

# Internet of Things

## Design goals and approaches

Tullio Facchinetti  
<tullio.facchinetti@unipv.it>

Wednesday 22<sup>nd</sup> February, 2023

<http://robot.unipv.it/toolleoo>

## Common solutions to monitoring problems

Typical options that are available to address the problem include:

- “Traditional” human operations
- Robotic automation
- IoT approach

All options are reasonable.  
The decision is driven by cost/benefit trade-offs.

# Technology convergence

Convergence of

- **Miniaturized embedded devices:** integrate processor, sensors and radios.
- **Low-power wireless:** connect millions of devices to the Internet.
- **Cloud:** scalable computing for big data processing.
- **Data analytics:** extract meaningful information from sensor data.

## Design approach

Full IoT stack includes:

- 1 Monitoring of the physical environment
- 2 Measurement of relevant parameters
- 3 Sending of measurements to a remote cloud infrastructure
- 4 Collection of data in the data warehouse
- 5 Data processing to extract meaningful/useful information
- 6 Use of information to *take decisions*, either manually or automatically

# Internet of Things



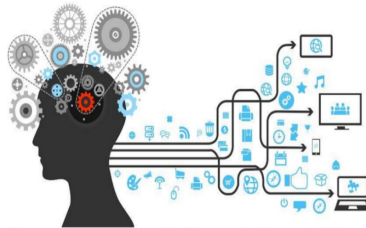
**Embedded devices**



**Wireless connectivity**

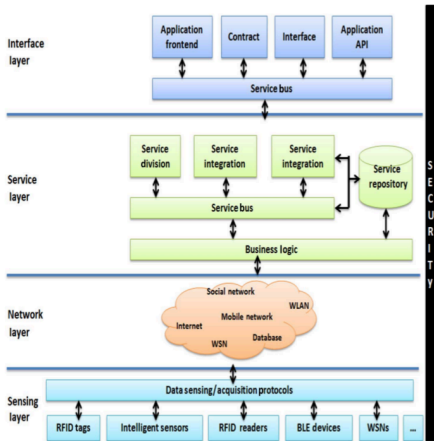


**Cloud**



**Data analytics**

# Service Oriented Architecture



L. Xu, W. He, S. Li, "Internet of Things in Industries", IEEE Transactions on Industrial Informatics, 2014.

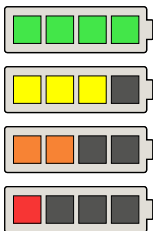
## Design goals

- Lifetime
- Performance (latency and throughput)
- Scalability
- Topology
- Security and safety
- Mobility
- Reliability
- Management
- Availability
- Interoperability

## Design goals - Lifetime

### Lifetime

How long can a IoT device operate with the available power supply?



- IoT devices are typically battery powered.
- Energy is used for computation, sensing and communication.
- Energy management is critical to achieve the continuous operation of the system.



## Design goals - Performance

### Latency

How long does it take to propagate and process the messages?

### Throughput

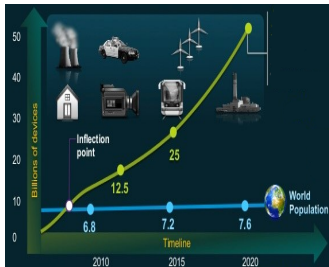
What is the maximum amount of data that can be transported over the network?

- The amount of data to transmit may be significant due to the large number of nodes.
- There could be timing constraints in the delivery of information or commands
- The communication typically happens over a wireless, unreliable medium.

## Design goals - Scalability

### Scalability

How many devices are supported?

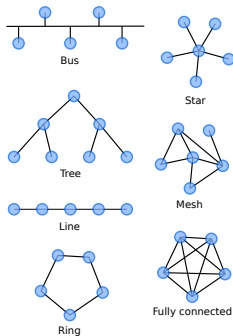


- IoT applications may need to handle millions and trillions of devices, connected on the same network.
- Their distribution must be properly managed.
- IoT applications should be tolerant of new services and devices constantly joining the network.
- Must be designed to enable extensible services and operations.

## Design goals - Topology kernel

### Topology kernel

Who can/must  
communicate with whom?



- The units composing an IoT can be connected in different way
- The **topology** describes the physical and logical structure of the networked units.
- Several aspects are related to the topology: organization of the units, their geographical displacement, the selected communication protocols, and the requirements of the application.

## Design goals - Security and safety

### Security and safety

How secure (prevention of malicious activities by people) and safe (prevention of non-intentional accidents) is the overall system?

- IoT systems can be used in critical applications (e.g., health/military).
- Correct operations are necessary to guarantee the behavior of the process.
- The reliability of both hardware and software components is a key factor.
- The access to the data must be protected (encryption).
- Policy management is challenging due to the many components involved in a IoT system.

## Design goals - Mobility

### Mobility

How to deal with moving devices?

- IoT devices often (not always) need to move freely.
- Their IP address and networks change based on their location.
- The routing tables (DODAG - Destination Oriented DAG in RPL) must be reconstructed each time a node goes off the network or joins the network, adding huge overhead.
- Mobility might result in a change of service provider, which can add another layer of complexity due to service interruption and changing gateway.

## Design goals - Reliability

### Reliability

Is the system capable of delivering the expected services without dysfunctioning?

- A reliable system should work perfectly and deliver all of its specifications correctly.
- It is critical in applications requiring emergency responses.
- IoT applications require highly reliable and timely collection of data, since important decisions may be based on such data.

## Design goals - Management

### Management

Are all the necessary operational aspects correctly tracked?

- Management includes keeping track of the failures, configurations, and performance.
- This is challenging in IoT applications due to the large number of devices.
- Providers should manage Fault, Configuration, Accounting, Performance and Security (FCAPS) of their interconnected devices and account for each aspect.

## Design goals - Availability

### Availability

Are services correctly available to the authorized users?

- Availability of IoT includes software and hardware levels being provided at anytime and anywhere for service subscribers.
- Software availability: the service is provided to anyone who is authorized to have it.
- Hardware availability: the existing devices are easy to access and are compatible with IoT functionality and protocols.
- Protocols should be compact to be able to be embedded within the IoT constrained devices.



## Design goals - Interoperability

### Interoperability

How making heterogeneous devices and protocols need to inter-work with each other?

- The challenge is due to the large number of different platforms used in IoT systems.
- Interoperability should be handled by both the application developers and the device manufacturers to deliver the services regardless of the platform or hardware specification used by the customer.

## Design approach

Full IoT stack includes:

- 1 **Monitoring of the physical environment**
- 2 Measurement of relevant parameters
- 3 Sending of measurements to a remote cloud infrastructure
- 4 Collection of data in the data warehouse
- 5 Data processing to extract meaningful/useful information
- 6 Use of information to *take decisions*, either manually or automatically

## Disclaimer: technical content only!

The considerations regarding the various aspects related to IoT design approaches will focus on **technical/engineering aspects**

In general, additional aspects need to be considered, such as:

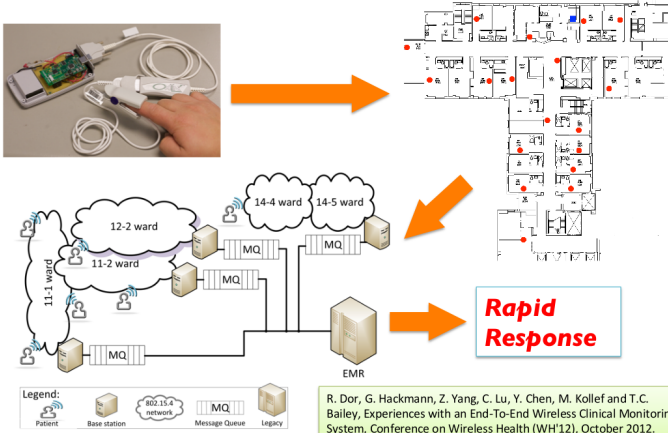
- costs
- availability on the market
- integration issues
- compatibility with existing systems
- ease of development
- support from vendors
- etc.

# 1. Monitoring of the physical environment

- What are the “things” to monitor?
- What are the physical characteristics of the environment?  
⇒ Indoor/outdoor, temperature, presence of water, harsh conditions, etc.
- Is there any constraint to the embedded device?  
⇒ Size, weight (e.g., wearable systems), operating temperature (e.g., in implantable systems), installation issues, etc.

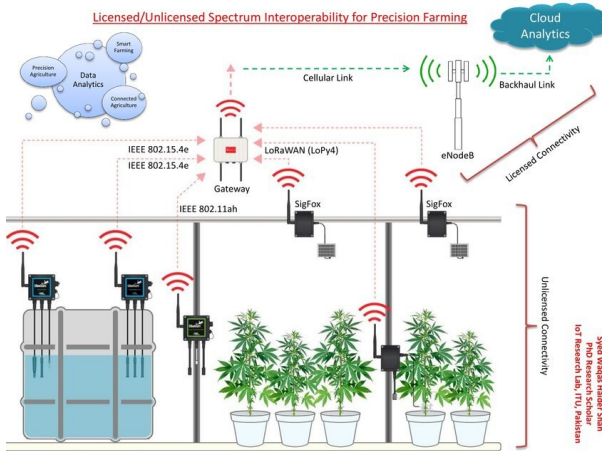
# 1. Monitoring of the physical environment

## Clinical warning application



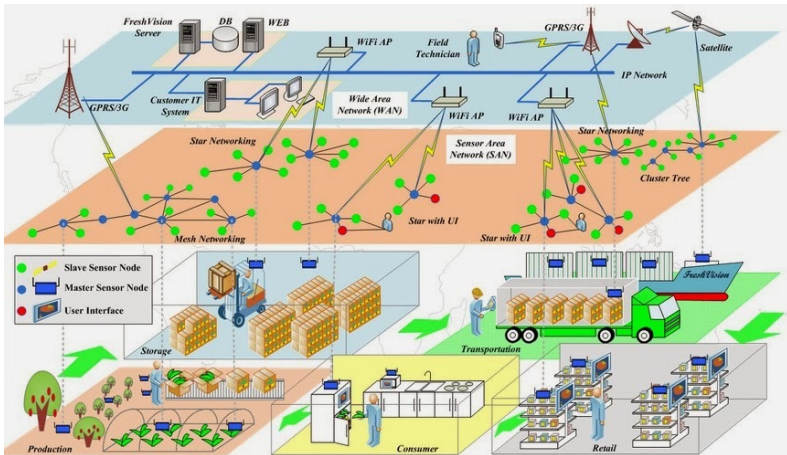
# 1. Monitoring of the physical environment

## Smart farming



# 1. Monitoring of the physical environment

## Logistics: transportation and warehousing management



## Design approach

Full IoT stack includes:

- 1 Monitoring of the physical environment
- 2 Measurement of relevant parameters
- 3 Sending of measurements to a remote cloud infrastructure
- 4 Collection of data in the data warehouse
- 5 Data processing to extract meaningful/useful information
- 6 Use of information to *take decisions*, either manually or automatically

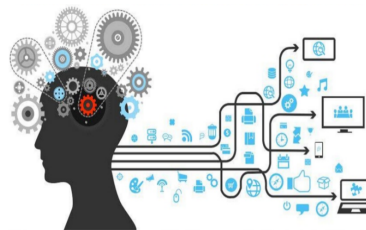
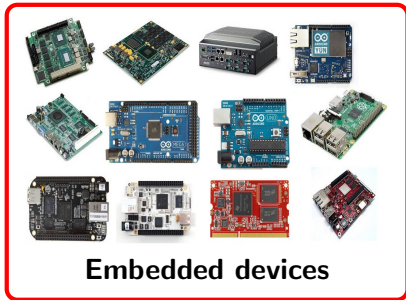


## 2. Measurement of relevant parameters

- What are the interesting parameters to measure?  
⇒ Use the necessary sensor technology.
- What is the sampling frequency?  
⇒ E.g. temperature measured every 5 minutes, acceleration measured every 10 ms
- What is the amount of data to collect?  
⇒ Amount of memory required by the embedded device
- Measured values require complex local processing?  
⇒ Signal processing, filtering, etc.

All these aspects affect the selection of the embedded device to use (processing power, memory, energy consumption), availability of sensors, etc.

## 2. Measurement of relevant parameters



## 2. Measurement of relevant parameters

### Smart sensors

---



Smart Agriculture Xtreme



Smart Water Xtreme

Source: <http://www.libelium.com/>

## Design approach

Full IoT stack includes:

- 1 Monitoring of the physical environment
- 2 Measurement of relevant parameters
- 3 **Sending of measurements to a remote cloud infrastructure**
- 4 Collection of data in the data warehouse
- 5 Data processing to extract meaningful/useful information
- 6 Use of information to *take decisions*, either manually or automatically

### 3. Sending of measurements to a remote cloud infrastructure

- What is the distance from the receiver?  
⇒ The IoT node is usually connected wirelessly
- What is the amount of data to send away?
- What is the frequency of the sending?
- What is the maximum tolerable latency?  
⇒ Every wireless technology has specific constraints in terms of bandwidth (byte/sec)
- How to deal with temporary offline operations?
- How to (re)configure the network?

### 3. Sending of measurements to a remote cloud infrastructure

SERVICES SPECIFICATION FOR THE PADOVA SMART CITY PROJECT

Service	Network type(s)	Traffic rate	Tolerable delay	Energy source	Feasibility
Structural health	802.15.4; WiFi and Ethernet	1 pkt every 10 min per device	30 min for data; 10 s for alarms	Mostly battery powered	1: easy to realize, but seismograph may be difficult to integrate
Waste management	WiFi; 3G and 4G	1 pkt every hour per device	30 min for data	Battery powered or energy harvesters	2: possible to realize, but requires smart garbage containers
Air quality monitoring	802.15.4; Bluetooth and WiFi	1 pkt every 30 min per device	5 min for data	Photovoltaic panels for each device	1: easy to realize, but greenhouse gas sensors may not be cost effective
Noise monitoring	802.15.4 and Ethernet	1 pkt every 10 min per device	5 min for data; 10 s for alarms	Battery powered or energy harvesters	2: the sound pattern detection scheme may be difficult to implement on constrained devices
Traffic congestion	802.15.4; Bluetooth and WiFi; Ethernet	1 pkt every 10 min per device	5 min for data	Battery powered or energy harvesters	3: requires the realization of both air quality and noise monitoring
City energy consumption	PLC and Ethernet	1 pkt every 10 min per device	5 min for data; tighter requirements for control	Mains powered	2: simple to realize, but requires authorization from energy operators
Smart parking	802.15.4 and Ethernet	On demand	1 min	Energy harvester	1: Smart parking systems are already available on the market and their integration should be simple
Smart lighting	802.15.4; WiFi and Ethernet	On demand	1 min	Mains powered	2: does not present major difficulties, but requires intervention on existing infrastructures
Automation and salubrity of public buildings	802.15.4; WiFi and Ethernet	1 pkt every 10 min for remote monitoring; 1 pkt every 30" for in-loco control	5 min for remote monitoring, few seconds for in-loco control	Mains powered and battery powered	2: does not present major difficulties, but requires intervention on existing infrastructures

Zanella, A., Bui, N., Castellani, A., Vangelista, L., Zorzi, M., "Internet of things for smart cities", IEEE Internet of Things Journal, Volume 1, Issue 1, 2014.

## Design approach

Full IoT stack includes:

- 1 Monitoring of the physical environment
- 2 Measurement of relevant parameters
- 3 Sending of measurements to a remote cloud infrastructure
- 4 **Collection of data in the data warehouse**
- 5 Data processing to extract meaningful/useful information
- 6 Use of information to *take decisions*, either manually or automatically

## 4. Collection of data in the data warehouse

- What kind of data?  
⇒ Temporal data, geolocalization, personal data, etc.
- How to arrange the data?  
⇒ Database schema, tables, etc.
- What is the amount of data to store?  
⇒ Storage solutions, technology, capacity, backups, etc.



## Design approach

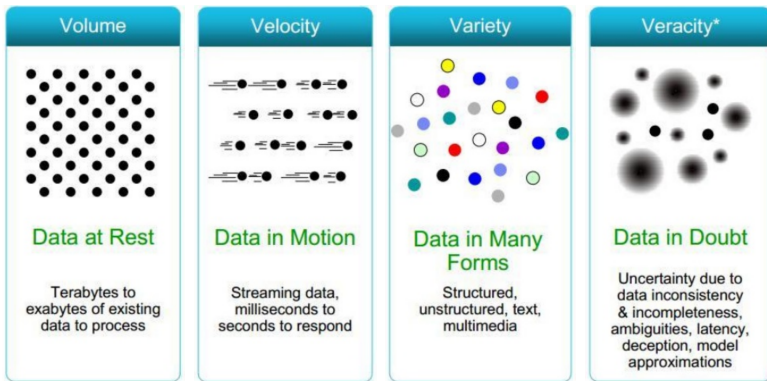
Full IoT stack includes:

- 1 Monitoring of the physical environment
- 2 Measurement of relevant parameters
- 3 Sending of measurements to a remote cloud infrastructure
- 4 Collection of data in the data warehouse
- 5 Data processing to extract meaningful/useful information
- 6 Use of information to *take decisions*, either manually or automatically

## 5. Data processing to extract meaningful/useful information

- What is the load of the computation?  
⇒ Evaluation on the scalability of the computation burden: what happens if more data need to be processed?
- What is the frequency to update the results?  
⇒ Seconds, milliseconds, no fixed frequency – generate alerts, etc.
- How could useful results be extracted?  
⇒ Statistics, machine learning, artificial intelligence, other algorithms.

## 5. Data processing to extract meaningful/useful information



Muralidhar Somisetty, "Data Analytics for IoT – specifically Industrial IoT"  
<https://www.slideshare.net/muralidhar9s/data-analytics-for-iot>

## Design approach

Full IoT stack includes:

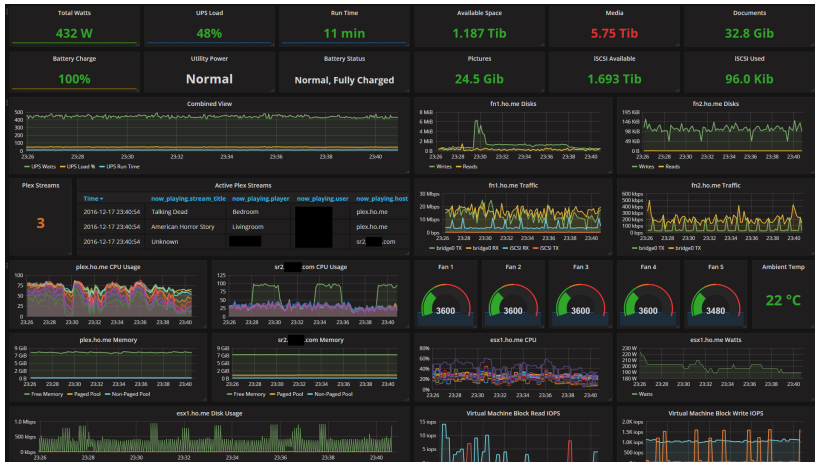
- 1 Monitoring of the physical environment
- 2 Measurement of relevant parameters
- 3 Sending of measurements to a remote cloud infrastructure
- 4 Collection of data in the data warehouse
- 5 Data processing to extract meaningful/useful information
- 6 Use of information to *take decisions*, either manually or automatically

## 6. Use of information to *take decisions*

Either manually or automatically...

- Graphical dashboards to present the results to a human operator/manager  
⇒ UI/UX design and guidelines
- Artificial intelligence and machine learning techniques to infer behaviors from large amount of data
- Use the results as a feedback for automatic control  
⇒ Close the loop on the physical environment using actuators

## 6. Use of information to *take decisions*



Source: Grafana The open platform for beautiful analytics and monitoring

# Embedded systems

An **embedded system** is a **computing system** – a combination of a computer processor, computer memory, and input/output peripheral devices – that has a **dedicated function** within a larger mechanical or electronic system.

## Embedded systems and IoT

In the context of IoT, **the node** that allows to **interact with the environment** is an embedded system

### Components of an IoT embedded system

#### Hardware

- Electronic board with case
- Processor
- Memory
- Peripherals (e.g., sensors)
- Communication devices (radio/transceiver)

#### Software

- Operating system (optional)
- Device drivers
- Network stack
- Application program



## Microprocessor vs. microcontroller

Both are electronic processors that are used to perform some computation.



Texas Instruments TMS-1802-NC  
Microcontroller Chip (1971)



National Semiconductor 4004 (1971)

However, they are used in different domains, and they have significant differences...

## Microprocessor vs. microcontroller

	Microprocessor	Microcontroller
Applications	General purpose computing	Embedded and control systems
Devices	Computers, smartphones, smart TV/watches, etc.	Embedded (remote controllers, washing machines, IoT, etc.)
Architecture type	Von Neumann (RAM shared by data and instructions)	Harvard (separated memory for data and instructions)
Architecture complexity	Complex (multicore, integrated GPUs, multiple cache levels, etc.)	Simple
Space to store programs	Large (on hard disks or SSD, up to several Terabytes)	Small (internal ROM, size hundreds of Kilobytes or less)
Clock speed	Fast (GHz)	Slow (tens/hundreds MHz)
Address space	Large (32/64 bit)	Limited (8/16 bit)
RAM	Large, external (4–128 Gb)	Small, internal (Kb)

## Microprocessor vs. microcontroller

	Microprocessor	Microcontroller
Power consumption	Tens of Watts (up to 150W)	Hundreds of mW
Power saving	Limited support (e.g., voltage scaling)	Usually the support is extended (several sleep modes)
Battery powered	No	Yes
Peripherals (memory, I/O, etc.)	External	Internal; extended support for communication interfaces such as CAN, I2C, SPI, etc.
Board	Large and expensive (due to external peripherals)	Compact (thanks to internal peripherals)
Cost	Up to hundreds of dollars per chip, without RAM	Few dollars per chip, including RAM
Operating System	Mandatory	Optional

# Common embedded boards for IoT applications

Brand	Models	CPU	RAM	+	Price	Type	Connectivity
Arduino	20+ and many clones (Spark, Intel, and so on)	ATmega, 8–64 MHz, Intel Curie, Linino	16 KB–64 MB	Largest community	~30 USD	RTOS, Linux, hobbyists	Pluggable extension boards (Wi-Fi, GPRS, BLE, Zigbee, and so on)
Raspberry Pi	A, A+, B, B+, 2, 3, Zero	ARMv6 or v7, 700 MHz–1.2 GHz	256–1 GB	Full Linux, GPU, large community	~5–35 USD	Linux, hobbyists	Ethernet, extension through USB, BLE (Pi3)
Intel	Edison	Intel Atom 500 MHz	1 GB	X86, full Linux	~50 USD	Linux, hobbyist to industrial	Wi-Fi, BLE
BeagleBoard	BeagleBone Black, X15, and so on	AM335x 1 GHz ARMv7	512 MB–2 GB	Stability, full Linux, SDK	~50 USD	Linux, hobbyist to industrial	Ethernet, extension through USB and shields
Texas Instruments	CC3200, SoC IoT, and so on	ARM 80 MHz, etc.	from 256 KB	Cost, Wi-Fi	<10 USD	RTOS, industrial	Wi-Fi, BLE, Zigbee
Marvell	88MC200, SoC IoT, and so on	ARM 200 MHz, etc.	from 256 KB	Cost, Wi-Fi, SDK	<10 USD	RTOS, industrial	Wi-Fi, BLE, Zigbee
Broadcom	WICED, and so on (also at the heart of the Raspberry Pis)	ARM 120 MHz, and so on	from 256 KB	Cost, Wi-Fi, SDK	<10 USD	RTOS, industrial	Wi-Fi, BLE, Zigbee, Thread

Source: Building the Web of Things book. [webofthings.io](http://webofthings.io)

# The Operating System (OS)

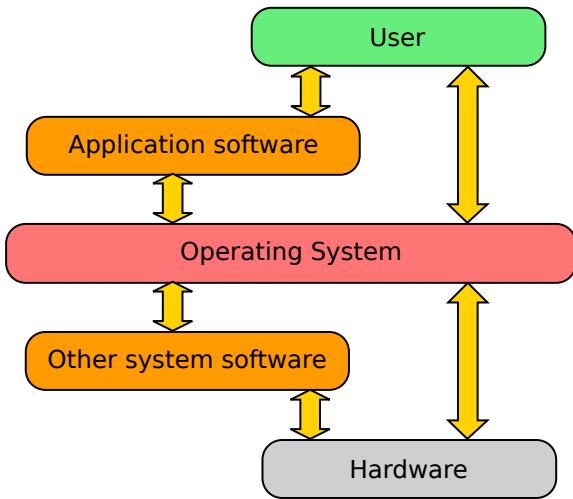
## Purpose of a traditional OS:

- Execute user programs and make solving user problems easier.
- Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.

## Services provided by the OS:

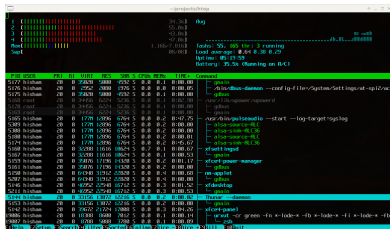
- (Graphical) User Interface (UI/GUI).
- Program execution.
- File-system manipulation.
- Communication (among processes, between machines).
- Error detection and management.

# The Operating System (OS)



# The Operating System (OS)

The main purpose of an Operating System (OS) is to manage the access of computer programs to **resources** such as CPU, memory, and peripherals (disks, files, etc.).



```
root@raspberrypi:~# top
top - 09:25:00 up 3 days, 10:00, user 1 load average: 0.04, 0.30, 0.13
Dn: 0.00s total, 0.00s running
Load average: 0.04, 0.30, 0.13
Uptime: 0:13:28
Battery: 75.3% (Charging on B-C)
```

PID	USER	PR	NI	VSZ	RSZ	SS	RES	%CPU	%MEM	TIME	COMMAND
1	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
2	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
3	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
4	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
5	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
6	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
7	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
8	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
9	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
10	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
11	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
12	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
13	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
14	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
15	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
16	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
17	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
18	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
19	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
20	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
21	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
22	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
23	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
24	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
25	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
26	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
27	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
28	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
29	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
30	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
31	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
32	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
33	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
34	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
35	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
36	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
37	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
38	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
39	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
40	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
41	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
42	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
43	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
44	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
45	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
46	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
47	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
48	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
49	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
50	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
51	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
52	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
53	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
54	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
55	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
56	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
57	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
58	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
59	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
60	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
61	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
62	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
63	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
64	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
65	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
66	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
67	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
68	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
69	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
70	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
71	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
72	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
73	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
74	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
75	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
76	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
77	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
78	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
79	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
80	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
81	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
82	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
83	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
84	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
85	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
86	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
87	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
88	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
89	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
90	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
91	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
92	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
93	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
94	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
95	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
96	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
97	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
98	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
99	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd
100	root	0	0	1024	0	0	0	0.0	0.0	0:00.00	systemd

In general, **several tasks** (or processes) require to access a limited amount of resources. The OS arbitrates the access to such limited resources.

For example, in uniprocessor systems, the available CPU must be shared among the different tasks.

# Embedded Operating System

An **embedded operating system** is simply an operating system  
designed for embedded systems

- The main characteristics are **resource efficiency** and **reliability**.
- They address the **very limited amount of hardware** like RAM, ROM, timer-counters and other on-chip peripherals.



## Characteristics of an embedded OS

- **Hardware support**, i.e., which hardware/boards are supported.
- **Programming** language – typically C language.
- **Scheduling**: e.g., single loop or multi-tasking.
- **Memory management**, which can be either present or not.

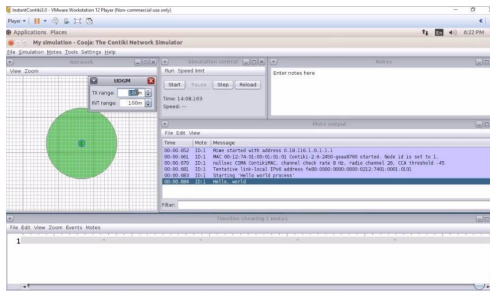
# Contiki-NG

GitHub repository:

`https://github.com/contiki-ng/contiki-ng`

- Invented in 2002
- Open-source
- Popular for **low-power microcontrollers**
- Support **Internet protocols** (IPv6 and IPv4) and wireless standard CoAP, 6lowpan, RPL
- Very suitable for **low-powered Internet connectivity**

# Contiki-NG



- Multitasking ability with a built-in Internet protocol suite
- Only 10kb of RAM and 30kb of ROM
- The core language written in C language
- The Cooja simulator can be used to test the software
- Both commercial and non-commercial applications

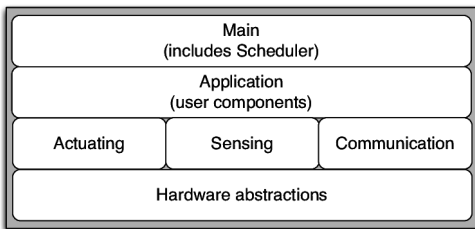
# TinyOS

Source code repository:

<https://github.com/saikatbsk/tinyOS>

- Component-based operating system
- Open-source
- The core language is nesC, a dialect of the C language
- Popular for memory optimization characteristics
- Components model abstractions of IoT systems, for example, sensing, packet communication, routing, etc.
- Developed by the TinyOS Alliance

# TinyOS



- Network protocols, sensor drivers, data acquisition tools are part of component libraries
- Largely used in wireless sensor networks
- Used in large-scale setups to simulate algorithms and protocols
- Used in the ESTCube-1 space program

## Example of program (hello-world.c for Contiki)

```
#include "contiki.h"

#include <stdio.h> /* For printf() */
/*-----*/
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
/*-----*/
PROCESS_THREAD(hello_world_process, ev, data)
{
    static struct etimer timer;

    PROCESS_BEGIN();

    /* Setup a periodic timer that expires after 10 seconds. */
    etimer_set(&timer, CLOCK_SECOND * 10);

    while(1) {
        printf("Hello, world\n");

        /* Wait for the periodic timer to expire and then restart the timer. */
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
        etimer_reset(&timer);
    }

    PROCESS_END();
}
```

## Example of program (hello-world.c for Contiki)

A minimal Contiki-NG example, simple printing out “Hello, world”:

```
make TARGET=native && sudo ./hello-world.native
```

This example runs a full IPv6 stack with 6LoWPAN and RPL.  
Look for the node’s global IPv6, e.g.:

```
[INFO: Native    ] Added global IPv6 address fd00::302:304:506:708
```

It is possible, for example to ping such a node:

```
$ ping6 fd00::302:304:506:708
PING fd00::302:304:506:708(fd00::302:304:506:708) 56 data bytes
64 bytes from fd00::302:304:506:708: icmp_seq=1 ttl=64 time=0.289 ms
```

# General-Purpose vs Real-Time Operating Systems

	GPOS	RTOS
Applications	Desktop computing, laptop, mobile devices	Embedded and industrial (cloud?)
Goal	Highest throughput (more task completed)	Guarantee of individual timing constraints
Determinism	No	Main feature
Latency	Best-effort	Predictable response time
Preemptive kernel	Optional	Yes
Examples	Windows, Linux, iOS, Android, Fuchsia, DOS, UNIX	<b>For microcontrollers:</b> Contiki, Erika Enterprise, Femto OS, FreeRTOS, NuttX, TinyOS, Zephyr; <b>Linux based:</b> RT-Linux, Xenomai, RTAI; <b>Proprietary:</b> Nucleus, QNX, VxWorks, Windows CE



## Real-time issues in the IoT domain

- **Miniaturized hardware**  $\implies$  real-time embedded systems
- **Low-power wireless**  $\implies$  real-time wireless communication
- **Cloud**  $\implies$  real-time data processing
- **Data analytics**  $\implies$  real-time analytics