Static (off-line) scheduling

Scheduling table: implementation option 2

- Implementing the scheduling table **with/without preemption**
  - The scheduling table is a parameter to a **dispatcher**
    - **Interrupts** summon the dispatcher which reads the table and starts the corresponding jobs
    - The table must be encoded in the program
  - In case of preemptions, it is necessary to save/restore the execution context of the preempted task
    - Usually Real-Time OS
      - Tasks => processes or threads
    - Save/restore context time must be accounted for
    - Cache-related preemption delays apply
    - Preemptions increase schedulability
    - Careful with locks and critical sections!

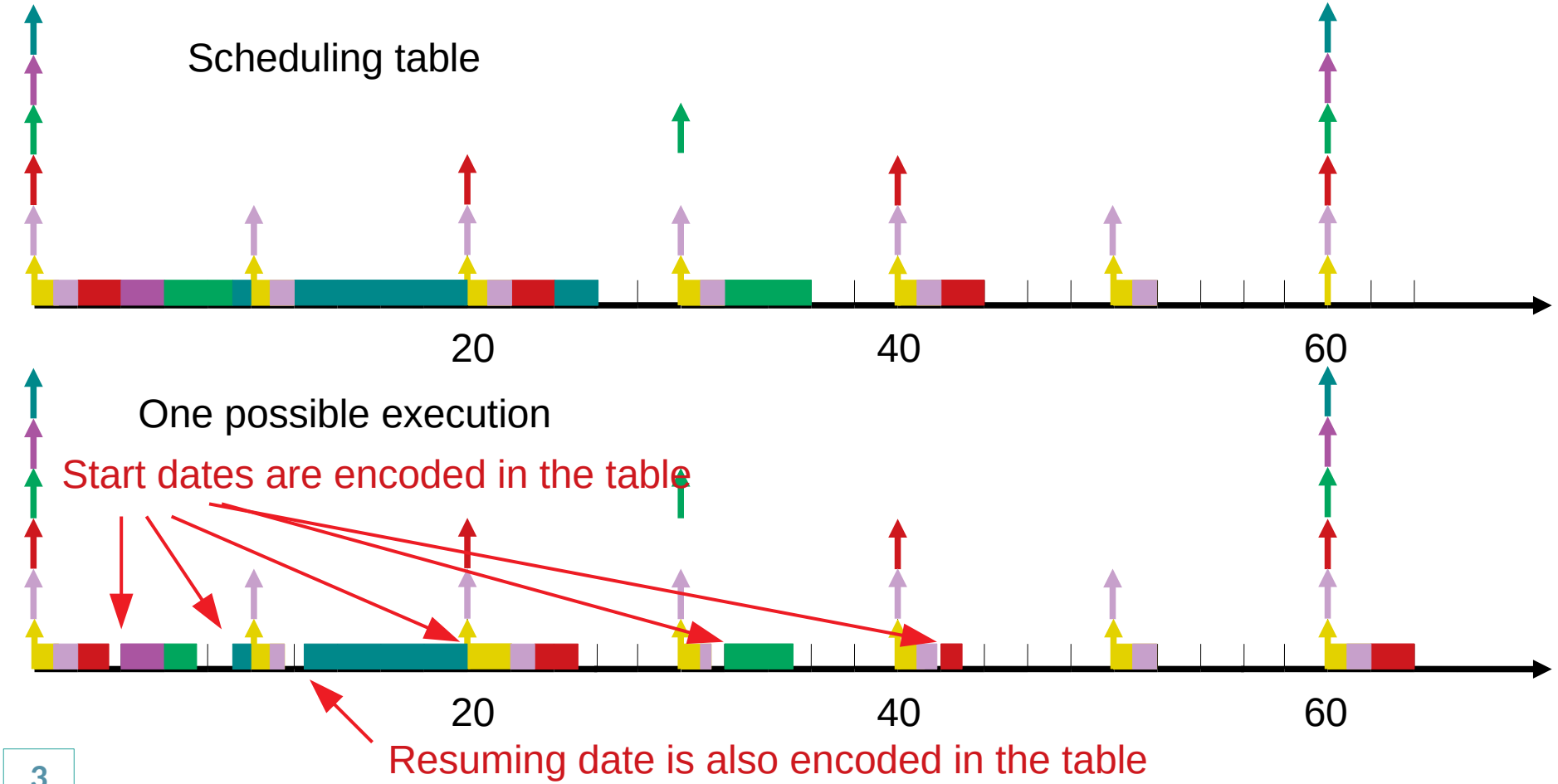
Static (off-line) scheduling

Scheduling table: Time Triggered Architecture

- TTA execution model : single source of interrupts => timer
  - Jobs starting dates strictly follow the scheduling table
  - Preemptions occur on timer interrupts
  - Preempted tasks can resume either on a timer interrupt or as soon as their preempting task is finished
    - Depends on the implementation of the **dispatcher** AND of the **interrupt handler**
- TTA is a subset of event-triggered systems
  - In ET systems, multiple sources of interrupts e.g. timers, software, sensors, etc.
  - Interrupts do not have to be periodic (however it makes it harder – or impossible – to verify the schedulability)

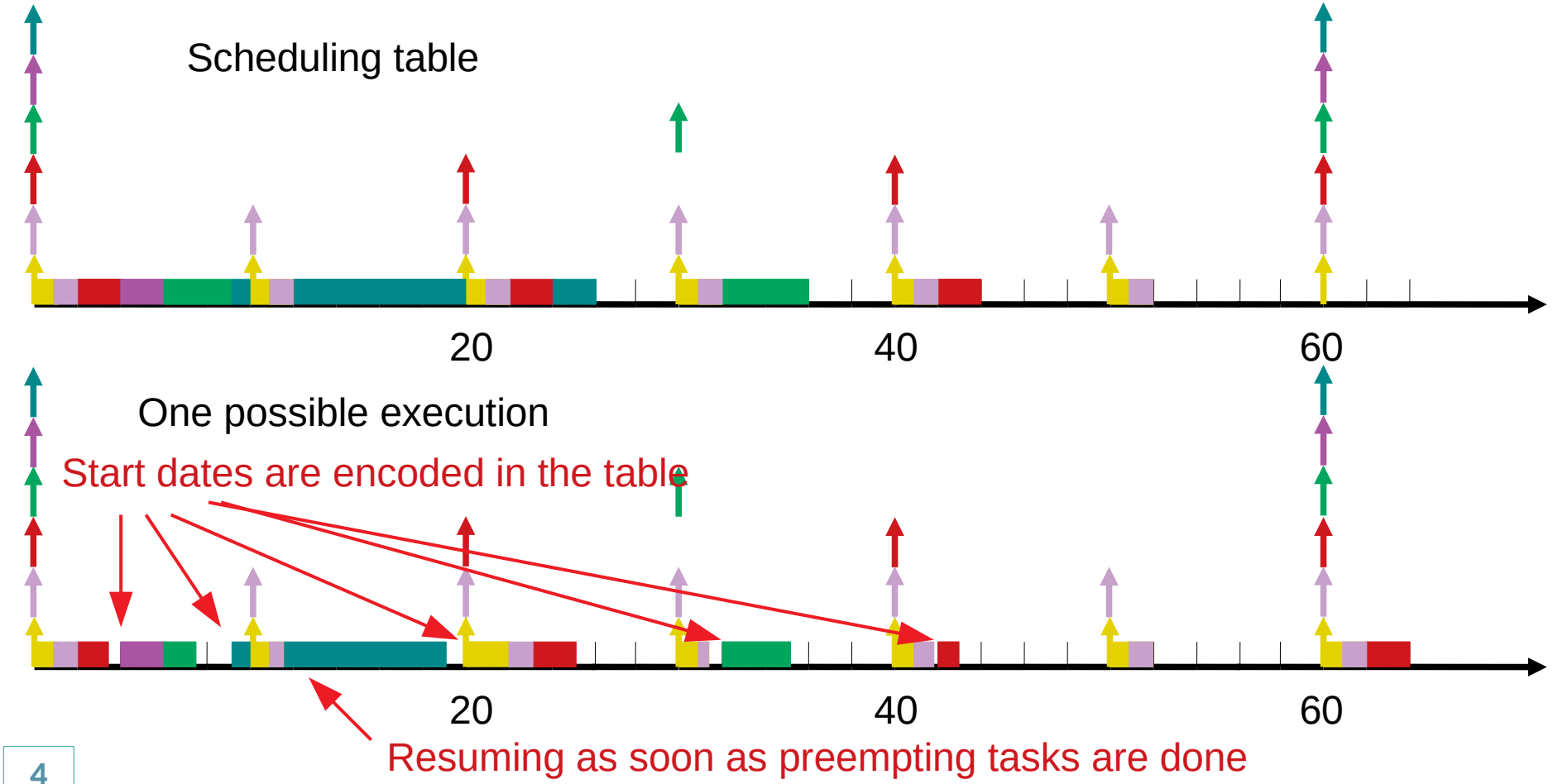
Static (off-line) scheduling

Scheduling table: TTA implementation



Static (off-line) scheduling

Scheduling table: TTA implementation



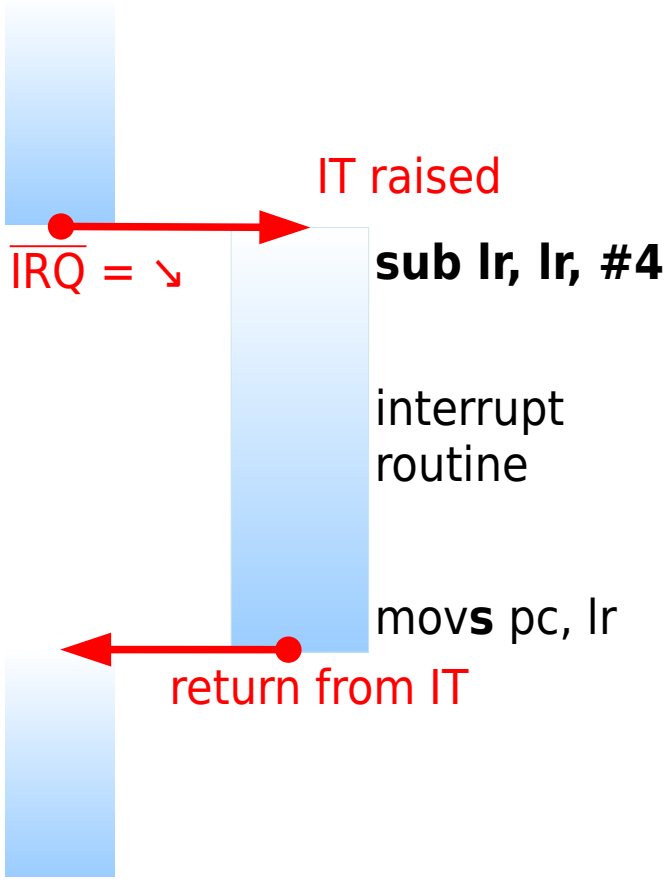
Static (off-line) scheduling

Scheduling table: TTA implementation (basic)

- Interrupts:
  - Asynchronous management on I/O
  - On peripheral event:
    - Current program stopped
    - Execution of interrupt handling routine
  - Realisation of the interrupt:
    - Current context is saved
    - Handling routine is executed
    - Saved context is reloaded
    - Current program goes on as if nothing happened

Static (off-line) scheduling

Scheduling table: TTA implementation (basic)



Static (off-line) scheduling

Scheduling table: TTA implementation (basic)

- Interrupts on ARM: ARM exception table
  - Located at memory start (address 0x00000000)
  - IRQ entry redirects to Advanced Interrupt Controller (AIC)

Address	Meaning
0x 0000 0000	Reset
0x 0000 0004	Undefined instruction
0x 0000 0008	Software interrupt
0x 0000 000C	Prefetch Abort
0x 0000 0010	Data Abort
0x 0000 0014	Reserved
0x 0000 0018	IRQ
0x 0000 001C	FIQ

Static (off-line) scheduling

Scheduling table: TTA implementation (basic)

- Interrupts on ARM: Advanced Interrupt Controller (AIC)
  - Multiplexer of ITs (up to 32)
  - Provides an IT vector for each peripheral
  - Register AIC_IVR is hardware managed and contains the address of the handler for the highest priority pending interrupt
  - AIC_SVR is a vector containing for each peripheral the address of the corresponding interrupt handler
  - AIC_SMR is a vector containing the priorities of each peripheral interrupts
  - Other registers can be used to clear pending interrupts and enable/mask interrupts at the AIC level

Static (off-line) scheduling

Scheduling table: TTA implementation (basic)

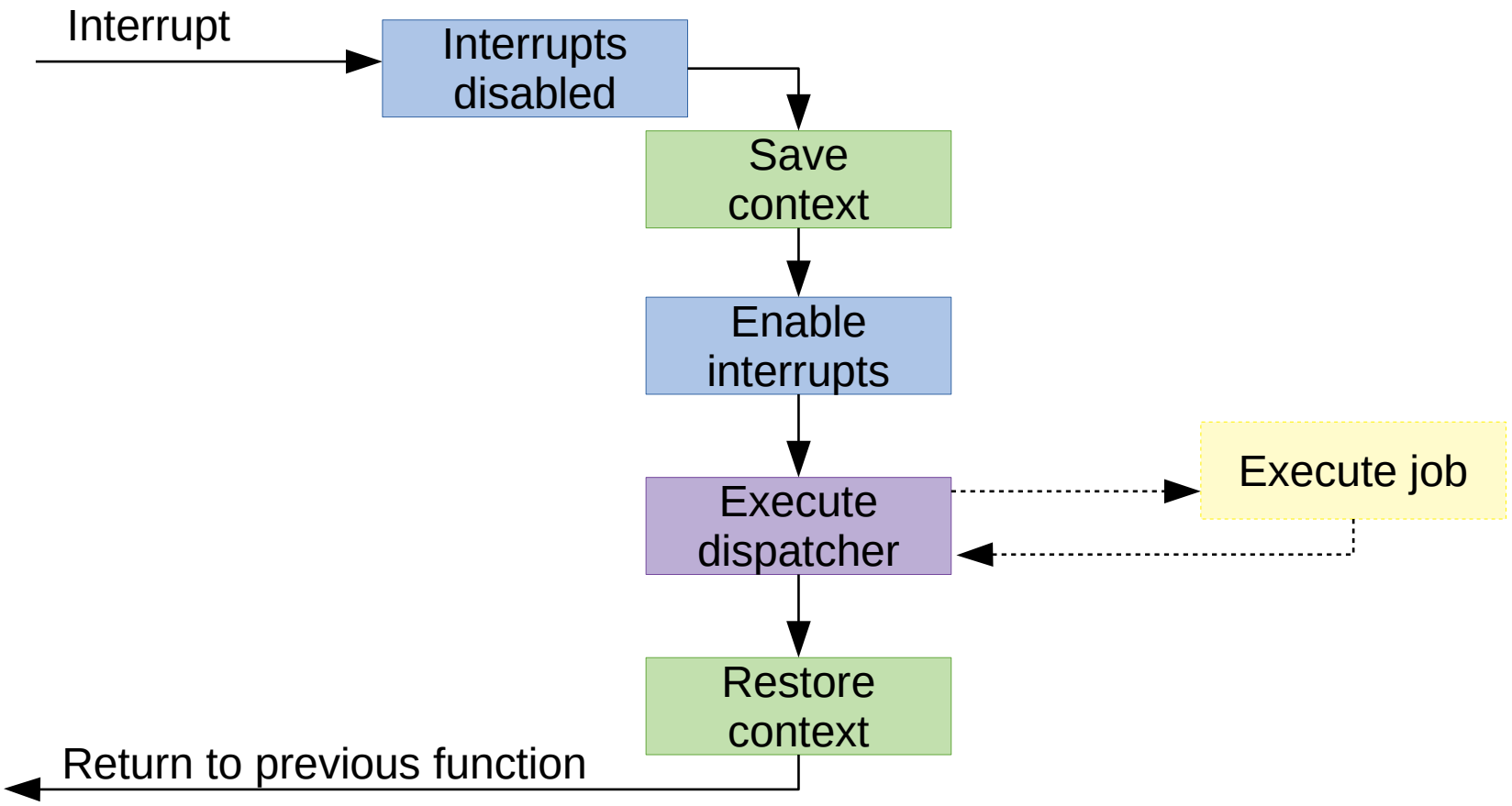
- In the main function
  - Configure timer and AIC
    - Period
    - Interrupt handler routine
    - Enable interrupts in timer and AIC

```
// initialize main timer
  TC0_CCR = TC_CLKDIS; /* disable timer */
  TC0_CMR = TC_DIV | TC_CPCTRG; /* configure timer counter and restart when CV = RC */
  TC0_RC = ONE_SECOND; /* delay = 1s (Master clock frequency / TC_DIV) */
  TC0_IER = TC_CPCS; /* generate interrupt when CV = RC */
// initialize Advanced Interrupt Controller
  AIC_SVR[ID_TC0] = (uint32_t)handle_tc0;  /* configure interrupt handler */
  AIC_SMR[ID_TC0] = 0; /* set interrupt priority */
  AIC_ICCR = 1 << ID_TC0; /* Clear potentially pending interrupt */
  AIC_IECR = 1 << ID_TC0; /* Enable interrupts in AIC */
  TC0_CCR = TC_SWTRG | TC_CLKEN; /* Enable TC0 and reset TC0 value */
```

  - Enable interrupts in processor
  - Execute a busy-waiting loop

Static (off-line) scheduling

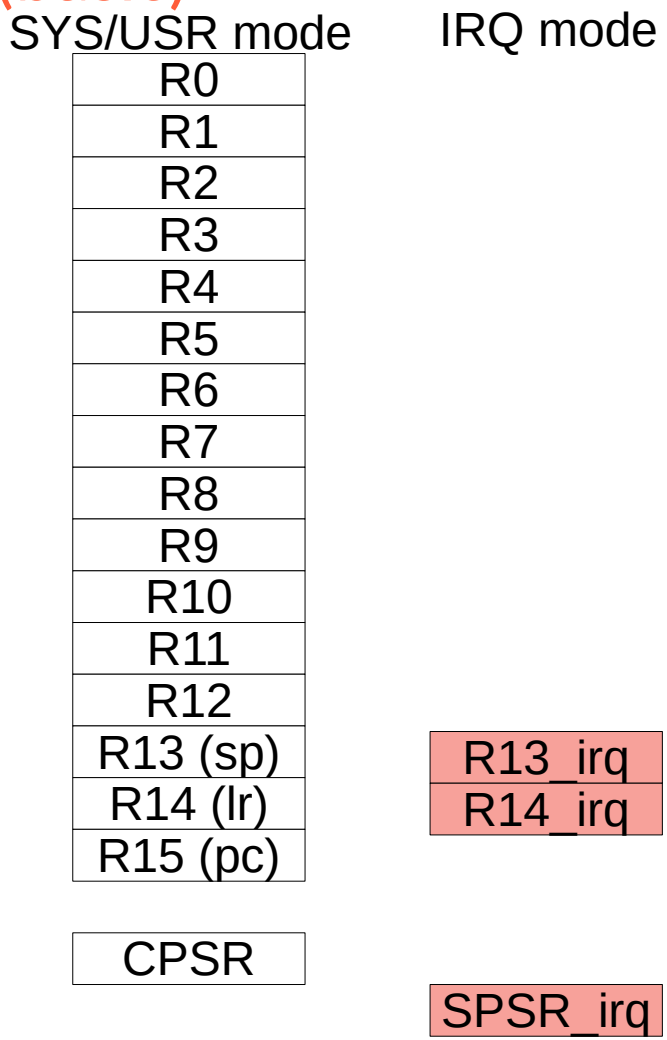
Scheduling table: TTA implementation (basic)



Static (off-line) scheduling

Scheduling table: TTA implementation (basic)

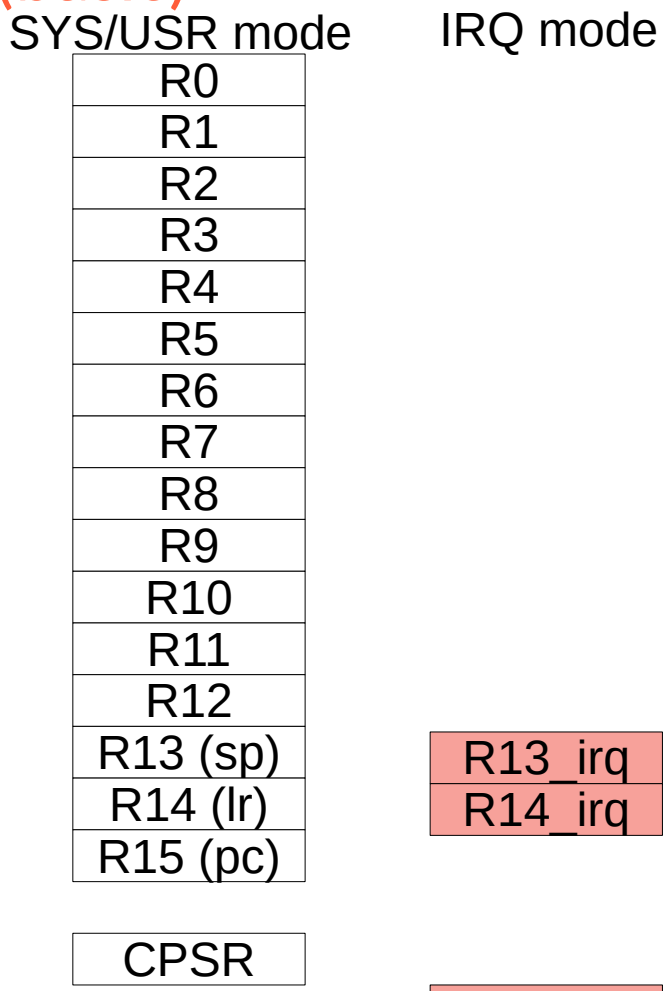
- In the interrupt handler routine
  - At first: IRQ mode
    - Interrupts disabled
    - Special registers
      - pc_sys is stored in lr_irq
      - CPSR is copied in SPSR_irq
  - In case of nested preemptions:
    - Save return address on irq stack
      - stack_irq ← lr_irq – 4
    - Save SPSR on irq stack
    - Switch to System mode



Static (off-line) scheduling

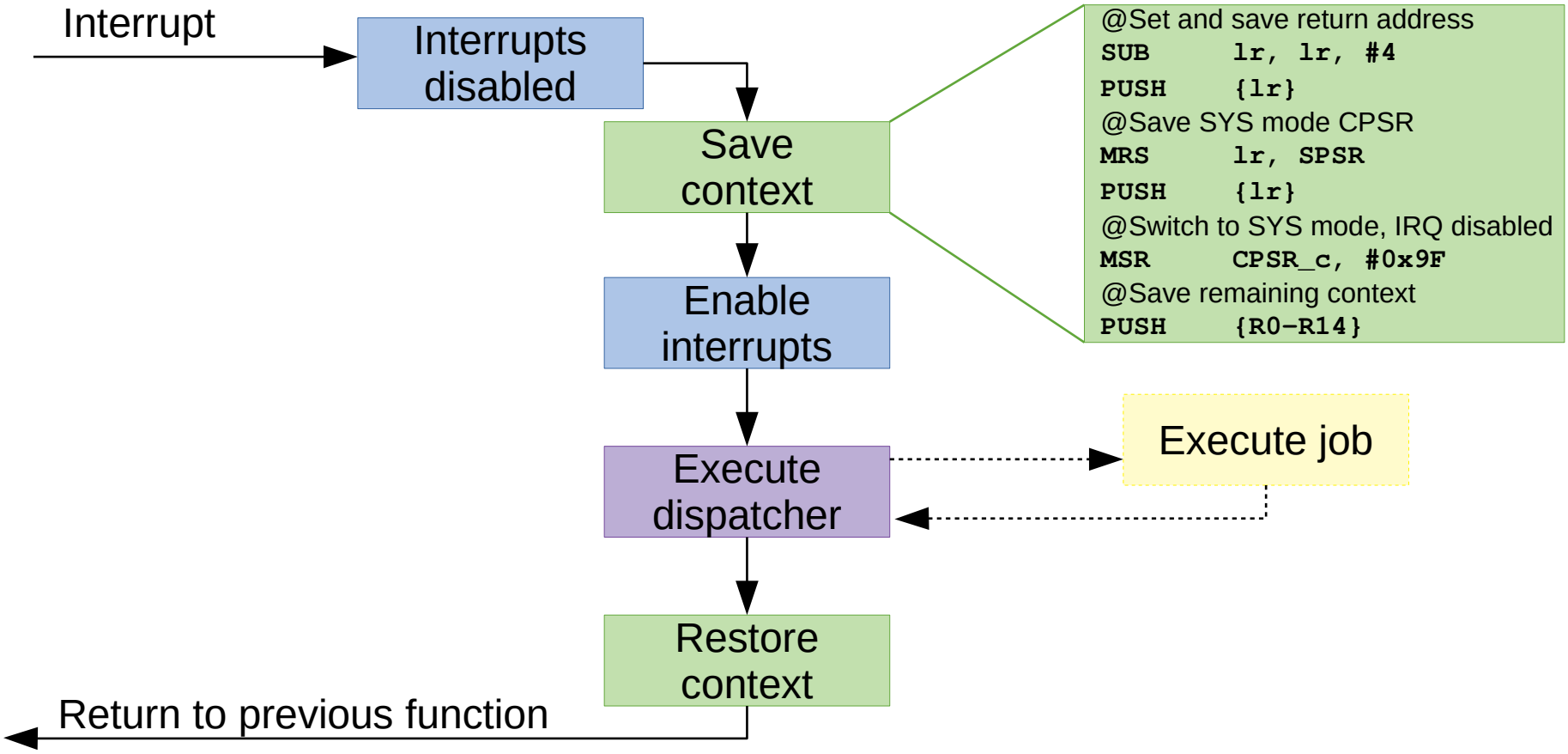
Scheduling table: TTA implementation (basic)

- In the interrupt handler routine (ctd.)
  - Save context (r0-r14)
  - Enable interrupts (for preemption)
  - Branch to dispatcher
  - Restore context (r0-r14)
  - Return from interrupt
    - Switch to irq mode
    - Restore SPSR from irq stack
    - Restore lr_irq from irq stack
    - pc ← lr_irq and copy SPSR to CPSR



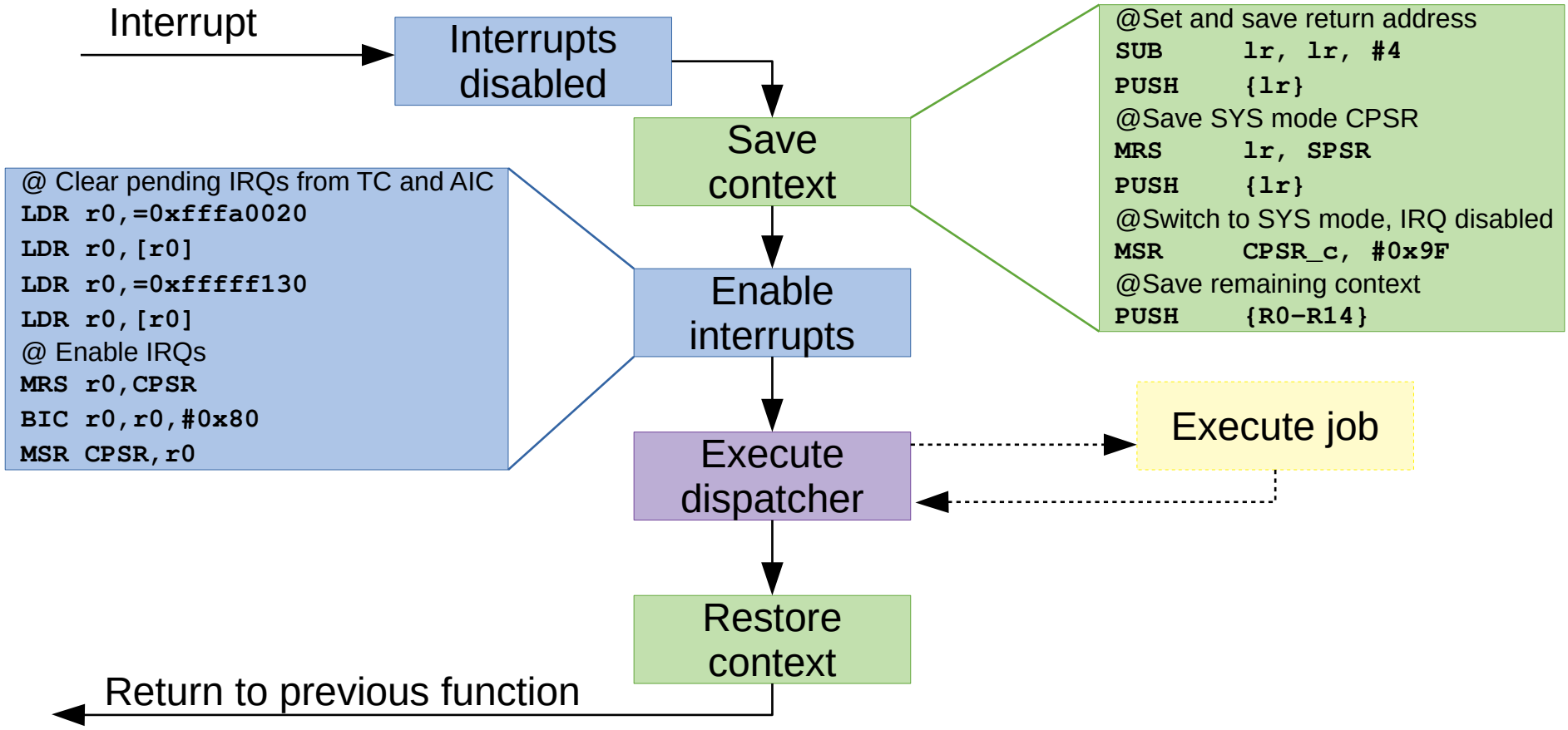
Static (off-line) scheduling

Scheduling table: TTA implementation (basic)



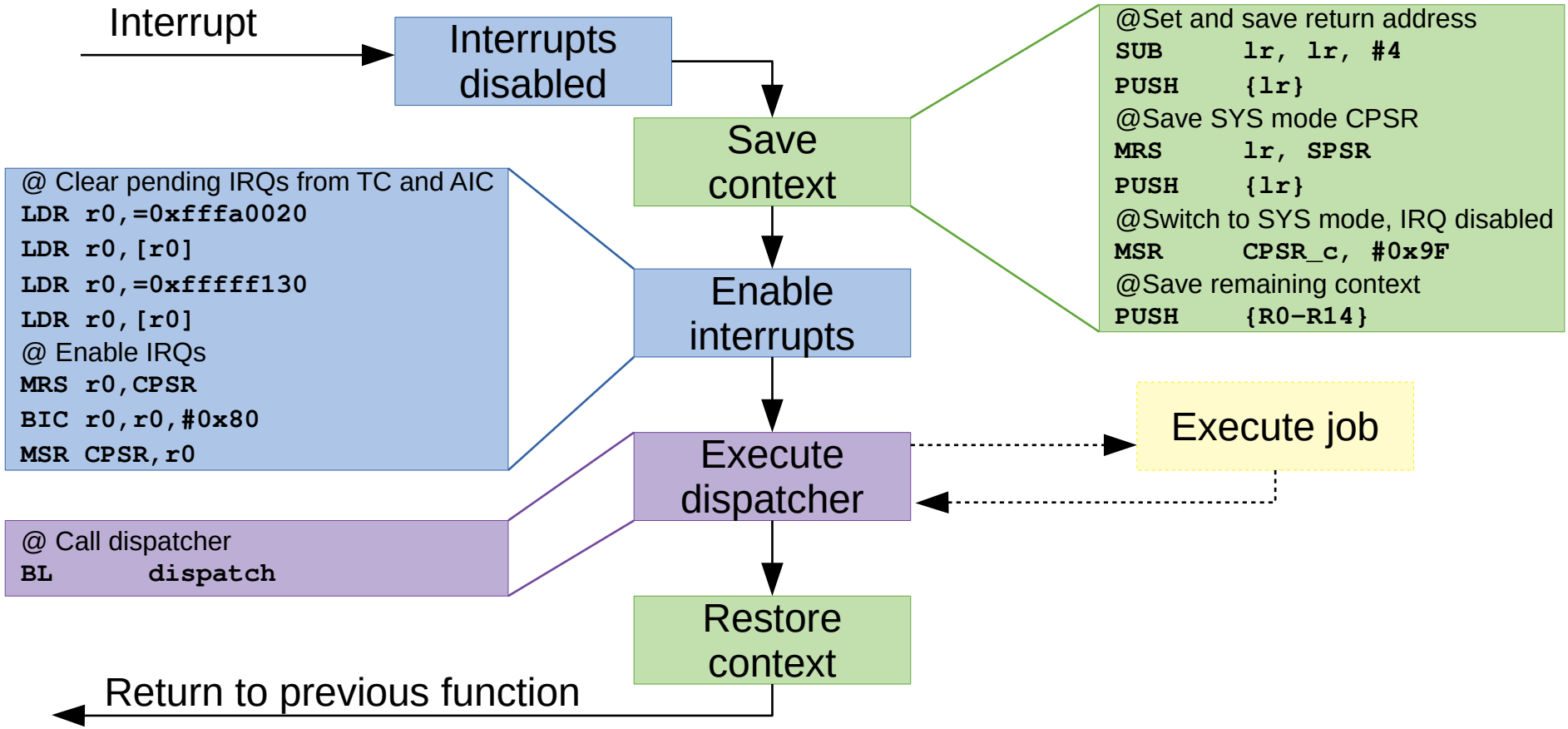
Static (off-line) scheduling

Scheduling table: TTA implementation (basic)



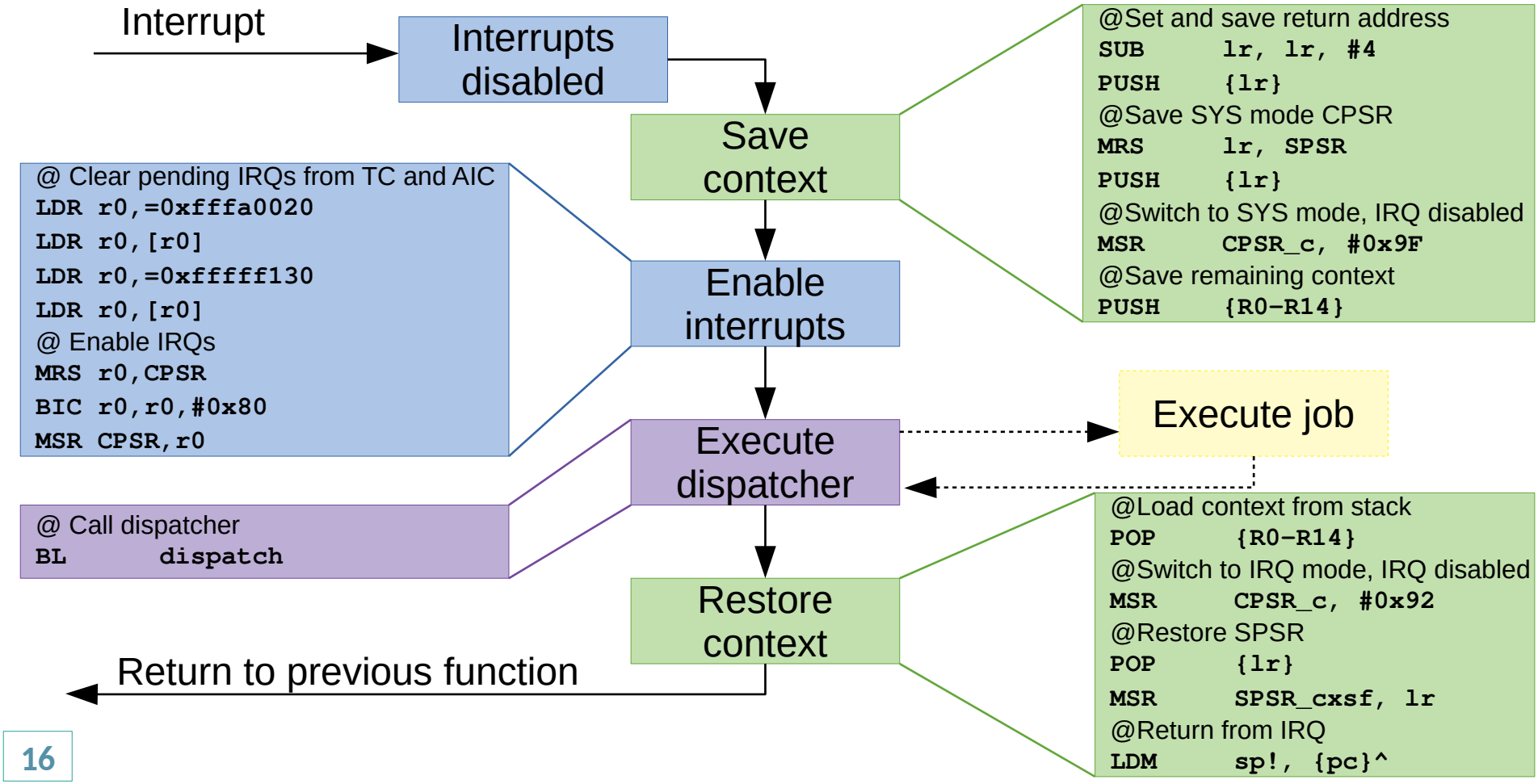
Static (off-line) scheduling

Scheduling table: TTA implementation (basic)



Static (off-line) scheduling

Scheduling table: TTA implementation (basic)



Static (off-line) scheduling

Scheduling table: TTA implementation (basic)

- In the dispatcher
  - Internal representation of time (variable for date)
    - Managed internally by the dispatcher (incremented at each interrupt, modulo the hyperperiod)
  - Internal representation of schedule (scheduling table)
    - If a job starts at current date, call corresponding function
    - Otherwise, do nothing

Static (off-line) scheduling

Practical work:

- Develop a dispatcher in C
  - Input: textual description of the scheduling table
  - Interface with the AT91 simulator
    - Interrupt handler is provided
  - Test it using functions that e.g. light up the LEDs in the simulator
- If necessary, adapt your static scheduler (or use a script) so that it produces a text file (or .h) that can be used as input of the dispatcher