TinyOS Lab Exercise

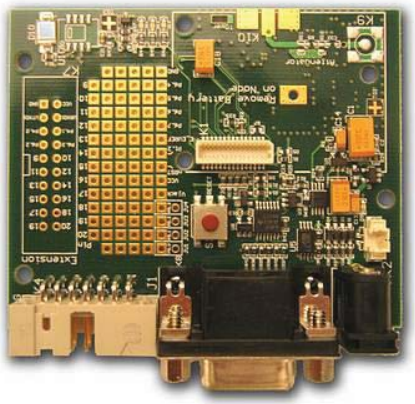
Introduction to TinyOS 2

TinyOS Lab Exercise in Ad Hoc and Sensor Networks

- **Sensor network programming in a nutshell**
  - Read ‘Getting started with TinyOS’ (at home)
  - Solve two Lab-style exercises on real hardware
    - Teams of two to three students are ideal
    - One lab working place is available in ETL F29
    - Reservation system on the course website
  - Expected time needed for all tasks: 3-4 hours

Wireless Sensor Nodes

- **Shockfish TinyNode**
  - Slow CPU
    - 8 MHz Texas Instruments MSP430 microcontroller
  - Little memory
    - 10 KByte RAM, 48 KByte ROM, 512 Kbyte external flash
  - Short-range radio
    - 868 MHz Xemics XE1205 ultra-low power wireless transceiver
  - Light sensor, temperature and humidity sensors



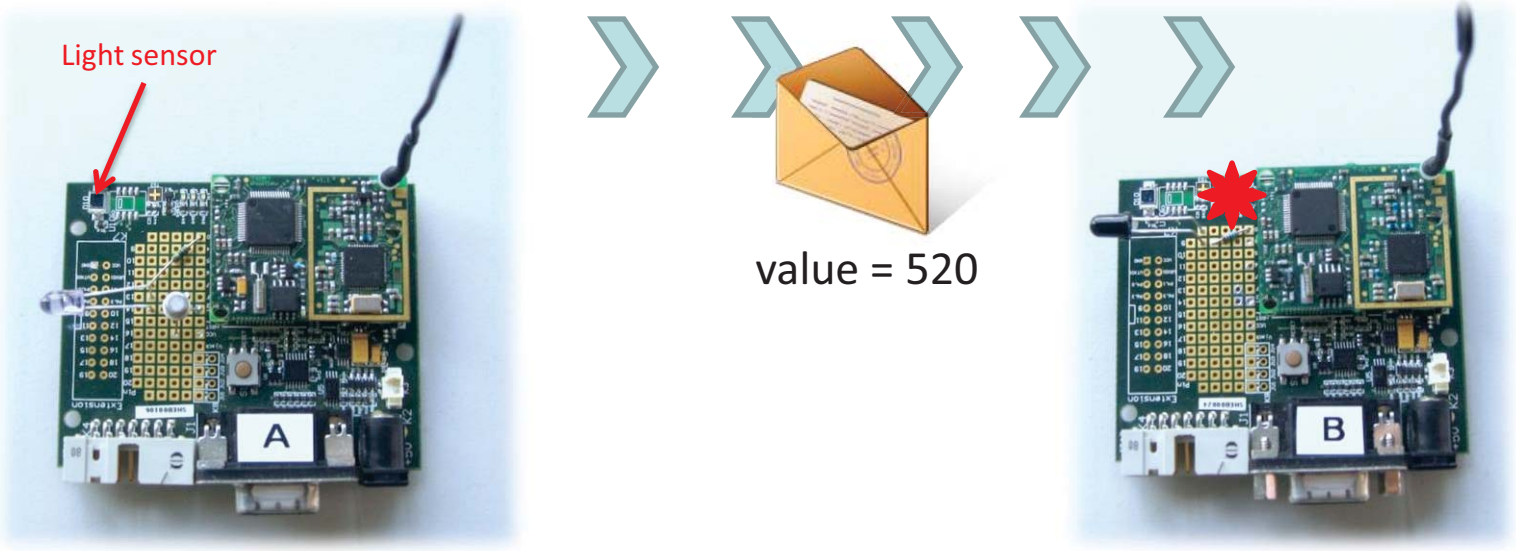
Extension Board

+

TinyNode 584

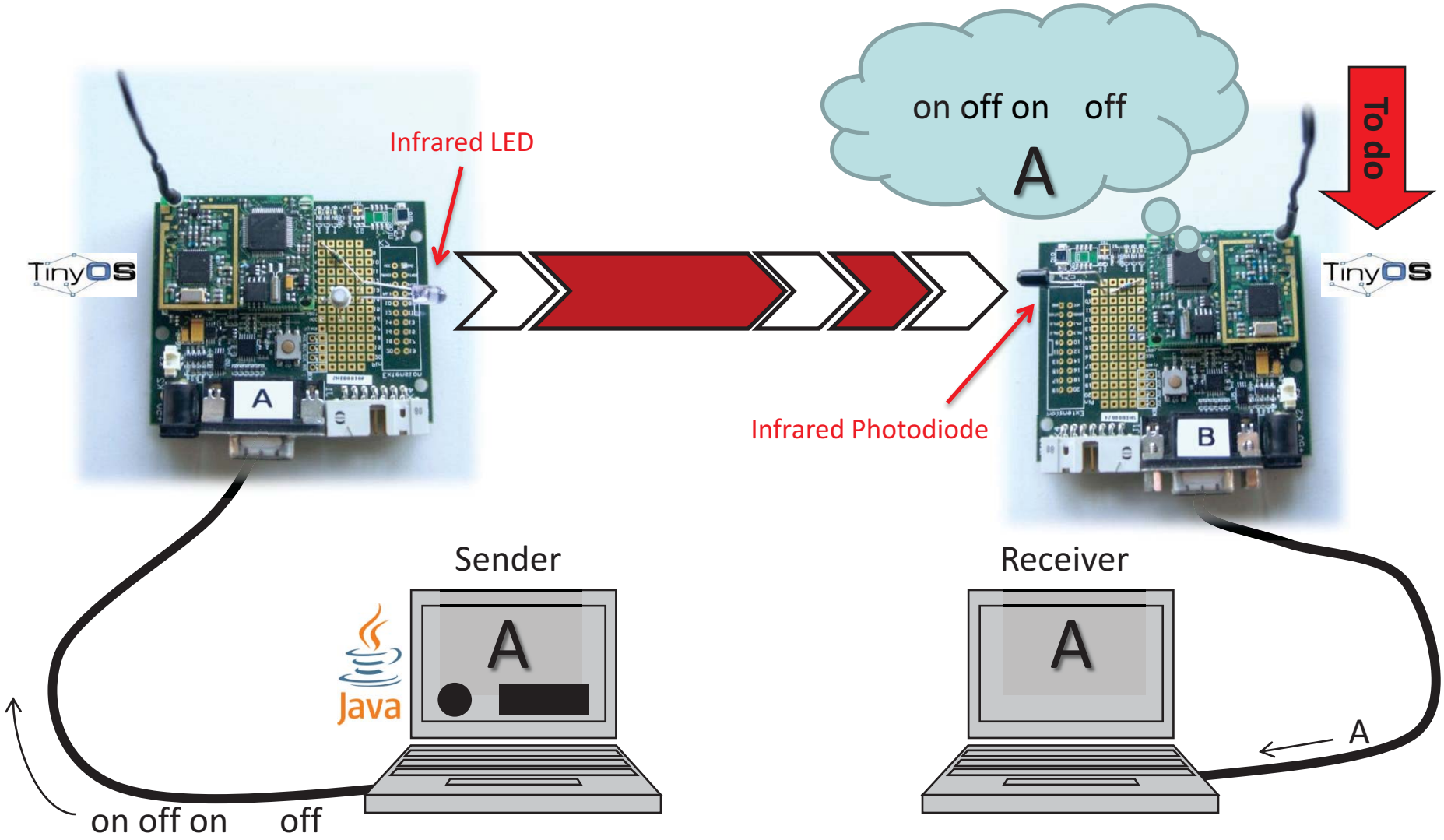
Exercise 1

- **Exchange of a sensor data**
  - Two sensor nodes are used for this task
  - One node periodically samples its light sensor and broadcasts the sensor reading over its radio
  - The other node listens for radio messages and signals if it is getting brighter or darker
    - Brighter → The green LED of the receiver is set
    - Darker → The red LED of the receiver is set
    - No significant change → The yellow LED is set



Exercise 2

- **Optical Communication using Morse Codes**



TinyOS

- TinyOS is an operating system for sensor nodes
  - Open source project with a strong academic background
  - Hardware drivers, libraries, tools, compiler

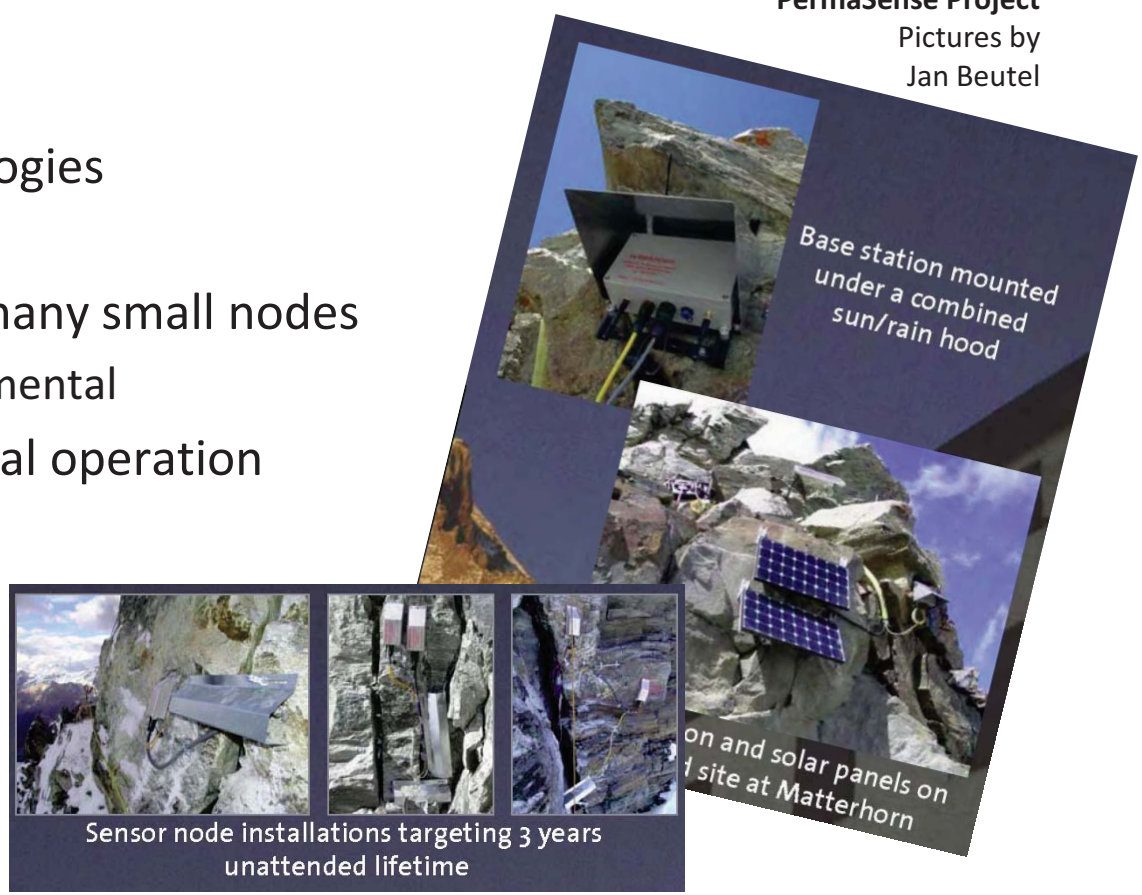
- TinyOS applications are written in nesC
  - C dialect with extra features
  - nesC compiler converts your application into plain C code

http://www.tinyos.net

Why using a new Operating System?

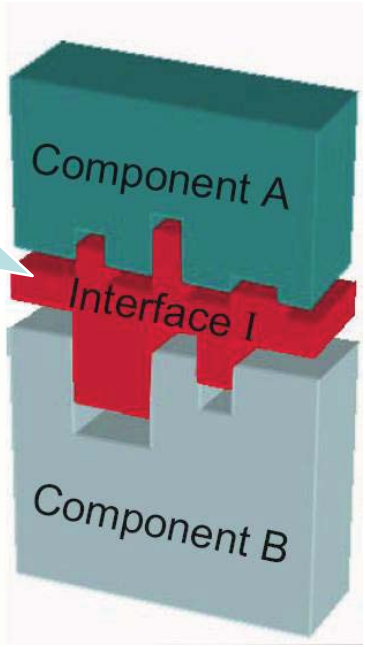
- Measure real-world phenomena
  - Event-driven architecture
- Resource Contraints
  - Hurry up and sleep!
- Adapt to changing technologies
  - Modularity & re-use
- Applications spread over many small nodes
  - Communication is fundamental
- Inaccessible location, critical operation
  - Robustness

PermaSense Project
Pictures by
Jan Beutel



NesC/TinyOS Programming Model

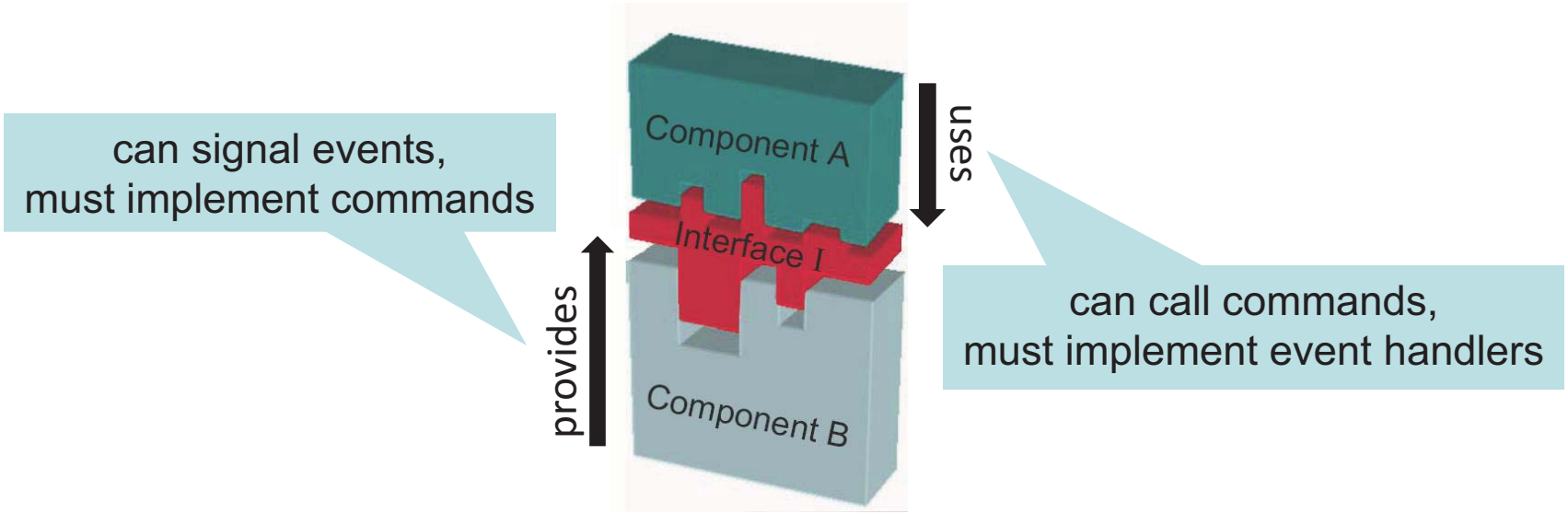
- Programs are built out of components
- Two types of components:
  - Modules: Implement program logic
  - Configurations: Wire components together
- Components use and provide interfaces
- Components are wired together by connecting interface users with interface providers



Programming Model

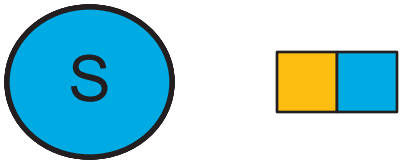
- Interfaces contain definitions of
  - Commands
  - Events

- Components implement the event handlers they use and the commands they provide



Concurrency Model

- Coarse-grained concurrency only
  - Implemented via tasks



- Tasks are executed sequentially by the TinyOS scheduler
  - no threads
  - Atomic with respect to other tasks (single threaded)
  - Longer background processing jobs

- Events (interrupts)
  - Time critical
  - Preempt tasks
  - Short duration (hand off computation to tasks if necessary)

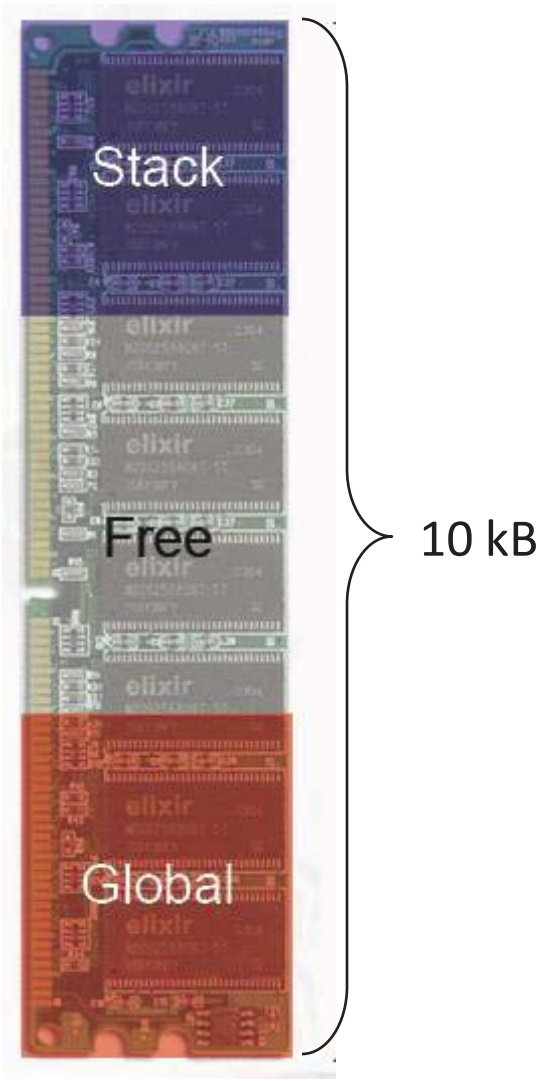
watch out for data races

Memory Model

- Static memory allocation
  - No heap (malloc)
  - No function pointers
- Global variables
  - One namespace per component
- Local variables
  - Declared within a function
  - Saved on the stack

bye-bye complex data structures

- Conserve memory
- Use pointers, don‘t copy buffers



nesC – Hello World

```
module BlinkC {
  uses interface Timer<TMilli>
            as BlinkTimer;
  uses interface Leds;
  uses interface Boot;
}
implementation{
  event void Boot.booted() {
    call BlinkTimer.startPeriodic(1000);
  }
  event void BlinkTimer.fired() {
    call Leds.led0Toggle();
  }
}
```

- Blink the red LED every second
- On boot start a 1 second timer
- On timer fire (countdown at 0)
  - Toggle the state of the red LED
  - Reset the timer to 1 second

```
interface Timer<precision_tag> {

   event void fired();

   command void startPeriodic(…);
   command void startOneShot(…);
   command void stop();
   …
}
```

nesC – Hello World

```
configuration BlinkAppC{
}

implementation {

  components MainC, BlinkC,
LedsC;
  components new TimerMilliC()
                as Timer0;

  BlinkC.Boot -> MainC.Boot;

  BlinkC.BlinkTimer -> Timer0;
  BlinkC.Leds -> LedsC.Leds;
}
```

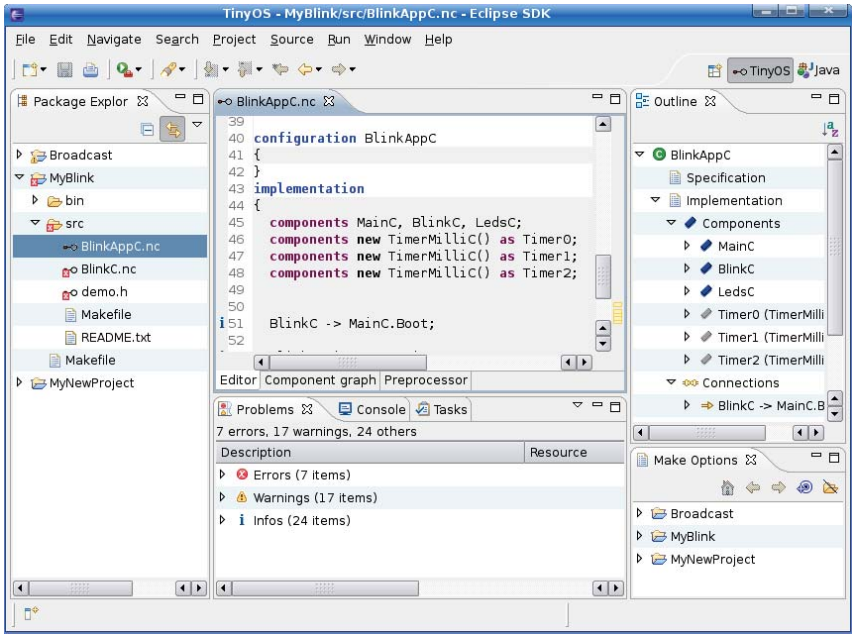
```
module BlinkC {
  uses interface Timer<TMilli>
            as BlinkTimer;
  uses interface Leds;
  uses interface Boot;
}
implementation {
  event void Boot.booted() {
    call BlinkTimer.startPeriodic(1000);
  }
  event void BlinkTimer.fired() {
    call Leds.led0Toggle();
  }

}
```

Sounds difficult?

- Code skeleton is provided for both exercises
  - Only a few line of codes are missing (your task)

- TinyOS Eclipse IDE will make your life easier (http://tos-ide.ethz.ch)
  - Error detection, Syntax highlighting, Code completion
  - One click compiling & flashing



Final Remarks

- Code skeletons for both applications are provided on the lab PC. All software required during the lab is already pre-installed.

- The lab work place is in the ETL building (ETL F29).
  Keys must be fetched in our office ETZ G64.1 when your lab slot starts.

➡ Register for your lab time slot on the course website

The End

- Thanks to Pascal von Rickenbach & Nicolas Burri for many of the slides