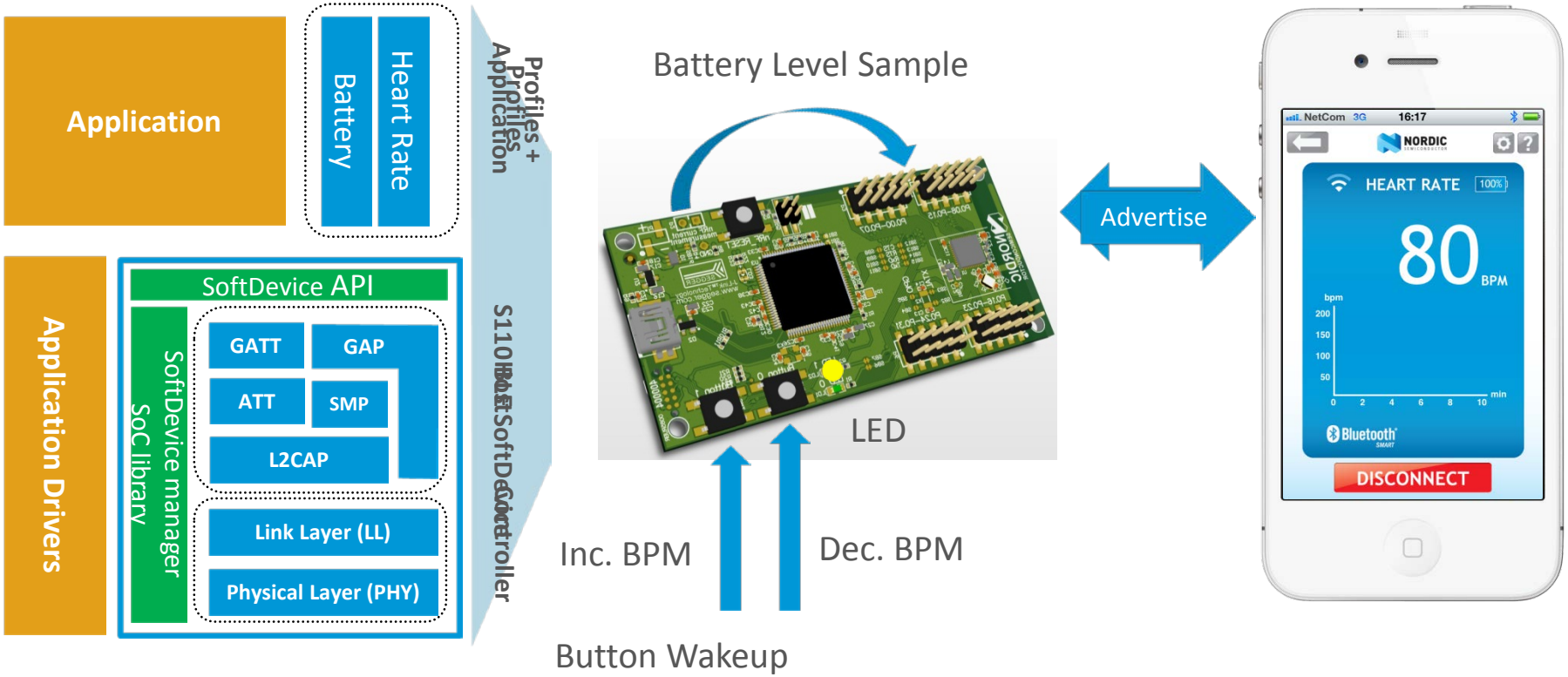
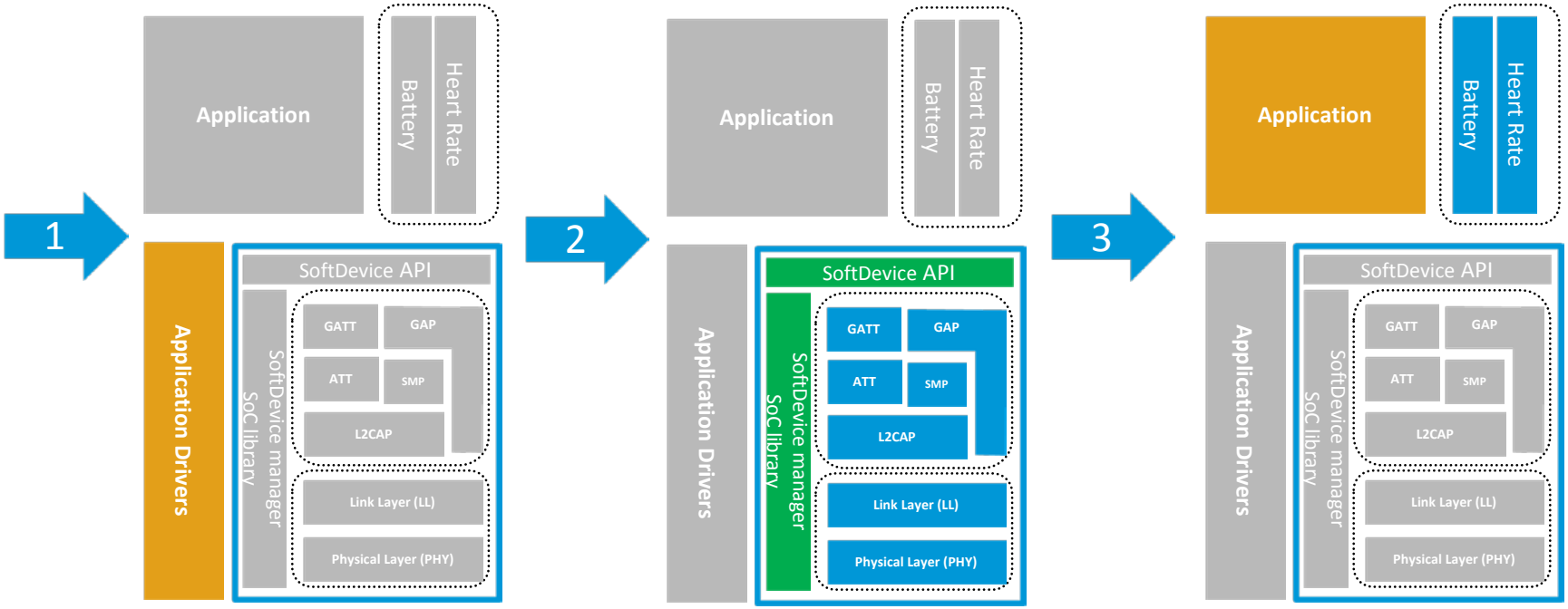
Simple BLE sensor application walk through

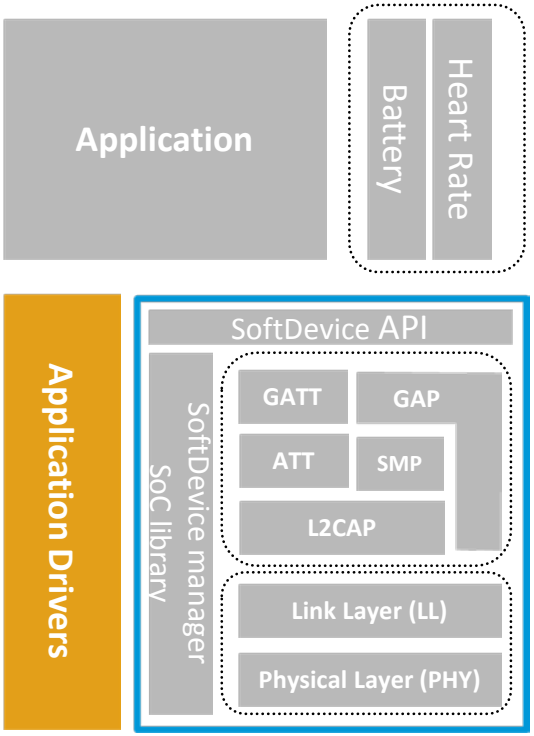
Application Overview – BLE HRS with Eval Kit



Development in 3 parts



Application drivers



CMSIS - Cortex Microcontroller Software Interface Standard

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

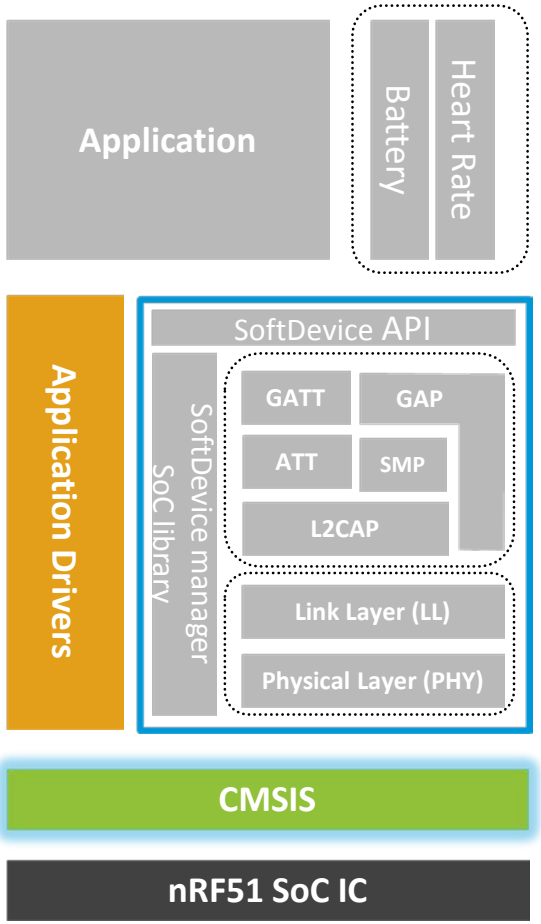
```
void SystemInit(void)
```

  - Cortex-M0 interface functions

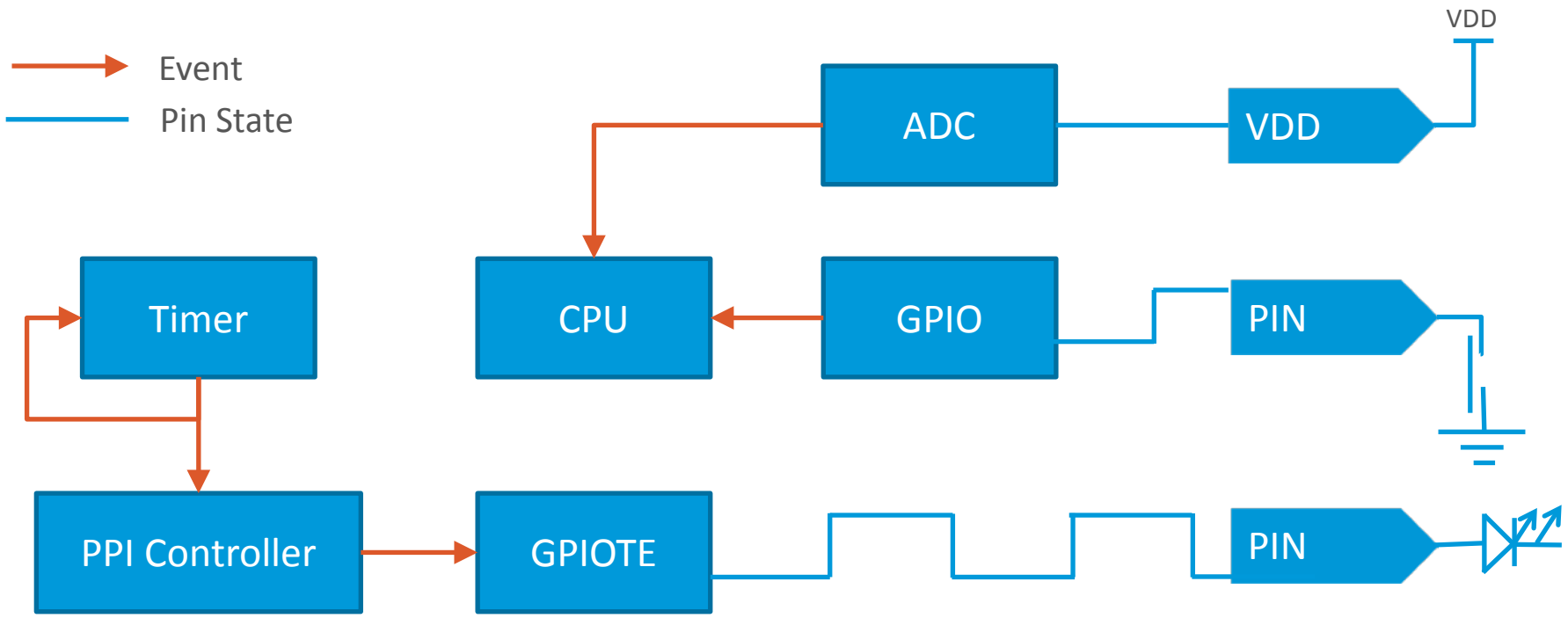
```
void NVIC_EnableIRQ(IRQn_Type IRQn)
```

  - Register memory mapping

```
NRF_POWER->SYSTEMOFF
```



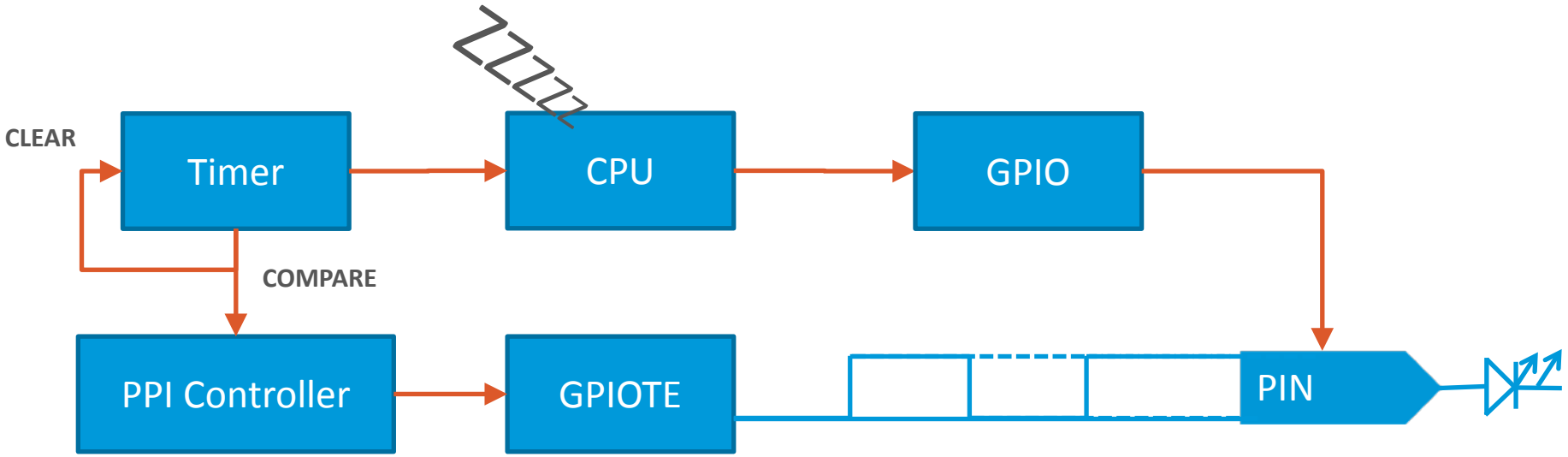
Application drivers block diagram



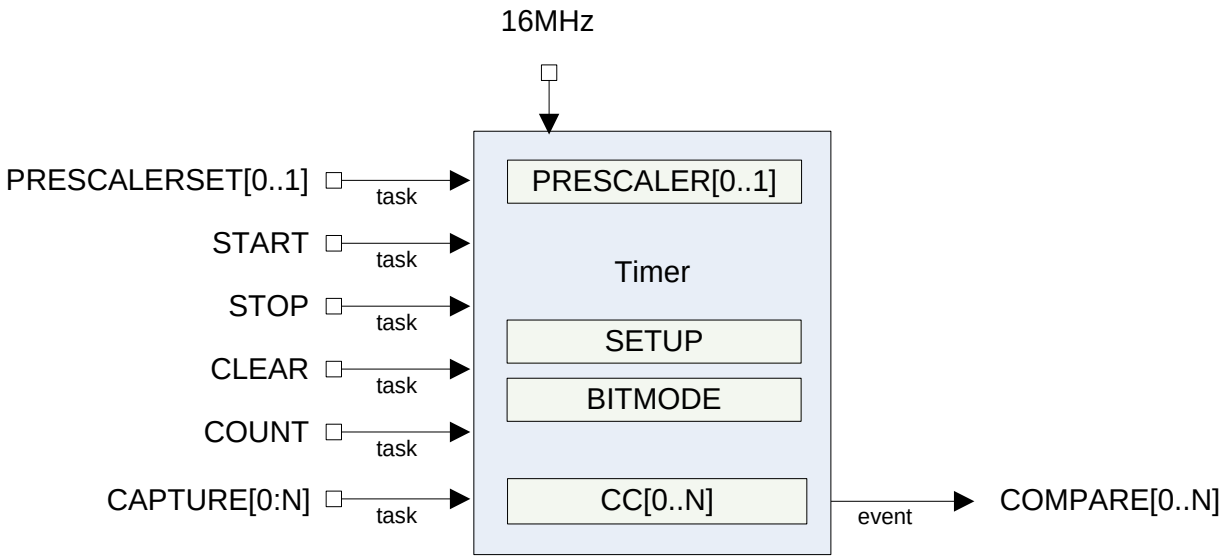
Part 1: Output – LED

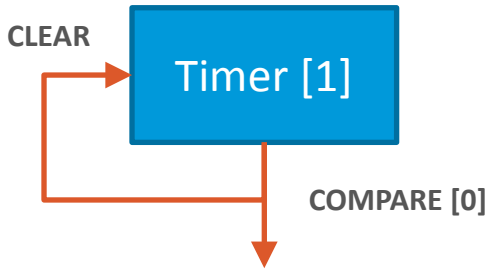
What we will do:

- Blink the LED without CPU (LED Flash)
- Using a Timer, PPI and GPIOTE



Part 1: Timer configuration



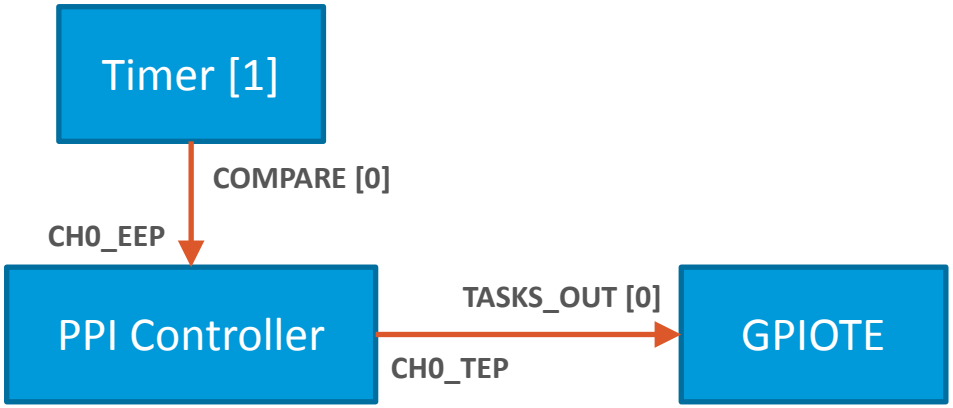


Part 1: Timer configuration (using CMSIS)

```
// Configure timer
NRF_TIMER1->MODE      = TIMER_MODE_MODE_Timer;
NRF_TIMER1->BITMODE   = TIMER_BITMODE_BITMODE_16Bit;
NRF_TIMER1->PRESCALER = 9;    // 31.25kHz → 32us tick
NRF_TIMER1->TASKS_CLEAR = 1;  // Clear the timer
NRF_TIMER1->CC[0] = 0x1E84;   // CC[0] occurs at 250ms from START

// Configure the shortcut to clear and restart on Compare[0]
NRF_TIMER1->SHORTS = (TIMER_SHORTS_COMPARE0_CLEAR_Enabled <<
                      TIMER_SHORTS_COMPARE0_CLEAR_Pos;)
NRF_TIMER1->TASKS_START = 1;
```

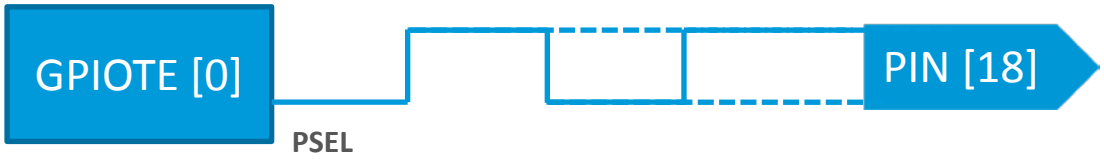
Part 1: PPI configuration (using CMSIS)

```
// Configure PPI channel 0 to set TASKS_OUT[0] on TIMER1 COMPARE[0]
NRF_PPI->CH0_EEP = &(NRF_TIMER1->EVENTS_COMPARE[0]);
NRF_PPI->CH0_TEP = &(NRF_GPIOTE->TASKS_OUT[0]);

// Enable PPI channels 0
NRF_PPI->CHEN = (PPI_CHEN_CH0_Enabled << PPI_CHEN_CH0_Pos);
```



Part 1: GPIOTE configuration (using an SDK helper function)

```
// Configure the GPIOTE Task to toggle the LED state.
nrf_gpiote_task_config(0,  // GPIOTE Channel 0
                       18, // GPIO pin 18
                       NRF_GPIOTE_POLARITY_TOGGLE,
                       NRF_GPIOTE_INITIAL_VALUE_LOW);
```

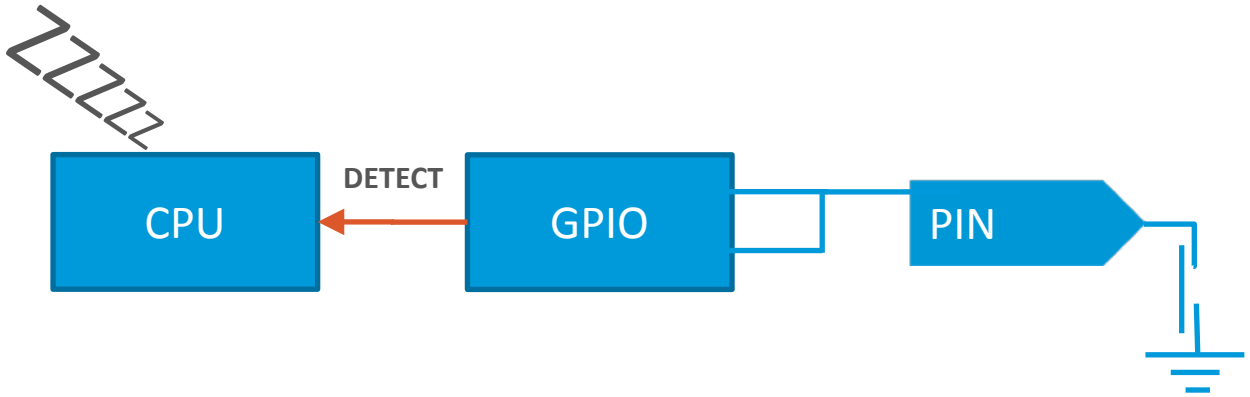


DEMO

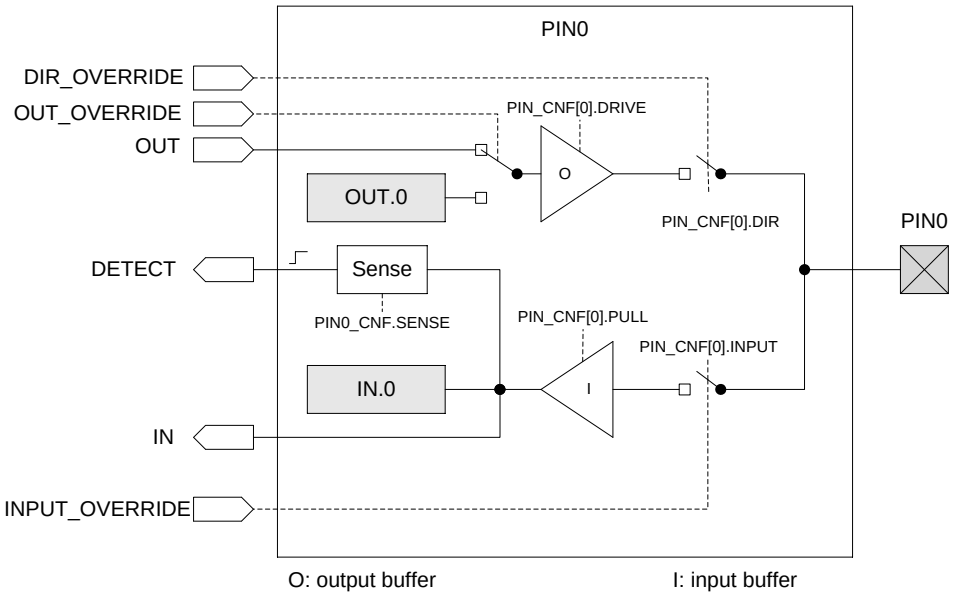
Part 2: Input – Button wakeup

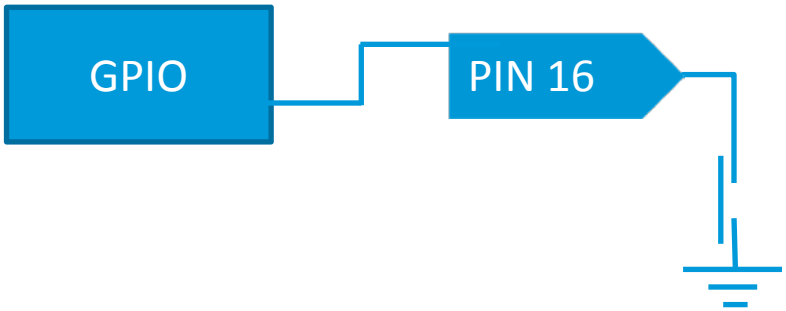
What do we want?

- Button wakeup from System OFF
  - Using GPIO sense / DETECT
- Configure the LED on wakeup



Part 2: GPIO configuration



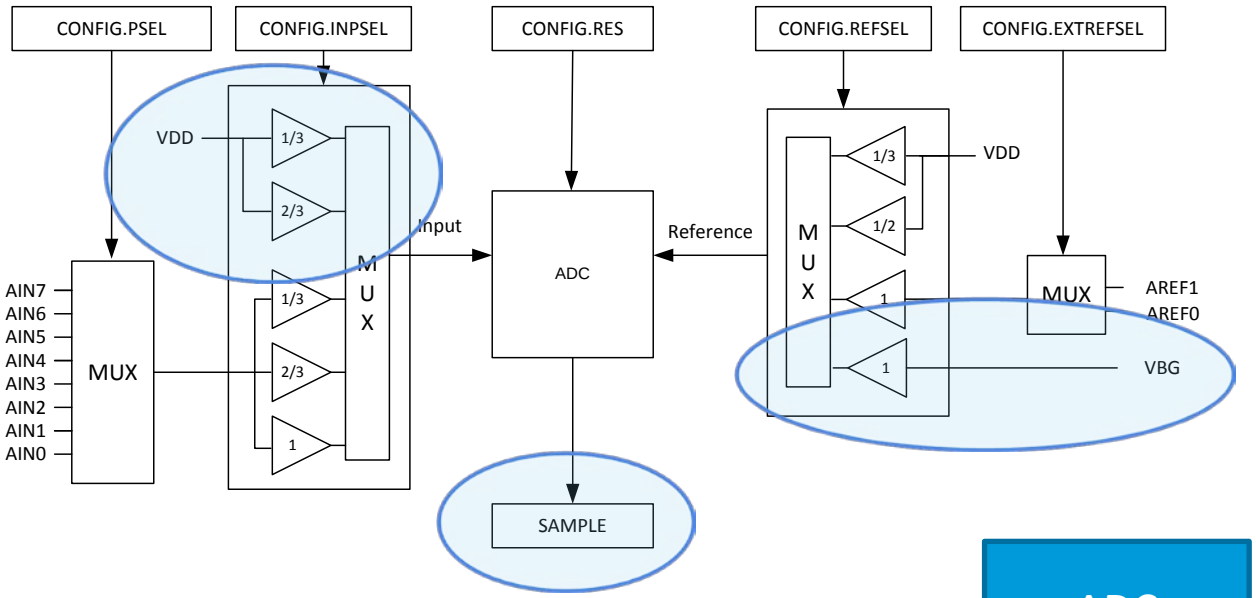


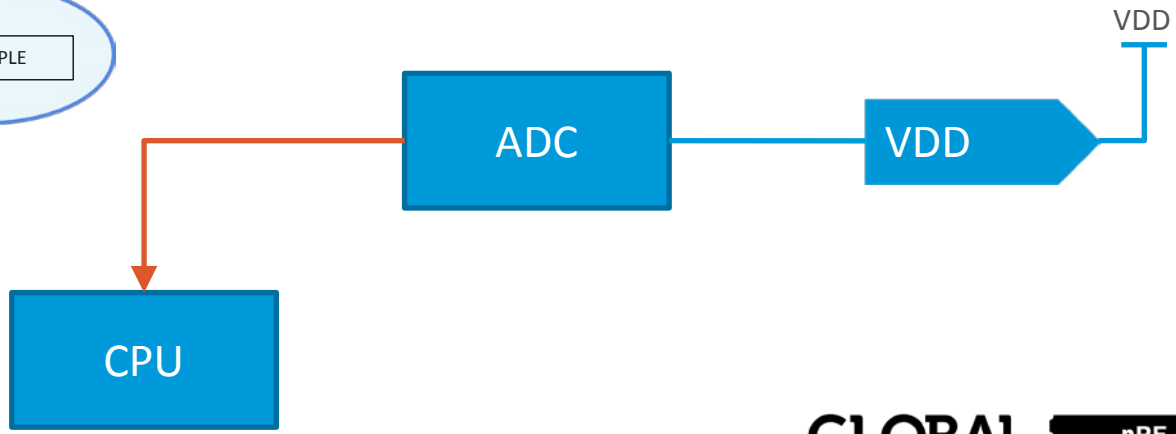
Part 2: GPIO configuration (using CMSIS)

```
// Configure button GPIO
NRF_GPIO->PIN_CNF[16] =
      (GPIO_PIN_CNF_DIR_Input     << GPIO_PIN_CNF_DIR_Pos  )|
      (GPIO_PIN_CNF_INPUT_Connect << GPIO_PIN_CNF_INPUT_Pos)|
      (GPIO_PIN_CNF_PULL_Pullup   << GPIO_PIN_CNF_PULL_Pos )|
      (GPIO_PIN_CNF_DRIVE_S0S1    << GPIO_PIN_CNF_DRIVE_Pos)|
      (GPIO_PIN_CNF_SENSE_Low     << GPIO_PIN_CNF_SENSE_Pos);

// Read button state
state = (NRF_GPIO->IN >> 16) & 0x1;
```

DEMO

Part 3: Exceptions (NVIC) & ADC





Part 3: Exceptions (NVIC) & ADC (using CMSIS)

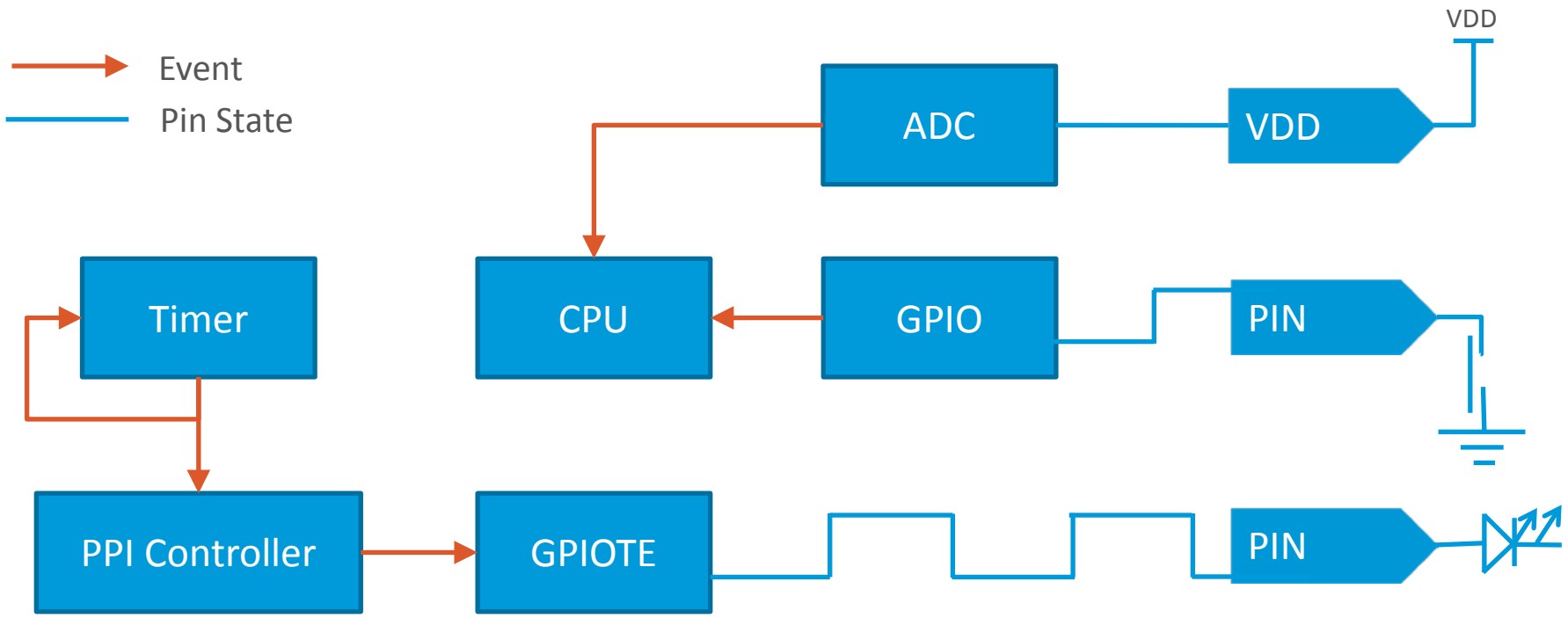
```
// Configure ADC
NRF_ADC->INTENSET   = ADC_INTENSET_END_Msk;

NRF_ADC->CONFIG     =
   (ADC_CONFIG_RES_8bit        << ADC_CONFIG_RES_Pos)   |
   (ADC_CONFIG_REFSEL_VBG      << ADC_CONFIG_REFSEL_Pos)|
   (ADC_CONFIG_INPSEL_VDD13    << ADC_CONFIG_INPSEL_Pos);

NRF_ADC->ENABLE     = ADC_ENABLE_ENABLE_Enabled;
```

Part 3: Exceptions (NVIC) & ADC (using CMSIS)

```
// Enable ADC interrupt and start sample
NVIC_ClearPendingIRQ(ADC_IRQn);

NVIC_SetPriority(ADC_IRQn, 3);

NVIC_EnableIRQ(ADC_IRQn);

NRF_ADC->TASKS_START = 1;
```

DEMO

Application drivers block diagram



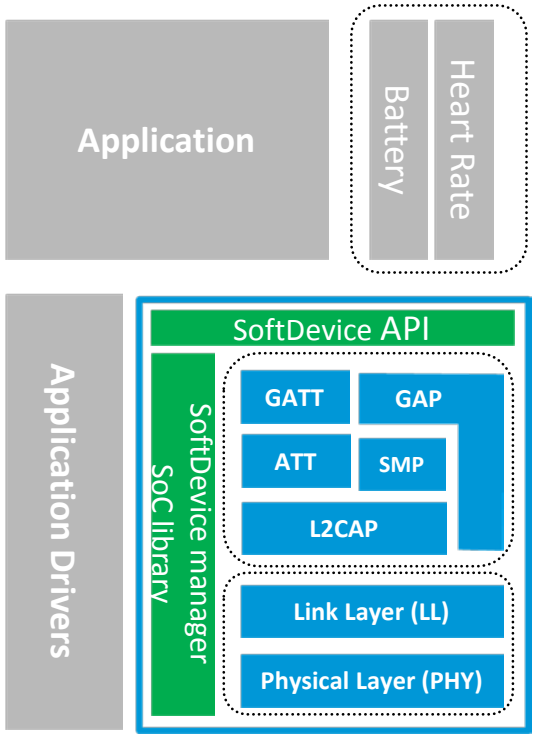
Break

[15 min]

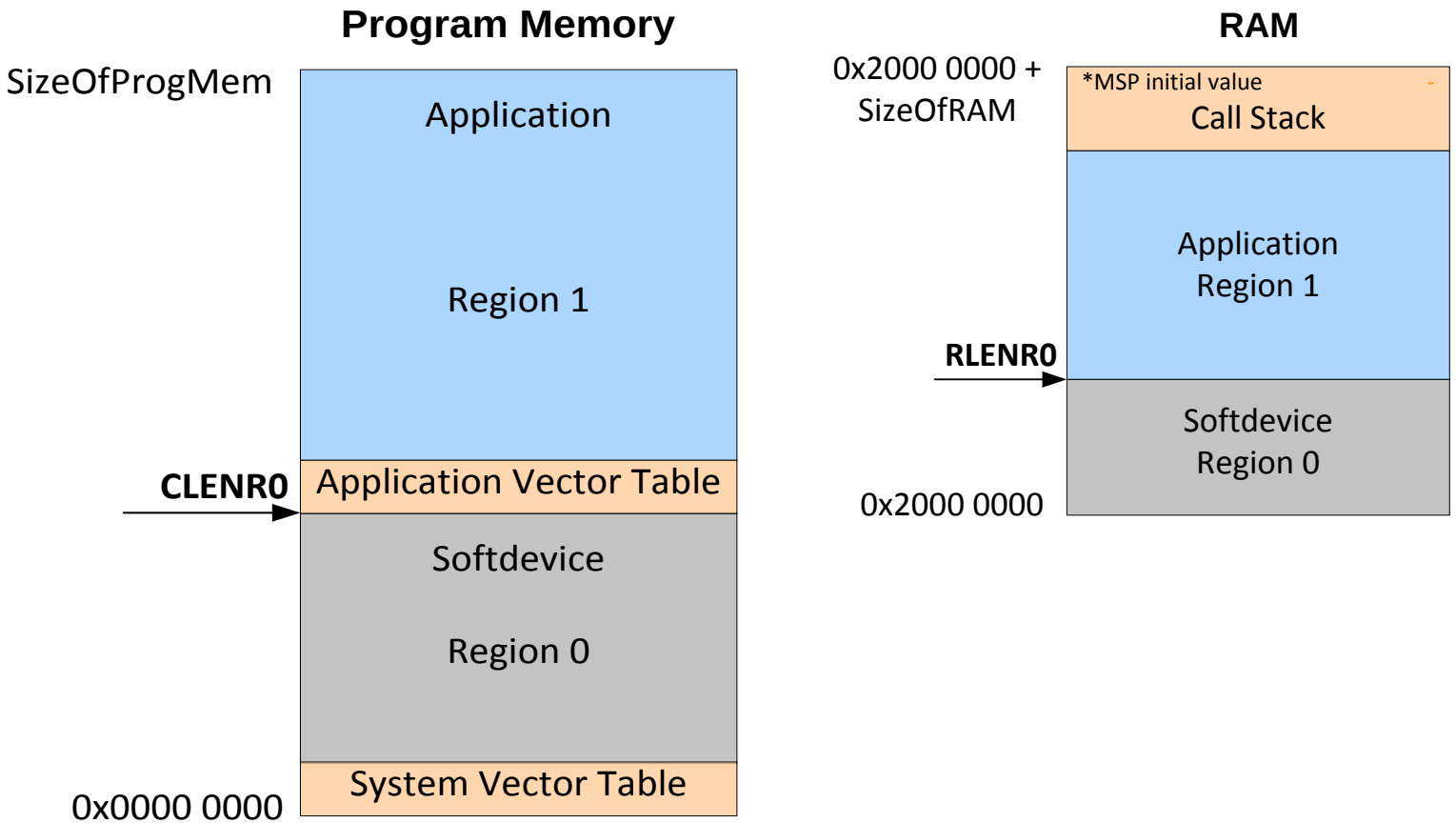
Part 2: nRF51 SoftDevice Architecture

- nRF51 Softdevices
  - Pre-compiled binary wireless protocols
  - Thread-safe SuperVisor Call (SVC) based APIs
  - 100% event-driven
  - No RTOS dependencies
  - Run-time protection of HW and Memory
    - Flash and RAM memory protection for SoftDevice
  - No link time dependencies w/ user application

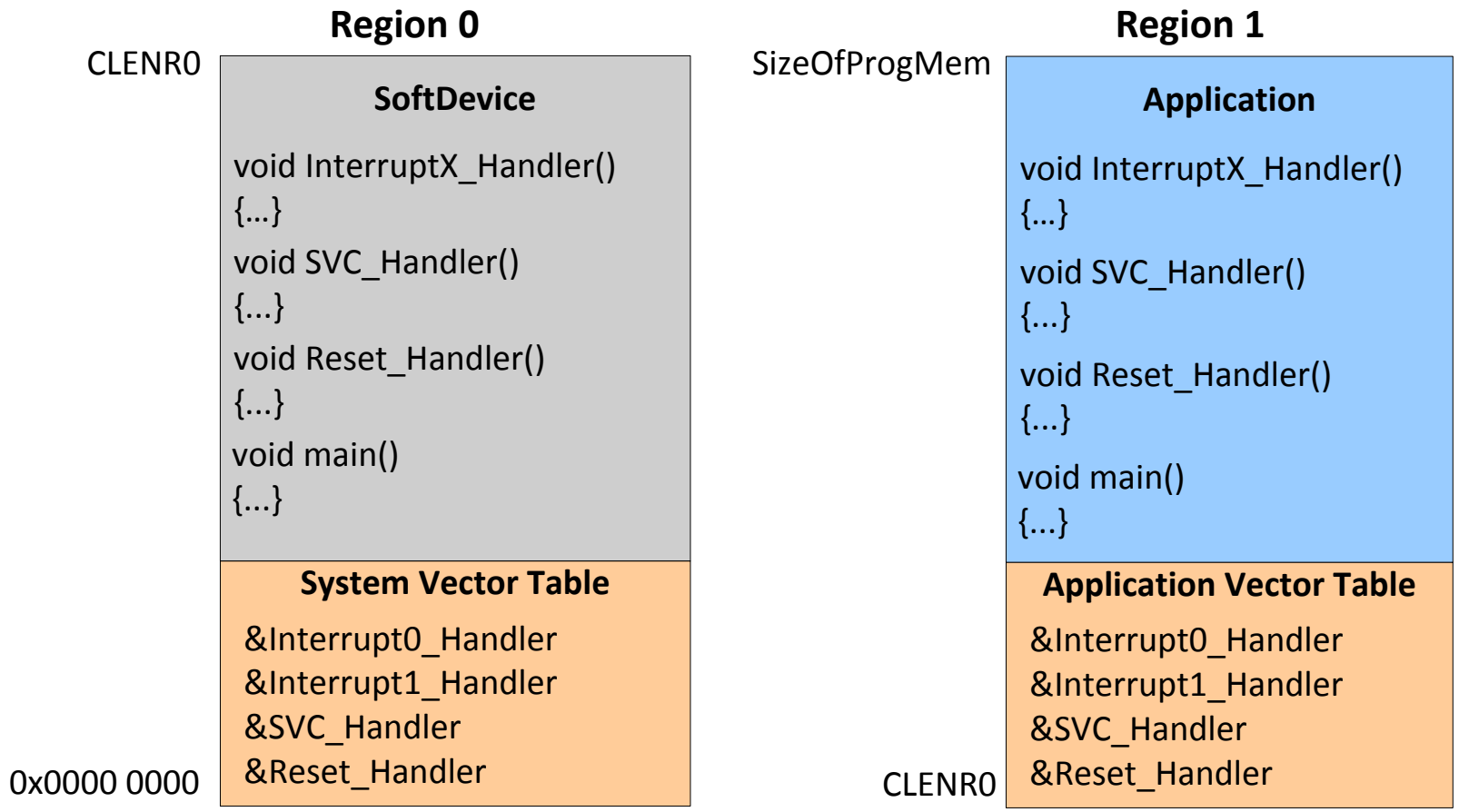
- BLE SoftDevice – Download from our website
- ANT Softdevice – Pre-programmed on nRF51422



Application Programming Model – Memory Regions

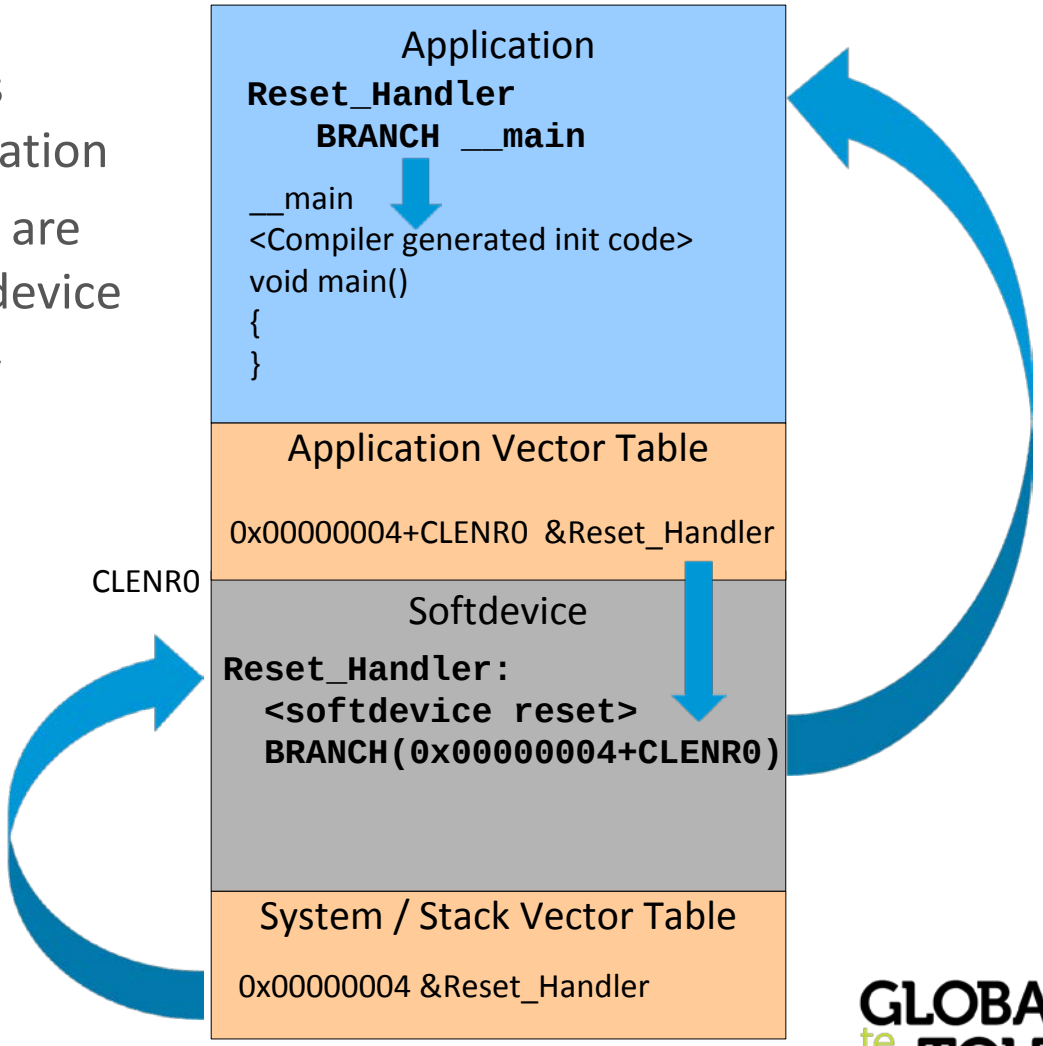


Application Programming Model – 2 programs

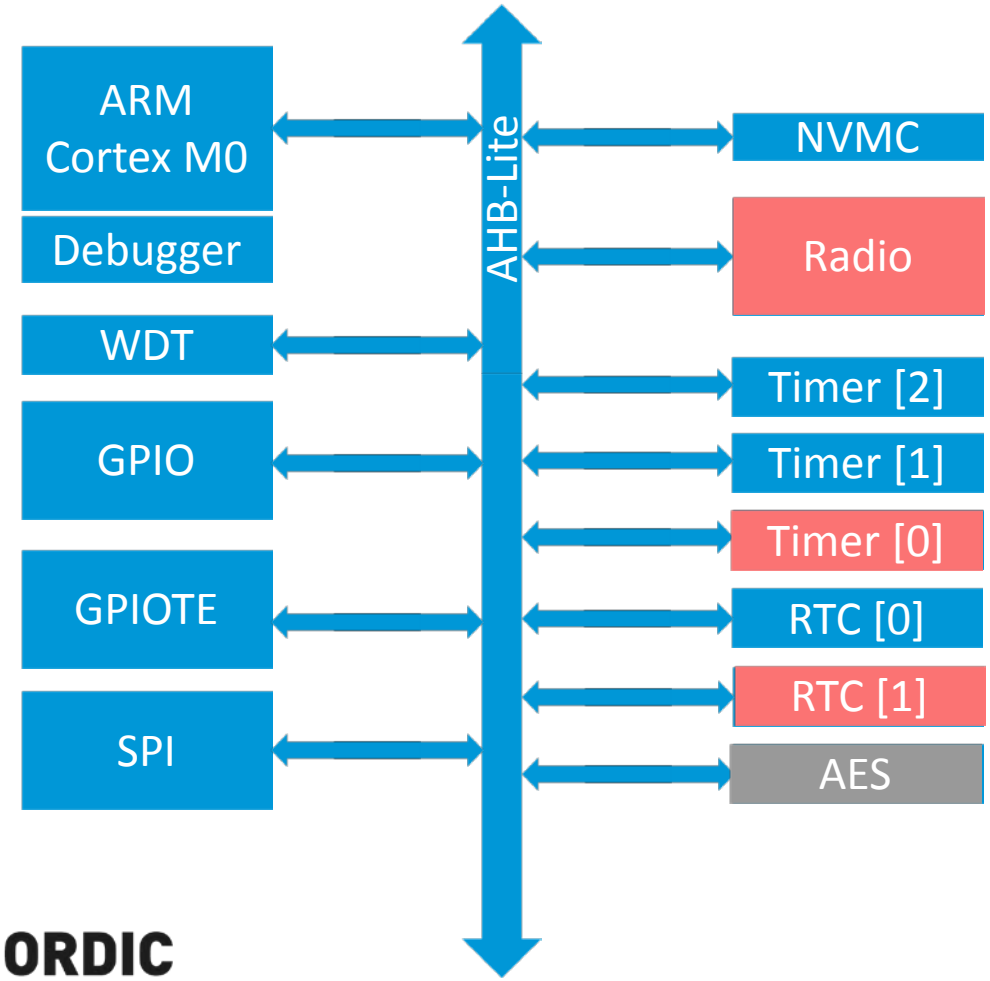


Exception forwarding, SoftDevice to Application

- SoftDevice forwards exceptions to application
- Exceptions handlers are “found” by the softdevice
- Small added latency
  - < 10 microseconds



Softdevice HW block protection



SoftDevice: ENABLED

Application has:

Open access

Blocked access

Restricted access

SoftDevice State - Enable & Disable

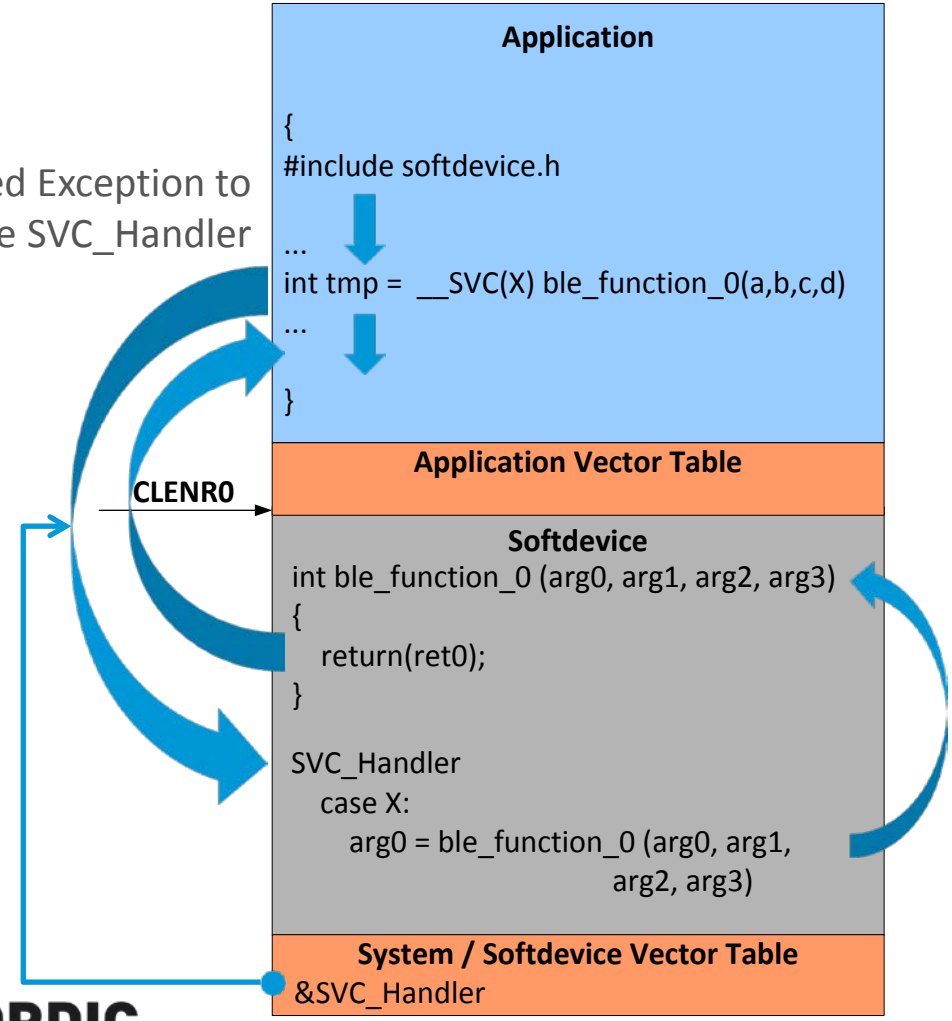
```
// Disable the SoftDevice
nrf_softdevice_disable();
```

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

SoftDevice Disabled (default)	SoftDevice Enabled
Open access to all HW blocks	**Blocked** or **Restricted** access to some HW blocks
The SoftDevice API is not available, except `nrf_softdevice_enable();`	The SoftDevice API, inc. all BLE functions, is available
Application can use all RAM	Application can use part of RAM
All Exceptions forwarded to Application	Exceptions for **Blocked** HW blocks not forwarded to Application

API call using SVC from Application to SoftDevice

CPU-generated Exception to the SoftDevice SVC_Handler



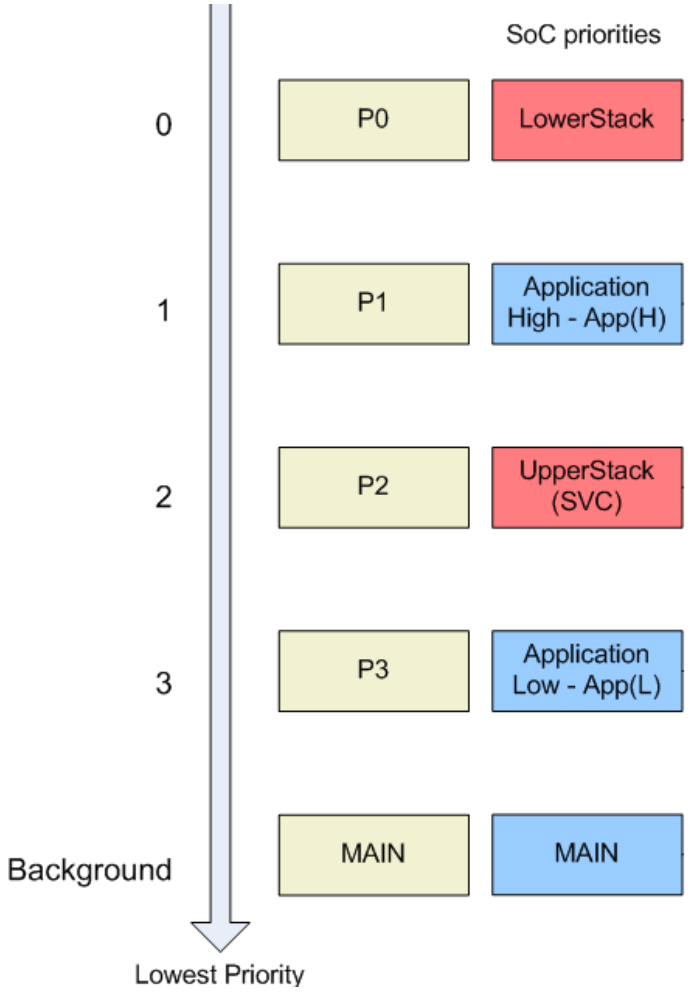
- SuperVisor Calls are exceptions to the SoftDevice
- SoftDevice finds the right function based on SVC number
- Include SoftDevice header files, then call funtions

NORDIC SEMICONDUCTOR

GLOBAL tech TOUR nRF 5 SERIES

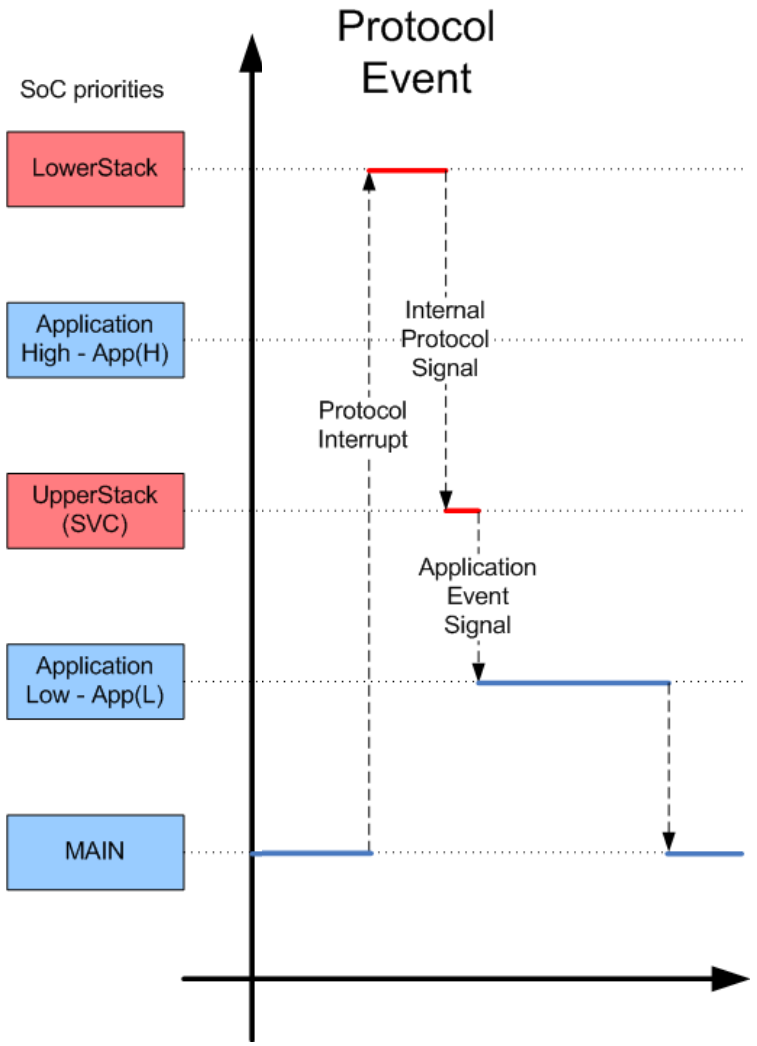
Softdevice@Run-time

- Cortex M0 has 4 interrupt priorities
- 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
  - Application High – for critical interrupts where low latency is required. Cannot call the SVC API.
  - Application Low. Used for Host events.



Softdevice@Run-time

- API Calls
  - Made from MAIN & App(L)
- Protocol Event
  - Signaling from LowerStack done with chained exceptions



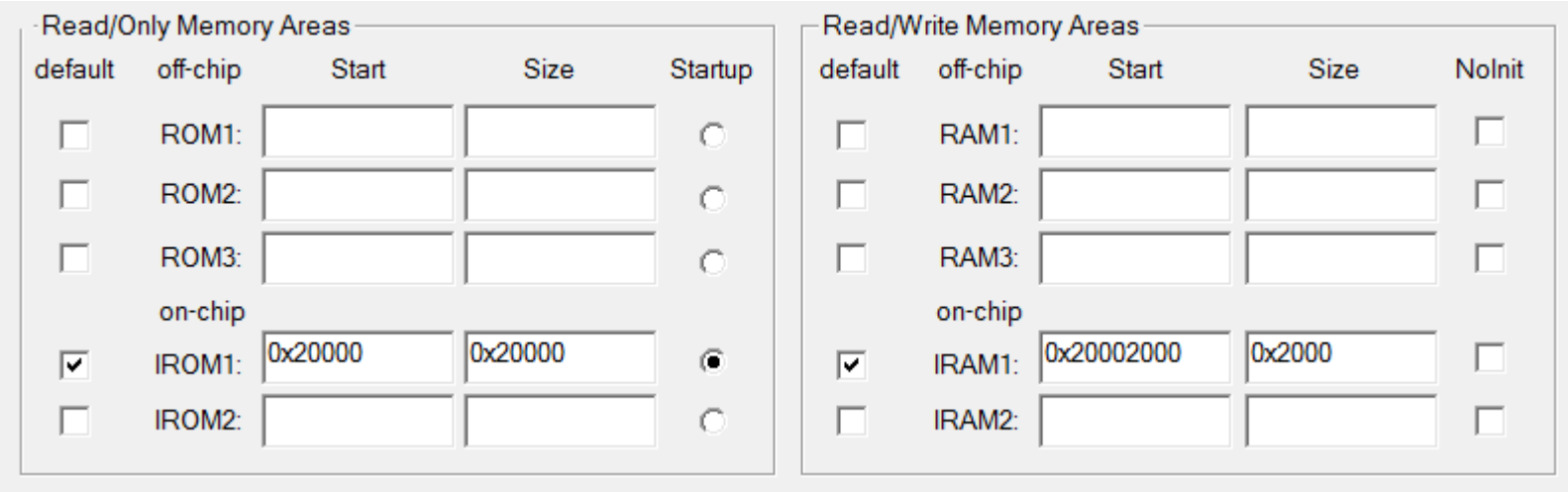
SoftDevice Architecture Benefits

Flexibility	▪ No proprietary application development model or RTOS ▪ The Application developed totally separate from the Stack
Ease of Development	▪ Simple Application programmer model ▪ No link time restrictions / dependencies ▪ Thread-safe API
Code safety	▪ Stack is not re-linked when you compile your Application ▪ Qualification is on a binary going into end-user product ▪ Stack is run-time protected

Faster Development,
Less Bugs

Working with the SoftDevice – Applicaton memory

- Need to change where we program our application
  - Applications must be programmed into Region 1 flash
  - SoftDevice must be programmed in Region 0 flash
  - System vector table will forward interrupts
  - SoftDevice will reserve RAM



SoftDevice Specification – Applicaton memory

Flash	S110 Enabled	S110 Disabled
Amount	128 kB	128 kB
CODE_R1_BASE	0x0002 0000	0x0002 0000

RAM	S110 Enabled	S110 Disabled
Amount	8 kB	0 kB
RAM_R1_BASE	0x2000 2000	0x2000 0000

Call stack	S110 Enabled	S110 Disabled
Maximum usage	1.5 kB (0x600)	0x00

Heap	S110 Enabled	S110 Disabled
Maximum allocated bytes	0 bytes (0x00)	0x00

Table 2 S110 Memory resource requirements

Working with the SoftDevice – HW interfaces

- Need to change the way we interface to:
  - CMSIS functions
  - Memory-mapped peripheral types for protected peripherals

SoftDevice Disabled (default)		SoftDevice Enabled
`NVIC_EnableIRQ(ADC_IRQn)`	→	`nrf_nvic_EnableIRQ(ADC_IRQn)`

`NRF_PPI->CH0_EEP = &(NRF_TIMER1->EVENTS_COMPARE[0])`
 	
`NRF_PPI->CH0_TEP = &(NRF_GPIOTE->TASKS_OUT[0])`
	→	`nrf_ppi_channel_assign(0, &(NRF_TIMER1->EVENTS_COMPARE[0]), &(NRF_GPIOTE->TASKS_OUT[0]))`

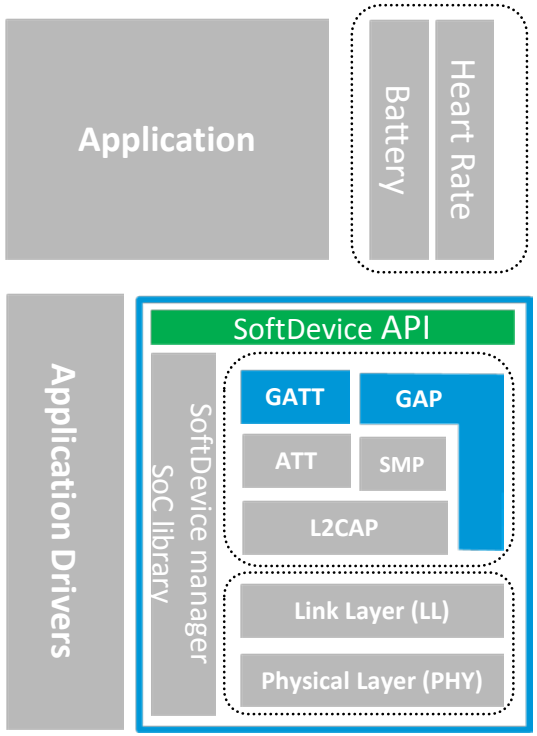
`NRF_TIMER1->TASKS_CLEAR = 1;`	→	`NRF_TIMER1->TASKS_CLEAR = 1;`

SoftDevice Specification – HW interfaces

Peripheral protection and usage by SoftDevice				
ID	**Base address**	**Instance**	**Access (S110 enabled)**	**Access (S110 disabled)**
0	0x40000000	POWER	Restricted	Open
0	0x40000000	CLOCK	Restricted	Open
1	0x40001000	RADIO	Blocked	Open
2	0x40002000	UART0	Open	Open
3	0x40003000	SPIM0 / 2W0	Open	Open
4	0x40004000	SPIM1 / 2W1	Open	Open
		…		
6	0x40006000	Port 0 GPIOTE	Open	Open
7	0x40007000	ADC	Open	Open
8	0x40008000	TIMER0	Blocked	Open
9	0x40009000	TIMER1	Open	Open

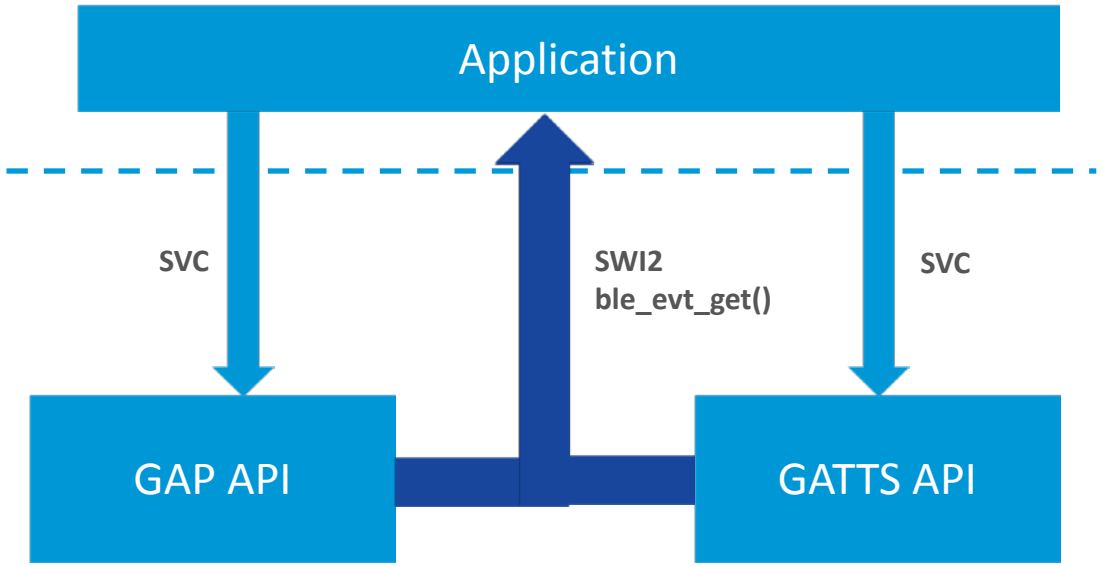
DEMO

Bluetooth® low energy API



Bluetooth® low energy API

- Generic Access Profile (GAP)
- Generic Attribute Profile Server (GATTS)
- API calls as **S**uper**V**isor **C**alls
  - Switches Core to SV priority
  - Each SV Call numbered
- Events as **S**oft**W**are **I**nterrupts
  - Always through SWI2
  - Interrupt priority: Application Low
  - `ble_evt_get()`
    - For all BLE events
    - From ISR or main context



Bluetooth® low energy API

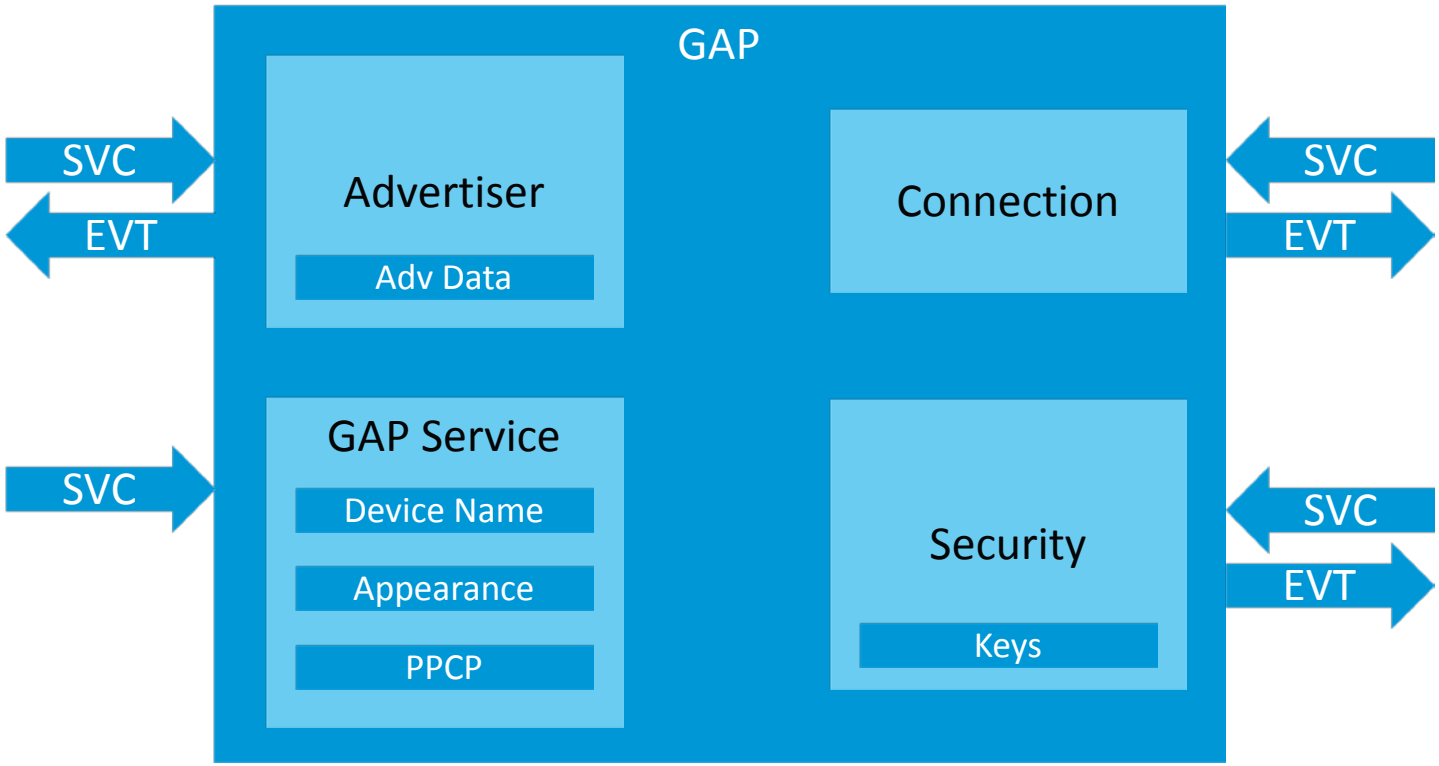
```
main()
{
    // Enable BLE event interrupt
    nrf_nvic_EnableIRQ(SWI2_IRQn);

    ble_gap_adv_start(adv_params);
}

void SWI2_IRQHandler(void)
{
    // Pull event from stack
    ble_evt_get((uint8_t *)&evt, &evt_len);
    ble_evt_dispatch (&evt, evt_len);
}
```

Events

SVC

Events

BLE API: GAP


GAP
SVC
EVT
Advertiser
Adv Data
Connection
SVC
EVT
SVC
GAP Service
Device Name
Appearance
PPCP
Security
Keys
SVC
EVT

BLE API: GAP SVC

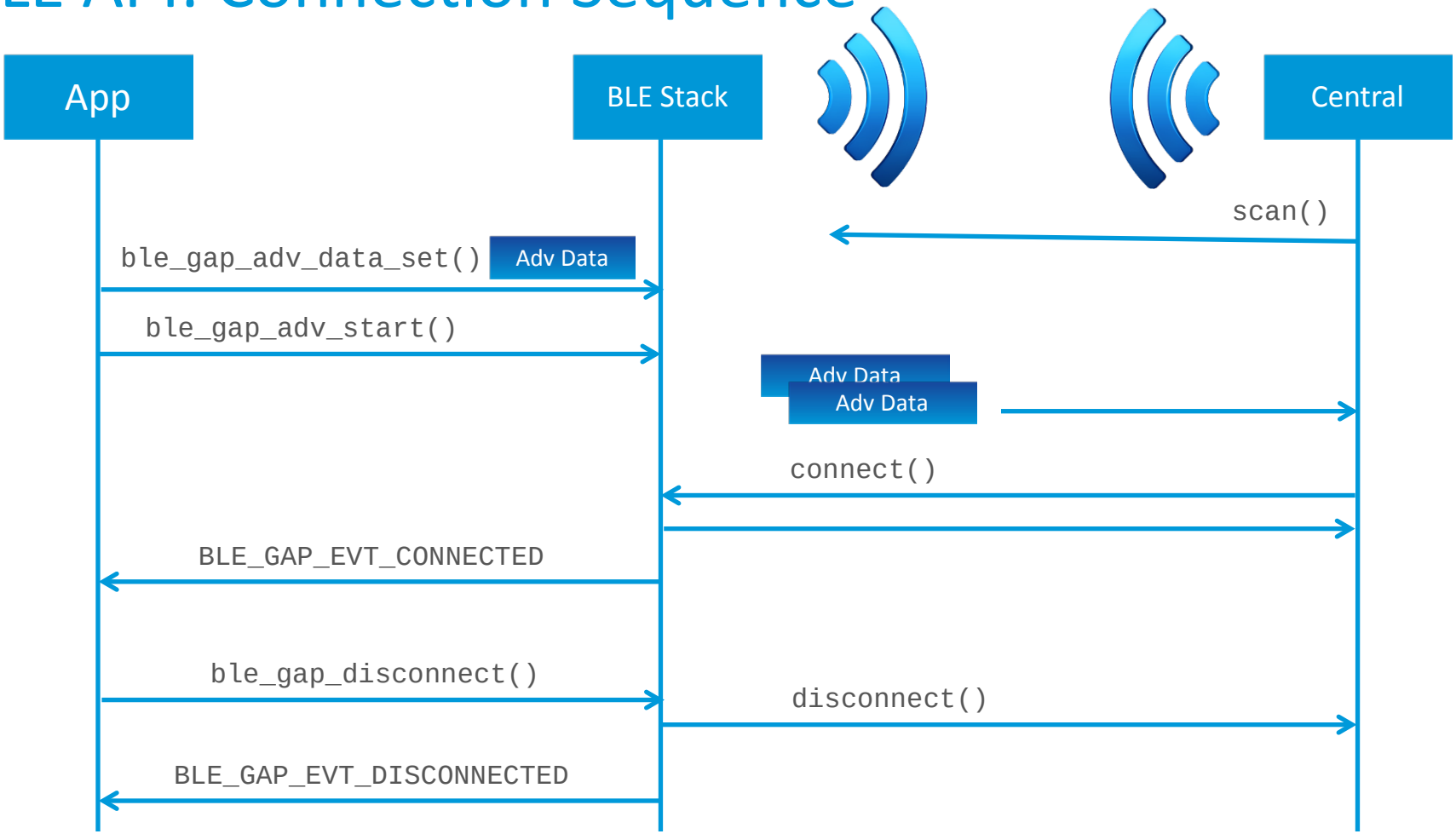
- GAP Service (built in to the stack)
  - **`ble_gap_device_name_set(security, name)`**
    - Sets the name that we give our device
  - **`ble_gap_appearance_set(appearance)`**
    - Describes what our device does to central peers
  - **`ble_gap_ppcp_set(ppcp)`**
    - Defines the connection parameters

- GAP Advertisement
  - **`ble_gap_adv_data_set(adv_data, ad_len, sr_data, sr_len)`**
    - Sets the advertisement data that central peers will receive
  - **`ble_gap_adv_start(adv_params)`**
    - Starts sending advertisement packets over the air

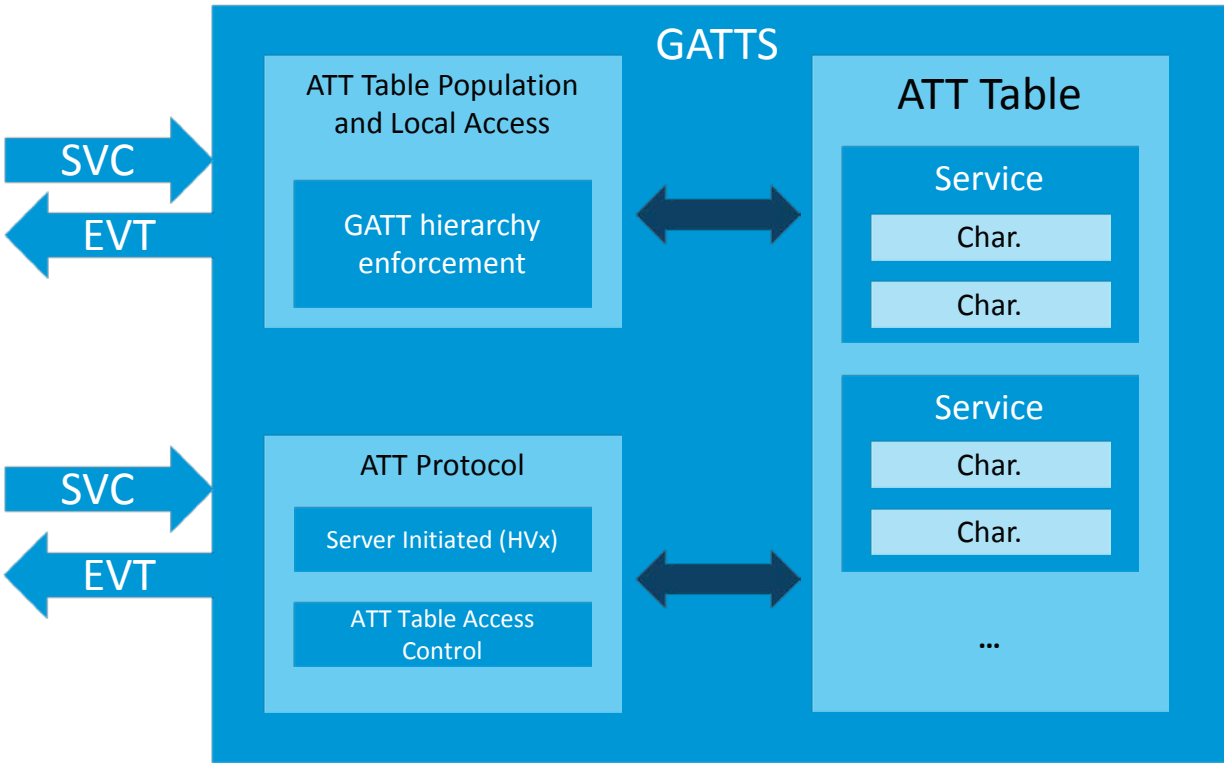
BLE API: GAP Events

- **`BLE_GAP_EVT_CONNECTED {conn_handle, peer_addr, conn_params}`**
  - A central peer has established a physical connection
- **`BLE_GAP_EVT_DISCONNECTED {conn_handle, reason}`**
  - The connection has been terminated, locally or remotely
- **`BLE_GAP_EVT_CONN_PARAM_UPDATE {conn_params}`**
  - A connection parameter update procedure has completed
- **`BLE_GAP_EVT_TIMEOUT {source}`**
  - A procedure has timed out (advertisement, security, …)

BLE API: Connection Sequence



BLE API: GATTS



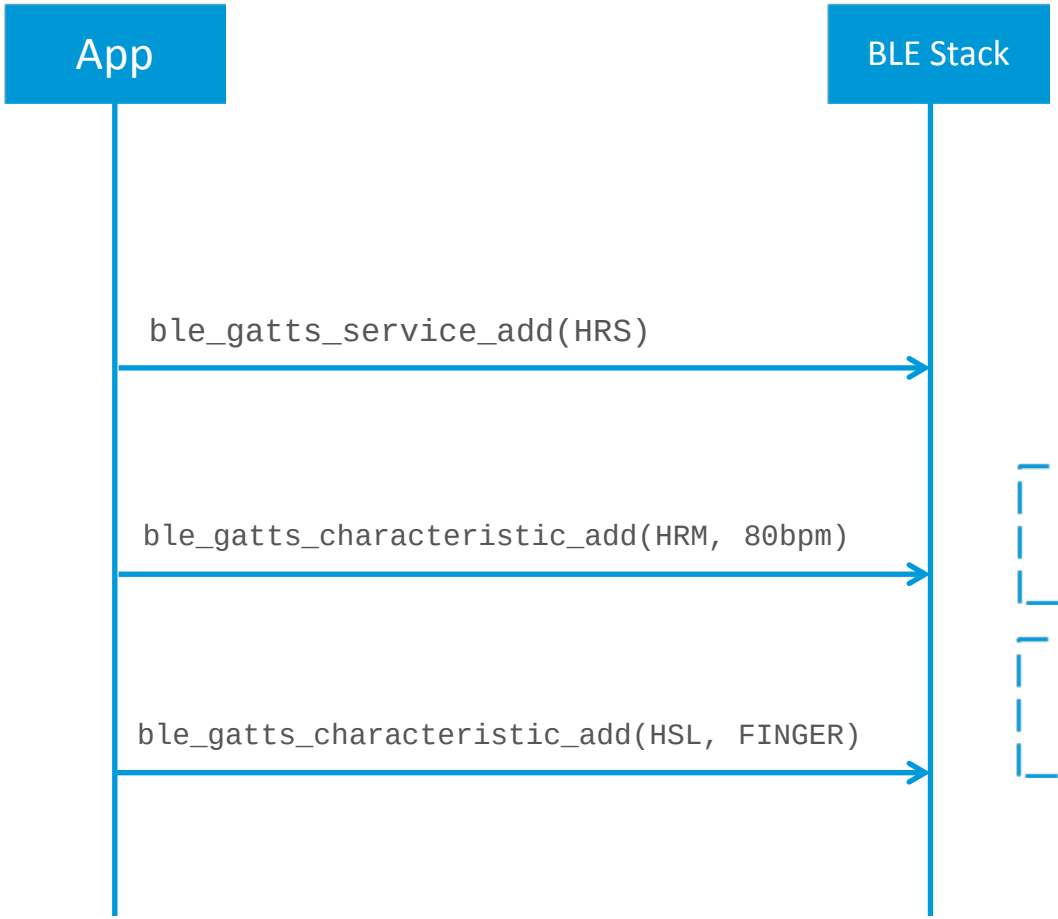
BLE API: GATTS SVC

- ATT Table Population
  - **`ble_gatts_service_add(type, UUID, out_handle)`**
    - Adds an empty Service to the ATT Table
  - **`ble_gatts_characteristic_add(svc_handle, md, value, out_handles)`**
    - Adds a Characteristic to the referenced service
- ATT Table Local Access
  - **`ble_gatts_value_set(handle, offset, len, value)`**
    - Sets the value of any particular attribute
  - **`ble_gatts_value_get(handle, offset, len, value)`**
    - Gets the value of any particular attribute
- Server Initiated
  - **`ble_gatts_hvx(conn_handle, params)`**
    - Sends an ATT Notification or Indication

BLE API: GATTS Events

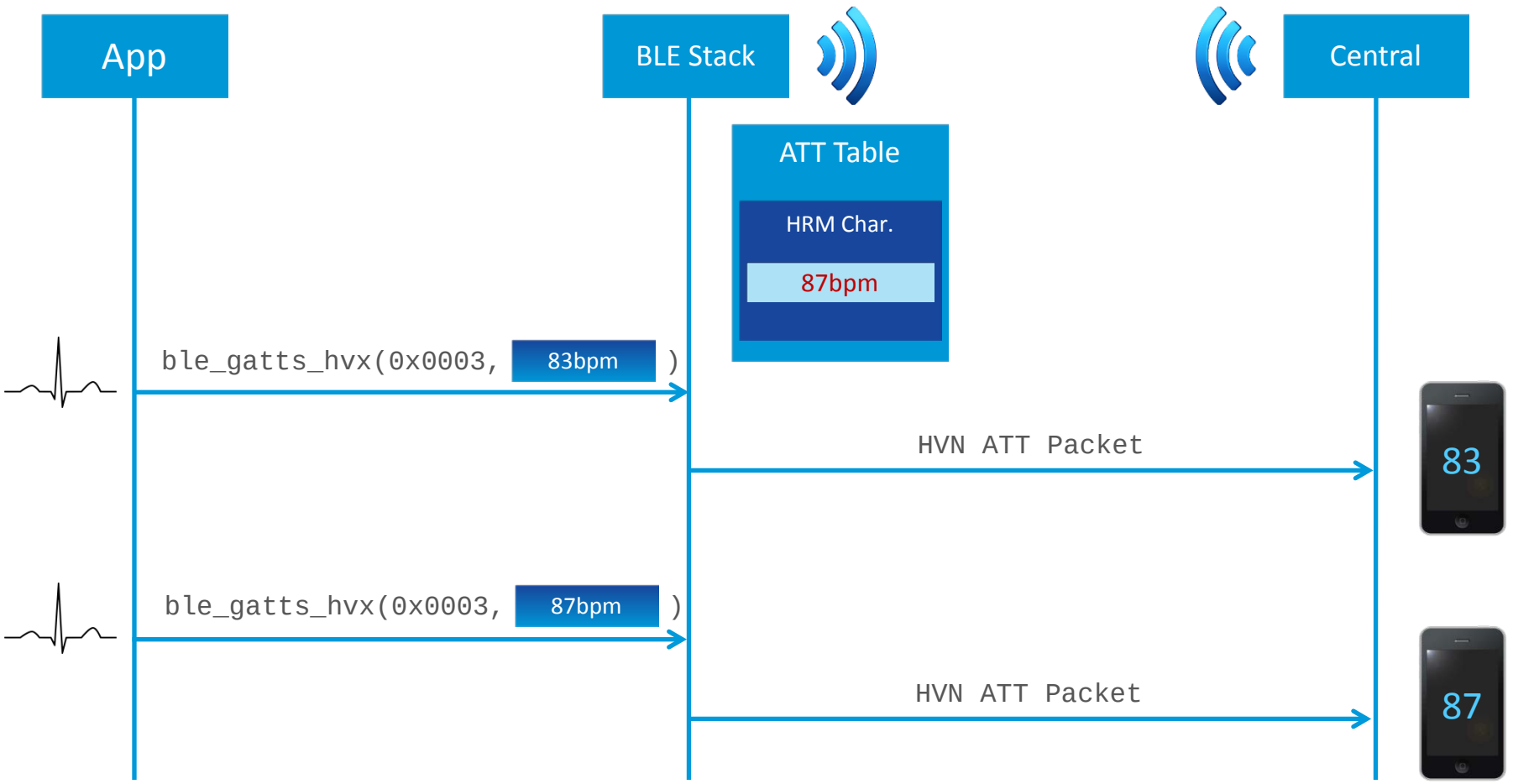
- **`BLE_GATTS_EVT_WRITE {conn_handle, handle, data}`**
  - An incoming client ATT Write operation has completed
- **`BLE_GATTS_EVT_HVC {conn_handle, handle}`**
  - A Handle Value Confirmation has been received from the peer

BLE API: Service population



Handle	UUID	Value
0x0001	*SERVICE*	*HRS*
0x0002	*CHAR*	*HRM*
0x0003	*HRM*	80bpm
0x0004	*CHAR*	*HSL*
0x0005	*HSL*	*FINGER*

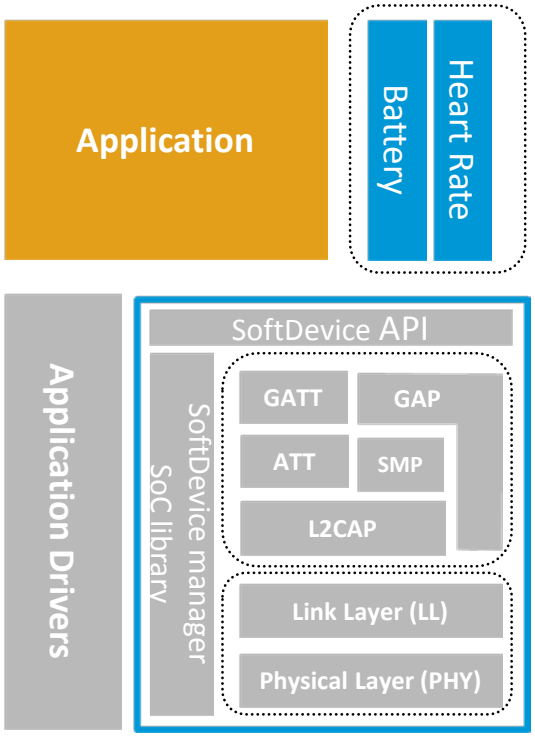
BLE API: Handle Value Notification



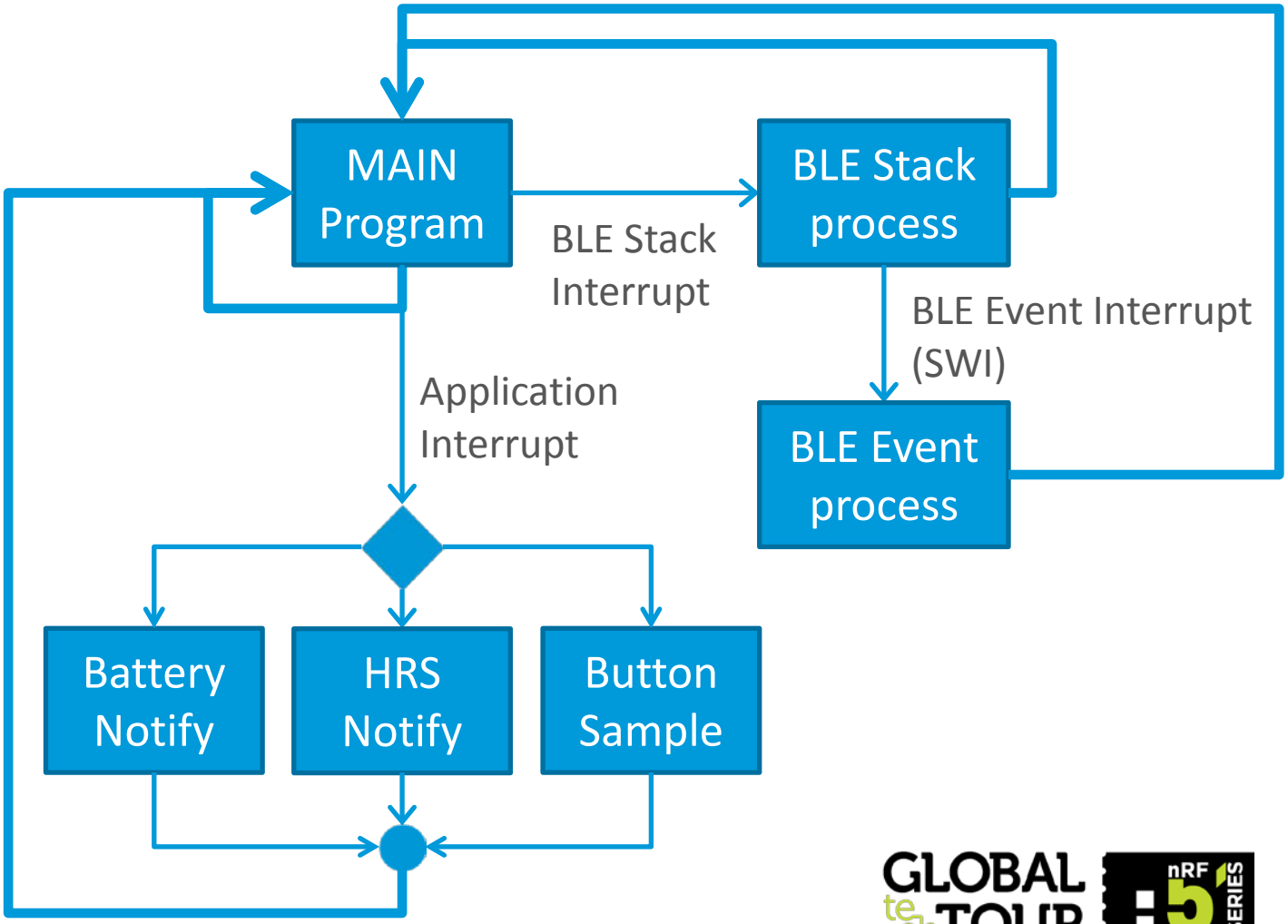
Break

[15 min]

Application



Application block diagram



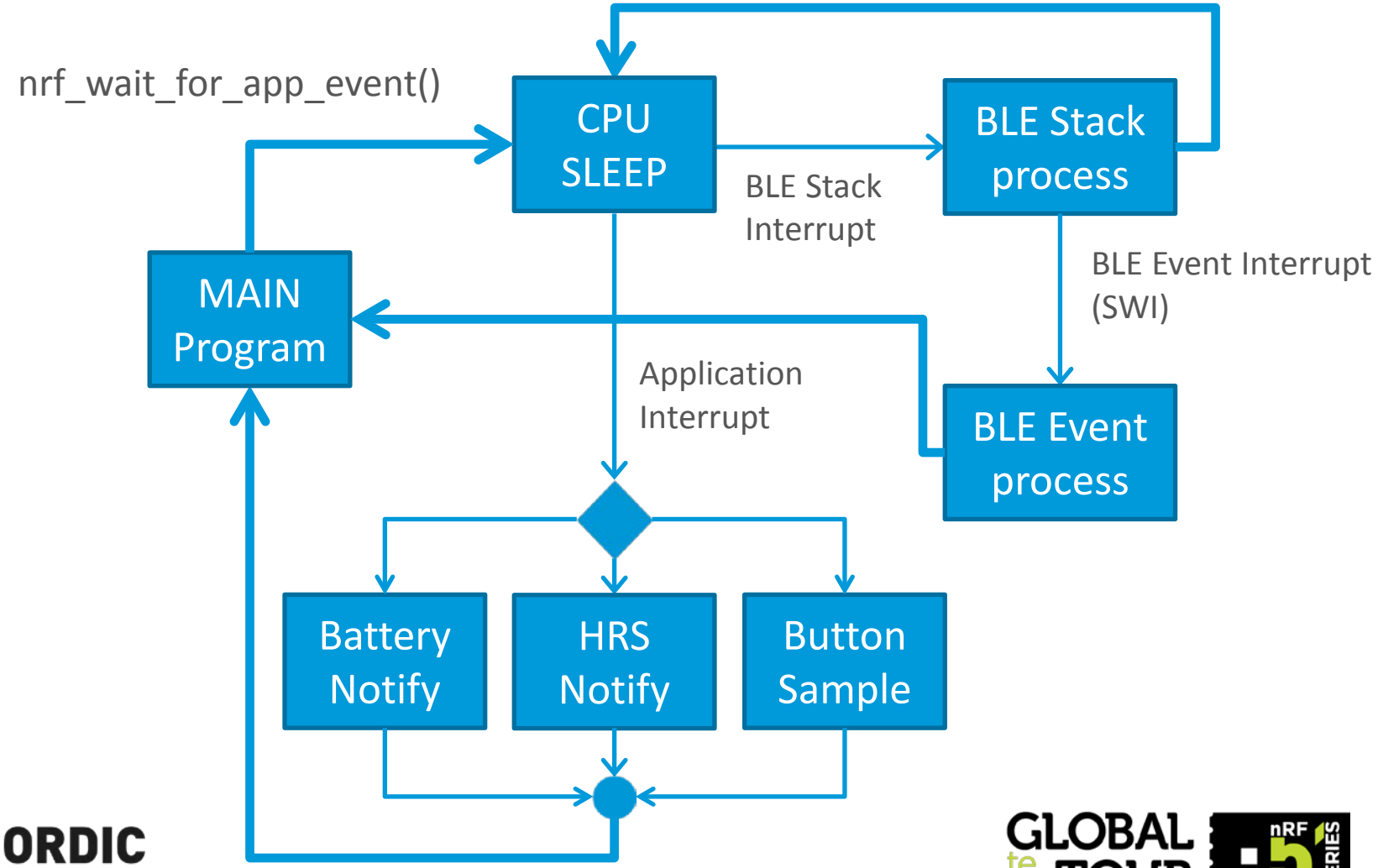
DEMO

: Measuring current

Adding Power Management

```
// Application Power Management –
// Switch to a low power mode,
// wake on Application events only
err_code = nrf_wait_for_app_event();

// Switch to OFF, reset on wakeup
err_code = nrf_power_system_off();
```

Application block diagram



DEMO

: Measuring current

Simple BLE sensor application walk through - QUESTIONS

NORDIC
SEMICONDUCTOR