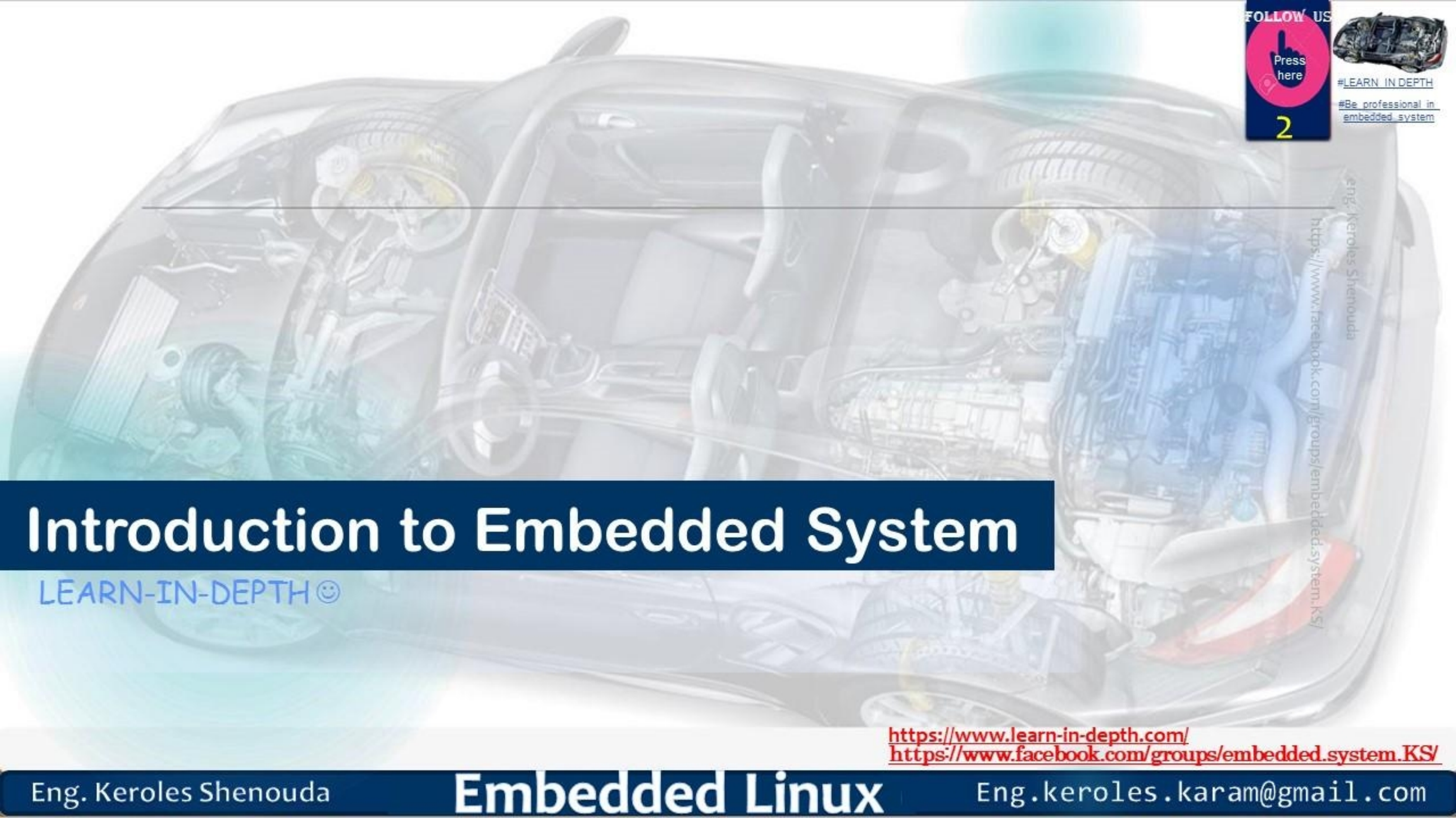
FOLLOW US
Press here
1
#LEARN IN DEPTH
#Be professional in embedded system

Embedded Linux (PART 1)

- INTRODUCTION TO EMBEDDED SYSTEM
- HISTORICAL BACKGROUND
- COURSE OUTLINE
- LOADING, LINKING AND ADDRESS BINDING
- MEMORY MANAGEMENT CONCEPTS
- FRAGMENTATION

ENG.KEROLES SHENOUDA

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Introduction to Embedded System

LEARN-IN-DEPTH ☺

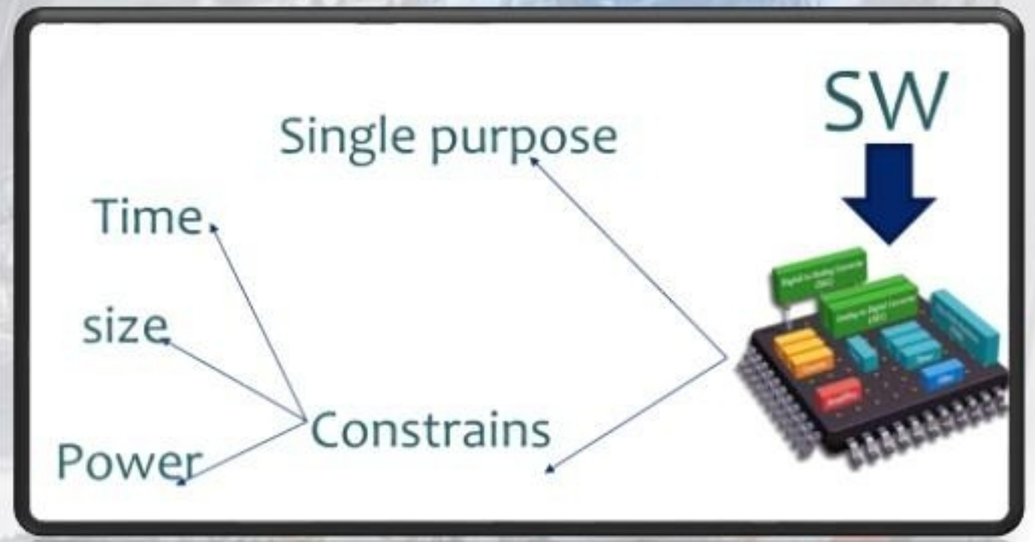
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
#LEARN IN DEPTH
#Be professional in embedded system

Embedded System What's that ?

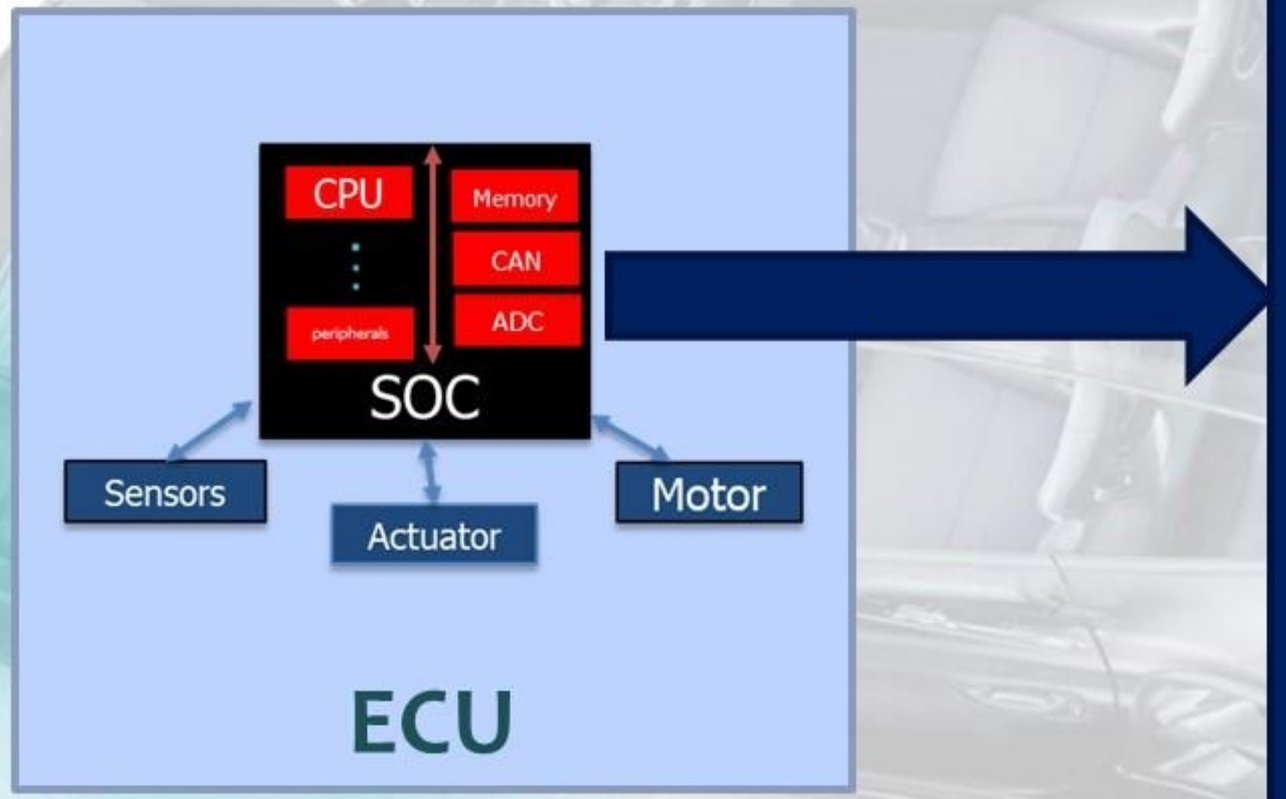
An embedded system **is a special-purpose computer system** designed to perform **one or a few dedicated functions**, often with real-time computing constraints.
It is usually embedded as part of a complete device including hardware and mechanical parts.
In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming.

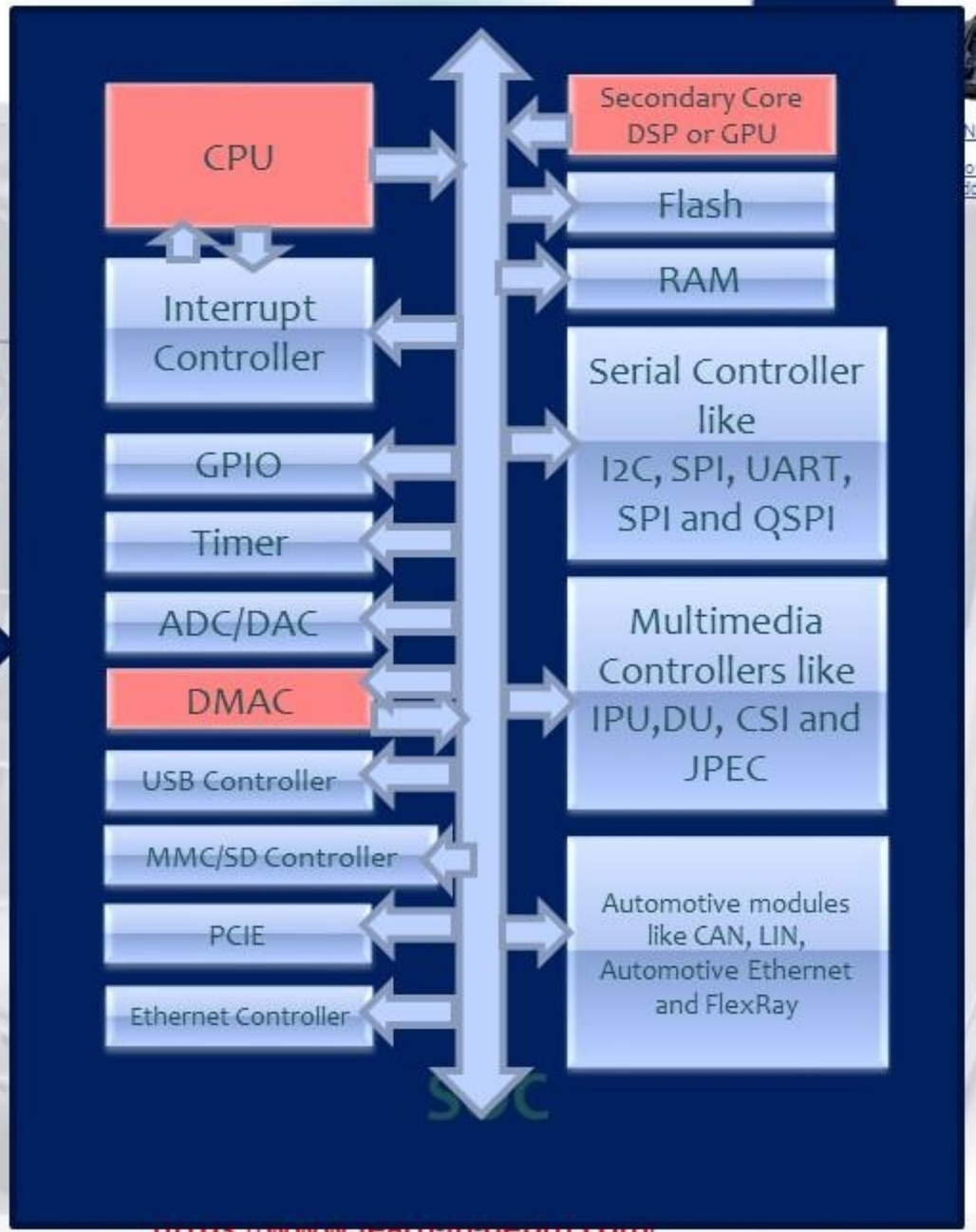
Embedded systems control many of the common devices in use today.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

System on Chip (SOC)



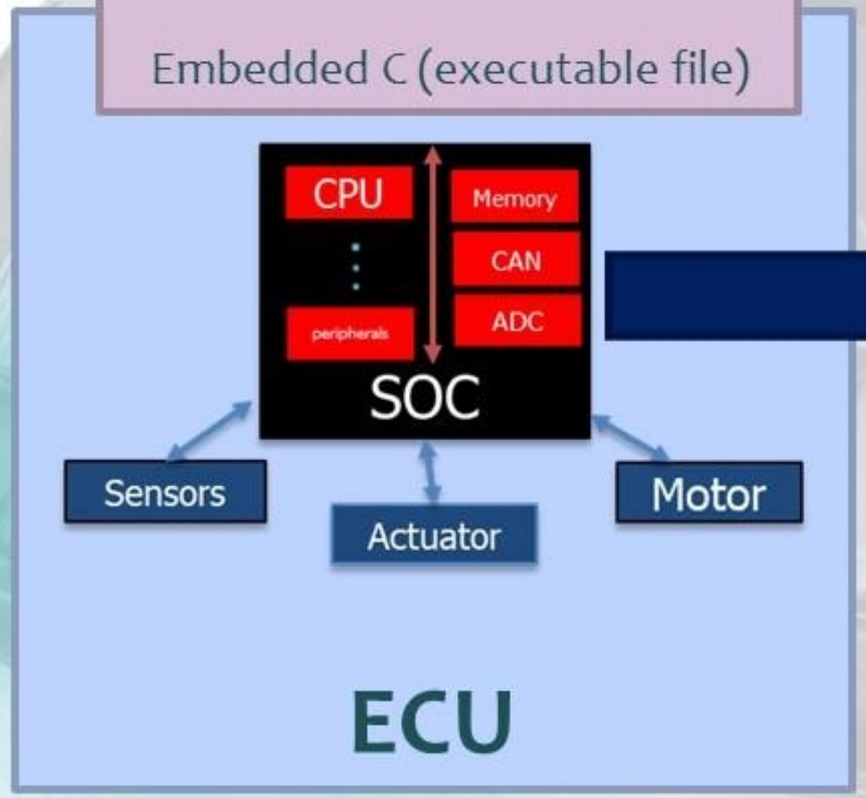


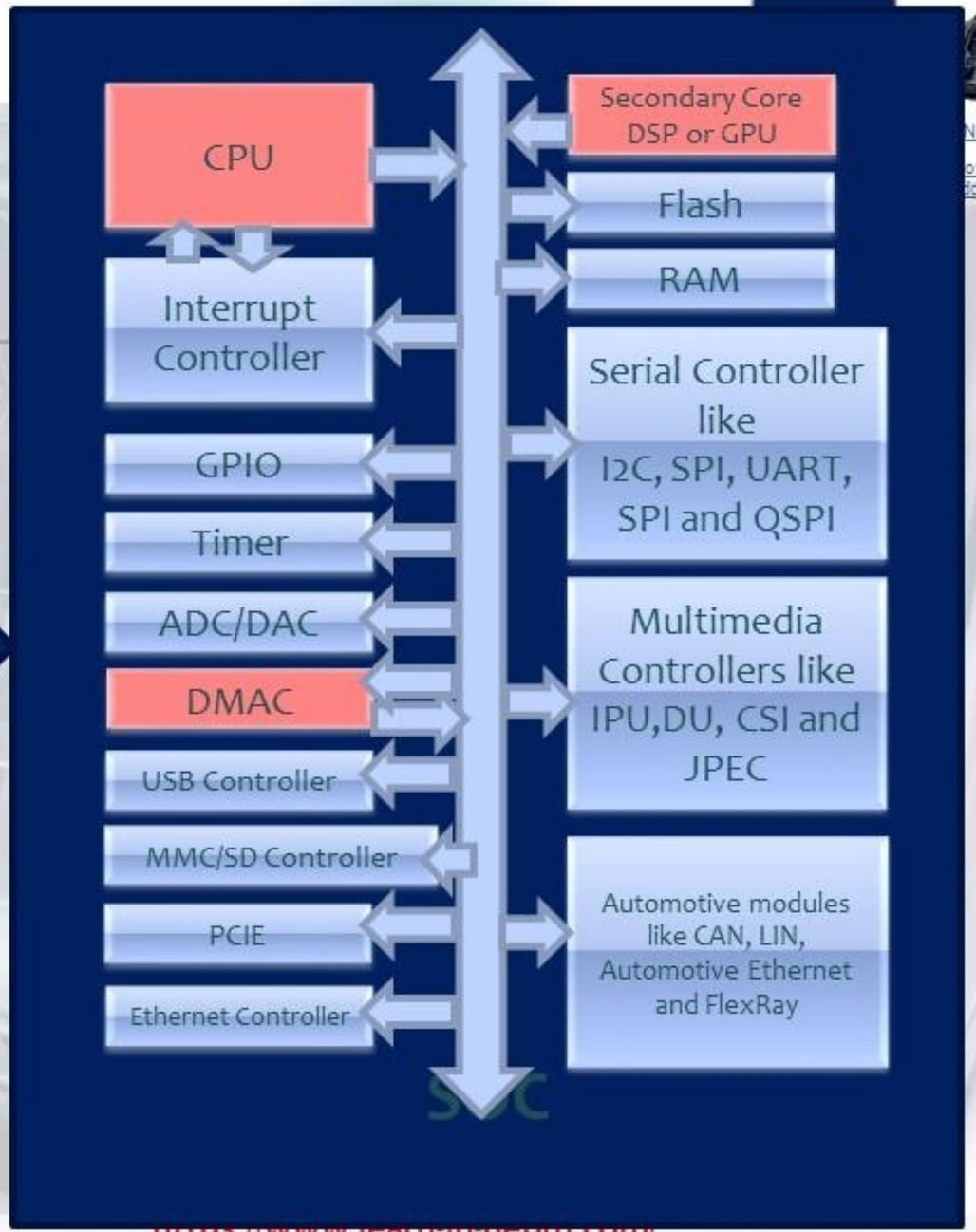
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

S ip (SOC)

```
(*(volatile unsigned char *)
0XFFFF0000) = 12 ;
```

Embedded C (executable file)





N IN DEPTH

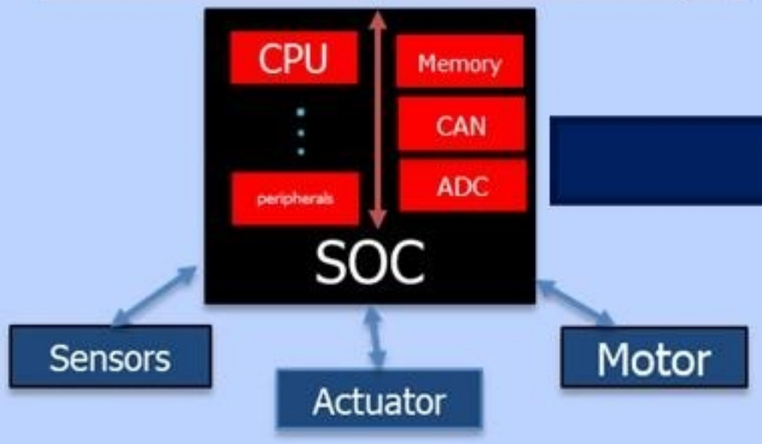
ofessional in
dded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

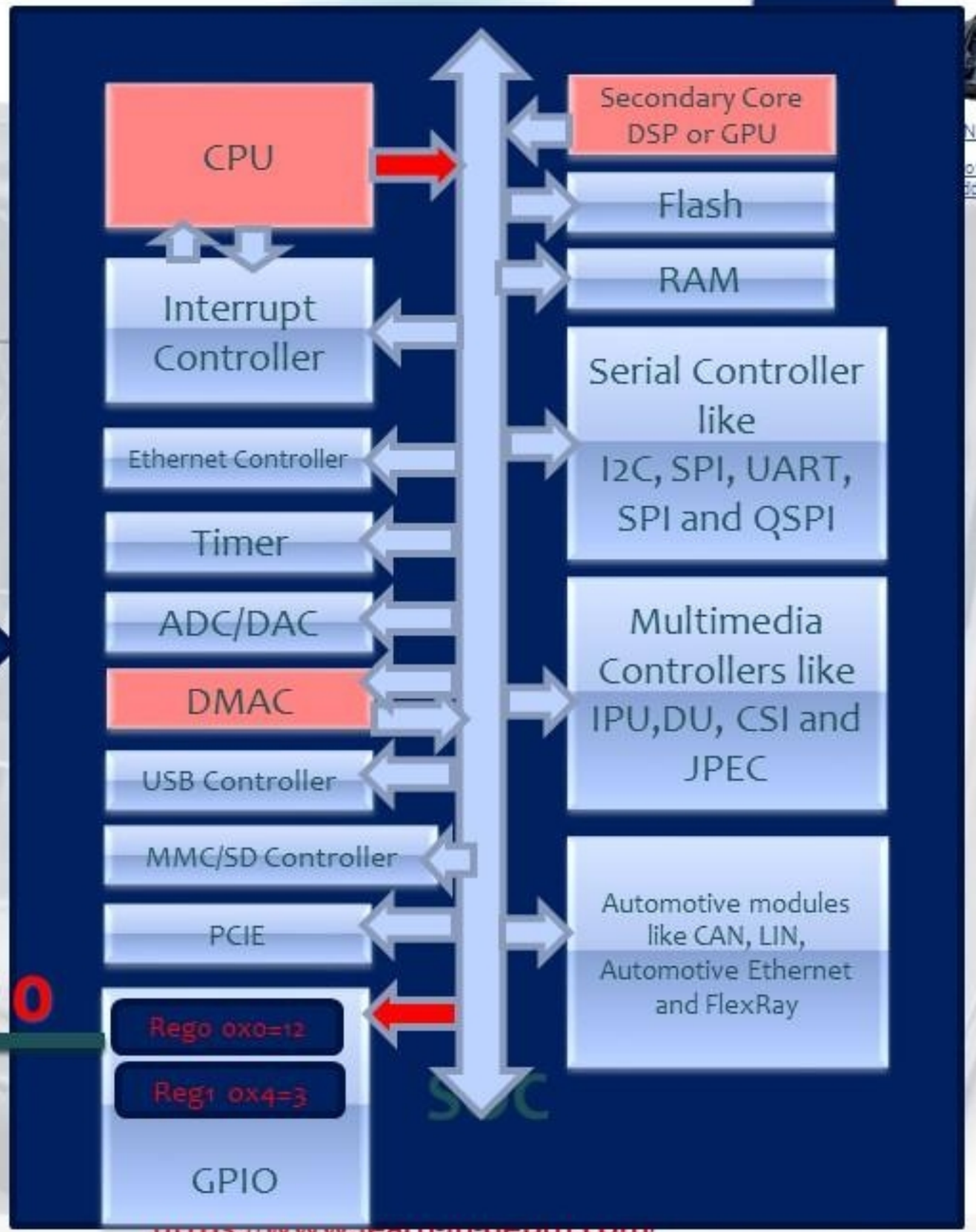
S ip (SOC)

```
(*(volatile unsigned long*)
0XFFFF0000) = 12 ;
(*(volatile unsigned long*)
0XFFFF0004) = 3 ;
```

Embedded C (executable file)



ECU



1 or 0

N IN DEPTH

ofessional in
dded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
7
#LEARN IN DEPTH
#Be professional in embedded system

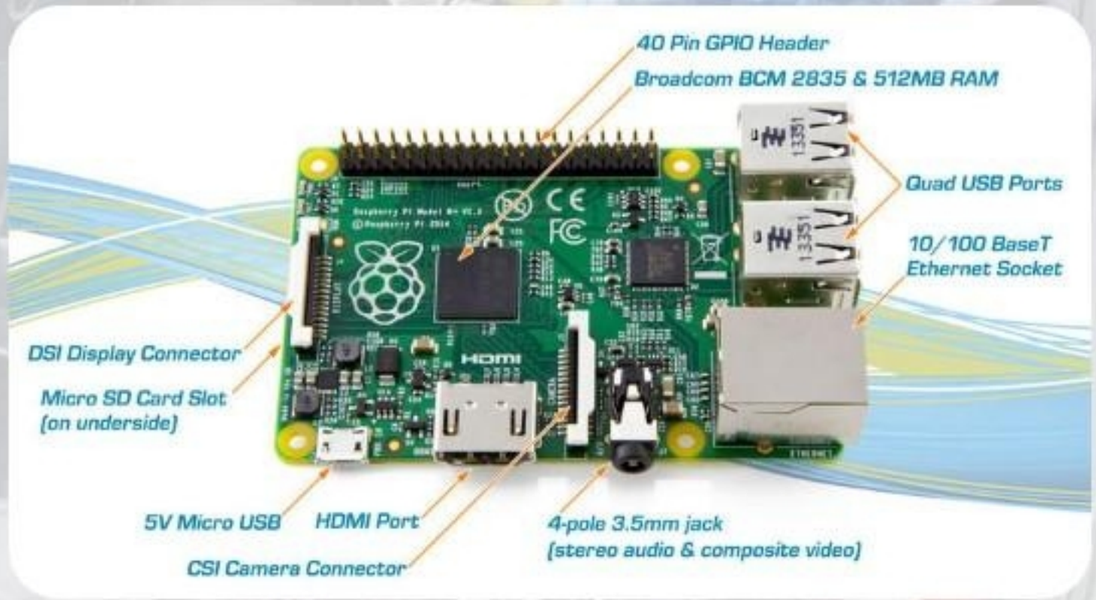
SOCs Examples: Raspberry Pi 2

The Raspberry Pi 2 Model B is the second generation Raspberry Pi. It replaced the original Raspberry Pi 1 Model B+ in February 2015. Compared to the Raspberry Pi 1 it has:

- A 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM

Like the (Pi 1) Model B+, it also has:

- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

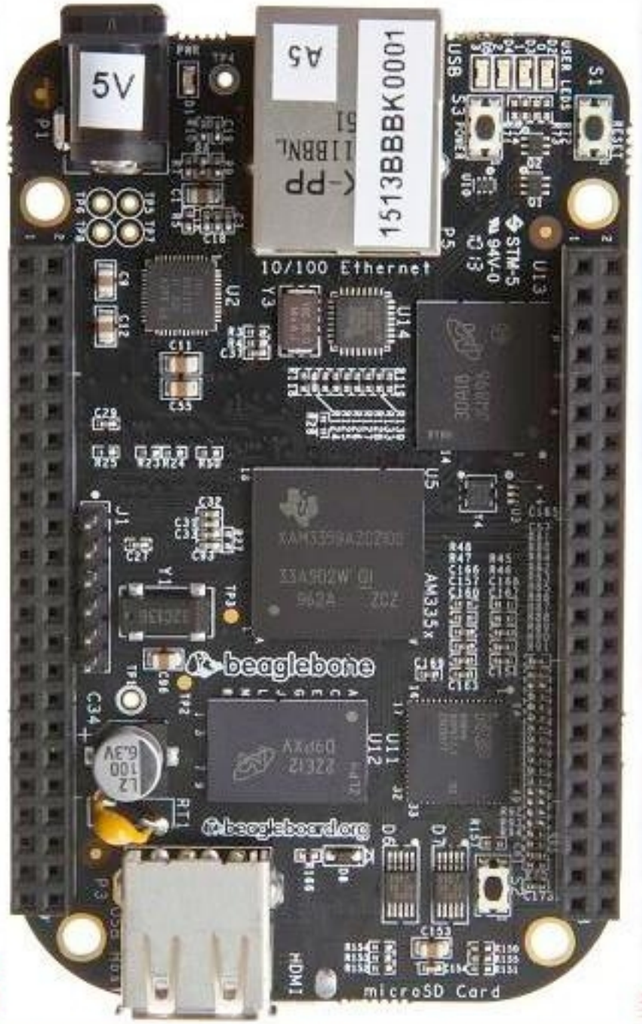


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
8
#LEARN IN DEPTH
#Be professional in embedded system

SOCs Examples: BeagleBone Black

- Processor: AM335x 1GHz ARM® Cortex-A8
  - 512MB DDR3 RAM
  - 4GB 8-bit eMMC on-board flash storage
  - 3D graphics accelerator
  - NEON floating-point accelerator
  - 2x PRU 32-bit microcontrollers
- Connectivity
  - USB client for power & communications
  - USB host
  - Ethernet
  - HDMI
  - 2x 46 pin headers



FOLLOW US
Press here
9
#LEARN IN DEPTH
#Be professional in embedded system

Renesas RH850D1M

484-pin product

System
- PLL & SSCG
- Security / SHE
- System Health Mon.
  - WDT
  - Temp. Sensor
  - Clock Monitor
  - POC/LVI

Package: BGA484 / 1mm

32-bit CPU
- RH850 Core
  240 MHz
  3.0 – 5.5V, 1.2V (core)
  TJ: -40 to +150°C
- FPU, MPU
- On-Chip debug

Memory
- 3.75 MB / 5 MB Code Flash 512 KB iRAM
- 8 KB Flash$ | 32 KB Data$ | 64 KB Data Fl.

Automotive Interfaces
- 3 x CAN / CAN FD*
- 6 x CSI
- 4 x LIN UART
- 2 x I²C
- Ethernet AVB
- MLB50

Standard
- 16 x DMA
- 20 x A/D (12-bit)
- 48 x 16-bit Timer
- 7 x 32-bit Timer
- 24 x 12-bit PWM
- 1x RTC

Graphic Subsystem (240MHz)
- 2D Drw (D/AVE-HD) OpenVG1.1, BitBLT, RLE
- JPEG & Warping Hardware Engines
- Sprite Engine
- Vout Checker
- 2 x Vout(RGB, RSDS)
- Ser. Flash SDR/DDR
- 2 x Vin(RGB, CSI2)
- 3 MB VRAM
- DDR2-480 32-bit

(*) CAN FD is optional

Cluster
- 6 x SMC/ZPD
- 5 x Sound Generator
- PCM Sound
- 2 x I²S Sound

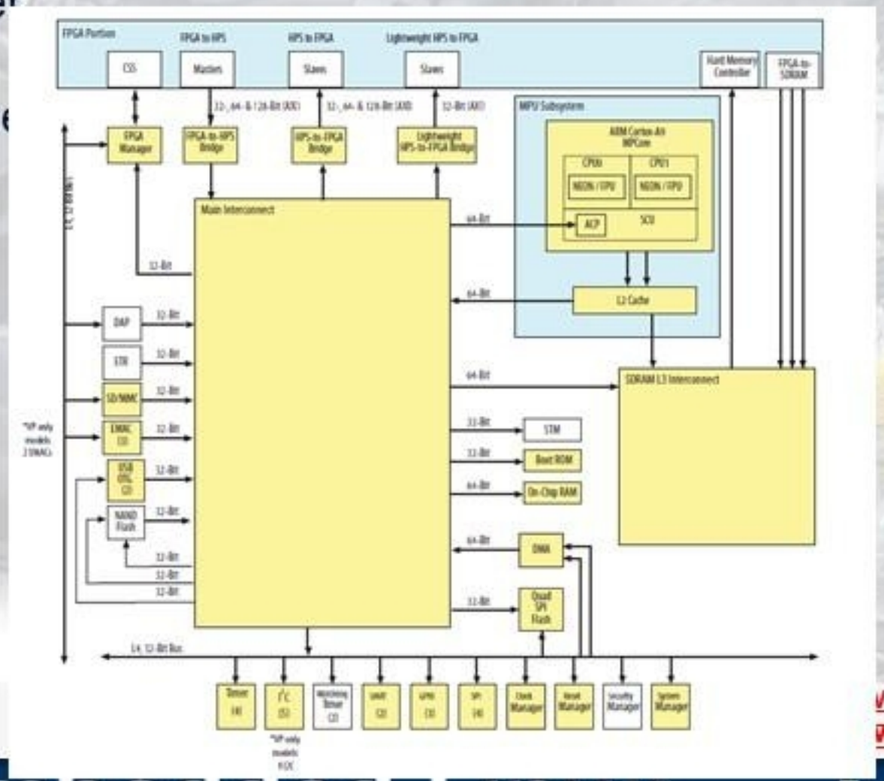
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

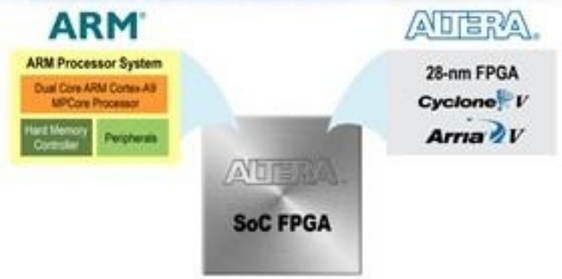
SOC And FPGA

SoC FPGA devices integrate both processor and FPGA architectures into a single device. Consequently, they provide higher integration, lower power, smaller board size, and higher bandwidth communication between the processor and FPGA. They also include a rich set of peripherals, on-chip memory, an FPGA-style logic array, and high speed transceivers.

#LEARN IN DEPTH

#Be professional in embedded system





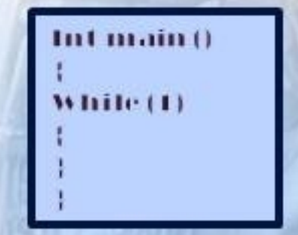
ARM + Altera = SoC FPGAs

www.learn-in-depth.com

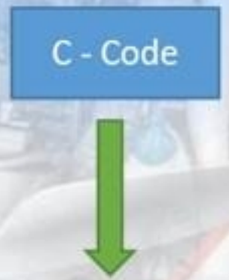
www.facebook.com/groups/embedded.system.KS/

Bare metal SW

- "Bare metal" means **your application** is accessing the silicon chip **directly** without any intermediary like an OS.
- The application is the only software that executes on the microprocessor/microcontroller.
- So we can considered that the Drivers is a bare metal Code and the bios also.



Bare-Metal application



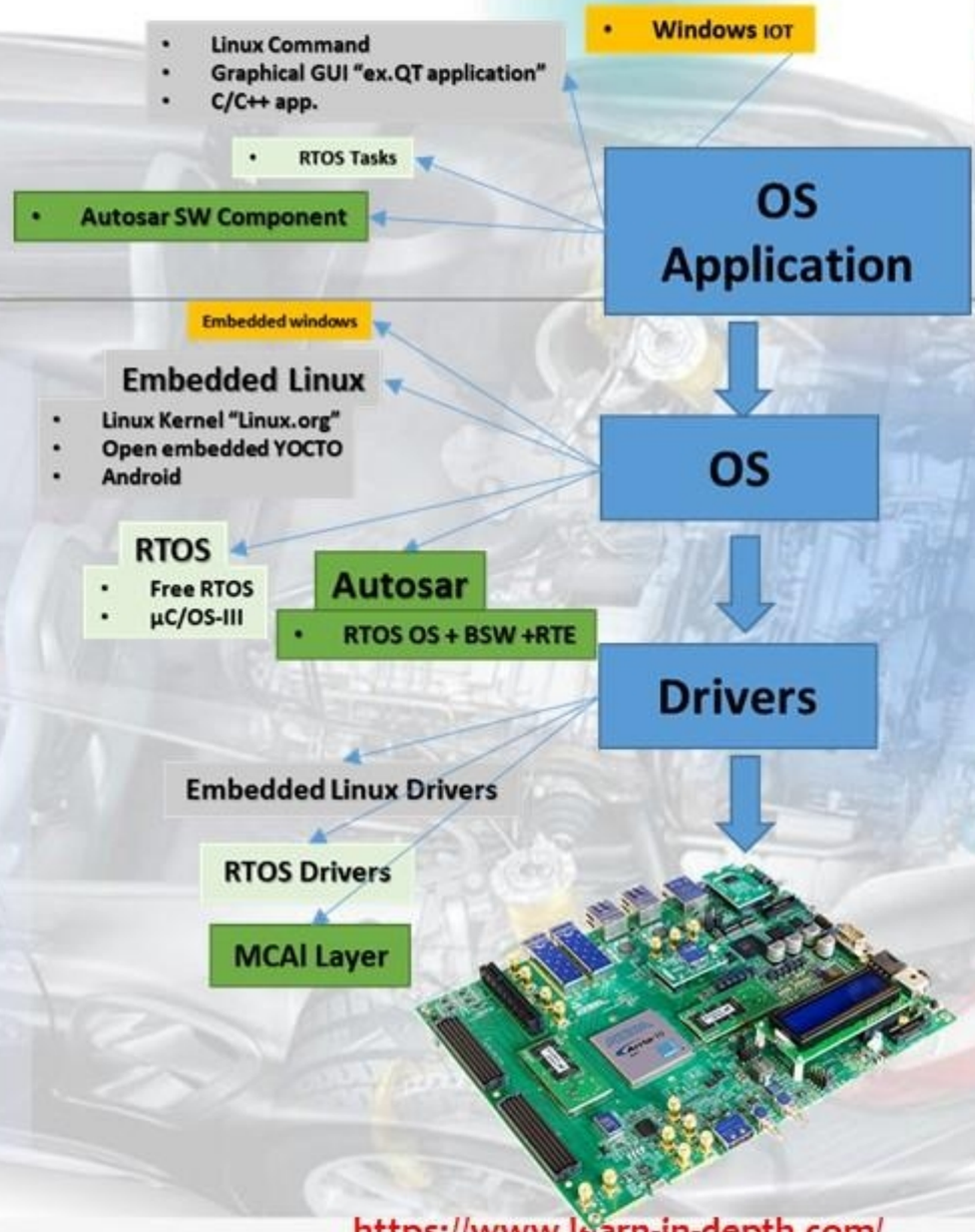
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
12

#LEARN IN DEPTH
#Be professional in embedded system

OS Application

- Write your application on the top of the Operating System.
- On desktop computers, the selection of an operating system (OS) is largely a matter of taste - Windows vs Apple vs Linux. There is relatively little choice.
- For an embedded system, the matter is much more complex. The large number of options available reflect the wide diversity of embedded applications.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
13

#LEARN IN DEPTH

#Be professional in embedded system

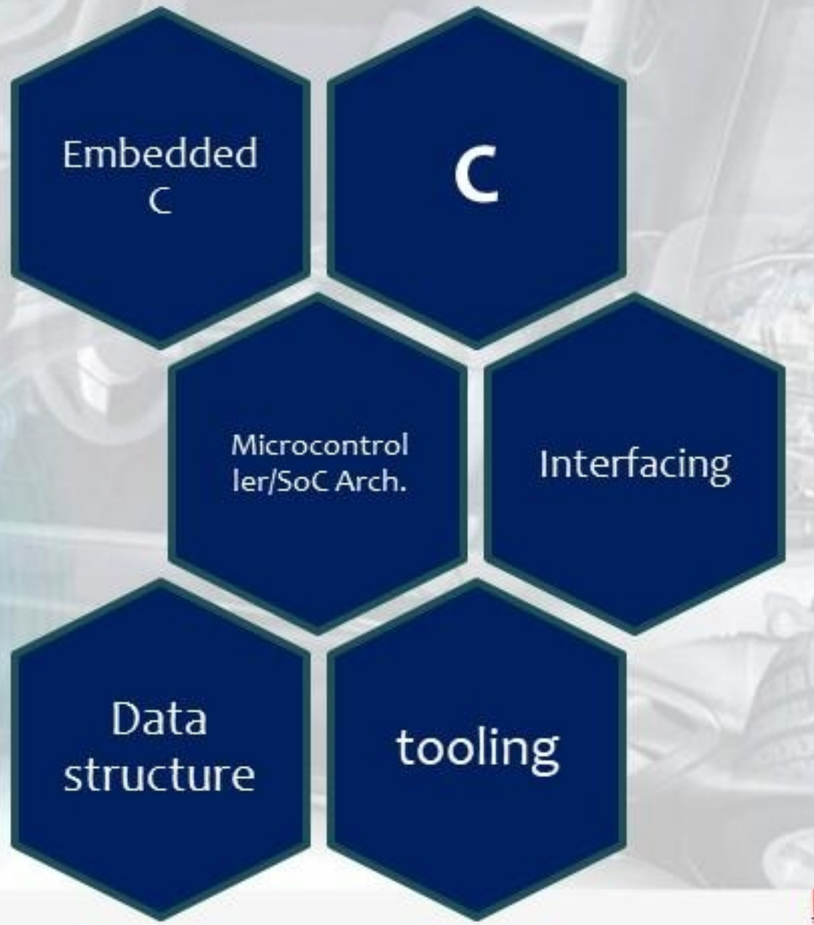
Embedded System Fields

- embedded System Basics
- Automotive Field
- Wireless Embedded System
- Embedded Linux field

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
#LEARN IN DEPTH
#Be professional in embedded system

embedded System Basics



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN IN DEPTH

#Be professional in embedded system

To learn C/EmbeddedC

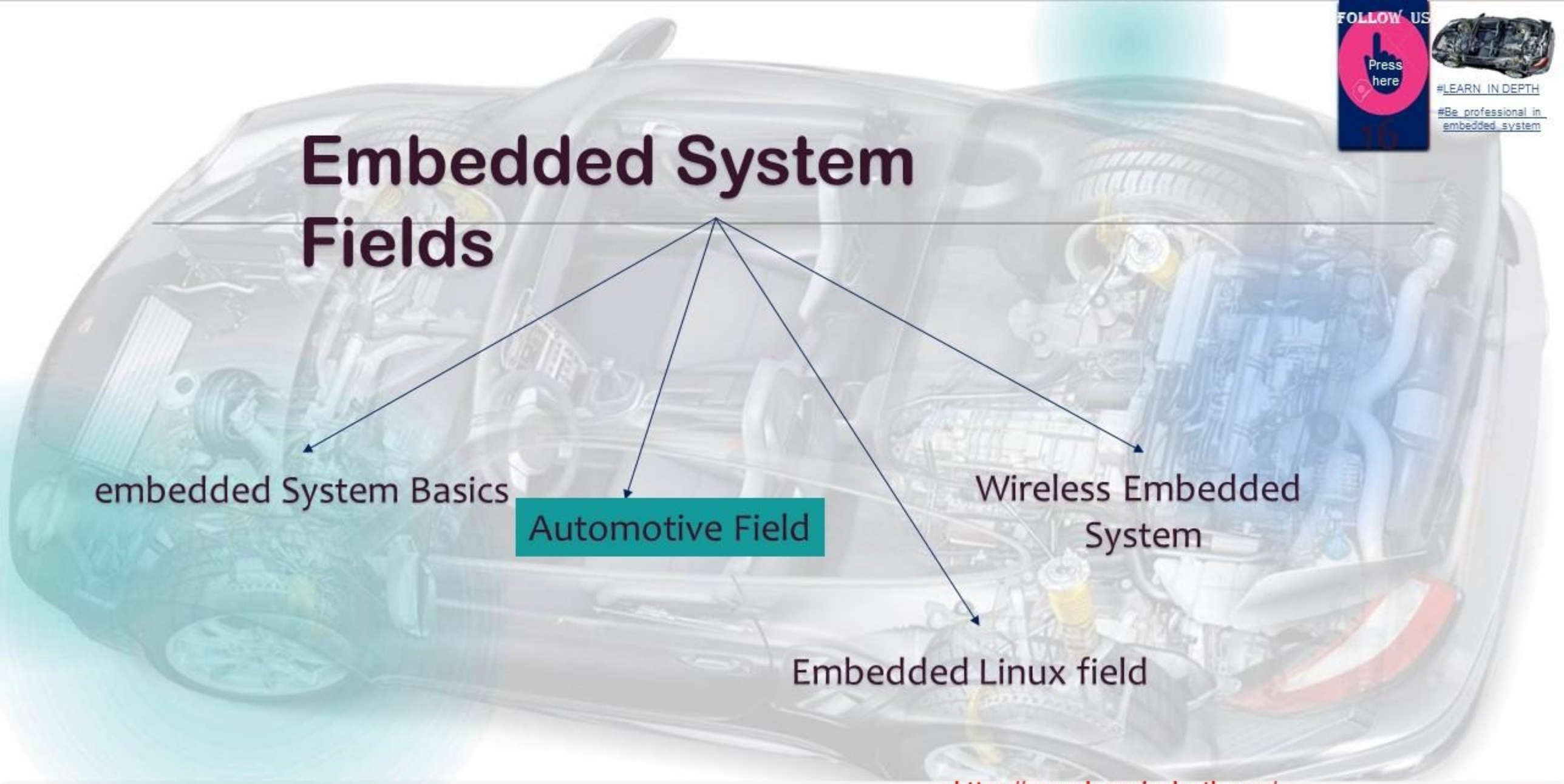
Step 1: Study the following Parts
==========
Part 1 : the Compiling process and charroduction to C Programming
Part2 : C Variables & Comments & Data Types & printf/scanf and Type casting
Part3: C fundamentals & C statements
Part4: Arrays & Strings
Part 5: C Functions
Part 6: Macros & #pragma
Part7: Structures & ENUM & UNION
Part8: pointers
Part9: Memory Allocation & Embedded C & Queue & Linked List
PART10: Embedded C (startup & Linker &)
Kindly you can find all the parts on:
https://www.learn-in-depth.com/embedded-c

Microcontroller/Interfacing

Step 1: Study the following Parts
==========
Part 1 : Memory, internal structure of a microcontroller, clocks, CPU registers, Von Neumann/Harvard Arch, pipeline And CISC/RISC
Part2 : GPIO Module
Part3: Keypad And LCD
Part4: Interrupt
Part 5: Timers
Part 6: ADC
Part7: UART
Part8: SPI
Part9: I2C
Part10: DDR4 SDRAM

Kindly you can find all the parts on:
https://www.learn-in-depth.com/soc

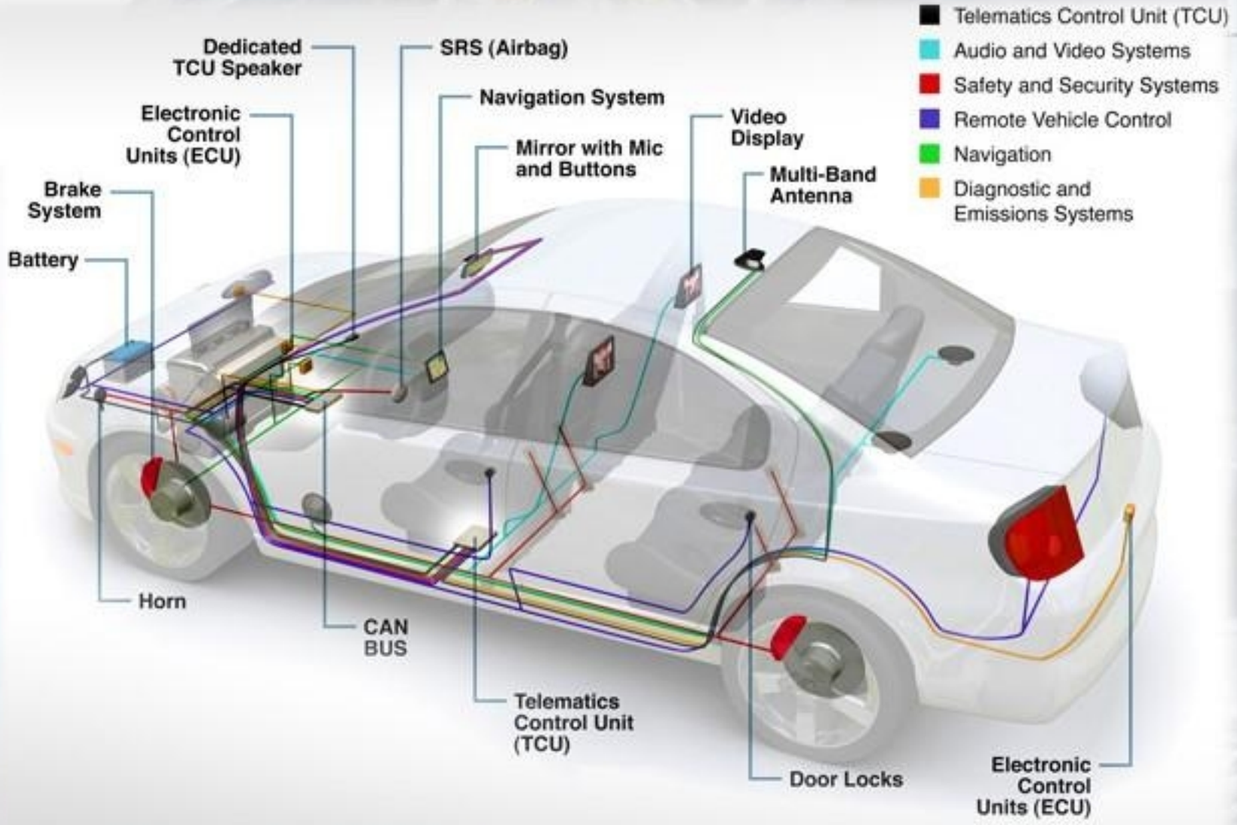
http
htt
bedded.system.KS/



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
#LEARN IN DEPTH
#Be professional in embedded system

Embedded Automotive



Domain	Domain	HW Scope	View	SW Scope
OEM	Vehicle		OEM	Application Layer (f_1 f_2 ... f_n)
OEM Tier 1	Subsystem			Middleware ($task_1$ ... $task_m$ msg_1 ... msg_i)
Tier 1	ECU			Std. Services, ECU Abstraction, MCU Abstraction (MCAL), Complex Drivers
Tier 2	MCU		Suppliers	

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
#LEARN IN DEPTH
#Be professional in embedded system

Automotive V Cycle

1 Vehicle requirement

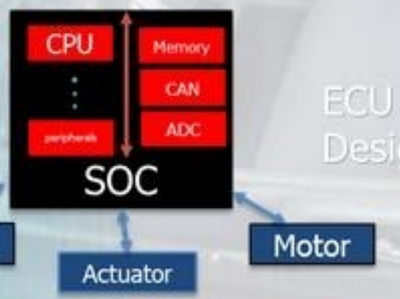
If Vehicle Validation failed

7 Vehicle Validation

2 System Design

If system Verification failed

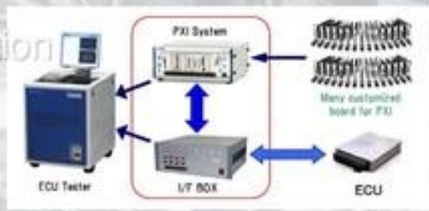
6 System Verification



3 ECU Design

If ECU Validation failed

5 ECU Verification



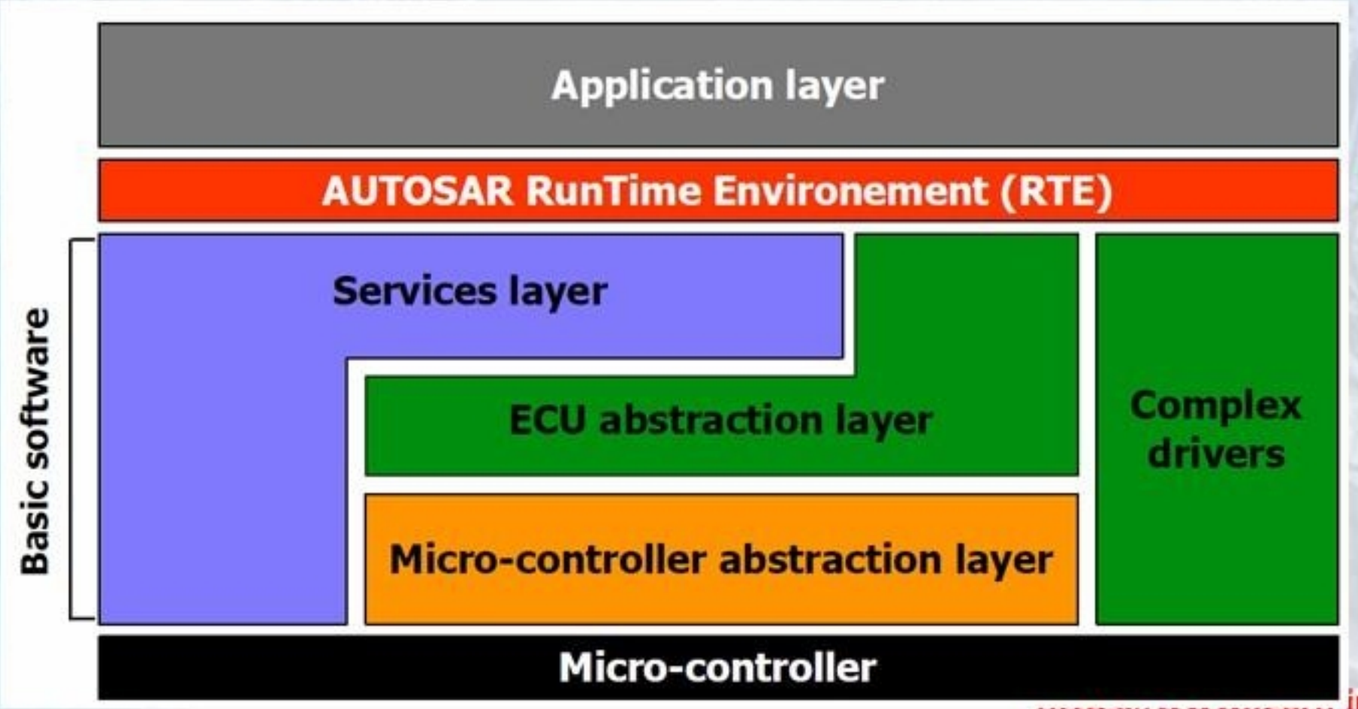
4 SOC Design

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
#LEARN IN DEPTH
#Be professional in embedded system

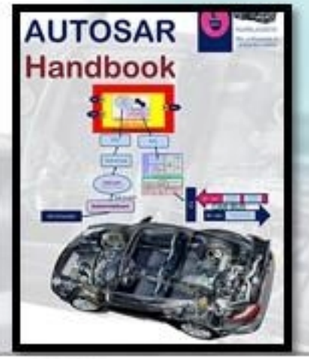
AUTOSAR

- AUTOSAR = AUTomotive Open System Architecture
- More informations on the site: http://www.autosar.org



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

learn Embedded Automotive (AUTOSAR/RTOS) Basics



#LEARN IN DEPTH

#Be professional in embedded system

Step 1: Study the following Parts

==========

Automotive Embedded System Part 1 (RTOS Basics)
Automotive Embedded System Part 2 (OSEK-vDX) PART1
Automotive Embedded System Part 3 (OSEK-vDX) PART2
Automotive Embedded System Part 4 (OSEK-vDX) PART3
Automotive Embedded System Part 5 (Introduction to AUTOSAR)
Automotive Embedded System Part 6 (AUTOSAR Application Layer)
Automotive Embedded System Part 7 (CAN)
Automotive Embedded System Part 8 (CANFD, TTCAN ,LIN and FlexRay)

Kindly you can find all the parts on

https://www.learn-in-depth.com/automotive

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

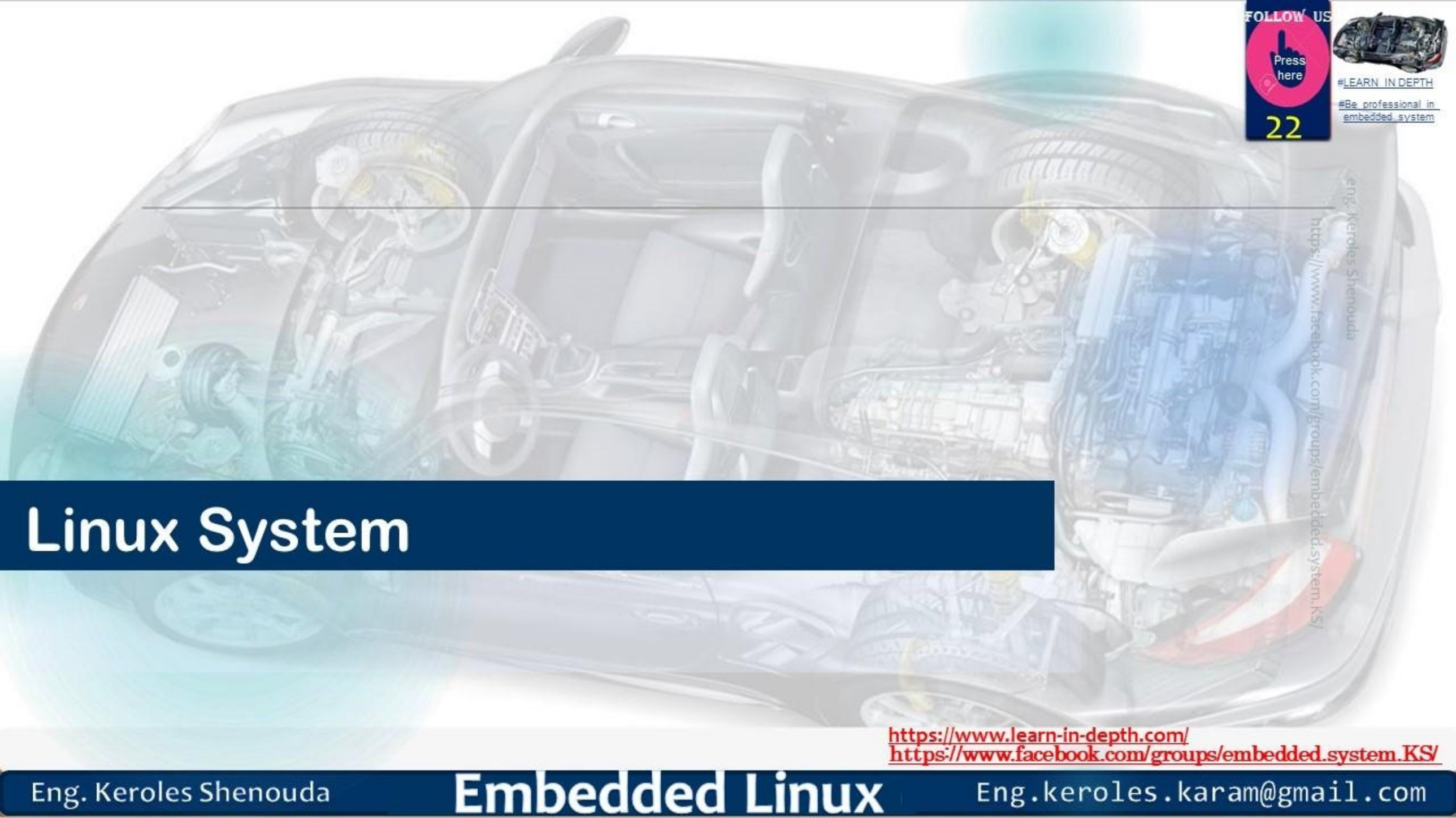
Eng.Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
#LEARN IN DEPTH
#Be professional in embedded system

Embedded System Fields

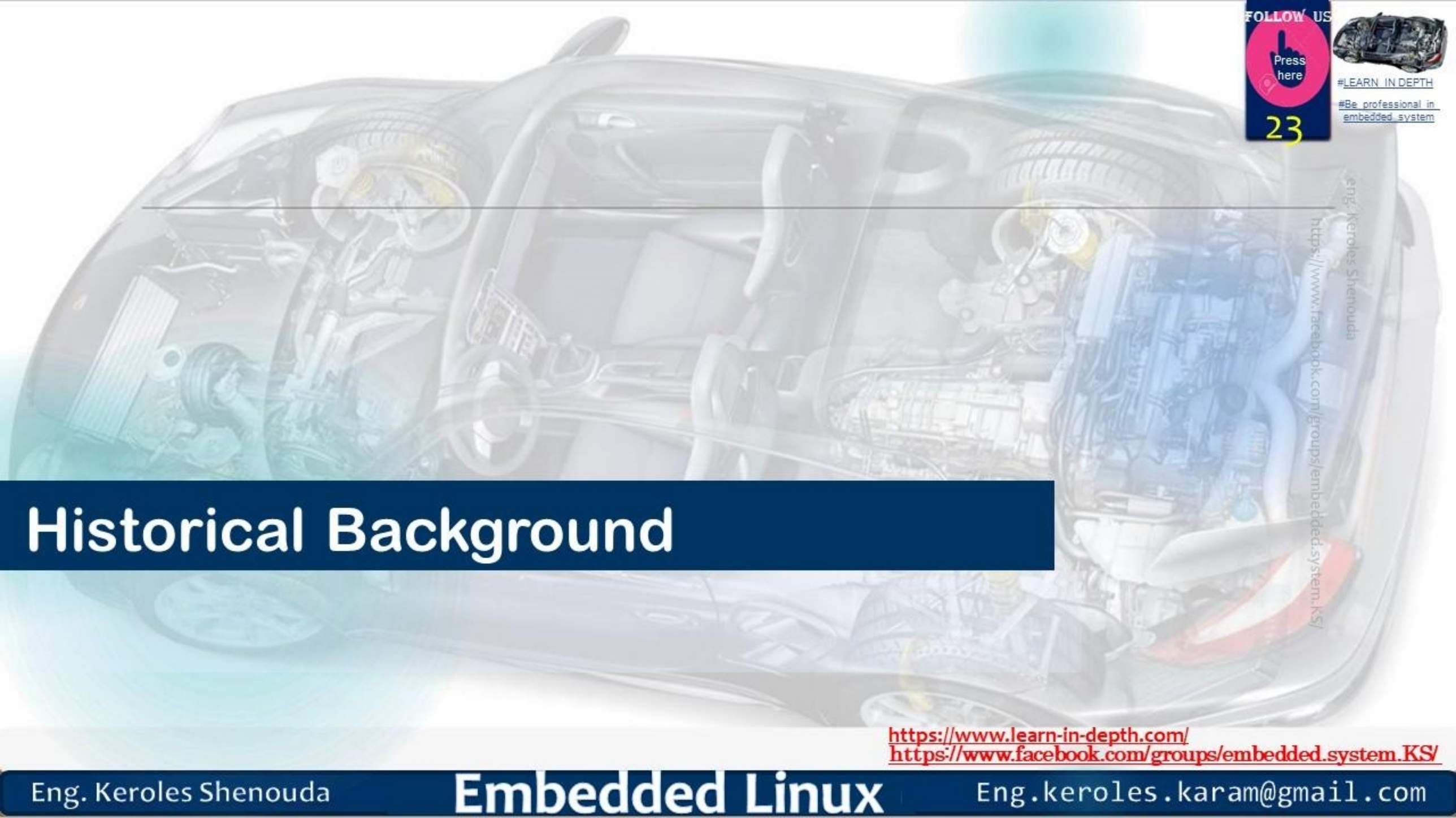
- embedded System Basics
- Automotive Field
- Wireless Embedded System
- Embedded Linux field

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Linux System

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Historical Background

LEARN IN DEPTH
fessional in
ed system

Historical Background

- It started in bell Labs with terminated project for Multics Multi-user operating system
- Dennis Ritchie and Ken Thomson started to work on Unix 1969: first Implementation of Unix.
- 1972:Creation of "C" Programming Language to facilitate the porting.
- 1973: Complete rewrite of UNIX into "c"
- Richard Stallman is Believing in free Software, he formed the Free Software Foundation and Started the GNU [Gnu Not Unix] Project in 1983.

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones.

(Linus Torvalds)

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
#LEARN IN DEPTH
#Be professional in embedded system

What is Linux?
Linux + GNU Utilities = Free Unix

- Linux is an O/S core written by Linus Torvalds and others AND

- a set of small programs written by Richard Stallman and others. They are the GNU utilities.

http://www.gnu.org/

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

What is Linux?

It's an Operating System

- Linux and Unix strive to be POSIX compliant.
- 64% of the world's servers run some variant of Unix or Linux.
- The Android phone and the Kindle run Linux.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
#LEARN IN DEPTH
#Be professional in embedded system

Linux Has Many Distributions

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
28

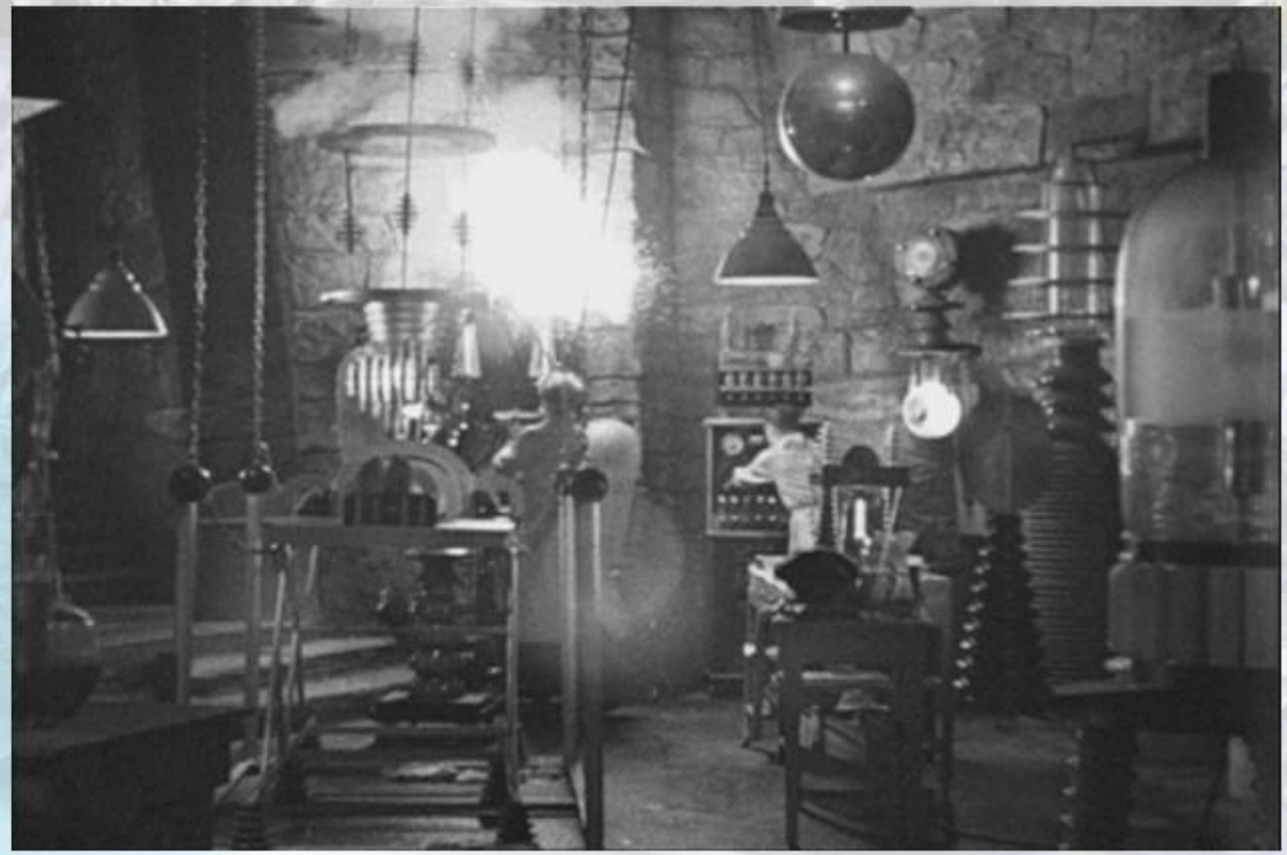
#LEARN IN DEPTH
#Be professional in embedded system

Linux Has Many Distributions

- Some users uses CentOS in its Linux cluster which is a free version of RedHat Enterprise Linux with the trademarks removed

Eng.Keroles Shenouda https://www.facebook.com/groups/embedded.system.KS/

Let the Linux Lab Begin!



The Ideal Lab Facility

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
30
#LEARN IN DEPTH
#Be professional in embedded system

How Linux is built

https://www.facebook.com/groups/embedded.system.KS/
Eng. Keroles Shenouda

https://www.youtube.com/watch?v=yVpbFMhOAwE

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
31

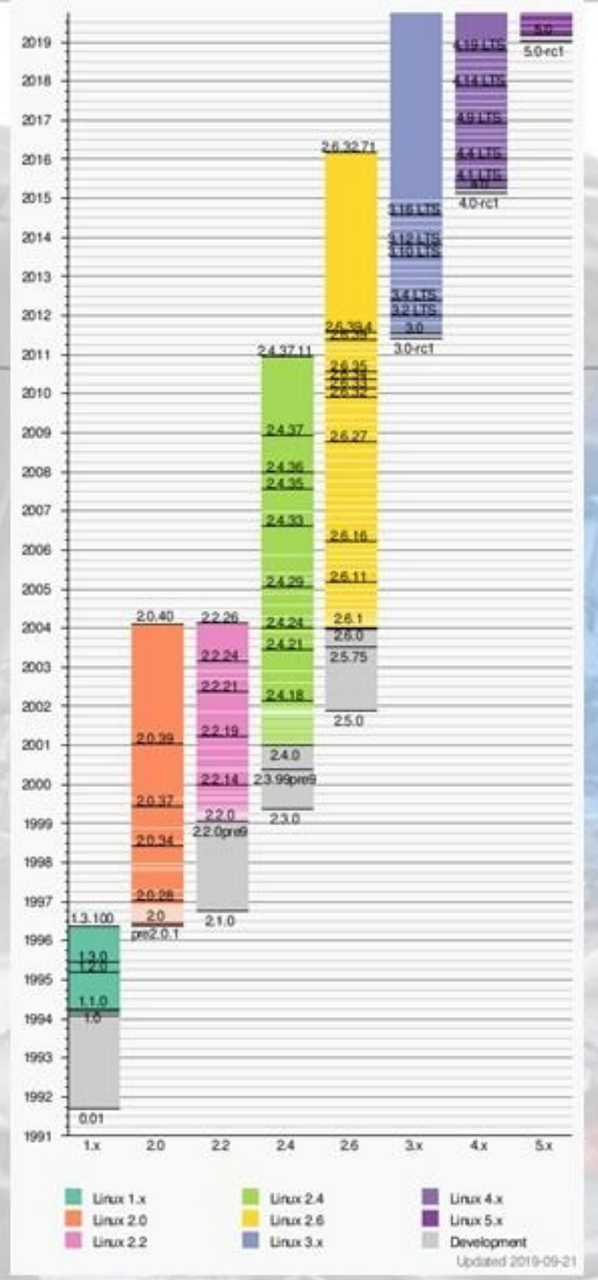
#LEARN IN DEPTH
#Be professional in embedded system

Terminologies

- The term **Linux** refers to the **operating system kernel itself**
- However Linux is also used to refer to **a complete Linux system** that includes not only kernel but also libraries and application programs.
- The Linux distribution includes the basic linux system system installation and management utilities
  - Ubuntu, fedora and ….
- Linux kernel is distributed by a license called **GPL**.
- Linux kernel is a free software.
  - It means people can get Linux source code modify it use it and give away their own copies without any restrictions. However nobody can claim the ownership of outputs developed by using Linux and they cannot distribute binary.
  - only products when they distribute outputs developed by software **under GPL license.**
    - It means the **source code** should be available in addition **to the binary**.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Linux version history



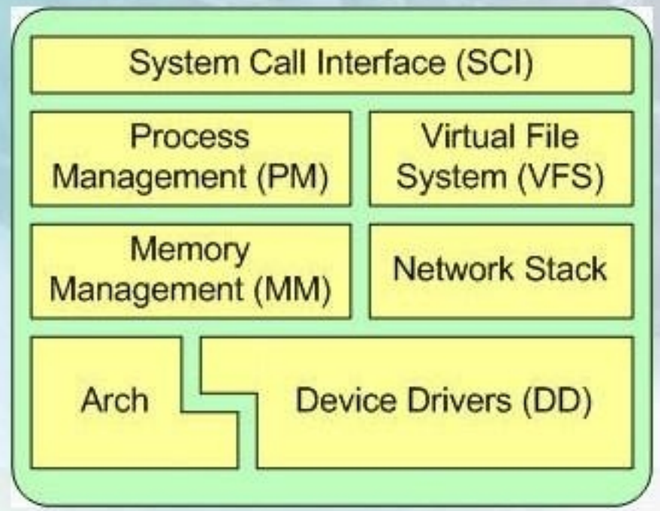
FOLLOW US
Press here
32
#LEARN IN DEPTH
#Be professional in embedded system
Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

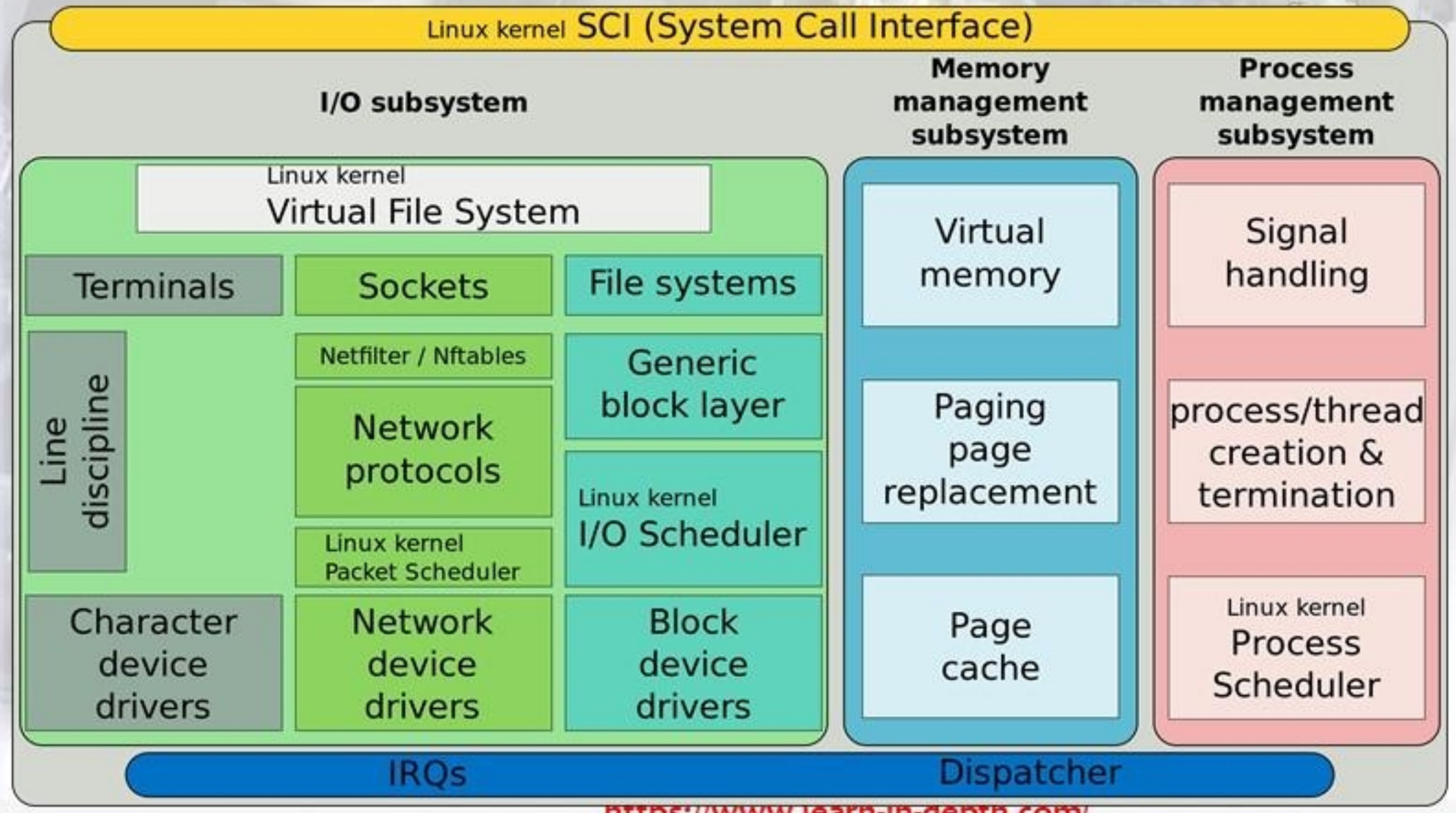
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
33
#LEARN IN DEPTH
#Be professional in embedded system

Linux kernel Components

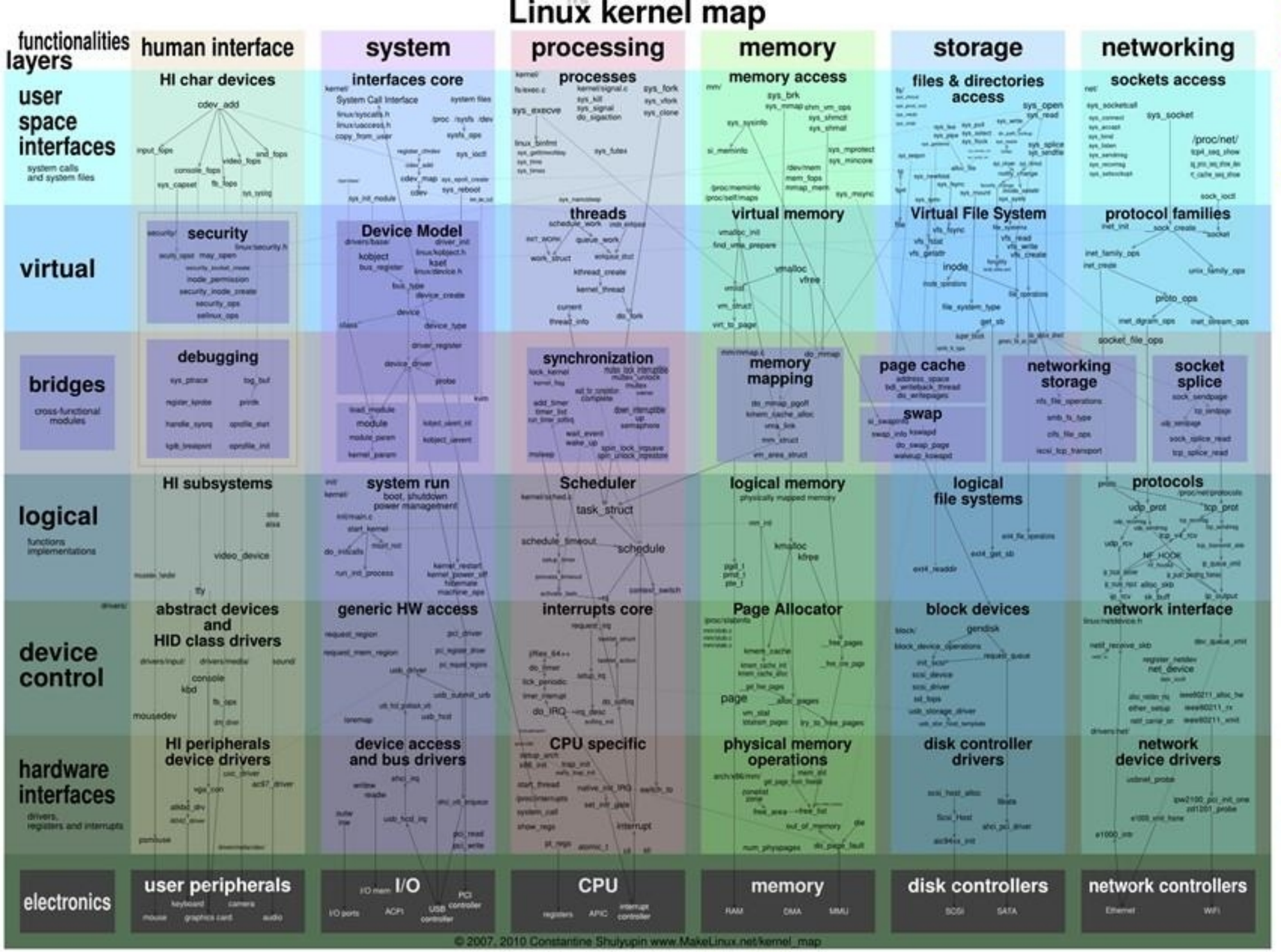
- It is mainly composed of
  - process management
  - memory management
  - files systems
  - network protocols and
  - device drivers.





https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Linux kernel (Big Picture)



#LEARN IN DEPTH

#Be professional in embedded system

https://www.facebook.com/groups/embedded.system.KS/ Eng.Keroles Shenouda

http://www.makelinux.net/kernel_map/?fbclid=IwARorqSwJgnPPwu2wjxoDvGhMdKbtNPJbp6sisiXdf-ETqPvvwM-YS6goNym8

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
#LEARN IN DEPTH
#Be professional in embedded system

Embedded Linux

Power ON

PreBootloader

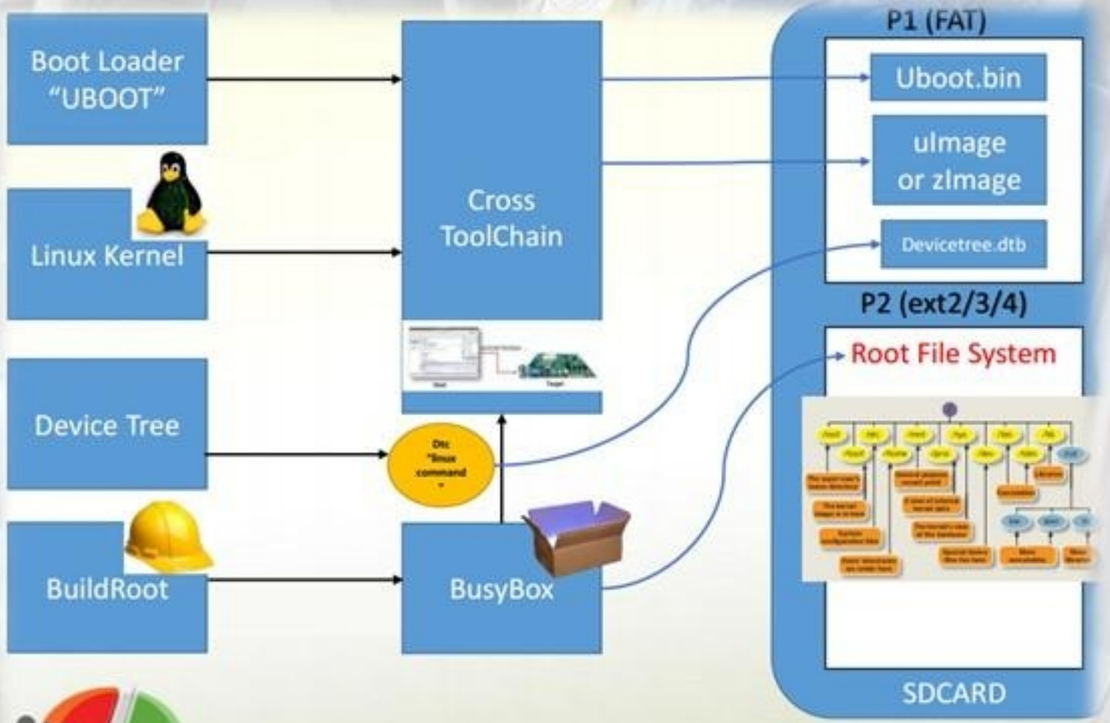
Main Bootloader "UBOOt"

Linux Kernel

Mount root FS

Invoke /sbin/init process

Linux Prompot



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
#LEARN IN DEPTH
#Be professional in embedded system

Deep Learning

- **Computer Vision Applications**

1. Security systems using, for example, face recognition
2. Medical Examinations
3. Traffic Sign Recognition
4. Paper reading for the blind
5. Google's Image search

- And many others…

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
#LEARN IN DEPTH
#Be professional in embedded system

Traffic Sign Recognition

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
#LEARN IN DEPTH
#Be professional in embedded system

Deep Learning – Datasets

Dataset

GTSDB

GTSRB

https:/
https-
.KS/

FOLLOW US
Press here
39

#LEARN IN DEPTH

#Be professional in embedded system

eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

Course Outline

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
40

#LEARN IN DEPTH

#Be professional in embedded system

What you will learn ?

Phase 1
Understanding LINUX

Phase 2
Building and Booting LINUX

Phase 3
Linux Device Drivers and embedded Linux SW

- **Memory Management Basics**
- Loading, Linking and Address Binding
- Memory Management Concepts
- Address Binding
  - **Compile Time Binding**
  - **Load Time Binding**
  - **Execution Time Binding**
- Memory Allocation
- Fragmentation
- Contiguous and Noncontiguous Memory Allocation
- Paging
- T.L.B (Translation look-aside Buffer)
- **I/O Systems**
- **POSIX APIs**
- **Process and Threads**
- **Linux Commands**

- Buildroot
- Root FileSystem
- Linux Kernel
- Bootloader
- ToolChain

- Kernel Modules
- Linux Device Derivers
- Programming I2C, UART
- Programming GPIO
- ACCESS Display
- Advanced Topics

Eng.Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
41

#LEARN IN DEPTH
#Be professional in embedded system

Virtual Platforms

- VPs are software architectural-level models of complete real systems that can include:
- Processor(s)
- Peripheral components
- Memories
- Interconnect

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
42

#LEARN IN DEPTH
#Be professional in embedded system

Virtual Platforms Benefits

- Hardware/software co-development at ESL (Electronic System Level)
- Efficient software development, debug, and performance analysis of application code, drivers, etc.
- Create early platform for architecture exploration and trade-off analysis
- Validate system functionality before implementation
- Easier, shorter, and less risk final integration stage where real silicon hardware is used
- Faster time to market and lower cost

Eng.Keroles Shenouda https://www.facebook.com/groups/embedded.system.KS/

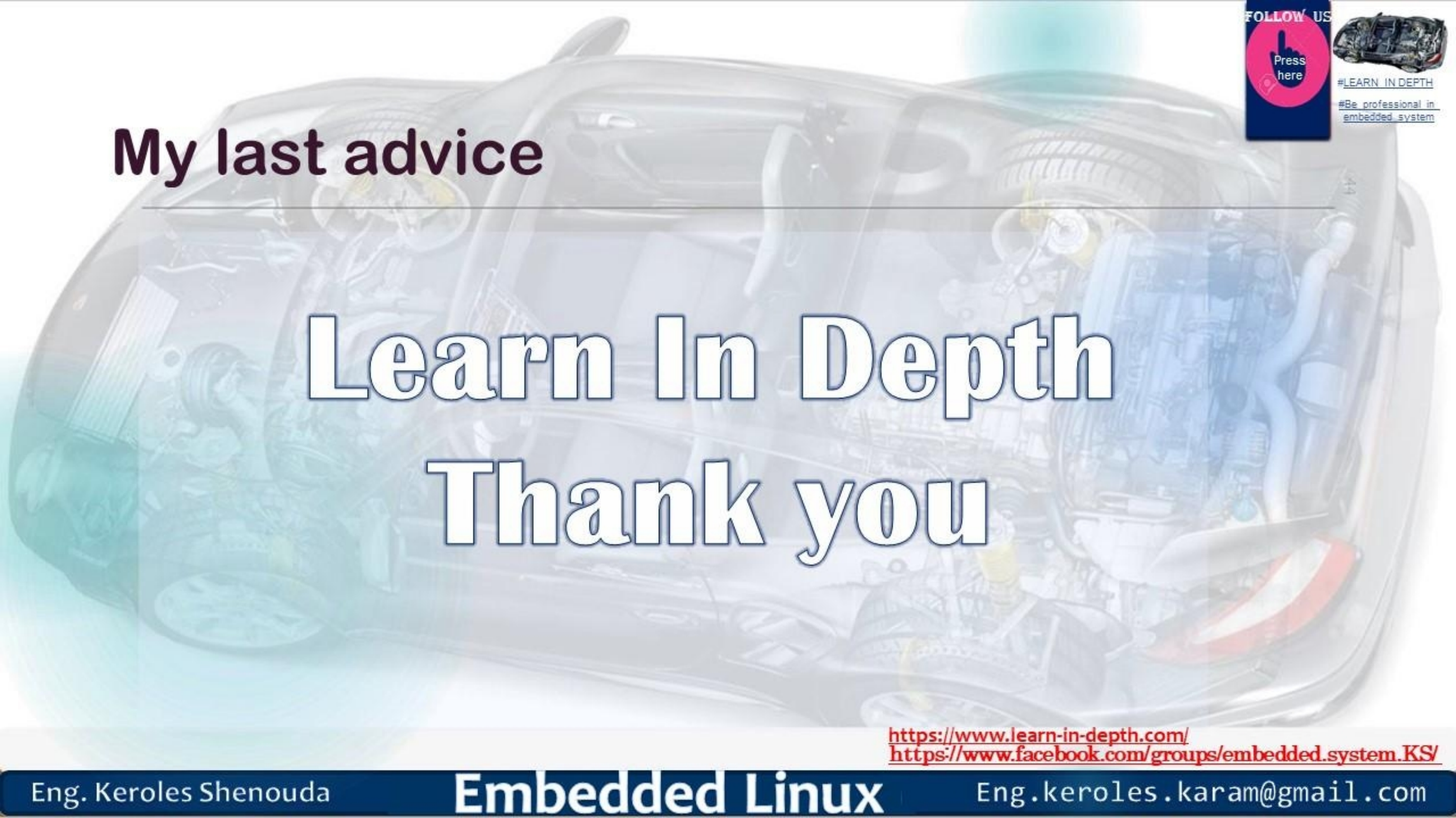
FOLLOW US
Press here
43

#LEARN IN DEPTH
#Be professional in embedded system

QEMU

- QEMU is a generic and open source machine emulator and virtualizer.
- When used as a machine emulator, QEMU can run OS and programs made for one machine (e.g. an ARM machine) on a different machine (e.g. your own PC).
- Machine ≡ Board ≡ Platform
- www.qemu.org

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/



My last advice

Learn In Depth
Thank you

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
45

#LEARN IN DEPTH

#Be professional in embedded system

eng.Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

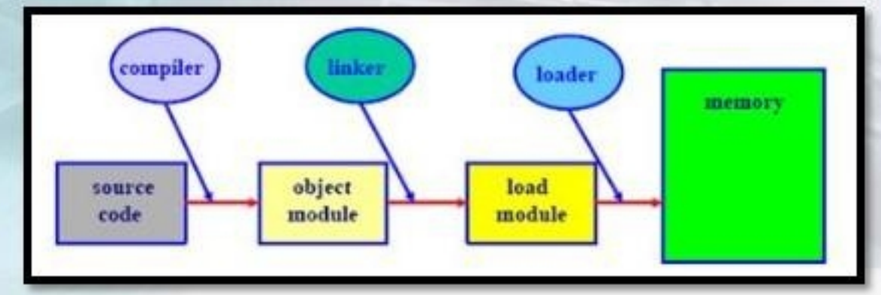
Define Loading, Linking and Address Binding

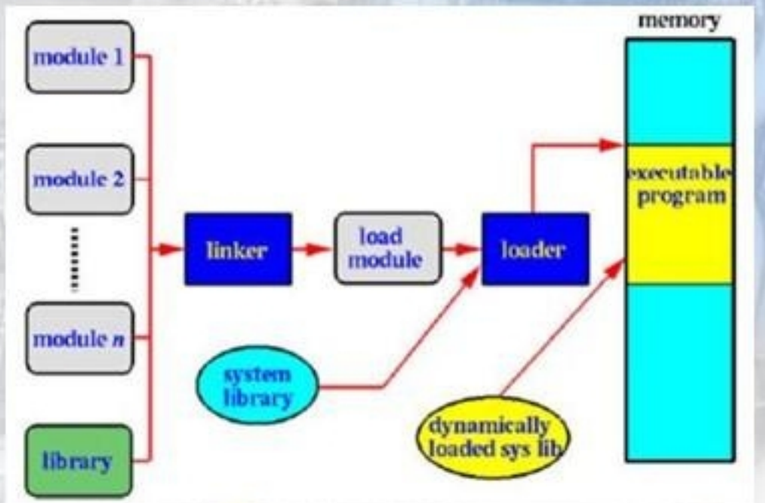
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
46
#LEARN IN DEPTH
#Be professional in embedded system

Define Loading, Linking and Address Binding

- Loading:
  - Bringing **Program** from **Secondary Memory** to main Memory
- Linking:
  - Establishing Linking between all modules of program (object files and libs) to one executable file according to Linker script.
  - Types of linking Static and dynamic Linking.



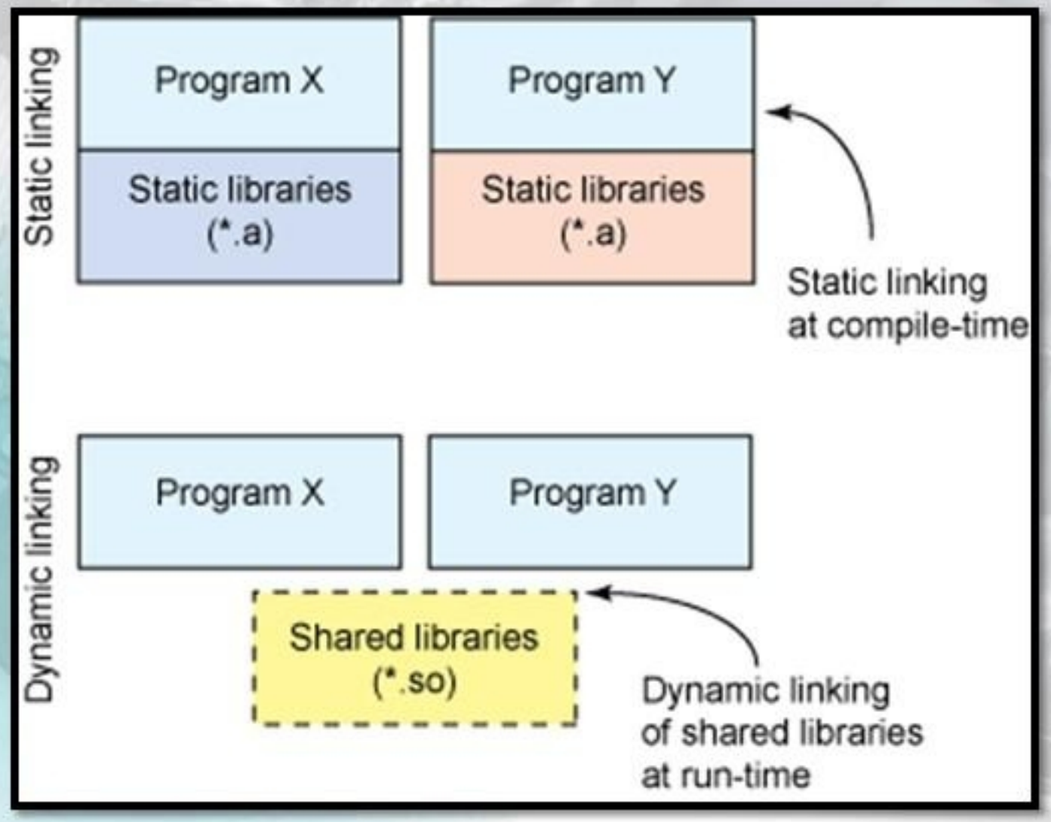


https://www.facebook.com/groups/embedded.system.KS/

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
47
#LEARN IN DEPTH
#Be professional in embedded system

Static Vs dynamic linking

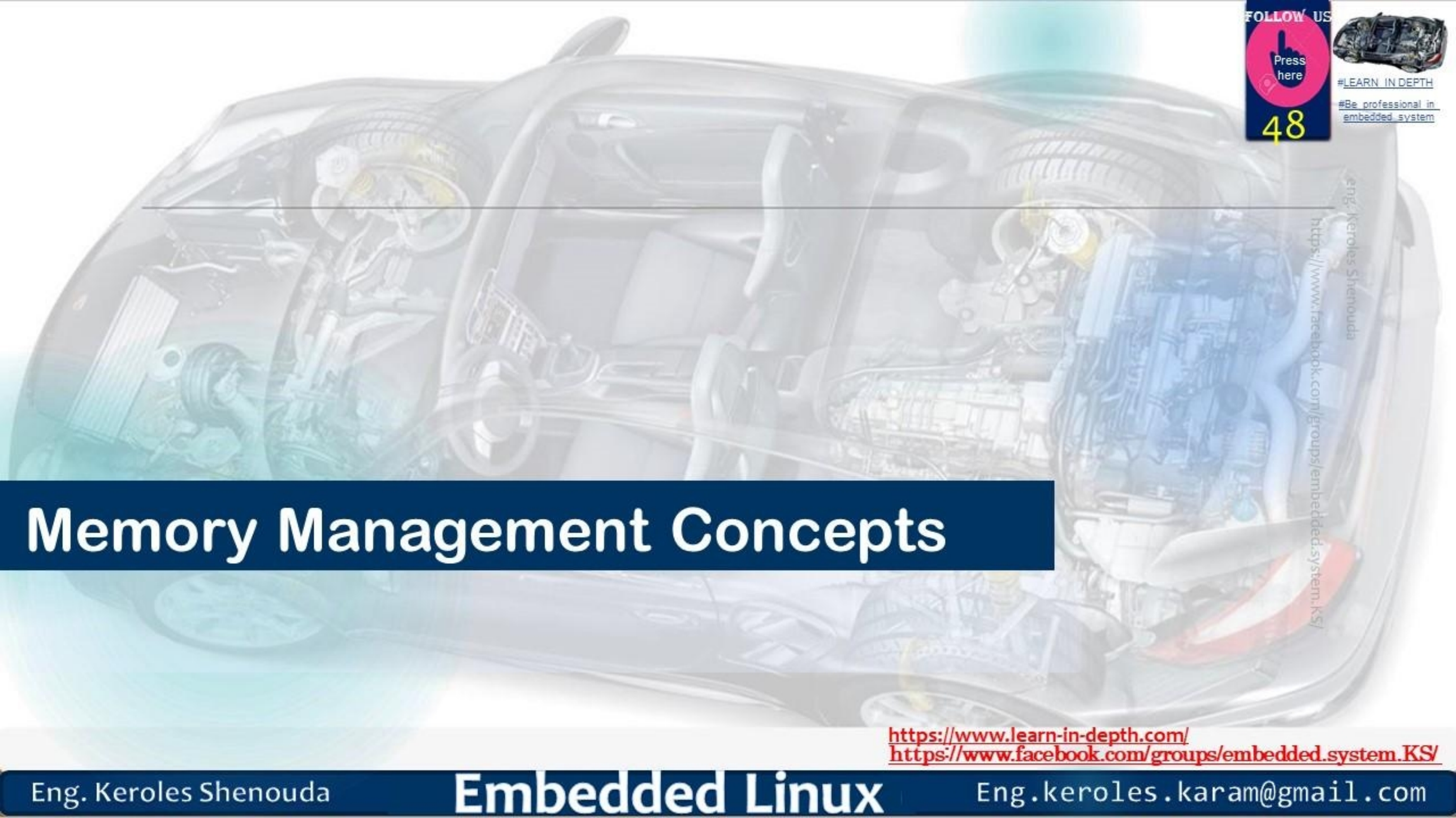


STATIC LINKING VERSUS DYNAMIC LINKING

STATIC LINKING	DYNAMIC LINKING
Process of copying all library modules used in the program into the final executable image	Process of loading the external shared libraries into the program and then bind those shared libraries dynamically to the program
Last step of compilation	Occurs at run time
Statistically linked files are larger in size	Dynamically linked files are smaller in size
Static linking takes constant load time	Dynamic linking takes less load time
There will be no compatibility issues with static linking	There will be compatibility issues with dynamic linking

Visit www.PEDIAA.com

https://www.facebook.com/groups/embedded.system.KS/



Memory Management Concepts

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
49
#LEARN IN DEPTH
#Be professional in embedded system

Address Binding

- Address binding is **the process of mapping** the program's logical or virtual addresses to corresponding physical or main memory addresses.
- This Process moves between memory and the storage device like disk or sdcard partition during execution.

Resolve the **symbolic names**

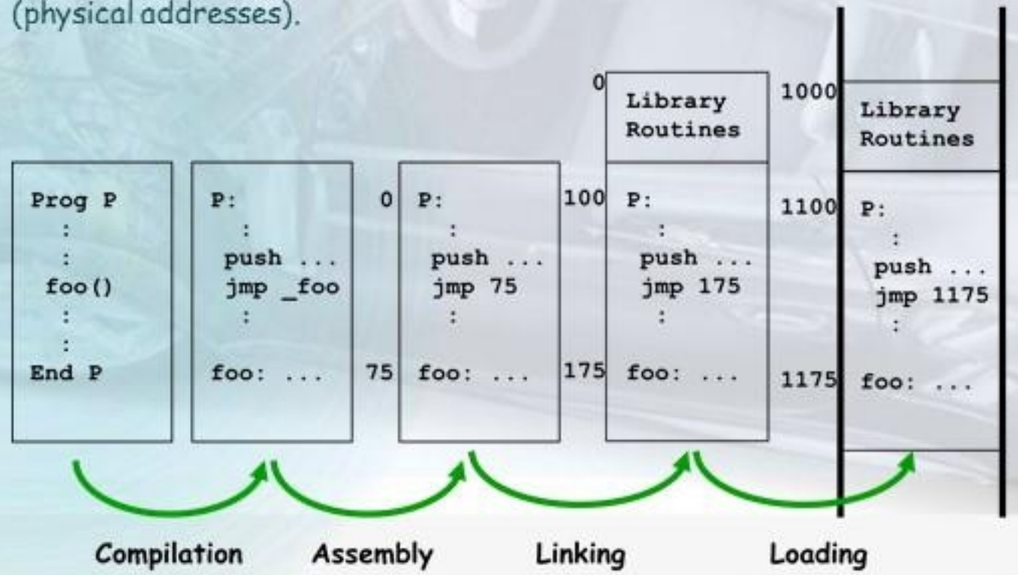
Each *symbolic name* should be converted *to memory address*

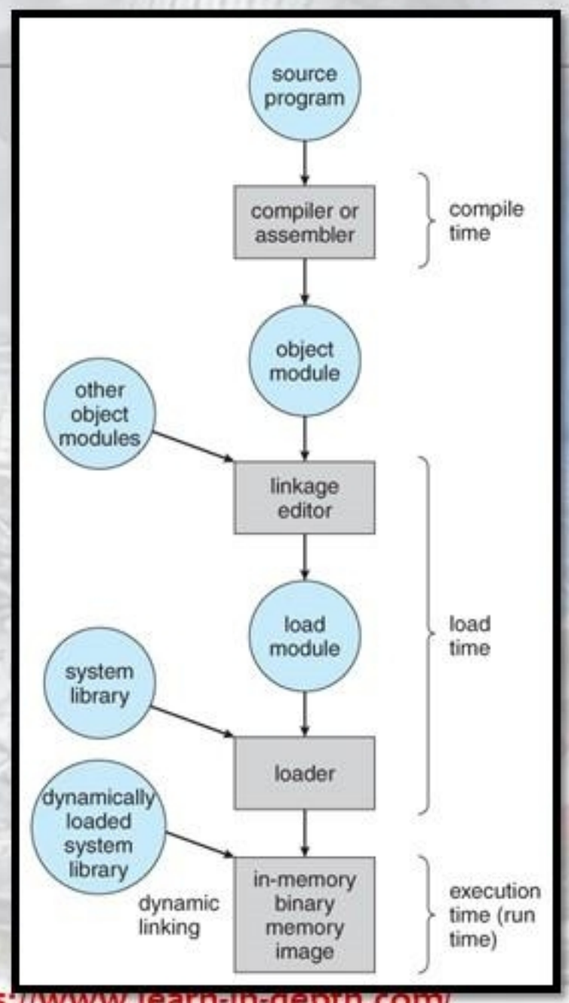
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
50
#LEARN IN DEPTH
#Be professional in embedded system

Types of Address

- Symbolic Addresses
  - Eg,variable names (such as variable i)
- Relocateable Addresses
  - A compiler typically binds symbolic addresses to relocatable addresses (interms of offsets).
- Absolute Addresses
  - The linkage editor or loader binds relocatable addresses to absolute addresses (physical addresses).





https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
51

#LEARN IN DEPTH

#Be professional in embedded system

Address Binding can happen at three different stages

Compile Time Binding

if program location is fixed and known ahead of time

Load Time Binding

if program location in memory is unknown until run-time AND location is fixed

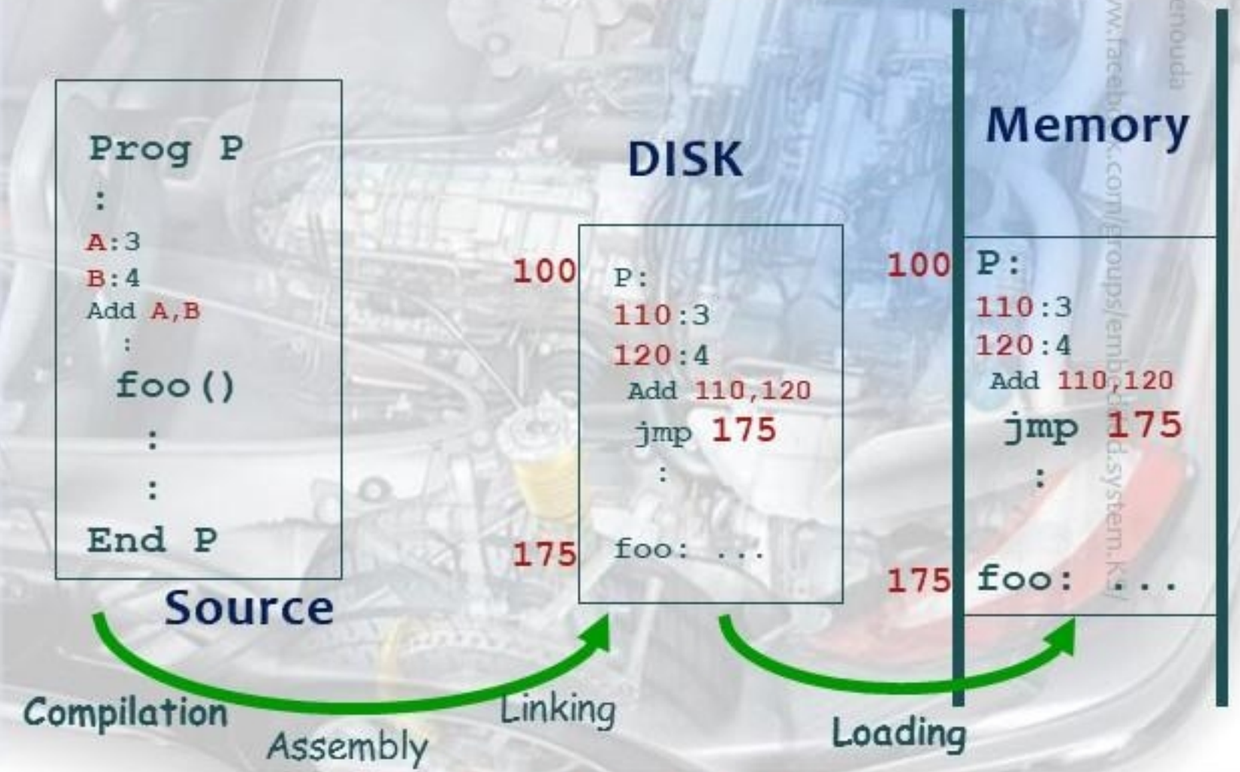
Execution Time Binding

if processes can be moved in memory during execution
Requires hardware support!

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

Compile Time Binding

- if memory **location** is **fixed** and **known** at compile time, absolute code with absolute address can be generated.
- Must Recompile Code if starting location changed.

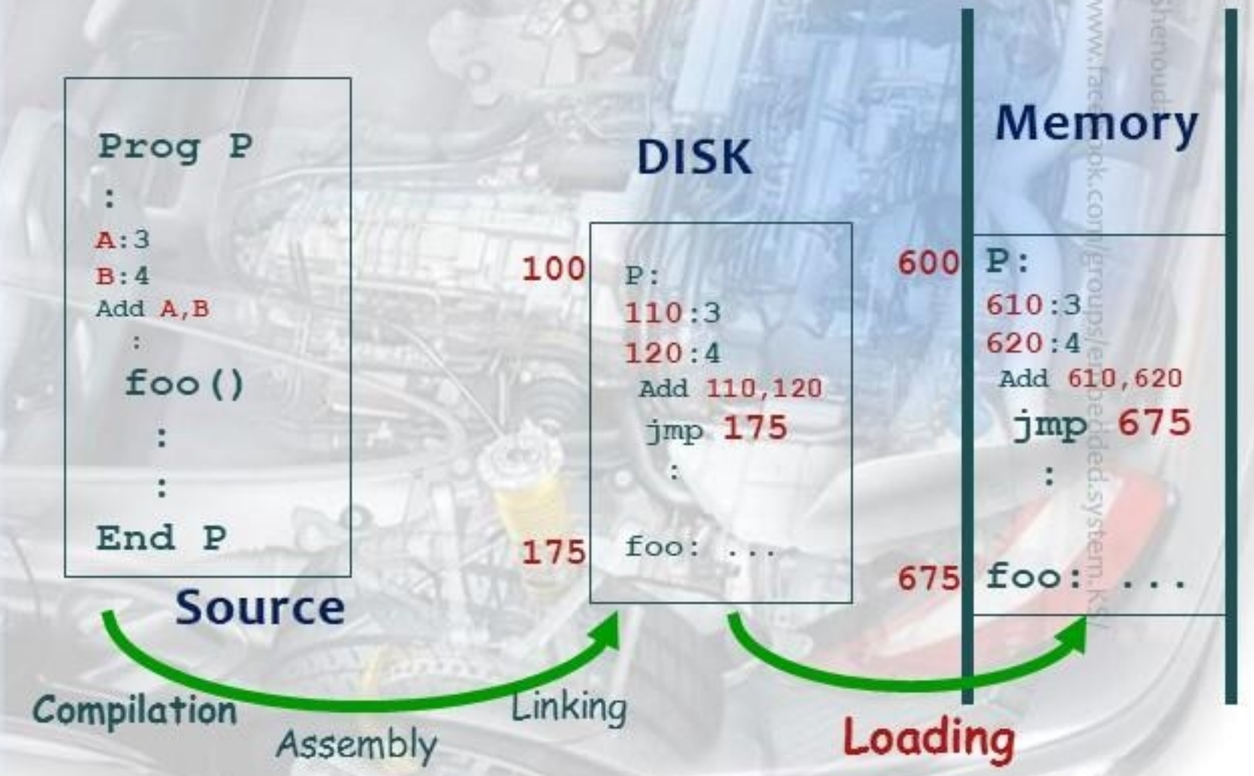


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
53
#LEARN IN DEPTH
#Be professional in embedded system

Load Time Binding

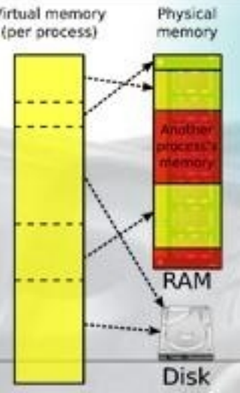
- if Memory location in memory is **unknown** at compile time AND location is fixed.
- The relocatable code with relative address can be generated.
- Binding is done when the program is **loaded** into the memory
- It is no need to recompile the program even **the starting location changes into the memory, as it needs only reload.**



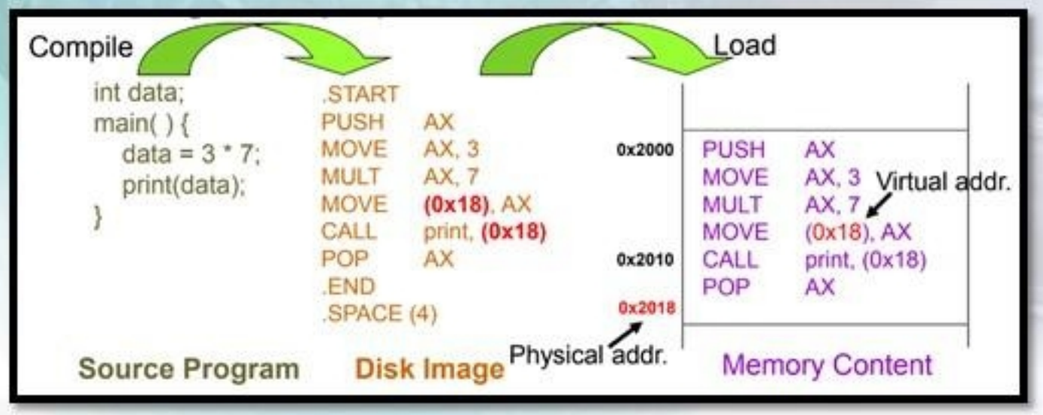
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

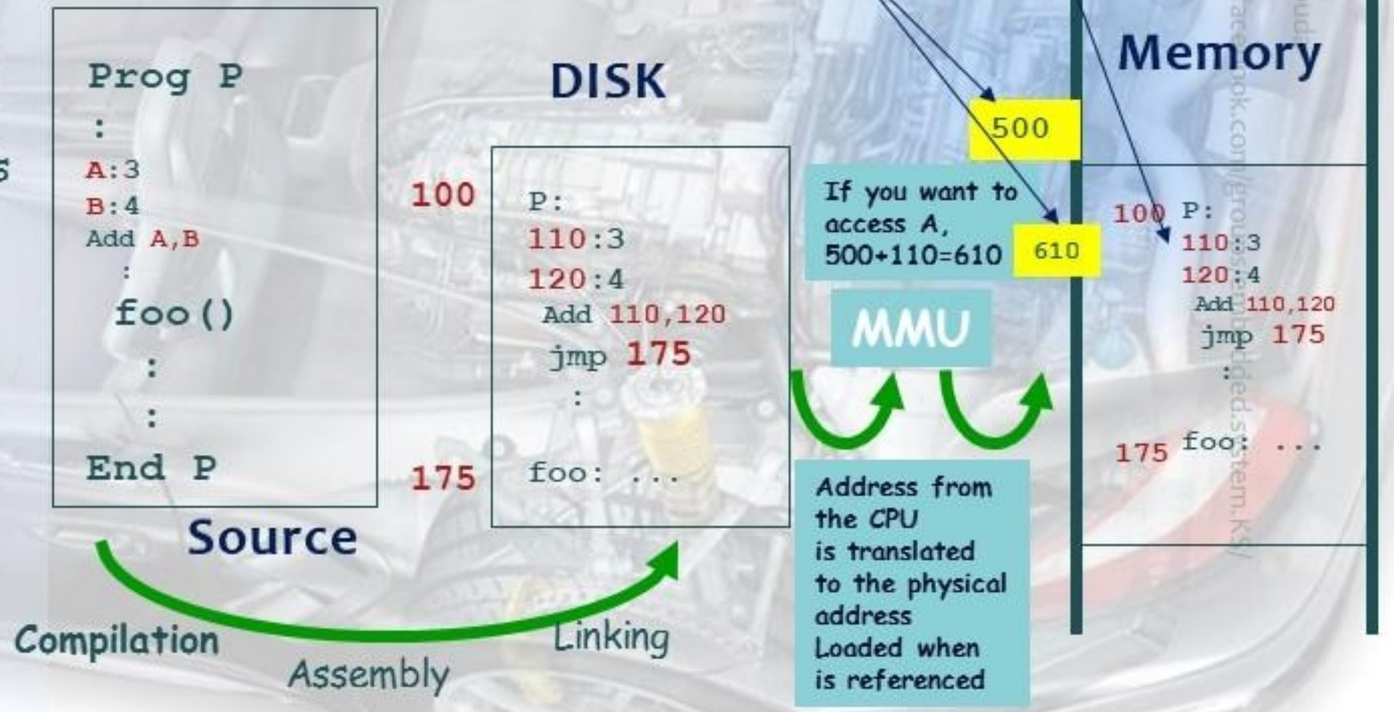
FOLLOW US
Press here
54
#LEARN IN DEPTH
#Be professional in embedded system

Execution Time Binding



- if processes can be moved in memory during execution
- Binding is delayed until **run time**
- Requires hardware support! (MMU) for address mapping
- Whenever CPU generates the address, binding is checked.
- Most general purpose OS use this method

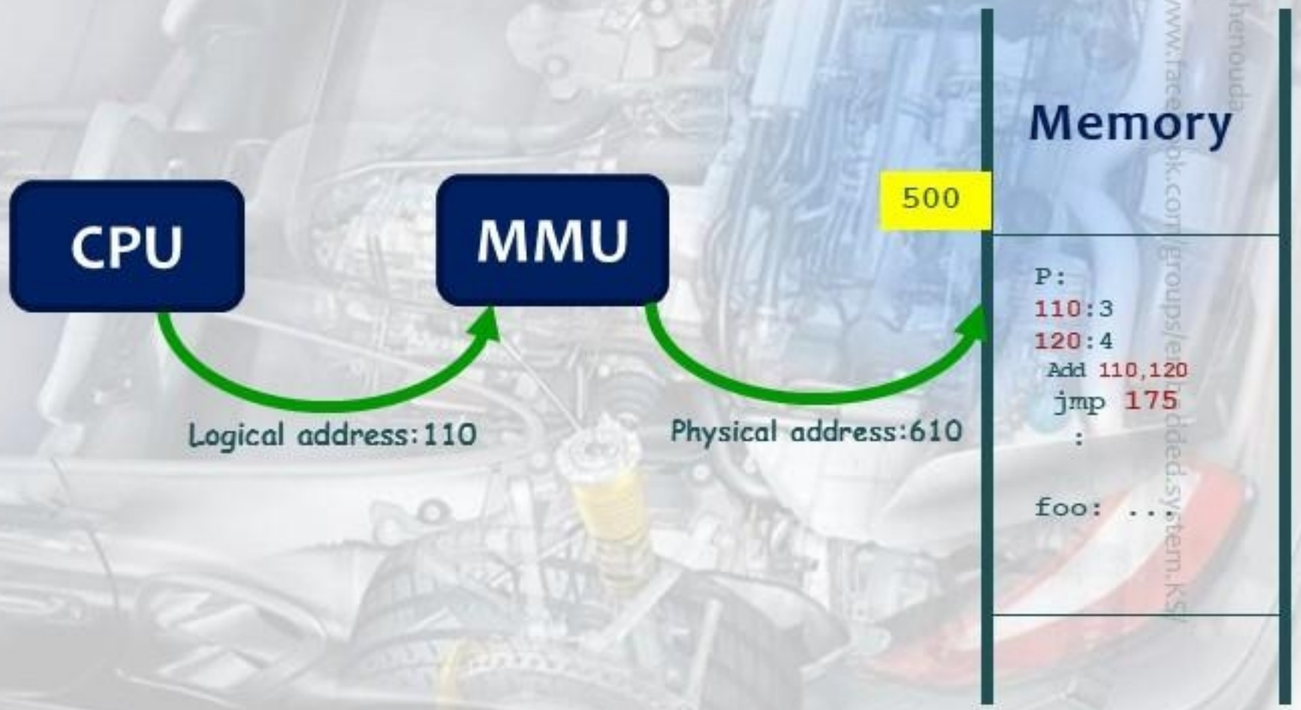




https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Memory Management Unit (MMU)

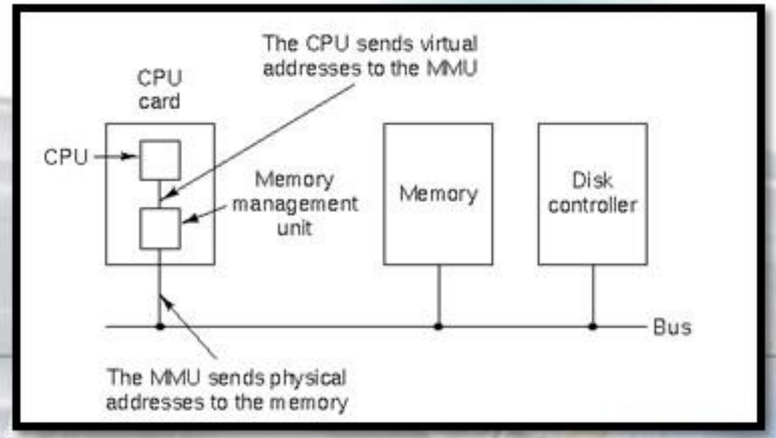
- Logical address-
  - Generated by the CPU
- Physical address-
  - Address seen by the memory unit
- CPU tries to access symbolic names via their logical addresses but uses physical address, thus logical addresses need to be changed to physical addresses by the MMU.



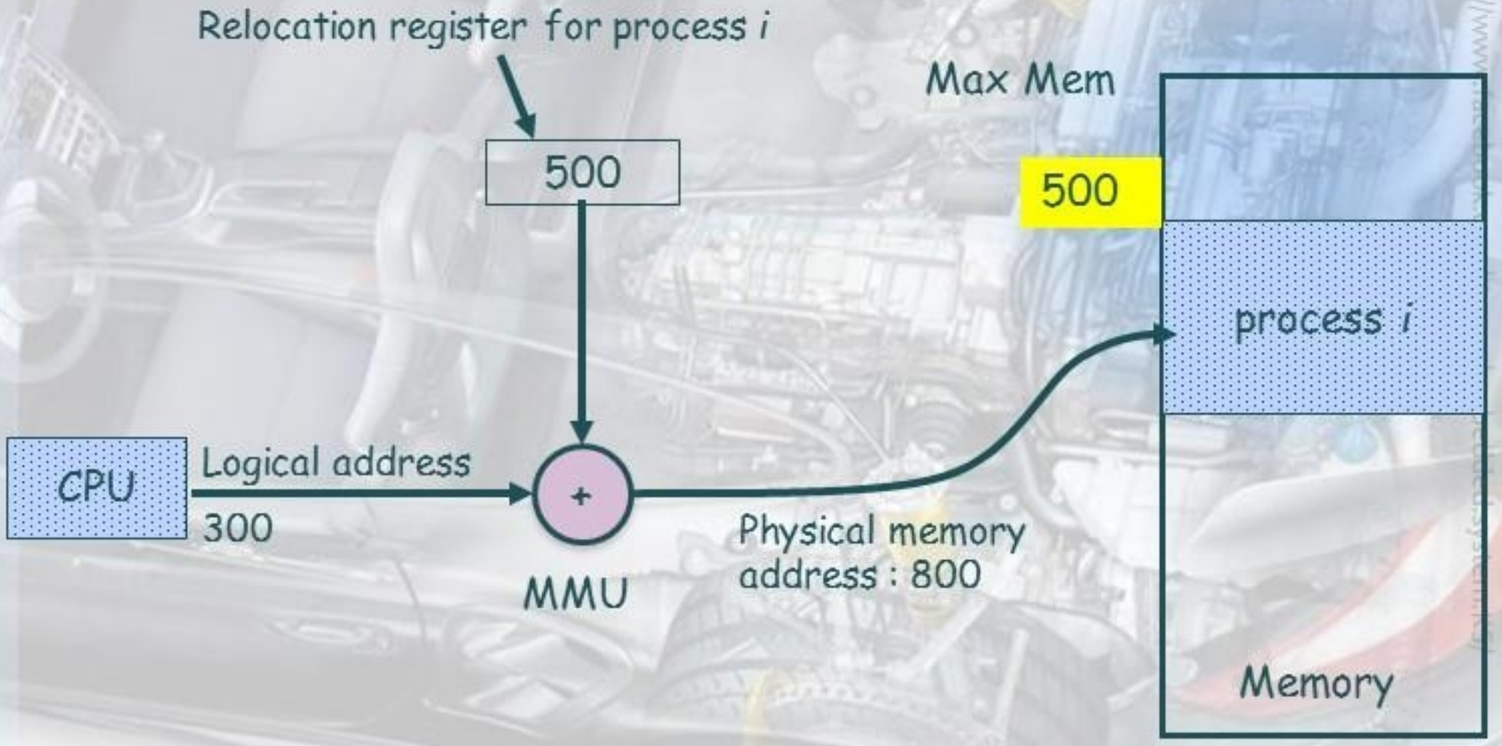
Eng.Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Memory Management Unit (MMU) Cont.



- MMU is the **Hardware device** that maps **virtual** to **physical** address.
- MMU contains ***base address register*** for running process
- Program & CPU
  - deal with logical address, but never sees physical address.

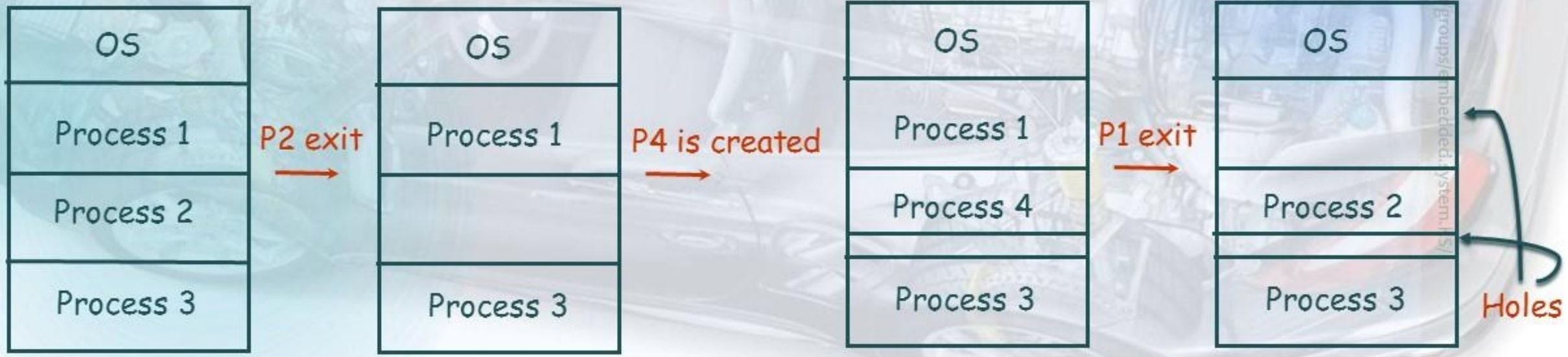


FOLLOW US
Press here
56
#LEARN IN DEPTH
#Be professional in embedded system
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
57
#LEARN IN DEPTH
#Be professional in embedded system

Memory Allocation

- Operating Systems occupies a certain part of memory till the system power is turned off.
- processes occupy or release memory space depending on creation and termination.
- Hole: is a large block of *available memory*.
- Holes of various size are **scattered** throughout memory.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
58

#LEARN IN DEPTH
#Be professional in embedded system

Memory Allocation Cont.

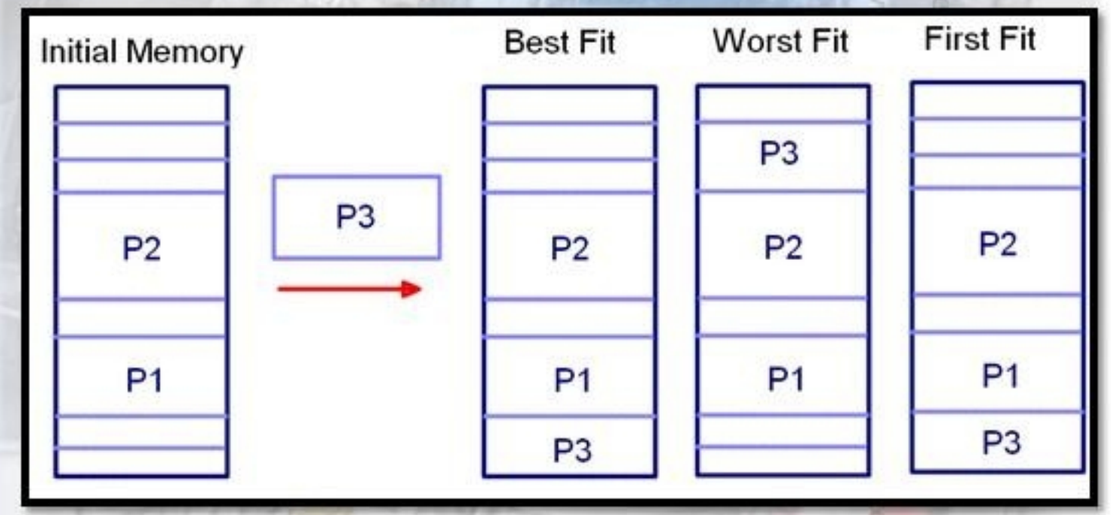
- When a process is created, it is allocated in a large enough hole.
- Operating system should continuously maintains information about
  - Allocated partitions
  - Free partitions (hole)

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

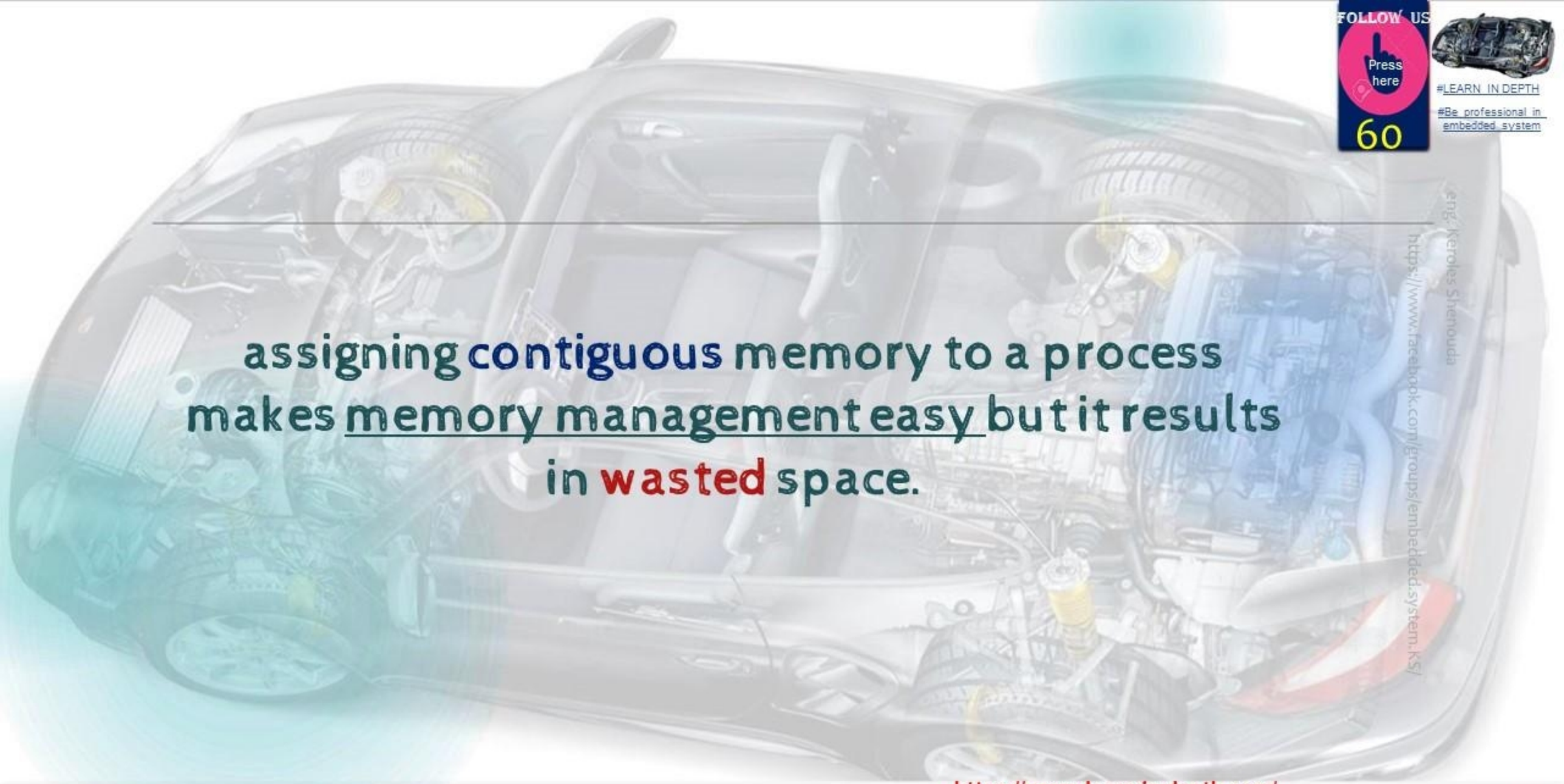
FOLLOW US
Press here
59
#LEARN IN DEPTH
#Be professional in embedded system

Dynamic Allocation problem

- How to satisfy a request of size *n* from a list of free holes.
- There are three algorithms:
  - **First-fit**: Allocate the *first* hole that is big enough.
  - **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
  - **Worst-fit**: Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole.
- First-fit and best-fit better than worst-fit in terms of speed and storage utilization.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

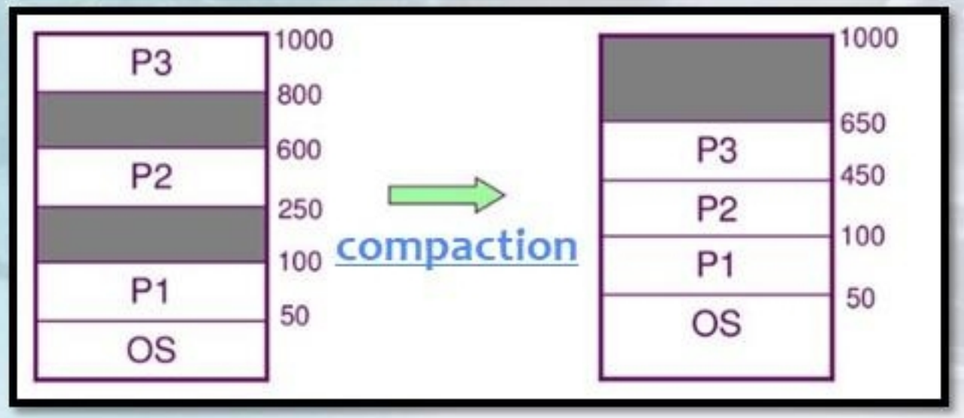


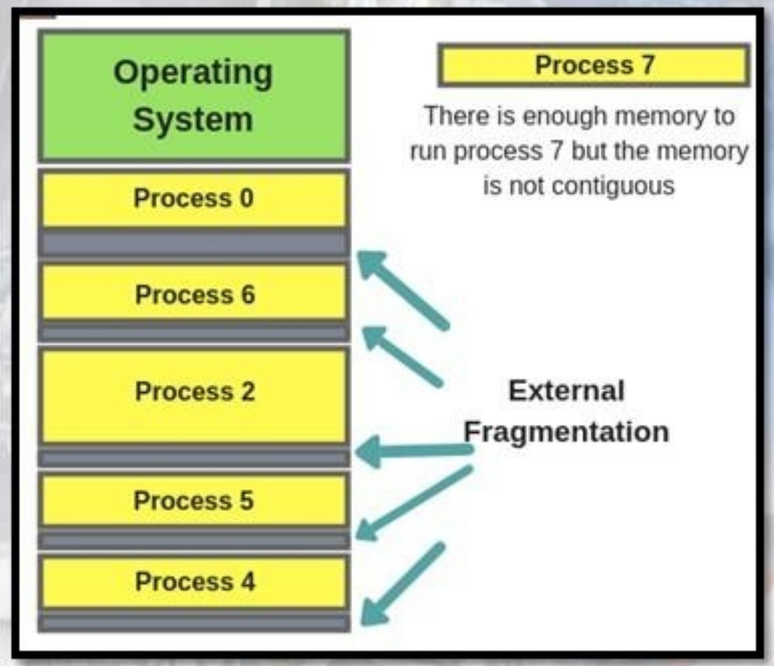
assigning contiguous memory to a process makes memory management easy but it results in wasted space.

FOLLOW US
Press here
61
#LEARN IN DEPTH
#Be professional in embedded system

Fragmentation

- External Fragmentation
  - Total free memory space exists to satisfy a request, **but it is not contiguous**
  - Can be reduced by **compaction:**
    - Places all free memory together in one large block.



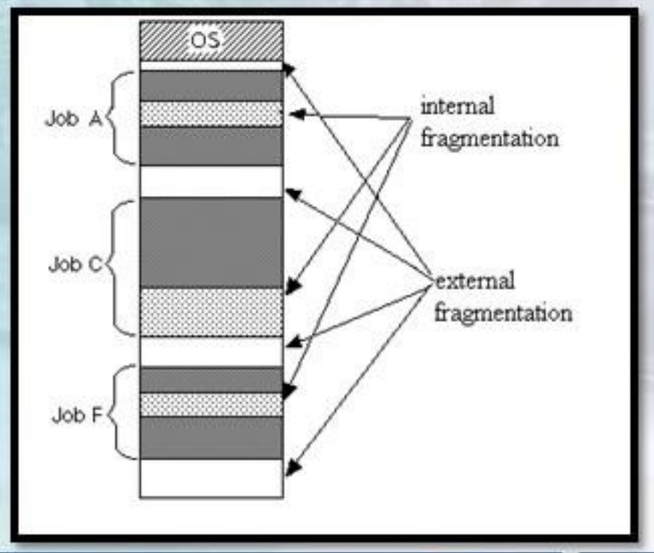


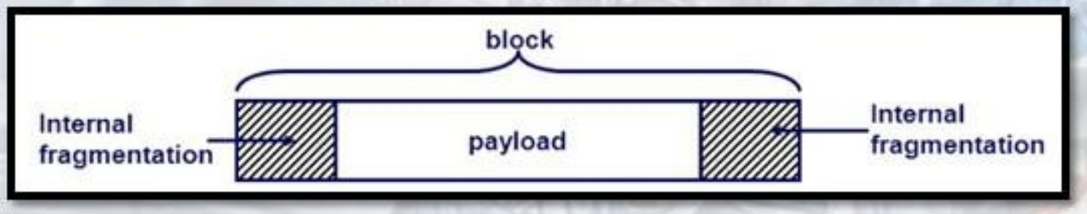
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
62
#LEARN IN DEPTH
#Be professional in embedded system

Fragmentation Cont.

- Internal Fragmentation
  - The allocated memory may be slightly larger than requested memory.
  - For some block, internal fragmentation is difference between the block size and the payload size.
  - Size partition is internal to a partition, but is not used.





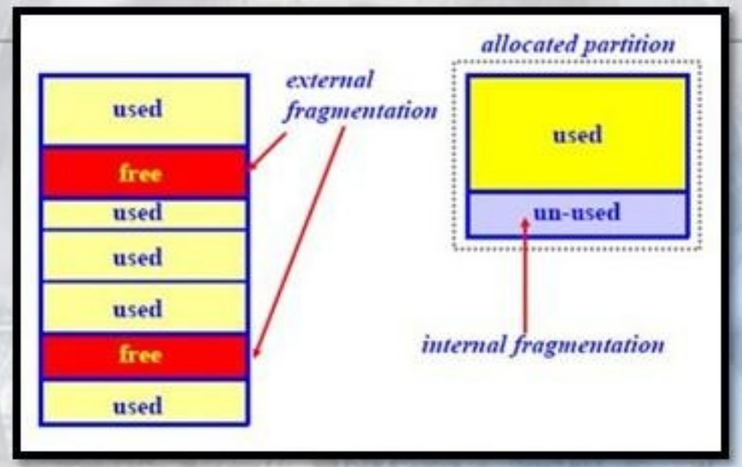
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

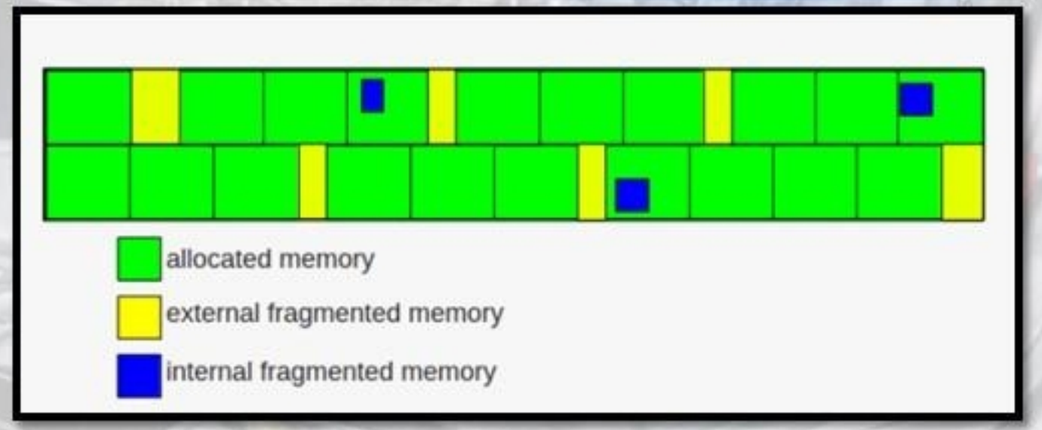
FOLLOW US
Press here
63
#LEARN IN DEPTH
#Be professional in embedded system

Internal Vs External Fragmentation

INTERNAL FRAGMENTATION VERSUS EXTERNAL FRAGMENTATION	
INTERNAL FRAGMENTATION	EXTERNAL FRAGMENTATION
A form of fragmentation that arises when there are sections of memory remaining because of allocating large blocks of memory for a process than required	A form of fragmentation that arises when there is enough memory available to allocate for the process but that available memory is not contiguous
Memory block assigned to a process is large - the remaining portion is left unused as it cannot be assigned to another process	Memory space is enough to reside a process, but it is not contiguous. Therefore, that space cannot be used for allocation
Solution is to assign partitions which are large enough for the processes	Compaction or shuffle memory content is the solution to overcome this

Visit www.PEDIAA.com





https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

eng.Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

Summary ☺

FOLLOW US
Press here
65

#LEARN IN DEPTH
#Be professional in embedded system

Address Binding

Address binding is **the process of mapping** the program's logical or virtual addresses to corresponding physical or main memory addresses

Compile Time Binding

if memory **location** is **fixed** and **known** at compile time, absolute code with absolute address can be generated.

Load Time Binding

if Memory location in memory is **unknown** at compile time AND location is fixed.
The relocatable code with relative address can be generated.
Binding is done when the program is **loaded** into the memory

Execution Time Binding

if processes can be moved in memory during execution Binding is delayed until **run time**

Requires hardware support! (**MMU**) for address mapping

MMU is the **Hardware device** that maps **virtual** to **physical** address.

Dynamic Allocation

There are three algorithms:
First-fit, **Best-fit** and **Worst-fit**

Problem

Internal Vs External Fragmentation

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

Learn-in-depth.com

FOLLOW US
Press here
2

#LEARN_IN DEPTH
#Be_professional_in embedded_system

References

- Embedded Linux training
  - https://bootlin.com/training/
- Linux kernel and driver development training
- Yocto Project and OpenEmbedded development training
- Buildroot development training
- Building Embedded Linux Systems, 2nd Edition
- Mastering Embedded Linux Programming - Second Edition
- Christopher Hallinan, **Embedded Linux Primer,** Prentice Hall, 2006.
- **Silberschatz**, Galvin, and Gagne's **Operating System Concepts**, **Eighth Edition**.
- Robert love, linux kernel develoepemnt 3 rd edition 2010
- **Advanced Linux Programming By Mark Mitchell, Jeffrey Oldham, Alex Samuel**
- **Linux OS in Embedded Systems & Linux Kernel Internals(2/2)**
  - Memory Management, Paging, Virtual Memory, File system and its implementation, Secondary Storage(HDD), I/O systems

eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

- Memory Management article
- Introduction to Operating Systems Class 9 - Memory Management
- Virtual Memory lecture for Introduction to Computer Architecture at Uppsala University.
- OS Lecture Note 13 - Address translation, Caches and TLBs
- CSE 331 Operating Systems Design lectures
- CSE 331 Virtual Memory
- CSE 332 Computer Organization and Architecture
- File Systems: Fundamentals.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

- **File Permissions in Linux/Unix with Example**
- Introduction to UNIX / Linux – 4
- **12.2 Basic I/O Concepts**
- Communicating with Hardware
- I/O Systems I/O Hardware Application I/O Interface
- COMPUTER ARCHITECTURE
- **OS**
- Linux Operating System
- W4118 Operating Systems, Instructor: Junfeng Yang
- CSNB334 Advanced Operating Systems 4. Process & Resource Management.
- Using the POSIX API
- **Linux Memory Management**
- Chapter 2: Operating-System Structures
- **POSIX Threads Programming**
- Pthreads: A shared memory programming model https://slideplayer.com/slide/8734550/