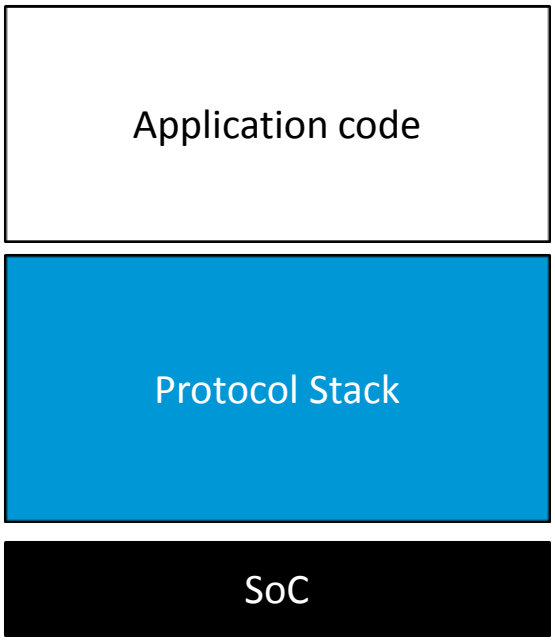
SoftDevices

nRF52 Global Tech Tour

The ideal SoC software architecture

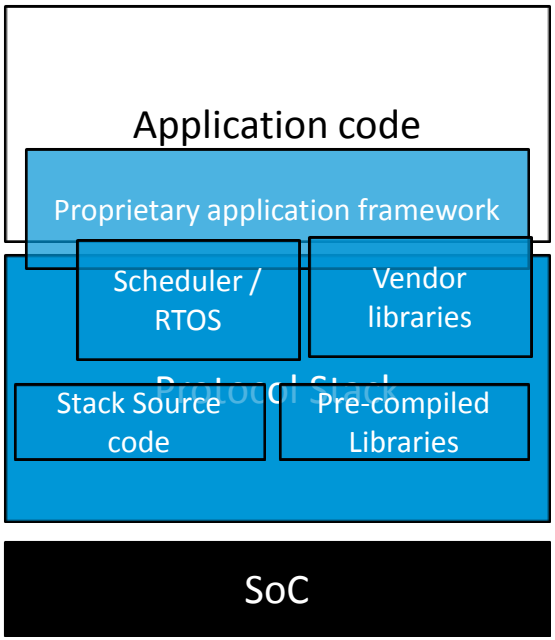
And how everyone presents it….



- "Developing an application on our product is easy"
- "Our software is clean and modular and easy to integrate"
- "We test our stack so you don't have to"
- "Your application will be re-usable and portable"
- And so on …

The harsh reality

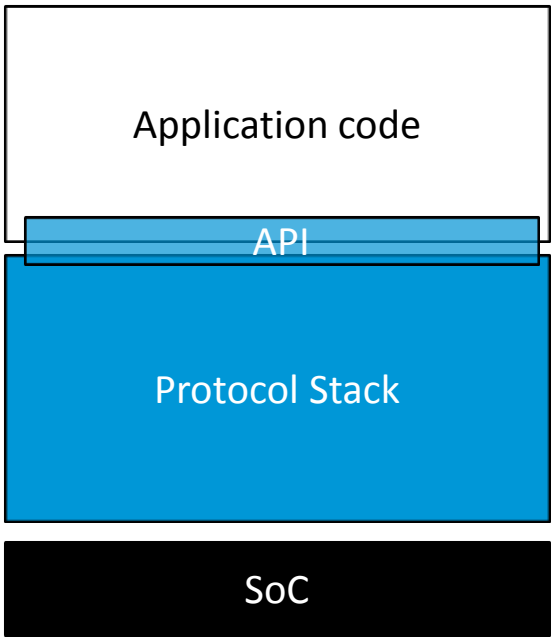
The typical picture for developers…



- It does not look like this
- There is no clean separation of code
  - One big compile
  - Link time dependencies
- Often have Scheduler / RTOS dependency
  - Not portable
- The protocol is re-linked every build
  - Not the same as what was verified

nRF52 SoftDevices are different

A cleaner solution…



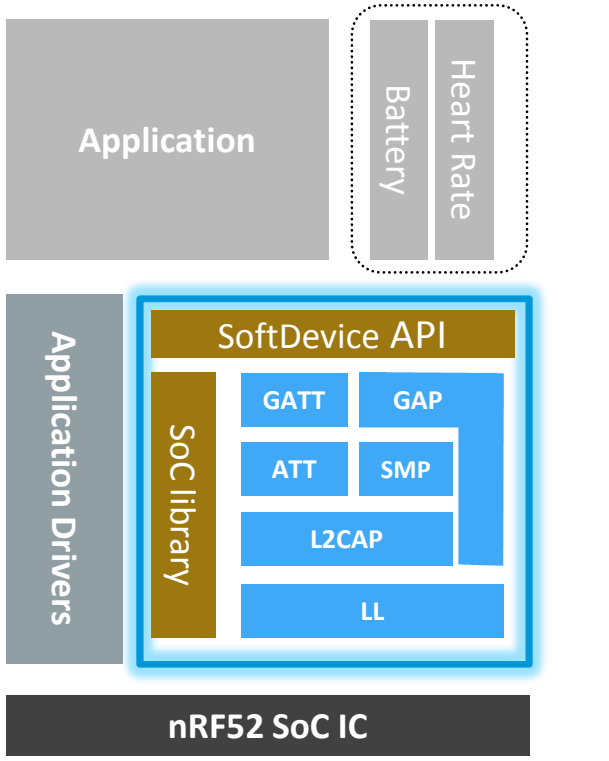
- It does look like this, we just add an API
- Separate binary application and protocol stack
  - No link time dependencies
- No proprietary application framework
  - No scheduler or RTOS dependencies
  - Choose your compiler and language
- Protocol verification is run on the same binary image you use in your product

SoftDevice basics

- The 3 important concepts of the SoftDevice:
  1. Features
  2. Software separation
  3. API implementation

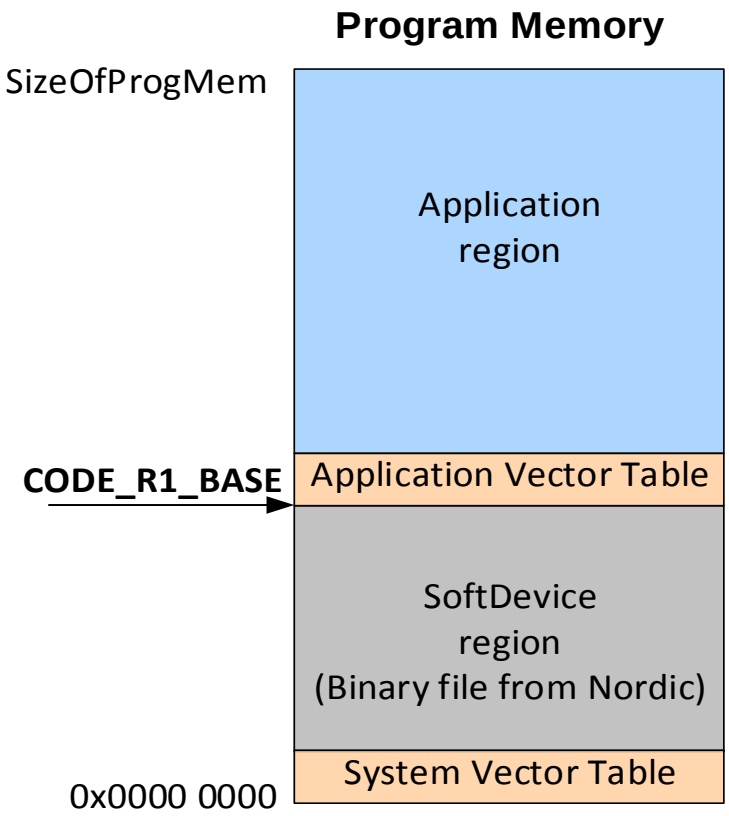
1. SoftDevice features

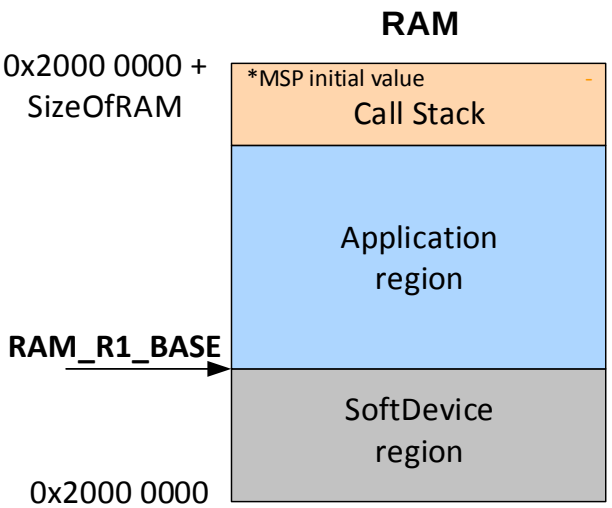
- Wireless protocol stack:
  - Bluetooth low energy
  - ANT
  - BLE + ANT combo
- Radio scheduler API:
  - Combine BLE with a proprietary radio protocol
- SoC library:
  - Safe access to peripherals: Flash, PPI, timers, and so on
  - Power management:
    - `sd_app_evt_wait()`



Example: S132 *Bluetooth* low energy protocol stack

2. Software separation





3. SoftDevice API implementation

application.c

```
// Use the SoftDevice API functions as normal C functions.
#include "ble_gap.h"
sd_ble_gap_connect(…);
```

application.i

```
// The functions are definitions that expands to SVCs.
uint32_t __svc(SD_BLE_GAP_CONNECT) sd_ble_gap_connect(…);
```

softdevice.c

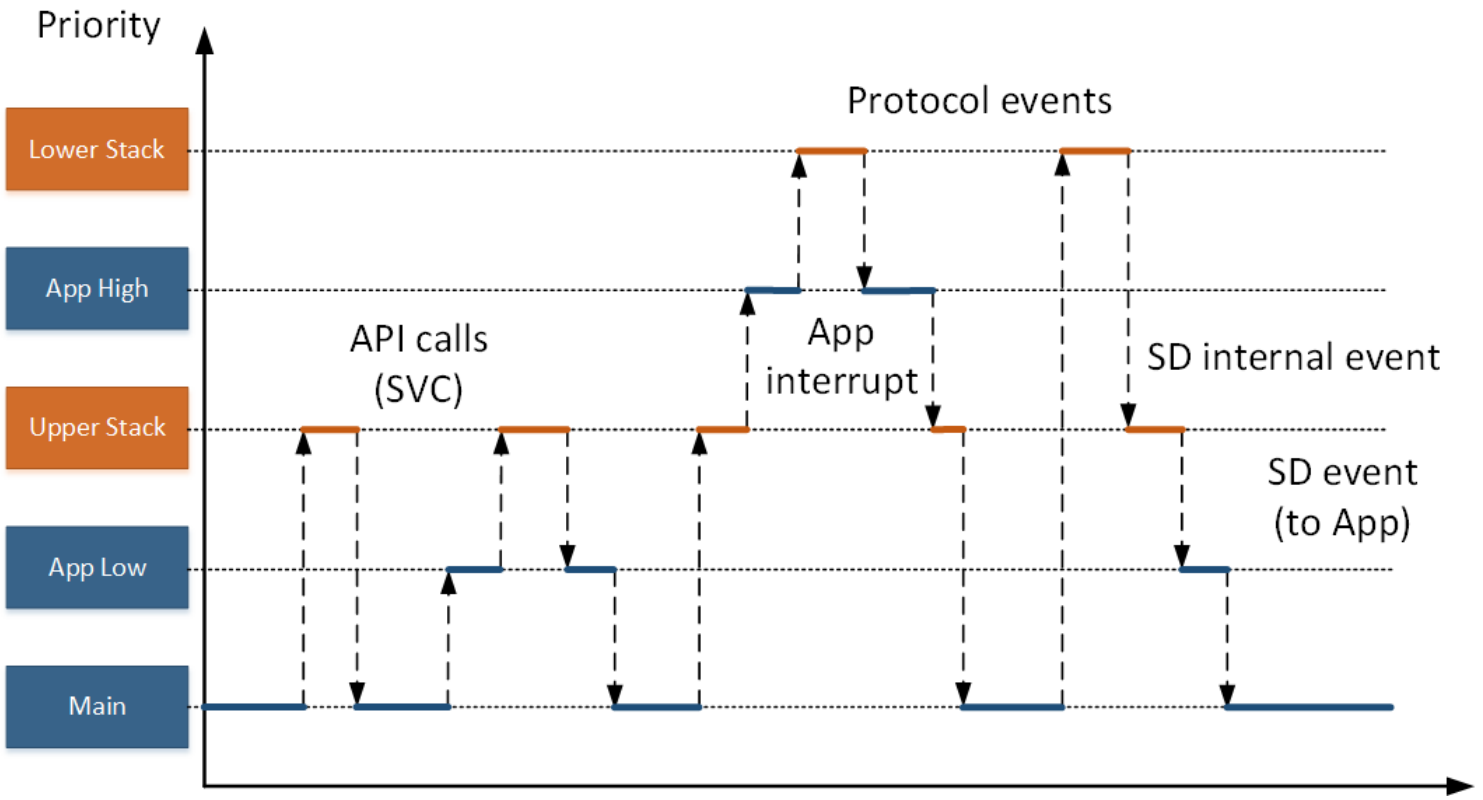
```
// SoftDevice functions are called from the
// SVC_Handler interrupt routine.
void SVC_Handler(int svc_num, int *svc_args)
{
    switch(svc_number)
    {
…
        case SD_BLE_GAP_CONNECT:
            sd_ble_gap_connect();
```

3. SoftDevice API implementation

- API calls in the Application cause SVC interrupts in the SoftDevice where the function is executed.
- The SoftDevice events are signaled to the Application by triggering a software interrupt (SWI).

```
// SoftDevice Event Notification
void SD_EVT_IRQHandler(void)
{
    // Poll for SoftDevice events.
}
```

3. SoftDevice API – interrupt context



SoftDevices and protocol stacks

nRF51	nRF52	Protocol
S110 (v8.0)	-	BLE peripheral
S120 (v2.0)	-	BLE 8-link central + non-concurrent peripheral
S210	S212	ANT
S310	-	BLE peripheral + ANT
S130	S132	BLE concurrent central and peripheral
-	S332	BLE concurrent central and peripheral + ANT

- Note: Easy SW migration from nRF51 to nRF52
  - compatible APIs and application code

Which IC and SoftDevice to use?

- Are you using: nRF51 with S110/S120?
  - SoftDevices will be maintained, continued use is OK.
- Are you using: nRF51 and need additional **Bluetooth** features?
  - Switch to (or continue with) S130 to access new BLE features.
  - Similar API, easy transistion.
- Are you using: nRF51 and need additional **Hardware** features?
  - Start using nRF52.
  - S130 and S132 are API compatible:
    - S130 could be an intermediate step from S110/S120 to S132.
    - Minimal effort moving from S130 to S132.

S132

- S132 will be ready for nRF52832 production (February 2016)
- We will release both S132 and a new major revision of the S130
  - Same feature set
  - S132 is for nRF52
  - S130 is for nRF51
- We are committed to supporting nRF51 for existing applications

- nRF52 enables new applications: nRF52 + S132 vs nRF51 + S130
  - Up to double battery life
  - Up to 5 times faster SoftDevice interrupts (1/5 interrupt latency)
  - Up to 4x more processing power for your applications

S132 First Release

- A familiar API with all of the features you expect
  - Timeslot API for multi-protocol
  - Memory efficient and secure DFU
  - Does everything our S110, S120 and S130 did on nRF51
- Flexible memory allocation
  - Configure memory available to your application
- Choose how many Central and Peripheral links you need for your application
- Configure the Bandwidth you want for each link
- + More

S132 What's Next

- Bluetooth features for IoT (IPv6 over BLE)
  - L2CAP Connection oriented channels
  - Long MTU size
  - Data packet length extension (up to 255 bytes transmitted in 1 packet)
    - Increases BLE maximum data throughput capability
- Multiple peripheral connections (2 or more)
  - More advanced network topologies
- LE secure connections
- Bluetooth support
  - New specification features are coming

Since our last Global Tech Tour...

- nRF52 is our 3rd generation of BLE chips
- We have 4 years experience developing and supporting SoftDevices
  - 10 production releases adding features
  - Over 40 Bluetooth qualifications (stacks, profiles and chips)
- We lead in quality
  - More than 11000 tests run on every release
  - Stress testing on chips over all supported temperature, voltage and clock conditions
  - Only 3 bug-fix releases required in 4 years
- We lead in support
  - Almost 10,000 questions answered on Developer Zone since 2012 (devzone.nordicsemi.com)
- We lead the market
  - More than 100 million BLE devices using our SoftDevices