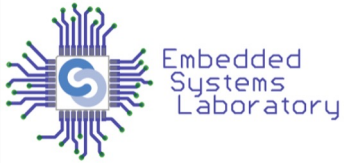


PROIECTAREA CU MICROPROCESOARE

Cursul 11
RTOS

Facultatea de Automatică și Calculatoare
Politehnica București

Ce este RTOS?

- Real-Time Operating System
- Rularea de task-uri ce au constrângeri foarte bine definite de timp de execuție
- Event-driven sau time-sharing, de obicei pre-emptive scheduling
- Algoritm predictibil pentru alocarea dinamică a memoriei
- În unele aplicații este permisă doar alocarea statică a memoriei
- De obicei pune la dispoziției mecanisme pentru intertask communication și resource sharing

Soft Real-Time Systems

- Nu este de dorit, dar nici nu este catastrofal dacă deadline-urile nu sunt îndeplinite
- Sunt sisteme “best effort”
- Majoritatea sistemelor moderne de operare cu care v-ați întâlnit până acum sunt soft real time (Linux, Windows, MacOS etc.)
- Exemple:
  - Transmisia și recepția multimedia
  - Rețelistică, rețelele celulare
  - Web sites, servicii cloud
  - Jocuri

Hard Real-Time Systems

- Au deadline-uri care trebuie să fie satisfăcute, altfel este posibilă o defecțiune catastrofală
- Prima dată, a doua dată și de fiecare dată!
- Necesită verificare formală sau garanții că deadline-urile sunt întotdeauna îndeplinite
- Exemple:
  - Controlul traficului aerian
  - Controlul subsistemelor unui vehicul
  - Controlul unui reactor nuclear

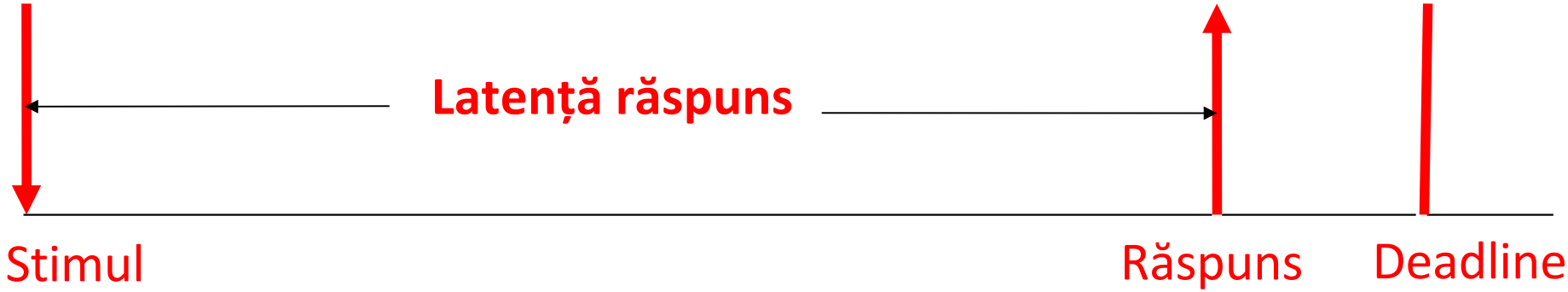
Firm Real-Time Systems

- Trebuie să îndeplinească deadline-urile
- Ratarea unui deadline poate să aibă un impact semnificativ sau să cauzeze efecte de nedorit, ca reducerea calității
- Exemple:
  - Aplicații multimedia
  - Linii de asamblare robotizată
  - Sisteme de planificare financiară

De ce am folosi un RTOS?

1. Dacă avem cerințe hard real-time
2. Rularea unei aplicații pe hardware low-end (de ex. microcontroler) care nu are resursele necesare pentru rularea unui OS complet.
3. Dacă avem nevoie de multitasking, sincronizare, planificare etc.

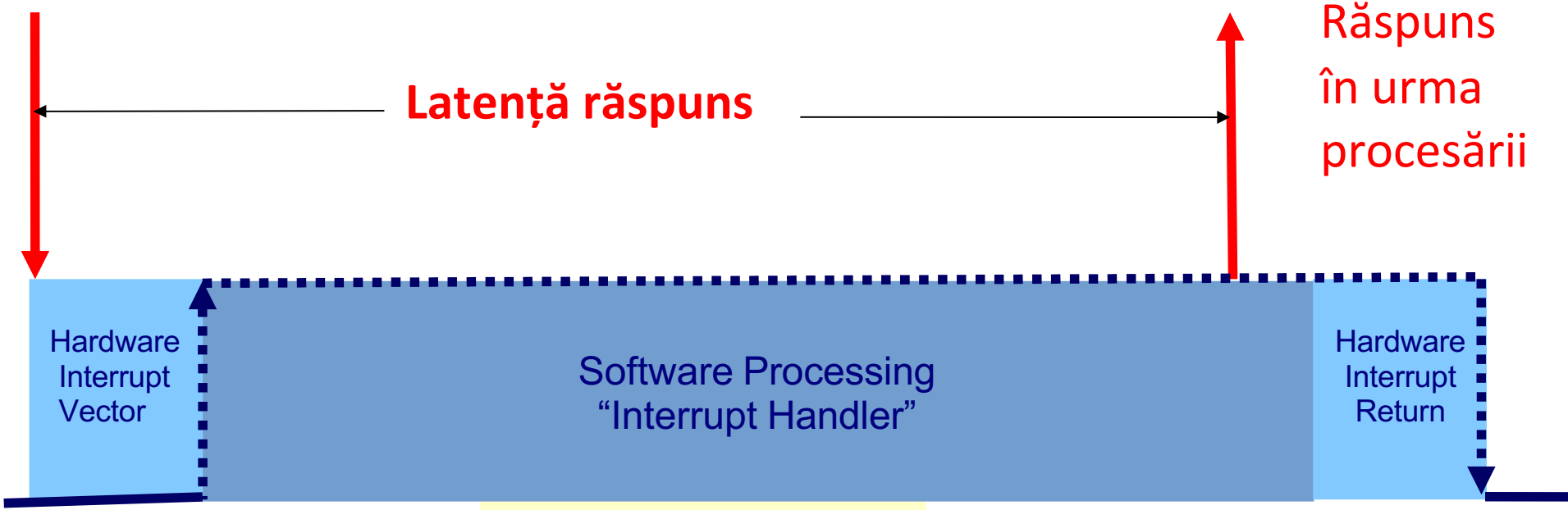
Real-time = deterministic



- Real-time nu înseamnă neapărat rapid
- Sistemele real-time au termene limită

Bare-metal, single interrupt

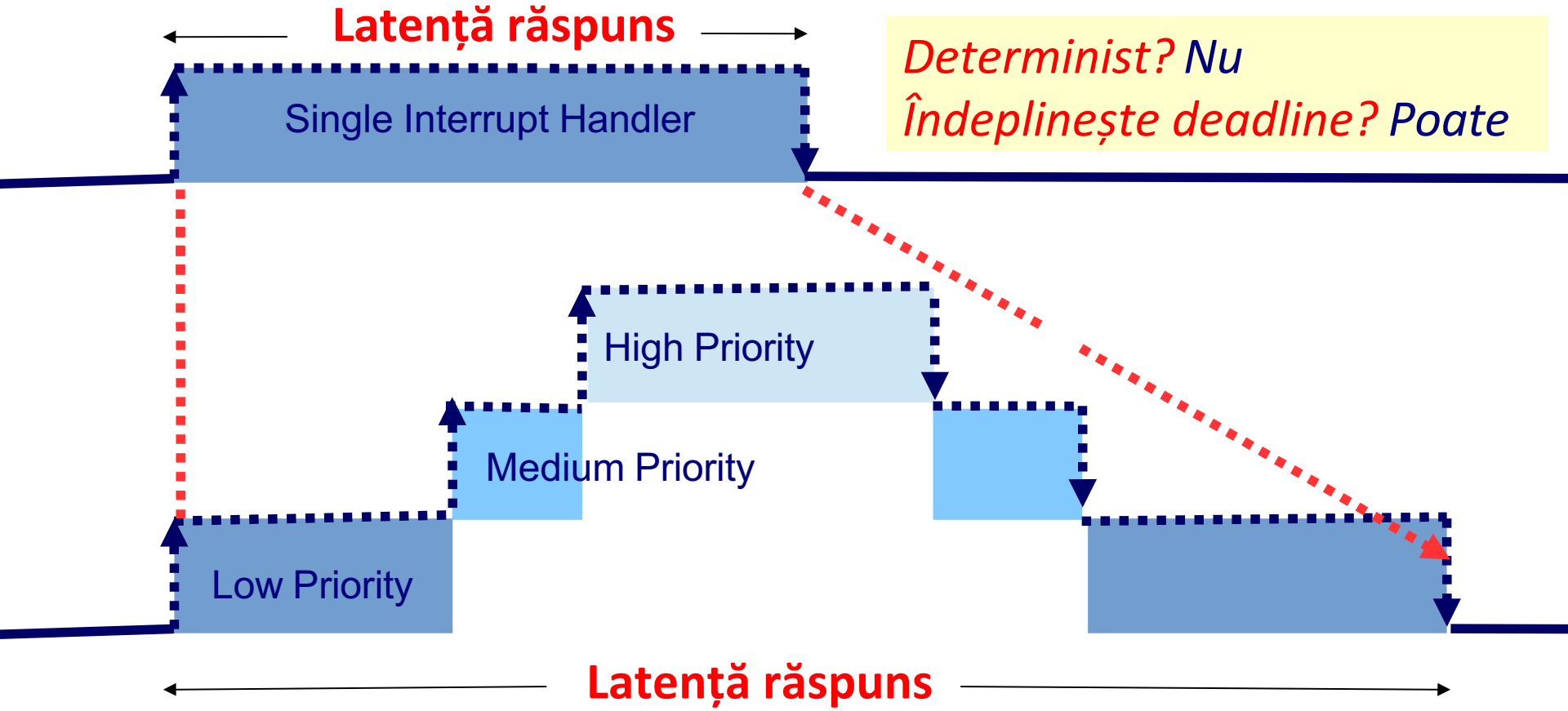
Stimul extern generează Hardware Interrupt Request (IRQ)



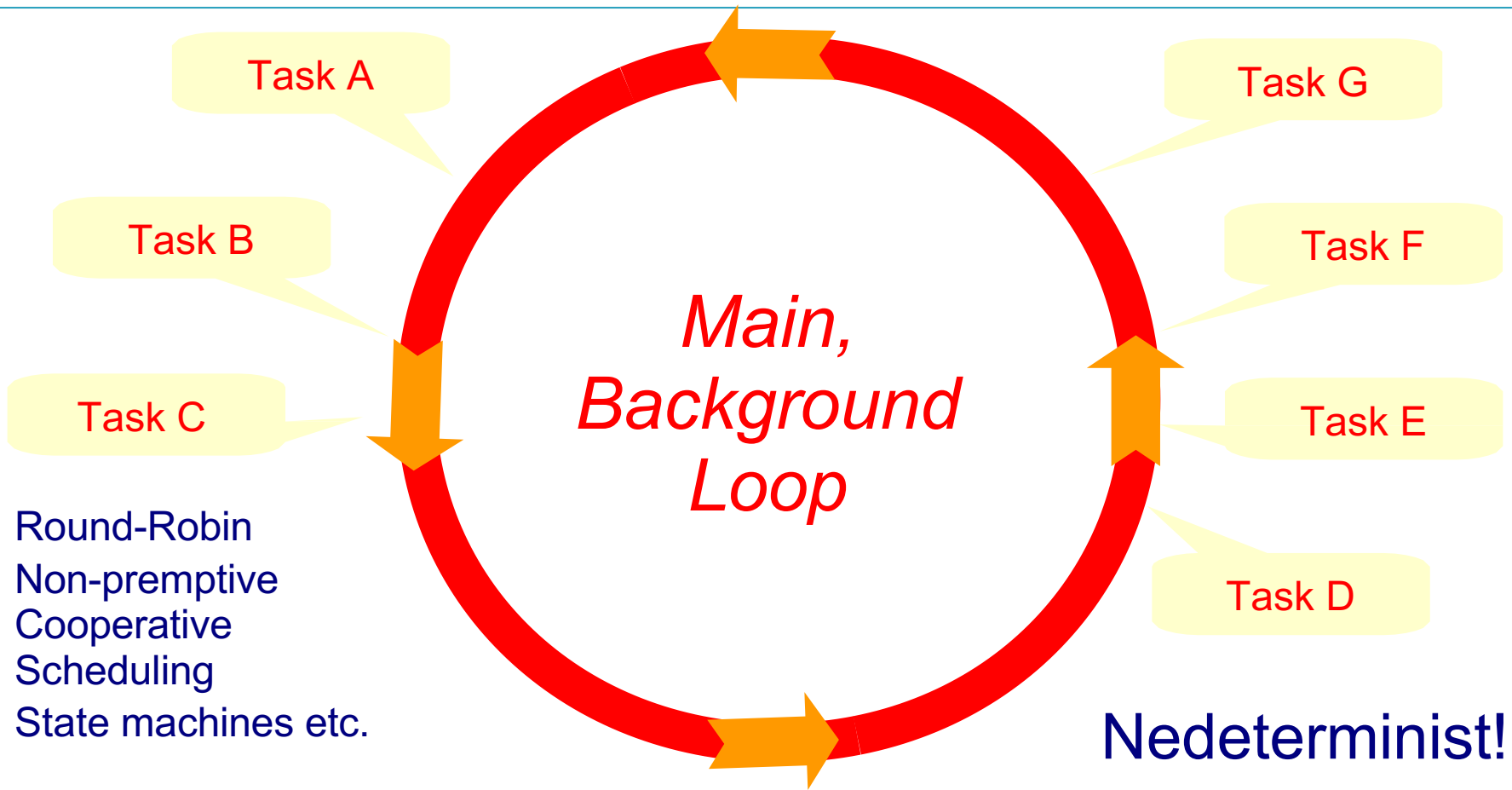
Foarte simplu

Deterministic dacă întreruperile nu sunt dezactivate niciodată!

Bare-metal, nested interrupts



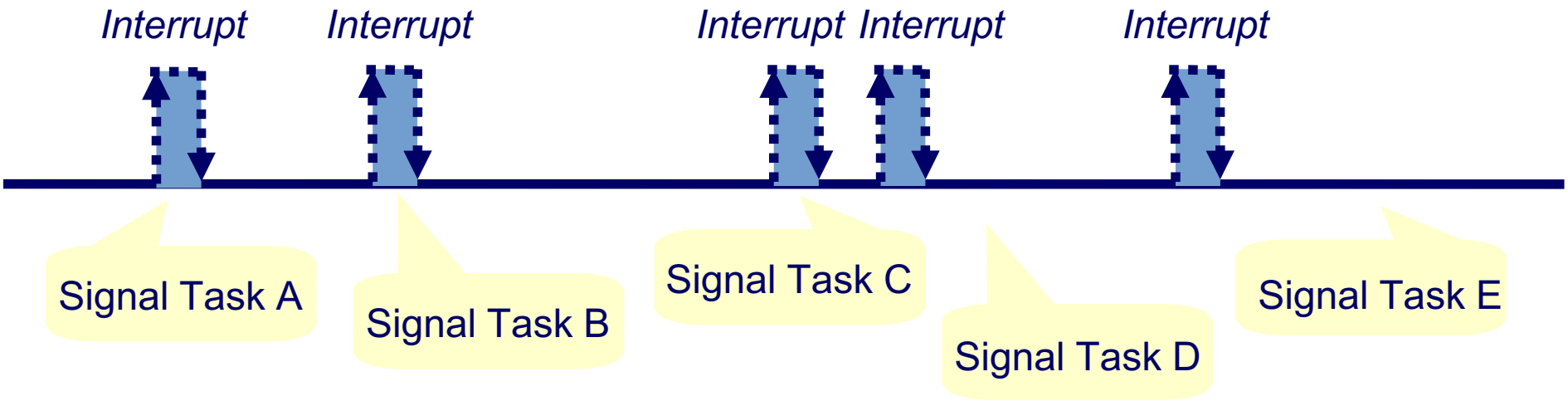
Bare-metal task scheduling



Round-Robin
Non-premptive
Cooperative
Scheduling
State machines etc.

Nedeterminist!

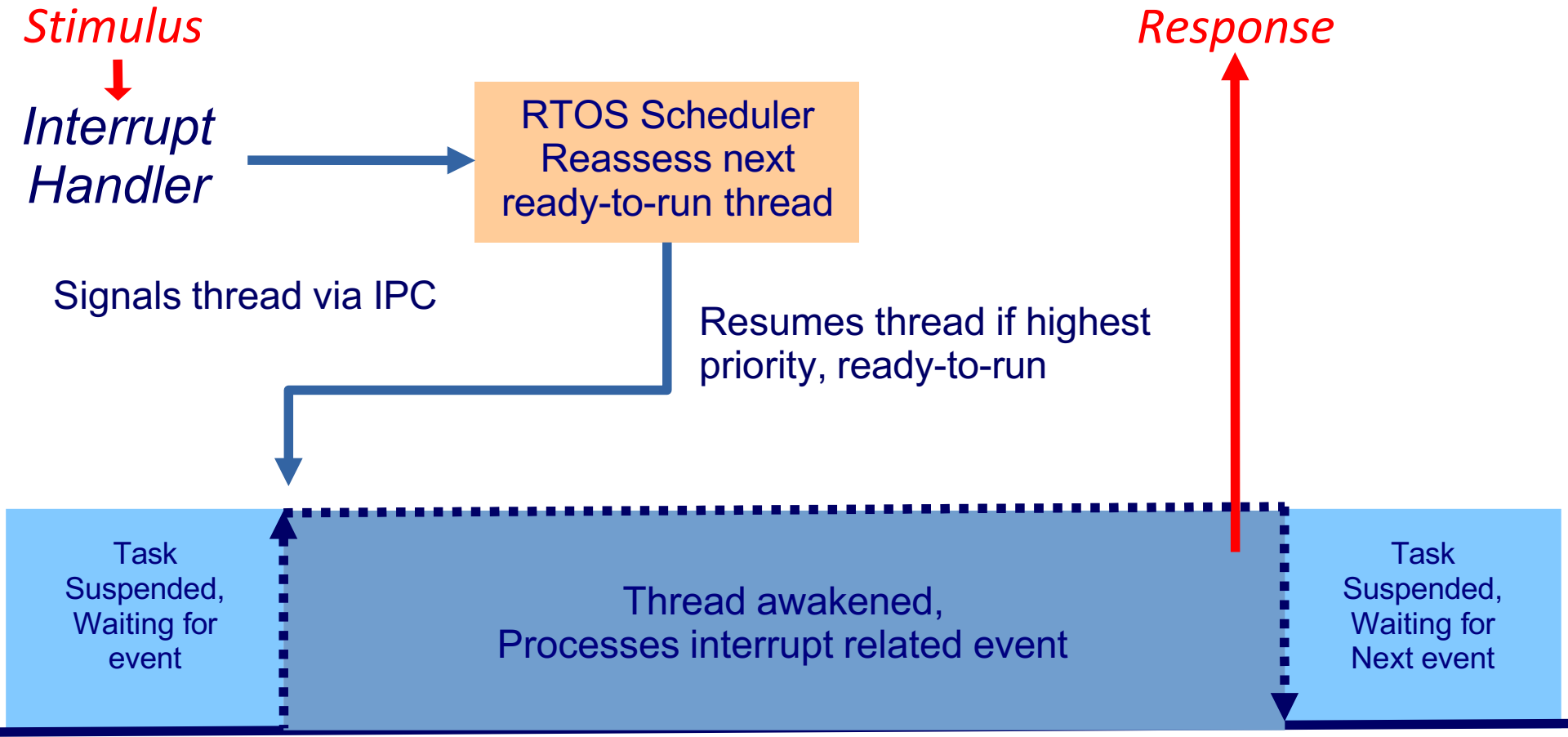
Întreruperi în contextul RTOS



RTOS way:

- Cod minim în rutina de tratare a întreruperii
- Rutinele de întrerupere doar semnalizează producerea unui eveniment către task-uri
- Planificatorul RTOS se ocupă de comportamentul real-time
- Sistem de priorități pentru task-uri, nu pentru întreruperi
- Nu se folosesc mai multe întreruperi (fără nested interrupts)

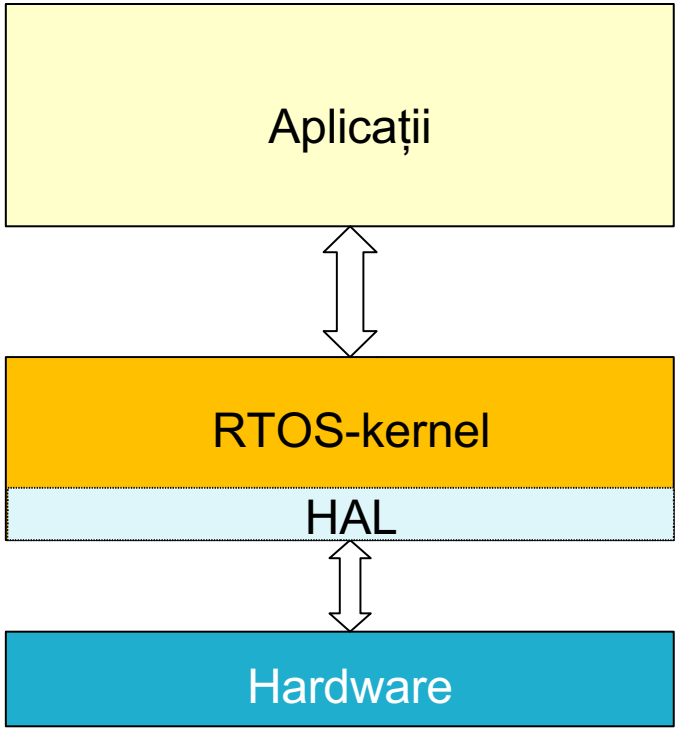
RTOS Interrupt Processing



Structura generală RTOS

Funcții kernel:

- Task Management
- Interrupt handling
- Memory management
- Exception handling
- Task synchronization
- Task scheduling
- Time management
- IO Management



Exemple

- VXWorks
- FreeRTOS
- NuttX
- ThreadX
- QNX
- Zephyr

FreeRTOS

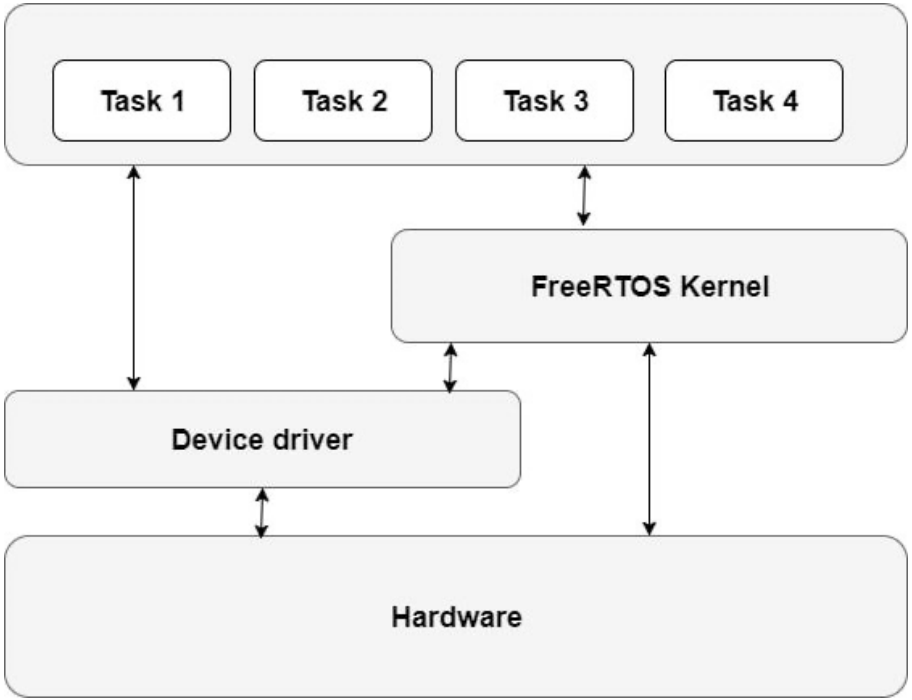
- Real-time operating system (RTOS) kernel pentru dispozitive embedded
- Rulează pe zeci de platforme
- Open Source, distribuit prin MIT License

FreeRTOS Features

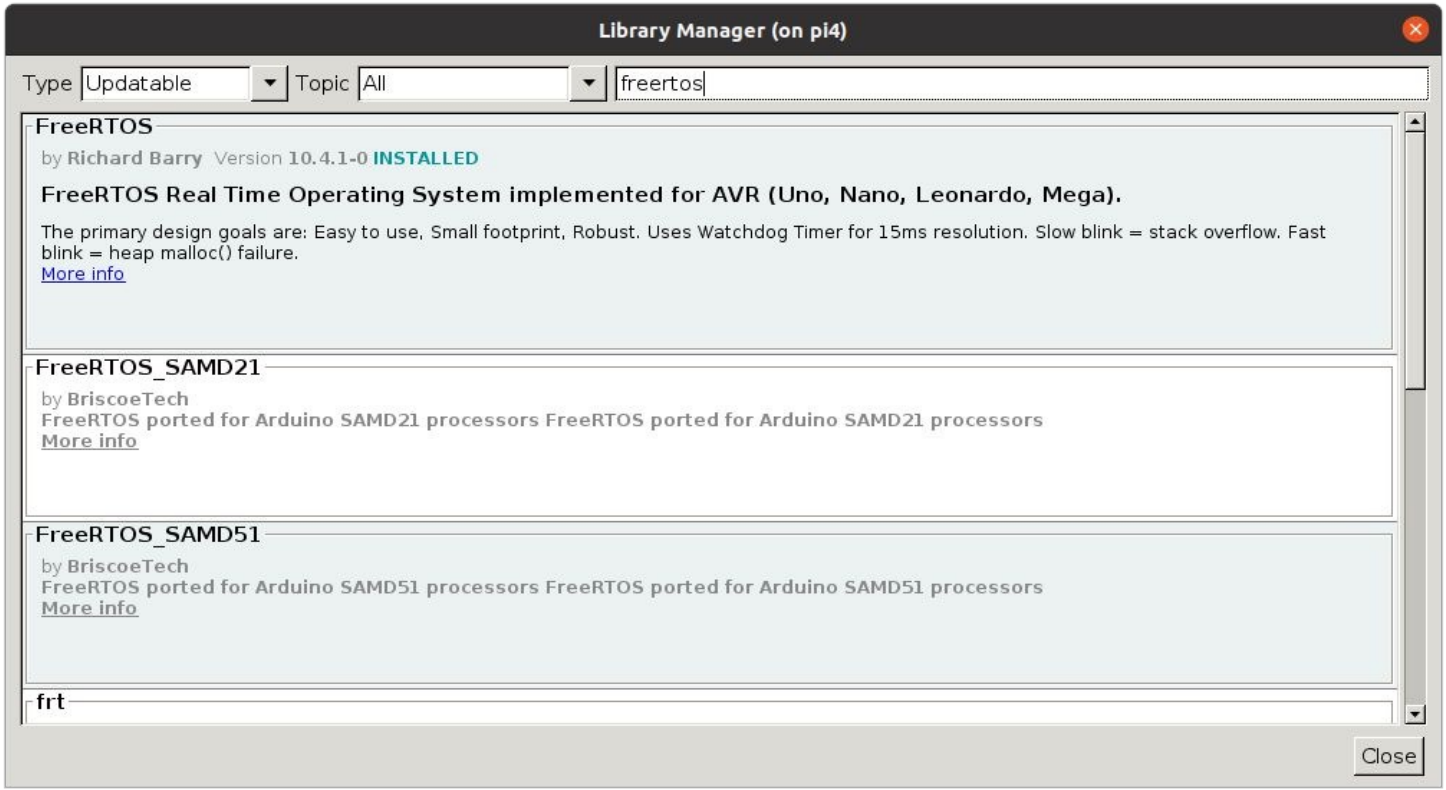
- Tasks
- Queues, Mutexes, Semaphores
- Direct To Task Notifications
- Stream & Message Buffers
- Software Timers
- Event Groups (or "Flags")
- Static vs dynamic memory
- Heap Memory Management
- Stack Overflow Protection
- Co-Routines (deprecated)
- Toate cele de mai sus sunt configurable la compilare

Arhitectura FreeRTOS

- Suport pentru rularea concurentă a mai multor fire de execuție
- Kernel de dimensiuni mici (kB)
- Task-urile au acces direct la resursele hardware
  - Memory management minim, cât poate să ofere arhitectura HW a procesorului embedded
  - Nu există mecanisme de protecție pentru accesul la memorie



Getting started - Arduino



Exemplu cod Arduino

```
#include <Arduino_FreeRTOS.h>

// define two tasks for Blink and AnalogRead
void TaskBlink(void *pvParameters);
void TaskAnalogRead(void *pvParameters);

// The setup function runs once when you press reset or power the board void setup() {

// Initialize serial communication at 9600 bits per second:
Serial.begin(9600);

while (!Serial) {
; // wait for serial port to connect. Needed for native USB on LEONARDO, MICRO, YUN etc.
```

Exemplu cod Arduino

```
// Now set up two tasks to run independently.
xTaskCreate(
TaskBlink   // Pointer to task entry function
,   "Blink" // Descriptive name
,   128     // Stack size
,   NULL    // Parameters - none
,   2      // Priority, with 3 being the highest, and 0 being the lowest.
,   NULL);  // Optional handle to created function

xTaskCreate( TaskAnalogRead
,   "AnalogRead"
,   128 // Stack size
,   NULL
,   1   // Priority
,   NULL);

// Now the task scheduler, which takes over control of scheduling individual tasks,
// is automatically started.
}
```

Exemplu cod Arduino

```
void loop()
{
// Empty. Things are done in Tasks.
}
```

Exemplu cod Arduino

```
void TaskBlink(void *pvParameters)  // This is a task.
{

/*
Blink: Turns on an LED on for one second, then off for one second, repeatedly.
*/

// Initialize digital LED_BUILTIN on pin 13 as an output. pinMode(LED_BUILTIN, OUTPUT);

for (;;) { // A Task never returns or exits.
    digitalWrite(LED_BUILTIN, HIGH);    // Turn the LED on (HIGH is the voltage level)
    vTaskDelay( 1000 / portTICK_PERIOD_MS); // Wait for one second
    digitalWrite(LED_BUILTIN, LOW); // Turn the LED off by making the voltage LOW
    vTaskDelay(1000 / portTICK_PERIOD_MS);  // Wait for one second
    }
}
```

Exemplu cod Arduino

```
void TaskAnalogRead(void *pvParameters) // This is a task.
{

/*
AnalogReadSerial:
Reads an analog input on pin 0, prints the result to the serial monitor.
*/

for (;;) {
    // Read the input on analog pin 0:
    int sensorValue = analogRead(A0);
    // Print out the value you read
    Serial.println(sensorValue);
    vTaskDelay(1);  // One tick delay (15ms) in between reads for stability
    }
}
```

Resurse utile

1. https://www.freertos.org
2. https://www.freertos.org/Documentation/RTOS_book.html
3. https://forums.freertos.org
4. https://github.com/feilipu/Arduino_FreeRTOS_Library
5. https://doc.qt.io/QtForMCUs/qtul-using-with-freertos.html

NuttX

- RTOS pentru microprocesoare/microcontrolere
- Rulează pe sisteme de la 8 la 64 de biți
- Licență Apache 2.0
- Implementare POSIX-compliant (Unix-like)
- *Tiny-Linux*: implementează diferite sisteme Linux

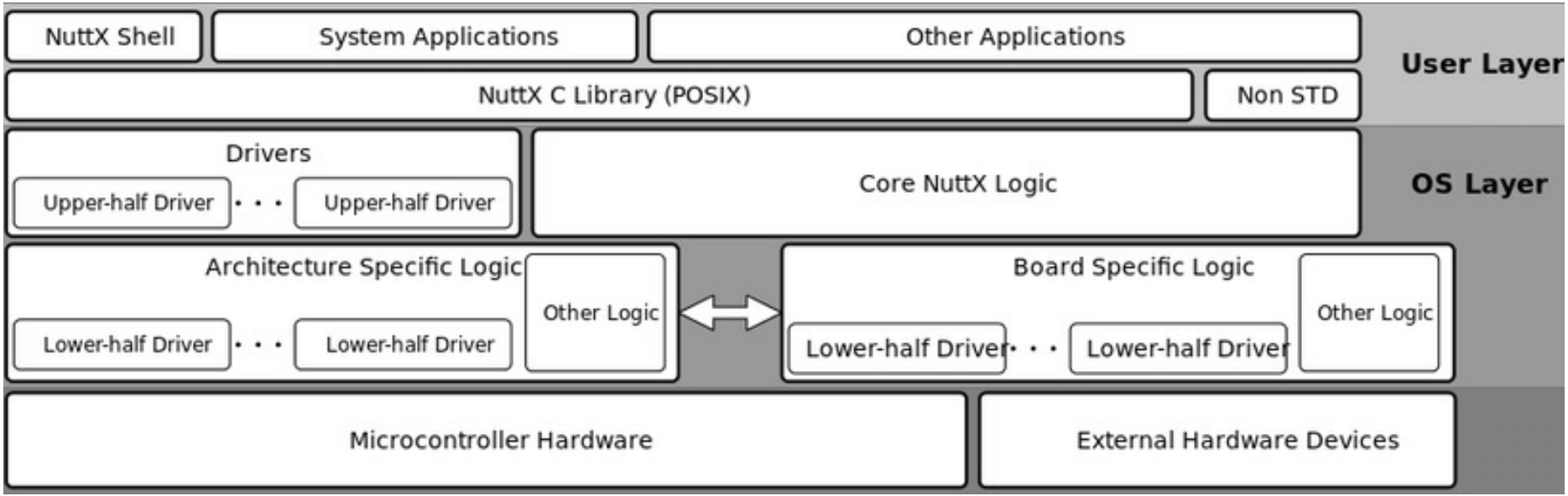
NuttX

- Gregory Ellis Nutt

Unde este folosit?

- Sute de microprocesoare și microcontrolere
  - De la AVR, ARM, RISC-V la x86
- Produse Sony, Xiaomi, Samsung, Motorola, Fitbit...

Arhitectura NuttX



Caracteristici

- Complet preemptiv
- Moduri de lucru cu memoria: FLAT, protected și Kernel
- Virtual File System (VSF): drivers & mountpoints
- Kernel module loading
- Multiprocesare simetrică (SMP)
- Real-time scheduling (RR, FIFO, Sporadic)
- Tickless operation

Caractieristici

- Pseudo-terminals (PTY)
- IO redirection (>, >>, ...)
- Debug logging nativ (ERR, WARN, INFO) și loguri de sistem (syslog)
- Power management
- Toate API-urile sunt POSIX
- Filesystem (FAT16, FAT32, NFS, BINFS, SmartFS, ROMFS etc.)

Networking

- Interfețe multiple de rețea (Ethernet, Wifi, SLIP etc.)
- IPv4, IPv6, TCP, UDP, ICMP etc.
- Unix sockets
- IEEE802.11 (WiFi) full MAC
- IEEE802.15.4 (MAC, 6LoWPAN)
- SLIP (Serial Line IP), PPP (LTE GSM)

Features

- Interfețe grafice – framebuffer (similar Linux)
  - Server grafic NX (echivalent X Server)
  - Gamă largă de display-uri (LCD, OLED, 7-segment, matriceal etc.)
- USB Host
  - CDC/ACM
  - Mass Storage
  - HID Mouse & Keyboard
  - USB Hub

NuttX Threads

- Fiecare thread are propria stivă
- Fiecare thread are o prioritate de execuție dată de sistemul de operare
- Fiecare thread este membrul unui “task group”
- Resurse partajate (ca un proces Linux)
- Poate să aștepte după evenmente sau disponibilitatea resurselor
- Thread-urile comunică prin IPC: POSIX Message Queues, Signals, Counting semaphores, etc.
- Standard / Linux compatible
- NuttX permite utilizarea standard IPCs din interrupt handlers

Resurse utile

- https://www.youtube.com/c/NuttXChannel
- https://nuttx.apache.org/
- https://nuttx.apache.org/docs/latest/platforms/xtensa/esp32/index.html
- https://ocw.cs.pub.ro/courses/si

Call to action!

- Alăturați-vă comunității și contribuiți activ la dezvoltarea NuttX pe ESP32 și pe platformele hardware din facultate (Sparrow, Hacktor Watch etc.)!
  - Comunitatea NuttX @ UPB

Sparrow

Hacktor Watch