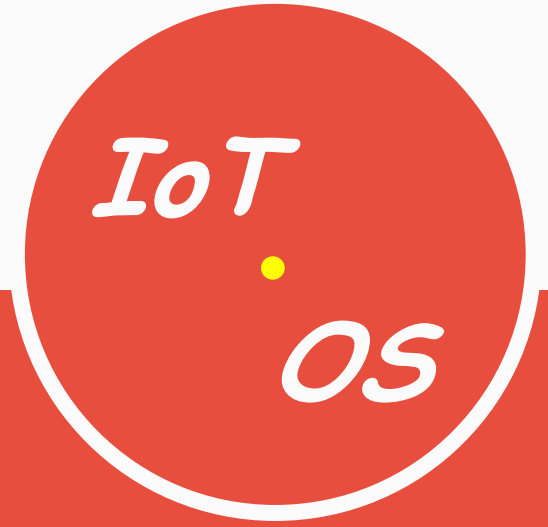
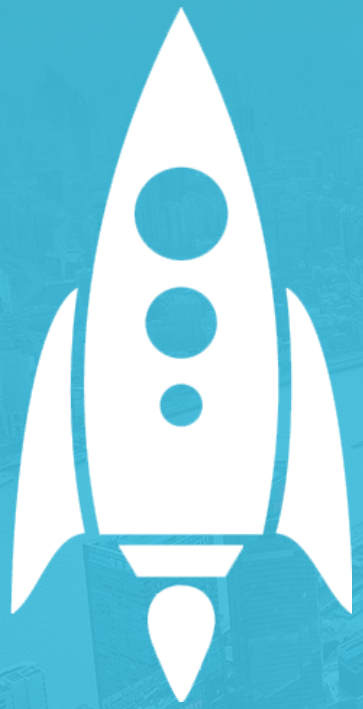


IoT OS on RISC-V with RT-Thread

Outline

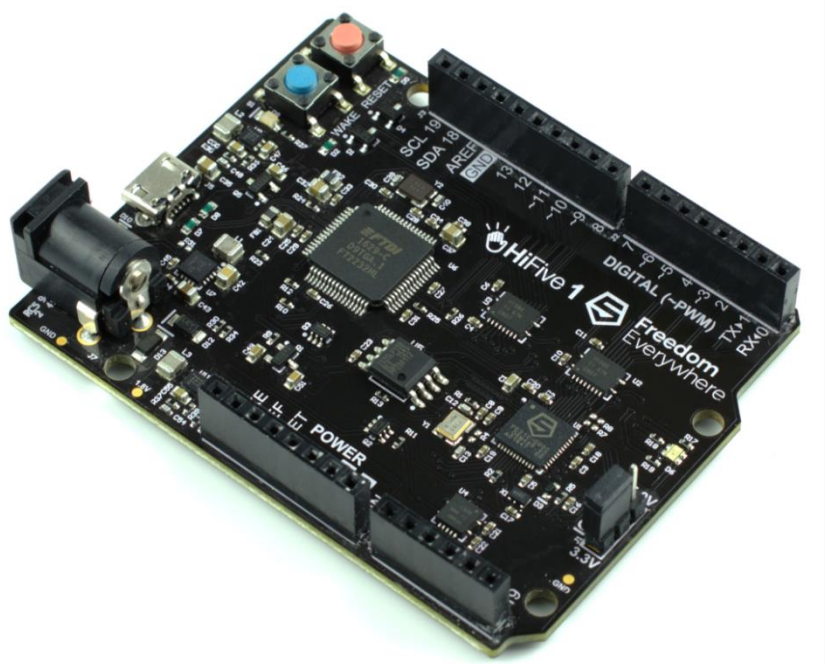
- RTOS Porting on RISC-V
- What is RT-Thread
- Highlights of RT-Thread
- Future of RT-Thread and RISC-V



RT-Thread porting on RISC-V

E310-based porting

- Base HiFive1 Board
- Base QEMU(branch: riscv-all)
  - qemu-system-riscv32 –M sifive_e …



E310-based porting

- RTOS Porting
  - Interrupt Enable/Disable
  - Context Switching
  - Interrupt Handling
  - OS Tick
  - [Cache]

E310-based porting

RT-Thread's libcpu porting interface

- Functions:
  - rt_hw_interrupt_enable
  - rt_hw_interrupt_disable
  - rt_hw_stack_init
  - rt_hw_context_switch_to
  - rt_hw_context_switch
  - rt_hw_context_switch_interrupt

- Variable
  - rt_thread_switch_interrupt_flag
  - rt_interrupt_from_thread
  - rt_interrupt_to_thread

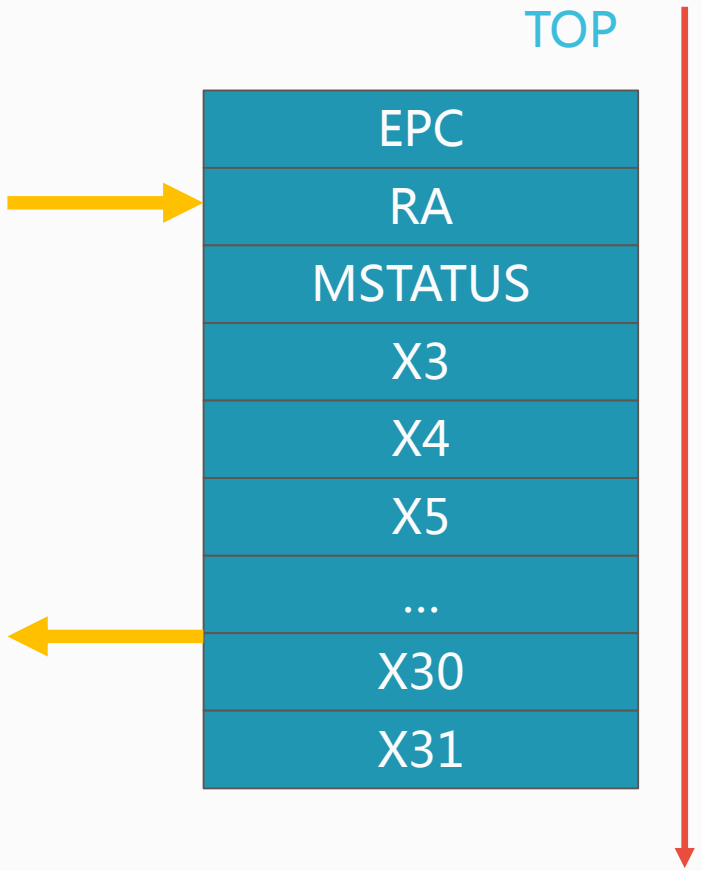
E310-based porting

Interrupt enable/disable

- rt_base_t rt_hw_interrupt_disable(void)
  - Saves the global interrupt status, then disable it and returns the saved state
- void rt_hw_interrupt_enable(rt_base_t level)
  - Restores global interrupt status from variable ‘level’

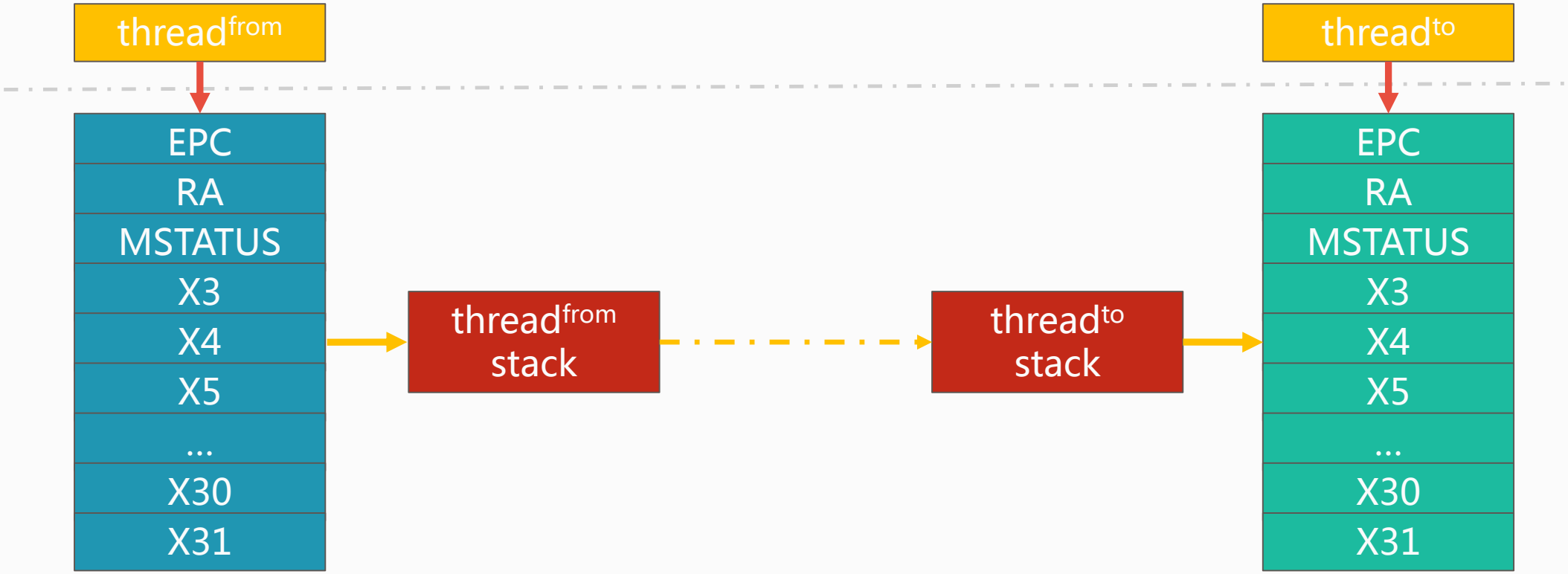
E310-based porting

- **Context switching**
  - Switch to the first thread:
    - **rt_uint8_t *rt_hw_stack_init(**
      void *tentry,
      void *parameter,
      rt_uint8_t *stack_addr,
      void *texit);
    - **void rt_hw_context_switch_to(rt_uint32 to);**



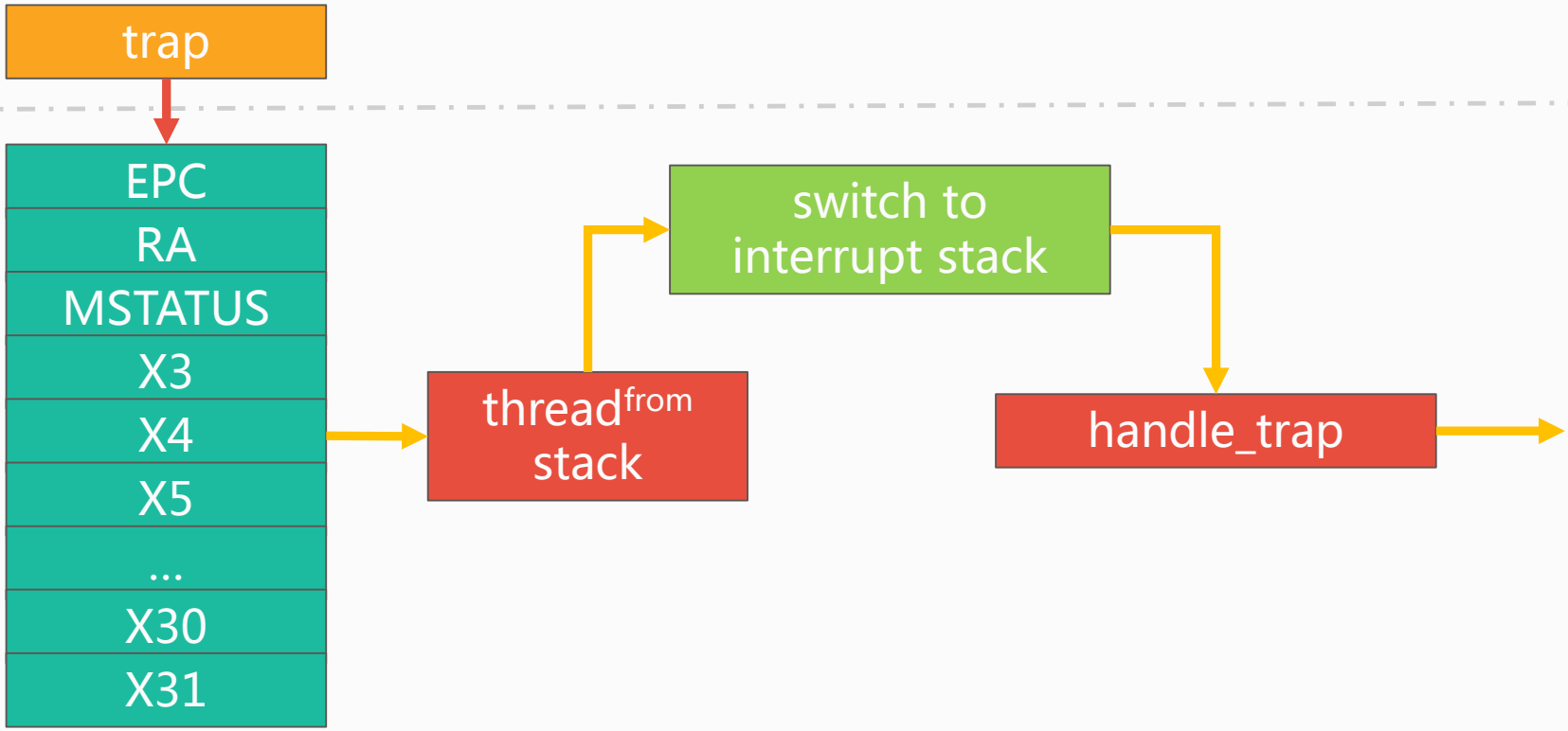
E310-based porting

- **Context switching**
  - Thread to Thread: rt_hw_context_switch(rt_uint32 from, rt_uint32 to)



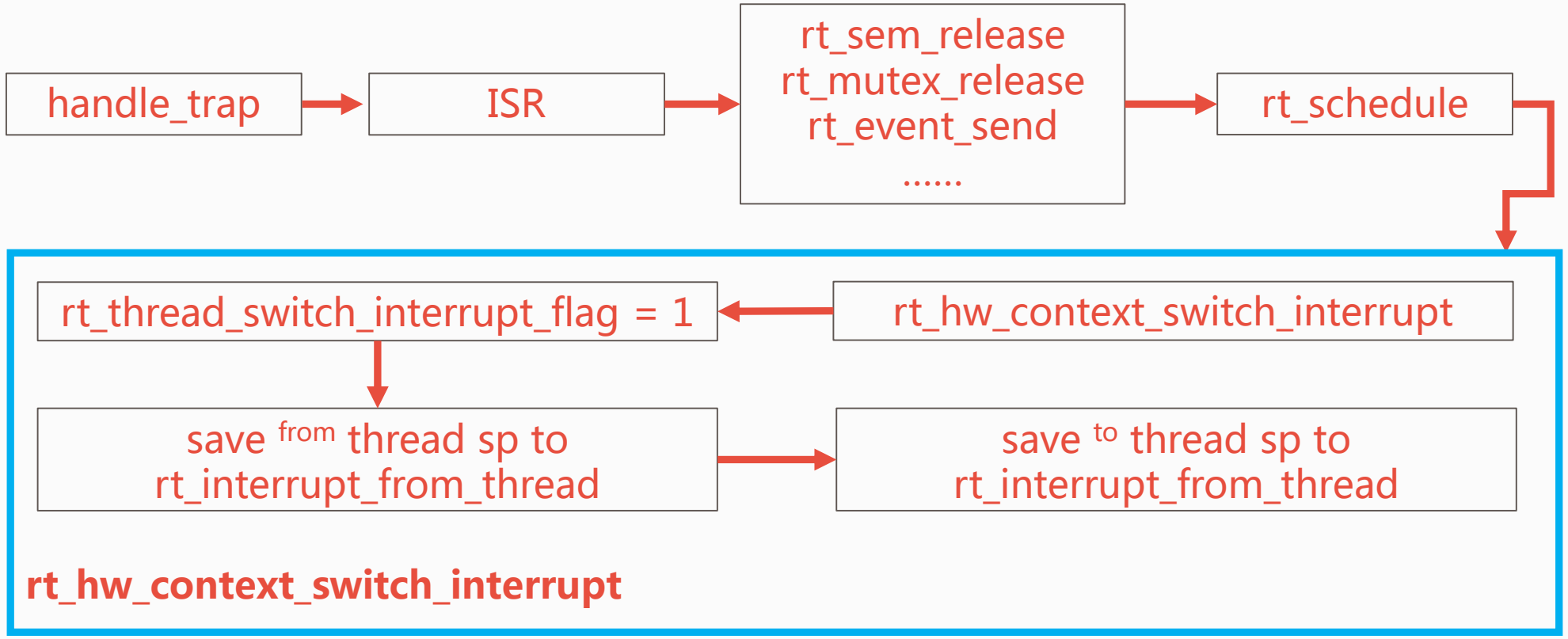
E310-based porting

- Interrupt
  - Save thread^{from} context after entering trap



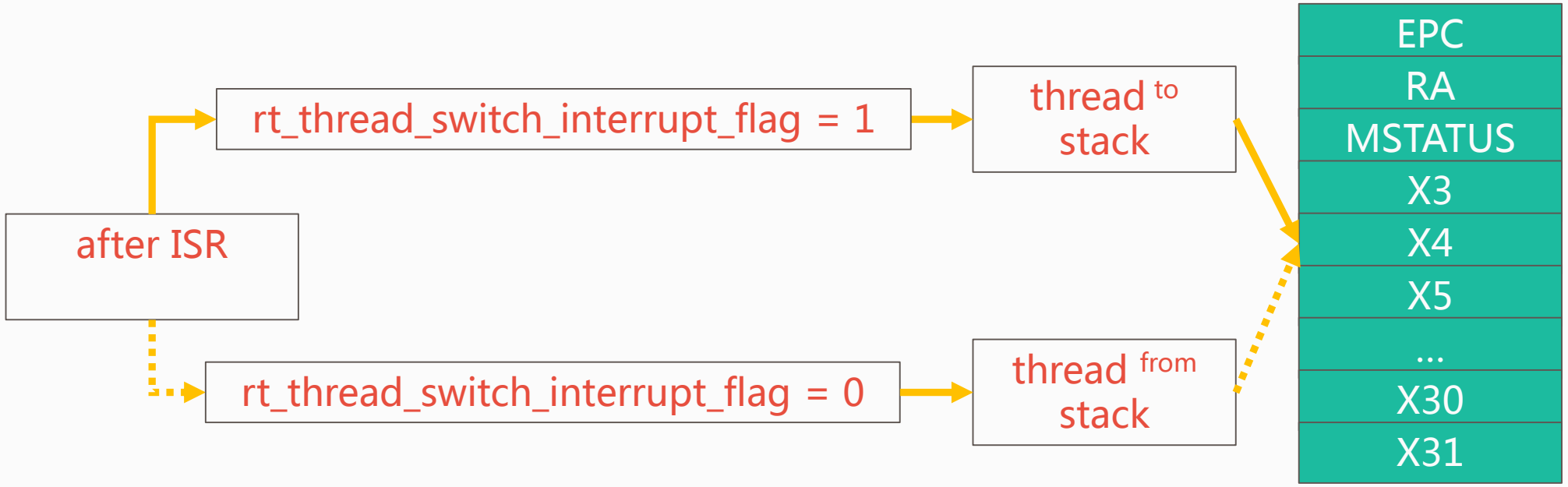
E310-based porting

- **Interrupt**
  - handle_trap() calls the ISR(Interrupt Service Routine)
  - Code in ISR may trigger Thread Scheduling



E310-based porting

- **Interrupt and Context switching**
  - Restore context when the ISR is exited



E310-based porting

- System Tick
  - RTOS requires a timer for OS tick,
  - In the porting of HiFive1 E310, using LFROSC as the clock source triggers the mtime interrupt;
  - In the mtime interrupt handler, every tick calls rt_tick_increase() once;
  - The current tick configuration is 10ms.

RT-Thread Introduction

What is RT–Thread?

1. RT-Thread is **a company**. Real-Thread Technology is the service company behind RT-Thread and promotes the development, maintenance, update, and operation of RT-Thread

2. RT-Thread is **a RTOS kernel** that was born in 2006. It was developed by Bernard Xiong. It is open source and has small footprint; It has been adopted by mainstream companies in many fields and has become the most mature and stable RTOS with the largest installed capacity in China.

3. RT-Thread is **an IoT OS/middleware platform,** includes many software components such as file systems, device frameworks, graphics libraries/GUIs, application frameworks, and more for IoT fields.

4. RT-Thread **is a developer community**, the biggest open source community for embedded software system in China, with tens of thousands of developers and rapid growth.

RT-Thread in Twelve Years

Software ecology

- Mainstream toolchain support for rapid development deployment;
- Rich application frameworks and third-party software tools support;

Industry/Customer

- Used in many high-reliability fields such as energy, medical, automotive, etc.
- Used by hundreds of well-known companies in various fields

Hardware support

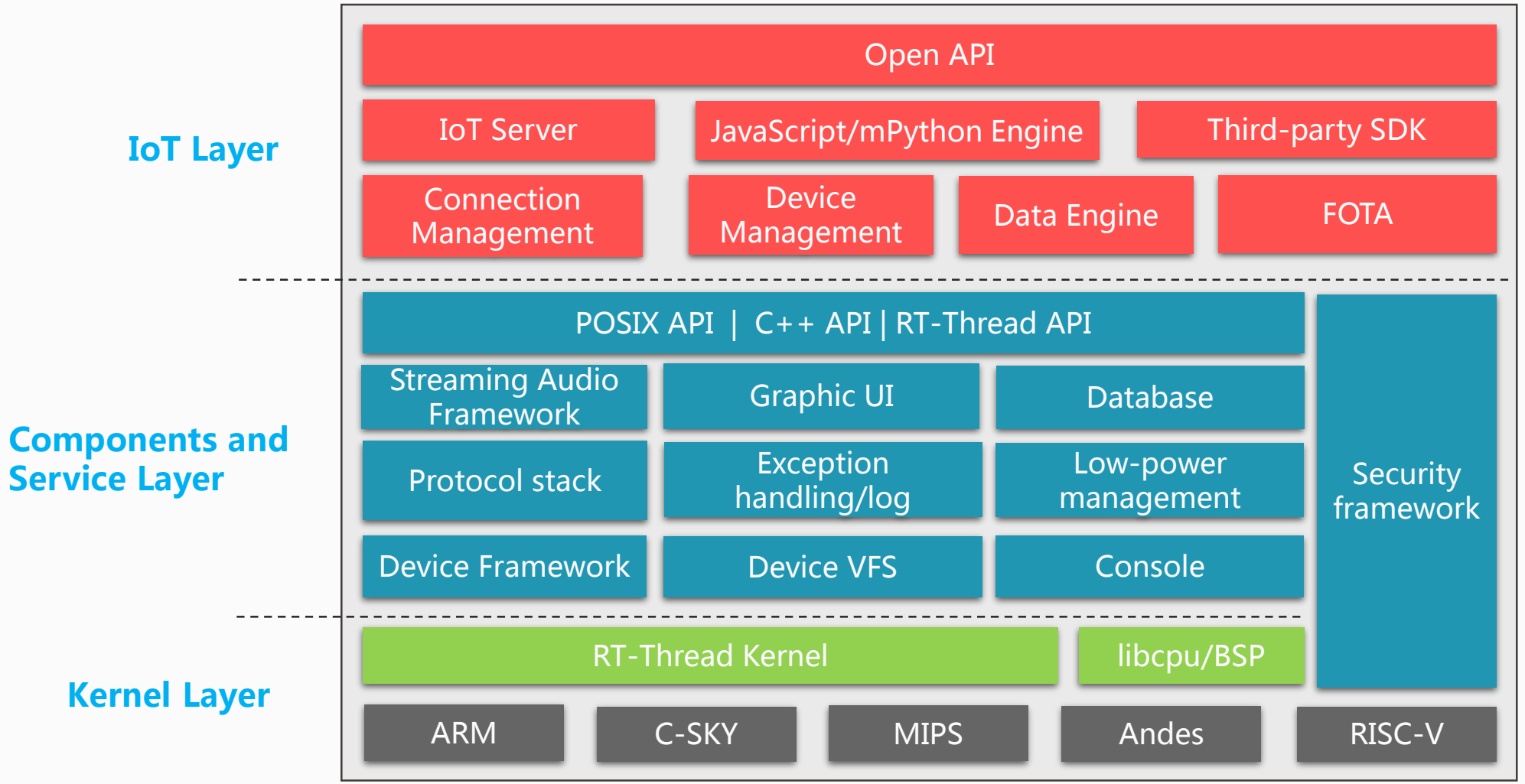
- Multiple major MCU architectures
- Almost all mainstream chip on the market

RT–Thread Application

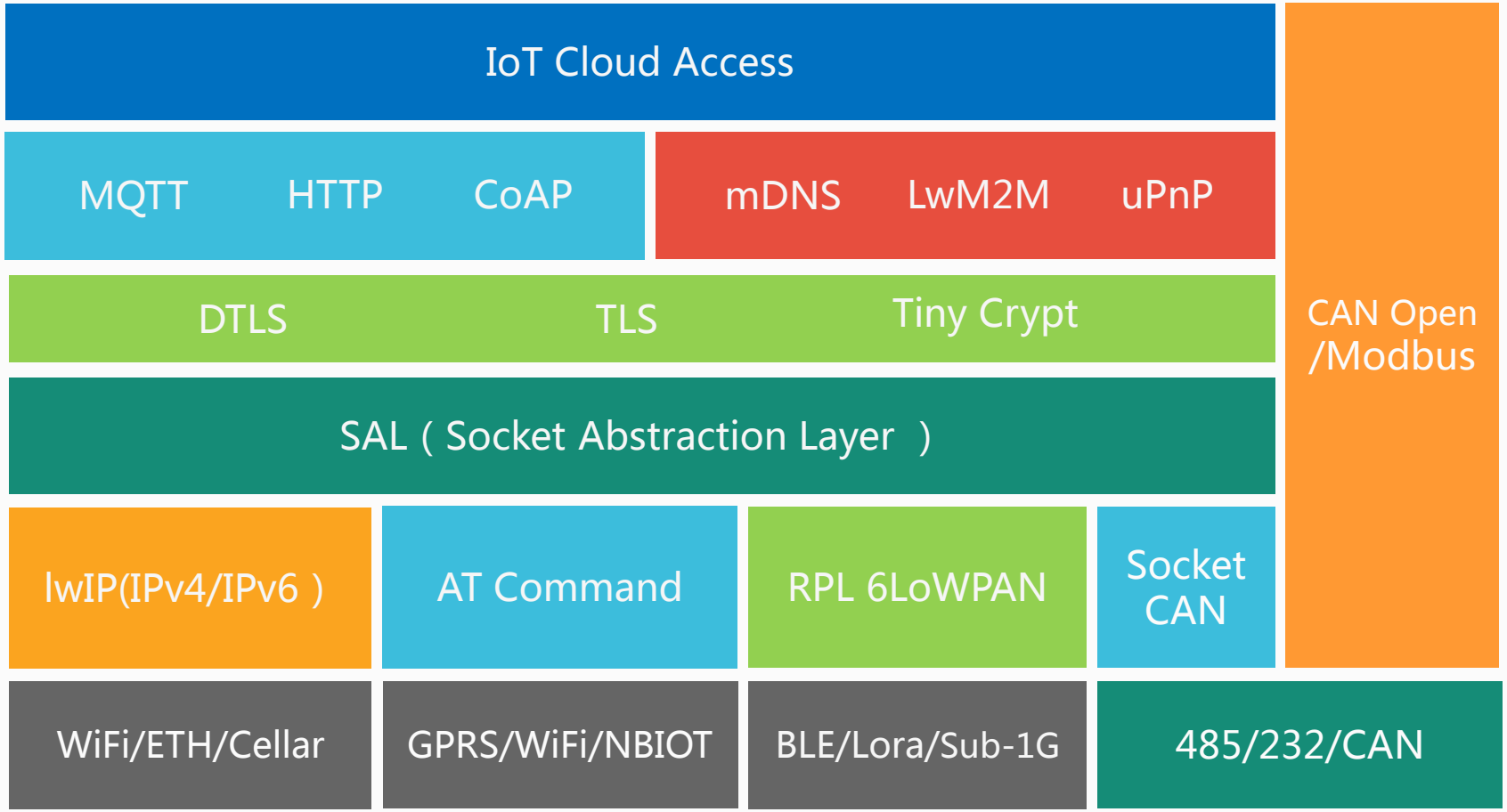
RT-Thread

Computing Device	Cortex-A MIPS32 74K CK810/807	High-performance computing Higher resolution graphics High-end image processing Complex touch interaction	
High-end Smart Devices	ARM9/11 MIPS32 24K CK610	complex but cost sensitive Harsh real-time High resolution graphics Touch interaction with UI	
Smart Device	Cortex-ARM7/M3/M4 MIPS32 M14K CK802/803T Tensilica L106/8	low power and cost sensitive High real-time Button or touch interaction	
Simple Device Node	Cortex-M0 MIPS32 M4K CK801	Sensor Very low power and cost Simple application	

RT-Thread Software Architecture



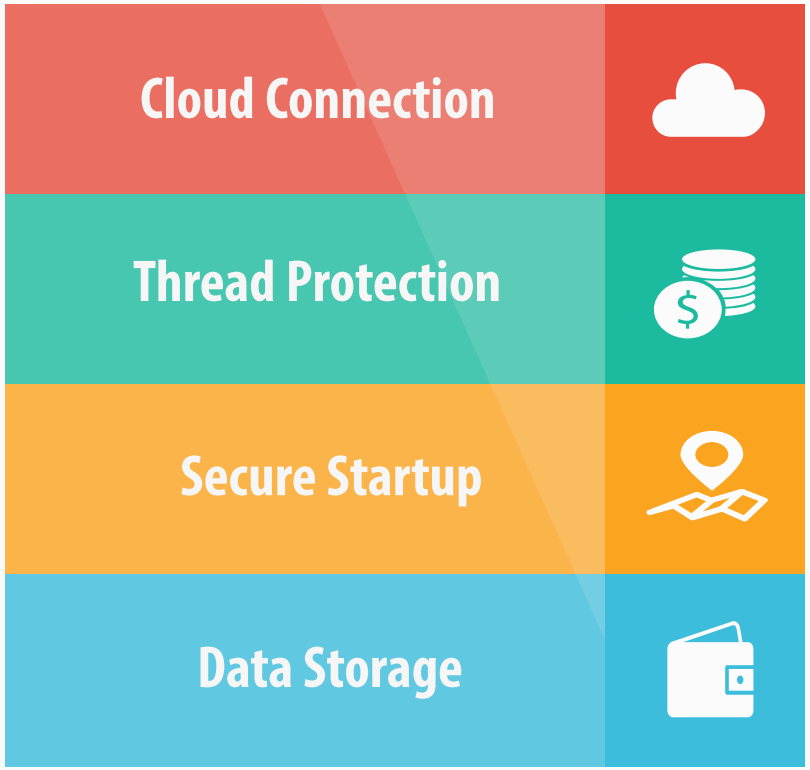
IoT Connection Components



Highlights of RT-Thread

Stable/Reliable | Rich Components | Easy to Use | Highly Scalable | Cross Platform

Security Features



All communications can support encrypted transmission
For example： https、mqtt(tls)、CoAP(dtls)

Isolated thread stack without affecting each other
Automatic detection for stack overflow

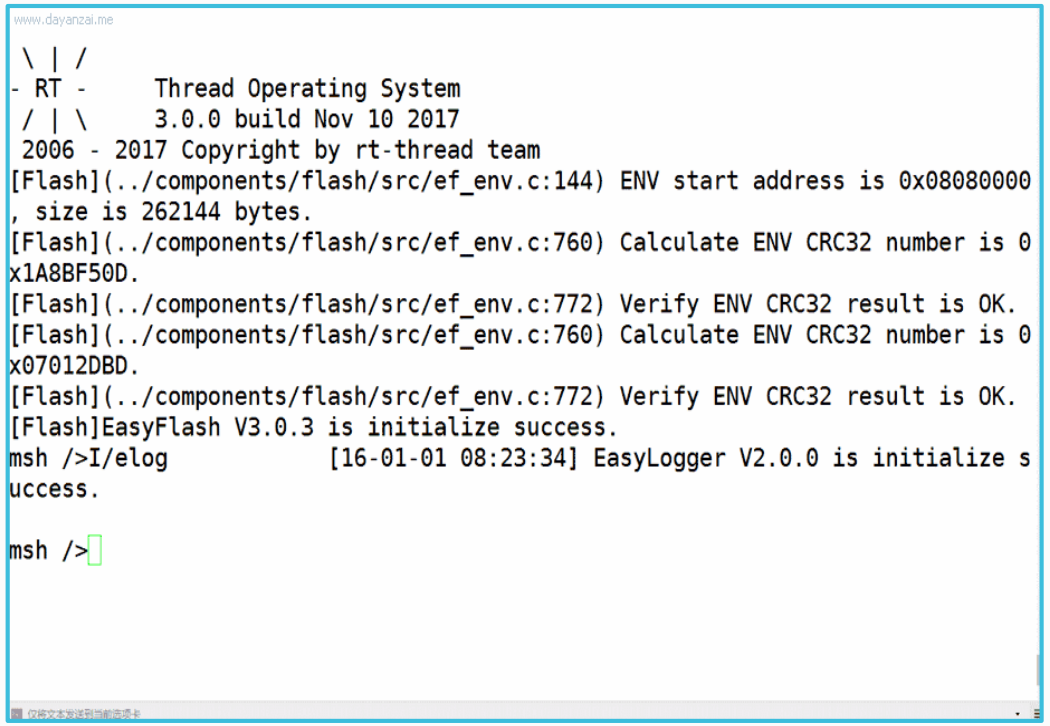
Secure bootloader
Integrated highly reliable OTA components

Cryptographic Library
Supports AES, base64, SHA, MD5, etc.

RT-Thread Script Engine Packages

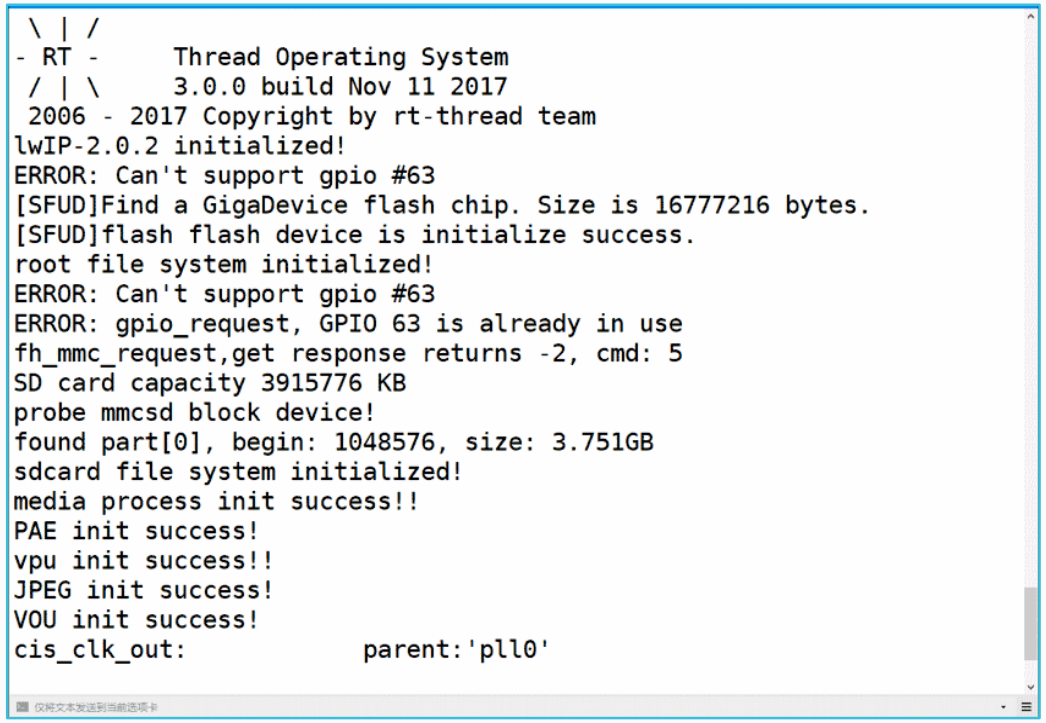
Python Script

- MicroPython Engine for MCU

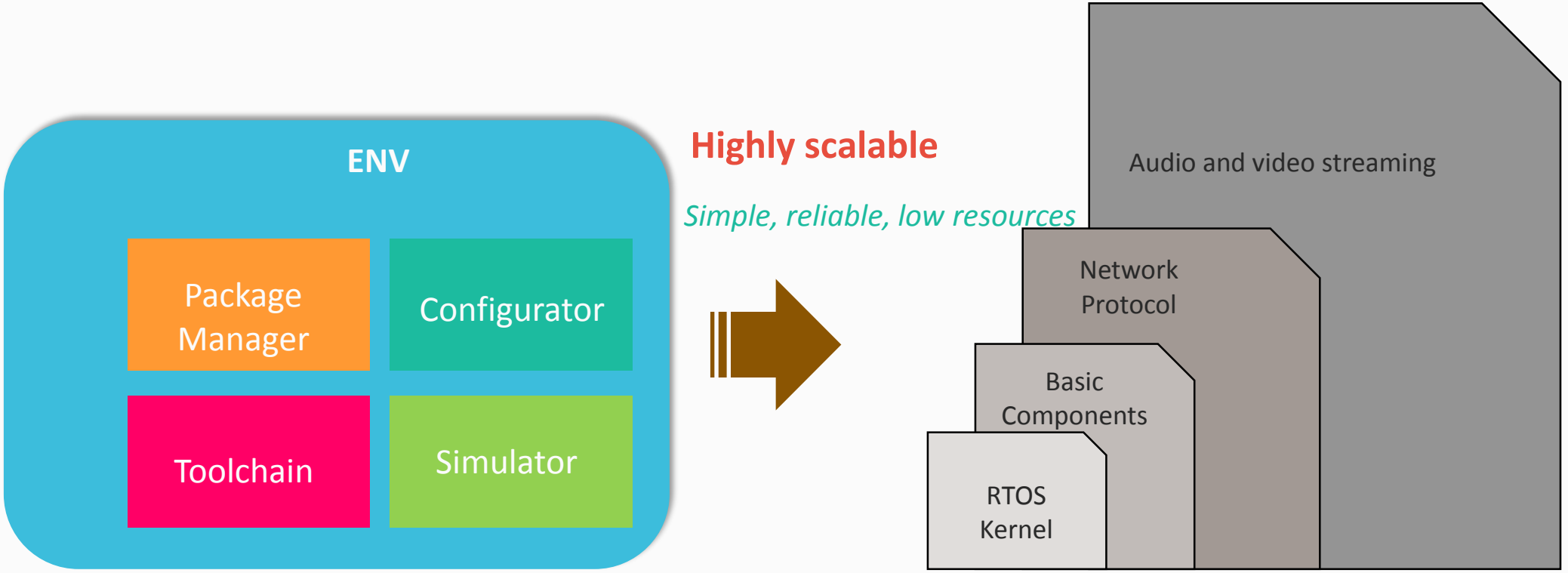


JavaScript Script

- Lightweight JerryScript Engine

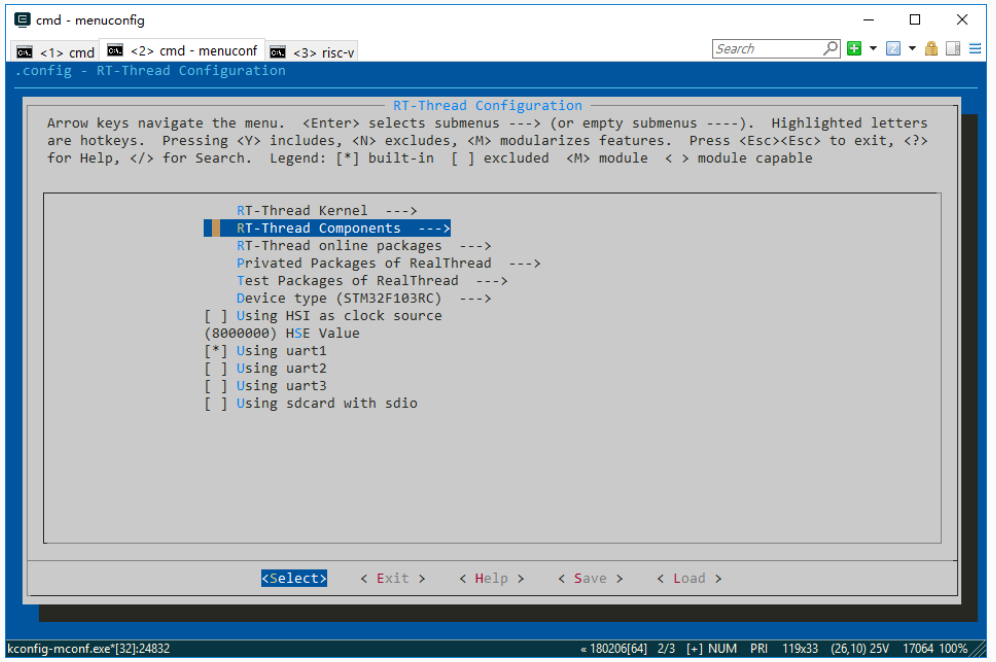


Highly scalable



Configuration Tool

- Configuration System： Kconfig
  - Easy-to-use configuration tools that can be used to adjust the configuration of the kernel and components
  - Set up the system as easy as building blocks
  - UI-based configuration with good interactivity
  - A text help explains the configuration
  - Automatic processing dependencies
  - Efficient configuration checking

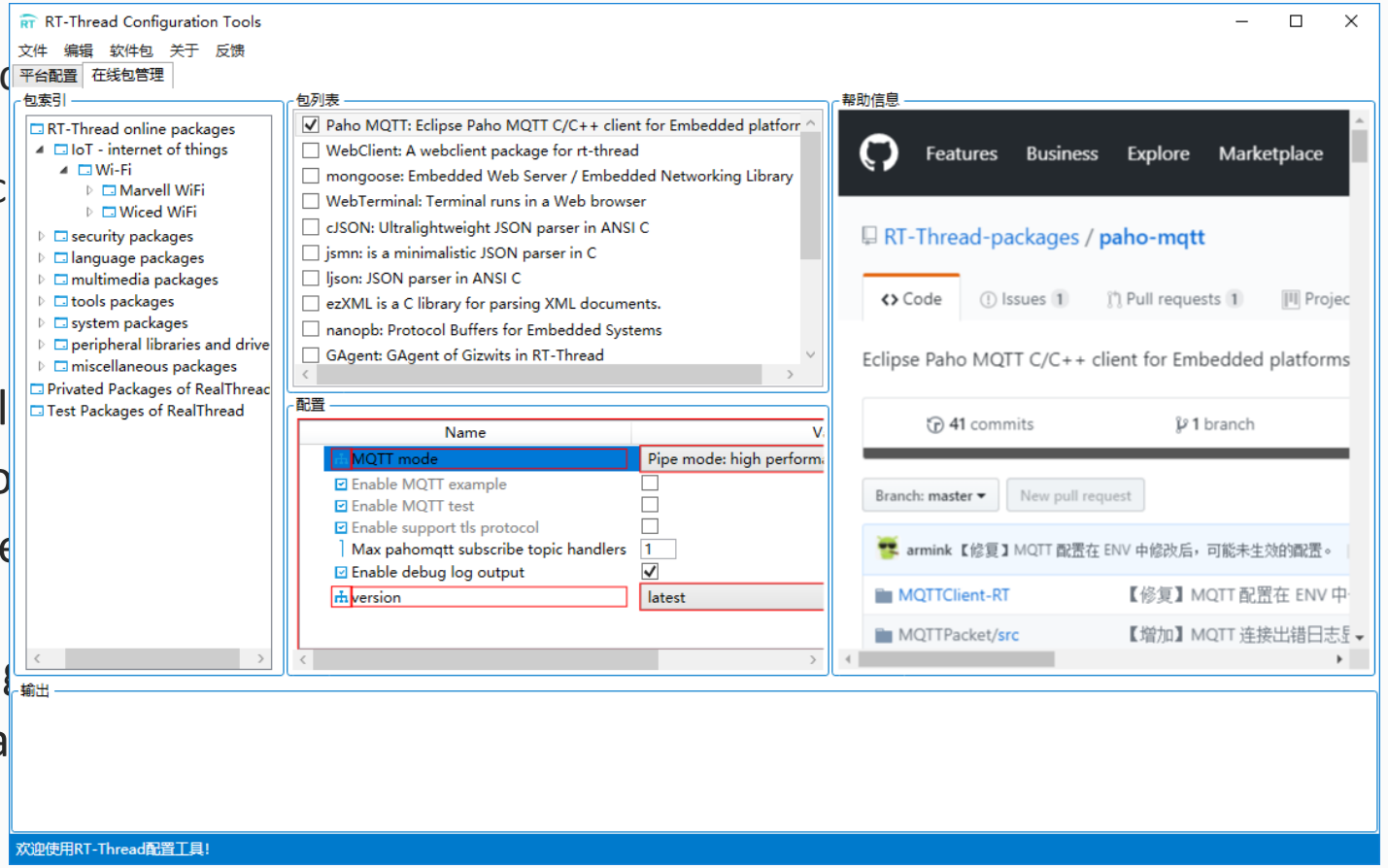


- Build System： SCons
  - Supports multiple tool chains;
  - Supports direct compilation using scons;
  - Supports generation of IDE project files including MDK/IAR, etc.

Package Management Tools

- Package Management System
  - Packages for different application areas, such as system, language, network protocol and application, security etc.
  - A package is composited with desc
    files.
  - Based on RT-Thread ENV tool, pac

- **Why use the package**
  - Splits kernel and packages for l
  - Supports continuous integratio
  - It's an open platform, everyone
    share the package in the open
  - With the more software packag
  - There are currently 50+ softwa



Future of RT-Thread/RISC-V

Why RT-Thread is suitable for RISC-V

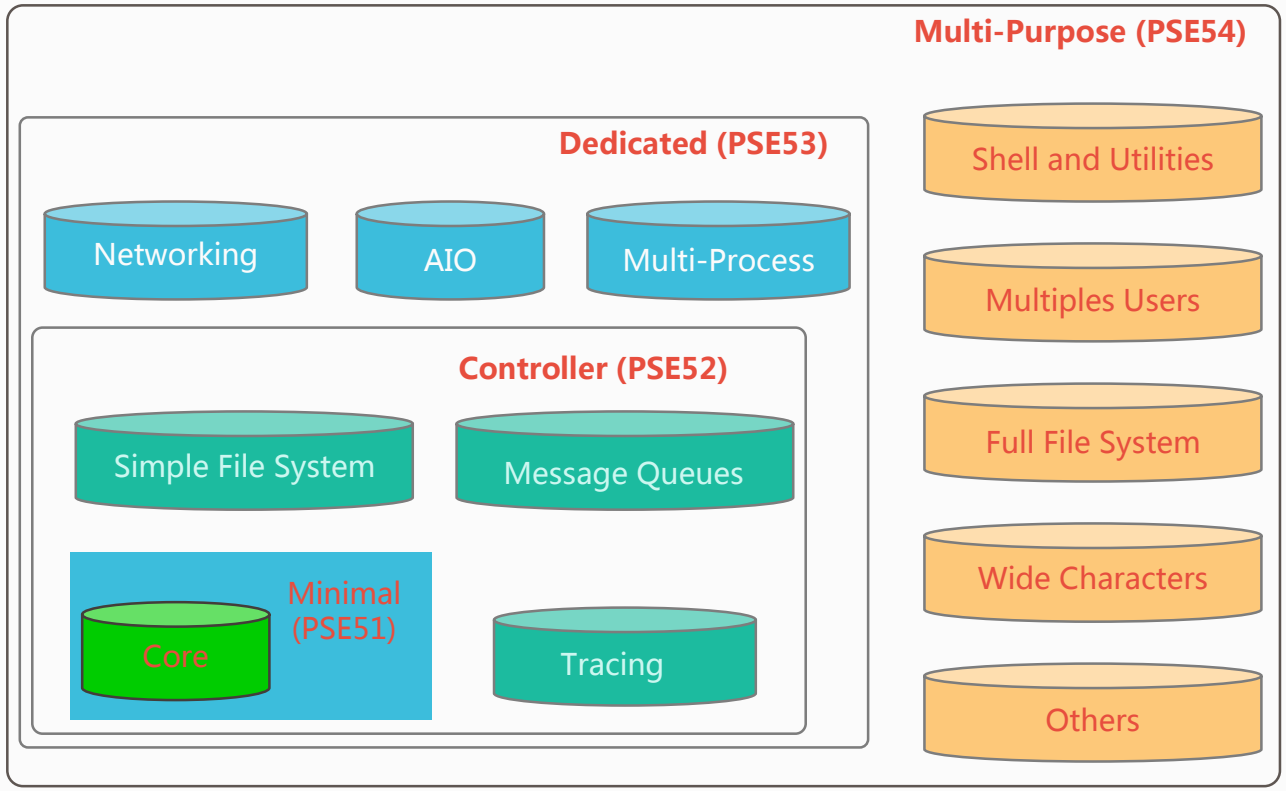
- Why RT-Thread is suitable for RISC-V?
  - Open source ISA + Open source OS
  - Rich components
  - Highly scalable, from MCU, IoT Soc, to MPU
  - The basic kernel support in HiFive1 E310 demand for resources is:
    - FLASH：8.8KB
    - RAM：5.4KB

Why RT-Thread is suitable for RISC-V

- Why RT-Thread is suitable for RISC-V?
  - IoT Chip + IoT OS
  - Tiny resource requirements with full features TCP/IP stack: lwIP
    - Typical footprint with 64KB ROM, 24KB RAM
  - Nano resource requirements with AT framework:
    - AT Client：4.3KB ROM，2.0KB RAM
    - AT Client + AT Socket + SAL： 14KB ROM，<4KB(with 5 sockets)

Why RT-Thread is suitable for RISC-V

- Why RT-Thread is suitable for RISC-V?
  - POSIX-compliant IoT OS:
    - File System
    - Dynamic Linking
    - PThread/Semaphore
    - Memory Management
    - Network
    - Device File
    - …



The Experience/Issues

- The Experience/Issues in RT-Thread porting, QEMU etc
  - Toolchain
    - There are several different versions of the tool chain, how to choose?
  - QEMU
    - RT-Thread/HiFive1 currently only runs on QEMU's RISC-ALL branch
    - "set riscv use_compressed_breakpoint no“
    - is required before using breakpoints
  - Debugger
    - Compared to the ARM platform, the debugger is hard to use.

RT–Thread for future planning of RISC–V

- **RT-Thread for future planning of RISC-V:**
  - Support E200 soft core CPU in Lichee Tang
  - Support GAP8 IoT application processor
  - Helping the promotion of RISC-V

Thank You