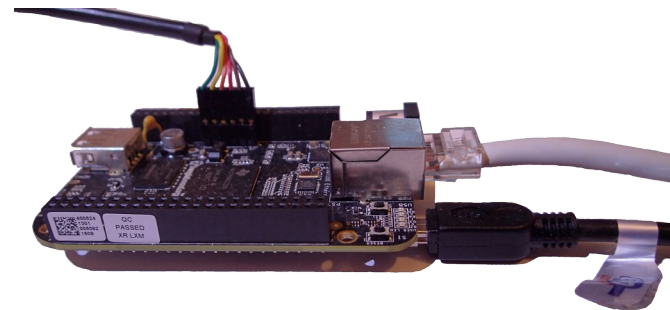




Embedded Linux



Kjell Enblom,
Mindroad.se



2025-03-04

MindRoad
Academy

MindRoad
Embedded

MindRoad
Application

MindRoad
Syd

MindRoad
Sourcing

Who am I ?

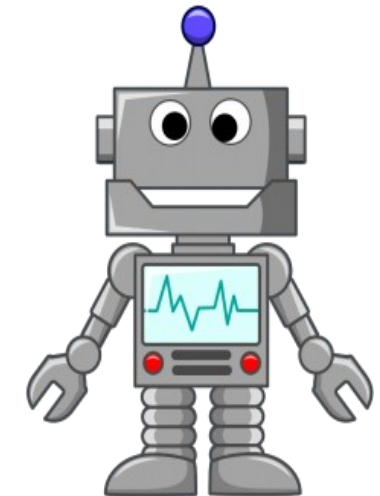
- Kjell Enblom
 - Computer Science, Linköping university
 - Computer consultant
 - I have used Unix since 1985.
 - I have been working with Linux since 1995.
 - Workstations
 - Servers
 - Embedded Linux since 2007

Content

- Where can we find Embedded Linux?
- When to use Linux and not to use Linux.
- The four elements.
- Using standard Linux distribution versus build an embedded Linux distribution.
- How to deploy.
 - Manually
 - Using Buildroot
 - Using Yocto Project

Where can we find Embedded Linux?

- Routers
- Firewalls
- Network encryption.
- Infotainment system
- TV, screens
- Robots
- NAS (network storage)
- Blast furnace
- Set-top-box (TV)
- Measuring equipment
- Washing machines.
- Coffee machines.
- Cameras.



When to use Linux

- We need network support.
- We need to be able to easily run multiple programs.
- The application require more than 1-3 megabytes of memory to run smoothly.
- If you need a lot of boilerplate to get the system up and running then it's easier to use Linux where much of this is already solved.
- We need network connectivity, touch screen, sound, video, embedded web server, encryption.
- There are many small systems with TCP/IP stacks, but often the quality of these is not very good and there is a high risk that they have security holes.
- The TCP/IP stack in Linux is well developed and well tested.

When to use Linux

- Benefits of using embedded Linux
 - its open source nature (free as in price and as in freedom),
 - flexibility and scalability,
 - support for a wide range of hardware architectures,
 - robustness and stability,
 - it has support for process management, support for different file systems, multi-user support, etc
 - With a lot of available libraries in Linux it is easier to develop applications.
 - large community of developers and users.
 - It's easy to restart an application without disrupting other applications or the system.

When to not to use Linux

- If we have limited resources.
 - Resources, like memory, storage and CPU etc. are severely limited.
 - If the hardware lacks MMU.
- Systems with high security requirements.
- Systems with high safety requirements.
- Situations when, for legal reasons, you cannot have GPL or LGPL as a license.
- If we only need a few resources and the price of the hardware is important.

The four elements

- Toolchain
- Bootloader
- Linux kernel
- Root-filesystem including application for system's usage.

The four elements - Toolchain

- A collection of tools for creating binaries or libraries from code. Contains usually:
 - Compiler or cross compiler
 - Assembler
 - Linker
 - System library, libc, e.g. glibc, uClibc-ng, Dietlibc, Bionic etc.

The four elements - Toolchain

- A toolchain based on gnu tools, consists of:
 - GCC (Compiler)
 - Binutils (assembler, linker, profiles, object file tool, etc)
 - Glibc (GNU libc, standard library)



The four elements - Toolchain

- Another available toolchain that is also Open Source is LLVM.



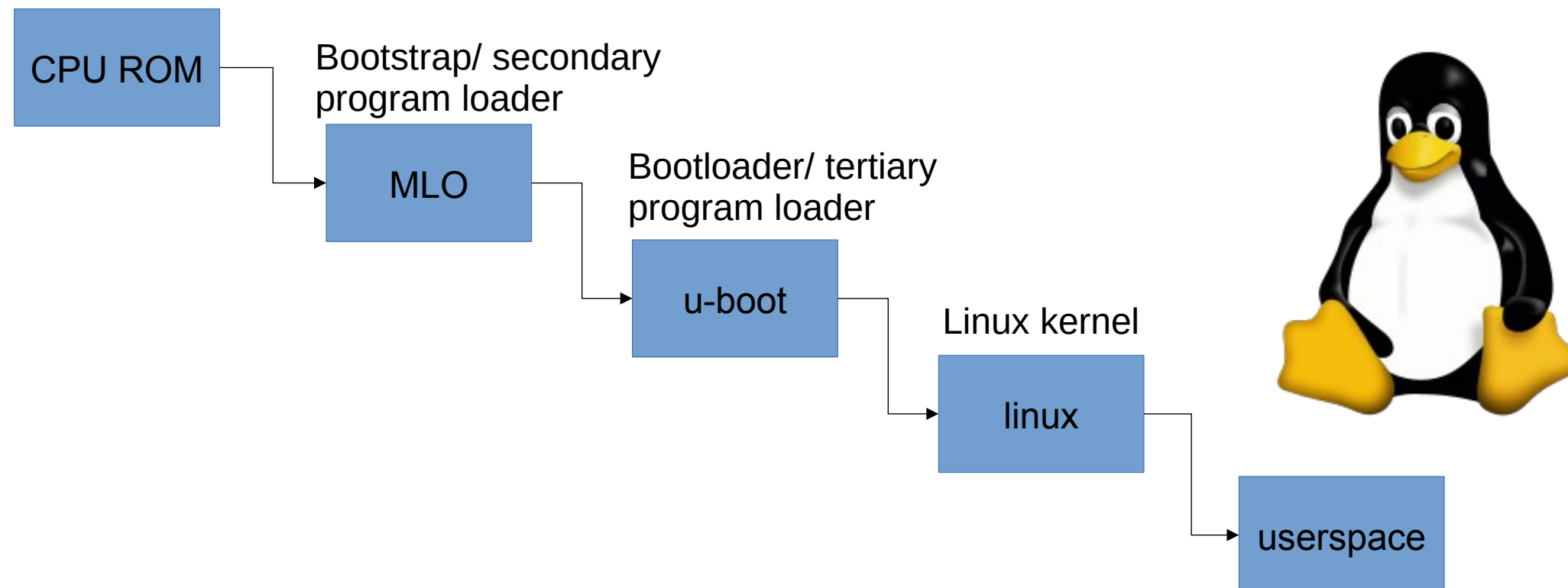
The four elements - Bootloader

- Hardware specific.
- Loads data from non-volatile memory to RAM
- Launches the loaded program, such as a linux kernel



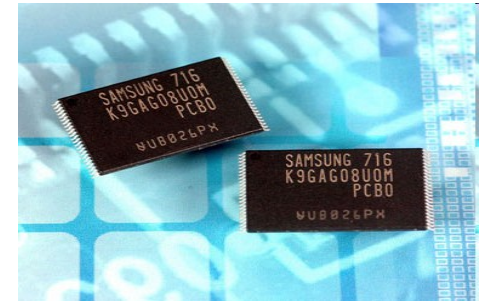
The four elements - Bootloader

- On embedded systems it is common that the bootloader is loaded from a flash memory.
- Really small systems only have a bootstrap.
- Larger systems have a bootstrap and a bootloader.



The four elements - Bootloader

- Some of the most common bootloaders are:
 - LILO (x86 and x86_64 based systems. Used to be standard bootloader)
 - Grub (x86 and x86_64. The standard bootloader in most Linux distributions)
 - U-Boot (System with ARM, AVR32, PPC, Blackfin, x86, Motorola m68k, MIPS, Super-H and more) common in many embedded systems.
 - RedBoot (ARM, x86, m68k, MIPS, PowerPC, Super-H and more)
 - CFE (Broadcom Common Firmware Environment for MIPS, PowerPC and x86)
 - BSSP (proprietary bootloader from STMicroelectronics)



The four elements - Linux Kernel

- An abstraction of hardware.
- A generic platform for applications.
 - Handles hardware, filesystems, network, processes, security (firewall, selinux, apparmor).
- Implementing a standard API against user space.
- In ARM and PowerPC systems we also need a device tree.
 - A device tree is a data structure for describing hardware layout.
 - Device trees are used by U-Boot and by the Linux kernel.
 - With device trees there are no need for hard coded hardware specific values in the device drivers.



The four elements - Root filesystem

- Set of files and application to start the system.
- Contains all necessary applications, libraries, device files, system files for the complete system function.
- Contains the applications that perform the important functions of the embedded system.
- Contains everything that is necessary to add extra file systems if needed.

Using a standard Linux distribution

- You don't need to build the OS.
- Vast number of packages, ready to install.
- You can use the distros package manager to install packages.
- Little setup time.
- When rebuilding a package you need to do than on the board or cross compile on your development station.

Using a standard Linux distribution

- Minuses:
 - Requires hardware support
 - Only works out-of-the-box on commodity hardware such as PC or Raspberry Pi.
 - Standard distros don't have support for most embedded hardware.
 - Native development means compiling on the machine.
 - Ok for PC, but not scalable for Raspberry Pi.
 - Too big
 - e.g. Ubuntu Core is 500 MB, but we might have only 256 MB storage.
 - Software update via package manager is not robust.
 - We need atomic update, e.g. the entire root filesystem image or root filesystem tree.
 - No support or little support for software supply chain including license management.

Build and use an embedded Linux distribution

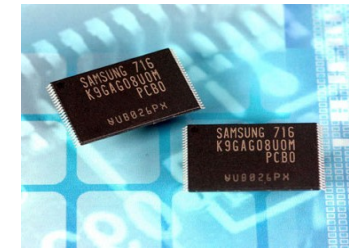
- With a BSP, Board Support Package, you can have support for your hardware.
 - There is plenty of support for many hardware boards.
- It is possible to build a small custom embedded distribution.
- There is good support for cross-compilation development for embedded Linux distributions.
 - You can do the development and cross compilation on your workstation.
- Atomic updates are supported.
 - The system could be in orbit around Jupiter or in a cave deep under the sea.
 - You must not brick the system.

Deploy - manually

- To deploy you need the four elements; cross compiler toolchain, boot loader source code, Linux kernel source code and root filesystem applications and root filesystem files.
- Toolchains come in several different variants; proprietary, ready-made open source, and the ones that you build yourself.
- The easiest way is to use one of the ready-made ones. Example:
 - Linaro, <https://www.linaro.org/downloads>
 - <https://toolchains.bootlin.com/>
- You get the most flexibility and customization with those you build yourself.
 - You can for example use crosstool-ng to build your own toolchain.

Deploy - manually

- Next step is to cross compile the bootloader.
- We need to decide which bootloader to use.
- Then we need the source code for the bootloader and configure it for our target system and for the functions we need and then cross compile it.
- A very common open source bootloader used in many embedded systems is u-boot from DENX Software Engineering.



Deploy - manually

- Example for u-boot for Beaglebone black:
 - Download the source code and extract it and go to the source code directory.
 - export ARCH=arm
 - export CROSS_COMPILE=arm-cortex_a8-linux-gnueabi- # Prefix part of toolchain name
 - make am335x_boneblack_defconfig # Select the beaglebone black configuration.
 - make menuconfig # Do some more configuration
 - make all # Compile U-Boot and U-Boot tools
 - We will get the files MLO and u-boot.img. We need to install them on the target.



Deploy - manually

- Next step is to compile the Linux kernel.
 - Download the Linux kernel and extract the source code.
 - You can get it from <https://www.kernel.org/>
 - Example for Beaglebone black:
 - `make ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabi-multi_v7_defconfig`
Select Beaglebone black configuration
 - `make menuconfig` # Do some more configuration
 - `make` # Compile the Linux Kernel
 - We will get a kernel image and a device tree to install to the target.



Deploy - manually

- Then we need to create the root filesystem.
- For that we need to create some directories and some files and cross compile the applications we need.
- Use the toolchain to do that.
- For all the standard Linux commands we can use busybox.
- Download the source code and configure it and cross compile it.
 - `export ARCH=arm`
 - `export CROSS_COMPILE=arm-cortex_a8-linux-gnueabi-` # Prefix part of toolchain name
 - `make menuconfig` # Do some configuration
 - `make && make install` # Install busybox in a directory tree on your development machine.



Deploy - manually

- Finally, we need to deploy everything to the target system.
- How the system is installed on the target varies between different systems.
 - Install the entire system on an SD card.
 - Copy the files to a bootable media and boot with this on the target to install.
 - Flash the files to a flash memory on the target.



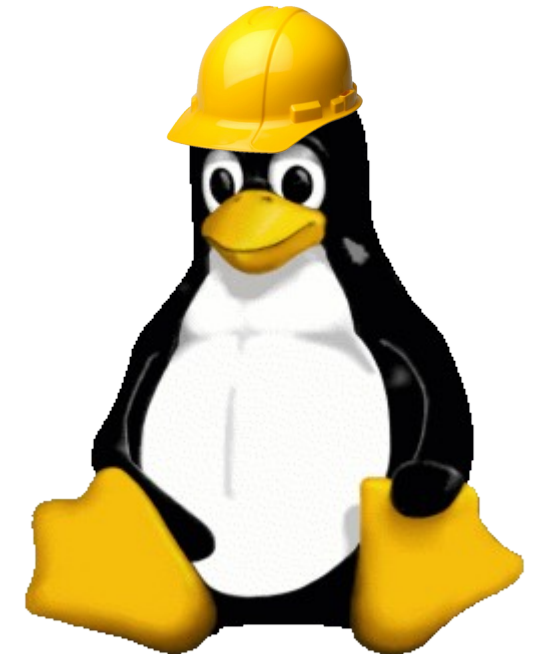
Deploy - manually

- For the Beaglebone black, we can e.g. install the whole system on an SD card and boot from this SD card.
- We create two partitions:
 - A partition that contains the bootloader files, the Linux kernel and a device tree file.
 - One partition containing the root filesystem.



Deploy - Buildroot

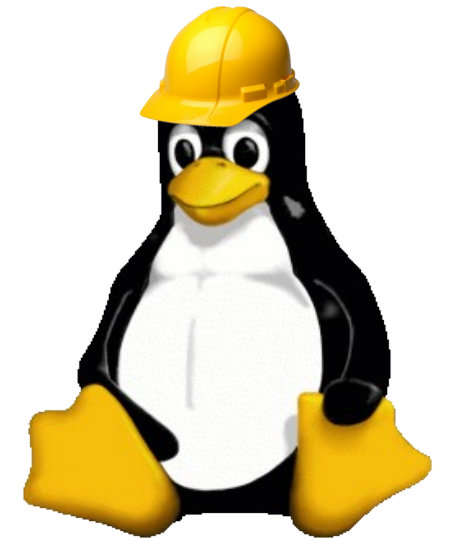
- Putting everything together manually is quite cumbersome and time-consuming.
- It's easier with a ready-made building system.
- Buildroot and Yocto Project are two such systems.
- They are both open source.
- We will first take a look at buildroot.
 - <https://buildroot.org/>



Deploy - Buildroot

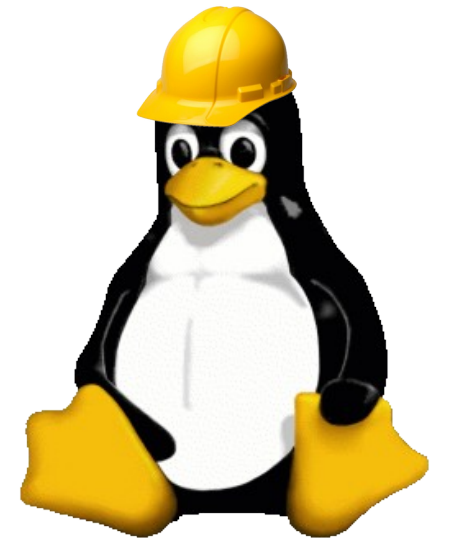


- Buildroot is built around make files and the program make.
- Buildroot can build the toolchain, it builds bootloader, Linux kernel and root filesystem including the programs you develop.
 - It is possible to use an external toolchain with buildroot.
- To build a distribution download the buildroot build system and extract the archive file.
- Run make menuconfig and do all the necessary configuration.
- Then run make to build.
 - Buildroot will download all source code and build it.



Deploy - Buildroot

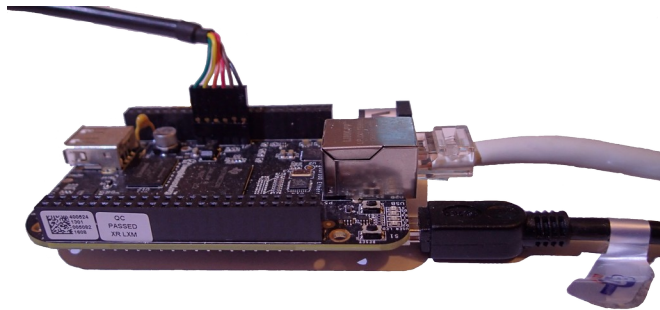
- For own applications we need to create a subdirectory under package and populate it with a configuration file and a makefile.
 - Config.in
 - mypackage.mk
- Then add a line in the file package/Config.in
- For this to work, you need basic knowledge of make and make files.
- With buildroot you can build archive files, complete images for different filesystems to deploy to the target.



Deploy - Buildroot



- Buildroot is relatively easy to learn.
- It is easier to learn Buildroot than to learn Yocto Project.
- If you only have one system, it works well to use buildroot.
- If you have several different systems, buildroot is not as good.
- There is more vendor support for Yocto Project than for Buildroot.
- There is more out-of-the-box BSP support for Yocto Project than for Buildroot.
- With buildroot, however, it is relatively easy to create your own BSP.
- With buildroot (and Yocto) we have support for Software Bill of Materials (SBOM), software supply chain.



Yocto Project is a system to build an embedded Linux distribution.

<https://www.yoctoproject.org/>

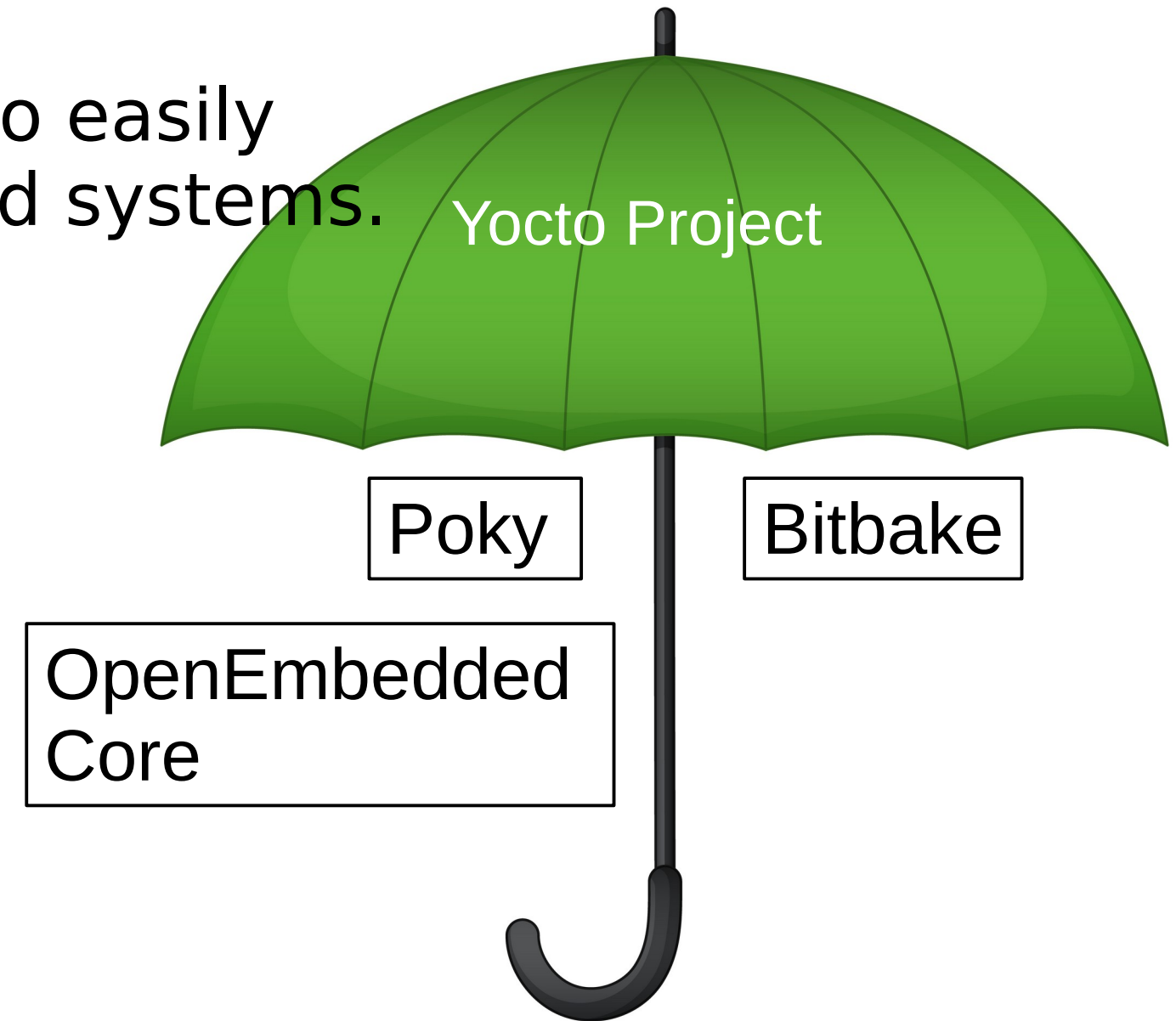
Deploy – Yocto Project

- Yocto project was created in 2010 when several companies wanted a unified system to build Linux distributions for embedded systems.
- The first version of Yocto was released in early 2011.
- The project is run by the Linux Foundation.
- Embedded linux world was sparse and difficult to get into.
- It needed something uniform, an industry standard.
- Huge support from both hardware and software companies and organizations:

Texas Instruments, Intel, Enea, Huawei, OpenEmbedded, etc.

Deploy – Yocto Project

- What is Yocto?
 - Large collection of recipes and tools to easily build Linux distributions for embedded systems.
 - Configurable to meet your needs.
- Yocto is an umbrella for some projects
 - Poky
 - OpenEmbedded Core
 - Bitbake

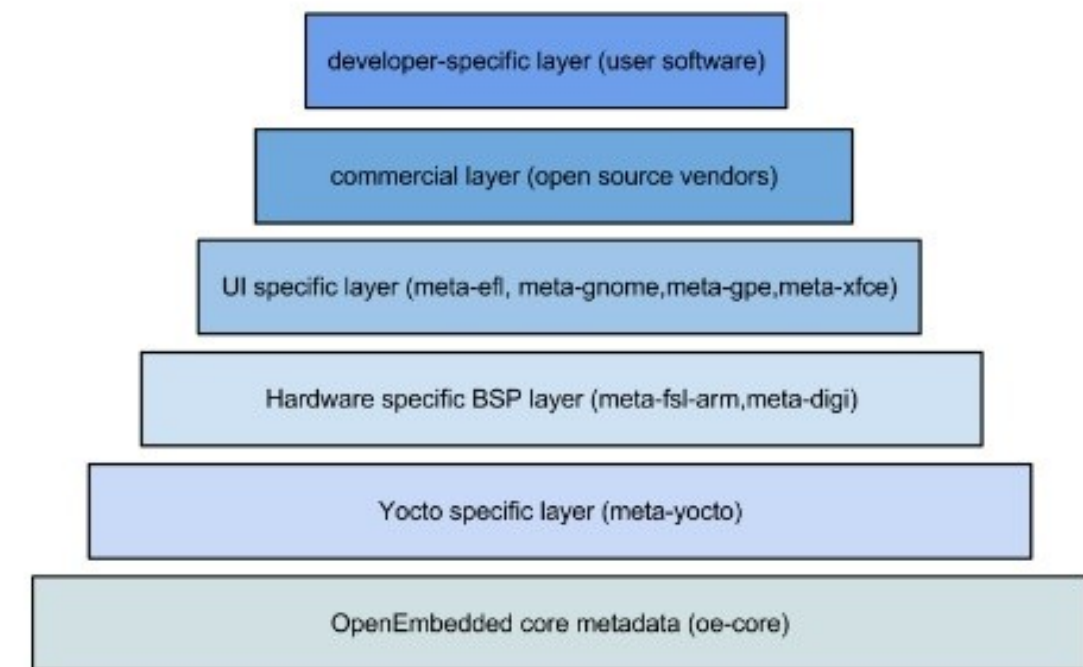


Deploy – Yocto Project

- Yocto is made of layers.
- Recipes are collected in the meta-layers.
- The recipes in a meta-layer contains instructions for how to build, e.g.:
 - An application
 - A BSP
 - An image

Yocto Layers Overview

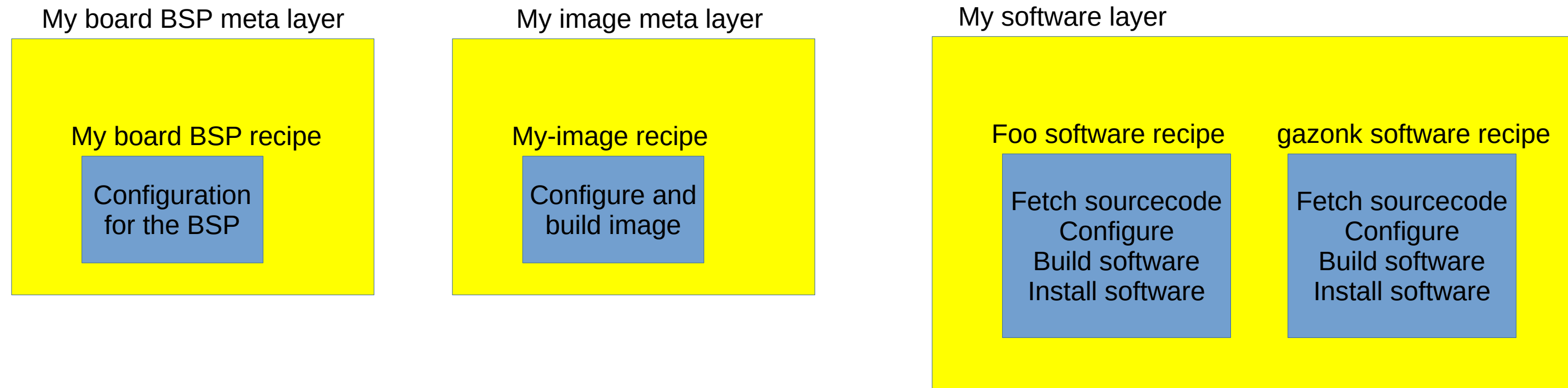
Collection of recipes that contain extensions and customizations to base systems.



They are directories to look for recipes and added to **BBLAYERS** in `build/conf/bblayers.conf`

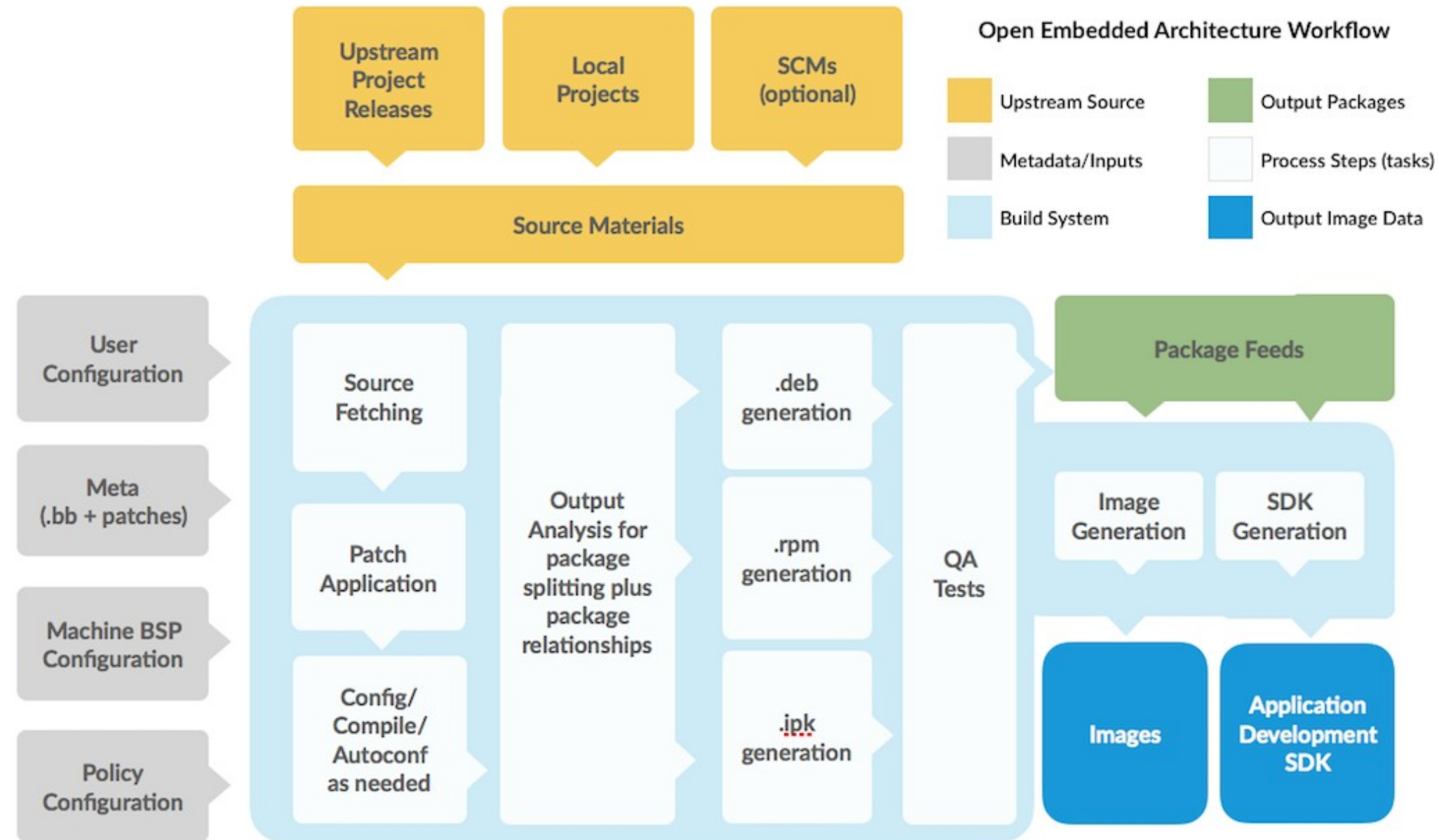
Deploy – Yocto Project

- A meta-layer consists of a collection of recipes.



- To build the software for different boards we switch the BSP layer.

Deploy – Yocto Project



Deploy – Yocto Project

- Yocto downloads all source code for all open source parts and compiles them when you build with Yocto.
- It will build toolchain, bootloader, Linux kernel and the root filesystem.
- To speed up later builds, it saves information from earlier builds in a state cache.
- Developers can share a sstate cache to speed up builds.
- As a result you will get; packages (rpm, deb, ipk), images for different filesystems, archive files (tar, zip).
 - Which of these are created depends on your configuration.



Deploy – Yocto Project

- Yocto project is well supported and developed by many companies.
- Yocto project is an industry standard.
- Yocto project is flexible.
- Yocto has a learning curve to learn the basics.
 - Learning Yocto is well worth it.
- For small projects it can be easier to use buildroot.
- More and more companies are using Yocto to build Linux for their embedded systems.
- With Yocto Project and some added layers we can have atomic updates.
- With Yocto Project (and buildroot) we have support for Software Bill of Materials (SBOM), software supply chain, and can manage licenses.



Deploy – Yocto Project

- Example: a virtual machine
 - # check out poky (Yocto 5.0.x)
git clone -b scarthgap git://git.yoctoproject.org/poky.git
 - # Go to poky and set up the environment
cd poky
source oe-init-build-env
 - # edit the MACHINE variable in the file conf/local.conf to for example
MACHINE ?= "**qemuarm**"
 - # build the image core-image-minimal
bitbake core-image-minimal
 - # take a long coffe break
 - # Test run the system in the qemu virtual machine
runqemu core-image-minimal

You should try this at home.

Deploy – Yocto Project

- Example: Beaglebone black
 - # check out poky (Yocto 5.0.x)
git clone -b scarthgap git://git.yoctoproject.org/poky.git
 - # Go to poky and set up the environment
cd poky
source oe-init-build-env build-bbb
 - # edit the MACHINE variable in the file conf/local.conf to for example
MACHINE ?= "**beaglebone-yocto**"
 - # build the image core-image-minimal
bitbake core-image-minimal
 - # take a long coffe break
 - # Install the bootloader, kernel and root filesystem image to a SD card and boot the Beaglebone black.

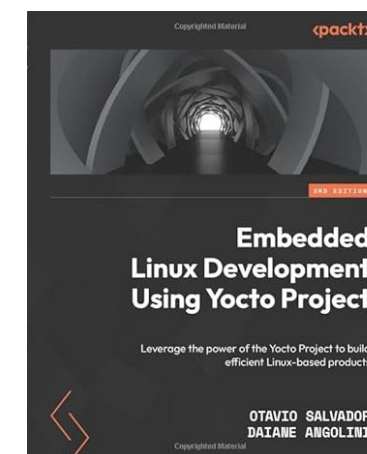
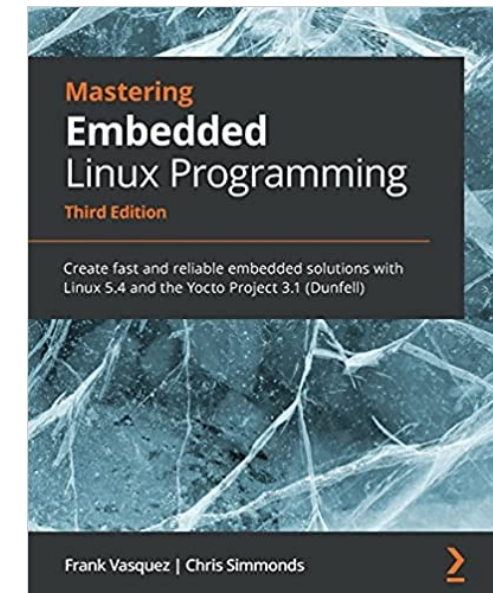
Links and books

- Links:

- <https://www.yoctoproject.org/> Yocto Project
- <https://buildroot.org/> Buildroot
- <https://www.linuxfoundation.org/> Linux Foundation
- <https://www.embeddedlinuxconference.com/> Previous Linux Foundation Embedded Linux Conferences
- <https://kernel.org/> The Linux kernel
- <https://github.com/u-boot/u-boot> U-Boot
- <https://www.lysator.liu.se/~kjell-e/tekla/linux> My pages about Linux and embedded Linux.

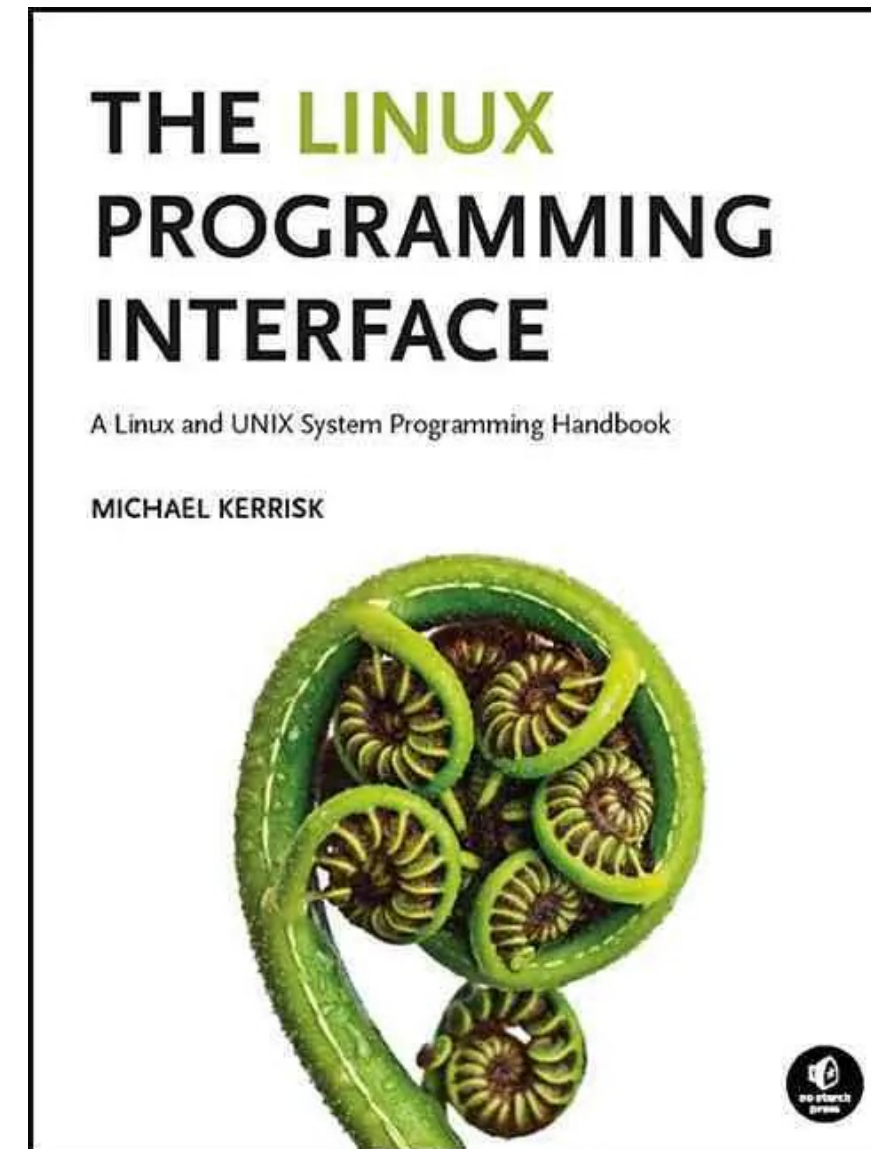
- Books:

- Mastering embedded Linux programming Third Edition, 2021
By Chris Simmonds & Frank Vasquez
- Embedded Linux Development Using Yocto Project, 3rd ed.
By Otavio Salvador, Daiane Angolini



Links and books

- Books:
 - The Linux programming interface
 - By: Michael Kerrisk
- If you are going to program in Linux, this is a book you should consider buying.



Questions?

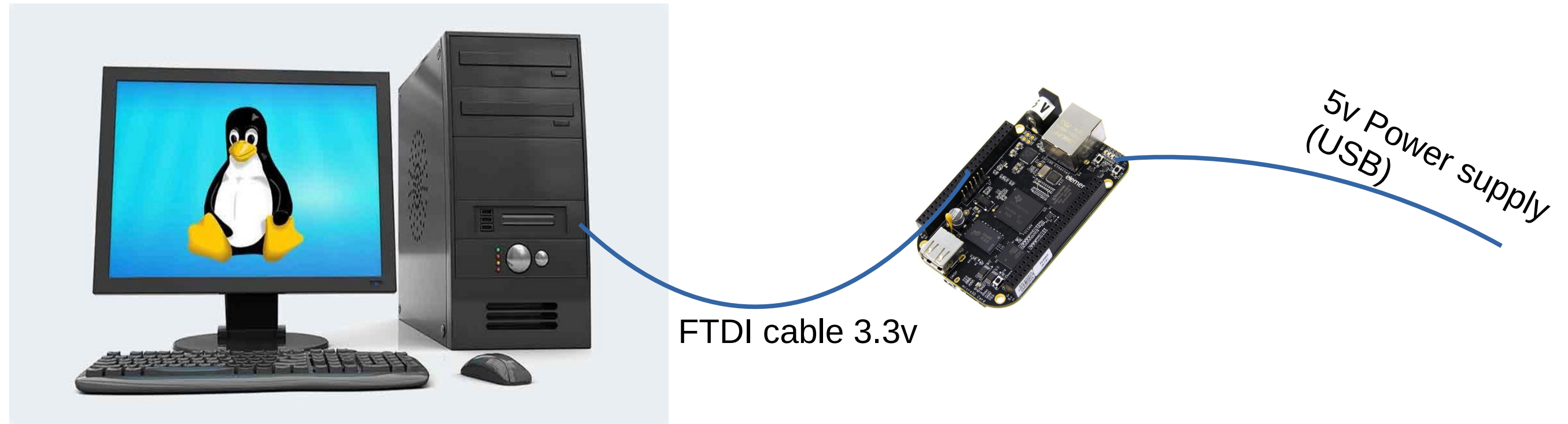


MindRoad



Thank you!

Beaglebone black



```
picocom -b 115200 /dev/tty/USB0
```

