UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Lecture 1- Introduction to the Internet of Things and its Hardware

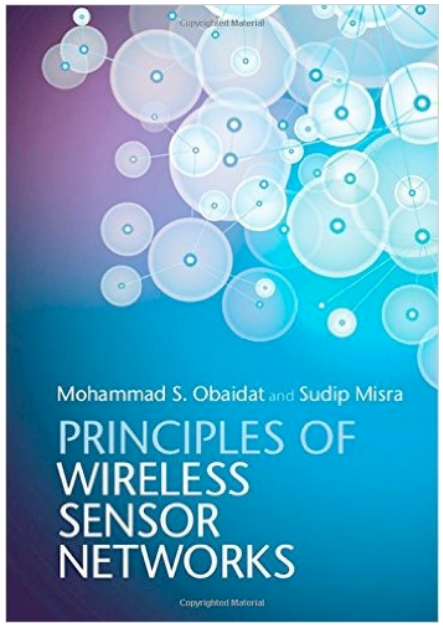
Prof. Esposito Christian

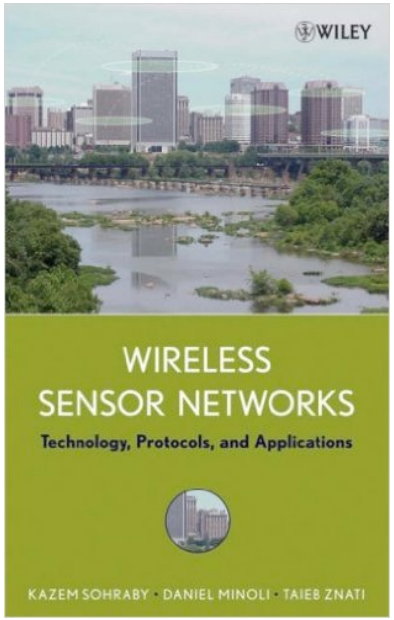
::.. Summary

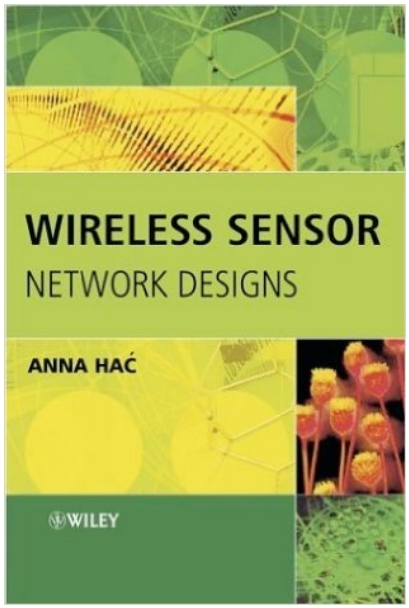
- Introduction to Distributed Sensor Networks and Internet of Things
- Evaluation means for sensors:
  - Emulation;
  - Simulation.
- Anatomy and taxonomy of sensors:
  - Types of Sensors and their Internals & Architectural Stack.
- Sensor hardware:
  - The Hardware Platforms;
  - Ad-hoc and Generic Hardware.

::.. 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)

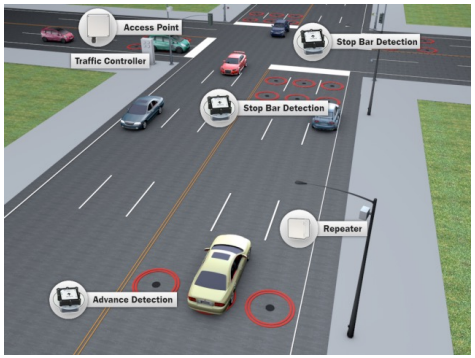
- Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A survey." Computer Networks, vol. 54, no. 15, pp. 2787-2805, October 2010.
- Da Xu, Li, Wu He, and Shancang Li. "Internet of Things in industries: A survey." IEEE Transactions on industrial informatics, vol. 10, no. 4, pp. 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.

Introduction

::.. Introduction (1/2)

A network of distributed sensors (Wireless Sensor Network - WSN) is a new class of wireless networks that became extremely popular with quite a number of civil and military systems with respect to the monitoring and control of physical parameters.

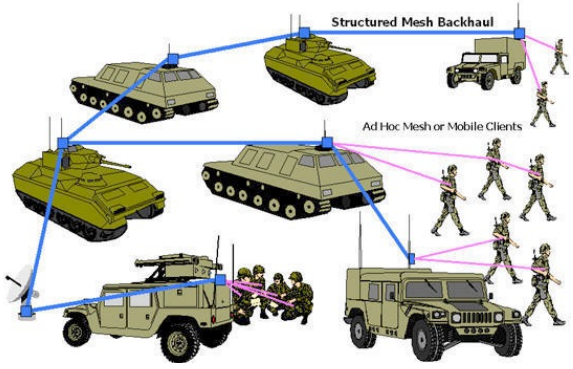
Environmental Monitoring



Smart Roads

Building Monitoring

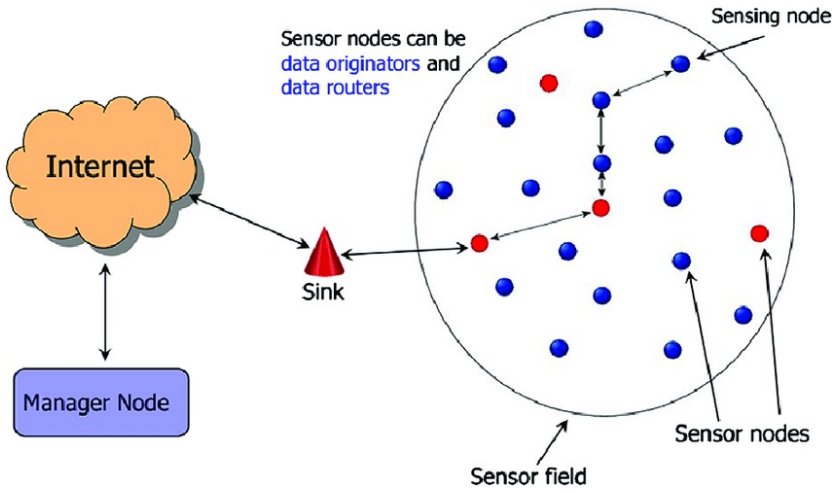
Troop Deployment and Rescue Personnel Control



Smart Agriculture

::.. Introduction (2/2)

A network of distributed sensors contains a series of small intercommunicating and independent sensing devices that are used for monitoring environmental and physical conditions and communicate among each other thanks to a proper network.



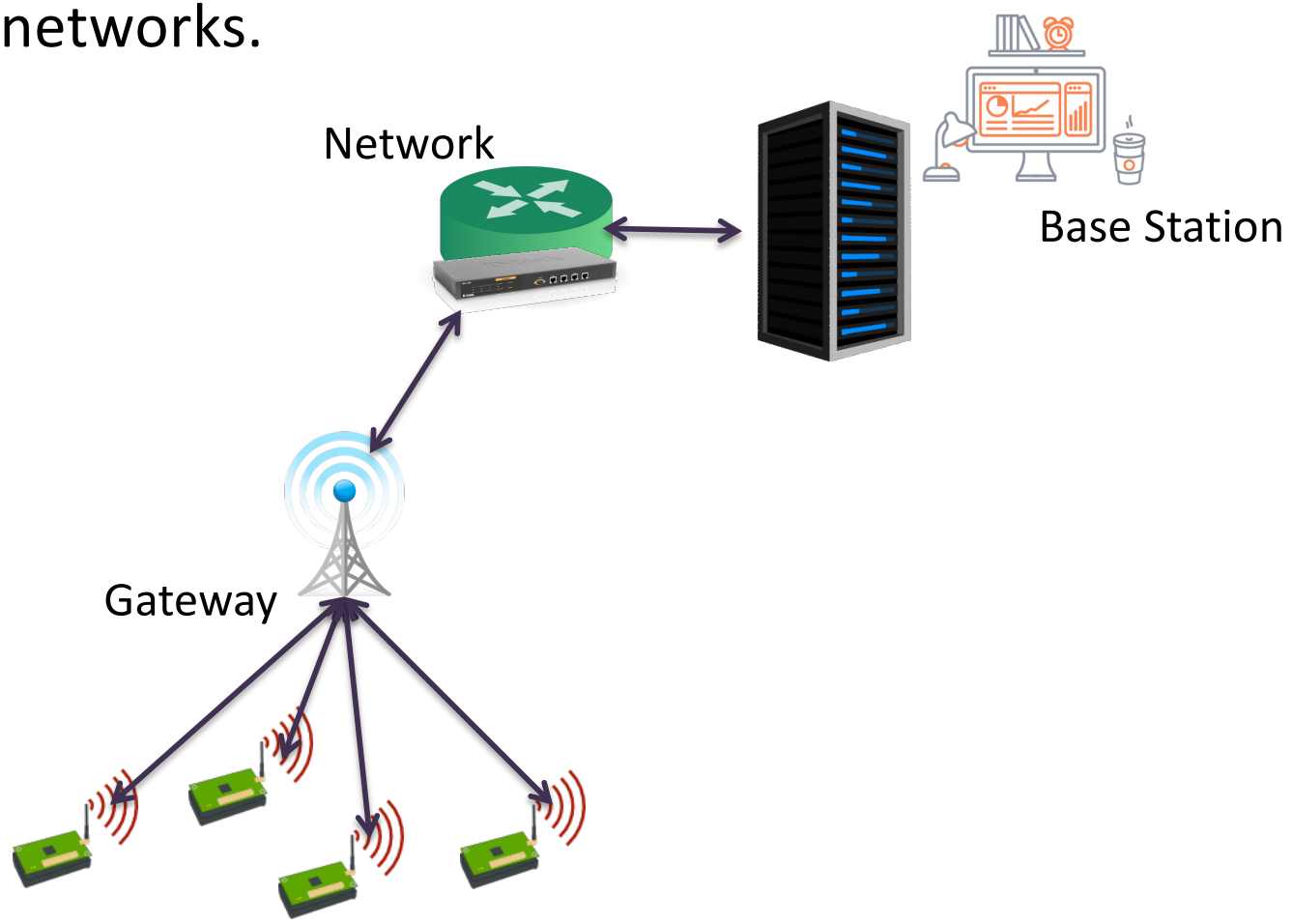
They obtain various kinds of information on the environment, and send them to gateway nodes, passing them to a base station, to activate an alarm or interface with other systems.

What is the Internet of Things (IoT)?

A network of physical objects, which connect and exchange data over the Internet.

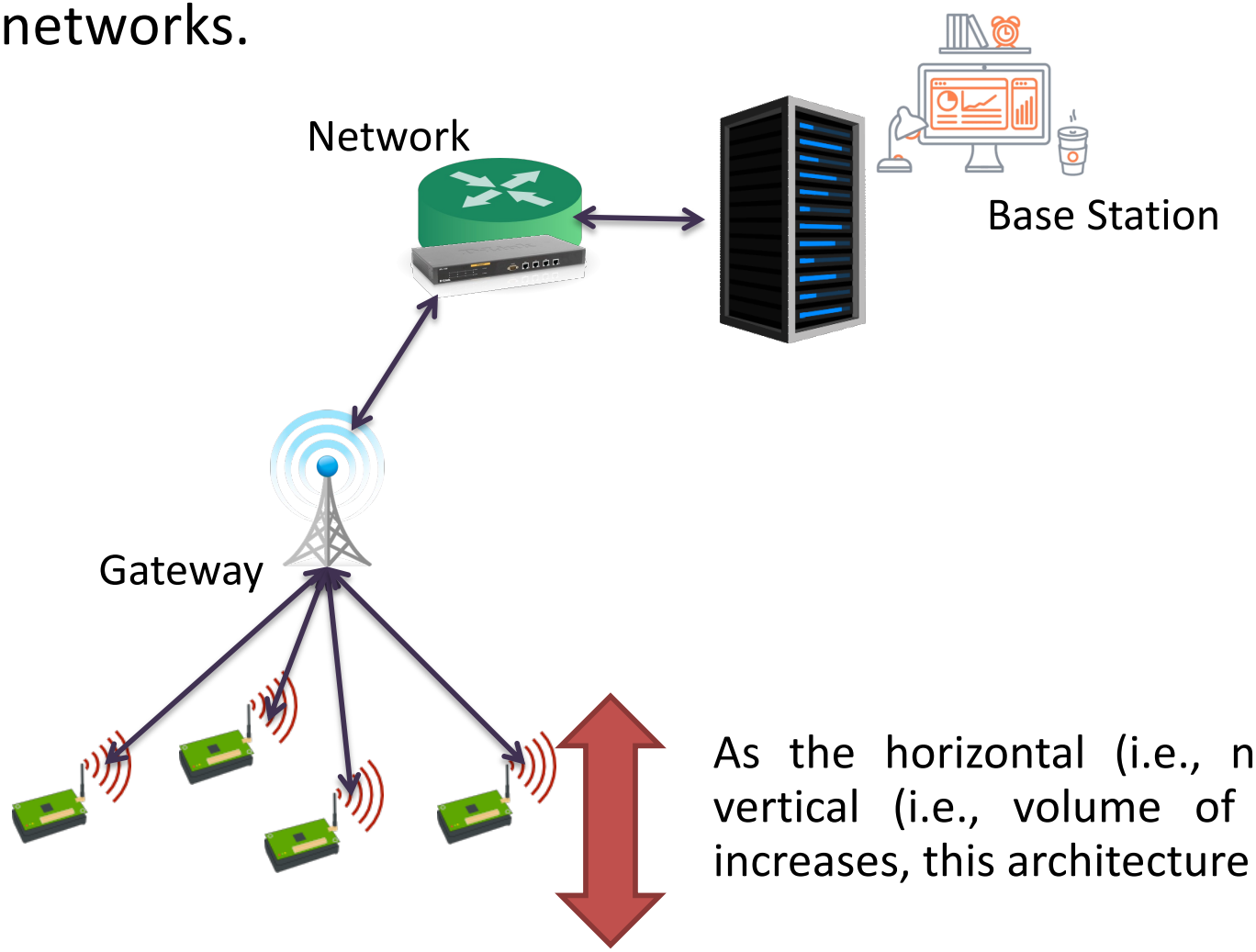
::.. Architectural Evolution

The initial Architecture of WSN is tailored to small/medium size networks.



::.. Architectural Evolution

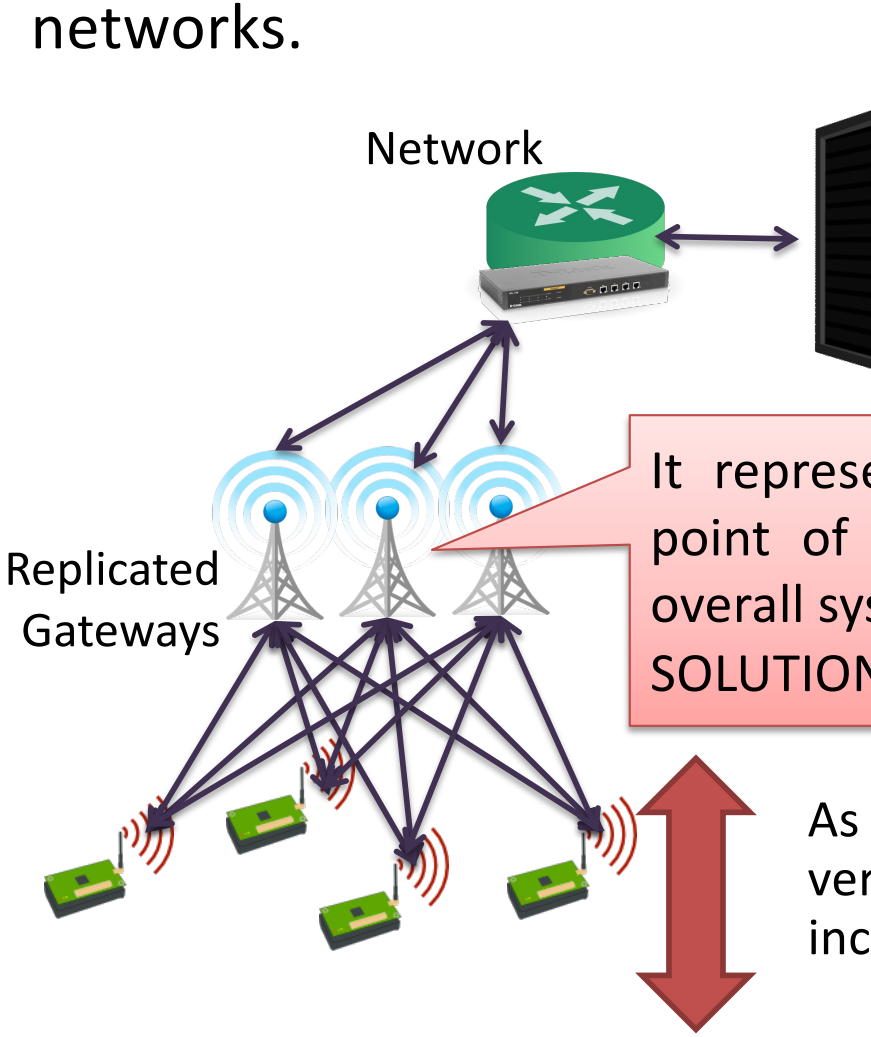
The initial Architecture of WSN is tailored to small/medium size networks.



As the horizontal (i.e., number of nodes) and vertical (i.e., volume of produced data) scale increases, this architecture presents some limits.

::.. Architectural Evolution

The initial Architecture of WSN is tailored to small/medium size networks.

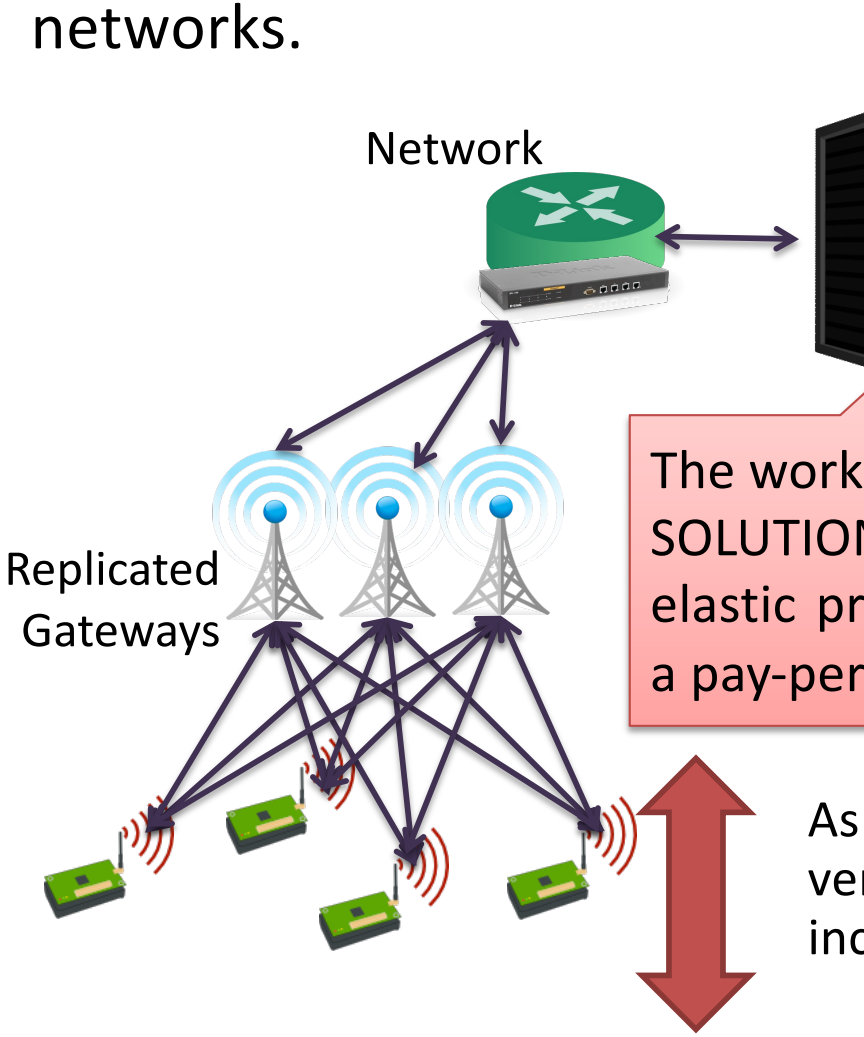


It represents a performance bottleneck, a single-point of failure and a vulnerability point for the overall system.
SOLUTION: Passive or Active Replication.

As the horizontal (i.e., number of nodes) and vertical (i.e., volume of produced data) scale increases, this architecture presents some limits.

::.. Architectural Evolution

The initial Architecture of WSN is tailored to small/medium size networks.

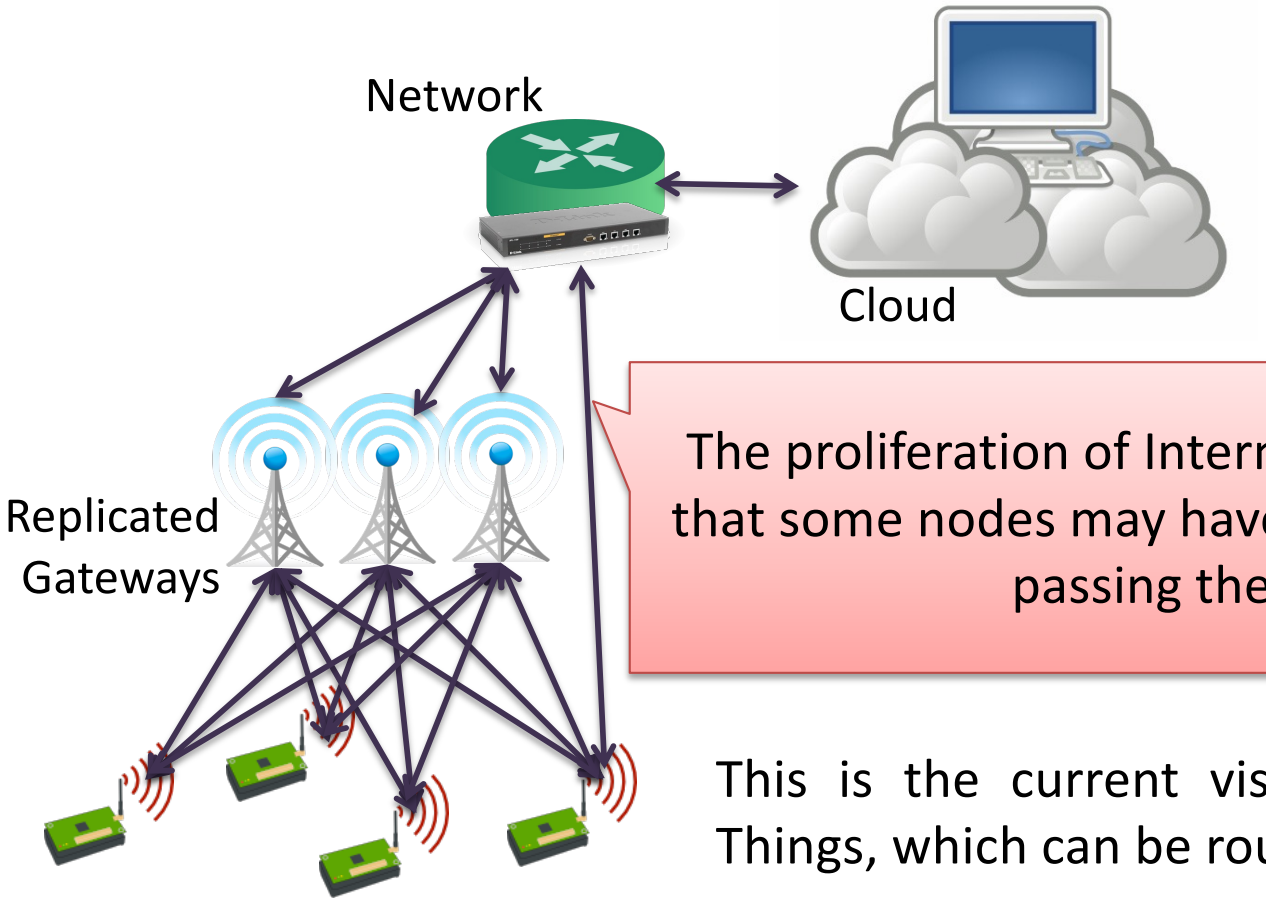


The workload may overwhelm the Base Station.
SOLUTION: Introduction of the Cloud, providing an elastic provisioning of the computing resources and a pay-per-use model.

As the horizontal (i.e., number of nodes) and vertical (i.e., volume of produced data) scale increases, this architecture presents some limits.

::.. Architectural Evolution

The initial Architecture of WSN is tailored to small/medium size networks.



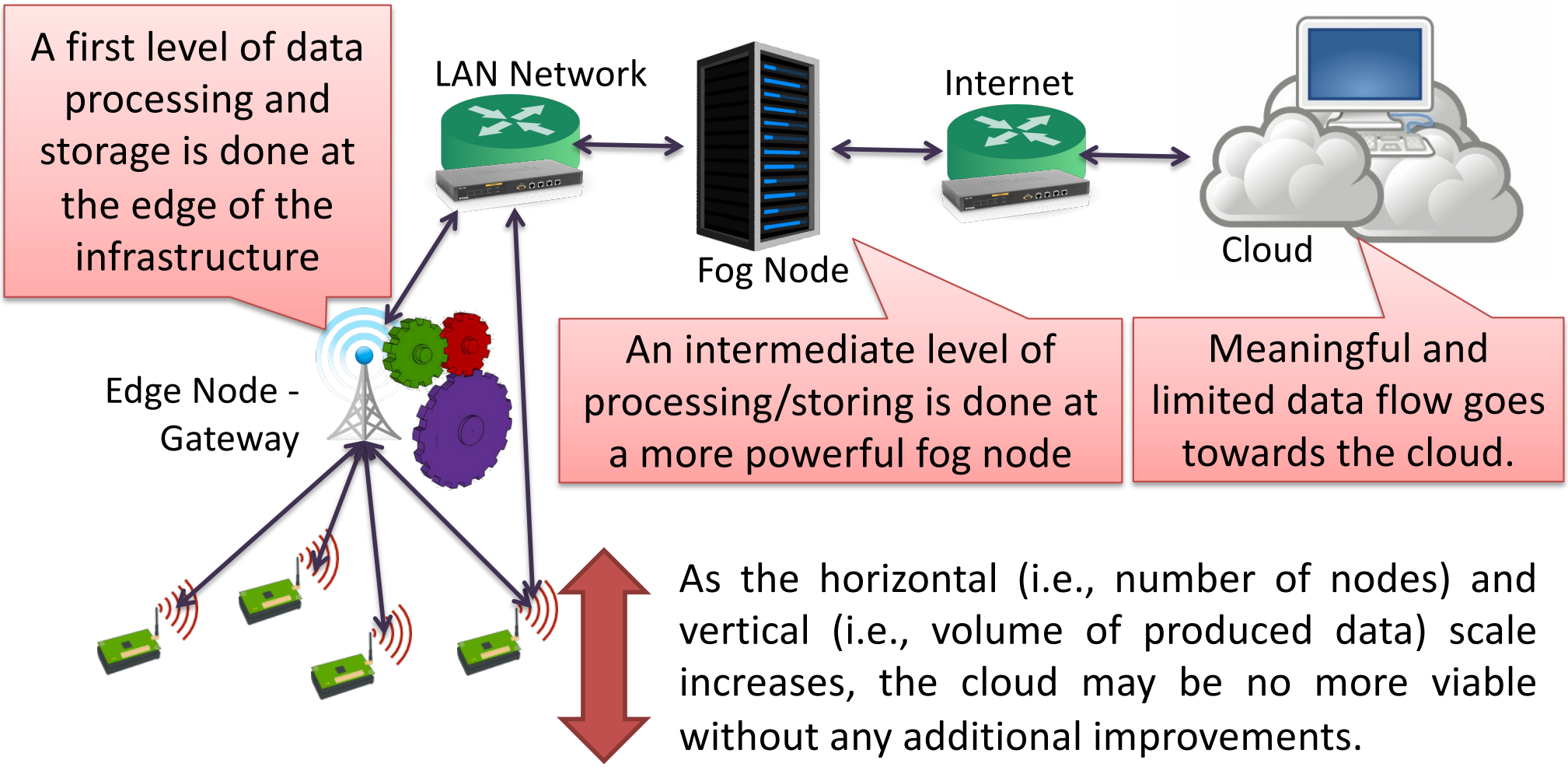
The proliferation of Internet access modes implies that some nodes may have a direct connectivity by-passing the gateways.

This is the current vision of the Internet of Things, which can be roughly defined as

WSN + Cloud

::.. Architectural Evolution

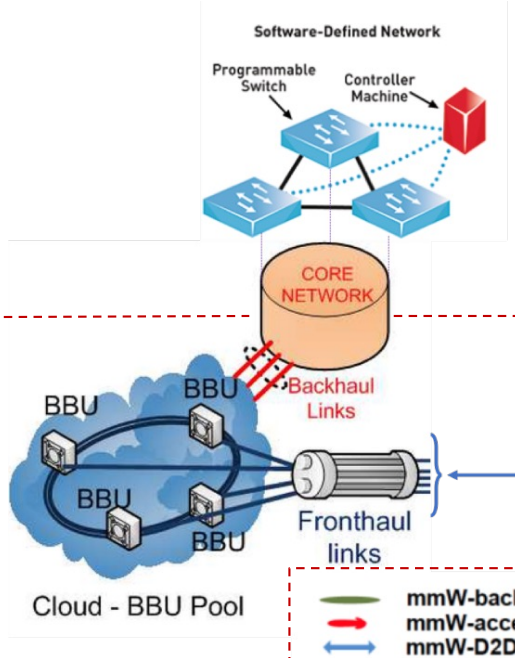
The initial Architecture of WSN is tailored to small/medium size networks.



As the horizontal (i.e., number of nodes) and vertical (i.e., volume of produced data) scale increases, the cloud may be no more viable without any additional improvements.

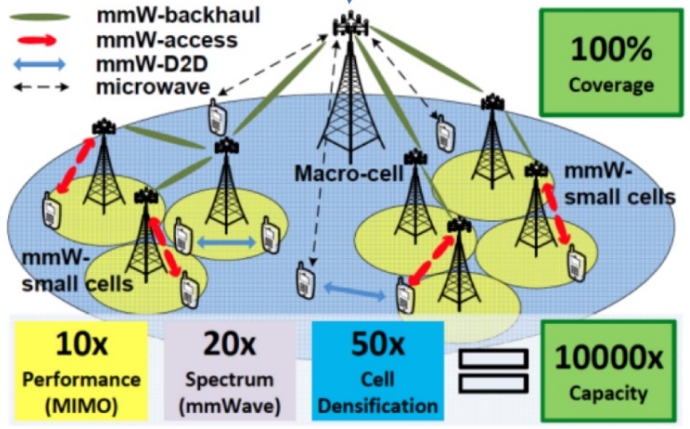
::.. 5G-enabled IoT

The next generation of telcom infrastructures under the 5G umbrella is paving the way for the Edge-Fog-based IoT.



The network softwarization and virtualization (due to SDN and NFV) is paving the way for the integration in the network of fog nodes.

The empowerment of antennas, allowing them to do more than just communication, will enable edge computing.



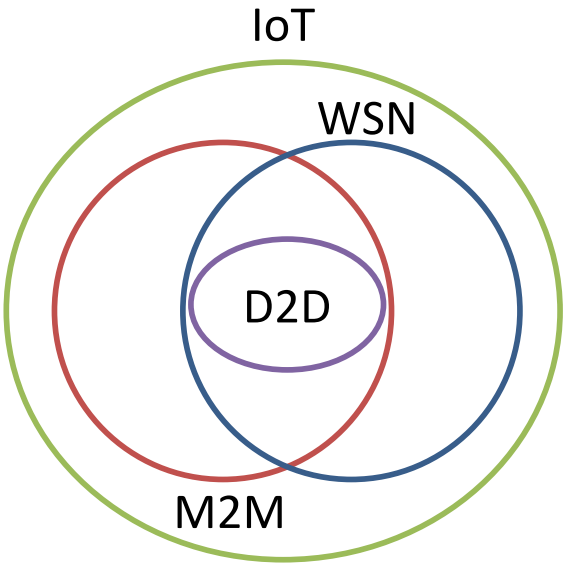
The mmWave and narrowband communications will allow low-latency and high-throughput long range communications.

::.. Overlapping Concepts (1/3)

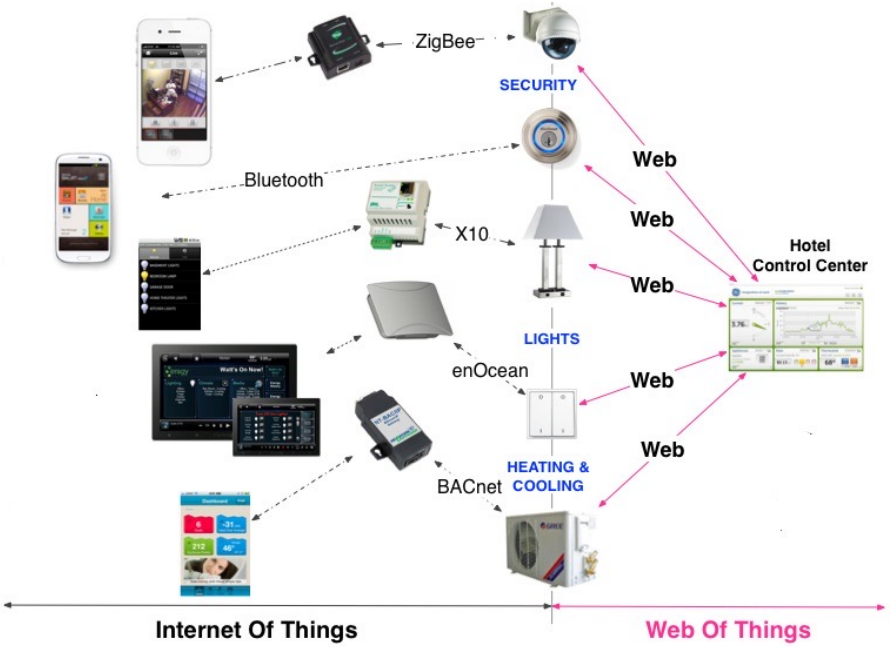
Machine-to-Machine (M2M) communications is a set of methods and protocols to allow devices to communicate and interact over the Internet (or other network) without human intervention.

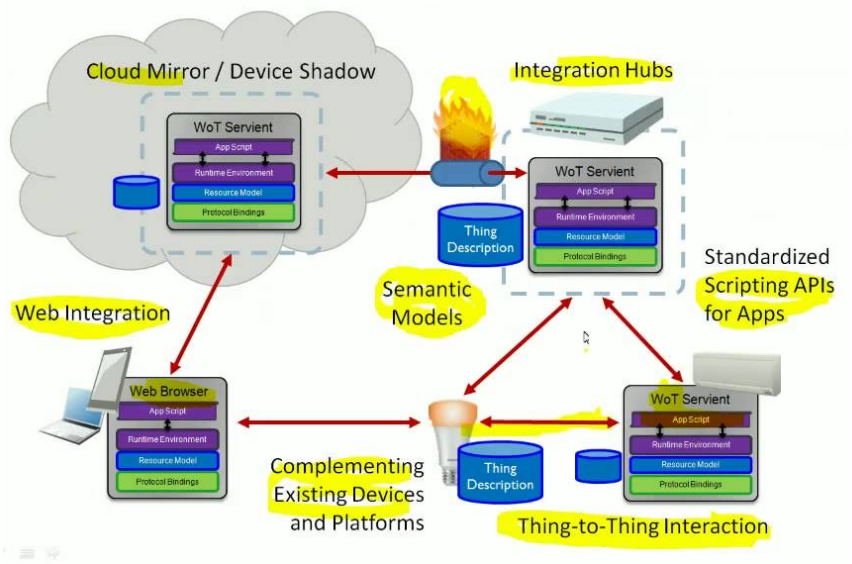
Device-to-Device (D2D) communication is defined as direct communication between two mobile devices without traversing the Base Station (BS) or core network.

M2M	IoT
Point-to-point communication usually embedded within hardware at the customer site	Devices communicate using IP Networks, incorporating with varying communication protocols
Many devices use cellular or wired networks	Data delivery is relayed through a middle layer hosted in the cloud
Devices do not necessarily rely on an Internet connection	In the majority of cases, devices require an active Internet connection
Limited integration options, as devices must have corresponding communication standards	Unlimited integration options, but requires a solution that can manage all of the communications

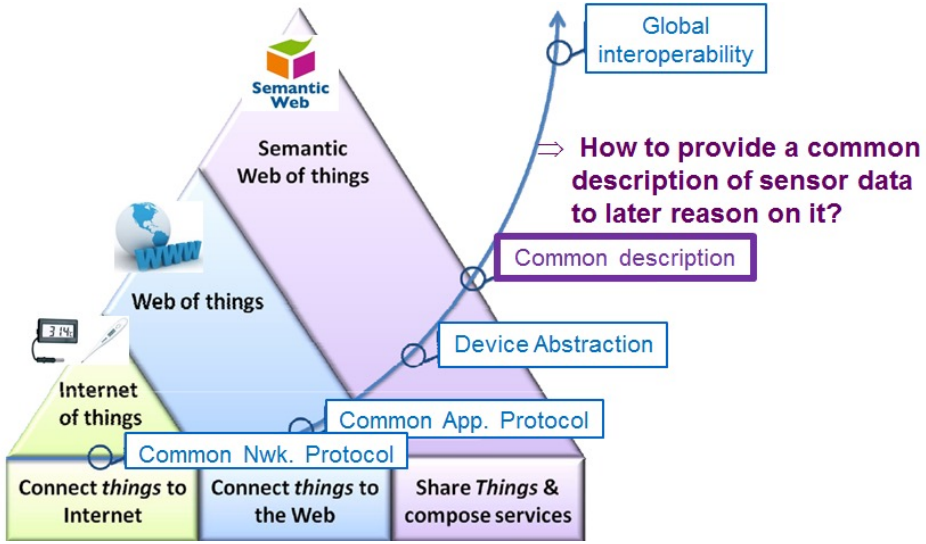


::.. Overlapping Concepts (2/3)



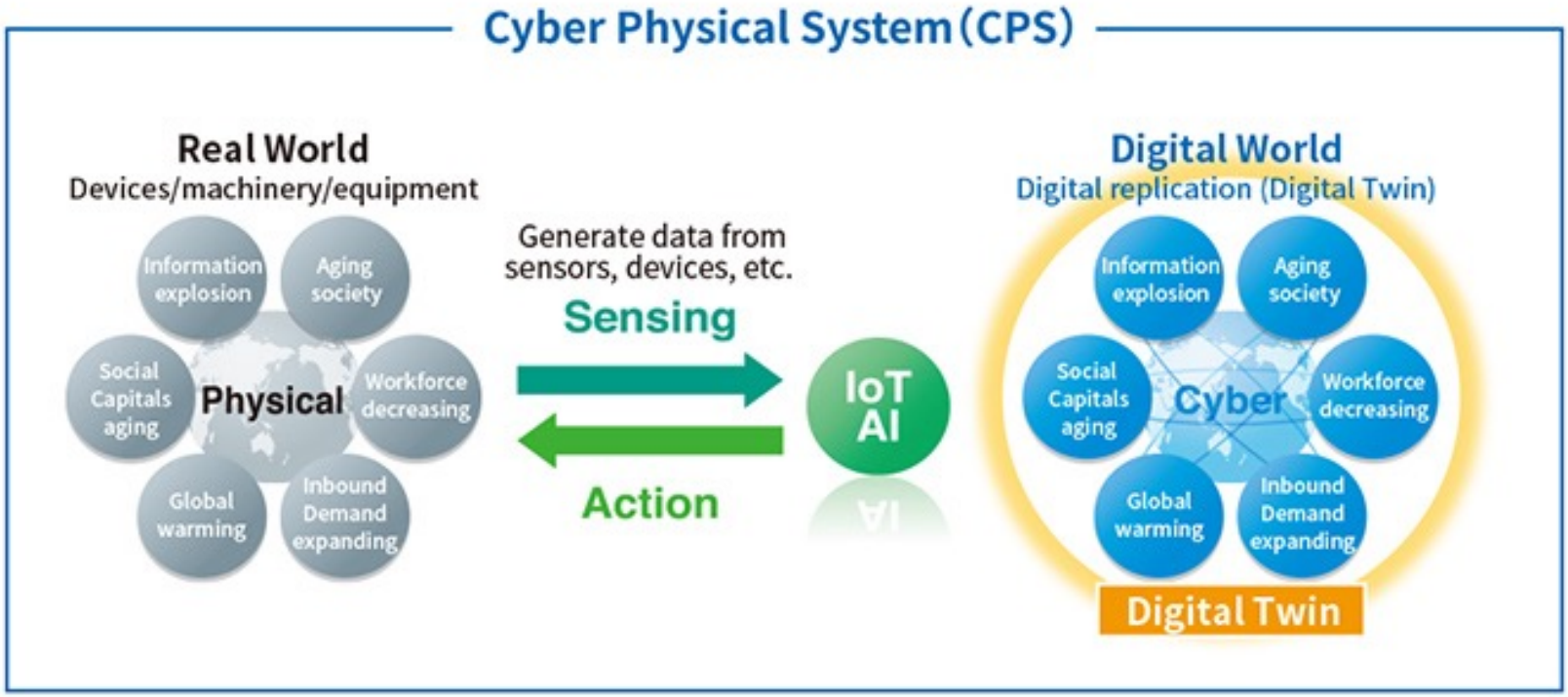


Web of Things (WoT) reuses and leverages readily available and popular Web protocols, standards and blueprints to make data and services offered by objects more accessible and fully integrated with the Web.



::.. Overlapping Concepts (3/3)

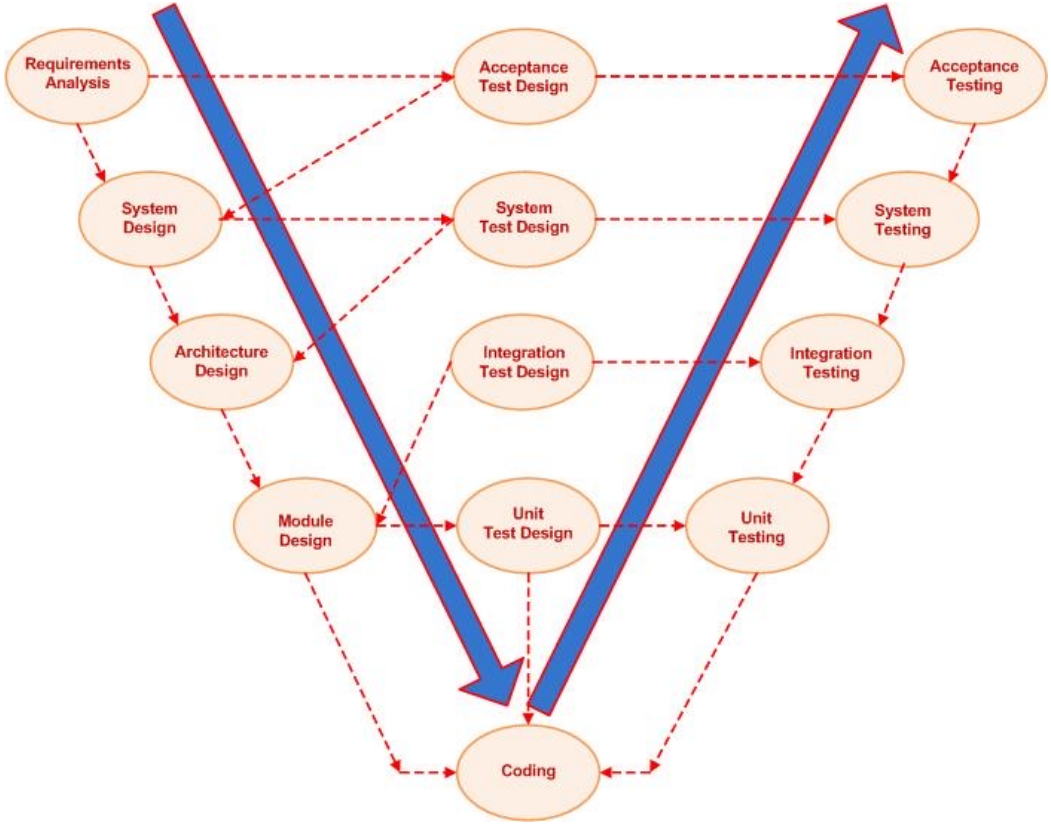
Cyber Physical Systems (CPS) merge real and virtual (cyber) worlds, focusing on systems that based on duly sampled representation of the physical world can intervene through digitized actuators to change behaviours in the physical world.



Simulation/Emulation

::.. System Development

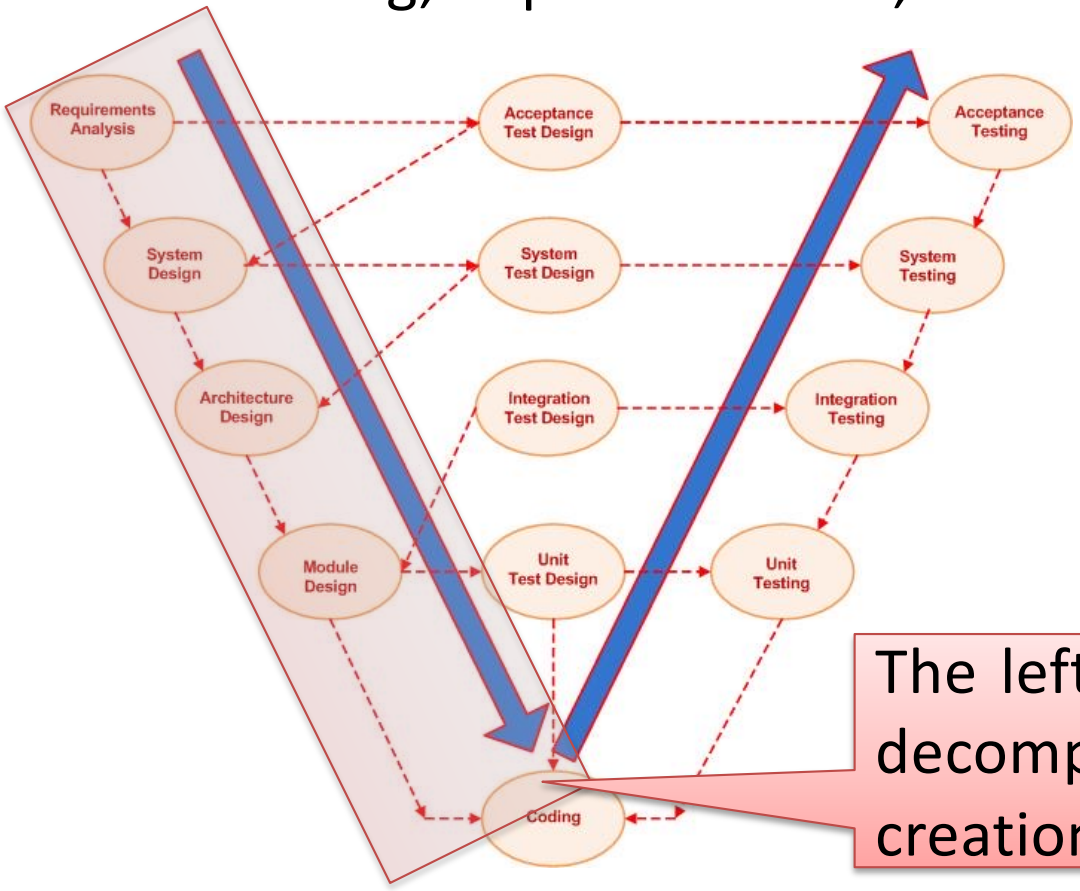
The systems development life cycle is a process for planning, creating, testing, and deploying an information system. There are usually six stages in this cycle: analysis, design, development and testing, implementation, documentation, and evaluation.



The V-model is a graphical representation of a systems development lifecycle, used to produce rigorous development lifecycle models and project management models.

::.. System Development

The systems development life cycle is a process for planning, creating, testing, and deploying an information system. There are usually six stages in this cycle: analysis, design, development and testing, implementation, documentation, and evaluation.

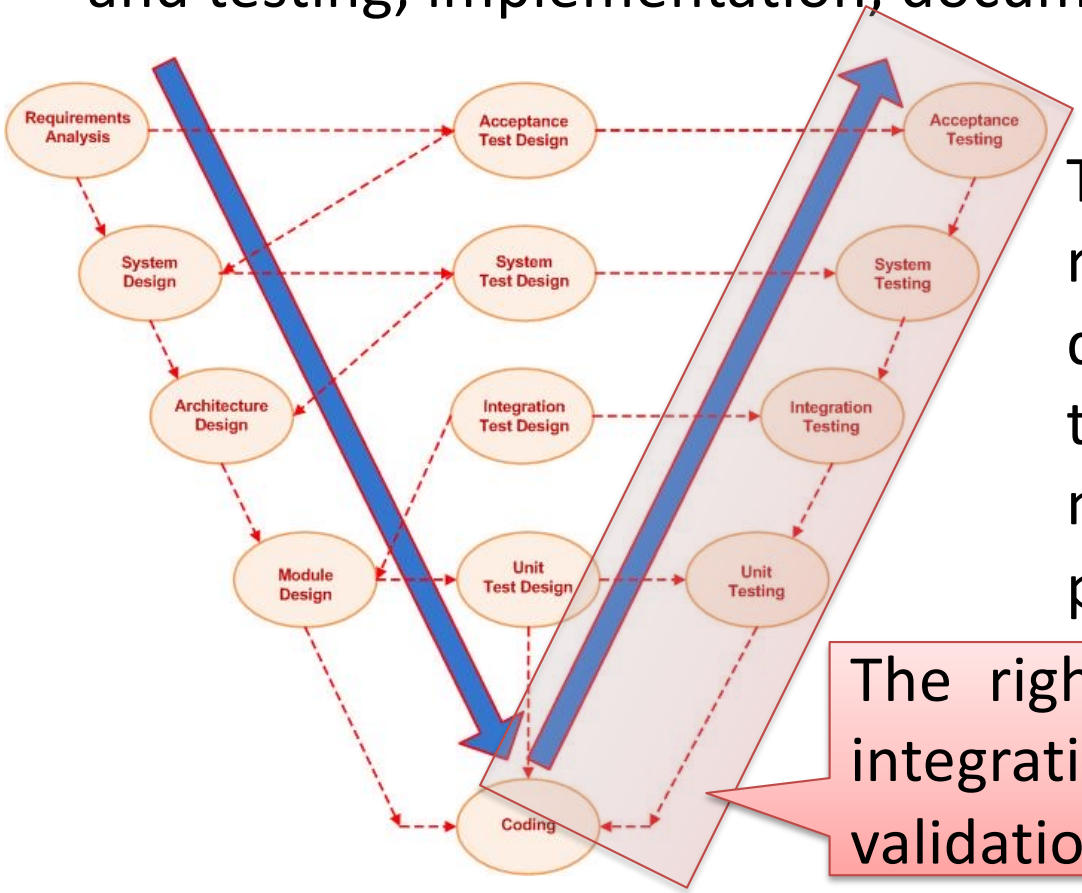


The V-model is a graphical representation of a systems development lifecycle, used to produce rigorous development lifecycle models and project management models.

The left side of the "V" represents the decomposition of requirements, and creation of system specifications.

::.. System Development

The systems development life cycle is a process for planning, creating, testing, and deploying an information system. There are usually six stages in this cycle: analysis, design, development and testing, implementation, documentation, and evaluation.

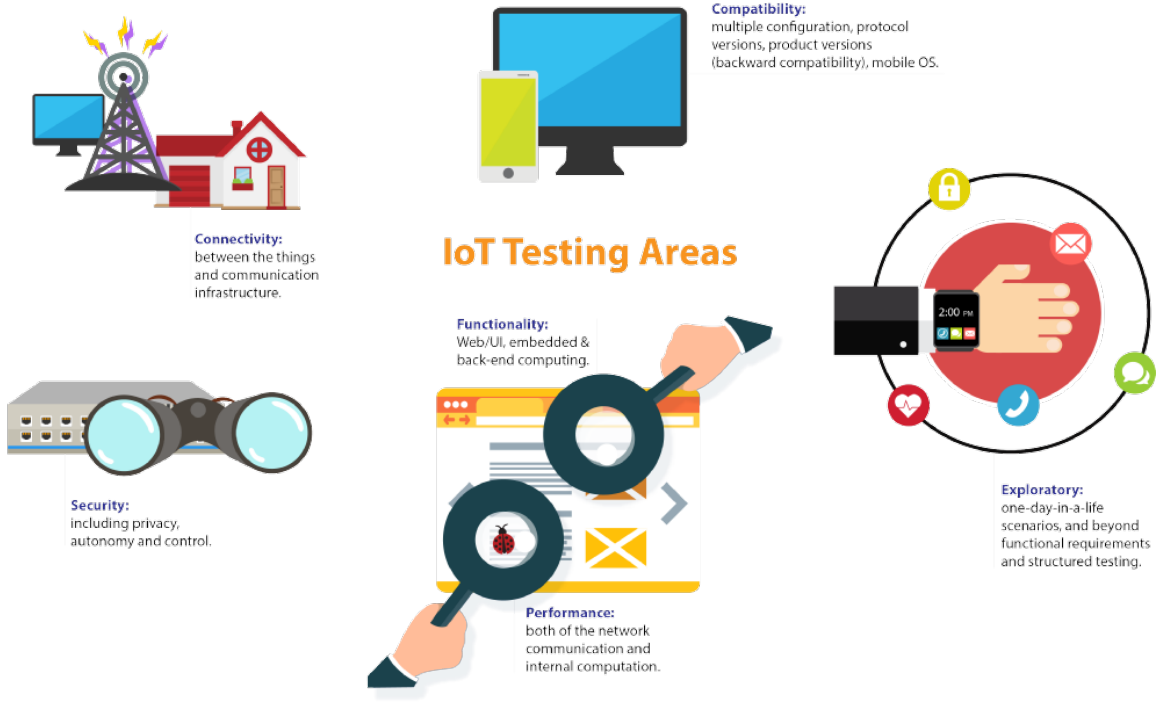


The V-model is a graphical representation of a systems development lifecycle, used to produce rigorous development lifecycle models and project management models.

The right side of the "V" represents integration of parts and their validation.

::.. Testing

Testing is another important part of development lifecycle that certifies the quality of the product that is delivered to the end customer. Testing IoT applications is very complex.



How to deal with the need of replicating a realistic deployment of IoT applications under test?

::.. Simulation/Emulation

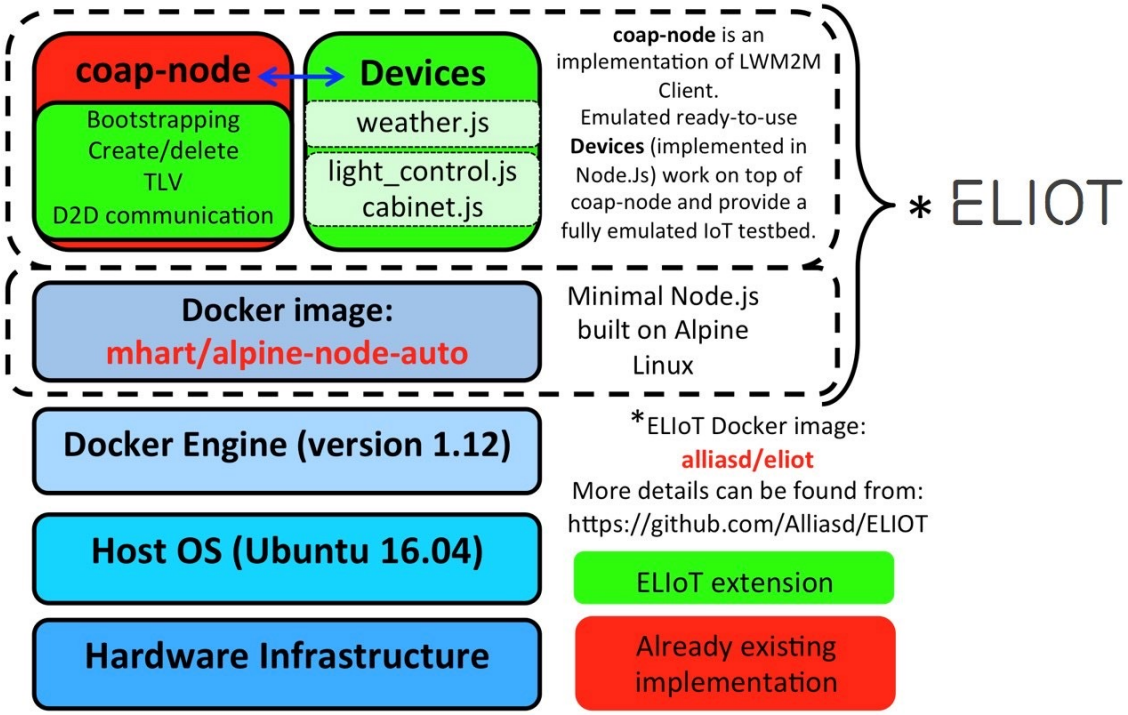
A viable solution is to exploit a simulated or emulated IoT test scenario. Both terms cover the act of mimicking a real thing in a virtual environment. The difference lies in the details:

- A simulator copies something from the real word into a virtual environment – often to give you an idea about how something works. It simulates the basic behavior but doesn't necessarily abide to all the rules of the real environment that it simulates.
- An emulator, on the other hand, duplicates the thing exactly as it exists in real life. The emulation is efficiently a complete imitation of the real thing – it just operates in a virtual environment instead of the real world.

In the IoT context, simulating the sensor network is the used approach. Many operating systems comes with an ad-hoc emulators of the supported hardware platforms.

::.. Emulated IoT Platform

Emulated IoT Platform (ELIoT) has its own Docker image to store device applications and the configuration files needed to run the emulated devices. Any number of containers can be launched on the platform, each with its own network interface and with a simple logic that simulates that of an IoT device.

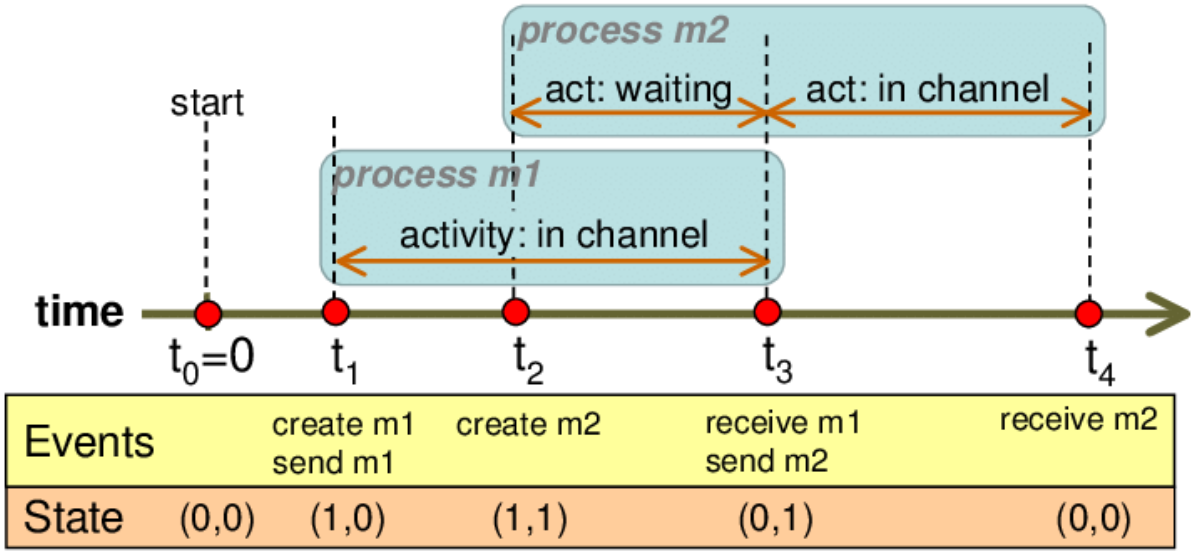


ELIoT can emulate lighting points and a control cabinet, as well as evaluate Lightweight M2M (LwM2M) function-ality in accordance to the Constrained Application Protocol (CoAP) and LwM2M specifications.

::.. Discrete Event Simulators (1/2)

A discrete-event simulation (DES) models the operation of a system as a (discrete) sequence of events in time.

- Each event occurs at a particular instant in time and marks a change of state in the system. Between consecutive events, no change in the system is assumed to occur; thus the simulation time can directly jump to the occurrence time of the next event, which is called next-event time progression.



::.. Discrete Event Simulators (2/2)

- There is also an alternative approach, called fixed-increment time progression, where time is broken up into small time slices and the system state is updated according to the set of events/activities happening in the time slice. Because not every time slice has to be simulated, a next-event time simulation can typically run much faster than a corresponding fixed-increment time simulation.

Both forms of DES contrast with continuous simulation in which the system state is changed continuously over time.

::.. OMNET++ (1/7)

OMNeT++ (Objective Modular Network Testbed in C++) is a modular, component-based C++ simulation library and framework, primarily for building network simulators.

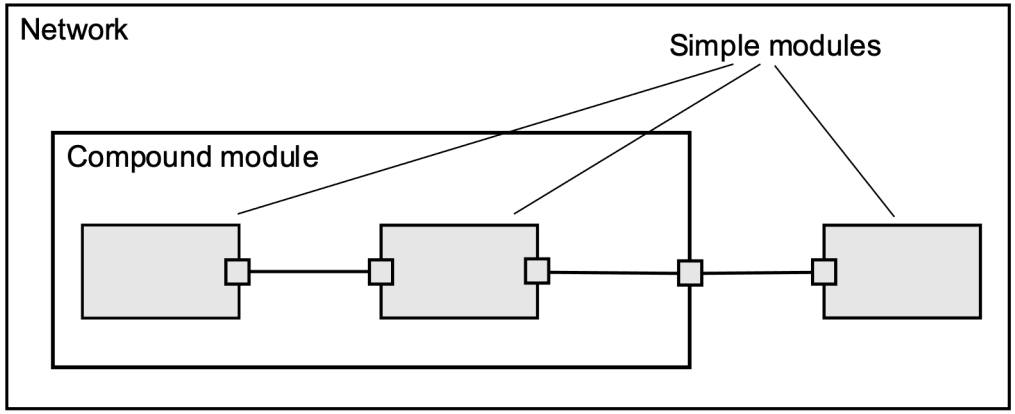
OMNeT++ itself is a simulation framework without models for network protocols like IP or HTTP. The main computer network simulation models are available in several external frameworks.

- INET offers a variety of models for all kind of network protocols and technologies like for IPv6, BGP etc. INET also offers a set of mobility models to simulate the node movement in simulations.

::.. OMNET++ (2/7)

The OMNeT++ model is a collection of hierarchically nested modules.

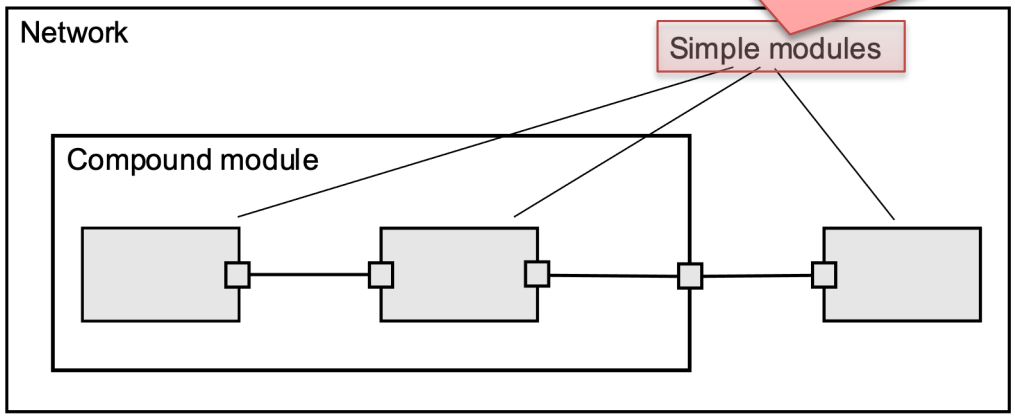
- The top-level module is also called the System Module or Network. This module contains one or more sub-modules each of which could contain other sub-modules.
- Modules are distinguished as being either simple or compound.



::.. OMNET++ (2/7)

The OMNeT++ model is a collection of hierarchically nested modules.

A simple module is associated with a C++ file that supplies the desired behaviors that encapsulate algorithms. Simple modules form the lowest level of the module hierarchy. Users implement simple modules in C++ using the OMNeT++ simulation class library.

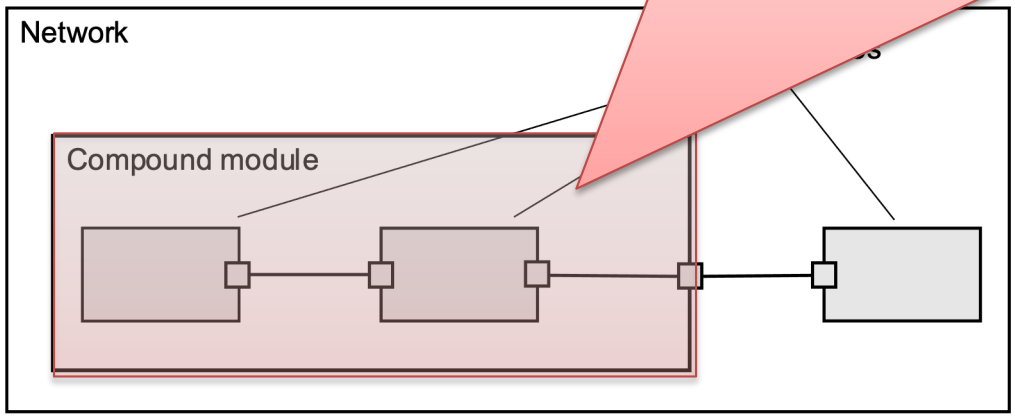


::.. OMNET++ (2/7)

The OMNeT++ model is a collection of hierarchically nested modules.

- The top-level module is also called the System Module or Network. This module contains one or more sub-modules

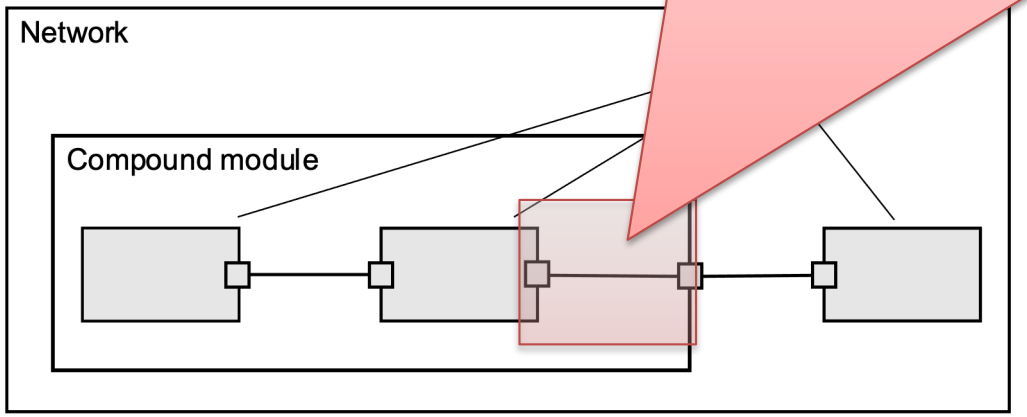
Compound modules are aggregates of simple modules and are not directly associated with a C++ file that supplies behaviors.



::.. OMNET++ (2/7)

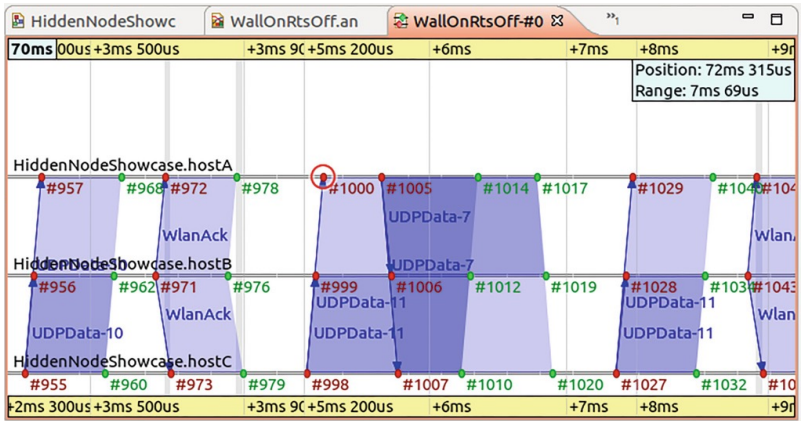
The OMNeT++ model is a collection of hierarchically nested modules.

Modules communicate by exchanging messages. Each message may be a complex data structure. Messages may be exchanged directly between simple modules or via a series of gates and connections. Messages represent frames or packets in a computer network.



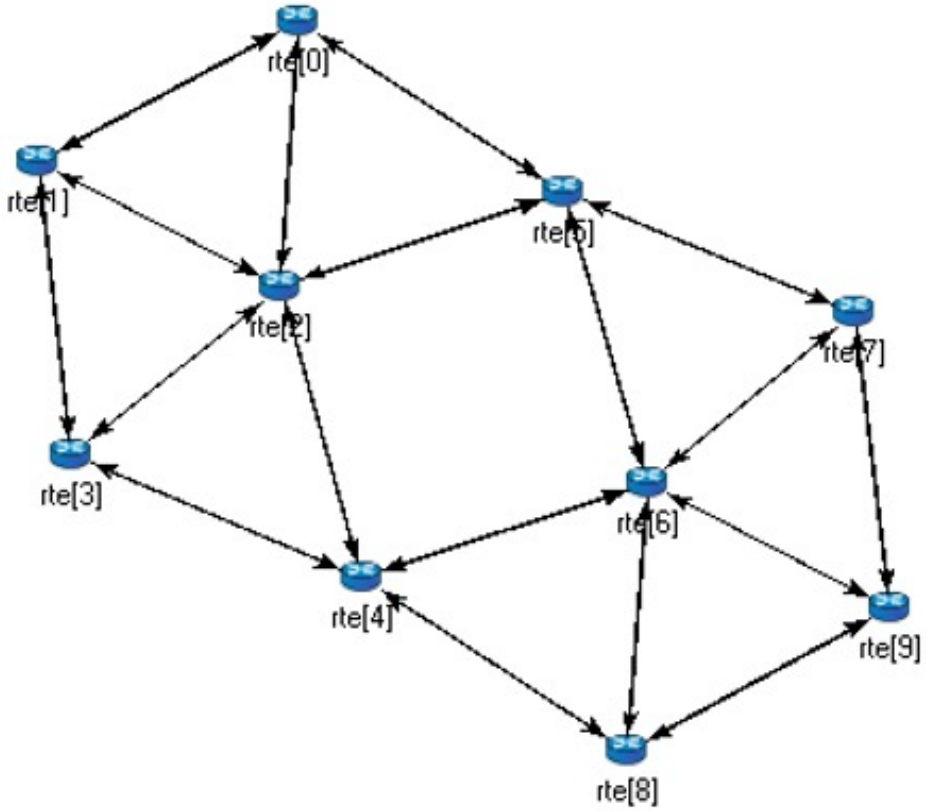
::.. OMNET++ (3/7)

The local simulation time advances when a module receives messages from another module or from itself. Self-messages are used by a module to schedule events at a later time.



The structure and interface of the modules are specified using a network description language or NED, which lets the user declare simple modules, and connect and assemble them into compound modules. Users exploit C++ to implement the underlying behaviors of simple modules.

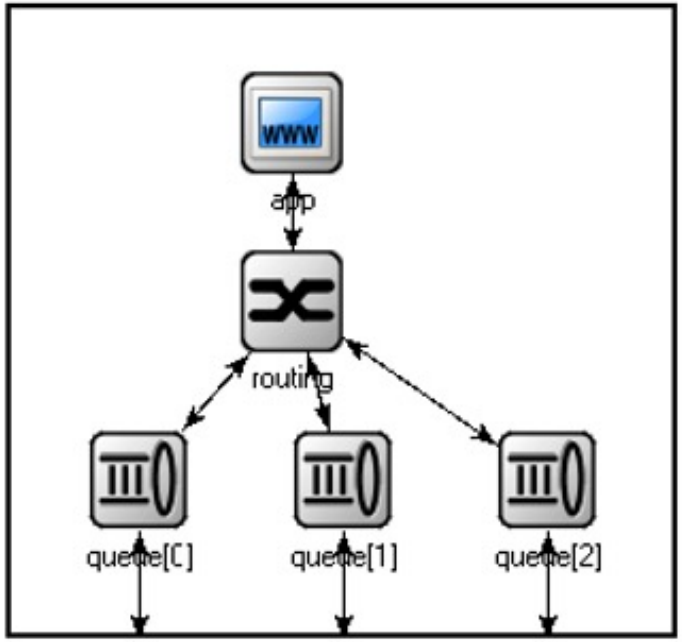
::.. OMNET++ (4/7)




```
//
// A network
//
network Network
{
    submodules:
        node1: Node;
        node2: Node;
        node3: Node;
        ...
    connections:
        node1.port++ <--> {datarate=100Mbps;} <--> node2.port++;
        node2.port++ <--> {datarate=100Mbps;} <--> node4.port++;
        node4.port++ <--> {datarate=100Mbps;} <--> node6.port++;
        ...
}
```

::.. OMNET++ (5/7)

```
module Node
{
    parameters:
        int address;
        @display("i=misc/node_vs,gold");
    gates:
        inout port[];
    submodules:
        app: App;
        routing: Routing;
        queue[sizeof(port)]: Queue;
    connections:
        routing.localOut --> app.in;
        routing.localIn <-- app.out;
        for i=0..sizeof(port)-1 {
            routing.out[i] --> queue[i].in;
            routing.in[i] <-- queue[i].out;
            queue[i].line <--> port[i];
        }
}
```



::.. OMNET++ (6/7)

Simple modules encapsulate C++ code that generates events and reacts to events, implementing the behaviour of the module. To define the dynamic behavior of a simple module, one of the following member functions need to be overridden:

- handleMessage(cMessage *msg) is invoked with the message as parameter at the reception of messages. handleMessage() is expected to process the message, and then return. Simulation time never elapses inside handleMessage() calls, only between them.
- activity() is started as a coroutine at the beginning of the simulation, and it runs until the end of simulation (or until the function returns or otherwise terminates). Messages are obtained with receive() calls. Simulation time elapses inside receive() calls.

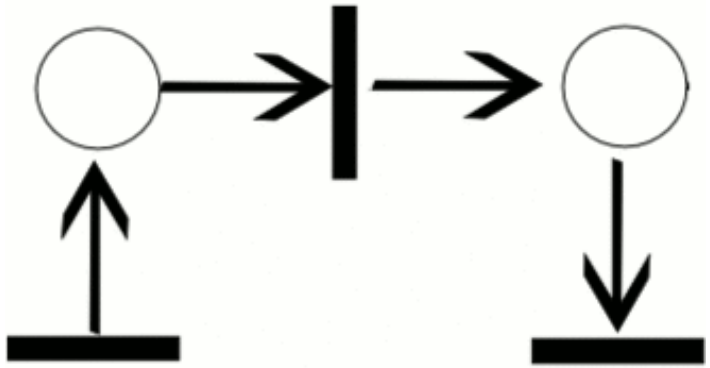
::.. OMNET++ (7/7)

```
// file: HelloModule.cc
#include <omnetpp.h>
using namespace omnetpp;

class HelloModule : public cSimpleModule
{
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

// register module class with OMNeT++
Define_Module(HelloModule);

void HelloModule::initialize()
{
    EV << "Hello World!\n";
}

void HelloModule::handleMessage(cMessage *msg)
{
    delete msg; // just discard everything we receive
}
```

::.. Petri Nets (1/7)

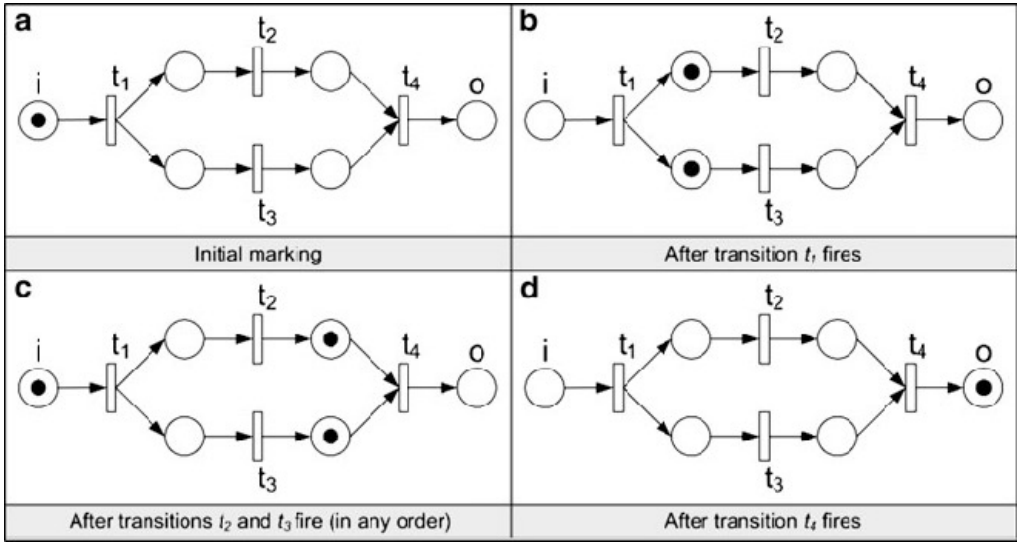
A Petri net is one of several mathematical modeling languages for the description of distributed systems, and a class of discrete event dynamic system.

A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e., events that may occur, represented by bars) and places (i.e., conditions, represented by circles). The directed arcs describe which places are pre- and/or post-conditions for which transitions (signified by arrows).



::.. Petri Nets (2/7)

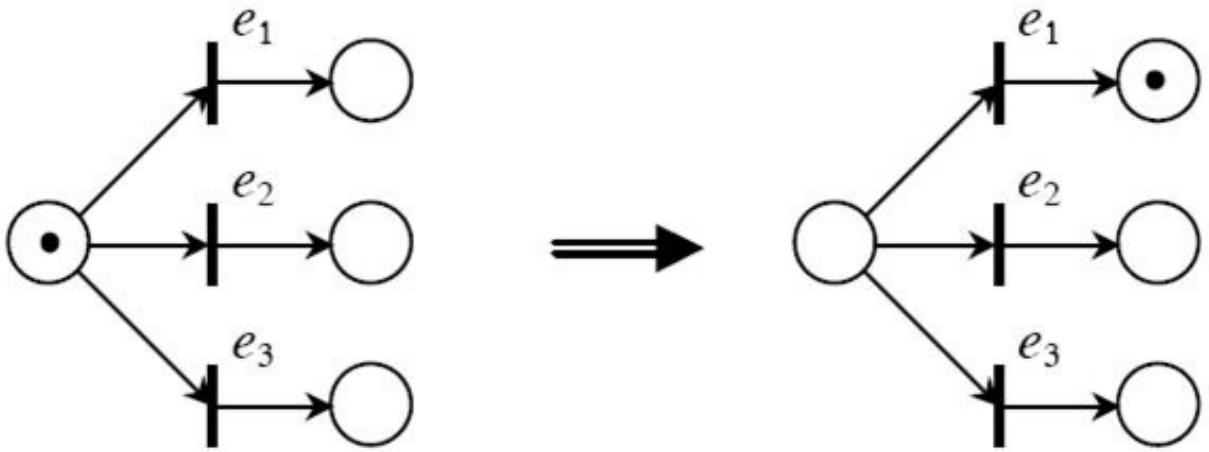
Places in a Petri net may contain a discrete number of marks called *tokens*. Any distribution of tokens over the places will represent a configuration of the net called a *marking*.



A transition of a Petri net may *fire* if it is *enabled*, i.e. there are sufficient tokens in all of its input places. When the transition fires, it consumes the required input tokens, and creates tokens in the output places – both operations are not interruptible.

::.. Petri Nets (3/7)

Unless an execution policy is defined, the execution of Petri nets is nondeterministic: when multiple transitions are enabled at the same time, they will fire in any order.



Since firing is nondeterministic, and multiple tokens may be present anywhere in the net (even in the same place), Petri nets are well suited for modeling the concurrent behavior of distributed systems.

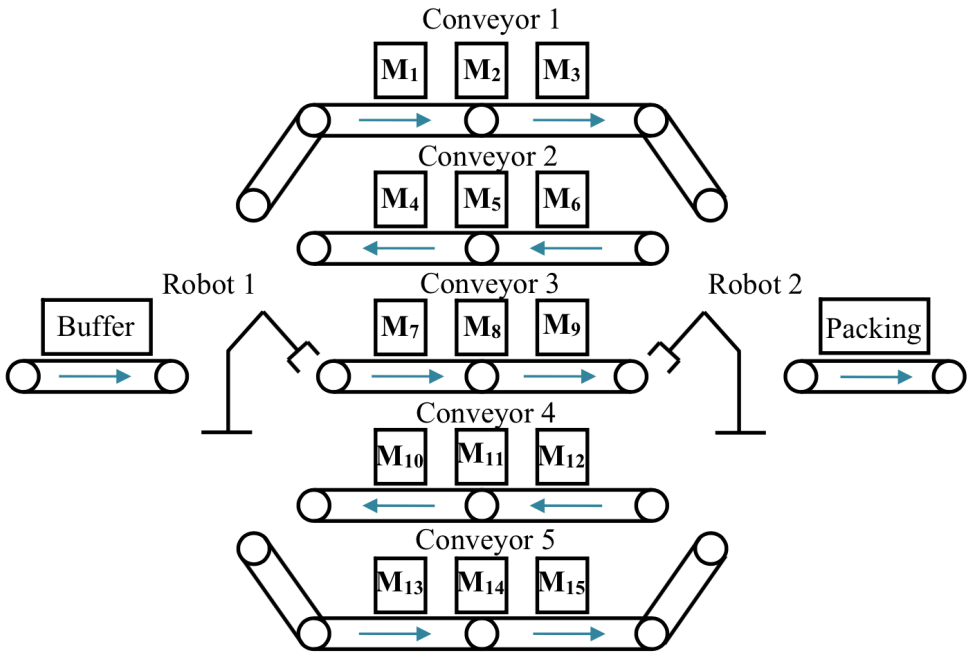
::.. Petri Nets (4/7)

There exists countless variants of Petri nets. Some of them are completely backwards-compatible with the original Petri net, some add properties that cannot be modelled in the original Petri net formalism:

- Colored Petri nets, where tokens are “colored” to represent different kinds of resources;
- Augmented Petri nets, where transitions additionally depend on external conditions;
- Timed Petri nets, where a duration is associated with each transitions;
- Stochastic Petri nets, where the transitions fire after a probabilistic delay determined by a random variable.

::.. Petri Nets (5/7)

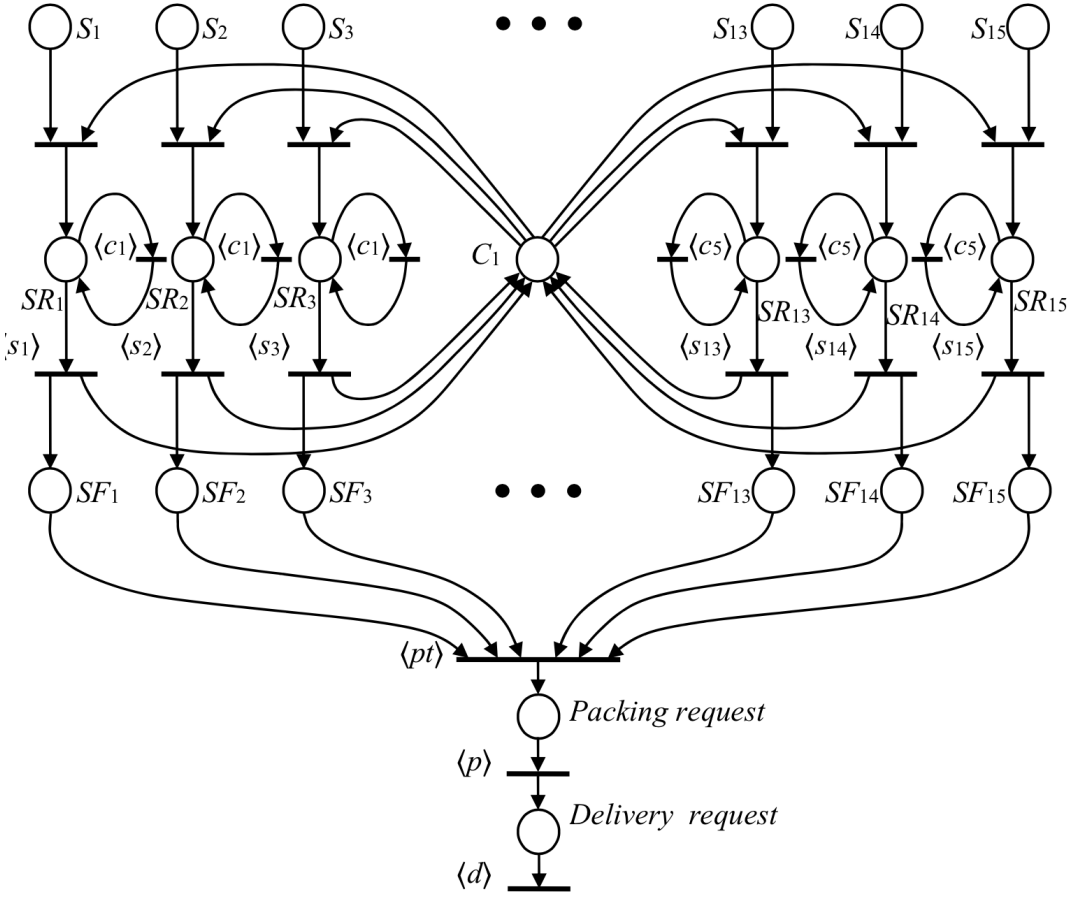
A manufacturing system has 15 machines. In order to convey the products, 2 robots and 6 conveyor belts are considered. Five of the conveyor belts transport semi-finished products to the production machines. Each conveyor belt moves products for reaching three of the production machines.



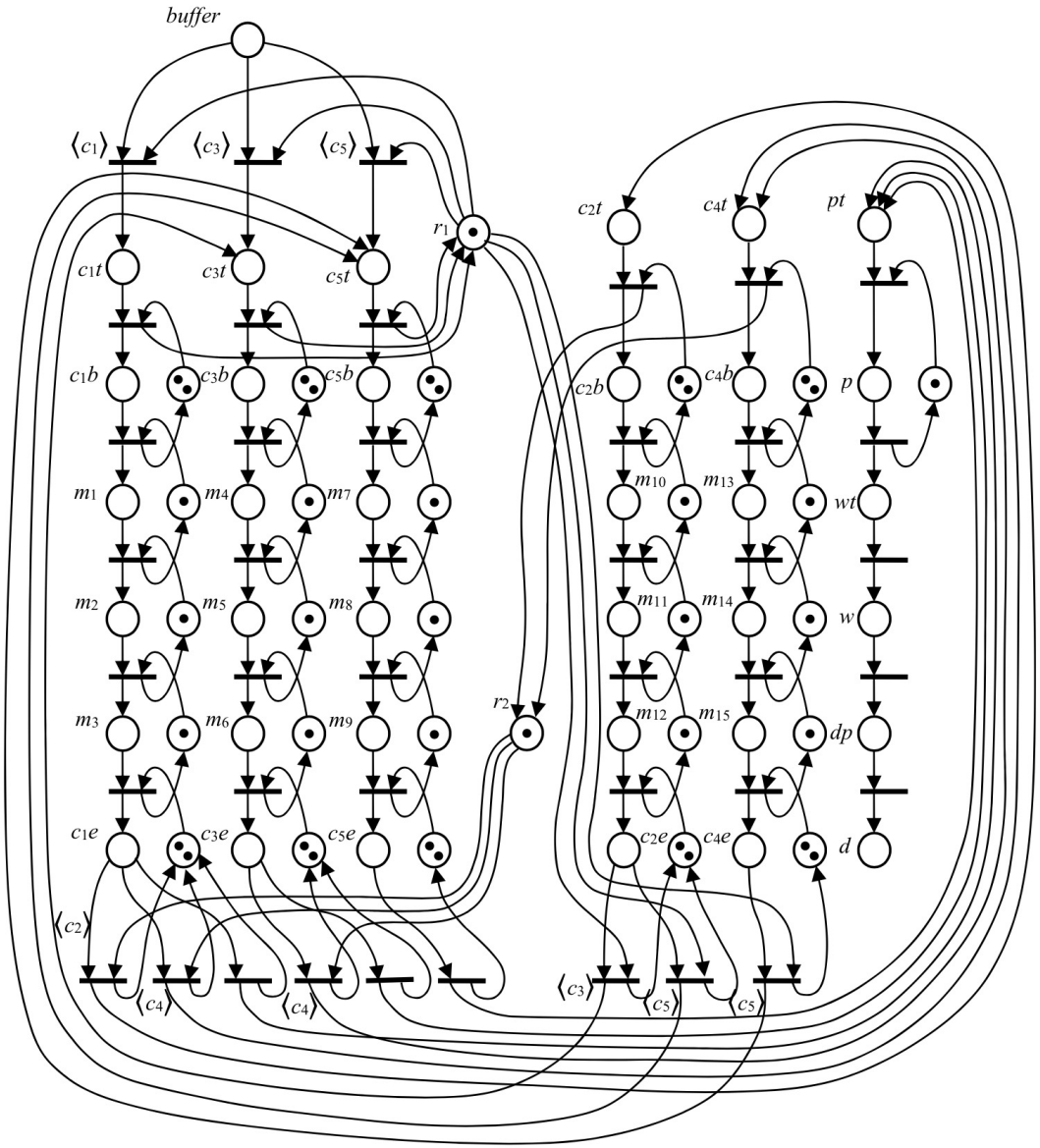
Robot 1 loads conveyor belts 1, 3, and 5 and unloads conveyor belts 2 and 4, and also the buffer of bases or containers for individual products. Moreover, robot 2 loads conveyors 2 and 4 and final conveyor to the packaging station and unloads conveyors 1, 3 and 5.

::.. Petri Nets (6/7)

It is a PN model of a single product, which can request different services along its manufacturing process. Every active product is modeled by a different token net.



:.. Petri Nets (7/7)



System net is the PN model of the manufacturing facility. Black tokens in some places represent products. The communication between system net and tokens allows requesting services.

::.. Queueing Theory (1/3)

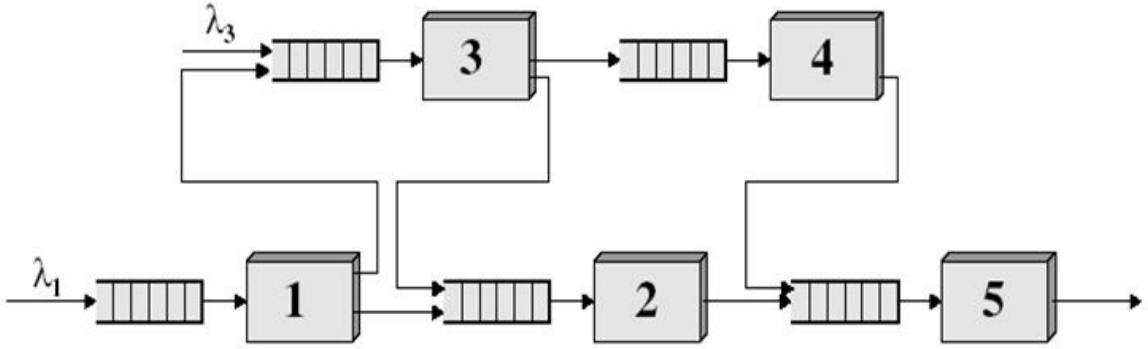
Queueing Theory is a mathematical formulation describing queues by predicting their lengths and waiting time.

- A queue is considered as a black box receiving jobs or "customers", which possibly wait some time, take some time being processed, and then depart from the queue.

- The internals is made of a buffer hosting the jobs waiting to be served, and one or more "servers" which can each be paired with an arriving job until it departs, after which that server will be free to be paired with another arriving job.

::.. Queueing Theory (2/3)

Queues can be properly interconnected as a network so as to represent and analyze communication and computer, such as IoT interaction where middleware protocol are represented as queues and the exchanged messages as jobs served.

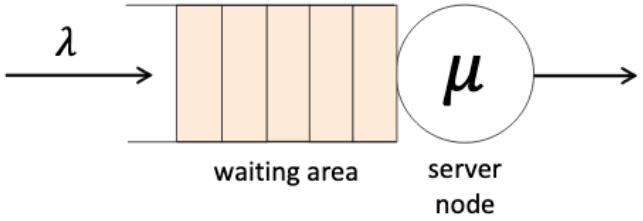


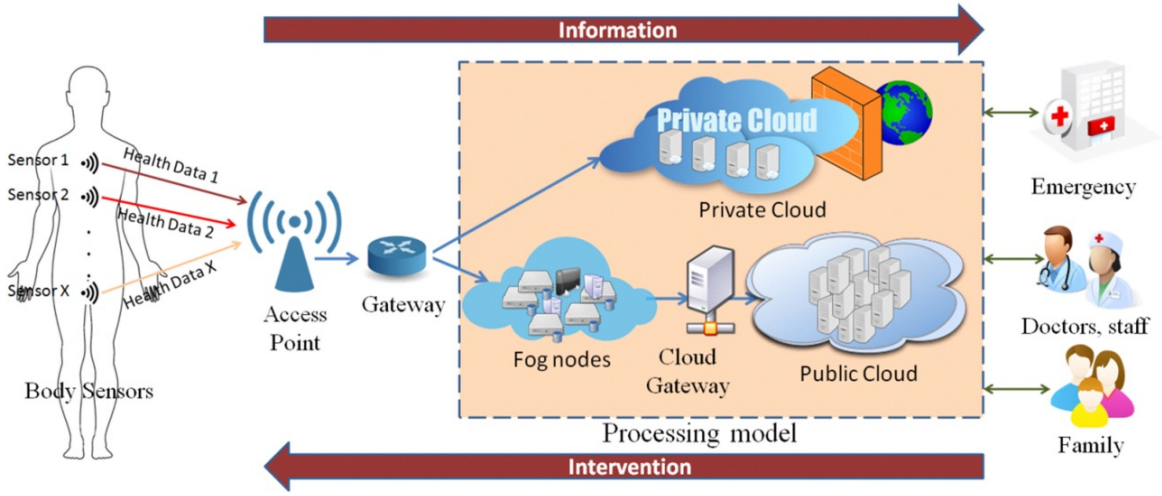
The standard system used to describe and classify queues is Kendall's notation (or sometimes Kendall notation) using three factors written A/S/c:

- A denotes the time between arrivals to the queue,
- S the service time distribution,
- c the number of service channels open at the node.

::.. Queueing Theory (3/3)

A widely known example of queues is the M/M/1, featuring Poisson arrivals and exponential service times by a single server. Λ is the input rate of messages to the queue and μ is the service rate for the processing of messages.

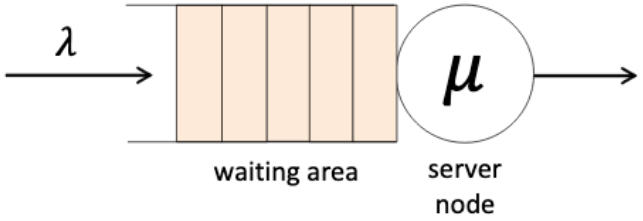


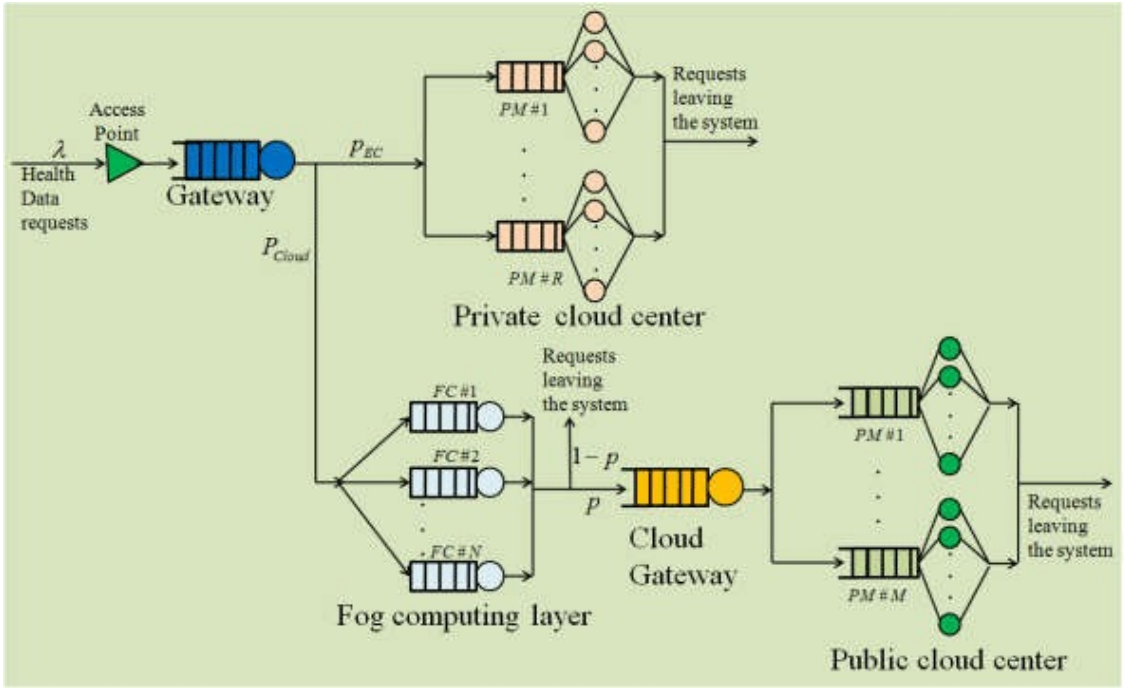


The IoT-based health monitoring system architecture is made of the WBAN on the patients, the gateway, the cloud infrastructure and a web service for end users.

::.. Queueing Theory (3/3)

A widely known example of queues is the M/M/1, featuring Poisson arrivals and exponential service times by a single server. Λ is the input rate of messages to the queue and μ is the service rate for the processing of messages.



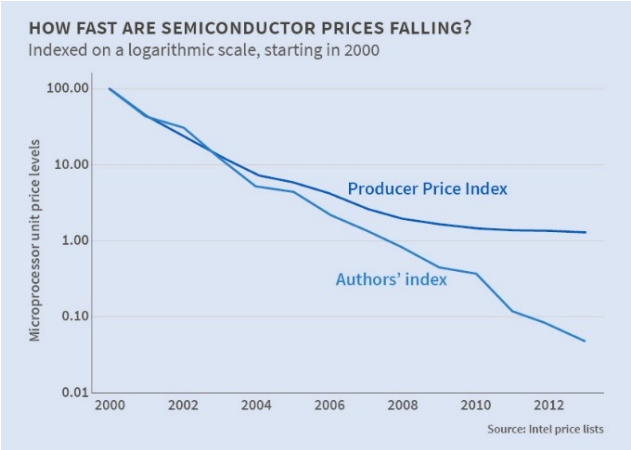


The queueing theory can be used to model the system and study its performance.

Sensor Anatomy and Taxonomy

::.. Sensors (1/3)

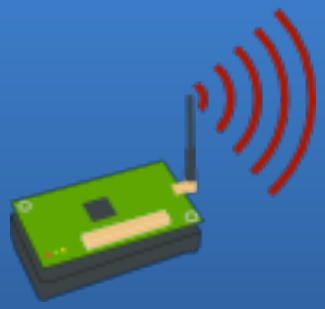
The basic component of a distributed sensor network is the sensor node, which is a small device, with a constrained battery, capable of monitoring certain characteristics of the environment in which it is located, and of processing, sending or storing the monitoring information.



Given the advances in semiconductor technology, the costs of these sensors are gradually decreasing over time.

We can find different characteristics and definitions of what a sensor node is, but probably the most exhaustive is the one proposed by the IEEE 1451.2 standard.

::.. Sensors (2/3)



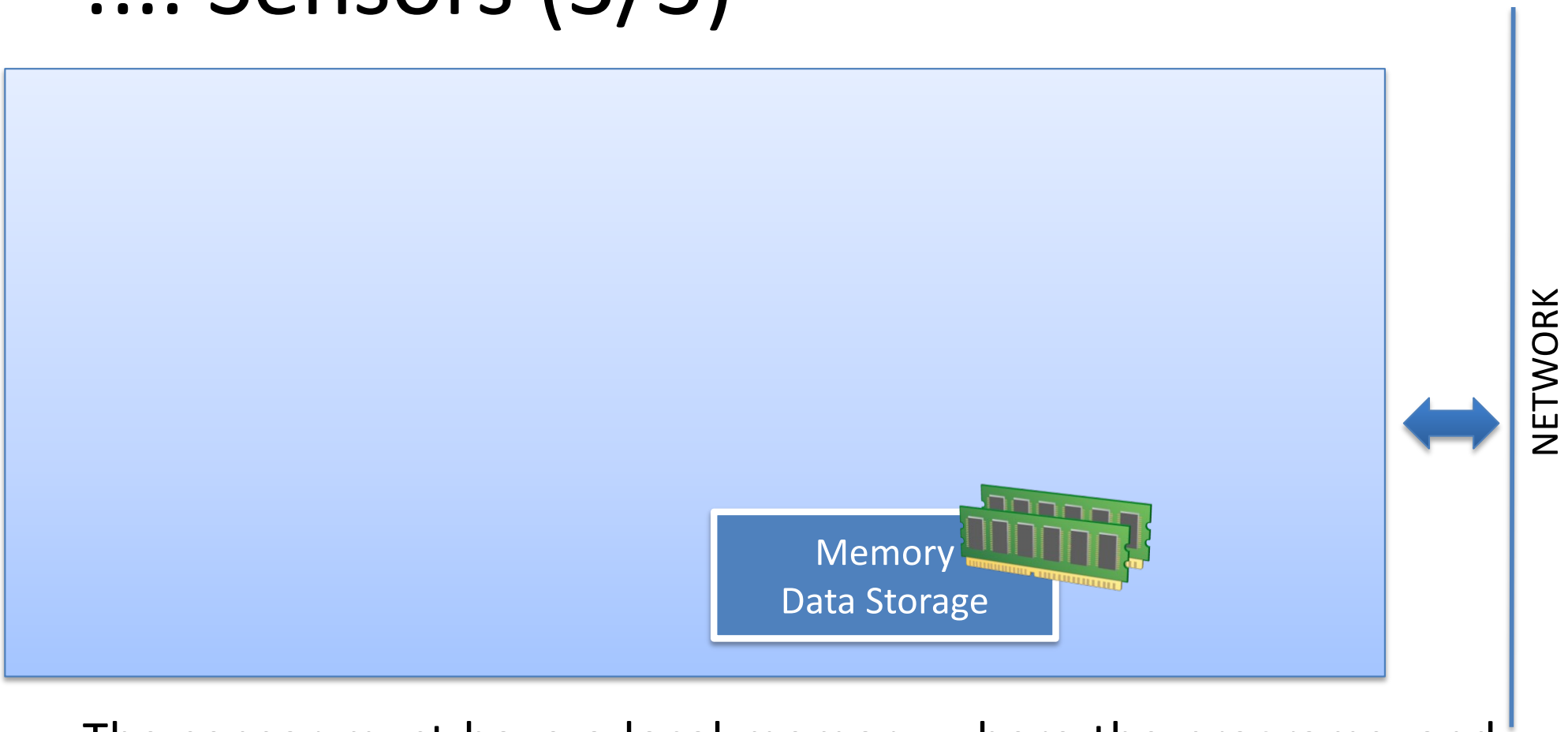
«A transducer that integrates the functions necessary for the correct representation of the measured or controlled values. These features typically simplify the integration of the transducer into applications that use network structures.»

::.. Sensors (2/3)

«A transducer that integrates the functions necessary for the correct representation of the measured or controlled values. These features typically simplify the integration of the transducer into applications that use network structures.»

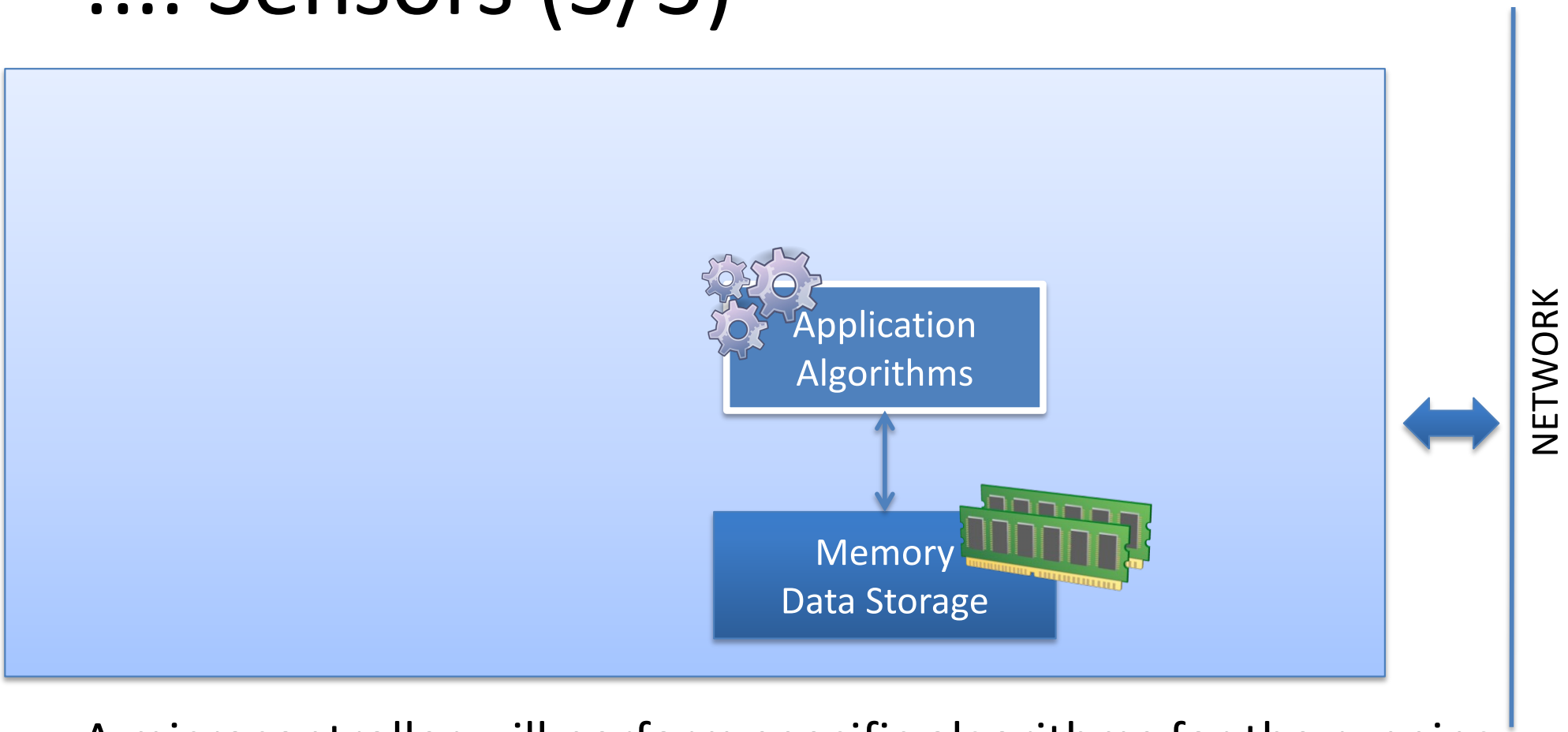
The standard outlines a sensor as an intelligent device that is simply only not able to correctly respond to requests for information or to communicate in digital format, but also must give an added value to the information itself by integrating various additional features.

::.. Sensors (3/3)



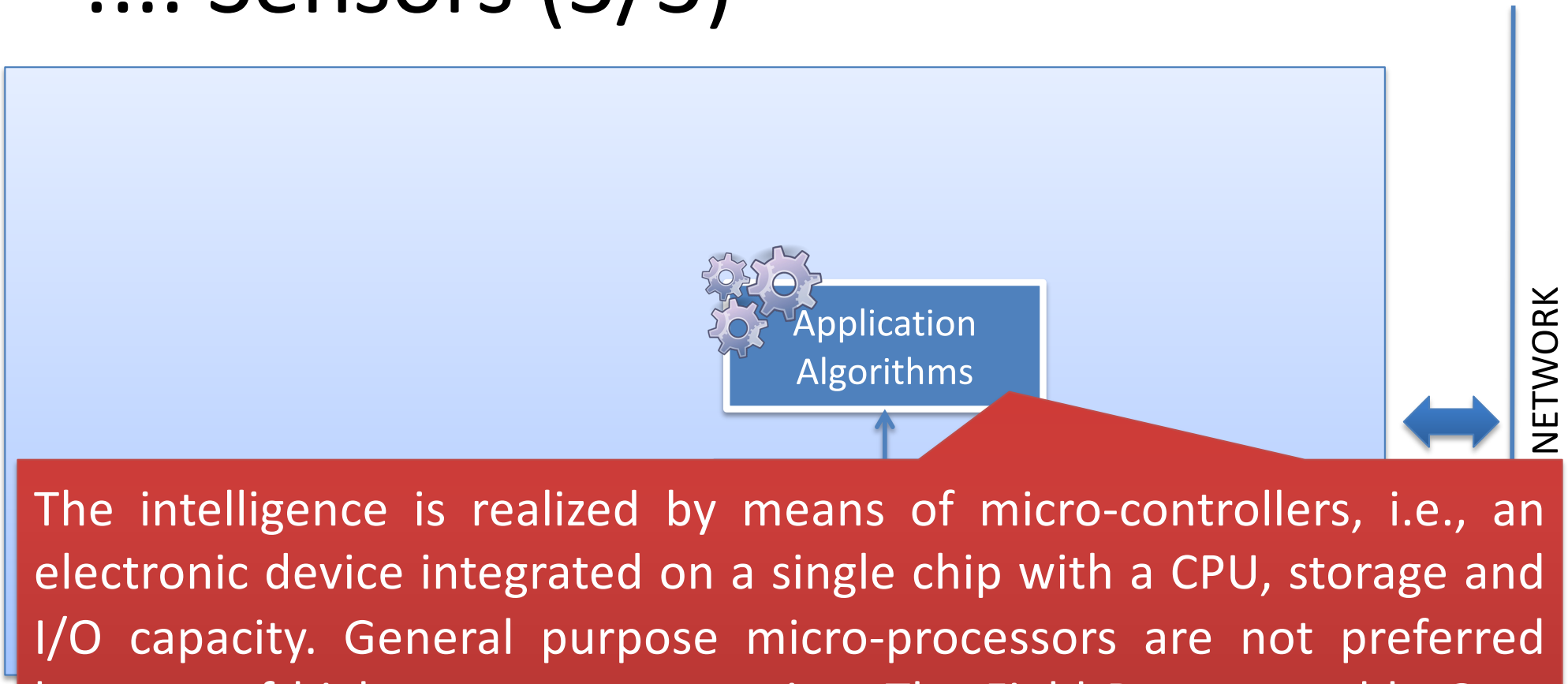
The sensor must have a local memory where the programs, and the operating and monitoring data are permanently placed. This memory has a limited capacity, and its use must be optimized. Flash memories can be added to the existing one to increase the storage capacity.

::.. Sensors (3/3)



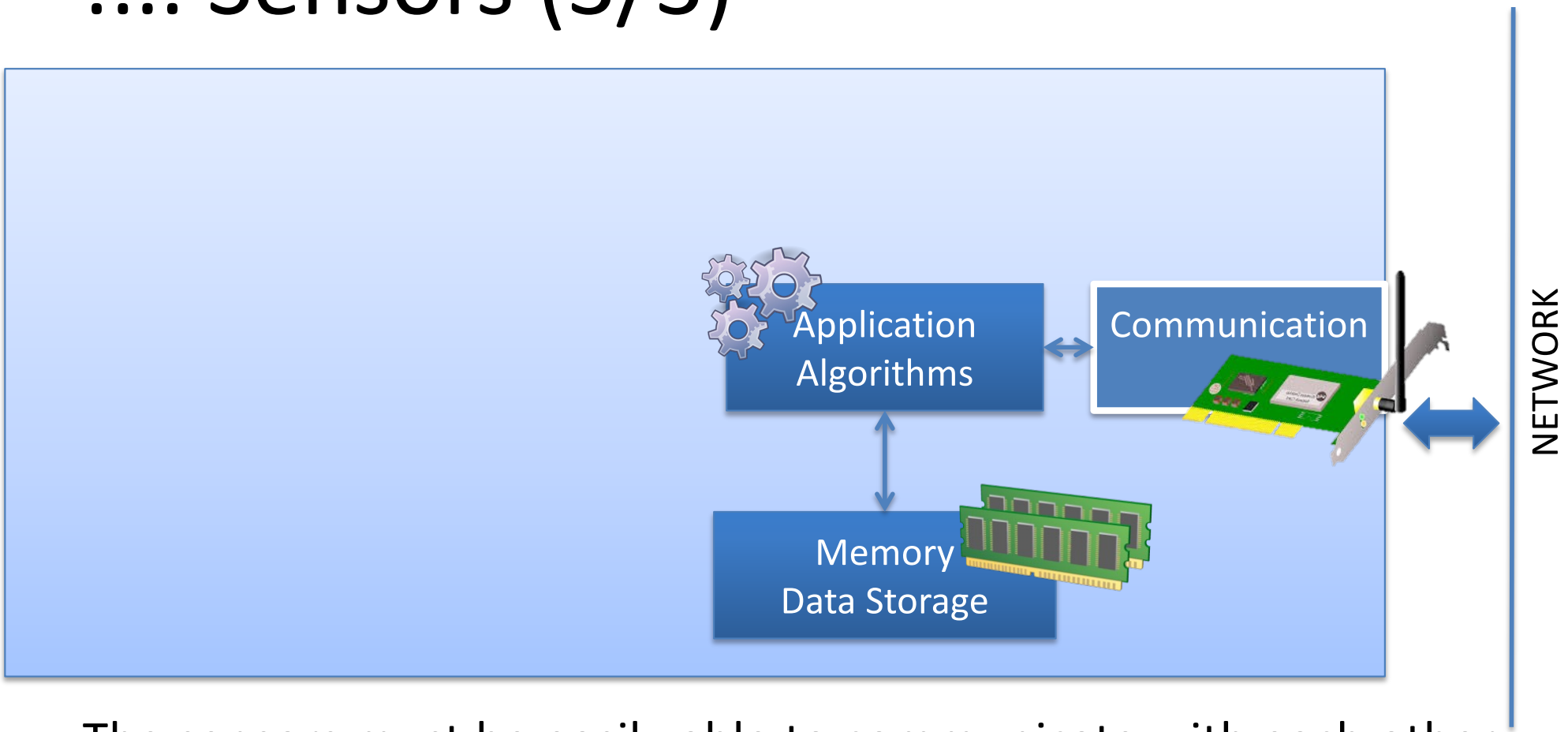
A microcontroller will perform specific algorithms for the running applications, but also routines for self-diagnosis activities, to determine the operating status of the device, to characterize measurements over time and space.

::.. Sensors (3/3)



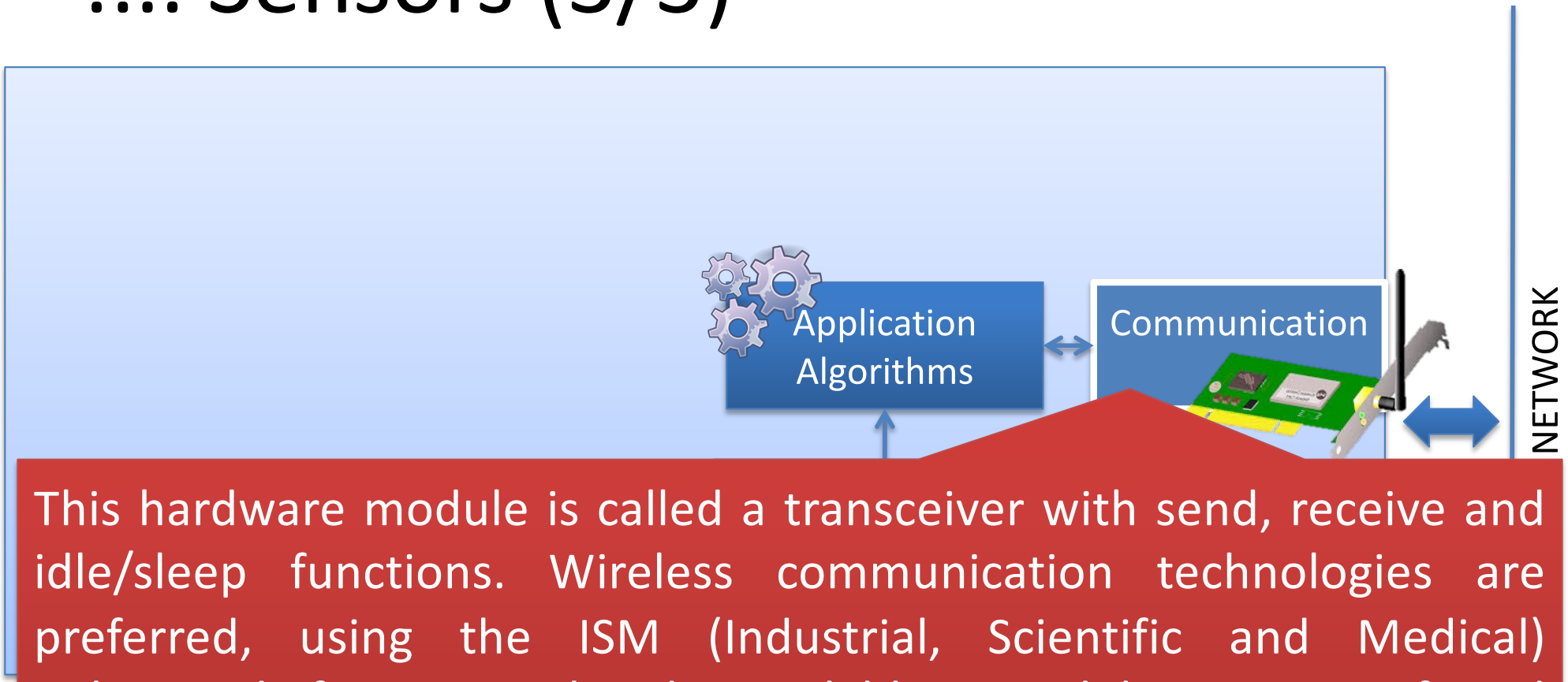
The intelligence is realized by means of micro-controllers, i.e., an electronic device integrated on a single chip with a CPU, storage and I/O capacity. General purpose micro-processors are not preferred because of high energy consumption. The Field Programmable Gate Arrays (FPGAs), i.e., a set of pre-fabricated cells and interconnections reconfigured via software, are used in the prototype phase. Application Specific Integrated Circuits (ASICs) are used when dealing with operations related to large numbers a wide volumes of data.

::.. Sensors (3/3)



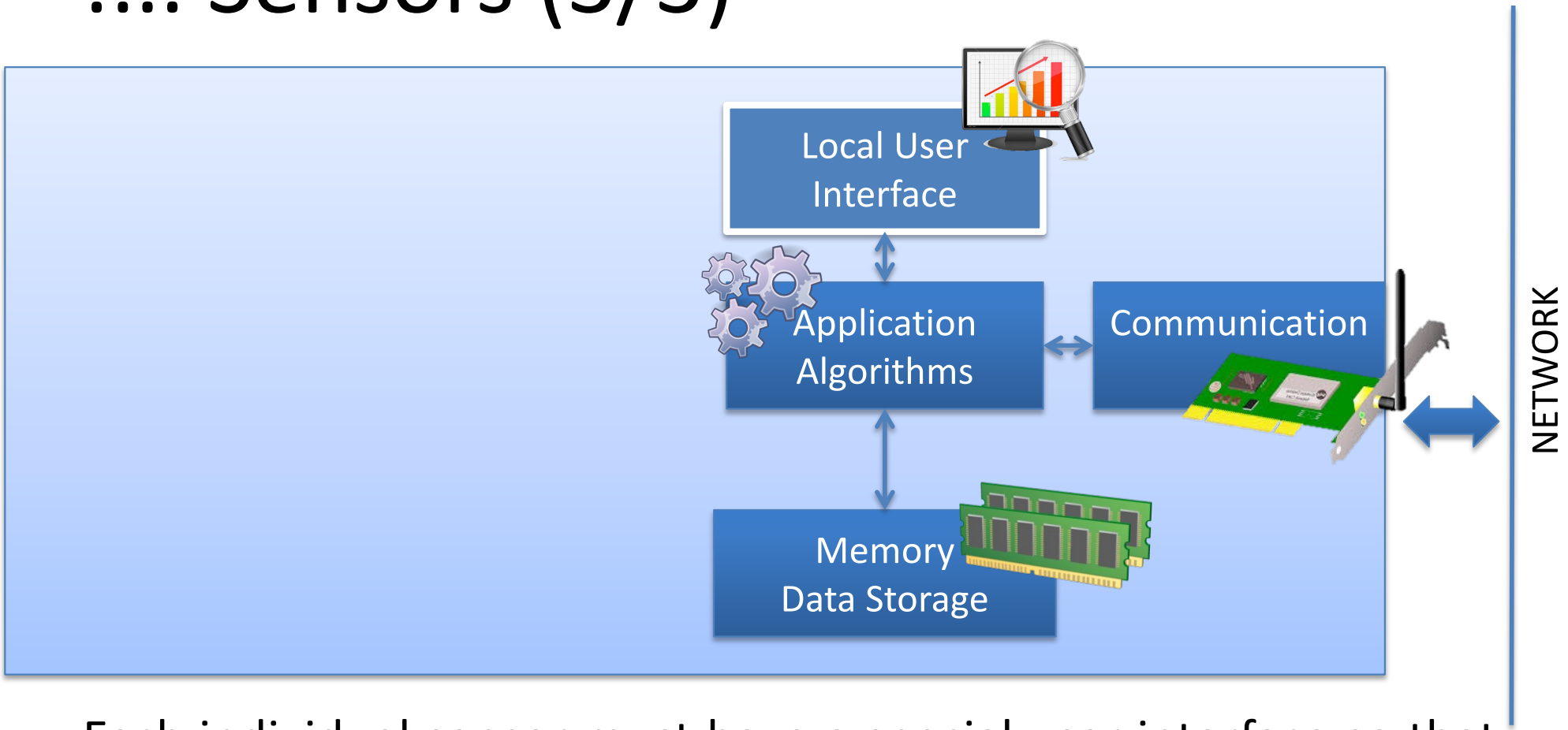
The sensors must be easily able to communicate with each other and with the base stations via a plug and play mode. For this purpose, one or more wireless communication technologies can be adopted and supported by the device.

::.. Sensors (3/3)



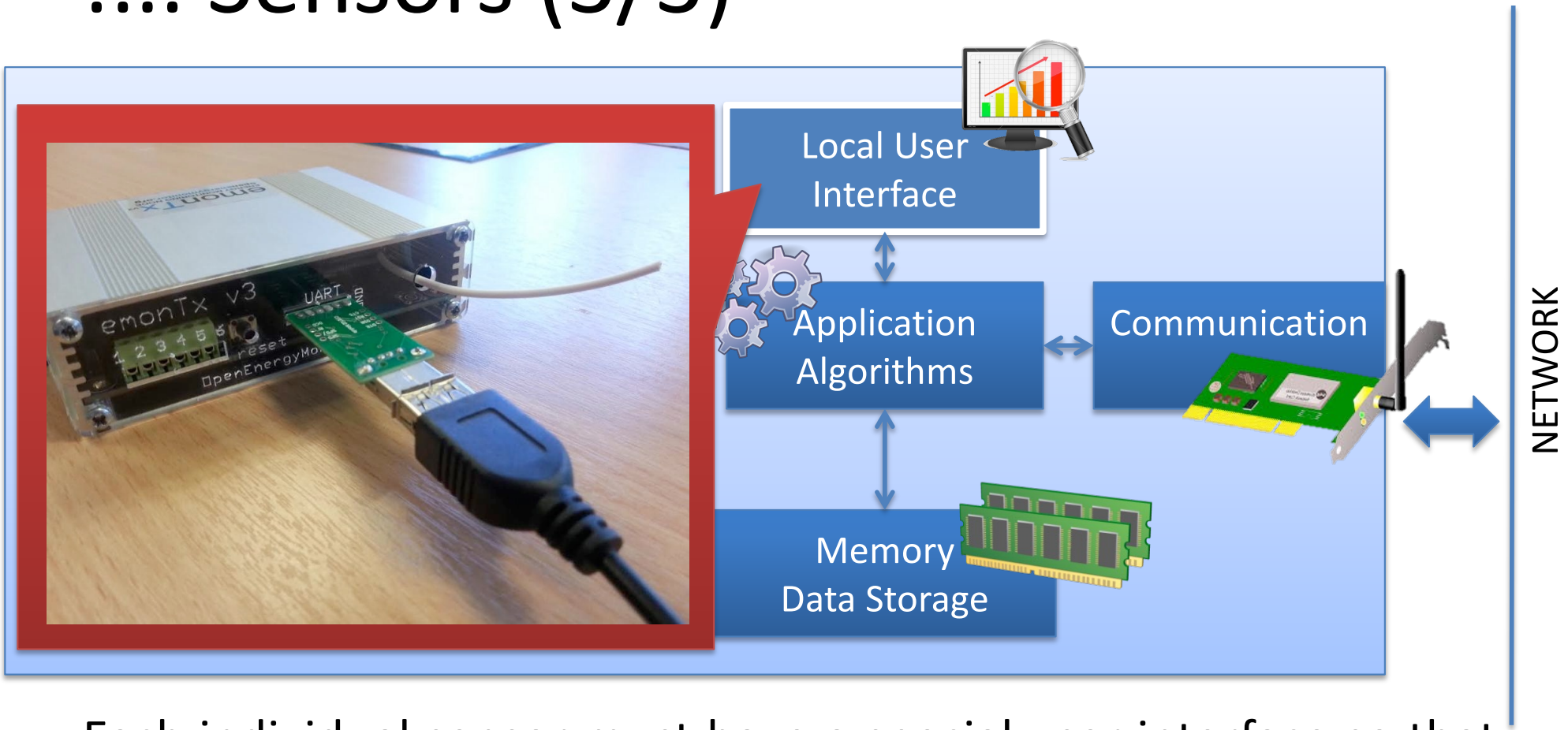
This hardware module is called a transceiver with send, receive and idle/sleep functions. Wireless communication technologies are preferred, using the ISM (Industrial, Scientific and Medical) unlicensed frequency band. Available possibilities are infrared solutions (inexpensive but with line-of-sight problems and interference), radio-frequencies with frequencies between 433 MHz and 2.4 GHz, or with optical wireless (with lower costs and higher data rates, but with the same problems of infrared communications).

::.. Sensors (3/3)



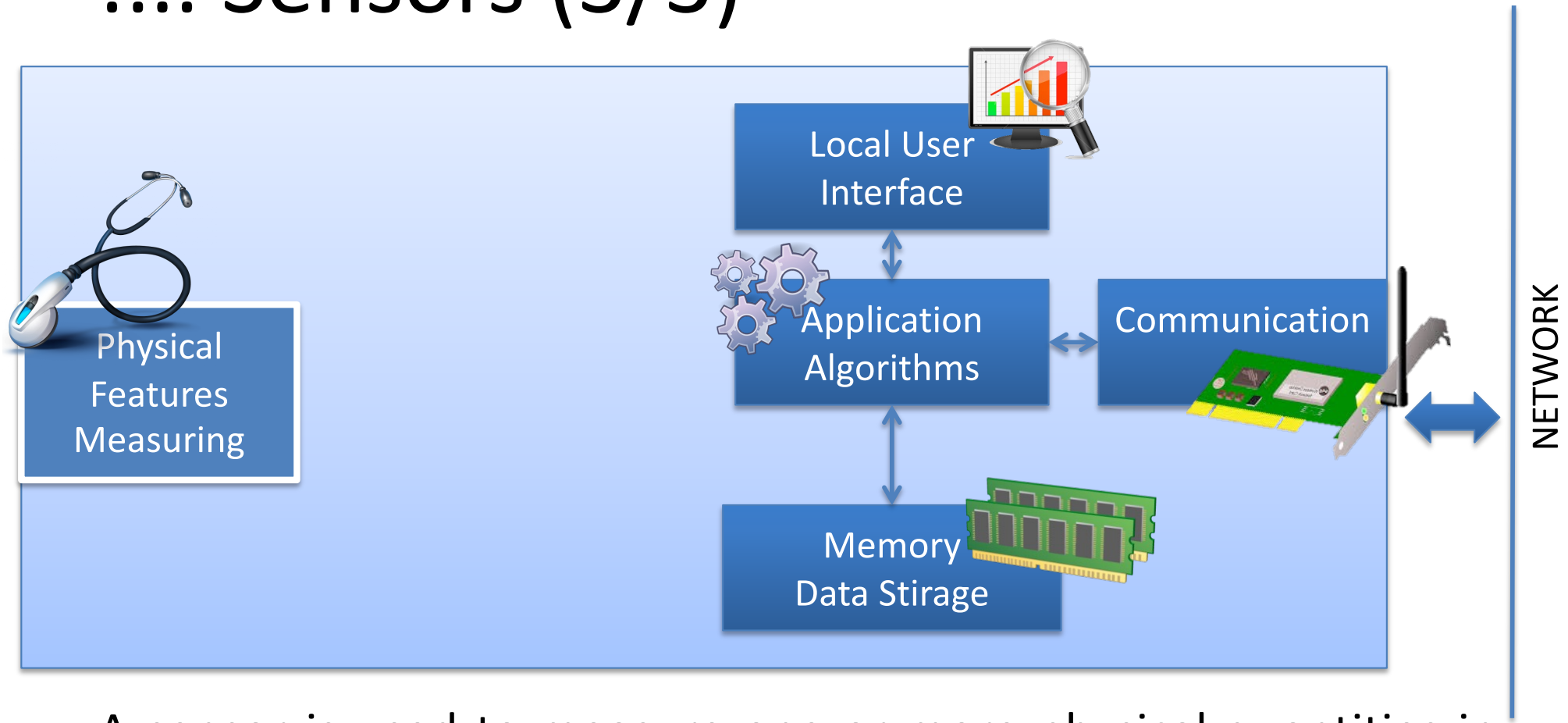
Each individual sensor must have a special user interface so that users can locally access operating and monitoring information. This interface can be the use of specialized or configurable pins by the programmer or a point-to-point connection (eg USB).

::.. Sensors (3/3)



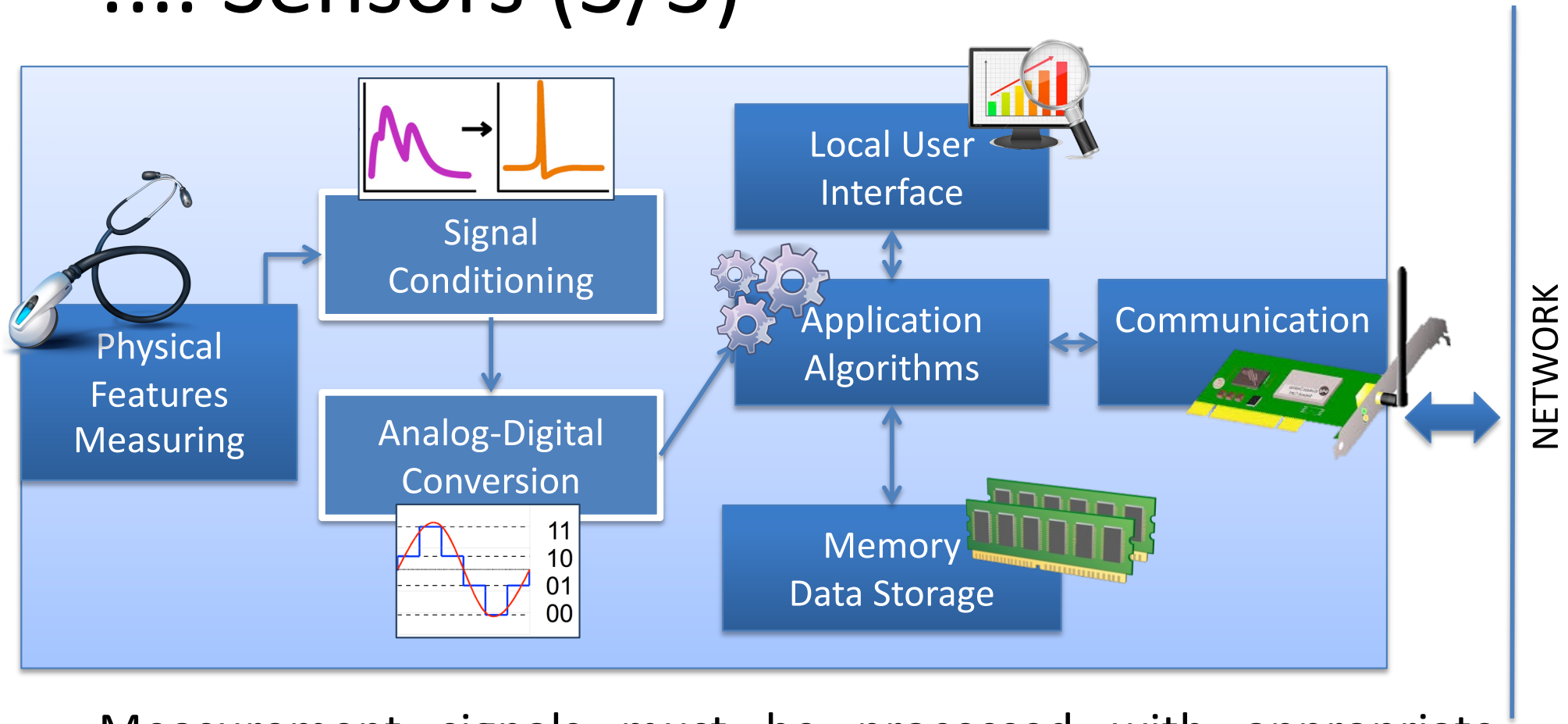
Each individual sensor must have a special user interface so that users can locally access operating and monitoring information. This interface can be the use of specialized or configurable pins by the programmer or a point-to-point connection (eg USB).

::.. Sensors (3/3)



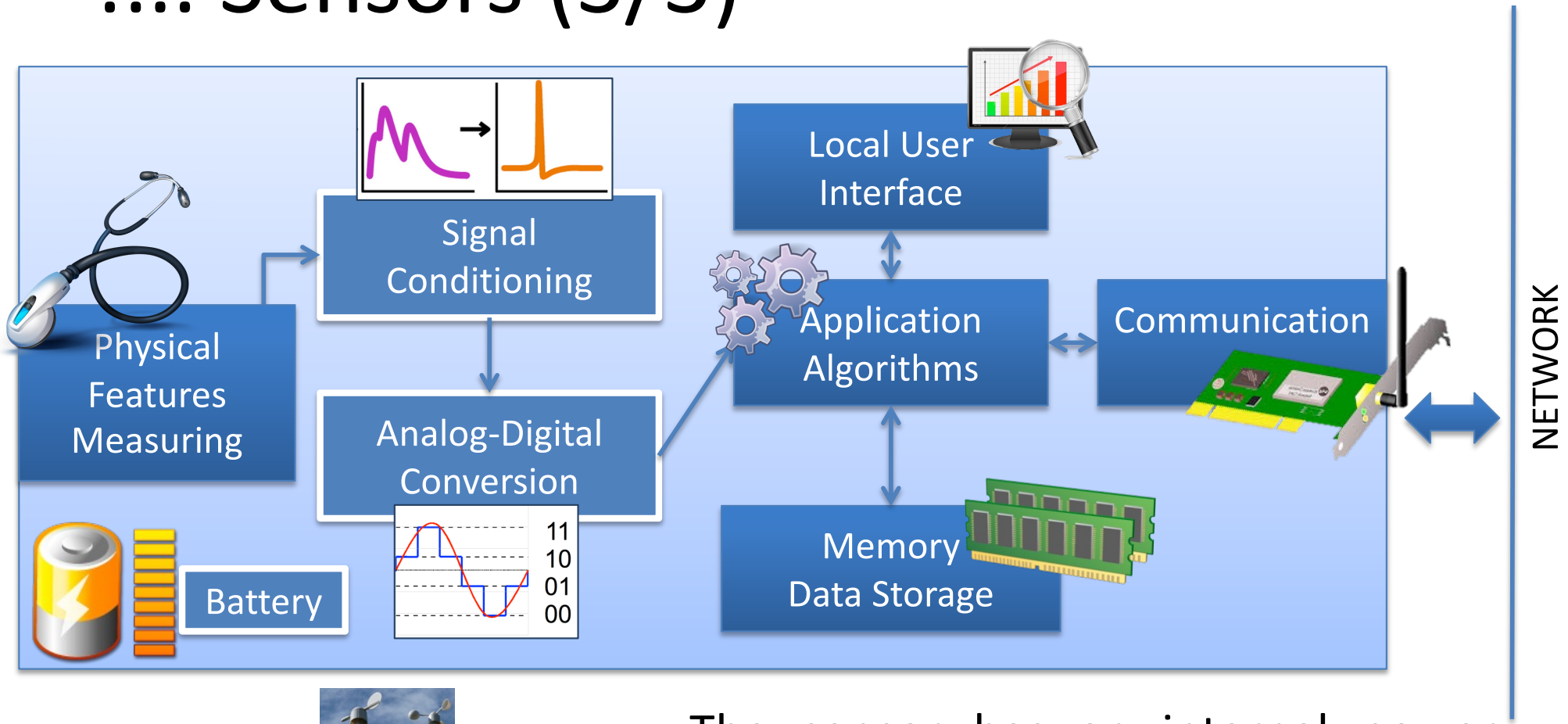
A sensor is used to measure one or more physical quantities in the environment in which it is placed, therefore it must be equipped with a special measuring hardware.

::.. Sensors (3/3)



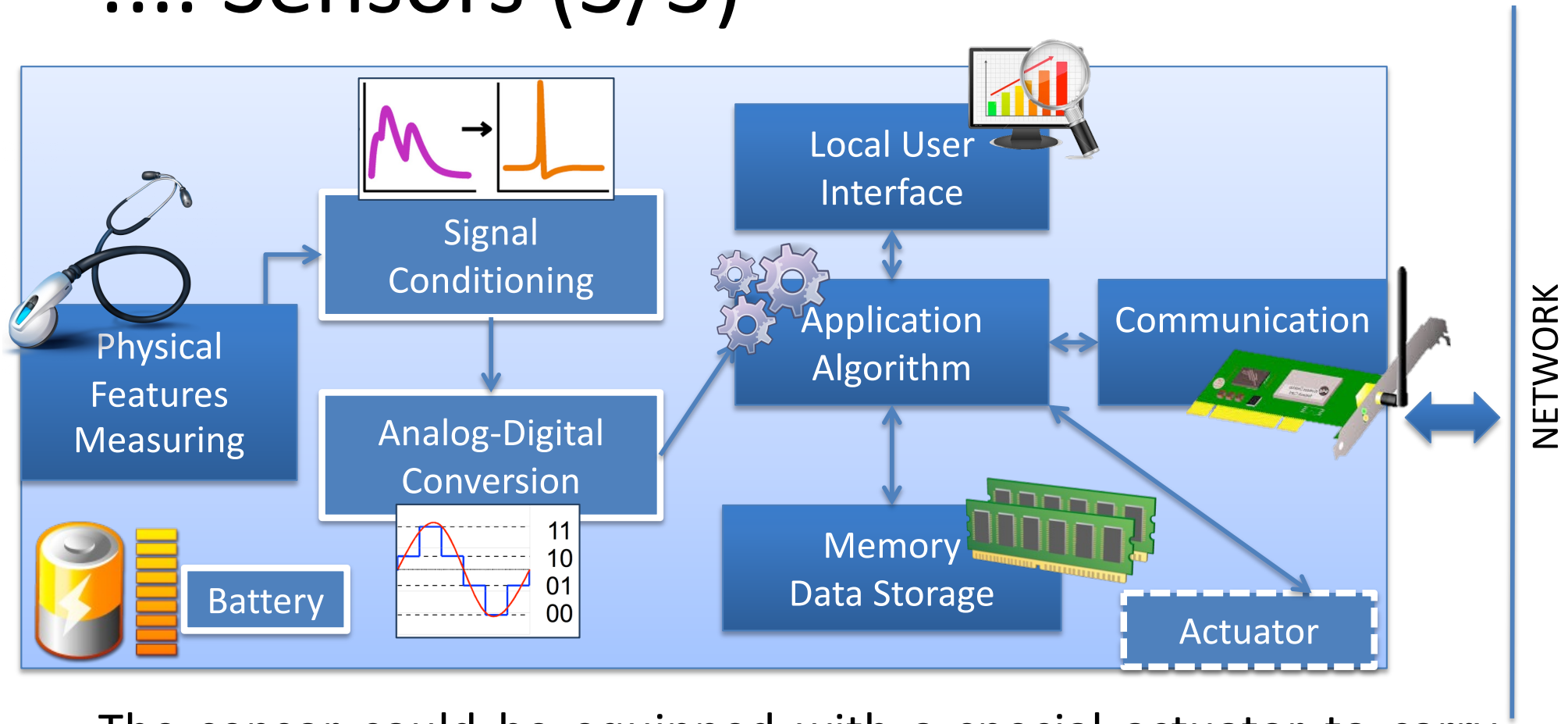
Measurement signals must be processed with appropriate operations (eg amplification, filtering, level adaptation). The main purpose is to optimize them so as to be compatible for the subsequent analog-to-digital conversion that makes them readable through software tools.

::.. Sensors (3/3)



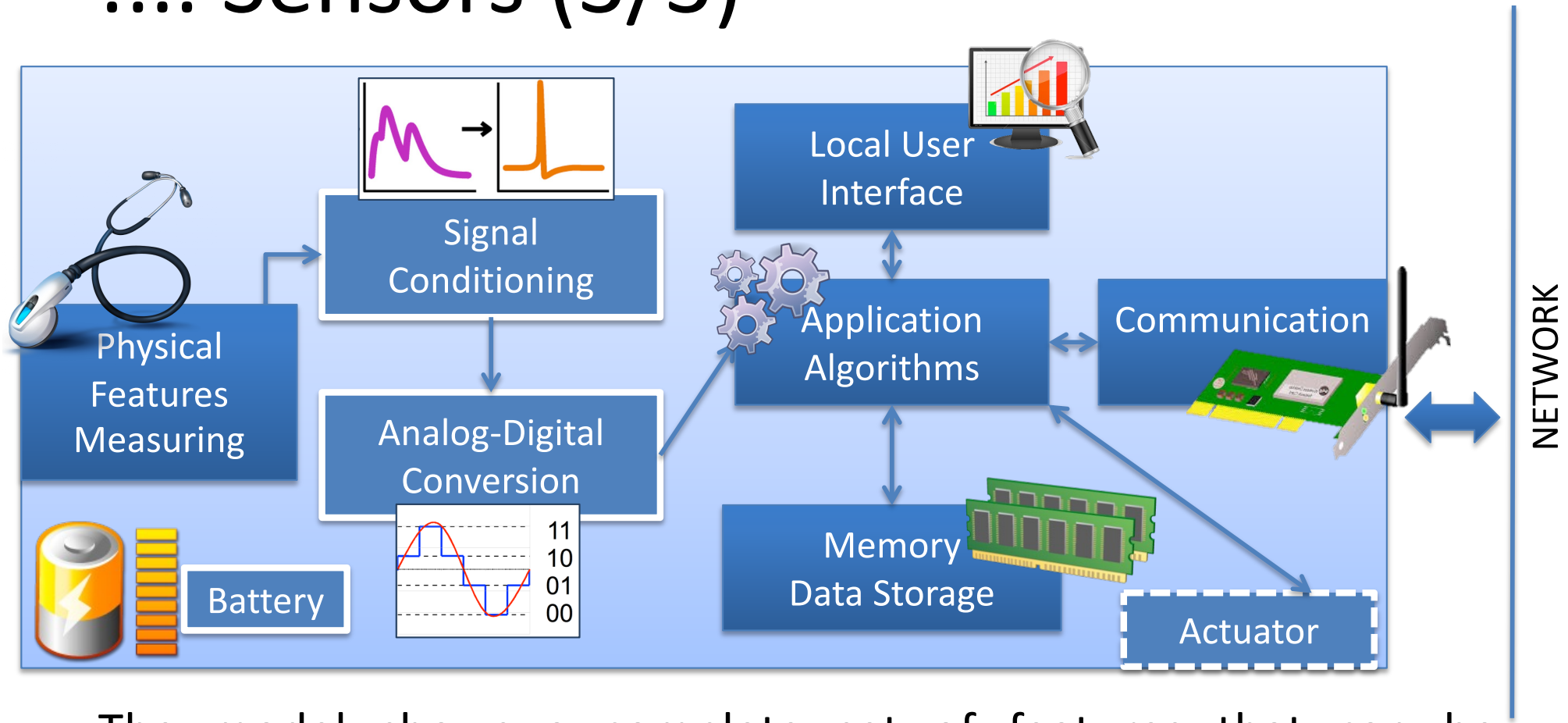
The sensor has an internal power supply battery, being rechargeable or not, by means of a renewable source, or by connecting the sensor to the mains.

::.. Sensors (3/3)



The sensor could be equipped with a special actuator to carry out actions on the monitored phenomenon in order to be able to make appropriate control strategies based on the monitoring data.

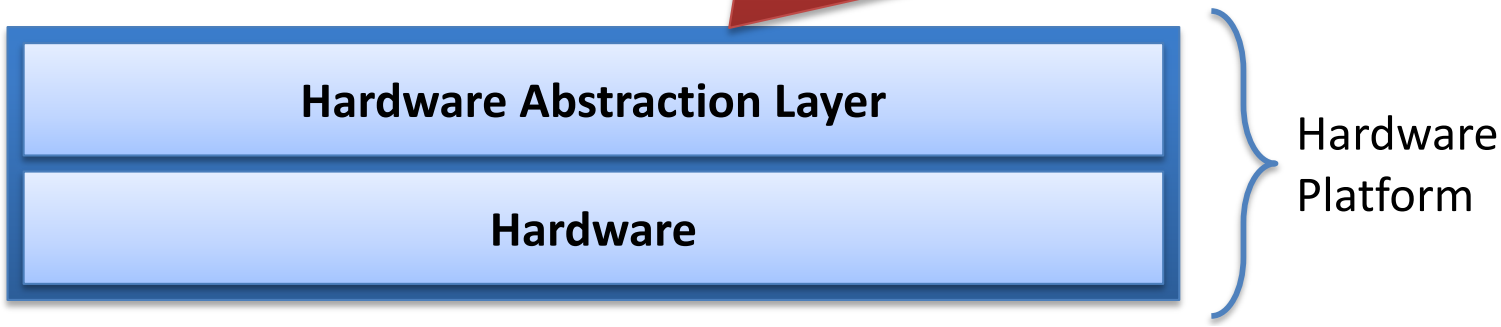
::.. Sensors (3/3)



The model shows a complete set of features that can be identified within a sensor, and wants to be as general as possible. Nothing prevents you from having intrinsically digital sensors without A / D conversion modules or particular signal conditioning circuits.

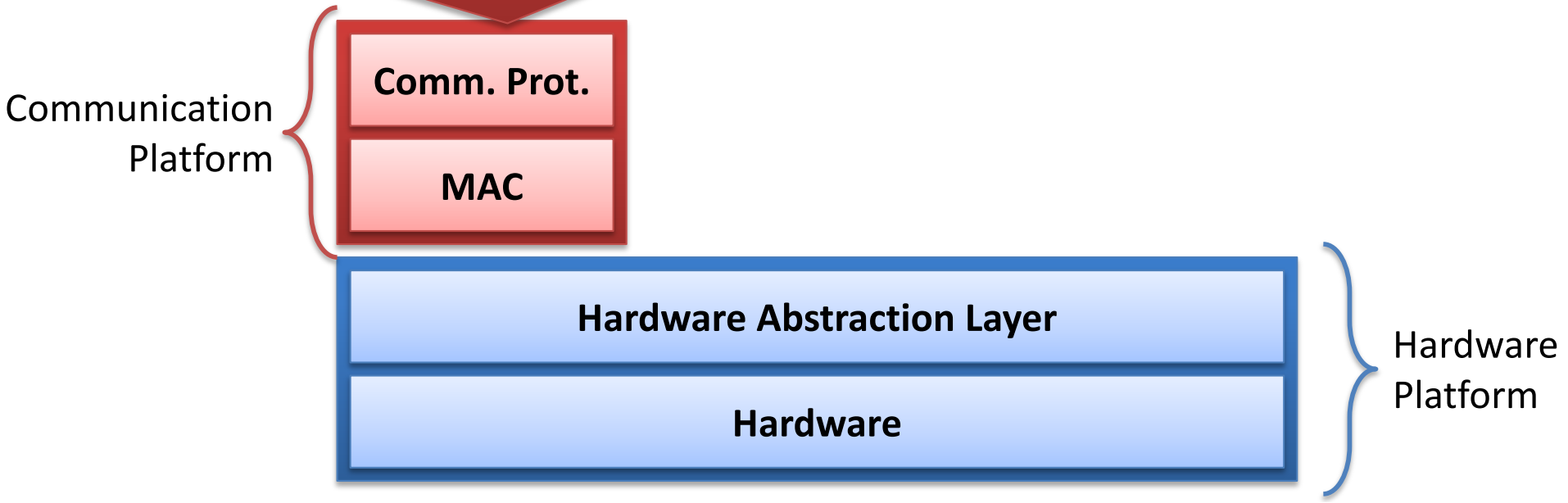
::.. Node Scheme

It represents the hardware with which the node was created, and its low-level programming interface with which to invoke the offered functionalities.



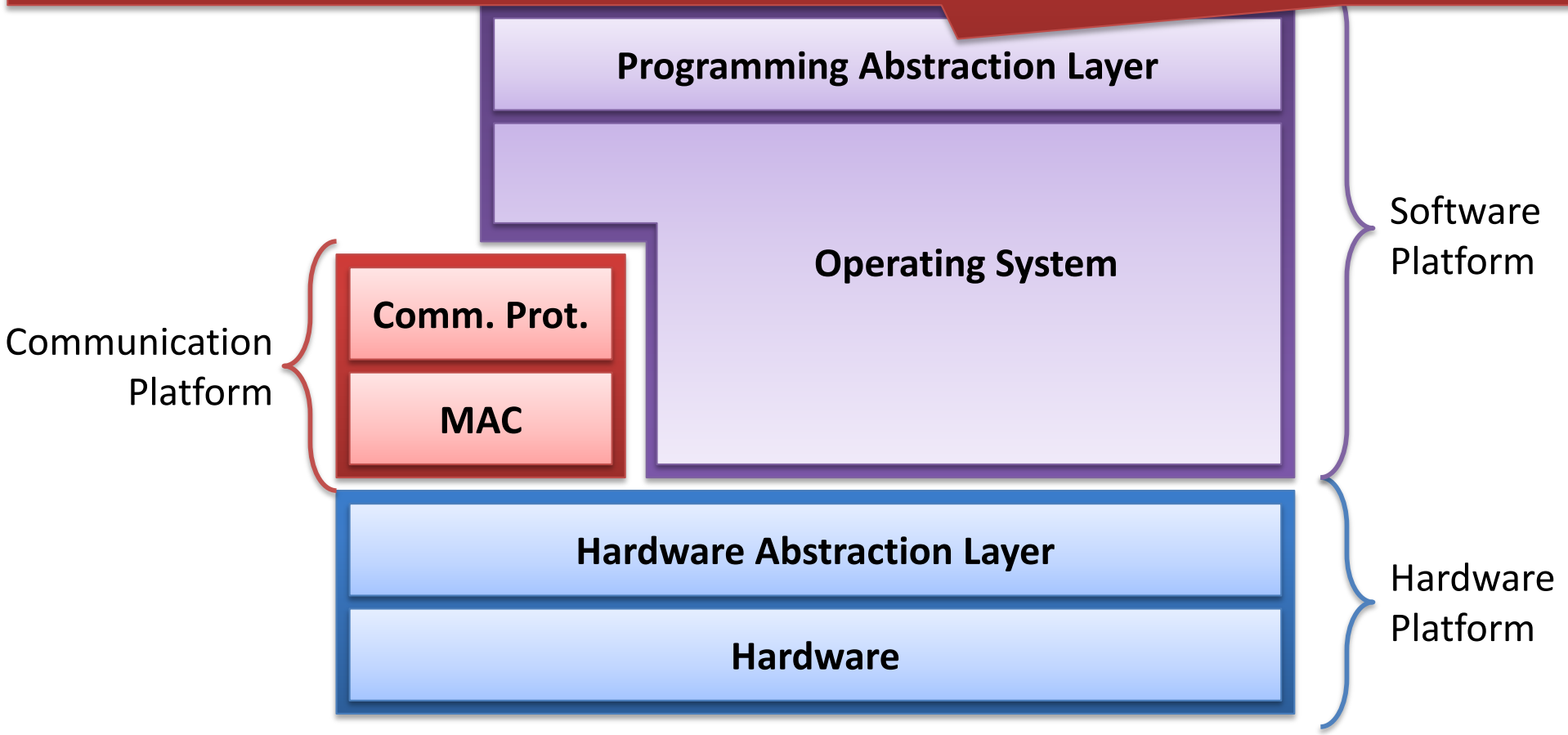
::.. Node Scheme

The communication support is composed of a series of algorithms for efficient access to the communication medium, taking into account energy saving requirements, and a set of network and transport protocols for wireless/cellular communication.

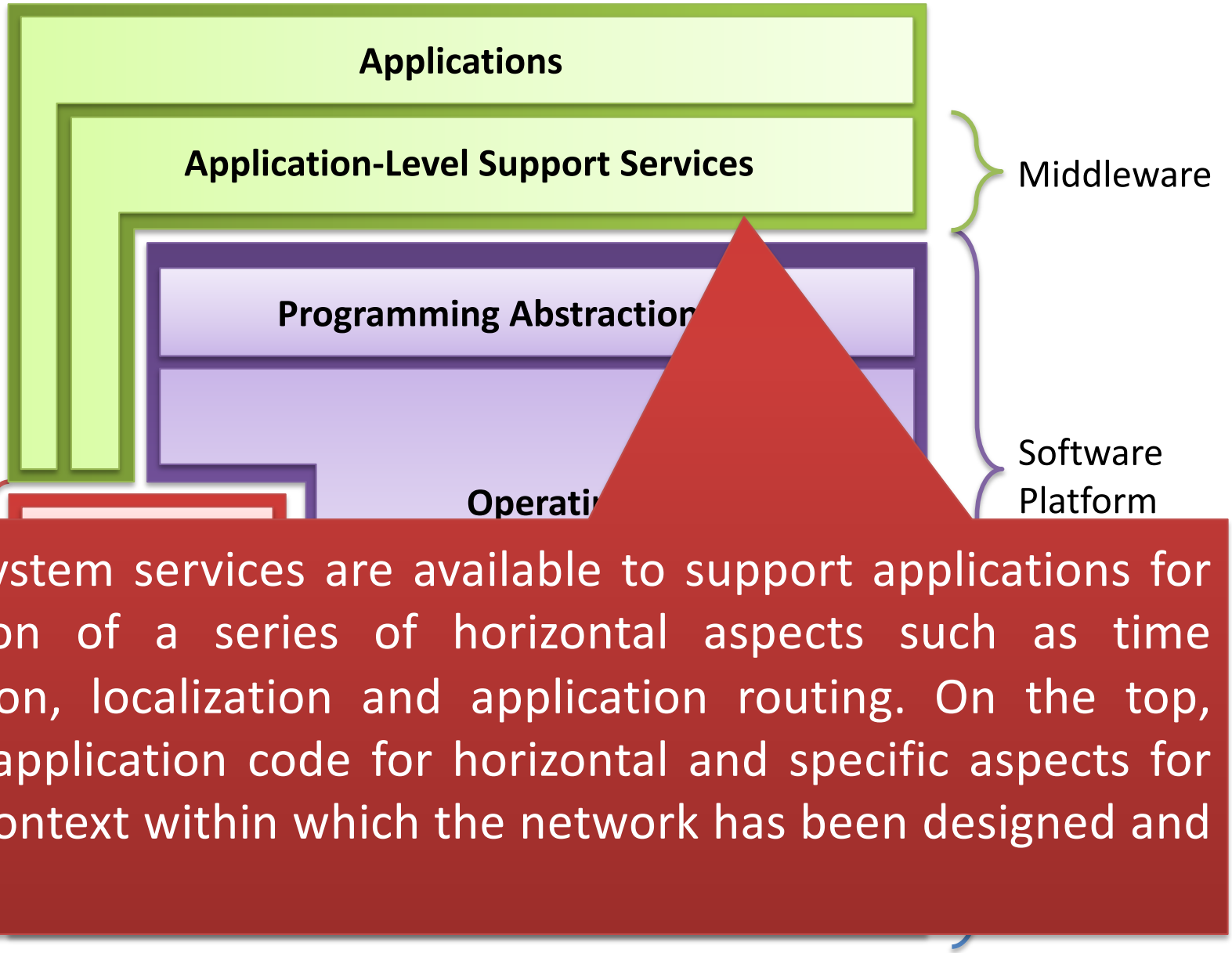


::.. Node Scheme

On sensor nodes, it represents a library linked to the application code to produce a binary for the execution of applications on the network node. It is usually supported by an ad-hoc programming language.

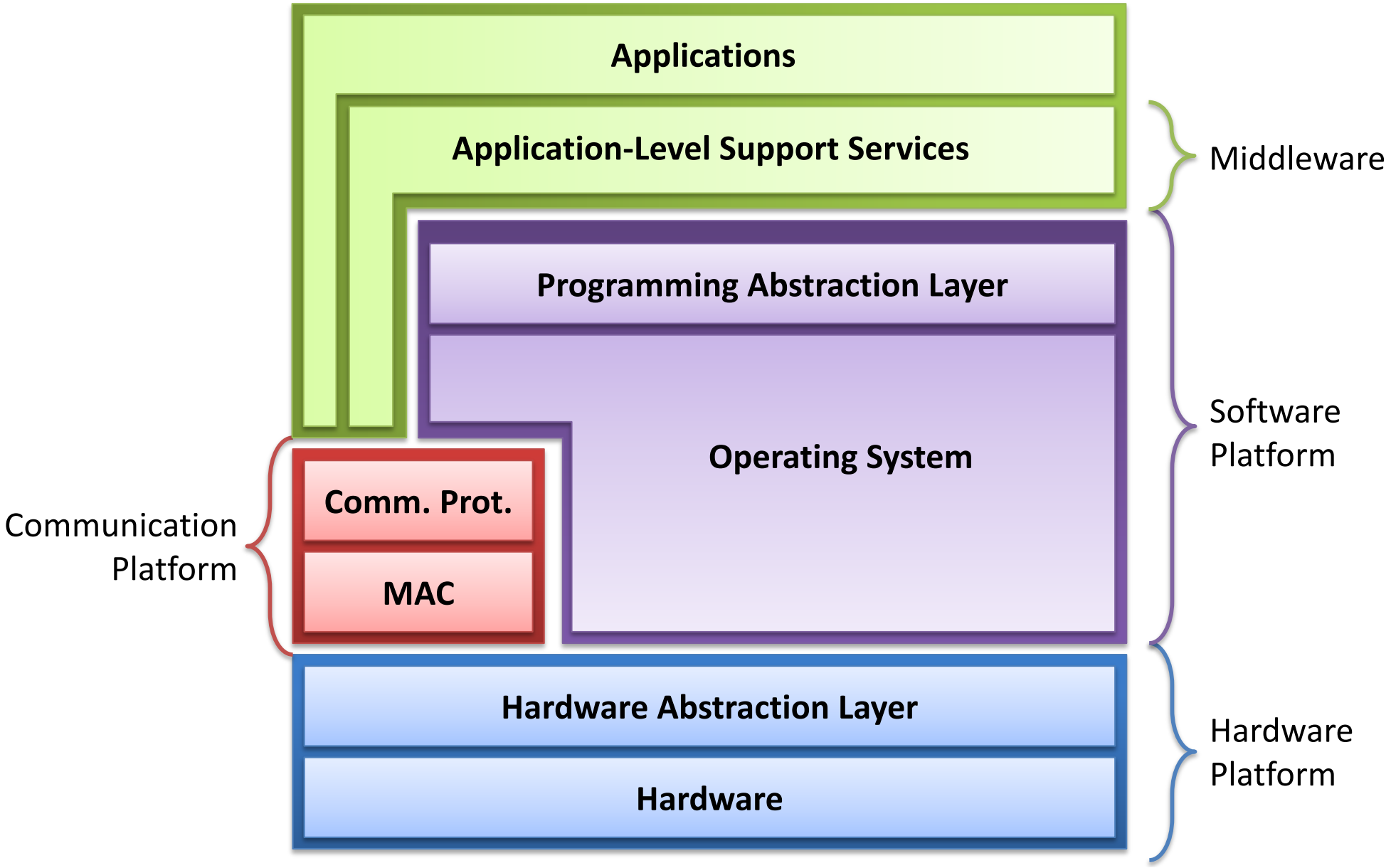


::.. Node Scheme



A series of system services are available to support applications for the realization of a series of horizontal aspects such as time synchronization, localization and application routing. On the top, there is the application code for horizontal and specific aspects for the applied context within which the network has been designed and used.

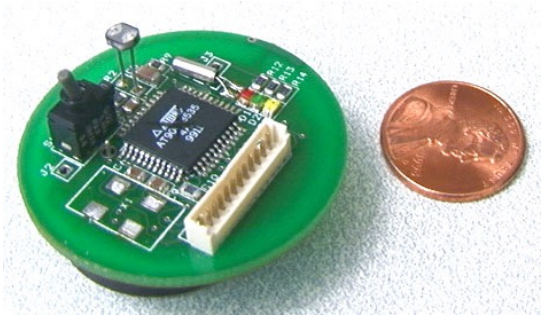
::.. Node Scheme

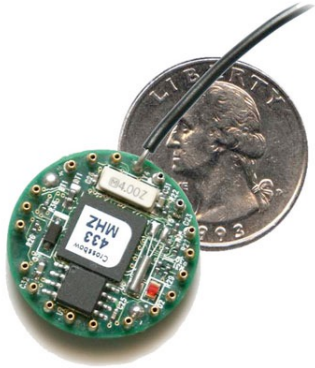

Applications
Application-Level Support Services
Middleware
Programming Abstraction Layer
Operating System
Software Platform
Comm. Prot.
MAC
Communication Platform
Hardware Abstraction Layer
Hardware
Hardware Platform

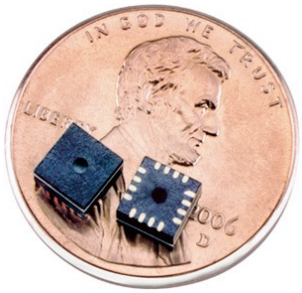
Sensor Hardware

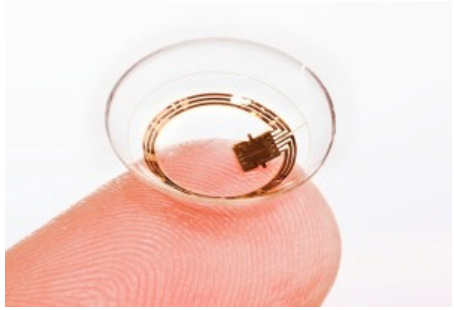
::.. Hardware Platforms (1/2)

Currently, various hardware platforms are available for the realization of sensor nodes to be integrated in order to realize distributed sensor networks.









WeC 1999 — Dot 2001 — CST sCap1 MEMS 2005 — micro MEMS 2010

We have witnessed the progressive miniaturization of this hardware, especially with the advent of Micro-Electro-Mechanical Systems (MEMS) technology, which can be defined as miniaturized mechanical and electro-mechanical elements, made using microfabrication.

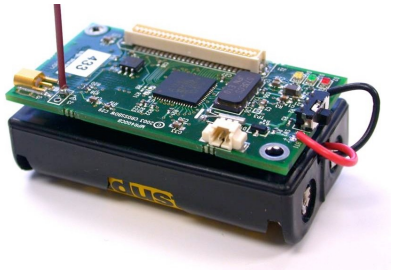
::.. Hardware Platforms (2/2)

We can classify the hardware platforms that can be used in sensor networks distributed in two classes:

::.. Hardware Platforms (2/2)

We can classify the hardware platforms that can be used in sensor networks distributed in two classes:

- General purpose hardware - it includes generic and programmable platforms;



::.. Hardware Platforms (2/2)

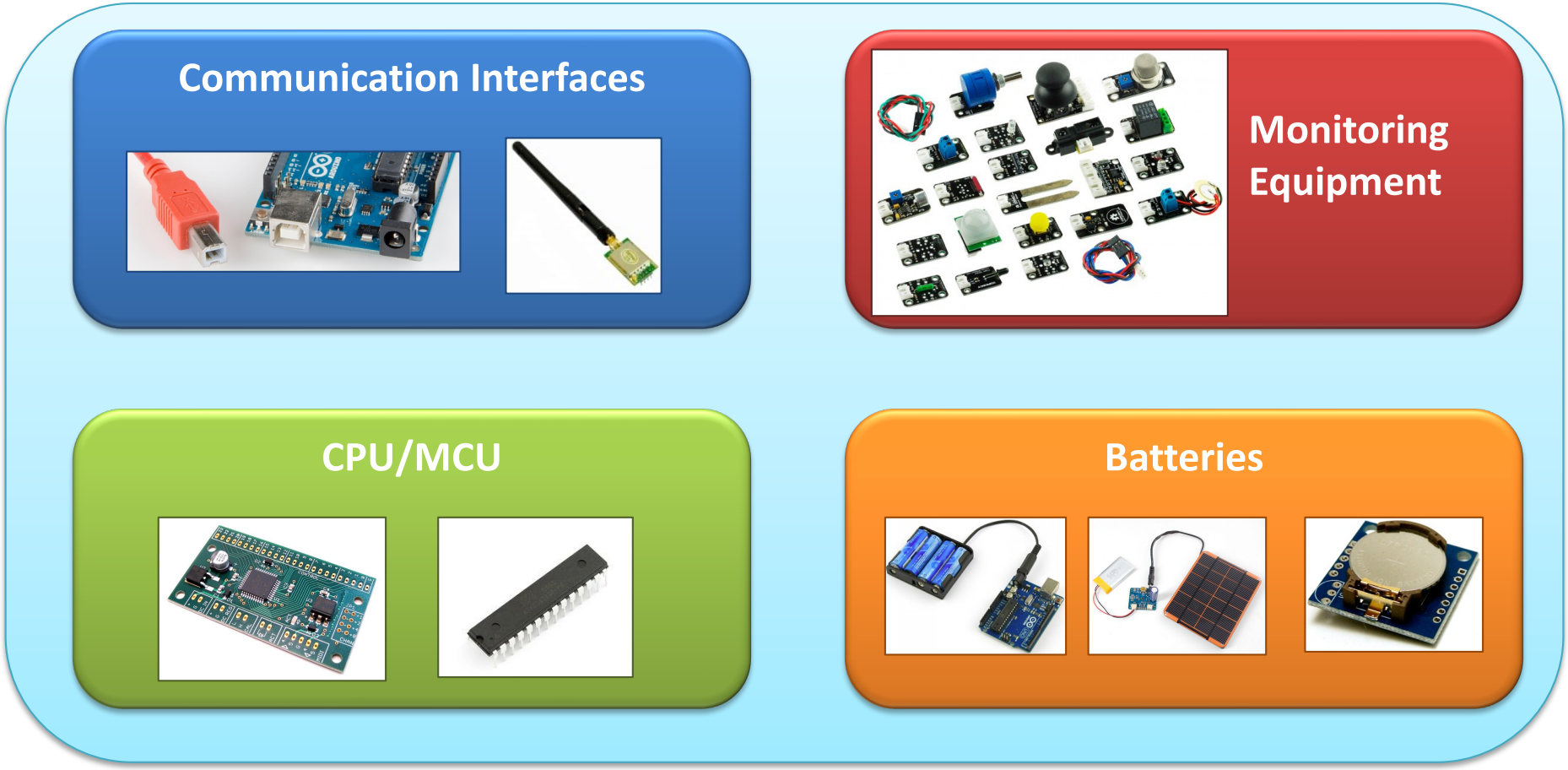
We can classify the hardware platforms that can be used in sensor networks distributed in two classes:

- General purpose hardware - it includes generic and programmable platforms;

- Ad-hoc hardware for specific monitoring and control tasks.

::.. Hardware ad Hoc (1/8)

As for the ad hoc hardware, we can outline it in four fundamental components:



Sensor Node

::.. Hardware ad Hoc (1/8)

As for the ad hoc hardware, we can outline it in four fundamental components:

The Central Processing Unit (CPU) is the main element with the task of conducting processing operations (ie executing the code). The MicroController Unit (MCU) provides additional features such as Timers, Oscillators, Memories (of program such as ROM, EPROM, FLASH or data such as RAM and EEPROM), I/O Ports and/or General Purpose Input/Output (GPIO) and more.

Sensor Node

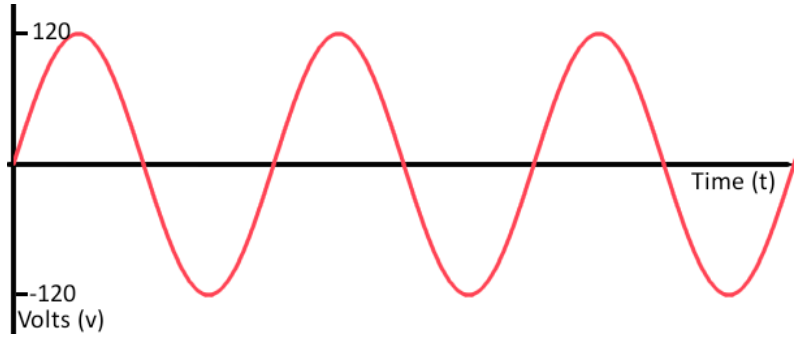
::.. Hardware ad Hoc (2/8)

The monitoring equipment can be classified as

::.. Hardware ad Hoc (2/8)

The monitoring equipment can be classified as

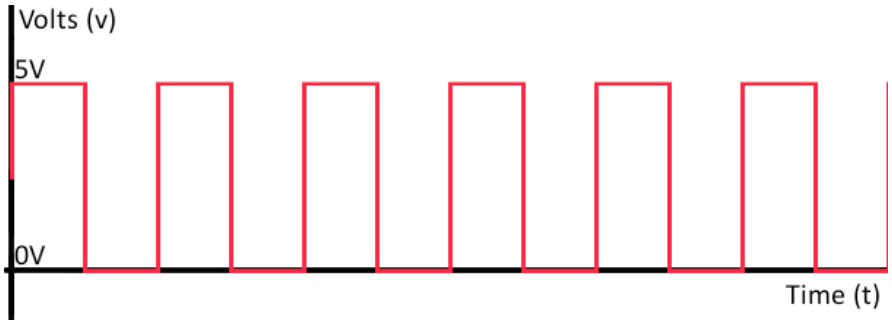
- analogue, when the electrical quantity produced in output varies continuously in response to the variation in the input quantity.

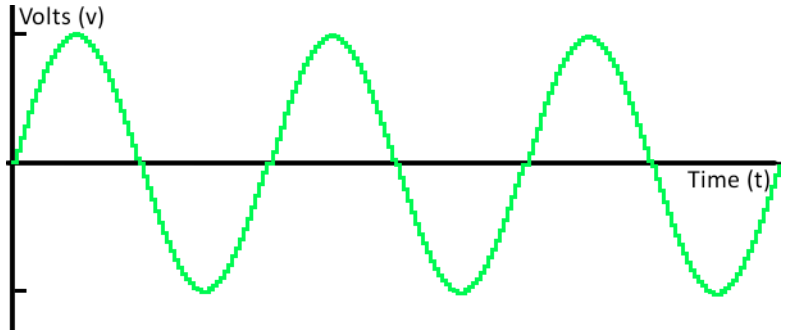


::.. Hardware ad Hoc (2/8)

The monitoring equipment can be classified as

- analogue, when the electrical quantity produced in output varies continuously in response to the variation in the input quantity.
- digital, when the possible output values are limited or are a discrete representation of an analog waveform. In the simplest case, that of the so-called on-off sensors, the output values can be only two.

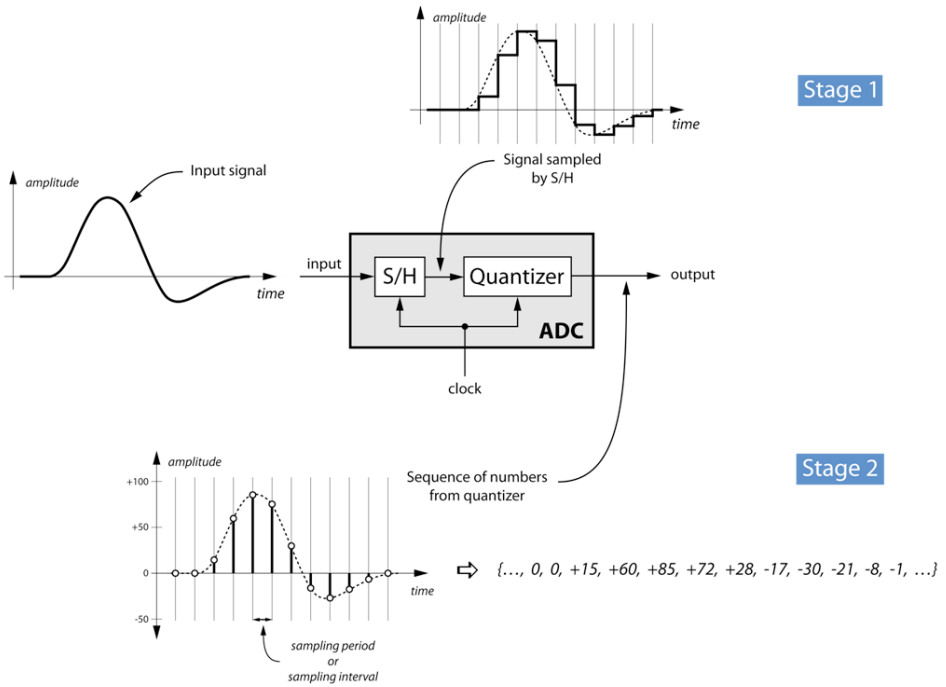




::.. Hardware ad Hoc (3/8)

The distinction between the two types of apparatus is important as the signal processing techniques produced by monitoring are remarkably different:

- the signal produced by a digital sensor can be processed directly by a digital circuit



- the signals produced by the analog sensors must be treated by analog circuits (filters, conditioning circuits, amplifiers). They can then be converted digitally (with analog-to-digital converters - ADC).

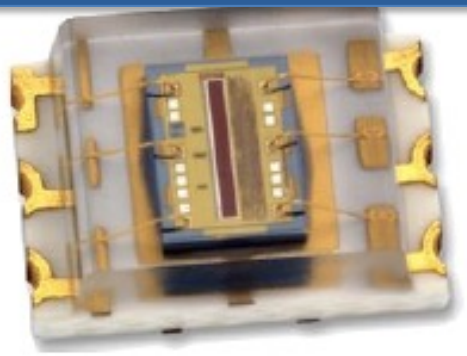
::.. Hardware ad Hoc (4/8)

The monitoring equipment can also be classified considering the monitoring objective.

Accelerometer
They return the acceleration along the 2 or 3 axes. They can be either digital or analog.

Temperature sensor
They return a temperature measurement, generally in an analogical way.

Camera
They provide images, or their portions, and generally require an external clock, configured and used with digital interfaces.



Light sensors
They provide light intensity. They can be connected digitally, or even analog.

GPS
They provide geoposition, to be filtered and interfaced with digital connections.

::.. Hardware ad Hoc (5/8)

The digital interconnections with the monitoring apparatus are made using standard protocols:

- U(S)ART - Universal (Syncronous) Asyncronous Receive Transmit: module to communicate asynchronously and synchronously with another module both transmitting and receiving data according to a protocol. Low speed.
- I2C - Inter Integrated Circuit: protocol that provides for the "exchange" of data between one or more Masters and one or more Slave slaves, by means of a data bus and a clock one. Three speeds are possible (100 Khz, 400 Khz, 1 Mhz).
- SPI - Serial Peripheral Interface: Master-Slave protocol, with each one equipped with a shift register containing the data and connected with a minimum of four buses. The transfer involves the exchange of the contents of the register.

::.. Hardware ad Hoc (6/8)

The possible communication interfaces with devices external to the sensor node can be of two types:

::.. Hardware ad Hoc (6/8)

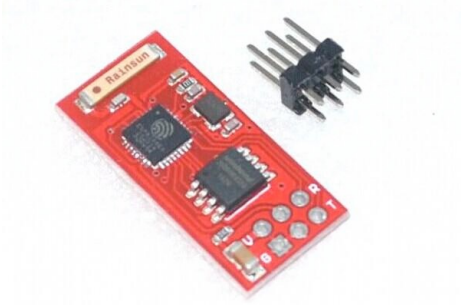
The possible communication interfaces with devices external to the sensor node can be of two types:

- Wire communication - the sensor node is interconnected with the other components by means of cables. The most famous example is represented by the Universal Serial Bus (USB), a serial communication standard designed to allow multiple devices to be connected through the use of a single standardized interface and a single type of connector, allowing to connect or unplug devices without having to restart the computer (hot swap).

::.. Hardware ad Hoc (6/8)

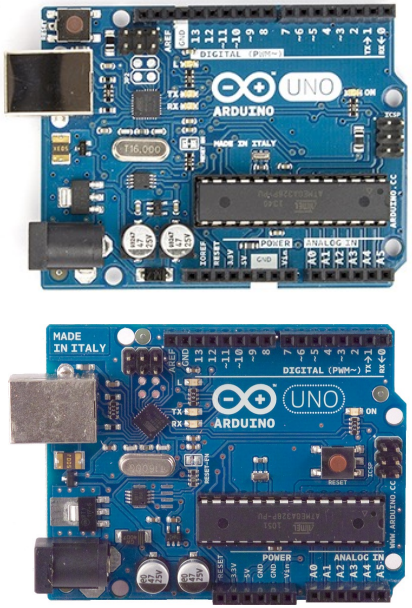
The possible communication interfaces with devices external to the sensor node can be of two types:

- Wire communication - the sensor node is interconnected with the other components by means of cables.
- Wireless communications - the sensor node is interconnected without the use of cables, but by means of infrared or radio frequency technologies. Examples are Bluetooth ZigBee. It reduces network installation costs, but makes it more vulnerable to interference and data interception.



::.. Hardware ad Hoc (7/8)

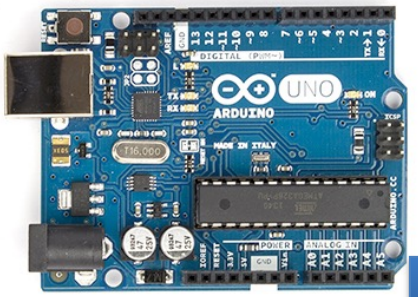
These various components are integrated and assembled (connecting the various input / output pins) using a board, such as those of Arduino.

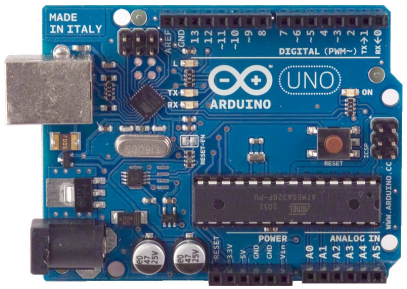


Board Arduino of the UNO family

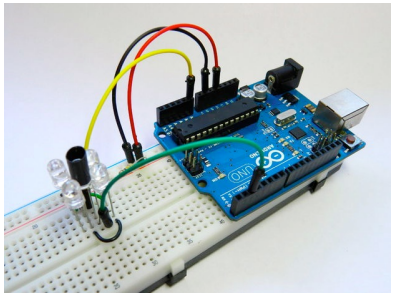
::.. Hardware ad Hoc (7/8)

These various components are integrated and assembled (connecting the various input / output pins) using a board, such as those of Arduino.

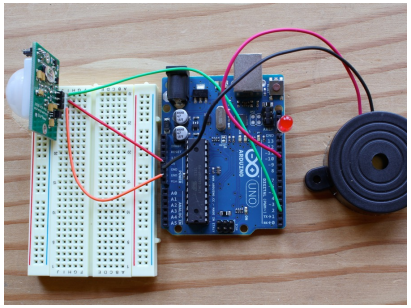




Board Arduino of the UNO family

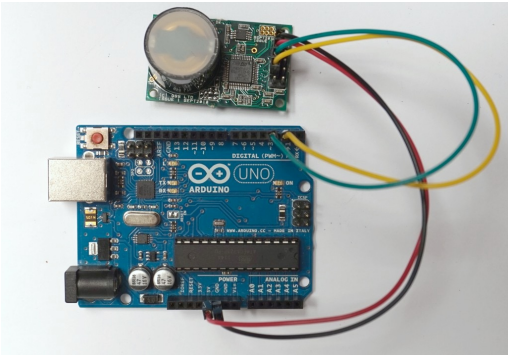


Infrared proximity sensor



Motion sensor, with sound alarm

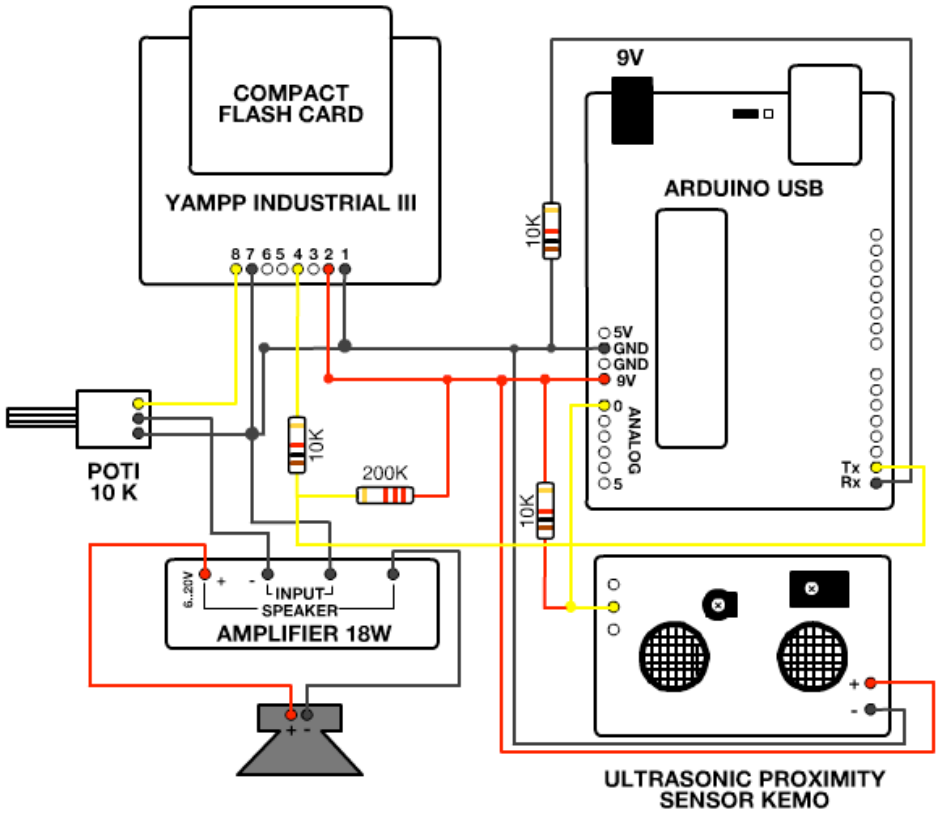
Radiation monitoring sensor with RF antenna



CO2 monitoring sensor

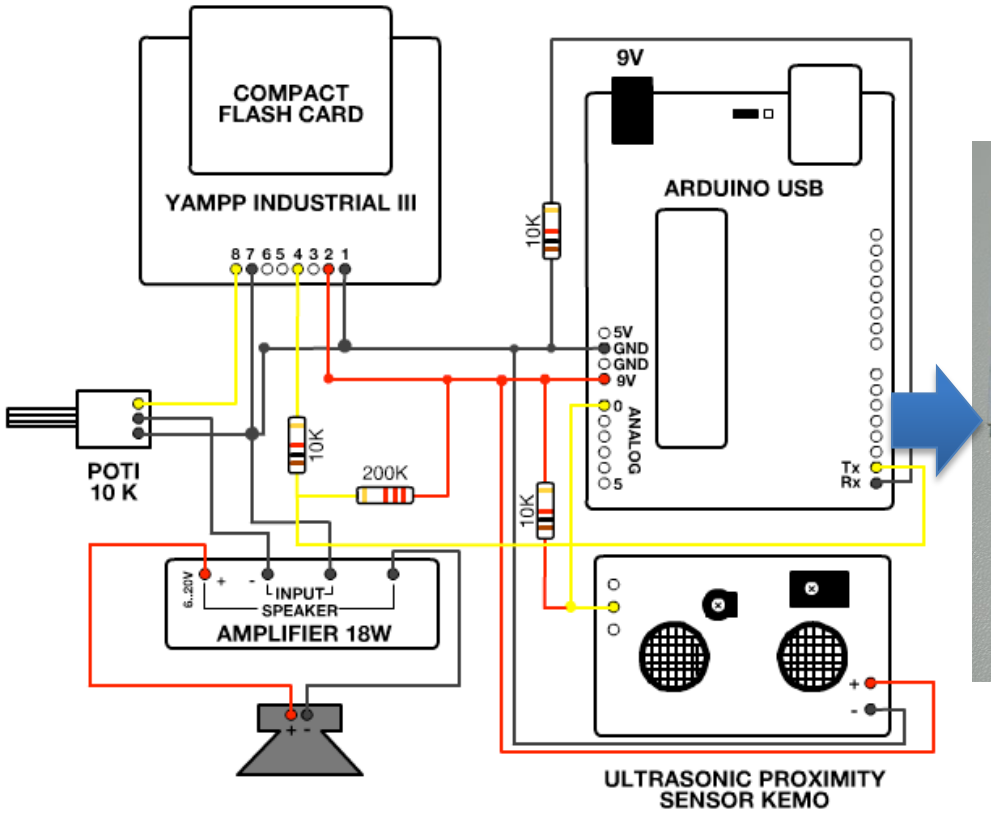
::.. Hardware ad Hoc (8/8)

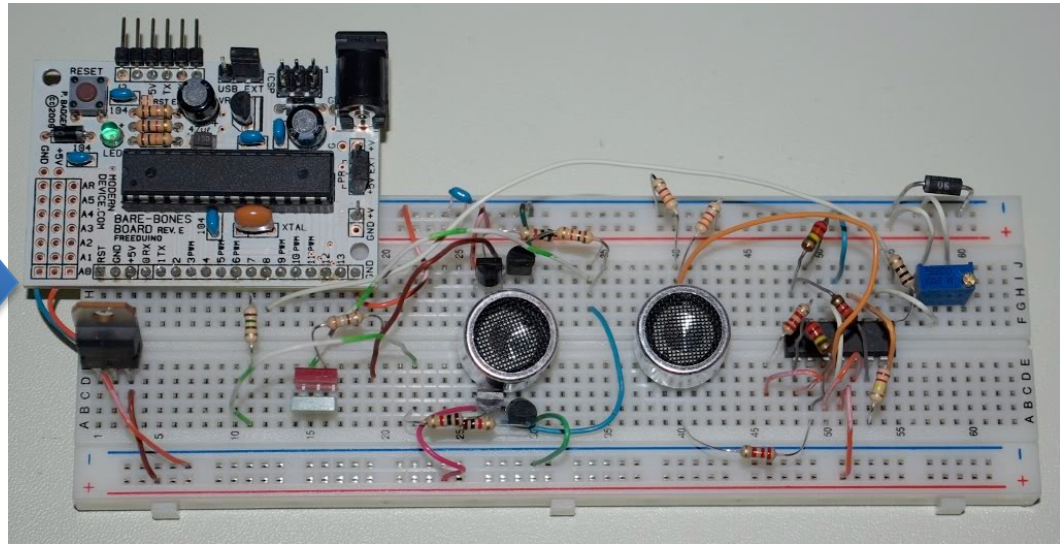
The ad hoc hardware design translates into the identification of the necessary components and a schematization of the interconnections between them:



::.. Hardware ad Hoc (8/8)

The ad hoc hardware design translates into the identification of the necessary components and a schematization of the interconnections between them:





::.. Hardware Abstraction Layer (1/4)

The programs running on the sensor nodes must be able to recall the hardware features offered by the node and execute the application logic, which specifies the operations that the node must perform when deployed in the context of the network.

Usually a special hardware abstraction is offered, especially in the case of ad-hoc hardware. For example, the Arduino platform offers a programming language deriving from C.

In the case of generic hardware there is a tendency not to offer the programmer a low-level programming environment. In fact, it is required that the programmer yields the operating system, or a special application built above it, so as to simplify the implementation of the code for the sensor node. E.g., MICAz and Imote2 use TinyOS.

::.. Hardware Abstraction Layer (2/4)

Let's see an example of a program written in the language derived from the C of Arduino. This program consists of two parts or functions containing instructions:

```
void setup()
{
  statements;
}
void loop()
{
  statements;
}
```

::.. Hardware Abstraction Layer (2/4)

Let's see an example of a program written in the language derived from the C of Arduino. This program consists of two parts or functions containing instructions:

```
void setup()
{
  statements;
}
```

It is the first function to be executed and has the purpose of declaring the variables used in the program and in the hardware settings, such as the initialization of the pin modes.

```
void loop()
{
  statements;
}
```

::.. Hardware Abstraction Layer (2/4)

Let's see an example of a program written in the language derived from the C of Arduino. This program consists of two parts or functions containing instructions:

```
void setup()
{
  statements;
}
```

```
void loop()
{
  statements;
}
```

Executed after the setup() function and contains the code to be executed continuously by implementing the application logic, reading the inputs and returning the outputs.

::.. Hardware Abstraction Layer (3/4)

In addition to these standard functions, there are functions that the user can declare, or retrieve from special libraries, by organizing their program in a modular way. Consider the following example of the delayVal () function that reads the value of a potentiometer, which returns a number between 0 and 1023, divides the value by 4 so as to return a final value between 0 and 255 to the caller:

```
int delayVal()
{
  int v;                    // create the temporary variable v
  v = analogRead(pot);      // read the value of potentiometer
  v /= 4;                   // convert 0-1023 in 0-255
  return v;                 // return the result
}
```

::.. Hardware Abstraction Layer (3/4)

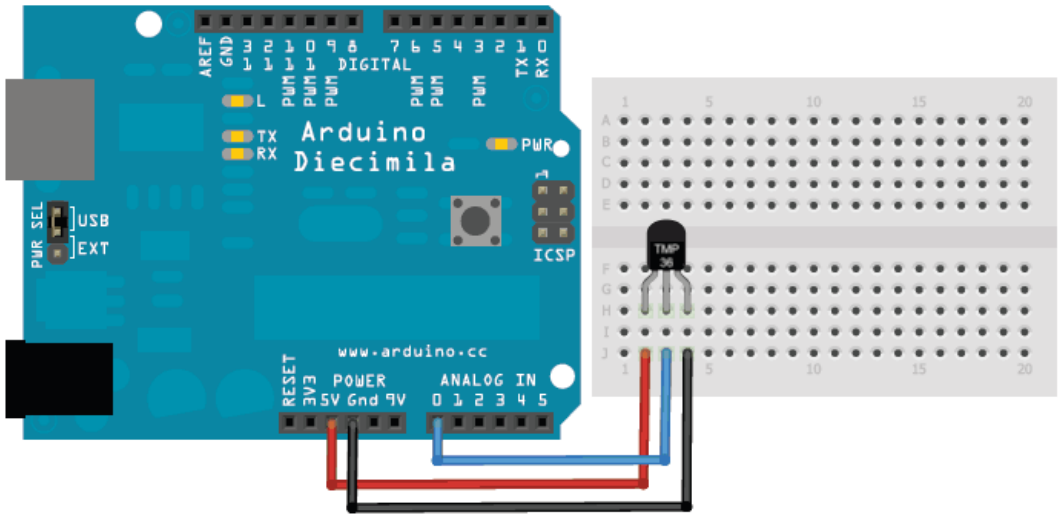
In addition to these standard functions, there are functions that the user can declare, or retrieve from special libraries, by organizing their program in a modular way. Consider the following example of the delayVal () function that reads the value of a potentiometer, which returns a number between 0 and 1023, divides the value by 4 so as to return a final value between 0 and 255 to the caller:

```
int delayVal()
{
  int v;
  v = analogRead(pot);          // read the value of potentiometer
  v /= 4;                       // convert 0-1023 in 0-255
  return v;                     // return the result
}
```

A series of built-in functions are available for the realization of the most recurring operations.

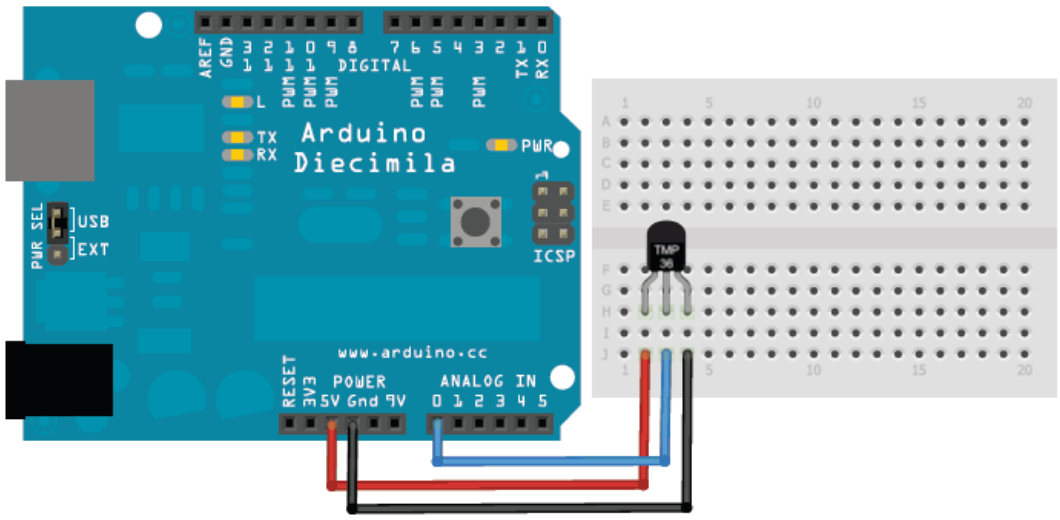
::.. Hardware Abstraction Layer (4/4)

Example of a sensor with Arduino
for temperature monitoring:



::.. Hardware Abstraction Layer (4/4)

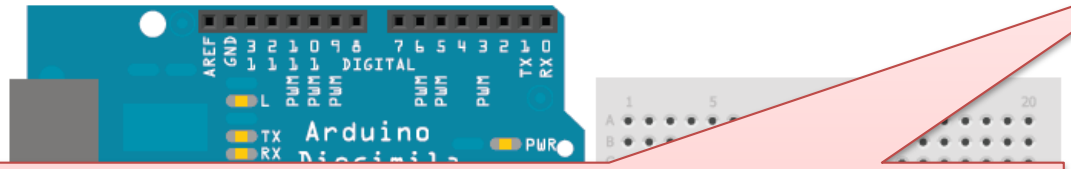
Example of a sensor with Arduino for temperature monitoring:




```
int val;
int tempPin = 1;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  val = analogRead(tempPin);
  float mv = ( val/1024.0)*5000;
  float cel = mv/10;
  float farh = (cel*9)/5 + 32;
  Serial.print("TEMPRATURE = ");
  Serial.print(cel);
  Serial.print("*C");
  delay(1000);
}
```

::.. Hardware Abstraction Layer (4/4)

Example of a sensor with Arduino for temperature monitoring:



It represents the communication between the Arduino board and a computer, or another device, by means of a serial port. It is possible to invoke a series of functions to set the transmission speed in bits (with begin), or for writing (with println/ write) and reading (with read) of data from the connection.

```
int val;
int tempPin = 1;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  val = analogRead(tempPin);
  float mv = ( val/1024.0)*5000;
  float cel = mv/10;
  float farh = (cel*9)/5 + 32;
  Serial.print("TEMPRATURE = ");
  Serial.print(cel);
  Serial.print("*C");
  delay(1000);
}
```