Advanced Networking Architectures and Wireless Systems

Carlo Vallati
PostDoc Researcher@ University of Pisa
c.vallati@iet.unipi.it

Who's that guy??

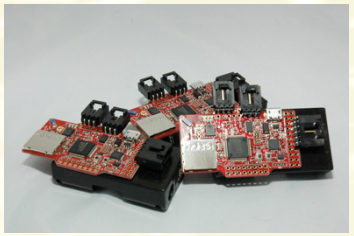
Carlo Vallati

c.vallati@iet.unipi.it

http://www.iet.unipi.it/c.vallati/

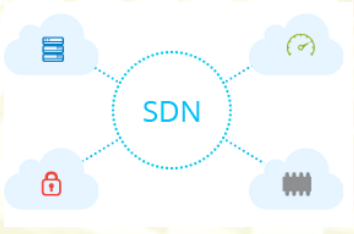
Hands-on labs

- Internet of Things labs

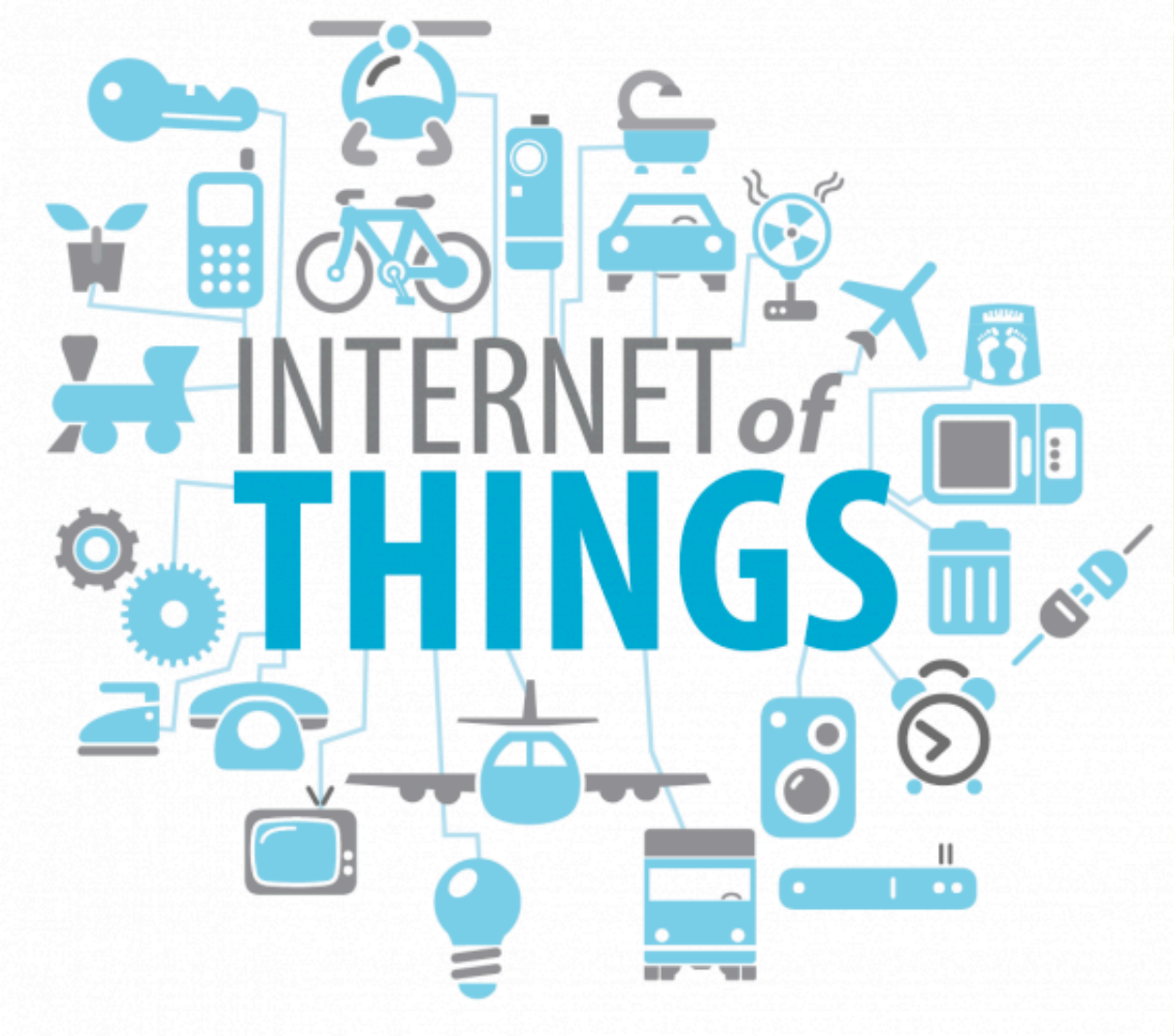


- Internet Backbone labs

- Software Defined Networking



Internet of Things Labs



Outline

- Introduction
- IPv6 configuration
- RPL configuration and settings
- CoAP

Zolerita Z1

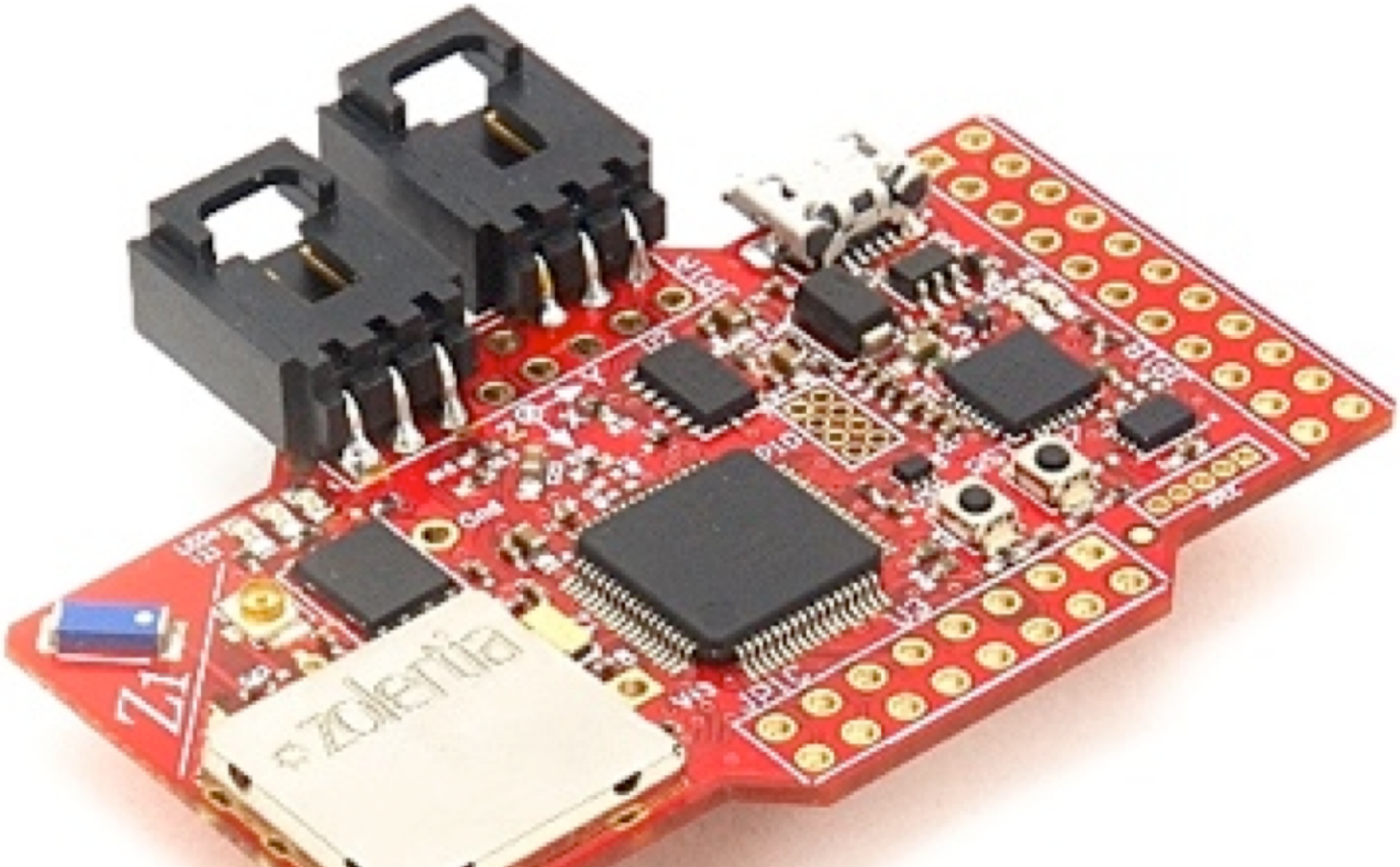
Carlo Vallati
PostDoc Researcher@ University of Pisa
c.vallati@iet.unipi.it

Z1

Low-Power WSN Platform

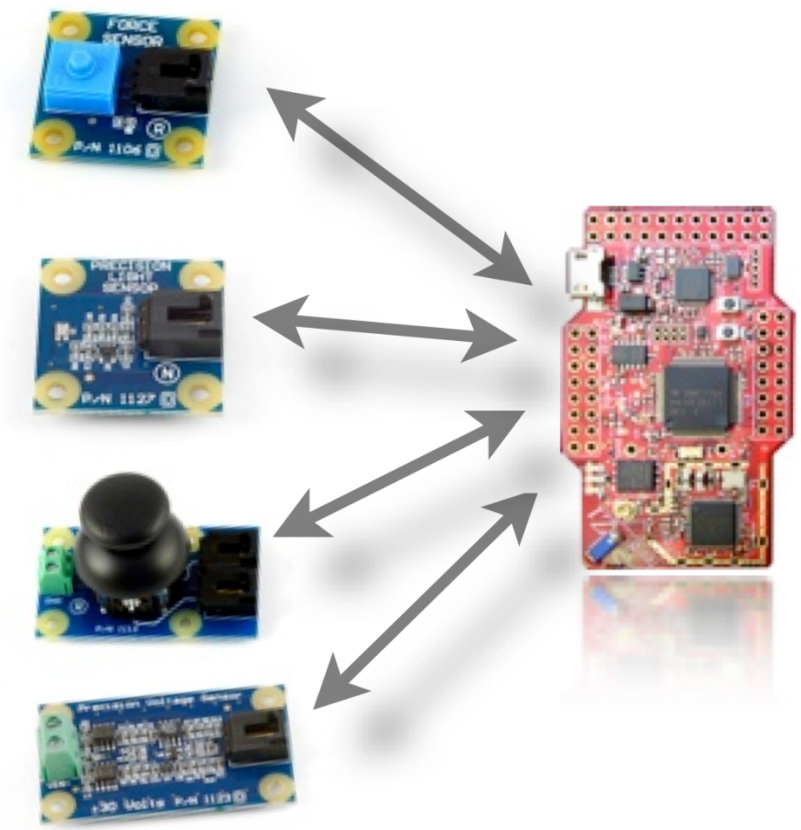
zolertia

Smarten it up



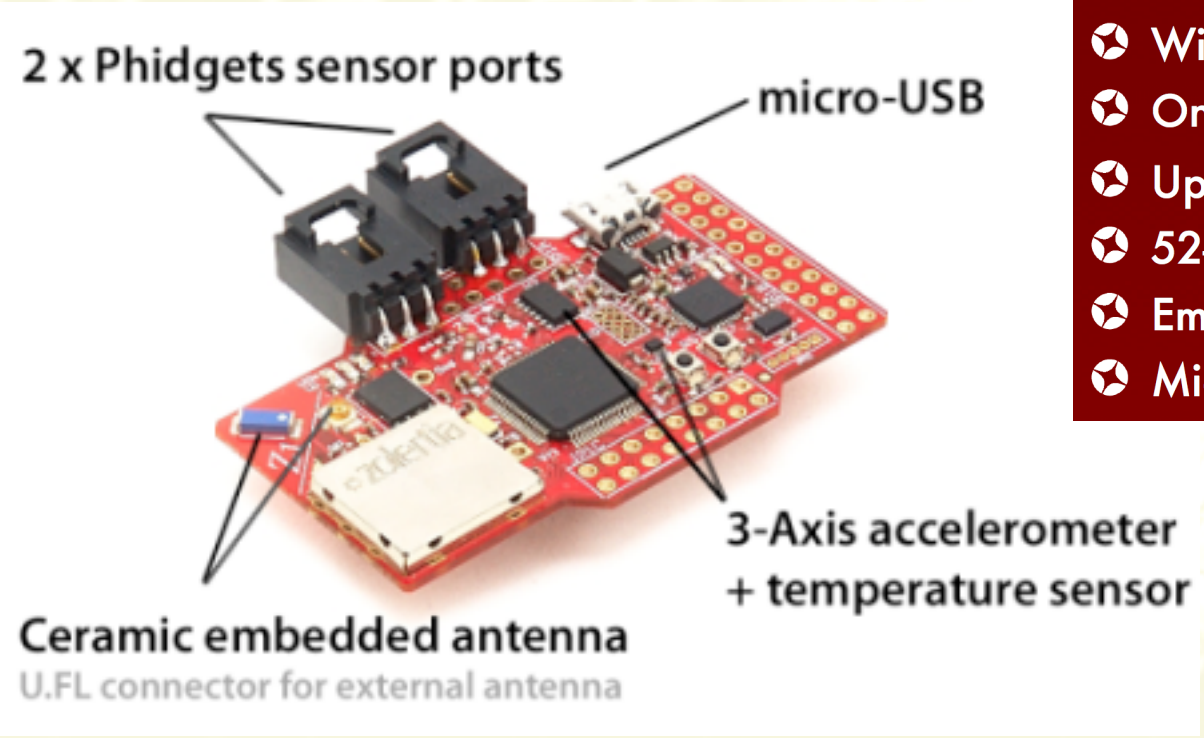
Zolertia Z1

- Out of the box support for Phidgets™



Zolertia Z1

Contiki and Tiny OS
support out of the box



Main Features

- 2.4GHz IEEE® 802.15.4 & 6LowPAN Compatible
- 2nd Generation MSP430(F2617)
- Widely Adopted Radio: CC2420
- On-board Digital Sensors (x2)
- Up to 4x Analog Phidgets™
- 52-pin Expansion Connector
- Embedded or External Antenna
- MicroUSB Connector

Instant Contiki 3.0

- Contiki 3.0 will be adopted as operating system
- Modified version of instant Contiki 3.0 will be provided in the VM that contains installed out of the box all the tools for the lab:
  - http://atlantis.iet.unipi.it/InstantContiki3.0.zip

Working folder

contiki/examples/z1

Get the list of motes connected:

make z1-motelist

Select permanently the architecture:

make TARGET=z1 savetarget

Load first program

Compile the program:

make MOTE=1 TARGET=z1 name-program

Load the program:

make MOTE=1 TARGET=z1 name-program.upload

In case of problems in loading the program:

chmod 777 /dev/ttyUSB0

Connect to the mote

To connect to the mote log:

make MOTE=1 login

To set the node id:

make burn-nodeid.upload nodeid=158 nodemac=158

Do it!

- Load test-adxl345 program to test 3-axis accelerometer (within the directory examples/z1)

Introduction to cooja

- Cooja is a java based emulator for contiki nodes
- The hardware a motes is emulated
- Wireless connection among motes is simulated

Contiki – OS basics

Carlo Vallati
PostDoc Researcher@ University of Pisa
c.vallati@iet.unipi.it

WSN Operating Systems

- The OS hides many HW details
  - Simplify the programmer life
- Contains drivers to radio and sensors, scheduling, network stacks, process & power management
- TinyOS, Contiki, FreeRTOS, Mantis OS

Contiki overview

- Contiki is a dynamic operating system for constrained devices
- Event driven kernel
  - Protothreds on top of it
- Support for many platform
  - Tmote Sky, Zolertia Z1, MicaZ …
- Support for many CPU

http://www.contiki-os.org/start.html

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

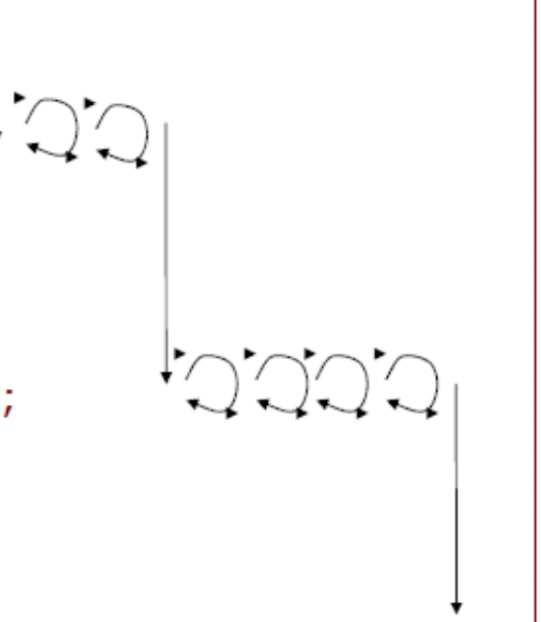
Event vs Thread

- Event driven OS: it only uses events
  - Difficult to program
  - No sequential flow of control
  - Low overhead
- **Threads**
  - **Easy to program**
  - **Sequential flow of control**
  - **High overhead (each thread has its own stack)**

Protothreads

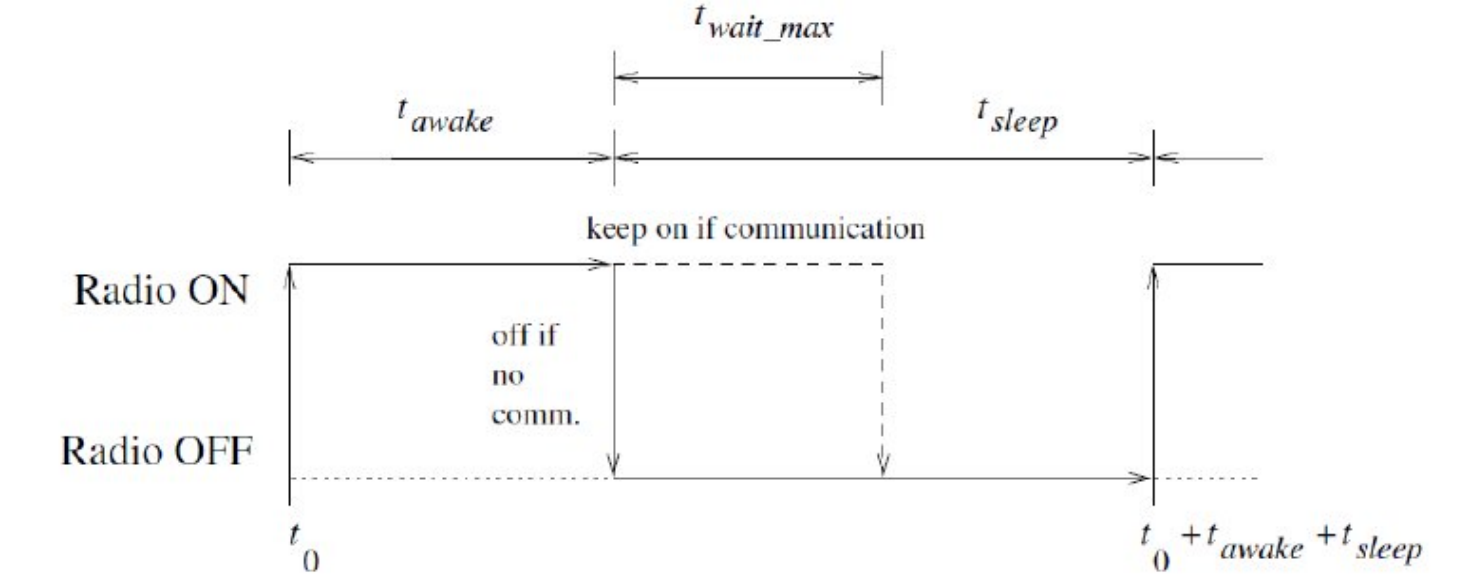
- Contiki implementation -> Protothreads
- Single stack: memory overhead is reduced (important on the memory-constrained systems on which Contiki runs.
- Sequential flow of control with conditional blocking

```
int a_protothread(struct pt *pt) {
  PT_BEGIN(pt);

  PT_WAIT_UNTIL(pt, condition1);

  if(something) {

    PT_WAIT_UNTIL(pt, condition2);

  }
  PT_END(pt);
}
```



Protothreads

- Example, simple MAC protocol:
  - Turn ON radio -> Wait for end of transmission or timeout -> Turn OFF radio



Protothreads

```
radio_wake_protothread:
  PT_BEGIN
  while (true)
    radio_on()
    timer ← t_awake
    PT_WAIT_UNTIL(expired(timer))
    timer ← t_sleep
    if (not communication_complete())
      wait_timer ← t_wait_max
      PT_WAIT_UNTIL(communication_complete() or
                    expired(wait_timer))
    radio_off()
    PT_WAIT_UNTIL(expired(timer))
  PT_END
```

Processes

- Processes are protothreads
- PROCESS_THREAD defines a new process
- Must start with PROCESS_BEGIN()
- Must end with PROCESS_END()
- Wait for new event:
  - PROCESS_WAIT_EVENT()
  - PROCESS_WAIT_EVENT_UNTIL(condition c)

Contiki directories

- core
  - System source code
- apps
  - System apps
- platform
  - Platform-specific code
    - Default mote configuration
- cpu
  - CPU-specific code
- example
  - Tons of examples. **USE** it as a starting point.
- tools
  - Cooja and other useful stuff

Hello World

```
#include “contiki.h”
#include <stdio.h>
/* Declare the process */
PROCESS(hello_world_process, “Hello world”);
/* Make the process start when the module is loaded */
AUTOSTART_PROCESSES(&hello_world_process);

/* Define the process code */
PROCESS_THREAD(hello_world_process, ev, data) {
        PROCESS_BEGIN(); /* Must always come first */
         printf(“Hello, world!\n”); /* code goes here. All printf must end with \n */
         PROCESS_END(); /* Must always come last */
}
```

Makefile

```
CONTIKI_PROJECT = hello-world

all: $(CONTIKI_PROJECT)

CONTIKI = /home/user/contiki

include $(CONTIKI)/Makefile.include
```

project-conf.h

- Used to override default configurations
- Add to Makefile

  CFLAGS += -DPROJECT_CONF_H=\"project-conf.h\“
- Example: change RDC protocol

  #define NETSTACK_CONF_RDC nullrdc_driver
- Change channel

  #undef RF_CHANNEL

  #define RF_CHANNEL 26
- See platform/z1/contiki-conf.h

Timers

- 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. Reserved for OS internals

POST and WAIT

- PROCESS_WAIT_EVENT();
  - Waits for an event to be posted to the process
- PROCESS_WAIT_EVENT_UNTIL(condition c);
  - Waits for an event to be posted to the process, with an extra condition. Often used: wait until timer has expired
  - PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
- PROCESS_POST(...) and PROCESS_POST_SYNCH(..)
  - Post (a)synchronous event to a process.
  - The other process usually waits with PROCESS_WAIT_EVENT_UNTIL(ev == EVENTNAME);

Sensors

```
#include “dev/button-sensor.h”
#include “dev/leds.h”

SENSORS_ACTIVATE(button_sensor);

PROCESS_WAIT_EVENT_UNTIL(ev==sensors_event &&
data==&button_sensor);

leds_toggle(LEDS_ALL);
```

See example/sky/test-button.c

Do IT!

- Write a program that loops indefinitely, check if a button has been pressed, and if so, toggles LEDs and prints out a message.

Timer functions

```
#include “sys/etimer.h”
static struct etimer et;
etimer_set(&et, CLOCK_SECOND*4);

PROCESS_WAIT_EVENT();
If(etimer_expired(&et)){
        etimer_reset(&et);
}
```

Do IT!

- Write a program that loops indefinitely, check if the timer has expired, and if so, toggles LEDs and prints out a message.
- Write a program that loops indefinitely, waits for an event, check if a button has been pressed, toggles LEDs and prints out “Button Press!”. If, instead, the timer has expired toggles LEDs and prints out “Timer!”