From Reset To Main(): Porting Infineon TriCore AURIX To Zephyr

Parthiban & Saravanan



Who we are

Parthiban & Saravanan, Linumiz

- 15+ years embedded software
- Linux kernel, U-Boot, Zephyr RTOS, recently NuttX
- Active upstream Maintainer & contributor: U-Boot, kernel, Zephyr
- Frequent OSS / Embedded Open Source Summit speaker

www.linumiz.com

Linumiz
open-source powered

Linumiz is a domain-agnostic embedded software engineering and consulting company specializing in Linux and Zephyr RTOS. Our expertise spans board bring-up, Board Support Package (BSP) development, device driver implementation, and over-the-air (OTA) software updates, primarily for ARM-based systems but not limited to them. We empower businesses with tailored embedded solutions, ensuring reliability, scalability, and seamless hardware-software integration.

Embedded Linux

- Board support package
- Driver Development
- Board Bring up
- Custom Linux build
- Application Development



RTOS

- Board porting
- Firmware Development
- Driver development
- Update management
- Upstream activities



Functional Safety

- Hazard & Risk Analysis
- Architecture design
- Requirements Engineering
- ISO/IEC Compliance
- Validation & Verification



Device security

- Static analysis
- Dynamic analysis
- Device hardening
- Root of Trust (RoT)
- Security recommendations



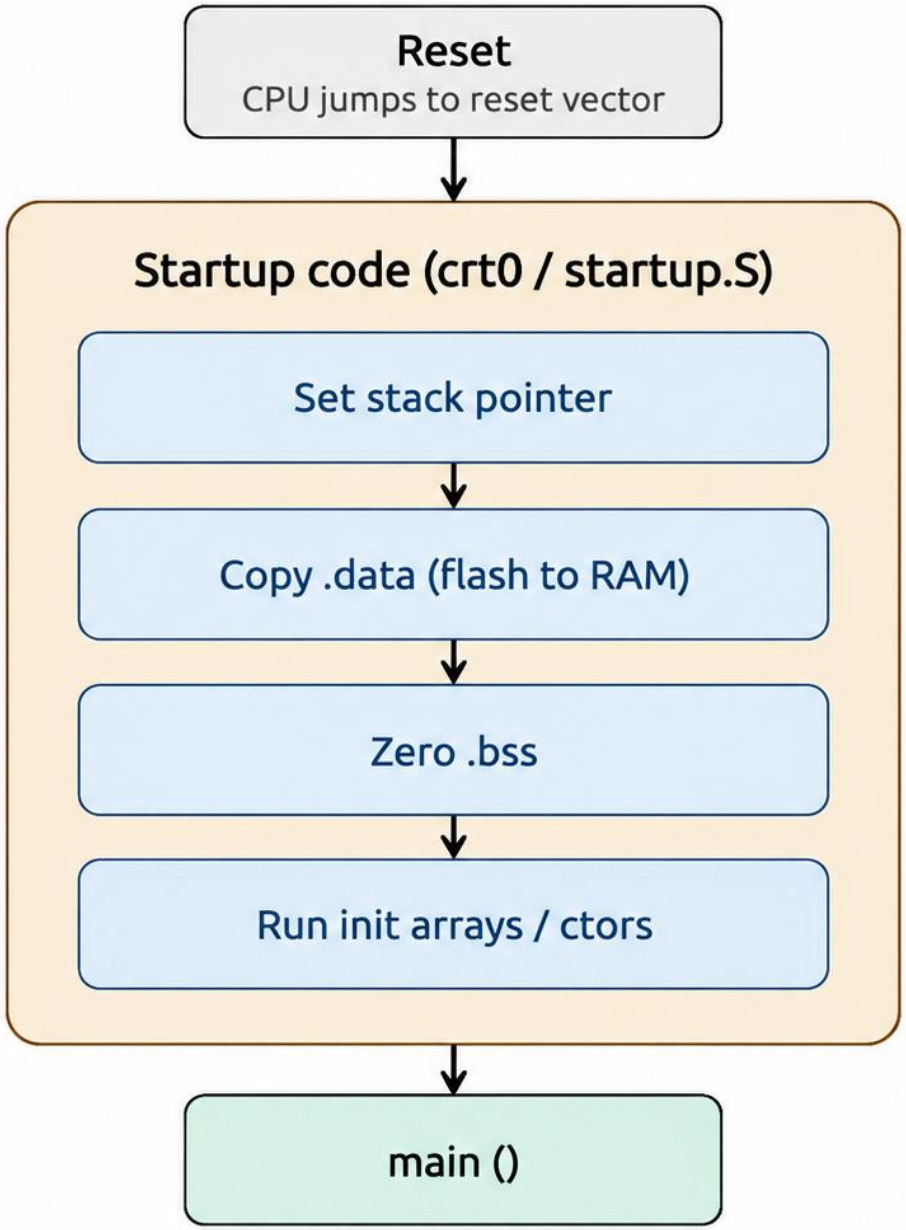
Linumiz
open-source powered

Agenda

1. Baremetal vs RTOS bootflow
2. Zephyr kernel to main
3. Beyond arch
4. ISR

Linumiz
open-source powered

Baremetal boot flow



Linumiz
open-source powered

Baremetal linker script snippet

```
ENTRY(_START)

MEMORY
{
     FLASH  (rx)   :  ORIGIN = 0xA0000000, LENGTH = 4M
     RAM    (rwx)  :  ORIGIN = 0x70000000, LENGTH = 240K
}

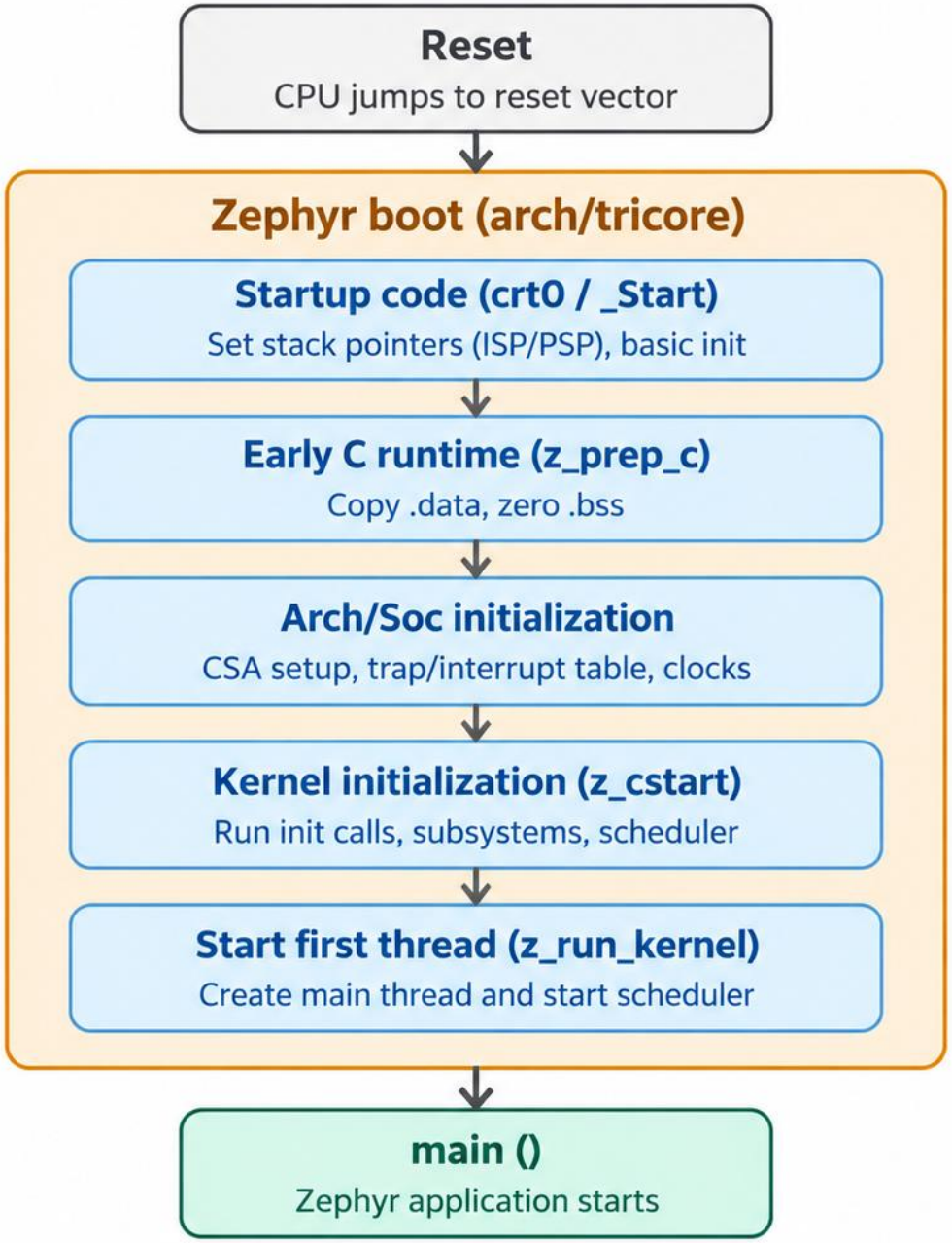
SECTIONS
{
     .start 0xA0000000 :
     {
          KEEP(*(.start))
     } > FLASH

     .data : { *(.data*) } > RAM AT > FLASH
     .bss  : { *(.bss*)  } > RAM

     __USTACK0 = ORIGIN(RAM) + LENGTH(RAM);
     __CSA0_END = __USTACK0 - 2K;
     __CSA0 = __CSA0_END - 8K;
}
```

Linumiz
open-source powered

Zephyr RTOS: Bootflow



Linumiz
open-source powered

Zephyr RTOS Linker script snippet

```
MEMORY {
    ROM    (rx) : ORIGIN = 0x80000000, LENGTH = 4M
    DSRAM0 (rw) : ORIGIN = 0x70000000, LENGTH = 240K
}

ENTRY("__start")

SECTIONS {
    .reset : {
        KEEP(*(.reset.*))
    } > ROM

    .vectors.trap : {
        __btv = .;
        KEEP(*(.vectors.trap*))
    } > ROM

    .vectors.irq : {
        __biv = .;
        KEEP(*(.vectors.irq*))
    } > ROM
}
```

Linumiz
open-source powered

Arch init

```
/* Set base trap vector table pointer */
mov     %d10, lo:__btv
addih   %d10, %d10, hi:__btv
mtcr    TRICORE_BTV, %d10

/* Set base interrupt vector table pointer */
mov     %d15, lo:__biv
addih   %d15, %d15, hi:__biv
mtcr    TRICORE_BIV, %d15

/* Set interrupt stack pointer */
mov     %d15, lo:(z_interrupt_stacks + CONFIG_ISR_STACK_SIZE)
addih   %d15, %d15, hi:(z_interrupt_stacks + CONFIG_ISR_STACK_SIZE)
mtcr    TRICORE_ISP, %d15

/* Set main stack pointer
movh.a  %a10, hi:(z_interrupt_stacks + CONFIG_ISR_STACK_SIZE)
lea     %a10, [%a10], lo:(z_interrupt_stacks + CONFIG_ISR_STACK_SIZE)
```

Linumiz
open-source powered

Why early arch init code is in Assembly?

1. Set special purpose registers directly (e.g. CSA/FCX, PCXI, trap/interrupt base registers).

2. Set initial stack pointers (ISP/PSP and main stack A10) before entering C code.

3. Control the reset/boot flow at a known address with no C runtime or stack.

4. No C runtime available yet to initialize memory and environment (.data/.bss, stack, global data).

Linumiz
open-source powered

Why to set stack pointer?

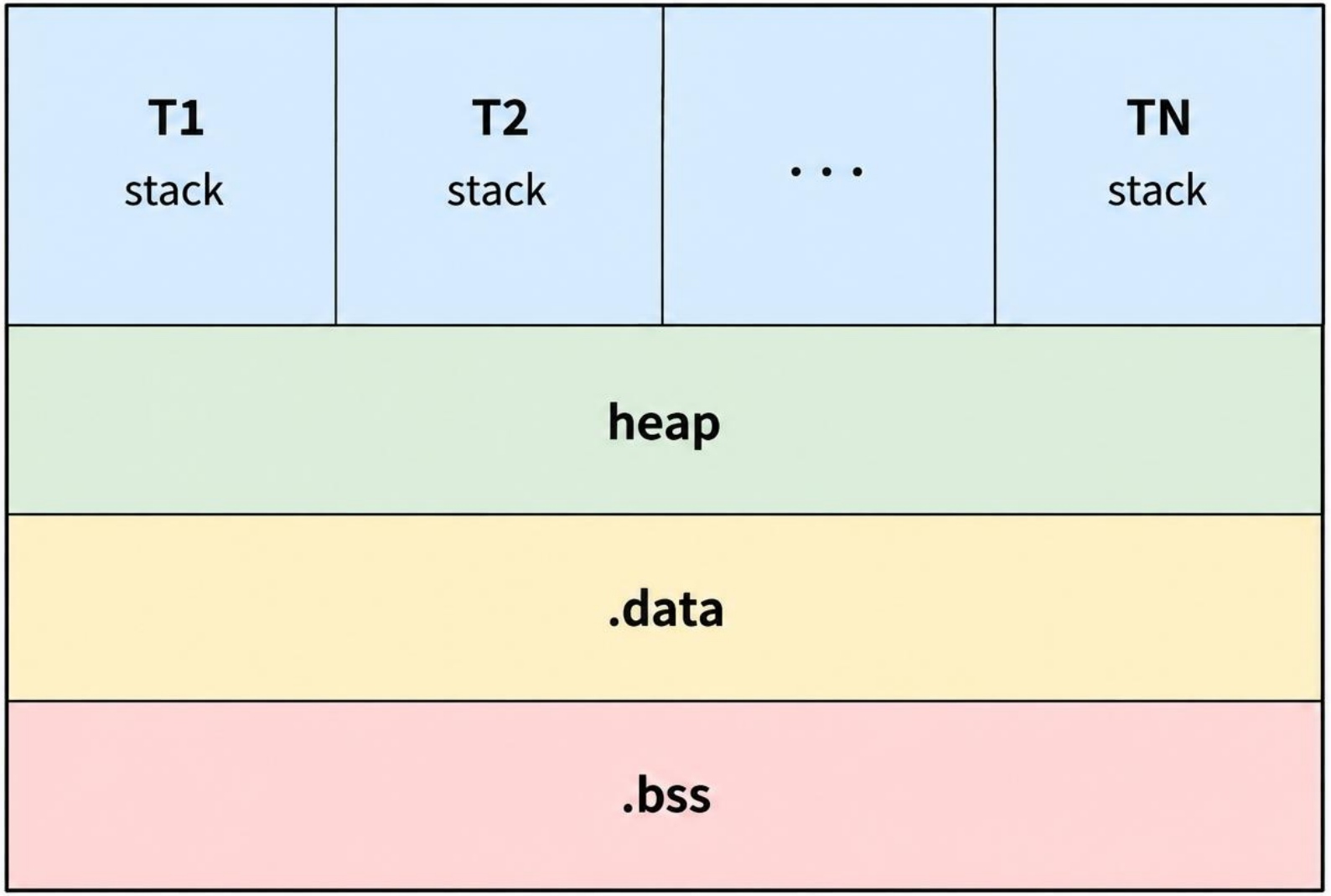
```
/* Boot: initialize stack pointer */
movh.a   %a10, 28672
lea      %a10, [%a10], lo:z_main_stack

          ...

/* C code: stack is now actually used */
main:
add.a    %a10, -8
st.a     [%a10], %a15
call     printf
```

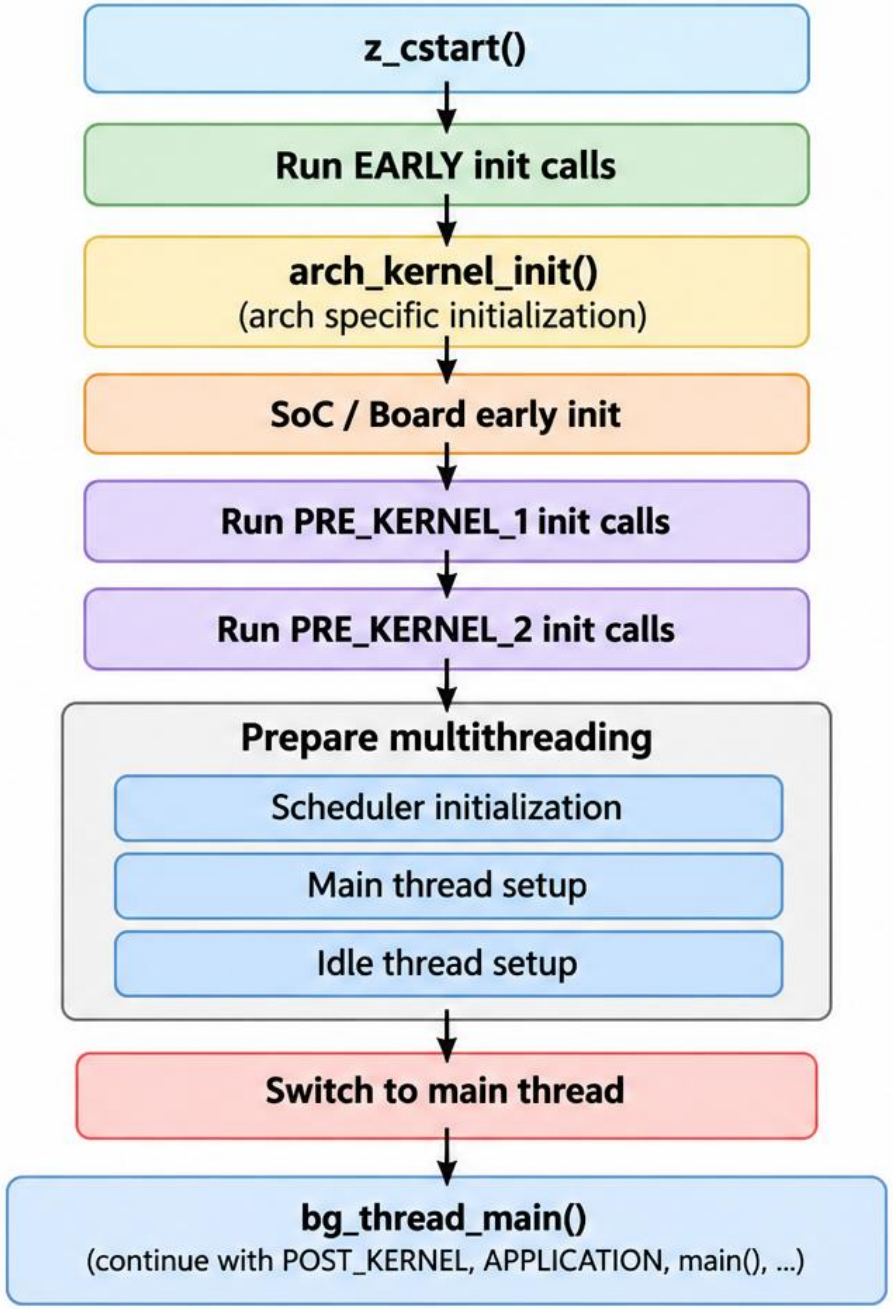
Linumiz
open-source powered

Zephyr RTOS: One process, multiple threads



Linumiz
open-source powered

z_cstart: arch independent init



Linumiz
open-source powered

Beyond Arch Init

1. **Thread handling**
   Creation, abort and exit

2. **Context switch**
   Save current thread and restore next thread

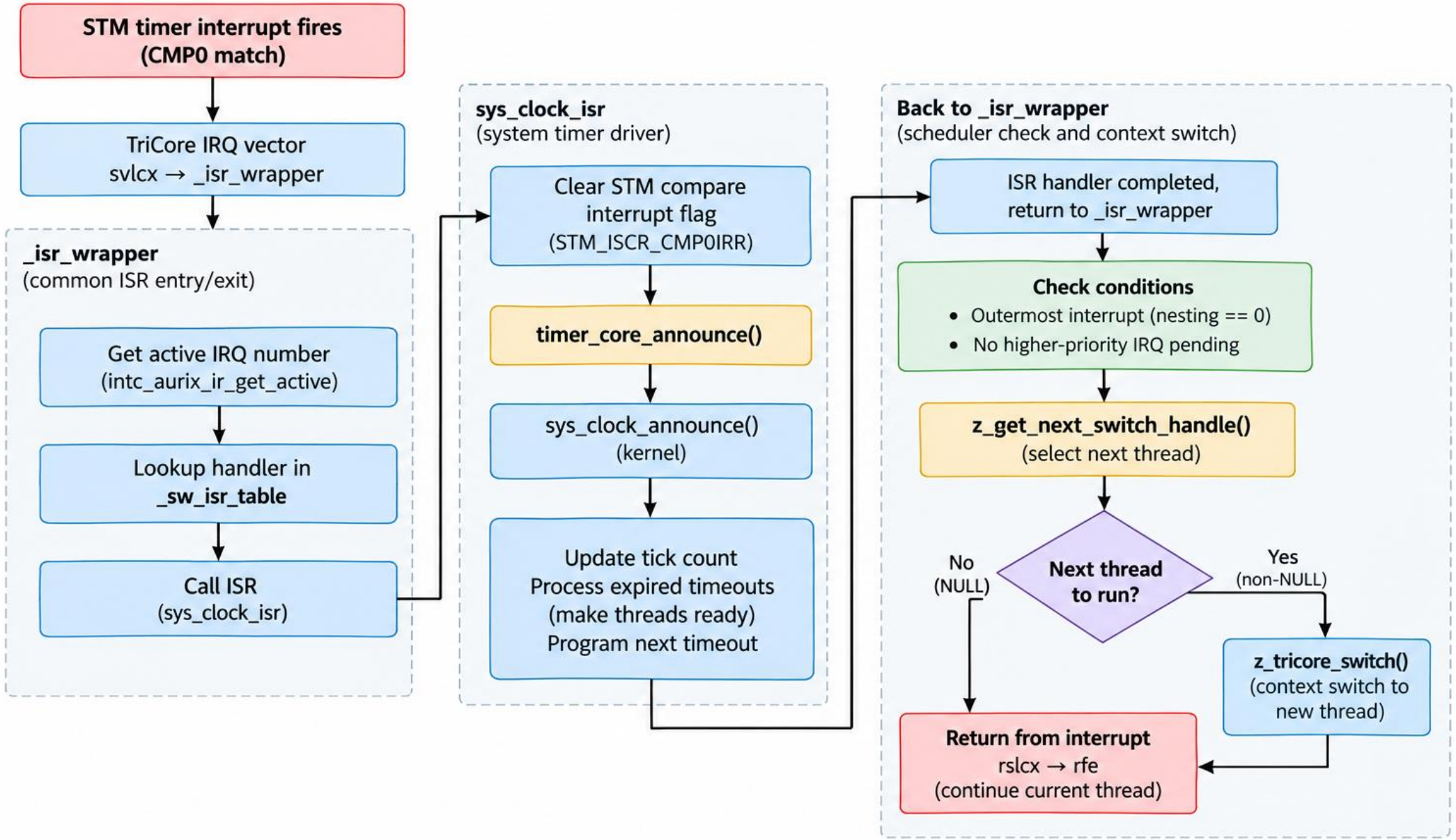
3. **Interrupt handling**
   Handle hardware interrupts and dispatch ISRs

4. **Exception / Trap handling**
   Handle CPU exceptions and system calls

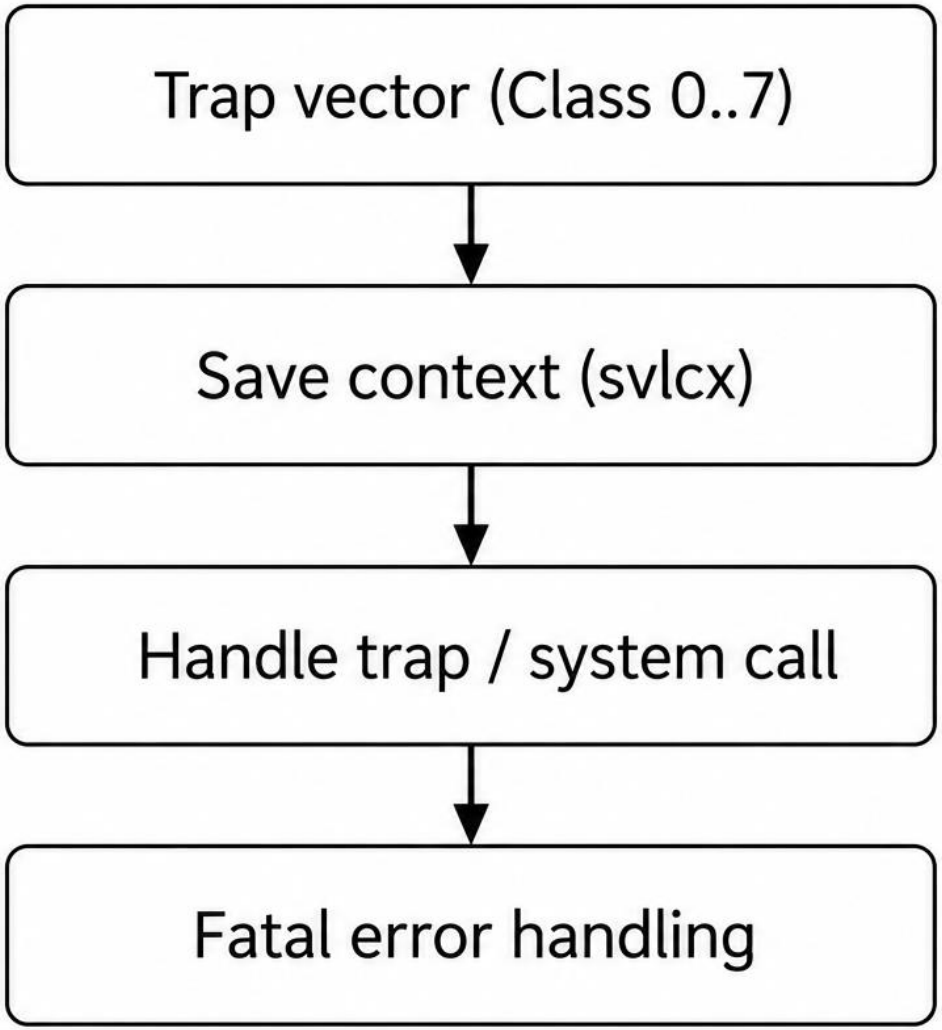
5. **CPU idle**
   Enter low-power wait state

Linumiz
open-source powered

System Timer Interrupt Flow and Scheduler Check

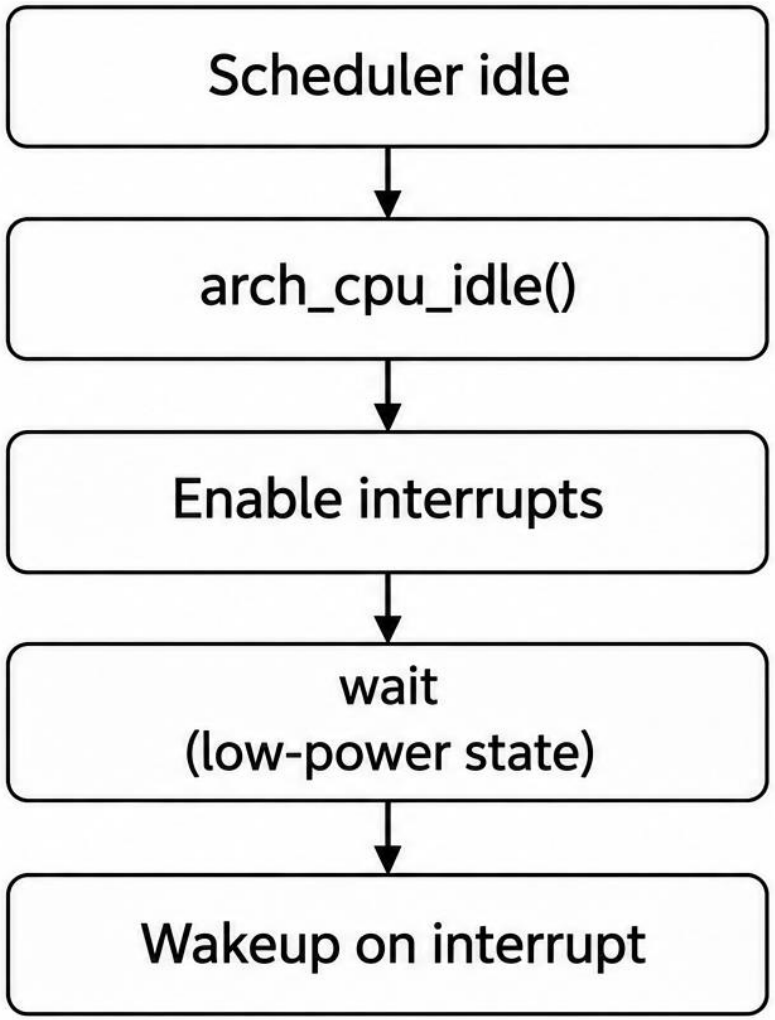


Exception / Trap Handling



Linumiz
open-source powered

CPU Idle



Linumiz
open-source powered

What is not covered?

- Thread handling
- Context switching
- voluntary re-sched / release CPU: sleep, mutex, sem
- Userspace and system calls
- Memory Protection Units
- SMP, AMP cases

Linumiz
open-source powered

Thank You

contact@linumiz.com

Linumiz
open-source powered