MaldOS:
a Moderately Abstracted Layer for Developing Operating Systems

Mattia Maldini

Relatore: Renzo Davoli

Università di Bologna

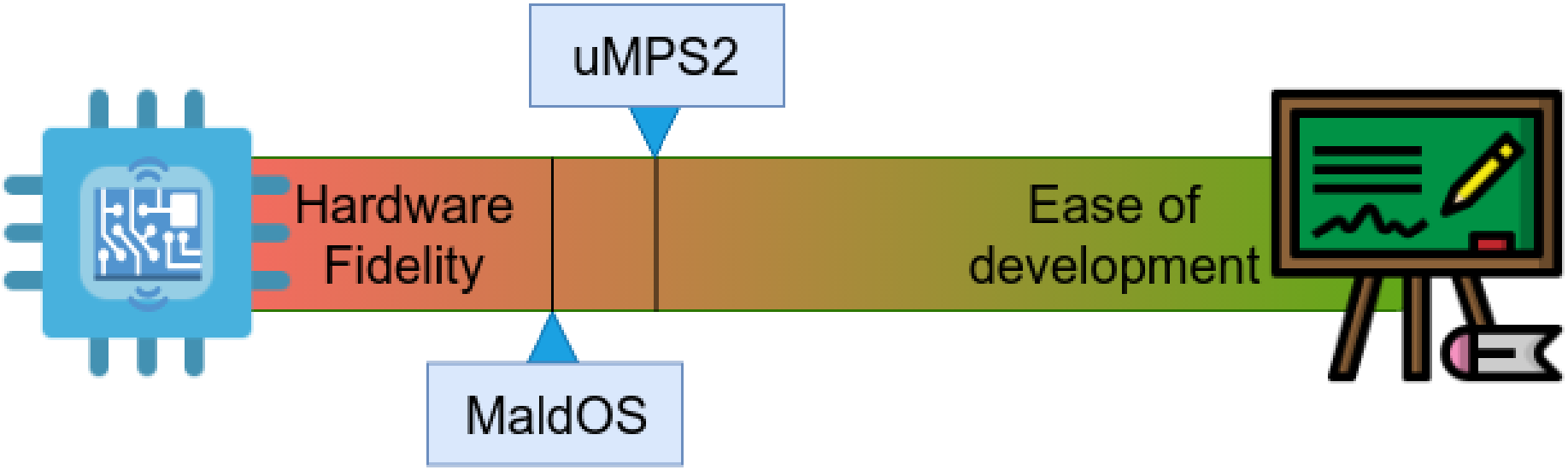
Sessione di laurea 14 marzo 2019

Structuring an Operating Systems course can be a thorny task.

Topics like scheduling, parallel programming, virtual memory and context switching are hard to grasp via a purely abstract approach, and developing a concrete example is inherently non trivial.

Examples might be implemented as user space programs, but they would lose some details that deeply characterize kernel development. On the other end one could work, bare metal, on a real processor, but this entails many nuances unnecessary for students.

A previous solution to this issue was the μMPS family of emulators; this work proposes a slight shift in the form of an abstraction layer allowing for easier development on real hardware.



Tradeoffs

μMPS2 allows students to work in a simplified environment while still being faithful to the real MIPS architecture, providing an interesting experience. However

- it is still abstract work, as the resulting OS will never run on a real device
- μMPS is an ad-hoc emulator
- Its debugging facilities lack step-by-step execution of C code
- MIPS is a dated architecture

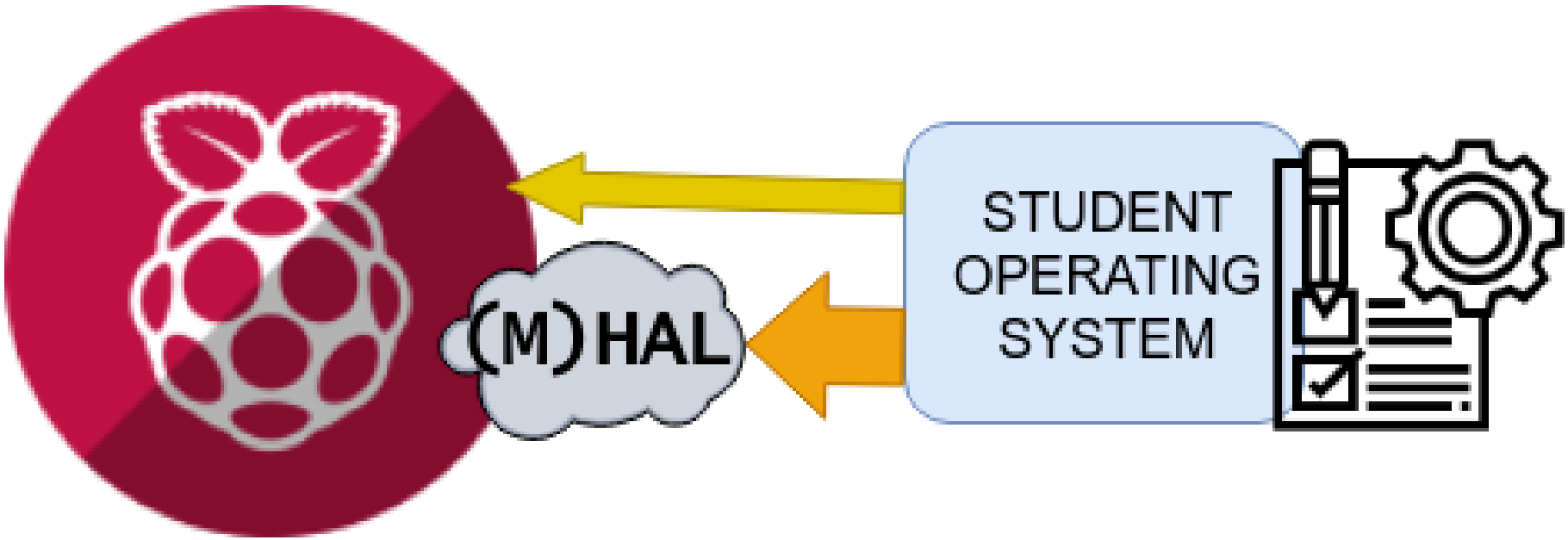
Tradeoffs

MaldOS is an abstraction layer devised to reproduce an μMPS-like simplified environment on a real device and architecture, the Raspberry Pi 3 and ARMv8.

- it is an extremely practical example, and the result can be run on a real machine
- it relies on general purpose software (namely, Qemu and GDB)
- its debugging facilities are less specific but more powerfull (full GDB support)
- ARM is a more modern and widespread environment

Bare Metal

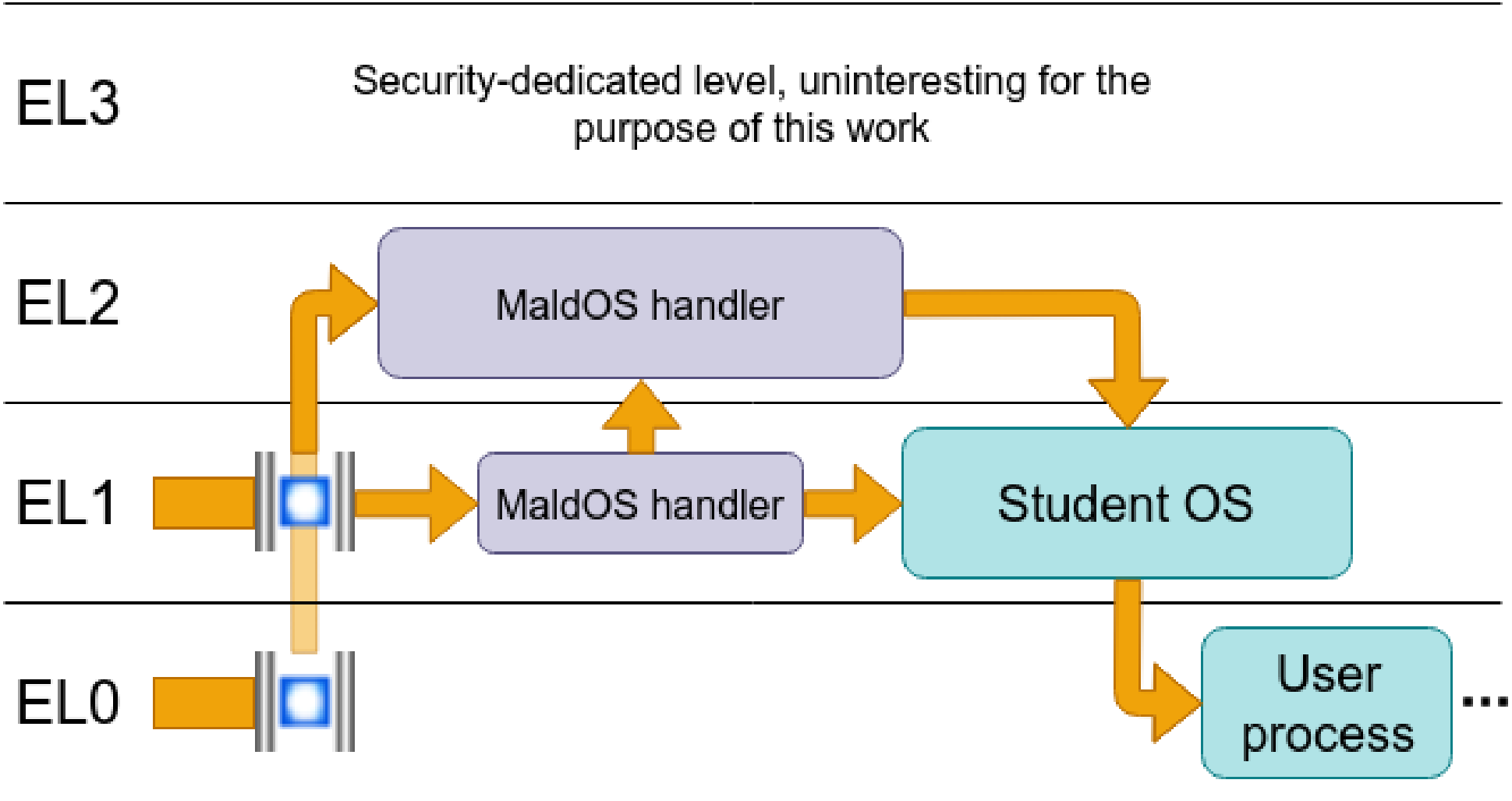
The problem with a real architecture is the intrinsic complexity of hardware management. The abstraction layer softens the approach by partially taking care of initialization and configuration procedures.



Initialization

- A proper linker script is setup beforehand
- Move to the correct level of execution
- Prepare memory structure (stack pointers and such)
- Boot all four kernels and park three
- Prepare the C environment (zero bss section)
- Configure and initialize all real devices (UART, screen, SD)

Interrupt Management



Emulated Devices

MaldOS creates μMPS-style emulated devices with an ideal interface. Thanks to the mailbox interface the integration is seamless.

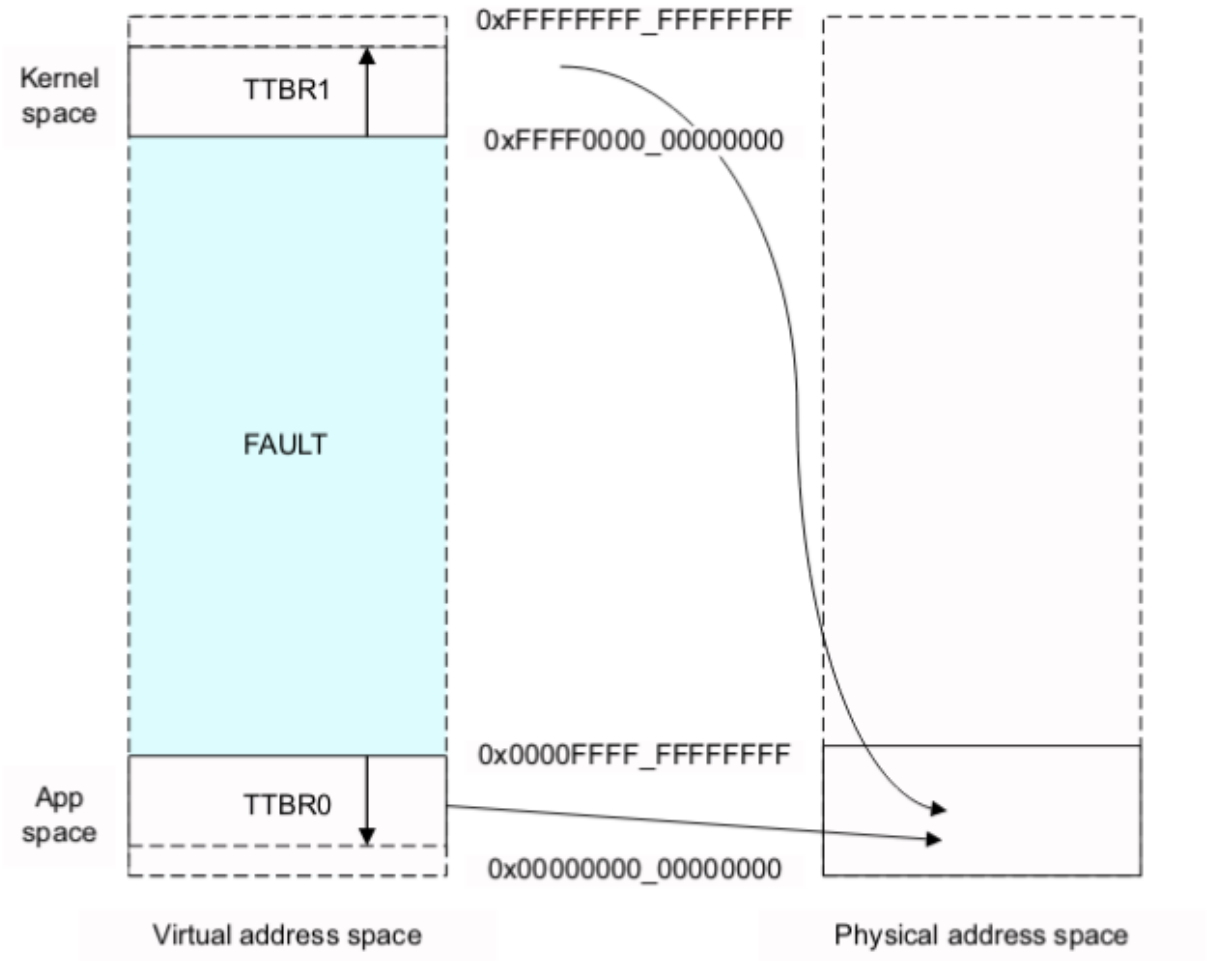
Virtual Disk device registers
status register
command register
data 0 register
data 1 register
mailbox register

EMMC device registers
second argument register
block size count register
first argument register
command register
response registers (4)
status register
control register (2 of 3)
interrupt register
interrupt mask register

... 11 more

Virtual Memory

The ARMv8 MMU is remarkably flexible, so the virtual memory configuration is left mostly untouched. Students are assisted by changing page tables every time the kernel context switched in.



Qemu and Debugging