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

Create a Custom Embedded Linux Distribution for Any Embedded Device Using the Yocto Project

Chris Hallinan
Mentor Graphics
October 23, 2013

Original Course Content:
Scott Garman, Intel Corporation

Agenda

- **Introduction to the Yocto Project**
- **Key Concepts**
  - Build System Overview & Workflow
  - Exercise 1: Poky Directory Tree Map
- **Recipes In-Depth**
  - Standard Recipe Build Steps
  - Exercise 2: Examining Recipes
- **Building and Booting an Image**
  - Exercise 3: Building Your First Linux Image
  - Exercise 4: Booting Your Linux Image Using QEMU
- **Layers and BSPs**
  - Exercise 5: Creating a Custom Layer
  - Exercise 6-7: Adding a graphical boot logo and SSH server
  - Exercise 8-9: Booting an embedded hardware board

Yocto Project Overview

- Embedded tools and a Linux distribution build environment
- Eglibc, prelink, pseudo, swabber, along with other tools
- Support x86 (32 & 64 bit), ARM, MIPS, PPC
- Shares build system and core metadata (oe-core) with the OpenEmbedded community
- Layer architecture allows for easy re-use of code
- Supports use of rpm/deb/ipk binary package formats (or none at all) in your final image
- Releases on a 6-month cadence
- Latest (stable) kernel, toolchain and packages, documentation
- App Development Tools including Eclipse plugin, ADT, hob

Yocto Project Overview

- **Governance**
  - Open source umbrella project
  - Organized under the Linux Foundation
  - Split governance model

Texas Instruments
SAKOMAN
WIND RIVER
HUAWEI
ENEA software
Mentor Graphics
intel
montavista
openembedded

Needs Updating

yocto PROJECT · THE LINUX FOUNDATION

Yocto Project Build System Overview

- OpenEmbedded (OE) – The overall build architecture used by the Yocto Project
- BitBake – Task executor and scheduler
- Metadata – Task definitions
- Configuration (*.conf) – global definitions of variables
- Classes (*.bbclass) – encapsulation and inheritance of build logic, packaging, etc.
- Recipes (*.bb) – the logical units of software/images to build

Yocto Project Build System Overview

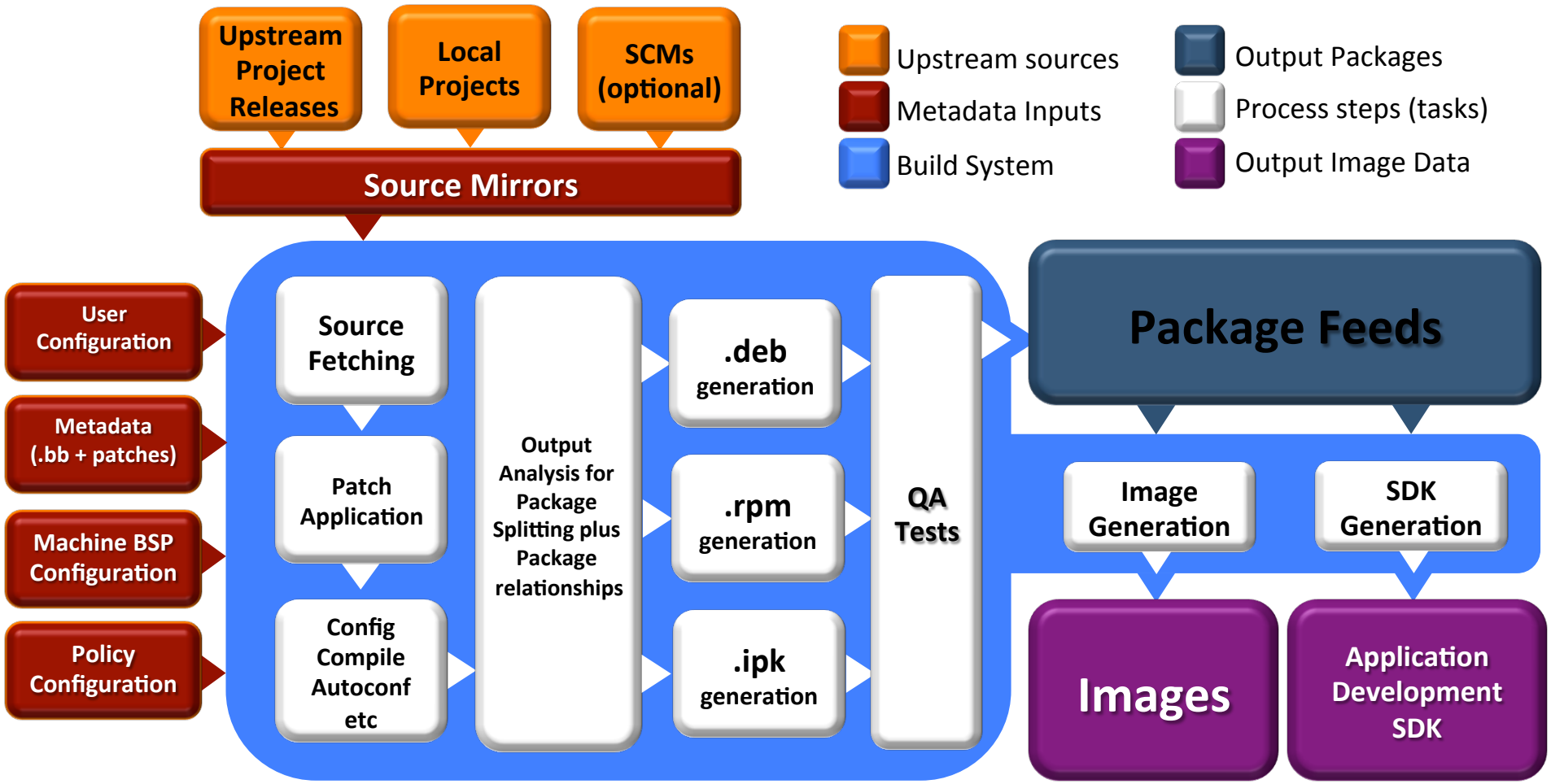
- **OpenEmbedded Core (oe-core)** – A core set of metadata shared by the OpenEmbedded and the Yocto Project
- **meta-yocto** – Reference policy/distro configuration and reference hardware support layer
- **Poky** – A pre-prepared combination of the build system components needed; also the name of our reference distro in meta-yocto

Poky = BitBake + OE-core + meta-yocto + docs

Key Concepts

- **The Yocto Project provides tools and metadata for creating custom Linux images**
- **These images are created from a repository of 'baked' recipes**
- **A recipe is a set of instructions for building packages, including:**
  - Where to obtain the upstream sources and which patches to apply
  - Dependencies (on libraries or other recipes)
  - Configuration/compilation options
  - Define which files go into what output packages

Build System Workflow



Yocto Project Release Versions

- **Major Version Releases**

Name	Revision	Release Date
Bernard	1.0	Apr 5, 2011
Edison	1.1	Oct 17, 2011
Denzil	1.2	Apr 30, 2012
Danny	1.3	Oct 24, 2012
Dylan	1.4	Apr 26, 2013
Dora	1.5	Oct 19, 2013 (planned)

Quick Start Guide in a Slide

- **Download Yocto Project sources:**

```
$ wget http://downloads.yoctoproject.org/releases/yocto/yocto-1.4.2/poky-dylan-9.0.2.tar.bz2
$ tar xf poky-dylan-9.0.2.tar.bz2
$ cd poky-denzil-9.0.2
```

  – Can also use git and checkout a known branch ie. dylan

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

- **Build one of our reference Linux images:**

```
$ source oe-init-build-env
$ MACHINE=qemux86 bitbake core-image-minimal
```

  – Check/Edit `local.conf` for sanity

- **Run the image under emulation:**

```
$ runqemu qemux86
```

Lab 1: Poky Directory Tree Layout

- **Objectives**
  - Familiarize yourself with how the Poky metadata sources are organized
  - Learn where you can find conf files, BitBake class files, and recipe files

Log into your lab computer using the "ilab01" account: Password: "ilab01"

Poky Directory Tree Map

- **bitbake:** the BitBake utility itself
- **documentation:** documentation sources
- **scripts:** various support scripts (e.g, runqemu)
- **meta/conf:** important configuration files, `bitbake.conf`, reference distro config, machine configs for QEMU architectures
- **meta/classes:** BitBake classes
- **meta/recipes-* :** recipes

Recipes In-Depth Agenda

- **Example Recipe: ethtool**
- **Standard Recipe Build Steps**
- **Exercise 2: Examining Recipes**

Example Recipe – ethtool_3.8.bb

chris@speedy: ~ — ssh — 89×22

```
SUMMARY = "Display or change ethernet card settings"
DESCRIPTION = "A small utility for examining and tuning the settings of your ethernet-bas
ed network interfaces."
HOMEPAGE = "http://www.kernel.org/pub/software/network/ethtool/"
SECTION = "console/network"
LICENSE = "GPLv2+"
LIC_FILES_CHKSUM = " \
        file://COPYING;md5=94d55d512a9ba36caa9b7df079bae19f \
        file://ethtool.c;beginline=4;endline=17;md5=c19b30548c582577fc6b443626fc1216"

PR = "r0"

SRC_URI = "${KERNELORG_MIRROR}/software/network/ethtool/ethtool-${PV}.tar.gz"

SRC_URI[md5sum] = "ddceef30c99cee26874798821e9d6ab8"
SRC_URI[sha256sum] = "648e1c311567571f806389d85efaa063a8035e09529d0467c4ea97c9ece829d6"

inherit autotools
~
~
                                                                18,0-1        All
```

Standard Recipe Build Steps

- Building recipes involves executing the following functions, which can be overridden when needed for customizations*

- **do_fetch**
- **do_unpack**
- **do_patch**
- **do_configure**
- **do_compile**
- **do_install**
- **do_package**

Note: to see the list of all possible functions (tasks) for a recipe, do this:

```
$ bitbake -c listtasks <recipe_name>
```

*Simplified for training purposes

Exercise 2: Examining Recipes

- **meta/recipes-extended/bc/bc_1.06.bb**
  - Uses `LIC_FILES_CHKSUM` and `SRC_URI` checksums
  - Note the `DEPENDS` build dependency declaration
- **meta/recipes-multimedia/flac/flac_1.2.1.bb**
  - Includes custom source patches to apply to the sources
  - Customizes autoconf configure options (`EXTRA_OECONF`)
  - Overrides the `do_configure()` build step
  - Breaks up output into multiple binary packages
- **meta/recipes-connectivity/ofono/**
  - Splits recipe into common .inc file to share metadata between multiple recipes
  - Sets a conditional build `DEPENDS` based on a distro feature (in the `.inc` file)
  - Sets up an init service via `do_install_append()`
  - Has a `_git` version of the recipe

Split into 3 slides

Exercise 3: Building a Linux Image

- `$ cd /build/intro-lab/projects`
- **Source ../poky-dylan-1.9.2/oe-init-build-env**
  - Sets up important environment variables
- **Set MACHINE = "qemux86" in conf/local.conf**
  - Specifies that we're building for the qemux86 target
- **bitbake core-image-minimal**
  - Builds a reference image for the qemux86 target

Needs Updating

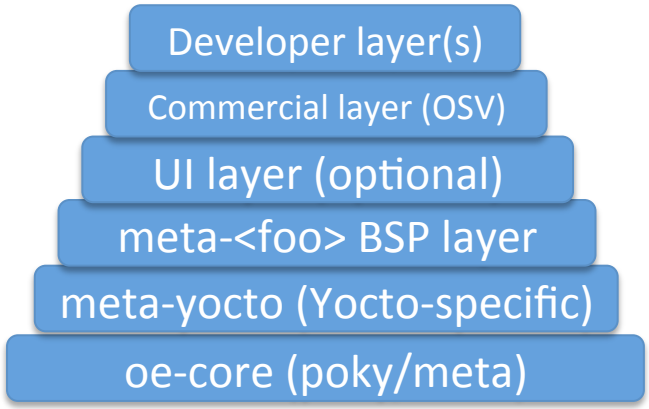
Exercise 4: Booting Your Image with QEMU

- Yocto uses QEMU, which supports all major architectures: x86(-64), arm, mips, power
- Simply set MACHINE to one of these qemu[arch] types in `local.conf` and build your image
- The `runqemu` script is used to boot the image with QEMU – it auto-detects settings as much as possible, allowing the following to boot our reference images:

- **`$ runqemu qemux86 [nograpic]`**

Layers Agenda

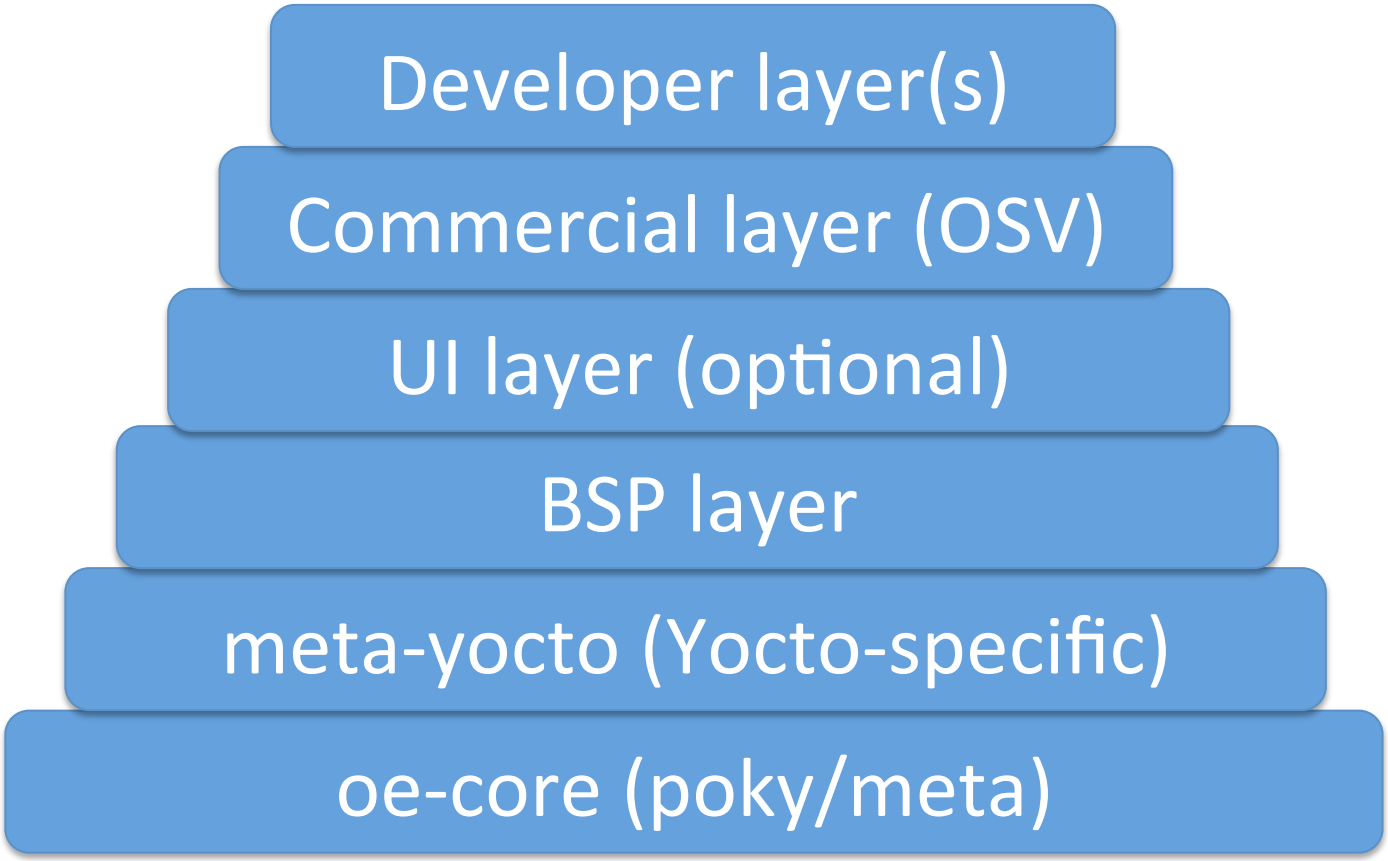
- **Introduction to Layers**
- **Stacking Customizations**
- **Adding Layers**
- **Board Support Packages**



Layers

- The Yocto Project build system is composed of layers
- A **layer** is a logical collection of recipes representing the core, a Board Support Package (BSP), or an application stack
- All layers have a priority and can override policy and config settings of the layers beneath it

Layer Hierarchy



Using Layers

- Layers are added to your build by inserting them into the BBLAYERS variable within your `build/conf/bblayers.conf` file:

```
BBLAYERS = "                    \
  /<install-path>/poky/meta              \
  /<install-path>/poky/meta-yocto  \
  /<my-meta-dir>/meta-my-custom-layer \
  "
```

Board Support Packages

- BSPs are layers to enable support for specific hardware platforms
- Defines machine configuration for the “board”

- Adds machine-specific recipes and customizations
  - Kernel config
  - Graphics drivers (e.g, Xorg)
  - Additional recipes to support hardware features

Exercise 5: Create a Custom Layer

- When doing development with Yocto, **do not edit files within the Poky source tree** – use a custom layer for modularity and maintainability
- Create a custom layer to hold a custom image recipe
- Let's call this layer **meta-ypdd**
- This layer must include:
  - `meta-ypdd/conf/layer.conf file`
  - Recipes directory (`meta-ypdd/recipes-ypdd/`)
  - A `meta-ypdd/README` file (basic documentation for the layer, including maintainer info)

meta-ypdd/conf/layer.conf

chris@speedy: ~ — ssh — 52×15

```
BBPATH .= ":${LAYERDIR}"
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \
            ${LAYERDIR}/recipes-*/*/*.bbappend"

BBFILE_COLLECTIONS += "ypdd"

BBFILE_PRIORITY_ypdd = "10"

BBFILE_PATTERN_ypdd = "^${LAYERDIR}/"

# BB_DANGLINGAPPENDS_WARNONLY = "1"

                                        12,0-1        Top
```

Exercise 6: Creating a Custom Image Recipe

- We'll derive this from **core-image-minimal**, but add support for a graphical boot logo (via **psplash**) and an SSH server (**dropbear**)
- We'll name our custom image **ypdd-image**, so the recipe will be **meta-ypdd/recipes-ypdd/images/ypdd-image.bb**
- The simplest way to add packages to a predefined image is to append them to **IMAGE_INSTALL** within the image recipe

Exercise 6: Creating a Custom Image Recipe

chris@speedy: ~ — ssh — 74×23

```
DESCRIPTION = "A core image for YPDD!"

IMAGE_INSTALL = "packagegroup-core-boot ${ROOTFS_PKGMANAGE_BOOTSTRAP} \
                 ${CORE_IMAGE_EXTRA_INSTALL}"

IMAGE_LINGUAS = " "

LICENSE = "MIT"

inherit core-image

IMAGE_ROOTFS_SIZE = "8192"

# remove not needed ipkg informations
ROOTFS_POSTPROCESS_COMMAND += "remove_packaging_data_files ; "
```

16,0-1 All

Exercise 7: Build and Boot Your Custom Image

- **Enable the meta-ypdd layer**
- Edit `conf/bblayers.conf` and add the path to `meta-ypdd` to the `BBLAYERS` variable declaration

- **Build your custom image:**
- `$ bitbake ypdd-image`

- **Boot the image with QEMU:**
- `$ runqemu qemux86 tmp/deploy/images/ypdd-image-qemux86.ext3`

Common Gotchas When Getting Started

- **Working behind a network proxy? Please follow this guide:**
- *https://wiki.yoctoproject.org/wiki/Working_Behind_a_Network_Proxy*
- **Do not try to re-use the same shell environment when moving between copies of the build system**
- **oe-init-build-env** script appends to your **$PATH**, so is not idempotent and can cause unpredictable build errors
- **Do not try to share sstate-cache between hosts running different Linux distros even if they say it works ;)**

Project Resources

- The Yocto Project is an open source project, and aims to deliver an open standard for the embedded Linux community and industry
- Development is done in the open through public mailing lists: openembedded-core@lists.openembedded.org, poky@yoctoproject.org, and yocto@yoctoproject.org
- And public code repositories:
- http://git.yoctoproject.org and
- http://git.openembedded.net
- Bug reports and feature requests
- http://bugzilla.yoctoproject.org

It's not an embedded Linux distribution

It creates a custom one for you