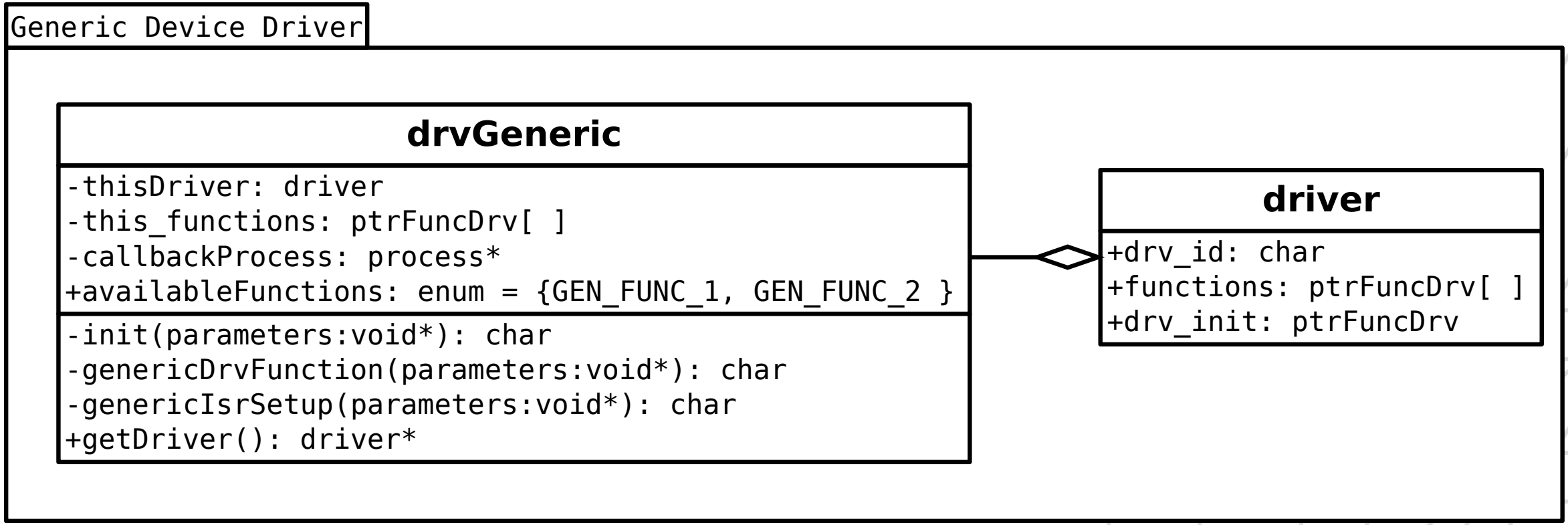
Embedded Operating Systems

Prof. Otávio Gomes
otavio.gomes@unifei.edu.br
/otavio-gomes

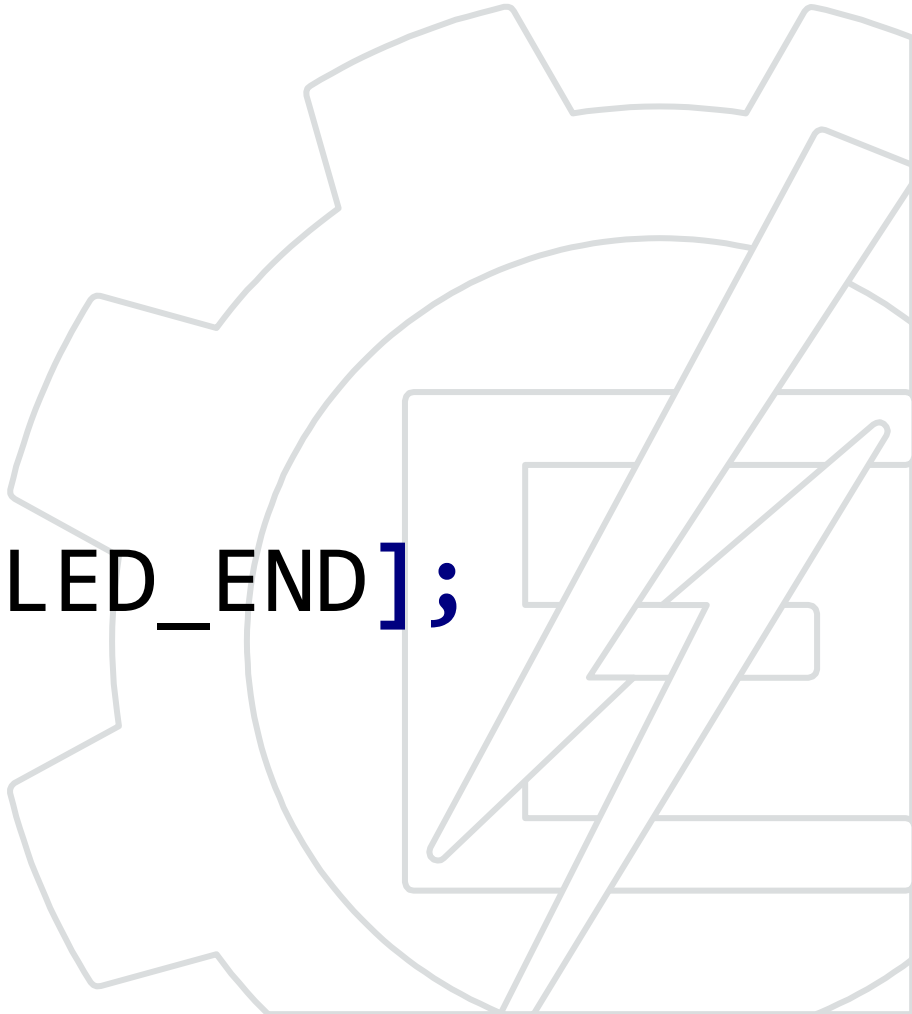
Driver overview

Driver example



+ visible
- invisible

```
#include "kernel.h"
#include "pic18f4520.h"
#include "drvGeneric.h "

static driver meu_cartao;

static ptrFuncDrv my_funcs[LED_END];
```



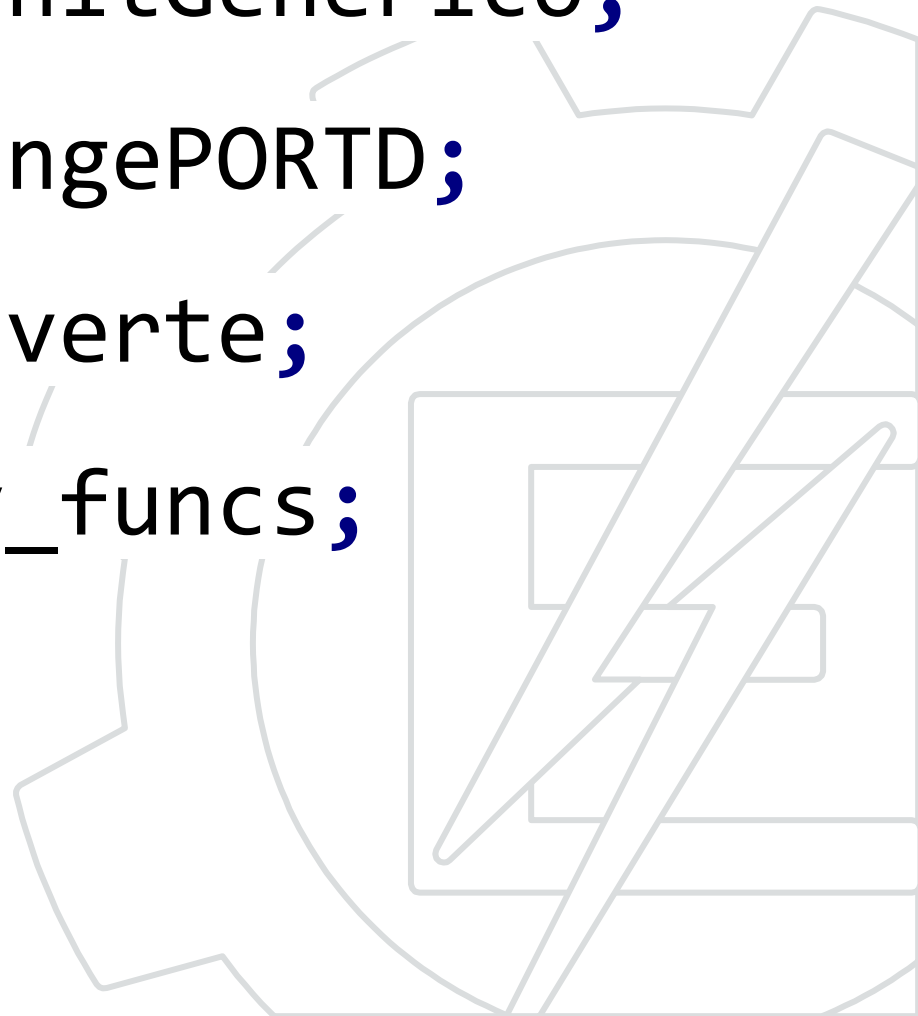
```
#ifndef drvGeneric_h
    #define drvGeneric_h
    #include "dd_types.h"

    //lista de funções do driver
    enum {
        LED_SET,LED_FLIP, LED_END
    };

    //função de acesso ao driver
    driver* getGenericDriver(void);

#endif // drvGenerico_h
```

Driver example

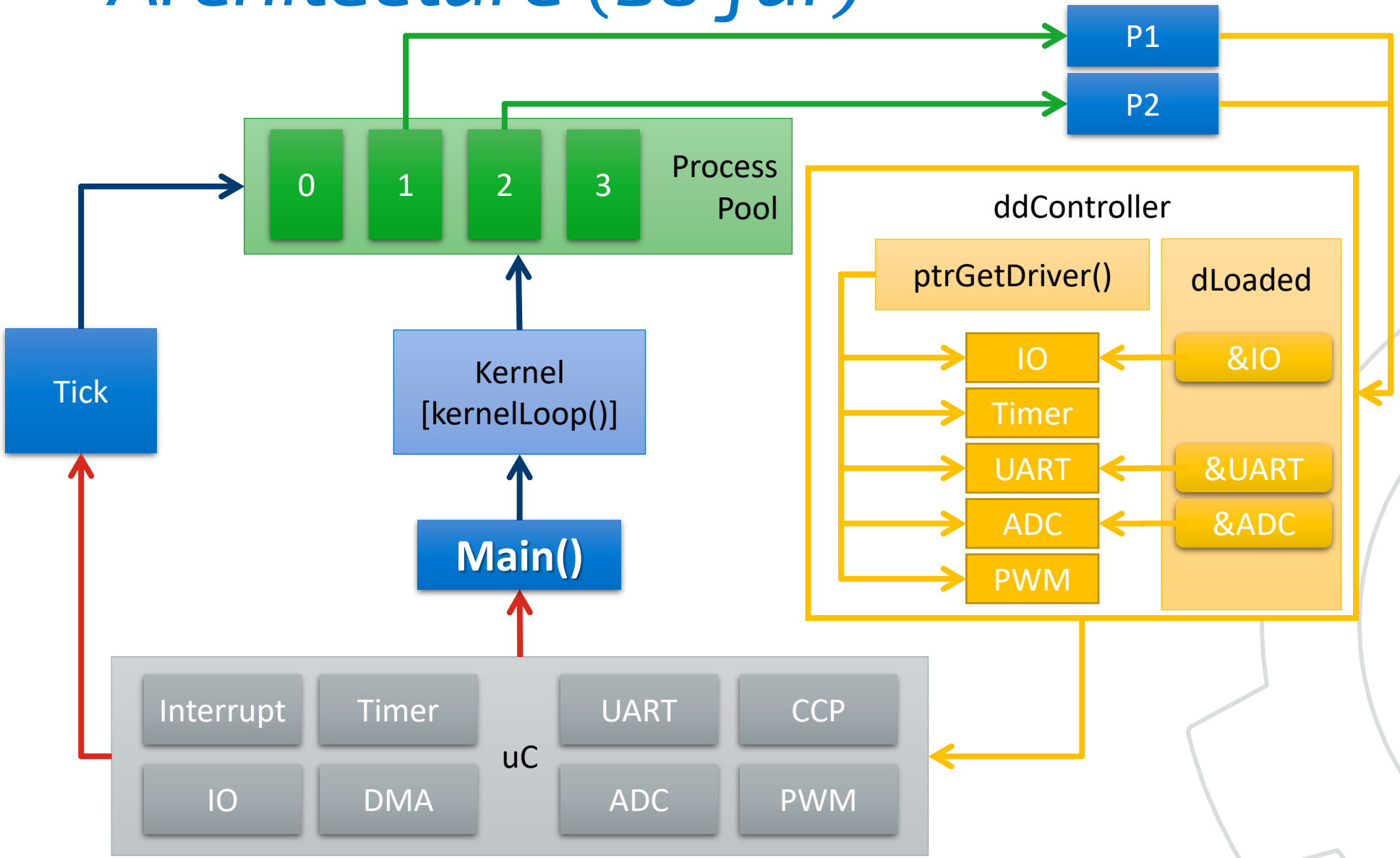
```
char changePORTD(void *parameters) {
    PORTD = (char) parameters;
    return SUCCESS;
}

char inverte(void * parameters){
    PORTD = ~PORTD;
    return SUCCESS;
}

char initGenerico(void *parameters) {
    TRISD = 0x00; PORTD = 0xFF;
    meu_cartao.id = (char) parameters;
    return SUCCESS;
}
```

Driver example

```
driver* getGenericDriver(void) {
    meu_cartao.initFunc = initGenerico;
    my_funcs[LED_SET] = changePORTD;
    my_funcs[LED_FLIP] = inverte;
    meu_cartao.funcoes = my_funcs;
    return & meu_cartao;
}
```



Architecture (so far)



- Pattern that allows kernel integration
- Could define privileged access with control and priority
- Process can't direct access hardware
- MMU can give more security to memory map access
- Could restart only one driver
- ddController block/enable the access to devices (segmentation fault, buffer overflow, malicious injection)

Abstraction layers

Abstraction layer

“Software that translates a high-level request into the low-level commands required to perform the operation.

The most common abstraction layer is the programming interface (API) between an application and the operating system.

High-level calls are made to the operating system, which executes the necessary instructions to perform the task.”

Source: https://www.pcmag.com/encyclopedia/term/abstraction-layer

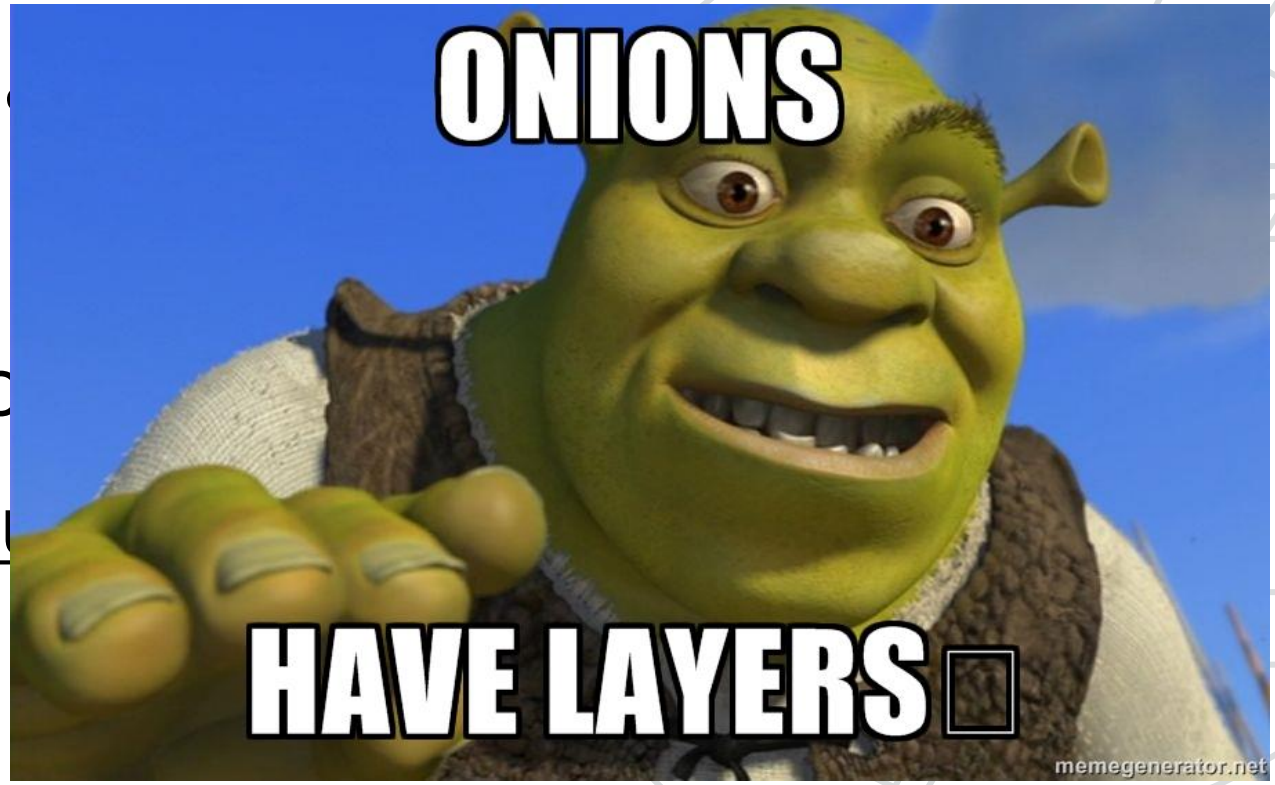
Abstraction layer

“Software that translates a high-level request into the low-level commands required to perform the operation.

The most common abstraction layer is the programming interface (API) between an a system.

High-level calls are made to executes the necessary instru

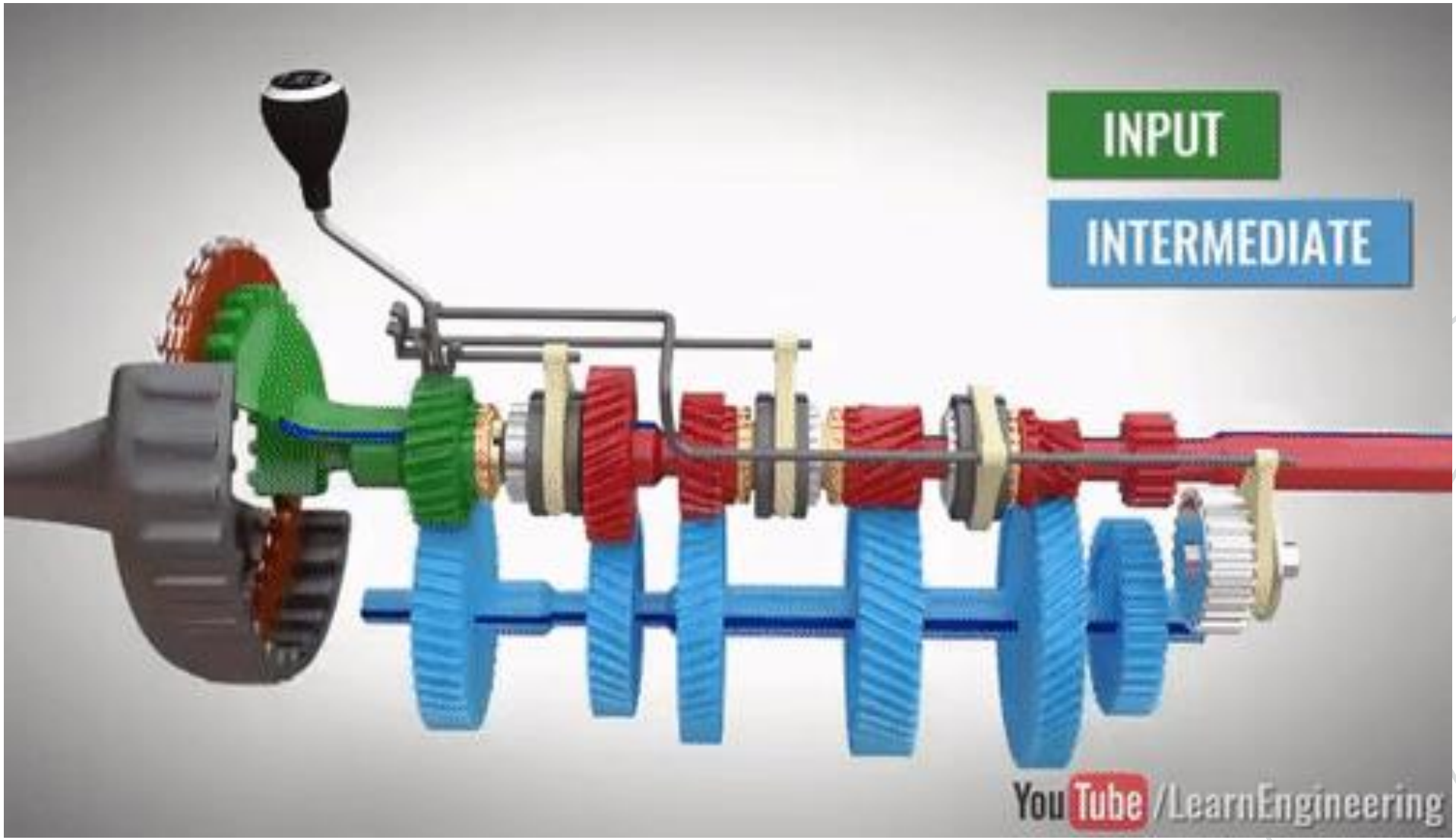
Source: https://www.pcmag.com/encyclopedia/term/abstraction-layer



Abstraction layer

https://www.mobiauto.com.br/revista/qual-cambio-e-o-melhor-para-voce-manual-automatico-ou-automatizado/149

Abstraction layer



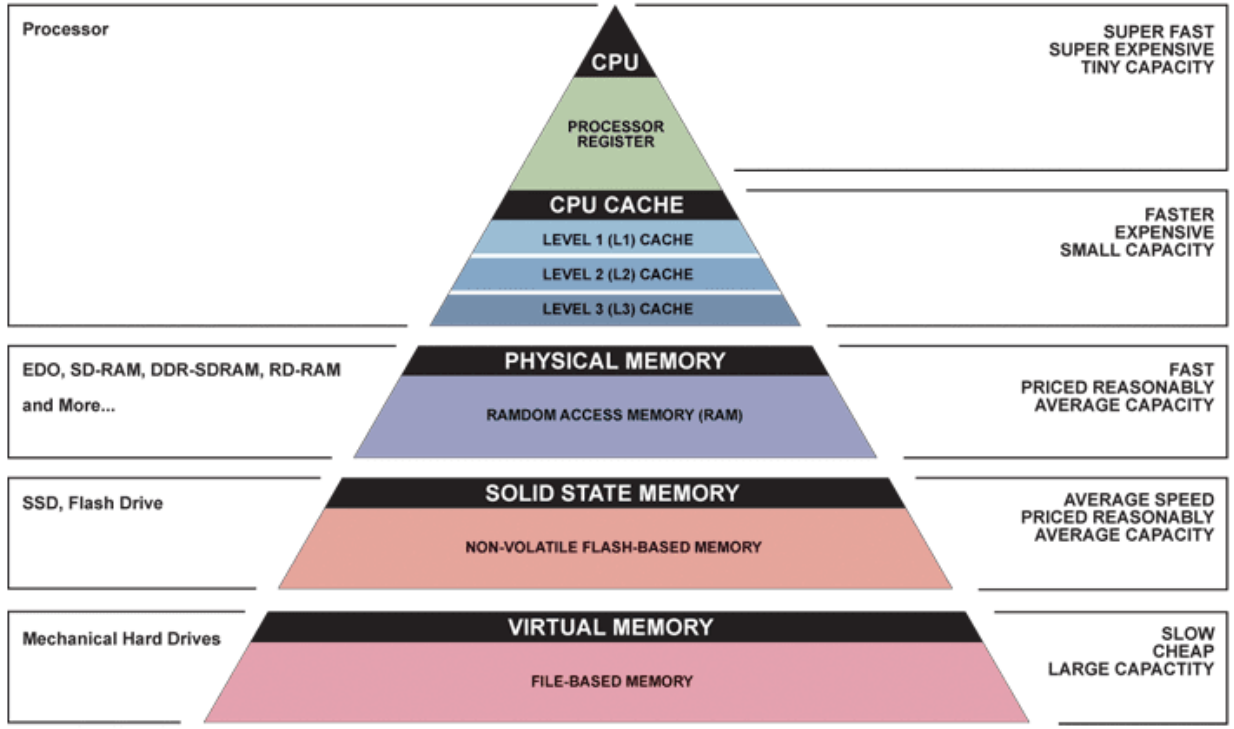
https://pt.quora.com/Qual-%C3%A9-a-diferen%C3%A7a-entre-um-carro-com-c%C3%A2mbio-autom%C3%A1tico-e-um-manual

Abstraction layer

OSI (Open Source Interconnection) 7 Layer Model

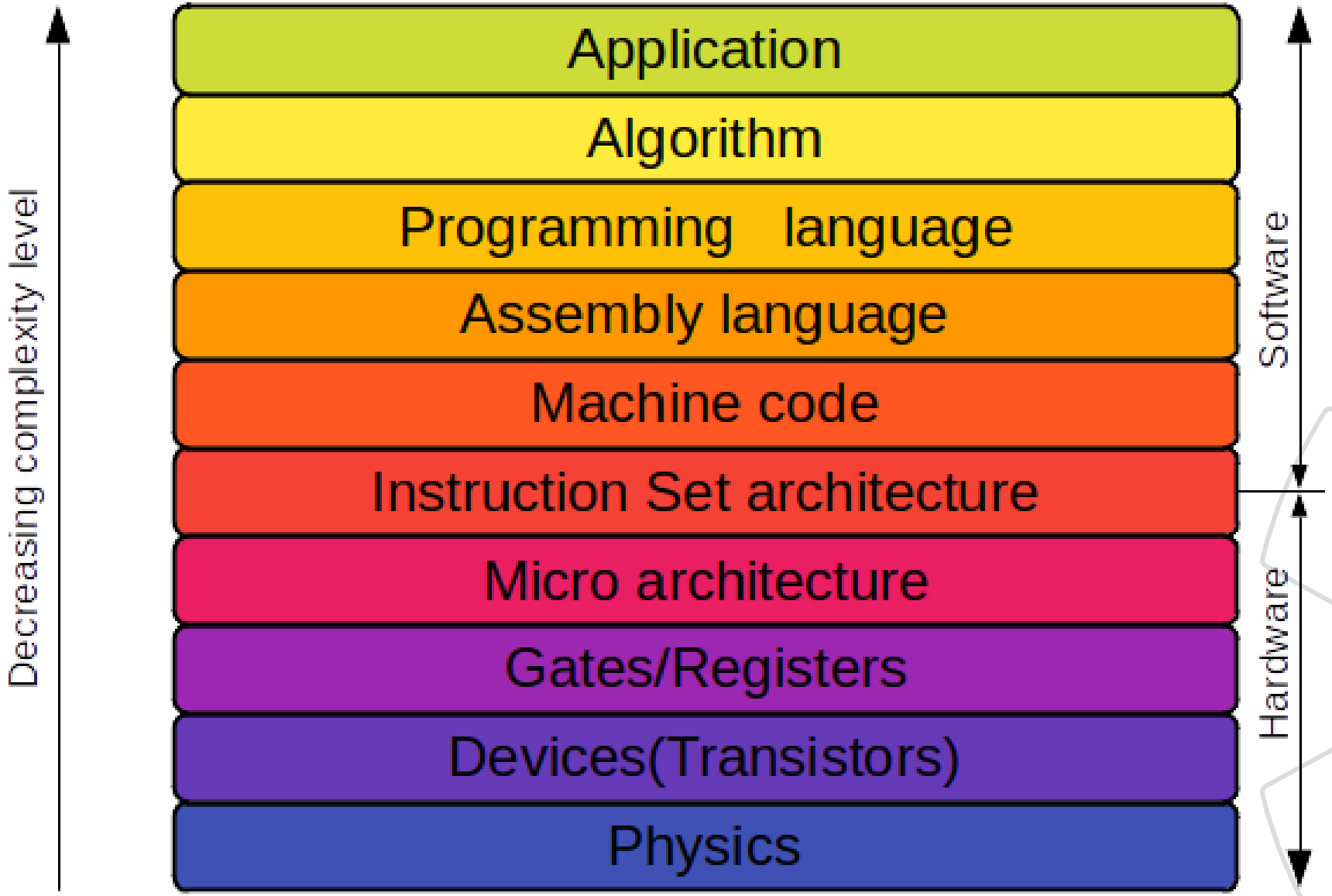
Layer	Application/Example		Central Device/ Protocols		DOD4 Model
Application (7) Serves as the window for users and application processes to access the network services.	**End User layer** Program that opens what was sent or creates what is to be sent Resource sharing • Remote file access • Remote printer access • Directory services • Network management		**User Applications** SMTP	GATEWAY	Process
Presentation (6) Formats the data to be presented to the Application layer. It can be viewed as the "Translator" for the network.	**Syntax layer** encrypt & decrypt (if needed) Character code translation • Data conversion • Data compression • Data encryption • **Character Set Translation**		JPEG/ASCII EBDIC/TIFF/GIF PICT		
Session (5) Allows session establishment between processes running on different stations.	**Synch & send to ports** (logical ports) Session establishment, maintenance and termination • Session support - perform security, name recognition, logging, etc.		**Logical Ports** RPC/SQL/NFS NetBIOS names		
Transport (4) Ensures that messages are delivered error-free, in sequence, and with no losses or duplications.	**TCP** Host to Host, Flow Control Message segmentation • Message acknowledgement • Message traffic control • Session multiplexing	PACKET FILTERING	TCP/SPX/UDP		Host to Host
Network (3) Controls the operations of the subnet, deciding which physical path the data takes.	**Packets** ("letter", contains IP address) Routing • Subnet traffic control • Frame fragmentation • Logical-physical address mapping • Subnet usage accounting		**Routers** IP/IPX/ICMP	Can be used on all layers	Internet
Data Link (2) Provides error-free transfer of data frames from one node to another over the Physical layer.	**Frames** ("envelopes", contains MAC address) [NIC card — Switch — NIC card] (end to end) Establishes & terminates the logical link between nodes • Frame traffic control • Frame sequencing • Frame acknowledgment • Frame delimiting • Frame error checking • Media access control		**Switch Bridge WAP** PPP/SLIP	Land Based Layers	Network
Physical (1) Concerned with the transmission and reception of the unstructured raw bit stream over the physical medium.	**Physical structure** Cables, hubs, etc. Data Encoding • Physical medium attachment • Transmission technique - Baseband or Broadband • Physical medium transmission Bits & Volts		**Hub**		

https://ramamotwani.wordpress.com/2015/09/17/computer-basics-osi-model/



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Abstraction layer



https://scimos.com/2020/07/07/layers-of-abstraction/

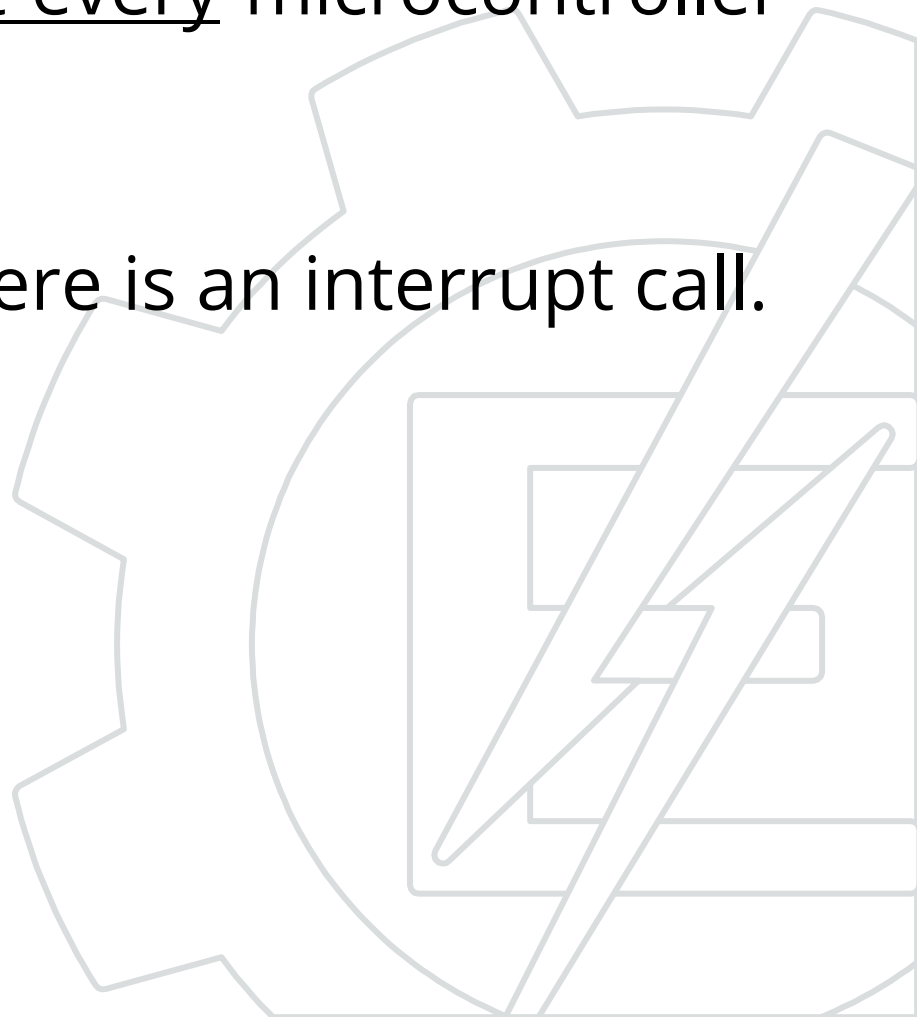
IAL

Interrupt Abstract Layer

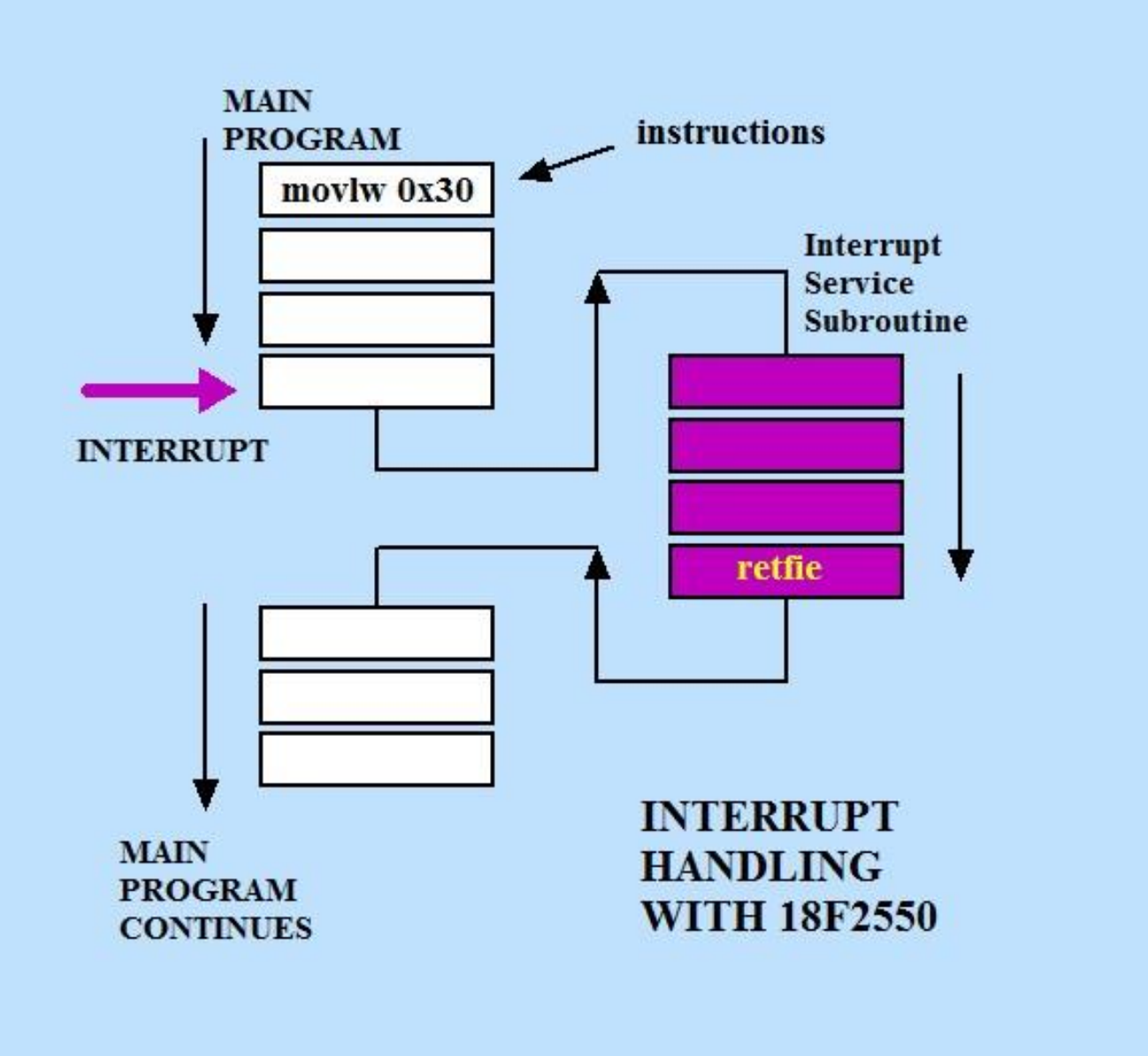
Working with interrupts

Interrupt Abstract Layer - IAL

- One type of hardware common to almost every microcontroller is the **interrupt device**.
- This device pauses the processor when there is an interrupt call.



Interrupt Abstract Layer



https://www.puntoflotante.net/BOLT-SYSTEM-INTERRUPT-TIMER-0.htm

Interrupt Abstract Layer - IAL

- One type of hardware common to almost every microcontroller is the interrupt device.
- This device pauses the processor when there is an interrupt call.
- It then checks the source of the call and, after saving the CPU variables on the stack, starts executing from a predefined address.

Interrupt Abstract Layer

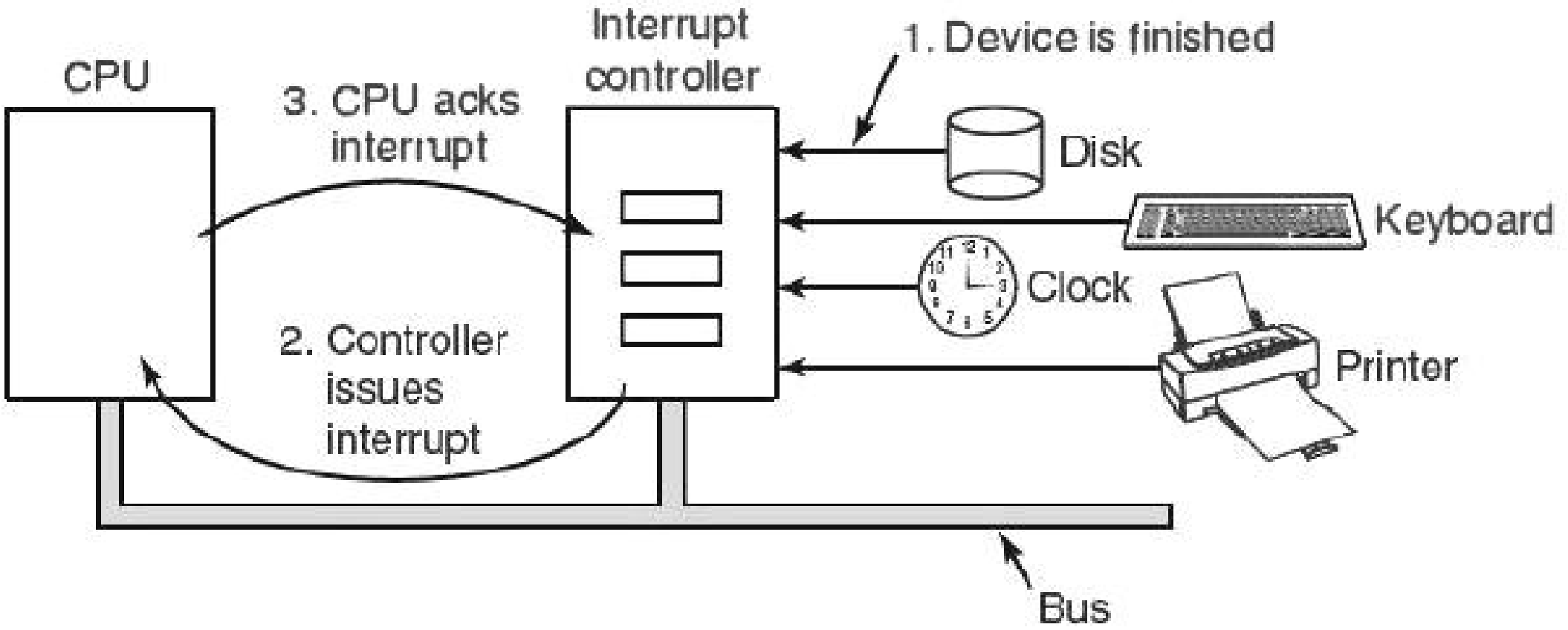
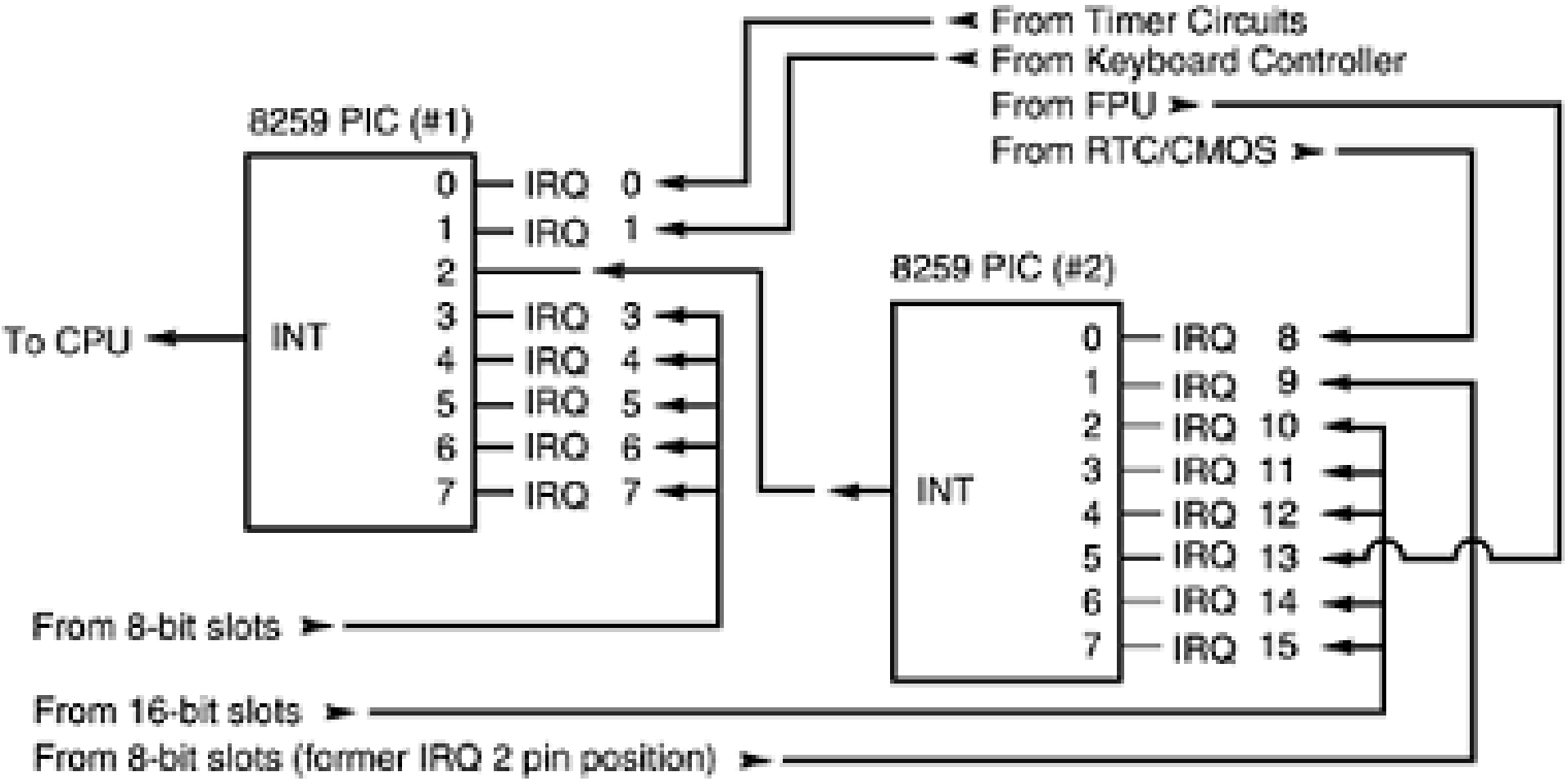


Figure 1. How an interrupt happens. The connections between the devices and the interrupt controller actually use interrupt lines on the bus rather than dedicated wires.

https://sungju.github.io/kernel/internals/interrupt_handling.html

Interrupt Abstract Layer



- The Intel 8259 is a Programmable Interrupt Controller (PIC) designed for the Intel 8085 and Intel 8086 microprocessors
- It's named *Programmable* as it can change the IRQ (Interrupt ReQuest) number of the signaled pin.
- On x86 architecture, two 8259 chips (Master PIC and Slave PIC) are used to make 15 IRQs

https://sungju.github.io/kernel/internals/interrupt_handling.html

Interrupt Abstract Layer - IAL

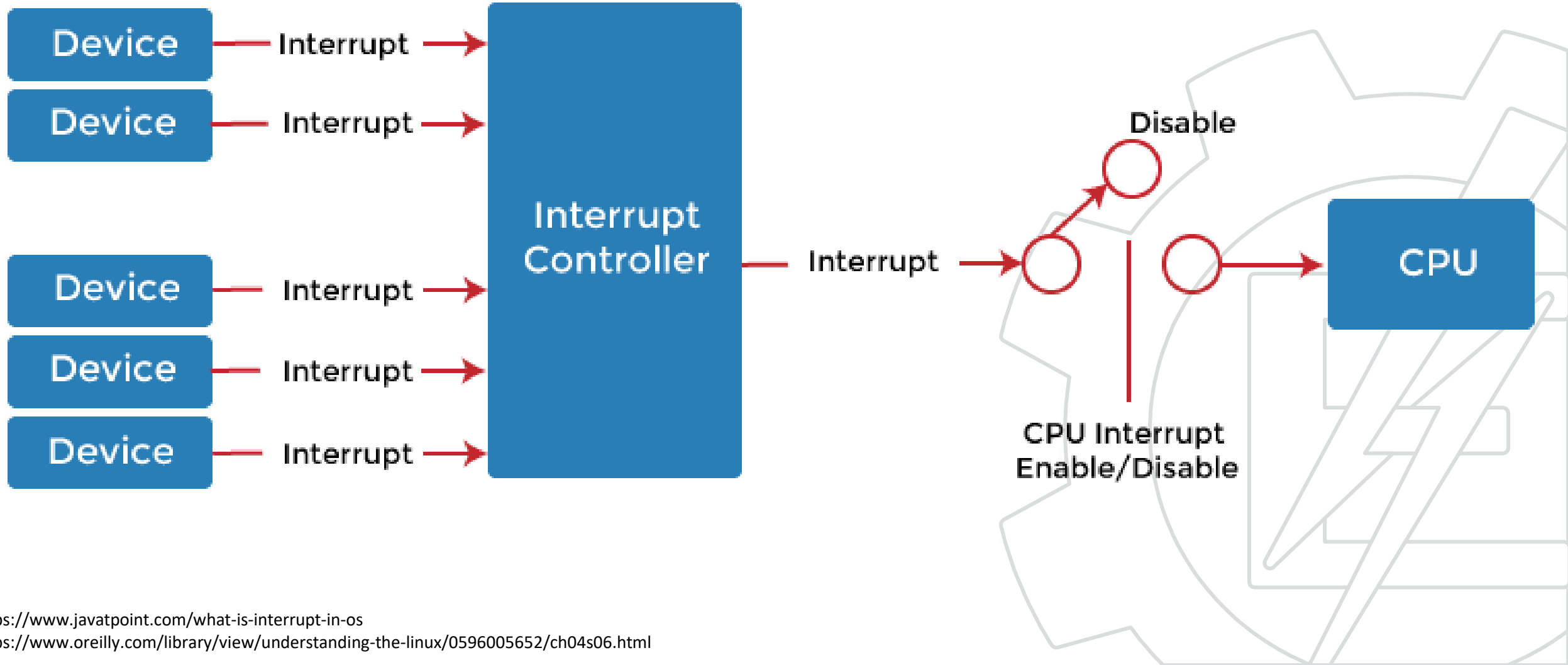
- Interrupts are closely related to **hardware**
- Each **architecture** AND **compiler** pose a different programming approach

```
//SDCC compiler way
void isr(void) interrupt 1{
    thisInterrupt();
}
```

```
//C18 compiler way
void isr (void){
    thisInterrupt();
}

#pragma code highvector=0x08
void highvector(void){
    _asm goto isr _endasm
}
#pragma code
```

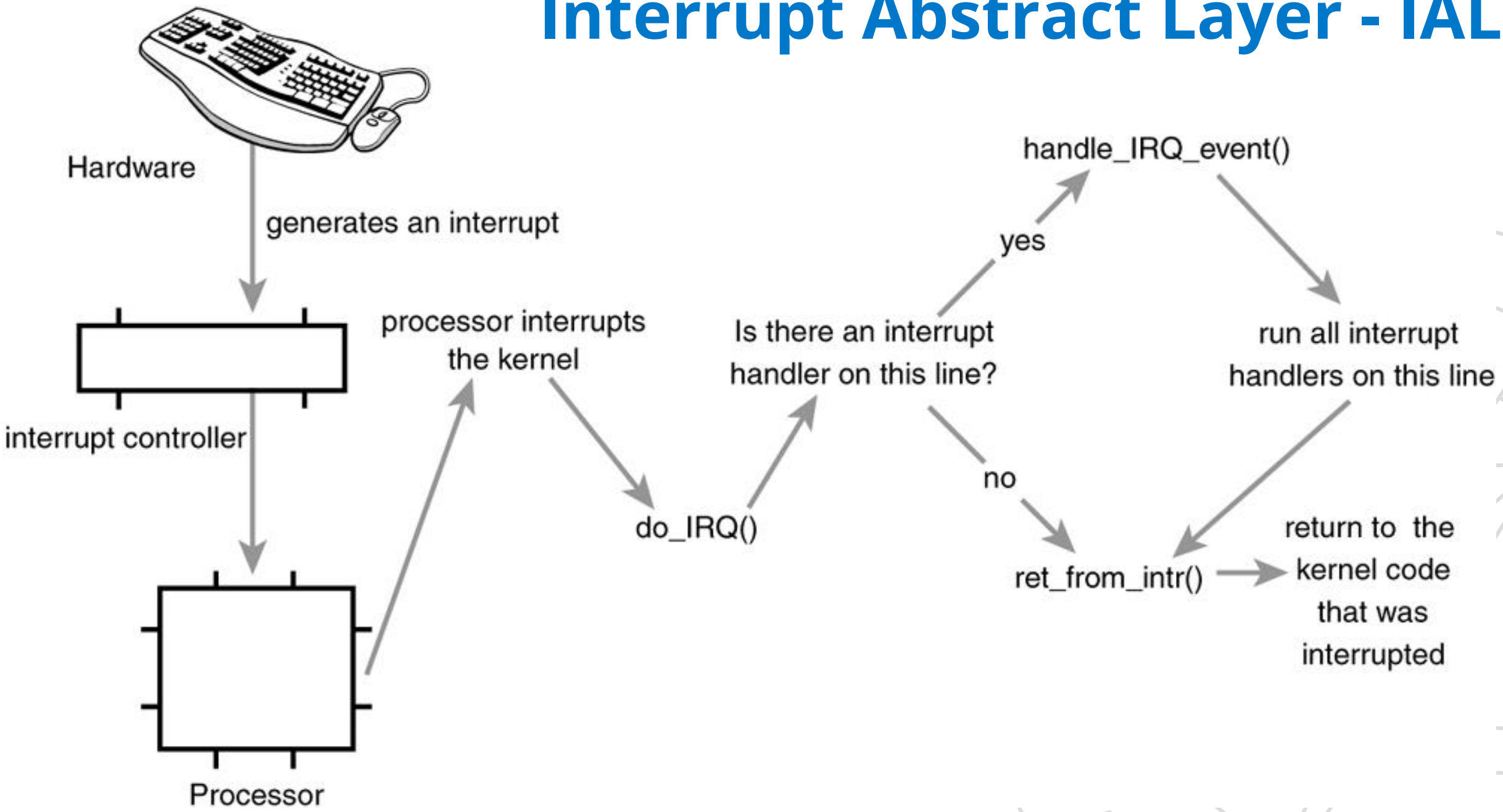
- How to hide this from programmer?

Interrupt Abstract Layer - IAL



https://www.javatpoint.com/what-is-interrupt-in-os
https://www.oreilly.com/library/view/understanding-the-linux/0596005652/ch04s06.html

Interrupt Abstract Layer - IAL



http://books.gigatux.nl/mirror/kerneldevelopment/0672327201/ch06lev1sec6.html

Interrupt Abstract Layer - IAL

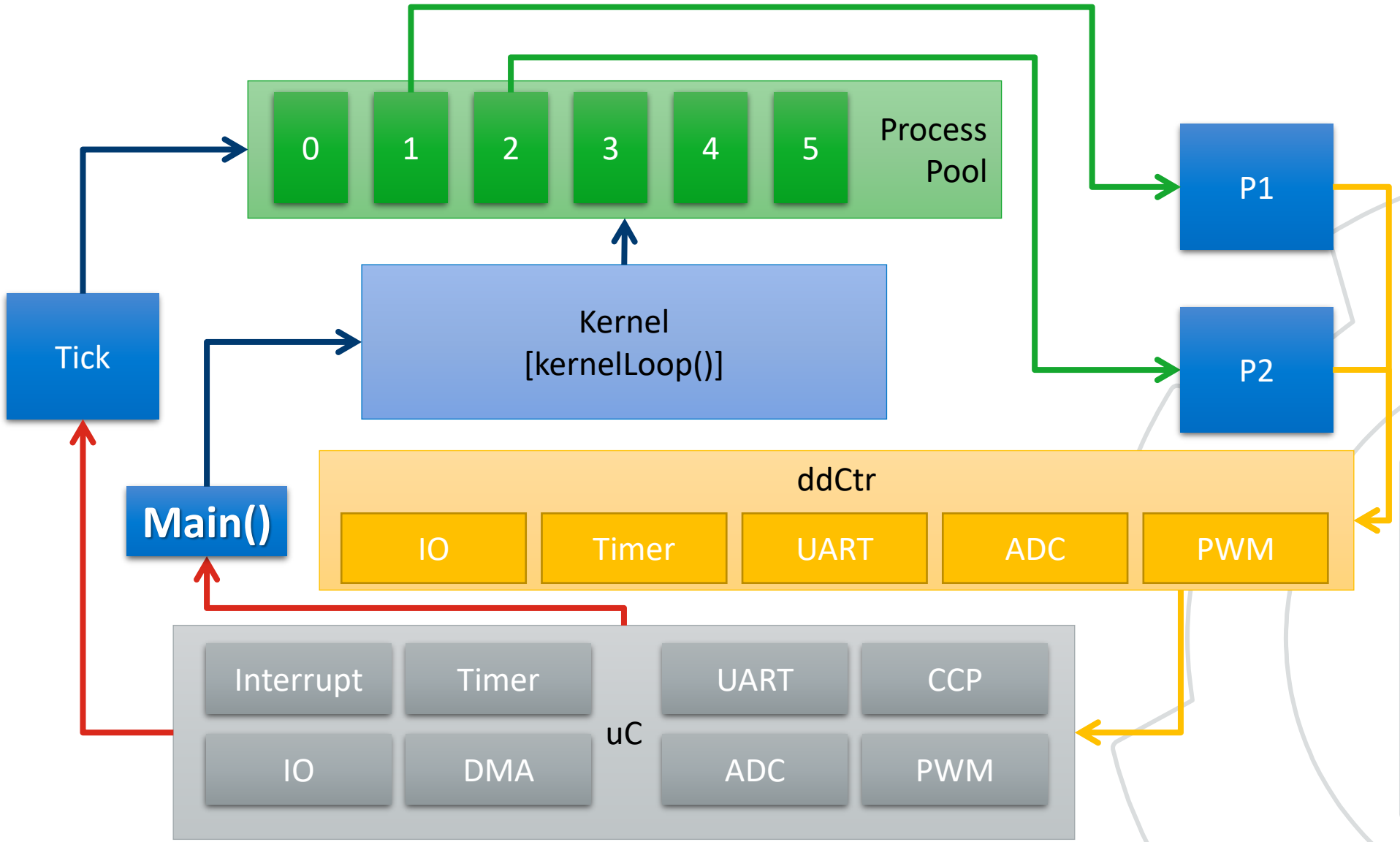
- In order to simplify this device from the point of view of the software it is common to **create a driver to manage the device**.
- This driver will receive the **address of functions** that will be executed when a certain interruption happens.
- You need a function to **receive the address** and a variable to store it internally

```
//Inside drvInterrupt.c

//defining the pointer to use in ISR callback
typedef void (*intFunc)(void);

//store the pointer to ISR here
static intFunc thisInterrupt;

//Set interrupt function to be called
char setInterruptFunc(void *parameters) {
    thisInterrupt = (intFunc) parameters;
    return SUCESS;
}
```

Interrupt Abstract Layer - IAL

- The IAL facilitates the use of interruptions by the programmer:
  1) The desired **driver** is initialized
  2) The **interrupt** driver is initialized
  3) The desired **function** is set

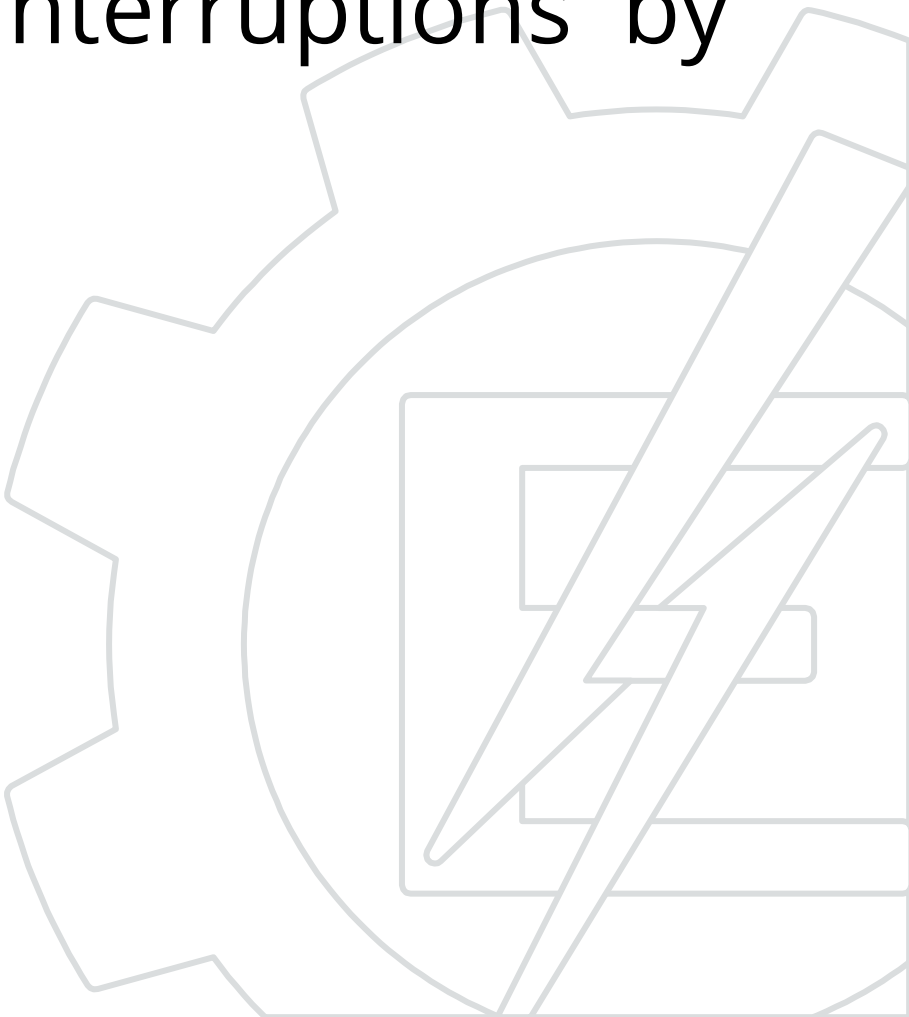
Architecture



```
//Interrupt function set without knowing hard/compiler issues
void timerISR(void) {
    callDriver(DRV_TIMER, TMR_RESET, 1000);
    kernelClock();
}

void main (void){
    kernelInit();

    initDriver(DRV_TIMER);
    initDriver(DRV_INTERRUPT);

    callDriver(DRV_TIMER, TMR_START, 0);
    callDriver(DRV_TIMER, TMR_INT_EN, 0);
    callDriver(DRV_INTERRUPT, INT_TIMER_SET, (void*)timerISR);
    callDriver(DRV_INTERRUPT, INT_ENABLE, 0);

    kernelLoop();
}
```

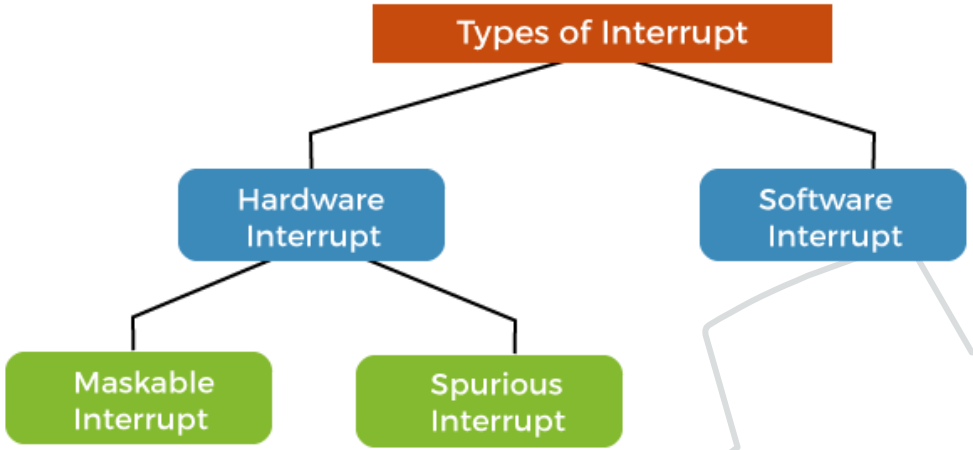
Interrupt Abstract Layer - IAL

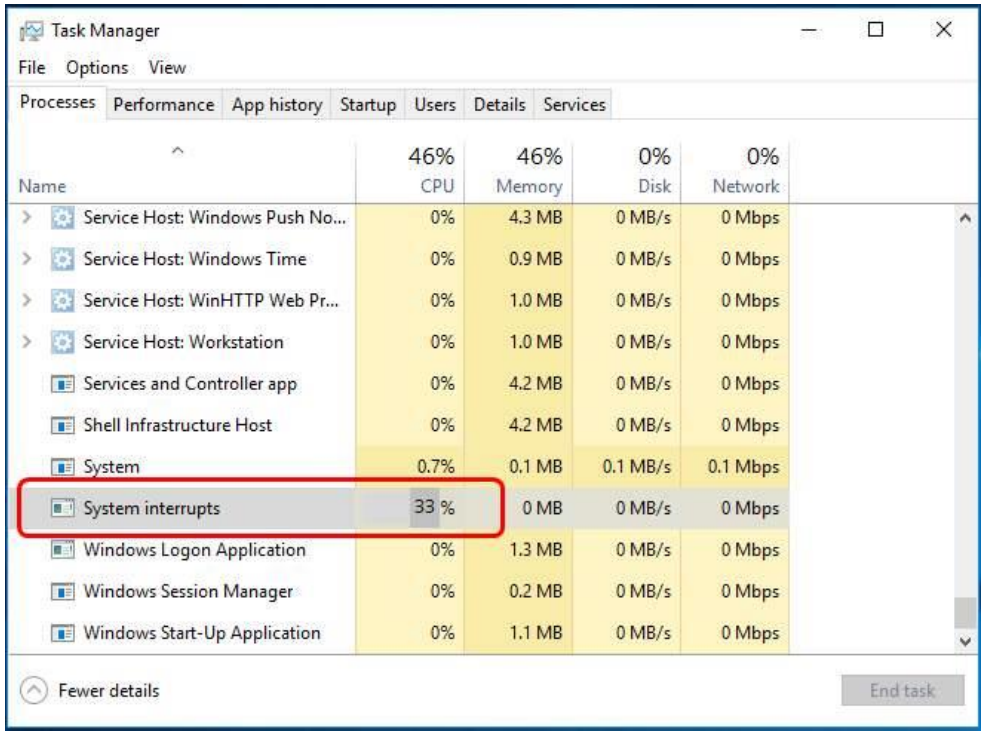
- The IAL facilitates the use of interruptions by the programmer:
  1) The desired driver is initialized
  2) The interrupt driver is initialized
  3) The desired function is set

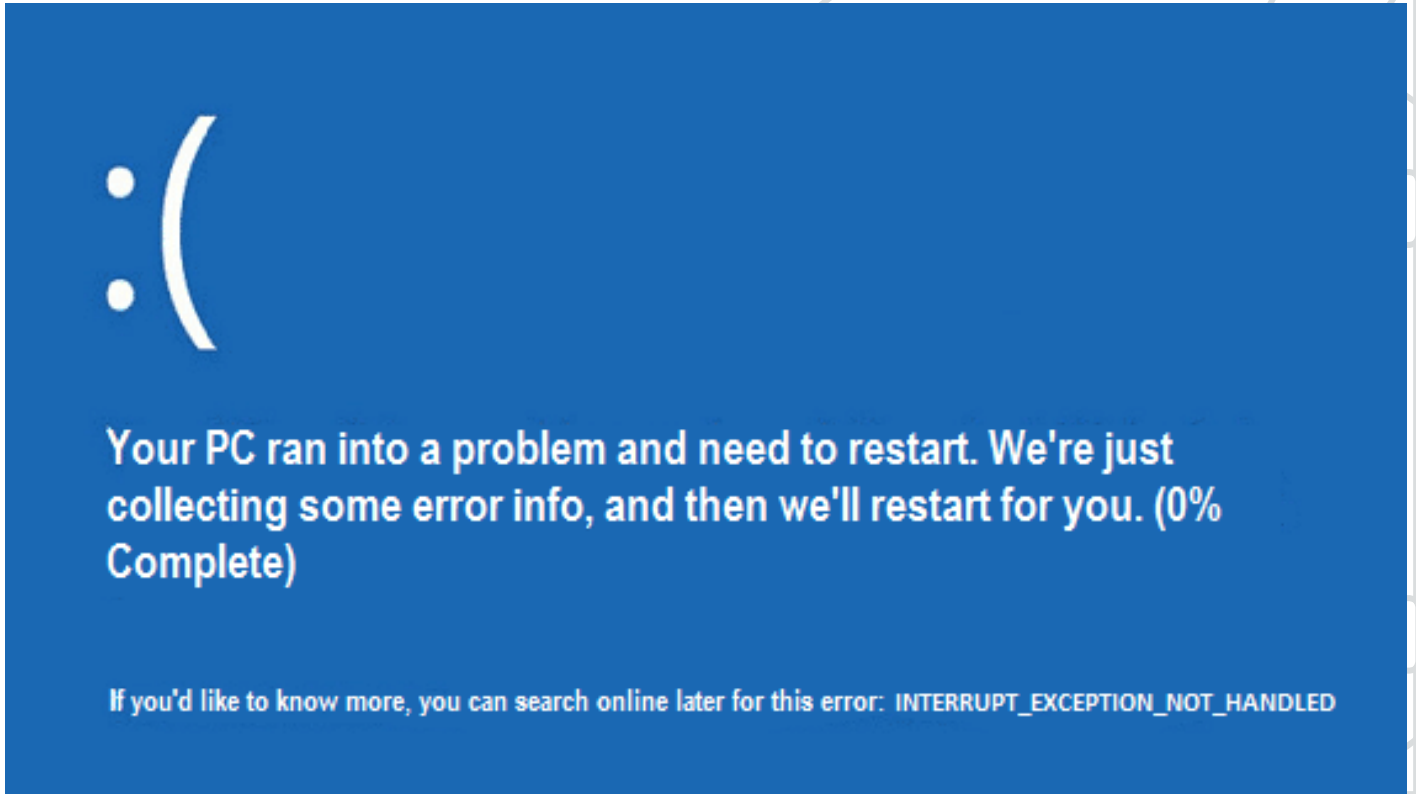


```
//Interrupt function set without knowing hard/compiler issues
void timerISR(void) {
    callDriver(DRV_TIMER, TMR_RESET, 1000);
    kernelClock();
}

void main (void){
    kernelInit();

    initDriver(DRV_TIMER);
    initDriver(DRV_INTERRUPT);

    callDriver(DRV_TIMER, TMR_START, 0);
    callDriver(DRV_TIMER, TMR_INT_EN, 0);
    callDriver(DRV_INTERRUPT, INT_TIMER_SET, (void*)timerISR);
    callDriver(DRV_INTERRUPT, INT_ENABLE, 0);

    kernelLoop();
}
```

Interrupt Abstract Layer - IAL





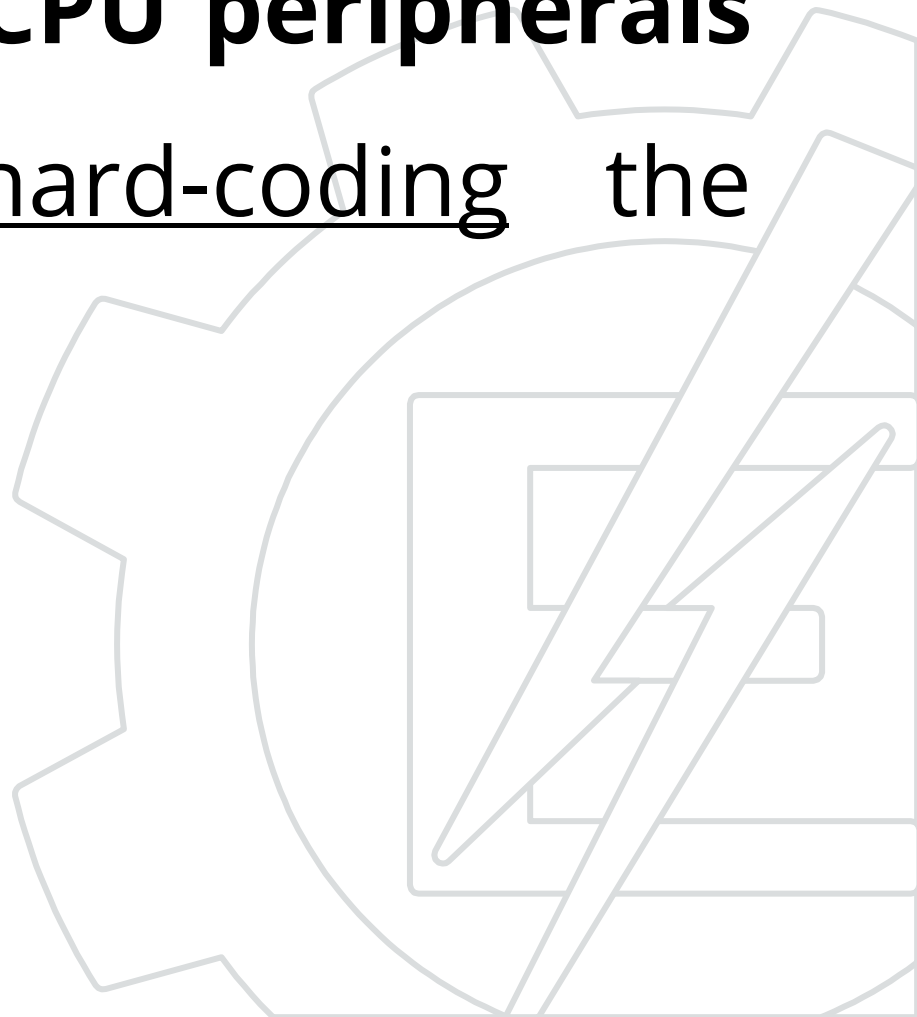


https://www.javatpoint.com/what-is-interrupt-in-os

Driver Callback

Driver Callback

- How to make **efficient use of CPU peripherals** without using pooling or hard-coding the interrupts?

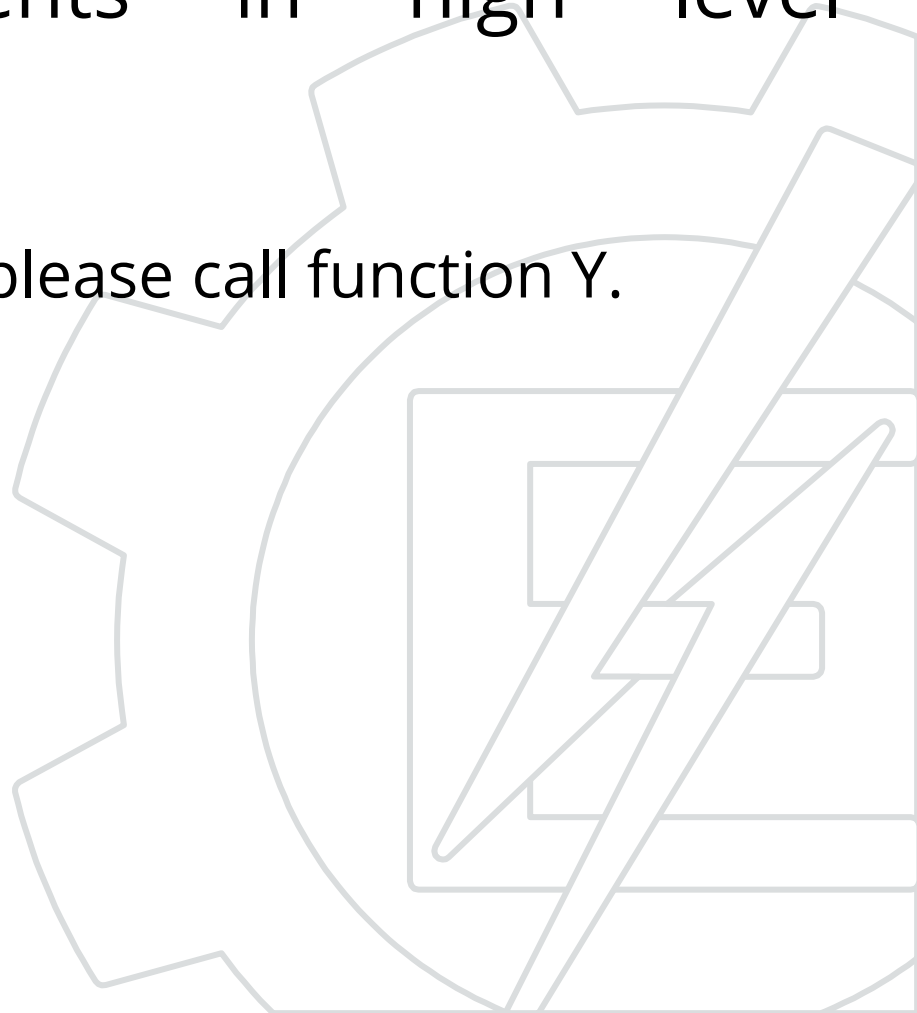


Callback functions

- Callback functions resemble events in high level programming

  OnKeyPress()
  OnMouseOn()

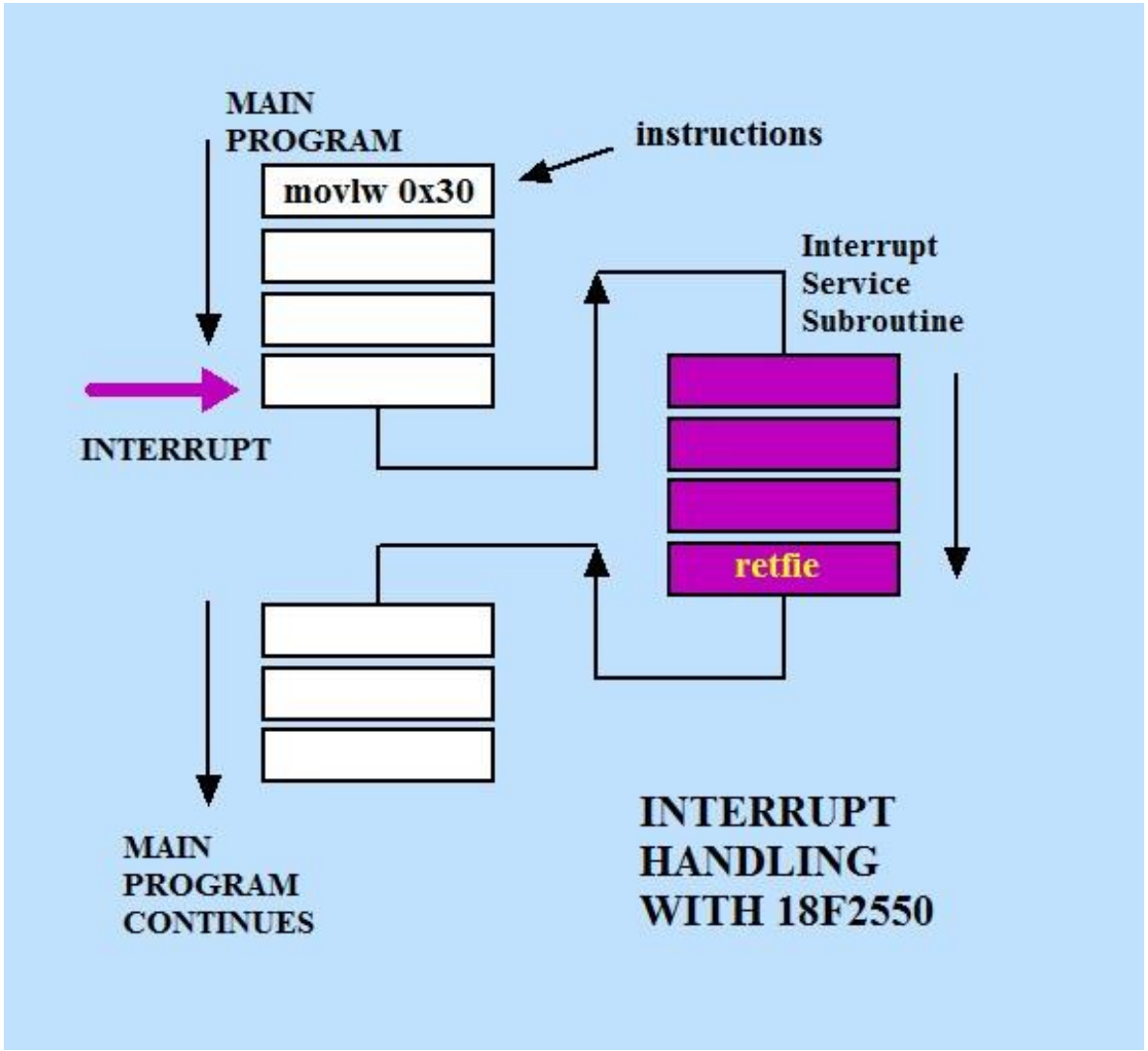
  - e.g.: When the mouse clicks in the button X, please call function Y.

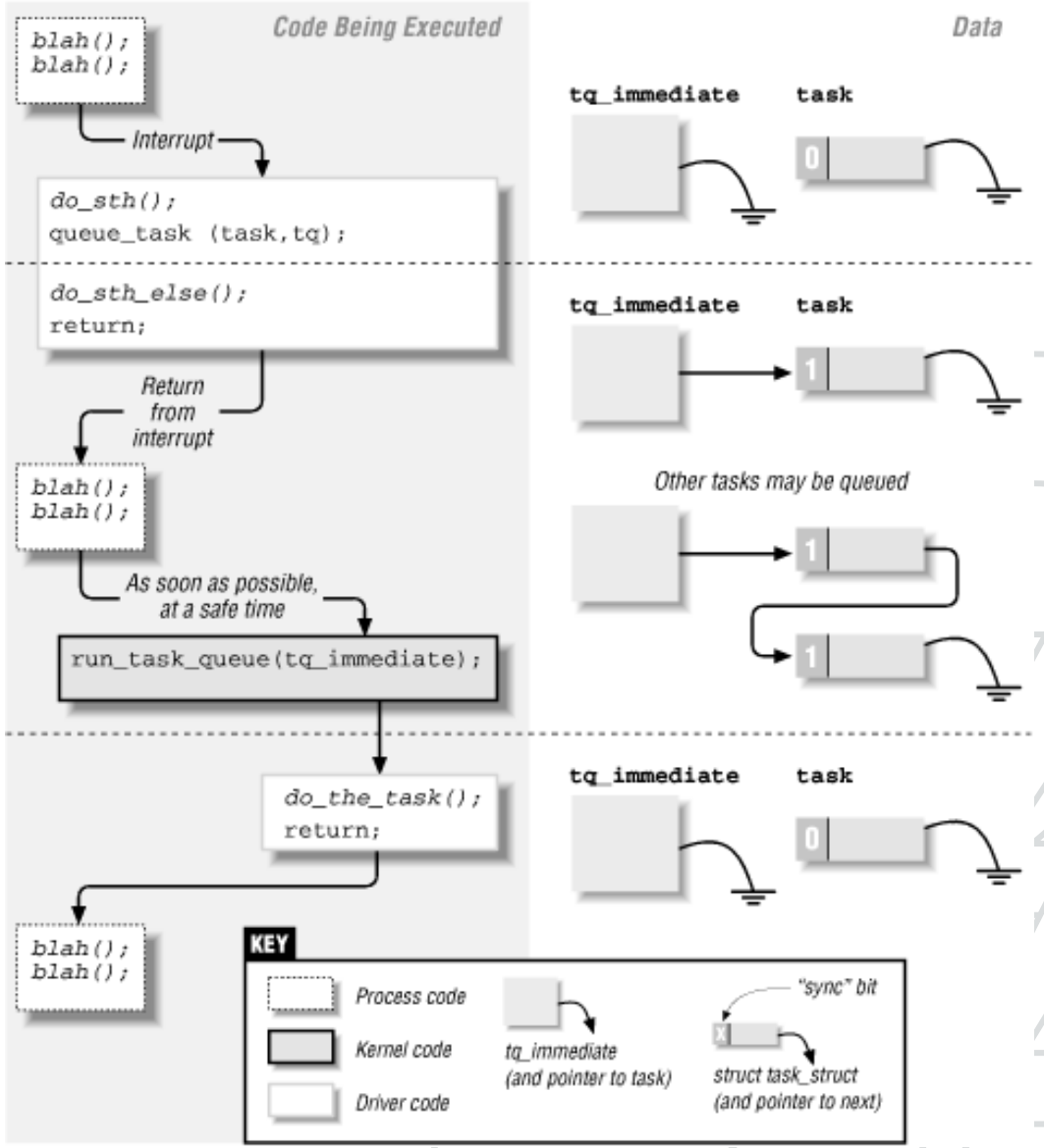


Callback functions

- Callback functions resemble events in high level programming
  - e.g.: When the mouse clicks in the button X, please call function Y.
- The desired hardware must be able to rise an interrupt
- Part of the work is done under interrupt context, preferable the **faster** part

Interrupt Abstract Layer

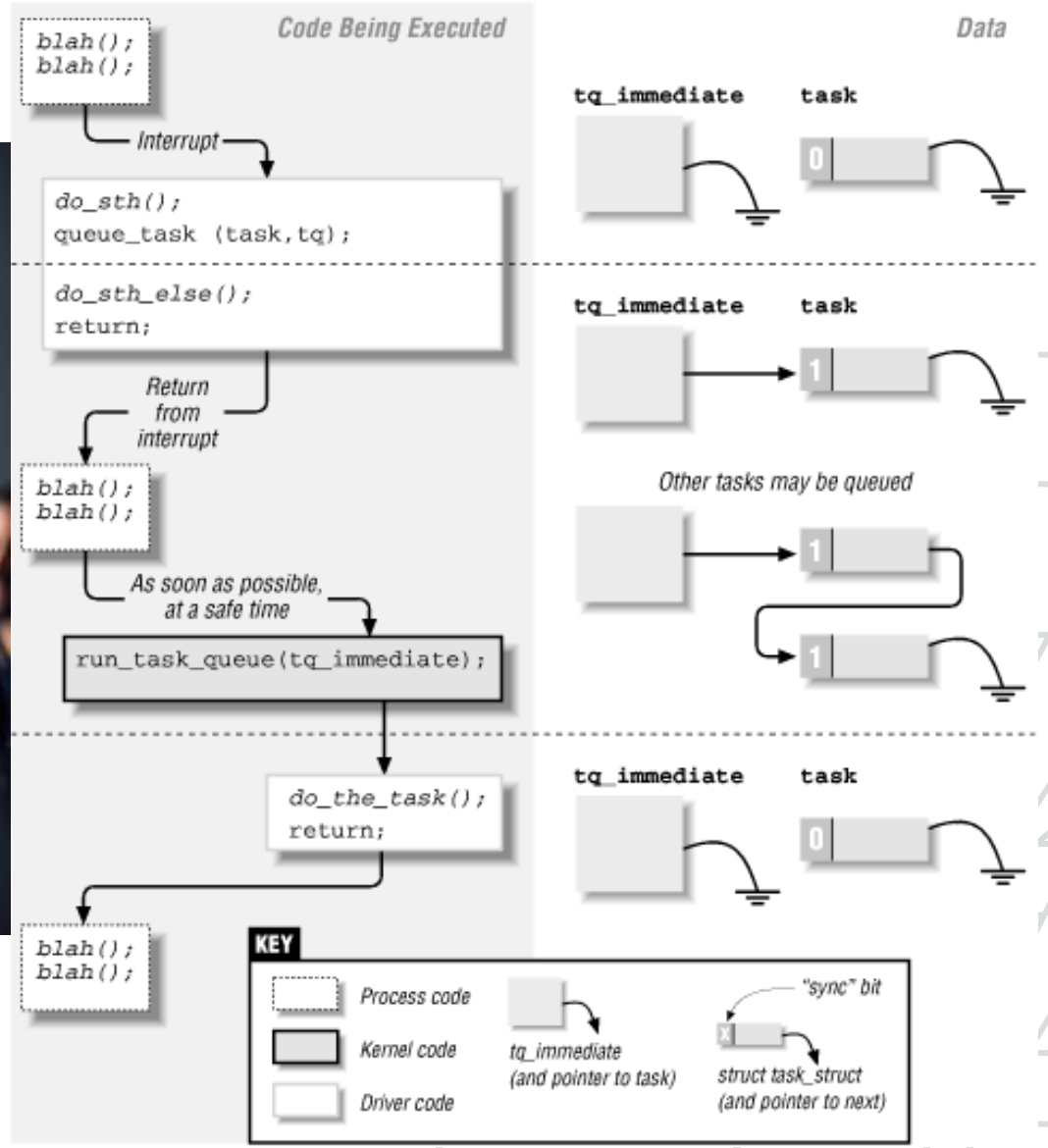




https://www.puntoflotante.net/BOLT-SYSTEM-INTERRUPT-TIMER-0.htm

https://lwn.net/Kernel/LDD2/ch06.lwn

Interrupt Abstract Layer



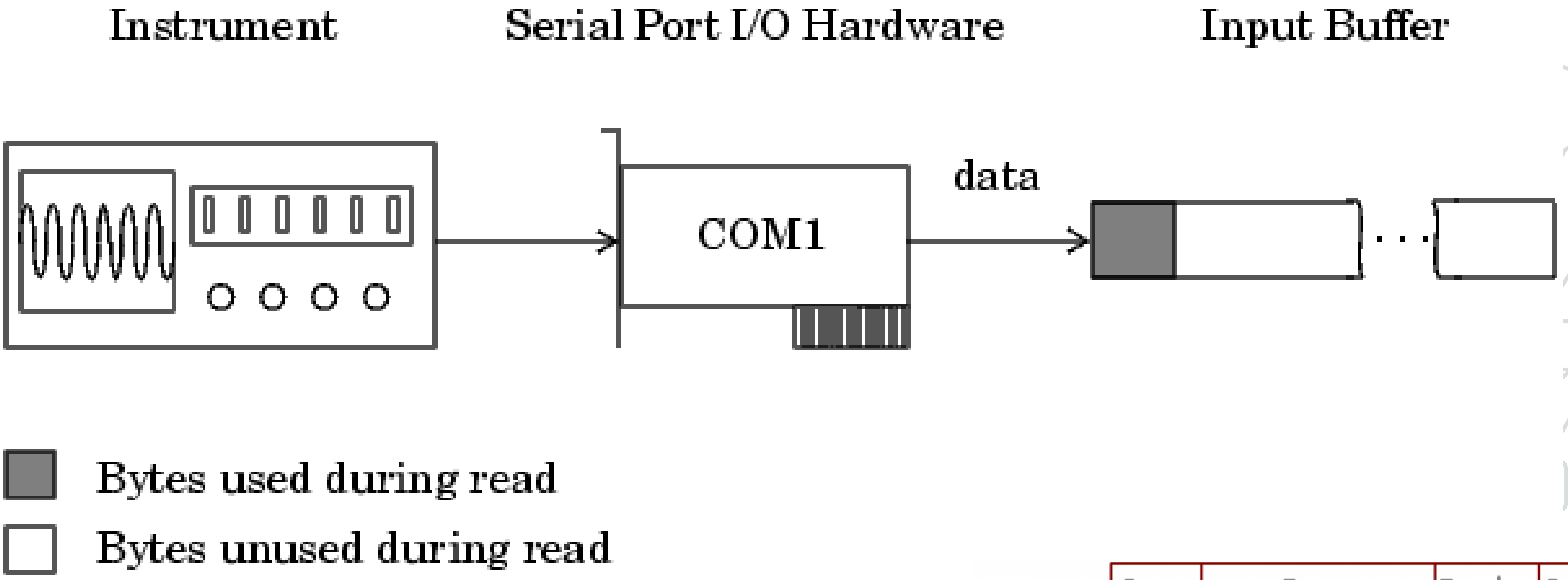
https://lwn.net/Kernel/LDD2/ch06.lwn

```
//Inside drvInterrupt.c

//defining the pointer to use in ISR callback
typedef void (*intFunc)(void);

//store the pointer to ISR here
static intFunc thisInterrupt;

//Set interrupt function to be called
char setInterruptFunc(void *parameters) {
    thisInterrupt = (intFunc) parameters;
    return SUCESS;
}
```

Callback functions

- In the **callback process** there are two separate parts that must be executed sequentially.
  - The first is the **code that runs inside the interrupt**. It should be fast and use few resources. Usually only the data or information generated by the interrupt is saved and its processing is delayed at this time
  - The second is the **callback process**, now run by the kernel can take longer without disturbing the system timing.

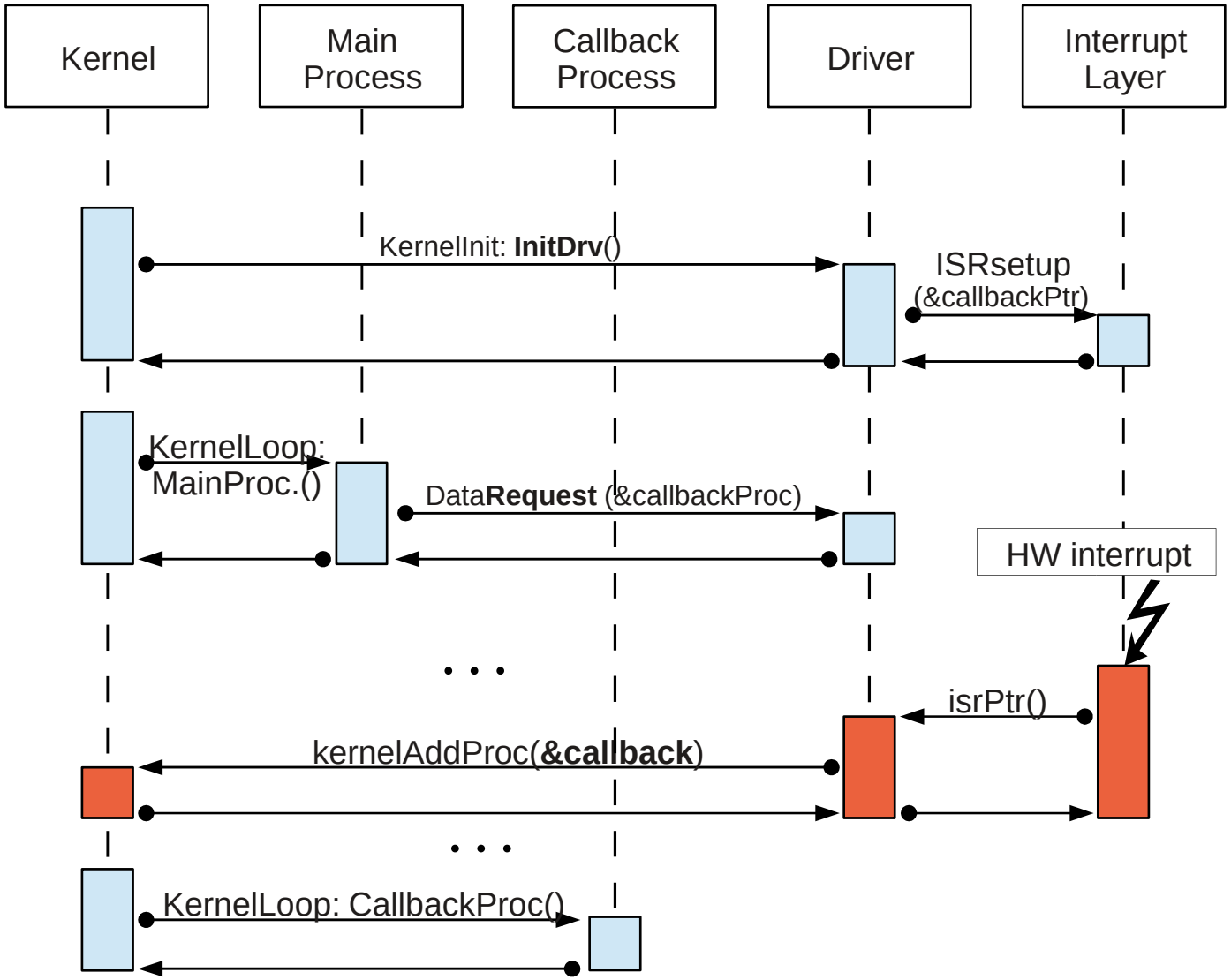
Callback functions



Buffer + Verfication (store message)
Execution (CRC, process data, etc)

http://www.ece.northwestern.edu/local-apps/matlabhelp/techdoc/matlab_external/ch_ser23.html
https://learn.sparkfun.com/tutorials/serial-communication/all

Callback functions

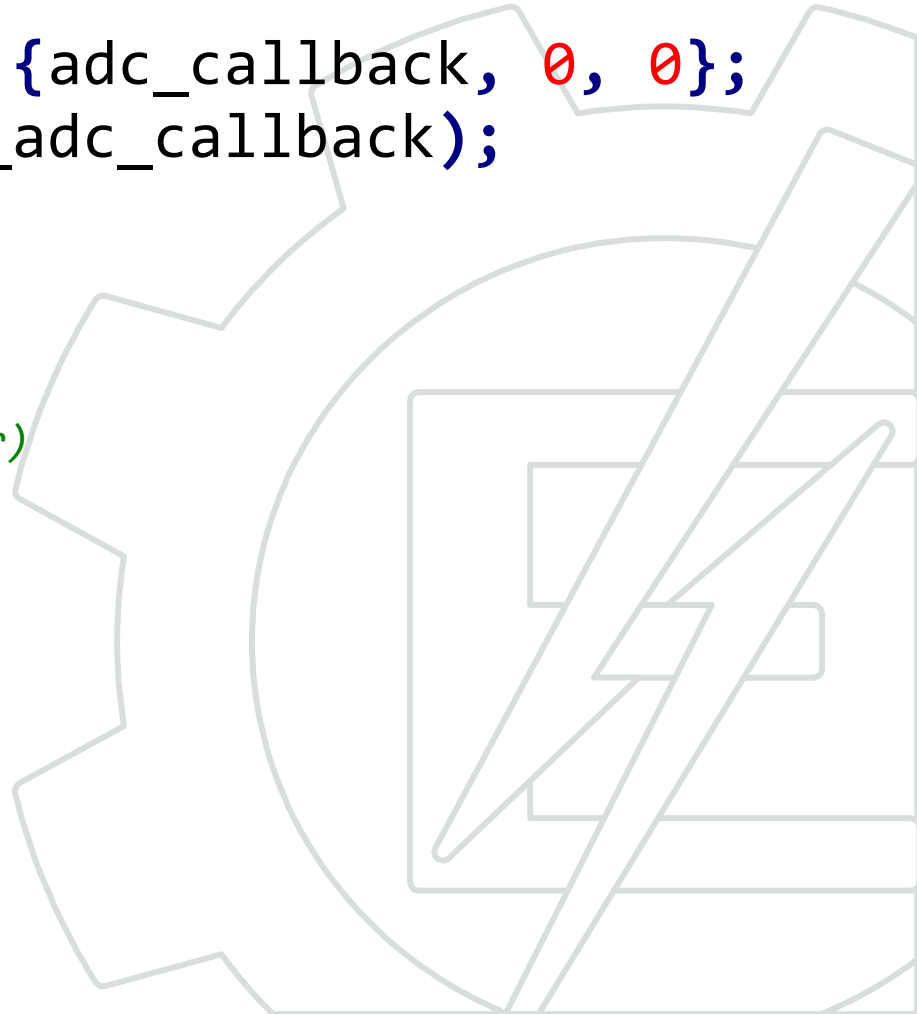


Calback functions

```
//********** Excerpt from drvAdc.c **********
//  called from setup time to enable ADC interrupt
//  and setup ADC ISR callback
char enableAdcInterrup(void* parameters){
      callDriver(DRV_INTERRUPT,INT_ADC_SET,(void*)adcISR);
      BitClr(PIR1,6);
      return FIM_OK;
}

//********** Excerpt from drvInterrupt.c **********
//  store the pointer to the interrupt function
typedef void (*intFunc)(void);
static intFunc adcInterrupt;

//  function to set ADC ISR callback for latter use
char setAdcInt(void *parameters) {
      adcInterrupt = (intFunc)parameters;
      return FIM_OK;
}
```

Callback functions

```
//********** Excerpt from main.c **********
//  Process called by the kernel
char adc_func(void) {
      //creating callback process
        static process proc_adc_callback = {adc_callback, 0, 0};
        callDriver(DRV_ADC,ADC_START,&proc_adc_callback);
        return REPEAT;
}

//********** Excerpt from drvAdc.c **********
//function called by the process adc_func (via drv controller)
char startConversion(void* parameters){
        callBack = parameters;
        ADCON0 |= 0b00000010;  //start conversion
        return SUCCESS;
}
```



Callback functions

```
//********** Excerpt from drvInterrupt.c **********
//interrupt function
void isr(void) interrupt 1 {
      if (BitTst(INTCON, 2)) { //Timer overflow
      }
      if (BitTst(PIR1, 6)) {   //ADC conversion finished
            //calling ISR callback stored
            adcInterrupt();
      }
}
//********** Excerpt from drvAdc.c **********
//ADC ISR callback function
void adcISR(void){
      value = ADRESH;
      value <<= 8;
      value += ADRESL;
      BitClr(PIR1,6);
      kernelAddProc(callBack);
}
```

ISR – Interrupt Service Routine

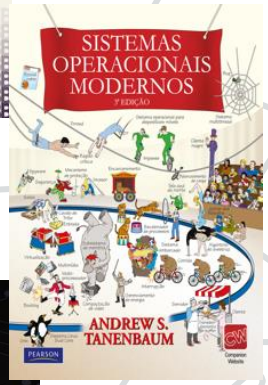
```
//********** Excerpt from main.c **********
//callback function started from the kernel
char adc_callback(void) {
    unsigned int resp;
    //getting the converted value
    callDriver(DRV_ADC,ADC_LAST_VALUE,&resp);
    //changing line and printing on LCD
    callDriver(DRV_LCD,LCD_LINE,1);
    callDriver(DRV_LCD,LCD_INTEGER,resp);
    return SUCCESS;
}
```

Bibliography

- Denardin, G. B.; Barriquello, C. H. **Sistemas operacionais de tempo real e sua aplicação em sistemas embarcados**. 1ª ed. Editora Blucher. ISBN: 9788521213970. https://plataforma.bvirtual.com.br/Acervo/Publicacao/169968

- Tanenbaum, A.S. **Sistemas Operacionais Modernos**. 3ª ed. 674 páginas. São Paulo: Pearson. ISBN: 9788576052371.
  https://plataforma.bvirtual.com.br/Acervo/Publicacao/1233

- Almeida, Moraes, Seraphim e Gomes. **Programação de Sistemas Embarcados**. 2ª ed. Editora GEN LTC. ISBN: 9788595159105.
  https://cengagebrasil.vitalsource.com/books/9788595159112



Available at: https://unifei.edu.br/ensino/bibliotecas/

Embedded Operating Systems

Prof. Otávio Gomes
otavio.gomes@unifei.edu.br
/otavio-gomes