The **Web of Things** and **Hypermedia for Smart Devices**

Summer School “AI for Industry 4.0”

27.-31.07.2020
Global (Saint-Étienne / Nuremberg / St. Gallen)

The Web of Things and Hypermedia for Smart Devices

In this lecture, we explore using the Web to **boost application-layer interoperability of Internet of Things devices and services** and as a foundation to **facilitate their usage by clients**.

We discuss principles of the Web of Things, focusing on the design of Web APIs for smart devices and on the usage of hypermedia as the engine of application state.

Sources:

S. Mayer & A. Ciortea: Ubiquitous Computing and the Internet of Things, University of St.Gallen, 2020
F. Mattern and S. Mayer: Ubiquitous Computing, ETH Zurich, 2019

Learning Goals

The goal of this lecture is to enable you to:

- **understand** how the Web supports application-level interoperability in the IoT.
- **build** better Web APIs for devices (and other resources).
- **appreciate** the role that hypermedia plays in guiding Web clients.

In the context of the summer school, this should enable you to **perceive hypermedia-based information systems as playgrounds for interacting humans and machines.**

Reinforcing the previous talks by Matthias and Andreas and together with technologies from the **Semantic Web (Tobias)** and from **Multi-Agent Oriented Programming (Jomi)**, this will supply you with a toolset to build **«ideally interoperable» industrial Internet of Things systems.**

Our Menu

From the IoT to the WoT: Network-level and Application-level Interoperability
Discussion: Good Practices for Web APIs
Tidbits from the Web Architecture
Hypermedia APIs

Our Menu

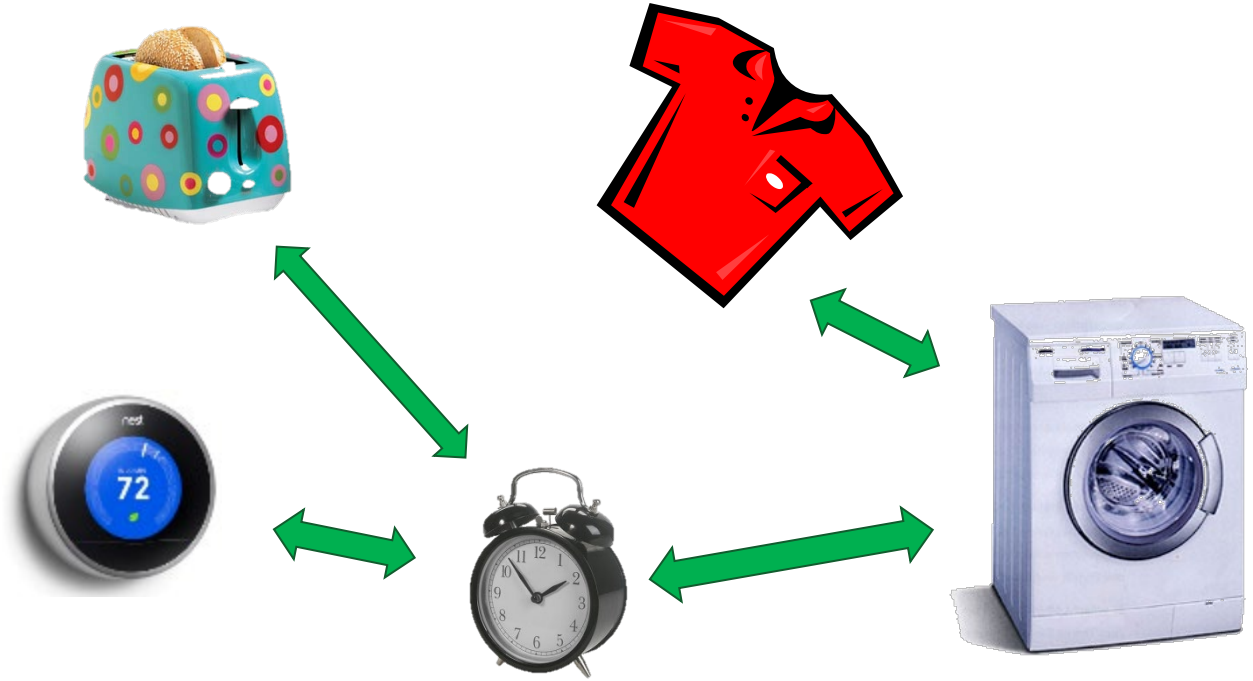
From the IoT to the WoT: Network-level and Application-level Interoperability

Discussion: Good Practices for Web APIs

Tidbits from the Web Architecture

Hypermedia APIs

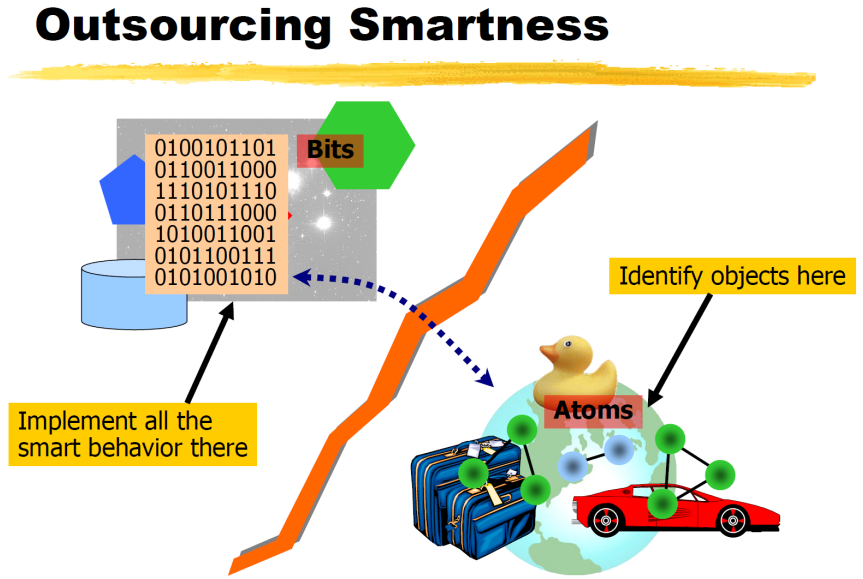
Ubiquitous Computing



The “Social Web of Things” (Ericsson, design study)

- Note the **explicit communication** between David and his home
- Note the **hidden communication** between his smart devices
- Note the **partially autonomous** behavior of the smart environment

Ubiquitous Computing

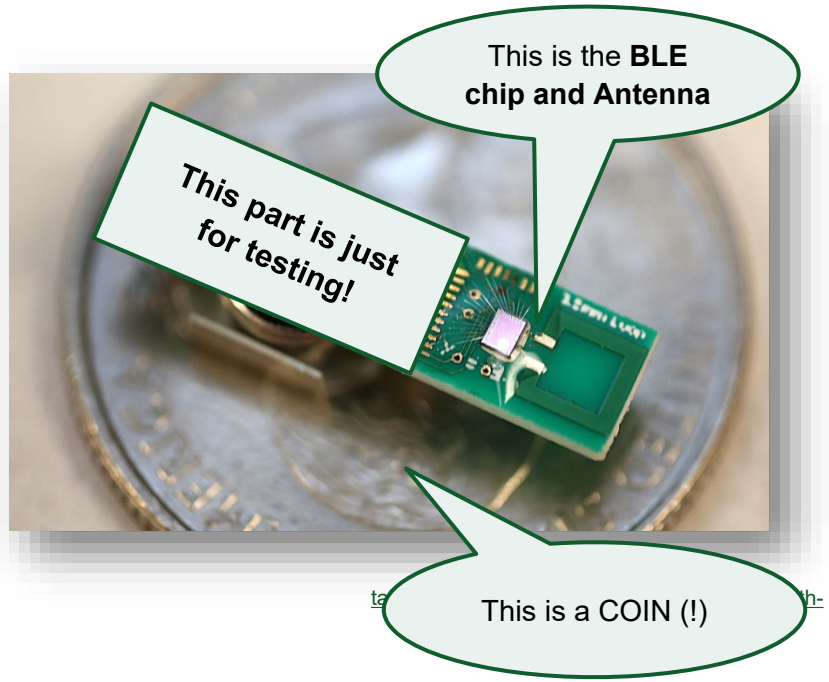


Wireless (and cheap) access to the Internet

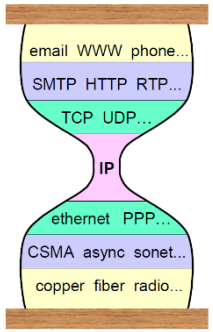
- For **any thing** (→ low-power, passive)
- At **any place** (→ mobile, ubiquitous)
- **Without configuration** ("plug & play" → "arrive & operate")

Object communication **dominates** network usage

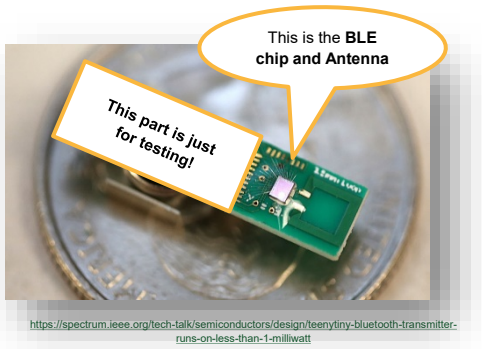
- **Real-world objects** will access and use services on the Web and services that are provided by other nearby devices
- They might even integrate with and use these services **autonomously**



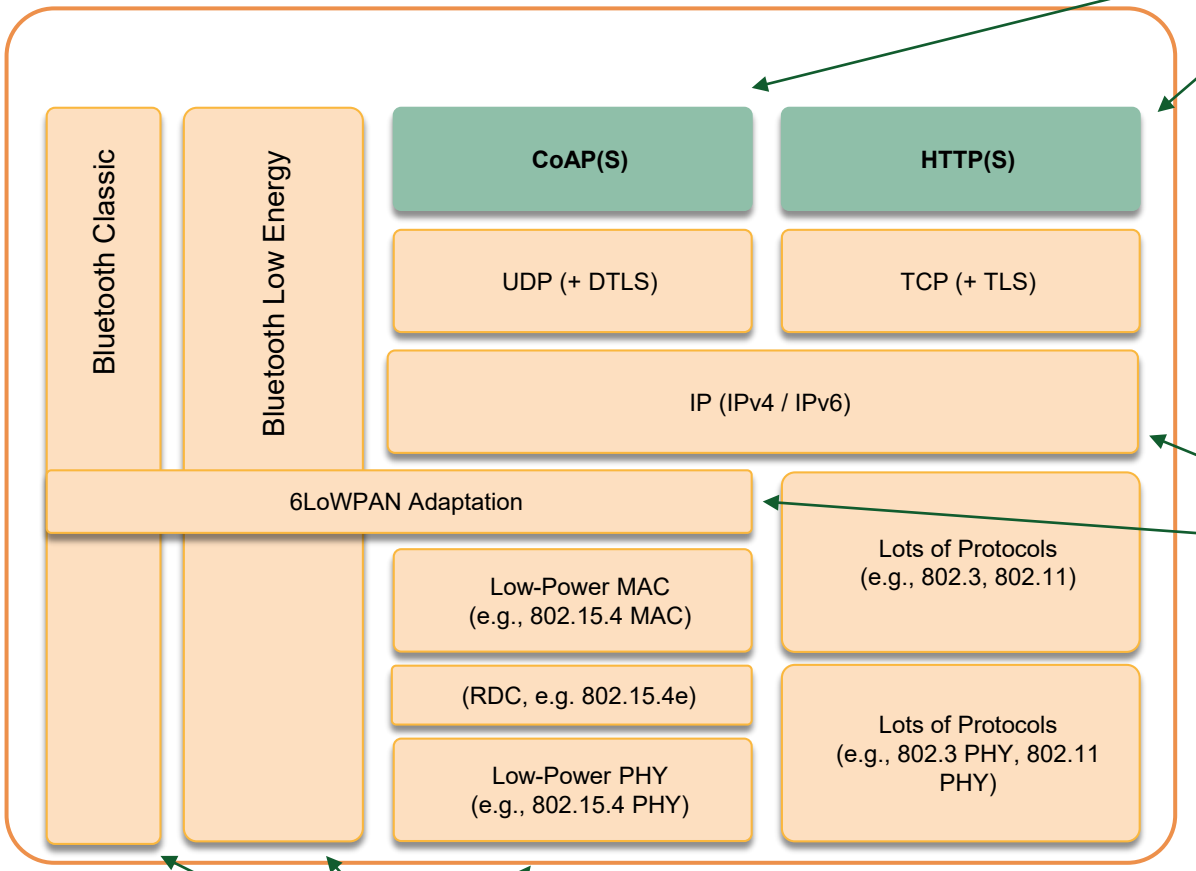
Distributed Systems Design Space



Source: Steve Deering, Talk at IETF 51, 2001



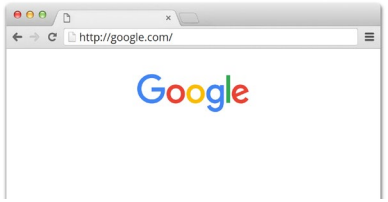
https://spectrum.ieee.org/tech-talk/semiconductors/design/teenytiny-bluetooth-transmitter-runs-on-less-than-1-milliwatt



CoAP

Interoperability on the **Application Layer**

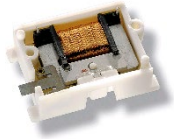
To understand hypermedia's importance, imagine using any website without it.



Interoperability on the **Network Layer**

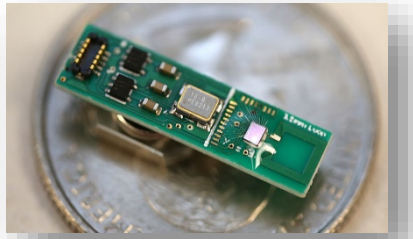
6LoWPAN
IPv6-based Low-power Wireless Personal Area Networks

Networking of **Tiny Devices**



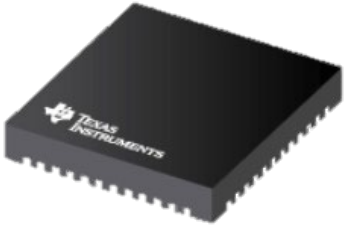
LoRa™

Ubiquitous Computing



e.g., BLE

e.g., IEEE 802.15.4



UbiComp applications require…

- Reduced power consumption, reduced cost, reduced size, near real-time operation (i.e., small latency)
- **Network-layer** connectivity and integration
- **Application-layer** integration and interoperability: **Web of Things**
- **Semantic-layer** interoperability and understanding: **See tomorrow** ;-)

6LoWPAN
IPv6-based Low-power Wireless Personal Area Networks

Avoid Network/Application/Semantic Fragmentation of Distributed Systems

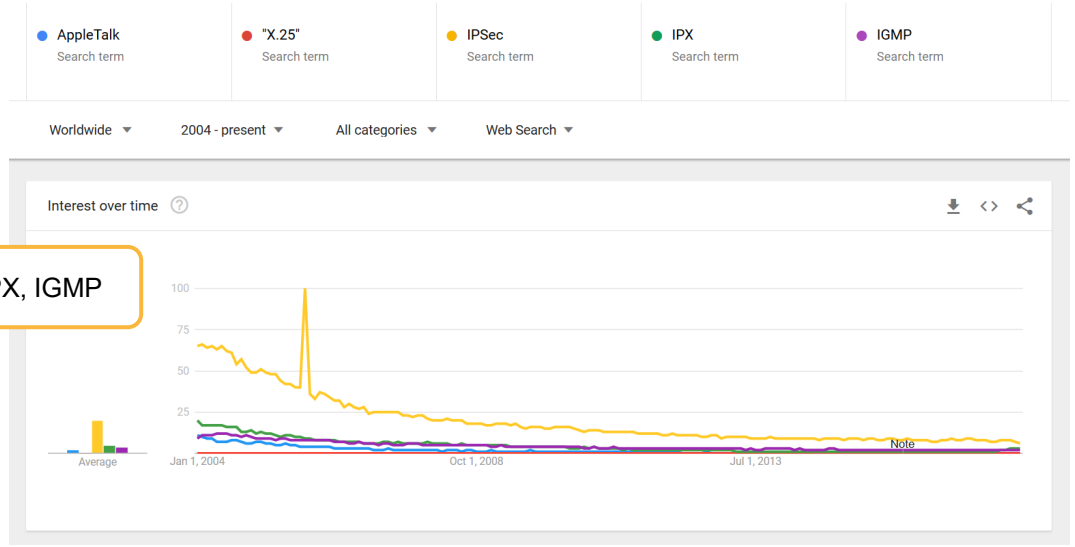
Why?

To enable exploiting «network effects»!

Network Effects: The Internet Protocol

ICMP: Not for data exchange…
IPsec: Security layer on top of IP
IGMP: *Replaced* by IPv6 Group Management
Xerox IPX: *Replaced* by IP (1998)
AppleTalk: *Replaced* by IP (1990-2009) [-> Bonjour]
ITU-T X.25 PLP: *Replaced* by IP (~2012; still used in niche applications)

3. Network layer [hide]
IP (IPv4 · IPv6) · ICMP · IPsec · IGMP · IPX · AppleTalk · X.25 PLP



Google Interest over time in AppleTalk, X.25, IPSec, IPX, IGMP

OSI model
by layer

7. Application layer [hide]
NNTP · SIP · SSI · DNS · FTP · Gopher · HTTP · NFS · NTP · SMPP · SMTP · SNMP · Telnet · DHCP · Netconf · *more....*

6. Presentation layer [hide]
MIME · XDR

5. Session layer [hide]
Named pipe · NetBIOS · SAP · PPTP · RTP · SOCKS · SPDY

4. Transport layer [hide]
TCP · UDP · SCTP · DCCP · SPX

3. Network layer [hide]
IP (IPv4 · IPv6) · ICMP · IPsec · IGMP · IPX · AppleTalk · X.25 PLP

2. Data link layer [hide]
ATM · ARP · IS-IS · SDLC · HDLC · CSLIP · SLIP · GFP · PLIP · IEEE 802.2 · LLC · MAC · L2TP · IEEE 802.3 · Frame Relay · ITU-T G.hn DLL · PPP · X.25 LAPB · Q.921 LAPD · Q.922 LAPF

1. Physical layer [hide]
EIA/TIA-232 · EIA/TIA-449 · ITU-T V-Series · I.430 · I.431 · PDH · SONET/SDH · PON · OTN · DSL · IEEE 802.3 · IEEE 802.11 · IEEE 802.15 · IEEE 802.16 · IEEE 1394 · ITU-T G.hn PHY · USB · Bluetooth · RS-232 · RS-449

V · T · E

IP: The Narrow Waist of Networking

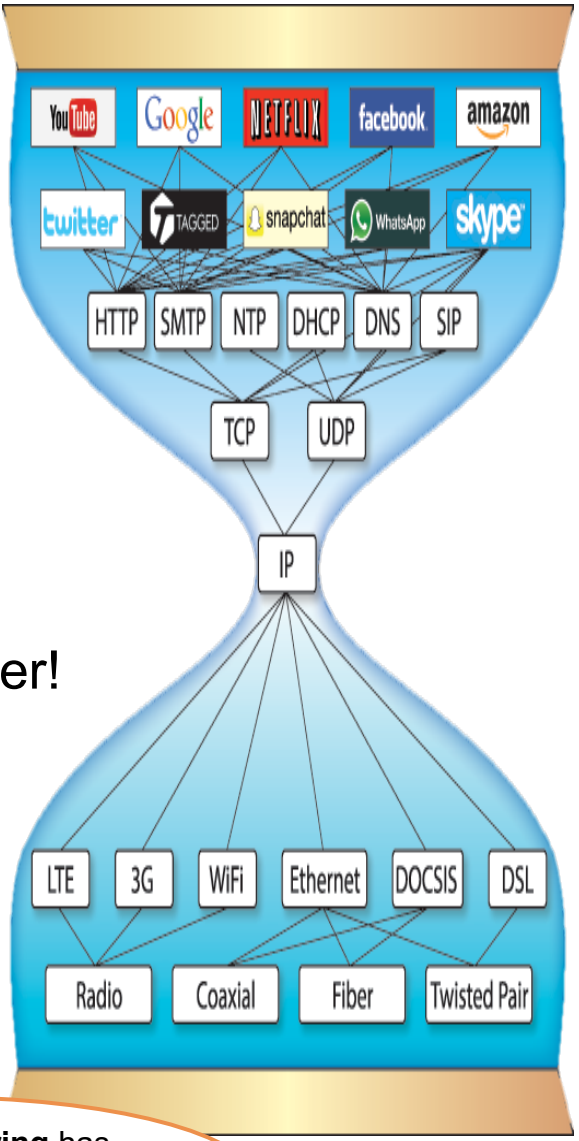
ICMP: Not for data exchange…
IPsec: Security layer on top of IP
IGMP: *Replaced* by IPv6 Group Management
Xerox IPX: *Replaced* by IP (1998)
AppleTalk: *Replaced* by IP (1990-2009) [-> Bonjour]
ITU-T X.25 PLP: *Replaced* by IP (~2012; still used in niche applications)

"*IP over everything, everything over IP"* is the **biggest network effect** ever!

- Fundamental goal: interconnect **heterogeneous** systems
- Fundamental requirement: **global** addressing scheme

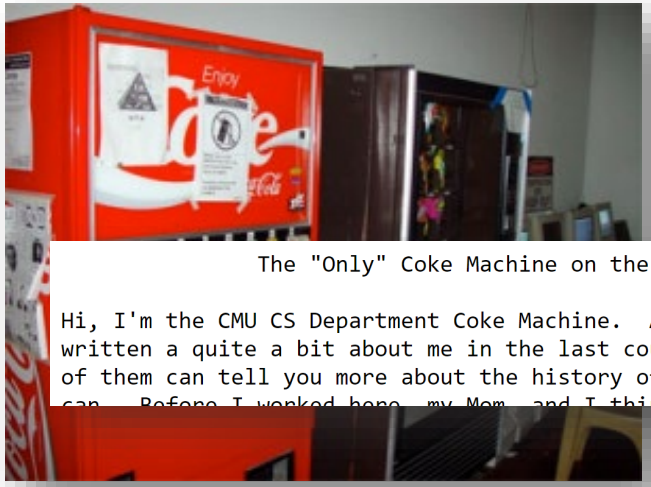
Drawback?

- Adding functionality in the network layer is extremely difficult!
- This is exemplified in the **tedious decade-long process** of switching from IPv4 to IPv6



In general, **layering** has worked out very well for computer networks…

The Internet of Things



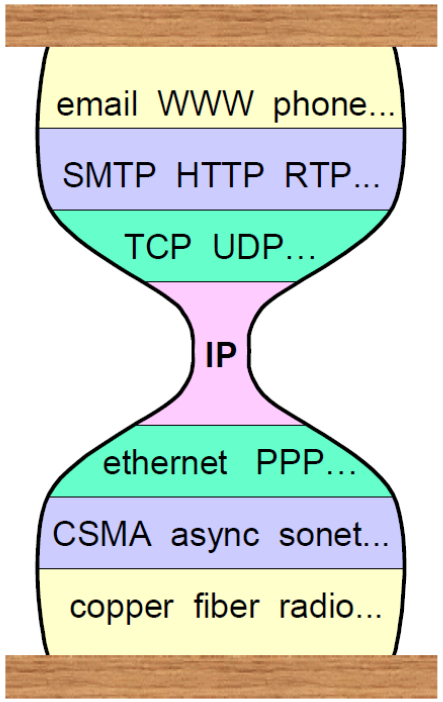
IoT = Internet of Things = **IP-enabled Physical Devices**

Original Internet of Things Idea

- Extend that biggest network effect ever to **physical devices**
- One network for **heterogeneous physical systems**
- Stretching into **low-power** systems as well (e.g., LoWPANs)

e.g., in the smart home space: https://www.connectedhomeip.com/

amazon · Google · zigbee alliance · IKEA · legrand · LEEDARSON · MMB NETWORKS · NXP · resideo · SmartThings · Schneider Electric · signify · SILICON LABS · somfy · Wulian



Source: Steve Deering, Talk at IETF 51, 2001

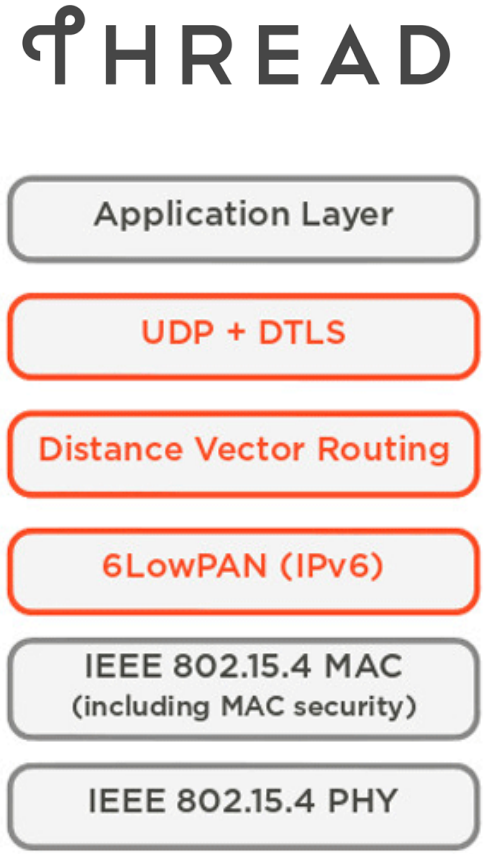
Thread: Branding Low-power Mesh Networking for the IoT

Backed by lots of relevant companies

- "Thread Group" Interest Group
- Nest Labs / Alphabet, Apple, Samsung
- ARM, Qualcomm, NXP
- OSRAM, Tyco

Thread = a "stack brand name"

- 802.15.4 for the lower layers
- **6LoWPAN for IPv6 compatibility**
- **IP routing for network**
- UDP for little overhead
- DTLS for security
- **"Any" application layer** (should support constrained devices…)



https://www.electronicdesign.com/iot/engineering-essentials-iot-standards-and-frameworks

History…

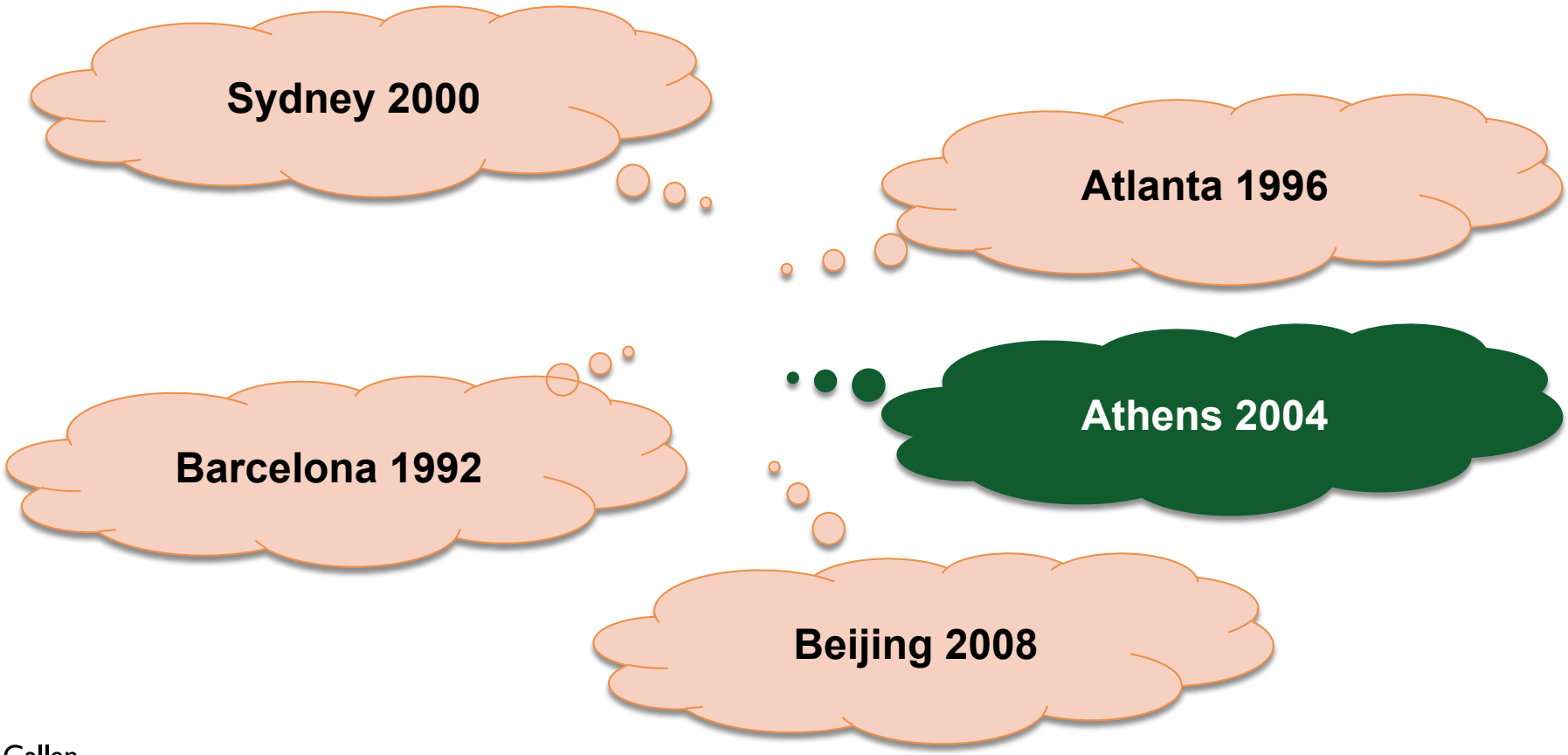
AUDIENCE PARTICIPATION

Which Olympic Games were the first to have an **official** Website?

- **Sydney 2000**
- **Atlanta 1996**
- **Barcelona 1992**
- **Athens 2004**
- **Beijing 2008**

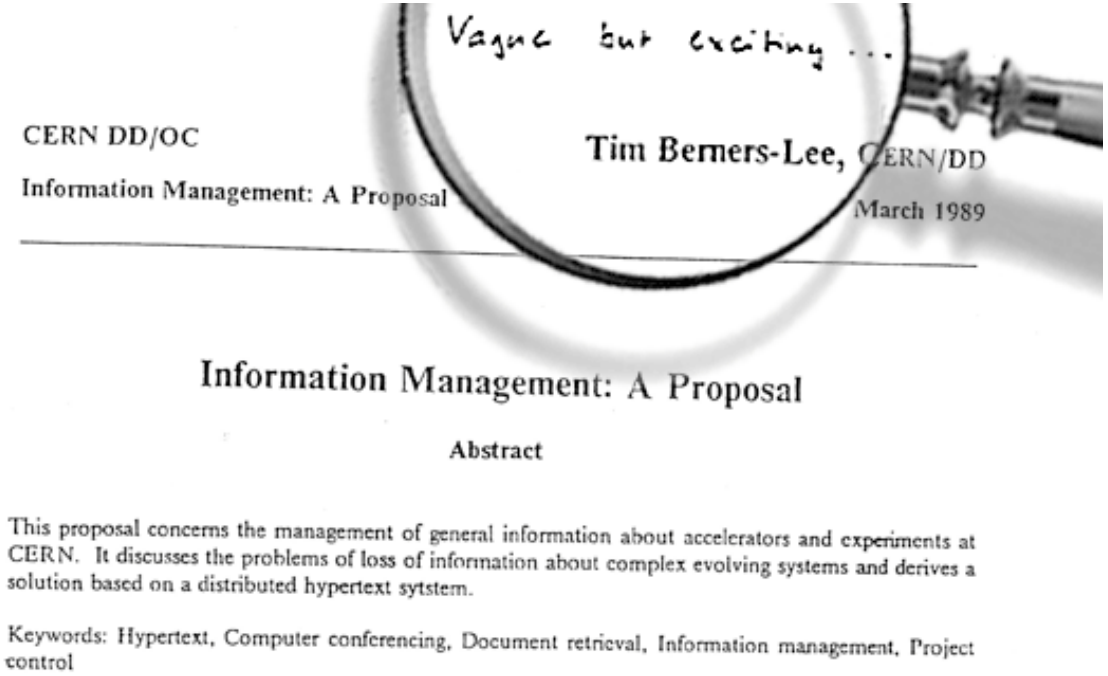
History…

Which Olympic Games were the first to have an **official** Website?



The **World Wide Web** made the Internet popular…

1989: Tim Berners-Lee submits a proposal for an information management system at CERN to his boss, Mike Sendall (http://info.cern.ch/Proposal.html)


Vague but exciting ...

CERN DD/OC

Information Management: A Proposal

Tim Berners-Lee, CERN/DD

March 1989

Information Management: A Proposal

Abstract

This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext sytstem.

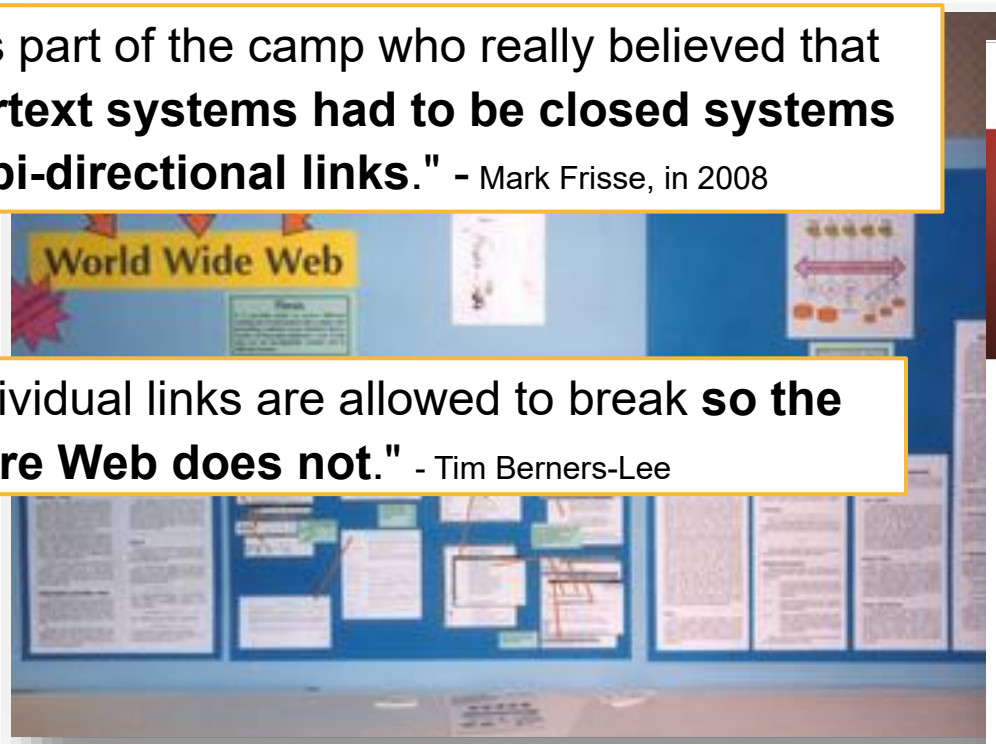
Keywords: Hypertext, Computer conferencing, Document retrieval, Information management, Project control

1990: Berners-Lee defines the terms **URI** (then UDI), **HTTP**, and **HTML**, and develops the first components and tools for the Web (browser, editor, server, line-mode browser)

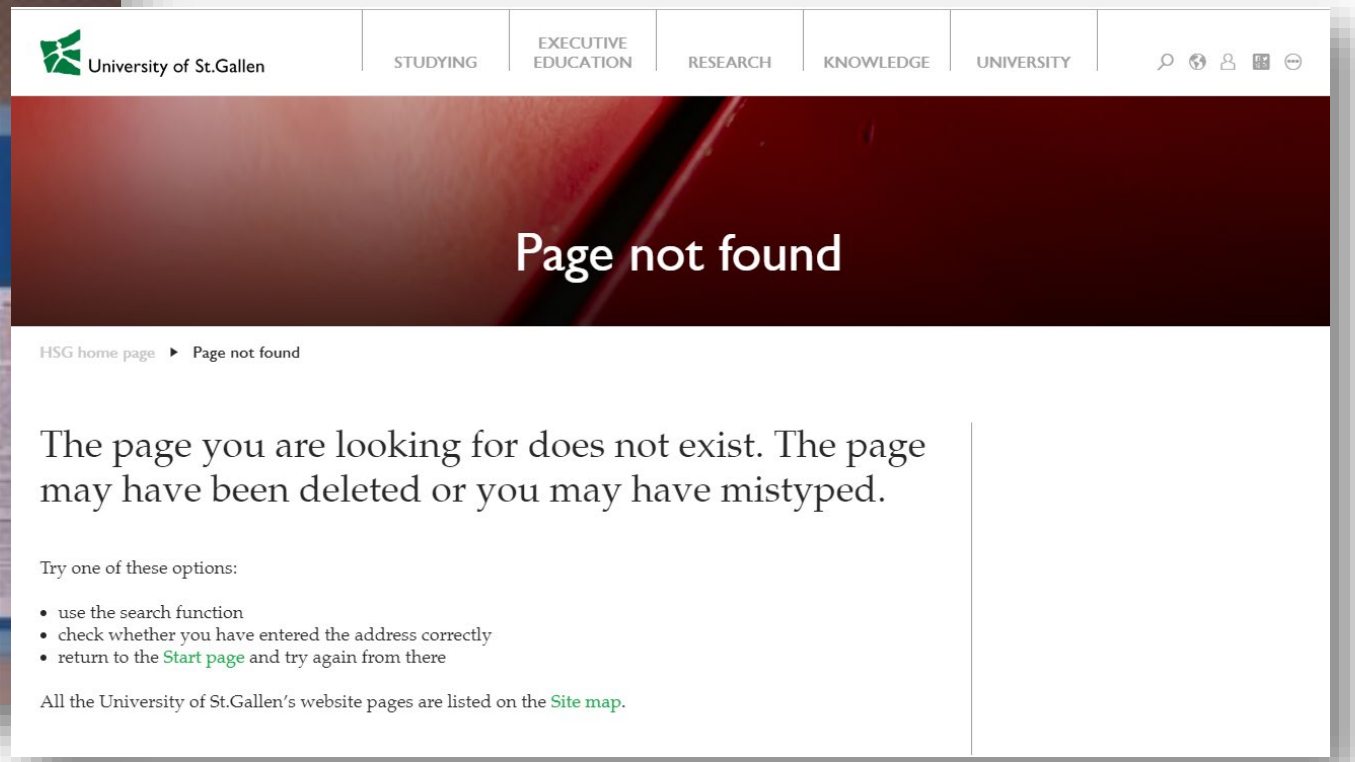
The **World Wide Web** made the Internet popular…

1991: Tim Berners-Lee submits a paper on the Web to the *Hypertext '91 Conference*.
The paper is **rejected as an oral presentation** and only accepted as a **poster** contribution…

"I was part of the camp who really believed that **hypertext systems had to be closed systems with bi-directional links**." - Mark Frisse, in 2008



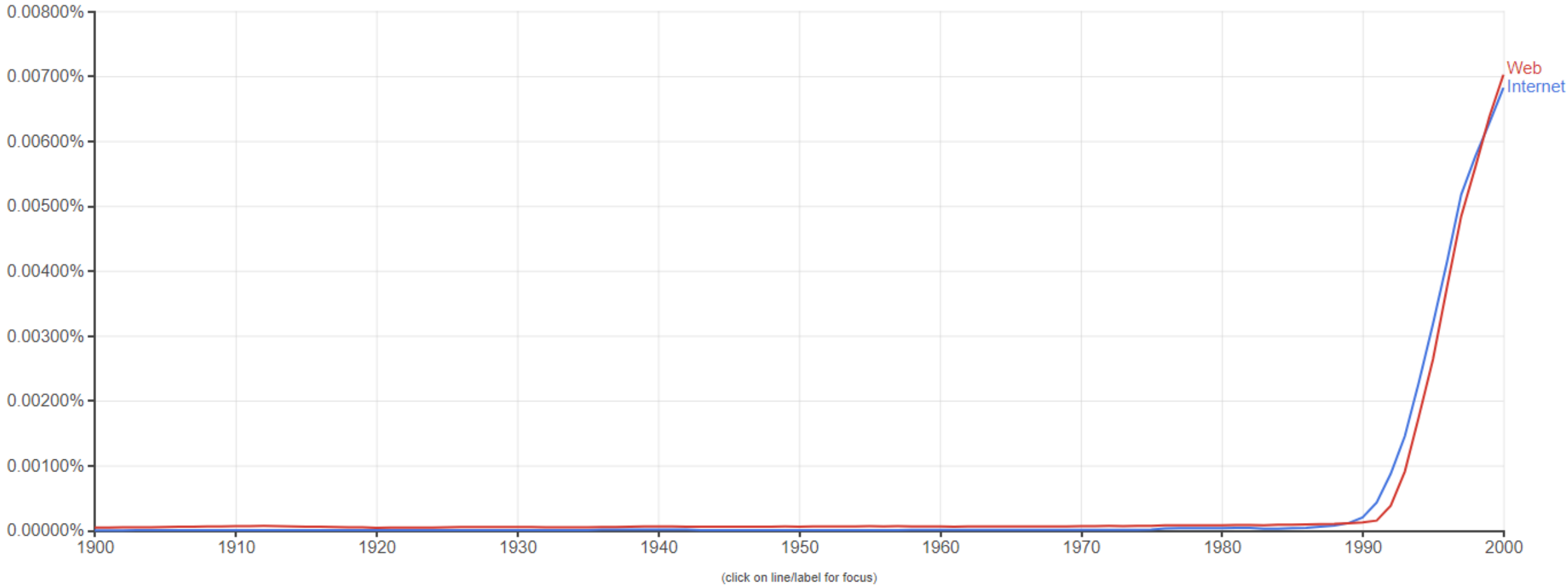
"Individual links are allowed to break **so the entire Web does not**." - Tim Berners-Lee



Google Books Ngram Viewer

Graph these comma-separated phrases: Internet,Web case-insensitive

between 1900 and 2000 from the corpus English with smoothing of 3 . Search lots of books



(click on line/label for focus)

The **World Wide Web** is an information system where **resources** are identified by **Uniform Resource Identifiers** (URIs) which are accessible over the **Internet**, and may be interlinked by **hyperlinks**.

….stretching to **physical devices** as well!

Even Tiny Devices!

HTTP is not the only player out there!

CoAP has been designed from scratch following the REST architectural style.

The target scenario for CoAP is one of **constrained** nodes and networks

- Constrained nodes can be sensors with **limited memory and processing** power
- Constrained networks can be **slow** and **unreliable** and **intermittent**
- **TCP performs poorly** in low-power wireless networks
- CoAP is optimized for bandwidth and processing efficiency

CoAP

CoAP Properties

- Very **efficient** REST on top of UDP
- **Lightweight security** to prevent eavesdropping, tampering, forgery
- Specialized for M2M applications: **Multicast** & **Observe**!
- Easy to proxy from/into HTTP but not a general replacement for HTTP

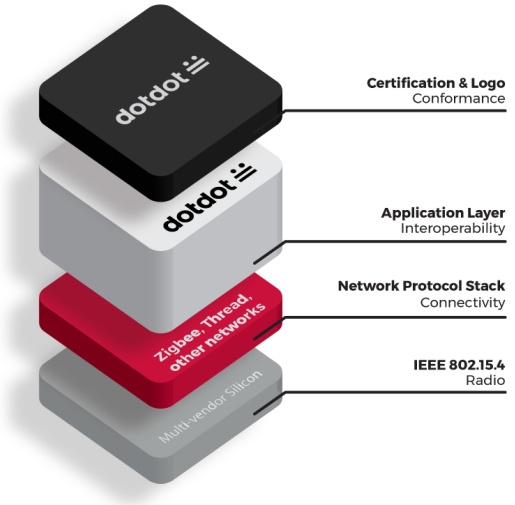
Dotdot over Thread

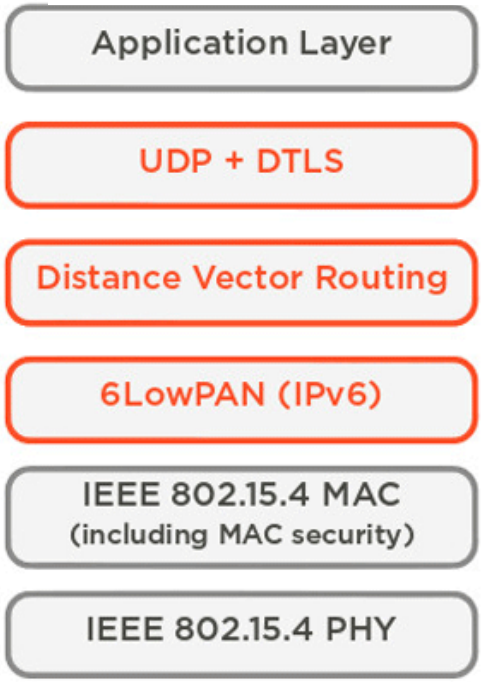
Thread

- 802.15.4 for the lower layers
- 6LoWPAN for IPv6 compatibility
- IP routing for network
- UDP for little overhead
- DTLS for security
- “Any” application layer (should be low-bandwidth though…)

Dotdot = An “application layer brand name”

- CoAP + CoRE Link Format



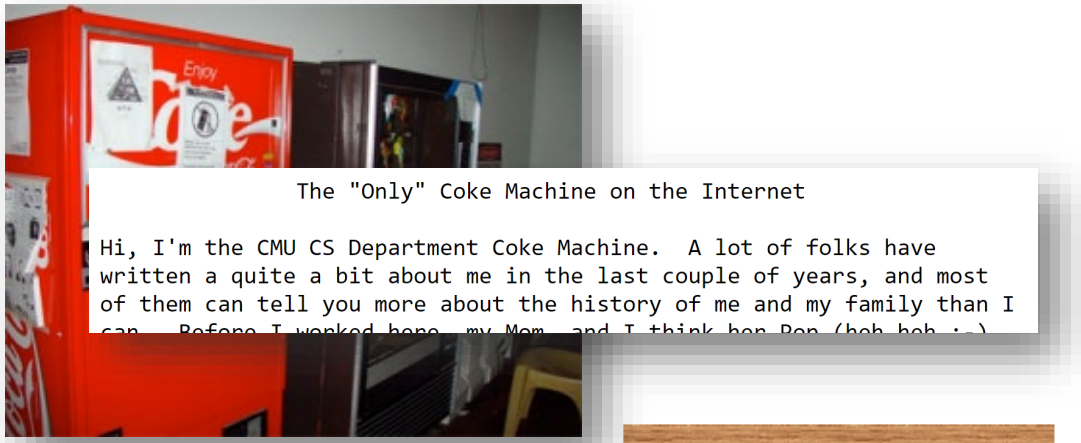


“Built with the open standards and global membership of the Zigbee Alliance, Dotdot gives us the freedom to choose the brands and products that transform the way we live, work, and play.”

https://zigbeealliance.org/solution/dotdot/

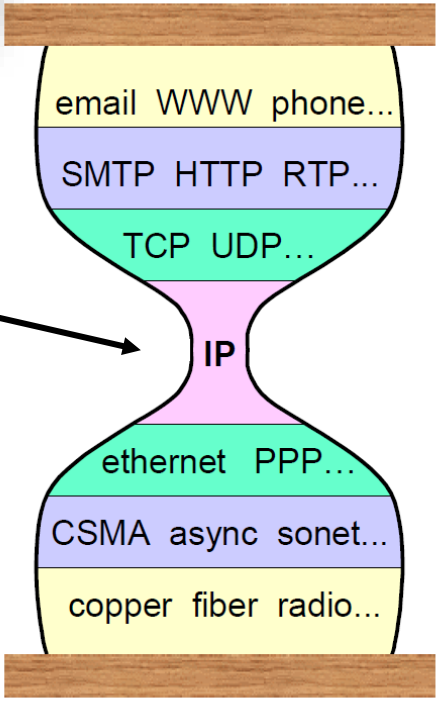
https://www.electronicdesign.com/iot/engineering-essentials-iot-standards-and-frameworks

The IoT and the WoT



Original Internet of Things Idea

- Extend the **biggest network effect ever** to physical devices
- **IP-over-Everything, Everything-over-IP**
- One network for **heterogeneous systems**
- Even integrating LoWPANs!



Original and Current WoT Ideas

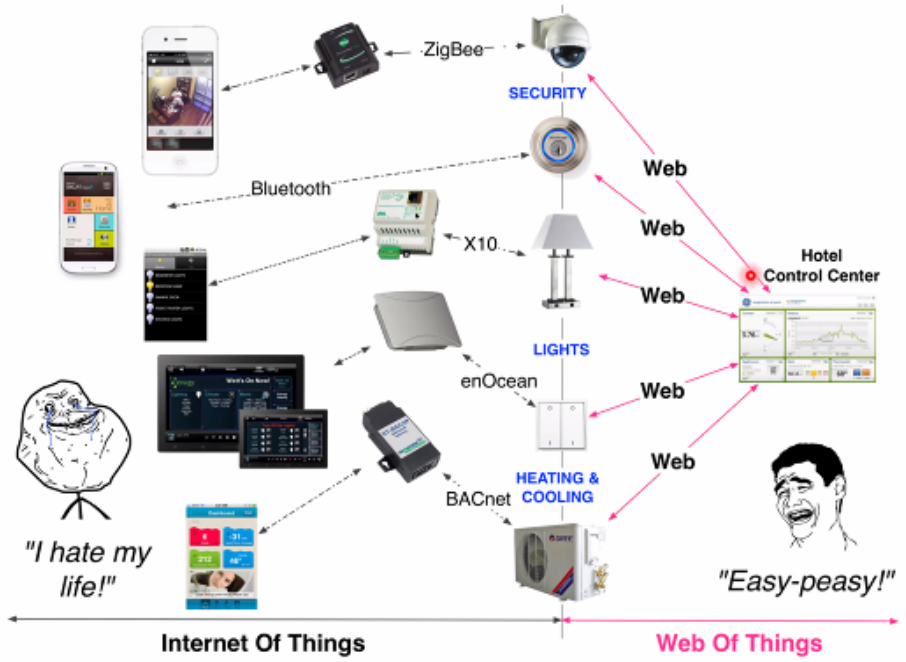
- Apply Web principles to **defragment the IoT's application layers**
- Apply Web principles to **simplify the creation of applications** on top of networked devices
- Apply Web principles to make such **WoT mashups easily usable by people**

Source: Steve Deering, Talk at IETF 51, 2001

yelp

BART

Bay Area
Rapid Transit



Guinard and Trifa, 2012

Demonstration: A Mashupping Exercise!

Setup

- Ubiquitous Computing course at ETH Zurich (2016)
- 50 Students (70% MSc, 30% BSc; 70% CS Students)

Task

– Work with the: **Bay Area Rapid Transit** (BART) & **Yelp Fusion APIs**

– Write a mashup that, given a BART station identifier, lists the 5 best-rated restaurants around that station

Time is of the essence: prizes for 1st 2nd 3rd place

Bay Area
Rapid Transit

Demonstration: A Mashupping Exercise!

(divided by the minutes needed to solve the challenge…)

Demonstration: A Mashupping Exercise!

`Teams of 2`

`Any language` (that you can demonstrate works)

`Any means` (if you copy code, **reference** properly)

`SILENT submission` (zip, rename to txt, submit by email)

`Max. until 7pm (i.e., max 1h);`

Recommended language: Python (quick wins, good libraries)

- *requests* for Web requests
- *xml.etree.ElementTree* for parsing XML

Instructions

- 1 paragraph of instructions
- Detailed walk-through

Demonstration: A Mashupping Exercise!

Solved successfully by 20 teams within the time limit

Submitting students: Felix, Jonathan, Johnathan, Dominik, Tobi, Martin, Andrei, Chris, Nino, Dominik, Thomas, Balz, Ingo, Martina, Simon, Emily, Krisztina, Yongzhe, Mrigya, Daniel

4th prize:	50min
3rd prize:	39min
2nd prize:	31min
1st prize:	**24min**

Bay Area Rapid Transit

i.e.: a **trained individual can use the Web to create a useful (potentially commercializable) application** while someone else **is in their coffee break**.

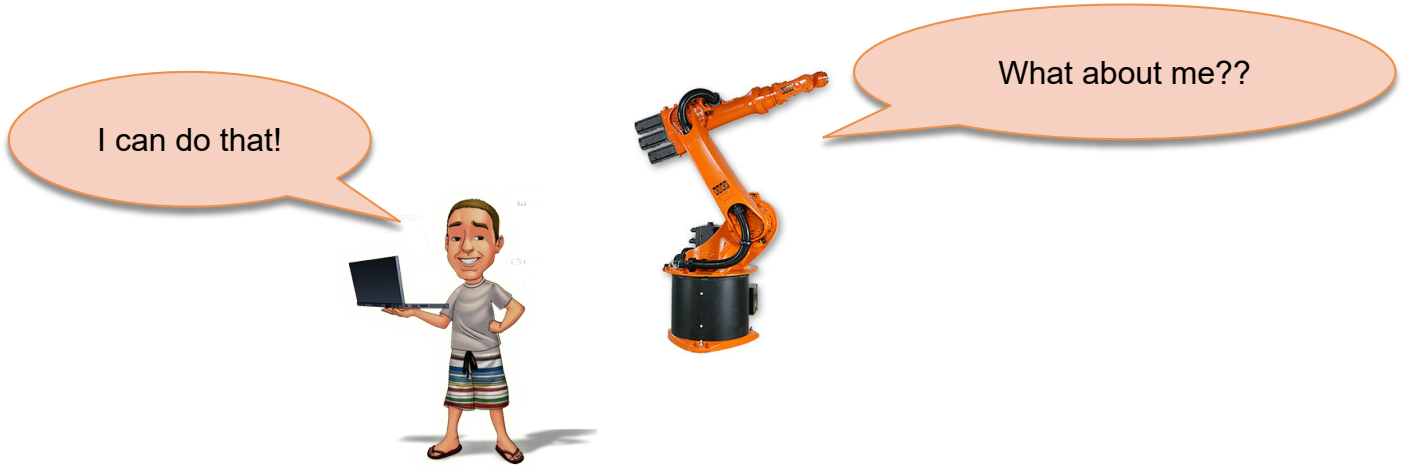
Demonstration: A Mashupping Exercise!

If you want people to use your API, it **must be easy to develop for**!

- **Simple to use**: use widespread patterns and formats
- **Simple to register for**: if a more advanced authentication scheme is used, this must be documented properly
- **High-quality API documentation**
- **Example code and/or client libraries**

These factors make your API easy to use – **for humans**!

Current research concerns **how to make APIs easy to use for machines** ;-)



Sneak Preview

University of St.Gallen

Institute of Computer Science

ICSN, HS2019
Simon Mayer & Iori Mizutani
simon.mayer@unisg.ch; iori.mizutani@unisg.ch

Assignment 4: The Bits go Global (7pt)

Deadline: December 11, 2019; 16:00 CET

(3) Your Sensor goes Global (2pt)

Although you'll only be creating a basic Web server in this task, have a look at the current IRTF recommendations for *RESTful Design for Internet of Things Systems*.[5] Next, create a simple Web server that enables you to read the most recently measured temperature value of your DS18B20 sensor via an HTTP GET request to your Raspberry Pi. You are free to design and implement this Web server from scratch – it is however also permissible that you use the *OpenAPI* API description format for specifying your sensor's Web API and then use the *swagger* tool[6] to automatically generate server code that you can deploy on your RPi. Finally, integrate the (generated) server with the code that accesses the temperature sensor and verify that your system works correctly by accessing the RPi's Web server remotely from a browser or using the Postman application. If you chose to have the server code generated, make sure that you understand it.

(4) Physical Mashups! (2pt)

For this task, we provide you with a Web API (OpenAPI description: `https://interactions.ics.unisg.ch/leubot`) to control a *PhantomX Reactor Robot Arm*[7] that is installed in our laboratory space (the robot's name is "Leubot").

Try out the API by reading the description and composing a set of requests to move the arm around. You can verify the robot arm's behavior by checking the live video stream of the robot[8].

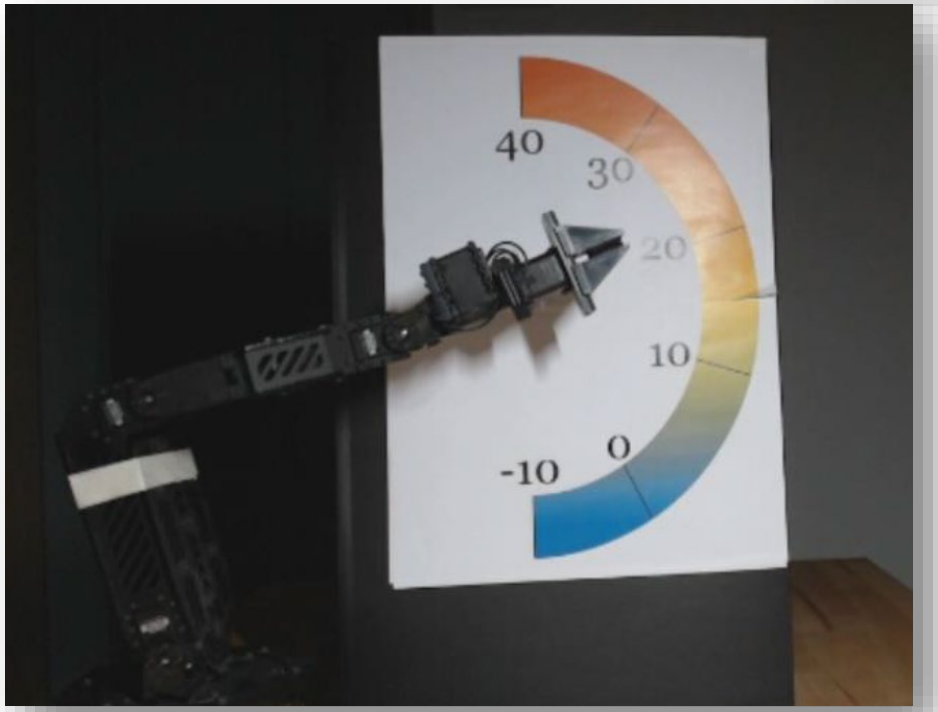
Sneak Preview

University of St.Gallen
Institute of Computer Science

ICSN, HS2019
Simon Mayer & Iori Mizutani
simon.mayer@unisg.ch; iori.mizutani@unisg.ch

Assignment 4: The Bits go Global (7pt)

Deadline: December 11, 2019; 16:00 CET



(3) Your Sensor goes Global (2pt)

Although you'll only be creating a basic Web server in this task, have a look at the current IRTF recommendations for *RESTful Design for Internet of Things Systems*.[5] Next, create a simple Web server that enables you to read the most recently measured temperature value of your DS18B20 sensor via an HTTP GET request to your Raspberry Pi. You are free to design and implement this Web server from scratch – it is however also permissible that you use the *OpenAPI* API description format for specifying your sensor's Web API and then use the *swagger* tool[6] to automatically generate server code that you can deploy on your RPi. Finally, integrate the (generated) server with the code that accesses the temperature sensor and verify that your system works correctly by accessing the RPi's Web server remotely from a browser or using the Postman application. If you chose to have the server code generated, make sure that you understand it.

(4) Physical Mashups! (2pt)

For this task, we provide you with a Web API (OpenAPI description: https://interactions.ics.unisg.ch/leubot) to control a *PhantomX Reactor Robot Arm*[7] that is installed in our laboratory space (the robot's name is "Leubot").

Try out the API by reading the description and composing a set of requests to move the arm around. You can verify the robot arm's behavior by checking the live video stream of the robot[8].

If creating a robot mashup is this simple, «Industry 4.0» is just a matter of time.
If creating a robot mashup is this simple, **this process can perhaps be fully automatied**…

Our Menu

From the IoT to the WoT: Network-level and Application-level Interoperability

Discussion: Good Practices for Web APIs

Tidbits from the Web Architecture

Hypermedia APIs

Point in Case: The Europeana Website

(example from R. Verborgh: Web Fundamentals Introduction)

Europeana: Aggregates and exposes **cultural heritage metadata** from more than 2000 institutions (British Library, Louvre, Stiftsbibliothek St. Gallen, etc.) – requires a **long-term architectural vision**

Europeana resources are identified by URIs:
https://www.europeana.eu/portal/en/record/9200143/BibliographicResource_2000069304654

Europeana publishes its data…

- On a website with content negotiation (HTML and JSON)
- Via a dedicated machine API

Using the Europeana website as a **machine** is thus possible in **two ways**

1. Use the regular website and request JSON data
2. Use the dedicated machine API

What do you think: **Which one is easier to implement?**

Point in Case: The Europeana Website

(example from R. Verborgh: Web Fundamentals Introduction)

Accessing https://www.europeana.eu/portal/en/record/9200143/BibliographicResource_2000069304654 as a machine client...

Option 1: Use the **regular website** and request JSON data

1. Set a request header to “Accept: application/json”
2. Request the data at https://www.europeana.eu/portal/en/record/9200143/BibliographicResource_2000069304654
3. Do whatever you wanted to do…

Option 2: Use the **dedicated machine API** (see https://gist.github.com/RubenVerborgh/7684361/#gistcomment-959892)

1. Read the documentation
2. Get an API key
3. Find the right URL template for the API call
4. Find out the object ID of your object (non-trivial beca
5. Fill out the template to construct the URL
6. Retrieve a representation using this URL
7. Do whatever you wanted to do…

*All of this was not required for the human-readable website, even though the **retrieved information is exactly the same**, just the format is different!*

Point in Case

https://ras.papercept.net/conferences/scripts/start.pl

Properly using URLs

Principle: A resource should relate a URL to a concept

http://example.org/songs/showDetails.php

vs.

http://example.org/songs/3563642

What does this URL represent?

Is it useful to **bookmark** this URL?

Is it useful to **share** this URL?

Source: R. Verborgh, Web Fundamentals

Supporting Statelessness

Principle: Each message should be self-descriptive

/songs?artist=5521		*/songs?artist=5521*
/?page=2	*vs.*	*/songs?artist=5521&page=2*
/?page=3		*/songs?artist=5521&page=3*

What does this URL represent?

Is it useful to **bookmark** this URL?

Is it useful to **share** this URL?

Source: R. Verborgh, Web Fundamentals

Making use of Hypermedia

Principle: The interaction should be driven by hypermedia

{ "title": "Baba O'Riley", "artist": { "id": 5521 } }

vs.

{ "title": "Baba O'Riley", "artist": { "id": "http://example.org/artists/5521" } }

Can I derive a **useful next interaction** from this?

Source: R. Verborgh, Web Fundamentals

Controlling API Access

Pattern in many online services: **Websites are open,** but require **API keys** for machine access

- *"We need to rate-limit machine access"*
- *"We need to track automated access"*
- *"We need to protect our data"*

But they **break things** by personalizing URLs and thereby blocking the bookmarking, sharing and caching of resource representations

API keys don't help with this, since **the HTML website contains the exact same information without a key.** In other words: **one can extract all data from HTML, it's just more tedious!**

More complex/costly ways of "protecting" API access have similar problems…

Example: Twitter Search API (https://developer.twitter.com/en/docs/tweets/search/overview)

Take-home idea: protect your **resources**, not your **resource representations**!

Better API Design for Machine Clients

Ideally, use the **same resources** that your human-centered website is based on and just **extend that site with machine-readable, linkable, representations.** Ideally, your machine API then **does not require documentation** (just like a human-centered website)

Take care though:

- Encouraging clients to have code **generated from an API description** is a can of worms, as clients are then tightly coupled with your API!
- This was (historically) on of the central problems of the WS-* specifications: clients were hard-compiled against API descriptions!

If your API gives access to a physical thing, consider these (informational) best practices:
https://tools.ietf.org/html/draft-irtf-t2trg-rest-iot-06

Next, we put these best practices in context through a (brief) look at the **Web Architecture**

Any Questions / Comments / Doubts / Concerns?

Our Menu

From the IoT to the WoT: Network-level and Application-level Interoperability

Discussion: Good Practices for Web APIs

Tidbits from the Web Architecture

Hypermedia APIs

What made the **Web** so successful?

In my (informed) opinion:

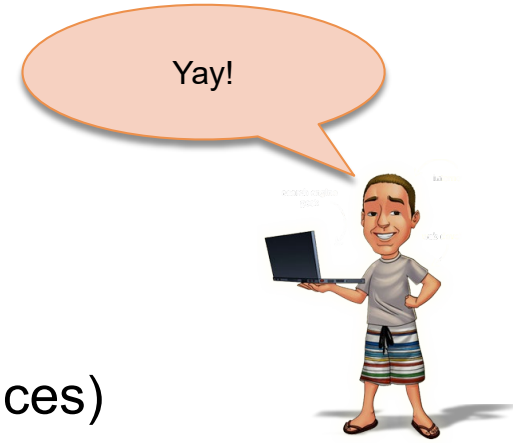
Its **scalability**, **flexibility**, **evolvability**, **usability** (among other **non-functional properties**), which in turn enabled ***innovation without bound***.

Design focus on open standards.
Still done today! (-> Matthias' keynote & Dave's keynote)

https://bayouwebdesignplus.com/www-world-wide-web-or-wild-wild-west/

These **non-functional properties** were ***designed into*** the architecture of the World Wide Web.

Properties of the Web Architecture



The Web and its architecture…

- …are **proven** and **scale**
- …can be deployed on almost **any computing platform** (incl. Class 1 devices)
- …are supported by virtually **all programming languages**
- …are **easy** and **intuitive** to **use** and **develop** for -> **huge developer community**
- …support relevant features for smart buildings, including **multicast** and **observing** (via CoAP)
- …map to **many other protocols** in relevant IoT target domains
  - → **Building Automation**: KNX, 802.15.4/6LoWPAN, BACnet, dot-dot over thread, etc.
  - → **Industrial Automation**: ModBus, OPC UA, etc.
  - → **Smart Grids**: IEC 61850

[F. D. Davis. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology, 1989]
[D. Gefen and M. Keil. The Impact of Developer Responsiveness on Perceptions of Usefulness and Ease of Use, 1998]
[D. Guinard, I. Ion, S. Mayer: REST or WS-*? A Developers' Perspective, 2011]
[L. Popa, P. Wendell, A. Ghodsi, I. Stoica: HTTP: An Evolvable Narrow Waist for the Future Internet, 2012]
[S. Mayer, R.G. McDaniel: Automatic Generation of OPC UA Servers and Clients from Knowledge Models. U.S. Patent Application No. CT/US2018/048682, 2018]

Properties of the Web Architecture

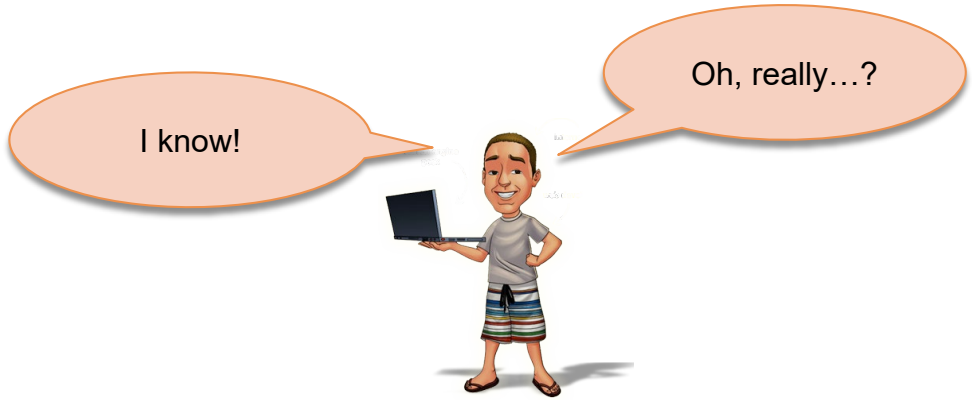
Where do these (beneficial!) properties come from?

The **idealized architectural style** of the Web is called "*Representational State Transfer*" (REST)

…and that is **more than a buzzacronym**!

Notes

- REST provides **architectural guidelines** for computing infrastructures, not only the Web
- In a Web context, REST is traditionally implemented via **HTTP** or **CoAP**
- In a Web context, REST is compatible with the **Resource-Oriented Architecture**



Representational State Transfer (REST)

Resource-Oriented Architecture (ROA) and REST

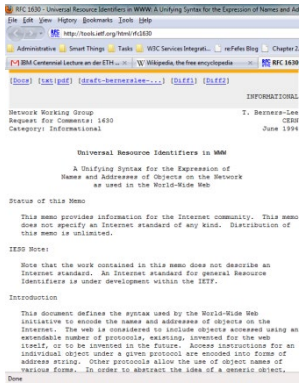
- Functionality is integrated into **resources**, not offered **by services**
- **REST is the idealized architectural style of the Web**

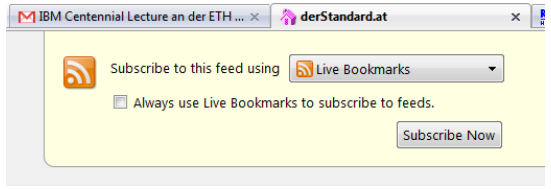
What are resources, actually?

- RFC **1630** „URL“ (1994): “Resource is anything that can be addressed”
- RFC **2396** „URI“ (1998): “Resource is anything that has identity (…) don’t have to be network-retrievable” + Examples
- RFC **3986** „URI“ (2005): “Abstract concepts can also be resources”

Examples of (Web) Resources

- Static Websites
- Dynamic Websites
- RSS Feeds
- Data Containers
- Physical Things





Representational State Transfer (REST)

The idealized architectural style of the Web

REST defines **six architectural constraints** that restrict how servers may process client requests and send responses **to gain desirable non-functional properties**

- Client-server Architecture
- Statelesseness
- Cacheability
- Layered System
- Code on Demand (optional)
- **Uniform Interface**

The Uniform Interface Constraint

Core idea: allow **heterogeneous** implementations of components to **interact in a uniform manner**.

- Component implementations are **heterogeneous** because they are implemented by different developers using different programming languages, frameworks, data models, data representation formats, etc.

How can "**heterogeneous people**" interact "**uniformly**"?

- via a shared language, e.g. English

What is the **tradeoff**?

from: Andrei Ciortea

The Uniform Interface Constraint

Core idea: allow **heterogeneous** implementations of components to **interact in a uniform manner**.

- Component implementations are **heterogeneous** because they are implemented by different developers using different programming languages, frameworks, data models, data representation formats, etc.

How can "**heterogeneous people**" interact "**uniformly**"?

- via a shared language, e.g. English

What is the **tradeoff**?

- they can now interact and understand one another ("interoperability")
- but the "uniform interface" can be inefficient (e.g., for non-native English speakers)

from: Andrei Ciortea

The Uniform Interface Constraint

The same small set of operations applies to everything (i.e. to every resource)

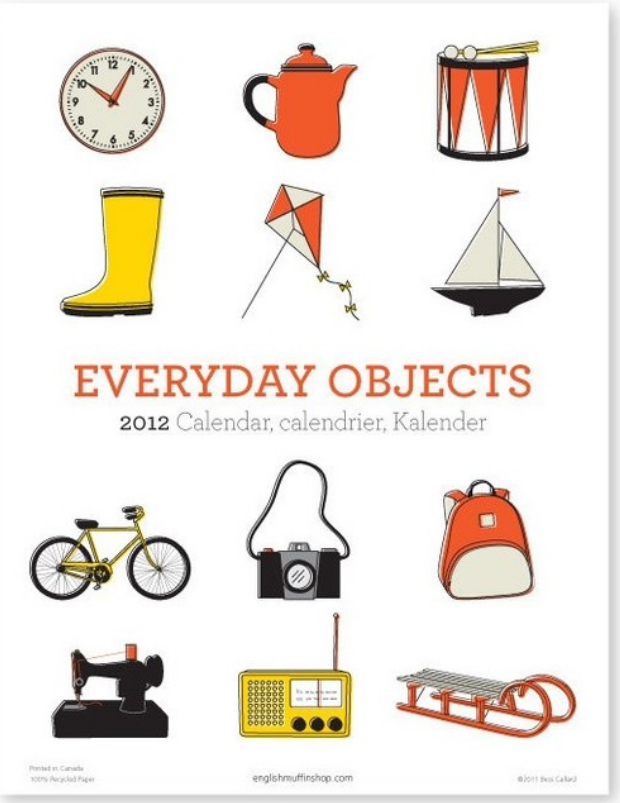
A small set of *verbs* applied to a large set of *nouns*

- Verbs are **universal** and **not invented per-application**
- If many applications need new verbs, the uniform interface can be extended
- Natural language works in a similar way: new verbs rarely enter language

HTTP works in this way (by defining a controlled set of HTTP methods)

- Other schemes have other (or no) interaction models
- *ftp:* works similar to HTTP (TCP/IP-based access with a defined set of commands)
- *geo:* interaction might mean “take me there,” or “that’s a great place,” etc.
- *isbn:* does not implement an interaction model with the identified resource

An Analogy to Linguistics



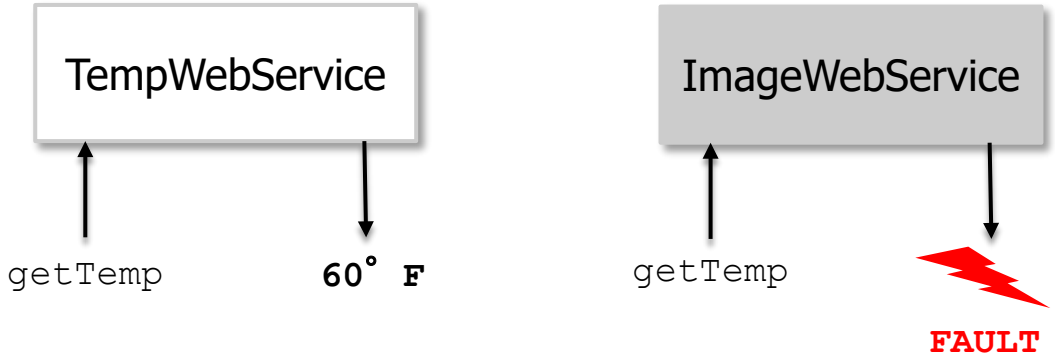
Nouns identify concepts

- Bike, Camera, Kite, etc.
- New nouns enter language regularly over time…

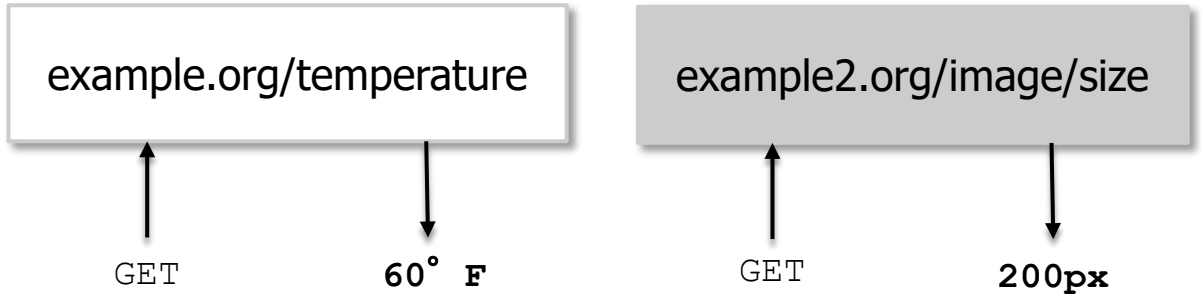
Verbs are “applied” to nouns

- “Pick up a bike”, “Pick up a camera”, etc.
- Their semantics vary **very little.** Comparatively **few verbs enter language** over time
- **Verbs are polymorphic!**

An Analogy to Linguistics: Polymorphism



What happens if we emphasize the **Nouns and fix the Verbs**?



The Uniform Interface Constraint and Linguistics

A **small set of verbs** with **well-defined semantics** that are agreeable among origin clients, origin servers, and any intermediaries in-between (i.e., the semantics of verbs are also *standard*)

- for HTTP: GET, POST PUT, DELETE, PATCH, OPTIONS, HEAD etc.
- for CoAP: GET, POST, PUT, DELETE

A **large set of nouns** with **well-defined semantics** that are (ideally) agreeable among origin clients, origin servers, and any intermediaries in between (i.e., the semantics are *standard)*

- in a REST-style system, such as the Web, nouns are **resources** and their semantics are defined by **media types** (e.g., `text/html` for HTML, `application/json` for JSON)

https://www.iana.org/assignments/media-types/media-types.xhtml

The Uniform Interface Constraint (example: HTTP)

Every Web browser on the planet can access every Web resource on the planet!

This is because HTTP works the same for every resource

- Interacting with the resource is possible **without knowing it beforehand**
- The interaction language is *only* about making interaction possible and scalable

What's more, the low-level interaction semantics of the (HTTP) verbs are well-defined!

We should consciously select which operation to use for an action on a resource

- HTTP GET and HTTP HEAD (and others) are *safe operations*
- HTTP PUT and HTTP DELETE (and others) are *idempotent operations*
- HTTP POST is a "catch-all" operation and may have side-effects

Caching + Pre-fetching!

Safe retries!

None of this!

POST bookstore.org/buy "Book 1"

The Uniform Interface Constraint

Four interface (sub-)constraints

1. ***Uniform Identification of Resources***: resources must have **uniform identifiers** that are uniquely addressable using a universal syntax; on the Web, we use URIs

2. ***Manipulation of resources via representations***: clients send and receive representations to and from servers (*representational state transfer*); in other words: no direct access to resources!

3. ***Self-descriptive messages***: a message (e.g., HTTP request, CoAP response) must include **metadata** that describes its meaning; this metadata should include (among others) the standard verbs and media type

4. ***HATEOAS***: Hypermedia as the Engine of Application State

The Uniform Interface Constraint

1. ***Uniform Identification of Resources***: resources must have **uniform identifiers** that are uniquely addressable using a universal syntax; on the Web, we use URIs

Name everything that you want to talk about in your application: those are **Web Resources**

- **Products** in an online shop
- **Categories** that are used for grouping products
- **Customers** that are expected to buy products
- **Shopping Carts** where customers collect products

Client state transitions also are represented as resources

- **Performing a next step** is also something you want to talk about!
- **Next** links on multi-page submission processes
- **Paged results** with URIs that identify the following page

The Uniform Interface Constraint

1. ***Uniform Identification of Resources***: resources must have **uniform identifiers** that are uniquely addressable using a universal syntax; on the Web, we use URIs

http://google.com/ means the same everywhere
- There is no (relevant) difference in terms of what the URI identifies
- Any ambiguity is created (and hopefully managed) **by Google**

Possible scenarios where this principle is violated:
- A **parallel internet** where google.com identifies a different entity
- **Intranet** applications identified with URIs such as http://magicservice/

Not only the Web makes use of global identification schemes…
- Not every scheme needs a computer: [tel:+15102061079]
- How about **locations**: [geo:37.859544,-122.327007]
- Or **physical resources**: [urn:isbn:3540642854]

The Uniform Interface Constraint

2. ***Manipulation of resources via representations***: clients send and receive representations to and from servers (*representational state transfer*); in other words: no direct access to resources!

Resources are **abstract** entities (they cannot be used *per se*)
- *Shared Identification* guarantees that they are clearly identified
- They are accessed through a *Uniform Interface*

Interaction with resources happens via **resource representations**
- It is communicated which kind of representation is used
- Representation formats can be **negotiated between peers**
- Whatever the representation is, in a REST API it **must support links** (hypermedia!)

When you access a Web resource, you see **one of its representations**
- Your browser **negotiates** for you! (HTML, locale/language, context attributes)
- If a machine accesses a Web resource instead of a person, it will (should!) see a **different representation**

The Uniform Interface Constraint

3. ***Self-descriptive messages***: a message (e.g., HTTP request, CoAP response) must include **metadata** that describes its meaning; this metadata should include (among others) the standard verbs and media type

Without a *Shared Representation Model* there can be no **shared understanding**

Two steps: **Syntactic** understanding (readability) and **Semantic** understanding

HTML works great for the human Web because of **two** factors:

- Browsers know how to render a Web resource so that a human can **read** it (e.g., via HTML)
- Humans can understand the representations because they have a **hidden shared model**: **language**

What about machine clients that interact with Web resources?

- Making content **machine-readable** is the smaller of the two problems
- Establishing **shared understanding** for machines is a **very deep problem**! (we will see tomorrow!)

```
GET / HTTP/1.1
Host: example.org

HTTP/1.1 200 OK
Age: 501391
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Wed, 22 Apr 2020 10:24:07 GMT
Etag: "3147526947+ident"
Expires: Wed, 29 Apr 2020 10:24:07 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (bsa/EB17)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256

<!doctype html>
<html>
<head>
    <title>Example Domain</title>
```

The Uniform Interface Constraint

4. ***HATEOAS***: Hypermedia as the Engine of Application State

Our Menu

From the IoT to the WoT: Network-level and Application-level Interoperability

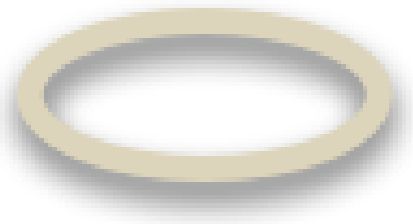
Tidbits from the Web Architecture

Lessons for (good) Web APIs

Hypermedia APIs

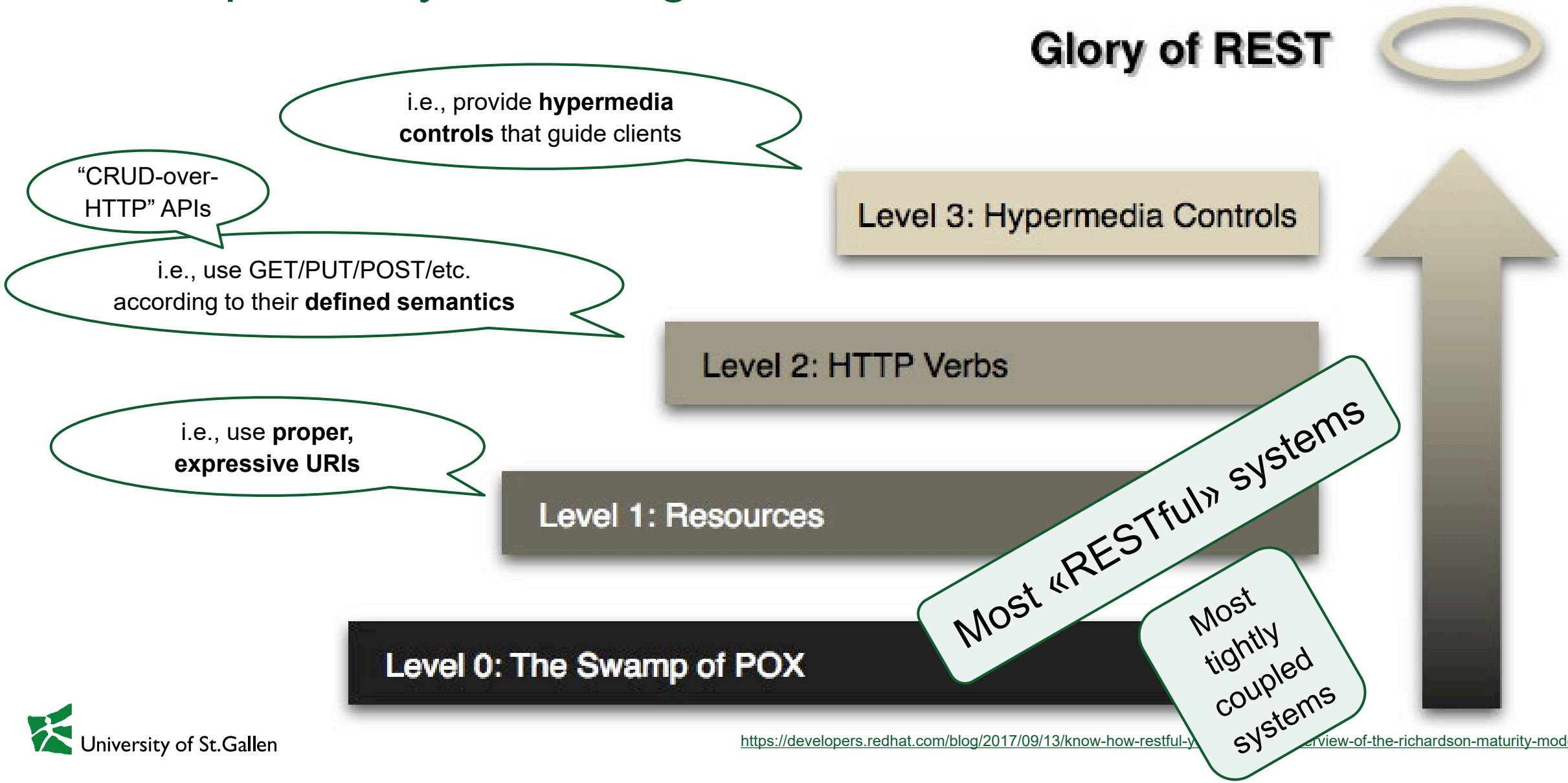
Interoperability and Integration on the Web

Glory of REST



https://developers.redhat.com/blog/2017/09/13/know-how-restful-your-api-is-an-overview-of-the-richardson-maturity-model/

Interoperability and Integration on the Web

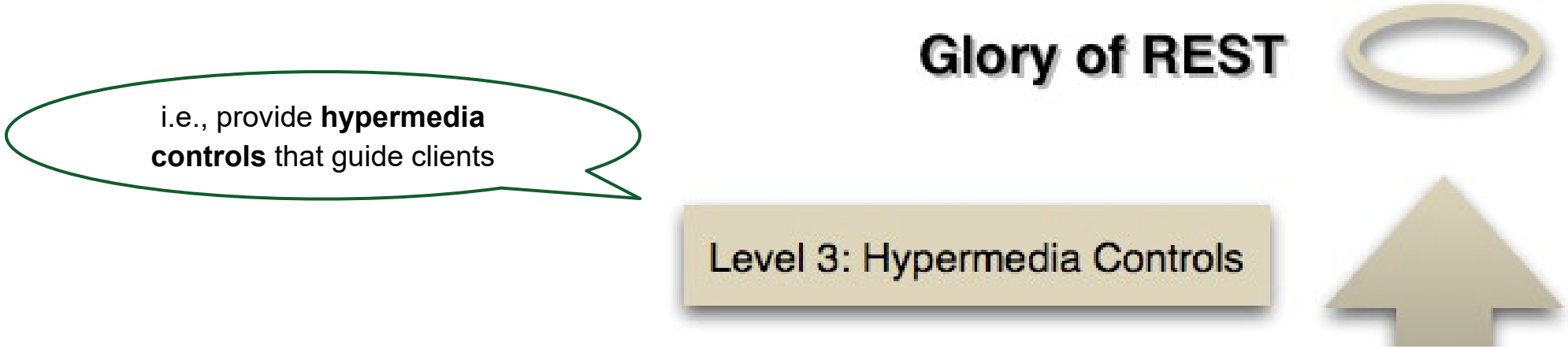


https://developers.redhat.com/blog/2017/09/13/know-how-restful-y erview-of-the-richardson-maturity-model/

Fundamental Challenge in HI and API Design

How to **facilitate the usage of applications** by human agents (“people”) and software agents (“machines”)?

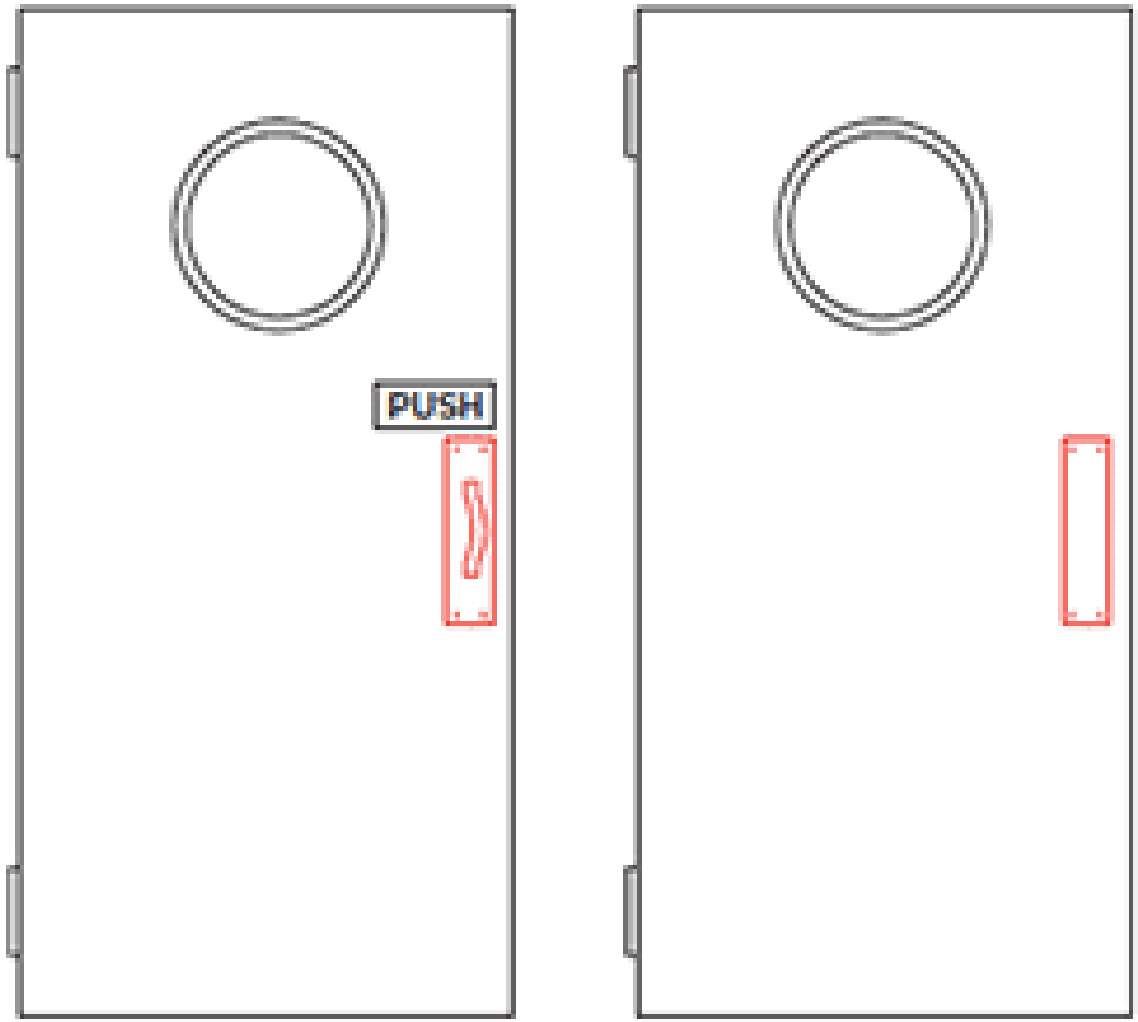
Hypermedia to enable (more) easily usable APIs?



Machines have only limited possibilities on the predominantly “human” Web

- Browsers are **made for people** and **operated by people**

How can we let machines **access information**?
How can we let machines **perform actions**?
How could machines **understand implications** of their actions? (-> Tuesday and Wednesday)

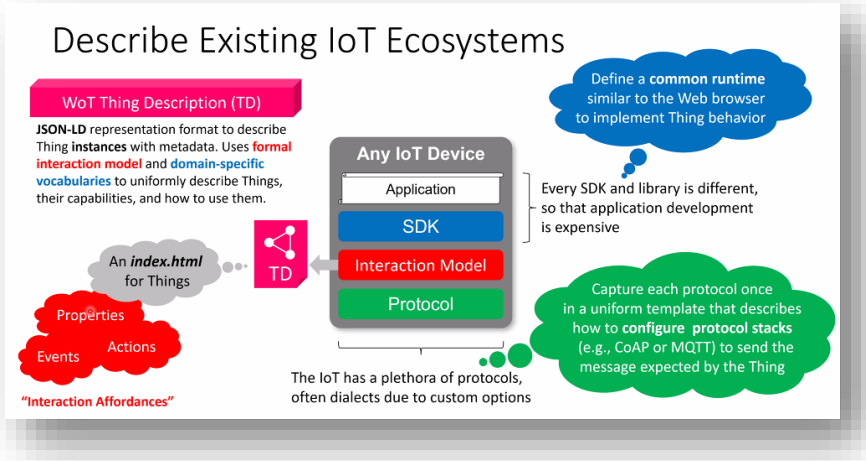


How can an application “afford” actions to its users?

How is this done in a Web context?

I am a hyperlink!

I am not a hyperlink.



Hyperlinks are the **core local affordance** that is provided to clients to help them navigate a hypermedia application!

This works great for people navigating hypermedia applications.
We **need to fully exploit this concept** for machines as well!

Hypermedia Driving Application State

Resource representations contain links to identified resources

Servers guide interactions by providing links (“HATEOAS”)

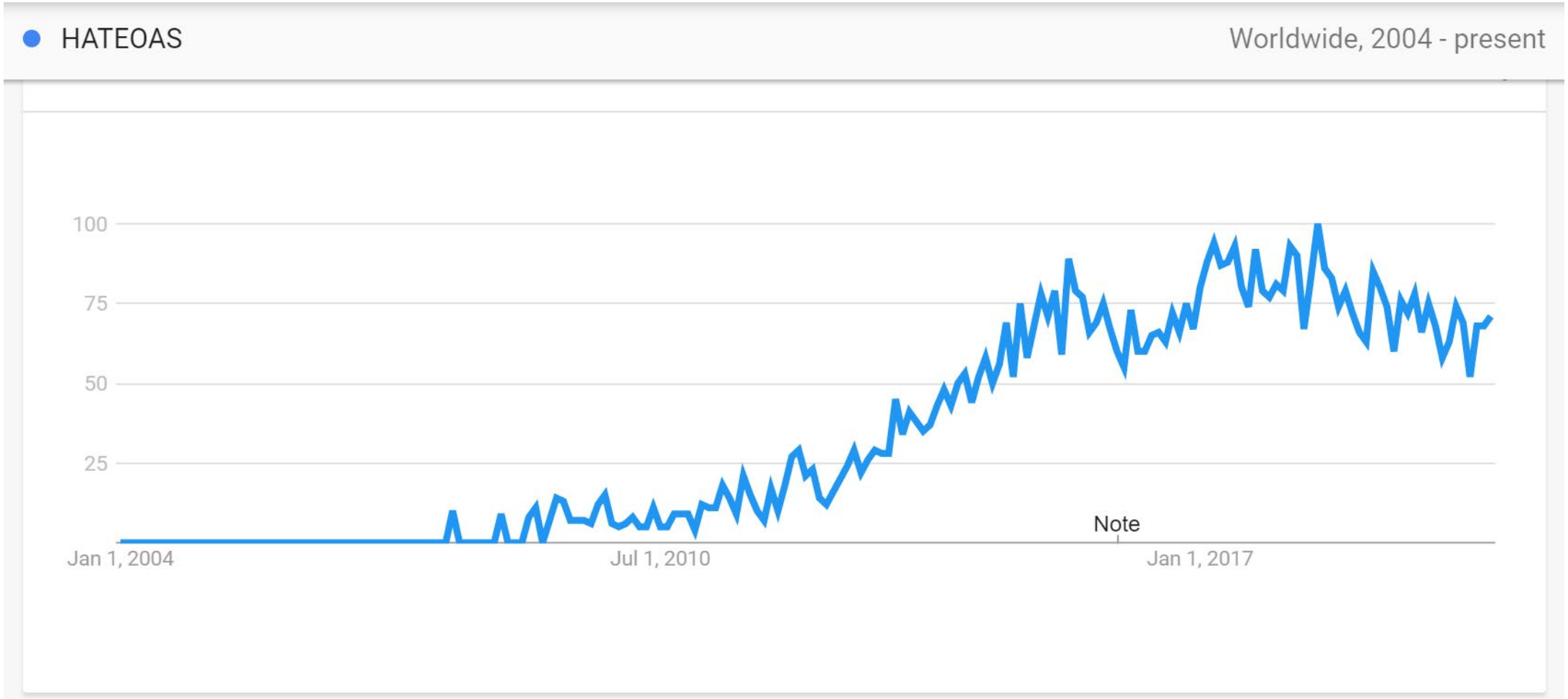
Hypermedia clients **navigate** instead of calling

- Representations contain information about **possible traversals**
- The client navigates to the next resource depending on **link semantics**
- Navigation can be **delegated** (e.g., for login processes) since all links use uniform identifiers

This is more important than it seems… especially for machines

- Server provides **local guidance** for its clients by providing links
- Links are **possible state transitions of the client/server application**

Hypermedia Driving Application State

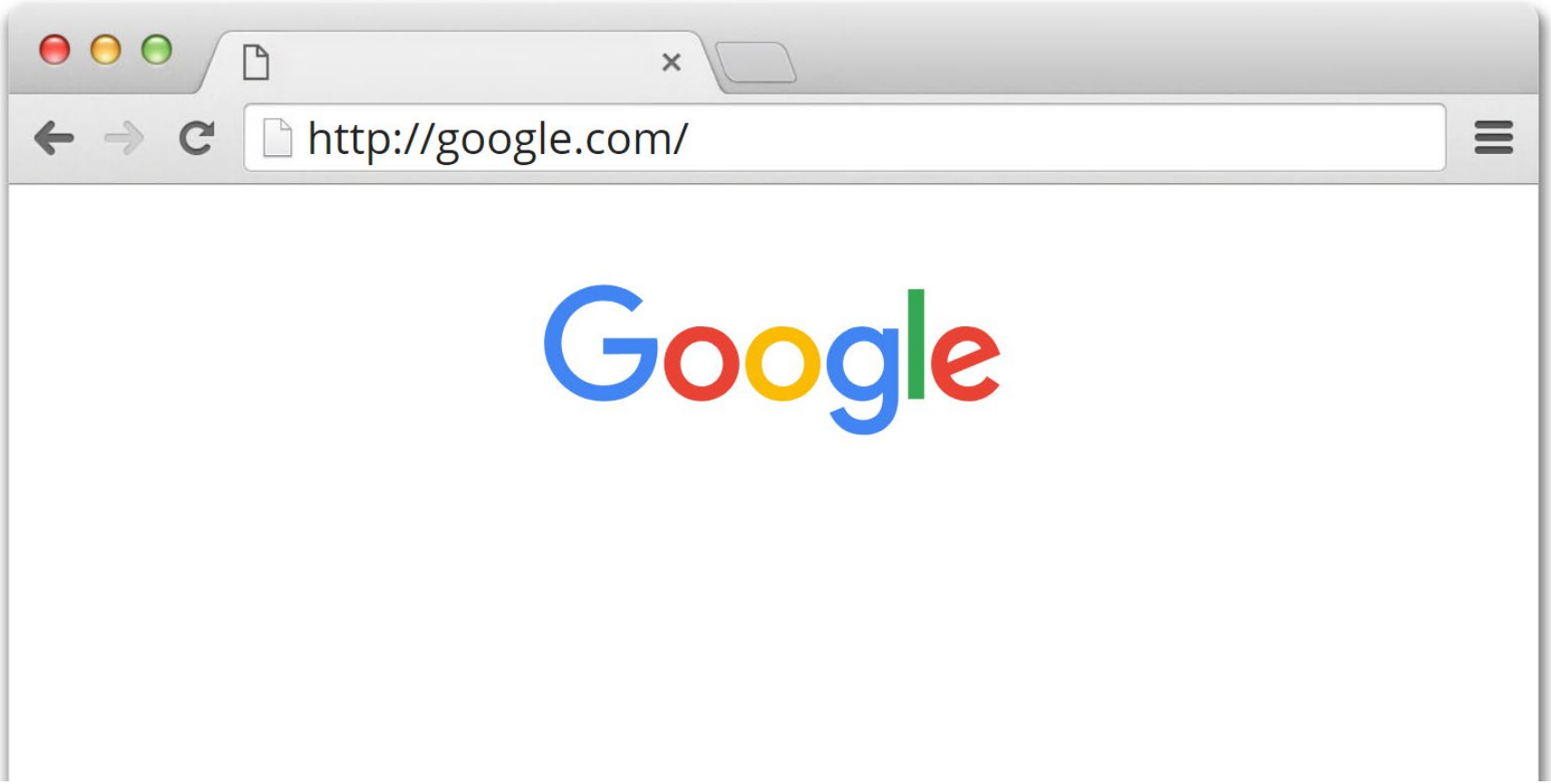


Hypermedia Driving Application State

Hypermedia is important for people!

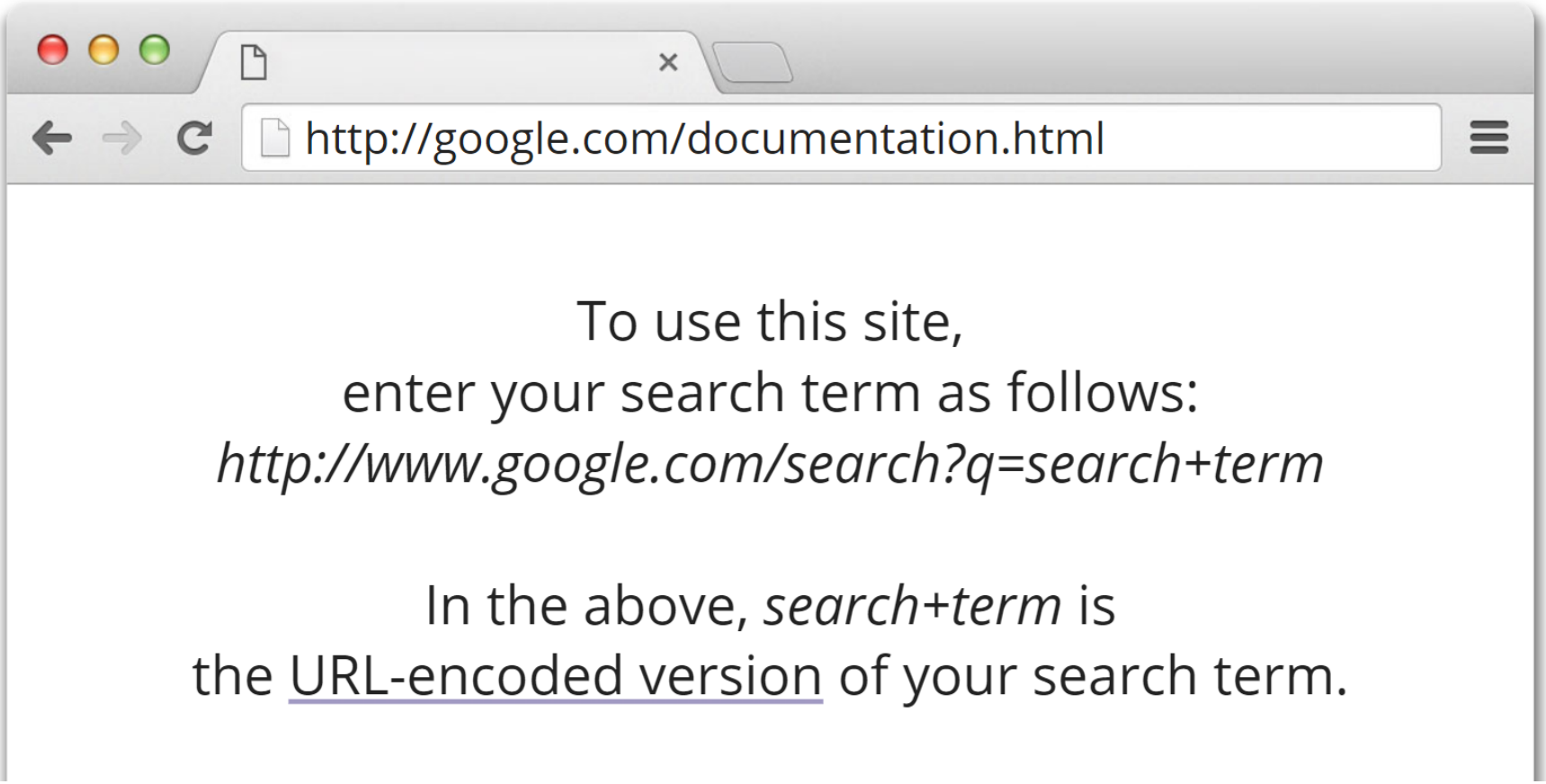
- How would we navigate the Web without it?

To understand hypermedia's importance, imagine using any website without it.

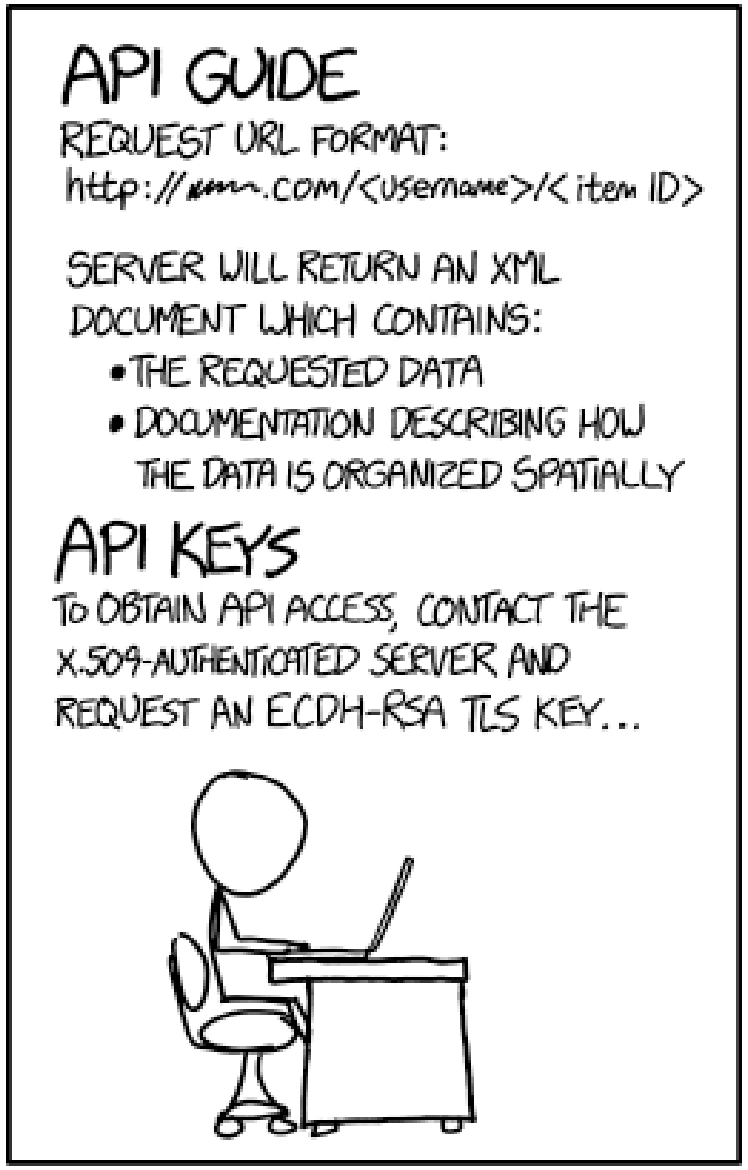


https://rubenverborgh.github.io/WebFundamentals/web-apis/#web-without-hypermedia

You'd need to read the documentation, and hard-code it in your implementation.

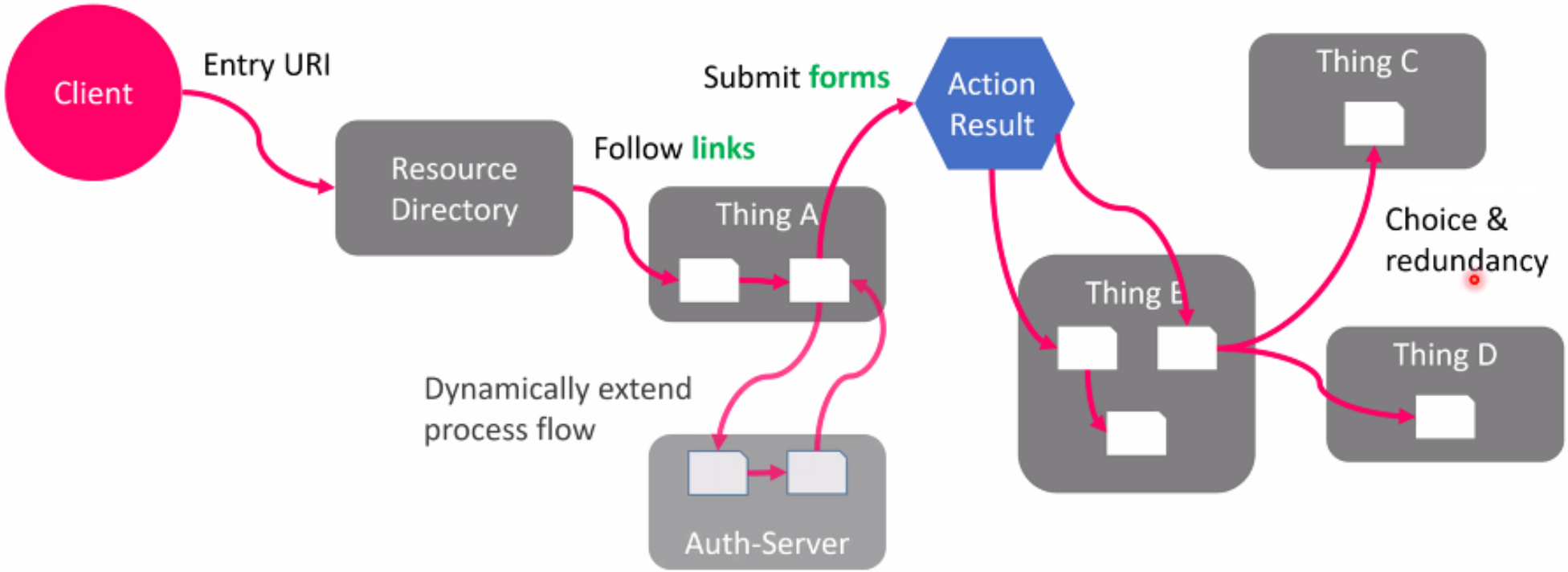


https://rubenverborgh.github.io/WebFundamentals/web-apis/#web-without-hypermedia



Looks like an **API description for people**!

Let Machines use Things Like We Browse the Web



Matthias' slide… ;-)

Hypermedia Driving Application State

With HATEOAS, a machine API **needs no syntactic documentation!**

- This is much closer to how **people navigate Web applications**!
- Adds robustness and flexibility to RESTful systems
- Links are discovered at run time! Clients will adapt if links change!
- This is why **Web browsers are general-purpose tools** and not specialized applications

Making **full use of the Web of Things** (incl. HATEOAS) for **devices and services…**

…facilitates the **autonomous usage** of their interfaces by **machine clients**!

…facilitates the creation of **flexible mashups** across services by **different** providers!

i.e. helps to **integrate** siloed systems!

Let‘s see a few **examples**…

Did you ever experience that the “back” button broke a hypermedia application?

e-banking, e-commerce, flights booking, etc.

Submission Form for the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

Home Log In PIN Refresh Help...

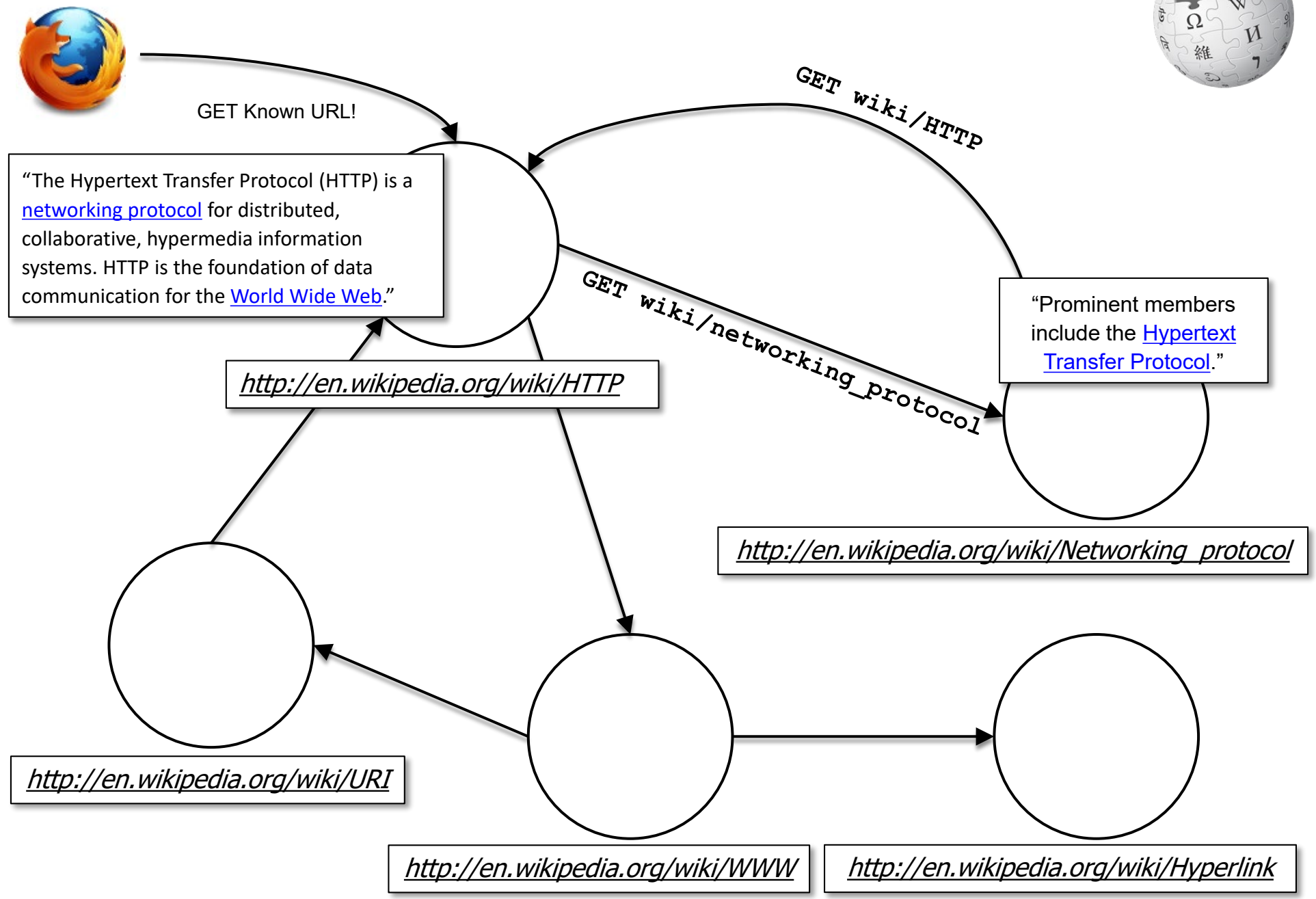
Warning: Test your pdf file before uploading it

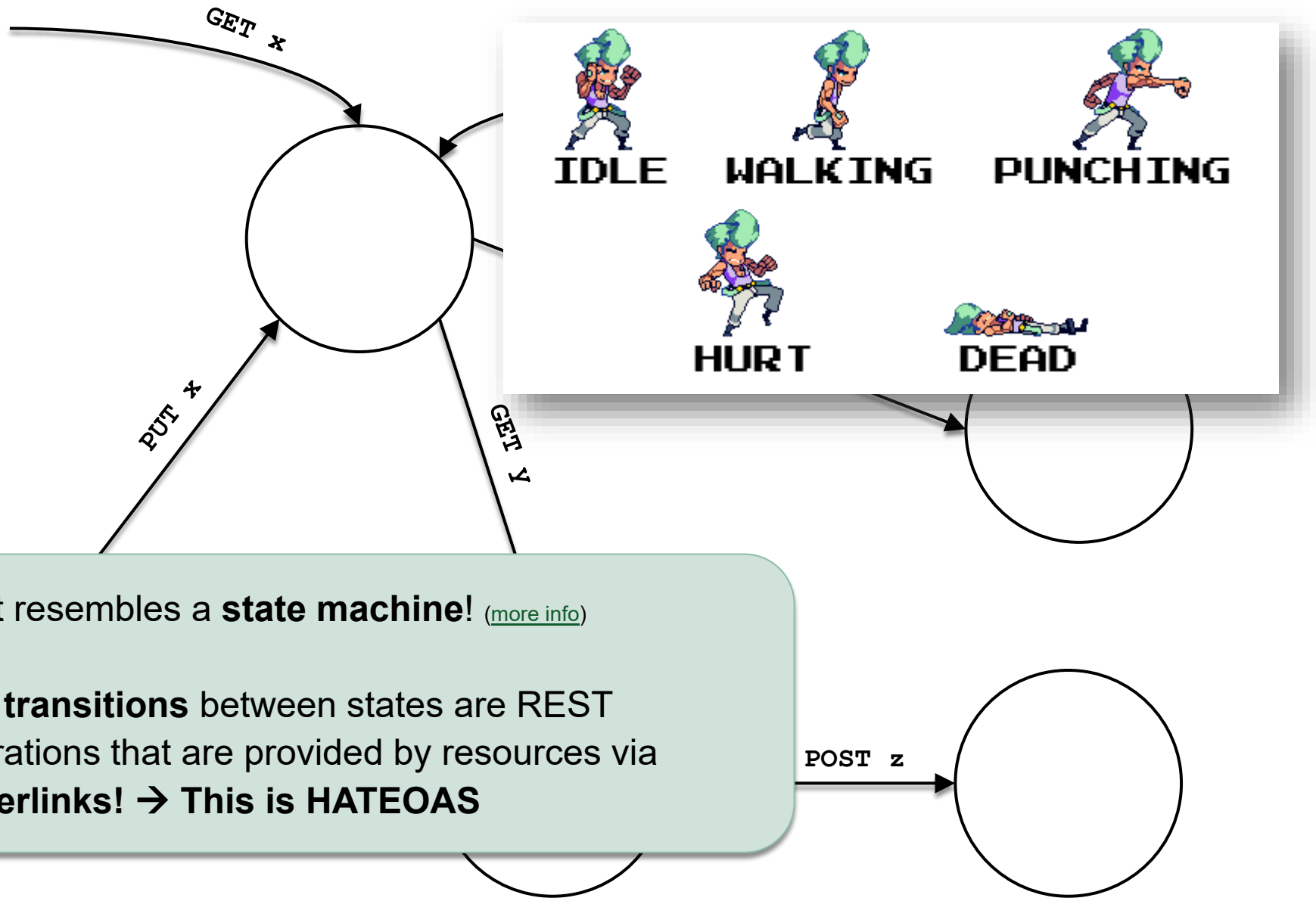
Cancel

Submit Late Breaking Results Poster to 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

Submission Information Section	Use this page to submit a contributed paper Complete the fields in this section and click on the Next button All fields marked with an asterisk * are required, including the keywords Warning: The browser Back button has been disabled. Do not use it during the submission
Type of submission	**Late Breaking Results Poster**
Title of the paper*	--Enter the title of the paper--
How many authors does the paper have?*	–Select a number–
	Follow the link PIN to look up or register your Personal Identificat of all your co-authors in a separate browser window List the PINs in the order in which the authors appear in the pape author. Note: Only the corresponding author may update or reupl
Keywords*	Click to choose an ordered set of keywords

«Warning: The browser Back button has been disabled. Do not use it during the submission.»


GET Known URL!
GET wiki/HTTP
"The Hypertext Transfer Protocol (HTTP) is a networking protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web."
GET wiki/networking_protocol
"Prominent members include the Hypertext Transfer Protocol."
http://en.wikipedia.org/wiki/HTTP
http://en.wikipedia.org/wiki/Networking_protocol
http://en.wikipedia.org/wiki/URI
http://en.wikipedia.org/wiki/WWW
http://en.wikipedia.org/wiki/Hyperlink



That resembles a **state machine**! (more info)

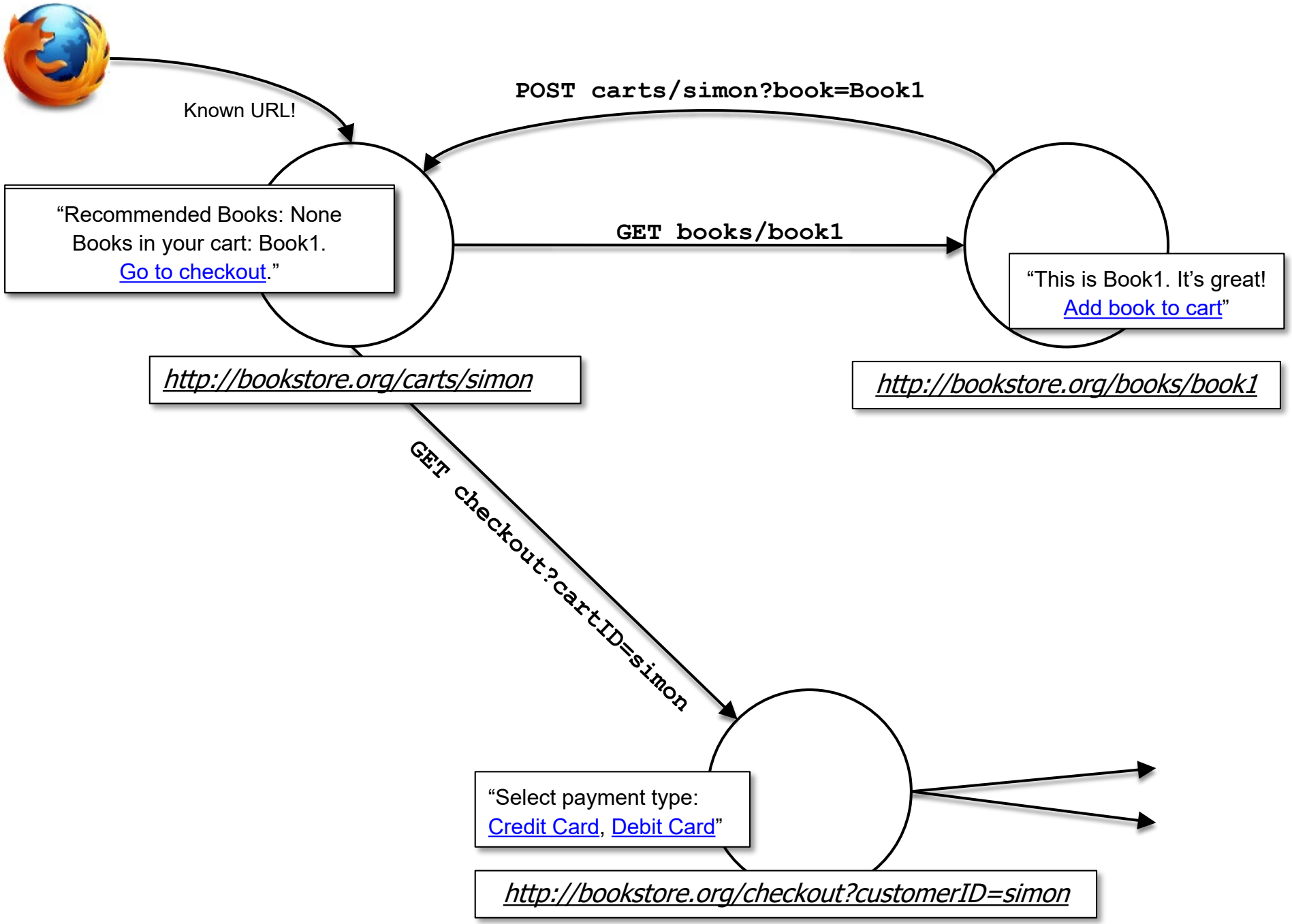
The **transitions** between states are REST operations that are provided by resources via **hyperlinks! → This is HATEOAS**

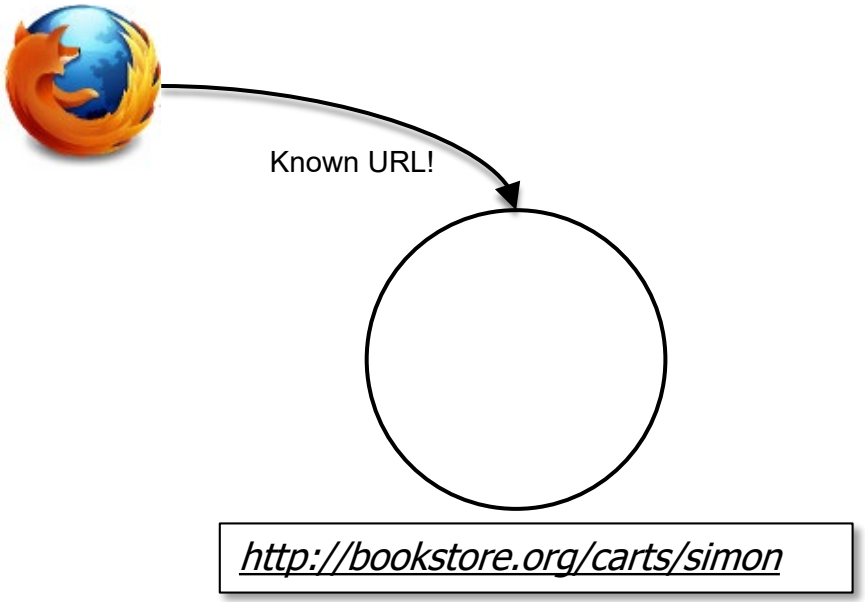
Did you ever experience that the “back” button broke a hypermedia application?

e-banking, e-commerce, flights booking, etc.

Clients that click the “**back**” button break the HATEOAS contract!

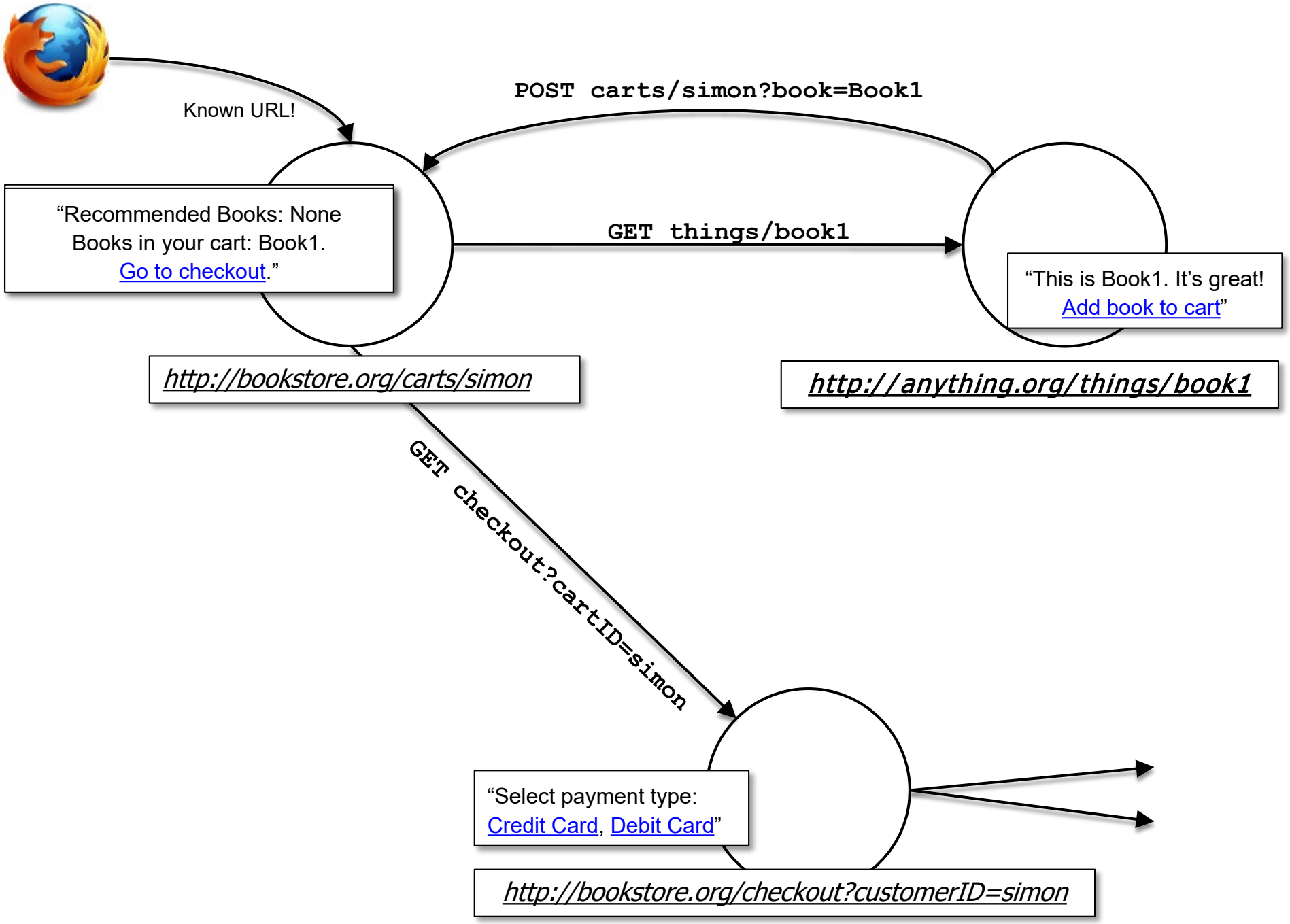
- The server **didn’t want a “back” link** to be available in the application!
- If it had wanted one, **there’d have been a hyperlink**!


Known URL!
POST carts/simon?book=Book1
"Recommended Books: None
Books in your cart: Book1.
Go to checkout."
GET books/book1
"This is Book1. It's great!
Add book to cart"
http://bookstore.org/carts/simon
http://bookstore.org/books/book1
GET checkout?cartID=simon
"Select payment type:
Credit Card, Debit Card"
http://bookstore.org/checkout?customerID=simon



http://bookstore.org/books/book1 → http://anything.org/things/book1

This (tiny) change would **break** any client that is statically compiled against the book store API….


Known URL!
POST carts/simon?book=Book1
"Recommended Books: None
Books in your cart: Book1.
Go to checkout."
GET things/book1
"This is Book1. It's great!
Add book to cart"
http://bookstore.org/carts/simon
http://anything.org/things/book1
GET checkout?cartID=simon
"Select payment type:
Credit Card, Debit Card"
http://bookstore.org/checkout?customerID=simon

Hypermedia Driving Application State

HATEOAS is what adds **robustness and flexibility** to REST systems

- Links are discovered by clients **at runtime**
- Clients will **automatically adapt** if links change!
- This is why **browsers are general-purpose tools** and not specialized applications

This means that hypermedia applications can change at runtime without breaking clients!

Browsers are not compiled against Hypermedia APIs!

Enable Machines to use IoT Functionality

All possible **state transitions** of an application are under control of the server. To **guide** clients in applications, the server provides **hyperlinks** that they may follow

Clients only follow provided hyperlinks! “follow your nose”

– **Take Care!** What if a client wants to “buy a book”, but only finds a link “Add to shopping cart”?

Enable Machines to use IoT Functionality

Can a machine client “follow its nose”?

What is required to “follow your nose”?

How to **combine services** from **different providers**?

- They don’t know about each other!
- This means that they cannot include links to each other!

It boils down to:

- Do machine clients **know their strategic «goal»**, i.e. what they want to accomplish ultimately?
- Can they **find the individual steps** through the hypermedia maze to reach that goal?

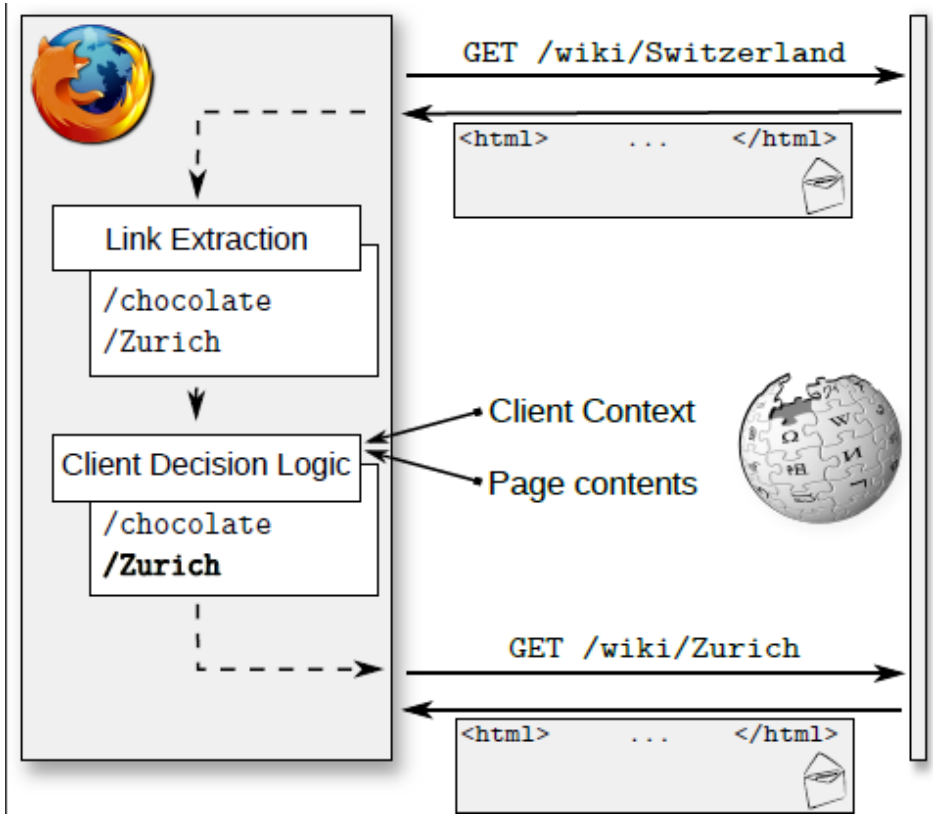
Imagine programming a client so that it uses Web APIs like you would…

Machines interacting with Web APIs…

How does a person decide which link to follow on a page?

```
<a href="somewhere">something</a>
```

1. Read “something”
2. Think “something” is really interesting
3. Click “something”



This isn’t trivial for machines at all!

- The machine would need a **deeper understanding of the high-level semantics of the domain in question** that we acquire by **years of experience** in the real world
- Semantically annotating links in a machine-readable way is **no easy way out** in the global Web…

Machines interacting with Web APIs…

Example: Teach a machine client to **buy a book on Amazon given its ISBN**

Let's make it even simpler:

- We give the machine the URL of the product
- And we give it the instruction to "buy the product"

The machine is able to discover the link "Add to Cart"

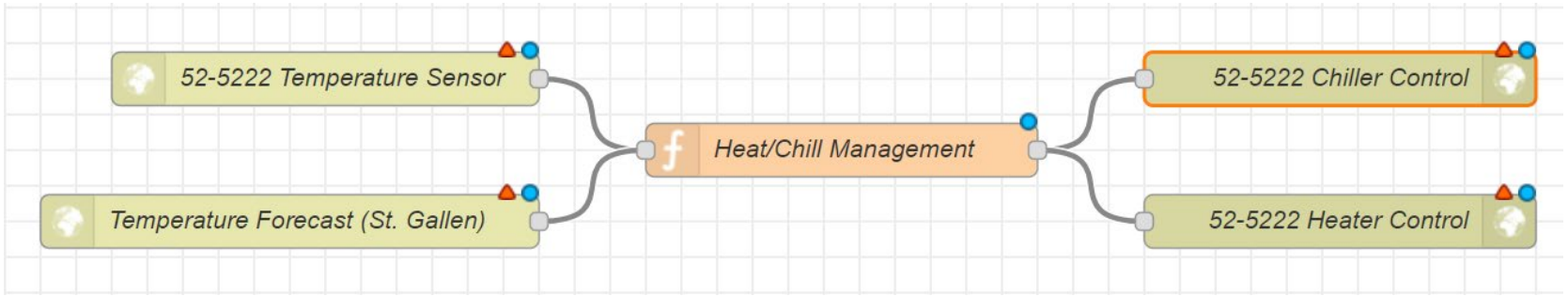
- But how does it know **that this link is connected to its goal** of "buy the product"?
- And if it knew that, how does it know that it needs to "**Go to checkout**" next?
- Answer: **current research!**

Machines interacting with Web APIs…

"Node-RED is a programming tool for wiring t… devices, APIs and online services in n… ways.

It provides a browser-bas… makes it easy to wire together flows using the wide range of nodes in the … that can be deployed to its runtime in a single-click."

i.e. Manually Engineered Interactions



Goal

Can we have **software create and execute such plans**, given a **functional goal**?

e.g., "Keep this room and all similar rooms in their current condition!"

Any Questions / Comments / Doubts / Concerns?

Conclusion

Web-based application-layer integration enables built-in **scalability, flexibility, evolvability, usability** – in the scope of the **Web of Things** these properties foster **open innovation** in the physical space as well.

Uniform interfaces, in particular the proper usage of **hypermedia**, support **flexibility and robustness**, as they enable clients discover state transitions at runtime.

These properties are important for humans and even more so **for interacting physical devices**

- Machines **cannot easily recover** if something goes wrong in an interaction
- A shared application-layer interaction style provides the **low-level interaction semantics** for all services on the Web
- **Such clearly defined protocol semantics** are a good basis for **enabling autonomous agents on the Web, i.e. to** make the Web (of Things) **navigable for machine clients**
- We then “only” need to solve the problem of **interoperability on the level of high-level domain semantics**

Zoomed-out Conclusion

Web-based application-layer integration enables built-in **scalability, flexibility, evolvability, usability** – in the scope of the **Web of Things** these properties foster **open innovation** in the physical space as well.

Words of Warning: **«Openness» does not only have a technology dimension!**

- **Dominant online services** (e.g., Facebook) replacing the open Web with **walled gardens**
- Platforms try to **own your data**, but are **not themselves interoperable** (GDPR helps)
- **Net Neutrality repeals threaten open innovation**

https://www.techdirt.com/articles/20140604/05594427450/entrepreneurs-explain-how-end-net-neutrality-would-mean-their-startups-dont-exist.shtml

Outlook: Autonomous Interactions?!

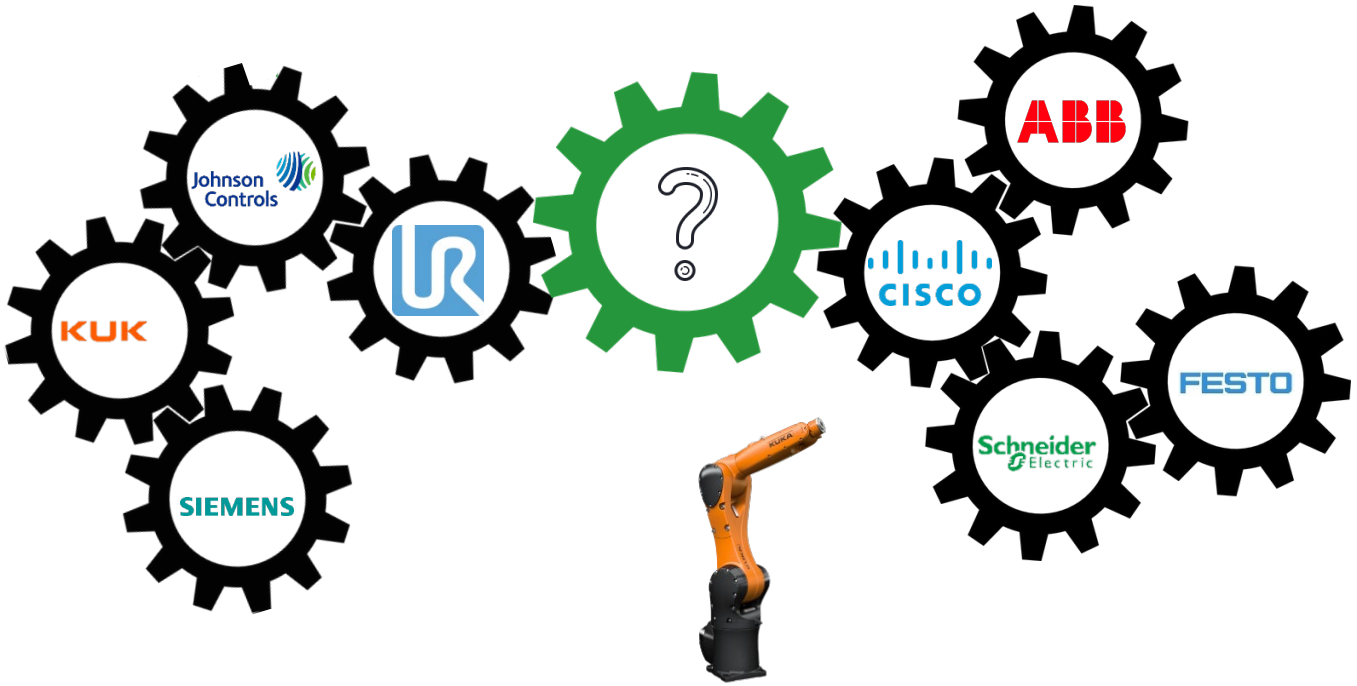
We know that **we today can scalably integrate devices with little manual effort** on the network and application levels thanks to the Internet (Protocol) and the Web (Protocols)

Can we also **scalably integrate their functionality**?

….can we **reduce** the integration and engineering effort in Web of Things mashups?

….can we even **automate** this engineering?

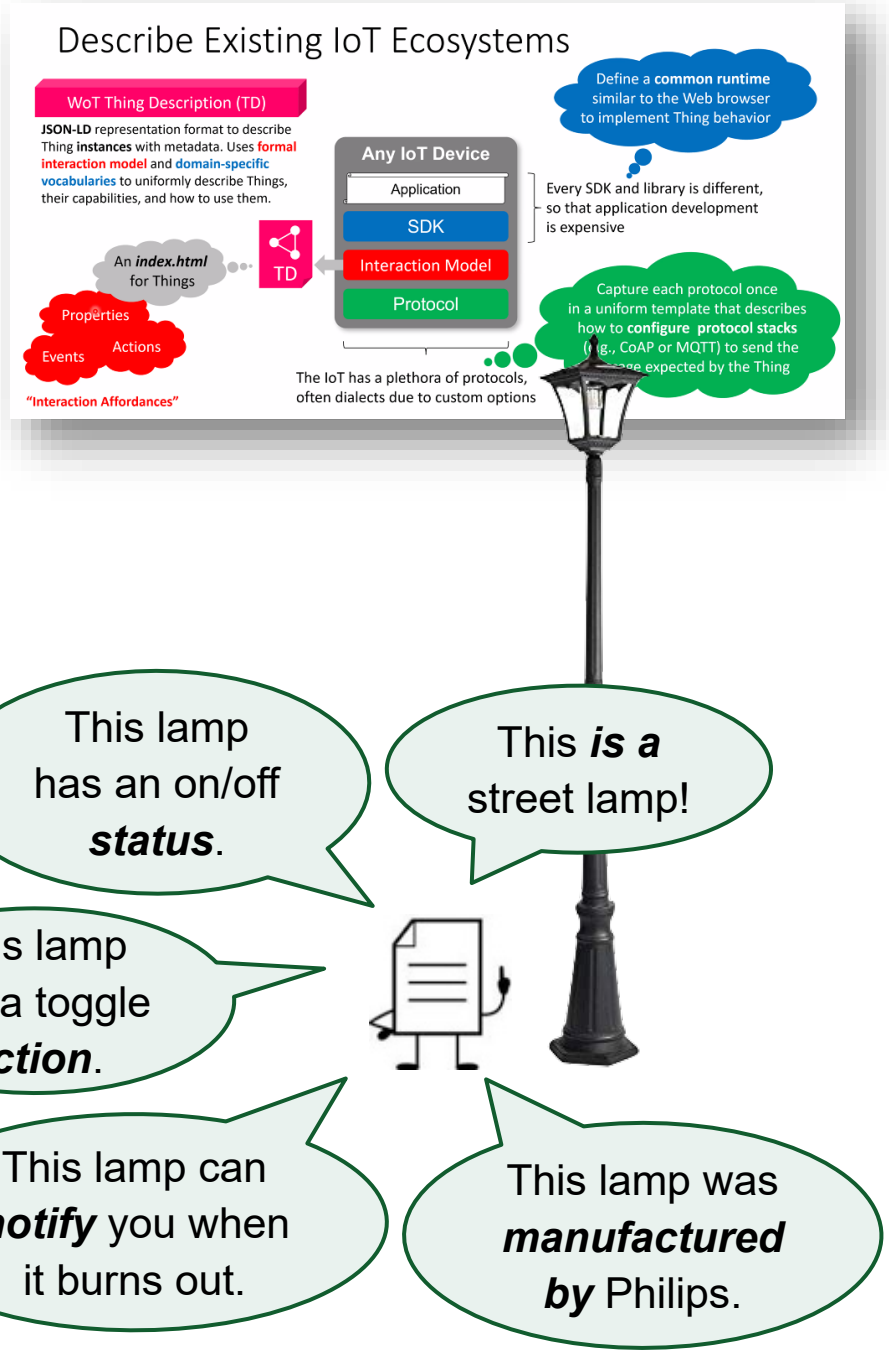
This would enable devices and services to **autonomously interact** with one another!



Outlook: Autonomous Interactions?!

The W3C Web of Things Thing Description

- general and domain-specific **metadata**
- **links** to related Things
- 3 types of **interaction affordances**:
  - **properties**: expose the state of a Thing; can be *observable*
  - **actions**: allow to manipulate the state of a Thing (e.g., toggle on/off) or trigger internal processes (e.g., dim a lamp over time)
  - **events**: expose state transitions of a Thing
- **hypermedia controls**:
  - convey to Consumers how to activate an **affordance** (e.g., translate affordances to HTTP requests)
  - 2 types: **hyperlinks** for navigation and **forms** for any kind of interaction affordance



Outlook: Autonomous Interactions?!

Machine-understandable description of a Thing that includes:

People rely on text and images that convey meaning: `<a href="somewhere">**something**</a>`

Machines rely on **structured data** that conveys **meaning**.

Tobias and Jomi know!
(and will tell you)

So how can we specify the **semantics**?

And how can these systems be used by **autonomous software agents**?