IoT workshop for firmware programming with ESP32 and mruby/c

HASUMI Hitoshi (@hasumikin)

Monstar Lab, Matsue

March 16, 2019 in Matsue, Japan
May 6, 2019 in Warszawa, Polska
May 15, 2019 in Kraków, Polska

Cześć!

about me

- HASUMI Hitoshi
  @hasumikin
- Ruby 💎
  Sake 🍶
  Soba 🍜
  Coffee ☕

about me

about me

Global Sourcing

VIETNAM

DANANG/HANOI

Strength: Mobile App Development (iOS,Android)
Languages: Japanese, English, Vietnamese
Development Languages: PHP, Java, Ruby, iOS,Android, Swift

BANGLADESH

DHAKA

Strength: Web Development
Languages: English, Japanese, Bengali
Development Languages: Java, PHP, JavaScript, Swift, Objective C

CHINA

BEIJING/SHANGHAI
QINGDAO/CHENGDU

Strength: Game Development
Languages: Japanese, English,Chinese
Development Languages: Java, JavaScript, PHP, iOS, Android

Strength: Planning, Design, Marketing, Project Management, Development, Testing
Languages: Japanese, English
Development Languages: Ruby, Java, PHP, HTML/CSS, JavaScript, Swift, Objective C

JAPAN

TOKYO/OSAKA/MATSUE/FUKUOKA

about me

Matsue.rb

message from Matz

```
# # video
# # src = images/matz.mp4
```

agenda

- we will learn how to start to make IoT product with Ruby
- assumed attendees are software programmers
- ESP32 microcontroller as the platform
- mruby/c (and C) as the firmware language
- we will iterate some combinations of lecture and hands-on

agenda

- we have 3 to 4 hours 😨
  - no worry, we will take several breaks

a short break

enquête

enquête

✋ please raise your hand ✋
✋ if you are a ✋
✋ firmware programming **newbie** ✋

enquête

✋ please raise your hand ✋
✋ if you don't have ✋
✋ **any** experience of mruby ✋

enquête

mruby/c

✋ please raise your hand ✋
✋ if you don't have ✋
✋ **much** experience of C language ✋

setup your laptop

- we have to install ESP-IDF and some dependent tools in order to develop mruby/c firmwares for ESP32
- the most important thing will be **USB**. we will write our firmware into ESP32 through USB cable

setup your laptop - Linux or macOS

- using **Linux distributions** or **macOS** (as a host machine) is the easiest way
  - less USB problem
- I'm not sure but docker will not work because of USB problem

setup your laptop - Windows

- you can choose both **Windows Subsystem for Linux** (WSL) and **MSYS2**
  - I recommend you to use WSL if your OS is Windows10 (64 bit) as compiling on WSL is much faster than MSYS2
- **WSL**
  - you should use WSL if your OS is 64bit of Win10 Pro

setup your laptop - Windows

- **MSYS2**
  - strongly recommended of using zipped one which Espressif Systems maintenances
  - besides, note that only 32 bit version of MSYS2 is available regardless of whether your Windows is 64 bit or 32 bit
- Docker for Windows
  - it appears not to work so far
  - but please tell me if it works

setup your laptop - VM

- I (hasumi) use Linux Mint with VirtualBox on Windows 10 Professional
    - but some people say that virtual environments tend to have problems around USB

setup your laptop

more information on

https://github.com/hasumikin/IoT_workshop

setup your laptop

please tell me if you have any doubt
during the hands-on

Hands On 01

Hello mruby/c World!

open the URL
github.com/hasumikin/IoT_workshop

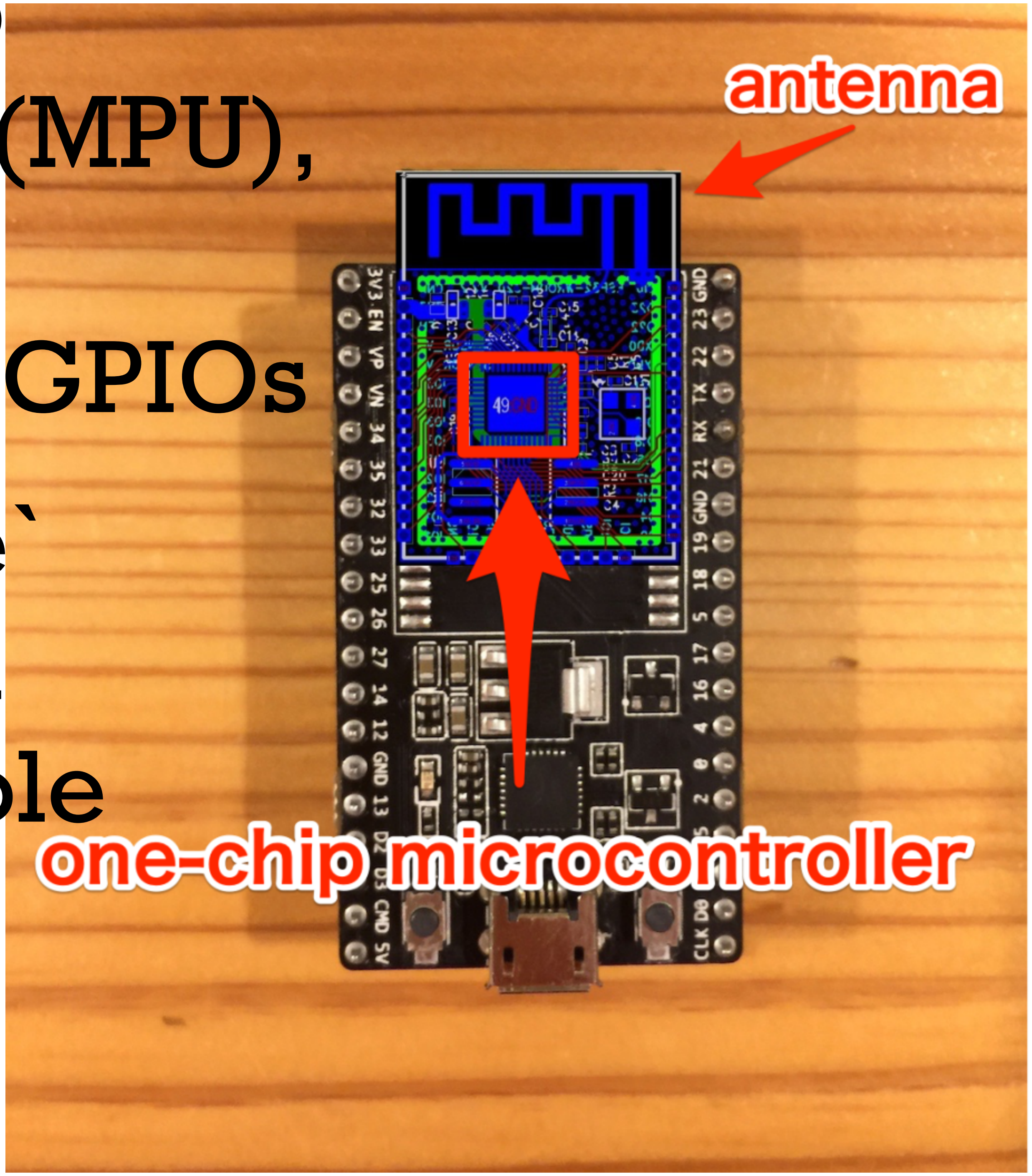
and find the link
Hands on 01

what is microcontroller?

- if you are not familiar with microcontroller, this section is very important to grab overview what we do in this workshop

one-chip microcontroller

- a single IC chip consists of CPU(MPU), RAM, ROM and programmable GPIOs
- `programmable` means user can configure the role of the pins

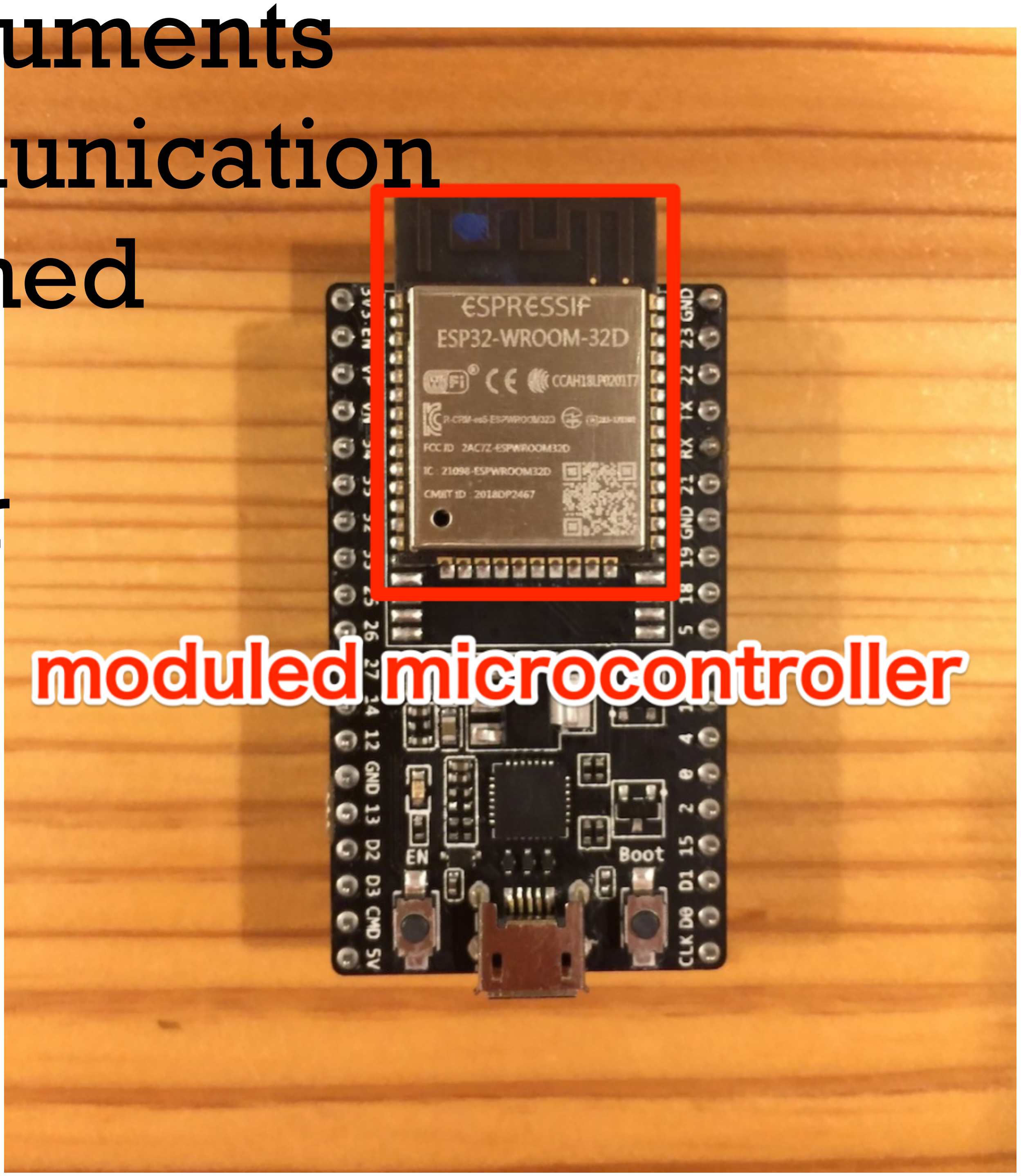


antenna

one-chip microcontroller

moduled microcontroller

- additional instruments like WiFi communication module combined with one-chip microcontroller



moduled microcontroller

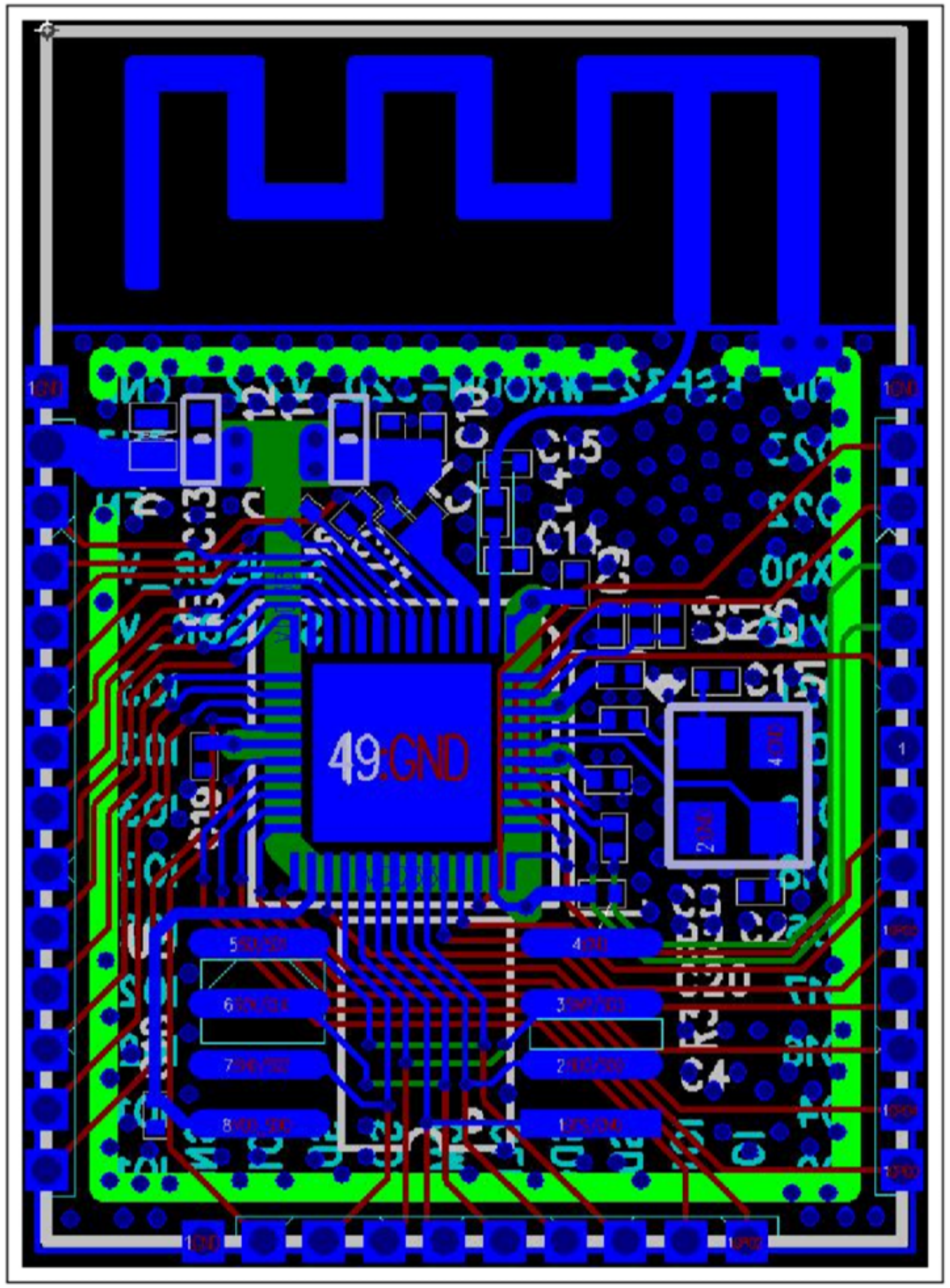
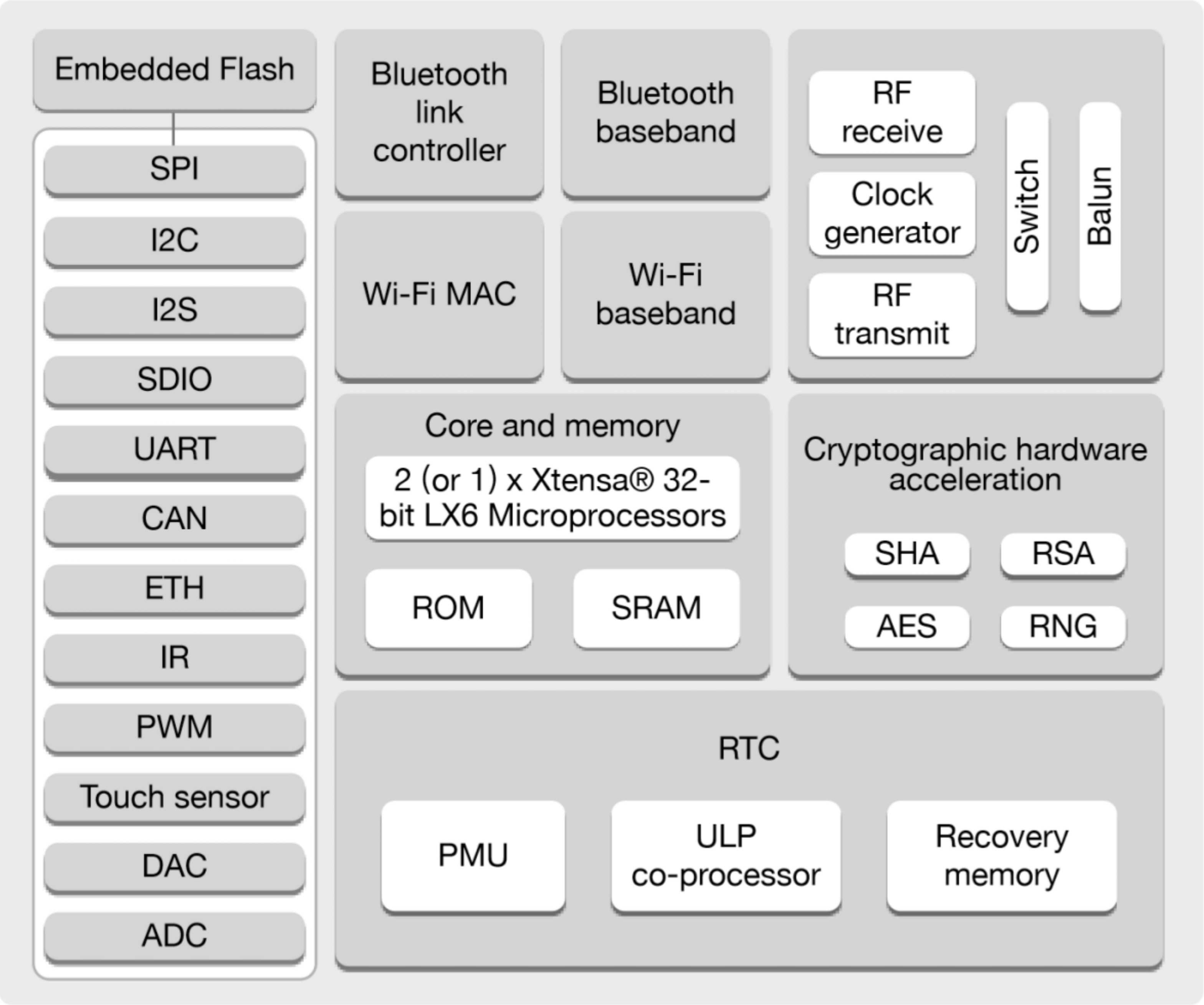


Figure 11: ESP32 PCB Layout



Embedded Flash

SPI

I2C

I2S

SDIO

UART

CAN

ETH

IR

PWM

Touch sensor

DAC

ADC

Bluetooth link controller

Bluetooth baseband

RF receive

Clock generator

RF transmit

Switch

Balun

Wi-Fi MAC

Wi-Fi baseband

Core and memory

2 (or 1) x Xtensa® 32-bit LX6 Microprocessors

ROM

SRAM

Cryptographic hardware acceleration

SHA

RSA

AES

RNG

RTC

PMU

ULP co-processor

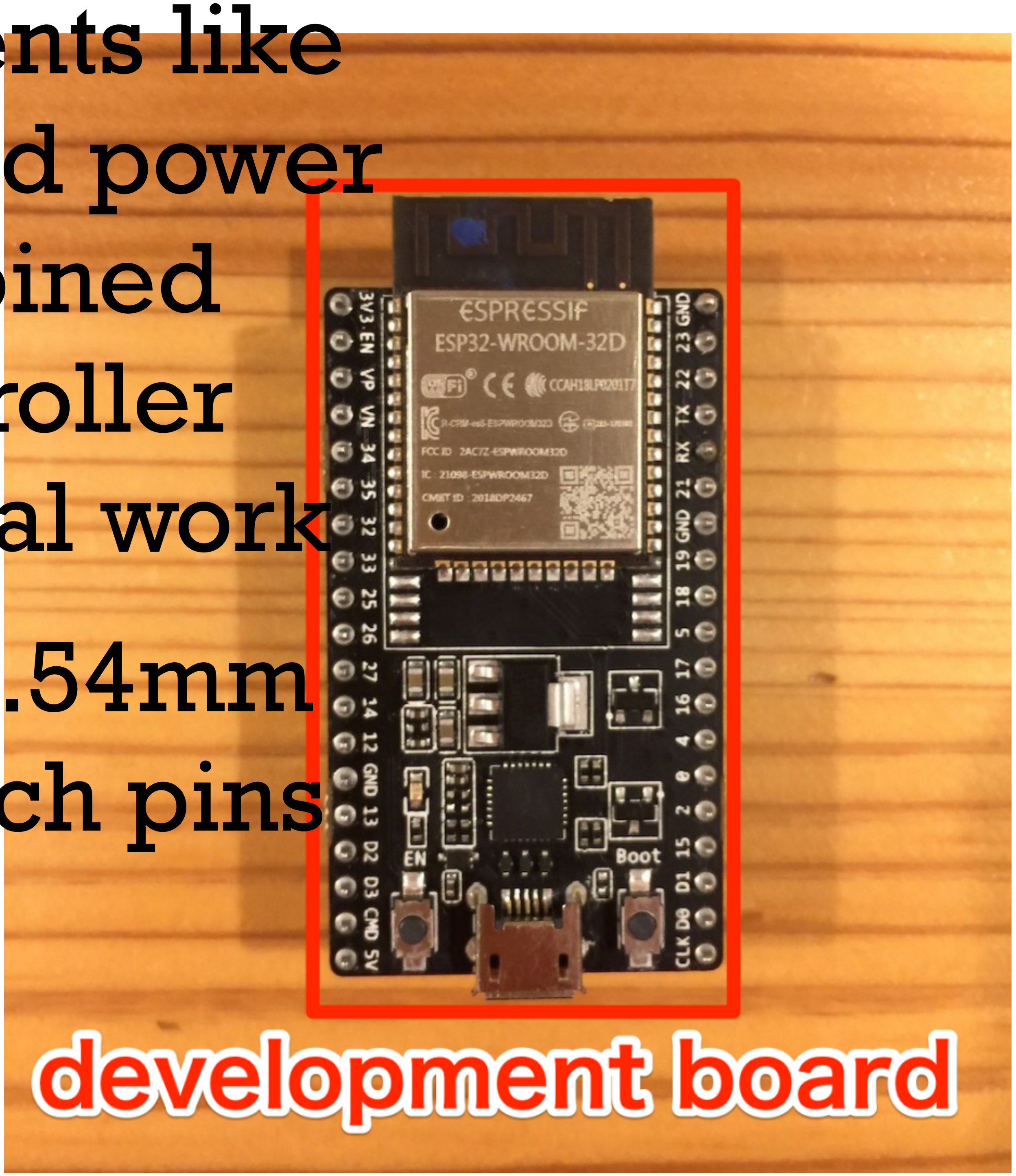
Recovery memory

Figure 1: Functional Block Diagram

"esp32_hardware_design_guidelines_en.pdf esp32_datasheet_en.pdf"

development board (devkit)

- useful equipments like USB adaptor and power regulator combined with microcontroller for experimental work
- generally has 2.54mm (=1/10inch) pitch pins to be fit with breadboard



what is microcontroller?

- what we call as \`microcontroller\` depends on the situation
- I call the development board as \`microcontroller\` in this workshop
- you may have to treat \`one-chip microcontroller\` as \`microcontroller\` if you plan for producing an IoT hardware

Hands On 02

Hello ESP32 World!

open the URL
github.com/hasumikin/IoT_workshop

and find the link
Hands on 02

a short break

peripherals

- **peripheral** is an important concept of microcontroller
- high-end microcontroller has rich peripherals and low-end one has less

peripherals

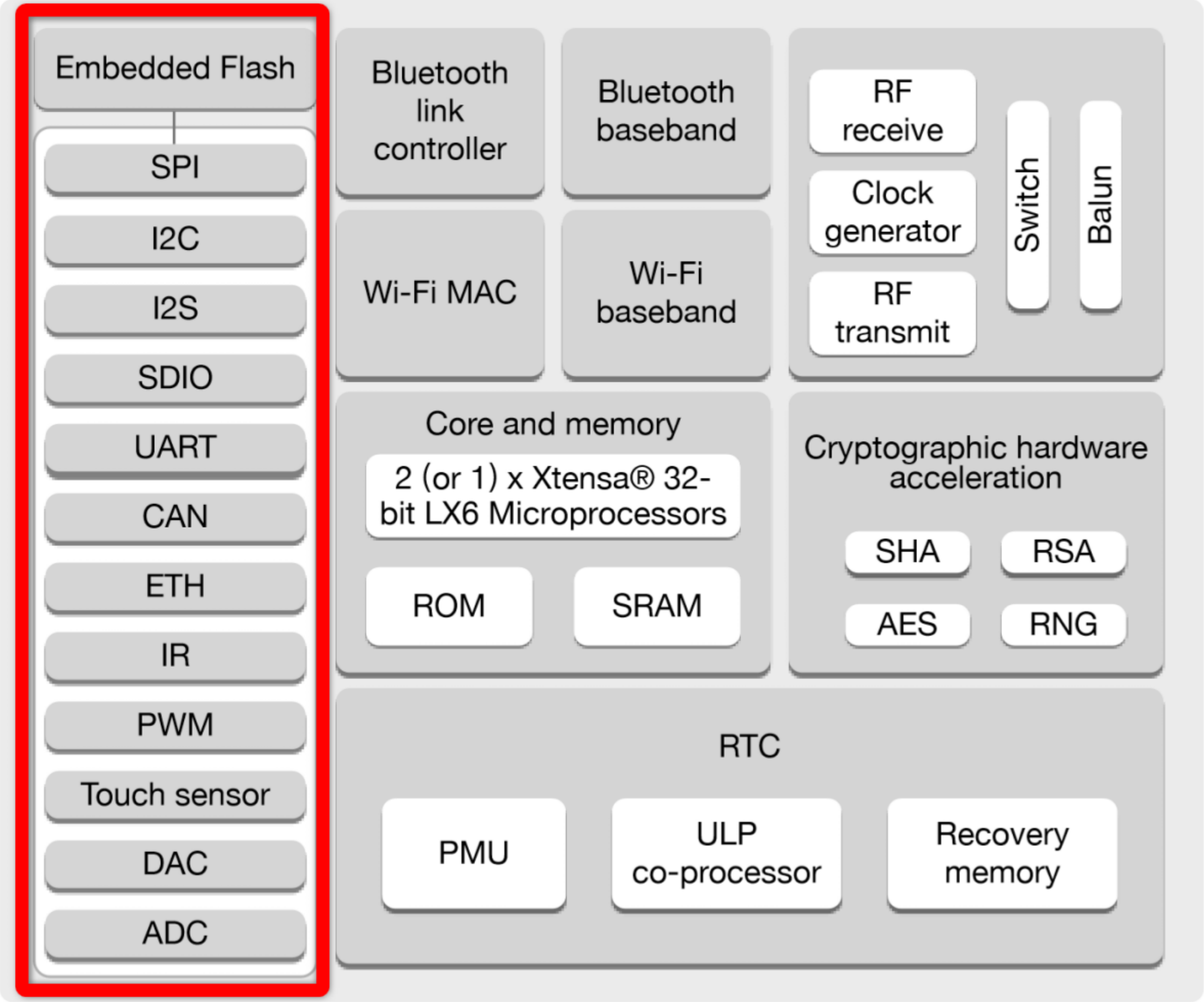


Figure 1: Functional Block Diagram

"esp32_datasheet_en.pdf"

peripherals

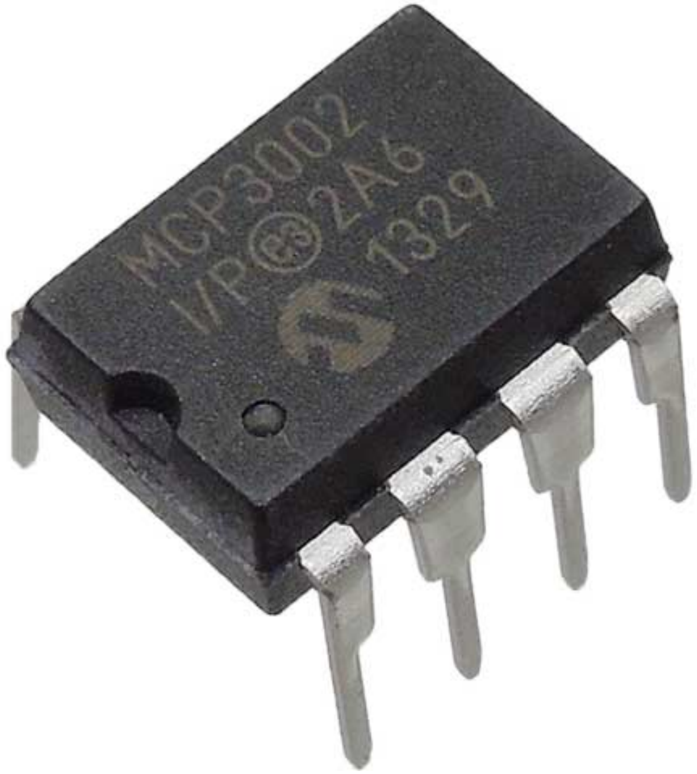
- GPIO (General Purpose Input/Output)
  - GPIO is a defining characteristic of microcontroller
  - GPIO basically has values of only 0 and 1 (digital value)
    - analog value will be mentioned later
  - usecases of Input:
    - switch as an user interface, getting sensor value
  - usecases of Output:
    - LED as a display, sending message to modem

peripherals

- ADC (analog to digital converter) and DAC (vice versa)
  - ADC converts analog value such as microphone input into digital value that computer can deal with
  - DAC converts digital value such as sound data of MP3 into analog output in order to play back the music on loud speaker

peripherals

- Raspberry Pi does neither have ADC nor DAC
- we can add an independent ADC part if we need it



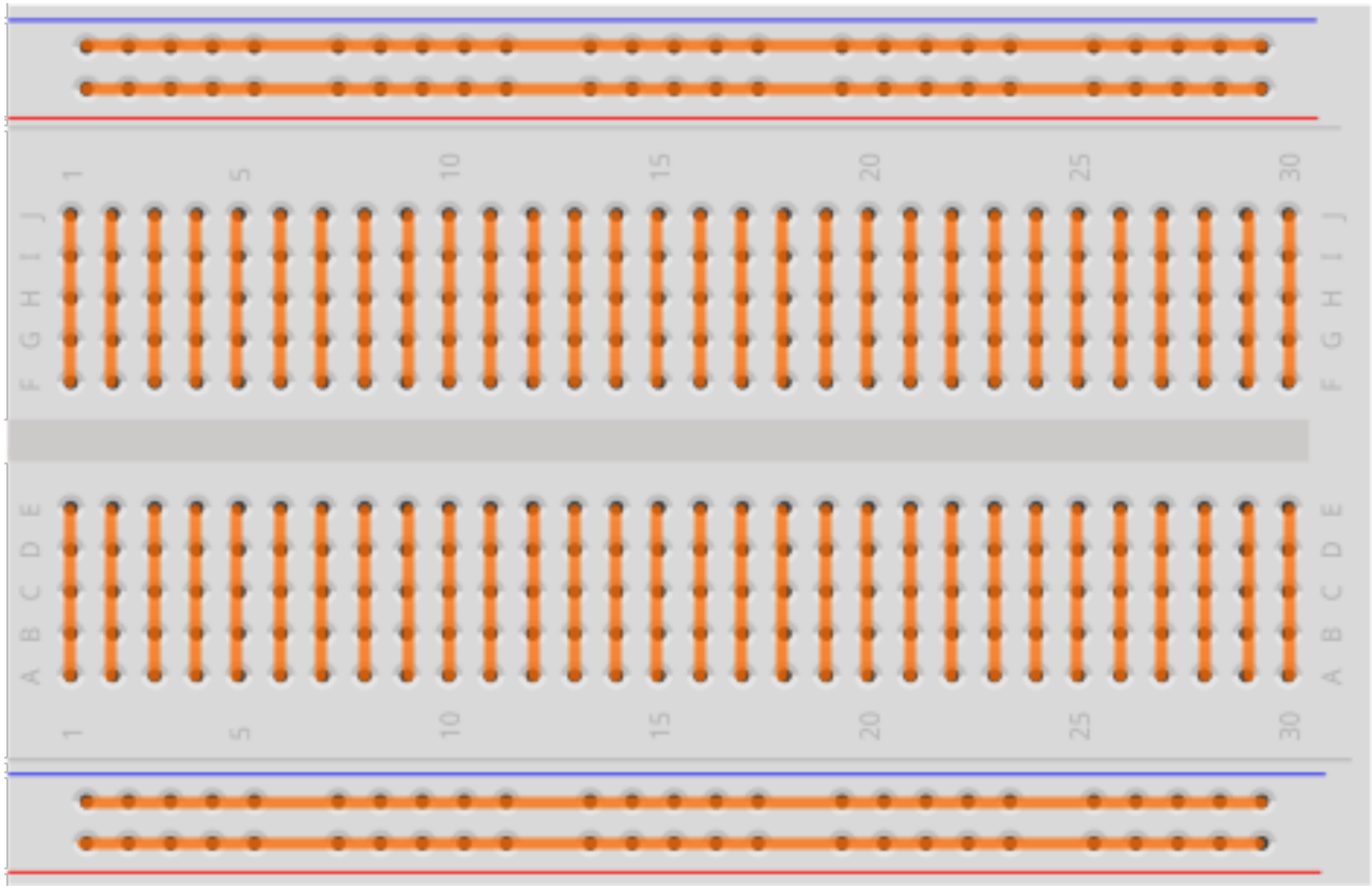
from "http://akizukidenshi.com/catalog/g/gI-02584/"

today's parts

- breadboard
- resistor
- LED
- thermistor

breadboard (protoboard)

- wired internally by 2.54 mm pitch so that we can experiment without soldering

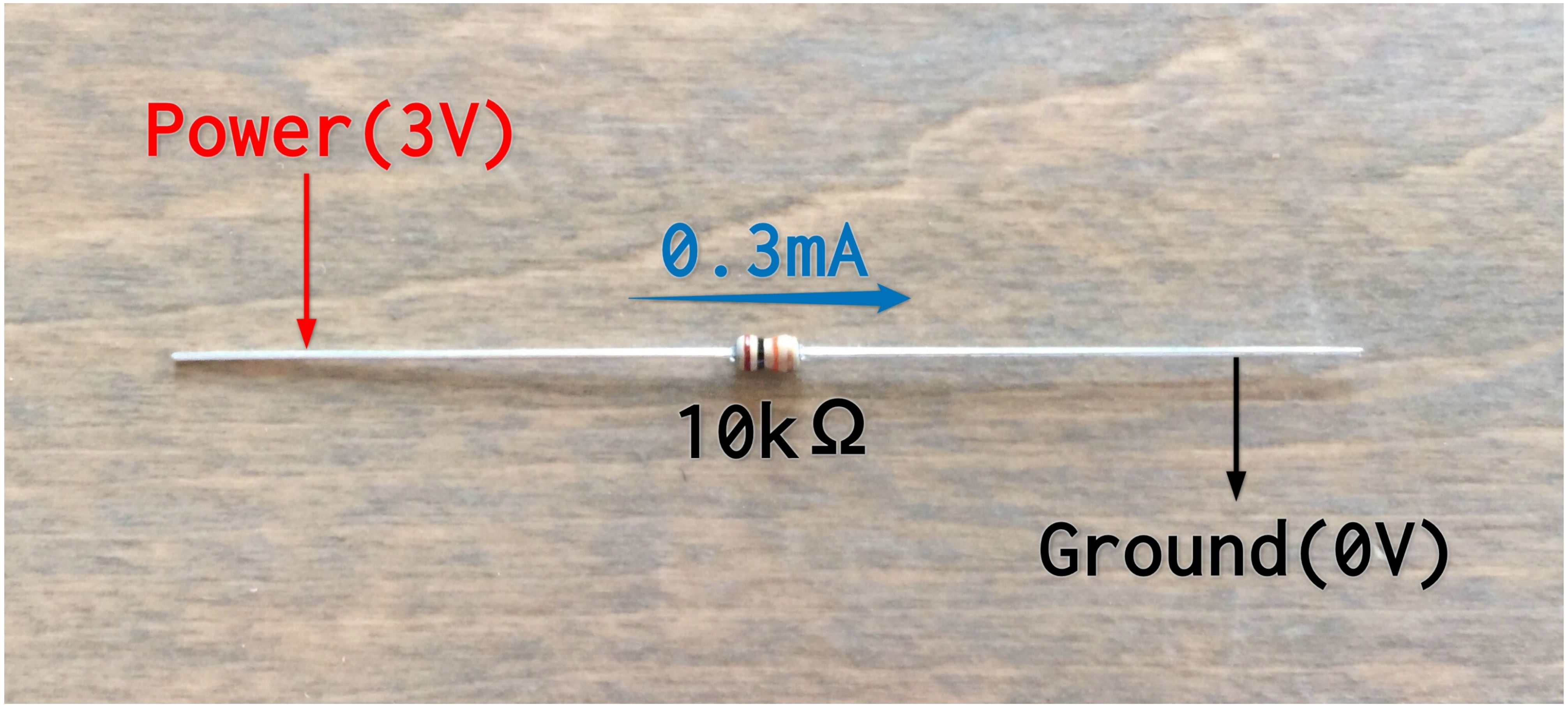


from "https://ht-deko.com/arduino/breadboard.html"

registor & Ohm's law

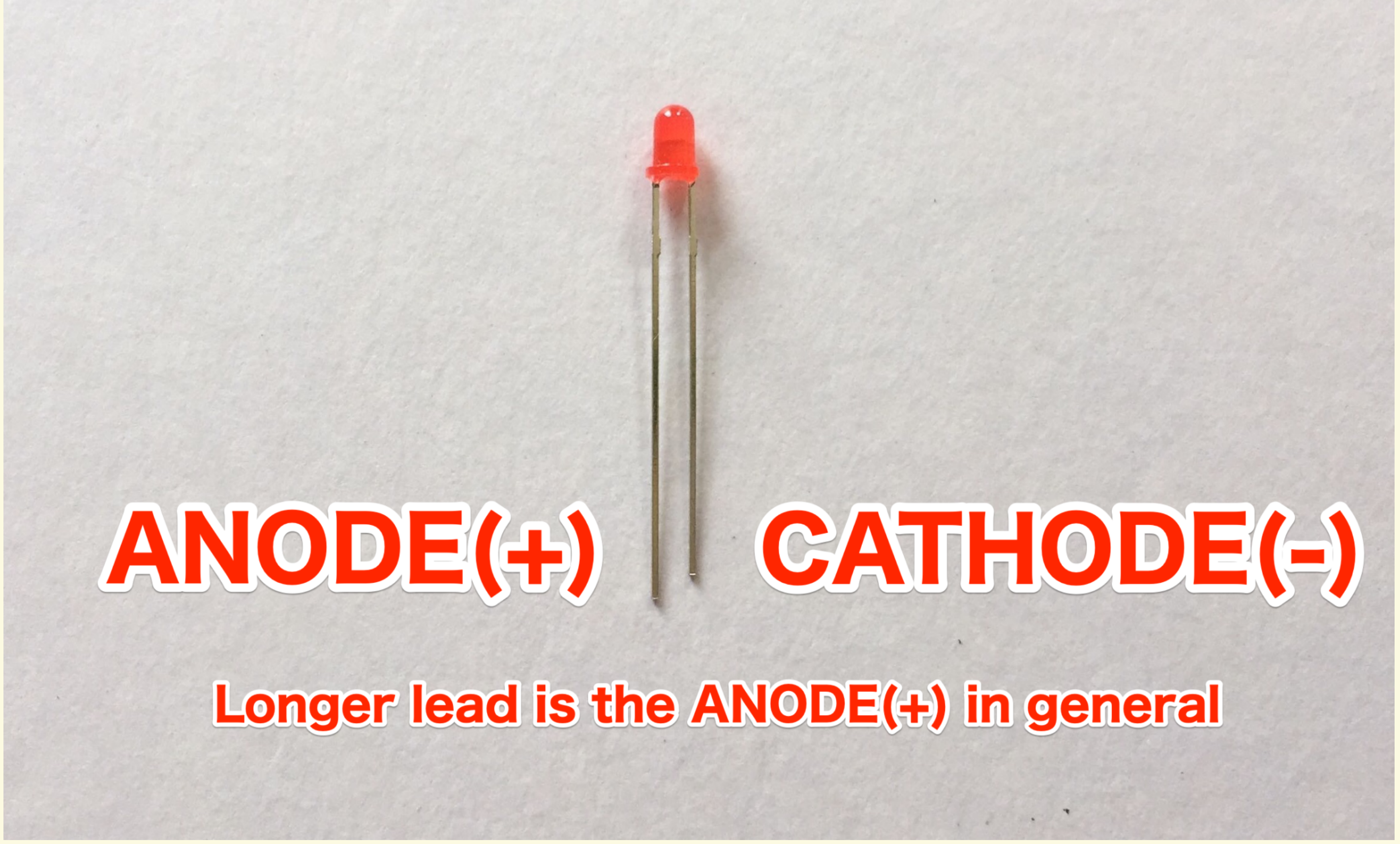
- if the voltage across the both ends of 10kOhm resistor is 3V, the current will be 0.3mA

3 / 10k = 3 / 10000 = 0.0003A = 0.3mA



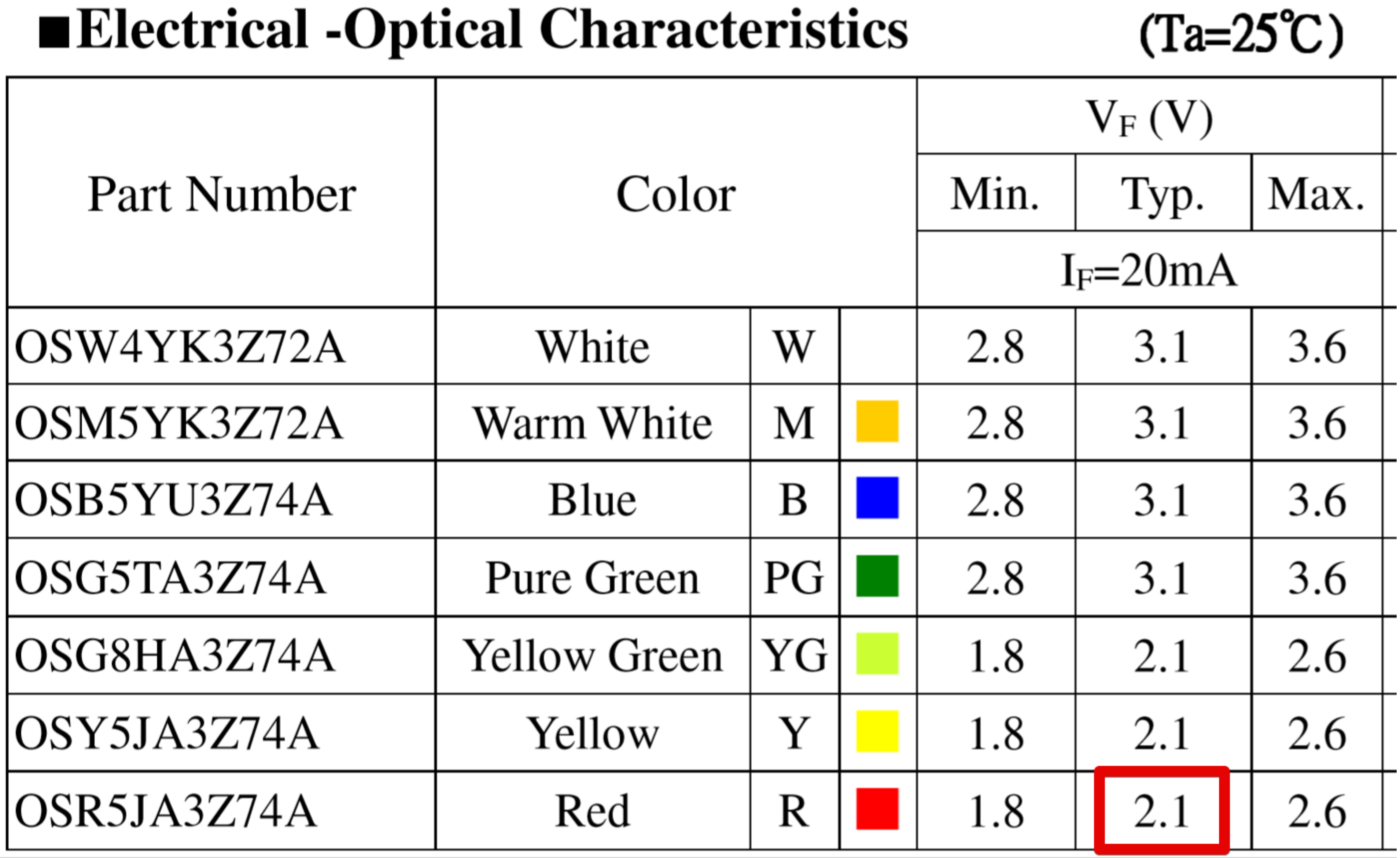
LED

A light-emitting diode (LED) is a semiconductor light source that emits light when current flows through it.



[「WIKIPEDIA」より引用]

LED - datasheet



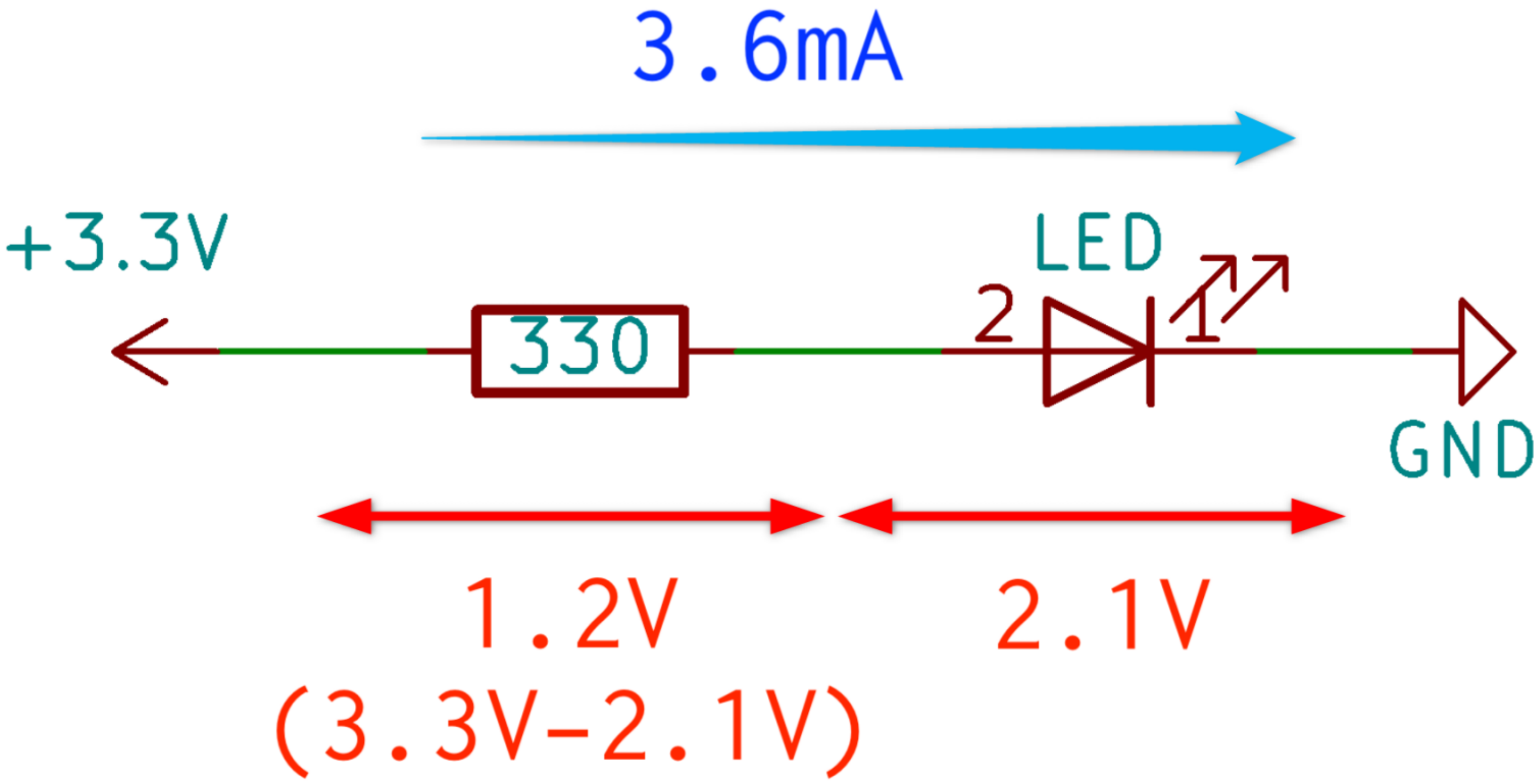
■Electrical -Optical Characteristics (Ta=25℃)

Part Number	Color			V_F (V)		
				Min.	Typ.	Max.
				I_F=20mA		
OSW4YK3Z72A	White	W		2.8	3.1	3.6
OSM5YK3Z72A	Warm White	M	🟧	2.8	3.1	3.6
OSB5YU3Z74A	Blue	B	🟦	2.8	3.1	3.6
OSG5TA3Z74A	Pure Green	PG	🟩	2.8	3.1	3.6
OSG8HA3Z74A	Yellow Green	YG	🟩	1.8	2.1	2.6
OSY5JA3Z74A	Yellow	Y	🟨	1.8	2.1	2.6
OSR5JA3Z74A	Red	R	🟥	1.8	2.1	2.6

from "http://akizukidenshi.com/download/ds/optosupply/OSXXXX3Z74A_VER_A1.pdf"

LED & Ohm's law

(3.3 - 2.1) / 330 = 0.0036A = 3.6mA



LED & GPIO

- small LED can be lighten by GPIO
- but instruments like huge LED which requires high current can not be driven even if its nominal voltage is less than 3.3V
- because microcontroller has some limit of supplying amount of electric current
- incorrect usage may break your microcontroller

Hands On 03

Blinking LED

open the URL
github.com/hasumikin/IoT_workshop

and find the link
Hands on 03

hint: you should use a **blue** resistor

a short break

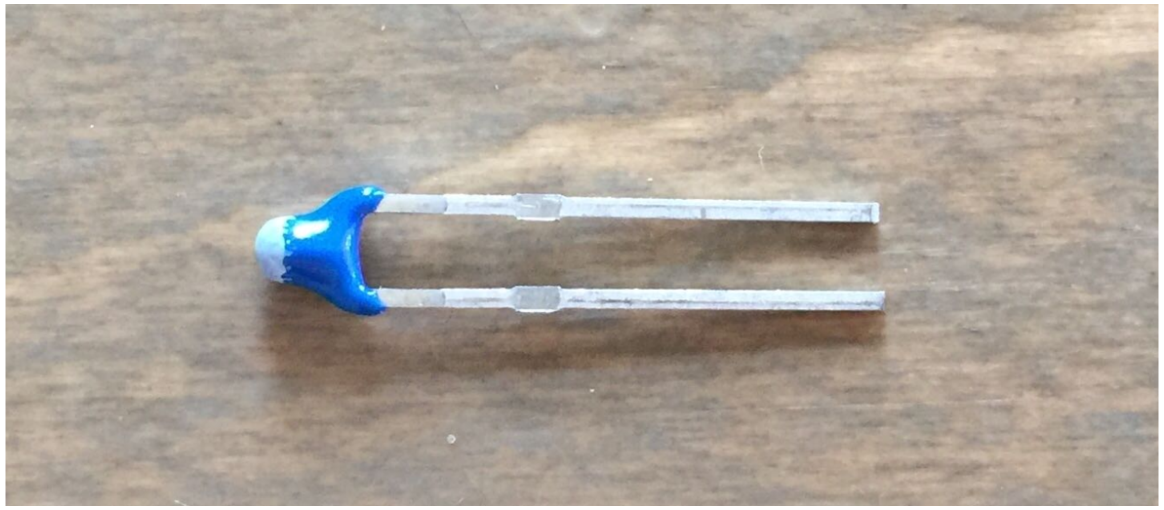
today's parts (again)

- breadboard
- resistor
- LED
- thermistor

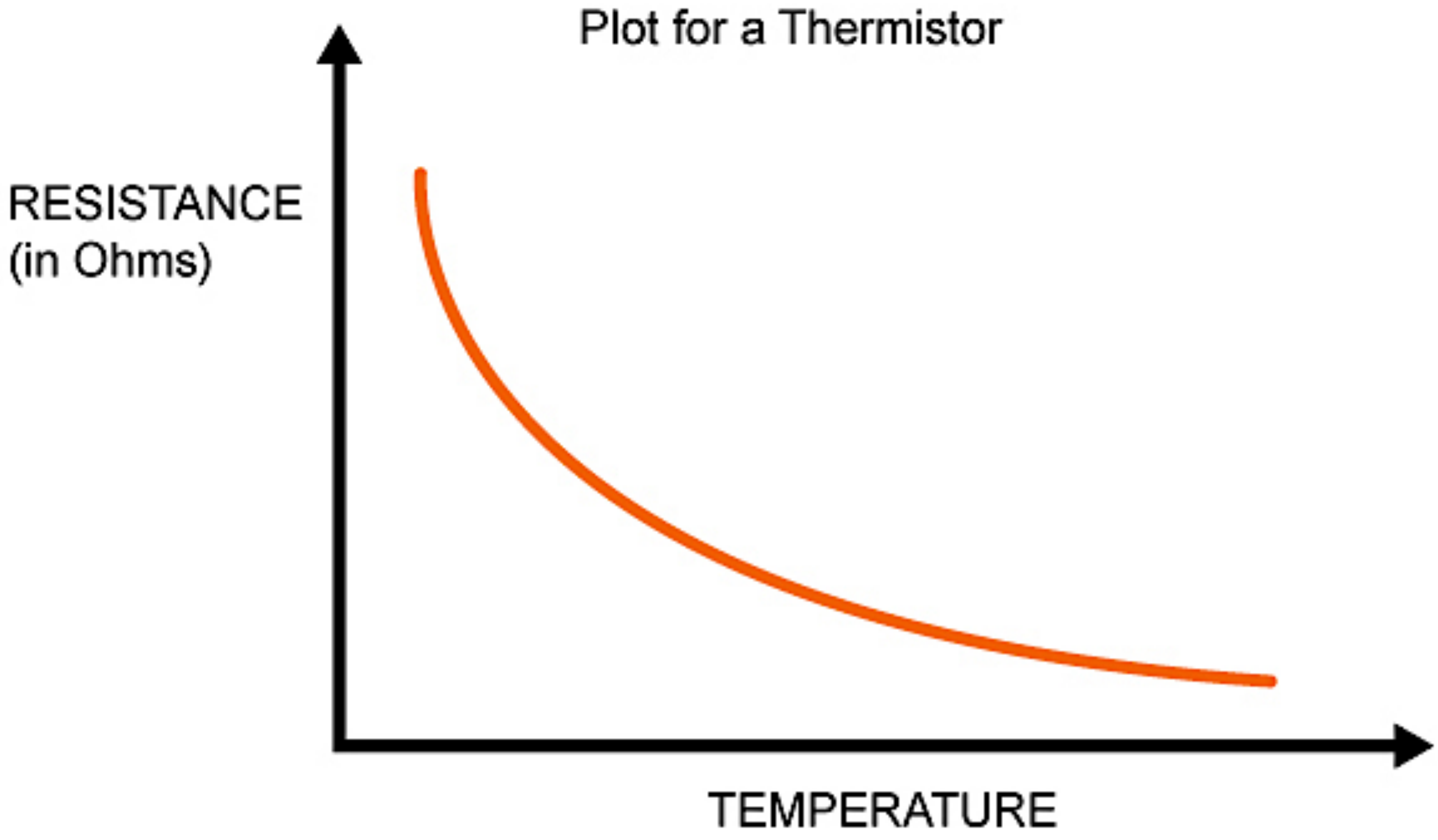
thermistor

> *A thermistor is a type of resistor whose resistance is dependent on temperature, more so than in standard resistors.*

[「WIKIPEDIA」より引用]



thermistor



from "https://www.allaboutcircuits.com/projects/measuring-temperature-with-an-ntc-thermistor/"

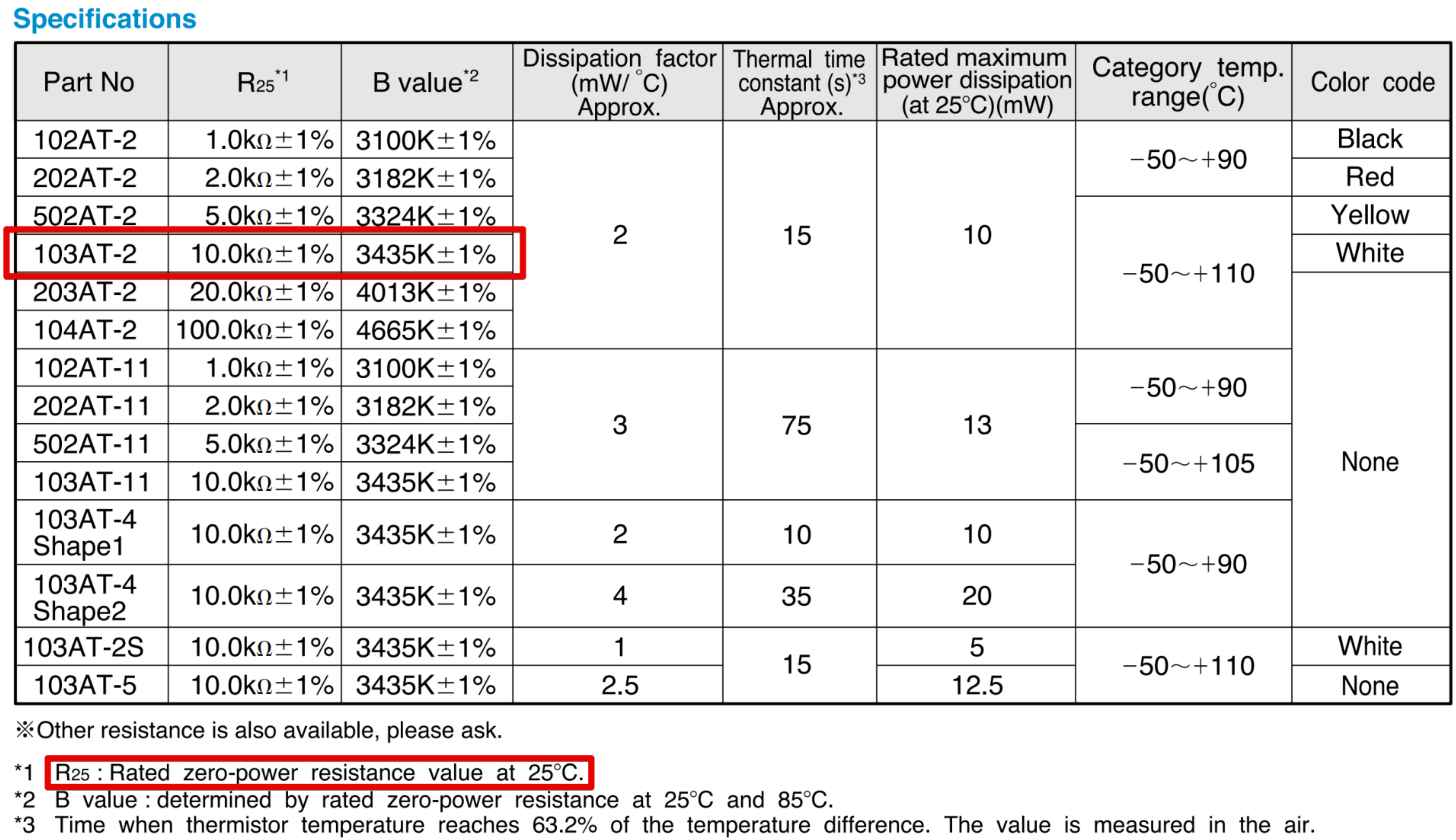
thermistor - approximation

$$R = R_{ref} \times e^{(B(\frac{1}{T+273} - \frac{1}{T_o+273}))}$$

hence,

$$T = \frac{1}{\frac{1}{B}\log\frac{R}{R_{ref}} + \frac{1}{T_o+273}} - 273$$

thermistor - datasheet



Specifications

Part No	R_{25}[*1]	B value[*2]	Dissipation factor (mW/°C) Approx.	Thermal time constant (s)[*3] Approx.	Rated maximum power dissipation (at 25°C)(mW)	Category temp. range(°C)	Color code
102AT-2	1.0kΩ±1%	3100K±1%	2	15	10	−50～+90	Black
202AT-2	2.0kΩ±1%	3182K±1%					Red
502AT-2	5.0kΩ±1%	3324K±1%				−50～+110	Yellow
103AT-2	10.0kΩ±1%	3435K±1%					White
203AT-2	20.0kΩ±1%	4013K±1%					None
104AT-2	100.0kΩ±1%	4665K±1%					
102AT-11	1.0kΩ±1%	3100K±1%	3	75	13	−50～+90	
202AT-11	2.0kΩ±1%	3182K±1%					
502AT-11	5.0kΩ±1%	3324K±1%				−50～+105	
103AT-11	10.0kΩ±1%	3435K±1%					
103AT-4 Shape1	10.0kΩ±1%	3435K±1%	2	10	10	−50～+90	
103AT-4 Shape2	10.0kΩ±1%	3435K±1%	4	35	20		
103AT-2S	10.0kΩ±1%	3435K±1%	1	15	5	−50～+110	White
103AT-5	10.0kΩ±1%	3435K±1%	2.5		12.5		None

※Other resistance is also available, please ask.

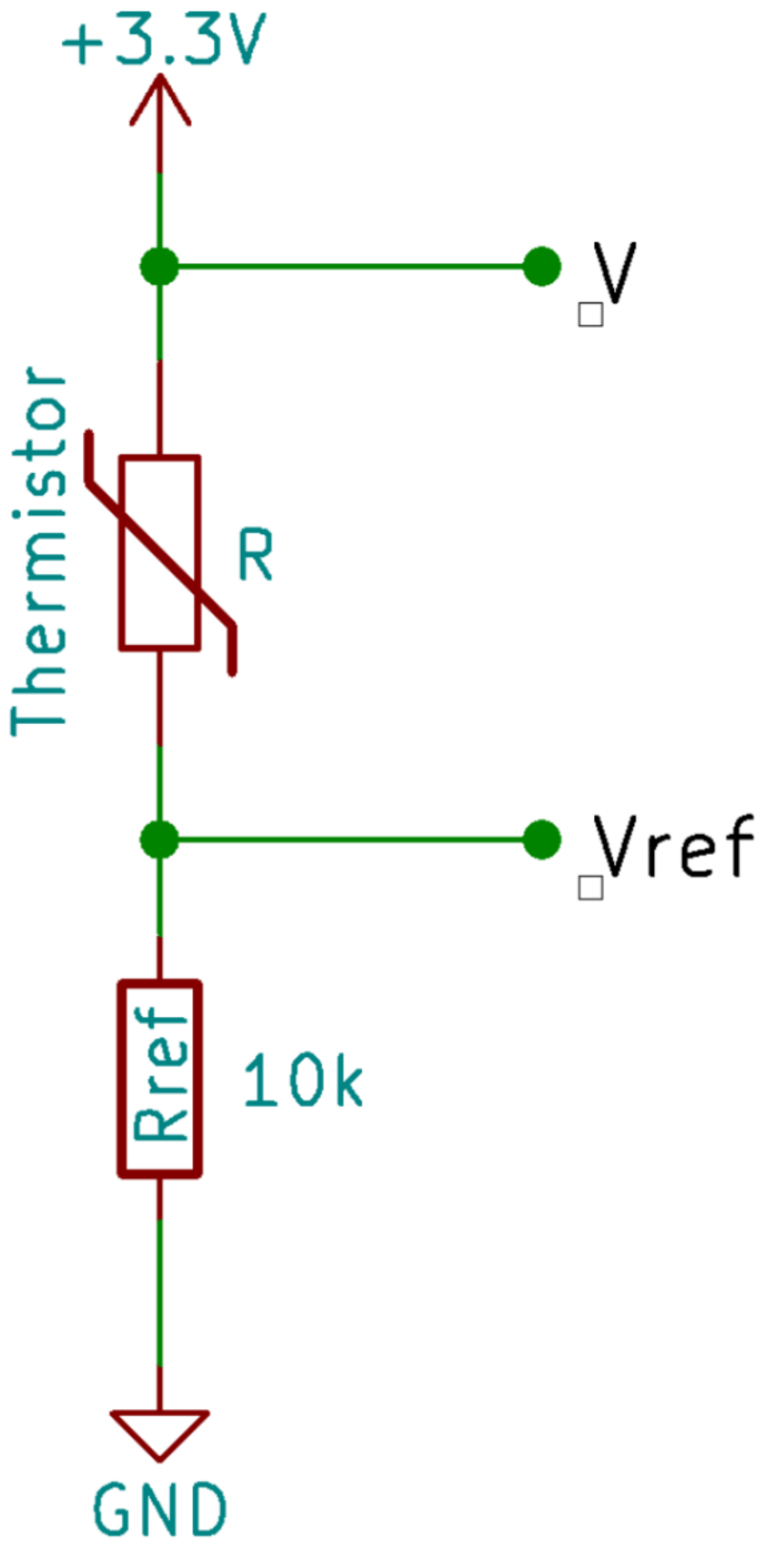
*1 R_{25} : Rated zero-power resistance value at 25°C.
*2 B value : determined by rated zero-power resistance at 25°C and 85°C.
*3 Time when thermistor temperature reaches 63.2% of the temperature difference. The value is measured in the air.

from "https://www.mouser.com/datasheet/2/362/semitec_atthermistor-1202913.pdf"

thermistor - approximation

```
# this is CRuby
include Math

# according to the datasheet
B    = 3_435  # from datasheet
To   = 25     # from datasheet
Rref = 10_000 # arbitrary but fixed

def temperature(r)
  1.to_f / ( 1.to_f / B * log(r.to_f / Rref)
    + 1.to_f / (To + 273) ) - 273
end

# if resistance of thermistor is 12kOhm
puts temperature(12_000)

=> 20.35988998853088
```

thermistor & Ohm's law



$$R = \frac{V - V_{ref}}{\frac{V_{ref}}{R_{ref}}}$$

Hands On 04

Taking temperature

open the URL
github.com/hasumikin/IoT_workshop

and find the link
Hands on 04

hint: you should use a **brown** resistor

a short break

what is mruby/c?

- github.com/mrubyc/mrubyc
- yet another implementation of mruby
- `/c` symbolizes compact, concurrent and capability
- especially dedicated to one-chip microcontroller

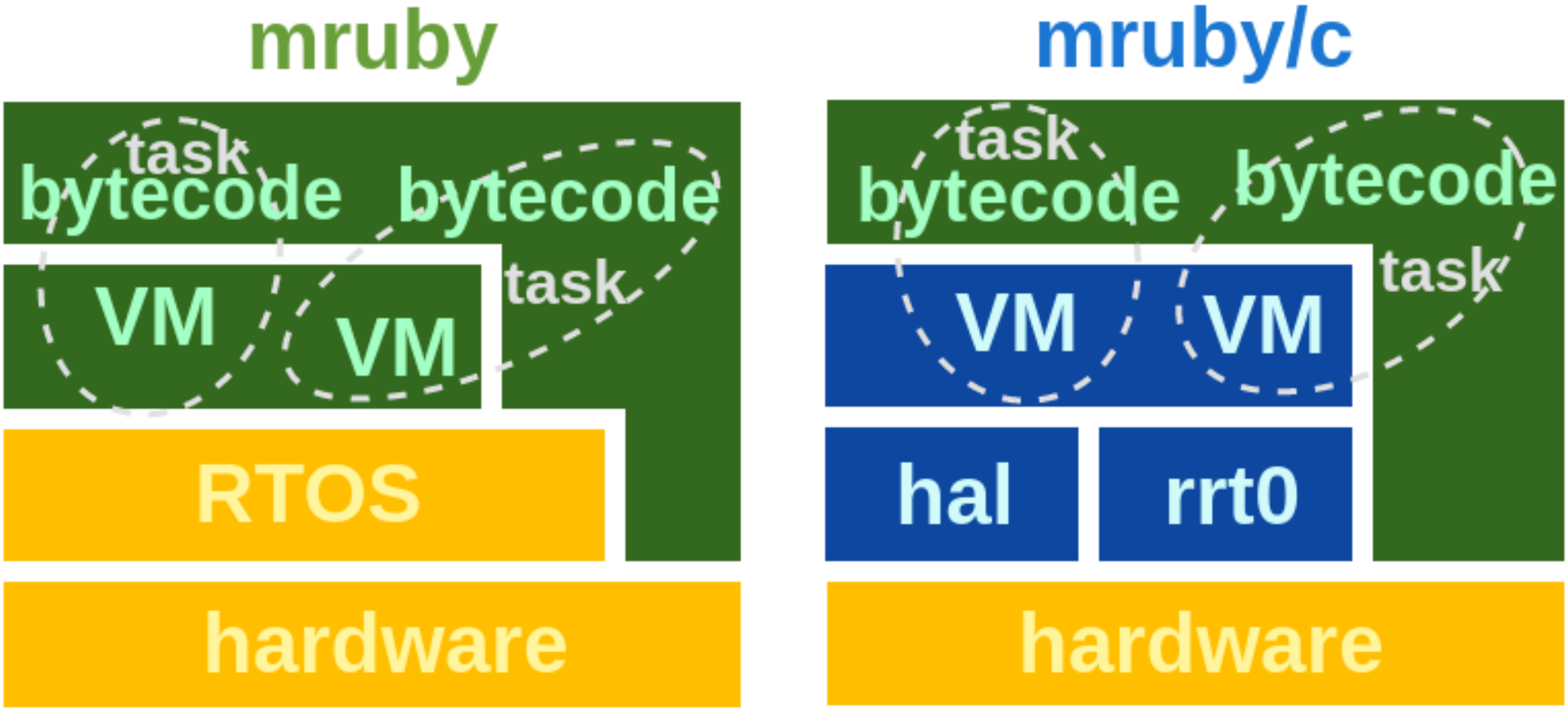
mruby and mruby/c

mruby	mruby/c
v1.0.0 in Jan 2014	v1.0 in Jan 2017
for general embedded software	for one-chip microcontroller
RAM < 400KB	RAM < 40KB

- sometimes mruby is still too big to run on microcontroller

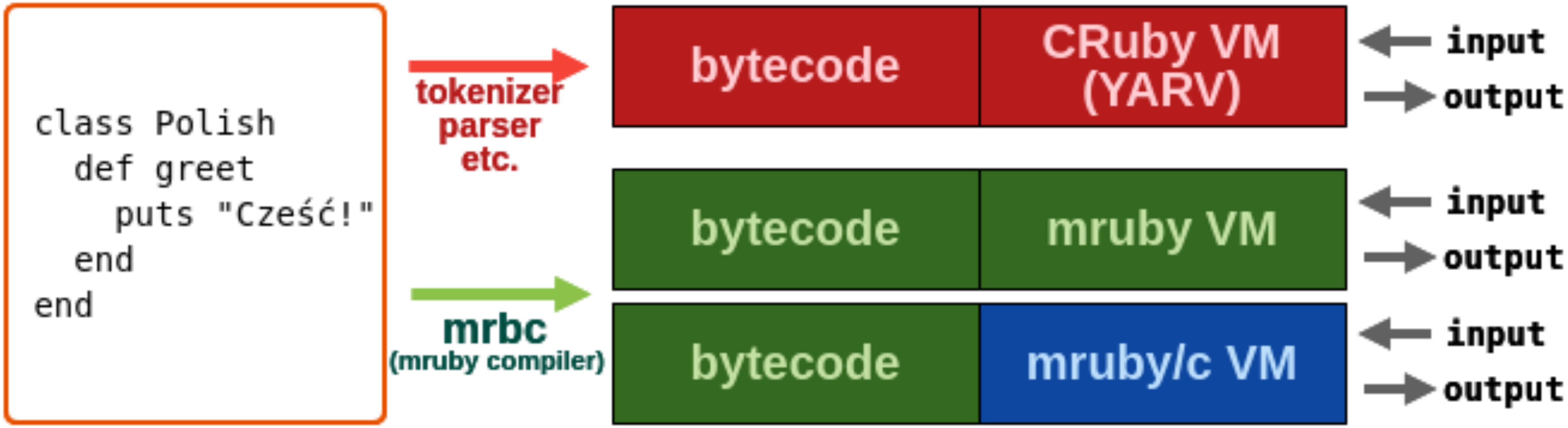
both mruby and mruby/c

- bytecodes are compiled by `mrbc` and VM executes the bytecode



bytecode

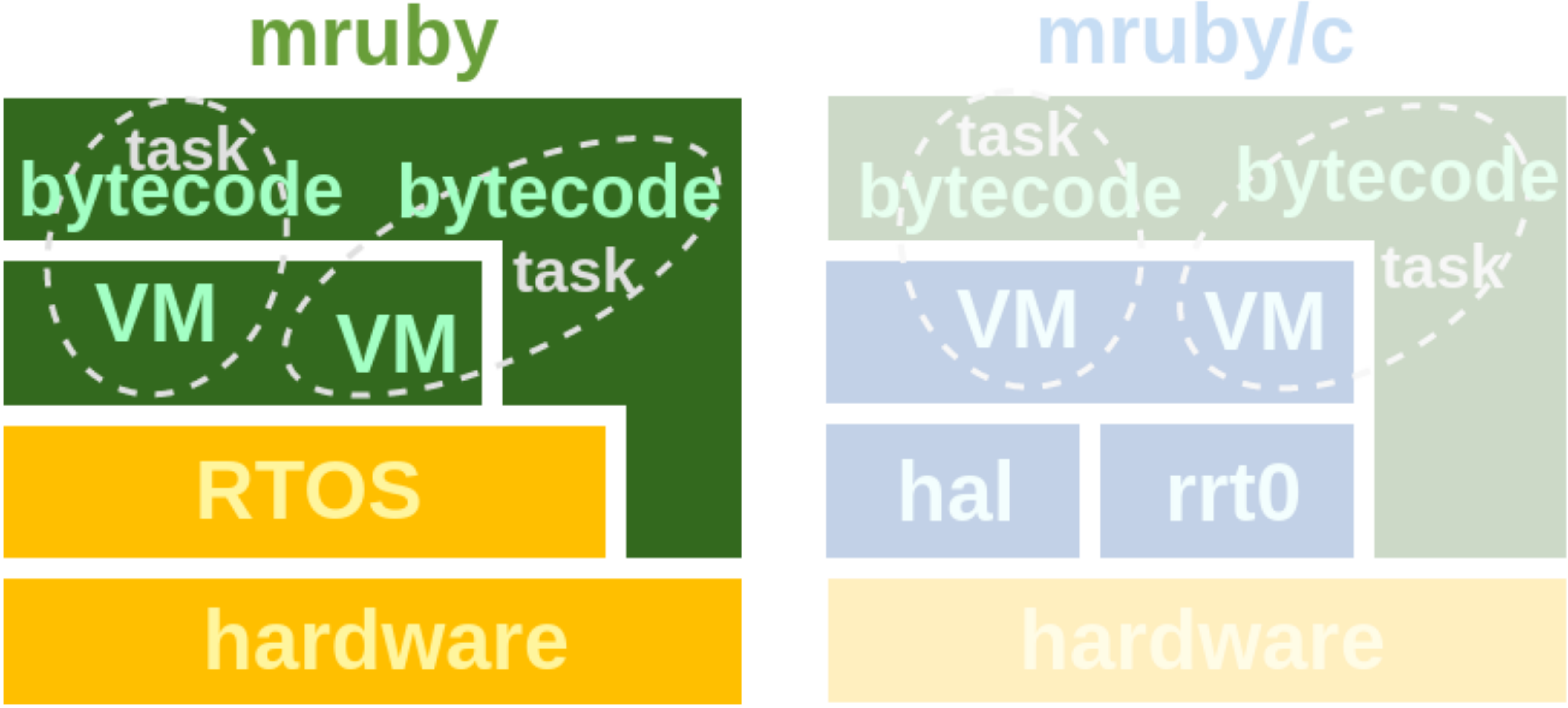
- a kind of intermediate representation
- virtual machine dynamically interprets the bytecode and processes the program



```
class Polish
  def greet
    puts "Cześć!"
  end
end
```

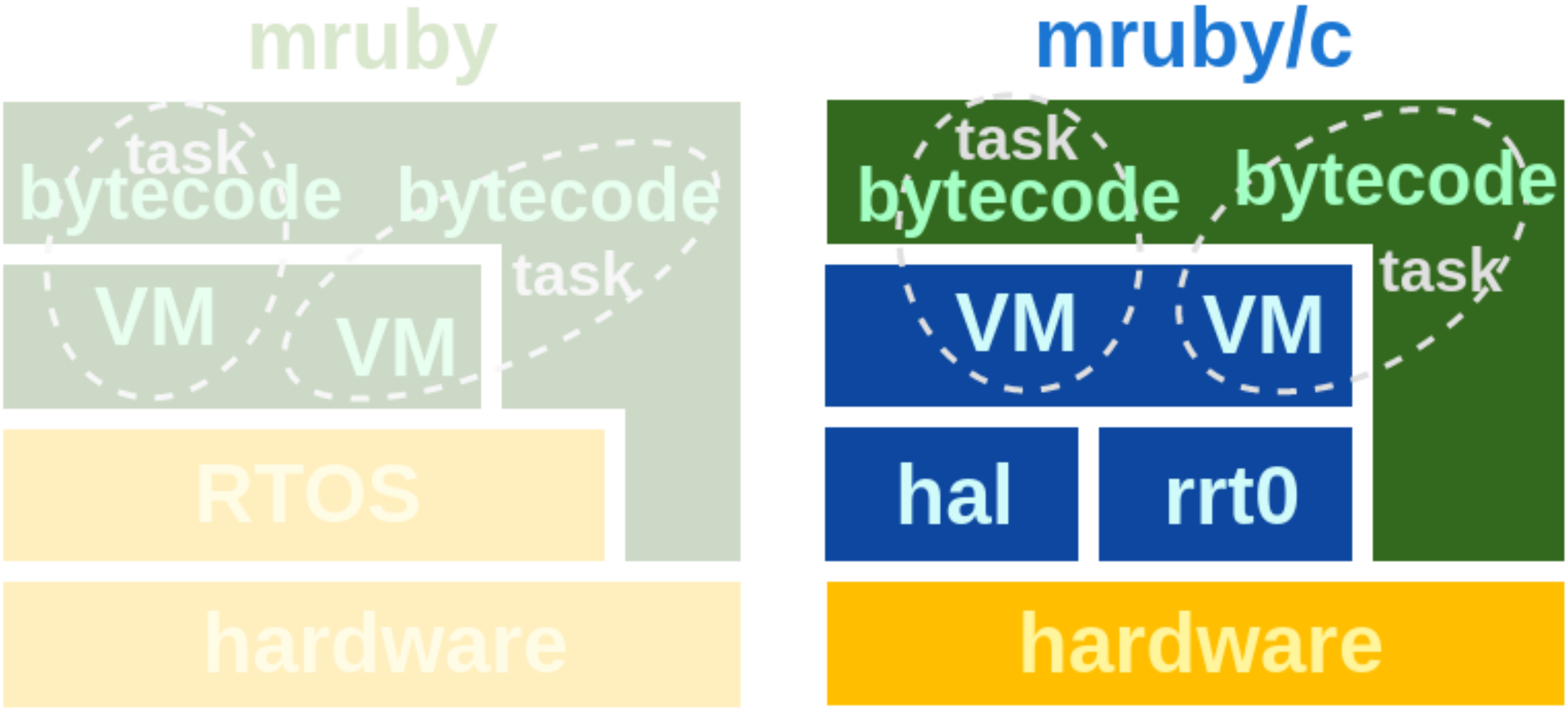
mruby on microcontroller

- RTOS (Real-Time OS) manages mruby VMs. RTOS has features like multi tasking



mruby/c on microcontroller

- mruby/c has its own mechanism to manage the runtime: **rrt0**



mruby/c - virtual machine (VM)

- much smaller than mruby's one
  - that's why mruby/c runs on smaller RAM
- accordingly, mruby/c has **less** functionality than mruby

how **less**?

how **less**? - for example

- mruby/c doesn't have module, hence there is no Kernel module
- then you must wonder how can you `#puts`?
- in mruby/c, `#puts` is implemented in Object class

how **less**? - for example

- mruby/c doesn't have #send, #eval, nor #method_missing
- moreover, mruby/c neither have your favorite features like TracePoint nor Refinements 😟

how **less**? - actually

- the full list of mruby/c's classes
  - Array, FalseClass, Fixnum, Float, Hash, Math, Mutex, NilClass, Numeric, Object, Proc, Range, String, Symbol, TrueClass, VM

despite the fact,

- no problem in practical use of microcontroller
- as far as IoT go, mruby/c is enough Ruby as I expect
- we can fully develop firmwares with features of mruby/c

Hands On 05

Multi-tasking with mruby/c

open the URL
github.com/hasumikin/IoT_workshop

and find the link
Hands on 05

conclusion

conclusion

All you need is **Ohm's law**

Thank you!