yocto PROJECT

It's not an embedded Linux distribution –
It creates a custom one for you.

Developing Embedded Linux Devices Using the Yocto Project™

Mark Hatle
mark.hatle@windriver.com
Wind River Systems
September, 2012

WIND RIVER yocto PROJECT THE LINUX FOUNDATION

Agenda

- What is the Yocto Project (YP)?
- How does it work?
- How to get started with building OS, apps, and debugging
- What's Next?
- Q&A

What is the Yocto Project?

- A collection of embedded projects and tooling
- Open source project with a strong community
- Key project is the distribution build environment (Poky)
  - Complete Linux OS with package metadata
  - Releases every 6 months with latest (but stable) kernel (LTSI), toolchain, and package versions
  - Place for Industry to publish BSPs
  - App Dev Tools which allow development against the stack, including Eclipse plug-ins and emulators
  - Full documentation representative of a consistent system

It's not an embedded Linux distribution – it creates a custom one for you

WIND RIVER yocto PROJECT THE LINUX FOUNDATION

Why is the Yocto Project Valuable?

- Linux is becoming increasingly popular for Embedded
- Non-commercial and commercial embedded Linux has many distributions

  (too many distributions)
- Do It Yourself or Starting w/ a desktop/server OS:
  - Long Term Maintenance is difficult
  - Upstream changes are difficult to track
  - Not embedded friendly
  - Licensing issues
  - No commercial Embedded support
  - *Developers spend lots of time porting or making build systems*
  - *Leaves less time/money to develop interesting software features*

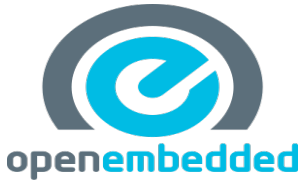
Why is the Yocto Project Valuable?

- The industry needed a common build system and core technology
  - Bitbake build system, OpenEmbedded base
- The benefit of doing so is:
  - Designed for the long term
  - Designed for embedded
  - Transparent Upstream changes
  - Vibrant Developer Community
  - *Less time spent on things which don't make money* (build system, core Linux components)
  - *More time spent on things which do make money* (application development, product development, …)

Who is the Yocto Project?

- Advisory Board and Technical Leadership:
  - Organized under the Linux Foundation
  - Individual Developers
  - Embedded Hardware Companies
  - Semiconductor Manufacturers
  - Embedded Operating System Vendors
  - OpenEmbedded / LTSI Community

Cavium Freescale Tilera
LSI Mindspeed
NetLogic Microsystems



Why Should a Developer Care?

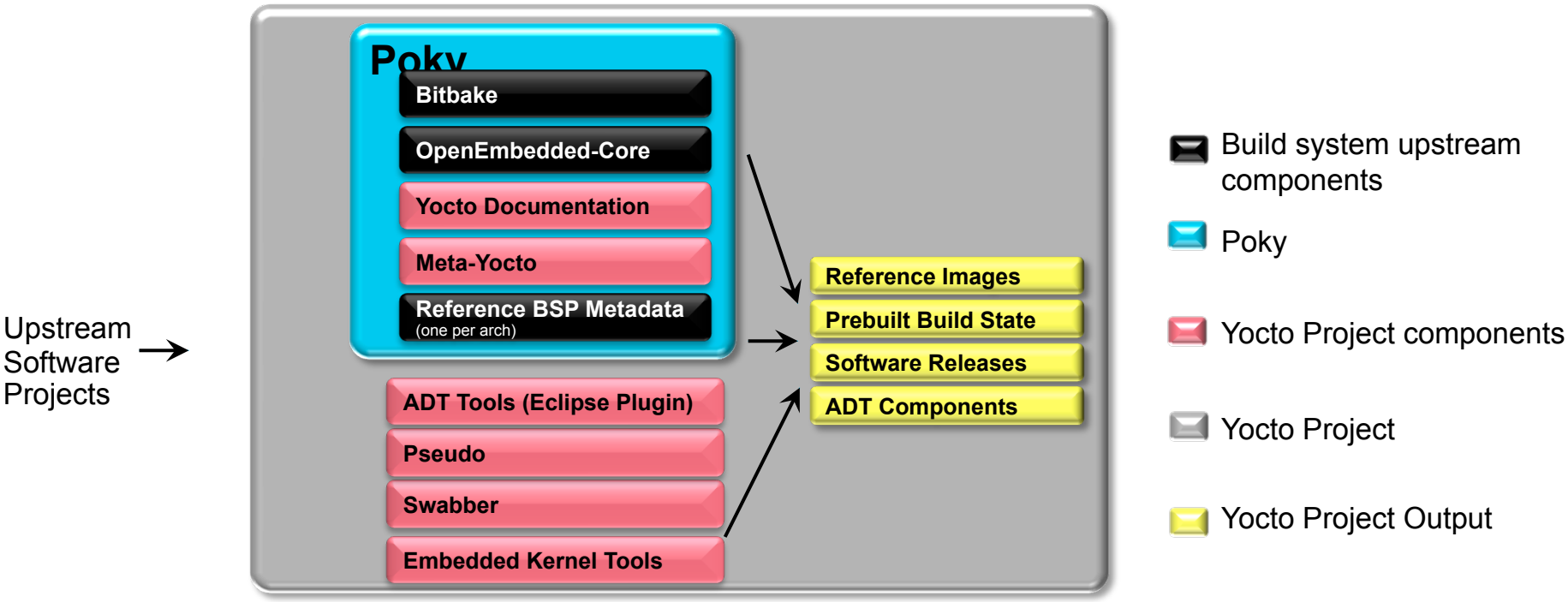
- Build a complete Linux system –from source– in about an hour (about 90 minutes with X).
- Start with a validated collection of software (toolchain, kernel, user space).
- Access to a great collection of app developer tools (performance, debug, power analysis, Eclipse). We distinguish app developers from system developers and we support both.
- Advanced kernel development tools.
- Supports all major embedded architectures (x86, x86-64, ARM, PPC, MIPS), just change a line in a config file and rebuild.
- Transitions easily to a commercial embedded Linux based on the Yocto Project.

How Does It Work? – Quick Start

1. Go to http://yoctoproject.org, click “documentation” and consult the Quick Start guide
2. Set up your Linux system with the right packages (and firewall access, if needed)
3. Click “Download” and download the latest stable release (or check out “denzil” from the git repo)
4. Source `oe-init-build-env` script
5. Edit `conf/local.conf` and set MACHINE, BB_NUMBER_THREADS and PARALLEL_MAKE
6. Run `$ bitbake core-image-sato`
7. Run `$ runqemu qemux86` (if MACHINE=qemux86)

Note: File or command names in this presentation are subject to change, several are different now in master.

Yocto Project = Poky + Upstream + Tools



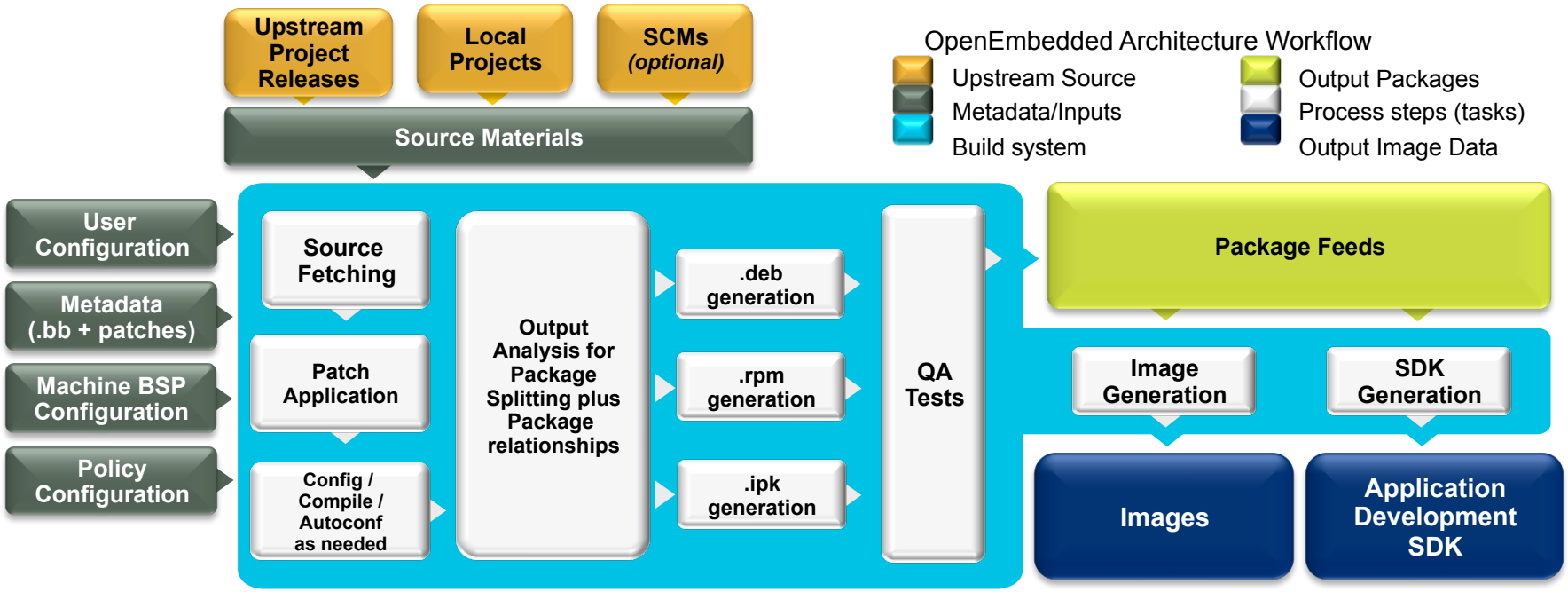
Yocto Project provides best of upstream for a stable base

Why not just use OpenEmbedded?

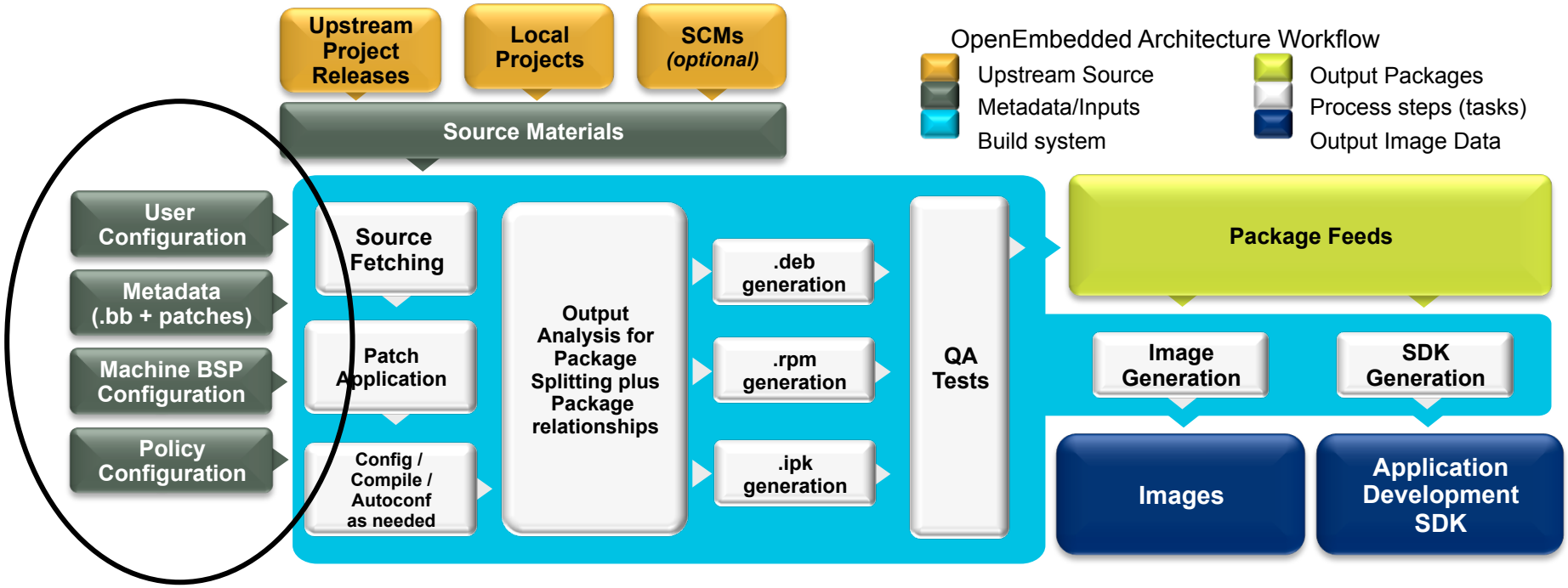
- OpenEmbedded is an Open Source project focused on enabling cross-compiled systems. It contains the build system, and meta data that describe many components of a system.

- Yocto Project is focused on enabling commerical developers. It does this by helping to improve the quality of OpenEmbedded. Leveraging the build system and core components of OpenEmbedded, and extending the support to key COTS boards.
  - Yocto Project includes autobuilder sessions
  - QA testings
  - Eclipse Plugins
  - …etc…

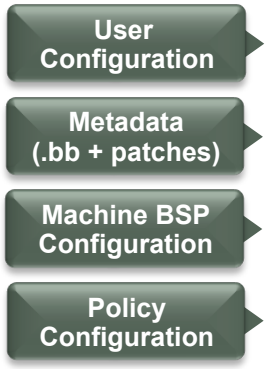
Yocto Project* Workflow



Configuration

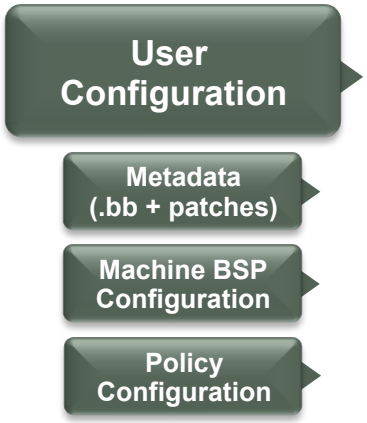


Configuration



- Configuration (*.conf) – global definition of variables
  - build/conf/local.conf (local user-defined variables)
  - meta-yocto/conf/distro/poky.conf (Yocto policy config variables)
  - meta/machine/include/tune-mips32.inc (processor specific configuration)
  - meta-yocto/machine/routerstationpro.conf (machine-specific variables)

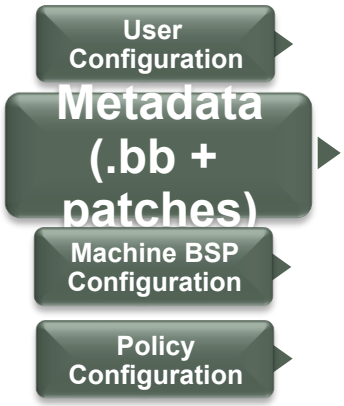
User Configuration



User configuration:

- conf/local.conf – some things to set:
  - Set BB_NUMBER_THREADS and PARALLEL_MAKE
  - Set MACHINE="foo" for the CPU architecture
  - EXTRA_IMAGE_FEATURES adds features (groups of packages)
  - INCOMPATIBLE_LICENSE = "GPLv3" eliminates packages using 'GPLv3' and 'LGPLv3'

Metadata



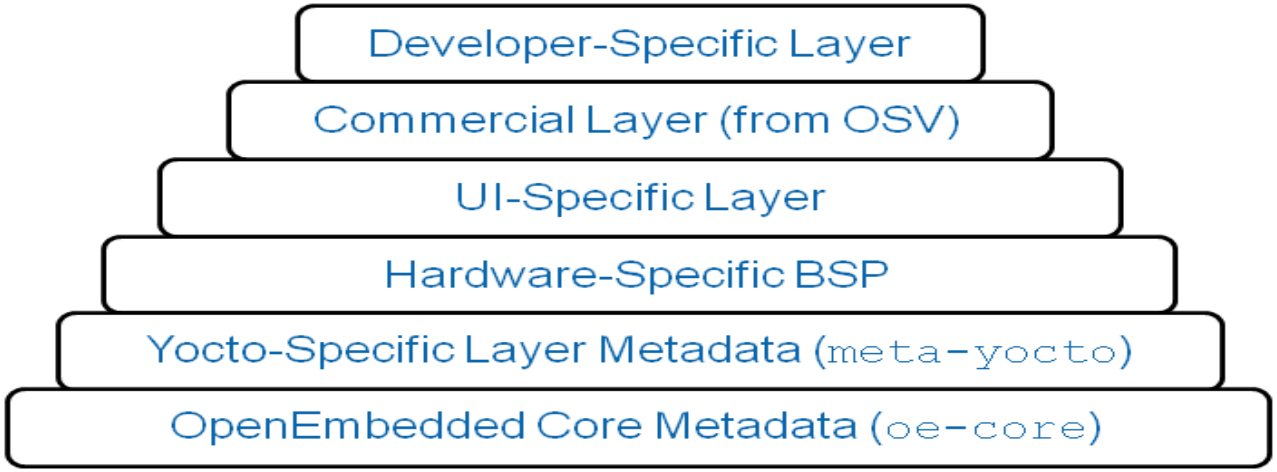
Metadata and patches:

- Recipes for building packages
- Eg, `meta/recipes-core/coreutils/coreutils_6.9.bb` builds the core utilities (version 6.9) and installs them
- `meta/recipes-core/coreutils/coreutils-6.9/` includes patches, also could include extra files to install
- Can be extended and enhanced via “layers”
- Compatible with OpenEmbedded

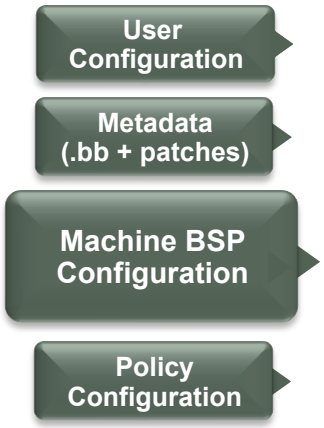
Yocto Project Layer Requirements

- Layers contain extensions and customizations to base system
  - Can include image customizations, additional recipes, modifying recipes, adding extra configuration
    - Really just another directory to look for recipes or recipe extensions
    - Added to the BBLAYERS variable in build/conf/bblayers.conf
- Best Practice: Layers are grouped by functional components
- Common Examples of Layers:
  - Custom Toolchains (compilers, debuggers, profiling tools, etc)
  - Distribution specifications and customizations
  - Functional areas (selinux, networking, etc)
  - OSV components
  - Project level changes
  - BSP/Machine specific components

Layers


Developer-Specific Layer
Commercial Layer (from OSV)
UI-Specific Layer
Hardware-Specific BSP
Yocto-Specific Layer Metadata (meta-yocto)
OpenEmbedded Core Metadata (oe-core)

BSPs



BSPs and Machine configurations:

- Configuration files to describe a machine
  - Processor/SOC tuning files
  - Formfactor configurations
- May include Linux kernel enhancements
- Includes board specific kernel configuration
- Eg, `meta-yocto/machine/routerstationpro.conf` describes the Routerstation Pro features, kernel filenames, etc.
- Machine specific package modifications
- Should be implemented via “layers”
- Compatible with OpenEmbedded

Kernel Development

- We try to develop upstream wherever possible
- Two major advances in the Yocto Project:
  - Branching tools: Per-BSP git branches contain machine-specific kernel sources. Tools collect up the relevant tree of branches
  - Kernel features: patches and configuration fragments managed as a functional block
- Results:
  - Can turn on a collection of features for a given BSP
  - Less code duplication
  - Easier to choose a config fragment and patches

Board Support Layers

- BSP Layer - Best Practices:
  - Machine settings and recipes specific to the BSPs only
  - Based on the Yocto Project kernel versions
  - Uses the Yocto Project kernel tooling
- Machine settings are specified in a layer's conf/machine/xxx.conf file(s)
- Examples:
  - Sandy Bridge + Cougar Point:
    - meta-intel/conf/meta-sugarbay/machine/sugarbay.conf
  - Routerstation Pro (MIPS)
    - yocto/meta/conf/machine/routerstationpro.conf

Linux Kernel Details

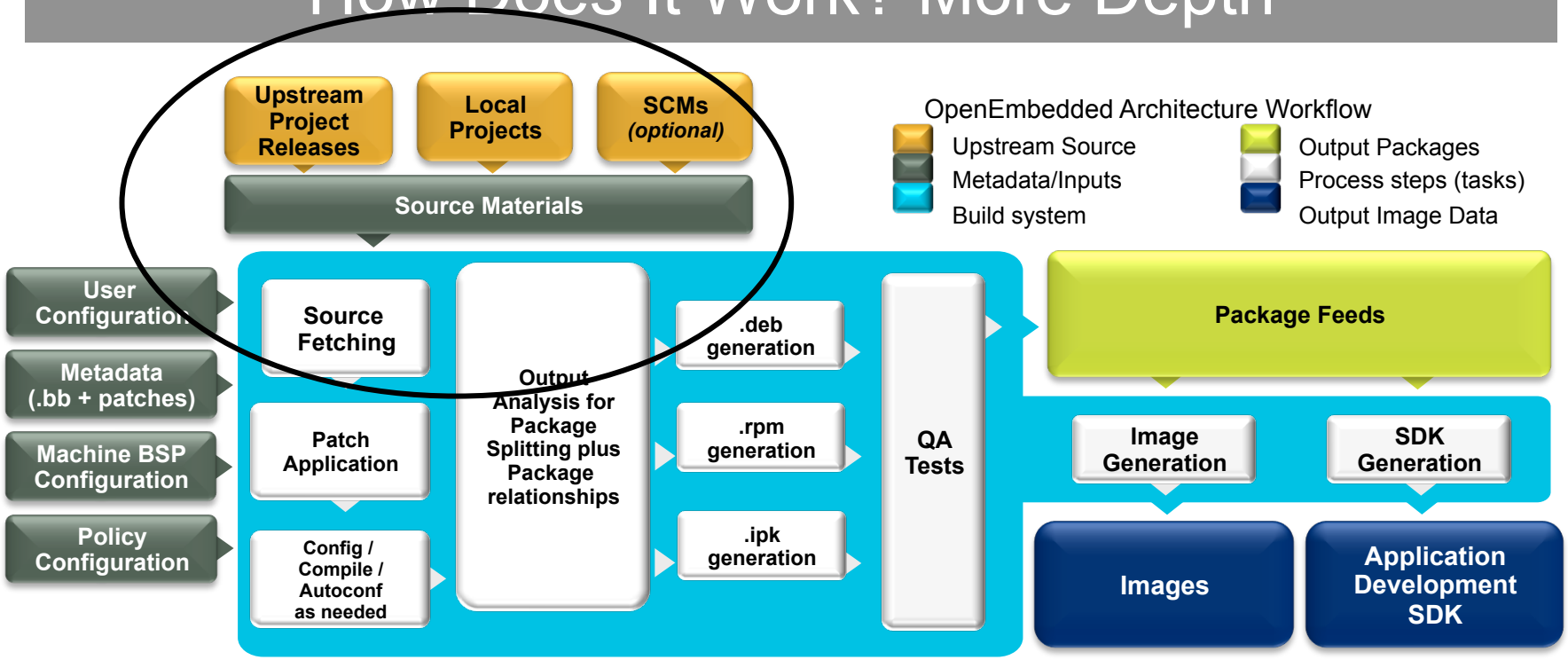
Kernel Tooling

- Kernel class
  - meta/classes/kernel.bbclass
- Linux-Yocto recipe
  - meta/recipes-kernel/linux/linux-yocto*bb
- Linux-Yocto git repository
  - http://git.pokylinux.org/cgit/cgit.cgi/linux-yocto-*

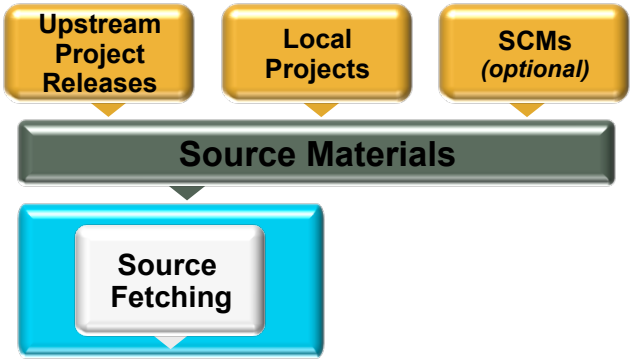
Kernel Versions

- linux-yocto (YP 1.2): 2.6.37, 3.0, 3.2
- linux-yocto (in dev): 3.4
- *linux-yocto-dev: 3.5-rc4 (latest)*

How Does It Work? More Depth



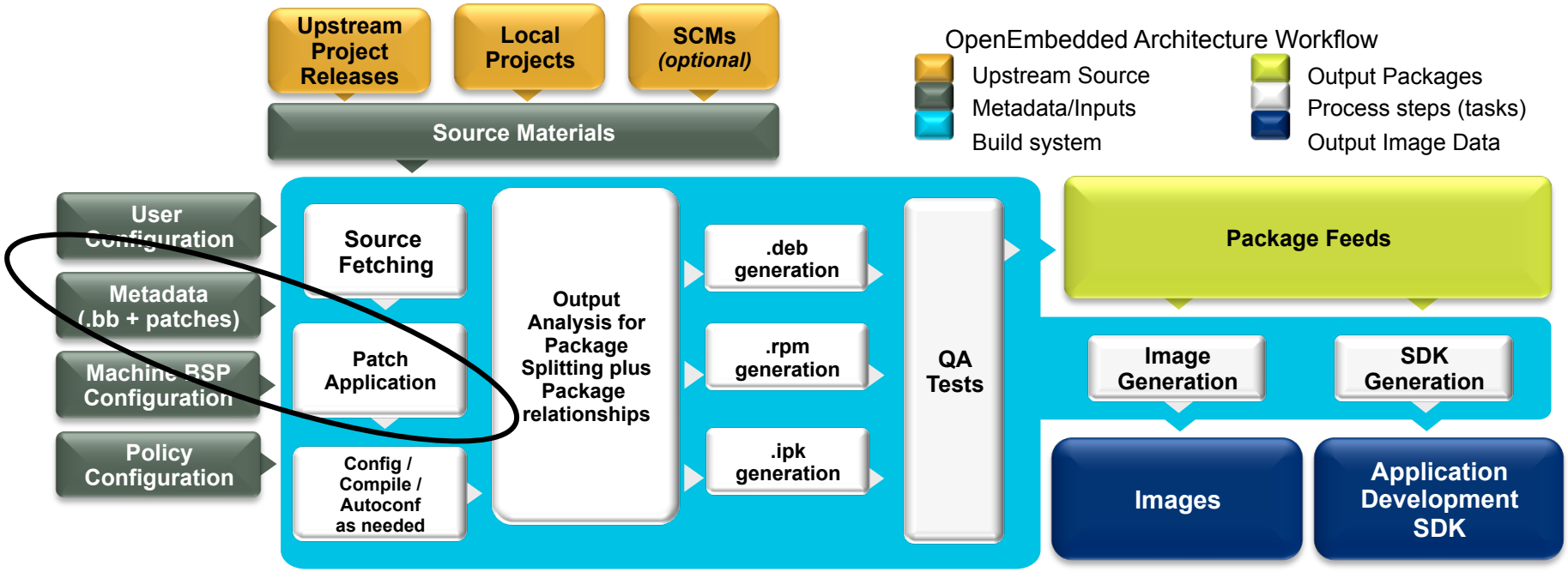
Source Fetching



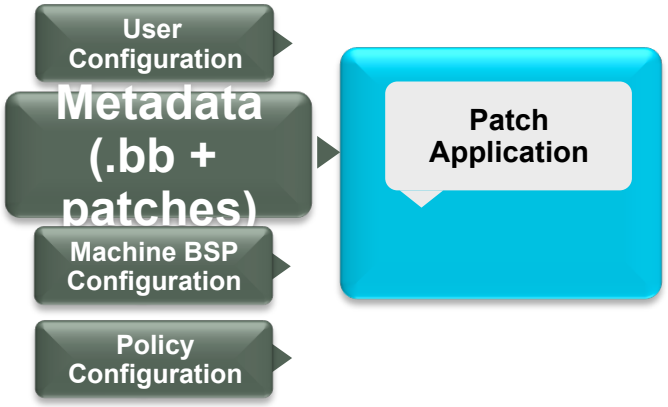
- Recipes call out location of all sources, whether on the internet or local (Look for SRC_URI in *.bb files)
- Bitbake can get sources from git, svn, bzr, from tarballs, and many, many more*
- Versions of packages can be fixed or updated automatically (Add SRCREV_pn- PN = "${AUTOREV}" to local.conf)
- Yocto Project sources mirror available as a fallback (source reliability)

* Complete list includes: http, ftp, https, git, svn, perforce, mercurial, bzr, cvs, osc, repo, ssh, and svk and the unpacker can cope with tarballs, zip, rar, xz, gz, bz2, and so on.

Patching

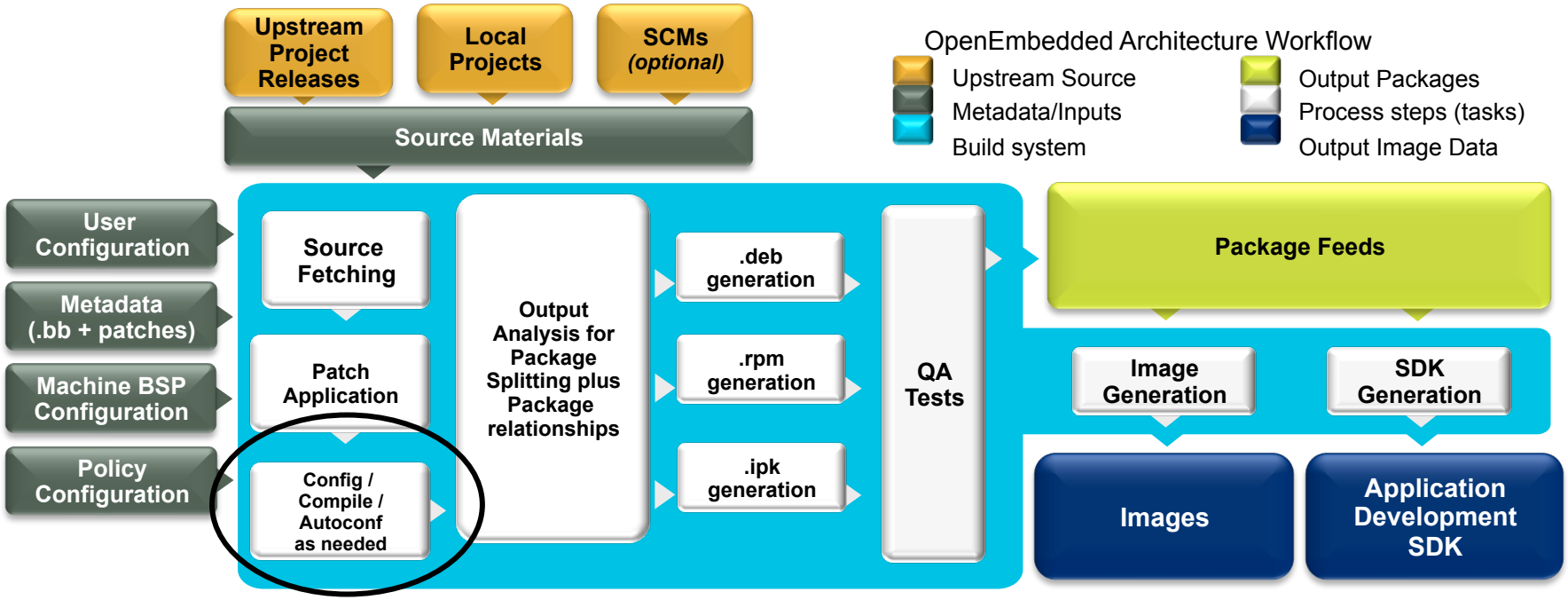

Upstream Project Releases
Local Projects
SCMs (optional)
Source Materials
OpenEmbedded Architecture Workflow
Upstream Source
Metadata/Inputs
Build system
Output Packages
Process steps (tasks)
Output Image Data
User Configuration
Metadata (.bb + patches)
Machine BSP Configuration
Policy Configuration
Source Fetching
Patch Application
Config / Compile / Autoconf as needed
Output Analysis for Package Splitting plus Package relationships
.deb generation
.rpm generation
.ipk generation
QA Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK

Patching

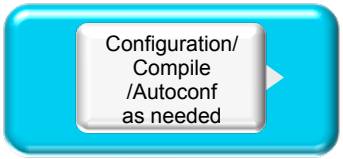


- Once sources are obtained, the patches are applied
- This is a good place to patch the software yourself
- However, we encourage you to contribute development upstream whenever possible (we try to)

Config / Compile



Configure/Compile



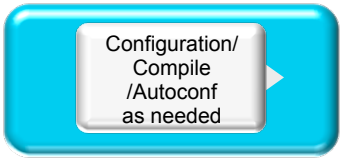
- Autoconf can be triggered automatically to ensure latest libtool is used

```
DESCRIPTION = "GNU Helloworld application"
SECTION = "examples"
LICENSE = "GPLv2+"
LIC_FILES_CHKSUM = "file://COPYING;md5=751419260aa954499f7abaabaa882bbe"
PR = "r0"

SRC_URI = "${GNU_MIRROR}/hello/hello-${PV}.tar.gz"

inherit autotools gettext
```

Configure/Compile



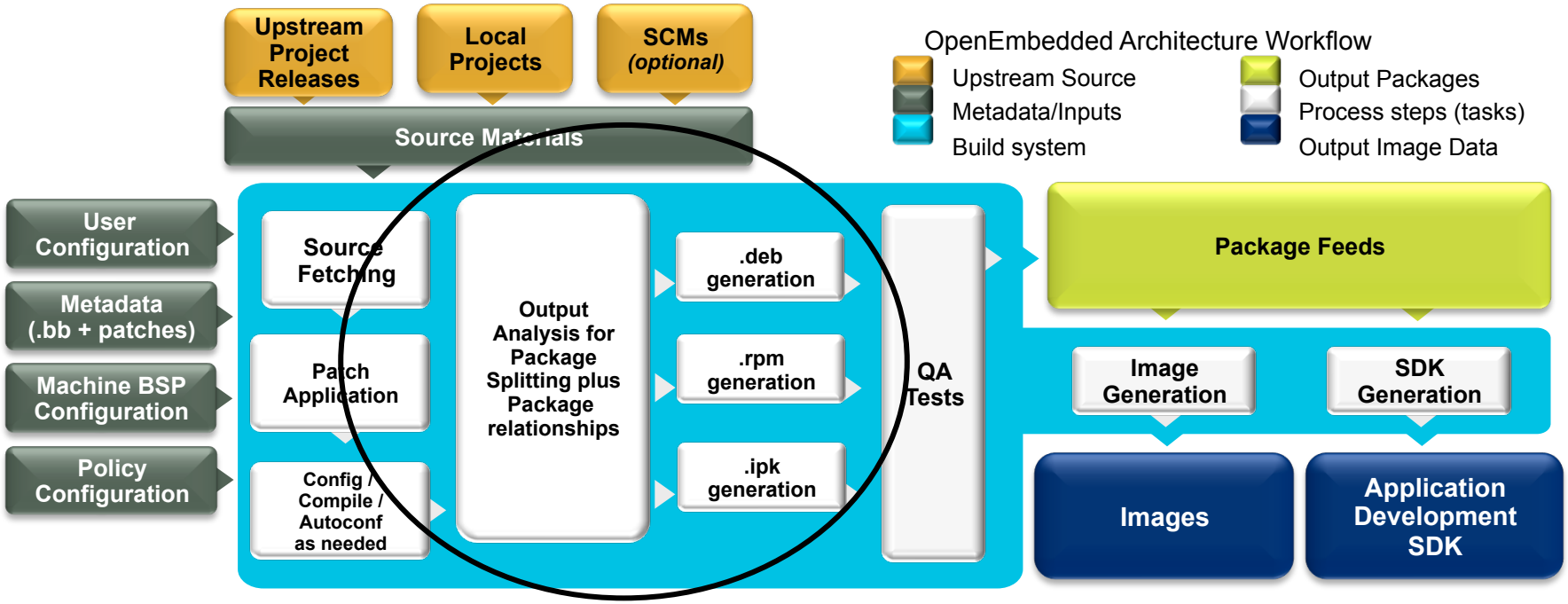
- Set standard environment flags (i.e. CFLAGS)

```
CFLAGS_prepend = "-I ${S}/include ”
```

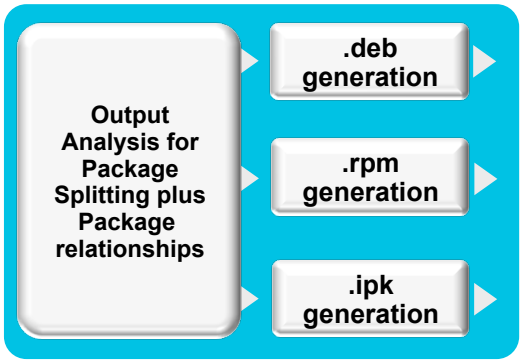
- Install task to set modes, permissions, target directories, done by “pseudo”

```
do_install () {
        oe_runmake install DESTDIR=${D} SBINDIR=${sbindir} MANDIR=${mandir}
```

Packaging


Upstream Project Releases
Local Projects
SCMs (optional)
Source Materials
OpenEmbedded Architecture Workflow
Upstream Source
Metadata/Inputs
Build system
Output Packages
Process steps (tasks)
Output Image Data
User Configuration
Metadata (.bb + patches)
Machine BSP Configuration
Policy Configuration
Source Fetching
Patch Application
Config / Compile / Autoconf as needed
Output Analysis for Package Splitting plus Package relationships
.deb generation
.rpm generation
.ipk generation
QA Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK

Packaging

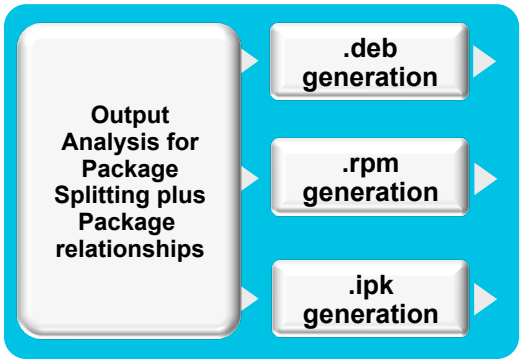


Once configure/compile/install is completed, packaging commences

The most popular package formats are supported: RPM, Debian, and ipk

- Set `PACKAGE_CLASSES` in `conf/local.conf`

Packaging



Once configure/compile/install is completed, packaging commences

You can split into multiple packages using PACKAGES and FILES in a *.bb file:

```
PACKAGES =+ "sxpm cxpm"
FILES_cxpm = "${bindir}/cxpm"
FILES_sxpm = "${bindir}/sxpm"
```

- Automatic splitting into debug, documentation, development and locale data.

Image Generation

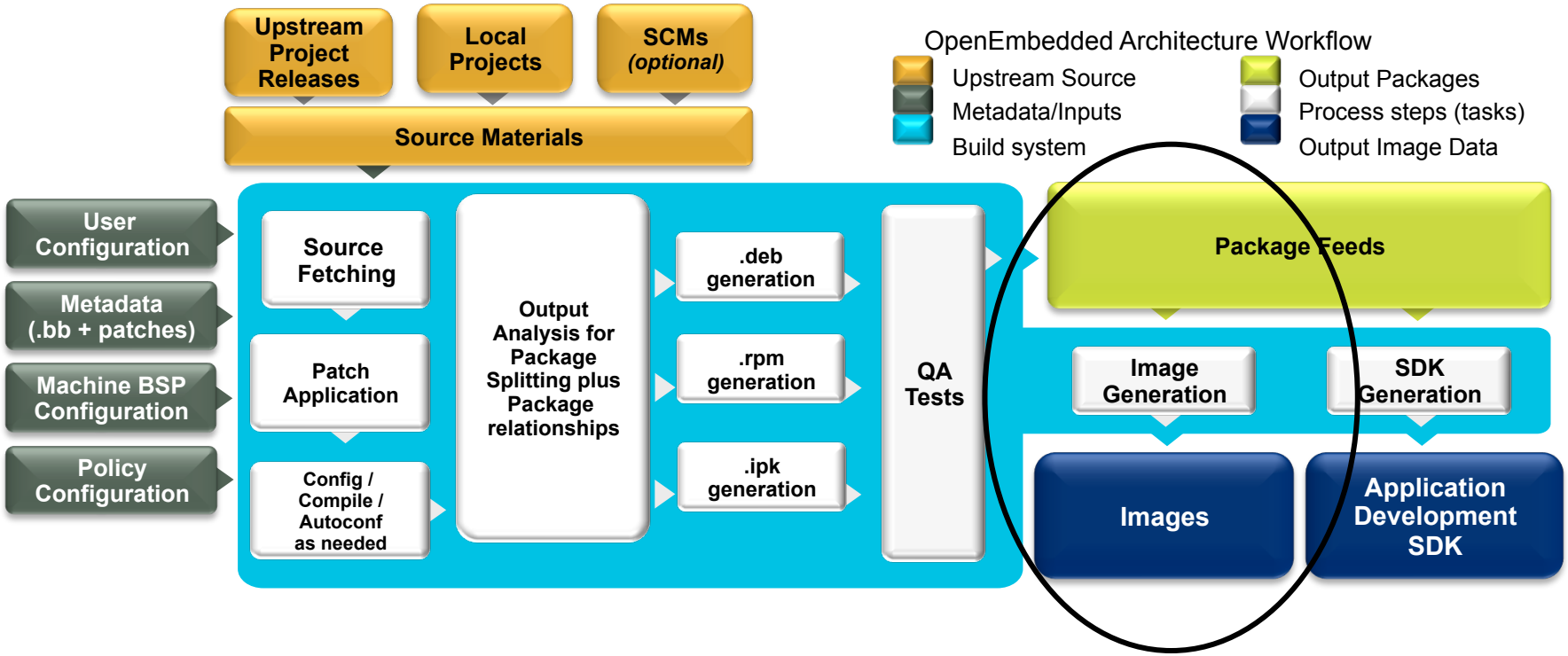
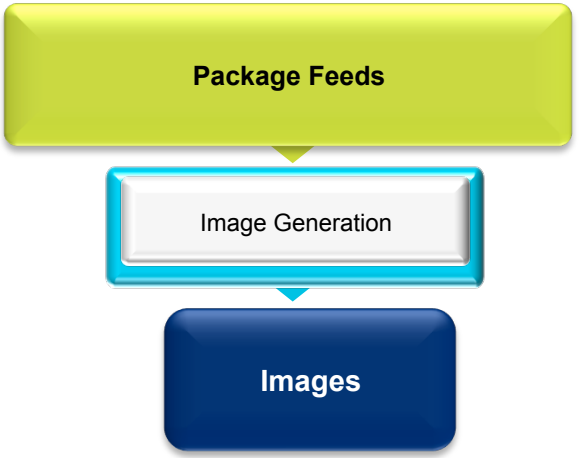


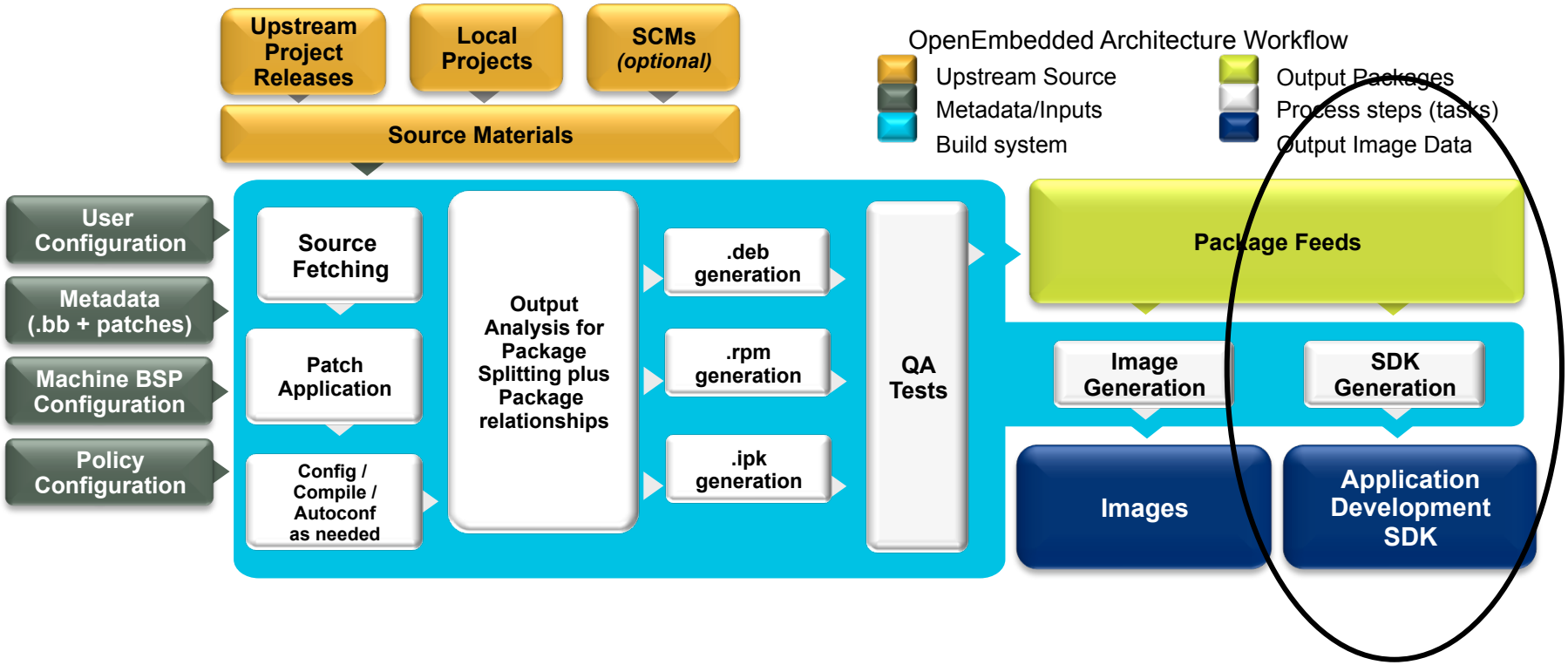
Image Generation

- Images are constructed using the packages built earlier in the process
- Decision of what to install on the image is based on a defined set of required components and automaticly resolved dependent components
- Uses for these images:
  - Live Image to boot a device
  - Root filesystem for QEMU emulator
  - Sysroot for App development



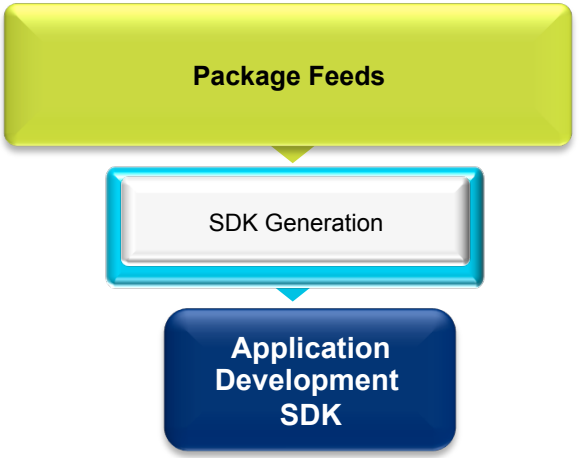
The Yocto Project allows you customize your embedded Linux OS

ADT Generation

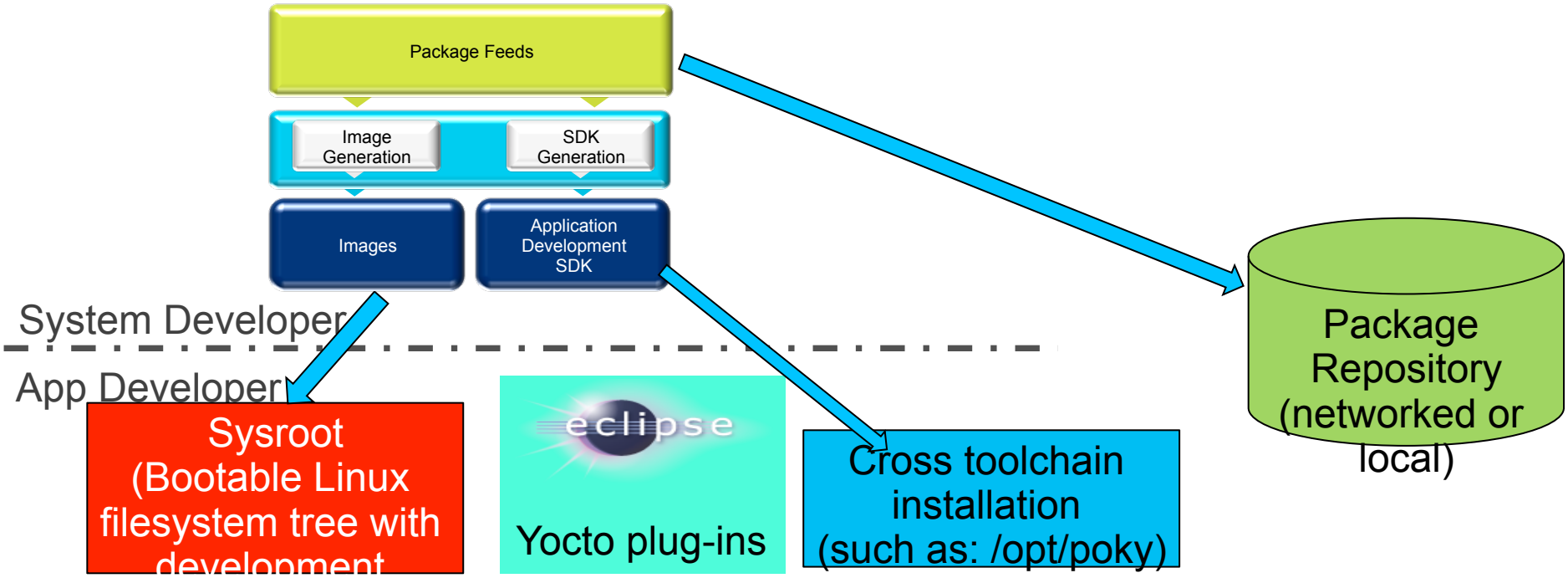


ADT Generation

- Cross toolchain and installation script generated.
- This can be used to set up an application developer's cross development environment to create apps
- QEMU built for target architecture emulation

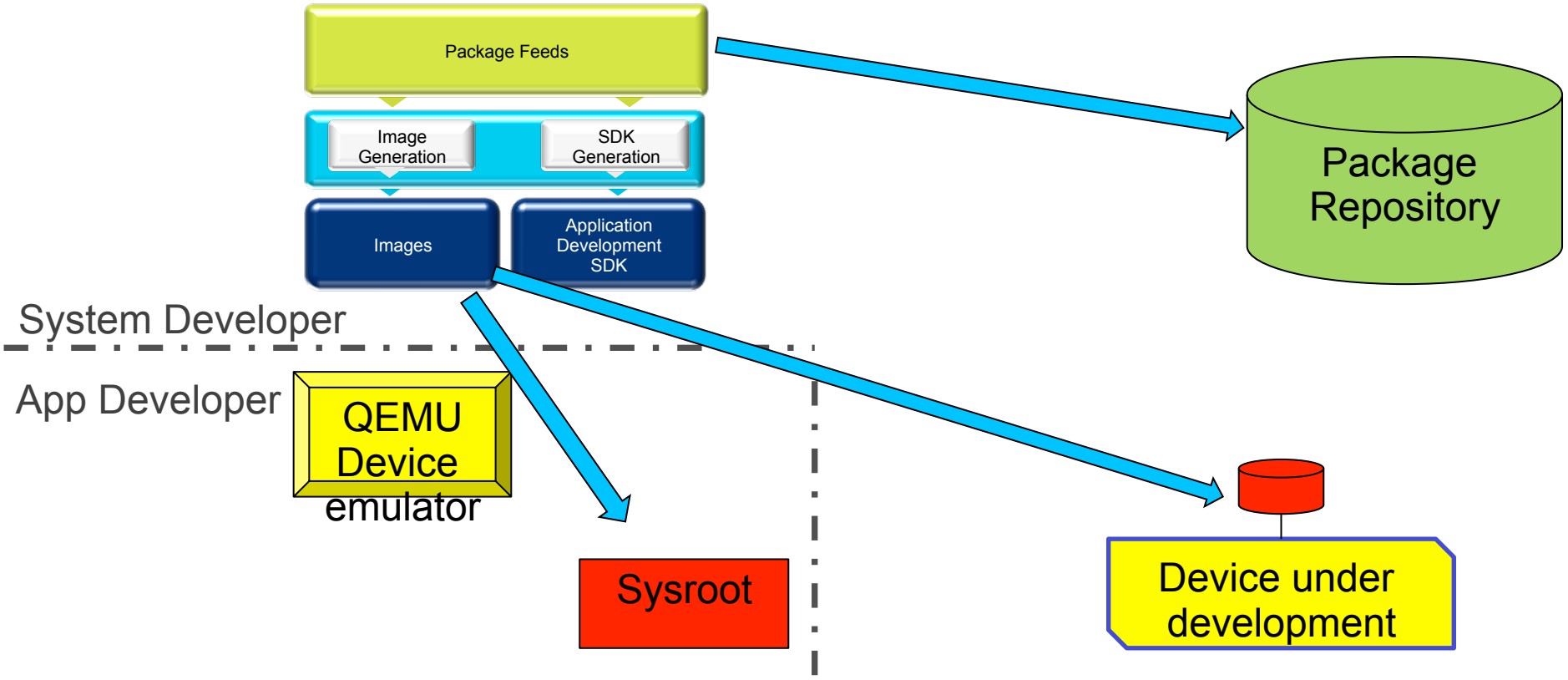


Setting up the App Developer

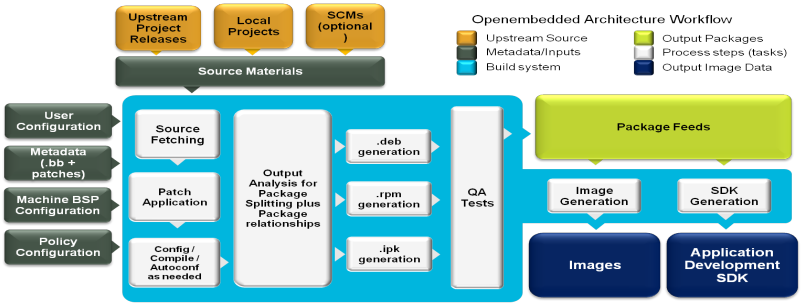


Yocto Project helps set up the embedded app developer

Use NFS/Local Disk, Pkg Manager



Use NFS/Local Disk, Pkg Manager



Both Device and App Development Models Supported

Hob v1

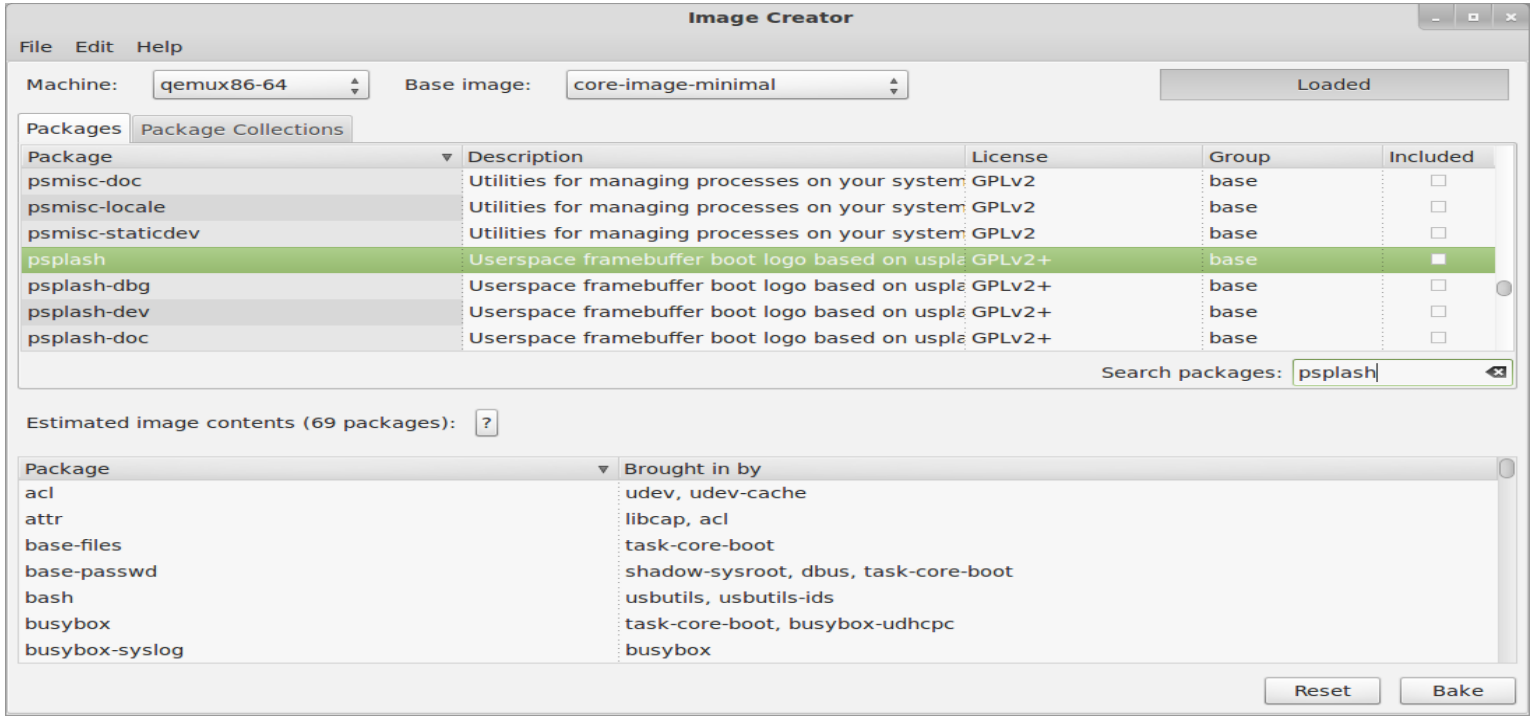


Image Creator
File Edit Help
Machine: qemux86-64
Base image: core-image-minimal
Loaded
Packages
Package Collections
Package
Description
License
Group
Included
psmisc-doc
Utilities for managing processes on your system
GPLv2
base
psmisc-locale
Utilities for managing processes on your system
GPLv2
base
psmisc-staticdev
Utilities for managing processes on your system
GPLv2
base
psplash
Userspace framebuffer boot logo based on uspla
GPLv2+
base
psplash-dbg
Userspace framebuffer boot logo based on uspla
GPLv2+
base
psplash-dev
Userspace framebuffer boot logo based on uspla
GPLv2+
base
psplash-doc
Userspace framebuffer boot logo based on uspla
GPLv2+
base
Search packages: psplash
Estimated image contents (69 packages):
?
Package
Brought in by
acl
udev, udev-cache
attr
libcap, acl
base-files
task-core-boot
base-passwd
shadow-sysroot, dbus, task-core-boot
bash
usbutils, usbutils-ids
busybox
task-core-boot, busybox-udhcpc
busybox-syslog
busybox
Reset
Bake

Hob v1

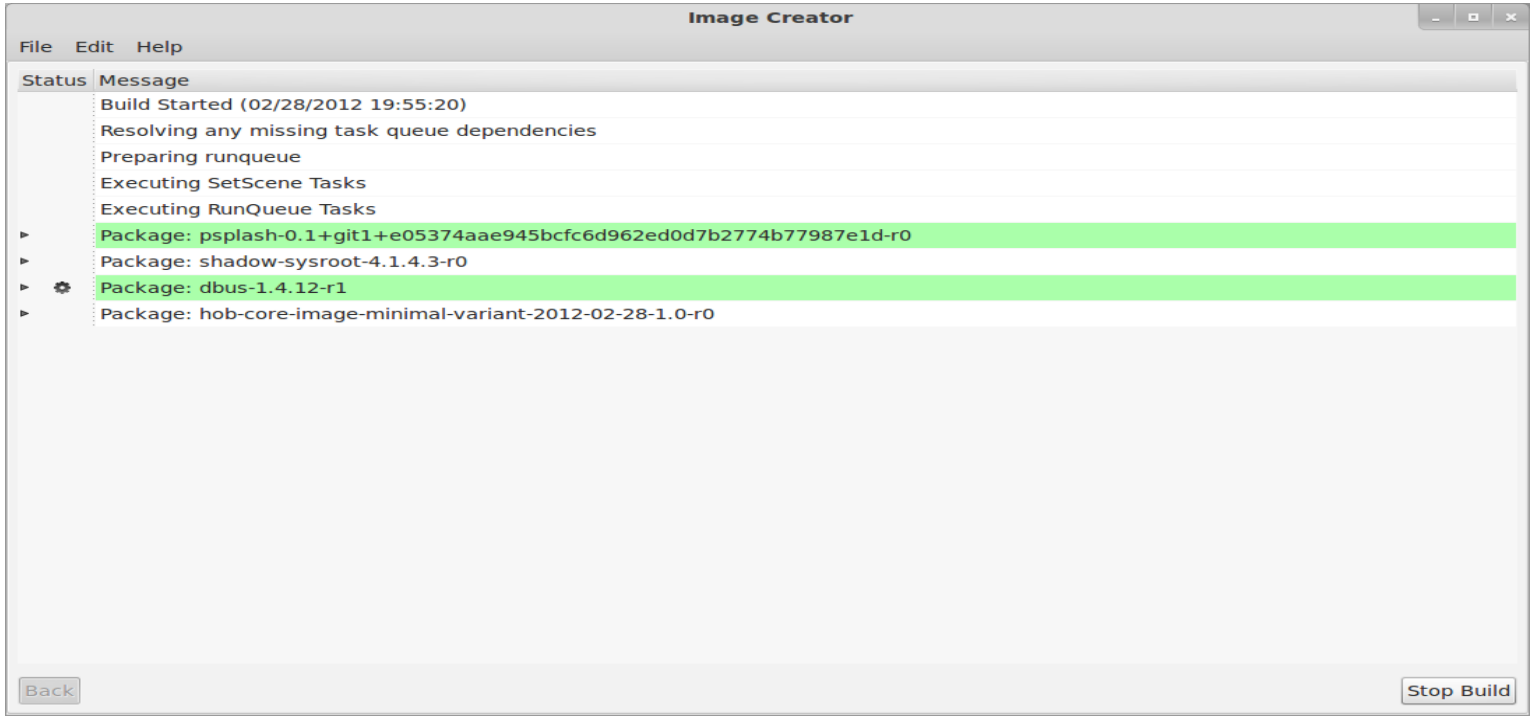


Image Creator
File Edit Help
Status Message
Build Started (02/28/2012 19:55:20)
Resolving any missing task queue dependencies
Preparing runqueue
Executing SetScene Tasks
Executing RunQueue Tasks
Package: psplash-0.1+git1+e05374aae945bcfc6d962ed0d7b2774b77987e1d-r0
Package: shadow-sysroot-4.1.4.3-r0
Package: dbus-1.4.12-r1
Package: hob-core-image-minimal-variant-2012-02-28-1.0-r0
Back
Stop Build

How to Get Started

- **Download the software today (http://www.yoctoproject.org/download)**
- Be sure you read the Quick Start to set up your system to use the Yocto Project
- Build, test on QEMU or real hardware, develop apps
- Join the community to get help
  - #yocto on freenode IRC
  - Mailing lists yocto@yoctoproject.org
    - http://lists.yoctoproject.org/listinfo/yocto

Getting started with the Yocto Project *is easy*

Get Involved

- The Yocto Project is a collaboration of individuals, non-profits, and corporations under the Linux Foundation
- We urge you or your organization to join
- yoctoproject.org/documentation/getting-started has a number of ways to learn and contribute
  - Contribute code, documentation, fix bugs, provide BSPs
  - Use YP for your embedded projects
  - Work with the community to make YP better

Make an impact – collaboration in its purest sense

It’s Time to Take Action

- It’s not an embedded Linux distribution – it creates a custom one for you
- The Yocto Project lets you customize your embedded Linux OS
- It helps set up the embedded app developer
- Both device and app development models supported
- Getting started is easy
- Make an impact – collaboration in its purest sense

Q & A

Thank You!

Enter to Win!

kindle fire
Web, movies, apps,
games, reading and more

Leave a business card or fill out a form to enter to win an Amazon Kindle Fire ($199 value)!

WIND RIVER


WIND RIVER

Additional Resources

Yocto Project:

http://www.yoctoproject.org

OpenEmbedded:

http://www.openembedded.org

Eclipse YouTube presentation:

http://www.youtube.com/watch?v=3ZlOu-gLsh0

Dirk Hohndel – LinuxCon 2011:

http://www.youtube.com/watch?v=BCxCnrcXctk