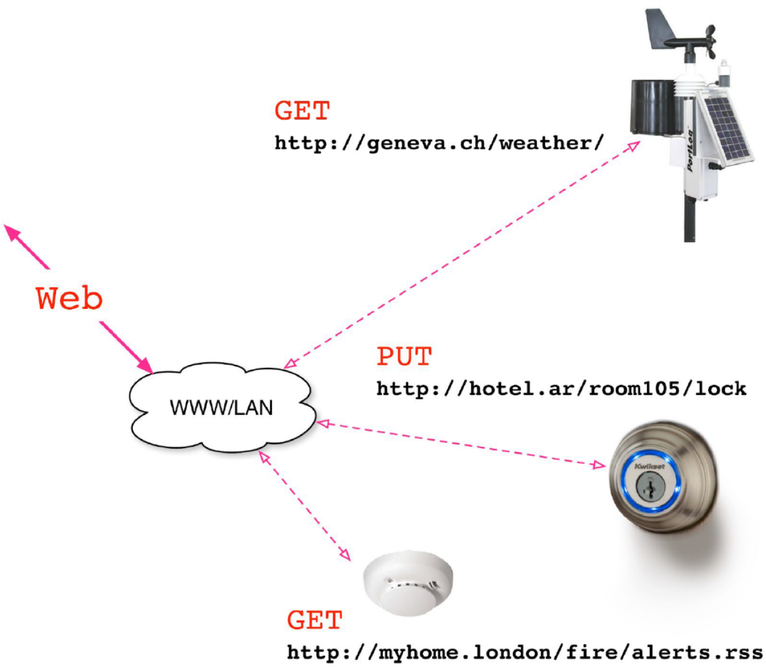
- Serendipity
- NabuBU
- SEMA
- Clothes closet
- SmartSwitch
- Miho
- Beagle-Badger
- Running App
- Headphone interrupt
- Horton
- Baby Monitor
- Traffic control
- WeighTrackr

Part III: The Web of Things Architecture

The Web of Things Architecture



Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

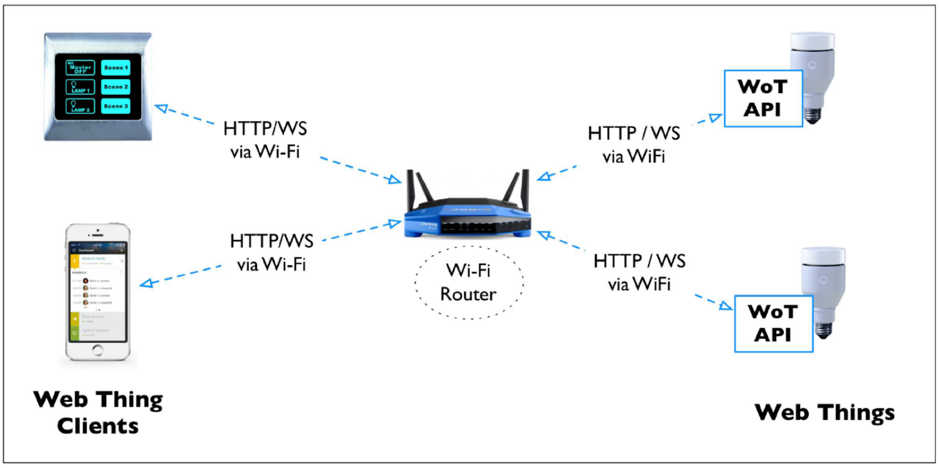


- Web Developers
- Native/Desktop Apps
- Web services & APIs
- Analytics, storage

Web API for Things: 5 Steps Design Process

- Integration strategy
  - Choose a pattern to integrate Things to the internet and the web.
- Resource design
  - Identify the functionality or services of a Thing, and organize the hierarchy of these services.
- Representation design
  - Decide which representations will be served for each resource.
- Interface design
  - Decide which commands are possible for each service, along with which error codes.
- Resource linking design
  - Decide how the different resources are linked to each other.

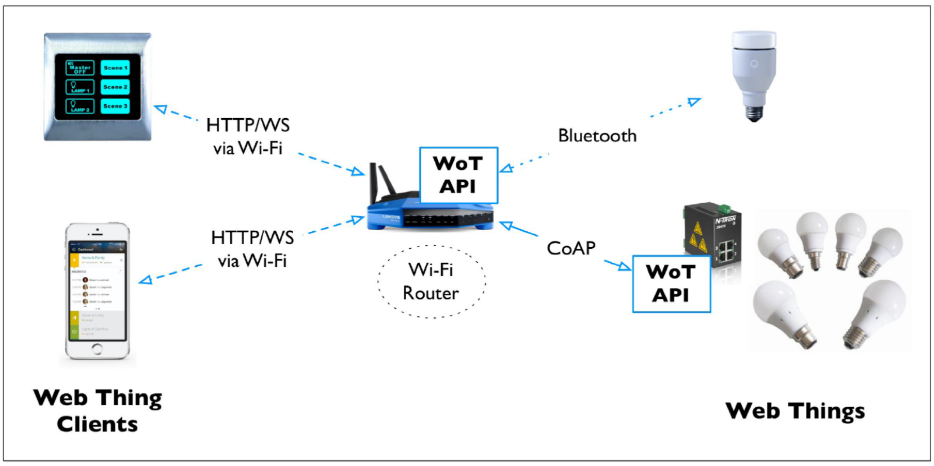
Integration Strategy



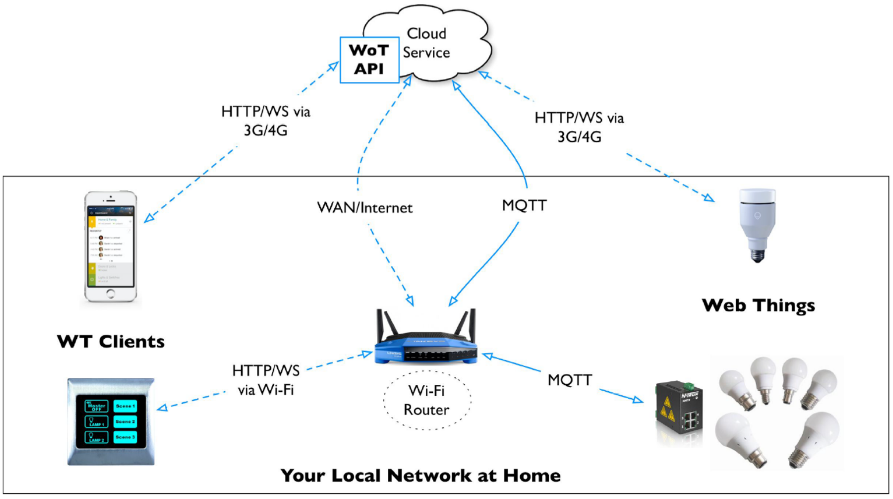
But: Not all devices can speak Web!

	Typical MQTT Protocol Stack	Typical MQTT-SN Protocol Stack	Typical CoAP Protocol Stack	
4. Application	MQTT	MQTT-SN	CoAP CoRE	Required
3. Transport	TCP	UDP	UDP	
2. Network (Internet)	IP	Not specified	6LoWPAN	Recommended
1. Physical (Link)	Not specified	Not specified	IEEE 802.15.4	
	MQTT.js		node-coap	

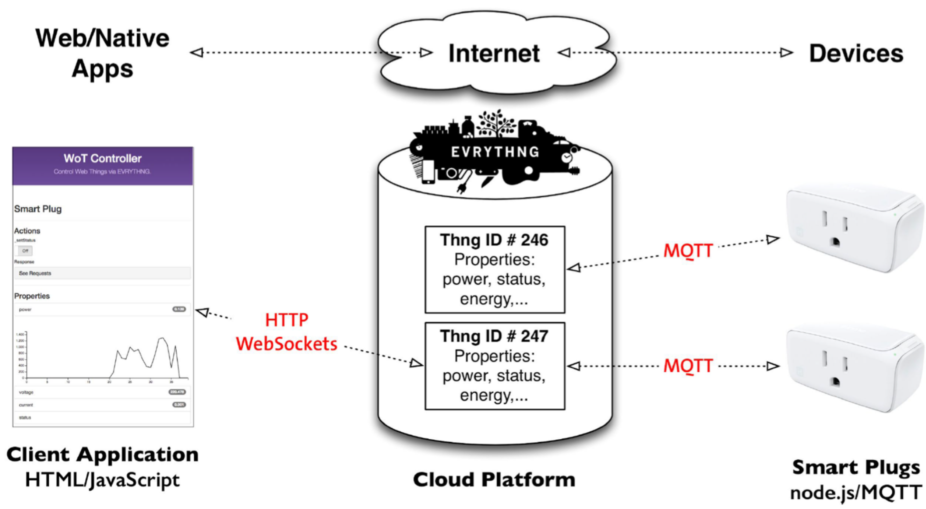
Integration via Gateway



The Cloud as a Gateway



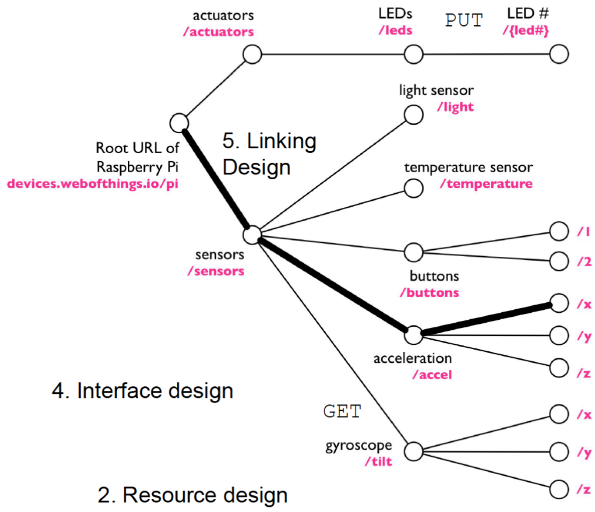
Example: EVRYTHNG [cloud solution] Smart Products Platform



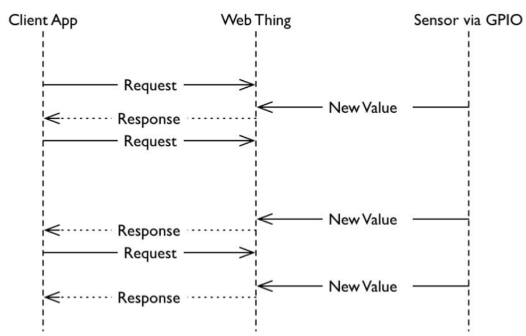
Resources, Representations and Links

3. Representation design

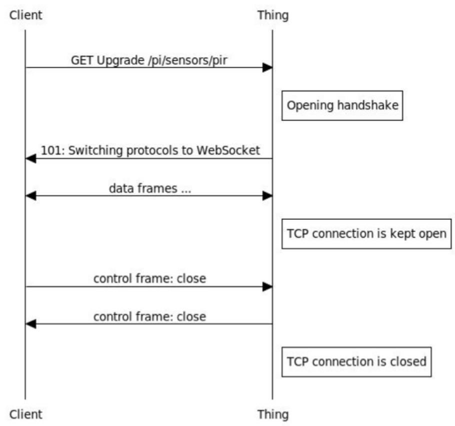
```
Request:
GET /pi
Host:
  devices.webofthings.io
Accept: application/json
Response:
200 OK
Content-Type:
  application/json
{
   "name" : "Pi"
   ...
}
```



Beyond HTTP: Websockets for Event Driven Communication



HTTP 1.1

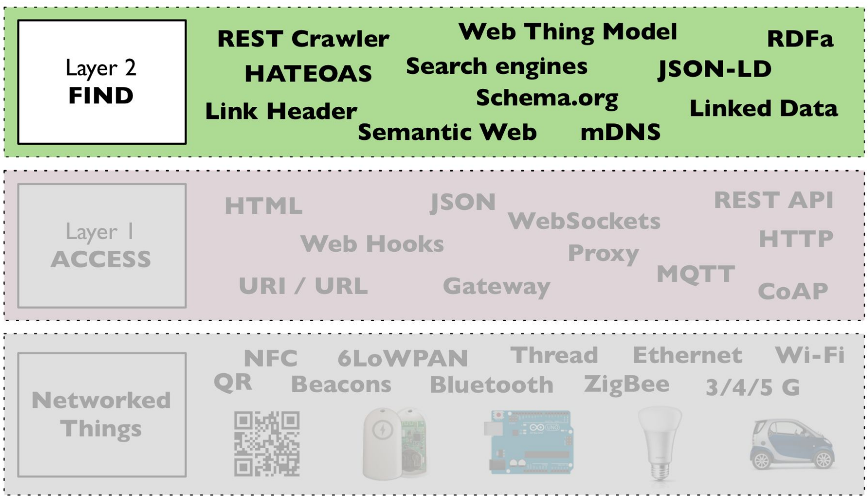


WebSockets

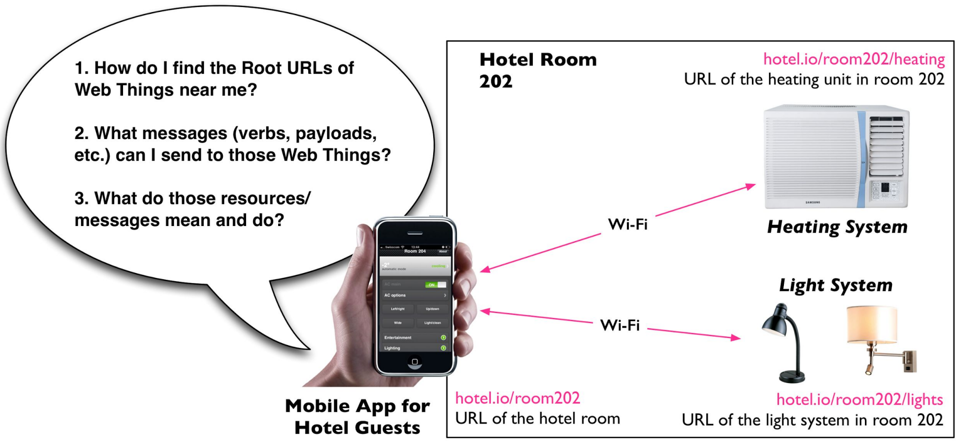
WebSocket Client

```
function subscribeToWs(url, msg) {
    var socket = new WebSocket(url);
    socket.onmessage = function (event) {
        console.log(event.data);
    };
    socket.onerror = function (error) {
        console.log('An error occurred while trying to connect to a Websocket!’);
        console.log(error);
    };
    socket.onopen = function (event) {
        if (msg) {
            socket.send(msg);
        }
    };
}
//subscribeToWs('ws://localhost:8484/pi/sensors/temperature');
```


WoT Architecture: Find

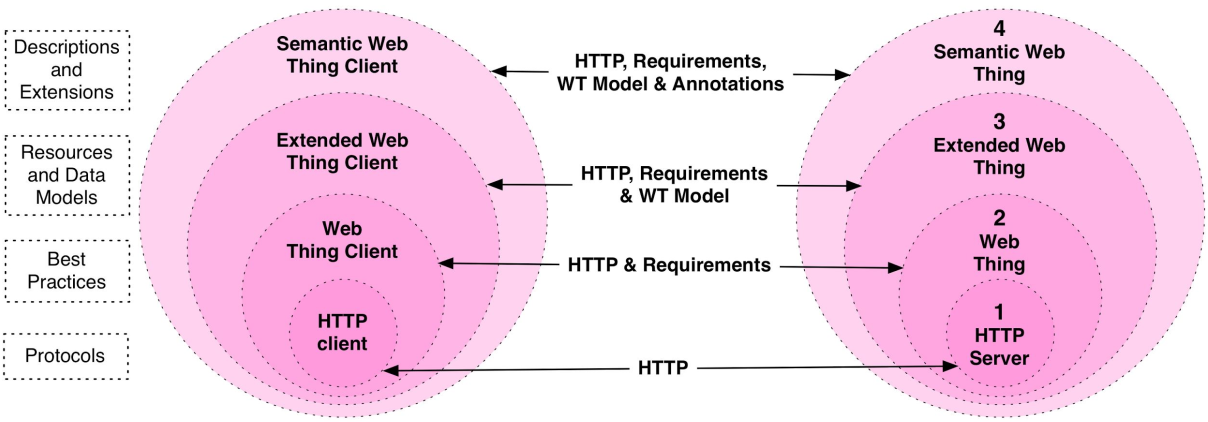


Three Challenges in IoT Findability



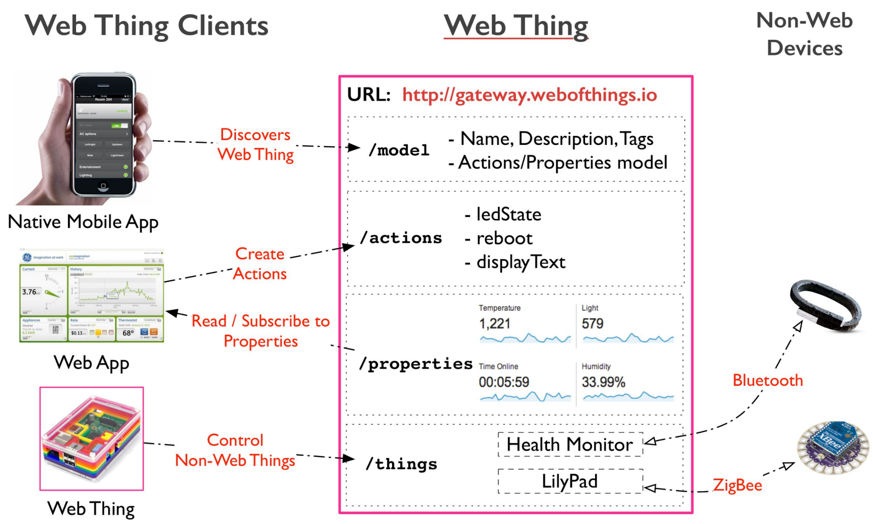
Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

Web Thing Model & Semantic Web



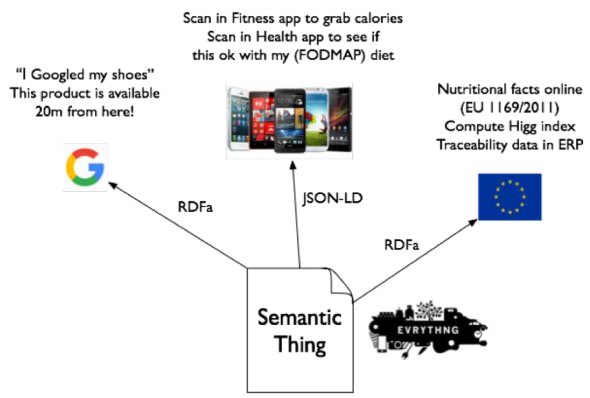
Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

Web Thing Model Resources

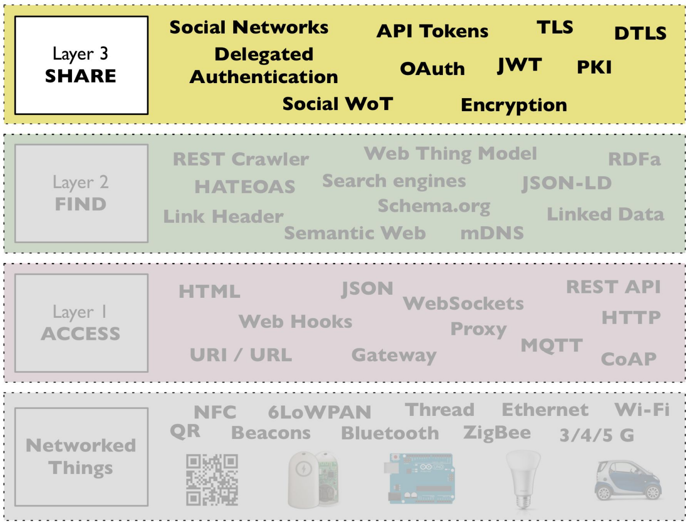


Say Hi to the Semantic Web (of Things!)

- Semantic extensions [via JSON-LD]
  - Enhance semantics: What is that Thing really?
  - Schema.org
- Fosters:
  - Findability
  - Interoperability
  - Compliance
- More details:
  - http://model.webofthings.io



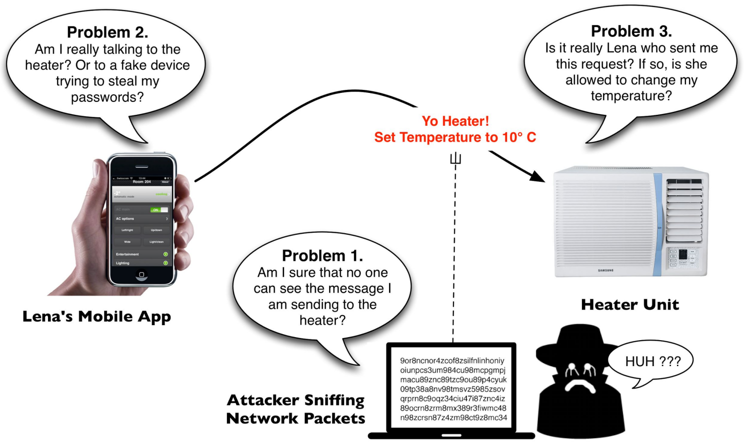
WoT Architecture: Share and Secure



Securing Things

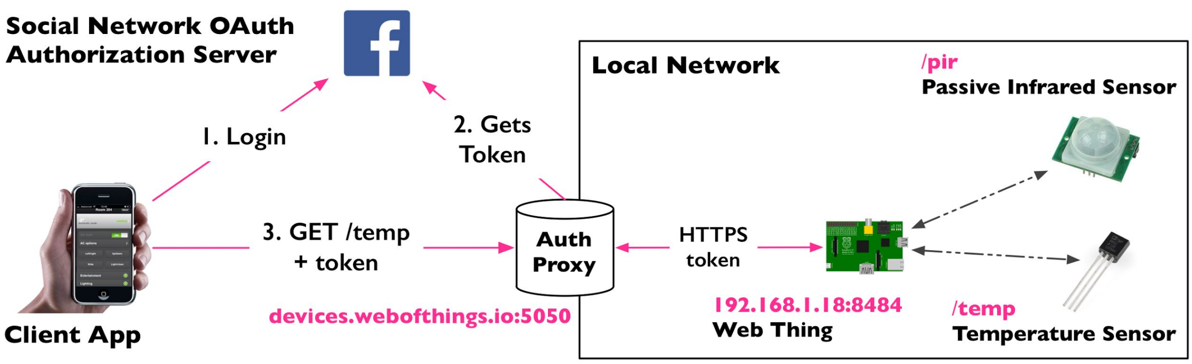
- The most dangerous thing about Web Things is to bring them to the Web!
- Problem 1:
  - Web Encryption
- Problem 2:
  - TLS (SSL) certificates
- Problem 3:
  - API keys (oAuth)
  - Authorization header
  - Token query param

Securing Things (over simplified)



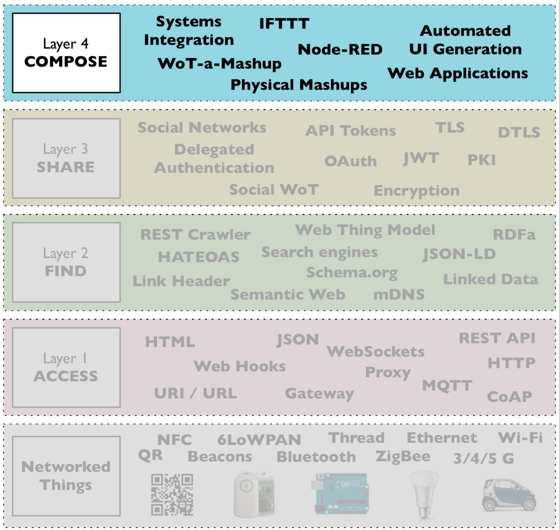
Source: Building the Web of Things: book.webofthings.io

Sharing Things: Social Web of Things



Source: Building the Web of Things: book.webofthings.io

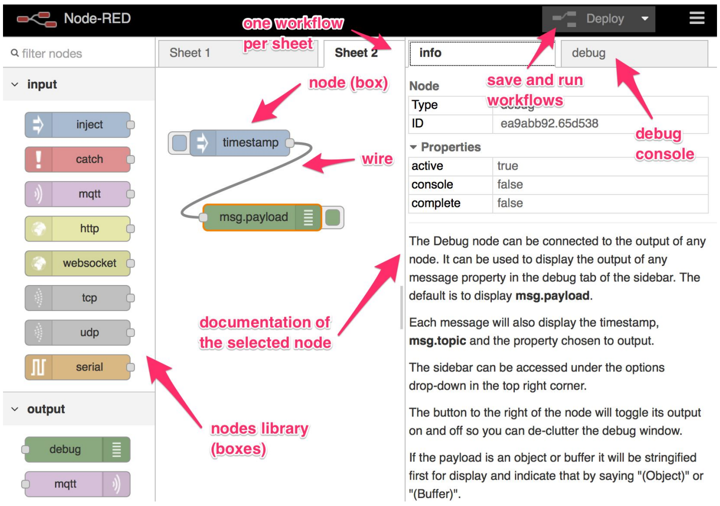
WoT Architecture: Compose



Node-RED

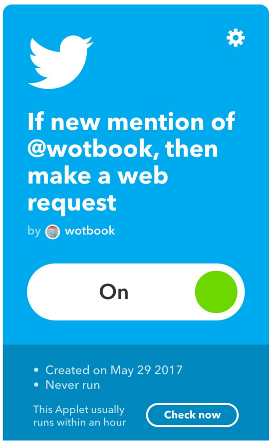
- Mashup tool for makers
- Box and wires
- Wire your prototypes
- Large community support
  - Nodes
  - E.g., https://flows.nodered.org/node/node-red-contrib-evrythng

Node-RED



IFTTT: Solid Mashups for the Masses

- https://ifttt.com/recipes
- IOS notifications
- Text
- Mail
- Facebook
- Lights



Slide References

http://book.webofthings.io