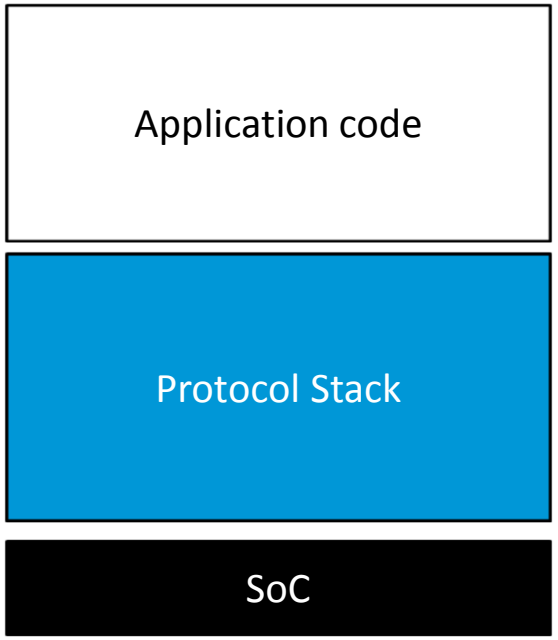
nRF51 series
Software Architecture

NORDIC SEMICONDUCTOR

The ideal SoC software architecture

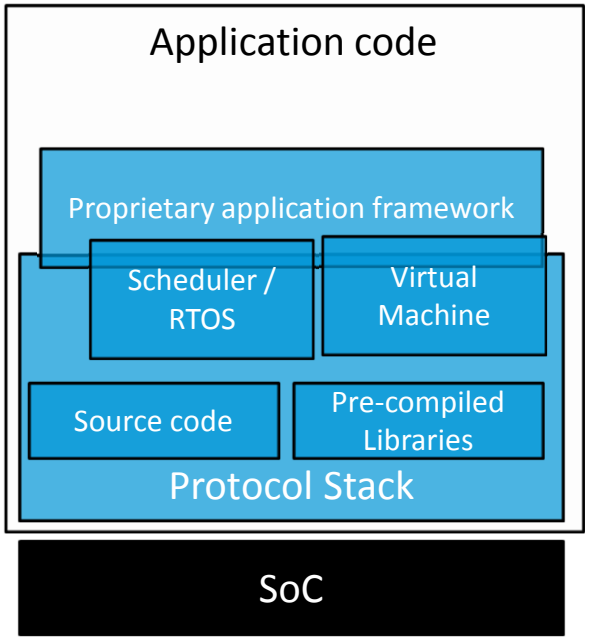
And how everyone presents it….



- Clean separation of application and stack code
- Independent development
  - Application
  - Protocol stack
- Independent testing and verification
- No linkages or run-time dependencies
- Reusable and portable application code
- Easy, fast and low risk application code development
- In theory…simple and risk-free

The harsh reality

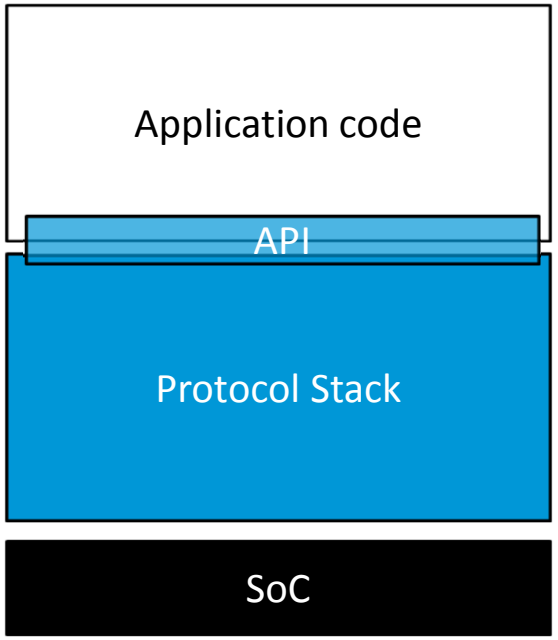
The typical picture for developers…



- No clean separation
  - One big compile
  - Link time dependencies
- Proprietary framework
  - Virtual machines…
  - Scheduler / RTOS dependencies
- Complex run-time dependencies
- Plenty of dependencies for development
- Plenty of dependencies for testing and verification
- Dependencies to IC vendor
- Bugs in app code affect stack and vice-versa
- Not reusable or portable application code

nRF51 is different

As close as you can get to the ideal situation



- Clean separation of application and stack code
  - Two different compiles
  - No link time dependencies
- Asynchronous, event-driven API
- No proprietary application framework
  - Application developer sees a standard Cortex-M
  - No scheduler or RTOS dependencies
  - Full control over the choice of operating environment
- Run-time protected stack
- Independent development
- Independent testing and verification
- Easy code porting, migration and reuse

Hardware abstraction layer

Cortex Microcontroller Software Interface Standard (CMSIS)

- Vendor-independent hardware abstraction layer for the Cortex-M processor series
- Enables consistent and simple software interfaces to the processor and the peripherals
- System functions

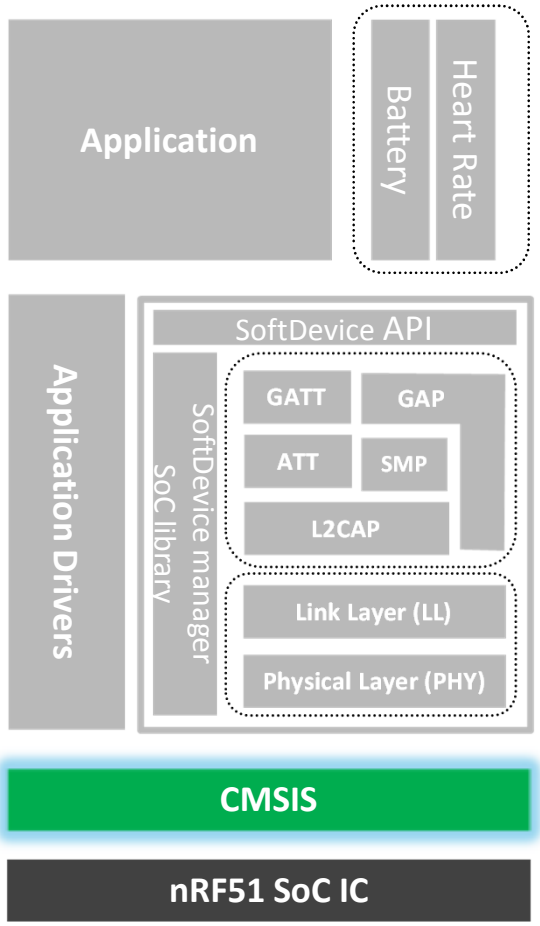
```
void SystemInit(void)
```

- Cortex-M0 interface functions

```
void NVIC_EnableIRQ(IRQn_Type IRQn)
```

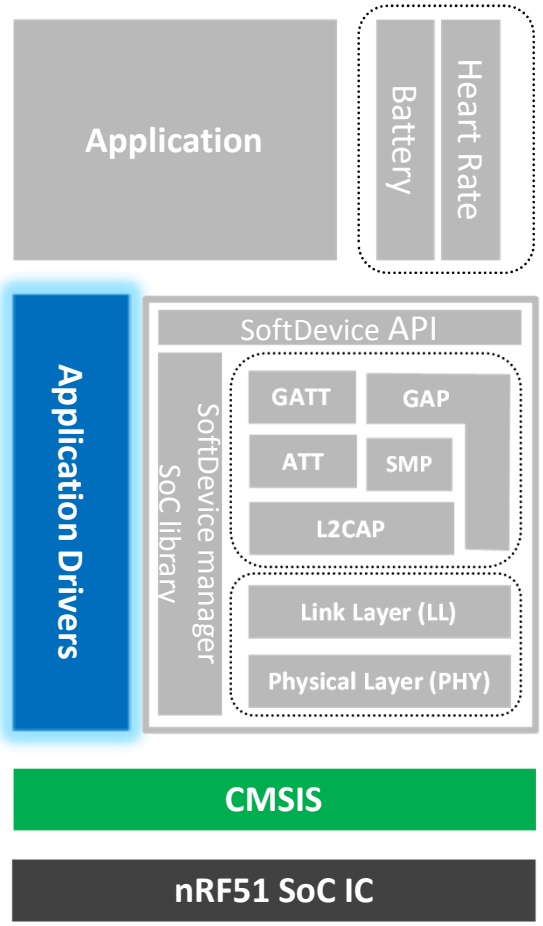
- Register memory mapping

```
NRF_POWER->SYSTEMOFF
```



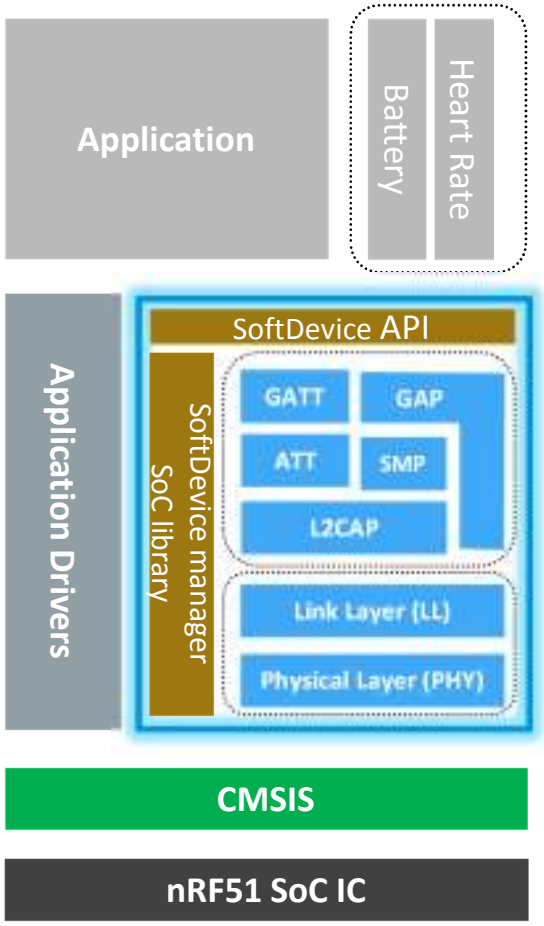
Application Drivers

- Drivers for accessing the SoC peripherals via CMSIS
- Provided libraries:
  - SPI Slave, SPI Master, 2 wire-interface
  - App Timer, App UART
  - Flash, ECB block
  - GPIO, GPIOTE
  - External sensors
  - Etc.
- Libraries are provided in the SDK for easy application development



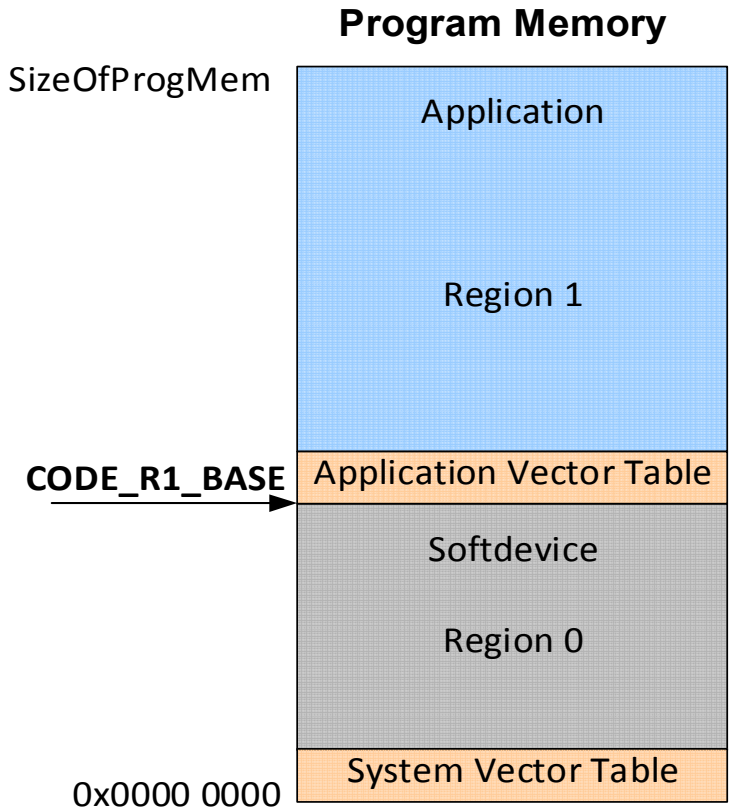
SoftDevice

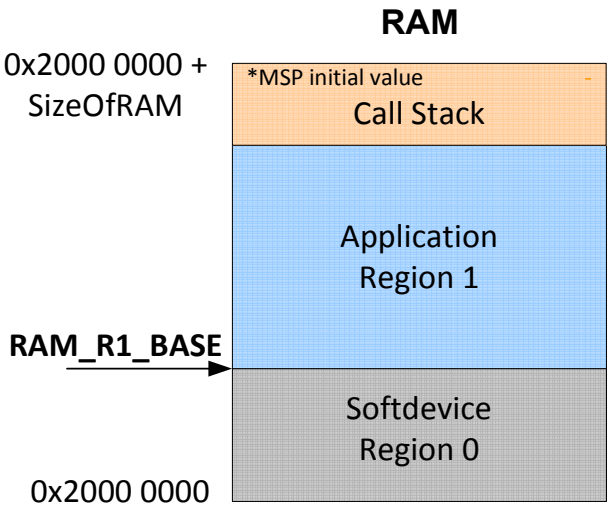
- What is a SoftDevice
  - Stand alone pre compiled FW block
  - Programmed separately from application
  - No link time dependency on application
- What does it contain?
  - ANT or BLE or combination protocol stacks
  - Support modules
  - Keeps stack and application separate
- Located in a reserved memory space
  - Ensures run time protection
- Service (system) Call (SVC) based APIs
- 100% event-driven
  - No RTOS dependencies



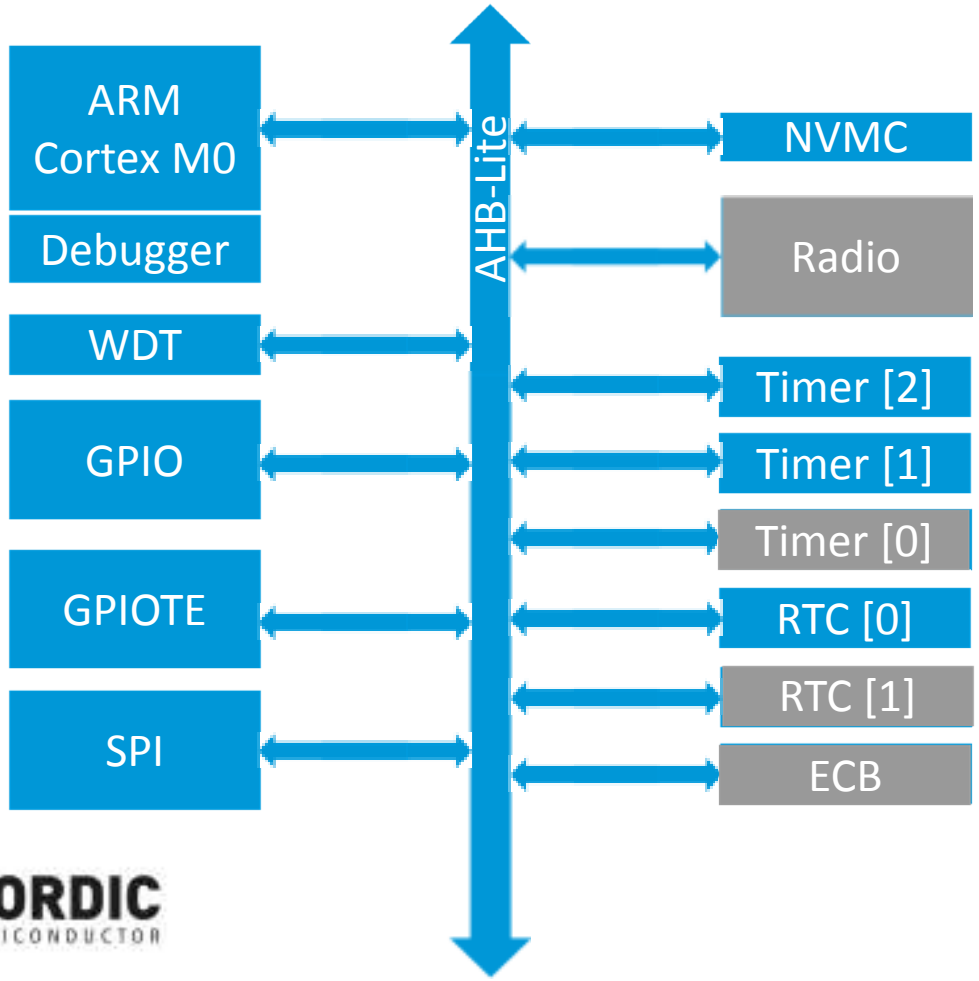
Example: S110 *Bluetooth* low energy Software stack

Softdevice memory layout





SoftDevice HW block protection



SoftDevice: ENABLED

Application has Open access

SoftDevice Restricts or Blocks access

SoftDevice State - Enable & Disable

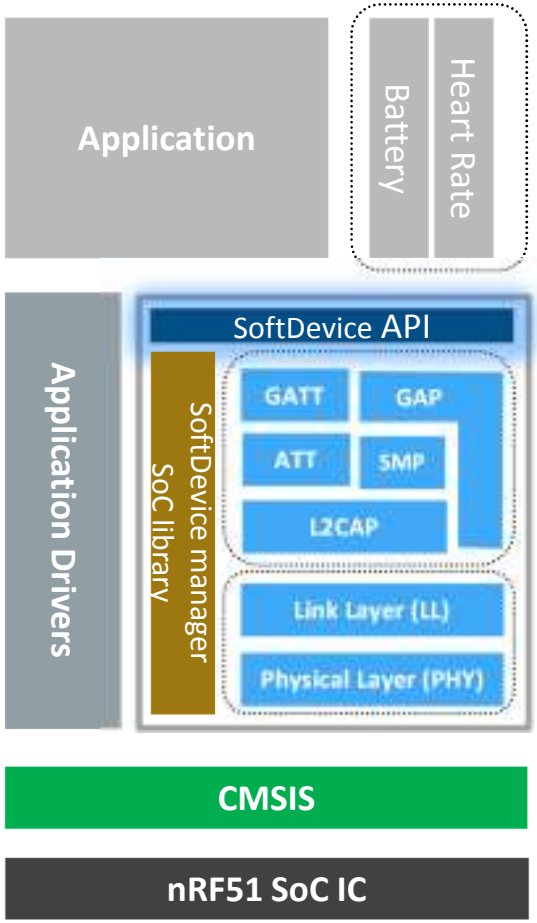
```
// Enable the SoftDevice
sd_softdevice_enable(…);

// Disable the SoftDevice
sd_softdevice_disable();
```

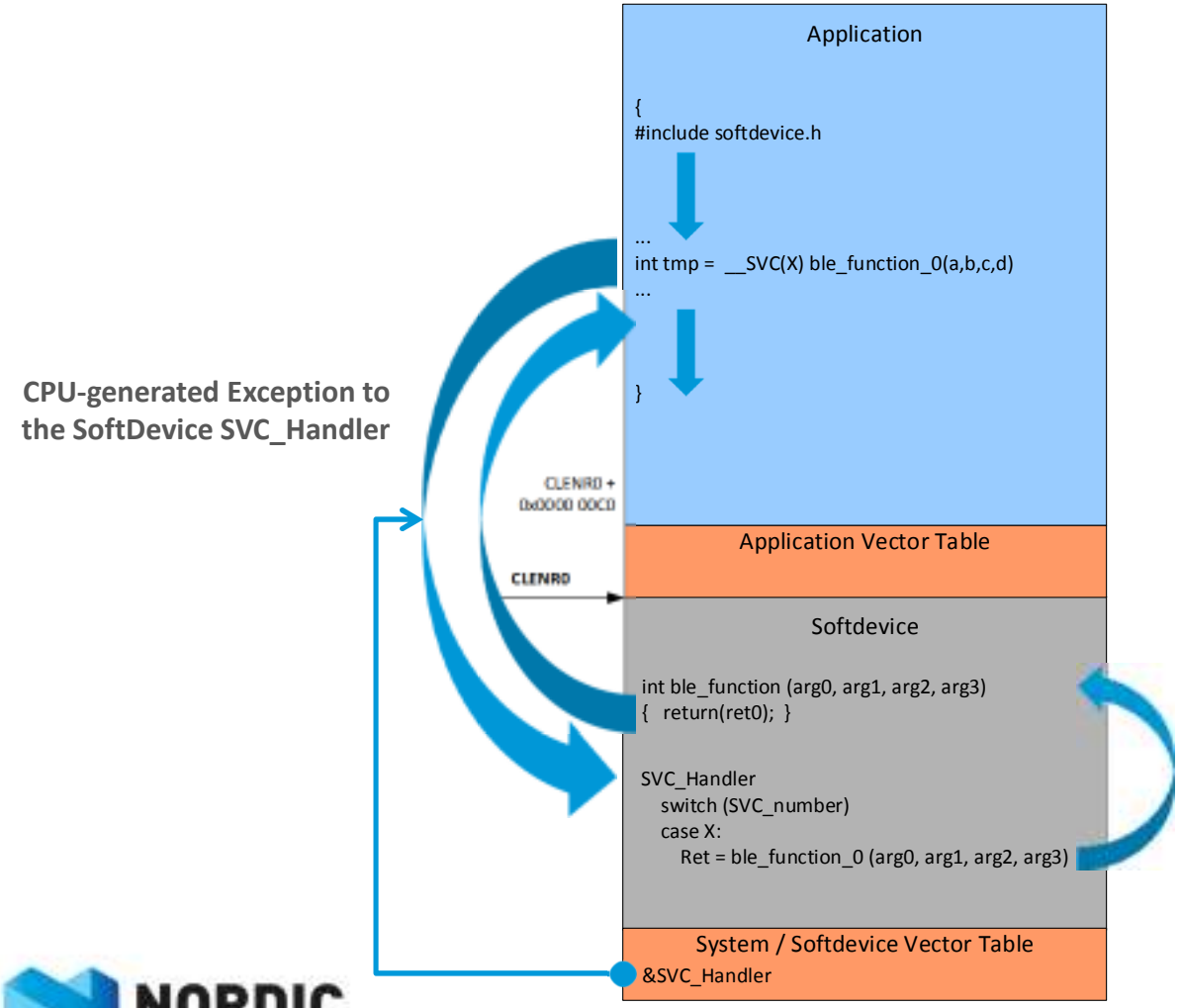
SoftDevice Disabled (default)	SoftDevice Enabled
All HW can be used by the Application	Some HW blocks used by the SoftDevice are protected
Softdevice API is not available, except `sd_softdevice_enable();`	The whole API, inc. all BLE functions, is available
All RAM can be used by the Application	Part of RAM used and protected by the SoftDevice
All Exceptions forwarded to Application	Exceptions for protected HW are handled in SoftDevice

Softdevice APIs

- Softdevice provides a list of APIs as the interface for application.
  - Protocol APIs give access to RF Protocol functionality
  - nRF APIs give access to the SoC restricted resource used by Softdevice
- APIs are implemented as supervisor calls (SVC)
  - An API call triggers an exception with a SVC number and is handled by SVC handler in the stack
  - API calls are non-blocking
- Examples:
  - `sd_softdevice_enable(…);`
  - `sd_ble_gap_adv_start(…);`
  - `sd_flash_write(…);`
  - `sd_ppi_channel_assign(…);`

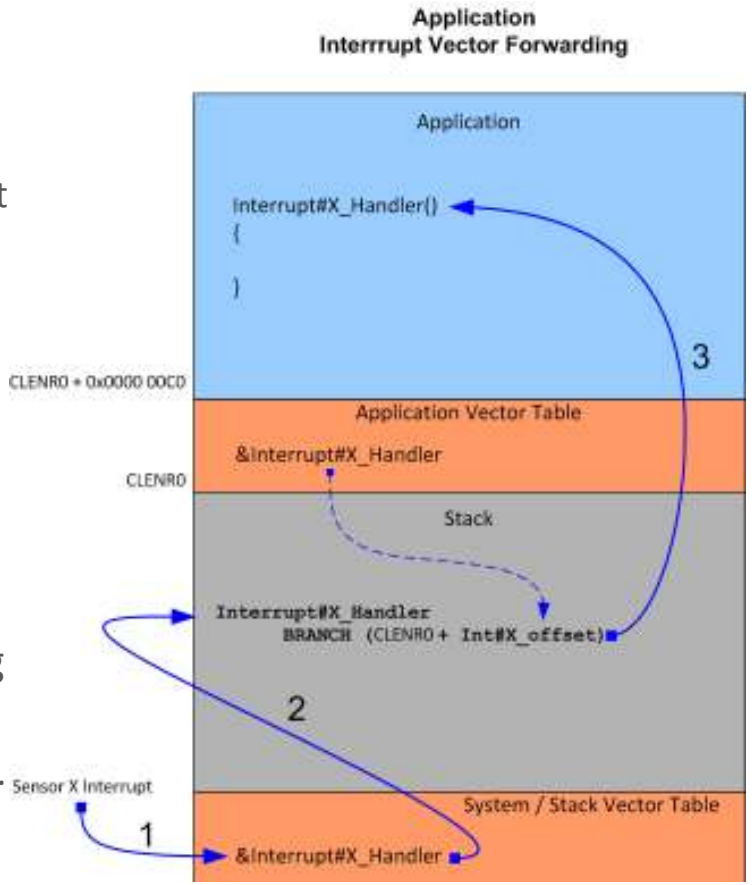


API call using SVC from Application to SoftDevice

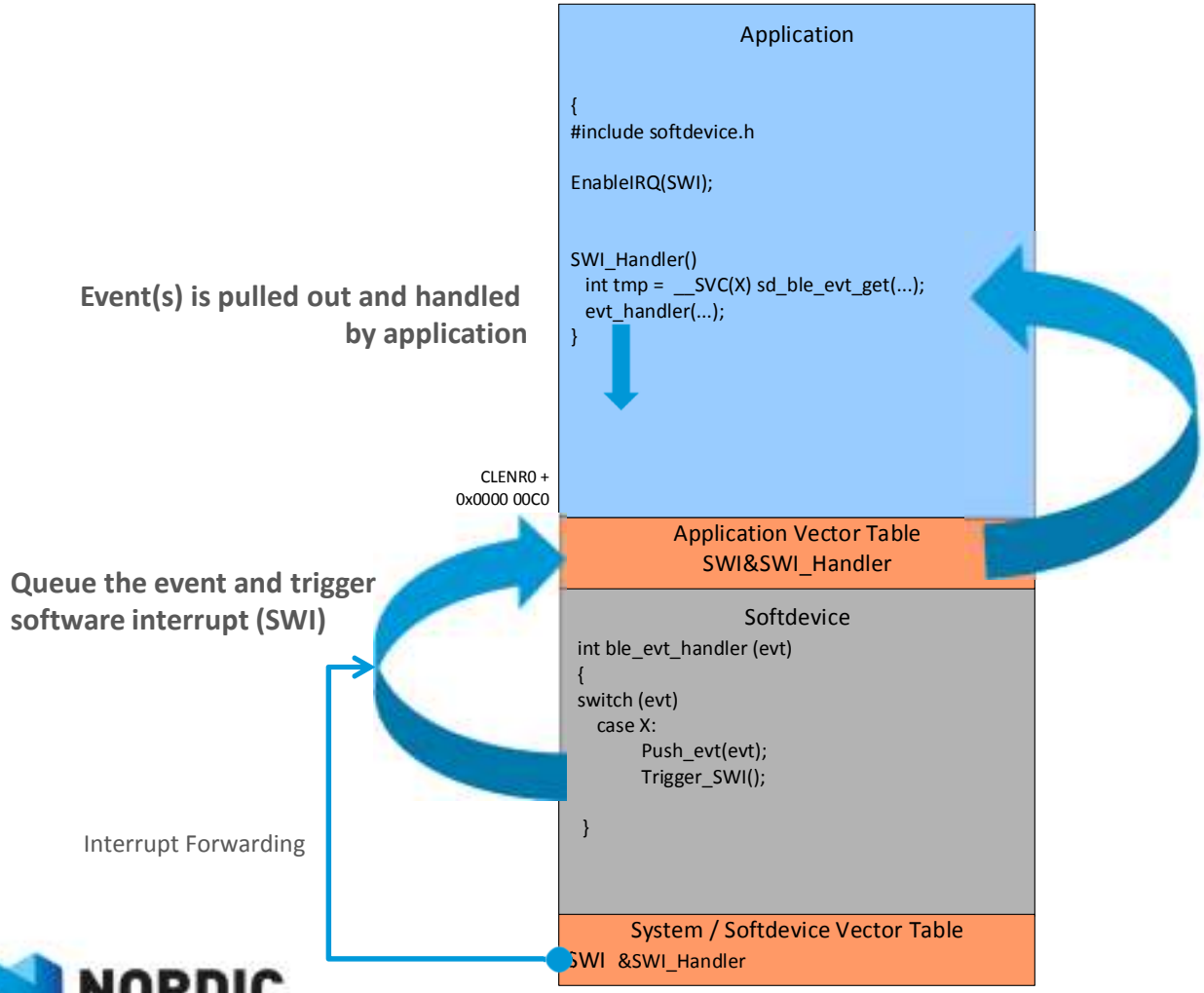


Interrupt forwarding, SoftDevice to Application

- SoftDevice controls the actual M0 vector table
  - Application defines its own vector table
  - i.e. Application same as with no SoftDevice present
  - Soft device handles vector forwarding
- SoftDevice enabled:
  - Vectors not used by the SoftDevice forwarded
- SoftDevice disabled
  - All vectors forwarded
- Due to this vector forwarding interrupt handling will be delayed by a few clock cycles
  - How many is given in each soft device specification.

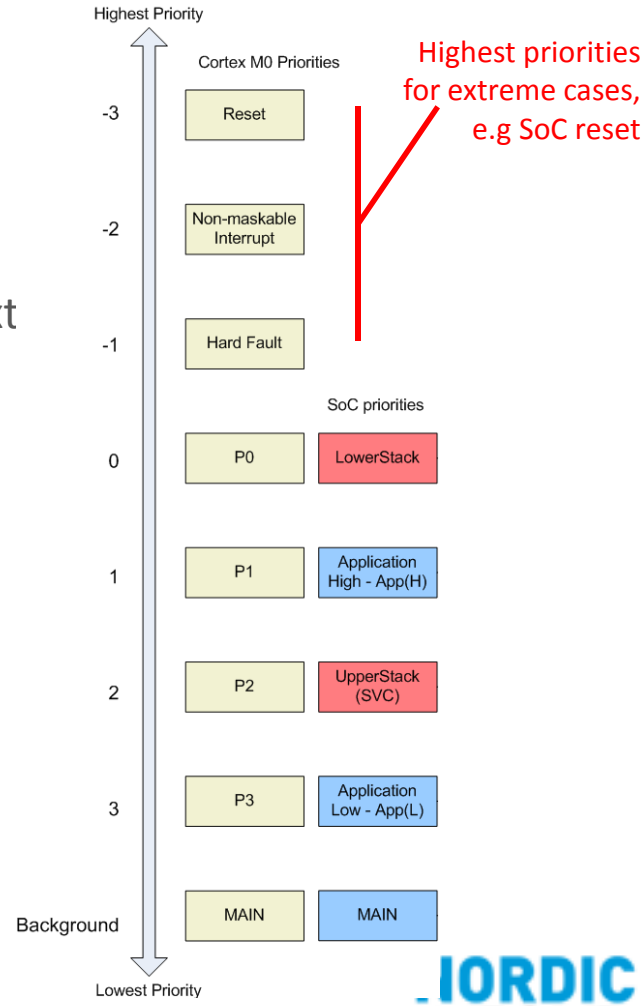


Events & Callbacks, Softdevice to Application

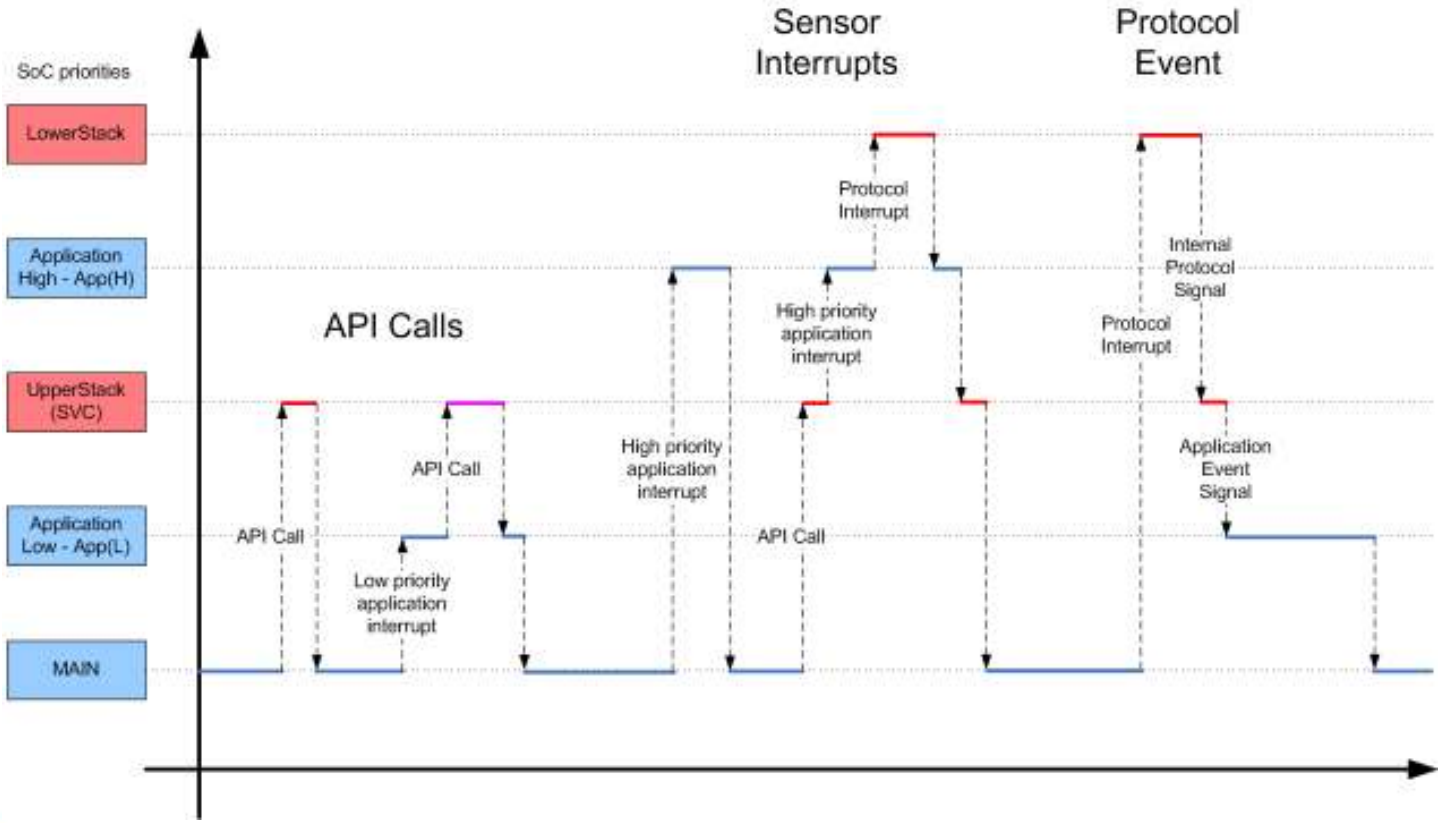


Softdevice@Run-time

- SoftDevice processing are interrupt driven
  - Access to CPU must be shared between Application and SoftDevice interrupts
- Cortex M0 has 4 interrupt priorities and main context
- SoftDevice uses 2 priorities to implement its event-driven behavior
  - Lower Stack (BLE Link Layer),
  - Upper Stack (SVC API, BLE Host)
- Application have access to 2 priorities and main context
  - Application High – for critical interrupts where low latency is required. Cannot call the SVC API.
  - Application Low. Used for Host events.



Example of nested interrupt handling



Summary

Software stack compatibility

Mix and match to your ideal

Bluetooth® SMART Peripheral Software Stack

S110 80kB

OR

Bluetooth® SMART Central Software Stack

S120 96kB

ANT/Bluetooth® SMART Peripheral Software Stack

S310 128kB

8-channel ANT Software Stack

S210 48kB

NORDIC SEMICONDUCTOR

NORDIC tech TOUR

nRF 51 Software Architecture Benefits

Flexibility	▪ Softdevice can be Enabled and Disabled at run-time to give application full access to HW resources including Radio and RAM ▪ No RTOS dependencies ▪ The application and Stack can be updated (programmed) separately
Ease of Development	▪ Simple application programmer model with no link time dependencies ▪ No proprietary application development model or RTOS ▪ Minor SoftDevice / stack updates do not require application to be re-compiled
Code safety	▪ Stack is not re-linked constantly during application development ▪ QA and qualification on binary going into end-user product ▪ Stack is run-time protected ▪ Reduced risk of application bugs affecting the Stack

nRF51 series
Software Architecture

NORDIC SEMICONDUCTOR