# Artificial intelligence and Machine Learning on NXP i.MX families using Yocto Project


*Marco Cavallini, KOAN*

EBV Elektronik
An Avnet Company

# Agenda

- Linux embedded
  - Sistemi di generazione automatica
- Yocto Project
  - Storia
    - OpenEmbedded
  - Panoramica
    - Recipes, Layers, Classes, etc...
  - Configurazione
  - Installazione
    - git clone
  - Utilizzo
    - bitbake
  - NXP eIQ software
    - Componenti


yocto
PROJECT

# Requisiti Linux Embedded

I requisiti indispensabili per un sistema linux embedded sono:

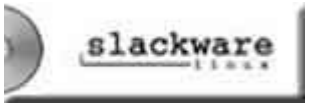
- Dimensione contenuta
  - Busybox, etc...
- Riproducibilità
  - Automatic build system
- Affidabilità
  - Cross-compilation toolchain

# Creazione di una distribuzione

Approcci per creare una distribuzione **Linux embedded** :

- Do It Yourself (DIY) Linux From Scratch (LFS)


- Downscaling (Debian, Fedora, Slack)


- Distro ARM preesistenti (Debian, Fedora)


- Tools per generazione automatica...

# Build Tools

Alcuni dei più noti tool per la generazione automatica di sistemi Linux embedded sono:

- Crosstool *(the precursor)*
- Crosstool-ng
- PTXdist
- Scratchbox
- uClinux
- OpenWRT
- T2 Project
- LTIB *(Linux Target Image Builder)*
- EmDebian
- Buildroot
- OpenEmbedded
- Yocto Project (Poky)


PTXdist

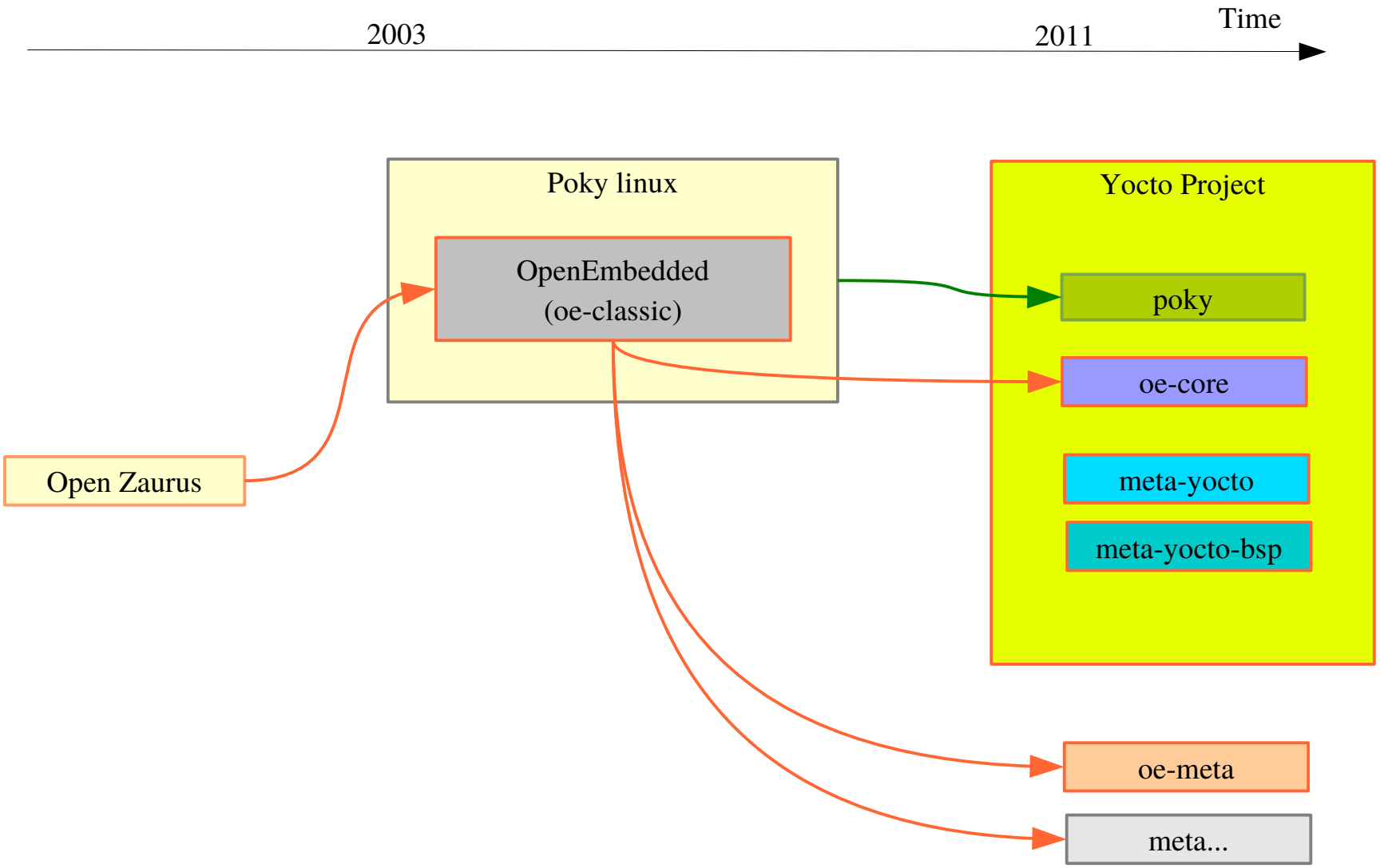

# OpenEmbedded: storia

Il progetto OpenEmbedded è stato creato originariamente nel 2003 da un gruppo di sviluppatori del progetto OpenZaurus ed in particolare da Chris Larson (overall architecture), Holger Schurig (first implementation), e Michael Lauer (first loads of packages and classes).


Altre distribuzioni hanno iniziato ad adottare OE: Unslug, OpenSimpad, GPE Phone Edition, Ångström, OpenMoko e KaeilOS.

Ognuna di queste distribuzioni apporta il proprio bagaglio di esperienze e di specifiche esigenze al progetto OE, aggiungendo pacchetti e architetture supportate.

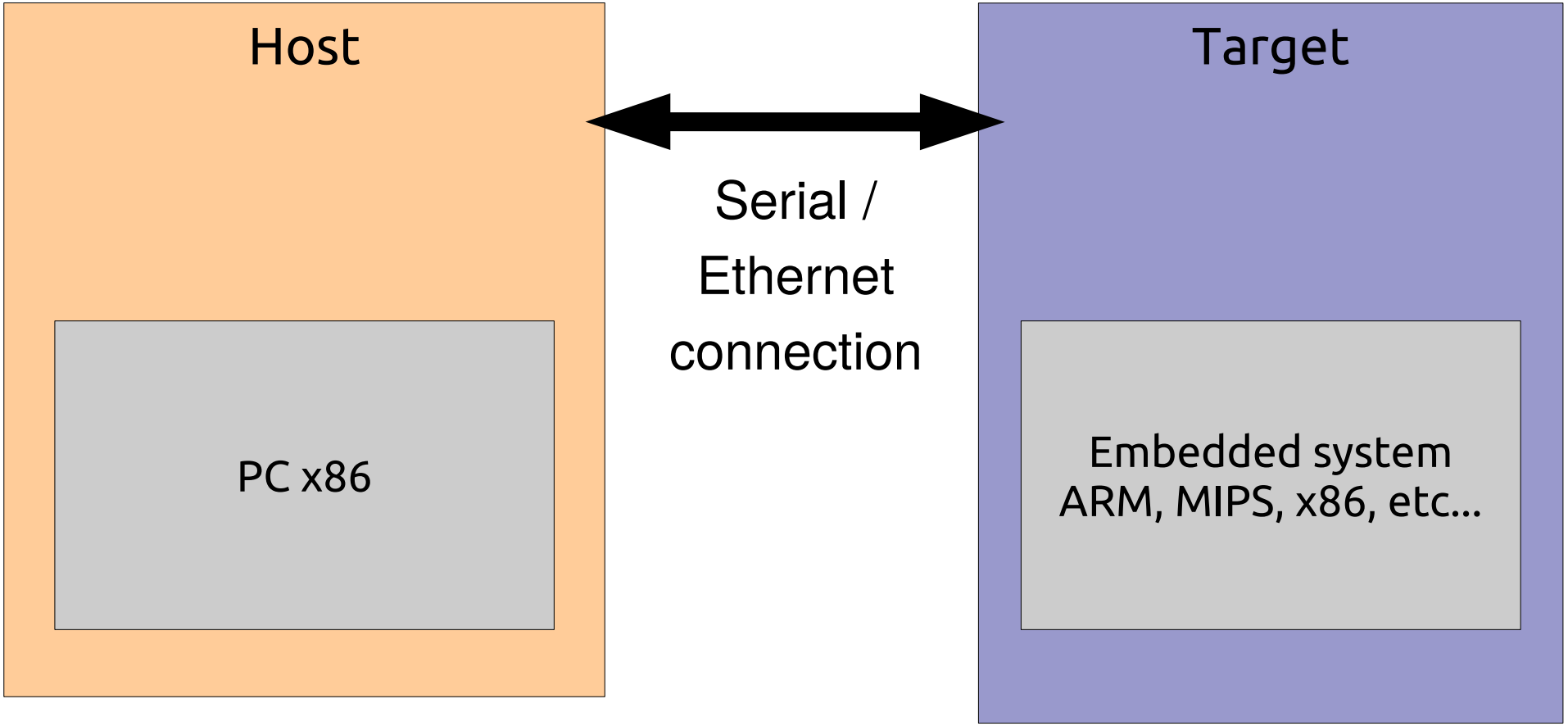
# OpenEmbedded, Poky, Yocto




yocto
PROJECT

# Build system - concetti

Il concetto sviluppato dai **Build System**, è di prendere del software e creare qualcosa che si può eseguire su un altro dispositivo.



# Aspetti critici

Nel caso di sistemi embedded, ciò che rende difficile queste procedure sono i seguenti aspetti:

Cross-compilazione: cross-compilare è difficile, e molto software non ha alcun supporto per la cross-compilazione - tutti i pacchetti inclusi nel OE sono cross-compilati;

Target e l'Host sono diversi: questo significa che non è possibile compilare un programma e poi eseguirlo – esso è compilato per funzionare con il sistema target, non sul sistema di Host di compilazione.

Toolchains (compilatore, linker, ecc...) sono spesso difficili da compilare. Le cross toolchains sono ancora più difficili. In genere si tende a scaricare una toolchain fatta da qualcun altro.

# Caratteristiche

Naturalmente c'è molto di più che la semplice compilazione dei pacchetti, alcune delle caratteristiche supportate comprendono:

Supporto per glibc e uclibc e recentemente eglibc;

Generazione per diversi dispositivi target da un'unica base di codice;

Automatizzare tutto ciò che è necessario per compilare e/o eseguire il pacchetto (**compilare le sue dipendenze**);

Creazione di immagini disco flash (jffs2, ext2, gz, UBI, ecc...)

Supporto per vari formati di pacchettizzazione;

Costruzione automatica di tutti gli strumenti di cross-compilazione necessari;


yocto
PROJECT

Yocto Project & OpenEmbedded

**Definizioni**

# Definizioni...

- Recipe

Pron. /'resəpi/ (http://www.oxfordlearnersdictionaries.com/definition/english/recipe)

Definisce dove scaricare e come compilare pacchetti sorgenti ed eventuali patches

*${LAYERDIR}/recipes-*/*/*.bb*

- Layer

Definisce liste di meta-pacchetti raggruppati

*$HOME/yocto/poky/meta-**

- Task

Definisce funzionalità predefinite all'interno di classi

*${LAYERDIR}/classes/*.bbclass*

# Fasi di generazione - Tasks

Per ogni progetto viene generalmente richiesta la stessa sequenza dei seguenti compiti:

Scaricare il codice sorgente, e relativi file di supporto (come initscripts);

Estrarre il codice sorgente e applicare tutte le patch che possono essere necessarie;

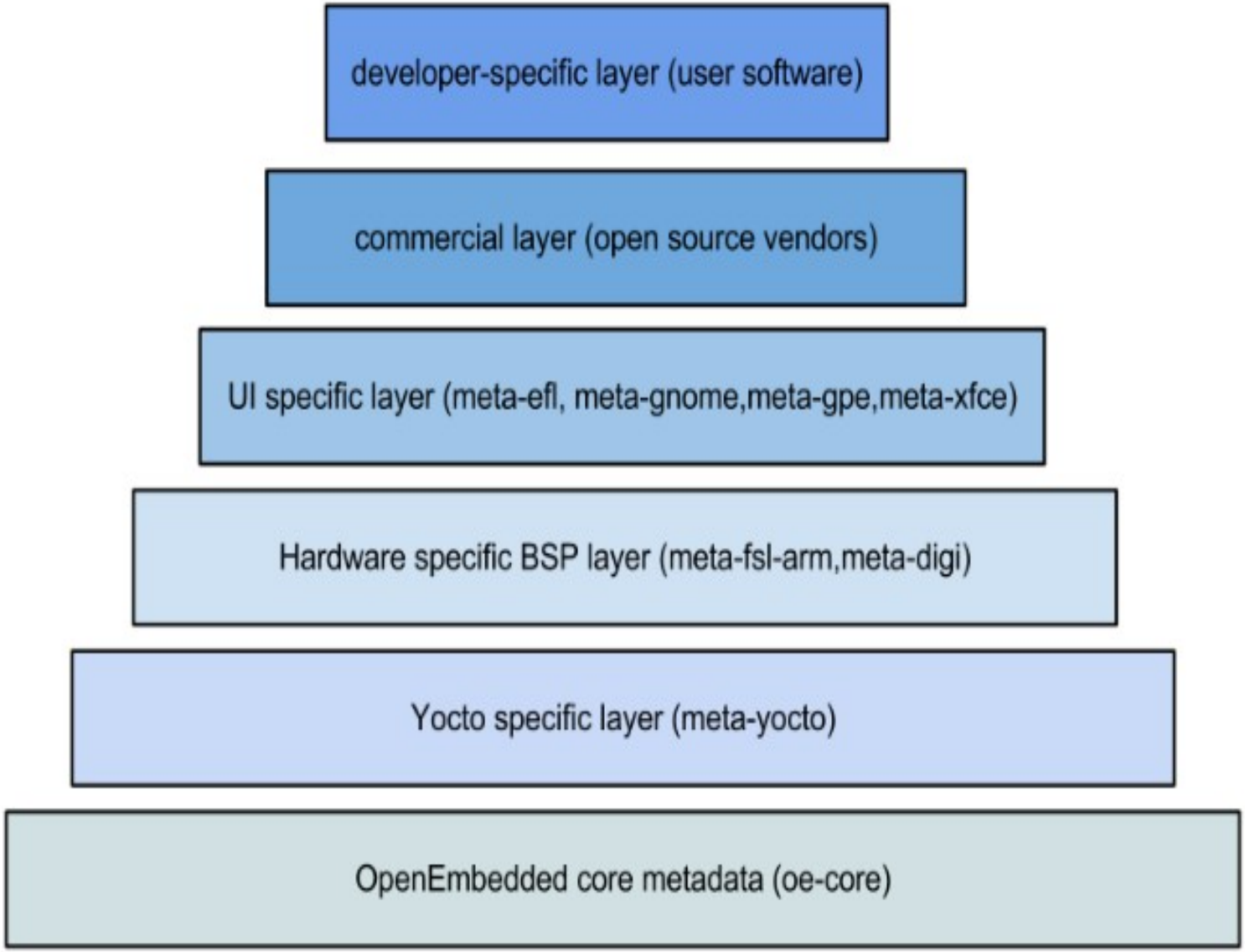
Configurare il software, se necessario (come si fa eseguendo lo script 'configure');

Compilare il tutto;

Pacchettizzare tutti i file in uno dei formati disponibili, come .deb o .rpm o .ipk, pronti per l'installazione.

# Tasks - Build steps

# Layers


developer-specific layer (user software)
commercial layer (open source vendors)
UI specific layer (meta-efl, meta-gnome,meta-gpe,meta-xfce)
Hardware specific BSP layer (meta-fsl-arm,meta-digi)
Yocto specific layer (meta-yocto)
OpenEmbedded core metadata (oe-core)

# Definizioni...

- Image

Definisce liste di meta-pacchetti usati per la generazione dell'immagine finale del sistema

*${LAYERDIR}/recipes-image/*/*.bb*

- Machine

Definisce il BSP per l'architettura hardware supportata

*${LAYERDIR}/conf/machine/*.conf*

- Distro

La distribuzione definisce tipo e versione dei pacchetti

*${LAYERDIR}/conf/distro/*.conf*

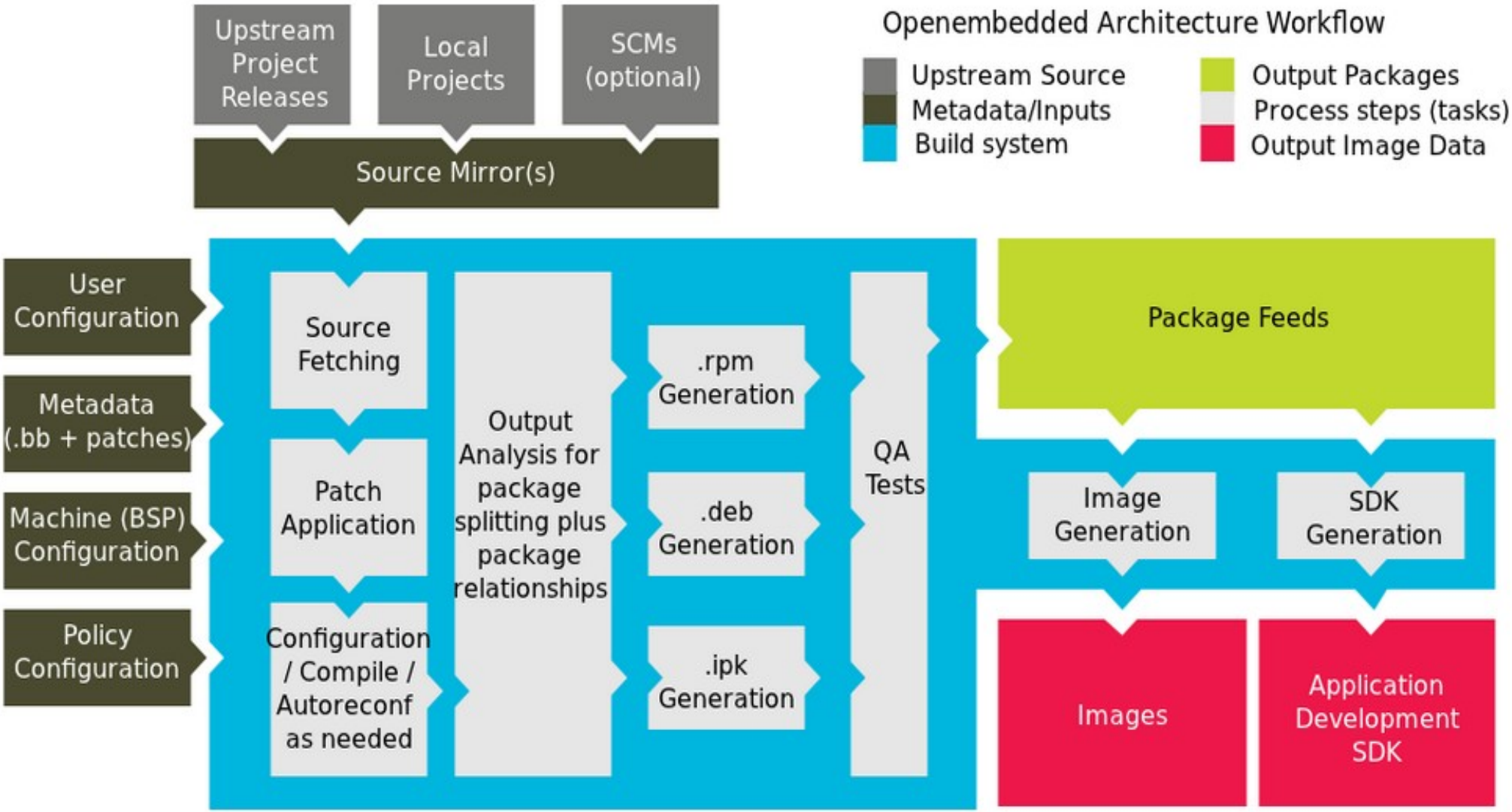
# Definizioni

| MACHINE | DISTRO | IMAGE |
|---|---|---|
| qemu86 | poky | core-image-minimal |
| imx6qsabresd | angstrom | core-image-sato |
| imx8mmevk | poky | core-image-minimal |

Le impostazioni di MACHINE+DISTRO+IMAGE definiscono **come** e **cosa** generare

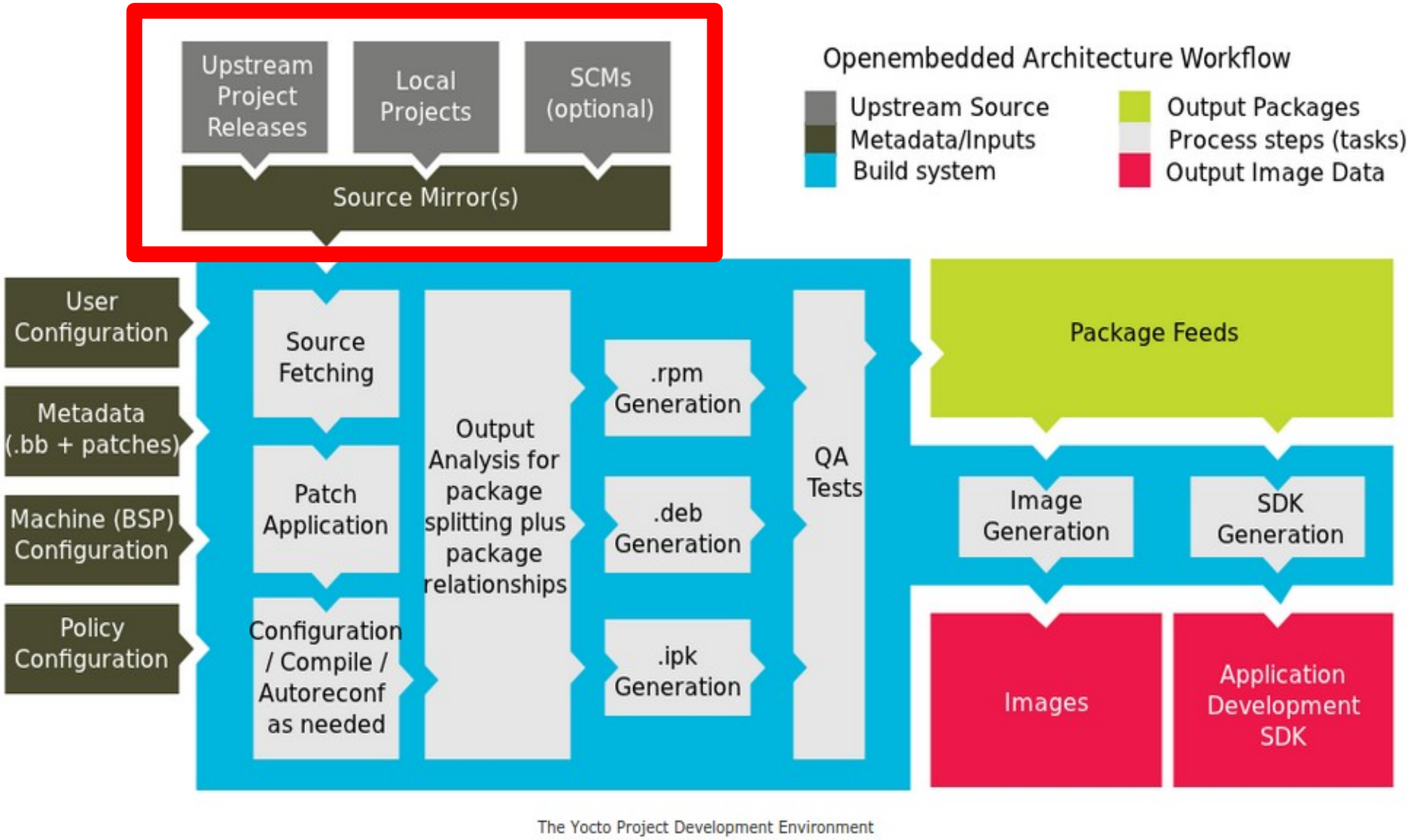

yocto
PROJECT

# Panoramica di Yocto Project



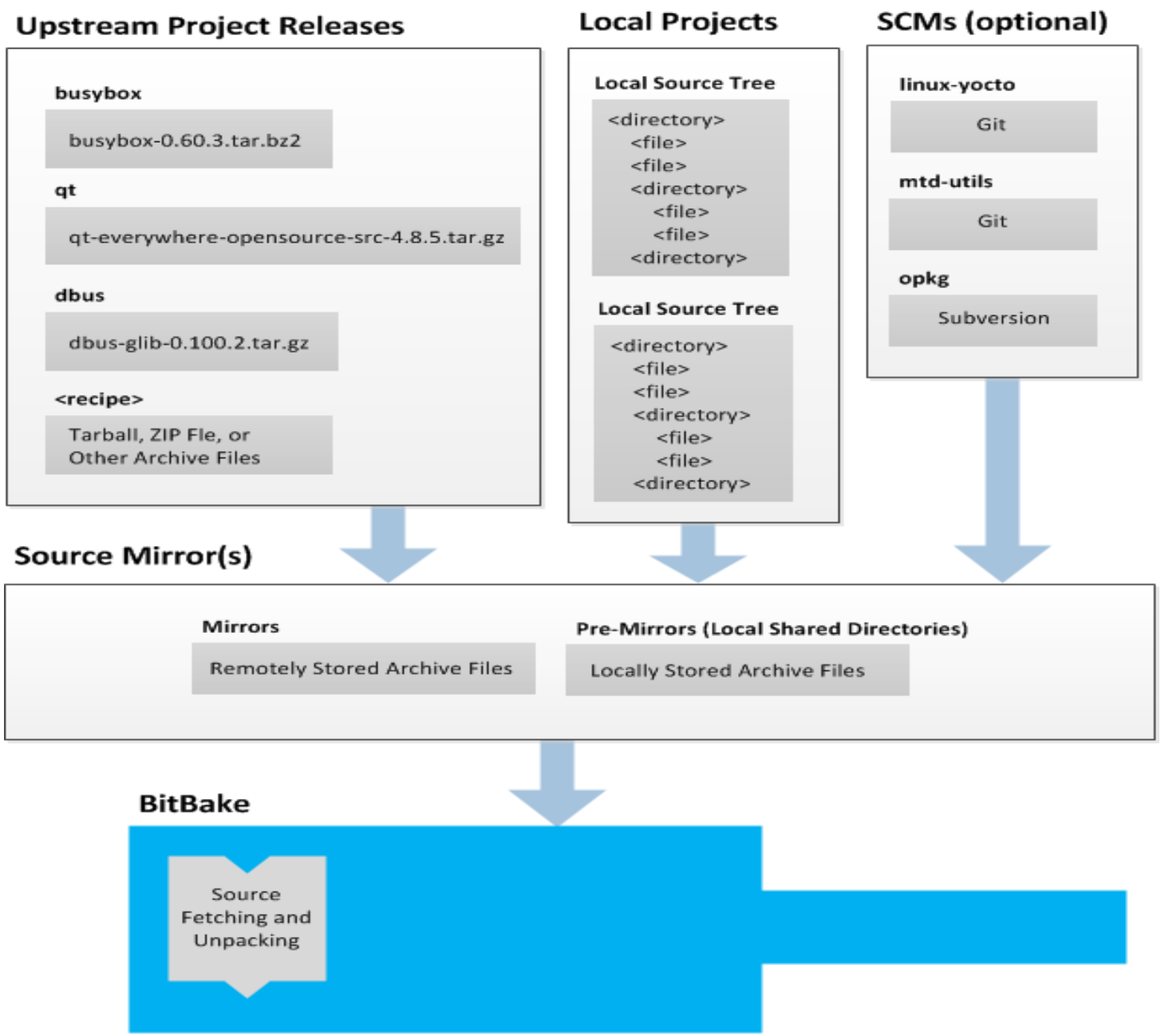

The Yocto Project Development Environment

# Sorgenti

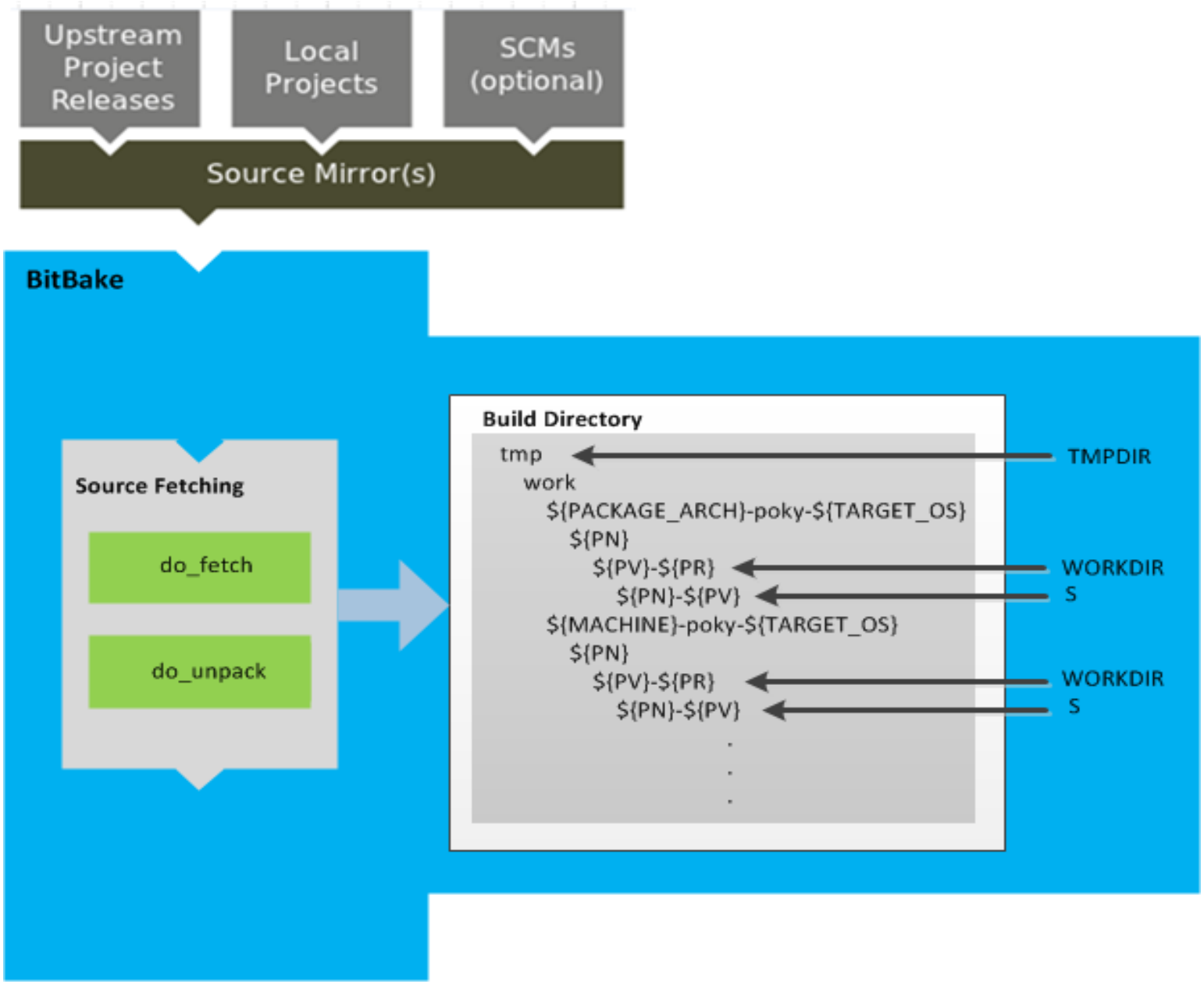



The Yocto Project Development Environment

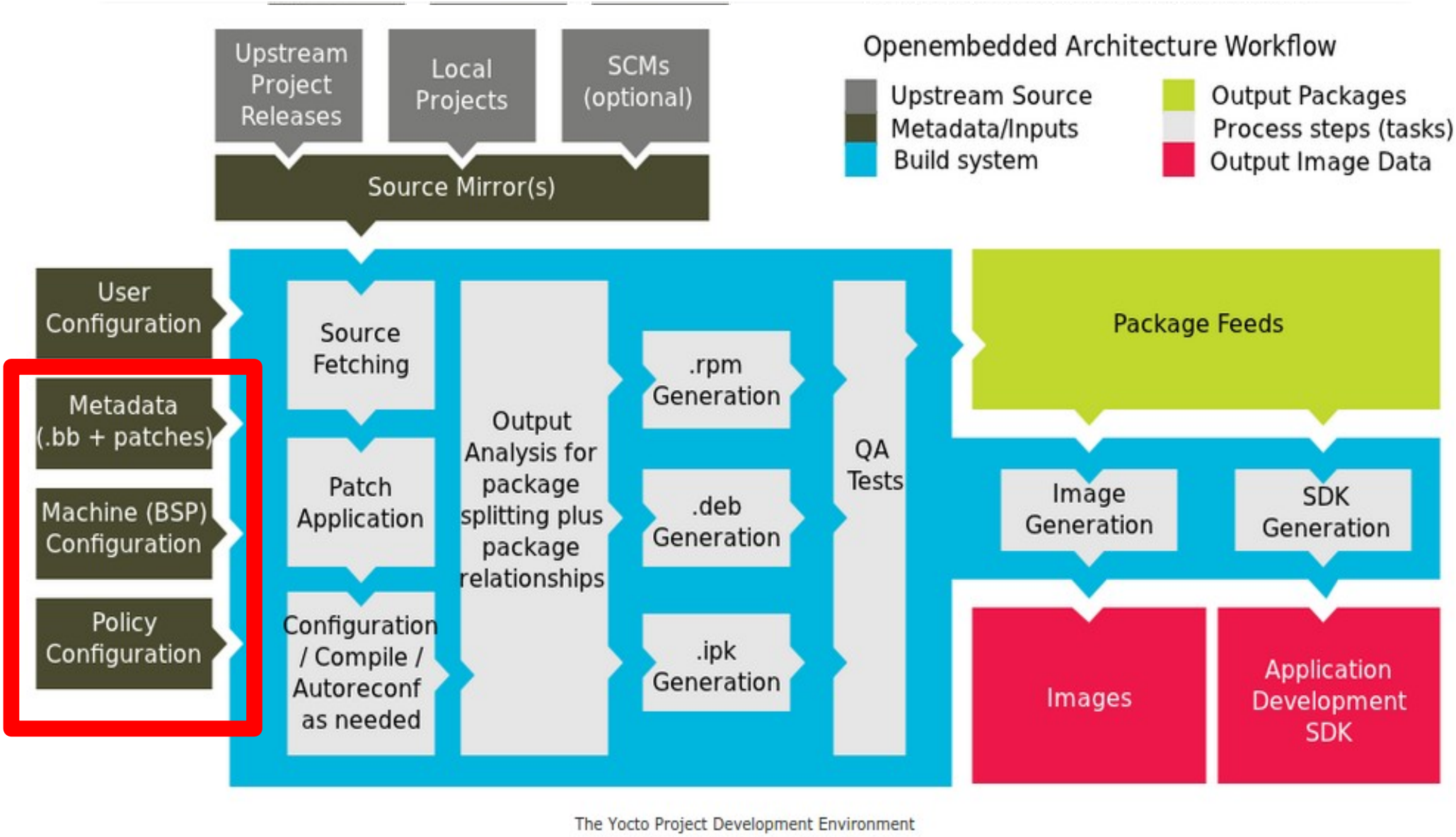
# Sorgenti - Fetch



# Sorgenti - Unpack

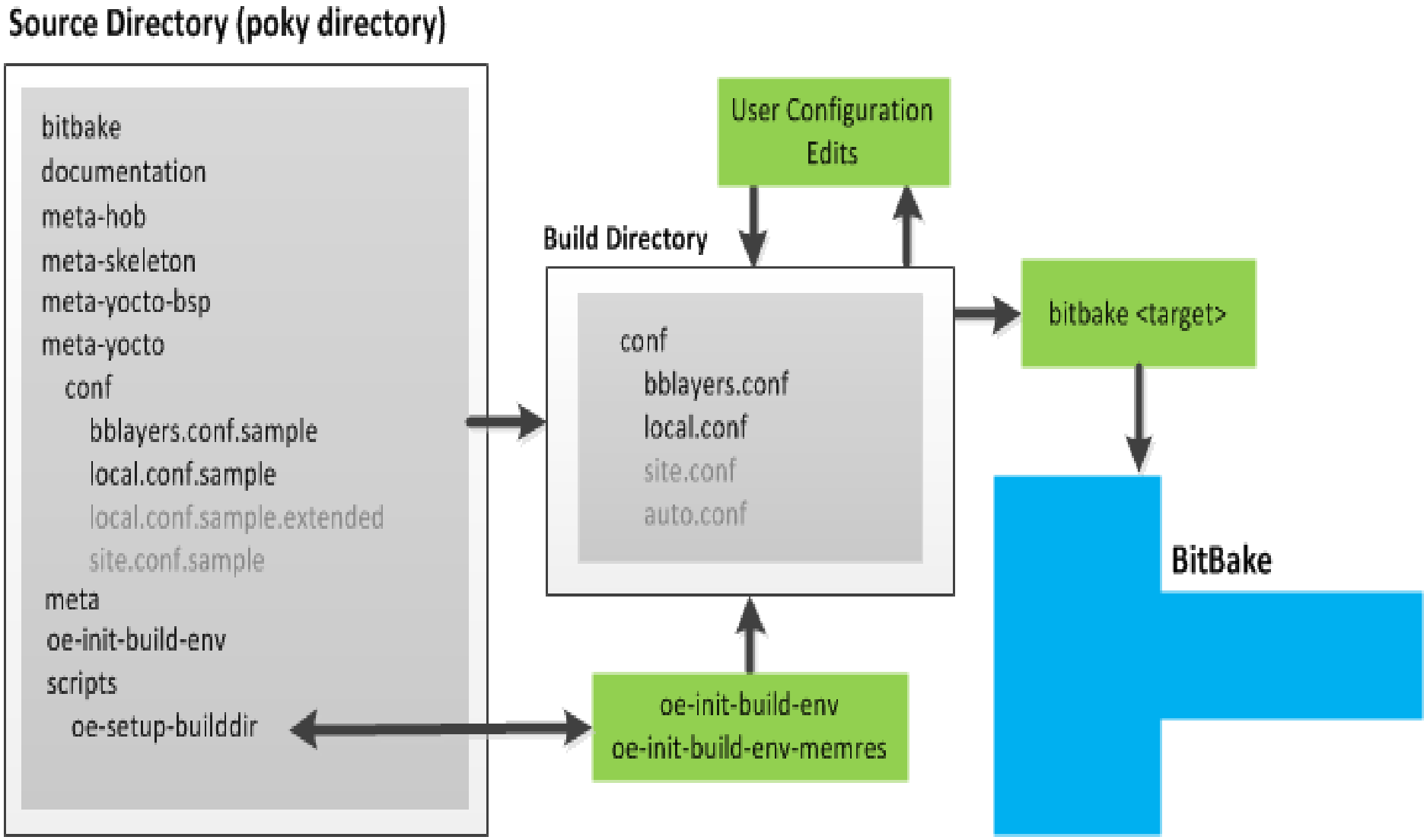


# Configurazione

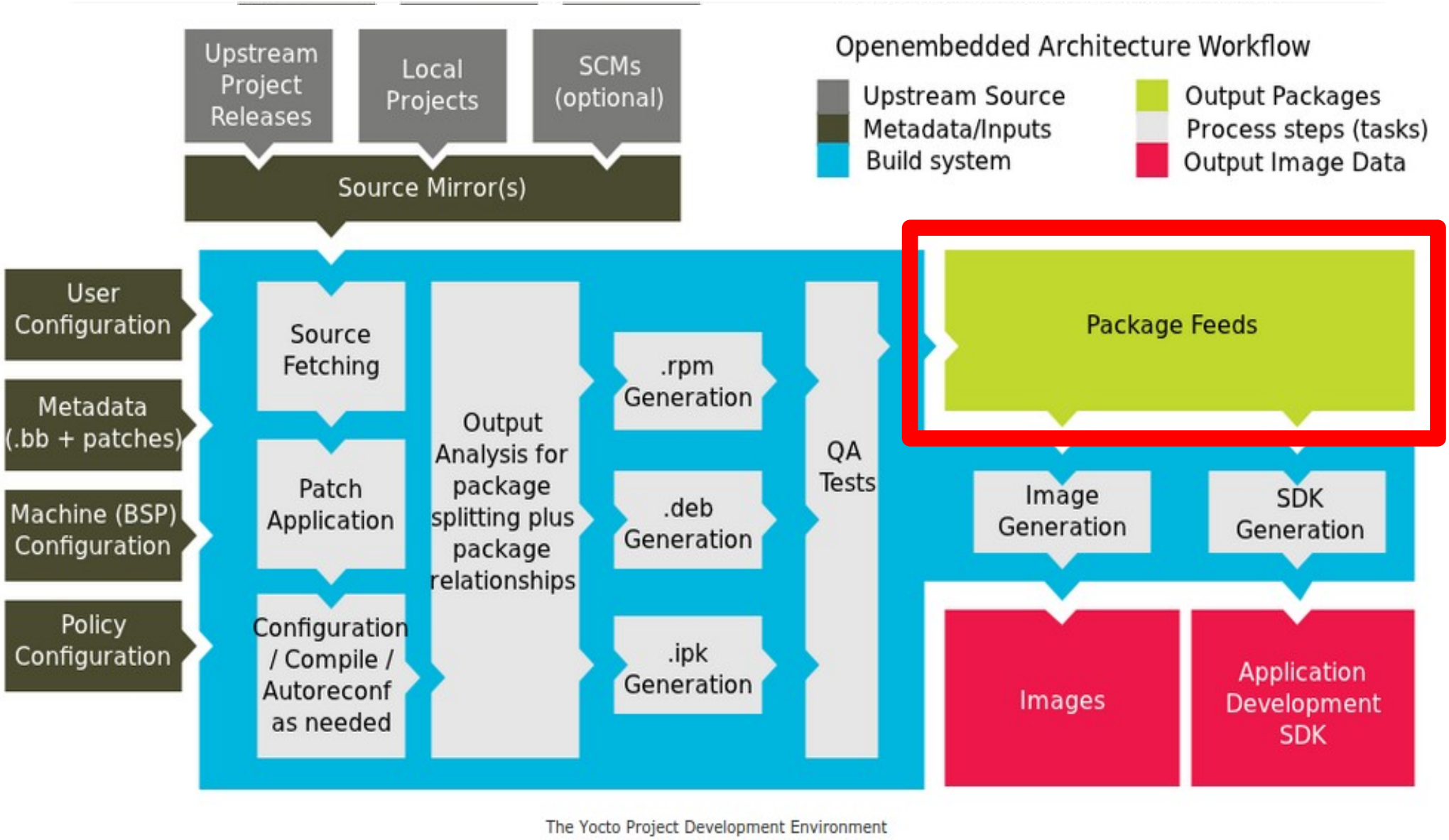



The Yocto Project Development Environment

# Configurazione dell'utente

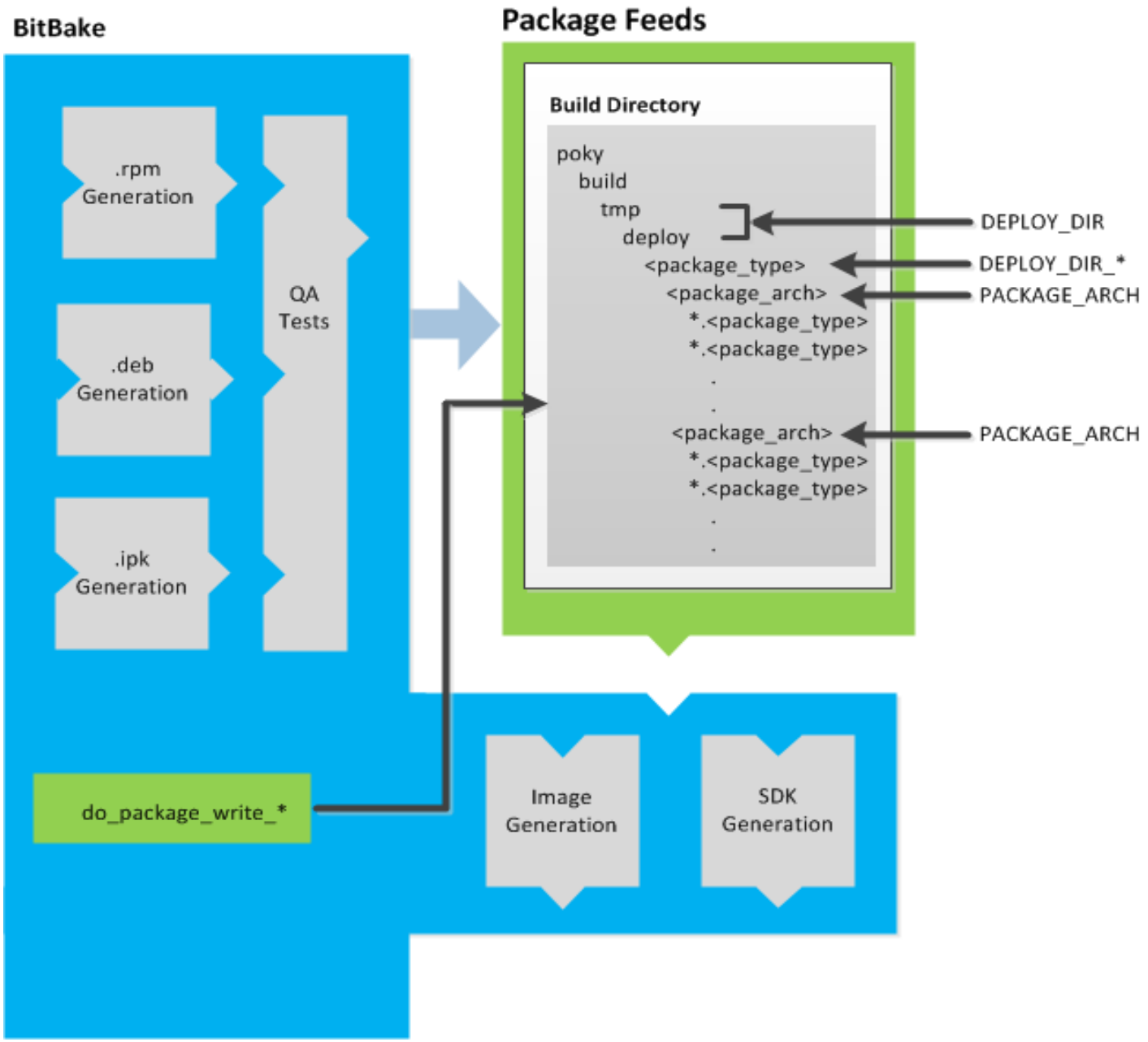


# Package feeds




The Yocto Project Development Environment

# Package feeds



# Yocto Project reference manual

Per maggiori dettagli

Yocto Project reference manual

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


yocto
PROJECT

# Installazione

Procedura di installazione di Yocto Project

https://www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.html

https://wiki.koansoftware.com/index.php/Yocto_Project_my_own_quick_start

# Fasi di utilizzo

Le fasi di utilizzo del sistema di build di Yocto :

- Installazione (una sola volta)
  - Git clone dai repository
- Configurazione (una sola volta)
  - Impostazione di MACHINE+DISTRO
- Compilazione (ogni volta che è necessario)
  - Bitbake <NOME_IMAGE>

# Installazione

Installazione dei pacchetti necessari dal repository della distribuzione

- Ubuntu e Debian

```
$ sudo apt-get install gawk wget git-core diffstat unzip \
texinfo gcc-multilib build-essential chrpath socat \
libsdl1.2-dev xterm
```

- Fedora

```
$ sudo dnf install gawk make wget tar bzip2 gzip python \
unzip perl patch diffutils diffstat git cpp gcc gcc-c++ \
glibc-devel texinfo chrpath ccache perl-Data-Dumper \
perl-Text-ParseWords perl-Thread-Queue perl-bignum socat \
findutils which SDL-devel xterm
```

# Installazione

- Scaricamento sorgenti di Yocto dal repository git

```
$ mkdir $HOME/yocto
$ cd $HOME/yocto
```

```
$ git clone git://git.yoctoproject.org/poky -b zeus
```

```
Cloning into 'poky'...
remote: Counting objects: 226790, done.
remote: Compressing objects: 100% (57465/57465), done.
remote: Total 226790 (delta 165212), reused 225887 (delta 164327)
Receiving objects: 100% (226790/226790), 100.98 MiB | 263 KiB/s, done.
Resolving deltas: 100% (165212/165212), done.
```

# Installazione

- Albero dei sorgenti scaricati *(lista ridotta)*

```
$ .
└── poky
    ├── bitbake
    ├── documentation
    ├── LICENSE
    ├── meta
    ├── meta-poky
    ├── meta-selftest
    ├── meta-skeleton
    ├── meta-yocto-bsp
    ├── oe-init-build-env
    ├── README
    ├── README.hardware
    └── scripts
```

- Layer aggiuntivi (opzionali)


```
$ cd $HOME/yocto/poky
$ git clone git://git.openembedded.org/meta-openembedded -b zeus
```


```
$ cd $HOME/yocto/poky
$ git clone git://git.yoctoproject.org/meta-freescale -b zeus
$ git clone https://github.com/Freescale/meta-freescale-3rdparty -b zeus
```

```
$ cd $HOME/yocto/poky
$ git clone https://github.com/meta-qt5/meta-qt5.git -b zeus
```

# Installazione

- Albero con i layer aggiuntivi opzionali

```
$ .
└── poky
    ├── bitbake
    ├── documentation
    ├── LICENSE
    ├── meta
    ├── meta-poky
    ├── meta-selftest
    ├── meta-skeleton
    ├── meta-yocto-bsp
    ├── meta-fsl-arm
    ├── meta-fsl-arm-extra
    ├── meta-openembedded/
    ├── meta-qt5
    ├── oe-init-build-env
    ├── README
    ├── README.hardware
    └── scripts
```


http://freescale.github.io/
FSL Community BSP
Documentation
Contributing
Download

# Installazione NXP

- NXP usa il sistema basato su ‘repo’

```
$ mkdir fsl-community-bsp

$ cd fsl-community-bsp

$ repo init -u \
  https://github.com/Freescale/fsl-community-bsp-platform -b rocko

$ repo sync
```

# Installazione NXP per iMX8 AI

- Lo stesso sistema basato su ‘repo’ per **eIQ Machine Learning**
- Link : https://www.nxp.com/docs/en/nxp/user-guides/UM11226.pdf

```
$ mkdir fsl-arm-yocto-bsp

$ cd fsl-arm-yocto-bsp


$ repo init -u \
  https://source.codeaurora.org/external/imx/imx-manifest \
     -b imx-linux-sumo \
     -m imx-4.14.78-1.0.0_machinelearning.xml


$ repo sync
```

# Installazione

- Albero con per **eIQ Machine Learning**

```
├── setup-environment -> sources/base/setup-environment
└── sources
    ├── base
    ├── meta-browser
    ├── meta-freescale
    ├── meta-freescale-3rdparty
    ├── meta-freescale-distro
    ├── meta-fsl-bsp-release
    ├── meta-imx-machinelearning
    ├── meta-openembedded
    ├── meta-qt5
    └── poky
        ├── bitbake
        ├── documentation
        ├── meta
        ├── meta-poky
        ├── meta-selftest
        ├── meta-skeleton
        └── meta-yocto-bsp
```

# Installazione

## Prima inizializzazione

- Questa procedura inizializza il sistema
- Deve essere eseguita ogni volta che si avvia una console

```
$ EULA=1 MACHINE=imx8qmmek DISTRO=fsl-imx-xwayland \
        source ./fsl-setup-release.sh -b build
```

# Configurazione

E' necessario configurare due files

- build/conf/bblayers.conf
- build/conf/local.conf

# Configurazione

- Editare il file : build/conf/bblayers.conf

```
# i.MX Yocto Project Release layers

BBLAYERS += " ${BSPDIR}/sources/meta-fsl-bsp-release/imx/meta-bsp "
BBLAYERS += " ${BSPDIR}/sources/meta-fsl-bsp-release/imx/meta-sdk "
BBLAYERS += " ${BSPDIR}/sources/meta-browser "
BBLAYERS += " ${BSPDIR}/sources/meta-openembedded/meta-gnome "
BBLAYERS += " ${BSPDIR}/sources/meta-openembedded/meta-networking "
BBLAYERS += " ${BSPDIR}/sources/meta-openembedded/meta-python "
BBLAYERS += " ${BSPDIR}/sources/meta-openembedded/meta-filesystems "
BBLAYERS += " ${BSPDIR}/sources/meta-qt5 "
BBLAYERS += " ${BSPDIR}/sources/meta-imx-machinelearning "
```

# Configurazione

- Editare il file : build/conf/local.conf
- Cambiare I seguenti valori

```
PACKAGE_CLASSES ?= "package_ipk"
MACHINE ?= "imx8mmevk"
```


EBV Elektronik
I An Avnet Company I

# Compilazione

- Verificare sempre:
  - 1. Impostazione dell'ambiente di sviluppo

```
$ cd $HOME/yocto/poky
```

  - 2. Impostazione della directory di build

```
$ EULA=1 MACHINE=imx8qmmek DISTRO=fsl-imx-xwayland \
  source ./fsl-setup-release.sh -b build
```

# Compilazione

- A questo punto possiamo creare la nostra prima immagine
  - Saremo nella directory di **build**

```
$HOME/yocto/poky/build
```

  - E potremo lanciare **bitbake**

```
$ bitbake fsl-image-qt5
```

# Compilazione

```
$ bitbake fsl-image-qt5


Parsing recipes: 100% |#########################################################| Time: 00:00:56
Parsing of 899 .bb files complete (0 cached, 899 parsed). 1330 targets, 38 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION        = "1.28.0"
BUILD_SYS         = "i686-linux"
NATIVELSBSTRING   = "Ubuntu-18.04"
TARGET_SYS        = "i586-poky-linux"
MACHINE           = "imx8mmevk"
DISTRO            = "poky"
DISTRO_VERSION    = "3.0.0"
TUNE_FEATURES     = "m32 i586"
TARGET_FPU        = ""

meta
meta-yocto
meta-yocto-bsp    = "zeus:60ba9abd43f7584178de52b623c603a5d4fce05c"
…
…

NOTE: Preparing RunQueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
Currently 4 running tasks (14 of 2050):
0: quilt-native-0.64-r0 do_compile (pid 3651)
1: autoconf-native-2.69-r11 do_fetch (pid 3653)
2: automake-native-1.15-r0 do_fetch (pid 3654)
3: libtool-native-2.4.6-r0 do_fetch (pid 3680)
```

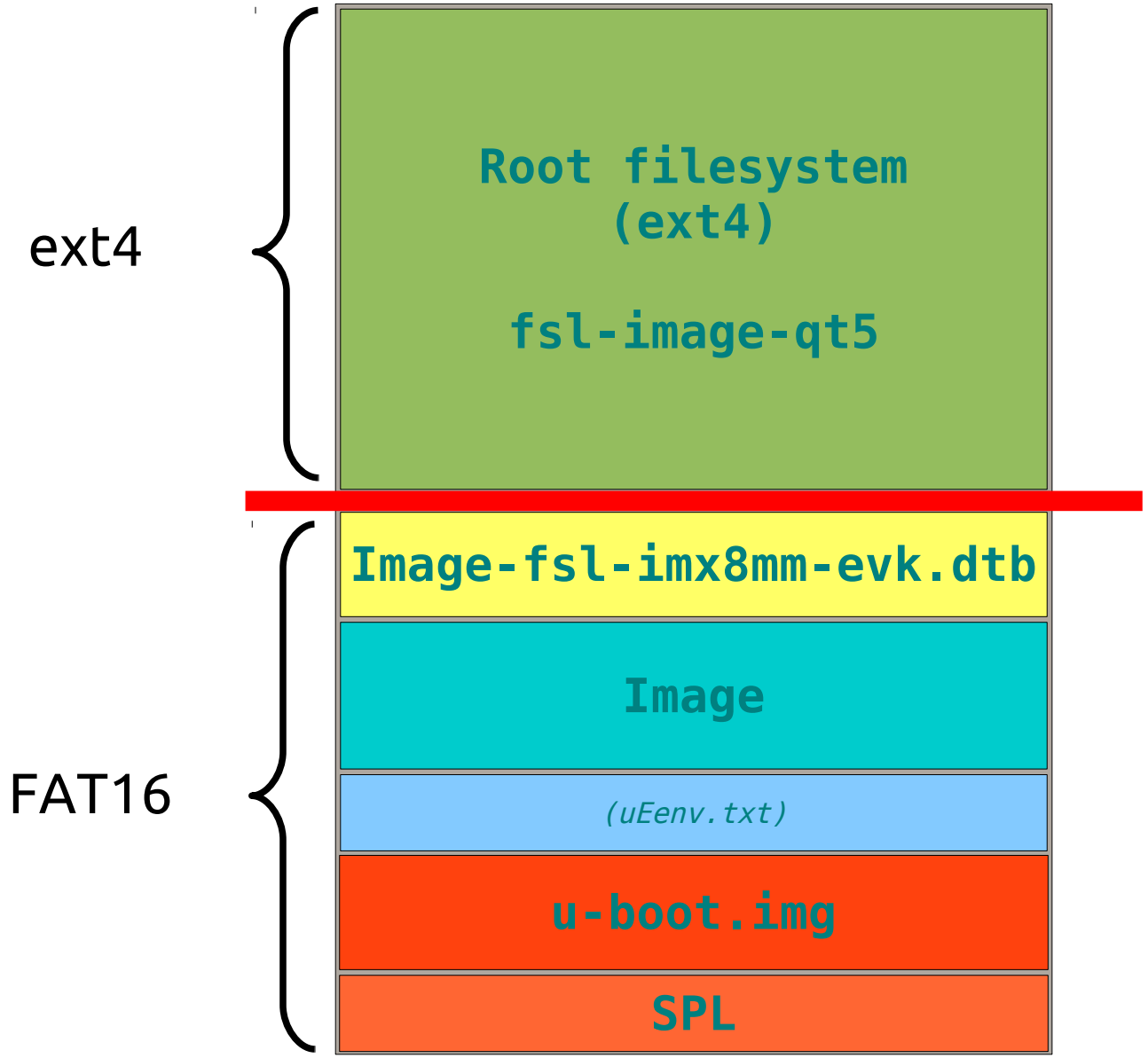
# Risultato della compilazione

```
$ cd $HOME/yocto/poky/build/tmp/deploy/images/imx8mmevk
```

```
fsl-image-qt5-imx8mmevk.ext4
fsl-image-qt5-imx8mmevk.manifest
fsl-image-qt5-imx8mmevk.sdcard.bz2
fsl-image-qt5-imx8mmevk.tar.bz2
fsl-image-qt5-imx8mmevk.testdata.json
Image
Image-fsl-imx8mm-evk.dtb
u-boot.bin
```

*lista dei files ridotta*

# Contenuto della microSD



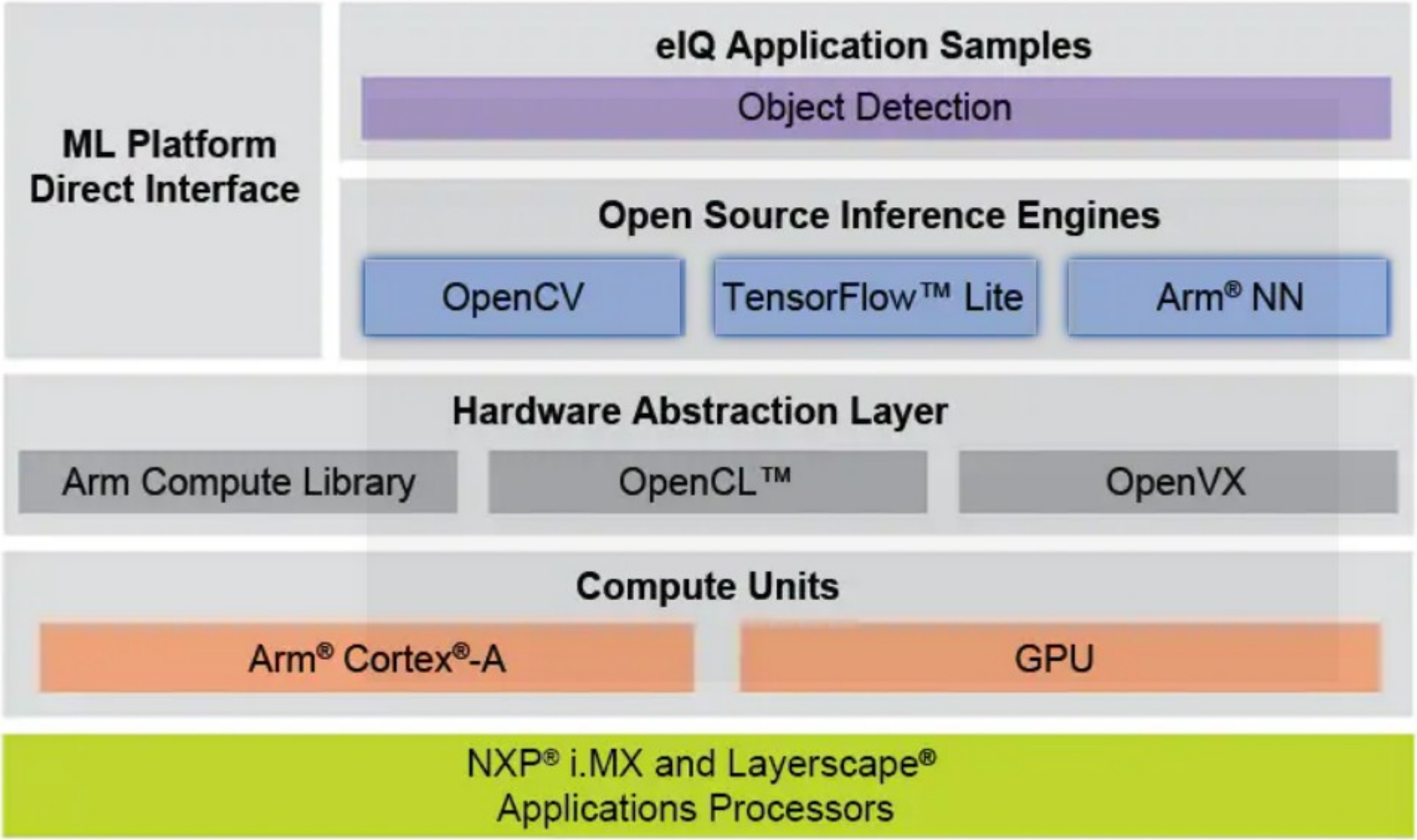

yocto
PROJECT

# NXP eIQ software introduction

- The NXP eIQ software contains these main Yocto recipes:
  - OpenCV 4.0.1
  - Arm Compute Library 19.02
  - Arm NN 19.02
  - ONNX runtime 0.3.0
  - TensorFlow 1.12
  - TensorFlow Lite 1.12

https://www.nxp.com/design/software/development-software/eiq-ml-development-environment:EIQ

# NXP eIQ software




yocto
PROJECT

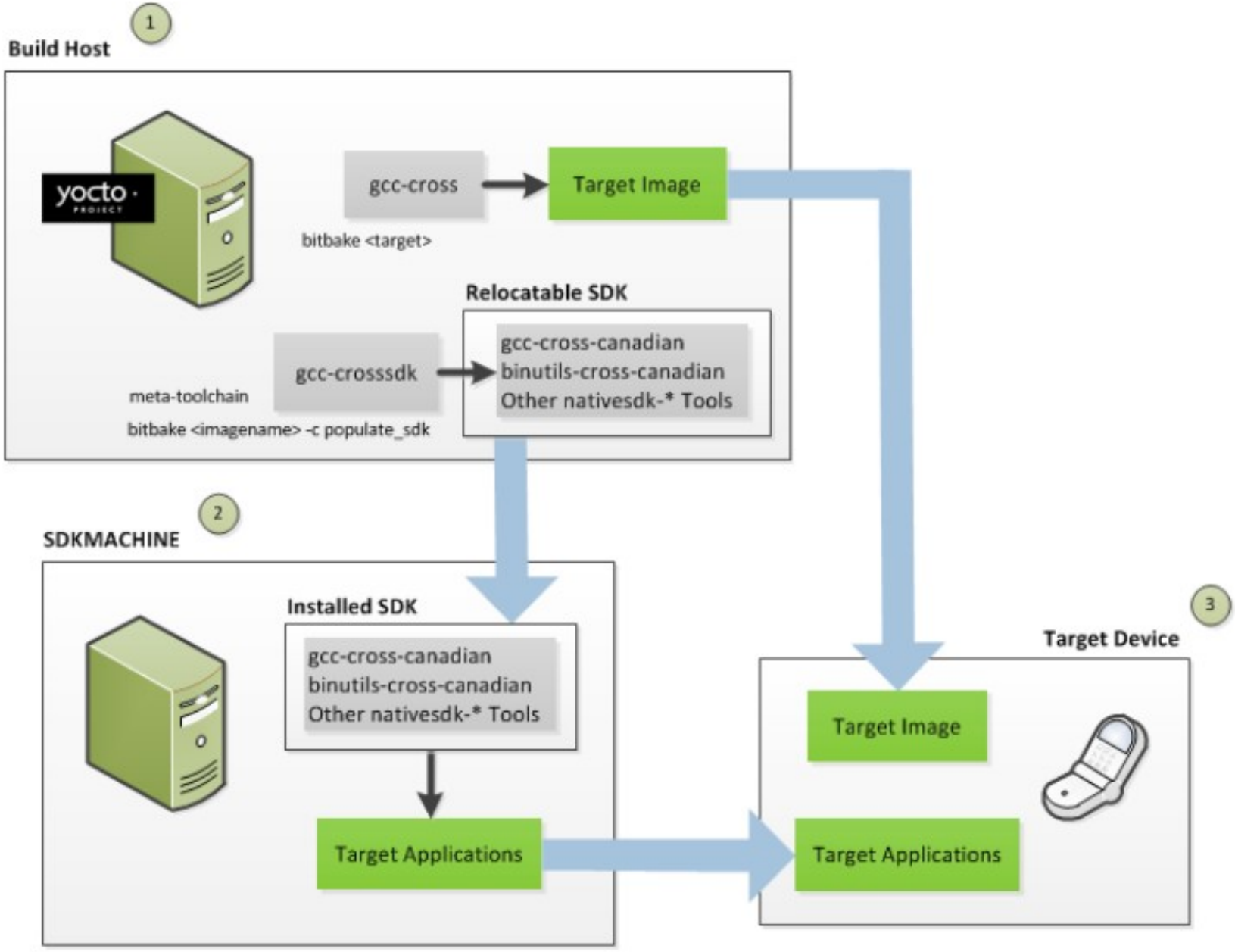
# Creazione del cross-compilatore

- Yocto ci permette di creare un cross-compilatore

```
$ bitbake meta-toolchain
```

```
$ bitbake fsl-image-qt5 -c populate_sdk
```

# Cross-Development Toolchain



# Installazione del cross-compilatore

- Installazione del cross-compilatore (redistribuzione)

```
$ cd $HOME/yocto/poky/build/tmp/deploy/sdk
$ ./fsl-imx-xwayland-glibc-x86_64-fsl-image-qt5-aarch64-toolchain-4.14-sumo.sh
```

- Verrà installato in /opt/poky/3.0



**KOAN** - Kernel, drivers and embedded Linux development, consulting, training and support **https//KoanSoftware.com**

# Installazione del cross-compilatore

- Il cross-compilatore verrà installato in

```
/opt/poky/3.0/
```

- E potrà essere usato dopo aver impostato

```
$ source /opt/poky/3.0/environment-setup-aarch64-poky-linux
```


yocto
PROJECT

# Domande?

# Rights to copy

© Copyright 2016-2020, Marco Cavallini - KOAN sas
m.cavallini <AT> koansoftware.com

Corrections, suggestions,
contributions and translations are welcome!

ftp://ftp.koansoftware.com/public/talks/Roadshow-EBV-2020/Roadshow-EBV-Koan-Yocto-2020.pdf

**Attribution – ShareAlike 3.0**

**You are free**

to copy, distribute, display, and perform the work
to make derivative works
to make commercial use of the work

creative commons
COMMONS DEED

**Under the following conditions**

**Attribution**. You must give the original author credit.

**Share Alike**. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

For any reuse or distribution, you must make clear to others the license terms of this work.
Any of these conditions can be waived if you get permission from the copyright holder.
**Your fair use and other rights are in no way affected by the above.**
License text: http://creativecommons.org/licenses/by-sa/3.0/legalcode

# Lavora con noi

# KOAN services

**Embedded Linux Training**

Unix and GNU/Linux basics
Linux kernel and drivers development
Real-time Linux
Yocto Project
Development and profiling tools
Lightweight tools for embedded systems
Root filesystem creation
Audio and multimedia
System optimization

**Custom Development**

System integration
BSP creation fir new boards
System optimization
Linux kernel drivers
Application and interface development

**Consulting**

Help in decision making
System architecture
Identification of suitable technologies
Managing licensing requirements
System design and performance review

**Technical Support**

Development tool and application support
Issue investigation and solution follow-up with mainstream developers
Help getting started

Follow us on LinkedIn

https://koansoftware.com

yocto PROJECT PARTICIPANT