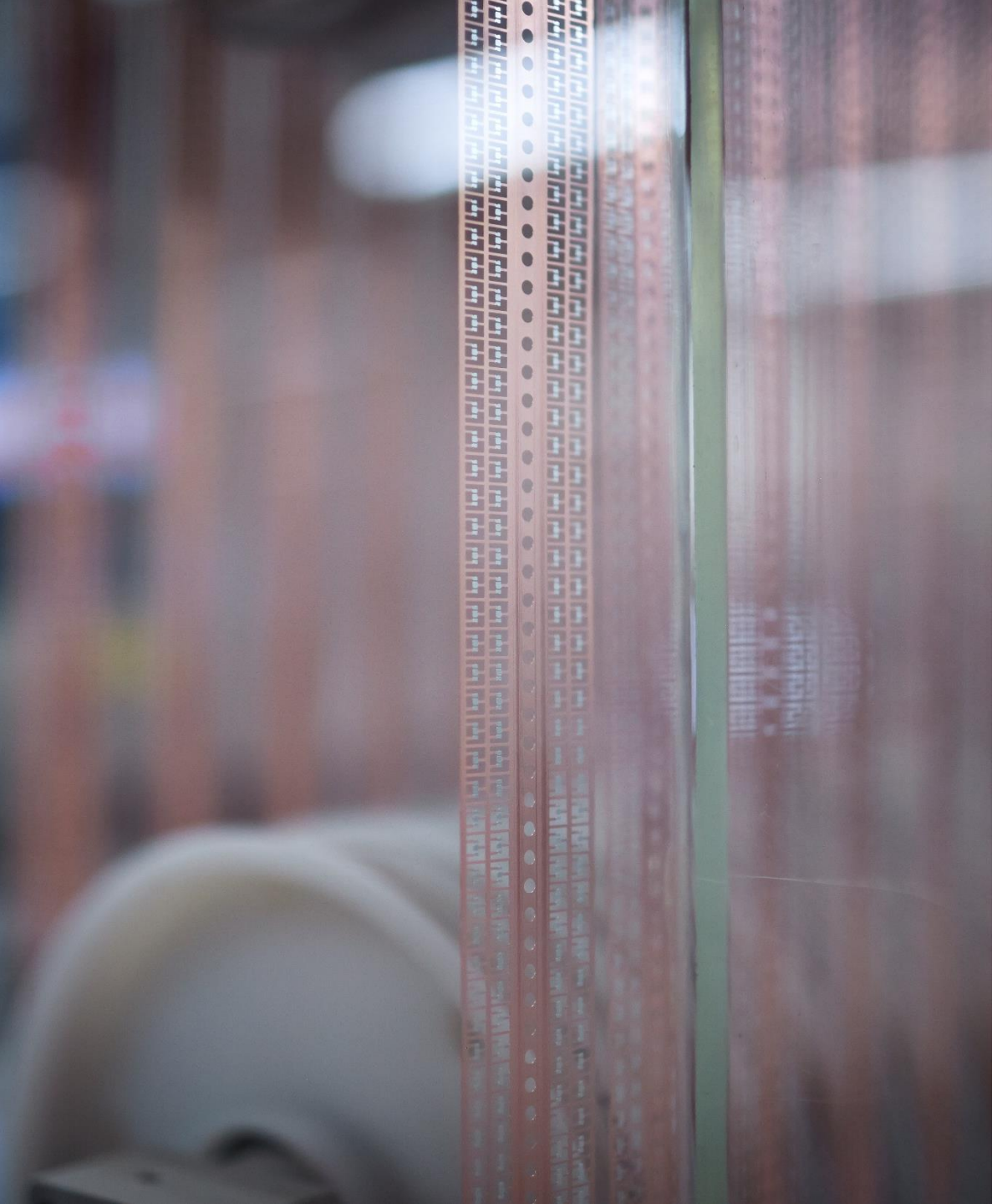
HOW TO INTEGRATE NFC FRONTENDS IN LINUX

WEBINAR SERIES:
NFC SOFTWARE INTEGRATION

JORDI JOFRE
NFC READERS
NFC EVERYWHERE
14/09/2017

SECURE CONNECTIONS
FOR A SMARTER WORLD



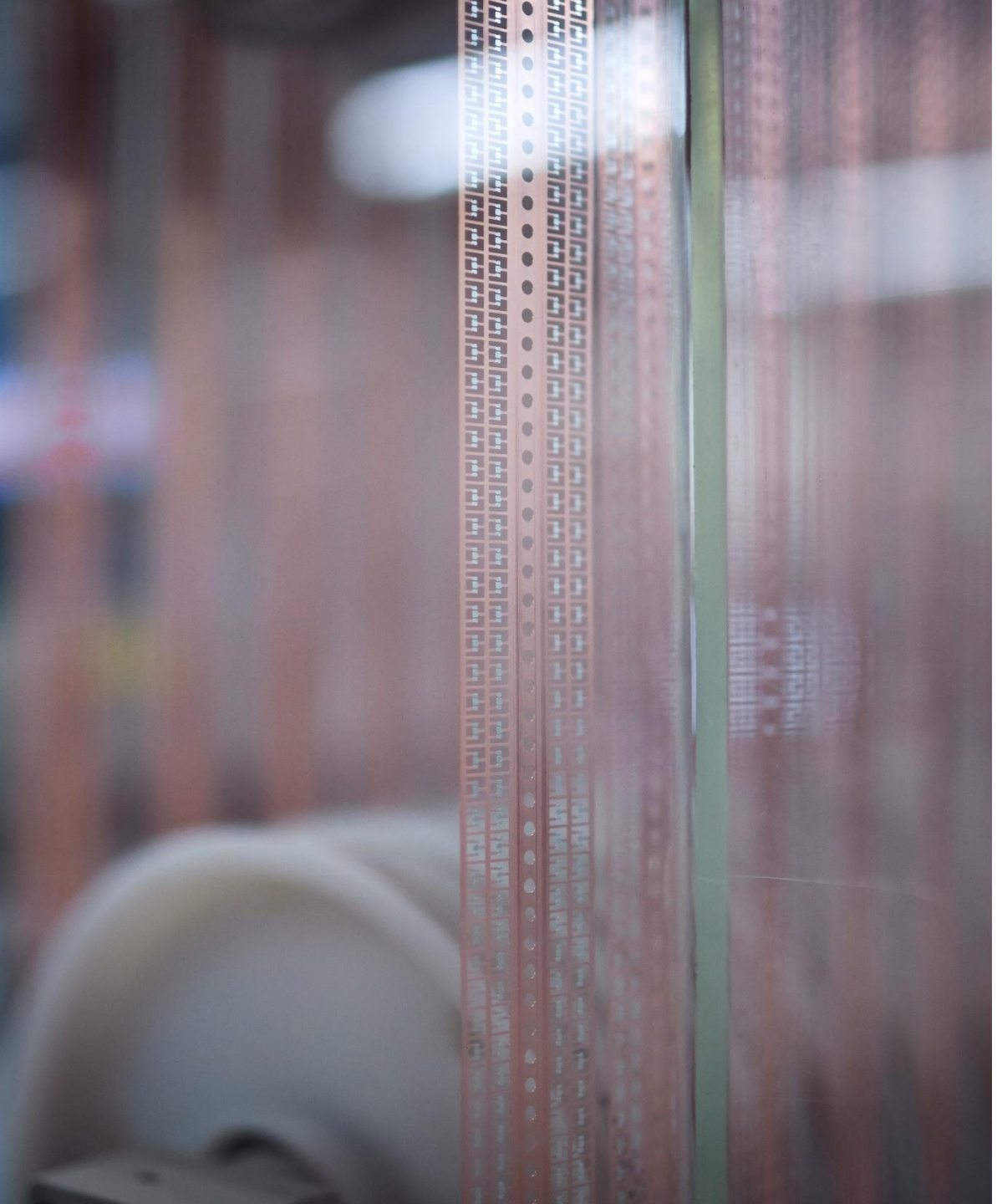
Agenda

NFC software integration webinar series

Session I, 14th September
How to integrate NFC frontends in Linux.

Session II, 28th September
How to integrate NFC controllers in Linux.

Session III, 11th October
How to port the NFC Reader Library to K64F.



Agenda

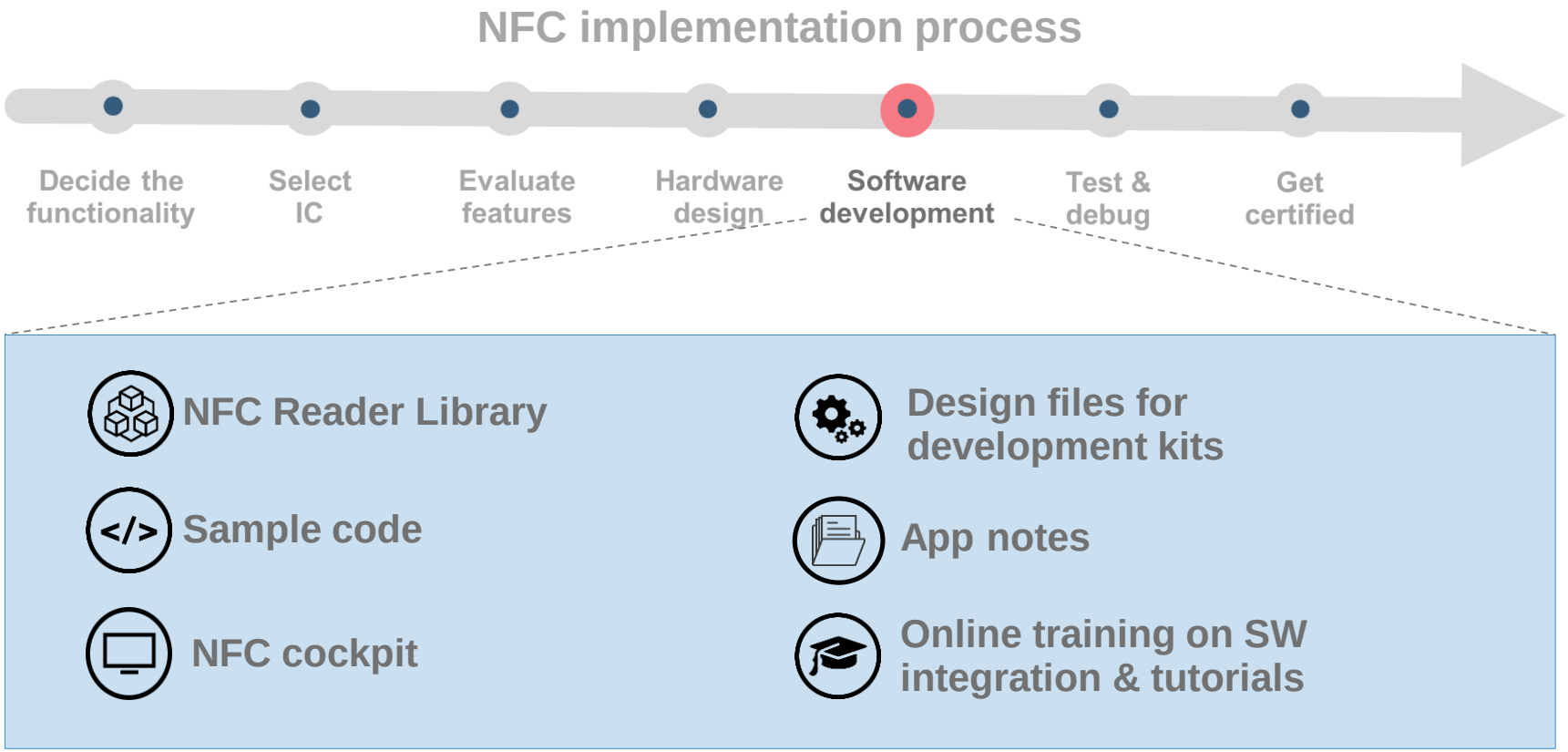
NFC software integration webinar series

Session I, 14th September
How to integrate NFC frontends in Linux.

- NFC readers software development design-in support.
- NFC Frontend concept.
- Linux OS architecture & NFC Reader Library integration in Linux.
- Host interface latency analysis in Linux.
- Wrap up & Q&A.

NFC readers software development design-in support

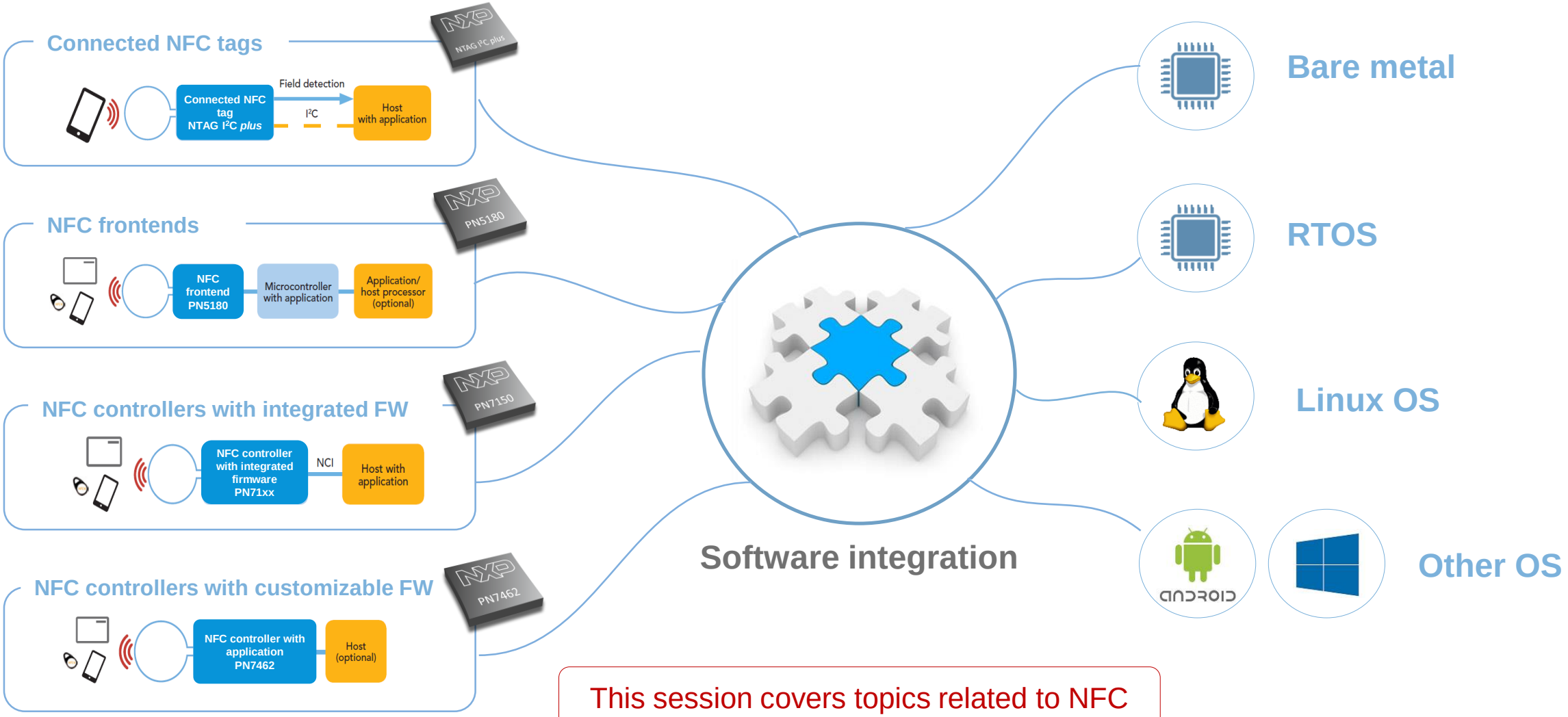
NXP’s software development support



You can re-use design of NXP development boards and sample code examples to speed up your SW development tasks.

Training Mobile Knowledge

NXP software support for integration into any platform



This session covers topics related to NFC frontend software integration in Linux

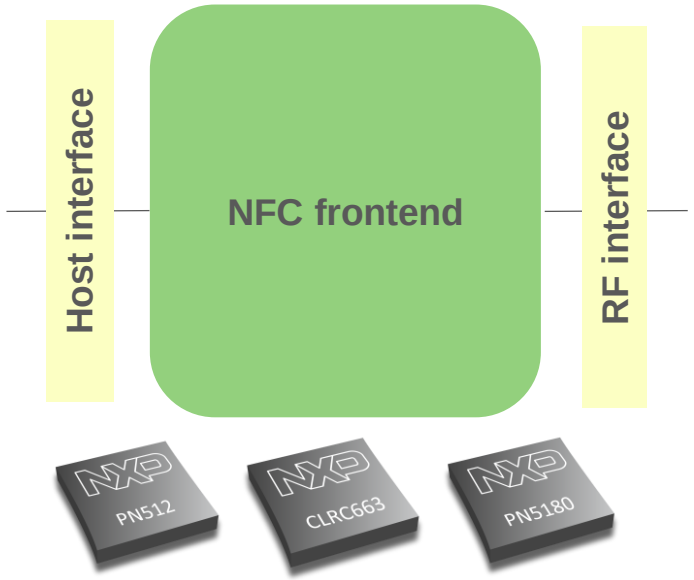
Training Mobile Knowledge

NFC Frontend concept

NFC frontend offers a host interface and a contactless interface

Host interface

- This register interface is a low level access to the contactless interface providing full access to this IP.
- This could be a direct CLIF-mapped interface (CLRC663, PN512) or a software emulated register interface (PN5180).
- The host controller uses the register access to the contactless interface:
  - to configure RF framing and signaling .
  - to finally transfer the RF digital protocol based blocks to/from a counterpart.



RF interface

- An NFC frontend is an RF transceiver enabling the contactless communication.
- It deals with the signal modulation and handles the data transmission through the RF interface.
- The NFC frontend needs to be selected according to application requirements:
  - RF performance
  - RF protocols
  - NFC modes of operation
  - Host interfaces
  - Power consumption
  - Device to interact with
  - Others…

NFC frontends expose a ‘register interface’ towards the host controller through the host interface

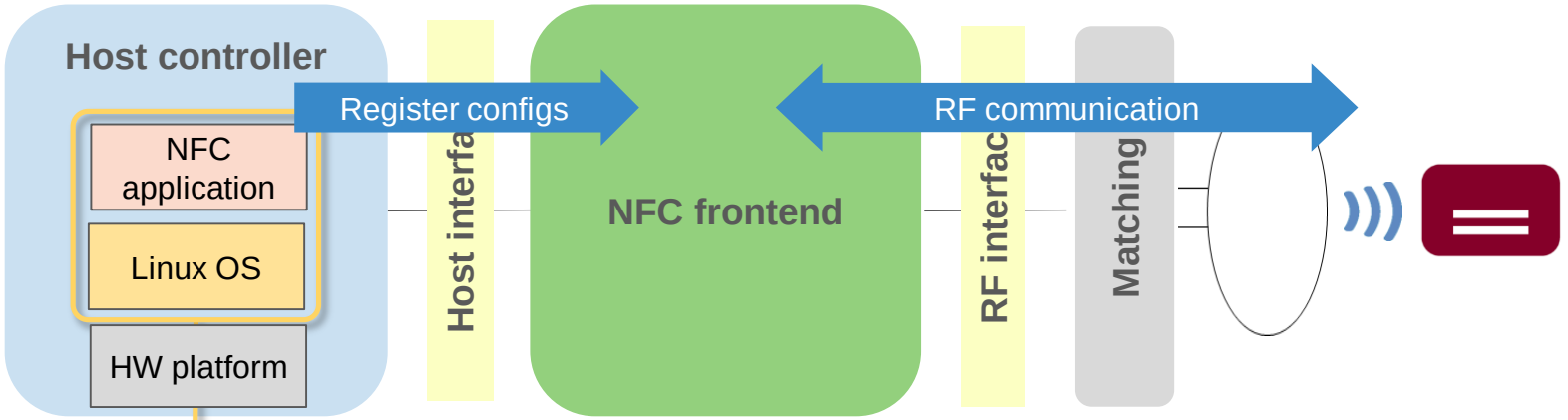
Training Mobile Knowledge

NFC frontend is controlled by the external host controller SW

Host controller

- Contains the software implementing the application logic
- The RF digital protocols are implemented on the host controller
- The host controller platform needs to be selected according to system requirements:
  - Memory requirements
  - Clock frequency
  - MCU architecture
  - Host interfaces
  - Power consumption
  - GPIOs and other peripherals

The host controller drives and controls the NFC frontend according to register settings configuration



TODAY: NFC application runs on Linux OS system

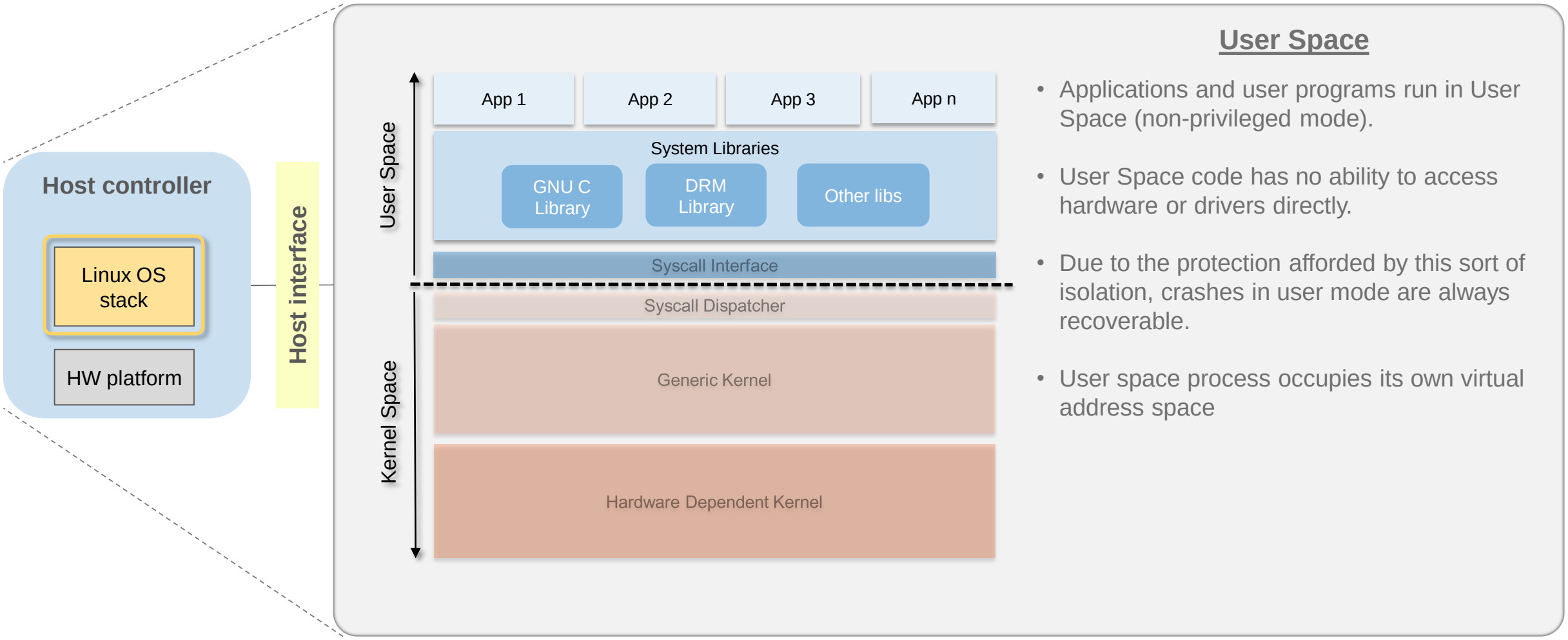
The NFC Reader Library can be installed on a generic GNU/Linux platform*

*Support on Raspberry Pi platform provided as reference.

Training Mobile Knowledge

Linux OS architecture

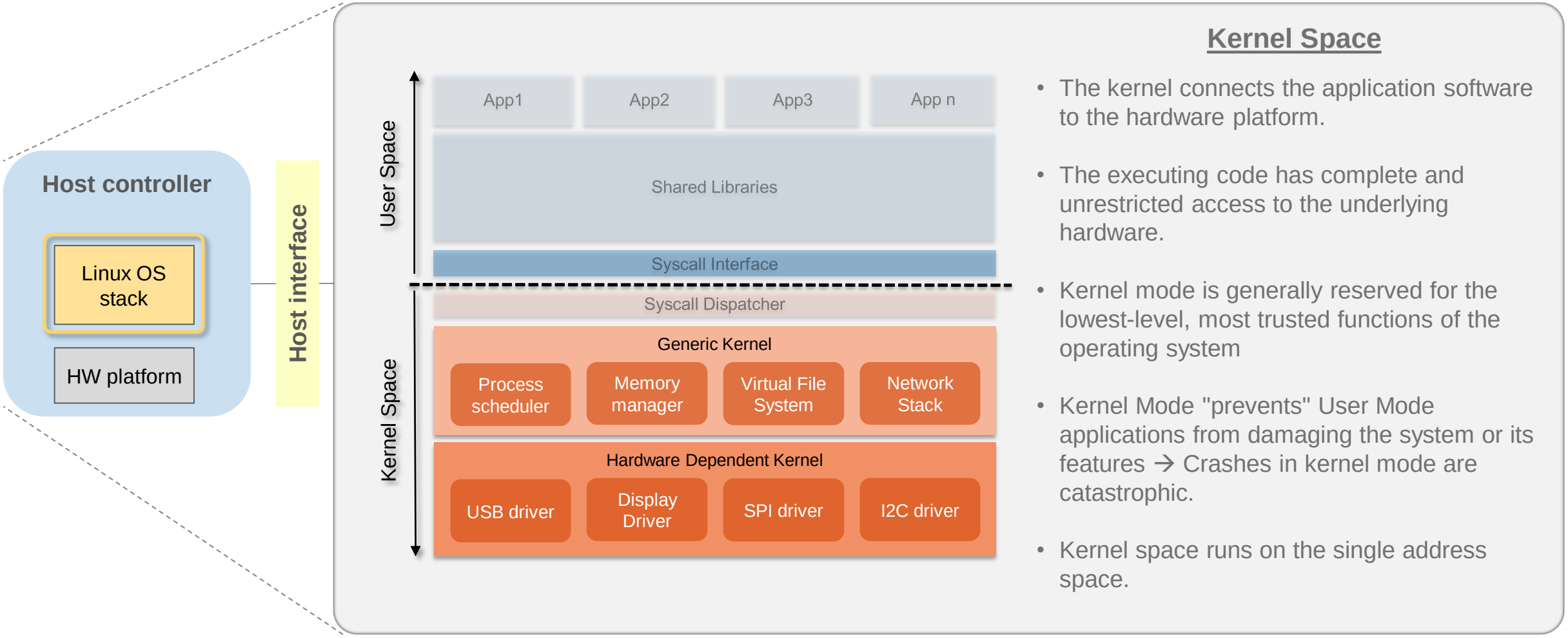
Host controller SW: Linux OS architecture - User space



User Space

- Applications and user programs run in User Space (non-privileged mode).
- User Space code has no ability to access hardware or drivers directly.
- Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable.
- User space process occupies its own virtual address space

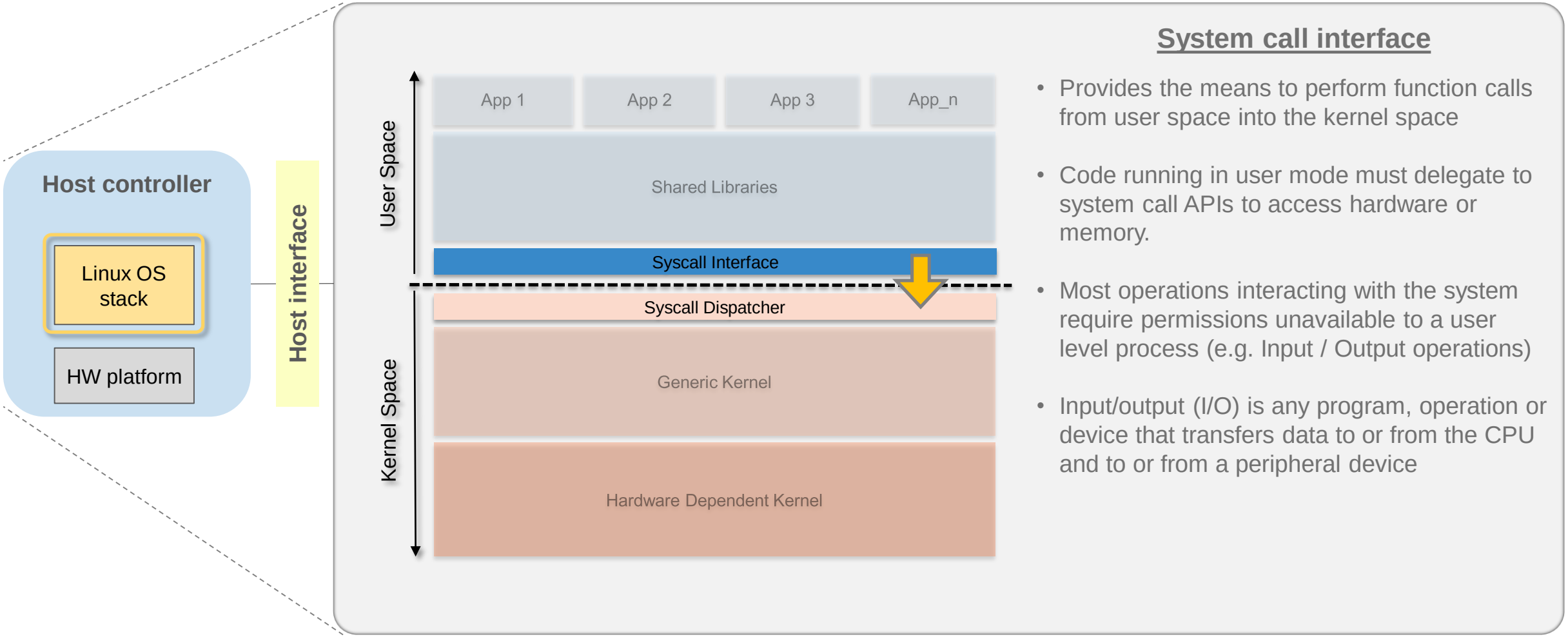
Host controller SW: Linux OS architecture - Kernel space



Kernel Space

- The kernel connects the application software to the hardware platform.
- The executing code has complete and unrestricted access to the underlying hardware.
- Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system
- Kernel Mode "prevents" User Mode applications from damaging the system or its features → Crashes in kernel mode are catastrophic.
- Kernel space runs on the single address space.

Host controller SW: Linux OS architecture – System call interface

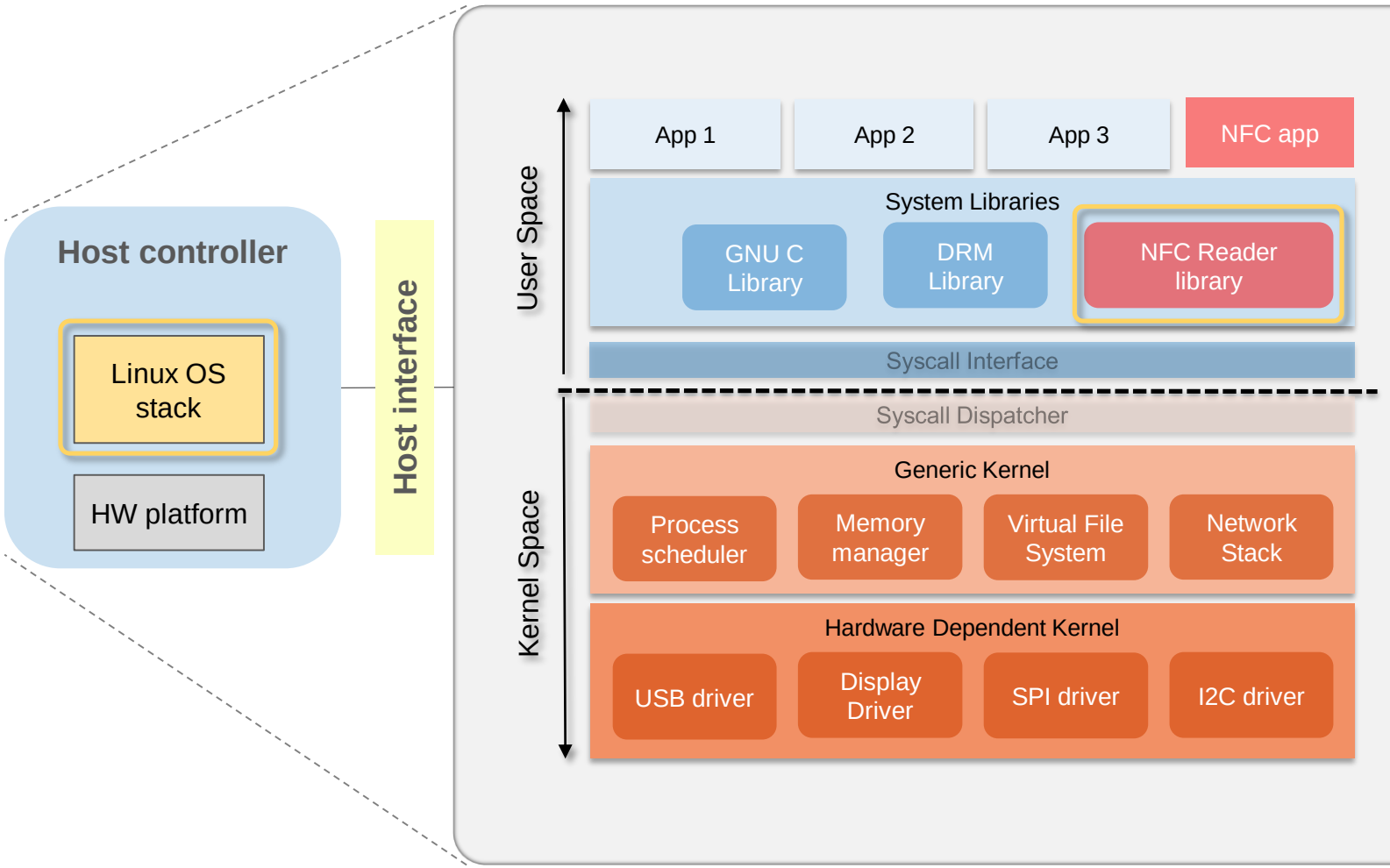


System call interface

- Provides the means to perform function calls from user space into the kernel space
- Code running in user mode must delegate to system call APIs to access hardware or memory.
- Most operations interacting with the system require permissions unavailable to a user level process (e.g. Input / Output operations)
- Input/output (I/O) is any program, operation or device that transfers data to or from the CPU and to or from a peripheral device

Integrating the NFC Reader Library in Linux

NFC Reader Library: The SW stack for developing NFC applications



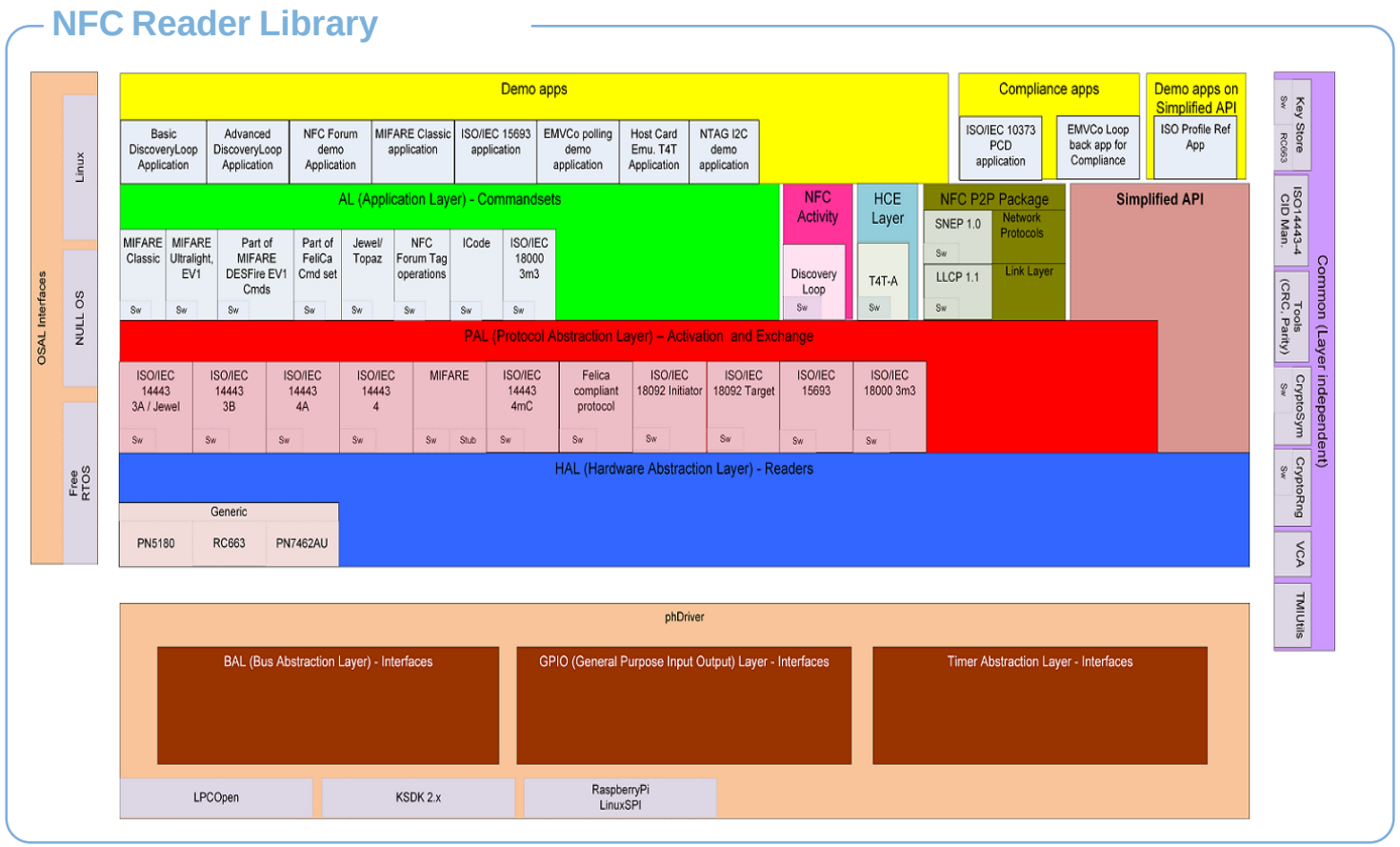
NFC Reader Library SW integration

- The NFC Reader Library provides an API, including everything you need to deploy for NFC applications:
  - Host platform drivers
  - RF digital protocols
  - Full feature set according to NFC Forum
  - NXP NFC frontends hardware drivers
  - MIFARE® product-based cards and NFC Forum tag operation
- The NFC Reader Library runs within the **User Space.**
- The customer NFC app is built on top of the NFC Reader Library, taking advantage of the offered API

The NFC Reader Library is the NXP software stack to develop NFC applications and there is an existing version for Linux OS architecture!

Training Mobile Knowledge

NFC Reader Library support for multiple products and platforms



Info and more information: www.nxp.com/pages/:NFC-READER-LIBRARY

Supported products:*

- CLRC663 *plus*
- PN5180
- PN7462AU

Supported dev boards:*

- CLEV6630B
- PNEV5180B
- PNEV7462B

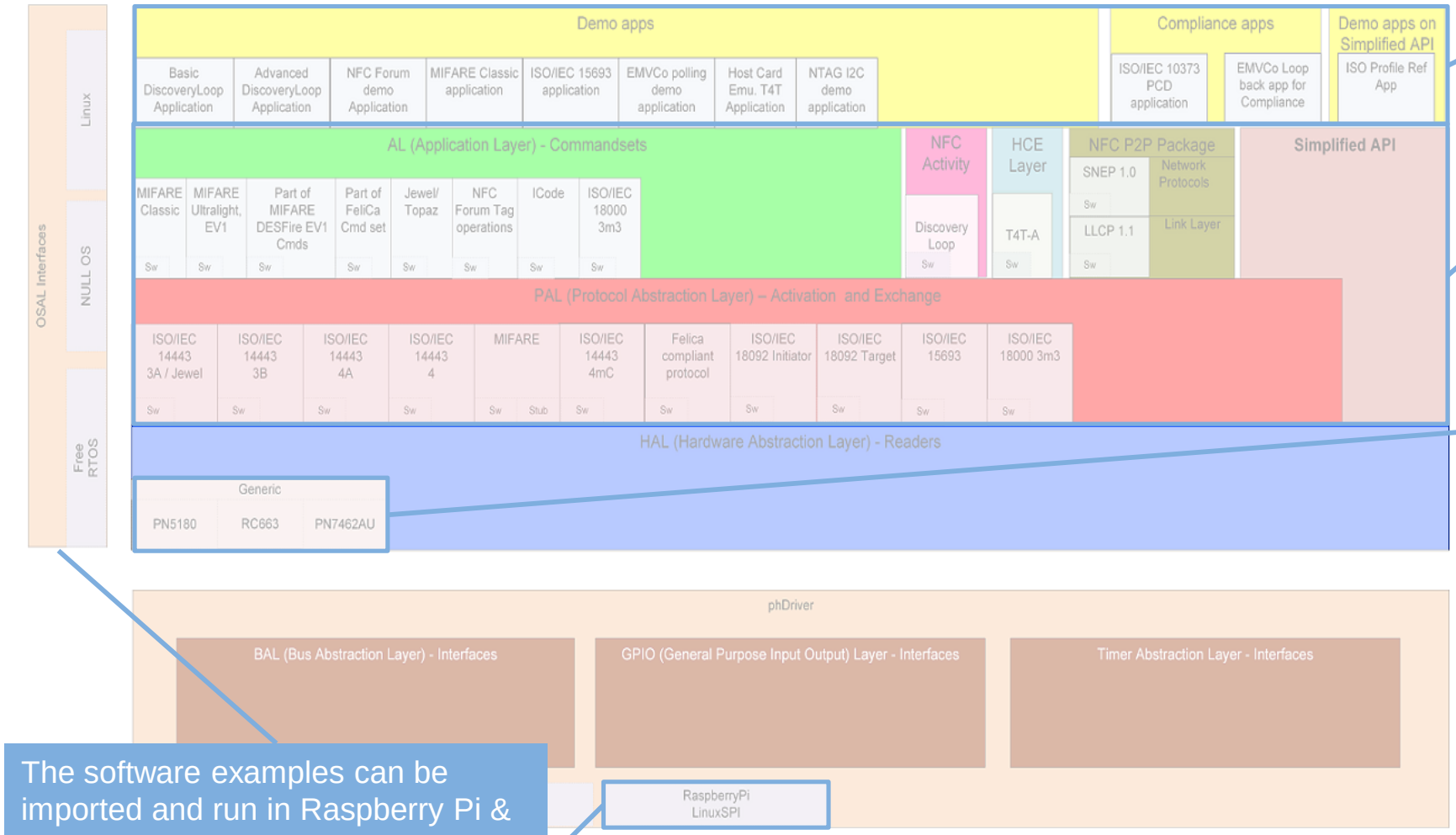
Supported platforms:*

- LPC1769
- FRDM-K82F
- ***Raspberry Pi Model 3 → Linux***
- … and portable to other MCUs and platforms.

The **NFC Reader Library** is everything you need to create your own software stack and application for a contactless reader

Training Mobile Knowledge

NFC Reader Library support for Linux



11 software examples available to be tested and re-used in Linux

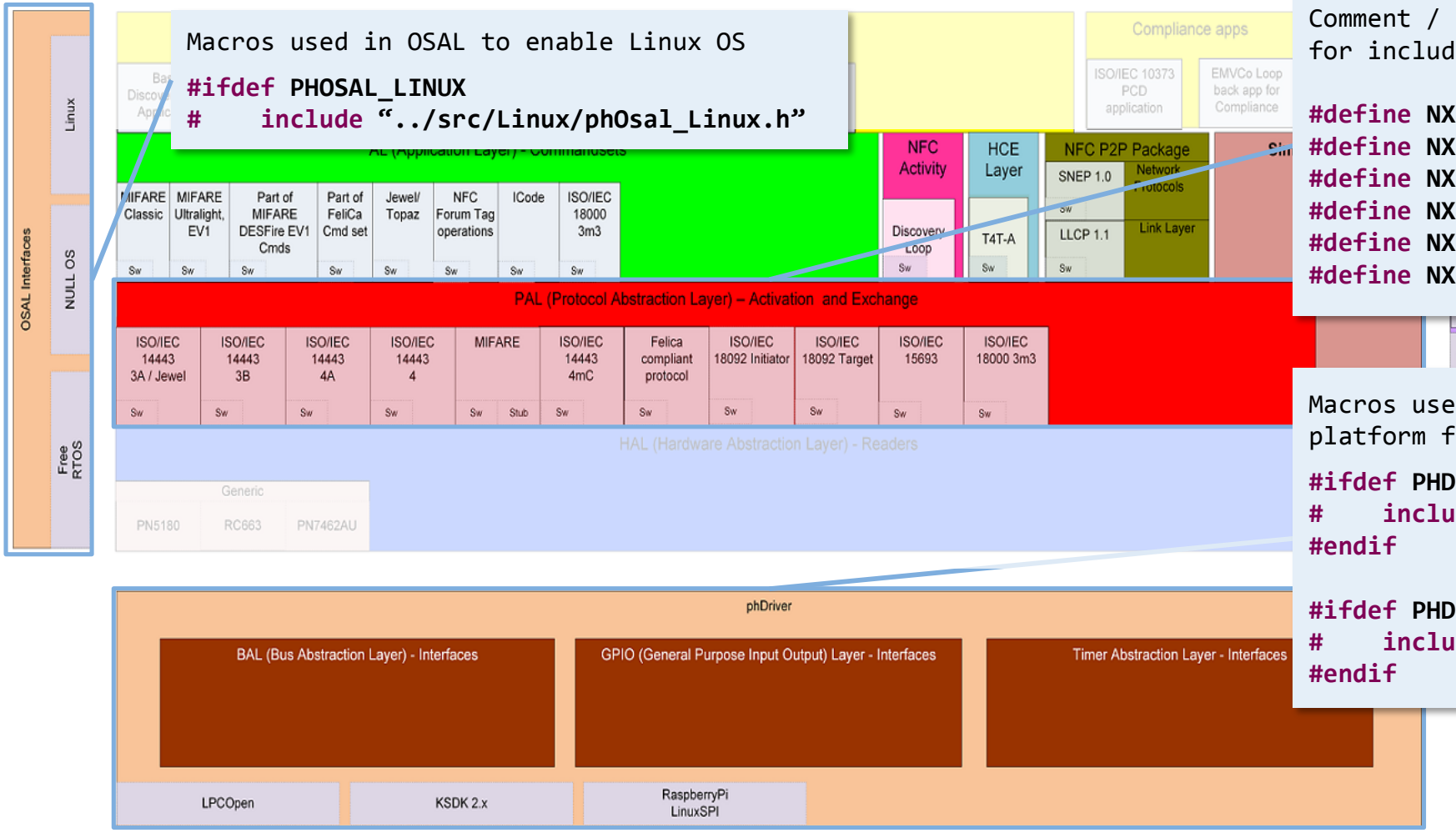
AL and PAL layers are **hardware and platform independent**, so they can be used in Linux as they are without any adaptation

HAL is **platform independent**, so NXP NFC readers can be used in Linux as it is without any adaptation

The software examples can be imported and run in Raspberry Pi & Linux without any adaptation.

Support for other platforms requires adaptations in OSAL and DAL.

NFC Reader Library - building the SW stack for Linux



Macros used in OSAL to enable Linux OS

```
#ifdef PHOSAL_LINUX
#    include "../src/Linux/phOsal_Linux.h"
```

Comment / uncomment the specific macro defined for including / excluding each SW component. E.g.

```
#define NXPBUILD__PHPAL_I14443P3A_SW
#define NXPBUILD__PHPAL_I14443P3B_SW
#define NXPBUILD__PHPAL_I14443P4A_SW
#define NXPBUILD__PHPAL_I14443P4_SW
#define NXPBUILD__PHPAL_MIFARE_SW
#define NXPBUILD__PHPAL_FELICA_SW
```

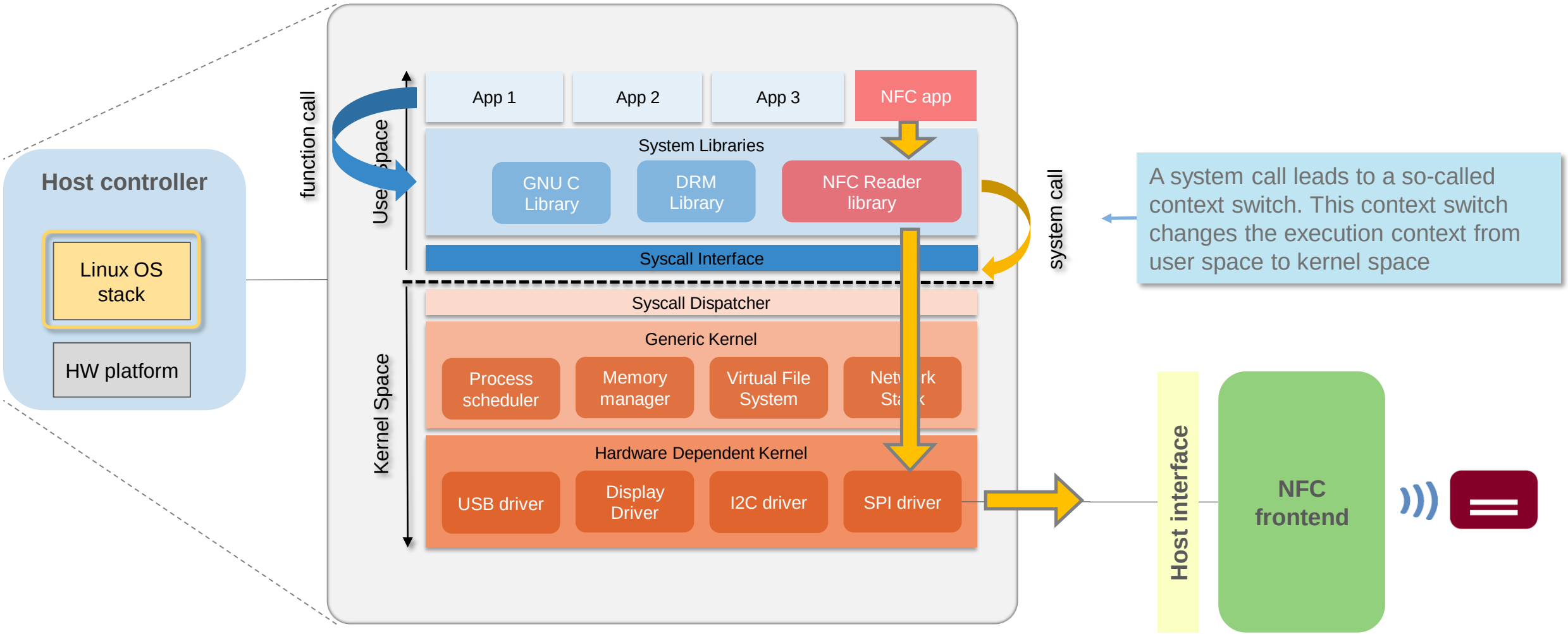
Macros used in DAL to enable Raspberry Pi platform for PN5180 and CLRC663

```
#ifdef PHDRIVER_PIPN5180_BOARD
#    include <Board_PiPn5180.h>
#endif

#ifdef PHDRIVER_PIRC663_BOARD
#    include <Board_PiRc663.h>
#endif
```


Training Mobile Knowledge

Host interface access on Linux systems

Linux based application: System call interface

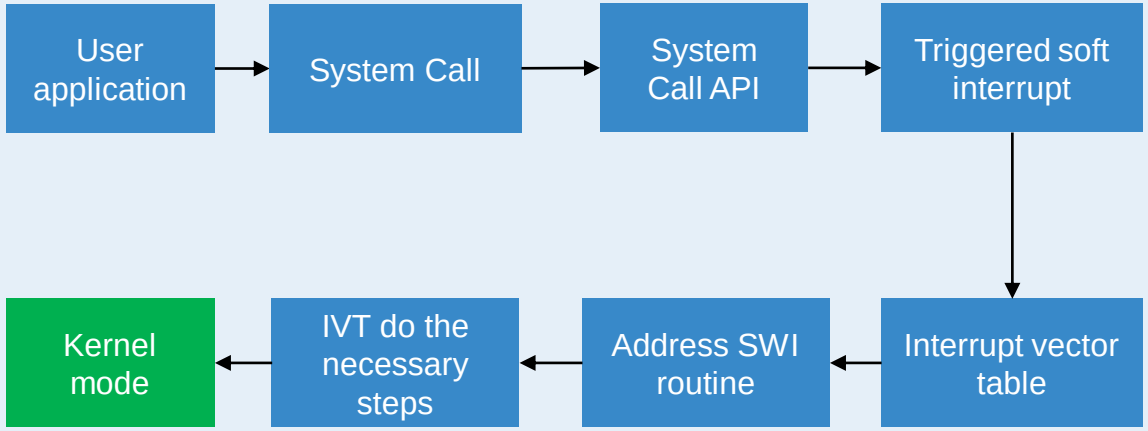


The NFC application needs to switch from User Space to Kernel Space for every SPI interface access

Training Mobile Knowledge

Transition between User mode and Kernel mode

Switching from User mode to Kernel mode



- User application initiates switching to kernel mode making a system call (e.g. open, read, write, etc)
- A software interrupt (SWI) is triggered
- The interrupt vector table launches the handler routine which performs all the required steps to switch the user application to kernel mode
- Start execution of kernel instructions on behalf of the user process.

Advantage:

- Well-defined interface.
- Horizontal separation: Avoids a crashing application crashing the whole system and protects system resources.

Disadvantage:

- Performance degradation: A system call is much **slower** than a direct function call

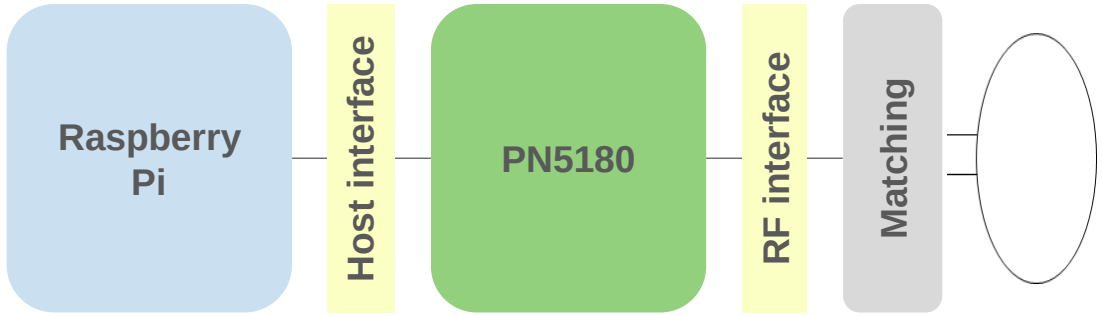
Could challenge the design of time-critical NFC applications

Training Mobile Knowledge

Latency analysis: Linux vs bare metal

Hardware setup

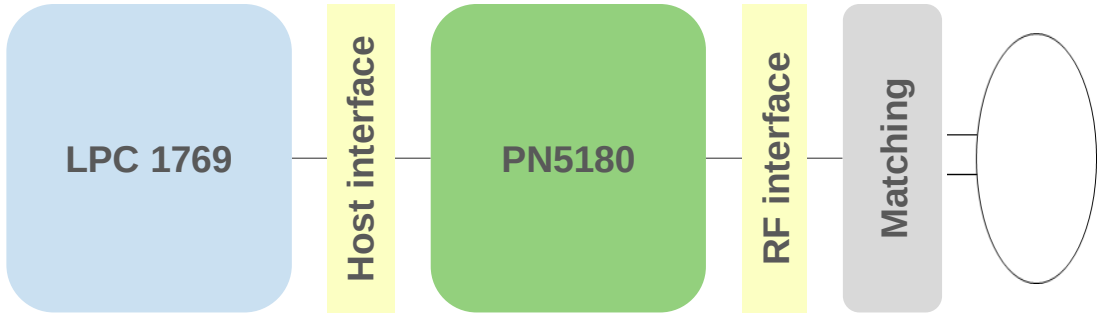
Linux setup



Linux hardware platform

- Raspberry Pi 3 Model B
  - 1.2 GHz 64-bit quad-core ARMv8 CPU
    - Limited to 1 Core @ 100 MHz (3 cores disabled)
  - 1 GB RAM
- PNEV5180B (with LPC bypassed)
  - SPI host interface

Bare metal setup



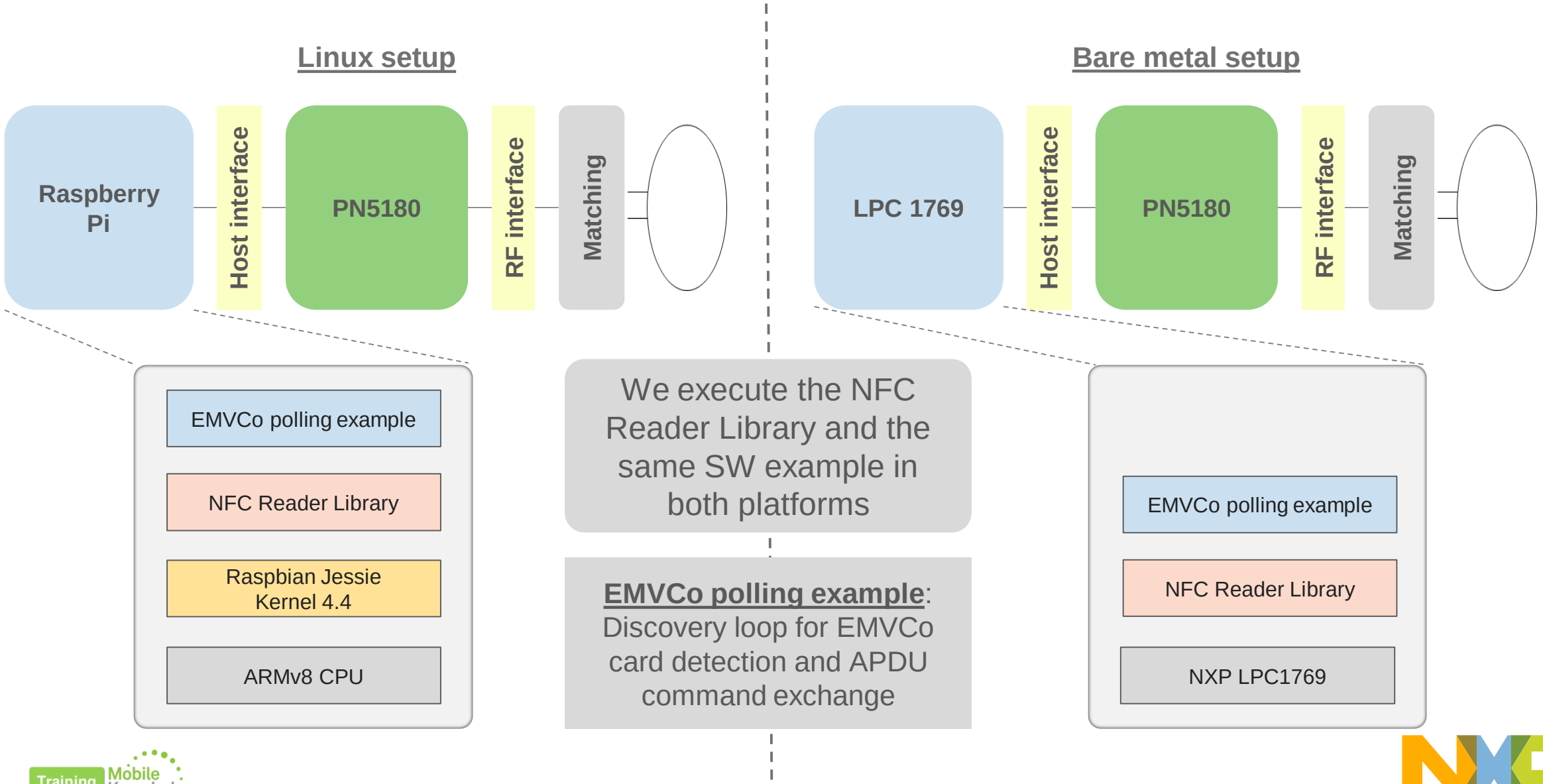
Bare metal hardware platform

- NXP LPC1769 uC
  - ARM 1 core @ 96 MHz
- LPC-Link2 connected for debugging
- PNEV5180B
  - SPI host interface

We limited Raspberry Pi clock and MCU cores to achieve a comparable setup with LPC1769

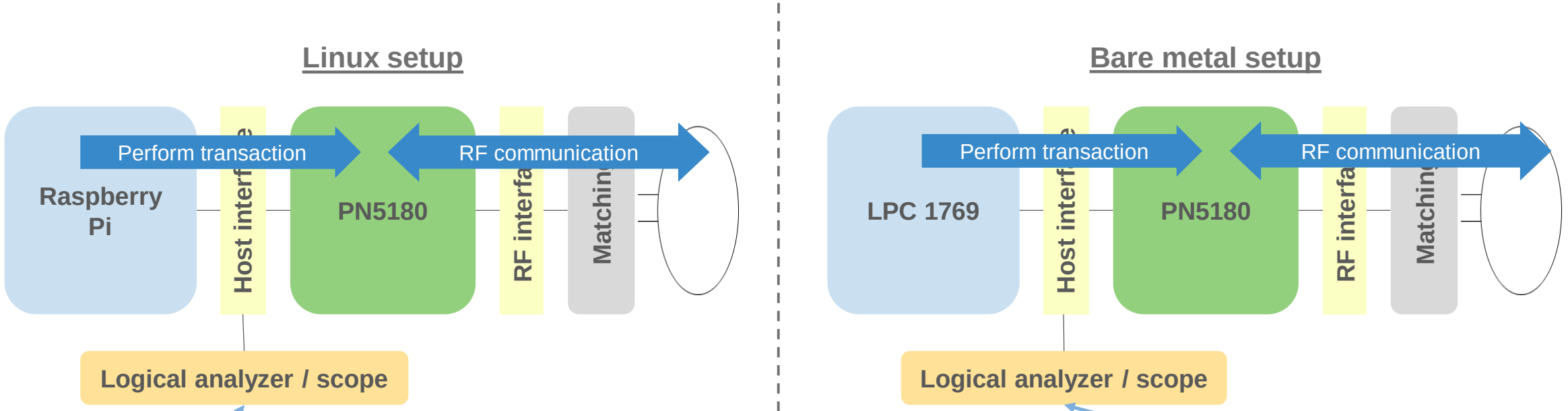
Training Mobile Knowledge

Software setup



Training Mobile Knowledge

Measurement setup



We use a logical analyzer / scope connected in the SPI interface between the Host controller & PN5180

Measurements conducted

- Time from GPIO toggle to SPI transfer
- Time between two SPI accesses
- Time for EMVCo polling initialization
- Time for EMVCo PPSE transaction

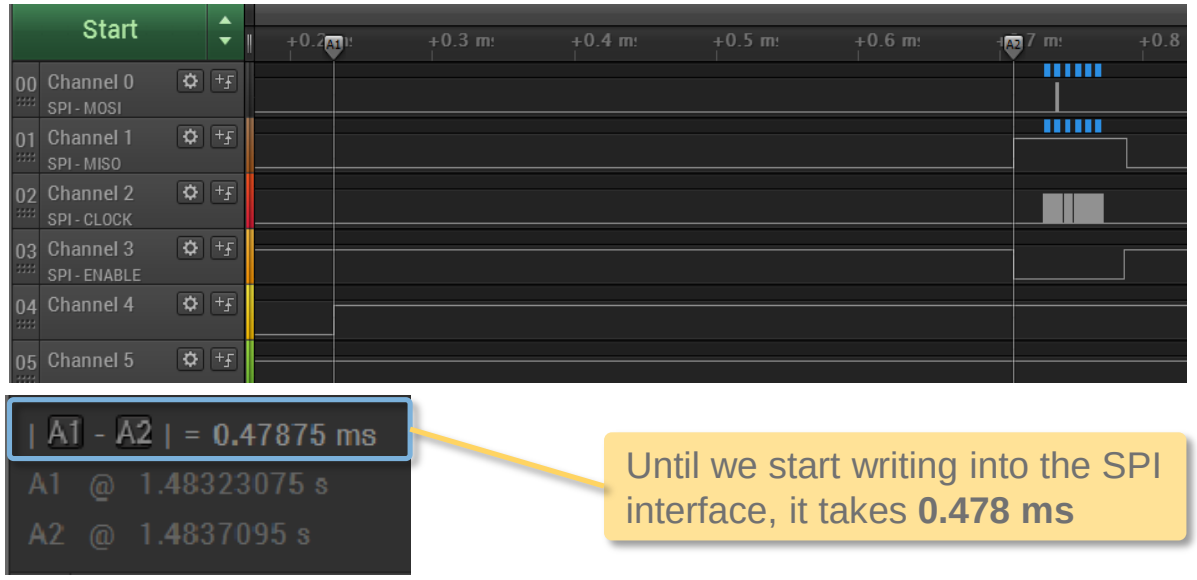
We use a logical analyzer / scope connected in the SPI interface between the Host controller & PN5180

We compared the results in the following slides

Training Mobile Knowledge

Measured time from GPIO toggle to SPI transfer

Linux setup

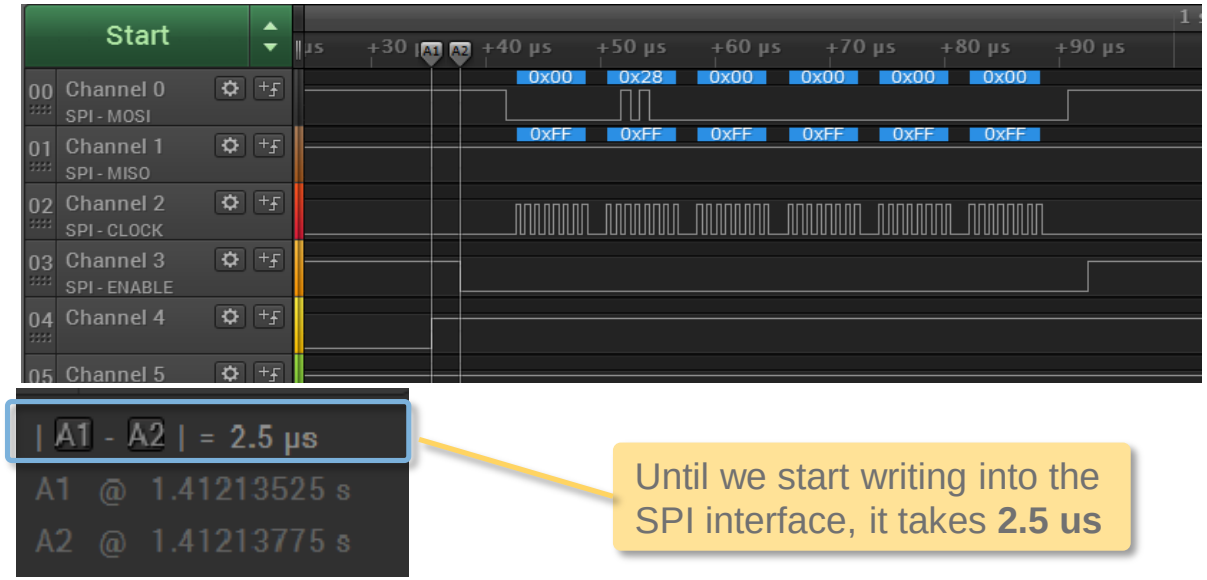


Until we start writing into the SPI interface, it takes **0.478 ms**

```
1. Set_GPIO(High)*;
2. phhalHw_PN5180_WriteRegister(…);
3. Set_GPIO(Low);
```

* Pseudo-code extracted from the real EMVCo polling source code example from the NFC Reader Library
* GPIO toggling execution takes less than 350us

Bare metal setup



Until we start writing into the SPI interface, it takes **2.5 us**

```
1. Set_GPIO(High)*;
2. phhalHw_PN5180_WriteRegister(…);
3. Set_GPIO(Low);
```

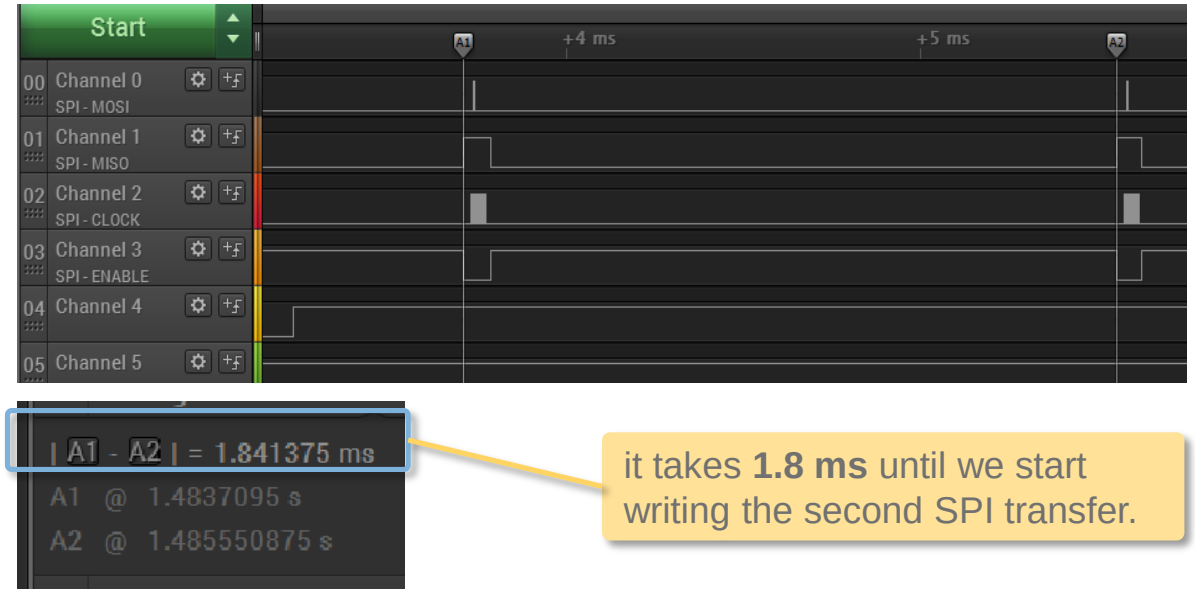
* GPIO toggling execution takes less than 3us

Linux setup is ~ 200x times slower than bare metal setup

Training Mobile Knowledge

Measured time between two SPI accesses

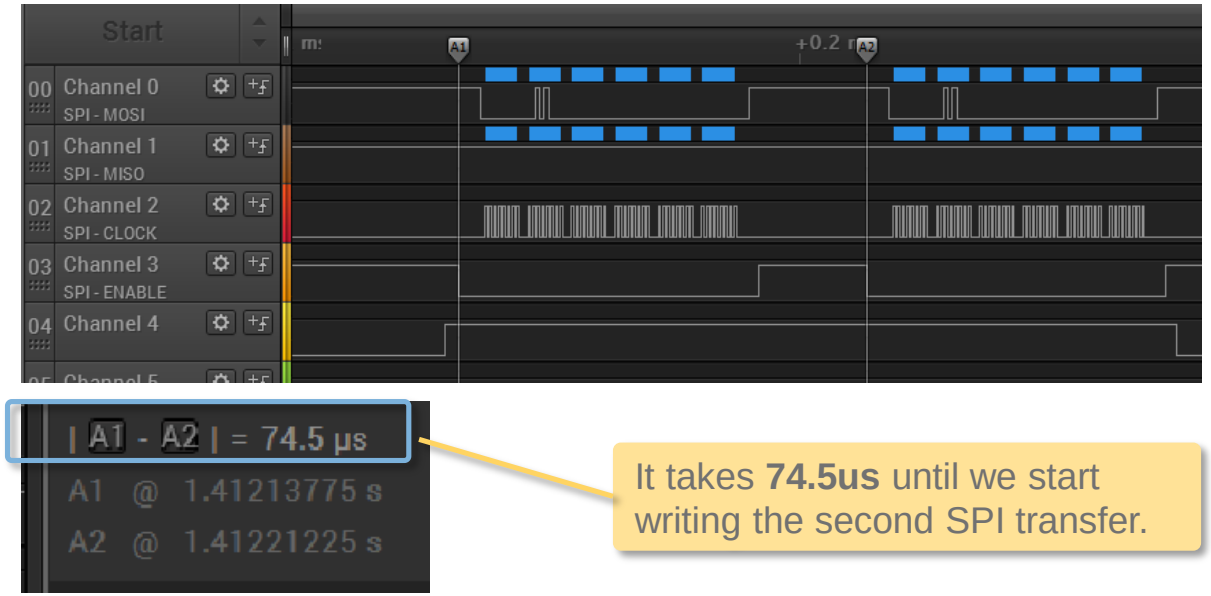
Linux setup



it takes **1.8 ms** until we start writing the second SPI transfer.

```
1. Set_GPIO(High)*;
2. phhalHw_PN5180_WriteRegister(…);
3. phhalHw_PN5180_WriteRegister(…);
4. Set_GPIO(Low);
```

Bare metal setup



It takes **74.5us** until we start writing the second SPI transfer.

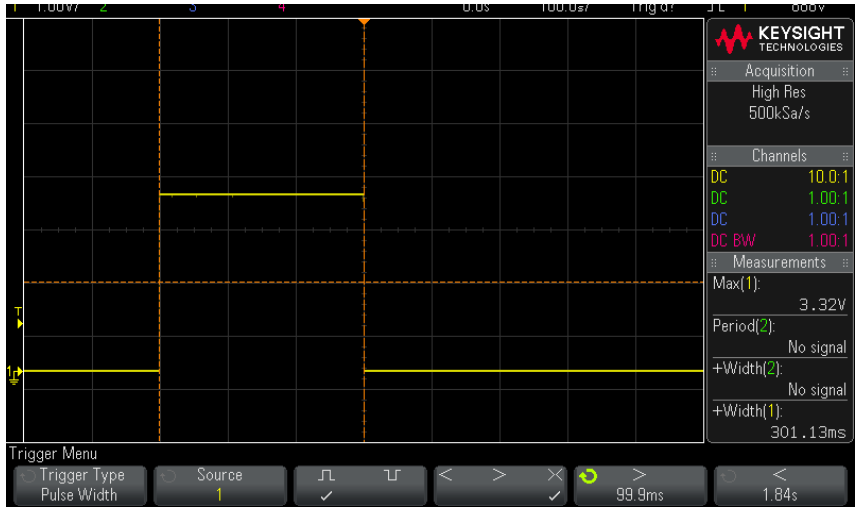
```
1. Set_GPIO(High)*;
2. phhalHw_PN5180_WriteRegister(…);
3. phhalHw_PN5180_WriteRegister(…);
4. Set_GPIO(Low);
```

* Pseudo-code extracted from the real EMVCo polling source code example from the NFC Reader Library

Linux setup is ~ 25x times slower than bare metal setup

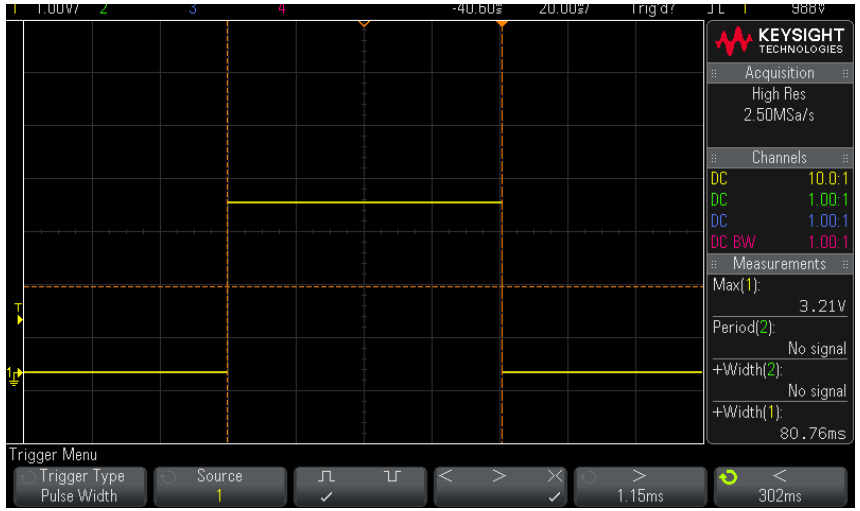
Measured time for EMVCo polling initialization

Linux setup



Measured time for 10 EMVCo polling inits.
1 init ~30.1ms

Bare metal setup



Measured time for 10 EMVCo polling inits.
1 init ~8.07 ms

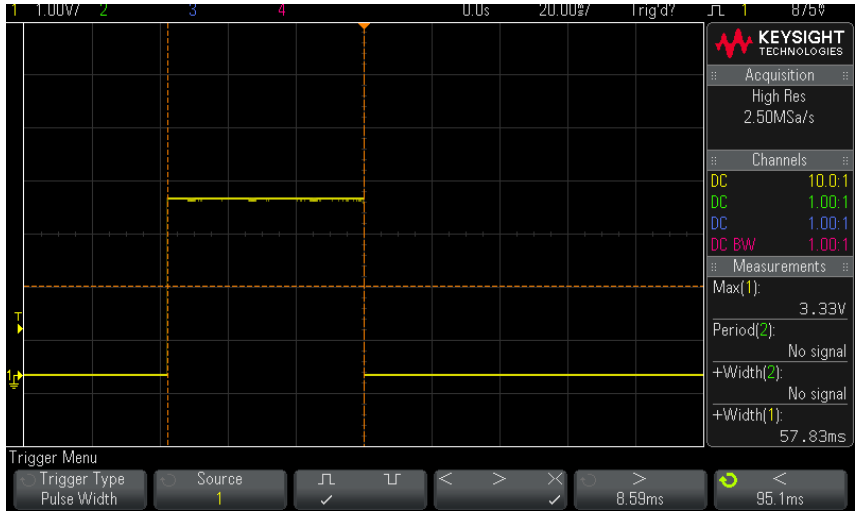
```
void Emvco_Polling(void * pHalParams)
Board_LED_Set(LED_NUM, 1);

for(i=0;i<10;i++)
{
    /* Load Emvco Default setting */
    status = LoadEmvcoSettings();
    CHECK_STATUS(status);

    /* Perform RF Reset */
    status = EmvcoRfReset();
    CHECK_STATUS(status);

    status = phhalHw_SetConfig(pHal, PHHAL_HW_CONFIG_SET_EMD, PH_OFF);
    CHECK_STATUS(status);
}

Board_LED_Set(LED_NUM, 0);
```

Linux setup is ~ 4x times slower than bare metal setup

* During the initialization, several registers are written. The process is repeated 10 times to get an average

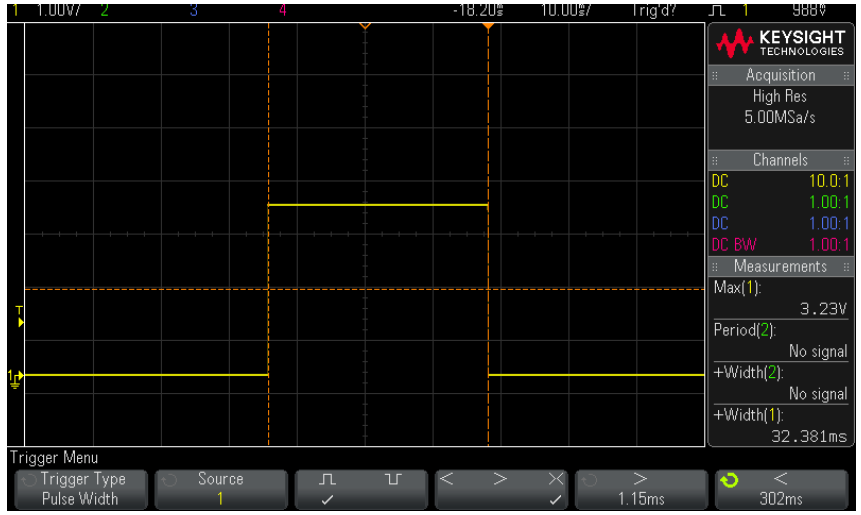
Measured time for EMVCo PPSE

Linux setup



Measured time for a EMVCo loopback transaction takes **57ms**

Bare metal setup



Measured time for a EMVCo loopback transaction takes **32ms**

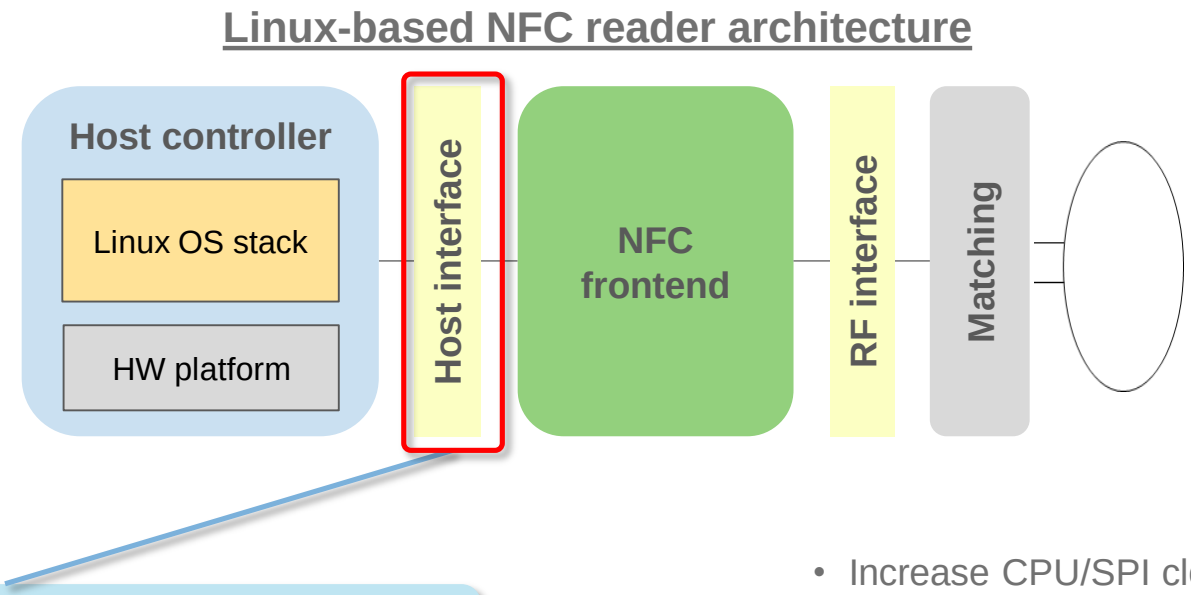
```
*phStatus EMVCoDataLoopBack(…){
1. Set_GPIO(High);
2. EMVCoDataExchange(…);
3. Set_GPIO(Low);
```

* Pseudo-code extracted from the real EMVCo polling source code example from the NFC Reader Library

Linux setup is ~ 2x times slower than bare metal setup

Overcoming Linux higher latency for time-sensitive applications

Recommendations to reduce Linux latency



- **High latency:** Access the SPI driver in Kernel space is slow.
- **High CPU load:** There is a lot of code involved just to write one register.

Solutions

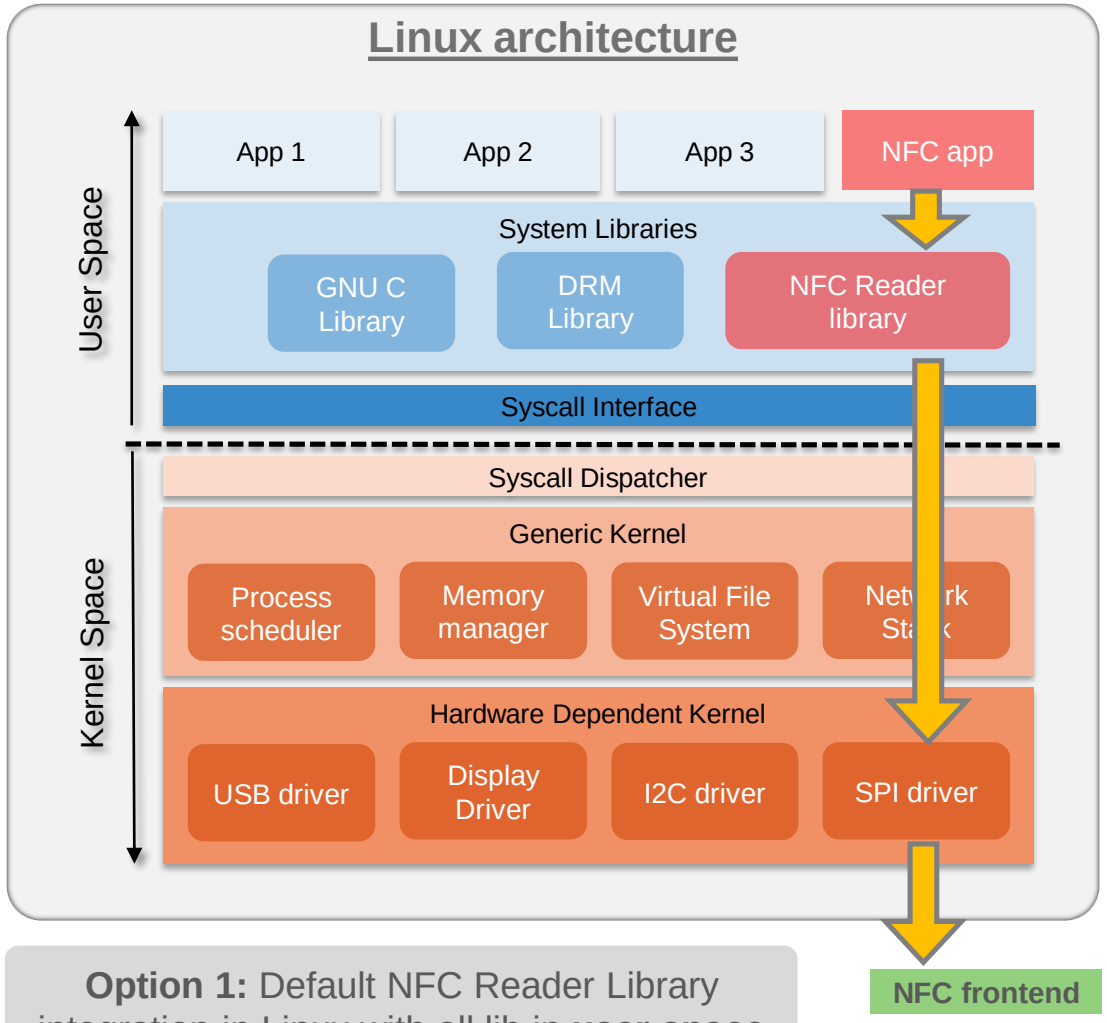
- Increase CPU/SPI clock as much as the MCU can process.
- Reduce SPI / host interface interactions as much as possible: Linux driver is optimized for few long transactions rather than lots of short ones

The most effective solution!!

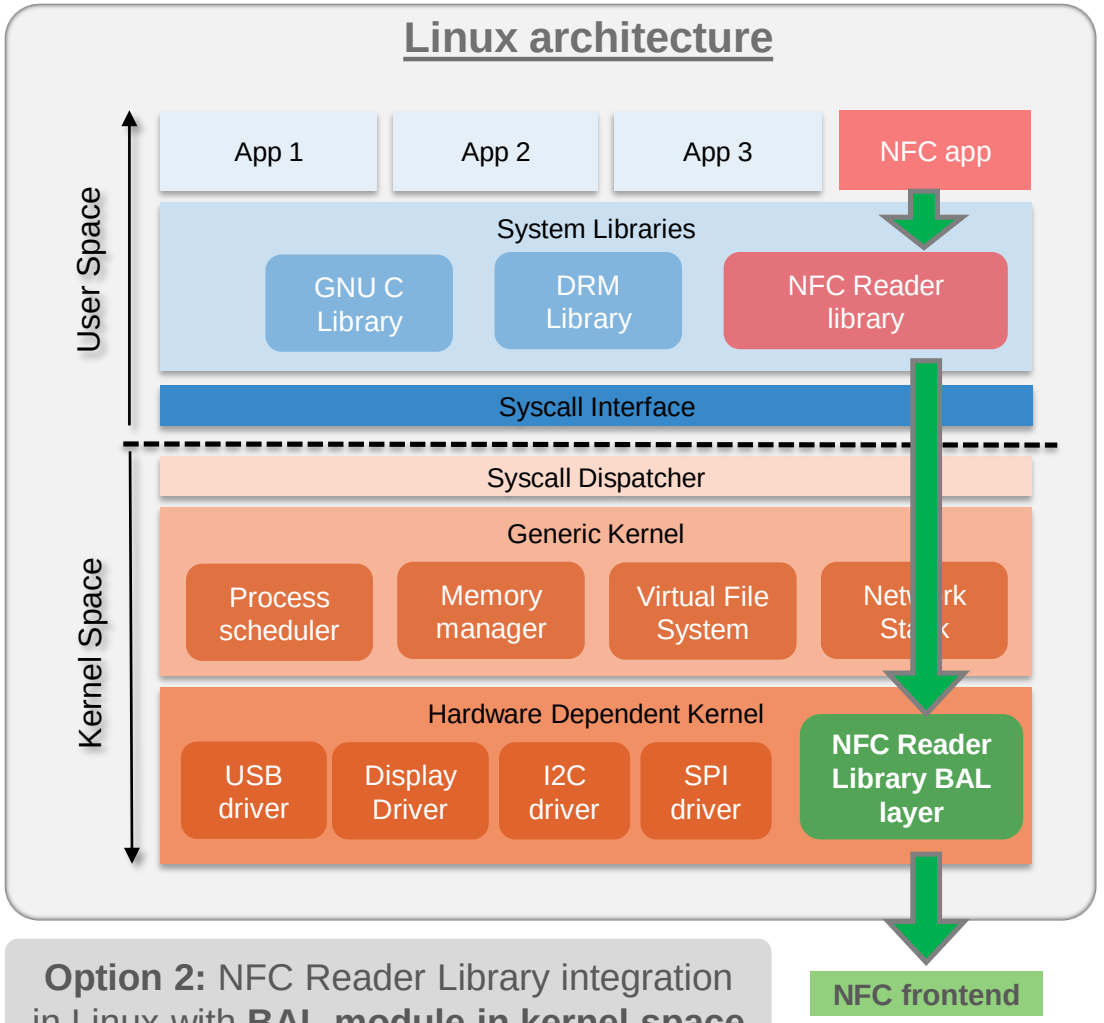
- **Move NFC Reader Library BAL module to Kernel space**

Training Mobile Knowledge

NFC Reader Library support of BAL module in Kernel space



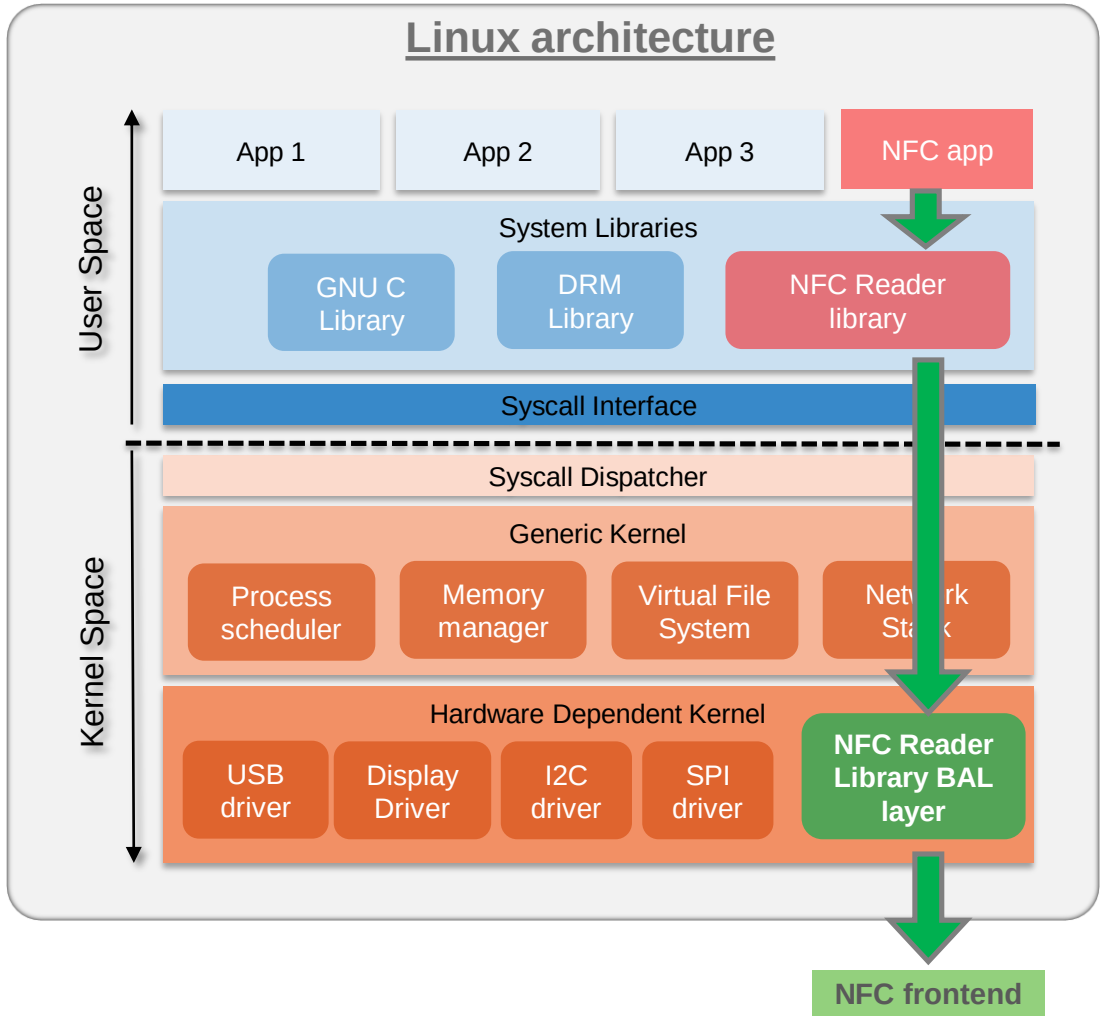
Option 1: Default NFC Reader Library integration in Linux with all lib in **user space**



Option 2: NFC Reader Library integration in Linux with **BAL module in kernel space**

Training Mobile Knowledge

NFC Reader Library BAL module: User Space vs Kernel space



BAL layer in User Space

1. Read GPIO to wait for BUSY line from previous command going low.
2. Setup and start first SPI transfer.
3. Read GPIO to wait for BUSY going low.
4. Setup and start second SPI transfer.
5. ....

Plenty of system calls and context switching operations

BAL layer in Kernel Space

1. System call read() leading to a context switch
2. Access BAL kernel module with direct access to the SPI and GPIO frameworks.

ONLY ONE SYSTEM CALL → Much more efficient, instead of having individual access from user space

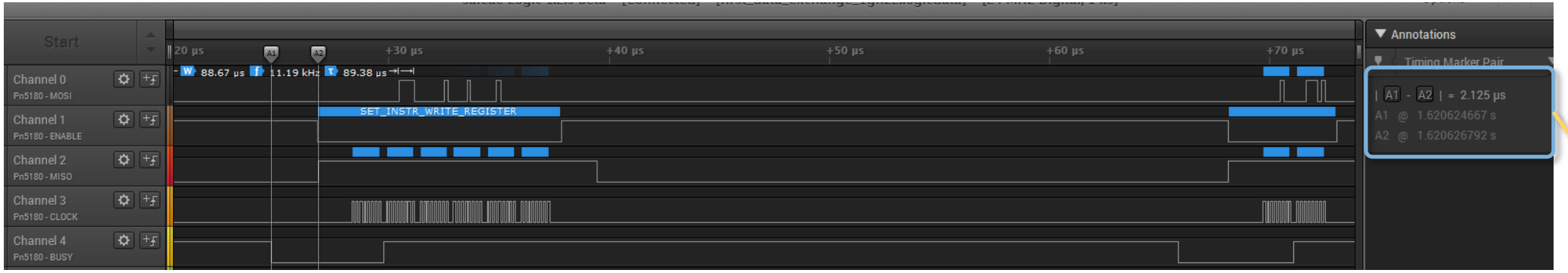
Training Mobile Knowledge

NFC Reader Library BAL module: User Space vs Kernel space

BAL layer in User Space: Measured time between two SPI transfers (Raspberry Pi 2 running Linux OS*)

Until we start writing into the SPI interface, it takes **86us**

BAL layer in Kernel Space: Measured time between two SPI transfers (Raspberry Pi 2 running Linux OS*)



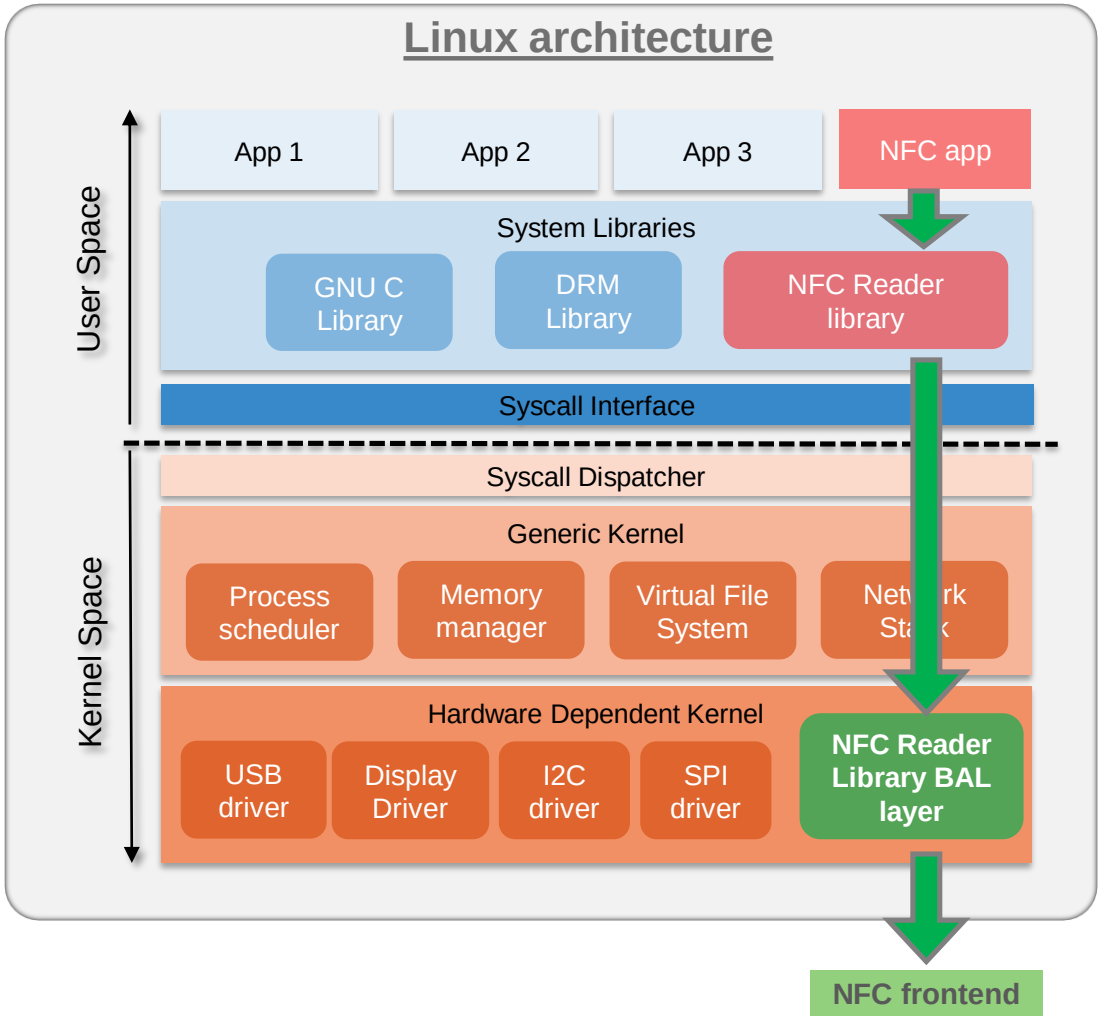
Until we start writing into the SPI interface, it takes **2us**

*This setup was conducted with Raspberry Pi 2, 1GHz Quad core Cortex-A7 at full power. Measurements are not comparable with the above sections.

BAL in Kernel space is ~ 40x times faster

Training Mobile Knowledge

NFC Reader Library BAL module in Kernel space: Resources



[1] GitHub repo with:

- Information about building, configuring and loading the module
- An example of the integration on Raspberry Pi.

[2] App note with:

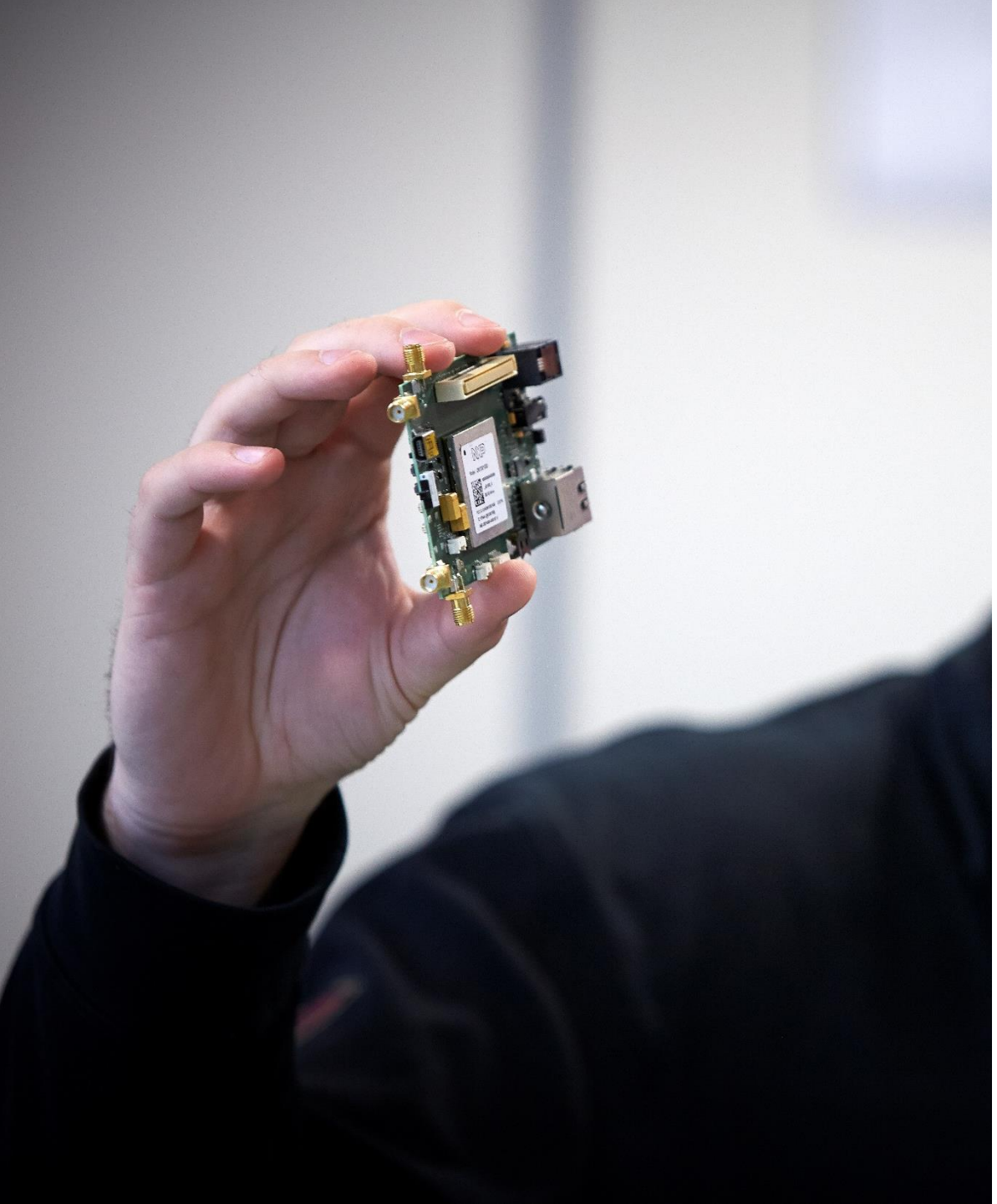
- Explanation of how the NFC Reader Library needs to be changed in order to call the kernel module instead of using the default BAL module running in user-space.

[1] https://github.com/NXPNFCLinux/nxprdlib-kernel-bal

[2] http://www.nxp.com/documents/application_note/AN11802.pdf

Training Mobile Knowledge

Wrap up & Q&A



Reference links & info

- NFC Reader Library
  www.nxp.com/pages/:NFC-READER-LIBRARY
- CLRC663 *plus*
  www.nxp.com/products/:CLRC66303HN
- PN5180
  www.nxp.com/products/:PN5180
- Github Repo:
  https://github.com/NXPNFCLinux/nxprdlib-kernel-bal
- NFC Reader Library for Linux installation guidelines
  http://www.nxp.com/documents/application_note/AN11802.pdf

MobileKnowledge

Software development in Android and iOS
Embedded software for MCUs
JCOP, Java Card operating Systems
Hardware design and development
Digital, analog, sensor acquisition, power management
Wireless communications WiFi, ZigBee, Bluetooth, BLE
Contactless antenna RF design, evaluation and testing

MIFARE® product-based applications
End-to-end systems, readers and card-related designs
EMVco applications
Readers, cards, design for test compliancy (including PCI)
Secure Element management
GlobalPlatform compliant backend solutions
Secure services provisioning OTA, TSM services

We help companies leverage the mobile and **contactless revolution**

MobileKnowledge

Roc Boronat 117, P3M3
08018 Barcelona
(Spain)

Get in touch with us
www.themobileknowledge.com
mk@themobileknowledge.com

How to integrate NFC frontends in Linux

Jordi Jofre (Speaker)
Angela Gemio (Host)

Thank you for your kind attention!

Please remember to fill out our **evaluation survey** (pop-up)

Check your email for **material download** and on-demand **video** addresses

Please check NXP and MobileKnowledge websites for **upcoming webinars** and **training sessions**

http://www.nxp.com/support/classroom-training-events:CLASSROOM-TRAINING-EVENTS

www.themobileknowledge.com/content/knowledge-catalog-0