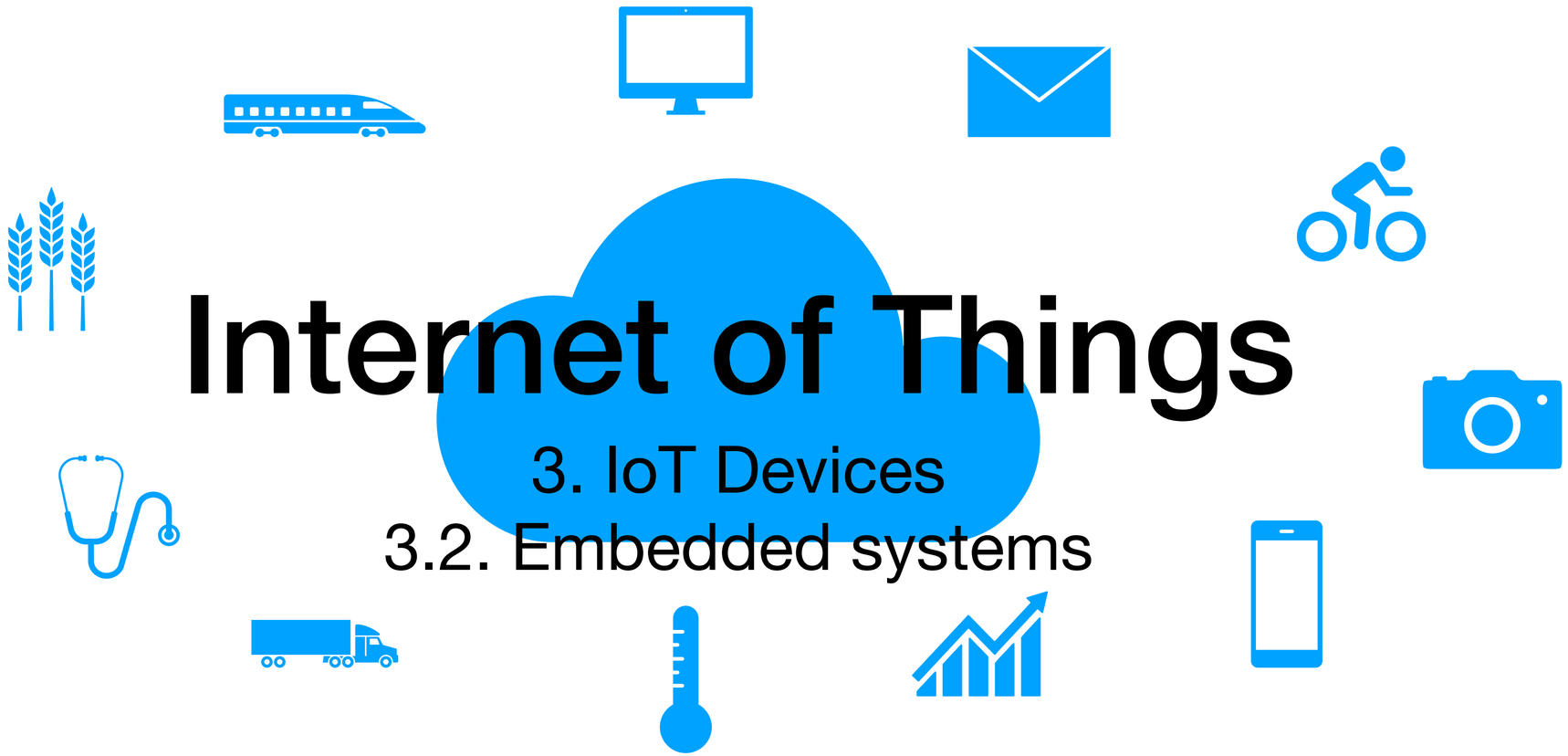


Internet of Things

3. IoT Devices

3.2. Embedded systems

M.Sc. Computer Science 2024-2025

Viviana Arrigoni

Things

Fitbit trackers

Camera with proximity sensor

Aircraft engines (nowadays equipped with hundreds of sensors)

Philips Hue smart lightbulb

Nest thermostats

Things + ⚙ + 📶 = Embedded systems

Fitbit trackers

Camera with proximity sensor

Aircraft engines (nowadays equipped with hundreds of sensors)

Philips Hue smart lightbulb

Nest thermostats

Embedded systems are computerised systems that are purpose-built for their applications (highly specialised in one application and few tasks around the application).

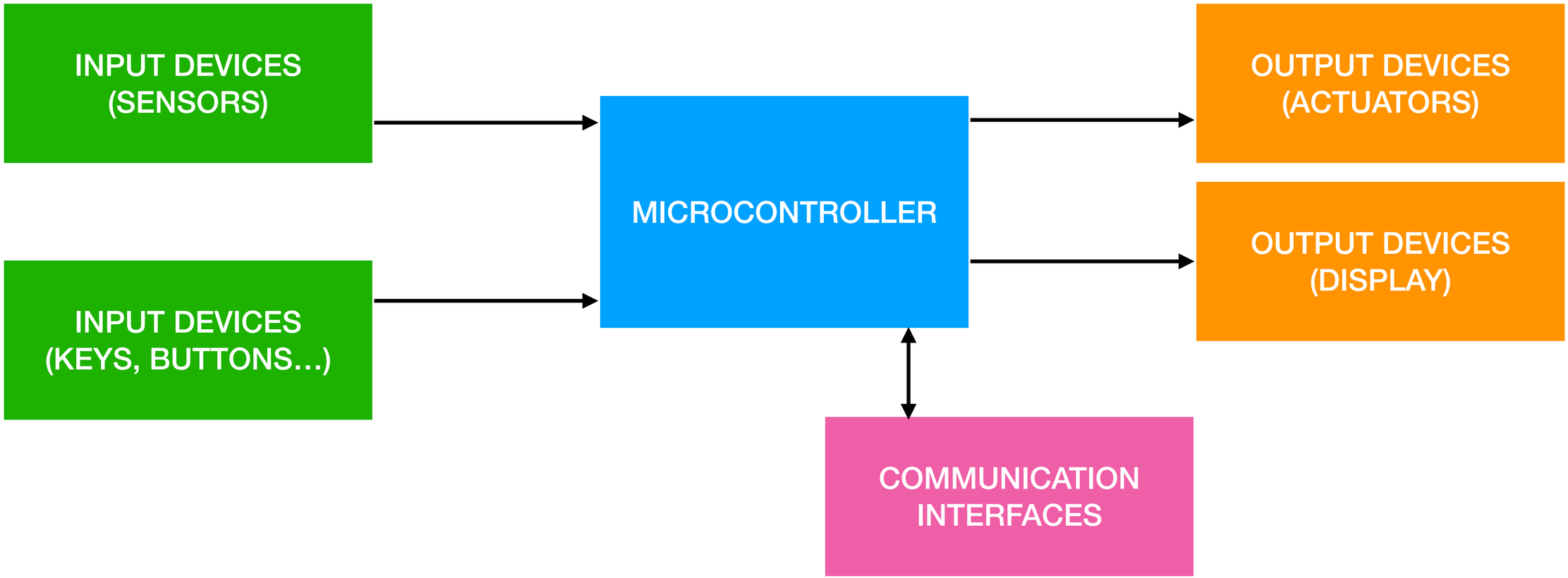
- While general purpose computers' main function is computation, that is not the same for IoT devices (e.g., smart cars, smart refrigerators, smart light bulbs).

Application centric approach (single purpose hardware/software)

- IoT devices are embedded systems with the additional feature that they can talk through the internet.

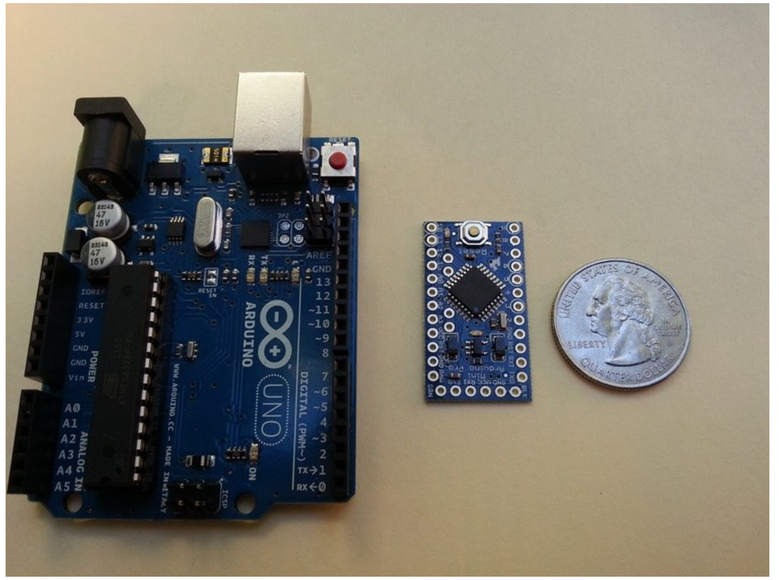
3.2.1. Embedded systems architecture

High level embedded system architecture



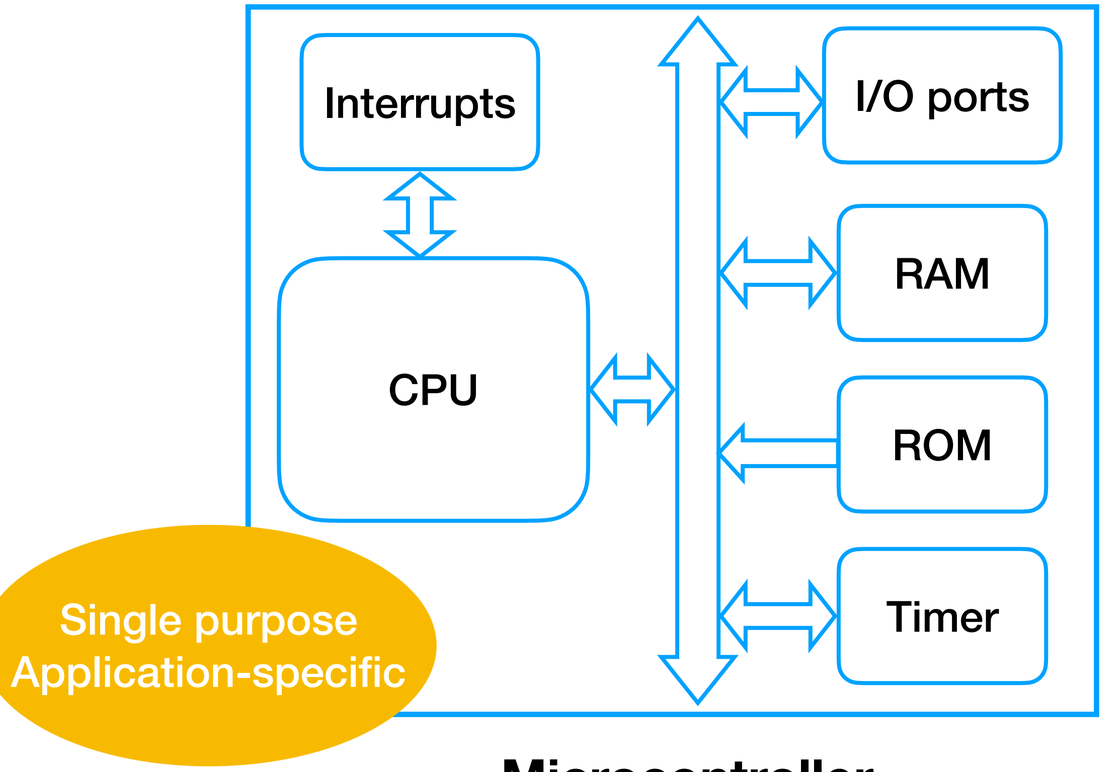
Microcontrollers & Microprocessors (1)

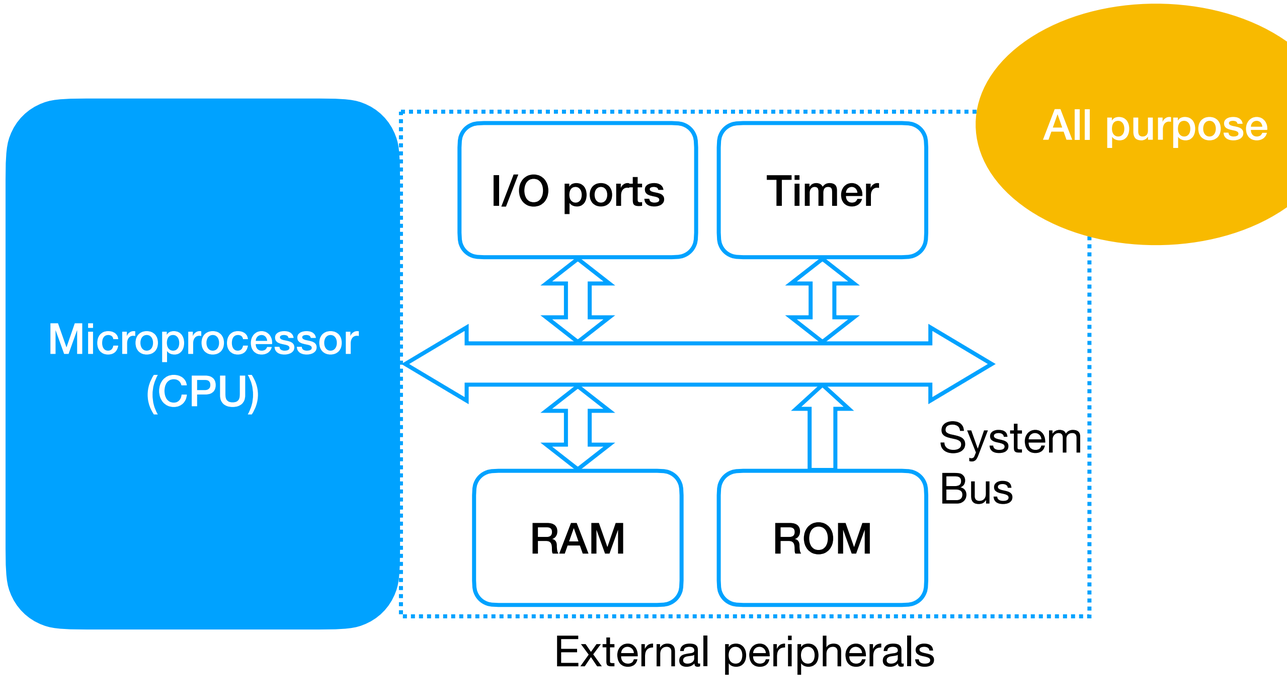
- Microcontrollers (MCU) are the brain of an embedded system and responsible for orchestrating all the operations.
  - Slower than microprocessors (16MHz-500MHz clock frequency), with less memory and fewer features.
- Microprocessors are mainly general-purpose systems (e.g., computers), whereas microcontrollers are designed to perform very few tasks.
- A typical microcontroller contains a CPU, interrupts, timer/counter, memory and other peripherals, all in a single integrated circuit.



Arduino UNO and Arduino Pro Mini (MCUs)

Microcontrollers & Microprocessors (2)





Microcontroller

- All components in a single chip
- Internal RAM/ROM integrated
- Lower power consumption, processing power and cost

Microprocessor

- Only CPU, needs external memories and peripherals
- Higher power consumption, processing power and cost

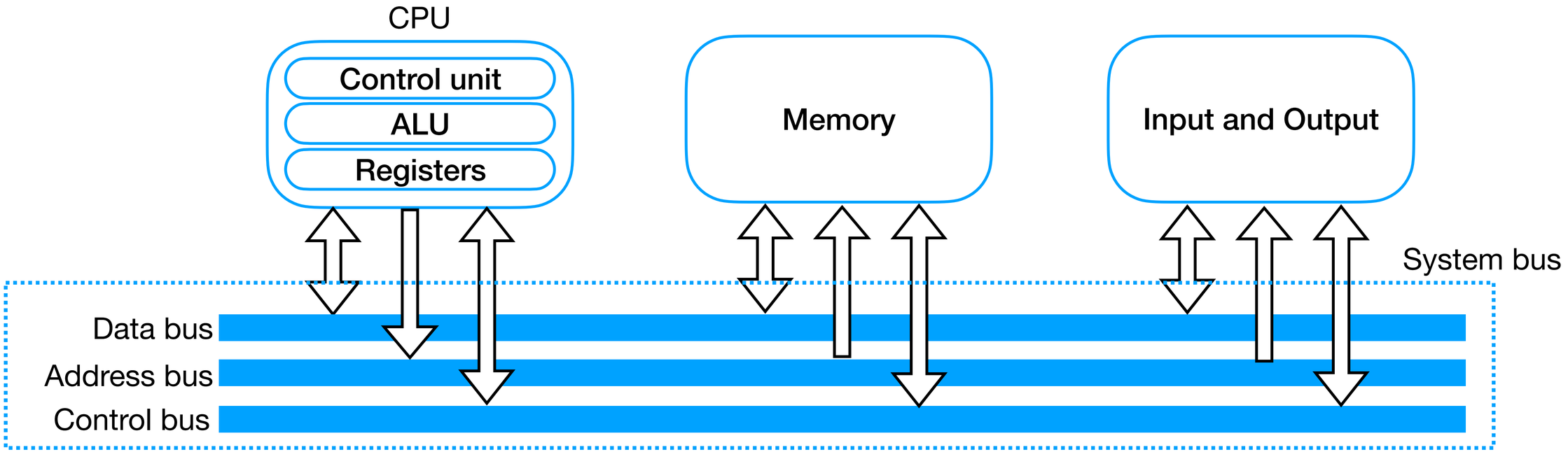
Microcontrollers & Systems on Chip (SoCs)

- Microcontrollers:
  - are the core processors executing code, have memory and on-chip peripherals.

- System-on-Chip (SoC):
  - combines a microcontroller core with more complex subsystems/ peripherals (e.g., digital signal processors) and can contain multiple CPUs.

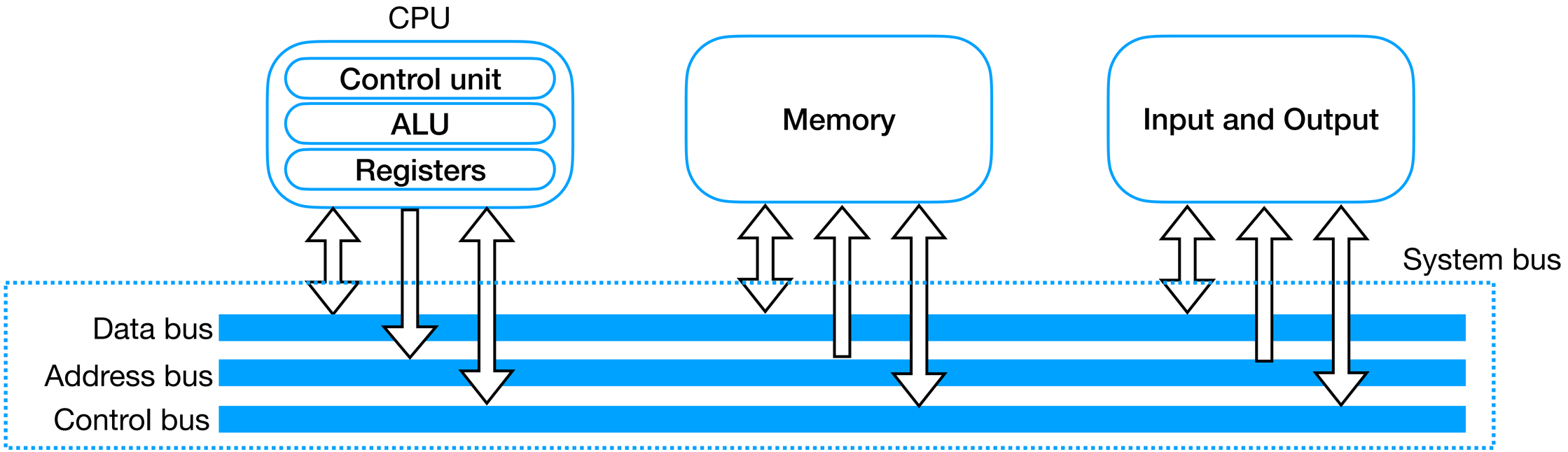
- Exact line between microcontrollers (MCU) and SoCs is blurry, but usually SoCs are more powerful, intended for higher complexity, and consume more power.

CPU

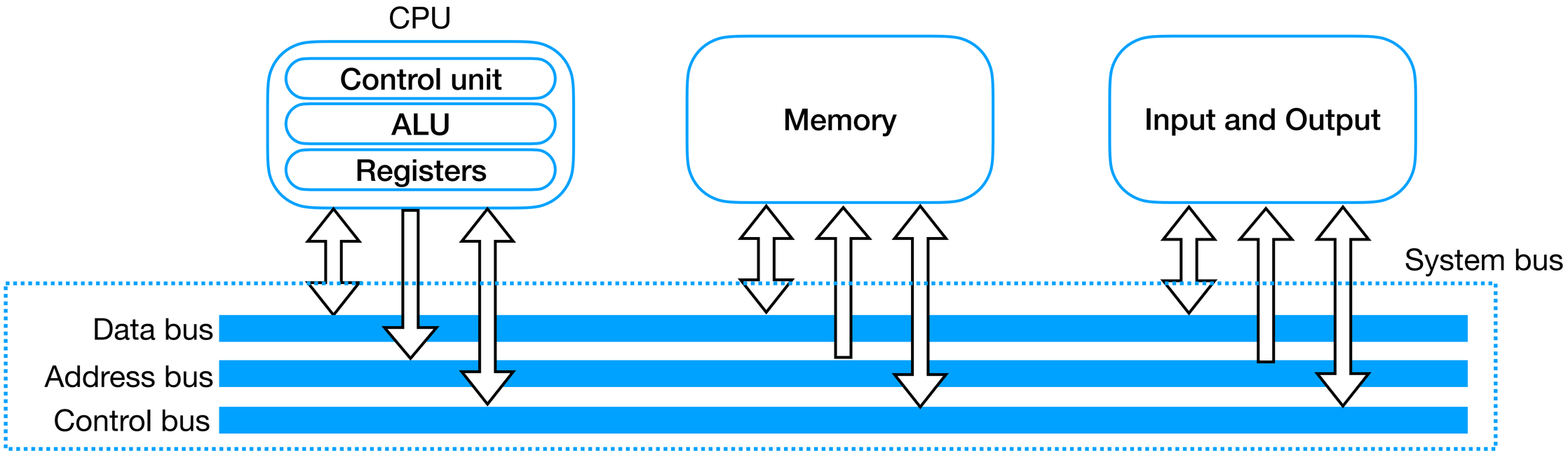


- CPU has three main components:
  - **Arithmetic central unit (ALU)**, performs arithmetic and logical operations
  - **Registers**, provide operands to ALU and store results of ALU operations
  - **Control Unit** controls the overall operations and communicates with ALU and registers

CPU

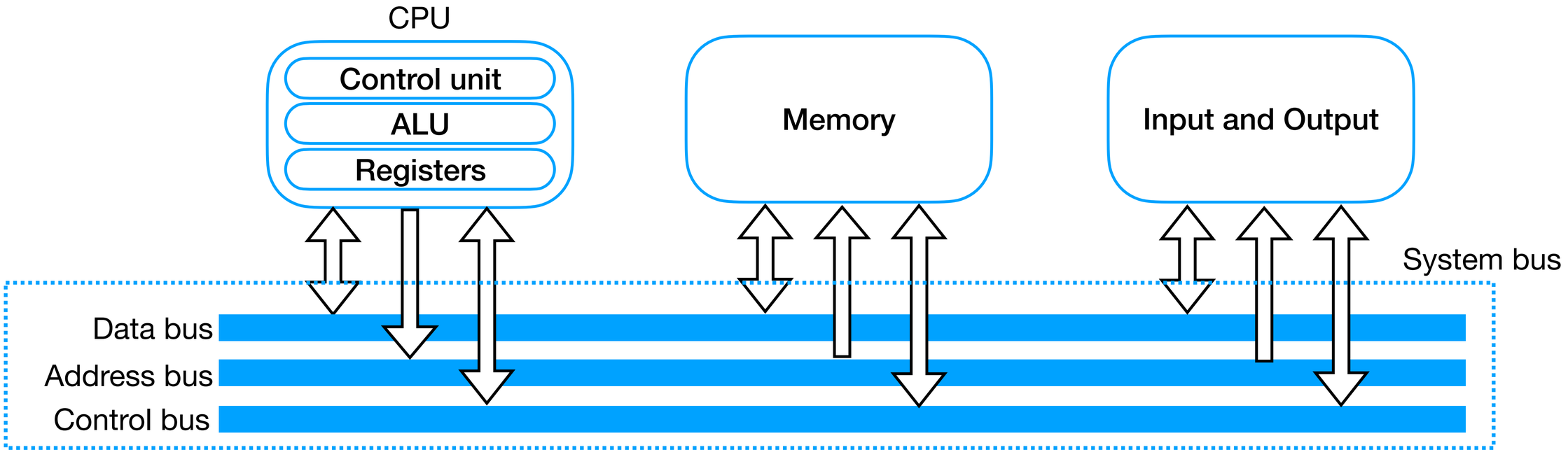

CPU
Control unit
ALU
Registers
Memory
Input and Output
System bus
Data bus
Address bus
Control bus

CPU

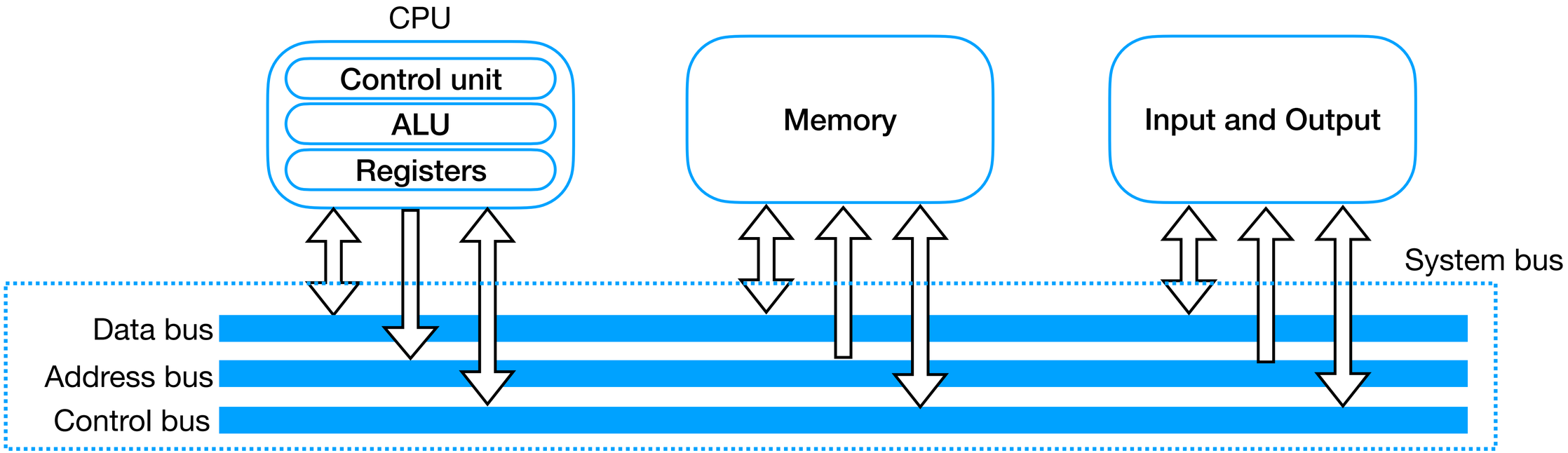


- CPU are characterised by their **instruction set architecture** (ISA), i.e., set of basic operations that CPU can perform:
  - **Complex Instruction set Computing (CISC)** has very large instruction sets (>300), more complex hardware, more compact software, takes more cycles per instruction.
  - **Reduced Instruction Set Computing (RISC) h**as small instruction sets (<100), simpler hardware, more complicated software, one instruction per cycle.

CPU


CPU
Control unit
ALU
Registers
Memory
Input and Output
System bus
Data bus
Address bus
Control bus

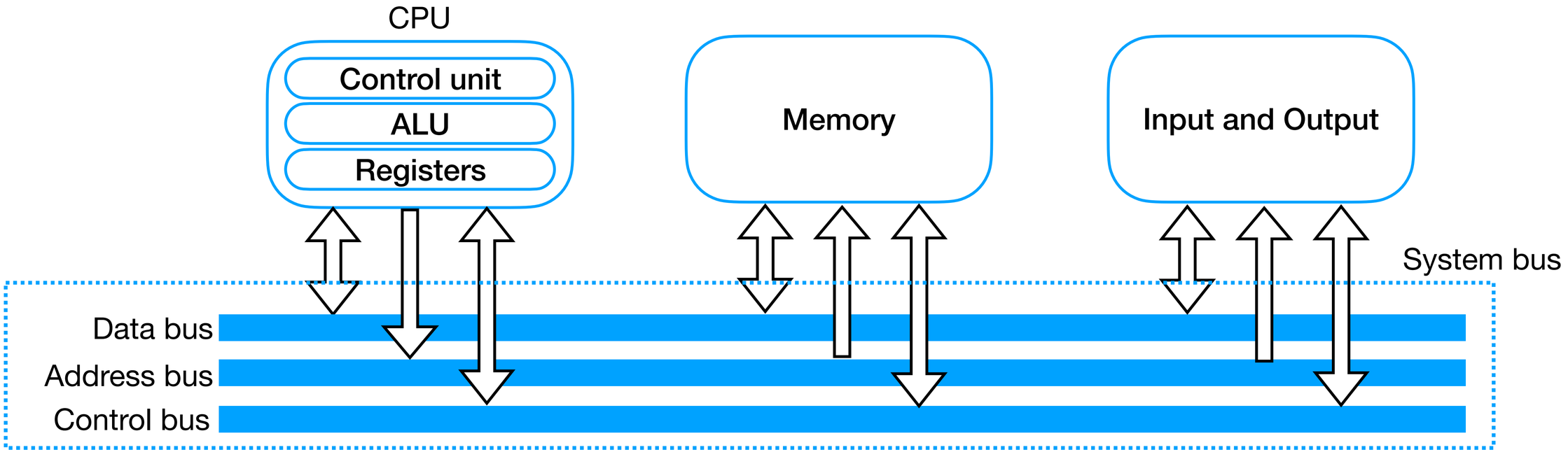
CPU



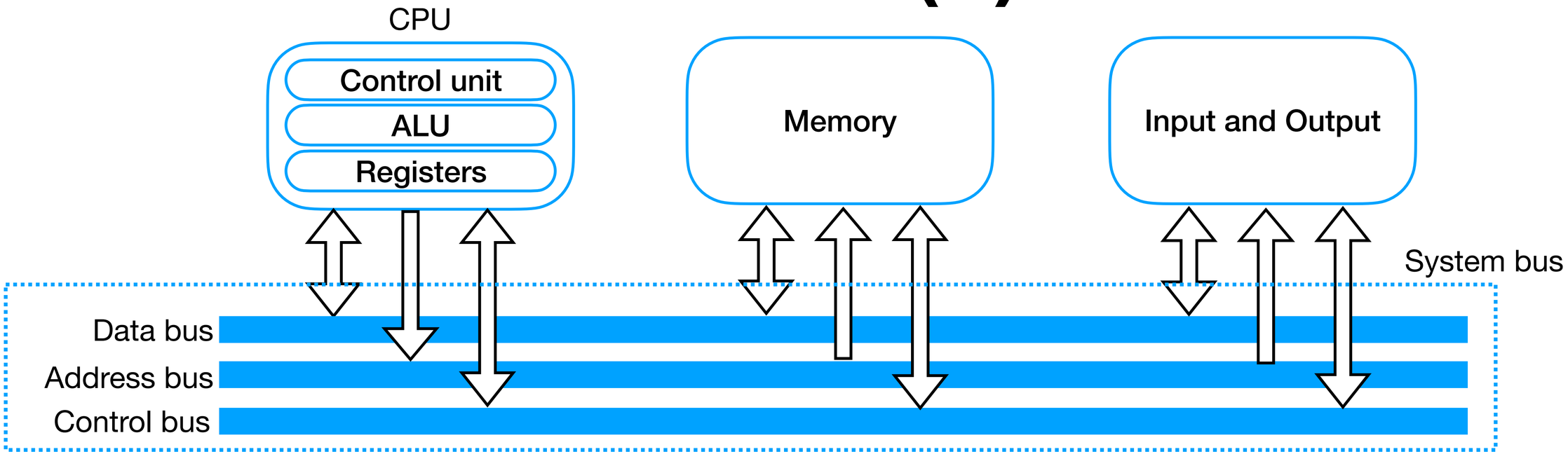
- Typical examples of CISC CPUs are AMD and Intel x86, mainly used on computers, workstations and servers.

- Typical examples of RISC CPUs are Atmel AVR, PIC and ARM, mainly used in microcontrollers

CPU

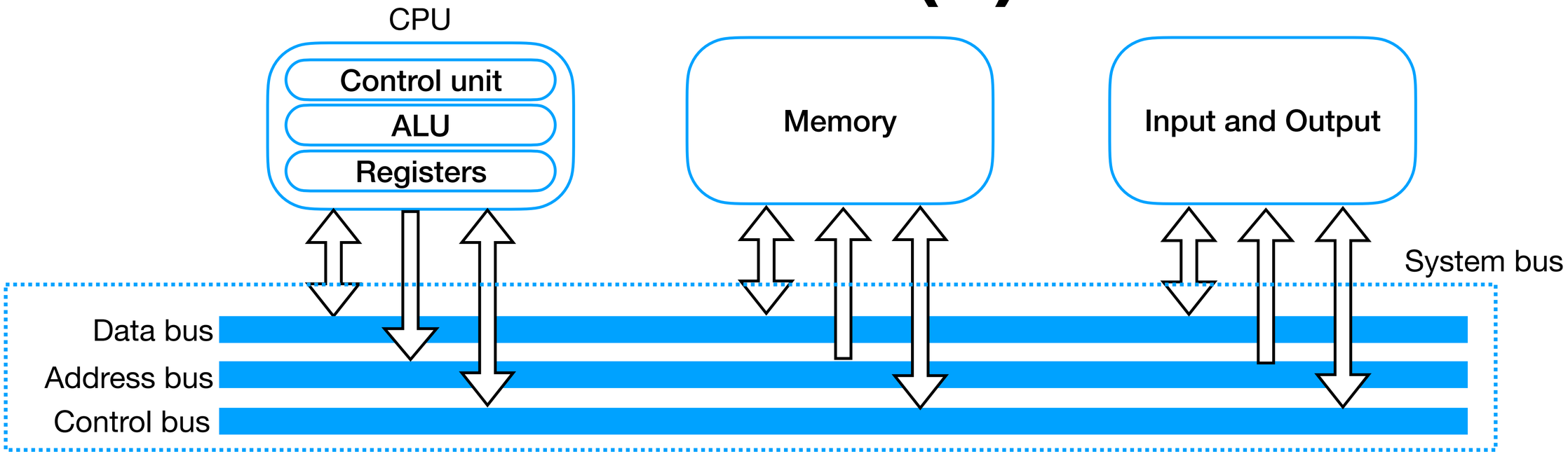

CPU
Control unit
ALU
Registers
Memory
Input and Output
System bus
Data bus
Address bus
Control bus

BUSes (1)



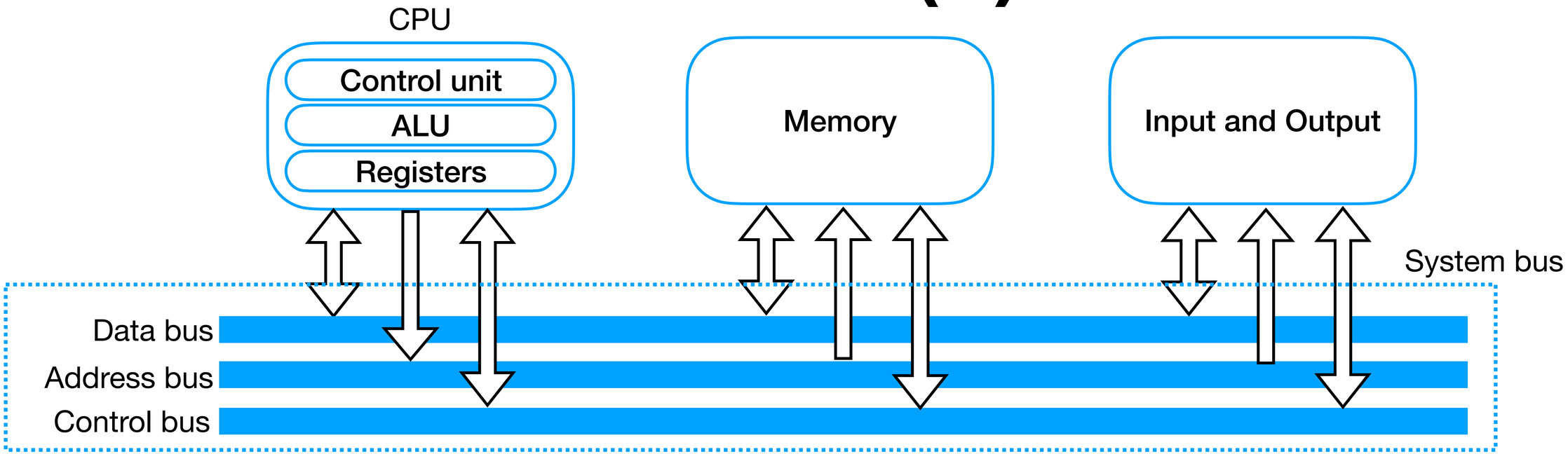
- CPU communicates with external peripherals (such as memory and I/O) through the system bus, which includes:
  - data bus, for carrying information
  - address bus, for determining where the information should be sent
  - control bus, for controlling the operation

BUSes (1)



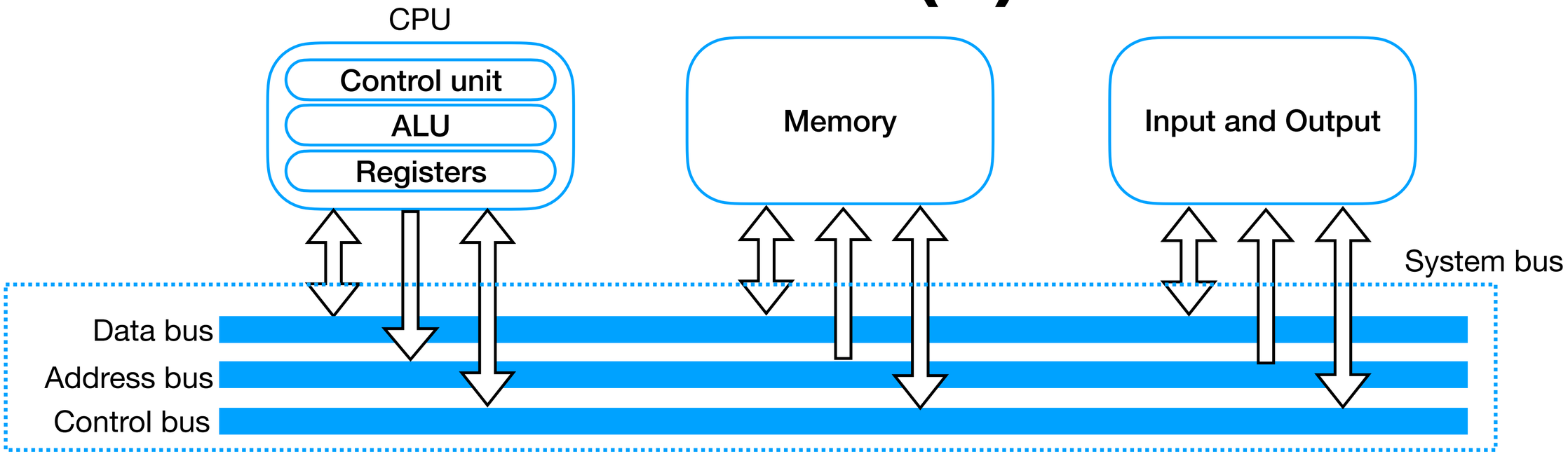
- The data bus carries the actual data being transferred between the processor, memory, and peripherals.
- Bi-directional connection
- The width of the data bus (e.g., 8-bit, 16-bit, 32-bit) determines how much data can be transferred in one cycle

BUSes (1)



- The address bus carries information about the memory address or I/O device where the data should be read from or written to.
- Uni-directional communication
- The width of the address bus (e.g., 32-bit in a Cortex-M MCU) determines the maximum addressable memory space.

BUSes (1)

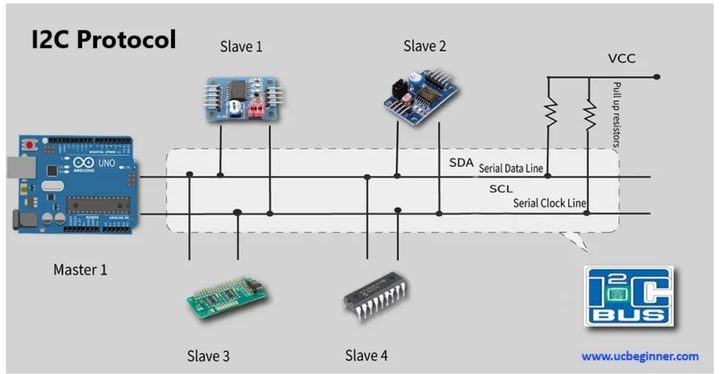


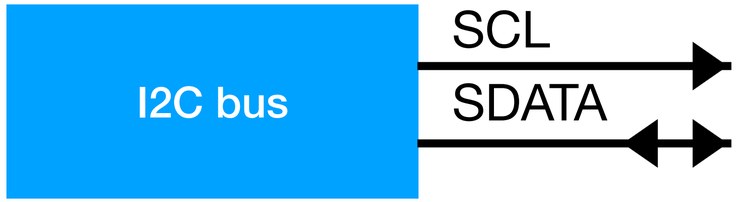
- The control bus carries control signals (commands) to coordinate the operations of the processor, memory, and peripherals.
- Some common control signals include: **Read** (RD): Instructs memory or I/O to send data to the processor. **Write** (WR): Instructs memory or I/O to store data from the processor. **Clock** (CLK): Synchronizes data transfers. **Interrupt** (IRQ): Indicates external events requiring CPU attention. **Chip Select** (CS): Enables a specific memory chip or peripheral.

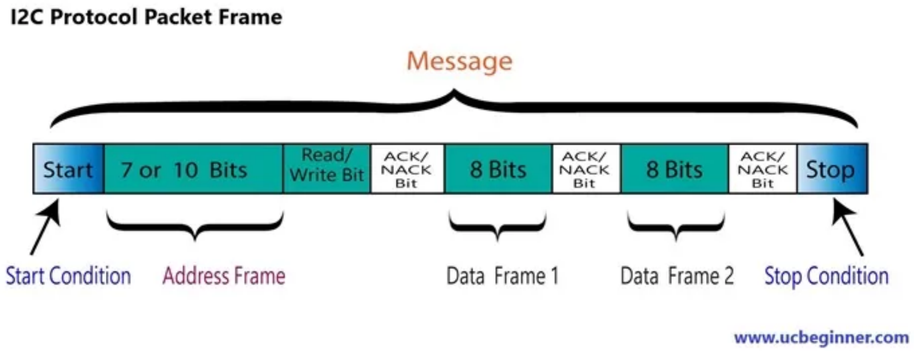
BUSes (2)

Inter-Integrated Circuit (I2C) bus

- The master sets the clock rate and sends it through the serial clock line (SCL). The data (SDATA) line is used for both the microcontroller to send data to external chips and viceversa (half-duplex transmission mode).
- It offers a direct and efficient way for multiple devices to communicate over a shared bus, without the need for individual communication lines per device.
- Accommodates a broad spectrum of devices, ranging from sensors to memory chips.





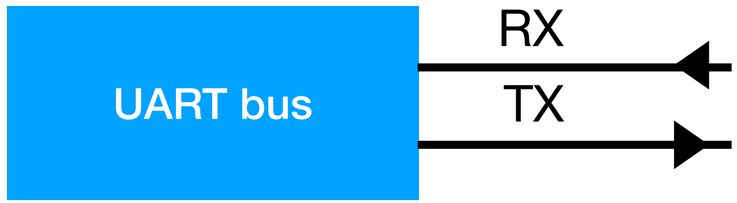


Each message consists of an address frame holding the binary slave address and one or more data frames with the transmitted data

BUSes (3)

Universal Asynchronous Receiver-Transmitter (UART) bus

- It is a bus connecting the CPU with UART ports/peripherals.
- No clock line. One line for transmitting and one for receiving messages.
- Allows for asynchronous communication, very versatile.
- Transmission is serial, bit by bit.



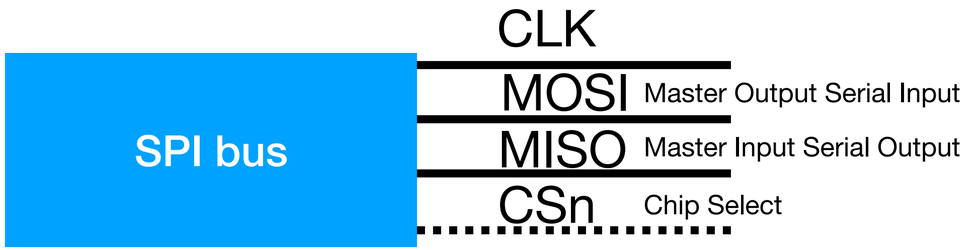
Data packet

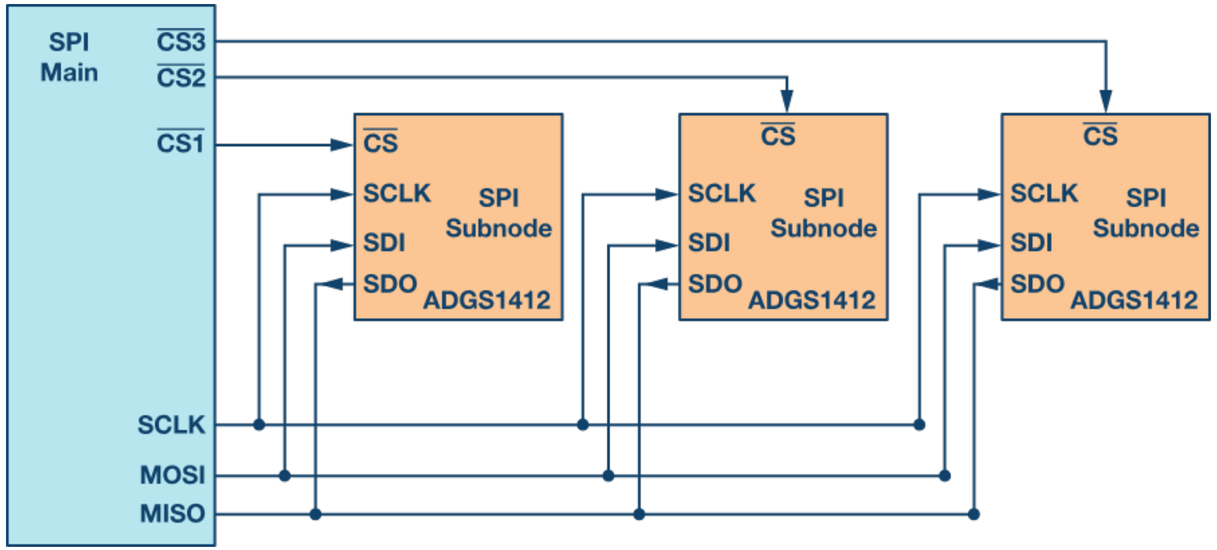
Start bit (1 bit)	Data Frame (5 to 9 Data Bits)	parity bits (0 to 1 bit)	Stop bits (1 to 2 bits)

BUSes (4)

Serial Peripheral Interface (SPI)

- Allows for synchronous, full duplex master-slave-based communication.
- Has a clock line, two lines for transmission (master starts the communication).
- There are n chip select lines, one for each connected peripheral.
- The master selects the slave by pulling the corresponding CS line low (0 V).





Memory

- In addition to classical external memories (RAM and ROM), microcontrollers are also equipped with:
  - **EEPROM** (Electrically Erasable Programmable Read-Only Memory), is a user-alterable read-only memory that can be erased and reprogrammed.
    - stores information in memory cells that use floating gate transistors to store and retrieve data.
    - High voltage traps electrons in the floating gate to store data, and when data needs to be wiped, the charge from the floating gate is realised.
    - data is erased byte-by-byte.
  - **NVRAM** (non-volatile RAM) works as a SRAM when powered (i.e., retains its contents as long as electrical power is applied to the chip), and uses a battery to retain data when power is off.

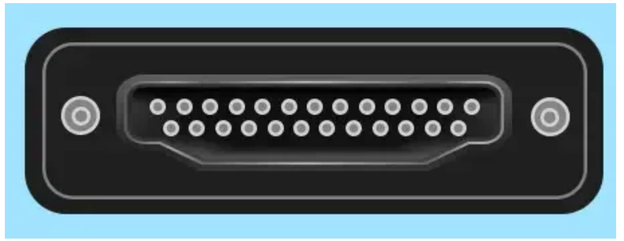
EEPROM vs NVRAM + flash memory

- **Flash memory** has a similar structure as EEPROMs but data is erased in blocks (256 bytes to few KB).

	Volatile	Writable	Erase size	Max erase cycles	Cost/byte	Speed
EEPROM	No	Yes	Byte	Limited	Expensive	Fast to read slow to erase/write
Flash memory	No	Yes	Blocks	Limited	Moderate	Fast to read slow to erase/write
NVRAM	No	Yes	Byte	Unlimited	Expensive	Fast

Parallel I/O ports

- Can move a set of 8 bits at a time on eight different wires.



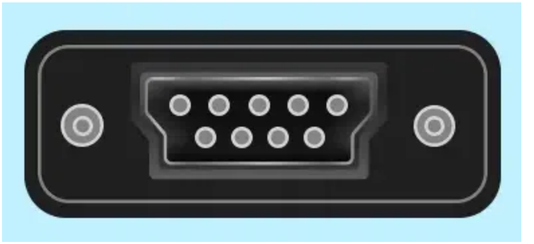
DB-25 connector

- Have multiple wires/pins running parallel to each other.
  - multiple signals can be accessed all at once.
  - high transmission rate

- To eliminate **cross talk** and **errors**, all the bitstreams need to transfer data at the same time speed in a parallel communication (impractical).
  - solution: use these lines for short communication
  - examples: monitors, printers, memories

Serial I/O ports

- Use a single data wire and only a single stream of data is transferred.
  - no cross talk, no errors due to data speed mismatch
  - can be used to communicate long distances (you can attach long cables to it)



DB-9 connector

- Universal Asynchronous Receiver/Transmitter (UART) peripheral is a commonly used serial input/output port in embedded systems.
  - kinda deprecated, replaced with faster ports as Ethernet and USB in computers but still used in embedded systems.

- Transmission speed is lower than parallel I/O ports'.

Analog to Digital Converter (ADC)

- ADC converts analog (continuous) signals to digital (discrete) signals.

- Used for **reading the analog output of sensors**, such as
  - sound picked up by a microphone
  - light entering a digital camera
  - voltage or current

- Performance of an ADC is primarily characterised by its bandwidth and its signal-to-noise ratio (SNR).
  - bandwidth is essentially the sampling rate.
  - SNR is characterised by several factors, including resolution, linearity and accuracy (how well the quantisation levels match the true analog signal).

Digital to Analog Converter (DAC)

- DAC converts digital (discrete) signals to analog (continuous) signals.

- It is used for controlling analog devices by translating a fixed-point binary number into a physical quantity (e.g., voltage, pressure). For example:
  - controlling audio speakers,
  - motors.

- FACT: ACDs and DACs can convert one type of signal into the other nearly perfectly if the sampling rate is greater than twice the bandwidth of the signal (**Nyquist-Shannon sampling theorem**).

3.2.2. CortexM Microcontrollers

CortexM microcontrollers

- There are many CortexM microcontrollers, but they all use 32 bit microcontrollers.
- Many types: M0 (smallest, weaker), M0+, M1, M2 M4, M7, M23, M33 (most powerful).
- Three major subfamilies/architectures:
  ARMv6-M: cortex M0; cortex M0+, cortex M1
  ARMv7-M: Cortex M3
  ARMv7E-M: Cortex-M4, Cortex M7.
- CortexM series defines:
  instruction set that you can issue
  pieces of core processor functionalities that all CortexM microcontrollers have

CortexM Optional functionalities

- **Memory Protection Unit.**
  Used to protect firmware from applications.
  Lets the microcontroller operate in one of two modes: **user more** or **kernel mode**.

  - Memory protection unit can limit what memory the untrusted code (user code) can access, so to protect the trusted code (kernel code).

- **SysTick**, standard timer.

- Cortex-M supports all sort of extra features which are presented as peripherals, which appear as **blocks of memory** within the processor architecture.

An MCU: Atmel SAM4L series

- Cortex M4 core
- 128-512 KB flash memory (program), 32-64 KB RAM
- Operates at up to 48MHz at 1.68 to 3.6V
- 4 USART (UART/SPI) buses and 4 i2C buses - many buses to which you can attach many external chips and control them in parallel
- USB hardware support
- ADC (3-15 channels depending on model)
- 15 DMA (Direct memory access) channels for I/O processing offload

  This means that there is a **peripheral dedicated hardware module (DMA controller)** with its own registers, control logic, etc. that the CPU can set up and that handles data transfer between memory and peripherals without involving the CPU.
  - ✅ Supports multiple transfer type (M2M, M2P, P2M, P2P).
  - ✅ Improves real time performance (CPU can handle other tasks while transfer happens in background).
- 1.5-3 μA sleep, 1.5 μs wakeup, 90μA/MHz active (4.3mA @ 48MHz)

A SoC: Nordic nRF51 SoC

- Cortex-M0 core
- Integrated BLE transceiver
- 128-256 flash memory, 16-32 KB RAM
- Operates at 16MHz at 1.68 to 3.6 V
- 1 UART, 1 SPI, one i2bus
- ADC
- 2.6 μA sleep, 4.2μs wakeup, 2.4-4.1mA active, 16mA TX, 13.4mA RX

A SoC: Nordic nRF51 SoC

- Cortex-M0 core
- Integrated BLE transceiver
- 128-256 flash memory, 16-32 KB RAM
- at 1.68 to 3.6 V
- 2bus
-
- 2.6 μA sleep, 4.2μs wakeup, 2.4-4.1mA active, 16mA TX, 13.4mA RX

Consumes more than SAM4L although the processor core is slower (this is because BLE and supporting circuits)

A SoC: Nordic nRF51 SoC

- Cortex-M0 core
- Integrated BLE transceiver
- 128-256 flash memory, 16-32 KB RAM
- Operates at 16MHz at 1.68 to 3.6 V
- 1 UART, 1 SPI, one i2bus
- ADC
- 2.6 μA sleep, 4.2μs wakeup, 2.4-4.1mA active, 16mA TX, 13.4mA RX

Consumption for transmitting and receiving (i.e., for using the radio) is very high compared to computation

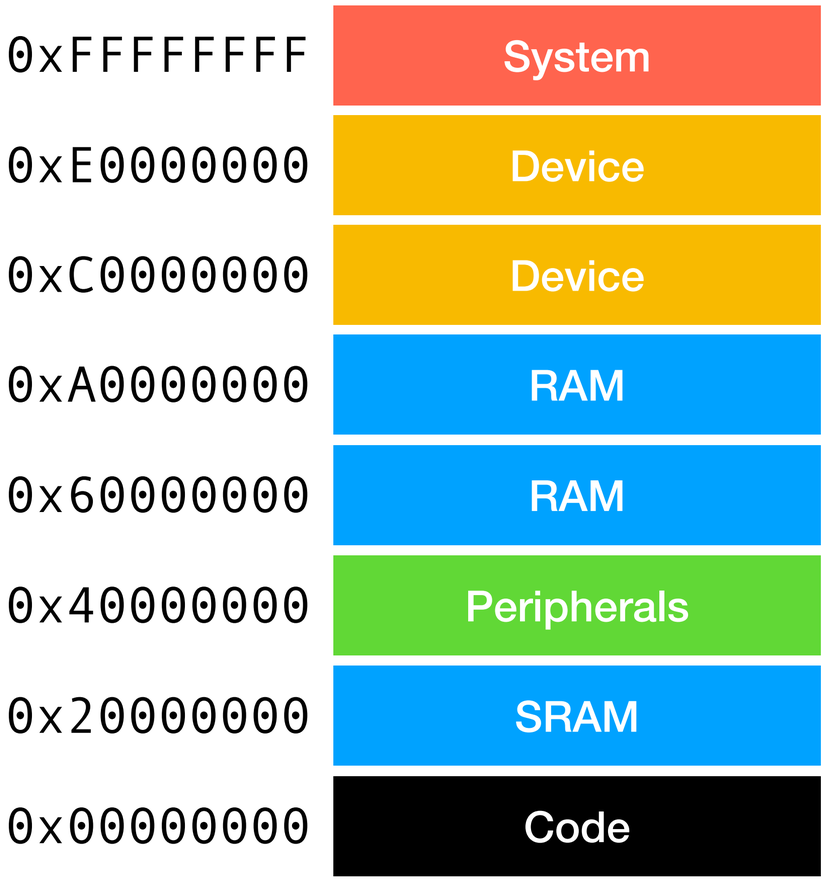
A SoC: Nordic nRF51 SoC

- Cortex-M0 core
- Integrated BLE transceiver
- 128-256 flash memory, 16-32 KB RAM
- Operates at 16MHz at 1.68 to 3.6 V
- 1 UART, 1 SPI, one i2bus
- ADC
- 2.6 μA sleep, 4.2μs wakeup, 2.4-4.1mA active, 16mA TX, 13.4mA RX

Consumption for transmitting and receiving (i.e., for using the radio) is very high compared to computation

Optimising for lifetime, power, energy means optimise for radio usage

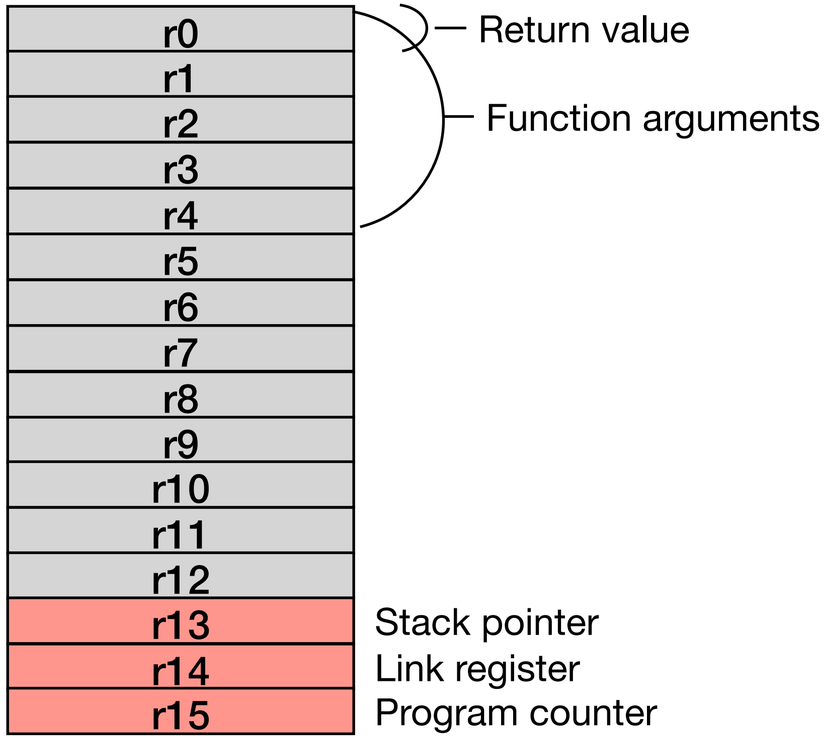
CortexM memory Map (ARMv6-M, ARMv7-M)



In CortexM microcontrollers, on chip features/components are modelled as memory-mapped peripherals.

Core Registers

- Let’s consider ARM architectures as an example.
- The CPU has 16 registers, r0, …, r15



The first registers are used for storing function arguments, and r0 is used to store the return value

REVIEW:
- stack pointer: contains the location of the last item put onto the stack, i.e., the memory region that stores local variables, function parameters
- Link register: stores the return address for a function call.
- Program counter: contains the memory address of the next program instruction to be executed

Running modes

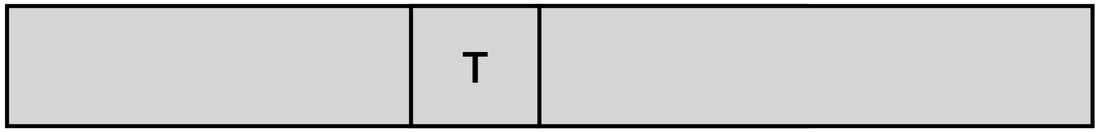
- MCU can run in two modes: **user** and **kernel**.
- In user mode, applications run with limited privileges to prevent direct access to hardware, ensuring system stability.
- In kernel mode, the operating system has unrestricted access to all hardware resources and can perform critical tasks such as memory management and process control.
- A MCU typically switches from user mode to kernel mode during events that require higher privilege, for instance:
  - **system calls**, i.e., when a user application needs to request services from the operating systems (e.g., access to hardware).
  - **interrupts**, e.g., a peripheral sends an interrupt request
  - **exceptions** like faults or error
  - **context switching**, e.g., in case of multithreading, switching to kernel mode allows to manage scheduling and resource allocation

Control Registers

- In addition to the register used for computation, r0,…,r15, there are **control registers,** important for interrupt control, switching the addressing mode, paging control…

APSR (application program status register), containing information about the status of the processor

IPSR (interrupt program status register), containing the exception number of the current interrupt



EPSR (execution program status register), containing the thumb state bit - used for when instructions that take multiple cycles are interrupted, so the processor can resume those instructions

See https://developer.arm.com/documentation/dui1095/a/The-Cortex-M23-Processor/Programmers-model/Core-registers for more details

Interrupts (1)

- A hardware **interrupt trigger** is an event that generates an **interrupt request** (**IRQ**) via an electrical signal to the controller

- When an interrupt is received, the MCU:
  - halts the execution of currently running tasks,
  - starts running in kernel mode and performs a list of commands associated with the interrupt (called **interrupt handler function** or **interrupt service routine - ISR**),
  - resumes normal operations after the interrupt is handled.

- Some embedded systems are predominantly controlled by interrupts - handling them is of key importance

Interrupt Controller and states

Interrupt Controller and states

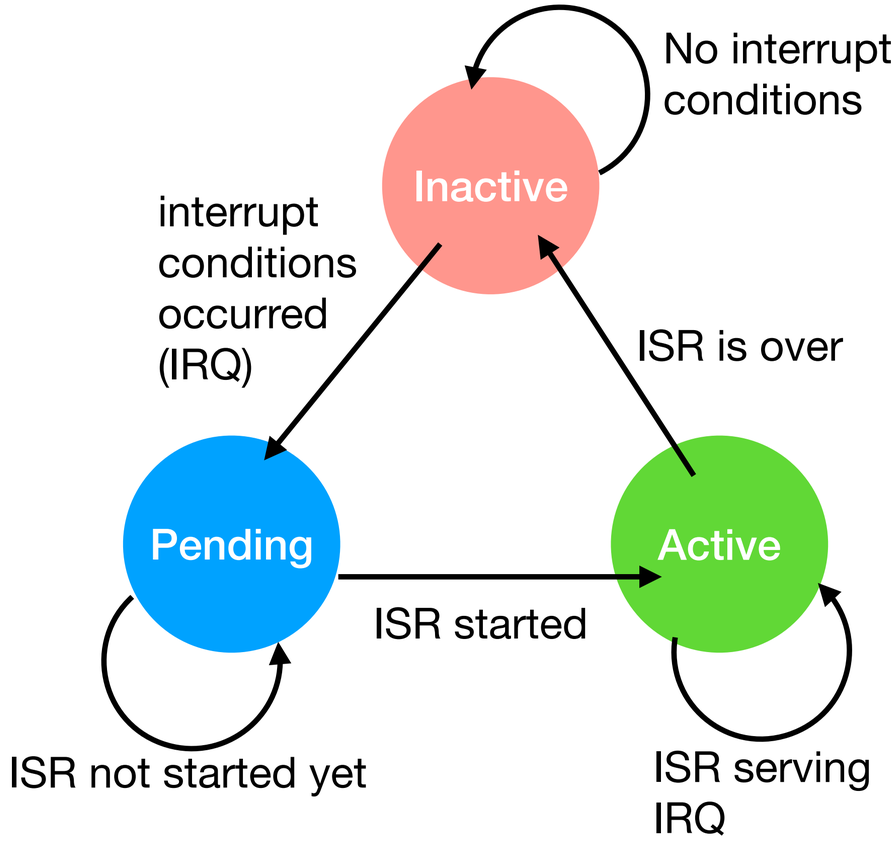
- The **interrupt controller** is a peripheral that helps the processor manage the interrupts.

Interrupt Controller and states

- The **interrupt controller** is a peripheral that helps the processor manage the interrupts.
- An **interrupt source** is any peripheral that can trigger an interrupt
  - ADC,
  - timer,
  - GPIO (General Purpose I/O),
  - UART.

Interrupt Controller and states

- The **interrupt controller** is a peripheral that helps the processor manage the interrupts.
- An **interrupt source** is any peripheral that can trigger an interrupt
  - ADC,
  - timer,
  - GPIO (General Purpose I/O),
  - UART.
- Interrupts have three states:
  - inactive, the conditions to generate the interrupt haven't been met.
  - pending, the conditions have been met, but the ISR has not been called.
  - active, the ISR is serving the interrupt



Interrupt Vector Table (IVT)

- Interrupt service routines have addresses called **interrupt vectors** which are stored in the **interrupt vector table** (**IVT**).
- The IVT matches each ISR with IRQs coming from different interrupt sources.
    - IVTs are usually stored in the flash memory and are different depending on the vendor or the microcontroller.
- Interrupts usually come with **priorities** to determine which interrupt to serve first if more than one interrupt is pending.

Types of Interrupts

- Interrupts can be of two types:

- **Hardware interrupts** occur when the interrupt request (IRQ) comes from an external device as an input to the microcontroller.
  - Hardware interrupts can happen **asynchronously** at any given point during the execution of the program.
  - example: A temperature sensor has been programmed such that when the temperature becomes greater than 30 degrees celsius, it should inform the CPU.

- **Software interrupts** are called upon by the microcontroller when it executes special instructions or certain conditions are met while executing the code.
  - exceptions and traps
  - example: divisions by zero

3.2.3. Costs & energy

Embedded systems costs

- When building an embedded system for IoT, the two main costs to keep in mind are **energy** and **money**, which have strong implications in the embedded system design.
- Energy:
  - embedded system have an expected workload - want to minimise energy required to handle the workload (longer battery life, more efficient)
  - More powerful highly featured MCPs draw more power - can be more *power efficient*, but consume more energy for a given task.
- Money:
  - Many embedded systems makers have tight margins (e.g., cars)
  - More powerful, highly featured MCUs cost more
  - want to use devices that do just exactly what they have to, especially when putting hundreds of them in a particular device.

Embedded systems costs

- When building an embedded system for IoT, the two main costs to keep in mind are **energy** and **money**, which have strong implications in the embedded system design.
- Energy:
  - embedded system h… want to minimise energy required to handle the… more efficient)
  - More powerful highly… power - can be more *power efficient*, but co… en task.
- Money:
  - Many embedded systems makers have tight margins (e.g., cars)
  - More powerful, highly featured MCUs cost more
  - want to use devices that do just exactly what they have to, especially when putting hundreds of them in a particular device.

TAKEAWAYS:
1- pick the minimal MCU that can meet your application requirements
2- optimise the code for it

Cost examples

Part	Family	Flash	RAM	Cost	Notes
ATSAMD20E15A	Cortex-M0+	16kB	2kB	$1.37	
ATSAMD21E16B	Cortex-M0+	64kB	8kB	$1.70	LIN, USB
NRF51422	Cortex-M0	256kB	32kB	$2.44	BLE
ATSAMD20J18A	Cortex-M0+	256kB	32kB	$2.69	20 12-bit ADC
ATSAM4S2BA	Cortex-M4	128kB	64kB	$2.80	SSC, USB
ATSAMD21G18A	Cortex-M0+	256kB	32kB	$3.15	LIN, USB
ATSAM4E8EA	Cortex-M4	512kB	128kB	$7.62	CAN, Ethernet, USB, IrDA

- Generally speaking, as flash memory and RAM go up, so does cost.

Computing Energy Budget

- Many embedded systems are battery-powered and so you want to minimise the amount of energy consumed
  - make a device last longer
  - requiring fewer recharges

- **Computing the energy budget allows you to reason about the design trade-offs.**
  - is it worth to make the batter slightly larger on your fitness tracker?
  - Can we optimise software to achieve less recharges?

High level energy consumption

- Roughly speaking, the energy consumed by a device is the energy consumed while they are sleeping and the energy consumed while they are active

$$E = P_s \cdot t_s + P_a \cdot t_a$$

E : energy
$P_{s/a}$: power in sleep/active more
$t_{s/a}$: time in sleep/active mode

- In practice it is more complicated, because there could be many terms and many different active modes (radio on/off, processor speed, etc).

Energy budget: example

- Consider a NRF51422 SoC that wakes up at 1Hz and transmits a single BLE advertisement.
  - $P_s = 4.8\mu$A
  - $P_a = 14.6$mA
  - time for sending advertisement: 0.3ms

$$E = P_s \cdot t_s + P_a \cdot t_a$$

Energy budget: example

- Consider a NRF51422 SoC that wakes up at 1Hz and transmits a single BLE advertisement.
  - $P_s = 4.8\mu$A
  - $P_a = 14.6$mA
  - time for sending advertisement: 0.3ms

$$E = P_s \cdot t_s + P_a \cdot t_a$$
$$= 4.8\mu A \cdot 999.7ms + 14.6mA \cdot 0.3ms$$

Energy budget: example

- Consider a NRF51422 SoC that wakes up at 1Hz and transmits a single BLE advertisement.
  - $P_s = 4.8\mu$A
  - $P_a = 14.6$mA
  - time for sending advertisement: 0.3ms

$$
\begin{aligned}
E &= P_s \cdot t_s + P_a \cdot t_a \\
&= 4.8\mu A \cdot 999.7ms + 14.6mA \cdot 0.3ms \\
&= (4798.56\mu A + 4.38mA) \cdot s
\end{aligned}
$$

Energy budget: example

- Consider a NRF51422 SoC that wakes up at 1Hz and transmits a single BLE advertisement.
  - $P_s = 4.8\mu$A
  - $P_a = 14.6$mA
  - time for sending advertisement: 0.3ms

$$
\begin{aligned}
E &= P_s \cdot t_s + P_a \cdot t_a \\
&= 4.8\mu A \cdot 999.7ms + 14.6mA \cdot 0.3ms \\
&= (4798.56\mu A + 4.38mA) \cdot s \\
&= (4.799mA + 4.38mA) \cdot s = 9.179mA \cdot s
\end{aligned}
$$

Energy budget: example

- Consider a NRF51422 SoC that wakes up at 1Hz and transmits a single BLE advertisement.
  - $P_s = 4.8\mu$A
  - $P_a = 14.6$mA
  - time for sending advertisement: 0.3ms

$$E = P_s \cdot t_s + P_a \cdot t_a$$
$$= 4.8\mu A \cdot 999.7ms + 14.6mA \cdot 0.3ms$$
$$= (4798.56\mu A + 4.38mA) \cdot s$$
$$= (4.799mA + 4.38mA) \cdot s = 9.179mA \cdot s$$

The radio is active 0.03% of the time but consumes as much as it does the 99.97% of the time when asleep

Energy budget: complication

- We assumed that system instantaneously transitioned from asleep to wake status.
  - MCU takes time to wake up
  - transceiver takes time to power up

- Transition times - high power but no work - can be significant
  - solution: wake up less often to amortise over wake periods

$$E = P_s \cdot t_s + P_a \cdot t_a + P_T \cdot t_T$$

P_T: power during transition

t_T : time to transition

Energy budget: example ++

- NRF51422 SoC waking up at 1Hz and transmits a single BLE advertisement
  - $P_s = 4.8\mu$A = 0.0048mA
  - $P_a = 14.6$mA
  - time for sending advertisement: 0.3ms
  - $P_T = 7.0$mA
  - $t_T = 0.14$ms

$$E = P_s \cdot t_s + P_a \cdot t_a + P_T \cdot t_T$$

Energy budget: example ++

- NRF51422 SoC waking up at 1Hz and transmits a single BLE advertisement
  - $P_s = 4.8\mu$A = 0.0048mA
  - $P_a = 14.6$mA
  - time for sending advertisement: 0.3ms
  - $P_T = 7.0$mA
  - $t_T = 0.14$ms

$$E = P_s \cdot t_s + P_a \cdot t_a + P_T \cdot t_T$$
$$= 0.0048mA \cdot 0.99956s + 14.6mA \cdot 0.0003s + 7.0mA \cdot 0.00014s$$

Energy budget: example ++

- NRF51422 SoC waking up at 1Hz and transmits a single BLE advertisement
  - $P_s = 4.8\mu$A = 0.0048mA
  - $P_a = 14.6$mA
  - time for sending advertisement: 0.3ms
  - $P_T = 7.0$mA
  - $t_T = 0.14$ms

$$
\begin{aligned}
E &= P_s \cdot t_s + P_a \cdot t_a + P_T \cdot t_T \\
&= 0.0048mA \cdot 0.99956s + 14.6mA \cdot 0.0003s + 7.0mA \cdot 0.00014s \\
&= (0.004798mA + 0.00438mA + 0.00098mA) \cdot s
\end{aligned}
$$

Energy budget: example ++

- NRF51422 SoC waking up at 1Hz and transmits a single BLE advertisement
  - $P_s = 4.8\mu$A = 0.0048mA
  - $P_a = 14.6$mA
  - time for sending advertisement: 0.3ms
  - $P_T = 7.0$mA
  - $t_T = 0.14$ms

Roughly 10% of energy goes to transitioning from sleep to active mode

$$E = P_s \cdot t_s + P_a \cdot t_a + P_T \cdot t_T$$
$$= 0.0048mA \cdot 0.99956s + 14.6mA \cdot 0.0003s + 7.0mA \cdot 0.00014s$$
$$= (0.004798mA + 0.00438mA + \boxed{0.00098mA}) \cdot s$$

Minimising energy consumption

- To minimise **sleep energy**, put microcontroller into lowest possible state
  - in the previous example we only saw “sleep” mode, but microcontrollers have different many low-power states and power-saving features.
  - the state the microcontroller is in has complex implications to software

- To minimise **active energy**, minimise time peripherals and MCU are active
  - perform operations in parallel to minimise active time
  - cluster/batch operations to minimise transition times (e.g., instead of sending one packet every second you can send 10 packets every ten seconds).
  - minimize clock rate.

Sleep states: SAM4L running modes

Mode	Current	Wakeup Latency

Sleep states: SAM4L running modes

- Four basic running modes:

Mode	Current	Wakeup Latency

Sleep states: SAM4L running modes

- Four basic running modes:
  1. RUN - MCU executes instructions, everything can be active

Mode	Current	Wakeup Latency
Run @48MHz	14.5 mA	-

Sleep states: SAM4L running modes

- Four basic running modes:
  1. RUN - MCU executes instructions, everything can be active
  2. SLEEP - MCU runs no instructions, clock/peripherals can be active

Mode	Current	Wakeup Latency
Run @48MHz	14.5 mA	-
Sleep	50μA	0.25μs

Sleep states: SAM4L running modes

- Four basic running modes:
  1. RUN - MCU executes instructions, everything can be active
  2. SLEEP - MCU runs no instructions, clock/peripherals can be active
  3. WAIT - no instructions, only the 32KHz clock active for peripherals (slowest rate clock, lowest power)

Mode	Current	Wakeup Latency
Run @48MHz	14.5 mA	-
Sleep	50μA	0.25μs
Wait	6μA	1.5μs

Sleep states: SAM4L running modes

- Four basic running modes:
  1. RUN - MCU executes instructions, everything can be active
  2. SLEEP - MCU runs no instructions, clock/peripherals can be active
  3. WAIT - no instructions, only the 32KHz clock active for peripherals (slowest rate clock, lowest power)
  4. RETENTION - no instruction, only 32KHz clock, no active peripherals (can be waken by external interrupts)

Mode	Current	Wakeup Latency
Run @48MHz	14.5 mA	-
Sleep	50μA	0.25μs
Wait	6μA	1.5μs
Retention	3μA	1.5μs

Active state: parallelism

- Parallelism is one technique to enhance efficiency in active mode

Sequential

```
loop {
    sample_sensor();
    radio_on();
    send_value();
    sleep();
}
```

Time: sum of time of each operation

Parallel

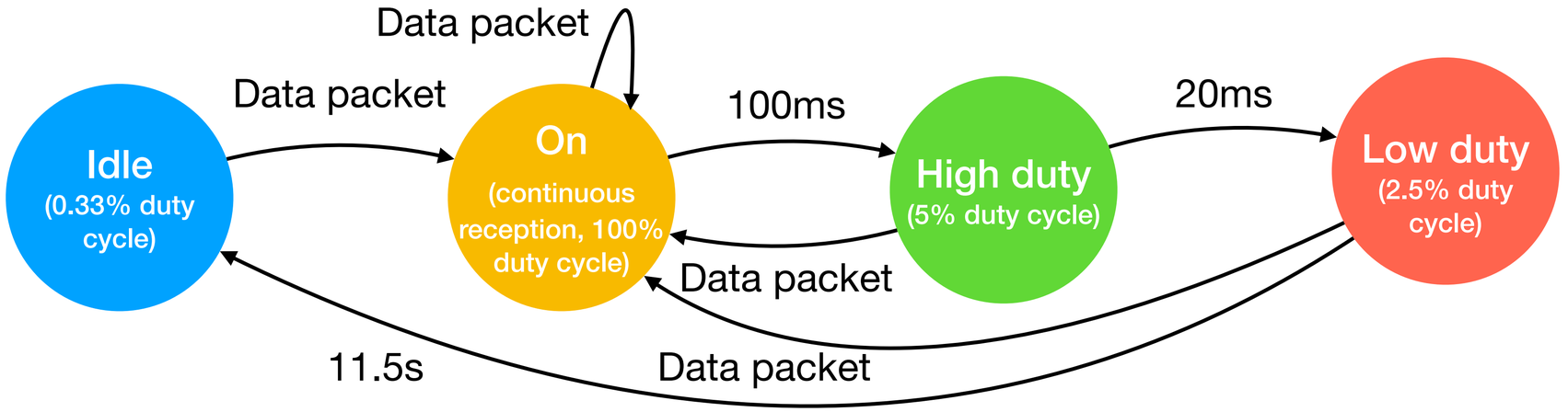
```
loop {
    parallel {
        sample_sensor();
        radio_on();
    }
    send_value();
    sleep();
}
```

Time: max(time for sample sensors and turning radio on) + time for sending values and enter sleep mode

Active state: Batching (1)

- If your system has very high transition costs (probably because it takes long to go from sleep to active mode), you can improve the efficiency of your system by **batching a whole bunch of operations** to amortise transition costs between them.

- Example: IoT system with LTE (Long Term Evolution) radio - often go through a variety of states depending on the network traffic they experience.



Duty cycle: % of time over a given period when the device is active

Active state: Batching (2)

- Assume to have an IoT device with an LTE radio. Assume time for sending a small packet is 5ms, i.e., 0.005s. The devices senses every 30 seconds and can decide whether to transmit the packet with data every 30s or transmit two packets every 60s.

- Sending one packet every 30 s:

$$0.105s \cdot 100\% + 0.02s \cdot 5\% + 11.5s \cdot 2.5\% + (30 - 11.625)s \cdot 0.33\,\%$$
$$\sim (0.105 + 0.001 + 0.2875 + 0.06)s/30s = 0.4535s/30s \sim 1.5\,\%$$

- Sending two packets every 60 s:

$$\sim 0.110s \cdot 100\% + 0.02s \cdot 5\% + 11.5s \cdot 2.5\% + (60 - 11.63)s \cdot 0.33\,\%$$
$$\sim (0.11 + 0.001 + 0.2875 + 0.16)s/60s = 0.5585s/60s \sim 0.93\,\%$$

Active state: Batching (2)

- Assume to have an IoT device with an LTE radio. Assume time for sending a small packet is 5ms, i.e., 0.005s. The devices senses every 30 seconds and can decide whether to transmit the packet with data every 30s or transmit two packets every 60s.

- Sending one packet every 30 s:

$$0.105s \cdot 100\% + 0.02s \cdot 5\% + 11.5s \cdot 2.5\% + (30 - 11.625)s \cdot 0.33\,\%$$
$$\sim (0.105 + 0.001 + 0.2875 + 0.06)s/30s = 0.4535s/30s \sim 1.5\,\%$$

- Sending two packets every 60 s:

$$\sim 0.110s \cdot 100\% + 0.02s \cdot 5\% + 11.5s \cdot 2.5\% + (60 - 11.63)s \cdot 0.33\,\%$$
$$\sim (0.11 + 0.001 + 0.2875 + 0.16)s/60s = 0.5585s/60s \sim 0.93\,\%$$

About 38% less time spent active

3.3.4. Energy Harvesting

Electrical Power sources

Batteries


Coin cell battery


Coin cell battery (with leads)

AA battery


AA battery holder

C battery


9V battery snap

9V battery


9V battery holder


USB battery

Plugging in to the Grid

9VDC 2.1mm wall adapter

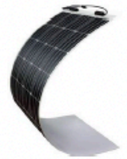

USB to TTL serial cable


2.1mm connector jack

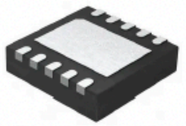

Variable power supply

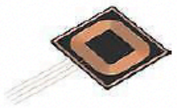
Energy Harvesting


Solar cell [light]


Solar cell [light] (thin film)


Motor/Generator [motion]


Piezoelectric generator [vibration]


Antenna/ Rectenna [radio waves]


Thermoelectric generator [heat]


Hand crank

Battery-powered IoT sensors

- Several IoT applications make use of battery-powered sensor nodes, e.g., habitat monitoring, volcano monitoring, structural monitoring, vehicle tracking.
- Major limitation: battery capacity is limited.
  - Bigger batteries are more expensive and heavier. Sometimes it is simply not possible to use them.
  - Sensor nodes can be deployed in hard-to-reach locations, want to minimise human interventions (workers cost money, it takes time to replace batteries).
- Possible solutions:
  - Equipping sensors with low-power processors and radio, but this diminishes computational ability and nodes have lower transmission rates
  - Energy-aware communication protocols - effective, yet battery capacity is still limited

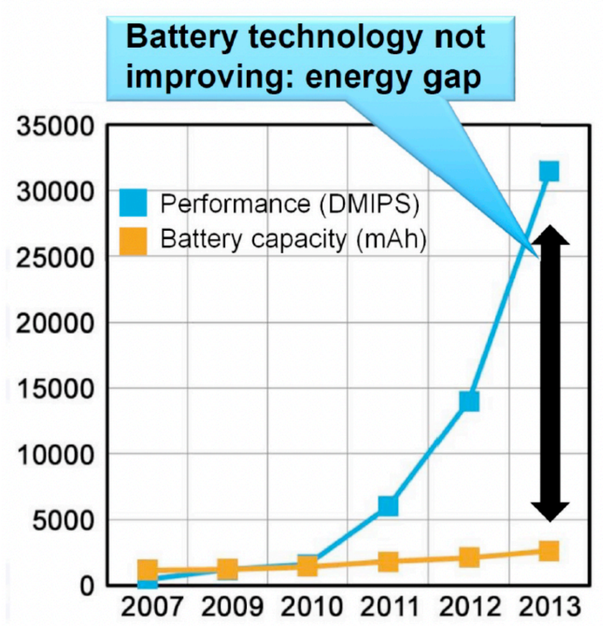
Batteries (1)

Type	2x AAAA	CR2032	CR123A	CR2
Material	Alkaline	LiMnO2*	Lithium	Lithium
Voltage	3 V	3 V	3 V	3 V
Capacity	1000 mAh	225 mAh	1500 mAh	800 mAh
Diameter	10.5 mm (x2)	20 mm	17 mm	15.6 mm
Height	45 mm	3.2 mm	34 mm	27 mm
Weight	24 g	3 g	17 g	11 g

Lithium Manganese Dioxide

Batteries (2)

- Although here has been progress in these batteries, the general complaint from circuit designers is that batteries are not keeping up with the trends in electronics.



- Semiconductor chip performance tends to improve rapidly (blue line, Dhrystone Million Instructions per Second in microprocessors)
- Battery capacity improves slowly, only about 5-8% per year

Another solution: Energy Harvesting

- **Energy Harvesting**: harvesting energy from the environment or other energy sources (e.g., body heat, movement)
- Different harvesting architectures:
  - **Harvest-use**: harvesting system directly powers the sensor node. If no sufficient energy is available, node is disabled.
  - **Harvest-store-use**: harvesting system has a storage component that stores harvested energy and powers the node.

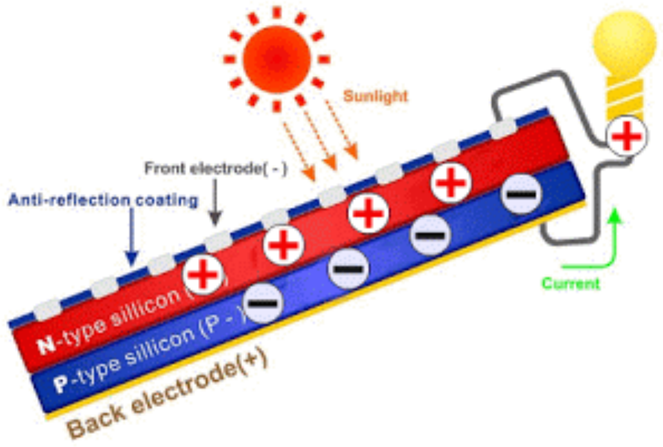
Sources of Harvestable Energy

- Energy sources have different characteristics depending on their controllability, predictability and magnitude.
  - **Controllable energy sources** provide harvestable energy whenever required (e.g., wrist movement). **Uncontrollable energy sources** do not (e.g., solar energy).
  - Availability of some energy sources can be predicted with more or less accuracy (e.g., solar and wind).
  - Some energy sources have more magnitude than others (e.g., solar and wind energies have more magnitude than breathing and blood pressure).

Ambient energy harvesting

Ambient energy harvesting

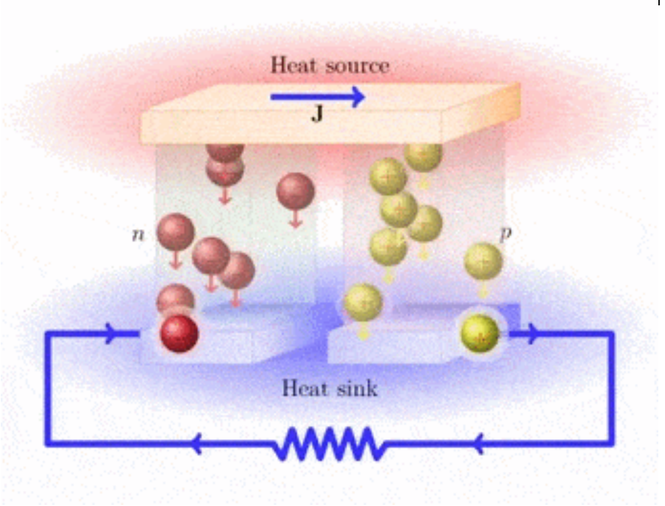
- Solar energy: most abundant energy source on Earth. Mainly exploit photovoltaic effect, i.e., the ability of converting solar radiation into direct current electricity.



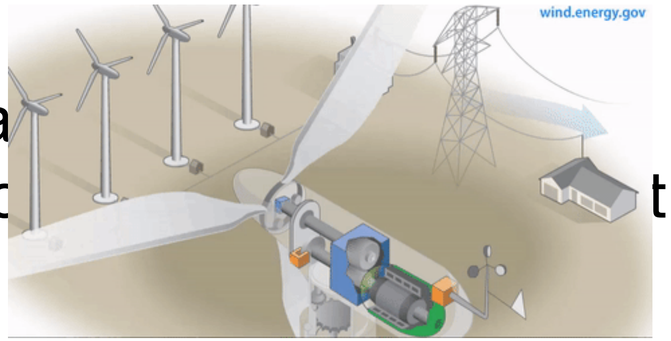
Ambient energy harvesting

- Solar energy: most abundant energy source on Earth. Mainly exploit photovoltaic effect, i.e., the ability of converting solar radiation into direct current electricity.

- Thermoelectric generators are devices made of junction of two dissimilar materials that, in presence of thermal gradient, are able to convert heat into electricity. Can be used to capture energy from industrial equipment, soil and human body.



Ambient energy harvesting



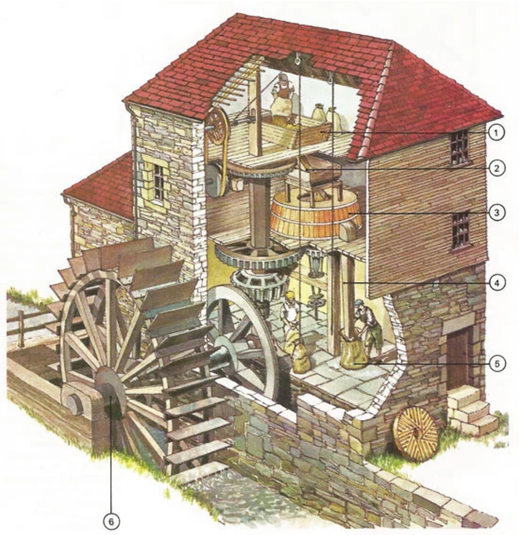
- Solar energy: most abundant energy source on Ea
  photovoltaic effect, i.e., the ability of converting so t
  current electricity.

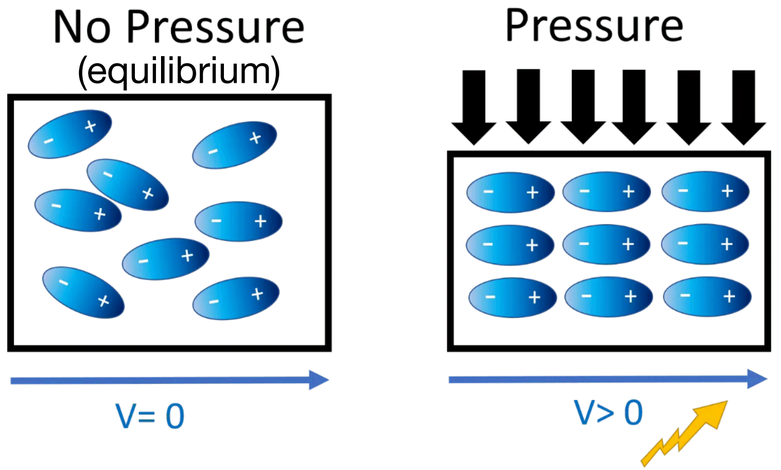
- Thermoelectric generators are devices made of junction of two dissimilar materials that, in presence of thermal gradient, are able to convert heat into electricity. Can be used to capture energy from industrial equipment, soil and human body.

- Wind energy: wind turbines harvest kinetic energy produced by rotating blades that are activated when wind hits them.

Mechanical Energy Harvesting

- Mechanical vibration energy harvesting is often achieved with Piezoelectric Energy Harvesters.
- Piezoelectric materials (e.g., crystals, bones, DNA) have the property of being able to accumulate electric charge in response to applied mechanical stress.
- This property depends on the arrangement of materials' molecules (e.g., hexagons or spirals).
- Used for wearable devices, vehicles





Human Energy Harvesting

- Human heat can be used by harvesters to produce electricity from heat.
    - Thermoelectrical energy harvesters use temperature gradient of the body.
- Human movement
- You often see a combination of all these technologies in devices.

3.3.5. In-depth analysis: Energy harvesting - Markov Decision Processes (MDP)

Markov Chains and Markov Decision Processes

- **On the blackboard.**
- A lot of online material, e.g.,:
  - Markov Chains: https://www.stat.auckland.ac.nz/~fewster/325/notes/ch8.pdf, https://web.stanford.edu/class/cs265/Lectures/Lecture14/l14.pdf
  - Markov Decision Processes: https://www.davidsilver.uk/wp-content/uploads/2020/03/MDP.pdf

A Learning Theoretic Approach to Energy Harvesting Communication System Optimization

Pol Blasco, Deniz Gündüz, and Mischa Dohler

IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, VOL. 12, NO. 4, APRIL 2013

Bibliography

Standford IoT course

Introduction to the Internet of Things and Embedded Systems
University of California, Irvine

Designing Embedded Systems and the Internet of
Things (IoT) with the ARM® Mbed™, Perry Xiao
London South Bank University

Sudevalayam, Sujesha, and Purushottam Kulkarni. "Energy harvesting sensor nodes: Survey and implications." IEEE communications surveys & tutorials 13.3 (2010): 443-461.

Sanislav, T., Mois, G. D., Zeadally, S., & Folea, S. C. (2021). Energy harvesting techniques for internet of things (IoT). IEEE access, 9, 39530-39549.

A Learning Theoretic Approach to Energy Harvesting Communication System Optimization
Pol Blasco, Deniz Gündüz, and Mischa Dohler
IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, VOL. 12, NO. 4, APRIL 2013