UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Lecture 2 - Introduction to the IoT Software Abstractions and Communication Means

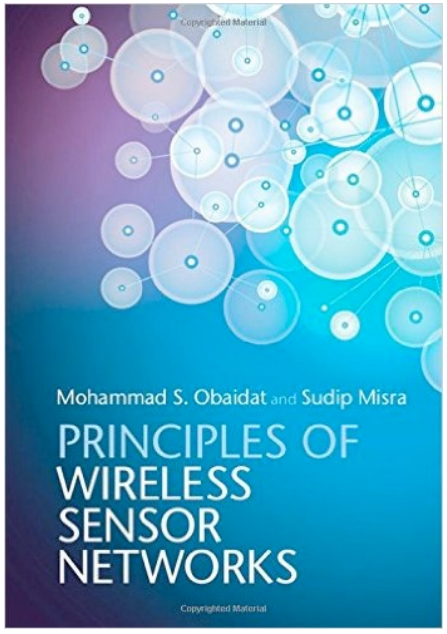
Prof. Esposito Christian

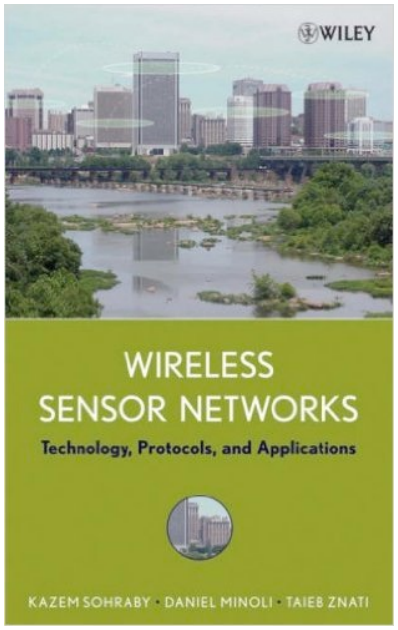
::.. Summary

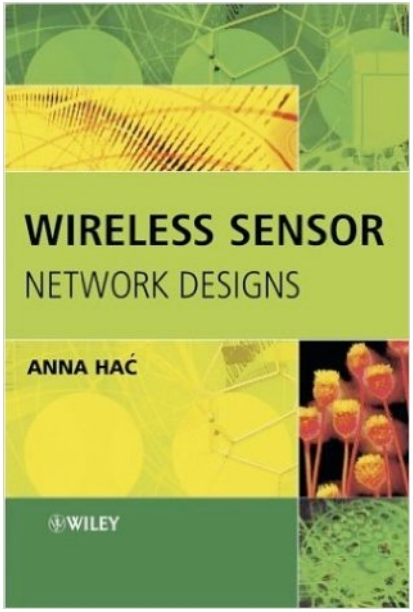
- Software abstractions for sensors:
  - Operating systems for sensors;
  - Programming languages.
- Communication means for sensors:
  - Communication technologies and protocols;
  - Cloud support.

::.. References

- Mohammad S. Obaidat e Sudip Misra, “Principles of Wireless Sensor Networks”, Cambridge University Press – 2015;
- Kazem Sohraby e Daniel Minoli, “Wireless Sensor Networks: Technology, Protocols, and Applications”, Wiley – 2007;
- Anna Hac, “Wireless Sensor Network Designs”, Wiley – 2003.







::.. Key Lectures (1/2)

- K. Romer, e F. Mattern, “The Design Space of Wireless Sensor Networks”, IEEE Wireless Communications, vol. 11, no. 6, pp. 54-60, December 2004.
- J. Yick, B. Mukherjee, e D. Ghosal, “Wireless sensor network survey”, Computer Networks, vol. 52, no. 12 , pp. 2292-2330, Agosto 2008.
- M.-M. Wang, J.-N. Cao, L. Li, e S. K. Dasi, “Middleware for Wireless Sensor Networks: A Survey”, Journal of Computer Science and Technology, vol. 23, no. 3, pp. 305-326, May 2008.
- V.C. Gungor, e G.P. Hancke, “Industrial Wireless Sensor Networks: Challenges, Design Principles and Technical Approaches”, IEEE Transactions on Industrial Electronics, vol. 56, no. 10, pp. 4258-4265, October 2009.
- L. Mottola, e G. P. Picco, “Programming wireless sensor networks: Fundamental concepts and state of the art”, ACM Computing Surveys (CSUR), vol. 43, no. 3, art. 19, Aprille 2011.

::.. Key Lectures (2/2)

- Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A survey." Computer Networks, vol. 54, no. 15, pp. 2787-2805, October 2010.
- Da Xu, Li, Wu He, and Shancang Li. "Internet of Things in industries: A survey." IEEE Transactions on industrial informatics, vol. 10, no. 4, pp. 2233-2243, November 2014.
- Al-Fuqaha, Ala, et al. "Internet of Things: A survey on enabling technologies, protocols, and applications», IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347-2376, Fourthquarter 2015.
- Dizdarević, Jasenka, et al. "A survey of communication protocols for Internet of Things and related challenges of Fog and Cloud computing integration", ACM Computing Surveys (CSUR), vol. 51, no. 6, art. 116, 2019, January 2019.

Sensor Software Abstraction

::.. Software Platforms (1/6)

To avoid the designer having to deal with low-level aspects when programming sensor nodes, a series of operating systems for sensor networks have been proposed, which essentially present themselves as a library linked to the application code to produce a binary code to execute.

The operating system usually supports a specific programming language, which is typically a dialect of C or specific to sensor networks. Also a set of low-level communication primitives are provided.

The programming abstraction and communication primitives represent the lowest abstraction for programmers, similar to the C programming and sockets are the basic abstractions provided by operating systems in traditional computing environments.

::.. Software Platforms (2/6)

Traditional operating systems are software systems to manage computing resources, peripheral control devices and provide software abstraction for applications.

- They are mainly used for managing processes, memory, CPU time, file systems and devices.
- They are usually made in a modular manner and at levels, including a low level called a kernel and a high level of system libraries.

Traditional operating systems are not optimal for sensor networks because these networks have limited resources and different data-centric applications, in addition to a variable topology. To this end, WSNs need new solutions that take into account their peculiarities and requirements.

::.. Software Platforms (3/6)

There are various problems to be addressed for a specific operating system for sensor nodes in a sensor networks:

- Process management and scheduling - traditional operating systems allocate processes in a separate memory space, causing high management times for the context switch between processes and high consumption.
- Memory management - Memory is allocated exclusively to a process or task in traditional systems, this is not possible in sensor nodes that have poor storage capacity.
- Kernel model - Traditionally, a kernel is composed of various interdependent modules and system calls for services that run in supervisor mode, with a multi-threaded model. Operating systems for sensor nodes have an event or state machine model.

::.. Software Platforms (4/6)

- Application Program Interface (API) - Sensor nodes require modular and general APIs for their applications, but also allow easy access to the underlying hardware.
- Code update and reprogramming - As the behavior of sensor nodes and their algorithms need various adjustments to their functionality or to optimize consumption, the operating system must be reprogrammed and updated.
- Since the sensors do not have an external disk, the operating system cannot have a file system.

::.. Software Platforms (5/6)

An operating system for sensor nodes must offer the following features and functionalities:

- It must be compact and small, since the sensor nodes have memories of only tens or hundreds of kilobytes.
- It must provide real-time support, as the monitoring applications have real-time constraints, especially when the actuators are involved.
- It must offer an efficient management of resources in order to manage the microprocessor time and limited memory allocating them carefully to guarantee fairness.
- It must support the distribution of the code in a reliable and efficient manner so as to allow updates after the hot deployment.

::.. Software Platforms (6/6)

- It must allow the management of electrical consumption so as to extend the life cycle of the sensor node and improve its performance.
- It must provide a series of communication primitives between nodes on the network that is efficient and flexible, given the heterogeneity of the nodes, implementing protocols at the transport, network and MAC level.
- It must present a generic API for application software or even middleware for sensors, also allowing direct access to hardware features so as to optimize system performance.

Some OS for sensor nodes have been proposed over the years, the most known one is TinyOS.

::.. Concurrency Models (1/4)

An operating system must manage the processor time so that each application can have an honest slice. In a sensor node, multiple activities can occur concurrently:

- The readings of the monitoring devices are collected, processed by the microcontroller and possibly stored in a secondary storage;
- The sensor data is transmitted by means of a communication device, or communications from neighboring nodes are received and passed to other nodes;
- Timed events can occur and must be managed.

The operating system must handle this concurrency in such a way that resources are used efficiently and the whole process is easy for the developer to understand.

::.. Concurrency Models (2/4)

The execution model of an operating system determines how competing applications are run.

- Sensor nodes operate in a highly concurrent environment with strong resource constraints, so much sensor operating systems adopted an event execution model, and the main execution unit is an event manager.
- An event handler is invoked in response to an external event (eg, reception of a message or a phenomenon detected by the monitoring equipment) or internal (eg, the expiration of a timeout) that occurs.
- The alternative model to the event model is the multi-threaded execution model, commonly used in traditional operating systems. In this model, applications are defined as more than one concurrent running thread. Threads stop responding when waiting for external events.

::.. Concurrency Models (3/4)

The event model has more efficiency in the use of memory:

- Since each manager returns directly to the operating system, it is not necessary to keep track of it after its conclusion. In the multi-threaded model, each thread must keep track of its status. The used memory is little, but cannot be freed because the thread may need it in the future.
- In the event model only one stack is necessary, while in the multi-threaded one, each thread has its own stack. When an event occurs, the operating system determines the specific manager, and executes it. However, when the manager has to wait for more events before its conclusion, it is necessary to divide it into several sub-managers and define a real state machine. This complicates the programming.

::.. Concurrency Models (4/4)

The compromise between efficiency in memory use and programming complexity has led to hybrid models:

- The model called protothread provides a sequential control flow, like the multi-threaded model, but without the overhead of managing multiple stacks. In fact, it offers primitives that allow programs to execute a blocking wait without a separate stack for each protothread.
- Another approach is to execute multiple threads above an event-based kernel, which allows the developer to choose which execution model to employ, depending on the needs of the program.

::.. Memory Allocation (1/2)

On sensor nodes, the memory size is constrained both on the physical (given by the number of transistors) and practical (to keep consumption and manufacturing costs low). Traditionally the memory is divided into two parts:

- A static part that contains the running program;
- A dynamic part that contains runtime variables, buffers, input and output data and the stack.

The static portion is usually kept in a read-only memory (ROM), while the dynamic portion is in a random access memory (RAM). Given the physical characteristics of existing architectures for microcontrollers, RAM has a higher energy consumption than ROM and requires a larger chip area. For these factors, RAM is typically smaller than ROM.

::.. Memory Allocation (2/2)

The main problem that the memory allocation mechanisms have to face is fragmentation, i.e., when unused memory is scattered among various non-contiguous regions of memory.

- When fragmentation occurs, allocations may fail even when the total amount of unused memory is sufficient.
- To avoid this phenomenon, operating systems for sensor nodes avoid dynamic allocation, but all the necessary memory is statically allocated.
- In the case of dynamic allocation, the designer must pre-allocate static buffers, which will be used at runtime. This allows developers to understand the memory requirements of an application prior to its execution and to reduce the risk of fragmentation situations that could be fatal.

::.. Energy Consumption (1/2)

The sensor nodes have a battery, with a fixed amount of energy. Energy consumption determines the life cycle, making energy a critical factor in a sensor network.

Component	Consumption (mW)
microcontroller, sleeping	0.163
microcontroller, active	5.40
flash read	12.3
flash write	45.3
radio transmit	58.5
radio listen	65.4

Energy management is essential for an operating system of sensor nodes. To reduce consumption, turn off non-used components. Consumption management for communication is handled differently by a radio duty cycling mechanism.

::.. Energy Consumption (2/2)

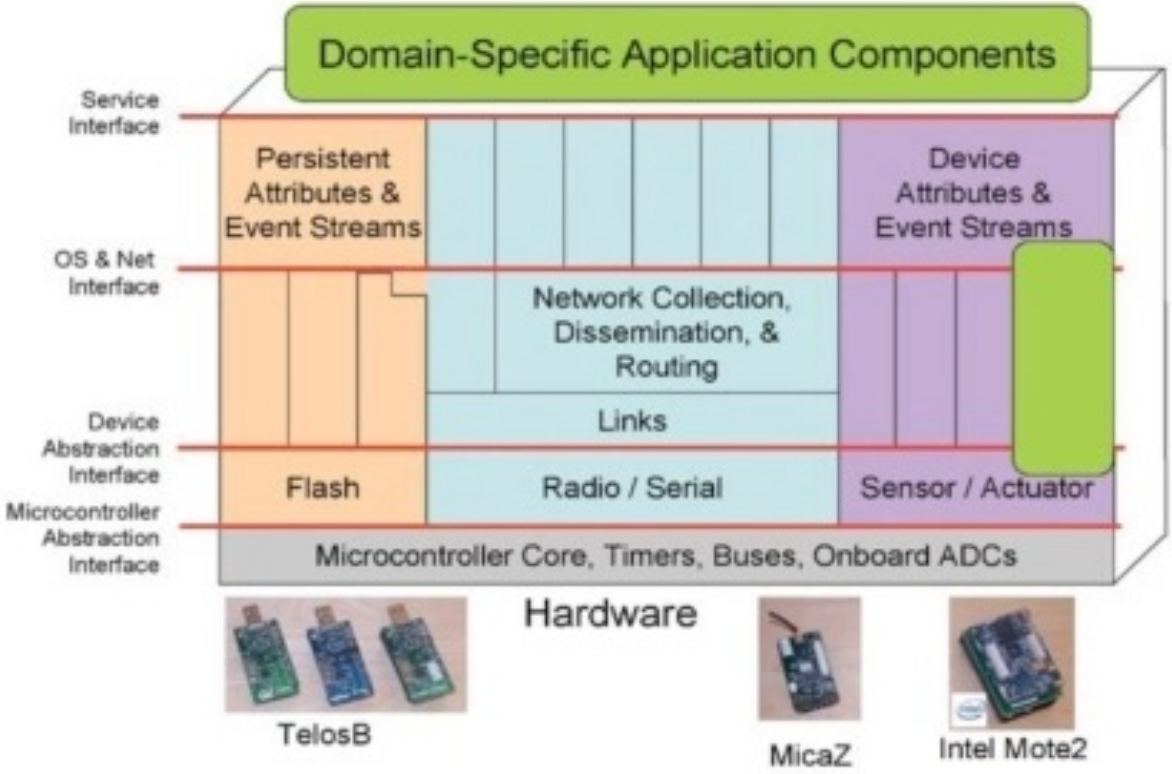
The operating system also keeps track of consumption, adopted both a hardware and software approach to this end.

- One approach is to divide the total consumption by attributing it to hardware devices or logical activities.
- Another solution is to monitor the components of the operating system and attribute consumption to them, which also include the amount spent on the hardware used.

Consumption tracking information can be a factor taken into account for task scheduling based on application requirements, or consumption management is left to the application by only providing activity monitoring information.

::.. TinyOS (1/4)

TinyOS is an open-source, component-based and application-specific operating system developed specifically for sensor networks. It can support competing programs with low energy consumption requirements and has a low memory occupation (about 400 bytes).



TinyOS has a library of components that include network protocols, distributed services, monitoring device drivers and data acquisition tools.

::.. TinyOS (2/4)

TinyOS represents an example of non-monolithic architecture, using a component model that is chosen and assembled according to the needs of the applications.

A component is an independent computation entity that has one or more interfaces. The components have three computational abstractions: commands, events and tasks.

- The first two are communication mechanisms between components (one synchronous and one asynchronous, respectively), while the latter is used to express competition within a component.

TinyOS provides a single shared stack, without any separation between kernel space and user space. TinyOS adopts a model of execution and concurrency with events.

::.. TinyOS (3/4)

TinyOS only supports non-preemptive scheduling, so tasks must have completion semantics, but they are not atomic with respect to possible interruptions, and commands or events they invoke. The waiting time of a task depends on its arrival time and on the task queue waiting to take over the microprocessor. For this, the TinyOS scheduler does not provide good real-time requirements if several tasks compete for resources.

Since version 2.1, a hybrid model has been introduced with multi-threading support above the event model, allowing applications to create TOS Threads. A message passing mechanism allows communication between TOS Threads and the kernel.

::.. TinyOS (4/4)

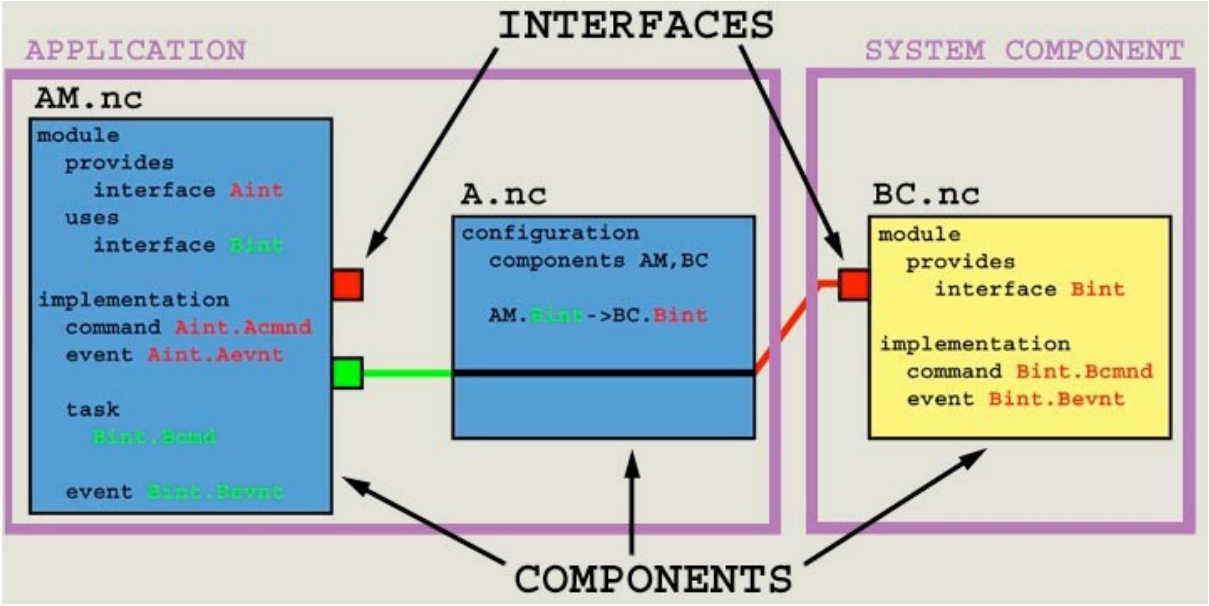
When an application needs to execute a system call, it does not make a synchronous call, but posts a message to the kernel requesting the execution of the desired system call. Next, the kernel takes precedence from the current thread and executes the system call. This mechanism ensures that only the kernel executes TinyOS code directly.

Concurrency and thread synchronization is provided by condition variables and mutexes. The synchronization primitives are implemented with the help of special hardware instructions.

Usually there are no memory protection mechanisms, but TinyOS offers Deputy abstraction to allow type and memory safety for applications. This is ensured by a mix of compilation and execution time checks, which if checked activate appropriate remediation codes.

::.. nesC (1/4)

NesC is an event-based programming language, based on components, for dedicated systems, and was born as a "C dialect". NesC was designed to develop TinyOS, therefore, it is also used to write application programs for sensor nodes using TinyOS. nesC has the characteristic of being modular, allowing the user to link pieces of code together.

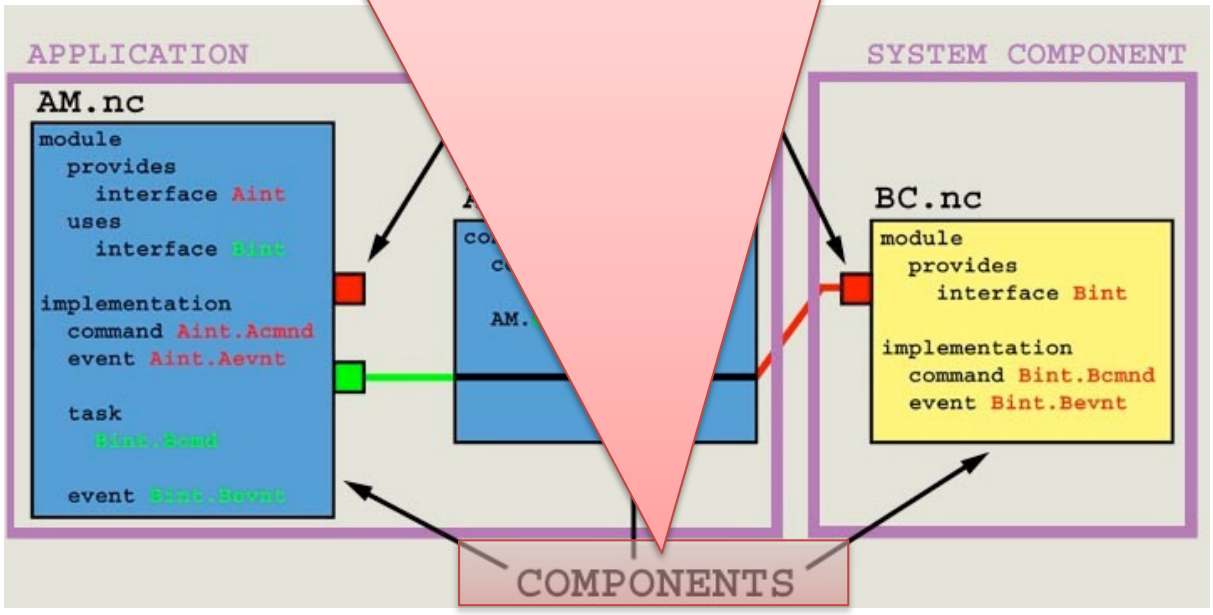


A complete application can be realized by assembling different software components (modules). Separate the implementation from the composition with interfaces.

::.. nesC (1/4)

NesC is an event-based programming language, based on components, for dedicated systems, and was born as a "C dialect". NesC was designed to develop TinyOS, therefore, it is also used to write application programs for sensor nodes using

The main constituents of an application are the components, which are connected to each other through their interfaces.

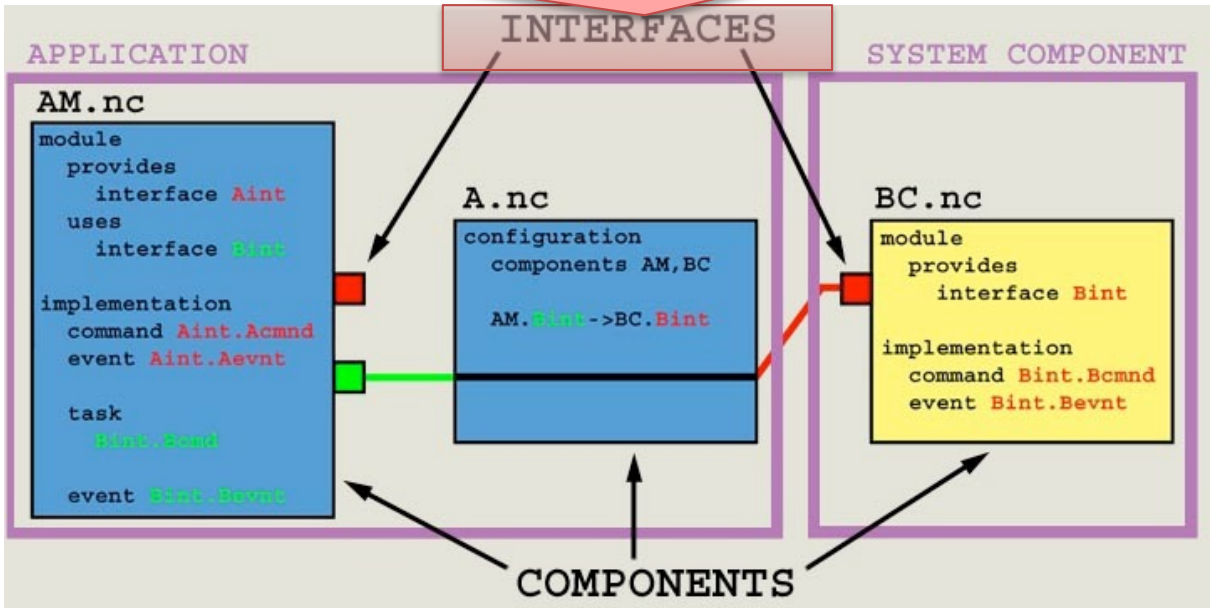


A complete application can be realized by assembling different software components (modules). Separate the implementation from the composition with interfaces.

::.. nesC (1/4)

NesC is an event-based programming language, based on components, for dedicated systems, and was born as a "C dialect". NesC was designed to develop TinyOS, therefore, it is

A component can provide the implementation of an interface or request it. Interfaces are the only way to send and receive data in the form of commands and events.

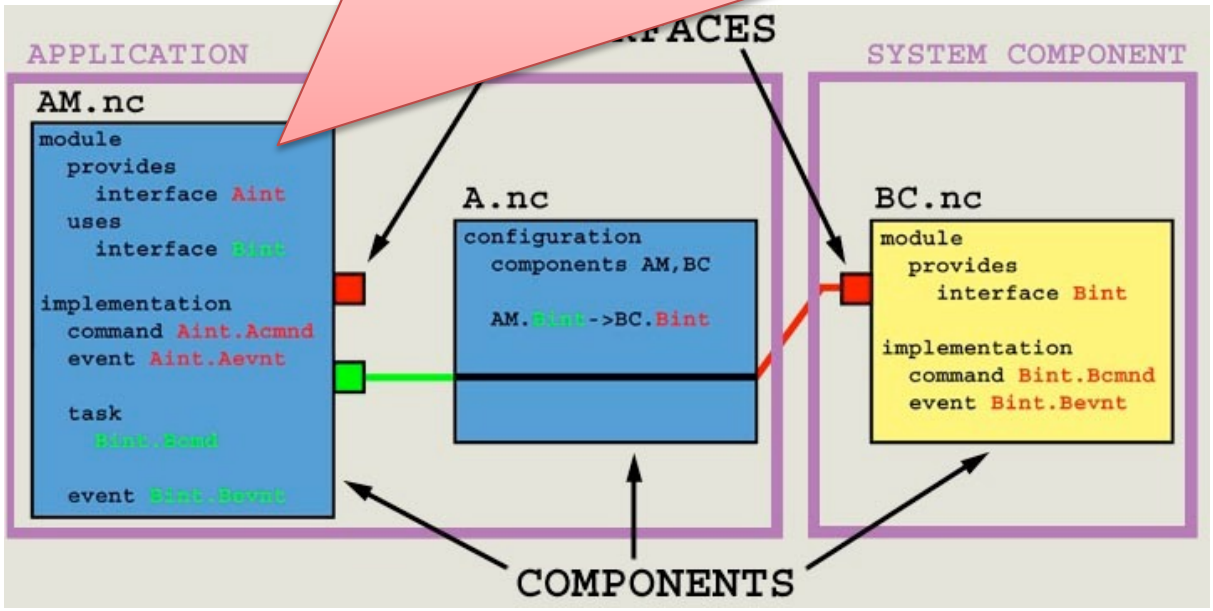


A complete application can be realized by assembling different software components (modules). Separate the implementation from the composition with interfaces.

::.. nesC (1/4)

NesC is an event-based programming language, based on components, for dedicated systems, and was born as a "C dialect". NesC was designed to develop TinyOS, therefore, it is

The specification of a component is divided into two sections: the module indicates the offered and requested interfaces; while, the implementation defines the data exchanged and the tasks.

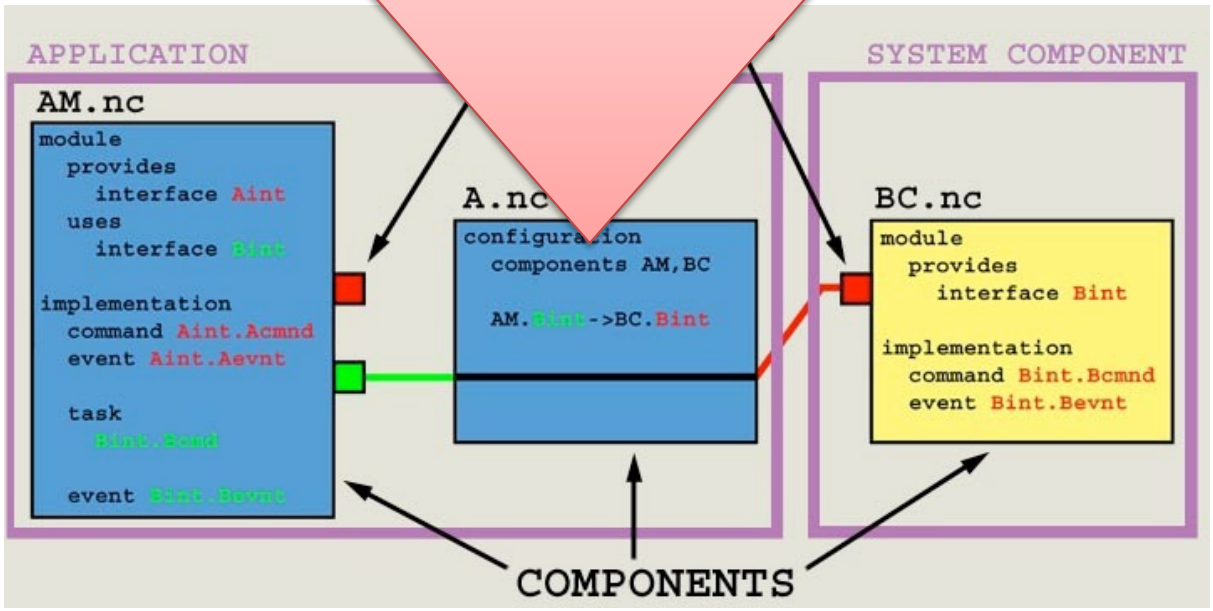


A complete application can be realized by assembling different software components (modules). Separate the implementation from the composition with interfaces.

::.. nesC (1/4)

NesC is an event-based programming language, based on components, for dedicated systems, and was born as a "C dialect". NesC was designed to develop TinyOS, therefore, it is

The components are connected to each other by means of a configuration. The definitions of components and configurations are contained in file extension nc.



A complete application can be realized by assembling different software components (modules). Separate the implementation from the composition with interfaces.

::.. nesC (2/4)

Let us consider a concrete case of an application, which causes the red LED of the sensor node to flash with a frequency of 1Hz, composed of

- a component called BlickM.nc;
- a Blink.nc configuration.

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

::.. nesC (2/4)

Let us consider a concrete case of an application, which causes the red LED of the sensor node to flash with a frequency of 1Hz, composed of

It defines the configuration, which in turn can supply or implement junctions to be able to implement complex applications by assembling them from sub-applications.

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

::.. nesC (2/4)

Let us consider a concrete case of an application, which causes the red LED of the sensor node to flash with a frequency of 1Hz, composed of

The implementation part defines which components the application consists of, followed by how these components connect together.

configuratio
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}

::.. nesC (2/4)

Let us consider a concrete case of an application, which causes the red

Main is a system component that marks the beginning of the application, like the main function in C. Main.StdControl.init() is the first command executed by TinyOS followed by Main.StdControl.start(). Thus, an application in TinyOS must have a Main component in its configuration. StdControl is a common interface used to initialize and launch TinyOS components.

```
}
implementation
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

::.. nesC (2/4)

Let us consider a concrete case of an application, which causes the red

Main is a system component that marks the beginning of the application, like the main function in C. Main.StdControl.init() is the first command executed by TinyOS followed by Main.StdControl.start(). Thus, an application in TinyOS must have a Main component in its configuration. StdControl is a common interface used to initialize and launch TinyOS components.

```
}
implementation
  components Main, BlinkM, SingleTim

  Main.StdControl -> BlinkM.StdContro
  Main.StdControl -> SingleTimer.StdC
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

StdControl.nc

```
interface StdControl {
  command result_t init();
  command result_t start();
  command result_t stop();
}
```

::.. nesC (2/4)

nesC defines a series of interfaces for the main components that can compose an application. Timer abstracts the functionality of a timer. Given an interface, there are various different implementations between which the designer can draw, as in the highlighted case, where a simple timer component is used and one that abstracts the red led of the node.

```
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

::.. nesC (3/4)

Now let's see the component called BlickM:

```
module BlinkM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
}
// Continued below...
```

::.. nesC (3/4)

Now let's see the component called BlickM:

```
module BlinkM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
}
// Continued below...
```

You specify the required interfaces, and those for which an implementation is provided. On the first, of which an implementation is provided, commands and events are received which trigger the execution of the application logic, while towards the second commands are sent or events are received.

::.. nesC (4/4)

```
implementation {
  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }
  command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 1000) ;
  }
  command result_t StdControl.stop() {
    return call Timer.stop();
  }
  event result_t Timer.fired()
  {
    call Leds.redToggle();
    return SUCCESS;
  }
}
```

```
interface Timer {
  command result_t start(char type,
uint32_t interval);
  command result_t stop();
  event result_t fired();
}
```

::.. nesC (4/4)

```
implementation {
  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }
  comm  result_t StdControl.start() {
    return  tart(TIMER_REPEAT, 1000) ;
  }
  {
    call Leds.redToggle();
    return SUCCESS;
  }
}
```

```
uint32_t interval);
  command result_t stop();
  event result_t fired();
}
```

Each command of the interface to implement implies a specific function that is executed when the command is received. In the first one the LED is initialized, passing a command on the LEDs interface and returning a success value.

::.. nesC (4/4)

```
implementation {
  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }
  command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 1000) ;
  }
  command result_t StdControl.stop() {
    return call Timer.stop();
  }
  event result_t Timer.fired()
  {
    ...
  }
```

```
interface Timer {
  command result_t start(char type,
uint32_t interval);
  ...
}
```

In the second one, the timer is started, indicating the periodic timout time. The third interrupts the timer.

::.. nesC (4/4)

```
implementation {
  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }
    retu          ner.stop();
  }
}
```

Finally, a timer interface event handler is implemented, which changes the status of the LED from on to off and vice versa.

```
event result_t Timer.fired()
{
  call Leds.redToggle();
  return SUCCESS;
}
```

```
interface Timer {
  command result_t start(char type,
uint32_t interval);
  command result_t stop();
  event result_t fired();
}
```

::.. Embedded Linux (1/10)

Although appearing in multiple shapes and forms, an embedded system typically performs a dedicated function, is resource-constrained and comprises a processing engine:

- Small-sized systems, comprising a low-powered CPU with at least 2 MB of read-only memory (ROM) and 4 MB of random-access memory (RAM).
- Medium-sized systems, with around 32 MB of ROM, 64 MB of RAM and a medium-powered CPU.
- Large-sized embedded systems, with powerful CPUs and a larger memory footprint.

As flash memory prices have decreased over time, and low-power small-footprint System on a Chip (SoC) hardware is increasingly becoming the developer's premier processor choice. Virtually every embedded system is capable of running Linux.

::.. Embedded Linux (2/10)

An embedded Linux system simply denotes an embedded system running on the Linux kernel. There is nothing as an “embedded version” of the Linux kernel.

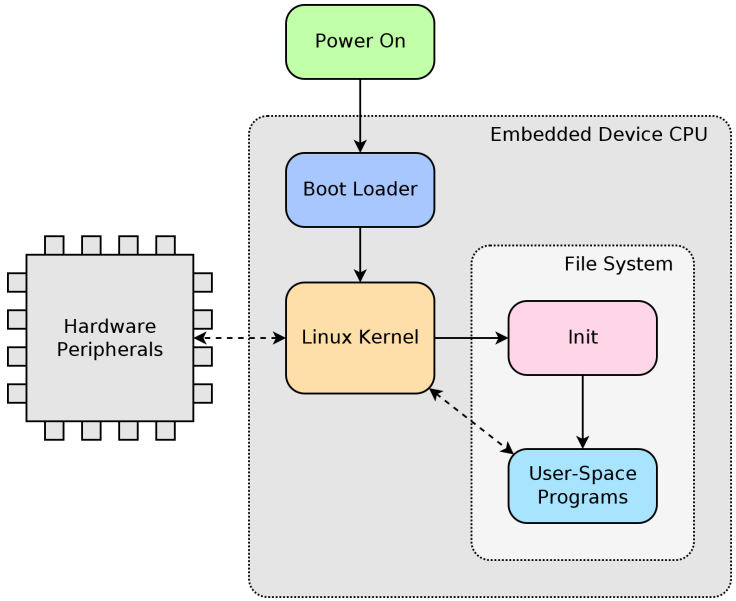
- Linux runs on 32 and 64-bit ARM, x86, MIPS, and PowerPC architectures. Processors below 32-bit aren’t capable of running Linux, ruling out traditional embedded systems.

A tailored kernel for an embedded system is never required, but an official release can be used instead. However, it is often the case of a kernel configured to support a custom hardware, as the kernel build configuration found in an embedded device usually varies from the one in a server or workstation.

::.. Embedded Linux (3/10)

“distribution” is an umbrella term usually comprising software packages, services and a development framework on top of the OS itself. Ubuntu Core is the flavour of Ubuntu for embedded devices, where all the necessary modules are kept.

- Linux’s prevalence in embedded systems is its modularity. With several software packages coming together to form a Linux OS stack, developers can customize it for any purpose.

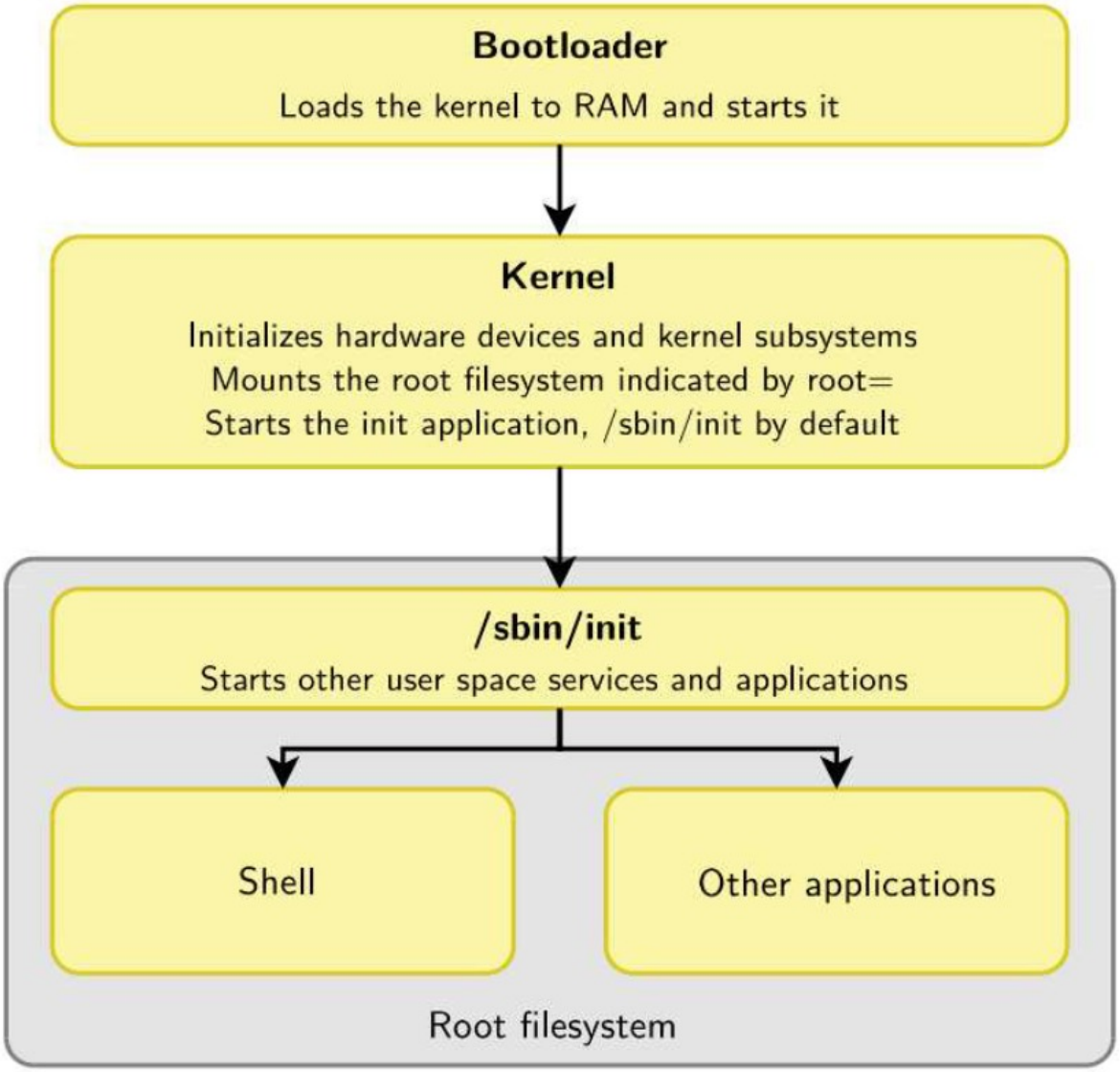


Most embedded Linux systems is made of the boot loader, the kernel and the file system. They are built separately, usually on a build host using cross-compiling, and usually loaded onto the embedded device all at once.

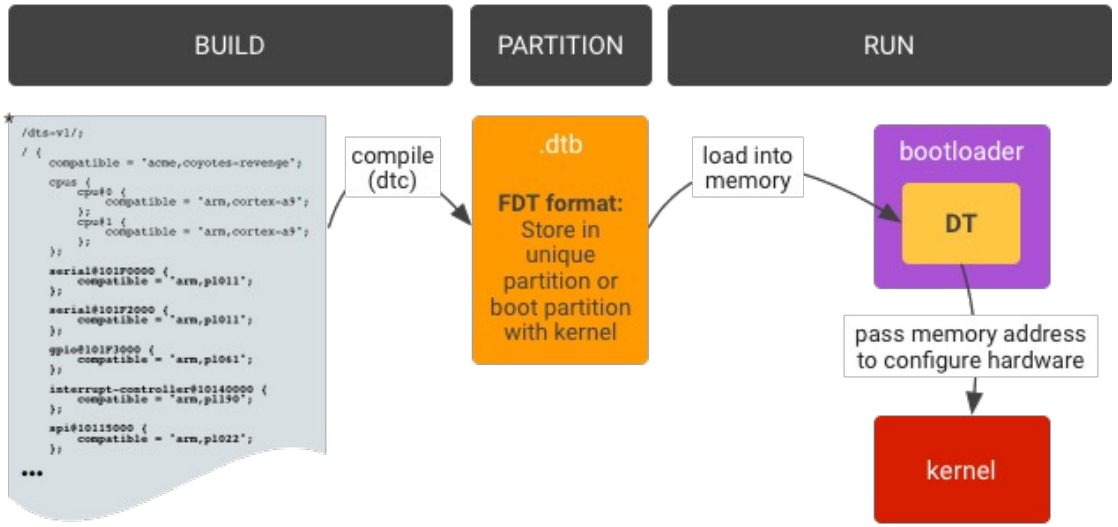
::.. Embedded Linux (4/10)

- The boot loader is small and uses minimal resources and its main job of the boot loader is to find and run the Linux kernel. It should bring up some peripherals, such as the system memory, or include network support and a command line interface for loading new software images from over the network.
- Once started, the Linux kernel runs continuously as the main process managing everything on the embedded device: it should have full support for all peripherals, run user-space applications, and provide device drivers that allow user-space applications to use the hardware through generalized APIs.
- The file system contains all of the user-space programs and data. A Linux filesystem is usually divided into a standard set of folders that each have a standard purpose. It usually has an init system which is the first user-space program to run on startup and brings up all the rest of the user-space programs.

::.. Embedded Linux (5/10)



::.. Embedded Linux (6/10)

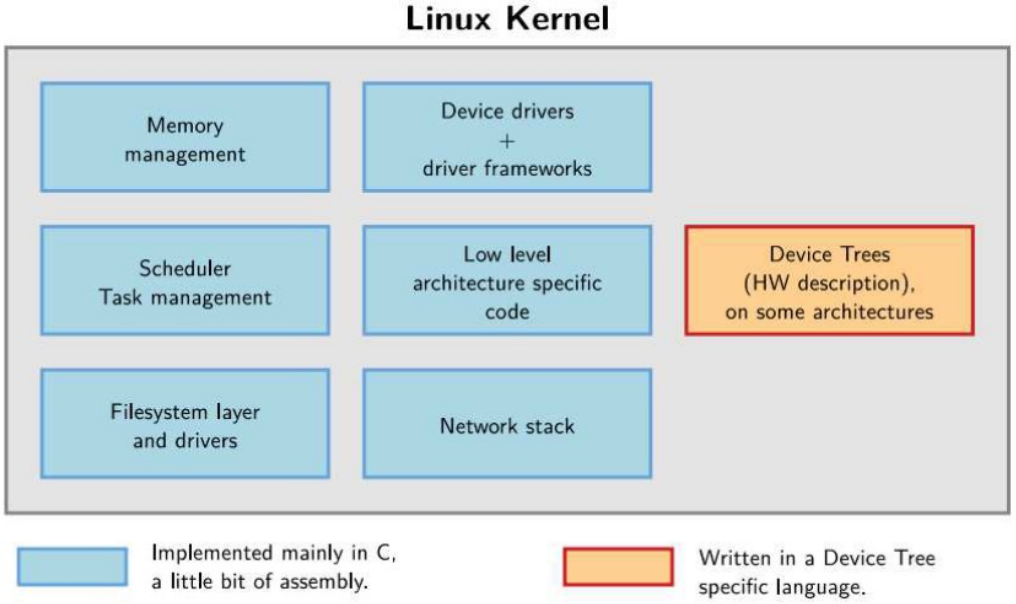


*http://elinux.org/Device_Tree_Usage#Devices

The bootloader also takes care of loading the device tree into RAM, which consists of a board configuration file. This file is a database containing information about the board's hardware components and is used to forward hardware information from the bootloader to the kernel.

As soon as the operating system takes control of the board, the bootloader is overwritten and ceases to exist. The only way to run it again is to reset the board itself.

::.. Embedded Linux (7/10)



The Linux kernel manages all hardware resources: CPU, memory and I/O. It provides a set of APIs that abstract these resources allowing user applications and libraries to easily use them. It also manages concurrent access to the various resources by the various applications.

It consists of various modules that are usually written in C language and assembly.

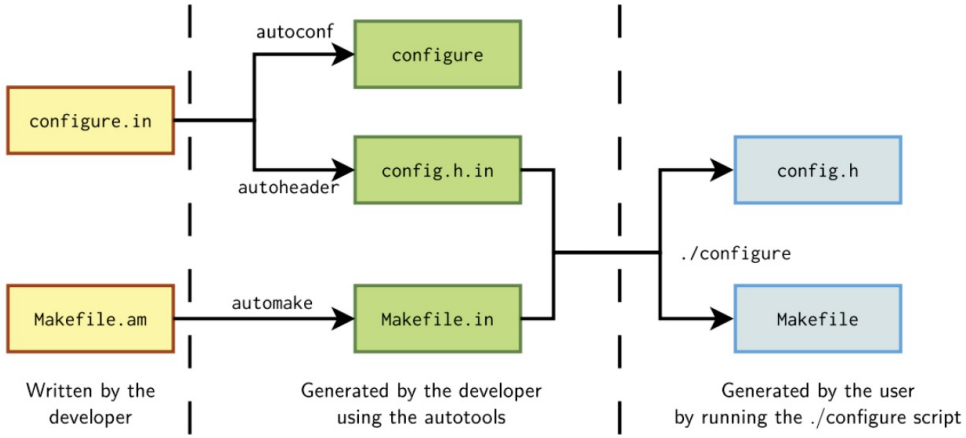
::.. Embedded Linux (8/10)

From the perspective of development for embedded Linux platforms, starting from simple applications up to entire distributions, root filesystems, kernels and bootloaders, build systems can be used; which integrate various software components within a root filesystem by downloading, extracting, configuring, compiling and installing these components. All this can be done:

- manually: it is the most complex solution as all the software must be configured, compiled, and installed in the correct order; therefore, any dependencies must be resolved manually. Libraries and applications also need to be repurposed for cross-compilation;
- with the help of automatic build systems, i.e. tools that automate the root filesystem and kernel build process. Some also allow you to build across the entire cross-compilation toolchain. This is the most advantageous solution as the dependencies are resolved automatically;
- using existing distributions or filesystems.

::.. Embedded Linux (9/10)

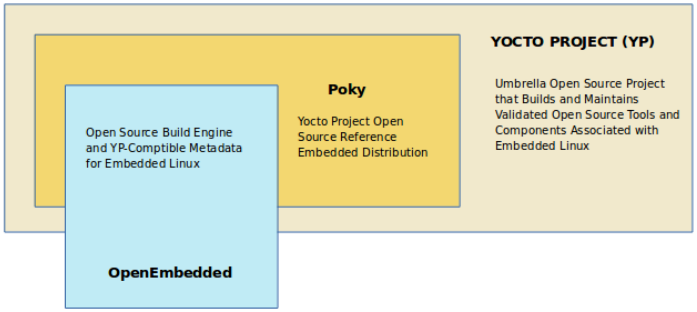
A project to be cross-compiled must be correctly configured, compiled and finally installed within the root filesystem. A tool that allows you to do this work is the GNU Make tool, whose main task is to generate executables from source code that can be written in various programming languages. Typically, these sources use macros, functions, constants, and data structures defined in other files. All you need to use the utility is a Makefile, in which the rules that manage the executable generation process are defined. The only difficulty is writing a Makefile.



There are two tools that generate Makefiles relatively easily: autotools and cmake.

::.. Embedded Linux (10/10)

Yocto is an open source project that provides templates, tools and metadata that allow you to create custom embedded Linux systems regardless of the architecture used.



It allows to build a complete Linux system –from source– in about an hour (about 90 minutes with X), support all major embedded architectures and provides advanced kernel development tools with modular development, reuse, and easy customizations.

A developer starts with a validated collection of software (toolchain, kernel, user space) and blueprints to get started quickly and customize for developers's own needs. It support both app developers and system developers by also providing access to a great collection of app developer tools (performance, debug, power analysis, Eclipse).

Middleware

::.. Introduction (1/5)

There is a gap between network protocols and applications, which must be filled by offering appropriate abstraction functions, in order to provide the quality of service to the applications using the limited resources of the nodes and extending their life cycle.

The solution to this need is the design of a middleware layer, located below the applications and above the operating system and network protocols. The task is to hide the details and the heterogeneity of the low level and facilitate the development, deployment and maintenance of applications.

Middleware for sensor networks are subject to different constraints compared to those for traditional IT environments and present significant differences.

::.. Introduction (2/5)

The challenges to face when designing middleware for sensor networks are as follows:

- Topology control, to organize sensor nodes in an interconnected network;
- Data-based processing with optimized energy consumption;
- Application-specific integration, given that the integration of application information in network protocols improves performance and conserves energy;
- Efficient use of computing and communication resources;
- Support for real-time applications to improve the predictability of application executions .

::.. Introduction (3/5)

The main functions of a middleware for sensor networks are:

- Standardized system services for various applications;
- An environment that coordinates multiple applications;
- Mechanisms for obtaining the adaptive and efficient use of system resources, i.e., implementations of algorithms that dynamically manage the limited and variable network resources;
- Efficient balancing between multiple quality parameters, to optimize the network resources required to supply them.

Initially, the community had shown no interest in the middleware layer because the simplicity of the first applications did not require such an abstraction. With the rapid evolution and success of these networks, the complexity of the applications has increased, like the gap with the underlying levels, therefore to the advent of such middleware.

::.. Introduction (4/5)

Traditional middleware cannot be applied to sensor networks:

- Traditional middleware seeks to provide transparency abstractions to hide context information, but sensor networks applications must consider the context;
- Often the middleware in the mobile context have as concern the satisfaction of the interests of a given mobile node, while in the sensor networks' context one must think about the interest of the entire network;
- Traditional middleware has a unique addressing mechanism, while in sensor networks, addressing is often data-centric;
- The middleware for sensor networks must be light, while the traditional ones has a sophisticated software stratification.

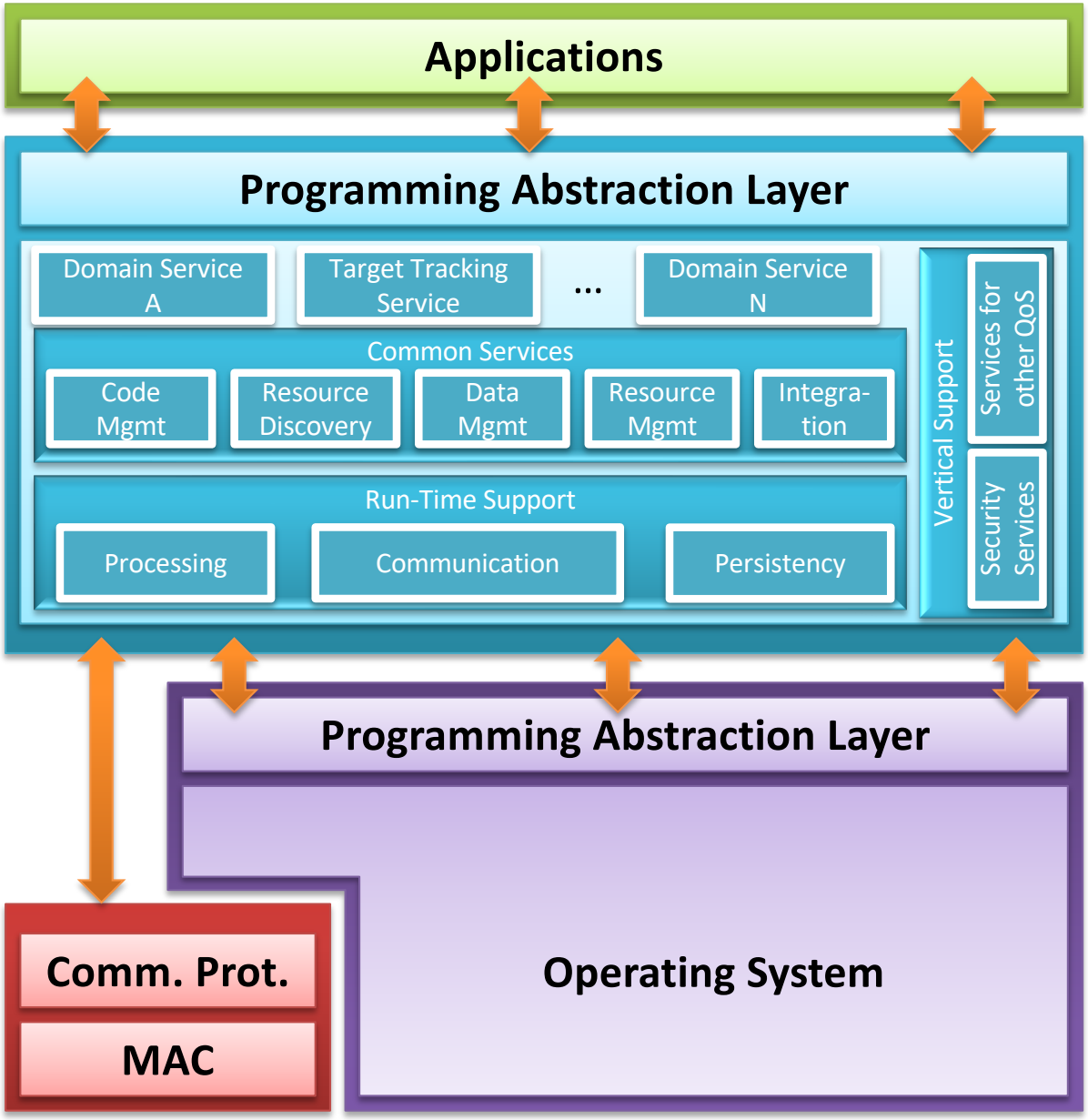
For this reason, a series of ad-hoc solutions have been proposed over the years, focusing on different aspects and / or needs.

::.. Introduction (5/5)

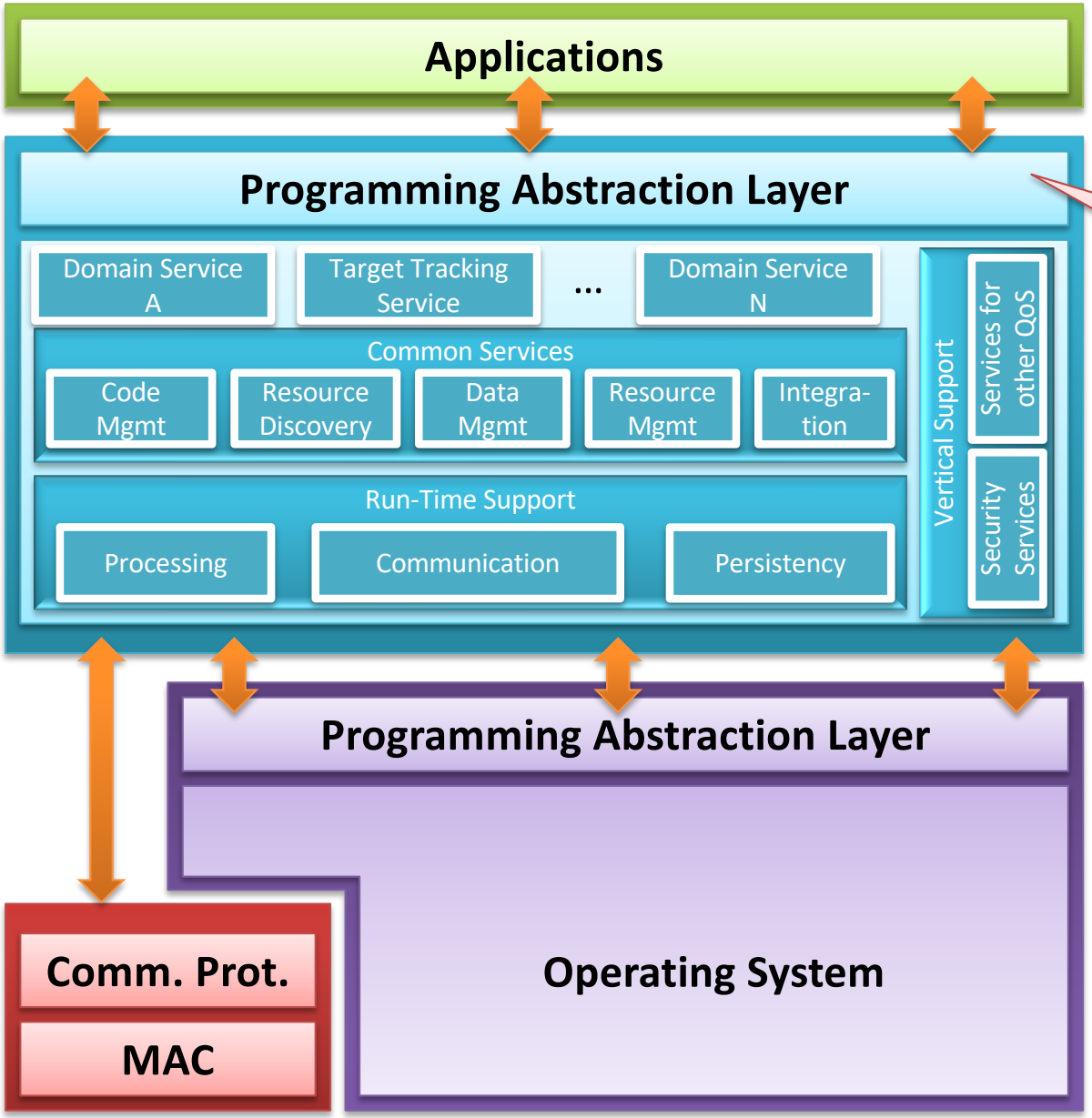
The principles that guide the realization of a middleware for sensor networks are the following:

- Use of localized algorithms, or algorithms that do not require the gathering of common knowledge at one point in the network to reach a common goal, but only use local processing based on knowledge acquired by neighbors;
- Use of adaptive precision algorithms, that is, algorithms in which the quality (fidelity) of the response (within acceptable limits) can be adapted to requirements on battery life, network bandwidth, or the number of active sensors;
- Need for data-centered mechanisms for processing and querying information, yielding the decoupling of data and sensor nodes;
- Realization of "light" services in order not to burden the available memory and processing capacity.

::.. Architecture (1/4)

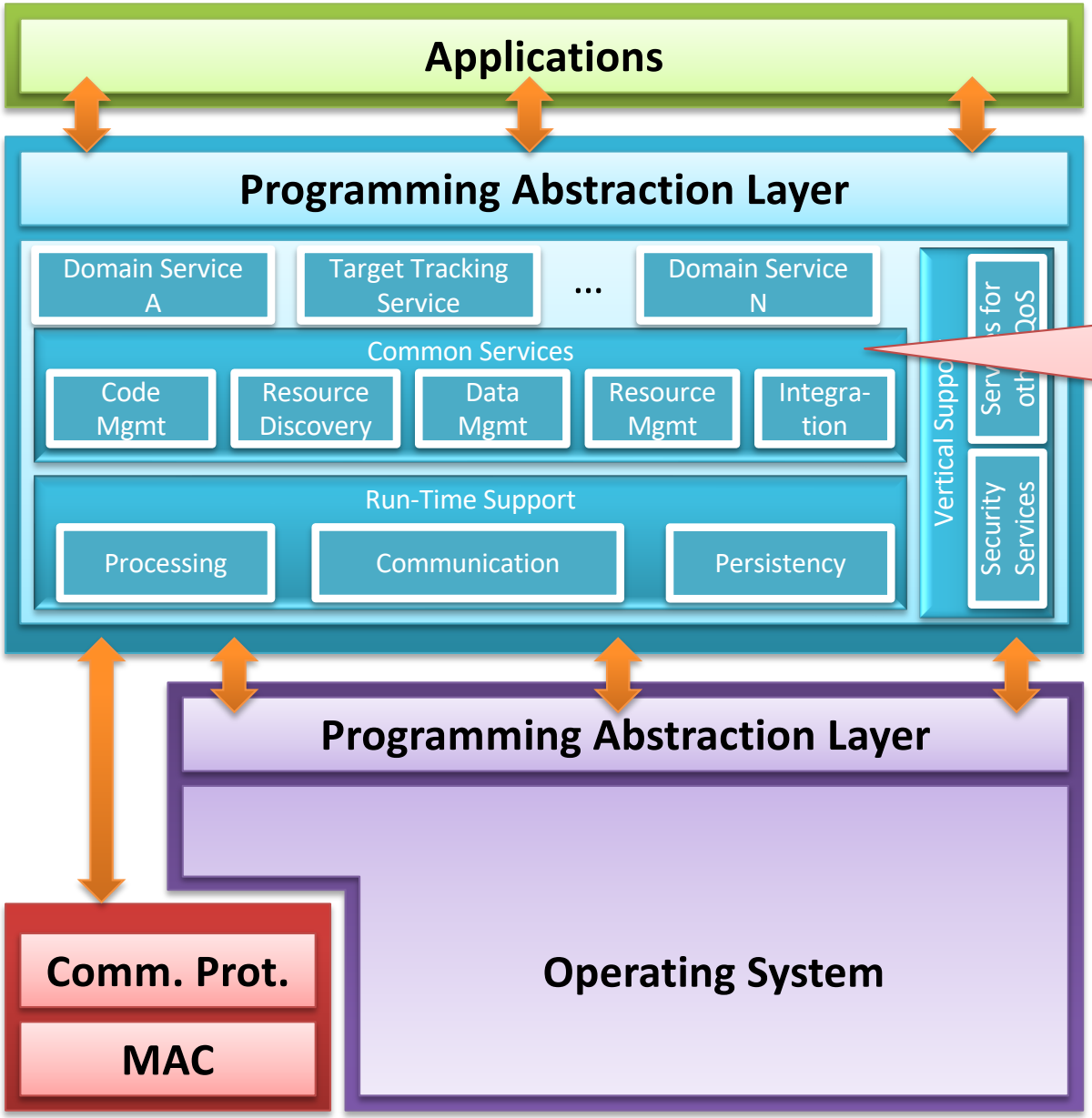


::.. Architecture (1/4)



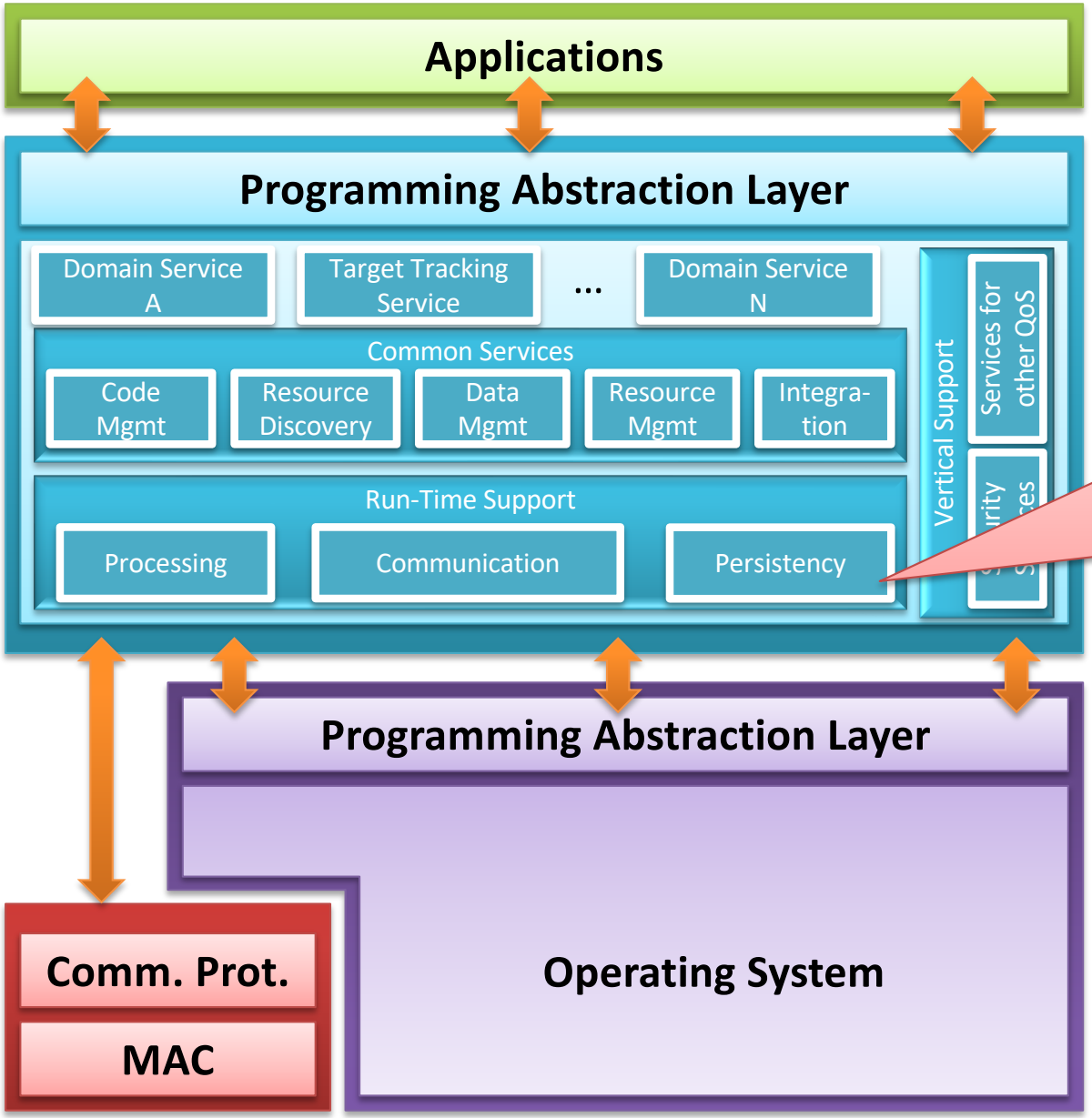
Represents the middle-ware interface to applica-tions.

::.. Architecture (1/4)



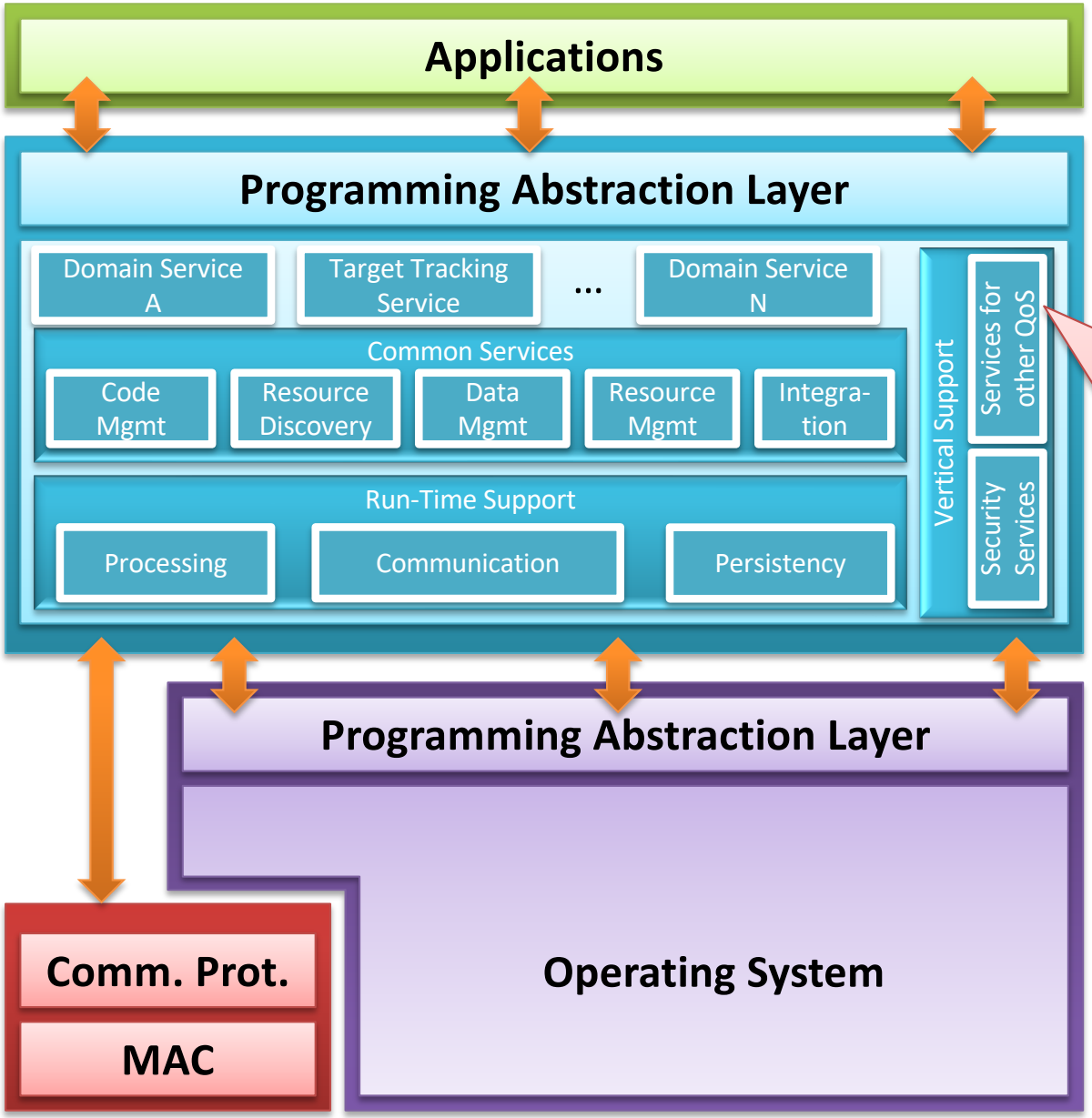
They contain implementations to the abstractions provided.

::.. Architecture (1/4)



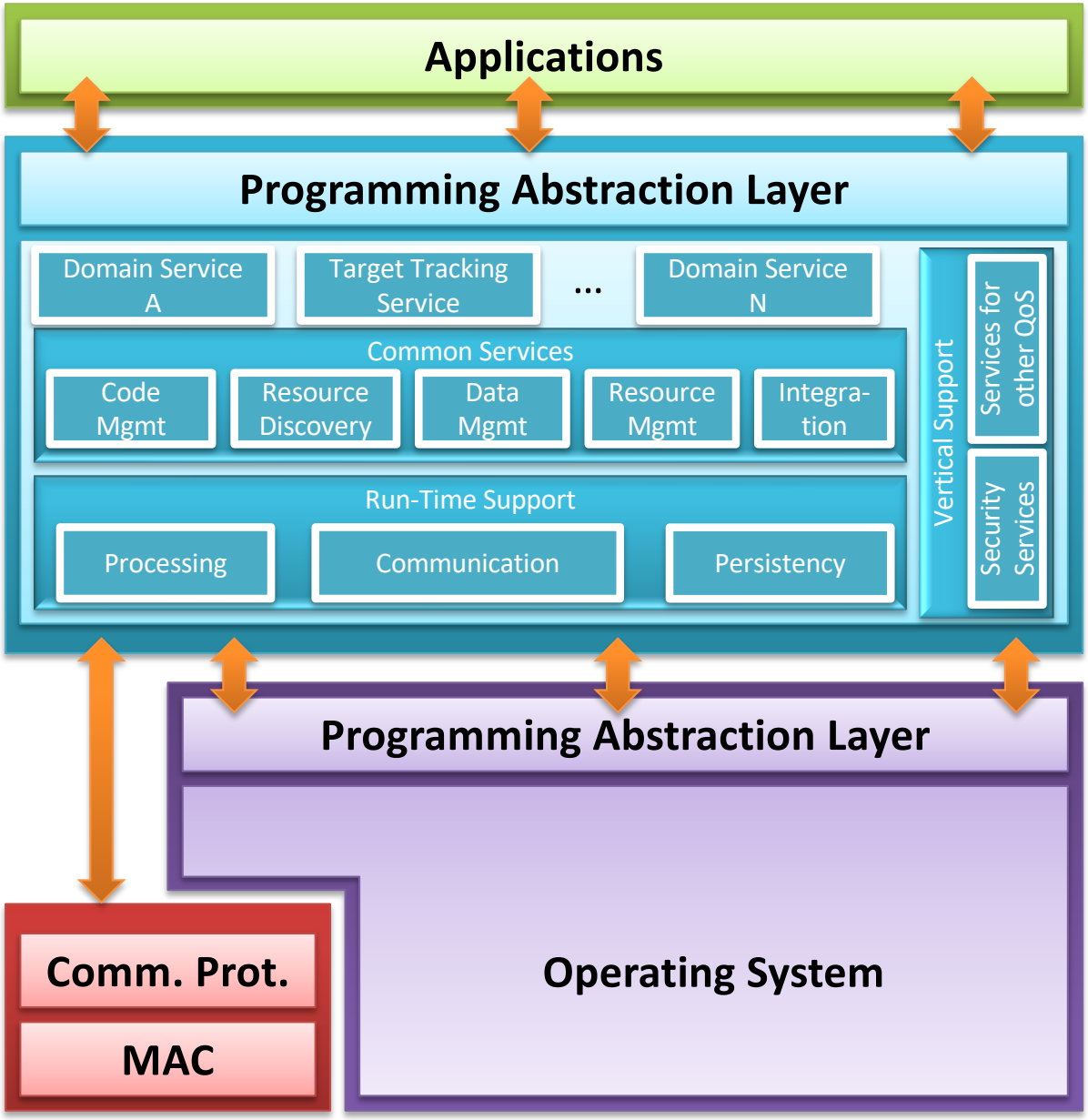
It serves as an extension of the operating system in order to support the overlying abstractions.

::.. Architecture (1/4)

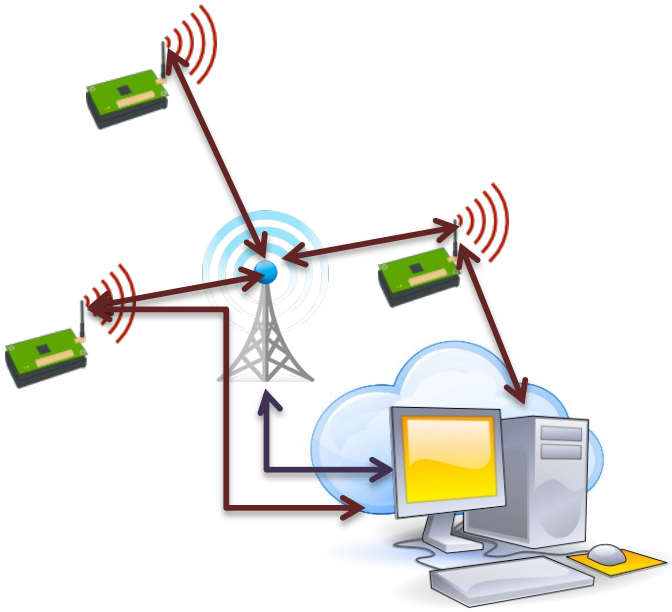


Mechanisms for the pur-pose of special restrict-ions on the quality of the service.

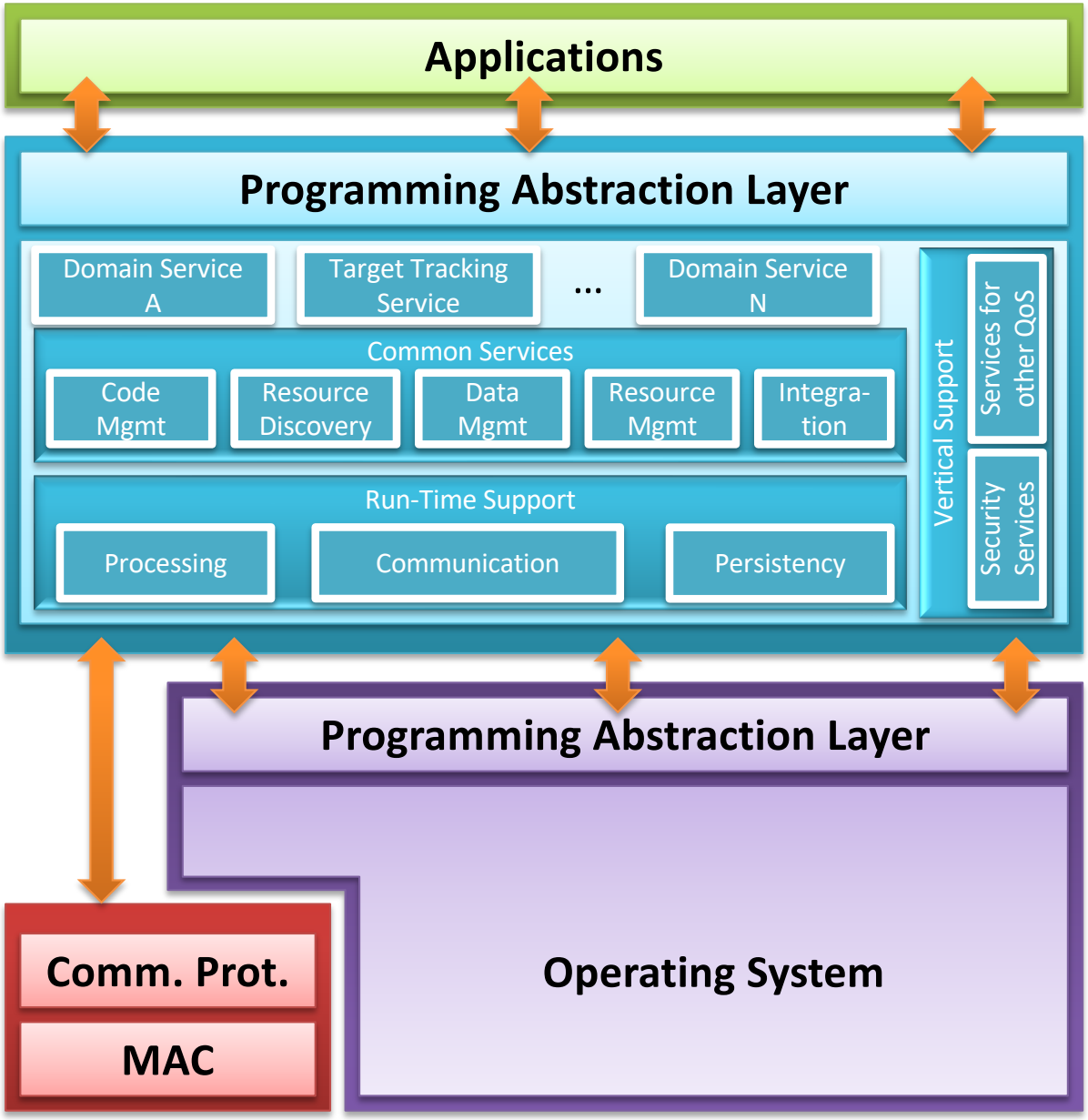
::.. Architecture (1/4)



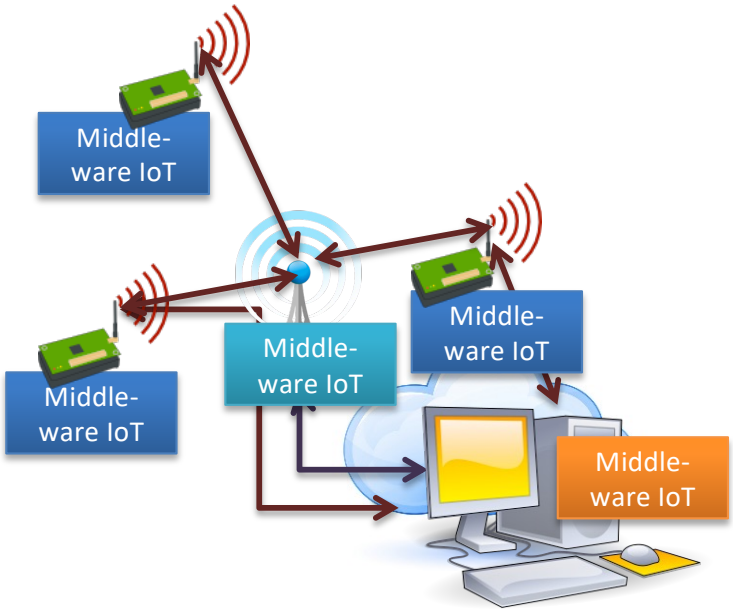
Often the middleware is used in the three different entities that make up a network of sensors, and therefore it must diversify:



::.. Architecture (1/4)



Often the middleware is used in the three different entities that make up a network of sensors, and therefore it must diversify:



::.. Architecture (2/4)

System services represent the core of middleware and can be classified as common or domain services with respect to their horizontal or vertical nature compared to the various application contexts. The common services are the following:

- Code Management - responsible for the migration and updating of the code in the network;
- Data Management - capable of acquiring, storing, synchronizing, analyzing and querying data shared within the network;
- Discovery Resources - responsible for identifying new nodes or detecting those that have become unavailable;
- Resource Management - responsible for optimizing node resources, such as memories or communication modules, and for the network, such as topology, or routing;
- Integration - used to bridge among other networks.

::.. Architecture (3/4)

Runtime support provides the underlying application execution environment and can be seen as an extension of the operating system that provides task scheduling, inter-process communication (IPC), advanced memory control, and energy control.

The need for this level is dictated by the fact that not all hardware platforms provide the same functionality and guarantees.

It is often implemented as a virtual machine above a specific operating system.

A Quality of Service (QoS) mechanism is an advanced feature of middleware, often transversal to a specific application context.

::.. Architecture (4/4)

The network infrastructure expresses a set of QoS parameters such as message delivery delay, jitter, message loss, bandwidth and throughput; while for applications the QoS requirements are expressed as data accuracy, aggregation delay, coverage and duration of the system.

Middleware behaves like a broker between applications and the network, translating and controlling QoS metrics between the two different components. If the application QoS parameters cannot be reconciled with those offered by the network, the middleware must negotiate new agreements according to the principle of supply and demand.

::.. Classification (1/5)

Different classes of middleware have been proposed for WSNs, many of them have been developed in the context of academic research projects and have not exceeded the mere prototype stage, others have matured as true commercial or open source products of wide use. The classification of the various middleware approaches is as follows:

- **Database approach** - in this approach the whole network of sensors is considered as a large database. An application can carry out queries using the Structured Query Language (SQL) proper to the relation model of the database. This approach facilitates a simple and easy communication scheme between users and the network, but generally does not offer temporal relationships between events in the network. An example is TinyDB.

::.. Classification (2/5)

- **Event-based Approach** - adopt a publish/subscribe model, to define, store/delete, detect and deliver events. The application is notified when a specific event is detected and delivered. The middleware identifies the publishers, in which the events are generated (the sensor nodes) and the subscribers, who are interested in receiving events of interest (the base stations):
  - The sensors advertise the available topics;
  - Applications choose which topics to subscribe to;
  - The interests of the subscribers are sent to the sensors, to activate the notification when there are events of interest.

  The scope is of minimizing the number of transmissions. An example is TinyDDS.

::.. Classification (3/5)

- **Application-driven approach** - applications are given more privileges up to the network protocol stack and network configuration based on application requirements. Therefore, the application dictates the management of the network considering its service quality problems as the main concern. The service quality requirements are appropriately modeled and represented, and the middleware determines the set of sensors that can satisfy these requirements. Because of the constraints on consumption and the dynamic network topology, the allowable set is bound to a subset of sensors.

::.. Classification (4/5)

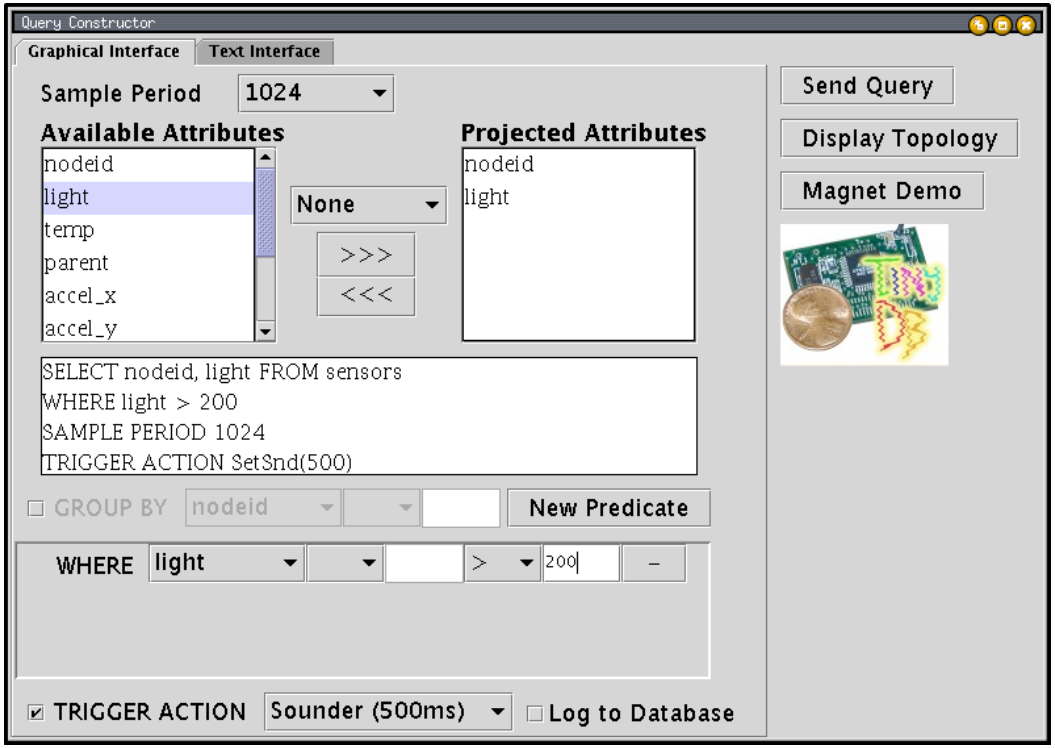
- **Approach with Virtual Machines (MV)** - it is based on the creation of abstractions, virtual machines for the note, built above the operating system hosted by sensor nodes. The designer programs the virtual machine, which will be executed on the various sensors that will execute it, according to the "write once and execute many times" paradigm, through a set of heterogeneous sensors. The modularity of the virtual machine code allows the writing of concise code, reducing a scarce memory occupation and the energy consumption when dynamically updating the applications through the networks. An example is the use of LabVIEW.

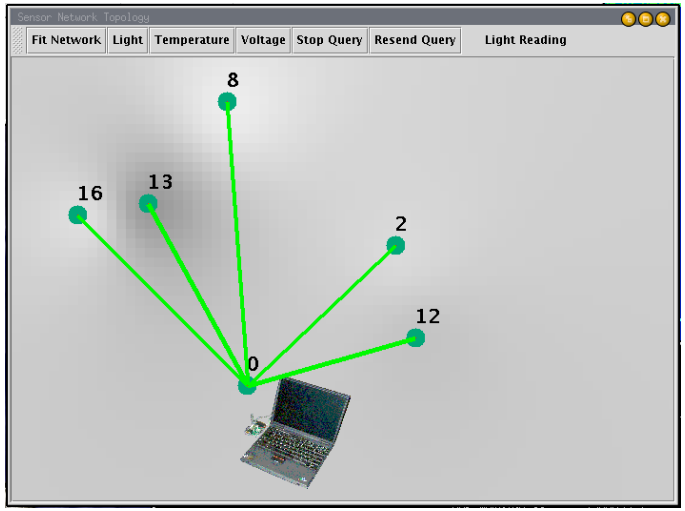
::.. Classification (5/5)

- **Approach based on tuple space** - based on the shared memory paradigm, which represents a repository of tuples that can be accessed concurrently. The model identifies the entities that use the data, called consumers, and the entities that produce the data, called producers. The producers put their data as tuples in space, while the consumers retrieve the data from the space that correspond to a special template. An example is TinyLime.
- **Service approach** - based on service middleware, which makes the services available and easily accessible according to standardized protocols, without worrying about the underlying technological details. This approach facilitates the design and implementation of sensor network applications. Each sensor is abstracted by a service to access the data detected by the sensor and additional sensor description metadata. An example is TinySOA.

::.. TinyDB (1/3)

TinyDB is a query processing system to extract information from a sensor network with TinyOS. Unlike the existing solutions for data processing in TinyOS, TinyDB does not require writing nesC code. Instead, TinyDB provides a simple and SQL-like interface to specify the data you want to extract, with additional parameters.





::.. TinyDB (2/3)

TinyDB GUI

TinyDB Client API

JDBC

DBMS

PC side

Mote side

0

0

1

2

3

8

TinyDB query processor

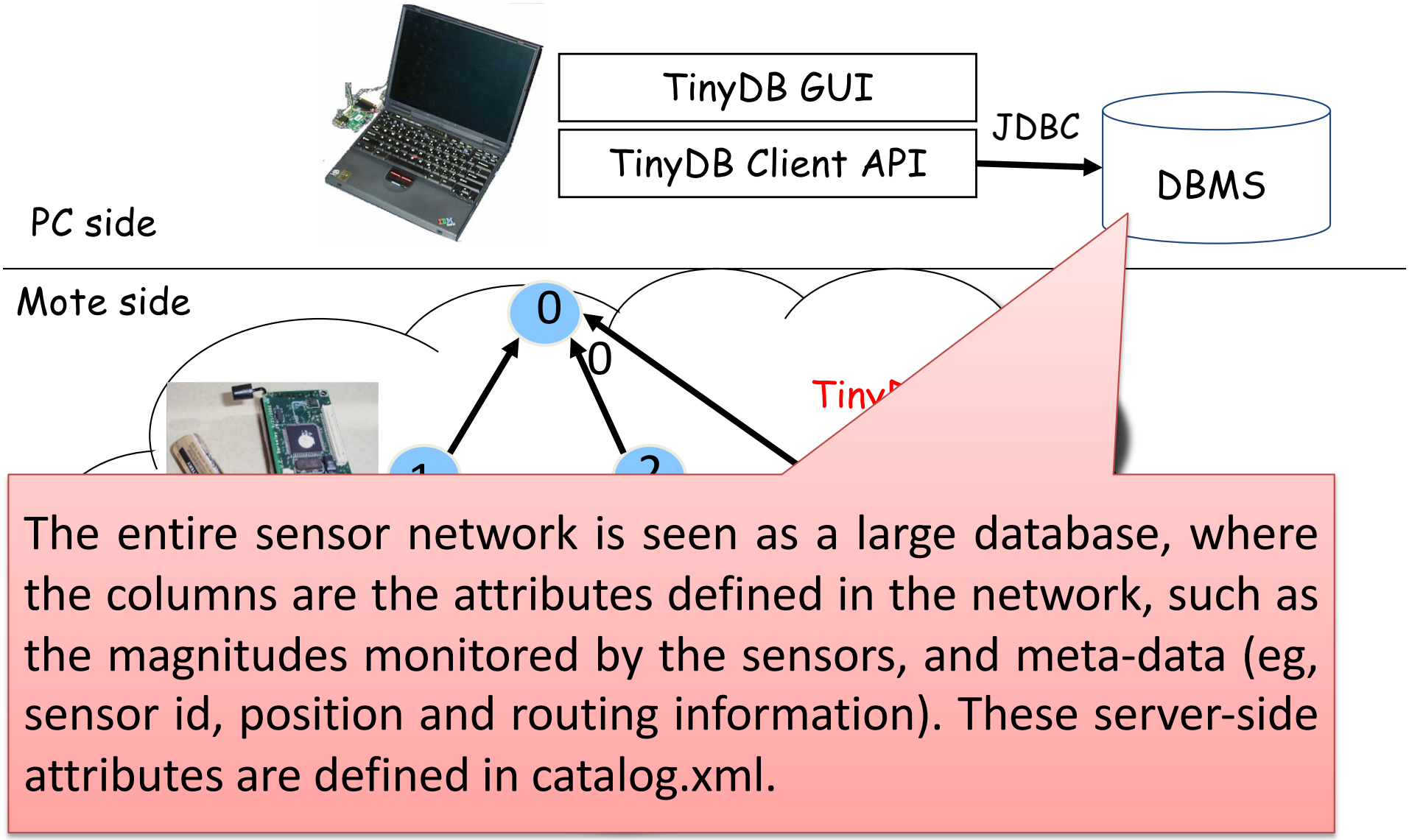
4

5

6

7

::.. TinyDB (2/3)



The entire sensor network is seen as a large database, where the columns are the attributes defined in the network, such as the magnitudes monitored by the sensors, and meta-data (eg, sensor id, position and routing information). These server-side attributes are defined in catalog.xml.

::.. TinyDB (3/3)

SELECT
AVG(temp)
WHERE
light > 400

Queries

Results

T:1, AVG: 225
T:2, AVG: 250

Multihop
Network

Query Processor

get ('t

getTempl

~10,000 lines of code nesC

~5,000 lines of code Java (server)

~3200 Byte in RAM (w/ 768 byte heap)

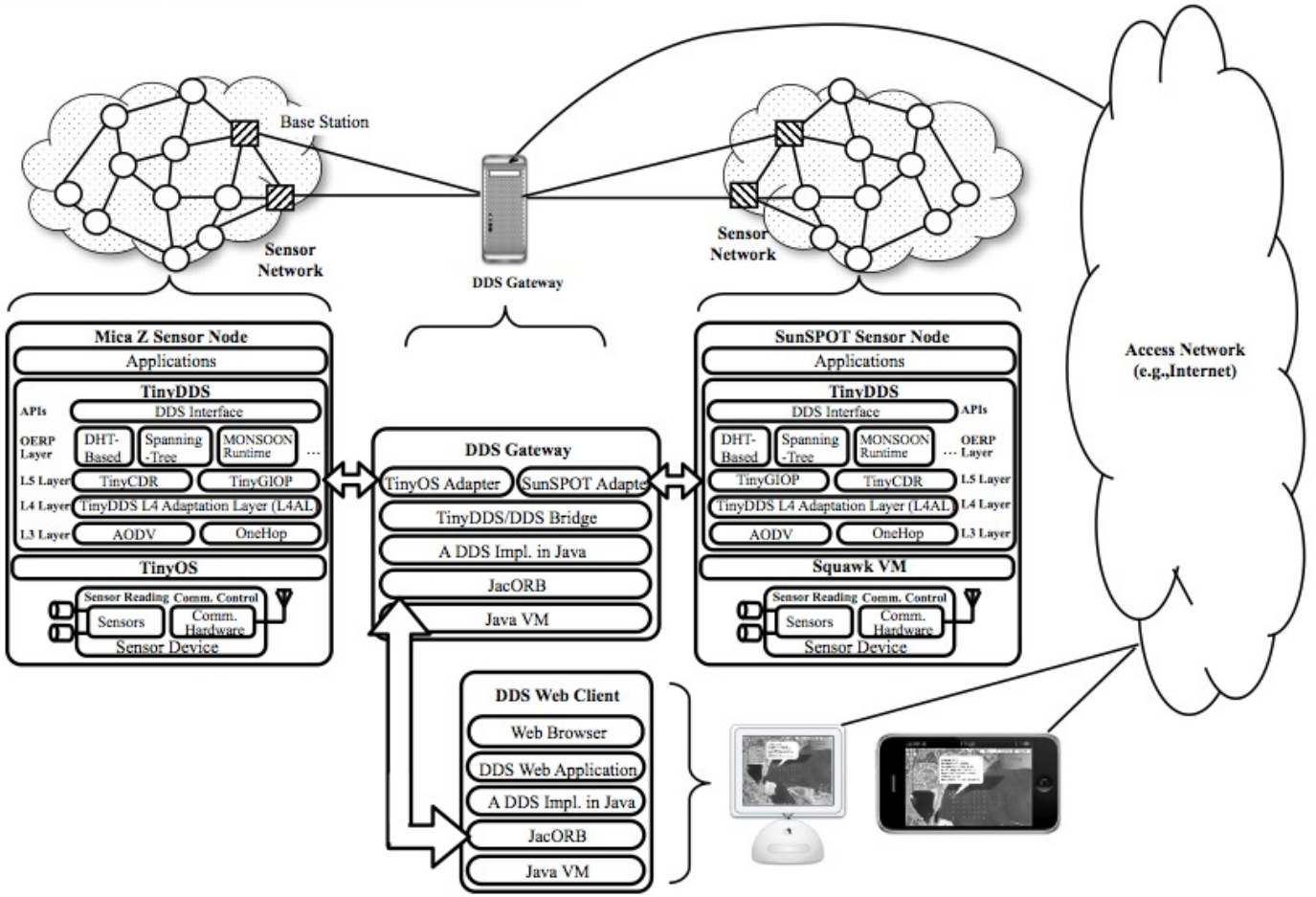
~58 kB of compiled code

(3 times larger than the second largest TinyOS program)

...

::.. TinyDDS

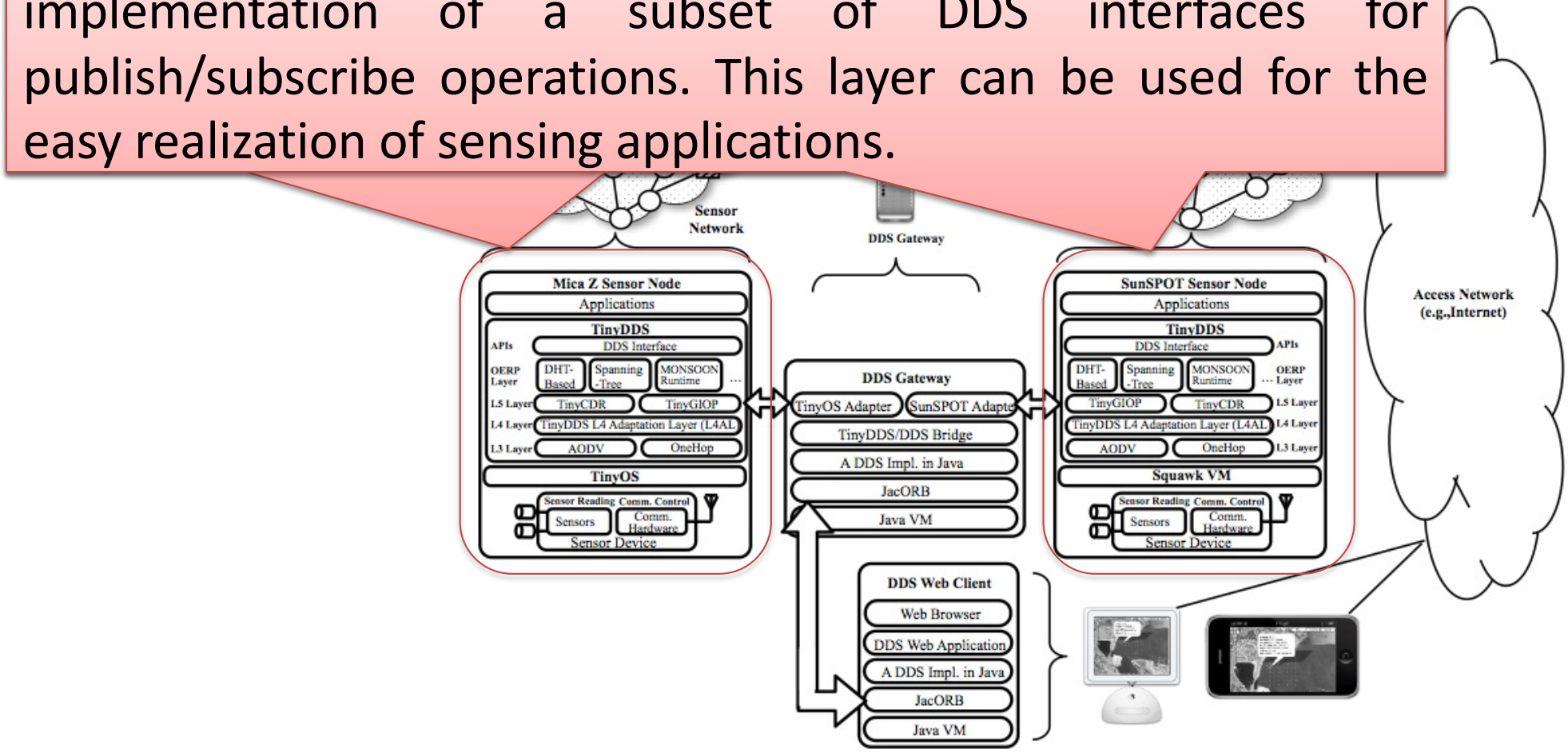
TinyDDS is the implementation of the OMG DDS standard for event middleware in the context of sensor networks using the TinyOS and SunSPOT operating system.



::.. TinyDDS

TinyDDS is the implementation of the OMG DDS standard for event middleware in the context of sensor networks using the

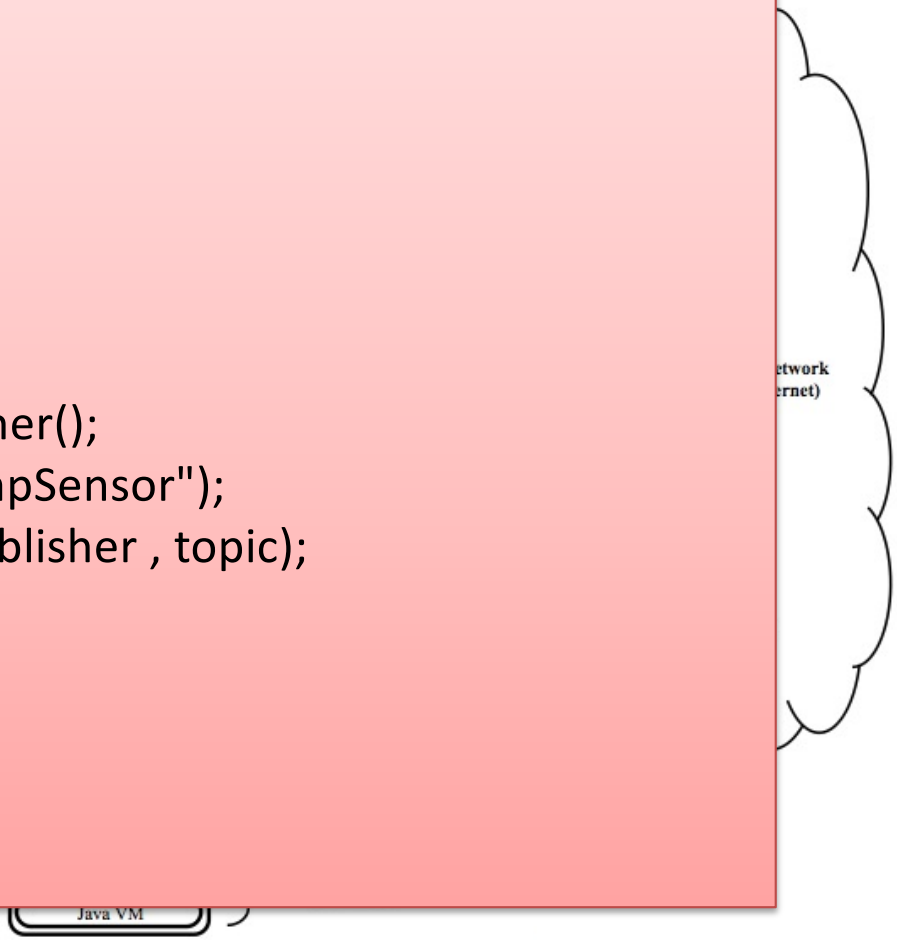
On sensor nodes, the middleware provides the implementation of a subset of DDS interfaces for publish/subscribe operations. This layer can be used for the easy realization of sensing applications.



::.. TinyDDS

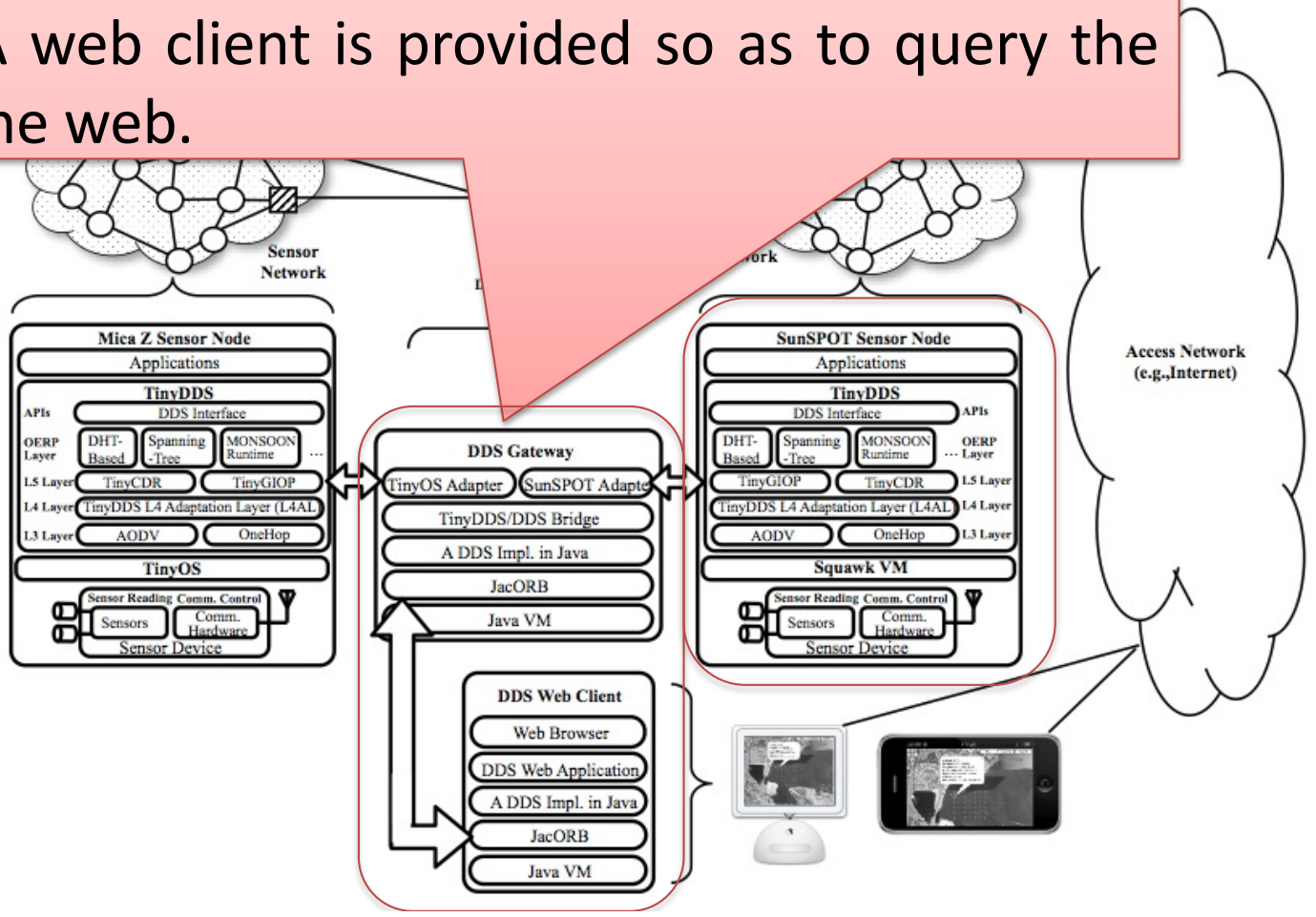
TinyDDS is the implementation of the OMG DDS standard for

```
typedef struct {
cdr_short temperature;
cdr_long time;
} TempData_t;
Publisher_t publisher;
Topic_t topic;
DataWriter_t data_writer;
TempData_t temp_data;
command result_t StdControl.start() {
  publisher = call DomainParticipant.create_publisher();
  topic  = call DomainParticipant.create_topic("TempSensor");
  data_writer = call Publisher.create_datawriter(publisher , topic);
  temp_data.temperature = TempSensor.read();
  temp_data.time = call Time.getLow32();
  call DataWriter.write(data_writer, serialize( data),
  sizeof(TempData_t));
}
```



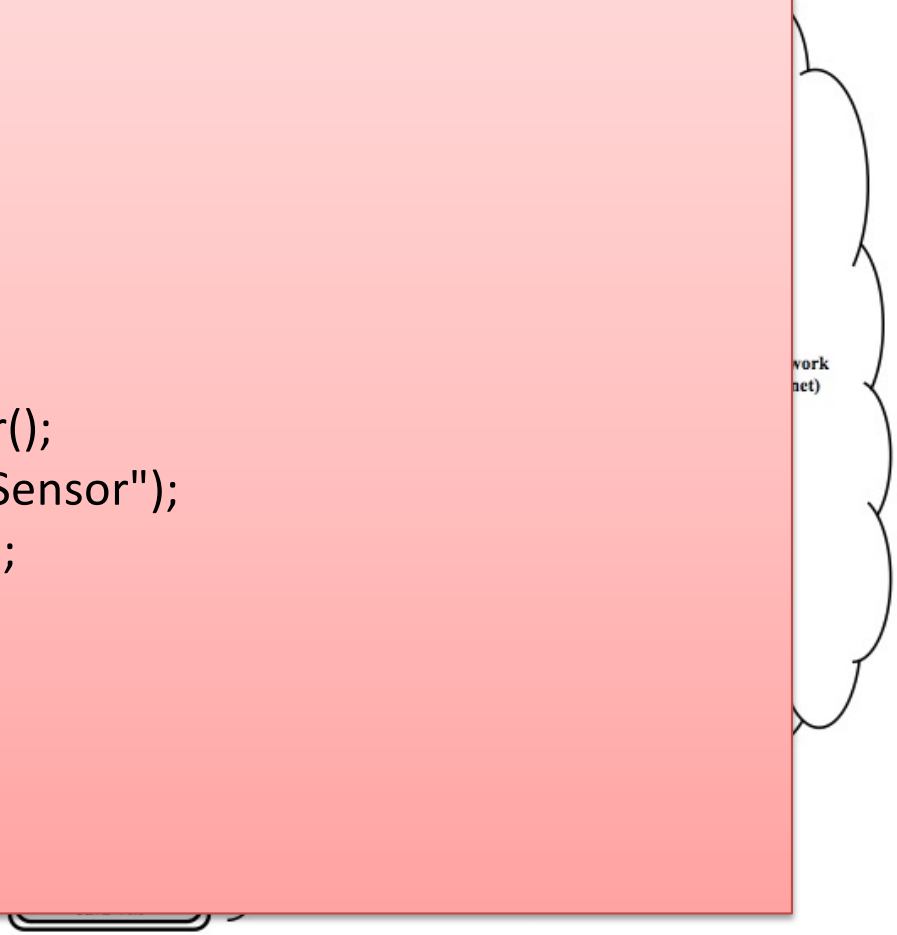
::.. TinyDDS

On the base station, the same subset of functionalities of the OMG DDS standard are implemented so as to easily implement applications that interact with events with applications running on sensor nodes. A web client is provided so as to query the sensors also from the web.



::.. TinyDDS

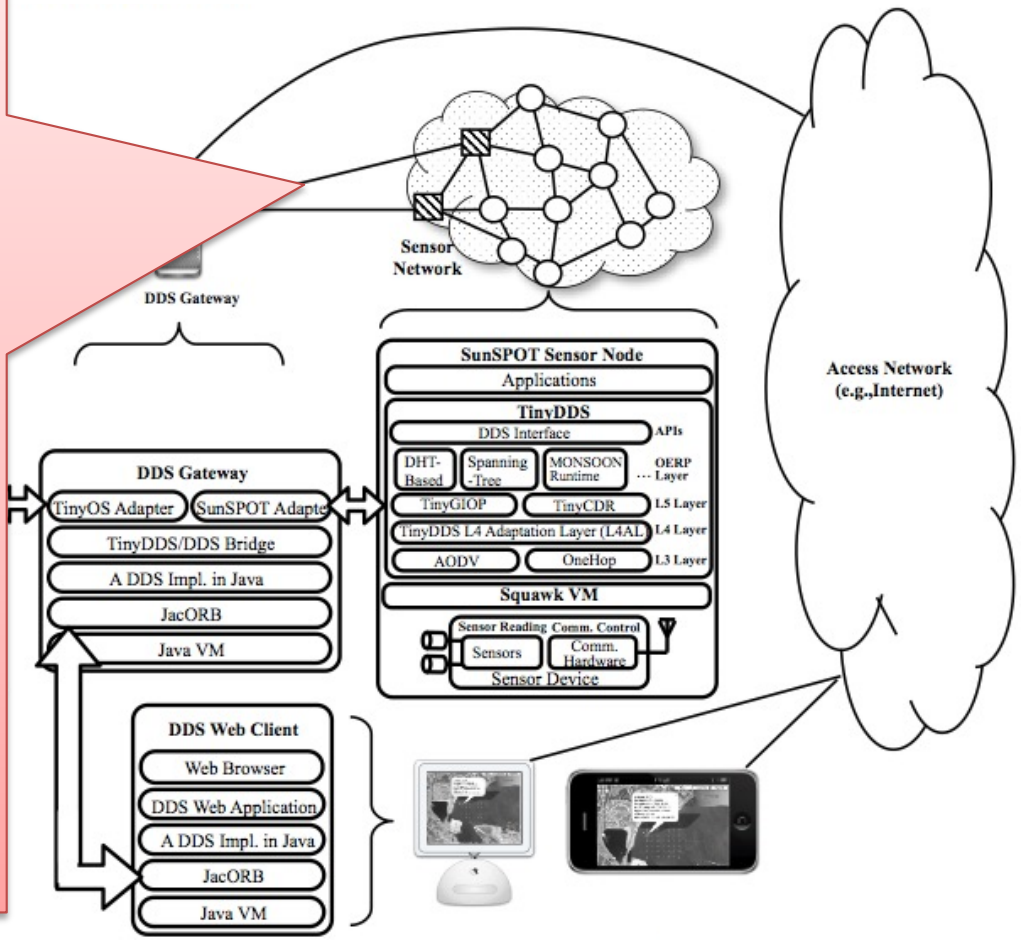
```
class TempData extends Data {
  public short temperature;
  public int time; }
public class Application {
  Publisher publisher;
  Topic topic;
  DataWriter dataWriter;
  TempData tempData;
  DomainParticipant domainParticipant;
  public Application() {
    domainParticipant = new DomainParticipant();
    publisher = domainParticipant.create_publisher();
    topic  = domainParticipant.create_topic("TempSensor");
    dataWriter = publisher.create_datawriter(topic);
    tempData = new TempData();
    tempData.temperature = TempSensor.read();
    tempData.time = (new Date()).getTime();
    dataWriter.write(data.marshall());
  }}
```



::.. TinyDDS

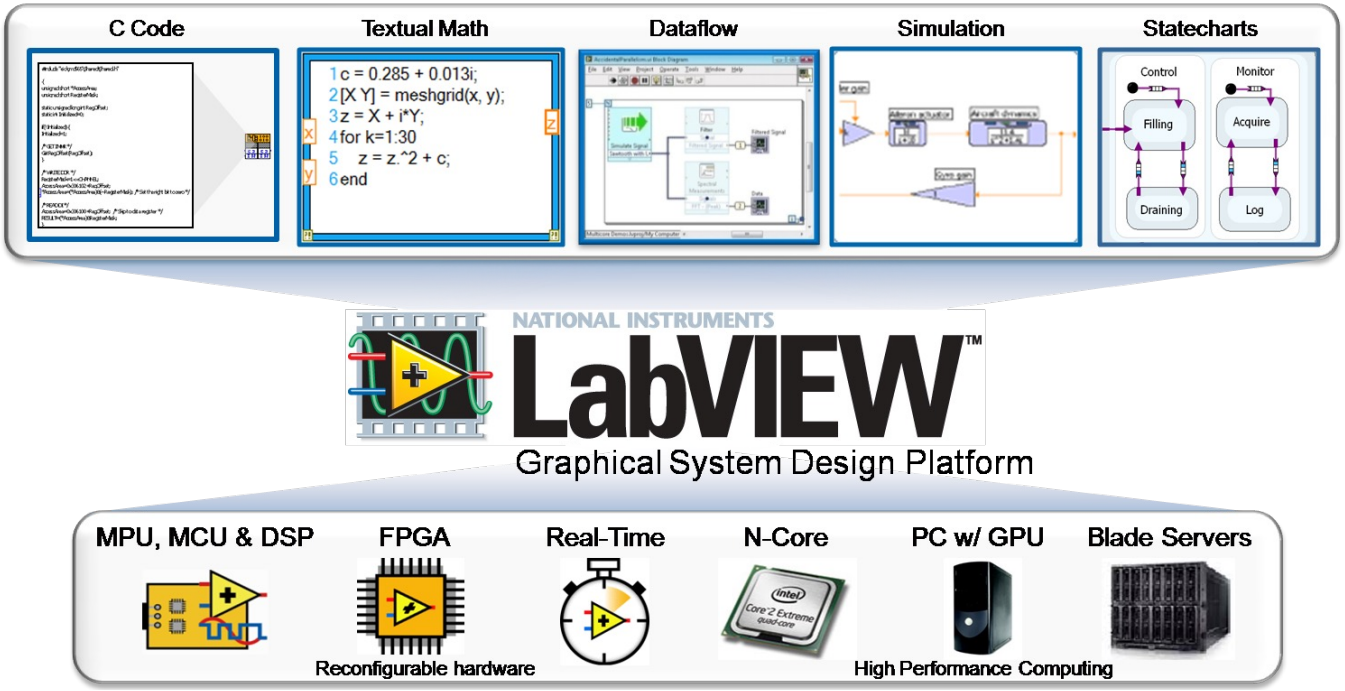
TinyDDS is the implementation of the OMG DDS standard for event middleware in the context of sensor networks using the TinyOS and SunSPOT operating system.

Several routing protocols can be used to implement the overlay network that interconnects the sensors and the base station, creating the Overlay Event Routing Protocols (OERP) layer. This OERP layer allows the application developer to choose an appropriate routing protocol to meet their needs and constraints.



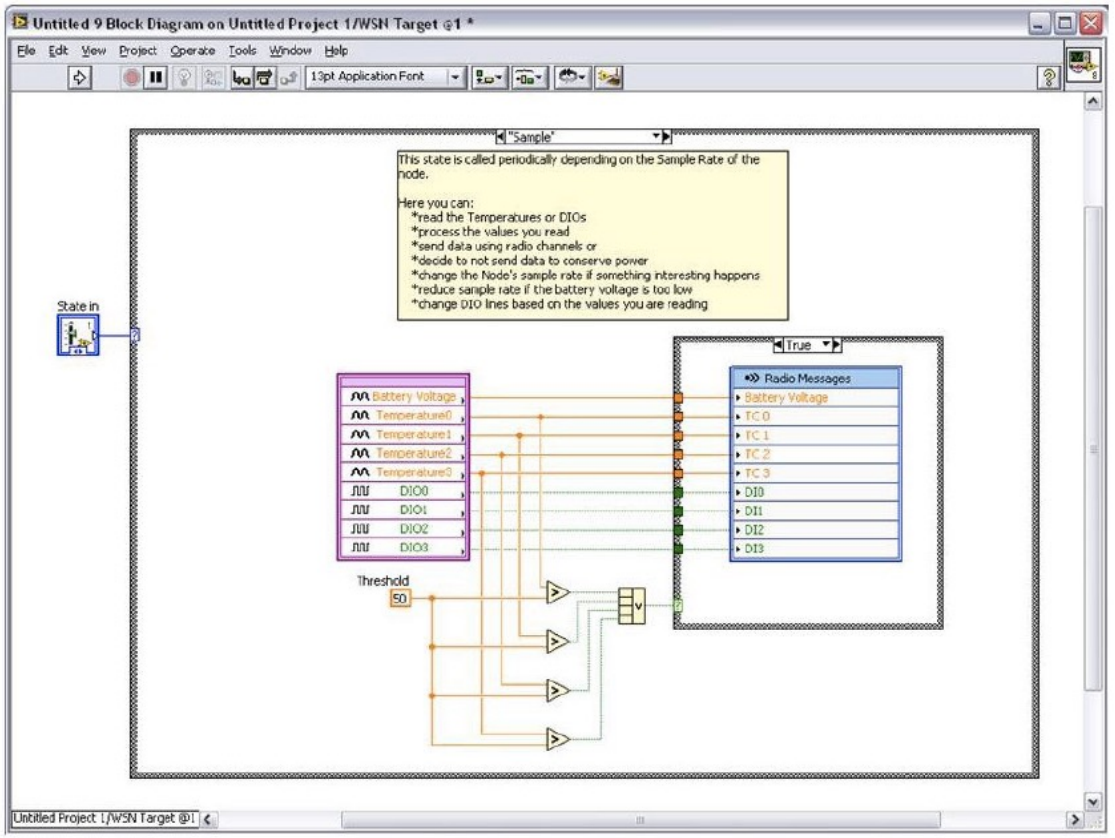
::.. LabVIEW (1/3)

LabVIEW (short for Laboratory Virtual Instrumentation Engineering Workbench) is National Instruments' integrated visual development environment. It is mainly used for data acquisition and analysis, process control, report generation, or for industrial automation. It can also be used to program sensors.



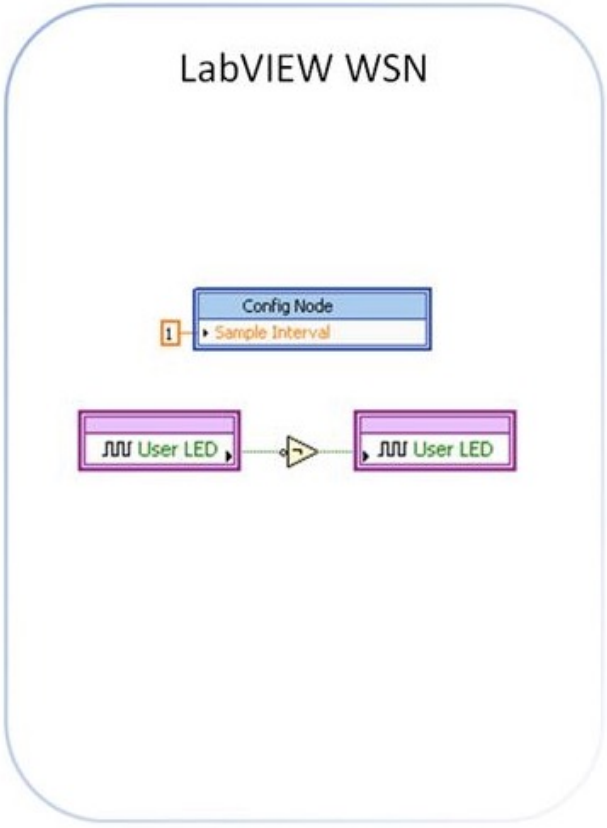
::.. LabVIEW (2/3)

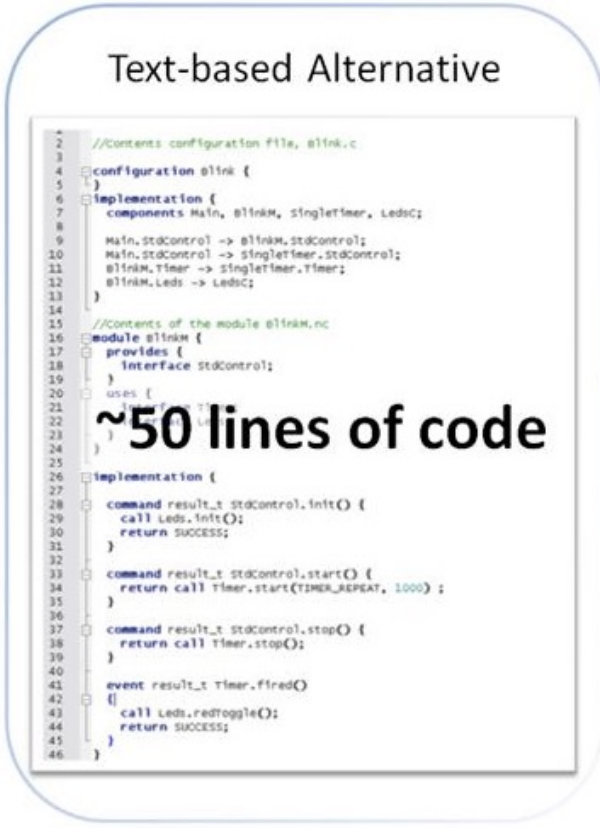
Let's assume we want to use LabVIEW to create and integrate an application in the measurement node so that it transmits data only when the acquired value exceeds a specified threshold. By sending only the most significant data, you save energy and extend battery life.



::.. LabVIEW (3/3)

The programming of sensor nodes traditionally requires the knowledge of embedded systems and the understanding of the specific language of the operating system or of the hardware platform of the sensors to be programmed.



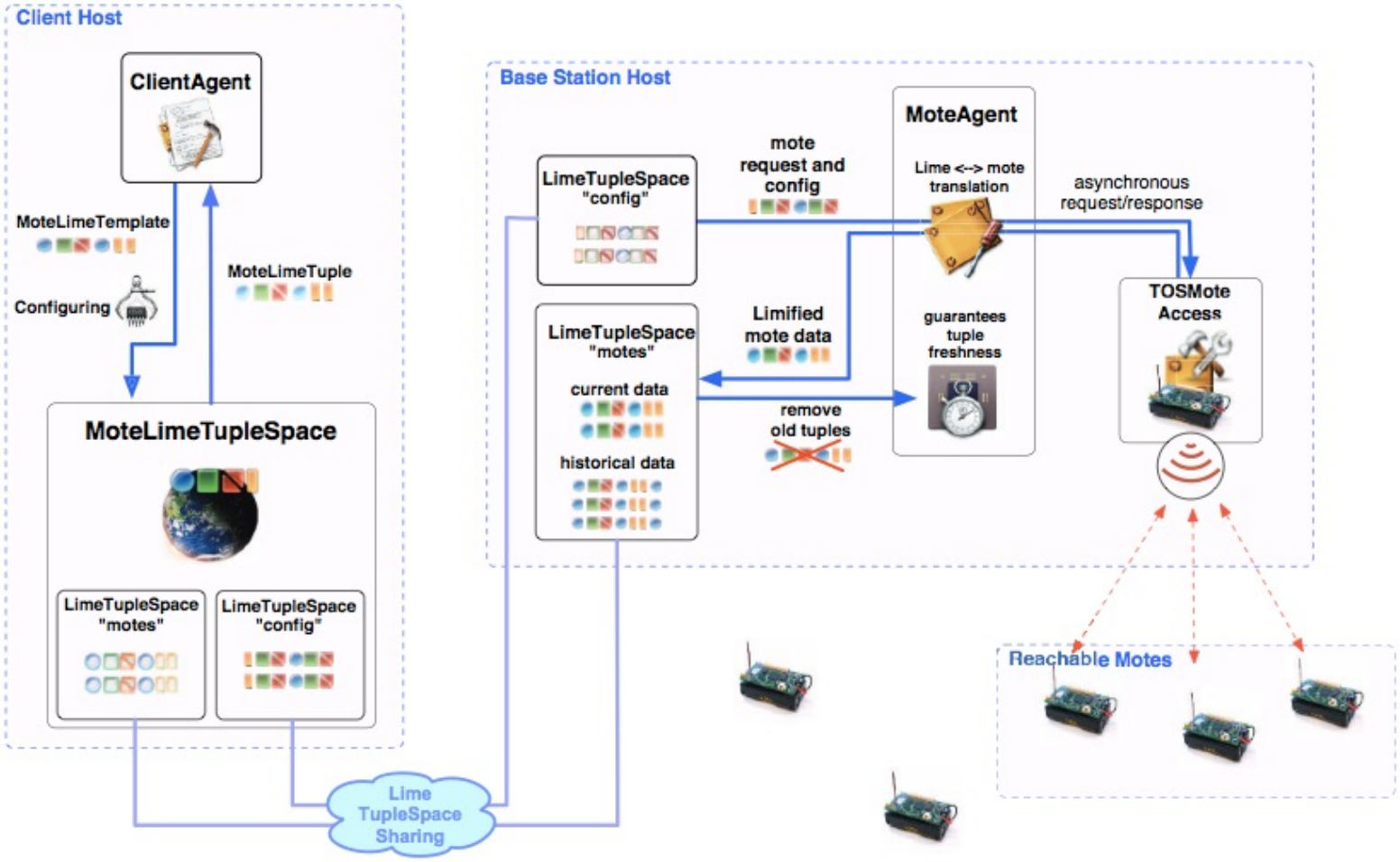


With LabVIEW, you can add intelligence to sensor nodes using a graphical programming approach.

Furthermore, LabVIEW also provides the flexibility to integrate C code directly in line with the graphic code.

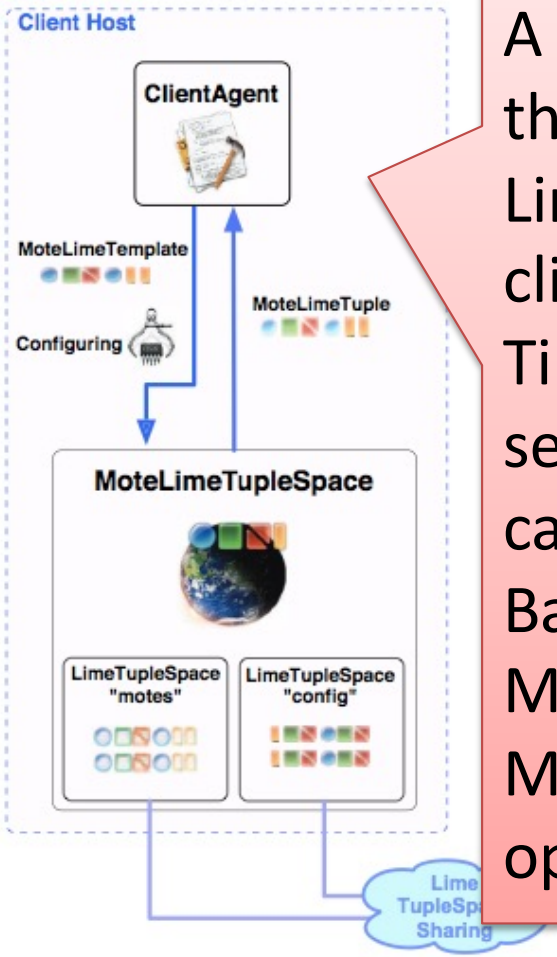
::.. TinyLime

TinyLime adapts the famous tuple space middleware called Lime for the context of sensor networks, making use of the TinyOS functionalities.



::.. TinyLime

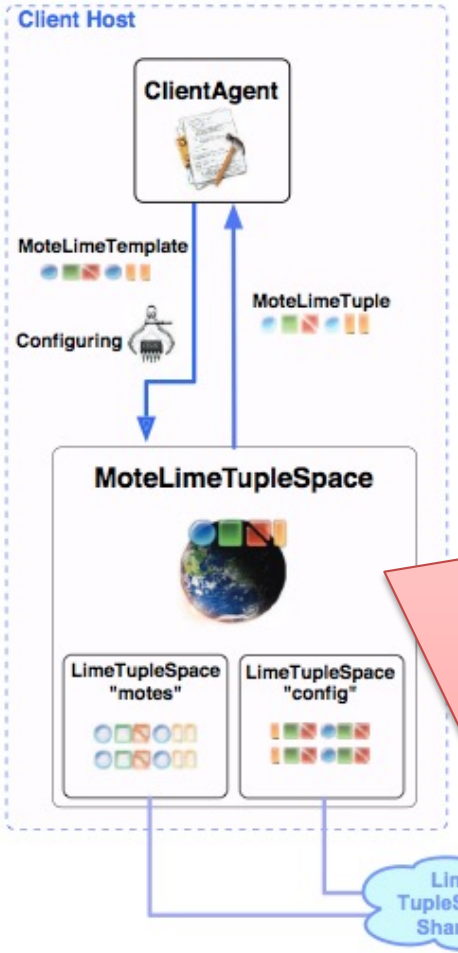
TinyLime adapts the famous tuple space middleware called Lime for the context of sensor networks, making use of the TinyOS functionalities.



A client interacts with the sensors through the MoteLimeTupleSpace class that extends LimeTupleSpace. To query the sensors, a client must know the format of the tuples. In TinyLIME, this format is predefined for sensors on the mote platform, although it can be easily adapted to different platforms. Based on this format, a client can create a MoteLimeTemplate for read operations. MoteLimeTupleSpace also provides operations dedicated to sensor control.

::.. TinyLime

TinyLime adapts the famous tuple space middleware called Lime for the context of sensor networks, making use of the TinyOS functionalities.

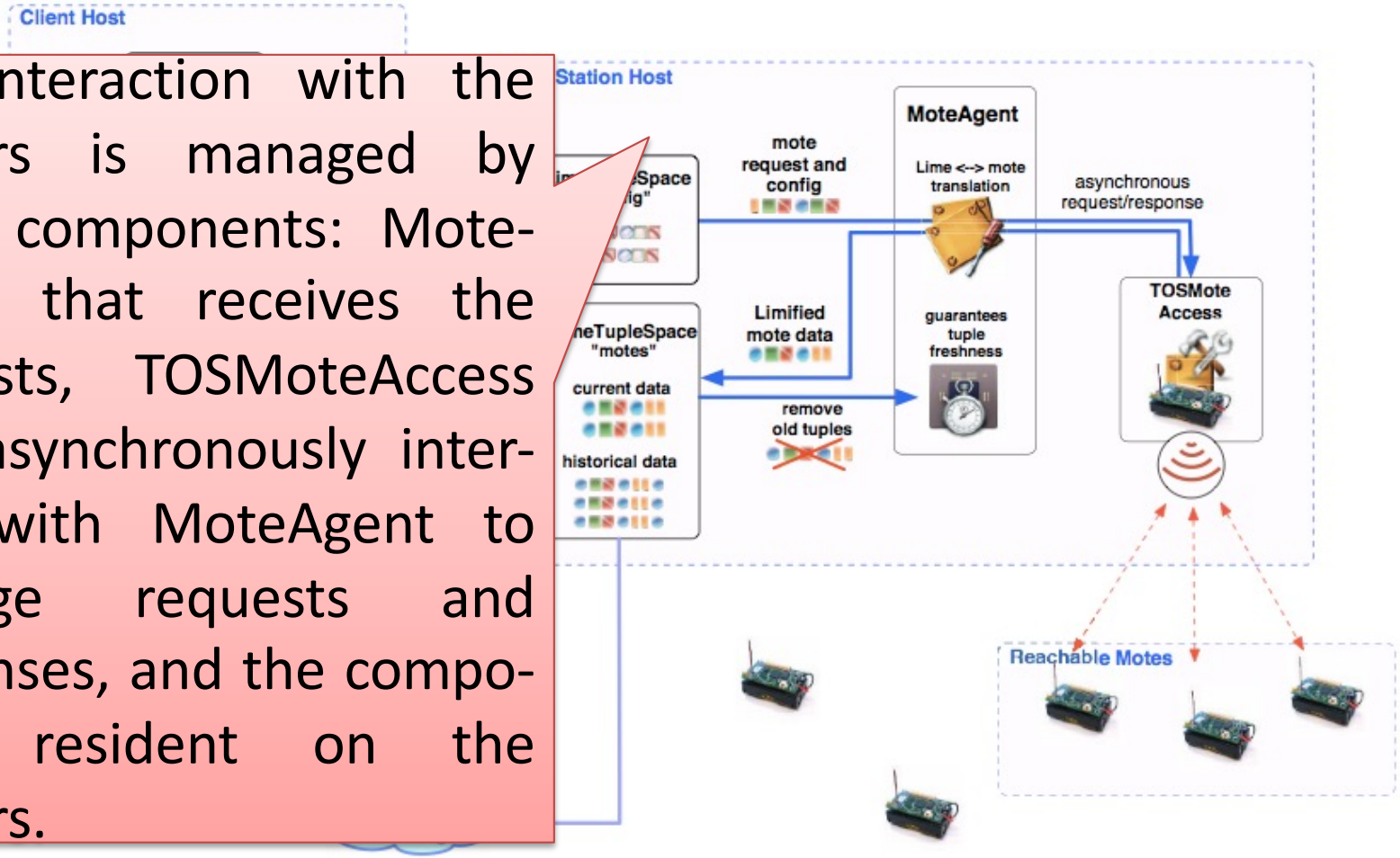


It provides clients with the abstraction of a single tuple space, internally, instead, it hosts two: one with data from the sensors and one with requests from the client. Sensor data is recovered only upon request, saving communications for data that no application needs. Therefore, when a client sends a request, the space of the tuples is queried first. If no match is found or the tuples are old, the sensors are queried.

::.. TinyLime

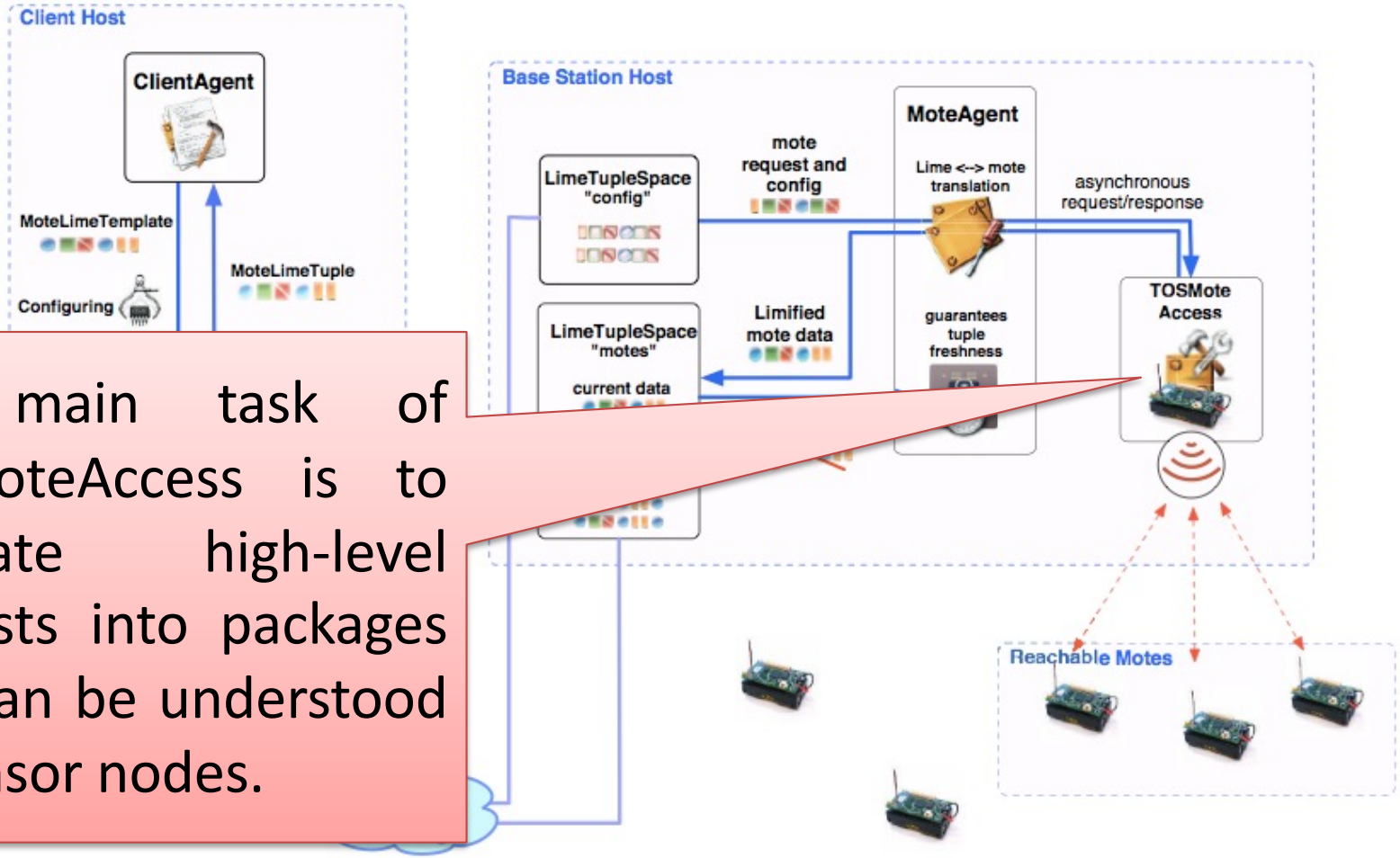
TinyLime adapts the famous tuple space middleware called Lime for the context of sensor networks, making use of the TinyOS functionalities.

The interaction with the sensors is managed by three components: MoteAgent that receives the requests, TOSMoteAccess that asynchronously interacts with MoteAgent to manage requests and responses, and the component resident on the sensors.



::.. TinyLime

TinyLime adapts the famous tuple space middleware called Lime for the context of sensor networks, making use of the TinyOS functionalities.

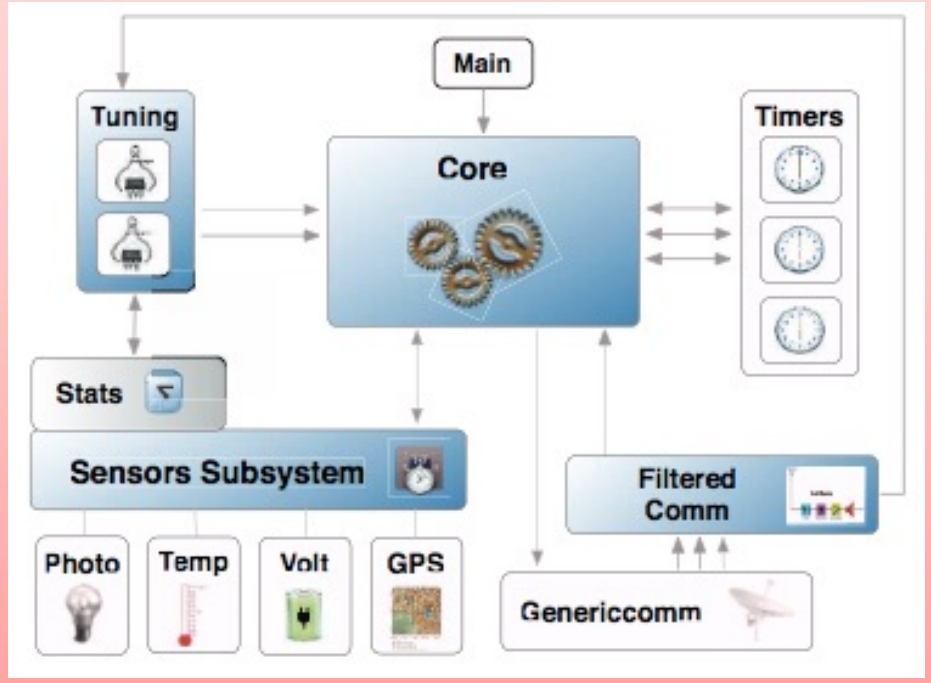


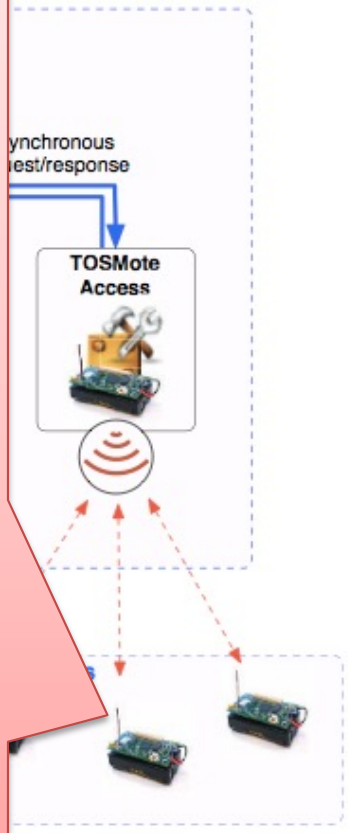
The main task of TOSMoteAccess is to translate high-level requests into packages that can be understood by sensor nodes.

::.. TinyLime

TinyLime adapts the famous tuple space middleware called Lime for the context of sensor networks, making use of the TinyOS functionalities.

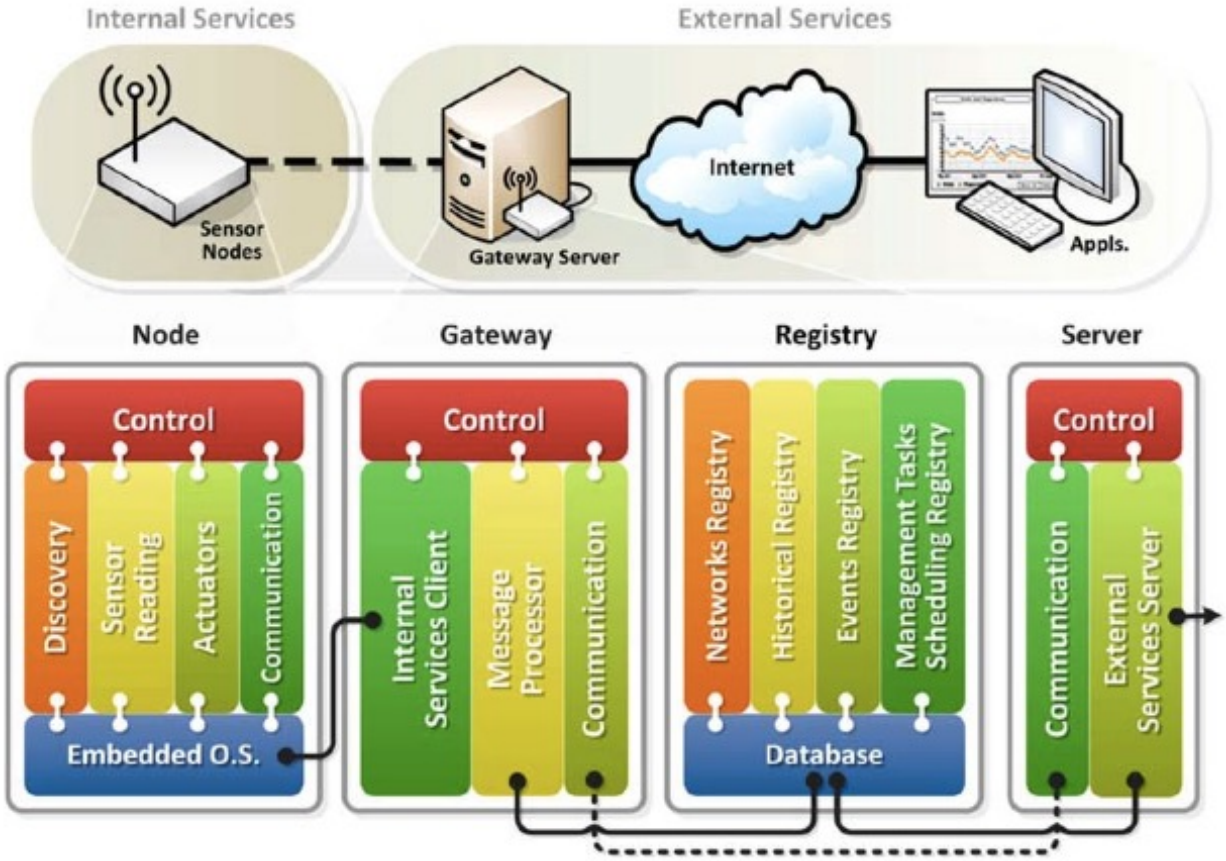
The sensor component has the task of reading the measurements and passing them to the middleware and/or of interpreting the requests.





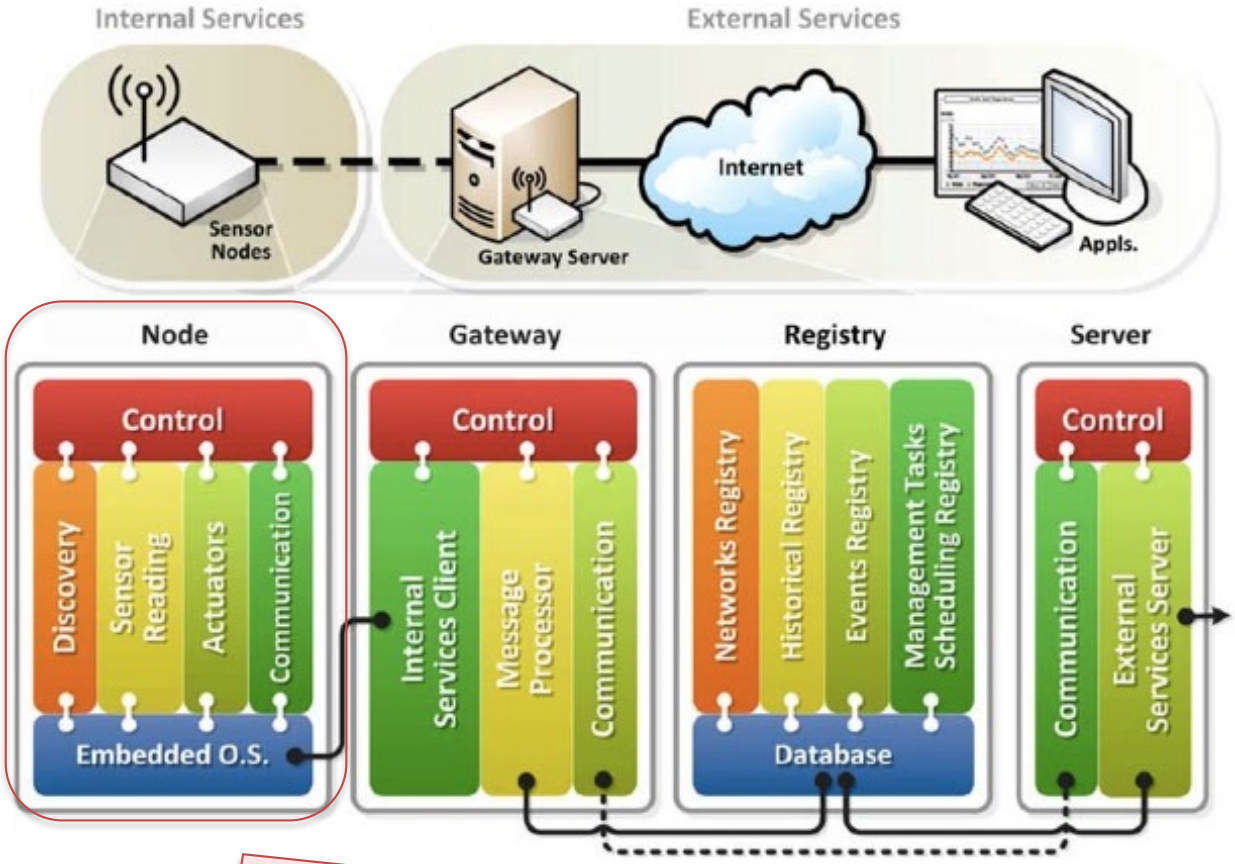
::.. TinySOA

TinySOA is a service middleware for the implementation of monitoring applications. It allows programmers to access wireless sensor networks from their applications using a simple API.



::.. TinySOA

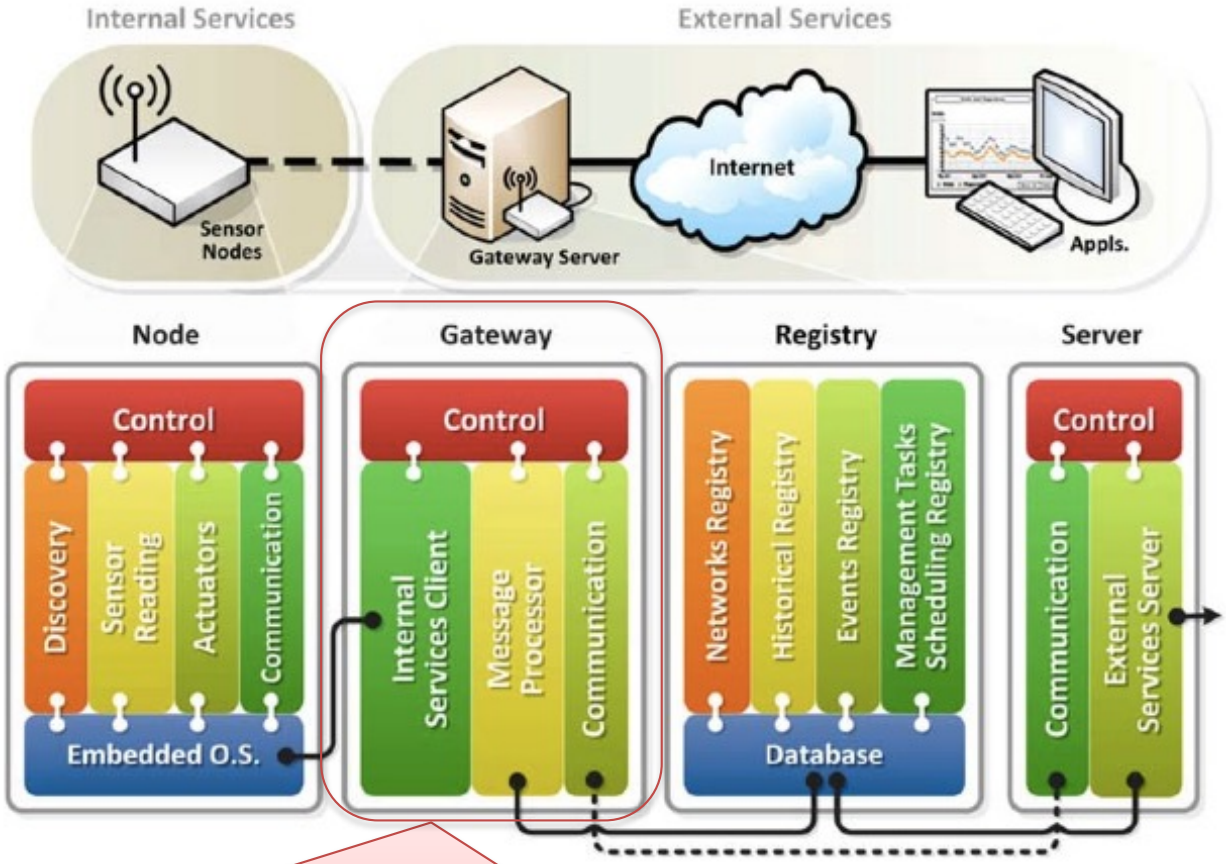
TinySOA is a service middleware for the implementation of monitoring applications. It allows programmers to access wireless sensor networks from their applications using a simple API.



This component encapsulates all the features of a monitoring node, and resides in all the detection nodes of the network.

::.. TinySOA

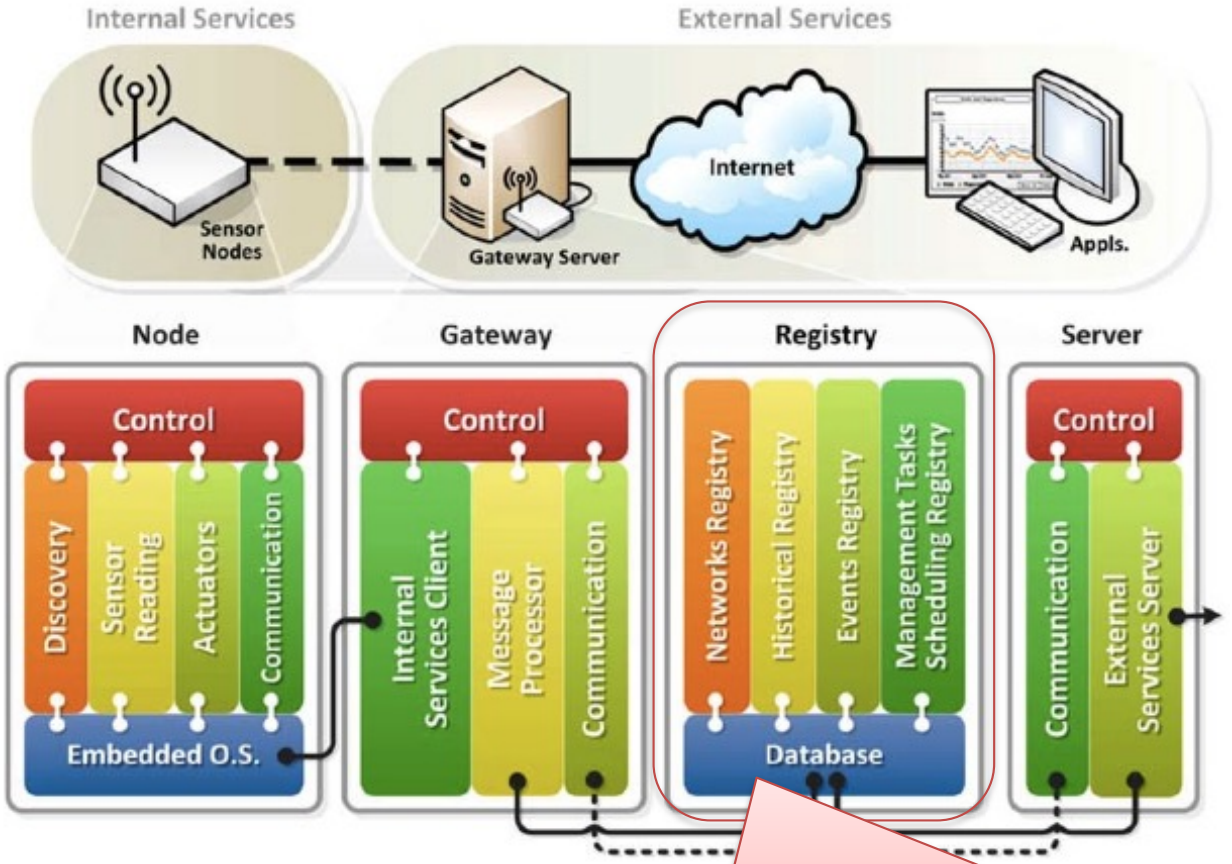
TinySOA is a service middleware for the implementation of monitoring applications. It allows programmers to access wireless sensor networks from their applications using a simple API.



This component is usually found in a specialized node or in a computer, and acts as a bridge between a sensor network and the outside world.

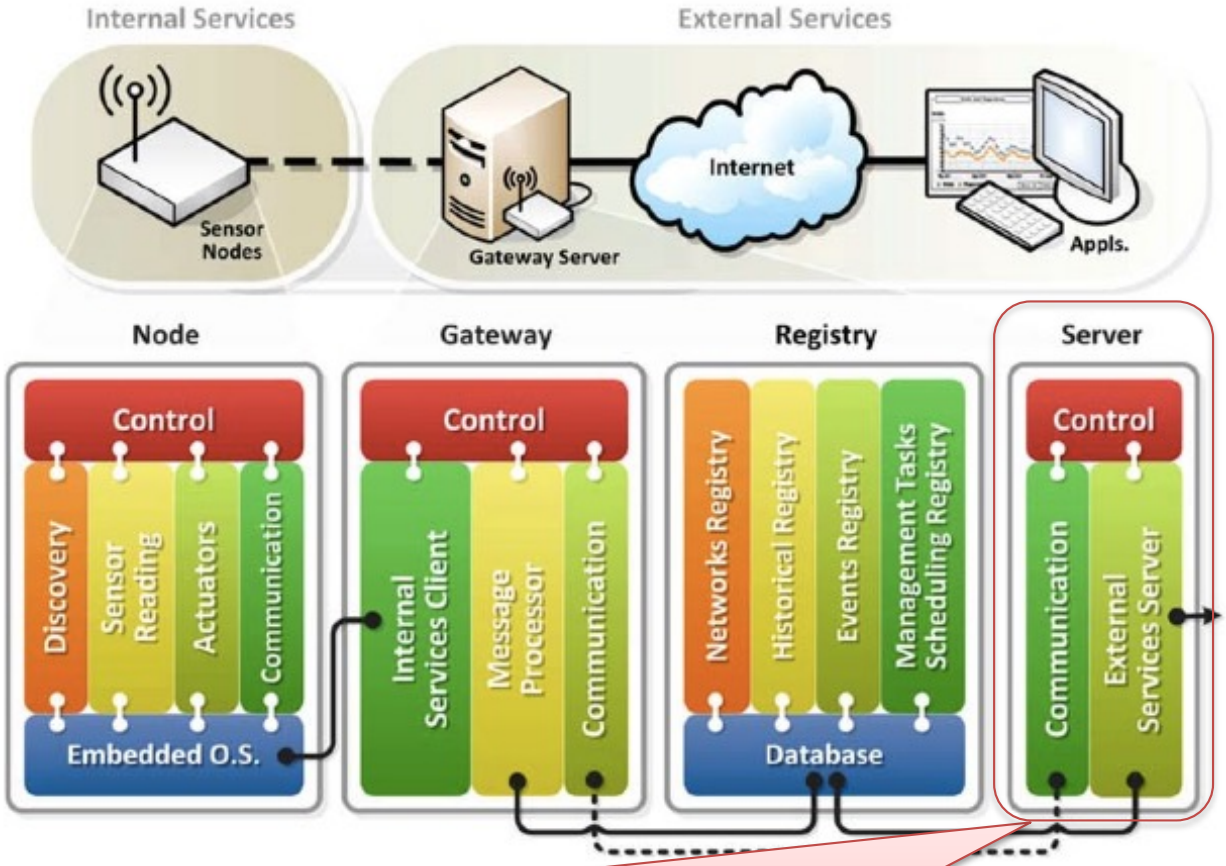
::.. TinySOA

TinySOA is a service middleware for the implementation of monitoring applications. It allows programmers to access wireless sensor networks from their applications using a simple API.



::.. TinySOA

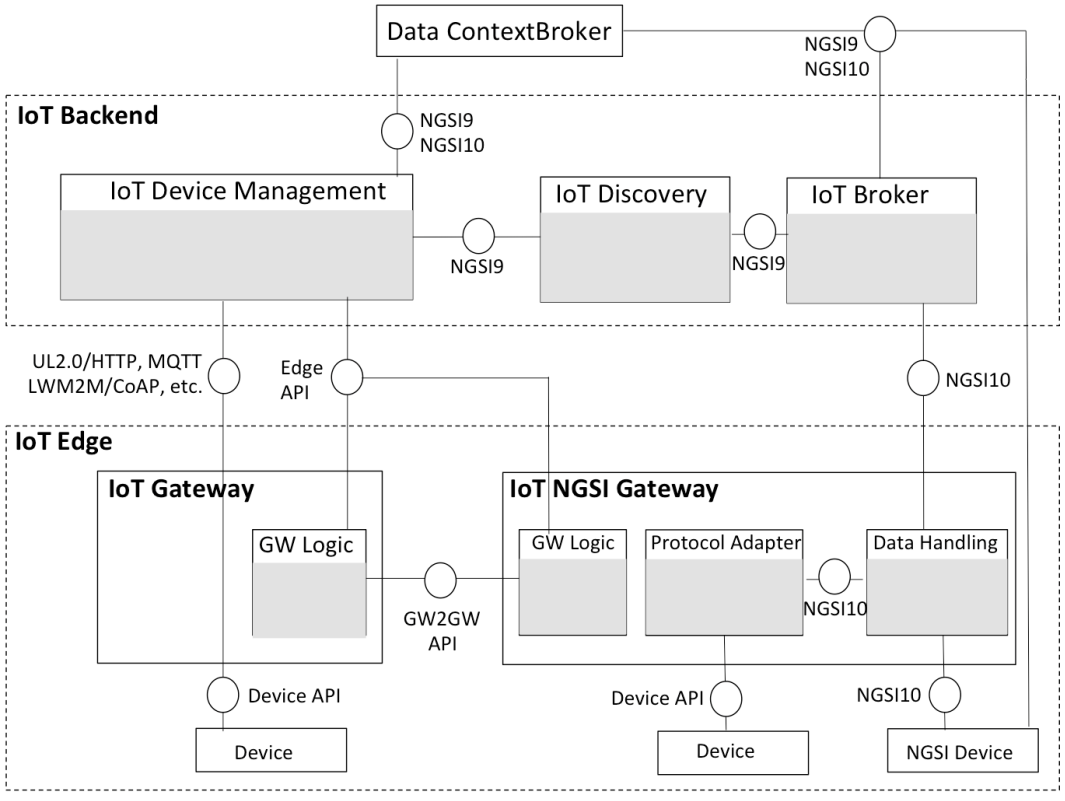
TinySOA is a service middleware for the implementation of monitoring applications. It allows programmers to access wireless sensor networks from their applications using a simple API.



This component acts as a web services provider, abstracting each available WSN as a separate web service to consult the services offered, check the register, consult and record events and maintenance activities.

::.. FIWARE (1/4)

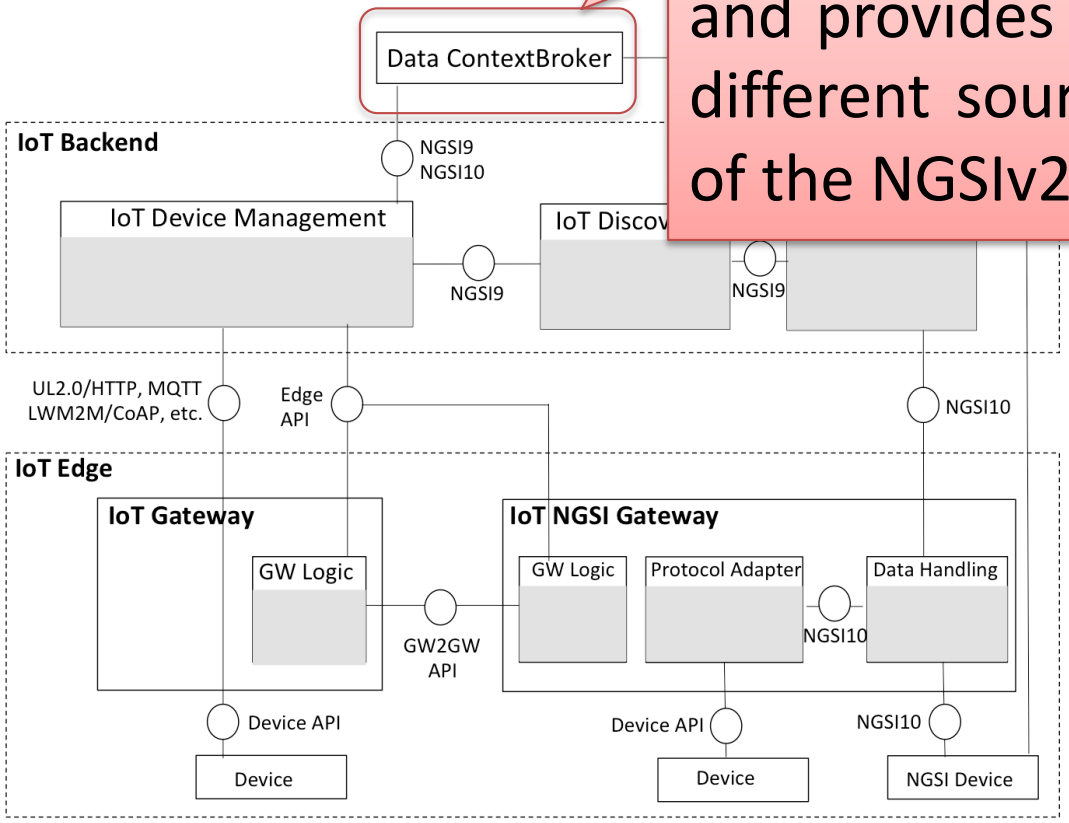
FIWARE is a curated framework of open source platform components which can be assembled together with other third-party platform components to accelerate the development of Smart Solutions.

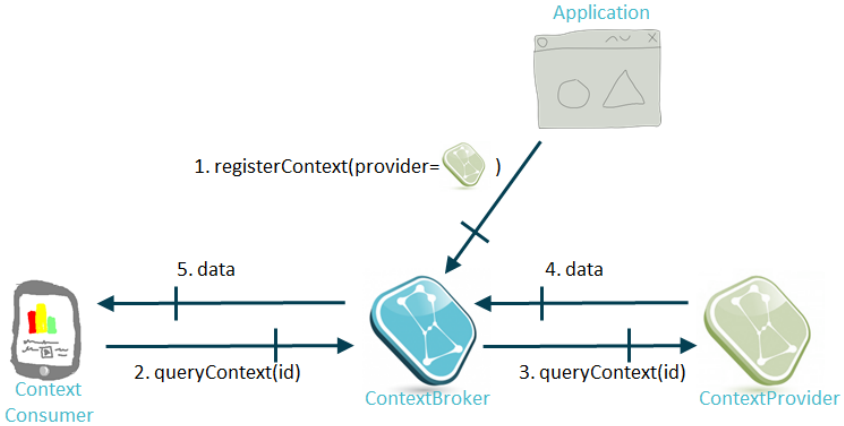


::.. FIWARE (1/4)

FIWARE is a curated framework of open source platform components which can be assembled together with other third-party platform comp
Smart Solutions.

The FIWARE Orion Context Broker is the core component of FIWARE: it gathers, manages and provides access to the data coming from different sources. It is a C++ implementation of the NGSIv2 REST API.

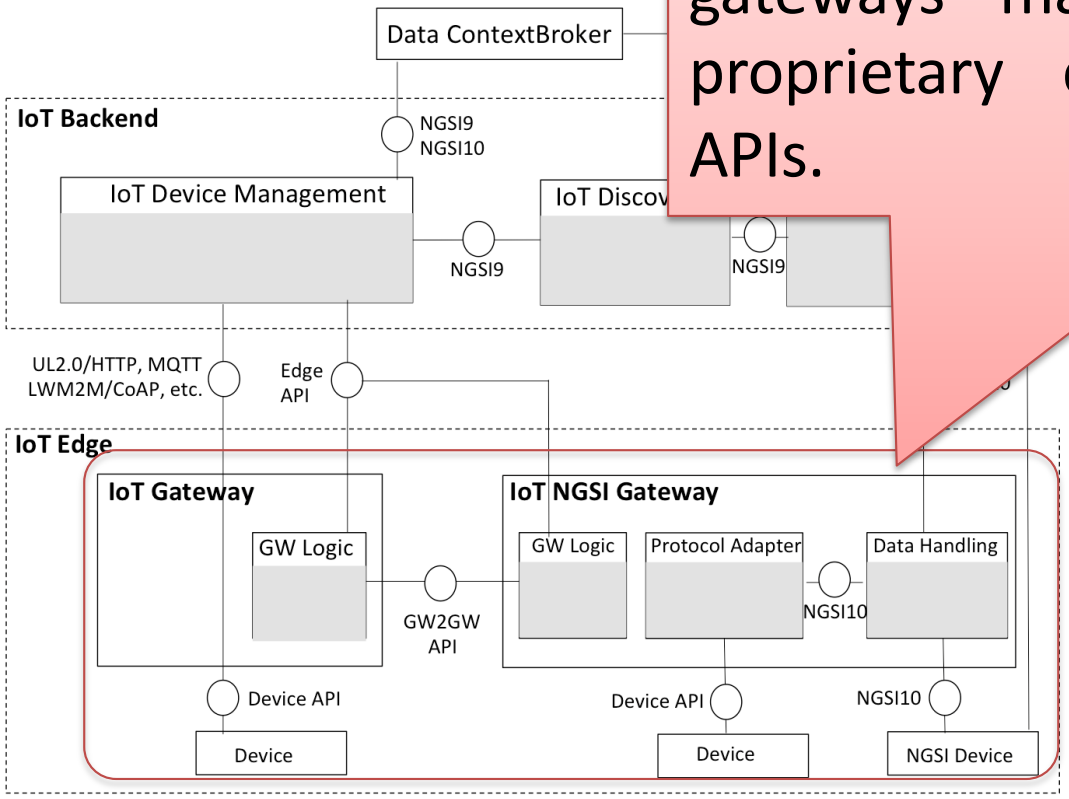


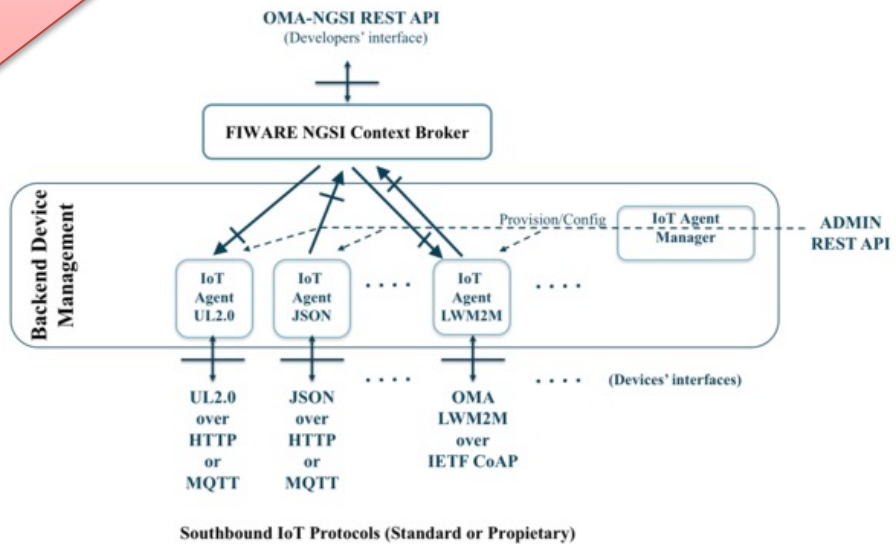


::.. FIWARE (1/4)

FIWARE is a curated framework of open source platform components which can be assembled together with other third-party platform comp
Smart Solutions.

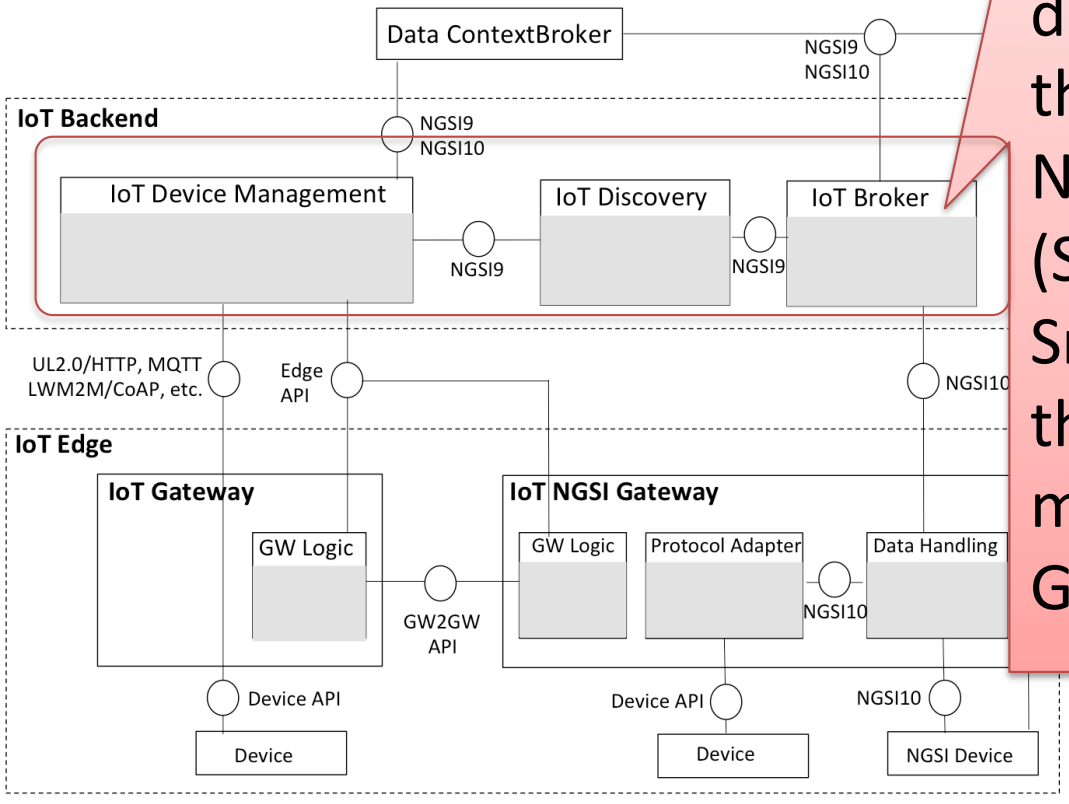
These components connect physical devices to a FIWARE platform. Devices and/or gateways may use different standard or proprietary communication protocols and APIs.





::.. FIWARE (1/4)

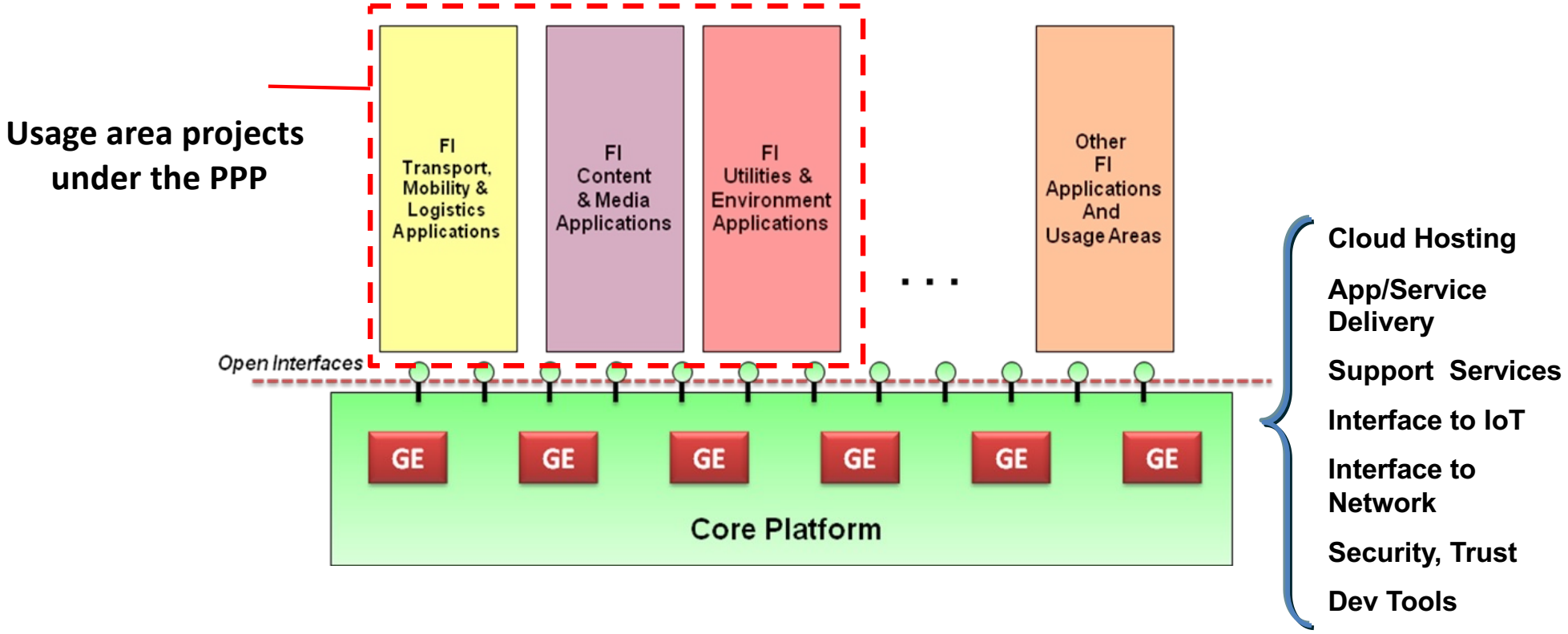
FIWARE is a curated framework of open source platform components which can be assembled together with other third-party platform components to a Smart Solutions.



It provides IoT Integrators with the ability of transforming devices specific Data Models into the Data Models defined at the NGSI level by different verticals (Smartcities, SmartAgrifood, Smartports, etc.). It also helps on the configuration, operation and monitoring of IoT end-nodes, IoT Gateways and IoT networks.

::.. FIWARE (2/4)

The FI Core Platform comprises a set of technological “Generic Enablers” which are considered general purpose and common to several current and future “usage areas”. Generic Enablers (therefore, the FI Core Platform) will provide open interfaces for development of Applications.



::.. FIWARE (3/4)

A device or IoT end-node migh use standard or proprietary communication protocols that are natively pushed to the Backend GEs or might be translated into any other standard or prpietary communication protocol at IoT Gateways. A particular case of communciation protocol is OMA NGSI. However, we do not expect devices and gateways to implement it and therefore FIWARE GEs provide a way to translate these protocols into NGSI.

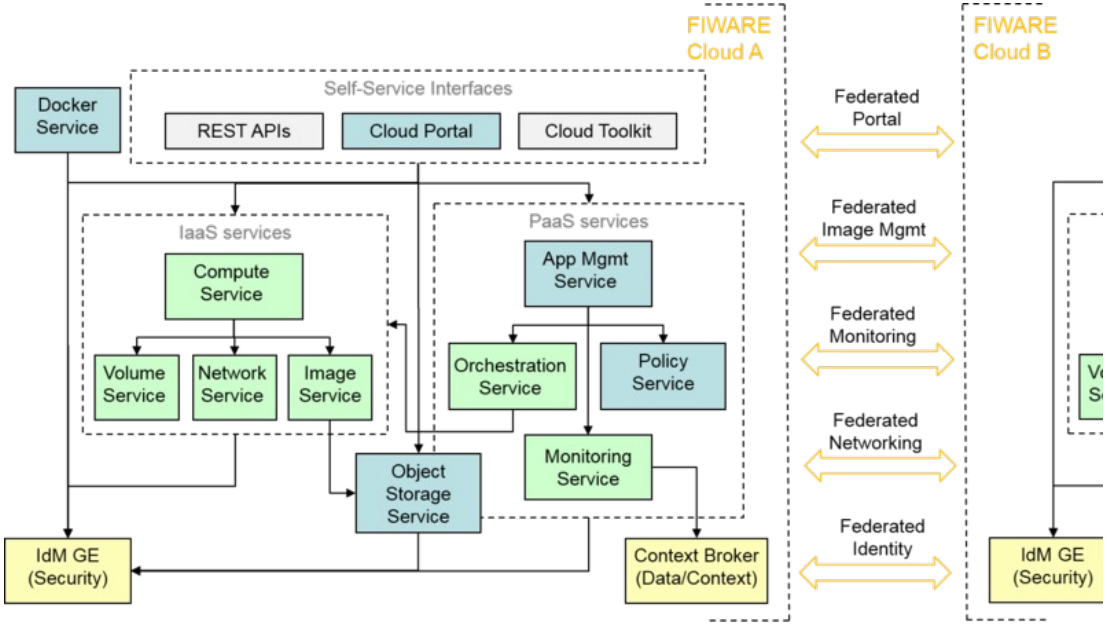
An IoT resource is a computational element providing access to sensor/actuator devices. An information model for the description of IoT resources can include context data like location, accuracy, status information, etcThe resource is usually hosted on the device but it has a logical representation in the backend as well.

A thing can be any object, person, or place in the real world represented as virtual things having an entity ID, a type and several attributes. Sensors can be modelled as virtual things, but other real-world things like rooms, persons, etc. can be modelled as virtual things as well.

::.. FIWARE (4/4)

FIWARE offers Generic Enablers that comprise the foundation for designing a modern cloud hosting infrastructure that can be used to develop, deploy and manage Future Internet applications and services.

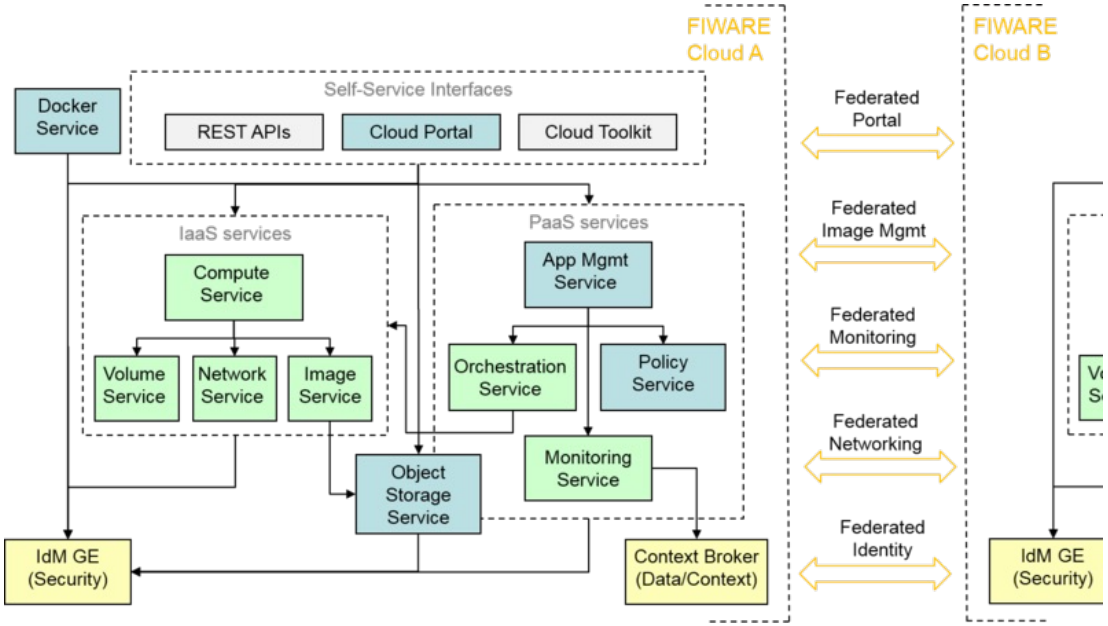
FIWARE Cloud architecture is heavily based on OpenStack, and provides and manages elements to compute, storage and network reosources.



::.. FIWARE (4/4)

FIWARE offers Generic Enablers that comprise the foundation for designing a modern cloud hosting infrastructure that can be used to develop, deploy and manage Future Internet applications and services.

FIWARE Cloud architecture is heavily based on OpenStack, and provides and manages elements to compute, storage and network reosources.

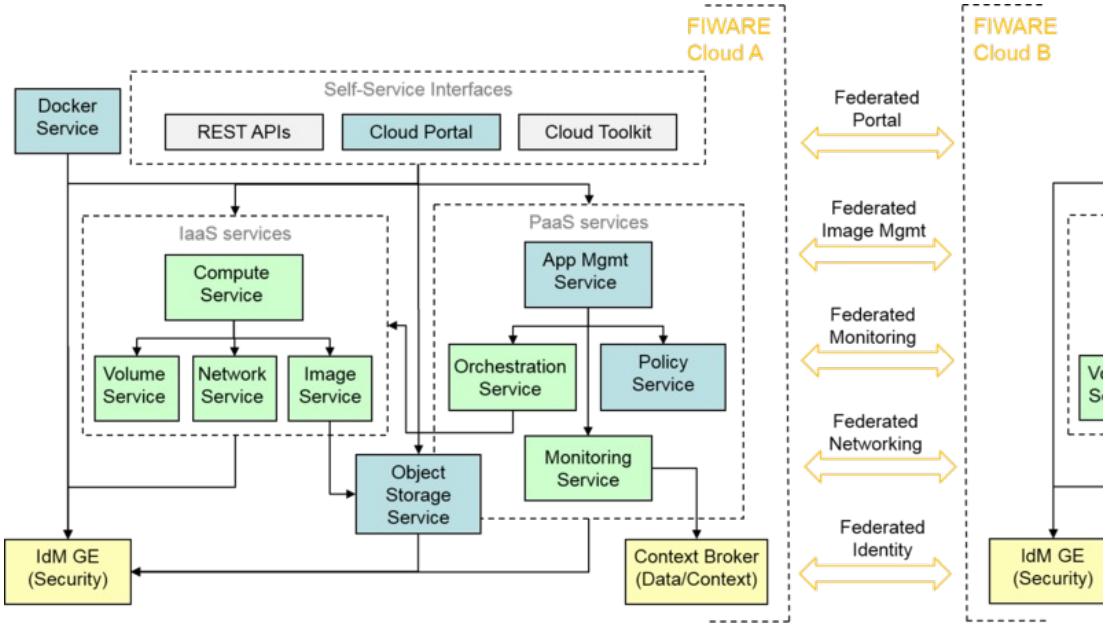


The components in Green indicate the native OpenStack services which are intended to be used in FIWARE Cloud as-is, without further modifications.

::.. FIWARE (4/4)

FIWARE offers Generic Enablers that comprise the foundation for designing a modern cloud hosting infrastructure that can be used to develop, deploy and manage Future Internet applications and services.

FIWARE Cloud architecture is heavily based on OpenStack, and provides and manages elements to compute, storage and network reosources.

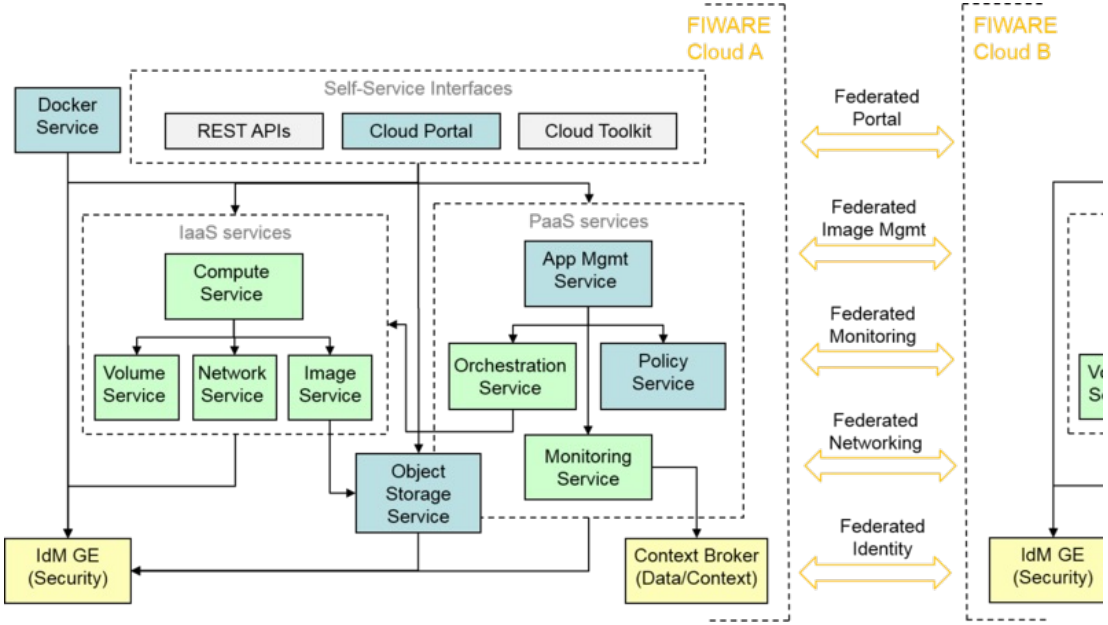


The Blue components indicate the GEs based on corresponding OpenStack services but with modifications, or are currently not based on OpenStack and.

::.. FIWARE (4/4)

FIWARE offers Generic Enablers that comprise the foundation for designing a modern cloud hosting infrastructure that can be used to develop, deploy and manage Future Internet applications and services.

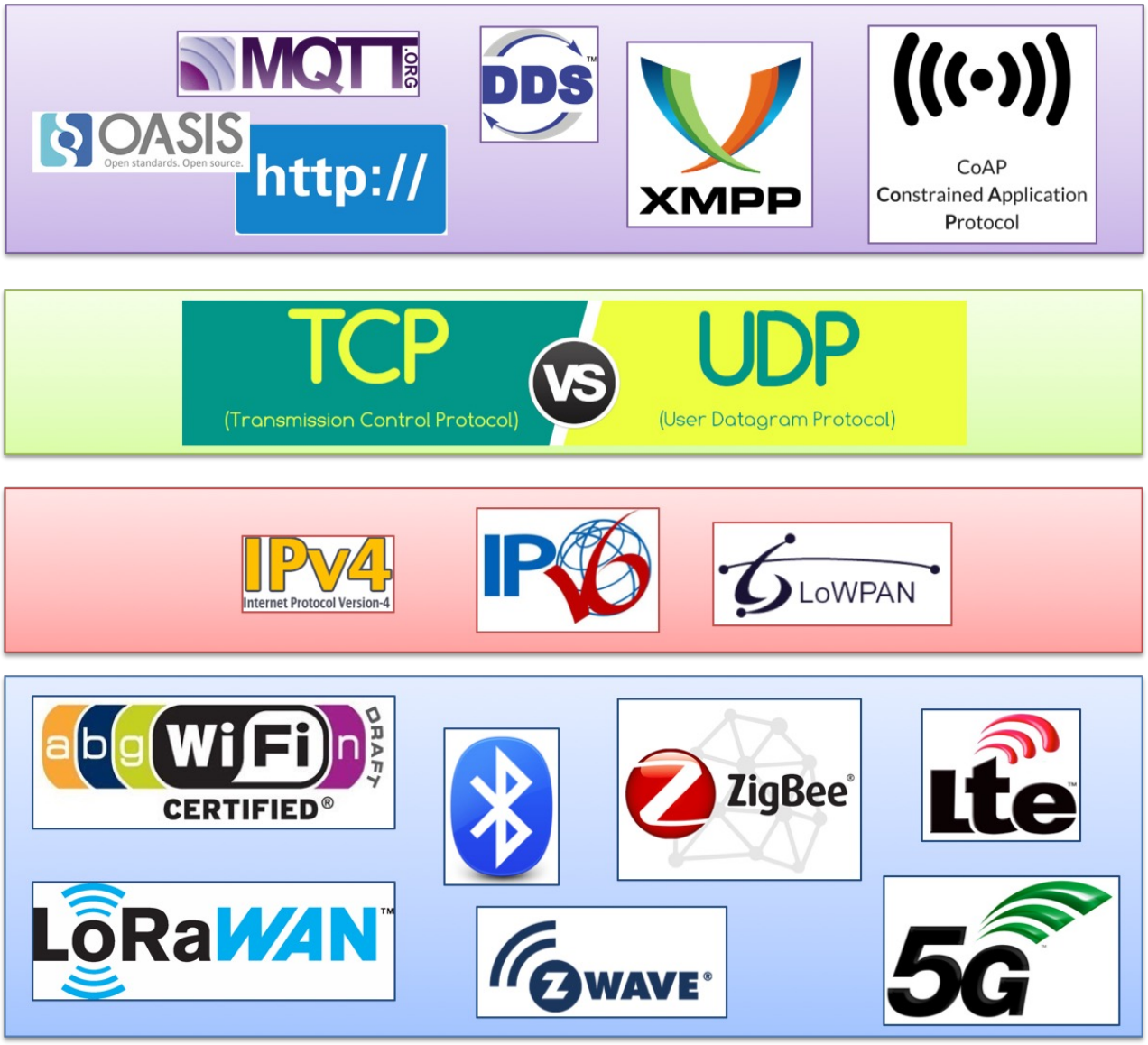
FIWARE Cloud architecture is heavily based on OpenStack, and provides and manages elements to compute, storage and network reosources.



Moreover, FIWARE Cloud architecture comprises the Docker GE, providing container hosting capabilities.

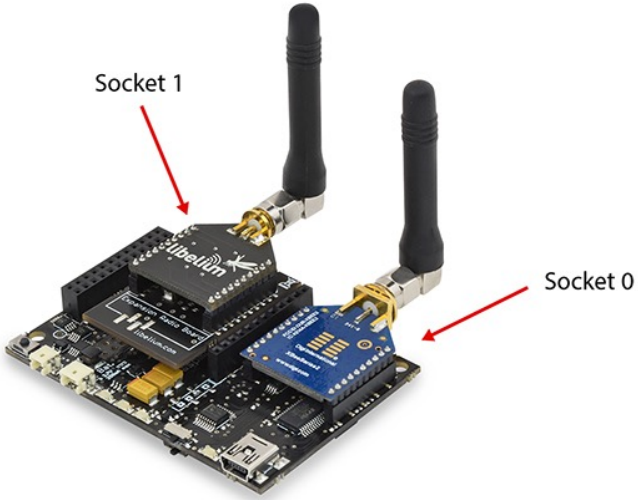
Communication Platform

::.. Communication Stack



::.. Wireless Technologies (1/6)

To maximize the opportunity for greater use and cost-effectiveness, it is necessary to use existing and/or emerging commercial wireless communications and infrastructures rather than having to develop a completely new apparatus designed specifically for sensor networks.



The Expansion Board allows the user to connect two communication modules at the same time in the Waspmote sensor platform.

A sensor node can use one or more wireless technologies.

The designer must not have a strong knowledge of the radio fundamentals of these technologies, but limit himself to knowing some quality features such as consumption, coverage range of signals, bandwidth, performance, safety and a few other factors.

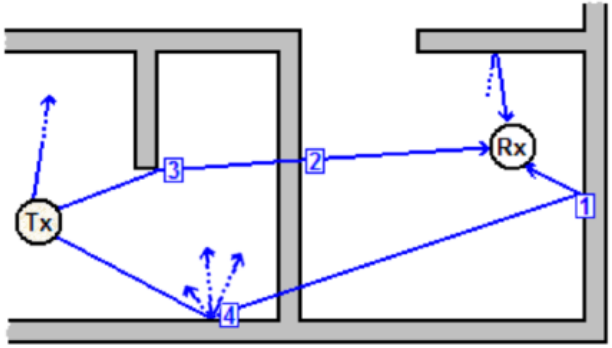
::.. Wireless Technologies (2/6)

Two frequency bands are typically used by the sensor networks:

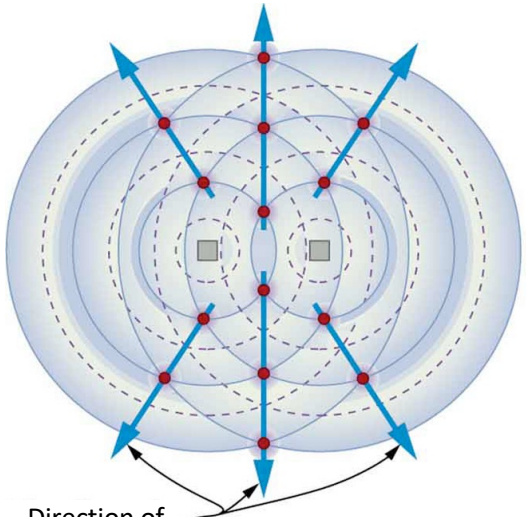
- The ISM bands, defined by the "Radiocommunication" sector of the ITU in the "Radio Regulations" 5.138 and 5.150. The ISM bands defined worldwide are: 900 MHz band (902-928 MHz); 2.4 GHz band (2.400-2.483.5 MHz); 5.8 GHz band (5.725-5.875 MHz). IEEE 802.11b / g and Bluetooth operate in the 2.4 GHz band, while IEEE 802.11a operates in the 5.8 GHz band.
- The UNII bands (Unlicensed National. Information Infrastructure), cover higher frequencies, in the ranges 5.15-5.35 GHz and 5.725-5.825 GHz, and allow higher speeds (up to 54 Mbit / second). They are used by 802.11a, much faster and more flexible than 802.11b, but with higher consumption.

::.. Wireless Technologies (3/6)

Interferences in indoor and outdoor environments may be due to both natural sources and/or phenomena (such as loss or attenuation, absorption, fading and multipath, but also by other users in the vicinity who use these unprotected bands. A WSN will experience interferences in its signals radio both if one of the IEEE PAN / LAN / MAN technologies p even if they use other generic radio technologies.



Multipath Channel - Phenomena such as reflection (1), refraction (2), diffraction (3) and scattering (4) cause interference of RF signals.



Direction of Constructive Interference

= Constructive Interference

::.. Wireless Technologies (4/6)

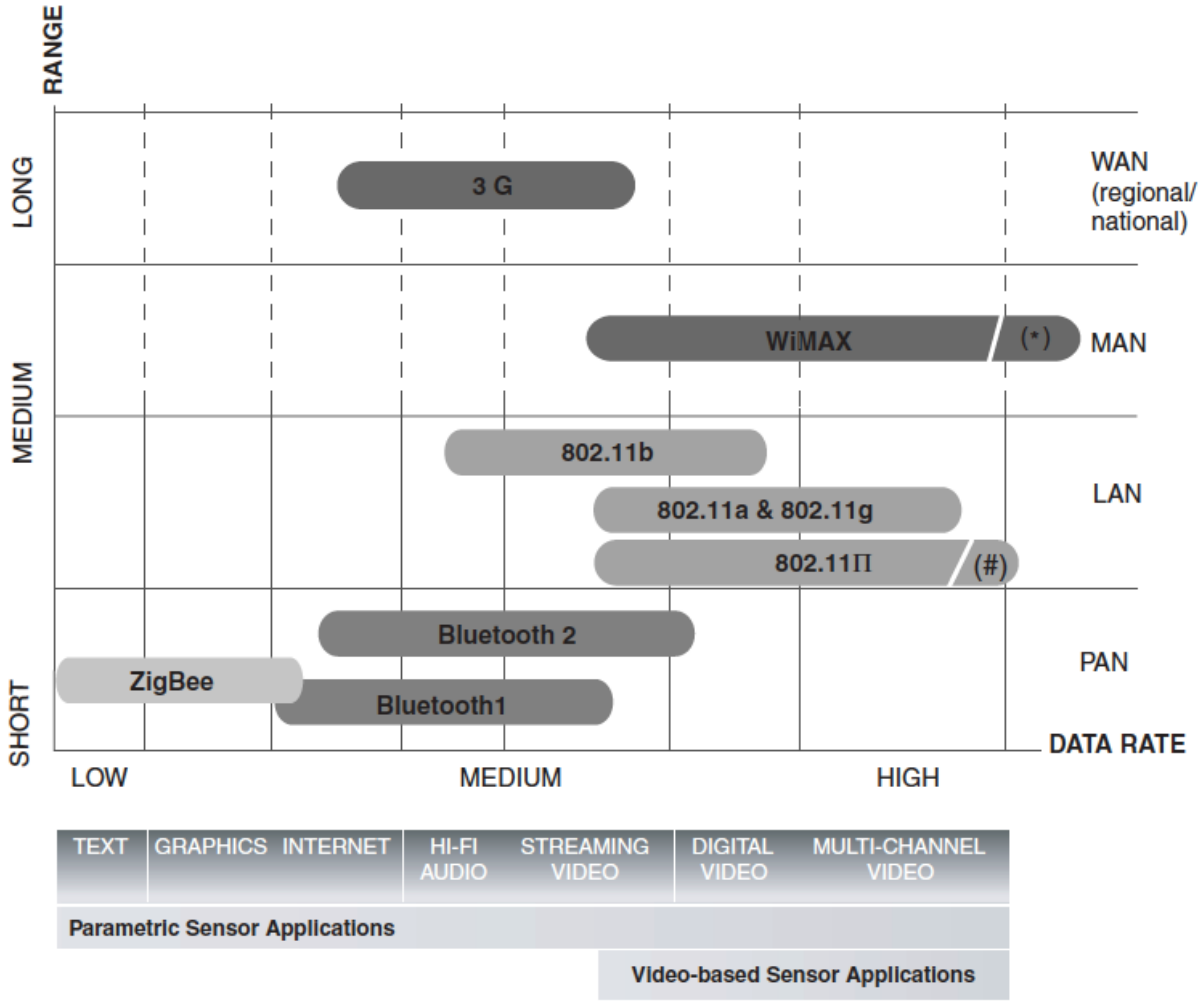
IEEE PAN / LAN / MAN technologies are widely implemented and probably the most used by the majority of WSNs (including commercial ones). The protocols determine (1) the physical coding of the transmitted signals, (2) the data link framing of the information, (3) the channel sharing and data management and event management procedures. There are various wireless technologies, the most used are the following:

1. IEEE 802.15.1 (also known as Bluetooth);
2. The IEEE 802.11a/b/g/n wireless LAN standards;
3. IEEE 802.15.4 (known as ZigBee);
4. The MAN standard known as IEEE 802.16, also known as WiMax;
5. Tagging radio-frequency identification (RFID).

Each standard has its benefits and its limitations.

::.. Wireless Technologies (5/6)

E.g., each standard has a range of coverage and a data rate:

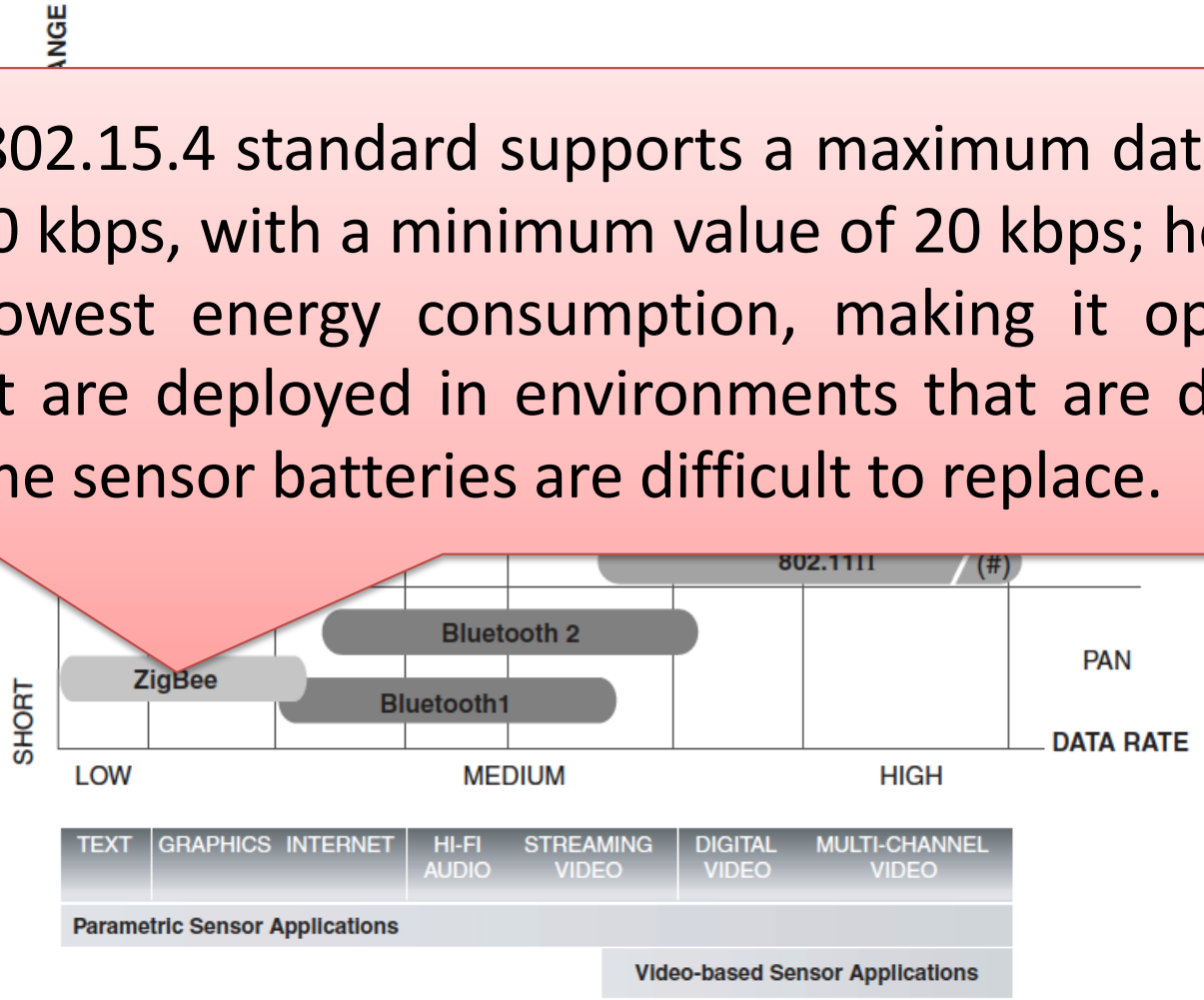


(*) Up to 268 Mbps
(#) Up to 108 Mbps

::.. Wireless Technologies (5/6)

E.g., each standard has a range of coverage and a data rate:

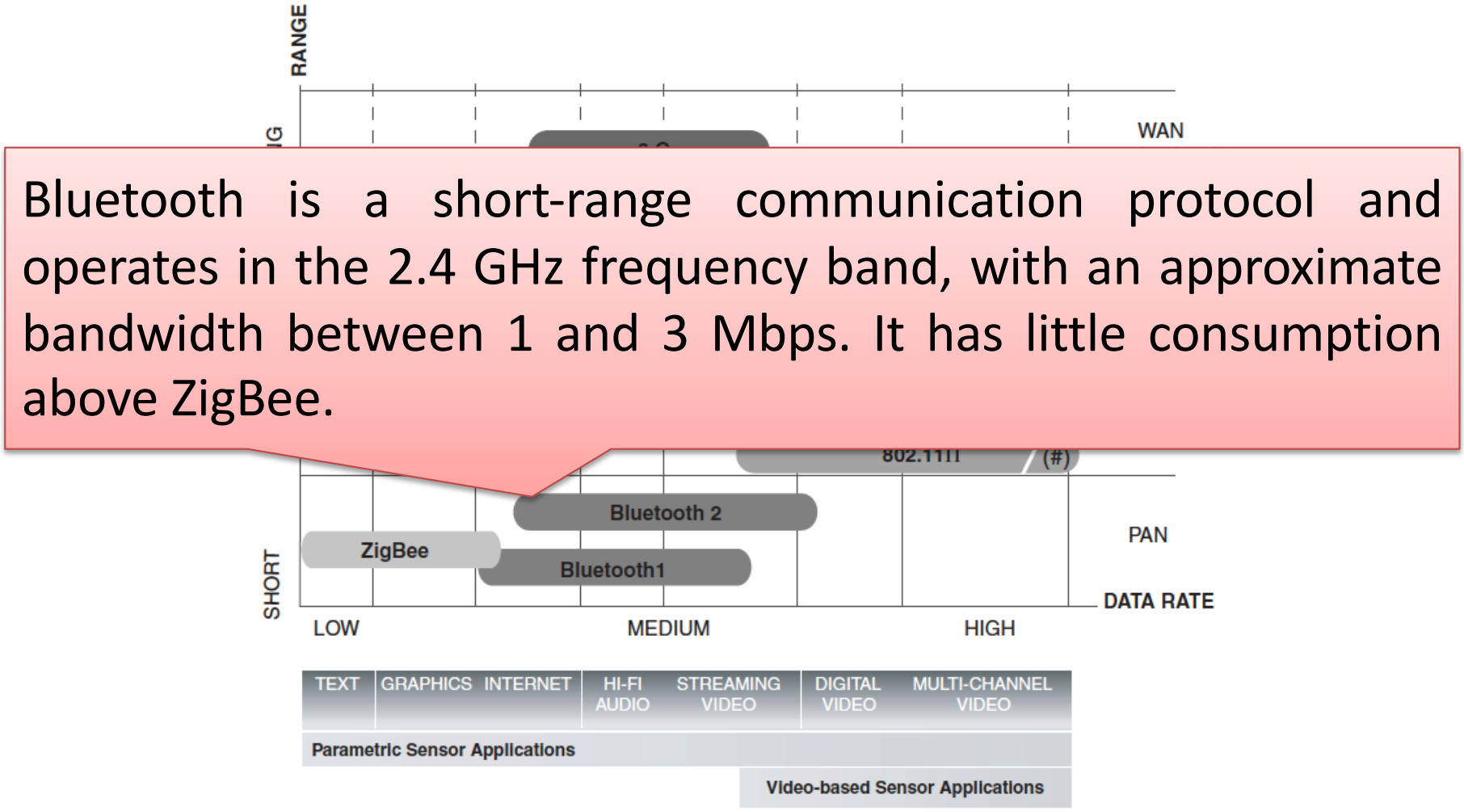
The IEEE 802.15.4 standard supports a maximum data transfer rate of 250 kbps, with a minimum value of 20 kbps; however, it has the lowest energy consumption, making it optimal for WSNs that are deployed in environments that are difficult to reach so the sensor batteries are difficult to replace.



(*) Up to 268 Mbps
(#) Up to 108 Mbps

::.. Wireless Technologies (5/6)

E.g., each standard has a range of coverage and a data rate:

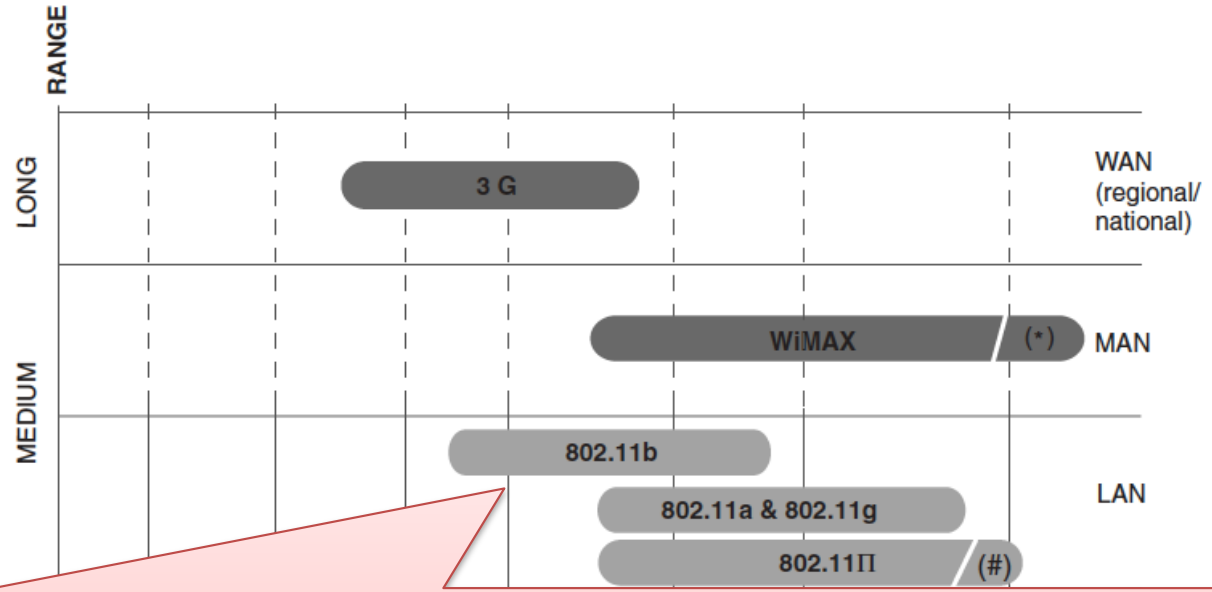


Bluetooth is a short-range communication protocol and operates in the 2.4 GHz frequency band, with an approximate bandwidth between 1 and 3 Mbps. It has little consumption above ZigBee.

(*) Up to 268 Mbps
(#) Up to 108 Mbps

::.. Wireless Technologies (5/6)

E.g., each standard has a range of coverage and a data rate:

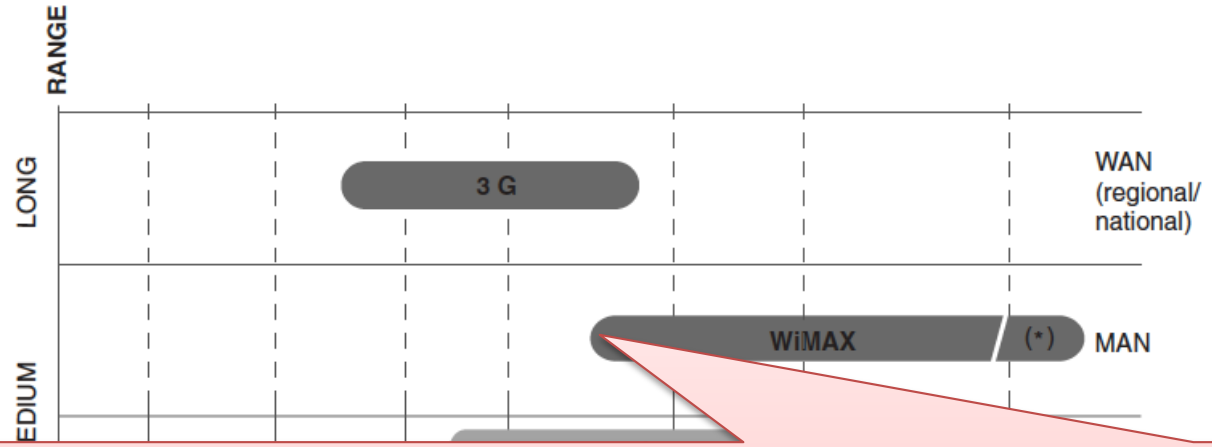


The 802.11 family of standards consists of three main standards operating in the 2.4 GHz, 5 GHz (ISM band) and 5 GHz (UNII band) frequencies respectively. It provides greater energy expenditure, but has a greater coverage range with a data rate of up to 54 Mbps (twice as high with the new IEEE 802.11n standard).

(#) Up to 108 Mbps

::.. Wireless Technologies (5/6)

E.g., each standard has a range of coverage and a data rate:

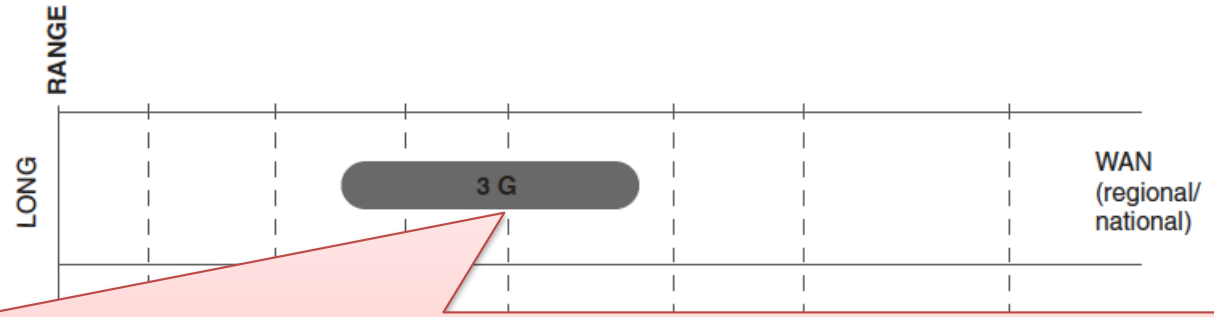


WiMAX has been designed to provide data transfer rates between 30 and 40 megabits per second. The original standard, called 802.16a, specifies the use of frequencies between 10 and 66 GHz, but 802.16d uses lower frequencies in the range from 2 to 11 GHz. The lower frequencies imply that the signals suffer less attenuation and therefore provide better covering in buildings.

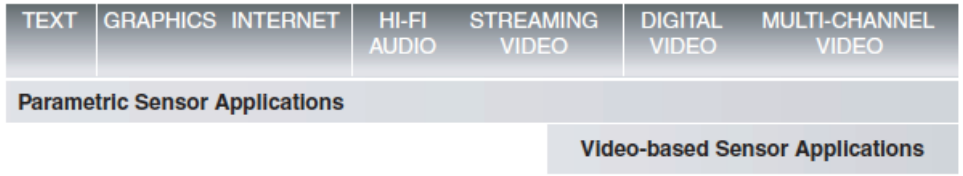
(#) Up to 108 Mbps

::.. Wireless Technologies (5/6)

E.g., each standard has a range of coverage and a data rate:



3G (symbol of 3rd Generation), in the field of cellular telephony, indicates the third generation technologies and standards. They provide a data transfer rate of at least 200 kbit / s. The signals are in a frequency between 1.8 and 2.5 GHz. Their use requires additional hardware.



(*) Up to 268 Mbps
(#) Up to 108 Mbps

::.. Wireless Technologies (6/6)

RFID is a technology for the identification and / or automatic storage of data relating to objects, animals or people based on the capacity of storing data by particular electronic labels, called tags. Tags can respond to remote interrogation by special devices called readers.

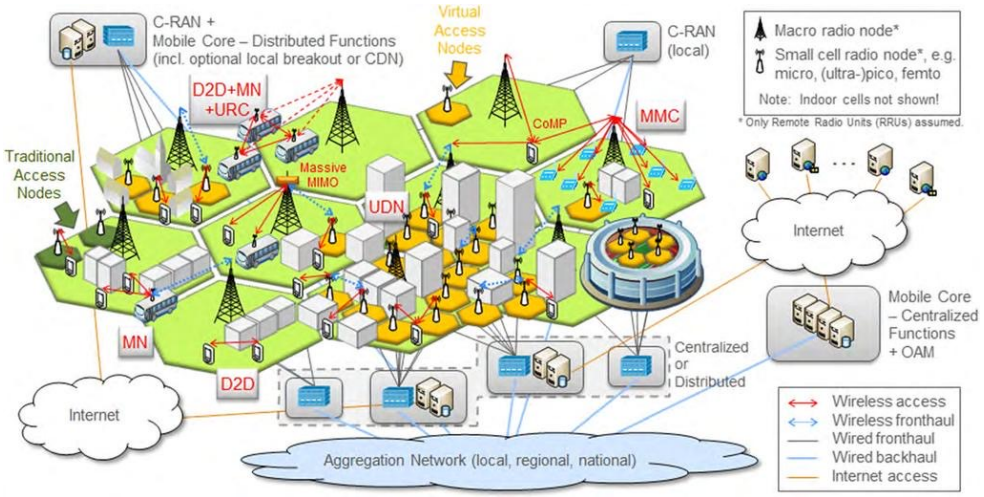
RFID devices can be assimilated to wireless reading and / or writing systems with various applications. In recent years, the NFC standard (Near Field Communication, 13.56 MHz and up to 10 cm, but with data transmission speeds up to 424 kBit / s) is also emerging, extending the standards to allow the exchange of information also between readers.

::.. 5G (1/3)

With 5G still in its early stages of implementation and not yet available in every country, you might be hearing about the 5G bandwidth spectrum, 5G spectrum auctions, mmW 5G, etc.

Higher Frequency	Lower Frequency
• Faster speeds	• Slower speeds
• Shorter distances	• Longer distances

The millimeter waves, which are in the high-band spectrum, have the advantage of being able to carry lots of data. However, they are also absorbed more easily by gases in the air, trees, and nearby buildings. mmWaves are therefore useful in densely packed networks, but not so helpful for carrying data long distances (due to the attenuation).



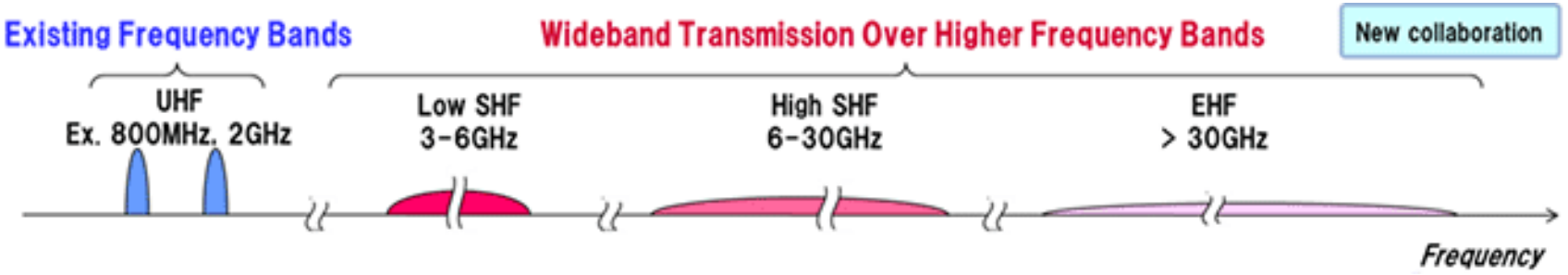
::.. 5G (2/3)

In 5G spectrum, different parts of the spectrum can be used to maximize distance, minimize problems, and get as much throughput as possible.

- A frequency of 600 MHz, for example, has lower bandwidth, but because it’s not affected as easily by things like moisture in the air, it doesn't lose power as quickly and is able to reach 5G phones and other 5G devices further away, as well as better penetrate walls to provide indoor reception.

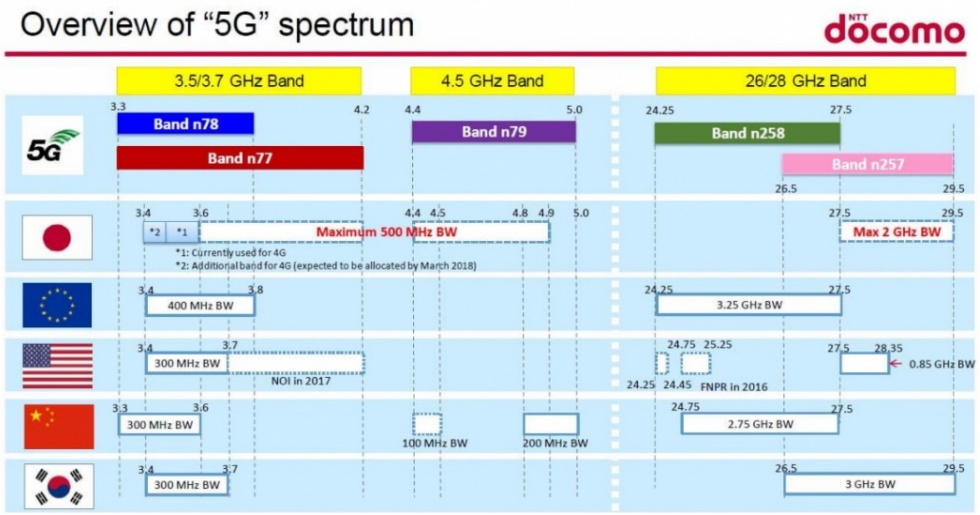
A service provider might use higher 5G frequencies in areas that demand more data, like in a popular city where there are lots of devices in use. However, low-band frequencies are useful for providing 5G access to more devices from a single tower and to areas that don't have direct line-of-sight to a 5G cell.

::.. 5G (3/3)

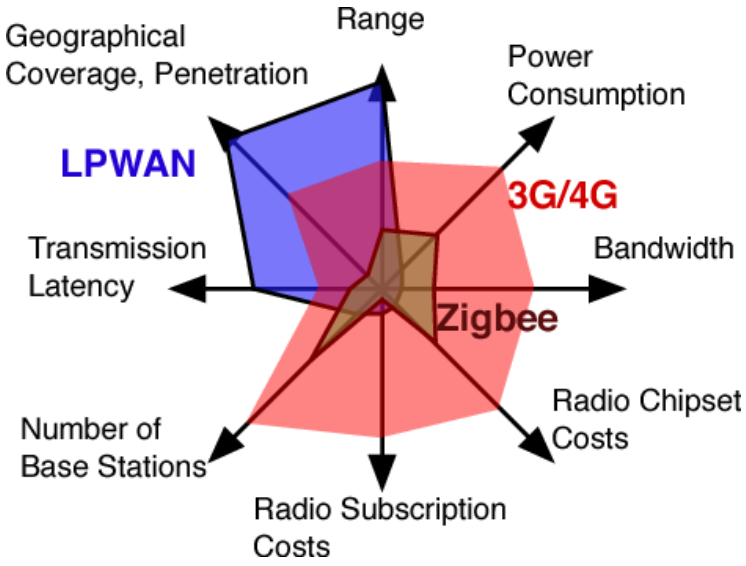


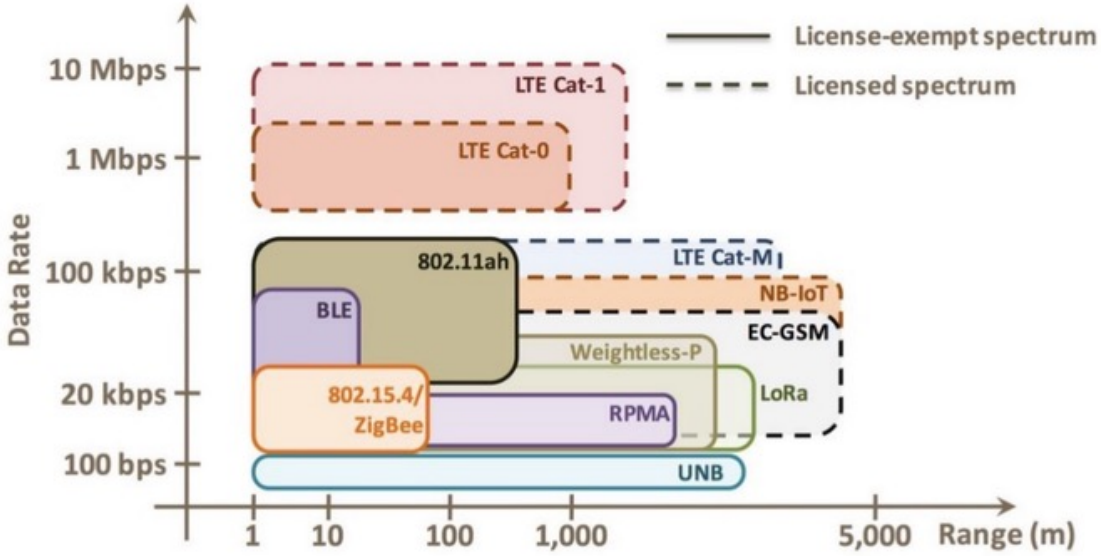
Here are some other 5G frequency ranges :

- C-band: 2–6 GHz for coverage and capacity;
- Super Data Layer: Over 6 GHz for high bandwidth;
- Coverage Area: Below 2 GHz for indoor and broader coverage.



::.. Low-Power Wide-Area Network (1/3)



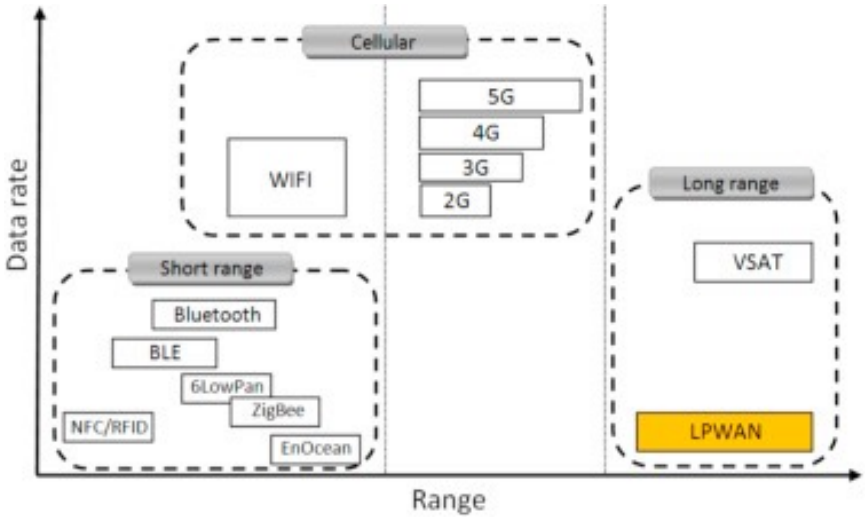


LPWAN is characterized by a low power consumption and a rather long range. It mainly use the sub-GHz band to achieve high noise robustness, but it offers low data rates.

LPWAN networks offer long ranges between 5 and 10 km at the expense of a smaller bandwidth.

This is related to their high noise robustness and their main frequency. This leads to poor data rates of several kbps.

::.. Low-Power Wide-Area Network (2/3)

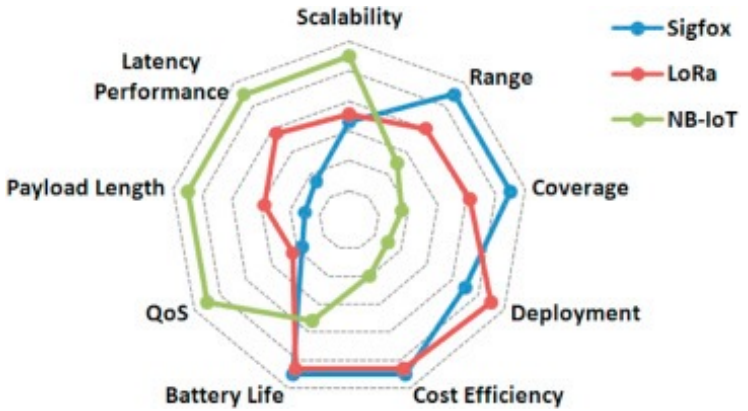


LPWAN is highly suitable for IoT applications that only need to transmit tiny amounts of data in long range. It is highly energy efficient and in-expensive, with the cost of a radio chipset being less than 2€ and an operating cost of 1€ per device per year.

	Local Area Network Short Range Communication	**Low Power Wide Area** (LPWAN) Internet of Things	**Cellular Network** Traditional M2M
	40%	**45%**	**15%**
☺	Well established standards In building	Low power consumption Low cost Positioning	Existing coverage High data rate
☹	Battery Live Provisioning Network cost & dependencies	High data rate Emerging standards	Autonomy Total cost of ownership
	Bluetooth 4.0 · ZigBee · Wi Fi	sigfox · LoRa · NB-IoT	GSM · 3G+ / H+ · 4G

::.. Low-Power Wide-Area Network (3/3)

	Sigfox	LoRaWAN	NB-IoT
Modulation	BPSK	CSS	QPSK
Frequency	Unlicensed ISM bands (868 MHz in Europe, 915 MHz in North America, and 433 MHz in Asia)	Unlicensed ISM bands (868 MHz in Europe, 915 MHz in North America, and 433 MHz in Asia)	Licensed LTE frequency bands
Bandwidth	100 Hz	250 kHz and 125 kHz	200 kHz
Maximum data rate	100 bps	50 kbps	200 kbps
Bidirectional	Limited / Half-duplex	Yes / Half-duplex	Yes / Half-duplex
Maximum messages/day	140 (UL), 4 (DL)	Unlimited	Unlimited
Maximum payload length	12 bytes (UL), 8 bytes (DL)	243 bytes	1600 bytes
Range	10 km (urban), 40 km (rural)	5 km (urban), 20 km (rural)	1 km (urban), 10 km (rural)
Interference immunity	Very high	Very high	Low
Authentication & encryption	Not supported	Yes (AES 128b)	Yes (LTE encryption)
Adaptive data rate	No	Yes	No
Handover	End-devices do not join a single base station	End-devices do not join a single base station	End-devices join a single base station
Localization	Yes (RSSI)	Yes (TDOA)	No (under specification)
Allow private network	No	Yes	No
Standardization	Sigfox company is collaborating with ETSI on the standardization of Sigfox-based network	LoRa-Alliance	3GPP



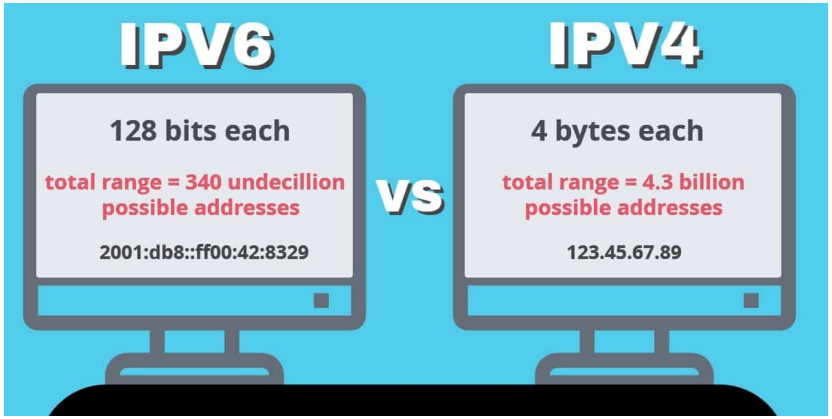
Sigfox and LoRa will serve with very long range, infrequent communications, and very long battery lifetime.

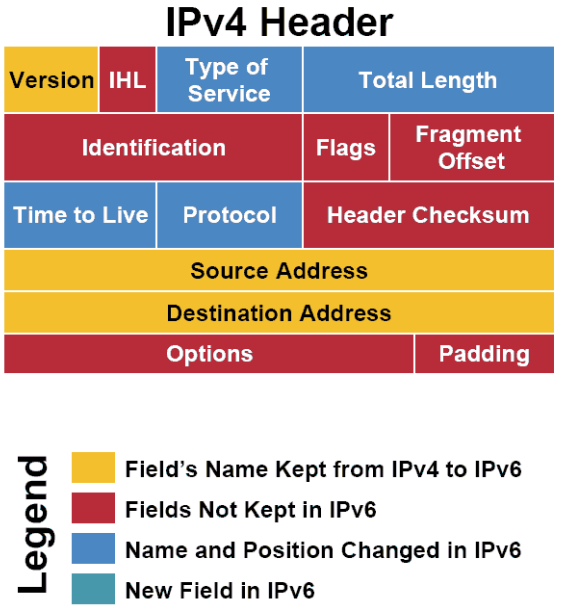
LoRa will also provide reliable communications when devices move at high speeds. By contrast, NB-IoT offers very low latency and high quality of service.

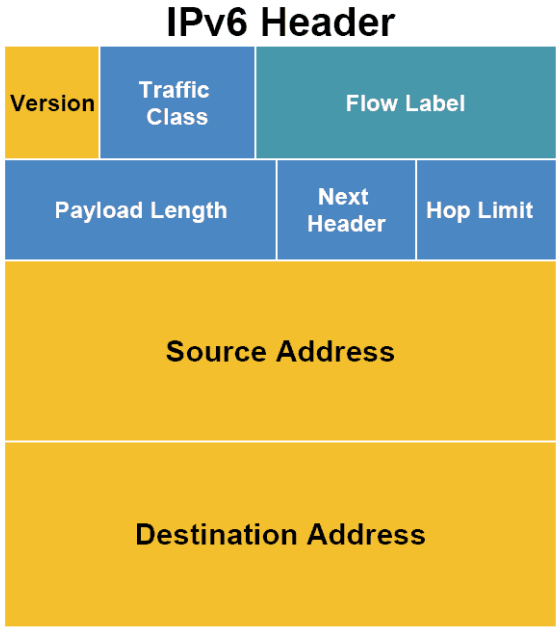
::.. IPv6 For IoT (1/3)

Internet Protocol Version 6 (**IPv6**) is the enhanced version of IPv4 and can support very large numbers of nodes as compared to IPv4. It allows for 2128 possible node, or address, combinations.

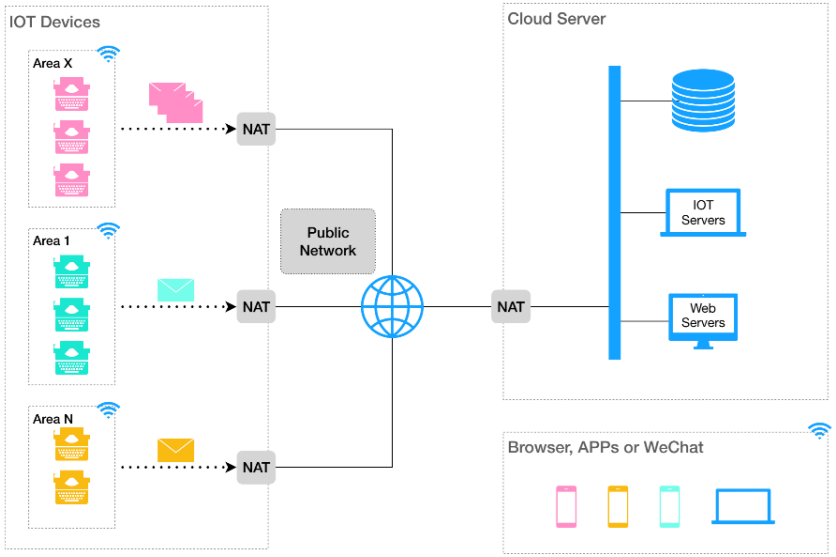
It simplifies some network functions, such as routing and mobility, and offers better security options thanks to better design and better calibrated management than IPSec, the security component of the IPv4 protocol.







::.. IPv6 For IoT (2/3)



Due to the limits of the IPv4 address space, the current Internet uses the Network Address Translation (NAT). This solution is working but with two main trades-off:

- The NAT users are borrowing and sharing IP addresses with others. Hence, they do not have their own public IP address, which turns them into homeless Internet users. They can access the Internet, but they cannot be directly accessed from the Internet.
- It breaks the original end-to-end connection and dramatically weakens any authentication process.

::.. IPv6 For IoT (3/3)

The IoT research community has developed a compressed version of IPv6 named 6LoWPAN, with a simple and efficient mechanism to shorten the IPv6 address size for constrained devices, while border routers can translate those compressed addresses into regular IPv6 addresses. In parallel, tiny stacks have been developed, such as Contiki, which takes no more than 11.5 Kbyte.

Thanks to its large address space, IPv6 enables the extension of the Internet to any device and service. it provides an address self-configuration mechanism (Stateless mechanism). The nodes can define their addresses in very autonomous manner. This enables to reduce drastically the configuration effort and cost.

::.. Routing (1/4)

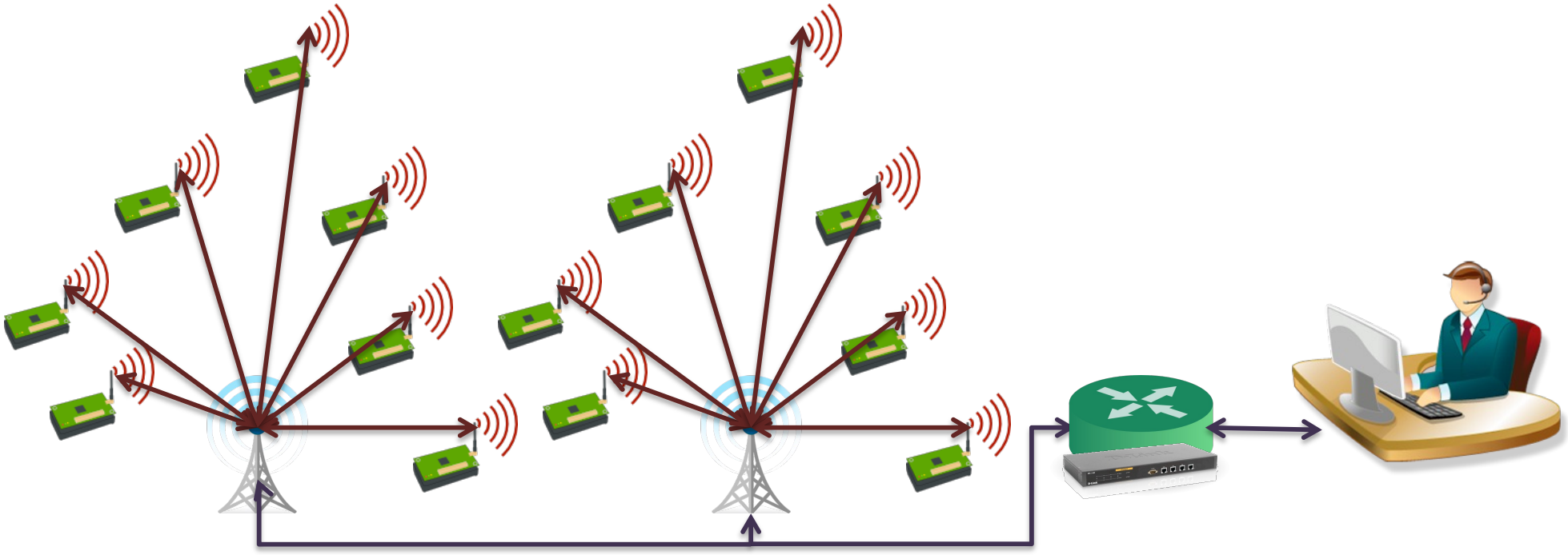
Sensor networks are extremely versatile and can be deployed to support a wide range of applications. Sensors are deployed depending on the application in an ad-hoc manner, with little or no maintenance.

Sensors have to transmit measurements and inferred data by means of a routing protocol with efficient energy consumption, so as to determine the paths between the sensor nodes and towards the intended destination.

Route selection must be made in such a way that the network life cycle is maximized. The characteristics of the environment in which the sensors operate, coupled with strong limitations on resources and power supply, makes the routing problem very complicated.

::.. Routing (2/4)

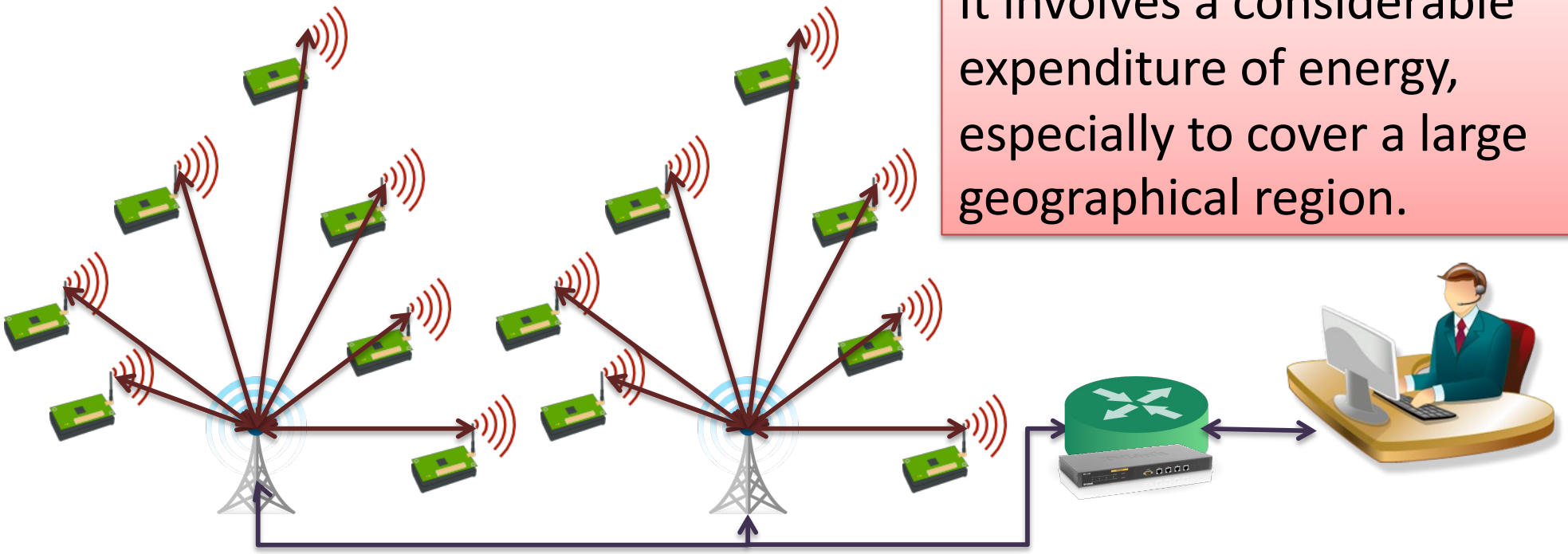
The way data and queries are forwarded between the base station and the location where the target phenomena are observed is an important aspect and a basic feature of the WSNs. A simple approach to perform this task is called single-hop and allows each sensor node to exchange messages directly with the base station.



::.. Routing (2/4)

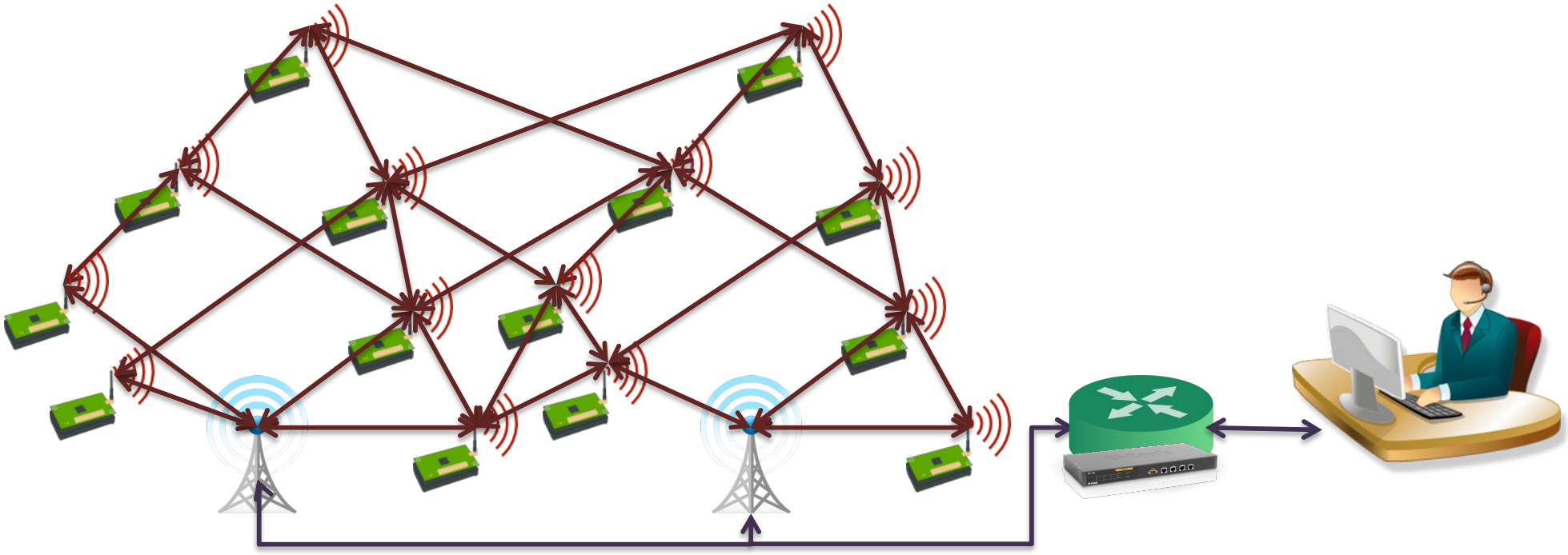
The way data and queries are forwarded between the base station and the location where the target phenomena are observed is an important aspect and a basic feature of the WSNs. A simple approach to perform this task is called single-hop and allows each sensor node to exchange messages directly with the base station.

It involves a considerable expenditure of energy, especially to cover a large geographical region.



::.. Routing (3/4)

To solve the limitations of the single-hop solution, data exchanges are usually made using a multi-hop transmission by using smaller communication range.

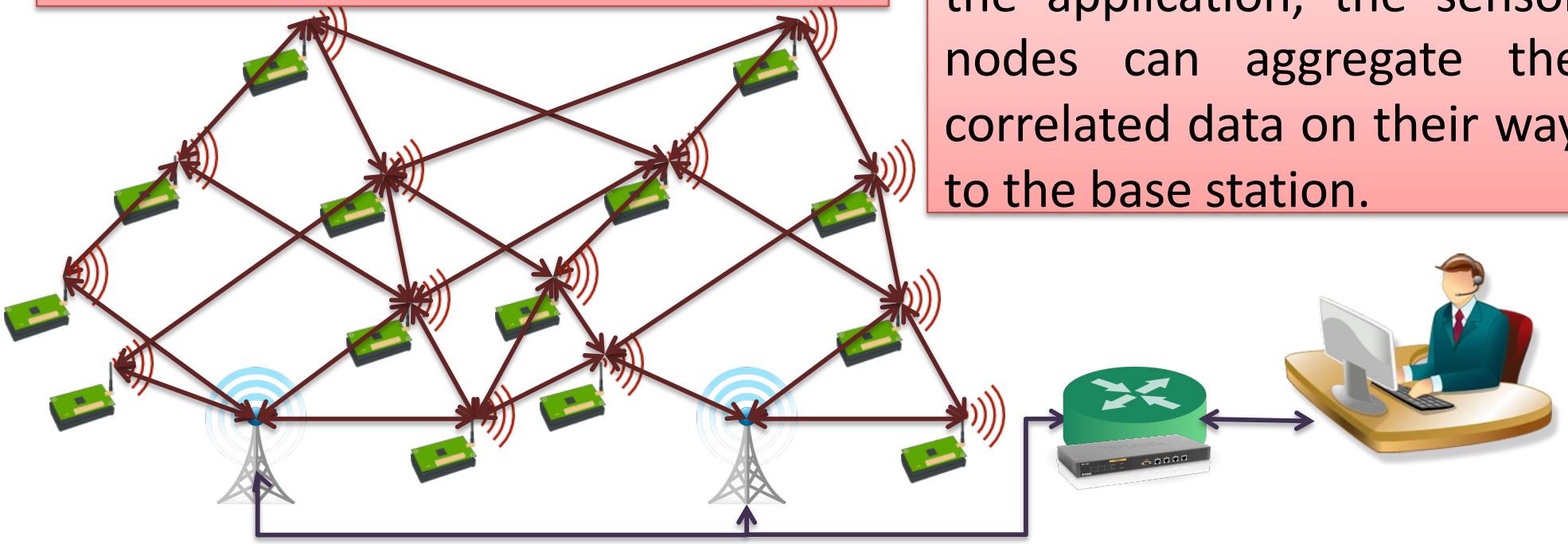


::.. Routing (3/4)

To solve the limitations of the single-hop solution, data exchanges are usually made using a multi-hop transmission by using smaller communication range.

It involves significant energy savings, and reduced interference between nodes.

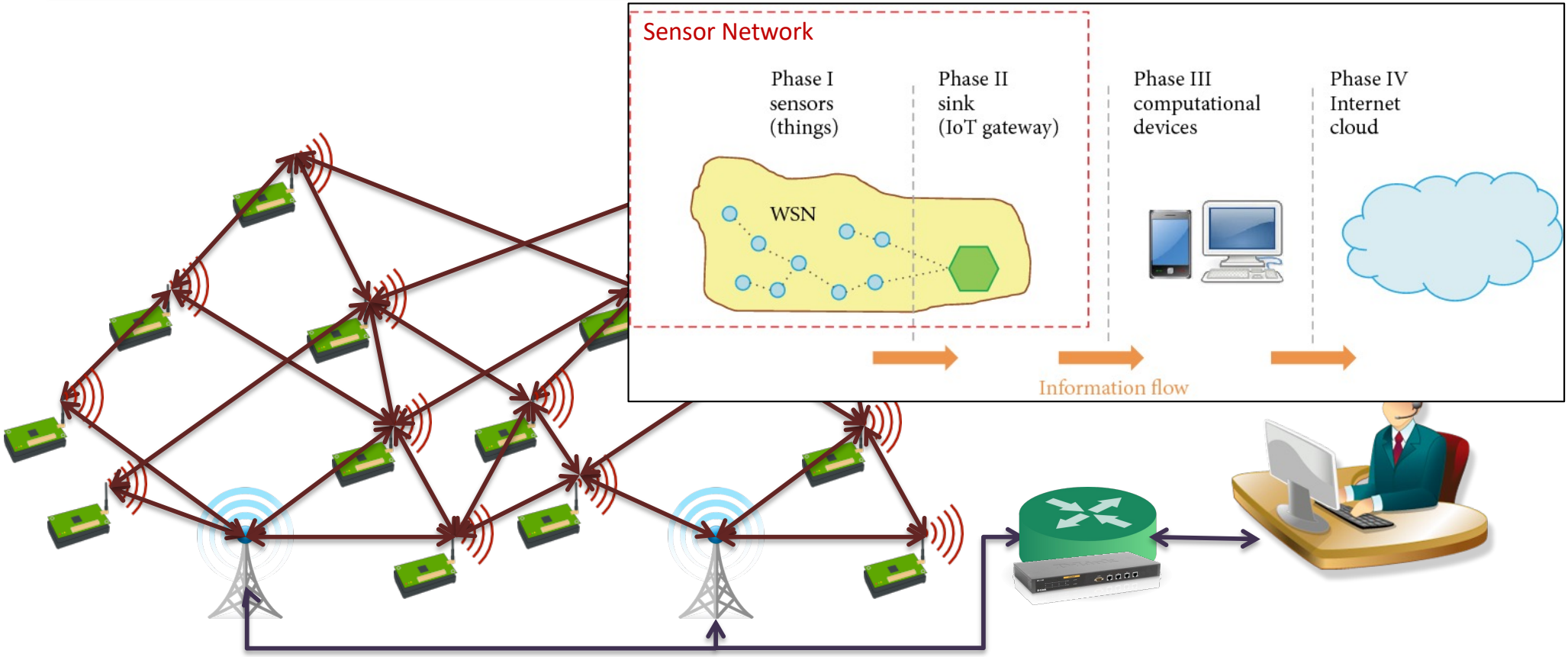
It should be noted that depending on the nature of the application, the sensor nodes can aggregate the correlated data on their way to the base station.



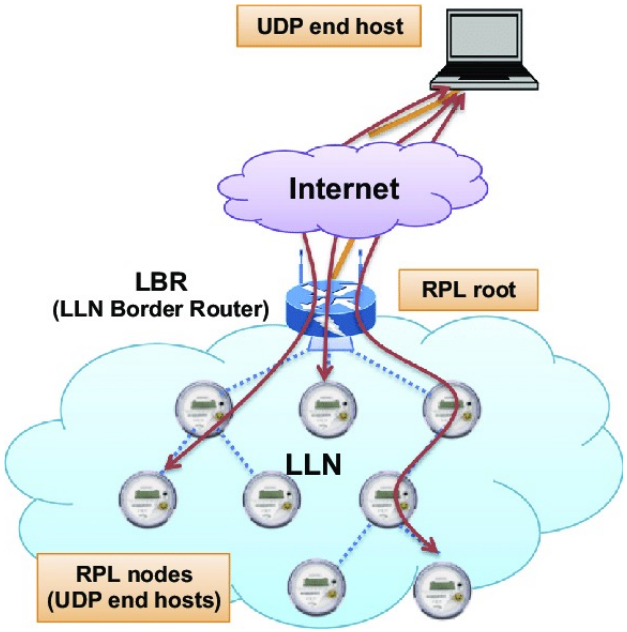
::.. Routing (3/4)

To solve the limitations of the single-hop solution, data transmission by

Also, IoT applies similar routing strategies where gateways collect sensory data and pass them over the Internet to the cloud or Edge/Fog nodes.

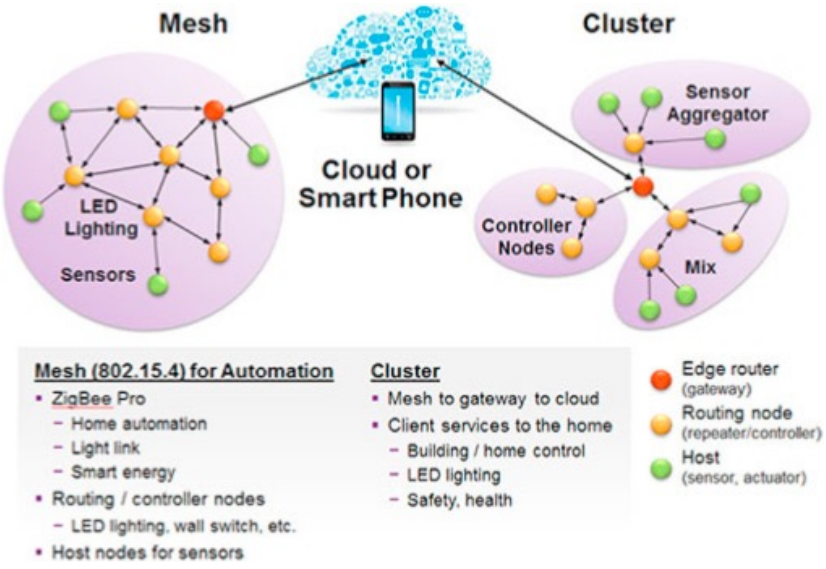


::.. Routing (4/4)



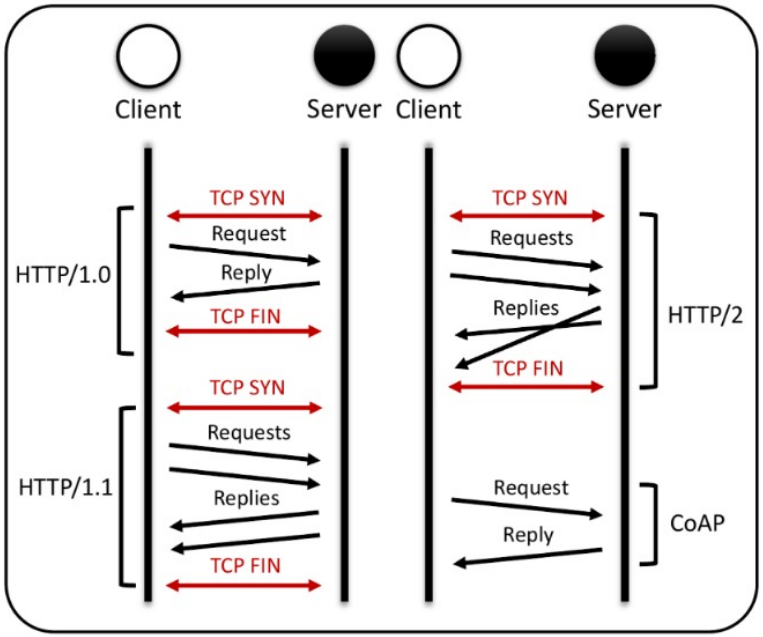
Multi-hop routing implies intermediate nodes to honestly participate in the packet forwarding. Determining which set of intermediate nodes must be selected to form the packet forwarding path is a crucial task but also a difficult problem.

Nodes can be organised as a mesh where nodes connect directly, dynamically and non-hierarchically to as many other nodes as possible. Mesh networks dynamically self-organize and self-configure. Alternativelly, they can be structured in clusters.



::.. Communication Protocols (1/2)

The candidate communication protocols differ in their interaction models.

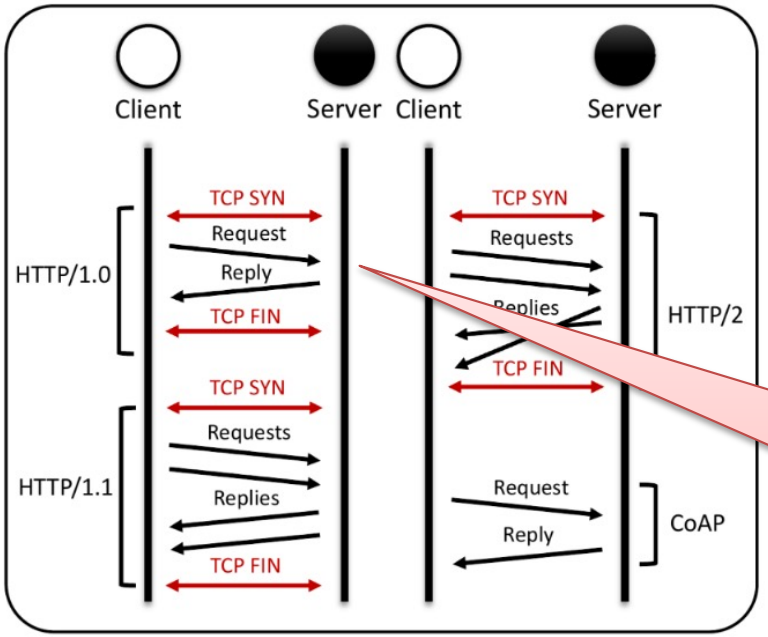


The request-reply communication model is one of the most basic communication paradigms: a message exchange pattern where a client requests information from a server that receives the request message, processes it, and returns a response message.

This kind of information is usually managed and exchanged centrally, and the two most known protocols based on the request/reply model are REST HTTP and CoAP.

::.. Communication Protocols (1/2)

The candidate communication protocols differ in their interaction models.



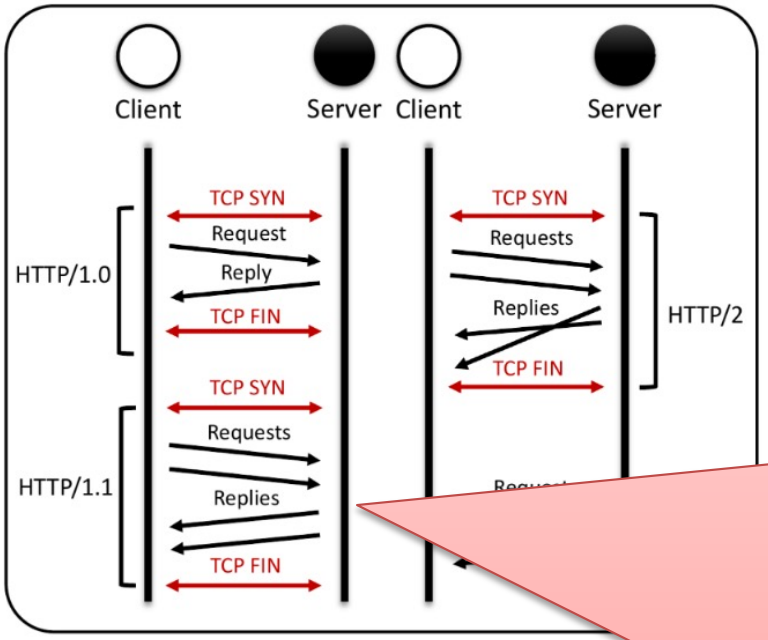
The request-reply communication model is one of the most basic communication paradigms: a message exchange pattern where a client requests information from a server that receives the request message, processes it, and returns a response message.

In HTTP 1.0, the TCP connection is closed after a single HTTP request/reply pair.

are REST HTTP and CoAP.

::.. Communication Protocols (1/2)

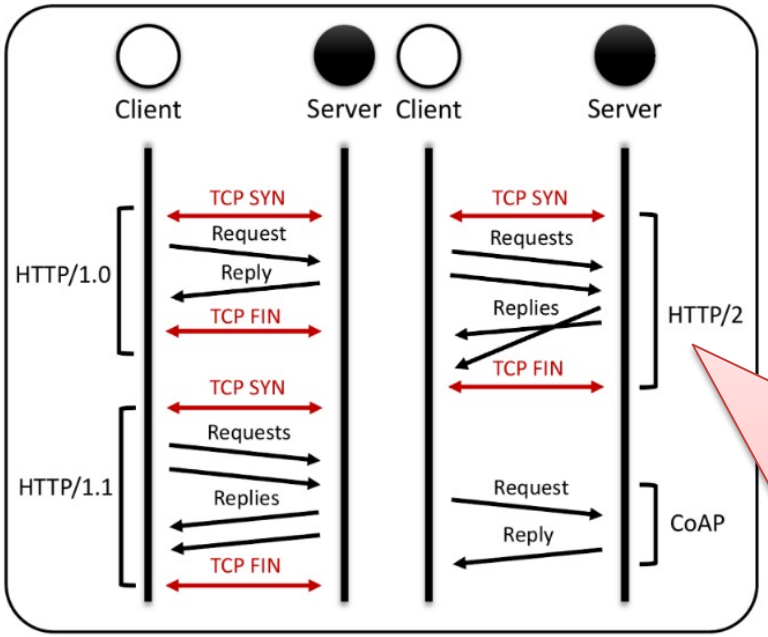
The candidate communication protocols differ in their interaction models.



In HTTP 1.1, a keep-alive-mechanism was introduced, where a TCP connection could be reused for sending multiple requests to the server without waiting for a response (pipelining). Once the requests are all sent, the browser starts listening for responses and HTTP 1.1 specification requires that a server must send its responses to those requests in the same order that the requests were received.

::.. Communication Protocols (1/2)

The candidate communication protocols differ in their interaction models.

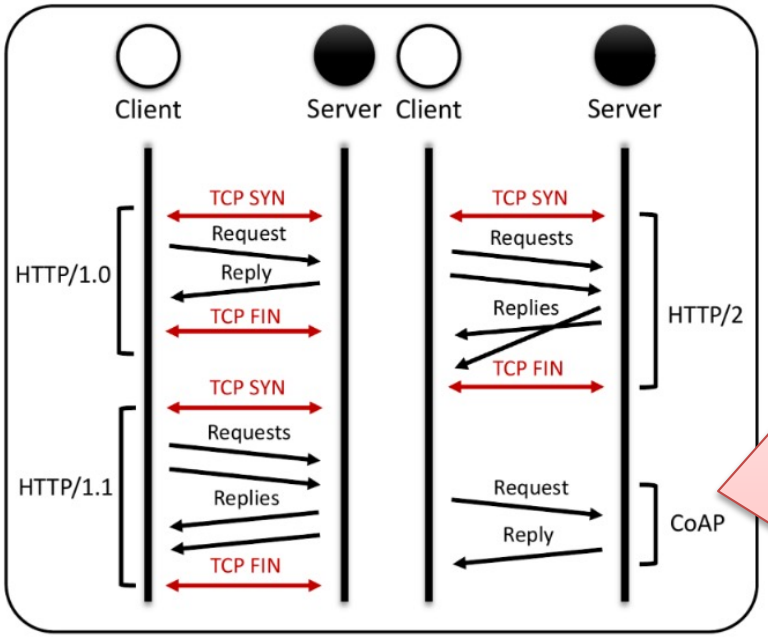


The request-reply communication model is one of the most basic communication paradigms: a message exchange pattern where a client requests information from a server that receives the request

The new HTTP 2.0 introduces a multiplexing method by which multiple HTTP requests can be sent and responses can be received asynchronously via a single TCP connection.

::.. Communication Protocols (1/2)

The candidate communication protocols differ in their interaction models.



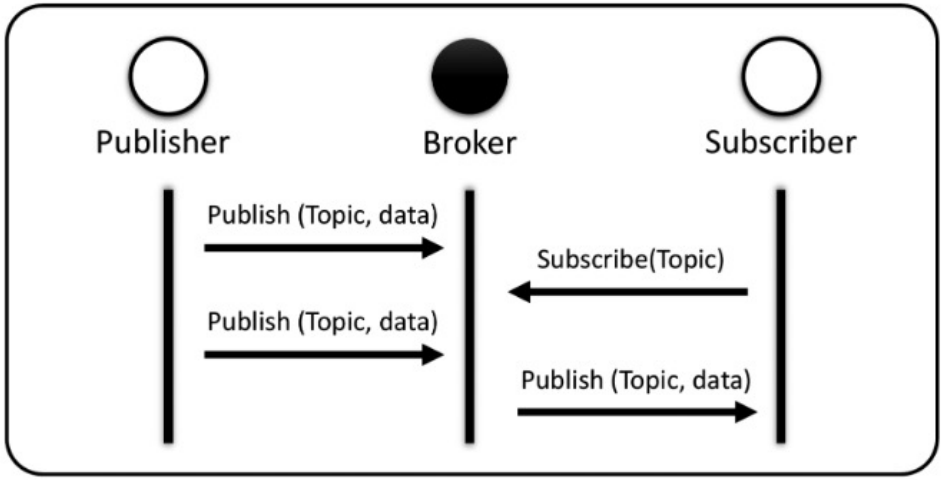
The request-reply communication model is one of the most basic communication paradigms: a message exchange pattern where a client requests information from a server that receives the request message, processes it, and returns a

The fourth interaction shown is for CoAP,and unlike the others it does not depend on an underlying reliable TCP connection to exchange request/reply messages between the client and the server.

::.. Communication Protocols (2/2)

The candidate communication protocols differ in their interaction models.

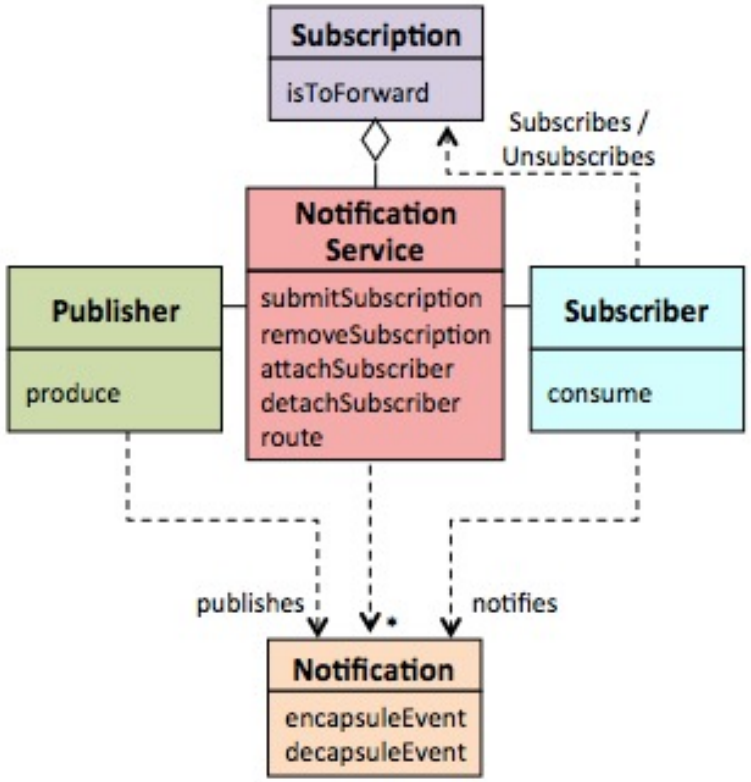
The publish-subscribe model, on the other hand, emerged out of the need to provide a distributed, asynchronous, loosely coupled communication between data generators and destinations. Numerous publish-subscribe Message-Oriented Middlewares (MoM) have recently been a subject of numerous research efforts and concrete deployments.

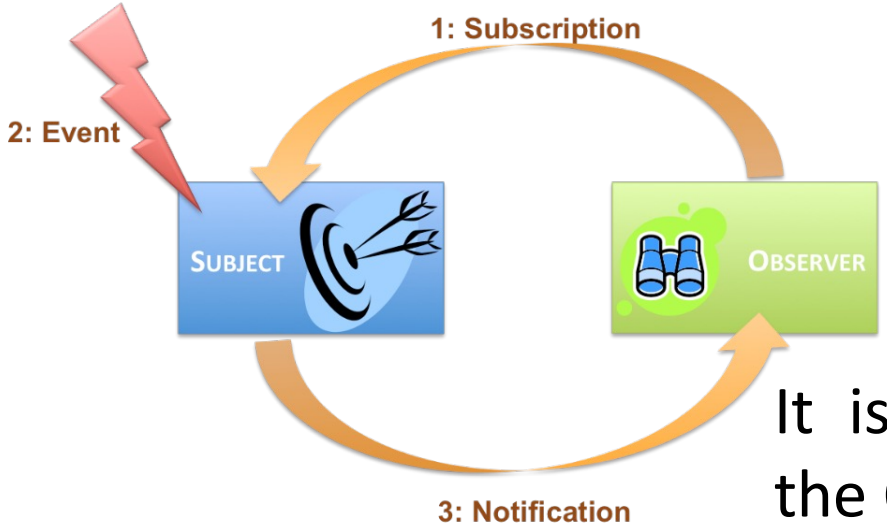


::.. Publish/Subscribe Services (1/6)

The Notification Service is key element that glues together publishers and subscribers and acts as a mediator. Its duties are

- Storing subscriptions;
- Buffering published events;
- Distributing notifications.

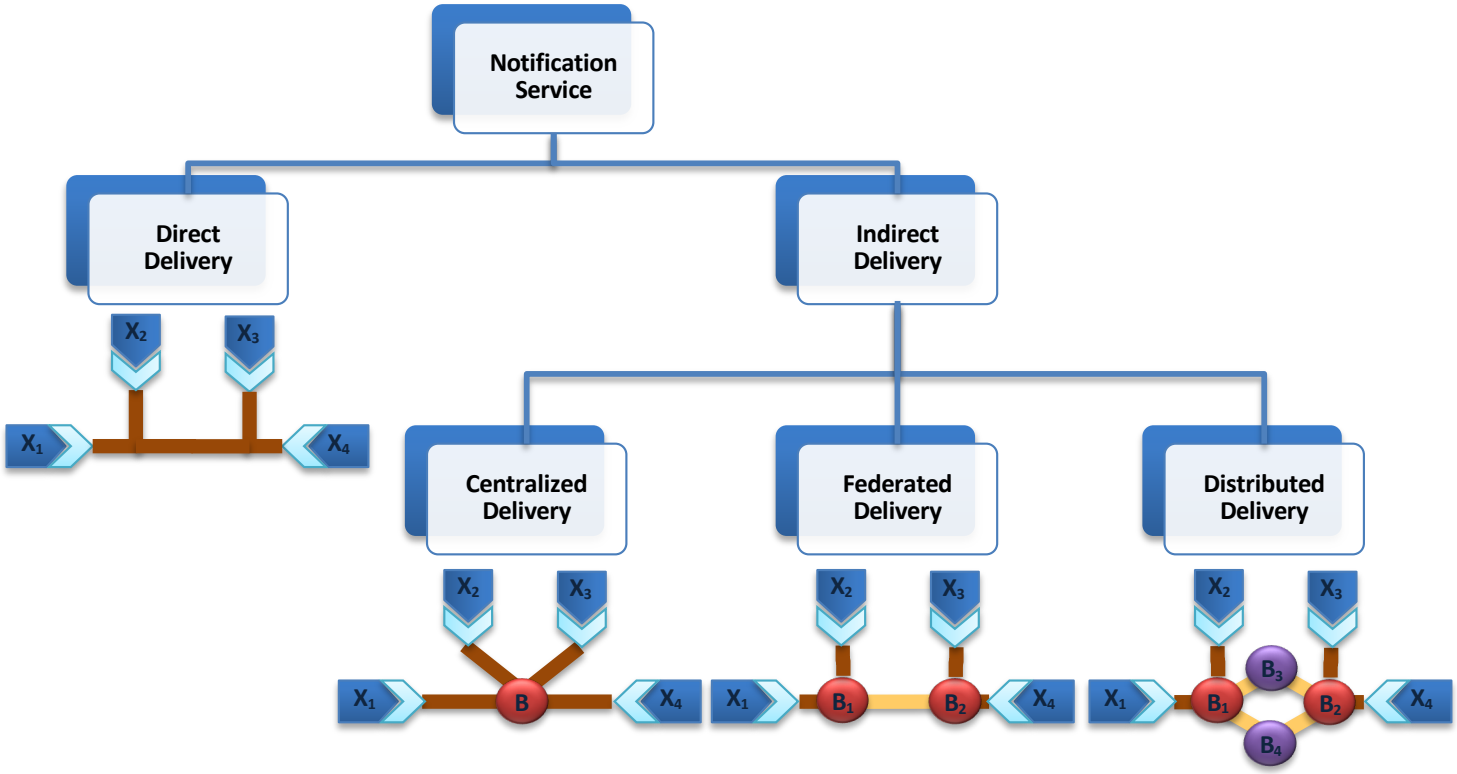




It is the distributed implementation of the Observer Design pattern.

::.. Publish/Subscribe Services (2/6)

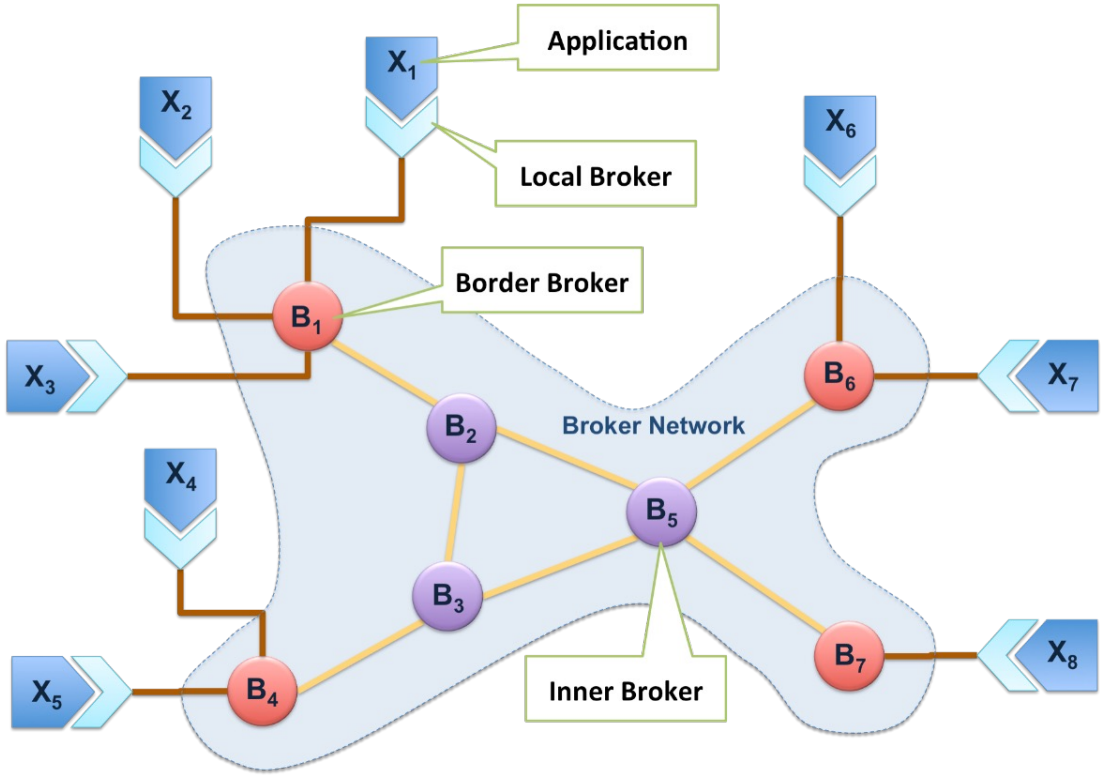
The implementation of Notification Service is based on the concept of broker, and can be realized according to several different architectures proposed during the last decades, and classified as follows:



::.. Publish/Subscribe Services (3/6)

The Notification Service provides an abstraction, which is implemented by applying the Broker design pattern:

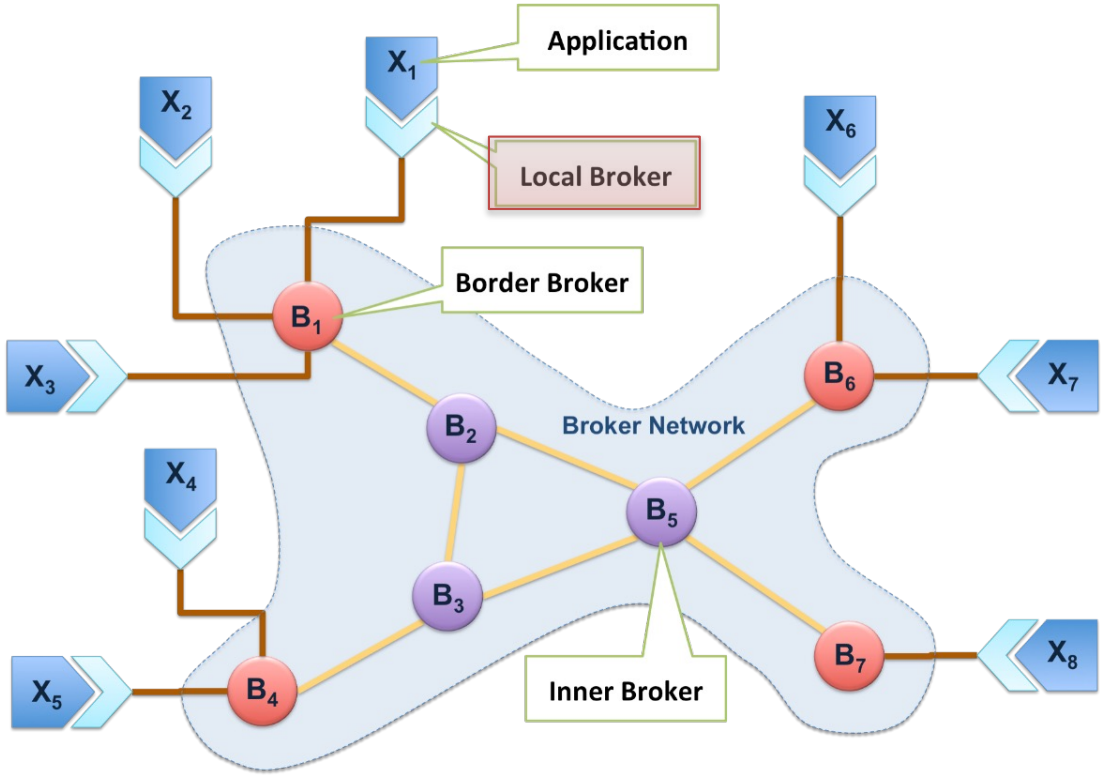
A broker is an intermediary messenger that appropriately retransmits requests, responses and errors, and is juxtaposed between distributed applications communicating with each other.



::.. Publish/Subscribe Services (3/6)

The Notification Service provides an abstraction, which is implemented by applying the Broker design pattern:

A broker is an intermediary messenger that appropriately retransmits requests, responses and errors, and is juxtaposed between distributed applications communicating with each other.

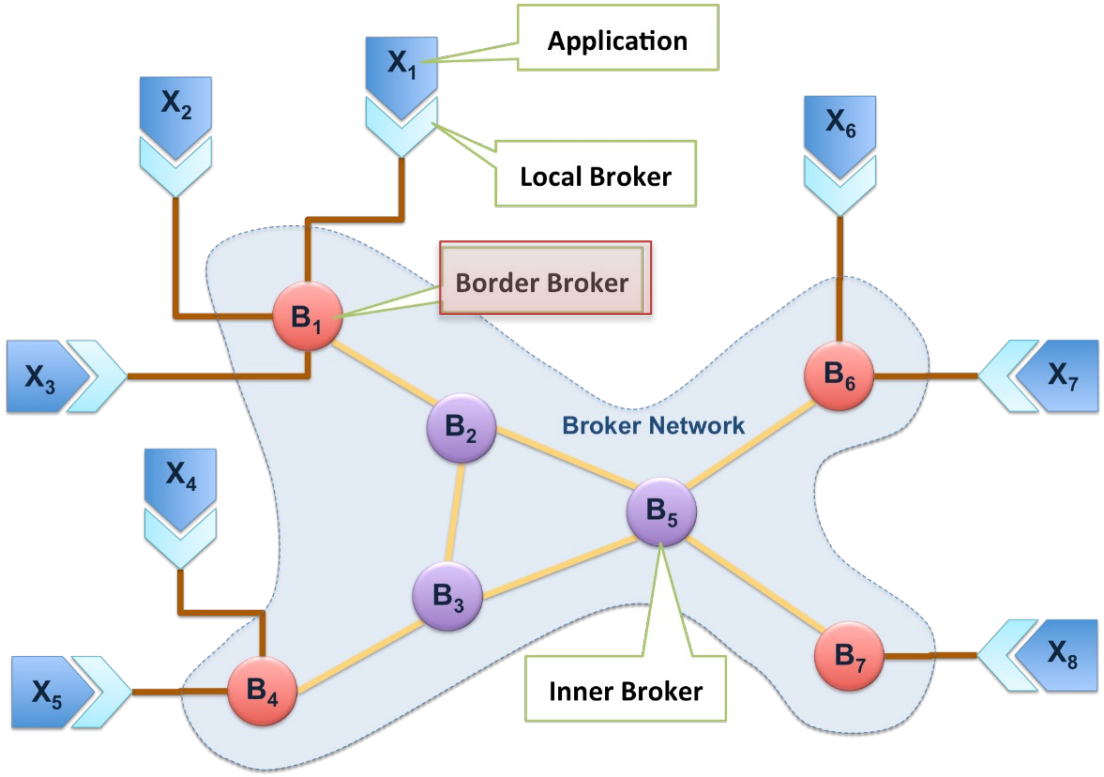


Local Brokers: access points to middleware given by libraries linked to the application that offer the functionality to publish/ receive events.

::.. Publish/Subscribe Services (3/6)

The Notification Service provides an abstraction, which is implemented by applying the Broker design pattern:

A broker is an intermediary messenger that appropriately retransmits requests, responses and errors, and is juxtaposed between distributed applications communicating with each other.

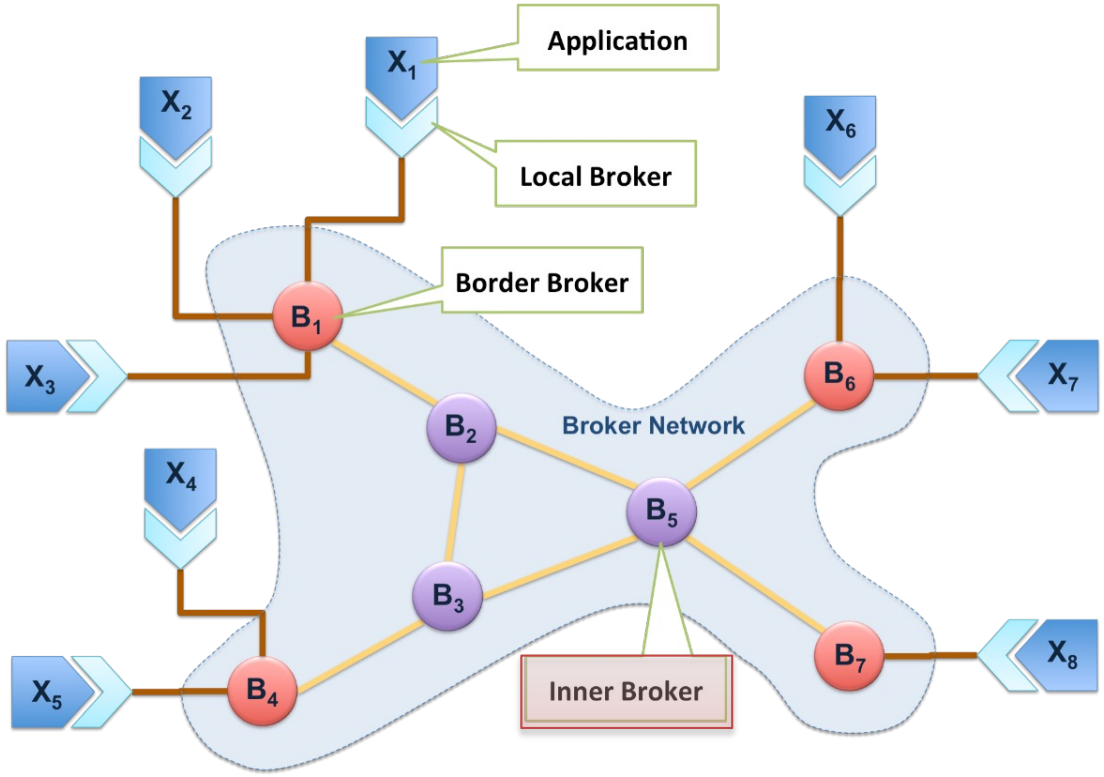


Border Brokers: they constitute the boundary of middleware and maintain connections between local brokers.

::.. Publish/Subscribe Services (3/6)

The Notification Service provides an abstraction, which is implemented by applying the Broker design pattern:

A broker is an intermediary messenger that appropriately retransmits requests, responses and errors, and is juxtaposed between distributed applications communicating with each other.

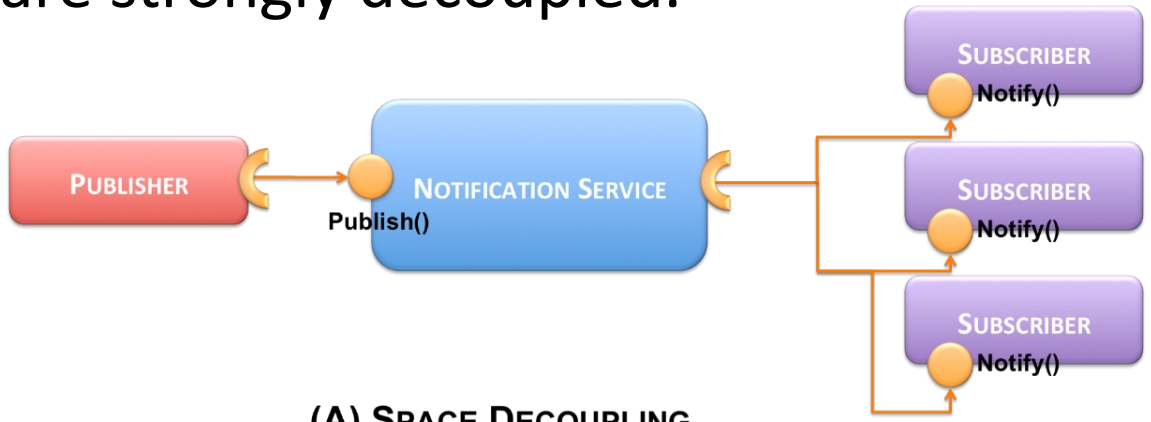


Inner Brokers: connected only to borders and/or other inner brokers. They are responsible for propagating events in the middleware.

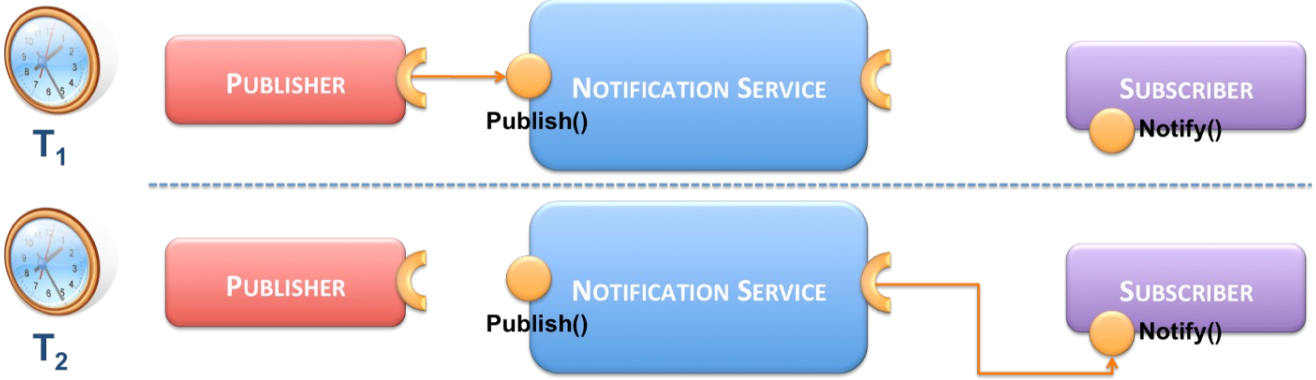
::.. Publish/Subscribe Services (4/6)

Thanks to the role of mediator of the Notification Service, publishers and subscribers are strongly decoupled.

Publishers and subscribers are under no obligation to know each other.



(A) SPACE DECOUPLING

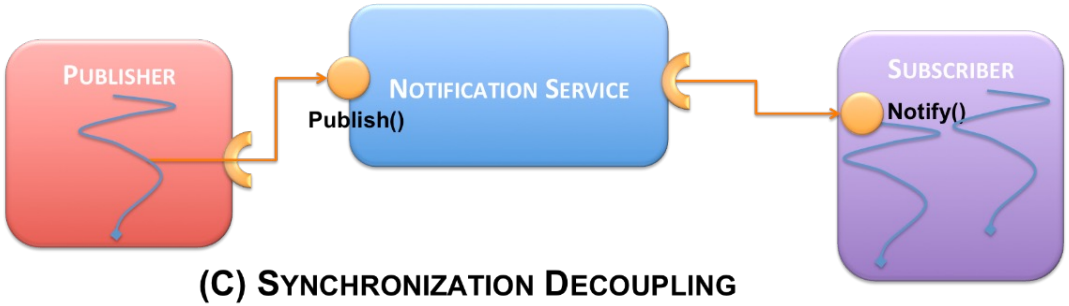


(B) TIME DECOUPLING

Publishers and subscribers do not have to be active at the same time to be able to exchange notifications.

::.. Publish/Subscribe Services (5/6)

The execution of publishers and subscribers is not interrupted to execute the communication primitives.



(C) SYNCHRONIZATION DECOUPLING

Decoupling the production and consumption of notifications has been shown to improve scalability, because all explicit dependencies are removed.

This makes the communication infrastructure optimal for large systems. The Notification Service realizes a complete separation between business and communication logics, favoring reuse.

::.. Publish/Subscribe Services (6/6)

Within the context of IoT, there are multiple standard-based solutions implementing the publish/subscribe communication pattern.

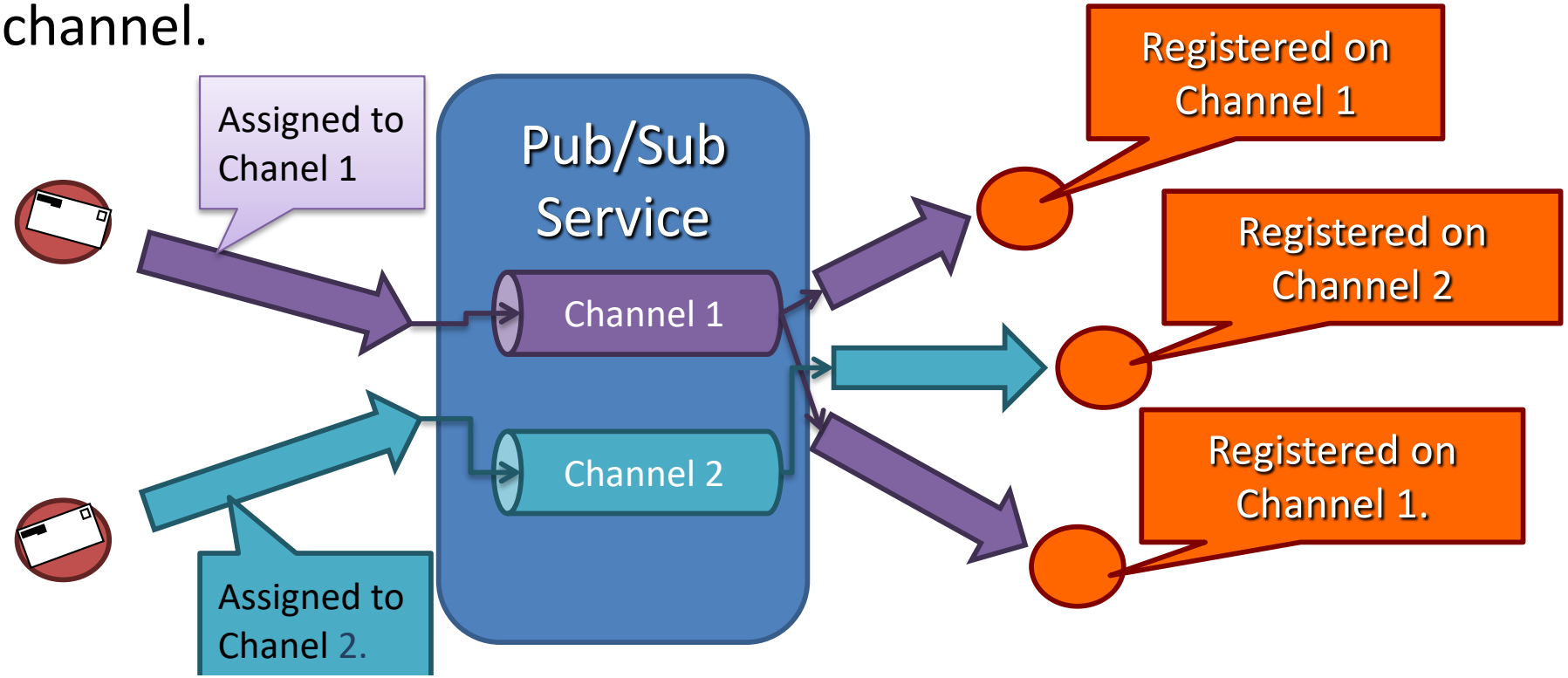
They have been successfully applied worldwide in multiple industrial projects.

Some of them require the presence of brokers. The IoT requires a zero infrastructure by letting nodes to discover and communicate among each others without assuming a fixed broker infrastructure.

::.. Subscription Types (1/4)

The various event middleware solutions are characterized by different subscription strategies:

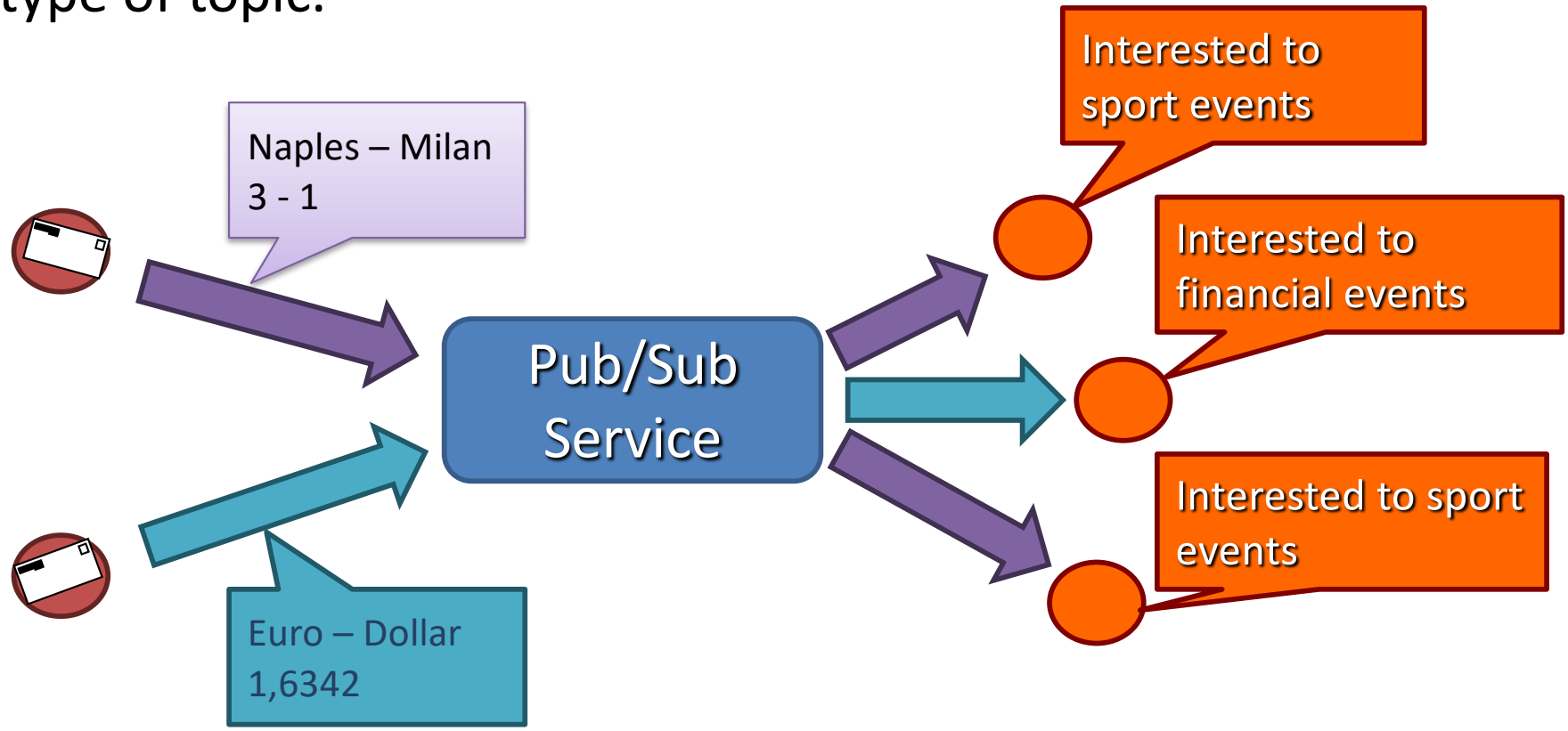
- **Channel-based**: the Notification Service has several "channels" and all events entered at the head of a channel are received by these subscribers registered at the queue of the channel.



::.. Subscription Types (2/4)

The various event middleware solutions are characterized by different subscription strategies:

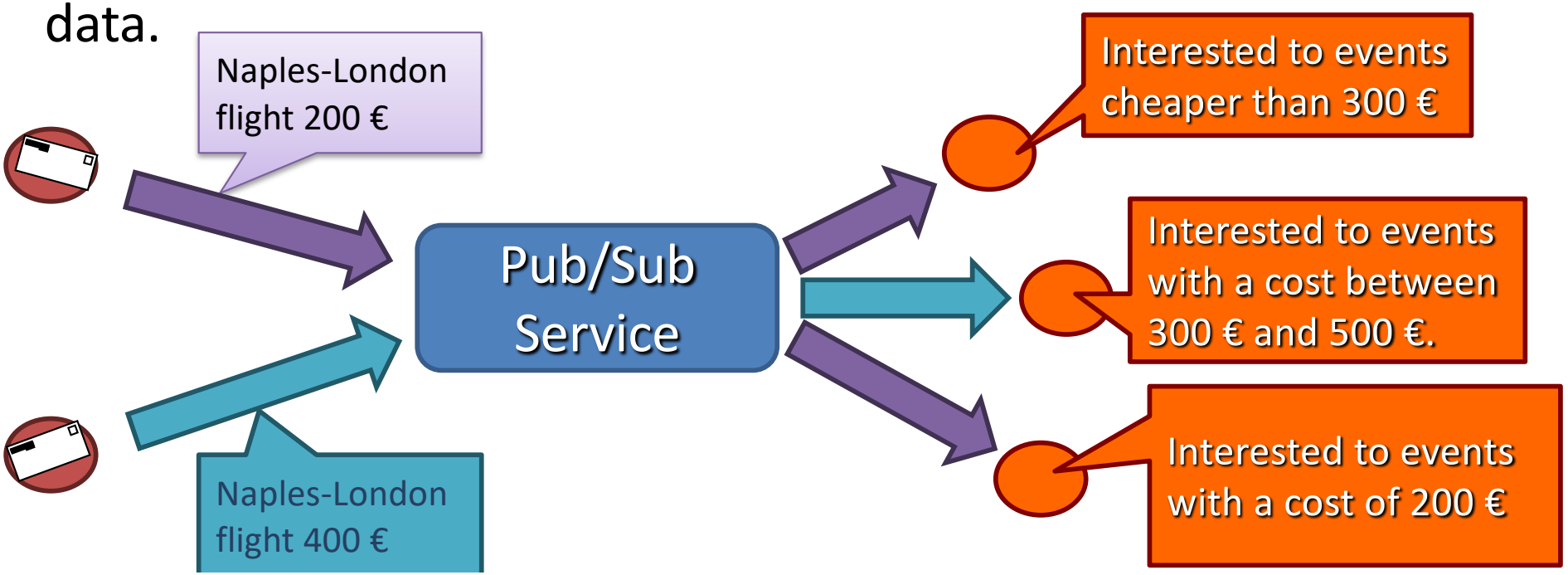
- **Topic-based**: events are associated with a string that defines the topic. The subscribers indicate their interest in a particular type of topic.



::.. Subscription Types (3/4)

The various event middleware solutions are characterized by different subscription strategies:

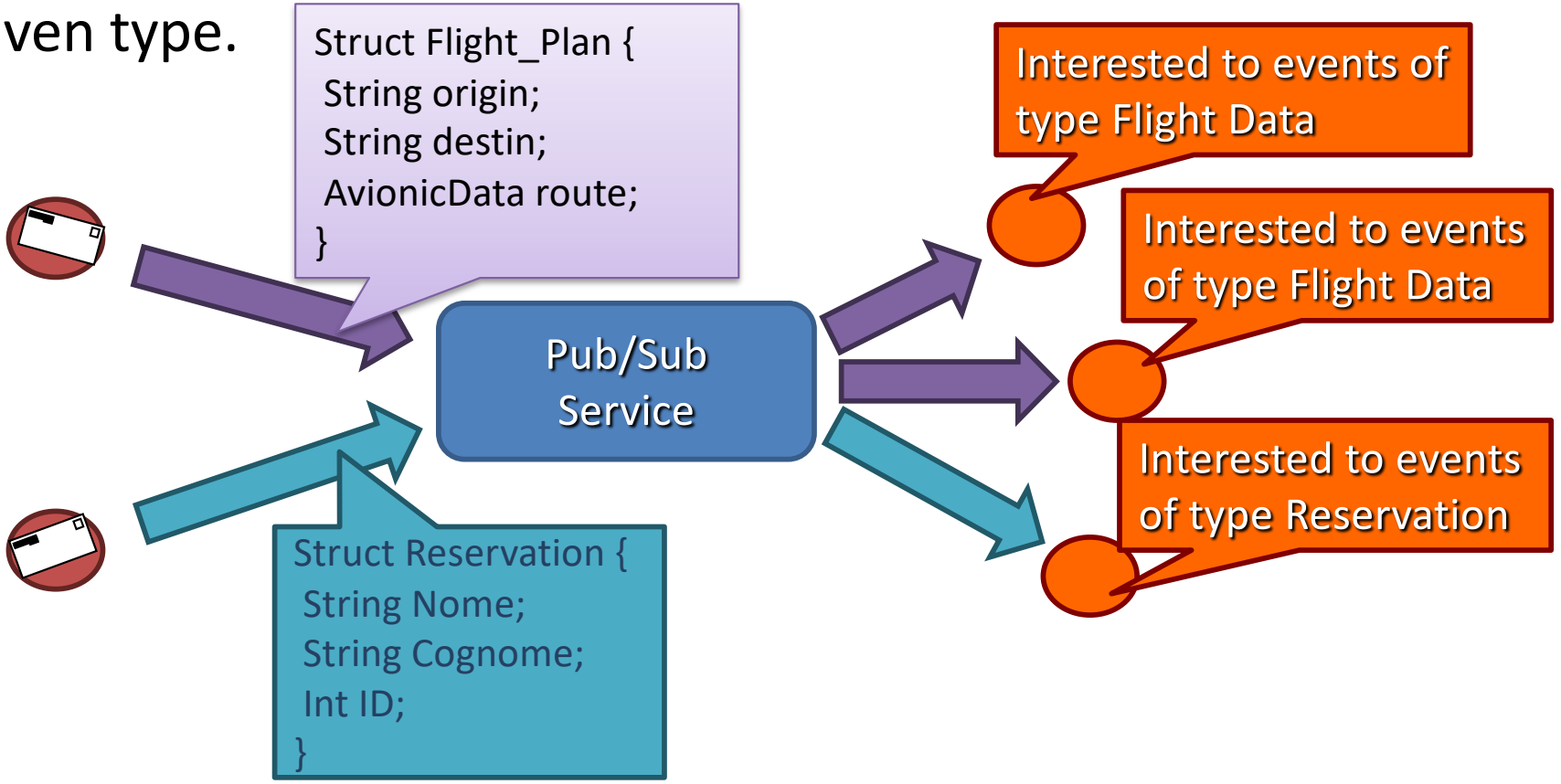
- **Content-based**: the interest of subscribers is encoded in a logical predicate on the content of the event. The event is forwarded to the subscriber only if the relevant subscription predicate is verified by the content of the published event data.



::.. Subscription Types (4/4)

The various event middleware solutions are characterized by different subscription strategies:

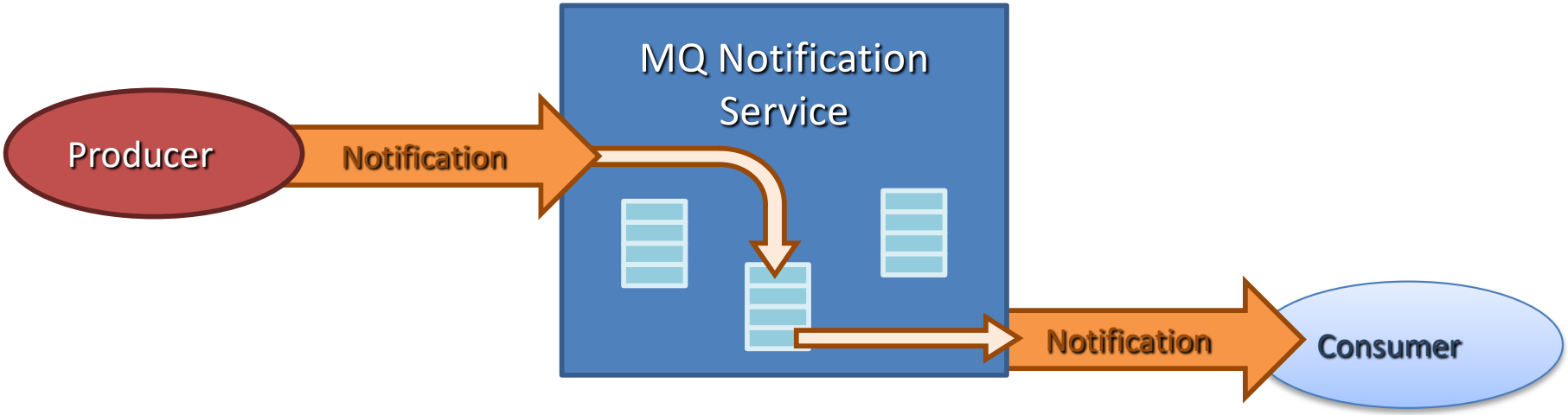
- **Type-based**: a data structure, or type, is associated with events. The subscribers are only interested in the events of a given type.



::.. Message Queueing (1/2)

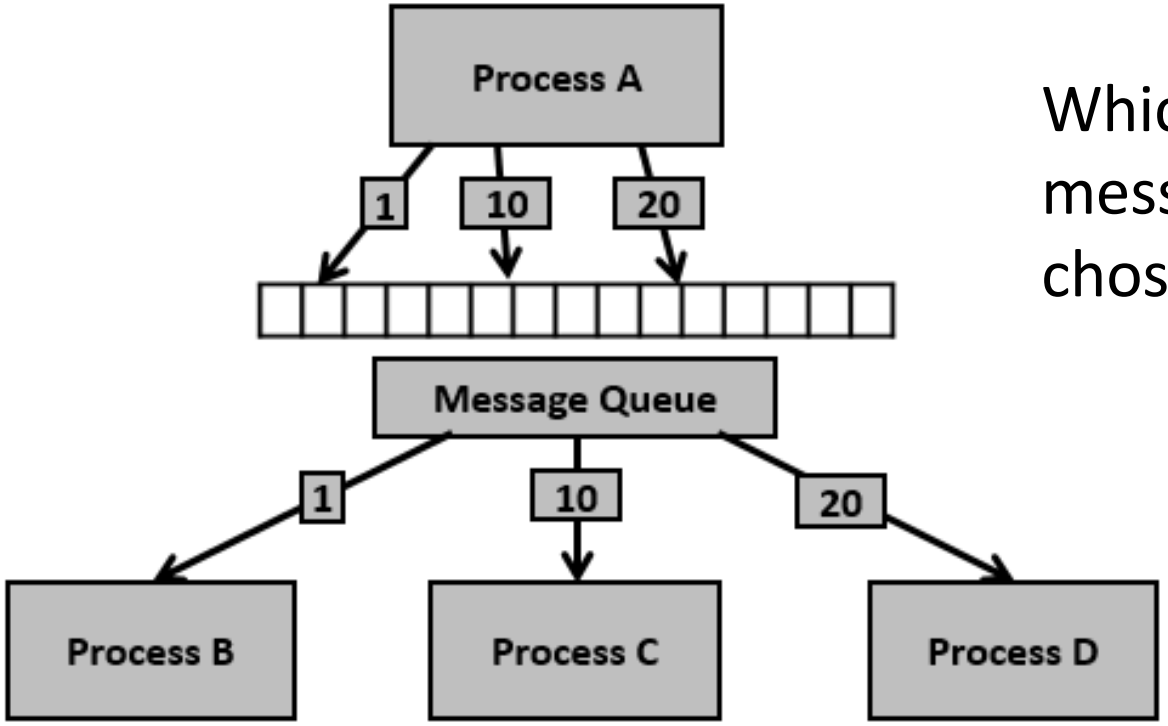
Another equally widespread approach is the Message Queueing:

- The Notification Service is implemented as a set of message queues, hosted on a centralized server, or a federation of servers;
- When an event is generated, the producer carries out a "push" of the message in a queue;
- Consumers subscribed to that queue can make a "pop" to get the message.



::.. Message Queueing (2/2)

Message queues can support high rates of consumption by adding multiple consumers for each queue (or topic), but only one consumer will receive each message on the topic: each message is deleted from the queue once it has been received and processed by a consumer.



Which consumer receives the message depends on the chosen implementation.

Some solutions can provide an hybrid approach by combining these two patterns.

::.. HTTP (1/3)

The Hyper Text Transport Protocol (HTTP) is the fundamental client-server protocol used for the Web, and the one most compatible with existing network infrastructure. The most widely accepted version is HTTP/1.1.

A client sends an HTTP request message and the server then returns a response message, containing the resource that was requested in case the request was accepted.

Recently, HTTP has been associated with REST, where the CRUD operations on a resource are mapped to the HTTP POST, GET, PUT and DELETE methods.

::.. HTTP (2/3)

HTTP uses TCP, which creates challenges in resource constrained environments:

- the constrained nodes most of the time send small amounts of data sporadically and setting up a TCP connection takes time and produces unnecessary overhead.

For QoS, HTTP does not provide additional options, but instead it relies on TCP, which guarantees successful delivery as long as connection is not interrupted.

While in general, HTTP presents one the most stable protocol options, there are still a few issues that have lead to the exploration of alternative protocol solutions, due to HTTP complexity, long header fields and high power consumption. Furthermore, HTTP uses the request/reply paradigm, which is not suitable for push notifications, where the server delivers notifications to the client without a client request.

::.. HTTP (3/3)

The TCP protocol overhead may be too large (three way handshake), especially in case of simple computing nodes in IoT architectures. HTTP does not explicitly define QoS levels and requires additional support for it. This has led to modifications and extension of HTTP, most notably in form of HTTP/2.0, that introduced a number of improvements, some of which are especially relevant in IoT context:

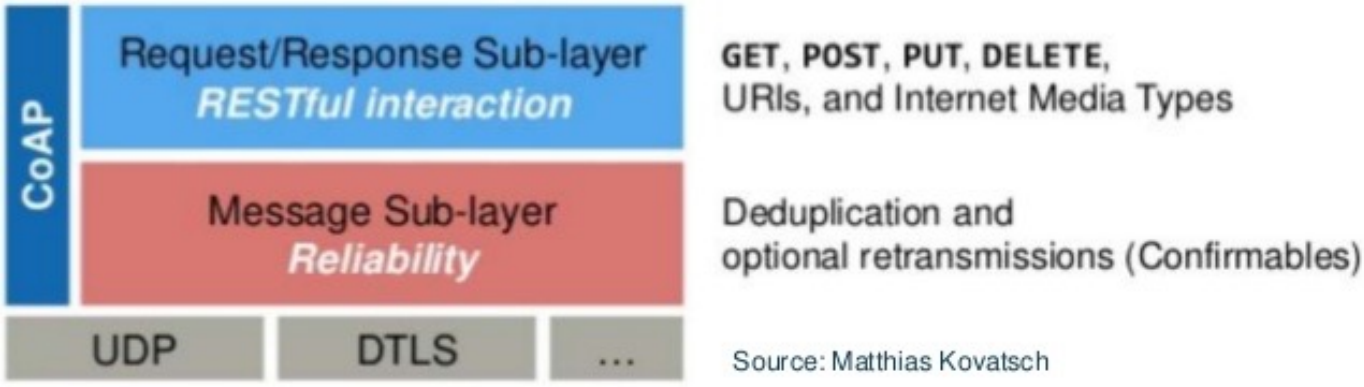
- a more efficient use of network resources and a reduced latency by introducing compressed headers, using a very efficient and low memory compression format, as well as allowing multiple concurrent exchanges on the same connection.
- the so-called server push, where the server can send content to clients with no need to wait for their requests.

The drawbacks of this version of the protocol in IoT based systems are not known yet, as to the best of our knowledge there are no implemented and tested solutions reported in the literature.

::.. COAP (1/2)

The Constrained Application Protocol (CoAP) is similar to HTTP, with the use of tested and well accepted REST architecture and the support of the request/response paradigm. CoAP is considered a lightweight protocol, so the headers, methods and status codes are all binary encoded, thus reducing the protocol overhead in comparison with many protocols.

It also runs over less complex UDP transport protocol instead of TCP, at the cost of reduced reliability (allowing CoAP running on TCP for those applications requiring reliable communication).

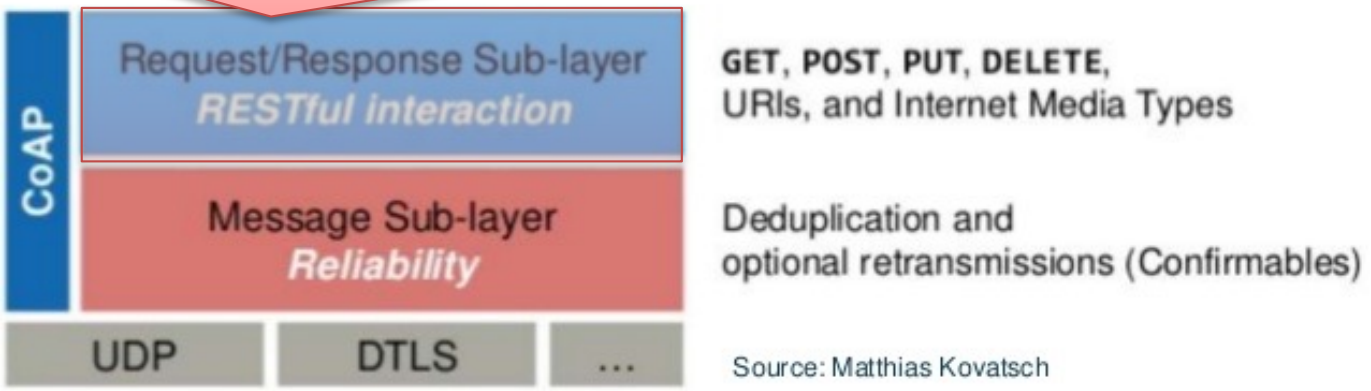


::.. COAP (1/2)

The Constrained Application Protocol (CoAP) is similar to HTTP, with the use of tested and well accepted REST architecture and the support of the request/response paradigm. CoAP is considered a lightweight protocol, so the headers, methods and status codes are all binary encoded, thus reducing the protocol overhead in comparison with many protocols.

It also runs over less complex UDP transport protocol instead of

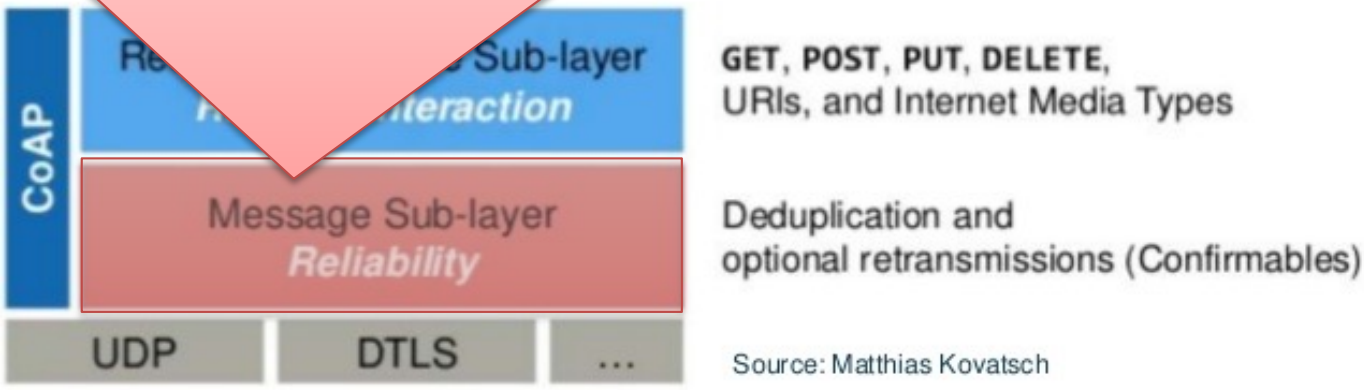
The first layer implements RESTful paradigm and allows for CoAP clients to use the HTTP-like methods when sending requests.



::.. COAP (1/2)

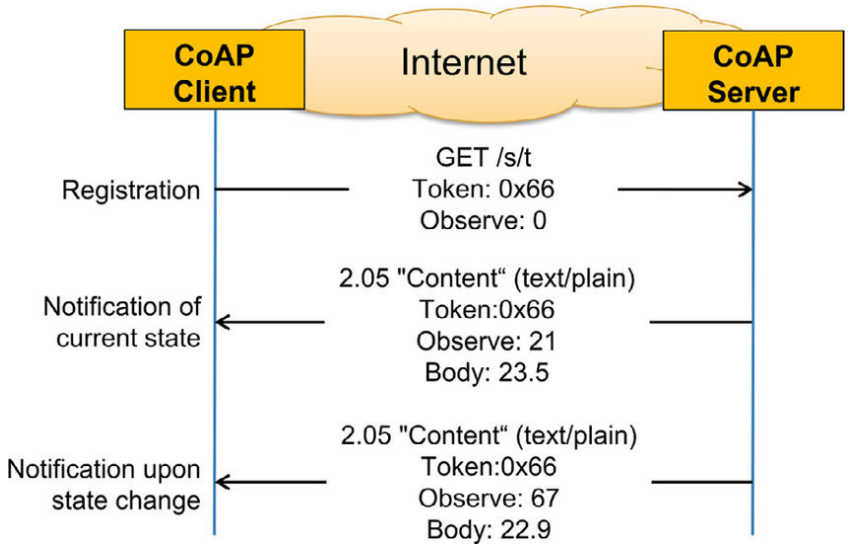
The Constrained Application Protocol (CoAP) is similar to HTTP, with the use of tested and well accepted REST architecture and the support of the request/response paradigm. CoAP is considered a lightweight protocol, so the headers, methods and status codes are all binary encoded, thus reducing the protocol

The second layer is designed for retransmitting lost packets by using an acknowledgment scheme for messages. Precisely,this feature that marks whether the messages need the acknowledgement is what enables QoS differentiation in CoAP, albeit in a limited fashion.



::.. COAP (2/2)

CoAP has an optional feature allowing clients to continue receiving changes on a requested resource from the server by adding an observe option to a GET request. With this option, the server adds the client to the list of observers for the specific resource, which will allow the client to receive the notifications when resource state changes.



IETF has recently released the draft of Publish-Subscribe Broker that extends the capabilities of CoAP for supporting nodes with long interruptions in connectivity and/or up-time, with preliminary performance evaluations showing promising results.

::.. MQTT (1/3)

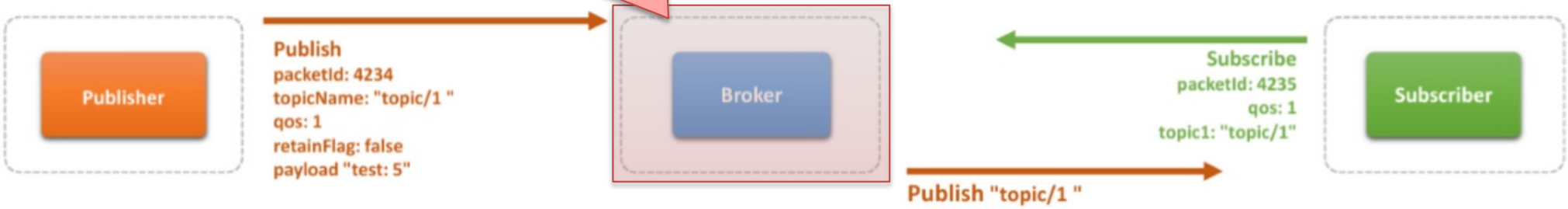
Message Queue Telemetry Transport Protocol(MQTT) is one of the lightweight messaging protocols that follows the publish-subscribe paradigm, which makes it rather suitable for resource constrained devices and for non-ideal network connectivity conditions.

MQTT runs on top of the TCP transport protocol, ensuring reliability, but thanks to its lighter header, MQTT comes with much lower power requirements than HTTP, making it one of the most prominent protocol solutions in constrained environments.

::.. MQTT (1/3)

Message Queue Telemetry Transport Protocol(MQTT) is one of the lightweight messaging protocols that follows the publish-subscribe paradigm, which makes it rather suitable for resource constrained devices and for non-ideal network connectivity

For the broker, it is necessary to install MQTT broker library, for example the Mosquitto broker. It should be noted that there are various other MQTT protocol brokers that are open for use, which differ by the way of implementing the MQTT protocol.



::.. MQTT (2/3)

For QoS, MQTT defines three QoS levels:

- QoS0 delivers on the best effort basis, without confirmation on message reception.
- QoS1 assures that messages will arrive by means of a retransmission scheme.
- QoS2 guarantees that the message will be delivered exactly once without duplications.

Another important feature MQTT offers is the possibility to store some messages for new subscribers by setting a ‘retain’ flag in published messages. In some situations, especially when the state of the followed topic does not change often, it is useful to enable for new subscribers to receive the information on that topic, without having to wait for the state to change in order to receive a message about the topic.

::.. MQTT (3/3)

MQTT uses TCP which can be critical for constrained devices. To this end, a solution has been proposed as MQTT for Sensor Networks (MQTT-SN) version that uses UDP and supports topic name indexing, which reduce the size of the payloads. This is done by numbering the data packets with numeric topic id's rather than long topic names.

The biggest disadvantage is that at the moment MQTT-SN is only supported by a few platforms, and there is only one free broker implementation known, called Really Small Message Broker.

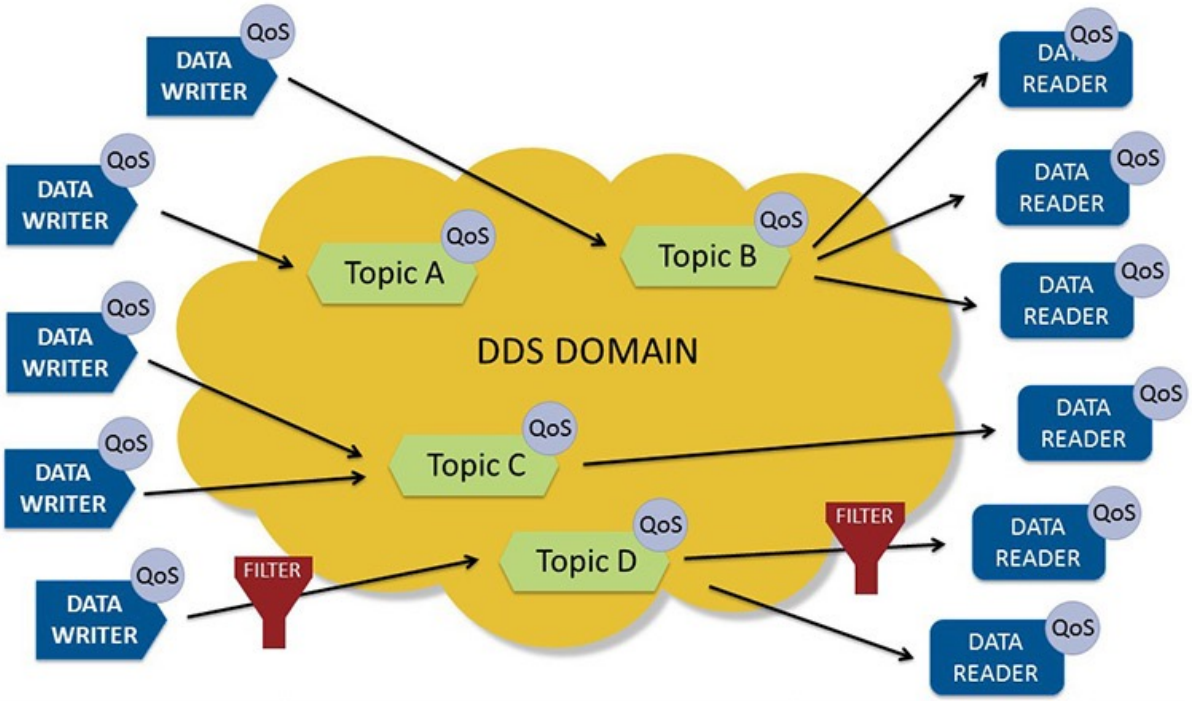
::.. DDS (1/3)

The Data Distribution Service (DDS) is a real-time data-centric interoperability standard that uses a publish-subscribe interaction model. Unlike some other publish-subscribe protocols, DDS is decentralized and based on peer-to-peer communication, and as such does not depend on the broker component.

The fact that there is no broker decreases the probability of system failure because there is no single point of failure for the entire system, making a system more reliable. Both communication sides are decoupled from each other, and a publisher can publish data even if there are no interested subscribers. The data usage is fundamentally anonymous, since the publishers do not enquire about who consumes their data.

::.. DDS (2/3)

One of the salient features of DDS is its scalability, which comes from it support for dynamic discovery. Another important and unique characteristic in DDS is its data-centricity, unlike most protocols that are message-centric. For data-centric paradigm, what matters the most is the data that clients want to access to, so the focus is on the content information itself.



In DDS, the data type and content define the communication, where-as in message-centric protocols the focus lies on the operations and mechanisms for deliver-ing that data.

::.. DDS (3/3)

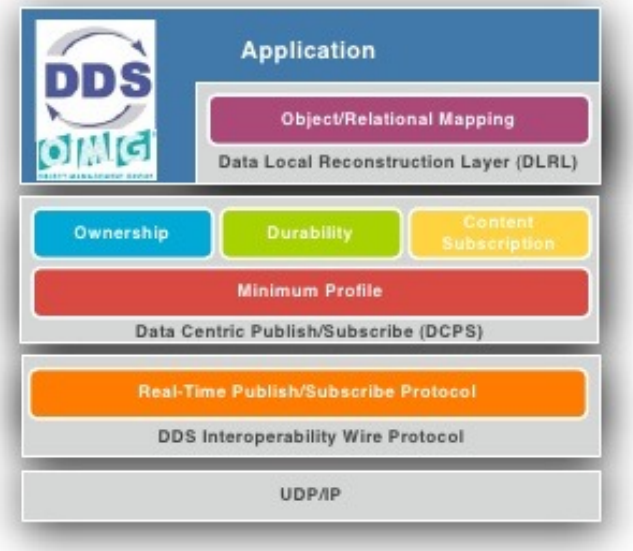
DDS uses UDP by default, but it can also support TCP. Another important protocol in DDS is the Real-Time Publish-Subscribe (RTPS) wire protocol, which represents DDS interoperability protocol that allows data sharing among different vendor implementations. One of the advantages of using DDS is a wide set of QoS policies offered (over 20 QoS as defined by the standard).

DDS v1.2 API Standard

- Language Independent, OS and HW architecture independent
- **DCPS.** Standard API for Data-Centric, Topic-Based, Real-Time Publish/Subscribe
- **DLRL.** Standard API for creating Object Views out of collection of Topics

DDSI/RTPS v2.1 Wire Protocol Standard

- Standard wire protocol allowing interoperability between different implementations of the DDS standard
- Interoperability demonstrated among key DDS vendors in March 2009



::.. DDS (3/3)

DDS uses UDP by default, but it can also support TCP. Another important protocol in DDS is the Real-Time Publish-Subscribe (RTPS) wire protocol which represents DDS interoperability

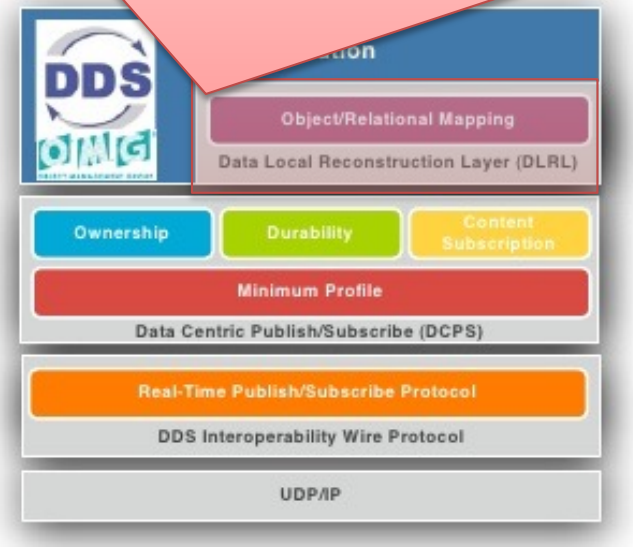
The DLRL hides the publish-subscribe mechanisms and operations and provides the object-oriented abstraction of having topic data local to the application.

DDS v1.2 API Standard

- Language Independent, OS and HW architecture independent
- **DCPS.** Standard API for Data-Centric, Topic-Based, Real-Time Publish/Subscribe
- **DLRL.** Standard API for creating Object Views out of collection of Topics

DDSI/RTPS v2.1 Wire Protocol Standard

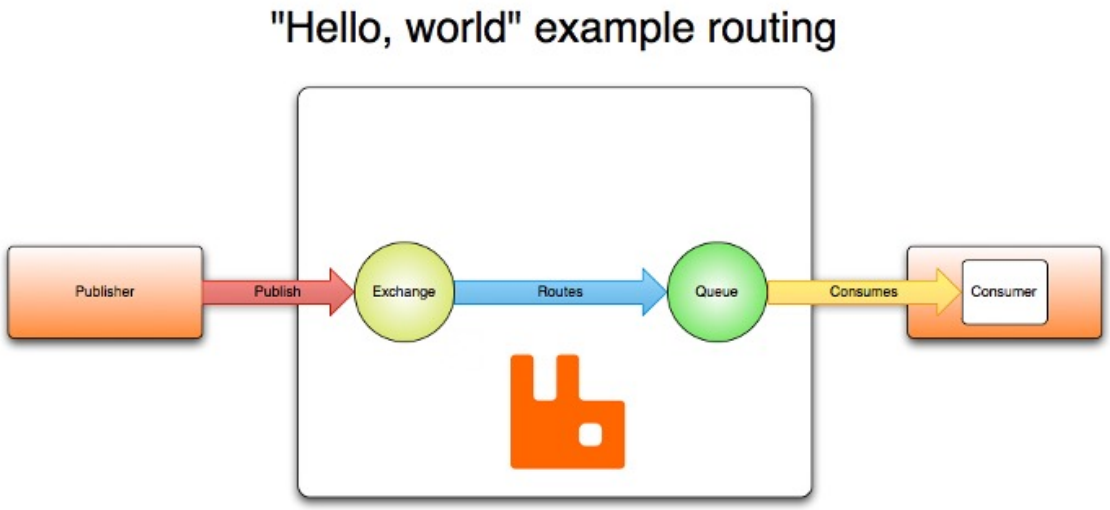
- Standard wire protocol allowing interoperability between different implementations of the DDS standard
- Interoperability demonstrated among key DDS vendors in March 2009



::.. AMQP (1/2)

The Advanced Message Queueing Protocol (AMQP) is an open standard protocol that follows the publish-subscribe paradigm and has been implemented in two very different versions:

- AMQP 0.9.1 implements the publish-subscribe paradigm, which revolves around two main AMQP entities in the broker.
  - The exchanges is used to direct the messages received from publishers. The publishing of messages to an exchange is followed by messages being routed into one or more appropriate queues.

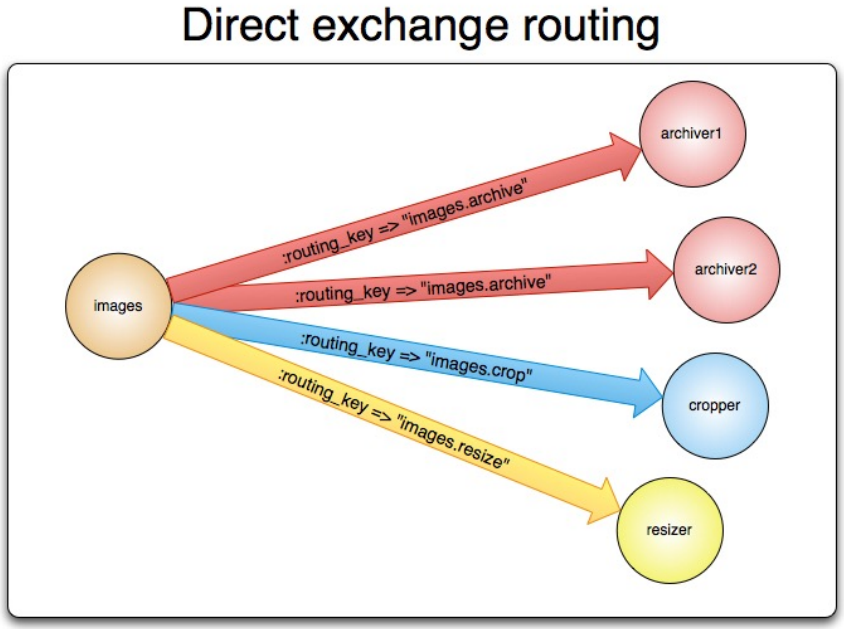


  - Having multiple queues depends on whether there are more subscribers interested in a particular message, in which case the broker can duplicate the messages and send their copies to multiple queues.

::.. AMQP (2/2)

The Advanced Message Queueing Protocol (AMQP) is an open standard protocol that follows the publish-subscribe paradigm and has been implemented in two very different versions:

- AMQP 1.0 is not tied to any particular messaging mechanism. It follow a peer-to-peer paradigm, and can be used without a broker in the middle.



AMQP uses TCP for reliable transport, and in addition it provides three different levels of QoS, same as MQTT.

::.. XMPP (1/2)

Extensible Messaging and Presence Protocol (XMPP) is a text-based protocol, based on Extensible Markup Language (XML) that implements both client-server and publish-subscribe interaction, running over TCP. In IoT solutions it is designed to allow users to send messages in real time, in addition to managing the presence of the user.

Every user on the network has a unique XMPP address, called JID (Jabber IDs), structured like an email address with a username and a domain name for the server where that user resides, separated by an at sign (@), such as username@example.com.

Since a user may wish to log in from multiple locations, they may specify a resource. A resource identifies a particular client belonging to the user (for example home, work, or mobile). This may be included in the JID by appending a slash followed by the name of the resource. For example, the full JID of a user's mobile account could be username@example.com/mobile.

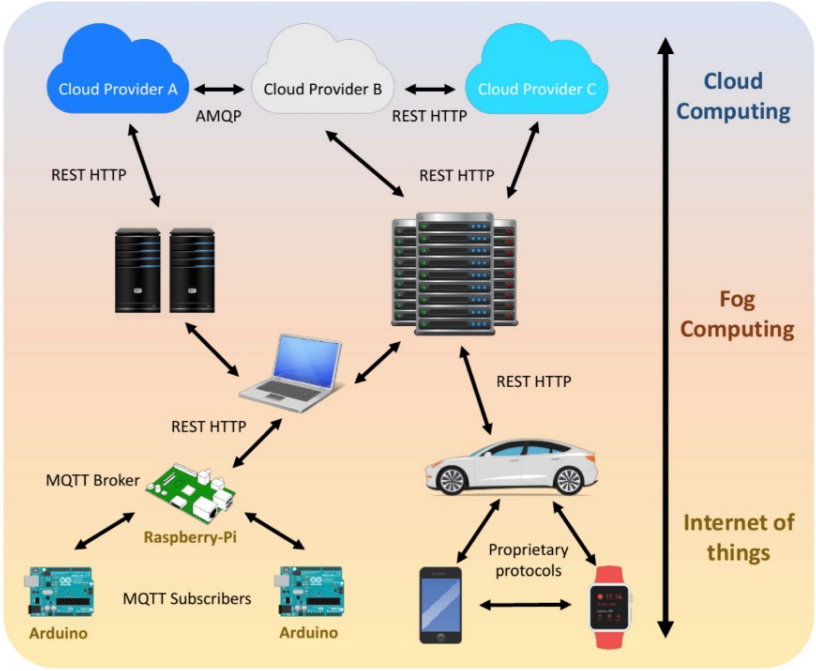
::.. XMPP (2/2)

XMPP provides a general framework for messaging across a network, which offers a multitude of applications beyond traditional Instant Messaging and the distribution of Presence data. XMPP provides a solid base for the discovery of services residing locally or across a network, and the availability of these services (via presence information), as specified by XEP-0030 DISCO. XMPP can be applied at a variety of levels and may prove ideal as an extensible middleware or MOM protocol. Widely known for its ability to exchange XML-based content natively, it has become an open platform for the exchange of other forms of content including proprietary binary streams

By using XML, the size of the messages makes it inconvenient in the networks with bandwidth constraints. Another downside is the absence of reliable QoS guarantees. Because XMPP runs on top of a persistent TCP connection and lacks an efficient binary encoding, it has not been practical for use over lossy, low-power wireless networks often associated with IoT technologies. However, lately, there has been a lot of effort to make XMPP better suited for IoT.

::.. Multi-Technology Deployment

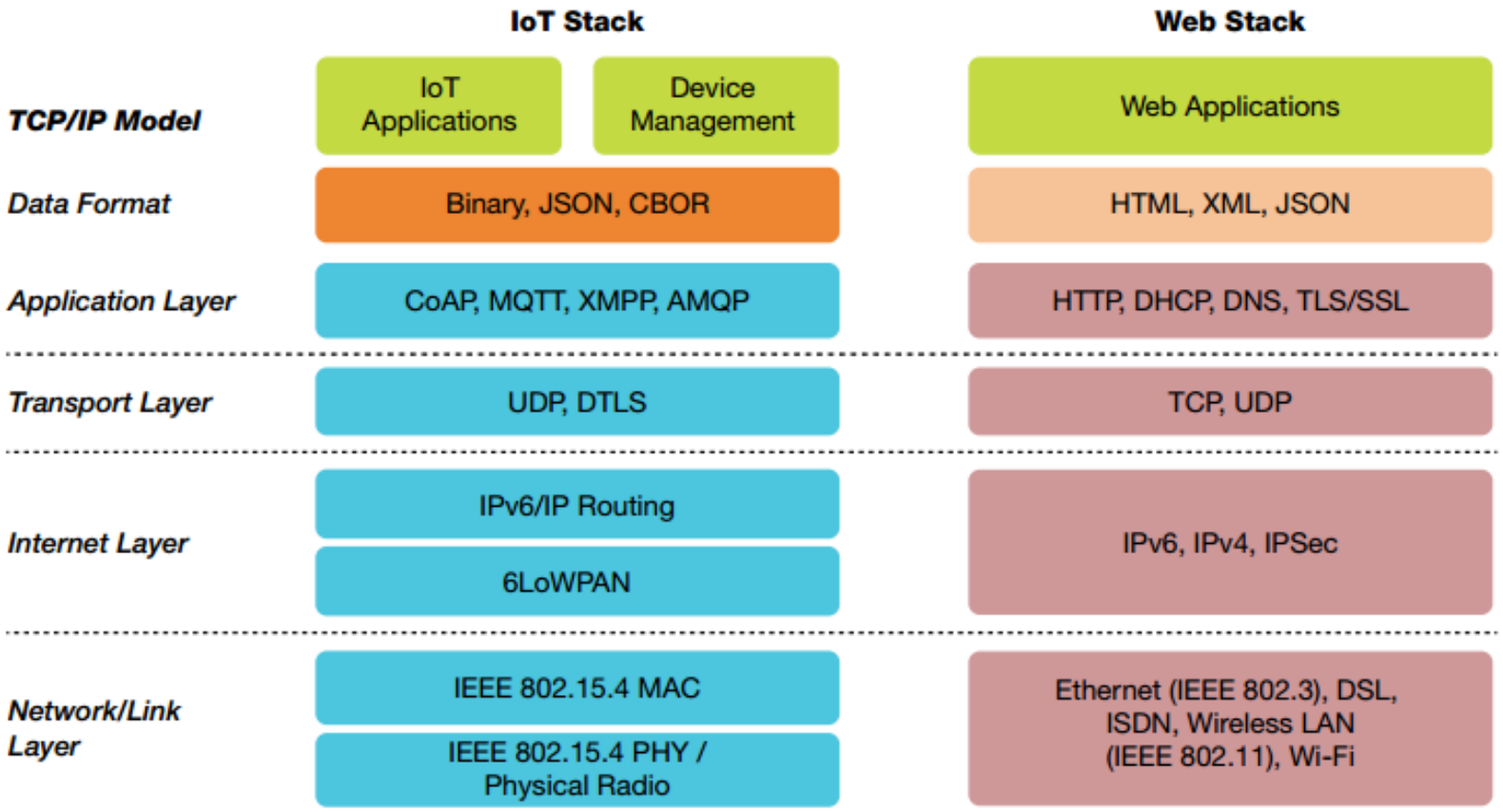
An IoT deployment may encompass more than one communication solution.



Protocol	Req.-Rep.	Pub.-Sub.	Transport	QoS
REST HTTP	X		TCP	-
MQTT		X	TCP	3 Levels
CoAP	X	X	UDP	Limited
AMQP	X	X	TCP	3 Levels
DDS		X	TCP/UDP	Extensive
XMPP	X	X	TCP	-
HTTP/2.0	X	X	TCP	-

::.. IoT vs Web

The IoT is typically characterized by a different software stratification with respect to the implemented communication protocols.



::.. Discovery Protocols (1/4)

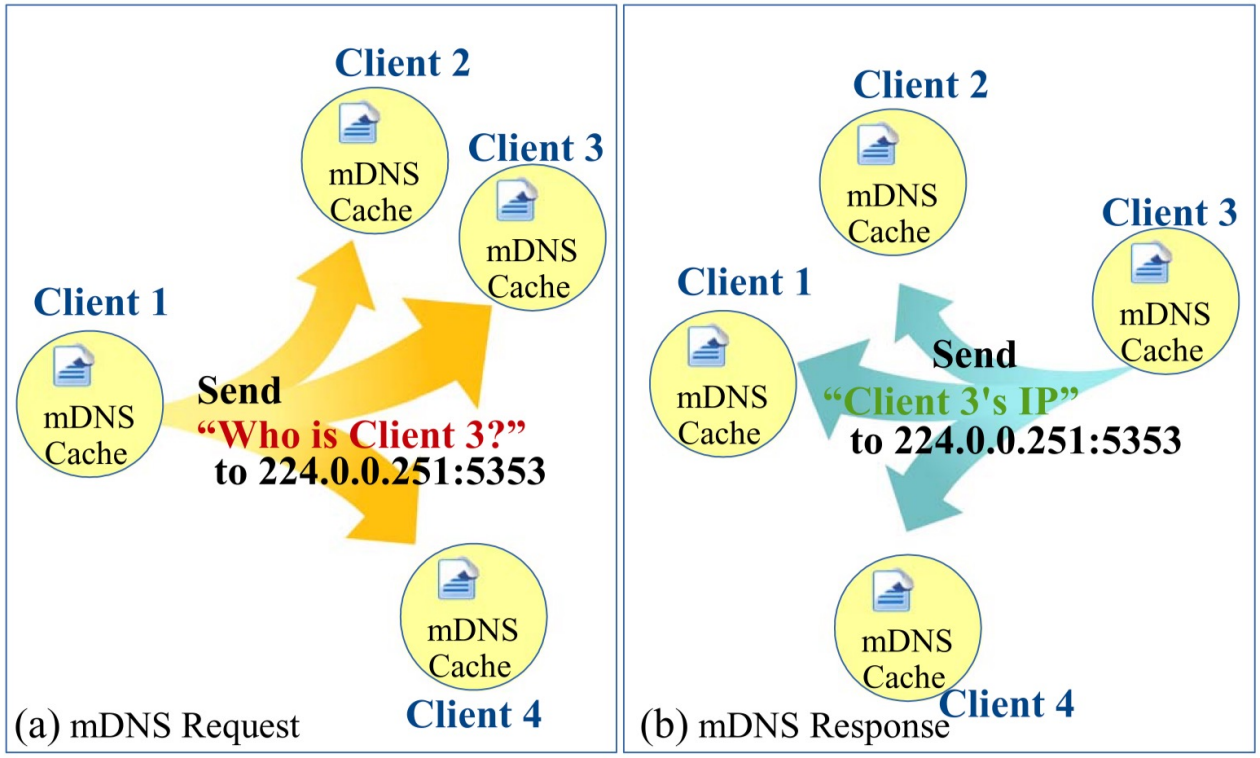
The high scalability of the IoT requires a resource management mechanism that is able to register and discover resources and services in a self-configured, efficient, and dynamic way. The most dominant protocols in this area are multicast DNS (mDNS) and DNS Service Discovery (DNS-SD) that can discover resources and services offered by IoT devices.

Multicast DNS (mDNS) is flexible due to the fact that the DNS namespace is used locally without extra expenses or configuration. mDNS is an appropriate choice for embedded Internet-based devices due to the facts that

1. There is no need for manual reconfiguration or extra administration to manage devices;
2. It is able to run without infrastructure;
3. It can continue working if failure of infrastructure happens.

::.. Discovery Protocols (2/4)

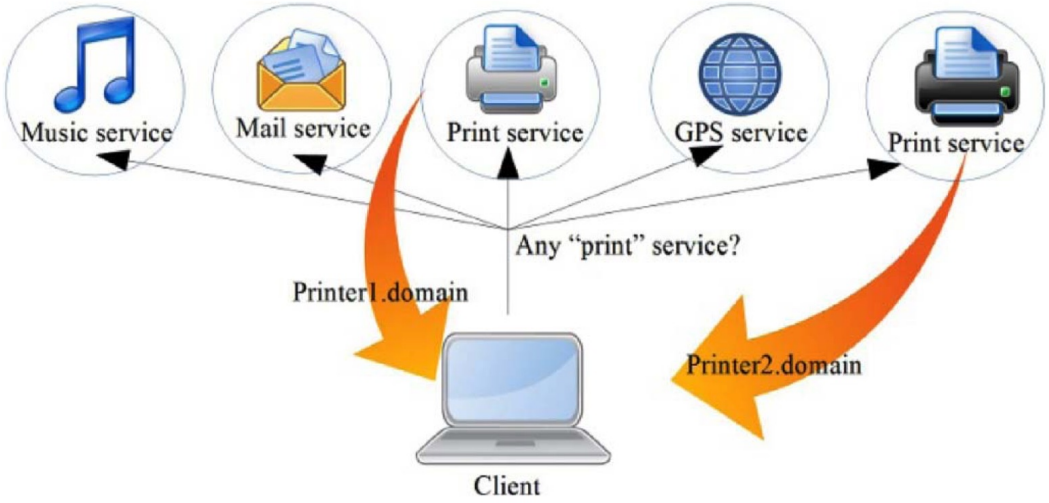
mDNS inquires names by sending an IP multicast message to all the nodes in the local domain (a). By this query, the client asks devices that have the given name to reply back. When the target machine receives its name, it multicasts a response message which contains its IP address (b).



All devices in the network that obtain the response message update their local cache using the given name and IP address.

::.. Discovery Protocols (3/4)

The pairing function of required services by clients using mDNS is called DNS-based service discovery (DNS-SD). Using this protocol, clients can discover a set of desired services in a specific network by employing standard DNS messages.



DNS-SD, like mDNS, is part of the zero configuration aids to connect machines without external administration or configuration.

::.. Discovery Protocols (4/4)

Essentially, DNS-SD utilizes mDNS to send DNS packets to specific multicast addresses through UDP. There are two main steps to process Service Discovery:

- finding host names of required services such as printers,
- pairing IP addresses with their host names using mDNS.

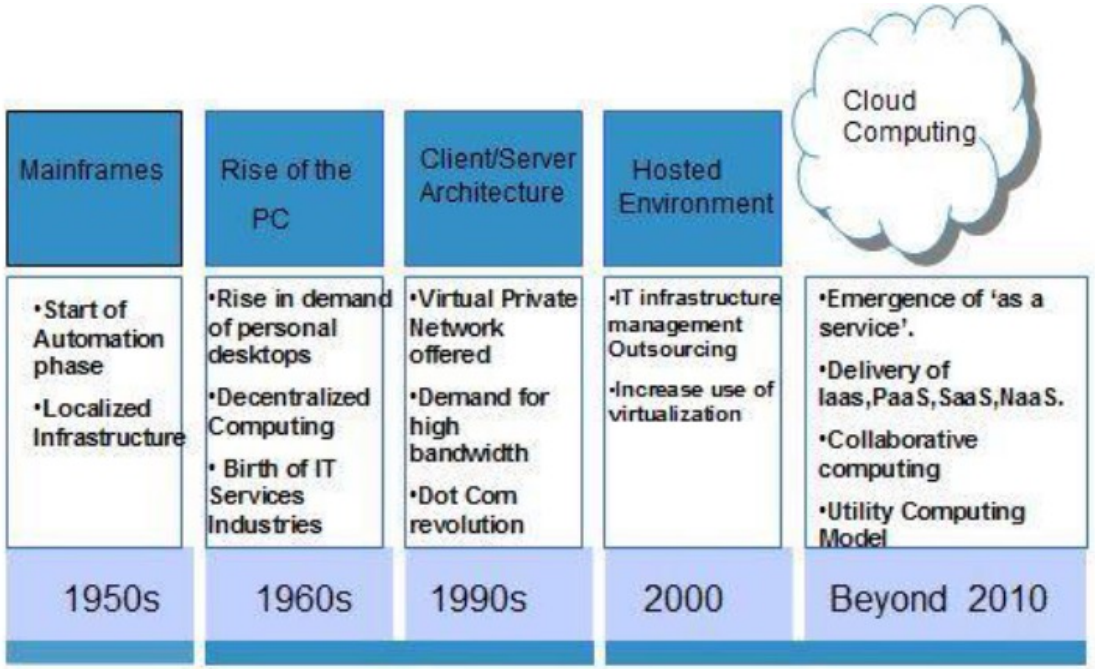
The Pairing function multicasts network attachments details like IP, and port number to each related host. Using DNS-SD, the instance names in the network can be kept constant as long as possible to increase trust and reliability.

The main drawback of these two protocols is the need for caching DNS entries especially when it comes to resource-constrained devices. However, timing the cache for a specific interval and depleting it can solve this issue. Bonjour and Avahi are two well-known implementations covering both mDNS and DNS-SD.

Cloud Support

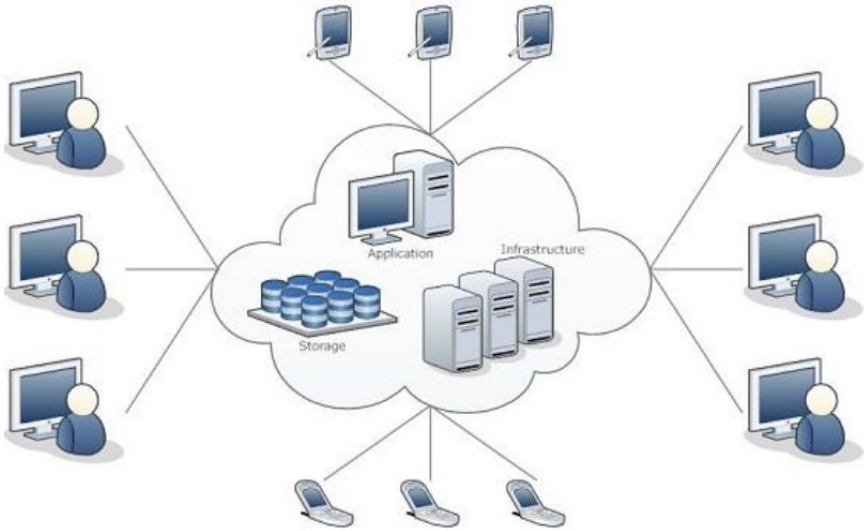
::.. Introduction to Cloud (1/4)

The term Cloud refers to a Network or Internet. In other words, we can say that Cloud is something, which is present at a remote location. Cloud can provide services over network, i.e., on public networks or on private networks. Applications such as e-mail, web conferencing, customer relationship management (CRM), all run in cloud.



::.. Introduction to Cloud (2/4)

Cloud Computing refers to manipulating, configuring, and accessing the applications online. It offers online data storage, infrastructure and application.

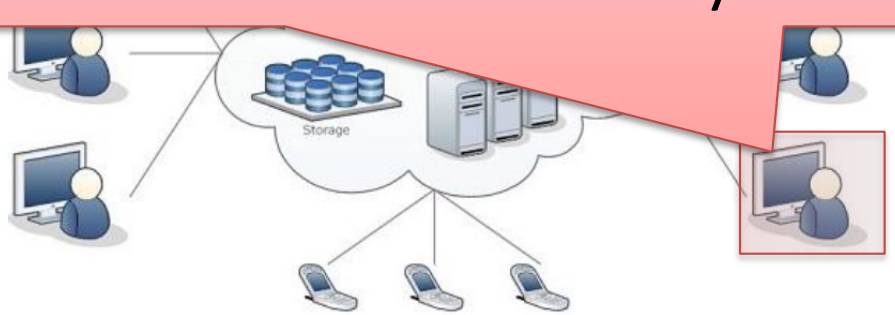


We need not to install a piece of software on our local PC and this is how the cloud computing overcomes platform dependency issues. Hence, the Cloud Computing is making our business application mobile and collaborative.

::.. Introduction to Cloud (2/4)

Cloud Computing refers to manipulating, configuring, and

A thin client is a lightweight computer that has been optimized for establishing a remote connection with a server-based computing environment. The server does most of the work.
This contrasts with a fat client or a conventional personal computer; the former is also intended for working in a client–server model but has significant local processing power, while the latter aims to perform its function mostly locally.



computing overcomes platform dependency issues. Hence, the Cloud Computing is making our business application mobile and collaborative.

::.. Introduction to Cloud (3/4)

There are several different deployment models:

1. The Public Cloud allows systems and services to be easily accessible to the general public. Public cloud may be less secure because of its openness, e.g., e-mail.

::.. Introduction to Cloud (3/4)

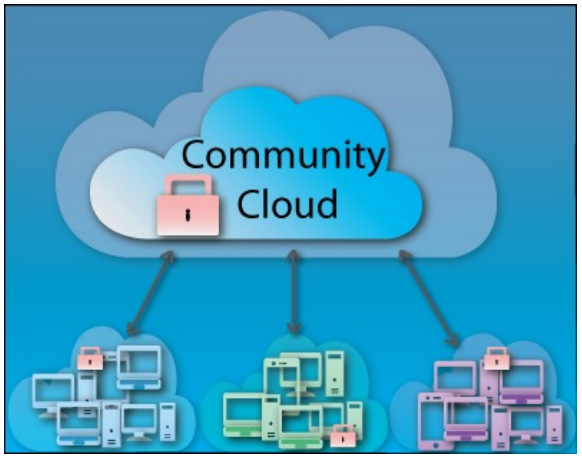
There are several different deployment models:

1. The Public Cloud allows systems and services to be easily accessible to the general public. Public cloud may be less secure because of its openness, e.g., e-mail.
2. The Private Cloud allows systems and services to be accessible within an organization. It offers increased security because of its private nature.

::.. Introduction to Cloud (3/4)

There are several different deployment models:

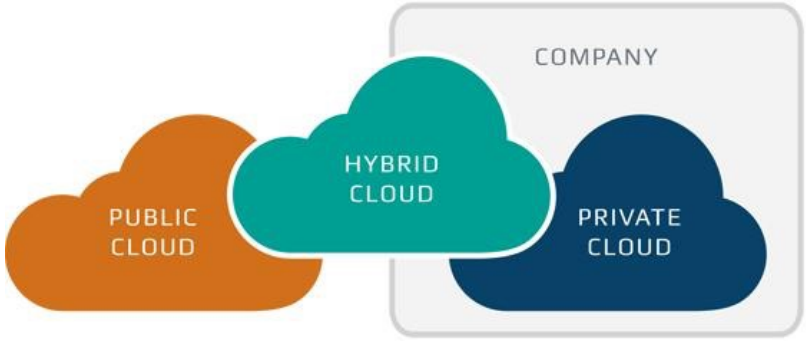
1. The Public Cloud allows systems and services to be easily accessible to the general public. Public cloud may be less secure because of its openness, e.g., e-mail.
2. The Private Cloud allows systems and services to be accessible within an organization. It offers increased security because of its private nature.
3. The Community Cloud allows systems and services to be accessible by group of organizations.



::.. Introduction to Cloud (3/4)

There are several different deployment models:

1. The Public Cloud allows systems and services to be easily accessible to the general public. Public cloud may be less secure because of its openness, e.g., e-mail.
2. The Private Cloud allows systems and services to be accessible within an organization. It offers increased security because of its private nature.
3. The Community Cloud allows systems and services to be accessible by group of organizations.
4. The Hybrid Cloud is mixture of public and private cloud. However, the critical activities are performed using private cloud while the non-critical activities are performed using public cloud.

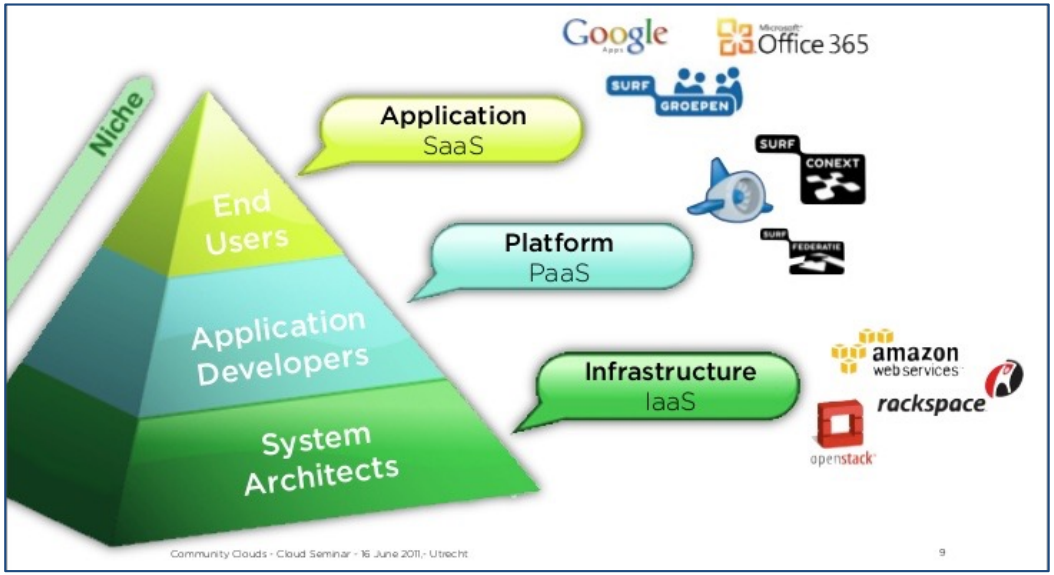


::.. Introduction to Cloud (4/4)

Cloud computing makes it possible to render several services, which can be defined according to the roles, service providers and the user companies. Cloud computing models and services are broadly classified as below:

- The Infrastructure As A Service (IAAS) means the outsourcing of the physical infrastructure of IT (network, storage, and servers) from a third party provider. The IT resources are hosted on external servers and users can access them via an internet connection.

The Benefits are



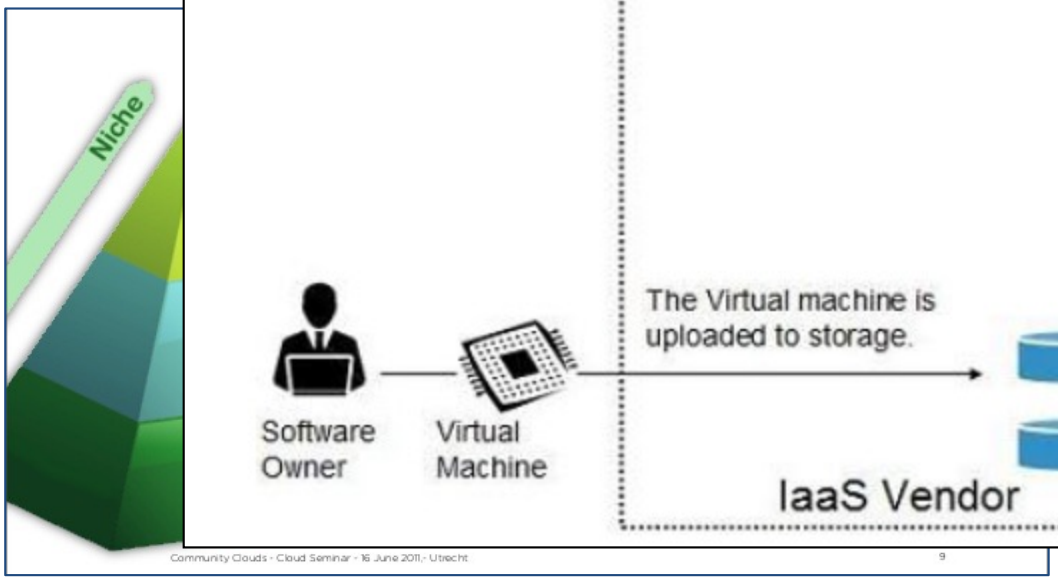
- Time and cost savings: more installation and maintenance of IT hardware in-house,
- Better flexibility: On-demand hardware resources that can be tailored to your needs,
- Remote access and resource management.

::.. Introduction to Cloud (4/4)

Cloud computing makes it possible to render several services, which can
com
class

- T
  p
  t
  a

The



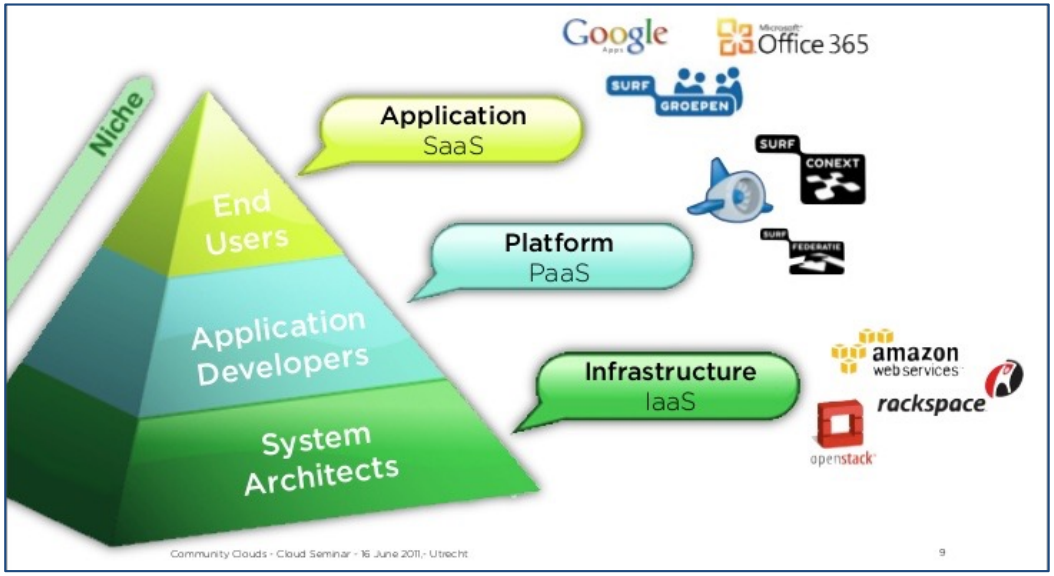
the user
broadly
g of the
) from a
l servers
savings:
and
f IT
: On-
ardware
can be
ds,
and
resource management.

::.. Introduction to Cloud (4/4)

Cloud computing makes it possible to render several services, which can be defined according to the roles, service providers and the user companies. Cloud computing models and services are broadly classified as below:

- Platform as a Service (PAAS) allows outsourcing of hardware infrastructure as well software environment, which includes databases, integration layers, runtimes and more.

The Benefits are



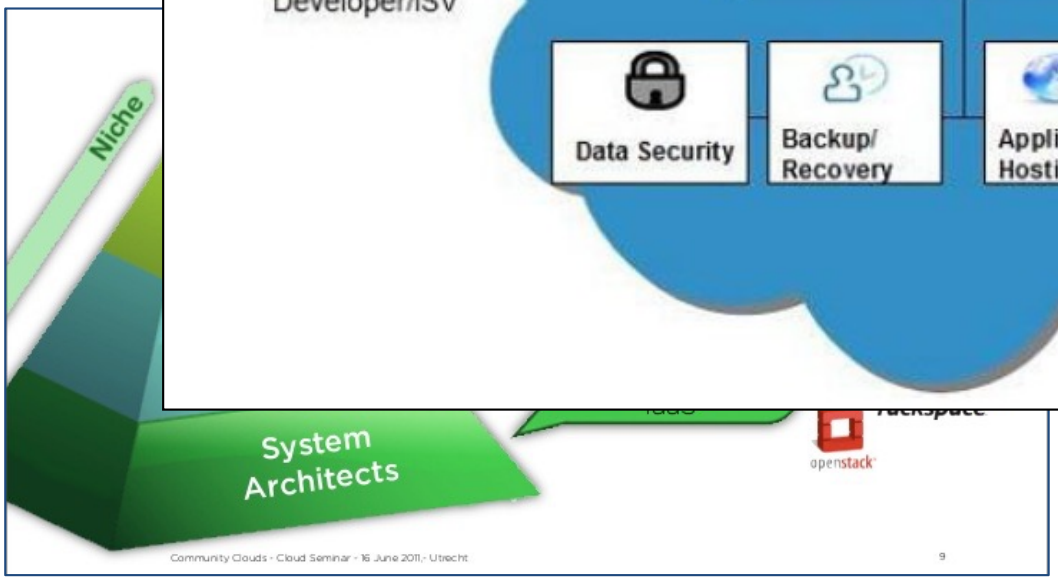
- Mastering the installation and development of software applications;
- No need to manage the platform;
- Data security: control on the distribution, protection, and backup of data.

::.. Introduction to Cloud (4/4)

Cloud computing makes it possible to render several services, which can
and the user
com
are broadly
clas

- P
of hardware
i
ch includes
c

The



installation
pment of
ations;

manage the

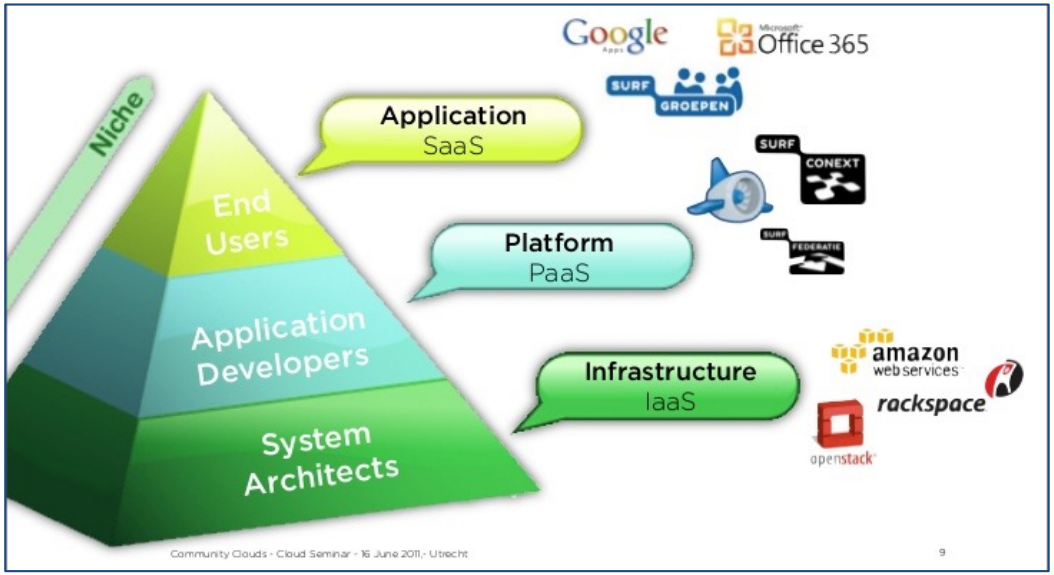
- Data security: control on the distribution, protection, and backup of data.

::.. Introduction to Cloud (4/4)

Cloud computing makes it possible to render several services, which can be defined according to the roles, service providers and the user companies. Cloud computing models and services are broadly classified as below:

- The Software as a Service (SaaS) is provided over the internet and requires no prior installation. These services can be availed from any part of the world at a minimal per month fee.

The Advantages are:



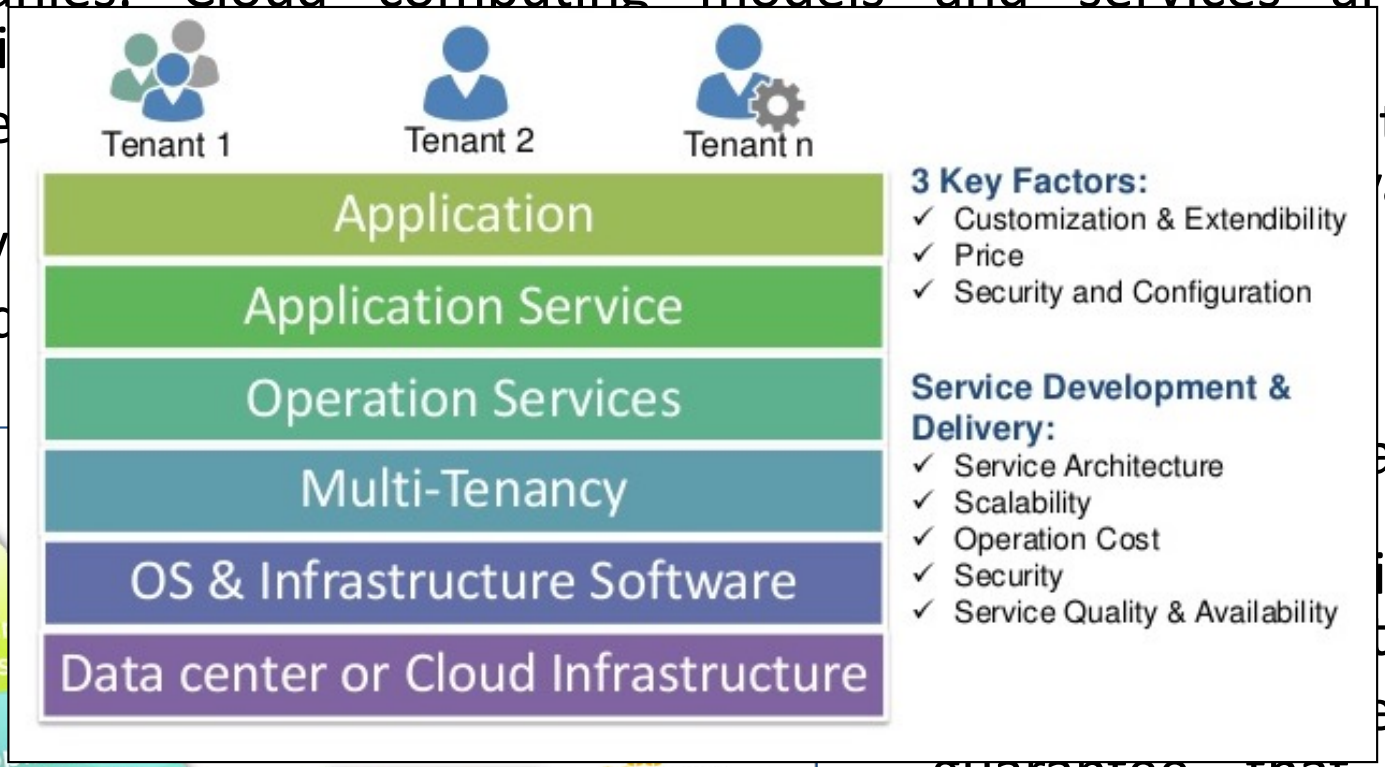
- Users are free from the infrastructure management and aligning software environment: no installation or software maintenance.
- Automatic updates with the guarantee that all users have the same software version.
- Easy and quicker testing of new software solutions.

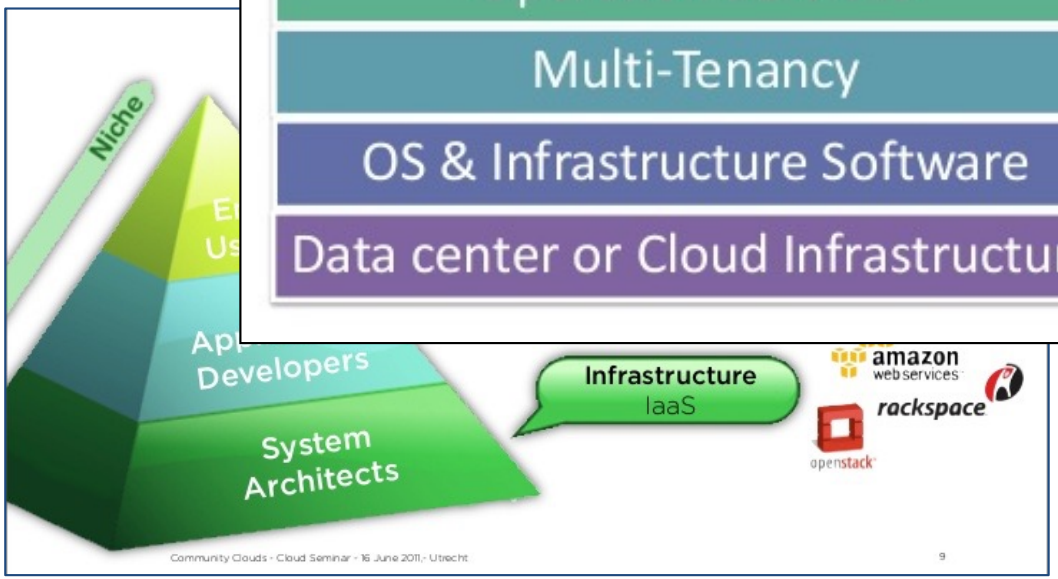
::.. Introduction to Cloud (4/4)

Cloud computing makes it possible to render several services, which can be defined according to the roles, service providers and the user companies. Cloud computing models and services are broadly classifi

- The ternet and req ailed from any

The Ac





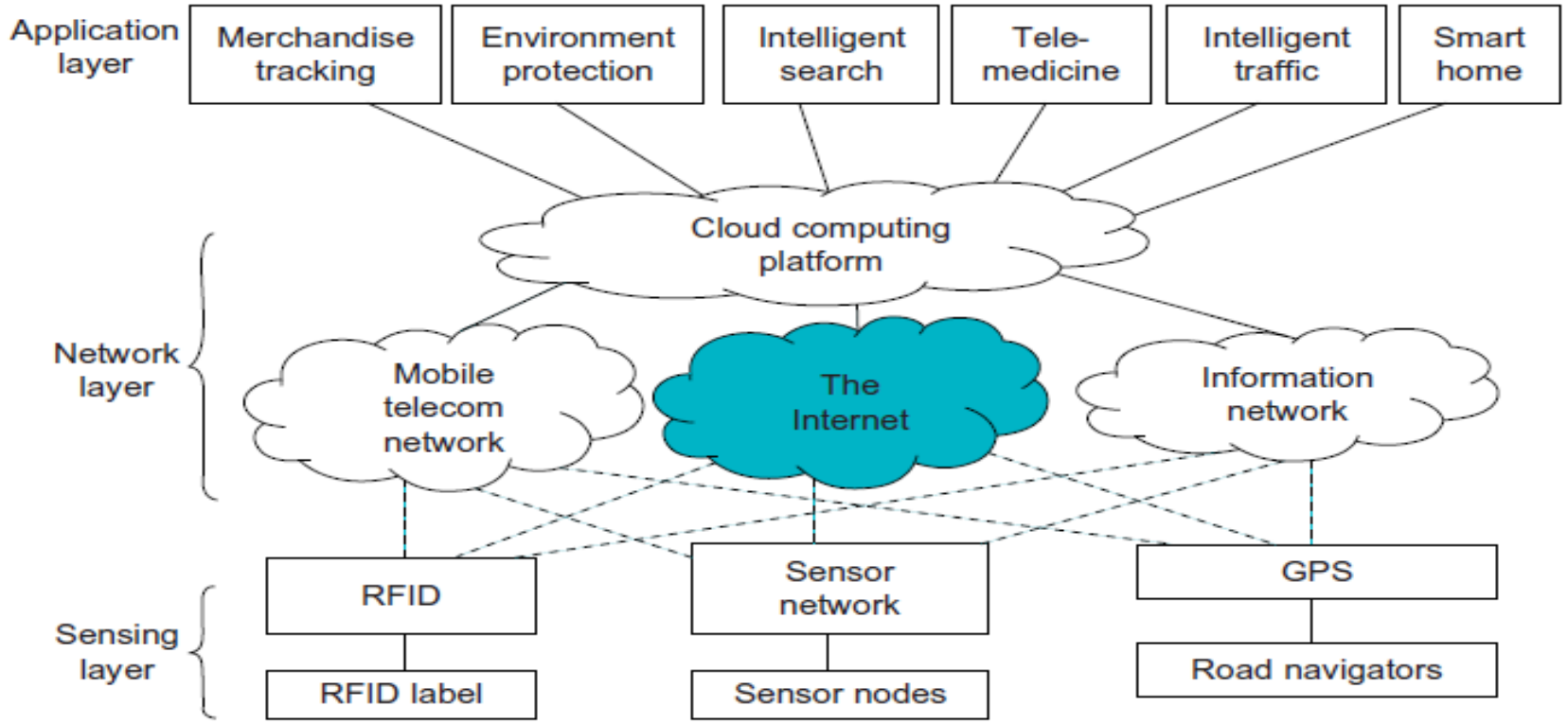
from the anagement software installation tenance.

es with the guarantee that all users have the same software version.

- Easy and quicker testing of new software solutions.

::.. Cloud 4 IoT (1/5)

Information processing can be done more efficiently on large farms of computing and storage systems accessible via the Internet and hosted within the cloud platforms.



::.. Cloud 4 IoT (2/5)

Cloud Platforms form another important computational part of the IoT. These platforms provide facilities

- for smart objects to send their data to the cloud,
- for big data to be processed in real-time,
- eventually for end-users to benefit from the knowledge extracted from the collected big data.

There are a lot of free and commercial cloud platforms and frameworks available to host IoT services.

The cloud provides to the IoT solutions a new management mechanism for big data that enables the processing of data and the extraction of valuable knowledge from it.

IoT can utilize numerous cloud platforms with different capabilities, such as ThingWorx, OpenIoT, Google Cloud, Amazon, GENI, etc.

::.. Cloud 4 IoT (3/5)

The available platforms for IoT can be assessed according to different dimensions or measures of interest:

- supporting gateway devices to bridging the short range network and wide area network,
- supporting discovery, delivery, configuration and activation of applications and services,
- providing proactive and reactive assurance of platform,
- supporting of accounting and billing of applications and services,
- exploiting of standard application protocols.

All the platforms typically support sensing or actuating devices, a user interface to interact with devices, and a web component to run the business logic of the application on the cloud.

::.. Cloud 4 IoT (4/5)

Platform	Gateway	Provision	Assurance	Billing	Application Protocol			
					REST	CoAP	XMPP	MQTT
Arkessa	-	+	+	-	+	-	-	+
Axeda	+	+	+	+	+	-	-	-
Etherios	+	+	+	-	+	-	-	-
LittleBits	-	-	-	-	+	-	-	-
NanoService	+	+	+	-	+	+	-	-
Nimbits	-	-	-	-	+	-	+	-
Ninja Blocks	+	-	-	-	+	-	-	-
OnePlatform	+	+	+	-	+	+	+	-
RealTime.io	+	+	-	-	+	-	-	-
SensorCloud	+	+	-	-	+	-	-	-
SmartThings	+	+	-	-	+	-	-	-
TempoDB	-	-	-	-	+	-	-	-
Thingworx	-	+	+	-	+	-	-	+
Xively	+	+	+	+	+	-	-	+

::.. Cloud 4 IoT (5/5)

Employing the cloud for the IoT is not an easy task due to the following challenges:

- Synchronization between different cloud vendors presents a challenge to provide real-time services since services are built on top of various cloud platforms.
- Standardizing the cloud also presents a significant challenge for IoT cloud-based services due having to interoperate with the various vendors.
- Making a balance between general cloud service environments and IoT requirements presents another challenge due to the differences in infrastructure.
- Security of IoT cloud-based services presents another challenge due to the differences in the security mechanisms between the IoT devices and the cloud platforms.
- Managing the cloud and IoT systems is also a challenging factor due to the fact that both have different resources and components.
- Validating IoT cloud-based services is necessary to ensure providing good services that meet the customers' expectations.

::.. Xively (1/2)

Xively represents one of the first IoT application hosting service providers allowing sensor data to be available on the web.

Xively aims to connect devices to applications securely in real-time, and provides a PaaS solution for the IoT application developers and service providers, exposing its services via RESTful APIs.

It is able to integrate devices with the platform by ready libraries (such as ARM mbed, Electric Imp and iOS/OSX) and facilitate communication via HTTP(S), Sockets/ Websocket, or MQTT. It could also integrate with other platforms using Java, JS, Python, and Ruby libraries.

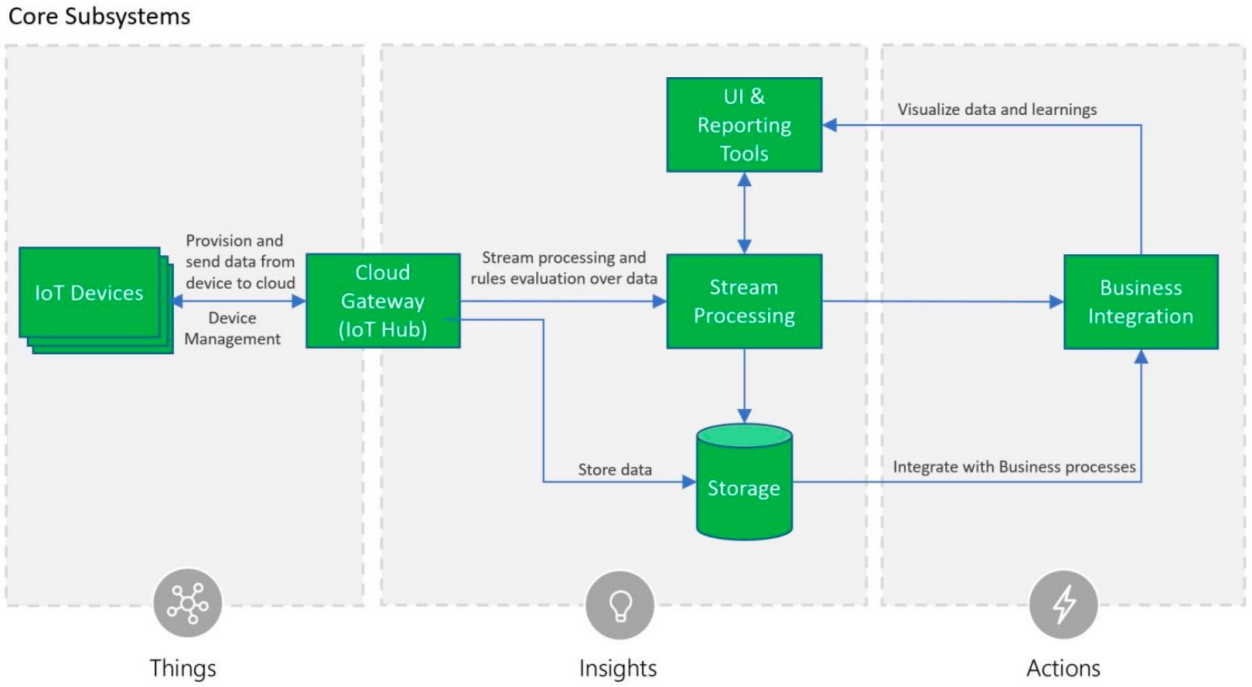
::.. Xively (2/2)

Some of the features that made Xively one of the preferred cloud-based service providers for IoT service offerings are

- Open source, free and easy to use as it exposes accessible Application Programming Interfaces (APIs).
- Interoperability with many protocols, environments and its ability to manage real-time sensors and distribute data in numerous formats such as JSON, XML and CSV.
- Enables users to visualize their data graphically in real- time using a website to monitor activities based on data sensors. Also, it enables users to control sensors remotely by modifying scripts to receive an alert.
- Supported by many Original Equipment Manufacturers (OEM) like Arexx, Nanode, OpenGear, Arduino, and mBed.

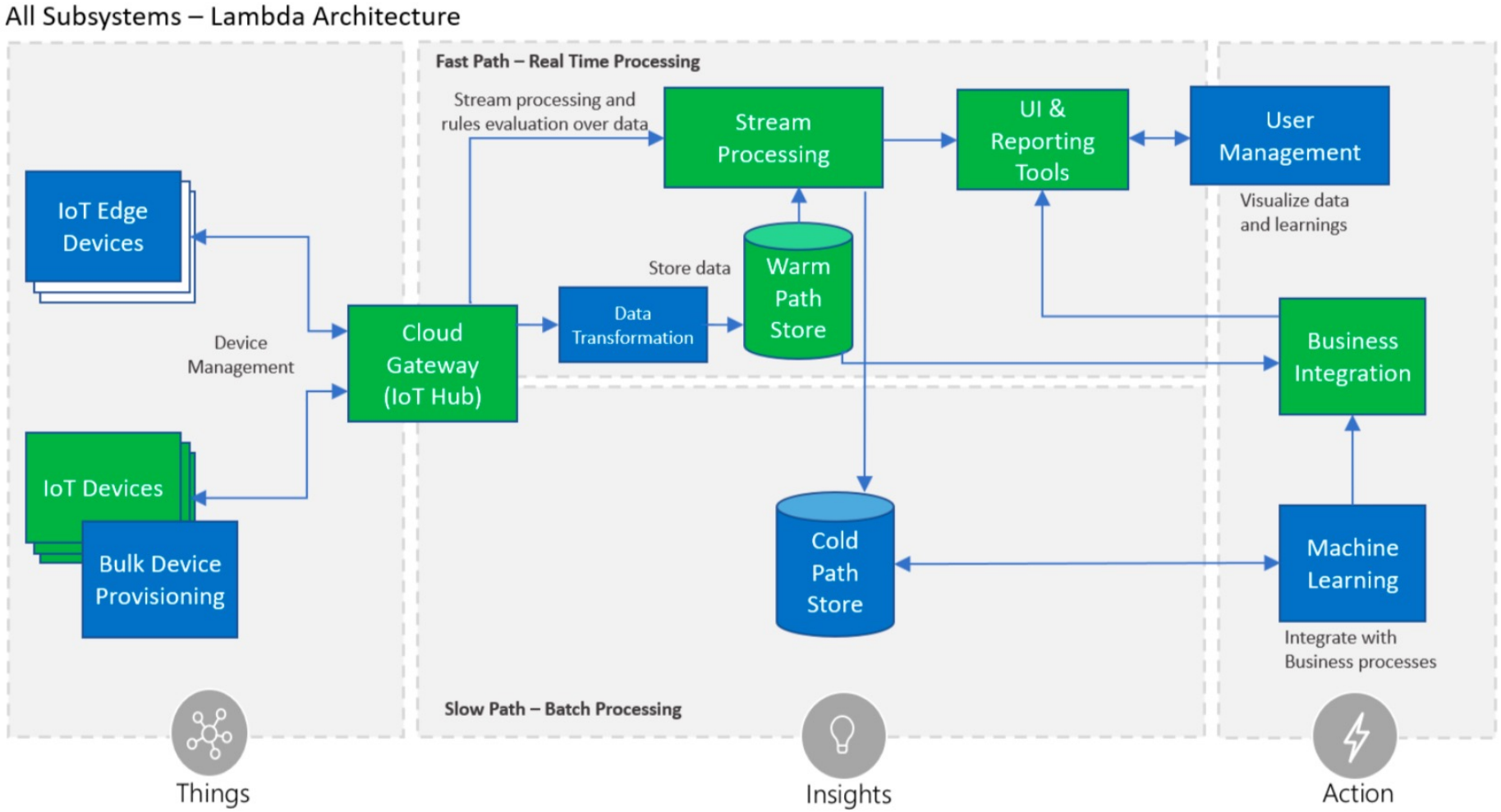
::.. Microsoft Azure IoT Suite (1/5)

It is a platform offering a set of services to build IoT application based on the following concept:



The Cloud Gateway provides a cloud hub for secure connectivity, telemetry and event ingestion and device management (including command and control) capabilities. The Azure IoT Hub service acts as the cloud gateway.

::.. Microsoft Azure IoT Suite (2/5)



Lambda architecture is a data-processing architecture designed to handle massive quantities of data by taking advantage of both batch and stream-processing methods.

::.. Microsoft Azure IoT Suite (3/5)

Here is a list of the services available, as well as what you may use them for.

- IoT Central: This is a SaaS solution that helps you connect, monitor, and manage your IoT devices. To start, you select a template for your device type and create and test a basic IoT Central application that the operators of the device will use. The IoT Central application will also enable you to monitor the devices and provision new devices. This service is for straightforward solutions that don't require deep service customization.
- IoT solution accelerators: This is a collection of PaaS solutions you can use to accelerate your development of an IoT solution. You start with a provided IoT solution and then fully customize that solution to your requirements. You need Java or .NET skills to customize the back-end, and JavaScript skills to customize the visualization.

::.. Microsoft Azure IoT Suite (4/5)

- IoT Hub: This service allows you to connect from your devices to an IoT hub, and monitor and control billions of IoT devices. This is especially useful if you need bi-directional communication between your IoT devices and your back end. This is the underlying service for IoT Central and IoT solution accelerators.
- IoT Hub Device Provisioning Service: This is a helper service for IoT Hub that you can use to provision devices to your IoT hub securely. With this service, you can easily provision millions of devices rapidly, rather than provisioning them one by one.
- IoT Edge: This service builds on top of IoT Hub. It can be used to analyze data on the IoT devices rather than in the cloud. By moving parts of your workload to the edge, fewer messages need to be sent to the cloud.

::.. Microsoft Azure IoT Suite (5/5)

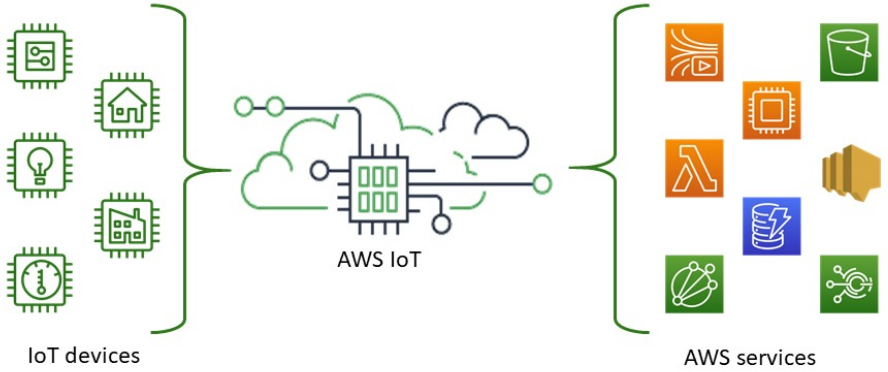
- Azure Digital Twins: This service enables you to create comprehensive models of the physical environment. You can model the relationships and interactions between people, spaces, and devices. For example, you can predict maintenance needs for a factory, analyze real-time energy requirements for an electrical grid, or optimize the use of available space for an office.
- Time Series Insights: This service enables you to store, visualize, and query large amounts of time series data generated by IoT devices. You can use this service with IoT Hub.
- Azure Maps: This service provides geographic information to web and mobile applications. There is a full set of REST APIs as well as a web-based JavaScript control that can be used to create flexible applications that work on desktop or mobile applications for both Apple and Windows devices.

::.. AWS IoT Platform (1/2)

Amazon made it much easier for developers to collect data from sensors and Internet-connected devices. They help you collect and send data to the cloud and analyze that information to provide the ability to manage devices.

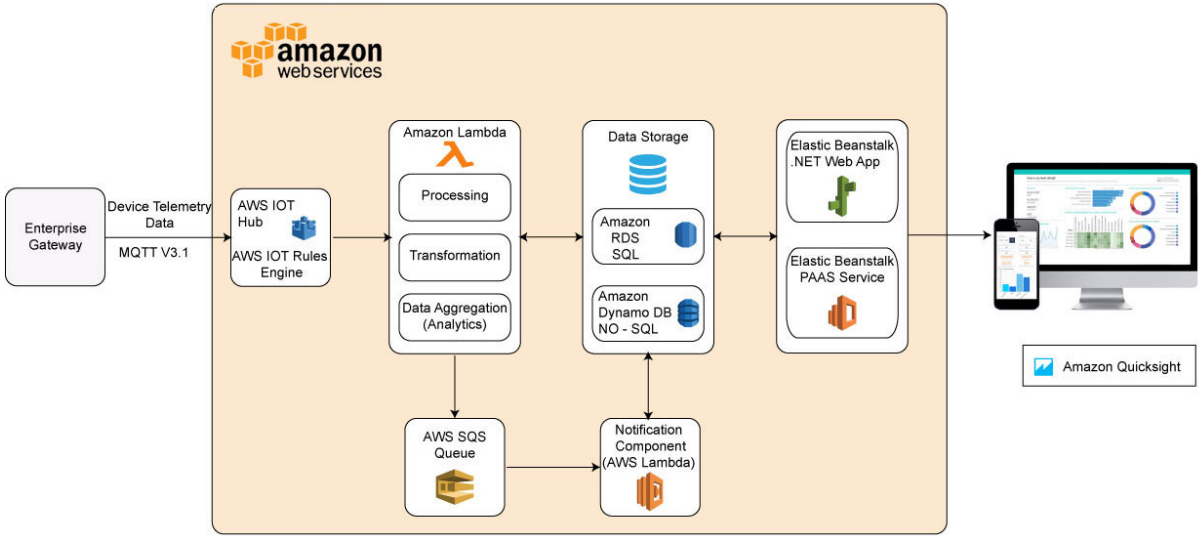
You can easily interact with your application with the devices even if they are offline.

::.. AWS IoT Platform (2/2)



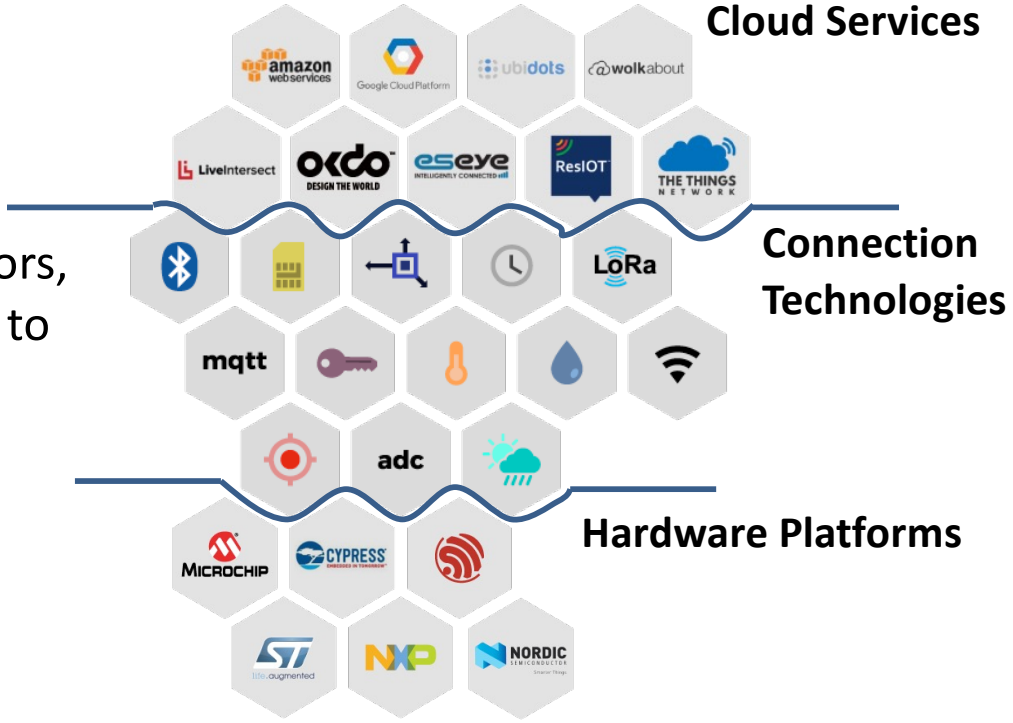
AWS IoT provides device software to integrate IoT devices and a SDK to design code on the IoT device and connect to AWS IoT.

To help managing and supporting IoT devices in the field, AWS IoT communication supports MQTT and offers a set of services.

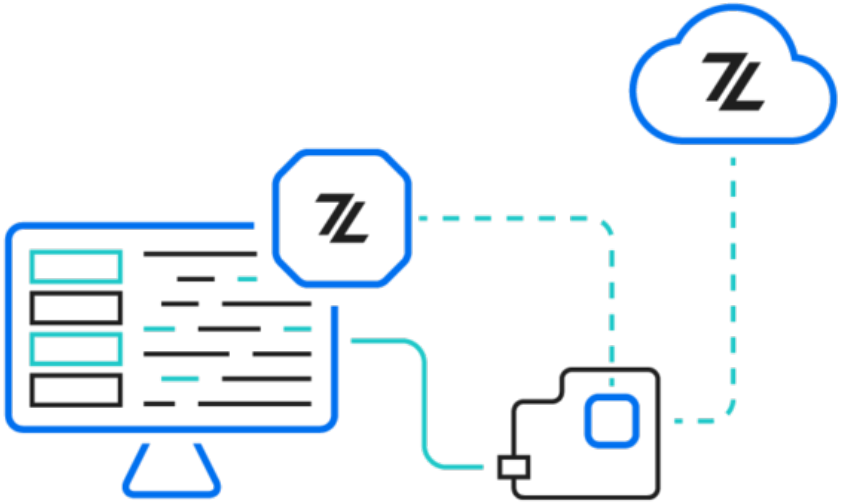


::.. Zerynth

It allows connecting IoT devices directly to the most popular Cloud infrastructures on the market.



Zerynth allows rapid integration with sensors, actuators, and industrial protocols, thanks to an extensive collection of libraries.

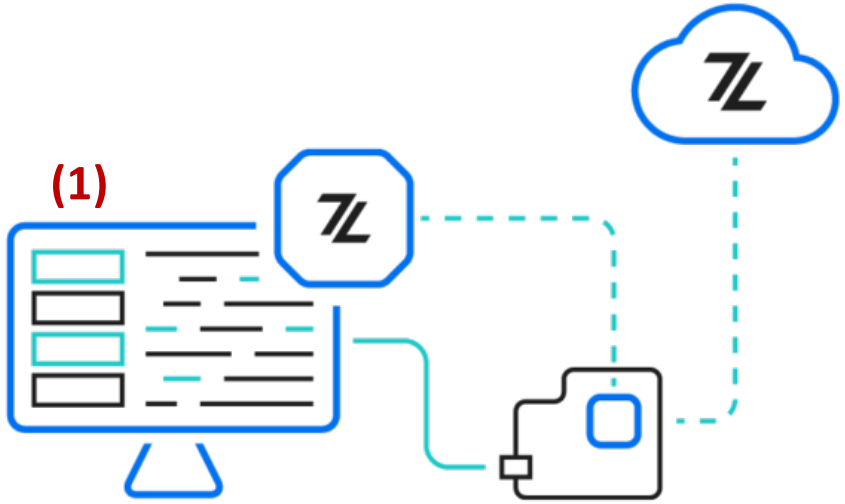


Write once, run everywhere.

::.. Zerynth

Zerynt is a platform made of

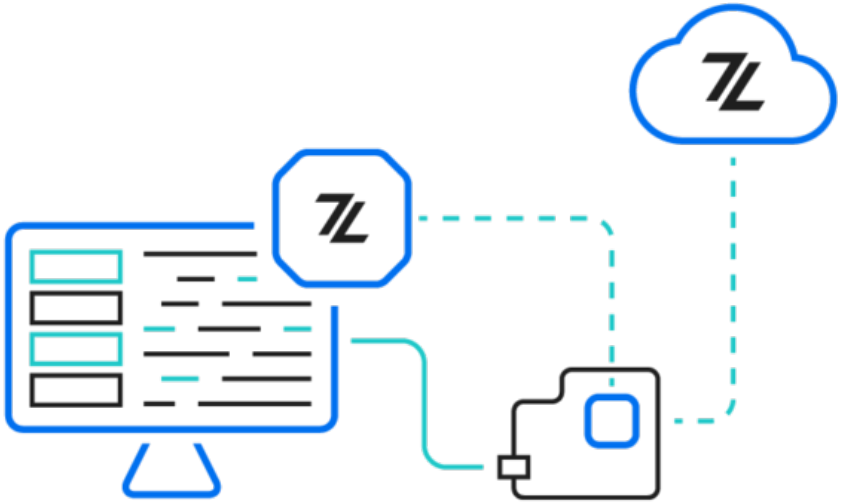
1. A cross-platform SDK optimized for developing Python or C IoT applications and manage device cloud connection, and for the development with Zerynth OS and the management of the Zerynth Device Manager cloud service;



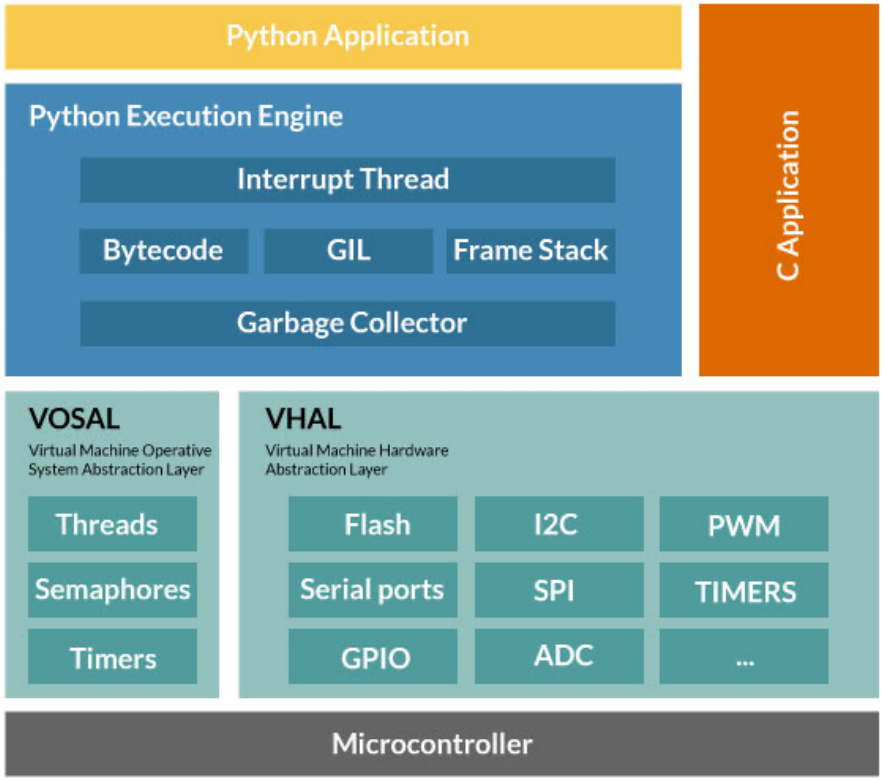
::.. Zerynth

Zerynt is a platform made of

2. A multithreaded Real-Time OS for 32-bit microcontrollers, with very low Flash and RAM requirements;



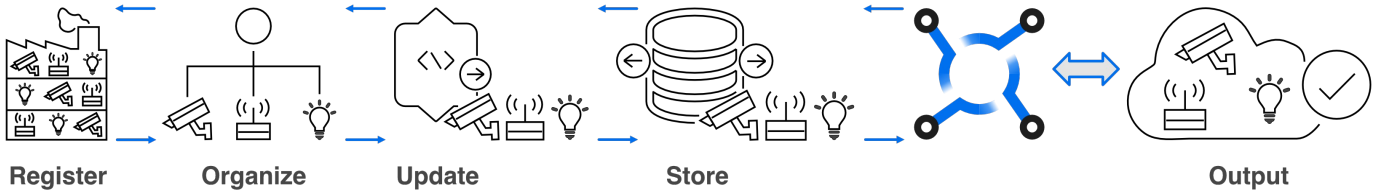
(2)

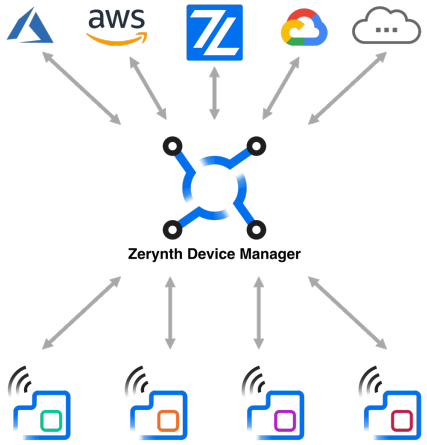


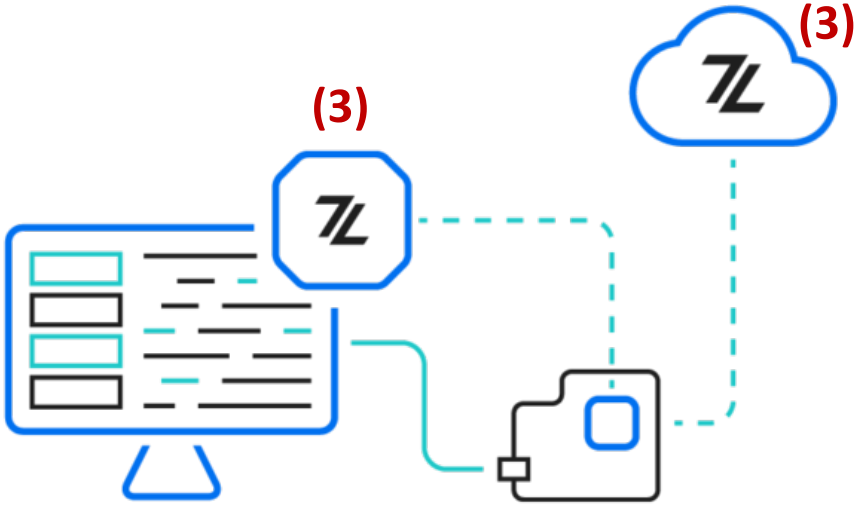
::.. Zerynth

Zerynt is a platform made of

3. A data aggregation and device management service to deploy scalable, and secure IoT solutions by securely registering, organizing, monitor ing, and remotely managing IoT devices at scale.







ZDM integrates with all primary cloud service providers to easily connect devices to the cloud and other devices so to remotely control your an IoT deployment.