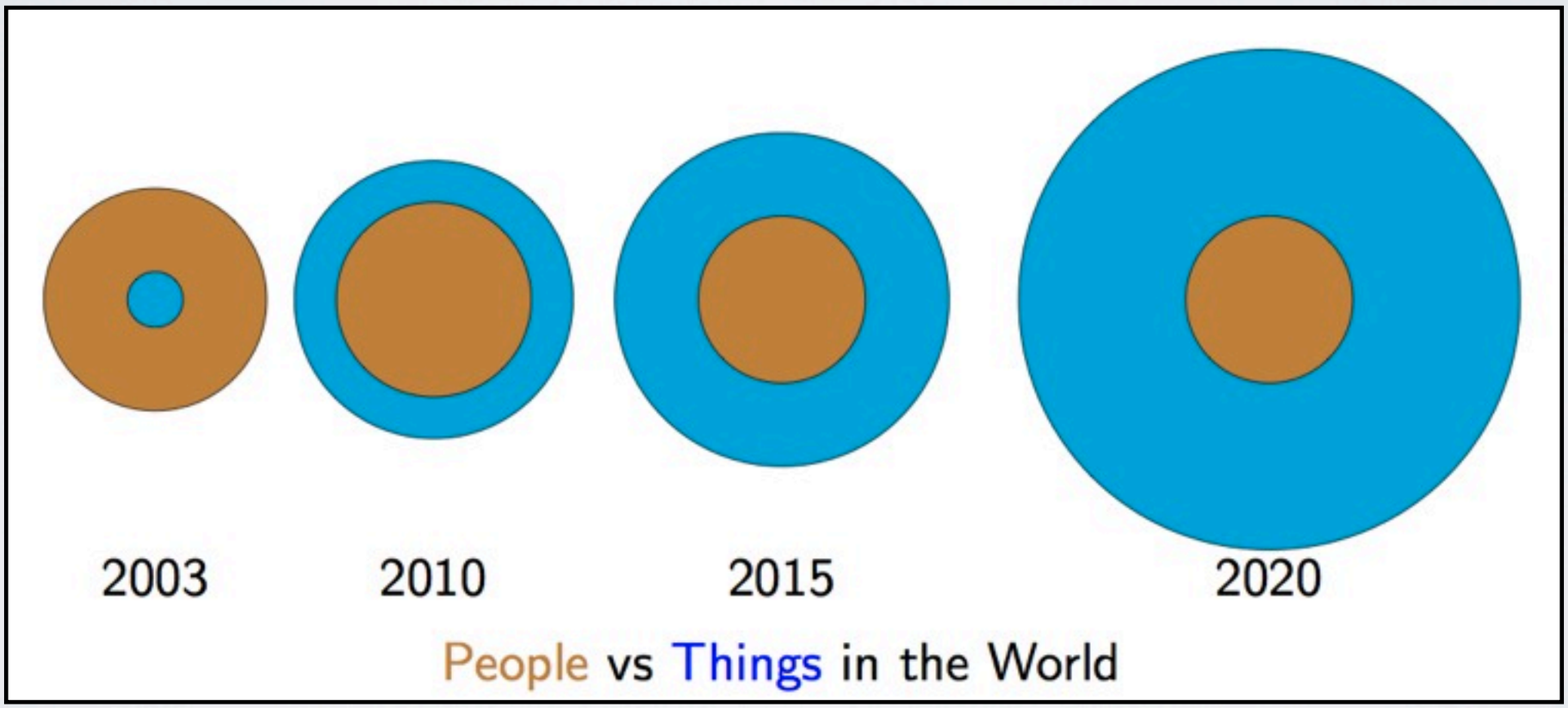
PMH

Home Automation Made Right

SPITFIRE

Internet of Things



Cisco Infographic describing the Internet of Things

So what?

Y U NO REDUCE POWER CONSUMPTION?

Y U NO REMOTE CONTROLS?

Why FOSS?

Why FOSS?

- We love FOSS!

Why FOSS?

- We love FOSS!
- The internet of things must be open

Why FOSS?

- We love FOSS!
- The internet of things must be open
- We embrace knowledge exchange

Why FOSS?

- We love FOSS!
- The internet of things must be open
- We embrace knowledge exchange

When you want something done quickly,
put a big team behind it

Internet of Things

Where does it lead?

Embedded Systems and Internet have evolved naturally to become extremely decentralized.

We need to establish a common language (like CoAP) and/or a system that will bridge the various sensor data formats.

Enter PMH

- Wireless Sensor Network
- Arduino Nodes
- Single Network Controller
  - Arduino Ethernet + XBee
  - PC + XBee
  - plug/embedded computer + XBee

Architecture

data providers

überdust

apps

Things

Controller

Users

Nodes

- Arduino (Pro Mini) based
- XBee Module for 802.15.4
- Sensors
- Actuators

SENSORS

- Air Quality
  - Carbon Monoxide (CO)
  - Methane (CH4)
- Light
- Temperature
- Motion
- Door/Window Open Alarm
- Water Flow
- IR Receiver
- Your own epic awesome sensor™

Actuators

- LEDs
- Motors
- Water Valve
- Any IR Controlled Device
  - TV/HiFi
  - Air Conditioning
- Relays
- Lights
- Water Heater
- Electric Shutters
- Electric door lock

Device Types

Power Strip

- Built-in Arduino + XBee
- Control Each device independently
- Monitor Power Consumption

Device Types

Desk Lamp

- Built-in Arduino + XBee
- Control the light

Device Types

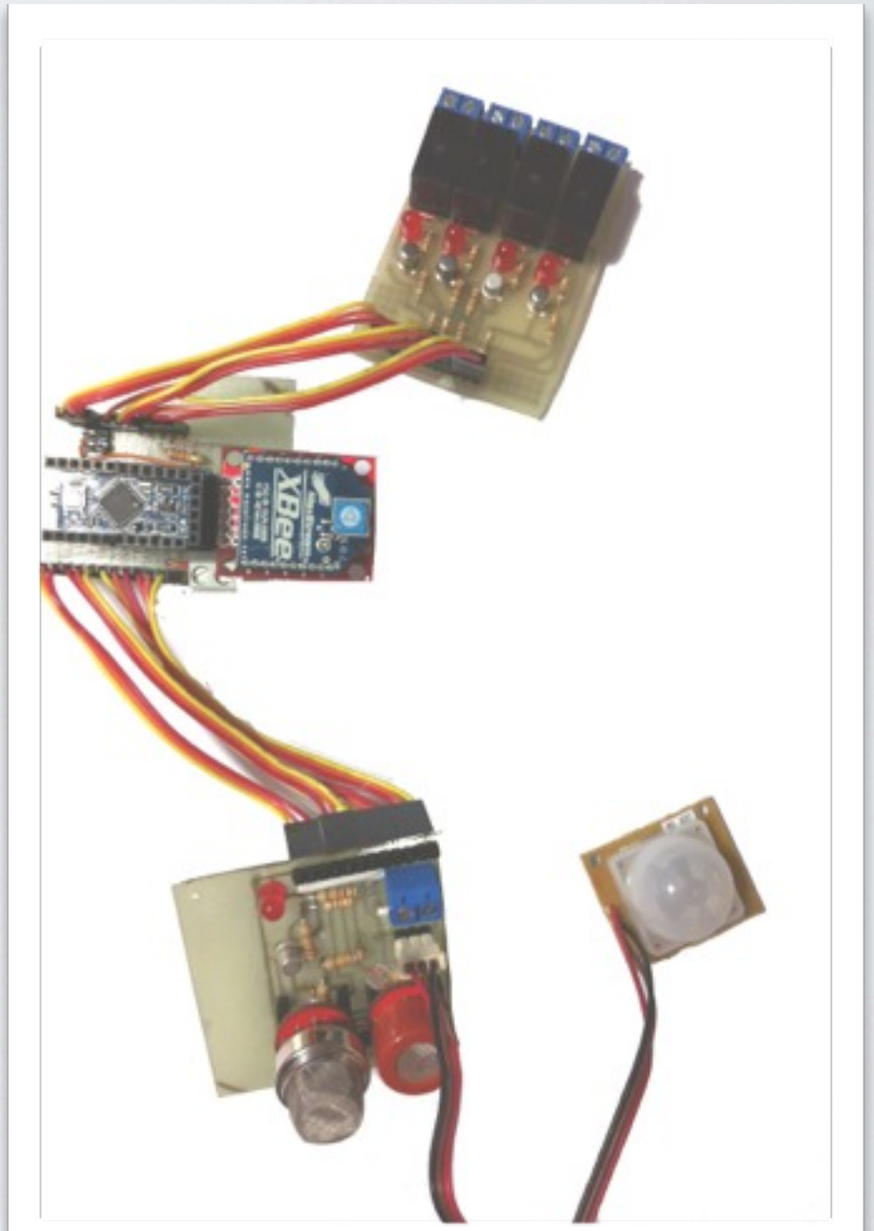
Distribution Board

- Control Power Lines
- Control Lighting
- Control Water Heater
- Power Consumption Meter

Device Types

Generic Nodes

- Built-in Arduino + XBee
- Relay shield
- Sensor shield



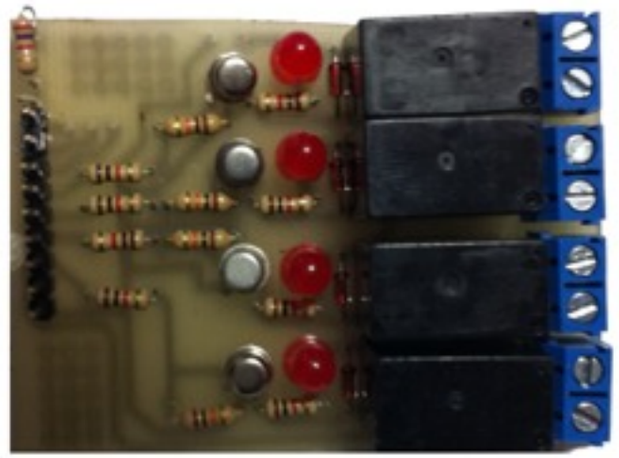
Device Types

Main PCB

- Arduino Pro Mini
- XBee
- Shield Existence Checking circuit

Device Types

Relay Shield



Device Types

Sensor Shield

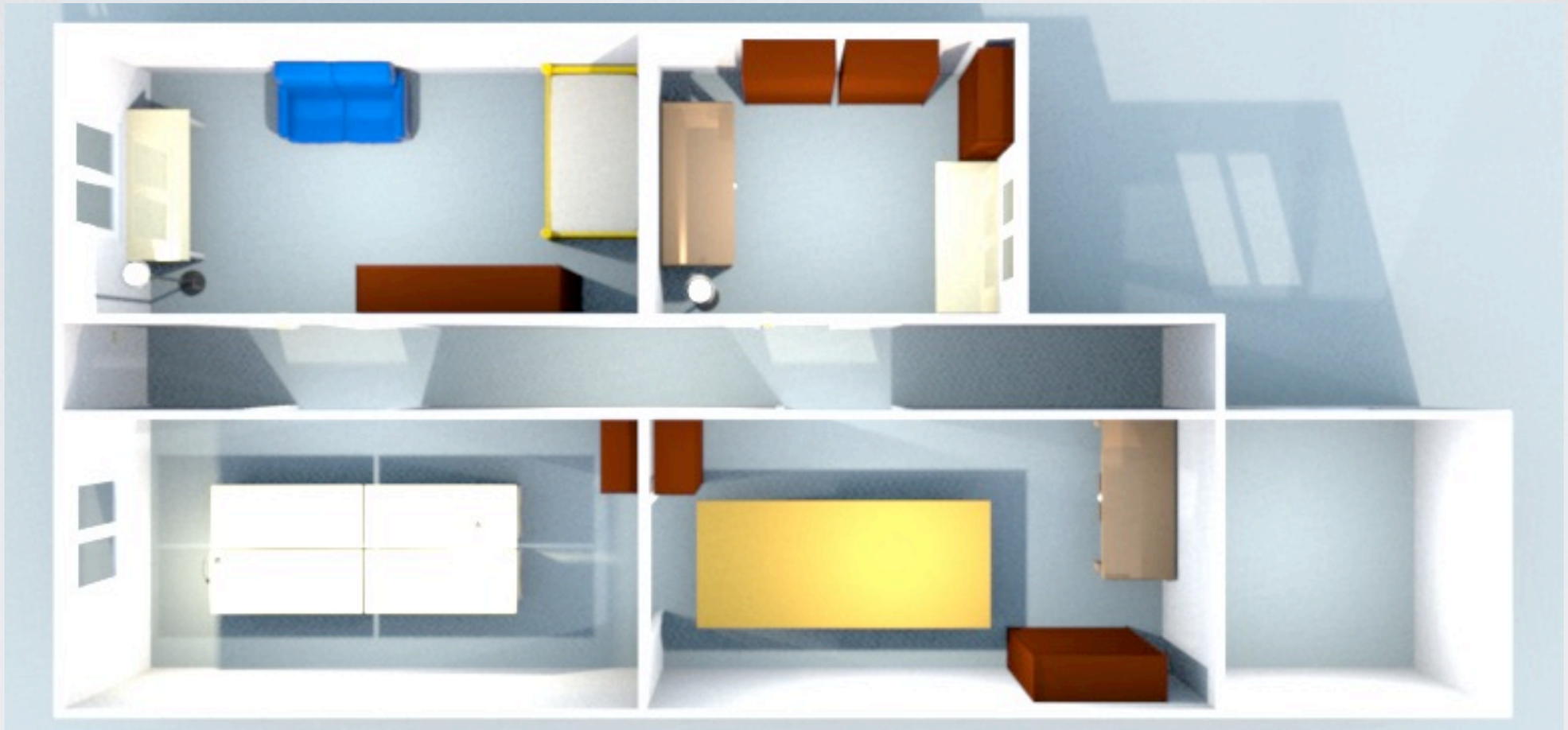
- Gas (CO, CH4)
- Motion
- Temperature
- Light
- Door/Window Alarm
- Status LED

Build your own sensor

You take care of the hardware, we take care of the software

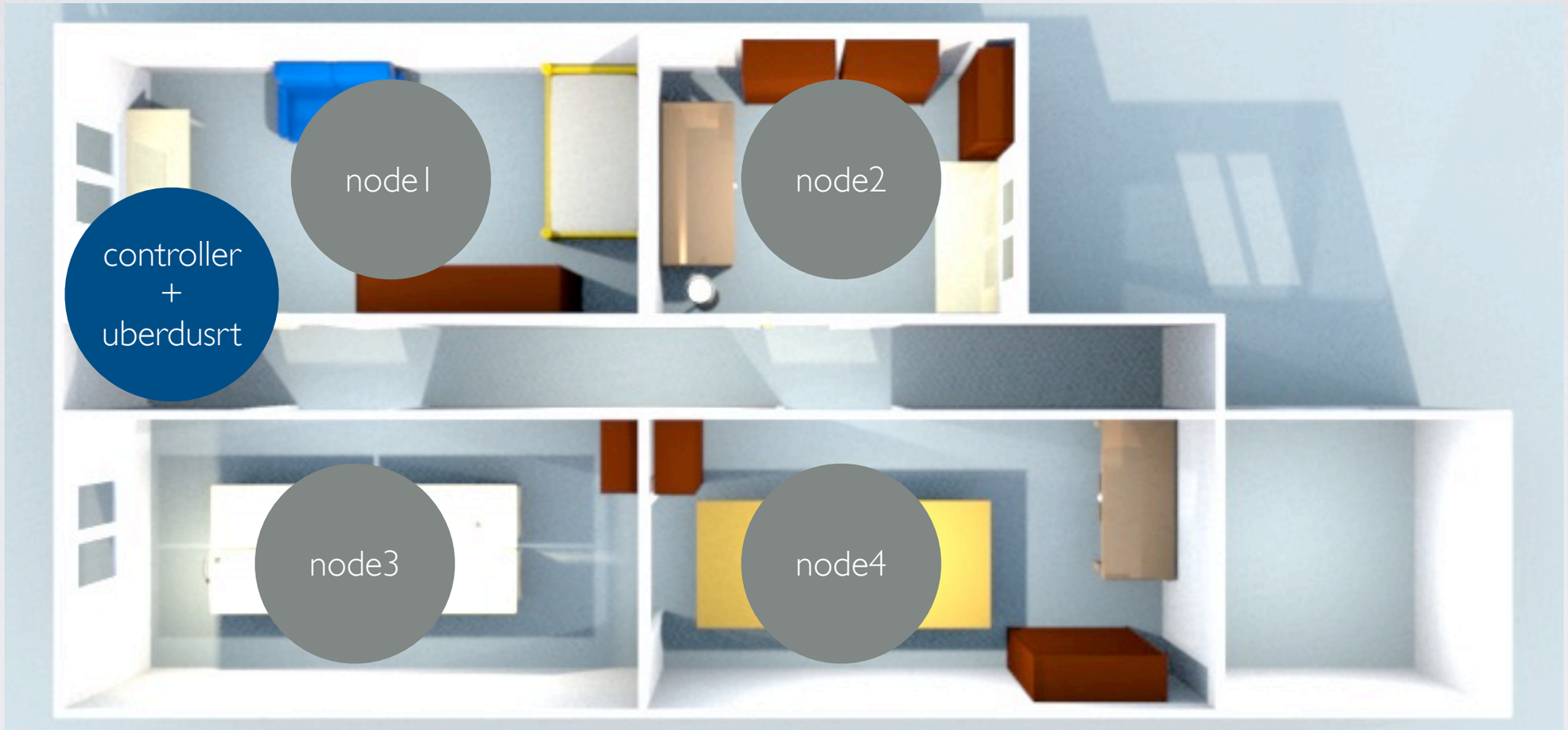
Compute your values, specify a new capability name, and start transmitting

Example



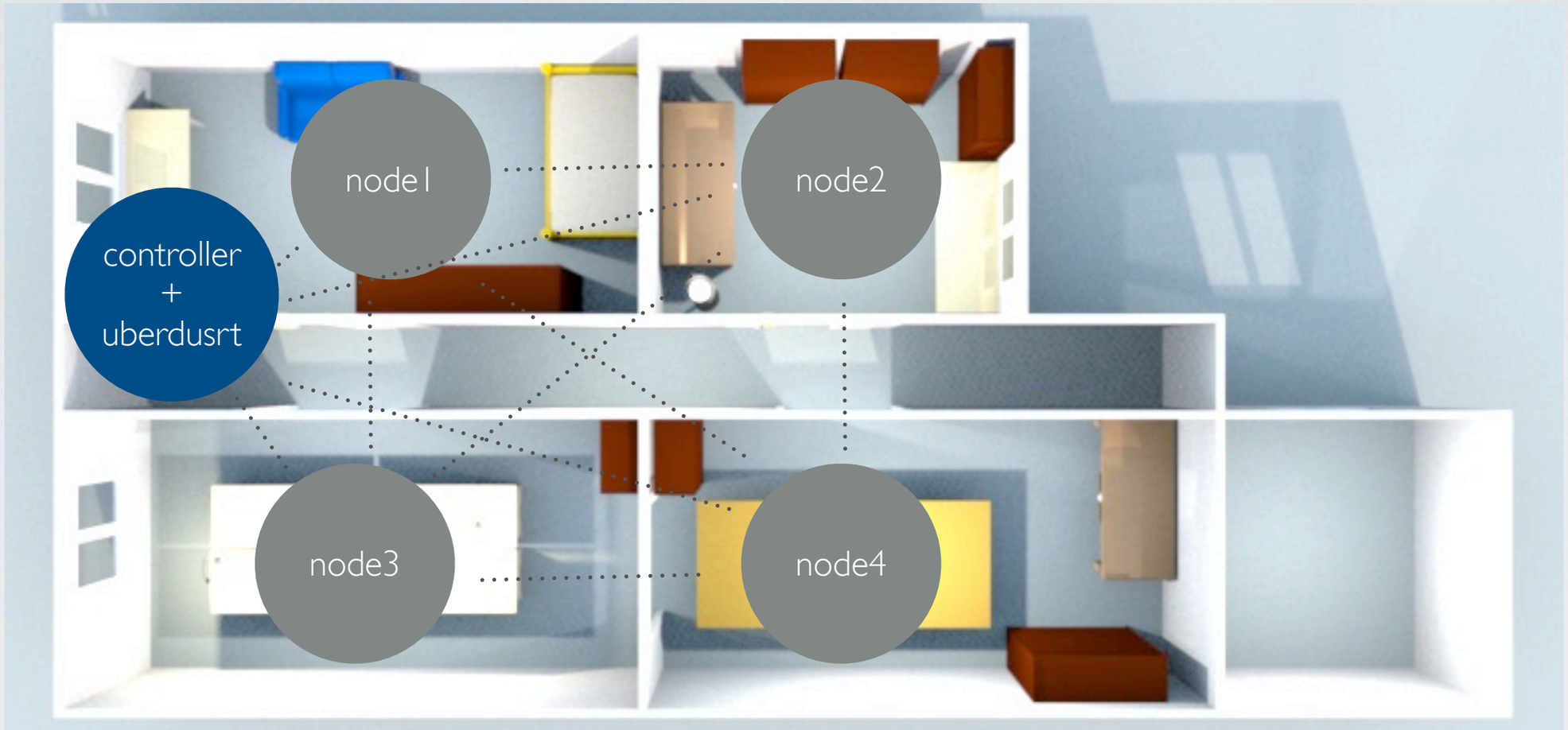
P-Space

Example



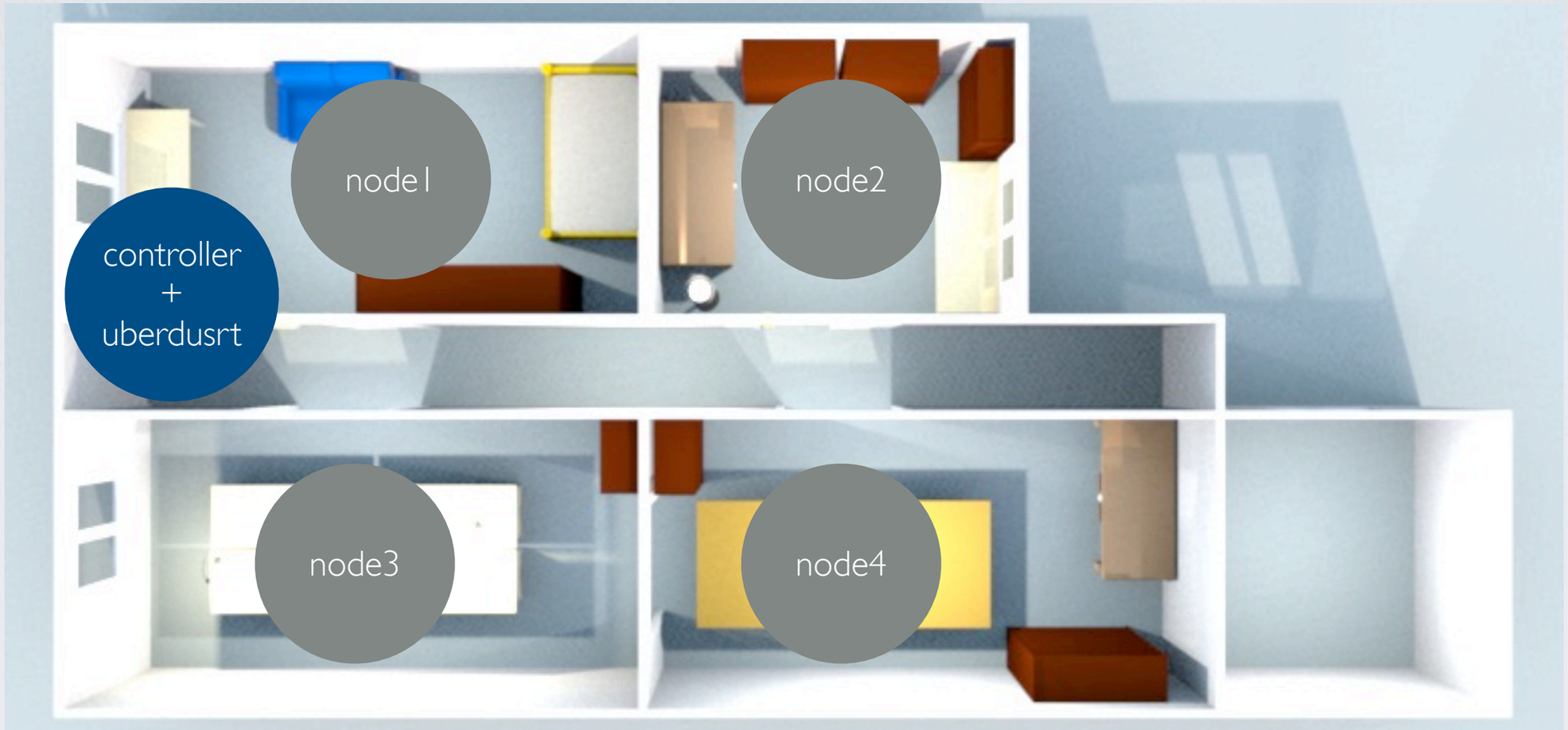
P-Space

Example



P-Space

Example



P-Space

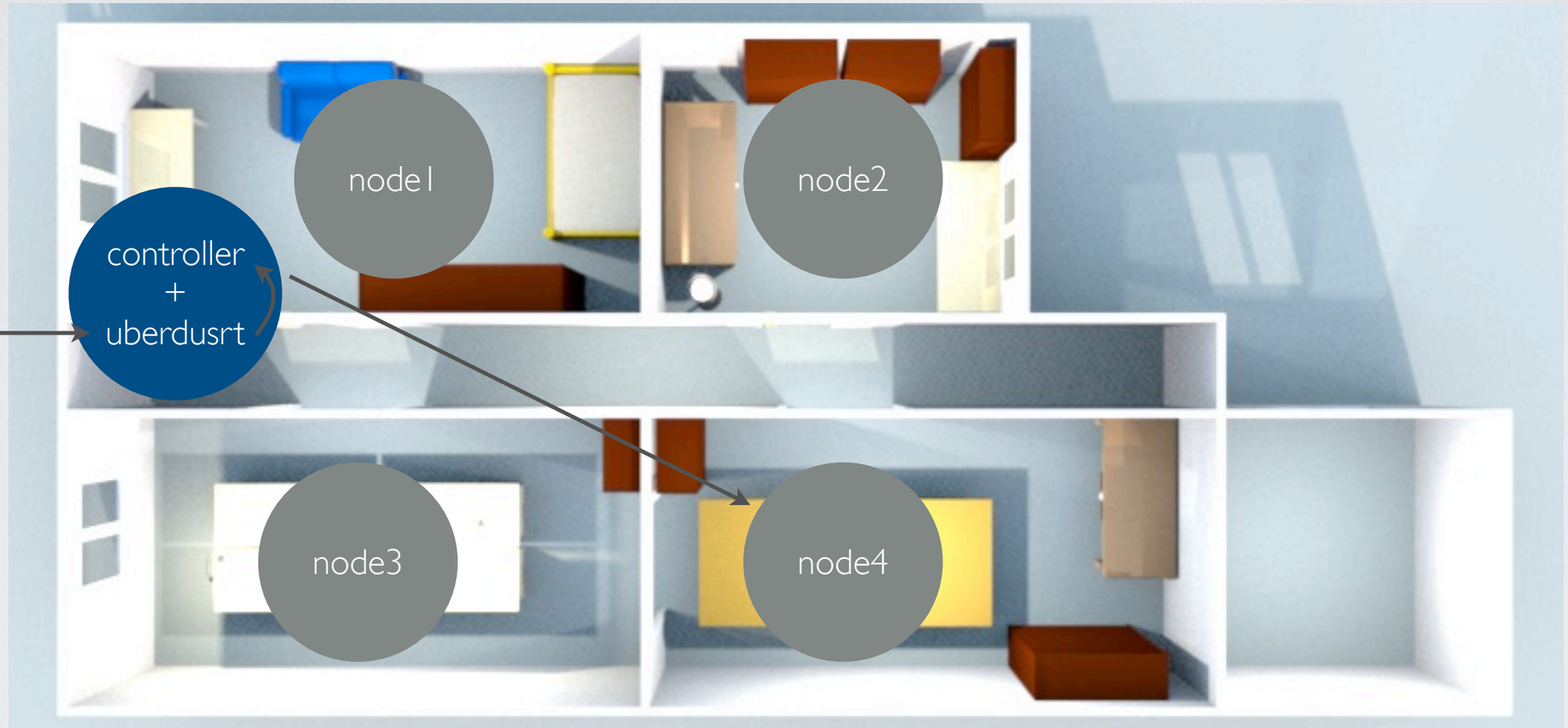
Example

P-Space

Example

P-Space

Example



P-Space

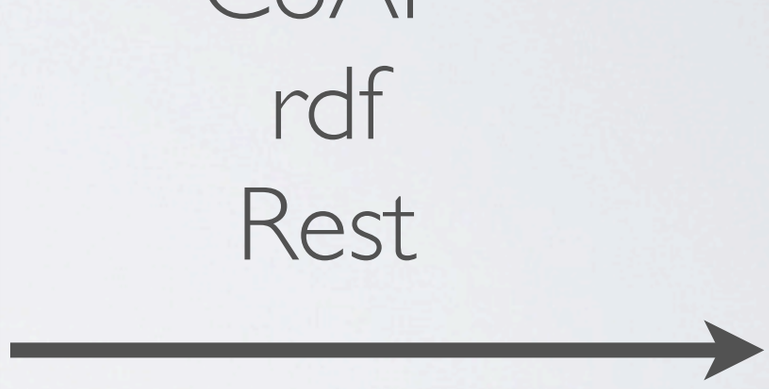
We are über

überdust

CoAP
rdf

Rest
WebSockets

Web UI
Spring
Hibernate
DB



CoAP
rdf
Rest

WebSockets
HTML
Plaintext
JSON

App Layer

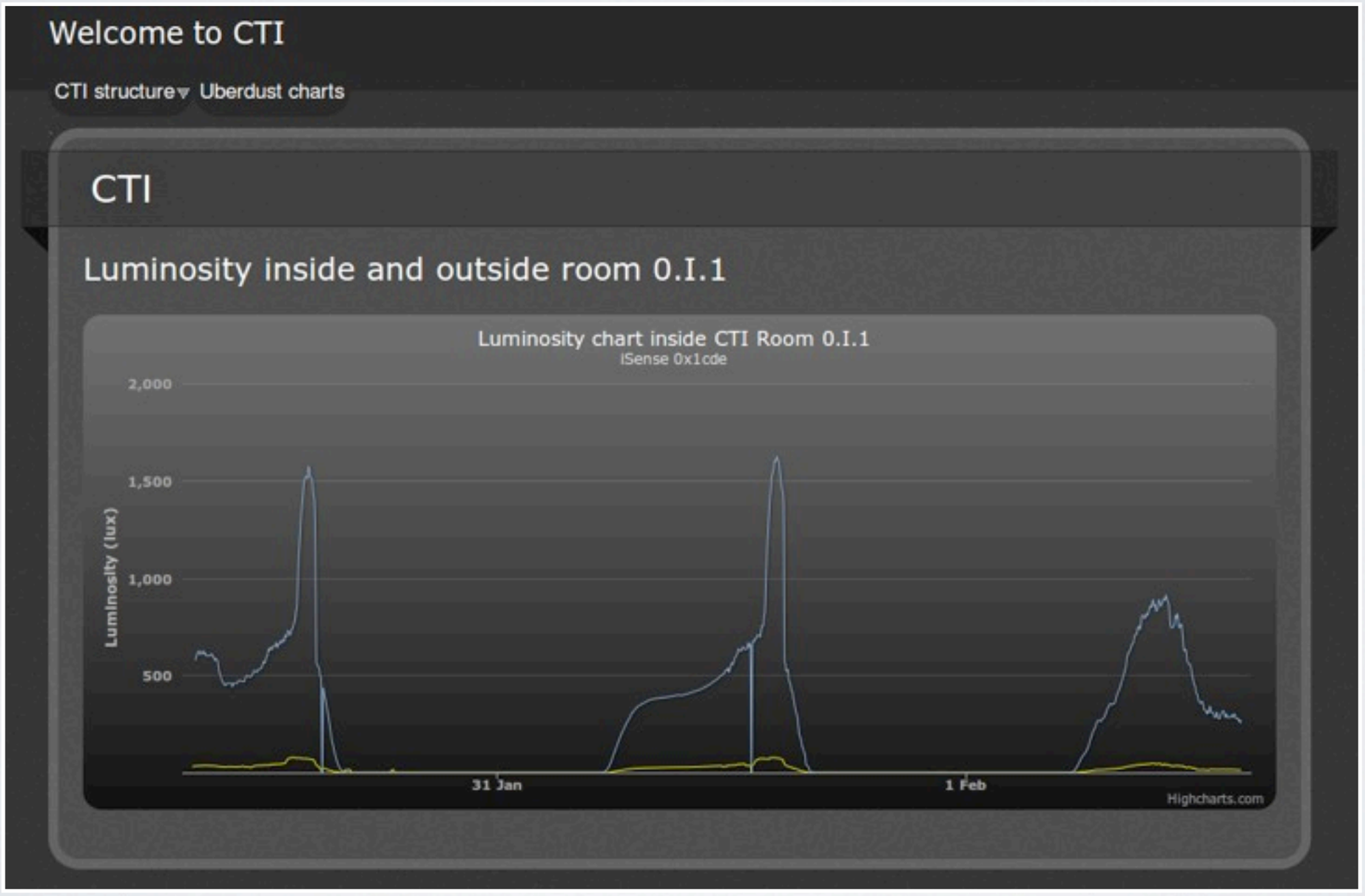
- Web Interface (in development)
- Smartphone Interface (in development)
- APIs

APIs

- REST
  - Get Value
  - Send Command
- WebSockets
  - Get Value
  - Cleaner and faster status updates

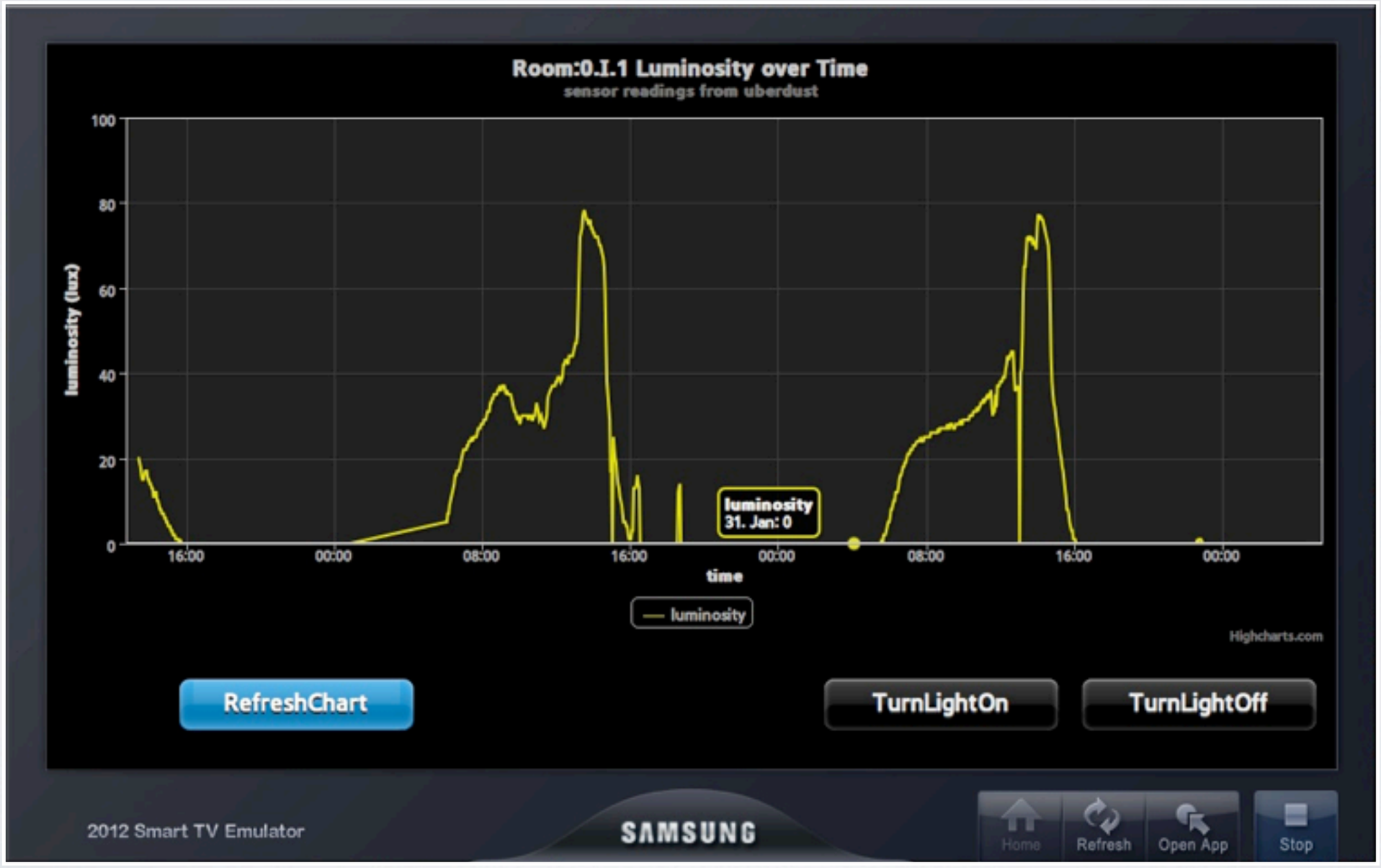
API Examples

Drupal Plugin



APIs

Samsung SmartTV App



APIs

Android App

Sample Code

Bash Script

```
#!/bin/bash
#execute commands
VAL=$(acpi | cut -d " " -f 4)

#truncate data
charge=${VAL%"%","*}

sen="charge"
val=$charge

time=$(date +%s)
#create url
prefix="localhost:8081/rest/testbed/1/node/"
URL=$prefix$HOSTNAME"/capability/"$sen\
"/insert/timestamp/"$time"000/reading/"$val"/"
#execute
wget $URL -O /dev/null
```

wget http://localhost:8081/rest/testbed/1/node/execusor/capability/charge/insert/timestamp/1328303716/reading/99%/ -O /dev/null

Sample Code

Python Script

```
import sys
import getopt
import httplib

node = "urn:wisebed:ctitestbed:0xa4a"
# form rest calls from options args
rest = "".join(("/rest/sendCommand/destination/"
                      ,node,"/payload/1,1,",str(state)))
conn = httplib.HTTPConnection("localhost:8081")
print "Connecting to http://localhost:8081"
conn.request("GET",rest)
print "GET ",rest
response = conn.getresponse()
if(response.status == 200):
    print response.read()
else:
    print response.status,response.reason
```

GET /rest/sendCommand/destination/urn:wisebed:ctitestbed:0xa4a/payload/1,1,1

Sample Code

Java WebSockets

```
final String PROTOCOL = "INSERTREADING";
final String webSocketUrl= "ws://"+server+"insertreading.ws";

factory = new WebSocketClientFactory();
factory.setBufferSize(4096);

factory.start();
client = factory.newWebSocketClient();
client.setMaxIdleTime(-1);
client.setProtocol(PROTOCOL);

// open connection
connection = client.open(new URI(webSocketUrl),
    new InsertReadingWebSocketIMPL()).get();
```

Sample Code

Java WebSockets

```
final NodeReading nodeReading = new NodeReading();
nodeReading.setTestbedId(String.valueOf(testbedId));
nodeReading.setNodeId(nodeUrn);
nodeReading.setCapabilityName(capabilityName);
nodeReading.setTimestamp(msec);
nodeReading.setReading(String.valueOf(value));

InsertReadingWebSocketClient.getInstance().sendNodeReading(nodeReading);
```

DEMO!

Murphey, please let it work for once

Output Styles

Text, HTML

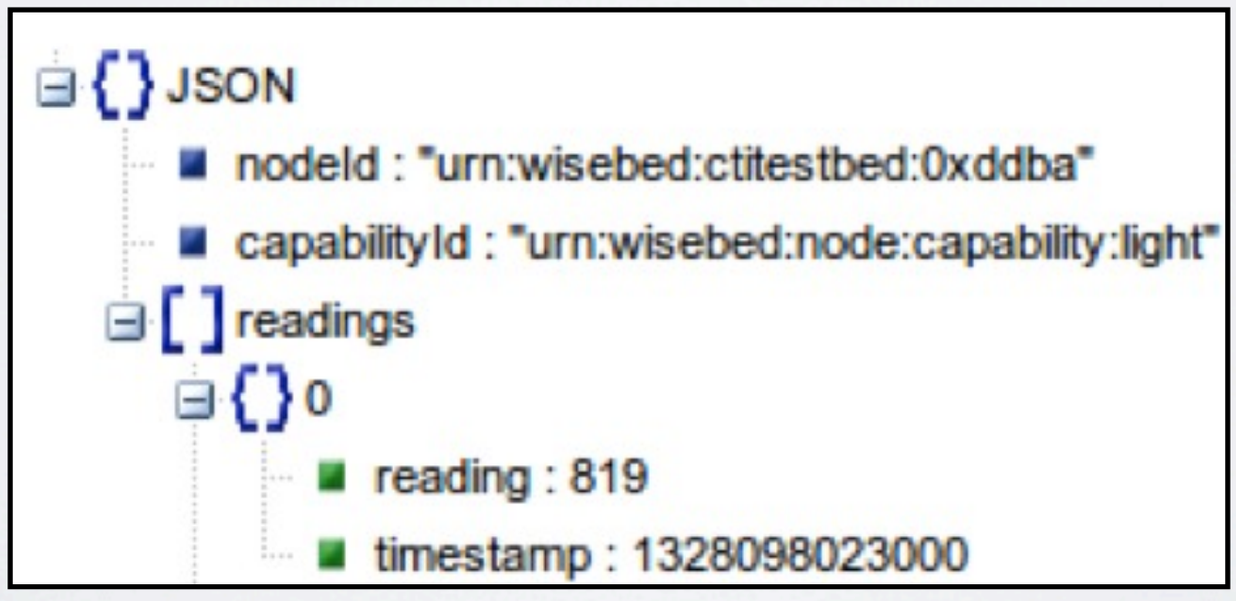
1328098023000	819.0
1328097964000	811.0
1328097904000	821.0
1328097844000	788.0
1328097783000	788.0
1328097723000	821.0
1328097663000	828.0
1328097544000	867.0
1328097484000	874.0
1328097424000	898.0

Timestamp	Readings(20)
2012-02-01 14:02:03.0	821.0
2012-02-01 14:01:03.0	828.0
2012-02-01 13:59:04.0	867.0
2012-02-01 13:58:04.0	874.0
2012-02-01 13:57:04.0	898.0
2012-02-01 13:56:03.0	922.0
2012-02-01 13:55:03.0	929.0
2012-02-01 13:54:03.0	936.0

Output Styles

JSON, Rdf

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/terms/"
    xmlns:dul="http://www.loa-cnr.it/ontologies/DUL.owl#"
    xmlns:spitfire="http://spitfire-project.eu/cc/spitfireCC_n3.owl#"
    xmlns:ssn="http://purl.oclc.org/NET/ssnx/ssn#" >
  <rdf:Description rdf:nodeID="A0">
    <dul:hasValue>0.0</dul:hasValue>
    <ssn:observedProperty rdf:resource="http://dbpedia.org/resource/Luminosity"/>
    <rdf:type rdf:resource="http://purl.oclc.org/NET/ssnx/ssn#Sensor"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <dul:hasValue>25.0</dul:hasValue>
    <ssn:observedProperty rdf:resource="http://dbpedia.org/resource/Temperature"/>
    <rdf:type rdf:resource="http://purl.oclc.org/NET/ssnx/ssn#Sensor"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://localhost:8081/urn:wisebed:ctitestbed:0x498/rdf#">
    <ssn:attachedSystem rdf:nodeID="A1"/>
    <ssn:attachedSystem rdf:nodeID="A0"/>
  </rdf:Description>
</rdf:RDF>
```



One more thing..

You can use anything you want
as a data provider

Just use our REST interface to send your
values to überdust

Links & Info

- https://github.com/itm/uberdust - PMH source (überdust layer, arduino hardware)
- https://github.com/itm/uberdust/wiki - Documentation
- https://github.com/mksense/mac - arduino software & mkSense, an arduino library used for 802.15.4 communication
- #pmh on freenode

Thank You

Q & A

Computer Technology Institute and Press
"Diophantus",Patras Greece

Vasilis Georgitzikis
billgeo13@gmail.com
irc: tzikis @ freenode
twitter: @tzikis

SPITFIRE

Semantic-Service Provisioning for the Internet of Things using Future Internet Research by Experiments
wwww.spitfire-project.eu