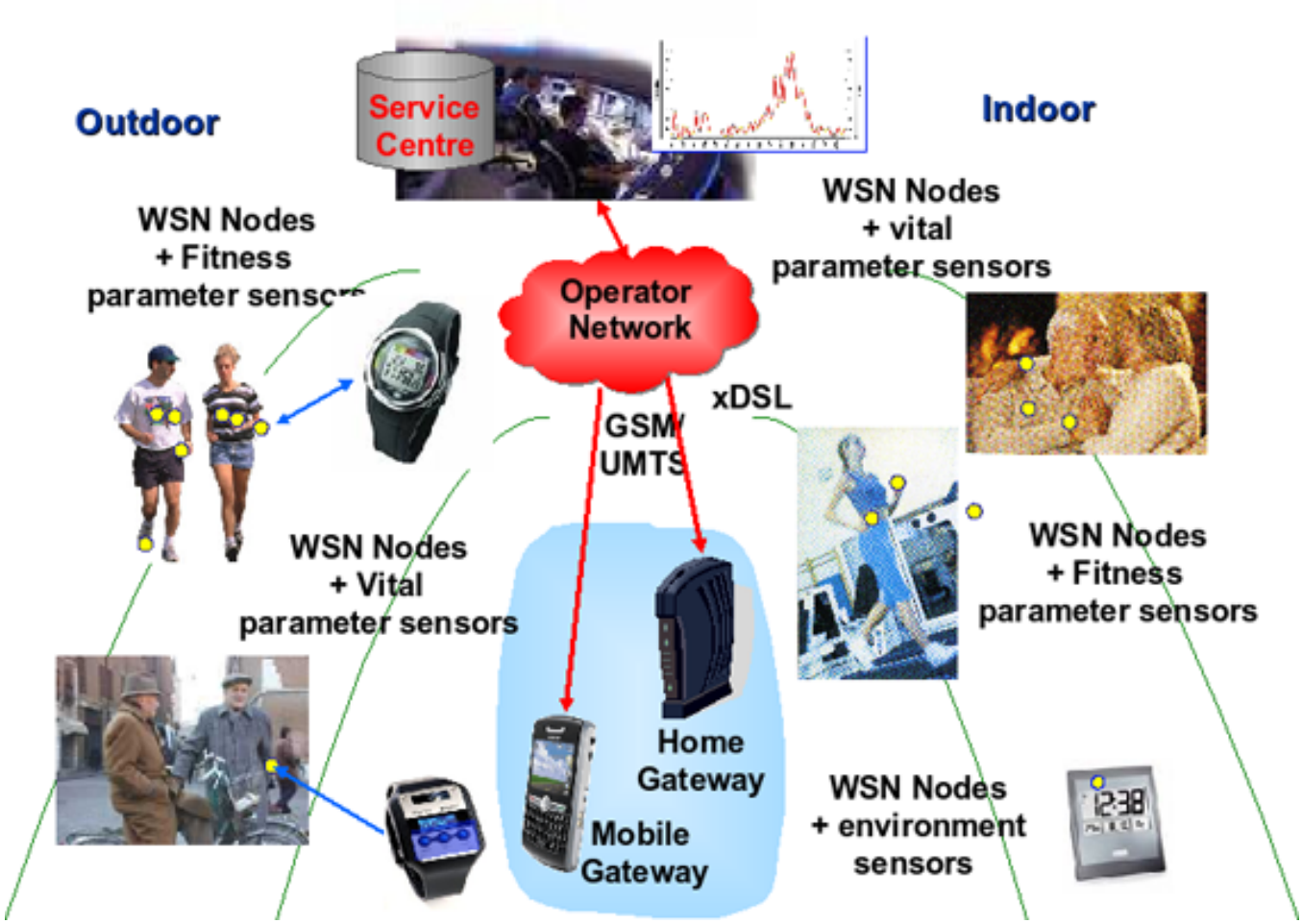
Lecture 6 Overview

- Last Lecture
  - Broadcast & Multicast Sockets
- This Lecture
  - Wireless Sensor Networks (WSNs)
  - Cyber-Physical Systems (CPSs)
  - Internet of Things (IoTs)
  - Source: lecture note
- Next Lecture
  - Routing & MAC protocol design in WSNs
  - Source: lecture note

A Big Picture

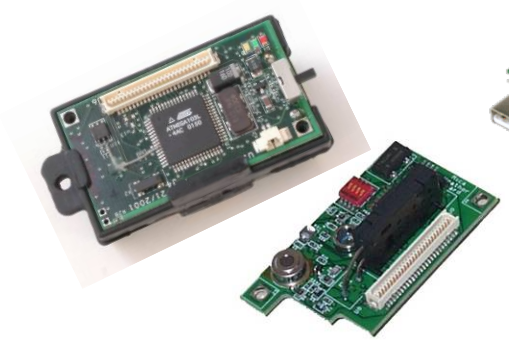


Ref:http://esd.sci.univr.it/images/wsn-example.png

What are sensor devices?

Rene 11/00

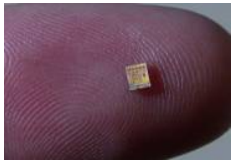
Mica 1/02

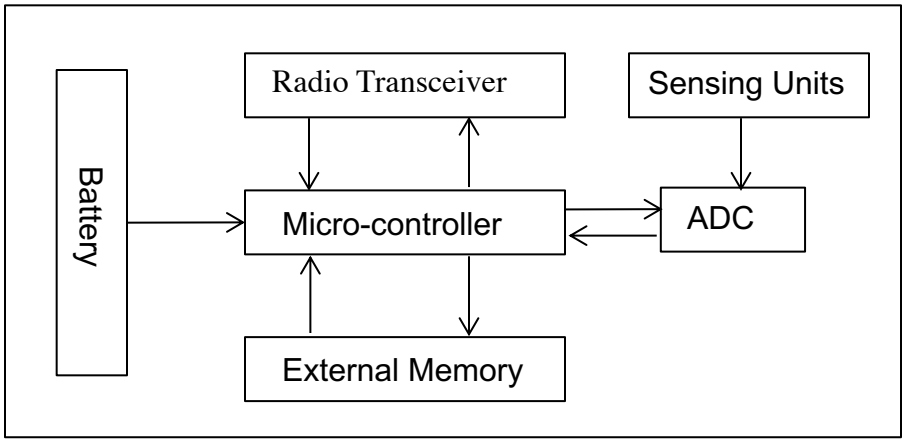


Telos 4/04

WeC 99 "Smart Rock"

Spec 6/03 "Mote on a chip"



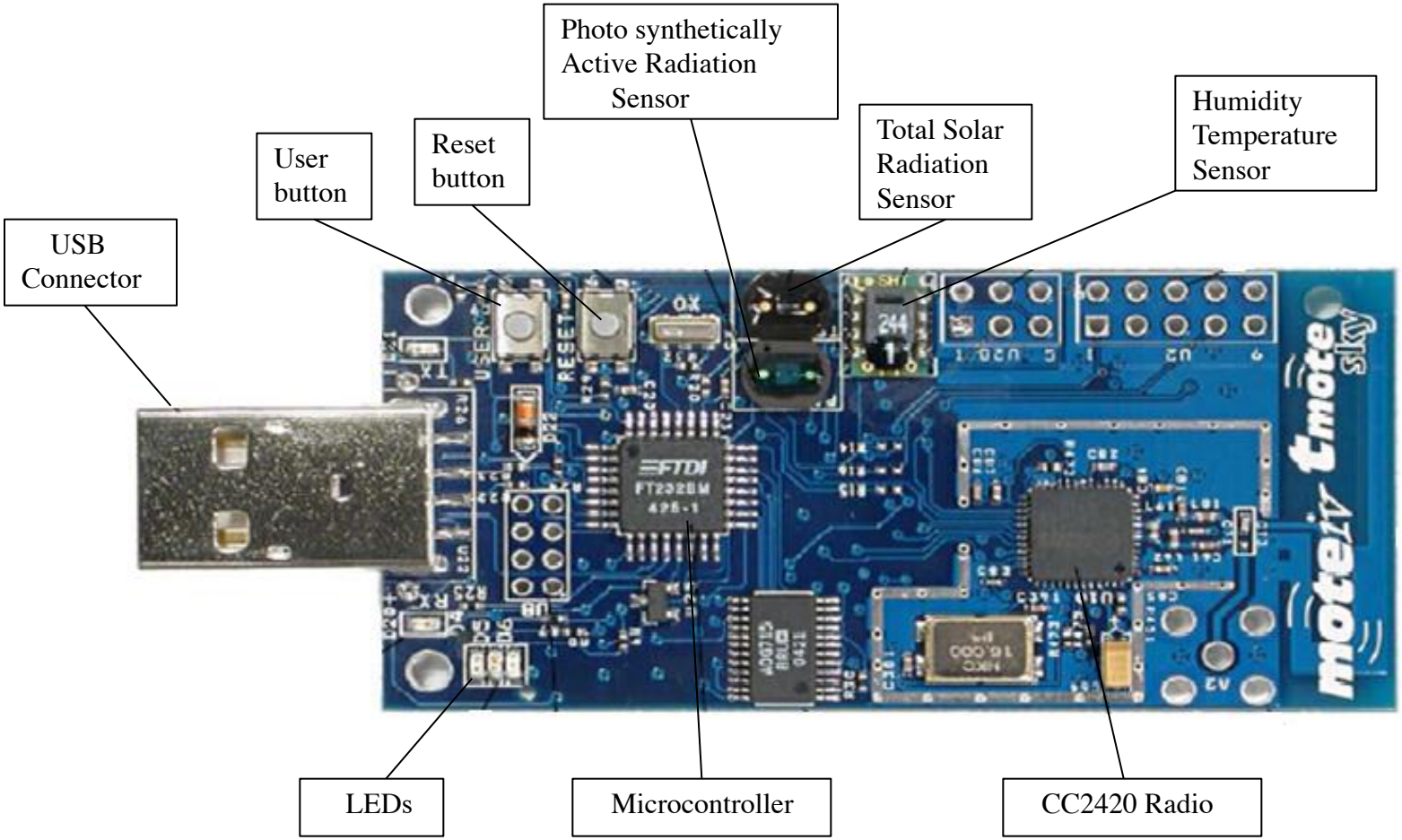


Sensor Platform Architecture

Sensing Unit:

- Temperature
- Humidity
- Light
- Pressure
- Image
- Accelerometer
- …

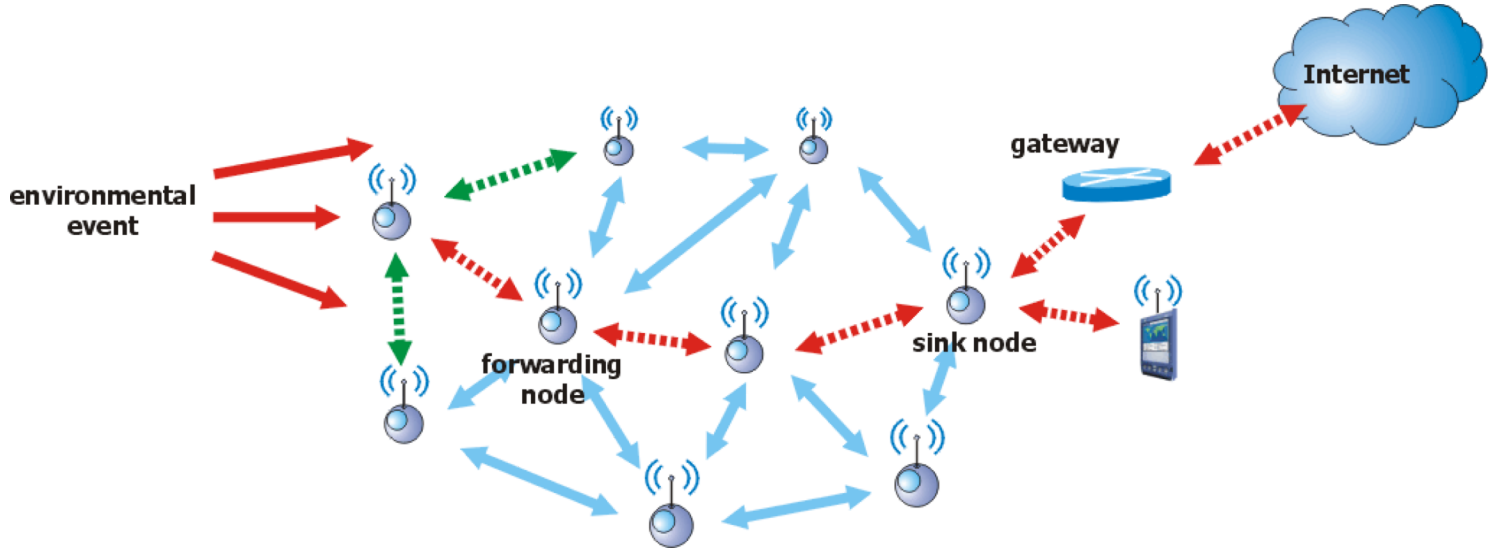
Close View of TelosB



Wireless Sensor Network

"Sensor networks are massive numbers of small, inexpensive devices pervasive throughout electrical and mechanical systems and ubiquitous throughout the environment that monitor and control most aspects of our physical world."

National Research Council



Applications (cont)

Applications (cont)

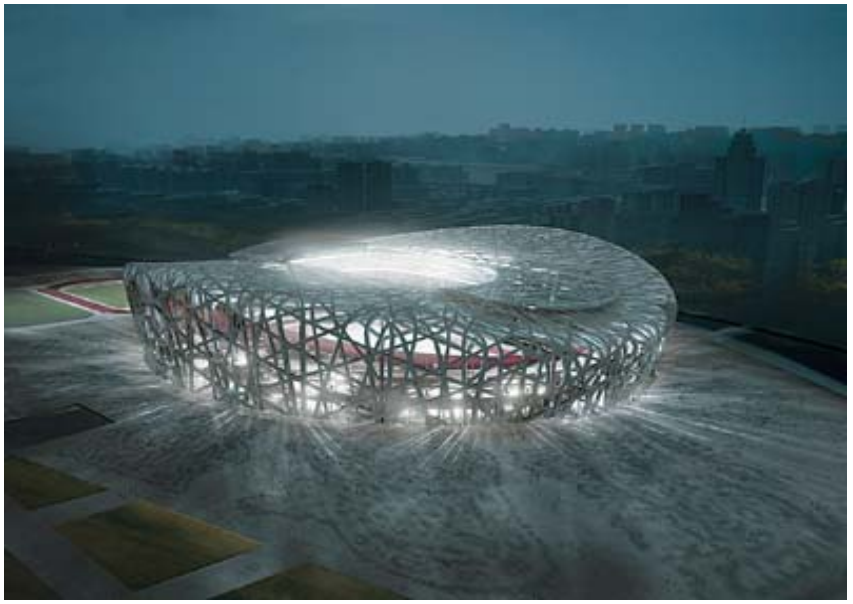
More than 100 sensors were installed to monitor the health of the bridge.

- **Weather stations** (2) (measures: wind speed and direction, ambient temperature and relative humidity)
- **Road temperature sensors** (4)
- **Concrete deck temperature sensors** (5)
- **Accelerometers** (42)
  - On shore (2)
  - On pylons (12)
  - On deck (15)
  - On stays (13)
- **Load cells on stays** (16)
- **Load cell on fuse** (4 digital+4 analog)
- **Joint displacement sensor** (on both expansion joints)
- **Water detection sensors** (4)
- **Strain gauges on gussets** (16)

Applications(cont.)

Bird's Nest stadium

Structural Health Monitoring of the 2008 Olympic Venues in Beijing
(*CGM Engineering*)

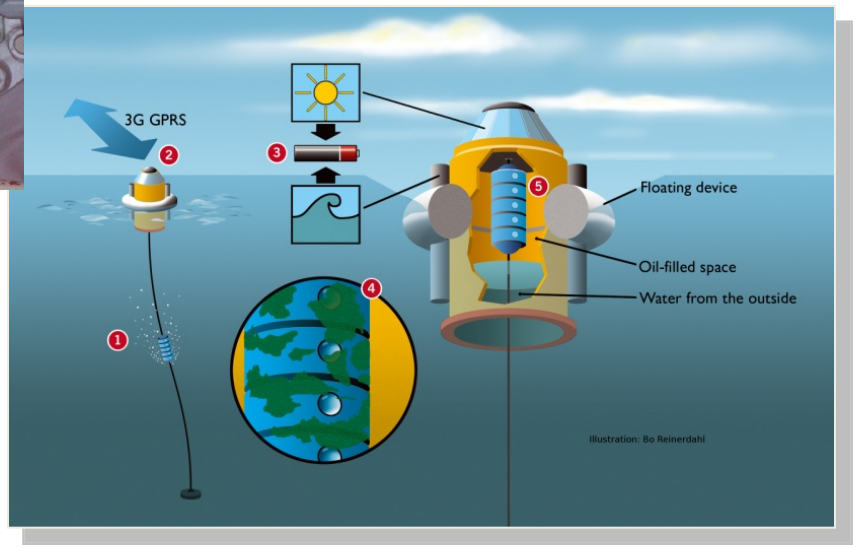


Forsyth Barr Stadium

Applications(cont.)

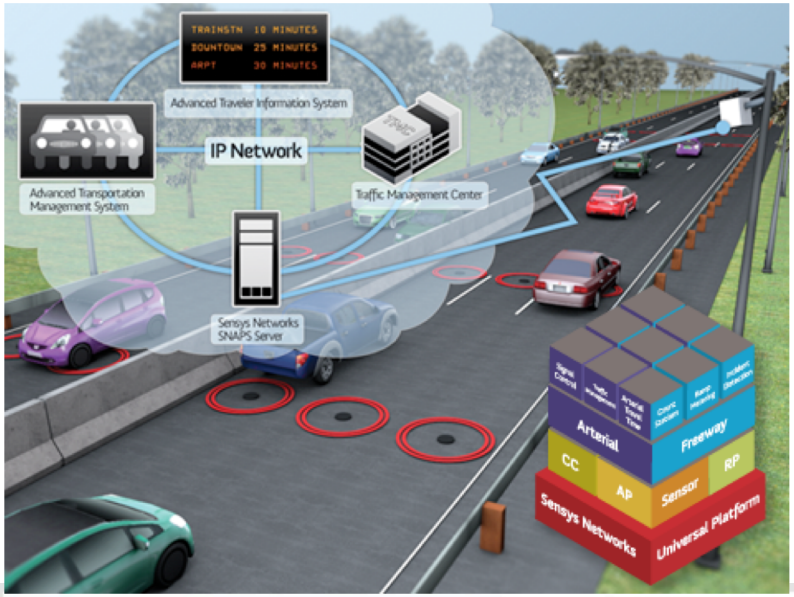
Industrial automation

Environmental monitoring



Any other potential applications?

Applications(cont.)

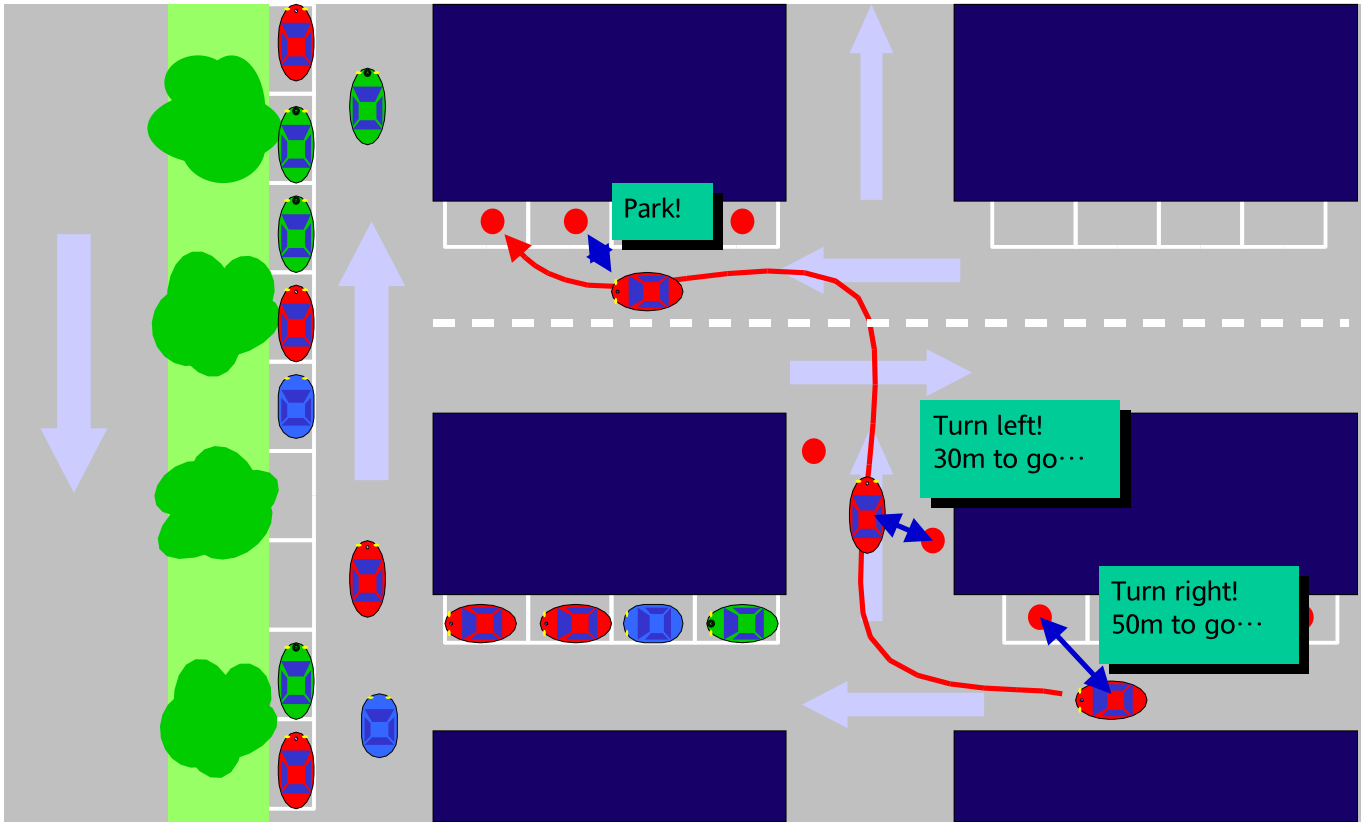


Traffic Control

& Inteligent Transport

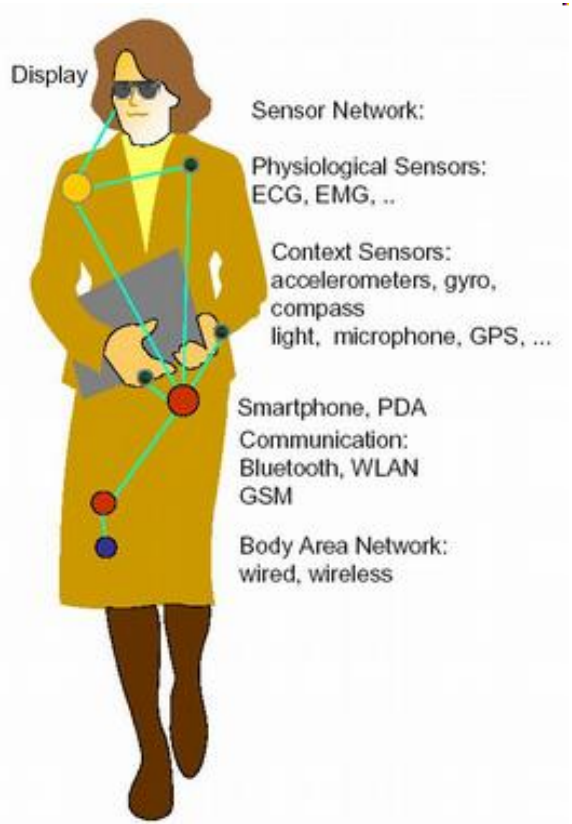
Applications(cont.)

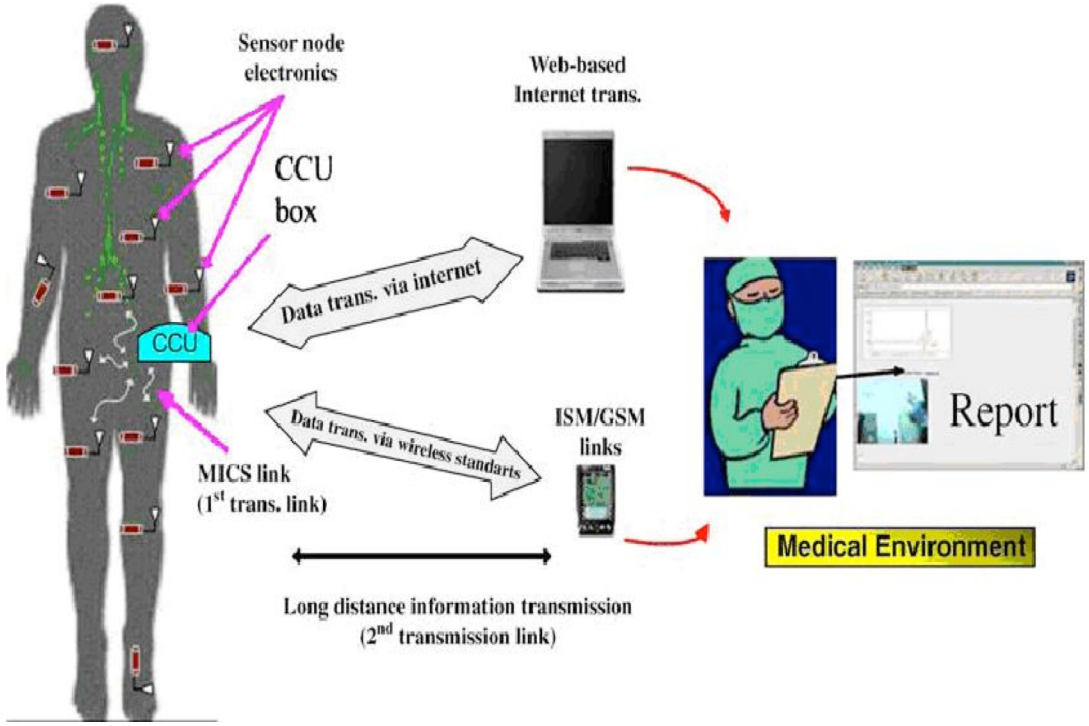
Smart Park



Applications(cont.)

E-health

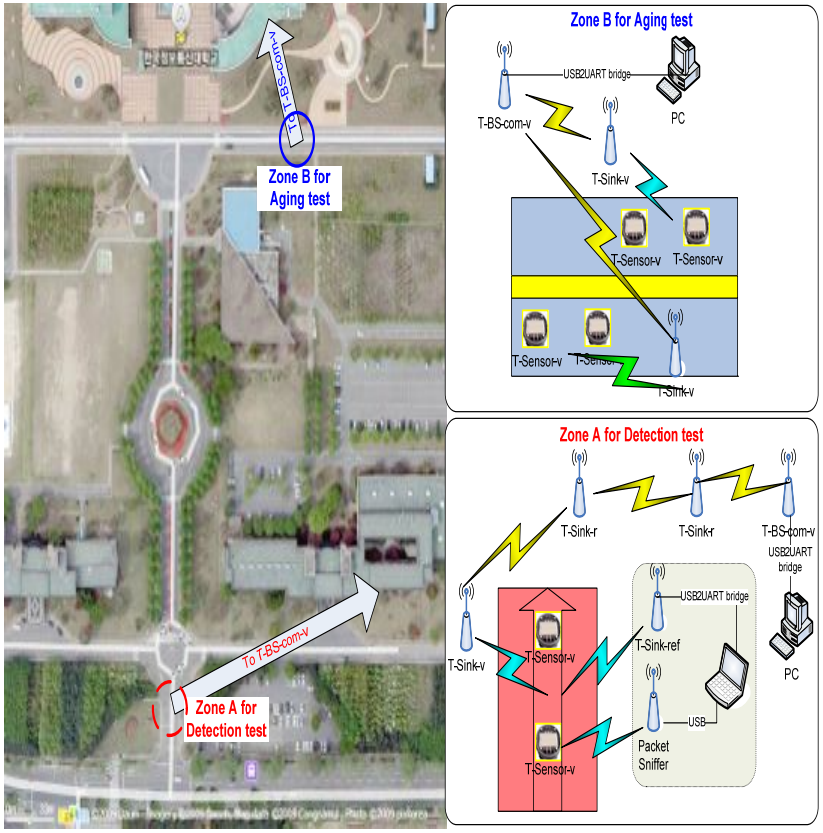




Applications(cont.)

Tracking

School Zone Safety System



Kids tracking in Kindergarten

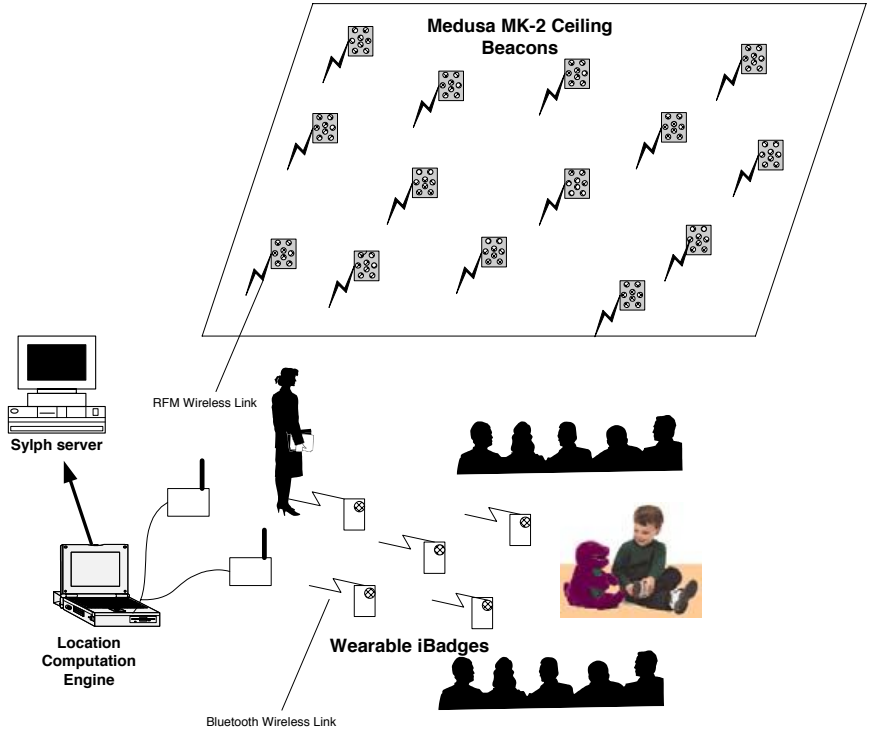


Figure 1.1. The Smart Kindergarten Localization Infrastructure

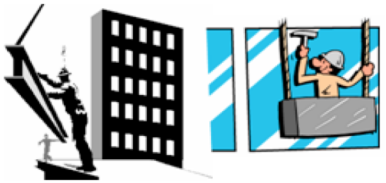
Applications(cont.)

Earthquake Early Warning

Controlling trains

Controlling factory lines
--> To mitigate damage

Controlling elevators
--> To prevent people from being trapped



Workers performing hazardous tasks
--> To secure safety

Suspending work in progress
--> To avoid mistakes

To prevent traffic accidents

At home
--> To enable personal protection

Alerting schools and assembly halls
--> To guide evacuation

Features & Challenges

- Tight resource constraints
  - Limited battery power
  - Limited computation capability
  - Limited memory
  - Limited bandwidth

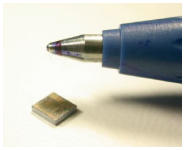
Mote Type Year	*WeC* 1998	*René* 1999	*René 2* 2000	*Dot* 2000	*Mica* 2001	*Mica2Dot* 2002	*Mica 2* 2002	*Telos* 2004
Microcontroller								
Type	AT90LS8535		ATmega163		ATmega128			TI MSP430
Program memory (KB)	8		16		128			60
RAM (KB)	0.5		1		4			2
Active Power (mW)	15		15		8		33	3
Sleep Power (μW)	45		45		75		75	6
Wakeup Time (μs)	1000		36		180		180	6
Nonvolatile storage								
Chip	24LC256				AT45DB041B			ST M24M01S
Connection type	I^2C				SPI			I^2C
Size (KB)	32				512			128
Communication								
Radio	TR1000				TR1000	CC1000		CC2420
Data rate (kbps)	10				40	38.4		250

Features & Challenges

- Tight resource constraints
  - Limited battery power
  - Limited computation capability
  - Limited memory
  - Limited bandwidth
- Dynamic network topology
  - Battery depletion
  - Node failure
  - Node mobility
  - Unreliable links
- Traffic pattern
  - Little activity in lengthy period
  - Intensive traffic in short time
  - Highly correlated traffic
  - End to end flows are required to be fair

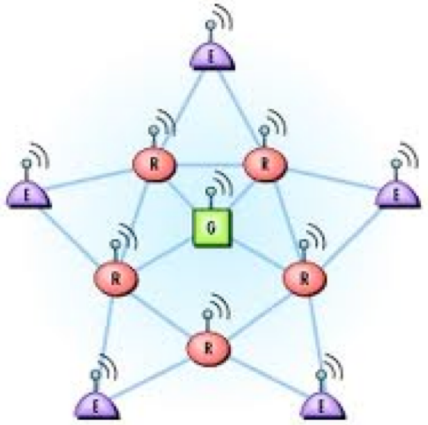
Design Issues and Challenges

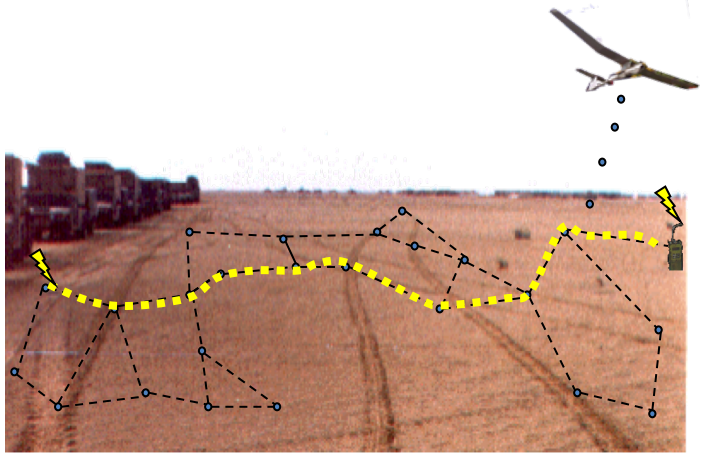
- Restricted resources
  - Battery power
  - Processing power
  - Memory
  - Bandwidth
- Portability & Customizability
  - Hardware evolvement
  - Different requirements
  - Reconfigurability



Design Issues and Challenges (cont)

- Multi-tasking
  - Sense data
  - Aggregate data
  - Encrypt/decrypt data
  - Routing data
- Network dynamics
  - Mobility
  - Failure of channel/nodes
- Distributed nature
  - Inter-node communication
  - Heterogeneity
  - Scalability





Design Characteristics

- Flexible architecture
  - Run-time reconfiguration
  - Small size of core kernel
- Efficient execution model
  - Accurate synchronization
  - Efficient task scheduling
- Clear application programming interface (API)
  - Networking API
  - Sensor data reading API
  - Memory manipulation API
  - Power management API
  - Task management API

Design Characteristics (cont)

- Reprogramming
  - Dynamic software update
  - Dynamic component linking
- Resource management
  - Dynamic memory allocation
  - Efficient task scheduling
  - Optimal sleep scheduling
- Scheduling
  - Real-time
  - Non real-time

IEEE 802.15.4 Physical Layer

- 26 different operational channels
  - Channel 0 is defined only in Europe, resides on the 868 MHz band
  - Channels 1 to 10, defined only in US, resides on the 902-982 MHz band, 2MHz channel spacing
  - Channels 11 to 26 are defined on the 2.4 GHz band, which makes them available everywhere. Channel spacing is 5MHz
- Radio modulation
  - Channels 0 to 10 use binary phase-shift keying (BPSK)
  - Channels 11 to 26 use quadrature phase-shift keying (QPSK)
- Radio channels in the 2.4GHz band share the frequency with 802.11(WiFi)

IEEE 802.15.4 Physical Layer

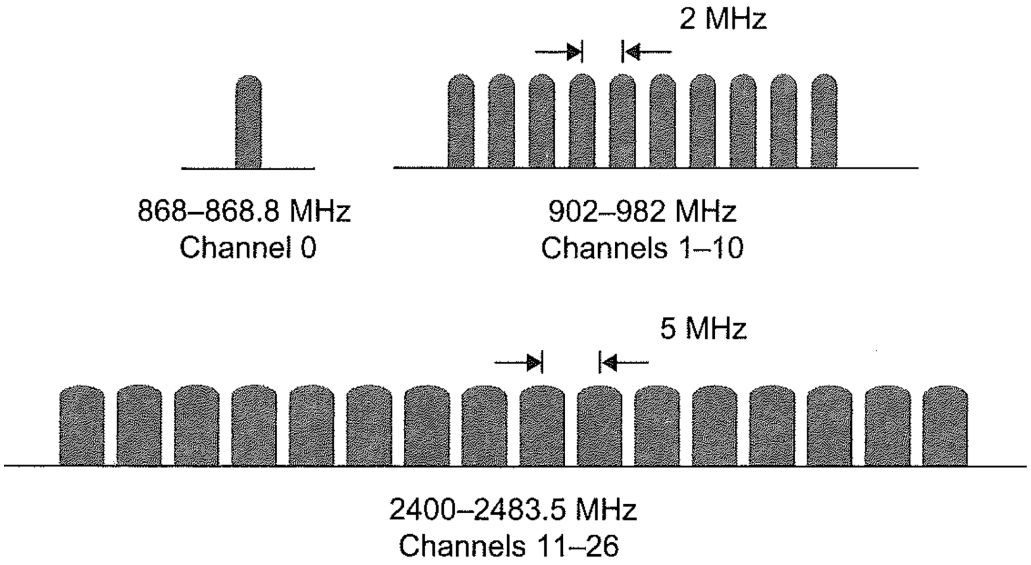
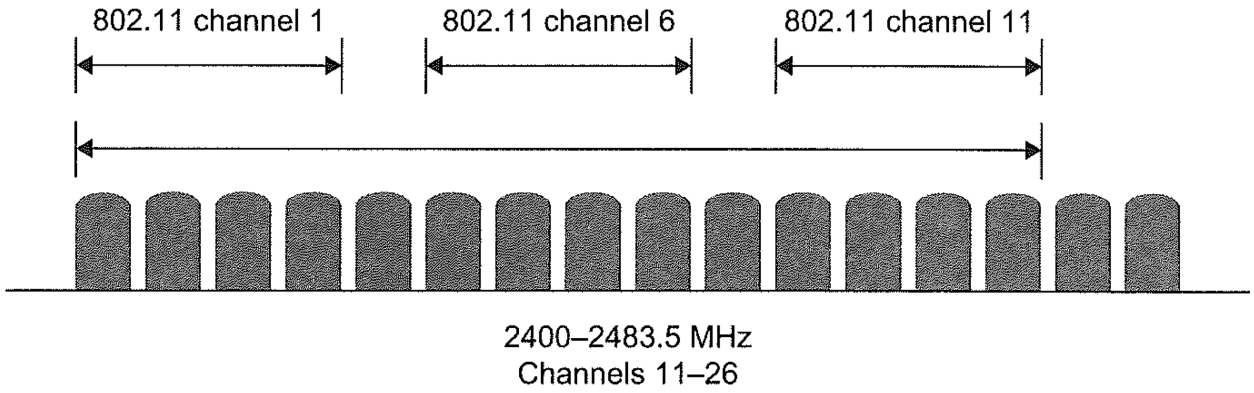


FIGURE 12.7

IEEE 802.15.4 defines 26 physical radio channels.

Research problem

- Mitigate the interference between WiFi and IEEE802.15.4



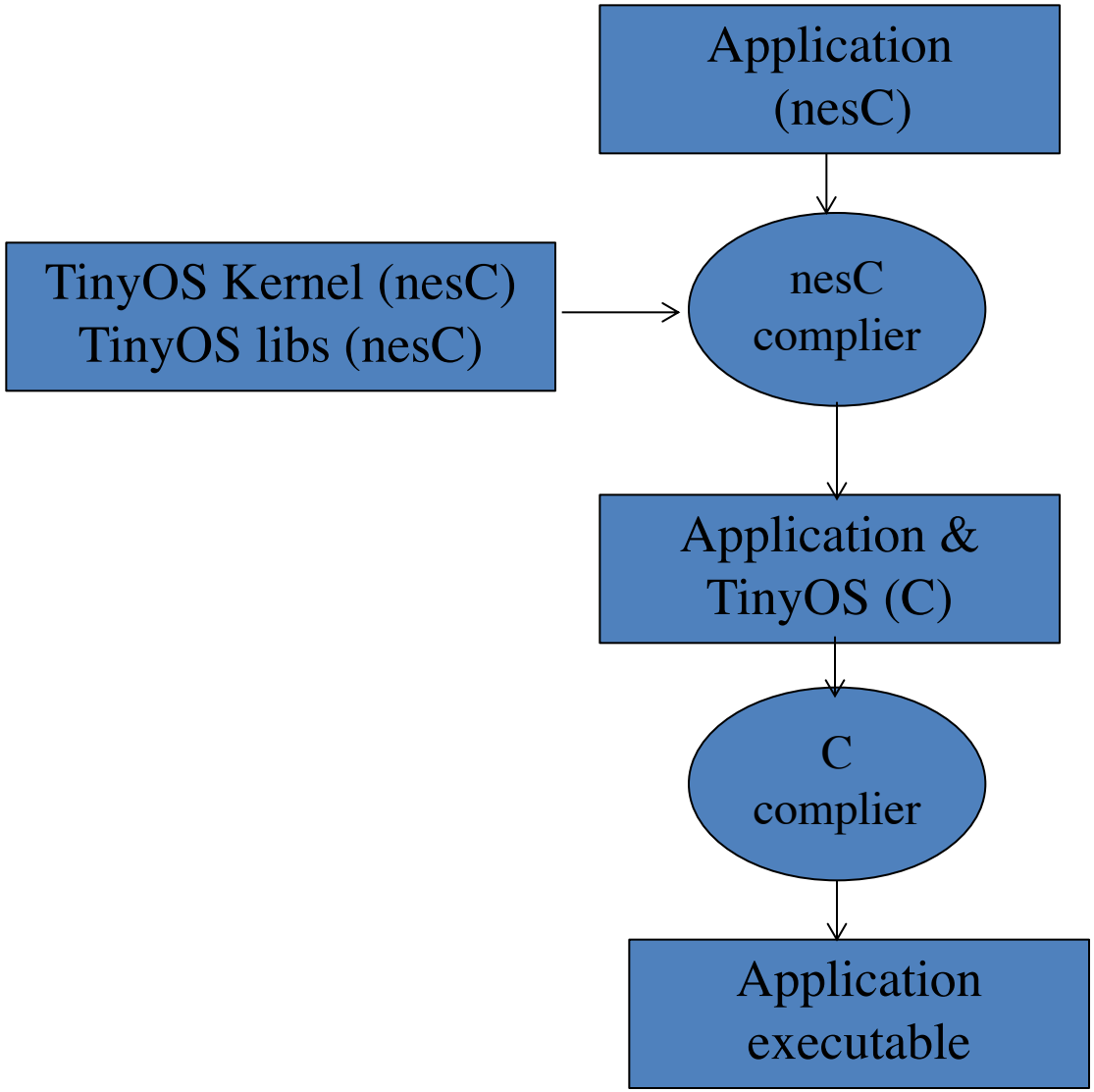
TinyOS

- Open source component-based operating system
- Written in the nesC programming language
- Started as a project at UC Berkeley
  - 1999: First TinyOS platform (Wec) and OS implementation.
  - 2000: Version 0.43 was made public via SourceForge.
  - 2002: Version 1.0 was implemented in nesC and released.
  - 2003: Version 1.1 includes data race detection.
  - 2006: Version 2.0 was released.
  - 2010: Version 2.1.1 was released.
  - 2012: Version 2.1.1 was released.
- Involve thousands of academic and commercial developers and users worldwide
  - ~35,000 downloads/year

nesC- The TinyOS Language

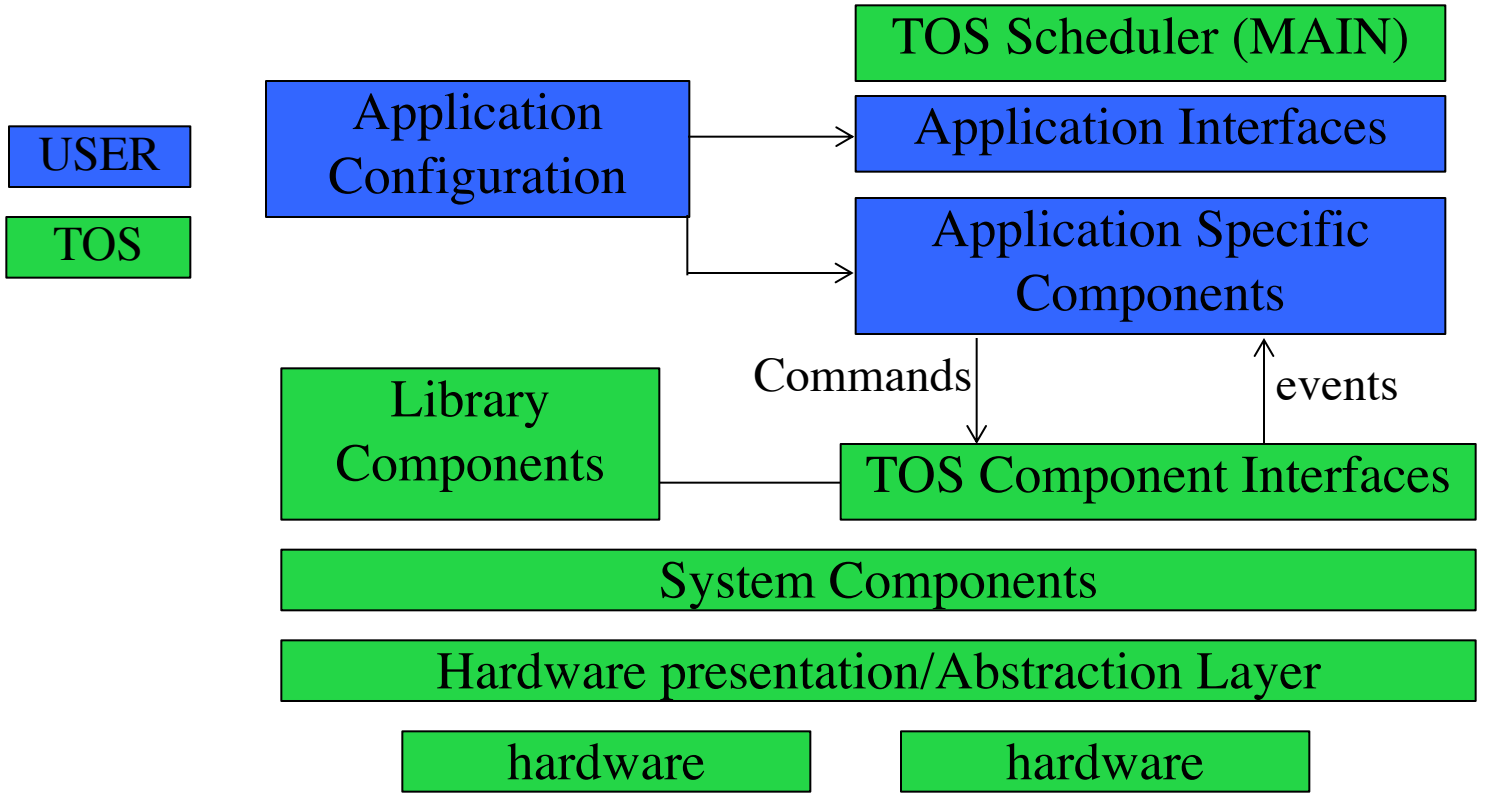
- A Dialect of C language
- Basic concepts:
  - Separation of construction and composition: components are assembled to form whole problems.
  - Specification of component behaviour in terms of set of interfaces.
  - Interfaces are bidirectional: commands and events
  - Components are statically linked to each other via their interfaces.
  - nesC is designed under the expectation that code will be generated by whole-program compilers.
    - nesC output is a c program file that is compiled and linked using GNU gcc tools.

TinyOS Program Compiling



TinyOS Architecture

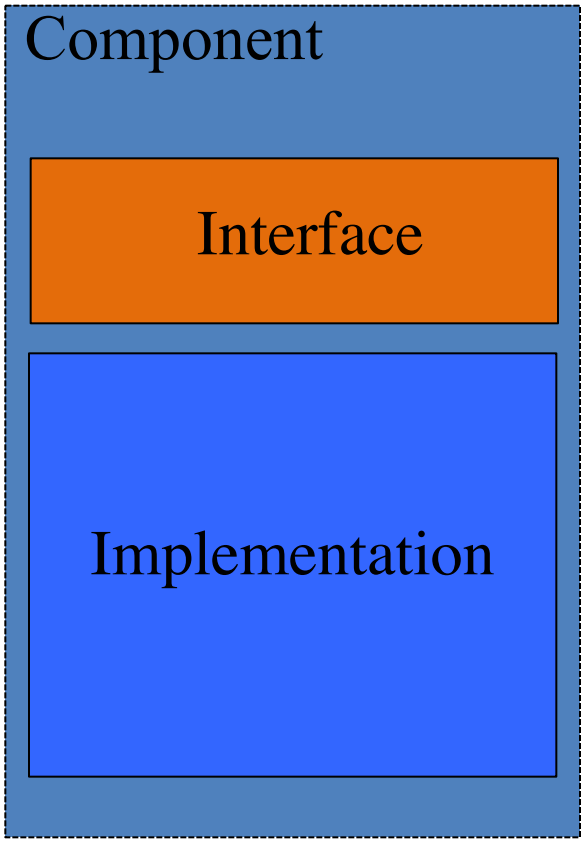
- Monolithic architecture
  - Component model at development and compile stages
  - Single static image at run time



TinyOS Basic Constructs (1)

- Component-based design
- A component consists of
  - Interfaces
    - The services it provides
    - The services it uses
  - Implementation
    - Defines internal working of a component
- Example

```
module  TimerM {
          provides {
              interface StdControl;
              interface Timer [uint8_t id];
          }
           uses interface Clock;
}
Implementation {
  …
}
```

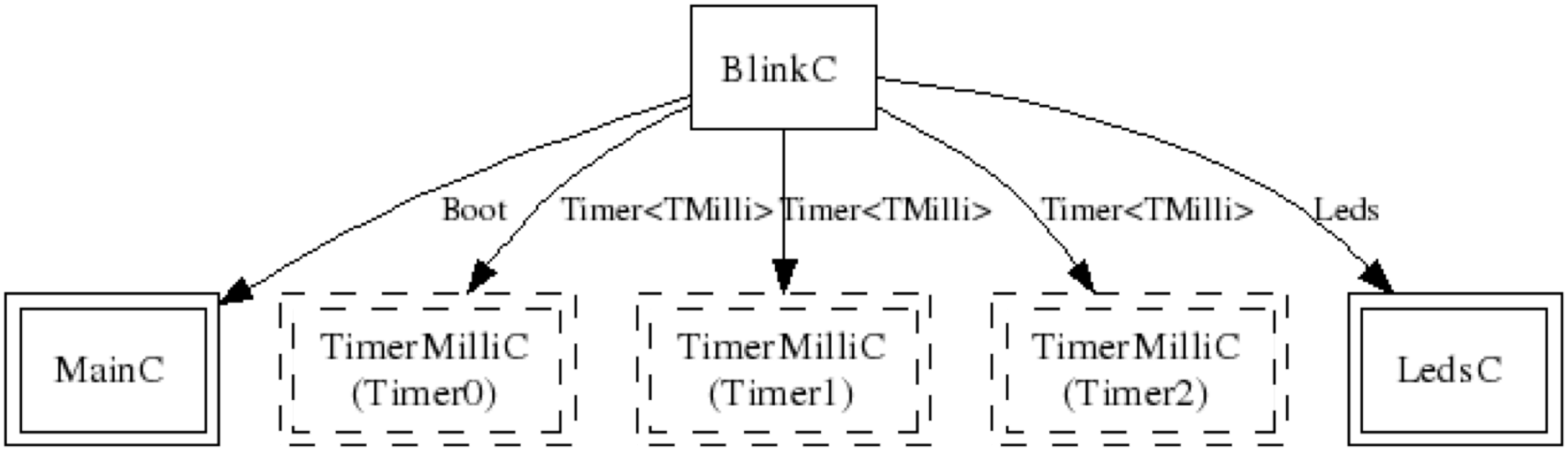


TinyOS Basic Constructs

- Components have three computational abstractions:
  - Commands
    - Requests to the component to perform some service.
    - e.g. to trigger a timer
  - Events
    - Signal the completion of services
    - e.g. hardware interrupts
  - Tasks
    - Intra-component concurrency
    - Intensive work done at low-priority

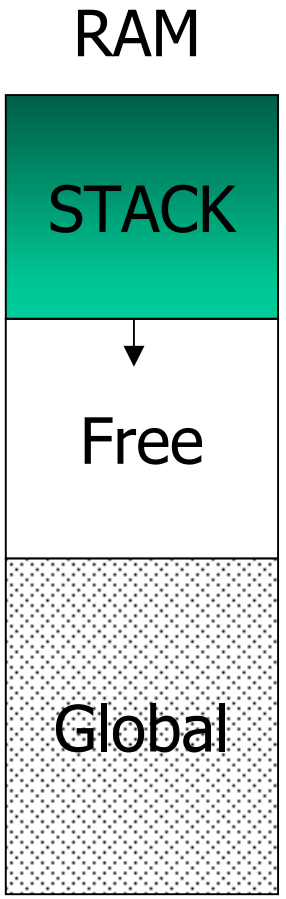
TinyOS Component Type

- Modules
  - provide code for function implementation
- Configurations
  - wire components together, connecting interfaces between each other



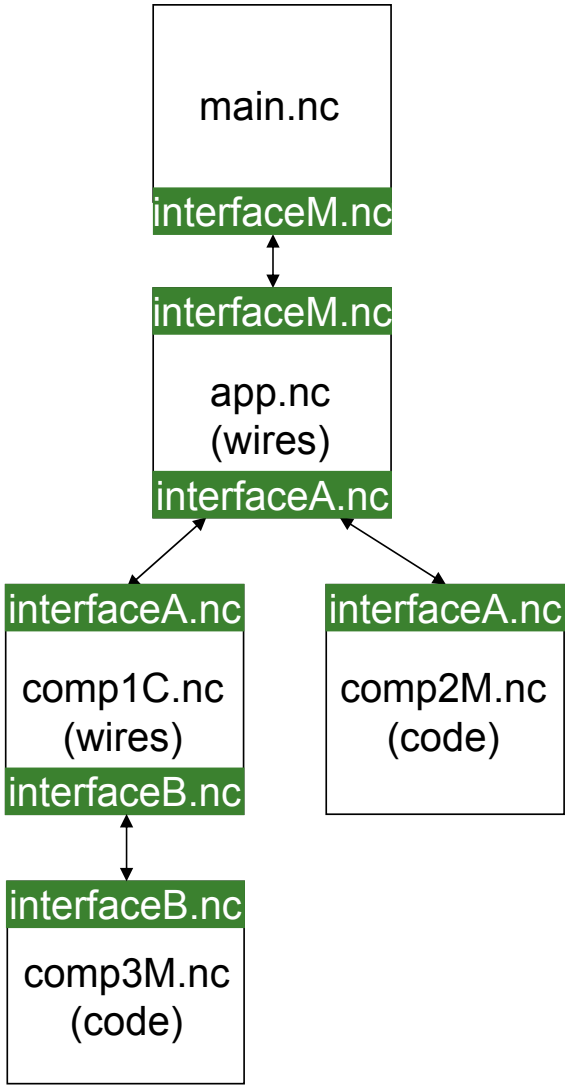
Memory Model

- Static
  - No heap
    - No dynamic run-time memory allocation
  - No function pointers
  - Components are statically linked
    - Size required determined at compile time
- Global variables
  - Conserve memory
  - Frame per component
- Local variables
  - Save on the stack



TinyOS File Types

- Interfaces
  - Specifies functionality to outside
  - What commands can be called
  - What events need handling
- Module
  - Code implementation
  - Code for interface functions
- Configuration
  - Wiring of components



BlinkTask – A simple TinyOS Application

- Blink an LED at a Periodic Rate
- Build by "wiring" together components
  - A timer component to provide periodic EVENTs
  - A LED component to control an LED
  - On Timer Event
  - Post a task to turn ON or OFF LED
  - Study Objective:
  - Understand the structure of a TinyOS program
  - Understand Commands, Events and Tasks in action

BlinkTask Files

- In the apps/tutorials/BlinkTask directory
  - BlinkTaskAppC.nc The Configruation
  - BlinkTaskC.nc The Implementation
  - Makefile Build information

```
COMPONENT = BlinkTaskAppC
include  $(MAKERULES)
```

- Understanding BlinkTask
  - Identify the TOS Component used
  - Determine the Interfaces the BlinkTask Component must handle
  - The wiring of the components

BlinkTaskAppC.nc

```
configuration BlinkTaskAppC {}
implementation {
  component MainC, BlinkTaskC, LedsC;
  component new TimerMilli() as Timer0;
  BlinkTaskC -> MainC.Boot;
  BlinkTaskC.Timer0 -> Timer0;
  BlinkTaskC.Leds -> LedsC;
}
```

Components used in this application

MainC is the TOS kernel

This defines the "wiring" between components

BlinkTaskC.nc

Kernel call this at boot time

```
#include "Timer.h"
module BlinkTask{
  uses interface Timer<TMilli> as timer0;
  uses interface Leds;
  uses interface Boot;
}
implementation {
  bool state;
  task void toggle() {
    if (state) {
      call Leds.led0On();
    }
    else {
      call Leds.le
    }
  }
```

```
event void Boot.booted(){
    state = FALSE:
    call Timer0.startPeriodic(1000);
  }

  event void Timer0.fired() {
     state = ! state;
     post toggle();
  }
}
```

Handle the Timer signal – post a task

Non-time critical task decides how to set the leds

Contiki

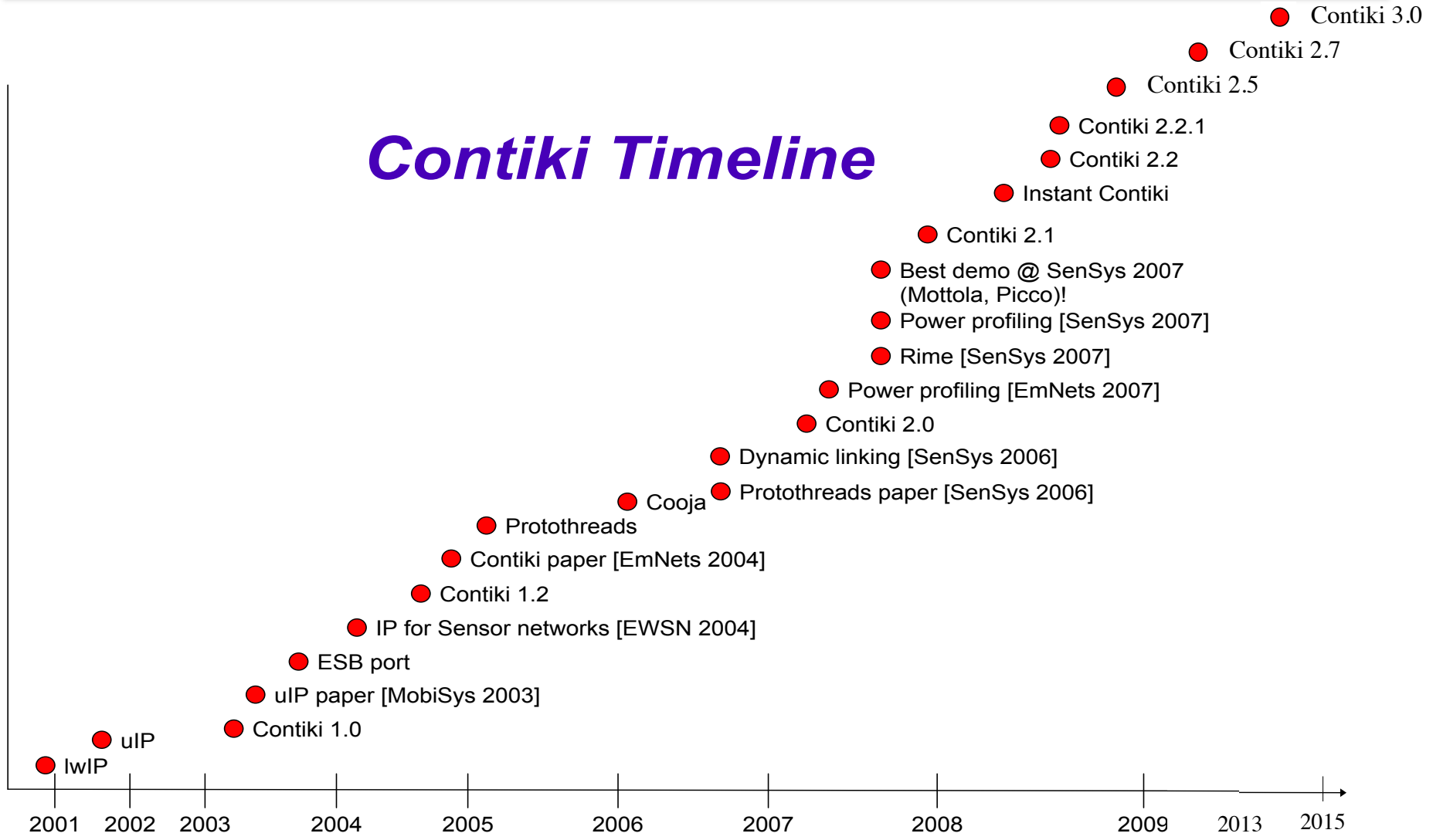
- What is Contiki and where does the name come from?

The Kon-Tiki raft: sailed across the Pacific Ocean with minimal resources
Used by a Norwegian explorer and writer Thor Heyerdahl in 1947

- Written in C programming language
- An open-source multitasking operating system
- The basic kernel and most core functions were developed by Adam Dunkels at Swedish Institute of Computer Science

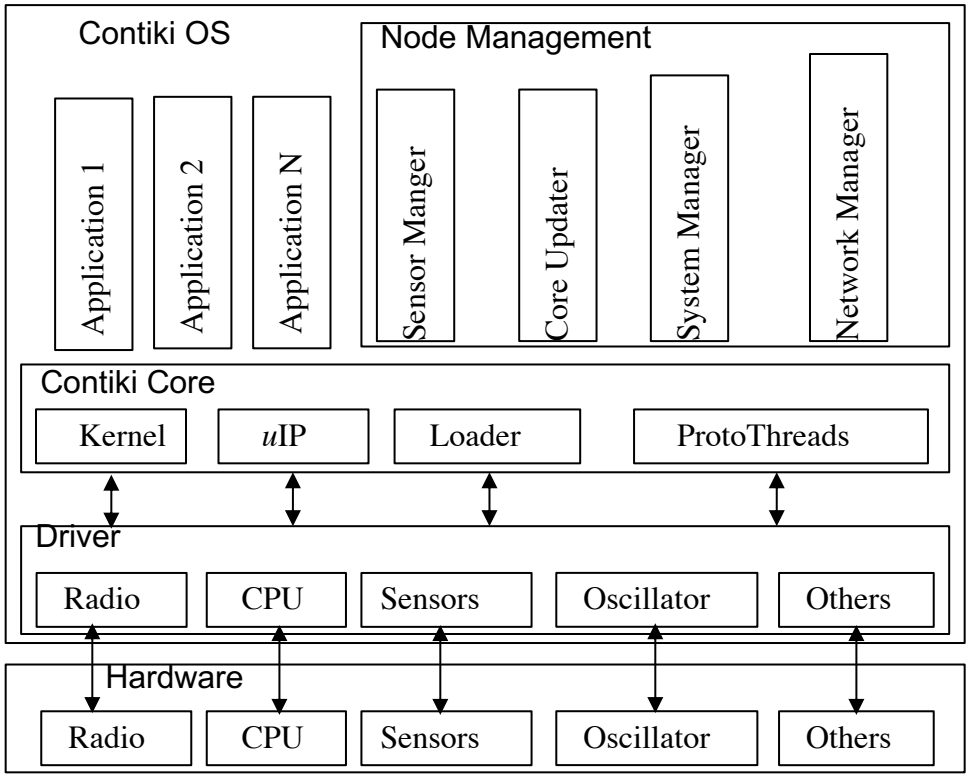
thingsquare

Contiki Development History



Contiki Architecture

- Modular architecture
- Event-driven kernel
- Other features
  - Multi-tasking
  - Protothreads
  - TCP/IP
  - IPV4/V6
  - Web browser
  - Dynamic program loading
  - Coffee file system
  - …



Memory Management

- Dynamic memory management
  - Managed Memory Allocator (MMA)
    - Dynamic allocate and deallocate memory
    - Free from fragmentation
  - Macro and functions
    - #define MMEM_PTR(m) // get a pointer to the managed memory
    - mmem_init (void)
    - memb_alloc (struct mmem *m, unsigned int size)
    - memb_free (struct mmem *m)

Timers in Contiki

- struct timer
  - Passive timer, only keeps track of its expiration time
- struct etimer
  - Active timer, sends an event when it expires
- struct ctimer
  - Active timer, calls a function when it expires
- struct rtimer
  - Real-time timer, calls a function at an exact time

Please read this essay to get more details on the timer library in Contiki.

https://github.com/contiki-os/contiki/wiki/Timers

Demo: BlinkTask Revisited

- Blink an LED at a Periodic Rate
- On Timer Event
  - turn ON or OFF LED
- Study Objective:
  - Understand the structure of a Contiki program
  - Understand how to use timers

```
#include "contiki.h"
#include "dev/leds.h"

#include <stdio.h> /* For printf() */
/*---------------------------------------------------------------------------*/
/* We declare the process */
PROCESS(blink_process, "LED blink process");

/* We require the processes to be started automatically */
AUTOSTART_PROCESSES(&blink_process);
/*---------------------------------------------------------------------------*/

/* Implementation of the second process */
PROCESS_THREAD(blink_process, ev, data)
{
  static struct etimer timer;
  static uint8_t leds_state = 0;
  PROCESS_BEGIN();
  while (1)
  {
   // we set the timer from here every time
    etimer_set(&timer, CLOCK_CONF_SECOND / 4);

  // and wait until the vent we receive is the one we're waiting for
    PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_TIMER);

  // update the LEDs
    leds_off(0xFF);
    leds_on(leds_state);
    leds_state += 1;
  }
  PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

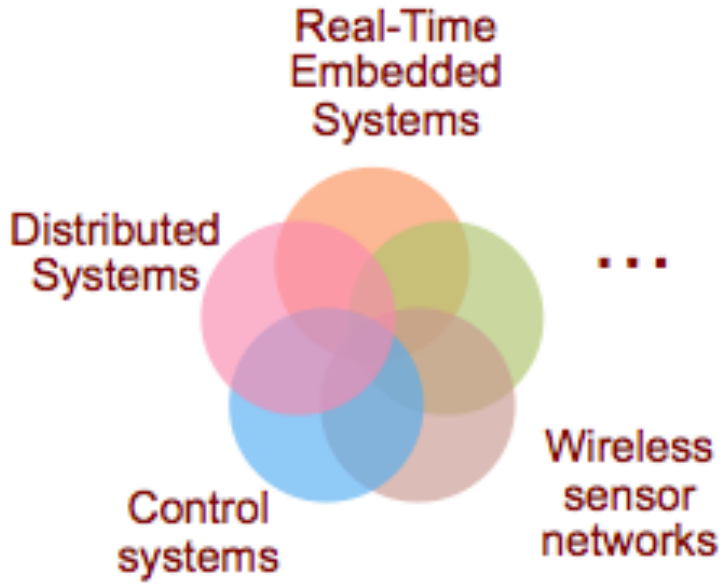
Cyber-Physical System (CPS)

- Cyber – computation, communication, and control that are discrete, logical, and switched
- Physical – natural and human-made systems governed by the laws of physics and operating in continuous time
- Cyber-Physical Systems – systems in which the cyber and physical systems are tightly integrated at all scales and levels

"CPS will transform how we interact with the physical world just like the Internet transformed how we interact with one another."

Cyber-Physical System (CPS)

- *Cyber-physical systems (CPSs)* are physical and engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communication core.

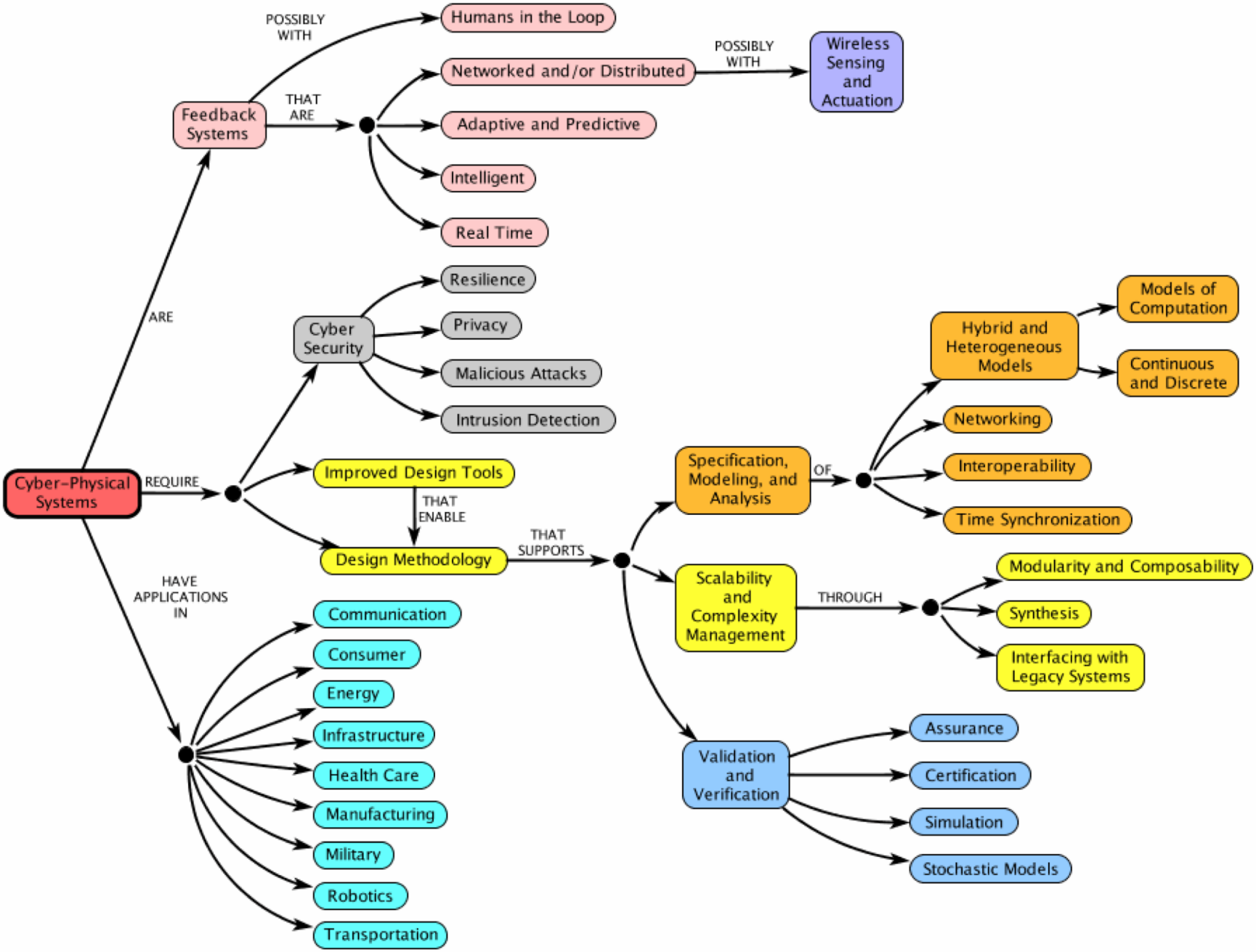


Sensing + Communication + Computation + Control

Cyber-Physical Systems – a Concept Map

See authors and contributors.

http://CyberPhysicalSystems.org



CPS Applications

Healthcare

High mental workload increases the likelihood of medical errors

The medical errors may cause severe complications.

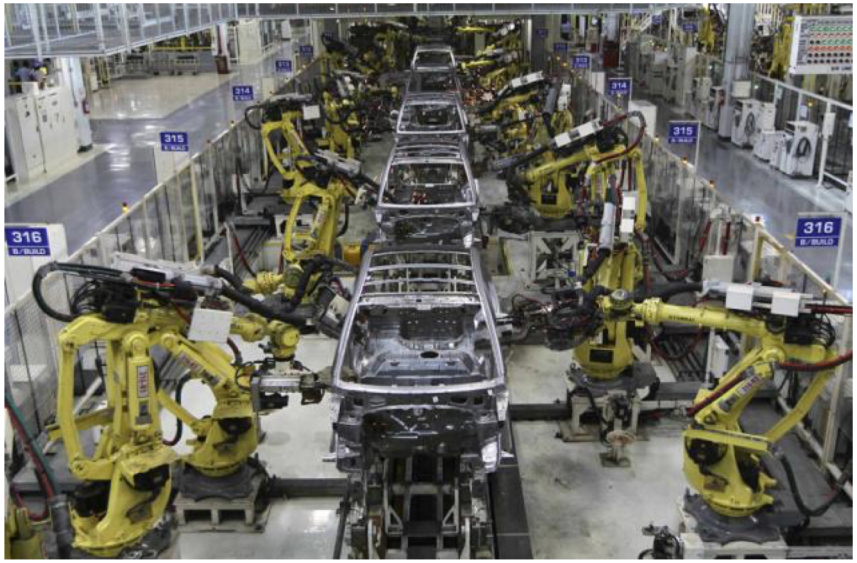
The degradation of patient conditions further increases the cyber-physical complexity

http://publish.illinois.edu/mdpnp-architecture/complexity-reduction/

http://www.nsf.gov/news/special_reports/cyber-physical/

CPS Applications

Manufacturing



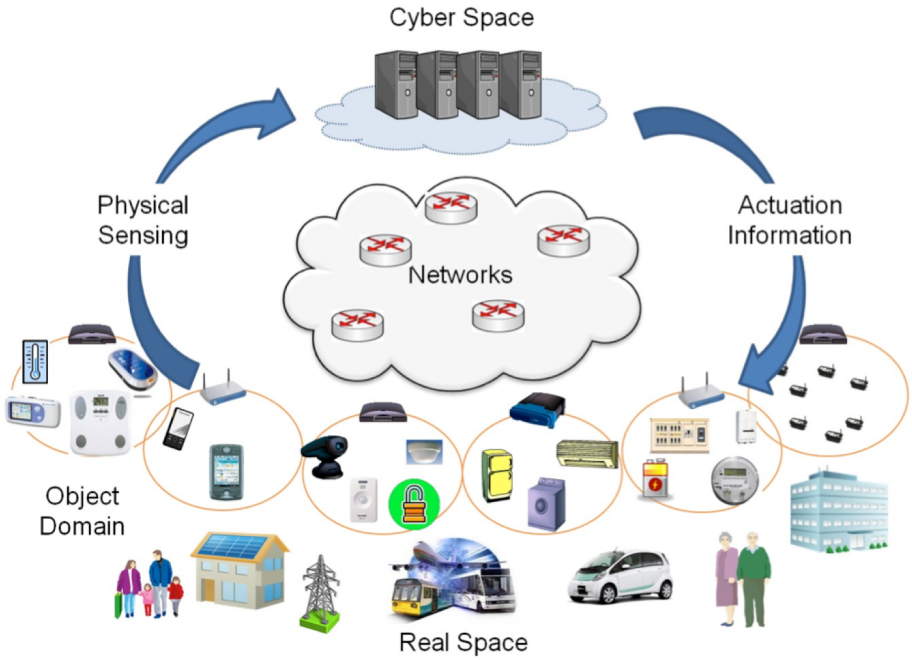
Applications(cont.)

Smart Building

1. sunblind
2. street light
3. power meter
4. Ethernet gateway
5. GSM gateway
6. mobile phone
7. Hi-Fi
8. PC
9. gate
10. door
11. light
12. router
13. heating
14. security system
15. thermometer
16. air-conditioning

CPS Applications

Transportation



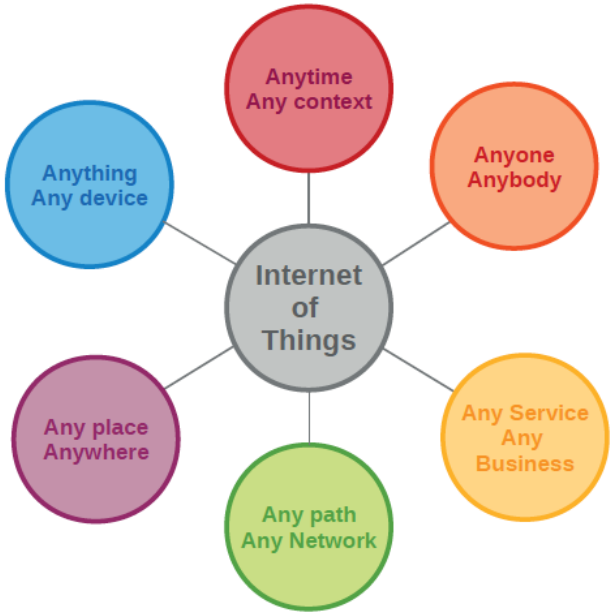
Internet of Things (IoTs)

What are Internet of Things?

"A network of items—each embedded with sensors—which are connected to the Internet. " -- IEEE

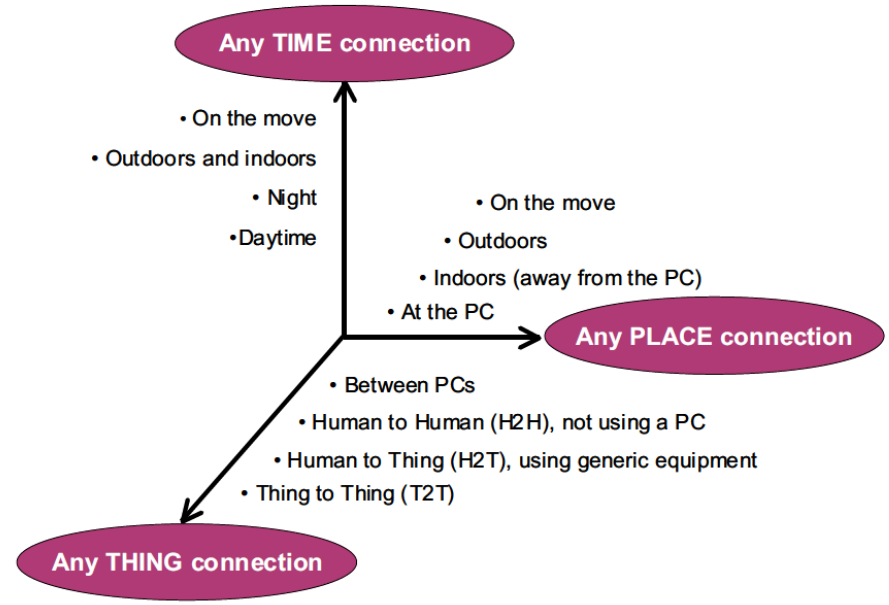
"The basic idea is that IoT will connect objects around us (electronic, electrical, non-electrical) to provide seamless communication and contextual services provided by them. Development of RFID tags, sensors, actuators, mobile phones make it possible to materialize IoT which interact and co-operate each other to make the service better and accessible anytime, from anywhere." -- IETF

Any-X Point of View



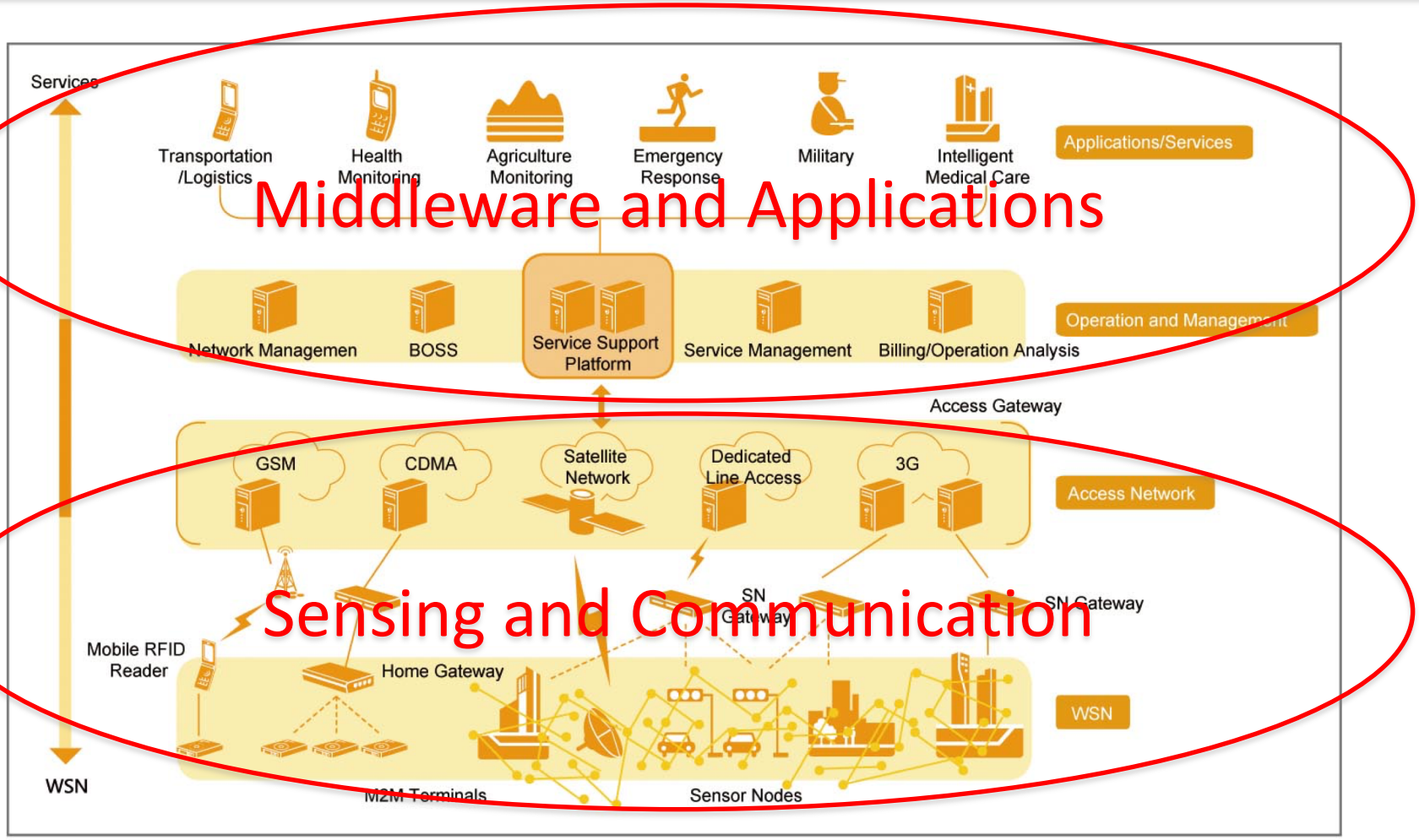
Source: Perera et al. 2014

Figure 1 – A new dimension



Source: ITU adapted from Nomura Research Institute

IoT Layered Architecture



Source: ZTE

Enabling Technologies

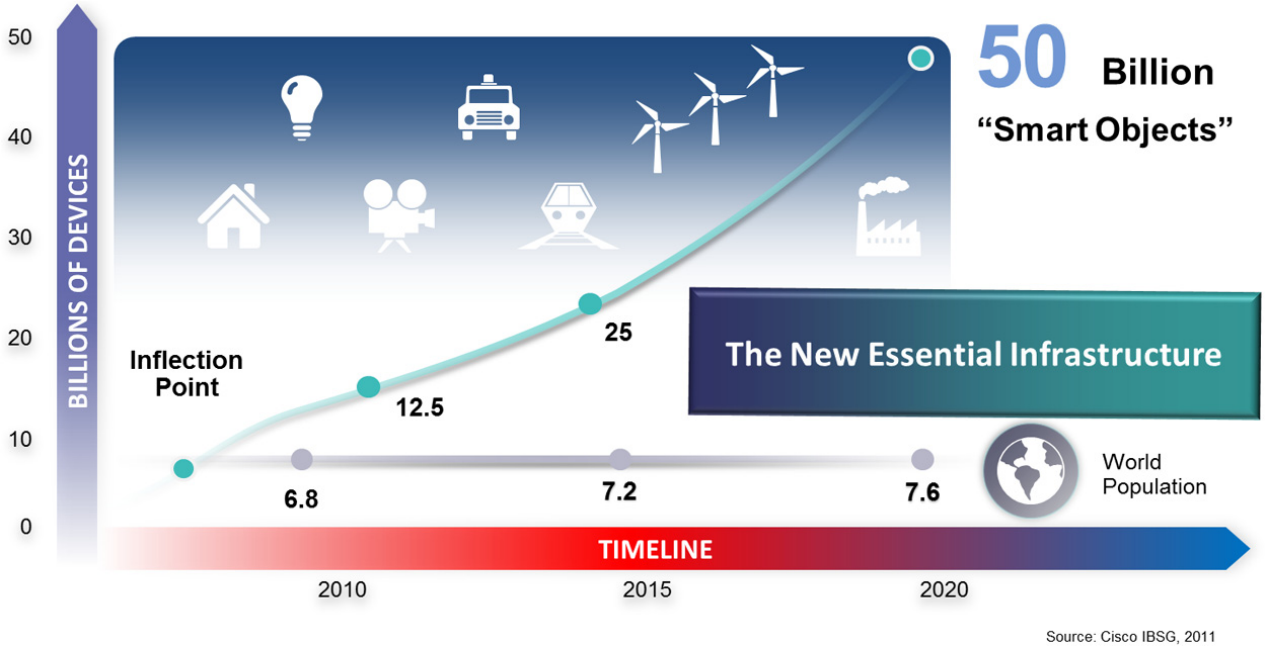
Advances in sensor and microprocessor design

- Bluetooth
- RFID
- ZigBee
- WiFi
- 4G networks
- …

Advances in connectivity and networks

- Smaller and more durable sensors
- Multi-processor chips
- Increasing processor performance and efficiency
- Lower costs

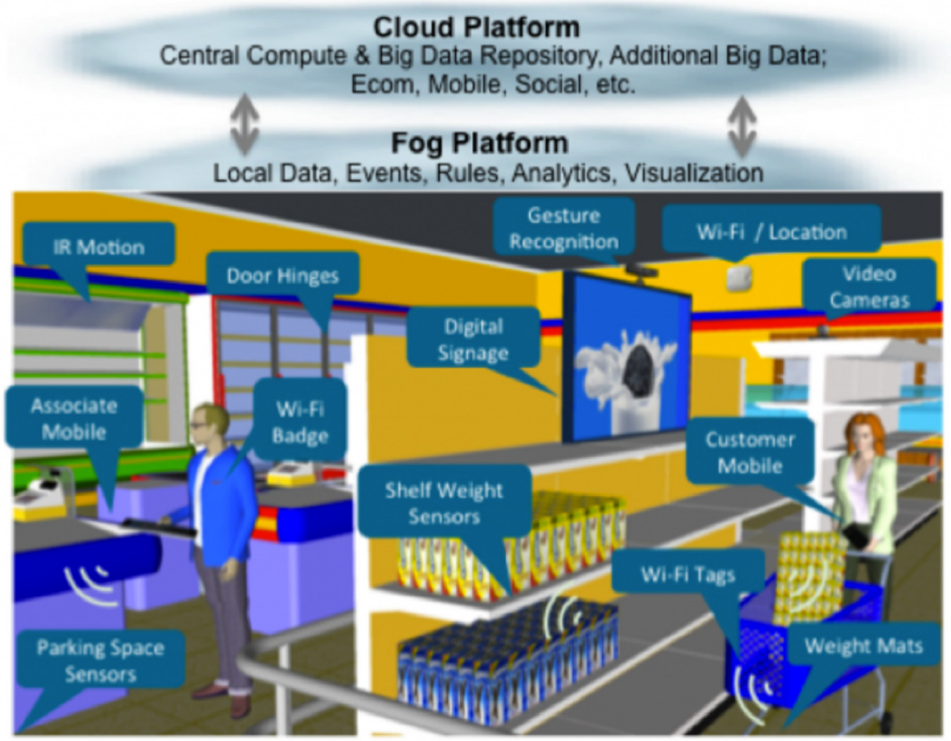
IoT Future Revolution



By 2020 the number of Internet-connected devices is expected to reach 50 billion.

IoT Applications

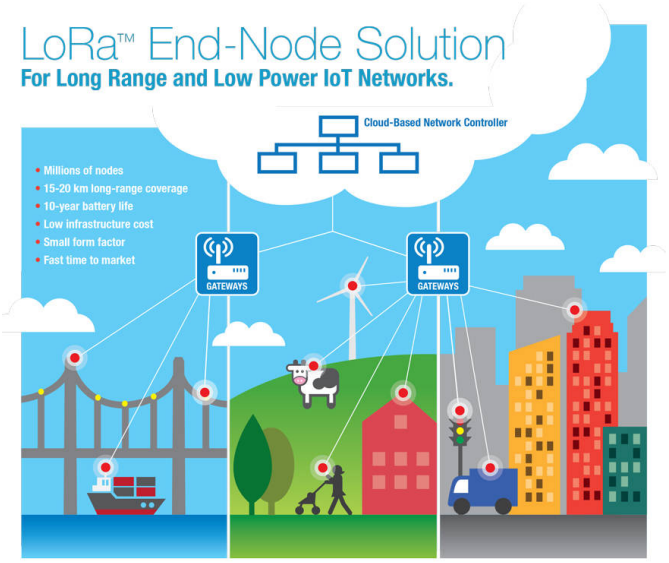
Digital Retail Store (source: Cisco)

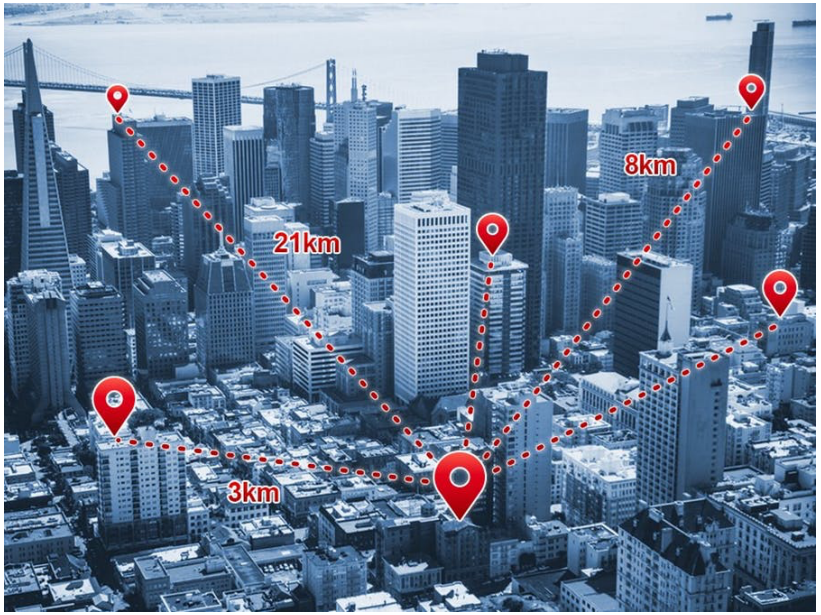


Flexible, hyper-local, real-time, sensor fusion, and big data analytics driving the next generation of Retail Value Chains

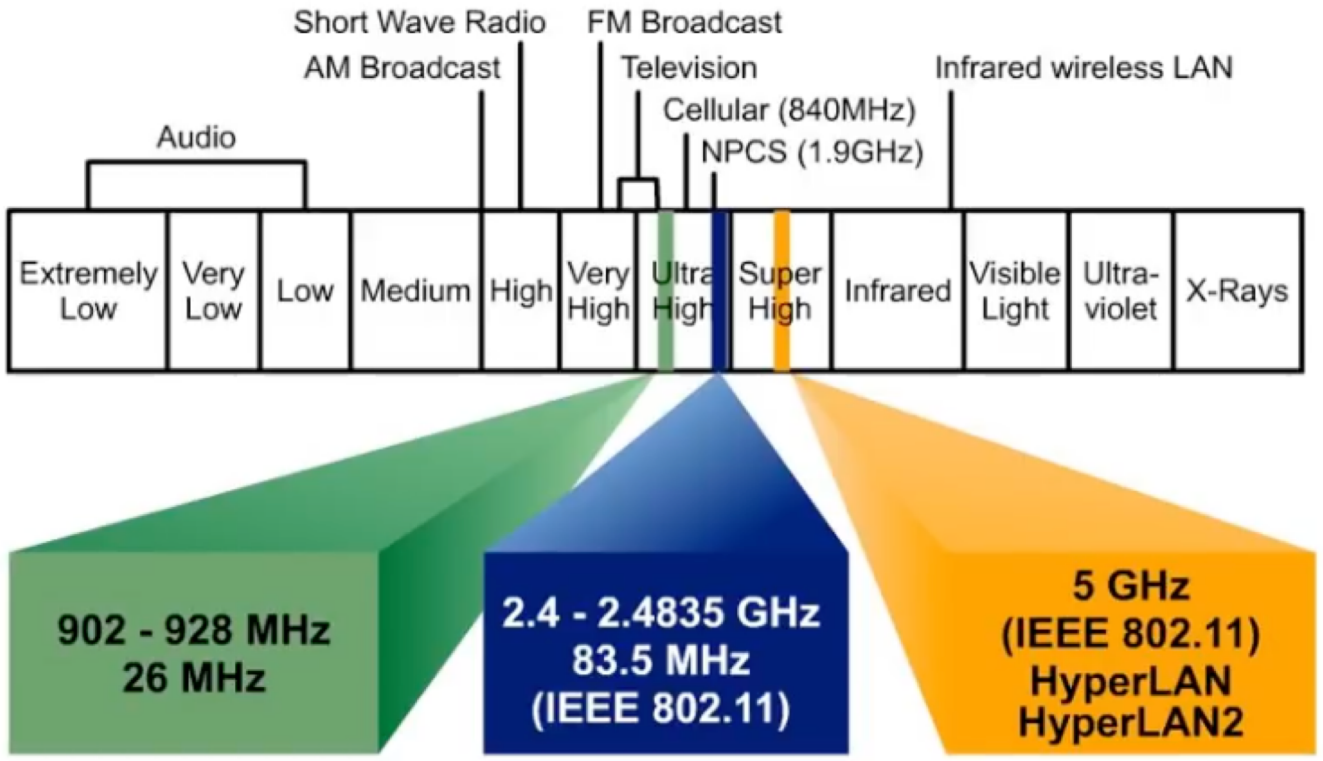
LoRa Technology

- A physical layer or wireless modulation for IoT
  - Long range communication (> 10 km in rural areas)
  - Robust communication (Chirp Spread Spectrum)
  - Low power (> 10 years battery life)
  - Large network capacity (a large number of nodes in a network)





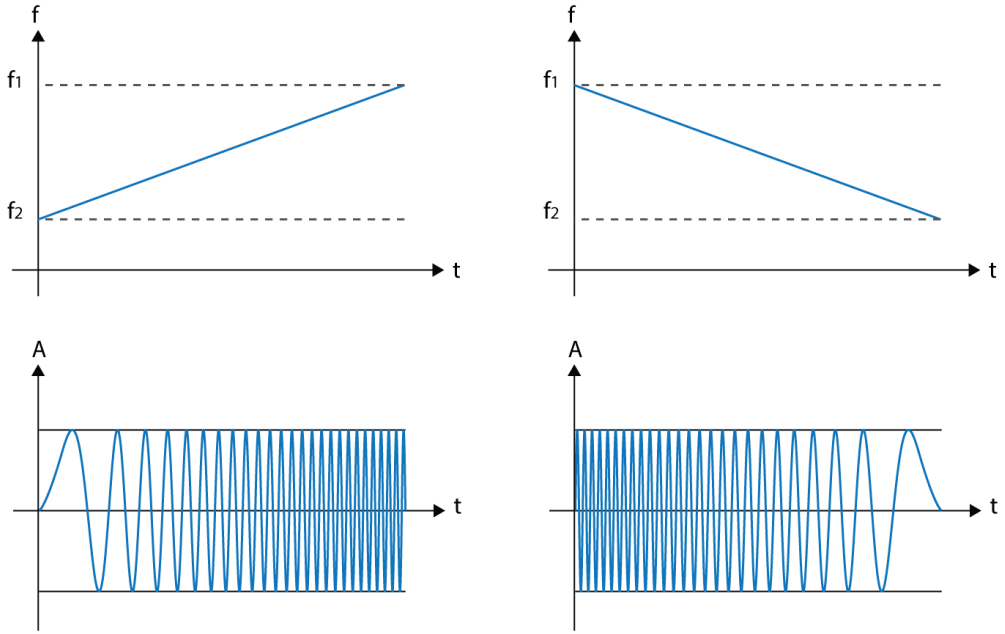
LoRa Frequency Bands



169 MHz, 433 MHz, 868 MHZ (Europe), 915 MHz (North America)

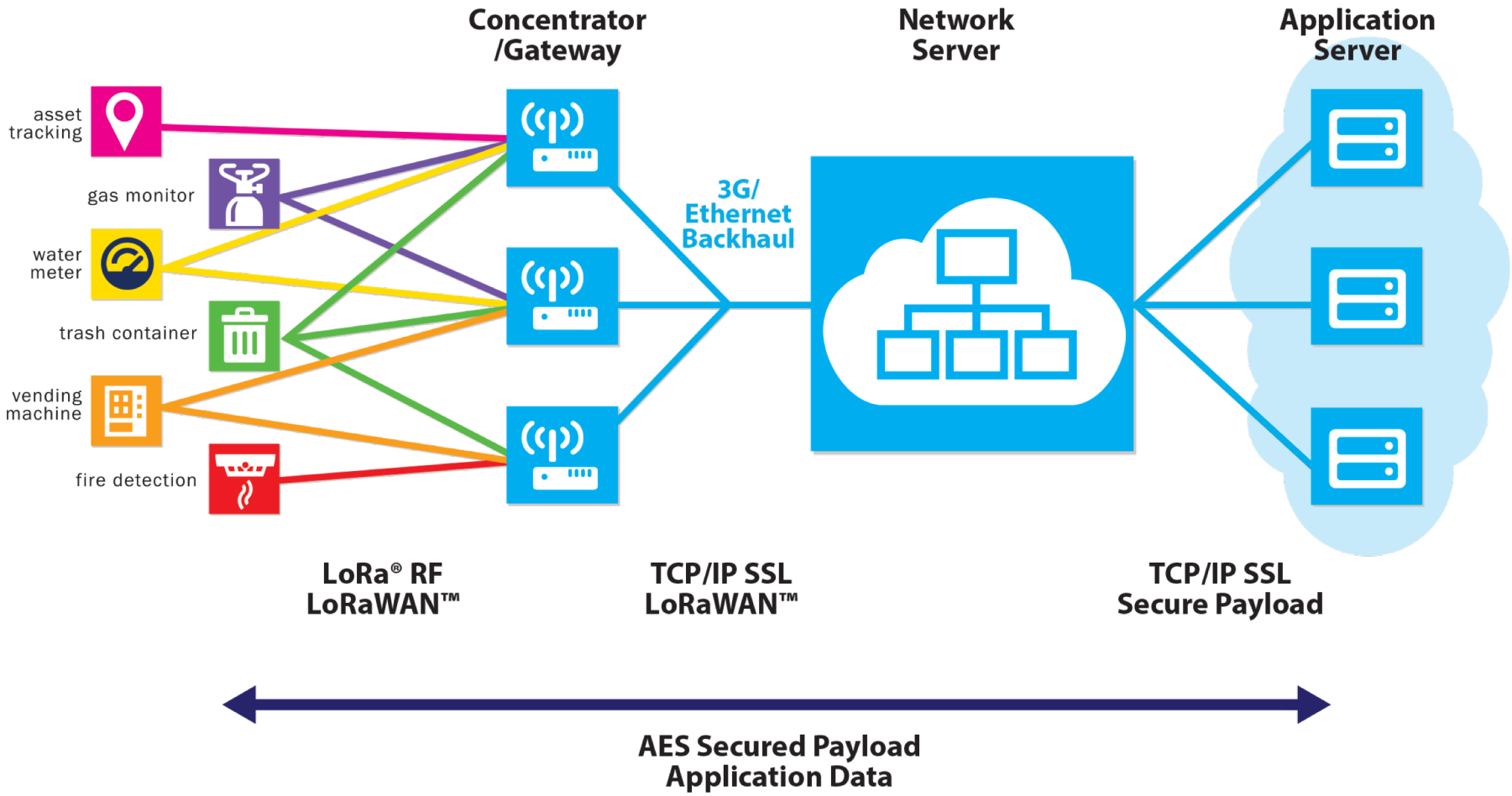
Chirp Spread Spectrum (CSS)

- A chirp is a signal in which the frequency increases (up-chirp) or decreases (down-chirp) with time.



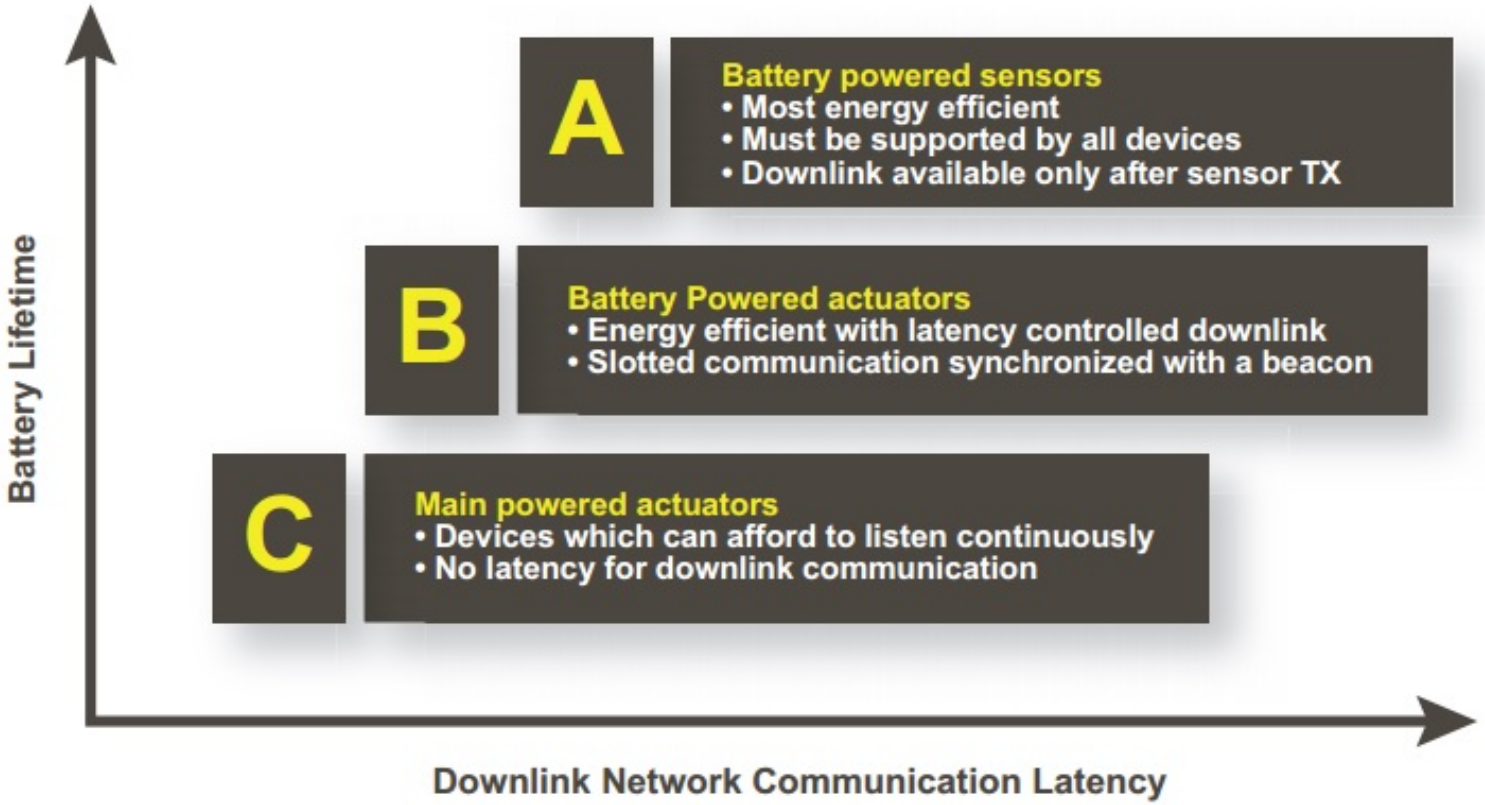
- Spread Factor = chip rate / symbol rate
  - a value between 7 and 12

LoRaWAN Architecture



Communication in LoRaWAN

- Three classes: Class A, Class B and Class C



Summary

- Wireless sensor networks
  - Characteristics and Applications
  - TinyOS and Contiki
- Cyber-physical systems
  - Definitions and applications
- Internet of Things
  - Architecture
  - Enable technologies
- LoRa and LoRaWAN
  - Features and Architecture