The micro:bit runtime Inside and Out

James Devine, Joe Finney
Lancaster University, UK

j.devine@lancaster.ac.uk, j.finney@lancaster.ac.uk

- 25 LED matrix screen
- Light sensor
- User definable buttons

- 17 Digital input/output
- 6 Analog input
- 3 PWM output
- 3 Touch sensitive
- I2C, SPI, UART

BBC micro:bit

- 16MHz ARM Cortex M0
- 16KB RAM, 256K FLASH

- USB Storage/Serial/Debug

- 3 axis accelerometer
- 3 axis magnetometer
- Temperature sensor
- Bluetooth Low Energy

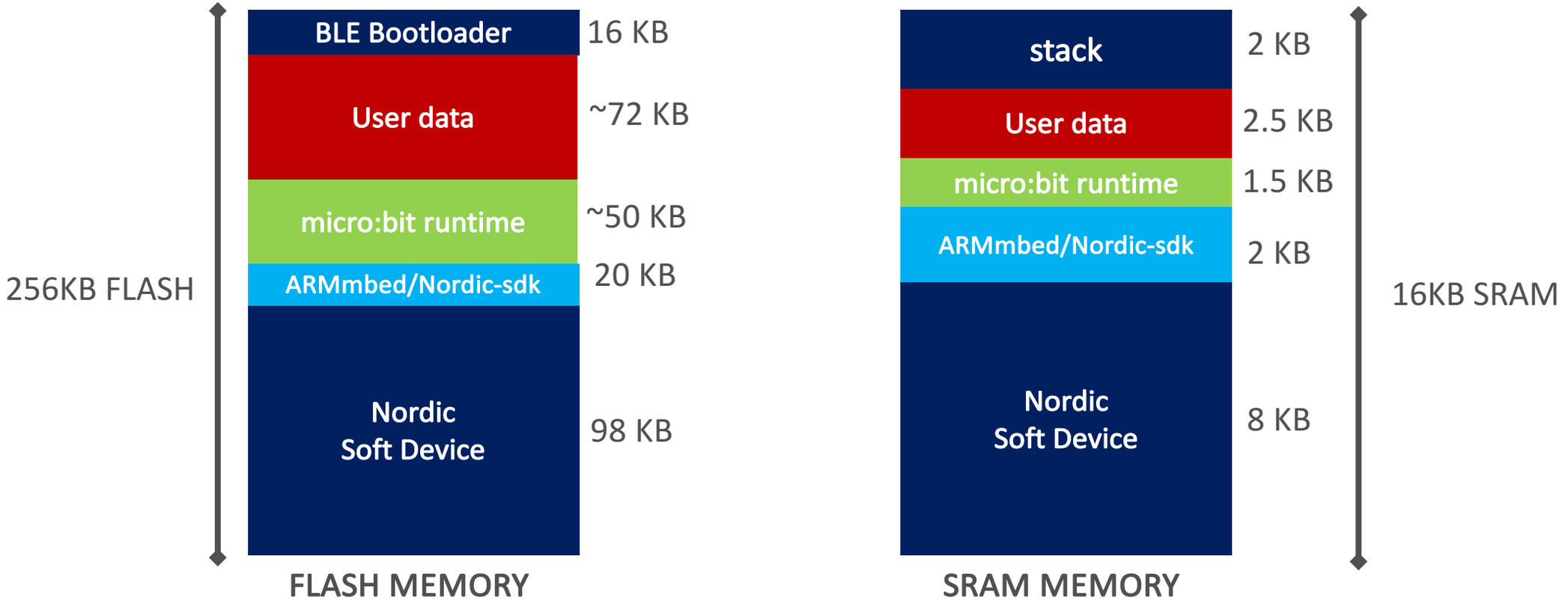
BBC micro:bit

Introducing the micro:bit runtime

- Provides a Device Abstraction Layer for the micro:bit
- Open source C/C++ component based API
- High level language features (concurrency, eventing models and memory safety)
- Native C/C++ friendliness
- Focused on RAM and power efficiency

Memory Footprint

- micro:bit has 16Mhz Nordic nrf51822 CPU (32 bit Cortex M0)

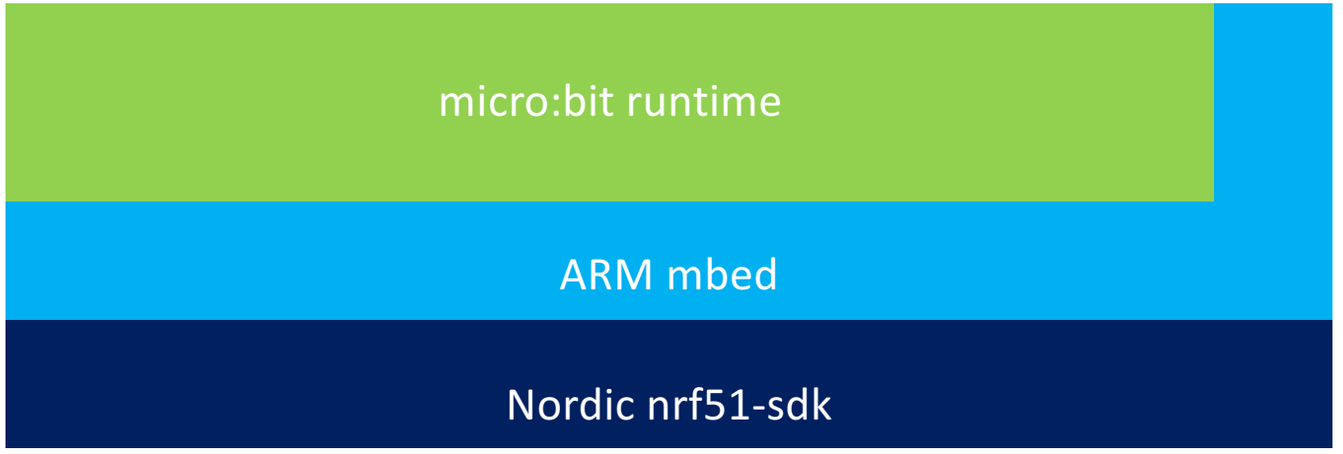


micro:bit runtime architecture

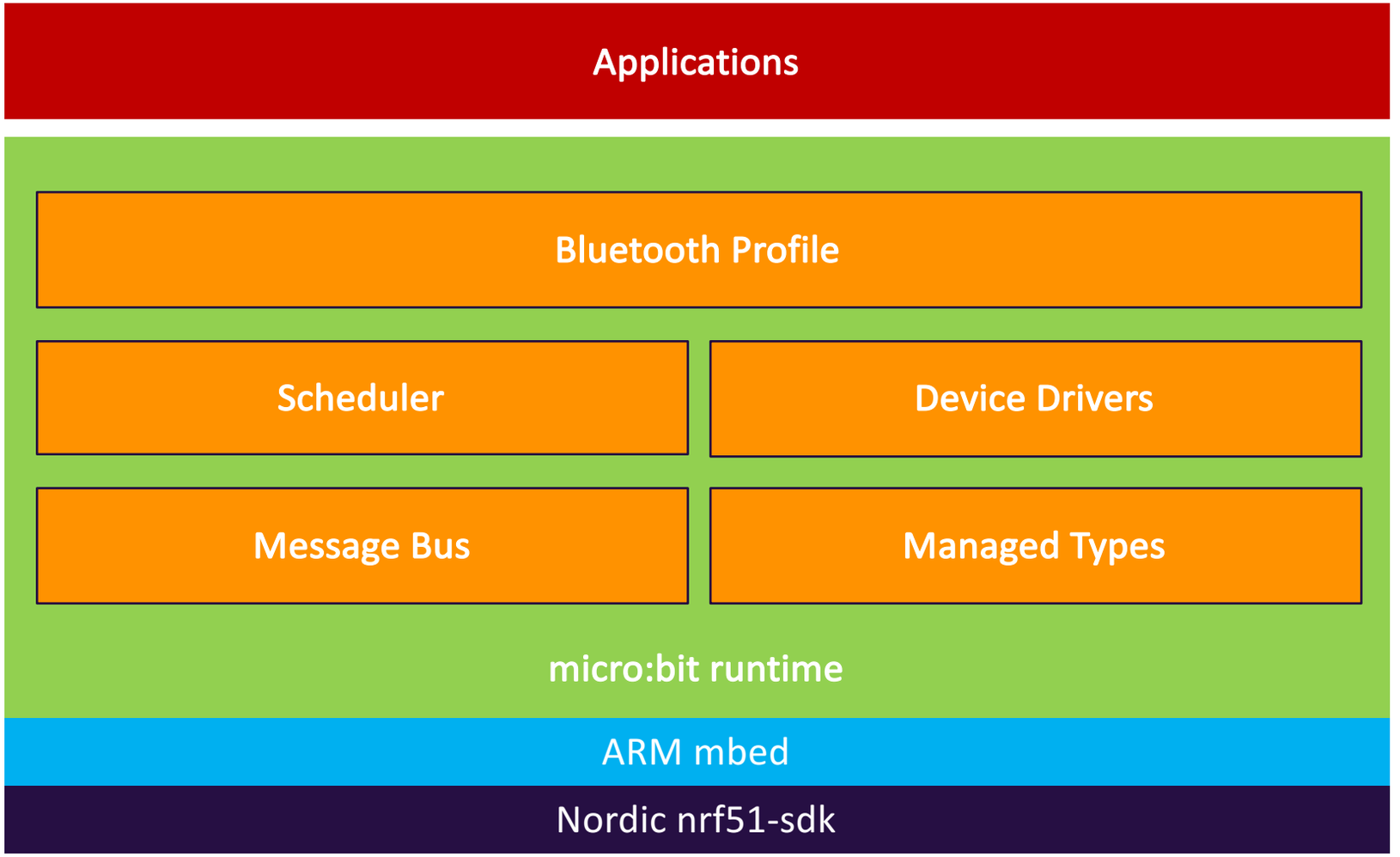
Development environments

Block Editor	Touch Develop	PXT	Java Script	C / C++	Python
Microsoft	Microsoft	Microsoft	Code Kingdoms	ARM mbed	PSF +friends

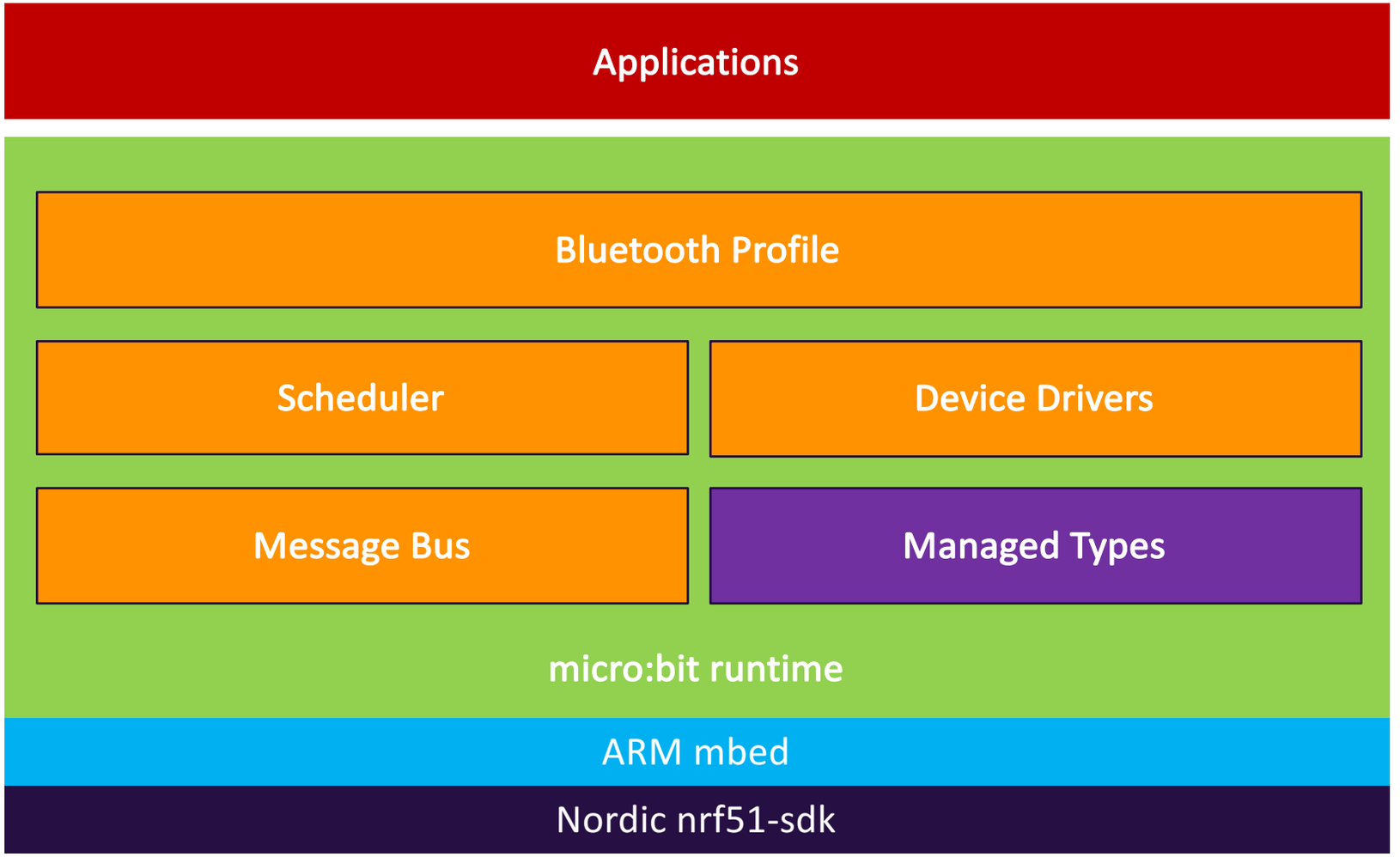
The runtime and supporting libraries



micro:bit runtime architecture



micro:bit runtime architecture



Managed Types

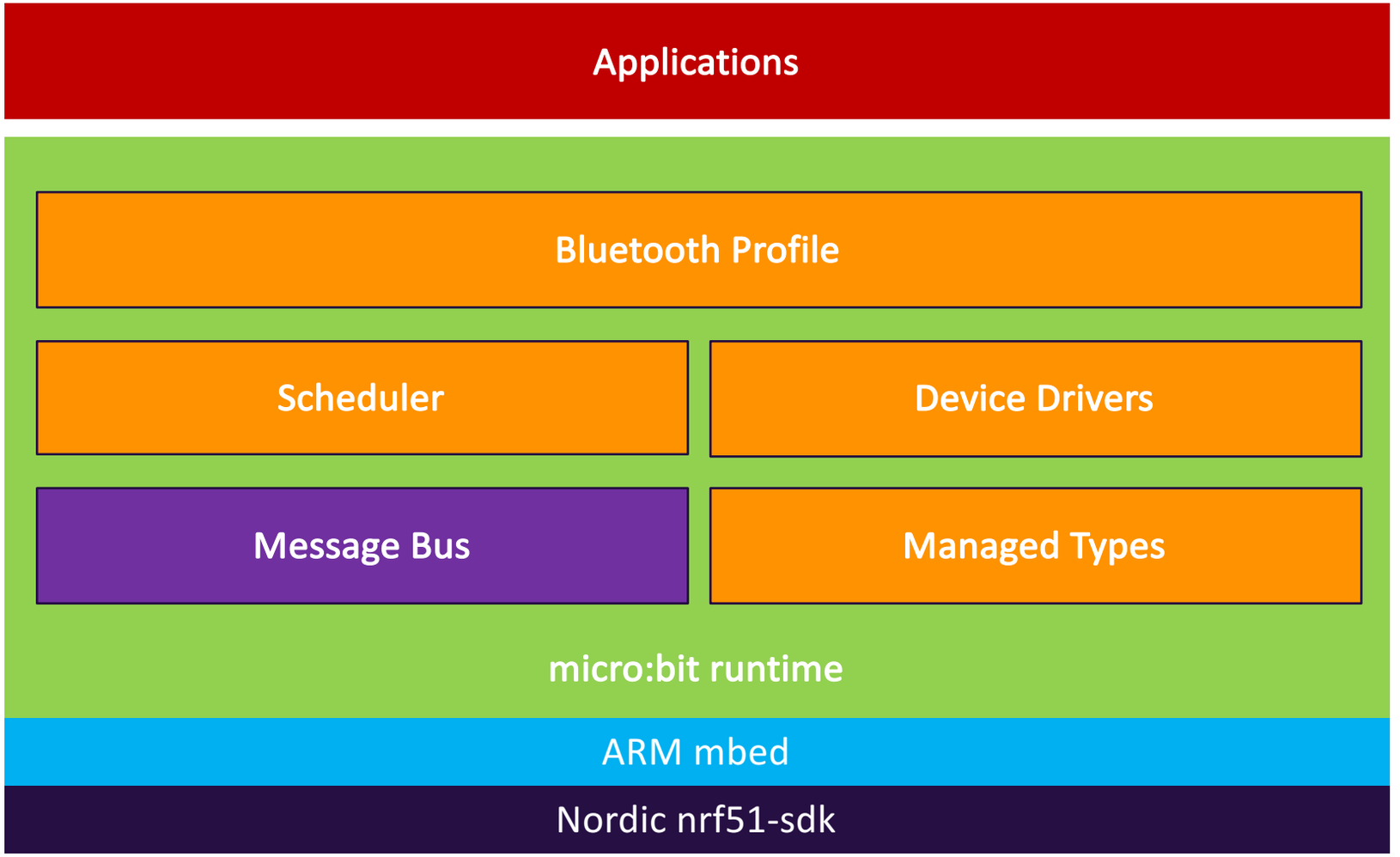
- Handles the messiness of memory allocation!

- Commonly used data types (strings, images, packets) all have their own data type
- Uses **reference counting** to track when data is used.

```
ManagedString s = “hello”;

doSomething(s);
```

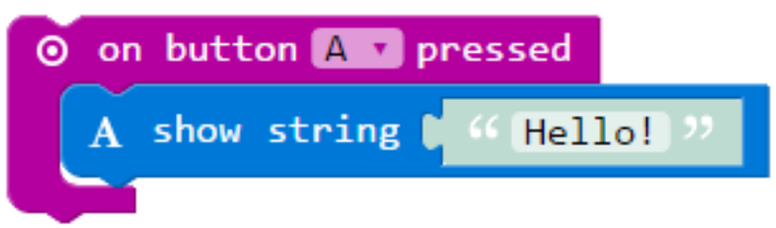
```
void
doSomething(ManagedString text)
{
    …
}
```

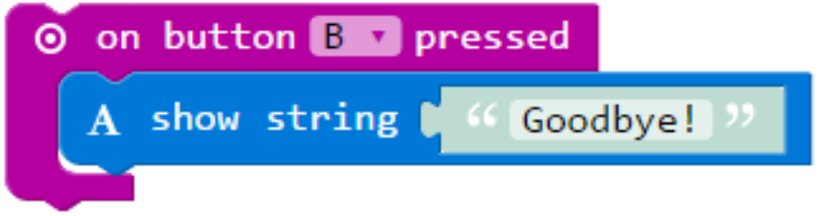
micro:bit runtime architecture



Eventing and the Message Bus

- Events are conceptually easy to understand
- A common mechanism used in many languages





Eventing and the Message Bus

- Events are lightweight and extensible
- Events are pass by value
- Contains 3 numbers:
  - A **source** id
  - A **value**
  - A **timestamp**

```
MicroBitEvent e(MICROBIT_ID_GESTURE, MICROBIT_ACCELEROMETER_EVT_SHAKE);
```

```
#define MICROBIT_ID_GESTURE                     27
#define MICROBIT_ACCELEROMETER_EVT_SHAKE        11
```

Eventing and the Message Bus

- The **MessageBus** then delivers events to any code that registers an interest.

```
void onShake(MicroBitEvent e)
{
    // do something cool here!
}

int main()
{
    uBit.messageBus.listen(MICROBIT_ID_GESTURE, MICROBIT_ACCELEROMETER_EVT_SHAKE, onShake);
}
```

Eventing and the Message Bus

- The **MessageBus** then delivers events to any code that registers an interest.

```
void onGesture(MicroBitEvent e)
{
    if (e.value == MICROBIT_ACCELEROMETER_EVT_SHAKE) …
}

int main()
{
    uBit.messageBus.listen(MICROBIT_ID_GESTURE, MICROBIT_EVT_ANY, onGesture);
}
```

Eventing and the Message Bus

- The **MessageBus** then delivers events to any code that registers an interest.

```
void onEvent(MicroBitEvent e)
{
    if (e.source == MICROBIT_ID_GESTURE) …
}

int main()
{
    uBit.messageBus.listen(MICROBIT_ID_ANY, MICROBIT_EVT_ANY, onEvent);
}
```

Eventing and Interrupt Context

- Events are used to decouple user code from interrupt context

- Interrupt context adds hidden complexities

```
void onPulse(MicroBitEvent e)
{
    // scroll the pulse width
    uBit.display.scroll(evt.timestamp);
}

int main()
{
    uBit.messageBus.listen(MICROBIT_ID_IO_P0, MICROBIT_PIN_EVT_PULSE_HI, onPulse);
    uBit.io.P0.eventOn(MICROBIT_PIN_EVENT_ON_PULSE);
}
```

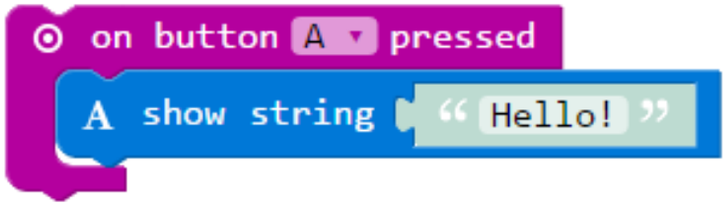
Eventing and Interrupt Context

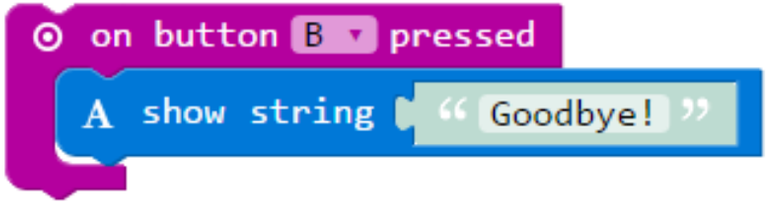
- Events are used to decouple user code from interrupt context
- Interrupt context adds hidden complexities

```
void onPulse(MicroBitEvent e)
{
    // get the pulse width, do something time critical
    int pulse_width = e.timestamp;
}

int main()
{
    uBit.messageBus.listen(MICROBIT_ID_IO_P0, MICROBIT_PIN_EVT_PULSE_HI, onPulse,
    MESSAGE_BUS_LISTENER_IMMEDIATE);
    uBit.io.P0.eventOn(MICROBIT_PIN_EVENT_ON_PULSE);
}
```

Eventing Behaviour

- The receiver defines the behaviour of event invocation
- Three different behaviours: **re-entrant**, **queue if busy**, **drop if busy**



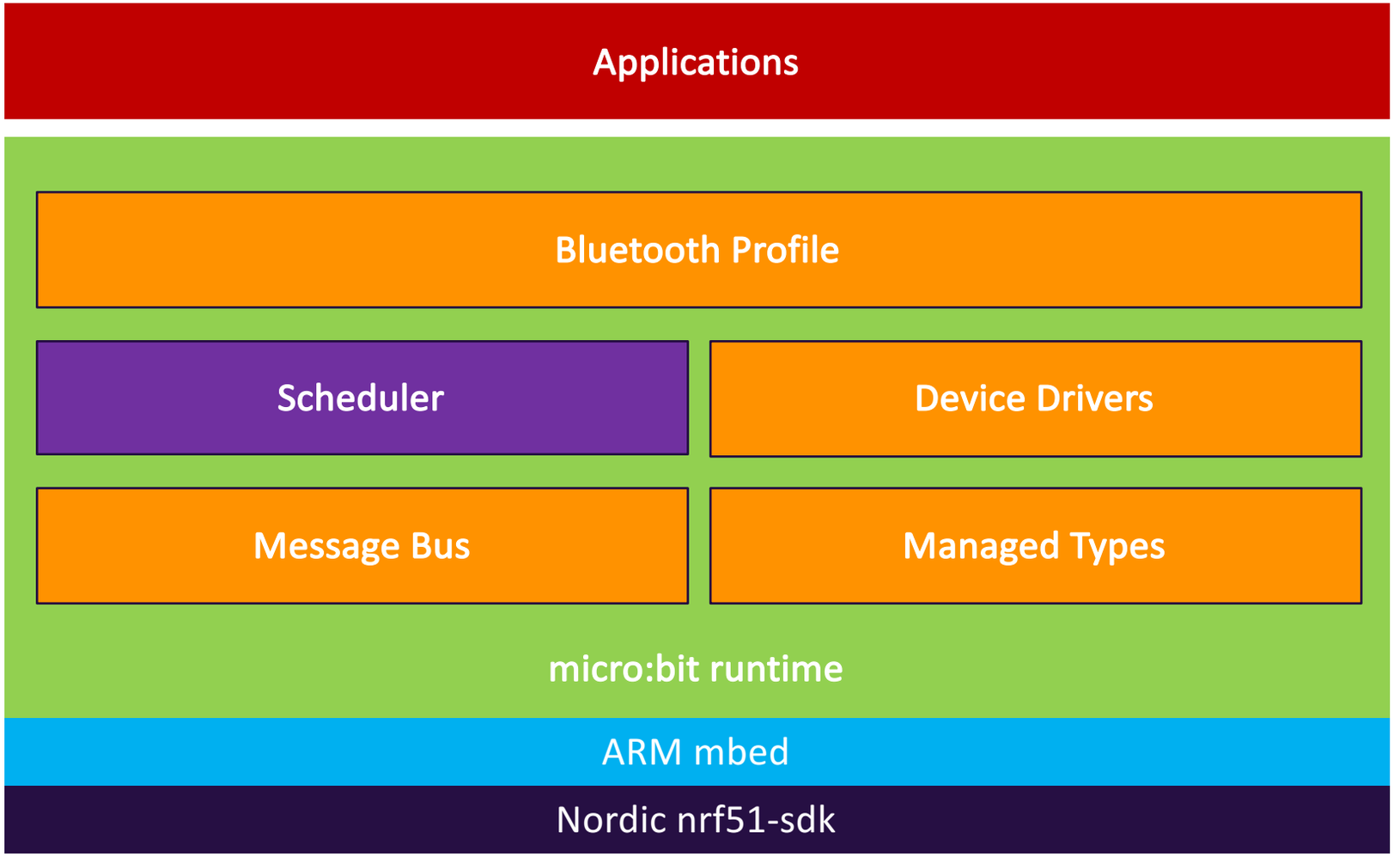


Take this simple example again. What behaviour would you expect?

Eventing Behaviour

```
int button_count = 0;

void onButtonA(MicroBitEvent){
    int count_before = button_count;

    button_count++;

    uBit.sleep(1000);

    uBit.display.scroll(ManagedString(count_before) + " : " + button_count);
}

// Then in your main program...

uBit.messageBus.listen(MICROBIT_ID_BUTTON_A, MICROBIT_BUTTON_EVT_CLICK, onButtonA,
MESSAGE_BUS_LISTENER_REENTRANT);
```

micro:bit runtime architecture



Fiber Scheduler: Providing Concurrent behaviour

- By design, a **non pre-emptive** scheduler to reduce potential race conditions.
- Simplicity is key
- Pre-emptive models add complexity

```
void doSomething()
{
    while(1)
    {
        uBit.display.print('A');
        uBit.sleep(100);
    }
}
```

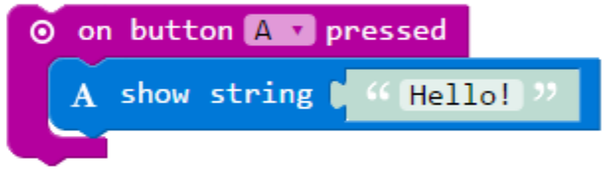
```
void doSomethingElse()
{
    while(1)
    {
        uBit.display.print('B');
        uBit.sleep(100);
    }
}
```

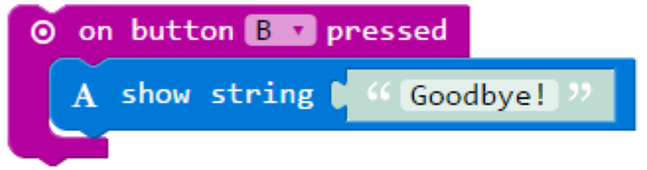
Fiber Scheduler: Providing Concurrent behaviour

- A **RAM optimised** thread scheduler for Cortex processors.
- We adopt a **stack duplication** approach
- Each fiber typically costs ~200 bytes.
- Event handlers (by default) run in their own fiber*
- Functions (e.g. scroll text) can block the calling fiber until the task completes

Fiber Scheduler: Providing Concurrent behaviour

```
void onButtonA(MicroBitEvent)
{
    uBit.display.scroll("Hello!");
}

void onButtonB(MicroBitEvent)
{
    uBit.display.scroll("Goodbye!");
}

// Then in your main program...

uBit.messageBus.listen(MICROBIT_ID_BUTTON_A, MICROBIT_BUTTON_EVT_CLICK, onButtonA);
uBit.messageBus.listen(MICROBIT_ID_BUTTON_B, MICROBIT_BUTTON_EVT_CLICK, onButtonB);
```





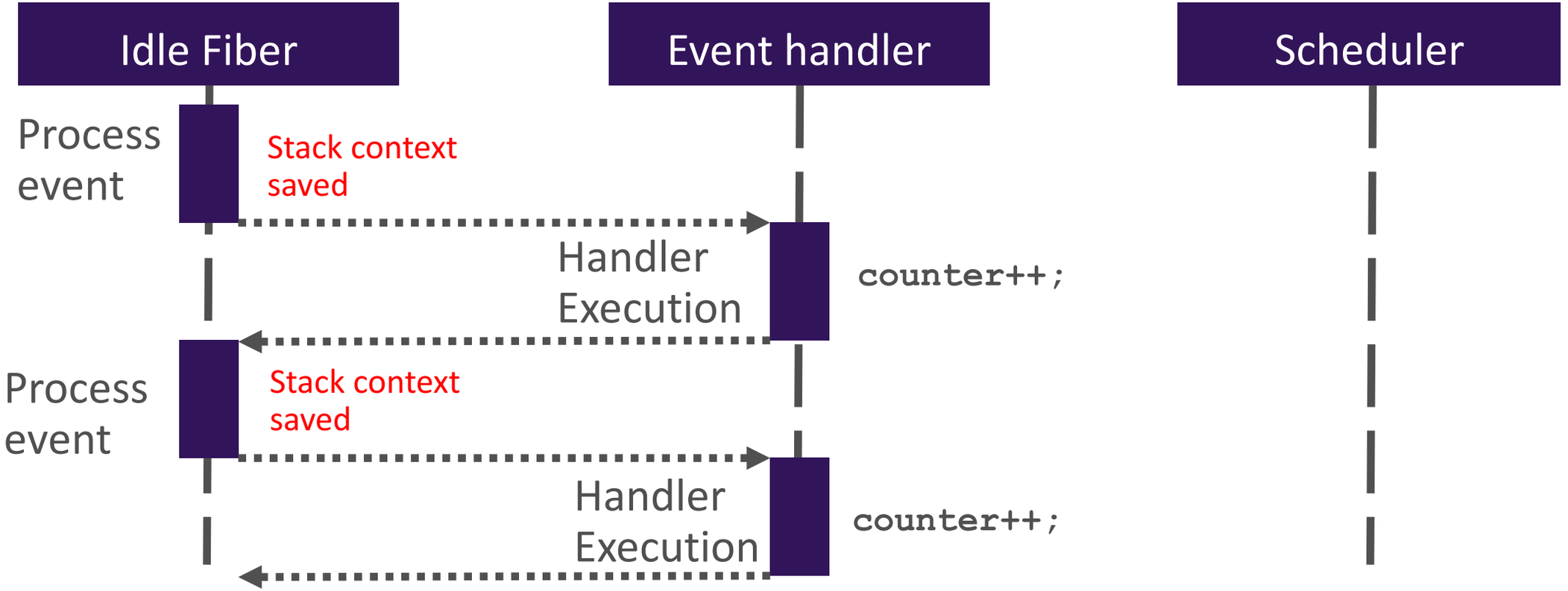
* Fiber Scheduler: Providing Concurrent behaviour

- Events **only** create a new fiber if they block the calling fiber

```
void onButtonA(MicroBitEvent)
{
    // Spawns a fiber
    uBit.display.scroll("Hello!");
}

void onButtonB(MicroBitEvent)
{
    // Doesn't spawn a fiber
    uBit.display.scrollAsync("Goodbye!");
}
```

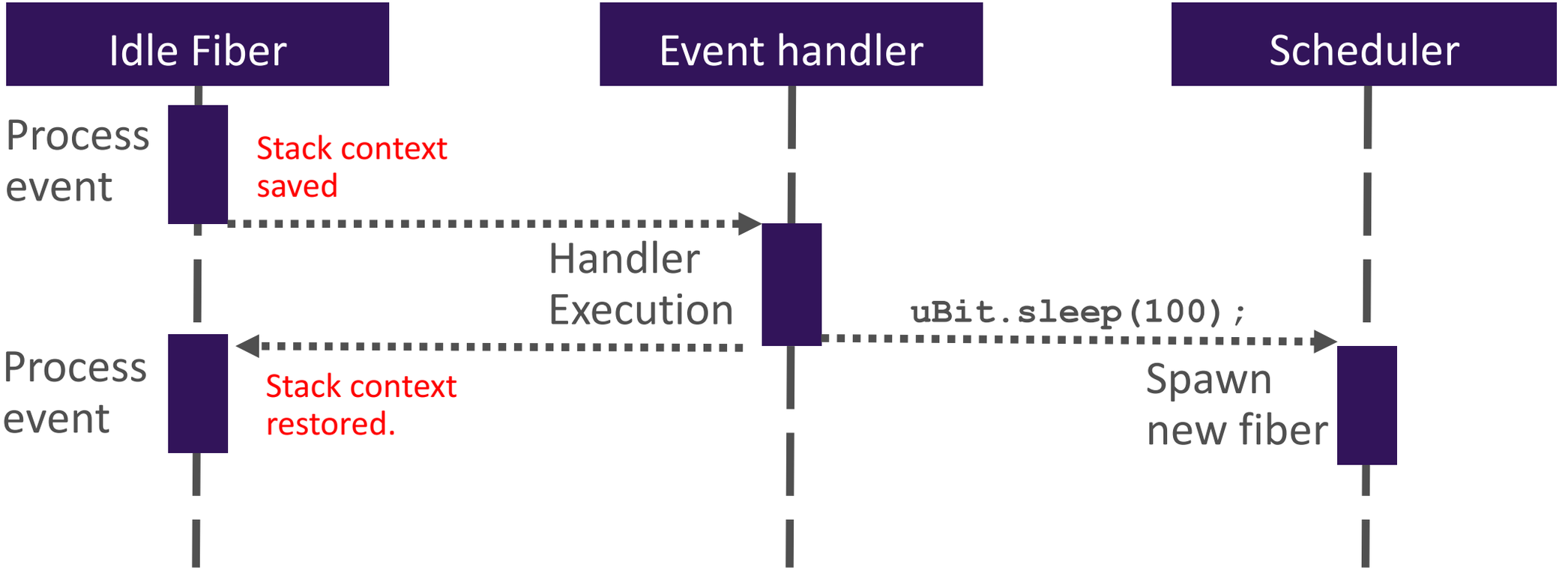
* Fiber Scheduler: Providing Concurrent behaviour

- Events **only** create a new fiber if they block the calling fiber

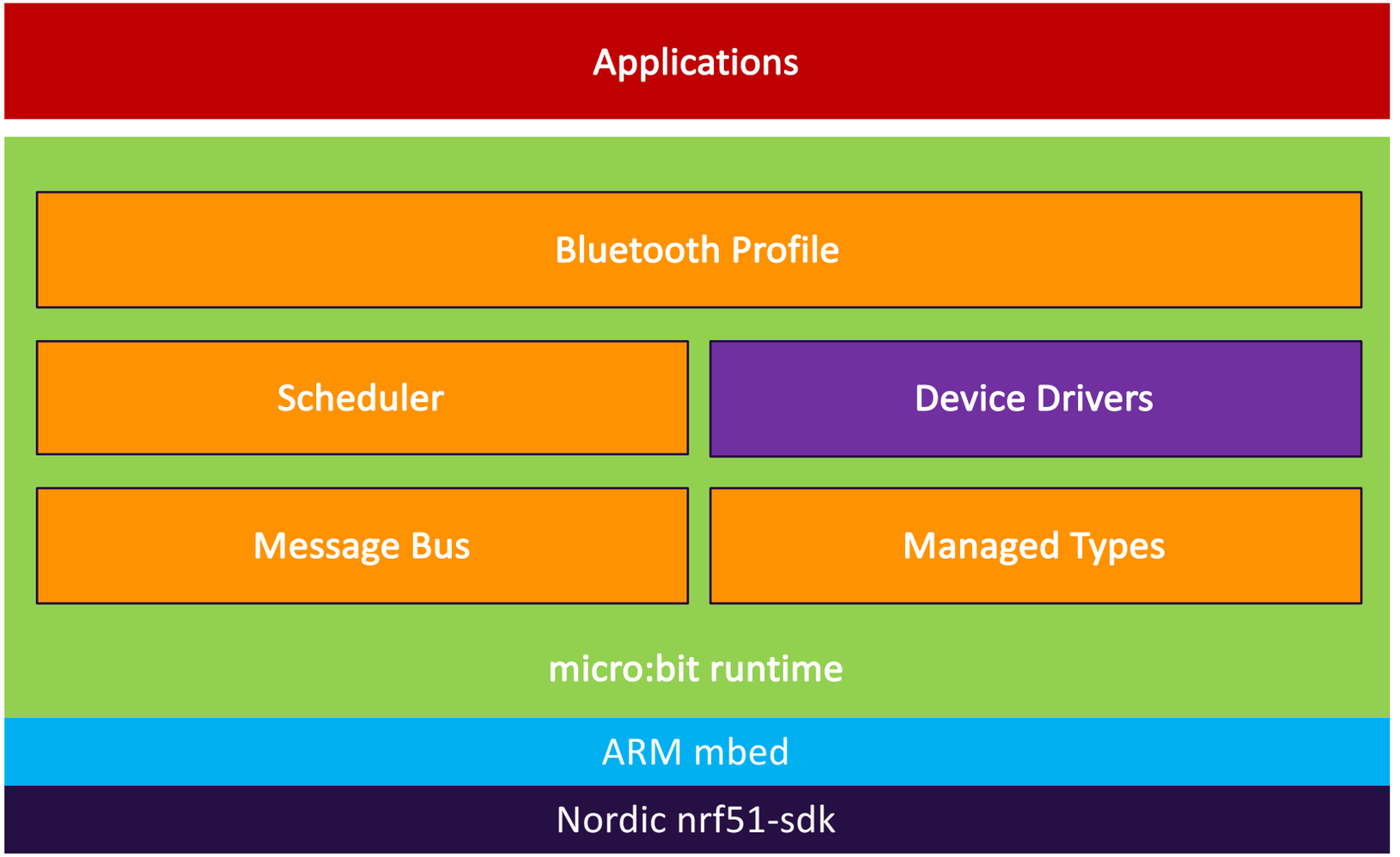


* Fiber Scheduler: Providing Concurrent behaviour

- Events **only** create a new fiber if they block the calling fiber



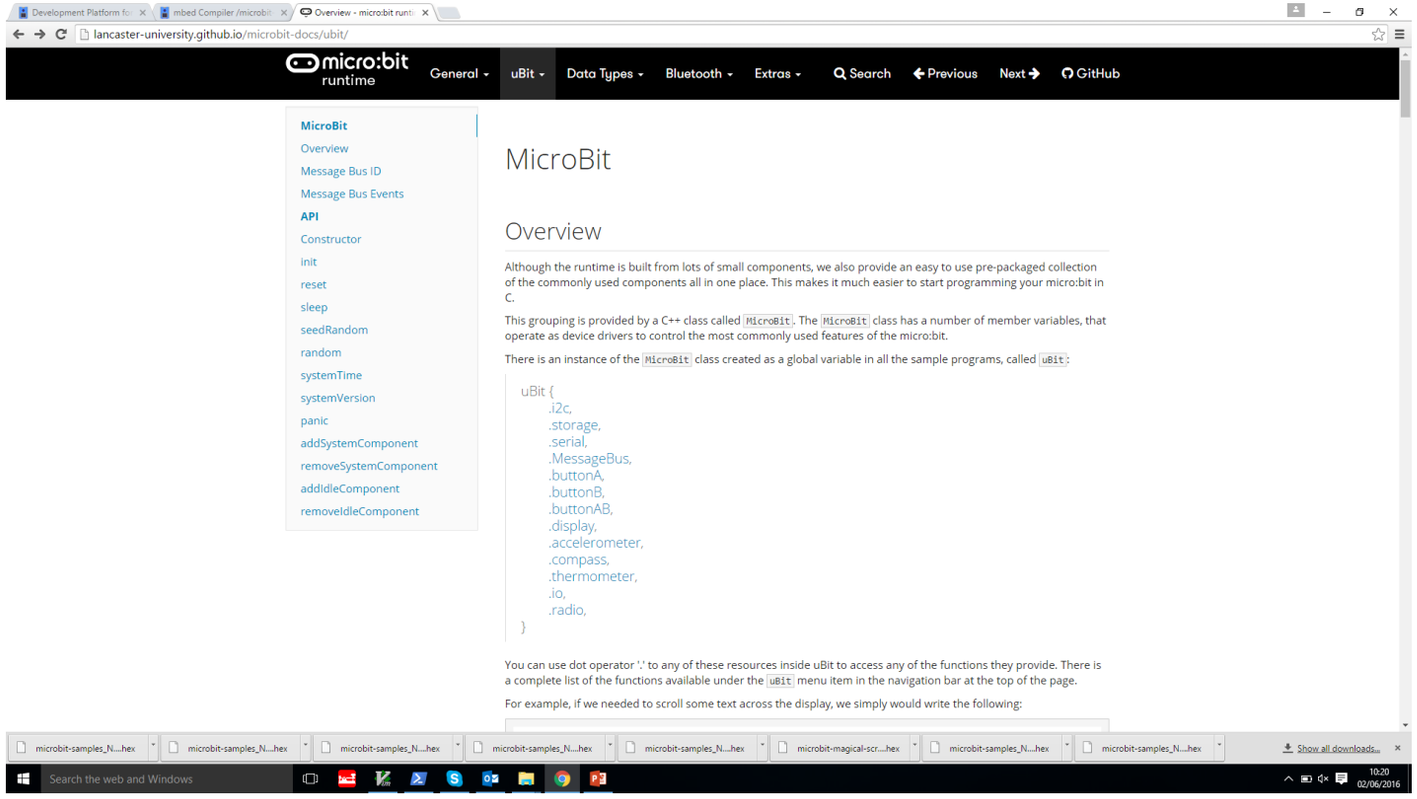
micro:bit runtime architecture



Device Drivers

- Each hardware component is supported by a corresponding C++ software component:
- Each component individually instantiable

  - MicroBitAccelerometer
  - MicroBitButton
  - MicroBitMultiButton
  - MicroBitCompass
  - MicroBitDisplay
  - MicroBitIO
  - MicroBitLightSensor
  - MicroBitRadio
  - MicroBitSerial
  - MicroBitStorage
  - MicroBitThermometer



@microbitruntime

lancaster-university/microbit-dal

Lancaster University

Device Drivers

- An abstract model that represents a device, in this case the micro:bit

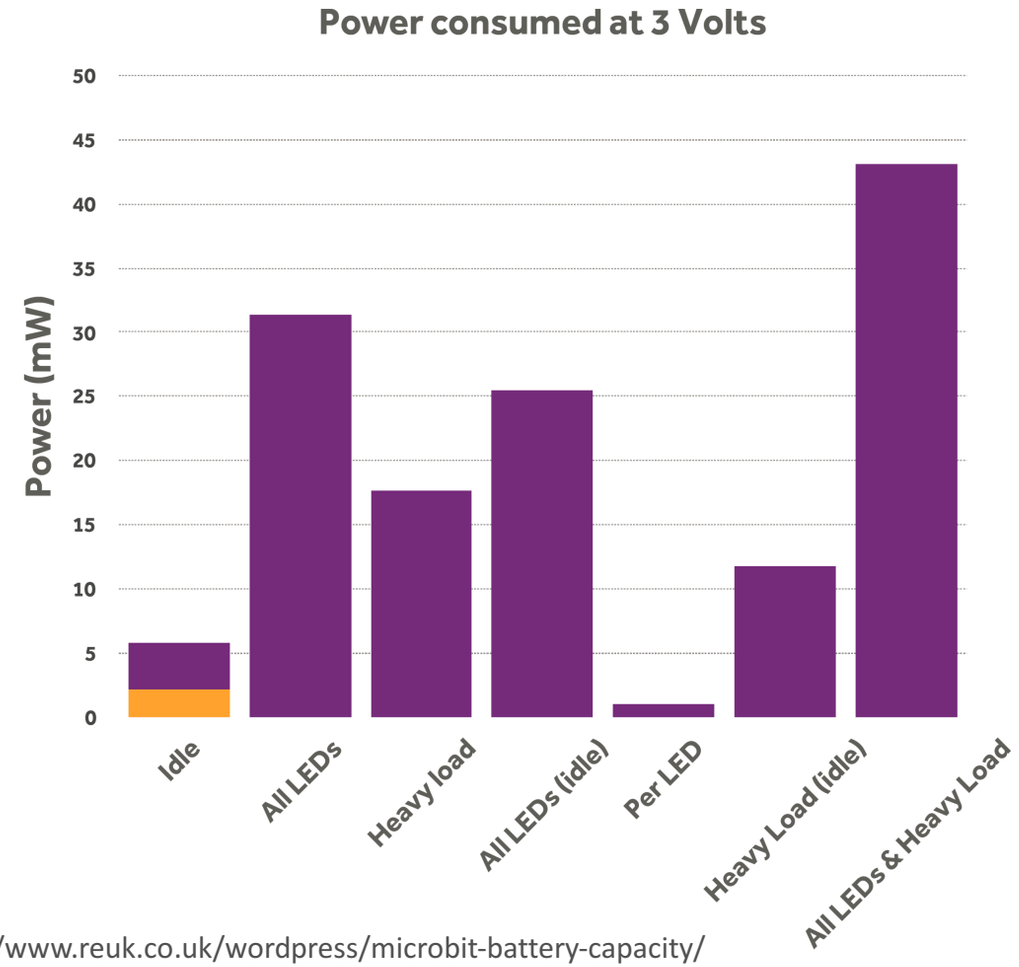
```
class MicroBit{
    MicroBitSerial          serial;
    InterruptIn             resetButton;
    MicroBitStorage         storage;
    MicroBitI2C             i2c;
    MicroBitMessageBus      messageBus;
    MicroBitDisplay         display;
    MicroBitButton          buttonA;
    MicroBitButton          buttonB;
    MicroBitMultiButton     buttonAB;
    MicroBitAccelerometer accelerometer;
    MicroBitCompass         compass;
    MicroBitThermometer     thermometer
    MicroBitIO              io;
    MicroBitRadio           radio;
    BLEDevice               ble;
}
```

Device Drivers

- Reduces complexity for high level languages and our users

```
MicroBit uBit;

int main()
{
    // initialise runtime
    uBit.init();

    // code!
    uBit.display.scroll("Hello World!");
}
```

Power Efficiency



http://www.reuk.co.uk/wordpress/microbit-battery-capacity/

Pi 3 ~ 2000mW

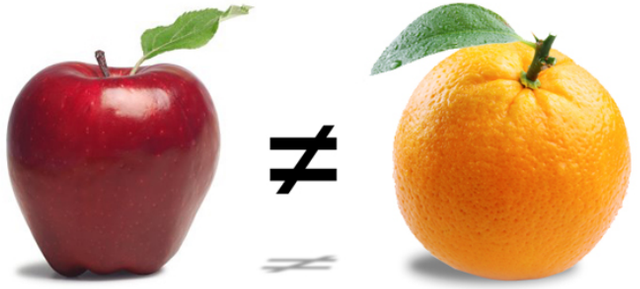
https://www.raspberrypi.org/help/faqs/

Pi Zero ~500mW

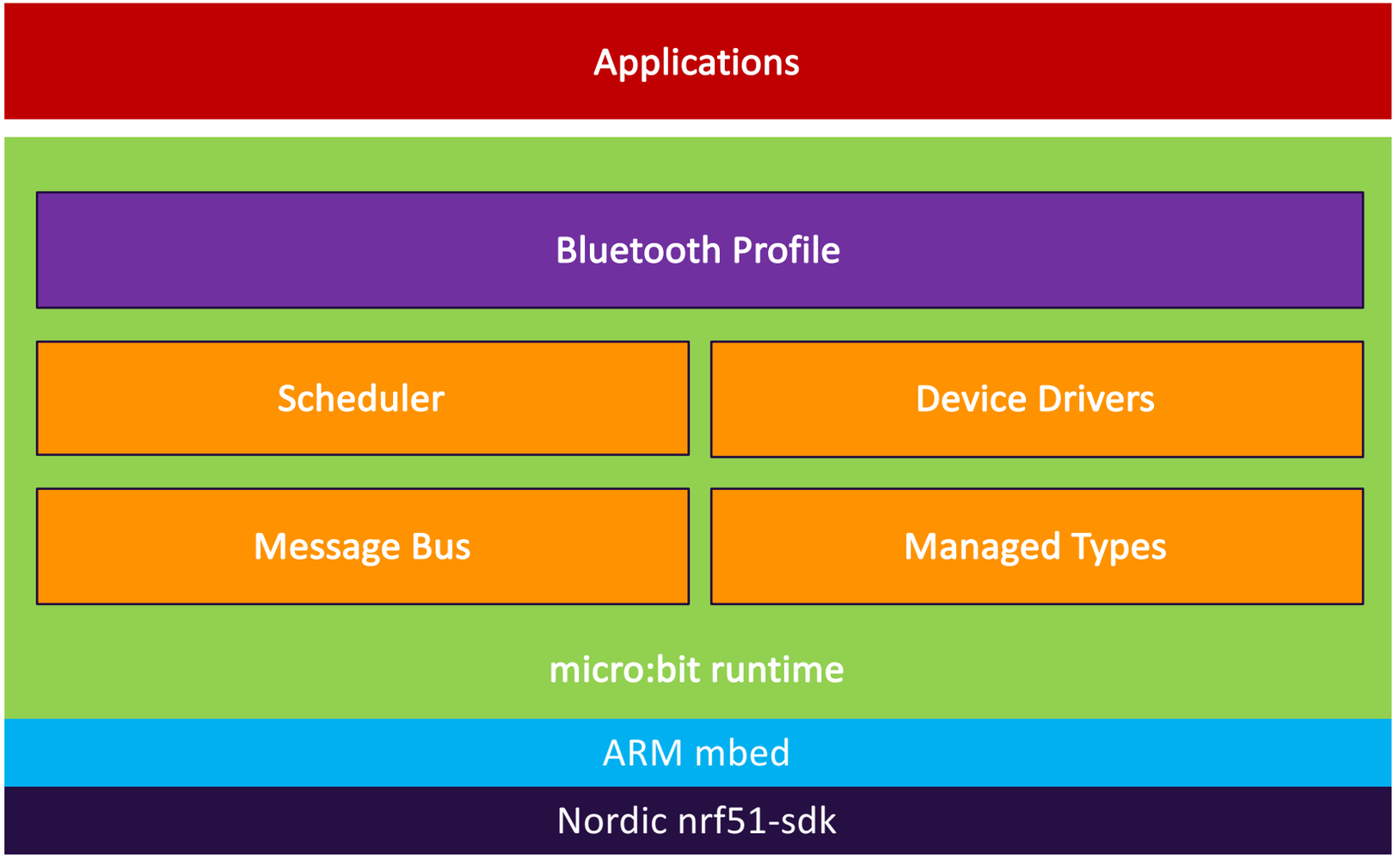
http://raspi.tv/2015/raspberry-pi-zero-power-measurements

Arduino Zero ~240mW

https://www.arduino.cc/en/Tutorial/ArduinoZeroPowerConsumption



micro:bit runtime architecture

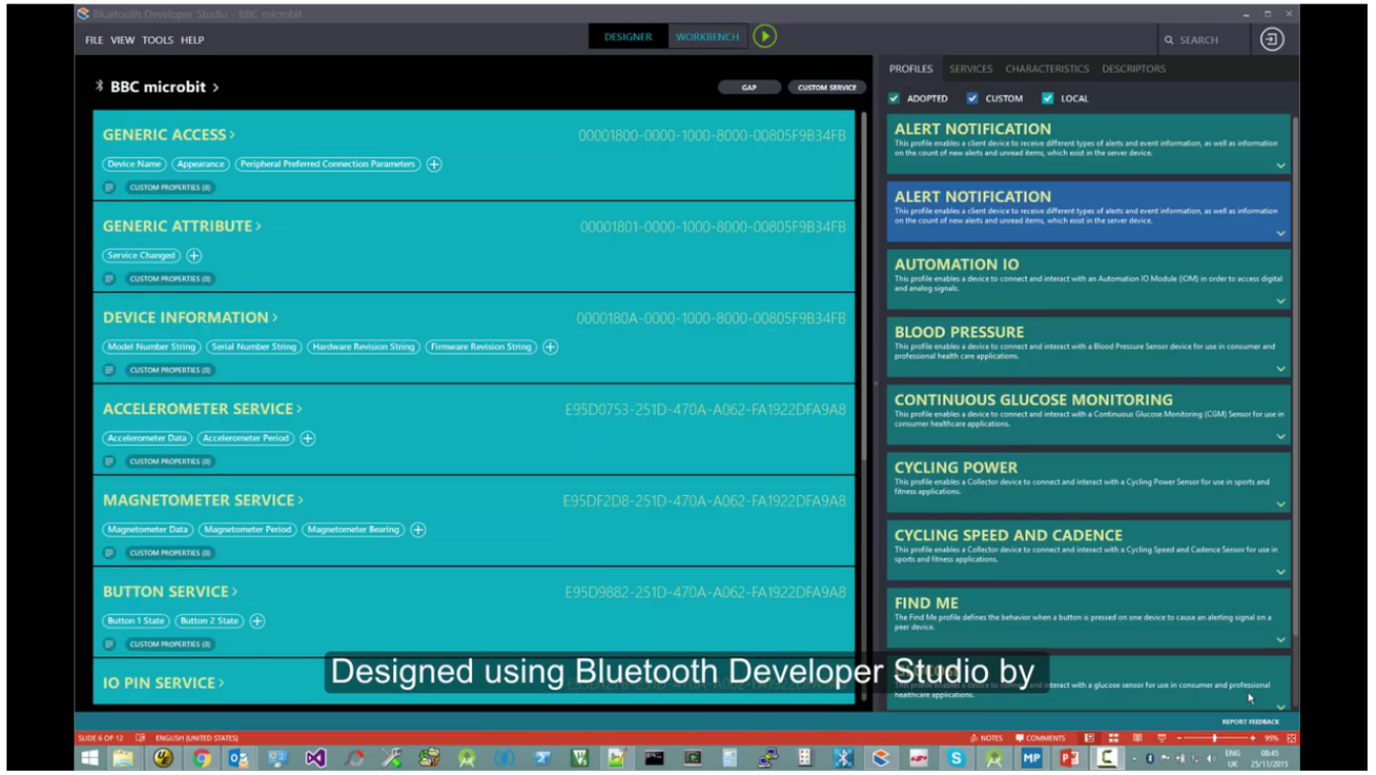


Bluetooth Profile

- Each driver component also mapped as RESTful Bluetooth API…

  - MicroBitAccelerometerService
  - MicroBitButtonService
  - MicroBitMagnetometerService
  - MicroBitLEDService
  - MicroBitIOPinService
  - MicroBitTemperatureService
  - MicroBitEventService
  - UARTService
  - DeviceFirmwareUpdate

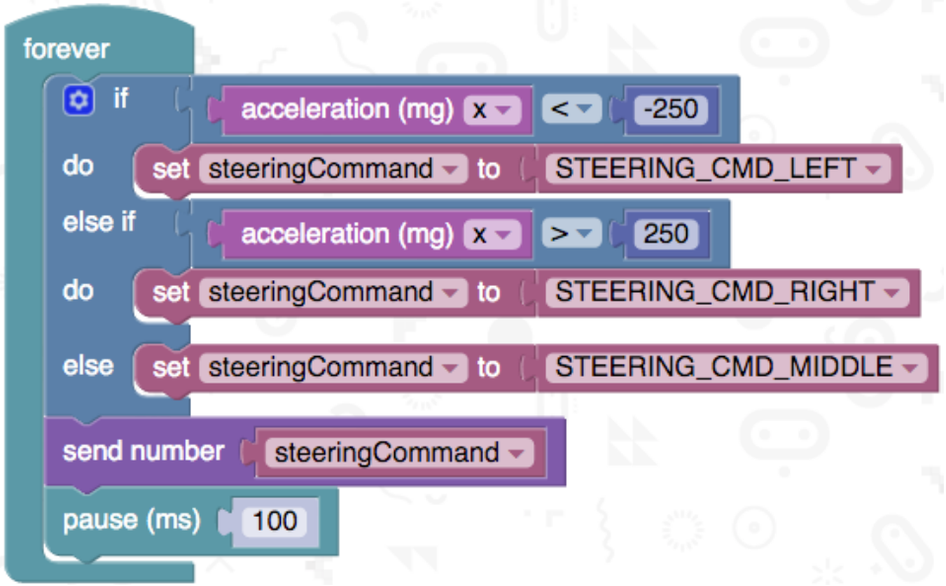
  - Keyboard HID (coming soon)
  - iBeacon/Eddystone (coming soon)

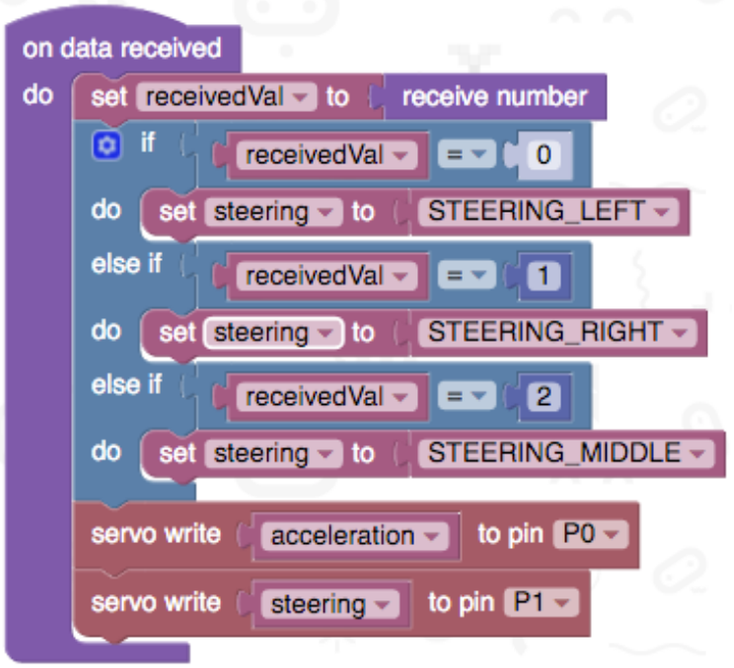


microbit_profile_overview from Martin Woolley on Vimeo.

MicroBitRadio

Simple, raw packet communications…





Future work: Platform Independence

- How do we support devices with varying capabilities and different hardware components?
- How do we add new boards?
- Arduino Zero and the MKR1000 are our case studies to help us generalise

Demo: On Chip File System

- Enables both reading and writing files
- Enables classroom scenarios around data logging

Summary

- Introduced the micro:bit runtime
- Outlined the core concepts of the runtime
- Explained some of the challenges we faced, and how we solved them
- Outlined our future challenges and direction

http://lancaster-university.github.io/microbit-docs/

https://developer.mbed.org/platforms/Microbit/

https://codethemicrobit.com/

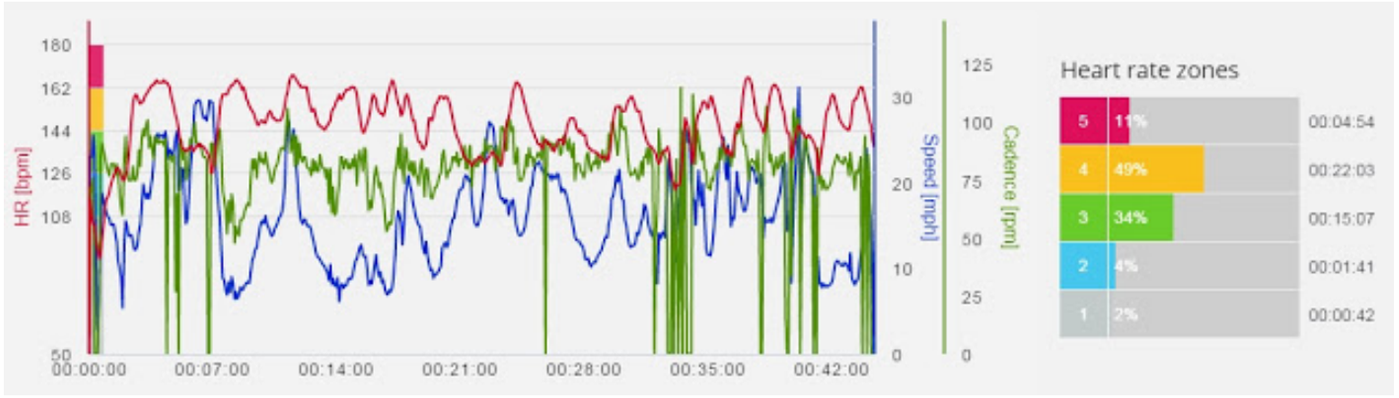
https://www.microbit.co.uk/

@microbitruntime

lancaster-university/microbit-dal

Lancaster University

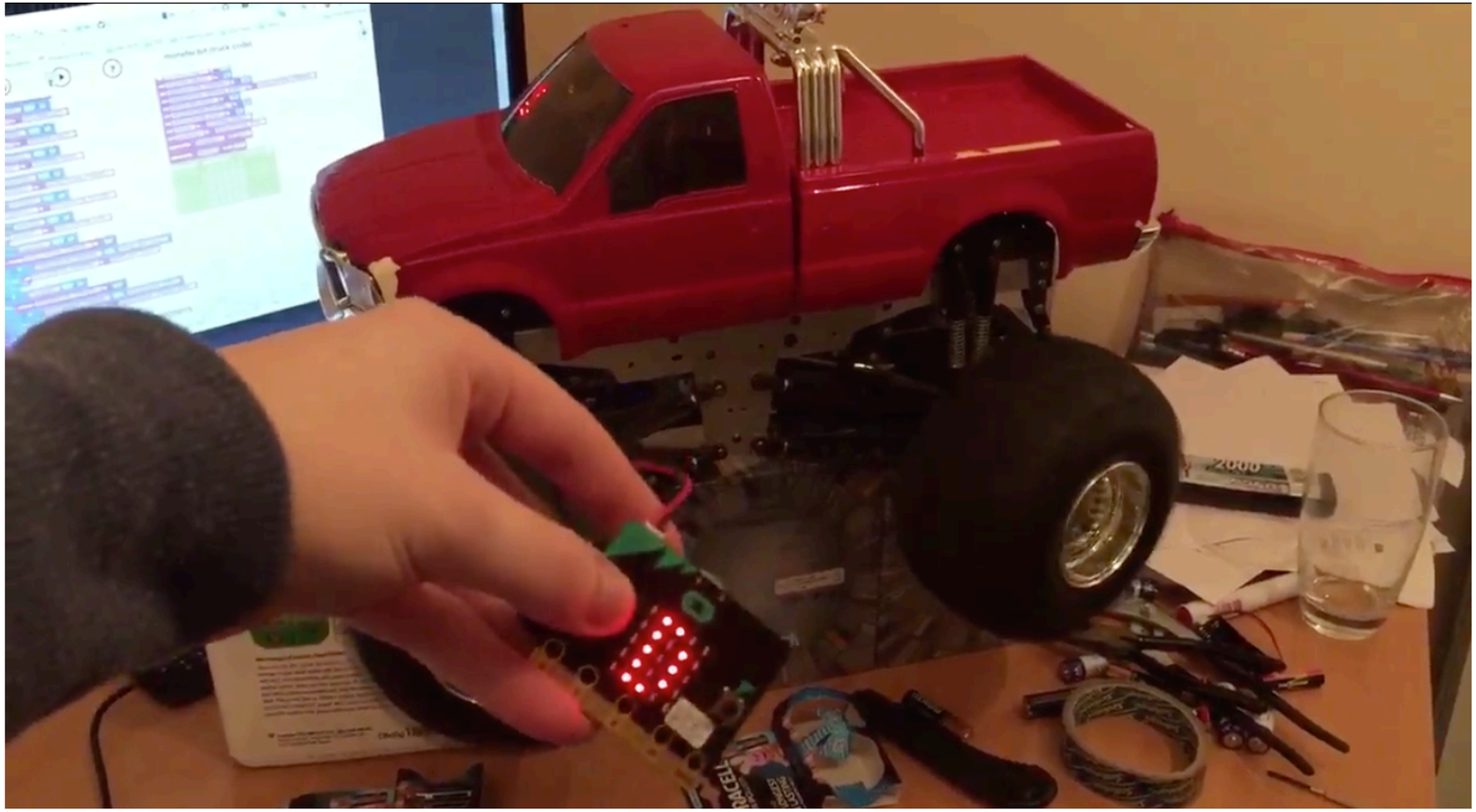
Bluetooth Profile

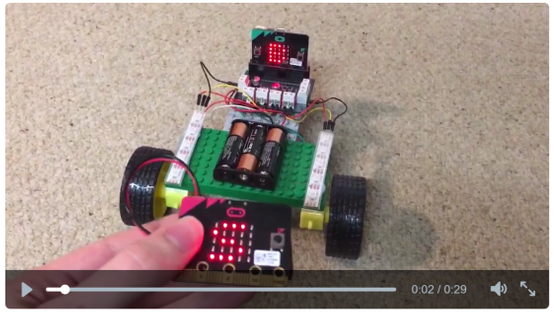


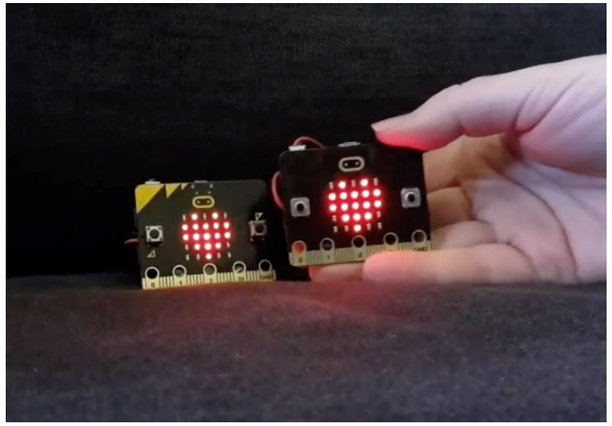
© Martin Woolley Bluetooth SIG

http://bluetooth-mdw.blogspot.co.uk/p/bbc-microbit.html
https://play.google.com/store/apps/details?id=com.bluetooth.mwoolley.microbitbledemo

MicroBitRadio







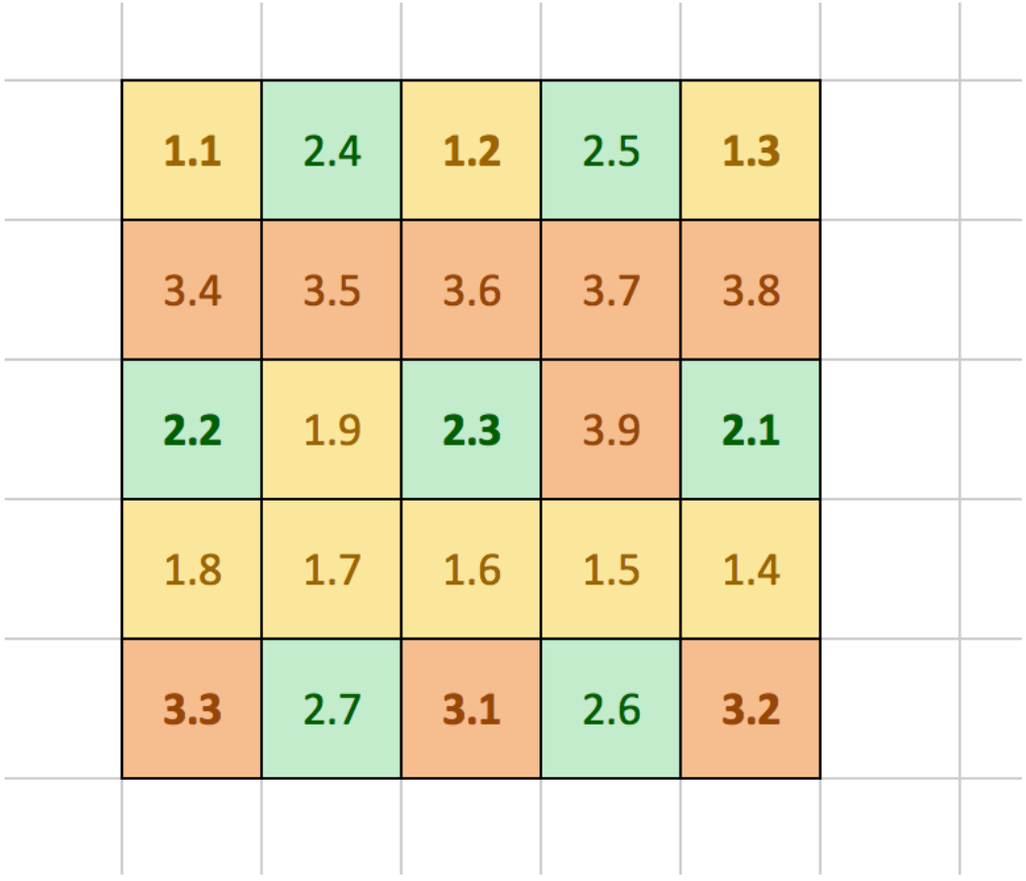
@microbitruntime

lancaster-university/microbit-dal

Lancaster University

The Display

- 3 rows, 9 columns
- One active row permitted at any time
- Our eyes are poor sensors
- Supports 8-bit greyscale.



Light Sensor

- Exploit the age old trick of reverse polarising LEDs
- 3 possible channels
- The display can be used at the same time as sensing