**Taken from a paper:** Hardware, the other way was to choose a height programming language like JAVA, Python or NodeJS. If we choose the first option we must develop as many version as existing OS and hardware, and it will take a lot of time. If we choose the second option, our middleware will be OS independent but we must guarantee the real time capability of the chosen language. We decided to study the second option in this contribution. We started by eliminating JAVA and NodeJS because there isn’t enough supported driver on embedded cards that we use in our robots projects, like (Raspberry PI3, BeagleBone…), and also, the biggest communities that develop application on these embedded card use either C or Python.

[Major Proof that ESP32 has underlying OS runs FreeRTOS and Micropython run as a Task in ESP32. 16k stack for the MicroPython task and 96k Python heap.](https://github.com/micropython/micropython/blob/master/ports/esp32/README.md)7

[ESP32 documentation](https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf)

For multitasking with micropython use [this](https://forum.micropython.org/viewtopic.php?t=3375). import \_thread(this library is used to do that and can be done easily). Simple [example](https://techtutorialsx.com/2017/10/02/esp32-micropython-creating-a-thread/)

**Import \_thread** is the library for thread including in python.

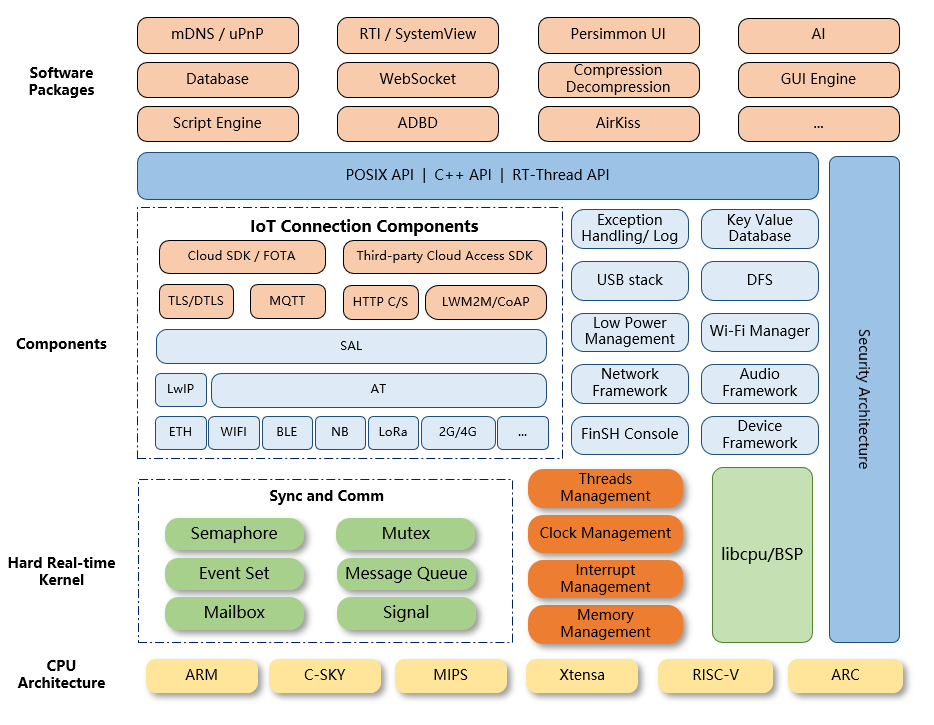


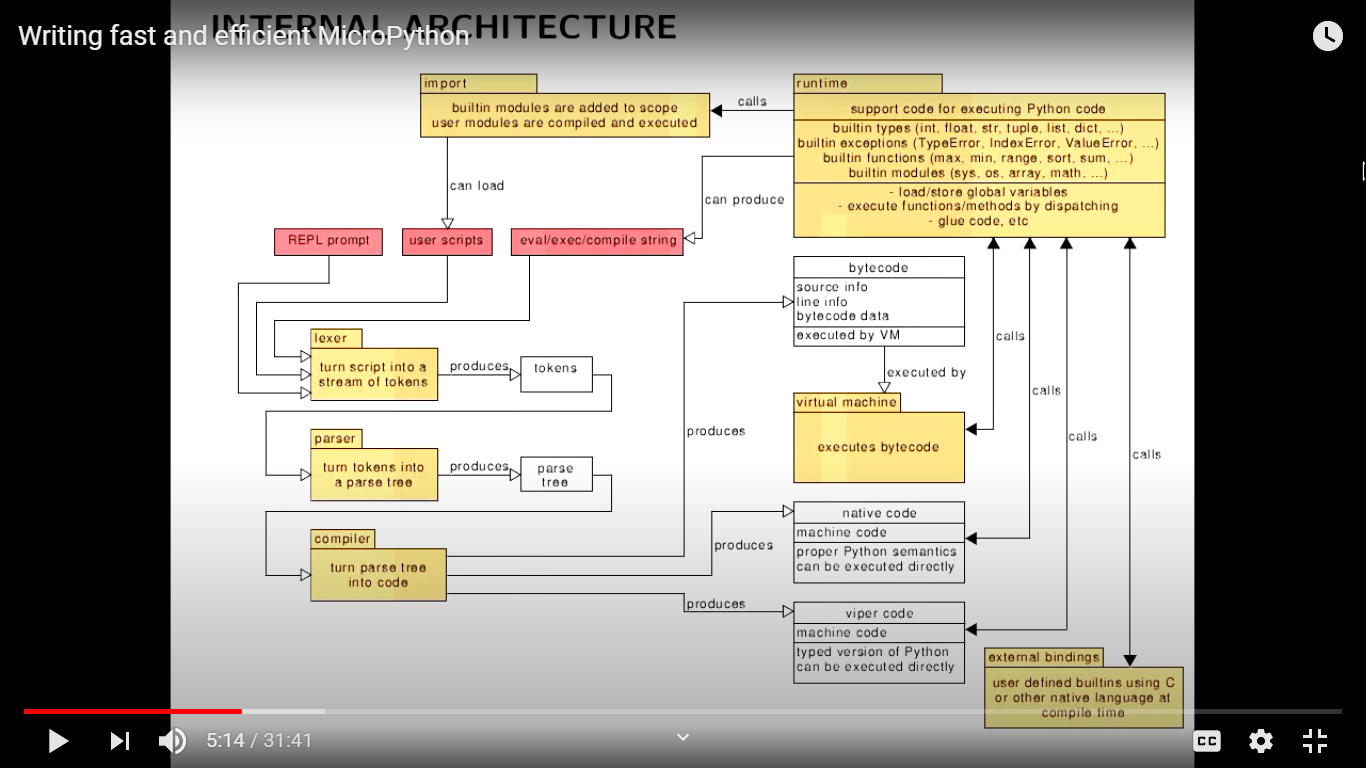
Figure 1: RT-Thread architecture

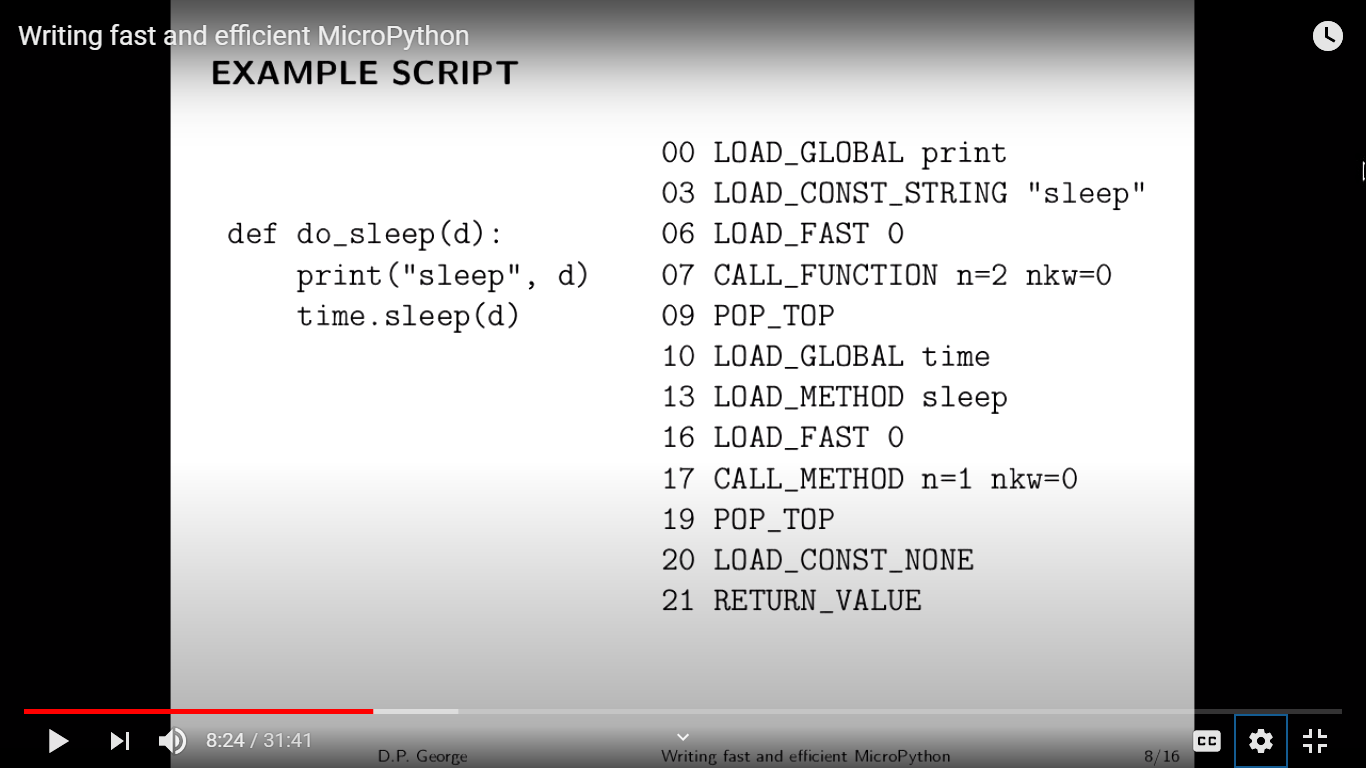
**About Micropython:**

* Micropython can easily run in system which has 16KB of flash memory too.
* Just a single line code you can blink an LED. Designed to run **bare metal** that’s why you get immediate responses. But this bare metal is only for STM32/pyboards and all but for ESP32 it has underlying OS known as FreeRTOS.

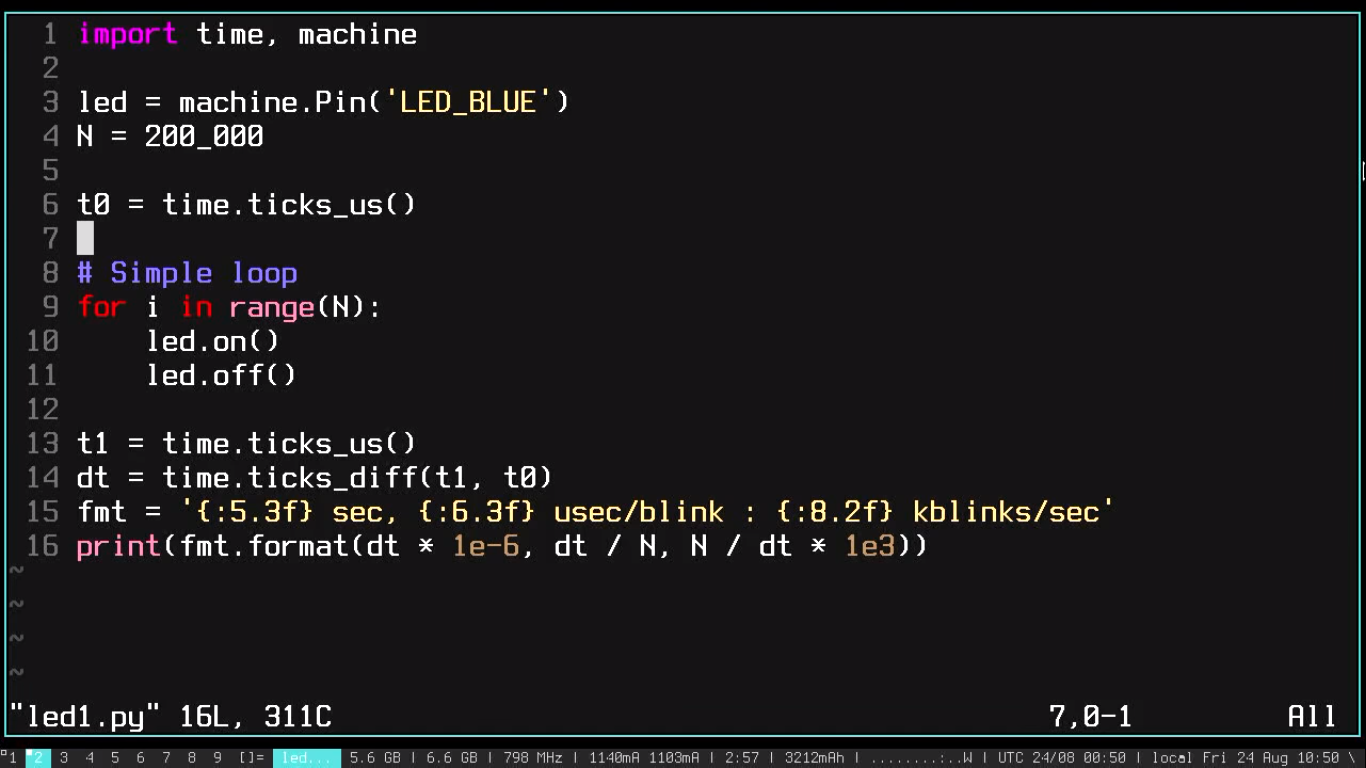
**Execution process in Micropython:**

* Turn script into a stream of token (lexer).
* Turn tokens into a Parse tree (Parser).
* Turn parse tree into code (compiler).
* Source code to bytecode data (bytecode).
* Executes byte code (Virtual machine).
* Machine code.





The above image shows the python code on left and bytecode on the right. This whole function is 22 bytes in memory. To make things work fast use local variables rather than global variables.



The above image tells how to see the time take and crystal oscillator working frequency numbers.

**References:**

<https://github.com/RT-Thread/mpy-snippets/blob/master/docs/MicroPythonPlug-in/README_EN.md> = How to use RT-thread with micropython on visual studio code.

<https://github.com/RT-Thread/mpy-snippets/blob/master/docs/MicroPythonPlug-in/README_EN.md> = This RT-Thread Micropython can be used

<https://www.youtube.com/watch?v=aDXgX0rGVDY> = This guy is doing multithreading with micropython.

<https://melbournemicropythonmeetup.github.io/November-2018-Meetup/> = This meetup has done many videos related to threading with micropython.

<https://itywik.org/2018/10/30/eight-micropython-python-experiments-for-the-esp32/> = Simple experiments with python and sensor integration. You can see this and code. Programming with thread is also done.

<https://github.com/RT-Thread/rt-thread> = Rt-Thread architecture.

[**https://www.quora.com/What-is-bare-metal-programming-in-Embedded-systems#:~:text=%22Bare%20metal%22%20means%20your%20application,considered%20as%20bare%20metal%20programming.**](https://www.quora.com/What-is-bare-metal-programming-in-Embedded-systems#:~:text=%22Bare%20metal%22%20means%20your%20application,considered%20as%20bare%20metal%20programming.) **= This states that Micropython doesn’t need an OS**

<https://www.youtube.com/watch?v=hHec4qL00x0> = Writing fast and efficient micropython