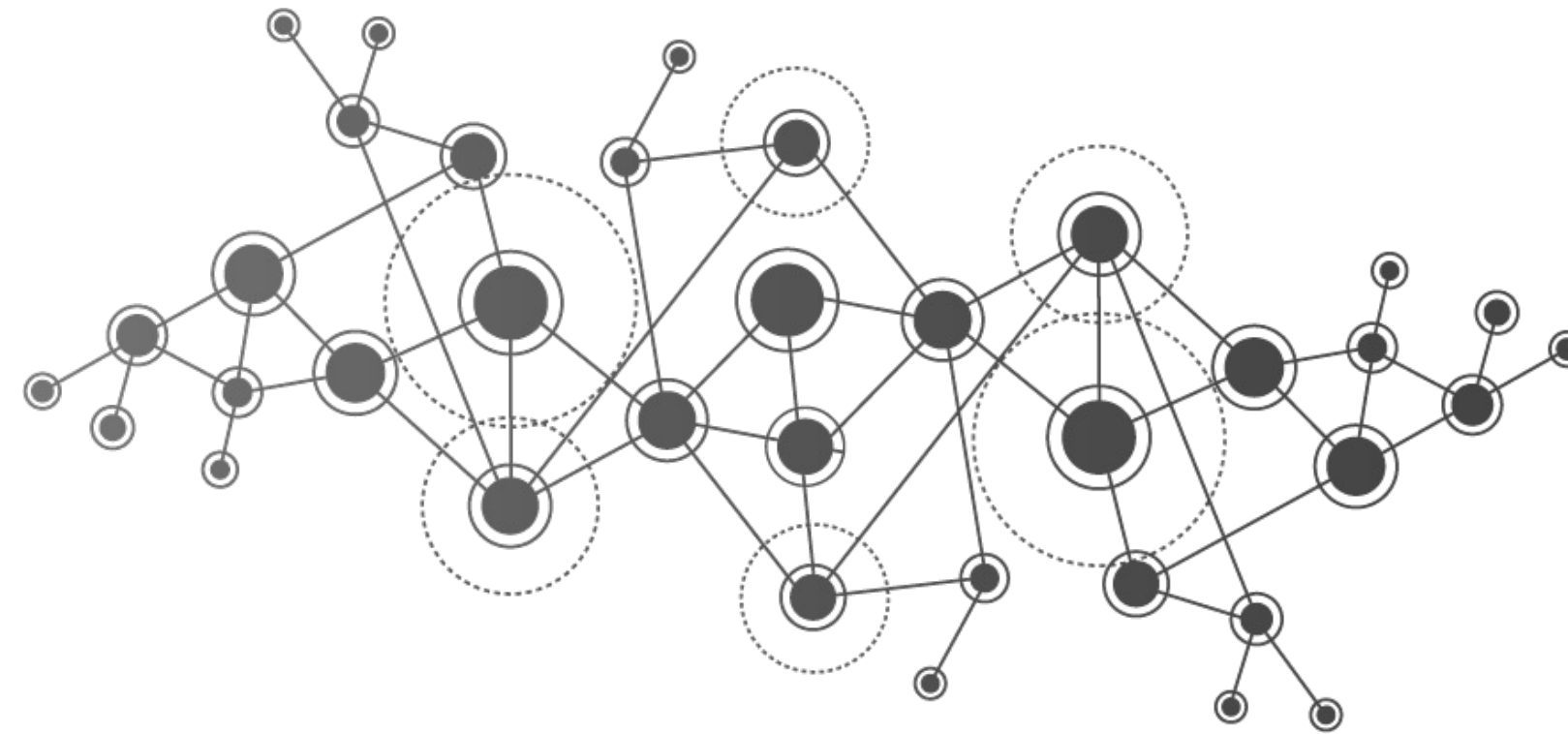




UNIMORE

UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA



Intelligent Internet of Things

Resource & Service Discovery

Prof. Marco Picone

A.A 2023/2024



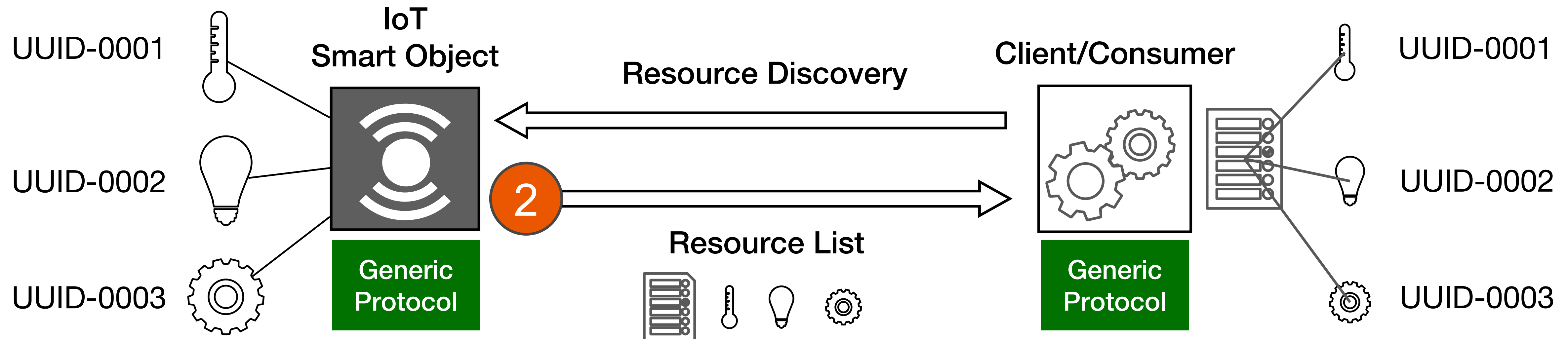
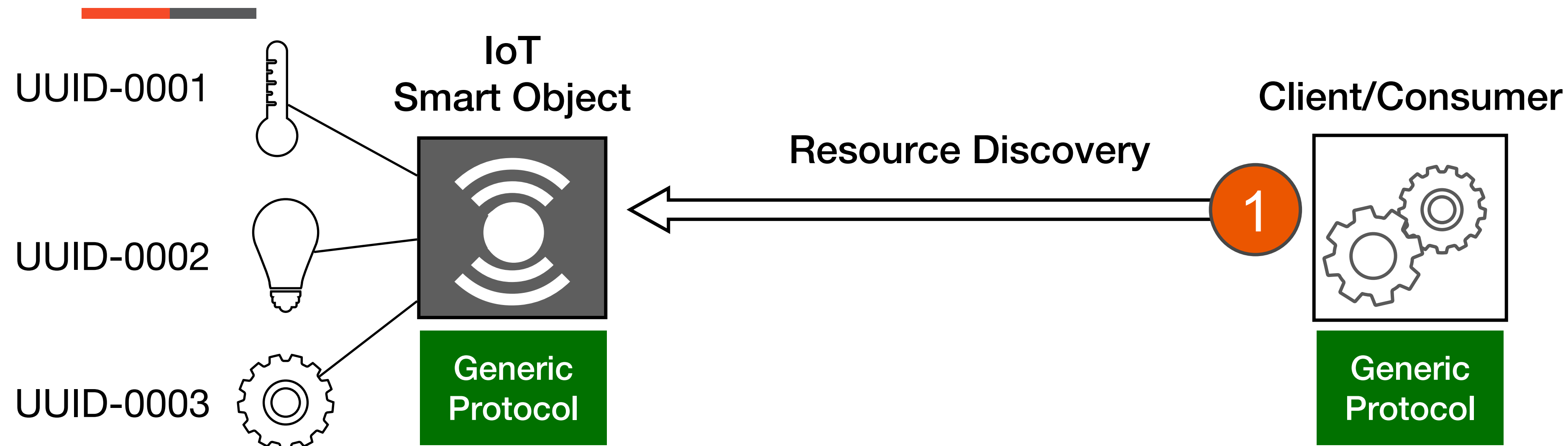
UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Resource & Service Discovery

- Resource Discovery
- CoAP - Resource Discovery
- Web Linking
- CoRE Link Format
- CoRE Interfaces
- Sensor Markup Language (SenML)
- CoRE Resource Directory
- Service Discovery
- ZeroConf
- UPnP
- URI Beacon & Physical Web

- The concept of **Resource Discovery** is associated to the possibility for a consumer/client application (*through a specific Application Protocol*) to find the list of available resource hosted and managed by an IoT Smart Object
- The final aim is to simplify and automate the seamless interaction and interoperability among objects without any kind of prior knowledge, configuration, other without human intervention
- The concept of modelling and designing IoT Applications around the concept of Resources is mostly associated to RESTful Protocols (such as CoAP) but can the **Resource Discoverability** can be conceptually generalized to any IoT protocols
- *CoAP natively include in the standard a communication pattern to support Resource Discovery in any standard CoAP Smart Object*
- Design and define a similar flow for other protocols such as MQTT is at the same time challenging and fundamental for a seamless interaction between heterogeneous things, people and services

Resource Discovery



- Resource discovery is based on discovering links to resources hosted by a CoAP service endpoint

Standard mechanism based on link representations

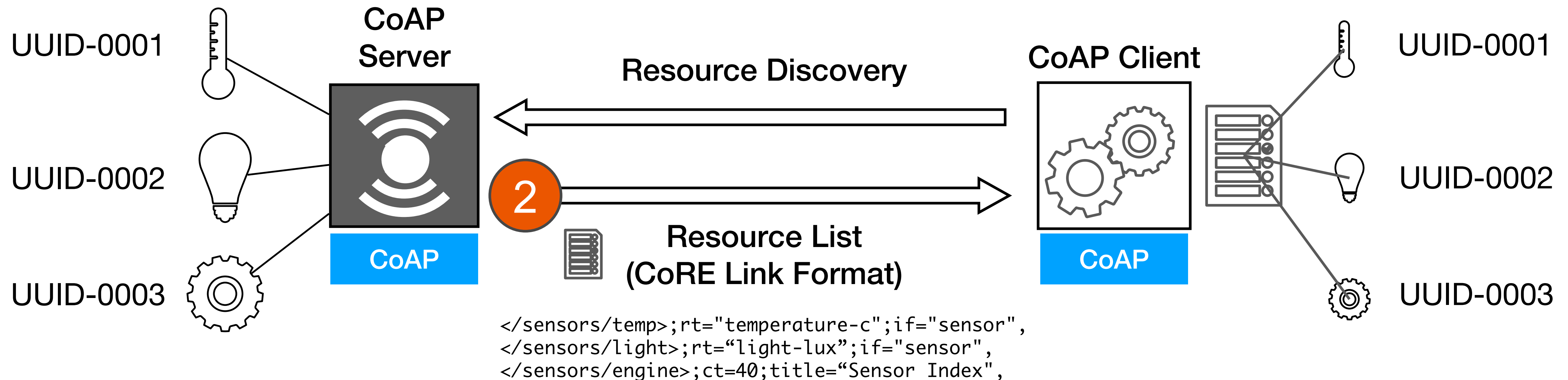
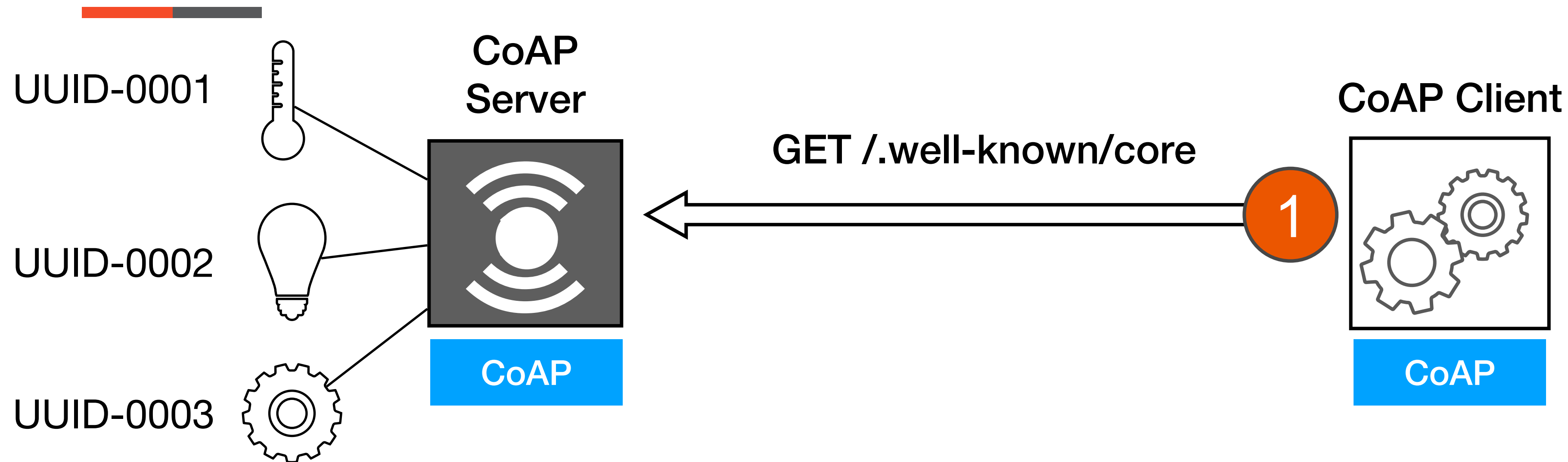
- Web Linking (RFC 5988)
- CoRE Link Format (RFC 6690)

URI

List of attributes (e.g., obs means observable; rt indicates the resource type)

- A well-known relative URI **"/.well-known/core"** is defined as a default entry point for requesting the list of links about resources hosted by a server and thus performing CoRE Resource Discovery
- A GET request to any CoAP server for the resource **/.well-known/core** results in a response that includes a list of links (formatted according to CoRE Link Format - RFC 6690)
- *This specification is applicable for use with Constrained Application Protocol (CoAP), HTTP, or any other suitable web transfer protocol*

CoAP - Resource Discovery



- A framework for indicating the relationships between web resources
- A type link consists of:
 - Context URI – What the link is from
 - Relation Type – Indicates the semantics of the link
 - Target URI – What the link is too
 - Attributes – Key value pairs describing the link or its target
- Relations include e.g. copyright, author, chapter, service etc.
- Attributes include e.g. language, media type, title etc.
- Example in HTTP Link Header format:

Link: <http://example.com/TheBook/chapter2>; rel="previous"; title="previous"

- The Constrained RESTful Environments (CoRE) implements the Representational State Transfer (REST) architecture in a suitable version in order to support constrained Smart Object and networks
- The discovery of resources hosted by a server is fundamental for machine-to-machine scenarios
- The CoRE Link Format (RFC 6690 - <https://tools.ietf.org/html/rfc6690>) specification defines the discovery of resources hosted by a constrained web server, their attributes, and other resource relations as CoRE Resource Discovery
- The main function of such a discovery mechanism is to **provide Universal Resource Identifiers** (URIs, called links) for the resources hosted by the Smart Object
- The **resources** are complemented by **attributes** about those resources and **possible further link relations**
- In CoRE, this **collection of links** is represented as a resource of its own (different from the HTTP headers delivered with a specific resource)

- Resource Discovery can be performed either unicast or multicast.
- When a server's IP address is already known, either a priori or resolved via the Domain Name System (DNS), unicast discovery is performed in order to locate the entry point to the resource of interest.
- In CoRE Link Format, this is performed using a **GET to `"/.well-known/core"`** on the server, which returns a payload in the CoRE Link Format.
- **A client would then match the appropriate Resource Type, Interface Description, and possible media type for its application.**
- These attributes may also be included in the query string in order to filter the number of links returned in a response.

CoRE Link Format (RFC 6690) - Format

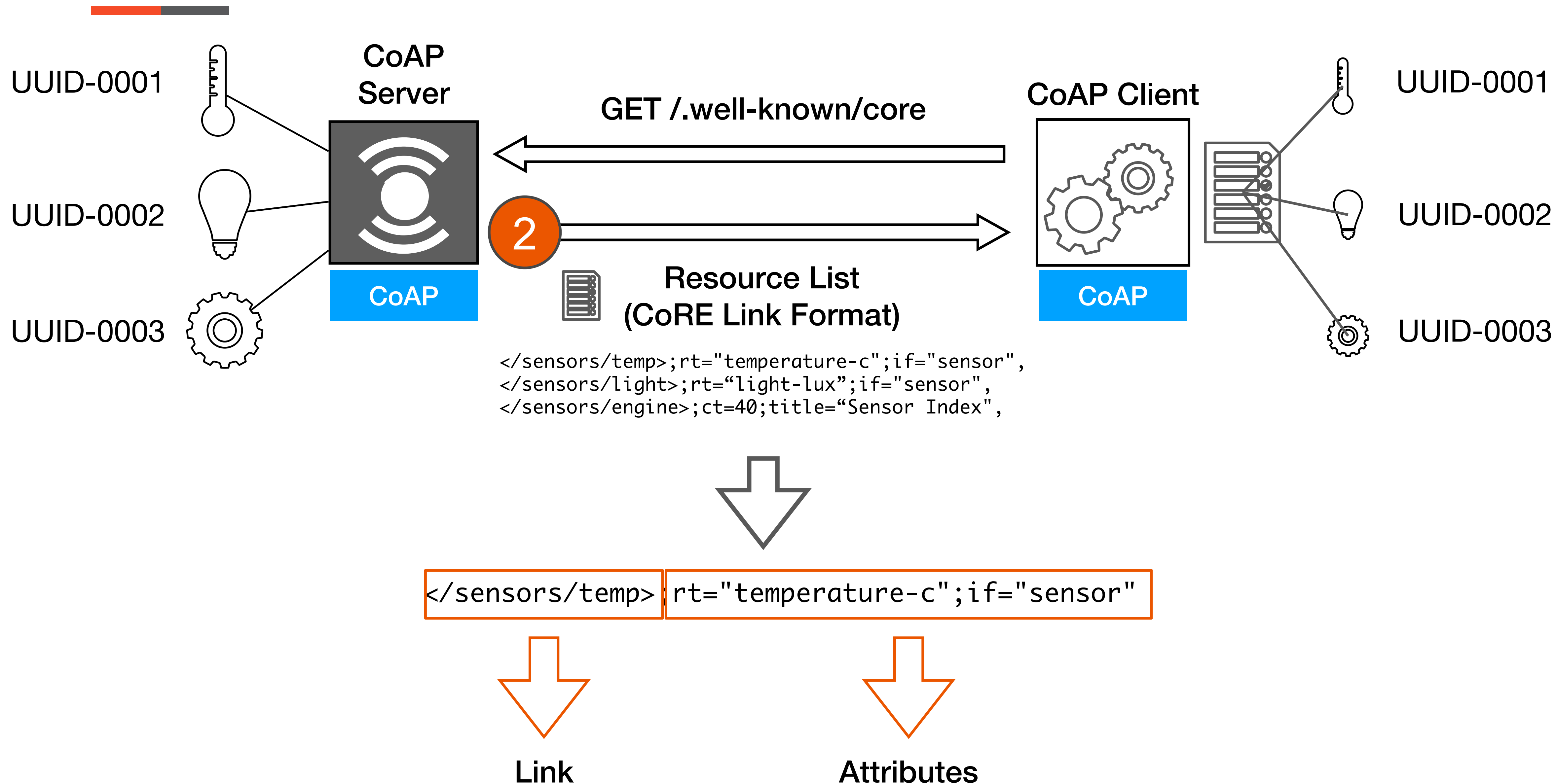
- The CoRE Link Format defines the Internet media type 'application/link-format' encoded as UTF-8
- UTF-8 data can be compared bitwise, which allows values to contain UTF- 8 data without any added complexity for constrained nodes
- **GET /.well-known/core?optional_query_string**
- Returns a link-header style format
- URL, relation, type, interface, content-type etc.

```
Link          = link-value-list
link-value-list = [ link-value *[ "," link-value ] ]
link-value     = "<" URI-Reference ">" *( ";" link-param )
link-param     = ( ( "rel" "=" relation-types )
                  / ( "anchor" "=" DQUOTE URI-Reference DQUOTE )
                  / ( "rev" "=" relation-types )
                  / ( "hreflang" "=" Language-Tag )
```

```
                  / ( "media" "=" ( MediaDesc
                                / ( DQUOTE MediaDesc DQUOTE ) ) )
                  / ( "title" "=" quoted-string )
                  / ( "title*" "=" ext-value )
                  / ( "type" "=" ( media-type / quoted-mt ) )
                  / ( "rt" "=" relation-types )
                  / ( "if" "=" relation-types )
                  / ( "sz" "=" cardinal )
                  / ( link-extension ) )
link-extension = ( parmname [ "=" ( ptoken / quoted-string ) ] )
                  / ( ext-name-star "=" ext-value )
ext-name-star  = parmname "*" ; reserved for RFC-2231-profiled
                  ; extensions. Whitespace NOT
                  ; allowed in between.

ptoken         = 1*ptokenchar
ptokenchar     = "!" / "#" / "$" / "%" / "&" / "'" / "("
                  / ")" / "*" / "+" / "-" / "." / "/" / DIGIT
                  / ":" / "<" / "=" / ">" / "?" / "@" / ALPHA
                  / "[" / "]" / "^" / "_" / "`" / "{" / "|"
                  / "~" / " "
media-type     = type-name "/" subtype-name
quoted-mt      = DQUOTE media-type DQUOTE
relation-types = relation-type
                  / DQUOTE relation-type *( 1*SP relation-type ) DQUOTE
relation-type  = reg-rel-type / ext-rel-type
reg-rel-type   = LOALPHA *( LOALPHA / DIGIT / "." / "-" )
ext-rel-type   = URI
cardinal       = "0" / ( %x31-39 *DIGIT )
LOALPHA        = %x61-7A ; a-z
quoted-string  = <defined in [RFC2616]>
URI            = <defined in [RFC3986]>
URI-Reference  = <defined in [RFC3986]>
type-name      = <defined in [RFC4288]>
subtype-name   = <defined in [RFC4288]>
MediaDesc      = <defined in [W3C.HTML.4.01]>
Language-Tag   = <defined in [RFC5646]>
ext-value      = <defined in [RFC5987]>
parmname       = <defined in [RFC5987]>
```

CoRE Link Format (RFC 6690) - Format



- CoRE Link Attributes attributes describe information useful in accessing the target link of the relation and, in some cases, can use the syntactical form of a URI
- RFC6690 = Simple semantics for machines
 - IANA registry for rt= and if= parameters
- Resource Type (rt=)
 - What is this resource and what is it for?
 - e.g. Device Model could be rt="ipso.dev.mdl"
- Interface Description (if=)
 - How do I access this resource?
 - e.g. Sensor resource accessible with GET if="core.s"
- Content Type (ct=)
 - What is the data format of the resource payloads? e.g. text/plain (0)

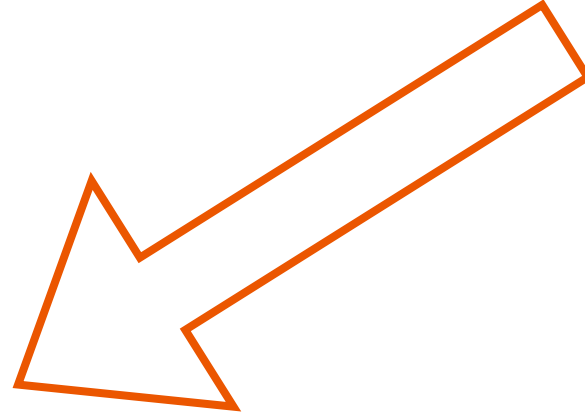
Resource Type 'rt' Attribute

- The Resource Type 'rt' attribute is an opaque string used to assign an application-specific semantic type to a resource.
- Multiple Resource Types MAY be included in the value of this parameter, each separated by a space, similar to the relation attribute
- The Resource Type attribute is not meant to be used to assign a human-readable name to a resource. The "title" attribute defined in [Web Linking RFC5988] is meant for that purpose.
- The Resource Type attribute MUST NOT appear more than once in a link.

REQ: GET /sensors

RES: 2.05 Content

</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor"



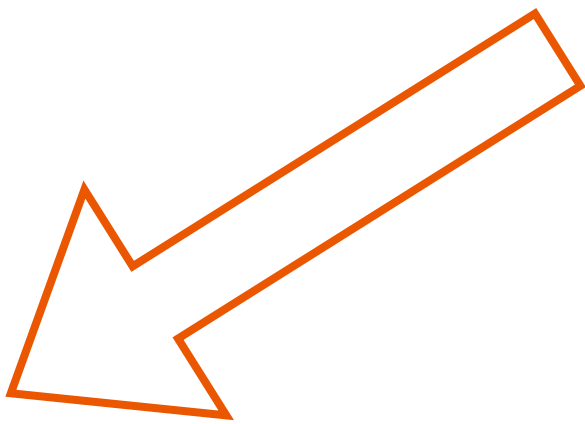
Interface Description 'if' Attribute

- The Interface Description 'if' attribute is an opaque string used to provide a name or URI indicating a specific interface definition used to interact with the target resource. It is meant to describe the generic REST interface to interact with a resource or a set of resources.
- A single Interface Description can be potentially be reused by different Resource Types. For example, the Resource Types "outdoor-temperature", "dew-point", and "rel-humidity" could all be accessible using the same Interface Description defining how to interact with that specific group of sensors
- Multiple Interface Descriptions MAY be included in the value of this parameter, each separated by a space, similar to the relation attribute.
- The Interface Description could be, for example, the URI of a Web Application Description Language (WADL) [WADL] definition of the target resource. Or it can be associate to the CoRE Interfaces standard (see next slides ...)
- The Interface Description attribute MUST NOT appear more than once in a link.

REQ: GET /sensors

RES: 2.05 Content

</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor"



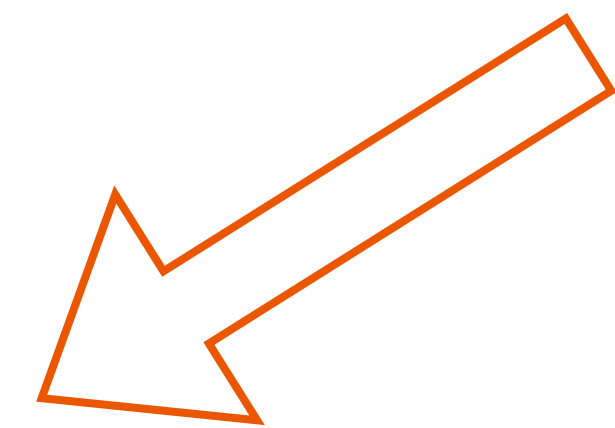
Maximum Size Estimate 'sz' Attribute

- The maximum size estimate attribute 'sz' gives an indication of the maximum size of the resource representation returned by performing a GET on the target URI
- For links to CoAP resources, this attribute is not expected to be included for small resources that can comfortably be carried in a single Maximum Transmission Unit (MTU) but should be included for resources larger than that
- The maximum size estimate attribute must not appear more than once in a link.
- Note that there is no defined upper limit to the value of the 'sz' attributes. Implementations must be prepared to accept large values.

```
REQ: GET /.well-known/core?rt=firmware
```

```
RES: 2.05 Content
```

```
</firmware/v2.1>;rt="firmware";sz=262144
```




- A web link [RFC5988] to a resource accessible over CoAP (for example, in a link-format document [RFC6690]) MAY include the target attribute **"obs"**
- The "obs" attribute, when present, is a hint indicating that the destination of a link is useful for observation and thus, for example, should have a suitable graphical representation in a user interface.
- Note that this is only a hint; it is not a promise that the Observe Option can actually be used to perform the observation.
- A client may need to resort to polling the resource if the Observe Option is not returned in the response to the GET request.
- A value MUST NOT be given for the "obs" attribute; any present value MUST be ignored by parsers. The "obs" attribute MUST NOT appear more than once in a given link-value; occurrences after the first MUST be ignored by parsers.

REQ: GET /.well-known/core?r=light-lux

RES: 2.05 Content

</sensors/light>;obs; rt="light-lux";if="sensor"



CoAP Resource Discovery - Query Filtering

- A server implementing the CoAP Resource Discovery can recognize the query part of a resource discovery URI as a filter on the resources to be returned. The path and query components together should conform to the following level-4 URI Template [RFC6570]:

`/ .well-known/core{ ?search* }`

- where the variable "search" is a 1-element list that has a single name/value pair, where
 - name is either "href" (a link-param name) or any other link-extension name
 - value is either a Complete Value String that does not end in an "*", or a Prefix Value String followed by an "*"
- The following are examples of valid query URIs:
 - `?href=/foo` → matches a link-value equals to /foo
 - `?href=/foo*` → matches a link-value that starts with /foo
 - `?foo=bar` → matches a link-value that has a target attribute named foo with the exact value bar
 - `?foo=bar*` → matches a link-value that has a target attribute named foo, the value of which starts with bar
 - `?foo=*` → matches a link-value that has a target attribute named foo

This example includes links to two different sensors sharing the same Interface Description. The default relation type for this link format is "hosts" in links with no rel= target attribute. Thus, the links in this example tell that the Origin server from which /.well-known/core was requested (the context) hosts the resources: i) /sensors/temp and ii) /sensors/light

```
REQ: GET /.well-known/core
```

```
RES: 2.05 Content
```

```
</sensors/temp>;if="sensor",  
</sensors/light>;if="sensor"
```

Without the linefeeds inserted here for readability, the format actually looks as follows.

```
</sensors/temp>;if="sensor",</sensors/light>;if="sensor"
```


CoAP Resource Discovery - Examples

This example arranges link descriptions hierarchically, with the entry point including a link to a **sub-resource** containing links about the sensors.

1

**First
Request**

REQ: GET /.well-known/core

RES: 2.05 Content
</sensors>;ct=40

application/link-format
(Identifier: 40)



2

**Second
Request**

REQ: GET /sensors

RES: 2.05 Content
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor"

An example query filter may look like:

```
REQ: GET /.well-known/core?rt=light-lux
```

```
RES: 2.05 Content
```

```
</sensors/light>;rt="light-lux";if="sensor"
```

Note that relation-type attributes like 'rt', 'if', and 'rel' can have multiple values separated by spaces.

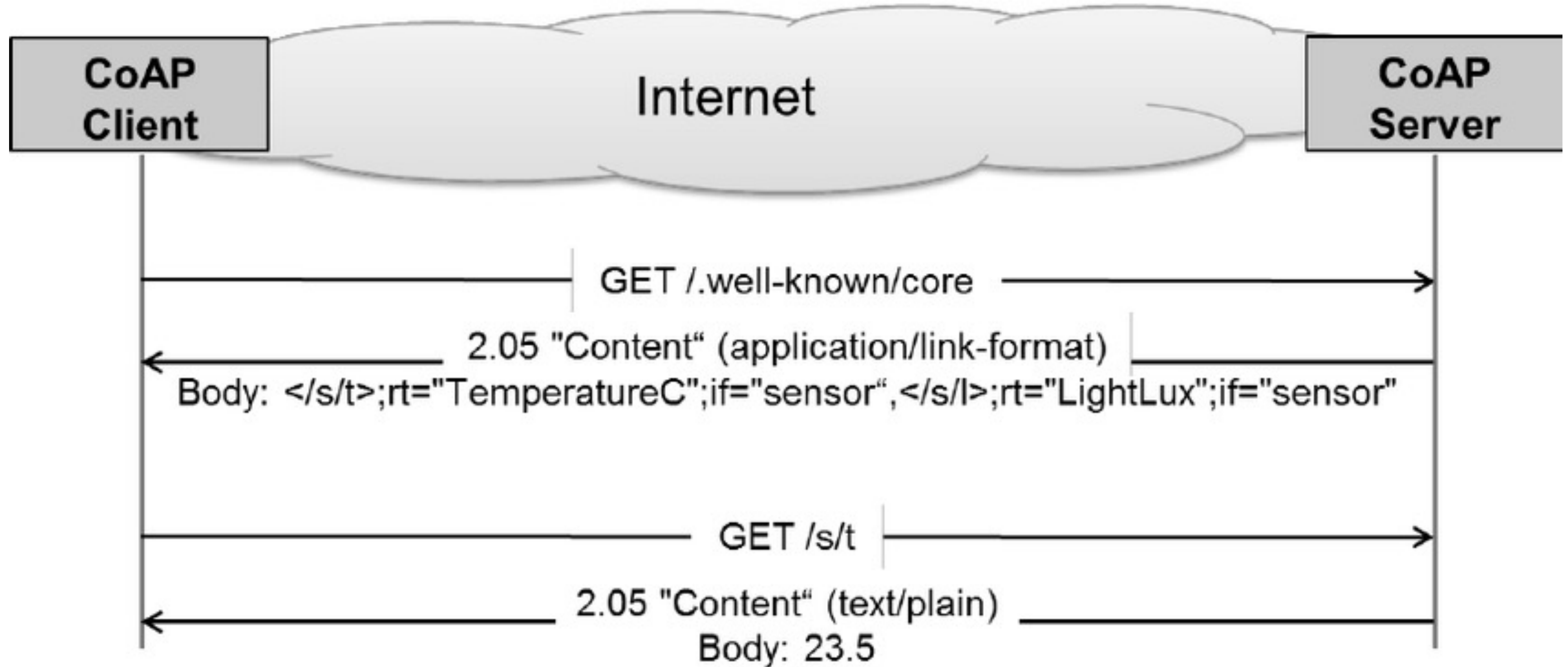
A query filter parameter can match any one of those values, as in this example:

```
REQ: GET /.well-known/core?rt=light-lux
```

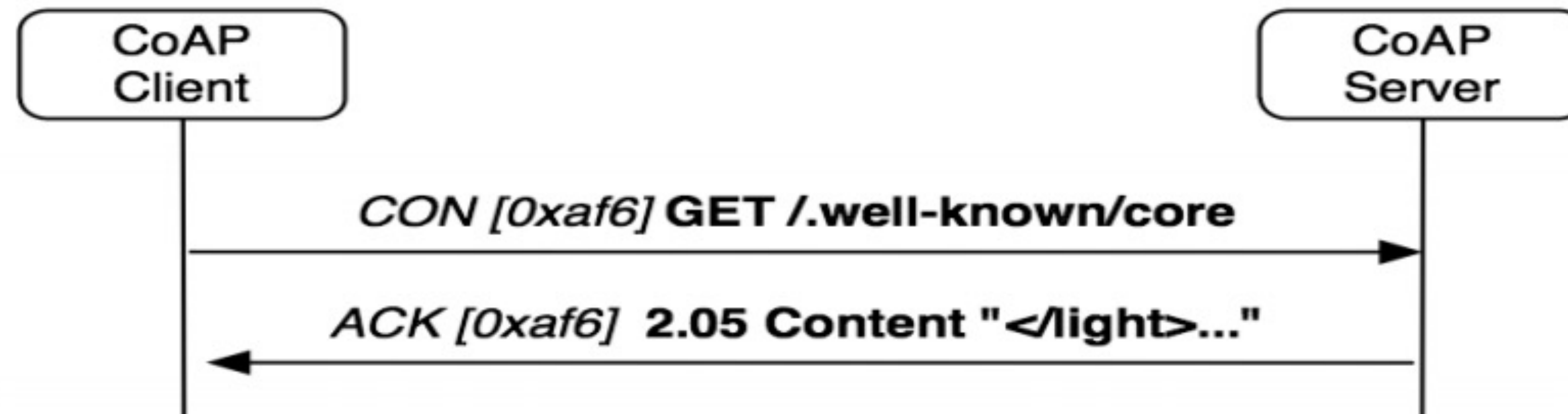
```
RES: 2.05 Content
```

```
</sensors/light>;rt="light-lux core.sen-light";if="sensor"
```

CoRE Link Format (RFC 6690) - Discovery



CoRE Link Format (RFC 6690) - Discovery



```
</dev/bat>;obs;rt="ipso:dev-bat";ct="0",  
</dev/mdl>;rt="ipso:dev-mdl";ct="0",  
</dev/mfg>;rt="ipso:dev-mfg";ct="0",  
</pwr/0/rel>;obs;rt="ipso:pwr-rel";ct="0",  
</pwr/0/w>;obs;rt="ipso:pwr-w";ct="0",  
</sen/temp>;obs;rt="ucum:Cel";ct="0"
```


- This document defines well-known REST interface descriptions for types in constrained web servers using the CoRE Link Format.
- Main types are:
 - Batch
 - Sensor
 - Parameter
 - Actuator
- A short reference is provided for each type that can be efficiently included in the interface description attribute of the CoRE Link Format.
- These descriptions are intended to be for general use in resource designs or for inclusion in more specific interface profiles.
- The document defines the concepts of Function Set and Binding. The former is the basis element to create RESTful profiles and the latter helps the configuration of links between resources located on one or more endpoints

Interface	if=	Methods	Content-Formats
Link List	core.ll	GET	link-format
Batch	core.b	GET, PUT, POST	senml
Linked Batch	core.lb	GET, PUT, POST, DELETE	link-format, senml
Sensor	core.s	GET	senml, text/plain
Parameter	core.p	GET, PUT	senml, text/plain
Read-only Parameter	core.rp	GET	senml, text/plain
Actuator	core.a	GET, PUT, POST	senml, text/plain

- The following is an example of links in the CoRE Link Format using these interface descriptions. The resource hierarchy is based on a simple profile defined in Appendix A. These links are used in the subsequent examples below.

```
Req: GET /.well-known/core
Res: 2.05 Content (application/link-format)
</s>;rt="simple.sen";if="core.b",
</s/lt>;rt="simple.sen.lt";if="core.s",
</s/tmp>;rt="simple.sen.tmp";if="core.s";obs,
</s/hum>;rt="simple.sen.hum";if="core.s",
</a>;rt="simple.act";if="core.b",
</a/1/led>;rt="simple.act.led";if="core.a",
</a/2/led>;rt="simple.act.led";if="core.a",
</d>;rt="simple.dev";if="core.ll",
</l>;if="core.lb",
```

- The Sensor interface allows the value of a sensor resource to be read (GET). The Media type of the resource can be either plain text or SenML.
- Plain text MAY be used for a single measurement that does not require meta-data.
- For a measurement with meta-data such as a unit or time stamp, SenML SHOULD be used. A resource with this interface MAY use SenML to return multiple measurements in the same representation, for example a list of recent measurements.

Examples of Sensor interface requests in both text/plain and application/senml+json.

Req: GET /s/humidity (Accept: text/plain)

Res: 2.05 Content (text/plain)

80

Req: GET /s/humidity (Accept: application/senml+json)

Res: 2.05 Content (application/senml+json)

```
{"e":  
  { "n": "humidity", "v": 80, "u": "%RH" },  
}
```

- The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read (GET) or set (PUT). Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading and setting parameter.

Req: GET /d/name

Res: 2.05 Content (text/plain)

node5

Req: PUT /d/name (text/plain)

outdoor

Res: 2.04 Changed

- The Read-only Parameter interface allows configuration parameters to be read (GET) but not set. Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading such a parameter.

```
Req: GET /d/model  
Res: 2.05 Content (text/plain)  
SuperNode200
```


- The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment).
- Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read (GET) or a new actuator value set (PUT).
- This interface defines the use of POST (with no body) to toggle an actuator between its possible values. Plain text or SenML Media types MAY be returned from this type of interface.
- A resource with this interface MAY use SenML to include multiple measurements in the same representation, for example a list of recent actuator values or a list of values to set.

The following example shows requests for reading, setting and toggling an actuator (turning on a led).

Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0

Req: PUT /a/1/led (text/plain)
1
Res: 2.04 Changed

Req: POST /a/1/led (text/plain)
Res: 2.04 Changed

Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0

- Connecting sensors to the internet is not new, and there have been many protocols designed to facilitate it.
SENML specification defines new media types for carrying simple sensor information in a protocol such as HTTP or CoAP called the Sensor Markup Language (SenML).
- This format was designed so that processors with very limited capabilities could easily encode a sensor measurement into the media type, while at the same time a server parsing the data could relatively efficiently collect a large number of sensor measurements.
- SenML is defined by a data model for measurements and simple metadata about measurements and devices. The data is structured as a single object (with attributes) that contains an array of entries. Each entry is an object that has attributes such as a unique identifier for the sensor, the time the measurement was made, and the current value. Serializations for this data model are defined for JSON, XML and Efficient XML Interchange (EXI).

<https://tools.ietf.org/html/draft-jennings-senml-10>

“This specification defines a format for representing simple sensor measurements and device parameters in Sensor Measurement Lists (SenML). Representations are defined in JavaScript Object Notation (JSON), Concise Binary Object Representation (CBOR), Extensible Markup Language (XML), and Efficient XML Interchange (EXI), which share the common SenML data model. A simple sensor, such as a temperature sensor, could use one of these media types in protocols such as HTTP or the Constrained Application Protocol (CoAP) to transport the measurements of the sensor or to be configured.”

- The data is structured as a single array that contains a series of SenML Records that can each contain fields such as a unique identifier for the sensor, the time the measurement was made, the unit the measurement is in, and the current value of the sensor. The following shows a measurement from a temperature gauge encoded in the JSON syntax.

```
[  
  { "n": "urn:dev:ow:10e2073a01080063", "u": "Cel", "v": 23.1 }  
]
```

- In the example above, the array has a single SenML Record with a measurement for a sensor named "urn:dev:ow:10e2073a01080063".

- The basic design is an array with a series of measurements. The following example shows two measurements made at different times. The value of a measurement is given by the "v" field, the time of a measurement is in the "t" field, the "n" field has a unique sensor name, and the unit of the measurement is carried in the "u" field.

```
[  
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020076e+09,"v":23.5},  
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020091e+09,"v":23.6}  
]
```

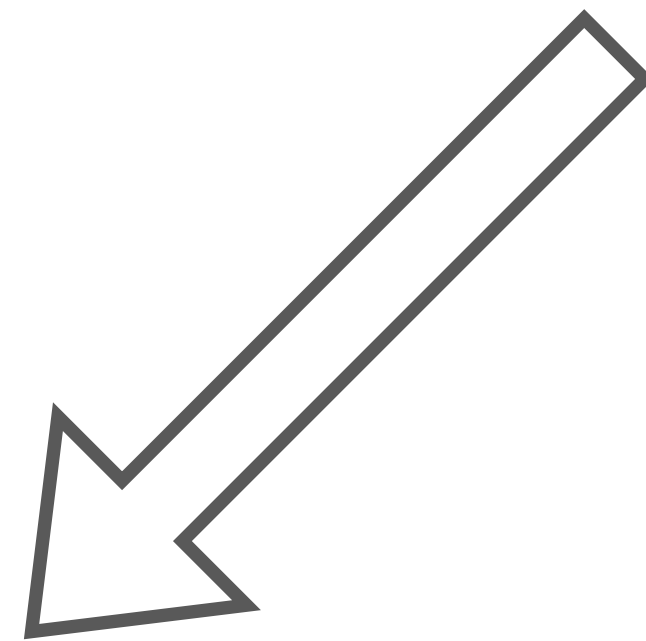
- To keep the messages small, it does not make sense to repeat the "n" field in each SenML Record, so there is a concept of a Base Name, which is simply a string that is prepended to the Name field of all elements in that Record and any Records that follow it. So, a more compact form of the example above is the following.

```
[  
  {"bn":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020076e+09,"v":23.5},  
  {"u":"Cel","t":1.276020091e+09,"v":23.6}  
]
```

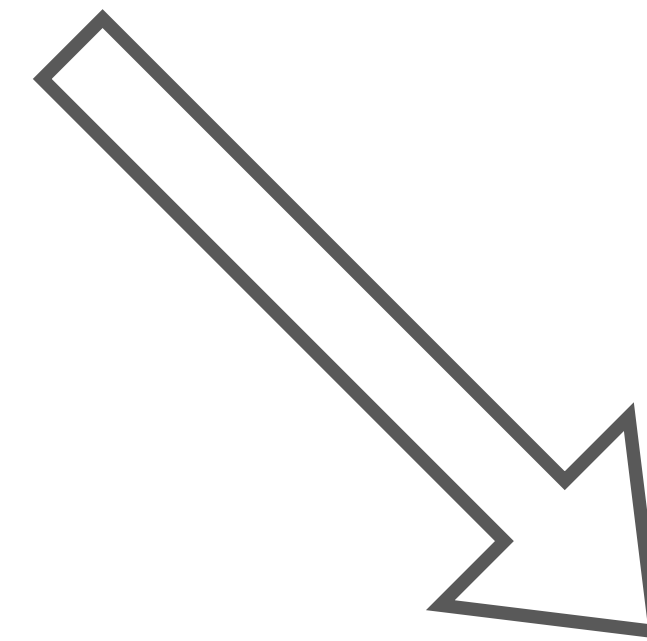
- In the above example, the Base Name is in the "bn" field, and the "n" fields in each Record are empty strings, so they are omitted.

- **SenML Record:** One measurement or configuration instance in time presented using the SenML data model
- **SenML Pack:** One or more SenML Records in an array structure
- **SenML Label:** A short name used in SenML Records to denote different SenML fields (e.g., "v" for "value")
- **SenML Field:** A component of a record that associates a value to a SenML Label for this record
- **SenSML:** Sensor Streaming Measurement List
- **SenSML Stream:** One or more SenML Records to be processed as a stream

- Each SenML Pack carries a single array that represents a set of measurements and/or parameters. This array contains a series of SenML Records with several fields described below. There are two kinds of fields: base and regular. Both the base and regular fields can be included in any SenML Record. The base fields apply to the entries in the Record and also to all Records after it up to, but not including, the next Record that has that same base field. All base fields are optional. Regular fields can be included in any SenML Record and apply only to that Record.



Base Fields



Regular Fields

- **Base Name:** This is a string that is prepended to the names found in the entries.
- **Base Time:** A base time that is added to the time found in an entry.
- **Base Unit:** A base unit that is assumed for all entries, unless otherwise indicated. If a record does not contain a Unit value, then the Base Unit is used. Otherwise, the value found in the Unit (if any) is used.
- **Base Value:** A base value is added to the value found in an entry, similar to Base Time.
- **Base Sum:** A base sum is added to the sum found in an entry, similar to Base Time.
- **Base Version:** Version number of the media type format. This field is an optional positive integer and defaults to 10 if not present.

Name	Label	CBOR Label	JSON Type	XML Type
Base Name	bn	-2	String	string
Base Time	bt	-3	Number	double
Base Unit	bu	-4	String	string
Base Value	bv	-5	Number	double
Base Sum	bs	-6	Number	double
Base Version	bver	-1	Number	int
Name	n	0	String	string
Unit	u	1	String	string
Value	v	2	Number	double
String Value	vs	3	String	string
Boolean Value	vb	4	Boolean	boolean
Data Value	vd	8	String (*)	string (*)
Sum	s	5	Number	double
Time	t	6	Number	double
Update Time	ut	7	Number	double

SenML Labels Registry



Name	Label	CL	JSON Type	XML Type	EI	Reference
Base Name	bn	-2	String	string	a	RFC 8428
Base Time	bt	-3	Number	double	a	RFC 8428
Base Unit	bu	-4	String	string	a	RFC 8428
Base Value	bv	-5	Number	double	a	RFC 8428
Base Sum	bs	-6	Number	double	a	RFC 8428
Base Version	bver	-1	Number	int	a	RFC 8428
Name	n	0	String	string	a	RFC 8428
Unit	u	1	String	string	a	RFC 8428
Value	v	2	Number	double	a	RFC 8428
String Value	vs	3	String	string	a	RFC 8428
Boolean Value	vb	4	Boolean	boolean	a	RFC 8428
Data Value	vd	8	String	string	a	RFC 8428
Sum	s	5	Number	double	a	RFC 8428
Time	t	6	Number	double	a	RFC 8428
Update Time	ut	7	Number	double	a	RFC 8428

- IANA has created a new registry for SenML Labels called the “SenML Labels” registry. The initial contents of the registry are as illustrated in the table of the previous slide
- All new entries must define the Name, Label, and XML Type, but the CBOR labels SHOULD be left empty as CBOR will use the string encoding for any new labels. The EI column contains the EXI schemald value of the first schema that includes this label, or it is empty if this label was not intended for use with EXI. The Reference column SHOULD contain information about where to find out more information about this label.
- The JSON, CBOR, and EXI types are derived from the XML type. All XML numeric types such as double, float, integer, and int become a JSON Number. XML boolean and string become a JSON Boolean and String,

SenML Units Registry

Symbol	Description	Type	Reference
m	meter	float	RFC 8428
kg	kilogram	float	RFC 8428
g	gram*	float	RFC 8428
s	second	float	RFC 8428
A	ampere	float	RFC 8428
K	kelvin	float	RFC 8428
cd	candela	float	RFC 8428
mol	mole	float	RFC 8428
Hz	hertz	float	RFC 8428
rad	radian	float	RFC 8428
sr	steradian	float	RFC 8428
N	newton	float	RFC 8428
Pa	pascal	float	RFC 8428
J	joule	float	RFC 8428
W	watt	float	RFC 8428
C	coulomb	float	RFC 8428
V	volt	float	RFC 8428
F	farad	float	RFC 8428
Ohm	ohm	float	RFC 8428
S	siemens	float	RFC 8428
Wb	weber	float	RFC 8428
T	tesla	float	RFC 8428
H	henry	float	RFC 8428
Cel	degrees Celsius	float	RFC 8428
lm	lumen	float	RFC 8428
lx	lux	float	RFC 8428
Bq	becquerel	float	RFC 8428
Gy	gray	float	RFC 8428
Sv	sievert	float	RFC 8428
kat	katal	float	RFC 8428
m2	square meter (area)	float	RFC 8428
m3	cubic meter (volume)	float	RFC 8428
l	liter (volume)*	float	RFC 8428
m/s	meter per second (velocity)	float	RFC 8428
m/s2	meter per square second (acceleration)	float	RFC 8428
m3/s	cubic meter per second (flow rate)	float	RFC 8428
l/s	liter per second (flow rate)*	float	RFC 8428
W/m2	watt per square meter (irradiance)	float	RFC 8428
cd/m2	candela per square meter (luminance)	float	RFC 8428
bit	bit (information content)	float	RFC 8428
bit/s	bit per second (data rate)	float	RFC 8428
lat	degrees latitude (Note 1)	float	RFC 8428
lon	degrees longitude (Note 1)	float	RFC 8428

pH	pH value (acidity; logarithmic quantity)	float	RFC 8428
dB	decibel (logarithmic quantity)	float	RFC 8428
dBW	decibel relative to 1 W (power level)	float	RFC 8428
Bspl	bel (sound pressure level; logarithmic quantity)*	float	RFC 8428
count	1 (counter value)	float	RFC 8428
/	1 (ratio, e.g., value of a switch; Note 2)	float	RFC 8428
%	1 (ratio, e.g., value of a switch; Note 2)*	float	RFC 8428
%RH	percentage (relative humidity)	float	RFC 8428
%EL	percentage (remaining battery energy level)	float	RFC 8428
EL	seconds (remaining battery energy level)	float	RFC 8428
1/s	1 per second (event rate)	float	RFC 8428
1/min	1 per minute (event rate, "rpm")*	float	RFC 8428
beat/min	1 per minute (heart rate in beats per minute)*	float	RFC 8428
beats	1 (cumulative number of heart beats)*	float	RFC 8428
S/m	siemens per meter (conductivity)	float	RFC 8428

- IANA has created a registry of SenML unit symbols called the “SenML Units” registry. The primary purpose of this registry is to make sure that symbols uniquely map to indicate a type of measurement.
- Definitions for many of these units can be found in other publications (See RFC for details).
- Units marked with an asterisk are NOT RECOMMENDED to be produced by new implementations but are in active use and SHOULD be implemented by consumers that can use the corresponding SenML units that are closer to the unscaled SI units.

Single Data Point: The following shows a temperature reading taken approximately “now” by a 1-wire sensor device that was assigned the unique 1-wire address of 10e2073a01080063:

```
[  
  { "n": "urn:dev:ow:10e2073a01080063", "u": "Cel", "v": 23.1 }  
]
```

Multiple Data Points: The following example shows voltage and current "now", i.e., at an unspecified time.

```
[  
  {"bn":"urn:dev:ow:10e2073a01080063:", "n":"voltage", "u":"V", "v":120.1},  
  {"n":"current", "u":"A", "v":1.2}  
]
```

The next example is similar to the above one, but it shows current at. Tue Jun 8 18:01:16.001 UTC 2010 and at each second for the previous 5 seconds.

```
[  
  {"bn":"urn:dev:ow:10e2073a0108006:", "bt":1.276020076001e+09,  
    "bu":"A", "bver":5,  
    "n":"voltage", "u":"V", "v":120.1},  
  {"n":"current", "t":-5, "v":1.2},  
  {"n":"current", "t":-4, "v":1.3},  
  {"n":"current", "t":-3, "v":1.4},  
  {"n":"current", "t":-2, "v":1.5},  
  {"n":"current", "t":-1, "v":1.6},  
  {"n":"current", "v":1.7}  
]
```

Multiple Measurements: The following example shows humidity measurements from a mobile device with a 1-wire address 10e2073a01080063, starting at Mon Oct 31 13:24:24 UTC 2011. The device also provides position data, which is provided in the same measurement or parameter array as separate entries. Note that time is used to correlate data that belongs together, e.g., a measurement and a parameter associated with it. Finally, the device also reports extra data about its battery status at a separate time.

```
[
  { "bn": "urn:dev:ow:10e2073a01080063", "bt": 1.320067464e+09,
    "bu": "%RH", "v": 20 },
  { "u": "lon", "v": 24.30621 },
  { "u": "lat", "v": 60.07965 },
  { "t": 60, "v": 20.3 },
  { "u": "lon", "t": 60, "v": 24.30622 },
  { "u": "lat", "t": 60, "v": 60.07965 },
  { "t": 120, "v": 20.7 },
  { "u": "lon", "t": 120, "v": 24.30623 },
  { "u": "lat", "t": 120, "v": 60.07966 },
  { "u": "%EL", "t": 150, "v": 98 },
  { "t": 180, "v": 21.2 },
  { "u": "lon", "t": 180, "v": 24.30628 },
  { "u": "lat", "t": 180, "v": 60.07967 }
]
```

Encoding	Size	Compressed Size
JSON	573	206
XML	649	235
CBOR	254	196
EXI	161	184

Multiple Data Types: The following example shows a sensor that returns different data types.

```
[
  { "bn": "urn:dev:ow:10e2073a01080063:", "n": "temp", "u": "Cel", "v": 23.1 },
  { "n": "label", "vs": "Machine Room" },
  { "n": "open", "vb": false },
  { "n": "nfc-reader", "vd": "aGkgCg" }
]
```

Collection of Resources: The following example shows the results from a query to one device that aggregates multiple measurements from other devices. The example assumes that a client has fetched information from a device at 2001:db8::2 by performing a GET operation on `http://[2001:db8::2]` at Mon Oct 31 16:27:09 UTC 2011 and has gotten two separate values as a result: a temperature and humidity measurement as well as the results from another device at `http://[2001:db8::1]` that also had a temperature and humidity measurement. Note that the last record would use the Base Name from the 3rd record but the Base Time from the first record.

```
[
  { "bn": "2001:db8::2/", "bt": 1.320078429e+09,
    "n": "temperature", "u": "Cel", "v": 25.2 },
  { "n": "humidity", "u": "%RH", "v": 30 },
  { "bn": "2001:db8::1/", "n": "temperature", "u": "Cel", "v": 12.3 },
  { "n": "humidity", "u": "%RH", "v": 67 }
]
```


The following example shows the SenML that could be used to set the current set point of a typical residential thermostat that has a temperature set point, a switch to turn on and off the heat, and a switch to turn on the fan

```
[
    override.
    { "bn": "urn:dev:ow:10e2073a01080063:" },
    { "n": "temp", "u": "Cel", "v": 23.1 },
    { "n": "heat", "u": "/", "v": 1 },
    { "n": "fan", "u": "/", "v": 0 }
]
```

In the following example, two different lights are turned on. It is assumed that the lights are on a network that can guarantee delivery of the messages to the two lights within 15 ms (e.g., a network using 802.1BA [IEEE802.1BA] and 802.1AS [IEEE802.1AS] for time synchronization). The controller has set the time of the lights to come on at 20 ms in the future from the current time. This allows both lights to receive the message, wait till that time, then apply the switch command so that both lights come on at the same time.

```
[
    { "bt": 1.320078429e+09, "bu": "/", "n": "2001:db8::3", "v": 1 },
    { "n": "2001:db8::4", "v": 1 }
]
```

The following shows two lights being turned off using a non-deterministic network that has high odds of delivering a message in less than 100 ms and uses NTP for time synchronization. The current time is 1320078429. The user has just turned off a light switch that is turning off two lights. Both lights are immediately dimmed to 50% brightness to give the user instant feedback that something is changing.

However, given the network, the lights will probably dim at somewhat different times. Then 100 ms in the future, both lights will go off at the same time. The instant, but not synchronized, dimming gives the user the sensation of quick responses, and the timed-off 100 ms in the future gives the perception of both lights going

off at the same time.

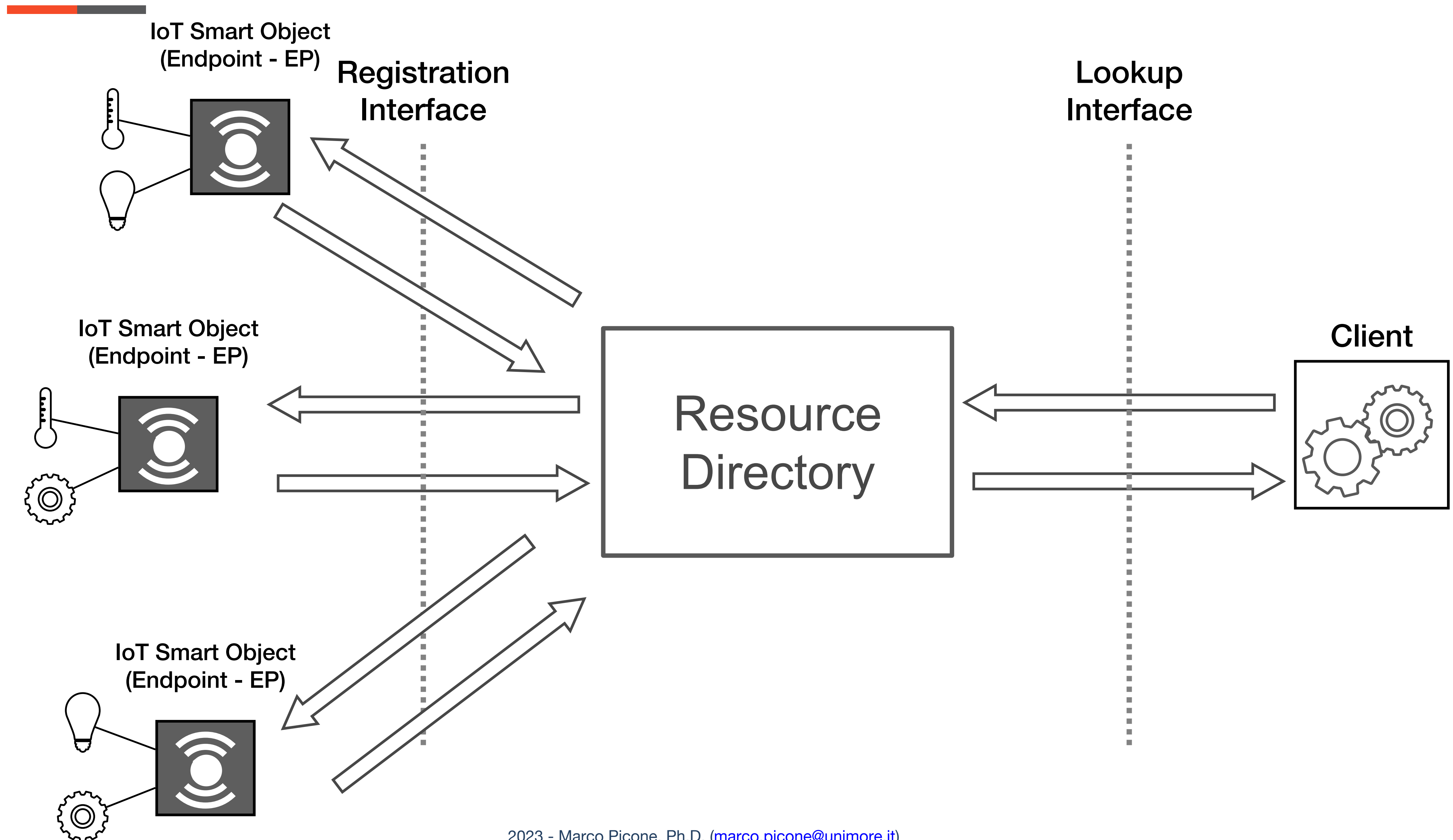
```
[  
  {"bt":1.320078429e+09,"bu":"/","n":"2001:db8::3","v":0.5},  
  {"n":"2001:db8::4","v":0.5},  
  {"n":"2001:db8::3","t":0.1,"v":0},  
  {"n":"2001:db8::4","t":0.1,"v":0}  
]
```

- The SenML format can be extended with further custom fields
- Both new **base** and **regular** fields are allowed (following Labels Registry options).
- Implementations MUST ignore fields they don't recognize unless that field has a label name that ends with the "_" character, in which case an error MUST be generated.
- All SenML Records in a Pack MUST have the same version number. This is typically done by adding a Base Version field to only the first Record in the Pack or by using the default value
- Systems reading one of the objects MUST check for the Base Version field. If this value is a version number larger than the version that the system understands, the system MUST NOT use this object. This allows the version number to indicate that the object contains structure or semantics that is different from what is defined in the present document beyond just making use of the extension points provided here.
- New version numbers can only be defined in an RFC that updates this specification or its successors.

- Sometimes direct discovery of resources through /.well-known/core is not practical
 - sleeping nodes
 - networks where multicast traffic is inefficient
- A resource directory (RD) is a CoAP endpoint that can be used as a registry to register, maintain, lookup and remove resources descriptions (draft-ietf-core-resource-directory-07)
- The RD contains CoRE Links to resources
 - it supports a function set to be populated
 - it supports a function set to be looked up (queries can be made to filter on the basis of certain attributes)

<https://tools.ietf.org/html/draft-ietf-core-resource-directory-19>

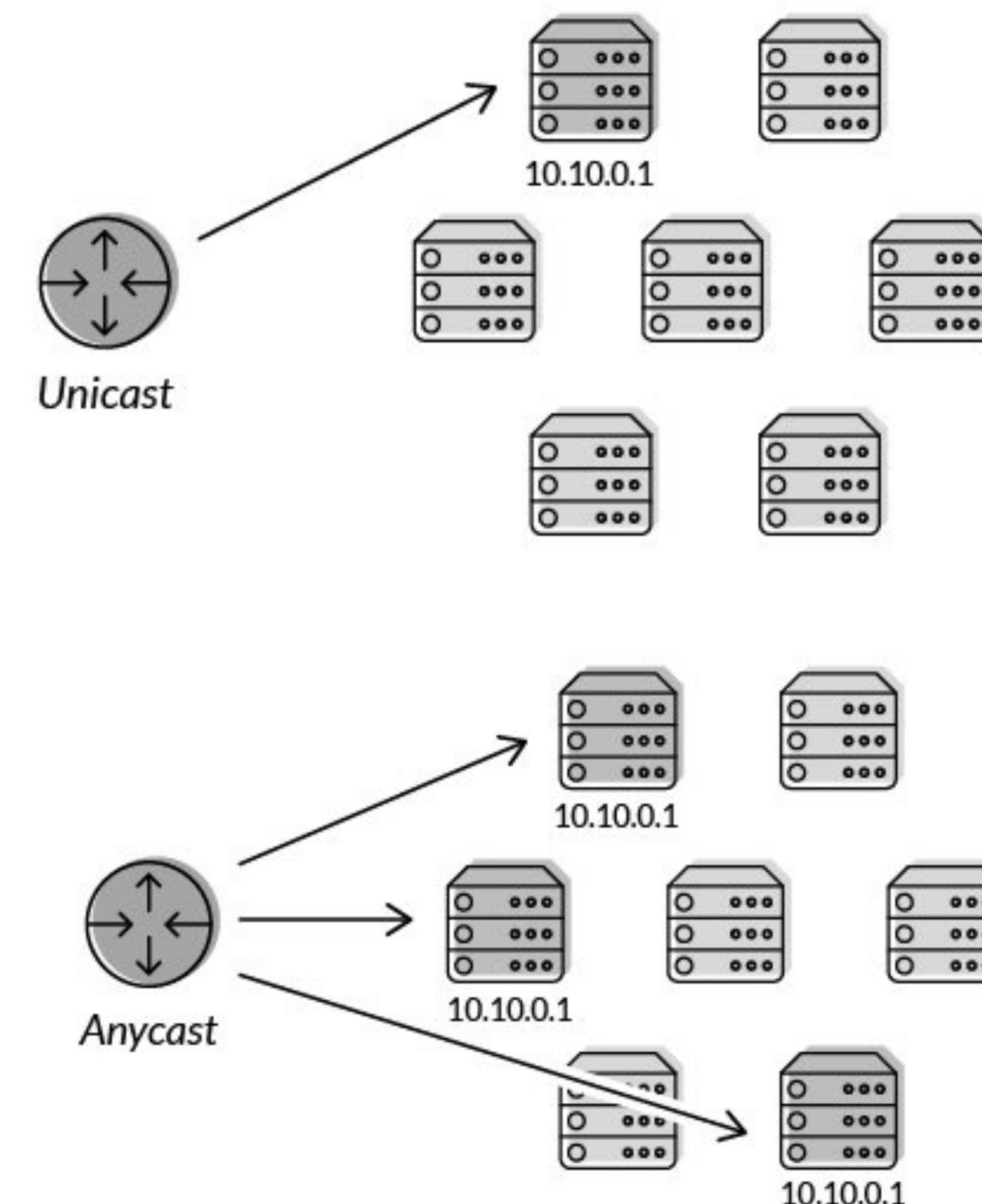
CoRE Resource Directory - Architecture



- The Resource Directory (RD) is used as a repository of registrations describing resources hosted on other web servers, also called endpoints (EP).
- An endpoint is a web server associated with a scheme, IP address and port (e.g., coap://192.168.1.127:5683)
- A physical node may host one or more endpoints
- The RD implements a set of REST interfaces for endpoints to register and maintain resource directory registrations, and for endpoints to lookup resources from the RD
- CoRE Link Format [RFC6690] is adopted to a RD and its resources
- Registrations in the RD are soft state and need to be periodically refreshed
- An endpoint (IoT Smart Object) uses specific interfaces to register, update and remove a registration.
- It is also possible for an RD to fetch Web Links from endpoints and add their contents to resource directory registrations.
- At the first registration of an endpoint, a "registration resource" is created, the location of which is returned to the registering endpoint. The registering endpoint uses this registration resource to manage the contents of registrations.
- A lookup interface for discovering any of the Web Links stored in the RD is provided using the CoRE Link Format.

- The main three options to Support Resource Discovery are the following:
 - It may be configured with a specific IP address for the RD. That IP address may also be an anycast address, allowing the network to forward RD requests to an RD that is topologically close.
 - It may be configured with a DNS name for the RD and use DNS to return the IP address of the RD; it can find a DNS server to perform the lookup using the usual mechanisms for finding DNS servers.
 - It may be configured to use a service discovery mechanism such as DNS-SD [RFC6763]. The present specification suggests configuring the service with name **rd._sub._coap._udp**, preferably within the domain of the querying nodes. (See next slide for additional information and details about Service Discovery)

*“Anycast is a network addressing and routing methodology in which a **single destination address has multiple routing paths to two or more endpoint destinations**. Routers will select the desired path on the basis of number of hops, distance, lowest cost, latency measurements or based on the least congested route. Anycast networks are widely used for content delivery network (CDN) products to bring their content closer to the end user.” Source: Wikipedia*



Resource Directory - Interaction - URI Discovery

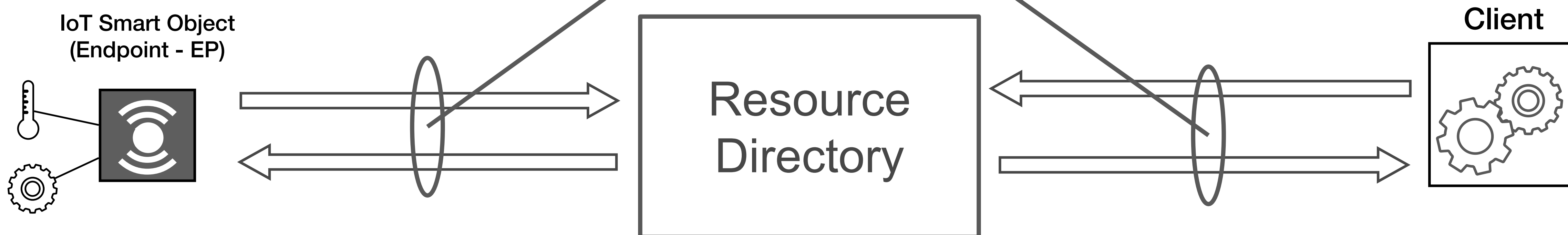
Before an endpoint can make use of an RD, it must first know the RD's address and port, and the URI path information for its REST APIs. URIs are defined using the well-known interface of the CoRE Link Format [RFC6690].

Discovery of the RD registration URI path is performed by sending either a multicast or unicast GET request to `"/.well-known/core"` and including a Resource Type (rt) parameter [RFC6690] with the value `"core.rd"` in the query string.

```
Req: GET coap://rd.example.com:5683/.well-known/core?rt=core.rd*
```

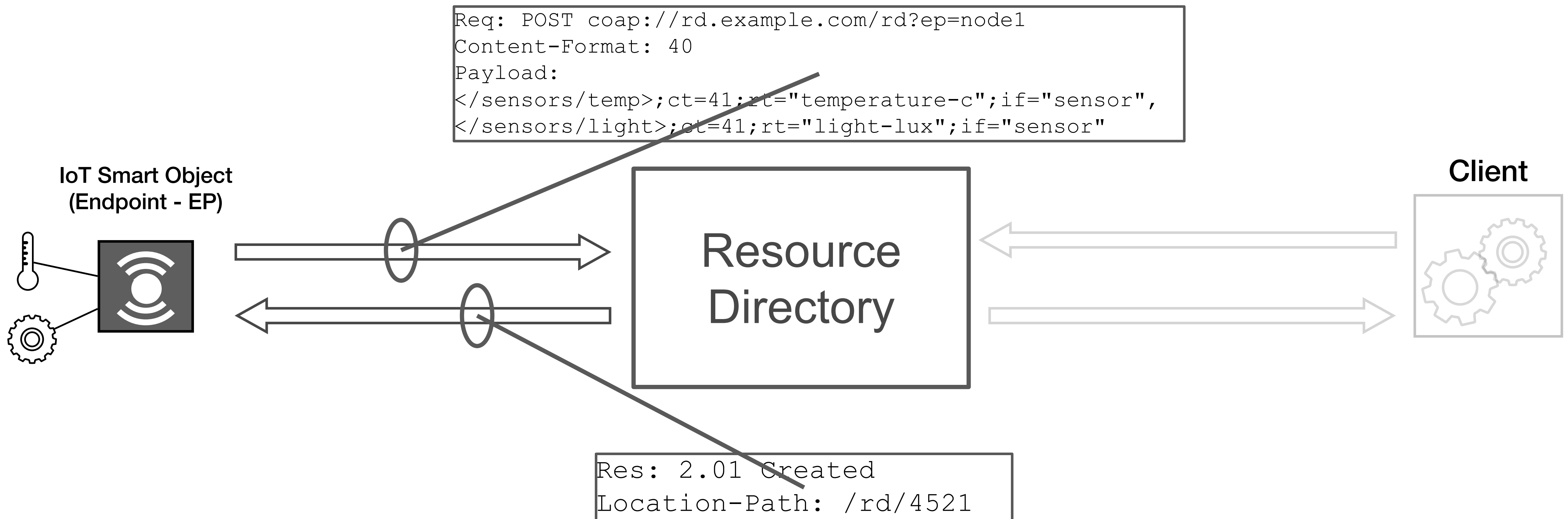
```
Res: 2.05 Content
```

```
</rd>;rt="core.rd";ct=40,  
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct=40,  
</rd-lookup/res>;rt="core.rd-lookup-res";ct=40,
```



Resource Directory - Interaction - Registration

After discovering the location of an RD, a registrant-ep or CT MAY register the resources of the registrant-ep using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format [RFC6690] or other representations of web links, along with query parameters indicating the name of the endpoint, and optionally the sector, lifetime and base URI of the registration.



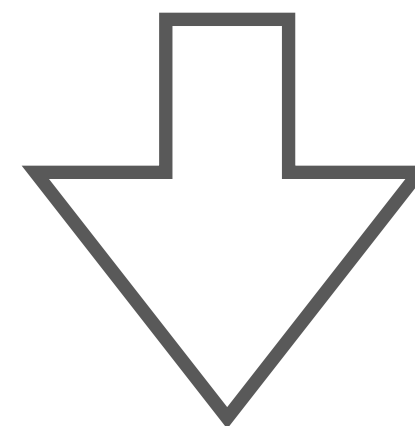
Simple Registration

Operations on the Registration Resource

Third-party registration

Registration Update

Registration Removal

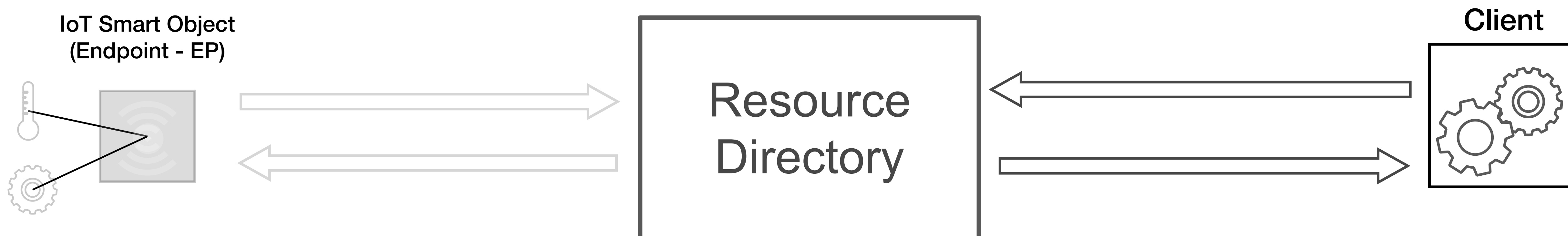


<https://tools.ietf.org/html/draft-ietf-core-resource-directory-19#section-5>

Resource Directory - Interactions - Lookup

To discover the resources registered with the RD, a lookup interface must be provided. RD Lookup allows **lookups for endpoints and resources** using attributes defined both in the **RD RFC** and with the **CoRE Link Format**. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Thus, an **endpoint lookup MUST return a list of endpoints** and a **resource lookup MUST return a list of links to resources**.

Lookup Type	Resource Type	Mandatory
Resource Endpoint	core.rd-lookup-res core.rd-lookup-ep	Mandatory Mandatory



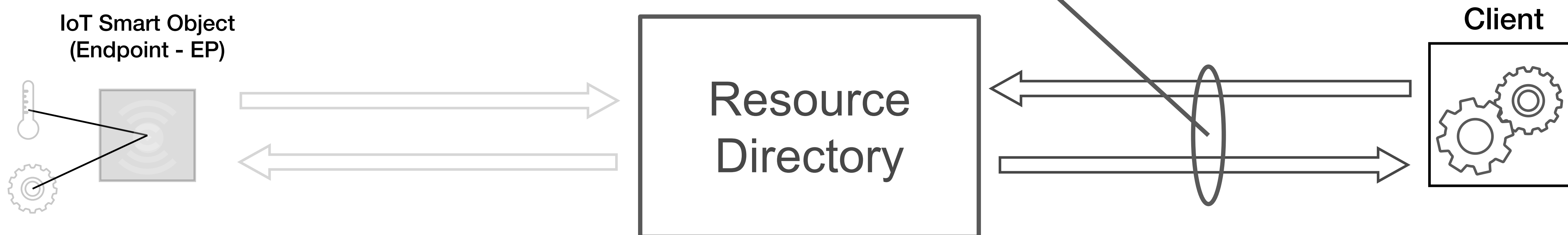
Resource Directory - Resource Lookup

Resource Lookup -> results in links that are semantically equivalent to the links submitted to the RD except that the target are fully resolved. Lookup filtering can be applied to target a specific group of resources.

```
Req: GET /rd-lookup/res?rt=temperature
```

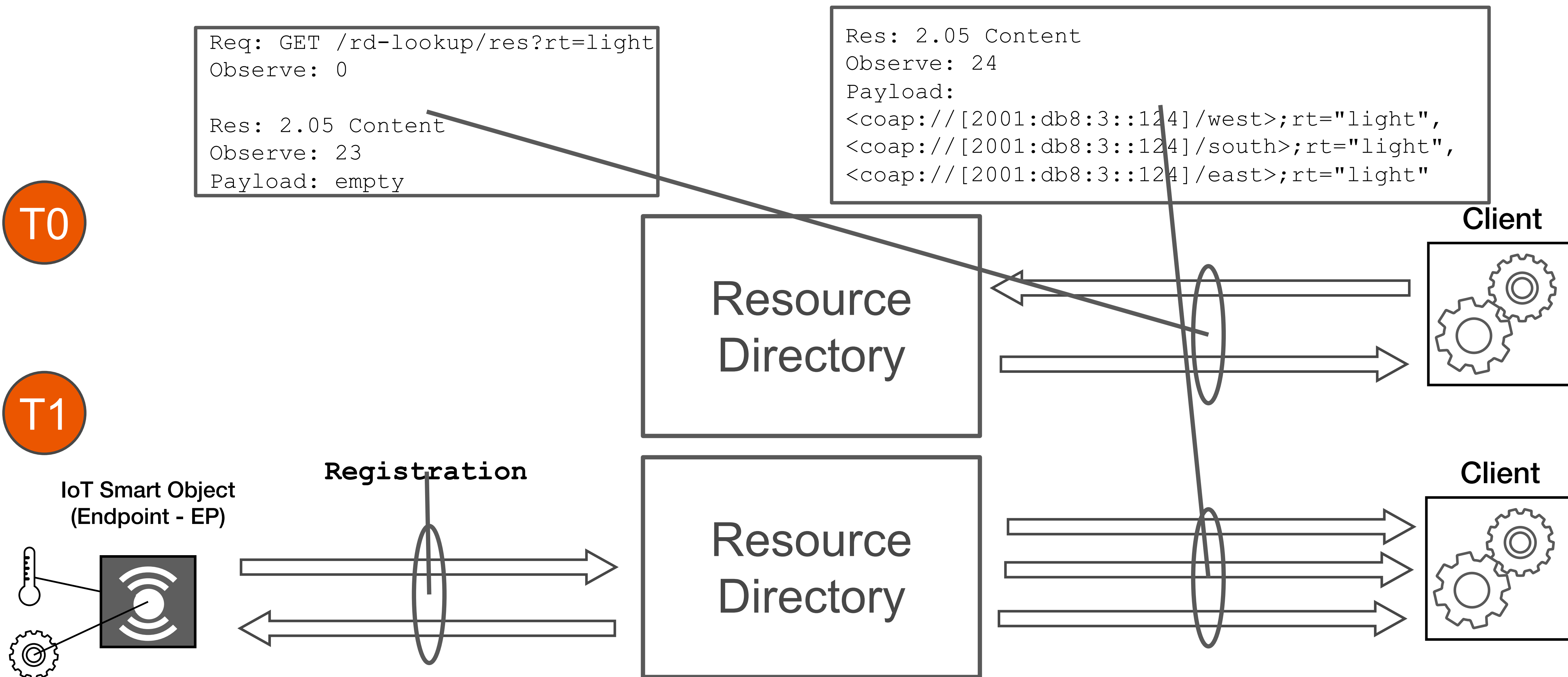
```
Res: 2.05 Content
```

```
<coap://[2001:db8:3::123]:61616/temp>;rt="temperature"
```



Resource Directory - Resource Lookup (Obs)

A client that wants to be notified of new resources as they show up can use observation



Resource Directory - Resource Lookup - Paging

A client can perform a paginated resource lookup

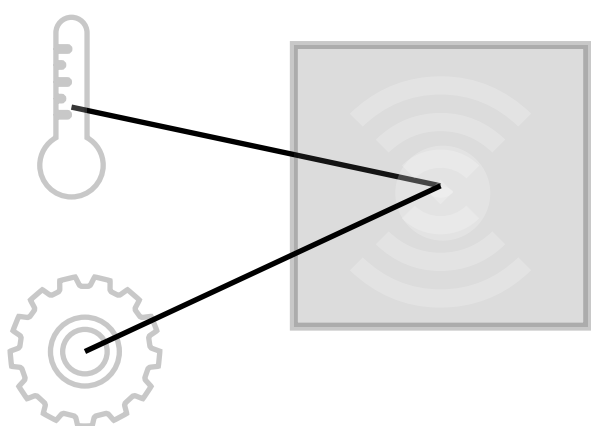
Req: GET /rd-lookup/res?page=0&count=5

Page Id & Page Length

Res: 2.05 Content

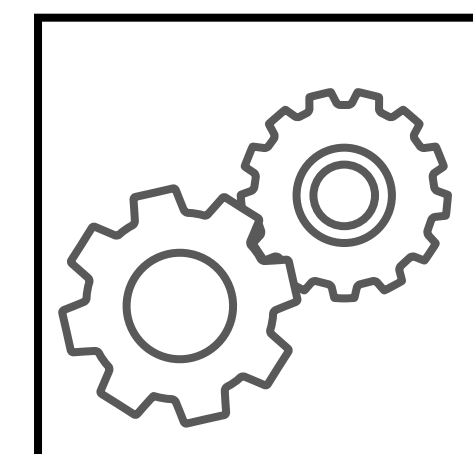
<coap://[2001:db8:3::123]:61616/res/0>;rt=sensor;ct=60,
<coap://[2001:db8:3::123]:61616/res/1>;rt=sensor;ct=60,
<coap://[2001:db8:3::123]:61616/res/2>;rt=sensor;ct=60,
<coap://[2001:db8:3::123]:61616/res/3>;rt=sensor;ct=60,
<coap://[2001:db8:3::123]:61616/res/4>;rt=sensor;ct=60

IoT Smart Object
(Endpoint - EP)



Resource
Directory

Client



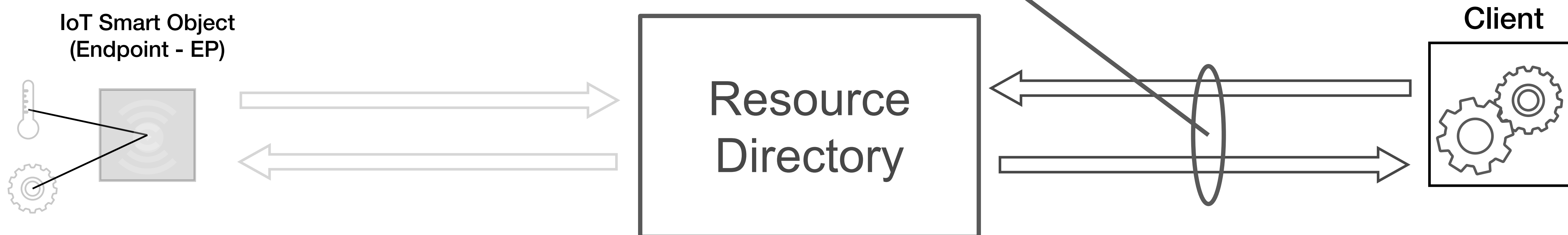
Resource Directory - Endpoint Lookup

The endpoint lookup returns registration resources which can only be manipulated by the registering endpoint. Endpoint registration resources are annotated with their endpoint names (ep), sectors (d, if present) and registration base URI (base; reports the registrant-ep's address) as well as a constant resource type (rt="core.rd-ep"); endpoint type (et) while the lifetime (lt) is not reported.

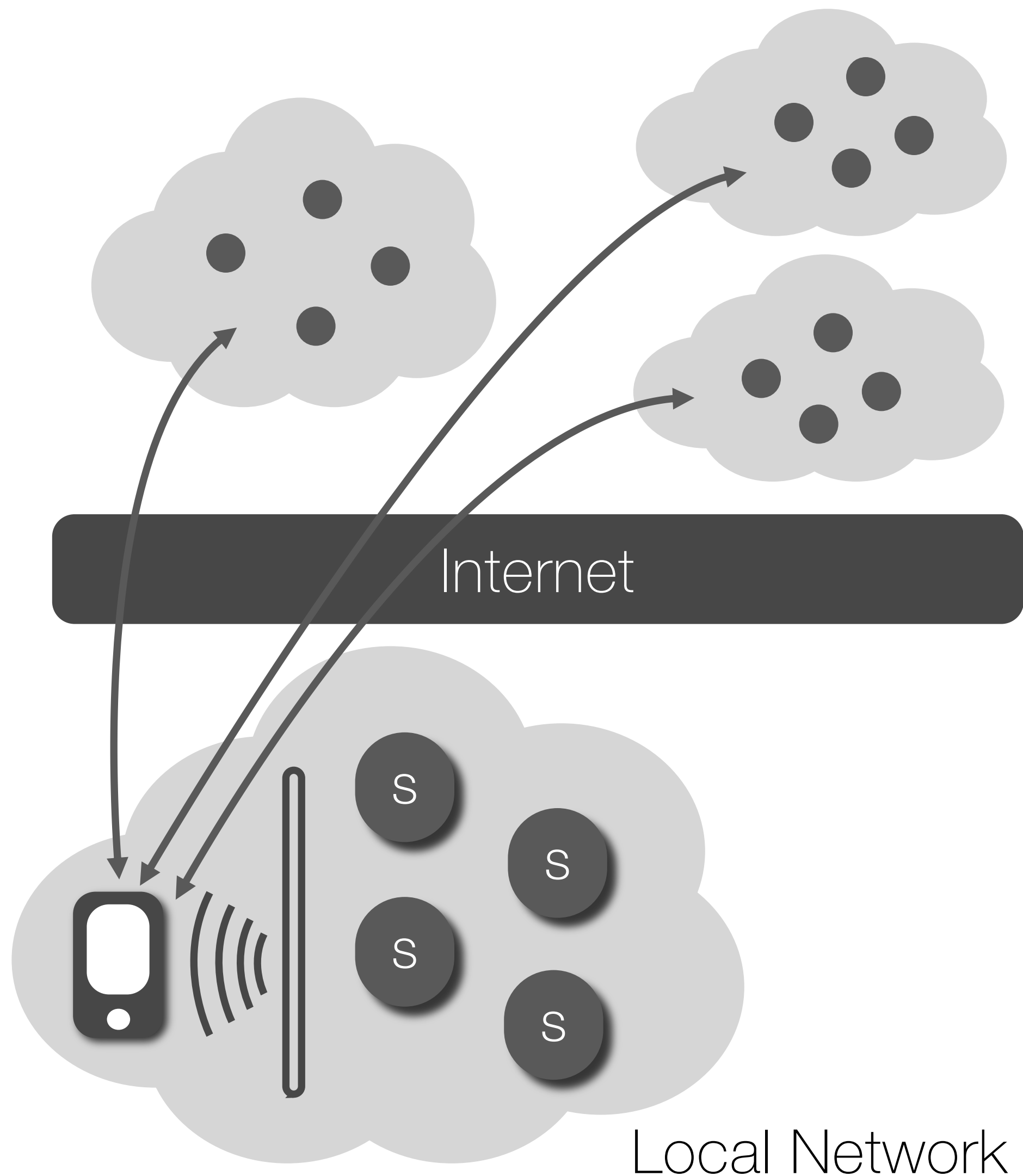
```
Req: GET /rd-lookup/ep?et=oic.d.sensor
```

```
Res: 2.05 Content
```

```
</rd/1234>;base="coap://[2001:db8:3::127]:61616";ep="node5";  
et="oic.d.sensor";ct="40";rt="core.rd-ep",  
</rd/4521>;base="coap://[2001:db8:3::129]:61616";ep="node7";  
et="oic.d.sensor";ct="40";d="floor-3";rt="core.rd-ep"
```



Service Discovery in IoT Networks



- Service Discovery is a fundamental component in dynamic environments to allow consumer devices and applications to find and interact with available services.
- SD could be performed:
 - In the local network [e.g. automatically accessing a building and connecting to the available WiFi Network]
 - Through different networks [e.g. Inside a target geographic region, “which services are available around me now ?”]
- Different Technologies could be used
 - Central infrastructure or repository
 - Distributed or peer-to-peer architecture
 - Multicast-based protocols

- As a part of discovering the services offered by a CoAP server, a client has to learn about the endpoint used by a server.
- A server is discovered by a client (knowing or) learning a URI that references a resource in the namespace of the server.
- The CoAP default port number 5683 MUST be supported by a server that offers resources for resource discovery and SHOULD be supported for providing access to other resources.
- The default port number 5684 for DTLS-secured CoAP MAY be supported by a server for resource discovery and for providing access to other resources. In addition, other endpoints may be hosted at other ports, e.g., in the dynamic port space.

- How can a CoAP client find out which services are available?
- Service discovery aims at getting the host port of the CoAP servers in the network
- Clients can discover services in many ways, such as by sending a multicast message to "All CoAP Nodes" group
- Several service discovery mechanisms have been defined and extensively treated in literature: **UPnP**, Service Location Protocol, **ZeroConf**, ...

- ZeroConf is a protocol that allows to automatically interconnect hosts with no prior configuration
- ZeroConf implements three main functionalities:
 - automatic network address assignment
 - automatic distribution and resolution of host names (mDNS)
 - automatic location of network services (DNS-SD)
- mDNS (Multicast DNS) uses the semantics of unicast DNS to provide host name resolution
- DNS-SD allows to discover a list of services in a domain through standard DNS queries

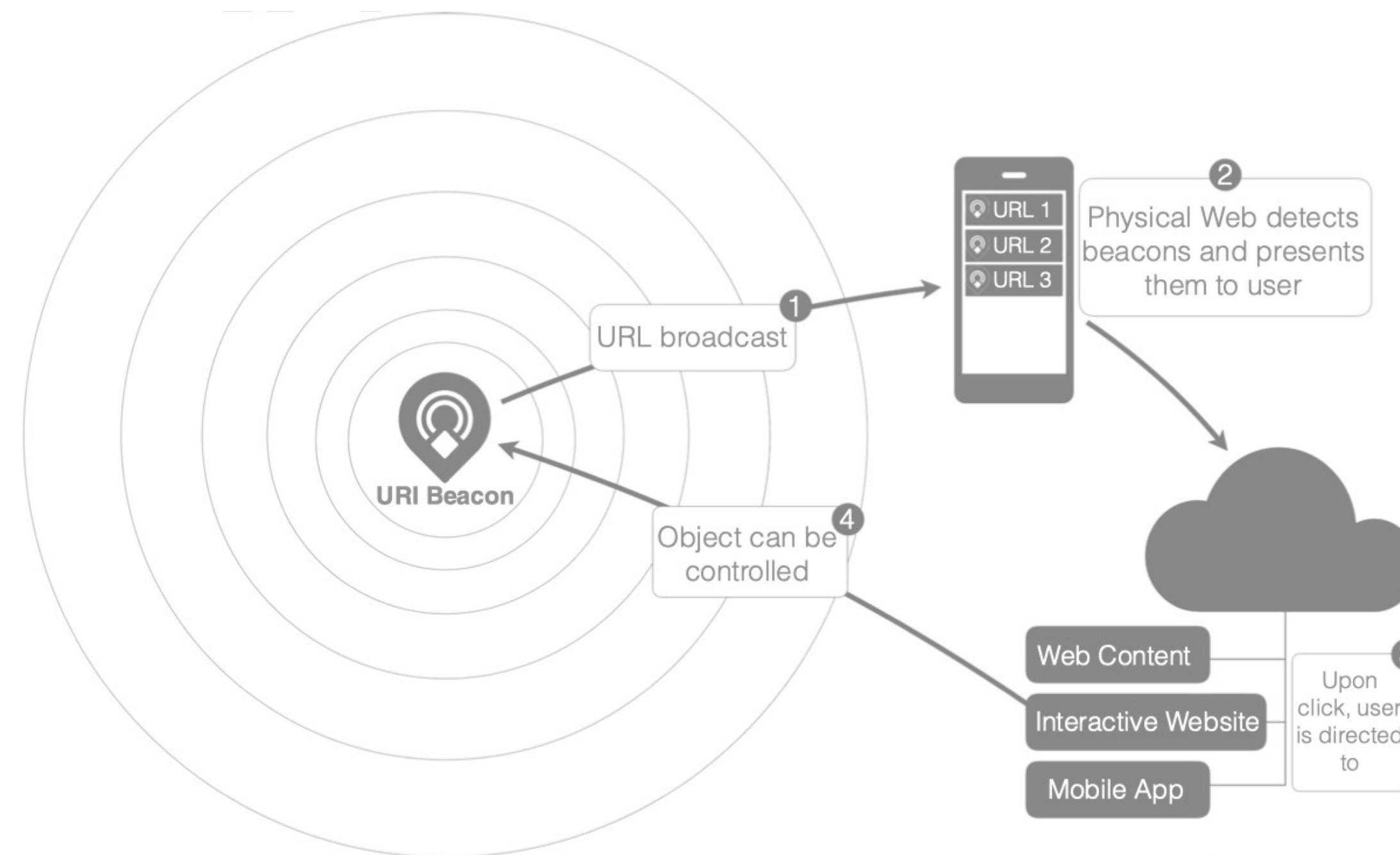
- ZeroConf service types: `_<service_name>._<protocol>.<domain>`
- `_http._tcp.local` for HTTP
- Need to define a suitable `_coap._udp.local` service type for CoAP servers
- A CoAP server advertises its presence by sending a mDNS message for the service type `_coap._udp.local` in the network
- A HTTP-to-CoAP proxy advertises its presence by sending a mDNS message for the service type `_httpcoap._udp.local` in the network
- mDNS resolution allows to find out the hostport (IP address and port) of the advertised service
- Once the hostport of the service is retrieved, a list of the maintained resources can be retrieved by sending a CoAP GET request targeting the `.well-known/core` URI
- JSON-formatted documents are a lightweight data format that can be used to describe services and resources

- With a similar intent to ZeroConf, the Universal Plug and Play (UPnP) protocol suite provides a way to perform dynamic and seamless discovery of devices in a network, without relying on DHCP and DNS servers
- UPnP has been defined by the **UPnP Forum** and uses HTTP over UDP and HTTP over multicast UDP and SOAP (Simple Object Access Protocol) to perform service description, discovery, and data transfer
- UPnP suits home appliances rather than enterprise-level deployments due to its security and efficiency issues.
- Many consumer-oriented smart objects, such as Philips Hue light bulbs, use UPnP as a zero-configuration service discovery mechanism for bridges.

- The Physical Web, a concept promoted by Google, is a different approach to provide seamless discovery and interaction with smart objects
- The assumption behind the Physical Web is that the web itself provides all the necessary means for a fruitful interaction with any endpoint, be that a website or an object
- As a consequence, the only operation that is needed in order to merge the physical world and the web is to discover the URL related to a web resource linked to a smart object
- After that, a web browser (or a client) is capable of delivering a user interface to the end user, which they can use to interact with the object (mediated by a backend that is actually connected to the object itself)
- The discovery mechanism defined by the Physical Web is based on the use of URI beacons; that is **Bluetooth Low Energy (BLE)** devices broadcasting URLs.
- The standard for data broadcasting over BLE is the **Eddystone** protocol, designed by Google
- The Eddystone protocol defines four packet types:
 - Eddystone-UID: used to contain a beacon identifier;
 - Eddystone-URL: used to broadcast a URL;
 - Eddystone-TLM: used for sending telemetry information
 - Eddystone-EID: used for carrying ephemeral IDs, in order to support protect against replay attacks or spoofing

URI Beacons and the Physical Web

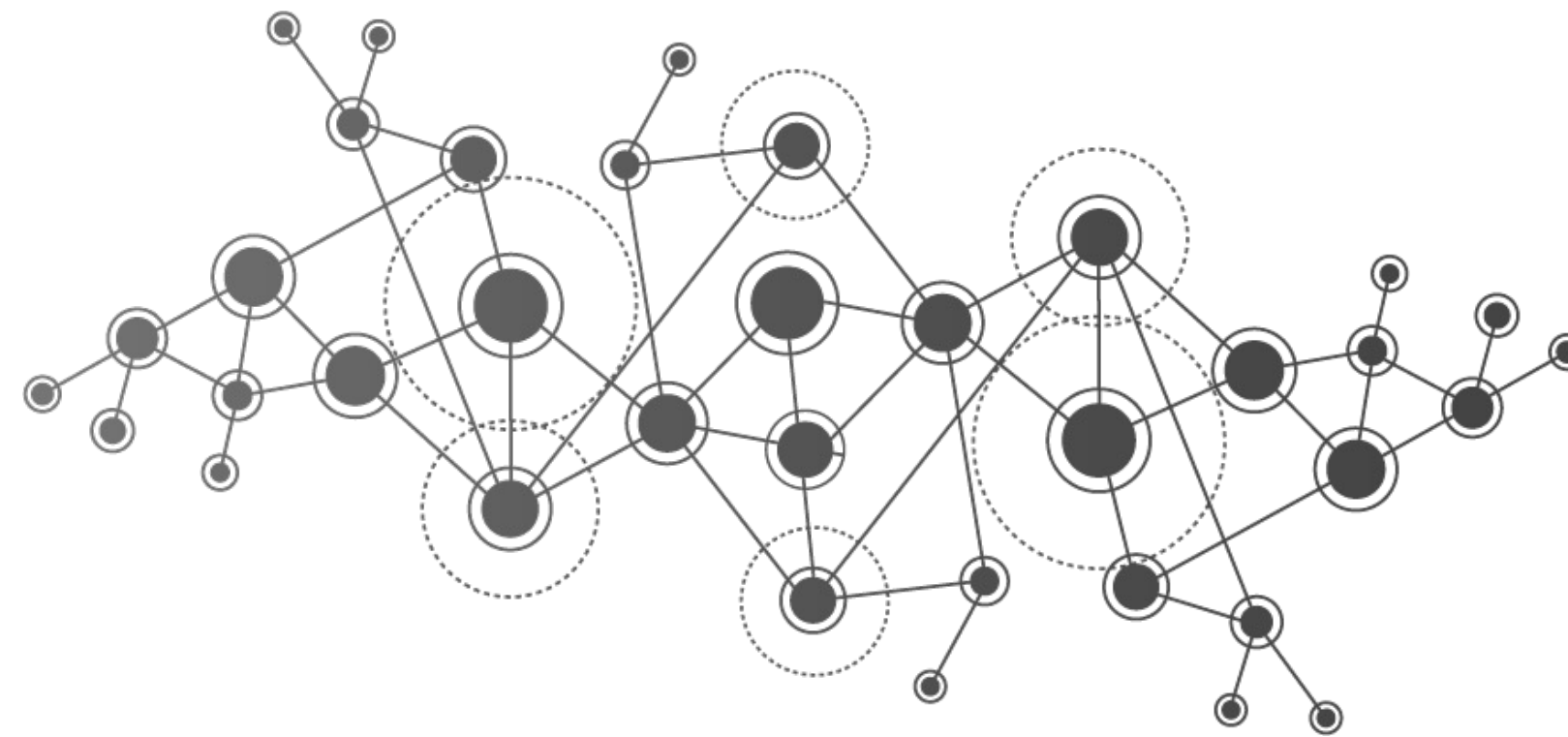
- The **advantage** in using URI beacons is the possibility to discover and interact with objects even if the user device is not connected to the same network
- However, this benefit may also become a **downside**, because the interaction with the object might not take into account context information related to the association of the user device with the network
- Moreover, it may be unsafe in some scenarios to openly broadcast object URLs: it might raise security issues and it could be impossible to restrict discovery to only authorized devices.
- The Physical Web is therefore particularly suited to public spaces, where no restricted access to objects should occur.





UNIMORE

UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA



Intelligent Internet of Things

Resource & Service Discovery

Prof. Marco Picone

A.A 2022/2023
