Internet des objets - IoT

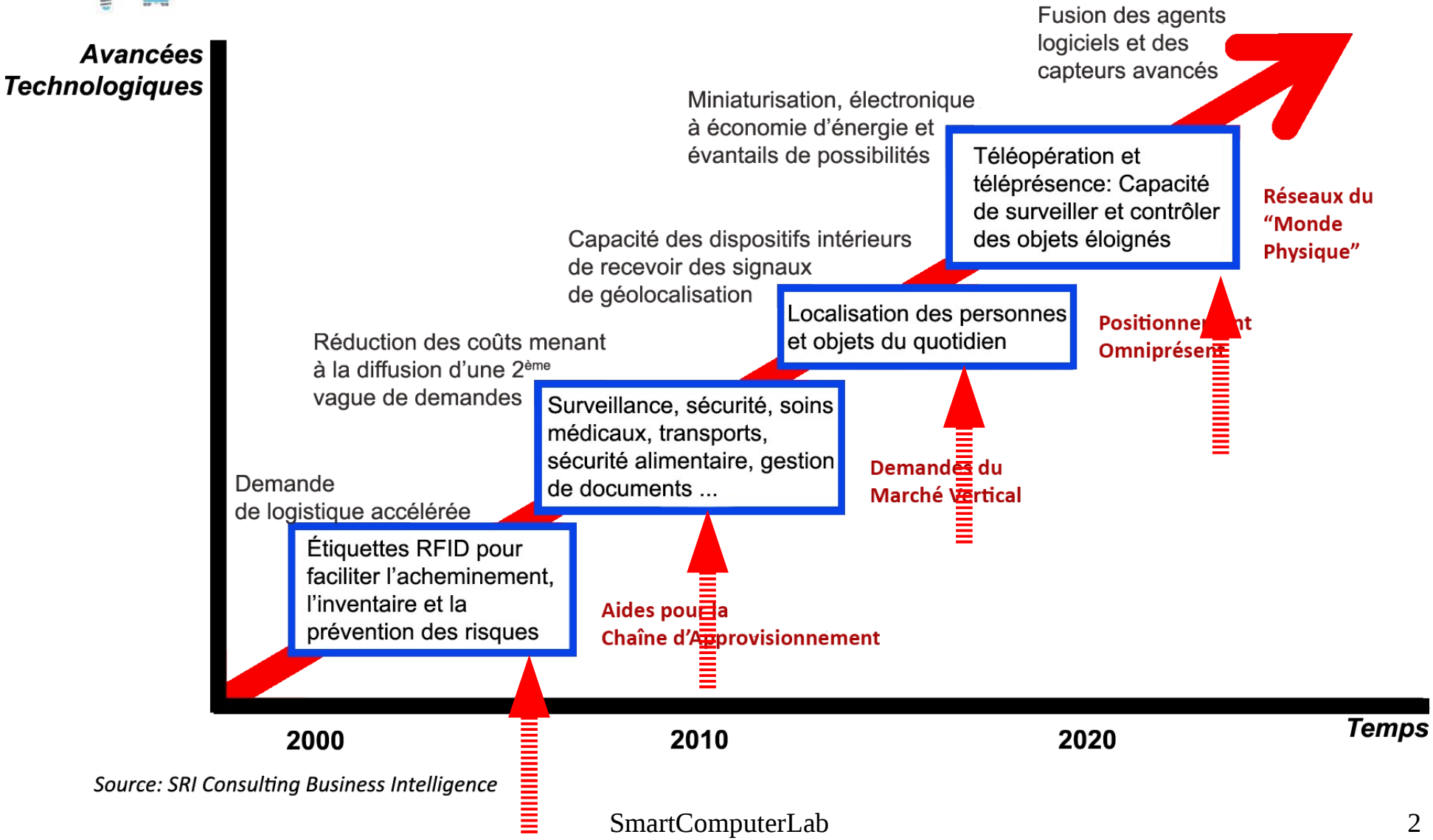
L'Internet des objets, ou **IdO** (en anglais *Internet of Things*, ou **IoT**) est l'interconnexion entre Internet et des objets, des lieux et des environnements physiques.

L'appellation désigne un nombre croissant d'**objets connectés à Internet** permettant ainsi une communication entre nos biens dits physiques et leurs existences numériques.

Ces formes de connexions permettent de rassembler de nouvelles masses de **données sur le réseau** et donc, de nouvelles connaissances et formes de savoirs.

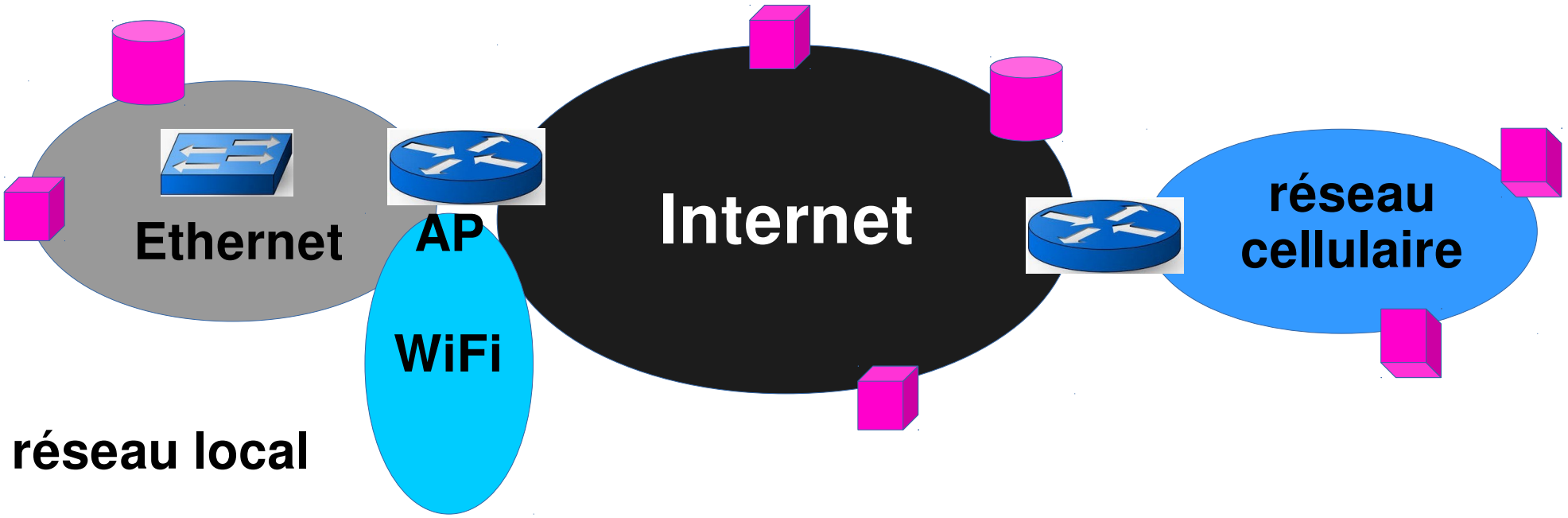
Internet des objets - IoT

Historique de la Technologie: la Connectivité des choses



Source: SRI Consulting Business Intelligence

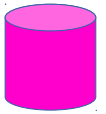
Principaux composants d'**Internet**



commutateur

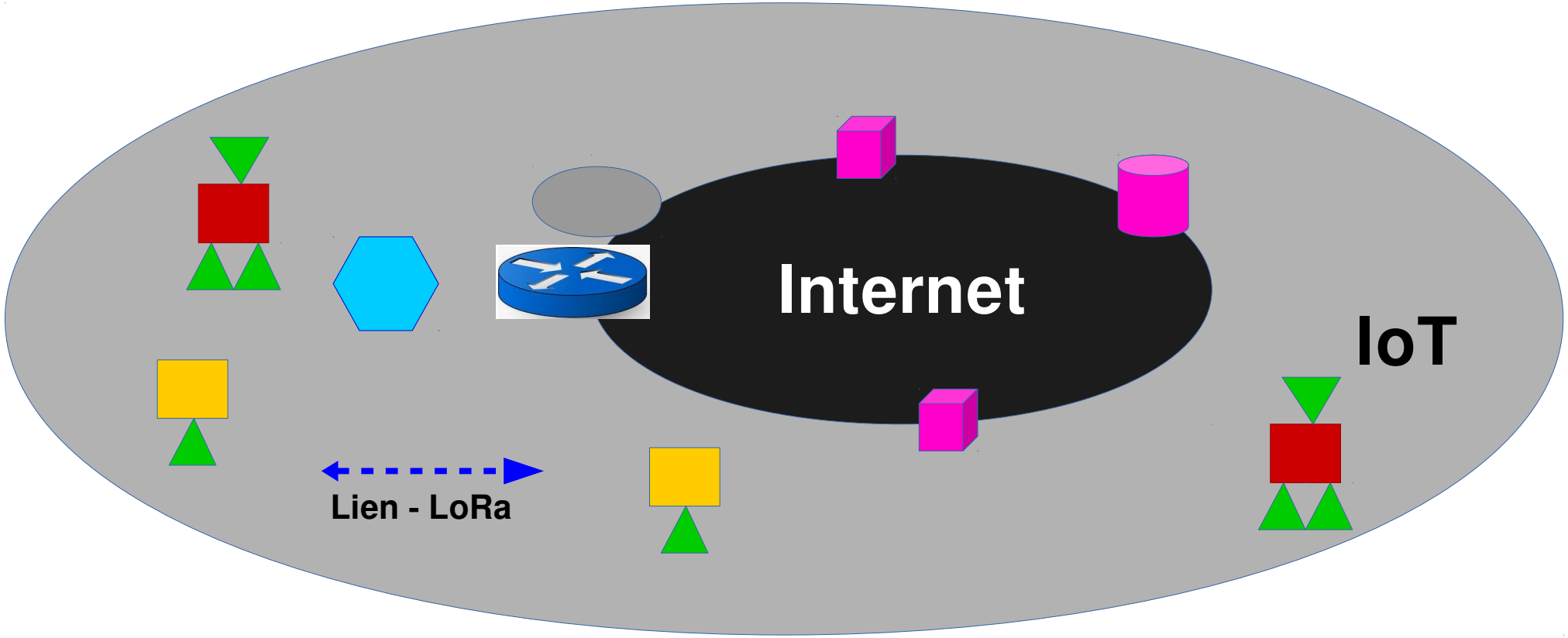
routeur avec un point d'accès - AP

client PC/Tablette/Smartphone



serveur

Principaux composants **IoT**



capteur

actionneur

terminal mono-tâche

terminal multi-tâche

passerelle

routeur

serveur

client

Evolution du Matériel pour IoT (IA)
unités centrales pour terminaux et passerelles

2003 -

- **MCU** - Arduino - simple microcontrôleur pour les applications embarquées
- **SBC** – Raspberry-Pi (2,3) – carte ordinateur embarqué

2012 -

- **MCU - ESP8266 (ESP12) puis ESP32**
  - puissant microcontrôleur avec WiFi et BT – `freeRTOS`
  modems LoRa
- **SBC Nvidia** - Jetson (TK1,TX1,TX2), Jetson Nano – puissantes cartes ordinateur embarqué - `Linux Ubuntu` unités IA pour *deep learning*, *deep streaming* etc. (GPU+TensorFlow,..)

Plate-forme pédagogique – **IoT DevKit**

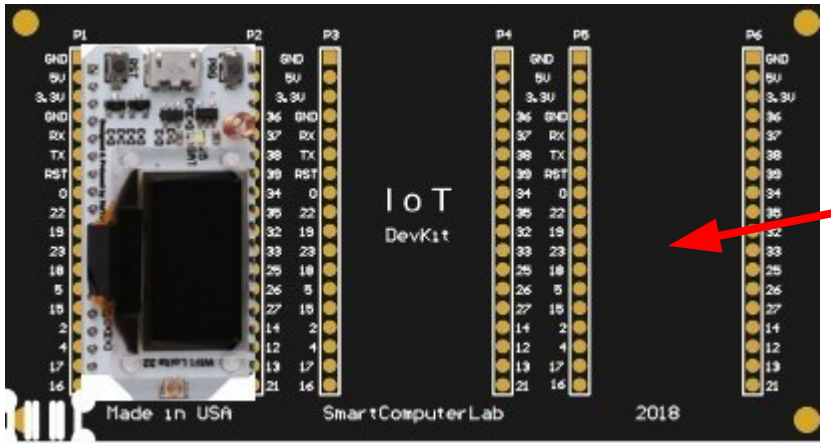
La plate-forme **IoT DevKit** a été développée à l'Université de Nantes (**Polytech'Nantes**) au sein de **SmartComputerLab** qui est associé au laboratoire **LS2N** – équipe **RIO** (Réseaux et Internet des Objets).

IoT DevKit consiste en :

- **cartes de base** (3 formats)
- **cartes MCU et SBC** (ESP12, ESP32, NanoPi)
- **cartes d'extension** pour de multiples **capteurs**, **actionneurs** et **modems** de communication

Le kit est complété par les **préparations** (logiciel) pour les **laboratoires IoT**

Plate-forme pédagogique – **IoT DevKit** format large avec une batterie



emplacements pour
les cartes d'extension

format mini D1

Cartes de base avec une unité MCU – (ESP32+WiFi,LoRa,..)

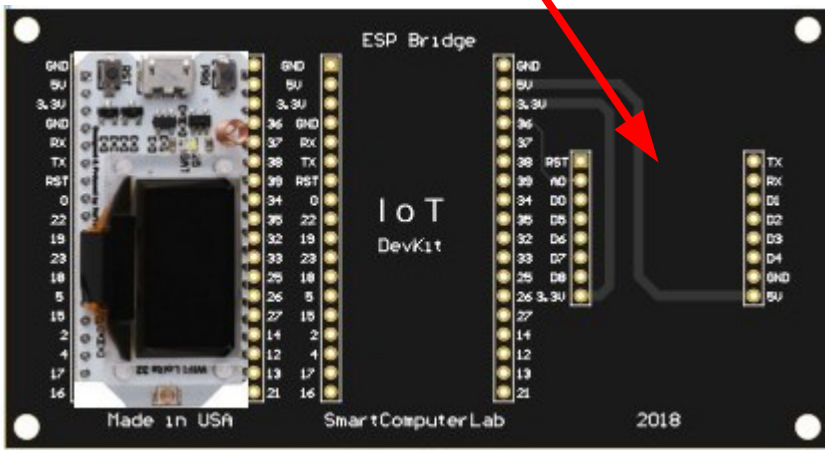
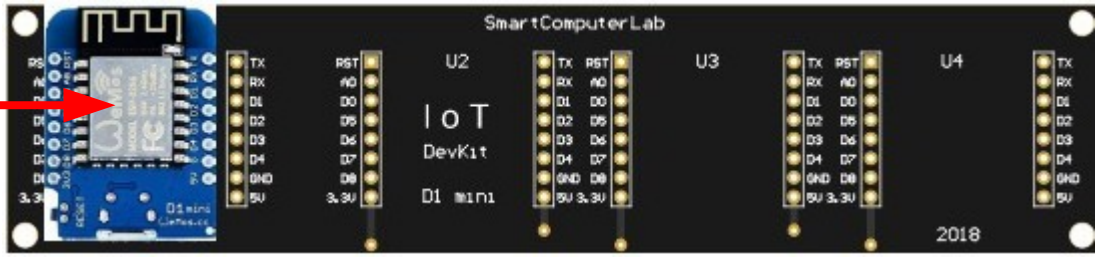
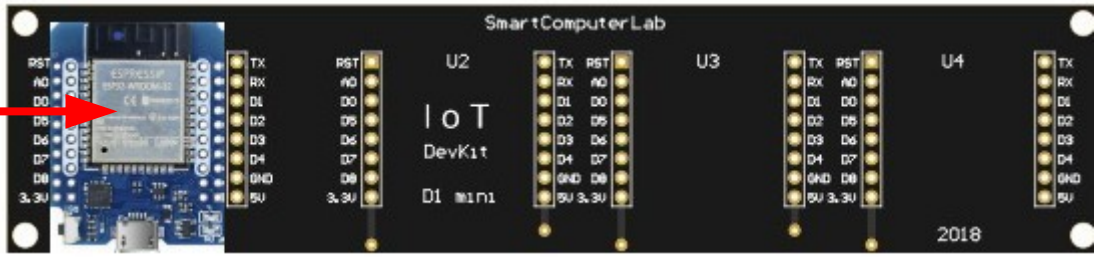


Plate-forme pédagogique – **IoT DevKit** format longue

ESP12
(Wemos)

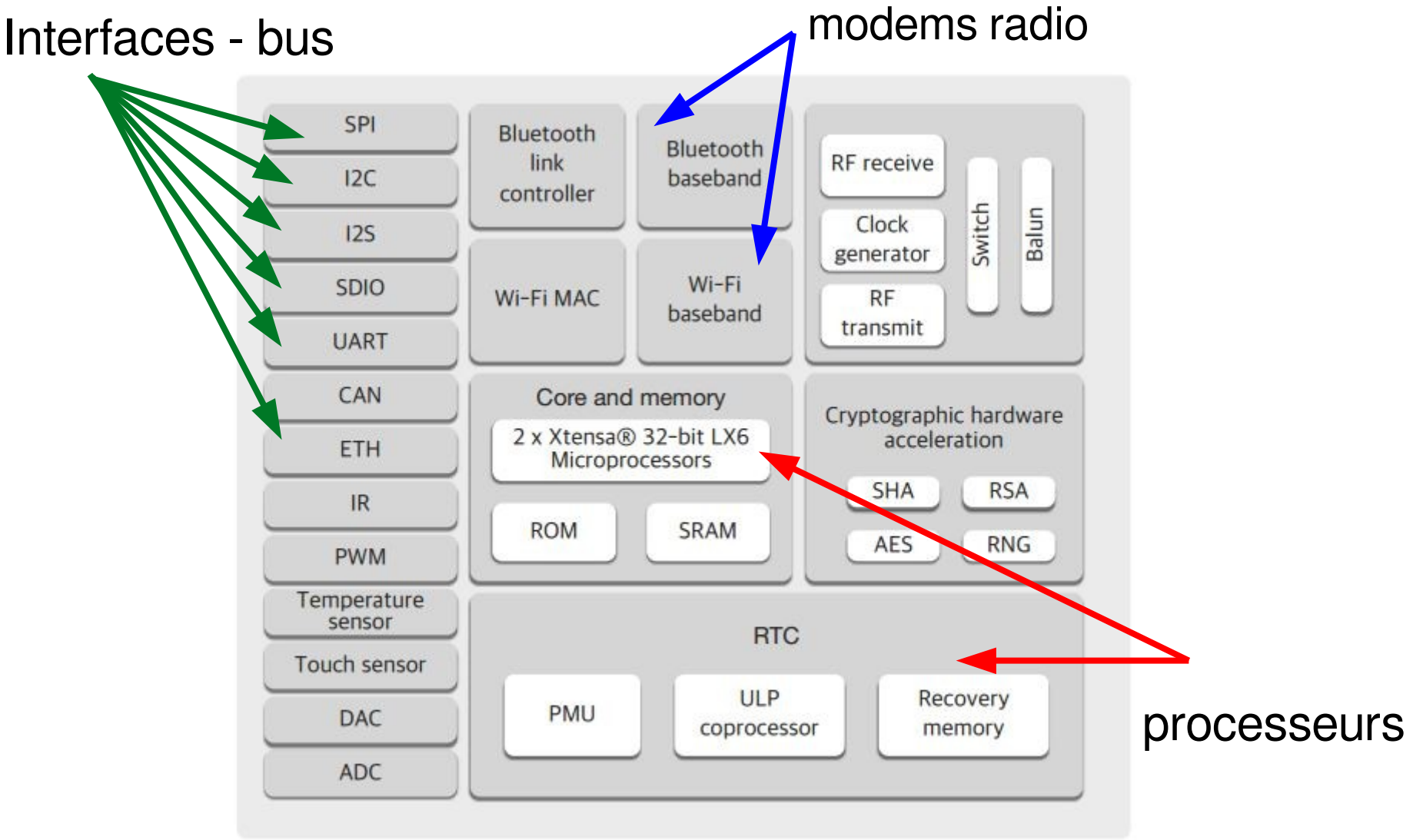


ESP32
(Wemos)



emplacements pour les cartes d'extension en format mini D1

Architecture ESP32 : le « top » de MCU pour IoT

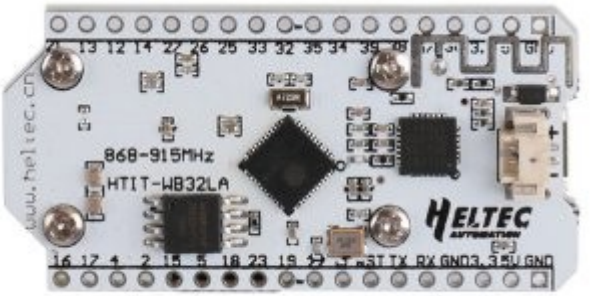


Cartes ESP32 (ESP12)

WiFi/BT/BLE

LoRa

Heltec Wifi LoRa 32 *V2* 868MHz *top*



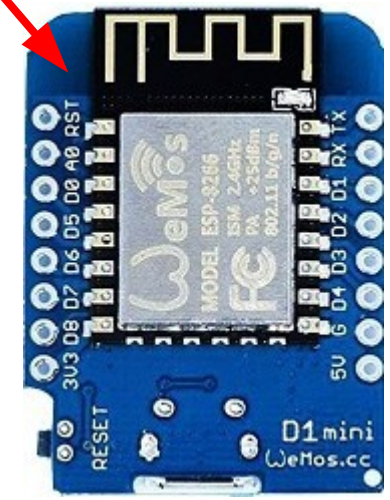
Heltec Wifi LoRa 32 *V2* 868MHz *bottom*

Carte MCU (Heltec) ESP32 + LoRa + OLED

pour la plate-forme large avec une batterie

Cartes ESP32 (ESP12)

WiFi



WiFi/BT/BLE

Carte MCU (Wemos) ESP12 et ESP32

pour la plate-forme longue – cartes d'extension au format mini D1

Cartes d'extension (exemples)

UART

Cartes d'extension pour les emplacements « longues »

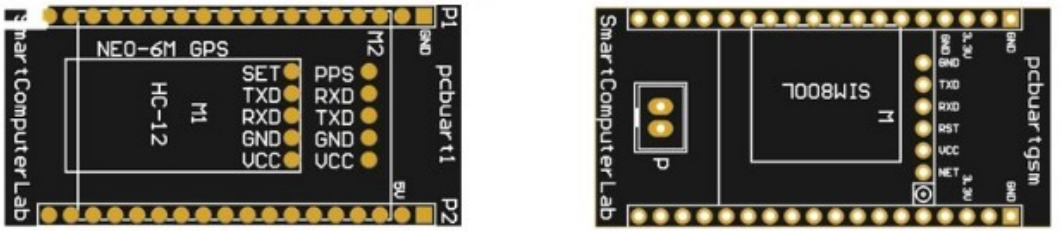
Une trentaine !

I2C

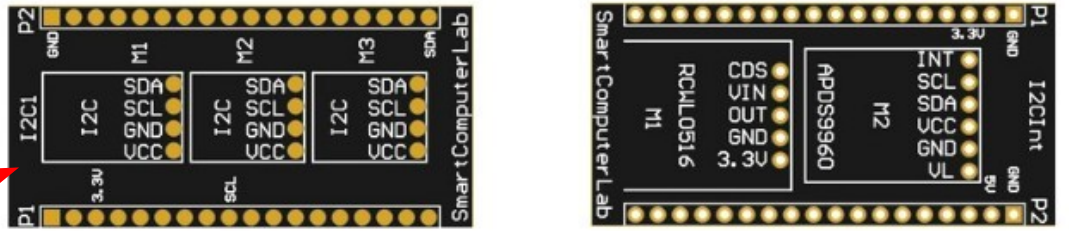
SPI

spécifique

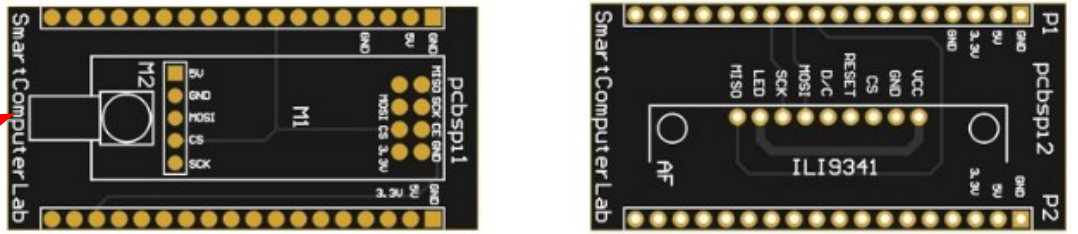
Cartes d'extension pour le bus **UART** (deux exemples):



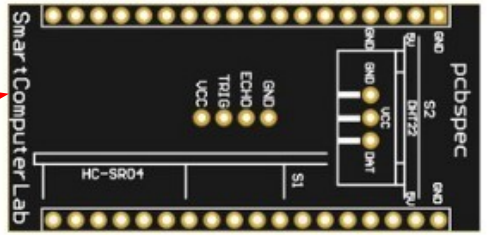
Cartes d'extension pour le bus **I2C** (deux exemples):



Cartes d'extension pour le bus **SPI** (deux exemples):

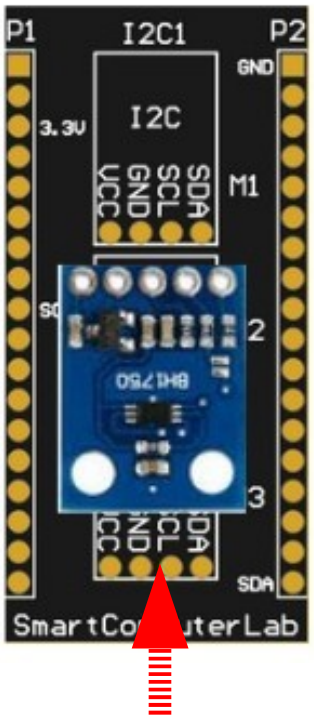


Cartes d'extension spécifiques (un exemple):

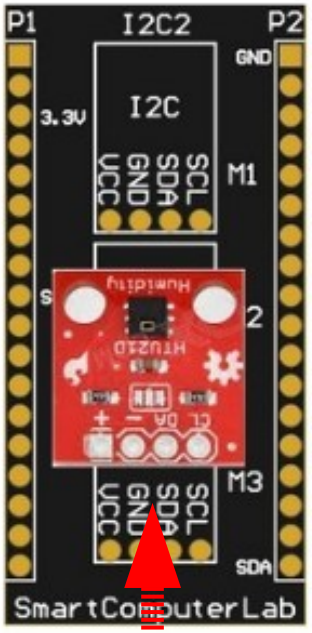


Cartes d’extension avec capteurs (exemples)

Cartes d'extension **I2C** avec capteurs **BH1750** et **HTU21D** et une carte d'extension **UART** avec module GPS NEO-M6



Carte - bus I2C et un capteur de luminosité - BH1750



Carte - bus I2C et un capteur de température/humidité - HTU21

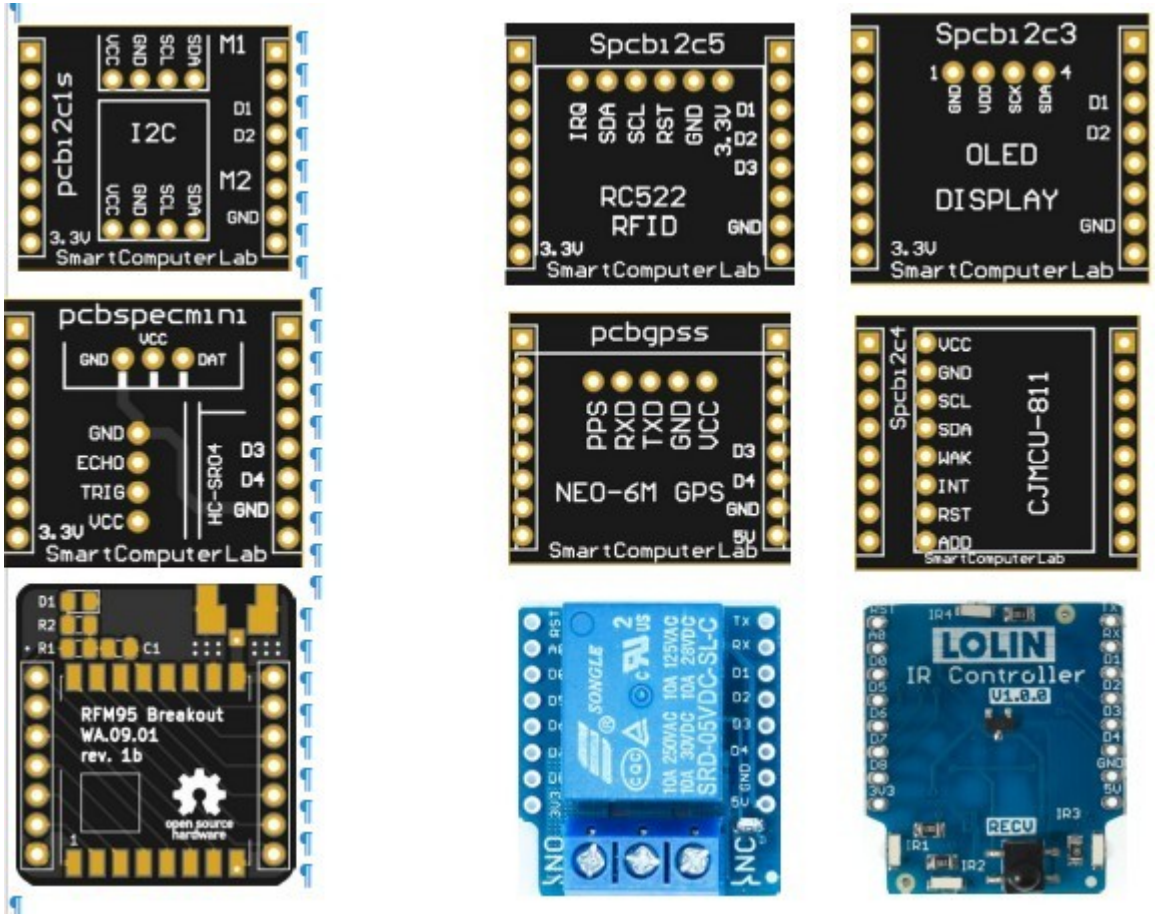
Carte - bus UARTet module GPS – NEO-M6

Cartes d’extension format mini D1

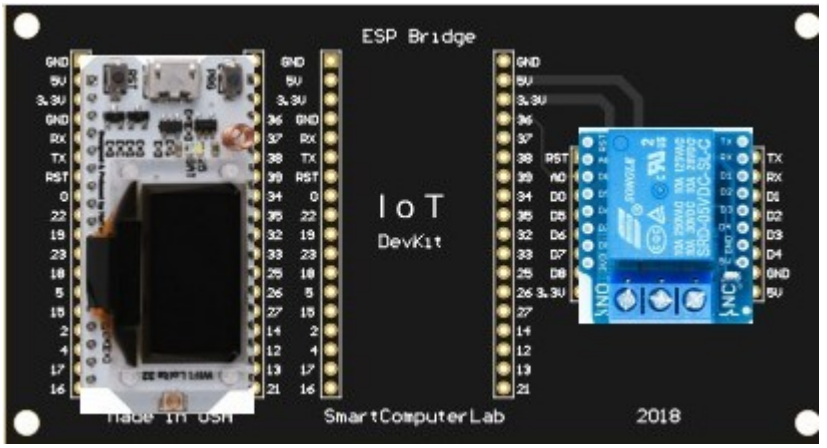
Cartes d’extension de **SmartComputerLab**

Une vingtaine !

Cartes d’extension « externes »
En dizaines !

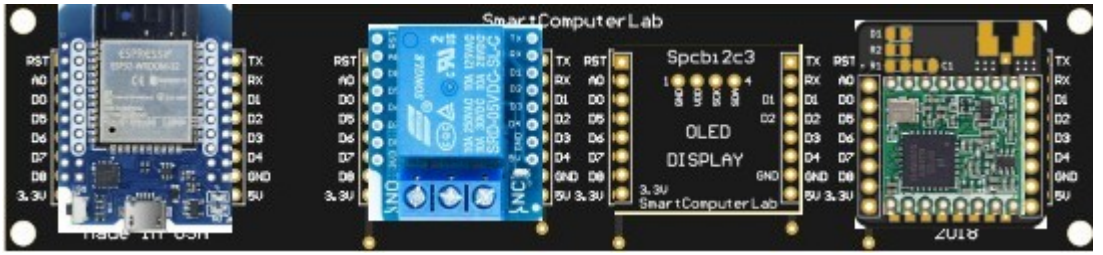


Exemple d'une architecture IoT

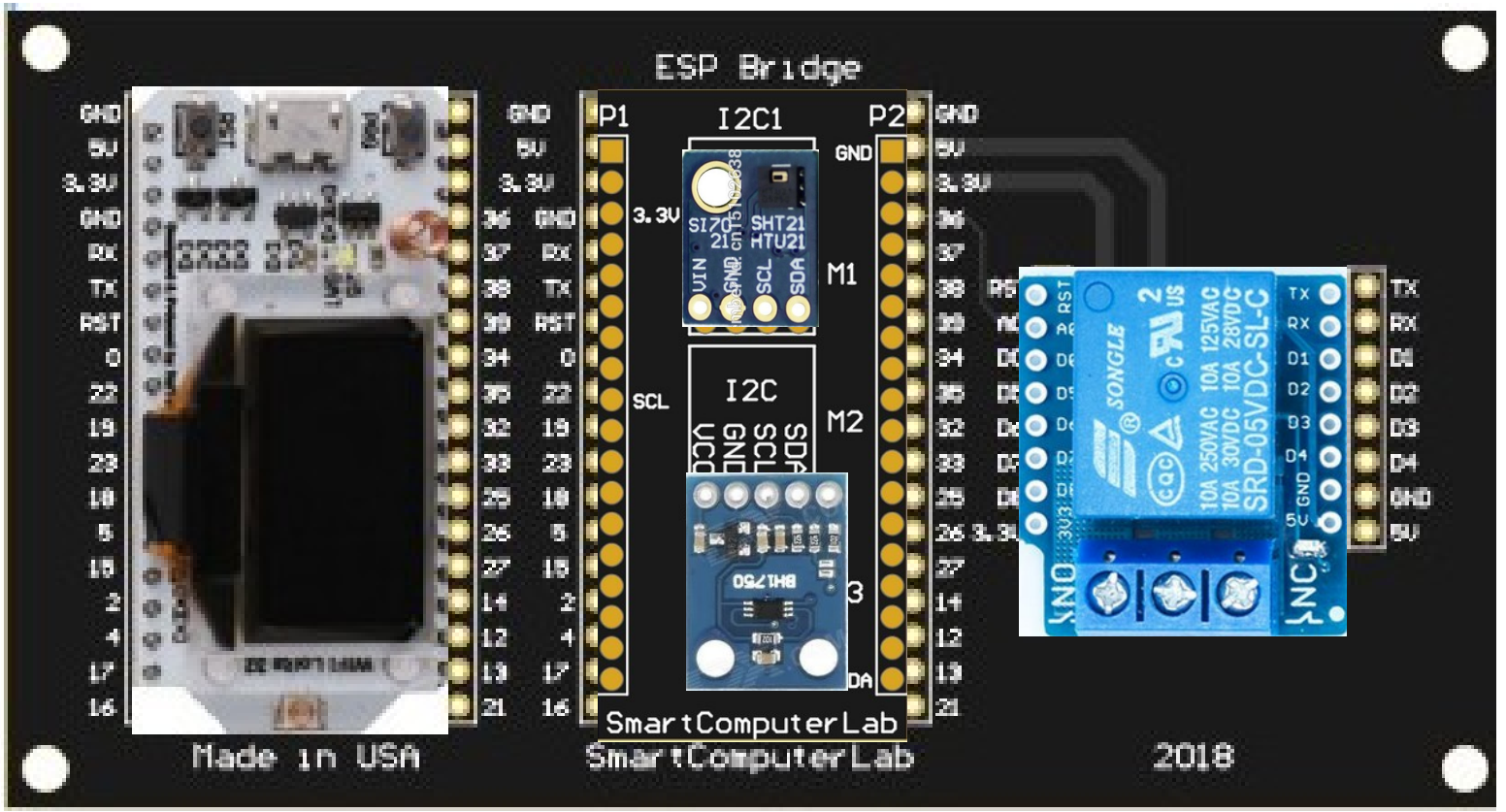


MCU – ESP32+LoRa+OLED
+ carte relais

MCU – ESP32
+ carte relais
+ carte OLED
+ carte LoRa

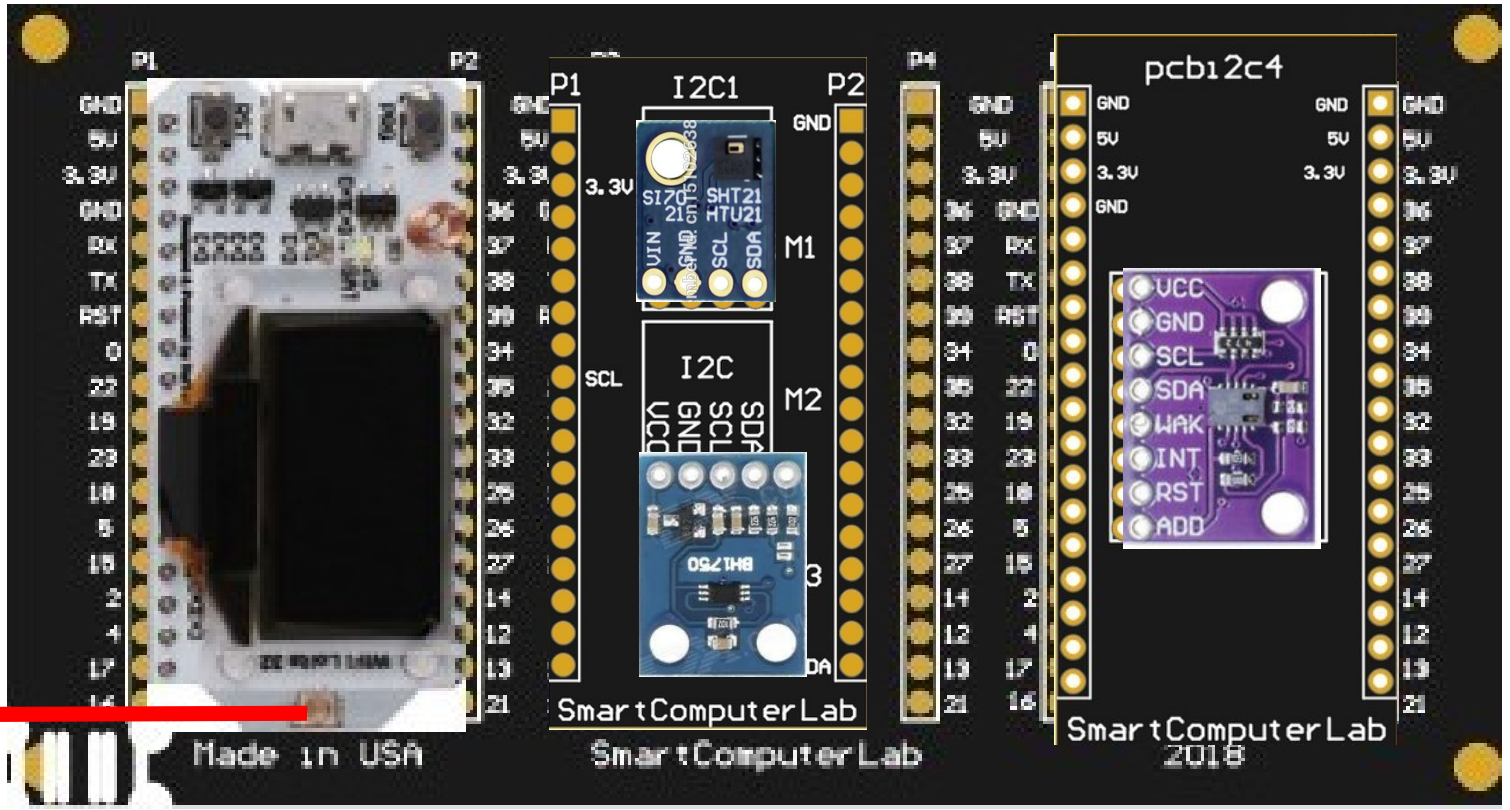


Exemples, exemples, exemples, ..



Lecture de température, humidité, luminosité
Envoi par WiFi (Internet) sur le serveur,
Réception de la commande pour le relais

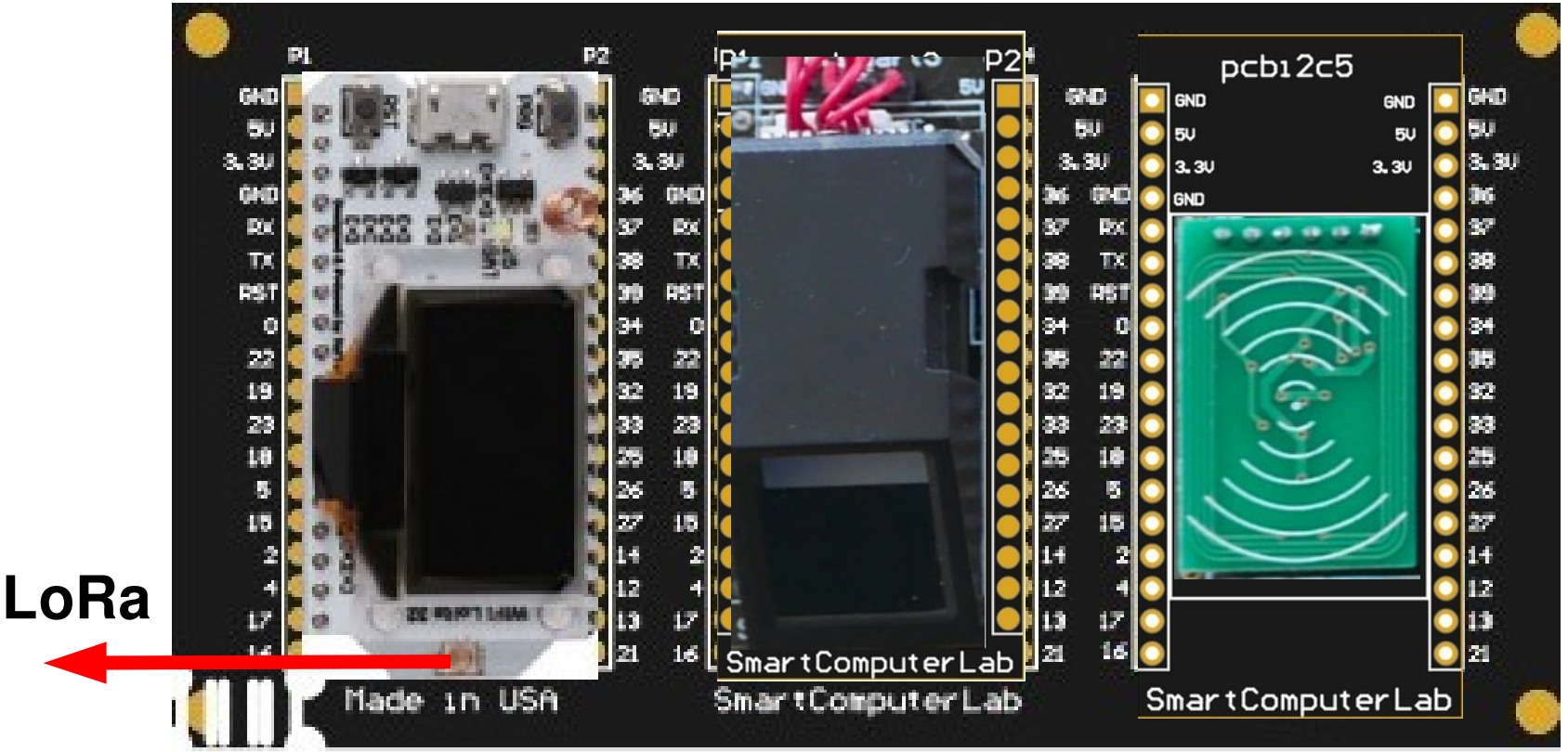
Exemples, exemples, exemples, ..



LoRa

Lecture de température, humidité, luminosité, CO2, particules,
Envoi par LoRa vers une passerelle LoRa-WiFi (Internet)
Réception de la commande pour le relais

Exemples, exemples, exemples, ..

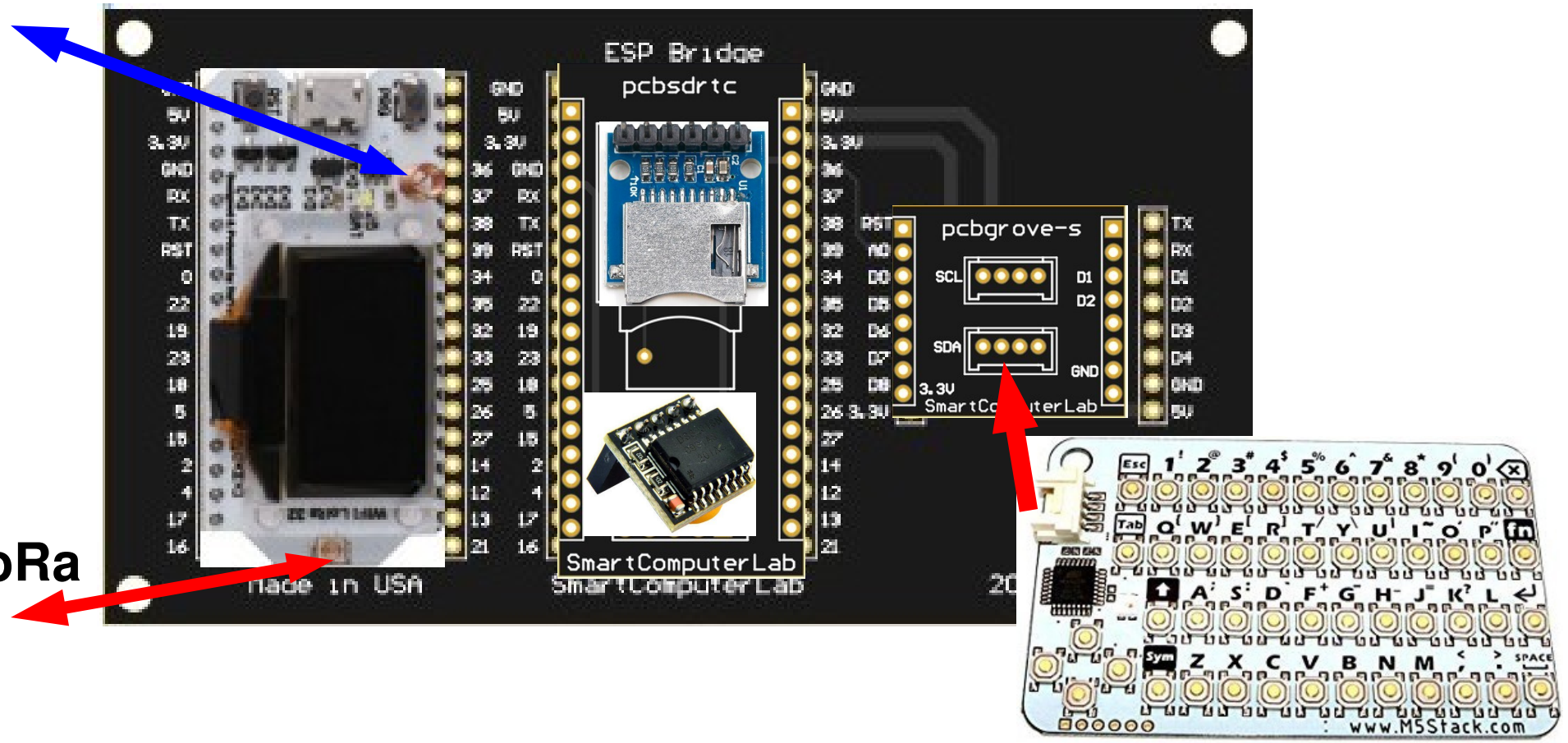


Lecture empreintes, carte RFID
Envoi par LoRa vers une passerelle LoRa-WiFi (Internet)

Exemples, exemples, exemples, ..

WiFi

LoRa



Passerelle **LoRa-WiFi** configurable

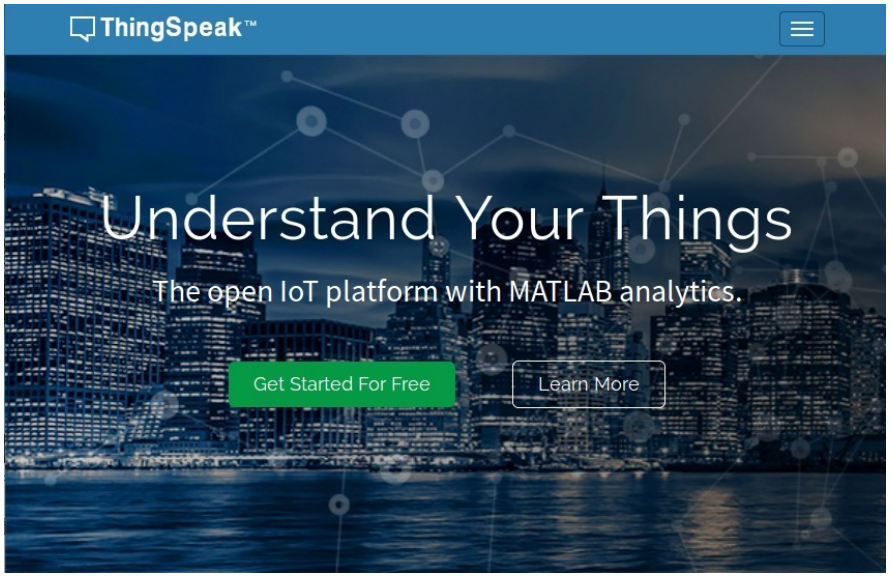
Réception-Envoi des données sur le serveur **Thingspeak**

Stockage des données horodatées sur une carte **SD**

Serveurs IoT

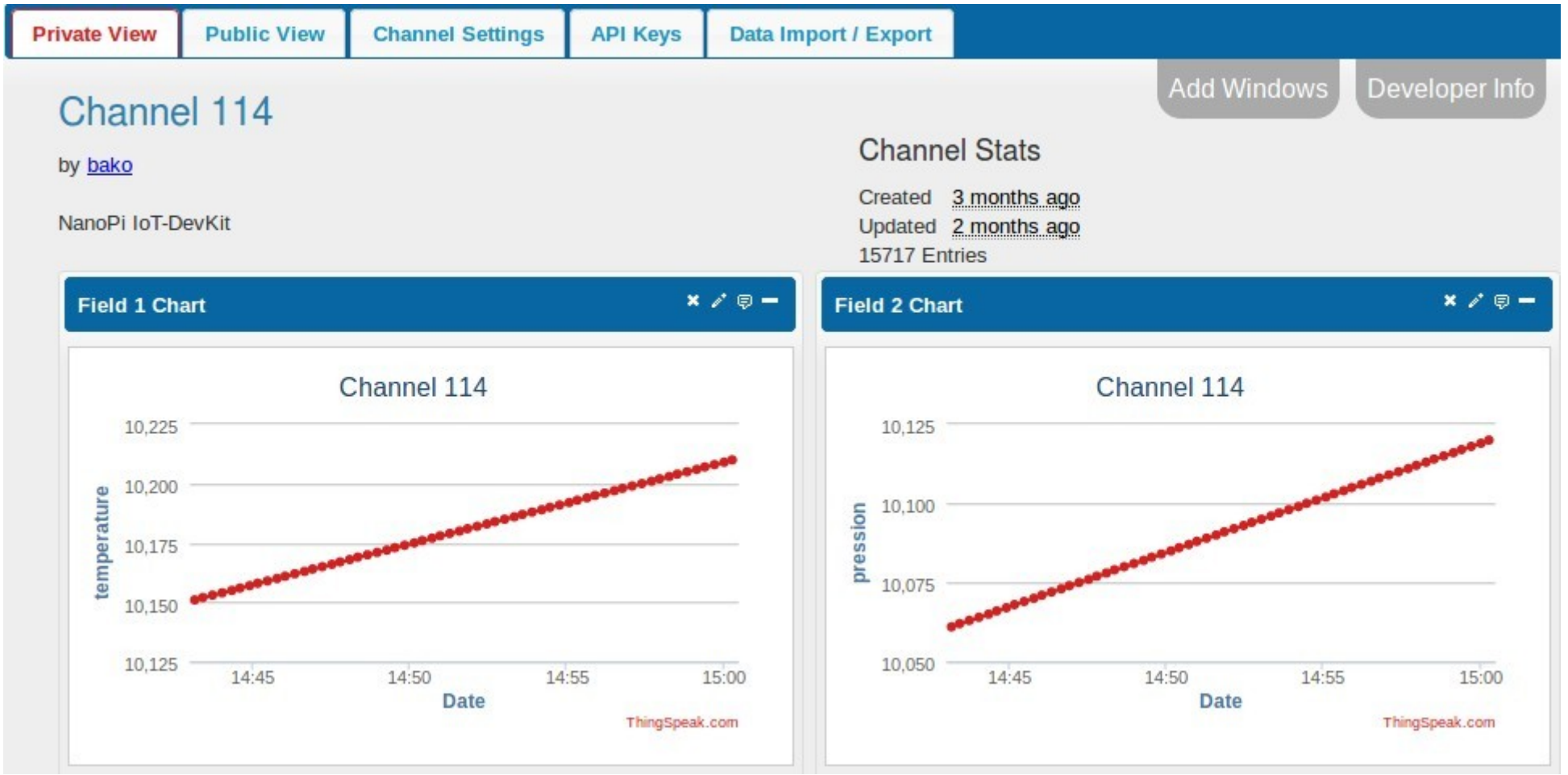
Il existe plusieurs types de serveurs pour IoT.
Fonctions : stockage , visualisation (IU), traitements

Exemple - **`ThingSpeak`** (open source)
`Thingspeak.com` (MatLab), **`Thingspeak.fr`** (SmartComputerLab)



Serveurs IoT

Un canal ThingSpeak avec les champs (*fields*) :



Programmation – Arduino IDE

Télécharger le fichier d'installation Arduino IDE

- Linux
- Windows

Installer l'IDE

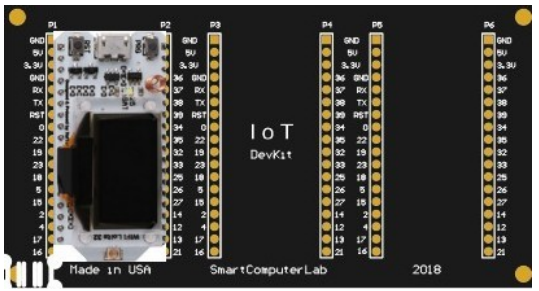
Ajouter les cartes ESP32 et ESP12 (8266)

Dans les préférences :
https://dl.espressif.com/dl/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json

Puis dans le *Boards Manager*

Un très simple exemple du code

Nous allons utiliser la carte « large » avec le MCU – Heltec : ESP32+OLED+WiFi+LoRa



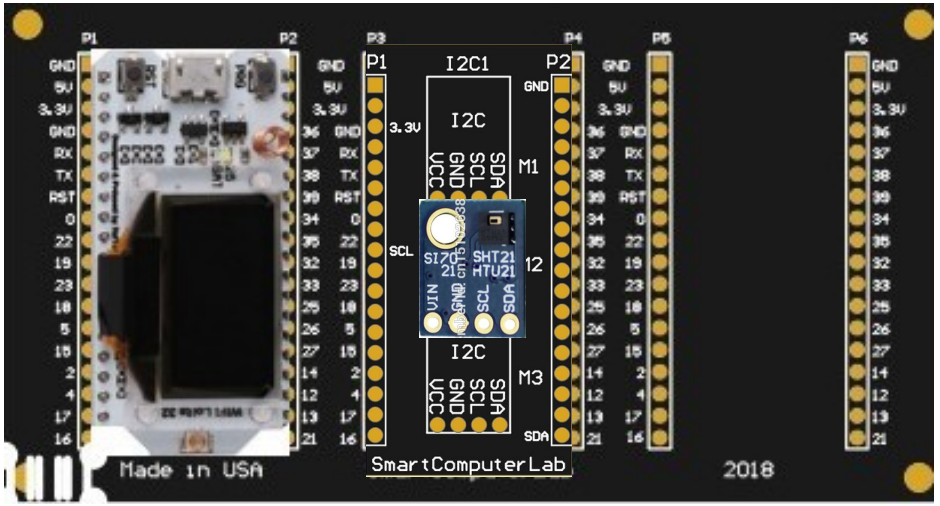
Nous allons tester un bouton (pin 0) et un LED (pin 25).

```
const int ButtonPin=0;  // PRG
const int LEDPin=25;    // LED

void setup() {
  pinMode(LEDPin, OUTPUT);
  // sens de fonctionnement : INPUT, OUTPUT
}
```

Un très simple exemple du code

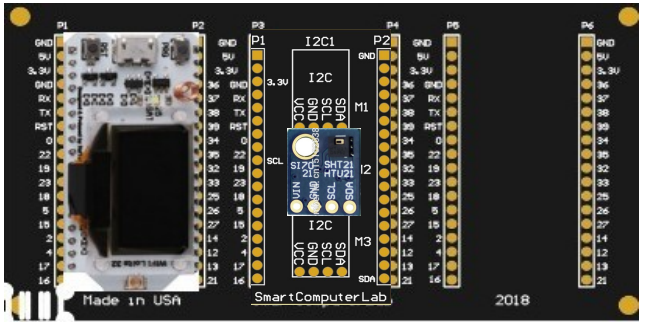
```
const int ButtonPin=0;  // PRG
const int LEDPin=25;    // LED

void setup() {
  pinMode(LEDPin, OUTPUT);  // sens du fonctionnement : INPUT, OUTPUT
}

void loop() {
{
int  buttonState;
  buttonState = digitalRead(ButtonPin);  // lecture de l'etat
  if(buttonState)
    digitalWrite(LEDPin,HIGH);      // envoi du signal : HIGH, LOW
  else
    digitalWrite(LEDPin,LOW);
  delay(30);                        // attente de 30 ms
  }
}
```

Lecture des données sur SHT21




```
#include <Wire.h>
#include <Sodaq_SHT2x.h>

void setup()
{
  Wire.begin(21,22);
  Serial.begin(9600);
}

void loop()
{
  Serial.print("Humidity(%RH): ");
  Serial.print(SHT2x.GetHumidity());
  Serial.print("     Temperature(C): ");
  Serial.println(SHT2x.GetTemperature());
  delay(1000);
}
```

Lecture des données sur SHT21 et l'affichage sur l'écran OLED




```
#include <Wire.h>
#include <Sodaq_SHT2x.h>

#include <U8x8lib.h>  // bibliothèque à charger a partir de
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15,4,16); // clock, data, reset

float temperature,humidite;

void dispData()
{
  char dbuf[16];
  u8x8.clear();
  u8x8.drawString(0,1,"SmartComputerLab");
  sprintf(dbuf,"temp:%f",temperature); u8x8.drawString(0,3,dbuf);
  sprintf(dbuf,"humi:%f",humidite); u8x8.drawString(0,4,dbuf);
  delay(6000);
}
```

Envoi des données sur ThingSpeak

Préparations :

Adresse IP et numéro de port du serveur (pour `Thingspeak.fr`) dans `ThingSpeak.h`

```
#define THINGSPEAK_IPADDRESS IPAddress(90,105,148,36)
#define THINGSPEAK_PORT_NUMBER 443
```

Ajouter au début du code :

```
#include <WiFi.h>
#include "ThingSpeak.h"
```

Utiliser les fonctions pour précharger et envoyer les données :

```
ThingSpeak.setField(1, sensor[0]);  // préparation du field1
ThingSpeak.setField(2, sensor[1]);  // préparation du field2
```

Puis

```
ThingSpeak.writeFields(myChannelNumber[1], myWriteAPIKey[1]);  // envoi
```

Connexion WiFi

```
#include <WiFi.h>
#include "ThingSpeak.h"
char ssid[] = "PhoneAP";              //   SSID du reseau
char pass[] = "smartcomputerlab";  //   mot de passe par defaut
unsigned long myChannelNumber = 100;
const char * myWriteAPIKey="MEH7AOFHAMNWJE8P" ;
WiFiClient  client;

void setup() {
Serial.begin(9600);
WiFi.disconnect(true);  // effacer de l'EEPROM WiFi credentials
delay(1000);
WiFi.begin(ssid, pass);
delay(1000);
   while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    }
```

Envoi des données sur ThingSpeak

```
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");Serial.println(ip);
  Serial.println("WiFi setup ok");delay(1000);
  ThingSpeak.begin(client); // connexion TCP au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
}

int tout=10000;  // en millisecondes
float luminosity=100.0, temperature=10.0;

void loop()
{
ThingSpeak.setField(1, luminosity);  // préparation du field1
ThingSpeak.setField(2, temperature);  // préparation du field1
    while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");}
ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
luminosity++;temperature++;
delay(tout);
}
```

LoRa sur IoT DevKit

LoRa (Long Range) est une technologie de communication de données numérique sans fil brevetée développée par Cycleo à Grenoble, et acquise par Semtech en 2012, membre fondateur de l'Alliance LoRa.

LoRa est un protocole de communication sans fil longue portée qui rivalise avec d'autres réseaux sans fil à faible puissance tels que SIGFOX ou 4G/5G à bande étroite (NB).

Les circuits – modems LoRa de référence sont SX1276/78, SX1301 (multicanal), ..

Les circuits SX1276/78 sont intégrés avec les SoC ESP32 sur les cartes Heltec utilisés sur la plate-forme IoT DevKit.

LoRa – émetteur/récepteur

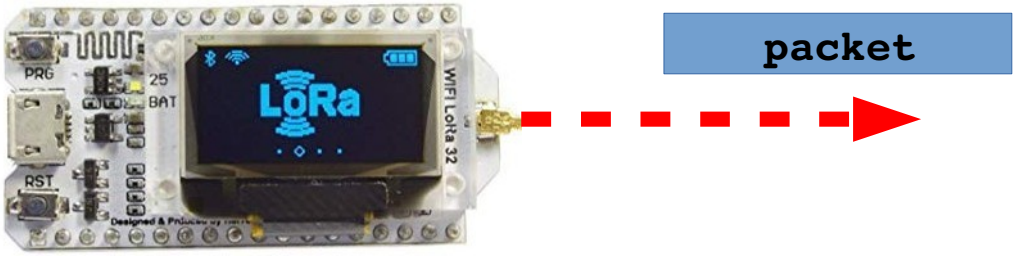
Préparations :

```
#include <SPI.h>                  // include libraries
#include <LoRa.h>
#define SCK     5    // GPIO5  -- SX127x's SCK
#define MISO    19   // GPIO19 -- SX127x's MISO
#define MOSI    27   // GPIO27 -- SX127x's MOSI
#define SS      18   // GPIO18 -- SX127x's CS
#define RST     14   // GPIO14 -- SX127x's RESET
#define DIO     26   // GPIO26 -- SX127x's IRQ(Interrupt Request)

#define freq    868E6
#define sf 8
#define sb 125E3
```

LoRa – émetteur/récepteur


```
union pack
{
    uint8_t frame[16]; // trames avec octets
    float  data[4];   // 4 valeurs en virgule flottante
} sdp ;  // paquet d'emission


void setup() {
  Serial.begin(9600);
  pinMode(DIO, INPUT);  // signal interruption
  SPI.begin(SCK,MISO,MOSI,SS);
  LoRa.setPins(SS,RST,DIO);
  if (!LoRa.begin(freq)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
LoRa.setSpreadingFactor(sf);
LoRa.setSignalBandwidth (sb);
}
```

LoRa – émetteur



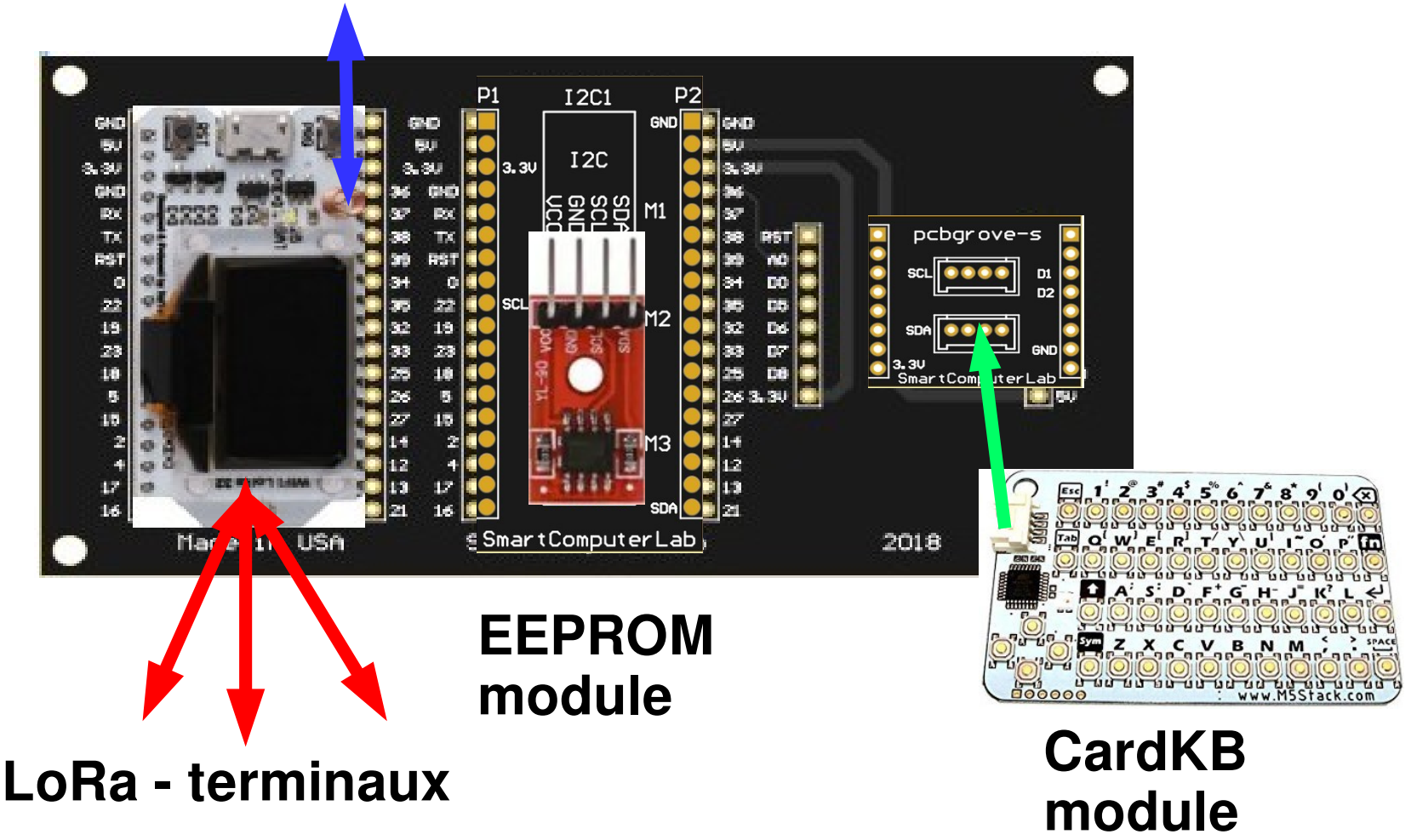

```
float d1=12.0, d2=321.54 ;

void loop()  // la boucle de l'emetteur
{
Serial.println("New Packet") ;
  LoRa.beginPacket();                    // start packet
  sdp.data[0]=d1;
  sdp.data[1]=d2;
  LoRa.write(sdp.frame,16);
  LoRa.endPacket();
d1++; d2+=2;
delay(2000);
}
```

LoRa – récepteur

packet

```
float d1,d2;
int i=0;

void loop()  // la boucle de recepteur
{
int packetLen;
packetLen=LoRa.parsePacket();
if(packetLen==16)
  {
    i=0;
  while (LoRa.available()) {
    rdp.frame[i]=LoRa.read();i++;
    }
  d1=rdp.data[0];d2=rdp.data[1];
  rssi=LoRa.packetRssi();  // force du signal en réception en dB
  Serial.println(d1);  Serial.println(d2);
  Serial.println(rssi);
  }
}
```

Une architecture IoT multitâche avec `freeRTOS` – exemple : passerelle LoRa-WiFi



Une architecture IoT multitâche avec freeRTOS – exemple : passerelle LoRa-WiFi

```
#include <Wire.h>      // bus I2C
#include "WiFi.h"
#include "ThingSpeak.h"
#include <U8x8lib.h>  // ecran OLED
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15, 4, 16);
#include <AT24Cxx.h>  // memoire EEPROM pour stocker les parametres
#define i2c_address 0x50  // adresse I2C pour le module EEPROM
uint16_t paddr=0;
AT24Cxx eep(i2c_address, 32);  // taille in KB
#define CARDKB_ADDR 0x5F // adresse I2C pour le clavier cardKB
char ssid[32],pass[32];  // parametres WiFi
WiFiClient  client;
#include <SPI.h>      // bus SPI
#include <LoRa.h>
#define SCK     5    // GPIO5  -- SX127x's SCK
#define MISO    19   // GPIO19 -- SX127x's MISO
#define MOSI    27   // GPIO27 -- SX127x's MOSI
#define SS      18   // GPIO18 -- SX127x's CS
#define RST     14   // GPIO14 -- SX127x's RESET
#define DIO     26   // GPIO26 -- SX127x's IRQ(Interrupt Request)
#define freq    434E6  // ou 868E6 en Europe
int  sf=8;           // facteur d'etalement
int sb=125E3;        // bande passante LoRa 125KHz,[250KHz,500Kz]
```

Une architecture IoT multitâche avec freeRTOS – exemple : passerelle LoRa-WiFi

```
union
  {
    uint8_t frame[24];
    struct
      {
        uint8_t head[4];
        float sens[5];
      } data;
  } pack;
int setwifi=0,setts=0;   // mode parametres
char preset[32];
int channel=1;  // thingspeak channel: 112
char chnum[32];
char wkey[32];  // thingspeak channel write key : "OWLWT3UPVO2KI818"

int weeprom(uint16_t addr, uint8_t *val, int len)
{
  for(int i=0;i<len;i++)  eep.write(addr++,val[i]);
}

int reeprom(uint16_t addr, uint8_t *val, int len)
{
  for(int i=0;i<len;i++)  val[i] = eep.read(addr++);
}
```

Une architecture IoT multitâche avec freeRTOS – exemple : passerelle LoRa-WiFi

```
int getstr(char *p, int len)
{
int i=0;  memset(p,0x00,len);
while(1)
 {
  Wire.requestFrom(CARDKB_ADDR, 1);
  while(Wire.available())
  {
    char c = Wire.read(); // receive a byte as character
    if (c != 0)
    {
      Serial.println(c, HEX);Serial.println(c);
      if(c==0x08 && i>0) { i--;}
      else
        {
        if(c=='\n' || c=='\r'){u8x8.clear();i=0;return(0);}
        else {p[i]=c;u8x8.drawString(0,i/16,p+16*(i/16));i++;}
        if(i==len) {u8x8.clear();i=0;memset(p,0x00,len);return(-1); }
        }
     }
  }
   delay(10);
 }
}
```

Une architecture IoT multitâche avec freeRTOS – exemple : passerelle LoRa-WiFi

```
void taskTS( void * parameter)
{
int i=0,packetSize=0;
char dbuff[32];
while(1)  // une tache s'execute en boucle !
   {
   while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
      }

    pack.data.sens[0]= (float) i++;
    ThingSpeak.setField(1, pack.data.sens[0]);

    Serial.println(channel);   Serial.println(wkey);
    ThingSpeak.writeFields(channel, wkey);
    delay(10000);
    }
}
```

Une architecture IoT multitâche avec freeRTOS – exemple : passerelle LoRa-WiFi

```
void taskTS( void * parameter)
{
int i=0,packetSize=0;char dbuff[32];
while(1)
  { packetSize = LoRa.parsePacket();
  if (packetSize==24)
    { i=0;
    while (LoRa.available())
    { pack.frame[i]=LoRa.read();i++; }
   while (WiFi.status() != WL_CONNECTED) {
     delay(500);Serial.print(".");
     }
   ThingSpeak.setField(1, pack.data.sens[0]);
   ThingSpeak.setField(2, pack.data.sens[1]);
   ThingSpeak.setField(3, pack.data.sens[2]);
   ThingSpeak.setField(4, pack.data.sens[3]);
   ThingSpeak.setField(5, pack.data.sens[4]);
   ThingSpeak.writeFields(channel, wkey);
   delay(10000); }
 }
}
```

Une architecture IoT multitâche avec freeRTOS – exemple : passerelle LoRa-WiFi

```
void setup()
{
char dbuf[32]; int cn=0;
  Serial.begin(9600);Wire.begin(21,22); ..
  SPI.begin(SCK,MISO,MOSI,SS);LoRa.setPins(SS,RST,DI0);
  if (!LoRa.begin(434E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  WiFi.disconnect(true);
   delay(100);
   WiFi.begin(ssid, pass);
    while (WiFi.status() != WL_CONNECTED) { delay(500); Serial.print(".");
    }
  IPAddress ip = WiFi.localIP(); ..
  sprintf(dbuf,"%d.%d.%d.%d",ip[0],ip[1],ip[2],ip[3]);
  u8x8.drawString(0,3,dbuf); ..
  ThingSpeak.begin(client);
  delay(1000);
  Serial.println("ThingSpeak begin");
    xTaskCreate(
                    taskTS,          /* Task function. */
                    "taskTS",        /* String with name of task. */
                    10000,            /* Stack size in words. */
                    NULL,             /* Parameter passed as input of the task */
                    0,                /* Priority of the task. */
                    NULL);
}

void loop(){ Serial.println("in the loop");delay(3000);}
```

IoT DevKit et architectures IoT

Architectures des terminaux IoT

- Terminaux simples – mono-tâche : ESP12/ESP32
- Terminaux complexes – multi-tâches (**`freeRTOS`**) : ESP32

Architectures des passerelles IoT : ESP32

- Multi-tâches et configurables (**`freeRTOS`**)

Une vaste sélection des cartes et des capteurs/actionneurs permettant de construire une grande variété d'**architectures IoT façon LEGO**.

IoT DevKit et architectures IoT

Serveurs IoT

- **`ThingSpeak.fr`**
- **`Thinger.fr`**

Support pédagogique complet

Incluant :

- Les préparations du logiciel - scriptes Arduino et bibliothèques de composants
- Les exemples d'utilisation de l'ensemble des cartes et des capteurs/actionneurs
- Exemples d'architectures dans les différentes domaines d'application