Embedded System Technologies for Deep Learning and Approximate Computing

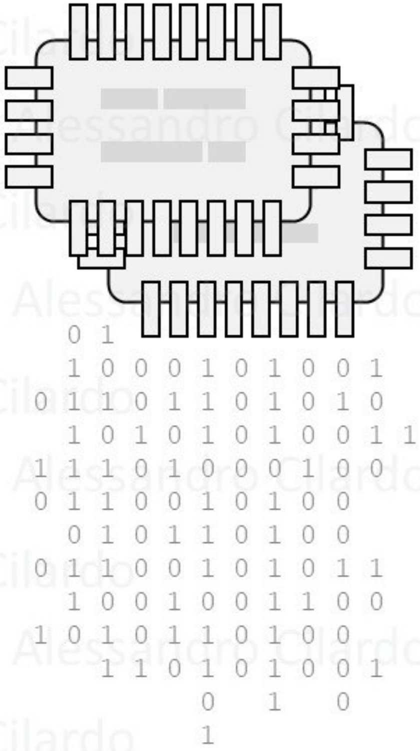
Under the SPARC Project P:271 – "Approximate Computing Techniques for Resource Constrained Edge Devices"

Prof. Alessandro Cilardo

acilardo@unina.it

Workshop overview

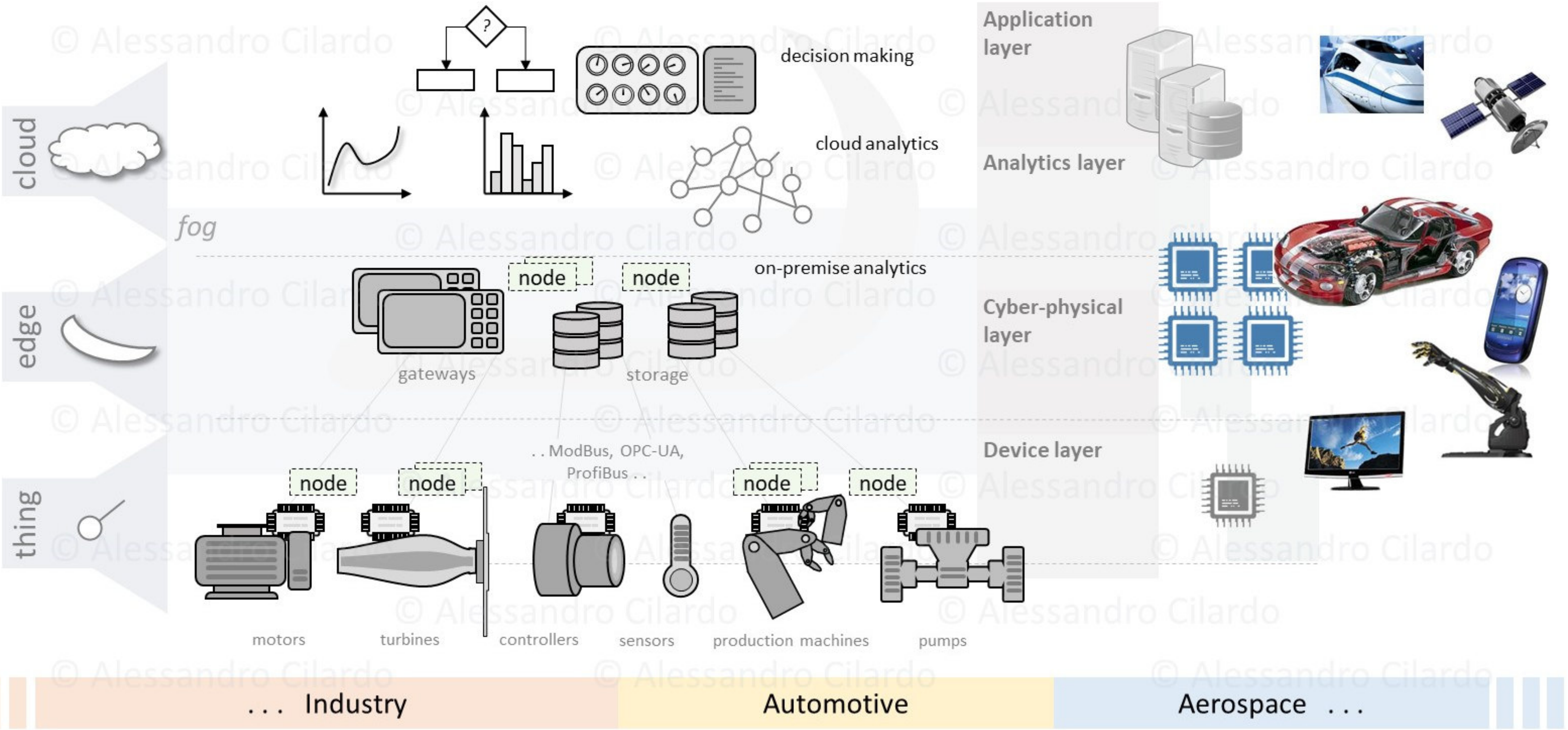
- Microcontrollers and Application processors
- ARM ecosystem: processor families and evolution of the ARM architecture
- Introduction to ARM Cortex-M architecture
- System-on-Chip technologies based on ARM Cortex-M
- A case-study: STM32 SoC devices
- Software development and debug tools, GNU toolchain
- ARM CMSIS framework
- CMSIS Neural Network library (CMSIS-NN)
- Real-Time Operating Systems
- A case-study: FreeRTOS
- Introduction to ARM Cortex-A architecture
- ARM Cortex-A software stack
- Hardware-customizable FPGA-based System-on-Chip technologies
- Case-study: Xilinx Zynq-7000 SoC architecture
- An overview of Xilinx FPGA design flows
- FPGA-based customized hardware acceleration and opportunities for Approximate Computing



Day 1
May 29th, 2021

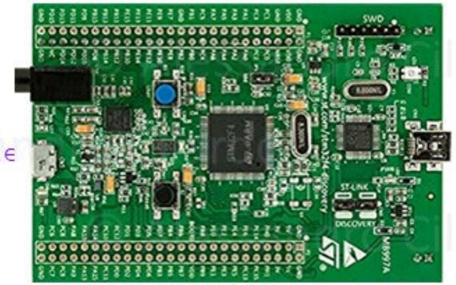
Microcontroller-based systems

Embedded systems are *everywhere* . . .

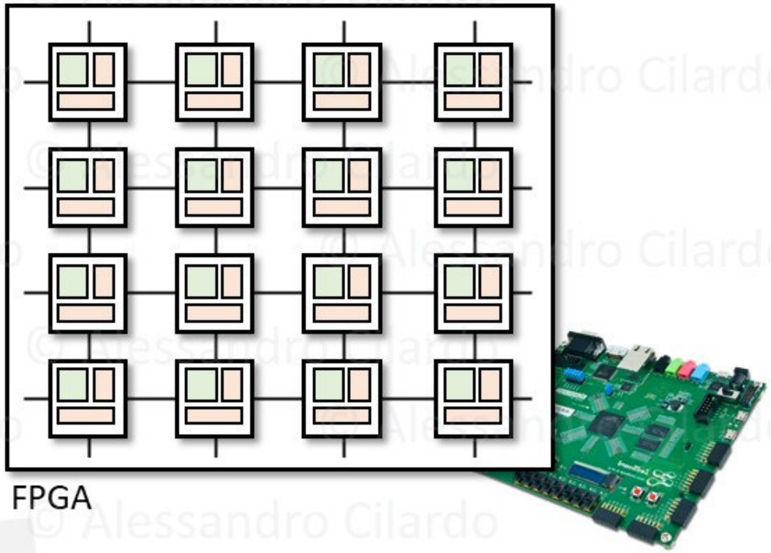


Embedded systems and embedded development flows

```
inline static double computeNeighbor(double
    if (v==0) return 0.0;
    int sgn=false;
    if (v<0) { v*=-1; sgn=true; }
    int e;
    . . .
```

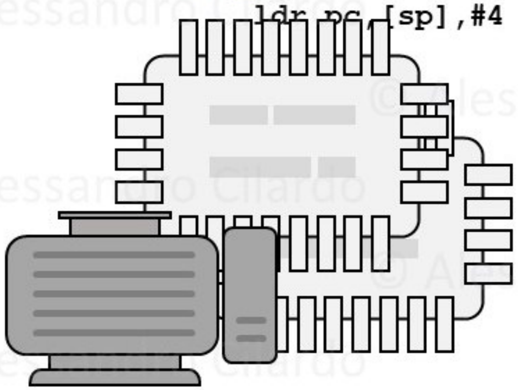


System-on-Chip

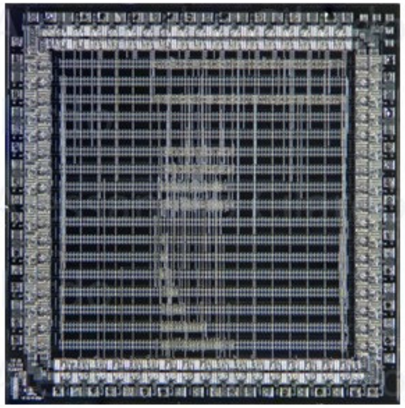


FPGA

```
lbl_1:
    ldr r0,[r6]
    ldr r14,=__Nil+2
    cmp r0,r14
    bne case_P2
lbl_4:
    ldr pc,[sp],#4
```



```
. . .
signal stop : std_logic;
begin
controller_inst: controller
    port map(
    ClkxCI => ClkxCI,
    ResetxRBI => ResetxRBI,
    . . .
```

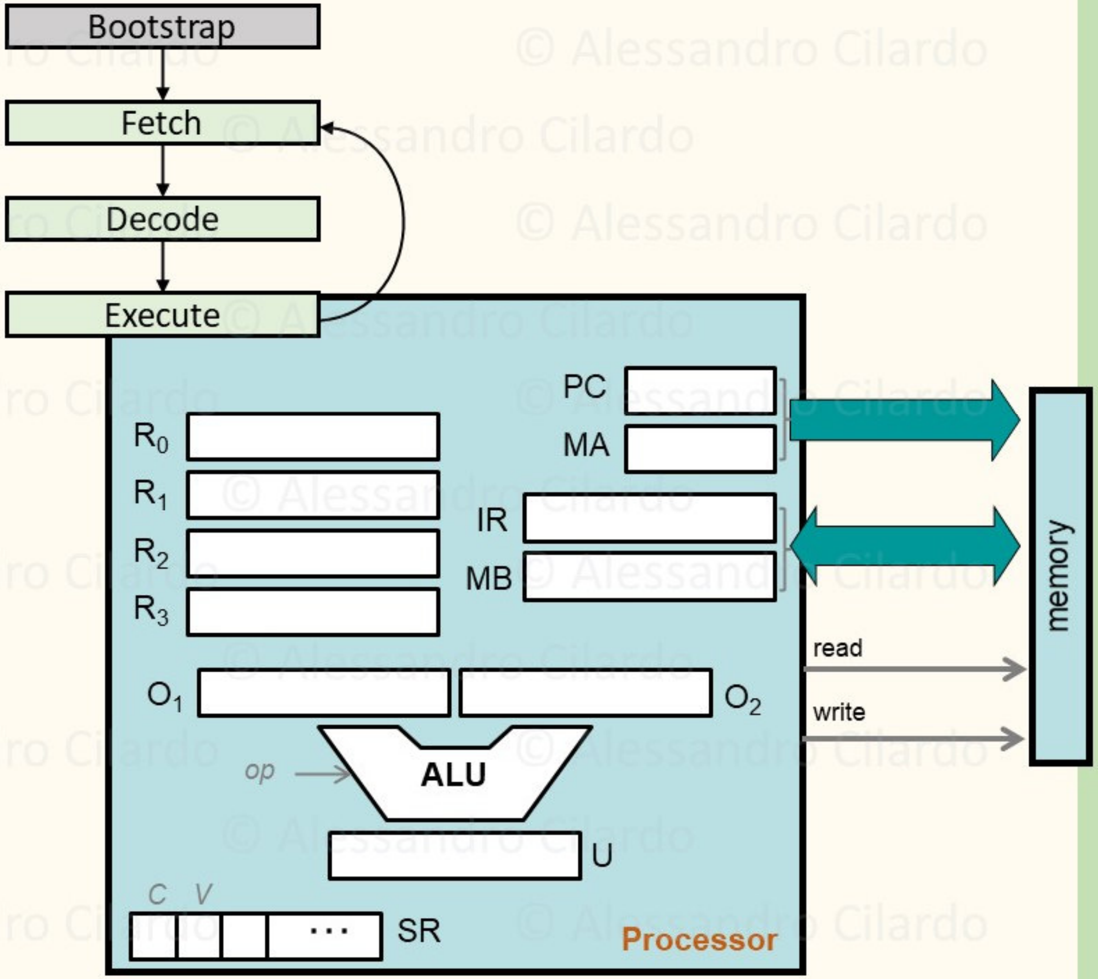


ASIC

Quick Review of Computer Architecture concepts (≈15 min)

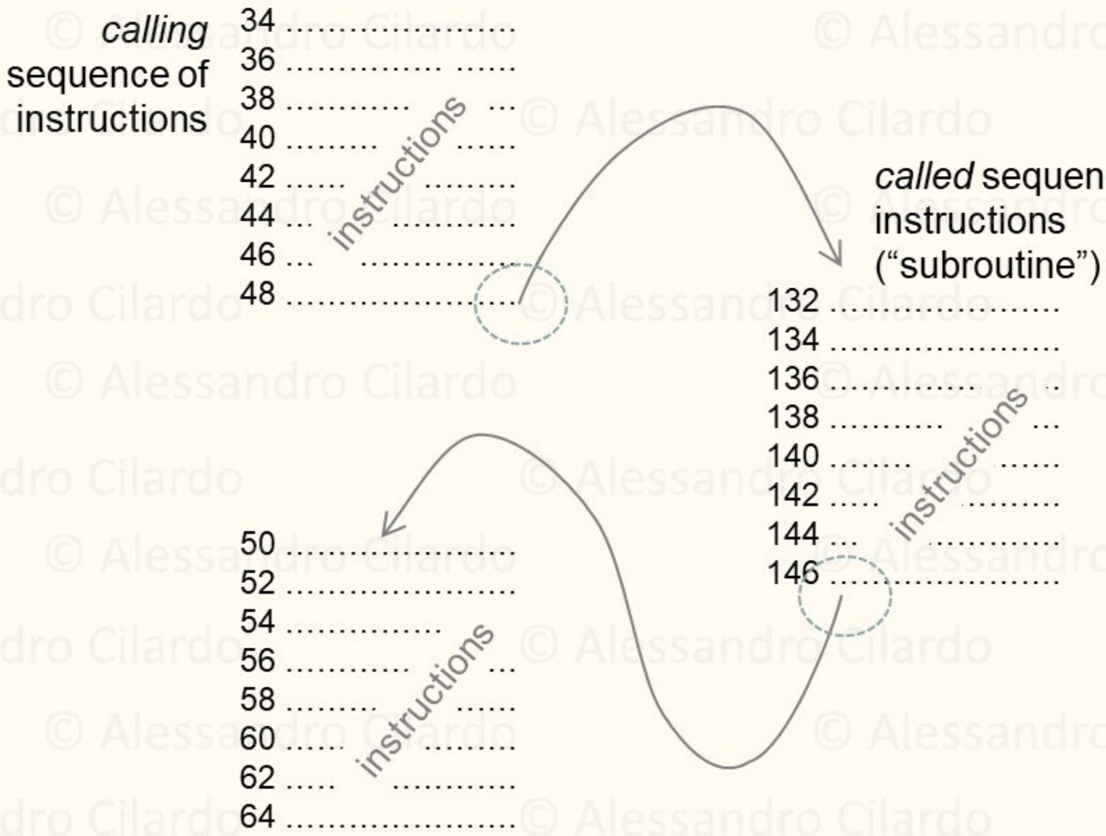
Quick Review of Computer Architecture concepts (≈15 min)

- Processor organization
  - General-purpose registers
  - Special registers: Program Counter, Status registers, ..
  - Arithmetic-Logic Unit (ALU)
  - Instruction Set Architecture (ISA)
  - RISC paradigm vs. CISC paradigm
  - Types of instructions:
    - data processing
    - data movement
    - control flow
    - special instructions



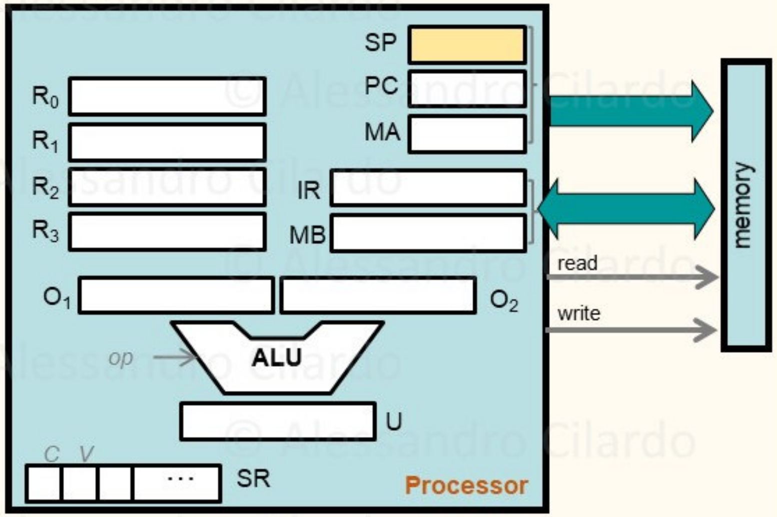
Quick Review of Computer Architecture concepts (≈15 min)

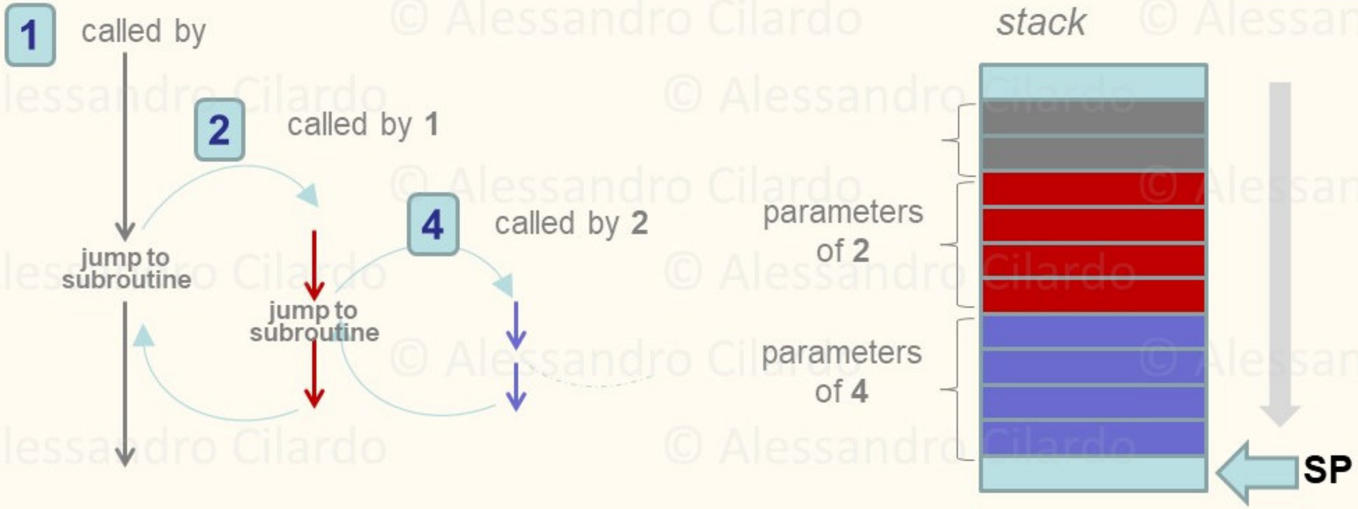
- Subroutines
  - Linkage
  - Parameter passage
  - Role of the stack



Quick Review of Computer Architecture concepts (≈15 min)

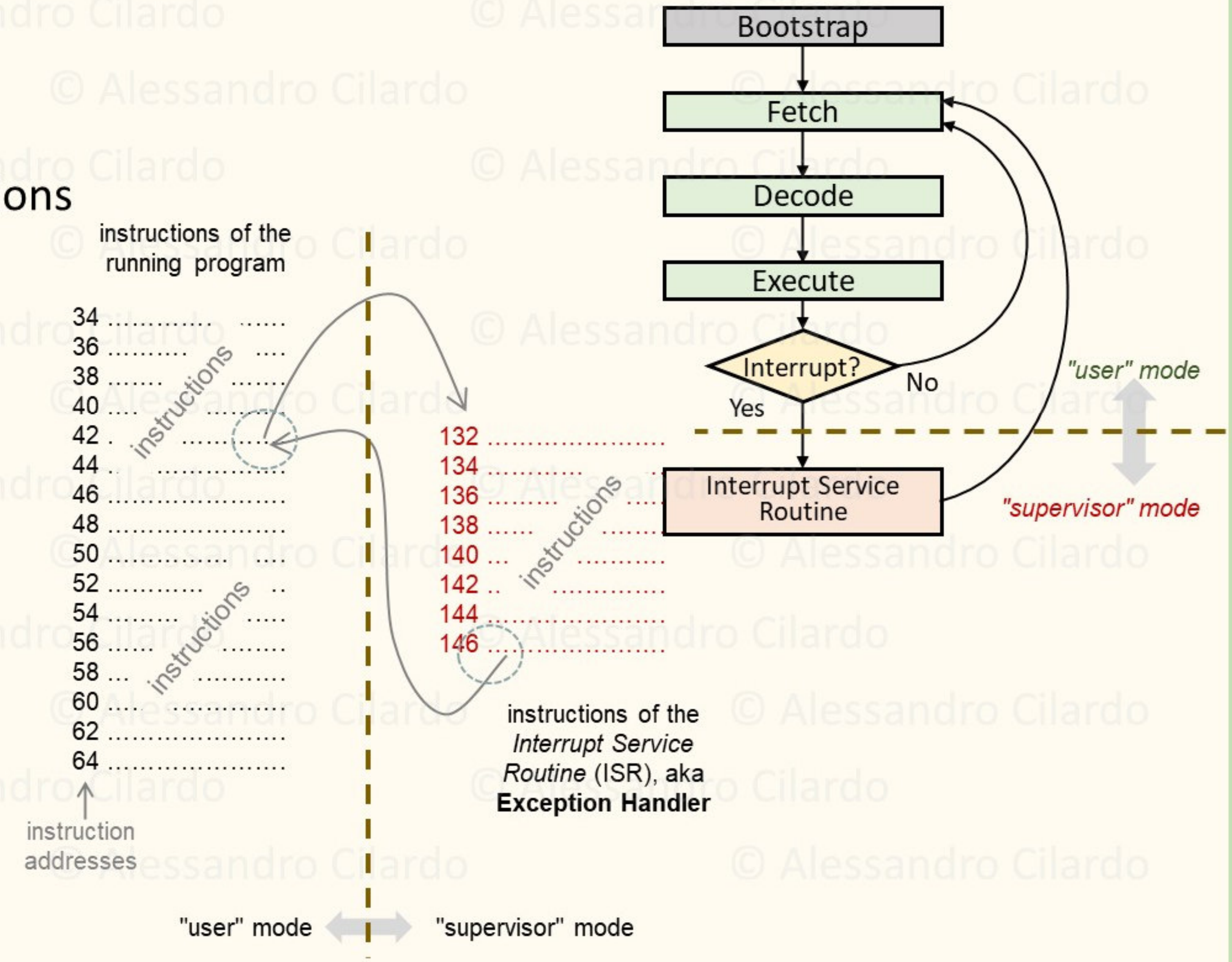
- Subroutines
  - Linkage
  - Parameter passage
  - Role of the stack





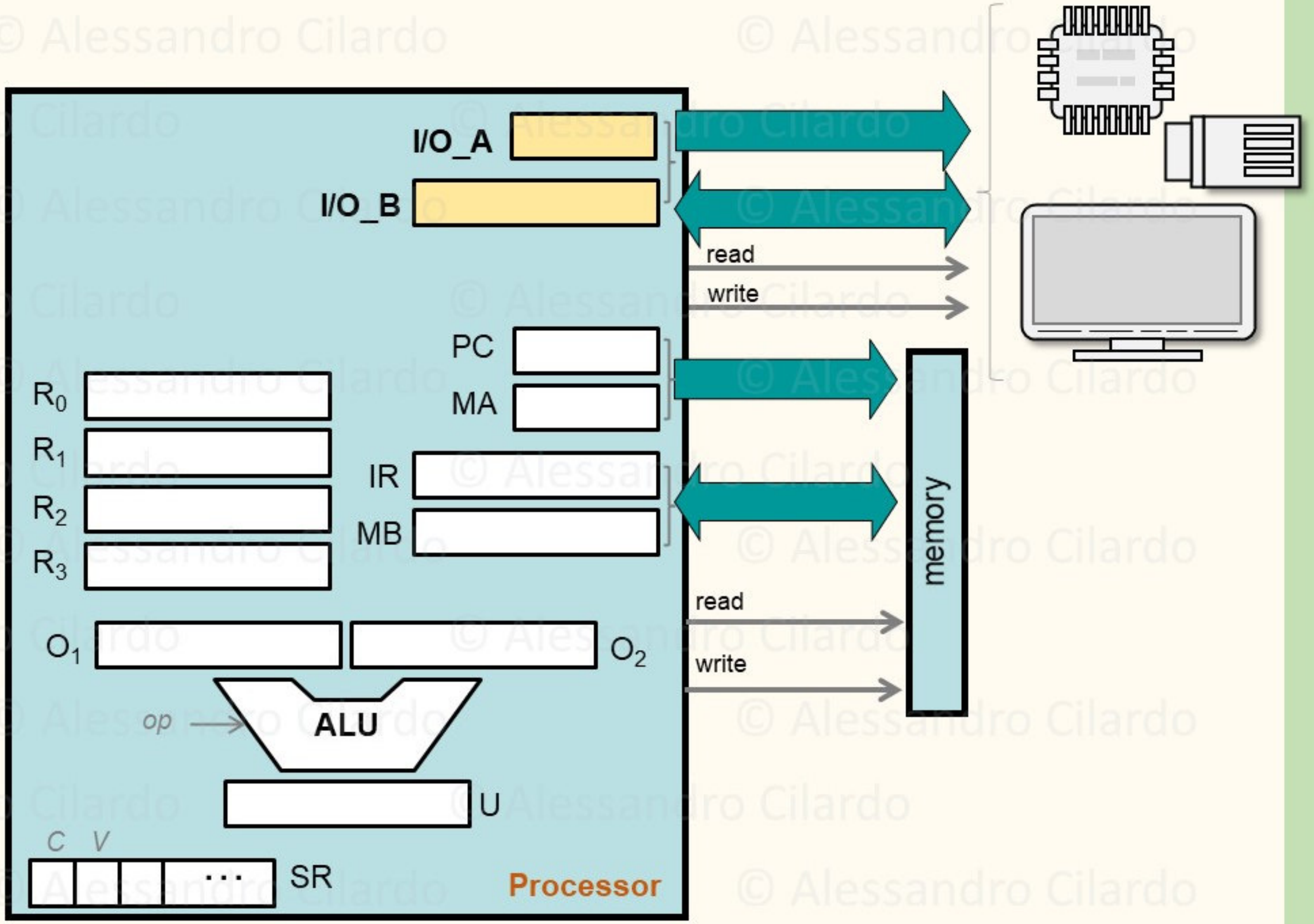
Quick Review of Computer Architecture concepts (≈15 min)

- Interrupts/exceptions
  - basic mechanism
  - types of interrupts/exceptions
  - interrupt priority
  - nested interrupts
- execution modes and privilege levels



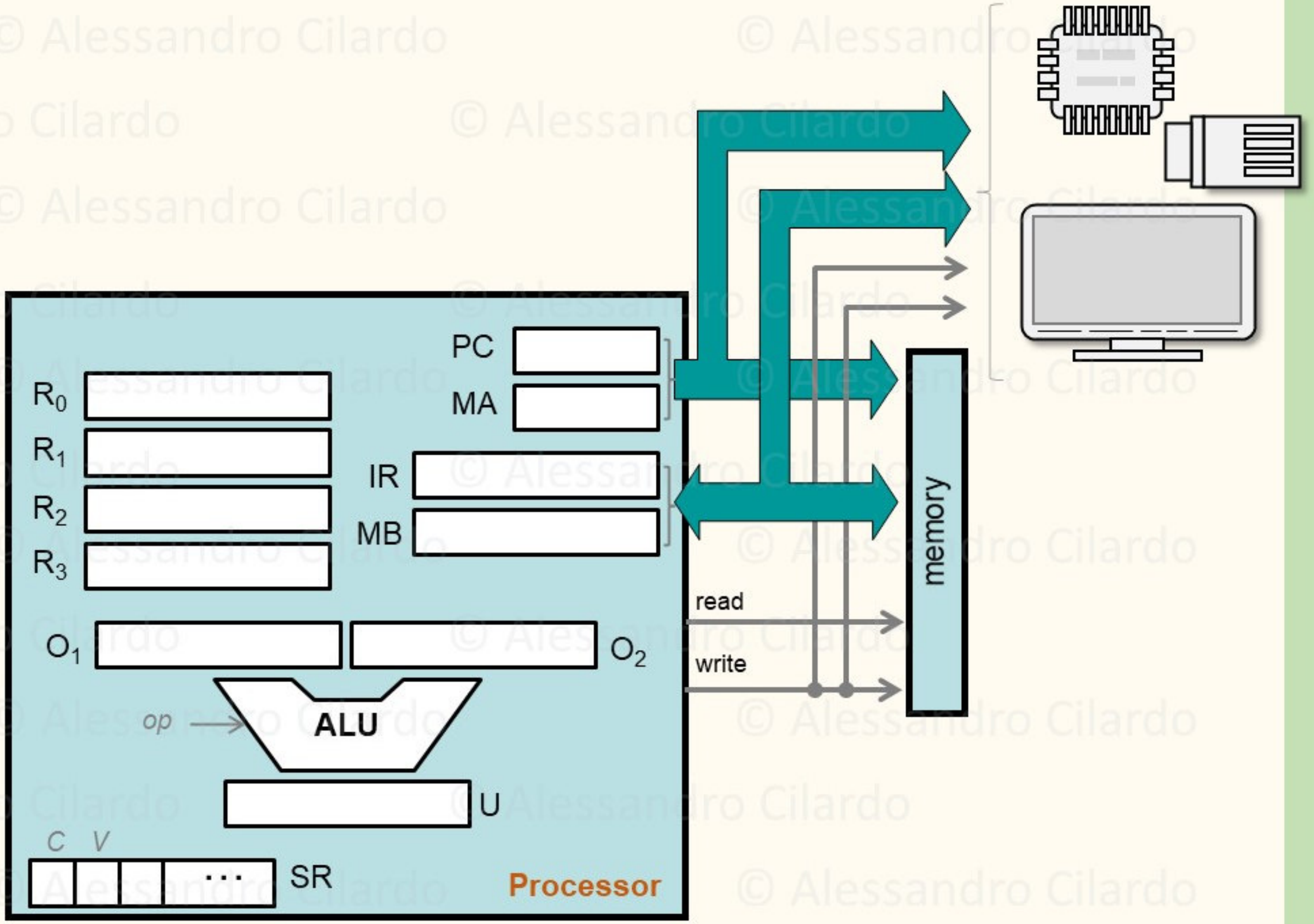
Quick Review of Computer Architecture concepts (≈15 min)

- Memory subsystem and I/O peripherals
- "***I/O***-mapped" approach



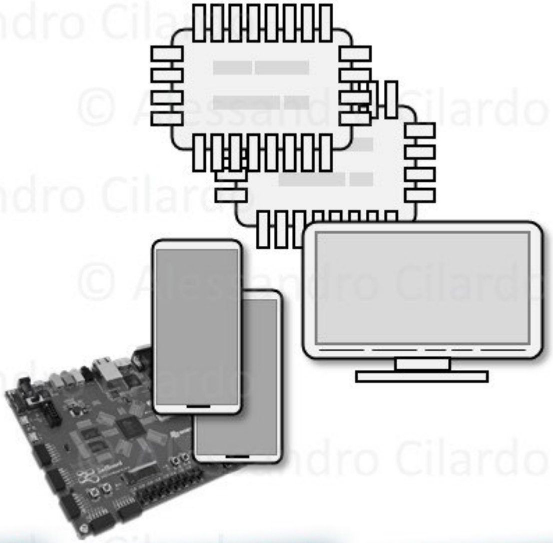
Quick Review of Computer Architecture concepts (≈15 min)

- Memory subsystem and I/O peripherals
- "***memory***-mapped" approach
- The bus to the I/O subsystem is typically hierarchical



ARM history and evolution of the ARM ecosystem

- Originally developed in 1982-1985 by Acorn Ltd
  - initially known as **A**corn **R**ISC **M**achine
- ARM (**A**dvanced **R**ISC **M**achine) founded in 1992
  - by Acorn, Apple, VLSI technologies
- Business model based on an IP licensing, *not* on manufacturing
- Many, many integrators today rely on Arm Intellectual Properties (IPs)



- For example:

  Cortex-M3/M4. Microcontroller vendors (as of 2014):
  - Analog Devices, Atmel, Cypress, EnergyMicro, Freescale, Fujitsu, Holtek, Infineon, Microsemi, Milandr, NXP, Samsung, SiliconLaboratories, ST Microelectronics, Texas Instrument, Toshiba

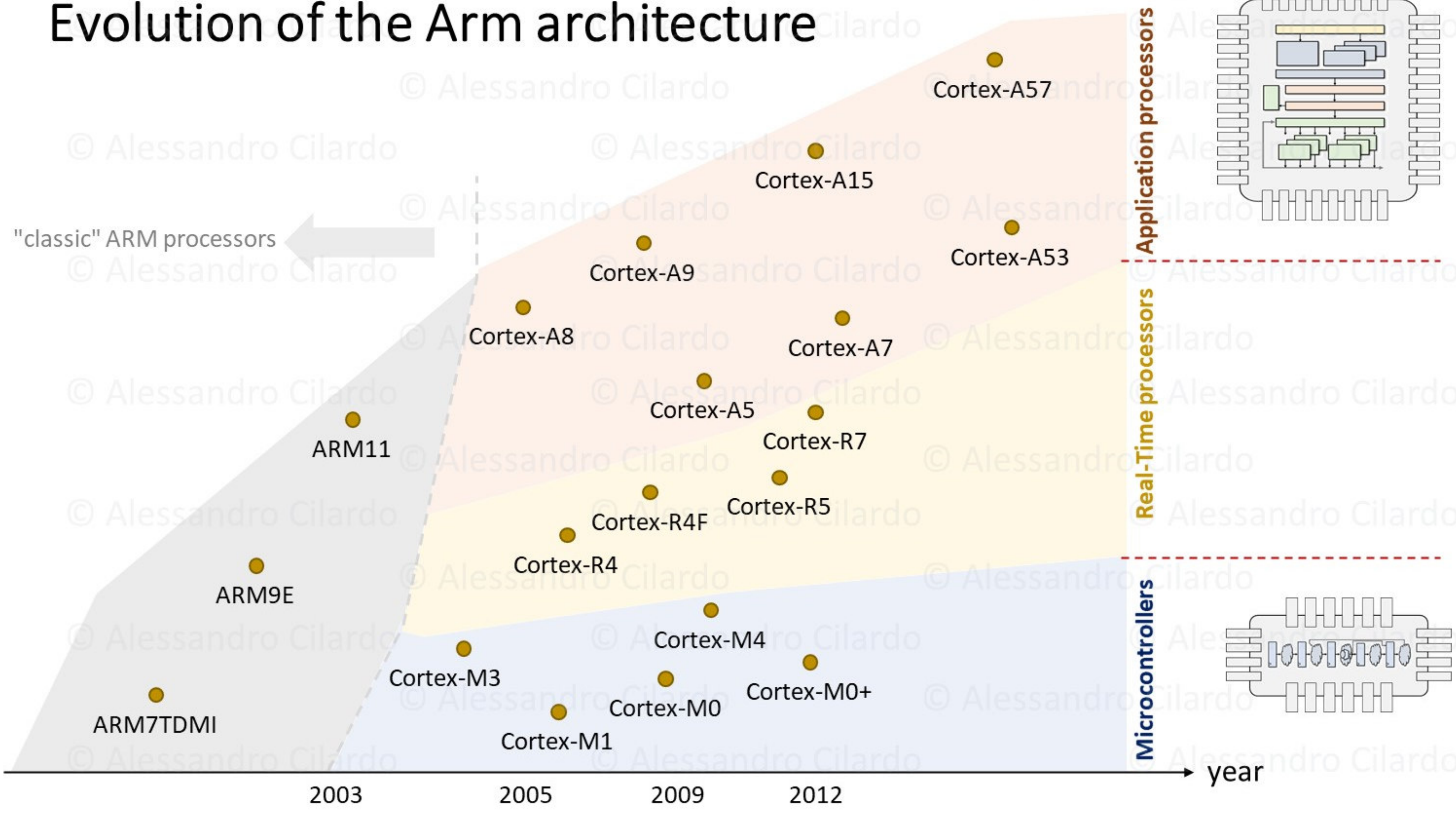
  Arm Cortex-A9. Some real products using it:
  - Apple A5 (iPhone 4S, iPad 2, iPad mini)
  - NVIDIA Tegra 2 (Motorola Xoom, Droid X2)
  - PlayStation Vita
  - Intel FPGA SoC
  - Xilinx Zynq
  - . . .

copyright © Arm Limited

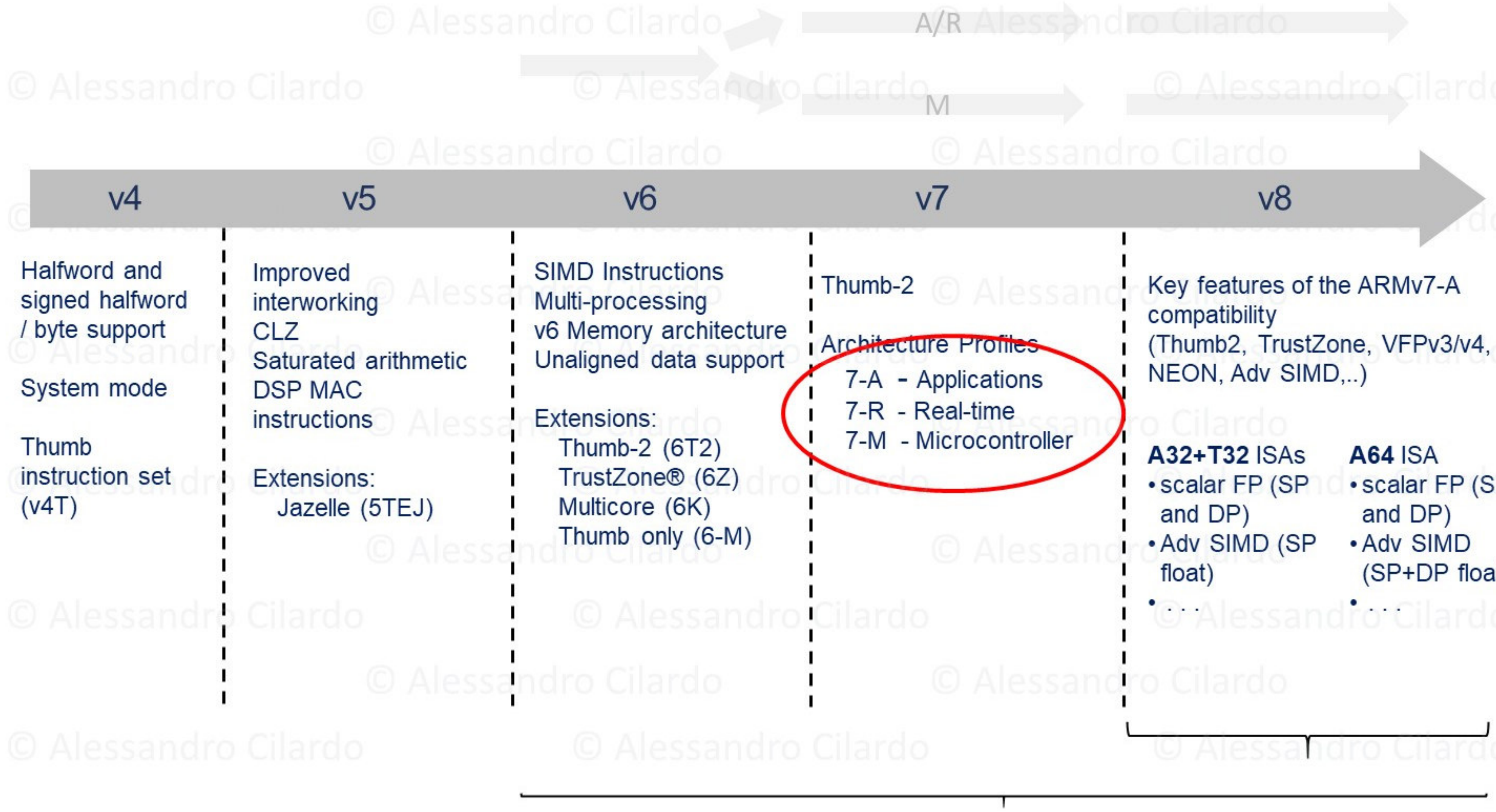
The ARM processor architecture

- Driving design principle: architectural simplicity
  - leads to very resource-efficient implementations
  - ..and hence, very low power consumption
  - implementation size, performance, and low power consumption are known to be key benefits of the ARM architecture
- ARM is a "RISC" architecture
  - uniform register file
  - load/store architecture
  - simple addressing
  - . . .
- An example of implementation: ARM Cortex-A9
  - high performance choice in a family of low power, cost-sensitive devices
  - Cortex-A9 microarchitecture is delivered either as
    - a single core processor
    - or a scalable multicore processor: the Cortex-A9 MPCore™ processor

Evolution of the Arm architecture



Evolution of the Arm architecture



v4

Halfword and signed halfword / byte support

System mode

Thumb instruction set (v4T)

v5

Improved interworking
CLZ
Saturated arithmetic
DSP MAC instructions

Extensions:
Jazelle (5TEJ)

v6

SIMD Instructions
Multi-processing
v6 Memory architecture
Unaligned data support

Extensions:
Thumb-2 (6T2)
TrustZone® (6Z)
Multicore (6K)
Thumb only (6-M)

v7

Thumb-2

Architecture Profiles
7-A - Applications
7-R - Real-time
7-M - Microcontroller

v8

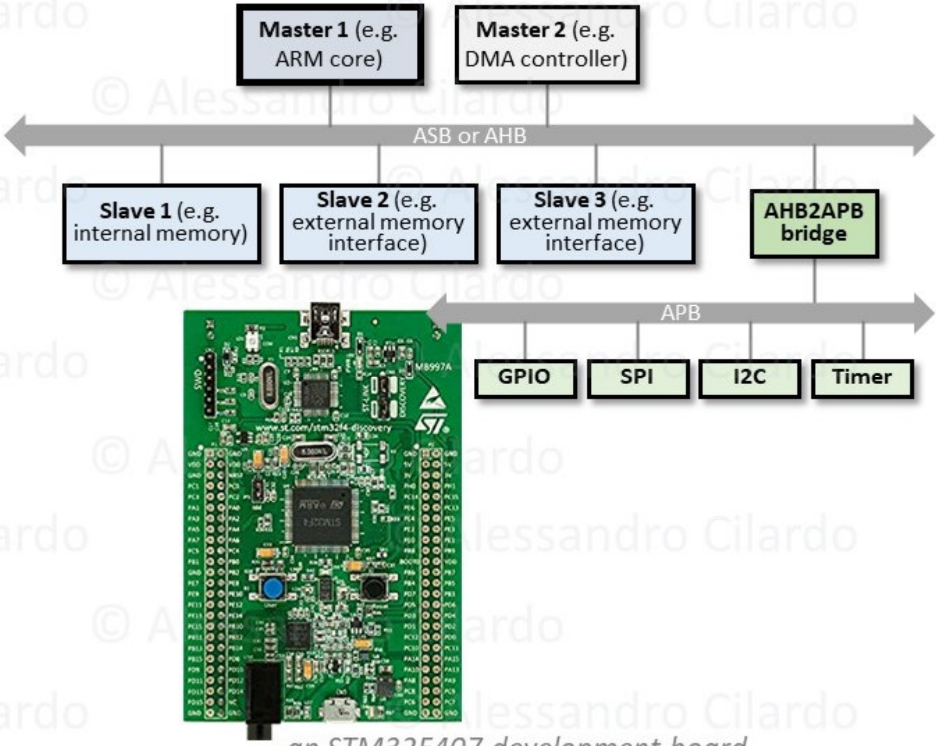
Key features of the ARMv7-A compatibility (Thumb2, TrustZone, VFPv3/v4, NEON, Adv SIMD,..)

A32+T32 ISAs
- scalar FP (SP and DP)
- Adv SIMD (SP float)
- . . .

A64 ISA
- scalar FP (SP and DP)
- Adv SIMD (SP+DP float)
- . . .

ARM-based systems on chip (SoC)

- One or more ARM cores and associated system peripherals
  - on-chip RAM and Flash memory
  - plus, possibly, off-chip RAM and non-volatile memory
- Multiple on-chip peripherals and controllers
- Interconnect:
  - Advanced High-performance Bus (AHB)
  - Advanced Peripheral Bus (APB)
- An example: STmicroelectronics STM32 family


Master 1 (e.g. ARM core)
Master 2 (e.g. DMA controller)
ASB or AHB
Slave 1 (e.g. internal memory)
Slave 2 (e.g. external memory interface)
Slave 3 (e.g. external memory interface)
AHB2APB bridge
APB
GPIO
SPI
I2C
Timer

an STM32F407 development board featuring an Arm Cortex-M4 based SoC

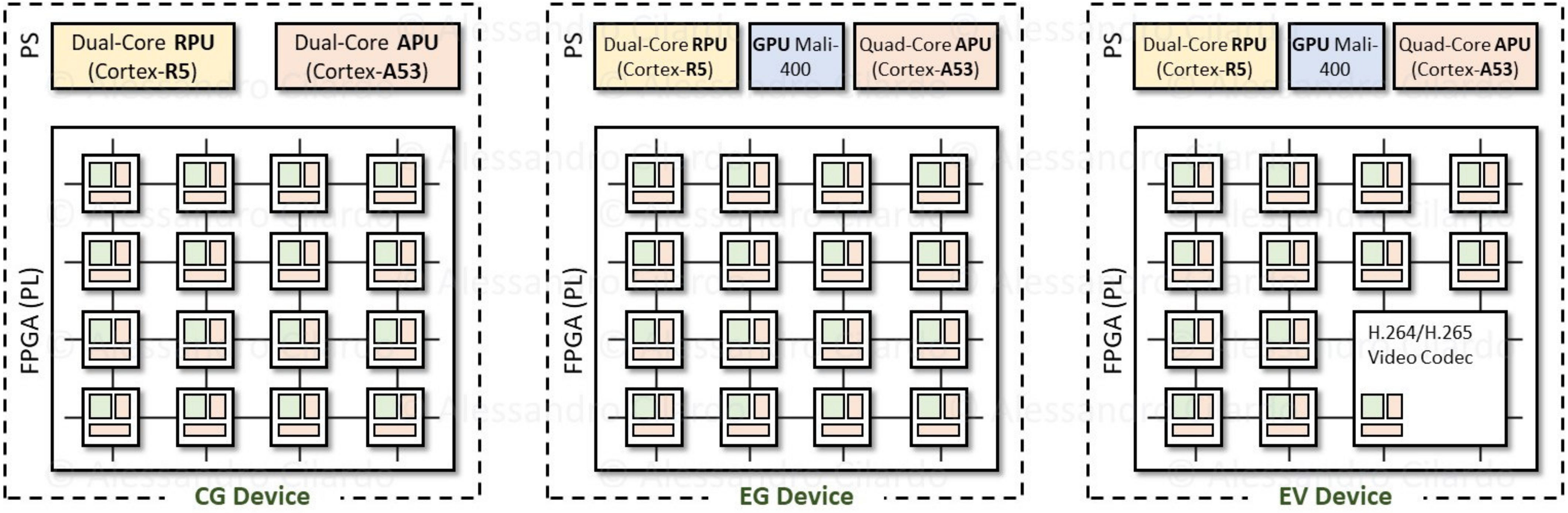
"Application"-class ARM-based MPSoC: Xilinx Zynq

Copyright © Xilinx Inc.

- Zynq-7000 SoC (2011)
  - Processing System
    - Application Processor Unit (APU)
    - Interconnects and memory interfaces (CAN, I2C, USB, ..)
    - I/O Peripherals
  - Programmable Logic (used for custom as well as programmable cores like Microblaze)
  - PS Frequency: up to 1GHz; PL Frequency up to 741GHz
- Zynq MPSoC (2015)
  - Dual or Quad APU, Dual RPU (opt. an Arm Mali GPU), General Purpose and Video domains
  - PS: Arm Cortex-A53 for APU, Cortex-R5 for RPU, VCU IP supporting H.265 and H.264
  - PL: Xilinx Ultrascale+, up to 1M+ Flip-Flops and 500k+ LUTs
  - PCIe Gen2, USB3.0, SATA 3.1, DisplayPort, Gigabit Ethernet, ..
  - Configuration and Security Unit, Platform Management Unit, ..
  - PS Frequency: up to 1.5GHz; PL Frequency up to 891GHz
- Application domains:
  - mobile base-station signal processing, video compression/decompression, broadcast camera equipment, navigation and radar systems, high speed switching, routing infrastructure for data centres, Advanced Driver Assistance Systems (ADAS), and even big data analytics, ..

ARM subsystem within the Zynq MPSoC

- Arm Intellectual Property (IP) licensed to Original Equipment Manufacturers (OEMs) such as Xilinx:
  - Cortex-A53 and Cortex-R5 processor + additional Arm IPs in the MPSoC
  - can be partly customized by the OEM

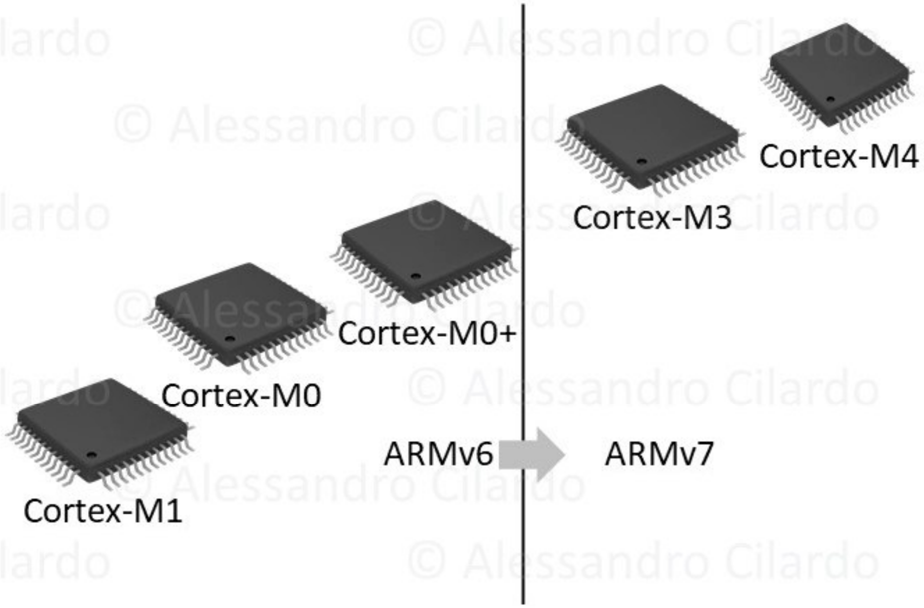

PS
Dual-Core RPU (Cortex-R5)
Dual-Core APU (Cortex-A53)
FPGA (PL)
CG Device
PS
Dual-Core RPU (Cortex-R5)
GPU Mali-400
Quad-Core APU (Cortex-A53)
FPGA (PL)
EG Device
PS
Dual-Core RPU (Cortex-R5)
GPU Mali-400
Quad-Core APU (Cortex-A53)
FPGA (PL)
H.264/H.265 Video Codec
EV Device

Arm Cortex-M profile

- Let's take a closer look at the "**M**" profile:
  - Arm M0, M0+, M1, M3, M4 cores
- what are we interested in?
  - Instruction set details and programmers model
  - Processor states/modes
  - Exception model
  - Memory model
  - Debug architecture
  - . . .

Arm Cortex-M profile

- M3 and M4 processor cores
- Architectural details
  - Three-stage pipeline
  - Harvard architecture
  - 32bit addresses (4GB memory)
  - on-chip AMBA bus
  - Nested Vectored Interrupt Controller (NVIC)
  - optional Memory Protection Unit (MPU)
  - 8 to 64 bit data, basic instructions + MAC and saturation arithmetic, bit fields, system control and OS support
  - M4: enhanced DSP support, SP FP operations

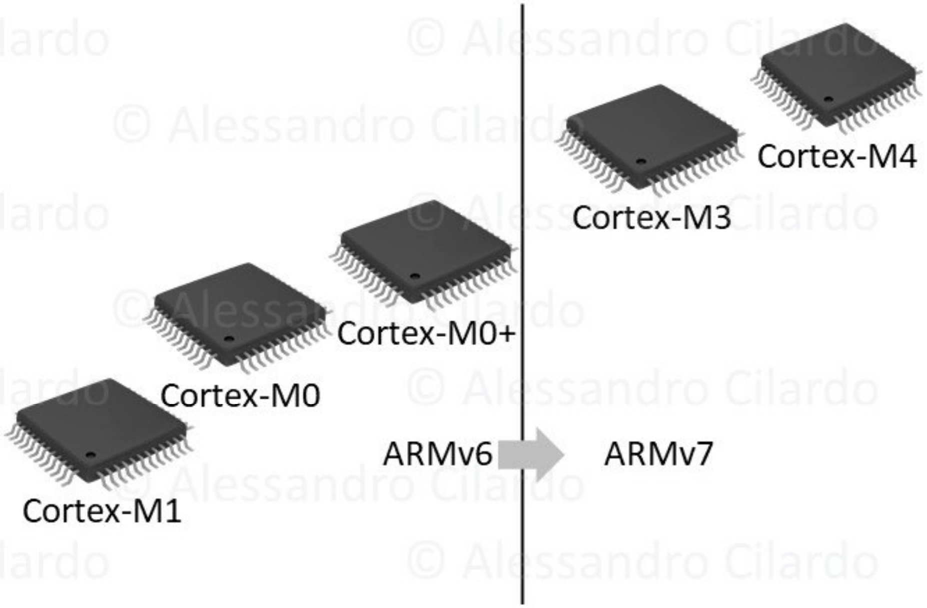


More detail in:

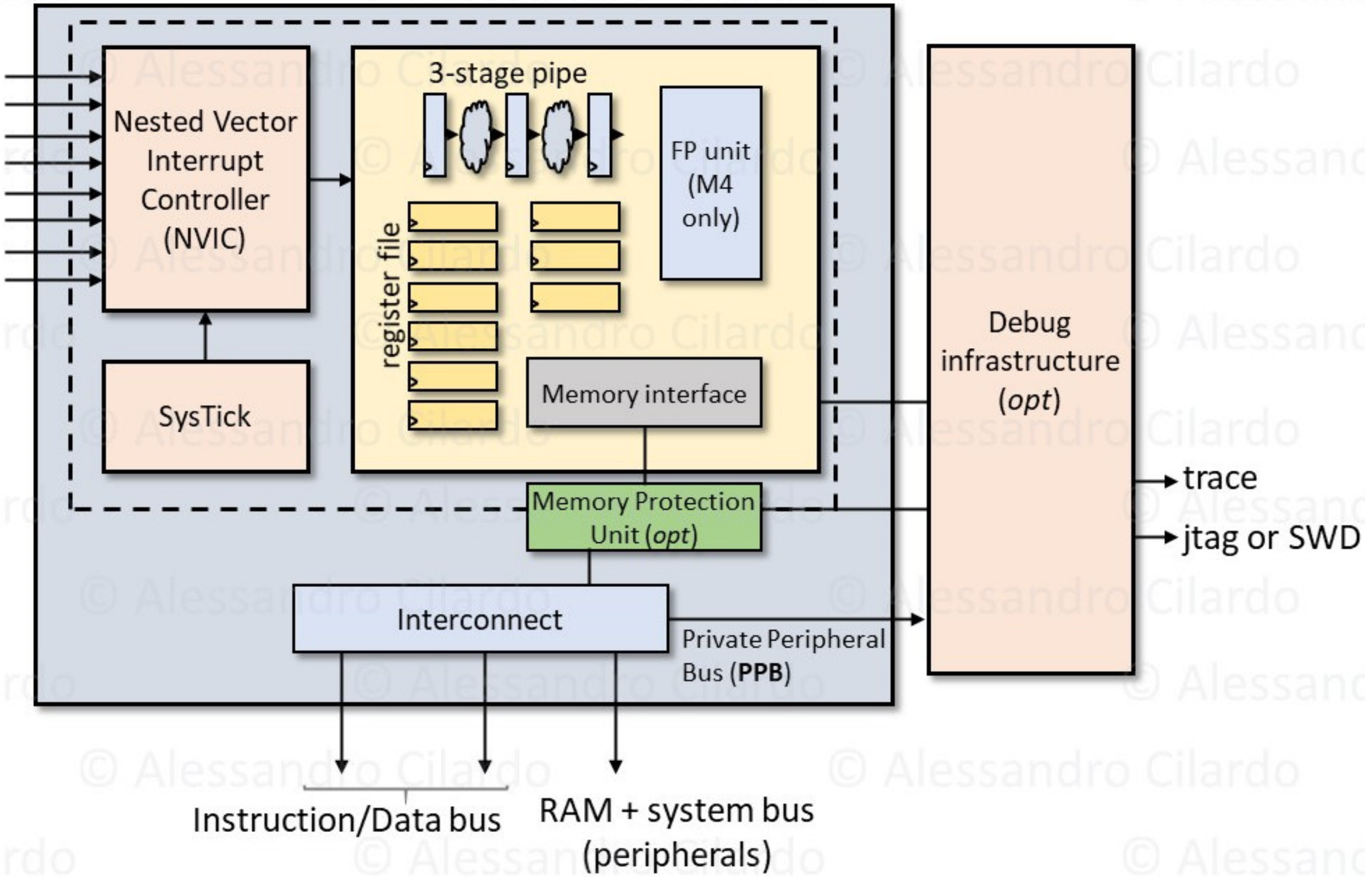
[1] J. Yiu, *The Definitive Guide to ARM® CORTEX®-M3 and CORTEX®-M4 Processors*, 3rd ed., Elsevier, 2014

Arm Cortex-M profile: advantages

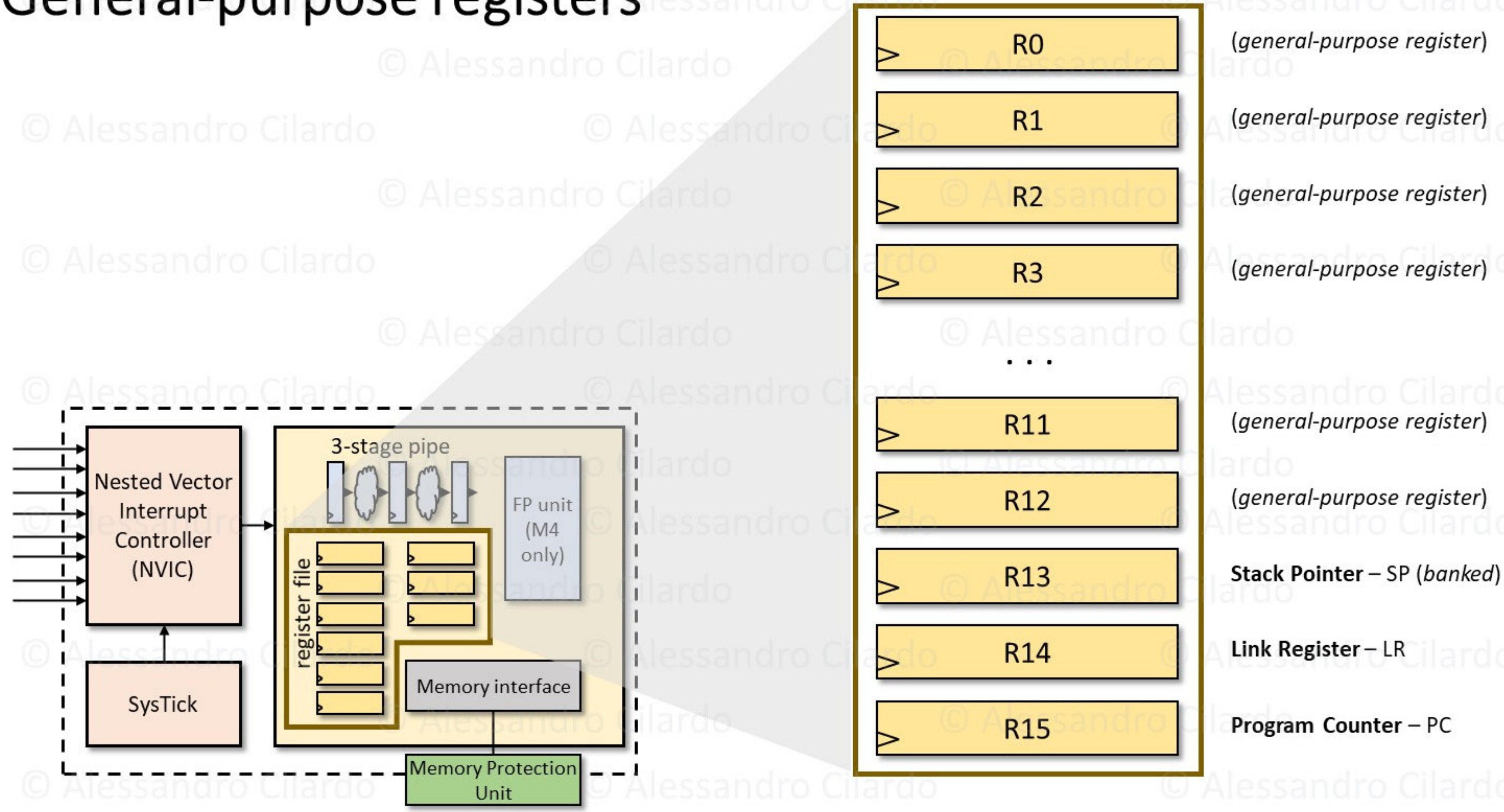
- Low power consumption
  - as low as 200 or even 100 µA/MHz
- Performance
  - e.g. 3 CoreMark/MHz and 1.25 DMIPS/MHz
- Code density
- Price and scalability
- "C-friendliness" and simple programming model
- Software portability and reusability
  - Cortex Microcontroller Software Interface Standard (CMSIS)
- Versatility and OS support
  - 30+ embedded OS ported to M3/M4
- Tool support and debug features
- Areas of application
  - Microcontrollers and Systems-on-Chips (SoC), Automotive, Data communications, Industrial control, Consumer products, Mixed signal designs, . .



Cortex-M3/M4 block diagram and bus interfaces


Nested Vector Interrupt Controller (NVIC)
SysTick
3-stage pipe
register file
FP unit (M4 only)
Memory interface
Memory Protection Unit (*opt*)
Interconnect
Private Peripheral Bus (**PPB**)
Debug infrastructure (*opt*)
trace
jtag or SWD
Instruction/Data bus
RAM + system bus (peripherals)

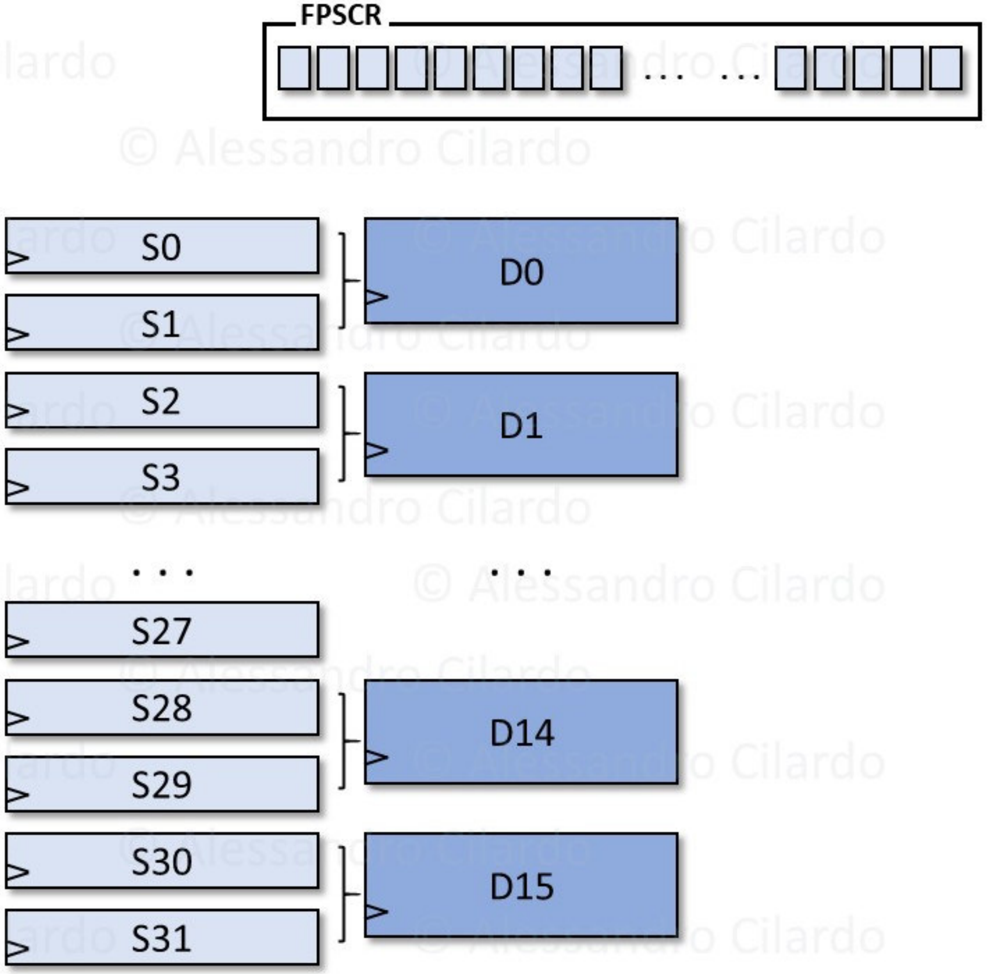
General-purpose registers



R0	(*general-purpose register*)
R1	(*general-purpose register*)
R2	(*general-purpose register*)
R3	(*general-purpose register*)
. . .	
R11	(*general-purpose register*)
R12	(*general-purpose register*)
R13	**Stack Pointer** – SP (*banked*)
R14	**Link Register** – LR
R15	**Program Counter** – PC

Floating Point Unit, or FPU (M4 only, and later)

- FPU provides further 32 single-precision registers
- Can be viewed as either
  - 32 x 32-bit registers
  - 16 x 64-bit doubleword registers
  - Any combination of the above
- Operation controlled by a *Floating Point Status and Control Register* (**`FPSCR`**)
- FPU also introduces several additional memory-mapped registers into the system
  - e.g., Coprocessor Access Control Register (**`CPACR`**)



Instruction set: Thumb and Thumb-2

- Thumb instruction set
  - a compact 16-bit encoding for a subset of the ARM instruction set
  - introduced since ARM7TDMI (released in 1994)
  - some of the instruction operands are implicit
  - limits the number of possibilities for operand addressing
  - only branches can be conditional
  - only half of all of the CPU's general-purpose registers can be accessed
  - can make a difference where RAM and memory bandwidth are an issue
  - of course, may require additional instructions for the same functionality
  - improves code density by 35%
- **Thumb-2**: variable-length ISA (some instructions are 32-bit long)
  - appeared with ARM1156 core, announced in 2003
  - bit-field manipulation, table branches, and conditional execution, . . .
  - ARM M processors only support **Thumb-2**
- Unified Assembly Language (UAL)
  - supports generation of either Thumb or ARM instructions from the same source code
  - e.g. use "if-then" instructions to emulate in Thumb the conditional instructions provided by ARM

Cortex-M3/M4 (Thumb2) instruction set overview

- General instruction types:
  - Moving data within the processor
  - Memory accesses
  - Arithmetic operations
  - Logic operations
  - Shift and Rotate operations
  - Conversion (extend and reverse ordering) operations
  - Bit field processing instructions
  - Program flow control (branch, conditional branch, conditional execution, and function calls)
  - Multiply accumulate (MAC) instructions
  - Divide instructions
  - Memory barrier instructions
  - Exception-related instructions
  - Sleep mode-related instructions
  - Other functions
- In addition, Cortex-M4 supports Enhanced DSP instructions:
  - SIMD operations and packing instructions
  - Fast multiply and MAC instructions
  - Saturation algorithms
  - Floating point instructions (if the floating point unit is present)

Cortex-M3/M4 (Thumb2) instruction set overview

- Recent ARM development tools support Unified Assembler Language (UAL)
  - allows better portability between architectures
  - allows use of a single Assembly language syntax in ARM processors with various architectures

```
.equ targetAddress, 0xE000E100
. . .
LDR R0,=targetAddress   /* Put 0xE000E100 into R0. Note: ARM does NOT
                           support absolute addressing. In fact,
                           LDR is a pseudo instruction converted by
                           the assembler to a PC relative load       */
MOVS R1, #1
STR R1, [R0]            /* Store value 0x1 to address 0xE000E100     */
```

```
LDR R0,=myVariable      /* Get the memory location of myVariable     */
LDR R1, [R0]            /* Read the data at memory address in R0     */
. . .
LDR R0,=myFunction      /* Get the starting address of a subroutine  */
. . .
BL myFunction           /* Call a function by its starting address   */
. . .
.align 4                /* Enforce data alignment                    */
myVariable:
.word 0xAB12CD34
```

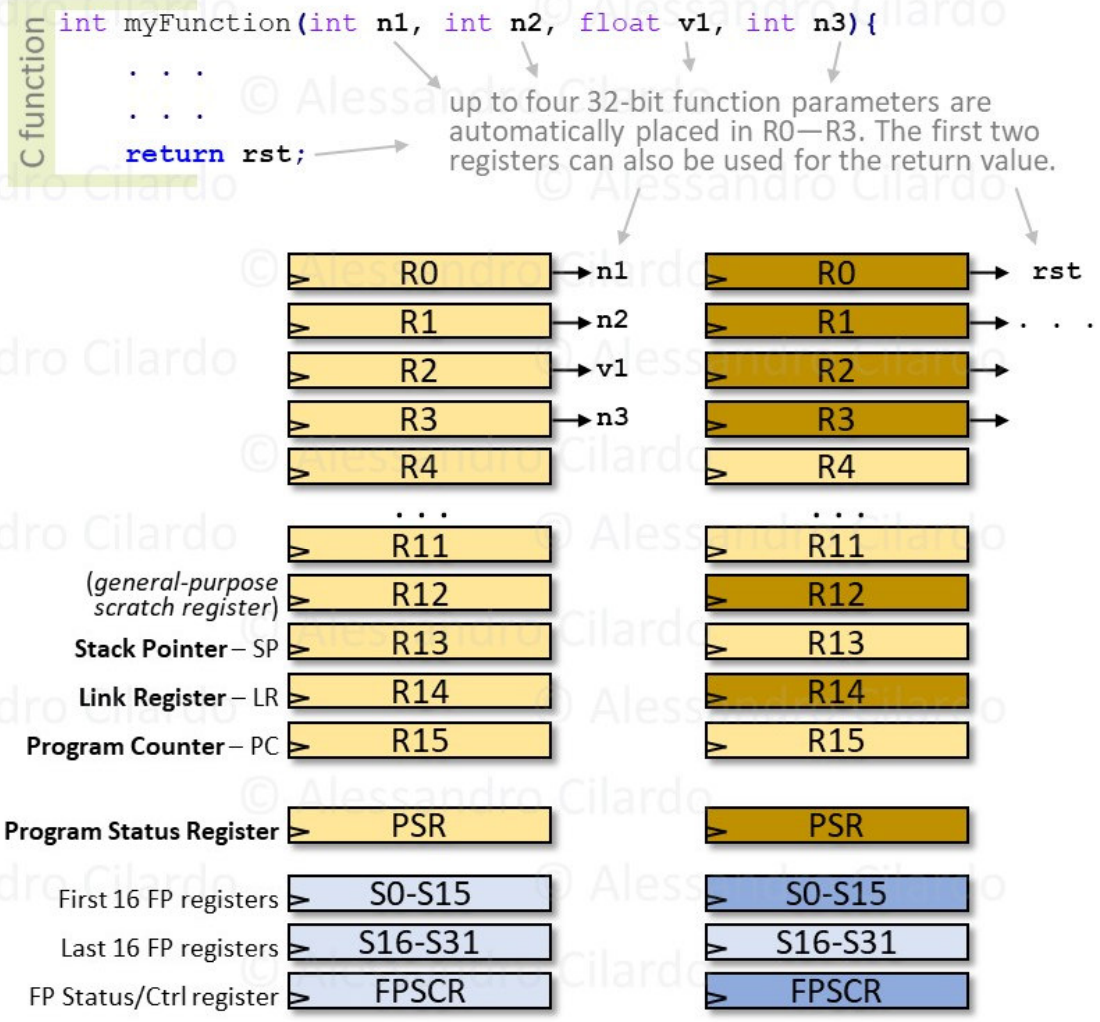
```
ADR R5, myData
. . .
ALIGN
myData
DCD -23, 1, 324, -543, . . .
```

A few random examples of ARM assembly code

```
MOVW R6, #0x5678     /* Set R0 to 32-bit value 0x00005678           */
MOVT R6, #0x1234     /* Set the upper 16 bits of R0 to 0x1234       */
                     /* After these two instructions, R0 = 0x12345678 */
                     /* Note: ARM cannot support 32-bit immediates   */
```

AAPCS standard

- **A**rm **A**rchitecture **P**rocedure **C**all **S**tandard
  - defines how C and asm subroutines can be separately written, compiled, and assembled to work together
- A *contract* between a calling routine and a called routine, involving:
  - obligations on the caller to create a certain program state for the called routine
  - obligations on the called routine to preserve the caller's program state across the call
  - the rights of the called routine to alter the caller's program state
- Can be used to easily write combined C/assembler routines
  - e.g. regulating parameter passing through registers
- *Note*: Exception handling mechanisms in ARM allows handlers to be written as normal C functions which follow AAPCS



```
int myFunction(int n1, int n2, float v1, int n3){
    . . .
    . . .
    return rst;
```

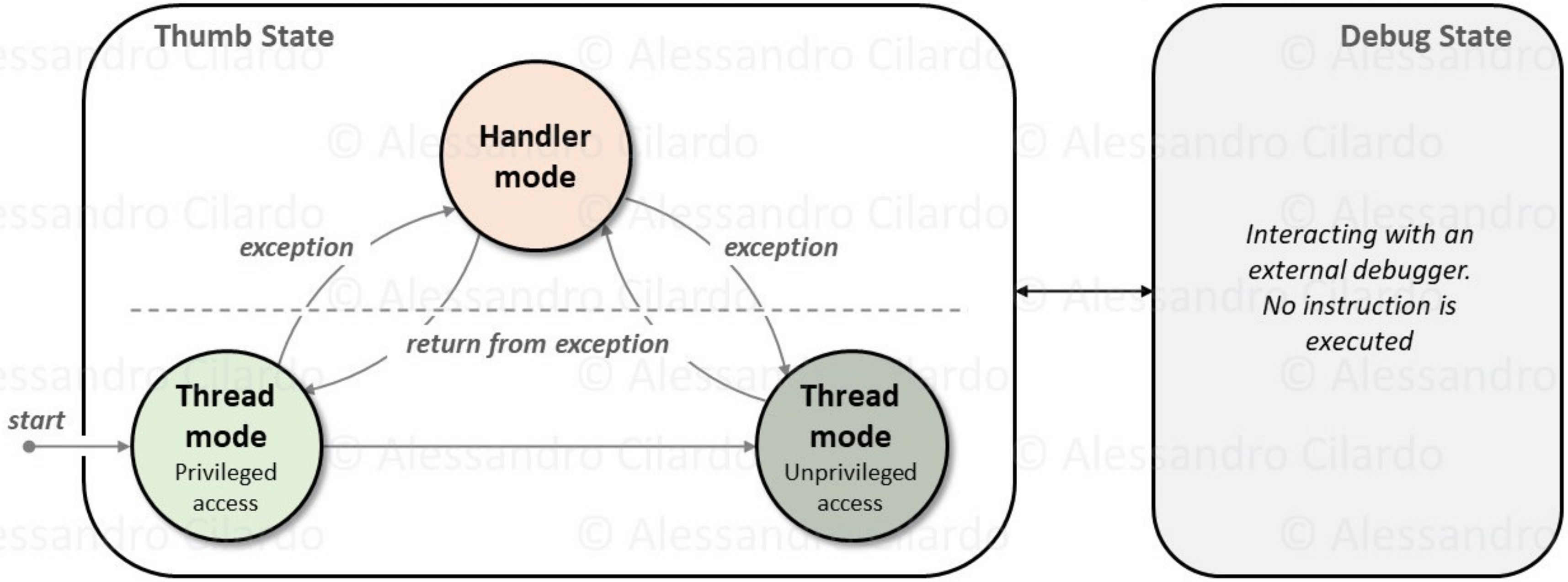
up to four 32-bit function parameters are automatically placed in R0—R3. The first two registers can also be used for the return value.

Register usage in a function call in AAPCS

(note: darker boxes denote registers that can be changed after a function call)

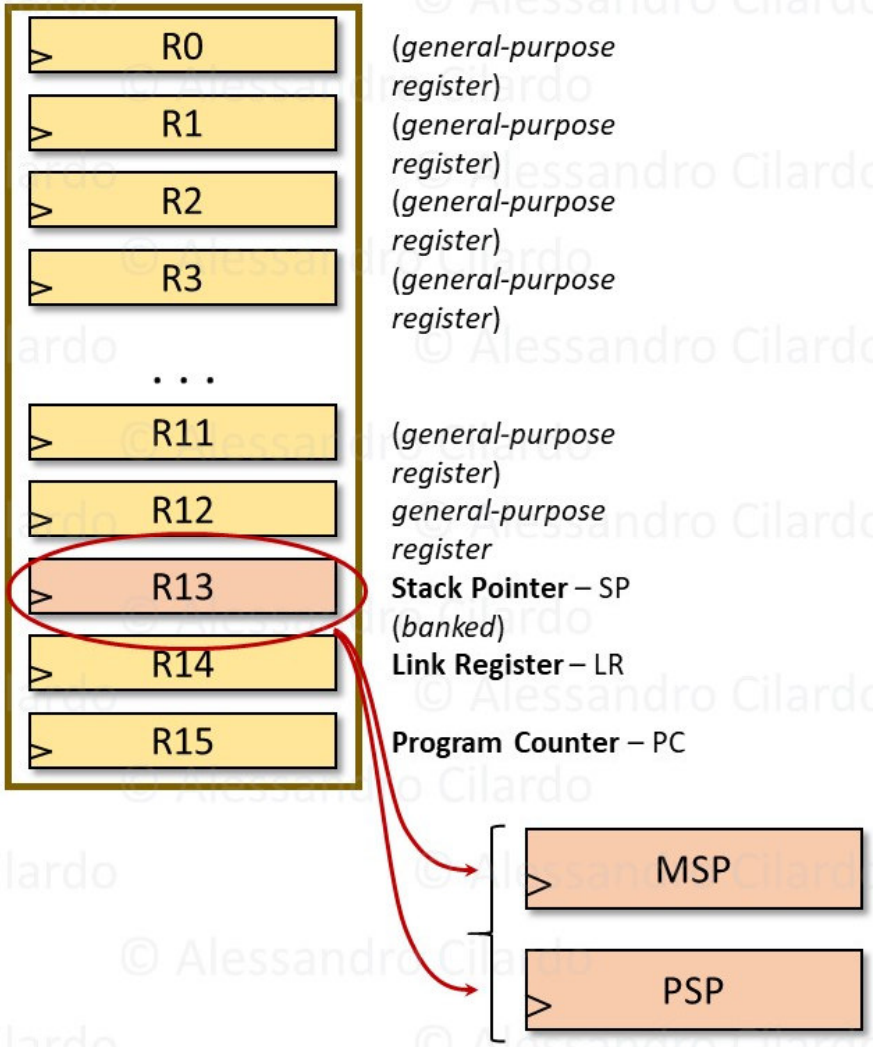
M3/M4 execution *states*, *modes* and *privilege* levels

- States: *Debug*, *Thumb* (note: no "ARM" state in M-profile)
- Modes: *Handler*, *Thread*
- Thread mode can be *privileged* and *unprivileged* (sometimes called "user state")

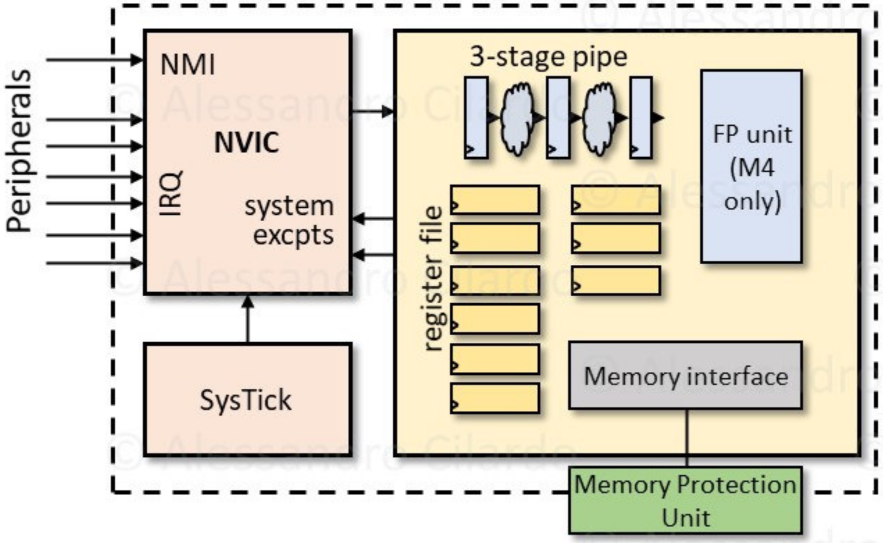


Banked Stack Pointers

- One of multiple, physically distinct Stack Pointers is used depending on the mode
- For Stack Pointers, note:
  - PSP can only be used in Thread Mode
  - selection of Stack Pointer is determined by a special register (`CONTROL`)


R0
(*general-purpose register*)
R1
(*general-purpose register*)
R2
(*general-purpose register*)
R3
(*general-purpose register*)
. . .
R11
(*general-purpose register*)
R12
general-purpose register
R13
Stack Pointer – SP (*banked*)
R14
Link Register – LR
R15
Program Counter – PC
MSP
PSP

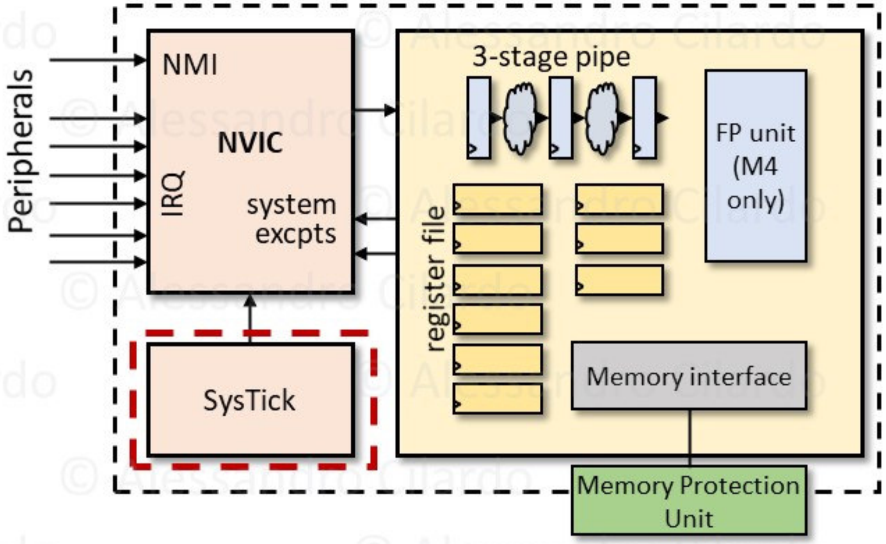
Exceptions and interrupts



Number (*)	Priority	Type/Description
1	-3	**Reset**
2	-2	**NMI** – Non-Maskable interrupt
3	-1	**HardFault** – all faults
4	*settable*	**MemManage** – MPU violation or invalid memory access fault
5	*settable*	**BusFault** – bus error (instruction prefetch abort or data access error)
6	*settable*	**Usage fault** – invalid/unsupported instruction or invalid state transition
7-10	-	(*reserved*)
11	*settable*	**SVC** – Supervisor Call, based on SVC instruction
12	*settable*	**Debug** – reserved to software-based debug settings
13	-	(*reserved*)
14	*settable*	**PendSV** – Pendable request for a system service
15	*settable*	**SYSTICK** – System timer interrupt
16-255	*settable*	**IRQ** – Interrupt requests from peripherals

(*) *Interrupt numbering used in the CMSIS framework is different (interrupt numbers are dimished by 16 in CMSIS)*

System Timer (SysTick)



- A flexible system timer, providing
  - a 24-bit self-reloading down counter
    - Reload whenever the counter reaches 0
    - Optionally raise a SysTick interrupt on 0
  - a Reload Register
  - a Calibration Register
- Clock source is CPU clock or optional external timing reference
  - Software selectable
  - The external reference pulse width must be larger than the processor clock period
    - since the processor clock is used for sampling the external reference
- The Calibration Register provides a counting value corresponding to 10ms
  - STCALIB physical inputs to the core can be used for indicating the used reference source and its properties (typically that depends on the specific SoC)

Vector table

- Contains the starting addresses of Exception Handlers
- Because handlers start at 4-byte aligned addresses, the two Least Significant Bits (LSBs) are always 0
  - in fact, as a trick, Arm uses the LSB to automatically change the state (Arm/Thumb) when jumping to a Handler:

    if the Handler is to be run in Thumb state, the LSB is set to 1
  - In Cortex-M cores (Thumb only) the LSB must always be 1
- Can be relocated and changed by a user application
  - for example, after boot, a new Vector Table can be loaded from an external SD and stored to a different starting address
  - relocation is controlled by a programmable register in the NVIC: Vector Table Offset Register (**VTOR**)

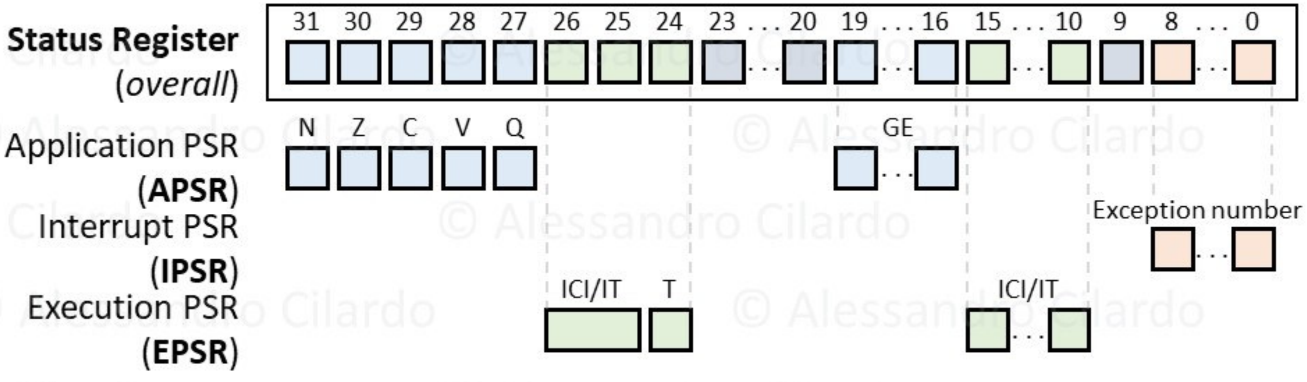
IRQ 239	0x000003FC
IRQ 238	0x000003F8
... ...	... ...
IRQ 1	0x00000044
IRQ 0	0x00000040
SysTick	0x0000003C
PendSV	0x00000038
(reserved)	0x00000034
Debug Monitor	0x00000030
SVC	0x0000002C
(reserved)	0x00000028
... ...	... ...
(reserved)	0x0000001C
Usage Fault	0x00000018
Bus Fault	0x00000014
MemManage Fault	0x00000010
HardFault	0x0000000C
NMI	0x00000008
Reset	0x00000004
Initial MPS	0x00000000

Note: In Cortex-M cores, Handler addresses in the Vector Table always have the Least Significant Bit set to 1

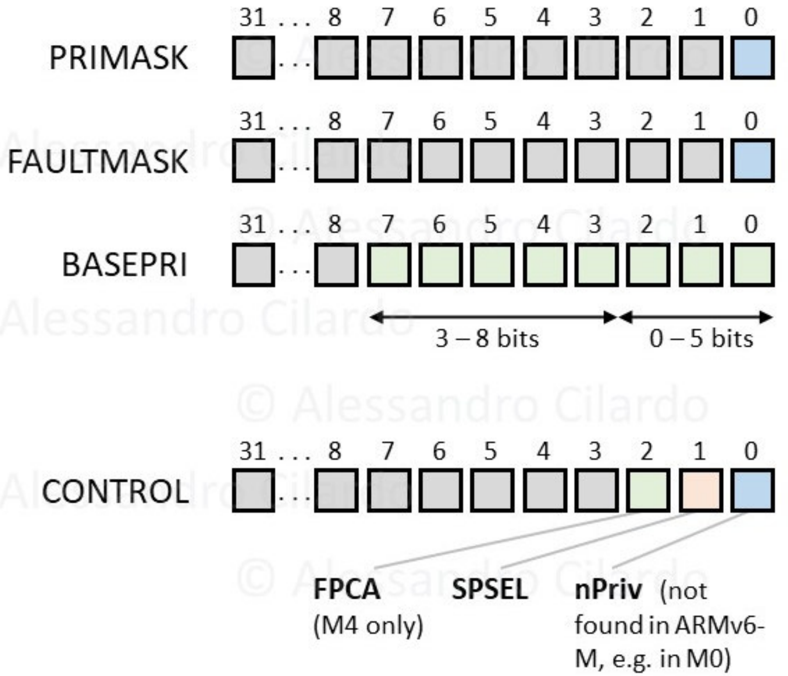
Note: The Vector Table used at boot starts at Address 0x00000000. An application-loaded Vector Table can start at a different address, keeping the same offsets in the table

Special Registers

- Program Status Register (PSR)
  - Application PSR (**APSR**)
  - Execution PSR (**EPSR**)
  - Interrupt PSR (**IPSR**)
  - status register can be read/written through special instructions (**MRS**, **MSR**)



- Exception/Interrupt masking registers
  - **PRIMASK**: mask all interrupts but hard faults and NMI
  - **FAULTMASK**: mask all interrupts but NMI
  - **BASEPRI**: mask all interrupts with priority ≤ *value*
  - Used to modify exception priorities
  - **PRIMASK** and **FAULTMASK** set/clear through Change Processor State (**CPS**) instructions
    - **CPSIE i / CPSID i / CPSIE f / CPSID f**
- Control register (**CONTROL**):
  - privilege level, SP selection, FP context (M4-only)
  - SPSEL: if 0, Thread mode uses MSP, otherwise PSP

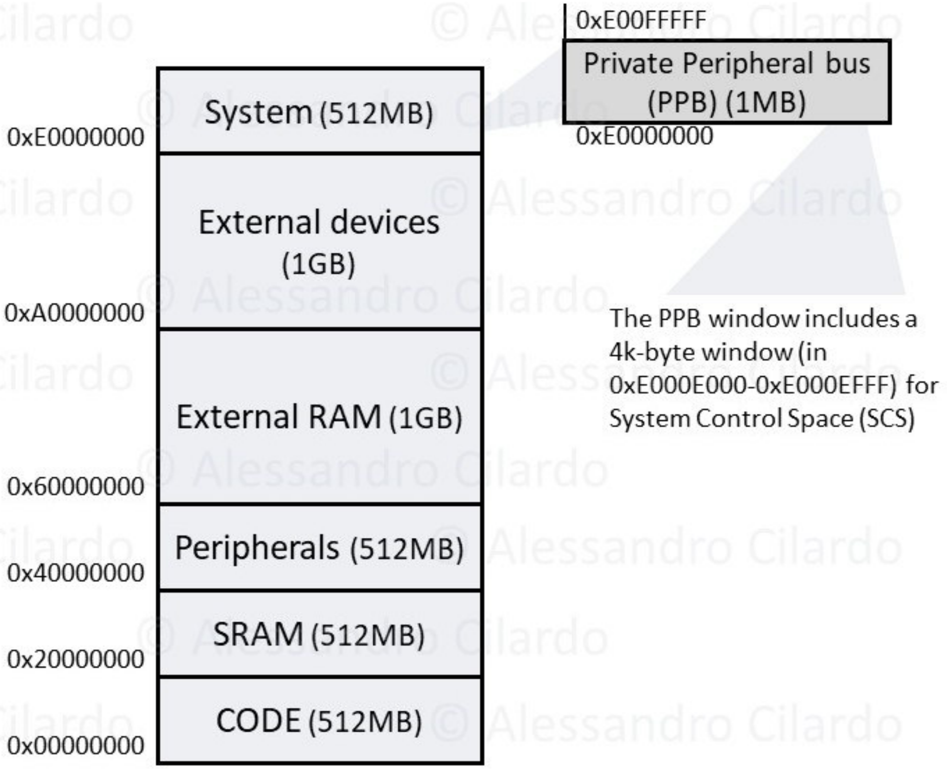


Memory subsystem

- 4GB linear address space
- Architecturally defined memory map
- Support for little endian and big endian memory systems
- Bit band accesses (*optional*)
- Write buffer
- Memory Protection Unit (*optional*)
- Unaligned transfer support

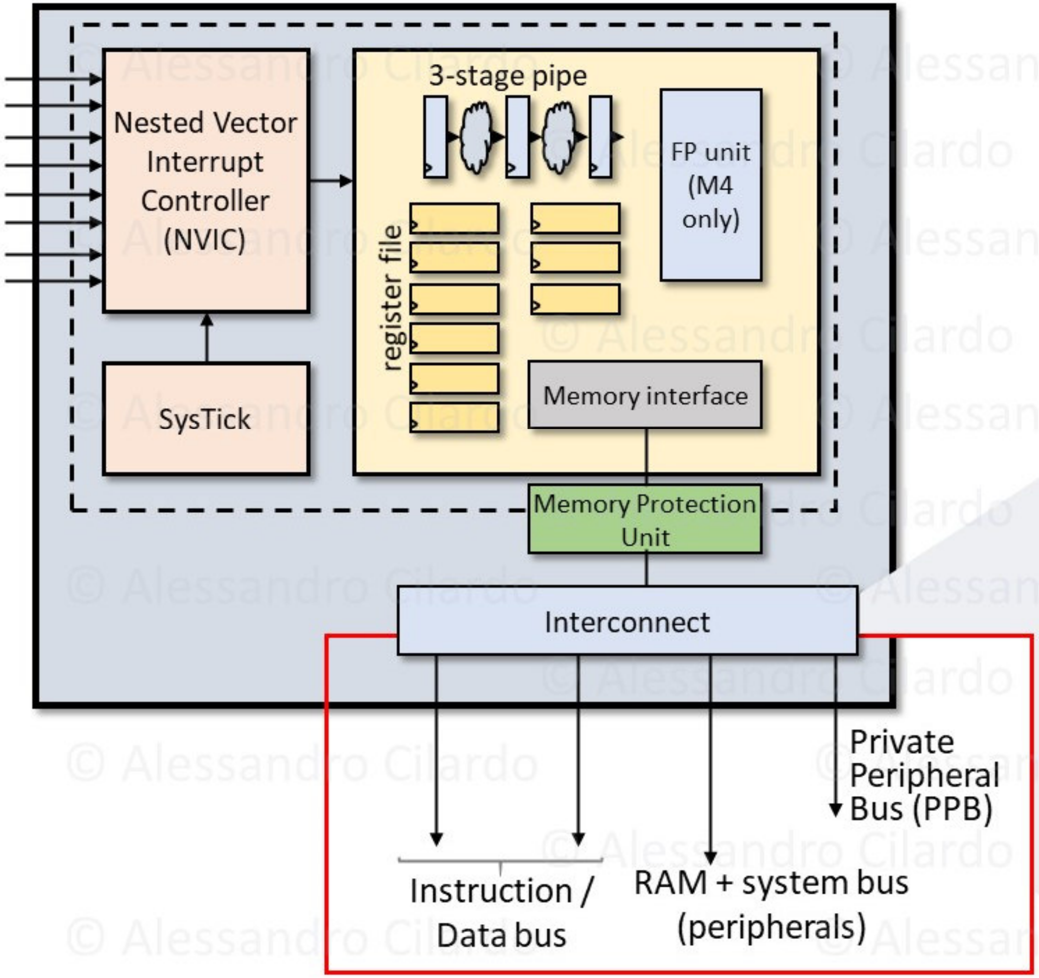
Memory subsystem: memory map

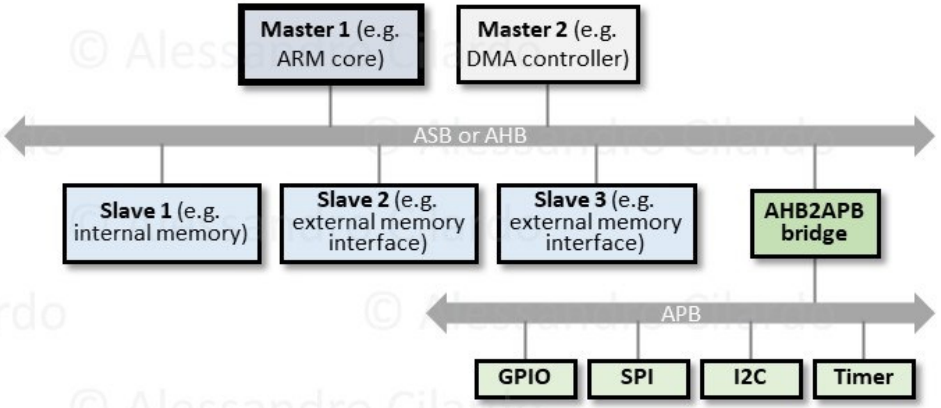
- CODE region
  - Application program code
  - Vector Table
- SRAM region
  - Application data
- Peripherals
- External Memory
- External Peripherals
- Private Peripheral Bus
  - processor's internal control, e.g. NVIC, and debug components

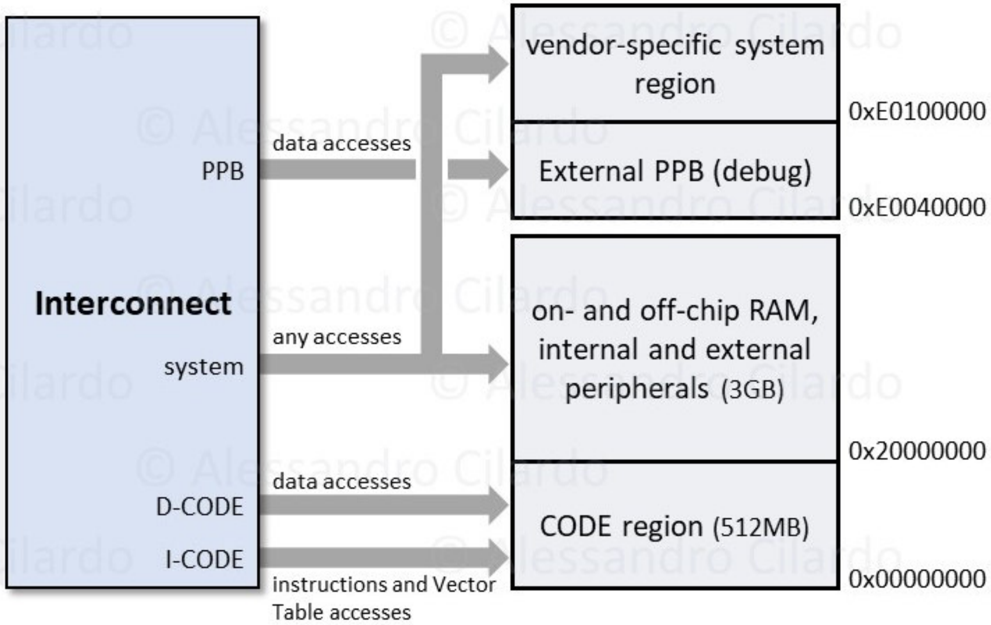


The PPB window includes a 4k-byte window (in 0xE000E000-0xE000EFFF) for System Control Space (SCS)

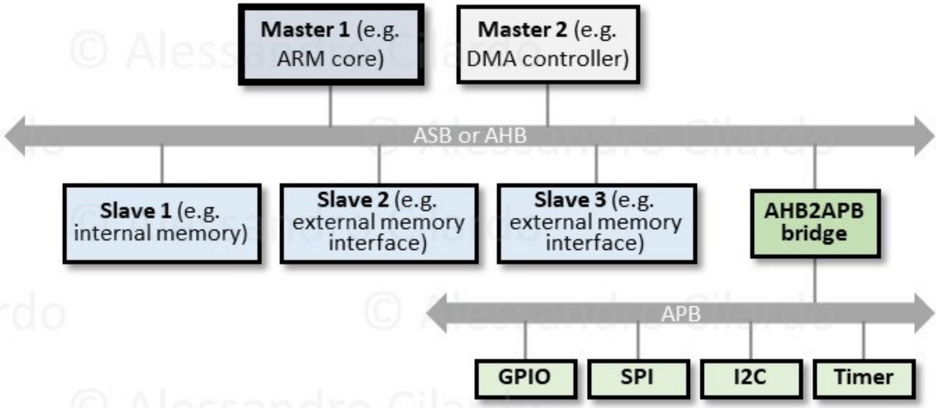
Memory subsystem: physical organization




Master 1 (e.g. ARM core)
Master 2 (e.g. DMA controller)
ASB or AHB
Slave 1 (e.g. internal memory)
Slave 2 (e.g. external memory interface)
Slave 3 (e.g. external memory interface)
AHB2APB bridge
APB
GPIO
SPI
I2C
Timer



Memory subsystem: physical organization



Master 1 (e.g. ARM core)

Master 2 (e.g. DMA controller)

ASB or AHB

Slave 1 (e.g. internal memory)

Slave 2 (e.g. external memory interface)

Slave 3 (e.g. external memory interface)

AHB2APB bridge

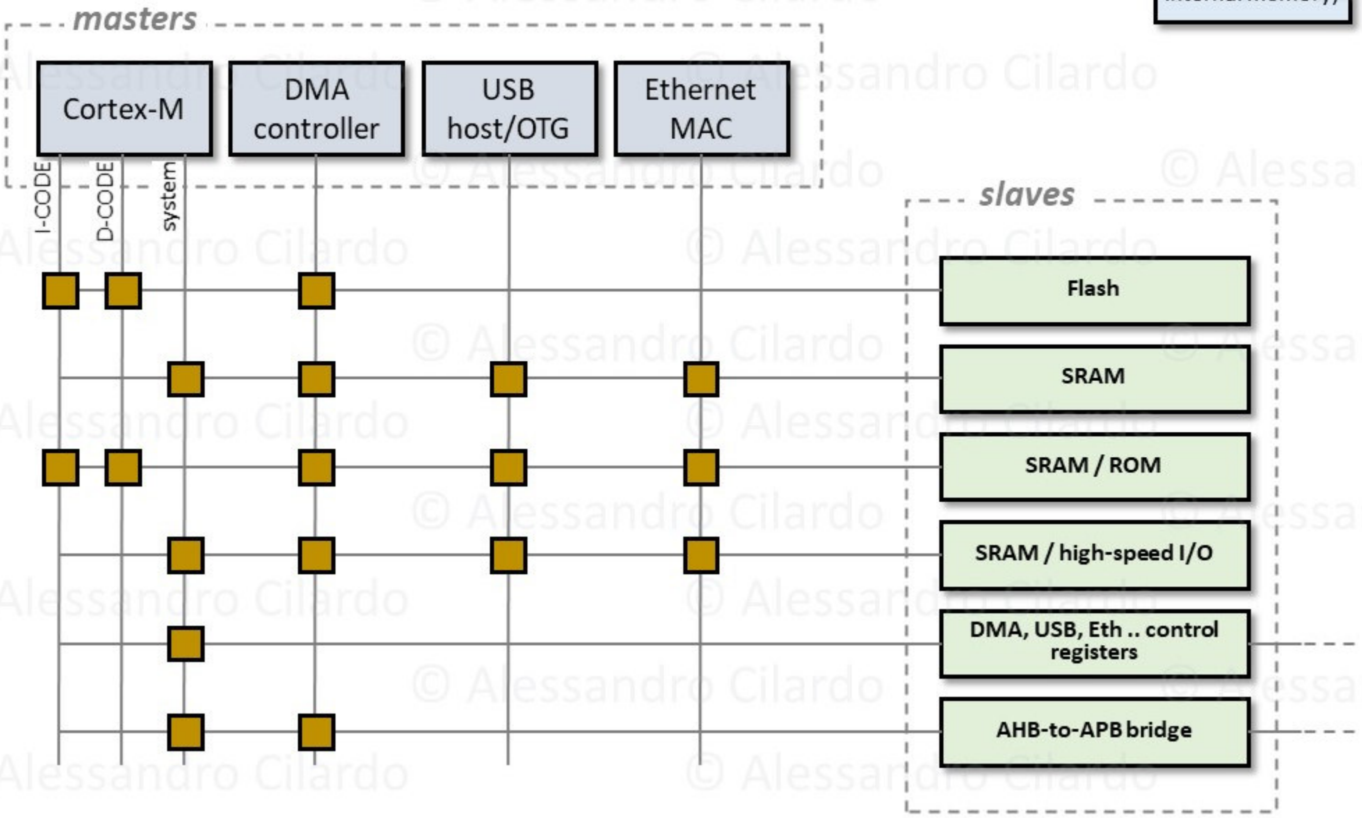
APB

GPIO

SPI

I2C

Timer



masters

Cortex-M

DMA controller

USB host/OTG

Ethernet MAC

I-CODE

D-CODE

system

slaves

Flash

SRAM

SRAM / ROM

SRAM / high-speed I/O

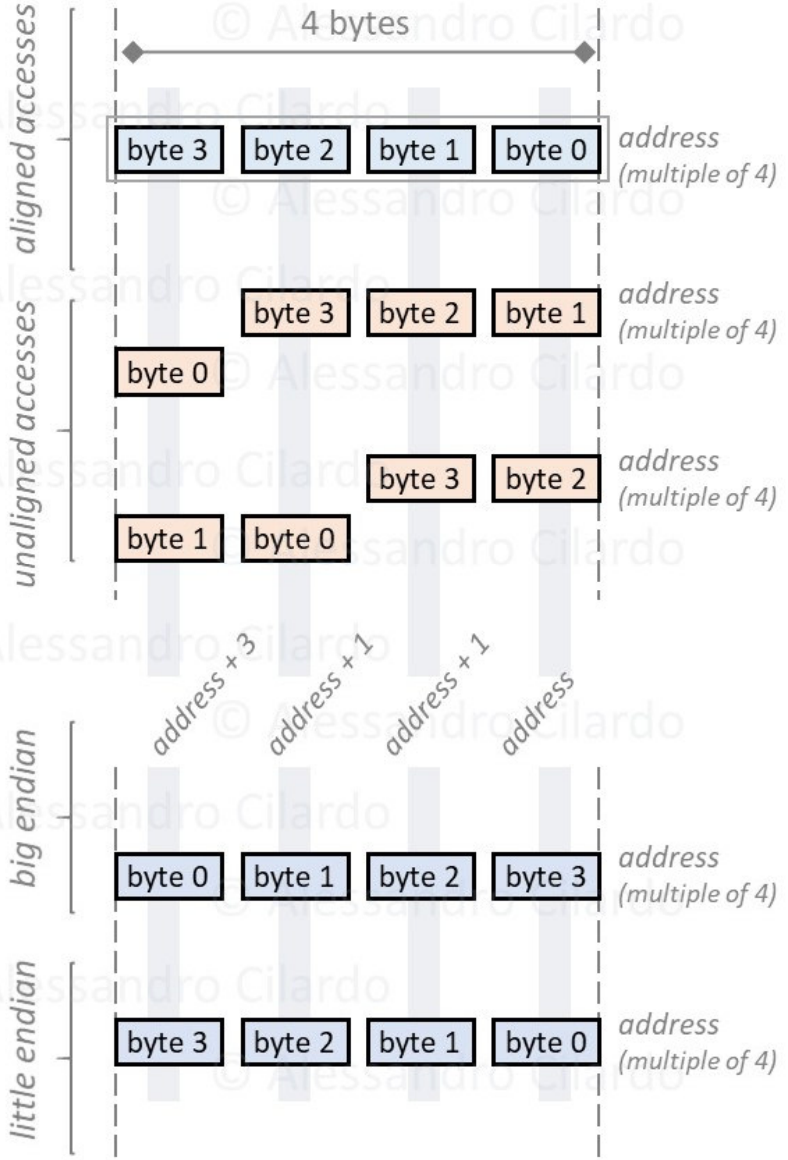
DMA, USB, Eth .. control registers

AHB-to-APB bridge

Alignment and Endianess

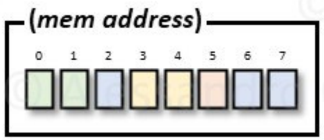
- Cortex-M3 and Cortex-M4 cores support unaligned data transfers in normal memory access instructions
- ..with a few limitations:
  - Unaligned transfers are not supported in Load/Store Multiple instructions (LDM/STM)
  - Stack operations (PUSH/POP) must be aligned
  - Exclusive accesses (such as LDREX or STREX) must be aligned (a usage fault is raised otherwise)
  - Unaligned transfers are not supported in bit-band operations (results will be unpredictable)

- Most of the existing Cortex-M microcontrollers are little endian
  - but both endianess types are supported

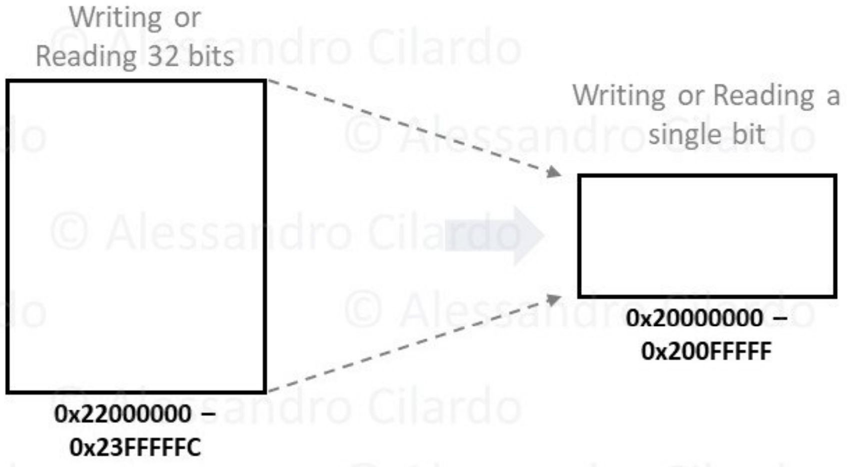


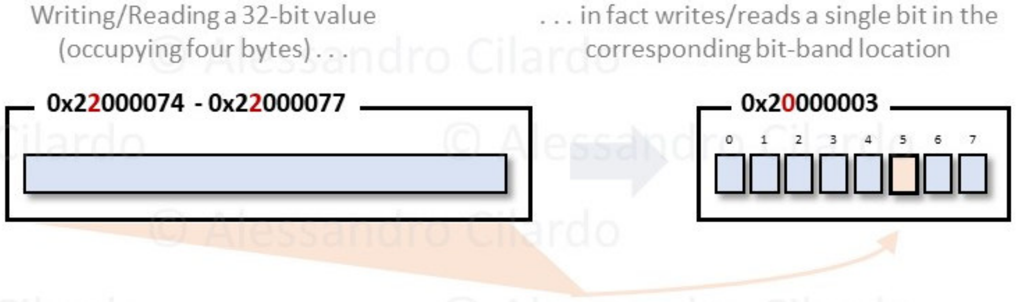
Bit-band region

- When device registers which contain distinct bit fields are memory mapped to the same byte location
  - accessing specific bit fields can be an issue
  - the processor needs to read the whole byte, modify the bit fields without touching the other bits, and write the byte back to memory
  - race conditions might occur if other players (other masters in the system or the device itself) change other bits concurrently in the same register
  - this situation would require an atomic *read-modify-write* sequence from the processor
- *Bit-band region*: an "alias" region for re-directing writes/reads (at the hardware level)
  - operations to an aligned 32-bit word (at addresses **0x220..0** to **0x23F..FC**) are turned into single bit operations in an associated location (located in **0x200..0** to **0x200F..F**)



A 1-byte memory location mapping an 8-bit device register, made of several distinct bit fields, meant to be accessed separately




Writing/Reading a 32-bit value (occupying four bytes) . . .
. . . in fact writes/reads a single bit in the corresponding bit-band location
0x22000074 - 0x22000077
0x20000003
0 1 2 3 4 5 6 7

Bit 5 of byte at 0x20000003 is bit 29 (3 x 8 + 5) as counted from base address 0x20000000. So, its corresponding bit band word is located at address 0x22000000 + 29x4 = 0x22000074

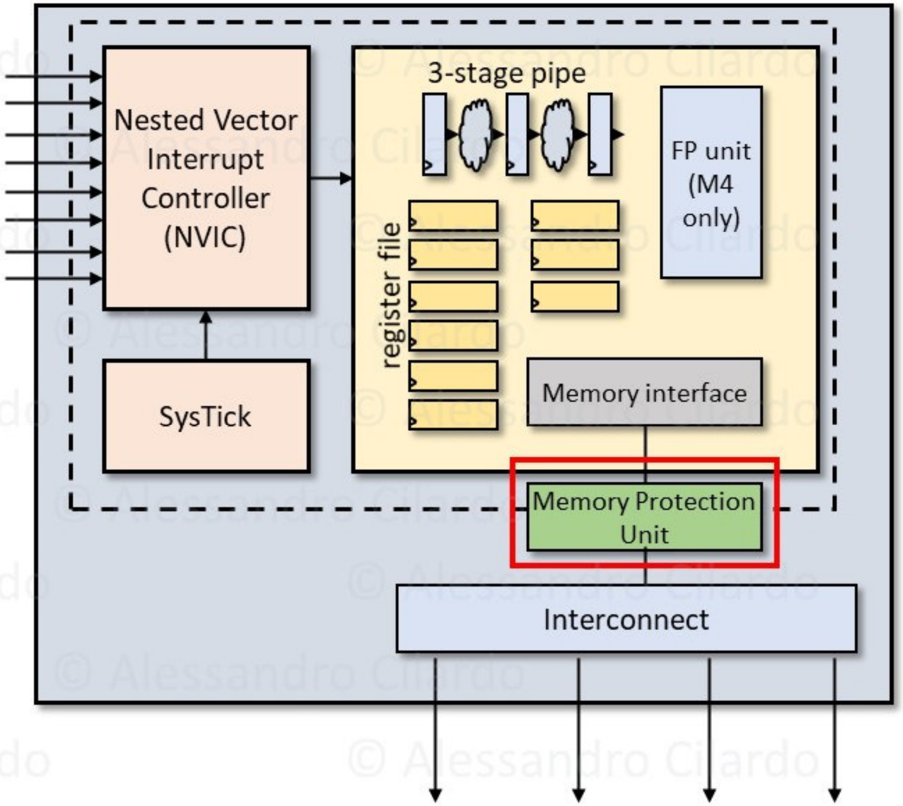
Memory attributes

- Bufferable
  - Write to memory carried out by a write buffer
- Cacheable
- Executable
  - processor can fetch and execute code from this memory region
- Sharable
  - Data could be shared by multiple bus masters (the memory system needs to ensure coherency)

Bufferable	Cacheable	Memory type and behavior
0	0	"***Strongly ordered***": wait until the transfer is completed on the bus interface before starting the next operation (if this operation is a Strongly Ordered of Device Type access)
1	0	"***Device type***": a write buffer can be used for handling a store operation
0	1	Normal memory with Write-Through cache
1	1	Normal memory with Write-Back cache

Memory protection unit (MPU)

- Divides the memory map into a number of regions
  - defines the location, size, access permissions, and memory attributes of each region
- Supports:
  - independent attribute settings for each region
  - overlapping regions
  - export of memory attributes to the system
- Memory *attributes* affect the behavior of memory accesses to the region
- The Cortex-M4 MPU defines:
  - Eight separate memory regions
  - A background region

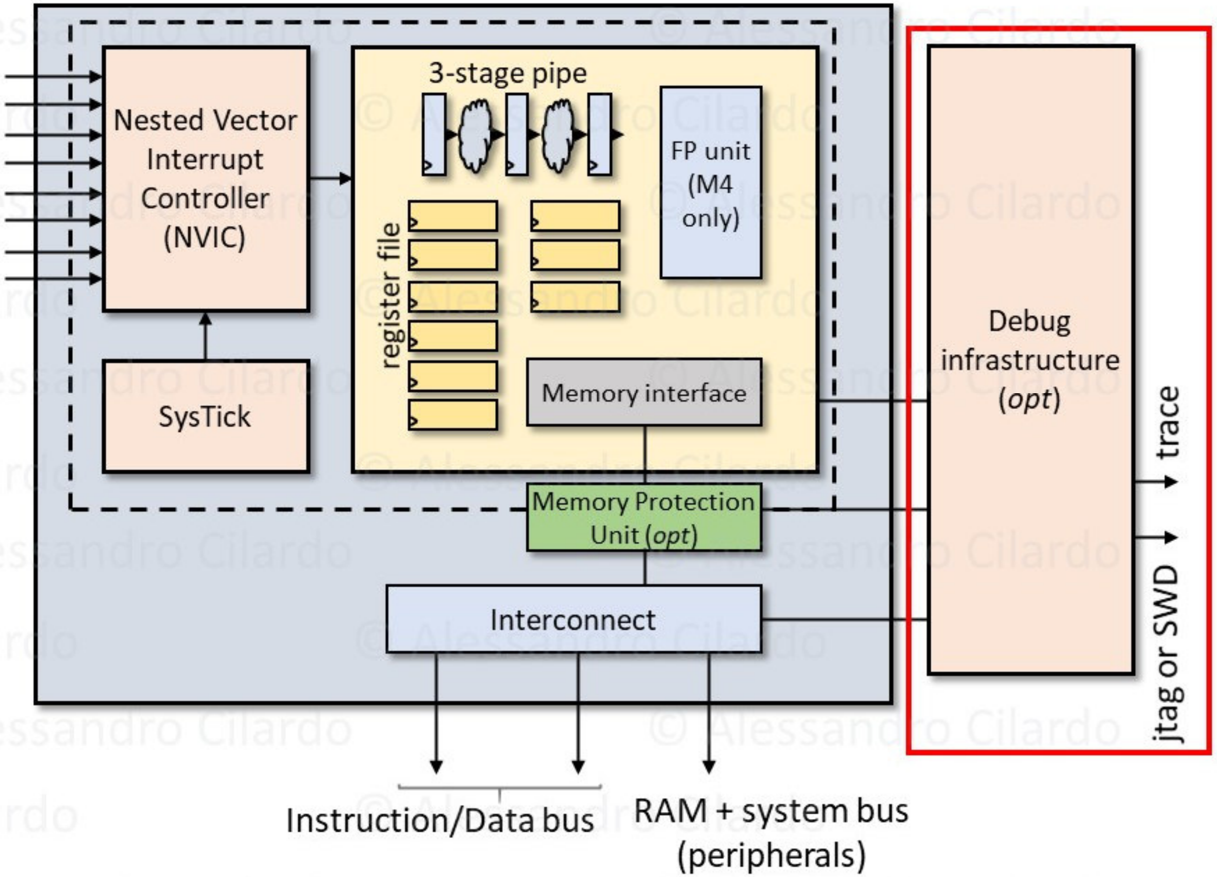


Memory operation: advanced aspects

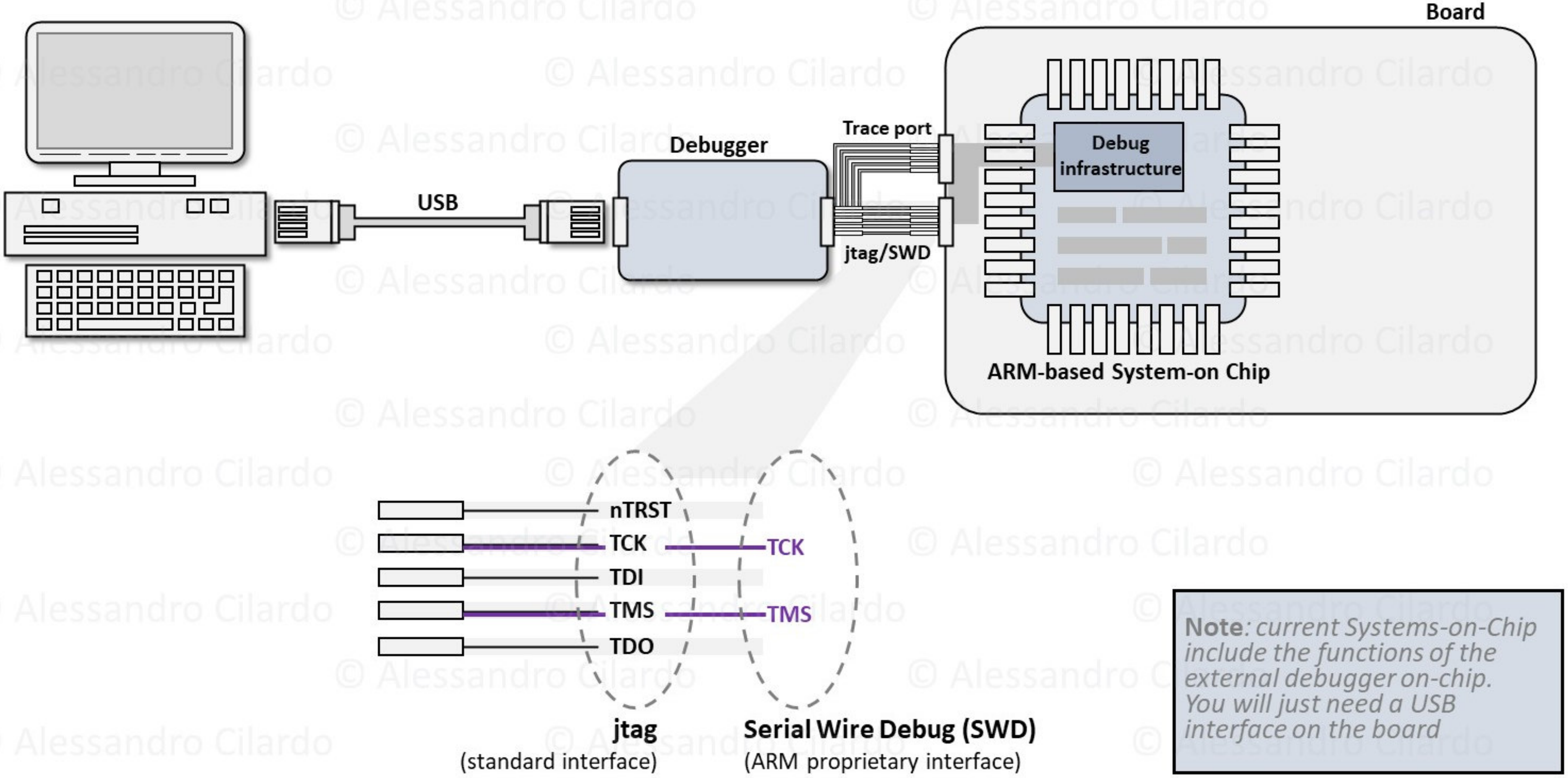
- Exclusive access instructions
  - special Load/Store instruction pair ensuring exclusive access
  - LDREX / LDREXH / LDREXB, STREX / STREXH / STREXB
- Memory barriers
  - memory barrier instructions (ISB, DSB, DMB)
  - note: Cortex-M3 and Cortex-M4 do not reorder instructions
  - barriers can still be used in a few specific cases (e.g. when activating memory remapping)

Cortex-M3 and Cortex-M4 debug support

- Includes comprehensive debugging features:
  - program execution controls
  - including halting and stepping
  - instruction breakpoints
  - data watchpoints
  - registers and memory accesses
  - profiling and traces
- Debug vs. trace modes


Nested Vector Interrupt Controller (NVIC)
SysTick
3-stage pipe
register file
FP unit (M4 only)
Memory interface
Memory Protection Unit (*opt*)
Interconnect
Debug infrastructure (*opt*)
trace
jtag or SWD
Instruction/Data bus
RAM + system bus (peripherals)

Cortex-M3 and Cortex-M4 debug support



Note: *current Systems-on-Chip include the functions of the external debugger on-chip. You will just need a USB interface on the board*