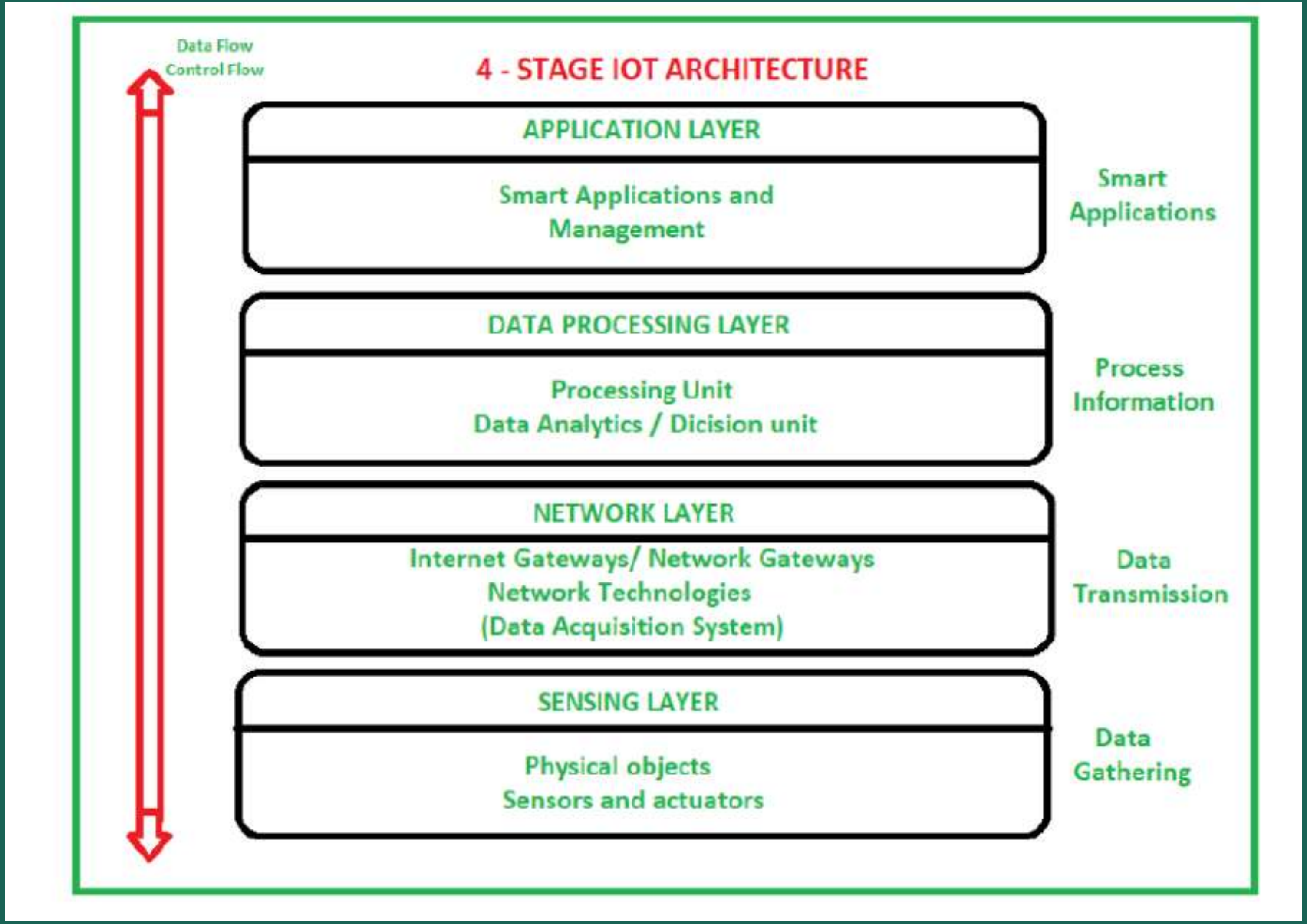
ARI 202
INTERNET OF THINGS

"Anything that can be connected, will be connected"

IOT ARCHITECTURE

- Internet of Things (IOT) technology has a wide variety of applications and use of Internet of Things is growing so faster.
- Depending upon different application areas of Internet of Things, it works accordingly as per it has been designed/developed. But it has not a standard defined architecture of working which is strictly followed universally. The architecture of IoT depends upon its functionality and implementation in different sectors.
- Still, there is a basic process flow based on which IoT is built here in this article we will discuss basic fundamental architecture of IoT i.e., 4 Stage IoT architecture. So, from the above image it is clear that there are 4 layers present that can be divided as follows: Sensing Layer, Network Layer, Data processing Layer, and Application Layer.



Data Flow
Control Flow

4 - STAGE IOT ARCHITECTURE

APPLICATION LAYER

Smart Applications and
Management

Smart
Applications

DATA PROCESSING LAYER

Processing Unit
Data Analytics / Dicision unit

Process
Information

NETWORK LAYER

Internet Gateways/ Network Gateways
Network Technologies
(Data Acquisition System)

Data
Transmission

SENSING LAYER

Physical objects
Sensors and actuators

Data
Gathering

1.Sensing Layer –

- Sensors, actuators, devices are present in this Sensing layer. These Sensors or Actuators accepts data(physical/environmental parameters), processes data and emits data over network.

2.Network Layer –

- Internet/Network gateways, Data Acquisition System (DAS) are present in this layer. DAS performs data aggregation and conversion function (Collecting data and aggregating data then converting analog data of sensors to digital data etc). Advanced gateways which mainly opens up connection between Sensor networks and Internet also performs many basic gateway functionalities like malware protection, and filtering also some times decision making based on inputted data and data management services, etc.

3.Data processing Layer –

- This is processing unit of IoT ecosystem. Here data is analyzed and pre-processed before sending it to data center from where data is accessed by software applications often termed as business applications where data is monitored and managed and further actions are also prepared. So here Edge IT or edge analytics comes into picture.

4.Application Layer –

- This is last layer of 4 stages of IoT architecture. Data centers or cloud is management stage of data where data is managed and is used by end-user applications like agriculture, health care, aerospace, farming, defense, etc. The best place to start is to define what an architecture is. Fundamentally, an architecture is a diagram or model that comprises two parts: the key technology components that make it up and the relationship between those components.

IoT architecture components:

- At a high level, the components involved in an IoT architecture include four key components.
- The applications and analytics component.
- This is the piece that processes and displays information collected via IoT. It includes analytics tools, AI and machine learning and visualization capabilities. Technologies for this component range from traditional analytics and visualization packages, such as R, IBM SPSS and SAS, to specialized IoT tools and dashboards from cloud providers, such as Amazon, Google, Microsoft, Oracle and IBM, as well as application suite vendors, including SAP and Salesforce.

1.The integration component.:-

This is the component that ensures that the applications, tools, security and infrastructure integrate effectively with existing companywide ERPand other management systems. Providers include the aforementioned software and cloud players, as well as a range of open source and middleware providers, such as Oracle Fusion Middleware, LinkSmart, Apache Kafka and DynThings Open Source IoT Platform.

2. The security and management component.:-

- IoT security includes securing the physical components of the system via firmware and embedded security providers, such as Azure Sphere, LynxOS, Mocana and Spartan.
- Traditional security vendors, such as Forescout, Symantec and Trend Micro, also offer packages that focus specifically on securing IoT.

3. The infrastructure component.

This includes physical devices -- IoT sensors, which capture information, and actuators, which control the environment.

- It also includes the network on which the sensors or actuators reside. Typically, though not always, this is a wireless network, such as Wi-Fi, 4G or 5G.The relationship between the components is the second part of the architecture.

- By relationship between, we mean how components communicate with each other, and what sorts of information are exchanged. This can include data flow, metadata flow, control information or no information at all. Software components often communicate via APIs, and network layer components typically communicate via network protocols.
- Speaking of layers, architects often think in terms of component layers, such as network layer, perception layer, processing layer, physical layer, gateway layer, platform layer, device layer, business layer, security layer, sensor layer and so on.
- The concept of a layer is that it comprises a set of capabilities that communicate with one another, but for the purpose of other components can be treated as a single entity, with a single transparent entity.
- In IoT architecture, the application layer need not know what type of physical network carries the data. All the network devices comprise the network layer that transports traffic as needed by the applications.

OPEN SOURCE ARCHITECTURE FOR IOT

- The seven high-level architecture requirements for an IoT based system are:

- 1. **Context:** Captures the context at all times, 24 hours a day, 365 days a year
- 2. **Standards:** Uses standards based communication protocols between IoT devices and enterprise systems
- 3. **Scalability:** Responds to increased load by a decline in performance, not failure; increases capacity proportionally as resources are added
- 4. **Data management:** Efficiently manages huge volumes of data
- 5. **Connectivity:** Provides high network connectivity for large data payloads and continuous streaming
- 6. **Security:** Moves and encrypts information securely even as IoT introduces new risks and vulnerabilities
- 7. **Interoperability:** Networks all systems together and ensures interoperability of all data
- According to the estimates of several experts, the IoT will consist of around 30 billion objects by 2020.

WHY OPEN SOURCE MAKES SENSE FOR IOT

- According to the estimates of several experts, the IoT will consist of around 30 billion objects by 2020.
- Today, IoT promotes the adoption of different open source technologies, standards and protocols that help devices communicate with one another. The following are the drivers that prompt organisations to adopt open source technologies for IoT.

- 1. **Cost:** Adoption of open source IoT frameworks involves no costs at all, as these are free for use. This encourages the community and organisations to implement IoT without any hesitation.
- 2. **Innovation:** Open source code from the community helps in building newer applications, leading to more innovation and agility. The developers are able to build different products, which will be interoperable across different OSs such as Android, Windows, iOS and Linux.
- 3. **Open APIs:** Use of open source APIs for the IoT framework offers a common gateway for different software, hardware and the systems to communicate with one another.
- 4. **Libraries:** An open source IoT framework offers a wide range of libraries, SDKs and open source hardware like Raspberry Pi and Arduino, ensuring that companies remain on the cutting-edge of technology by using different open sourced tools to customise IoT platforms.
- 5. **Security:** Open source software can protect individuals’ data by implementing really strong encryption for the use of the general public (SSH, SSL, PGP, etc), and hence supply the building blocks for mobile security and the protection of data.
- 6. **Interoperability:** Adoption of open source solves the problem of interoperability.

- **Open source IoT architecture**

- There is a need for organisations to provide a validated, modular, flexible IoT architecture that is built to be open, interoperable and cost effective. The architecture should deliver end-to-end open source IoT that addresses enterprise level IoT needs.

The characteristics of open source IoT architecture are:

- Loosely coupled, modular and secure
- Platform independent
- Scalable, flexible and can be deployed anywhere
- Based on open standards
- Streaming analytics and machine learning
- Open and interoperable on the hybrid cloud
- Application agility and integration
- No vendor lock-in, since there are no rigid architectures or proprietary formats and components

Following are some of the **Architecture Design Principles** related to IoT system design that I have come across in many years of designing IoT systems. This list is only indicative and in no way definitive for any particular IoT solution or domain. Here you go:

1. Standardized Integration

- Some enterprises mandate the use of an enterprise service bus for all communication between systems while others may restrict integration to using API only. Whichever integration mechanism is employed, such a mandate ensures external and enterprise systems are integrated uniformly which enhances the maintainability of the overall IT inventory. Based on the enterprise mandate, a standard integration mechanism could be a guiding principle to follow while designing IoT solutions. It can considerably increase the speed of on-boarding new systems thereby adding more value to an IoT ecosystem.

2. Platform Agnosticism

- While architecting IoT solutions based on AWS, Azure, GCP or any other cloud platform, the usage of PaaS services binds the solutions to that vendor. Avoiding such a vendor lock-in may be part of the enterprise directive but it entails that the full capability of the underlying platform is under utilized. A hybrid approach is sometimes followed where the business logic is at least made more portable than the rest of the components. For instance, using Kubernetes based on serverless (Functions as a Service) components instead of Azure Functions.

3. Data Residency and Compliance

- Data Residency is a requirement that is gaining prominence across all organizations implementing IoT or any other cloud based solution. Set as a principle, it affects the choice of components or sometimes even the cloud provider itself, for implementing the solution. Some cloud services are not available in all regions or countries and in some cases a service could be available in a country but sub-processing of data by that service may happen outside the boundary of the country. Such strong mandates affect the vendor selection, technical architecture and design choices quite significantly.

4. Using existing investments

- This is especially more frequent in case of Big Data Analytics and Data lake existing in an enterprise. Data Analytics and Big Data technologies have existed even before IoT entered the mainstream. Though all IoT platforms are capable of processing and storing big data and applying analytical models on this data, such capability already exists in many enterprises. The idea behind this principle is to utilize existing investments on such functionality.

5. Prefer Edge Intelligence to Cloud Processing

- With most cloud platforms offering machine learning and serverless constructs on the edge, the acceptance of edge as a processing node is more than ever. The primary benefit being that this form of distributed computing reduces processing load in the cloud. Additionally, edge computing also reduces the network bandwidth consumption of sending every bit of raw data to the backend.

6. Security by design

- Device and Data security is paramount in ensuring end to end security of an IoT system. Data security involves protecting data in transit and at rest. Device security usually involves device identity and authentication based on security certificates. With such a directive in place, all IoT development happens defensively. And hence it influences the low level design of IoT functions.

7. Real time processing of Alarms/Alerts

- While the usual telemetry data from devices can take a path of ingestion, storage, processing, analytics and visualization, the device alarms and alerts must take an emergency path to notify business apps or configured email/sms route for quicker action. This entails multiple data processing pipelines with varying degrees of preferential treatment.

8. Support multiple connectivity protocols

- High variability in devices and device types that connect to an IoT platform necessitates a protocol agnostic approach to connectivity. An enterprise grade IoT platform is expected to ingest and process data riding on any transport and with any data structure.

9. Support switch off mode

- This mode implies that the system works even if there is a loss in connectivity between devices and platforms. It may result in reduced functionality but wouldn’t cause complete disruption of system availability. Such a feature is made possible by services like Device Shadow which maintain the current and intended state of devices. Devices and gateways are also designed so that they can sustain operations for a long time without being connected to the backend platform.

10. No downtime

- It essentially means high availability of close to say 99.99% for an IoT solution. Many cloud services follow SLAs of 99.95% or less and therefore to reach the four 9s availability, the overall design of the platforms requires ingenious design. Such highly fault tolerant systems must take care of disaster recovery by ensuring redundancy of each service and all data across availability zones and regions.

IOT ARCHITECTURE REFERENCE MODEL

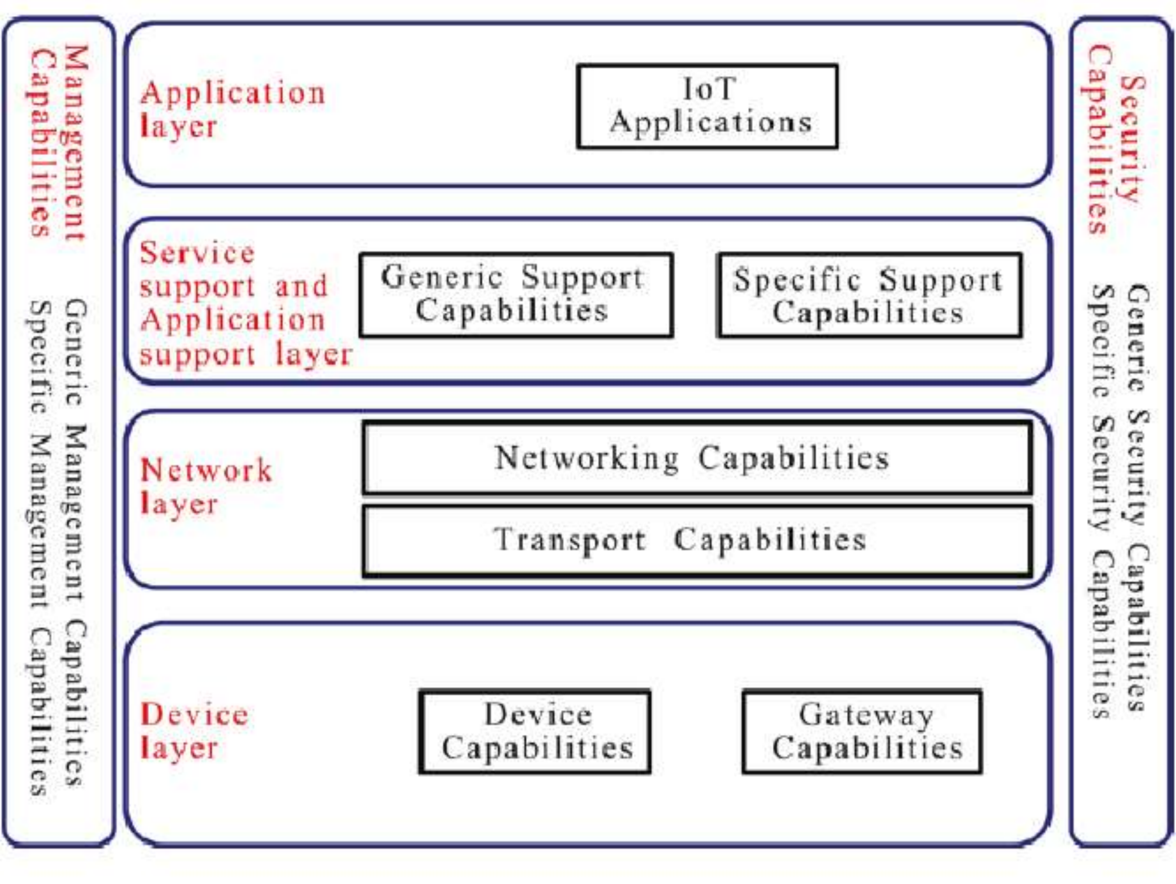
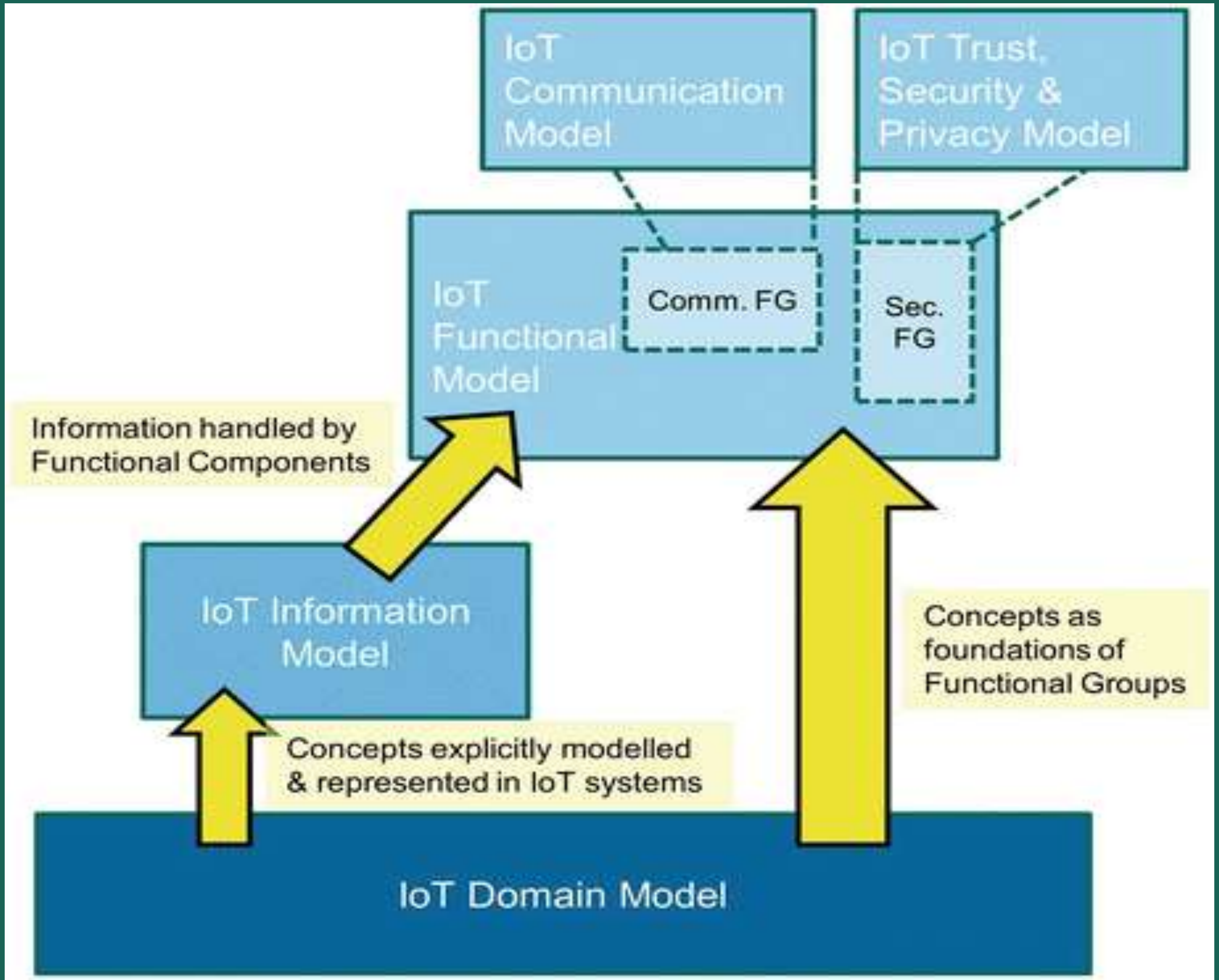
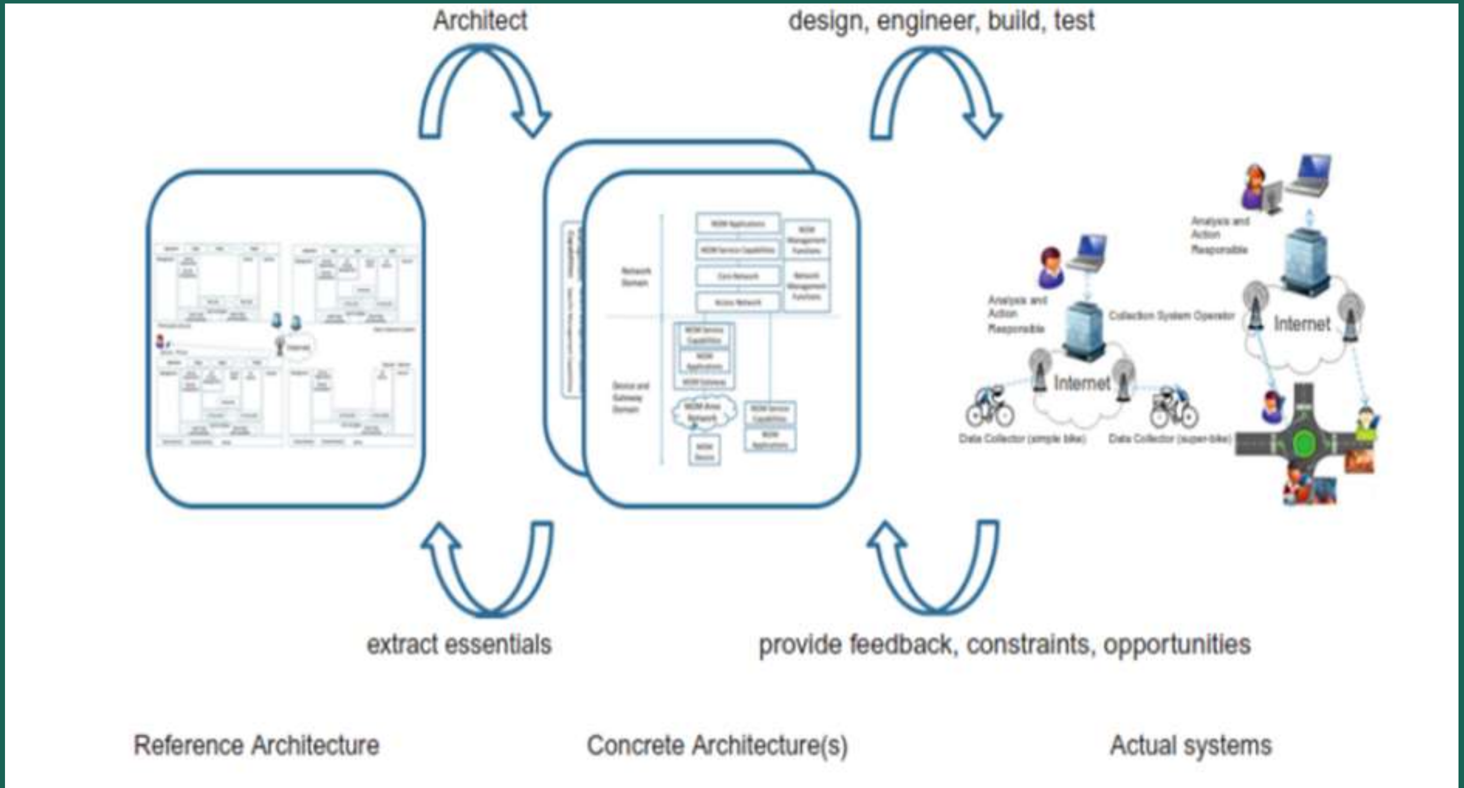

Management Capabilities
Generic Management Capabilities
Specific Management Capabilities
Application layer
IoT Applications
Service support and Application support layer
Generic Support Capabilities
Specific Support Capabilities
Network layer
Networking Capabilities
Transport Capabilities
Device layer
Device Capabilities
Gateway Capabilities
Security Capabilities
Generic Security Capabilities
Specific Security Capabilities

Fig. 2. ITU-T recommended IoT Reference Model

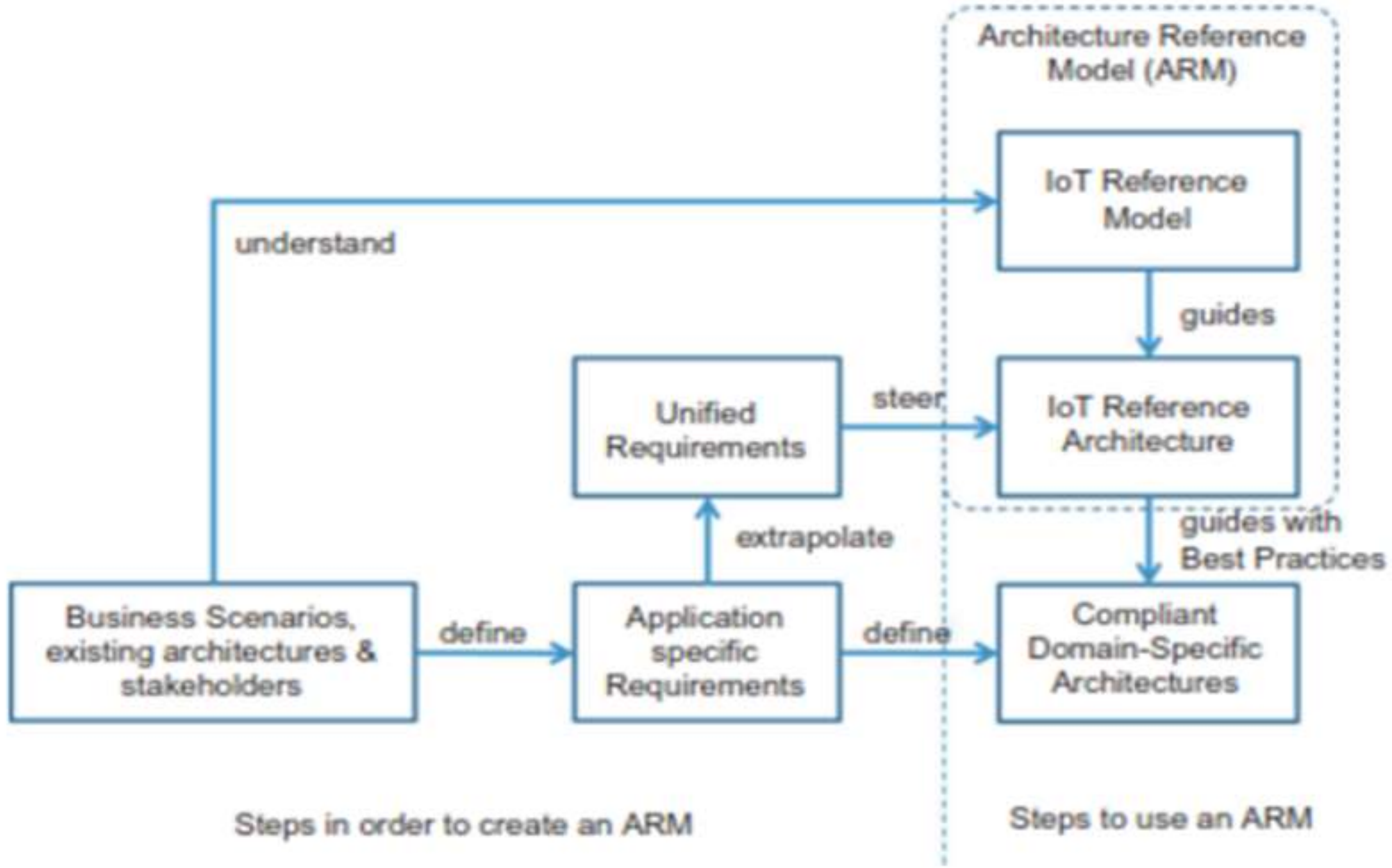
- An ARM consists of two main parts: a Reference model and a Reference Architecture.

- •A reference model describes the domain using a number of sub-models

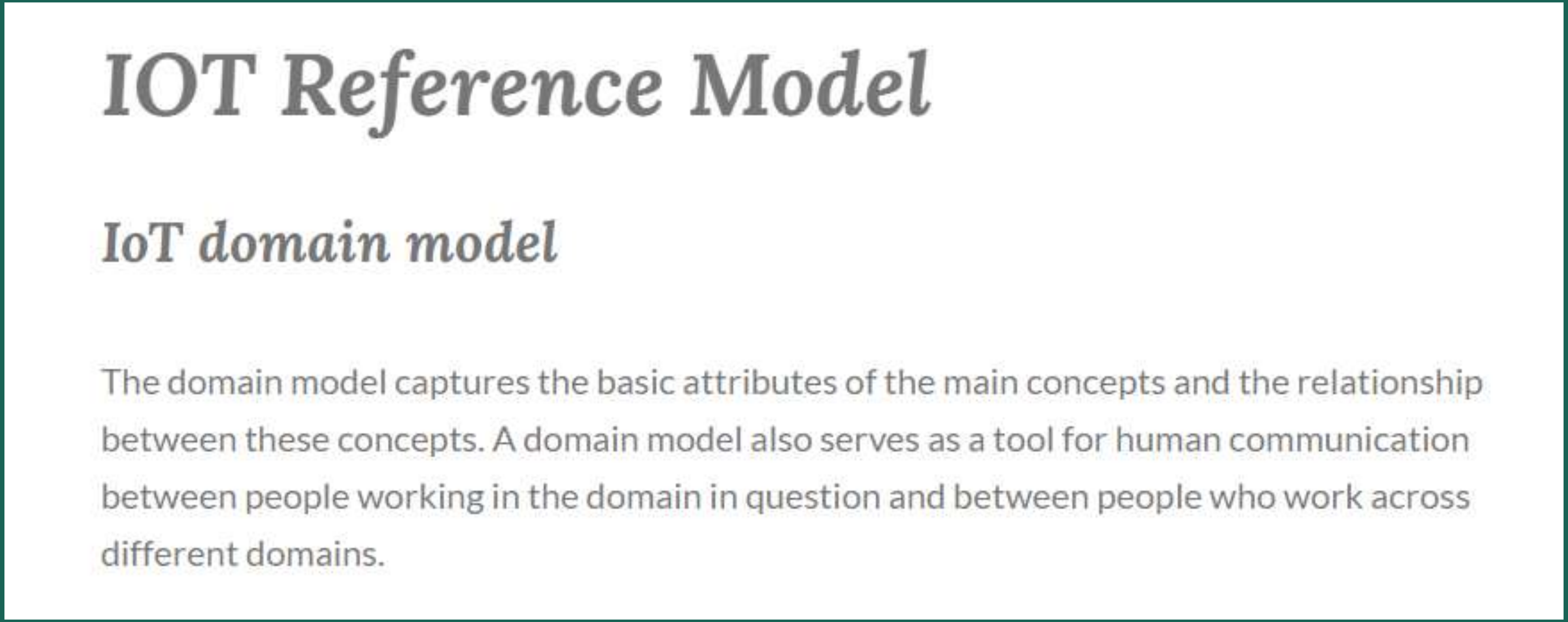

IoT Communication Model
IoT Trust, Security & Privacy Model
IoT Functional Model
Comm. FG
Sec. FG
Information handled by Functional Components
IoT Information Model
Concepts as foundations of Functional Groups
Concepts explicitly modelled & represented in IoT systems
IoT Domain Model


Architect
design, engineer, build, test
Analysis and Action Responsible
Collection System Operator
Internet
Analysis and Action Responsible
Internet
Data Collector (simple bike)
Data Collector (super-bike)
extract essentials
provide feedback, constraints, opportunities
Reference Architecture
Concrete Architecture(s)
Actual systems

From Reference to concrete architecture and actual system


Architecture Reference Model (ARM)
IoT Reference Model
understand
guides
Unified Requirements
steer
IoT Reference Architecture
extrapolate
guides with Best Practices
Business Scenarios, existing architectures & stakeholders
define
Application specific Requirements
define
Compliant Domain-Specific Architectures
Steps in order to create an ARM
Steps to use an ARM

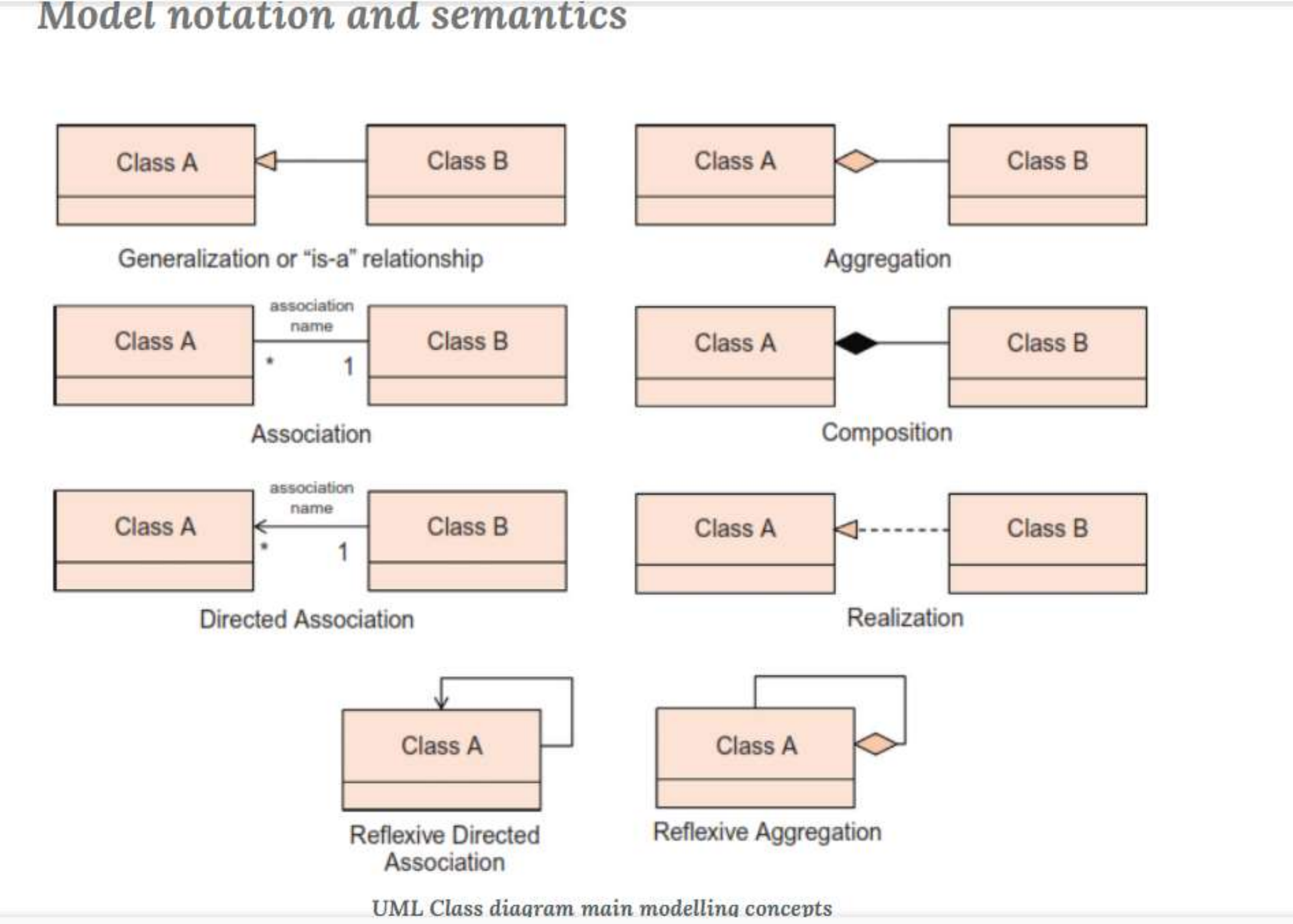
IOT Reference architecture and reference model dependency



IOT Reference Model

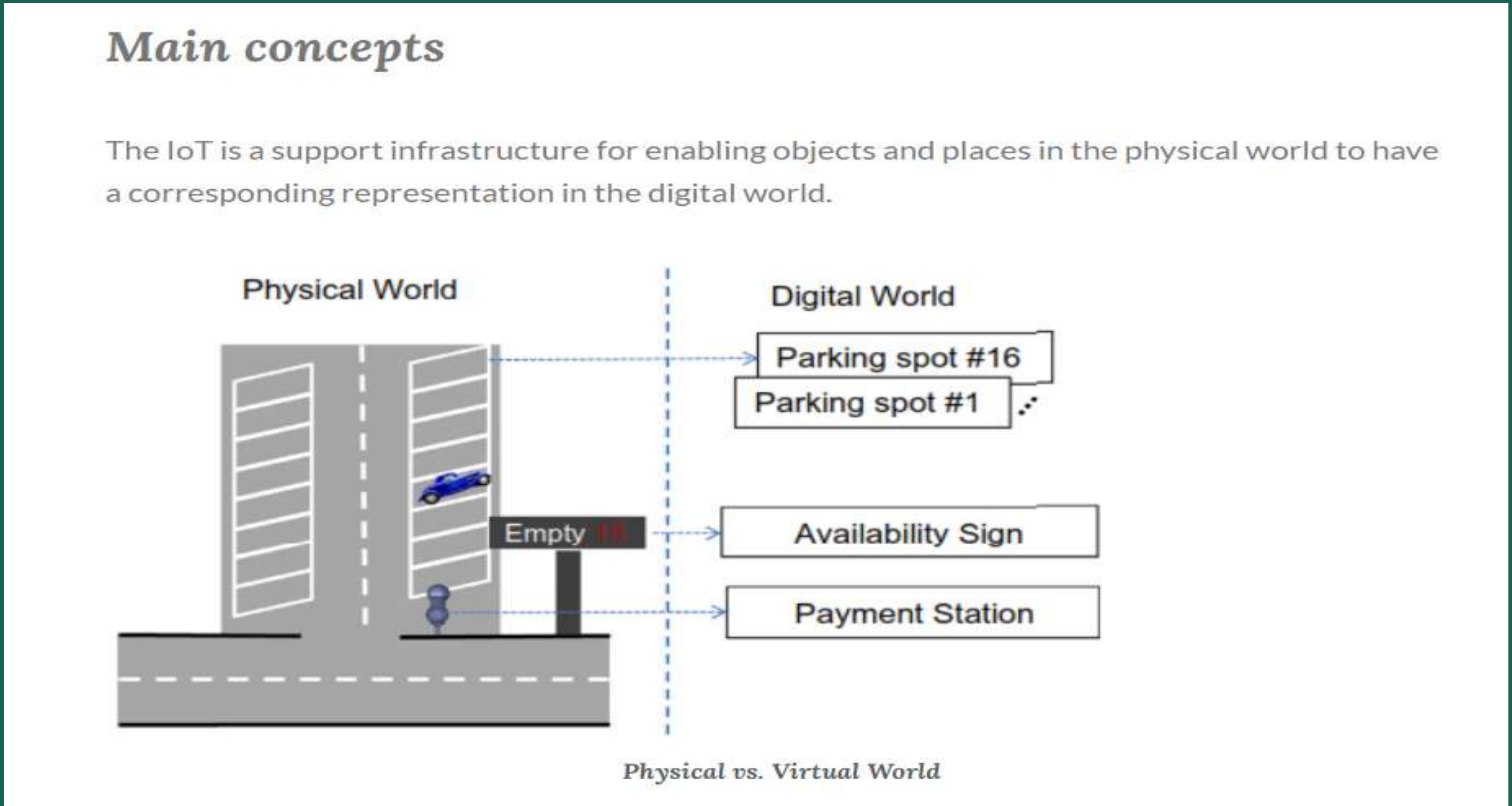
IoT domain model

The domain model captures the basic attributes of the main concepts and the relationship between these concepts. A domain model also serves as a tool for human communication between people working in the domain in question and between people who work across different domains.



Model notation and semantics

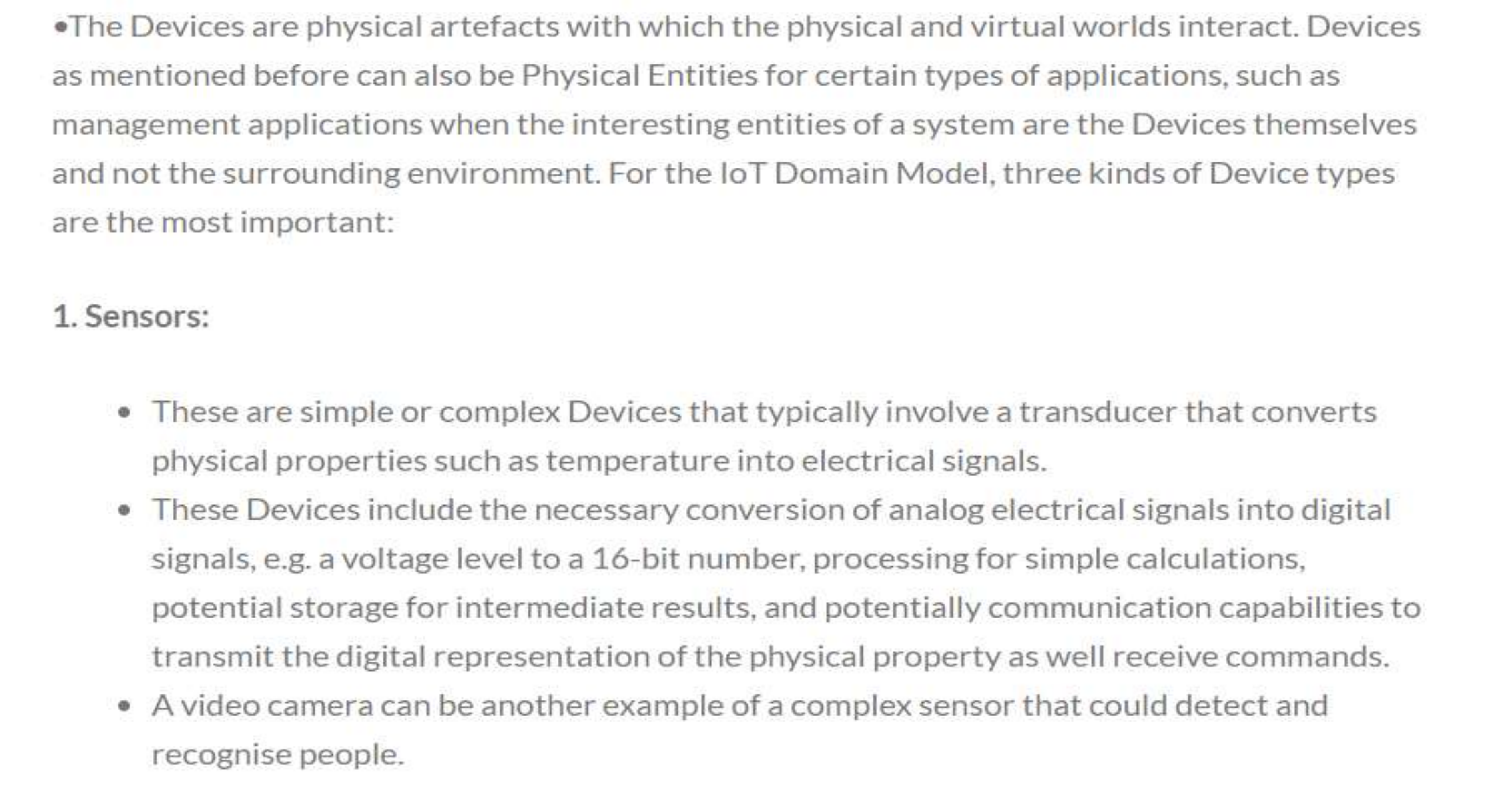
UML Class diagram main modelling concepts



Main concepts

The IoT is a support infrastructure for enabling objects and places in the physical world to have a corresponding representation in the digital world.

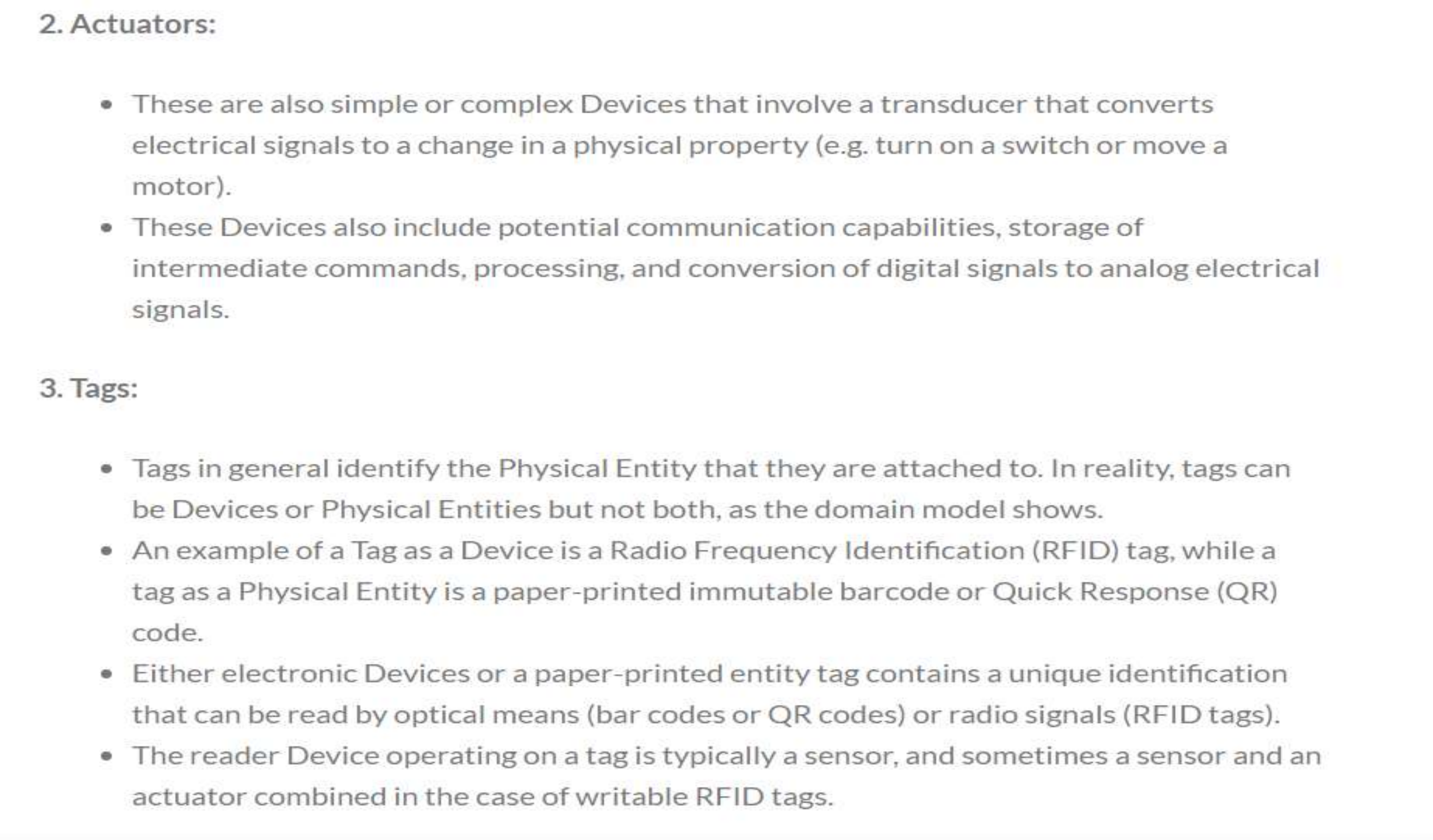
Physical vs. Virtual World



•The Devices are physical artefacts with which the physical and virtual worlds interact. Devices as mentioned before can also be Physical Entities for certain types of applications, such as management applications when the interesting entities of a system are the Devices themselves and not the surrounding environment. For the IoT Domain Model, three kinds of Device types are the most important:

1. Sensors:

- These are simple or complex Devices that typically involve a transducer that converts physical properties such as temperature into electrical signals.
- These Devices include the necessary conversion of analog electrical signals into digital signals, e.g. a voltage level to a 16-bit number, processing for simple calculations, potential storage for intermediate results, and potentially communication capabilities to transmit the digital representation of the physical property as well receive commands.
- A video camera can be another example of a complex sensor that could detect and recognise people.

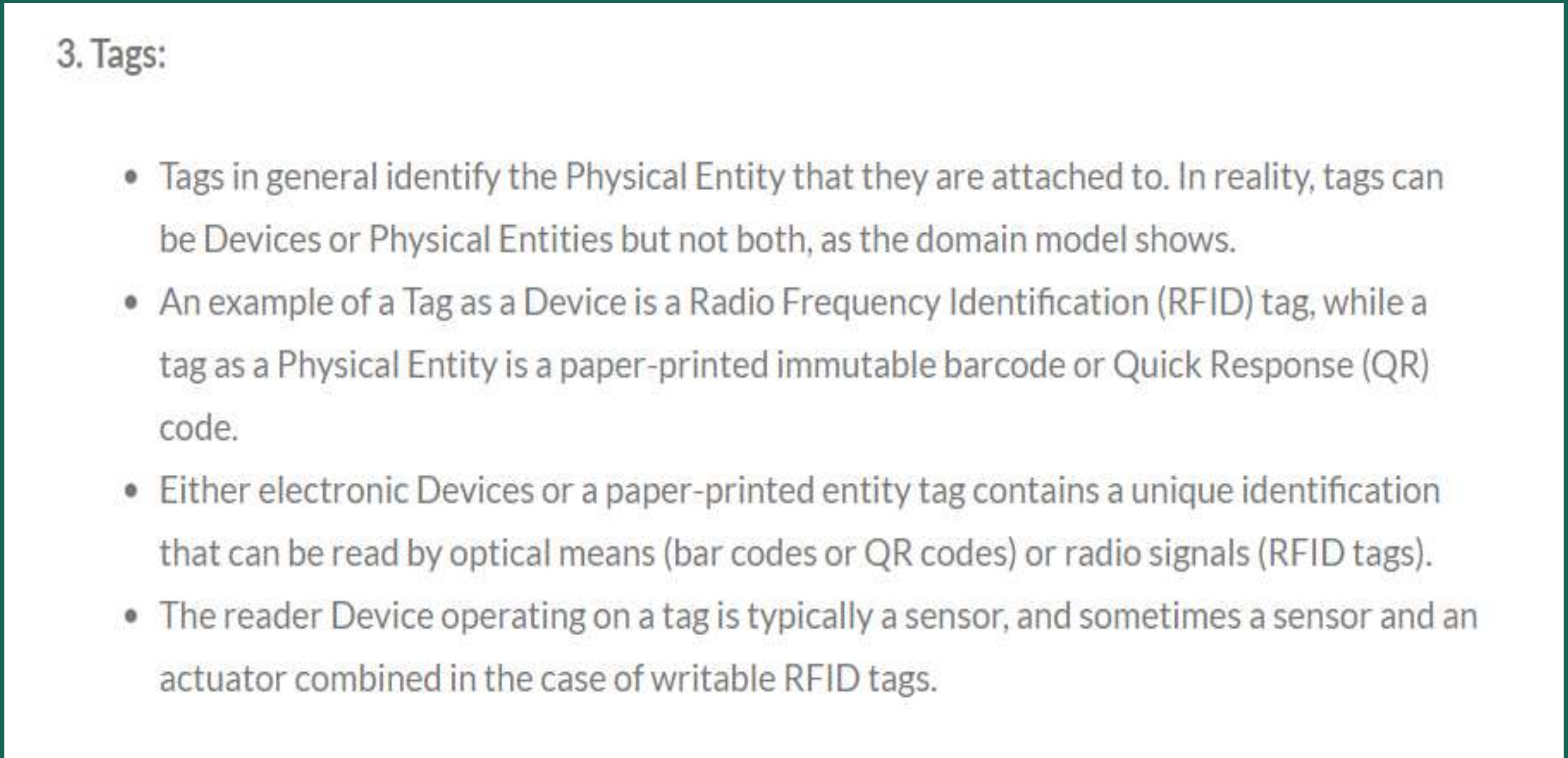


2. Actuators:

- These are also simple or complex Devices that involve a transducer that converts electrical signals to a change in a physical property (e.g. turn on a switch or move a motor).
- These Devices also include potential communication capabilities, storage of intermediate commands, processing, and conversion of digital signals to analog electrical signals.

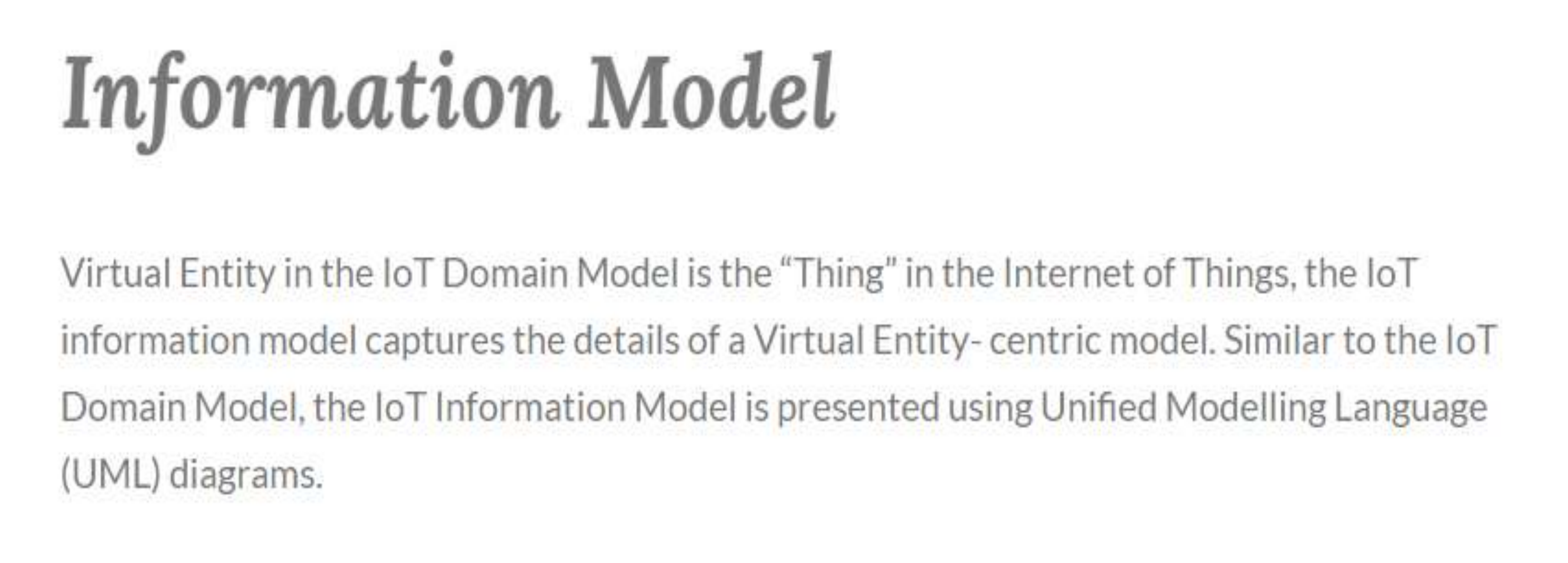
3. Tags:

- Tags in general identify the Physical Entity that they are attached to. In reality, tags can be Devices or Physical Entities but not both, as the domain model shows.
- An example of a Tag as a Device is a Radio Frequency Identification (RFID) tag, while a tag as a Physical Entity is a paper-printed immutable barcode or Quick Response (QR) code.
- Either electronic Devices or a paper-printed entity tag contains a unique identification that can be read by optical means (bar codes or QR codes) or radio signals (RFID tags).
- The reader Device operating on a tag is typically a sensor, and sometimes a sensor and an actuator combined in the case of writable RFID tags.



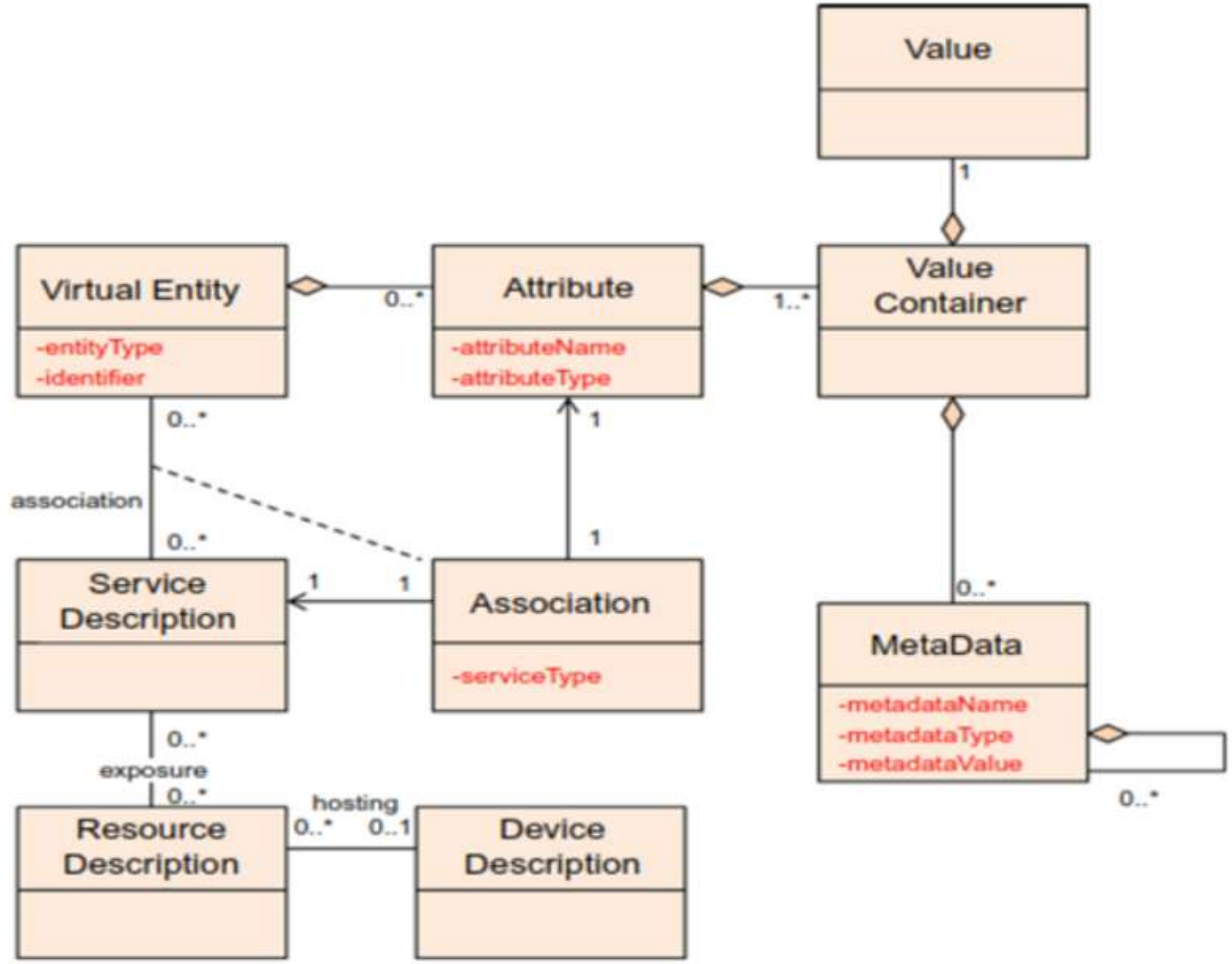
3. Tags:

- Tags in general identify the Physical Entity that they are attached to. In reality, tags can be Devices or Physical Entities but not both, as the domain model shows.
- An example of a Tag as a Device is a Radio Frequency Identification (RFID) tag, while a tag as a Physical Entity is a paper-printed immutable barcode or Quick Response (QR) code.
- Either electronic Devices or a paper-printed entity tag contains a unique identification that can be read by optical means (bar codes or QR codes) or radio signals (RFID tags).
- The reader Device operating on a tag is typically a sensor, and sometimes a sensor and an actuator combined in the case of writable RFID tags.



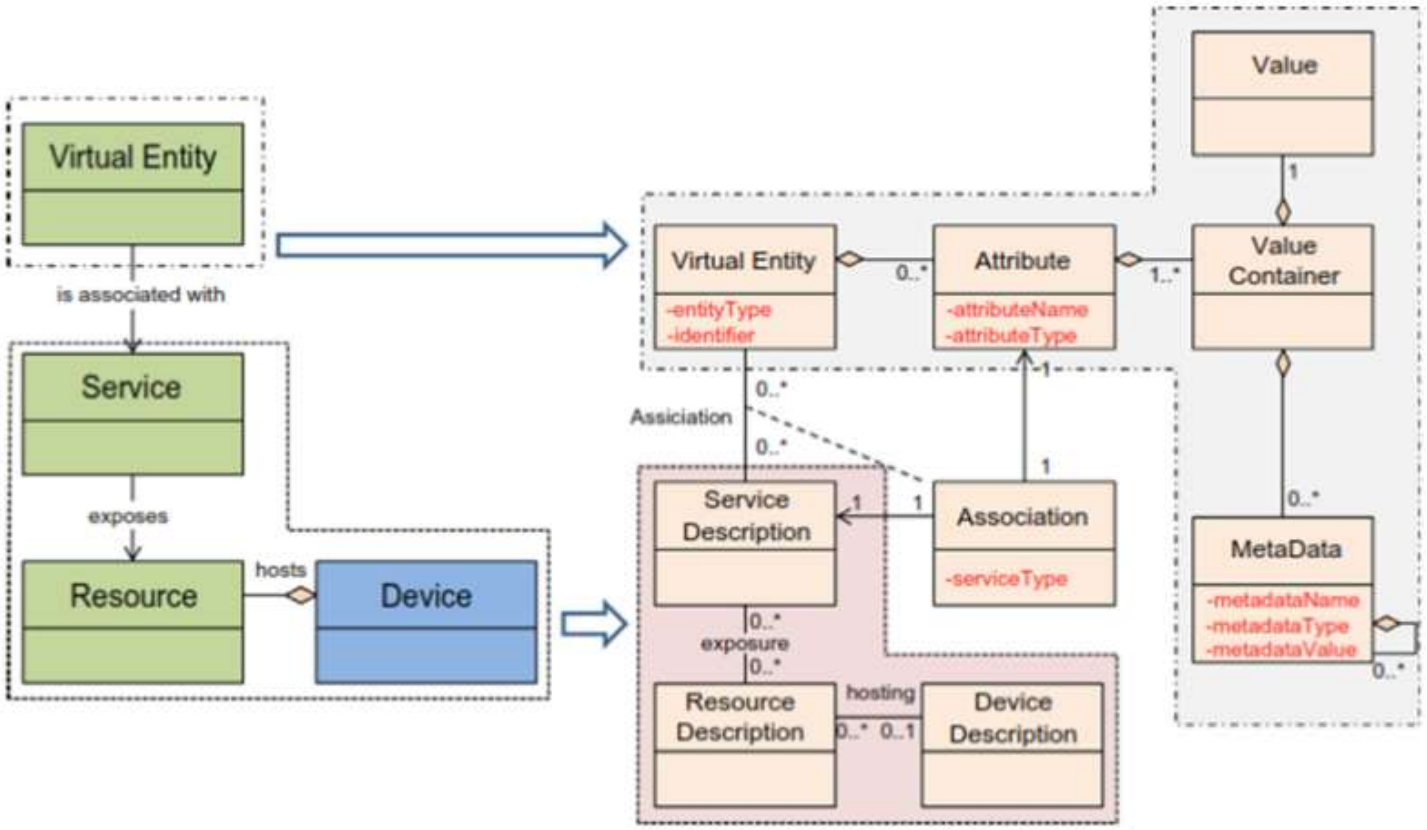
Information Model

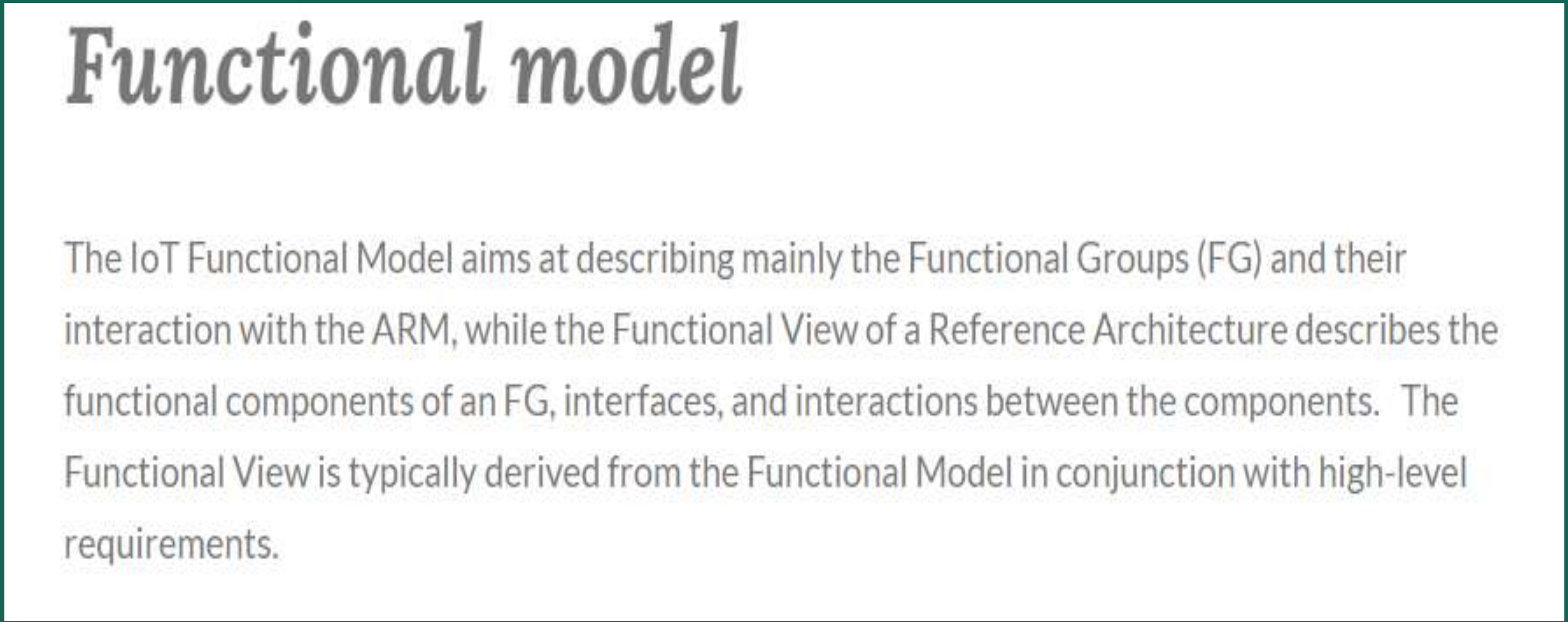
Virtual Entity in the IoT Domain Model is the “Thing” in the Internet of Things, the IoT information model captures the details of a Virtual Entity- centric model. Similar to the IoT Domain Model, the IoT Information Model is presented using Unified Modelling Language (UML) diagrams.



High-level IoT Information Model

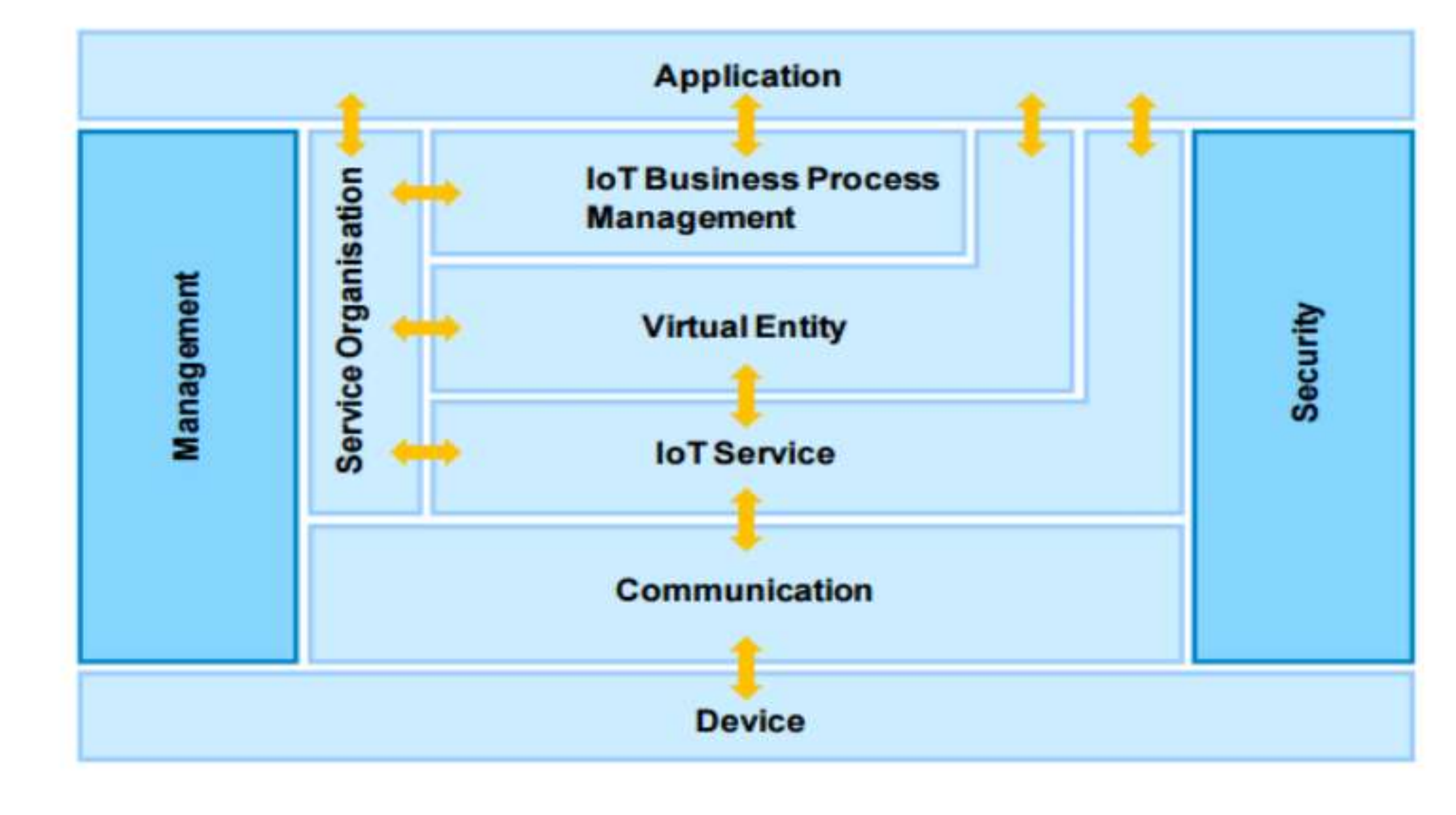
Relationship between core concepts of IoT Domain Model and IoT Information Model.

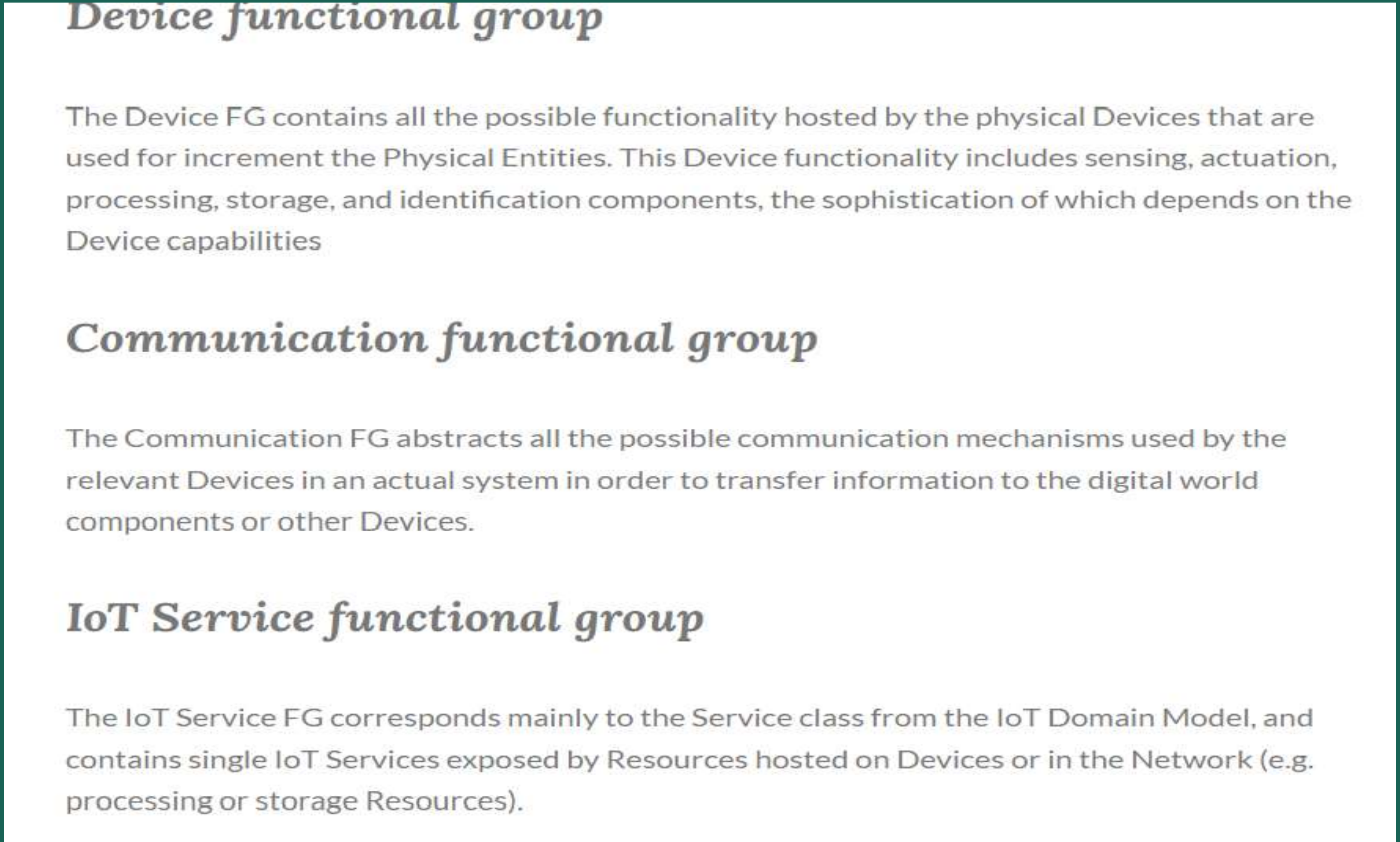




Functional model

The IoT Functional Model aims at describing mainly the Functional Groups (FG) and their interaction with the ARM, while the Functional View of a Reference Architecture describes the functional components of an FG, interfaces, and interactions between the components. The Functional View is typically derived from the Functional Model in conjunction with high-level requirements.


Application
IoT Business Process Management
Virtual Entity
IoT Service
Communication
Device
Management
Service Organisation
Security



Device functional group

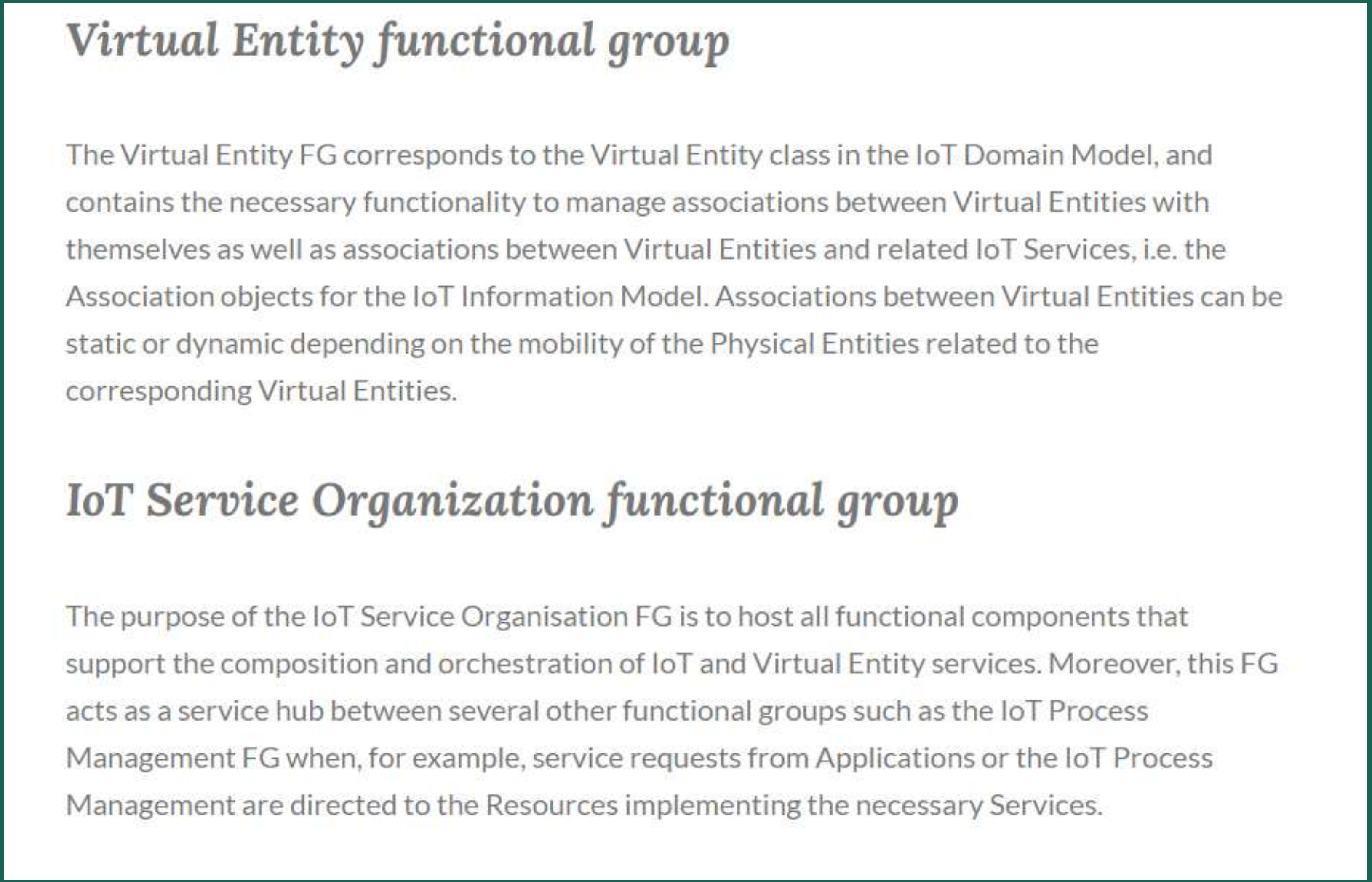
The Device FG contains all the possible functionality hosted by the physical Devices that are used for increment the Physical Entities. This Device functionality includes sensing, actuation, processing, storage, and identification components, the sophistication of which depends on the Device capabilities

Communication functional group

The Communication FG abstracts all the possible communication mechanisms used by the relevant Devices in an actual system in order to transfer information to the digital world components or other Devices.

IoT Service functional group

The IoT Service FG corresponds mainly to the Service class from the IoT Domain Model, and contains single IoT Services exposed by Resources hosted on Devices or in the Network (e.g. processing or storage Resources).

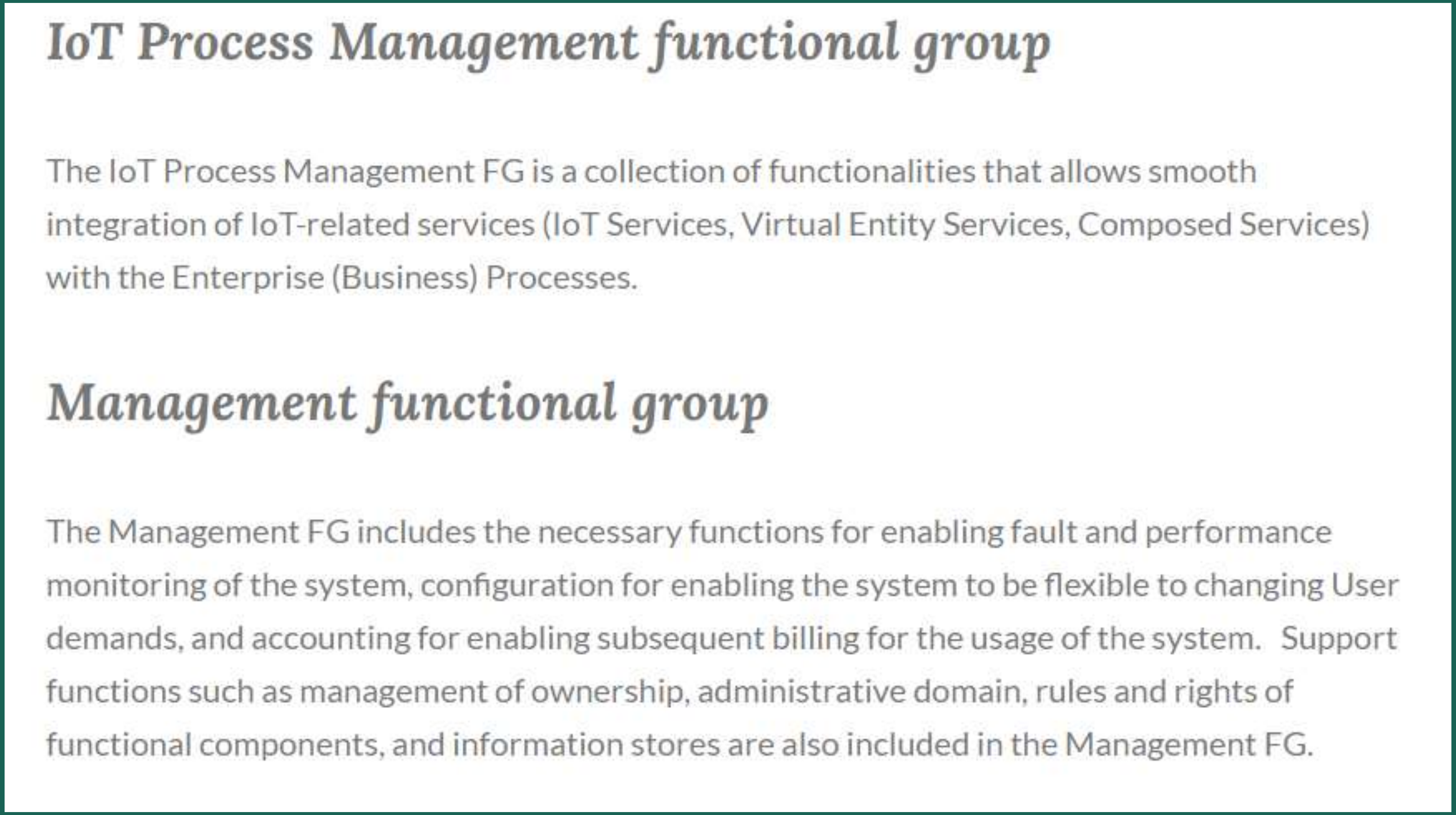


Virtual Entity functional group

The Virtual Entity FG corresponds to the Virtual Entity class in the IoT Domain Model, and contains the necessary functionality to manage associations between Virtual Entities with themselves as well as associations between Virtual Entities and related IoT Services, i.e. the Association objects for the IoT Information Model. Associations between Virtual Entities can be static or dynamic depending on the mobility of the Physical Entities related to the corresponding Virtual Entities.

IoT Service Organization functional group

The purpose of the IoT Service Organisation FG is to host all functional components that support the composition and orchestration of IoT and Virtual Entity services. Moreover, this FG acts as a service hub between several other functional groups such as the IoT Process Management FG when, for example, service requests from Applications or the IoT Process Management are directed to the Resources implementing the necessary Services.

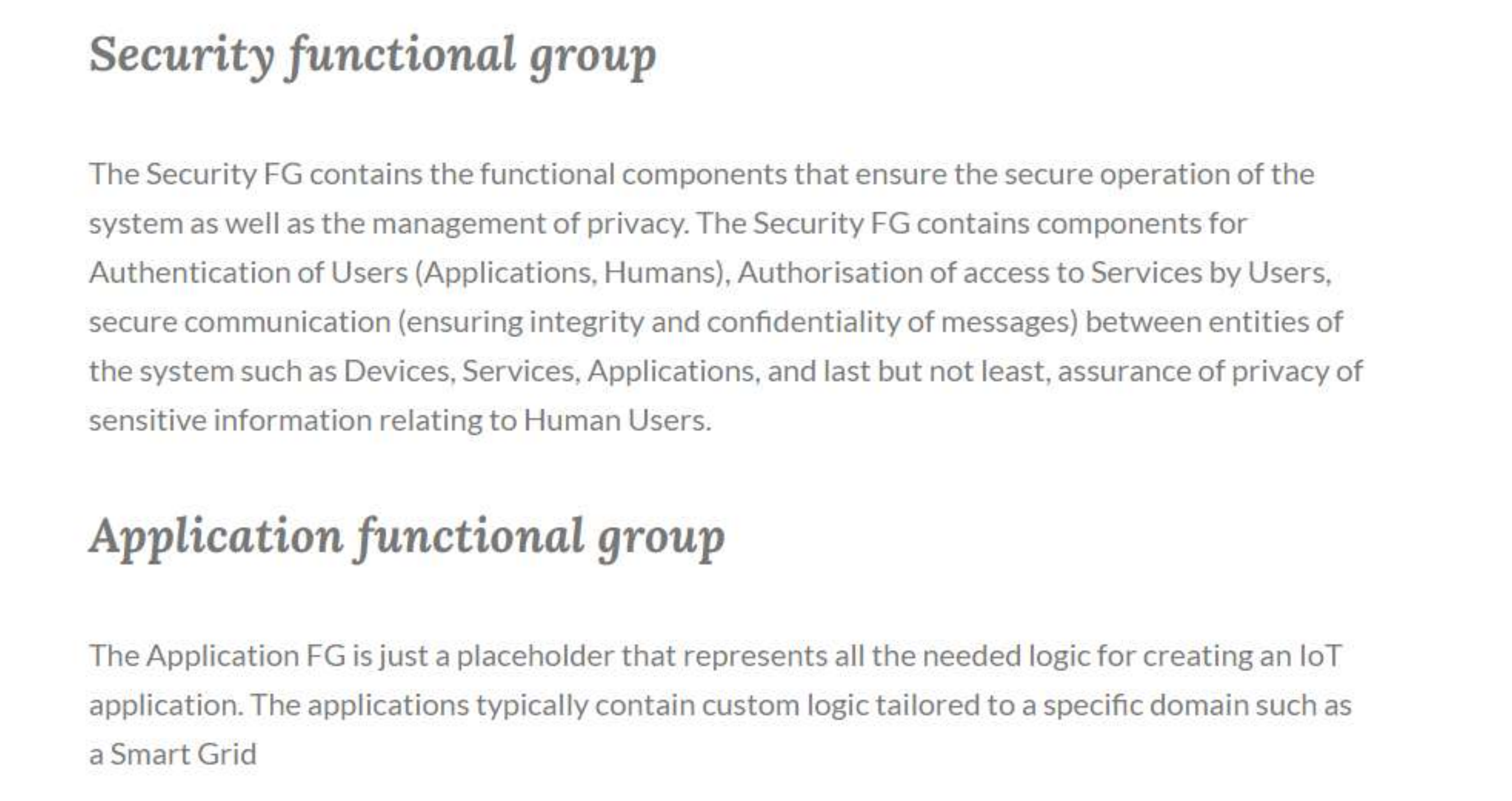


IoT Process Management functional group

The IoT Process Management FG is a collection of functionalities that allows smooth integration of IoT-related services (IoT Services, Virtual Entity Services, Composed Services) with the Enterprise (Business) Processes.

Management functional group

The Management FG includes the necessary functions for enabling fault and performance monitoring of the system, configuration for enabling the system to be flexible to changing User demands, and accounting for enabling subsequent billing for the usage of the system. Support functions such as management of ownership, administrative domain, rules and rights of functional components, and information stores are also included in the Management FG.

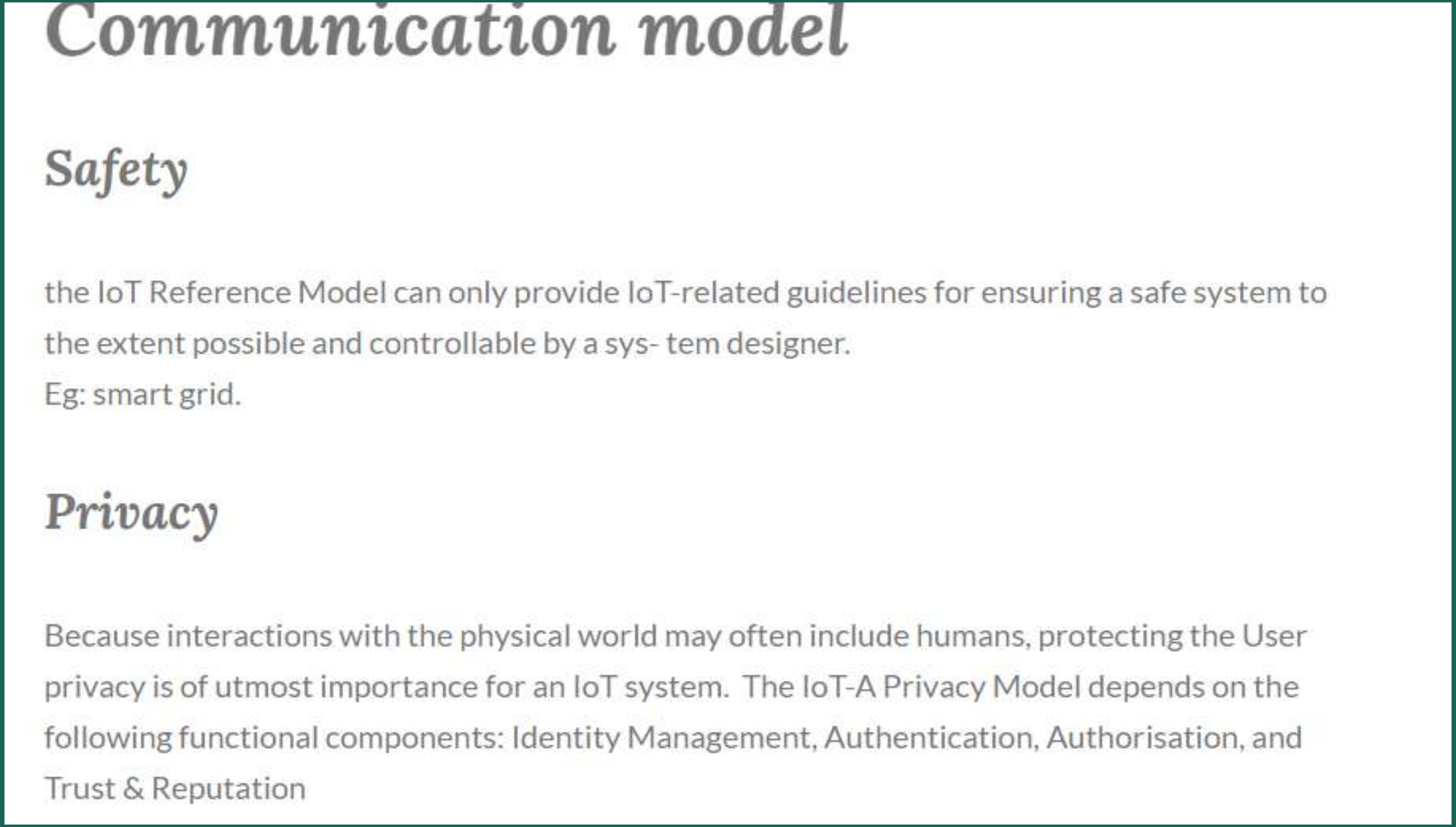


Security functional group

The Security FG contains the functional components that ensure the secure operation of the system as well as the management of privacy. The Security FG contains components for Authentication of Users (Applications, Humans), Authorisation of access to Services by Users, secure communication (ensuring integrity and confidentiality of messages) between entities of the system such as Devices, Services, Applications, and last but not least, assurance of privacy of sensitive information relating to Human Users.

Application functional group

The Application FG is just a placeholder that represents all the needed logic for creating an IoT application. The applications typically contain custom logic tailored to a specific domain such as a Smart Grid



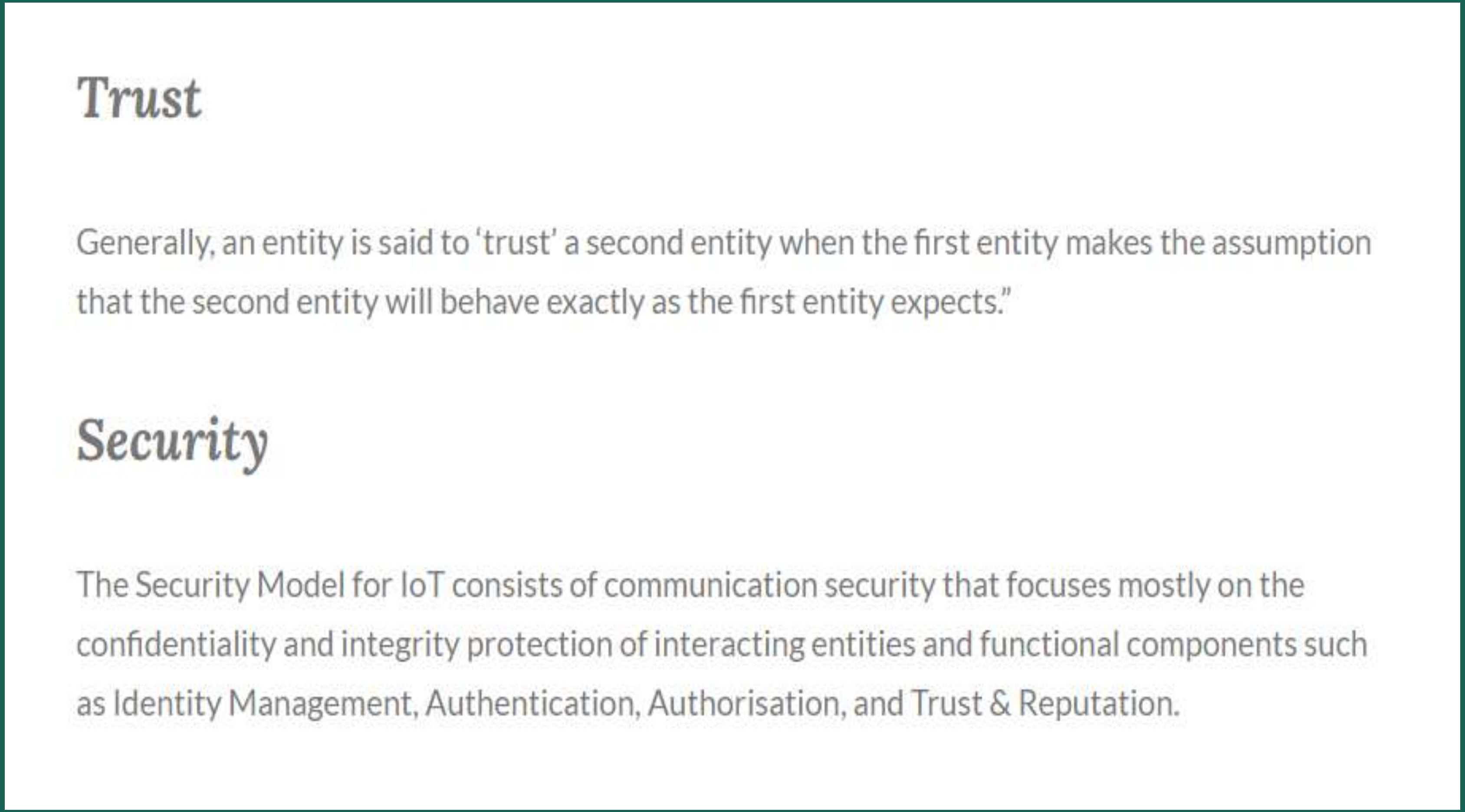
Communication model

Safety

the IoT Reference Model can only provide IoT-related guidelines for ensuring a safe system to the extent possible and controllable by a sys- tem designer.
Eg: smart grid.

Privacy

Because interactions with the physical world may often include humans, protecting the User privacy is of utmost importance for an IoT system. The IoT-A Privacy Model depends on the following functional components: Identity Management, Authentication, Authorisation, and Trust & Reputation



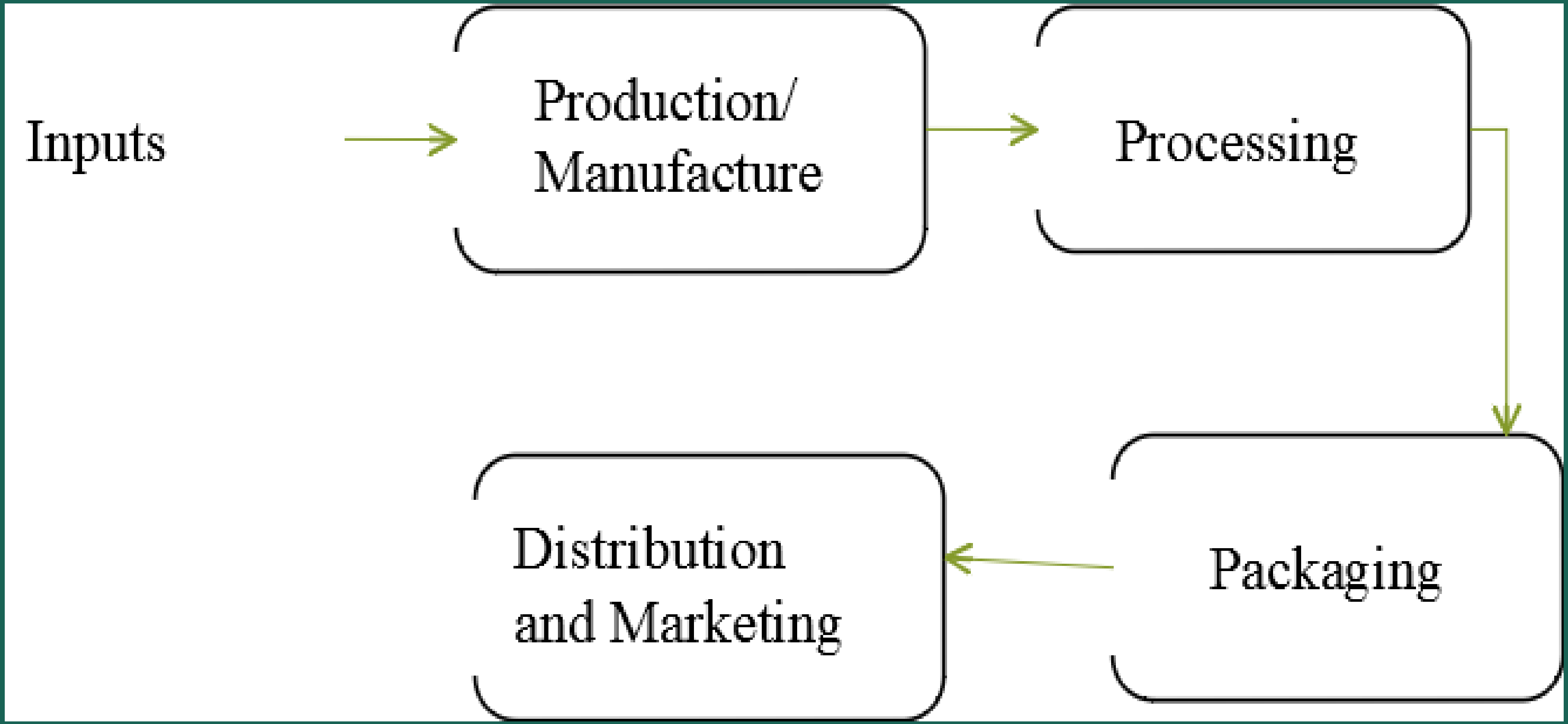
Trust

Generally, an entity is said to ‘trust’ a second entity when the first entity makes the assumption that the second entity will behave exactly as the first entity expects.”

Security

The Security Model for IoT consists of communication security that focuses mostly on the confidentiality and integrity protection of interacting entities and functional components such as Identity Management, Authentication, Authorisation, and Trust & Reputation.

M2M VALUE CHAIN



Inputs

Production/ Manufacture

Processing

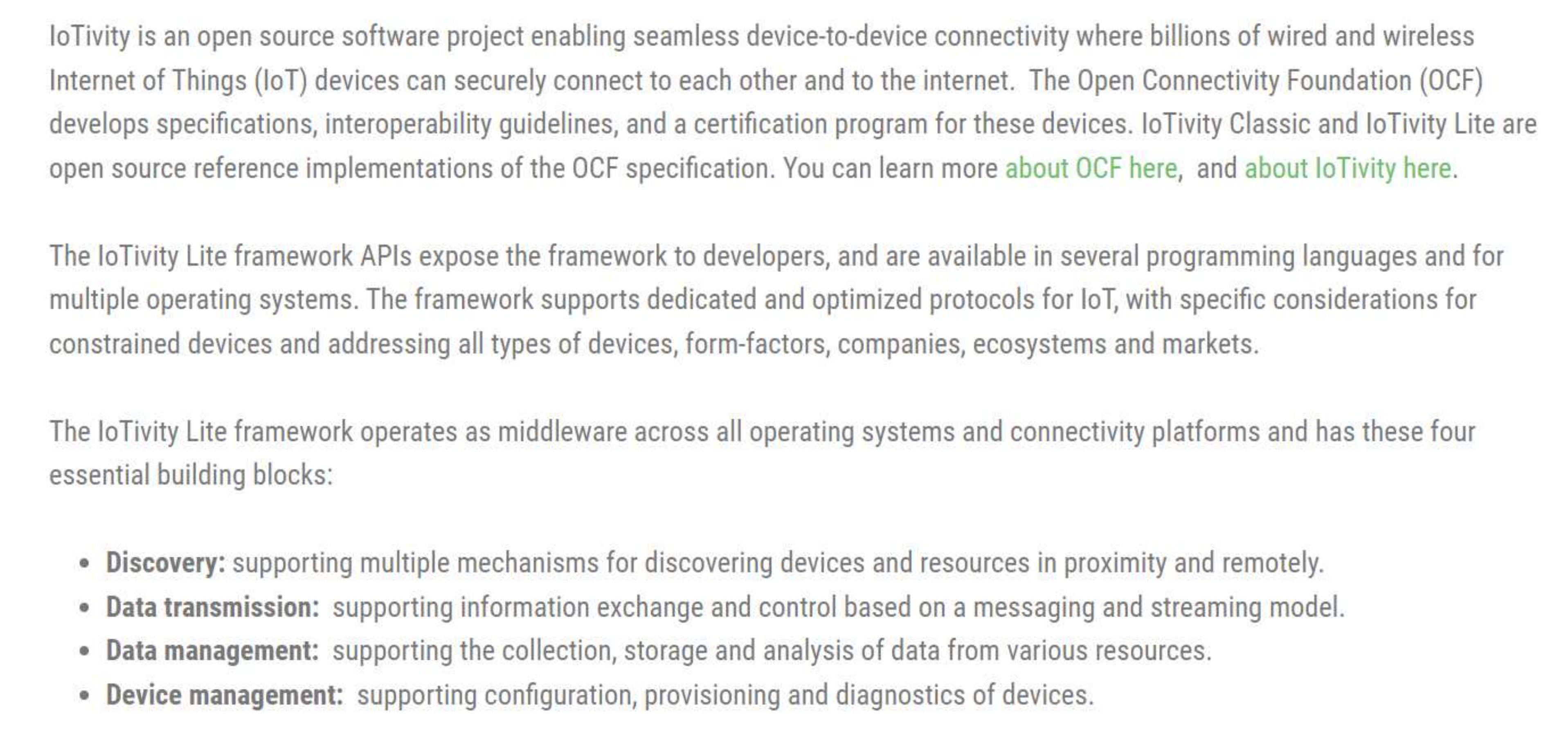
Packaging

Distribution and Marketing

IOT VALUE CHAIN

The picture can't be displayed.

IoTIVITY



IoTivity is an open source software project enabling seamless device-to-device connectivity where billions of wired and wireless Internet of Things (IoT) devices can securely connect to each other and to the internet. The Open Connectivity Foundation (OCF) develops specifications, interoperability guidelines, and a certification program for these devices. IoTivity Classic and IoTivity Lite are open source reference implementations of the OCF specification. You can learn more about OCF here, and about IoTivity here.

The IoTivity Lite framework APIs expose the framework to developers, and are available in several programming languages and for multiple operating systems. The framework supports dedicated and optimized protocols for IoT, with specific considerations for constrained devices and addressing all types of devices, form-factors, companies, ecosystems and markets.

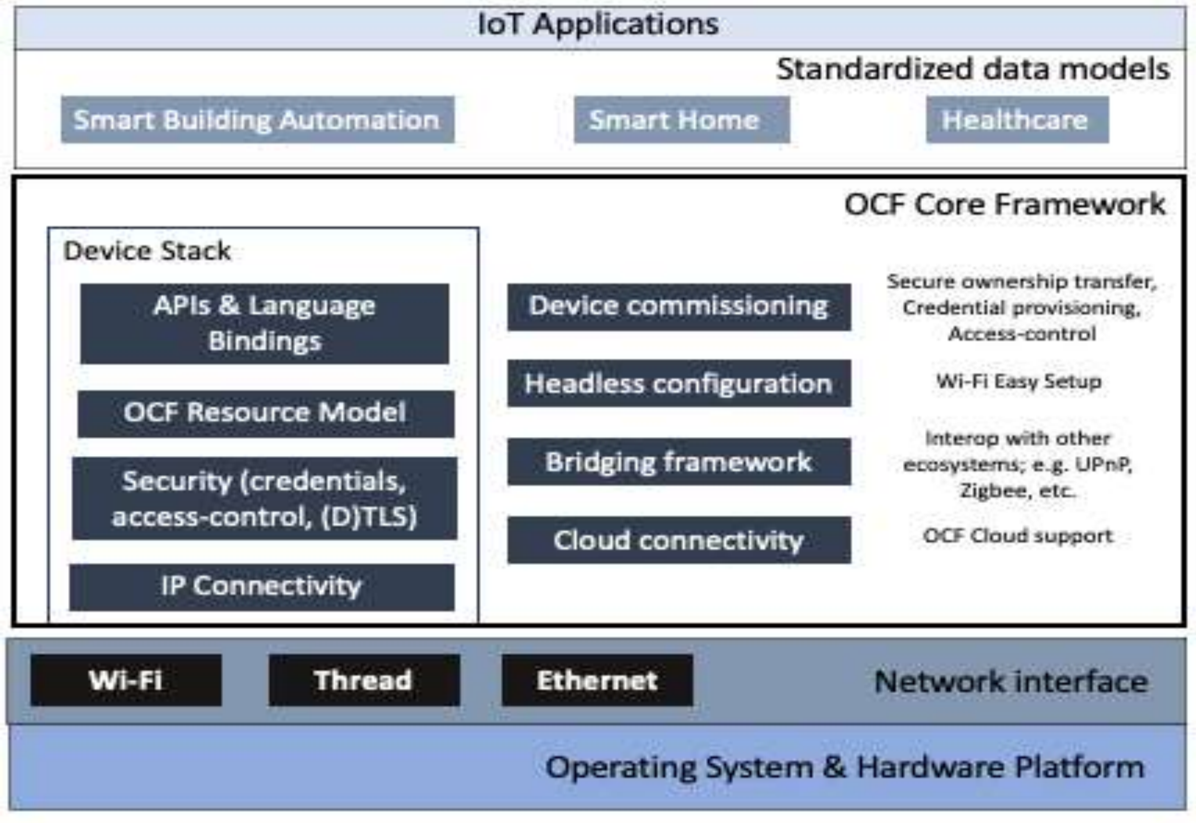
The IoTivity Lite framework operates as middleware across all operating systems and connectivity platforms and has these four essential building blocks:

- **Discovery:** supporting multiple mechanisms for discovering devices and resources in proximity and remotely.
- **Data transmission:** supporting information exchange and control based on a messaging and streaming model.
- **Data management:** supporting the collection, storage and analysis of data from various resources.
- **Device management:** supporting configuration, provisioning and diagnostics of devices.

- IoTivity is an open-source, reference implementation of the OCF Secure IP Device Framework.

- The OCF Secure IP Device Framework provides a versatile communications layer with best-in-class security for Device-to-Device (D2D) and Device-to-Cloud (D2C) connectivity over IP. IoT interoperability is achieved through the use of consensus-derived, industry standard data models spanning an array of usage verticals. The OCF Secure IP Device Framework may be harnessed alongside other IoT technologies in a synergistic fashion to lend a comprehensive and robust IoT solution.

IOTIVITY STACK FEATURES



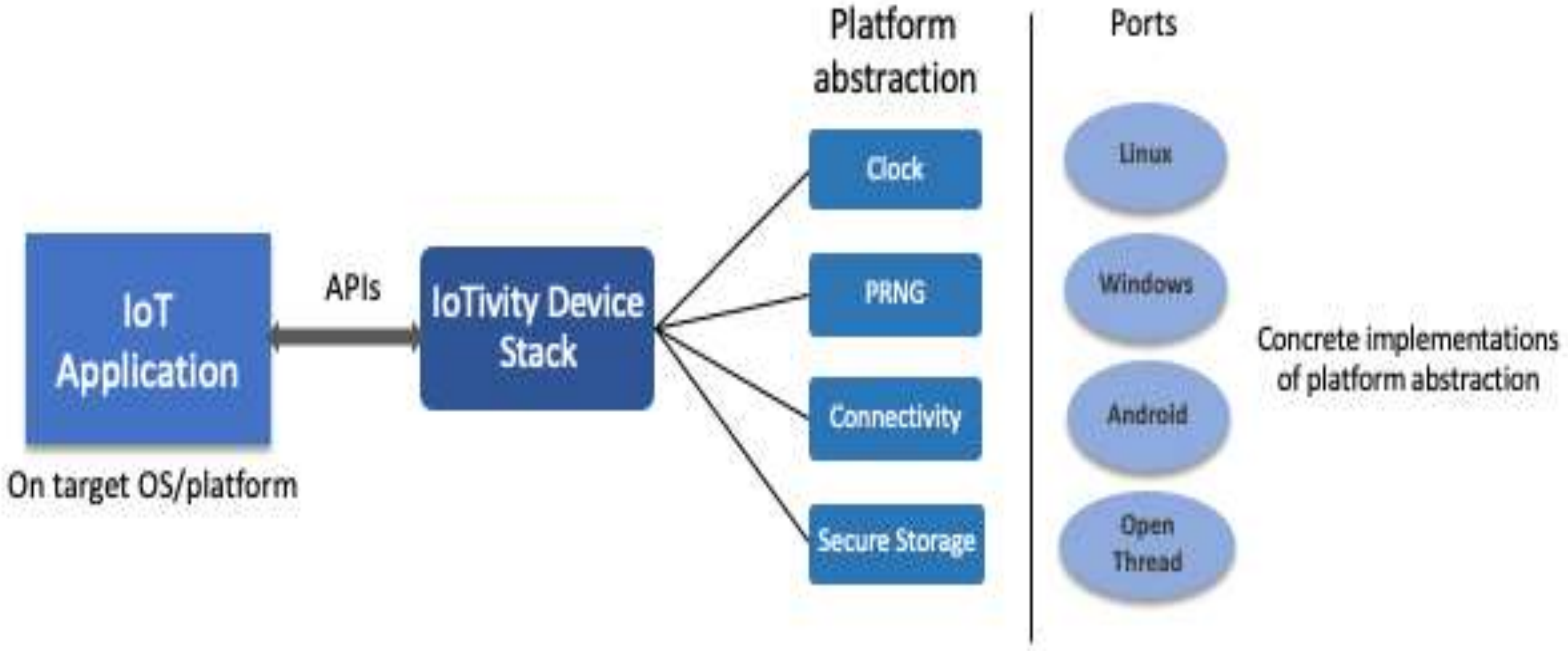
IoTivity device stack and modules

- OS agnostic: The IoTivity device stack and modules work cross-platform (pure C code) and execute in an event-driven style. The stack interacts with lower level OS/hardware platform-specific functionality through a set of abstract interfaces. This decoupling of the common OCF standards related functionality from platform adaptation code promotes ease of long-term maintenance and evolution of the stack through successive releases of the OCF specifications.

- Porting layer: The platform abstraction is a set of generically defined interfaces which elicit a specific contract from implementations. The stack utilizes these interfaces to interact with the underlying OS/platform. The simplicity and boundedness of these interface definitions allow them to be rapidly implemented on any chosen OS/target. Such an implementation constitutes a "port".

- Optional support for static memory: On minimal environments lacking heap allocation functions, the stack may be configured to statically allocate all internal structures by setting a number of build-time parameters, which by consequence constrain the allowable workload for an application.

- C and Java APIs: The API structure and naming closely aligns with OCF specification constructs, aiding ease of understanding.



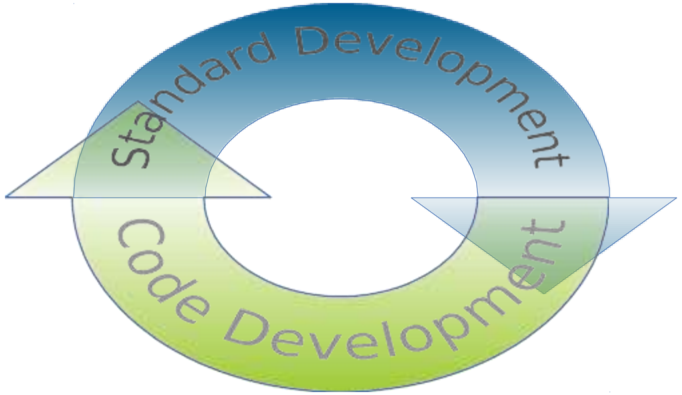
IoTivity was designed for rapid portability to any deployment target

Outline

- What is IoTivity and why is it useful?
- IoTivity stack architecture
- IoTivity resource model and request-response flow
- Role in the IoT ecosystem
- Cross-platform support

What is IoTivity?

OPEN INTERCONNECT CONSORTIUM℠



Standard Development

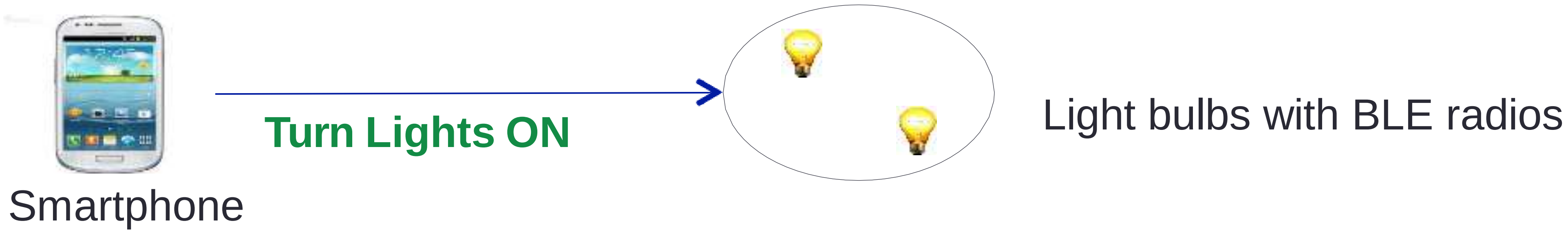
Code Development

IoTivity

- Open source framework and SDK for building IoT applications

- Independently governed

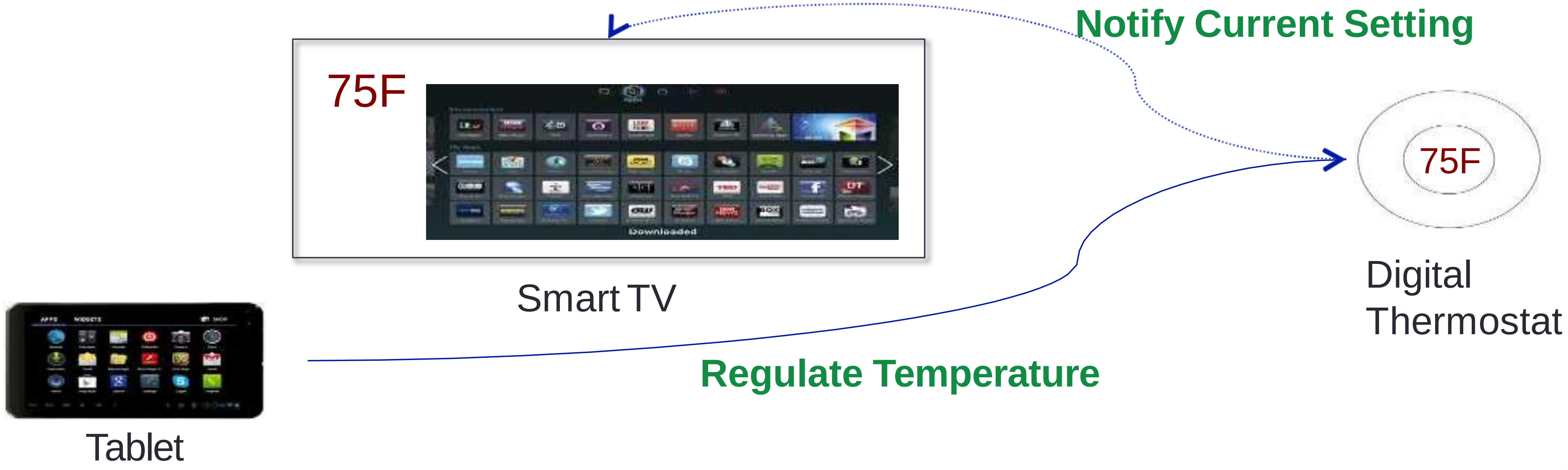
Why is it useful?



Turn Lights ON

Light bulbs with BLE radios

Smartphone



Notify Current Setting

75F

Smart TV

75F

Digital Thermostat

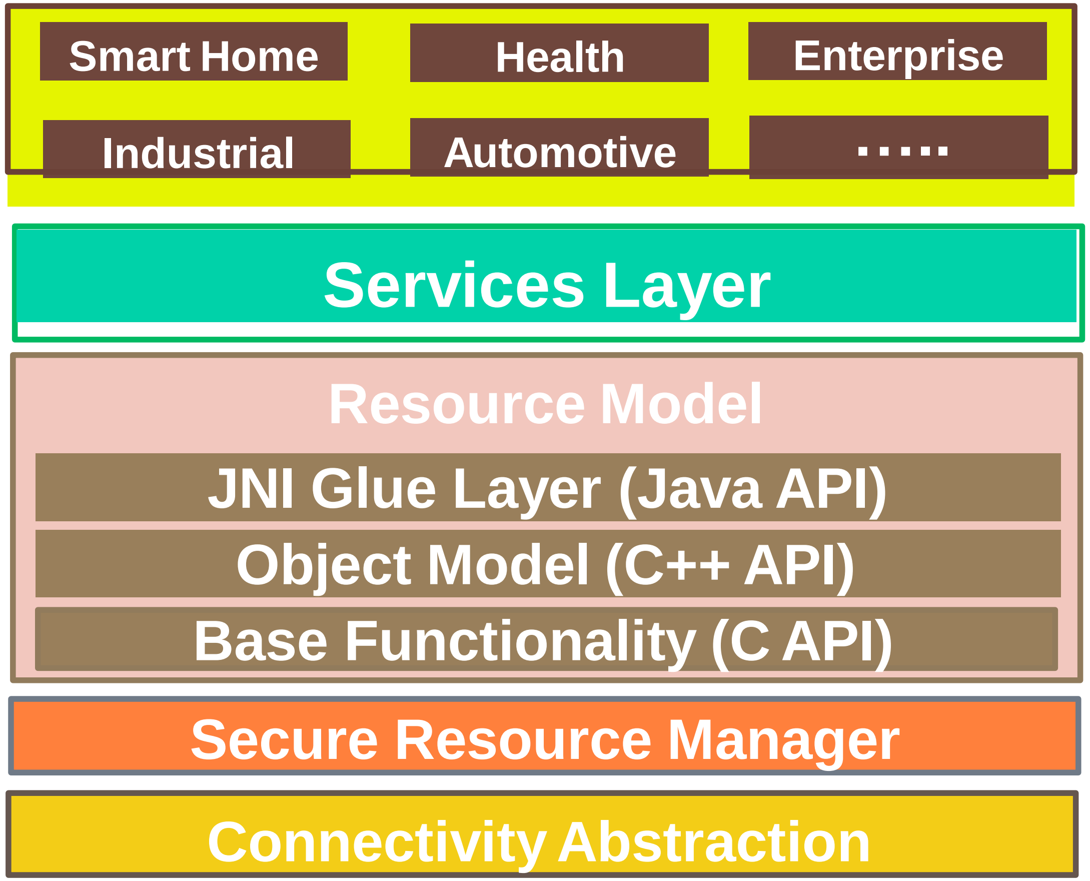
Regulate Temperature

Tablet

Why is it useful?

- Cross-platform support
- Uniform and easy-to-use APIs
- Based on open standards
- Support for multiple connectivity types
- Extensible to support proprietary protocols

IoTivity stack architecture

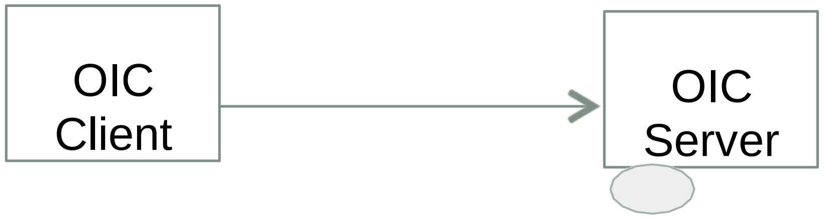


OIC protocol & connectivity types

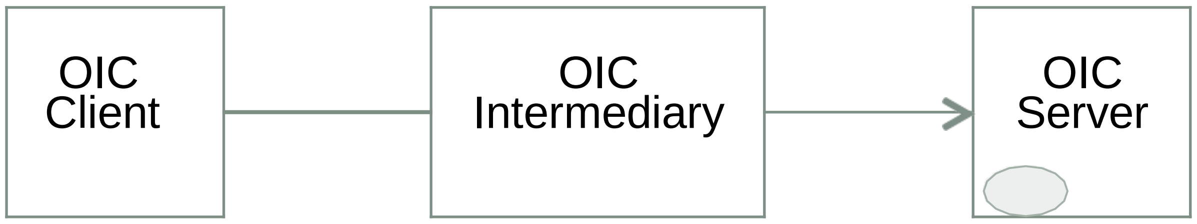
- Messaging protocol
  - Currently based on CoAP (RFC 7252)
- OIC payloads encoded using CBOR (RFC 7049)
- Adapter abstraction
  - Handle multiple connectivity types
    - Dual-stack IPv4 / IPv6
    - Bluetooth Low Energy using GATT
    - Bluetooth EDR using RFCOMM

IoTivity resource model

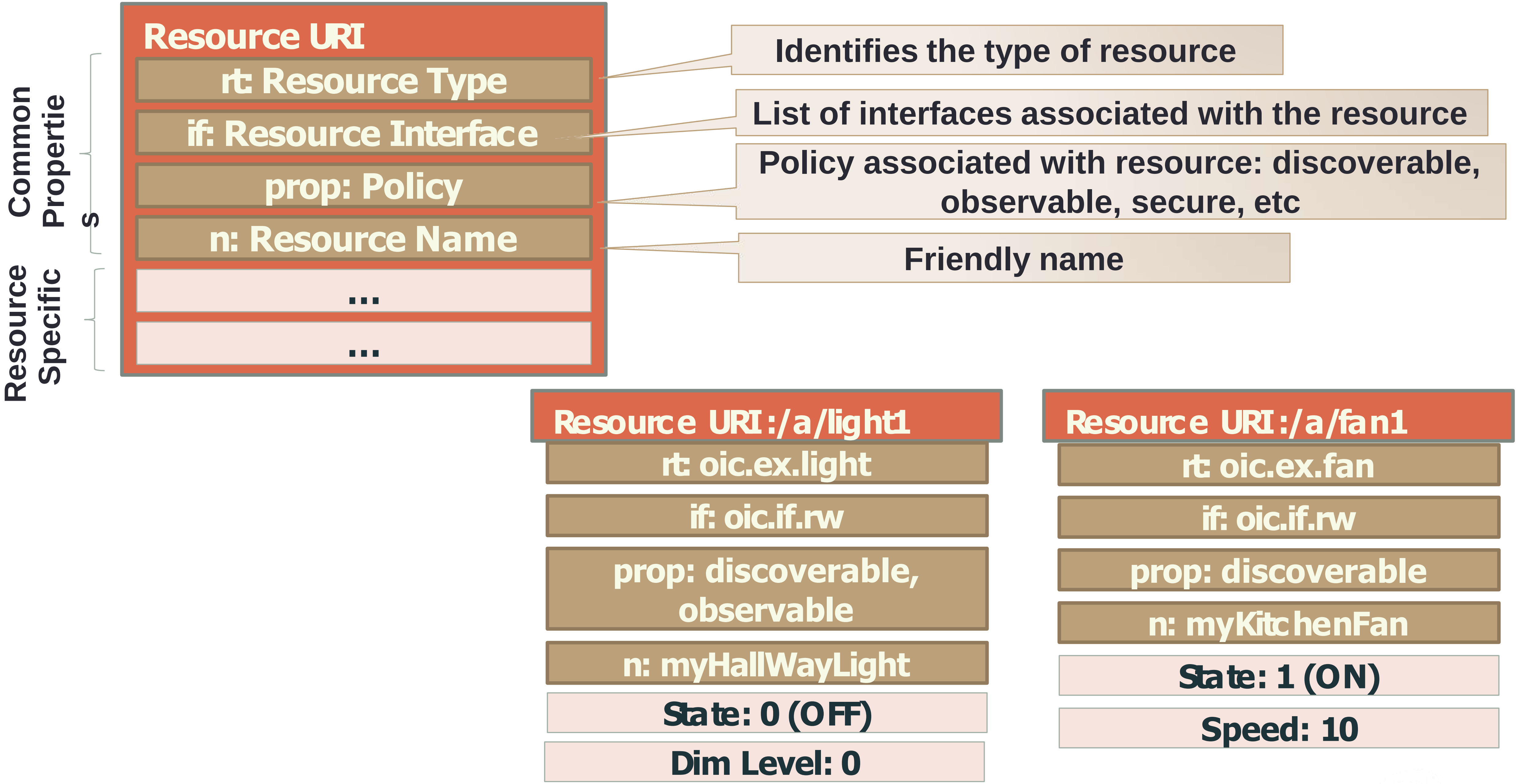
- RESTful design -> Things modeled as resources
- Server role: Exposes hosted resources
- Client role: Accesses resources on a server
- Intermediary role: Bridges messaging between client and server



Model 1



Model 2



Common Properties

Resource Specific

Resource URI

rt: Resource Type

if: Resource Interface

prop: Policy

n: Resource Name

...

...

Identifies the type of resource

List of interfaces associated with the resource

Policy associated with resource: discoverable, observable, secure, etc

Friendly name

Resource URI:/a/light1

rt: oic.ex.light

if: oic.if.rw

prop: discoverable, observable

n: myHallWayLight

State: 0 (OFF)

Dim Level: 0

Resource URI:/a/fan1

rt: oic.ex.fan

if: oic.if.rw

prop: discoverable

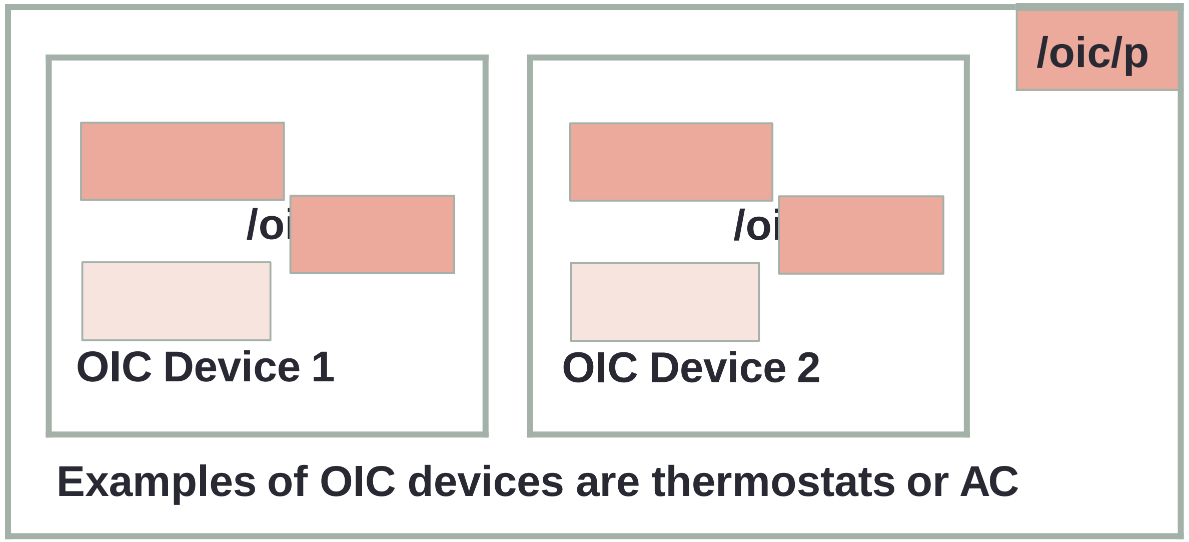
n: myKitchenFan

State: 1 (ON)

Speed: 10

“Well-Known” resources

Functionality	Fixed URI
Discovery	/oic/res
Device	/oic/d
Platform	/oic/p
Presence	/oic/ad
Security	…

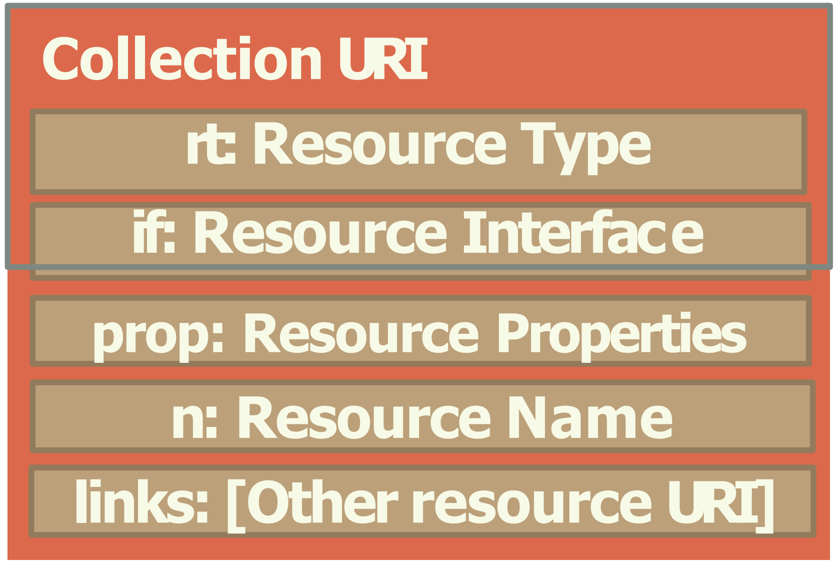


Examples of OIC devices are thermostats or AC

- Resources are associated with “Entity Handlers”
- Execute OIC methods on resources

Resource collections

- Links to other resources (RFC 5988)
- Express hierarchy, groups, indexes



Collection URI

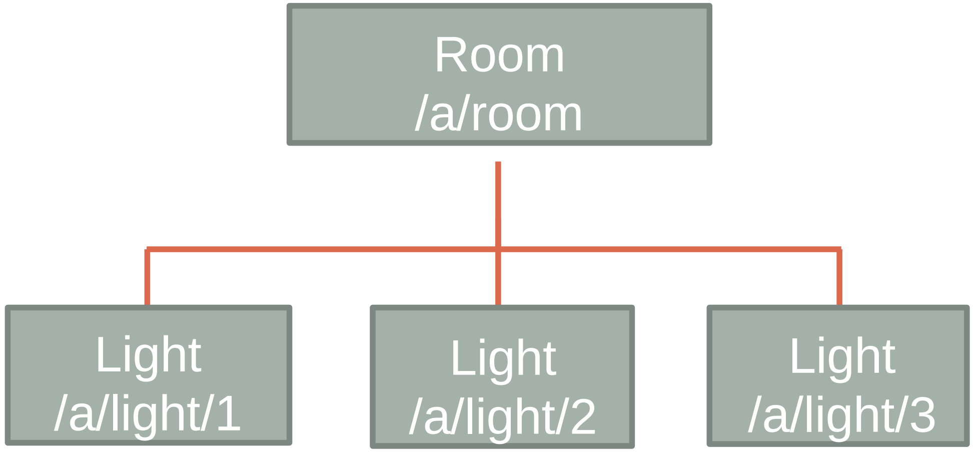
rt: Resource Type

if: Resource Interface

prop: Resource Properties

n: Resource Name

links: [Other resource URI]



Room
/a/room

Light
/a/light/1

Light
/a/light/2

Light
/a/light/3

IoTivity request-response flow

Client

Application

Resource Model

CoAP over UDP

L2 Connectivity + IP

URI: /a/light; rt = ‘oic.ex.light’,
if = ‘oic.ex.rw’,
prop = discoverable,
observable

Server

Application EH

Resource Model

CoAP over UDP

L2 Connectivity + IP

Resource discovery

Client
192.168.1.2

Multicast GET coap://224.0.1.187:5683/oic/res

Unicast response

[URI: /a/light; rt = ‘oic.ex.light’, if = ‘oic.ex.rw’, prop = discoverable, observable]

Server
192.168.1.1

Application

Resource Model

CoAP over UDP

L2 Connectivity + IP

Application EH

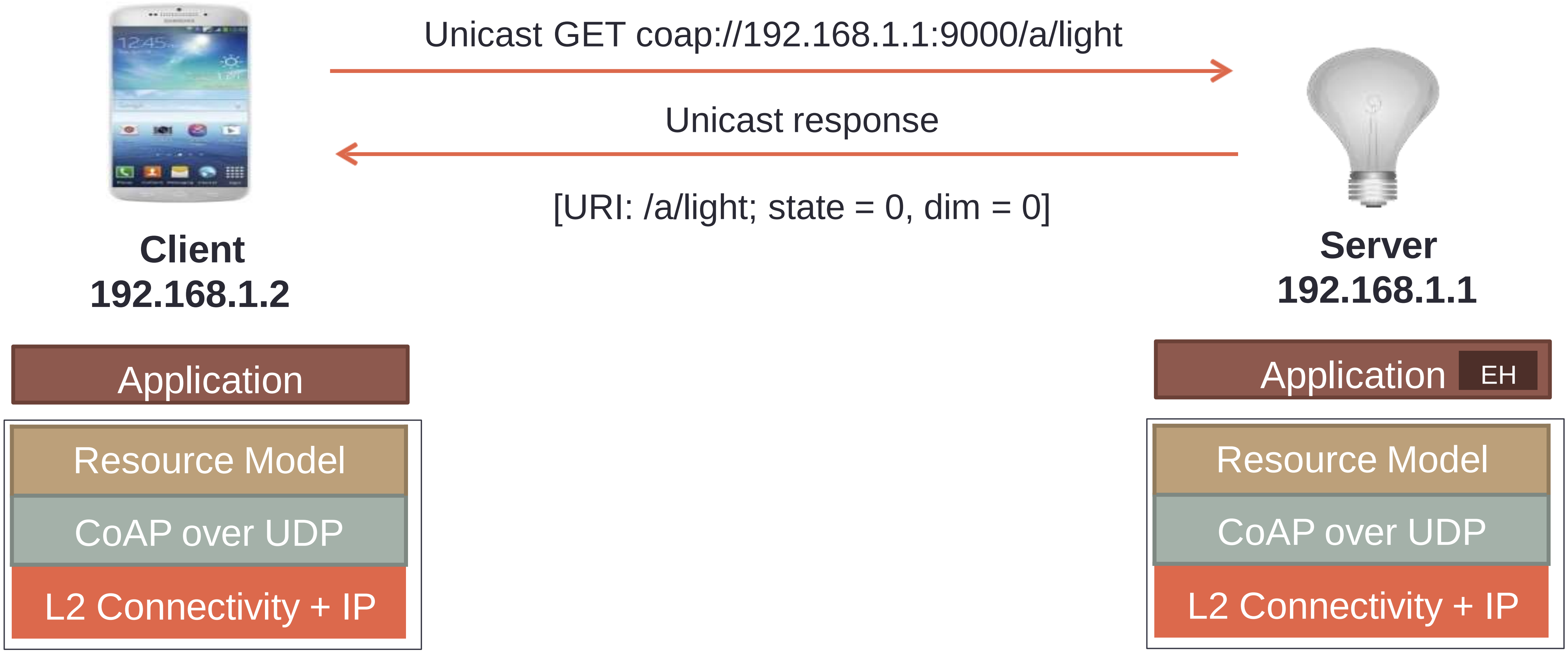
Resource Model

CoAP over UDP

L2 Connectivity + IP

IPv4	**224.0.1.187: 5683**
IPv6	**FF0X::FD: 5683**

GET operation



Unicast GET coap://192.168.1.1:9000/a/light

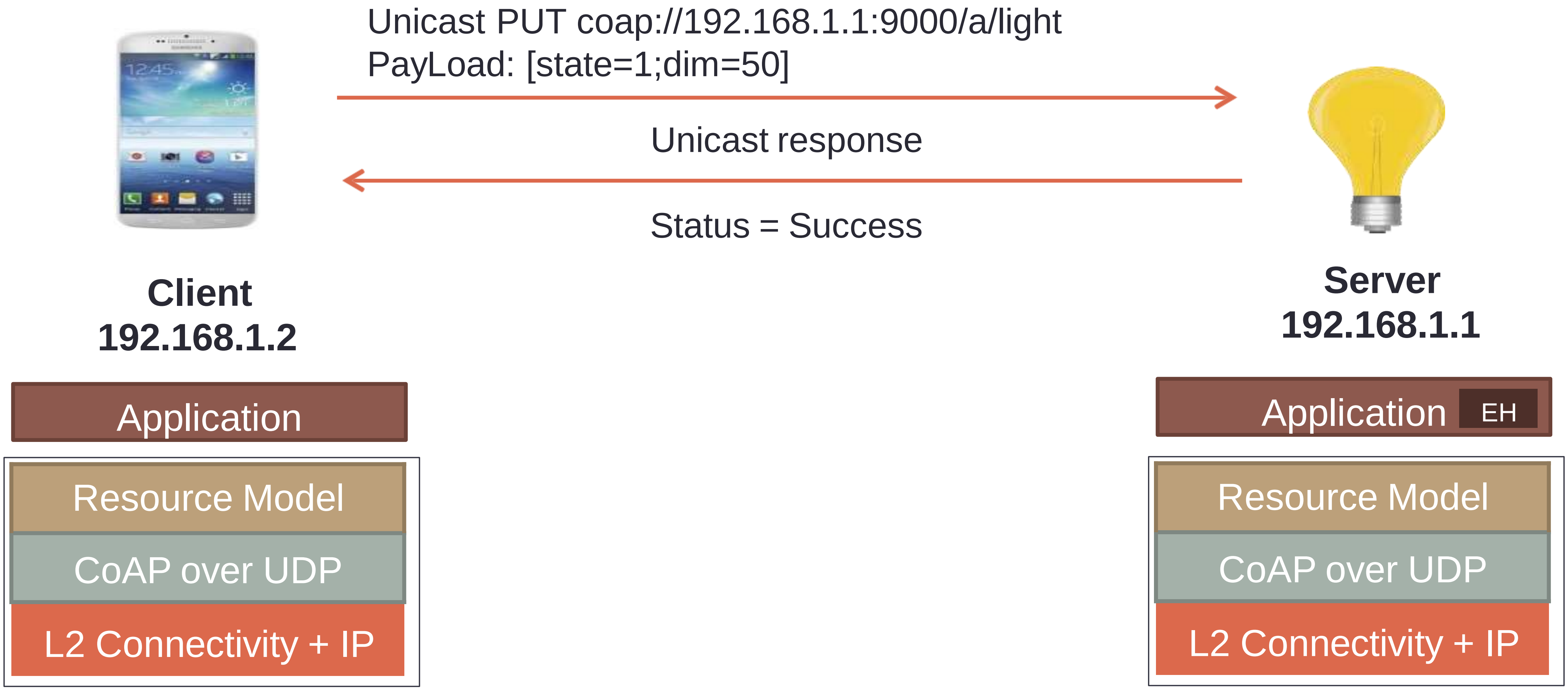
Unicast response

[URI: /a/light; state = 0, dim = 0]

Client
192.168.1.2

Server
192.168.1.1

PUT operation



Unicast PUT coap://192.168.1.1:9000/a/light
PayLoad: [state=1;dim=50]

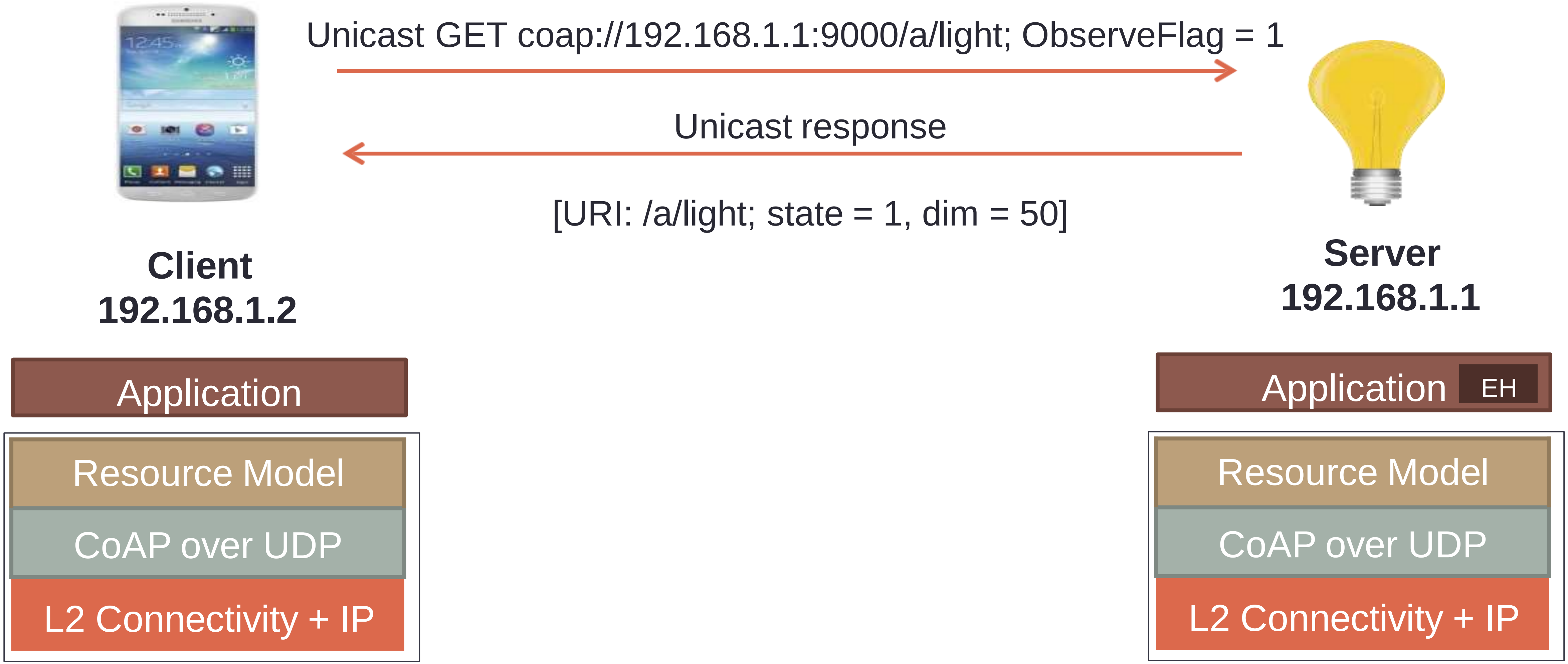
Unicast response

Status = Success

Client
192.168.1.2

Server
192.168.1.1

OBSERVE operation



Unicast GET coap://192.168.1.1:9000/a/light; ObserveFlag = 1

Unicast response

[URI: /a/light; state = 1, dim = 50]

Client
192.168.1.2

Server
192.168.1.1

Application

Resource Model

CoAP over UDP

L2 Connectivity + IP

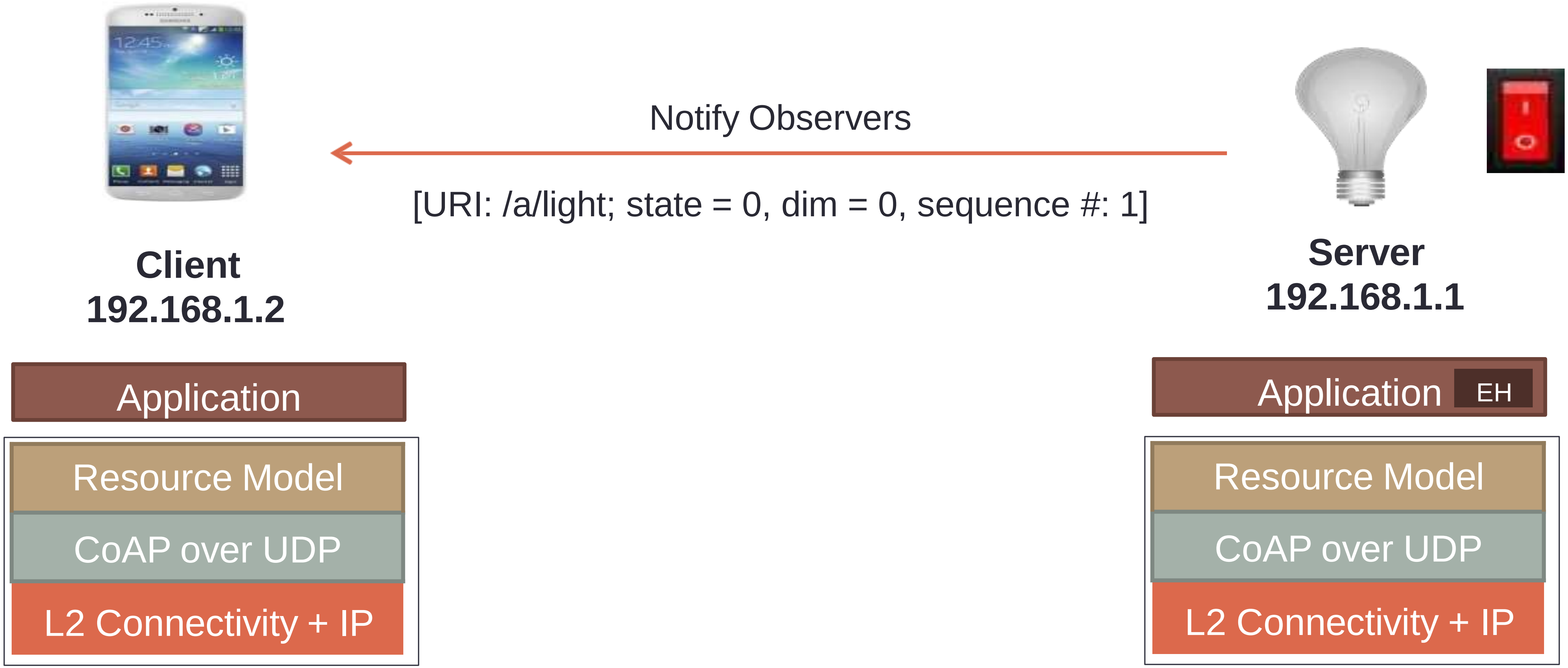
Application EH

Resource Model

CoAP over UDP

L2 Connectivity + IP

OBSERVE notification



Notify Observers

[URI: /a/light; state = 0, dim = 0, sequence #: 1]

Client
192.168.1.2

Server
192.168.1.1

Application

Resource Model

CoAP over UDP

L2 Connectivity + IP

Application EH

Resource Model

CoAP over UDP

L2 Connectivity + IP

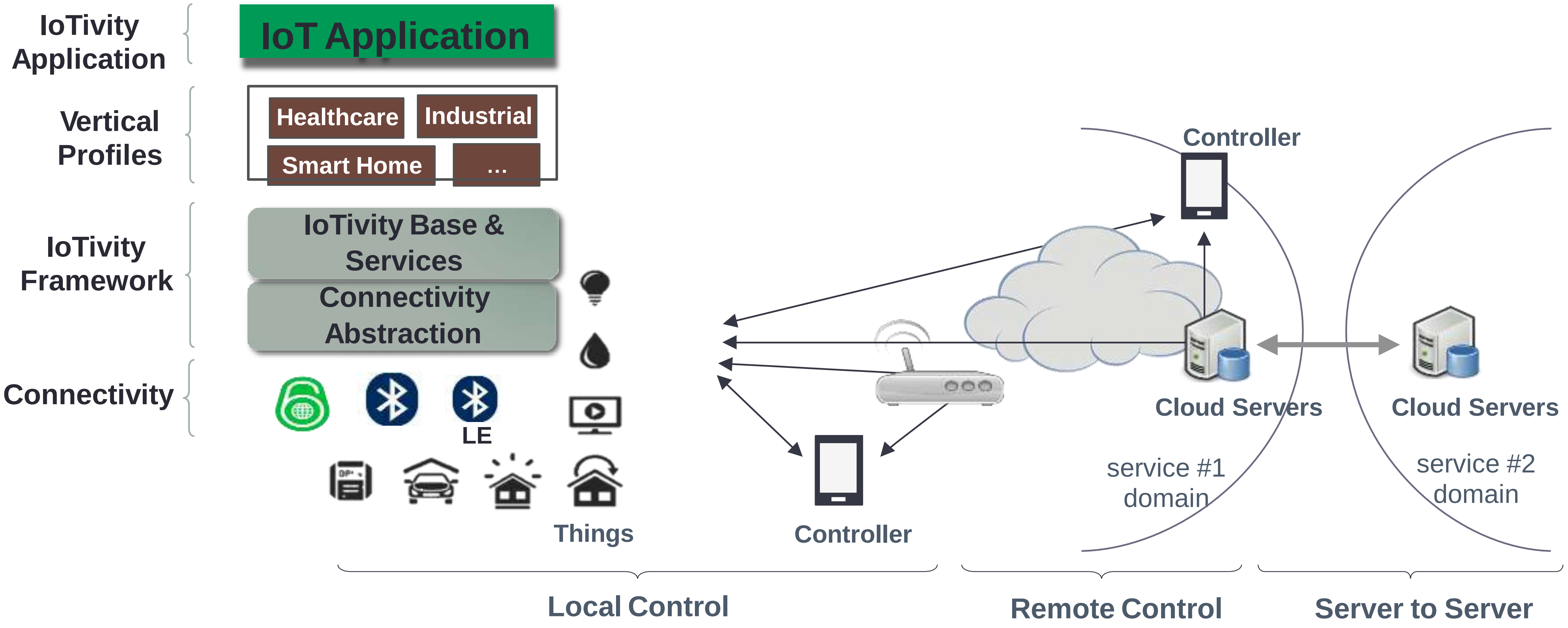
PRESENCE: “Active discovery”

- Servers can advertise themselves to clients
- Clients can request unicast or multicast notifications
  - Server coming online
  - Server going off-line
  - Changes to resources
- Clients may indicate interest in specific resource types

Notable IoTivity features

- Discovery
- Messaging and data model
- Message switching
- Remote access
- Services
  - Protocol plug-ins
  - Group management
- Security

Role in the IoT ecosystem



IoTivity Application

IoT Application

Vertical Profiles

Healthcare

Industrial

Smart Home

…

IoTivity Framework

IoTivity Base & Services

Connectivity Abstraction

Connectivity

LE

Things

Controller

Controller

Cloud Servers

service #1 domain

Cloud Servers

service #2 domain

Local Control

Remote Control

Server to Server

Cross-platform support

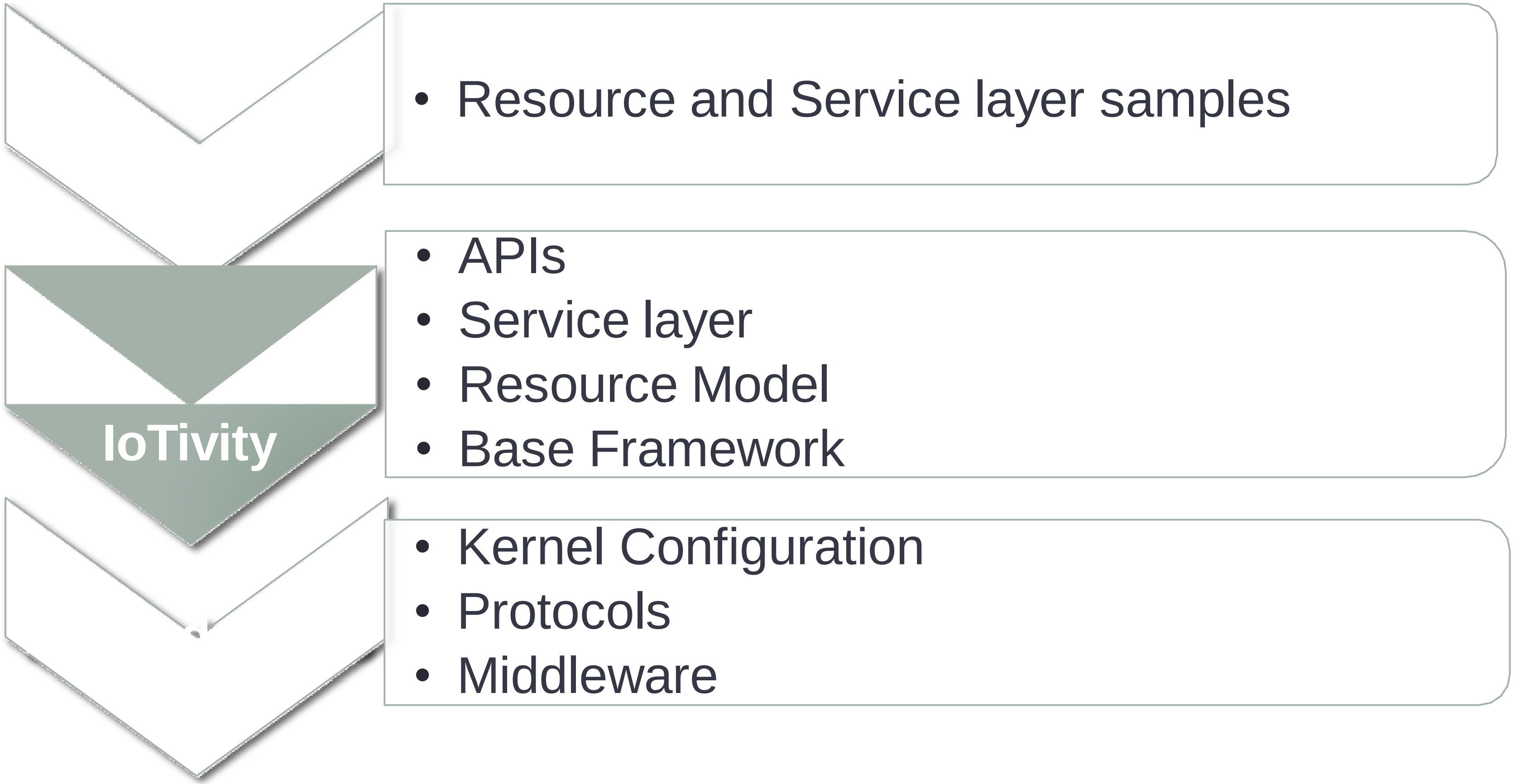
- Linux (Ubuntu 12.04)
- Arduino: Due, ATMega 2560
- Android
- Tizen

Embedded support: Yocto Project

- http://www.yoctoproject.org/
  - Hosted at the Linux Foundation
  - Create customized OS images for embedded targets
  - Ready-to-use BSPs for multiple platforms
  - Supports major CPU architectures
- Layers and recipes

meta-oic software layer for Yocto

- git://git.yoctoproject.org/meta-oic



- Resource and Service layer samples

- APIs
- Service layer
- Resource Model
- Base Framework

- Kernel Configuration
- Protocols
- Middleware

Constrained peripherals

- Storage and memory constraints
- Lightweight IoTivity server stack
  - Base framework, resource model, messaging
- Work in progress…

END

Thanks for listening