INTELLIGENT SYSTEMS
CONFERENCE &PAVILION

Introduction to the Yocto Project

Tom Zanussi

Intel

About Me

- Part of the Yocto Kernel and System-level Team
- Author and maintainer of several meta-intel BSPs
- Co-maintainer of meta-intel
- Co-author of the current 'Yocto BSP Guide'
- Author of the new 'Yocto BSP Tools'
- Author of the ‘Yocto Tracing and Profiling Manual’
- Previously worked in the kernel mainly on tracing
  - Author of kernel/relay.c (relayfs), perf scripting interface and Perl/Python bindings, perf 'live mode', kernel event filters (kernel/trace/trace_events_filter.c)
  - Major contributor to blktrace, LTT, and systemtap
- Current kernel contributor to tracing and perf systems
  - Most recently ‘trace event triggers’
- Created systemtap, blktrace, sysprof recipes
- Other odds and ends related to BSPs, kernel, and tracing

Agenda

- What is the Yocto Project?
- Who is the Yocto Project?
- Poky (Bitbake/OpenEmbedded-Core)
  - Getting Started
  - Build Configuration
  - Build System Workflow
- The Intel/Yocto Minnow Board
  - Example BSP and Application Layers
- Questions

What is the Yocto Project?

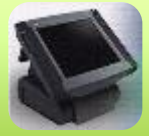
The Yocto Project is an open source collaboration project that provides templates, tools, and methods to help you create custom Linux*-based systems for embedded products, regardless of the hardware architecture.

- For “roll your own” OS developers as well as companies with multiple embedded product lines
- Quickly growing user base and industry leader participation
- Helps control adherence to Open Source Licensing thru Filtering and Manifest creation and archiving
- Enables easy transition from proof of concept (POC) to supported commercial Linux (Wind River)
- Provides common format/repository for Linux* Board Support Packages (BSPs) for easy porting, sorted by Architecture
- Generates a custom designed application development kit for each specific device
- Simple to port across architectures – “1 line in config”

Simple Electronics

M2M

Point of Sale



Networking & Storage

Industrial

It’s Not an Embedded Linux Distribution, it Creates a Custom One for You;
Learn More at www.yoctoproject.org

What is the Value on Intel® Architecture?

Intel is a major contributor to the Yocto Project because of the value it brings to Intel customers.

- BSPs for all Intel® Atom™, Core™ and Xeon™ based platforms and processors in the Intel® Architecture Embedded Roadmap
- BSPs demonstrate to users how to standardize and best deliver Intel technologies on IA based BSPs, for example: compiler flags for performance optimization, codec libVA support, and Intel graphics acceleration supports
- Intel-fostered technologies such as WayLand and systemd integrated into the project
- Sub $200 Open Hardware/Software Intel® Architecture based MinnowBoard* for an enabling vehicle

It's Not an Embedded Linux Distribution, it Creates a Custom One for You;
Learn More at www.yoctoproject.org

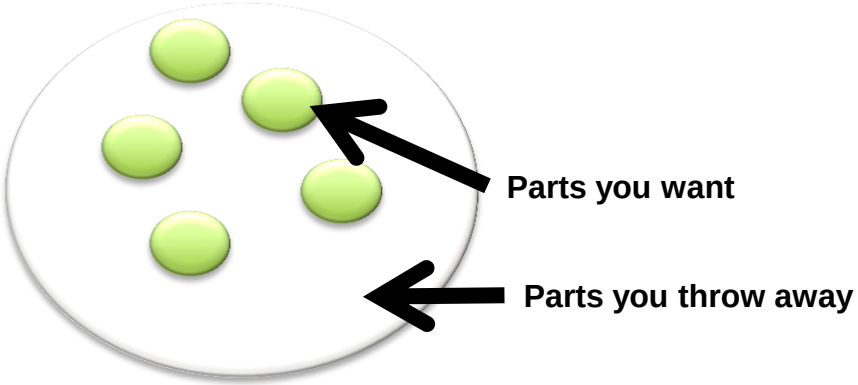
Examples of Who Could Benefit from Using Yocto Project

- Anyone interested in running one software package across a diverse set of devices or platforms
  - Performance analysis comparisons
- Companies with product lines that span multiple hardware architectures or devices
  - Use a base software model that is added to for more feature rich product models (build recipe reuse)
- Those interested in automating the difficult task of package specification and controlling open source type license use
- Companies with an exit strategy of being acquired who need a complete report out of open source software use for risk analysis
- Corporations with a Linux competency center who want to own the base Linux definition and update
  - Whose product groups then build the stack on top of that kernel who work and share within Yocto Project Layers

Most Distributions Used are Not Suited for Embedded

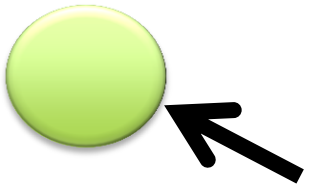
Existing Distributions

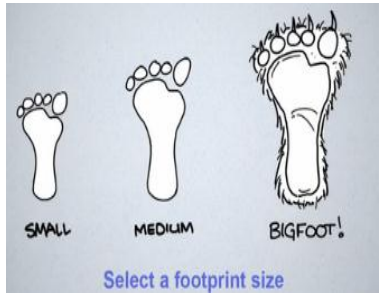
(Enterprise or Desktop)



- Hack away pieces that aren't needed
- Add missing packages
- Rebuild
- Debug
- Test
- Tune

Yocto Project





- Build profiles—in multiple sizes
- Package choice tuned for chosen build profile
- Used by commercial Linux OS vendors as their upstream

The State of Embedded Linux…

- DIY/Roll-Your-Own or modified Enterprise/Desktop distro:
  - Long Term Maintenance is difficult
  - Upstream changes are difficult to track
  - Not embedded friendly
  - Licensing issues
  - No commercial embedded support
- Commercial/Community Embedded Linux:
  - Too many competing systems
  - Incompatible distributions/build systems
- *Much Fragmentation: developers spend lots of time porting or making build systems*
- *Leaves less time/money to develop interesting software features*

What the Yocto Project Provides

- The industry needed a common build system and core technology
  - Bitbake and OpenEmbedded build system
- 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 (app development, product development, …)*

Who is involved in 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

THE LINUX FOUNDATION

Advisory Board

Texas Instruments, Sakoman Inc, Long Term Support Initiative, Wind River, Huawei, Enea software, Juniper Networks, LSI, Mentor Graphics, intel, montavista, openembedded

The Project also has a growing Compliance Program

Yocto Project Compliance Program Goals

- Reduce fragmentation in the embedded market by encouraging collaborative development of a common set of tools, standards, and practices.
- Ensure that these tools, standards, and practices are architecturally independent, as much as possible.
- Encourage Interoperability and Contributions.

Yocto Project Compliance Program

Yocto Project Registered Compatible Products

- The Arago Project
- The Angstrom Distribution
- Enea* Linux*
- Fish River Island 2 Board (Kontron* M2M Device)
- Intel® System Studio for Linux* 2013
- Mentor Graphics* Embedded Linux
- Meta-intel layer (all Intel BSPs are Compatible)
- MinnowBoard
- Wind River* Linux 5

Yocto Project Registered Participants

- Minnowboard.org
- NetModule AG
- Move Innovation
- DENX Software Engineering
- ChargeStorm AB
- Qtechnology
- KOAN
- Long Term Support Initiative (LTSI)
- The Angstrom Distribution
- Sidebranch
- Juniper Networks
- O.S. Systems
- Huawei
- Gumstix
- Mentor Graphics
- Texas Instruments
- Sakoman, Inc.
- OpenEmbedded eV
- MontaVista Software
- LSI Corporation
- Intel Corporation
- Enea AB
- Wind River Systems

New Participants are applying weekly

Why Should a Developer Care?

- Build a complete Linux system –from source– in about an hour (about 90 minutes with X).
  - Bitbake and OpenEmbedded build system
  - Filter for license versions (e.g., GPLv3)
- 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.

Why Should a Developer Care?

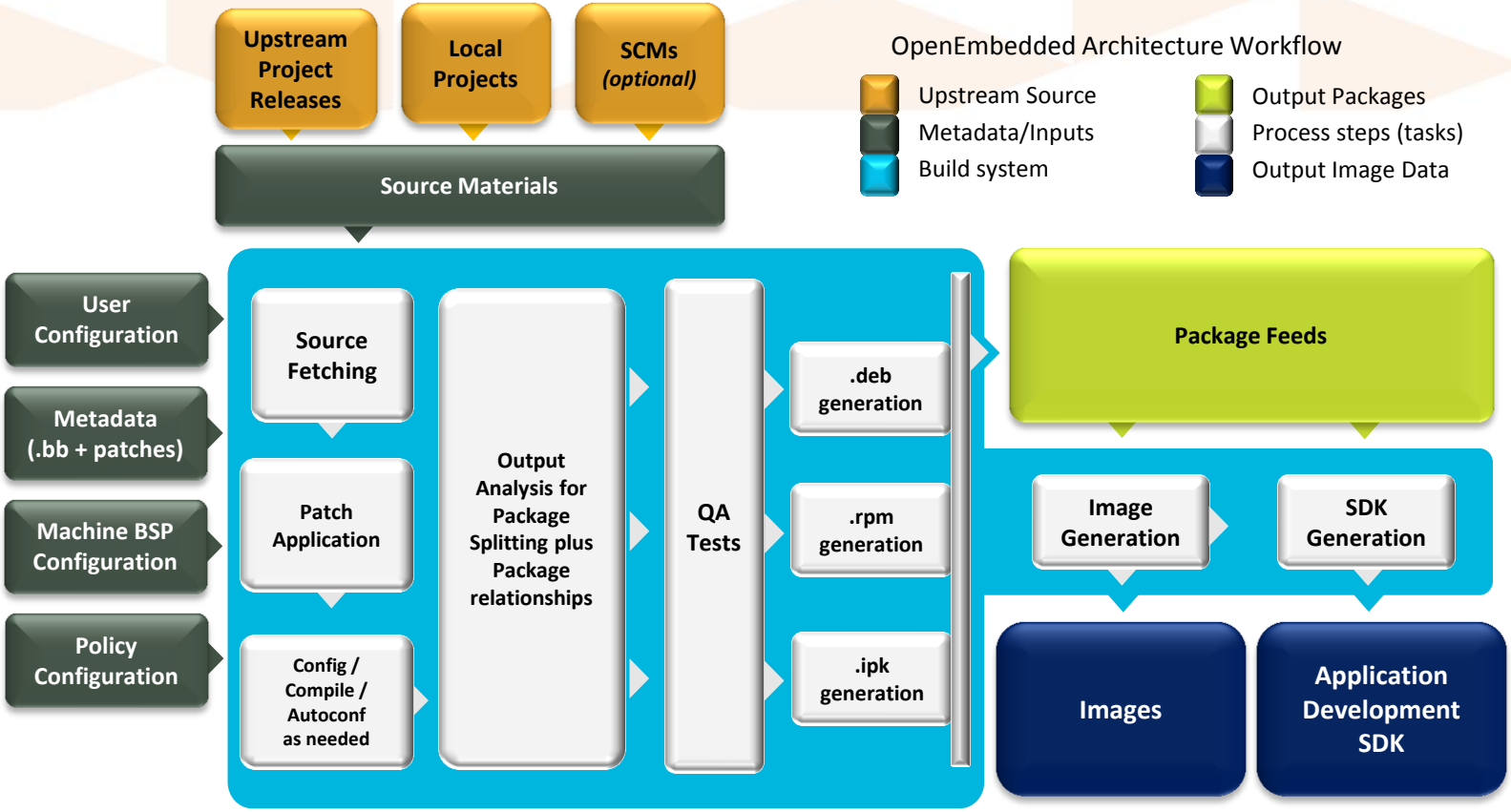
- Supports all major embedded architectures (x86, x86-64, ARM, PPC, MIPS), just change a line in a configuration file and rebuild.
- High level build interface (HOB) and tools (Eclipse Plug-in)
- Advanced kernel development tools.
- Layer model encourages modular development, reuse, and easy customizations.
- Compatibility program that is used to encourage interoperability and best practices.

Enough about the Project, what about using it?

Quick Start

1. Go to http://yoctoproject.org click "documentation" and read the Quick Start guide
2. Set up your Linux build system with the necessary packages (and firewall as needed)
3. Go to http://yoctoproject.org click "download" and download the latest stable release (9.0.0 "Dylan") – extract the download on your build machine
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)

Build System Workflow



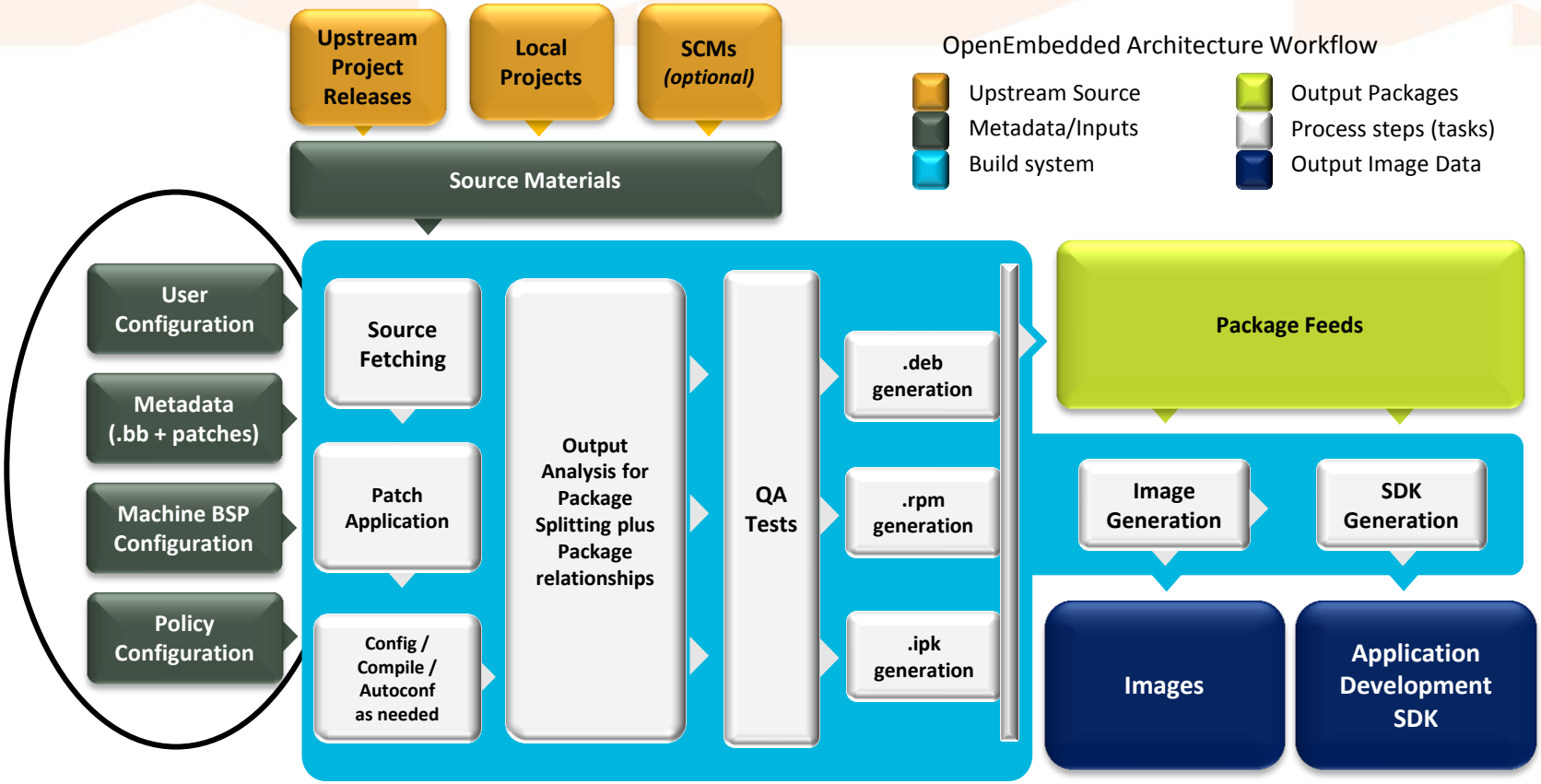
Layers

- Layers are a way to manage extensions, and customizations to the system
  - Layers can extend, add, or modify recipes
  - Recipes comprise metadata defining packages
  - Layers can add or modify configurations
  - Layers are added via BBLAYERS in build/conf/bblayers.conf
- Common Examples of Layers
  - Custom Toolchains (compilers, debuggers, profiling tools)
  - Distribution specifications (i.e. meta-yocto)
  - BSP/Machine settings (i.e. meta-yoct-bsps)
  - Functional areas (selinux, networking, etc)
  - Project-specific changes

Layers (cont'd)

- Layers are the building blocks used to construct the system
- Best Practice: Contents of a layer should be grouped by functional components

It All Starts with Configuration



Configuration

User Configuration

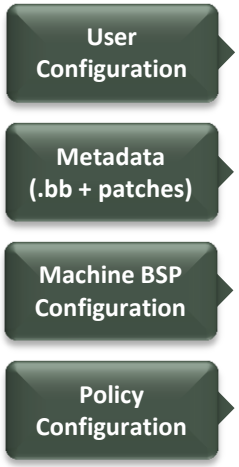
Metadata (.bb + patches)

Machine BSP Configuration

Policy Configuration

- Configuration files (*.conf) – global build settings
  - Normally variable settings of the form x = y, x ?= y, …
  - meta/conf/bitbake.conf (default settings)
  - */conf/layers.conf (one per layer)
  - meta-yocto/conf/distro/poky.conf (distro policy)
  - build/conf/local.conf (user-defined overrides)
    - MACHINE ?= “beagleboard”
    - EXTRA_IMAGE_FEATURES += “tools-profile”
  - meta-yocto-bsp/conf/machine/beagleboard.conf (BSP-specific configuration)
- build/conf/bblayers.conf is where you configure which layers to use
  - Add Yocto Project Compatible layers to BBLAYERS
  - Default: meta (oe-core), meta-yocto, meta-yocto-bsp

Metadata (Recipes)



- Metadata and patches:
  - Recipes are for building packages
  - Recipe:
    - meta/recipes-core/busybox_1.20.2.bb
  - Patches:
    - meta/recipes-core/busybox/busybox-1.20.2/*.patch
- Recipes inherit the system configuration and adjust it to describe how to build and package the software
- Can be extended and enhanced via layers
  - Recipe modification (in another layer):
    - meta/recipes-core/busybox_1.20.2.bbappend
- Compatible with OpenEmbedded

Machine (BSP) Configuration

User Configuration

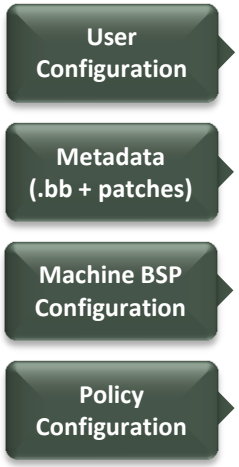
Metadata (.bb + patches)

Machine BSP Configuration

Policy Configuration

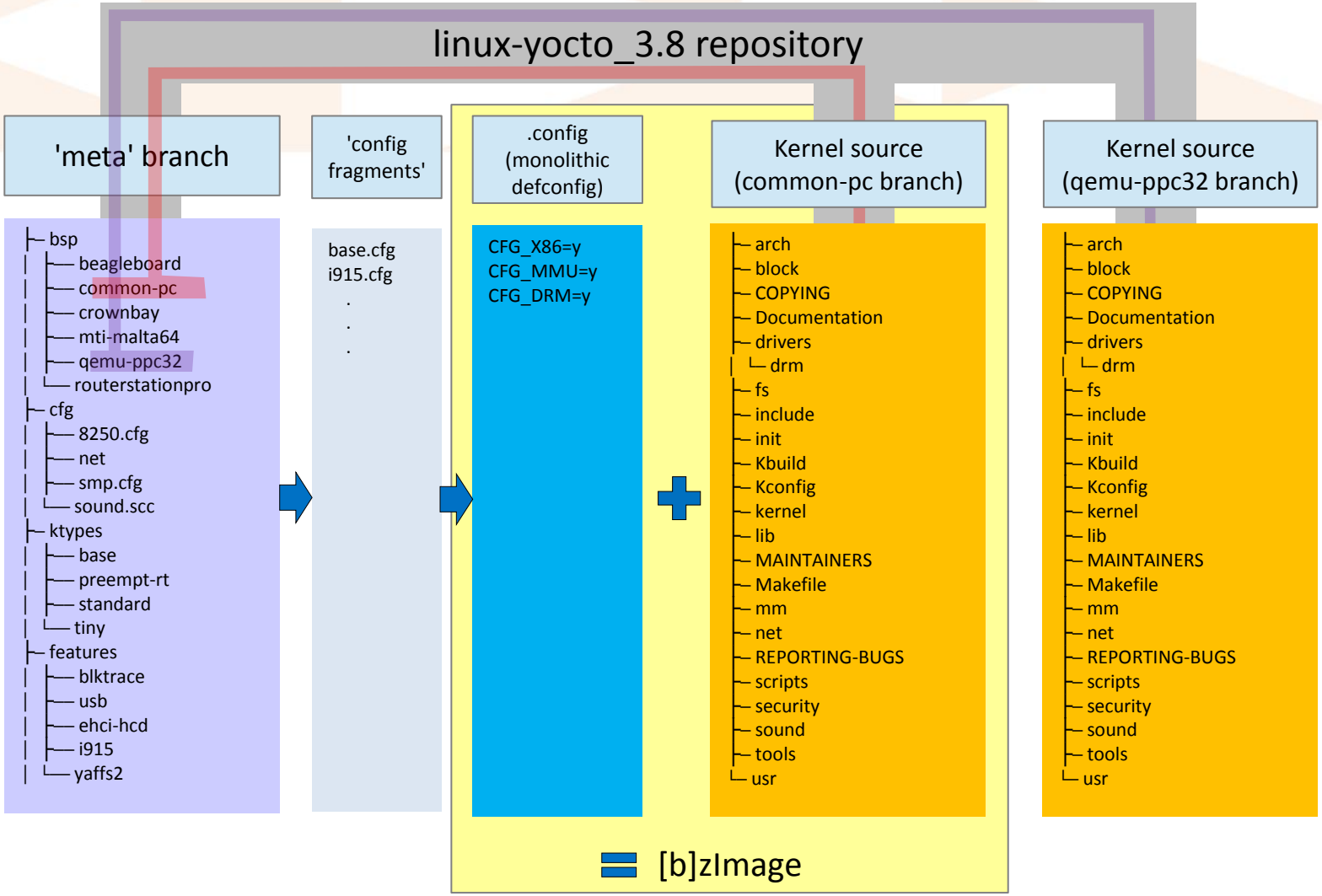
- Configuration files that describe a machine
  - Define board specific kernel configuration
  - Formfactor configurations
  - Processor/SOC Tuning files
- Examples:
  - meta-yocto-bsp/conf/machine/beagleboard.conf
  - meta-intel/meta-minnow/conf/machine/minnow.conf
- Best Practices:
  - Manage BSPs in layers
  - Use the Yocto Project kernel tooling (but not required)
  - Follow the guidelines specified in the Yocto Project BSP Developer's Guide:
    - http://www.yoctoproject.org/docs/current/bsp-guide/bsp-guide.html

Yocto Project Kernels

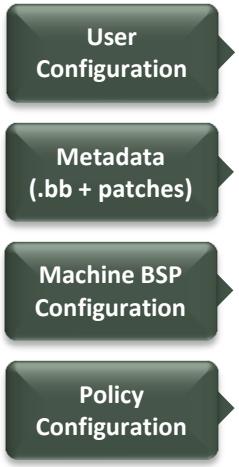


- linux-yocto recipes point to 'Yocto kernel repos‘
- 'Yocto kernel repos' are based on upstream kernels
  - kernel.org linux/linux-stable, ltsi-kernel, etc
  - Patches are temporary and expected to go upstream
  - Unless they're BSP-specific
- Yocto simply adds machinery and metadata on top
- Two major kernel advances in the Yocto Project:
  - Kernel Features: patches and configuration fragments managed as functional blocks (supports reuse)
  - Branching Tools: Per-BSP git branches define machine-specific kernel sources. Tools collect the relevant kernel features to produce kernel sources
- Results:
  - Can toggle collections of features for a given BSP
  - Reuse – less code duplication

Yocto Project Kernels (cont'd)

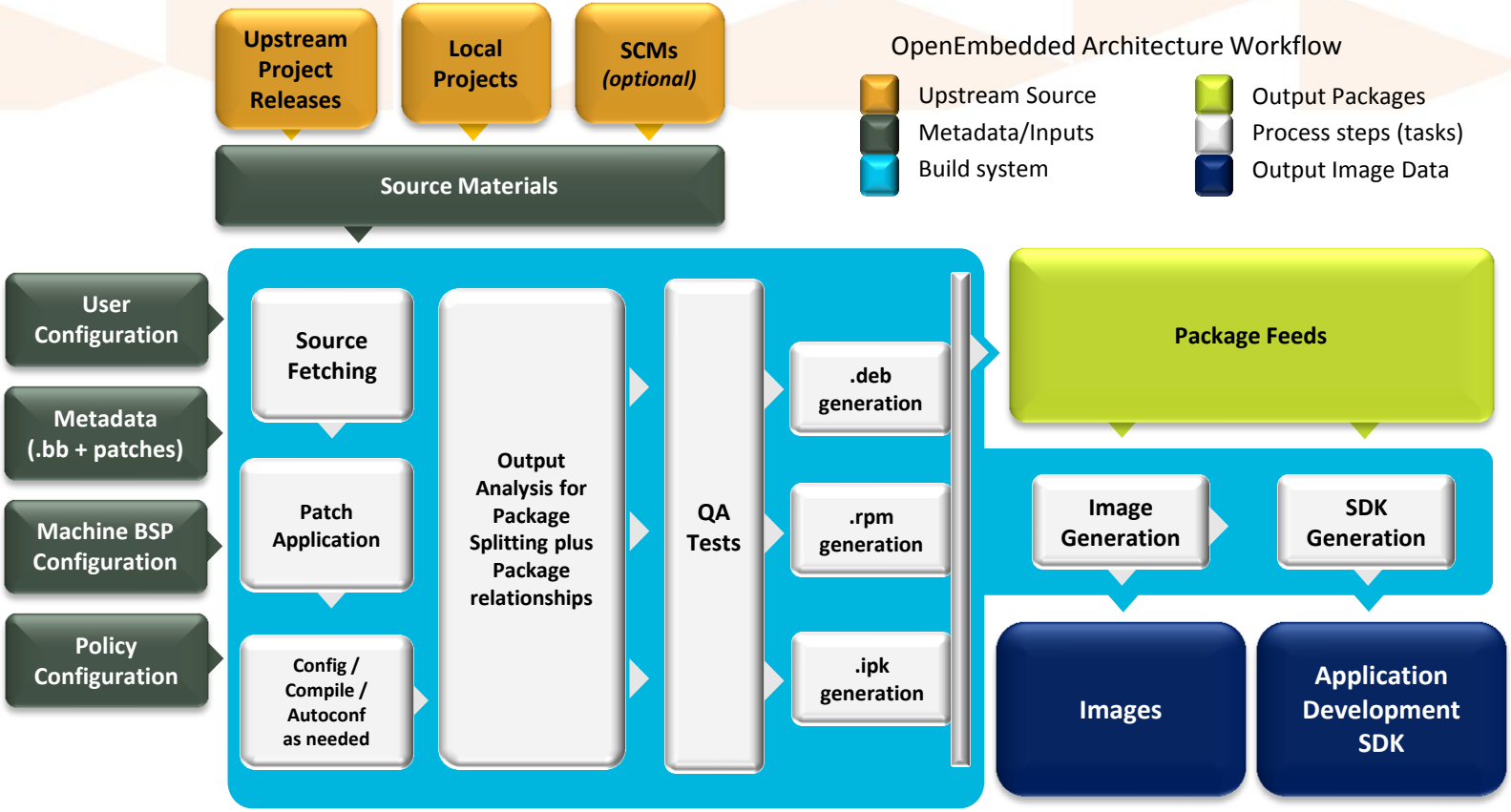


Distro Policy

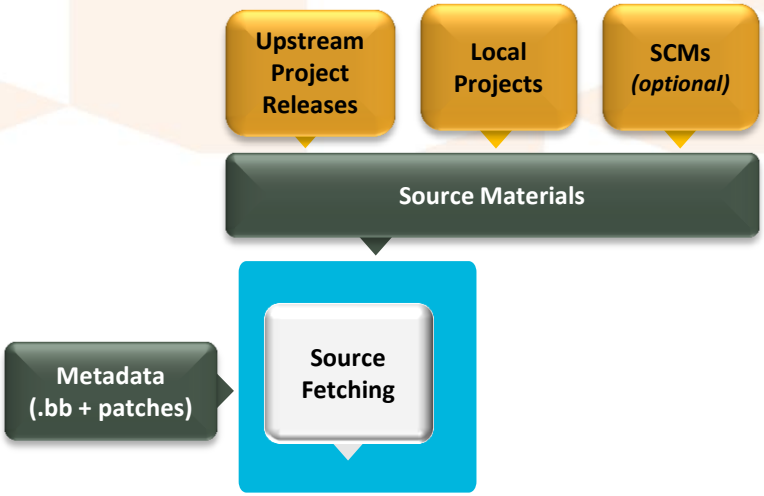


- Defines distribution/system wide policies that affect the way individual recipes are built
  - May set alternative preferred versions of recipes
  - May enable/disable LIBC functionality (i.e. i18n)
  - May enable/disable features (i.e. pam, selinux)
  - May configure specific package rules
  - May adjust image deployment settings
- Enabled via the DISTRO setting
- Four predefined settings:
  - poky: Core distribution definition, defines the base
  - poky-bleeding: Enable a bleeding edge packages
  - poky-lsb: enable items required for LSB support
  - poky-tiny: construct a smaller than normal system

The Build Process In-depth

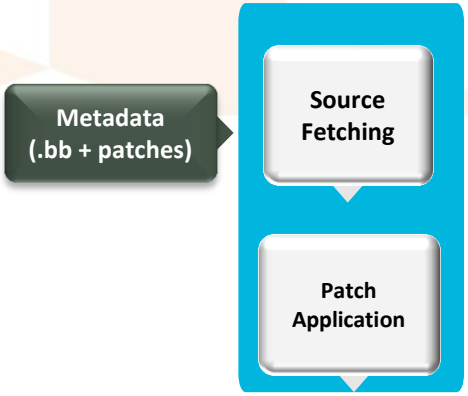


Source Fetching



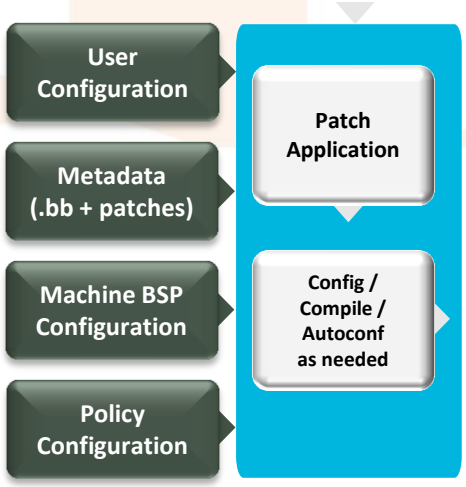
- Recipes call out the location of all sources, patches and files. These may exist on the internal or be local. (See SRC_URI in the *.bb files)
- Bitbake can get the sources from git, svn, bzr, tarballs, and many more
- Versions of packages can be fixed or updated automatically (Add SRCREV_pn-PN = "${AUTOREV}" to local.conf)
- Yocto Project mirrors sources to ensure source reliability

Patching



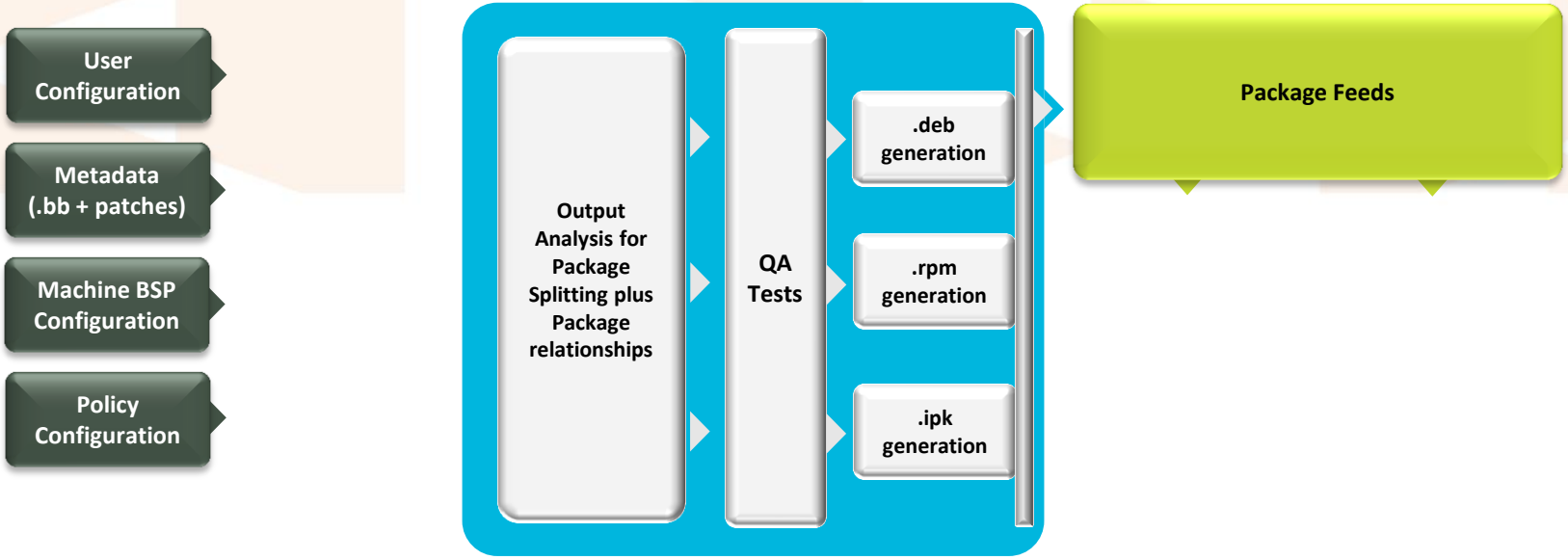
- Once sources are obtained, they are extracted
- Patches are applied in the order they appear in SRC_URI
  - quilt is used to apply patches
- This is where local integration patches are applied
- We encourage all patch authors to contribute their patches upstream whenever possible

Configure / Compile / Install



- Recipe specifies configuration and compilation rules
  - Various standard build rules are available, such as autotools and gettext
  - Standard ways to specify custom environment flags
  - Install step runs under ‘pseudo’, allows special files, permissions and owners/groups to be set

Output Analysis / Packaging



- Output Analysis:
  - Performs QA checks
  - Categorize generated software (debug, dev, docs, locales)
  - Split runtime and debug information
- Package Generation:
  - Support the popular formats, RPM, Debian, and ipk
  - Set preferred format using PACKAGE_CLASSES in local.conf
  - Package files based on manual settings and categorized software:

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

Image Generation

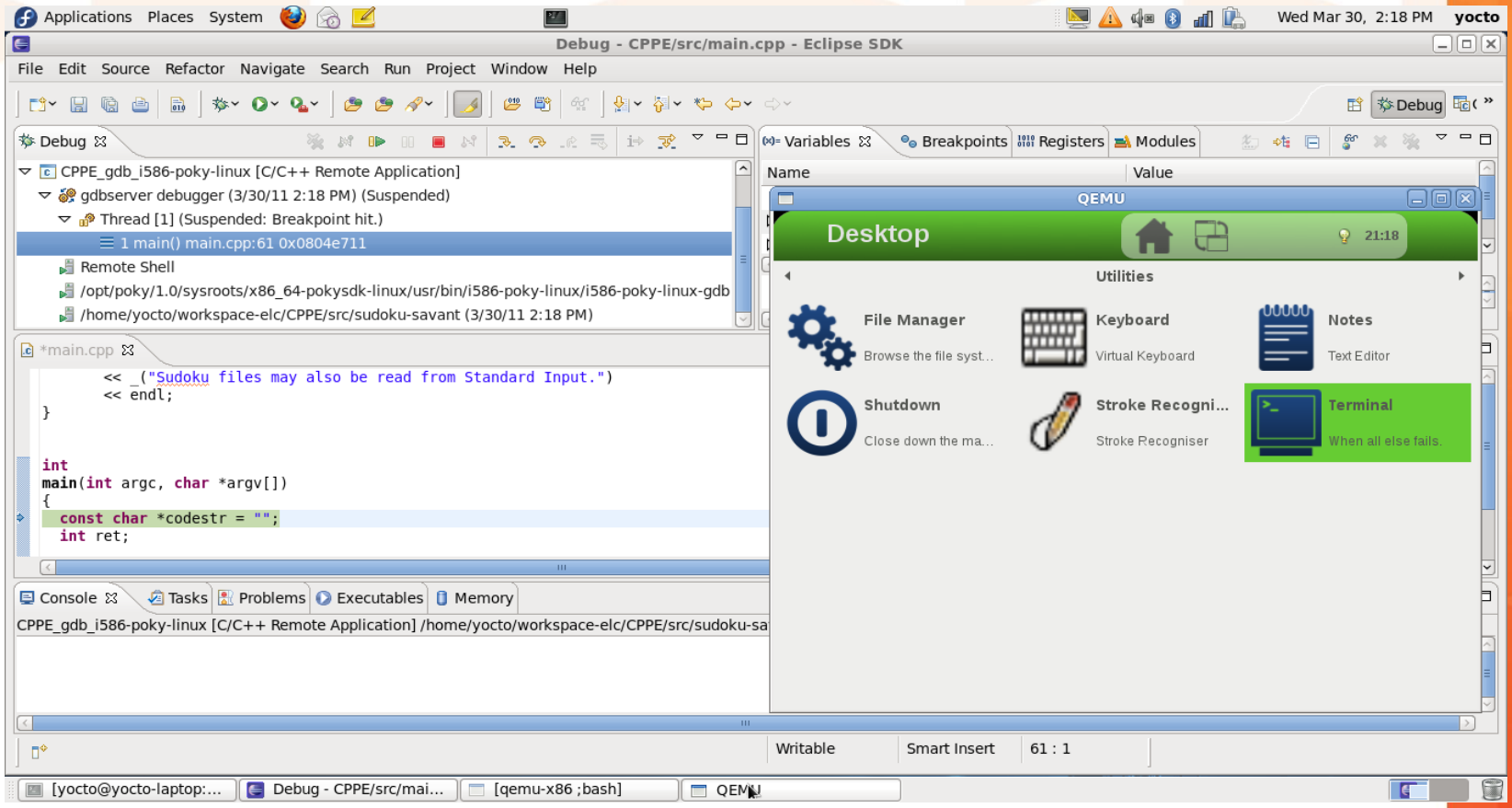
- Images are constructed using the packages built earlier and put into the Package Feeds
- Decisions of what to install in the image is based on the defined set of components in the image recipe, expanded to include all dependencies.
- Images may be generated in a variety of formats (tar.bz2, ext2, ext3, jffs, etc)

SDK Generation

- A specific SDK recipe may be created, which allows a user to build an SDK with specific interfaces in it. (e.g. meta-toolchain-gmae)
- An SDK can be created based on the contents of the image generation
- The SDK contains native applications, cross toolchain and installation scripts
- May be used by the Eclipse Yocto Application Developer Tool to enable App Developers
- May contain a QEMU target emulation to assist app developers

IDE Support: Eclipse

Eclipse plug-in and user space tool suite offerings further improve developer productivity



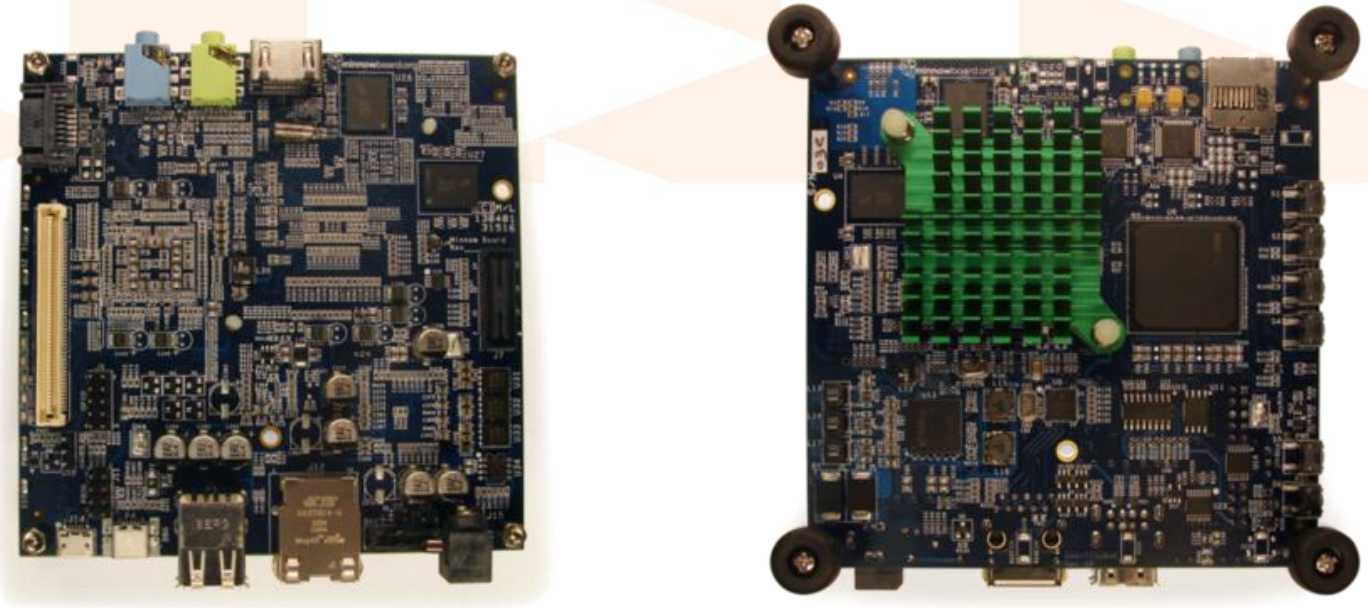
An example Yocto BSP and Application (Minnow Robot Arm)

The MinnowBoard

- The MinnowBoard is an Intel® Atom™ - based board which introduces the Intel Architecture to the small and low cost embedded market for the developer and maker community. It has exceptional performance, flexibility, openness and standards for the price.
- Both the board and the BSP were developed by the Yocto Project Team

minnowboard

The MinnowBoard (cont'd)



- Intel® Atom™ 1.0 GHz CPU with Hyper-Threading and Virtualization technology
- Generous I/O powered by PCI Express:
  - SATA, USB
  - Gigabit Ethernet
- SPI, I2C, CAN, GPIO – Embedded system standards
- UEFI firmware with Fast Boot (expected July 2013)

The MinnowBoard (cont'd)

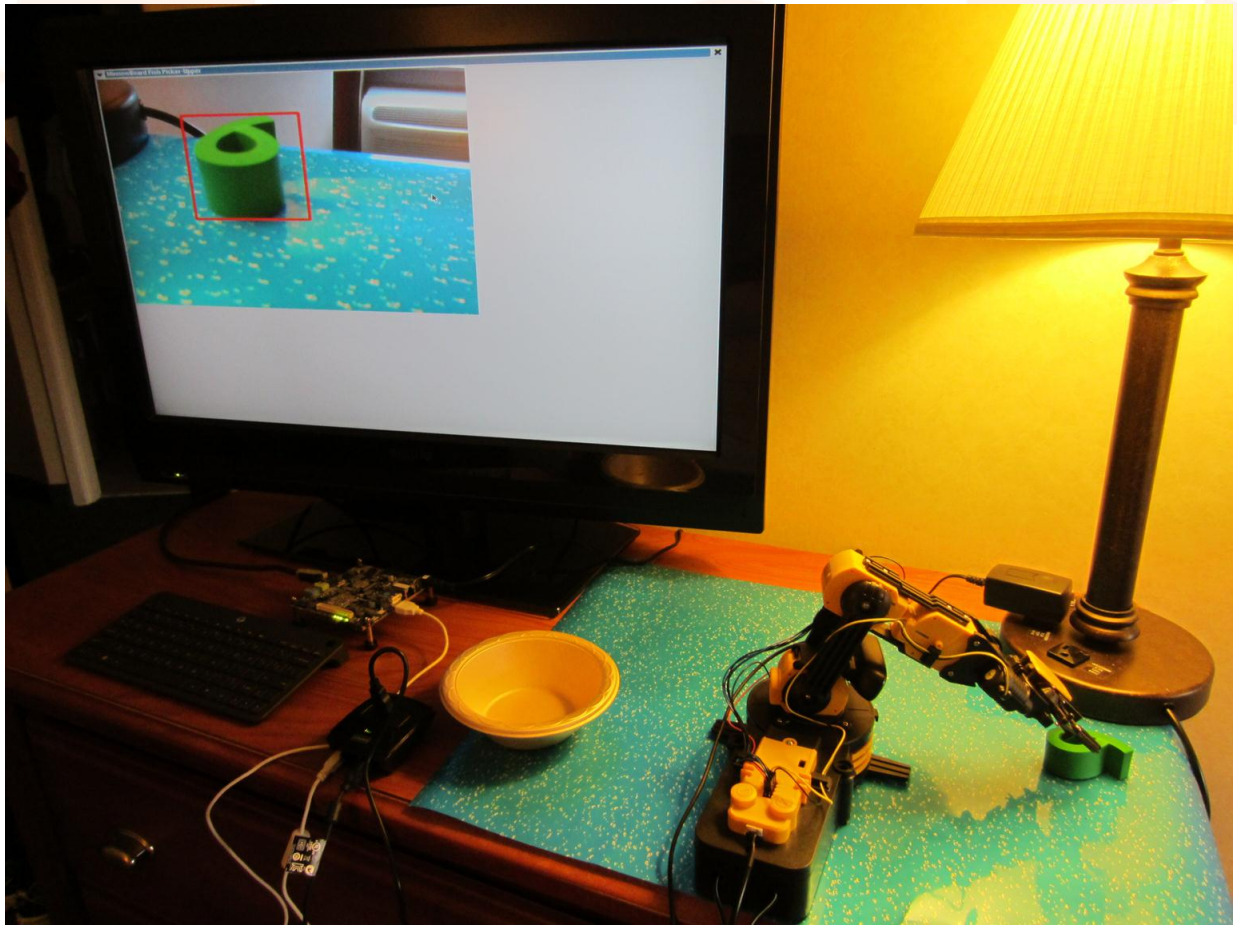
- Affordable Intel® Atom™ platform
  - $199 MSRP
- Scales up to higher workloads
- Small form factor
  - 4.2"x4.2"
- Extensive firmware capabilities
  - UEFI Development Platform
  - Modern, standards-based firmware environment
  - Develop & debug your own firmware
- Open Source hardware platform
  - Customizations possible without signing NDAs
- Stackable and Expandable via Open Source MinnowBoard 'Lures'
  - Add displays, wireless, more I/O options

MinnowBoard Info

- Visit our website at www.minnowboard.org and be the first to know when the board is available for purchase
  - Initial board shipments expected around June 12
- @minnowboard on Twitter
- MinnowBoard on Google Plus

minnowboard.org

The Minnow Robot Arm App



The Minnow Robot Arm App

- Webcam mounted to robot arm rotating base used to scan and identify foam “fish”
- Arm centers on and picks up desired fish
- Moves the fish to a dinner plate
- A fun & family-friendly demo that relates to a serious embedded application space
- Affordable (robot arm < $100)
- Developed by Scott Garman, Minnow lead

The Minnow Robot Arm App

- Uses OWI 535 Robotic Arm
- Also uses the OpenCV Library
  - Open Source (BSD licensed) computer vision and machine learning library started by Intel in 1999
  - More than 2500 algorithms in object detection and classification, motion tracking, complex image processing
  - Cross-platform, offers C, C++, Python, and Java interfaces

But First, the Minnow BSP

- We need a machine-specific BSP for the Minnow in order to run the Robot App on the Minnow hardware
- meta-minnow is the board-specific metadata layer for the MinnowBoard
- meta-minnow lives in the meta-minnow git repository:
  - http://git.yoctoproject.org/cgit/cgit.cgi/meta-minnow

meta-minnow BSP metadata

- Standard Yocto BSP layout

```
meta-minnow
├── conf
│   ├── layer.conf
│   └── machine
│       ├── minnow.conf
│       └── README
├── COPYING.MIT
├── README
├── README.sources
├── recipes-bsp
│   ├── alsa-state
│   │   ├── alsa-state
│   │   │   └── minnow
│   │   │       ├── asound.state
│   │   │       └── README
│   │   └── alsa-state.bbappend
│   ├── formfactor
│   │   ├── formfactor
│   │   │   └── minnow
│   │   │       └── machconfig
│   │   └── formfactor_0.0.bbappend
│   └── gummiboot
│       ├── files
│       │   ├── 0001-configure.ac-Add-option-to-disable-configuring-the-B.patch
│       │   └── 0002-Add-32-bit-compatible-rdtsc-asm.patch
│       └── gummiboot_git.bb
```

meta-minnow (cont'd)

```
├── recipes-core
│   ├── dmidecode
│   │   └── dmidecode_2.11.bb
│   └── tiny-init
│       ├── tiny-init
│       │   └── init
│       └── tiny-init.bbappend
├── recipes-graphics
│   └── xorg-xserver
│       ├── xserver-xf86-config
│       │   └── minnow
│       │       └── xorg.conf
│       └── xserver-xf86-config_0.1.bbappend
├── recipes-kernel
│   └── linux
│       ├── linux-yocto-minnow
│       │   ├── media.cfg
│       │   └── user.cfg
│       └── linux-yocto-minnow_3.8.bb
└── TODO

23 directories, 27 files
```

meta-minnow BSP metadata

- meta-minnow/conf/machine/minnow.conf

```
#@WEBTITLE: Intel Atom E640T Processor with Intel EG20T Controller Hub
 Development Kit (Queens Bay) with Proprietary IEMGD Accelerated Graphics.

#@DESCRIPTION: Machine configuration for the Minnow Board systems

require conf/machine/include/tune-atom.inc
require conf/machine/include/ia32-base.inc
require conf/machine/include/meta-intel.inc

MACHINE_HWCODECS ?= "va-intel"
XSERVERCODECS ?= "emgd-driver-video emgd-gst-plugins-va gst-va-intel"

MACHINE_FEATURES += "efi va-impl-mixvideo"

PREFERRED_PROVIDER_virtual/kernel ?= "linux-yocto-minnow"
PREFERRED_VERSION_linux-yocto-minnow = "3.8%"

XSERVER ?= "${XSERVER_IA32_BASE} ${XSERVER_IA32_EXT} ${XSERVER_IA32_EMGD}"

PREFERRED_VERSION_xserver-xorg ?= "1.9.3"

APPEND += "console=ttyPCH0,115200 console=tty0 vmalloc=256MB"

# Linux kernel drivers for onboard hardware
MACHINE_ESSENTIAL_EXTRA_RRECOMMENDS += " kernel-module-snd-hda-intel \
  kernel-module-pch-gbe kernel-module-gpio-sch kernel-module-minnowboard"
```

meta-minnow BSP metadata

- minnow.conf selected the linux-yocto 3.8 kernel
- Our 3.8 minnow kernel modifications are in meta-minnow/recipes-kernel/linux/linux-yocto-minnow_3.8.bb

```
require recipes-kernel/linux/linux-yocto.inc

KBRANCH_DEFAULT = "standard/minnow"
KBRANCH = "${KBRANCH_DEFAULT}"
KMETA = "meta"

SRC_URI = "git://git.infradead.org/users/dvhart/linux-yocto-minnow-3.8; \
  protocol=git;nocheckout=1;branch=${KBRANCH},${KMETA},emgd-1.16; \
  name=machine,meta,emgd \
           file://media.cfg file://user.cfg"

LINUX_VERSION ?= "3.8.4"

COMPATIBLE_MACHINE = "minnow"
KMACHINE_minnow  = "minnow"

# Functionality flags
KERNEL_FEATURES_minnow_append = " features/drm-emgd/drm-emgd-1.16 \
                                  features/netfilter/netfilter.scc"

# Autoload modules for on-board hardware
module_autoload_snd_hda_intel = "snd_hda_intel"
module_autoload_pch_gbe = "pch_gbe"
module_autoload_minnowboard = "minnowboard"
```

meta-minnow BSP metadata

- We selected the minnow machine by adding the following to our build/conf/local.conf:

```
MACHINE ?= "minnow"
LICENSE_FLAGS_WHITELIST = "license_emgd-driver-bin commercial"
```

- We told the build system to add the meta-minnow layer in our build/conf/bblayers.conf:

```
BBLAYERS ?= " \
  /home/trz/yocto/minnow-demo/meta \
  /home/trz/yocto/minnow-demo/meta-yocto \
  /home/trz/yocto/minnow-demo/meta-yocto-bsp \
  /home/trz/yocto/minnow-demo/meta-intel \
  /home/trz/yocto/minnow-demo/meta-intel/meta-minnow \
  /home/trz/yocto/minnow-demo/meta-robot-opencv-demo \
```

- We also told it to add our meta-robot-opencv-demo application layer

meta-robot-opencv-demo layer

```
meta-robot-opencv-demo/
├── conf
│   └── layer.conf
└── recipes-robot-opencv-demo
    ├── images
    │   └── robot-opencv-demo-image.bb
    ├── libav
    │   ├── libav-0.8.4
    │   │   └── 0001-configure-enable-pic-for-AArch64.patch
    │   ├── libav_0.8.4.bb
    │   └── libav.inc
    ├── opencv
    │   ├── opencv
    │   │   └── opencv-fix-pkgconfig-generation.patch
    │   ├── opencv_2.4.3.bb
    │   └── opencv-samples_2.4.3.bb
    ├── python
    │   ├── python-numpy
    │   │   ├── arm
    │   │   │   ├── config.h
    │   │   │   └── numpyconfig.h
    │   │   ├── i586
    │   │   │   ├── config.h
    │   │   │   └── numpyconfig.h
    │   │   ├── trycompile.diff
    │   │   └── unbreak-assumptions.diff
    │   ├── python-numpy_1.7.0.bb
    │   ├── python-pyusb_1.0.0a2.bb
    │   └── python-wxpython_2.8.12.1.bb
```

meta-robot-opencv-demo (cont)

```
├── swig
│   ├── swig
│   │   └── 0001-Use-proc-self-exe-for-swig-swiglib-on-non-Win32-plat.patch
│   ├── swig_2.0.9.bb
│   └── swig.inc
├── v4l2apps
│   ├── files
│   │   └── openat.patch
│   └── v4l-utils_0.8.8.bb
├── webm
│   ├── libvpx
│   │   ├── CVE-2010-4203.patch
│   │   ├── do-not-hardcode-softfp-float-api.patch
│   │   └── libvpx-configure-support-blank-prefix.patch
│   ├── libvpx_0.9.5.bb
│   └── libvpx.inc
├── wxwidgets
│   └── wxwidgets_2.8.12.1.bb
└── x264
    ├── x264
    │   └── don-t-default-to-cortex-a9-with-neon.patch
    └── x264_git.bb

22 directories, 34 files
```

- Meta-robot-opencv-demo lives in the git repository at https://github.com/MinnowBoard

Creating the Image

- To build our image containing the minnow hardware support and robot demo app:

```
$ source oe-init-build-env
$ bitbake robot-opencv-demo-image
```

- The demo image creation is also driven by a recipe, images/robot-opencv-demo-image.bb:

```
DESCRIPTION = "robot-opencv-demo - Contains a basic X11 environment that boots
to a terminal and lets you to run the OWI Robot Arm & OpenCV MinnowBoard demo."

IMAGE_FEATURES += "splash package-management x11-base ssh-server-dropbear"

LICENSE = "MIT"

inherit core-image

IMAGE_INSTALL += "opencv-apps opencv-dev python-opencv python-modules
python-pyusb python-wxpython mesa-demos"
```

Application Recipes

- The image recipe pulls in other recipes (packages) via its dependencies:

```
DESCRIPTION = "Opencv : The Open Computer Vision Library"
HOMEPAGE = "http://opencv.willowgarage.com/wiki/"
SECTION = "libs"

LICENSE = "BSD"
LIC_FILES_CHKSUM = "file://include/opencv2/opencv.hpp;endline=41; \
md5=6d690d8488a6fca7a2c192932466bb14"

DEPENDS = "python-numpy v4l-utils libav gtk+ libtool swig swig-native python"

SRC_URI = "${SOURCEFORGE_MIRROR}/opencv/opencv-unix/${PV}/OpenCV-${PV}.tar.bz2\
           file://opencv-fix-pkgconfig-generation.patch"

S = "${WORKDIR}/OpenCV-${PV}"

# Do an out-of-tree build
OECMAKE_SOURCEPATH = "${S}"
OECMAKE_BUILDPATH = "${WORKDIR}/build-${TARGET_ARCH}"

EXTRA_OECMAKE = "-DBUILD_PYTHON_SUPPORT=ON -DWITH_FFMPEG=ON -DGSTREAMER=OFF"
inherit distutils-base pkgconfig cmake

PACKAGES += "${PN}-apps python-opencv"
```

Summary

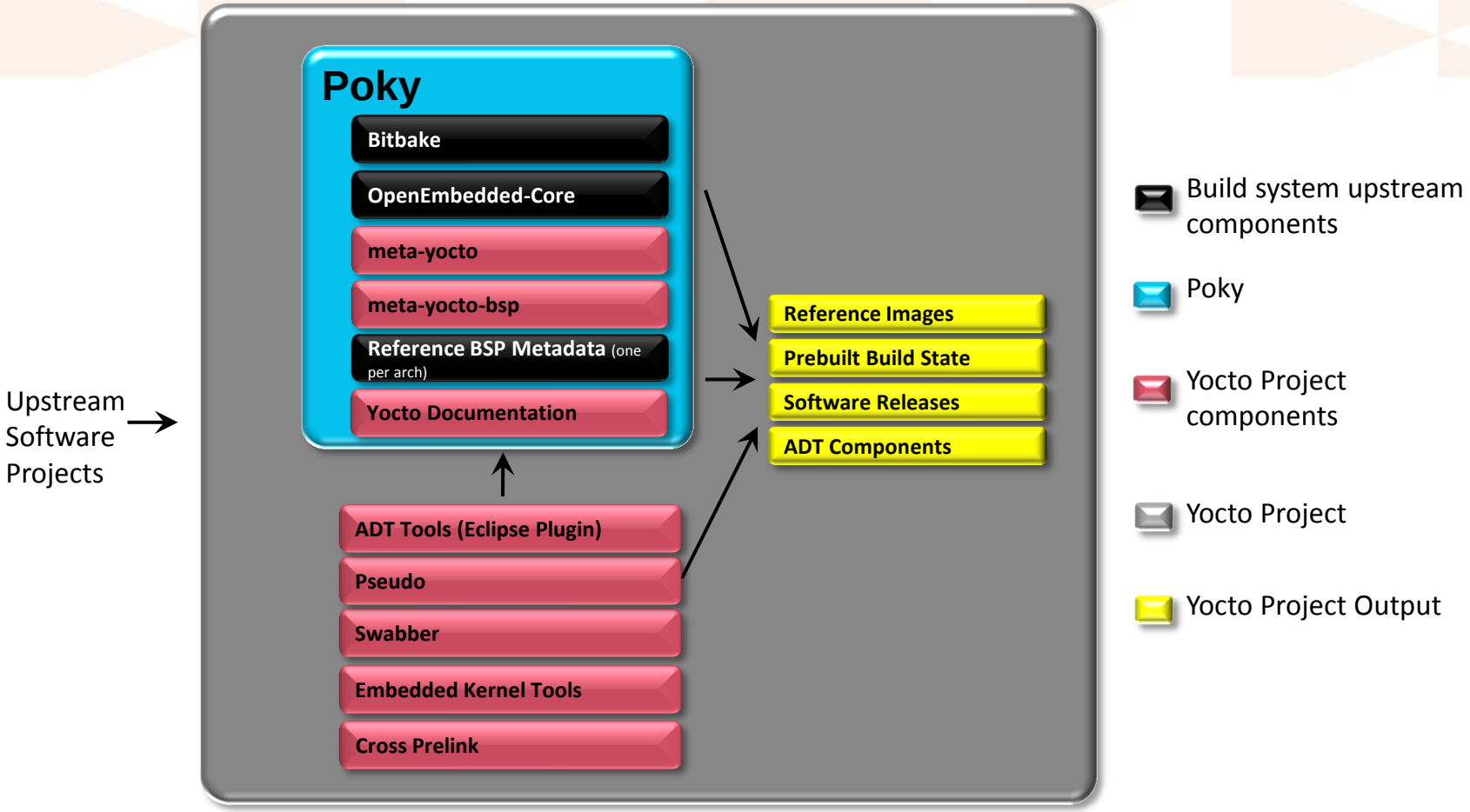
- The Yocto Project helps you to create a custom embedded Linux OS
- Works on any architecture
- Industry supported with a vibrant community
- Intel Contributions make Yocto Project an excellent choice across the Embedded Intel® Architecture Roadmap
- It provides a set of reference distribution components in one place to make it easy to get started
  - It helps set up the embedded app developer
  - Both device and app development models supported
  - Getting started is easy
- Helps with Open Source Licensing monitoring

Go to www.yoctoproject.org for documentation, training, and code.

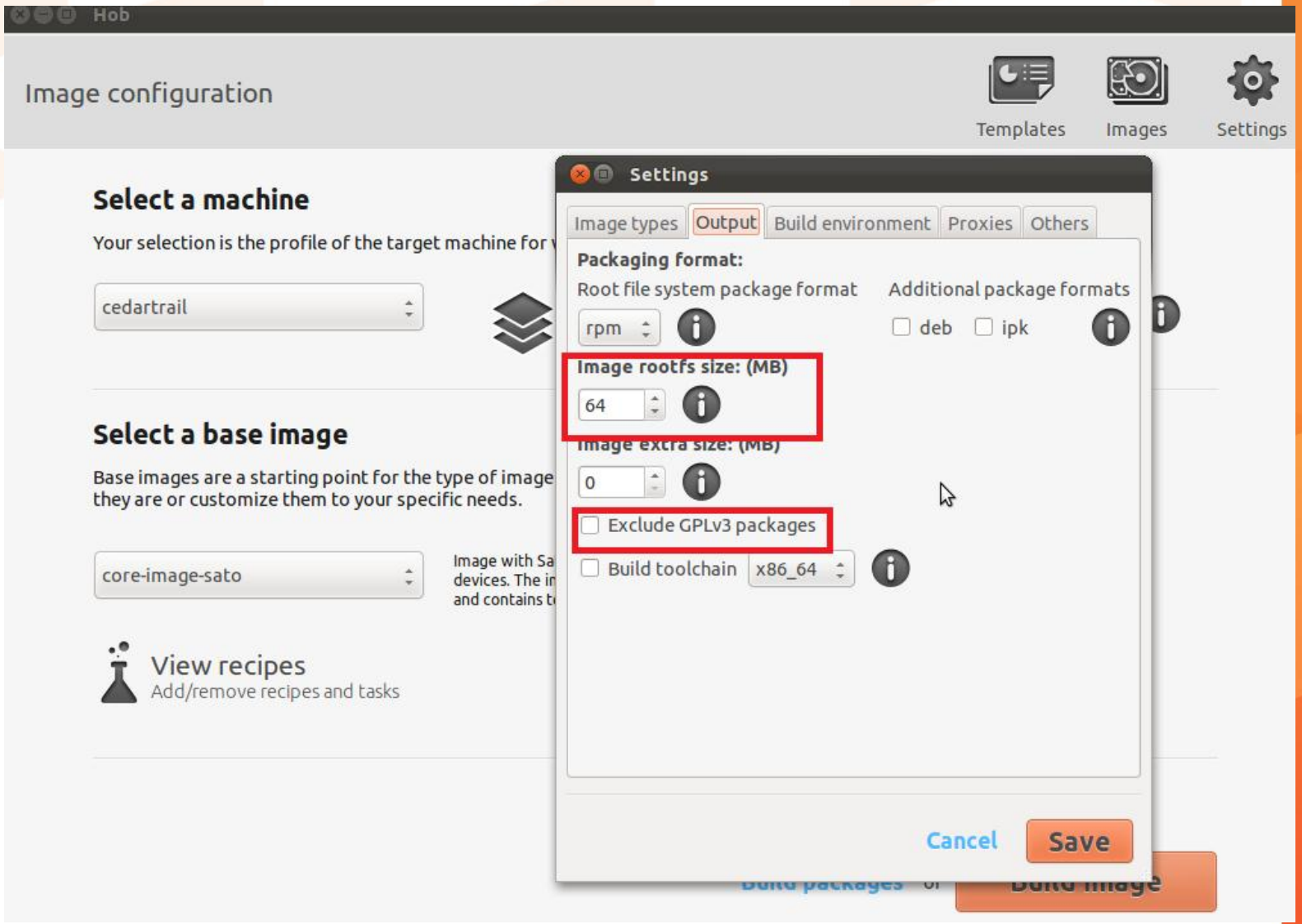
Questions?

Backup

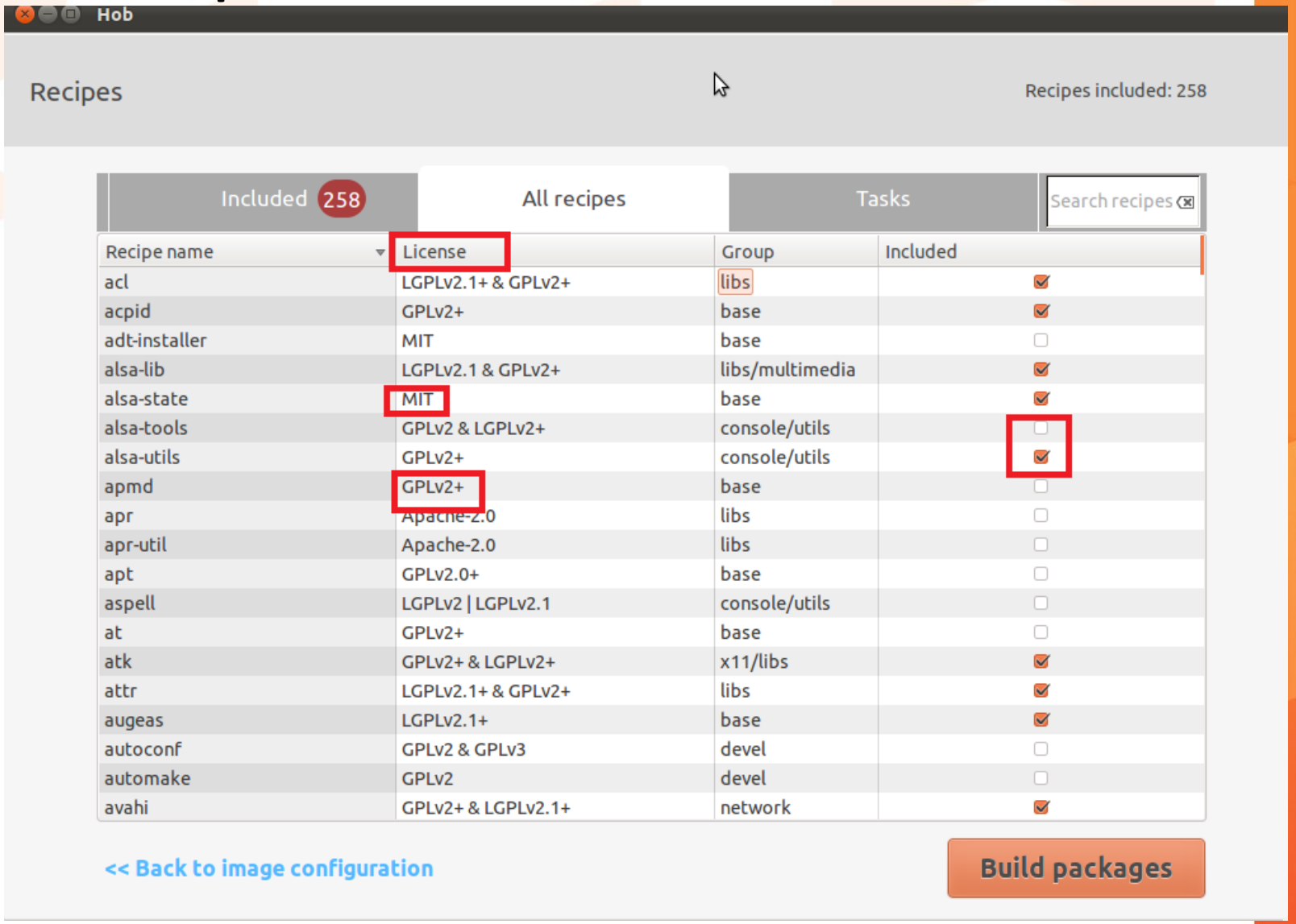
Embedded Tools, Best Practices, and Reference Implementation



Hob: Package Format Control, etc.



Hob: Open Source License Control



Hob: Image Management