

IoT Software: From Things to Clouds

CSE291A WI16

Yeseong Kim



System Energy Efficiency Lab

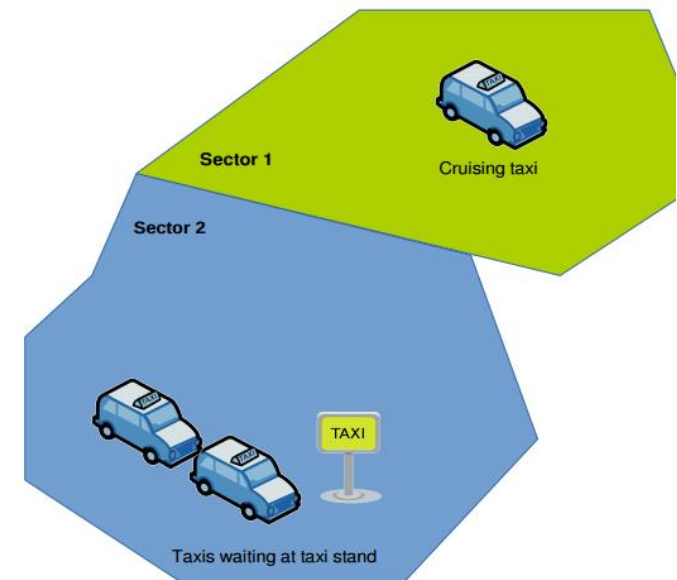
seelab.ucsd.edu

Things, Data, Action and Software

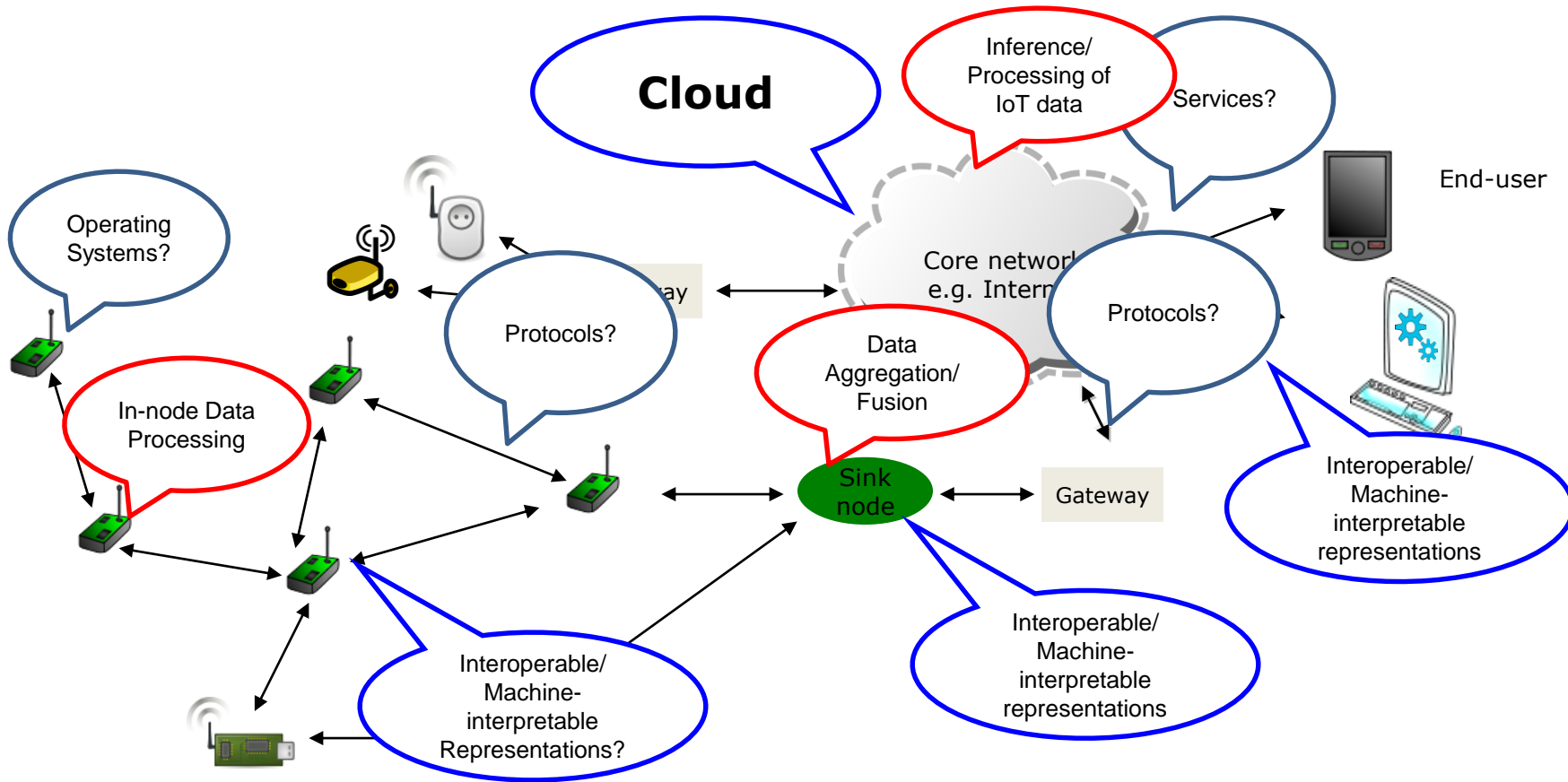
- Data is collected by sensor devices.
 - Motion, Pressure, Temperature, Light sensors
 - Cameras, Microphones, GPS enabled devices
- Why we need data? To take an action
- What would the problem be if a sensor processes information locally?
 - Constrained battery, processor and storage
 - Noisy and the varying quality -> inaccurate
 - Need more useful data from other things



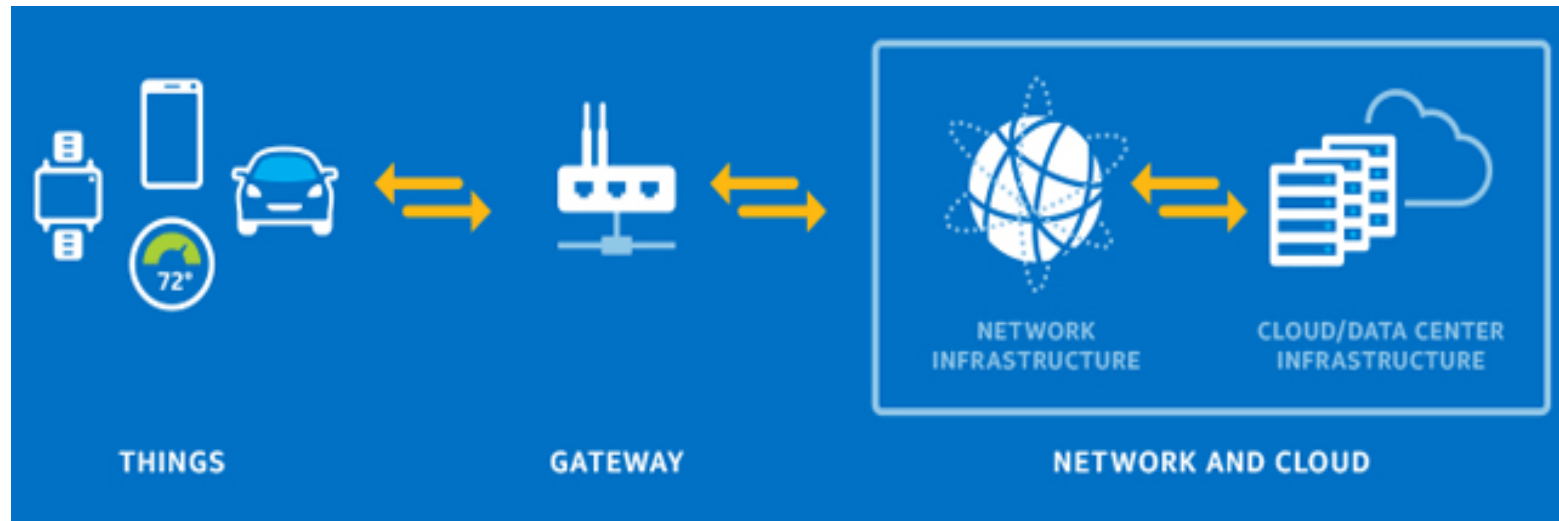
Need to fill the gap with software



Where's IoT Software?



IoT Architecture and Software



We will discuss IoT software platforms for three building blocks + actuation

1. Software platform for things
2. Semantic data delivery across gateway
3. Data processing for network infrastructures & clouds
4. Actuation to things

Software Platform for Things

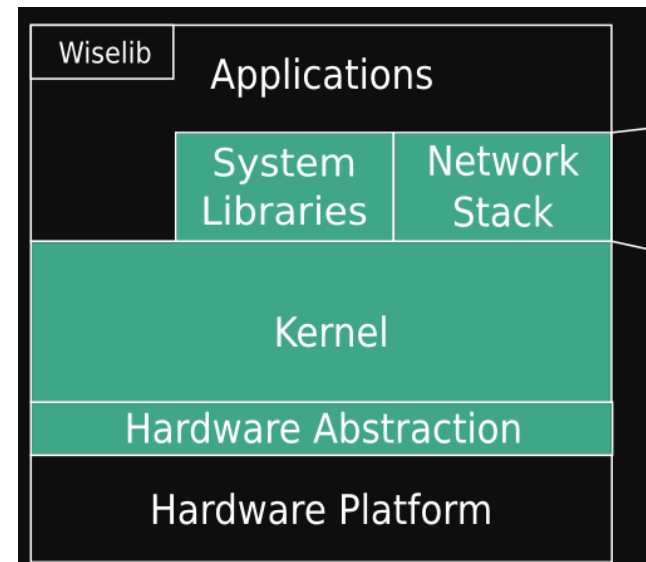
- What are the requirements for OS running on things?
 - Energy efficient
 - Handle large sleep time automatically
 - Hardware constraints
 - Complexity of operations must be kept very low
 - Limited Features (wo/ MMU, FPU)
 - Minimal memory footprints
 - Real time functionality
 - Connectivity
 - Adaptive network stack
 - Programmability
 - Standard API
 - Standard programming language

Example of OS for Things: RIOT



- Resource friendly
 - Energy efficient
 - Real-time capability due to small interrupt latency (~50 clock cycles)
 - Multi-threading (<25 bytes per thread)
- IoT friendly
 - Diverse network protocol
 - 6LoWPAN, IPv6, RPL, and UDP
 - Static and dynamic memory allocation
 - High resolution and long-term timers
- Developer friendly
 - Standard programming in C or C+
 - Standard tools such as gcc, gdb, valgrind
 - Code once, run on 8 to 32-bit platforms (e.g. Arduino Mega 2560 to ARM)
 - Partial POSIX compliance

<http://www.riot-os.org/>

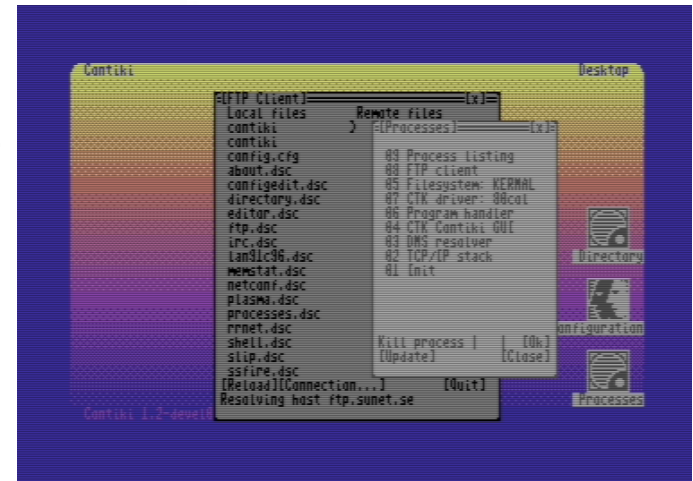


Example of OS for Things: Contiki



<http://www.contiki-os.org/>

- Selection of hardware
 - Low-power wireless devices
 - Runs for years on a pair of AA batteries
- Internet standards
 - Fully standard IPv6 and IPv4
 - Low-power wireless standards (6lowpan, CoAP)
- Rapid development
 - Written in standard C
 - Cooja network simulator
- Memory allocation
 - A few kilobytes of memory
 - Mechanisms for memory allocation



OS	Min RAM	Min ROM	C Support	C++ Support	Multi-Threading	Real-Time
Contiki	<2kB	<30kB	○	✗	○	○
Tiny OS	<1kB	<4kB	✗	✗	○	✗
Linux	~1MB	~1MB	✓	✓	✓	○
RIOT	~1.5kB	~5kB	✓	✓	✓	✓

How to Interpret the Raw Data?

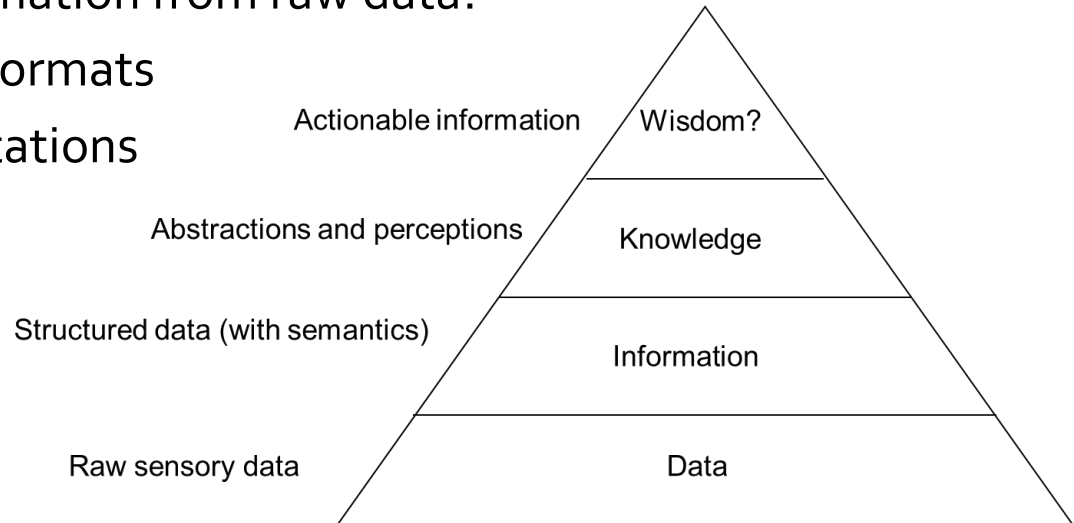
“Raw data is both an oxymoron and bad data”, Geoff Bowker, 2005

- Is the generated raw data meaningful to all users?
15, C, 08:15, 51.243057, -0.589444
 - Generated data is meaningful to few users only
 - Only sensor itself and its deployer knows
- Is the generated raw data machine-understandable?
 - Celsius vs. kelvin

Data to Information

What is required to get information from raw data?

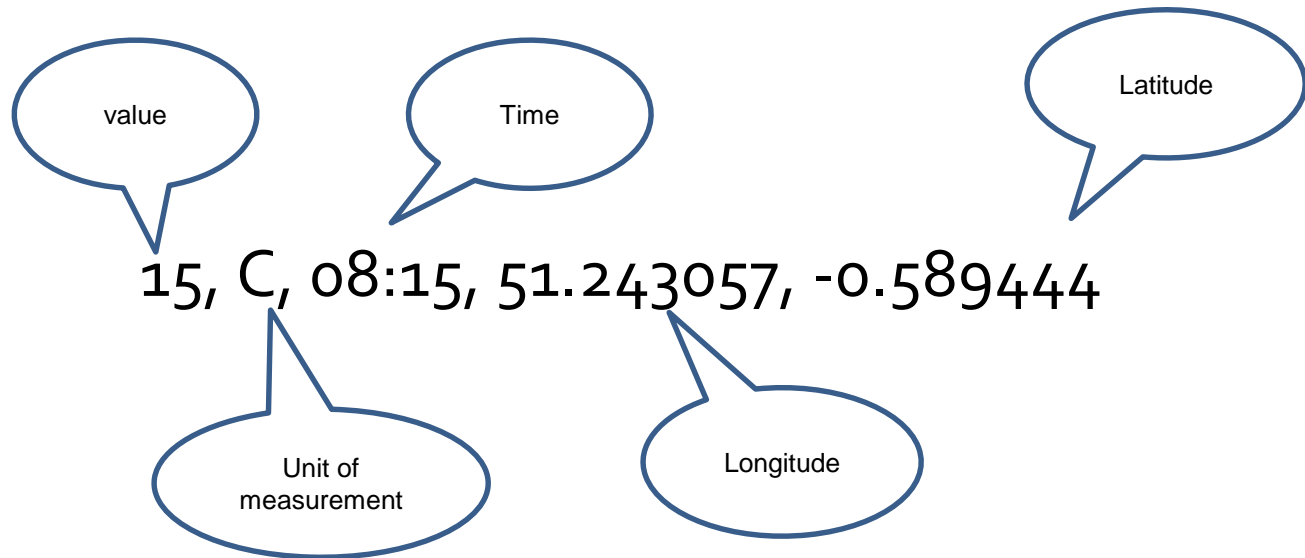
- Interpretable/structured formats
- Data with semantic annotations
- Background knowledge, domain information



Interoperable and semantically described data is the starting point :

- To make information understandable by software
- To create an efficient set of actions.

Data in Reality



- What happens if we have "08:15, 15, C, 51.243057, -0.589444" ?
- How about this?

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

Extensible Markup Language (XML)

- One of the most widely-used formats for sharing structured information.
 - Simple
 - Flexible for data representation and annotation.
 - Can exchange a wide variety of data

```
<?xml version="1.0"?>
```

XML Prolog- the XML declaration

```
<measurement>
```

Root element

```
<value>15</value>
```

```
<unit>C</unit>
```

XML elements

```
<time>08:15</time>
```

```
<longitude>51.243057</longitude>
```

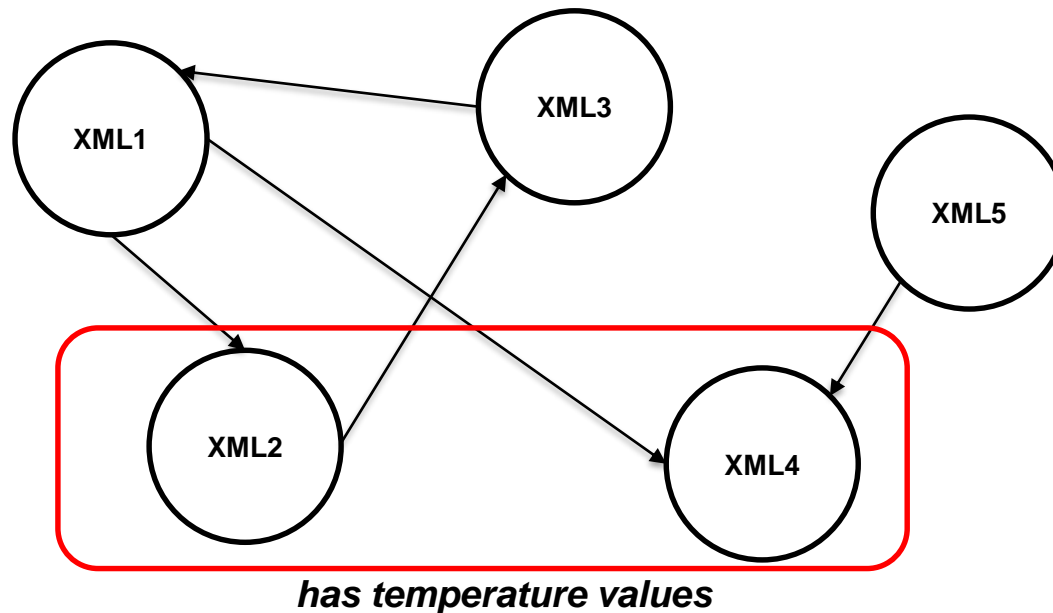
```
<latitude>-0.58944</latitude>
```

```
</measurement>
```


XML documents
MUST be "well
formed"

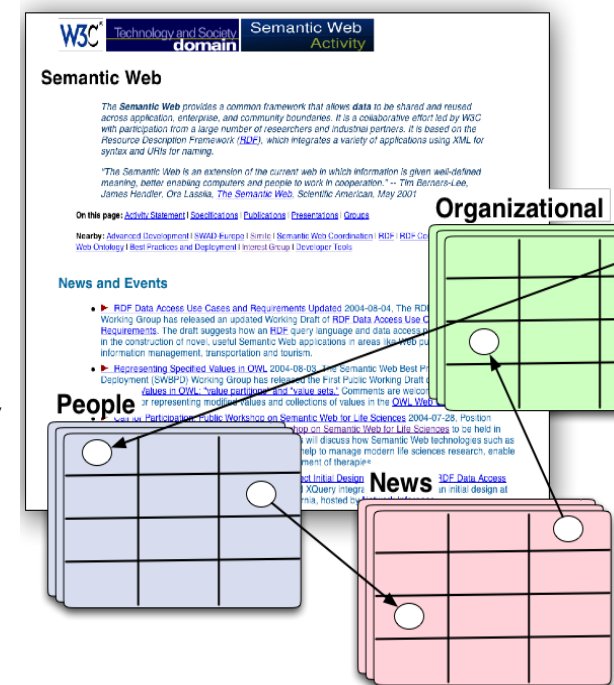
Combining Different Data Sources

- Now we have well-structured data with semantic annotations.
- What do we need to combine this data?
 - e.g., “Two pieces of data include temperature values”
 - e.g., “Taxi is a subclass of vehicle”



Semantic Web and IoT

- Semantic Web: an extension of the current web
 - Well-defined meaning
 - Better cooperation for objects, devices and people
- RDF (Resource Description Framework)
 - W3C standard 
 - Relationships between documents
 - Flexible: data relationships can be explored
 - Efficient: large scale, data can be read more quickly
- Consisting of **Triples**:
 - <subject> <property> <object> .
e.g.,
<"Sensor01"> <hasType> <"Temperature"> .
<"Taxi01"> <subClassOf> <"Vehicle" > .



The image shows a screenshot of the Semantic Web website. The page header includes the W3C logo and the text "Technology and Society domain Semantic Web Activity". The main heading is "Semantic Web". Below this, there is a paragraph explaining the Semantic Web as a common framework for sharing and reusing data. A quote from James Hendler and Ora Lassika is included. The page is divided into sections: "On this page" (with links for Activity Statement, Specifications, Publications, Presentations, Groups), "Nearby" (with links for Advanced Development, SWAD Europe, Similar, Semantic Web Coordination, RDF, RDF Co-Web Ontology, Best Practices and Deployment, Interest Group, Developer Tools), "News and Events", "People", and "News". Overlaid on the screenshot are three organizational charts: a green one labeled "Organizational", a blue one labeled "People", and a pink one labeled "News". Arrows point from these charts to specific content on the page, such as the "RDF Data Access Use Cases and Requirements" update under "News and Events" and the "Public Workshop on Semantic Web for Life Sciences" under "People".

Triples and Statements

- Each data is combined to make simple **statements** in the form of triples.
- So, triples are sometimes called "statements":
 - <"Sensor01"> <hasType> <"Temperature"> .
 - Sensor 01 **has the type of** Temperature.
 - <"Taxio1"> <subClassOf> <"Vehicle" > .
 - Taxio1 **is the sub class of** Vehicle.
- Based on the triples, a software can know how to interpret the given data set.

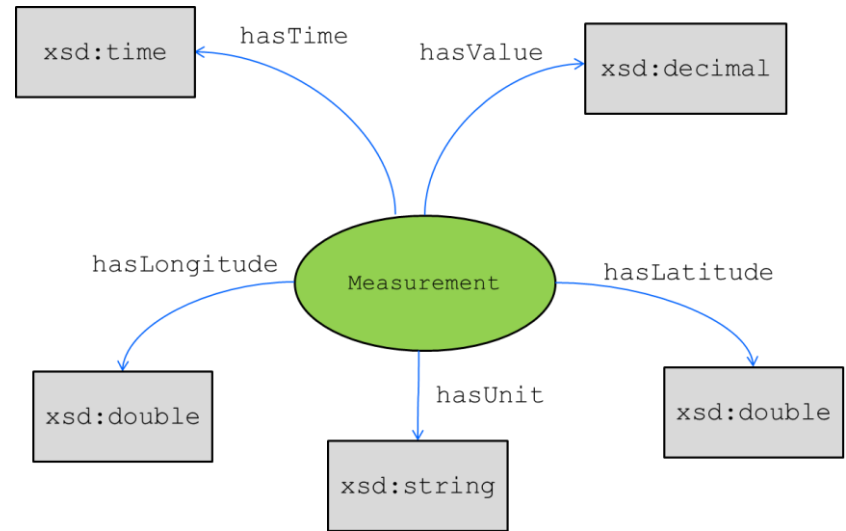
RDF with Graph

- RDF triples form a graph
 - Possible to know which properties belong to the subject (instance)

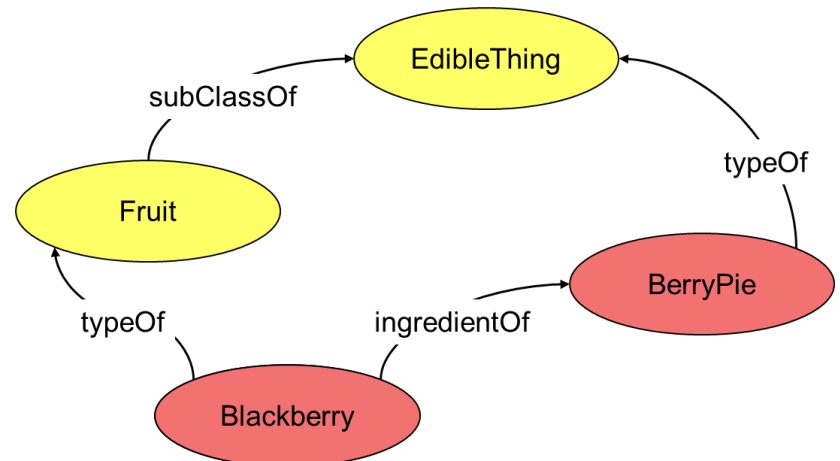
```

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

```



- RDF also represents the relationship between different objects.
 - typeOf
 - subClassOf
 - ...



RDF Vocabulary

- **Vocabulary**
 - Sets of terms used to describe things
 - A data model including classes, properties and relationships
 - A term is either a class or a property
- Why do we use existing RDF vocabularies?
 - Easier and cheaper
 - Interoperability of your data
 - Adds credibility to your schema
- Many vocabularies are available:
 - IOT sensors: <https://github.com/dpjanes/iotdb-vocabulary>
 - FOAF(People description): <http://www.foaf-project.org/>
 - Core Location: https://joinup.ec.europa.eu/asset/core_location/description

Database for RDF

- **SPARQL: SPARQL Protocol and RDF Query Language**
 - Standard language to query graph data represented as RDF triples.
 - One of the three core standards of the Semantic Web, along with RDF and OWL.
 - Became a W3C standard January 2008.

Sample data

```
comp:A rov:haslegalName "Niké" .  
comp:A org:hasRegisteredSite site:1234 .
```

```
Comp:B rov:haslegalName "BARCO" .
```

```
site:1234 locn:fullAddress "Dahliastraat 24, 2160 Wommelgem .
```

Query

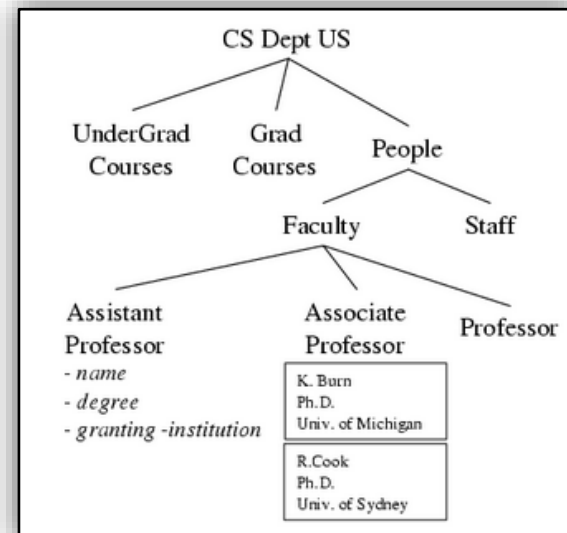
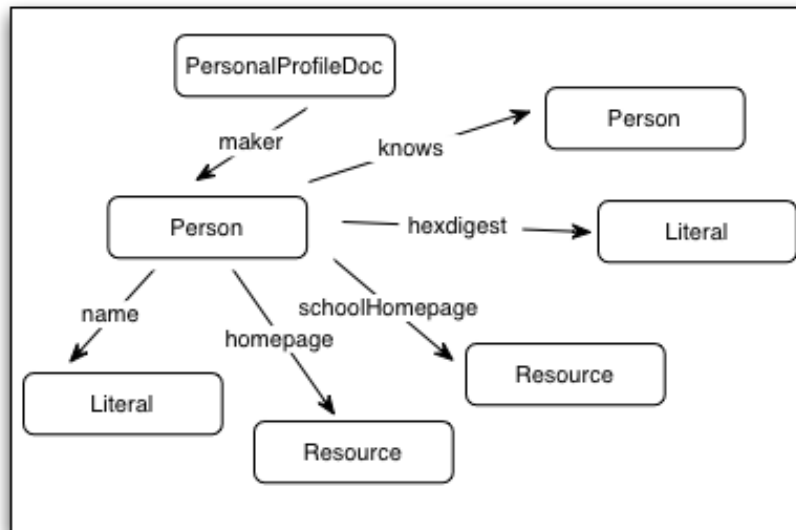
```
SELECT ?name
```

```
WHERE
```

```
{ ?x org:hasRegisteredSite site:1234 .  
  ?x rov:haslegalName ?name . }
```

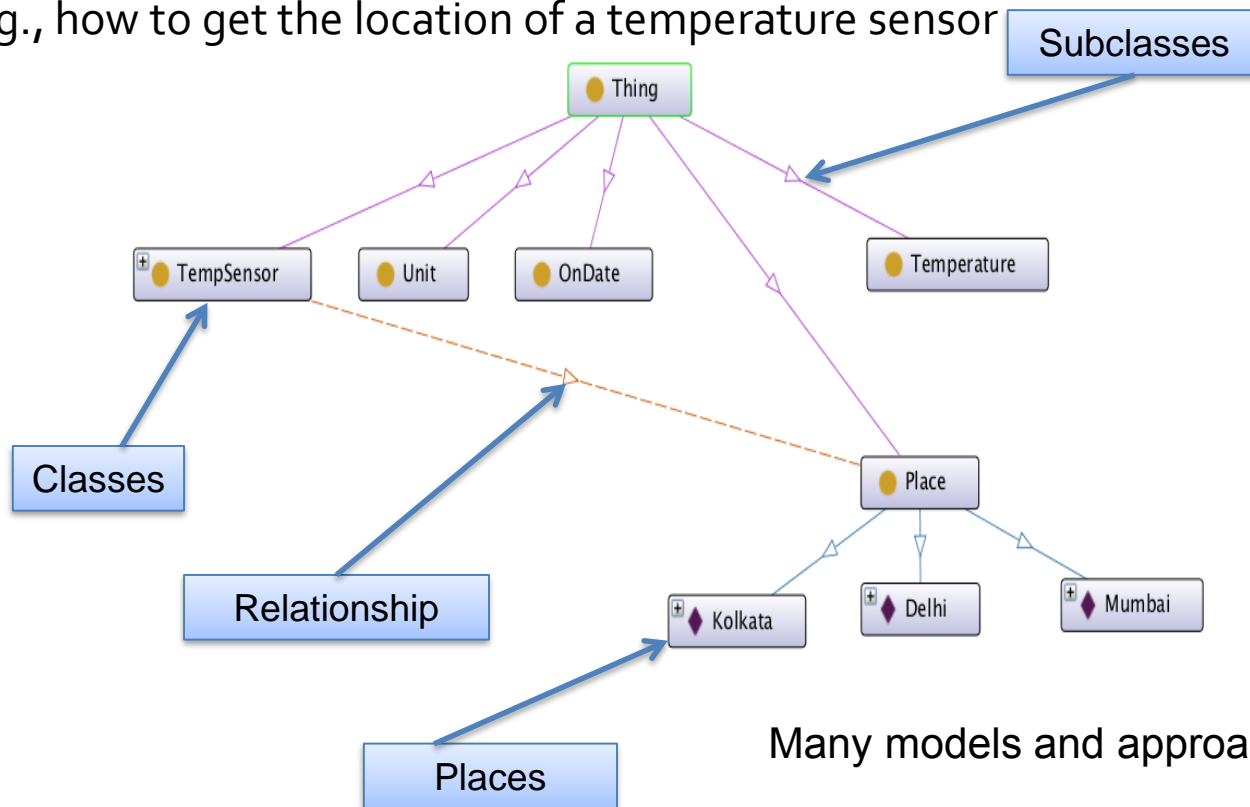
Ontology

- **Ontology**: A formal data model that represents **knowledge** as
 - A set of concepts within a **domain**
 - The **relationship** between these concepts
- Originated from philosophy, the study of the nature of existence
- **Triple** is widely used here too.
 - It is be used to support reasoning about concepts.



Ontology and IoT

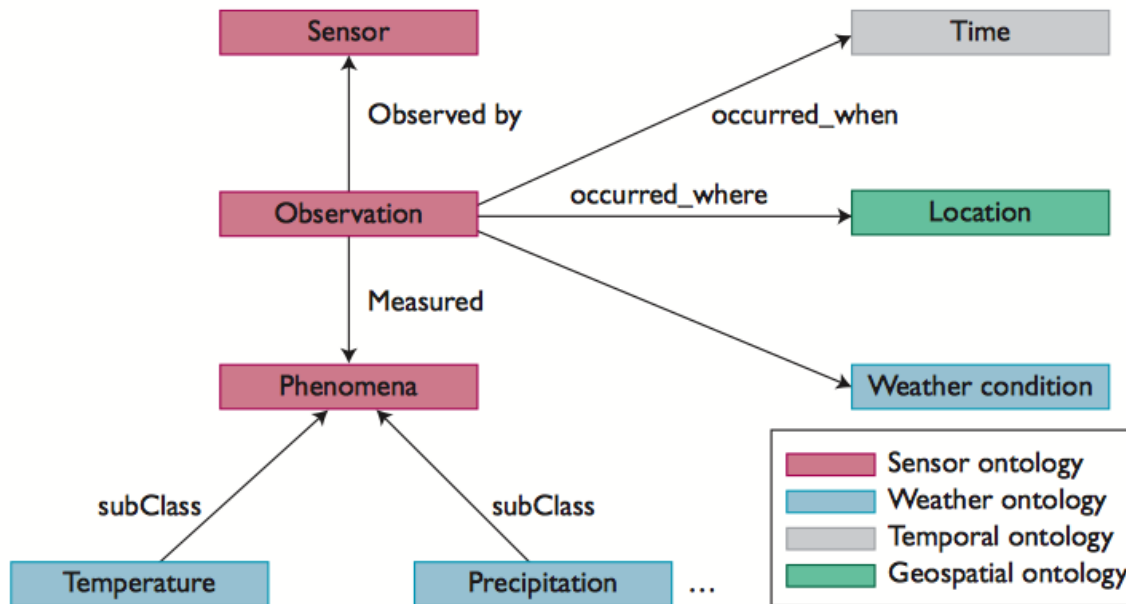
- Inferring context from multiple data sources
- Enable reuse of domain knowledge
- Allow us to infer extra knowledge from basic facts encoded
 - e.g., how to get the location of a temperature sensor



Many models and approaches exist

Semantic Sensor Web

- “The semantic sensor Web enables interoperability and advanced analytics for situation awareness and other advanced applications from heterogeneous sensors.”
(Amit Sheth et al, 2008)



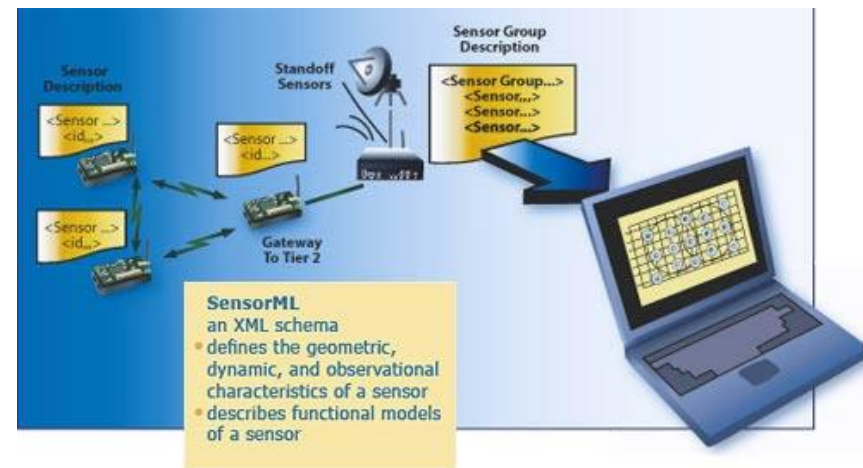
- Describes domains
 - Sensor, Weather, Temporal, Geospatial
- Describes the relationships between different ontologies as a list of terms
 - Observed by
 - Measured
 - Occurred when

Web Ontology Language (OWL)

- Purpose: to develop ontologies that are compatible with the WWW.
- RDF is useful to describe the concepts and their relationships, but does not solve all possible requirements.
 - similarity and/or differences of terms (properties or classes)
 - can a program reason about some terms
 - each «Sensor» resource «A» has at least one «hasLocation»
 - each «Sensor» resource «A» has maximum one ID
- Based on the basic elements of RDF, OWL adds more vocabulary for describing properties and classes.
 - Relationships between classes (to specify domains)
 - Equality
 - Richer properties
 - Class property restrictions

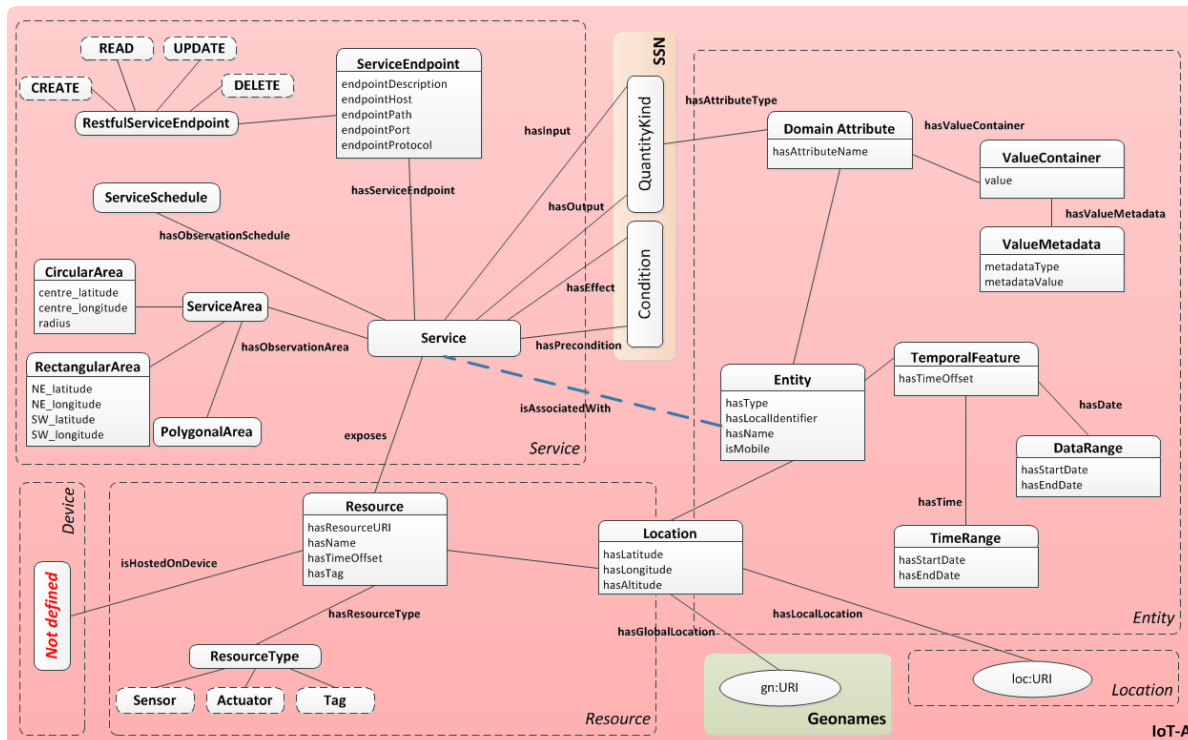
Other Approaches for Ontology

- SKOS: Simple Knowledge Organization System
 - Based on RDF
 - Designed specifically to express information that's more hierarchical
 - Limited expressivity compared to OWL to avoid expressiveness not desired in some application
- SensorML: Sensor Model Language Encoding
 - Specialized for sensors (supports standard dictionary)
 - Represents geometric, dynamic, and observational characteristics of sensors and sensing systems
 - Based on XML



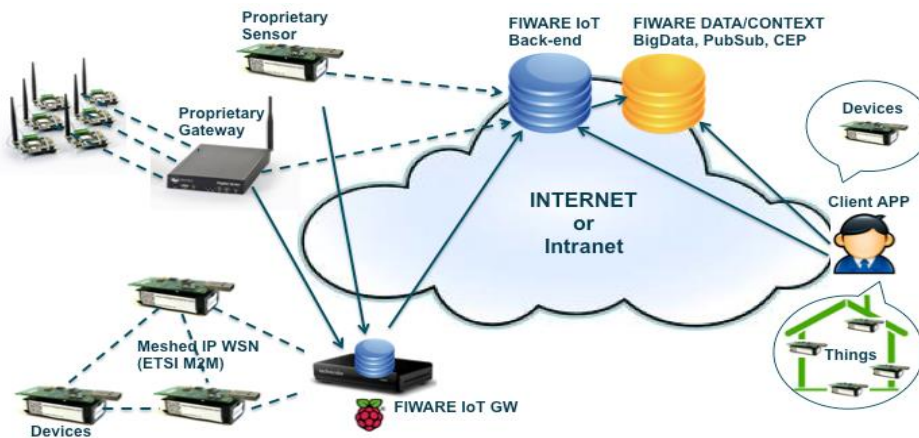
Ontology Models: IoT-A Information Model

- IoT-A: Internet of Things Architecture
 - European Lighthouse Integrated Project
 - Partner: HITACHI, NXPO, SIEMENS, SAP, IBM, ...
 - Propose architecture reference models for protocol, interface, and algorithm
 - Information Model: Ontology-based model



Use of Ontology: FIWARE IoT Discovery

- Provide Open API to register sensors and discover information
 - Work with RDF/OWL descriptions for the "Things"
 - Support querying via SPASQL
 - Possible to be deployed with Raspberry Pi

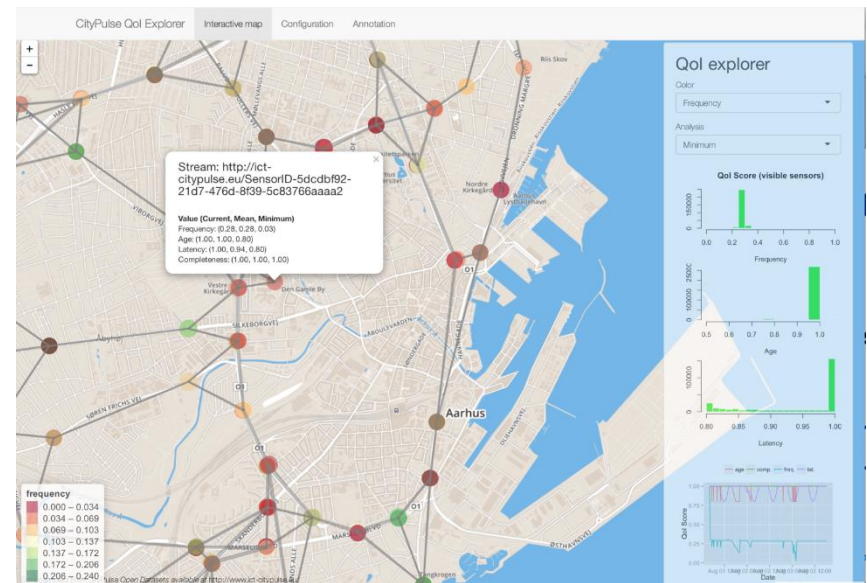
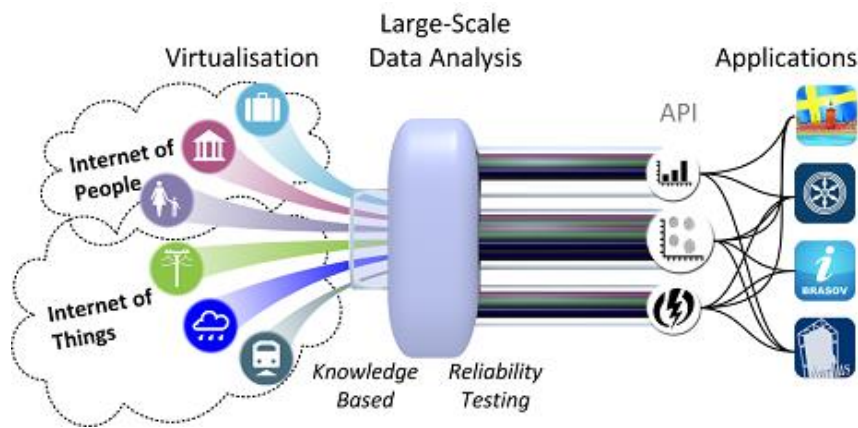


28



Use of Ontology: City Pulse

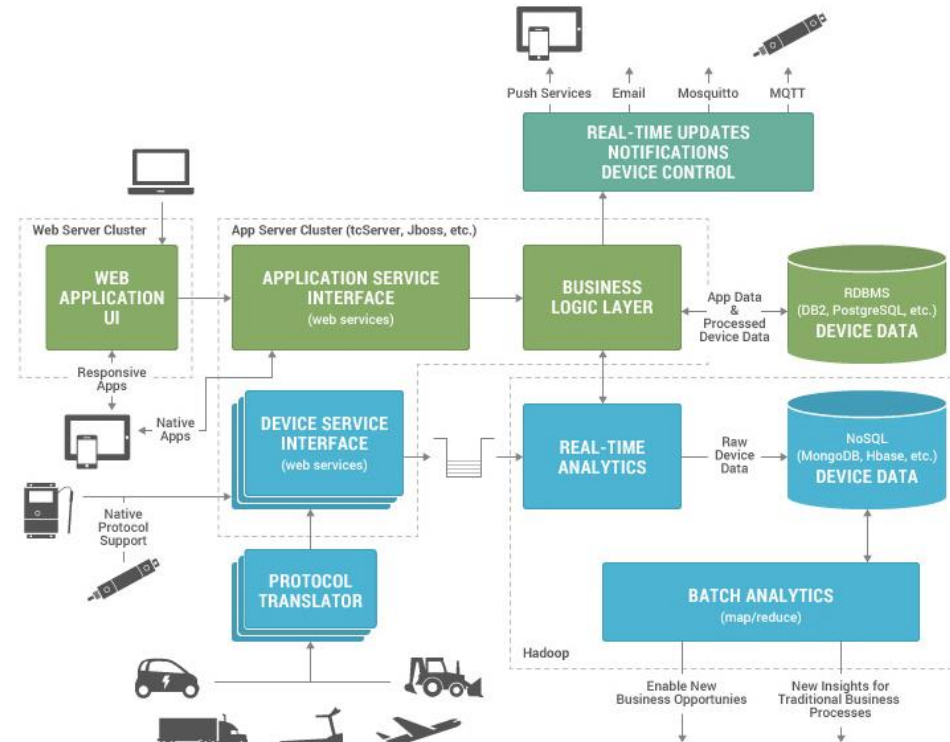
- Smart city applications for IoT
 - Developing a framework for the semantic discovery and processing of large-scale IoT data defined by ontology
 - Providing collected data
 - Road Traffic Data, Pollution Data, Weather data, Parking Data, ...



Software in Clouds

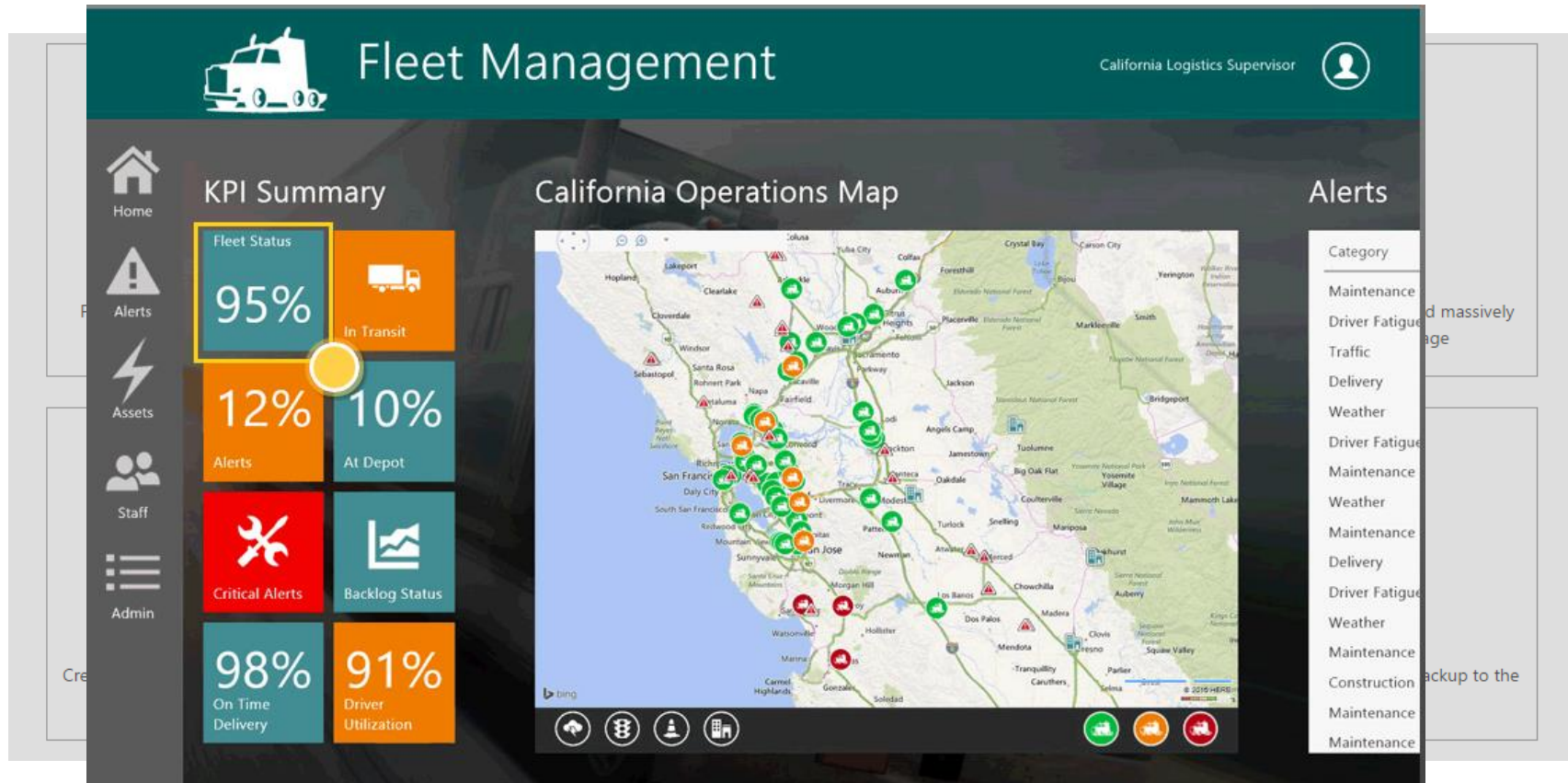
Main requirements of Cloud OS

- Quality of service assurance
 - Isolation of applications
 - Efficient resource management
- Cloud infrastructure scalability
 - Scalable network, computing and storage capacity
- Reliability
 - Fault tolerance Infrastructures
- Security and privacy
 - Secure multi-tenant environments
 - Data integrity mechanism for storage
- Energy efficient cloud management
 - Energy efficiency models, metrics and tools at datacenter levels
- Interoperability and portability
 - Common and standard interfaces for cloud computing



Microsoft Azure

- Different programming languages, tools and frameworks, including both Microsoft-specific and third-party software and systems
- Developing IoT-specific cloud applications (IoT Hub)

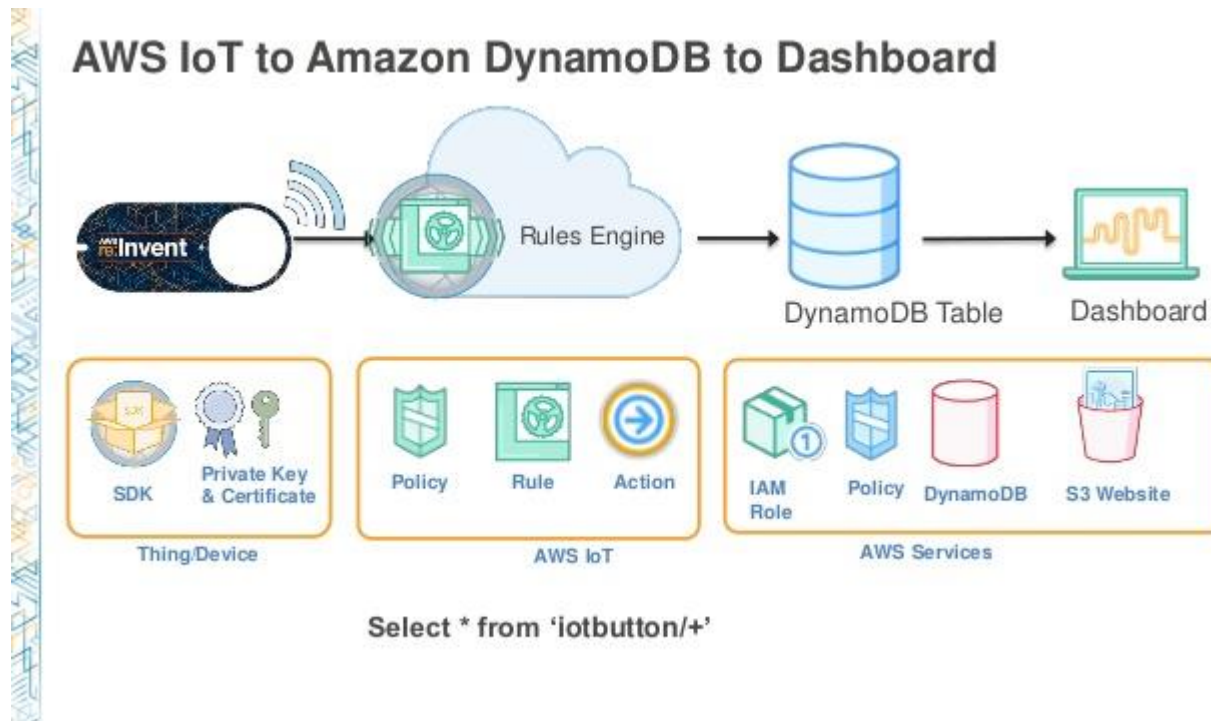


The screenshot displays a 'Fleet Management' dashboard for a 'California Logistics Supervisor'. The interface is divided into several sections:

- KPI Summary:** A grid of eight tiles showing various metrics:
 - Fleet Status: 95% In Transit
 - Alerts: 12%
 - At Depot: 10%
 - Critical Alerts: (indicated by a red wrench icon)
 - Backlog Status: (indicated by a blue line graph icon)
 - On Time Delivery: 98%
 - Driver Utilization: 91%
- California Operations Map:** A map of California with numerous green and red circular markers indicating vehicle locations and status across various cities like San Francisco, San Jose, and Sacramento.
- Alerts:** A list on the right side of the dashboard showing categories such as Maintenance, Driver Fatigue, Traffic, Delivery, and Weather, with some entries partially obscured by a grey box.
- Navigation:** A sidebar on the left contains icons for Home, Alerts, Assets, Staff, and Admin.

Amazon AWS

- AWS(Amazon Web Service)
- Launch IoT platform in beta
- Interesting data delivery model
 - Publishing messages to the message broker through topics
 - The broker deliveries messages to all client subscribed on the specific topics



Issues with Clouds

- Cloud assumes enough bandwidth
 - Strong assumptions for Industrial IoT applications
- Cloud centralizes the analytics thus resulting in slower reaction times
 - Latency-sensitive IoT applications
- Connectivity to the cloud is a prerequisite
 - Need to work even when connection is temporarily unavailable



Oil
Platform
Limited Bandwidth



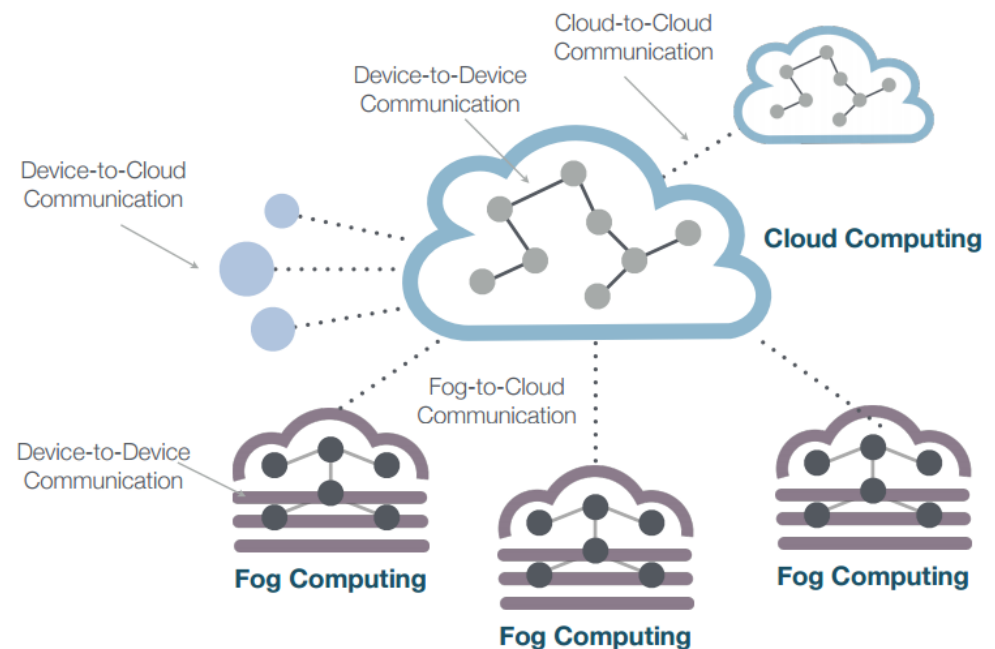
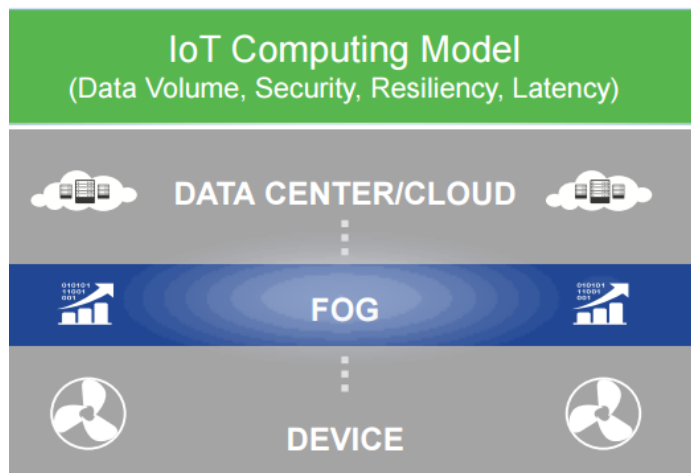
Manufacturing
Robot
Latency



Construction
Crane
Network Reliability

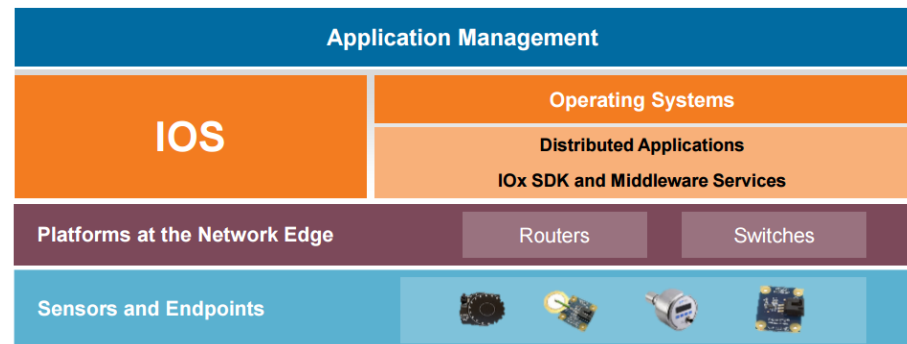
Fog Computing

- Fog computing: computing on the edge
 - Close to the ground, right where things are getting done.
 - Communicate peer-to-peer to efficiently share data and take local decisions
 - Faster processing and interaction time
 - Fewer resources consumed
- IoT systems require both of fog devices and clouds

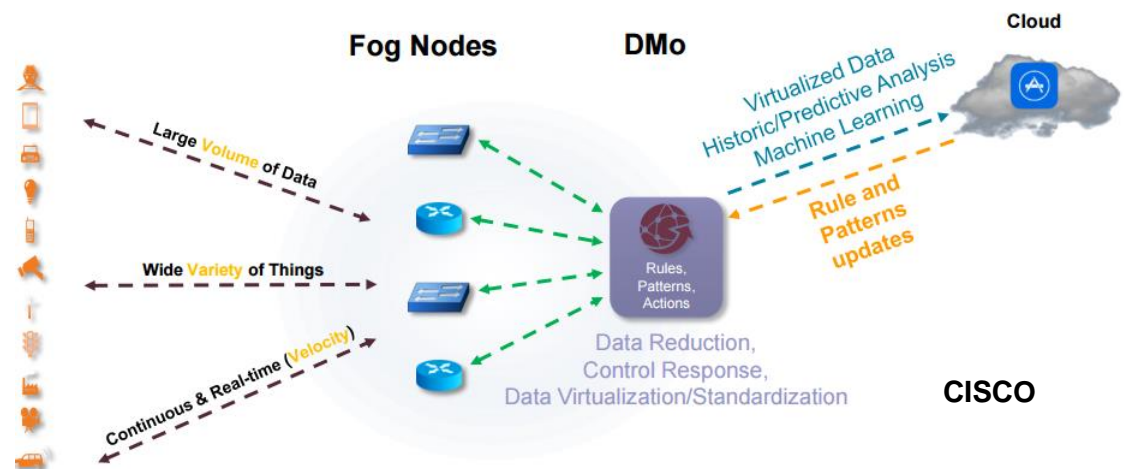


Example of Fog Computing : Cisco IoT Solution

- Developing software architecture for fog computing
 - Target to large harsh environments such as roadways, railways, and utility field substations
 - Data management
 - Redundancy and failover

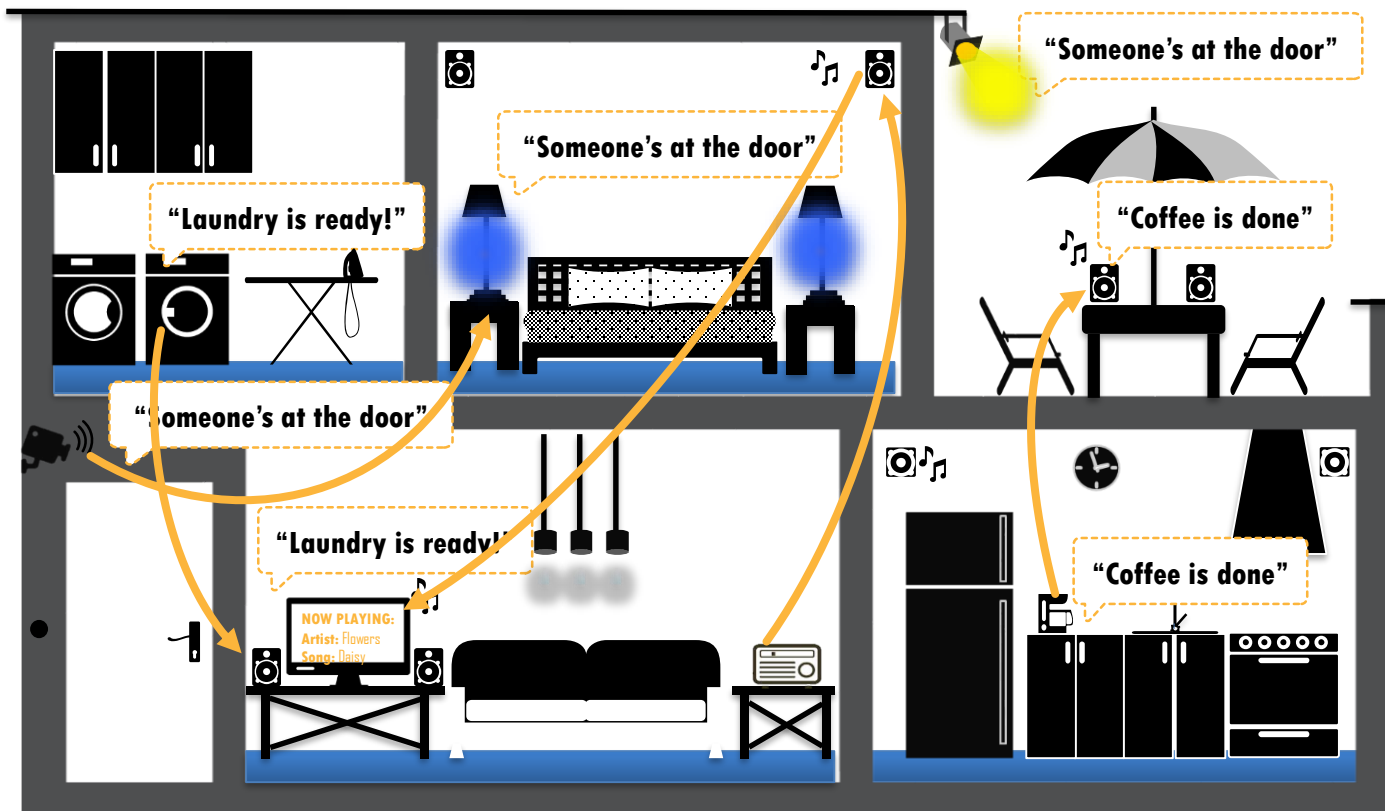


- DMO (Data in Motion)
 - Get updated rules for data from cloud
 - Cache and standardize raw data of sensor
 - Compress and Reduce data



Revisit Action: Actuator Control

- How to take an action?
- IoT devices are characterized by the software they run so:
 - When IoT devices talk to IoT devices it is software talking to software



Software for Actuator Controls

- We have logs of different “things” that have different functionalities. Any problem in designing software platforms?
- What we don’t want:
 - There’s an App for that thing
 - And an App for that thing
 - And yet another App for that thing
 - ...
- APIs are how software talks to software
 - Clear boundaries between the internals of actuator “things” and **the external interfaces** exposed to other software.
 - Generally call external interfaces Application Programming Interfaces (APIs)
 - Makes it possible to **incorporate existing functionality** into new code

Software API Design

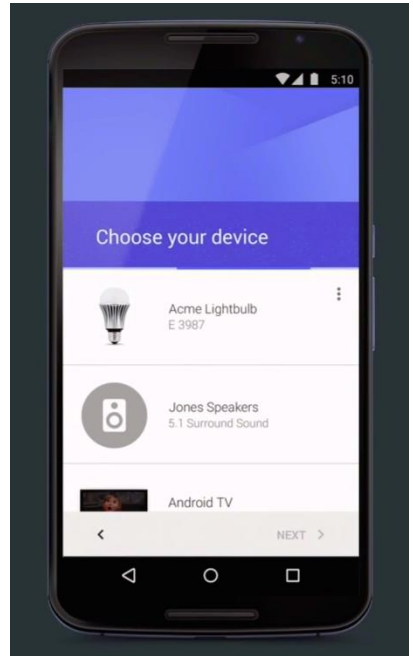
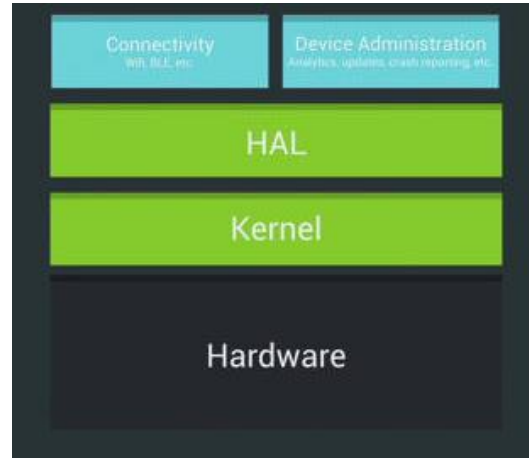
- Devices that have similar functionality should expose the same APIs
 - Every device with a clock should expose a set-clock API
 - Every device that has a battery should expose a battery level API
 - Manufacturers need to expose necessary functionality in a same way
- IoT devices should have APIs
 - IoT devices are all so different
 - Standard APIs + device-specific APIs
- Also need to support by software:
 - Security against unintended actuations
 - Timeliness, robustness
 - Energy saving and autonomic capabilities

Google Brillo & Weave

- Brillo: IoT OS
 - Derived from Android
 - Minimal system requirement
 - Secure
 - Unified interface

- Weave: Network Protocol
 - XML-like semantic for exposure of each “thing”
 - Support communication across architectures

- OnHub: First product
 - WiFi AP with monitoring
 - Streaming



Apple HomeKit



- Home automation solution from Apple
 - User can set settings of “things” over different contexts
 - “Leaving home”: turns off the lights, locks your doors, lowers the thermostat
- SDK published
 - Can implement as usual iOS app
 - Provide simulators and sample layout for a home
 - APIs abstract home elements to objects: HMHome, HMRoom, HMAccessory, ...
 - Work with secured data base per home



Lighting



Locks



Heating + Cooling



Plugs + Switches



Sensors



Shades



Summary

- System software is required to fill the gap between sensor and action
- We review diverse software platform and methodology in hierarchy.
 - OS for small “things”
 - Raw data standardization
 - Semantic data assimilation across layers
 - Cloud solution + fog computing
 - Programming model for actuation of things