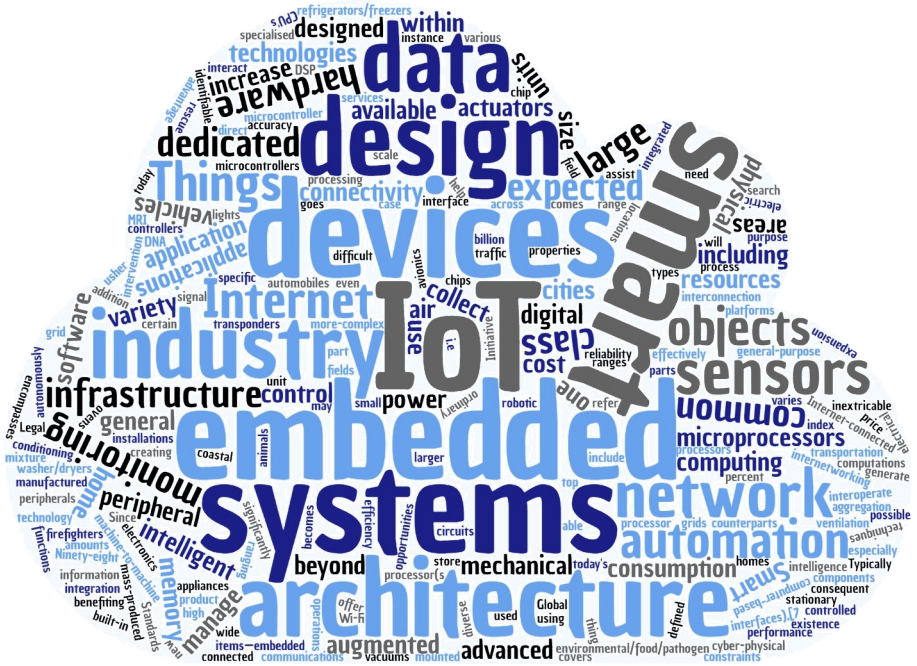
Associate Professorship of Embedded Systems and Internet of Things
Department of Electrical and Computer Engineering
Technical University of Munich

Lecture
IoT Remote Lab

06 – Scripting API and node-wot

Ege Korkan



Zoom Guidelines

- The lectures and tutor sessions happen on Zoom meetings following the link sent to you via email.
- Participatition to Zoom sessions is optional
- You can choose a random string for your name
- The Zoom chat will not be recorded and we will not save the chat.
- All the participants except the lecturer is muted. The participants are free to unmute. You can also go to participants, click the hand icon to raise your hand.
- You can also do other things, like asking me to go slower. I have a separate window where I look at the requests from the participants.
- You can ask quick questions in Zoom chat or use the Tweedback link provided in each session.

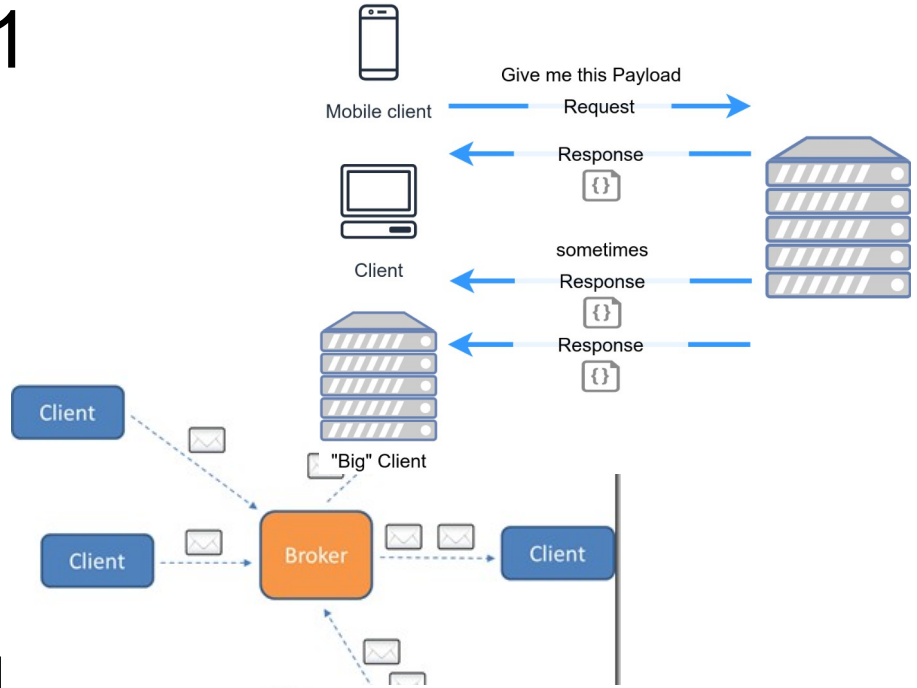
Tweedback for Real-time Q&A During the Lecture

https://arsnova.eu/mobile/#id/13971057

Recap: Season 1

```
{
  "productId": 1,
  "productName": "An ice sculpture",
  "price": 12.50,
  "tags": [ "cold", "ice" ],
  "dimensions": {
    "length": 7.0,
    "width": 12.0,
    "height": 9.5
  },
  "warehouseLocation": {
    "latitude": -78.75,
    "longitude": 20.4
  }
}
```

```
{
  "$schema": "http://json-schema.org/draf
  "$id": "http://example.com/product.sche
  "title": "Product",
  "description": "A product from Acme's c
  "type": "object",
  "properties": {
    "productId": {
      "description": "The unique identifi
      "type": "integer"
    },
    "productName": {
      "description": "Name of the product
      "type": "string"
    },
    "price": {
      "description": "The price of the pr
      "type": "number",
      "exclusiveMinimum": 0
    },
```




```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "id": "urn:dev:ops:32473-WoTLamp-1234",
    "title": "MyLampThing",
    "securityDefinitions": {
        "basic_sc": {"scheme": "basic", "in":"header"}
    },
    "security": ["basic_sc"],
    "properties": {
        "status" : {
            "type": "string",
            "forms": [{
                "href": "https://mylamp.example.com/status",
                "htv:methodName":"GET"
            }]
        }
    },
    "actions": {
        "toggle" : {
            "forms": [{
                "href": "https://mylamp.example.com/toggle",
                "htv:methodName":"POST"
            }]
        }
    },
    "events":{
        "overheating":{
            "data": {"type": "string"},
            "forms": [{
                "href": "https://mylamp.example.com/oh",
                "htv:methodName":"GET",
                "subprotocol": "longpoll"
            }]
        }
    }
}
```

Recap

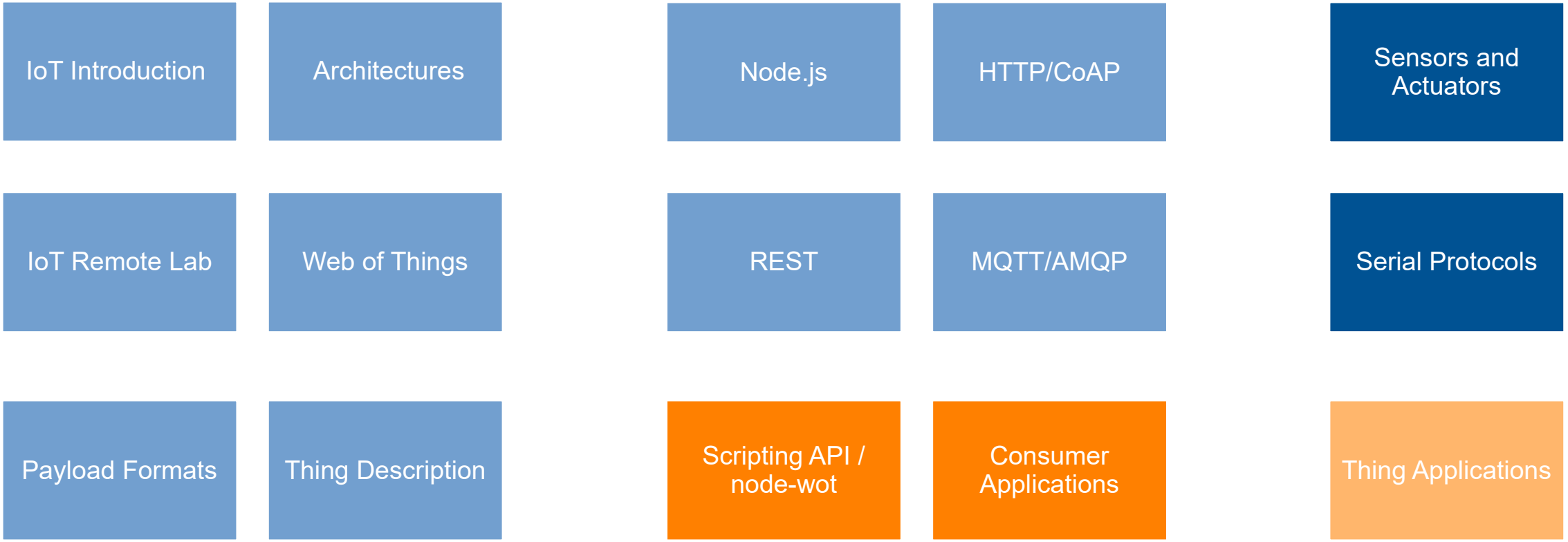
```
async function myFunction() {

  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("done!"), 1000)
  });

  let result = await promise; // wait until the promise resolves (*)

  console.log(result); // "done!"
}

myFunction();
```

```
class Student {
    fullName: string;
    constructor(public firstName: string, public middleInitial: string, public lastName: string {
        this.fullName = firstName + " " + middleInitial + " " + lastName;
    }
}

interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person: Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}

let user = new Student("John", "M.", "Doe");

console.log(greeter(user));

```

Vocabulary term	*Description*	*Assignment*	*Type*
htv:methodName	HTTP method name (Literal).	optional	string (one of "GET", "PUT", "POST", "DELETE", "PATCH")
htv:headers	HTTP headers sent with the message.	optional	array of htv:MessageHeader
htv:fieldName	Header name (Literal), e.g., "Accept", "Transfer-Encoding".	mandatory within htv:MessageHeader	string
htv:fieldValue	Header value (Literal).	mandatory within htv:MessageHeader	string

Vocabulary term	*Description*	*Assignment*	*Type*
mqv:controlPacketValue	MQTT Control Packet type (Literal).	optional	string (one of "PUBLISH" (3), "SUBSCRIBE" (8), "UNSUBSCRIBE" (10))
mqv:options	MQTT options sent with the message, e.g., [{ "mqv:optionName": "qos", "mqv:optionValue": 1 }].	optional	array of mqv:MessageOption
mqv:optionName	Option name (Literal).	mandatory within mqv:MessageOption)	string (one of "qos", "retain", "dup")
mqv:optionValue	Header value (Literal).	mandatory within mqv:MessageOption)	One of 0, 1 or 2 (only for qos)

Course Contents



Can we write some code please?

- Until now, you were writing JSON files…
- The goal was to get you really familiar with JSON, JSON Schema and TD!

Coding in the context of WoT

You can imagine different types of programs:

- Consumer applications: Using a Thing by understanding its TD, i.e. sending requests, processing responses
- Thing applications: Creating a Thing and exposing it via its TD, i.e. handling requests, generate data from physical processes.
- Both! → Servient

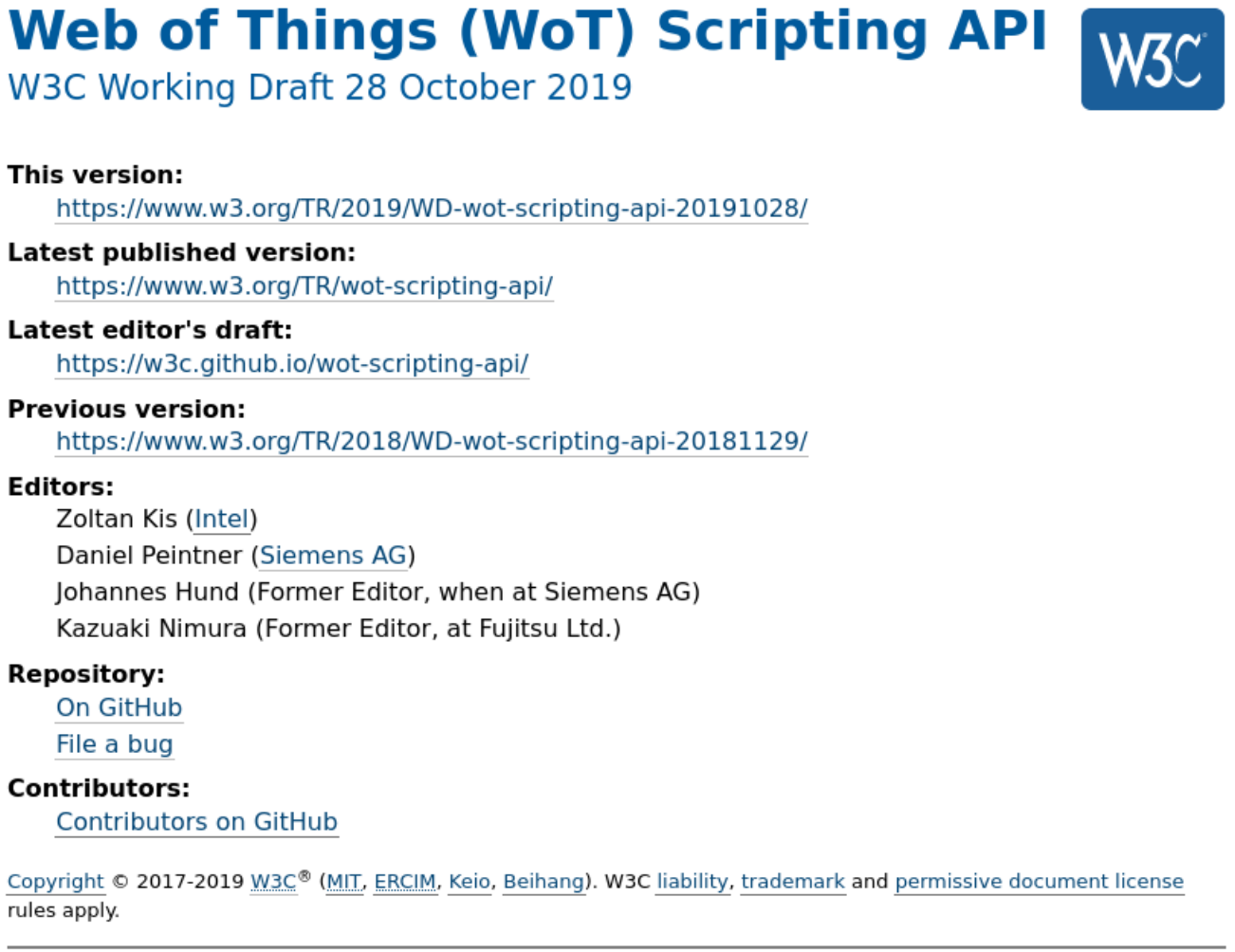
Coding in the context of WoT

- How would you do it?
  - Example libraries, frameworks
  - Languages

```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "id": "urn:dev:ops:32473-WoTLamp-1234",
    "title": "MyLampThing",
    "securityDefinitions": {
        "basic_sc": {"scheme": "basic", "in":"header"}
    },
    "security": ["basic_sc"],
    "properties": {
        "status" : {
            "type": "string",
            "forms": [{
                "href": "https://mylamp.example.com/status",
                "htv:methodName":"GET"
            }]
        }
    },
    "actions": {
        "toggle" : {
            "forms": [{
                "href": "https://mylamp.example.com/toggle",
                "htv:methodName":"POST"
            }]
        }
    },
    "events":{
        "overheating":{
            "data": {"type": "string"},
            "forms": [{
                "href": "https://mylamp.example.com/oh",
                "htv:methodName":"GET",
                "subprotocol": "longpoll"
            }]
        }
    }
}
```

Scripting API

- A standardized way to write both types of applications: https://www.w3.org/TR/wot-scripting-api/



Web of Things (WoT) Scripting API

W3C Working Draft 28 October 2019

This version:
https://www.w3.org/TR/2019/WD-wot-scripting-api-20191028/

Latest published version:
https://www.w3.org/TR/wot-scripting-api/

Latest editor's draft:
https://w3c.github.io/wot-scripting-api/

Previous version:
https://www.w3.org/TR/2018/WD-wot-scripting-api-20181129/

Editors:
Zoltan Kis (Intel)
Daniel Peintner (Siemens AG)
Johannes Hund (Former Editor, when at Siemens AG)
Kazuaki Nimura (Former Editor, at Fujitsu Ltd.)

Repository:
On GitHub
File a bug

Contributors:
Contributors on GitHub

Scripting API

Scripting API Implementations

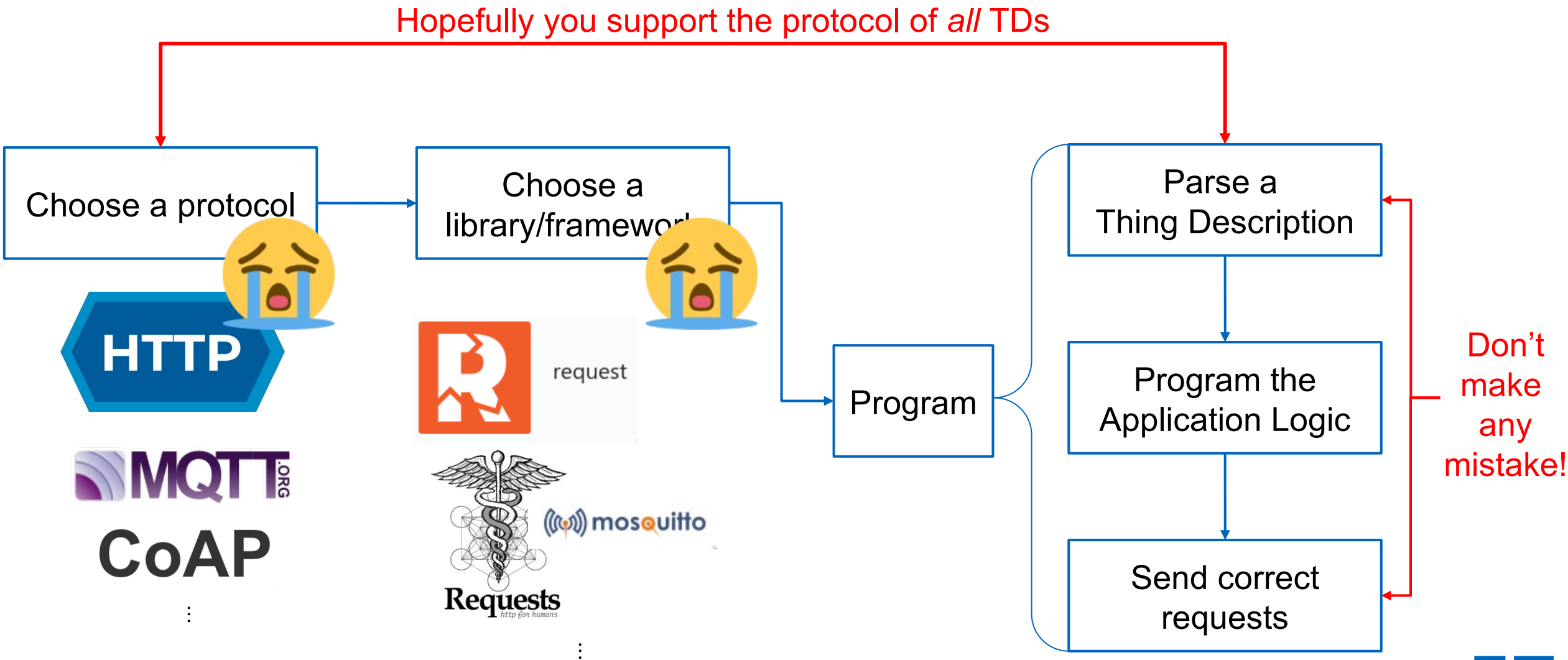
- **Node.js:** Eclipse Thingweb‘s **node-wot**: https://github.com/eclipse/thingweb.node-wot/
- Python (not 100% conform): https://github.com/agmangas/wot-py
- Java (not 100% conform): https://github.com/sane-city/wot-servient

Why node-wot?

- How do I write a device implementation that is WoT compatible without node-wot ?

Why node-wot?

- How do I write programs that interact with WoT devices without node-wot ?



Why node-wot?

- How do I write a device implementation that is WoT compatible with node-wot ?

Program the Backend

- How do I write programs that interact with WoT devices with node-wot ?

Program the Application Logic

What is node-wot?

- Features:
  - Protocols: HTTP, CoAP, MQTT, OPC-UA, Websockets with more that can be implemented
  - Media Types: `application/json`, text/plain, image formats (experimental)
  - WoT Operation Types: `readproperty`, `writeproperty`, `observeproperty`, `unobserveproperty`, `invokeaction`, `subscribeevent`, `unsubscribeevent`, `readallproperties`, `writeallproperties`, `readmultipleproperties`, `writemultipleproperties` (all of them)

What is node-wot ?

Features:

- Security: Basic Authentication, Bearer Tokens, API Key, PSK

Most importantly:

It is **not** an end to end framework!

You can write a Thing implementation and let another library, Web browser, REST client interact with it

Now, let’s see some action!

We will start with simplified concepts and then go into its more advanced uses

Installation

Prerequisites:

- Node.js (10.13.0 +) is needed for all operating systems

Linux

Meet the node-gyp requirements:

- Python v2.7, v3.5, v3.6, v3.7, or v3.8
- make
- A proper C/C++ compiler toolchain, like GCC

MacOS

Meet the node-gyp requirements:

- `xcode-select --install`

Windows

Windows build tools:

```
npm install -g --production
windows-build-tools
```

Installation

1. Clone the repository: `git clone https://github.com/eclipse/thingweb.node-wot`
2. Change into the directory: `cd thingweb.node-wot`
3. Install dependencies: `npm install`
4. Build the source code: `npm run build`
5. Link the packages to enable the `wot-servient` command:

   ```
   sudo npm run link
   ```

If the last command doesn't work, it is not mandatory! In the upcoming slides, replace the `wot-servient` command with `node packages\cli\dist\cli.js`

Alternative installation as npm packages, explained later ;)

Let’s use it!

W3C WoT Scripting API standard is always a good reference to understand the use of different methods of node-wot

How to write a Thing implementation

In the following slides, we will learn how to program a temperature controlling Thing. It:

- provides temperature values as a property
- allows to increase or decrease the temperature via actions
- alerts when the temperature reaches 45°C

```
WoT.produce({
    title: "TemperatureController",
    description: "A Thing to control the temperature of the room and also get alerts in too high
temperatures",
    properties: {
        temperature: {
            type: "integer",
            description: "Current temperature value",
            observable: true,
            readOnly: true,
            unit:"Celsius"
        }
    },
    actions: {
        increment: {
            description: "Incrementing the temperature of the room with 0 to 5 increments",
            input:{
                type: "integer",
                minimum: 0,
                maximum: 5
            }
        },
        decrement: {
            description: "Decrementing the temperature of the room with 0 to 5 increments",
            input:{
                type: "integer",
                minimum: 0,
                maximum: 5
            }
        }
    },
    events: {
        overheat: {
            description: "Alert sent when the room temperature is too high"
        }
    }
})
```

How to write a Thing implementation

Then, we have to program:

- How to read the temperature from the internal sensor
- What happens when increment or decrement actions are invoked
- Have a logic that emits the overheat alert when the temperature exceeds 45°C

Some dummy functions to mimic temperature related operations

```
    function getTemperature() {
        // normally, you would call the temperature sensor's function to read the actual
temperature value
        return Math.random() * Math.floor(50);
        // return 5; //uncomment to test incrementing etc.
    }

    function changeTemperature(newValue){
        // normally, you would do physical action to change the temperature
        //do nothing
        thing.writeProperty("temperature",newValue);
        return;
    }
```

```
thing.setPropertyReadHandler("temperature",function(){
    return new Promise((resolve, reject) => {
        resolve(getTemperature());
    });
});

// set action handlers
thing.setActionHandler("increment", function (value, options) {
    changeTemperature(getTemperature()+value)
});

thing.setActionHandler("decrement", function (value, options) {
    changeTemperature(getTemperature()-value)
});
```

```
setInterval(() => {
    var curTemp = getTemperature();
    thing.writeProperty("temperature",curTemp)
    if (curTemp > 45) {
        thing.emitEvent("overheat")
    }
}, 5000);

// expose the thing
thing.expose().then(function () { console.info(thing.getThingDescription().title +
                                                " ready"); });
```

Full code available at:
https://gist.github.com/egekorkan/ddf2e03f40fb976d9d4b925fbbb9d381

How to use the `wot-servient` command

The program cannot be run directly via node.js, i.e. via

`node myTempController.js`, since the WoT object in the beginning is unknown for the node.js.

Huh?

How to use the `wot-servient` command

`wot-servient` command builds the necessary *infrastructure* and creates the WoT object for our scripts to use. It is the Command Line Interface (CLI) of node-wot !

It uses a default configuration file that can be changed where the desired protocols, security configuration, static address for the generated TD, etc. are specified.

How to use the `wot-servient` command

```
{
  "servient": {
    "clientOnly": false,
    "staticAddress": "example.com",
    "scriptAction": false
  },
  "http": {
     "port": 8080,
     "allowSelfSigned": true
  },
  "coap": {
     "port": 5683,
     "allowSelfSigned": true
  },
  "mqtt" : {
    "broker": "test.mosquitto.org",
    "username": "john",
    "password": "doe",
    "clientId": 123,
    "protocolVersion": 5
  },
  "credentials": {
    "urn:dev:ops:32473-example-1234": {
      "token": "abcde"
    },
    "urn:dev:ops:32473-WoTLamp-1235": {
      "username": "john",
      "password": "doe"
    }
  }
}
```

The command uses this configuration file that has default values.

You can change it and supply your own configuration file with the -f option.

Run `wot-servient -h` to learn all the options!

How to write code to interact with a Thing

You can use any browser or REST Client software such as Postman, Insomnia, cURL for HTTP;

Copper for Chrome for CoAP, MQTT.fx for MQTT, or Node-RED which has multiple protocols.

As long as they use the protocol of the Thing, you can use it!

How to write code to interact with a Thing

But you can do this easier, faster with node-wot and the code you will write will be protocol independent!

```
const TemperatureThingAddress = "http://localhost:8080/TemperatureController";

WoTHelpers.fetch(TemperatureThingAddress).then(async (TD) => {

    try{
        let temperatureThing = await WoT.consume(TD);
```

```
setInterval(async() => {
    let curTemp = await temperatureThing.readProperty("temperature");

    console.log("Room's Current Temperature is ", curTemp);

    if (curTemp < 20) {
        await temperatureThing.invokeAction("increment",4)
    }
}, 1000);
```

```
temperatureThing.subscribeEvent("overheat",
x => console.log("!!!CONTACT THE FIRE DEPARTMENT!!!!"),
e => console.error("Error: %s", e),
() => console.info("Completed")
);
```

How to write code to interact with a Thing

You can use the same configuration file.

The full code is available at:

https://gist.github.com/egekorkan/dfd0f999c22396a997eb10994e11aed6

Let’s get a bit more advanced ;)

How to use as an npm dependency

If you are building anything more complex than our examples, we recommend you to use it as an npm dependency and import into your project code via require statements such as:

```
Servient = require("@node-wot/core").Servient
```

This also means that you would **not** use the `wot-servient` command but build the WoT object yourself!

How to use as an npm dependency

Same with any npm package, you should download the required packages from npm. You should run the following in your project's folder:

`npm install @node-wot/core` (mandatory core component)

`npm install @node-wot/binding-coap` (optional bindings)

Let's see how the previous code look like:

```
Servient = require("@node-wot/core").Servient
HttpServer = require("@node-wot/binding-http").HttpServer

Helpers = require("@node-wot/core").Helpers

// create Servient add HTTP binding with port configuration
let servient = new Servient();
servient.addServer(new HttpServer({
    port: 8081 // (default 8080)
}));

servient.start().then((WoT) => {
    WoT.produce({
        title: "TemperatureController",
    description: "A Thing to control the temperature of the room and also get alerts in too high
temperatures",
    properties: {
        temperature: {
            type: "integer",
            description: "Current temperature value",
            observable: true,
            readOnly: true,
                                    :
                                    :
```

Exactly same as before!

```
Servient = require("@node-wot/core").Servient
HttpClientFactory = require("@node-wot/binding-http").HttpClientFactory

Helpers = require("@node-wot/core").Helpers

// create Servient and add HTTP binding
let servient = new Servient();
servient.addClientFactory(new HttpClientFactory(null));

let wotHelper = new Helpers(servient);

const TemperatureThingAddress = "http://localhost:8080/TemperatureController";
wotHelper.fetch(TemperatureThingAddress).then(async (td) => {
    // using await for serial execution (note 'async' in then() of fetch())
    try {
        servient.start().then((WoT) => {
            WoT.consume(td).then((thing) => {
                // read a property "string" and print the value
                setInterval(async() => {
                  let curTemp = await temperatureThing.readProperty("temperature");

                  console.log("Room's Current Temperature is ", curTemp);

                  if (curTemp < 20) {
                      await temperatureThing.invokeAction("increment",4)
```

Exactly same as before!

How to use as an npm dependency

Advantages:

- Better version control of node-wot
- More suitable for installing on more constrained devices, like Raspberry Pi
- Install only what your project needs
- Better compatibility when combined with other npm packages
- Self-contained
- Possibility to use Typescript and Intellisense

This way is required for the Deliverable 2

What is next?

- Deliverable 2 is about writing Consumer applications.
- First, you will be interacting with virtual devices/instances
- Secondly, only with simpler devices
- Robots in Deliverable 3

- Available test instances of node-wot as of now:
  - http://plugfest.thingweb.io:8083/TestThing
  - http://plugfest.thingweb.io/examples/smart-coffee-machine.html
  - http://plugfest.thingweb.io:8083/counter

Wrap-Up

- Scripting API allows writing code for Things and Consumers in a protocol independent way
- Eclipse Thingweb node-wot is an implementation of Scripting API that we will use
- Start practicing!