Introduction

Ramona Marfievici
(ramona.marfievici@cit.ie)

Some slides originally by
Gian Pietro Picco and Carlo Boano

Logistics

Software *will be provided*

- VMWare Player or VirtualBox if Windows OS
- VMWare Fusion if MacOS
- Virtual Machine
  - Instant Contiki: all necessary toolchains and software
  - https://sourceforge.net/projects/contiki/files/Instant%20Contiki/
  - **Username/password: user/user**

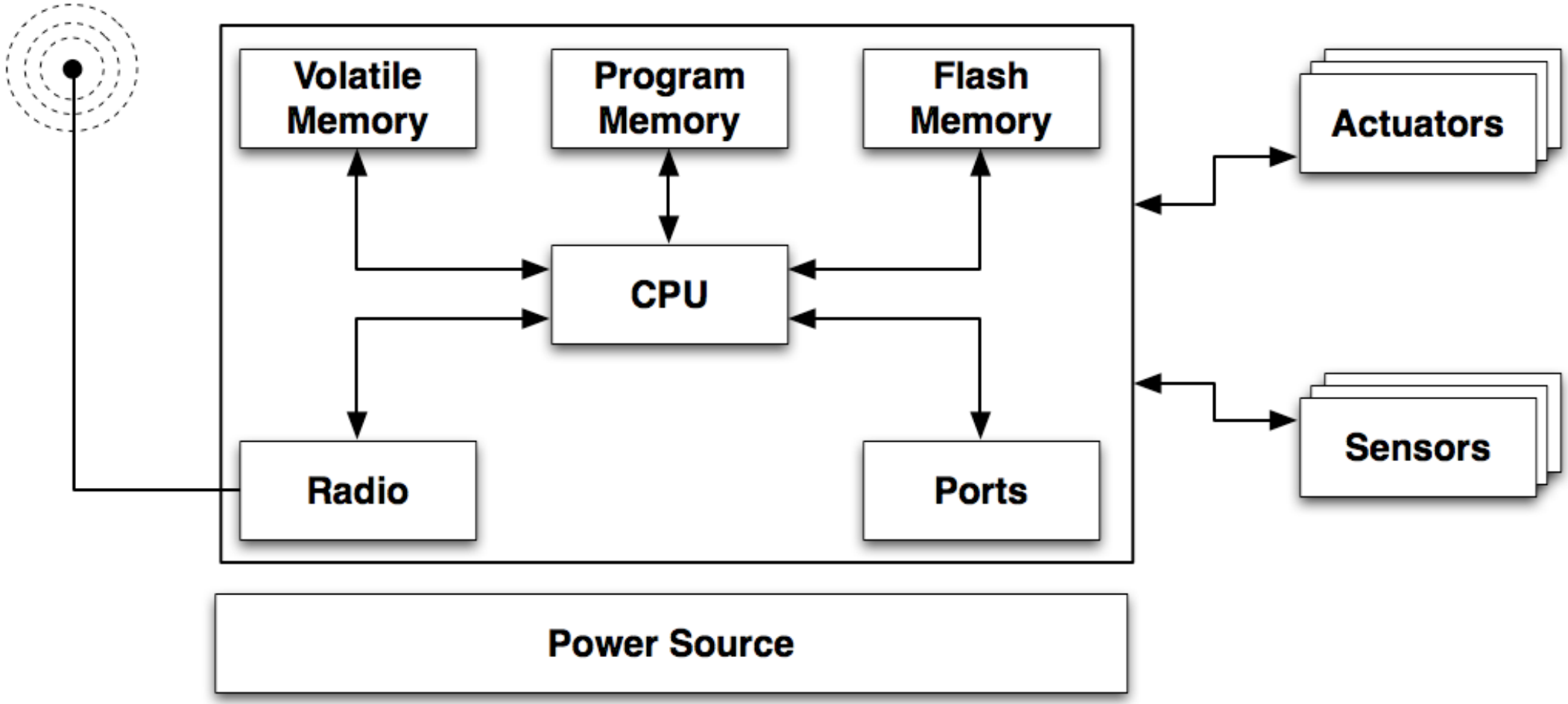
Hardware *will be provided*

- TelosB/TMoteSky mote

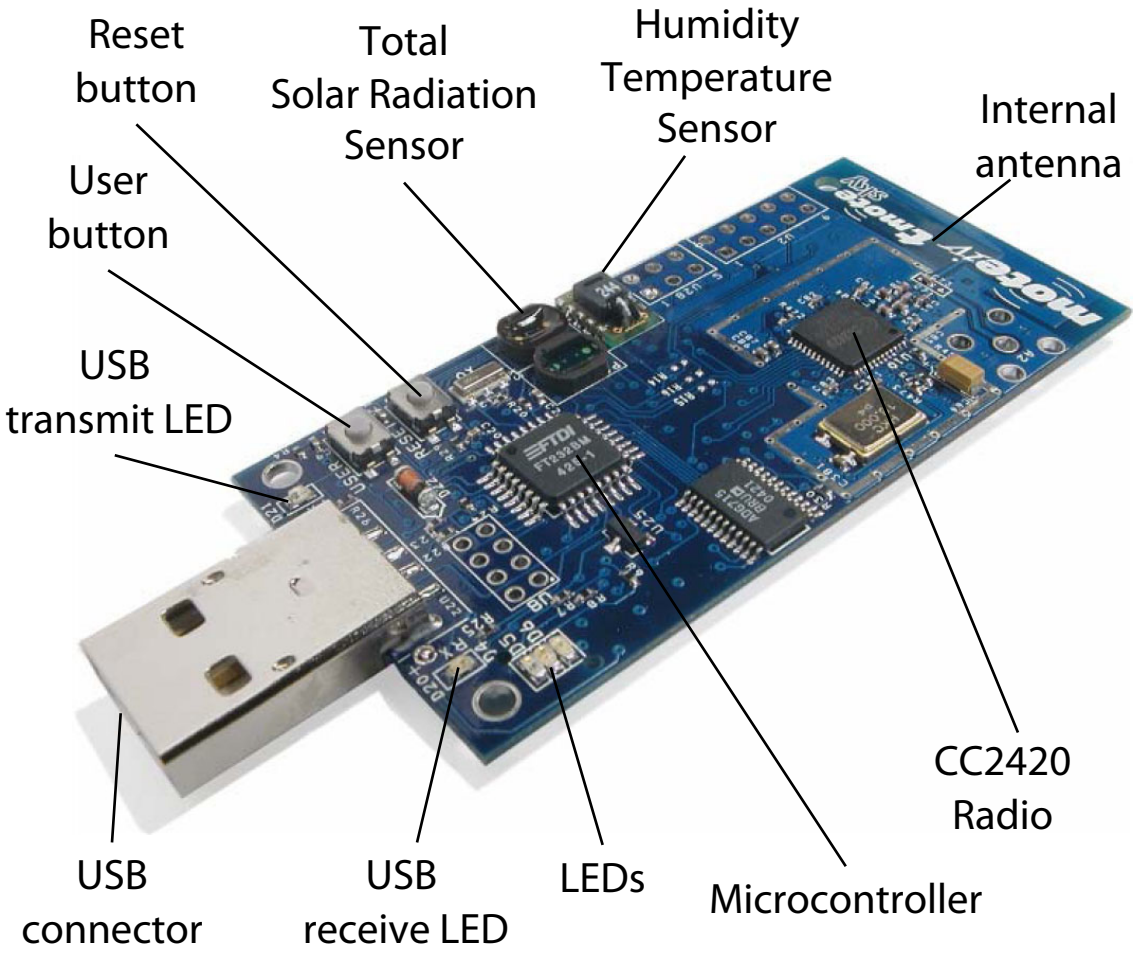
Important

- For each lab session use the virtual machine provided

Anatomy of a WSN Node



HW Platform for the Hands-on Lab

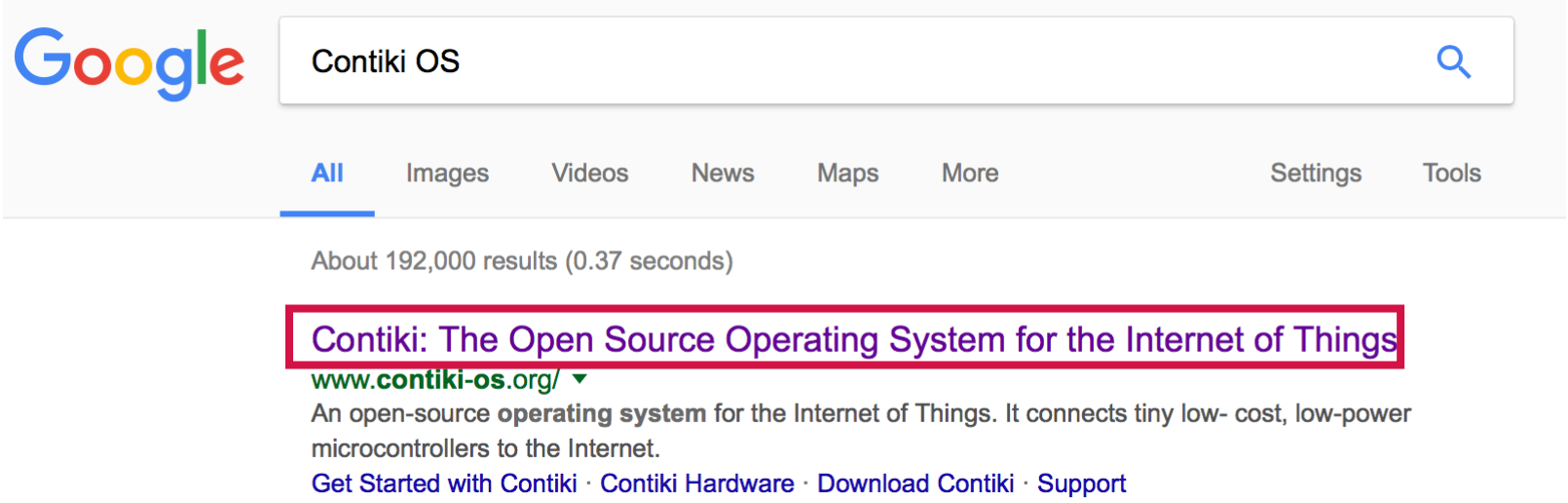


TelosB/TMoteSky

	TelosB/TMoteSky
CPU	TI MSP430
RF chip	CC2420 @2.4 GHz
Clock speed	8 MHz
RAM/code	10 kB/48 kB
External flash memory	1 MB
Build-in sensor	Temp, humid, light

How will we program this device?

Software Platform for the Hands-on Lab



- Open-source (BSD license)
- OS based on C (plus protothreads)
- Supports many embedded platforms
- Internet connectivity

Very first OS with Internet connectivity for platforms with limited resources

Contiki History

Name from “Kon-Tiki”, the rudimental raft that sailed the Pacific Ocean in 1947

http://en.wikipedia.org/wiki/Kon-Tiki

Movie

Kon-Tiki Museum, Olso, Norway

Contiki History

1947: Thor Heyerdahl and his crew surf the ocean using a rudimental raft

2001: Contiki is the first OS that can surf the Web with minimal resources

- Developed by Adam Dunkels at SICS, Sweden
- 2 kB RAM and 40 kB ROM
- C programming language
- Embeds µIP, world's smallest open-source, RFC-compliant TCP/IP stack

2003: Contiki v1.0 released to public

- Includes port to early WSN platforms

2003 – 2006: new features and support to many platforms

- 8, 16 and 32 bits MCUs
- Typically based on IEEE 802.15.4 and BLE radios
- Battery powered

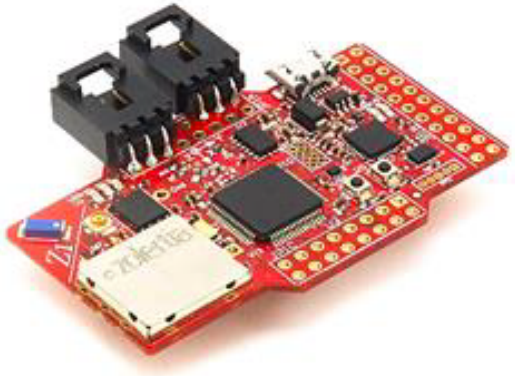
Contiki History

2003 – 2006: new features and support to many platforms

- 8, 16 and 32 bits MCUs
- Typically based on IEEE 802.15.4 and BLE radios
- Battery powered
- RAM 4-10s kB
- ROM 10-100s kB
- Sensors and actuators
- Full list at: http://www.contiki-os.org/hardware

MicaZ

TelosB/TmoteSky



Zolertia Z1

NXP/Jennic's JN5139/48

Contiki Today

Used both in Academia and Industry

- Lots of papers referring to it
- More and more commercial products
- Brings standards to the most constrained devices

Large community

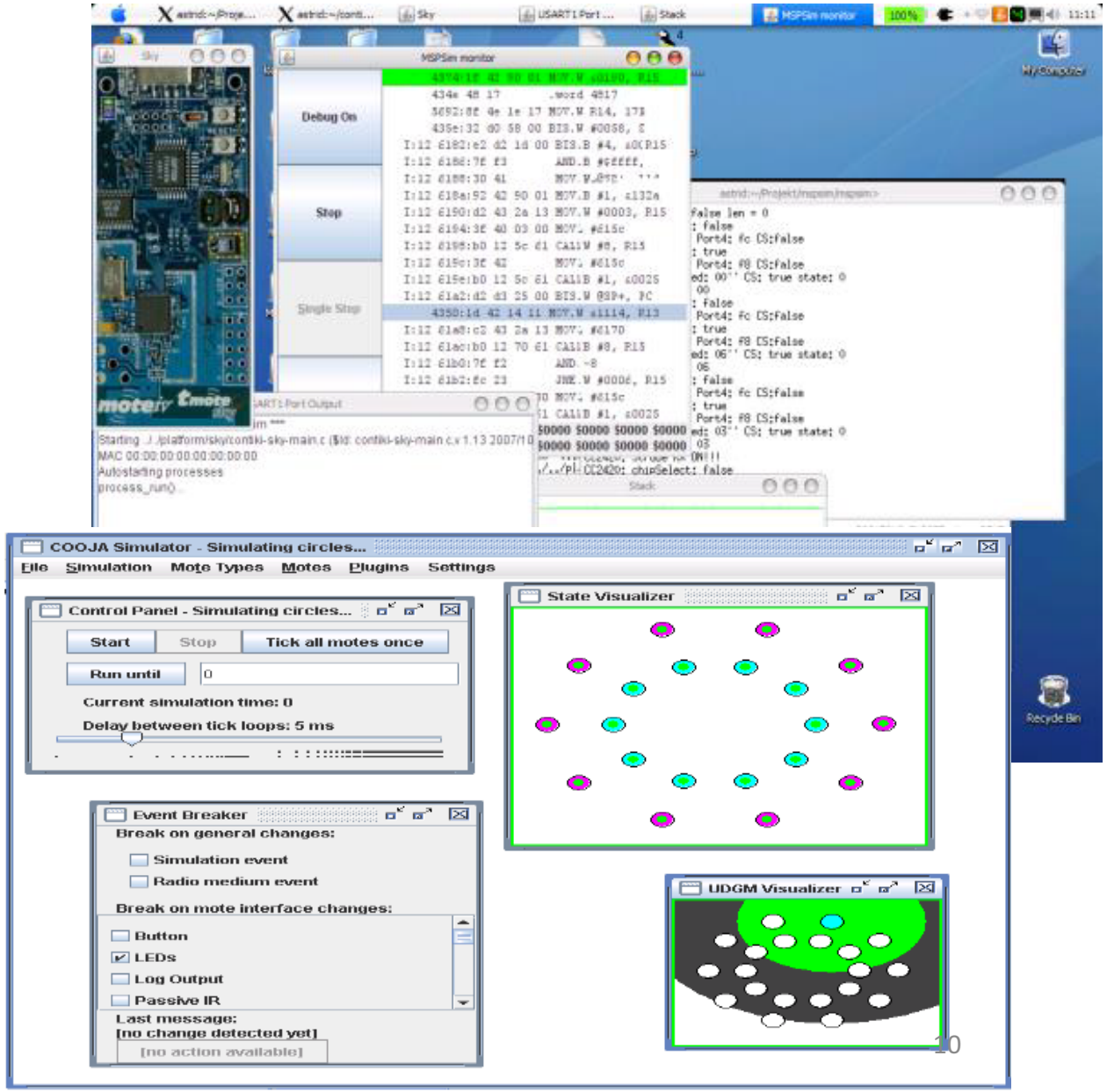
- Led by Adam Dunkels (Thingsquare, former SICS)
- 11 maintainers (Thingsquare, SICS, Bristol University, INRIA, Zolertia,…)
- 141 contributors, 2000 followers, many more users
- Contributions: https://github.com/contiki-os/contiki

Not the only IoT OS of this kind

Contiki Main Features

Among others

- Hybrid threading model (protothreads)
- IPv6/RIME network stacks
- Power profiling (Energest)
- File system (Coffee)
- DBMS (Antelope)
- Simulator (COOJA, MSPSim)



Contiki Main Features

IPv6 networking stack

- µIP: world's smallest, fully compliant TCP/IP stack
- Supports unicast, multicast, TCP, UDP, ICMP

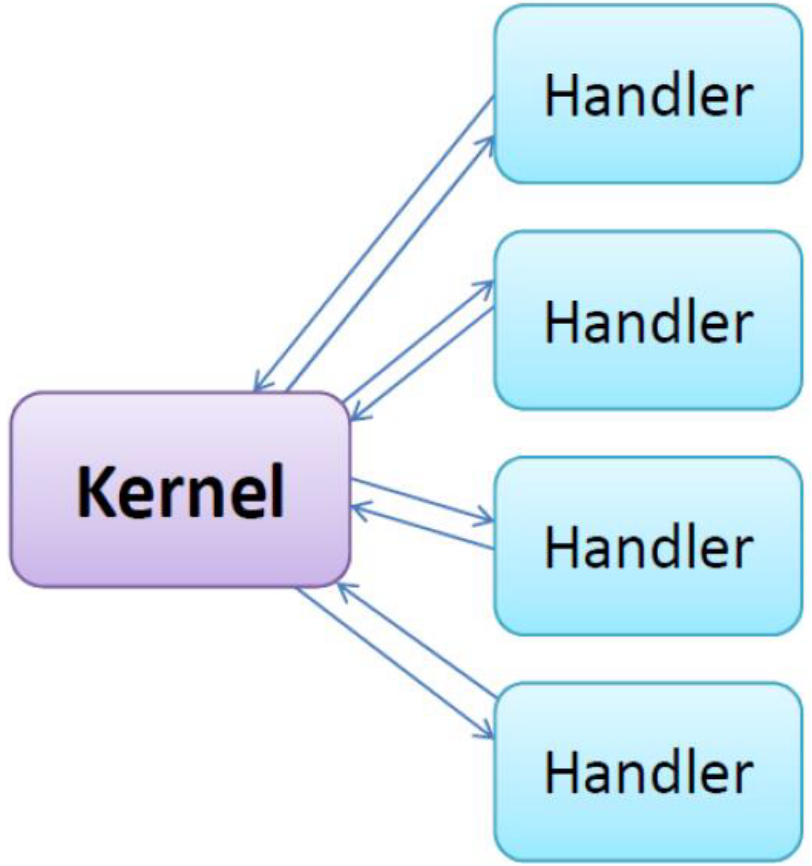
COOJA

- Cross level JAVA-based simulator
- Simplifies development and debugging
- Emulated different platforms
- Adding nodes, change TX power…
- Radio status, ongoing packets
- Serial output of motes (printf())

Contiki Kernel and Protothreads

The Contiki kernel is **event-based**

1. Wait for interrupt
2. Run interrupt handler
3. Call kernel and applications
4. Sleep

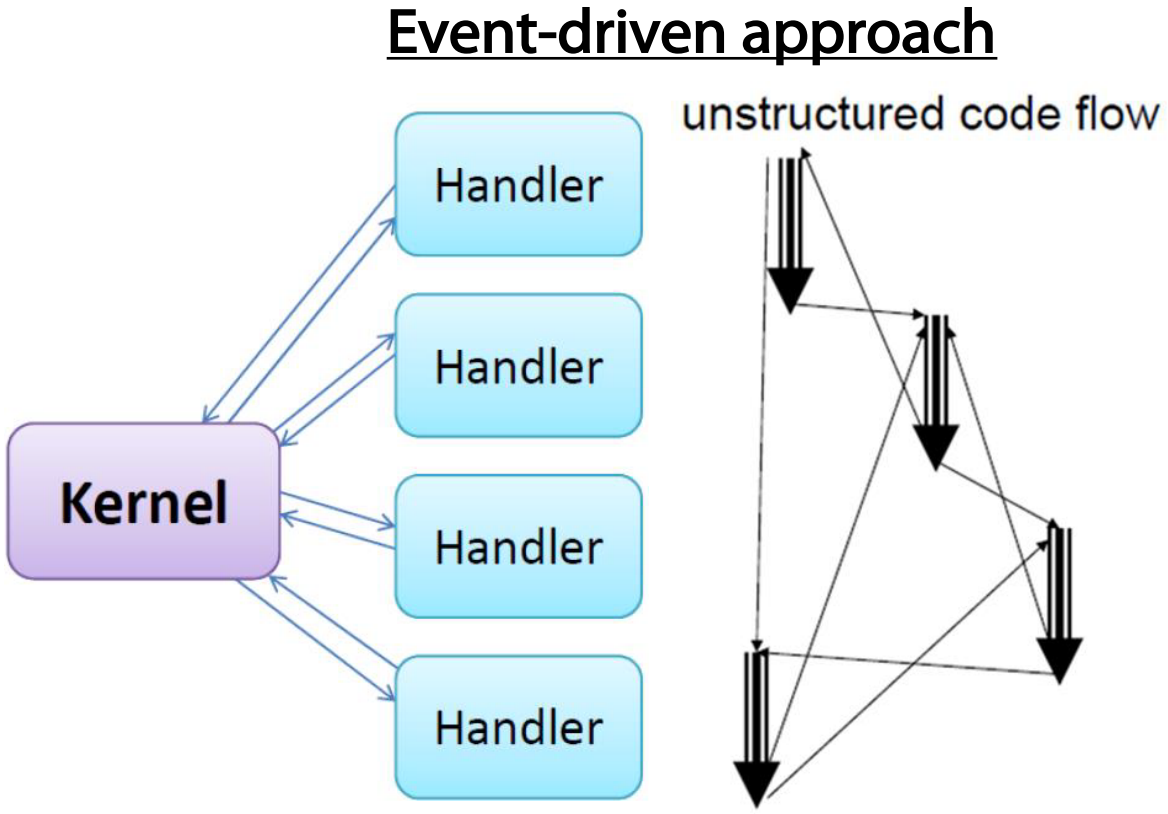


Three Types of events

1. Timer events
2. External events (e.g., hw interrupts)
3. Internal events (e.g., inter-process communication)

Contiki Kernel and Protothreads

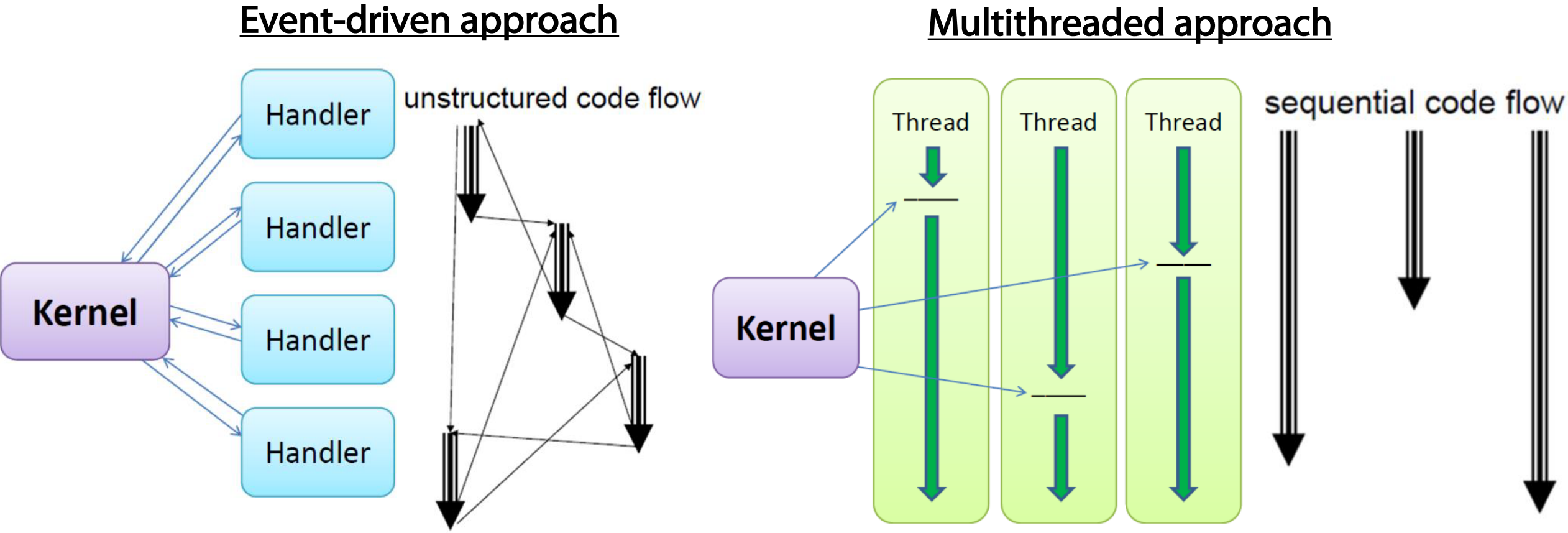
Event-driven programming is not straightforward!!!



- Not all programs easily expressible as state machines
- Many pitfalls (e.g., recursive callbacks!)
- Memory efficient (same stack for all processes)
- Low context switching overhead
- No preemption possible
- No need for locking mechanism

Contiki Kernel and Protothreads

Using threads would be much easier!



Contiki Kernel and Protothreads

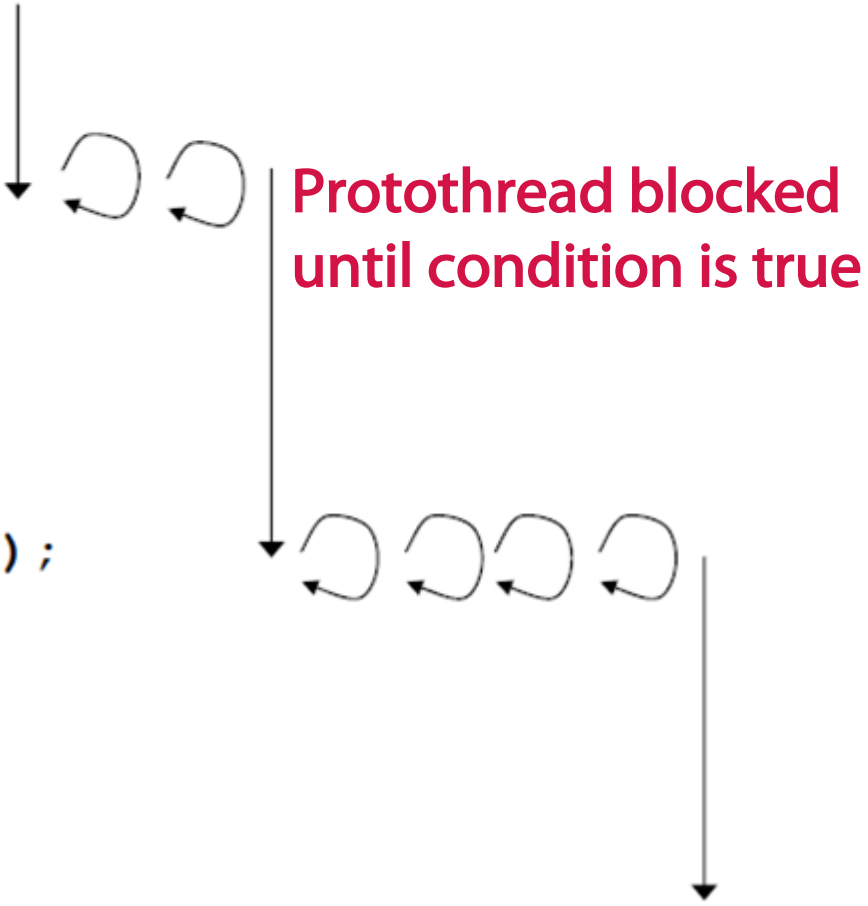
Protothreads provide sequential flow of control on top of an event-based kernel

- Programming abstractions combining the advantages of event-driven systems and multithread systems
- Significantly simplifying event-driven programming

Protothreads

Example: conditional blocking wait

```
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

Conditional blocking waits

- PT_WAIT_UNTIL (pt, condition*)
  - The protothread is blocked until the statement evaluates to true
- PT_WAIT_WHILE (pt, condition*)
  - The protothread is blocked while the statement evaluates to true

Yielding a protothread

- PT_YIELD (pt)
  - Yield the protothread, thereby allowing other processing to take place in the system

IMPORTANT!

- Stack info is lost across blocking calls
- Work around: use static rather than automatic variables

Contiki Processes

Contiki processes are implemented using protothreads

PROCESS_THREAD (name, events, data)

- Defines a new process

PROCESS_BEGIN() and PROCESS_END()

- Enclose the process in its declaration

PROCESS_WAIT_EVENT()

- Wait for a new event to be posted to the process

PROCESS_WAIT_EVENT_UNTIL (condition c)

- Wait for an event to be posted with an extra condition
- May be an hw interrupt (e.g., button) or an expired timer

PROCESS_YIELD ()

- Wait for a new event to be posted to the process

Example: Hello World!

Simple process that prints “Hello World!” to stdout

Define a new process

PROCESS_THREAD (name, event, data)

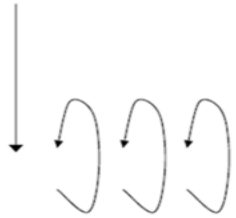
Enclose the process

PROCESS_BEGIN()
PROCESS_END()

Wait for an event to be posted to the process

PROCESS_WAIT_EVENT()

```
PROCESS_THREAD (hello_world, ev, data) {
  PROCESS_BEGIN();
        printf(“Hello World!\n”);
        while(1) {
              PROCESS_WAIT_EVENT();
        }
  PROCESS_END();
}
```



Waits forever!

Walk Through the Code

Base directory

- `core`: processes, protothreads, timers, file system, network stack (`core/net`)
- `cpu`: microcontroller specific code (e.g., msp430)

apps	Orchestra: avoid a link error for nbr_routes	17 days ago
core	Merge pull request #2073 from cetic/pr-fix-send-na	16 days ago
cpu	Revert to static Ethernet driver for the ATARI.	10 days ago
dev	Merge pull request #1957 from alignan/pull/bme280-sensor	2 months ago
doc	Update code-style.c	2 months ago
examples	Merge pull request #2073 from cetic/pr-fix-send-na	16 days ago
lib	Add support for the FAT file system	2 months ago
platform	Merge pull request #2023 from OpenMote/master	3 days ago
regression-tests	Merge pull request #2073 from cetic/pr-fix-send-na	16 days ago
tools	Revert to static Ethernet driver for the ATARI.	10 days ago
.gitattributes	Add binary files file extension	a year ago
.gitignore	Merge pull request #1469 from wbober/nrf52dk-pr	8 months ago
.gitmodules	Add sensniff as a submodule	2 months ago
.travis.yml	frame802154: comply with IEEE 802.15.4-2015 on PAN ID Field Handling	2 months ago
CONTRIBUTING.md	Updated CONTRIBUTING.md to reflect Contiki's new merging policy	2 years ago
LICENSE	Removed the explicit year 2012 to make it more generic	4 years ago
Makefile.include	Extended SOURCEDIRS variable with EXTRALDIRS variable in Makefile.inc...	a year ago

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

Walk Through the Code

TARGET=platform when compiling

Base directory

- *platform*: specific board drivers (e.g., c64, micaz, sky, z1)
- Each platform has a *contiki-conf.h* containint all definitions of Contiki settings for the specific platform and a contiki-main.c containing the main loop

Branch: master ▾ contiki / platform / sky / Create new file | Find file | History

laurentderu Fix semantic of UIP_ND6_SEND_NA and add UIP_ND6_SEND_NS — Latest commit 12c9308 20 days ago

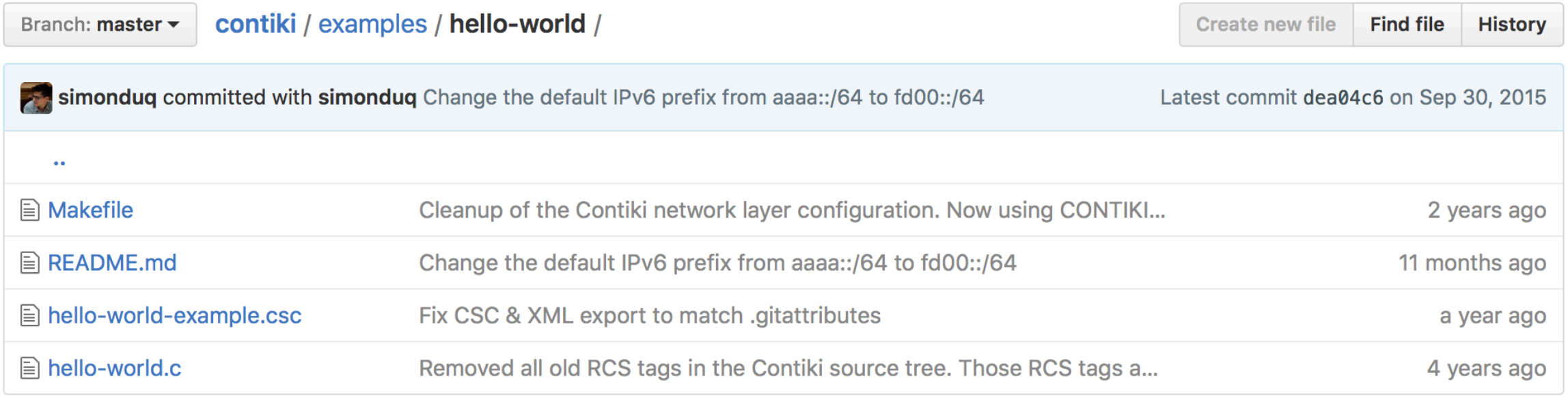
..		
apps	Cleanup of the node-id.h files. The node-id.h file contains	4 years ago
dev	Remove old unused light drivers	2 years ago
doc	Short description of the Tmote Sky platform	10 years ago
Makefile.common	Cleanup of the Contiki network layer configuration. Now using CONTIKI...	2 years ago
Makefile.sky	Cleanup of the Contiki network layer configuration. Now using CONTIKI...	2 years ago
cfs-coffee-arch.h	Make the I/O semantics functionality in Coffee unconditional so that ...	8 months ago
contiki-conf.h	Fix semantic of UIP_ND6_SEND_NA and add UIP_ND6_SEND_NS	20 days ago
contiki-sky-main.c	Change the default IPv6 prefix from aaaa::/64 to fd00::/64	11 months ago
contiki-sky-platform.c	Removed all old RCS tags in the Contiki source tree. Those RCS tags a...	4 years ago
node-id.c	Cleanup of the node-id.h files. The node-id.h file contains	4 years ago
platform-conf.h	Merge pull request #1375 from myrfy001/myrfy001-patch-1	11 months ago

https://github.com/contiki-os/contiki/tree/master/platform/sky

Walk Through the Code

Base directory

- `dev`: specific radios and sensors (e.g., cc2420, sht11)
- `tools`: bootstrap loaders, COOJA, MSPSim, serialdump, …
- `examples`: collection of example applications (e.g., `hello-world`)

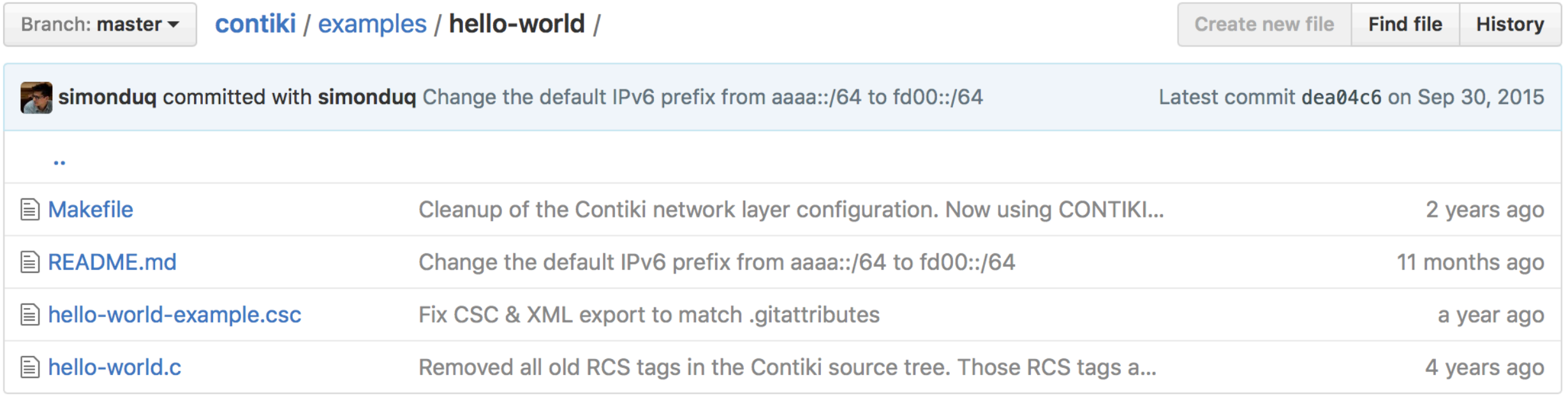


https://github.com/contiki-os/contiki/tree/master/examples/hello-world

Example: Hello World!

A Contiki application is typically a folder with

- C files with application code
- `Makefile`
- [optional] custom configuration file (`project-conf.h`)
- [optional] custom COOJA simulation file (`*.csc`)



https://github.com/contiki-os/contiki/tree/master/examples/hello-world

Example: Hello World! hello-world.c

```
#include "contiki.h" //Contiki core
#include <stdio.h>   //necessary for the printf

//declare the process
PROCESS (hello_world_proc, "Hello world process");
//make the process start when the module is loaded
AUTOSTART_PROCESSES (&hello_world_proc);

//define the process code
PROCESS_THREAD(hello_world_proc, ev, data)
{
      PROCESS_BEGIN();
      printf("Hello World!\n");
      PROCESS_END();
}
```

PROCESS_BEGIN and PROCESS_END always delimit a process!!!

Example: Hello World! Makefile

```
//name of the c file containing the main application
CONTIKI_PROJECT = hello-world

//what to compile
all: $(CONTIKI_PROJECT)

//path to main Contiki folder
CONTIKI = ../..

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

Compiling and running

Application code is ready! What's next?

Verify that your system works

- `$ make TARGET=native`
- `$ ./hello-world.native`

TelosB/TmoteSky

Compile and run for a specific target platform

- Same name as in `contiki/platform`
- `$ make hello-world.upload TARGET=sky`
- Connect the TmoteSky to the USB port
- `$ make TARGET=sky savetarget`
- `$ make hello-world.upload [MOTE=xxx]`
- `$ make login [MOTE=xxx]`

Target platform is usually embedded in the `Makefile`

```
ifndef TARGET
  TARGET=sky
endif
```

- `$ make sky-motelist` `//finds all connected sky nodes and their associated tty file`

Compiling and running

Simulate application in COOJA

- Emulates TMoteSky and other nodes
- The code is exactly the same firmware you upload on physical nodes
- `$ make TARGET=cooja CONTIKI-PROJECT`
  - Opens a new simulation of current project
- `$ make TARGET=cooja FILENAME.csc`
  - Opens an existing simulation file

Running in Cooja

- Start Cooja
  - `$ contiki/tools/cooja` and `ant run`
- Create new simulation
  - `File -> New Simulation -> Create new simulation dialog`
- Panels
  - Network: shows the nodes
  - Timeline: shows communication events
  - Mote output: serial port printout
  - Simulation control: start, pause, reload simulation)
- Add nodes to simulation
  - `Motes -> Add motes -> Create new mote type`
- Add code: `Browse` and select `hello-world.c`
- `Compile, Create, Start`

To Do List

- Verify that your system works using `hello-world` example
  - Note1. The program should print “Hello World!” on the screen and then appear to hang. In reality, Contiki is still running correctly but will not produce any more output because the example program has finished. Press Ctrl+C to quit.
  - Note2. Contiki uses `printf` to issue logs and messages. This can also be used when you have a node connected via USB.
- Run `hello-world` in Cooja
- Run `hello-world` on a TMoteSky node
  - Do not forget to press the reset button on the node

Tips and Tricks

- Check: device (e.g., TMoteSky node) IS connected to the Virtual Machine
- Check: user has permissions to USB ports
  - Add your user to the dialout group so you have appropriate permissions
    - navigate to `/etc/` folder and edit the `group` file add your username like this `dialout:x:20:USER`
  - OR `$ sudo chown <username> <port>`
- In case of problems with `make login`
  - Download the new serialdump-linux file from the webpage: https://github.com/cmorty/contiki/blob/pull/serialdump/tools/sky/serialdump-linux
  - Go to directory: `contiki/tools/sky`
  - Rename the current serialdump-linux with serialdump-linux.bck
  - Copy the new serialdump-linux file in the directory
  - Change the permissions: `$ chmod 775 serialdump-linux`
  - Install packages (might be needed☺)

    `$ sudo apt-get install lib32bz2-1.0`

    OR

    `$ sudo apt-get install lib32z1 lib32ncurses5 lib32stdc++6`

References

Contiki

- A lightweight and flexible operating system for tiny networked devices (http://www.dunkels.com/adam/dunkels04contiki.pdf)
- Official website: http://www.contiki-os.org

Protothreads

- Using protothreads for sensor network programming (http://compilers.cs.ucla.edu/emsoft05/DunkelsSchmidtVoigt05.pdf)
- Protothreads: Simplifying event-driven programming of memory-constrained embedded systems (http://dunkels.com/adam/dunkels06protothreads.pdf)