yocto PROJECT

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

Building Custom Embedded Images with the Yocto Project

Saul Wold / Tom Zanussi
Intel Corporation
April 13th, 2011

yocto PROJECT · THE LINUX FOUNDATION

Overview

- Some basic background (images, recipes, etc)
- Customizing images with layers
  - BSP Layers
    - Enabling a new machine
    - Dive into a couple key components, the kernel and X
  - Application Layers
    - Media Server Demo
- Future
  - Image Creator
- Q & A

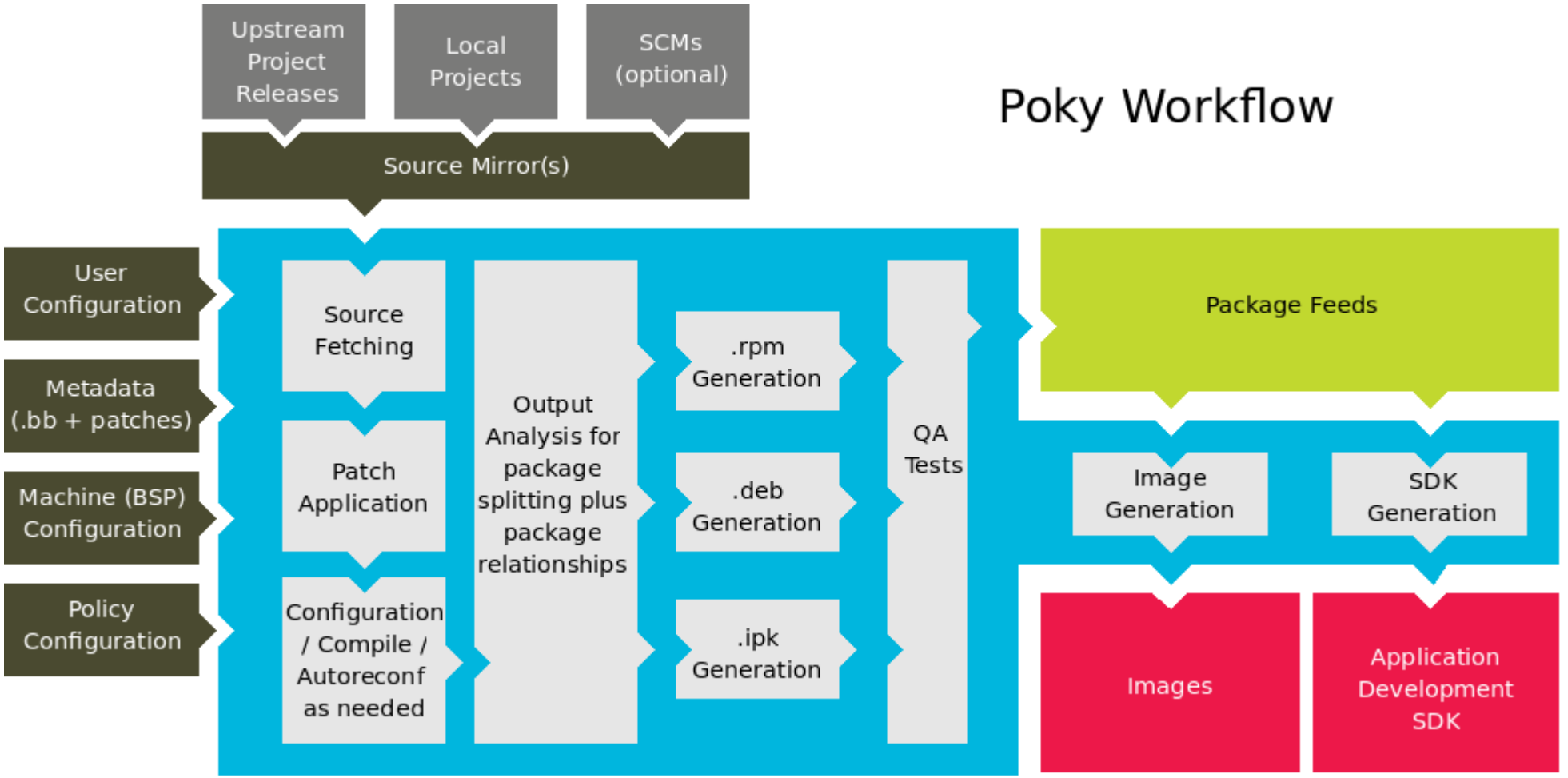
The Yocto Project in a Nutshell

- Tools and metadata for creating custom embedded systems
  - Images are tailored to specific hardware and use cases
  - But metadata is generally arch-independent
  - Unlike a distro, 'kitchen sink' is not included (we know what we need in advance)
- An image is a collection of 'baked' recipes (packages)
- A 'recipe' is a set of instructions for building 'packages'
  - Where to get the source and which patches to apply
  - Dependencies (on libraries or other recipes, for example)
  - Config/compile options, 'install' customization
- A 'layer' is a logical collection of recipes representing the core, a board support package (BSP), or an application stack

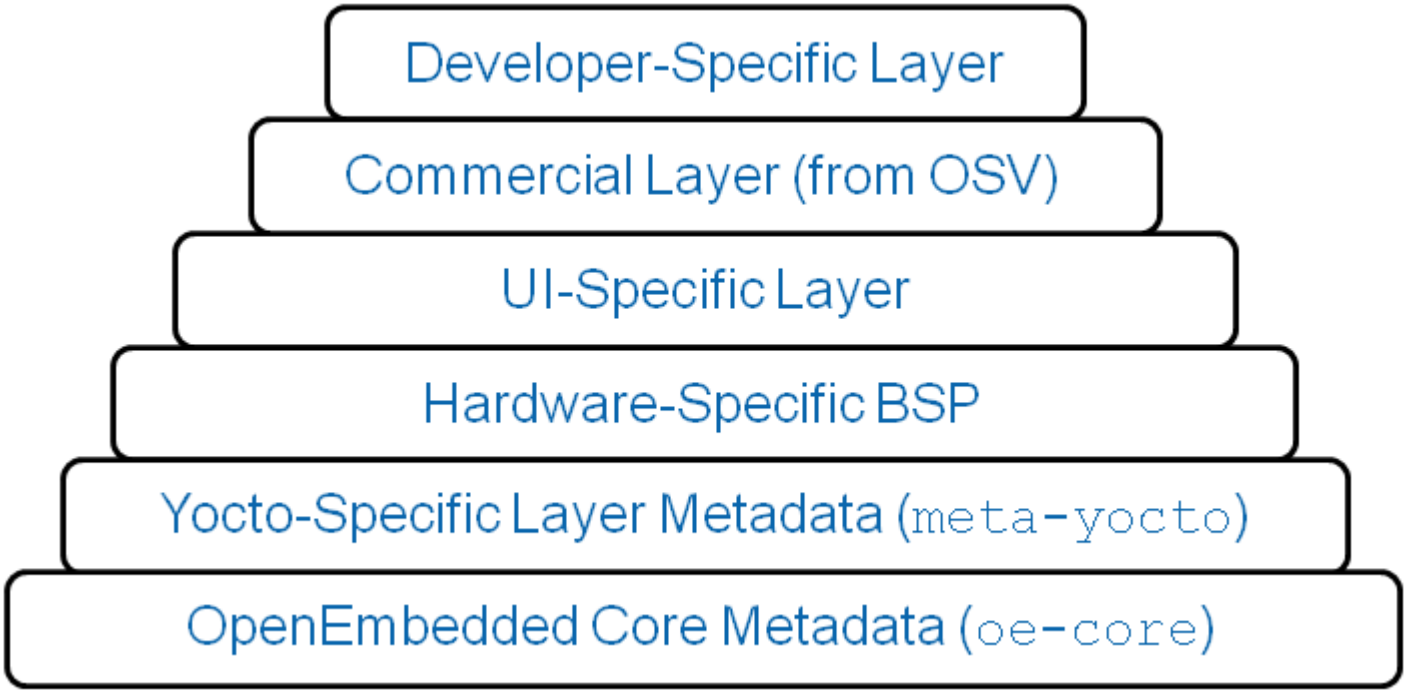
The Yocto Project Build System

- bitbake + metadata (Poky)
  - bitbake - a task executor and scheduler
  - metadata – task definitions in recipes, classes + config
- configuration (.conf) – global definition of variables
  - build/conf/local.conf (local user-defined variables)
  - distro/poky.conf (Yocto 'distro' config variables)
  - machine/beagleboard.conf (machine-specific variables)
- classes (.bbclass) – encapsulation and inheritance of logic
- recipes (.bb) – the logical unit of execution, software/images to build

The Yocto Project and Poky



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)

BSP 'Layers'

- Image contents can be modified by 'layers'
- Layers specialize images by adding or modifying recipes
  - Really just another directory to look for recipes in
  - Added to the BBLAYERS variable in build/conf/bblayers.conf
- BSPs are layers that add machine settings and recipes
- Machine settings are specified in a layer's conf/machine/xxx.conf file(s)
- Examples:
  - Sandy Bridge + Cougar Point:
    - meta-intel/meta-sugarbay/machine/sugarbay.conf
  - Beagleboard (arm)
    - yocto/meta/conf/machine/beagleboard.conf

Practical Requirements for a BSP

- A BSP developer's task is to create a machine layer
  - Define a machine configuration to match hardware
  - Add machine-specific recipes or extend existing recipes
- Very few hard requirements, but there are some
  - All BSPs must select and configure a kernel
  - Machines with graphics capabilities probably also want to select and configure graphics recipes
  - Custom recipes can be added for special hardware
  - Most other things covered by standard recipes
  - New BSPs should follow a standard layout

The Layout of a BSP Layer

```
trz@elmorro:/usr/local/src/yocto/snb/meta-intel$ find ./meta-sugarbay

./meta-sugarbay
./meta-sugarbay/recipes-bsp
./meta-sugarbay/recipes-bsp/formfactor/formfactor/sugarbay
./meta-sugarbay/recipes-bsp/formfactor/formfactor/sugarbay/machconfig
./meta-sugarbay/recipes-bsp/formfactor/formfactor_0.0.bbappend
./meta-sugarbay/COPYING.MIT
./meta-sugarbay/recipes-kernel
./meta-sugarbay/recipes-kernel/linux/linux-yocto_git.bbappend
./meta-sugarbay/README
./meta-sugarbay/recipes-graphics
./meta-sugarbay/recipes-graphics/xorg-xserver/xserver-xf86-config_0.1.bbappend
./meta-sugarbay/recipes-graphics/xorg-xserver/xserver-xf86-config
./meta-sugarbay/recipes-graphics/xorg-xserver/xserver-xf86-config/xorg.conf
./meta-sugarbay/recipes-graphics/xorg-xserver/xserver-xf86-lite_1.9.3.bbappend
./meta-sugarbay/binary
./meta-sugarbay/conf
./meta-sugarbay/conf/machine/sugarbay.conf
./meta-sugarbay/conf/layer.conf
```

A Machine Configuration (meta-sugarbay/conf/machine/sugarbay.conf)

```
TARGET_ARCH = "x86_64"

MACHINE_FEATURES = "kernel26 screen keyboard pci usbhost ext2 ext3 x86”
KERNEL_IMAGETYPE = "bzImage"

PREFERRED_PROVIDER_virtual/kernel = "linux-yocto"
PREFERRED_PROVIDER_linux-libc-headers ?= "linux-libc-headers-yocto"

PREFERRED_PROVIDER_virtual/libx11 ?= "libx11-trim"
PREFERRED_PROVIDER_virtual/libgl  ?= "mesa-dri"
PREFERRED_PROVIDER_virtual/xserver ?= "xserver-xf86-dri-lite"
PREFERRED_PROVIDER_virtual/xserver-xf86 ?= "xserver-xf86-dri-lite"
XSERVER ?= "xserver-xf86-dri-lite \
           xf86-input-mouse \
           xf86-input-keyboard \
           xf86-video-intel"

MACHINE_EXTRA_RRECOMMENDS = "kernel-modules eee-acpi-scripts"
GUI_MACHINE_CLASS = "bigscreen"

IMAGE_ROOTFS_SIZE_ext3 = "2000000"
IMAGE_FSTYPES ?= "ext3 cpio.gz"

MACHINE_ESSENTIAL_EXTRA_RDEPENDS = "grub"
PREFERRED_VERSION_grub ?= "1.98"

SRCREV_machine_pn-linux-yocto_sugarbay ?= "41ec30ddc42912fec133a533b924e9c56ecda8f9"
SRCREV_meta_pn-linux-yocto_sugarbay ?= "5a32d7fe3b817868ebb697d2d883d743903685ae"
```

Selecting and Configuring a Kernel

- The Yocto Project supports several kernels (recipes in meta/recipes-kernel)
  - linux-yocto is the current release's kernel (2.6.37)
  - linux-yocto-stable is the previous release's kernel (2.6.34)
  - linux-yocto-dev is the cutting edge (2.6.39-rc1)
  - All kernels used by the Yocto Project are kernel.org based (plus patches)
- No 'single-kernel' lock-in – you can use any kernel you want
  - You can provide a kernel recipe for any kernel and use it (see e.g. laverne)
  - You can create a git repo usable by the Yocto Project's kernel tools
- The -yocto kernels are contained in standalone git repos
  - The kernel recipes reference the git repos via SRC_URIs:

```
SRC_URI = "git://git.pokylinux.org/linux-yocto-2.6.37;branch=${KBRANCH}
```

- The kernel for a given machine is actually built from two git branches
  - The 'machine' branch (KBRANCH above), and the 'meta' branch

The 'Machine' Branch

- The machine branch is a base kernel plus patches
  - Branch names reflect an inheritance hierarchy

```
yocto/base
yocto/standard/base
yocto/standard/beagleboard
yocto/standard/common-pc-64/base
yocto/standard/common-pc-64/sugarbay
yocto/standard/common-pc/atom-pc
yocto/standard/common-pc/base
yocto/standard/crownbay
```

- You derive your machine branch from one of the */base branches
- Commit machine-specific patches on top of that

```
$ git checkout -b yocto/standard/mymachine yocto/standard/base
$ patch -p1 < mypatch.patch
$ git commit -a -s
```

The 'meta' Branch

- The meta branch defines 'feature descriptions'
  - These are compiled and executed to produce the kernel .config
- For example, here are the elements of the 'logbuf' feature:
- The size-normal.cfg file contains a 'config fragment':

```
CONFIG_LOG_BUF_SHIFT=16
```

- The size-normal.scc file contains a feature command to execute:

```
kconf non-hardware size-normal.cfg
```

- The top-level .scc file adds it by including the feature:

```
include features/logbuf/size-normal.scc
```

- Each 'include features/x' appends another cfg fragment
- The config options up the inheritance hierarchy are also added
- The end result is the .config used to build the kernel

Tying It All Together

- The final step is to tell the BSP about the machine branch
- Recall the SRC_URI from the kernel recipe:

```
KBRANCH = ${KMACHINE}
SRC_URI = "git://git.pokylinux.org/linux-yocto-2.6.37;protocol=git;
    branch=${KBRANCH},meta;name=machine,meta
```

- In the layer, 'append' the machine branch to the recipe:

```
$ cat meta-sugarbay/kernel-recipes/kernel/linux-yocto_git.bbappend

COMPATIBLE_MACHINE_sugarbay = "sugarbay"
KMACHINE_sugarbay  = "yocto/standard/common-pc-64/sugarbay"
```

- The actual machine branch is named by KMACHINE
- The kernel tools know where in 'meta' to start .config
  - At the top-level feature .scc with a matching scc_leaf
- Machine + meta + kernel recipe + bitbake + kernel tools --> kernel

Adding Graphics Capabilities

- Yocto provides an extensive set of X recipes
- Enabling X means selecting the right components
  - The machine configuration defines its set of XSERVER components:

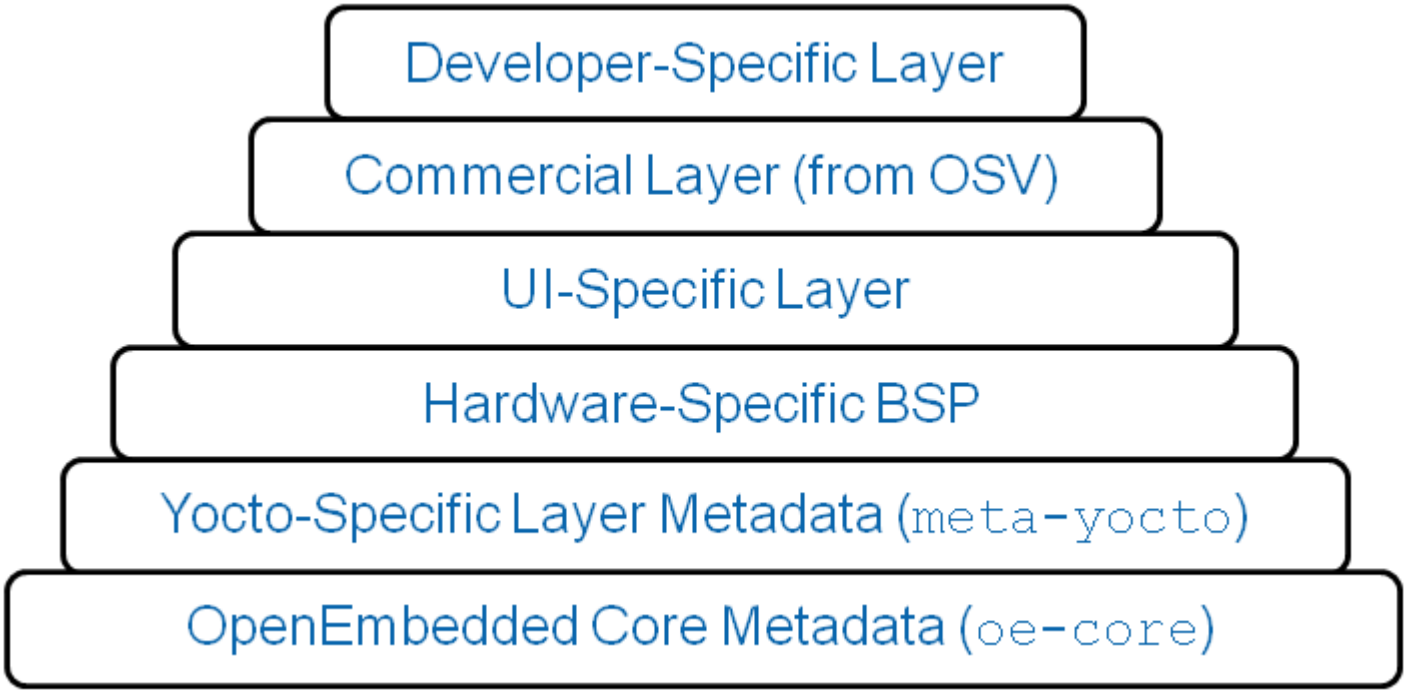
```
XSERVER ?= "xserver-xf86-dri-lite xf86-input-mouse xf86-input-keyboard \
        xf86-video-intel mesa-dri mesa-dri-driver-i915"
```

  - There are several possible implementation choices for each component
  - 'virtual/xserver-xf86' - several different implementation choices for each component
    - xserver-xf86-dri-lite is one implementation, which adds –enable-dri
    - xserver-xf86-lite is another 'lite' implementation that removes dri
    - xserver-kdrive is yet another, that uses –enable-kdrive
  - Need matching kernel graphics options
  - As with any other kernel option, use or create a kernel feature

```
include features/i915/i915.scc
```

- Now that we have Yocto running on our machine, let's build our application!

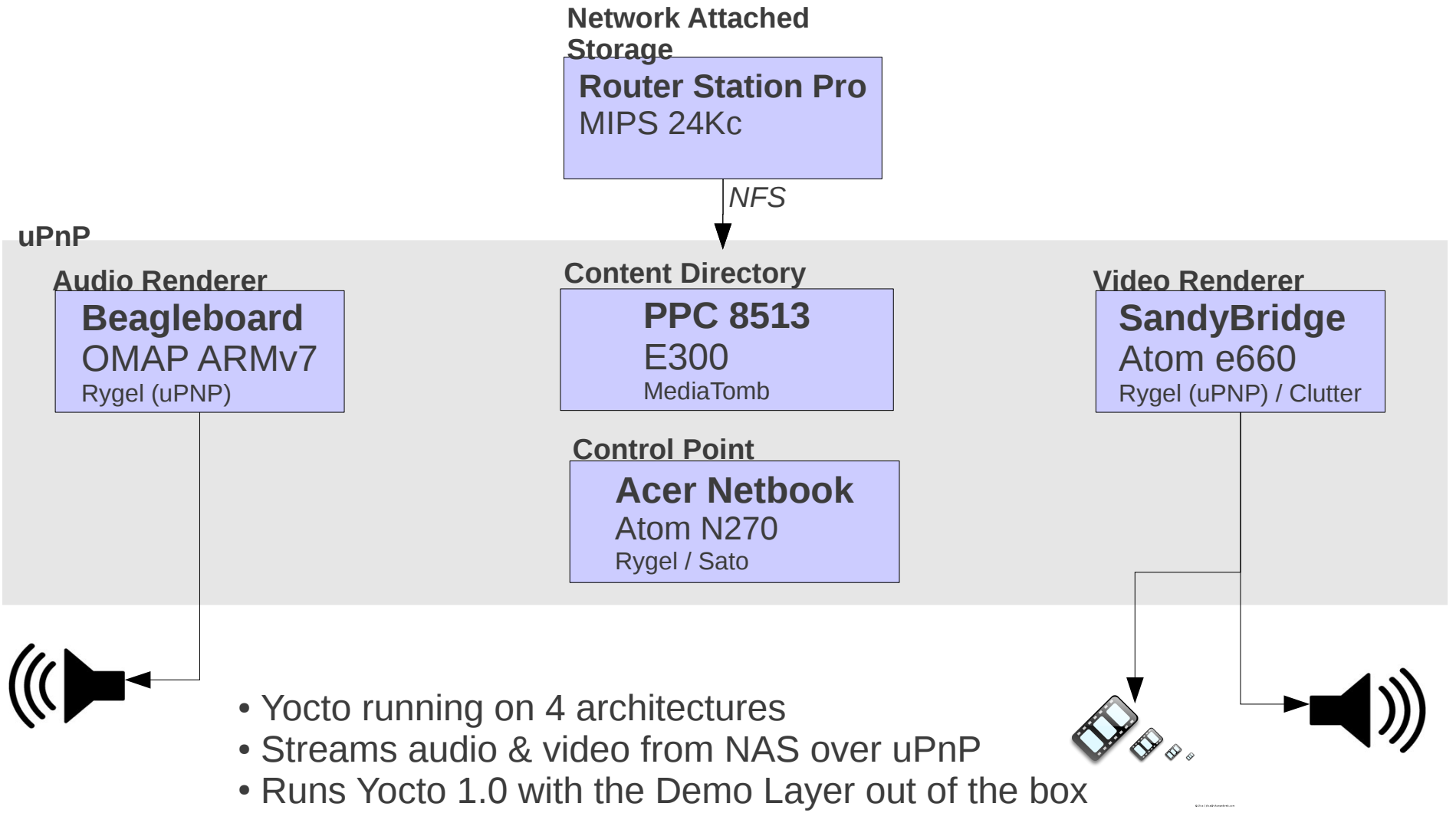
Layers


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

Application Layer on the BSP layer

- Suppose we're working on a project to create a Media Server Demo
- We're going to use the Yocto Project Build System
- We know we'll need a custom image definition for our product, and a layer to keep our changes in, let's start there

Media Server Layer Demo



- Yocto running on 4 architectures
- Streams audio & video from NAS over uPnP
- Runs Yocto 1.0 with the Demo Layer out of the box

conf/layer.conf

```
# Default to first disk/first partition on the Router Station Pro
RSP_ROOT ?= "sda1"
#RSP_ROOT ?= "sda2"

# We have a conf and classes directory, add to BBPATH
BBPATH := "${BBPATH}:${LAYERDIR}"

# We have an images and various recipe-* directories, add to BBFILES

BBFILES := "${BBFILES} ${LAYERDIR}/images/*.bb $
{LAYERDIR}/images/*.bbappend ${LAYERDIR}/recipes-*/*/*.bb $
{LAYERDIR}/recipes-*/*/*.bbappend"


BBFILE_COLLECTIONS += "demo"
BBFILE_PATTERN_demo := "^${LAYERDIR}/"
BBFILE_PRIORITY_demo = "6"

# Setup the audio mixer for the Beagleboard xM
MACHINE_EXTRA_RRECOMMENDS_append_beagleboard += " bbxm-audio"
```

Configure bblayers.conf

- The build/conf/bblayers.conf needs to know where to find the new layer

```
# LCONF_VERSION is increased each time
build/conf/bblayers.conf
# changes incompatibly
LCONF_VERSION = "4"

BBFILES ?= ""
BBLAYERS = " \
  /yocto/poky/meta \
  /yocto/elc/layers/bsp \        # BSP Layer
  /yocto/elc/layers/demo \       # Demo Layer
  "
```

DLNA Library – gupnp-dlna_0.5.0.bb

```
DESCRIPTION = "A utility library for various DLNA-related
functionality useful for DLNA MediaServer implementations."
HOMEPAGE = "http://www.gupnp.org/"

LICENSE = "LGPLv2.1+"
LIC_FILES_CHKSUM = " \

file://COPYING;md5=4fbd65380cdd255951079008b364516c \
                    file://libgupnp-dlna/gupnp-dlna-
discoverer.c;endline=20;md5="

DEPENDS = "gupnp gstreamer gst-plugins-base"
PR = "r0"

SRC_URI = "http://gupnp.org/sites/all/files/sources/${PN}-$
{PV}.tar.gz"

SRC_URI[md5sum] = "c97ffbada5cb9f700d910995fab6ab46"
SRC_URI[md256sum] = "<sha256 sum>"
inherit autotools pkgconfig
```

MediaTomb - mediatomb_0.12.1.bb

```
DESCRIPTION = "MediaTomb - UPnP AV MediaServer for Linux"
HOMEPAGE = "http://mediatomb.cc/"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM =
"file://COPYING;md5=0b609ee7722218aa600220f779cb5035 \

file://src/main.cc;beginline=14;endline=25;md5=<md5sum>"

DEPENDS = "expat ffmpeg sqlite3 libexif js zlib file id3lib
ffmpegthumbnailer curl”
PR = "r1"

SRC_URI = "${SOURCEFORGE_MIRROR}/mediatomb/mediatomb-${PV}.tar.gz \
       file://youtube_warning.patch \
       file://init \
       file://default \
       file://config.xml \
       "

inherit autotools pkgconfig update-rc.d

INITSCRIPT_NAME = "mediatomb"
INITSCRIPT_PARAMS = "defaults 90"
```

MediaTomb (cont)

```
EXTRA_OECONF = "--disable-mysql –disable-rpl-malloc --enable-sqlite3 –enable-
libjs \
        --enable-libmagic --enable-id3lib --enable-libexif --enable-inotify \
        --enable-db-autocreate --disable-largefile --with-sqlite3-h=$
{STAGING_INCDIR} \
        --with-sqlite3-libs=${STAGING_LIBDIR} \
        --with-magic-h=${STAGING_INCDIR} \
        --with-magic-libs=${STAGING_LIBDIR} \
        --with-exif-h=${STAGING_INCDIR} \
        --with-exif-libs=${STAGING_LIBDIR} \
        --with-zlib-h=${STAGING_INCDIR} \
        --with-zlib-libs=${STAGING_LIBDIR} \
        --with-js-h=${STAGING_INCDIR}/js \
        --with-js-libs=${STAGING_LIBDIR} \
        --with-id3lib-h=${STAGING_INCDIR} \
        --with-id3lib-libs=${STAGING_LIBDIR} \
        --with-ffmpeg-h=${STAGING_INCDIR} \
        --with-ffmpeg-libs=${STAGING_LIBDIR} \
                --with-search=${STAGING_DIR_HOST}${prefix}/local \
        ac_cv_header_sys_inotify_h=yes"

SRC_URI[md5sum] = "e927dd5dc52d3cfcebd8ca1af6f0d3c2"
SRC_URI[sha256sum] =
"31163c34a7b9d1c9735181737cb31306f29f1f2a0335fb4f53ecccf8f6
2f11cd"
```

Creating an Image

- IMAGE_INSTALL
  - List tasks and packages to install to create image
- IMAGE_FEATURES
  - Used to further customize what's installed in the image
  - Maps to additional tasks or packages
    - -dev or -dbg packages
- Tasks - arbitrary groups of software, useful when creating several similar images. i.e.
  - A companies proprietary/value-add software
  - All software from a project
    - Graphics / UI
    - Standards

Task Example – task-poky-nfs.bb

```
DESCRIPTION = "NFS tasks for Poky"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${POKYBASE}/LICENSE;md5=<md5sum> \
                    file://${POKYBASE}/meta/COPYING.MIT;md5=<md5sum>"
PR = "r0"

PACKAGES = "\
    task-poky-nfs-server \
    task-poky-nfs-server-dbg \
    task-poky-nfs-server-dev \
    "

ALLOW_EMPTY = "1"

RDEPENDS_task-poky-nfs-server = "\
    nfs-utils \
    "

# rpcinfo can be useful but only with glibc images
GLIBC_DEPENDENCIES = "glibc-utils"

RRECOMMENDS_task-poky-nfs-server_append_linux = "${GLIBC_DEPENDENCIES}"
RRECOMMENDS_task-poky-nfs-server_append_linux-gnueabi = "${GLIBC_DEPENDENCIES}"
```

MediaTomb Image - poky-image-mediatomb.bb

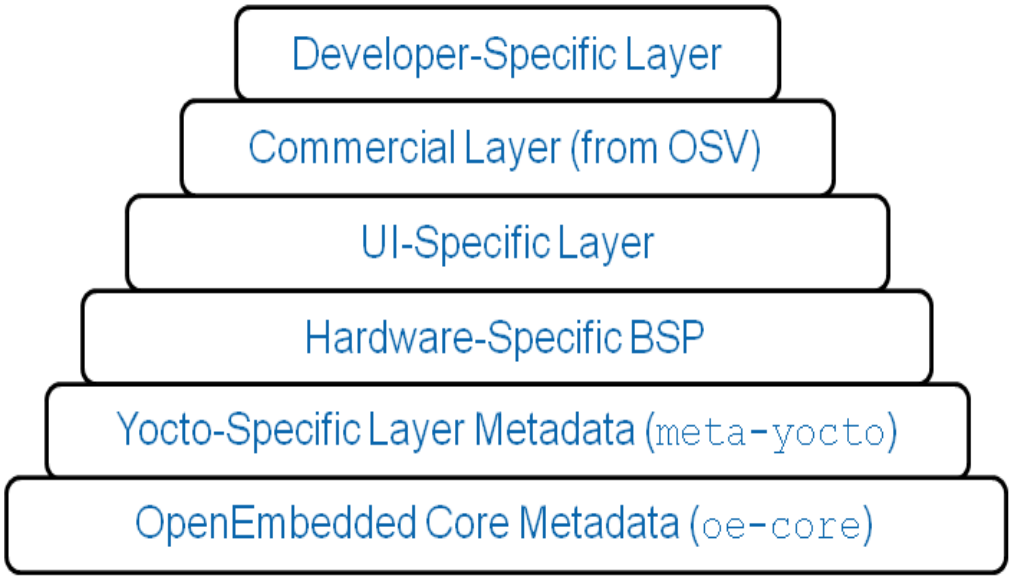
```
#
# Copyright (C) 2010 Intel Corporation.
#
require recipes-core/images/poky-image-minimal.bb

SRC_URI = "file://interfaces"

IMAGE_INSTALL += "dropbear mediatomb task-poky-nfs-server"

LICENSE = "MIT"

ROOTFS_POSTPROCESS_COMMAND += "setup_target_image ; "

# Manual workaround for lack of auto eth0 (see bug #875)
setup_target_image() {
       install -m 0644 ${WORKDIR}/interfaces $
{IMAGE_ROOTFS}/etc/network/interfaces
}
```

poky-image-mediatomb-live.bb

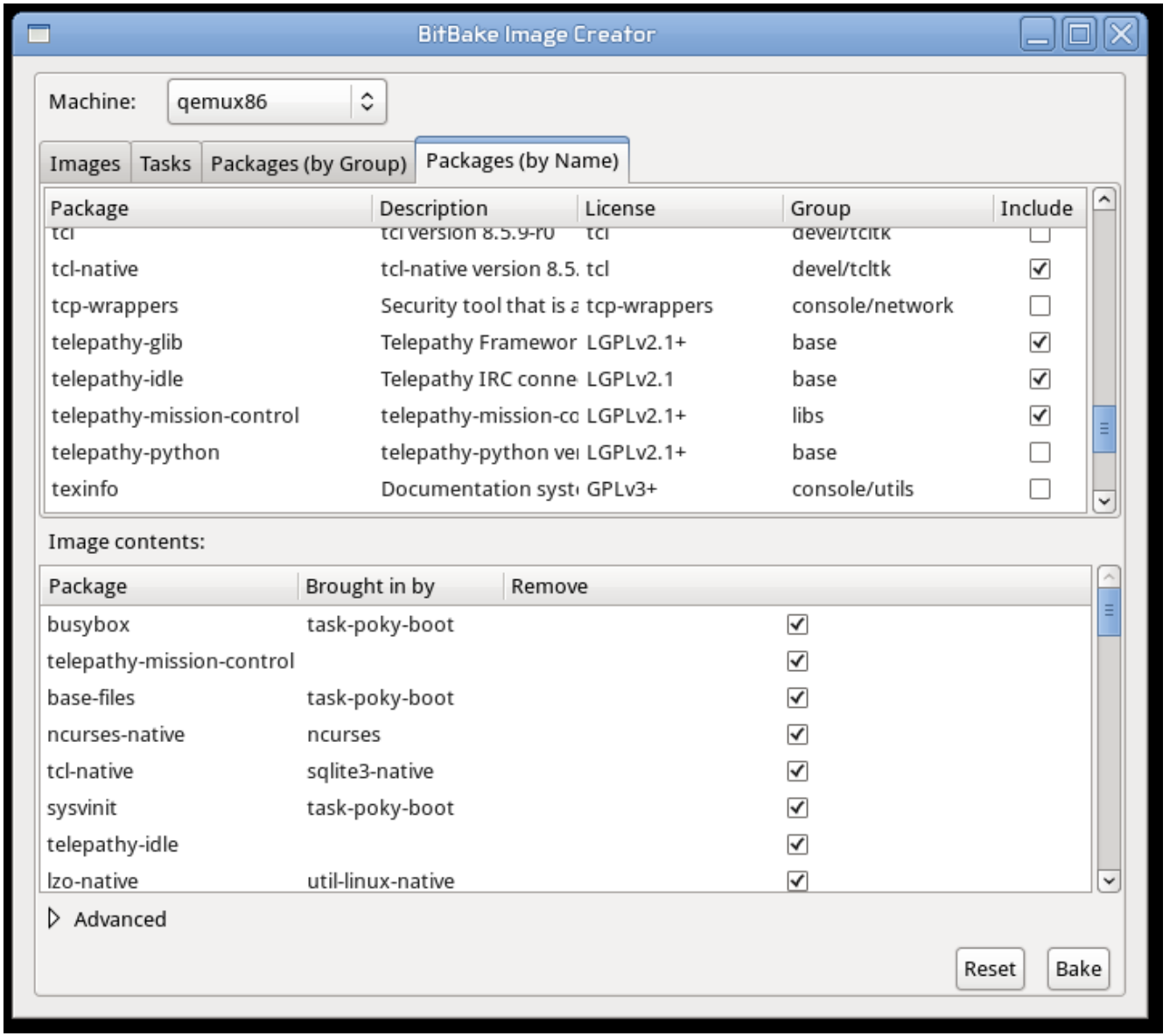
```
DESCRIPTION = "Bootable Live Media Renderer Image"

require recipes-core/images/poky-image-live.inc

LABELS += "boot install"

ROOTFS = "${DEPLOY_DIR_IMAGE}/poky-image-mediatomb-${MACHINE}.ext3"

LICENSE = "MIT"

do_bootimg[depends] += "poky-image-mediatomb:do_rootfs"
```

Layers Summary

- 3 image recipes built for any architecture
- All share same set of recipes, just compiled for the target
- Created Layer which contains
  - Recipes
  - Tasks
  - Images



An Image Creator GUI



What does it do?

- Select and review software to include in an image
- Change some configuration options without having to edit conf files
  - MACHINE
  - Disable GPLv3 recipes
- Future
  - Policy management (distributions, licenses, package format)
  - Layer management
  - Build cross-compiler toolchains and development images
  - Save/load customized configurations

Q & A

Resources

- http://www.yoctoproject.org
- http://wiki.yoctoproject.org

Legal Information


intel
Leap ahead™