yocto PROJECT

un generatore automatico di distribuzioni linux embedded

Marco Cavallini

Agenda

- Linux embedded
  - Sistemi di generazione automatica
- Yocto Project
  - Storia
    - OpenEmbedded
  - Panoramica
    - Recipes, Layers, Classes, etc...
  - Configurazione
  - Installazione
    - git clone
  - Utilizzo
    - bitbake
  - Tools
    - HOB, Toaster


yocto
PROJECT

Requisiti Linux Embedded

I requisiti indispensabili per un sistema linux embedded sono:

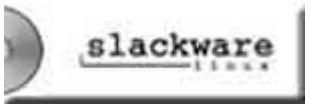
- Dimensione contenuta
  - Busybox, etc...
- Riproducibilità
  - Automatic build system
- Affidabilità
  - Cross-compilation toolchain

Creazione di una distribuzione

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

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

- Downscaling (Debian, Fedora, Slack)



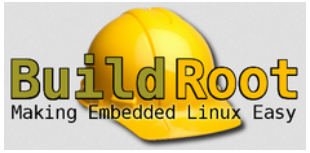
- Distro ARM preesistenti (Debian, Fedora)

- 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

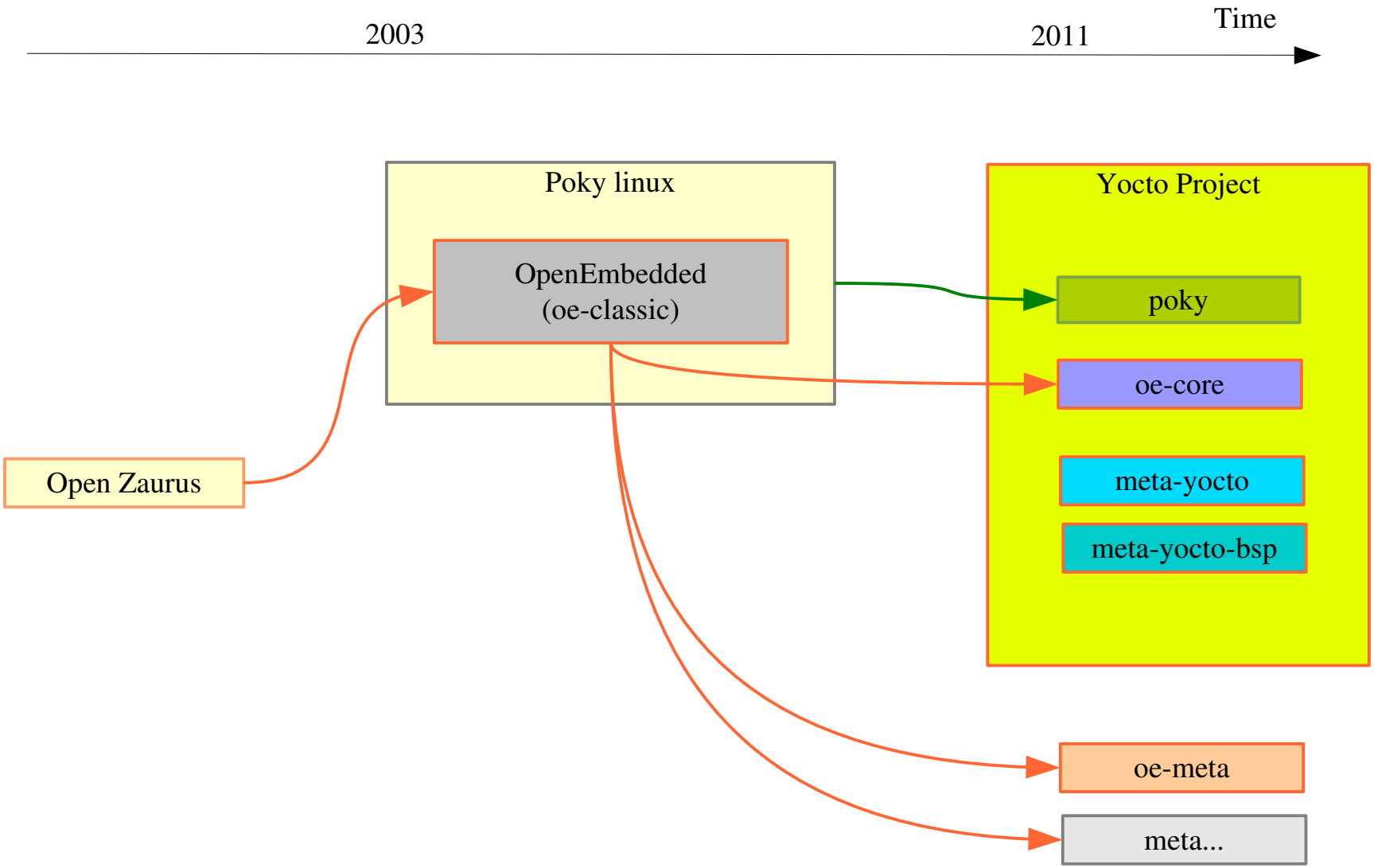
OpenEmbedded: storia

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.

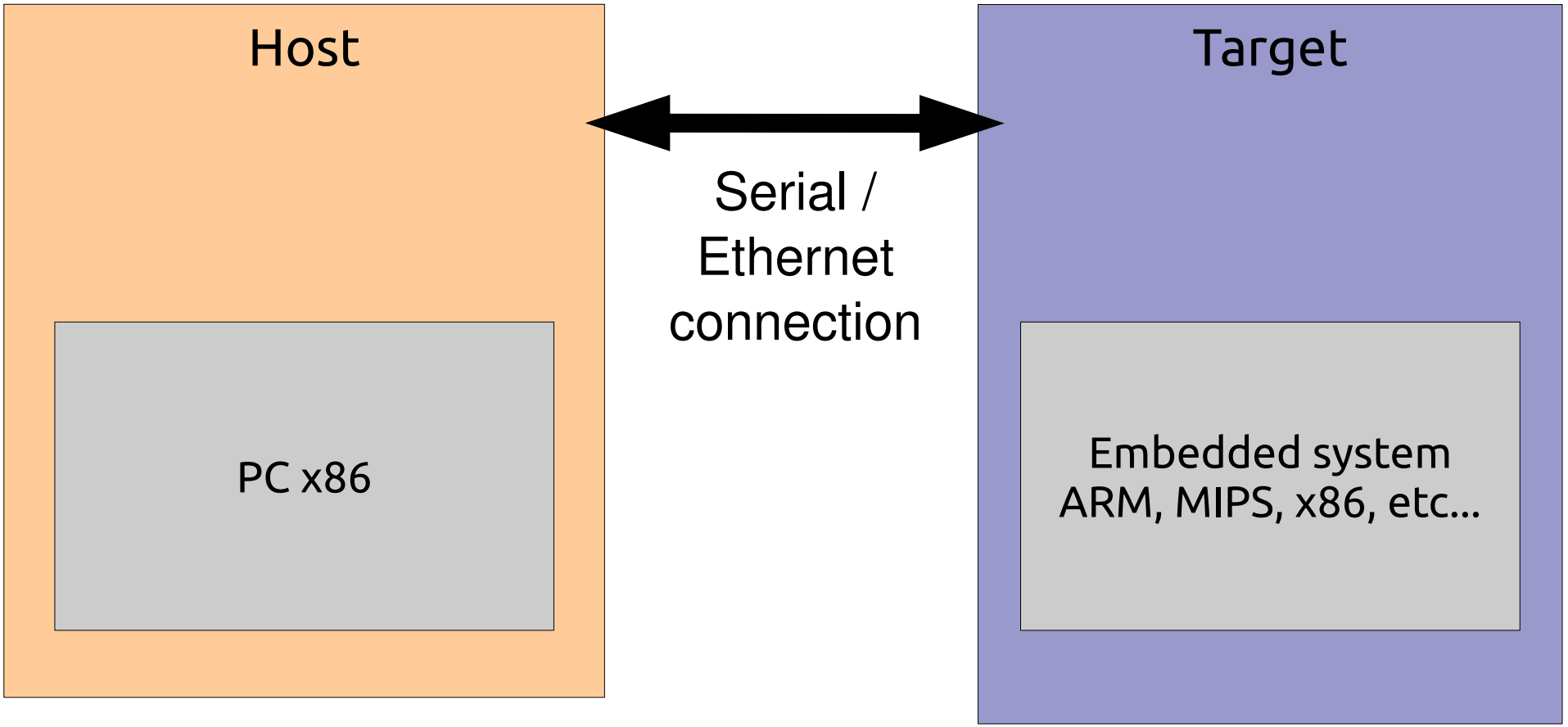
OpenEmbedded, Poky, Yocto




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 elibc;

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);

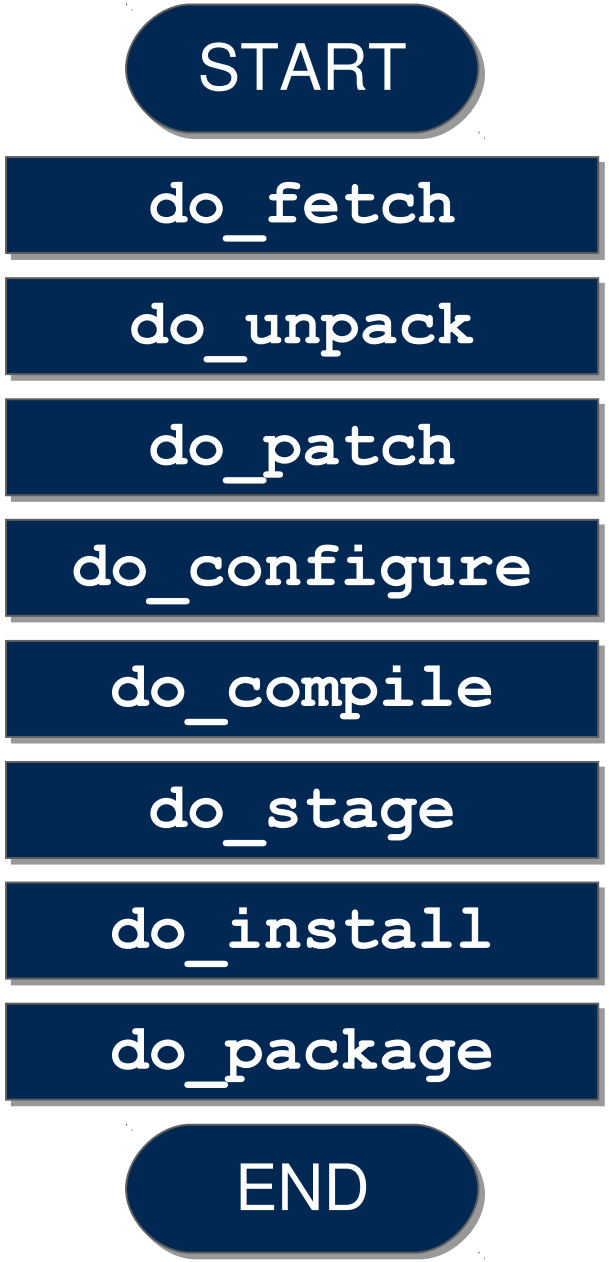
Estrarre il codice sorgente e applicare tutte le patch che possono essere necessarie;

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

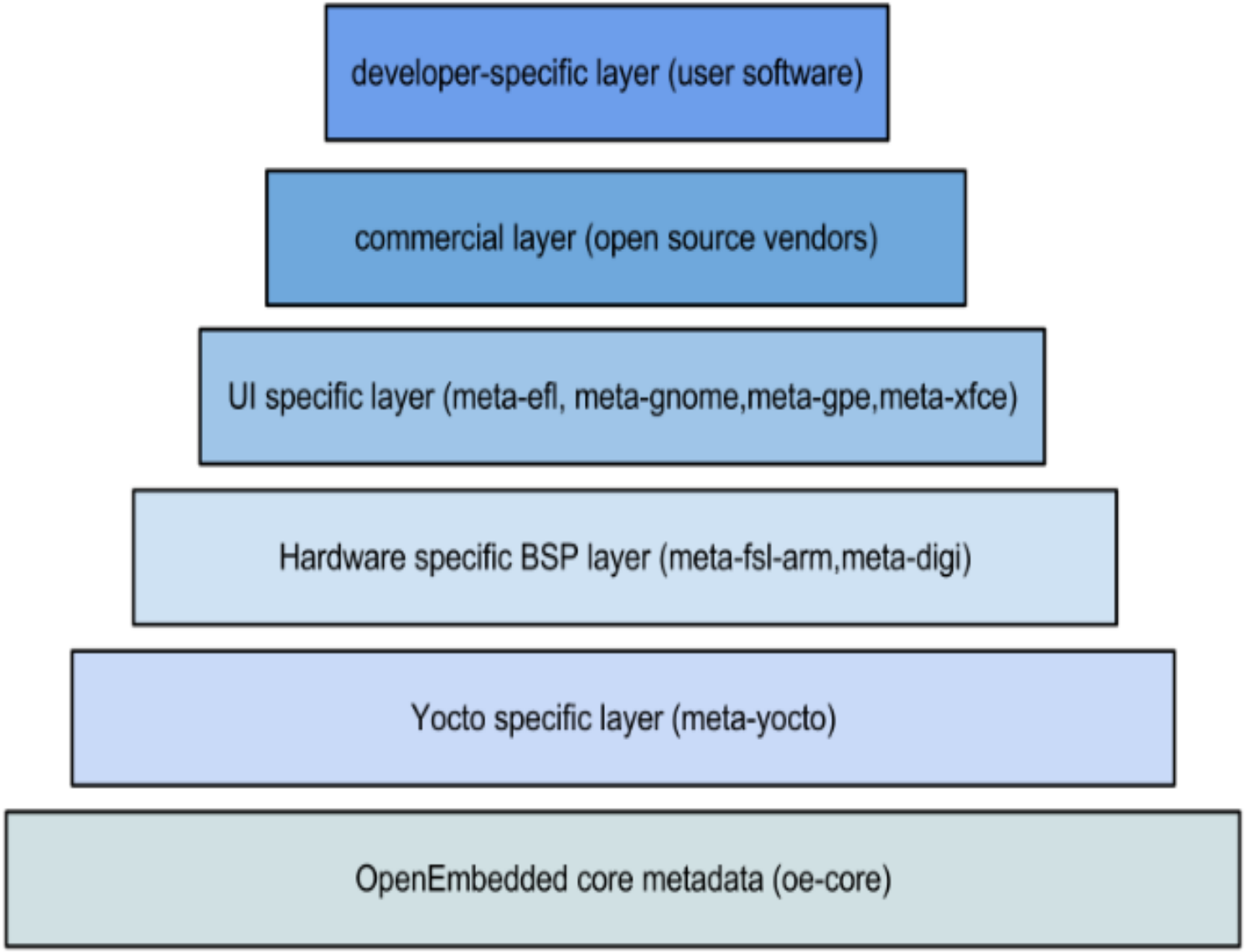
Compilare il tutto;

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*

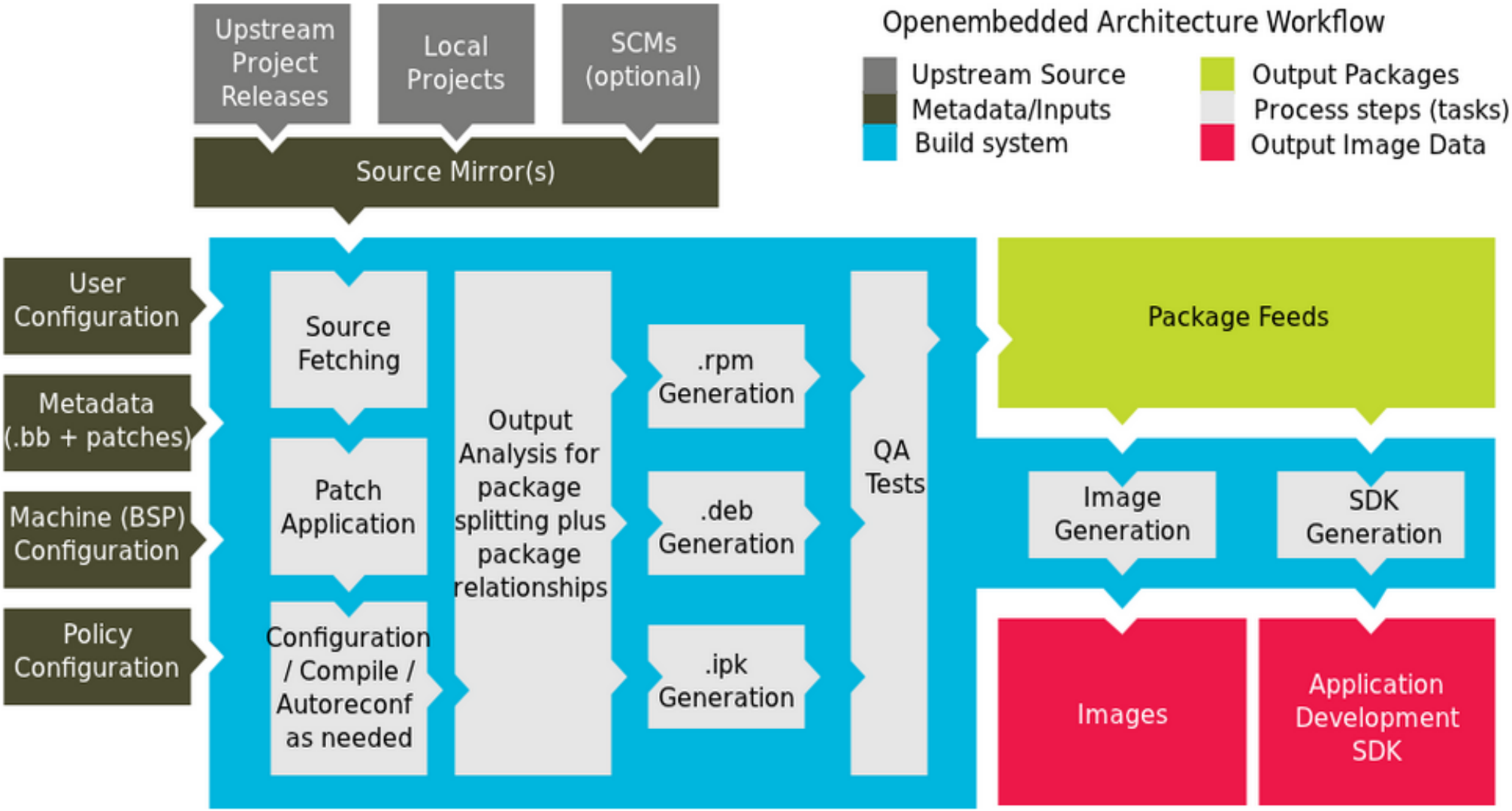
Definizioni.

MACHINE	DISTRO	IMAGE
qemu86	poky	core-image-minimal
at91sama5d3	angstrom	core-image-sato
BeagleBone	poky	core-image-minimal

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

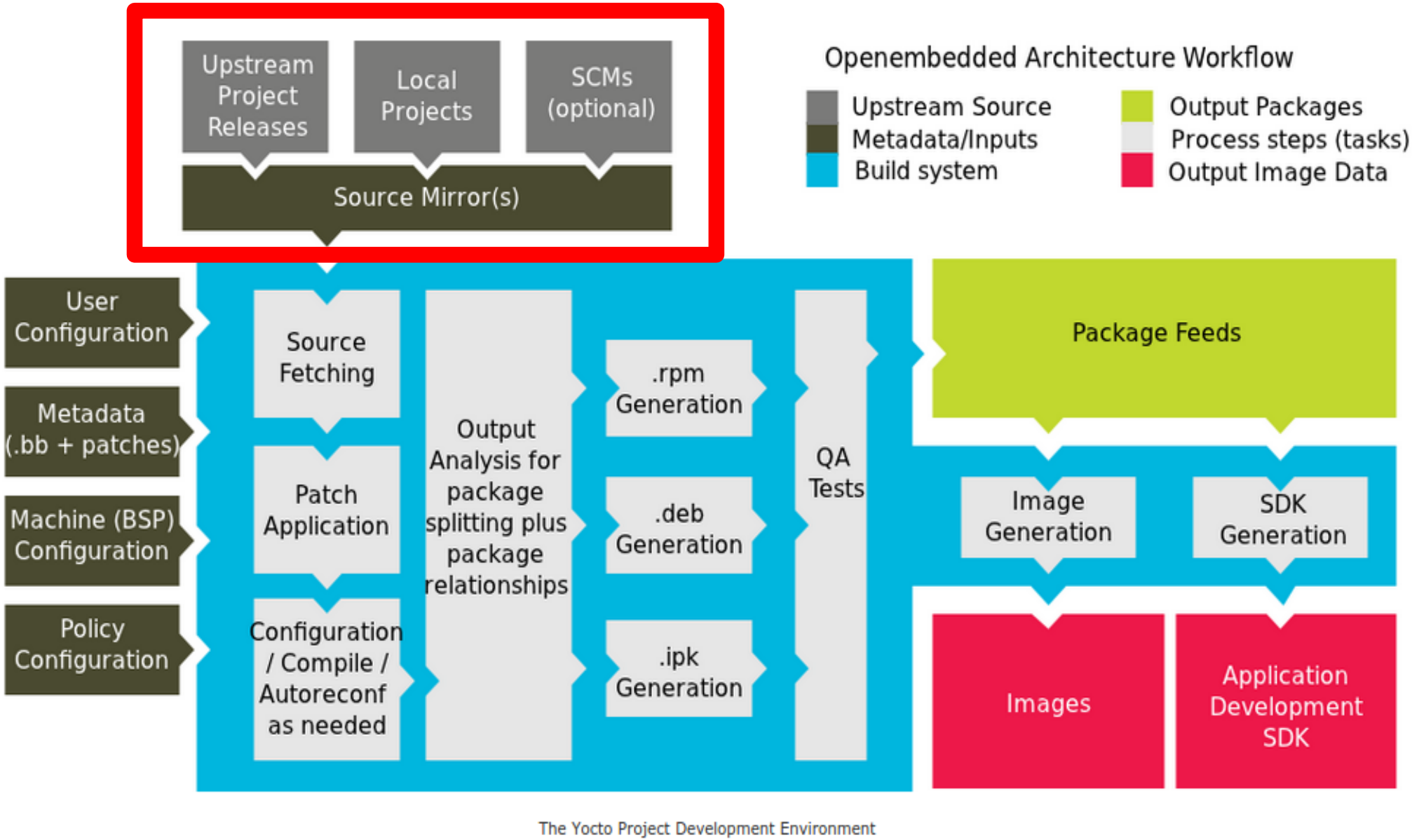

yocto
PROJECT

Panoramica di Yocto Project



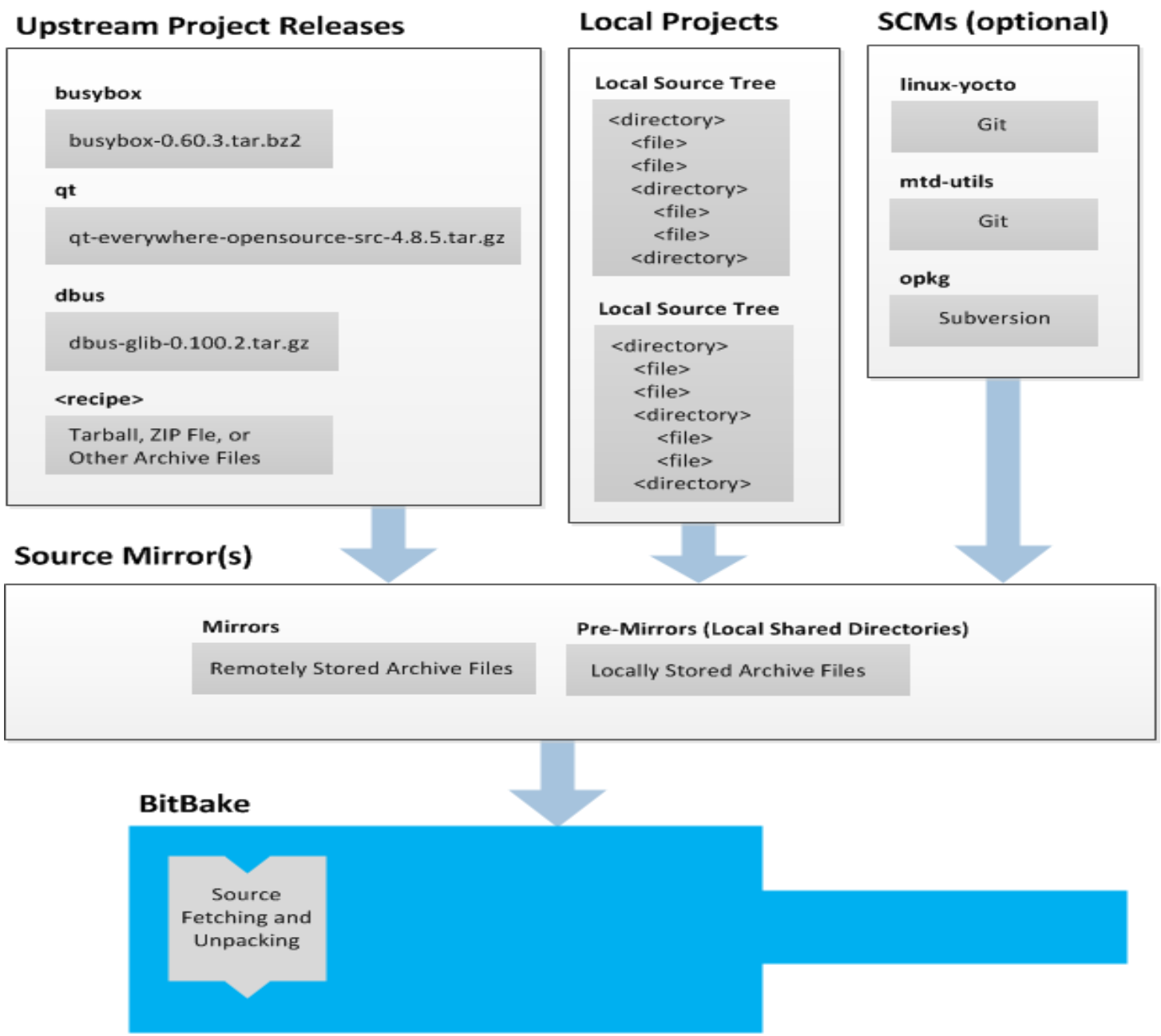
The Yocto Project Development Environment

Sorgenti

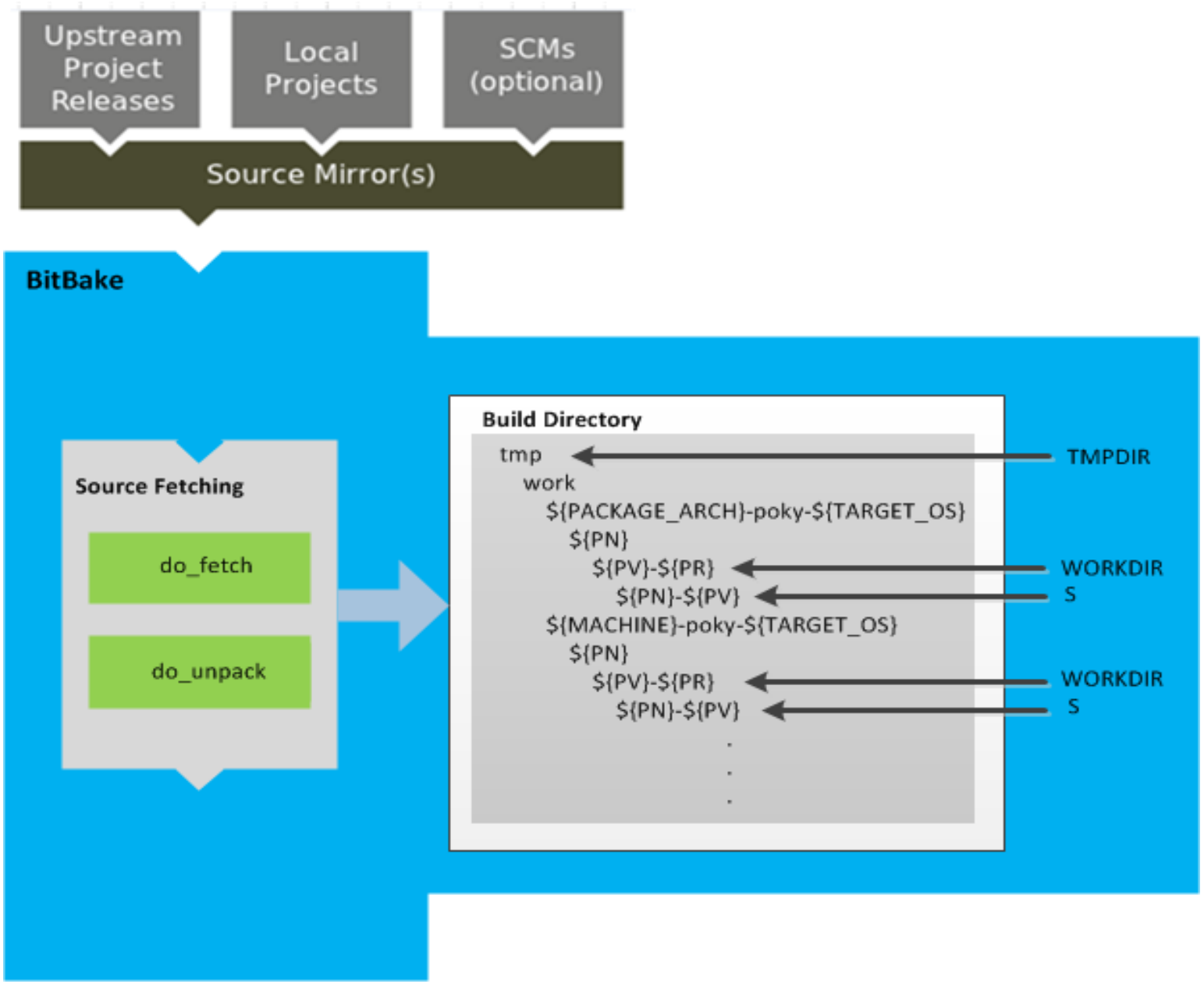


The Yocto Project Development Environment

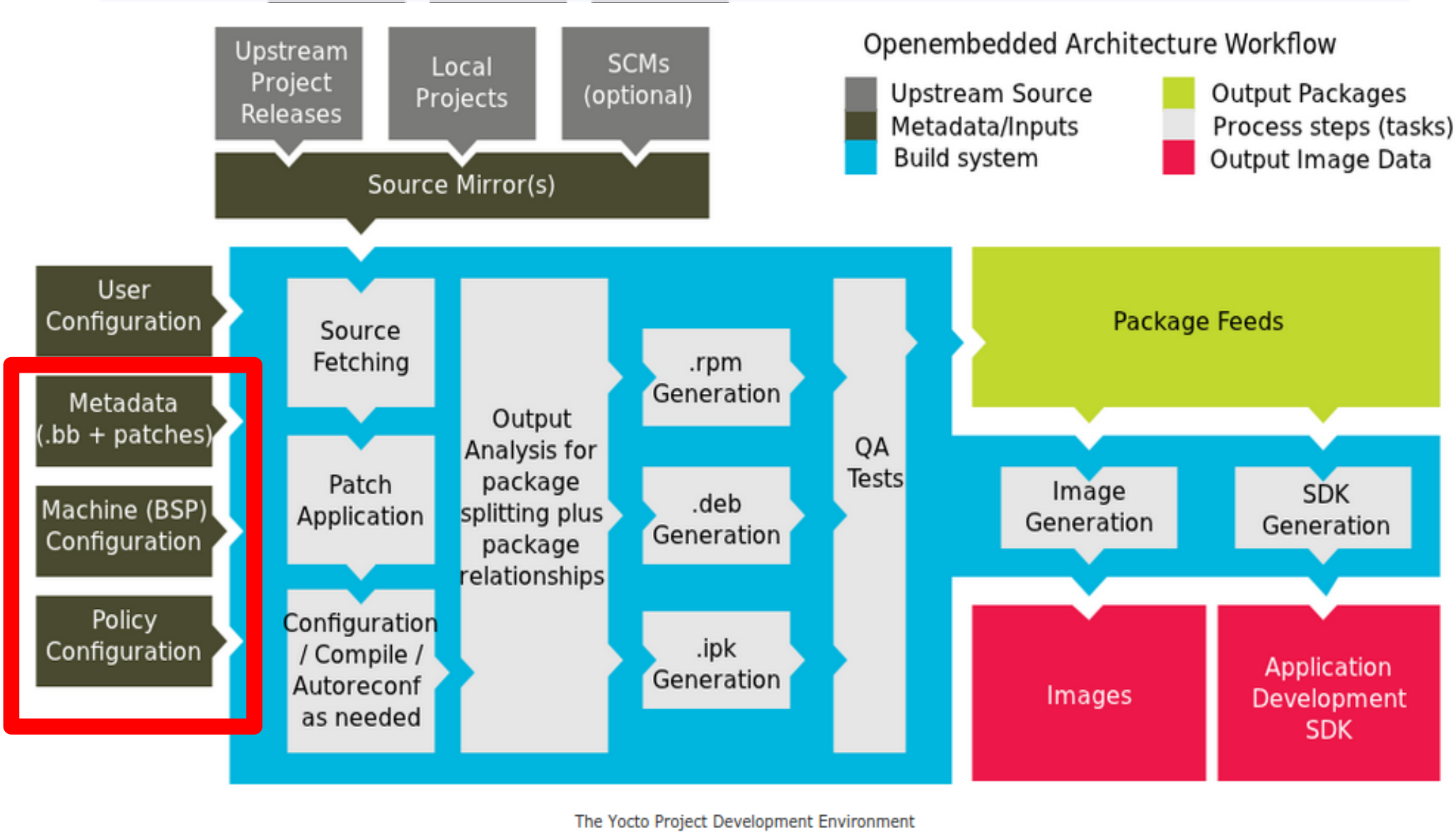
Sorgenti - Fetch



Sorgenti - Unpack

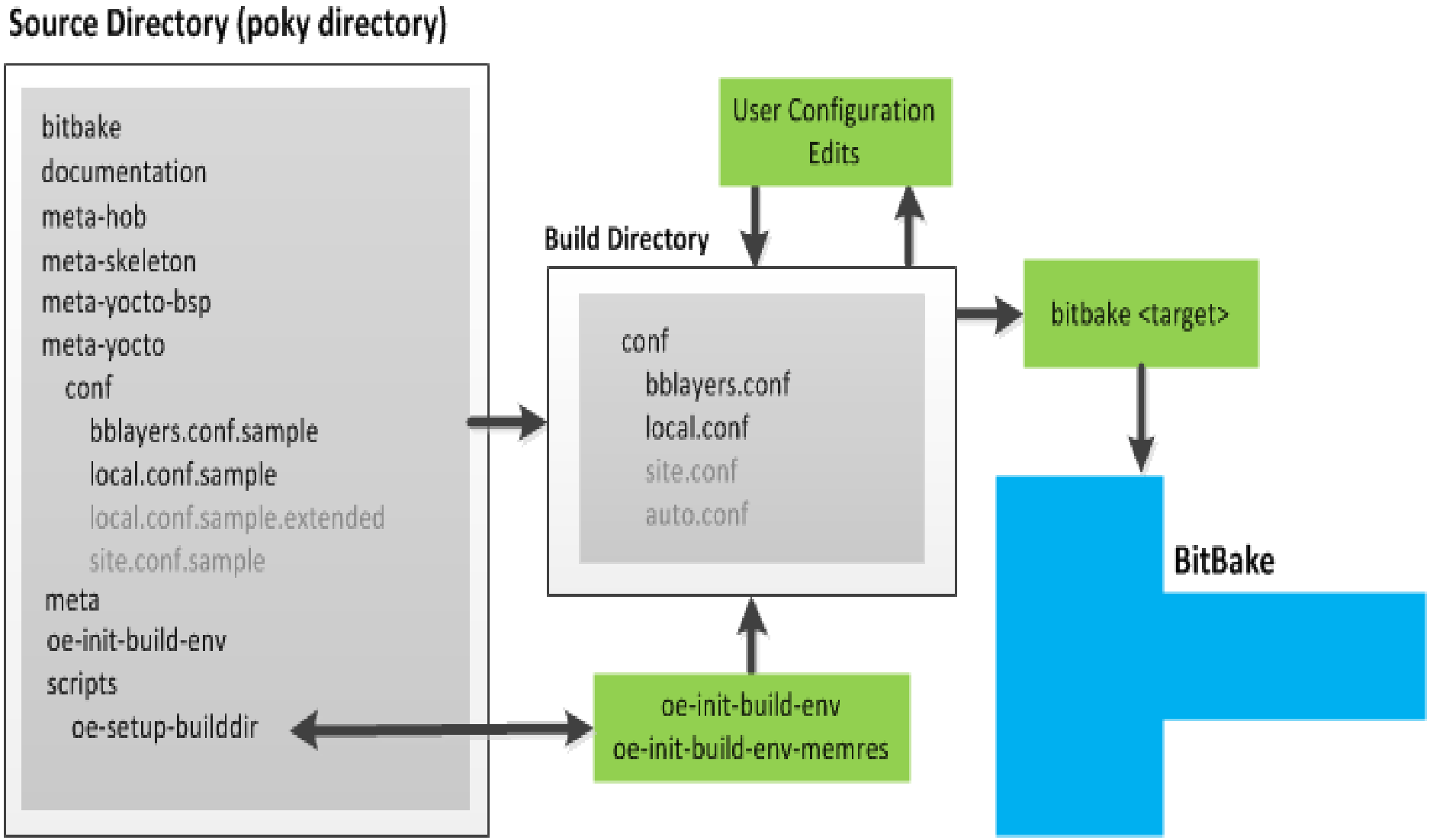


Configurazione

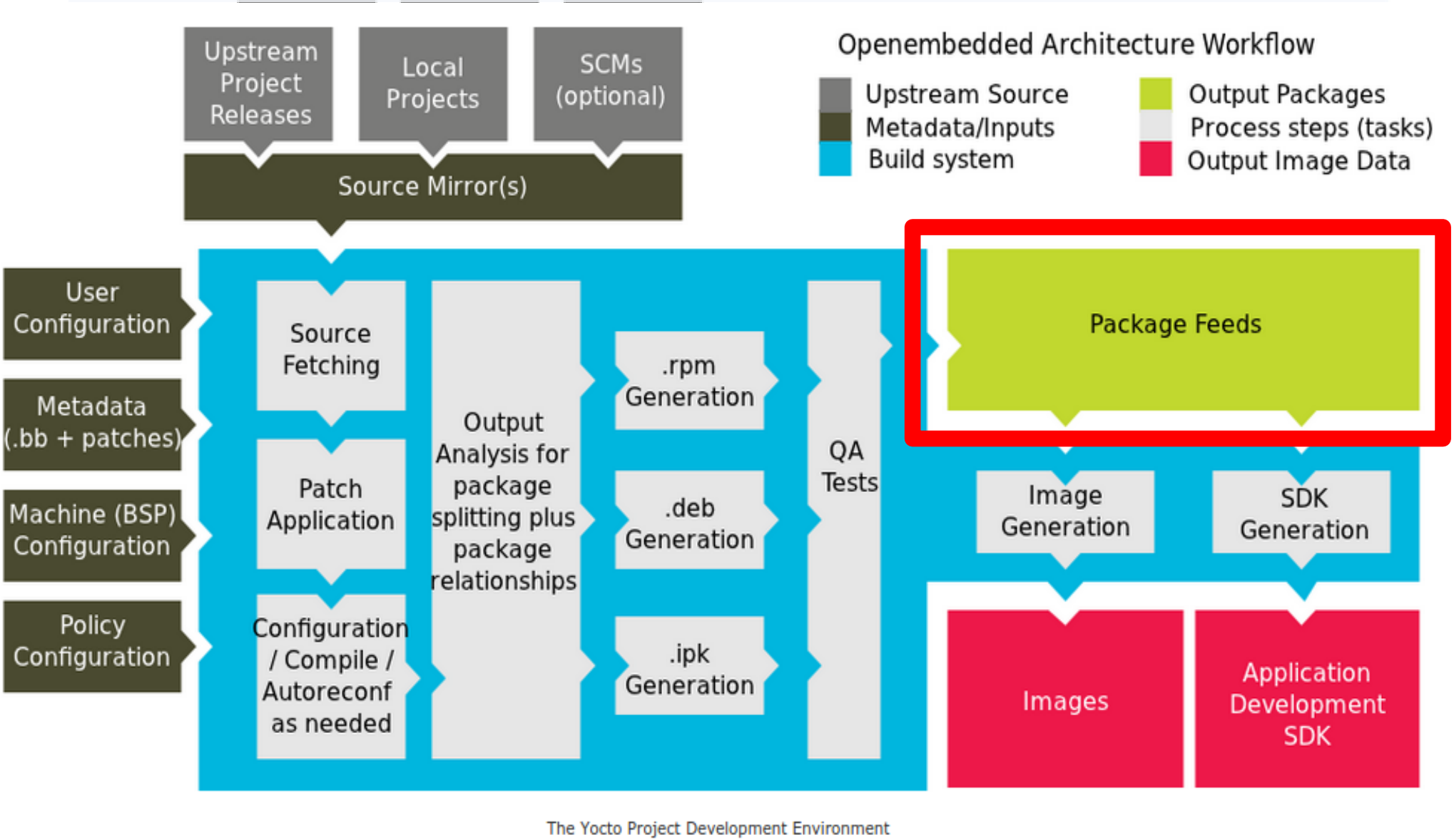


The Yocto Project Development Environment

Configurazione dell'utente

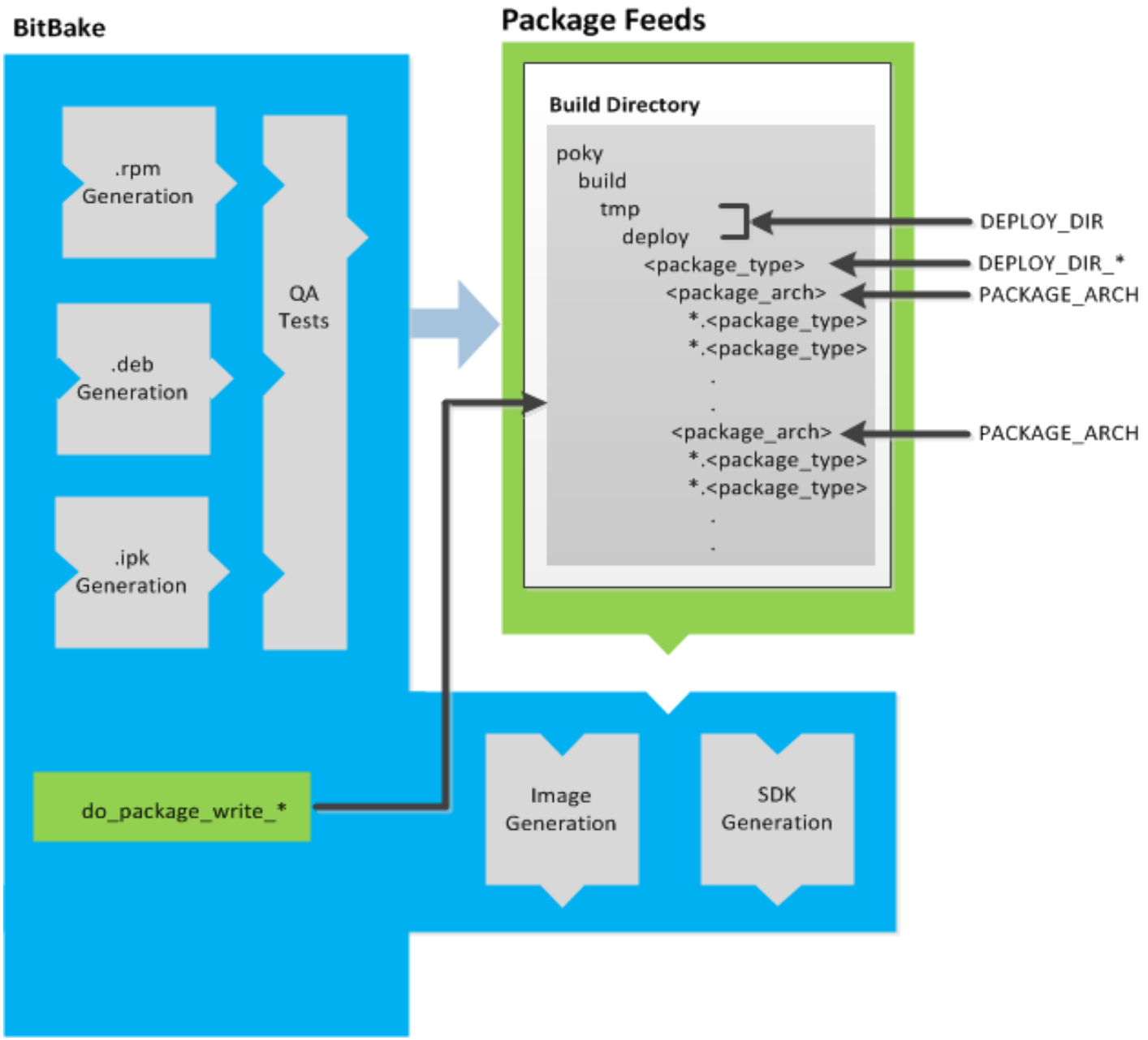


Package feeds



The Yocto Project Development Environment

Package feeds



Per maggiori dettagli

Yocto Project reference manual

http://www.yoctoproject.org/docs/2.0/ref-manual/ref-manual.html


yocto
PROJECT

Procedura di installazione di Yocto Project

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

http://wiki.kaeilos.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 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
```

- Scaricamento sorgenti di Yocto dal repository git

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

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

```
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.
```

- Albero dei sorgenti scaricati

```
$ .
└── poky
    ├── bitbake
    ├── documentation
    ├── LICENSE
    ├── meta
    ├── meta-hob
    ├── meta-skeleton
    ├── meta-yocto
    ├── 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 krogoth
```


```
$ cd $HOME/yocto/poky
$ git clone git://git.yoctoproject.org/meta-fsl-arm -b krogoth
$ git clone https://github.com/Freescale/meta-fsl-arm-extra.git -b krogoth
```

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

- Albero con i layer aggiuntivi opzionali

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

Prima inizializzazione

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

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

- Directory di build

```
└── poky
    ├── bitbake
    ├── build                    <----- lanciare bitbake dentro questa dir
    ├── documentation
    ├── LICENSE
    ├── meta
    ├── meta-hob
    ├── meta-fsl-arm
    ├── meta-fsl-arm-extra
    ├── meta-openembedded/
    ├── meta-qt5
    ├── meta-skeleton
    ├── meta-yocto
    ├── meta-yocto-bsp
    ├── oe-init-build-env
    ├── README
    ├── README.hardware
    └── scripts
```

Configurazione

E' necessario configurare due files

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

Configurazione

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

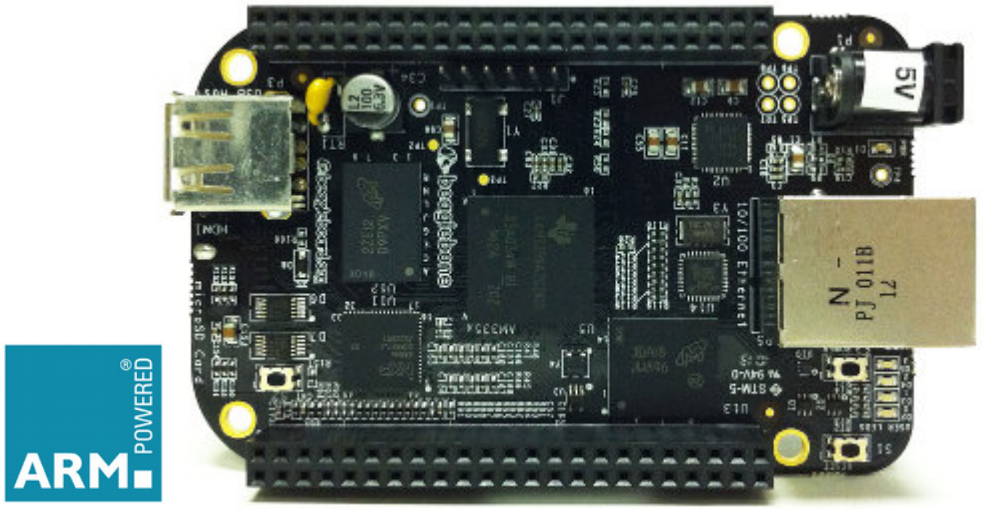
```
BBLAYERS ?= " \
/home/tux/yocto/poky/meta \
/home/tux/yocto/poky/meta-yocto \
/home/tux/yocto/poky/meta-yocto-bsp \
/home/tux/yocto/poky/meta-openembedded/meta-oe \
/home/tux/yocto/poky/meta-fsl-arm \
/home/tux/yocto/poky/meta-fsl-arm-extra \
"
```

Configurazione

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

```
PACKAGE_CLASSES ?= "package_ipk"

MACHINE ?= "beaglebone"
```



Compilazione

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

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

  - 2. Impostazione della directory di build

```
$ source ./oe-init-build-env
```

Compilazione

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

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

  - E potremo lanciare **bitbake**

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

Compilazione

```
$ bitbake core-image-minimal


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-14.04"
TARGET_SYS           = "i586-poky-linux"
MACHINE              = "beaglebone"
DISTRO               = "poky"
DISTRO_VERSION       = "2.0.1"
TUNE_FEATURES        = "m32 i586"
TARGET_FPU           = ""

meta
meta-yocto
meta-yocto-bsp       = "jethro:6dba9abd43f7584178de52b623c603a5d4fcec5c"

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)
```

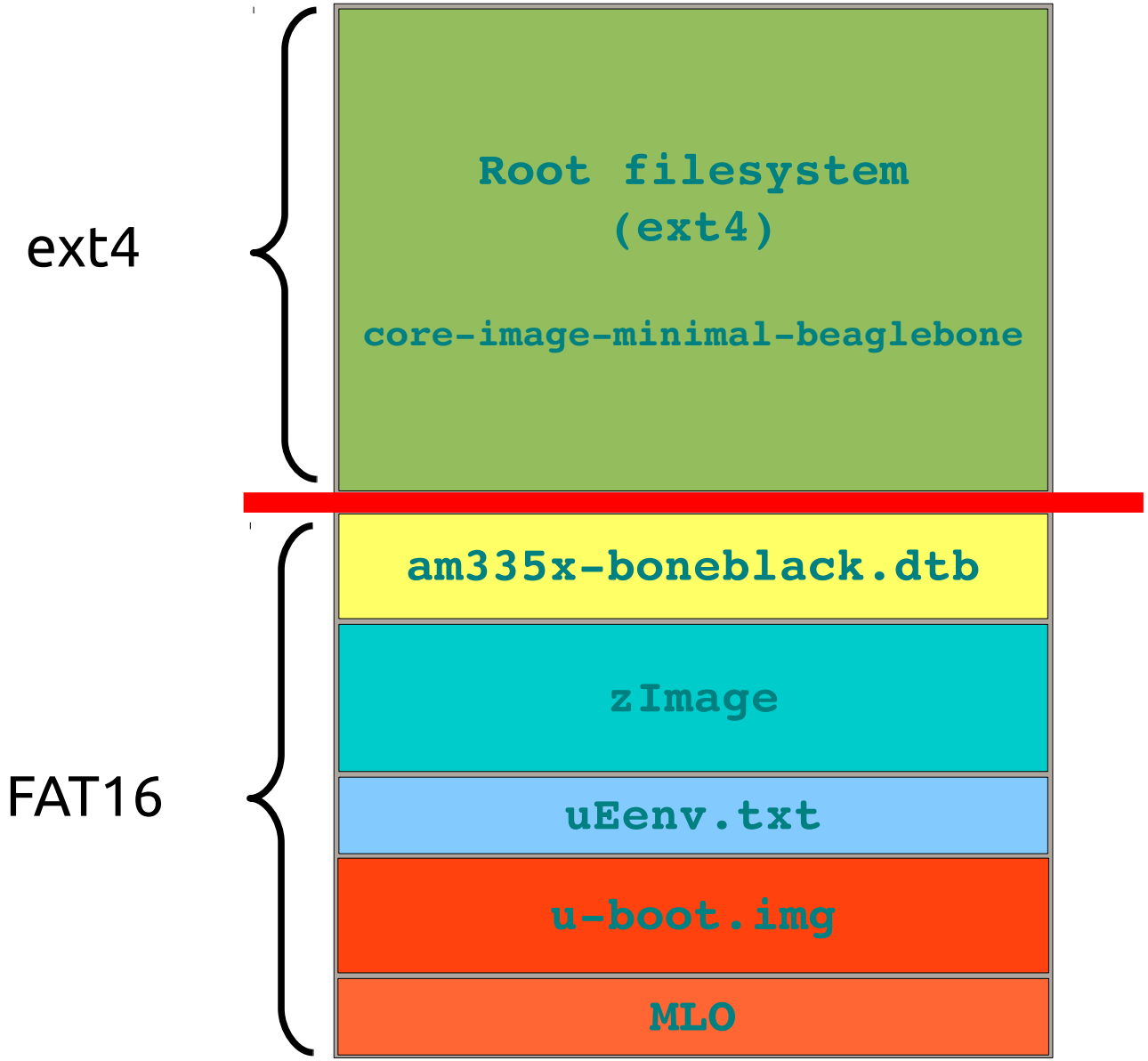
Risultato della compilazione

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

core-image-minimal-beaglebone.jffs2
core-image-minimal-beaglebone.manifest
core-image-minimal-beaglebone.tar.bz2
MLO
README_-_DO_NOT_DELETE_FILES_IN_THIS_DIRECTORY.txt
u-boot.img
zImage
zImage-am335x-boneblack.dtb
```

lista dei files ridotta

Contenuto della microSD



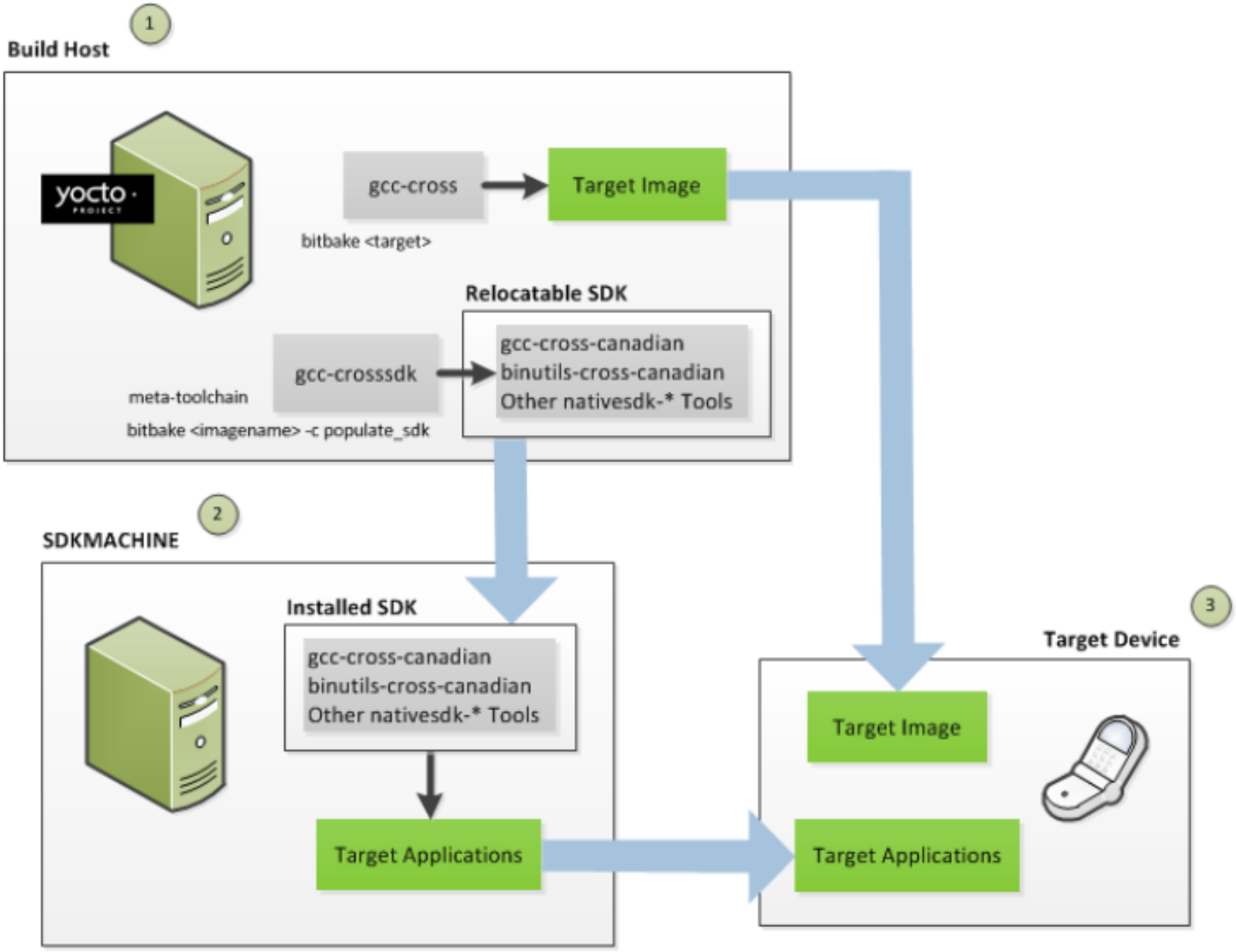

yocto
PROJECT

Creazione del cross-compilatore

- Yocto ci permette di creare un cross-compilatore

```
$ bitbake meta-toolchain
```

Cross-Development Toolchain



Installazione del cross-compilatore

- Installazione del cross-compilatore (redistribuzione)

```
$ cd $HOME/yocto/poky/build/tmp/deploy/sdk
$ ./poky-glibc-x86_64-meta-toolchain-cortexa8hf-vfp-neon-toolchain-2.0.1.sh
```

- Verrà installato in /opt/poky/2.0.1

Installazione del cross-compilatore

- Il cross-compilatore verrà installato in

```
/opt/poky/2.0.1/
```

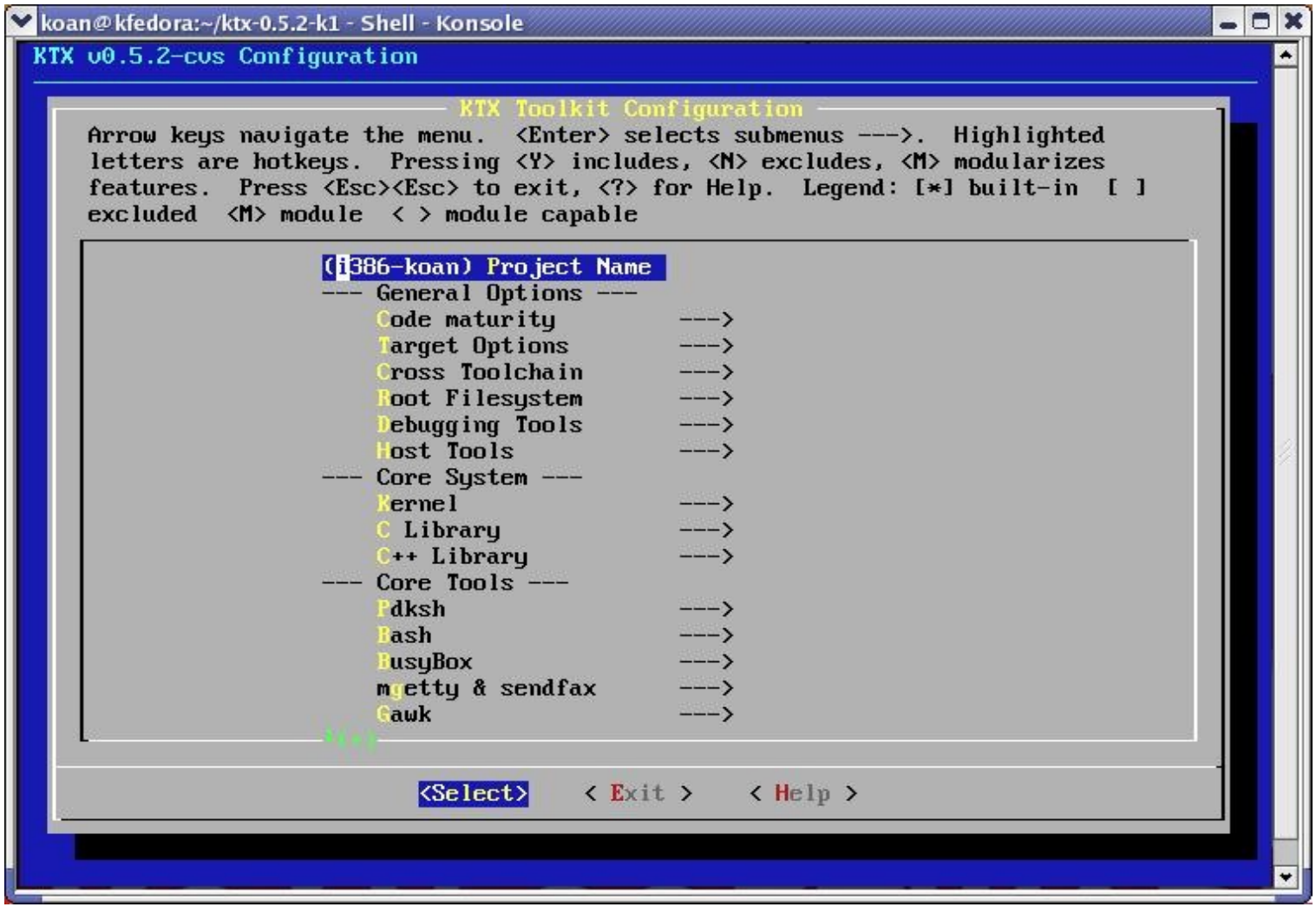
- E potrà essere usato dopo aver impostato

```
$ source /opt/poky/2.0.1/environment-setup-
cortexa8hf-vfp-neon-poky-linux-gnueabi
```

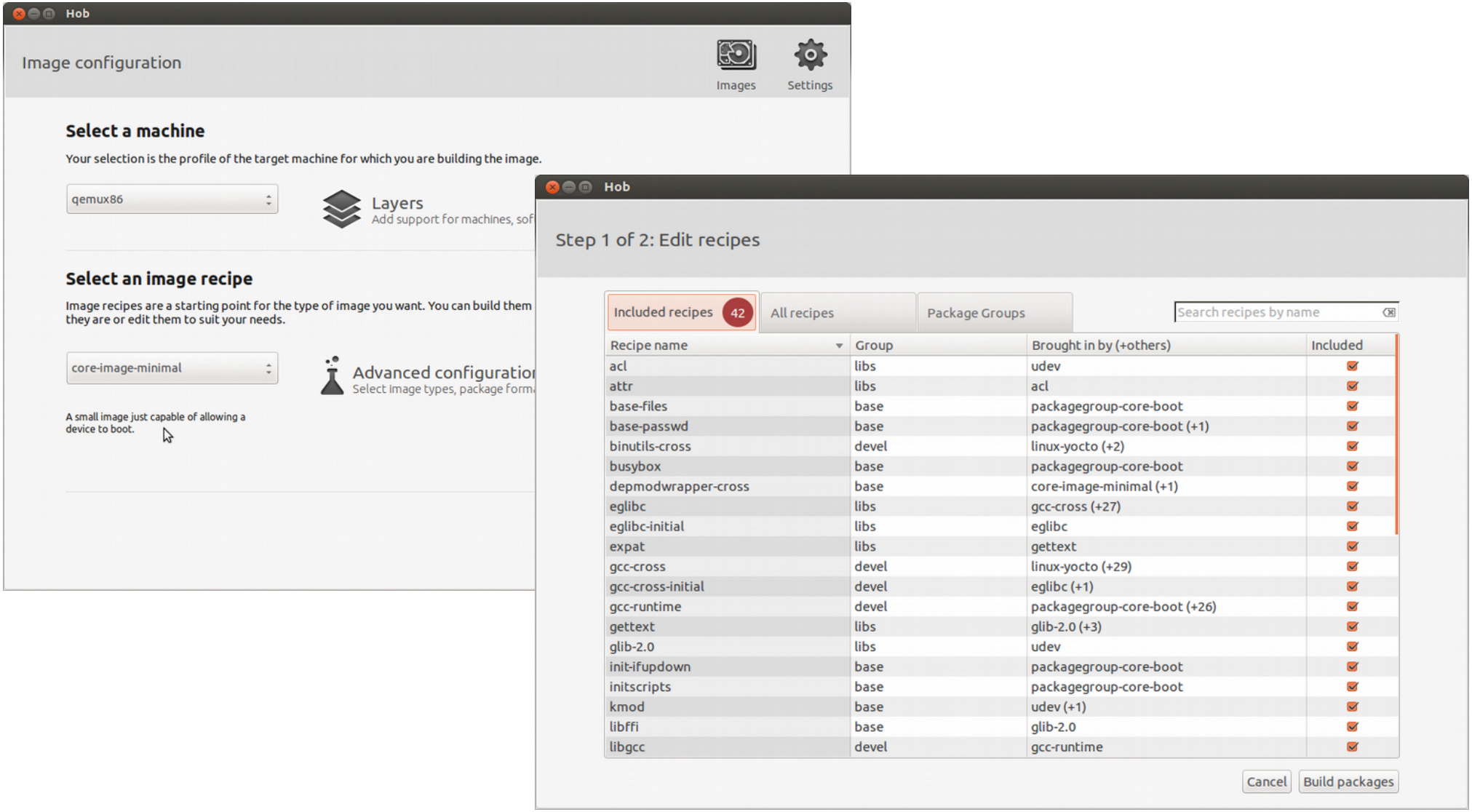

yocto
PROJECT

Configurazione - menuconfig

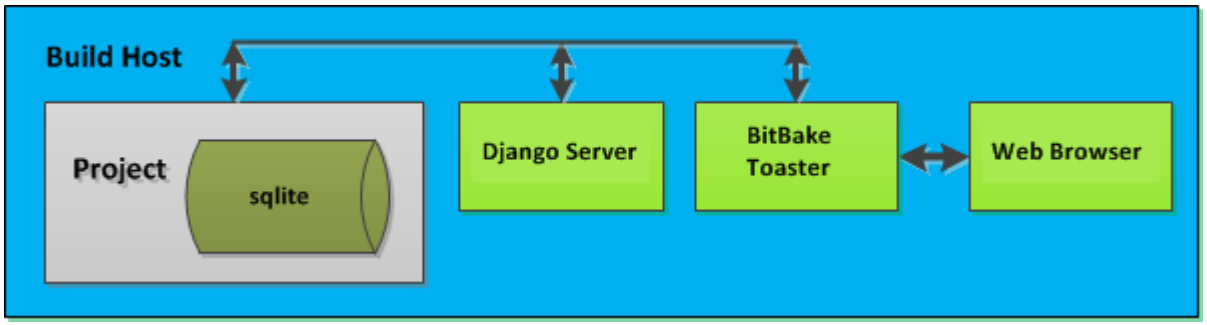
Configurazione tramite un'interfaccia a caratteri “menuconfig”.



Configurazione tramite pacchetto “HOB”.



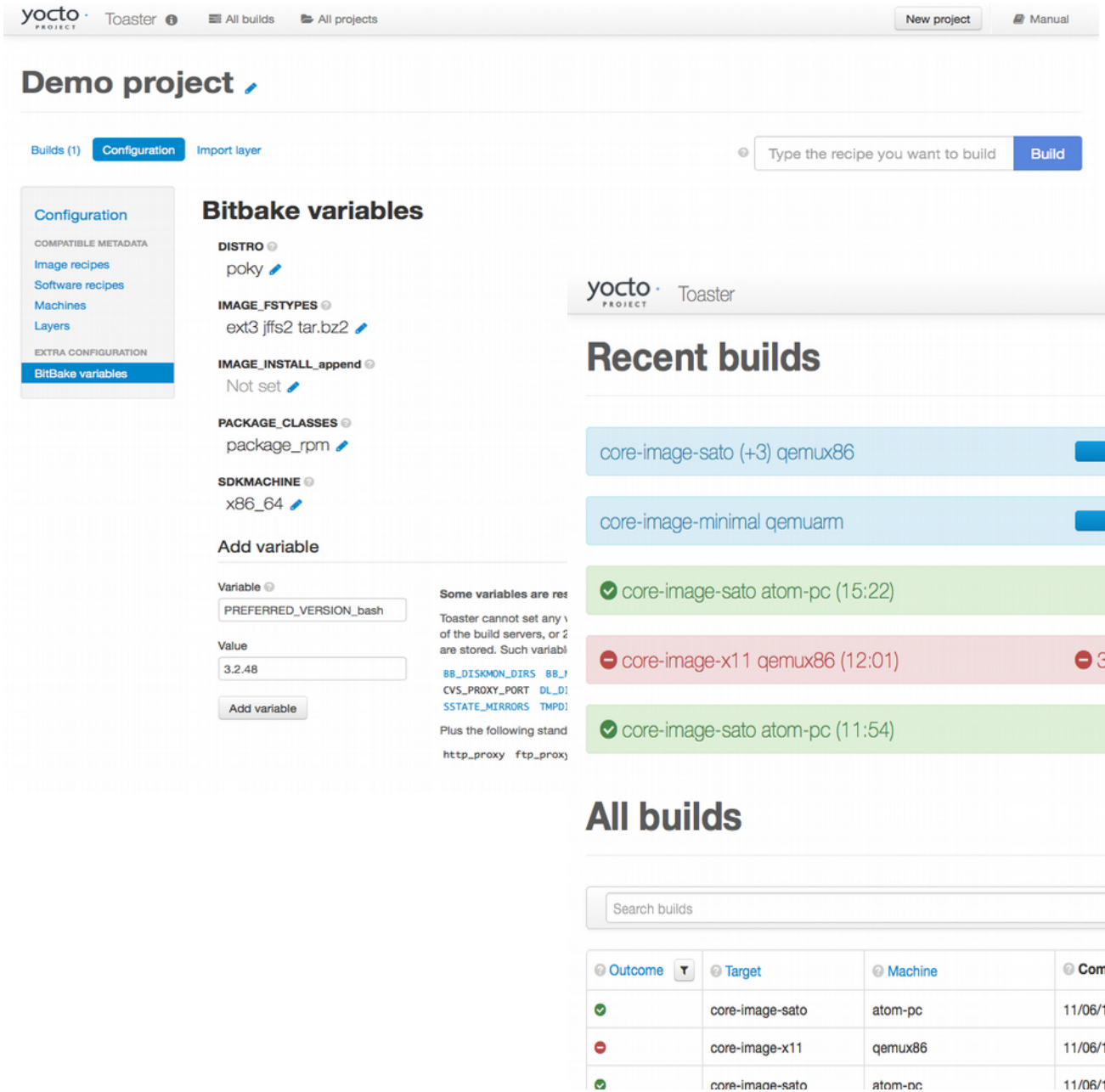
Configurazione tramite pacchetto “Toaster”.

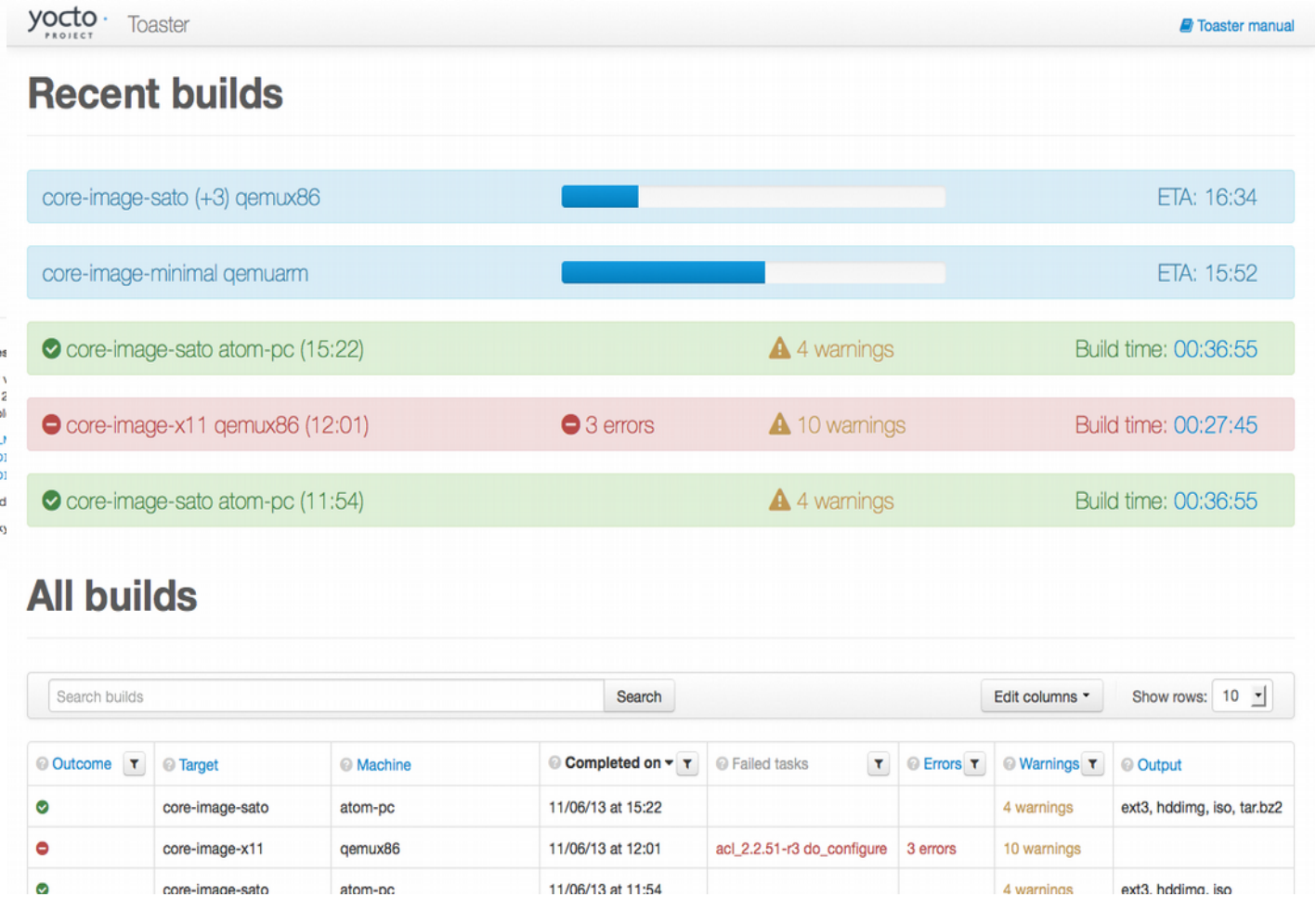


Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

https://www.yoctoproject.org/docs/2.0/toaster-manual/toaster-manual.html

Toaster - Configurazione






yocto
PROJECT

Libri

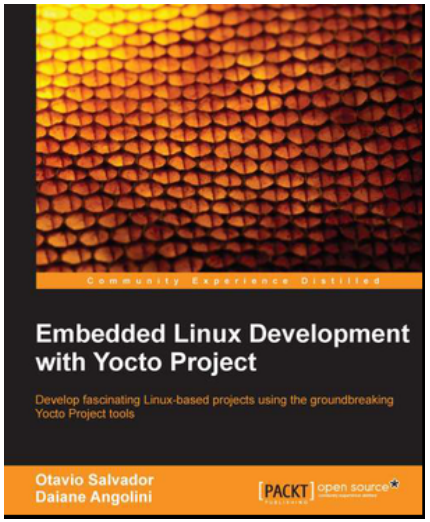
Embedded Linux Development with Yocto Project, by Otavio Salvador and Daiane Agolini

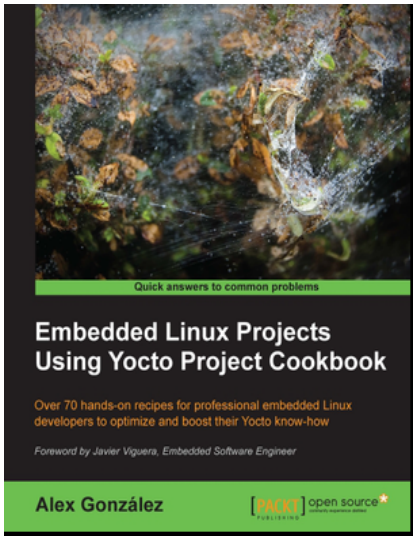
Embedded Linux Projects Using Yocto Project Cookbook, by Alex Gonzaléz ***

Learning Embedded Linux using the Yocto Project, by Alexandru Vaduva

Using Yocto Project with BeagleBone Black, by H M Irfan Sadiq

Yocto for Raspberry Pi, by Pierre-Jean Texier and Petter Mabäcker





***

http://yoctoproject.org

Domande?

Rights to copy

m.cavallini <AT> koansoftware.com

Corrections, suggestions,
contributions and translations are welcome!

ftp://ftp.koansoftware.com/public/talks/LinuxDay2016/

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

KOAN
embedded software engineering

Follow us on LinkedIn

http://koansoftware.com

yocto PROJECT PARTICIPANT