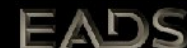




Performing cognitive radio experiments on the
LOG-a-TEC sensor network testbed

Tomaž Šolc
tomaz.solc@ijs.si

CREW Training Days
20 February 2013



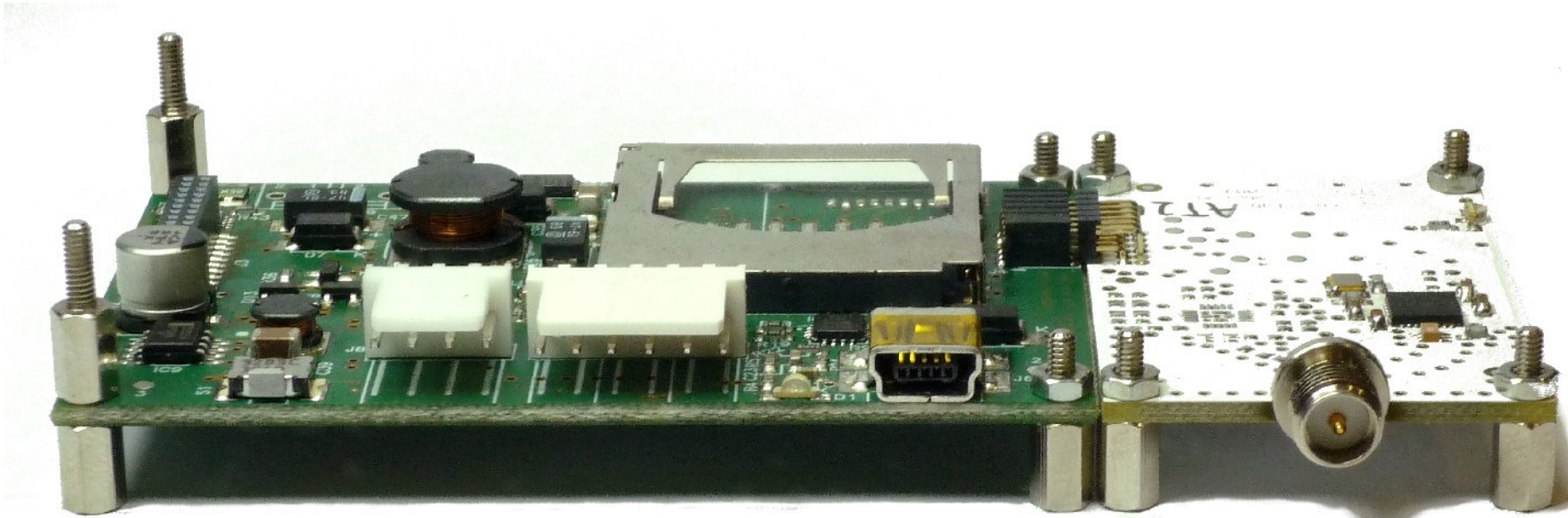
The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n°258301 (CREW project).

Overview

- Overview of the VESNA platform
 - hardware
 - software stack
- Overview of Log-a-tec testbed
 - how remote access works
- Building a basic experiment with Python
 - required software
 - step-by-step demonstration
- Conclusion and further references

VESNA

- “VErsatile platform for Sensor Network Applications”



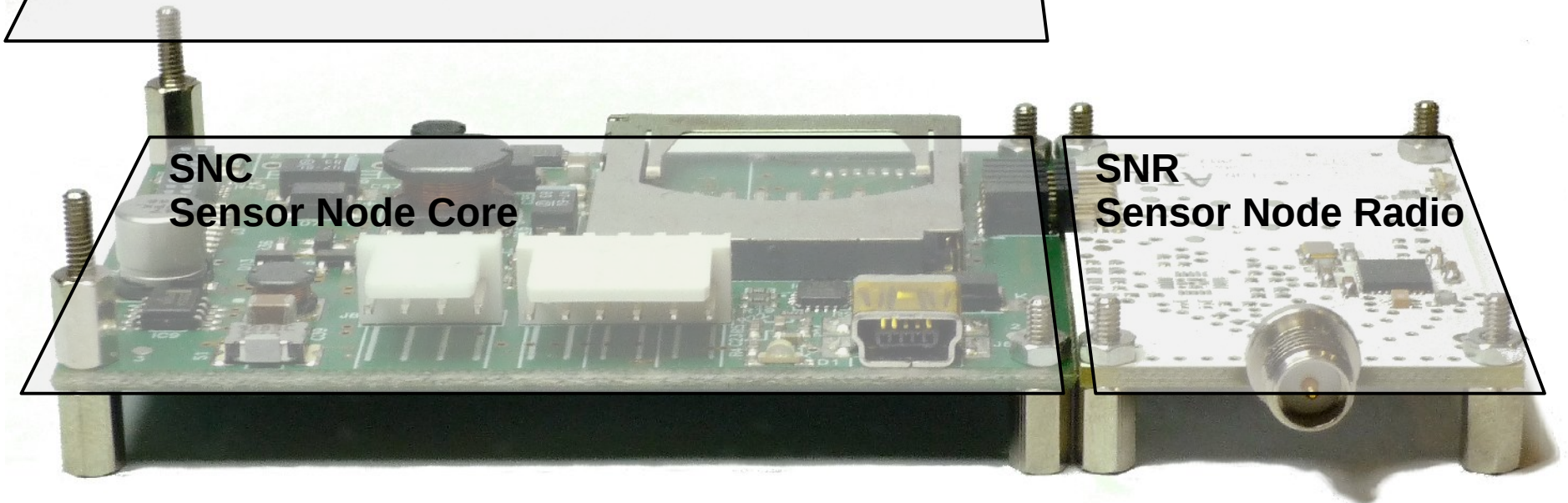
VESNA

- “VErsatile platform for Sensor Network Applications”

SNE
Sensor Node Expansion

SNC
Sensor Node Core

SNR
Sensor Node Radio

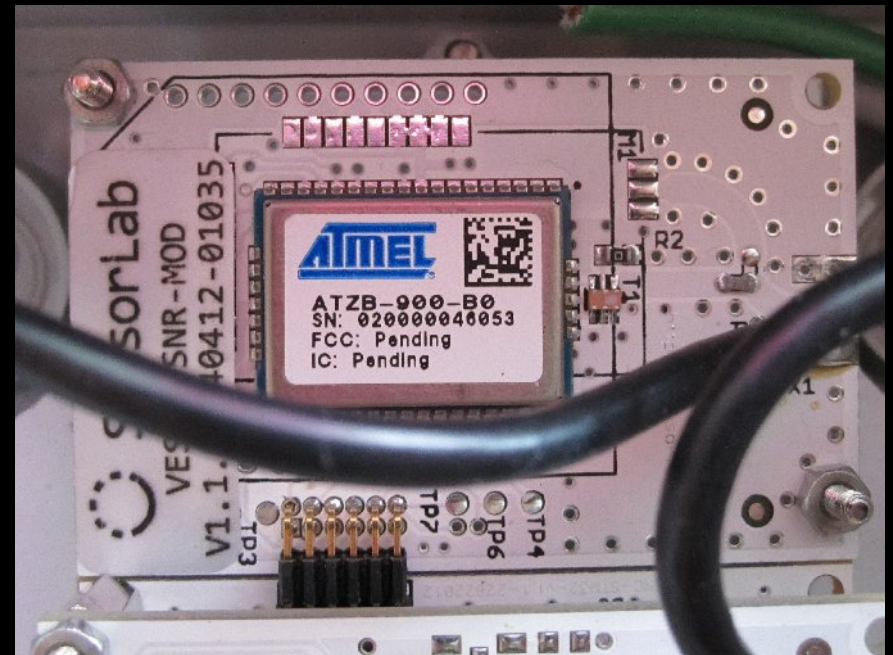


Sensor Node Core

- STM ARMv7 Cortex-M3
 - up to 72 MHz clock, 1 MB Flash, 96 kB RAM
- 3 x 1 MS/s ADC, w/ instrumental amplifier
- Non-volatile storage
 - 128 kB fast MRAM
 - SD or microSD
- Multi-purpose power supply
 - rechargeable, non-rechargeable battery, solar cell, external power
- USB 2.0, RS-232, I2C, SPI, UART, ...

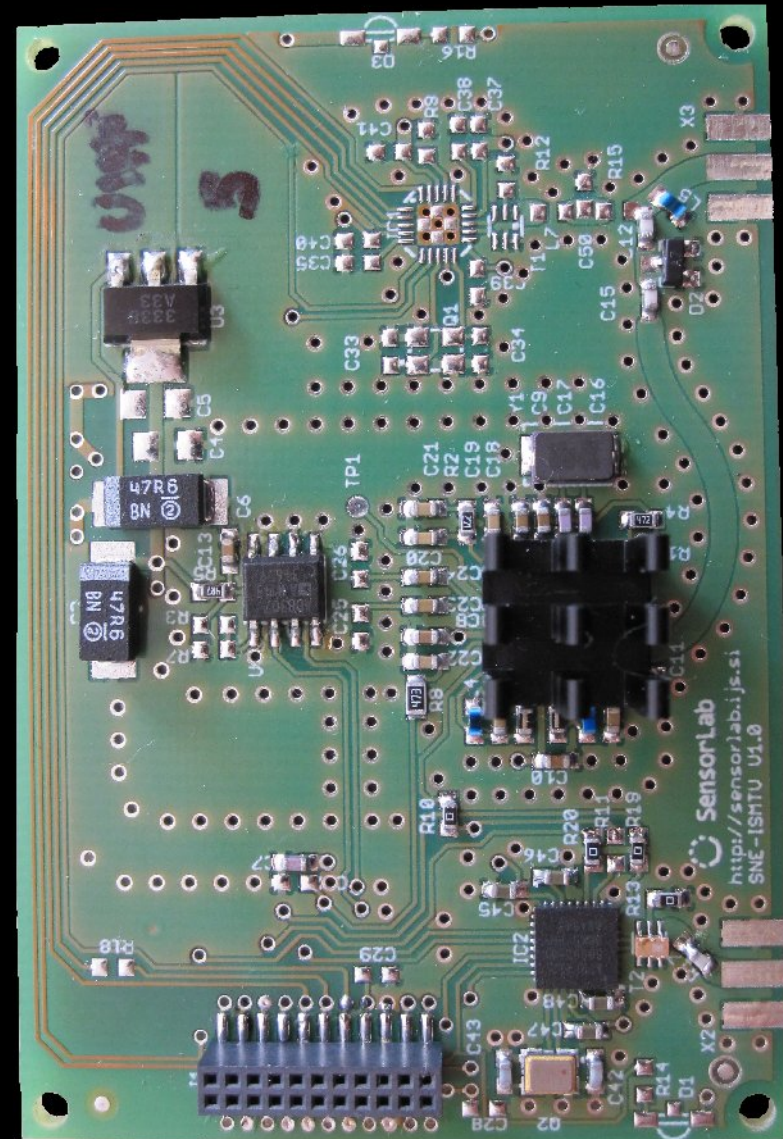
Sensor Node Radio

- Connect nodes into a wireless mesh network
 - 868 MHz European SRD band
 - 2.4 GHz ISM band
- IEEE 802.15.4
 - Atmel proprietary BitCloud / ZigBit / SerialNet
 - 6lowPAN
- Sensor node control and management



Sensor Node Expansion

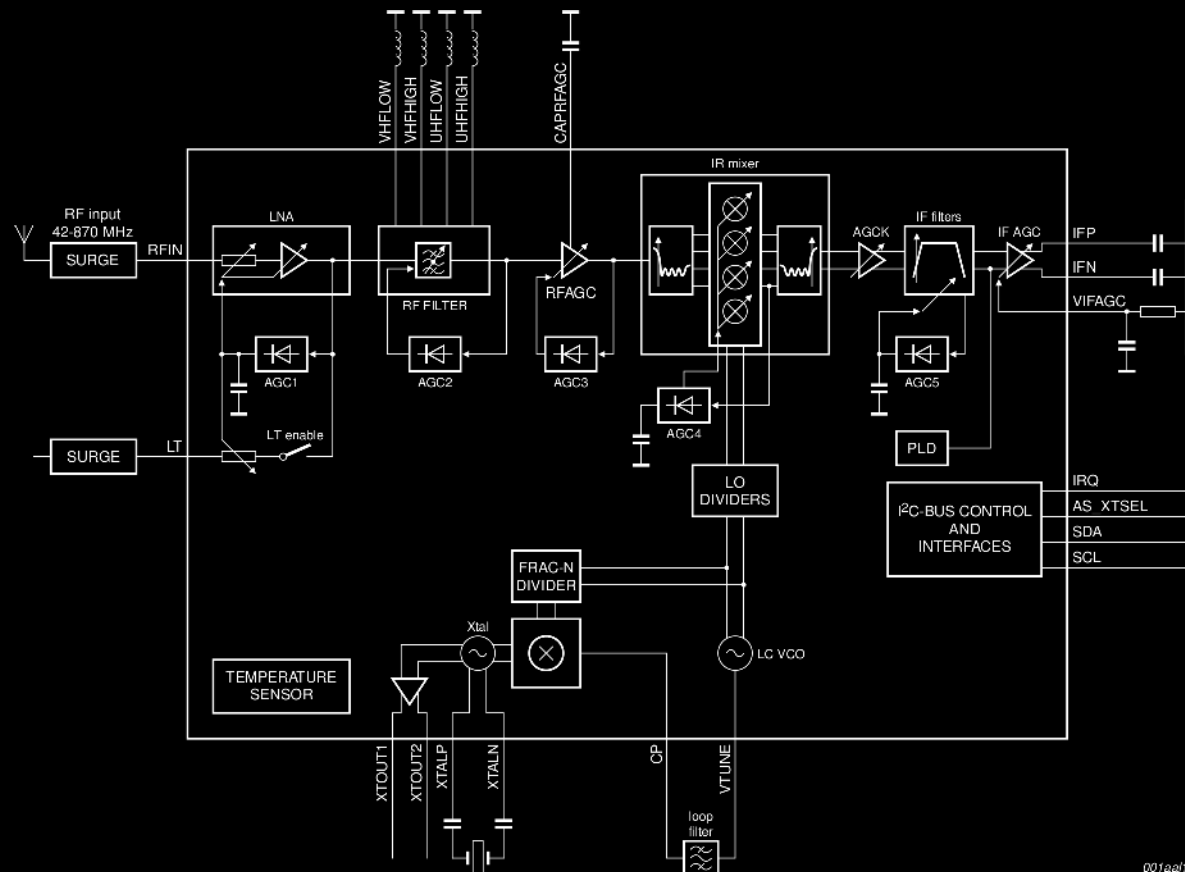
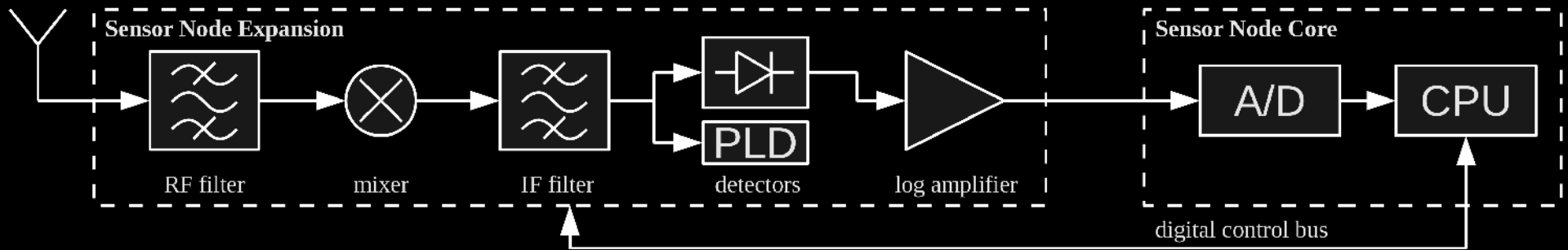
- SNE connector allows for application specific expansions
 - Data acquisition,
 - sensor interfaces,
 - wired/wireless communications,
 - extra power supply,
 - etc.



SNE-ISMTV

- Designed for spectrum sensing and cognitive radio applications
- Collection of radio-frequency hardware
 - UHF/VHF wide-band energy detection receiver
 - reconfigurable narrow-band sub-1 GHz transceiver
 - reconfigurable narrow-band 2.4 GHz transceiver
 - additional IEEE 802.15.4 radio (868 MHz)
- Independent of the testbed management network

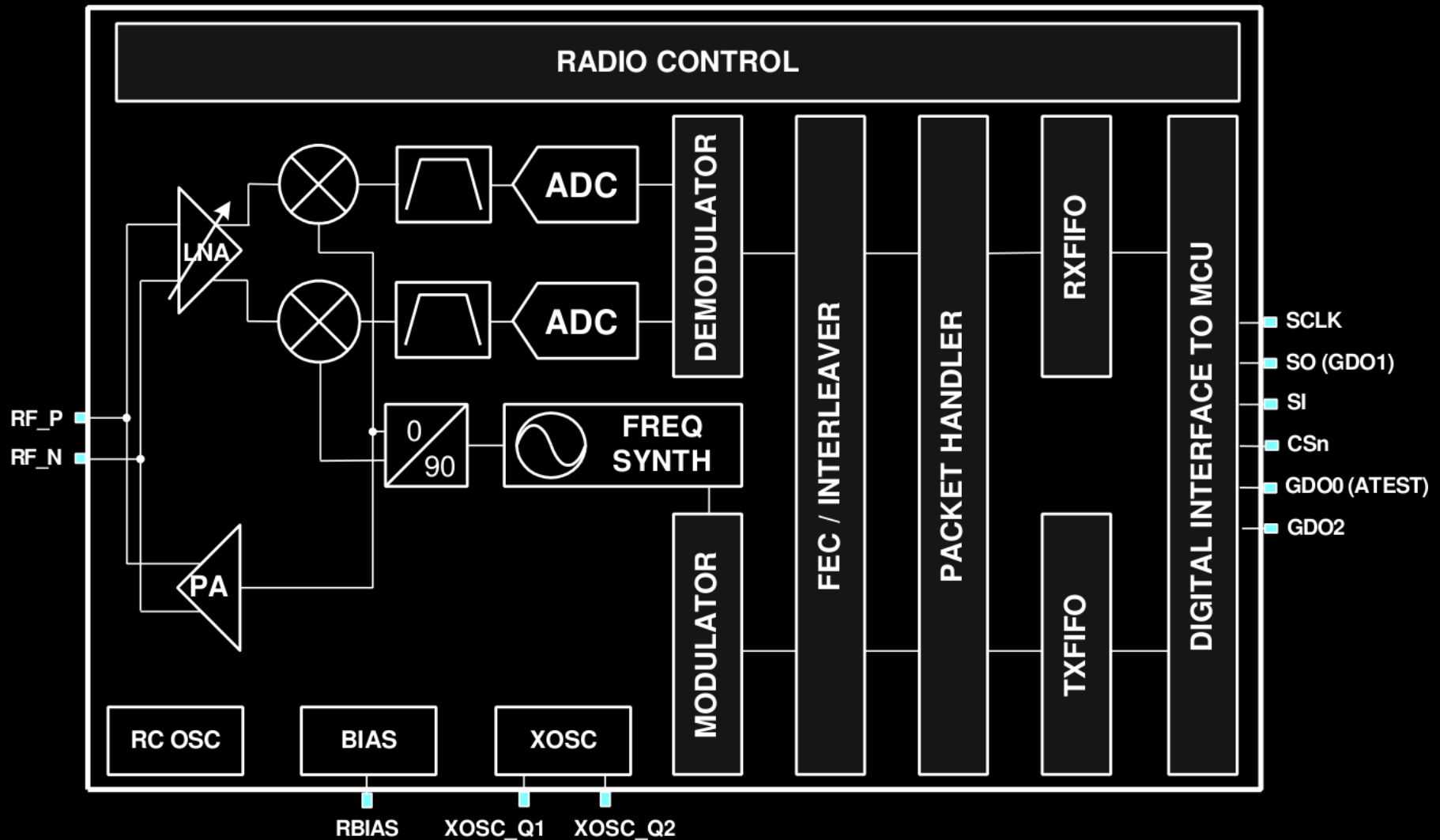
SNE-ISMTV-UHF



SNE-ISMTV-UHF

- Based on **NXP TDA18219** silicon tuner
- Frequency range 470 - 862 MHz
- Bandwidth 1.7 MHz, 8 MHz
- Power detector uncertainty 1.8 dBm
- Detector read-out time 50 ms / 1 μ s
- Average noise level -169 dBm (@ 1Hz)
- Dynamic range 60 dBm

Texas Instruments CCxxxx



SNE-ISMTV-TI868, TI2400

- Integrated tuner, modem, packet handling hardware
- Reception and transmission
 - FSK, MSK, ASK modulations
 - packet-based or continuous
- Energy detection measurements
- Test signal generation, interferer simulation
- Experiments with packet-based protocols

SNE-ISMTV-TI868

- Based on TI **CC1101** sub-1 GHz transceiver
- Frequency range 780 - 871 MHz
 - 868 MHz European SRD band
 - upper channels of the UHF band
- Bandwidth 50, 100, 200 kHz
- Power detector resolution 0.5 dBm
- Average noise level -150 dBm (@ 1Hz)
- Maximum TX power 12 dBm

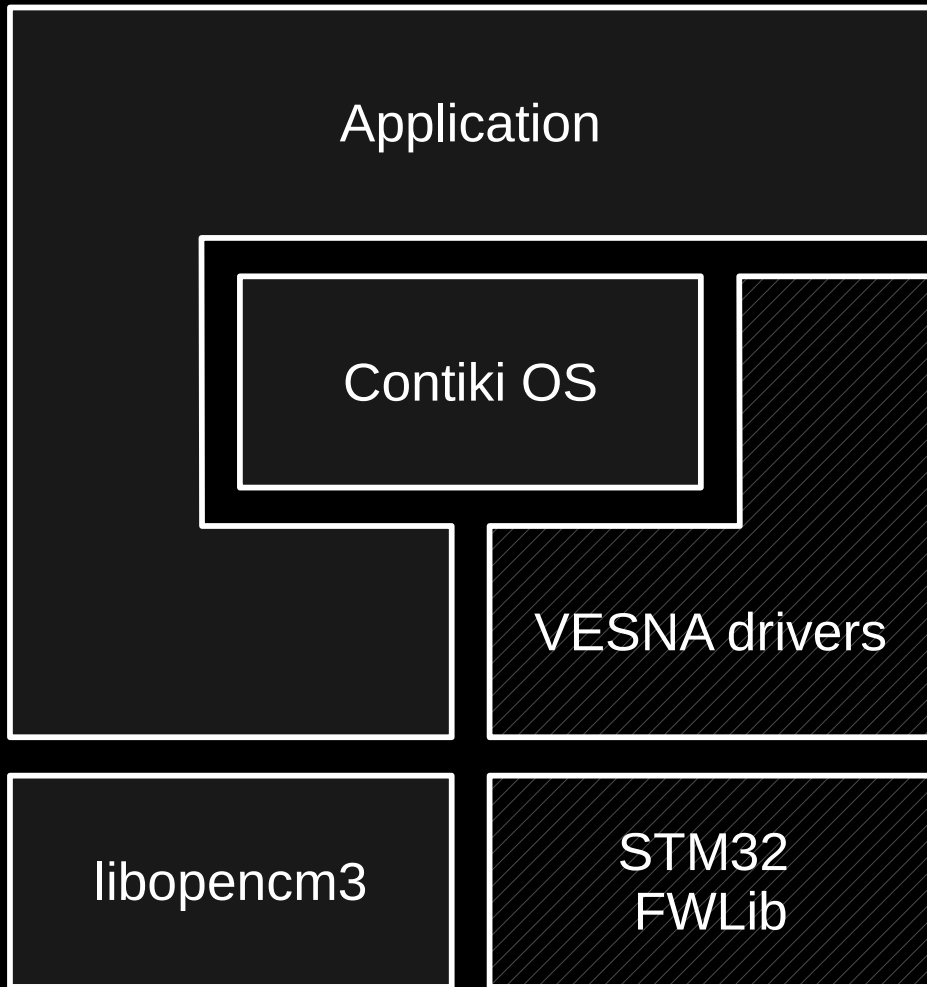
SNE-ISMTV-TI24

- Based on **TI CC2500** 2.4 GHz transceiver
- Frequency range 2.40 – 2.48 GHz
 - 2.4 GHz ISM band
- Bandwidth 200, 400 kHz
- Power detector resolution 0.5 dBm
- Average noise level -159 dBm (@ 1Hz)
- Maximum TX power 0 dBm

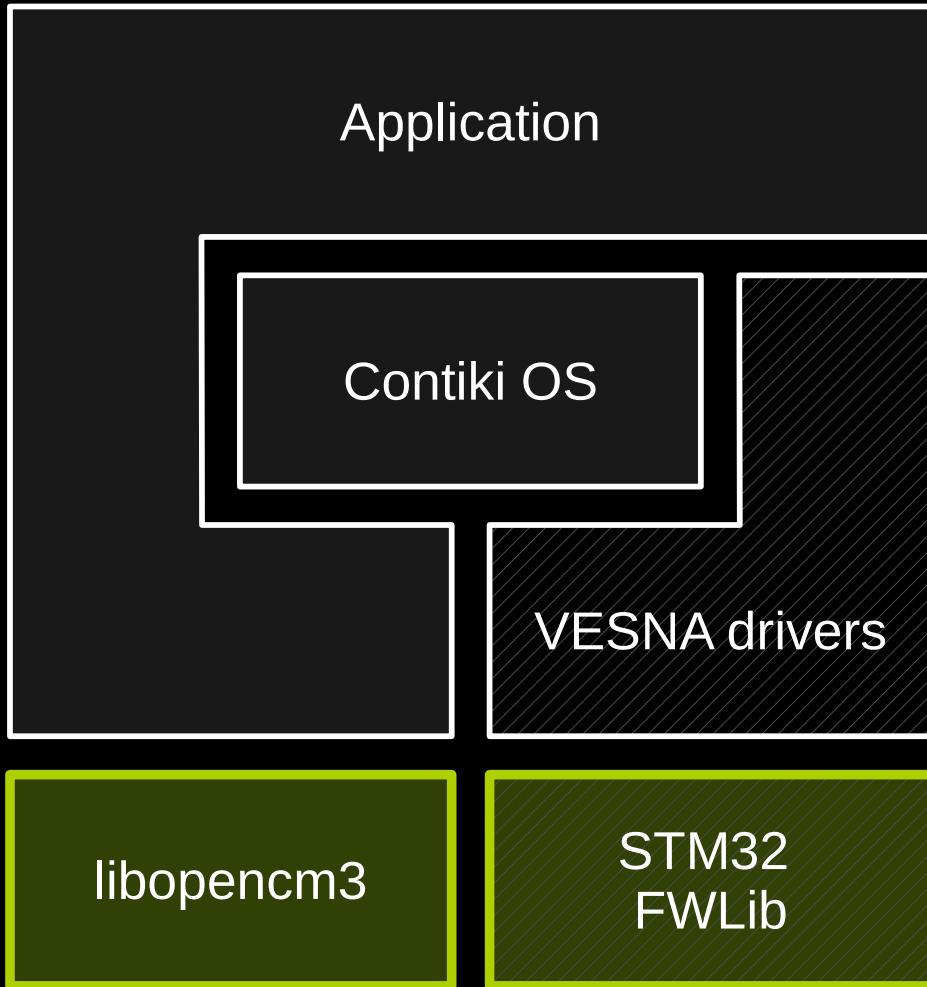
SNE-ISMTV

- Designed for spectrum sensing and cognitive radio applications
- Collection of radio-frequency hardware
 - UHF/VHF wide-band energy detection receiver
 - reconfigurable narrow-band sub-1 GHz transceiver
 - reconfigurable narrow-band 2.4 GHz transceiver
 - additional IEEE 802.15.4 radio (868 MHz)
- Independent of the testbed management network

VESNA software stack

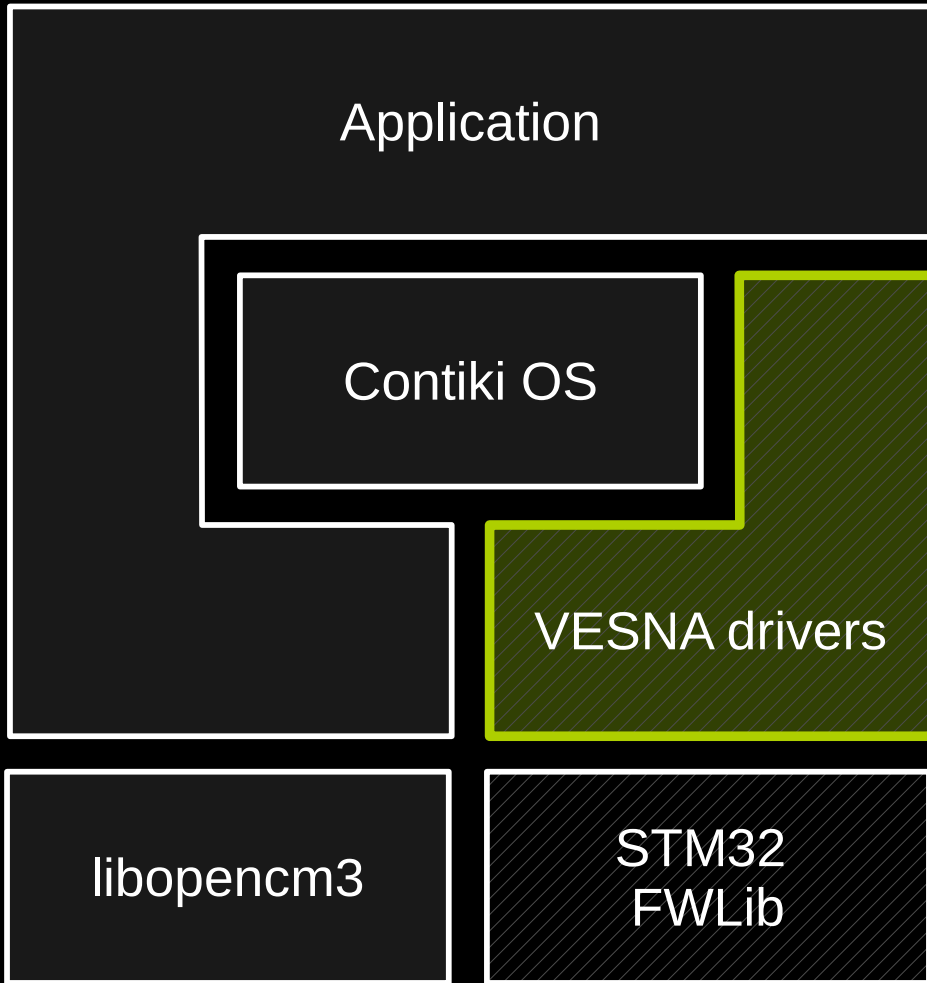


VESNA software stack



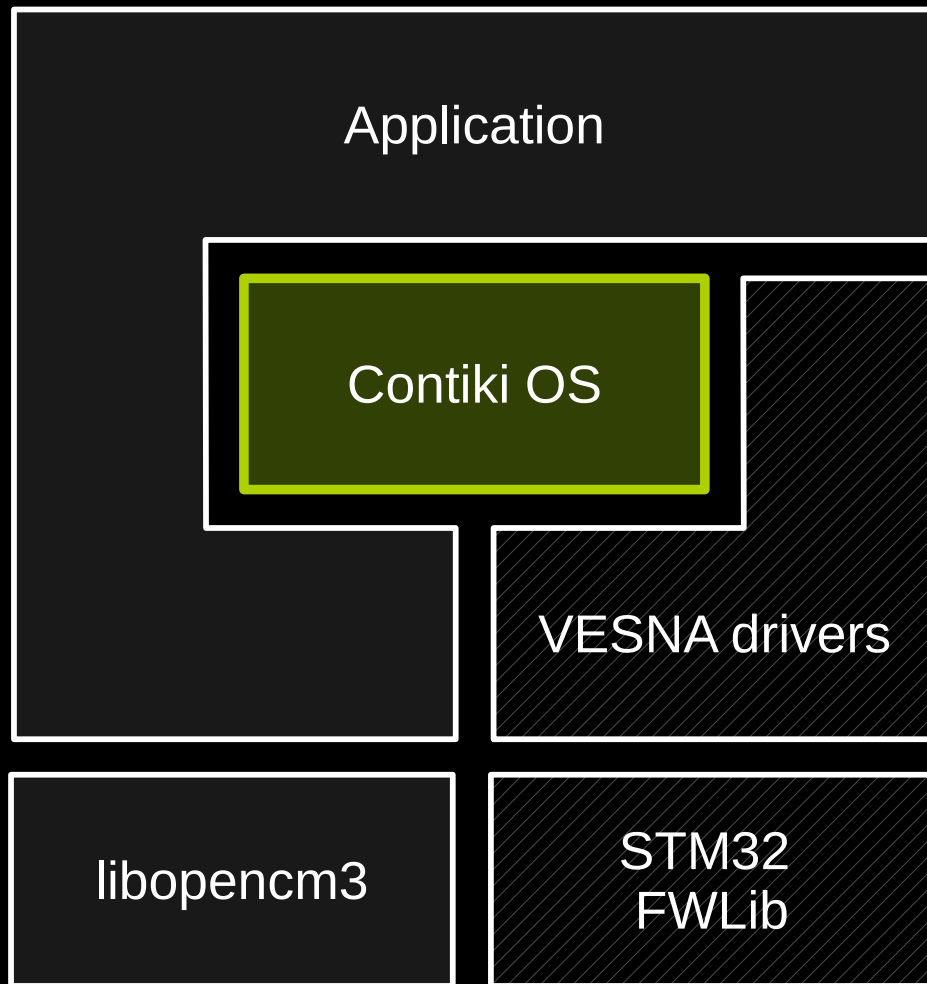
- Hardware register definitions
- Abstraction of MCU peripherals
- **libopencm3**
 - LGPL
- **STM32 FWLib**
 - open source, proprietary license

VESNA software stack



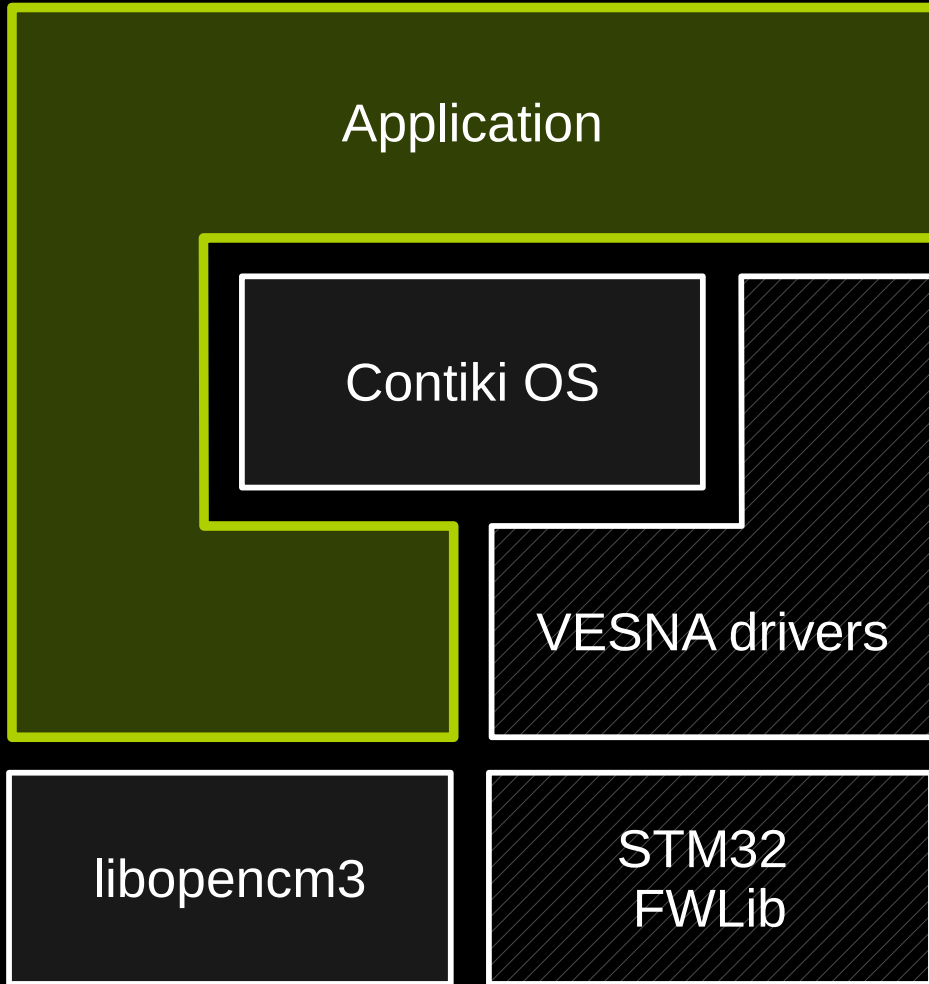
- High level abstraction
- Standard C library support
 - `#include <stdio.h>`
- Device drivers
 - storage, sensors, radio, SNE support, ...
- Networking
- Convenience functions

VESNA software stack



- Embedded operating system
- Cooperative multi-tasking
- Networking
 - IPv4, IPv6, 6lowPAN, RPL, CoAP
- Permissive BSD-style license

VESNA software stack

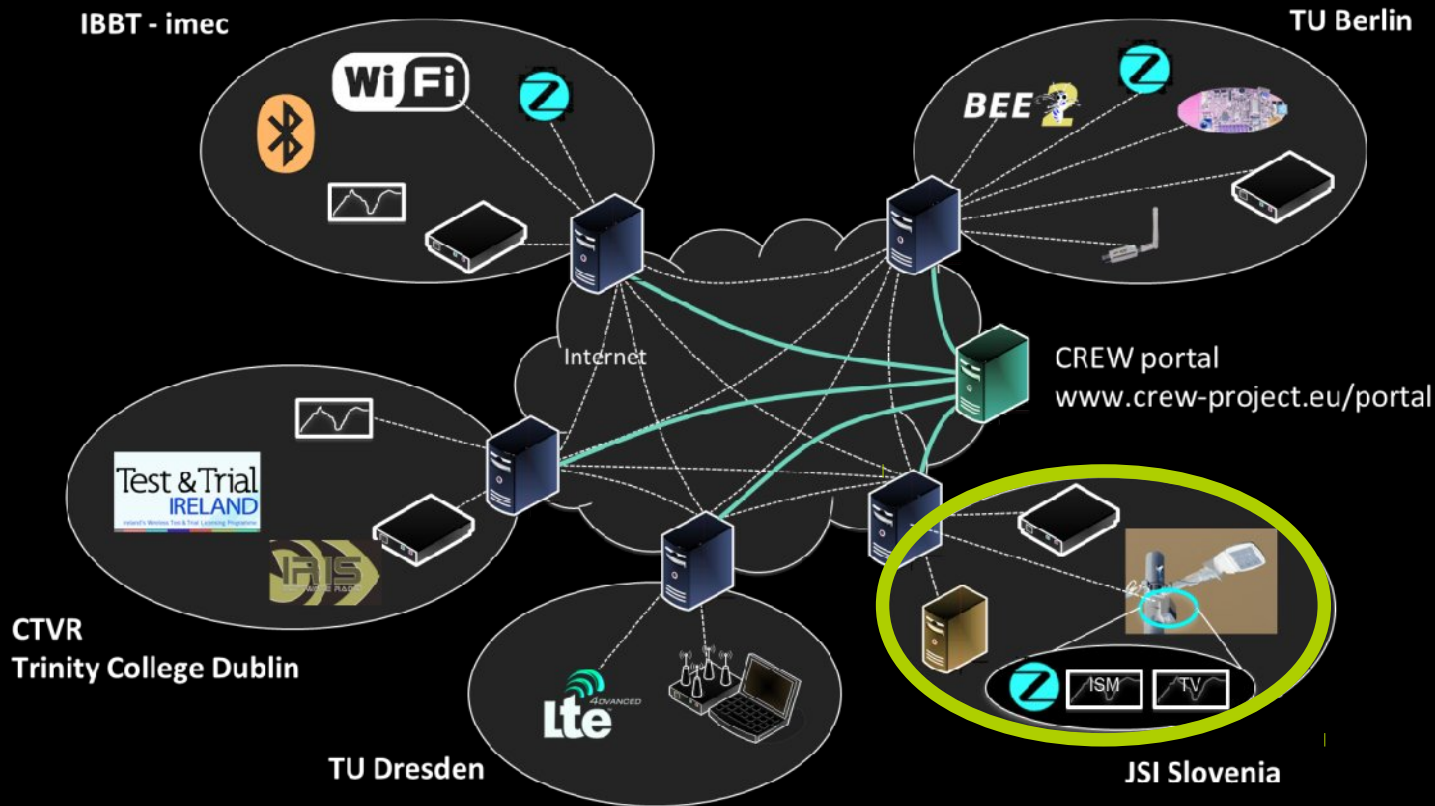


- Application can
 - Run on Contiki OS
 - Run without OS using VESNA drivers
 - Run without OS using libopencm3
- Depends on licensing, complexity

Overview

- ~~Overview of the VESNA platform~~
 - ~~– hardware~~
 - ~~– software stack~~
- Overview of Log-a-tec testbed
 - how remote access works
- Building a basic experiment with Python
 - required software
 - step-by-step demonstration
- Conclusion and further references

Log-a-tec testbed



WiFi IEEE 802.11

Bluetooth IEEE 802.15.1

Z-Wave IEEE 802.15.4

lte LTE-advanced

EyesIFX nodes

CR data base



IRIS GPP-based software radio platform



Comreg spectrum licenses



BEE2 FPGA platform



USRP software radio



Versatile Sensor Node on Light pole



imec Sensing Agent



UHF/VHF TV sensing



ISM bands sensing



THALES advanced sensing platform



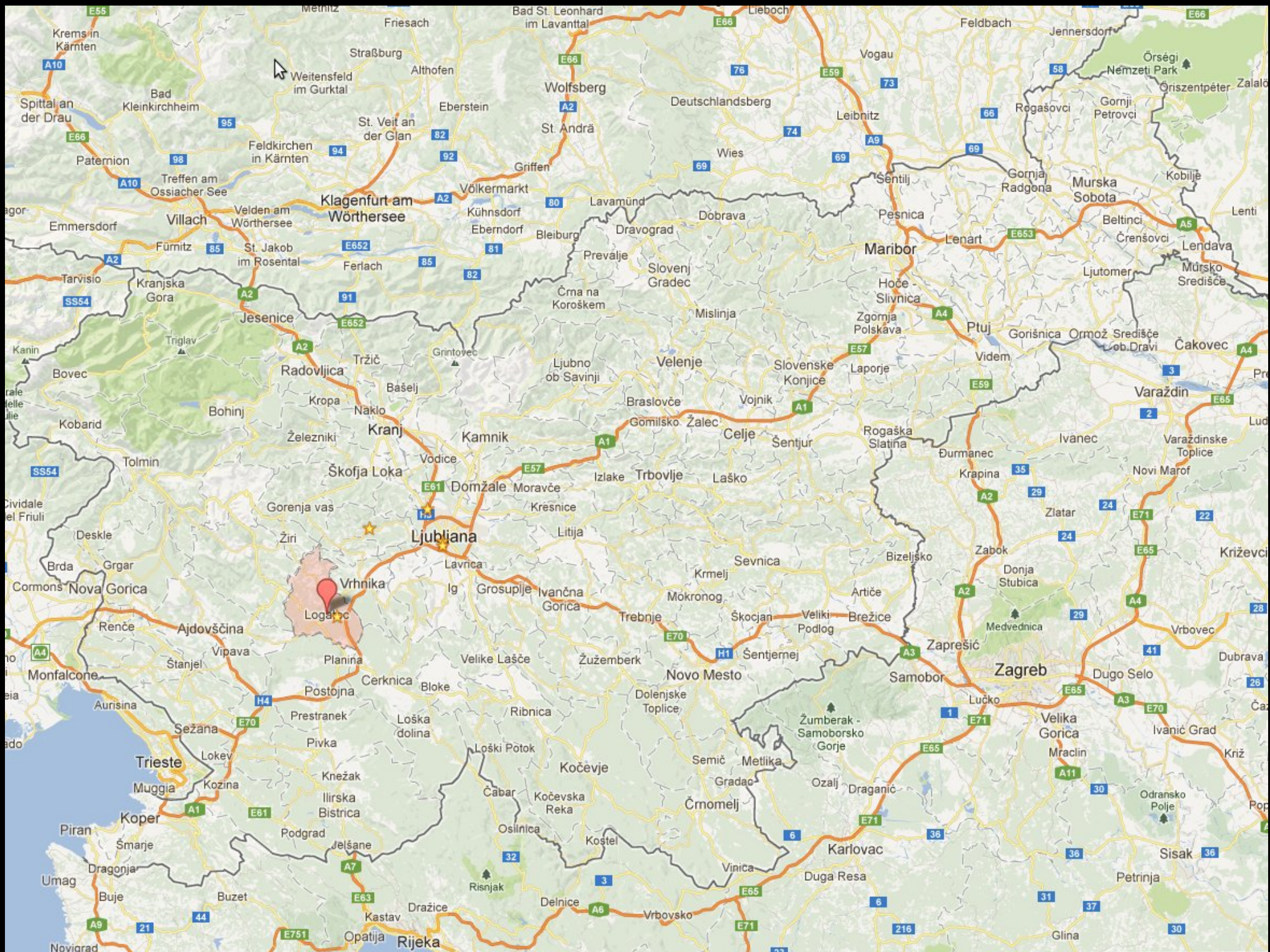
WiSpy Spectrum analyzer

Interconnection of portals

Interconn. between testbed elements

Log-a-tec testbed

- 51 VESNA wireless sensor nodes
 - mostly mounted on street lights, some on rooftops
 - 32 x SNE-ISMTV-TI24
 - 11 x SNE-ISMTV-TI868
 - 5 x SNE-ISMTV-UHF w/ low-gain antenna
 - 3 x SNE-ISMTV-UHF w/ high-gain antenna
- Two clusters in municipality of Logatec
 - industrial zone (23 nodes)
 - city center (28 nodes)







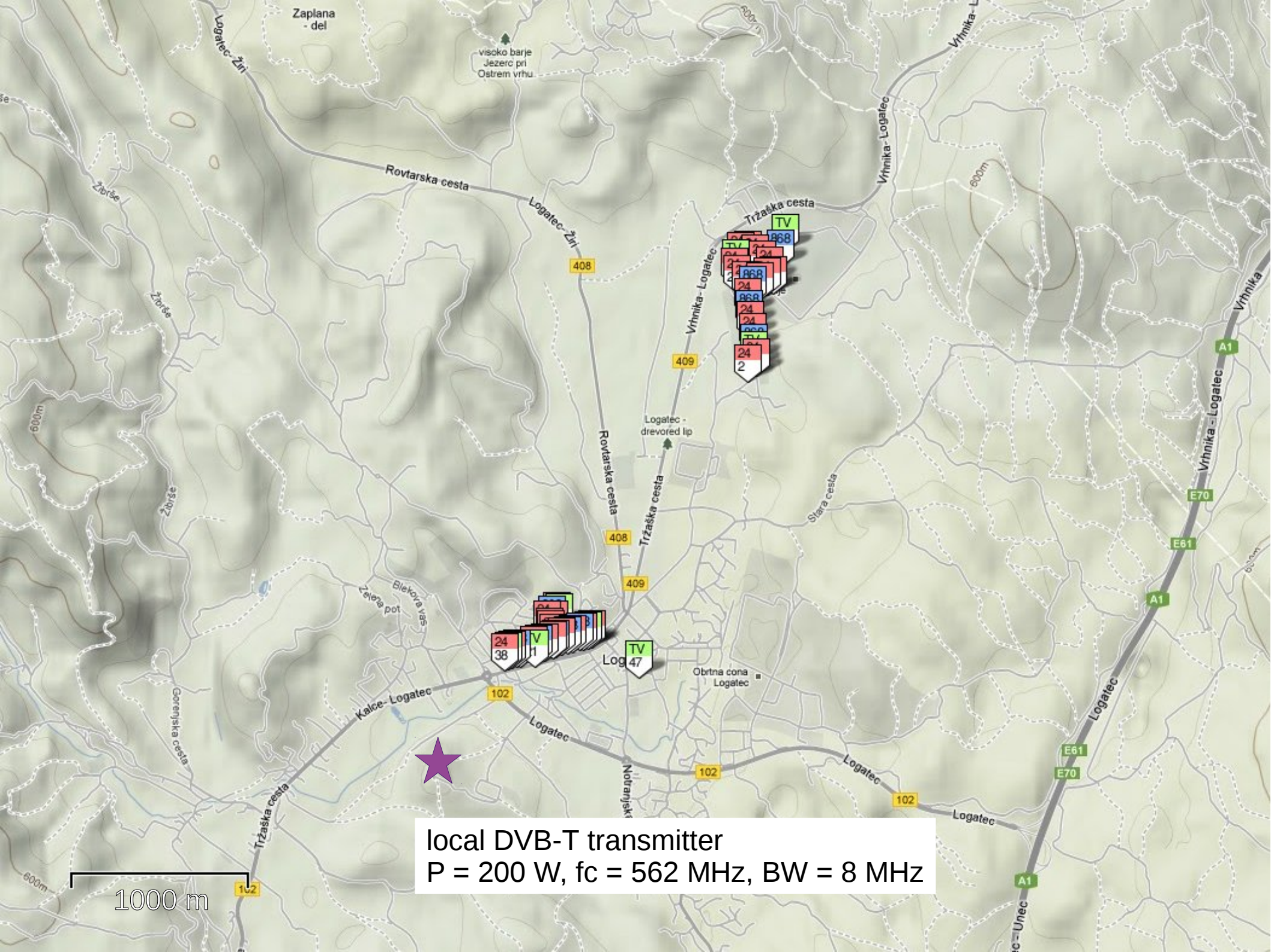






100 m

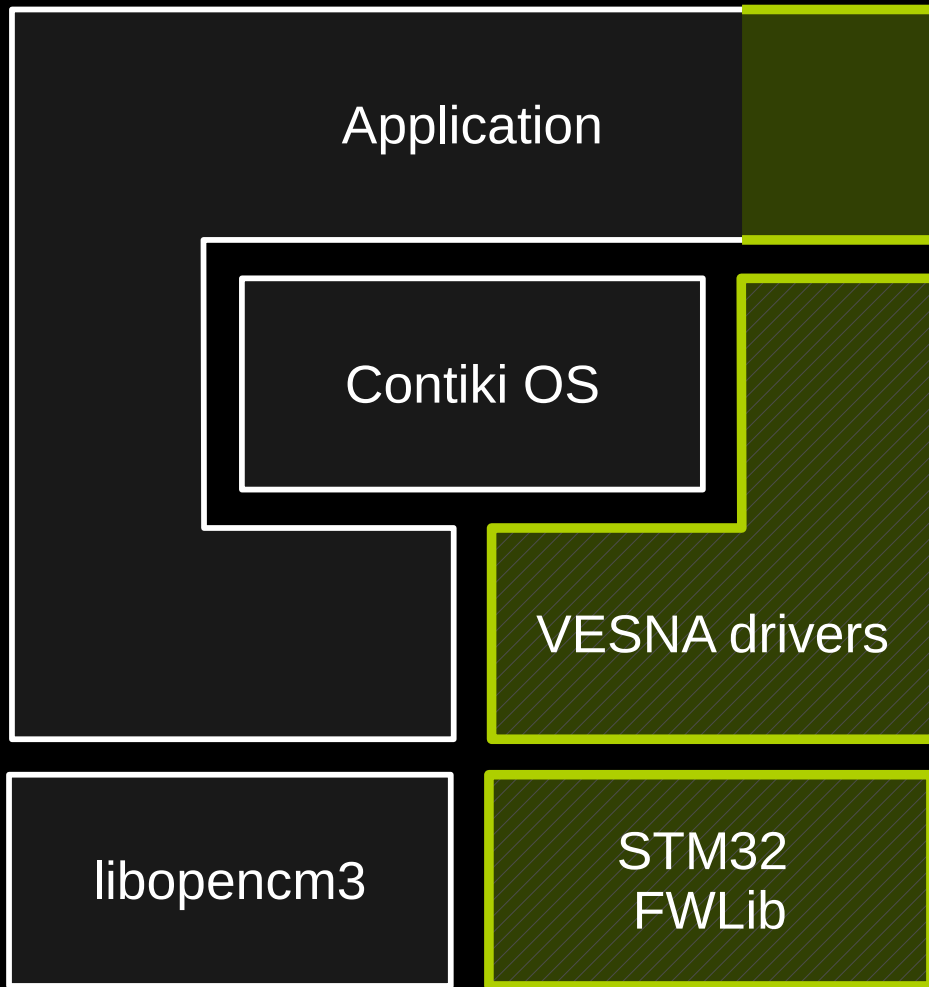




local DVB-T transmitter
 $P = 200 \text{ W}$, $f_c = 562 \text{ MHz}$, $BW = 8 \text{ MHz}$

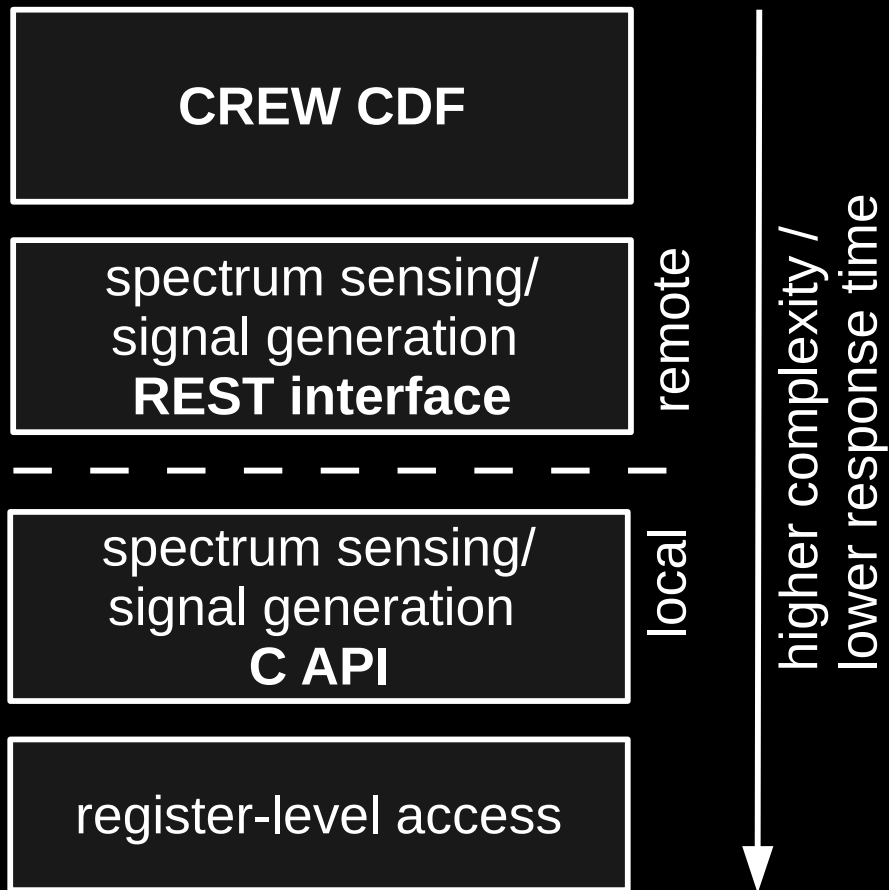


Log-a-tec VESNA application

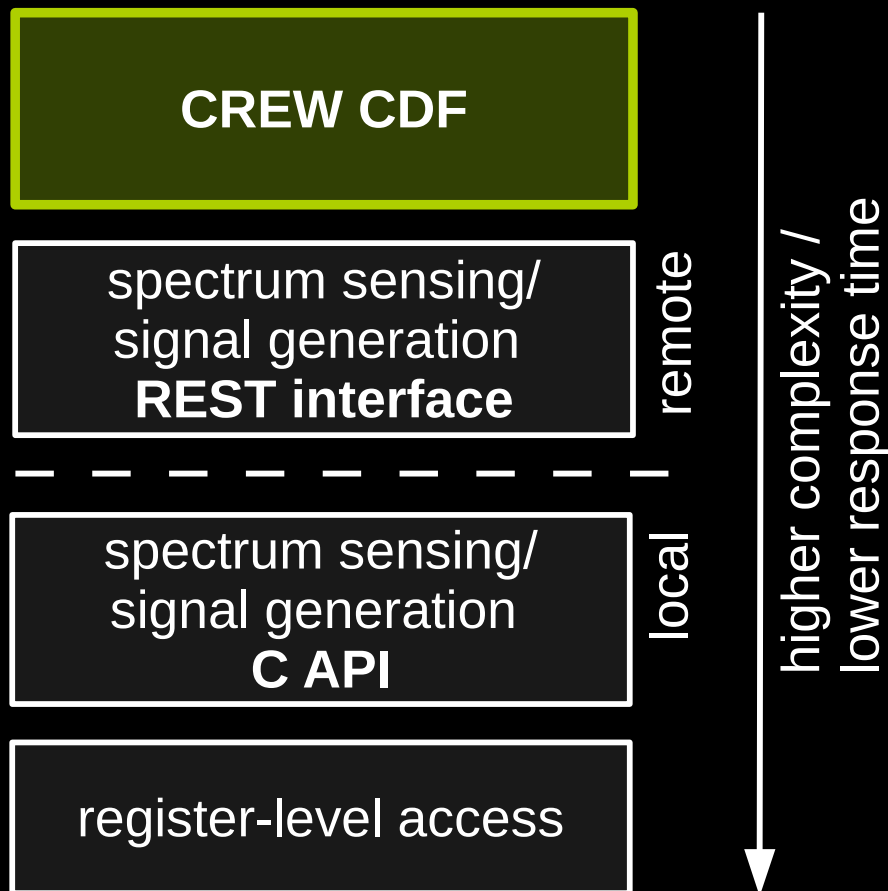


- REST interface for remote experiments
- Native code for local experiments
- Test bed Management
 - OTA reprogramming
 - monitoring

Controlling the SNE-ISMTV

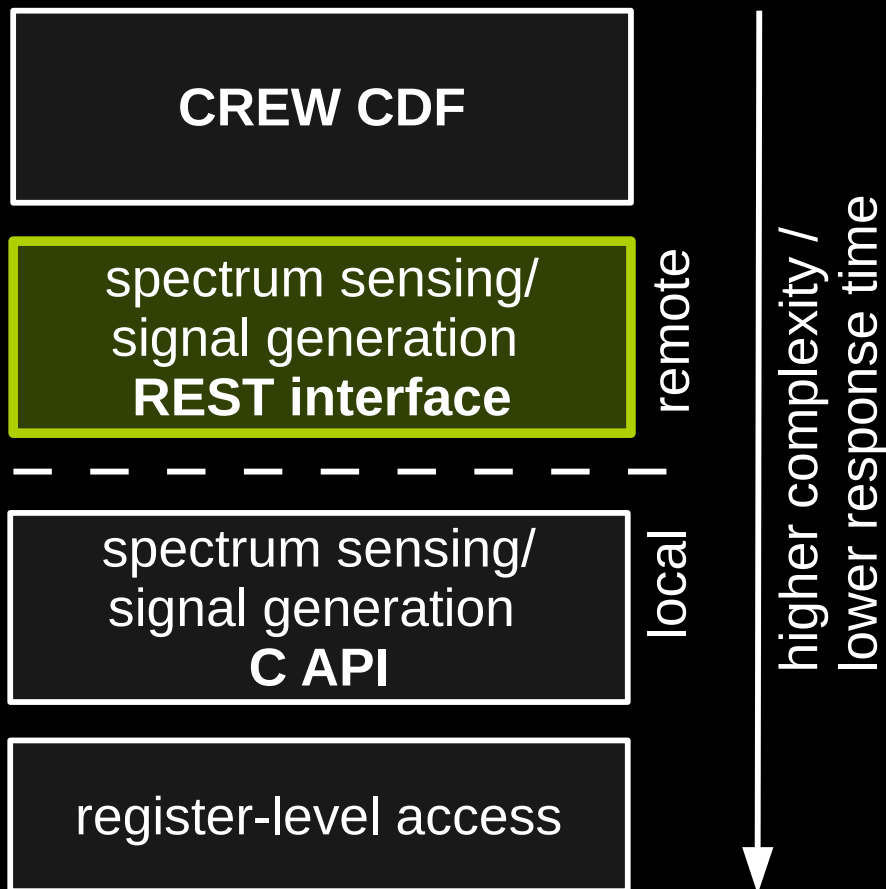


Controlling the SNE-ISMTV



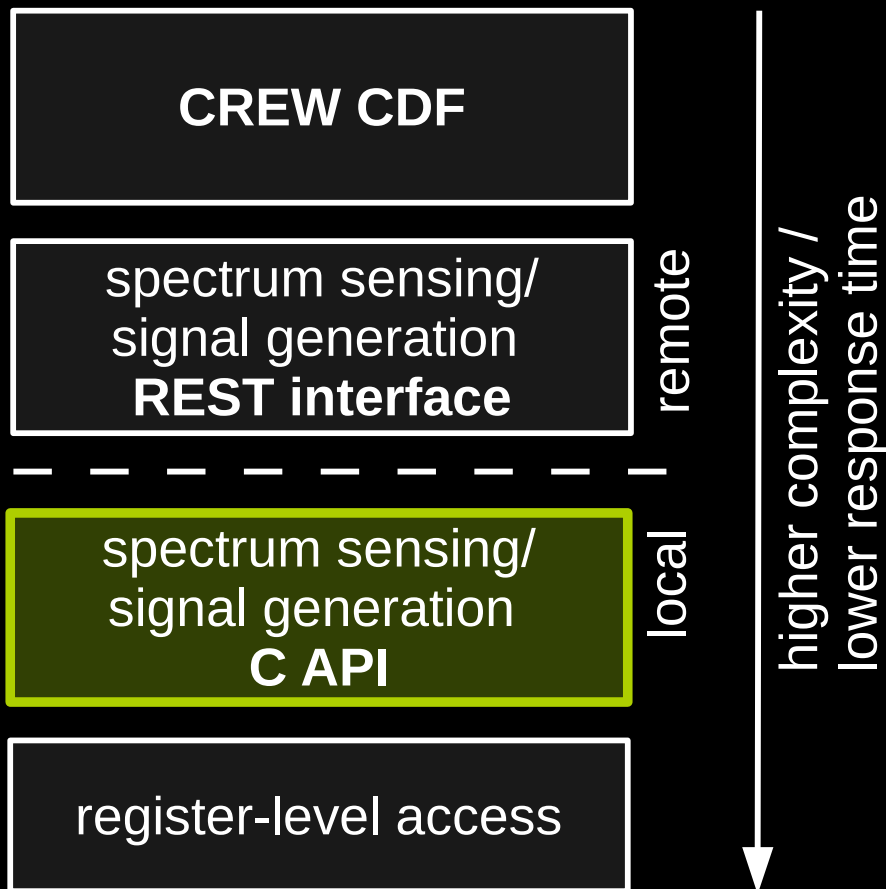
- Spectrum sensing setup
- Meta-data in XML
 - hardware setup,
 - frequency band,
 - date, author, ...
- Data in CSV or MatLab format
 - spectrum sensing results
 - (time, frequency, power)

Controlling the SNE-ISMTV



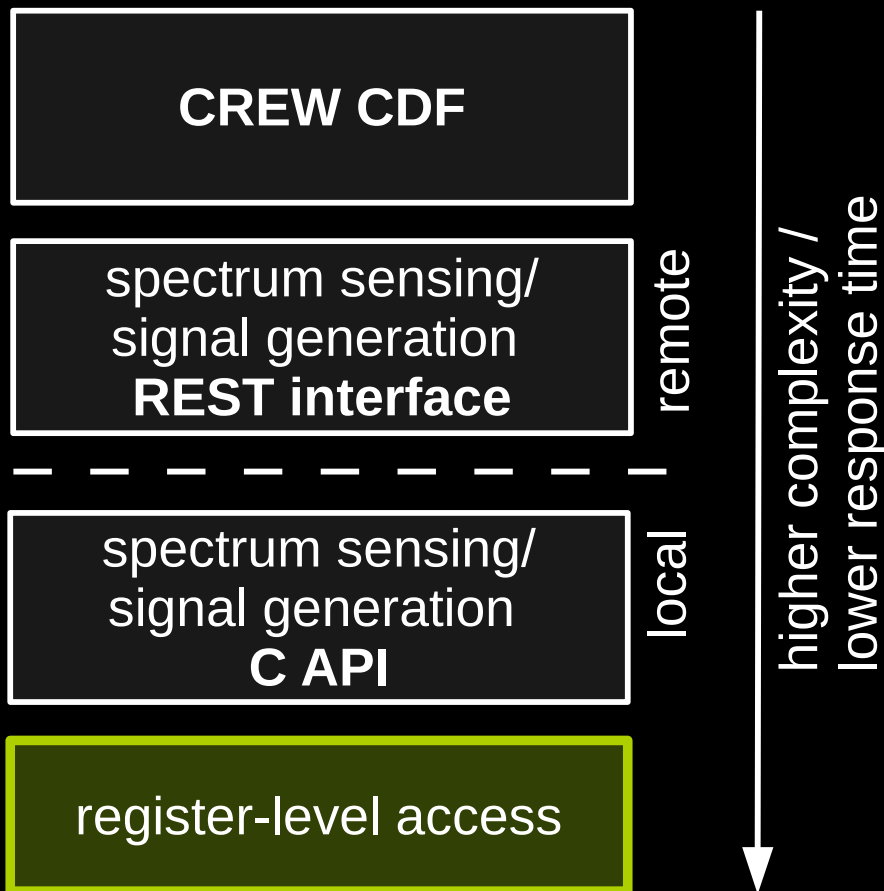
- High-level abstraction of hardware
- Spectrum sensing
 - sense band B at time T using device D and method M
- Signal generation
 - transmit at frequency F and power P

Controlling the SNE-ISMTV



- Direct access to the API from native code
- Removes network round-trips
 - lower latency
 - data processing on the sensor node
 - no interference on 868 MHz band

Controlling the SNE-ISMTV



- Directly programming radio hardware
 - CC1101, CC2500, TDA18219
- Exploit full capabilities of available hardware
- Time consuming testing required

REST interface

- Each node has a 16-bit network address
- Nodes act as a servers on the network
 - expose various resources
 - resources addressed by name
e.g. “**sensing/deviceStatus**”
 - GET and/or POST method
- Experimenter's computer acts as a client
 - issues requests to nodes and receives responses
 - **only one request at a time**

REST interface

- **GET method**

- retrieve data from the node
(e.g. measurement results, status messages, ...)
- doesn't change state

GET sensing/slotInformation?id=1

method

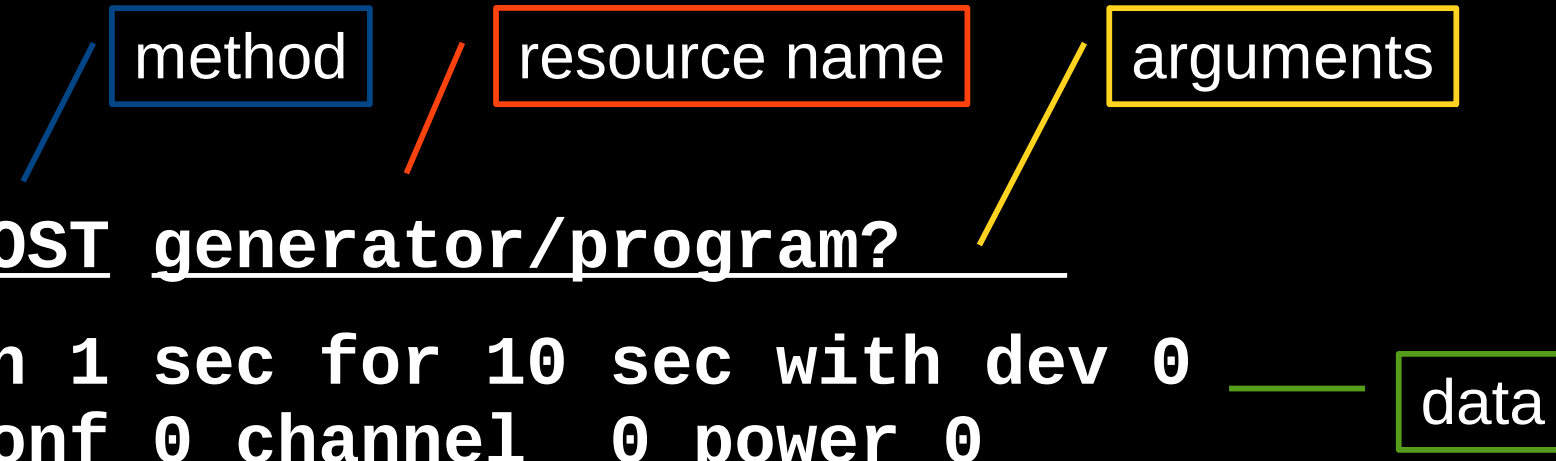
resource name

arguments

REST interface

- **POST method**

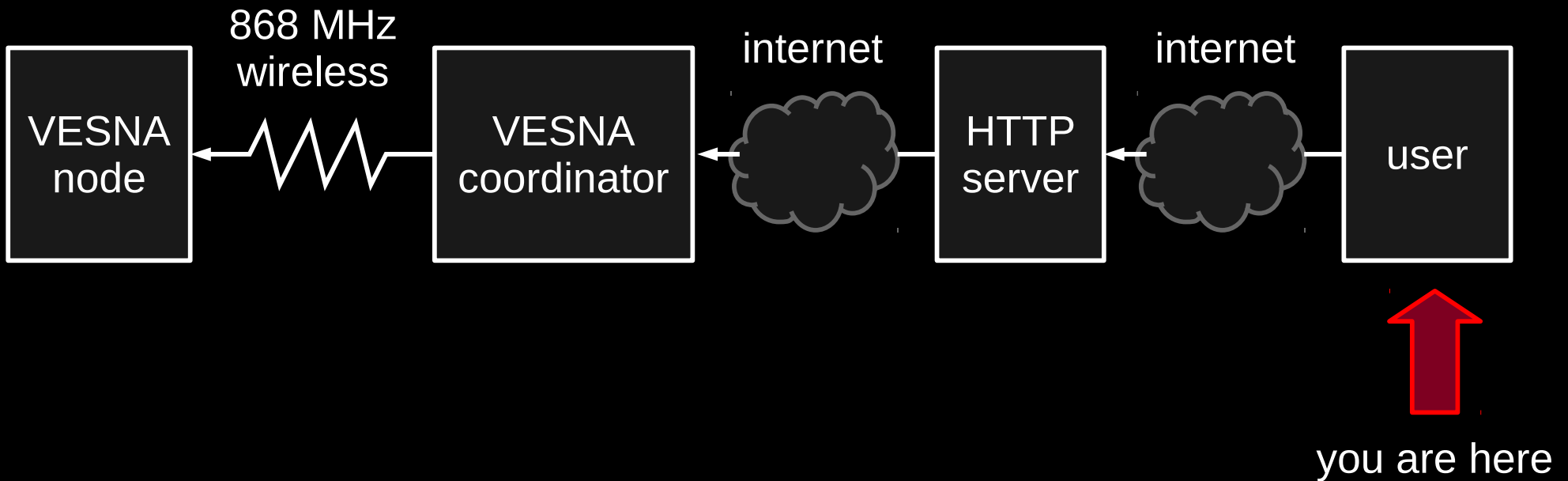
- send data to the node
(e.g. configuration parameters, trigger events, ...)
- changes state



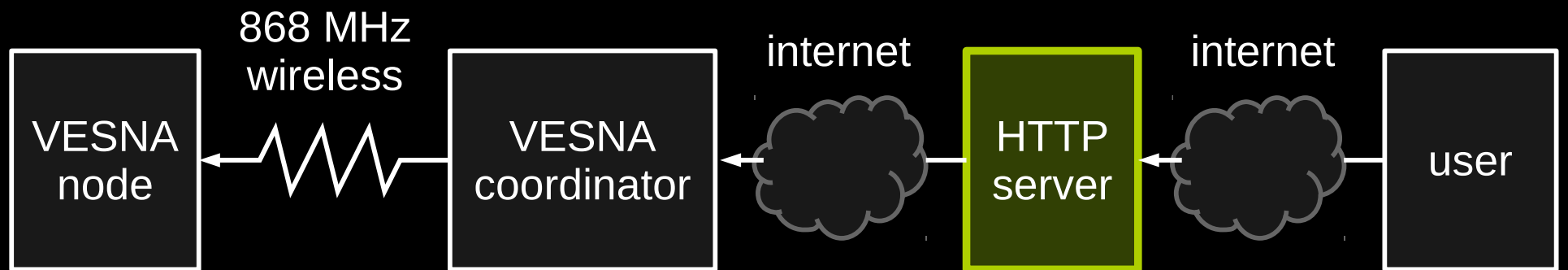
The diagram illustrates the components of a REST POST request. It shows a sequence of parts: a method, a resource name, arguments, and data. The method 'POST' is highlighted with a blue box. The resource name 'generator/program?' is highlighted with an orange box. The arguments 'in 1 sec for 10 sec with dev 0' and 'conf 0 channel 0 power 0' are highlighted with a yellow box. The data 'data' is highlighted with a green box. Colored lines connect the boxes to their respective parts in the request: a blue line from 'method' to 'POST', an orange line from 'resource name' to 'generator/program?', a yellow line from 'arguments' to the arguments, and a green line from 'data' to 'data'.

POST generator/program?
in 1 sec for 10 sec with dev 0
conf 0 channel 0 power 0 data

Network overview



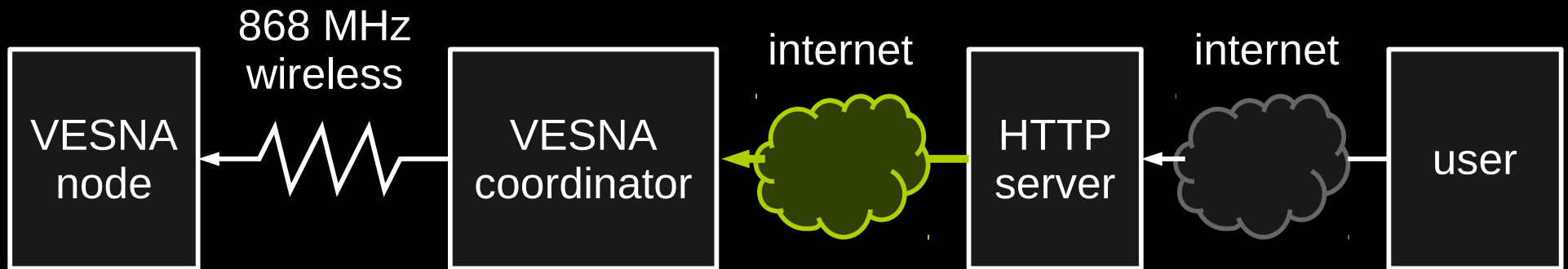
Log-a-tec web portal



- Web portal
 - map with locations of VESNA nodes
 - manually issue GET and POST requests
 - simulations with GRASS-RaPlaT
- HTTP API end-point
 - programmatic access to VESNA REST interface

you are here

Accessing the coordinator



- HTTP server forwards GET & POST requests to coordinator
- Simple HTTP-like (ALH) protocol over SSL tunnel
- One coordinator per cluster
 - coordinator identified by cluster ID

LOG-a-TEC

SensorLab
Jožef Stefan Institute

GENERAL

SIMULATIONS

EXPERIMENTS

REPROGRAM

Cognitive Radio Networking

Choose the cluster: JSI

GRASS-RaPlaT Simulation:

< Select the Simulation >

Opacity: 50

Download request-response log file in text or in hexadecimal format:

[Text request-response log file](#) [Hex request-response log file](#)

Direct communication with the nodes:

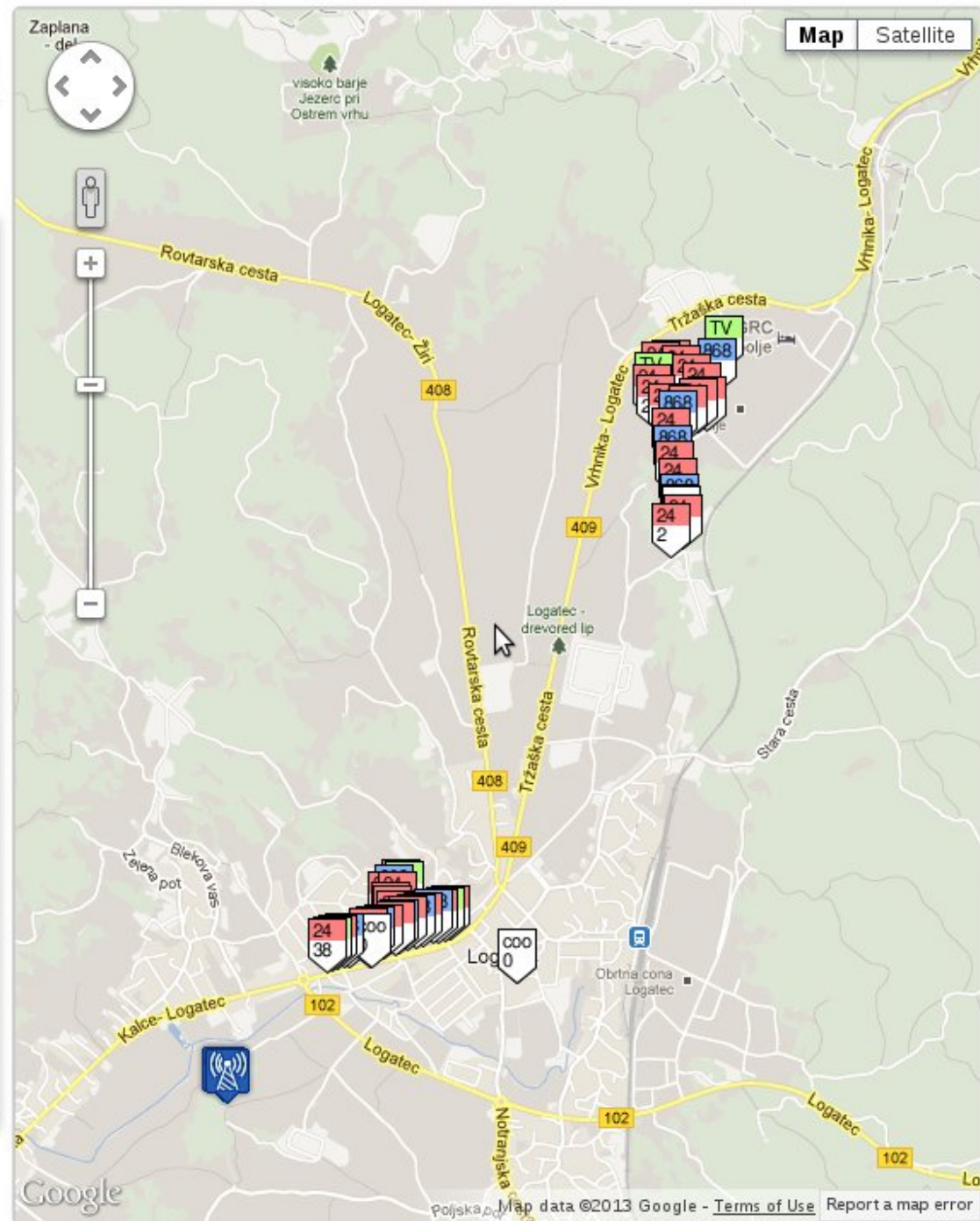
Enter Resource

GET

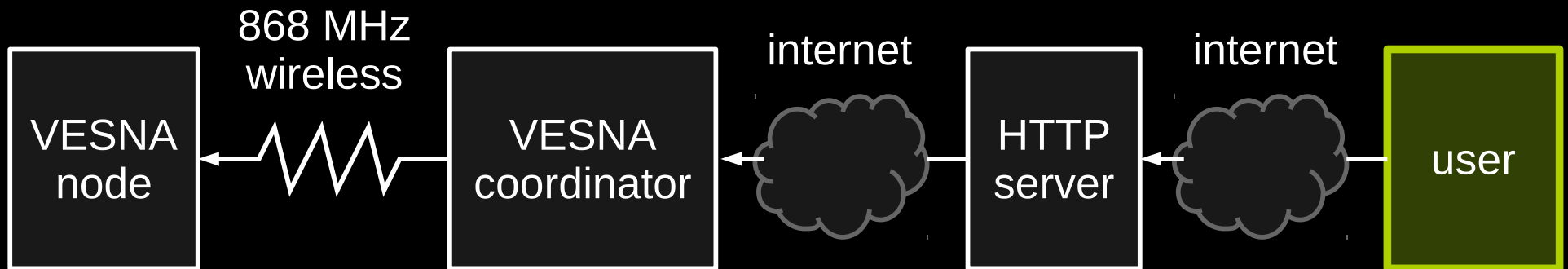
Enter Resource

Enter Content

POST



Python libraries



- <https://crn.log-a-tec.eu/communicator?cluster=10001&method=get&resource=hello>
- We provide a Python module to communicate with VESNA nodes

Overview

- ~~Overview of the VESNA platform~~
 - ~~hardware~~
 - ~~software stack~~
- ~~Overview of Log-a-tec testbed~~
 - ~~how remote access works~~
- Building a basic experiment with Python
 - required software
 - step-by-step demonstration
- Conclusion and further references

Installing Python libraries

- **set \$PYTHONPATH**

- add to ~/.bashrc:
export PYTHONPATH="\$HOME/local/lib/python"

- **install vesna-alh-tools**

- <https://github.com/sensorlab/vesna-alh-tools>
- \$ python setup.py install --home=~/.local

- **install vesna-spectrum-sensor**

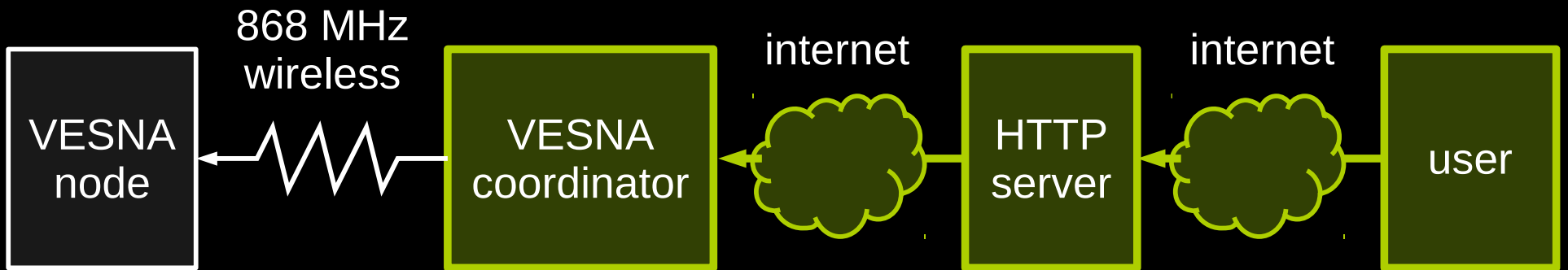
- <https://github.com/sensorlab/vesna-spectrum-sensor>
- \$ cd python
- \$ python setup.py install --home=~/.local

Installing Python libraries

- set authentication details
 - create ~/.alhrc with:

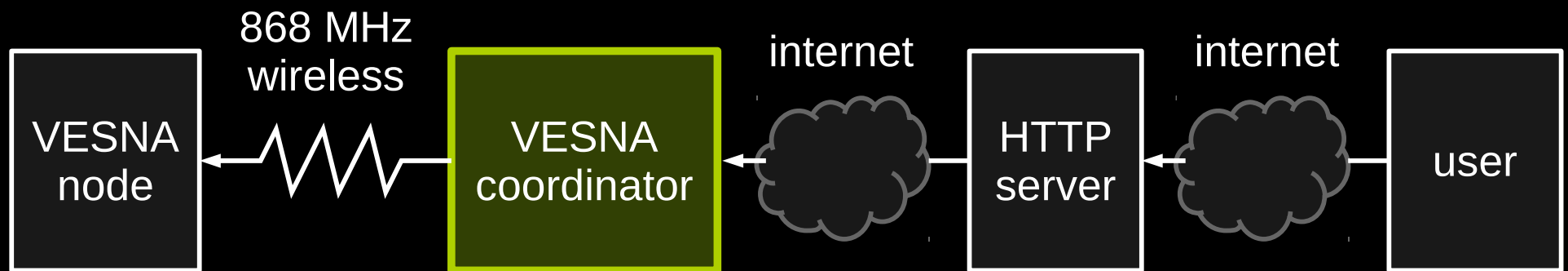
```
Host crn.log-a-tec.eu
User <username>
Password <password>
```

Demo



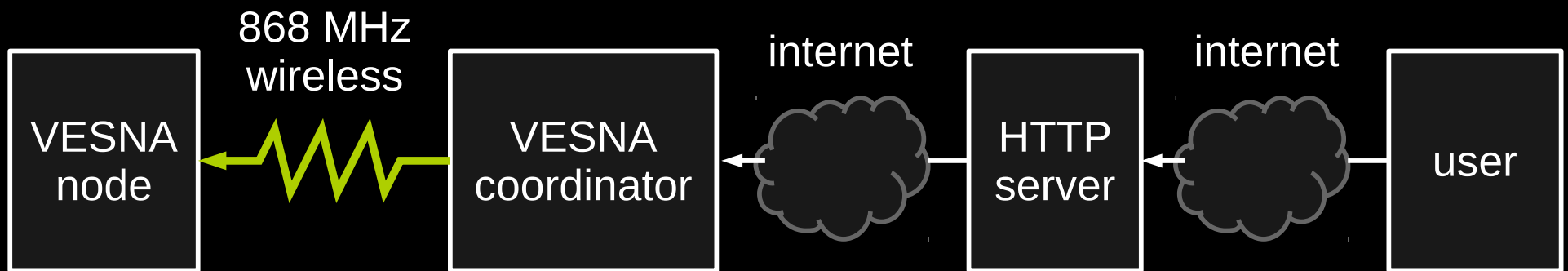
01-hello.py

Coordinator proxy



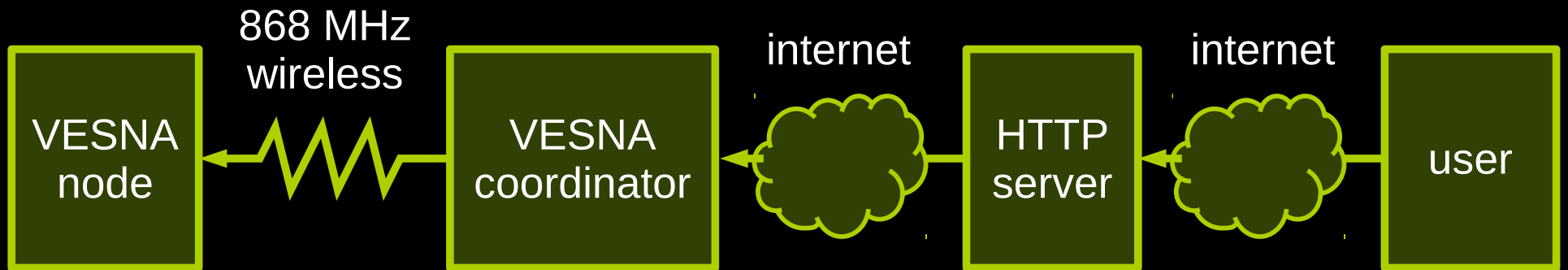
- Coordinator can proxy GET & POST requests to nodes over the management network
- GET nodes?19/sensor/mcuTemp
issues
GET sensor/mcuTemp
to node 19

VESNA networking



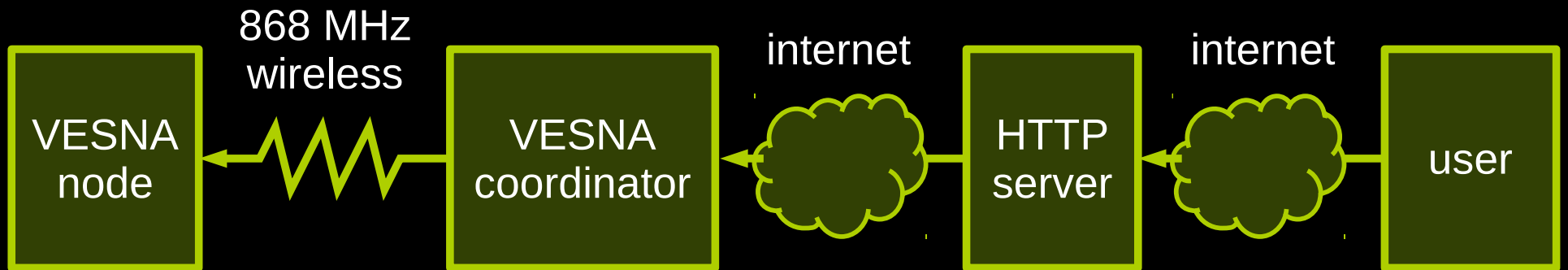
- management mesh network
- 868 MHz (European SRD band)
- typical bulk transfers ~ 300 bytes/s
- typical round-trip times ~ 600 ms
- HTTP-like protocol over IEEE 802.15.4 mesh

Demo



02-proxy.py

Demo



manage.py interactive session

Spectrum sensing interface

- Hardware abstraction
 - each node has one or more physical devices
 - each device has one or more configurations
- Device configuration determines
 - usable frequency range
 - channel number → central frequency relation
 - settle time required after channel change
 - channel bandwidth
 - averaging / post processing parameters (if any)

Demo

03-spectrum-sensing-devices.py

Spectrum sensing interface

- class **SpectrumSensor**
 - wrapper around **ALHPProxy** for convenience methods
- class **ConfigList**
 - describes possible hardware device configurations
- class **SweepConfig**
 - describes frequency sweep
 - hardware configuration, start, stop, step frequency
- class **Sweep**
 - results of spectrum sensing sweep
(timestamp, frequency, power)

Demo

04-single-sweeps.py

Signal generation interface

- Hardware abstraction
 - each node has one or more physical devices
 - each device has one or more configurations
- Device configuration determines
 - usable frequency, power range
 - channel number → central frequency relation
 - transmitted waveform

Signal generation interface

- class **SignalGenerator**
 - wrapper around **ALHProxy** for convenience methods
- class **ConfigList**
 - describes possible hardware device configurations
- class **TXConfig**
 - describes signal transmission
 - hardware configuration, frequency, power
- class **SignalGeneratorProgram**
 - Transmission configuration, start time, duration

Demo

05-signal-generation.py

Spectrum sensing interface

- class **SpectrumSensorProgram**
 - frequency sweep, start time, duration
 - SD card slot to write results to
- **SpectrumSensor.program()**
 - send task to the node
- **SpectrumSensor.is_complete()**
 - check if task complete
- **SpectrumSensor.retrieve()**
 - retrieve results from the SD card
- class **SpectrumSensorResult**
 - collection of sweeps

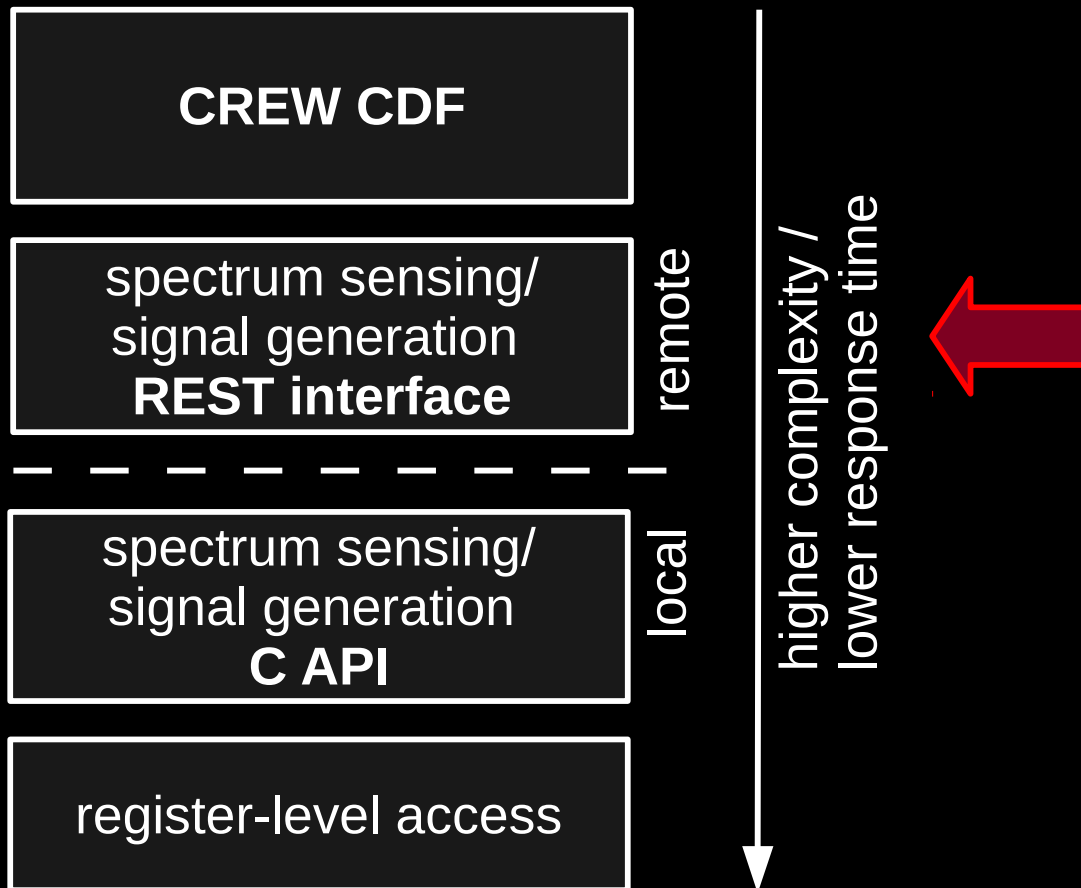
Demo

06-programmed-tasks.py

Overview

- ~~Overview of the VESNA platform~~
 - ~~– hardware~~
 - ~~– software stack~~
- ~~Overview of Log-a-tec testbed~~
 - ~~– how remote access works~~
- ~~Building a basic experiment with Python~~
 - ~~– required software~~
 - ~~– step-by-step demonstration~~
- Conclusion and further references

Controlling the SNE-ISMTV



Developing VESNA applications

- Setup manual for Linux based development
 - <http://sensorlab.github.com/vesna-manual>
- Spectrum sensing and signal generation C API
 - see CREW deliverable 3.2
- Overview
 - add code to Logatec application
 - review & testing by JSI
 - over-the-air upload to nodes
 - communicate with the application over REST

Register level access

- See reference documentation for SNE-ISMTV receiver, transceiver ICs
 - TDA18219, CC1101, CC2500 datasheets
 - (SNE-ISMTV datasheet - WIP)
- read_reg(), write_reg(), interrupts
- Usually extensive testing required
 - tuners have bugs, unexpected features
 - for new RF front-end configurations calibration required if accurate RSSI measurements / TX power levels are desired

Questions?

tomaz.solc@ijs.si

<http://sensorlab.ijs.si>

<http://github.com/sensorlab>