

# NFC Integration in Real-Time and Non-Real-Time Operating Systems

Todd Nuzum

FAE

---

April 2019 | AMF-SOL-T3522



SECURE CONNECTIONS  
FOR A SMARTER WORLD

# Agenda

---

- NFC Readers Software Development Design-in Support
- NFC Frontend Integration in Linux
- Integrating the NFC Reader Library in Linux
- Host Interface Access on Linux Systems
- Latency Analysis: Linux vs Bare Metal
- What is Real-time
- Overcoming Linux Higher Latency for Time-sensitive Applications
- Selecting the Right Product

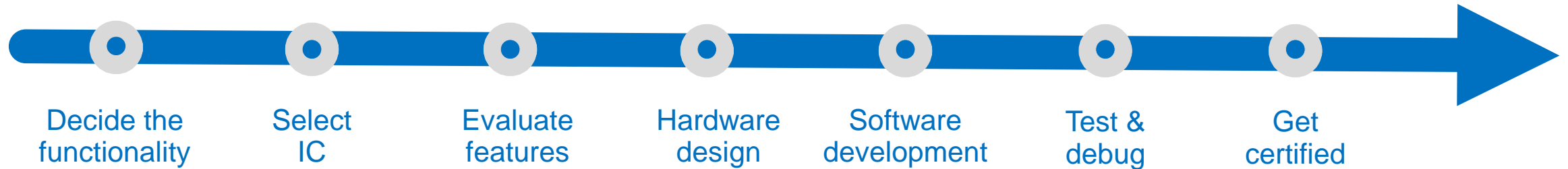
# NFC Readers Software Development Design-in Support



# We Make NFC Easy

## NFC implementation process

We reduce complexity, streamline tasks, and add flexibility at every point in development, so you can deliver a competitive advantage in record time.



**We know each step in the NFC implementation process**

Our support package simplifies the process and reduces time to market



**We have the right material for each design step**

Full range of development kits, design files, sample code, app notes, online training, tutorials

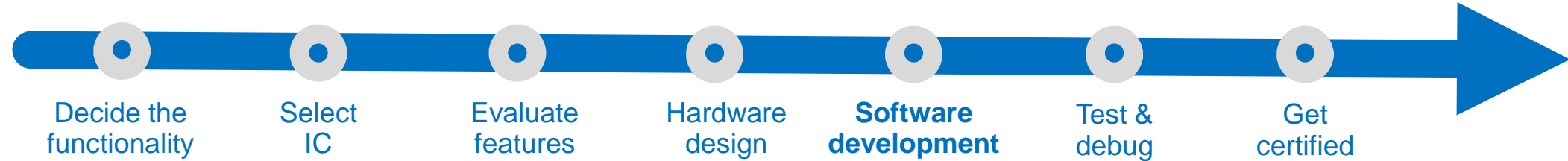


**Directly find answers to your questions**

Through our technical NFC community and NXP certified Independent Design Houses (IDHs)

# NXP's Software Development Support

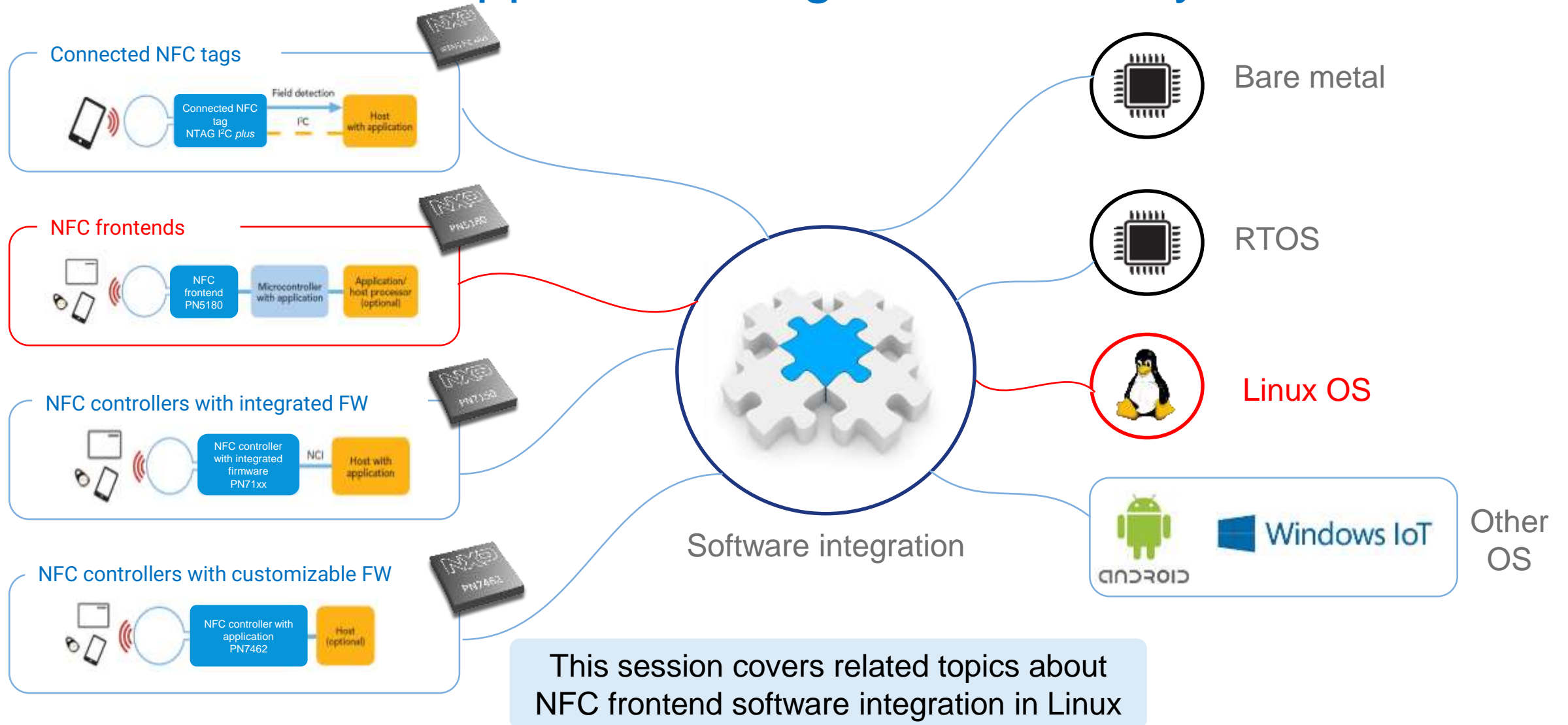
## NFC implementation process



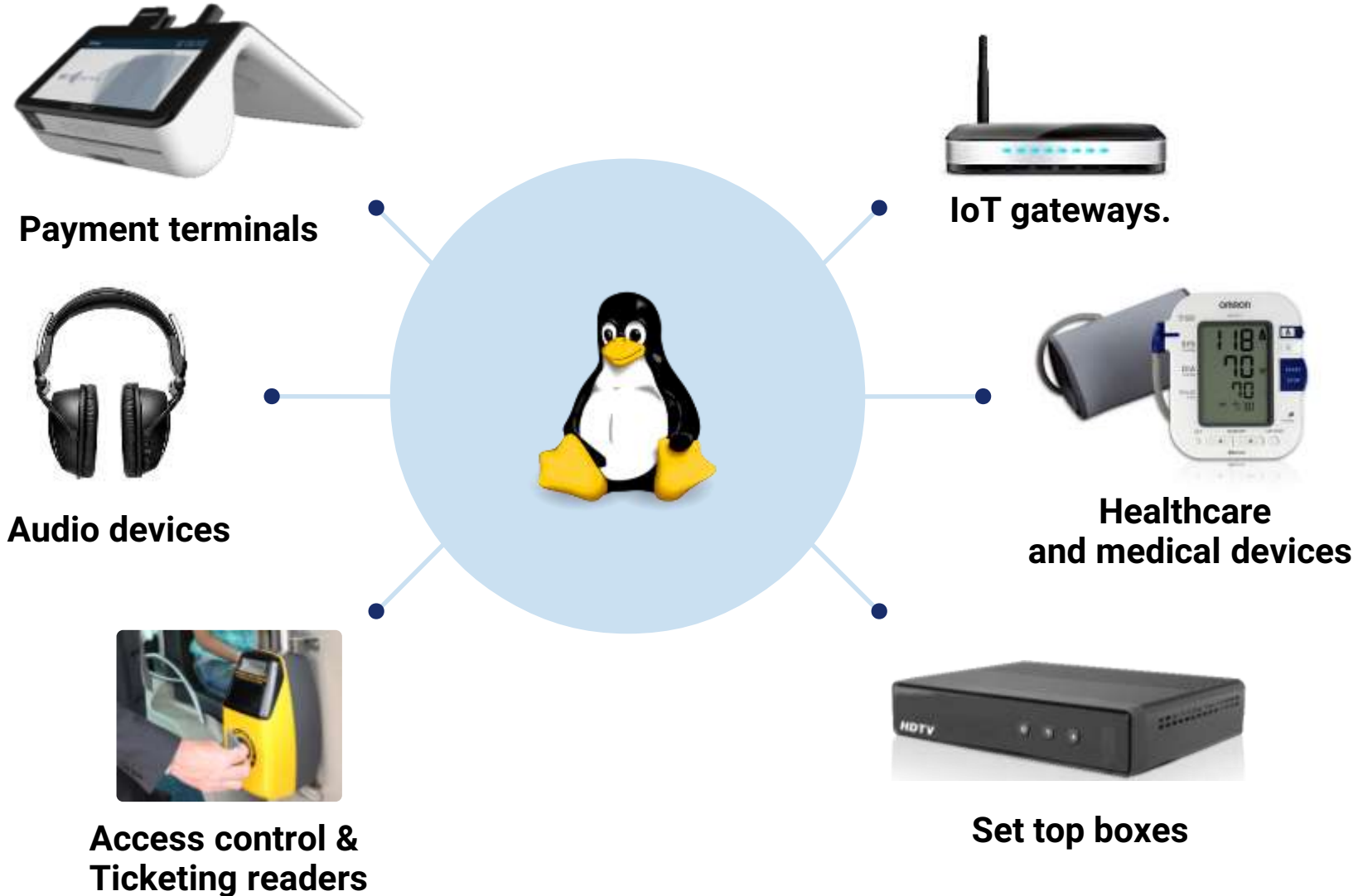
- NFC Reader Library
- Sample code
- Design files for development kits
- App notes
- Online training on SW integration & tutorials
- NFC Cockpit

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

# NXP Software Support for Integration Into Any Platform



# An Increasing Number of Devices Running Linux



# NFC Frontend Integration in Linux

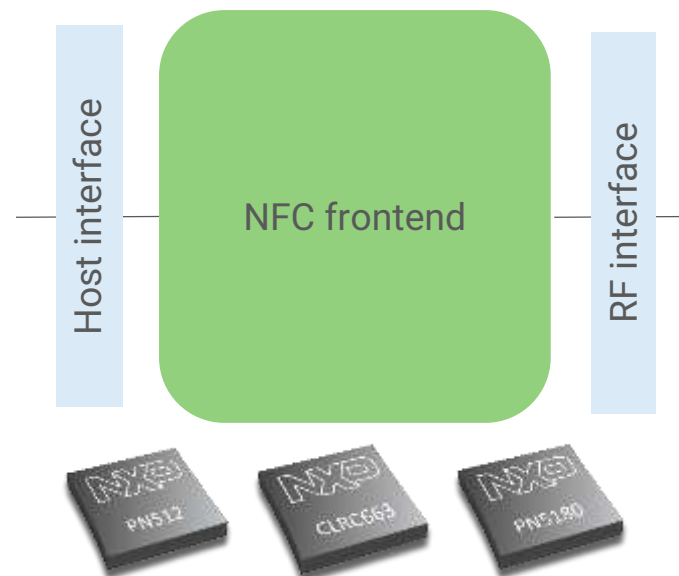




# NFC Frontend Expose a Host Interface and a Contactless Interface

## 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 for:
  - to configure RF framing and signaling .
  - to finally transfer the RF digital protocol based blocks to/from a counterpart.



## RF interface

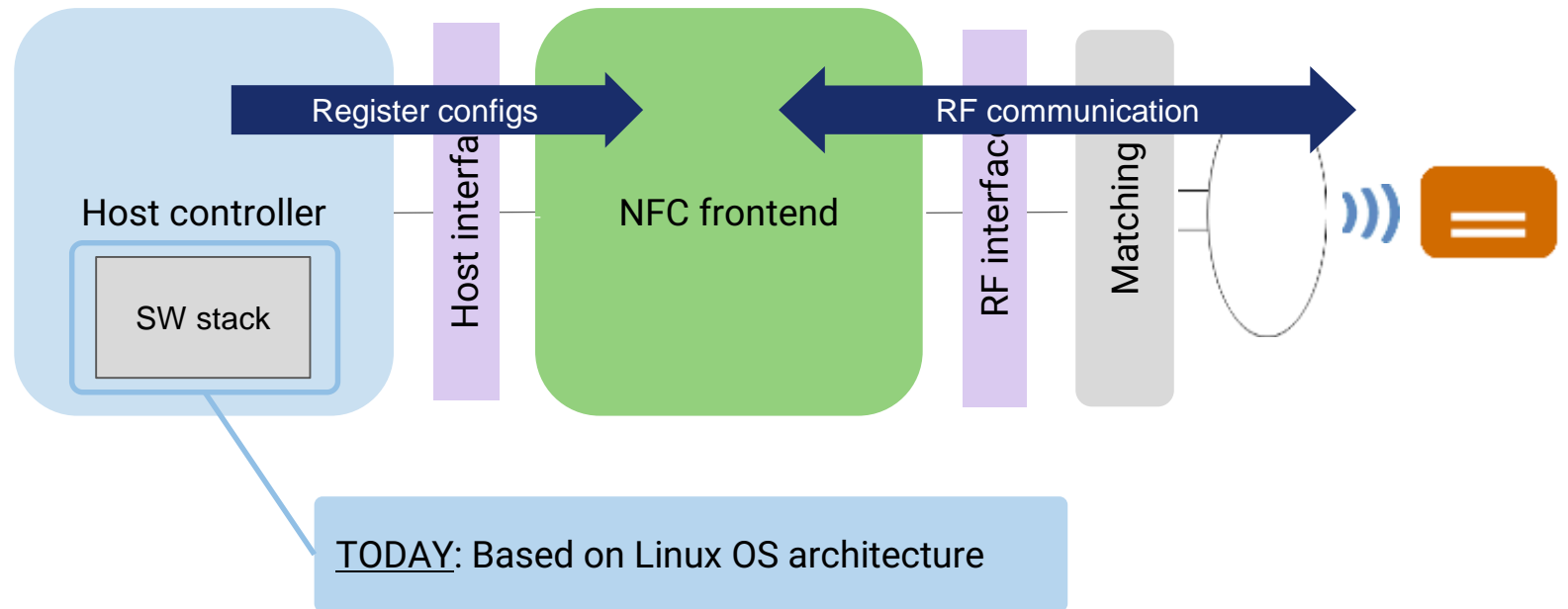
- An NFC frontend is an RF transceiver enabling the contactless communication.
- It deals with the signal modulation and handles the data transmission through the RF interface.
- The NFC frontend needs to be selected according to application requirements:
  - RF performance
  - RF protocols
  - NFC modes 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

# NFC Frontend is Controlled by the External Host Controller SW

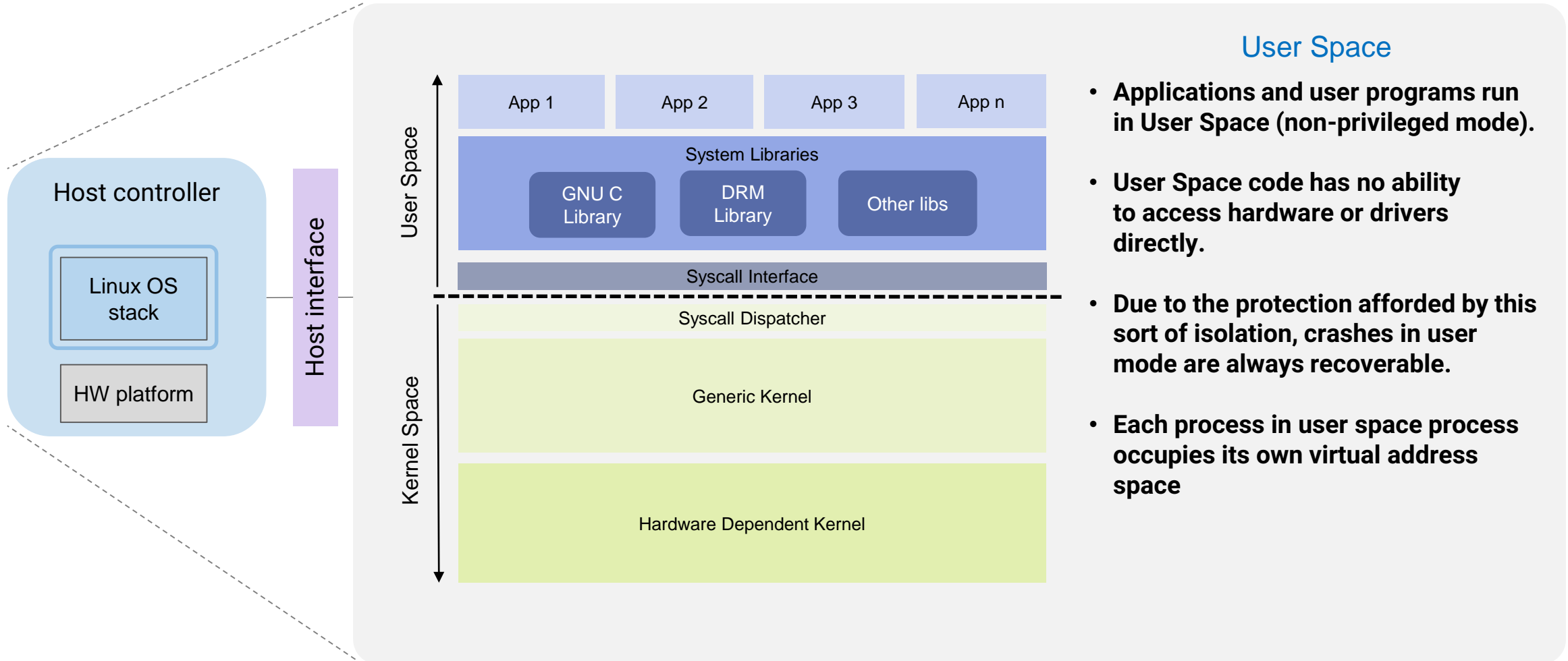
## Host controller

- 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



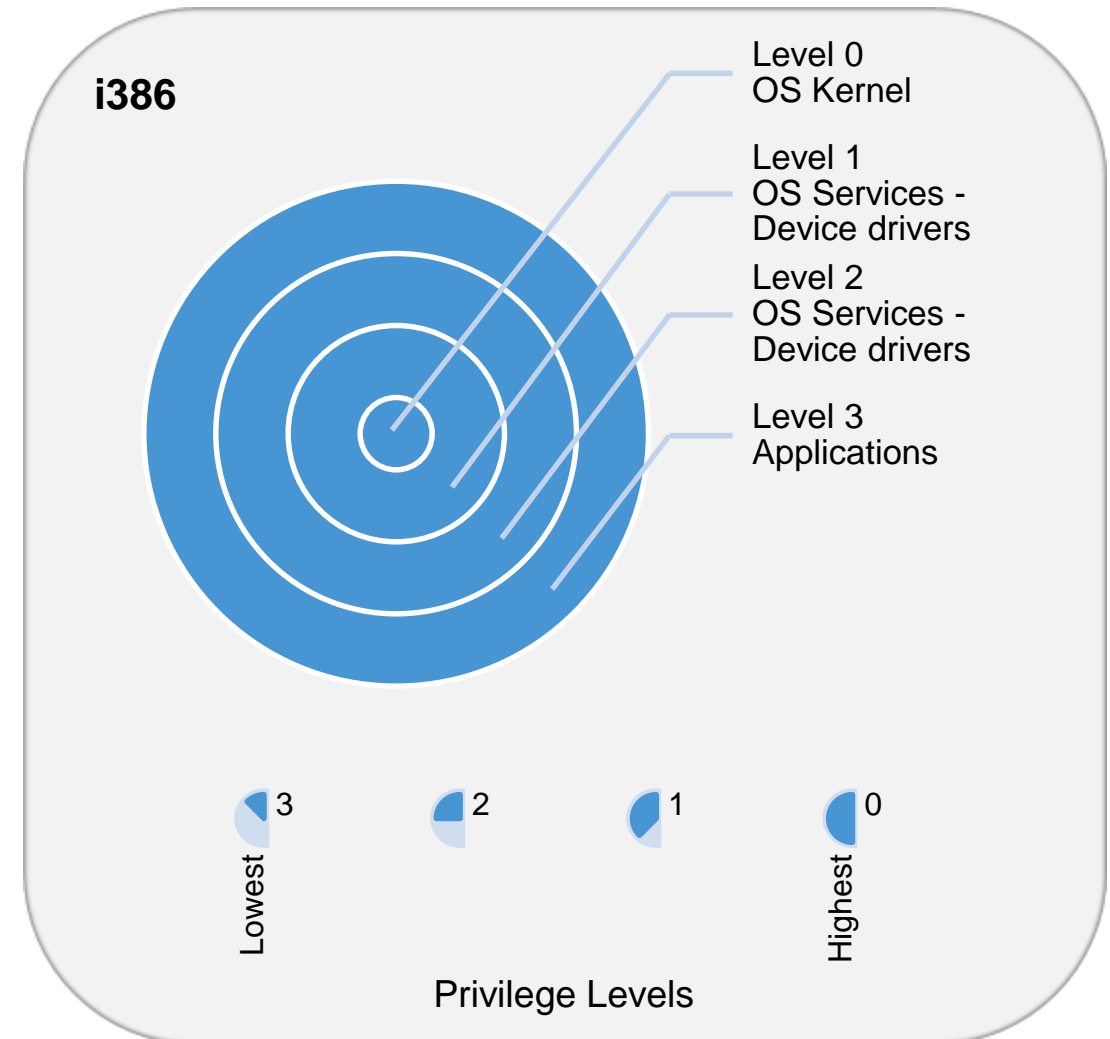
The host controller drives and controls the NFC frontend according to register settings configuration

# Host Controller SW: Linux OS Architecture – User Space



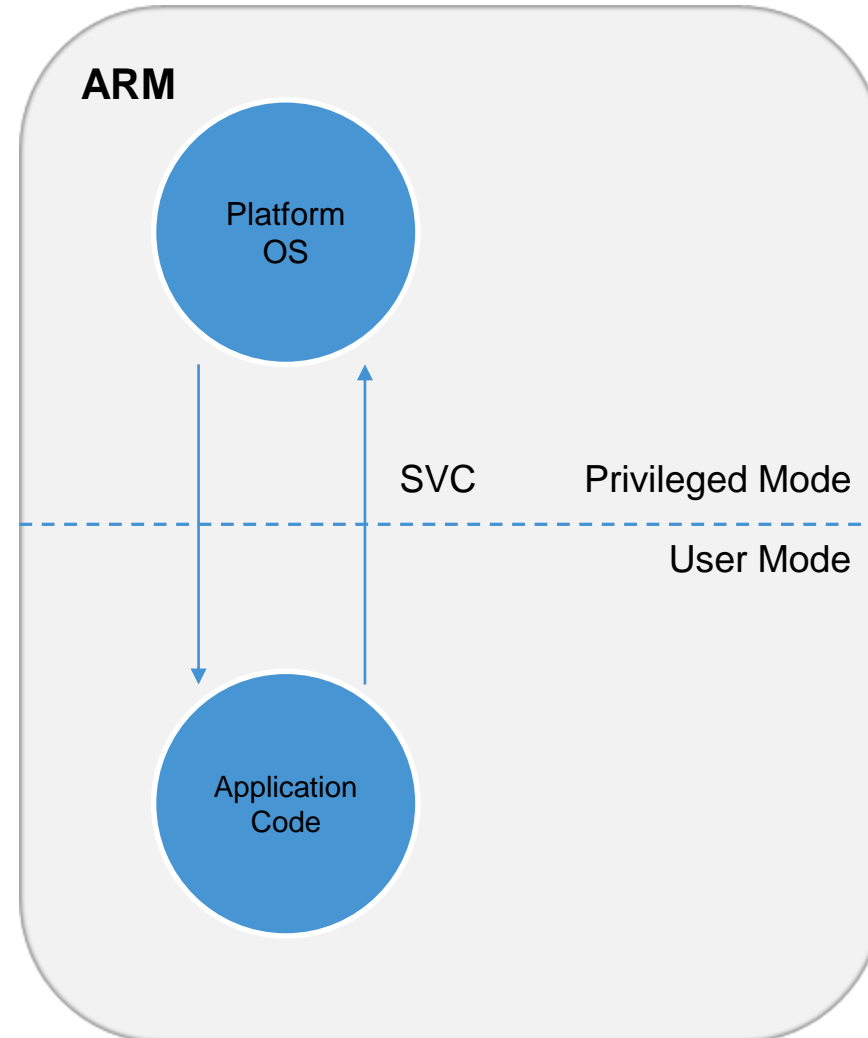
# Privilege Separation: i386 and ARM Architecture

- Most processors define so called privilege levels.
- i386 knows 4.
- ARM v7 knows 3.
  
- ARM v7
  - PL0 – Unprivileged level for user applications. User mode
  - PL1 – Privileged level for operating system.
  - PL2 – Hypervisor mode. Can switch between guest OS that execute in PL0.



# Privilege Separation: Switching the Level

- Switching the privilege level must be controlled
- On ARM a supervisor call (SVC) is used to enable user mode code to access OS functions
- SVC provides a well defined handler to switch the processor mode
- The SVC triggers a processor exception

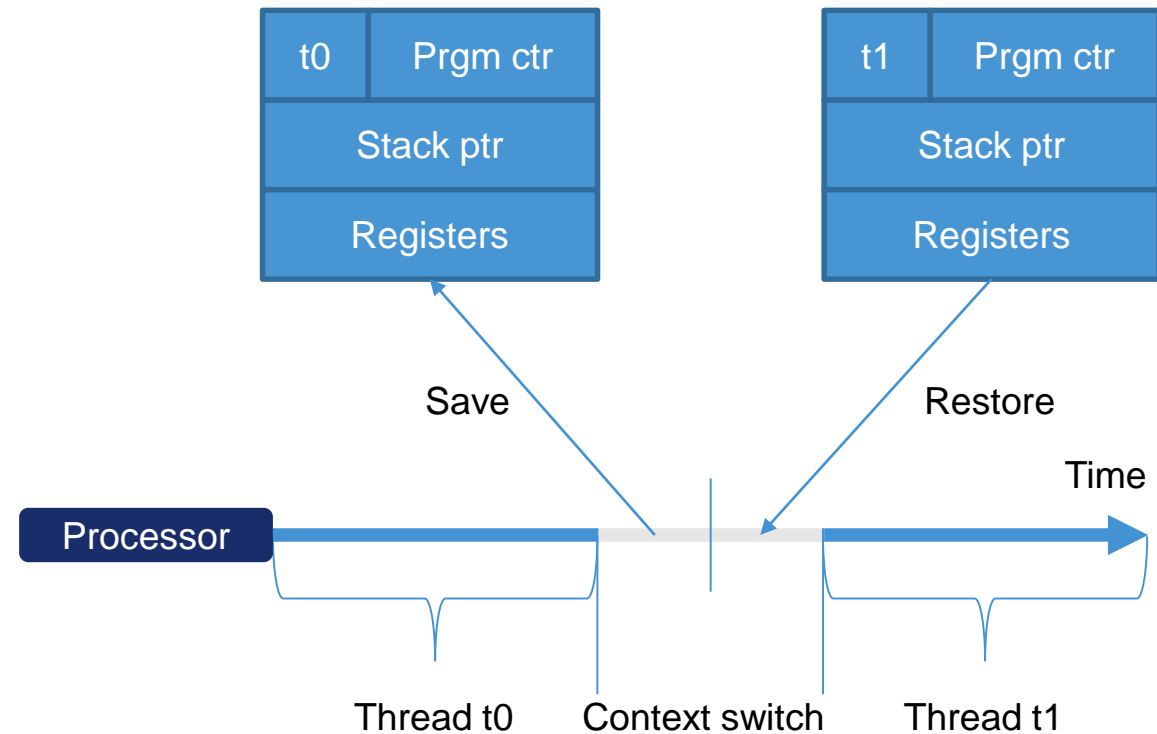


# Privilege Separation: Context Switch

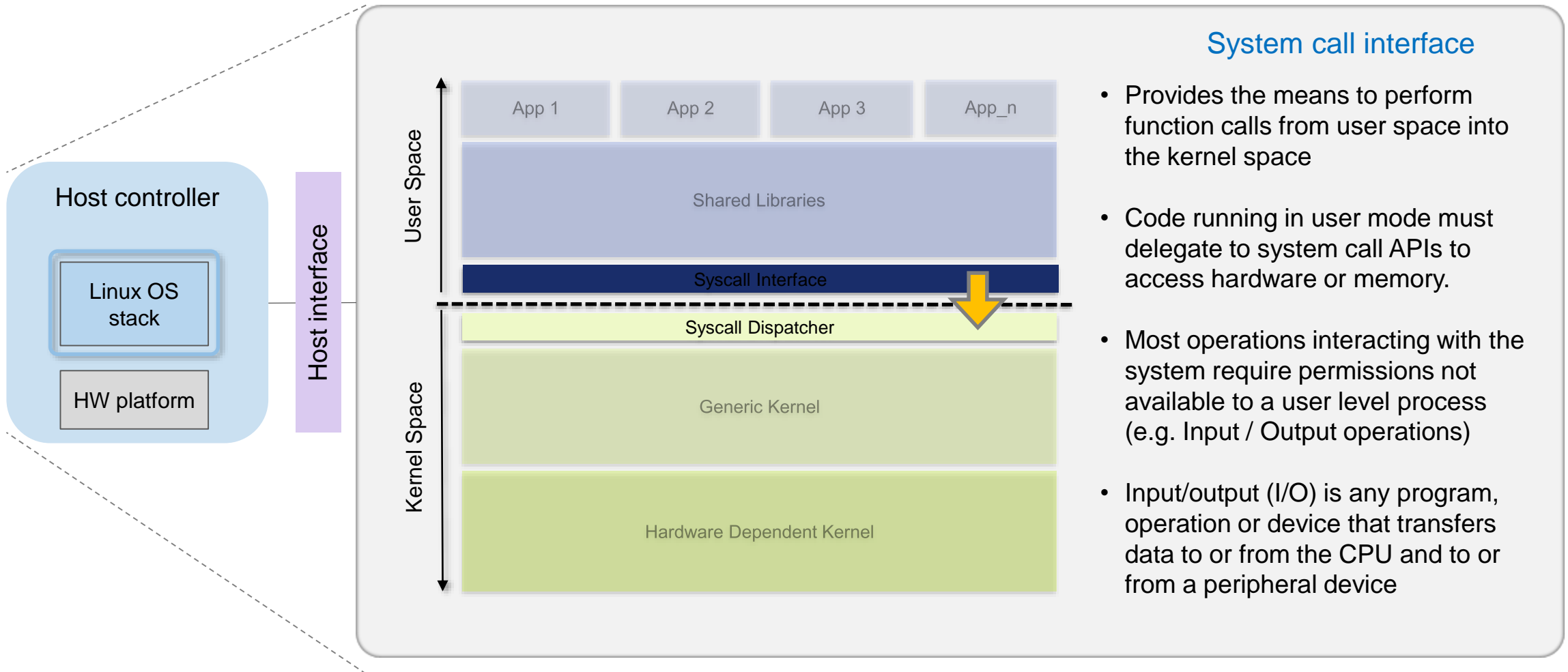
- Changing privilege level on an OS always comes with a context switch.

What is a context switch?

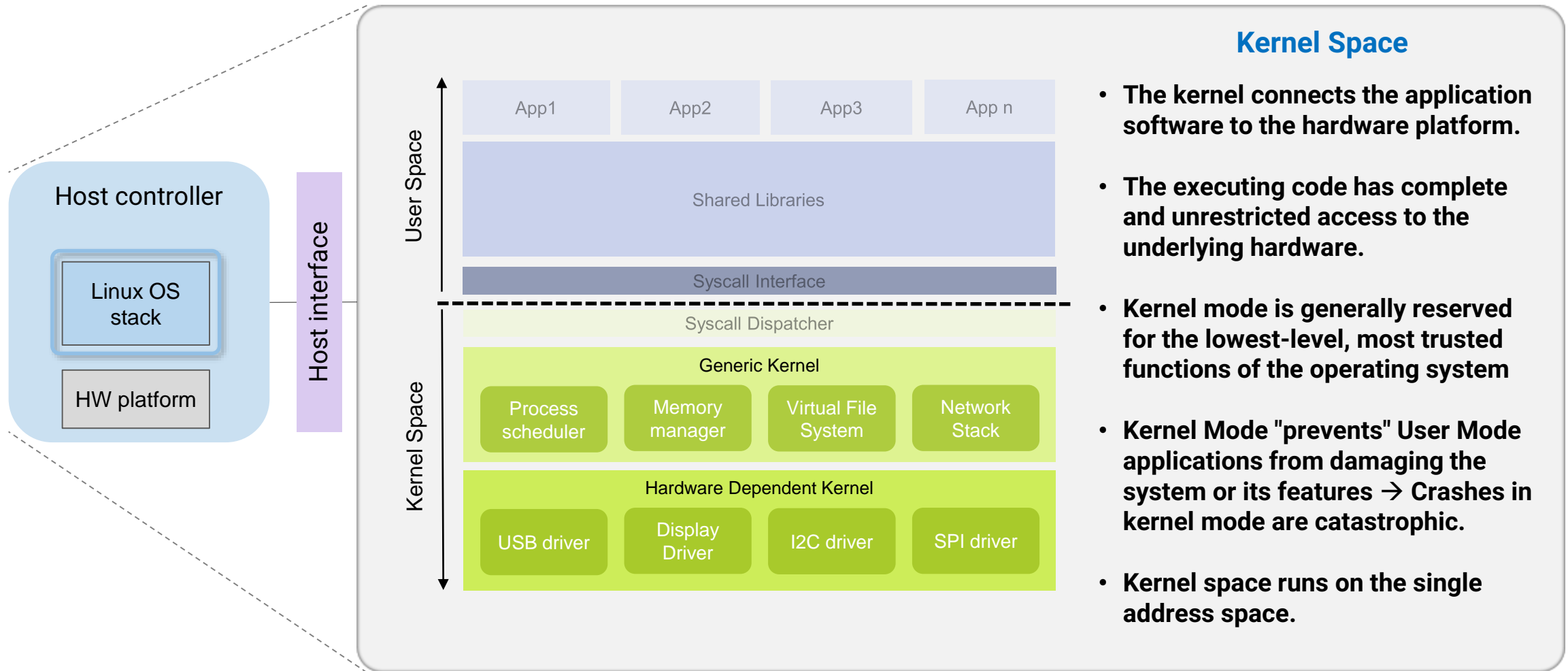
- Storing current processor state and restoring another.
- The interrupt handler manages the context switch.
- The interrupt handler has to:
  - Switch to privileged mode
  - Save defined registers to the process stack.
  - Save current task's Process Stack Pointer (PSP) to memory.
  - Load next tasks stack pointer and assign to PSP.
  - Load registers from process stack.
  - Switch back to unprivileged mode.



# Host Controller SW: Linux OS Architecture – System Call Interface



# Host Controller SW: Linux OS Architecture – Kernel Space

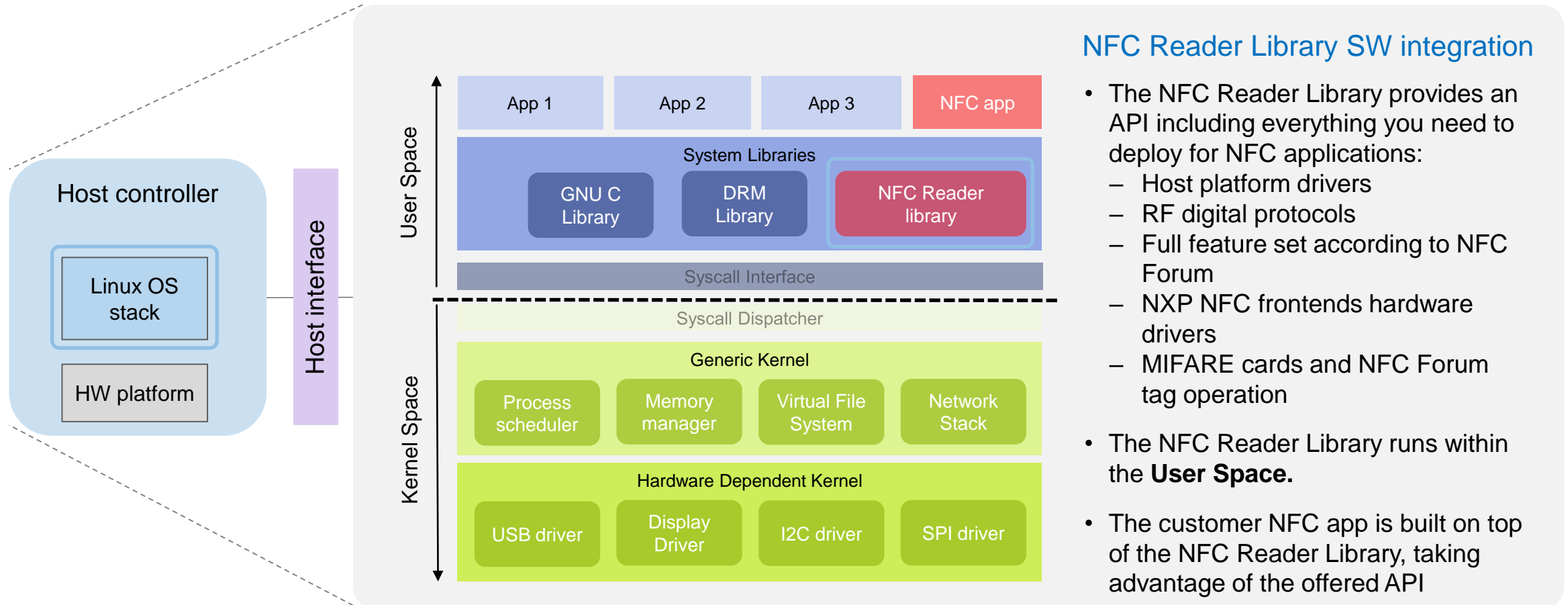




# Integrating the NFC Reader Library in Linux



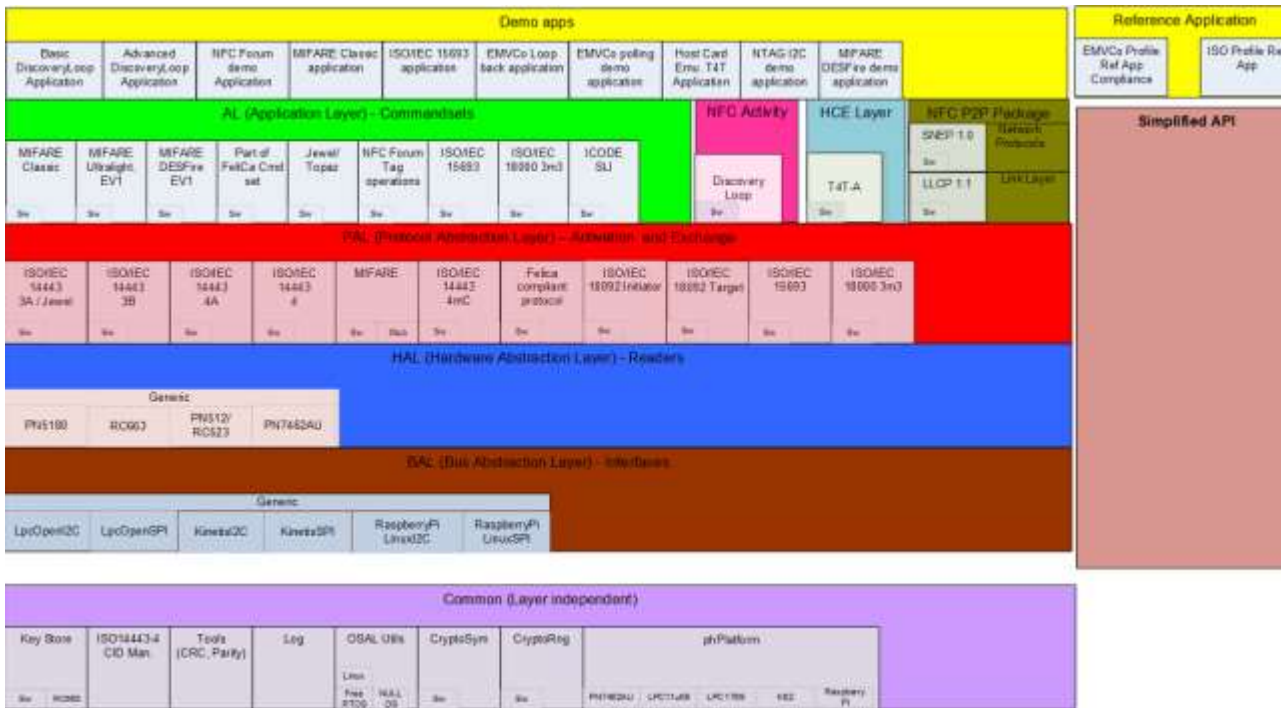
# NFC Reader Library: The SW Stack to Develop NFC Apps



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

# NFC Reader Library & Available Resources

## NFC Reader Library



**Features:** Modular, multi-layered, ANSI-C language, portable to multiple platforms and free download

Don't start from scratch, available software examples to test and re-use

## Software examples

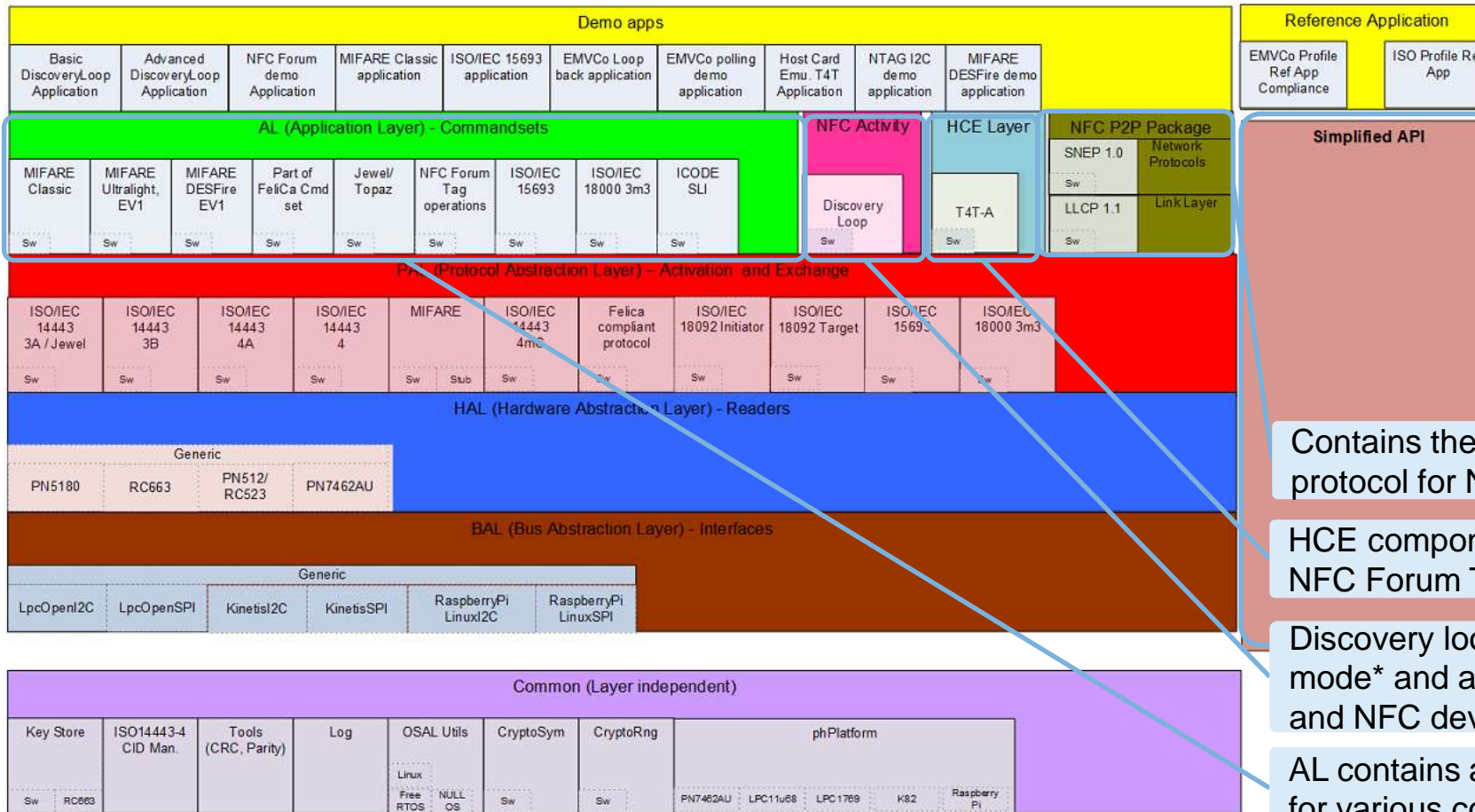


- Example 1: BasicDiscoveryLoop
- Example 2: AdvancedDiscoveryLoop
- Example 3: NFCForum
- Example 4: MIFARE Classic
- Example 5: ISO15693
- Example 6: EMVCo Loopback
- Example 7: EMVCo Polling
- Example 8: HCE T4T
- Example 9: NTAG I2C
- Example 10: SimplifiedAPI\_EMVCo
- Example 11: SimplifiedAPI\_ISO

For additional information and source code, please visit: [www.nxp.com/pages/:NFC-READER-LIBRARY](http://www.nxp.com/pages/:NFC-READER-LIBRARY)

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

# NFC Reader Library Architecture



High level abstraction of the NFC Reader Library. Two profiles for: EMVCo and ISO

Contains the implementation of LLPC and SNEP protocol for NFC P2P mode

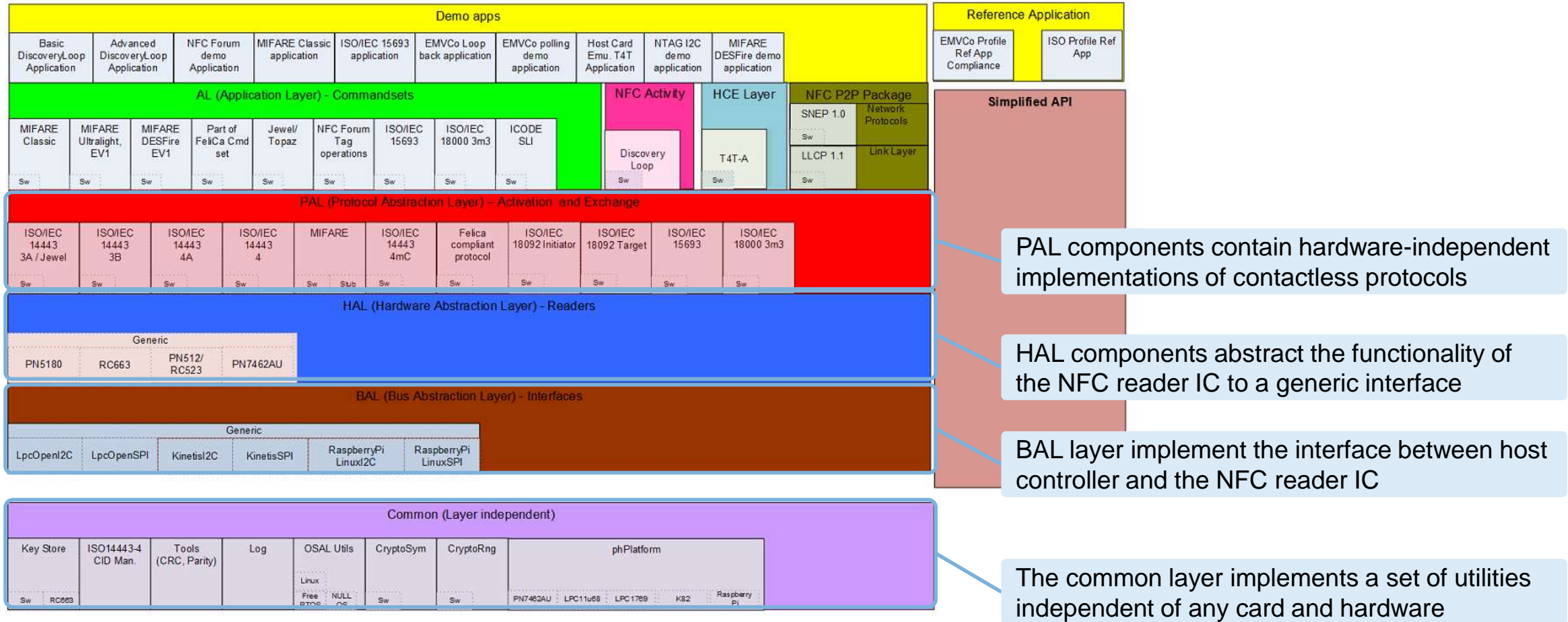
HCE component implements the card emulation of NFC Forum Type 4A tag

Discovery loop component implements a poll mode\* and a listen mode\*\* for contactless card and NFC device detection

AL contains application-specific implementations for various contactless cards (card command sets)

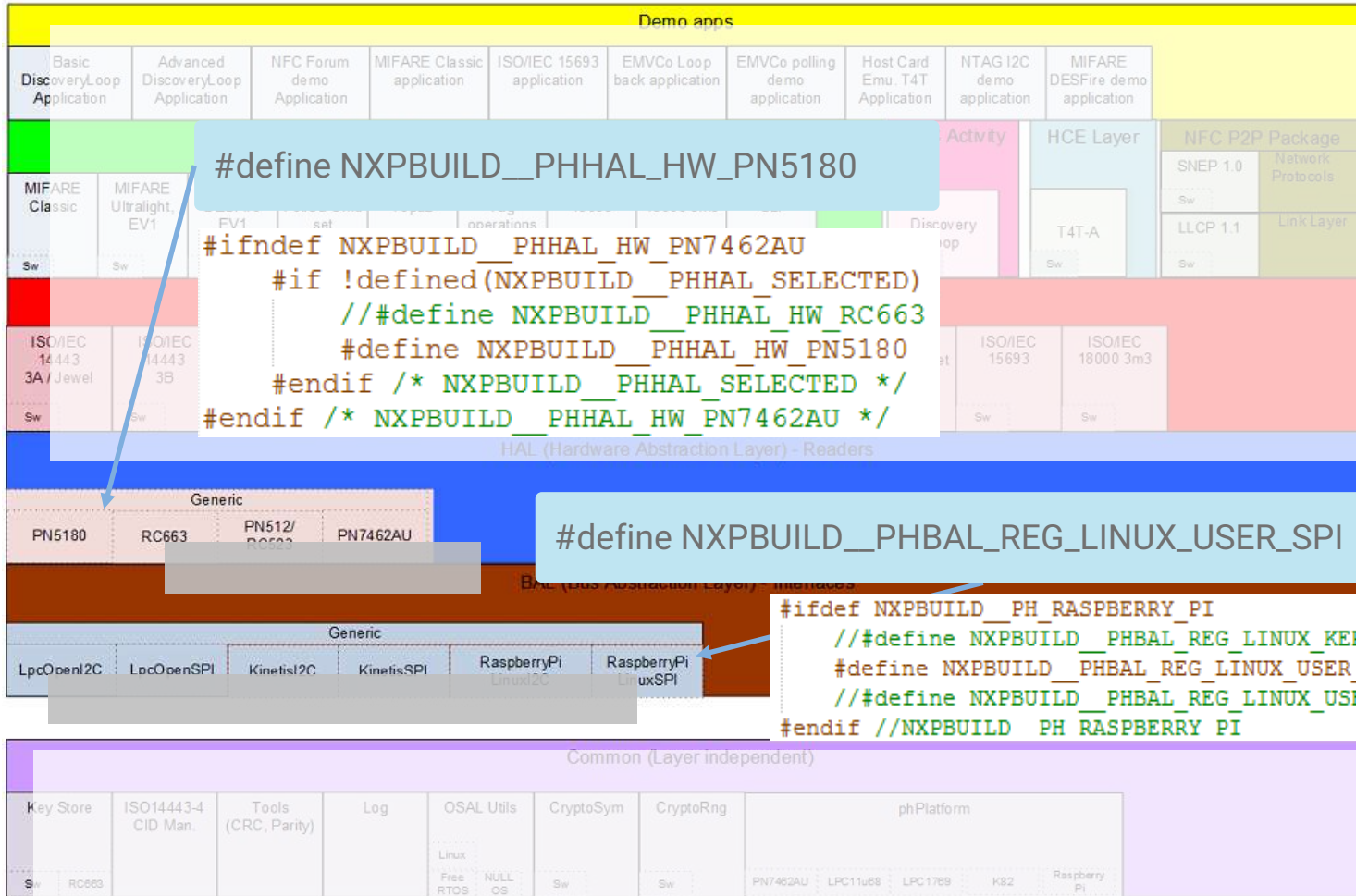


# NFC Reader Library Architecture (II)



Raspberry Pi is used as reference platform for Linux version of the NFC Reader Library

# NFC Reader Library – Building the SW Stack



```
#define NXPBUILD__PHHAL_HW_PN5180
#ifndef NXPBUILD__PHHAL_HW_PN7462AU
  #if !defined(NXPBUILD__PHHAL_SELECTED)
    // #define NXPBUILD__PHHAL_HW_RC663
    #define NXPBUILD__PHHAL_HW_PN5180
  #endif /* NXPBUILD__PHHAL_SELECTED */
#endif /* NXPBUILD__PHHAL_HW_PN7462AU */
```

```
#define NXPBUILD__PHBAL_REG_LINUX_USER_SPI
```

```
#ifndef NXPBUILD__PH_RASPBERRY_PI
  // #define NXPBUILD__PHBAL_REG_LINUX_KERNEL_SPI
  #define NXPBUILD__PHBAL_REG_LINUX_USER_SPI
  // #define NXPBUILD__PHBAL_REG_LINUX_USER_I2C
#endif //NXPBUILD__PH_RASPBERRY_PI
```

- The build setup and functionality is set in the file: `../intfs/ph_NxpBuild.h`.
- This file defines the modules to be **included** into the build setup or to be **excluded** from the build setup .
- There is a specific **macro** defined for including / excluding each SW component
- Components can be included / excluded depending on the application requirements or to optimize memory footprint.

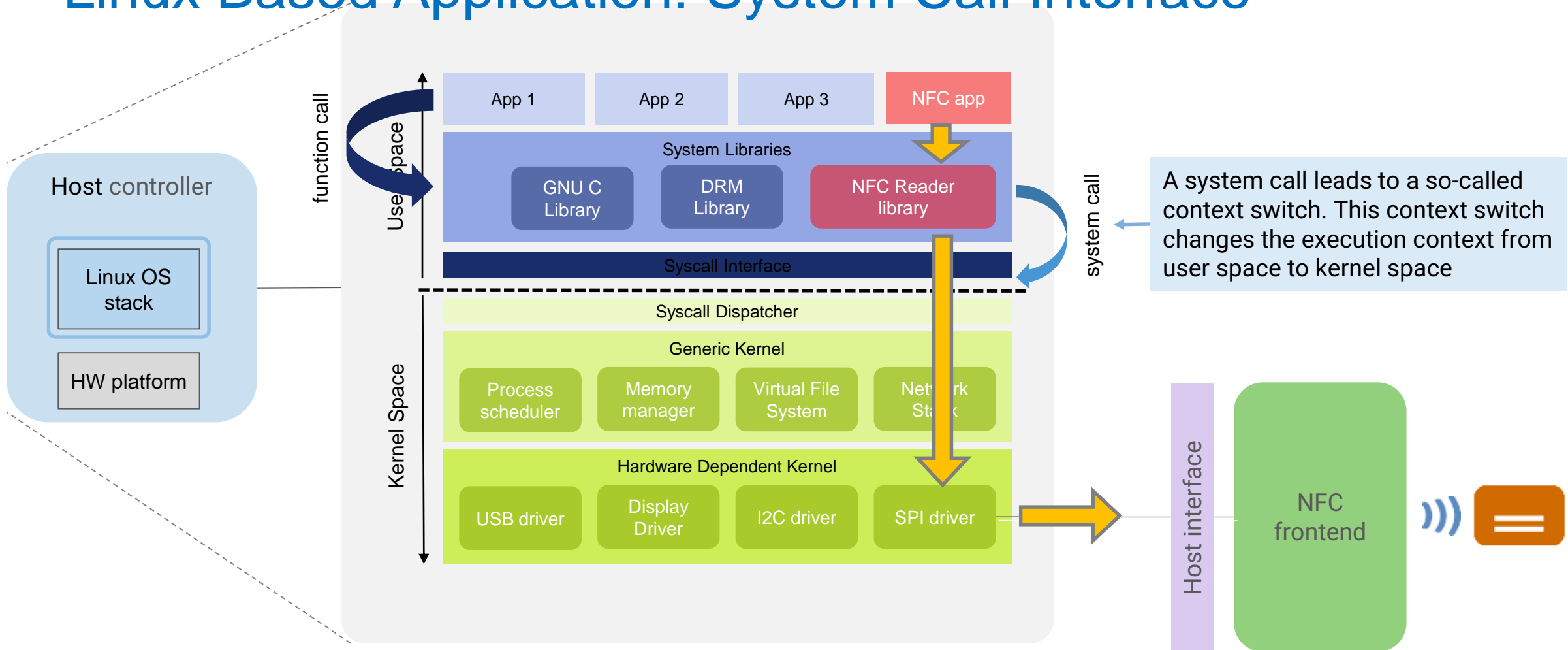
Components not included in the project build

Modules can be enabled / disabled to optimize code size and memory footprint

# Host Interface Access on Linux Systems



# Linux Based Application: System Call Interface

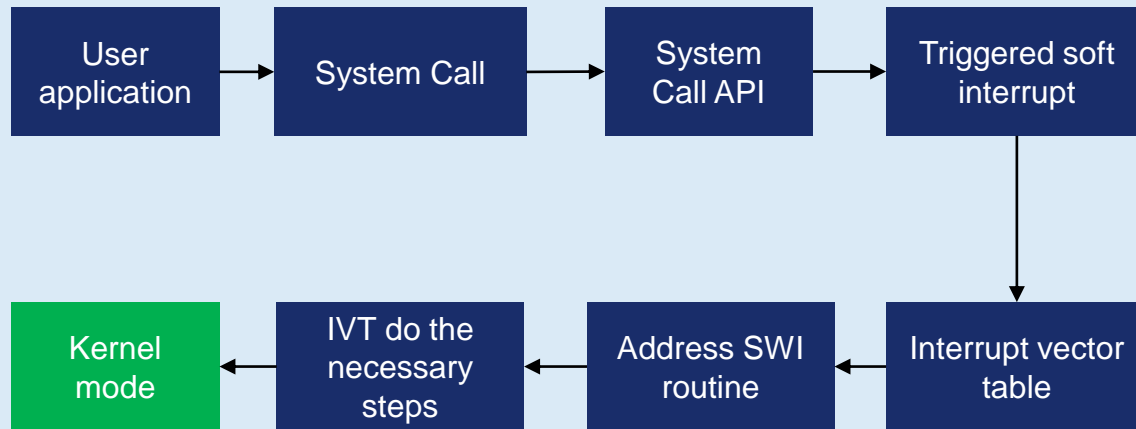


The NFC application needs to switch from User Space to Kernel Space for every SPI interface access



# Transition Between User Mode and Kernel Mode

## Switching from User mode to Kernel mode



- User application initiate switching to kernel mode making a system call (e.g. open, read, write, etc)
- A software interrupt (SWI) is triggered
- The interrupt vector table launch 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 that a crashing application crashes the whole system and protects system resources.

## Disadvantage:

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



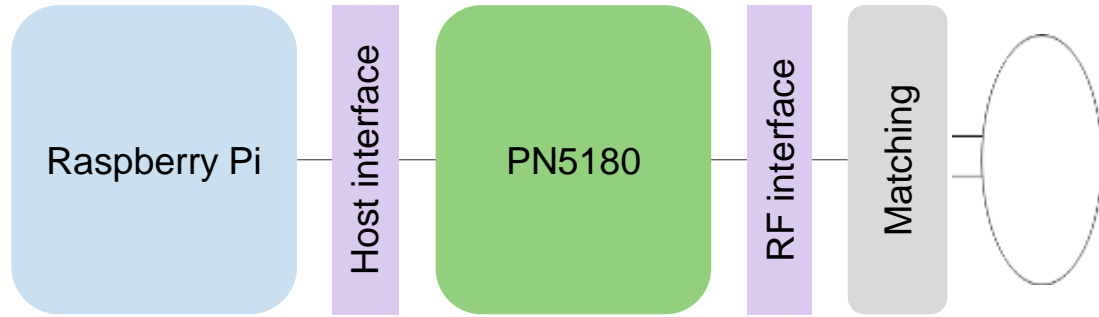
Could challenge the design of time-critical NFC applications

# Latency Analysis: Linux vs Bare Metal

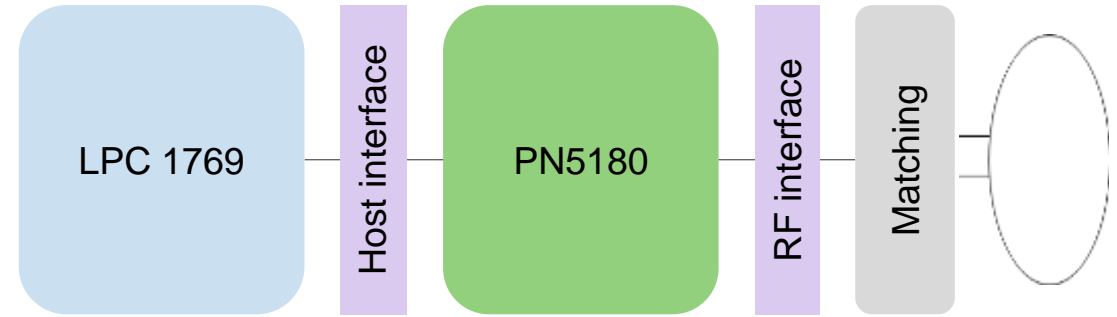


# Hardware Setup

Linux setup



Bare metal 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

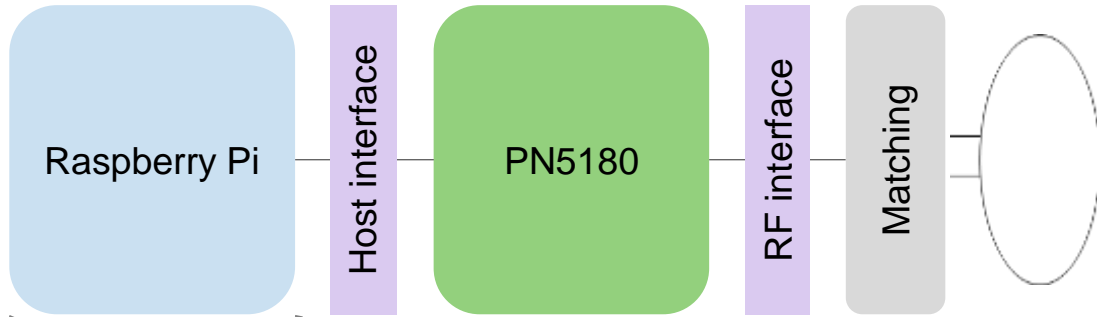
## Bare metal hardware platform

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

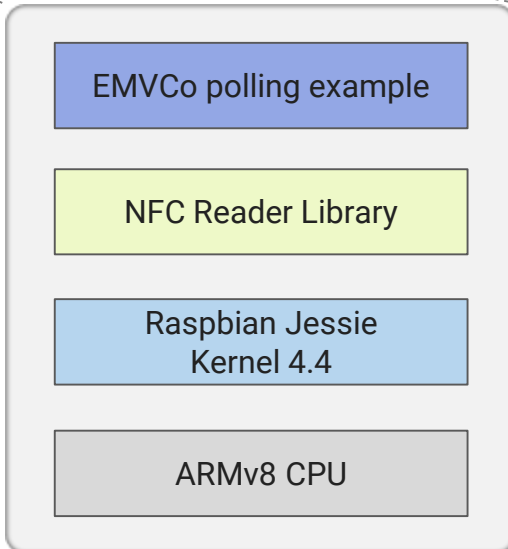
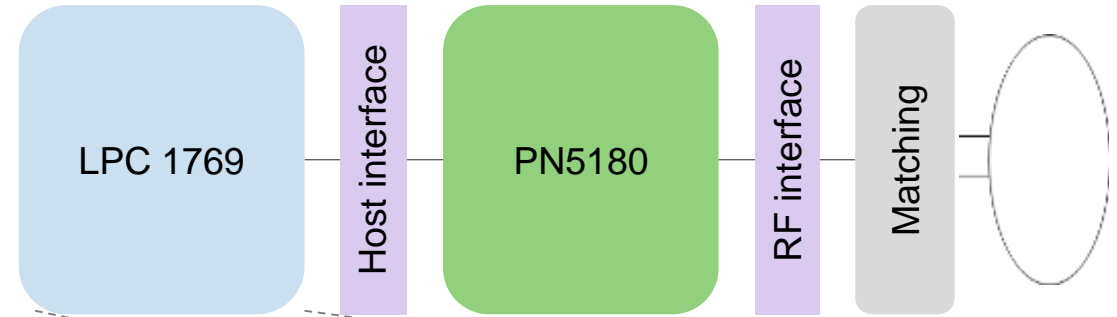
We limited Raspberry Pi clock and MCU cores to achieve a comparable setup with LPC1769

# Software Setup

Linux setup

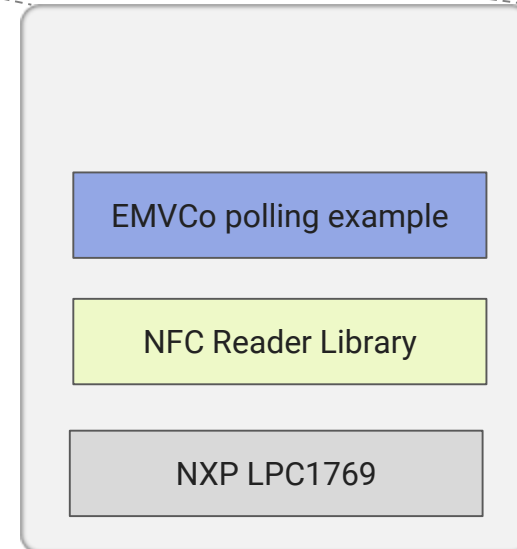


Bare metal setup

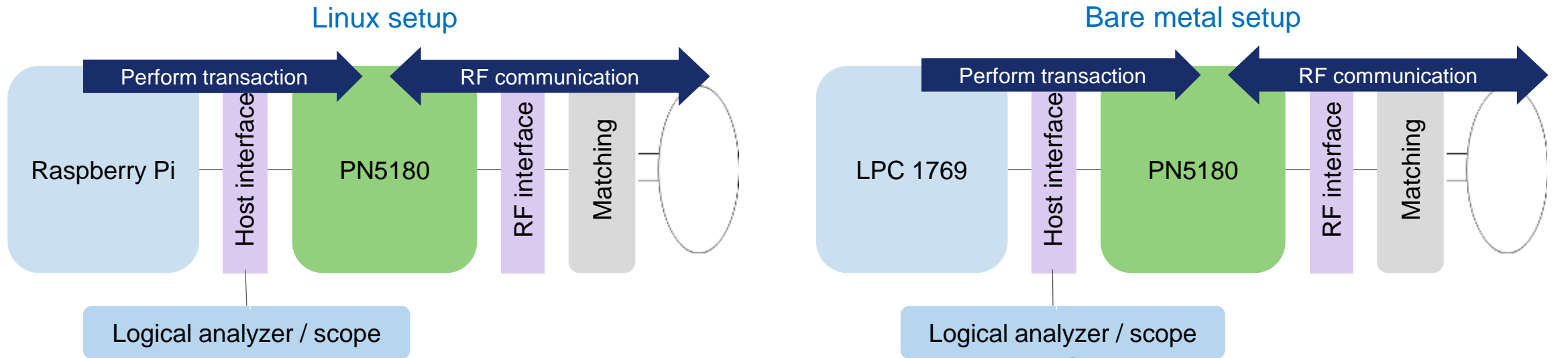


We execute the NFC Reader Library and the same SW example in both platforms

**EMVCo polling example:** Discovery loop for EMVCo card detection and APDU command exchange



# Measurement Setup



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

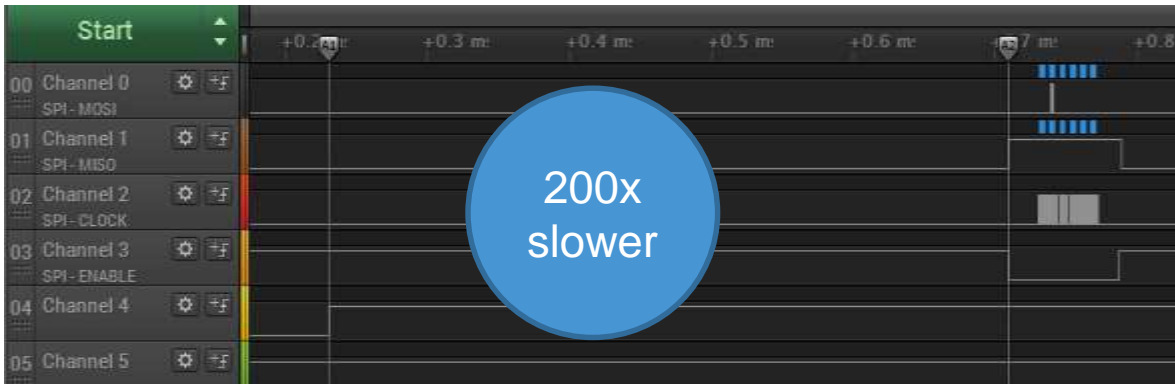
- Measurements conducted**
- Time to set up SPI transfer
  - Time between two SPI transfers
  - Time for EMVCo polling initialization
  - Time for EMVCo loopback transaction

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

We compared the results in the next slides

# Measured Time Setting Up SPI Transfer

Linux setup



```
| A1 - A2 | = 0.47875 ms  
A1 @ 1.48323075 s  
A2 @ 1.4837095 s
```

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



```
| A1 - A2 | = 2.5 us  
A1 @ 1.41213525 s  
A2 @ 1.41213775 s
```

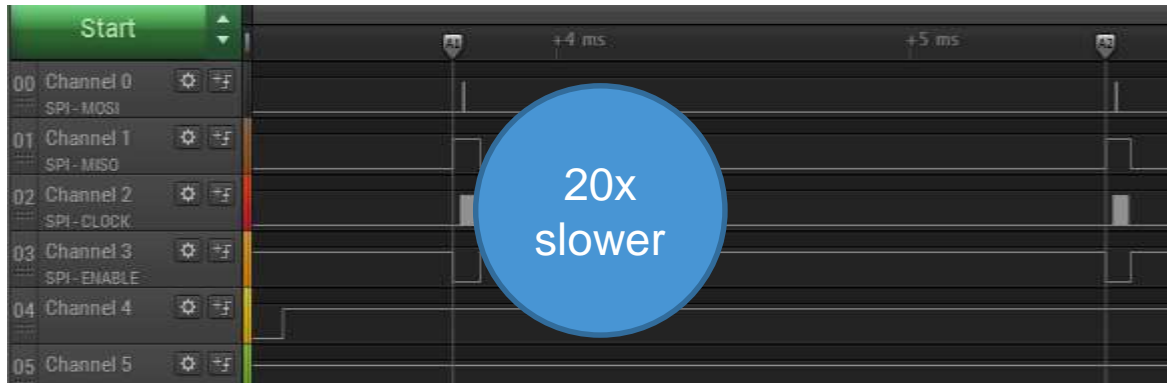
Until we start writing into the SPI interface, it takes **2.5 us**

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

\* GPIO toggling execution takes less than 3us

# Measured Time Between Two SPI Transfers

Linux setup

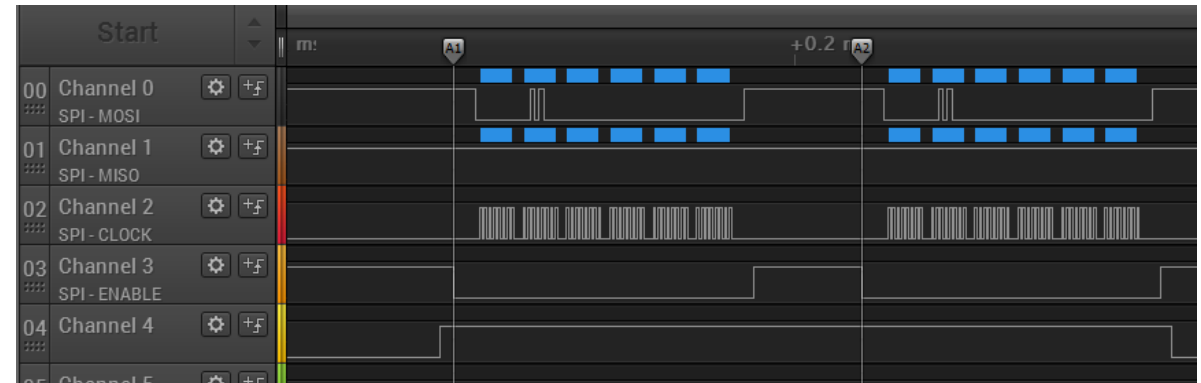


```
| A1 - A2 | = 1.841375 ms  
A1 @ 1.4837095 s  
A2 @ 1.485550875 s
```

Until we start writing the second SPI transfer, it takes 1.8 ms

1. Set\_GPIO(High)\*;
2. phhalHw\_PN5180\_WriteRegister(...);
3. phhalHw\_PN5180\_WriteRegister(...);
4. Set\_GPIO(Low);

Bare metal setup



```
| A1 - A2 | = 74.5 us  
A1 @ 1.41213775 s  
A2 @ 1.41221225 s
```

Until we start writing the second SPI transfer, it takes 74.5us

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

# Measured Time for EMVCo Polling Initialization (Multiple Transfers)

Linux setup



+Width(1):  
301.13ms

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

Bare metal setup



+Width(1):  
80.76ms

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

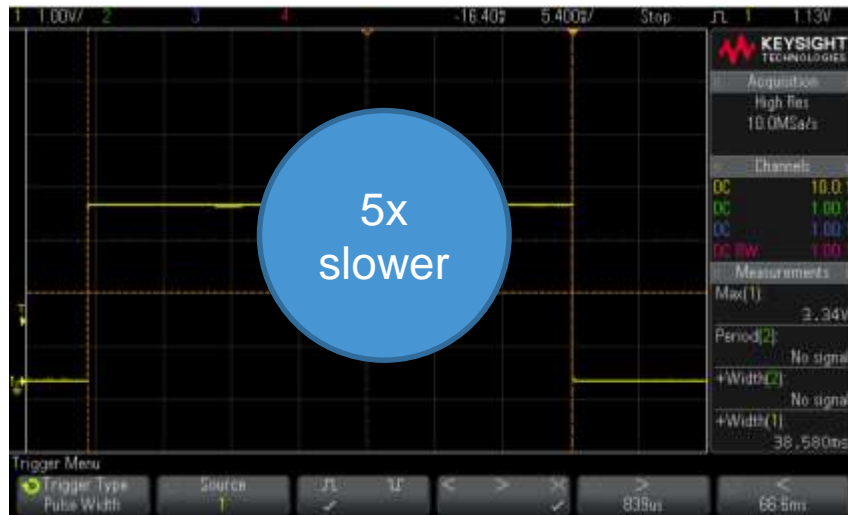
```
void Emvco_Polling(void * pHalParams)  
Board_LED_Set(LED_NUM, 1);  
  
for(i=0;i<10;i++)  
  
/* Load Emvco Default setting */  
status = LoadEmvcoSettings();  
CHECK_STATUS(status);  
  
/* Perform RF Reset */  
status = EmvcoRFReset();  
CHECK_STATUS(status);  
  
status = pHalHw_SetConfig(pHal, PHAL_HW_CONFIG_SET_EMD, PH_OFF);  
CHECK_STATUS(status);  
  
Board_LED_Set(LED_NUM, 0);
```

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



# Measured Time for EMVCo Loopback

Linux setup



+Width(1):  
38.580ms

Measured time for a EMVCo loopback transaction takes **33.5ms**

```
*phStatus EMVCoDataLoopBack(...){  
1. Set_GPIO(High);  
2. EMVCoDataExchange(...);  
3. phhalHw_PN5180_WriteRegister();
```

Bare metal setup



+Width(1):  
6.955ms

Measured time for a EMVCo loopback transaction takes **6.9ms**

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

# Real Time

What is real-time



# Real Time – Definition

## What is Real Time?

- In case a system needs to execute a certain action or task within a given time frame then we are talking about real time
- Hard real-time means that exceeding this time frame is not allowed and could lead to malfunction/failure
- In Soft real-time there is no hard deadline but rather a typical limit until certain tasks can be finished
- Firm real-time also “allows” exceeding the deadline, but the result could be invalid/outdated

## Hard real-time – Examples

- During EMVCo L1 certification of a terminal the measured guard time between a WUPA and a WUPB must not exceed 10ms
- So, the system must guarantee that the WUPB frame is sent after latest 10ms
- If this is not achieved the device is not EMVCo L1 compliant and fails certification

## Soft real-time – Examples

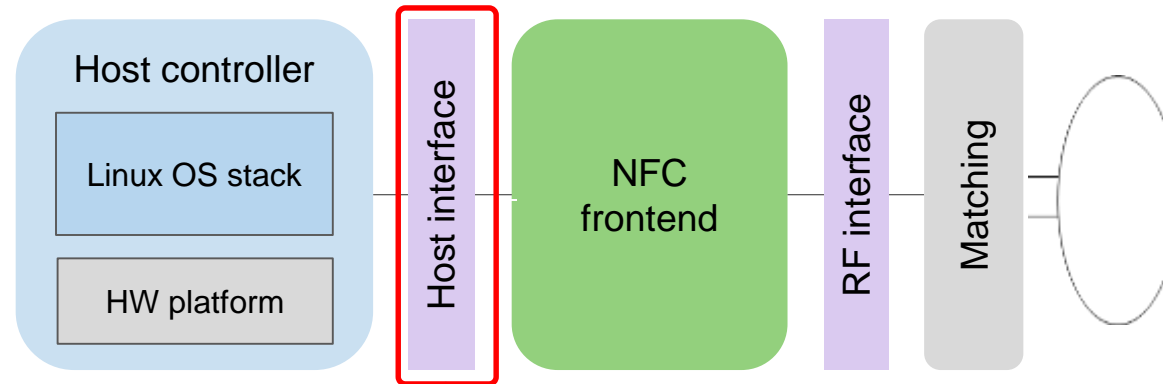
- The same terminal in field operation should not exceed the guard time of 10ms between a WUPA and a WUPB
- If it's exceeded the system is still working and operable with typically no negative impact

# Overcoming Linux Higher Latency for Time-sensitive Applications



# Recommendations to Reduce Linux Latency

Linux-based NFC reader architecture



The major parameter influencing the Linux latency is the large time required to access the host interface from the host controller due to the Linux SW stack architecture

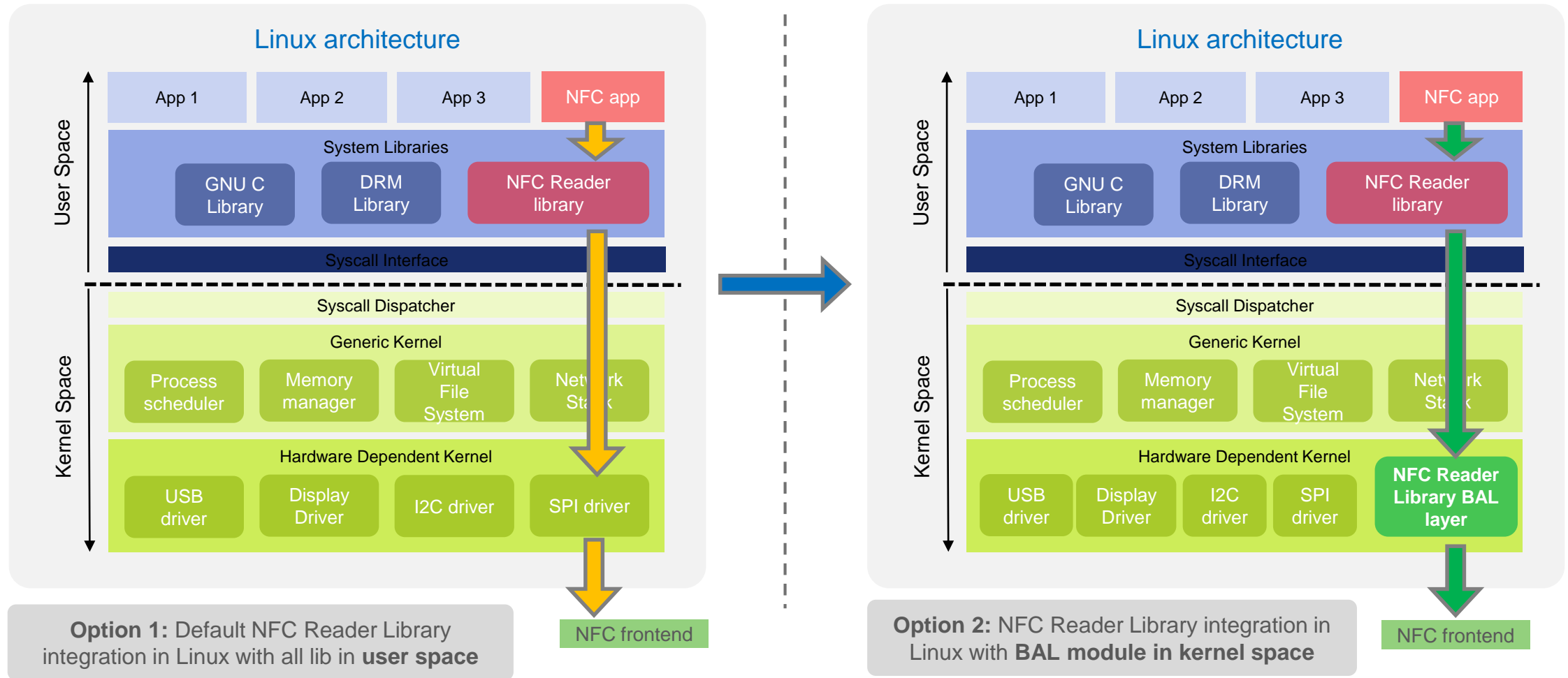
Solutions

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

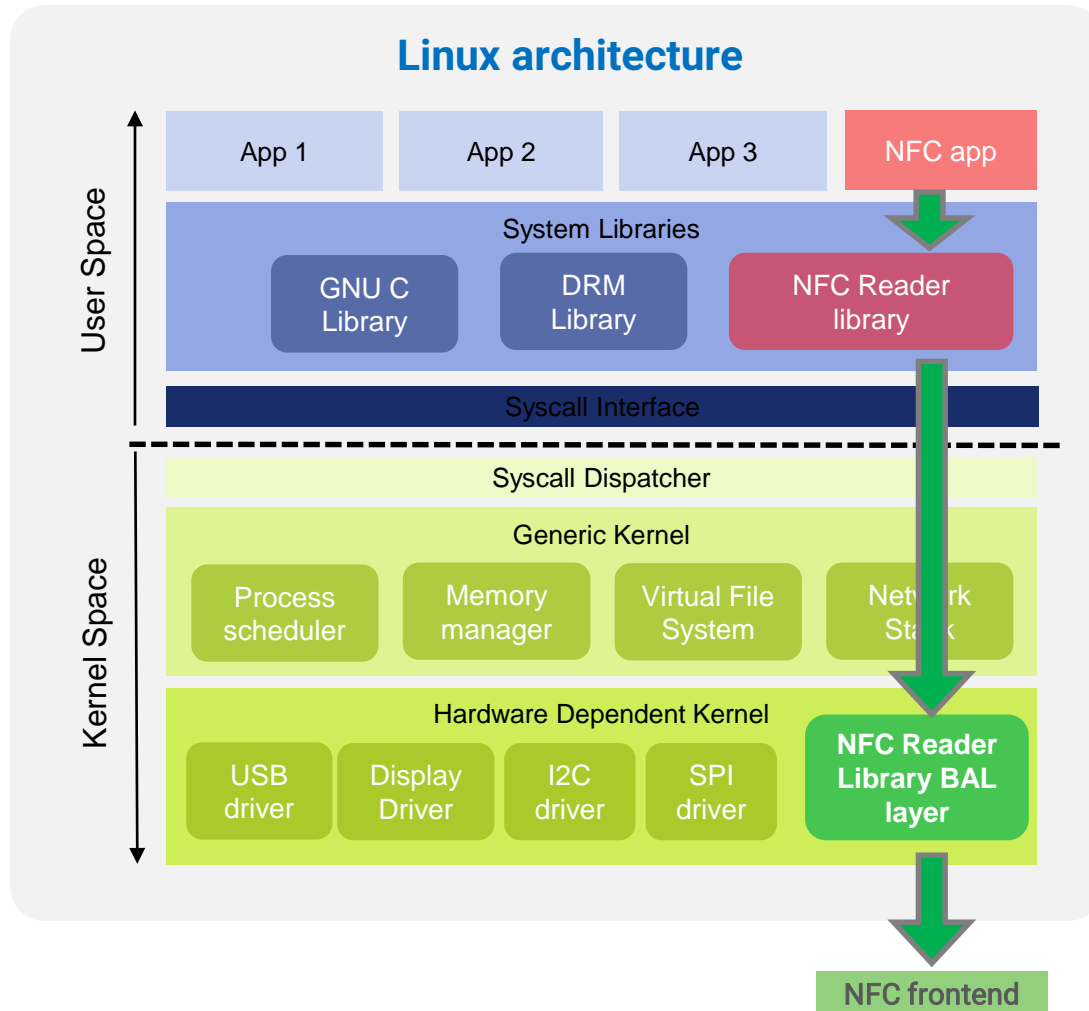
The most effective solution !!

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

# NFC Reader Library Support of BAL Module in Kernel Space



# 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

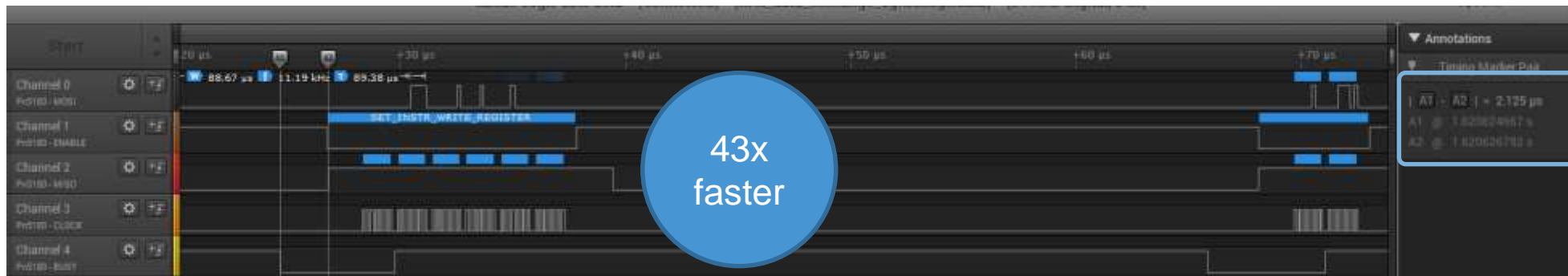
# NFC Reader Library BAL Module: User Space vs Kernel Space

**BAL layer in User Space:** Measured time for setting up one SPI transfer (Raspberry Pi 2 running Linux OS)



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

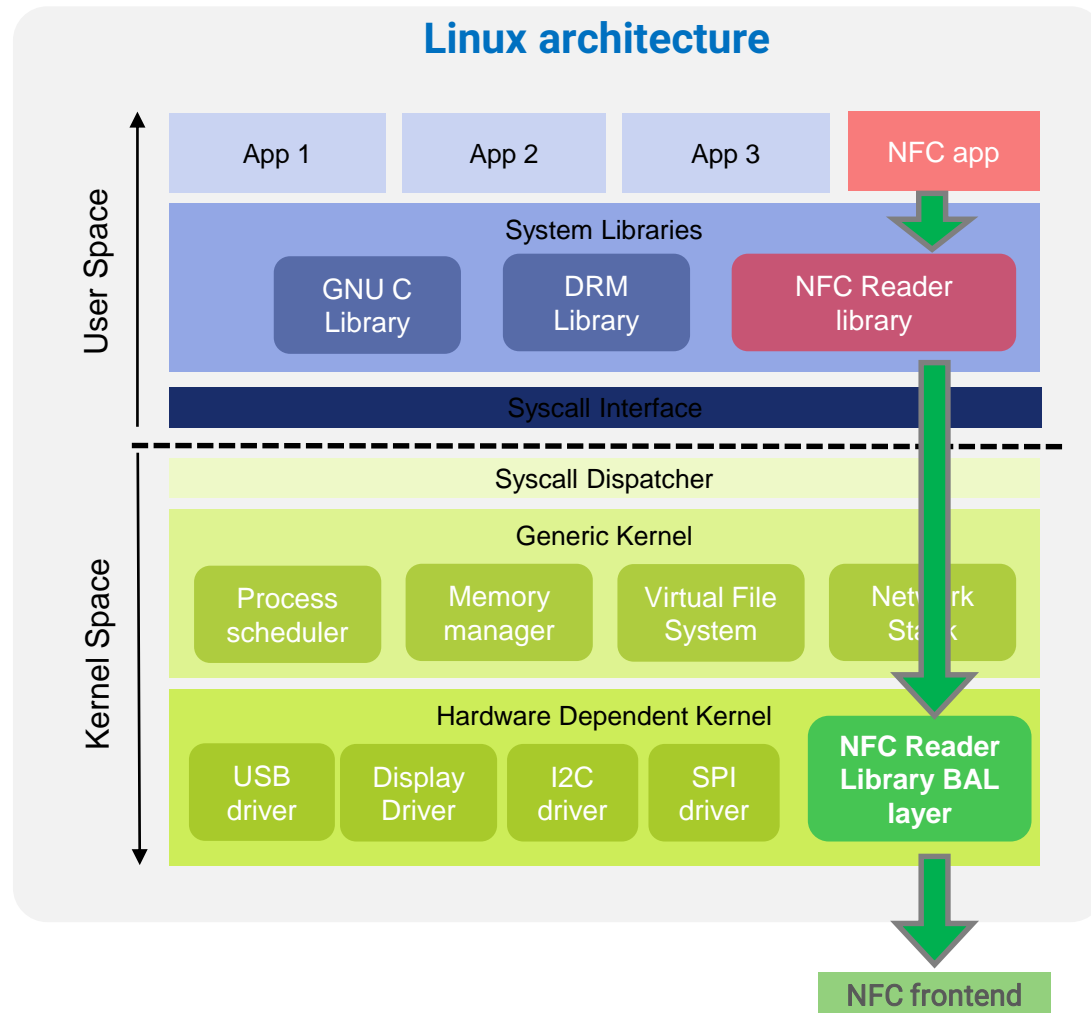
**BAL layer in Kernel Space:** Measured time for setting up one SPI transfer (Raspberry Pi 2 running Linux OS)



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



# NFC Reader Library BAL Module in Kernel Space: Resources



[1] GitHub repo with:

- Information about building, configuring and
- An example the integration on Raspberry Pi is given.

[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/nxprplib-kernel-bal>

[2] [http://www.nxp.com/documents/application\\_note/AN11802.pdf](http://www.nxp.com/documents/application_note/AN11802.pdf)

# Further Considerations



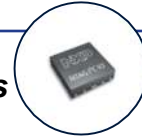


- Changing the scheduling policy
  - FIFO and RT.
- RT-Preempt Linux Kernel patch [1]
  - Not part of Linux mainline. Needs to be applied manually.
- Dedicated MCU for timing sensitive parts
  - E.g. i.MX6 CPU with dedicated Cortex-M4 core.

[1] [https://rt.wiki.kernel.org/index.php/Main\\_Page](https://rt.wiki.kernel.org/index.php/Main_Page)

# Selecting the Right Product



# Latest Non-mobile NFC Products

	PN5180 	CLRC663 plus 	NTAG I <sup>2</sup> C plus 	PN7462 & derivatives 	PN7150 
<b>Commercial tagline</b>	<i>The best full NFC frontend in the market</i>	<i>Best performance at lowest power consumption</i>	<i>Simplest and lowest BoM NFC solution</i>	<i>All-in-one full NFC solutions</i>	<i>Best plug'n play, high-performance full NFC solution - Easy integration into any OS environment</i>
<b>Positioning</b>	Building on NXP's trusted leadership in the core NFC markets	NXP next-gen multi-protocol NFC frontend	Easy and reliable entry to the world of NFC, incl. password protection	The true innovation: The all-in-one product	Following the success of PN7120, PN7150 brings the same plug 'n play experience with higher performance
<b>Target markets</b>	Payment Access	Access Payment Gaming Industrial	IoT Mass market	Access Gaming Home banking	IoT Consumer Mass market
Required NFC know-how, targeted applications	<ul style="list-style-type: none"> <li>› NFC experts who want to further optimize and/or customize their implementation</li> </ul>	<ul style="list-style-type: none"> <li>› NFC intermediates</li> <li>› High performance with low power requirements for the most demanding applications</li> </ul>	<ul style="list-style-type: none"> <li>› NFC beginners</li> <li>› Simple applications, where no reader functionality needed</li> <li>› Applications requiring simple protection of data</li> </ul>	<ul style="list-style-type: none"> <li>› Applications requiring multiple functionalities (NFC, CT, USB)</li> <li>› Freely programmable</li> </ul>	<ul style="list-style-type: none"> <li>› NFC integration into Linux and Android</li> <li>› Small and medium sized enterprises (SMEs)</li> </ul>



**SECURE CONNECTIONS  
FOR A SMARTER WORLD**