

UNIVERSIDADE DE SÃO PAULO
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica e de Computação

SEL0337/SEL0630
PROJETOS EM SISTEMAS EMBARCADOS

Capítulo 10
Desenvolvimento e Customização de Linux
Embarcado

Prof. Pedro de Oliveira C. Junior
pedro.oliveiracjr@usp.br

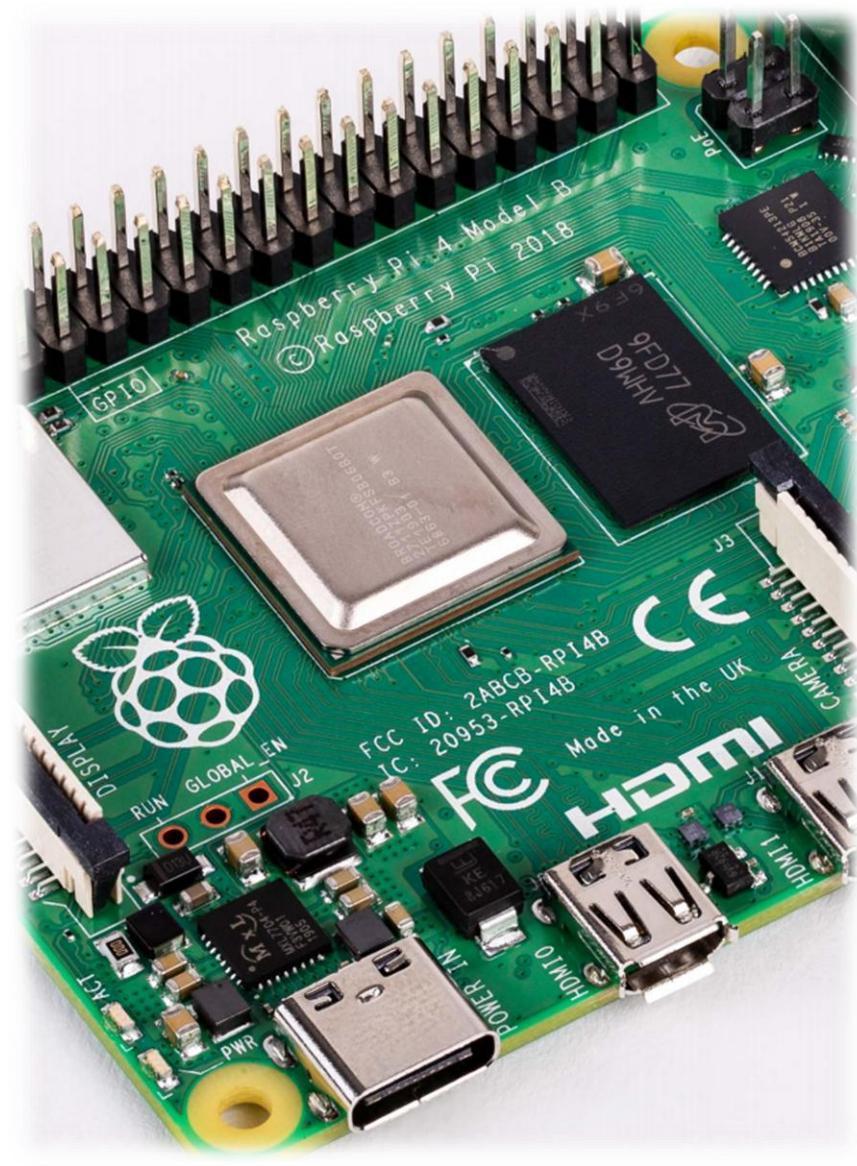
OBJETIVOS



Explorar ferramentas build para criação e customização de distribuições Linux para sistemas embarcados;

Customização do Raspberry Pi para aplicações;

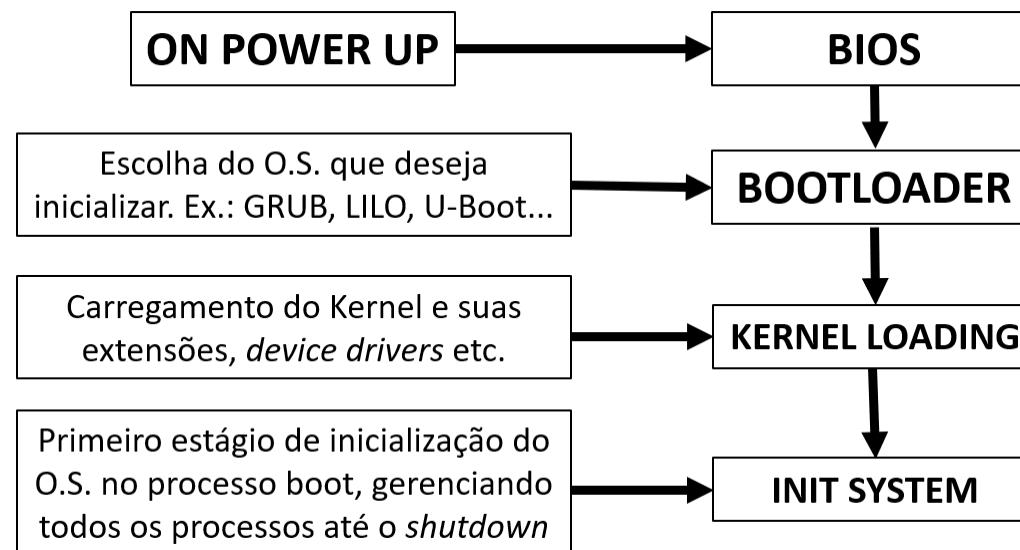
Gerar uma imagem Linux customizada usando um build



Systemd – Gerenciando serviços personalizados em Linux Embarcado

Init System

- ✓ **Boot:** compreende toda a sequência de estágios que ocorrem desde quando o computador é ligado (energizado ou ao pressionar botão ‘ligar’) até a completa inicialização do sistema operacional (O.S.).
- ✓ O **Init System** é o mecanismo responsável pelo processo de inicialização de uma distribuição com o Kernel Linux.
- ✓ Conforme imagem, o **Init System** entra em ação logo após o carregamento do Kernel pelo **bootloader**, durante o processo **Boot**.



System D

- ✓ Enquanto **Init System** é o conceito, o **systemd** é a ferramenta (software) moderna responsável pela inicialização de uma “distro”(distribuição) Linux (existem outras opções: systemv, Upstart, OpenRC).
- ✓ É um conjunto de programas e bibliotecas (é um software livre) **responsáveis pela sequência correta de inicialização e desligamento do sistema operacional (</etc/init.d>).**
- ✓ É usado em distros modernas, tais como RedHat (criadora da ferramenta), Suse, CentOS e Debian.

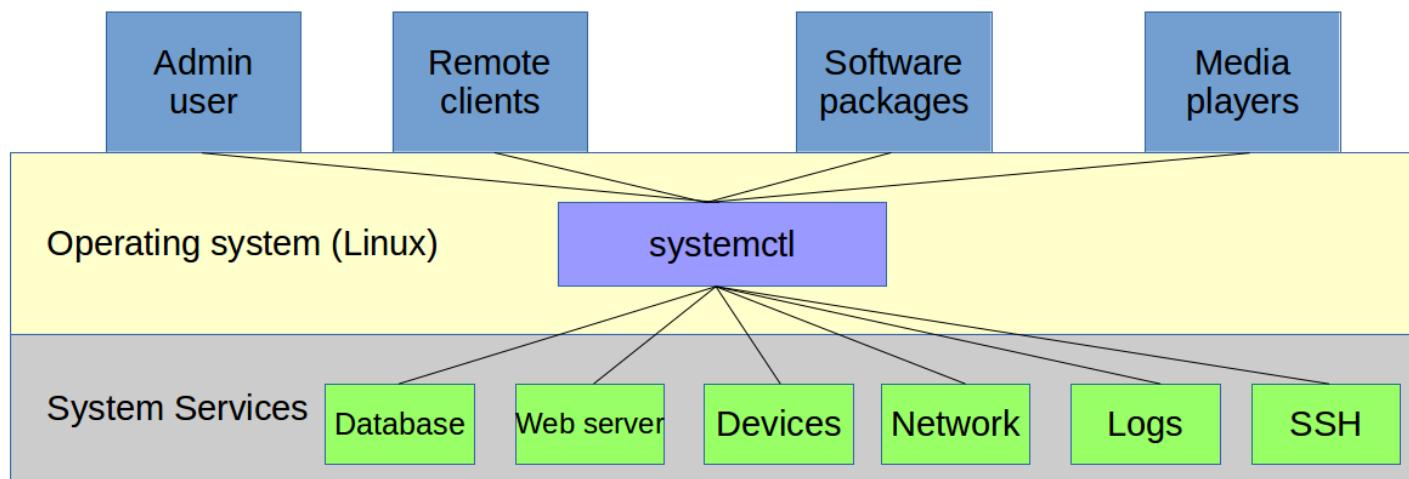
O **systemd** é usado na Raspberry Pi: ao ligar/desligar a placa, **note que aparece a tela abaixo com “Ok” verde em cada operação** no sistema operacional Raspbian, **indicando que o referido serviço foi inicializado (ou finalizado) com sucesso** (a lista de serviços disponíveis pode ser vista em </lib/systemd/system>).



```
[ OK ] Started Bluetooth service.  
[ OK ] Started Raspberry Pi bluetooth helper.  
[ OK ] Reached target Bluetooth.  
      Starting Hostname Service...  
[ OK ] Started Hostname Service.  
[ OK ] Started dhcpcd on all interfaces.  
[ OK ] Reached target Network.  
      Starting Permit User Sessions...  
      Starting /etc/rc.local Compatibility...
```

Comando “*systemctl*”

- ✓ O comando *systemctl* é o gerenciador utilitário de vários serviços do *systemd*
- ✓ É uma linha de comando usada para verificar o status dos serviços inicializados ou finalizados pelo *systemd*,
- ✓ Também exibe mensagens de erro com maiores detalhes, tais como **erros de tempo de execução de serviços, erros de inicialização** (<`sudo systemctl status processoX`>; <`sudo systemctl start/stop/enable processoX`>).



: ® Medium/Harckernon

System D vs. System V

- ✓ **SysVinit** ou **systemv** (“*system five*” / *Unix System V/ SysV*) é o sistema clássico e tradicional de inicialização e gerenciamento de processos em distros Linux;
- ✓ Uma das primeiras versões comerciais do OS Unix.
- ✓ Utiliza *shell script*, invocando um “**daemon binary**” que irá, então, realizar o “**fork**” de um processo em background.
- ✓ A implementação de tarefas é mais complexa no systemv quando se trata de paralelismo;
- ✓ Neste quesito, o **systemd** usa um método moderno mais vantajoso e introduziu conceito “**cgroups**” (grupos de controle).
 - ✓ Organiza os processos por grupos, seguindo uma hierarquia. Por exemplo, quando processos geram outros processos, os “processos filhos” tornam-se automaticamente membros de um “**cgroup** pai”

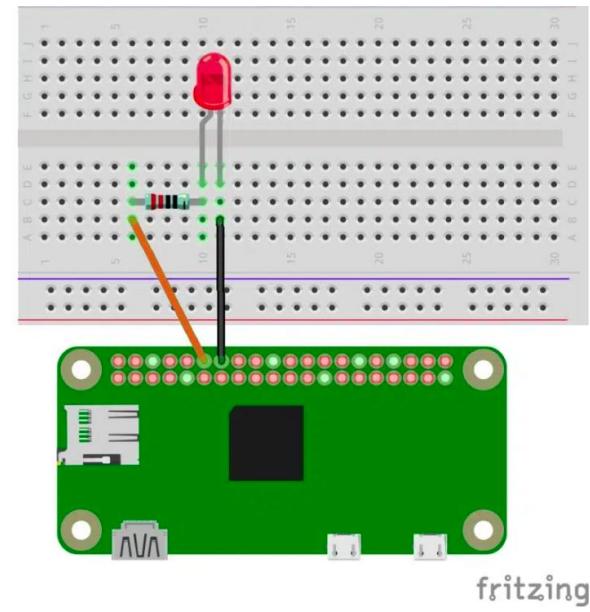
Comparativos:
systemd vs.
systemv

systemv	systemd
Mais antigo e popular <i>init system</i> , seguindo o padrão Unix	Mais recente e moderno <i>init system</i> , usado em várias distros da atualidade (incluindo Raspbian)
Inicialização por meio de arquivos de script <i>shell</i>	Inicialização via arquivos na extensão “ <i>.service</i> ”
API mais antiga	API aprimorada
Mais complexo	Design simplificado com maior eficiência
Inicialização mais lenta e maior uso de memória e hardware	Inicialização mais rápida utilizando pouca memória

Aplicação prática do *System D*

- ✓ A motivação para uma aplicação prática embarcada utilizando recursos do *systemd* se baseia na necessidade de um aplicativo começar a executar sua função já durante a inicialização do OS, sem ter que aguardar alguém “logar” no sistema e inicializar o programa manualmente via terminal.

LAB: vamos supor que uma aplicação deva inicializar imediatamente após o *power on* da Raspberry Pi. Neste caso, nosso serviço a ser inicializado no processo boot será um script que realiza o blink de um LED. Primeiramente devemos criar um script segundo a montagem



Inicializando uma aplicação durante o Boot da Rasp.

LAB - Passo 1 - Escreva um programa para realizar o *blink* de um LED conectado à GPIO da Rasp. Desta vez, vamos fazê-lo em *bash script*, para aprendermos a manipular o acesso direto à GPIO via sistema.

- ✓ No script, é possível observar como ocorre caminho de acesso direto à GPIO, auxiliando na compreensão de como o sistema de arquivos Linux é montado (**rootfs**).
- ✓ Salve o programa como *blink.sh*, que será o serviço que deverá entrar em operação na inicialização da Rasp.
- ✓ Em seguida, é necessário conferir a permissão de execução (“x”) neste arquivo, utilizando o comando “*chmod*”, responsável por alterar permissões.

```
#!/bin/bash

echo 18 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio18/direction

while [ 1 ]
do
    echo 1 > /sys/class/gpio/gpio18/value
    sleep 0.2s
    echo 0 > /sys/class/gpio/gpio18/value
    sleep 0.2s
done
```

```
chmod +x blink.sh
```

```
./blink.sh #executando o script
```

```
# Verifique o caminho de acesso a
GPIO: ls /sys/class/gpio
```

Inicializando uma aplicação durante o Boot da Rasp.

LAB - Passo 2 - criar um *unit file*, i.e, um arquivo responsável por colocar o serviço que criamos à disposição do *systemd*.

- ✓ **[Unit]** - primeira seção responsável pelas informações do serviço e descrição de suas dependências. O comando **Description** entrega essa informação, que será escrita na tela de inicialização do *systemd*, quando da inicialização da Rasp., com “OK” na cor verde se o serviço foi inicializado com sucesso ou “Failed” em vermelho.
- ✓ **[Service]** - configurações da execução do serviço que será inicializado: **Type** (forma como os processos/scripts serão executados); **ExecStart** (arquivo que será executado na inicialização); e **ExecStop** (arquivo que será executado ao parar o serviço). Note que foi dado “*export*” na GPIO 18, conforme consta no script *blink.sh*. Logo, seria possível a escrita de um outro script que poderia efetuar o “*unexport*” da GPIO 18 e, portanto, parando o serviço se informando em **ExecStop**.
- ✓ **[Install]** - descreve o comportamento da inicialização do serviço. O **WantedBy** informa ao *systemd* o grupo alvo no qual o serviço que deverá ser inicializado faz parte.

Salvar como “*blink.service*. Lembrando que o *systemd* executa serviços na extensão “.service”.

O *systemd* é composto por vários destes *unit files* que especificam quais serviços, como devem ser inicializados. assamos esses parâmetros ao *systemd*. Em nosso caso, pedimos para inicializar o *blink.sh*.

```
[Unit]
Description= Blink LED
After=multi-user.target

[Service]
ExecStart=/home/sel/blink.sh
#ExecStop=
user=SEL

[Install]
WantedBy=multi-user.target
```

Inicializando uma aplicação durante o Boot da Rasp.

LAB - Passo 3 - Copie o arquivo “*blink.service*” para o diretório do padrão de serviços do *systemd*, para que de fato seja reconhecido e esteja à disposição no estágio *init system*.

- ✓ Em seguida, initialize o utilitário *systemctl* para testar o serviço
- ✓ Neste caso, o *systemctl start* faz com que o *systemd* execute o serviço informado em **ExecStart** no unit file “*blink.service*”, que em nosso caso é o script “*blink.sh*”.
- ✓ Observe também que não é necessário informar a extensão, pois o *systemctl* comprehende que o arquivo passado é o *blink.service*, pois estruturalmente, o *systemd* executa arquivos com essa extensão.
- ✓ Para parar o serviço, utilize também o utilitário *systemctl stop*:

```
sudo cp blink.service /lib/systemd/system/
```

```
sudo systemctl start blink
```

```
sudo systemctl stop blink
```

Inicializando uma aplicação durante o Boot da Rasp.

LAB - Passo 4 - O próximo e último passo é a habilitação do serviço durante o *Boot* do OS, i.e., até o passo anterior, o serviço que criamos está funcional e à disposição do *systemd*. No entanto, irá funcionar somente ao executar *systemctl start*.

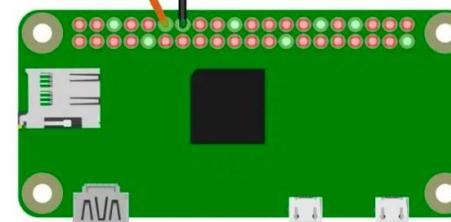
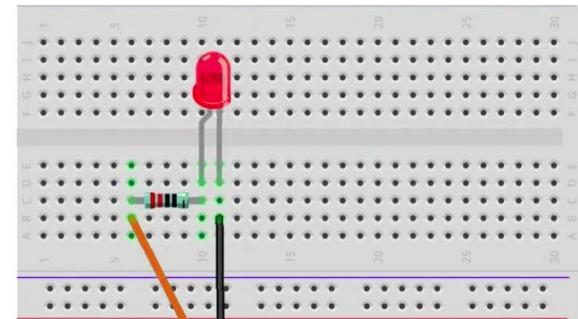
- ✓ Habilite o serviço para que o LED comece a piscar já na inicialização da Rasp. usando a função “enable”.
- ✓ Reinicie a Raspberry Pi para testar a funcionalidade criada. Se necessário, desabilite a opção “*Splash Screen*” em Configurações da Rasp. (*sudo raspi-config - system - splash screen*) para visualizar as mensagens de inicialização.
- ✓ O serviço é identificado pela mensagem que foi digitada em **Description**, no *unit file* “*blink.service*”, com um “OK” verde na frente, conforme resultado abaixo.
- ✓ Para solução de problemas, utilize *systemctl status* e verifique mensagens de erro.

```
sudo systemctl enable blink  
systemctl status blink.service
```

```
[ OK ] Started Blink LED . .
```

Inicializando uma aplicação durante o Boot da Rasp.

LAB - Desafio: Repita o processo acima, criando e testando um novo serviço que deve ser inicializado no Boot da Raspberry Pi pelo *systemd* e que também deve realizar o blink de um LED. Contudo, use um **script em Python** para piscar o LED, ao invés do *bash script* usado no exemplo acima. Observe que neste caso o Python (*/usr/bin/python*) também é um serviço deve ser chamado no *unit file*, pois o *script.py* depende dele para execução em razão de suas bibliotecas.



```
[ OK ] Started Blink LED . . .
```

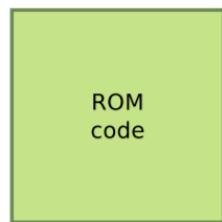
Boot da Raspberry Pi

Sequência de Boot do S.O. embarcado (ARM)

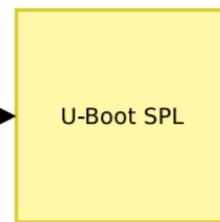
Firmware
gravado na
ROM no
SoC

1º estágio do
bootloader na
partição boot
(SD card/ flash) –
inicialização
básica da RAM

2º estágio do bootloader na
partição boot (SD card/
flash) – inicialização de
drivers e carregamento do
kernel (ex. **firmware U-**
Boot)

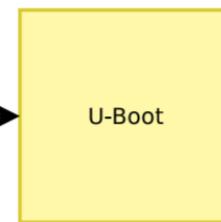


Stored in ROM



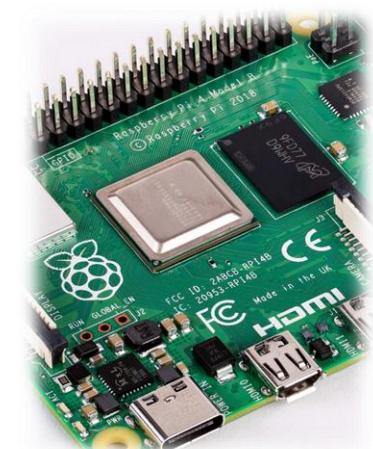
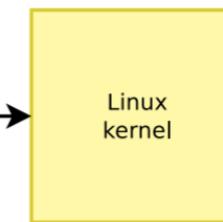
Loaded from a file
called MLO in a FAT
filesystem in the
first bootable
partition

Runs from SRAM



Loaded from a file
called u-boot.img in a FAT
filesystem

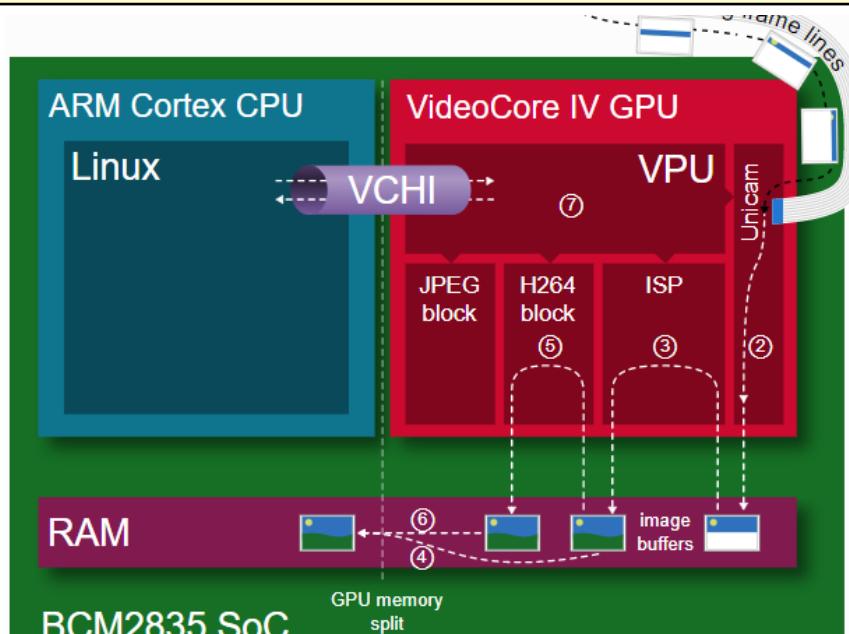
Runs from RAM



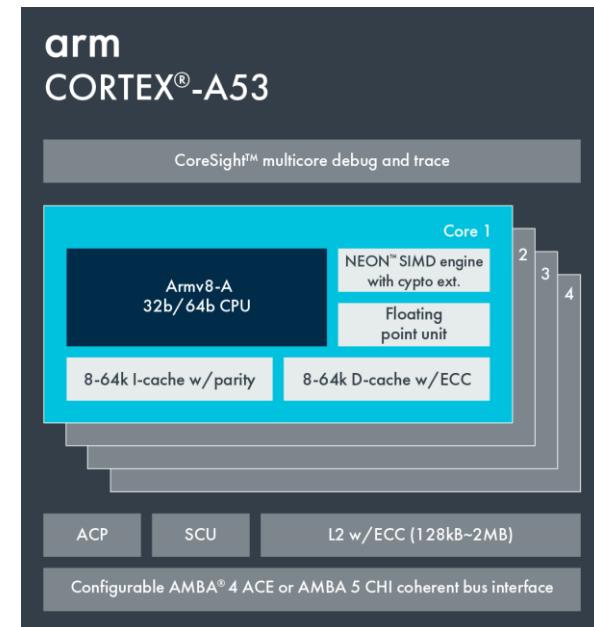
<https://bootlin.com>

O “Boot” da Raspberry Pi

- ✓ Todo o sistema embarcado é comandado pelo SoC (*system on chip*) do fabricante Broadcom
- ✓ Na Rasp.3B+, o modelo é o **BCM2837B0** quad-core Cortex A-53 ARMv8 - [datasheet aqui](#) (série BCM283X – BCM2835, BCM2836 ...)
- ✓ Dessa forma, quando a Rasp. é energizada, o **primeiro estágio do Boot** ocorre neste **SoC**.



ReadTheDocs

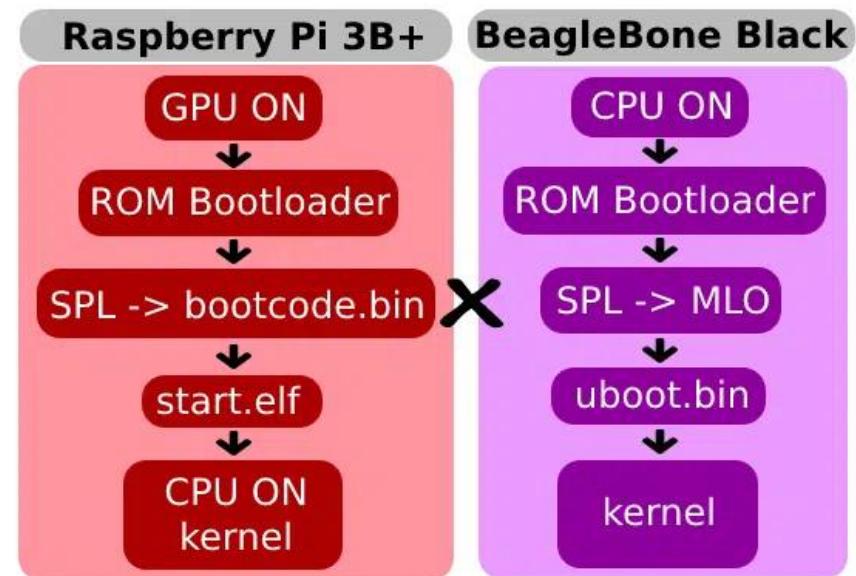


Fonte: [ARM](#)

Boot da Rasp: *Bootloader* – 1º estágio

- ✓ O 1º estágio, imediatamente após o *power on*,
- ✓ Um código a ser executado (buscado pela **GPU**) é armazenado em ROM do SoC, cujo acesso não é disponibilizado pela Broadcom;
- ✓ A função deste código é inicializar alguns clocks do processador e buscar por uma memória externa (cartão microSD inserido na Rasp e com Raspbian instalado).

O detalhamento sobre o *bootloader* é restrito e não divulgado pela Broadcom. Apesar da especificação acima, o **datasheet disponibilizado pela Broadcom não é completo**, e as informações são parciais (o datasheet completo é disponibilizado via **NDA** – “acordo de não-divulgação”).



<https://www.filipeflop.com/blog/boot-pela-rede-com-u-boot-e-raspberry-pi/>

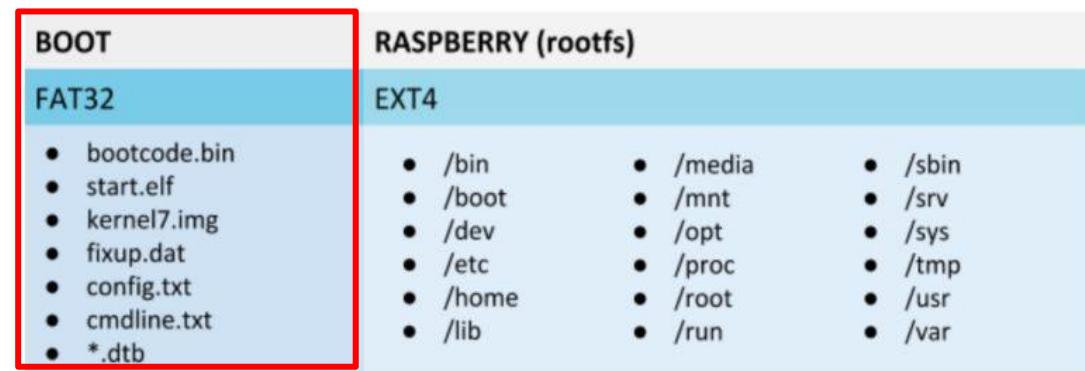
Boot da Rasp: *Bootloader* – 1º estágio

- ✓ Efetivamente, este primeiro código irá buscar na partição Boot do cartão microSD (primeira partição, do tipo **FAT32** - *file allocation table*) um segundo código “**bootcode.bin**”, para carregá-lo para uma memória interna (**cache L2 do SoC**) executar esse arquivo na GPU.

```
#Verifique os arquivos em
cd /boot
ls
ls *.bin

#Visualize o particionamento do cartão
SD com gparted:
sudo gparted /dev/sd1 # sdX - sd1 ou sd2
...
...

# para listagem de discos e mídias:
sudo fdisk -l
lsblk
ls /dev/sdb*
```

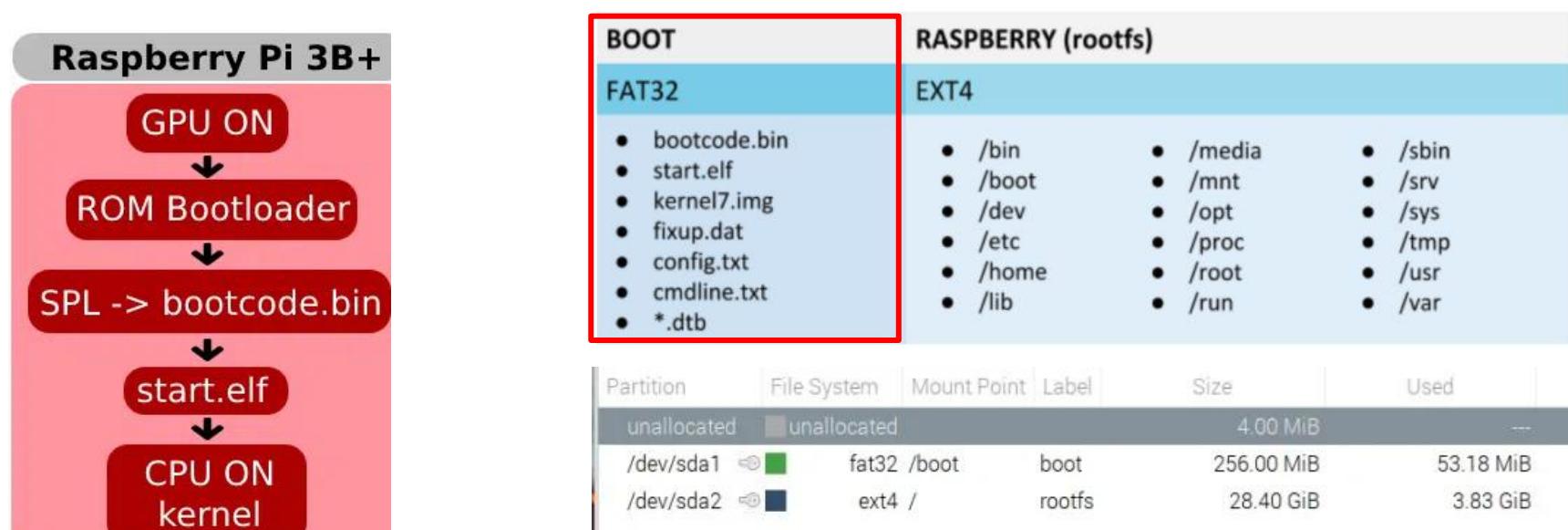


<https://argus-sec.com/raspberry-pi-remote-flashing/>

Partition	File System	Mount Point	Label	Size	Used
unallocated	unallocated			4.00 MiB	—
/dev/sda1	fat32	/boot	boot	256.00 MiB	53.18 MiB
/dev/sda2	ext4	/	rootfs	28.40 GiB	3.83 GiB

Boot da Rasp: *Bootloader* – 2º estágio

- ✓ Ocorre quando o arquivo ***bootcode.bin*** é executado, sua função é inicializar a SDRAM (de 1 GB no caso da Rasp. 3B+) para receber outros arquivos de inicialização.
- ✓ Com a SDRAM inicializada, é também buscado no cartão microSD (na partição boot) um outro arquivo de bootloader chamado ***start.elf***, o qual também é carregado na memória SDRAM.



<https://argus-sec.com/raspberry-pi-remote-flashing/>

Boot da Rasp: *Bootloader* – 3º estágio

- ✓ O arquivo *start.elf* é o firmware da GPU, o qual dá suporte à renderização gráfica.
- ✓ Sua função é buscar os arquivos de device trees (arquivos “.dtb”: compilado de representações do hardware que formam o “device tree”) necessários para carregar o kernel Linux.
- ✓ Sua função também (recebendo ajuda de outro binário: *fixup.dat*) é configurar o hardware de acordo com os parâmetros que constam em um arquivo de inicialização chamado *config.txt* (também localizado na partição *Boot* do cartão microSD).

No arquivo *config.txt* constam: config. do hardware, mapeamento de memória, parâmetros para carregar o kernel Linux.

BOOT	RASPBERRY (rootfs)
FAT32 <ul style="list-style-type: none">• bootcode.bin• start.elf• kernel7.img• fixup.dat• config.txt• cmdline.txt• *.dtb	EXT4 <ul style="list-style-type: none">• /bin• /boot• /dev• /etc• /home• /lib• /media• /mnt• /opt• /proc• /root• /run• /sbin• /srv• /sys• /tmp• /usr• /var

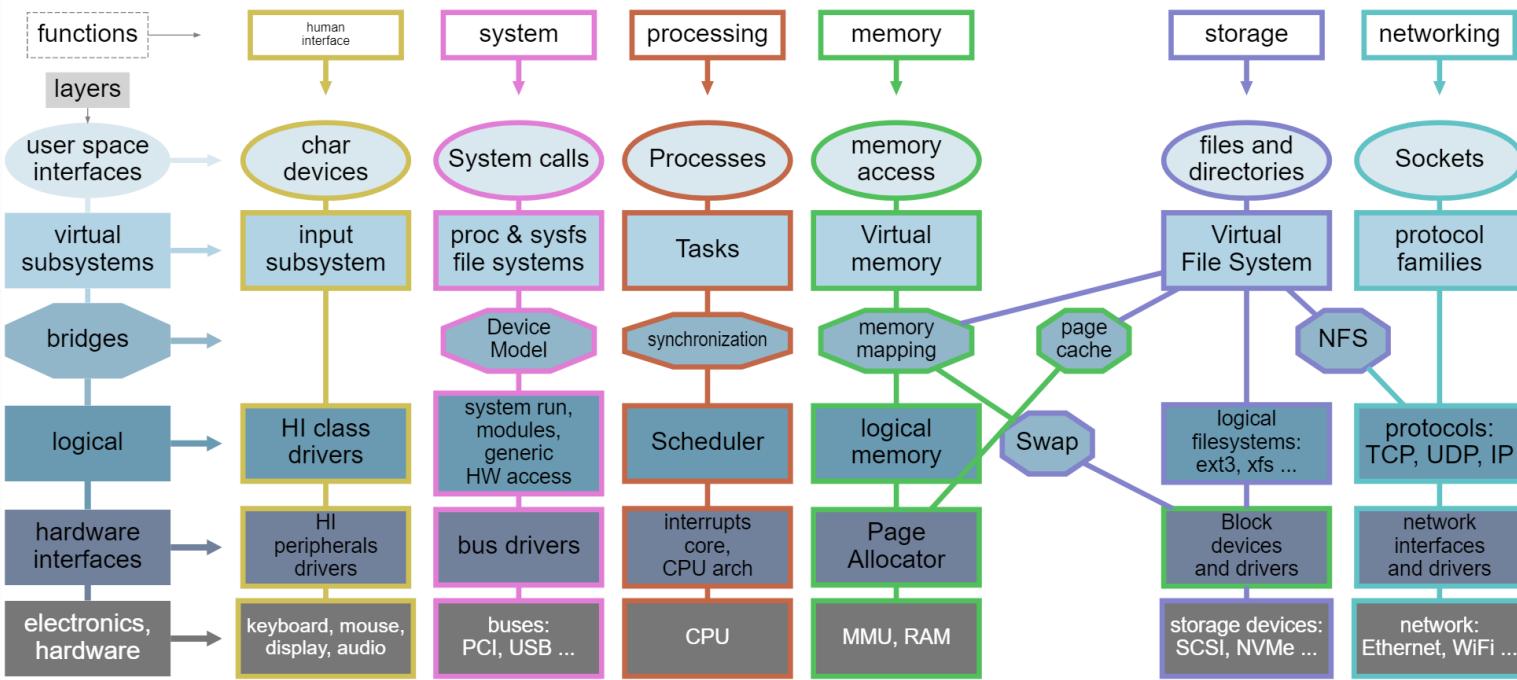
® <https://argus-sec.com/raspberry-pi-remote-flashing/>

Boot da Rasp: Bootloader – 3º estágio

- ✓ Por fim, o código *start.elf* também busca (sempre na partição Boot do cartão micro SD) e carrega a imagem do kernel Linux “*kernel.img*” para a memória SDRAM.
- ✓ A partir daqui, o controle é passado para o kernel Linux conforme parâmetros constantes em um arquivo *cmdline.txt*, e os núcleos da CPU ARMv8 são inicializados e entram em operação para rodar o kernel Linux.

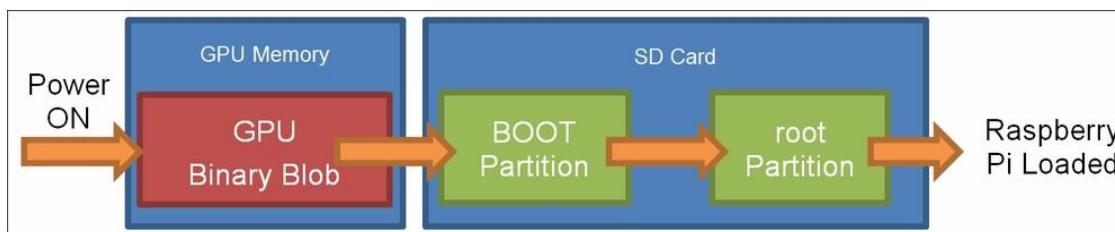
Linux kernel diagram

© 2007-2022 Costa Shulyupin <http://www.MakeLinux.net/kernel/diagram>



Boot da Rasp: Rootfs

- ✓ Com o kernel Linux em execução, o próximo passo é a **inicialização da distro** (no caso da Rasp. o **Raspbian ou Raspberry Pi OS**).
- ✓ Aqui, quem entra em operação é o *init system* visto anteriormente, realizado pelo *systemd*.
- ✓ Neste último estágio, **são carregados os arquivos da segunda partição do cartão microSD, i.e., a partição “rootfs” do tipo “ext4” (extended file system)**, responsável pelo sistema de arquivos Linux seguindo formato root “/” que conhecemos.
- ✓ O *systemd*, por sua vez, como já vimos, inicializa todos os serviços necessários ao funcionamento da distro Linux, os quais ficam em execução até o momento em que o OS é desligado.

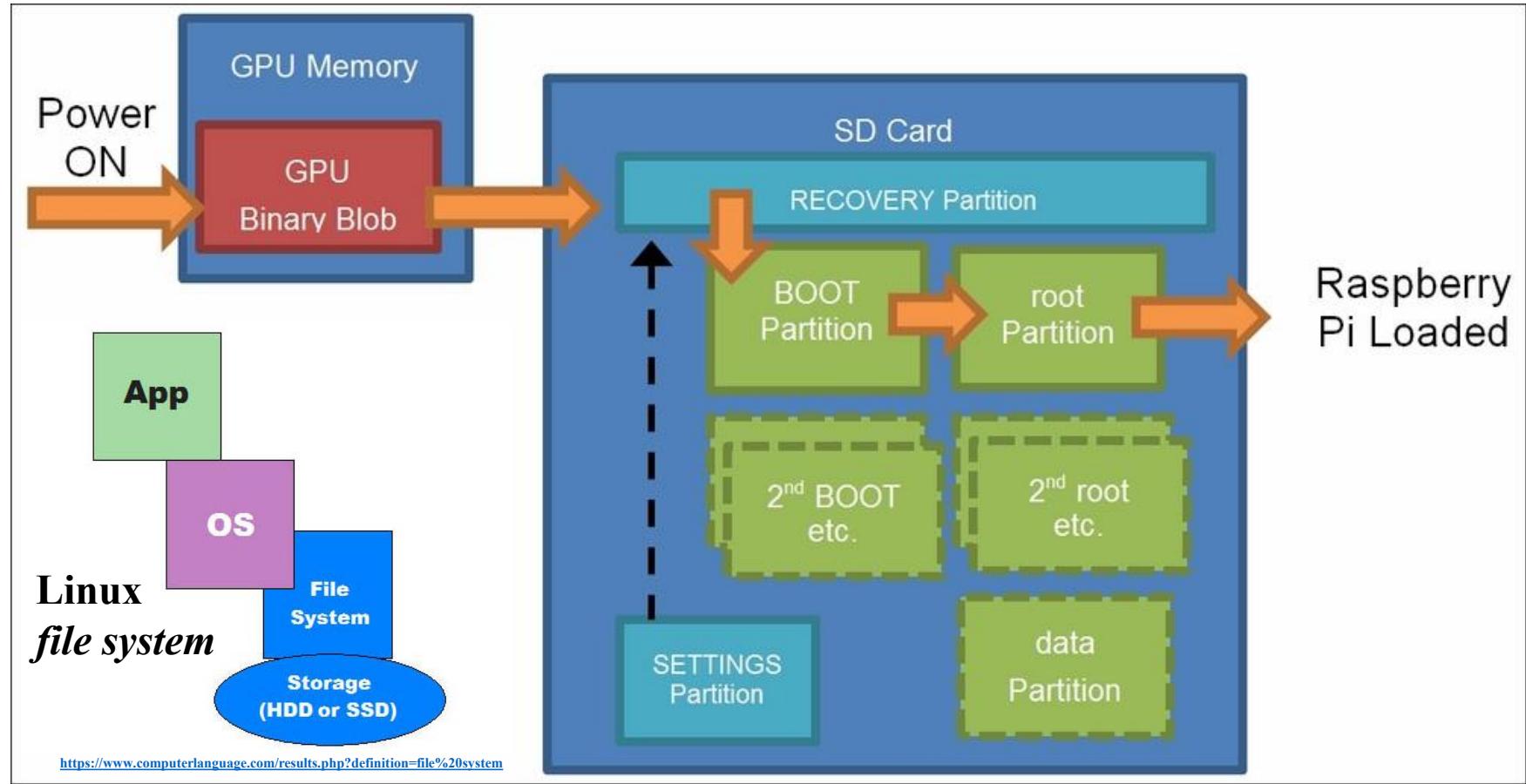


⑧ Packet Pub - <https://subscription.packtpub.com/book/iot-and-hardware/9781785288326/1/ch01lvl1sec12/using-noobs-to-set-up-your-raspberry-pi-sd-card>

RASPBERRY (rootfs)		
EXT4		
• /bin	• /media	• /sbin
• /boot	• /mnt	• /srv
• /dev	• /opt	• /sys
• /etc	• /proc	• /tmp
• /home	• /root	• /usr
• /lib	• /run	• /var

<https://argus-sec.com/raspberry-pi-remote-flashing/>

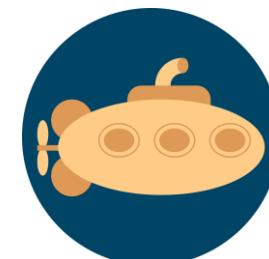
Resumo do “Boot” da Raspberry Pi



Customizando o *Boot* da Rasp: “Das U-Boot”

- ✓ Já sabemos que o *bootloader* da Raspberry Pi não é aberto e, portanto, não é possível customizar o processo de boot da Rasp para ser mais rápido, ou ainda, para realizar o boot diretamente pela rede.
 - ✓ Ex.: Sistemas embarcados em produtos comerciais, por vezes, podem exigir tempo de boot menor.
- ✓ O **Das U-Boot**, um *bootloader* open-source que permite customizar o processo Boot de um sistema embarcado, sendo um dos mais usados em sistemas ARM.
- ✓ Em síntese, ele permite exibir informações do hardware, manipular RAM e memórias flash, **Boot em diversas interfaces (USB, Boot pela rede)**, opera com diferentes tipos de arquivos em interfaces (fat, ext4, gpio, I2C, SPI). **Permite a criação de scripts que otimizam o processo Boot.**

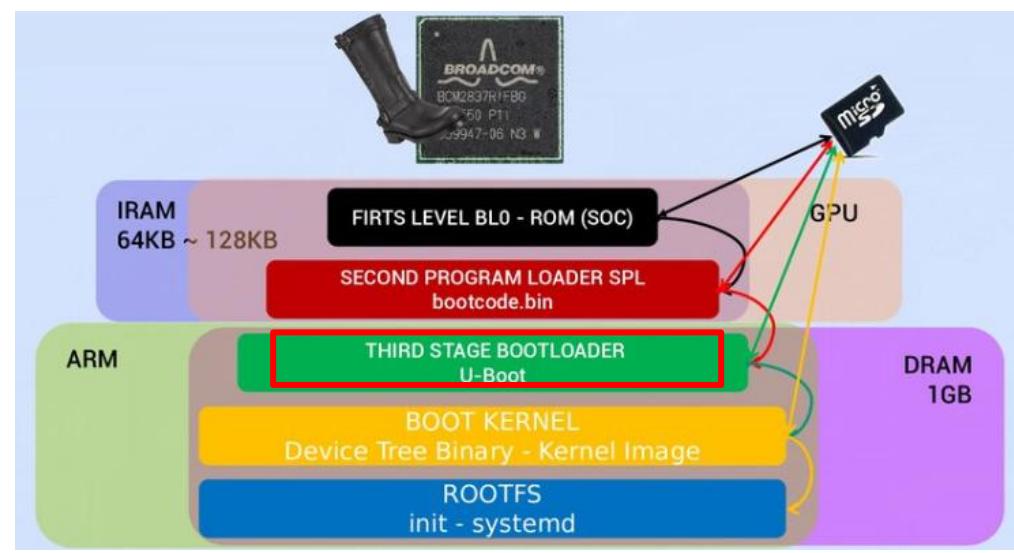
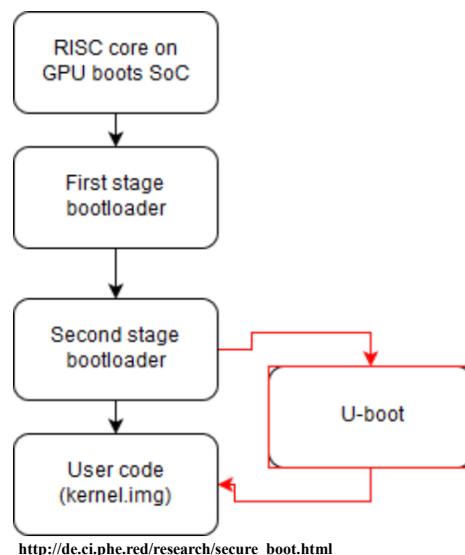
- ✓ DENX Software Engineering
- ✓ Licença GPL
- ✓ Escrito em C/Assembly
- ✓ Usado em ARM, RISC-V, X86



U-Boot

Customizando o Boot da Rasp: “Das U-Boot”

- ✓ A boa notícia é que é possível instalar o U-Boot na Raspberry Pi, e usá-lo como bootloader permitindo explorar suas vantagens.
 - ✓ Permitindo, por exemplo, Boot pela rede, sem usar SD card, o que é mais próximo de um produto com Linux embarcado.
- ✓ Não é possível substituir integralmente o bootloader da Rasp. pelo U-Boot pelas razões já comentadas. Entretanto, é possível fazer com que o bootloader da Rasp. carregue o U-Boot e este último irá carregar o kernel. Será, portanto, uma adaptação no Boot da Rasp.



© Micro Hobby – Matheus Castello - <https://speakerdeck.com/microhobby/linux-embarcado-com-raspberry-pi?slide=12>

Leitura Recomendada

Operating Systems Foundations with Linux on the Raspberry Pi

By Wim Vanderbauwhe and Dr. Jeremy Singer - ARM Education Media -

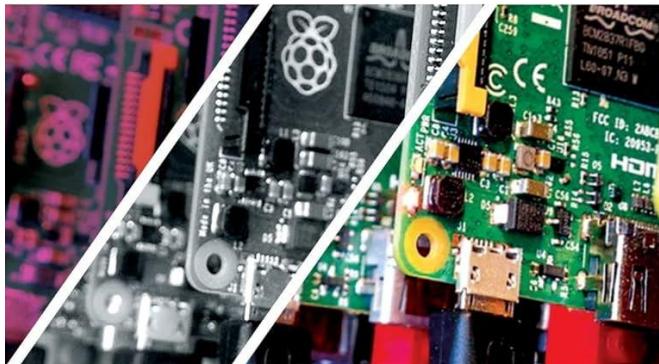
<https://www.arm.com/resources/education/books/operating-systems>

<https://armkeil.blob.core.windows.net/developer/Files/pdf/ebook/arm-operating-systems-foundations-with-linux-on-the-raspberry-pi.pdf>

arm Education Media

Operating Systems Foundations with Linux on the Raspberry Pi

TEXTBOOK



Wim Vanderbauwhe

Jeremy Singer

Operating Systems Foundations

with Linux on the Raspberry Pi



9 781911 531203 >

The aim of this book is to provide a practical introduction to the foundations of modern operating systems, with a particular focus on GNU/Linux and the Arm platform. The unique perspective of the authors is that they explain operating systems theory and concepts but also ground them in practical use through illustrative examples of their implementation in GNU/Linux, making the connection with the Arm hardware supporting the OS functionality. For use in ECE, EE, and CS Departments.

Contents

- 1 A Memory-centric System Model
- 2 A Practical View of the Linux System
- 3 Hardware Architecture
- 4 Process Management
- 5 Process Scheduling
- 6 Memory Management
- 7 Concurrency and Parallelism
- 8 Input / Output
- 9 Persistent Storage
- 10 Networking
- 11 Advanced Topics

"While the modern systems software stack has become large and complex, the fundamental principles are unchanging. Operating Systems must trade off abstraction for efficiency. In this respect, Linux on Arm is particularly instructive. The authors do an excellent job of presenting Operating Systems concepts, with direct links to concrete examples of these concepts in Linux on the Raspberry Pi. Please don't just read this textbook - buy a Pi and try out the practical exercises as you go."

Steve Furber CBE FRS FREng,
ICL Professor of Computer Engineering,
The University of Manchester

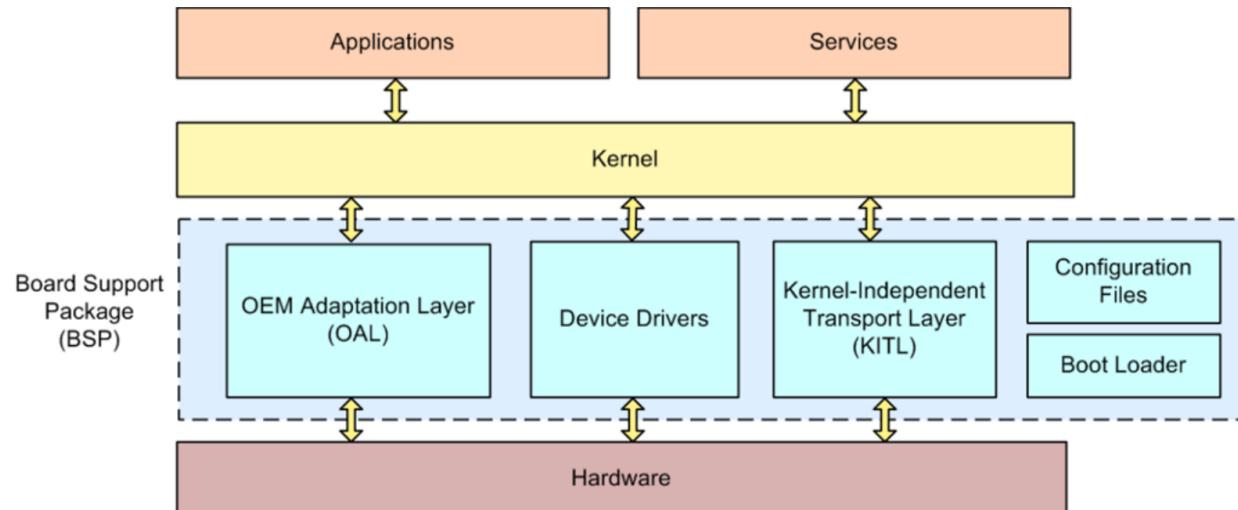
arm Education Media

Arm Education Media is a publishing operation with Arm Ltd, providing a range of educational materials for aspiring and practicing engineers.
For more information, visit: armedmedia.com

Ferramentas build para criação de imagens Linux

Board Support Package (BSP)

- ✓ É um conjunto de software que ajuda um sistema operacional a funcionar corretamente em um hardware específico.
- ✓ Inclui bootloader (que inicializa o sistema), drivers de dispositivos (que permitem ao sistema operacional interagir com o hardware), e arquivos de configuração.
- ✓ Basicamente, adapta o sistema operacional para que ele possa rodar em um determinado dispositivo



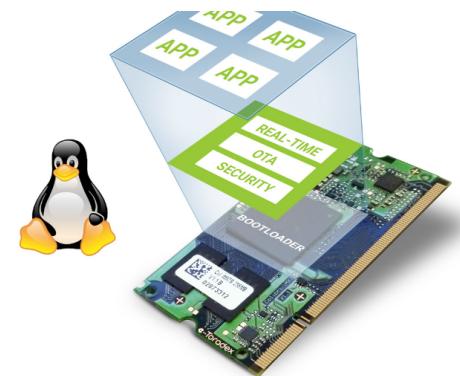
® Microsoft – Windows Embedded Compact

Distribuição Linux para sistemas embarcados

- ✓ Exemplo: Torizon OS – Toradex - <https://www.torizon.io/torizon-os>



Ready-to-Use Industrial Embedded
Linux Distribution



Fast Time-to-Market

Ready-to-use maintained Linux distributions. Simple customization to your Hardware.



Secure by Default

Frequent updates, accessible and simple security features, including Secure Boot and SBOM generation. Integrated OTA and Monitoring Solution.

[Learn More >](#)



Free and Open Source

Based on open software, no lock-in. Portable on almost any Hardware Platform. Customizable to your needs. And it is completely free, including frequent updates for the Toradex System on Modules (SoMs).



Real-Time Enabled

Torizon OS also comes in RT-enabled releases.

[Learn More >](#)

Distribuição Linux para sistemas embarcados

- ✓ **Torizon OS – Layer de referência (Yocto) - <https://www.toradex.com/pt-br/operating-systems/yocto>**



Gratuito e de código aberto

Yocto é um software gratuito e de código aberto

[Saiba mais](#)

BSPs de qualidade de produção

Nossas BSPs são desenvolvidas internamente e são atualizadas constantemente

[Saiba mais](#)

Compável com o Yocto Project

As BSPs de Linux Embarcado da Toradex são totalmente compatíveis com Yocto Project

[Saiba mais](#)

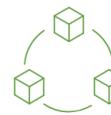
Torizon Ready

Instantly add OTA Remote Updates, Fleet Monitoring, and Remote Access with Zero Setup Security

[Learn more](#)

Suporte de Longo Prazo (LTS)

Nós fazemos uso de software LTS



Totalmente Customizável

Construa seu próprio Linux Embarcado com os componentes que você deseja

[Saiba mais](#)

Distribuições de Referência

Acelere o seu desenvolvimento com uma de nossas imagens de referência

[Saiba mais](#)

Distribuição Linux para sistemas embarcados

- ✓ Explorar o Toradex Developer Center: <https://developer.toradex.com>

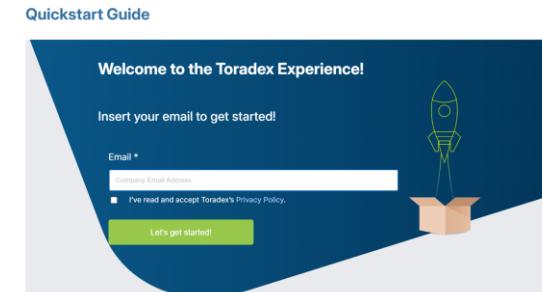


The screenshot shows the main interface of the Toradex Developer Center. At the top, there's a search bar labeled "Search docs". Below it, four main categories are displayed in cards:

- Quickstart Guide**: Contains a brief description and a link to the guide.
- Hardware Offering**: Contains a brief description and a link to learn more about Toradex's hardware offerings.
- Torizon**: Contains a brief description and a link to the software platform.
- Embedded Linux**: Contains a brief description and a link to operating system support.

Below these, a section titled "Get Started with Hardware" lists four product families:

- Verdin Family**: Includes links to System on Modules, Carrier Board, and Accessories & Add-ons.
- Apalis Family**: Includes links to System on Modules, Carrier Board, and Accessories & Add-ons.
- Colibri Family**: Includes links to System on Modules, Carrier Board, and Accessories & Add-on.
- Accessories**: Includes links to Displays, Camera, Networking, and Add-ons.

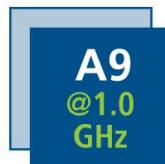


CoM – *Computer on Module*

- Colibri IMX6 e placa base Aster da Torex disponíveis no laboratório (doação)



<https://www.toradex.com/pt-br/computer-on-modules/colibri-arm-family/nxp-freescale-imx6>



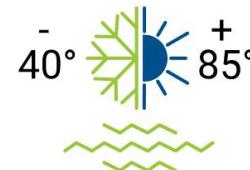
Movido pelo i.MX 6 da NXP



Pronto para multimídia



<https://www.toradex.com/pt-br/products/carrier-board/aster-carrier-board>



Robusto e confiável

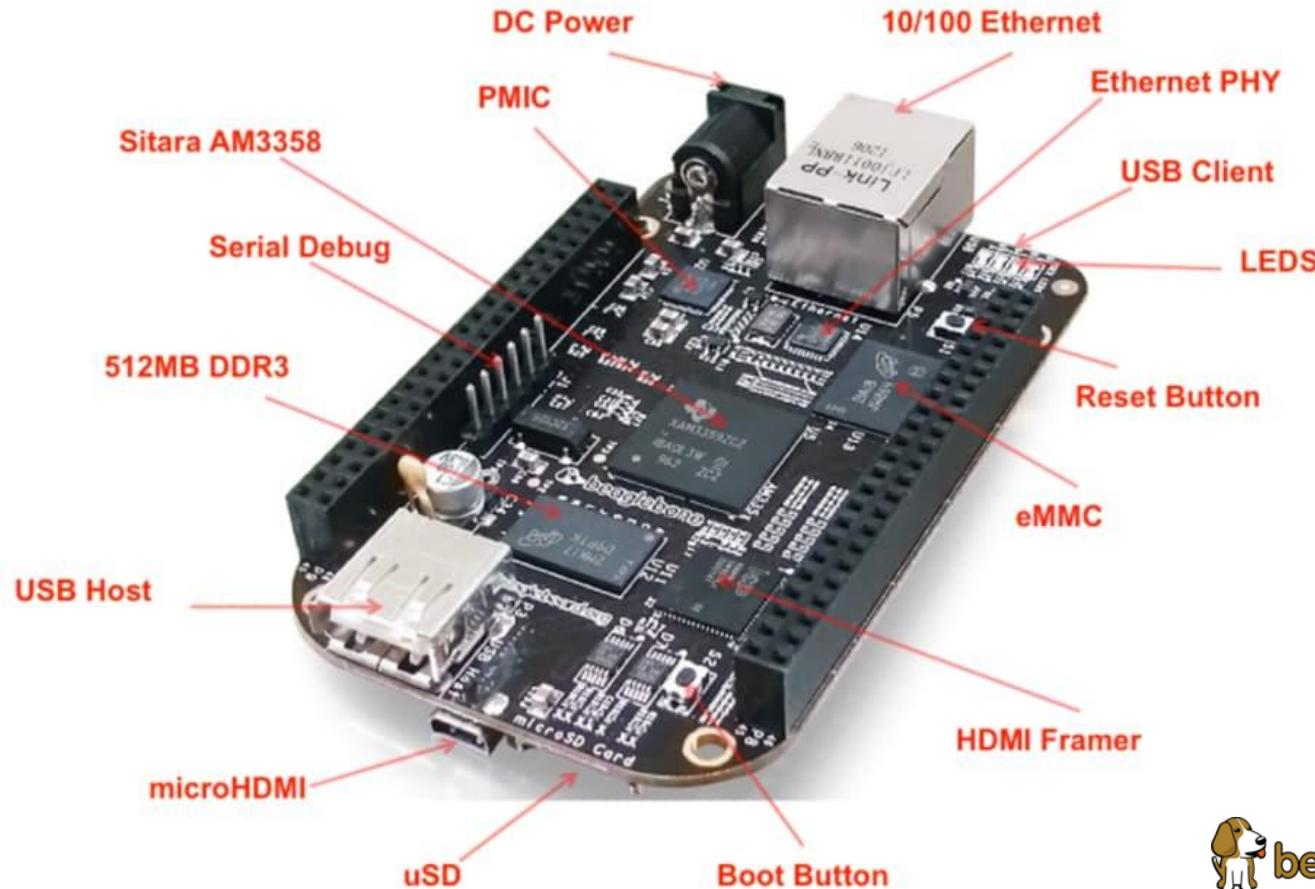


Torizon para atualizações remotas e monitoramento de frota



SBCs - BeagleBoard

- Placa Beagle Bone Black - <https://www.beagleboard.org/boards/beaglebone-black>

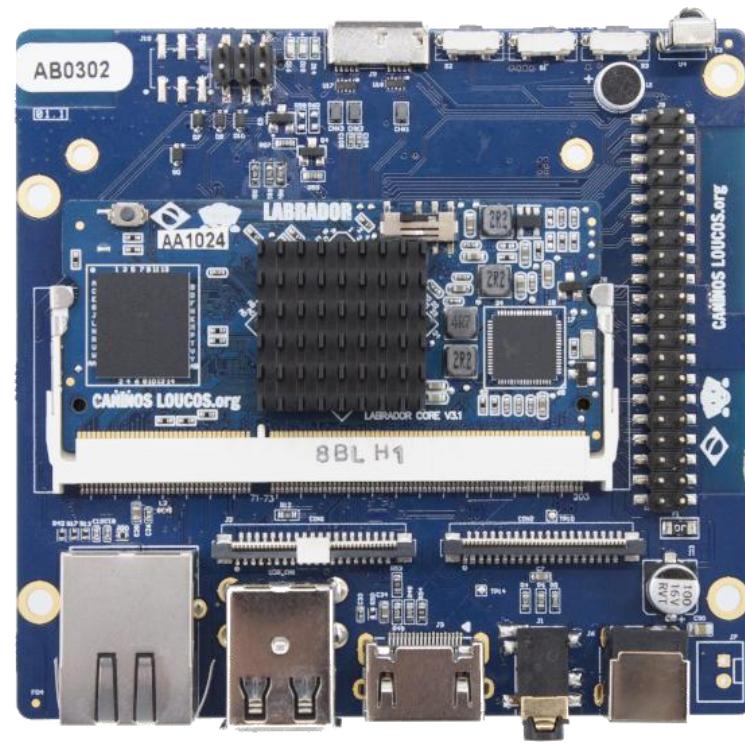


SBCs - Labrador

➤ Projeto brasileiro!

- ✓ Iniciativa do Laboratório de Sistemas Integráveis Tecnológico (LSI-TEC) com o apoio da Escola Politécnica da Universidade de São Paulo (Poli-USP)
- ✓ <https://caninosloucos.org/pt/>

**Versões de 32 e 64 bits, CPU quad-core
ARM Cortex A53, 1,3 GHz, GPU,
memória RAM DDR3 de 2GB e placa
base com chip Wireless e conexões na
imagem ao lado.**

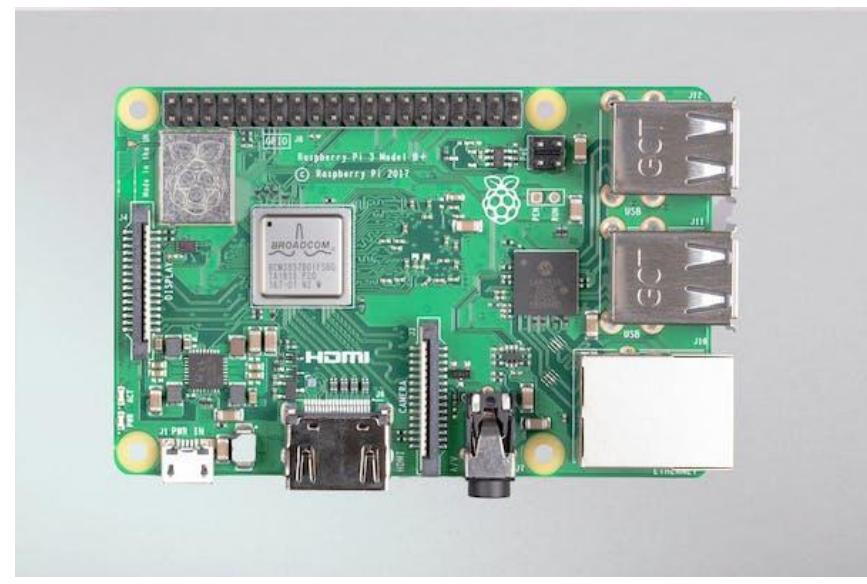
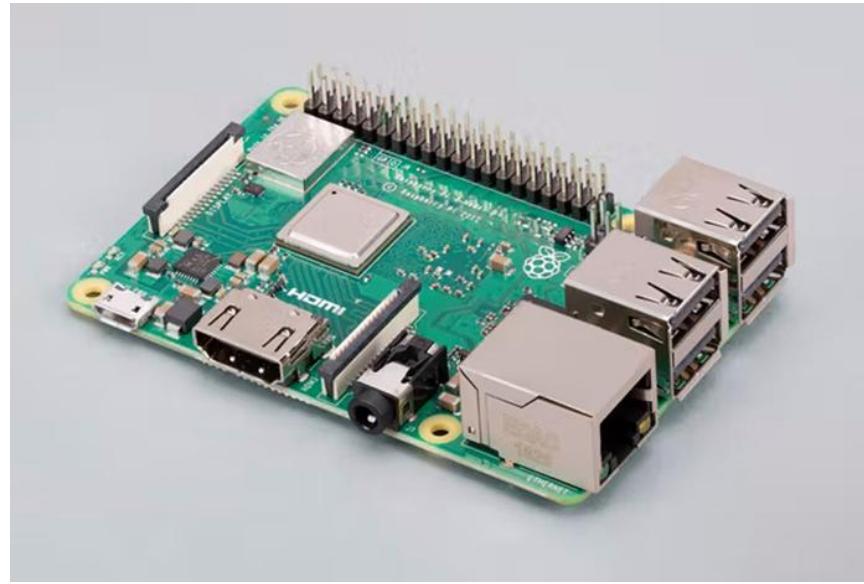


Gerações, modelos e evolução

➤ RASPBERRY PI 3 B+ (3^a GERAÇÃO)

Aprimoramento do modelo “3B” com **novo SoC BCM2837B0 quad-core Cortex A-53 ARMV8 64-bit de 1,4 GHz, com Ethernet de 300 Mbps, Wi-Fi e Bluetooth 4.2.** Lançada em 2018!

- Manteve RAM de 1GB, 4 x USB e 1 x HDMI.



<https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/>

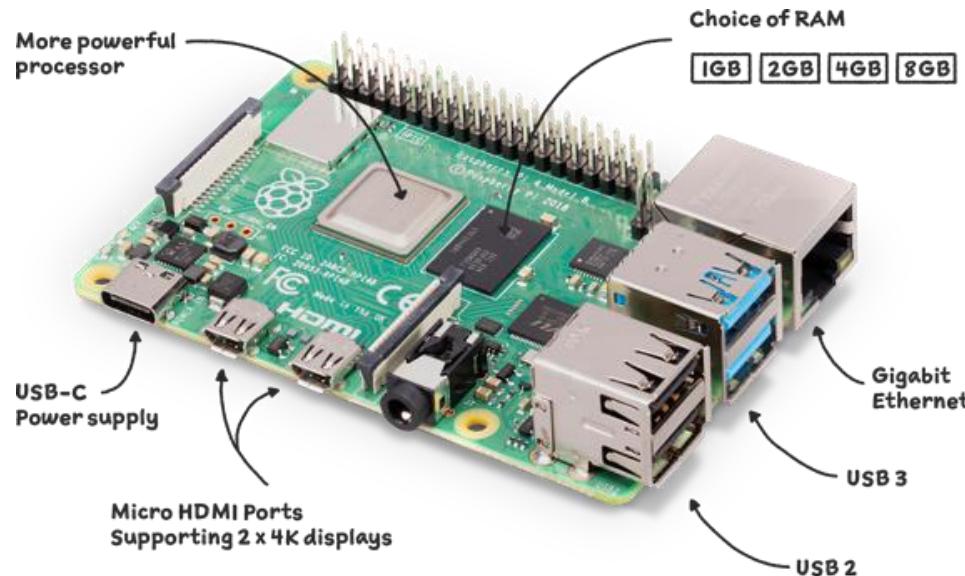
Gerações, modelos e evolução

➤ RASPBERRY PI 4 B (4^a GERAÇÃO)

A versão mais completa atualmente, com novo SoC BCM2711 quad-core Cortex A72 ARMv8 de 64-bit, em versões de 1,5 ou 1,8 GHz (dependendo das versões com 1, 2, 4, e 8 GB de RAM). Lançado em 2019.

- Com 4x portas USB (2x no padrão 3.0), 2x micro HDMI, microSD, Ethernet, Wi-Fi e Bluetooth 5.0..

<https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>



Gerações, modelos e evolução

➤ RASPBERRY PI 400 [4^a GERAÇÃO]

Computador em forma de teclado, possui um RPi 4B na versão de 4GB de RAM, SoC BCM2711C0 quad-core Cortex A72 de 1,8 GHz, GPU videoCore VI de 500 MHz. Lançado em 2020.

➤ Com 3x USB, 2 x micro HDMI, microSC, Ethernet, Wi-Fi e Bluetooth 5.0.



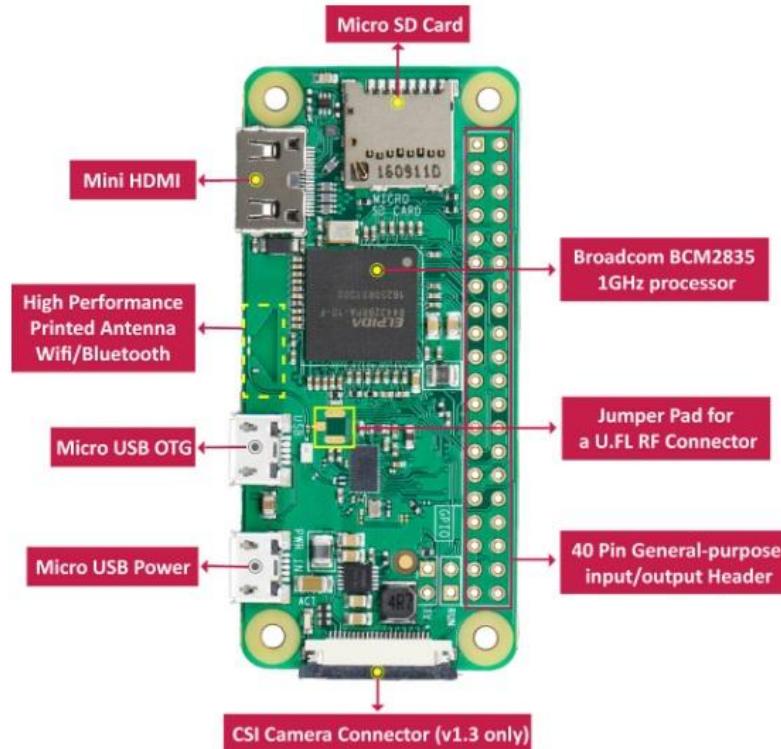
<https://www.raspberrypi.com/products/raspberry-pi-400/>



Gerações, modelos e evolução

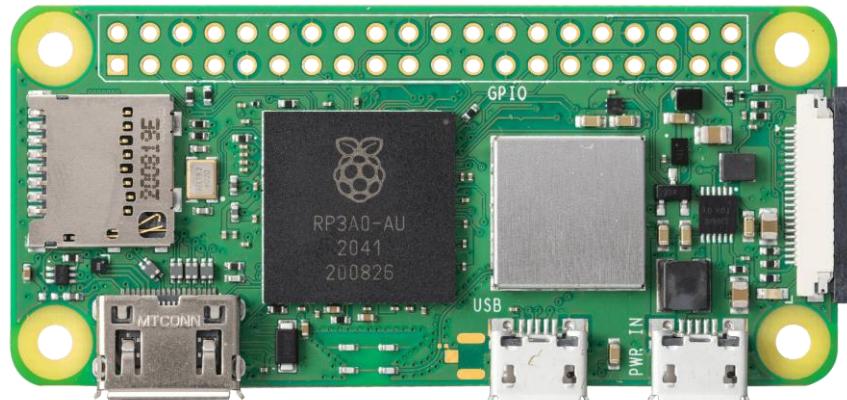
➤ RASPBERRY PI 2W (4^a GERAÇÃO)

- Modelo Raspberry Pi Zero mais recente, **lançado em 2021**, 5x mais rápida do que o modelo RPi Zero W.



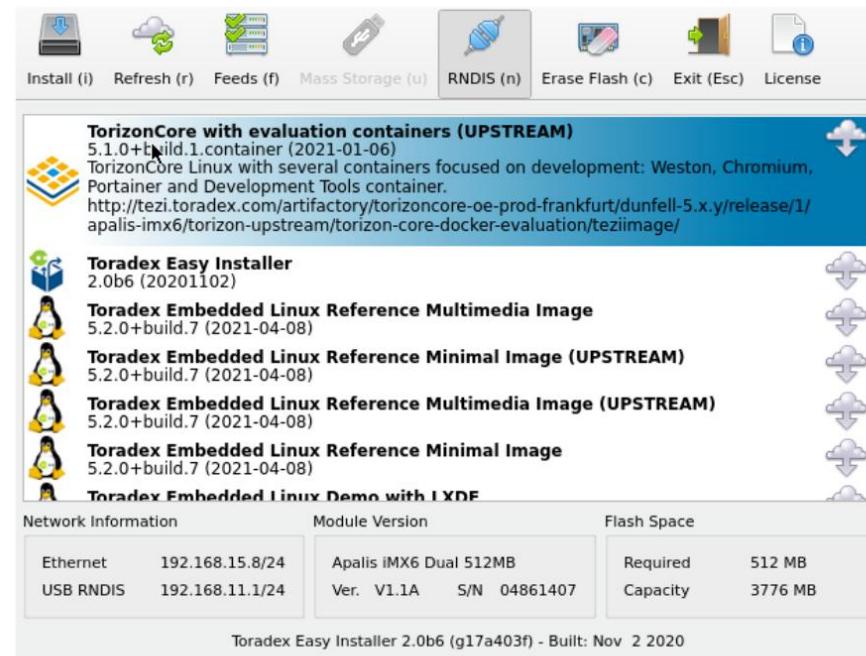
Com SoC **BCM2710A1 quad-core Cortex A53 de 1 GHz, GPU IV 400 MHz**, 512 MB de RAM, 1x microUSB OTG, 1 x mini HDMI, Wi-Fi, Bluetooth 4.2 e microSD.

<https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/>

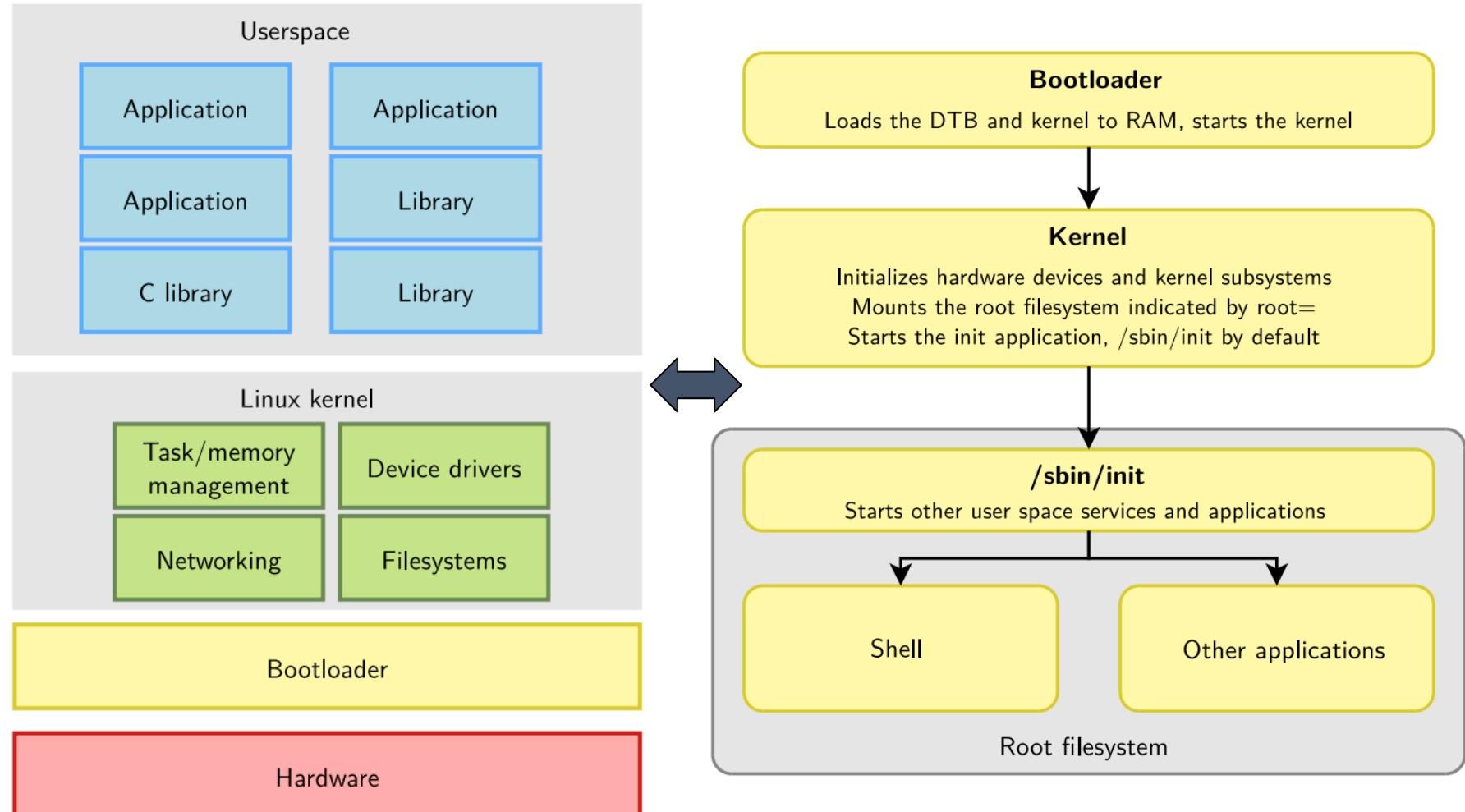


Torizon OS

- Para trabalhar com placas da Toradex ou instalar o Torizon OS na Raspberry Pi
- <https://developer.toradex.com/torizon/>



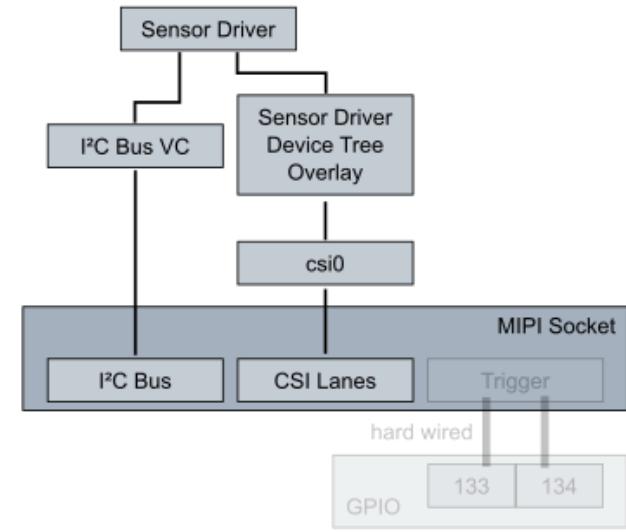
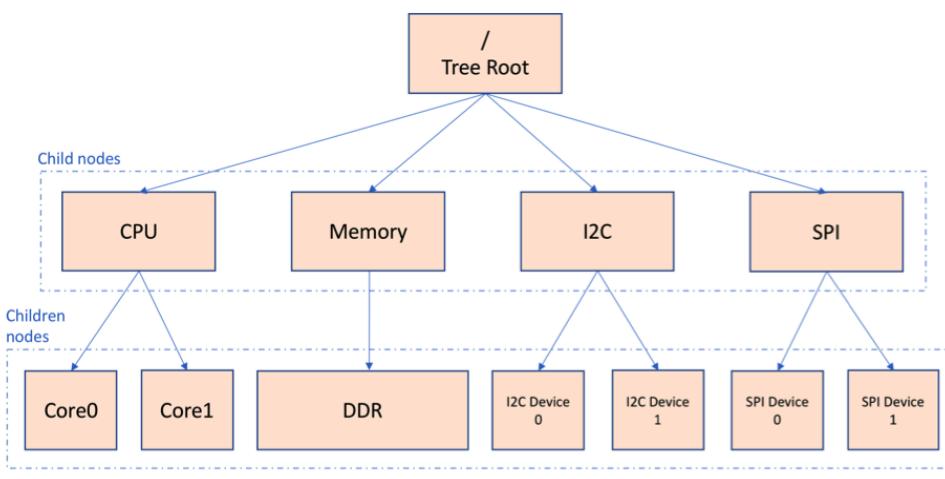
Arquitetura de um sistema embarcado com Linux



Fonte: ® Bootlin - <https://bootlin.com/doc/training/buildroot/buildroot-slides.pdf>

Arquitetura de um sistema embarcado com Linux

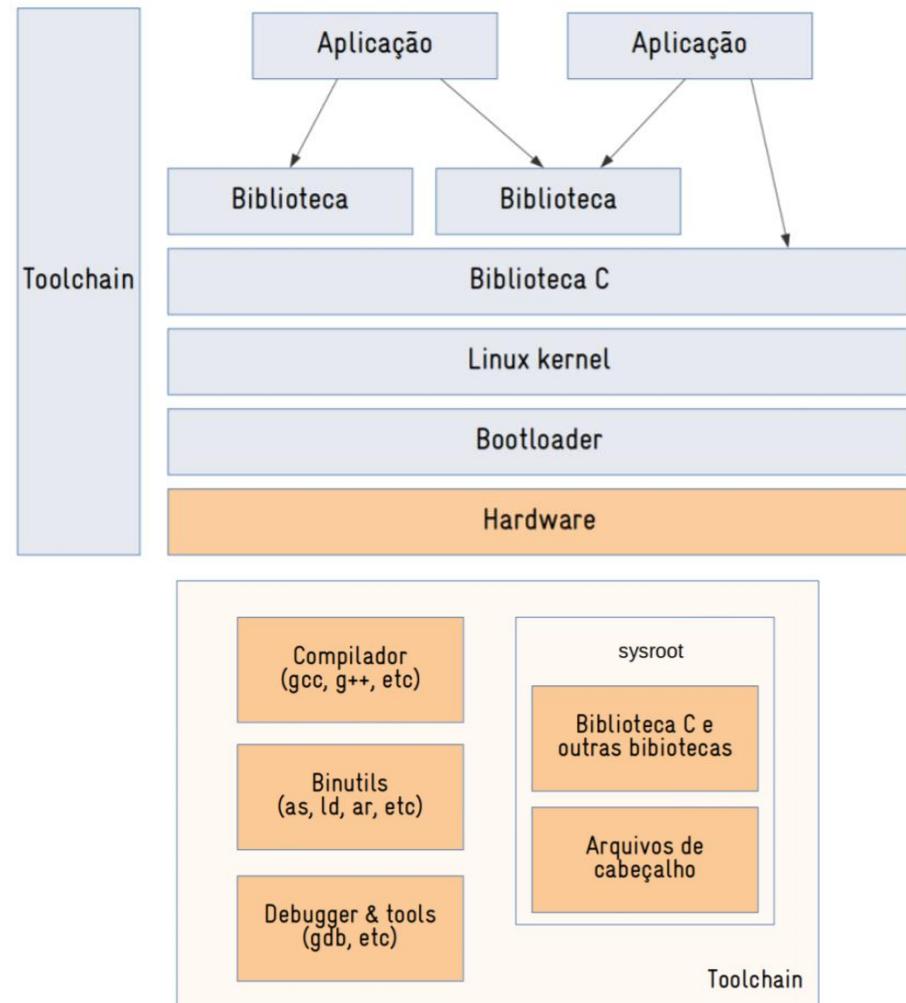
- ✓ **Device tree:** estrutura de dados que descreve/mapeia a configuração do hardware no sistema operacional. Essa informação é usada pelo Kernel, durante o Boot, para reconhecer, carregar e gerenciar os drivers apropriados dos dispositivos de hardware.
- ✓ **Linux Device driver:** biblioteca de rotinas para manipulação em baixo nível de informações contidas na memória para o funcionamento e reconhecimento de dispositivos de hardware no sistema Linux.
- ✓ **Biblioteca C Standard:** interface entre Kernel e aplicações do usuário (funções implementadas em C: I/O, manipulação de strings, compressão, tool-kit gráfico, códigos de erros etc.). GNU C Library (glibc)



* https://www.vision-components.com/fileadmin/external/documentation/hardware/VC_MIPI_Raspberry_PI_CM4IO/index.html

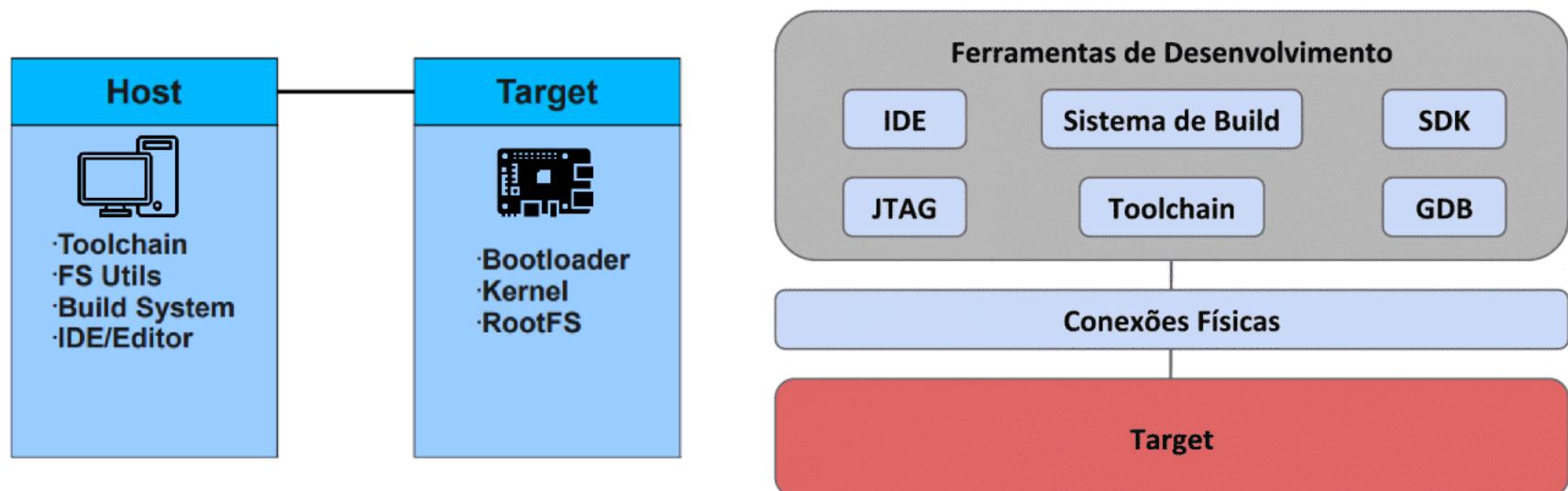
Desenvolvimento de um Sistema Operacional (Linux)

- ✓ **Hardware:** plataforma alvo (target).
- ✓ **Bootloader**
- ✓ **Kernel Linux:** núcleo do sistema operacional. Gerencia CPU, memória e I/O, exportando serviços para a camada de aplicações.
- ✓ **Rootfs:** sistema de arquivos principal (camada de aplicações do usuário).
- ✓ **Biblioteca C:** API do sistema operacional, interface entre o kernel e as aplicações.
- ✓ **Bibliotecas e aplicações do usuário.**
- ✓ **Toolchain:** conjunto de ferramentas para manipular e gerar os artefatos do sistema operacional (bootloader, kernel, rootfs), conforme a arquitetura (ARM, X86...)



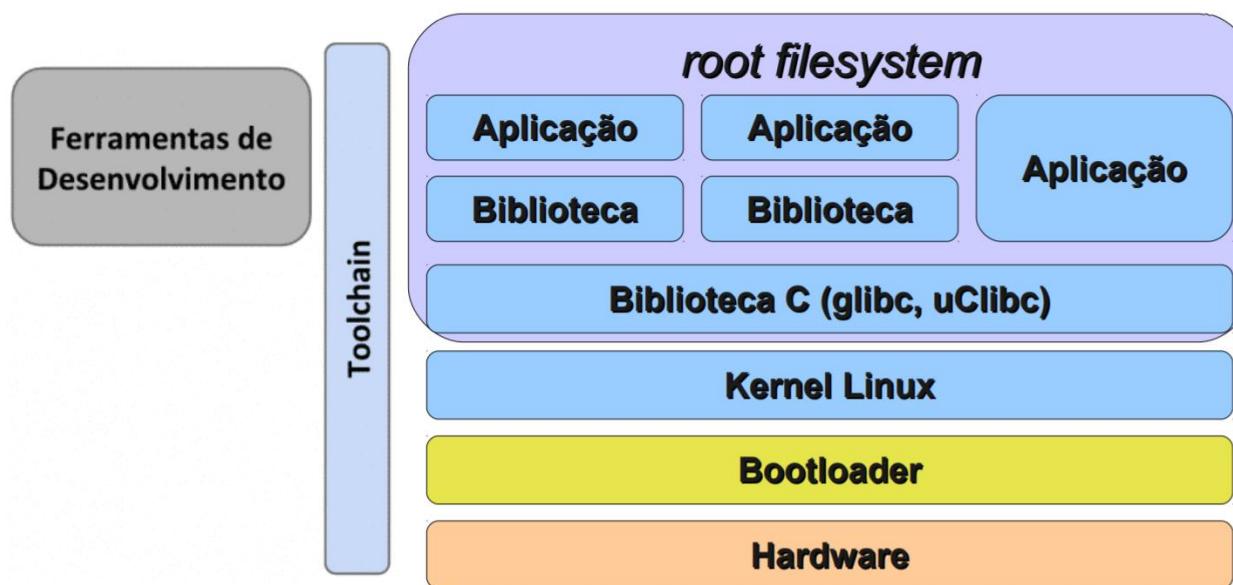
Ambiente e ferramentas desenvolvimento

- ✓ Dentre os elementos já comentados anteriormente (**bootloader**, **kernel**, **rootfs**), é importante destacar que o hardware é nosso alvo (**target**). O ambiente de desenvolvimento, **onde a distro é construída**, é geralmente um computador com OS Linux (**host**). Ao final, os arquivos gerados são transferidos ao target.
- ✓ As ferramentas de desenvolvimento são responsáveis por gerar a estrutura do sistema operacional Linux, como: **IDE**, **build system** (vista a seguir), **SDK** (kit de ferramentas, libs e source codes), **JTAG** (inc-circuit emulator para debug de código on-chip), **Toolchain** (vista a seguir), **GDB** (GNU debugger), **conexões físicas** (serial/I2C/SPI, Ethernet, USB etc.).



Construção do sistema com Linux embarcado

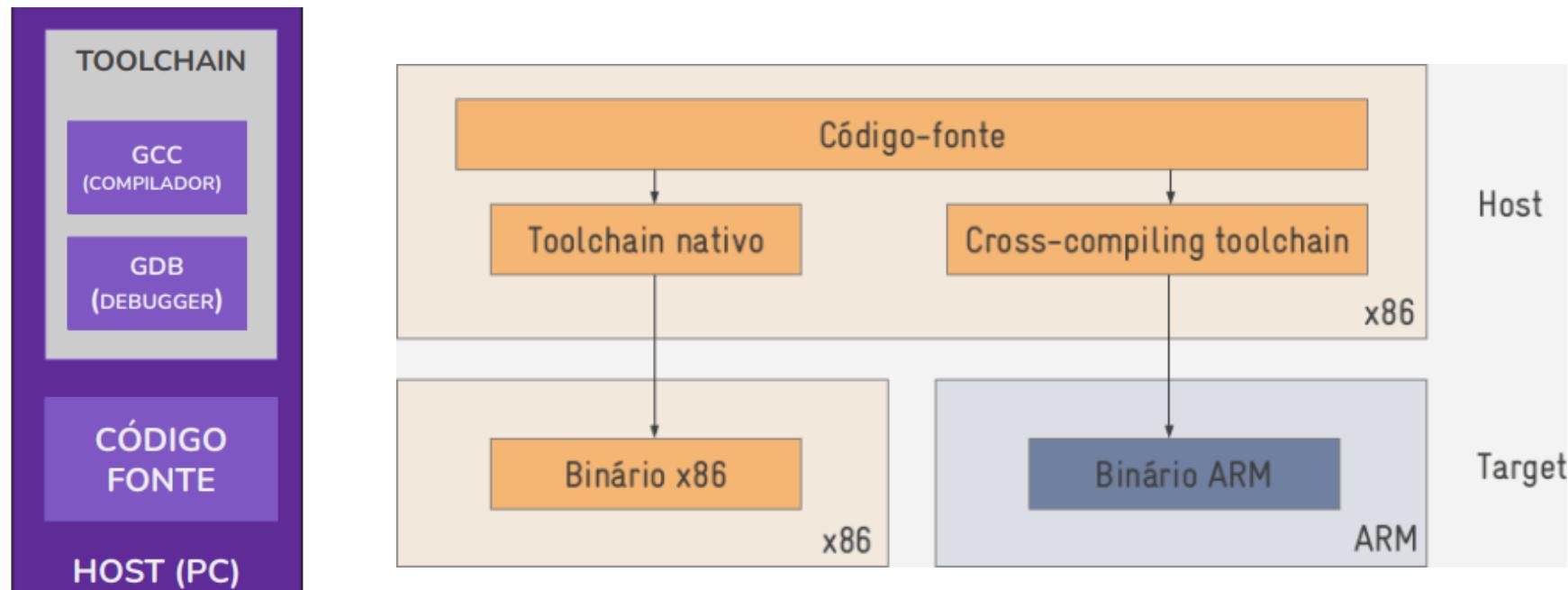
- ✓ **Toolchain:** conjunto de ferramentas de compilação para gerar os binários que estruturam o sistema operacional no target: bootloader, kernel, rootfs.
- ✓ Por exemplo, no processo de compilação de um código em C, é necessário pré-processamento (código C intermediário), compilação (conversão para assembly), arquivo objeto, e Linker (binário final, firmware, aplicação).
- ✓ Esse é o papel da toolchain, que integra todas essas funcionalidades, incluindo uma biblioteca C padrão (predominante em sistemas Linux - **GNU C library - glibc**).



® BoardWare/ Embarcados - <https://www.embarcados.com.br/wp-content/uploads/filebase/eventos/IIIssemccppse/Introducao-ao-Linux-Embarcado-IIIssemCCppSE.pdf> / <https://embarcados.com.br/yocto-project-introducao/>

Toolchain e compilação cruzada

- ✓ **Cross-compiling toolchain (compilação cruzada):** quando a host possui arquitetura diferente da target (ex.: X86-64 para ARM). A diferença entre Toolchain nativa e *cross-compiling* é ilustrada abaixo.
- ✓ Um exemplo de Toolchain bastante usada em sistemas embarcados é o projeto [Linaro](#).



B2Open Systems CC BY-SA 4.0- <https://www.b2open.com/treinamentos/le/B2OpenSystemsTreinamentoLinuxEmbarcado.pdf>
Embedded Labworks CC BY-SA 4.0 - <https://e-labworks.com/training/lem/slides.pdf>

Customização de um sistema com Linux embarcado

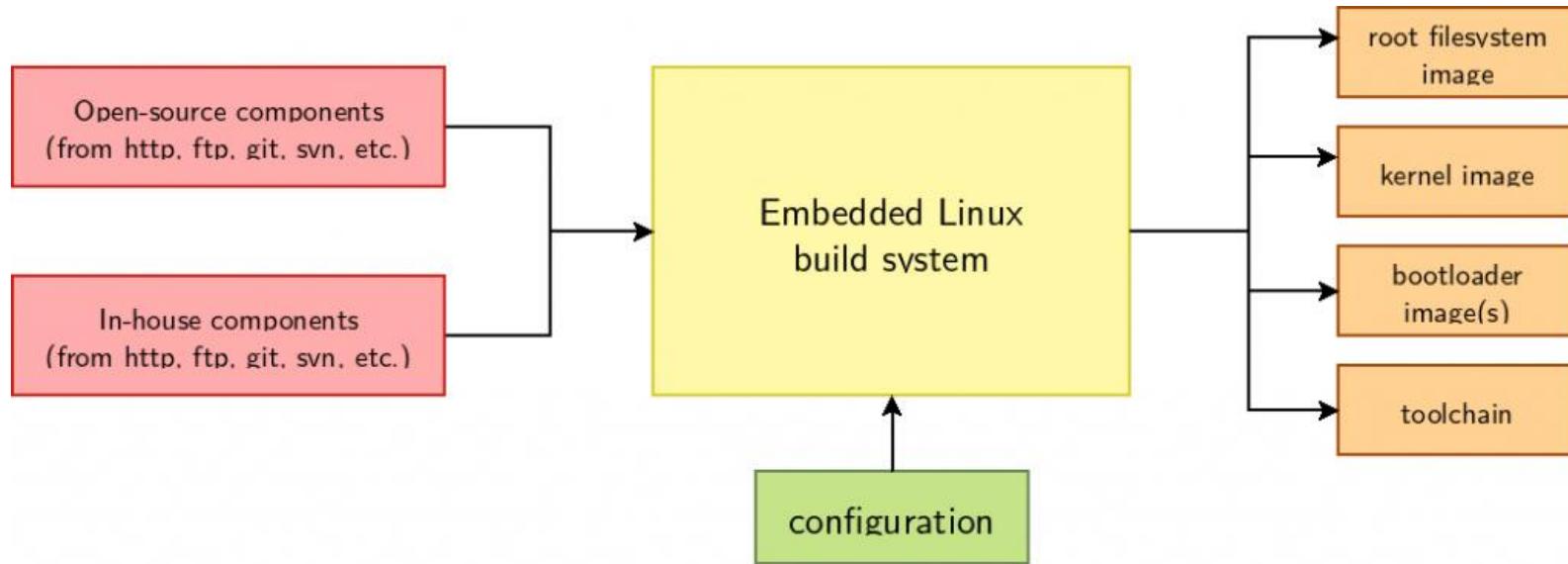
- ✓ O processo de desenvolvimento de um sistema com Linux embarcado depende de requisitos funcionais da aplicação: **o que a placa deve controlar? Quais as condições ambientais? Necessidade de tempo real? Quais interfaces externas?**
- ✓ Ademais, a seleção da plataforma de hardware (target) adequada também depende diretamente das condições acima **e da relação custo vs. funcionalidades vs. suporte disponível.**
- ✓ As 3 principais opções são: **distro pronta (binary distrib.) vs. Build systems vs. Distro do zero (manual).** Vide vantagens e desvantagens na tabela abaixo.

	Pros	Cons
Building everything manually Debian, Ubuntu, Fedora, etc.	Full flexibility Learning experience	Dependency hell Need to understand a lot of details Version compatibility Lack of reproducibility
Binary distribution Debian, Ubuntu, Fedora, etc.	Easy to create and extend	Hard to customize Hard to optimize (boot time, size) Hard to rebuild the full system from source Large system Uses native compilation (slow) No well-defined mechanism to generate an image Lots of mandatory dependencies Not available for all architectures
Build systems Buildroot, Yocto, PTXdist, etc.	Nearly full flexibility Built from source: customization and optimization are easy Fully reproducible Uses cross-compilation Have embedded specific packages not necessarily in desktop distros Make more features optional	Not as easy as a binary distribution Build time

®<https://bootlin.com/doc/training/buildroot/buildroot-slides.pdf> - CC BY-SA 4.0

Build systems

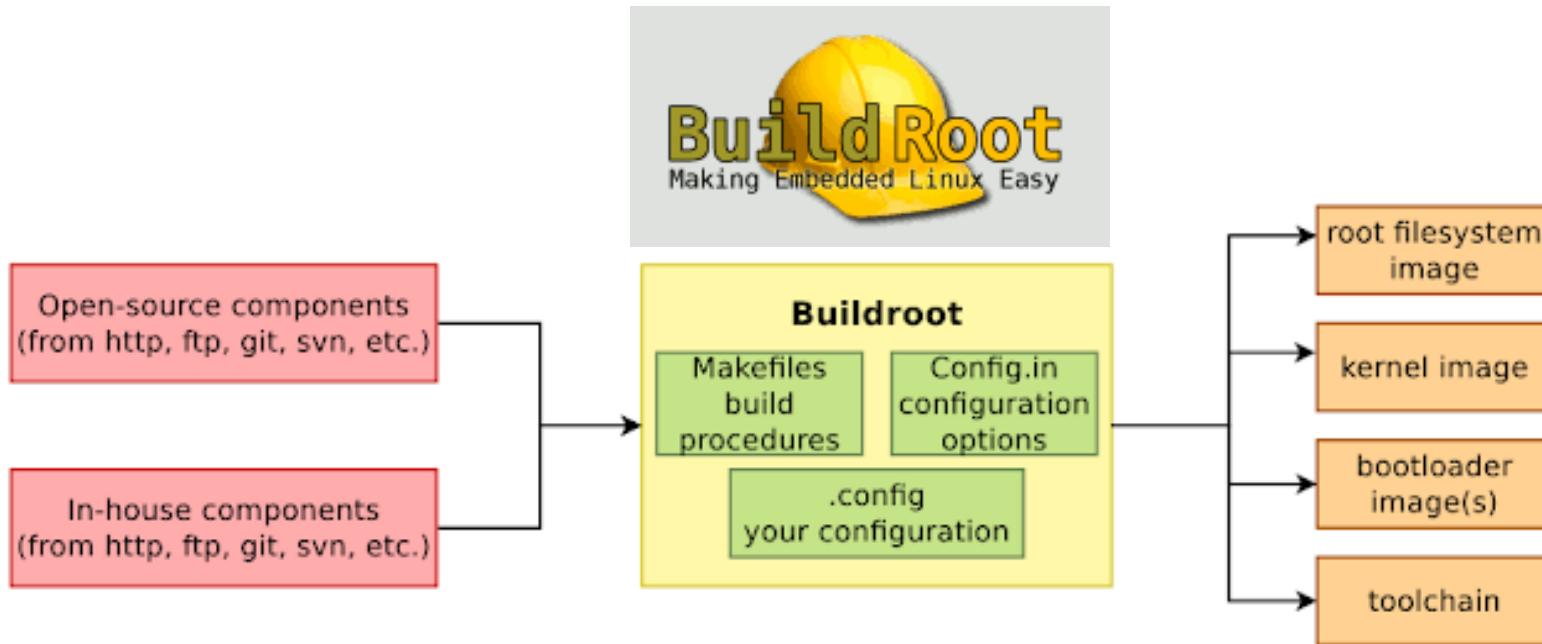
- ✓ Ferramenta para criação de uma distro Linux do zero (**processo manual**): [Linux From Scratch](#)
- ✓ **Build systems:** permite gerar uma distro Linux customizada para um target com vantagem de usar ferramentas, por vezes open-source, parcialmente já configuradas, para construir o sistema operacional, facilitando o trabalho.



® <https://bootlin.com/blog/building-a-linux-system-for-the-stm32mp1-basic-system/>

Buildroot

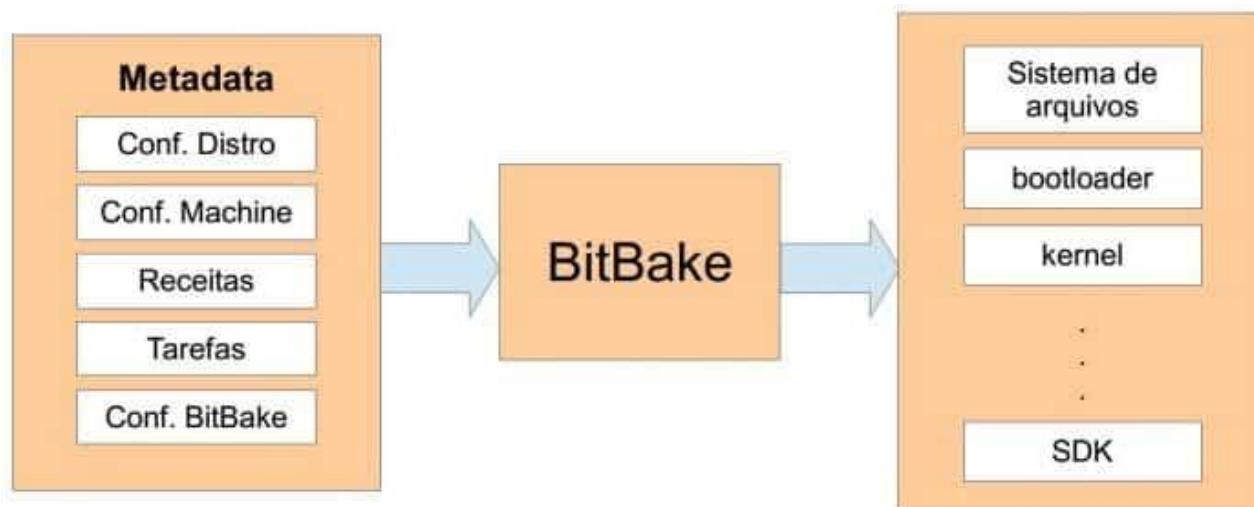
- ✓ Em resumo, é um conjunto de *makefiles* e *patches* que simplifica e automatiza o processo de construção de um ambiente Linux
- ✓ É um sistema mais simples e com maior facilidade de operar.
- ✓ Página: <https://buildroot.org/>
- ✓ [Documentação](#).



® <https://bootlin.com/blog/building-a-linux-system-for-the-stm32mp1-basic-system/>

Yocto Project

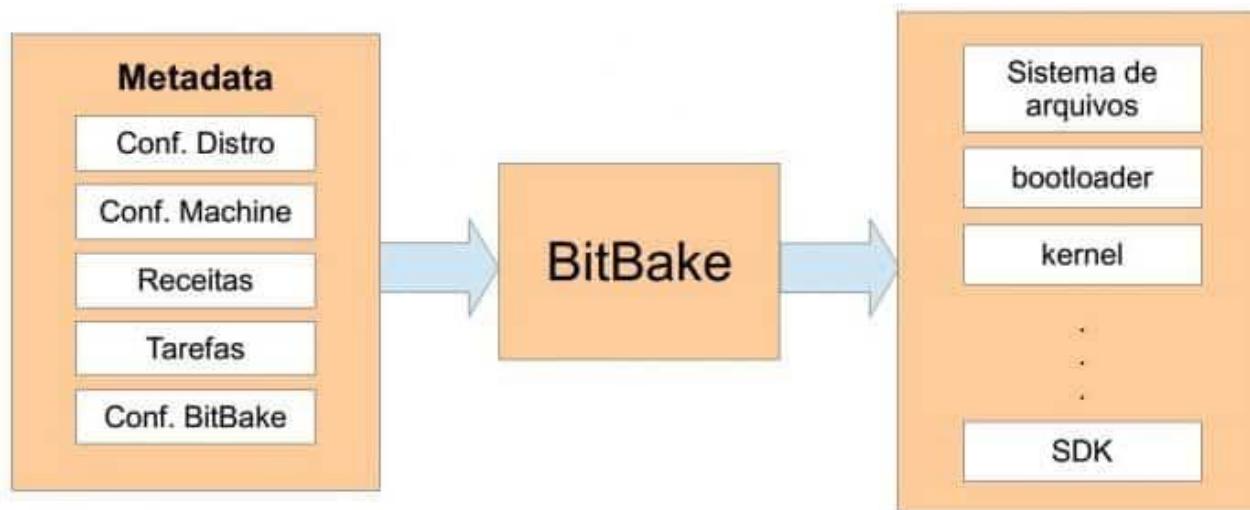
- ✓ Utiliza o build Open Embedded
- ✓ É um projeto open-source colaborativo para criação de distribuições Linux.
- ✓ É mais completo, porém, não tão simples de se trabalhar como Buildroot.
- ✓ Trabalha com um sistema de referência denominado “**Poky**”, que contém uma coleção de várias receitas para criação e customização de novas distros.
- ✓ Documentação e suporte para nova distro (incluindo para Rasp.)



® Yocto Project / Embarcados - <https://embarcados.com.br/yocto-project-definicoes-e-conceitos/> <https://embarcados.com.br/yocto-project-introducao/>

Yocto Project – Poky, BitBake e metadados

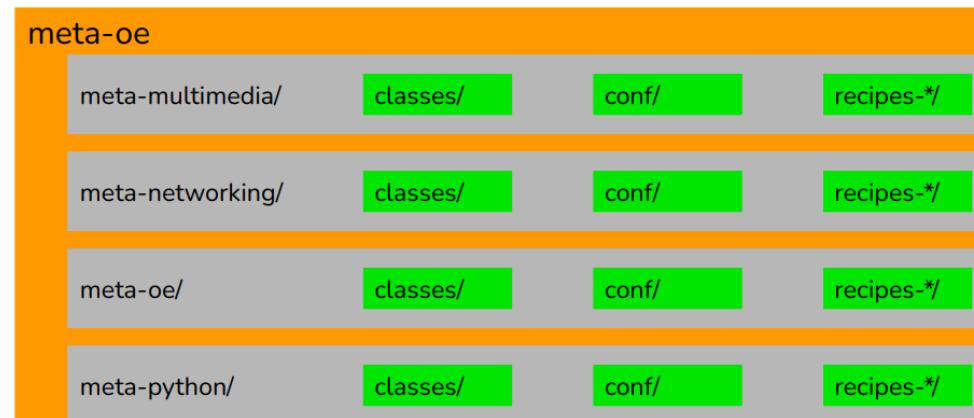
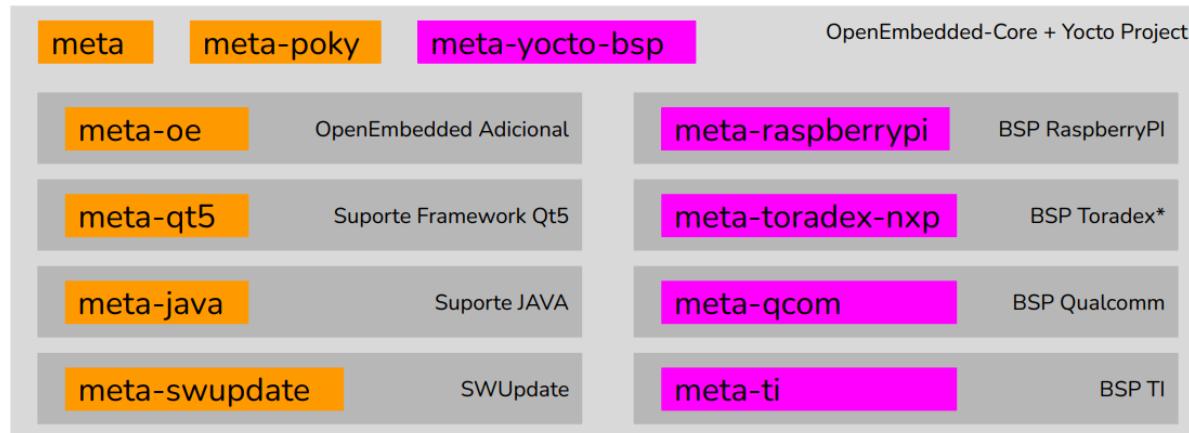
- ✓ Opera com uma ferramenta para execução de tarefas e gerenciamento de metadados escrita em python chamada **BitBake**;
- ✓ E com **metadados** (coleção estruturada de “receitas” e arquivos que “dizem” ao Bitbake o que e como construir o sistema operacional).



© Yocto Project / Embarcados - <https://embarcados.com.br/yocto-project-definicoes-e-conceitos/> <https://embarcados.com.br/yocto-project-introducao/>

Yocto Project - Layers

- ✓ Layers: agregam recursos, funcionalidades, suporte ao sistema embarcado e ferramentas para construção da imagem



Yocto Project - gerando uma imagem para Rasp.

- ✓ Exemplo de construção de uma imagem (distro minimalista) para Rasp.:
- ✓ `~/buildsystem-yocto/poky/build-raspberrypi3 $ bitbake core-image-sato`

```
Build Configuration:  
BB_VERSION      = "1.46.0"  
BUILD_SYS       = "x86_64-linux"  
NATIVELSBSTRING = "universal"  
TARGET_SYS      = "aarch64-poky-linux"  
MACHINE         = "raspberrypi3-64"  
DISTRO          = "poky"  
DISTRO_VERSION  = "3.1.2"  
TUNE_FEATURES   = "aarch64 cortexa53 crc"  
TARGET_FPU       = ""  
  
meta  
meta-poky  
meta-yocto-bsp  = "dunfell:569b1f5d67c57de957e243997c53ec2f81dc8dfe"  
meta-networking  
meta-multimedia  
meta-oe  
meta-python      = "dunfell:cc6fc6b1641ab23089c1e3bba11e0c6394f0867c"  
meta-raspberrypi = "dunfell:eb2f6d460cac81cdf46da1deb5c6227d2f34e9af"
```

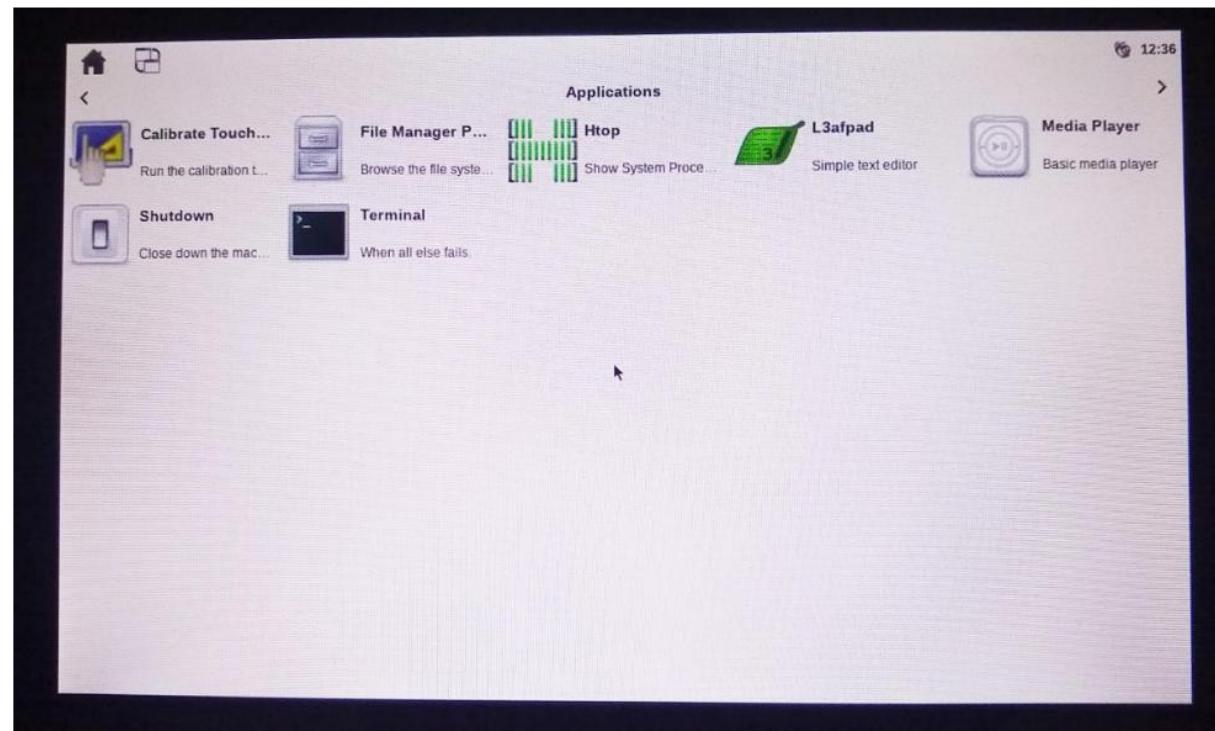
Yocto Project – gerando uma imagem para Rasp.

- ✓ Exemplo de construção de uma imagem (distro minimalista) para Rasp.:
- ✓ O processo de construção da imagem pode levar algumas horas.
- ✓ Ao final o arquivo “**core-image-sato-raspberrypi3-64.rpi-sdimg**” deve ser gravado no cartão SD e será possível inicializar na Rasp.



Flash. Flawless.

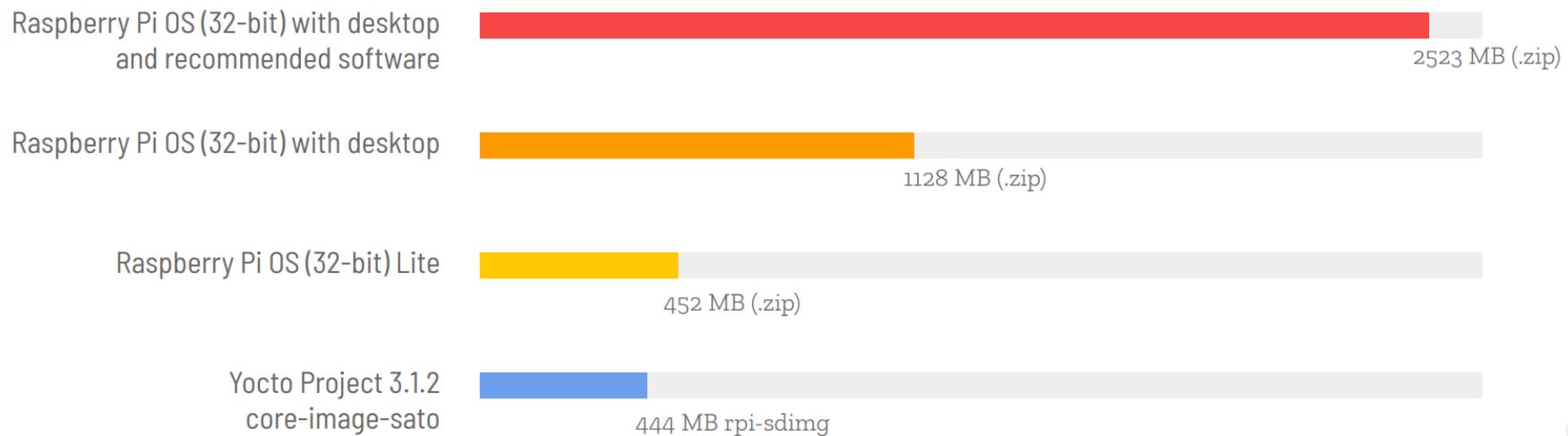
Flash OS images to SD cards & USB drives, safely and easily.



B2Open Systems CC BY-SA 4.0- <https://publicacoes.b2open.com/#listDL>

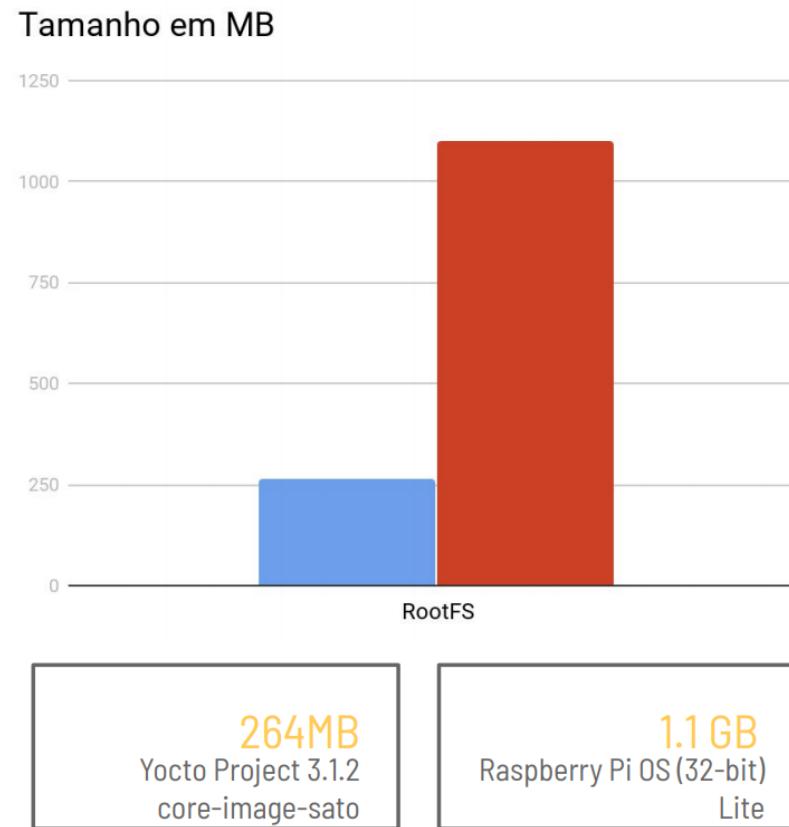
Comparando tamanhos: Raspbian vs. Yocto

- ✓ Imagens (compactada) e `rootfs` (descompactado) dos O.S. Raspbian e distribuição mínima gerada com Yocto



Comparando tamanhos: Raspbian vs. Yocto

- ✓ Imagens (compactada) e `rootfs` (descompactado) dos O.S. Raspbian e distribuição mínima gerada com Yocto



B2Open Systems CC BY-SA 4.0- <https://publicacoes.b2open.com/#listDL>

Armbian

- ✓ Armbian (“ARM Debian” – gera imagem Linux customizada para sistemas embarcados, bastante usada para Set-Top-Box) - <https://www.armbian.com>



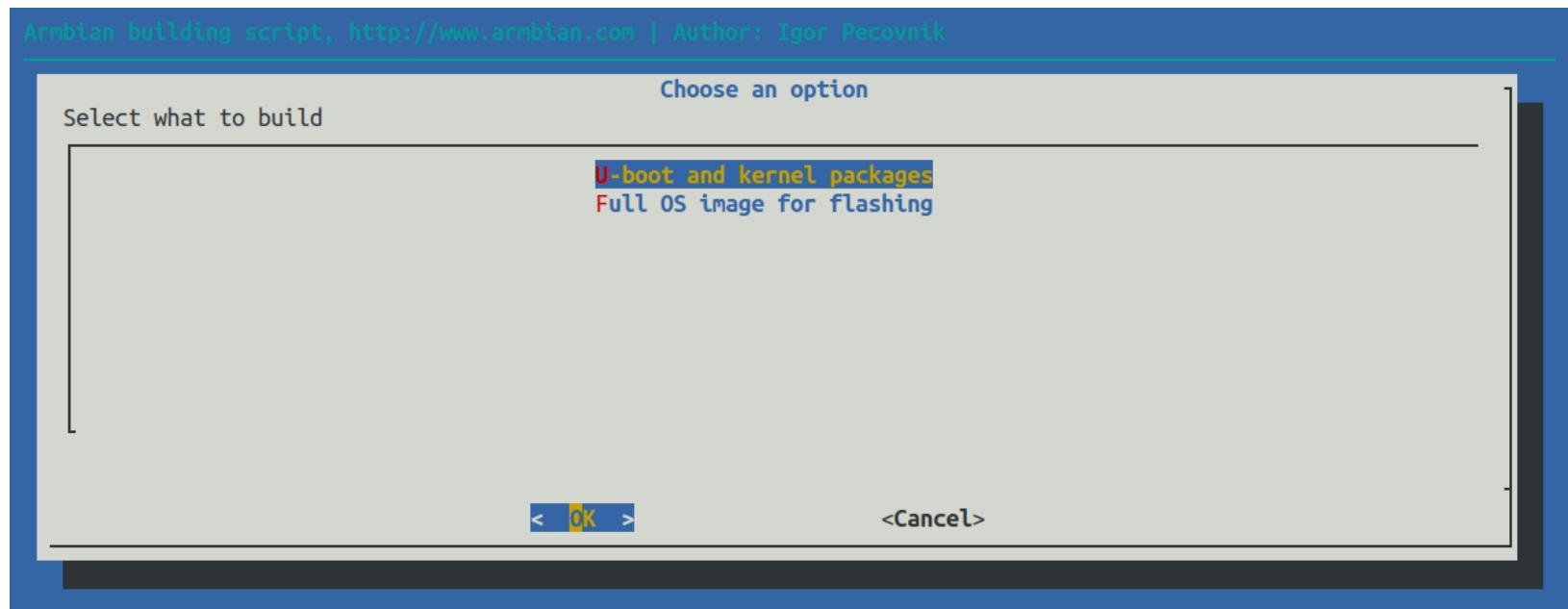
Distribuição Linux para TV-Box

- ✓ A ideia é retirar o sistema operacional instalado (descaracterização) e instalar um **Armbian** customizado para o hardware da TV-Box



Distribuição Linux para TV-Box

- ✓ Armbian – documentação no GitHub: <https://github.com/armbian>
- ✓ Criação de imagem customizada: <https://github.com/armbian/build>
- ✓ <https://wikicheetah.readthedocs.io/en/latest/api.html#tvbox>
- ✓ <https://colab.research.google.com/drive/12H454cBhw3xDux9VxbK3AEj3Q7kZd-gj?usp=sharing#scrollTo=rv1p0uRff4W9>



Tutorial para geração de uma imagem Linux customizada para TV Box

Modelos de TV-Box já descaracterizados

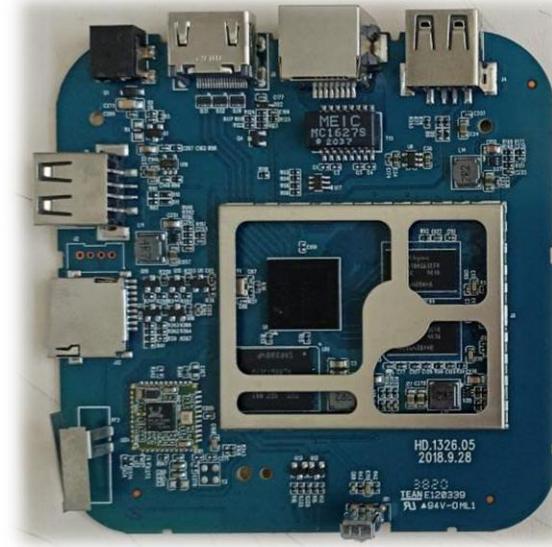
- Modelos: **BTV B11, H6+, TX9, TX2, E9, E10, MXQ, MX9, MX10, H7 ...**
- Por vezes o procedimento de descaracterização é o mesmo para vários modelos de TV-Box, pois compartilham do mesmo hardware (SoC)
- Por exemplo:
 - ✓ SoC Amlogic S905X3 (**modelos B11, H6, H7**)
 - ✓ Tutorial de descaracterização e imagem Linux:
<https://xooh4pms.srv-108-181-92-66.webserverhost.top/installacao-do-ambian-na-tv-box-h6/>
<https://xooh4pms.srv-108-181-92-66.webserverhost.top/installacao-do-armbian-na-tv-box-btv-b11/>

Modelos de TV-Box disponíveis (unidades para testes)

Gerando uma imagem customizada para Set-Top-Box H7 ou BTV11 com Armbian

- ✓ Seguir o tutorial disponibilizado e realizar o boot

➤ Modelos: **BTV B11, H7 (Amlogic S905X3)**



Gerando uma imagem customizada para TV Box

- ✓ Seguir o tutorial disponibilizado e realizar o boot
- ✓ Primeira parte: criação da imagem conforme <https://github.com/armbian/build>
- ✓ Antes de executar o passo acima, é necessário instalar a seguinte dependência: Docker

O Docker é uma aplicação que simplifica o processo de gerenciamento de processos de aplicativos em contêineres no Linux. Os contêineres permitem a execução de aplicativos em processos isolados em termos de recursos (são semelhantes a máquinas virtuais, porém, mais portáteis, consomem menos recursos e dependem mais do sistema operacional físico).

Gerando uma imagem customizada para TV Box

- ✓ Instalando e configurando o Docker
- ✓ Os passos abaixo atualizam pacotes, instalam pré-requisitos, adiciona uma chave GPG e repositório do Docker, instala o Docker e adiciona o usuário comum ao grupo

```
sudo apt update
sudo apt install apt-transport-https ca-certificates curl gnupg2 software-properties-common
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs) stable"
```

```
apt-cache policy docker-ce
sudo apt install docker-ce
sudo systemctl status docker
```

```
sudo usermod -aG docker ${USER}
su - ${USER}
```

```
id -nG
```

```
sudo usermod -aG docker sel
```

Gerando uma imagem customizada para TV Box

- ✓ Seguir o tutorial disponibilizado e realizar o boot
- ✓ Primeira parte: criação da imagem conforme <https://github.com/armbian/build>
- ✓ Utilizar um PC com Linux (Debian ou preferencialmente Ubuntu) e seguir os passos abaixo

```
sudo apt update
```

```
sudo apt -y install git
```

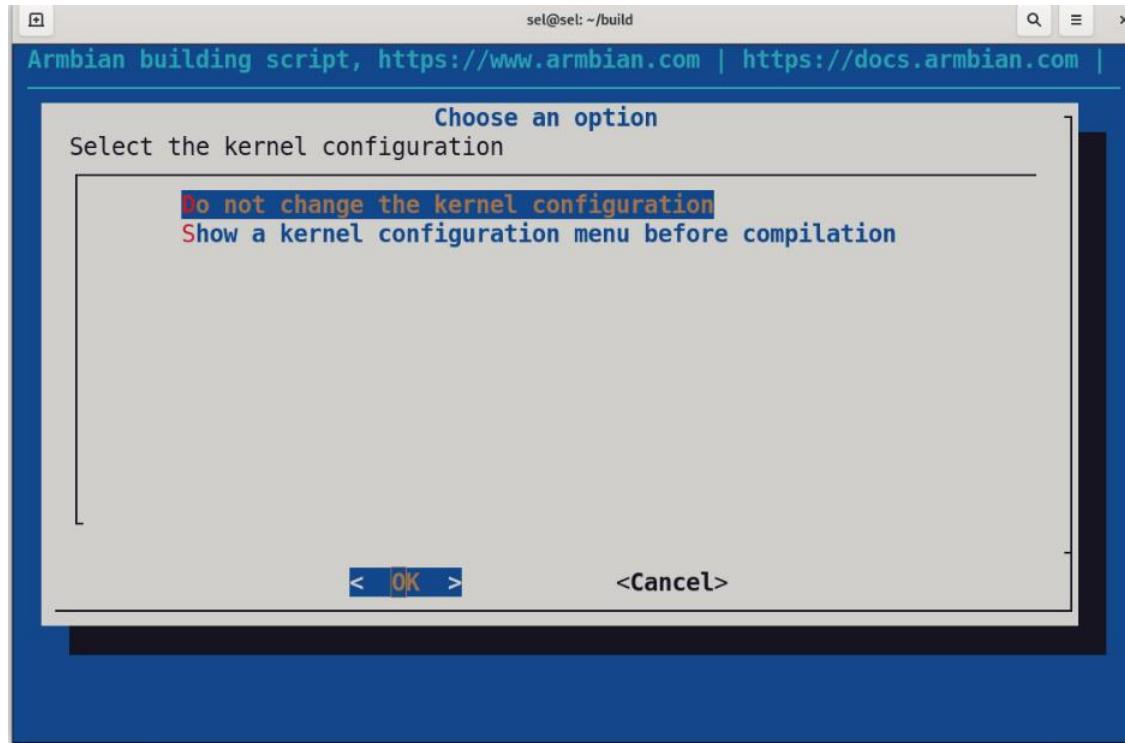
```
git clone --depth=1 --branch=main https://github.com/armbian/build
```

```
cd build
```

```
./compile.sh
```

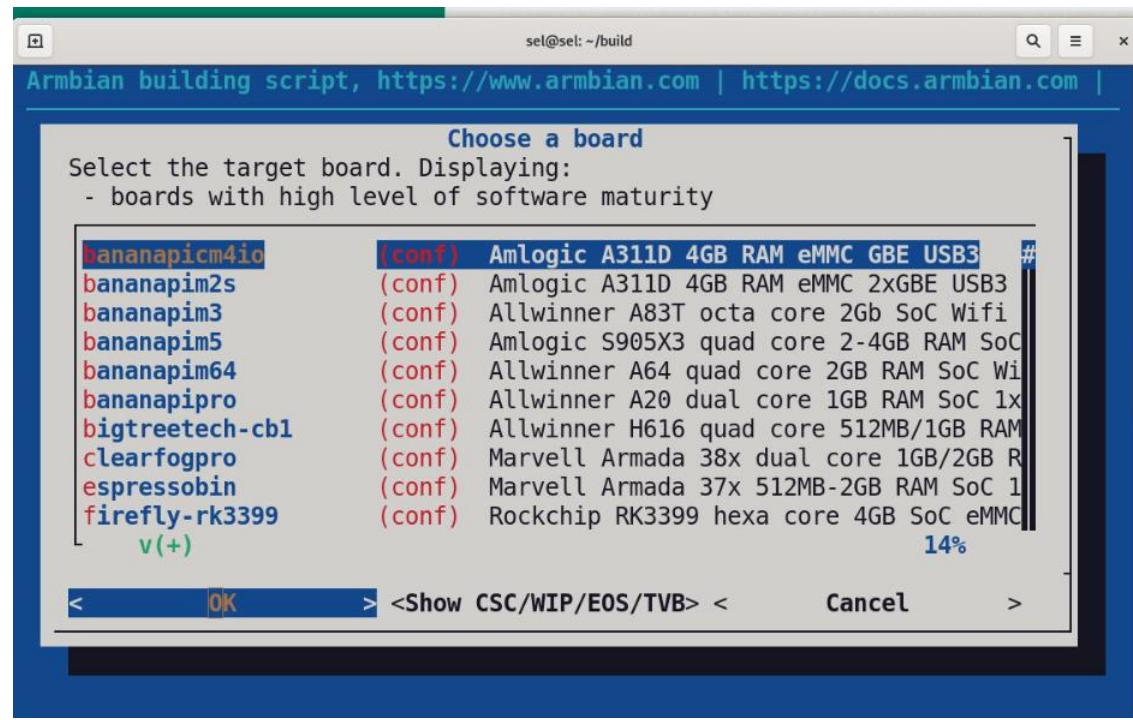
Gerando uma imagem customizada para TV Box

- ✓ Após executar: `./compile.h`, o primeiro passo, na tela a seguir, é escolher a configuração do kernel. Caso desejar verificar/modificar, escolher a segunda opção. Navegar nesta interface usando o teclado do computador (setas e enter)



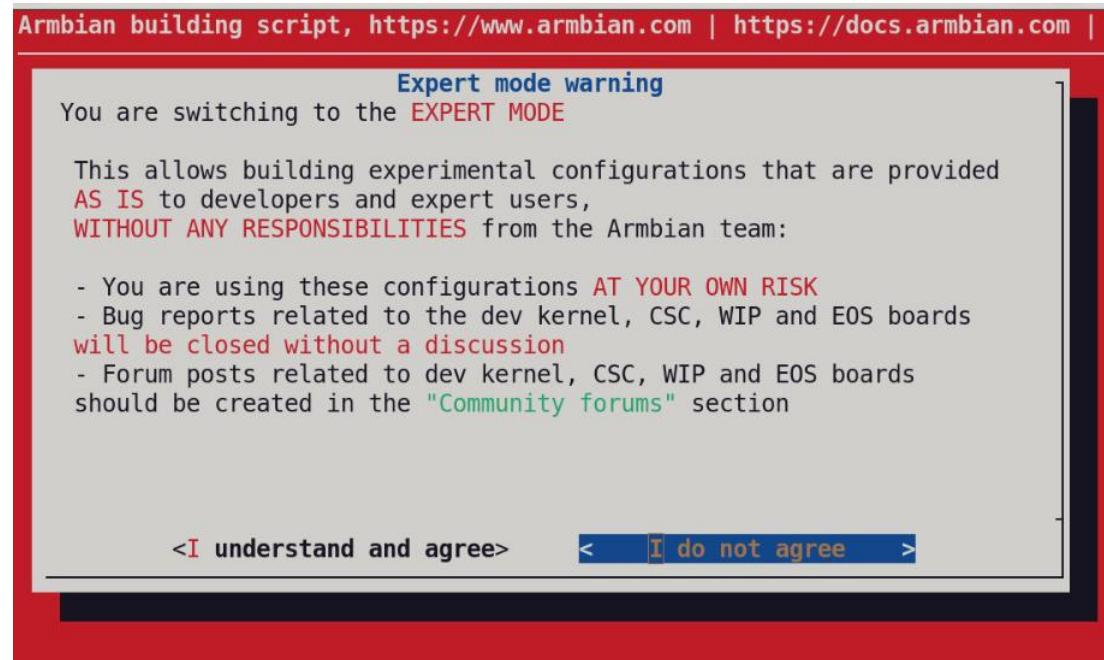
Gerando uma imagem customizada para TV Box

- ✓ Na próxima janela, poderá escolher uma imagem a ser compilada para um target da lista, segundo o modelo de SBCs e do SoC descrito ao lado.
- ✓ Entretanto, para a TV Box, é necessário acessar a opção <Show CSC/WIP/EOS/TVB> ao lado de OK



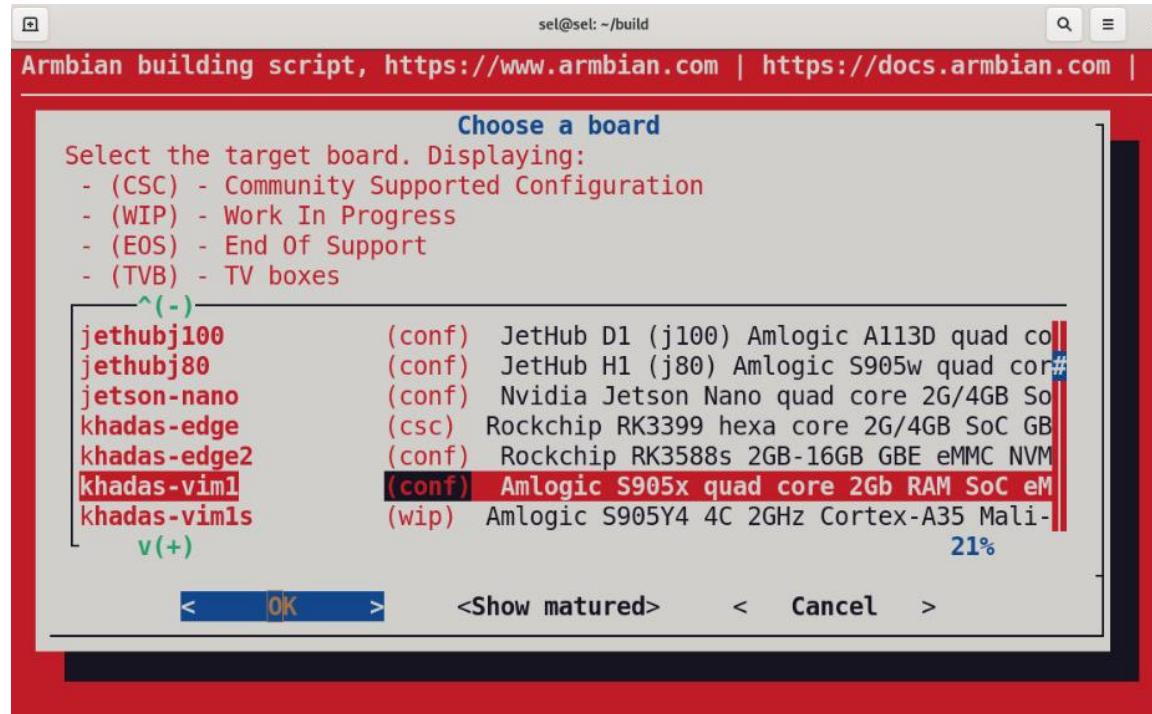
Gerando uma imagem customizada para TV Box

- ✓ Na próxima janela, confirme que irá compilar uma imagem experimental por sua conta e risco, na opção < I understand and agree>



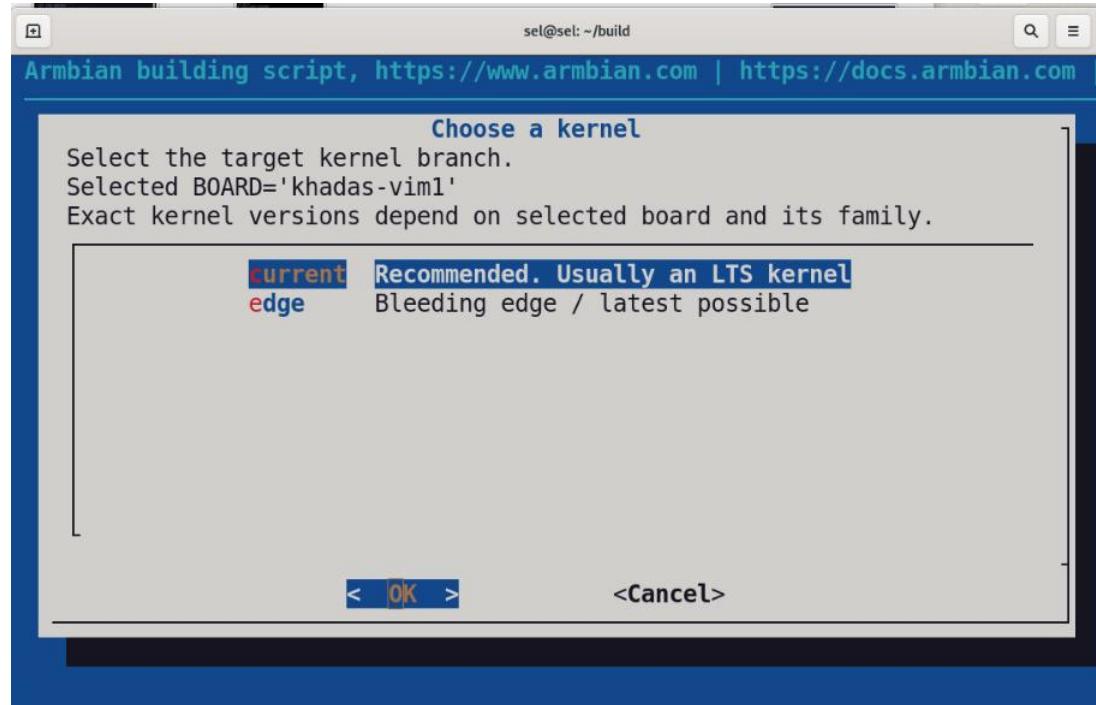
Gerando uma imagem customizada para TV Box

- ✓ Na próxima janela, procure pelo SoC (processador) do modelo de TV Box para qual deseja gerar a imagem (consulte a especificação na placa) e depois “OK”.



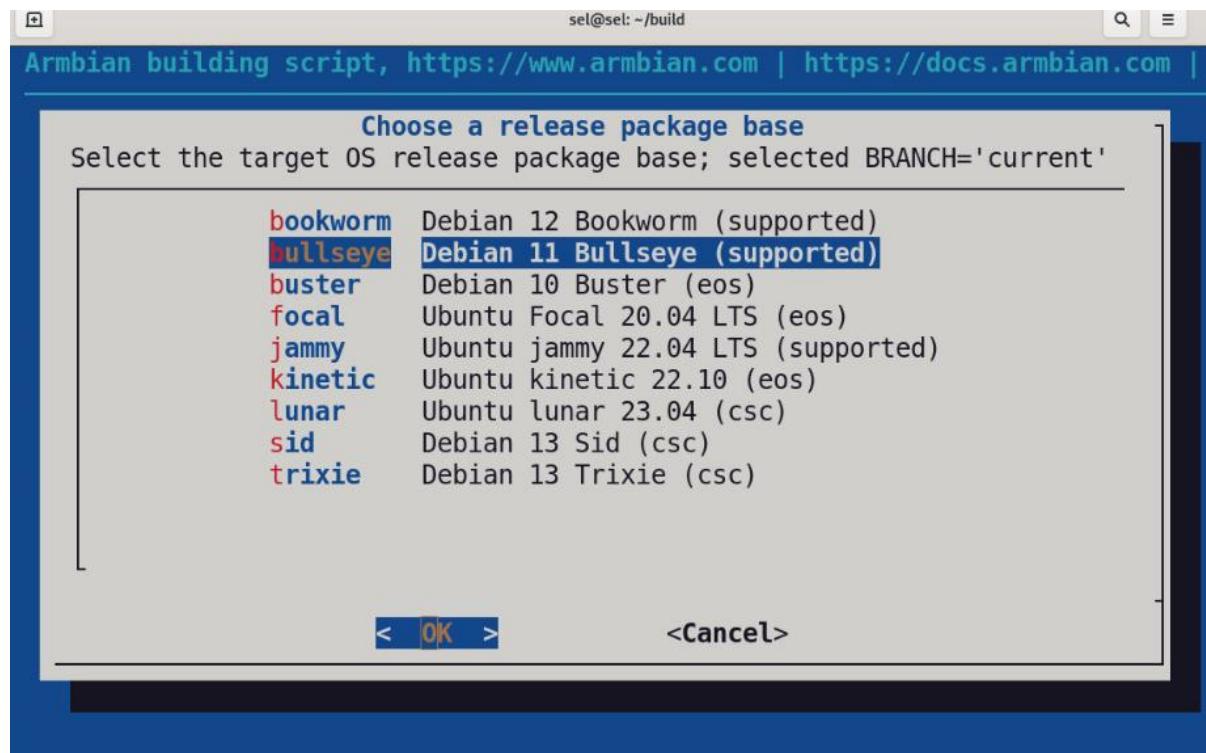
Gerando uma imagem customizada para TV Box

- ✓ Na próxima janela, selecione a versão do kernel (pode ser a recomendada, pois é mais consolidada) e depois OK.



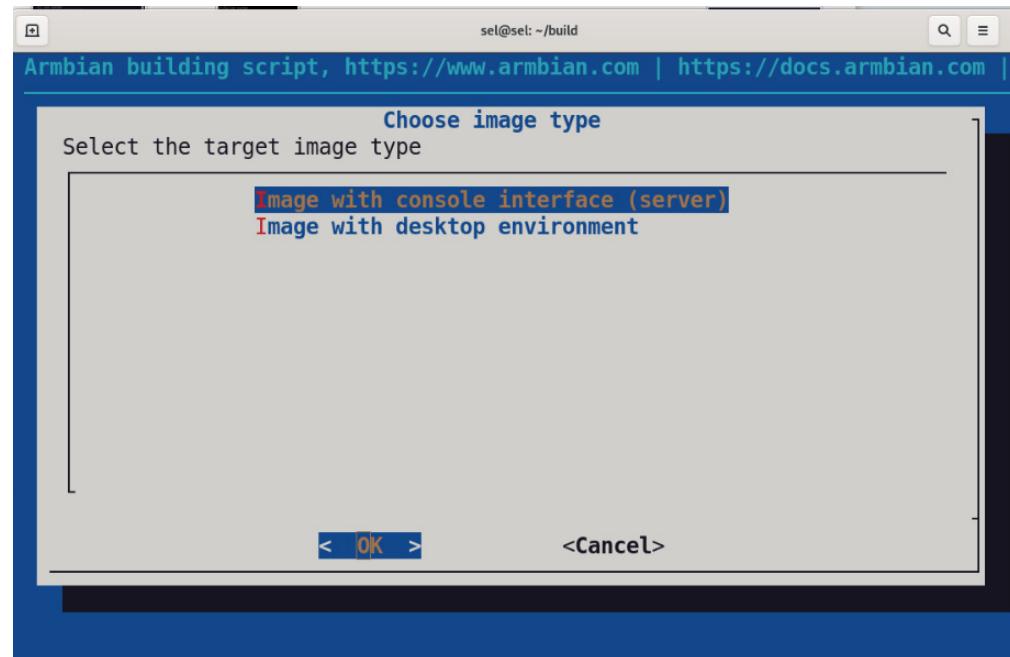
Gerando uma imagem customizada para TV Box

- ✓ Na próxima janela, selecione o pacote base para instalação da distribuição que será a base da imagem a ser gerada. A “bullseye” está sendo a mais popular para Debian



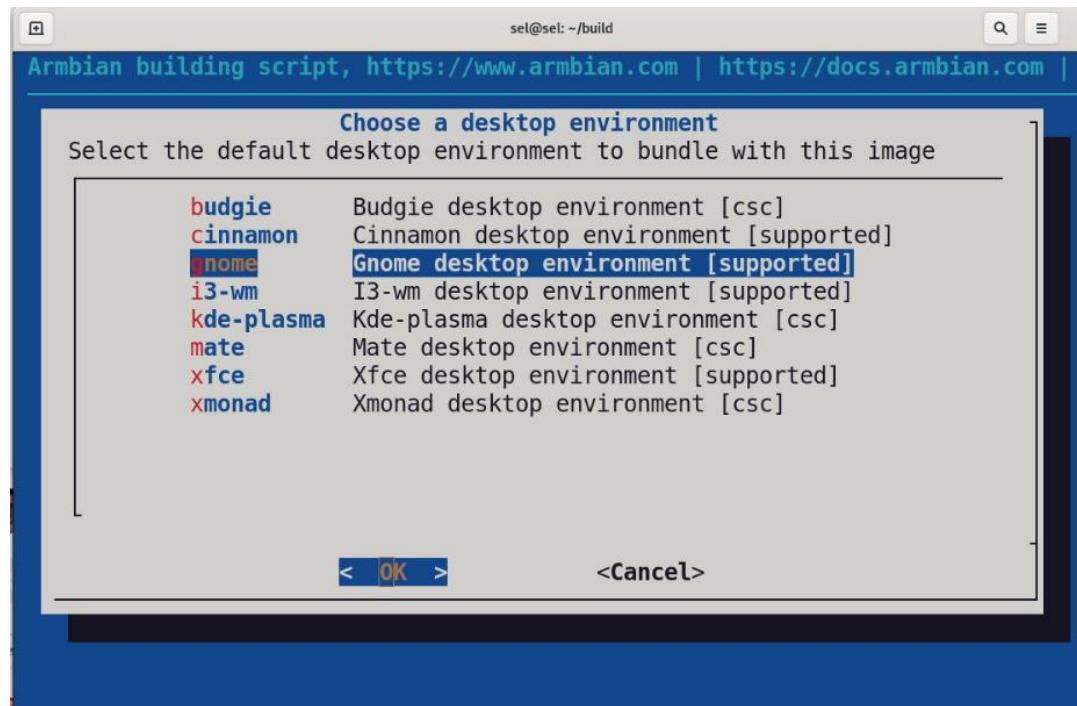
Gerando uma imagem customizada para TV Box

- ✓ Na próxima janela, escolha o tipo de imagem: com interface gráfica (se desejar transformar a TV Box em um minicomputador de propósito geral) ou a versão “server”, sem interface gráfica, caso for usa-la para projetos específicos, como servidor.



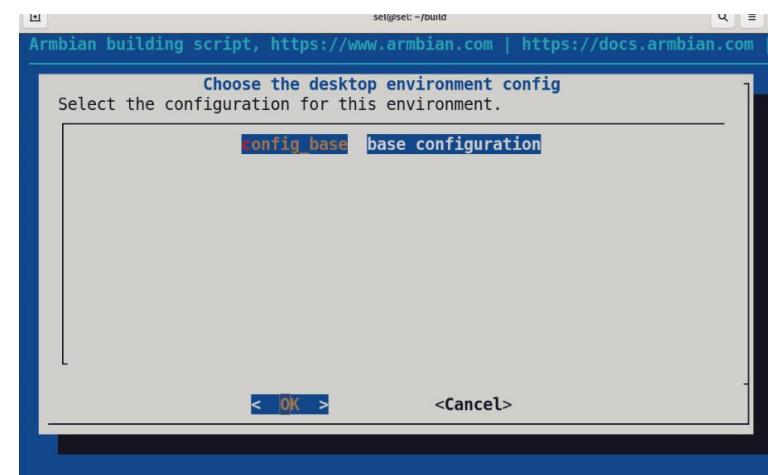
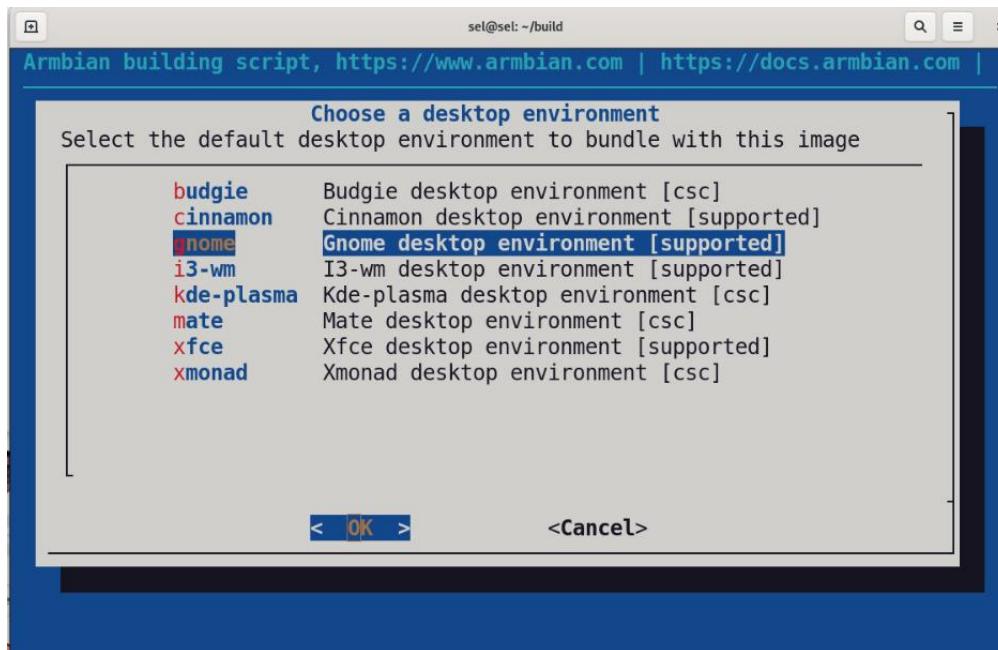
Gerando uma imagem customizada para TV Box

- ✓ Na próxima janela, escolha qual interface gráfica deseja usar, como Gnome, KDE etc.



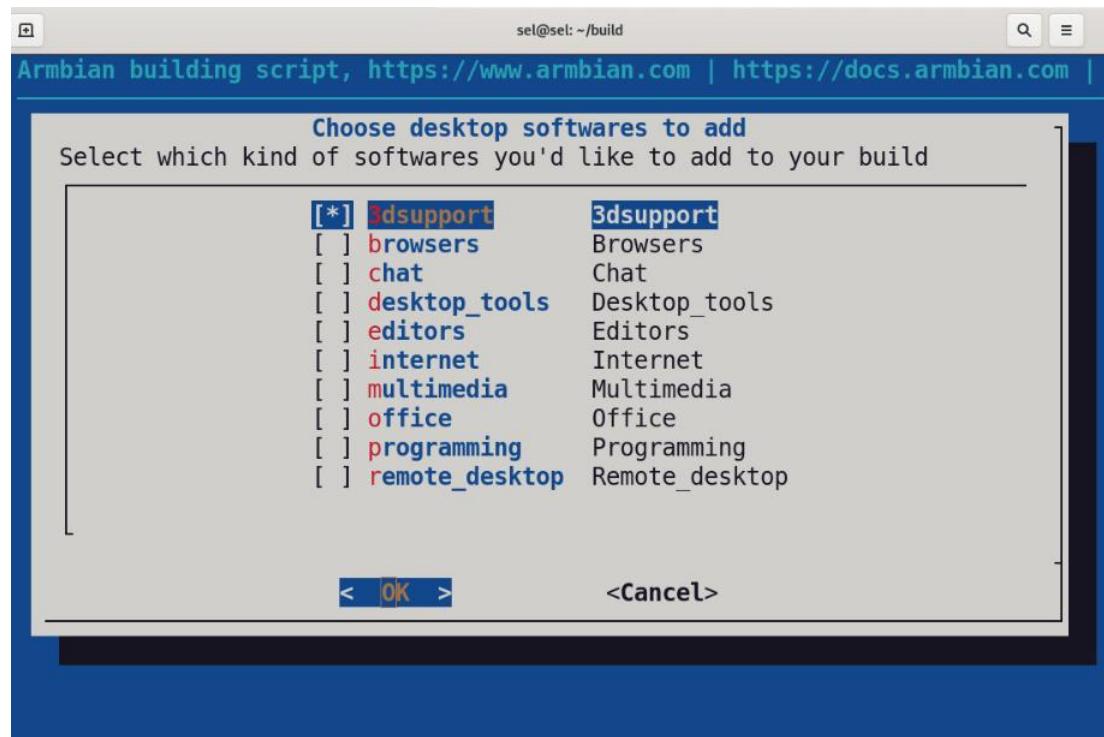
Gerando uma imagem customizada para TV Box

- ✓ Na próxima janela, escolha qual interface gráfica deseja usar, como Gnome, KDE etc.
- ✓ Em seguida, selecione a configuração base para essa interface.



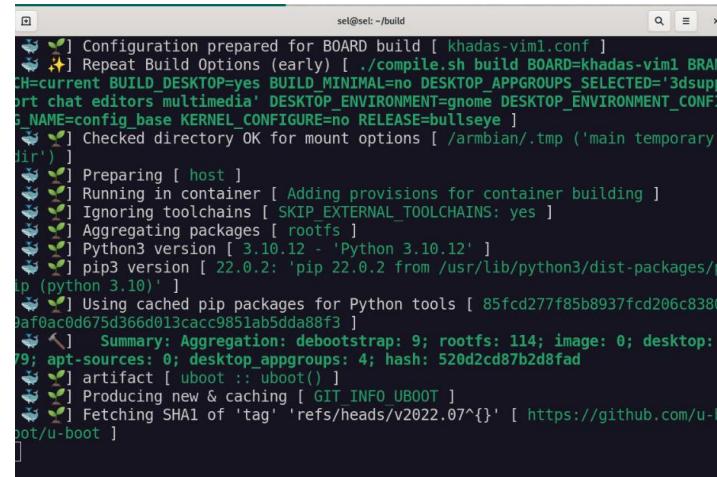
Gerando uma imagem customizada para TV Box

- ✓ Em seguida, selecione quais camadas de software deseja adicionar na imagem, a depender da aplicação/destino da TV Box (navegar com o teclado, e clicar com mouse para selecionar).



Gerando uma imagem customizada para TV Box

- ✓ Após o passo do slide anterior, a imagem será compilada com base nas configurações escolhidas. Esta etapa, no entanto, leva várias horas para completar (geralmente 4, 5, 6 horas). Ao final da compilação, buscar a imagem em: `/home/sel/build/outputs/image`.
- ✓ Não é necessário aguardar se estiver usando o modelo BTV11 ou H7, pois já temos uma imagem gerada. Nesse caso, pode cancelar essa operação e seguir os próximos tutoriais. No entanto, é importante ter executado esses passos para conhecimento da ferramenta.



```
sel@sel: ~/build
[ sel ] Configuration prepared for BOARD build [ khadas-vim1.conf ]
[ sel ] Repeat Build Options (early) [ ./compile.sh build BOARD=khadas-vim1 BRAN
CH=current BUILD_DESKTOP=yes BUILD_MINIMAL=no DESKTOP_APPGROUPS_SELECTED='3dsupp
ort chat editors multimedia' DESKTOP_ENVIRONMENT=gnome DESKTOP_ENVIRONMENT_CONFIG_N
AME=config_base KERNEL_CONFIGURE=no RELEASE=bullseye ]
[ sel ] Checked directory OK for mount options [ /armbian/.tmp ('main temporary
file') ]
[ sel ] Preparing [ host ]
[ sel ] Running in container [ Adding provisions for container building ]
[ sel ] Ignoring toolchains [ SKIP_EXTERNAL_TOOLCHAINS: yes ]
[ sel ] Aggregating packages [ rootfs ]
[ sel ] Python3 version [ 3.10.12 - 'Python 3.10.12' ]
[ sel ] pip3 version [ 22.0.2 from /usr/lib/python3/dist-packages/pip (python 3.10) ]
[ sel ] Using cached pip packages for Python tools [ 85fcfd277f85b8937fc206c8380
9af0ac0d675d366d013cacc9851ab5dd88f3 ]
[ sel ] Summary: Aggregation: debootstrap: 9; rootfs: 114; image: 0; desktop:
79; apt-sources: 0; desktop_appgroups: 4; hash: 520d2cd87b2d8fad
[ sel ] artifact [ uboot :: uboot() ]
[ sel ] Producing new & caching [ GIT_INFO_UBOOT ]
[ sel ] Fetching SHA1 of 'tag' 'refs/heads/v2022.07^{}' [ https://github.com/u-boot/u-boot ]
```

Boot a partir de uma imagem gerada

- Caso estiver com modelo H7, seguir o tutorial abaixo, fazendo o download da imagem e instalando na TV Box (usar um adaptador USB-microSD e dar boot pelo Pen Drive)

<https://xooh4pms.srv-108-181-92-66.webserverhost.top/installacao-do-ambian-na-tv-box-h6/>

- Ou seguir o tutorial deste link caso estiver com modelo BTV11

<https://xooh4pms.srv-108-181-92-66.webserverhost.top/installacao-do-armbian-na-tv-box-btv-b11/>

Ações que podem ser necessárias

- Primeiramente, pode ser necessário ligar o aparelho (Android), conectando-o em um monitor e navegando com controle remoto para conseguir ter acesso à loja de aplicativos e instalar o AIDA64. Este software irá mostrar quais são as especificações técnicas do hardware da TV Box, incluindo a versão do processador.
- Da mesma forma, pode ser necessário conectar ao Wi-Fi para depois ser possível instalar pacotes na imagem Linux gerada. Outra solução é solicitar o cadastro do endereço MAC do aparelho para usar a internet cabeada.

Modelos de TV-Box já descaracterizados

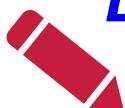
➤ Outro exemplo:

- ✓ SoC RK3229 (comum modelos **TX2, MXQ, MX9**)
- ✓ Imagem customizada:

<https://drive.google.com/file/d/1t94qk6aXMK88nJljy2SslVIKb2jPGHF3/view>

- ✓ Tutorial:

[https://drive.google.com/file/d/1U_BD89vh0A5dCHJJ39ZhpwR88rM_rfgx/vie
w?usp=sharing](https://drive.google.com/file/d/1U_BD89vh0A5dCHJJ39ZhpwR88rM_rfgx/view?usp=sharing)



Documentação do projeto, resultados e tutoriais

- TV-Box já descaracterizadas e projetos desenvolvidos pela comunidade e instituições
https://drive.google.com/drive/folders/1I87ltaU_g0cA1aZQSU-Xps61Qm0HB6x6
- Lista completa de modelos, configurações firmwares
<https://telegra.ph/OTT-BOX-BRASIL-FIRMWARES-02-14>
<https://telegra.ph/OTT-BOX-BRASIL-FIRMWARES-02-14>
- Tutoriais
 - <https://wikicheetah.readthedocs.io/en/latest/api.html#tvbox>
 - <https://colab.research.google.com/drive/12H454cBhw3xDux9VxbK3AEj3Q7kZd-gj?usp=sharing#scrollTo=rv1p0uRff4W9>
 - <https://www.youtube.com/c/MarcosTvBoxAndroid>
 - <https://www.youtube.com/watch?v=8Ch7nUBiQa0&feature=youtu.be>
 - <https://www.youtube.com/watch?v=-jCgbhX5NNU&t=13s>
 - <https://github.com/armbian/build>
- <https://xooh4pms.srv-108-181-92-66.webserverhost.top/installacao-do-ambian-na-tv-box-h6/>
- <https://xooh4pms.srv-108-181-92-66.webserverhost.top/installacao-do-armbian-na-tv-box-btv-b11/>
- <https://drive.google.com/file/d/1t94qk6aXMK88nJljy2SsIVKb2jPGHF3/view>
- https://drive.google.com/file/d/1U_BD89vh0A5dCHJJ39ZhpwR88rM_rfgy/view?usp=sharing

- Obrigado pela atenção!

FIM

Coffee Break

