Yocto Project, un generatore automatico di distribuzioni linux embedded

Marco Cavallini

whoami

https://koansoftware.com

eMail : m.cavallini@koansoftware.com

- Linux developer and evangelist since 2000
- Openembedded board member since 2009
- Yocto Project ambassador since 2019

Company introduction

- Engineering company
  - In business since 1996
- Based in Bergamo, Italy
- Serving customers worldwide
- Highly focused and recognized expertise
  - Embedded Linux
  - Linux kernel
  - Yocto Project build system
- https://koansoftware.com

KOAN engineering services

Bootloader / firmware development

U-Boot, HAB
OP-TEE, TF-A, .../

Linux kernel porting and driver development

Linux BSP development, maintenance and upgrade

Embedded Linux build systems

Yocto, OpenEmbedded

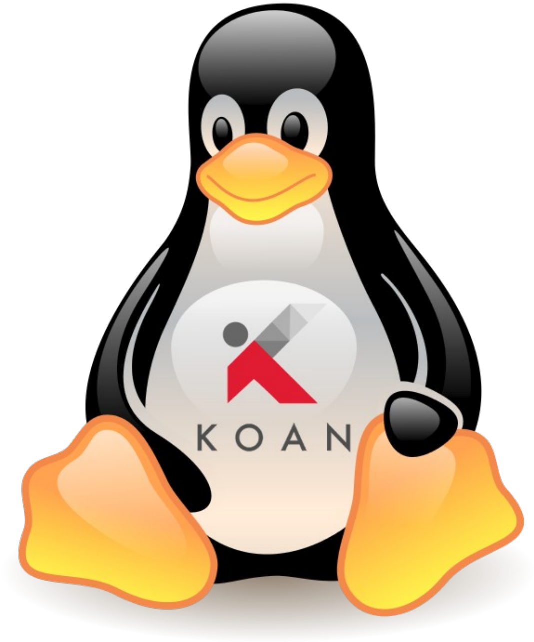
Embedded Linux integration

OTA updates, Real-time, Security, Multimedia, Networking

Training courses

in Italian and English
Linux embedded,
Yocto Project,
Device drivers

- LINUX EMBEDDED



Linux Embedded requirements

- The essential requirements for an embedded Linux system:
  - Small size
  - Busybox, etc...
  - Riproducible
  - Automatic build system
  - Reliable
  - Cross-compilation toolchain / SDK

Creation of a distribution

- Typical approaches to create an embedded Linux distribution:
  - Do It Yourself (DIY) AKA Linux From Scratch (LFS)
  - Downscaling (Debian, Fedora, Slack)
  - Legacy ARM distros (Debian, Fedora)
  - Tools for automatic generation...

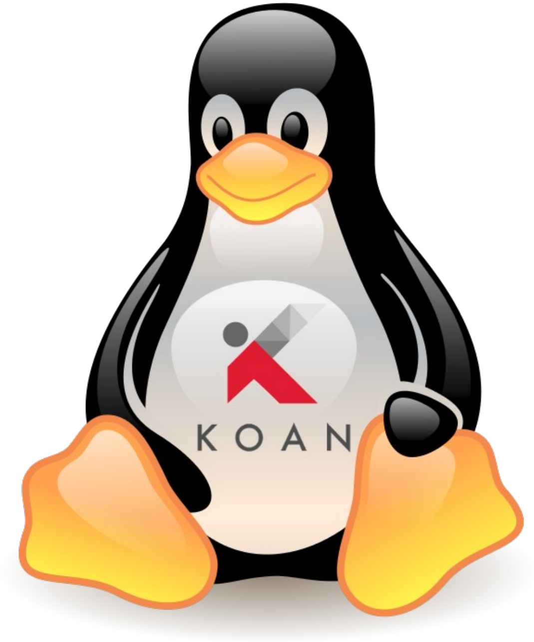
Automatic build tools

- Some of the best-known tools for the automatic generation of embedded Linux systems:
  - Crosstool (the precursor)
  - Crosstool-ng
  - PTXdist
  - Scratchbox
  - uClinux
  - OpenWRT
  - Buildroot
  - OpenEmbedded
  - Yocto Project (Poky)

- YOCTO PROJECT

yocto

PROJECT



Yocto Project benefits

- The strong points of the Yocto Projects are:
  - **Modular**, thanks to the layers concept
  - **Open**, thanks to the widest community without vendor lock-in
  - **Standard**, because is the de-facto standard system
  - **Agnostic**, because can run seamless on every architecture
  - **Safe**, from the legal point of vew, thanks to license check
  - **Reliable**, thanks to a CVE check feature

Yocto vs Binary distro

	Yocto Project	Binary distro (Debian)
Packages availability	Based on the available recipes or customized	Wide package selection in binary format
Learning curve	Not user friendly	Easy to add/remove binary packages
System flexibility	Full, packages built from scratch	Limited, using fixed configuration
Build time	Very long the first time	Fast, prebuilt binary packages
Architecture optimization	Optimized for the specific SoC	Optimized for generic Arch (x86, ARM)
Scalability	Designed for embedded	Designed for PCs
License check	Accurate check	Not supported
Image size	~50MB (minimal-image)	~1GB

Origin

- The **OpenEmbedded** project (OE for short) is an open source project created by Chris Larson, Michael Lauer, and Holger Schurig, merging the achievements of **OpenZaurus**.
- OpenEmbedded is a software framework used for creating Linux distributions aimed for, but not restricted to, embedded devices. The build system is based on BitBake recipes which behave like Gentoo Linux ebuilds.

What is the Yocto Project

- It is an open source project initiated by the **Linux Foundation** in **2010** and is still managed by one of its fellows: Richard Purdie.
- The Yocto Project is an open source collaboration project that helps developers create custom Linux-based systems...

https://docs.yoctoproject.org/overview-manual/yp-intro.html#what-is-the-yocto-project

Yocto Project

- Umbrella organization under the Linux Foundation

- Backed by many companies interested in making **Embedded Linux** easier for the industry

- Co-maintains **OpenEmbedded Core** and other tools (including **opkg**)

Yocto Project member organizations

Members

Technical Partner: openembedded

Associate Member: LONG TERM SUPPORT INITIATIVE

Platinum: intel, Texas Instruments, COMCAST, Microsoft, facebook, cisco, arm, XILINX

Gold: WIND, Mentor Graphics A Siemens Business, Juniper Networks, RENESAS, AUTOMOTIVE GRADE LINUX

Silver: ENEA, LINEO Solutions, NXP, ST life.augmented, MontaVista, LG Electronics, Linaro, AMD, DELL

Yocto Project governance model

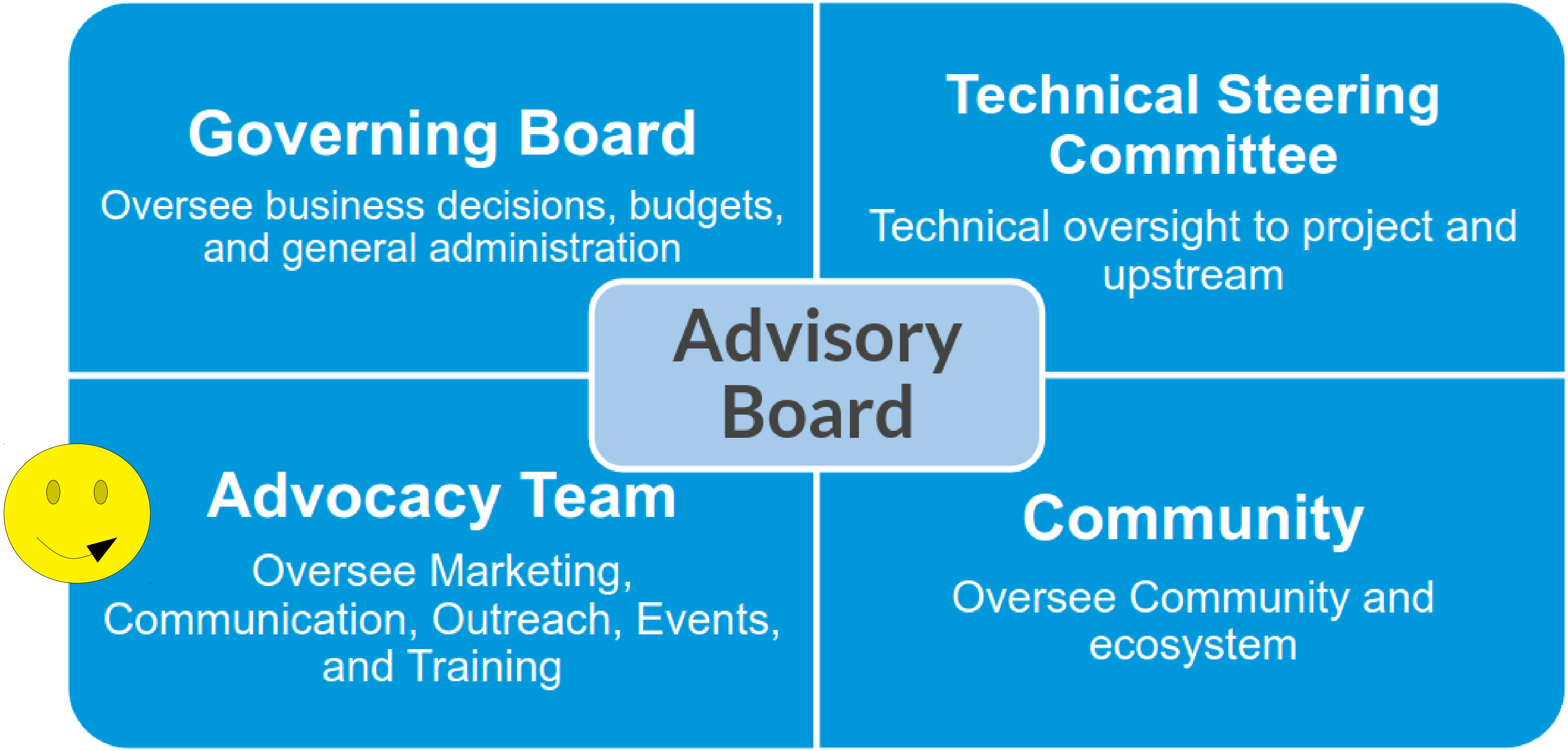
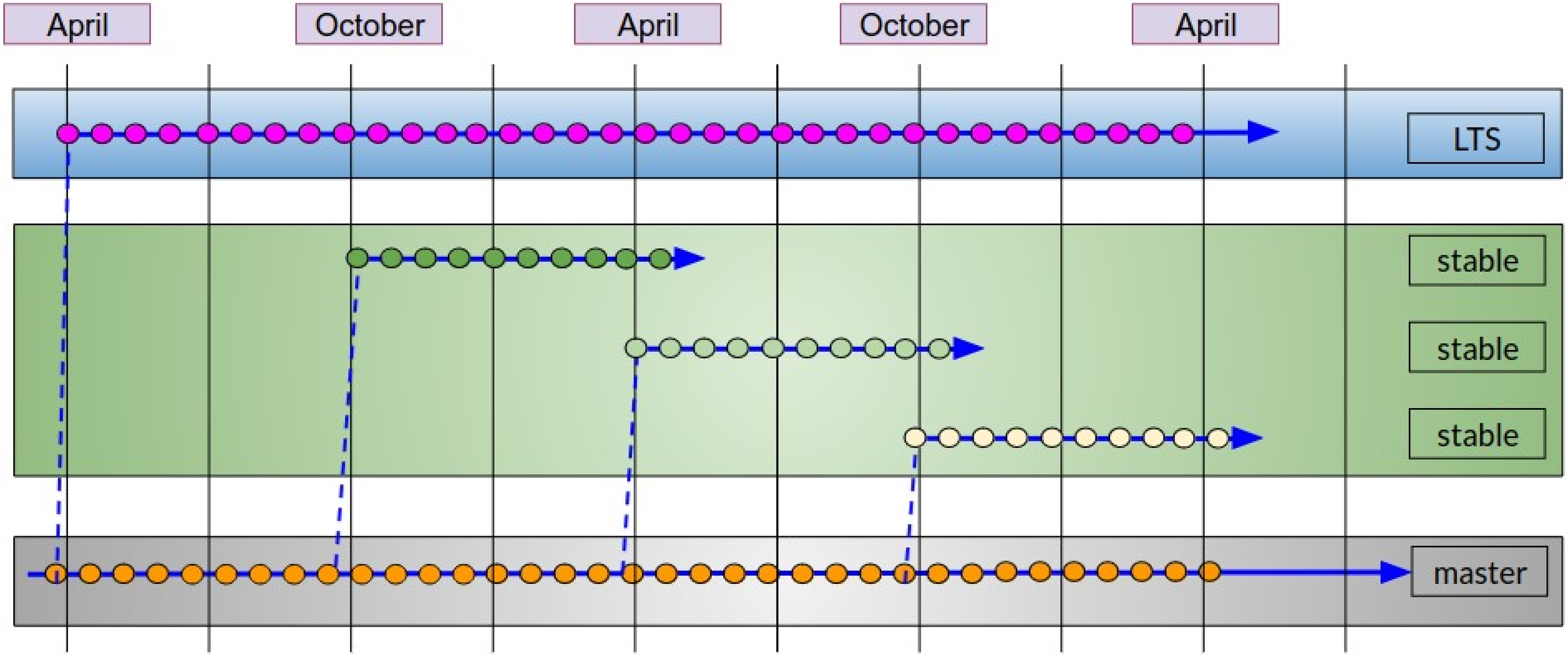


Image credits: David Reyna, Wind River & Nicolas Dechesne, Linaro - ELC North America, 2020

6 months release cycle overview



- See next slide...

Release cycle details

- A new version is released every 6 months, and maintained for 7 months
- LTS versions are maintained for 2 years,
  and announced before their release.
- Each release has a codename such as **kirkstone** or **dunfell**, corresponding to a release number.
- A summary can be found at
  https://wiki.yoctoproject.org/wiki/Releases

- See next slide…

Yocto Project versions

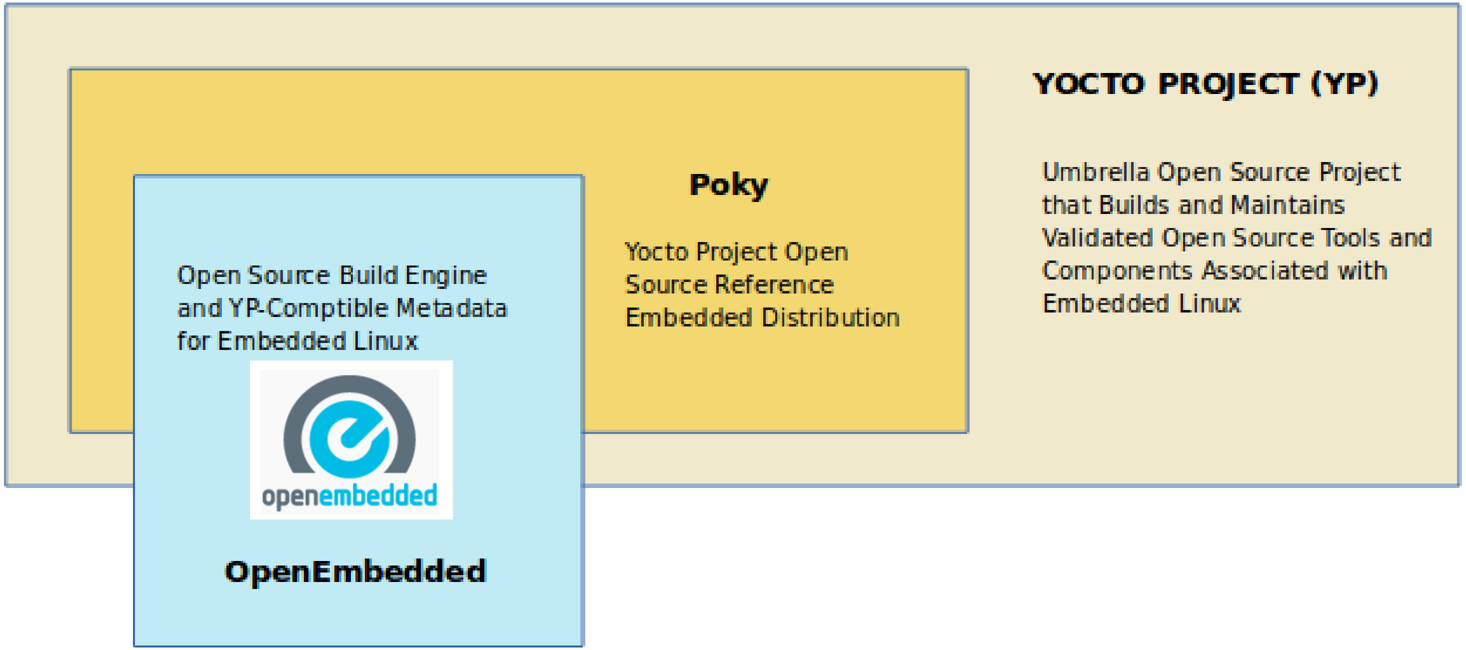
Codename	Yocto Project Version	Release Date	Current Version	Support Level
Scarthgap	5.0	April 2024		Future - Long Term Support (until April 2028)
Nanbield (like 'man field')	4.3	October 2023		Future - Support for 7 months (until April 2024)
Mickledore	4.2	May 2023	4.2.3 (September 2023)	Support for 7 months (until November 2023)
Langdale	4.1	October 2022	4.1.4 (May 2023)	EOL
Kirkstone (like 'kirk stun')	4.0	May 2022	4.0.13 (October 2023)	Long Term Support (Apr. 2026[1])
Honister	3.4	October 2021	3.4.4 (May 2022)	EOL
Hardknott	3.3	April 2021	3.3.6 (April 2022)	EOL
Gatesgarth	3.2	Oct 2020	3.2.4 (May 2021)	EOL
Dunfell	3.1	April 2020	3.1.28 (September 2023)	Long Term Support (until Apr. 2024[1])
Zeus	3.0	October 2019	3.0.4 (August 2020)	EOL
Warrior	2.7	April 2019	2.7.4 (June 2020)	EOL
Thud	2.6	Nov 2018	2.6.4 (October 2019)	EOL
Sumo	2.5	April 2018	2.5.3 (April 2019)	EOL
Rocko	2.4	Oct 2017	2.4.4 (November 2018)	EOL
Pyro	2.3	May 2017	2.3.4 (July 2018)	EOL

Yocto Project

- **Collection of tools and methods enabling:**
  - Rapid evaluation of embedded Linux on many popular off-the-shelf boards
  - Easy customization of distribution characteristics
- Supports x86_32/64, ARM32/64, MIPS, PowerPC
- Based on technology from the **OpenEmbedded** project
- Layer architecture allows for easy re-use of code

Main components

- The Yocto Project is based on **Openembedded** that provides the core system
- The reference distribution is called **Poky**.

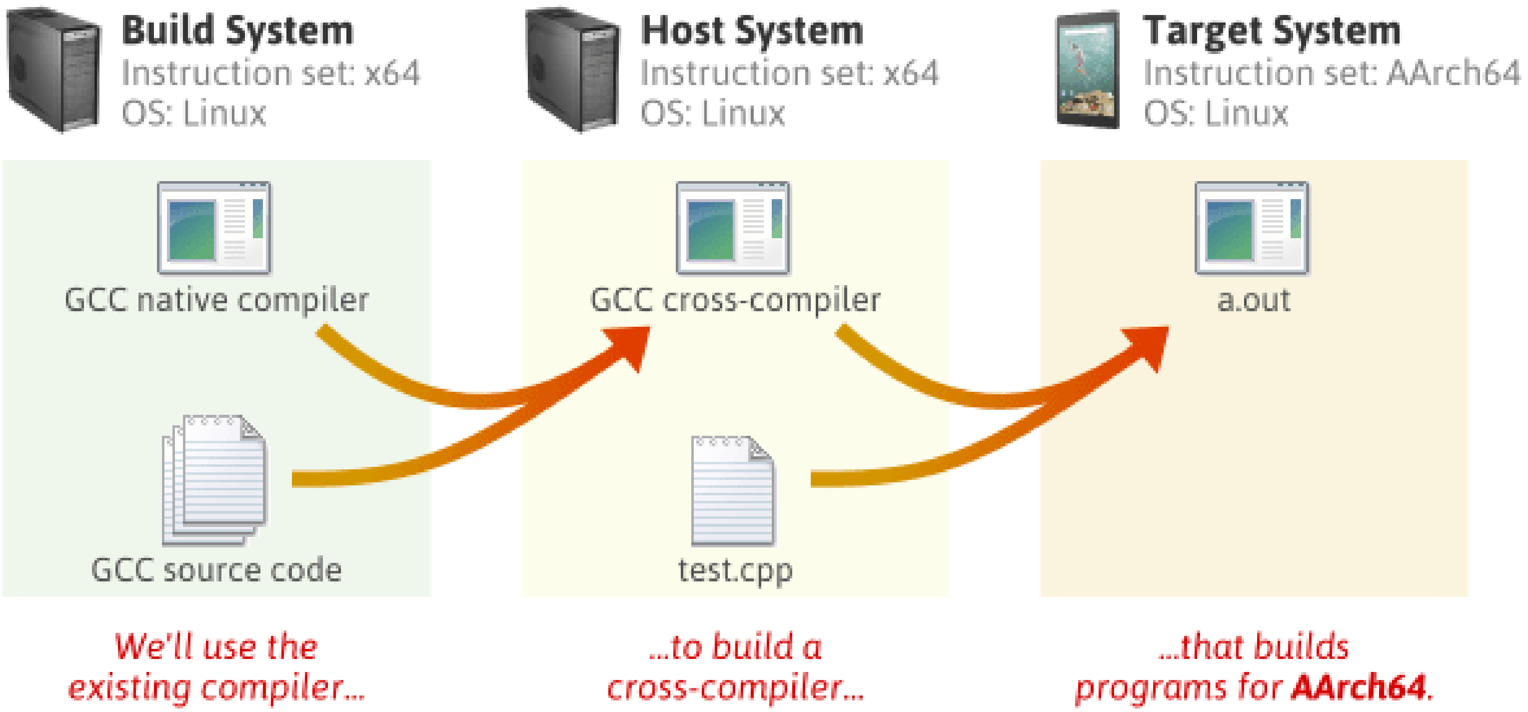


Cross-compilation critical aspects

- In the case of embedded systems, what makes these procedures difficult are the following aspects:
- **Cross-compilation**: Cross-compiling is difficult, and much software has no support for cross-compilation - all packages included in Yocto Project are cross-compiled;
- **Target and Host are different**: this means that you cannot compile a program and then run it – it is compiled to work on the target system, not on the compilation Host system.
- **Toolchains** (compiler, linker, etc...) are often difficult to compile. Cross toolchains are even more difficult. In fact you typically tend to download a binary toolchain made by someone else.

How to build using a Cross-Compiler

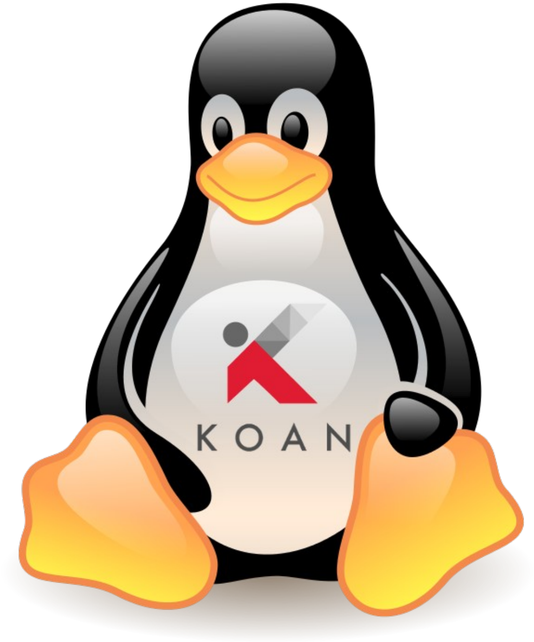
- A cross-compiler is a compiler that builds programs for another machine/architecture



Technical features

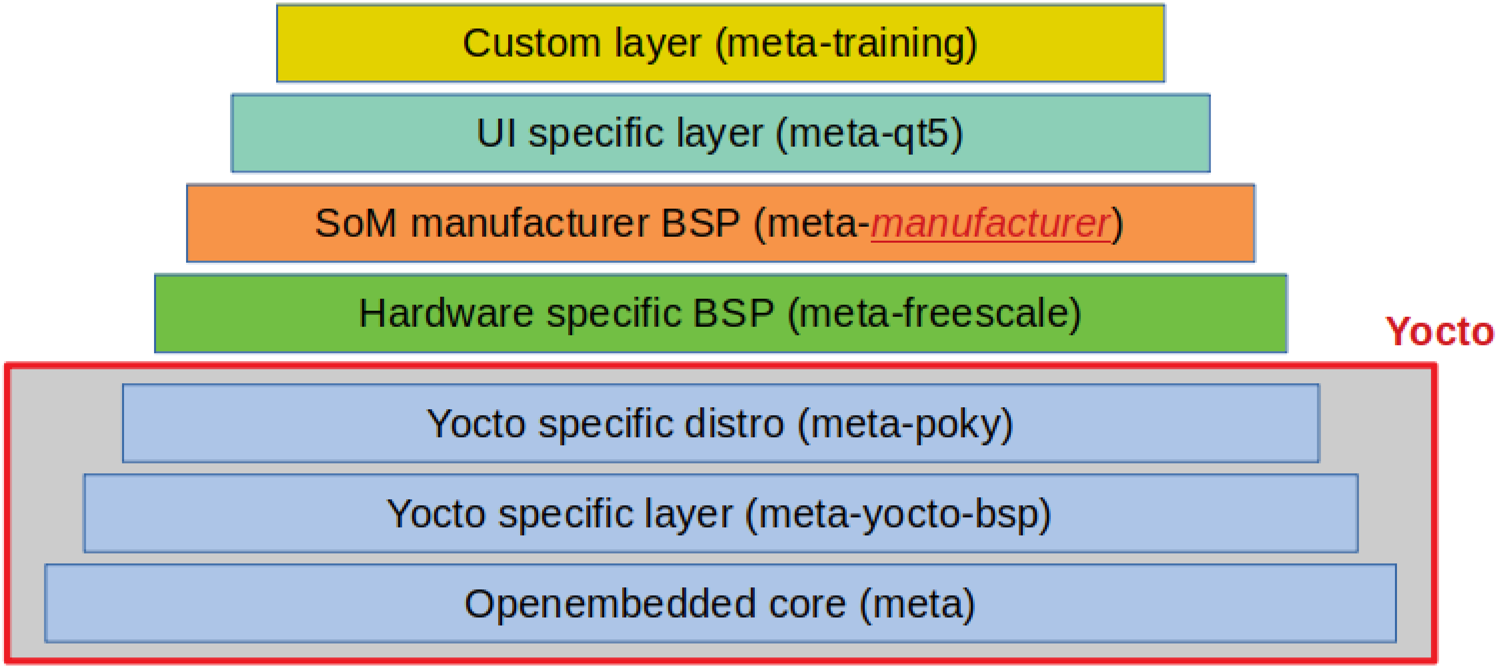
- Of course there's more to it than just building packages, some of the features supported by Yocto Project include:
- Support for glibc, musl and others;
- Generation for **different target devices** from a single code base;
- Automate everything needed to compile and/or run the package (build its **dependencies**);
- Creation of flash disk images (ext4, gz, UBI, wic, etc…);
- Support for various **packaging** formats (deb, rpm, ipk);
- **Automatic generation** of all necessary cross-compilation tools;

- TECHNICAL ASPECTS



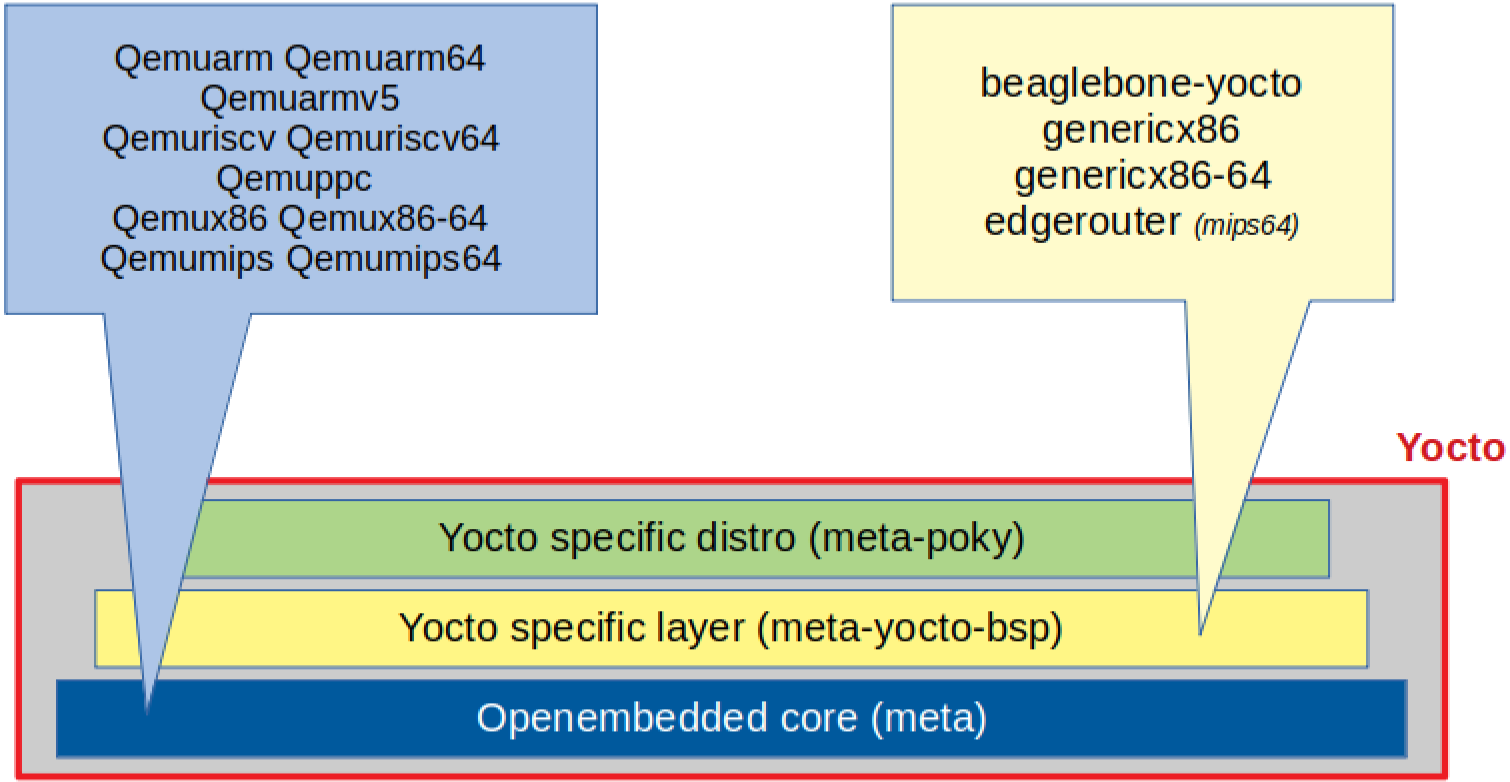
Yocto Project layers model

- The Yocto Project’s Layer Model is a development model for embedded Linux creation that distinguishes it from other simpler build systems.



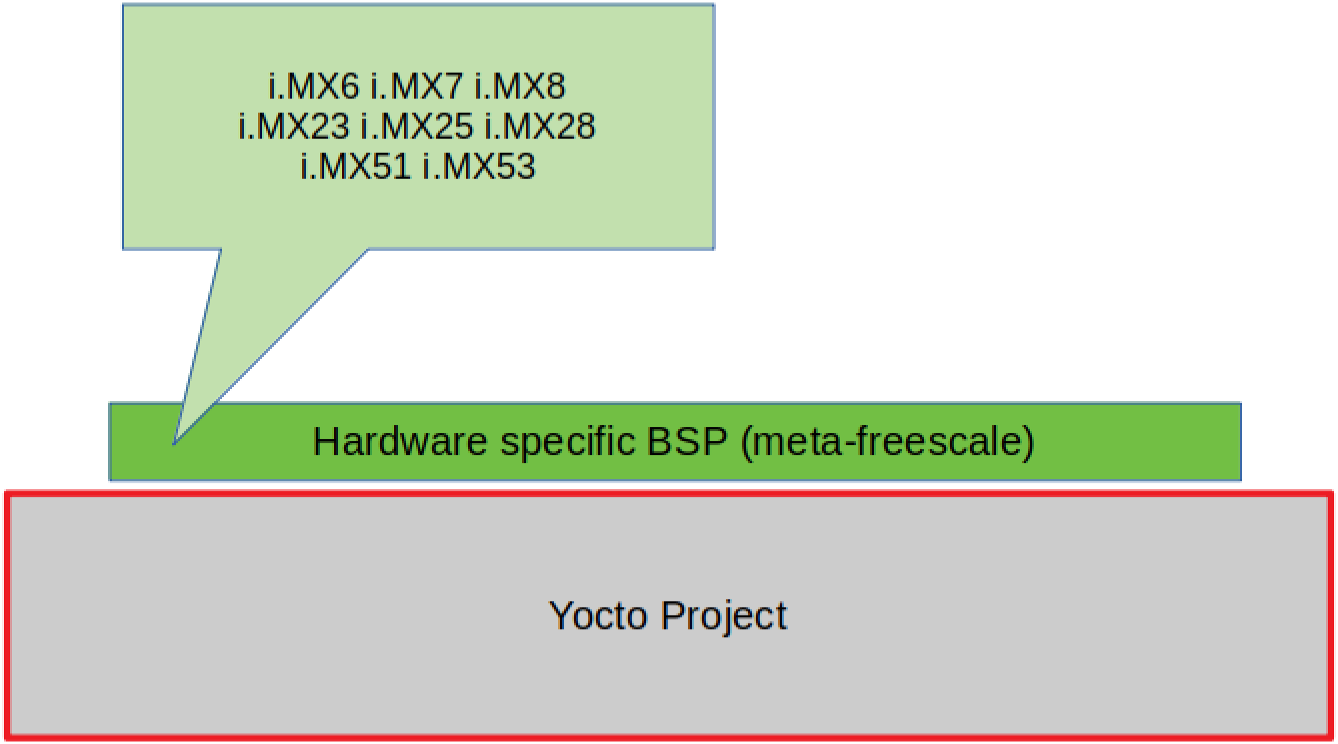
The BSP layers in Yocto Project

- Layers that are providing some specific BSP



The Freescale/NXP BSP layer

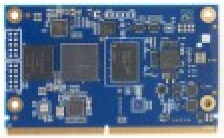
- Layer that provides the BSPs for NXP evaluation boards



The manufacturer’s BSP layer

- Layer that provides the BSPs for a specific manufacturer’s boards

board-a
board-b
board-c



SoM manufacturer specific BSP (meta-*manufacturer*)

Hardware specific BSP (meta-freescale)

The Yocto Project lexicon (1)

- **Recipe** Pron. /ˈresəpi/ (http://www.oxfordlearnersdictionaries.com/definition/english/recipe)
  - Describes how to fetch, configure, compile and package applications and images. They have a specific syntax.
- **Layer**
  - Is a set of recipes, matching a common purpose
  - Sometimes used for board support (BSP) or additional libraries
- **Task**
  - Defines functionalities pre-defined into classes

The Yocto Project lexicon (2)

- **Bitbake**
  - the **build engine**. It is a task scheduler, like **make**. It interprets configuration files and recipes (also called metadata) to perform a set of tasks, to download, configure and build specified packages and filesystem images.
- **OpenEmbedded-Core**
  - a set of base layers. It is a set of recipes, layers and classes which are shared between all OpenEmbedded based systems.
- **Poky distribution**
  - the reference system. It is a collection of projects and tools, used to bootstrap a new distribution based on the Yocto Project.

Bitbake tasks (short list)

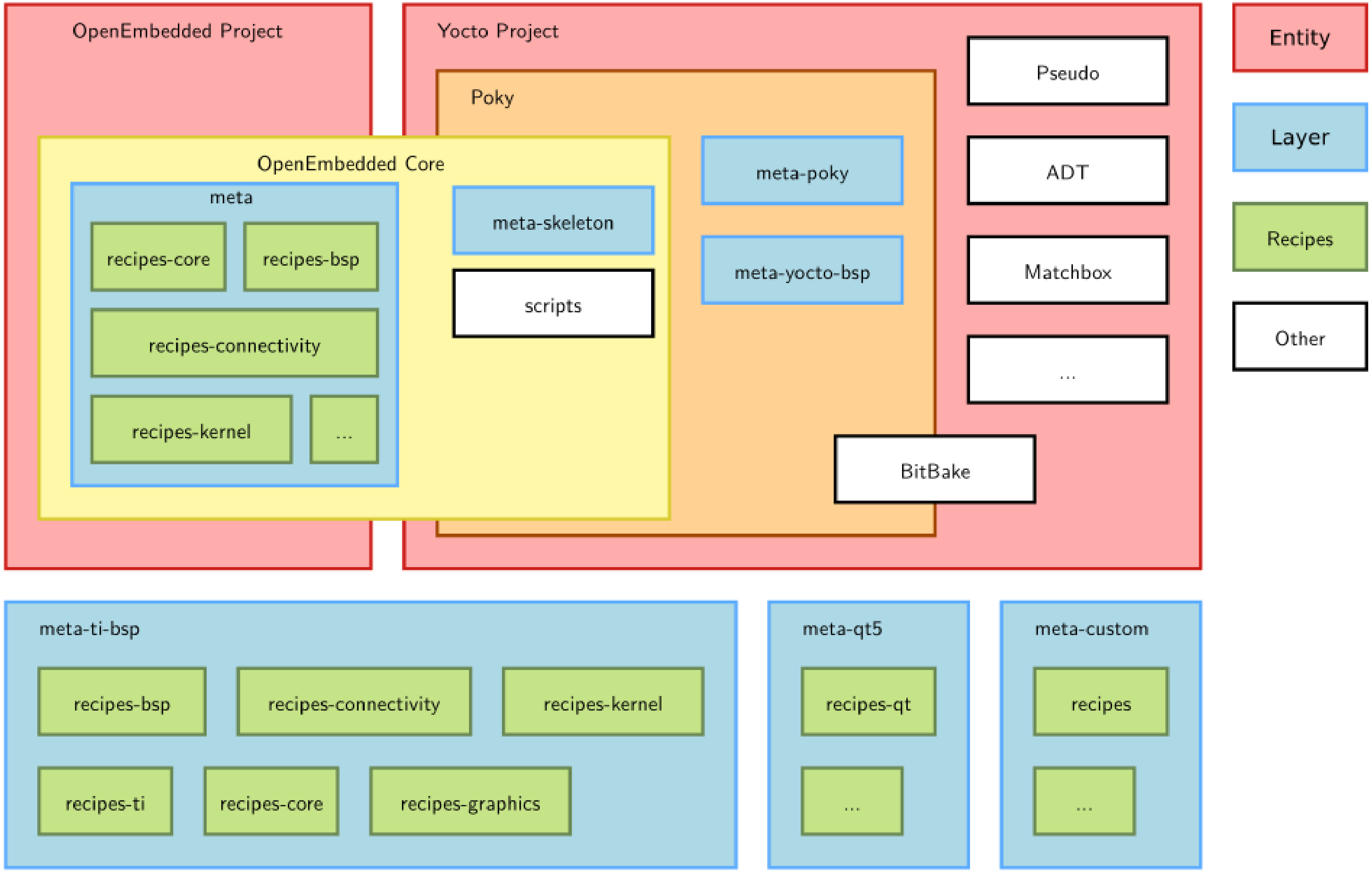
- Tasks are executed for every package included in the Operating System
- Every package is managed by its recipe

Definitions

- **Image**
  - Defines the name of the final image for rootfs
- **Machine**
  - Defines the name of the hardware in use
- **Distro**
  - Defines particular software settings

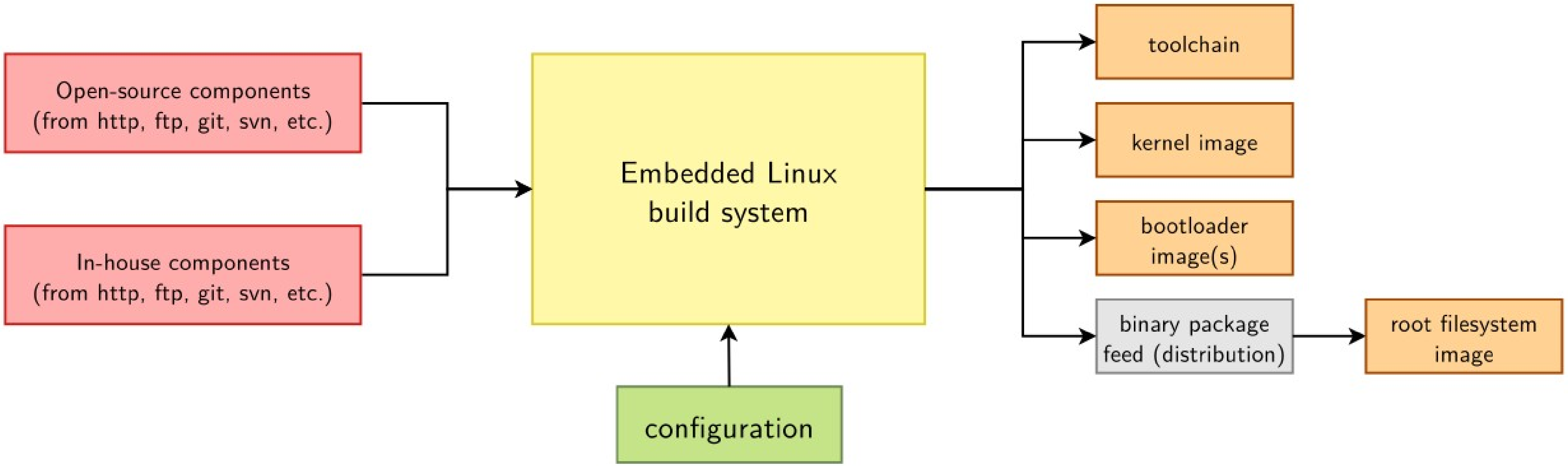
- Those definitions are describing the ‘**triplet**’
- **MACHINE+DISTRO+IMAGE** define <u>how</u> and <u>what</u> to generate

The Yocto Project lexicon

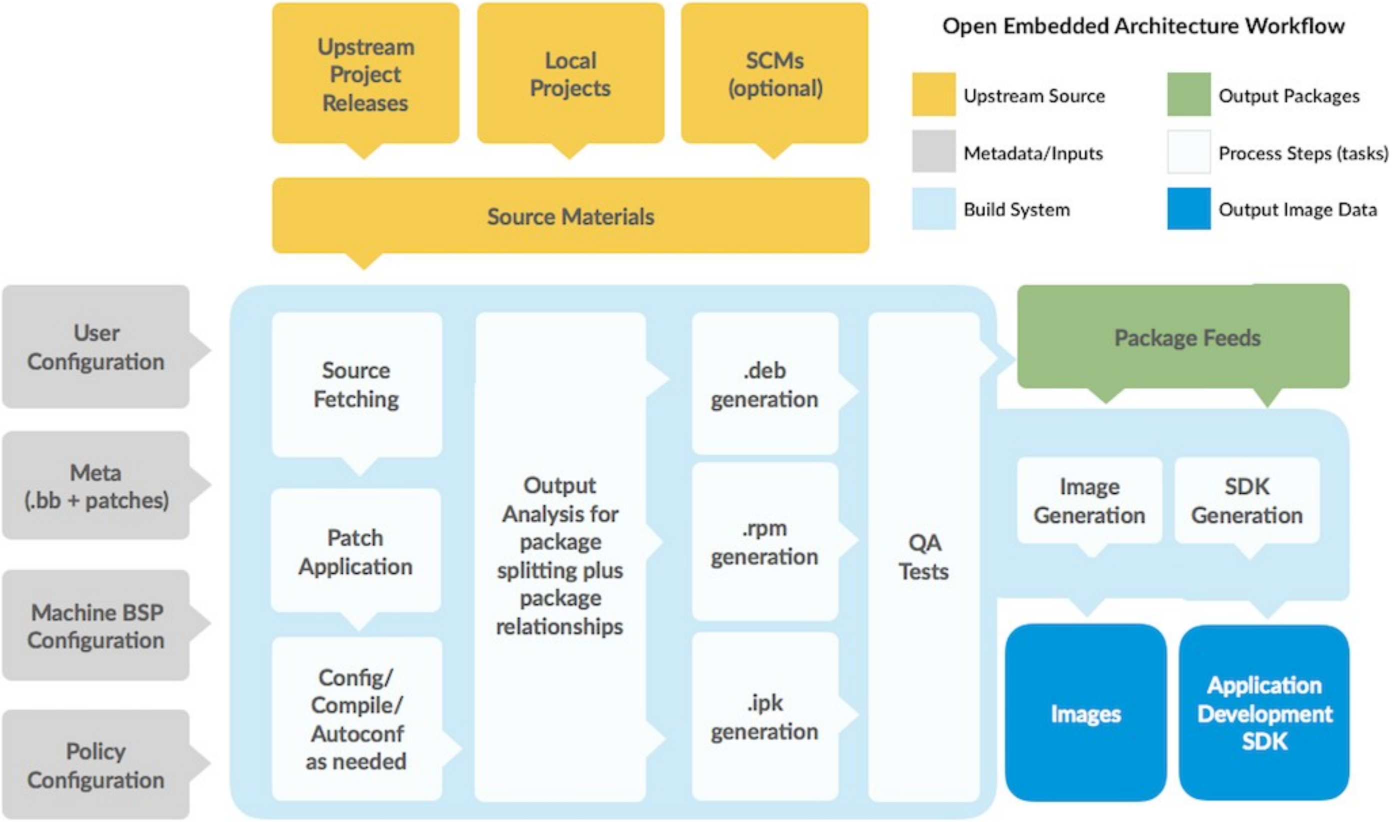


Yocto: principle

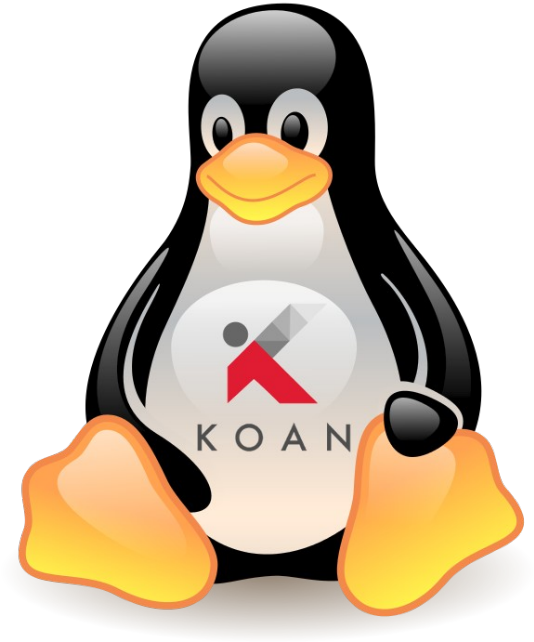
- Yocto Project always builds binary packages (a “distribution”)
- The final root filesystem is generated from the package feeds
- The ‘big picture’ is way more complex



Yocto Project environment overview



- INSTALL AND SETUP



Yocto Project setup from zero

- Setup the Linux machine
- Install the Yocto Project (Poky repo)
- Install the additional layers (if needed)
- Launch bitbake <image-name>
- Program the board (microSD or eMMC)
- Run Linux embedded on the target system

Host machine requirements

- Use a supported distribution (e.g. (L)Ubuntu 22.04)
- https://docs.yoctoproject.org/ref-manual/system-requirements.html#detailed-supported-distros

- CPU : >= 4 cores
- RAM : >= 8 GB
- DISK : >= 150GB – SSD
- Internet connection
- Firewall settings for git

Prepare the Linux machine

```
$ sudo apt install gawk wget git-core diffstat \
unzip texinfo gcc-multilib build-essential \
chrpath socat  cpio python3 python3-pip \
python3-pexpect xz-utils debianutils \
iputils-ping python3-git python3-jinja2 \
libegl1-mesa libsdl1.2-dev pylint3 xterm
```

https://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html#ubuntu-packages

Install the Poky reference system

- The **Yocto Project** repository is named ‘**Poky**’
- All official projects part of the Yocto Project are available at
  - **https://git.yoctoproject.org/**

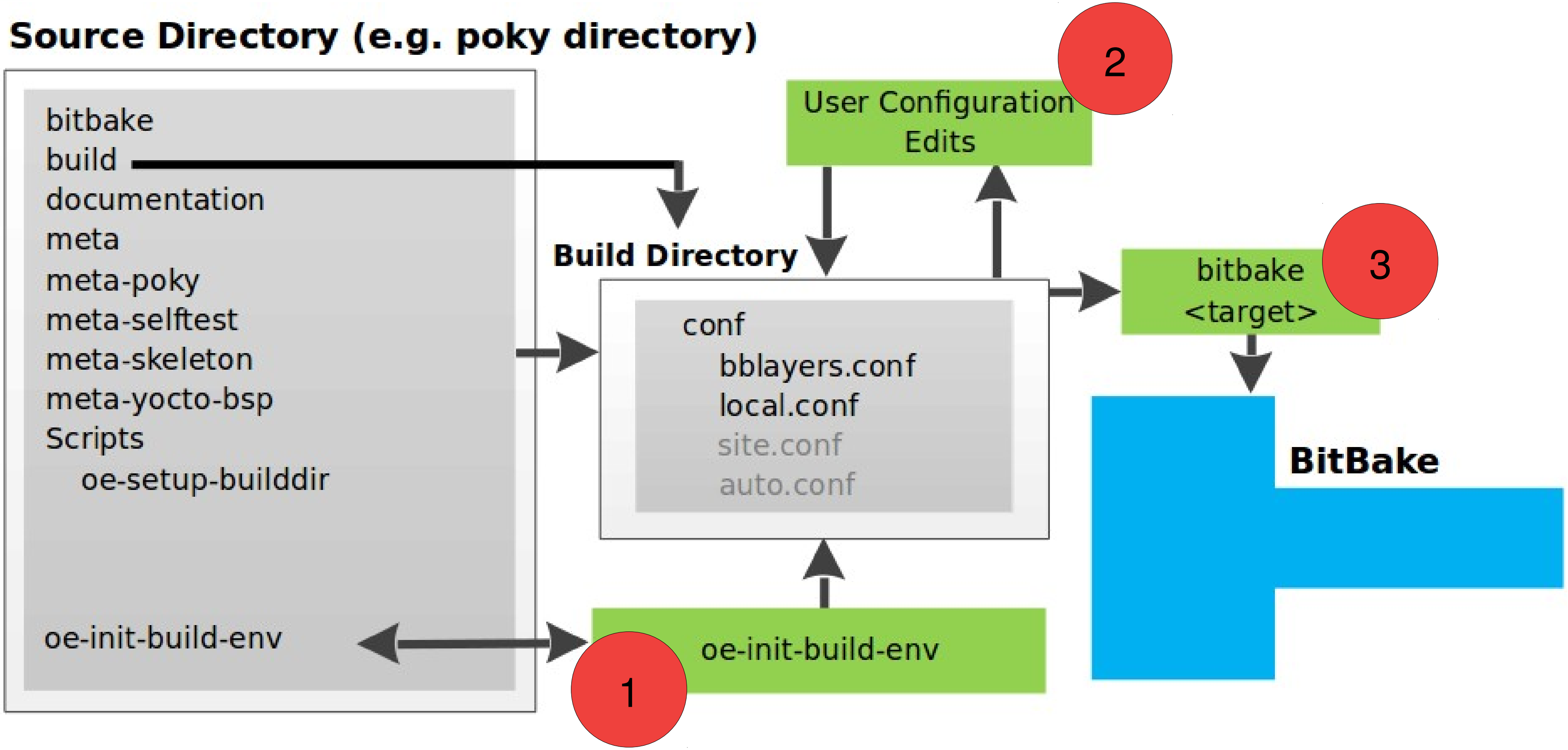
- To download the Poky reference system:
  - **git clone -b kirkstone https://git.yoctoproject.org/git/poky**

branch name

Workspace directory layout

```
└── poky
    ├── bitbake
    ├── contrib
    ├── documentation
    ├── LICENSE
    ├── LICENSE.GPL-2.0-only
    ├── LICENSE.MIT
    ├── MEMORIAM
    ├── meta
    ├── meta-openembedded
    ├── meta-poky
    ├── meta-selftest
    ├── meta-skeleton
    ├── meta-yocto-bsp
    ├── oe-init-build-env
    ├── README.hardware -> meta-yocto-bsp/README.hardware
    ├── README.OE-Core
    ├── README.poky -> meta-poky/README.poky
    ├── README.qemu
    └── scripts
```

Yocto build environment setup



https://docs.yoctoproject.org/overview-manual/concepts.html?highlight=tasks#user-configuration

Workspace directory layout

```
└── poky
    ├── bitbake
    ├── build
    ├── contrib
    ├── documentation
    ├── LICENSE
    ├── LICENSE.GPL-2.0-only
    ├── LICENSE.MIT
    ├── MEMORIAM
    ├── meta
    ├── meta-openembedded
    ├── meta-poky
    ├── meta-selftest
    ├── meta-skeleton
    ├── meta-yocto-bsp
    ├── oe-init-build-env
    ├── README.hardware -> meta-yocto-bsp/README.hardware
    ├── README.OE-Core
    ├── README.poky -> meta-poky/README.poky
    ├── README.qemu
    └── scripts
```

Build directory

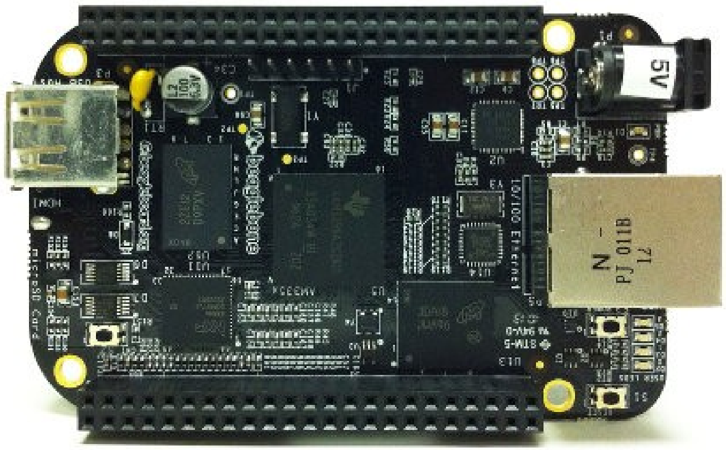
Build directory layout

```
~/yocto/poky/build/
|---cache/                (bitbake cache files)
|---conf/
|     |--bblayers.conf    (bitbake layers)
|     |--local.conf       (local configuration)
`---tmp/                  (Build artifacts)
```

Configure for the target system

- Edit the file **build/conf/local.conf**
- And set the following variables

```
PACKAGE_CLASSES = "package_ipk"
MACHINE = "beaglebone-yocto"
```



Configure for the target system

- The file **build/conf/bblayers.conf**
- Defines the list of layers used by the system

```
BBLAYERS ?= " \
  /home/tux/yocto/poky/meta \
  /home/tux/yocto/poky/meta-poky \
  /home/tux/yocto/poky/meta-yocto-bsp \
  /home/tux/yocto/poky/meta-openembedded/meta-oe \
  "
```

Add packages into the image

- Edit the configuration file

```
$ vi conf/local.conf
```

- Add new packages into the final image

```
IMAGE_INSTALL:append = " i2c-tools"
```

- Always remember to set the Yocto build environment

```
$ cd $HOME/yocto/poky
$ source oe-init-build-env
```

Build an image

- Building a minimal image
- This will generate all the Linux Operating System components

```
$ bitbake core-image-minimal
```

- The build artefacts will be located into

`tmp/deploy/images/`*`MACHINENAME`*`/`

Build output

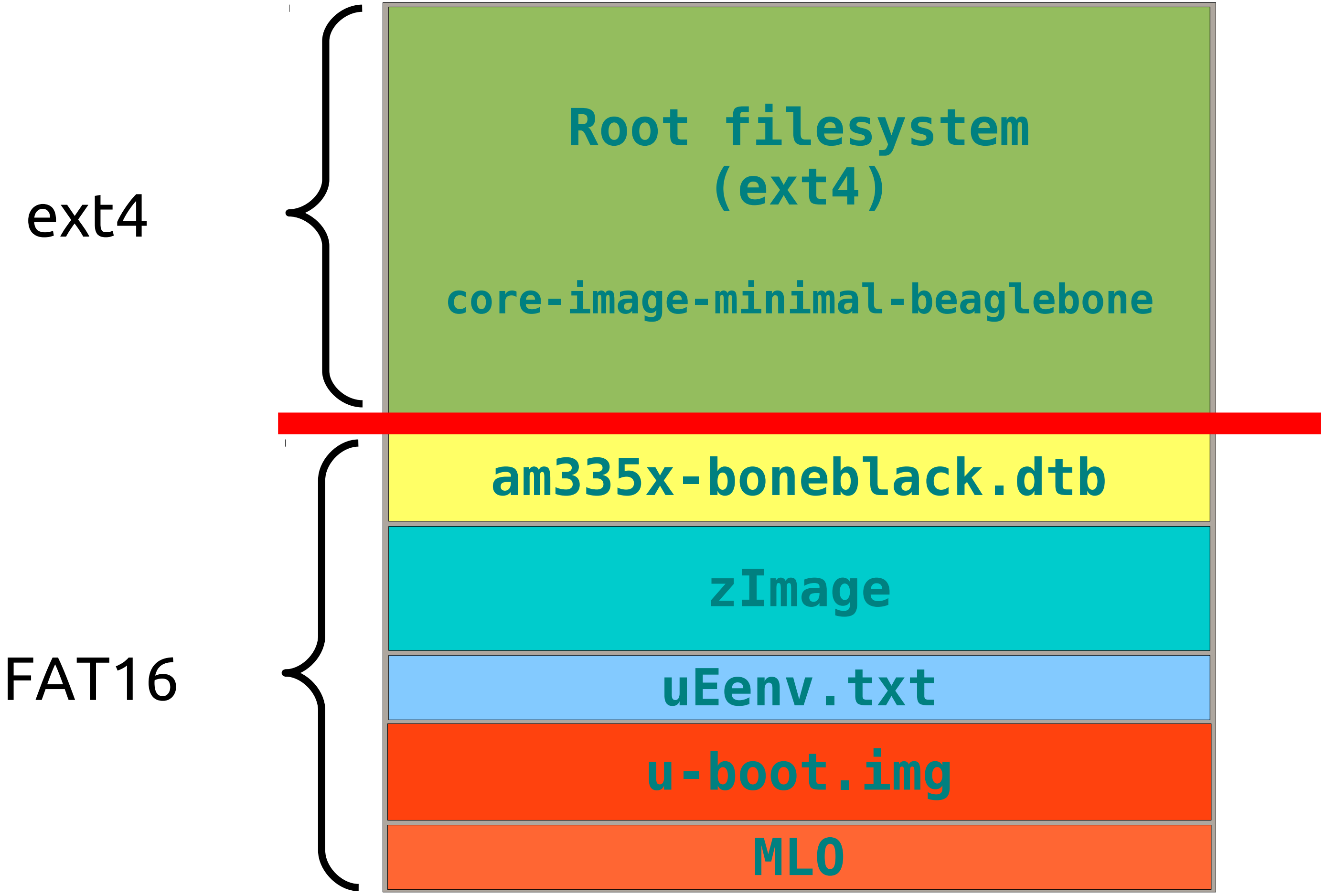
```
Build Configuration:
BB_VERSION           = "1.46.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "universal"
TARGET_SYS           = "arm-poky-linux-gnueabi"
MACHINE              = "qemuarm"
DISTRO               = "poky"
DISTRO_VERSION       = "3.1.25"
TUNE_FEATURES        = "arm armv7ve vfp thumb neon callconvention-hard"
TARGET_FPU           = "hard"
meta
meta-poky
meta-yocto-bsp       = "dunfell:a631bfc3a38f7d00b2c666661a89a758a0af9831"
meta-oe              = "dunfell:e39b002df9675776cc99dccdcac07607ce783b15"
```

Anlayze the built artefacts

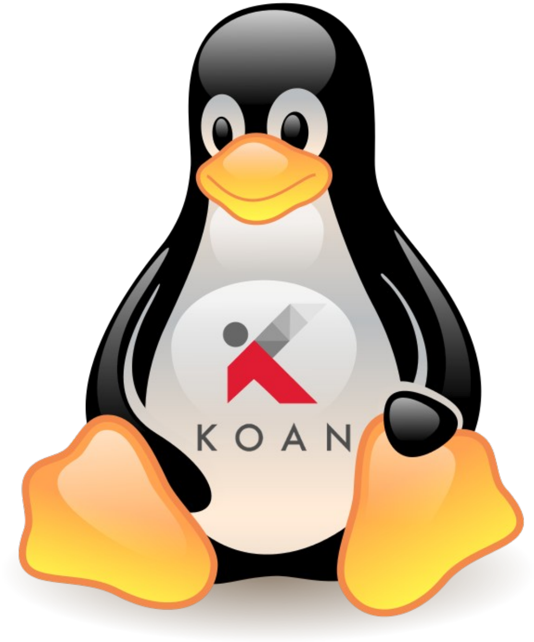
- `$ cd tmp/deploy/images/beaglebone-yocto`
- ```
  core-image-minimal-beaglebone.jffs2
  core-image-minimal-beaglebone.manifest
  core-image-minimal-beaglebone.tar.bz2
  MLO
  u-boot.img
  zImage
  zImage-am335x-boneblack.dtb
  ```

* reduced list
  ```

Typical content of the microSD image



- EXTEND WITH A GRAPHICAL FRAMEWORK



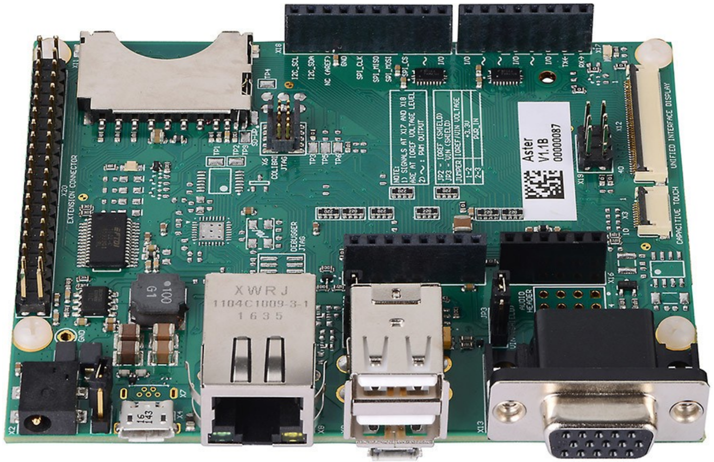
Let’s run Flutter on an i.MX6DL board

yocto PROJECT

Flutter

Toradex Colibri i.MX6

- The **Toradex Colibri i.MX6** dual-core
  - NXP/Freescale i.MX6 Cortex-A9 1GHz
  - 512MB DDR3 RAM
  - 4GB eMMC flash
  - available in commercial or industrial -40 to 85°C versions



- Follow Toradex installation instructions
- https://developer.toradex.com/linux-bsp/os-development/build-yocto/build-a-reference-image-with-yocto-projectopenembedded
-

```
$ mkdir ${HOME}/yocto-tdx
$ cd ${HOME}/yocto-tdx

$ repo init -u \
   git://git.toradex.com/toradex-manifest.git \
   -b kirkstone-6.x.y -m tdxref/default.xml

$ repo sync
```

- Setup the environment
- Note: Toradex uses a different setup script with the same effect

```
$ source export
```

- Define the board

```
MACHINE = "colibri-imx6"
```

* minor setup details are omitted

Build Flutter using Yocto Project

- To build a Flutter-enabled Linux distribution, you need several Yocto layers:
- **poky** layer, containing the base content of openembedded-core
- **meta-flutter** layer, which contains all the Flutter related recipes
- **meta-clang** layer, which is needed by the meta-flutter layer, as Flutter is built using the Clang compiler
- The metalayer for your MACHINE you are using, provided by the silicon vendor

Find available layers and recipes

OpenEmbedded Layer Index

Submit layer | Log in

Branch: **kirkstone** | Layers | Recipes | Machines | Classes | Distros

Search layers

Filter layers

Layer name	Description	Type	Repository
openembedded-core	Core metadata	Base	git://git.openembedded.org/openembedded-core
meta-oe	Additional shared OE metadata	Base	git://git.openembedded.org/meta-openembedded
de-ensc-bpi-router	router + telephony bsp	Machine (BSP)	https://gitlab.com/ensc-groups/bpi-router/de.ensc.bpi-router
meta-96boards	BSP Layer for 96boards platforms	Machine (BSP)	https://github.com/96boards/meta-96boards
meta-arm-bsp	BSP layer for Arm reference and virtual platforms	Machine (BSP)	git://git.yoctoproject.org/meta-arm
meta-artesyn	Artesyn MVME platforms	Machine (BSP)	https://github.com/voltumna-linux/meta-artesyn
meta-askoma-bsp	A Board Support Package for Askoma boards	Machine (BSP)	https://github.com/AskomaEmbedded/meta-askoma-bsp

http://layers.openembedded.org

Get meta-flutter

- Clone the layer providing **Flutter**

```
$ git clone -b kirkstone \
https://github.com/meta-flutter/meta-flutter.git
```

- Clone the **Clang** layer needed by Flutter

```
$ git clone -b kirkstone \
https://github.com/kraj/meta-clang.git
```

Add meta-flutter

- Edit the configuration file

```
$ vi conf/bblayers.conf
```

- Add **Flutter** into the list

```
BBLAYERS += " \
   ${TOPDIR}/../layers-extra/meta-clang \
   ${TOPDIR}/../layers-extra/meta-flutter \
   "
```

Add Flutter into the image

- Edit the configuration file

```
$ vi conf/local.conf
```

- Add **Flutter** into the final image

```
IMAGE_INSTALL:append = " \
    flutter-auto \
    flutter-gallery \
    weston-init"
```

Build an image

- Build a minimal image
- This will generate all the Linux Operating System components

```
$ bitbake core-image-minimal
```

- The build artefacts will be located into `tmp/deploy/images/colibri-imx6/`

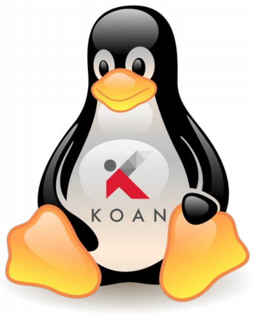
- Then flash the eMMC or the SDcard

- Example running the Flutter demo application with Linux embedded generated with Yocto Project on a board with NXP based system i.MX6DL.

Flutter

*** you can see the live demo at the end of the presentation !**

Thank you!



Questions?

m.cavallini <AT> koansoftware.com

Corrections, suggestions,
contributions and translations are welcome!

https://koansoftware.com/pub/talks/LinuxDay2023/

Follow us on LinkedIn