# Internet of Things Workshop
## with Netduino Plus

@tamberg, 11.09.2014

# Internet

Computers, connected through Internet protocols

Display or manipulate **documents**

http://blog.com/2011-09-15/todays-post.html

# Internet of Things (IoT)

Computers, **sensors** and **actuators** connected through Internet protocols

Measure or manipulate **physical properties**

http://e-home.com/tamberg/kitchen/light

# Internet-connected devices

John Romkey's Toaster (1990, Ethernet)

Ambient Orb (2002, via pager network)

iPod (2001), iTunes Store (2003, via USB/PC)

Nike+ iPod (2006), Bracelet (2008 via USB/PC)

Rafi Haladjian's Nabaztag (2006, Wifi)

Rob Faludi's Botanicalls (2006, Ethernet)

Schulze&Webb Availabot (2006, via USB/PC)

iPhone (2007, GSM)

Amazon Kindle (2007, 3G)

Wafaa Bilal's Shoot an Iraqi (2007, ?)

Withings BodyScale (2008, Wifi)

Vitality GlowCap (2008, Wifi; 2011, 3G)

BakerTweet (2009, 3G)

Adrian McEwen's Bubblino (2009, Ethernet)

David Bowen's Telepresent Water (2011, ?)

Nest Thermostat (2011, Wifi)

BERG's Little Printer (2011, ?)

Supermechanical's Twine (2012, Wifi)

Olly & Polly (2012, via USB/PC)

Koubachi Sensor (2012, Wifi)

Descriptive Camera (2012, Ethernet)

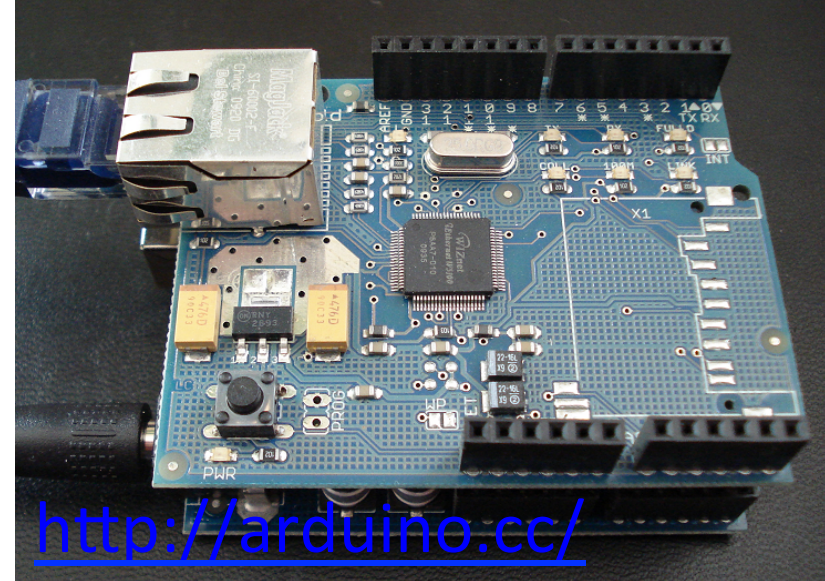# IoT reference model
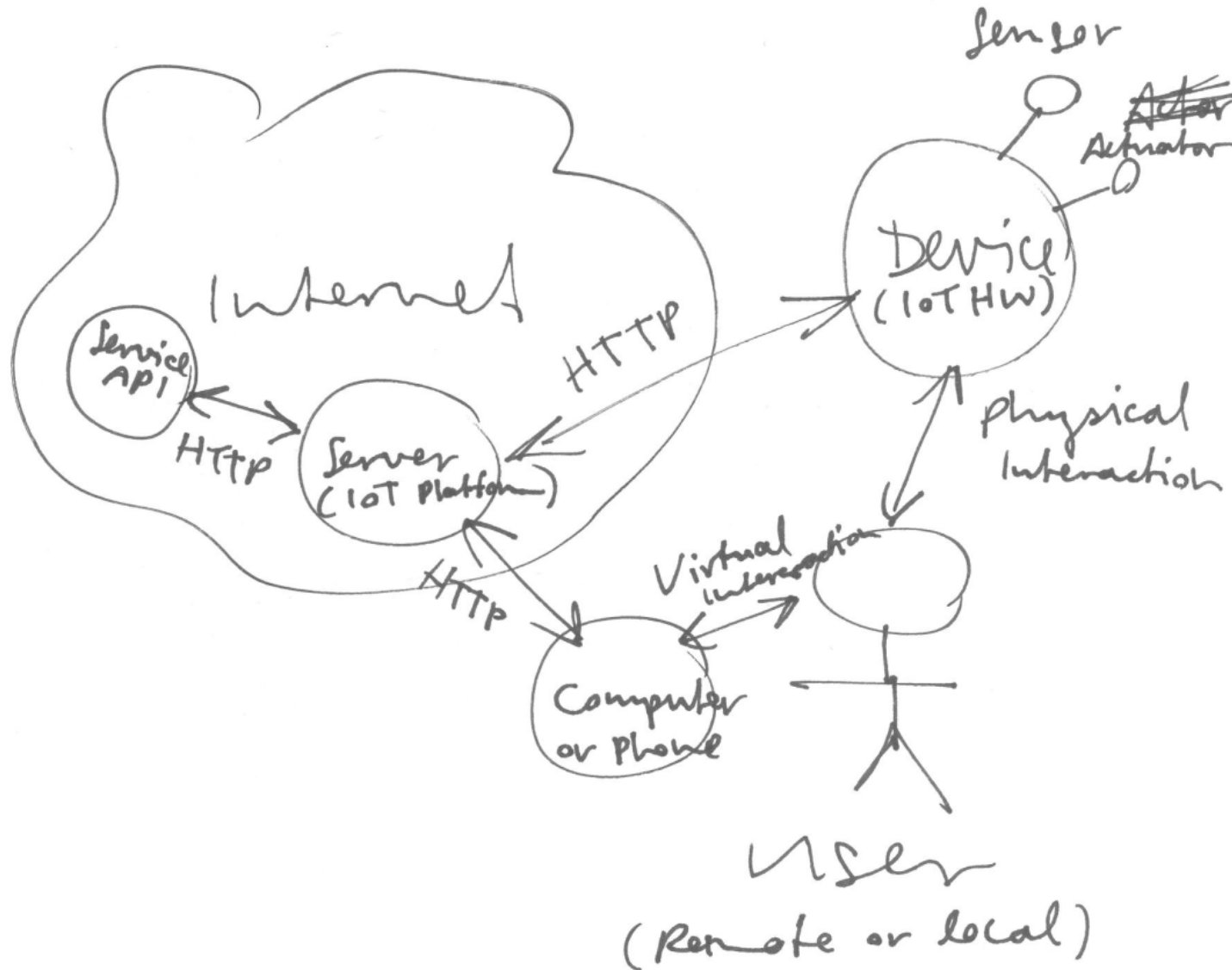
# IoT hardware

Any Internet-connected computer with an **interface to the real world** (sensors, actuators)
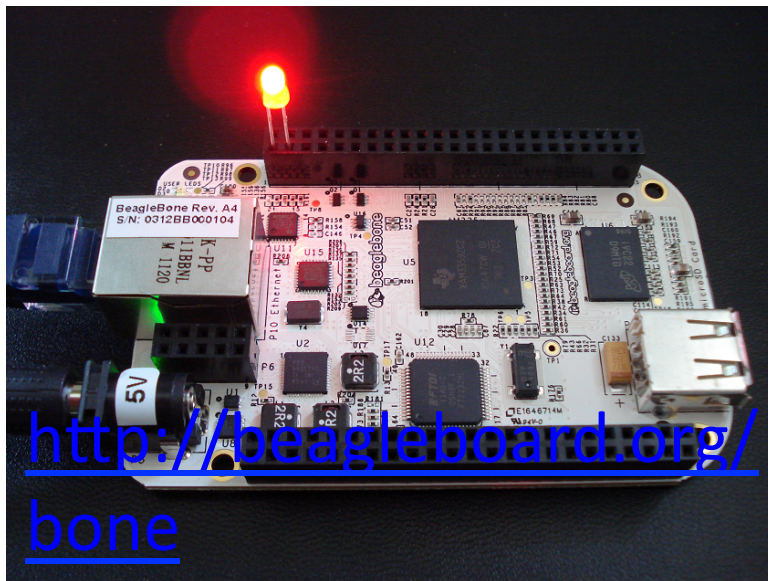
Small => can be **embedded into things**

Small computer = **microcontroller** (or **board**), e.g. Arduino, Netduino Plus, BeagleBone, …

**Note**: connecting your board to the Internet via a desktop PC and USB is also fine, just a bit overkill
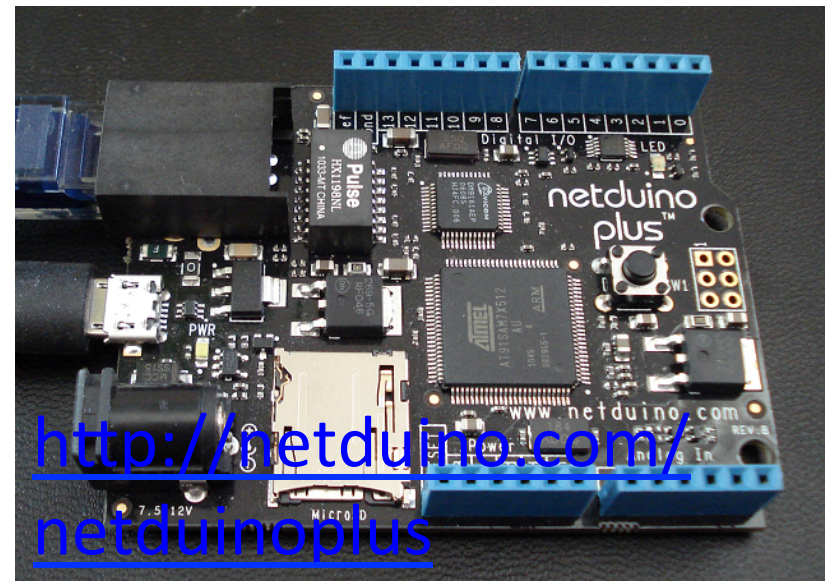
# IoT hardware

**Note**: Thanks to TCP/IP & HTTP, any client can talk to any service, no matter which hardware you choose

# IoT infrastructure services

**Thingspeak** and **Xively** to store and use sensor measurements
e.g. https://thingspeak.com/channels/9

**Twitter** allows objects to talk to humans or receive commands
e.g. @twrbrdg_itself (f.k.a. @towerbridge)

**Yaler** enables remote access to Internet-connected devices
e.g. http://try.yaler.net/~arduino/led  *(Disclosure: I'm a founder)*

**Zapier** and **IFTTT** allow mash-ups of Webservices
e.g. http://goo.gl/7Y8a7z

# Just a beginning

Reactive buildings, flying / crawling IoT devices, underused devices selling themselves on Ebay...

Connected products become service avatars, or "everything becomes a service" (e.g. car sharing, home sharing, shoe sharing)

**"Once it's here it will no longer be called the Internet of Things"** Open IoT Assembly 2012

# Topics of this workshop

**Getting started**
(setup and programming of IoT hardware)

**Measuring** and **manipulating**
(physical computing: sensors and actuators)

**Connecting** your device **to the Internet**
(IoT: monitoring sensors, controlling actuators)

**Mash-ups** with Web-enabled devices

(together, if time permits)

**How the Internet works** under the hood

# Hands on

Broad range of topics => **learn by doing**

**Copy&paste** examples, make 'em work for you,
https://bitbucket.org/tamberg/iotworkshop/get/tip.zip

Focus on **end-to-end results, not details**

**Google**, help each other, ask us

# Getting started

The **IDE** (**I**ntegrated **D**evelopment **E**nvironment) allows you to **program** your board, i.e. "make it do something new"

You **edit** a program on your computer, then **upload** it to your board where it's stored in the program memory (flash) and **executed** in RAM

**Note**: Once it has been programmed, your board can run on its own, without another computer

# **Getting started** with Netduino Plus

To install the Netduino IDE and SDKs, and to connect your Netduino Plus board to your computer via USB, see

MS Visual C# Express IDE (free)

http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-csharp-express

MS .NET Micro Framework v4.1 SDK

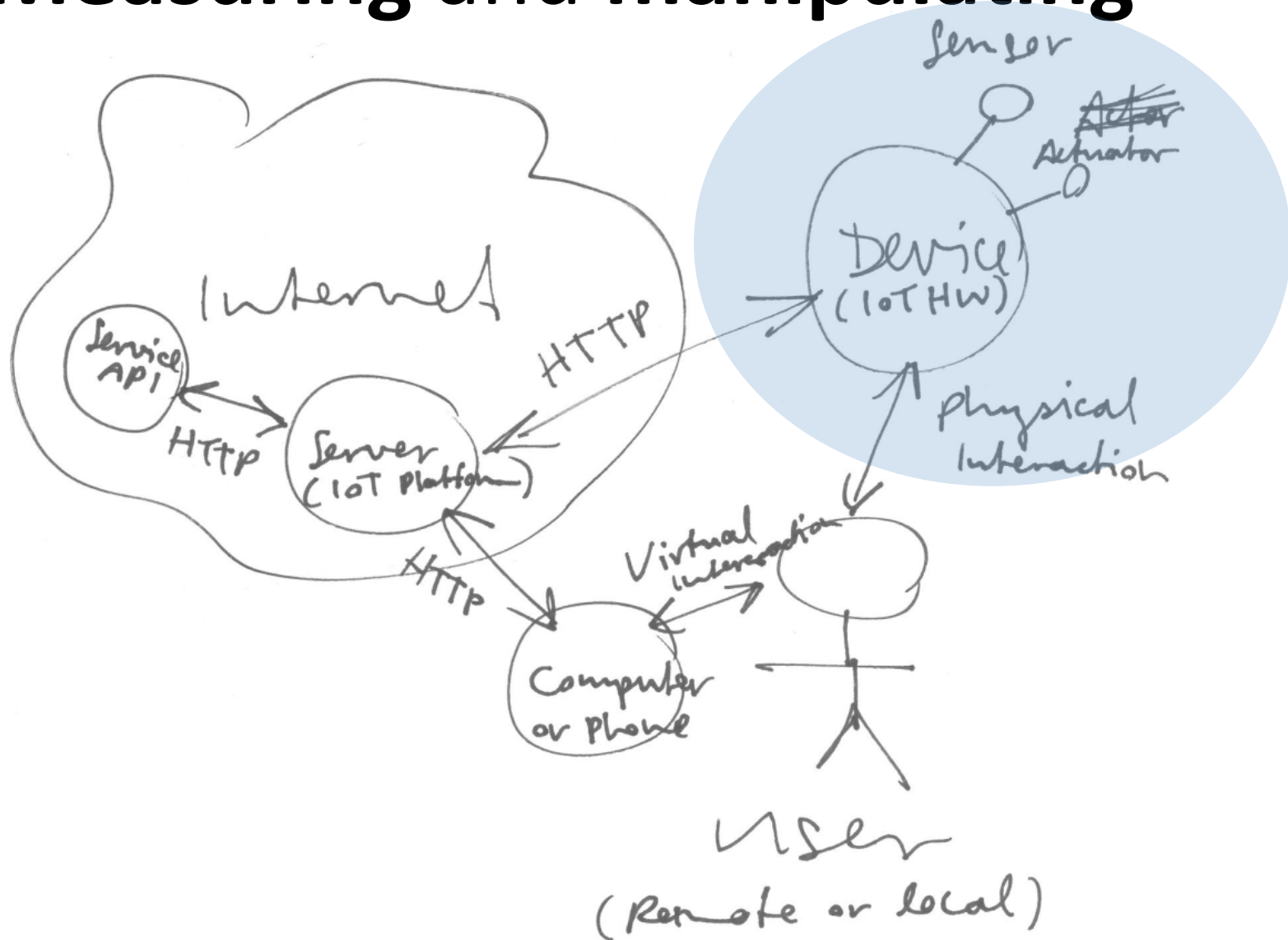http://www.netduino.com/downloads/MicroFrameworkSDK.msi

Netduino v4.1 SDK (for 32-bit / 64-bit Windows)

http://www.netduino.com/downloads/netduinosdk_32bit.exe

http://www.netduino.com/downloads/netduinosdk_64bit.exe

# **Measuring** and **manipulating**

# **Measuring** and **manipulating**

IoT hardware has an **interface to the real world**

**GPIO** (**G**eneral **P**urpose **I**nput/**O**utput) pins

Measure: **read** sensor value from **input** pin

Manipulate: **write** actuator value to **output** pin

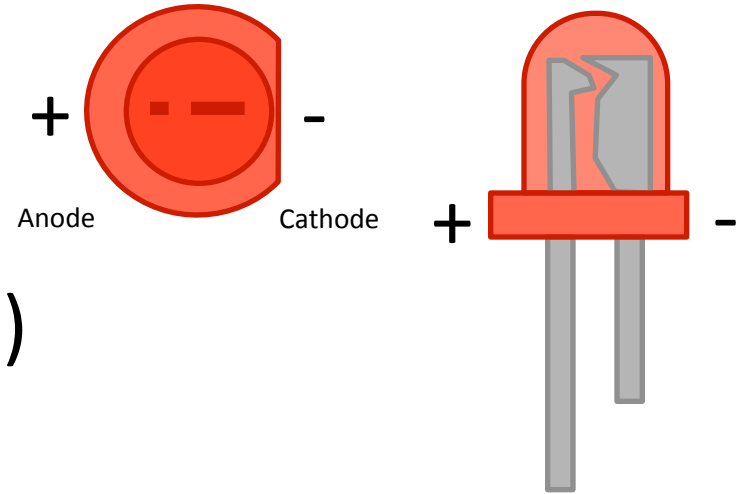Inputs and outputs can be **digital or analog**

# The resistor

Resistors are the **workhorse of electronics**

Resistance is **measured in Ω** (Ohm) and adds up in series; a resistors orientation doesn't matter

A resistors Ω value is **color-coded** right on it

**Note**: color codes are great, but it's easier to use a multi-meter if you've got one, and just measure Ω

# The LED

The **LED** (**L**ight **E**mitting **D**iode)
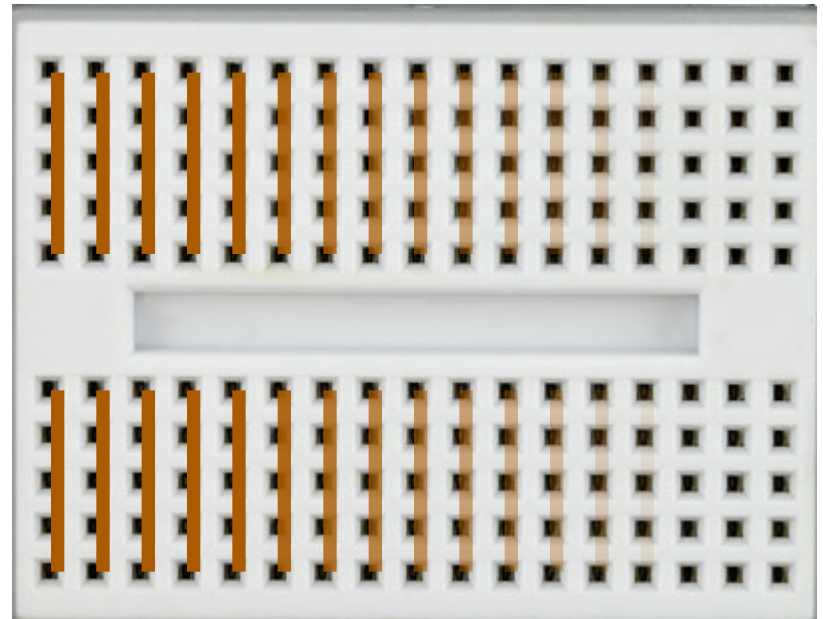
is a simple, digital **actuator**

LEDs have a **short leg** (**-**) and a **long leg** (**+**)

and it matters how they are oriented in a circuit

To prevent damage, LEDs are used together with
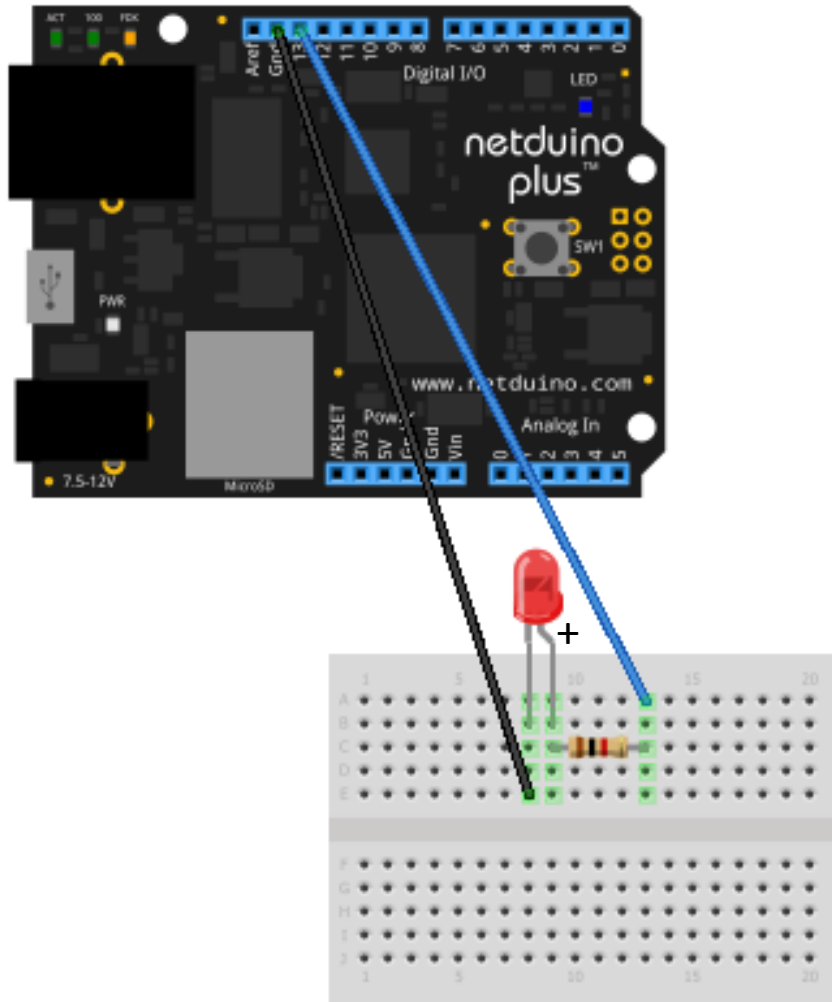a 1KΩ **resistor** (or anything from 300Ω to 2KΩ)

# The breadboard

A breadboard lets you wire electronic components without any soldering

Its holes are connected "under the hood" as shown here

# **Wiring a LED** with Netduino Plus



**Note**: the additional **1K Ω** resistor should be used to prevent damage to the pins / LED if it's reversed

The long leg of the LED is connected to **pin 13**, the short leg to ground (**GND**)

Made with ▢ Fritzing.org

# **Digital output** with Netduino Plus

```csharp
using System.Threading;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

public class Program {
  public static void Main() {

    var ledPort = new OutputPort(Pins.GPIO_D13, false);

    while (true) { // loop in C#

      ledPort.Write(true);

      Thread.Sleep(500); // wait 500ms

      ledPort.Write(false);

      Thread.Sleep(500);

    }
  }
}
```
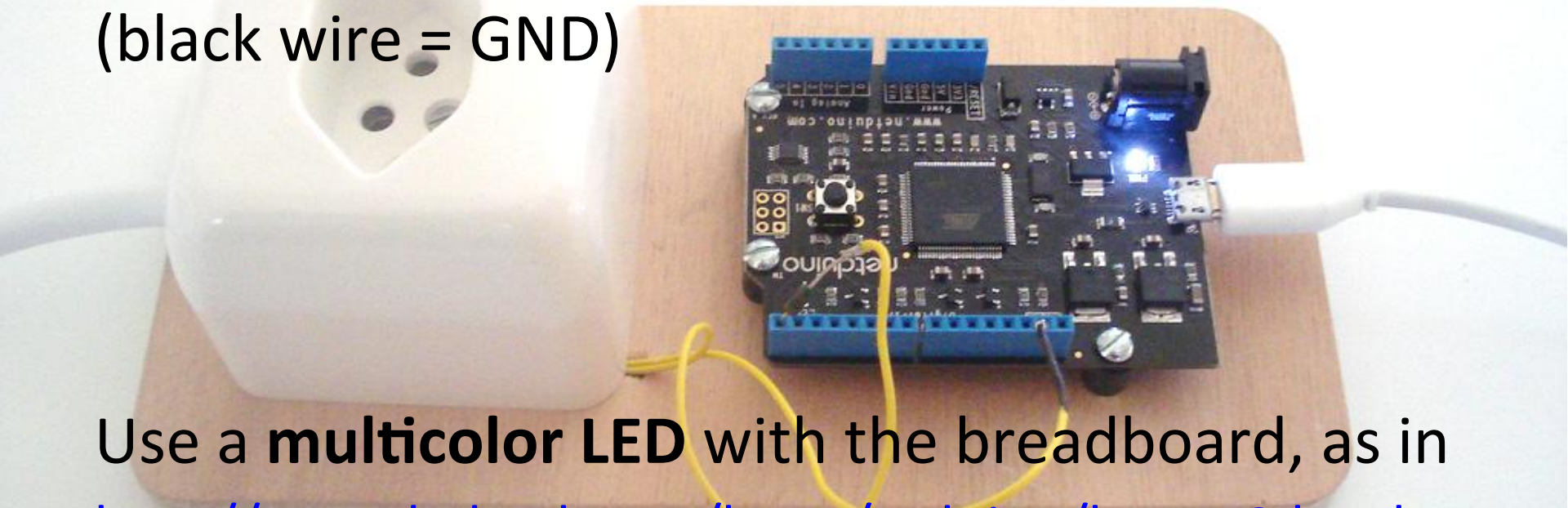
**Note**: blinking a LED is the *Hello World* of embedded software

Set *ledPort* as wired in your LED circuit

**true** = digital 1 (5V) means LED is **on**, **false** = digital 0 (0V) means LED is **off**

# Actuator bonus stage
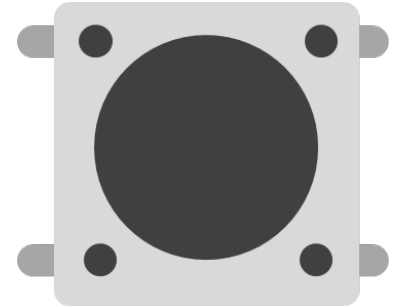
Try a **switched power outlet** instead of a LED
(black wire = GND)

Use a **multicolor LED** with the breadboard, as in
http://www.ladyada.net/learn/arduino/lesson3.html

Or **solder** resistors to a multicolor LED, as in
http://www.instructables.com/id/Arduino-Web-LED/

# The switch

A switch is a simple, digital **sensor**

Switches come in different forms, but all of them in some way **open** or **close** a gap in a wire

The **pushbutton** switch has four legs for easier mounting, but only two of them are needed

**Note**: you can also easily build your own switches, for inspiration see e.g. http://vimeo.com/2286673

# **Wiring a switch** with Netduino Plus



Made with ⚡ Fritzing.org
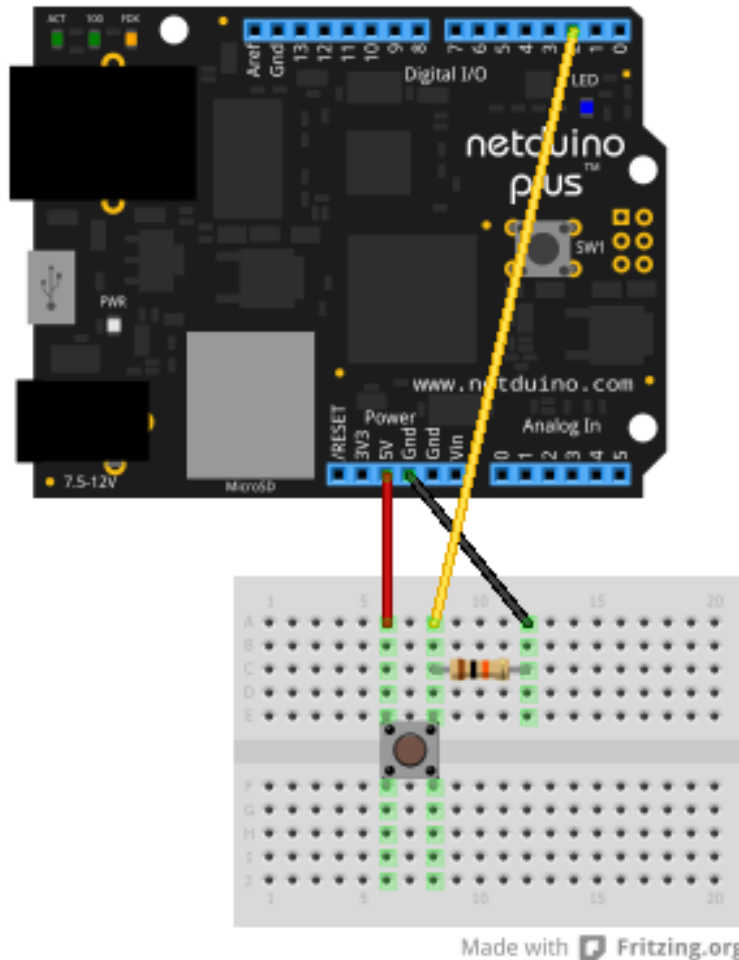
**Note**: the resistor in this setup is called *pull-down* 'cause it pulls the pin voltage down to GND (0V) if the switch is open

Pushbutton **switch**
**10K Ω** resistor
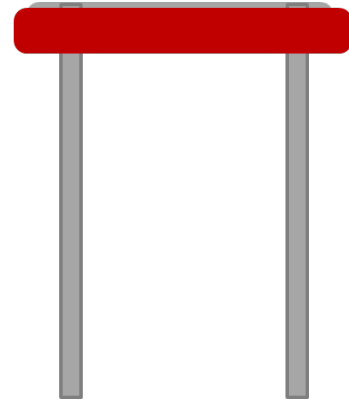**5V**
**GND**
**D2** (max input 5V!)

# Digital input with Netduino Plus

```csharp
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

class Program {
  static void Main () {

        var sensorPort = new InputPort( // e.g. button switch
            Pins.GPIO_D2, false, Port.ResistorMode.Disabled);
        while (true) { // loop in C#
            bool sensorValue = sensorPort.Read();
            Debug.Write(sensorValue); // log true or false
        }
  }
}
```

Check the output window in VS to see the log output

# **Photoresistor (LDR)**

A photoresistor or **LDR**
(**l**ight **d**ependent **r**esistor) is
a resistor whose resistance
depends on light intensity

An LDR can be used as a simple, **analog sensor**

The orientation of an LDR does not matter

# **Wiring an LDR** with Netduino Plus



Made with **Fritzing.org**
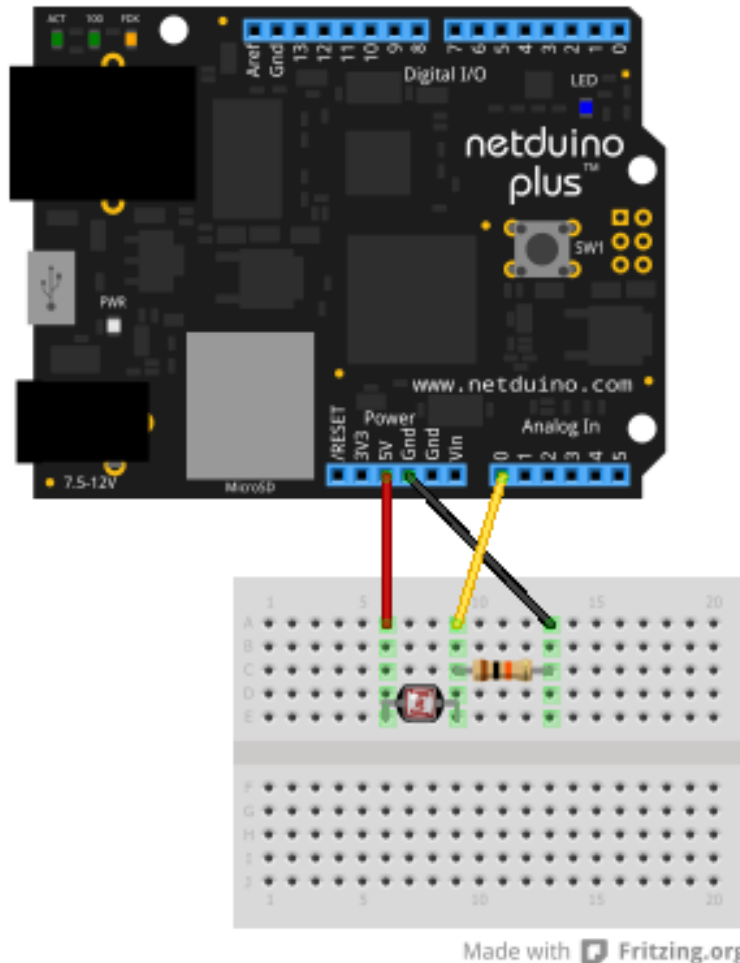
**Note**: this setup is a *voltage-divider,* as the 5V total voltage is divided between LDR and resistor to keep 0V < **A0** < 2.5V
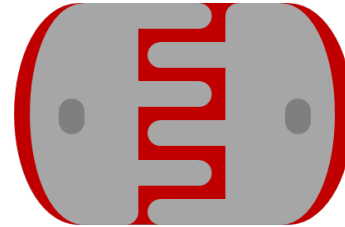
Photoresistor (LDR)
**10K** Ω resistor
**5V**
**GND**
**A0**

# **Analog input** with Netduino Plus

```csharp
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

public class Program {
    public static void Main () {
        var sensorPort = new AnalogInput(Pins.GPIO_PIN_A0);
        while (true) {
            int sensorValue = sensorPort.Read();
            Debug.Print(sensorValue);
        }
    }
}
```

Check the output window in VS to see the log output

**Note**: use e.g. Excel to visualize values over time

# Sensor bonus stage

Switch the LED depending on analog input
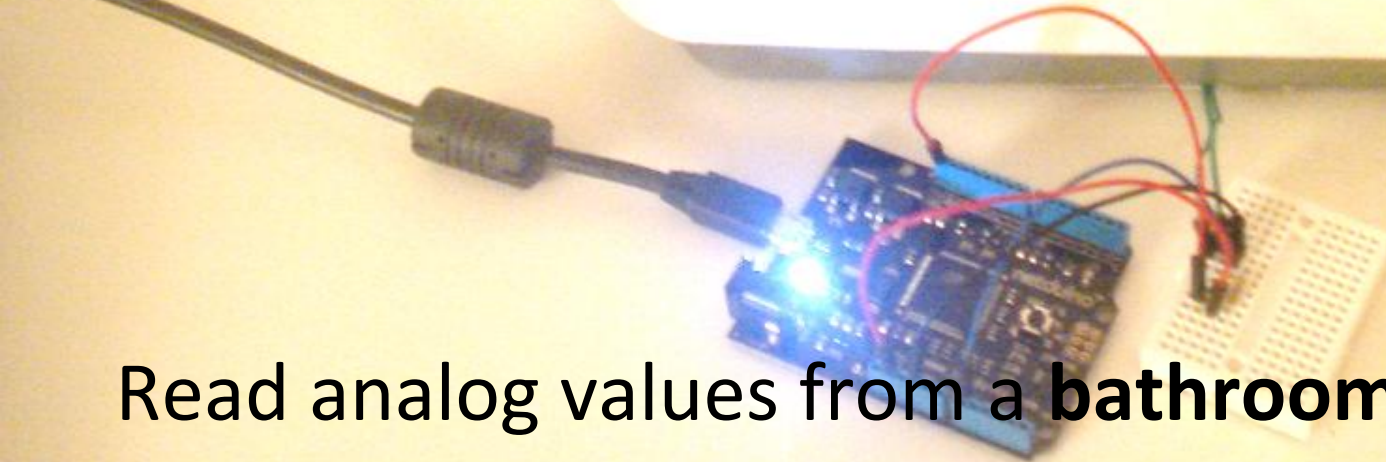
Read analog values from a **bathroom scale**

Or use sensors with other wire protocols, e.g. i2c

# **Connecting** to the Internet
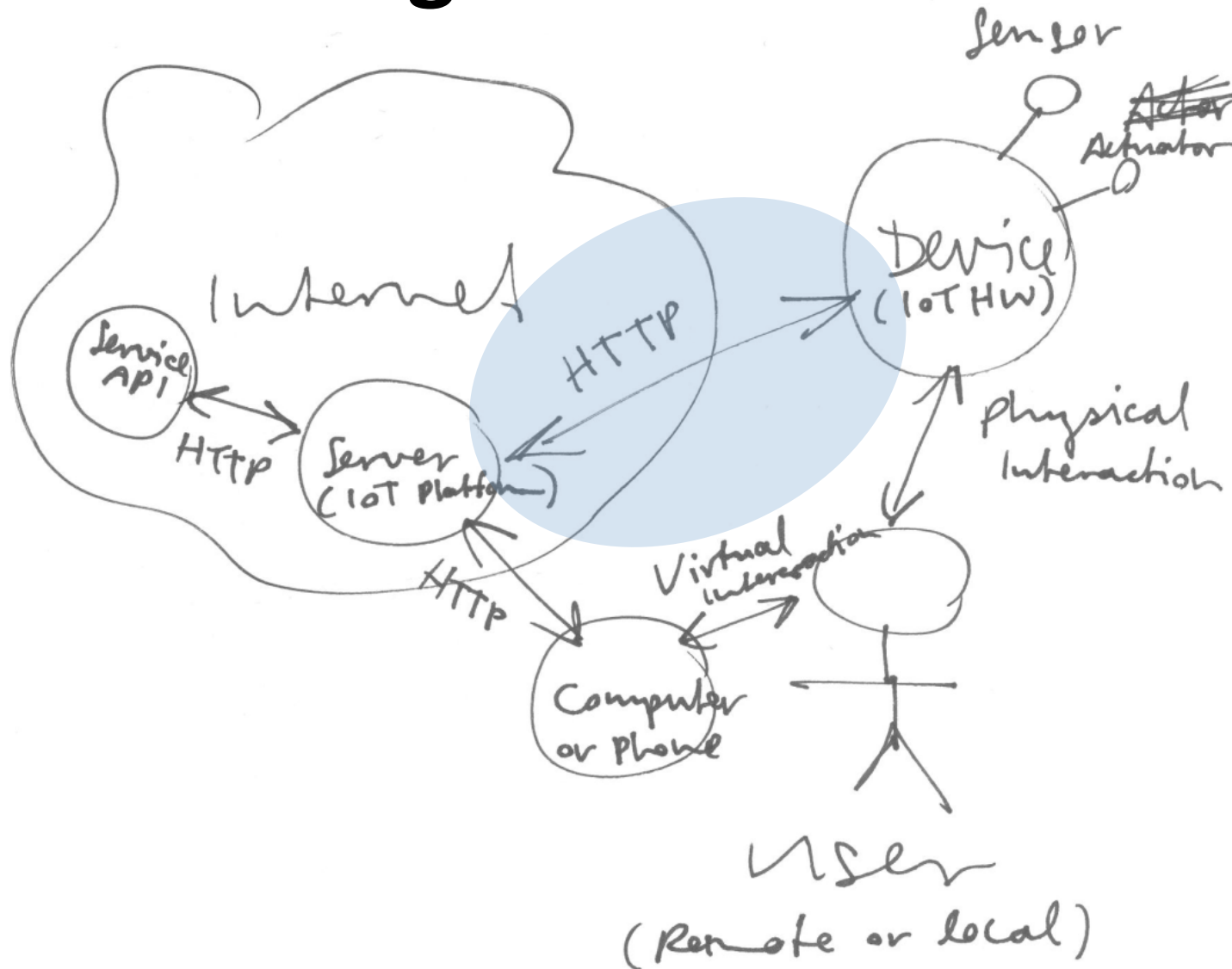
# **Connecting** to the Internet

**Ethernet** (built-in or shield), plug it in anywhere

**Wi-Fi** (module), configured once per location

**3G** (module), configured once, easy to use


**Bluetooth/BLE** (module), via 3G/Wi-Fi of phone

**ZigBee** (module), via ZigBee gateway

**USB** (built-in), via desktop computer

**Note**: in this workshop we focus on Ethernet and Wi-Fi

# **Monitoring** sensors
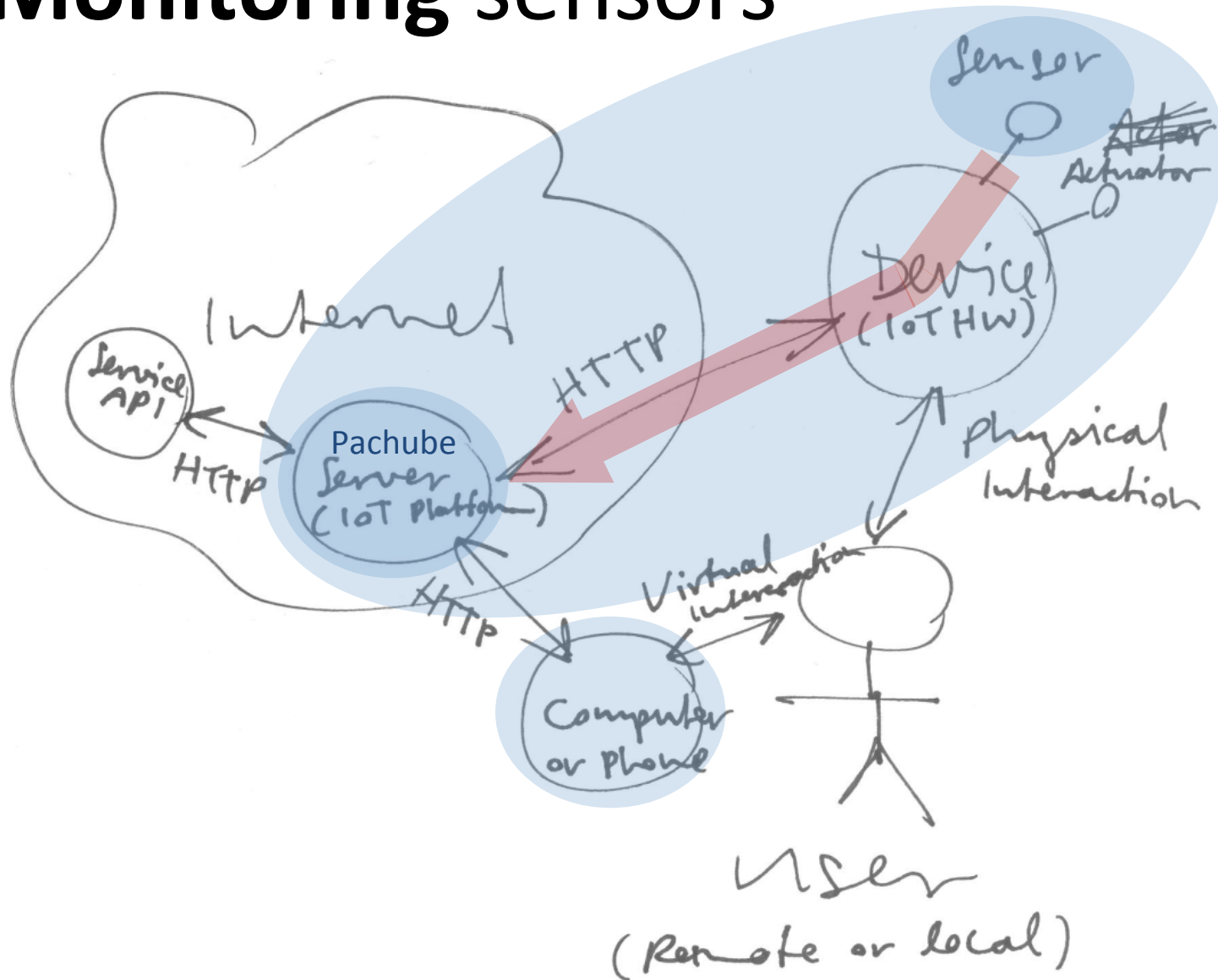
# **Monitoring** sensors

Devices read (and cache) **sensor** data

Devices **push** data to a service with POST, PUT

Some services **pull** data from devices with GET

Service stores measurements, to be consumed by humans or computers (incl. other devices)

# Pachube (now **Xively**)

The Pachube (now Xively) service lets you store, **monitor** and share **sensor data** in open formats

**PUT /v2/feeds/<*your feed id*>.csv** HTTP/1.1\r\n

Host: api.xively.com\r\n

X-ApiKey: *<your API key>*\r\n

Content-Length: *<content length>*\r\n

\r\n

**<sensor name>,<sensor value>**

**GET /v2/feeds/<feed id>.json** HTTP/1.1\r\n

Host and X-ApiKey as above...\r\n\r\n

**Note**: please visit http://xively.com/ to sign up, create a feed with a data stream per sensor and get an API key

# **Pachube** with Netduino Plus

[http://www.gsiot.info/download/](http://www.gsiot.info/download/) *> Projects V20110526 Version 0.9 (beta release).zip for NetduinoPlus > **HelloPachube.sln***

Insert your *feedId, apiKey* and a data stream name (e.g. "voltage")

Analog input: **LDR** on **A0**
http://xively.com/feeds/<feed-id>

**Note**: as you send data to xively.com the Netduino Plus has to be hooked up to the Internet

Check the output window in VS to see the log output

# **Controlling** actuators
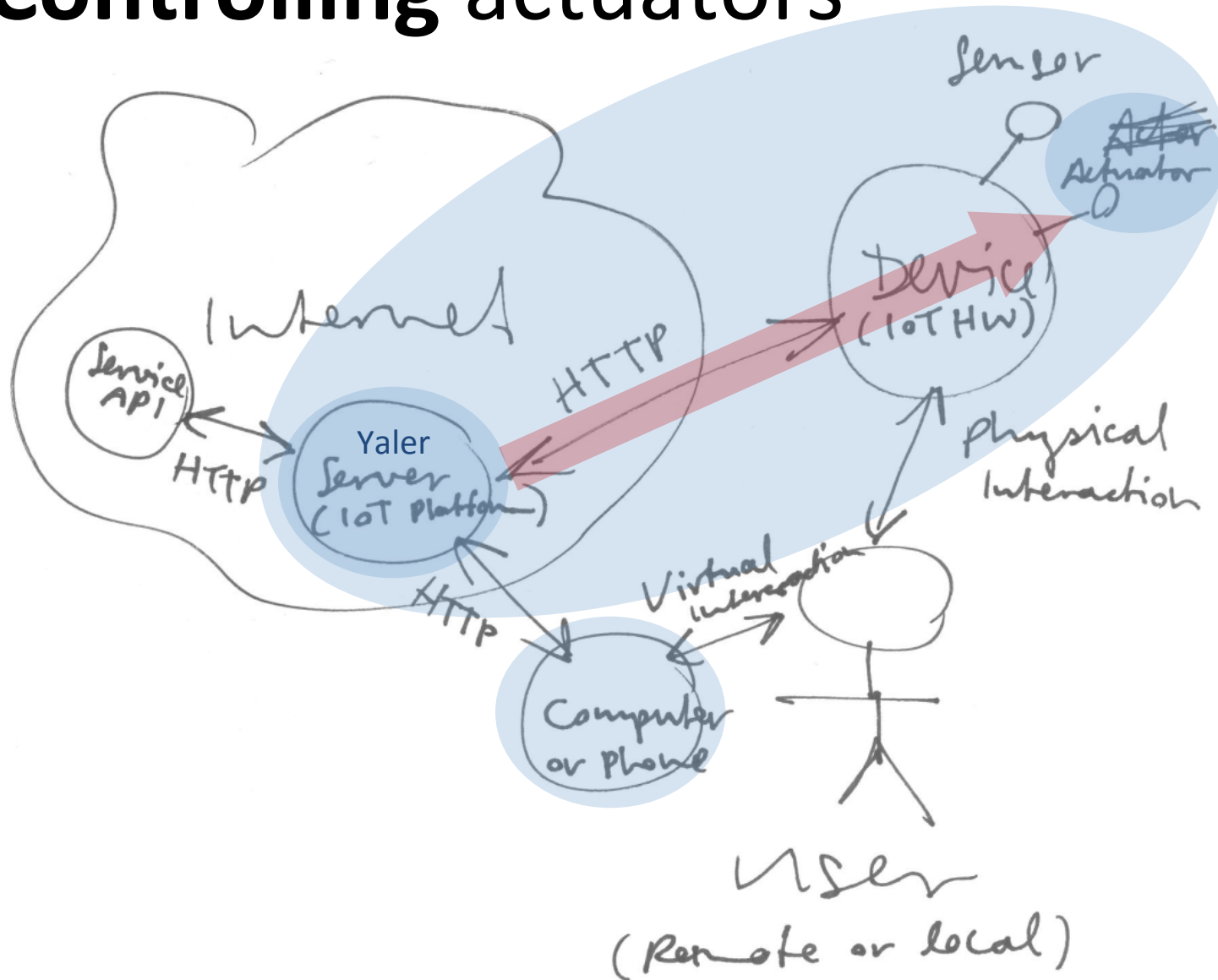
# **Controlling** actuators

Service offers **UI** or **API** to control actuators

Device **poll**s service for control data with GET

Or, service **push**es control data to device with POST or PUT

Device writes control data to **actuators**

# **Web service** with Netduino Plus

[http://www.gsiot.info/download/](http://www.gsiot.info/download/) *> Projects V20110526 Version 0.9 (beta release).zip for NetduinoPlus >* **LedControllerHtml.sln**

# Yaler

The **Yaler** relay provides a public and stable URI to access Web services behind a firewall or NAT
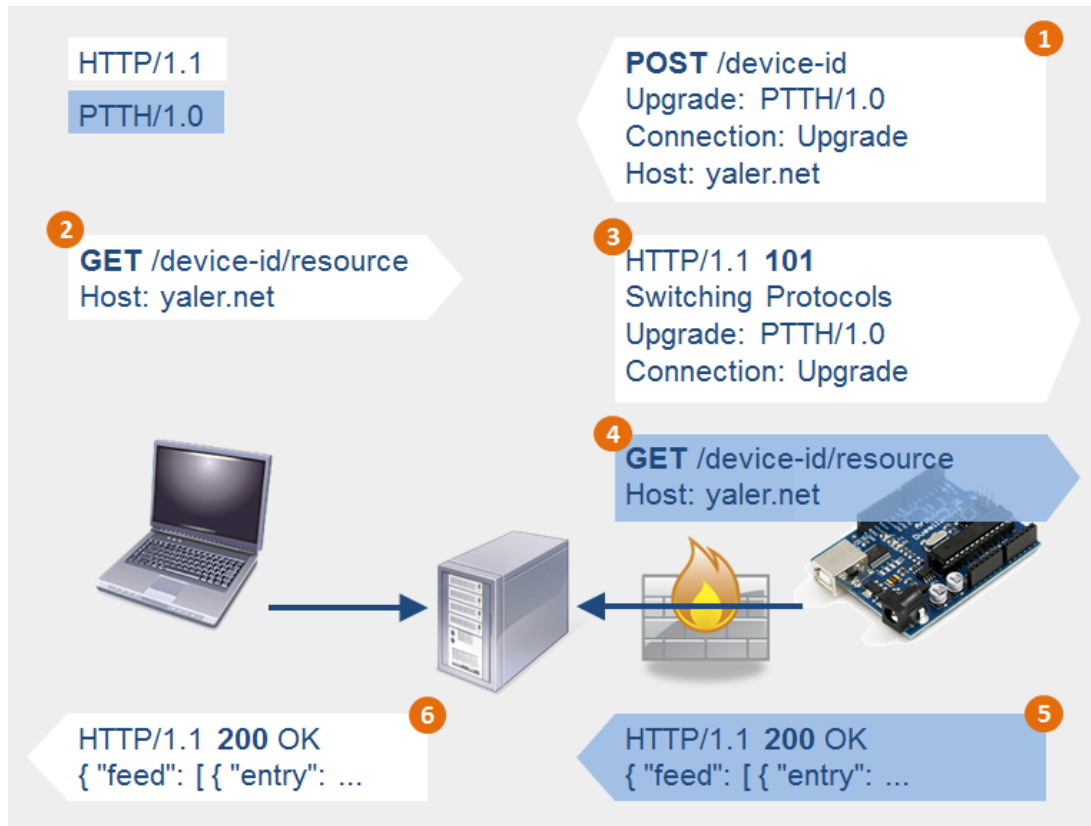
# **Yaler** with Netduino Plus

To set up remote Web access, follow
[https://yaler.net/netduino](https://yaler.net/netduino)
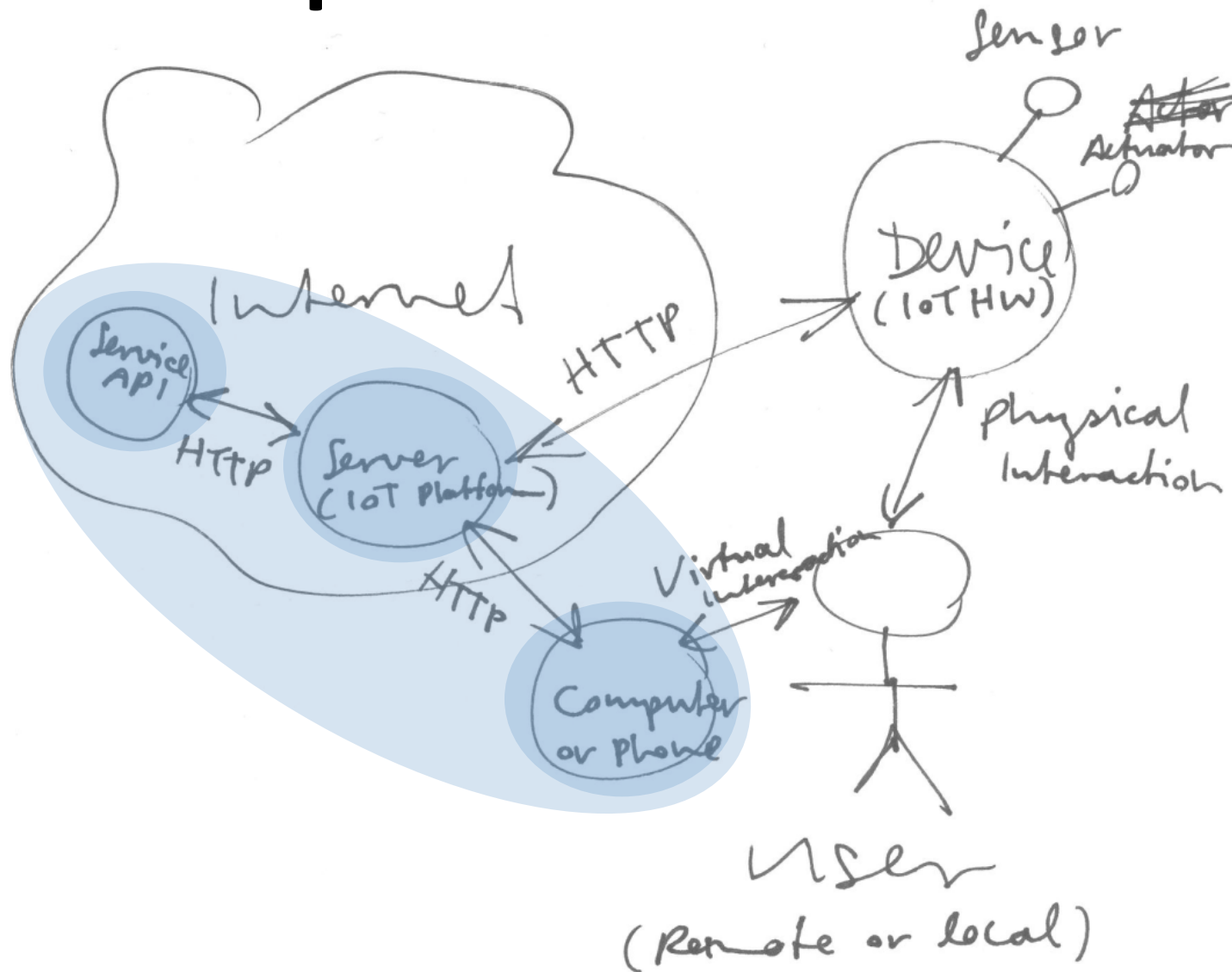
# Mash-ups

# Mash-ups

A **mash-up** combines two or more Web services

Once **devices** have APIs, they **become scriptable**

**Logic moves** out of device, **into the Cloud**, e.g.
Web-enabled LED + Yahoo Weather API =
ambient weather notification

**Note**: the IoT enables physical mash-ups of things

# Mash-ups

HTML combining data from multiple APIs **on the Web client**, using **Javascript** XMLHttpRequest to get data (in **JSONP**, to bypass same origin policy)

**Scripting** (C#, Python, Go, …) glue code hosted on a **desktop** or **in the cloud** (EC2, AppEngine …)

**Mash-up platforms** (IFTTT.com, Zapier.com, …)

**Note**: open data formats and APIs enable mash-ups

# How the Internet works

If you wonder what TCP/IP, HTTP or DNS means - or care about the difference between protocol, data format and API, read on…
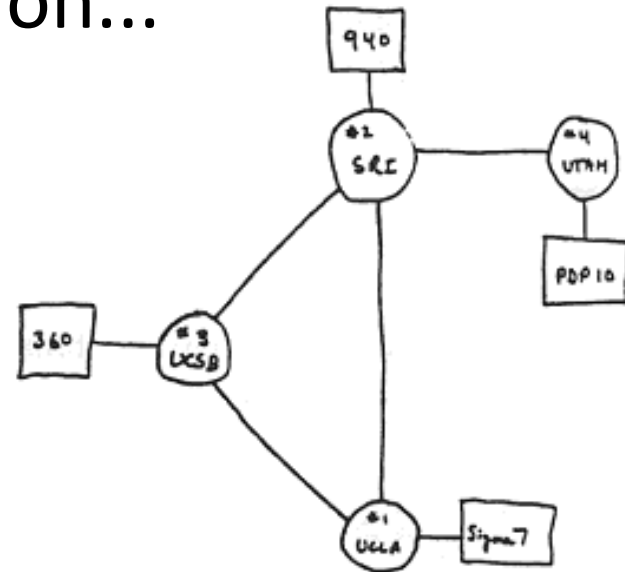


THE ARPA NETWORK

DEC 1969

# Protocols

Parties need to agree on **how to exchange** data (communicating = exchanging data according to a protocol)

e.g. **Ethernet** links local computers physically, **TCP/IP** is the foundation of the **Internet**, and **HTTP** is the protocol that enables the **Web**

**Note**: protocols are layered, e.g. HTTP messages transported in TCP/IP packets sent over Ethernet

# TCP/IP

**IP** (**I**nternet **P**rotocol) deals with host addressing (each host has an **IP address**) and packet routing

**TCP** (**T**ransmission **C**ontrol **P**rotocol): connection oriented, reliable data stream (**packets** in-order, errors corrected, duplicates removed, discarded or lost packets resent) from client to server

**Note**: *DHCP* assigns an *IP address* to your device which is mapped to the device's *MAC address*

# HTTP

**HTTP** (**H**yper**t**ext **T**ransfer **P**rotocol) enables the distributed, collaborative system we call the **Web**

The **client** sends an HTTP **request**, the **server** replies with a **response**

HTTP **Message** = Request|Response

**Request** = (GET|POST|…) Path CRLF *(Header CRLF) CRLF Body

**Response** = "HTTP/1.1" (200|404|…) CRLF *(Header CRLF) CRLF Body

**CRLF** = "\r\n"

(Read the spec: http://tools.ietf.org/html/rfc2616)



Browser (Client)

Google (Server)

GET /search?q=IoT
Host: www.google.com

HTTP/1.1 200 OK
Content-Length: …
HTML Content…
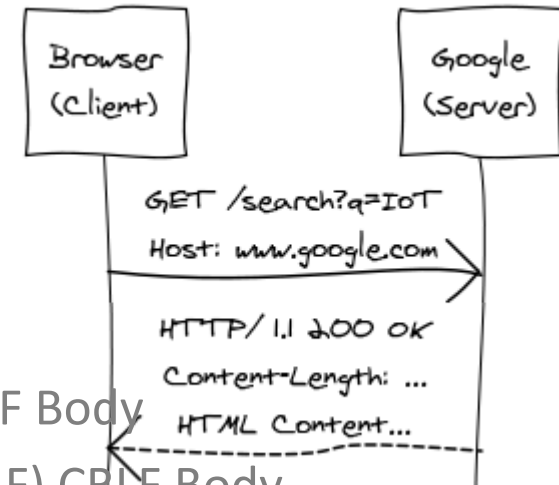
**Note**: HTTP is human readable, i.e. it's easy to debug

# URIs

The **URI** (**U**niform **R**esource **I**dentifier) is a string of characters used to identify a resource

http://blog.tamberg.org/2011-10-17/side-projects.html

scheme     authority = host [':' port]           path

(Read the spec: http://tools.ietf.org/html/rfc3986)

*QR codes*, *NFC tags* can contain a machine readable URI

**IoT**: URIs can refer to things or their physical properties

**Note**: good URIs can be hand-written on a napkin and re-typed elsewhere, without any ambiguity

# DNS

**DNS** (**D**omain **N**ame **S**ystem) maps Internet domain names to one or more IP addresses

Try it in your desktop computer terminal, e.g.

*$ nslookup **google.com***

*173.194.35.6 …*

**Note**: if your device doesn't support DNS you can connect to the server's IP, but beware of changes

# Data formats

Parties need to agree on **what is valid** content (parsing = reading individual content tokens)

CSV: easy to parse, suited for tables, old school
**JSON**: easy to parse, de facto standard
XML: used by many services, W3C standard
Semi-structured text, e.g. Twitter's @user, #tag
**Binary** formats, e.g. PNG, MP3, …

# RSS

In addition to generic data formats like CSV, JSON, XML there are refinements that **add semantics** to the document

**RSS** (or **Atom**) is a data format for **lists** of **items**

Invented for blogs, RSS is great for **data feeds**

**Note**: RSS documents are also XML documents, but not all XML documents contain valid RSS

# HTML

**HTML** (**H**yper**t**ext **M**arkup **L**anguage) is a data format describing how a Web page should be structured and displayed

Look at the HTML (and Javascript) code of any Web page with "**view source**" in your browser

**Note**: HTML documents are not always valid XML documents, but Web browsers are very forgiving

# APIs

An **API** (**A**pplication **P**rogramming **I**nterface), is an agreement between clients and providers of a service on **how to access a service**, how to **get data out** of it or **put data into it**

The **UI** (User Interface) of a service is made **for humans**, the **API** is made **for other computers**

# REST

**REST** (**Re**presentational **S**tate **T**ransfer) is a style of designing an API so that it is easy to use

REST APIs use **HTTP methods** (GET, PUT, POST, DELETE) to let you perform actions on **resources**

REST APIs can be explored by following links

**Note**: good Web UIs are often built following the same principles, therefore REST APIs feel natural

# Sharing network connections

Most newer computer operating systems allow sharing network connections with other devices

**Mac OSX**: *System Preferences > Sharing > Internet Sharing >* From *Wi-Fi* to *Ethernet*

**Windows 7**: *Control Panel > View network status and tasks > Change adapter settings >* right click *Wireless Network Connection > Properties > Sharing > [x] Allow other network users to connect ... > Local Area Connection*

**Note**: helpful for demos, if there's Wi-Fi but no LAN

# Debugging Web services

**Chrome** > *Inspect Element > Network, Console*

**cURL** for HTTP requests (http://curl.haxx.se/)

**Requestbin** for Webhooks (http://requestb.in/)

Fiddler (http://www.fiddler2.com/)

WireShark (http://www.wireshark.org/)

# Debugging USB or Bluetooth

On Mac OSX and Linux

list connected devices with *ls /dev/tty\**

display output with *screen /dev/tty... 9600*


On Windows

list devices, fix drivers with *devmgmt.msc*

display serial output with **PuTTY**

# Energy

Wall socket, Power over Ethernet (w/ adapters), batteries (direct or Minty Boost USB charger), LiPo batteries (also shields), solar panels, …

Low power: lets hardware sleep to save energy

Future: new battery technologies, ultra low power hardware, energy harvesting

**Note**: Moore's law does not apply to batteries

# Learning more

**Electronics**: Ohm's law, Kirchhoff's current and voltage law (KCL & KVL), *Make: Electronics* by Charles Platt

**Interaction Design**: *Smart Things* by Mike Kuniavsky, *Emoticomp* blog post by Ben Bashford, BERG blog

**Physical Computing**: *Making Things Talk* by Tom Igoe

**REST**: *RESTful Web Services* by Leonard Richardson

**Programming**: read other people's code

**IoT**: *Designing the Internet of Things* by Adrian McEwen and Hakim Cassimally, Postscapes.com, IoTList.co

**Note**: MechArtLab Zürich has an OpenLab on Tuesdays

# Reducing E-waste

Tired of hacking?

Donate your hardware to a local hackerspace…

e.g. MechArtLab

Hohlstrasse 52

8004 Zürich

# DIY IOT FTW

Thank you