
# 1AD Tutorial Session

# Part One: Introducing Ariel OS

# Constrained Embedded — Target Devices

Little flash memory:
hundreds of kibibytes

Little RAM:
dozens/hundreds of kibibytes
=> no `alloc`

Low energy consumption:
multi-month battery life

Networking capabilities:
Wi-Fi, USB, Ethernet,
LTE-M, LoRa

# Constrained Embedded — Target Devices

Little flash memory:
hundreds of kibibytes

Little RAM:
dozens/hundreds of kibibytes
=> no `alloc`

Low energy consumption:
multi-month battery life

Networking capabilities:
Wi-Fi, USB, Ethernet,
LTE-M, LoRa

# Constrained Embedded — Target Devices

Little flash memory:
hundreds of kibibytes

Little RAM:
dozens/hundreds of kibibytes
=> no `alloc`

Low energy consumption:
multi-month battery life

Networking capabilities:
Wi-Fi, USB, Ethernet,
LTE-M, LoRa

# Constrained Embedded — Target Devices

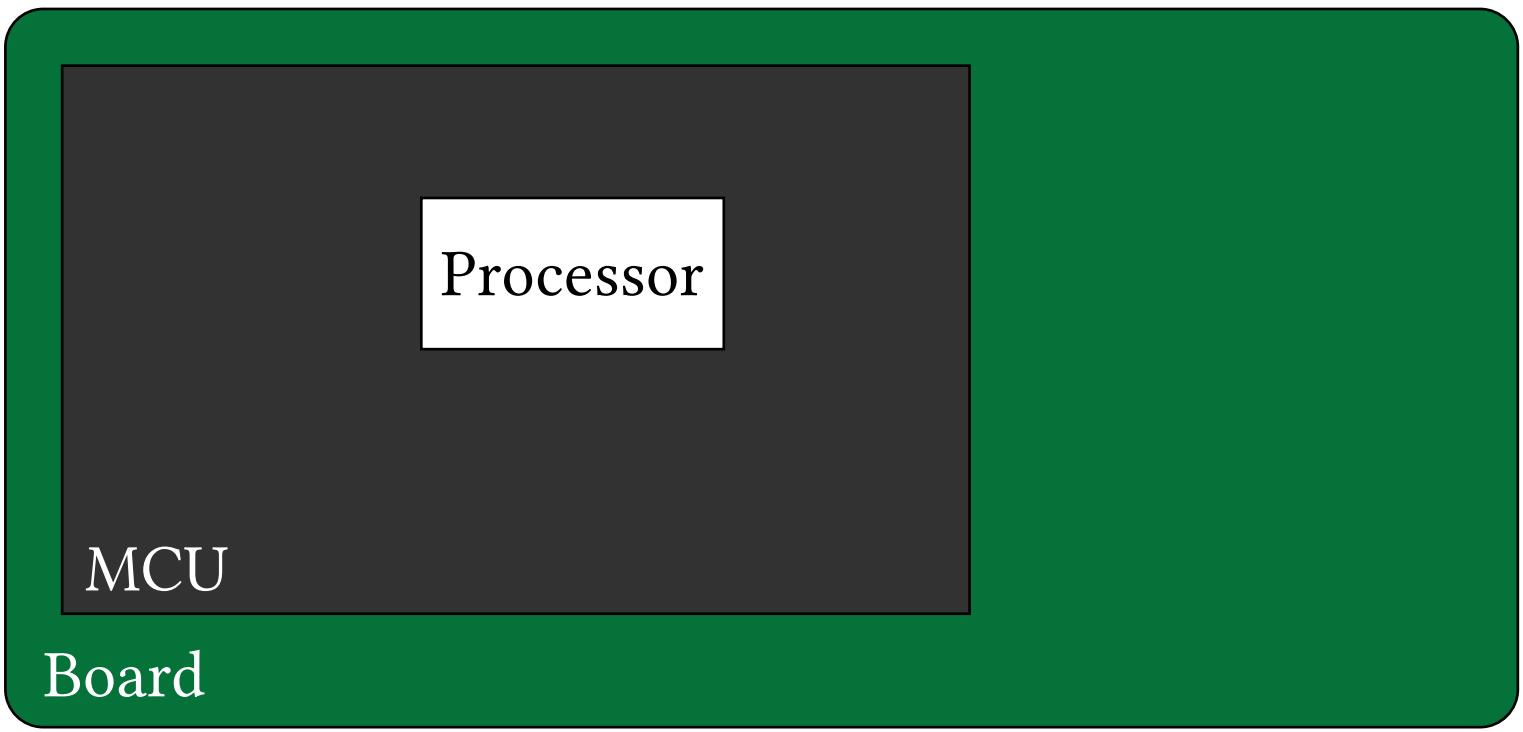
Little flash memory:
hundreds of kibibytes

Little RAM:
dozens/hundreds of kibibytes
=> no `alloc`

Low energy consumption:
multi-month battery life

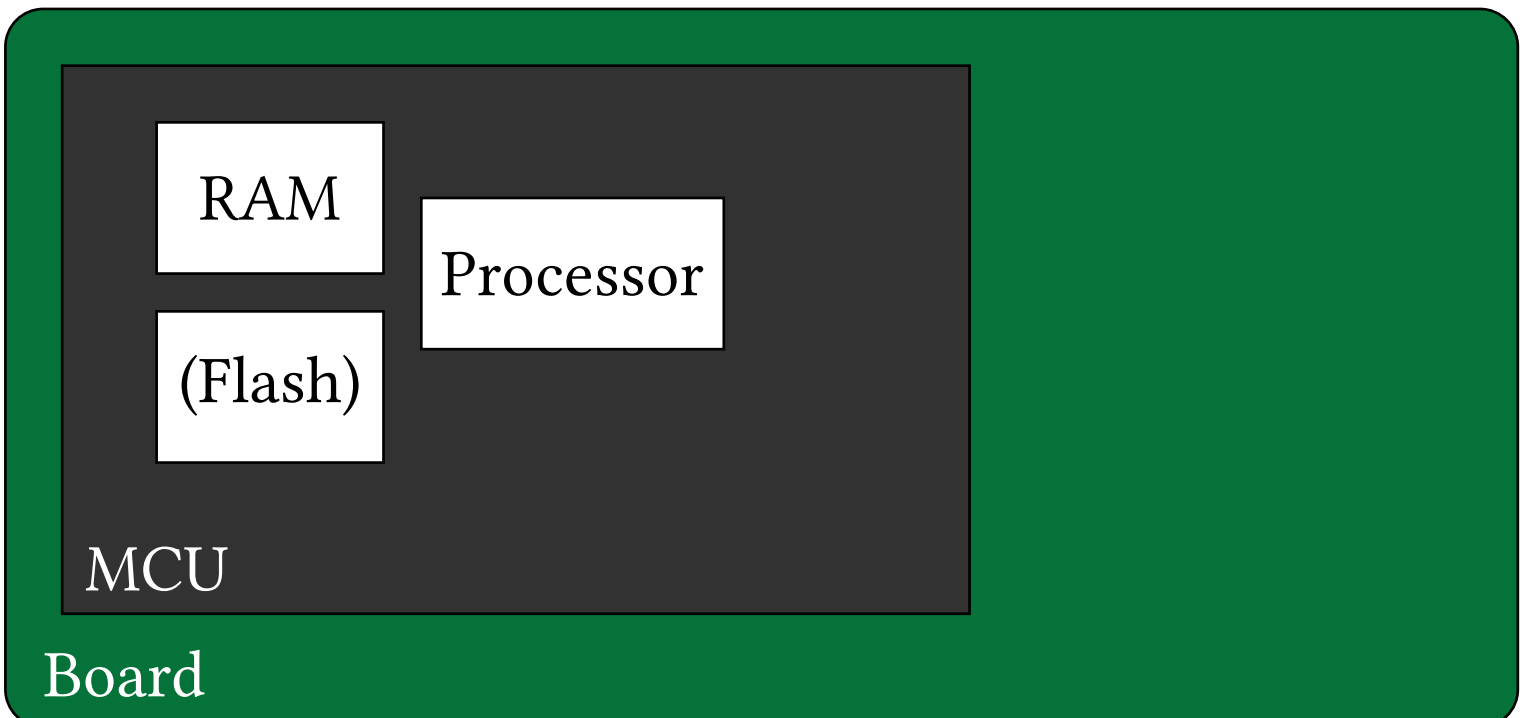
Networking capabilities:
Wi-Fi, USB, Ethernet,
LTE-M, LoRa

# Microcontrollers and Boards



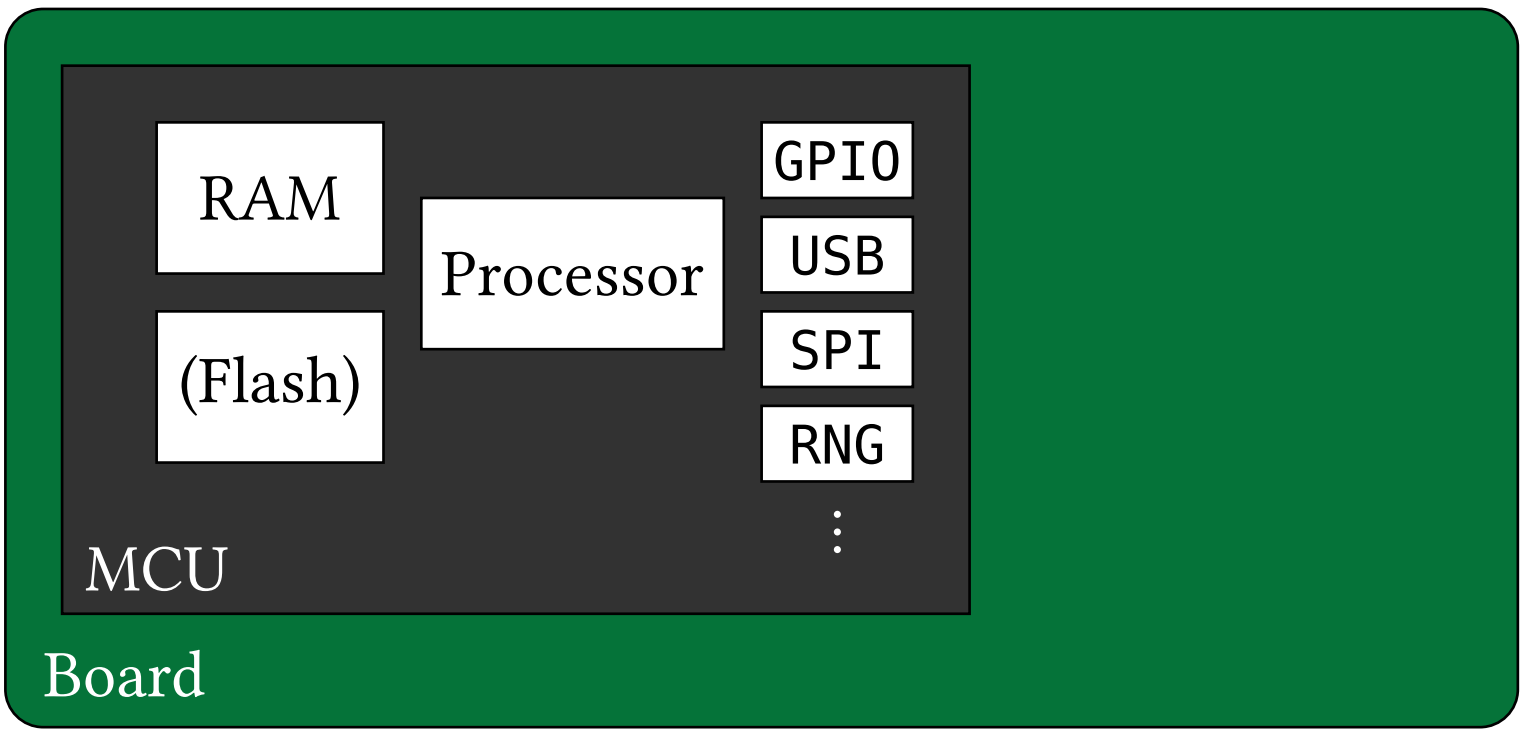

MCU: Micro-Controller Unit

# Microcontrollers and Boards



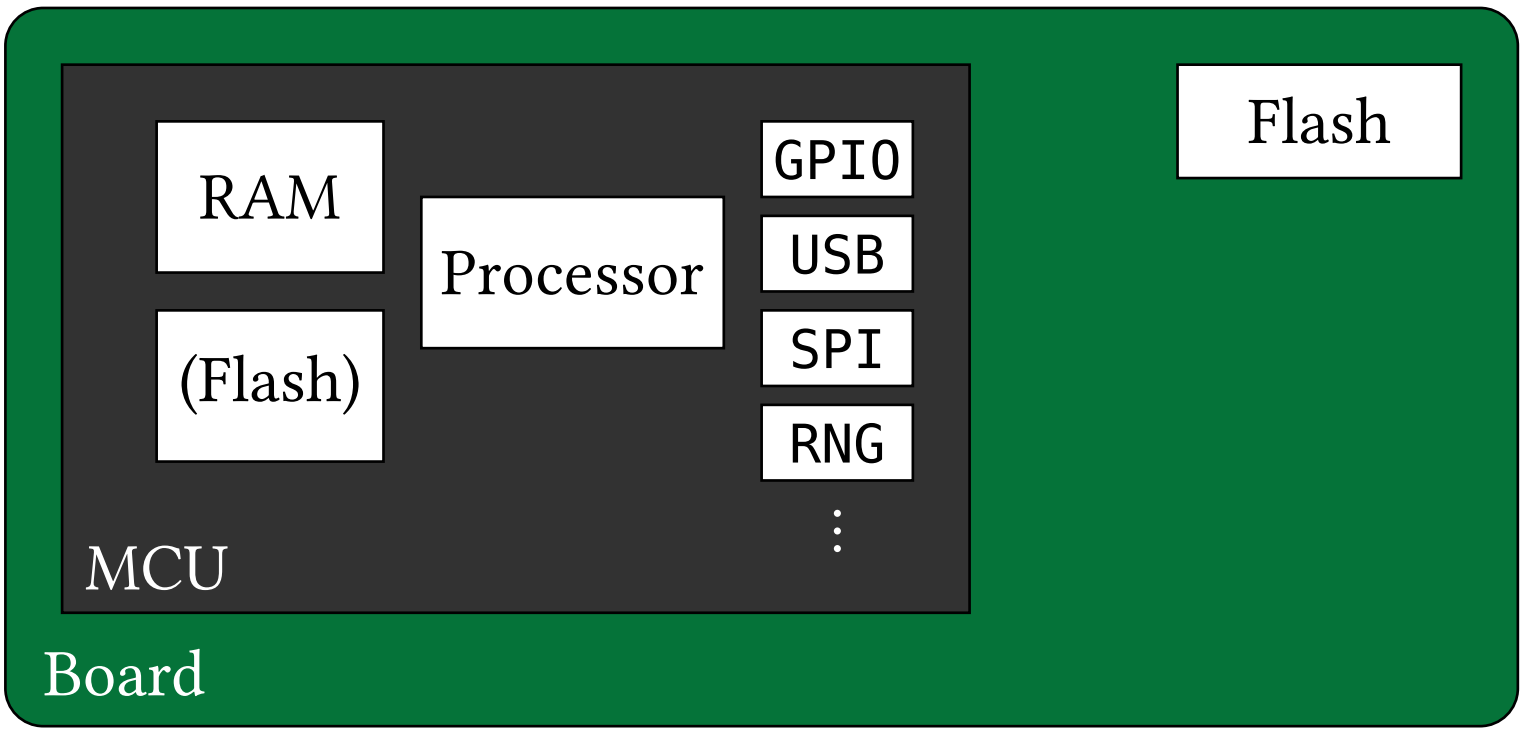

MCU: Micro-Controller Unit

# Microcontrollers and Boards



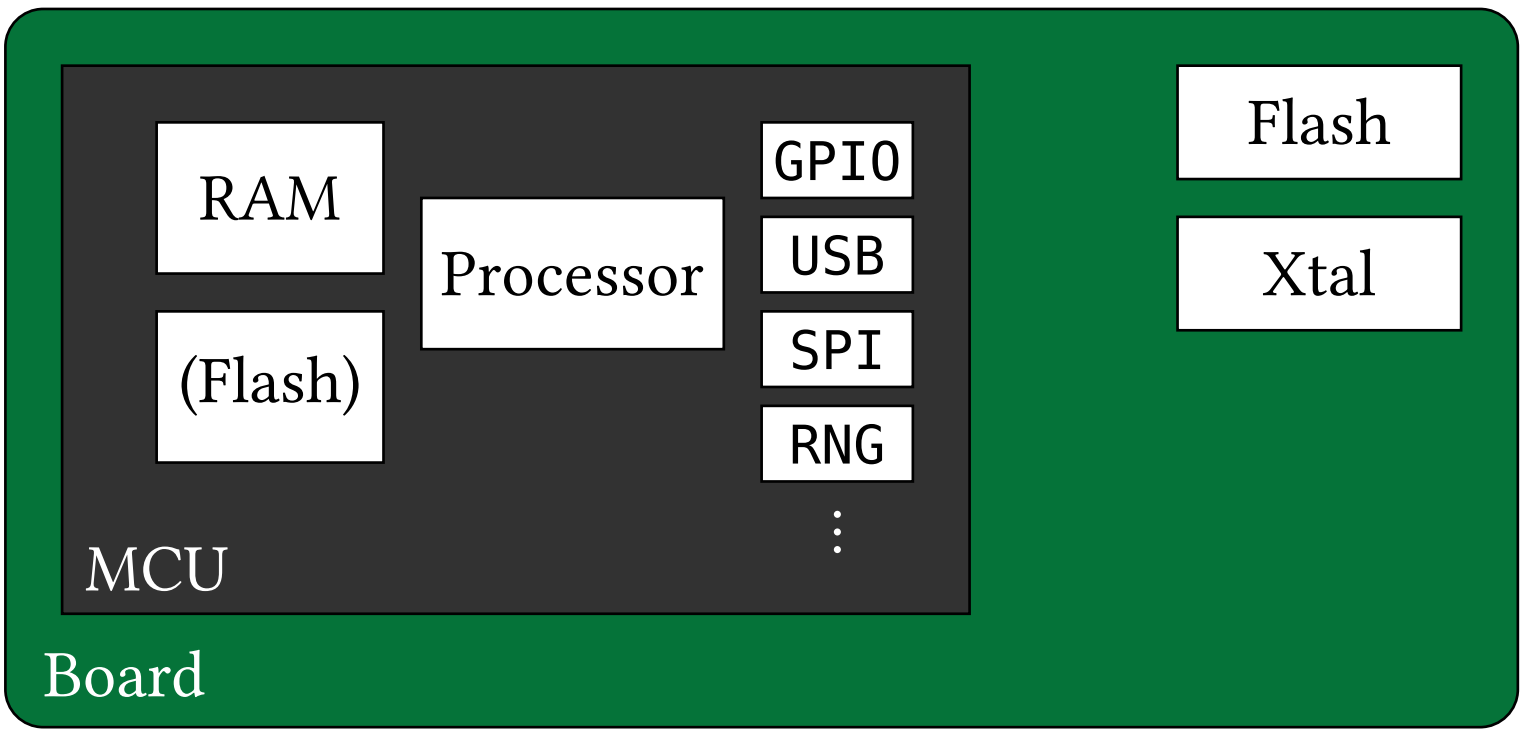

MCU: Micro-Controller Unit

# Microcontrollers and Boards



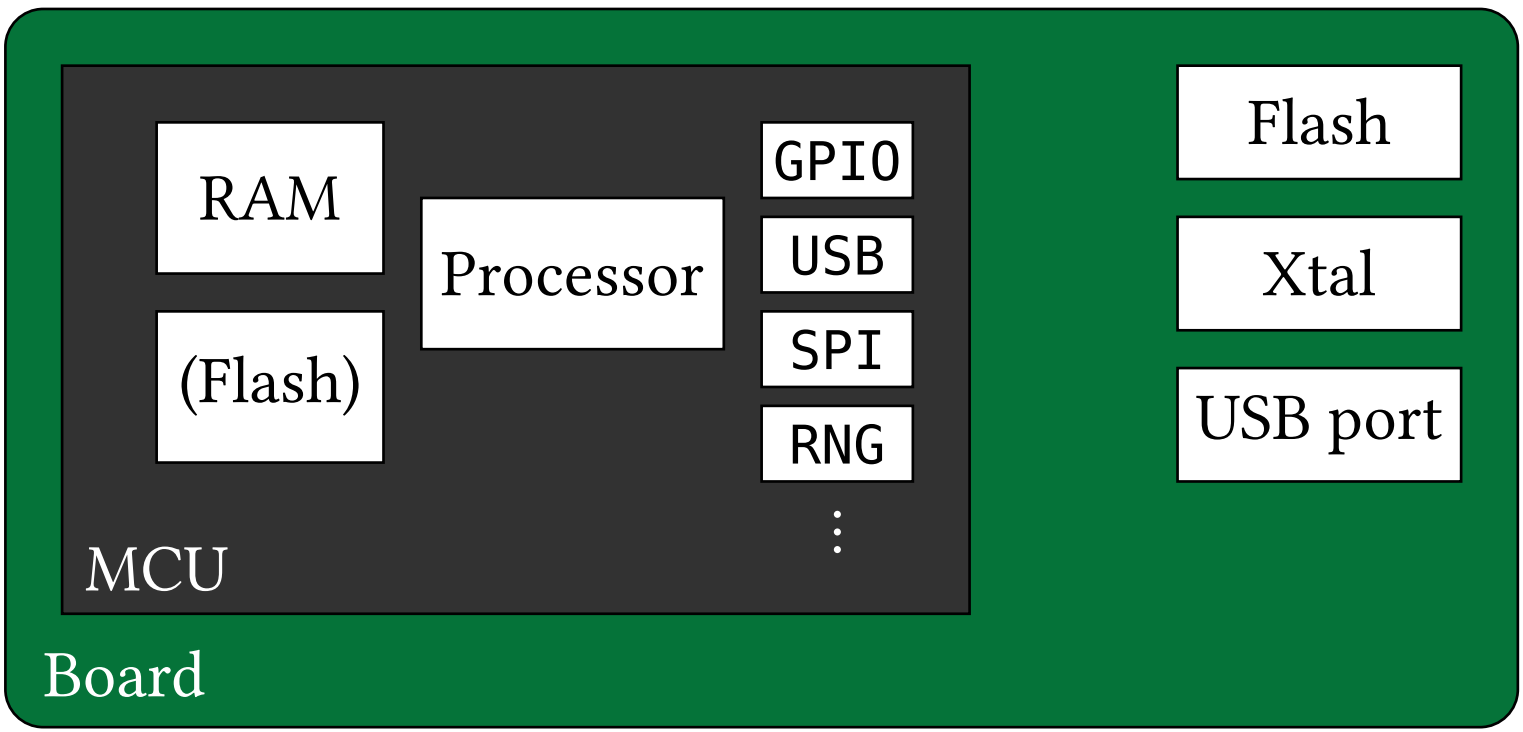

MCU: Micro-Controller Unit

# Microcontrollers and Boards



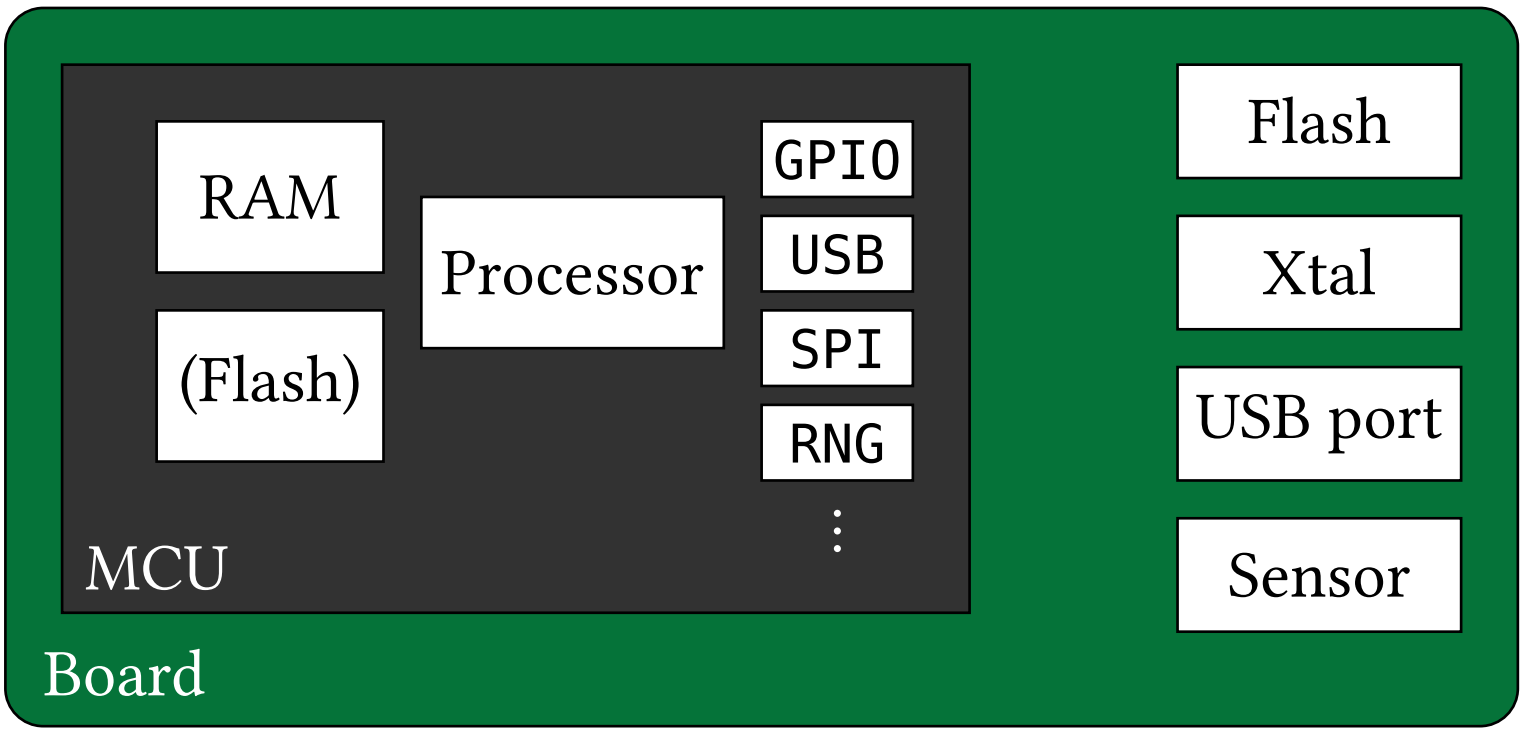

MCU: Micro-Controller Unit

# Microcontrollers and Boards




MCU: Micro-Controller Unit

# Microcontrollers and Boards




MCU: Micro-Controller Unit

# Memory-Mapped Peripherals and Abstraction Layers

Hardware Abstraction Library (HAL)
e.g., `embassy-nrf`

```rust
let mut led = Output::new(P0_13, Level::High);
led.set_low();
```

Peripheral Access Crate (PAC), e.g., `nrf-pac`

```rust
// + GPIO configuration
nrf_pac::P0.outclr().write(|w| w.set_pin(13, true));
```

Hardware (nRF52840)

```rust
// + GPIO configuration
unsafe {
    let gpio_p0_outclr = 0x5000_050c as *mut u32;
    // Set P0.13 low
    gpio_p0_outclr.write_volatile(1 << 13);
}
```

GPIO: General-Purpose Input/Output

# Embassy and Application Portability — RNG example

stm32l4

```
bind_interrupts!(struct Irqs {
  RNG =>
    rng::InterruptHandler<peripherals::RNG>;
});

let config = /* board-specific clk config */;
let p = embassy_stm32::init(config);

let mut buf = [0u8; 16];


let mut rng = Rng::new(p.RNG, Irqs);
rng.fill_bytes(&mut buf);
```

rp235x

```
bind_interrupts!(struct Irqs {
  TRNG_IRQ =>
    trng::InterruptHandler<peripherals::TRNG>;
});

let config = /* board-specific clk config */;
let p = embassy_rp::init(config);

let mut buf = [0u8; 16];

let c = RngConfig::default();
let mut rng = Trng::new(p.TRNG, Irqs, c);
rng.blocking_fill_bytes(&mut buf);
```

# Embassy and Application Portability — RNG example

stm32u0

```
bind_interrupts!(struct Irqs {
  RNG_CRYP =>
    rng::InterruptHandler<peripherals::RNG>;
});

let config = /* board-specific clk config */;
let p = embassy_stm32::init(config);

let mut buf = [0u8; 16];


let mut rng = Rng::new(p.RNG, Irqs);
rng.fill_bytes(&mut buf);
```

stm32l4

```
bind_interrupts!(struct Irqs {
  RNG =>
    rng::InterruptHandler<peripherals::RNG>;
});

let config = /* board-specific clk config */;
let p = embassy_stm32::init(config);

let mut buf = [0u8; 16];


let mut rng = Rng::new(p.RNG, Irqs);
rng.fill_bytes(&mut buf);
```

# Ariel OS: an Embedded OS for IoT — Features

Application portability:
ESP32, nRF, RPi RP, STM32

Modularity & ecosystem integration

Preemptive scheduler with multicore support

Networking capabilities

# Ariel OS: an Embedded OS for IoT — Features

Application portability:
ESP32, nRF, RPi RP, STM32

Modularity & ecosystem integration

Preemptive scheduler with multicore support

Networking capabilities

# Ariel OS: an Embedded OS for IoT — Features

Application portability:
ESP32, nRF, RPi RP, STM32

Modularity & ecosystem integration

Preemptive scheduler with multicore support

Networking capabilities

# Ariel OS: an Embedded OS for IoT — Features

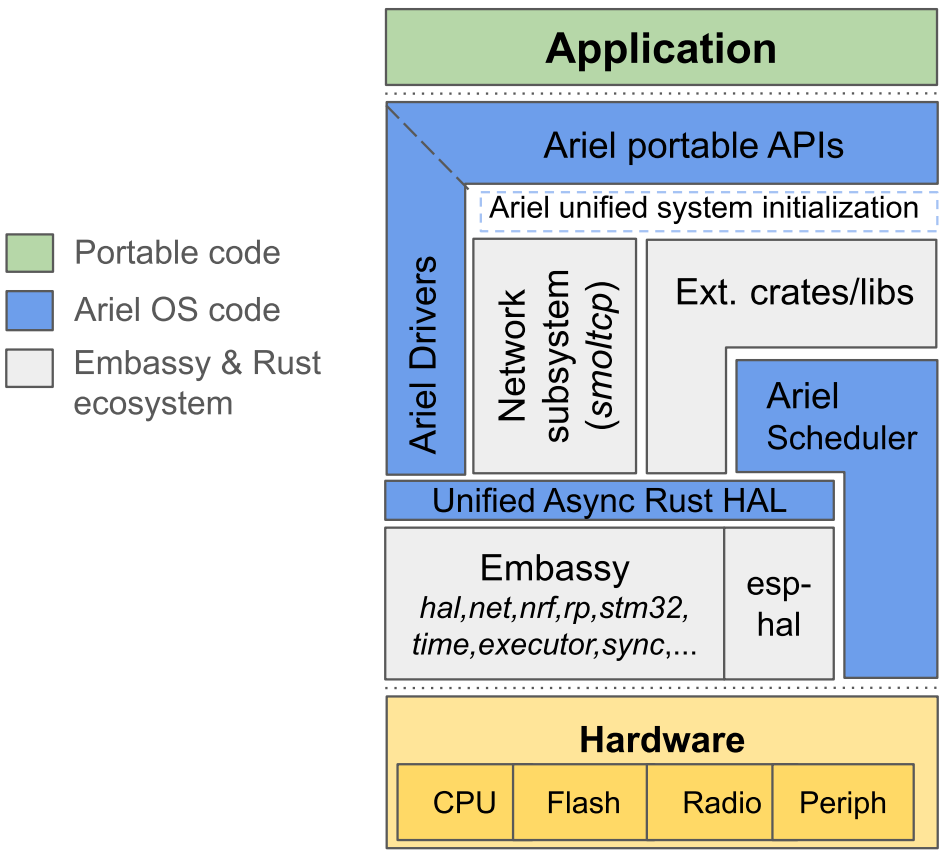
Application portability:
ESP32, nRF, RPi RP, STM32

Modularity & ecosystem integration

Preemptive scheduler with multicore support

Networking capabilities

# Position in the Ecosystem



# Part Two: A Tour

# A Hello World Application

src/main.rs

```rust
#![no_main]
#![no_std]

use ariel_os::debug::{
  exit, log::info, ExitCode,
};

#[ariel_os::task(autostart)]
async fn main() {
    info!("Hello world!");
    exit(ExitCode::SUCCESS);
}
```

Cargo.toml

```toml
[package]
name = "ariel-os-hello"
version = "0.1.0"
edition = "2024"

[dependencies.ariel-os]
path = "build/imports/ariel-os/src/ariel-os"

[dependencies.ariel-os-boards]
path = "build/imports/ariel-os/src/ariel-os-boards"
```

# The Build System: laze

Project layout

```
.
├── build/
│   ⋮
├── src/
│   └── main.rs
├── Cargo.lock
├── Cargo.toml
└── laze-project.yml
```

laze-project.yml

```yaml
imports:
  - git:
      url: https://github.com/ariel-os/ariel-os
      tag: v0.5.0
    dldir: ariel-os

apps:
  - name: ariel-os-hello
```

## Compiling & flashing:

```
laze build -b nrf52840dk run
```

# Using Digital Outputs

src/main.rs

```rust
#[ariel_os::task(autostart, peripherals)]
async fn blinky(p: LedPeripherals) {
    let mut led = Output::new(p.led, Level::Low);

    loop {
        led.toggle();
        Timer::after_millis(500).await;
    }
}

#[cfg(context = "nrf52840dk")]
ariel_os::hal::define_peripherals!(LedPeripherals {
    led: P0_13
});
```

Cargo.toml

```toml
[dependencies.ariel-os]
# [...]
features = ["time"]
```

# Using Digital Outputs — Board Descriptions (SBD)

src/main.rs

```rust
#[ariel_os::task(autostart, peripherals)]
async fn blinky(p: LedPeripherals) {
    let mut led0 = Output::new(p.led0, Level::Low);

    loop {
        led0.toggle();
        Timer::after_millis(500).await;
    }
}
```

boards/stm32u083c-dk.yaml

```yaml
version: 0.5.0
boards:
  stm32u083c-dk:
    chip: stm32u083mc
    flags:
      - has_usb_device_port
    ariel:
      swi: USART4_LPUART3
    leds:
      led0:
        pin: PC13

    # [...]
```

# Using an RNG

src/main.rs

```rust
use rand::Rng as _;

#[ariel_os::task(autostart)]
async fn main() {
  let mut rng = ariel_os::random::fast_rng();
  let value = rng.gen_range(1..=6);
  info!("Dice roll: {}", value);
}
```

laze-project.yml

```yaml
apps:
  - name: <name>
    selects:
      - random
```

Cargo.toml

```toml
[dependencies]
# [...]
rand = "0.9.2"
```

# Setting up Networking

src/main.rs

```rust
#[ariel_os::task(autostart)]
async fn main() {
    let stack = net::network_stack().await.unwrap();

    let mut socket = UdpSocket::new(
        stack,
        // ...buffers
    );
}
```

laze-project.yml

```yaml
apps:
  - name: <name>
    selects:
      - network
```

See the `http-client`, `http-server`, `tcp-echo`, `udp-echo` examples.

# Selecting the Network Link

Defaults to Wi-Fi:

```
CONFIG_WIFI_NETWORK=<ssid> CONFIG_WIFI_PASSWORD=<pwd> laze build -b rpi-pico-w
```

Equivalent to:

```
[CONFIG_WIFI_*] laze build --select wifi-cyw43 -b rpi-pico-w
```

Selecting Ethernet-over-USB instead:

```
laze build --select usb-ethernet --disable network-config-dhcp -b rpi-pico-w
```

# Debug Logging — `defmt`

Logging macros: e.g., `debug!()`, `info!()`, `error!()`

`defmt`: `defmt`-encoding on the wire, strings interned in the ELF:

`defmt-print`, probe-rs, `espflash`:

```
-- this is printed via `println!()`
[INFO ] -- info log level enabled (example_log log/src/main.rs:11)
[WARN ] -- warn log level enabled (example_log log/src/main.rs:12)
[ERROR] -- error log level enabled (just testing) (example_log log/src/main.rs:13)
```

# Debug Logging — `log`

Logging macros: e.g., `debug!()`, `info!()`, `error!()`

Stores strings in-binary:

```
-- this is printed via `println!()`
[INFO ] -- info log level enabled (example_log log/src/main.rs:11)
[WARN ] -- warn log level enabled (example_log log/src/main.rs:12)
[ERROR] -- error log level enabled (just testing) (example_log log/src/main.rs:13)
```

# Multithreading with Multicore Support

src/main.rs

```rust
#[ariel_os::thread(autostart, priority = 2)]
fn thread0() {
    info!("Hello from thread0!");
}

#[ariel_os::thread(autostart)]
fn thread1() {
    info!("Hello from thread1!");
}
```

laze-project.yml

```yaml
apps:
  - name: threading-multicore
    selects:
      - sw/threading
      - multi-core
```

# Compiling for Native

- Ariel OS applications can be compiled as Linux programs
- Useful for testing and simulation
- Supports networking on Linux

```
laze -C examples/random build -b native run
```

# Portable I2C

src/main.rs

```
async fn main(p: pins::Peripherals) {
    let mut i2c_config = hal::i2c::controller::Config::default();
    i2c_config.frequency
        = const { highest_freq_in(Kilohertz::kHz(100)..=Kilohertz::kHz(400)) };
    let mut i2c_bus = pins::SensorI2c::new(p.i2c_sda, p.i2c_scl, i2c_config);

    let mut buf = [0u8];
    i2c.write_read(address, &[Register::Status as u8], &mut buf).await?;
    // Read `buf[0]` to get the status.
}

enum Register {
    Status = 0x05,
}
```

# USB Support: USB CDC-ACM

src/main.rs

```rust
#[ariel_os::task(autostart, usb_builder_hook)]
async fn main() {
    static STATE: StaticCell<State<'_>> = StaticCell::new();
    let mut class = USB_BUILDER_HOOK.with(|builder| {
        CdcAcmClass::new(
            builder,
            STATE.init_with(State::new),
            MAX_FULL_SPEED_PACKET_SIZE.into(),
        )
    }) .await;

    let data = /* data */;
    class.write_packet(data).await?;
}
```

laze-project.yml

```yaml
apps:
  - name: usb-serial
    selects:
      - usb
```