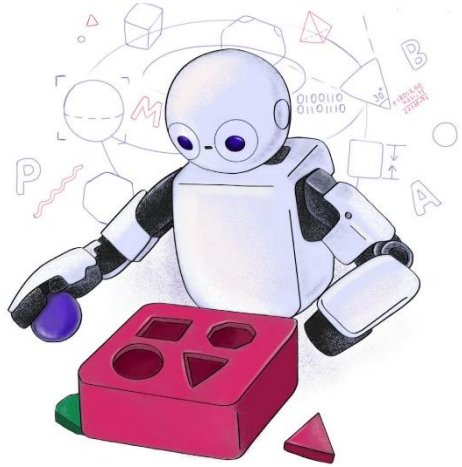
TP557 - Tópicos avançados em IoT e Machine Learning:

Desafios do TinyML:

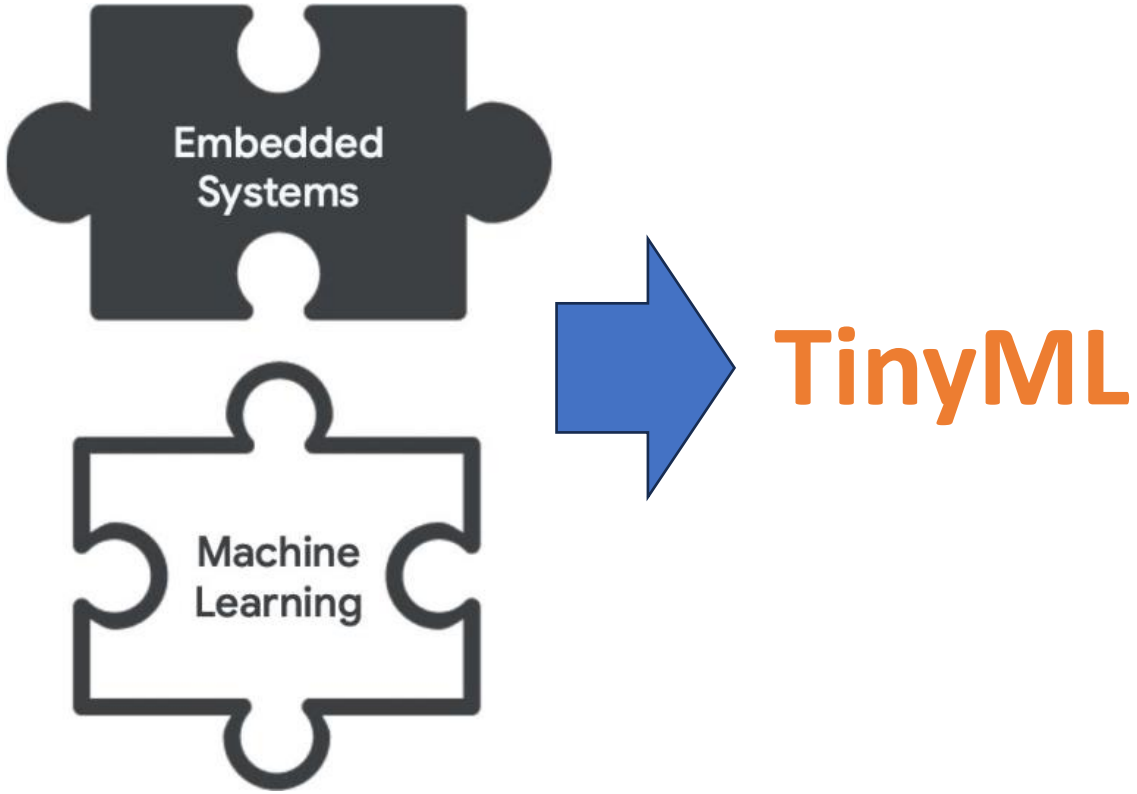
Sistemas Embarcados



Inatel

Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

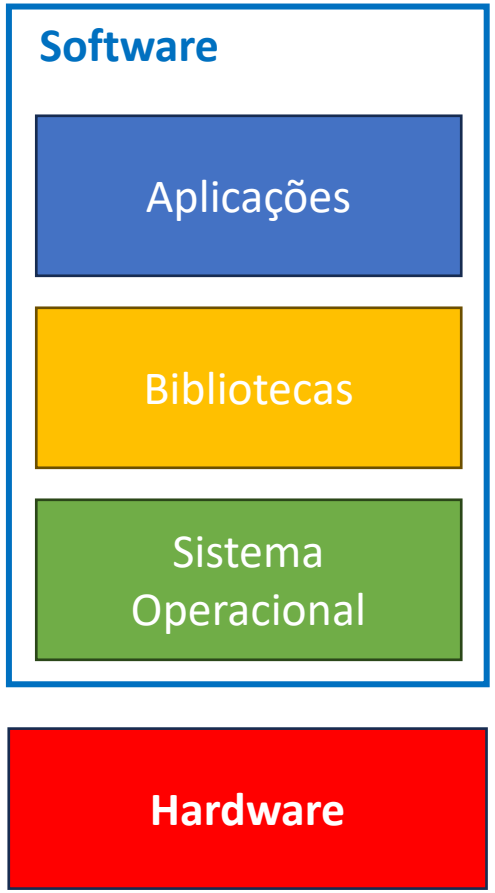
Recapitulando



- Vimos anteriormente que ***TinyML*** é a junção de ***ML*** com ***sistemas embarcados***.
- Ou seja, é a execução de algoritmos de ML em sistemas com ***restrições de processamento, memória e consumo de energia***.
- Nesta aula, falaremos sobre os ***desafios de hardware (HW)*** para execução de ***aplicações de ML*** nesses dispositivos com restrições.

Quais desafios encontraremos na execução de *machine learning* em sistemas embarcados?

Sistema computacional de uso geral



- Antes de falarmos de sistemas embarcados, vamos fazer um ***paralelo*** com o ***sistema de um computador de uso geral***.
- Em termos gerais, o que temos é o software (SW) (aplicações, bibliotecas e sistema operacional (SO)) rodando em cima do HW.

Hardware

CPU

Armazenamento de curto prazo

Armazenamento de longo prazo

- Os componentes mais importantes do HW de um computador de uso geral são:
  - Unidade de processamento central (CPU)
  - E as memórias de armazenamento volátil e de longo prazo.

Hardware

CPU

Armazenamento de curto prazo

Armazenamento de longo prazo

- **Unidade de processamento central** (CPU): é o ***cérebro do computador***. É responsável por
  - cálculos,
  - tomada de decisões,
  - controle de dispositivos,
  - e execução de programas.

Hardware

CPU

Armazenamento de curto prazo

Armazenamento de longo prazo

- **Memória volátil**: ***armazena temporariamente*** dados e instruções sendo processados pela CPU.
  - Exemplos: RAM e a cache do processador.

Hardware

CPU

Armazenamento de curto prazo

Armazenamento de longo prazo

- **Memória não-volátil**: ***armazena permanentemente*** arquivos, programas e outros dados que podem ser acessados posteriormente mesmo após sistema ser reiniciado.
  - Exemplos: discos rígidos (HDs), unidades de estado sólido (SSDs), memória *flash* e memória de acesso aleatório não volátil (NVRAM).

Hardware

- Mas percebam que todos eles são ***componentes separados***.
- Discutiremos mais sobre isso na sequência

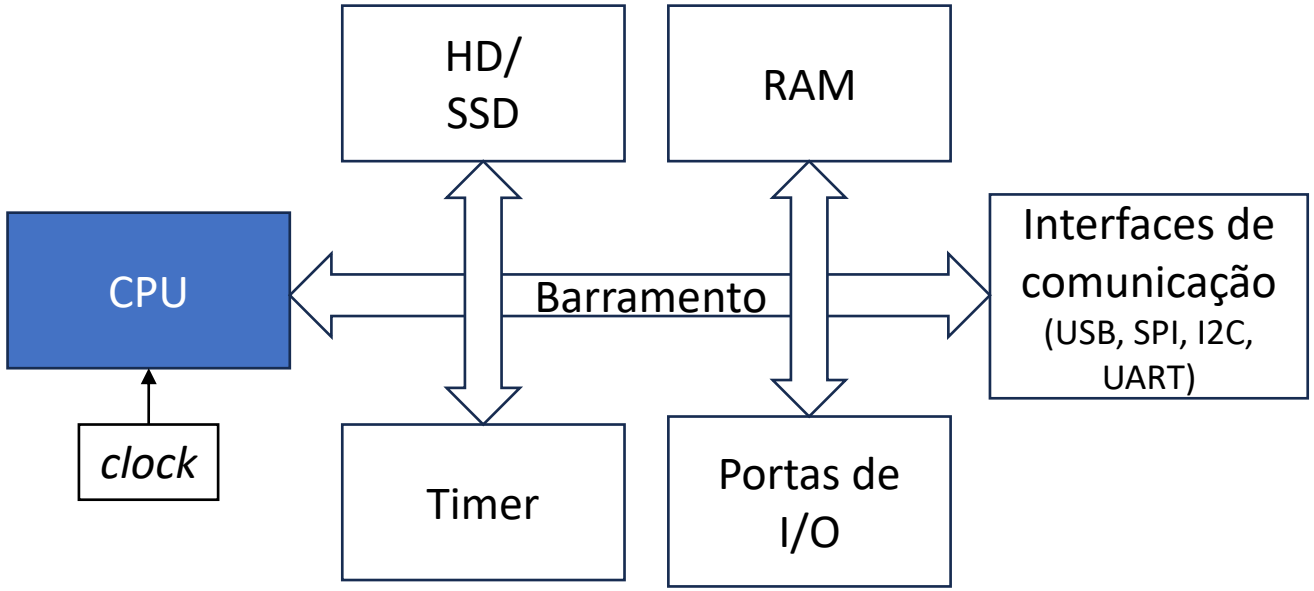
Porém, em geral, sistemas embarcados possuem microcontroladores!

Micro***processador***

versus

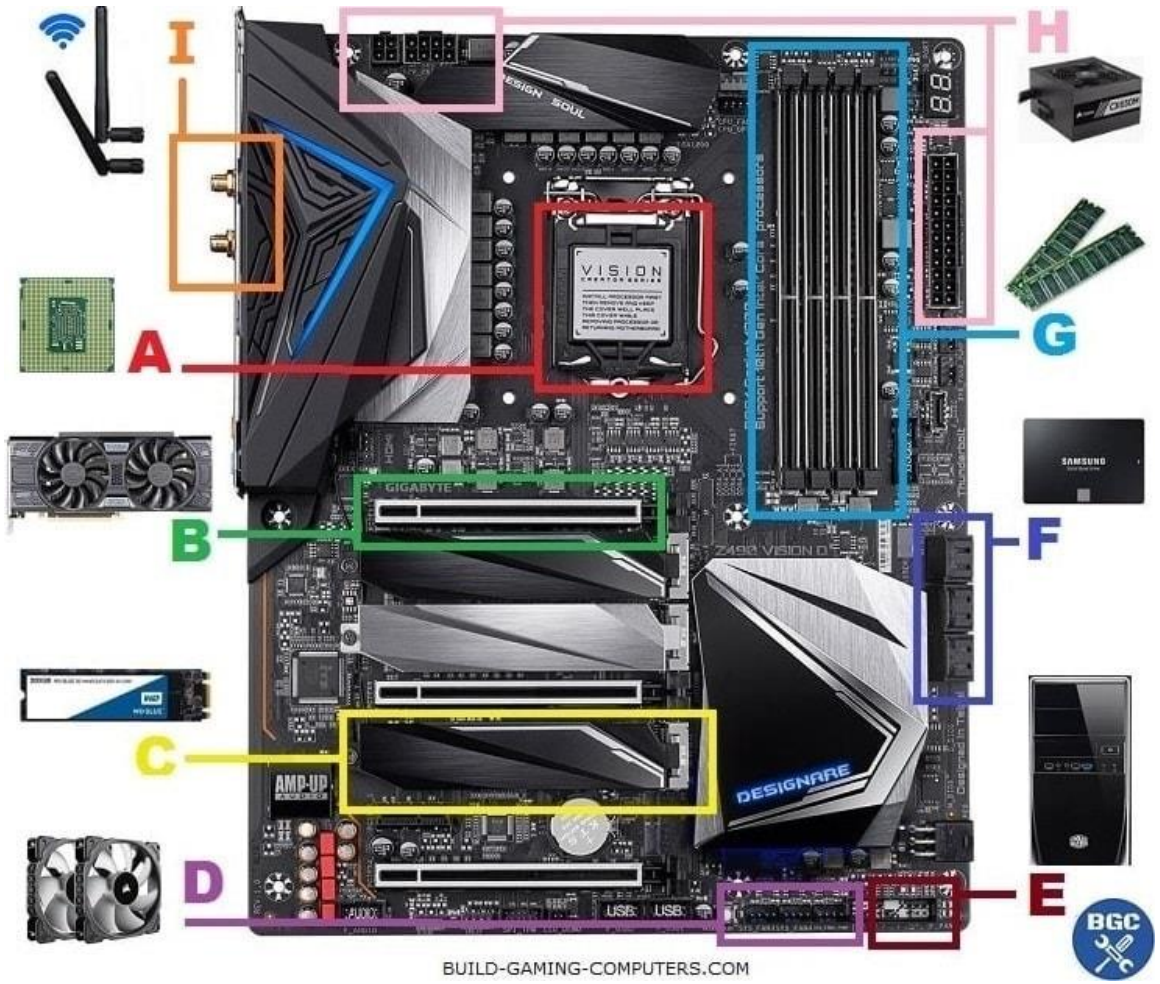
Micro***controlador***

Microprocessador: apenas uma peça do quebra-cabeças



- A CPU é a ***parte principal do computador***, mas para que ela funcione, ***são necessários outros componentes***, como:
  - sistema de *clock*
  - memórias volátil e não-volátil,
  - controladores de I/O,
  - interfaces de comunicação,
  - timers,
  - etc.

Microprocessador: apenas uma peça do quebra-cabeças

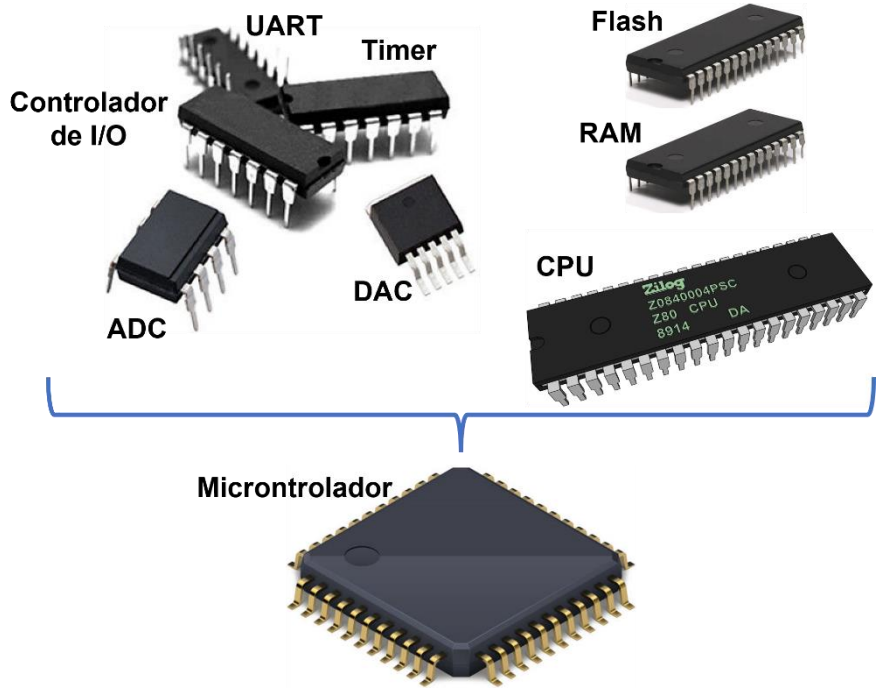


- Em um computador de uso geral, ***todos estes componentes são chips separados*** soldados ou conectados à placa mãe e ***interligados através do barramento*** de comunicação disponibilizado pela ***placa mãe***.

Microcontrolador, tudo junto e misturado

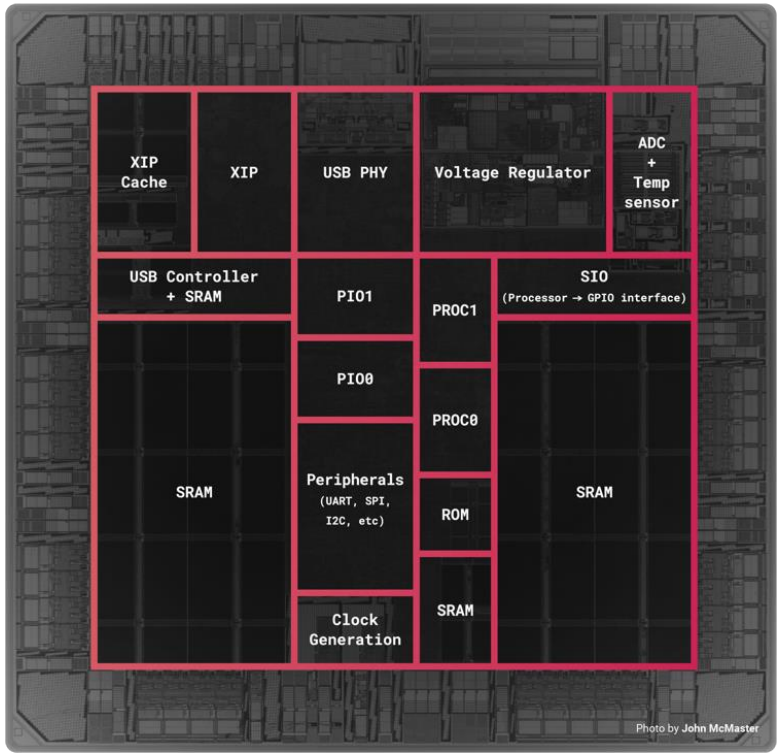
Microcontrolador

CPU	HD/ SSD	RAM
Timer	Portas de I/O	Interfaces de comunicação



- O microcontrolador é um ***dispositivo completo em um único chip***.
- Ele combina uma CPU com memórias, periféricos de entrada/saída, temporizadores e outros recursos, tornando-o ***adequado para aplicações embarcadas***.
- O barramento é interno ao chip.

Microcontrolador, tudo junto e misturado



- ***Die*** (ou pastilha) do microcontrolador RP2040.
  - *Die* é bloco de material semicondutor onde um circuito é fabricado.
- Primeiro microcontrolador projetado pela ***Raspberry Pi Ltd***.
- Contém um ARM Cortex-M0+ dual core de 32 bits.
- Percebam que ele possui os componentes necessários para funcionar integrados em um pequeno *chip*.

Microprocessador

- ***Cérebro*** de um ***sistema de uso geral***.
- É ***apenas a CPU***, memória, armazenamento, etc. são externos a ele.
- Usado principalmente ***sistemas de uso geral*** como desktops, laptops e servidores.
- Oferece ***flexibilidade*** porque é projetado para ser ***usado em uma ampla variedade de aplicações***.
- O ***sistema é grande*** e, em geral, ***não apresenta restrições*** de processamento, memória, armazenamento e consumo de energia.

Microcontrolador

- **Cérebro** de um ***sistema embarcado***.
- ***Memória, armazenamento e outros periféricos*** (e.g., controladores de I/O, ADC, DAC, timers, etc.) ***são internos ao chip***.
- Usado principalmente em ***sistemas especializados***, com função específica, como MP3 players, telefones, sensores, etc.
- ***Não oferece flexibilidade*** porque é projetado para atender a um ***propósito específico***.
- O ***sistema é pequeno*** e ***apresenta restrições*** de processamento, memória armazenamento e consumo de energia.

Diferenças de magnitude

	Microprocessador	>	Microcontrolador	nRF52840*
Freq. de operação	1 GHz – 4 GHz	> 2.5 x	1 MHz – 400 MHz	64 MHz
Memória	512 MB – 128 GB	> 1000 x	2 KB – 512 KB	256 KB
Armazenamento	64 GB – 16 TB	> 30000 x	32 KB – 2 MB	1 MB
Consumo	30 – 700 W	> 1000 x	150 μW – 24 mW	< 1 mW

* μC usado no kit tinyML

- Quanto ***mais elevada é a frequência de operação*** de uma CPU, ***mais instruções por segundo são processadas*** e, consequentemente, ***maior é o consumo de energia***.
- As diferenças são muito grandes, principalmente devido ao tamanho dos sistemas e suas limitações.
- Percebam que ***vários desafios devem ser superados*** para que aplicações de ML possam ser executadas nesses dispositivos com várias restrições.

Quais são as implicações das limitações?

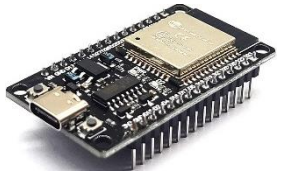
nRF52840
64 MHz
256 KB
1 MB
< 1 mW

- Com ***sistemas computacionais de uso geral***, nós praticamente ***não precisamos nos preocupar com memória, consumo, poder computacional***.
- Porém, com sistemas embarcados precisamos saber
  - o ***quanto de memória a aplicação requer,*** pois pode não caber na memória.
  - o ***consumo de energia da aplicação***, pois a bateria pode não durar muito.
  - a ***quantidade necessária de processamento de dados requerido pela aplicação***, pois a CPU pode não atender.

Comparação de hardwares embarcados

Os ***endpoints*** são, em geral, dispositivos com restrições de recursos.

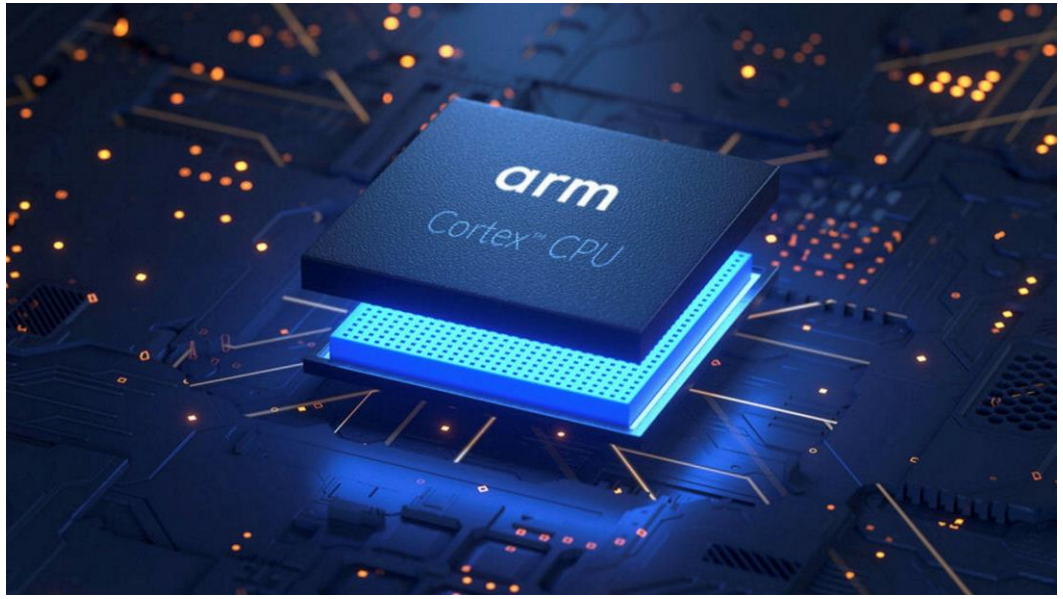


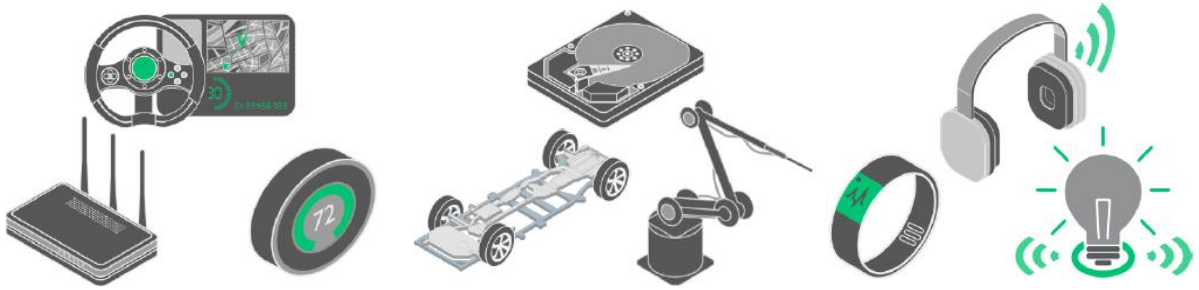


	Raspberry Pico	Arduino Nano Sense	ESP 32	Seeed Wio Terminal	Arduino Portenta
CPU*	Dual Core ARM Cortex M0+	Single Core ARM Cortex M4	Dual Core Cadence Xtensa LX6	Single Core ARM Cortex M4	Dual Core ARM Cortex M4/M7
Clock	133 MHz	64 MHz	240 MHz	120 MHz	240/480 MHz
RAM	264 KB	256 KB	520 KB	192 KB	1 MB***
Flash	2 MB	1 MB	2 MB	4 MB	2 MB
Rádio	-	BLE**	BLE/WiFi	BLE/WiFi	BLE/WiFi
Sensores	Não	Sim	Não	Sim	Não
Preço	~ $ 5,00	~ $ 40,00	~ $ 16,00	~ $ 44,00	~ $ 100,00

* Todas as CPUs são de 32 bits. ** Chip BLE multiprotocolo: suporta os protocolos ZigBee e Thread. *** É possível rodar aplicações de detecção de objetos.

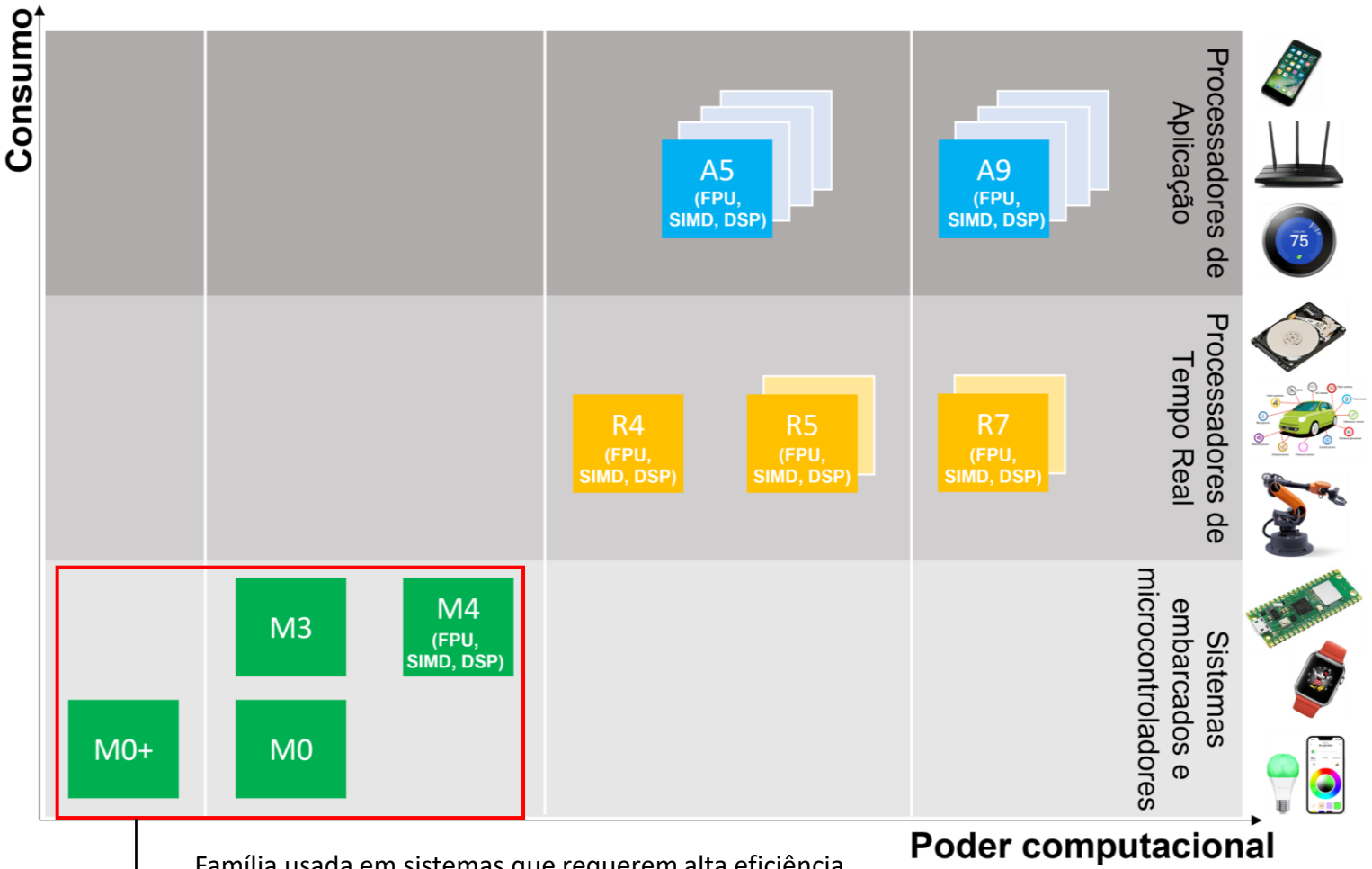
Dominância dos processadores ARM





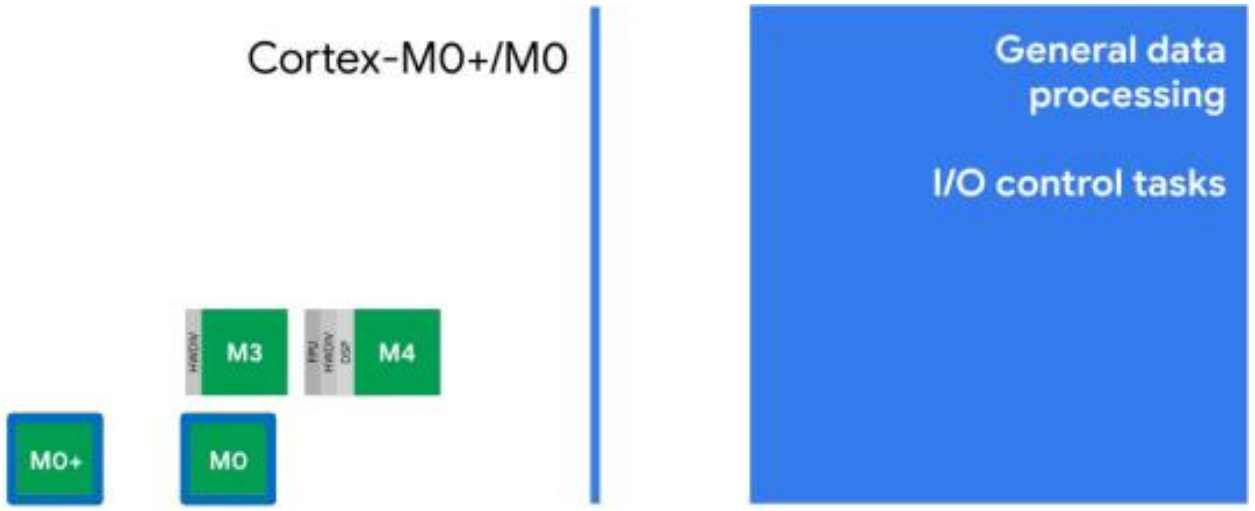
- Processadores ARM estão em praticamente todos os sistemas embarcados.
- Isso se deve a alguns fatores principais:
  - ***Alta eficiência energética***: consomem menos energia em comparação com outras arquiteturas de processadores.
  - ***Baixo Custo***: como são focados em sistemas embarcados, têm alta demanda e com isso têm custo bem menor do que outros processadores.
  - ***Flexibilidade***: fabricantes podem produzir *chips* personalizados para atender a necessidades específicas (número de núcleos, FPU, MMU, DSP, SIMD, etc.).

CPUs ARM Cortex para sistemas embarcados



- **Cortex A**: alta performance.
  - Suportam até 4 cores.
  - Por terem MMU, executam SOs completos, como Android e Linux.
- **Cortex R**: alta performance e resposta rápida.
  - Suportam até 2 cores.
  - Possuem recursos de redundância e recuperação de falhas.
  - Não têm MMU, assim, não executam SOs completos, apenas SOs mais simples como RTOSs.
- **Cortex M**: pequenos e com baixíssimo consumo.
  - Não executam SOs completos, mas podem executar RTOSs.

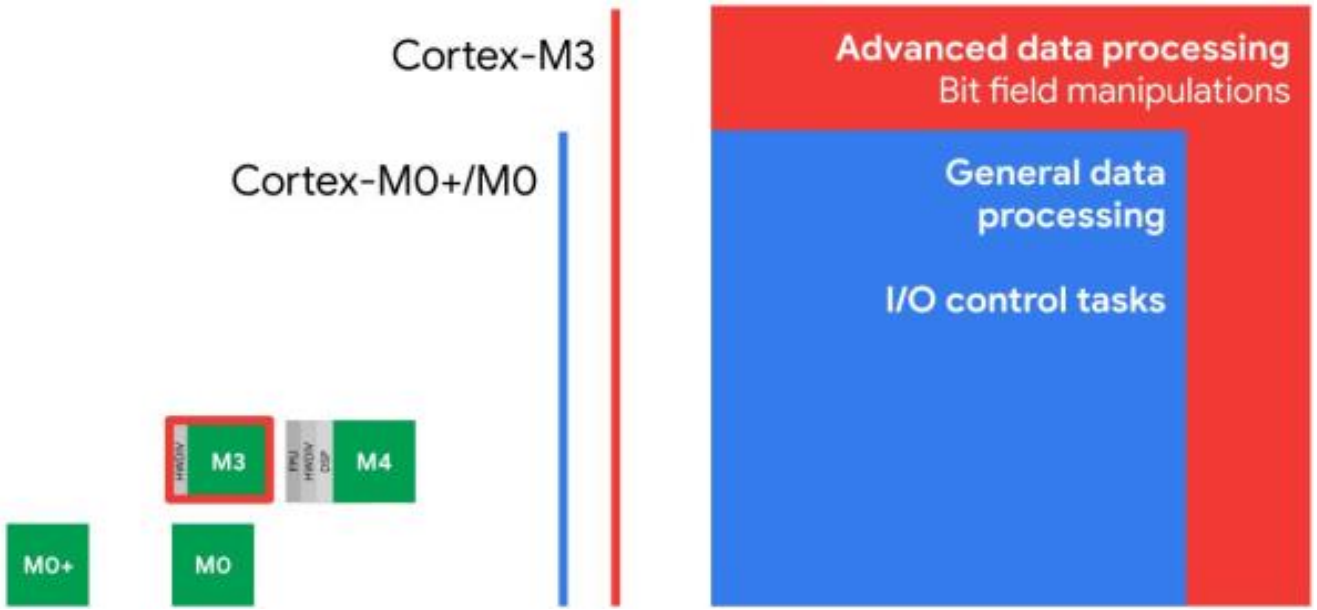
Conjunto de instruções da família ARM Cortex-M



Cortex-M0+ e M0

- Apresentam um ***conjunto de instruções simplificado***, focando em ***operações essenciais***, projetado para ***economizar memória e energia*** ***em sistemas com recursos limitados***.
- Possui um conjunto de instruções para ***processamento de dados de uso geral*** e ***tarefas simples de controle de E/S***.

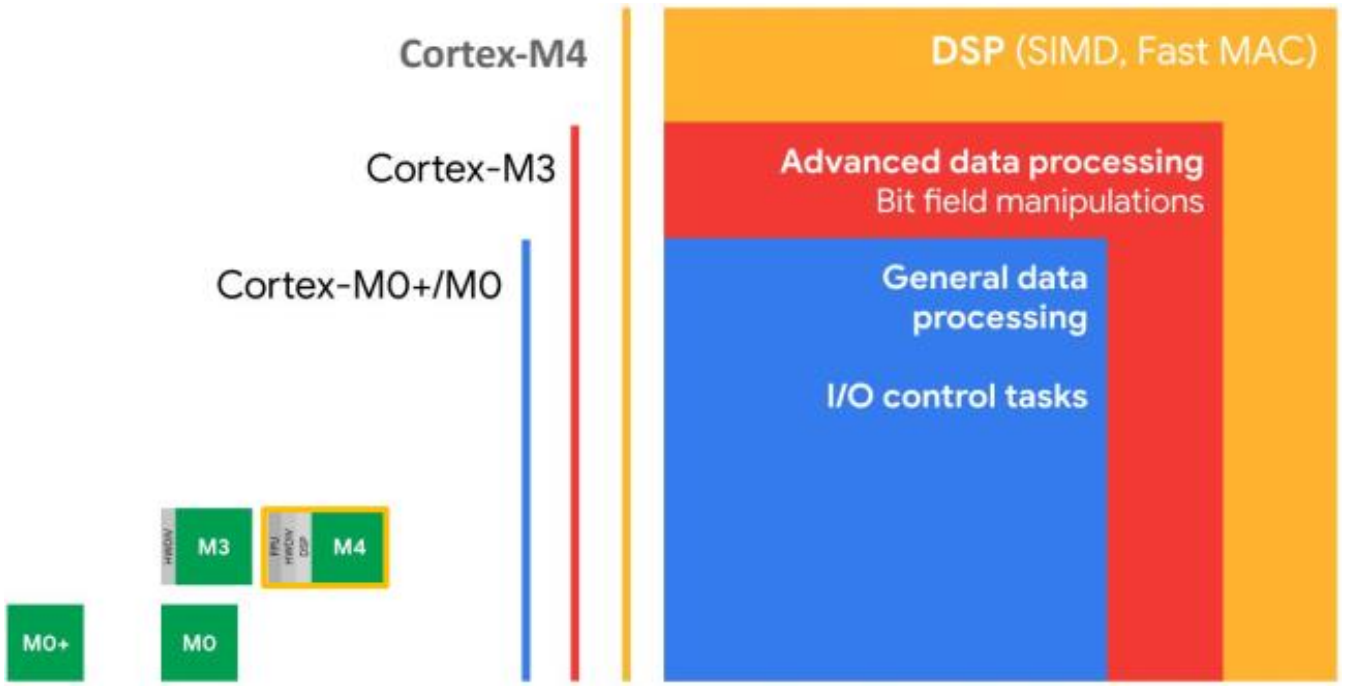
Conjunto de instruções da família ARM Cortex-M



Cortex-M3

- O conjunto de instruções aumenta com o modelo.
- Cada modelo mais alto oferece um ***superconjunto*** das instruções do modelo anterior.
  - Por exemplo, um M3 deve executar códigos do M0/M0+.
- Os Cortex-M3 ***adicionam*** instruções para ***manipulação de bits e divisão em hardware***.

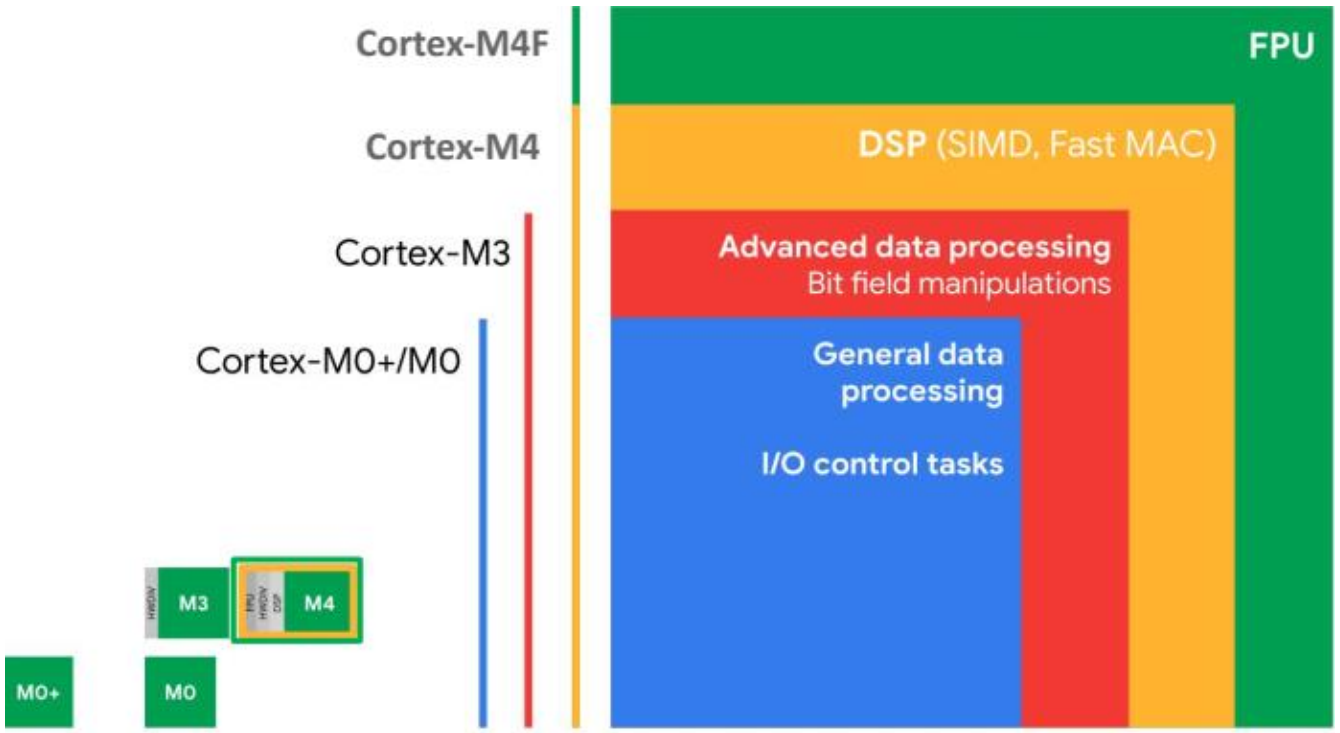
Conjunto de instruções da família ARM Cortex-M



Cortex-M4

- Adicionam instruções básicas de processamento digital de sinais (DSP) como:
  - ***Single Instruction Multiple Data* (SIMD)**: a mesma instrução é aplicada a grandes quantidades de dados.
  - ***Multiply and Accumulate* (MAC)**: realiza multiplicação e soma em ***um único ciclo de clock***.
- Esses instruções ***aceleram tarefas complexas de processamento de dados***, como as encontradas em ***aplicações de áudio e vídeo***.
- Podem ter ***opcionalmente*** uma unidade de ponto flutuante (FPU).

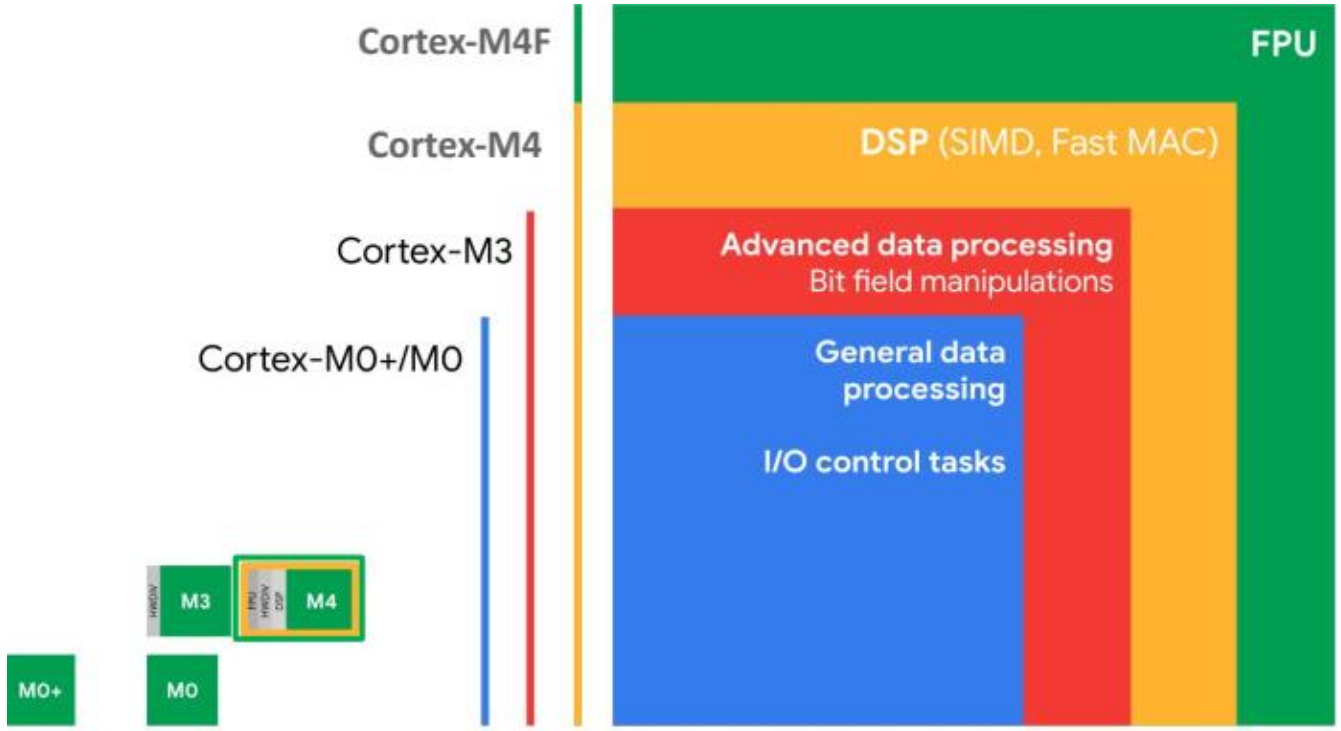
Conjunto de instruções da família ARM Cortex-M



Cortex-M4F

- Superconjunto contendo todas as instruções dos modelos anteriores.
- Diferença para o M4 é que já vem com uma unidade de ponto flutuante (FPU).
- Como eles têm FPU, possuem as instruções necessárias para cálculos em ponto flutuante.

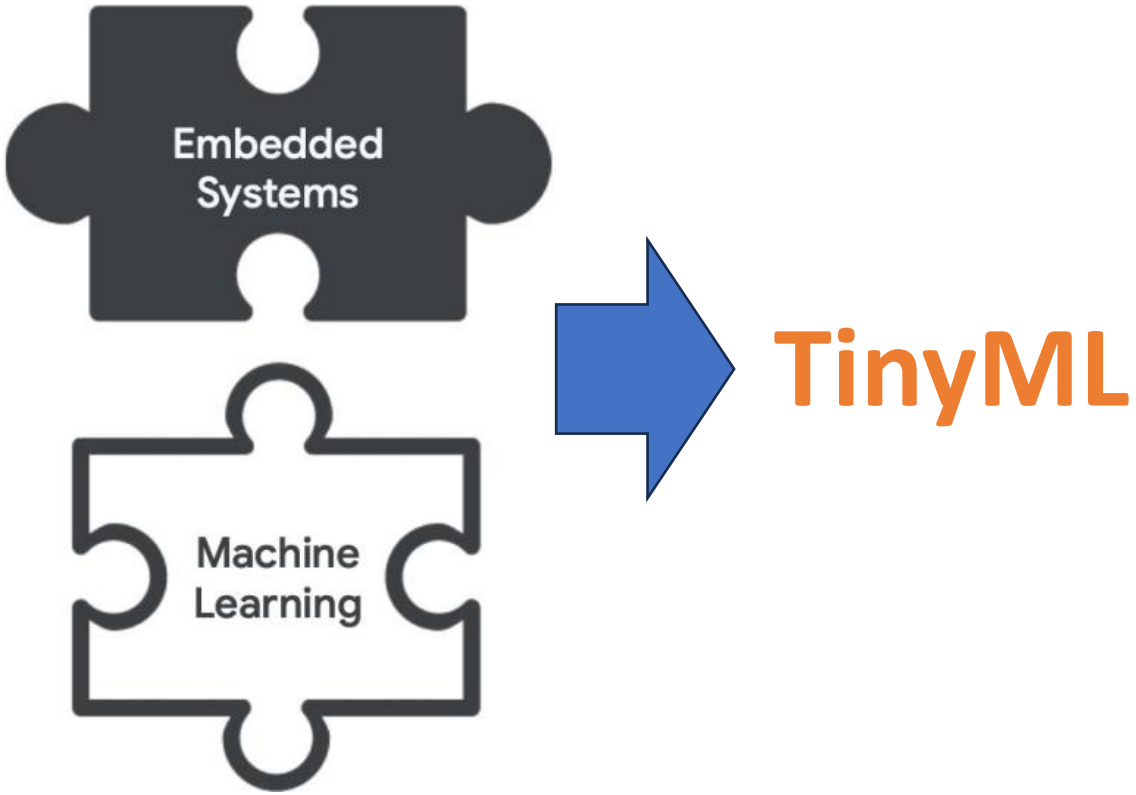
Conjunto de instruções da família ARM Cortex-M



Observações

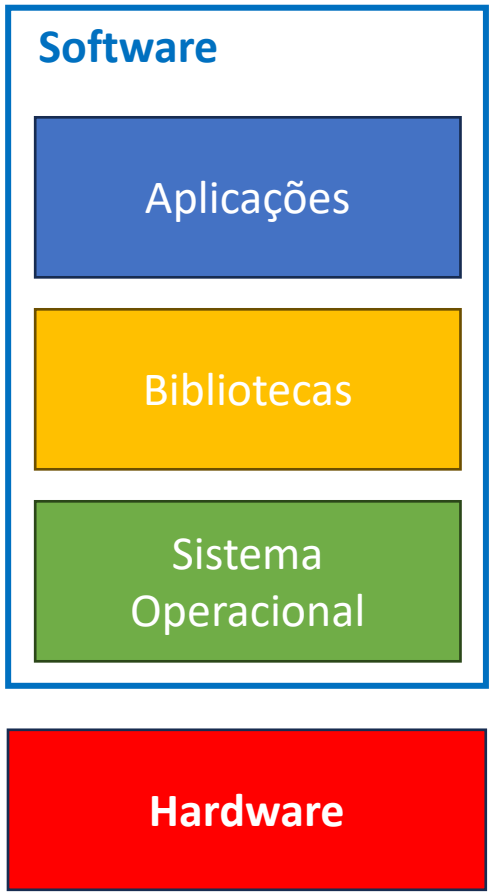
- Até os modelos M3 e M4 não há FPU, todas as operações são feitas em ***ponto fixo***.
- Com o TinyML, vamos evitar ao máximo usar operações em ***ponto flutuante***, pois ***consomem mais memória e processamento, o que implica em mais consumo de energia***.
  - Uma variável de ponto flutuante de precisão simples ocupa 4 bytes.
  - O uso de operações em ***ponto fixo*** é uma das técnicas para reduzir o tamanhos de modelos de ML.

Desafios da execução de SW em sistemas embarcados



- Anteriormente, falamos dos desafios de hardware, agora falaremos do ***software de um sistema computacional de uso geral para fazermos um paralelo*** com o mundo do ***tinyML***.
- Na sequência, veremos os desafios para execução de SW, incluindo algoritmos de ML, em dispositivos ***pequenos***, com ***restrições de custo***, ***recursos computacionais e consumo***.

Software



- Vamos começar pelo sistema operacional (SO).
- Um sistema operacional ***gerencia recursos e fornece uma interface entre os componentes físicos do computador (i.e., hardware) e os programas*** que são executados nele.
- Ele atua como uma ***camada de software intermediária*** que permite que os usuários interajam com o computador e executem tarefas de forma eficiente.

Sistemas Operacionais Amplamente Usados

Sistemas operacionais de uso geral

Sistemas operacionais móveis

Sistemas operacionais embarcados

- Podemos dividir os OSs nos três grandes grupos ao lado.

Sistemas Operacionais Amplamente Usados

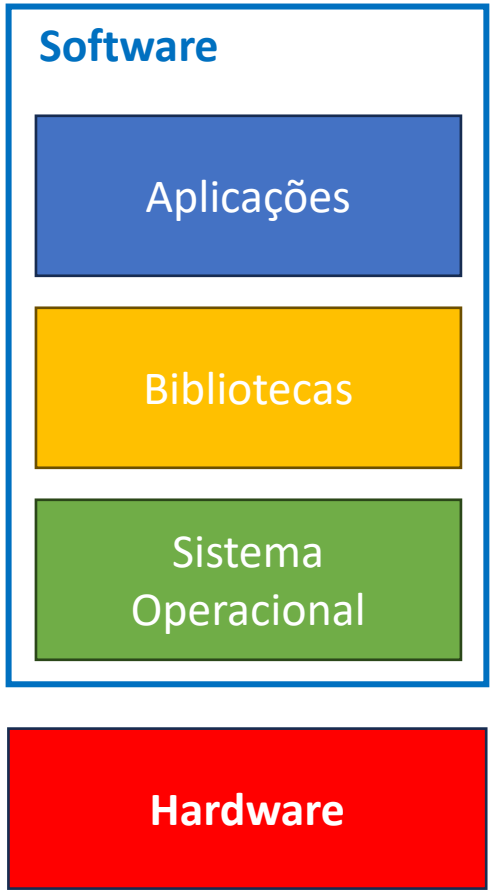
Sistemas operacionais de uso geral

Sistemas operacionais móveis

Sistemas operacionais embarcados

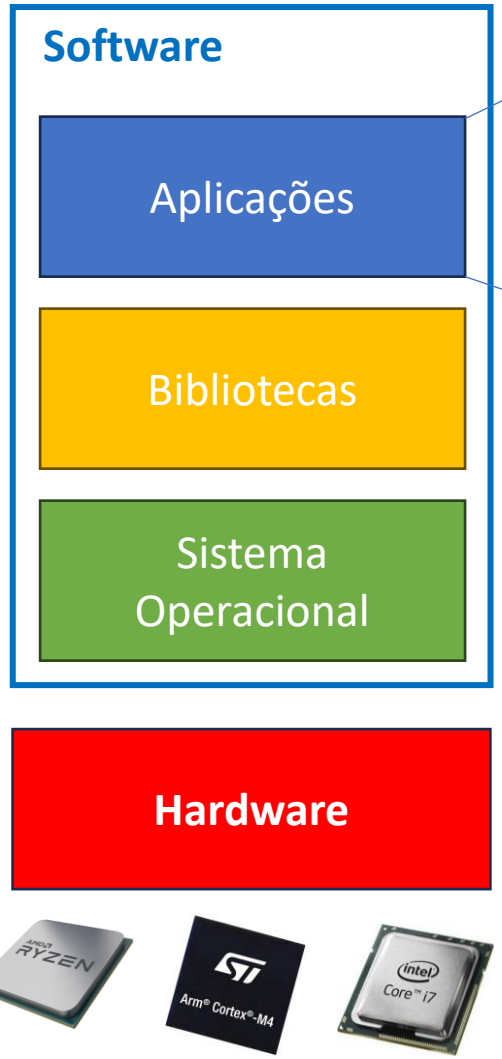
- Por exemplo, o arduino nano 33 pode rodar os SOs: ***Free RTOS*** e ***ARM mbed***.
- Entretanto, mesmo podendo usar um SO para controlar as aplicações sendo executadas no sistema, no mundo do tinyML, nós iremos optar pela execução em "***bare metal***", ou seja, sem SO, em cima do metal nu.
- Pois queremos ter ***controle direto sobre o HW***, criar ***soluções extremamente específicas***, ***eficientes e enxutas***.
  - Ex.: Aplicação *always on* para *keyword spotting*.

Bibliotecas



- Em cima do OS nós temos as bibliotecas.
- Bibliotecas são ***conjuntos de código reutilizável*** (funções, classes, etc.) para ***adicionar funcionalidades a um programa sem a necessidade de escrever todo o código do zero***.
  - Por exemplo a biblioteca TensorFlow, que possibilita a criação e treinamento de modelos de ML.
  - Ou a biblioteca NumPy que fornece código para trabalharmos com matrizes multidimensionais (chamadas de *arrays*).

Bibliotecas



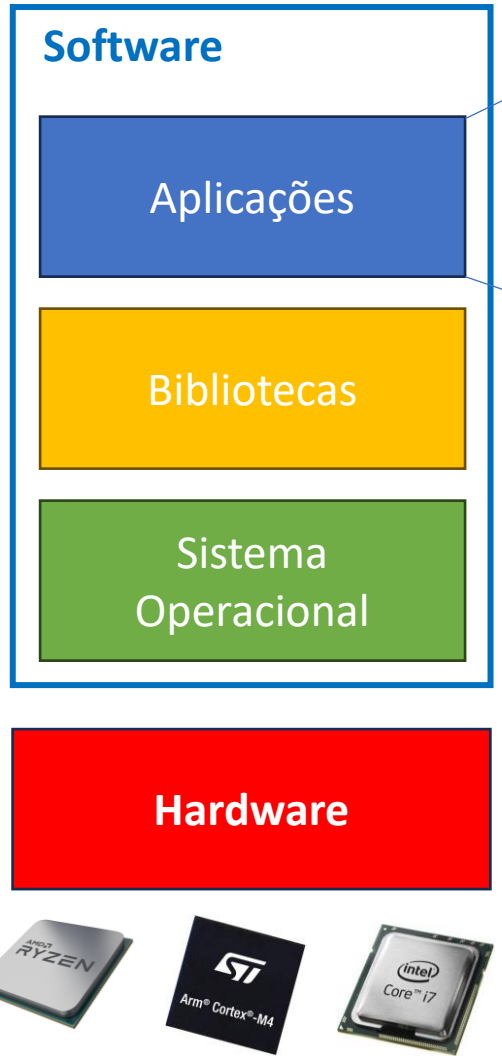

```
import numpy as np

for x in range(10):
    resultado = np.sqrt(x)
    print(resultado)
```

```
0.0
1.0
1.4142135623730951
1.7320508075688772
2.0
2.23606797749979
2.449489742783178
2.6457513110645907
2.8284271247461903
3.0
```

- O código (i.e., a aplicação) ao lado importa a biblioteca ***Numpy*** e utiliza a função ***sqrt()*** para calcular as raízes quadradas de 0 a 9.
- O resultado são valores em ***ponto flutuante*** (*float*).
- Uma vez escrito o código, ***devido à portabilidade do Python***, podemos ***executá-lo virtualmente em qualquer sistema, com qualquer tipo de CPU***.
- Nós não precisamos nós preocupar com a portabilidade, ***apenas em escrever e executar o código***.

Bibliotecas



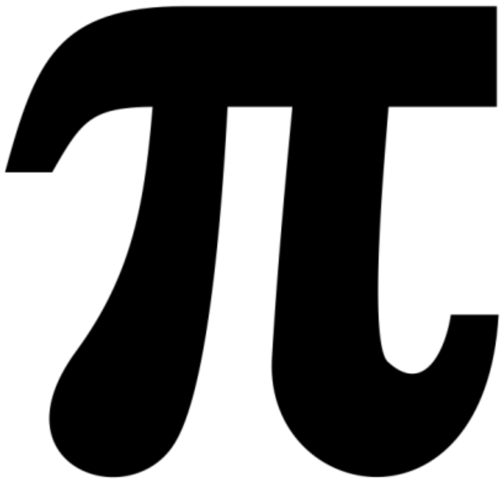

```
import numpy as np

for x in range(10):
    resultado = np.sqrt(x)
    print(resultado)
```

```
0.0
1.0
1.4142135623730951
1.7320508075688772
2.0
2.23606797749979
2.449489742783178
2.6457513110645907
2.8284271247461903
3.0
```

- A ***máquina virtual*** do Python se encarrega da ***portabilidade***.
- Ela cria uma camada de abstração e lida com os detalhes de baixo nível.
- Desta forma, o código roda perfeitamente em qualquer SO e HW.
- Essa é a beleza de um sistema de uso geral.

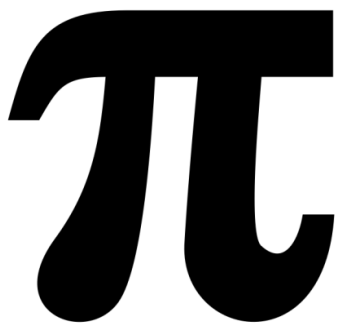
Portabilidade

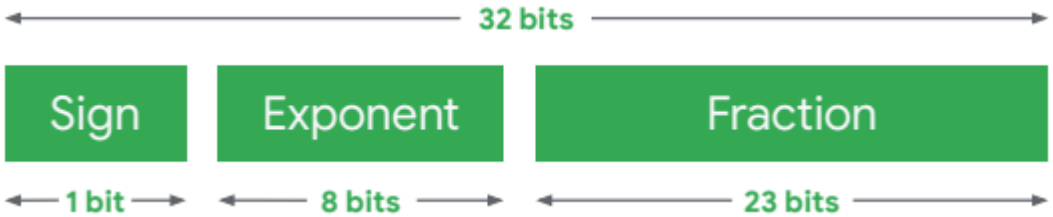


3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034
82534211706798214808651328230664709384460955058223172535940812848111745028410270193852110
55596446229489549303819644288109756659334461284756482337867831652712019091456485669234603
48610454326648213393607260249141273724587006606315588174881520920962829254091715364367892
59036001133053054882046652138414695194151160943305727036575959195309218611738193261179310
51185480744623799627495673518857527248912279381830119491298336733624406566430860213949463
95224737190702179860943702770539217176293176752384674818467669405132000568127145263560827
78577134275778960917363717872146844090122495343014654958537105079227968925892354201995611
21290219608640344181598136297747713099605187072113499999983729780499510597317328160963185
95024459455346908302642522308253344685035261931188171010003137838752886587533208381420617
17766914730359825349042875546873115956286388235378759375195778185778053217122680661300192
78766111959092164201989380952572010654858632788659361533818279682303019520353018529689957
73622599413891249721775283479131515574857242454150695950829533116861727855889075098381754
63746493931925506040092770167113900984882401285836160356370766010471018194295559619894676
78374494482553797747268471040475346462080466842590694912933136770289891521047521620569660
24058038150193511253382430035587640247496473263914199272604269922796782354781636009341721
64121992458631503028618297455570674983850549458858692699569092721079750930295532116534498
72027559602364806654991198818347977535663698074265425278625518184175746728909777727938000
81647060016145249192173217214772350141441973568548161361157352552133475741849468438523323
90739414333454776241686251898356948556209921922218427255025425688767179049460165346680498
86272327917860857843838279679766814541009538837863609506800642251252051173929848960841284
88626945604241965285022210661186306744278622039194945047123713786960956364371917287467764
65757396241389086583264599581339047802759009946576407895126946839835259570982582262052248
94077267194782684826014769909026401363944374553050682034962524517493996514314298091906592
50937221696461515709858387410597885959772975498930161753928468138268683868942774155991855
92524595395943104997252468084598727364469584865383673622262609912460805124388439045124413
65497627807977156914359977001296160894416948685558484063534220722258284886481584560285060
16842739452267467678895252138522549954666727823986456596116354886230577456498035593634568
17432411251507606947945109659609402522887971089314566913686722874894056010150330861792868
09208747609178249385890097149096759852613655497818931297848216829989487226588048575640142
70477555132379641451523746234364542858444795265867821051141354735739523113427166102135969
53623144295248493718711014576540359027993440374200731057853906219838744780847848968332144
57138687519435064302184531910484810053706146806749192781911979399520614196634287544406437
45123718192179998391015919561814675142691239748940907186494231961567945208095146550225231
60388193014209376213785595663893778708303906979207734672218256259966150142150306803844773
45492026054146659252014974428507325186660021324340881907104863317346496514539057962685610
05508106658796998163574736384052571459102897064140110971206280439039759515677157700420337
86993600723055876317635942187312514712053292819182618612586732157919841484882916447060957
52706957220917567116722910981690915280173506712748583222871835209353965725121083579151369
88209144421006751033467110314126711136990865851639831501970165151168517143765761835155650
88490998985998238734552833163550764791853589322618548963213293308985706420467525907091548
14165498594616371802709819943099244889575712828905923233260972997120844335732654893823911
93259746366730583604142813883032038249037589852437441702913276561809377344403070746921120
19130203303801976211011004492932151608424448596376698389522868478312355265821314495768572
62433441893039686426243410773226978028073189154411010446823252716201052652272111660396665
57309254711055785376346682065310989652691862056476931257058635662018558100729360659876486
11791045334885034611365768675324944166803962657978771855608455296541266540853061434443185
86769751456614068007002378776591344017127494704205622305389945613140711270004078547332699

- Portabilidade é ***algo que damos como certo e que não pensamos muito a respeito*** quando criamos códigos para sistemas de uso geral.
- Porém quando partimos para sistemas embarcados, isso se torna um problema.
- Vamos ver um problema envolvendo cálculos em ponto flutuante: o calculo de π.

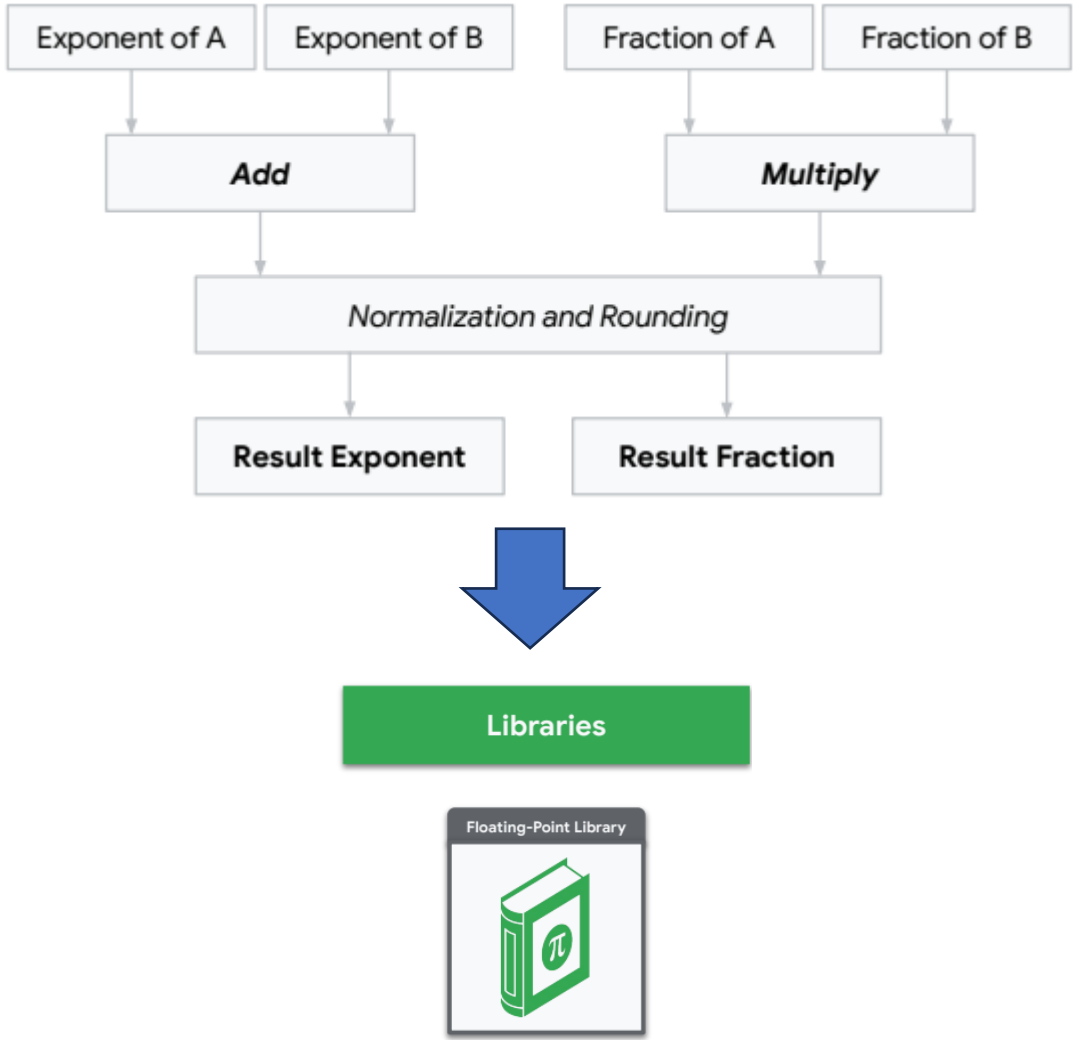
Portabilidade





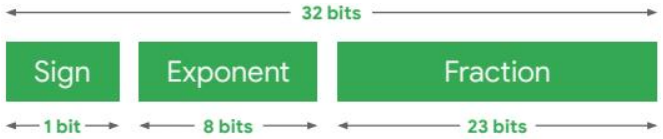
- Vamos supor que em meu código eu tenha o seguinte cálculo:
  - 22/7 = 3.14159265359…
- A resposta é um valor em ponto flutuante, que é tipicamente expresso como 3 componentes: sinal, expoente e mantissa (ou fração).
- Operações com valores em ponto flutuante ***são complexas e computacionalmente custosas***.

Portabilidade

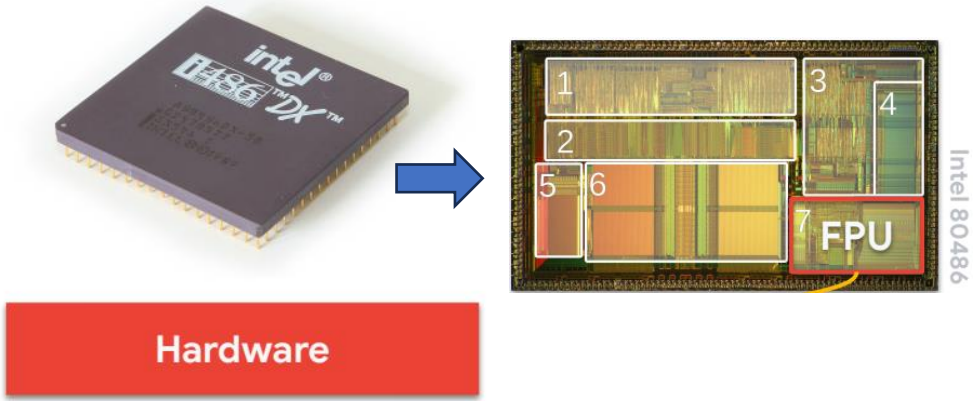


- Como forma de ***abstrair a complexidade dos cálculos e facilitar o uso*** de operações em ponto flutuante, podemos ***implementá-las como uma biblioteca*** que teria funções para adição, multiplicação, etc. destes valores.
- Entretanto, uma biblioteca puramente feita em SW para tal finalidade tende a ser ***muito lenta***.

Portabilidade



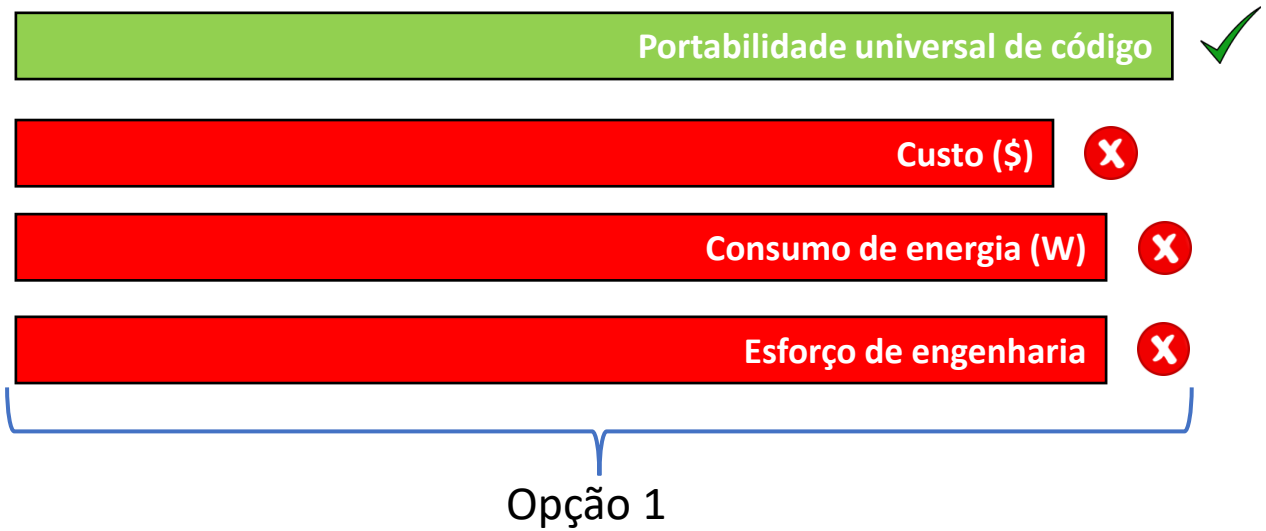
Single Precision
IEEE 754 Floating-Point Standard



Até as CPUs 386, o coprocessador matemático, contendo a FPU, era um *chip* separado instalado opcionalmente em um *socket* da placa mãe.

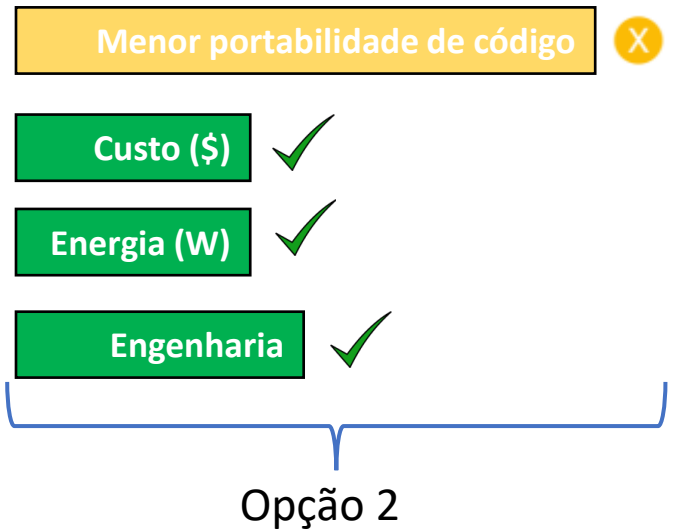
- Ao longo dos anos, para acelerar essas operações, o que ocorreu foi a criação de ***partes da CPU especializadas em operações de ponto flutuante***, as chamadas *Floating Point Units* (FPUs).
- Isso foi ótimo porque desde sua proliferação, praticamente toda CPU tem uma FPU dentro dela.
- Portanto, ***escrevemos o código uma única vez e ele é executado em qualquer lugar***.
- Mas isso expõe um ***trade-off***.

O perde-e-ganha da portabilidade



- A ***portabilidade universal de código têm altos custos*** associados.
- Sua ***implementação custa caro*** ($$$).
- ***Mais energia é consumida*** por esses sistemas de uso geral (W).
  - Por exemplo, a FPU quando usada aumenta o consumo de energia da CPU.
- Necessita-se de ***um esforço de engenharia (i.e., de projeto) grande*** para construí-los e que, consequentemente, ***eleva*** mais ainda ***os custos***.
- Portanto, se queremos portabilidade universal, temos que pagar estes preços.

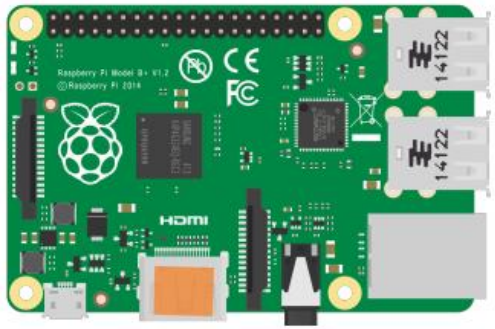
O perde-e-ganha da portabilidade



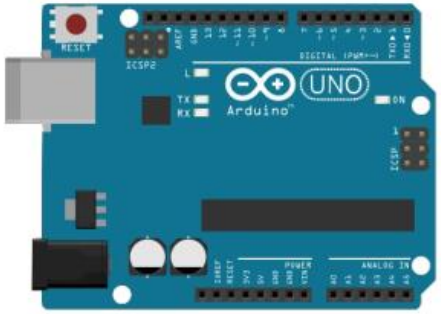
- Agora digamos que queremos trabalhar com sistemas embarcados.
- Eles são por natureza ***baratos***, ***pequenos***, ***eficientes energeticamente***, ***enxutos*** em termos de memória e poder computacional e ***fáceis de serem projetados***.
- Essas características são ótimas para um sistema embarcado.
- Porém, uma implicação direta delas é que a ***preocupação com a portabilidade universal é mínima ou inexistente***.

O perde-e-ganha da portabilidade

Com FPU



Sem FPU

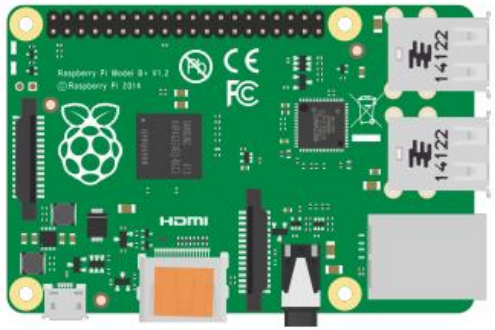


✓	X
Implementação de HW específica de uma biblioteca	

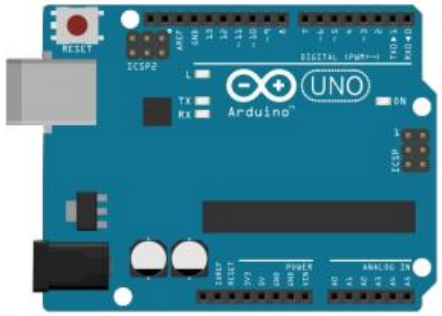
- Por exemplo, o que acontece se quisermos rodar um código em outro sistema e nele não tivermos os recursos que o primeiro sistema tinha e que utilizamos no nosso código, como, por exemplo, uma FPU?
- Outros exemplos de recursos que podem ser exclusivos são:
  - Sensores, atuadores, conversores AD/DA, etc.
- O código não irá ser executado!

O sacrifício da portabilidade

Com FPU



Sem FPU

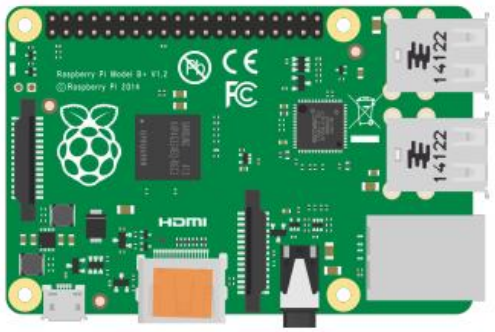


✓	X
Implementação de HW específica de uma biblioteca	

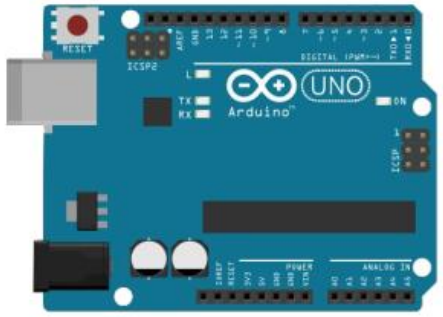
- O código especializado para um sistema embarcado não é portável para outro.
- Devido à limitação de recursos, ***sacrifica-se a portabilidade*** entre sistemas para ***obter-se:***
- ***Otimização do desempenho***: o ***código é adaptado à arquitetura de HW*** específica do sistema, resultando em ***código altamente eficiente e rápido***, ***porém menos portável***.
- ***Otimização de memória***: otimizações para ***economia de memória*** podem exigir ***estruturas de dados específicas e técnicas de programação menos portáteis***.

O sacrifício da portabilidade

Com FPU



Sem FPU



✓	X
Implementação de HW específica de uma biblioteca	

- ***Eficiência Energética*: *otimizar*** o código para ***minimizar o consumo de energia*** pode envolver ***estratégias que não são necessariamente portáveis***.
- Portanto, esse sacrfício levanta uma questão fundamental:
  - ***Como habilitamos o TinyML uniformemente em sistemas embarcados tão diferentes se a portabilidade é baixa ou inexistente?***
- Esse é um desafio crítico e que veremos como abordá-lo ao longo do curso.

Resumo do que vimos até agora

- ***Hardware embarcado*** é ***extremamente limitado*** em termos de ***poder computacional***, ***consumo de energia*** e ***armazenamento***.

Essas ***limitações*** são ***comuns*** em ***sistemas embarcados*** devido ao ***objetivo*** de ***otimização para tarefas específicas*** e ***eficiência no uso dos recursos***.

- ***Software embarcado não é tão portável e flexível*** quanto o software para sistemas de computação de uso geral.

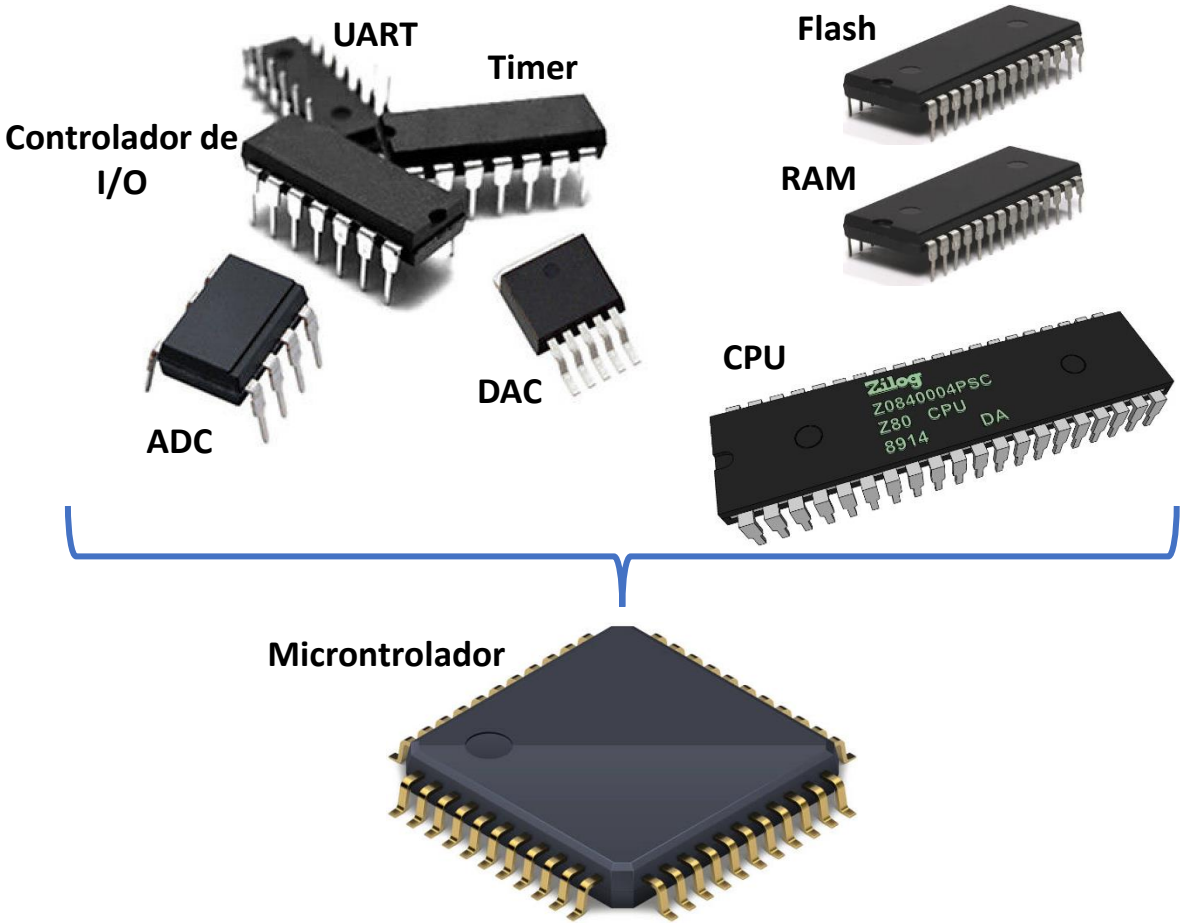
Isso ocorre porque o ***SW embarcado*** é ***altamente adaptado ao HW*** e aos ***requisitos específicos da aplicação***, resultando em uma ***menor capacidade de reutilização*** (portabilidade e adaptação) a diferentes plataformas e contextos.

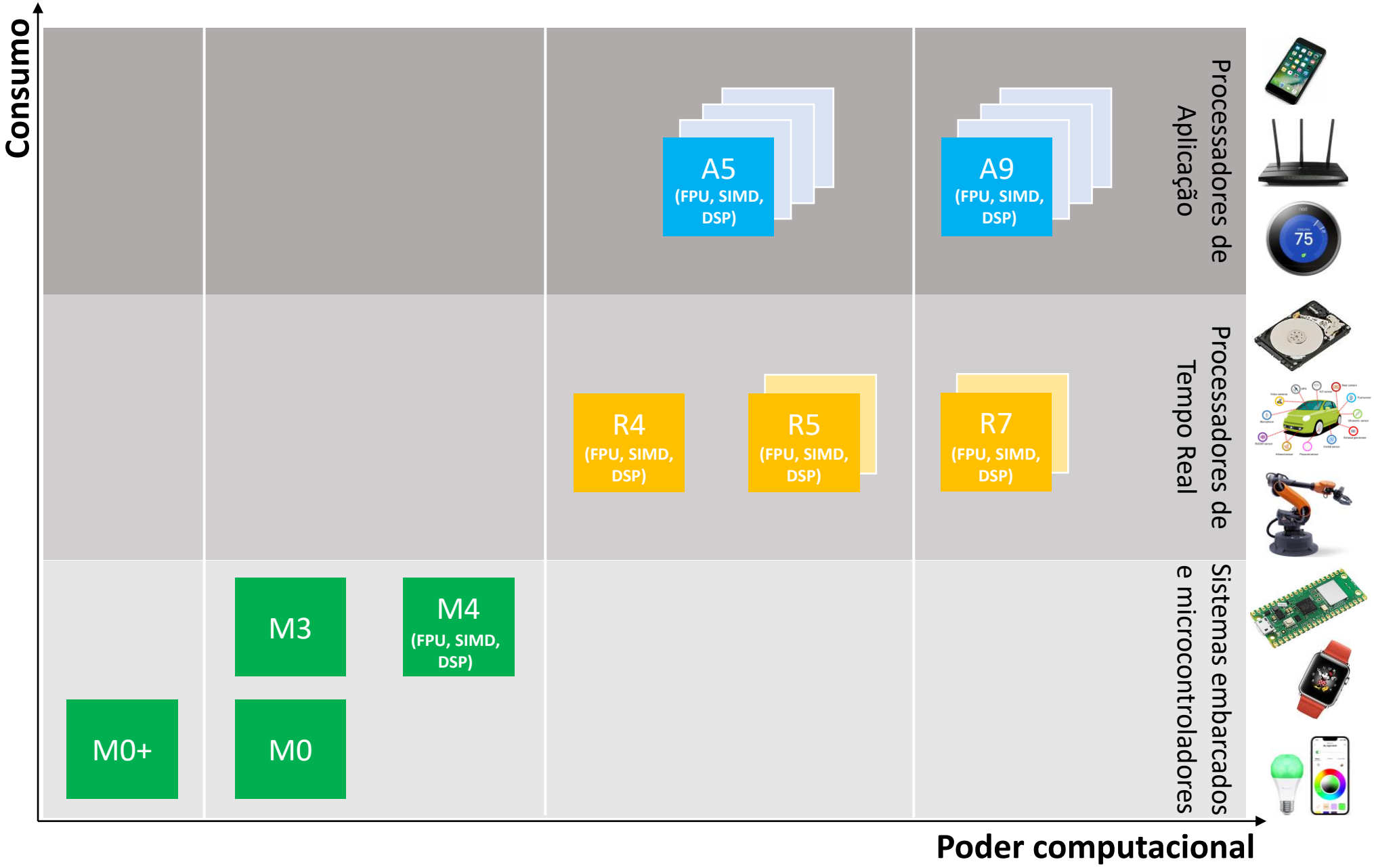
Atividades

- Quiz: “***TP557 – Desafios do TinyML – Sistemas Embarcados***”
- Exercícios de programação em Python.

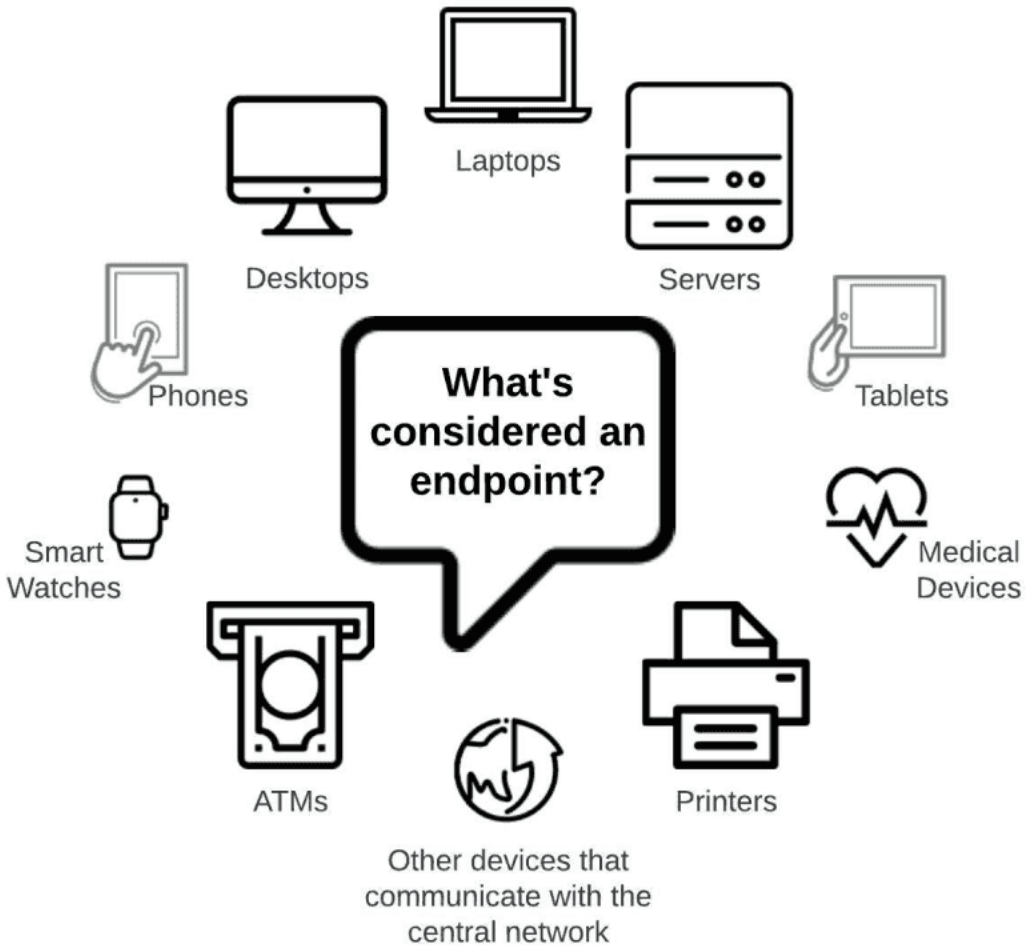
Perguntas?

Obrigado!


UART
Timer
Controlador de I/O
ADC
DAC
Flash
RAM
CPU
Zilog
Z0840004PSC
Z80 CPU
8914
DA
Microntrolador


Consumo
Poder computacional
Processadores de Aplicação
Processadores de Tempo Real
Sistemas embarcados e microcontroladores
A5
(FPU, SIMD, DSP)
A9
(FPU, SIMD, DSP)
R4
(FPU, SIMD, DSP)
R5
(FPU, SIMD, DSP)
R7
(FPU, SIMD, DSP)
M0+
M0
M3
M4
(FPU, SIMD, DSP)

Endpoint



- Um *endpoint* é um dispositivo de computação remota que se comunica com uma rede ou com a Internet.
- Em outras palavras, se um dispositivo estiver conectado a uma rede, ele será considerado um *endpoint*.