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

March 26, 2015

yocto PROJECT · THE LINUX FOUNDATION

Agenda

- **Introduction to the Yocto Project**
- **Key Concepts**
  - Build System Overview & Workflow
  - Exercise 1: Exploring the Host System
- **Recipes In-Depth**
  - Standard Recipe Build Steps and sstate
  - 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: Adding a custom application

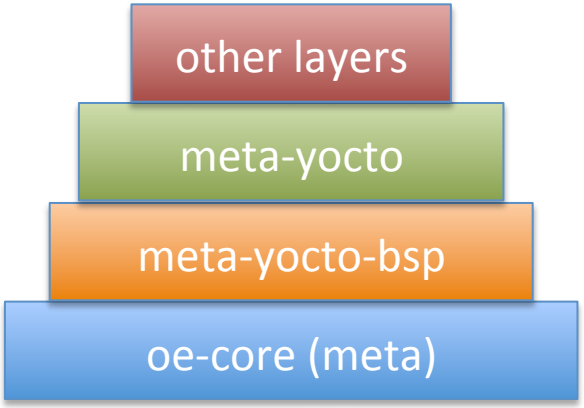
Yocto Project Overview

Governance

- Organized under the Linux Foundation
- Split governance model
- Technical Leadership Team
- Advisory Board made up of participating organizations

Yocto Project Overview

- **Collection of tools and methods enabling**
  - Rapid evaluation of embedded Linux on many popular off-the-shelf boards
  - Easy customization of distribution characteristics
- **Supports x86, ARM, MIPS, Power**
- **Based on technology from the OpenEmbedded Project**
- **Layer architecture allows for easy re-use of code**



Yocto Project Overview

- **Supports use of popular package formats including:**
  - rpm, deb, ipk
- **Releases on a 6-month cadence**
- **Latest (stable) kernel, toolchain and packages, documentation**
- **App Development Tools including Eclipse plugin, ADT, hob**

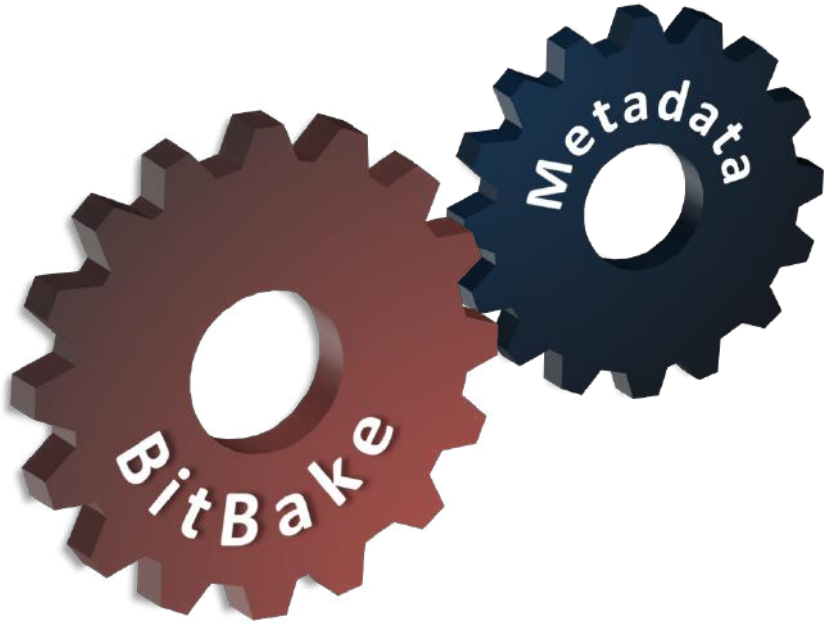
Yocto Project Release Versions

Major Version Releases

Name	Revision	Poky	Release Date
Bernard	1.0	5.0	Apr 5, 2011
Edison	1.1	6.0	Oct 17, 2011
Denzil	1.2	7.0	Apr 30, 2012
Danny	1.3	8.0	Oct 24, 2012
Dylan	1.4	9.0	Apr 26, 2013
Dora	1.5	10.0	Oct 19, 2013
Daisy	1.6	11.0	Apr 24, 2014
Dizzy	1.7	12.0	Oct 31, 2014
Fido	1.8	tbd	April 24, 2015 (planned)

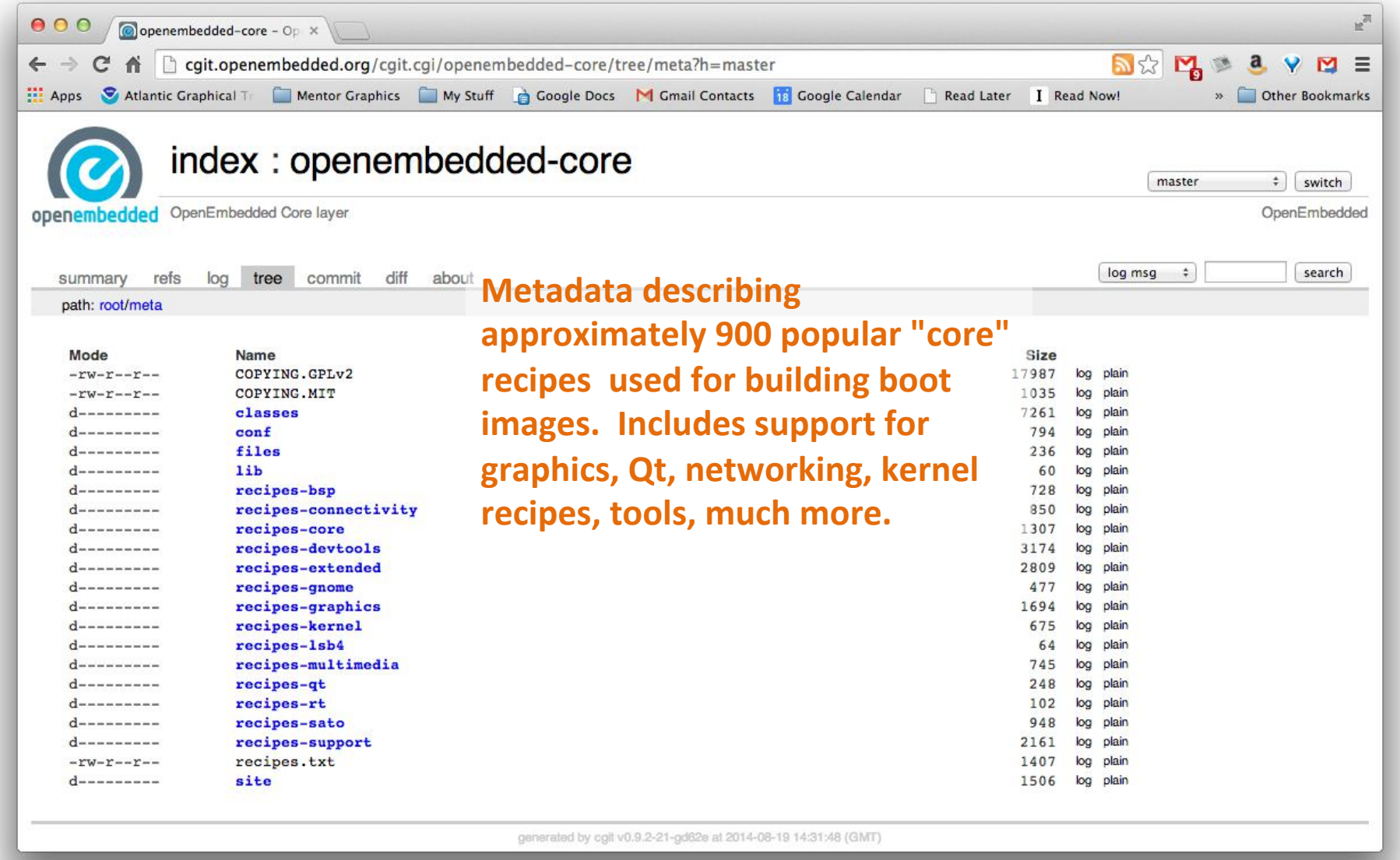
Intro to OpenEmbedded

- **Bitbake**
  - Powerful and flexible build engine
  - Determines dependencies and schedules tasks



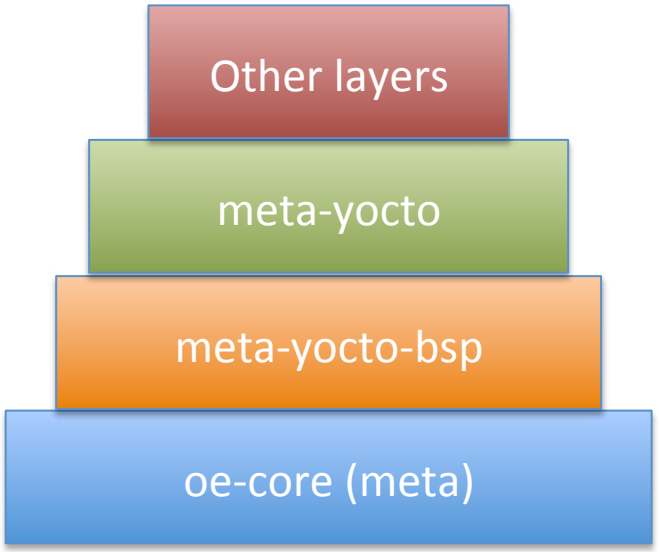
Metadata – a structured collection of "recipes" which tell BitBake what to build, organized in layers

Yocto is based on openembedded-core



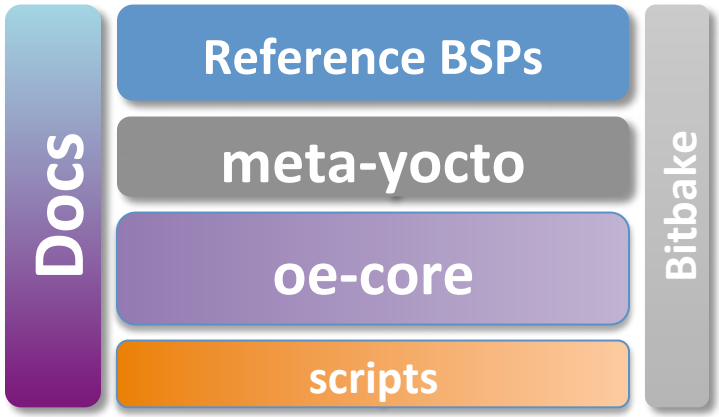
OK, so what is Poky?

- **Poky is both a reference distribution and a build system**
- **Poky has its own git repo**
  - git clone git://git.yoctoproject.org/poky
- **Primary poky layers**
  - oe-core (poky/meta)
  - meta-yocto-bsp
  - meta-yocto
- **Poky is the foundation of YP**



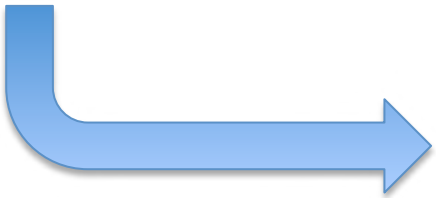
Poky in Detail

- **Poky is the foundation of the build system and reference distribution**
- **Contains core components**
  - Foundation package recipes (**oe-core**)
  - Yocto Project documentation
  - Bitbake: A python-based build engine
  - Build scripts (infrastructure)
  - Reference BSPs
  - meta-yocto
    - Contains distribution policy

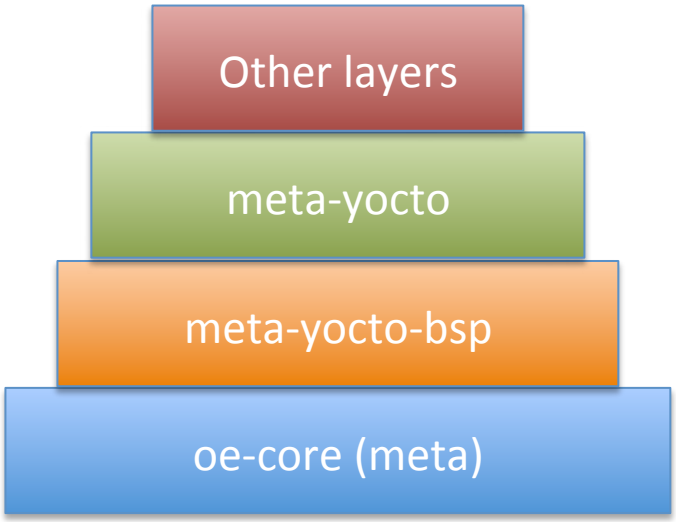


Putting It All Together

- Yocto Project is a large collaboration
- Poky is the Yocto Project's reference distribution and build system
- Poky contains several "layers" of *metadata*



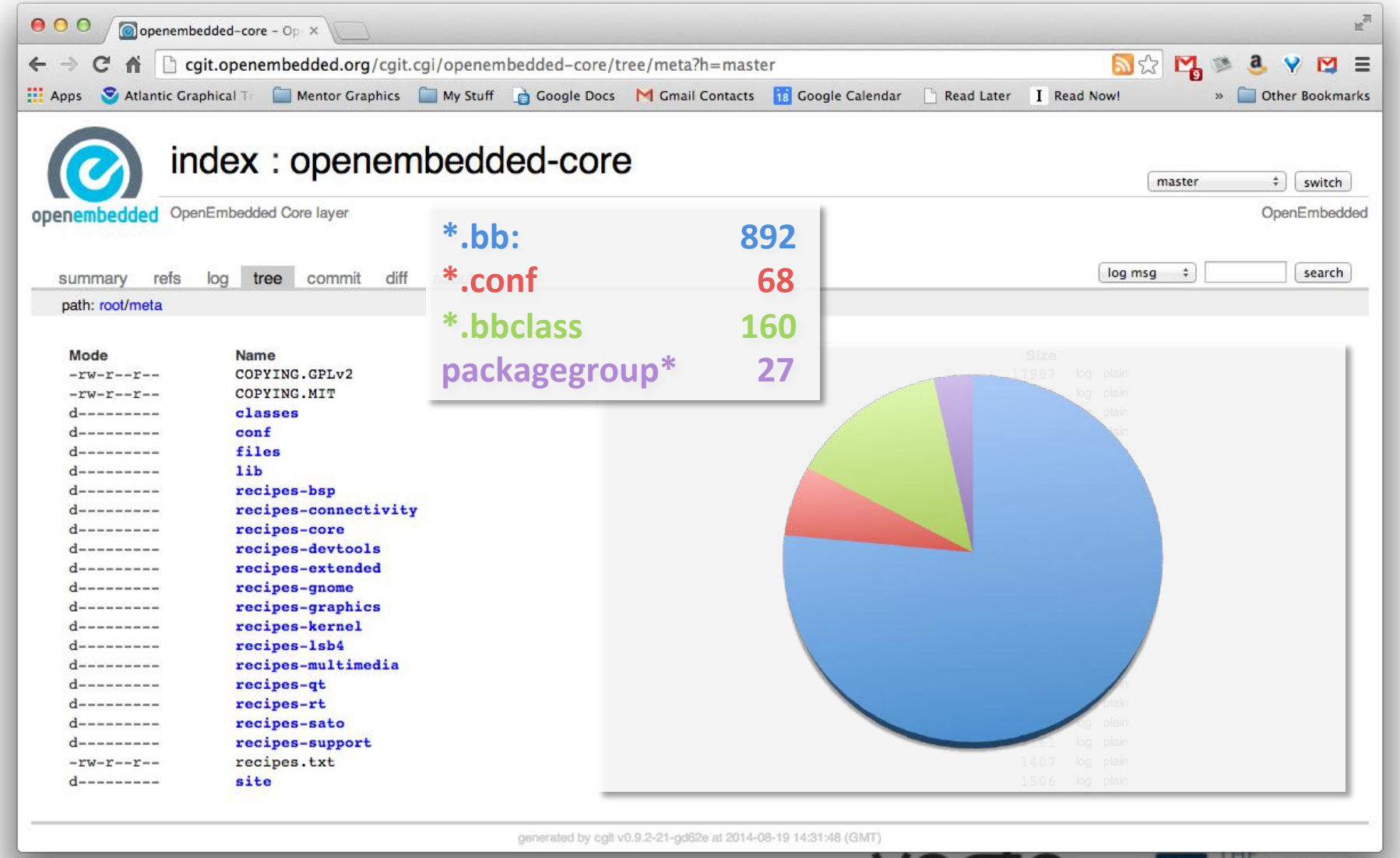
Metadata: generic term for the language that describes how to build things using OE



What is Metadata?

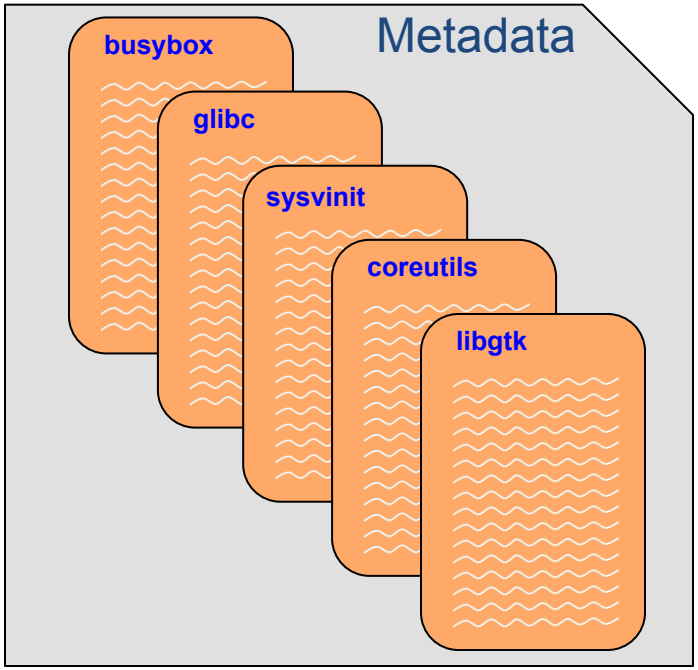
- **Metadata exists in four general categories:**
- **Recipes (*.bb)**
  - Usually describe build instructions for a single package
- **PackageGroups (special *.bb)**
  - Often used to group packages together for a FS image
- **Classes (*.bbclass)**
  - Inheritance mechanism for common functionality
- **Configuration(*.conf)**
  - Drives the overall behavior of the build process

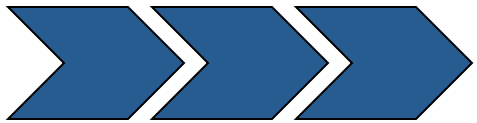
OE-CORE Breakdown



Recipe Basics

- **Most common form of metadata – The Recipe**
- **Provides the "list of ingredients" and "cooking instructions" to build a package(s)**
- **Has a common set of tasks…**





Recipe Basics

- **A recipe is a set of instructions for building packages, including:**
  - Where to obtain the upstream sources and which patches to apply
    - `SRC_URI`
  - Dependencies (on libraries or other recipes)
    - `DEPENDS, RDEPENDS`
  - Configuration/compilation options
    - `EXTRA_OECONF, EXTRA_OEMAKE`
  - Define which files go into what output packages
    - `FILES_*`

Example Recipe – ethtool_3.15.bb

chris — ssh — 80×24

```
SUMMARY = "Display or change ethernet card settings"
DESCRIPTION = "A small utility for examining and tuning the settings of your eth
ernet-based network interfaces."
HOMEPAGE = "http://www.kernel.org/pub/software/network/ethtool/"
SECTION = "console/network"
LICENSE = "GPLv2+"
LIC_FILES_CHKSUM = "file://COPYING;md5=b234ee4d69f5fce4486a80fdaf4a4263 \
                    file://ethtool.c;beginline=4;endline=17;md5=c19b30548c582577
fc6b443626fc1216"

SRC_URI = "${KERNELORG_MIRROR}/software/network/ethtool/ethtool-${PV}.tar.gz \
           file://run-ptest \
           file://avoid_parallel_tests.patch \
           file://ethtool-uint.patch \
           "

SRC_URI[md5sum] = "7e94dd958bcd639aad2e5a752e108b24"
SRC_URI[sha256sum] = "562e3cc675cf5b1ac655cd060f032943a2502d4d59e5f278f02aae9256
2ba261"

inherit autotools ptest
RDEPENDS_${PN}-ptest += "make"

                                                              1,1           Top
```

Example Recipe – ethtool_3.15.bb

chris — ssh — 80×24

```
SUMMARY = "Display or change ethernet card settings"
DESCRIPTION = "A small utility for examining and tuning the settings of your eth
ernet-based network interfaces."
HOMEPAGE = "http://www.kernel.org/pub/software/network/ethtool/"
SECTION = "console/network"

LICENSE = "GPLv2+"
LIC_FILES_CHKSUM = "file://COPYING;md5=b234ee4d69f5fce4486a80fdaf4a4263 \
                    file://ethtool.c;beginline=4;endline=17;md5=c19b30548c582577
fc6b443626fc1216"

SRC_URI = "${KERNELORG_MIRROR}/software/network/ethtool/ethtool-${PV}.tar.gz \
           file://run-ptest \
           file://avoid_parallel_tests.patch \
           file://ethtool-uint.patch \
           "

SRC_URI[md5sum] = "7e94dd958bcd639aad2e5a752e108b24"
SRC_URI[sha256sum] = "562e3cc675cf5b1ac655cd060f032943a2502d4d59e5f278f02aae9256
2ba261"

inherit autotools ptest
RDEPENDS_${PN}-ptest += "make"

                                                    1,1           Top
```

Example Recipe – ethtool_3.15.bb

chris — ssh — 80×24

```
SUMMARY = "Display or change ethernet card settings"
DESCRIPTION = "A small utility for examining and tuning the settings of your eth
ernet-based network interfaces."
HOMEPAGE = "http://www.kernel.org/pub/software/network/ethtool/"
SECTION = "console/network"
LICENSE = "GPLv2+"
LIC_FILES_CHKSUM = "file://COPYING;md5=b234ee4d69f5fce4486a80fdaf4a4263 \
                    file://ethtool.c;beginline=4;endline=17;md5=c19b30548c582577
fc6b443626fc1216"

SRC_URI = "${KERNELORG_MIRROR}/software/network/ethtool/ethtool-${PV}.tar.gz \
           file://run-ptest \
           file://avoid_parallel_tests.patch \
           file://ethtool-uint.patch \
           "

SRC_URI[md5sum] = "7e94dd958bcd639aad2e5a752e108b24"
SRC_URI[sha256sum] = "562e3cc675cf5b1ac655cd060f032943a2502d4d59e5f278f02aae9256
2ba261"

inherit autotools ptest
RDEPENDS_${PN}-ptest += "make"

                                                    1,1           Top
```

Example Recipe – ethtool_3.15.bb

chris — ssh — 80×24

```
SUMMARY = "Display or change ethernet card settings"
DESCRIPTION = "A small utility for examining and tuning the settings of your eth
ernet-based network interfaces."
HOMEPAGE = "http://www.kernel.org/pub/software/network/ethtool/"
SECTION = "console/network"
LICENSE = "GPLv2+"
LIC_FILES_CHKSUM = "file://COPYING;md5=b234ee4d69f5fce4486a80fdaf4a4263 \
                    file://ethtool.c;beginline=4;endline=17;md5=c19b30548c582577
fc6b443626fc1216"

SRC_URI = "${KERNELORG_MIRROR}/software/network/ethtool/ethtool-${PV}.tar.gz \
           file://run-ptest \
           file://avoid_parallel_tests.patch \
           file://ethtool-uint.patch \
           "

SRC_URI[md5sum] = "7e94dd958bcd639aad2e5a752e108b24"
SRC_URI[sha256sum] = "562e3cc675cf5b1ac655cd060f032943a2502d4d59e5f278f02aae9256
2ba261"

inherit autotools ptest
RDEPENDS_${PN}-ptest += "make"

                                                              1,1           Top
```

Recipe Basics – Default Tasks*

`do_fetch`	**Locate and download source code**
`do_unpack`	**Unpack source into working directory**
`do_patch`	**Apply any patches**
`do_configure`	**Perform any necessary pre-build configuration**
`do_compile`	**Compile the source code**
`do_install`	**Installation of resulting build artifacts in WORKDIR**
`do_populate_sysroot`	**Copy artifacts to sysroot**
`do_package_*`	**Create binary package(s)**

Note: to see the list of all possible tasks for a recipe, do this:
`$ bitbake -c listtasks <recipe_name>`

*Simplified for illustration

Simple recipe task list*

chris — sleep — 117×32 — ⌘5

```
$ bitbake hello
NOTE: Running task 337 of 379 (ID: 4, hello_1.0.0.bb, do_fetch)
NOTE: Running task 368 of 379 (ID: 0, hello_1.0.0.bb, do_unpack)
NOTE: Running task 369 of 379 (ID: 1, hello_1.0.0.bb, do_patch)
NOTE: Running task 370 of 379 (ID: 5, hello_1.0.0.bb, do_configure)
NOTE: Running task 371 of 379 (ID: 7, hello_1.0.0.bb, do_populate_lic)
NOTE: Running task 372 of 379 (ID: 6, hello_1.0.0.bb, do_compile)
NOTE: Running task 373 of 379 (ID: 2, hello_1.0.0.bb, do_install)
NOTE: Running task 374 of 379 (ID: 11, hello_1.0.0.bb, do_package)
NOTE: Running task 375 of 379 (ID: 3, hello_1.0.0.bb, do_populate_sysroot)
NOTE: Running task 376 of 379 (ID: 8, hello_1.0.0.bb, do_packagedata)
NOTE: Running task 377 of 379 (ID: 12, hello_1.0.0.bb, do_package_write_ipk)
NOTE: Running task 378 of 379 (ID: 9, hello_1.0.0.bb, do_package_qa)
```

*Output has been formatted to fit this slide.

*Simplified for illustration

SSTATE CACHE

- Several bitbake tasks can be accelerated
- Examples include

do_packagedata	Creates package metadata used by the build system to generate the final packages
do_package	Analyzes the content of the holding area and splits it into subsets based on available packages and files
do_package_write_rpm	Creates the actual RPM packages and places them in the Package Feed area
do_populate_lic	Writes license information for the recipe that is collected later when the image is constructed
do_populate_sysroot	Copies a subset of files installed by do_install into the sysroot in order to make them available to other recipes

Simple recipe build from sstate cache*

chris — sleep — 117×32 — ⌘5

```
$ bitbake -c clean hello
$ bitbake hello
NOTE: Running setscene task 69 of 74 (hello_1.0.0.bb, do_populate_sysroot_setscene)
NOTE: Running setscene task 70 of 74 (hello_1.0.0.bb, do_populate_lic_setscene)
NOTE: Running setscene task 71 of 74 (hello_1.0.0.bb, do_package_qa_setscene)
NOTE: Running setscene task 72 of 74 (hello_1.0.0.bb, do_package_write_ipk_setscene)
NOTE: Running setscene task 73 of 74 (hello_1.0.0.bb, do_packagedata_setscene)
```

*Output has been formatted to fit this slide.

*Simplified for illustration

Quick Start Guide in a Slide

- **Download Yocto Project sources:**
  - `$ wget http://downloads.yoctoproject.org/releases/yocto/yocto-1.7.1/poky-dizzy-12.0.1.tar.bz2`
  - `$ tar xf poky-dizzy-12.0.1.tar.bz2`
  - `$ cd poky-dizzy-12.0.1`
  - Can also use git and checkout a known branch ie. Dizzy (Preferred)
    - `$ git clone git://git.yoctoproject.org/poky.git`
    - `$ cd poky`
    - `$ git checkout -b dizzy`

- **Build one of the reference Linux distributions:**
  - `$ source oe-init-build-env`
  - Check/Edit `local.conf` for sanity
    - Modify `MACHINE=qemuarm`
  - `$ bitbake core-image-sato`

- **Run the image under emulation:**
  - $ runqemu qemux86 [nographic] (for ssh-only sessions, etc.)

Lab 1: Exploring the Host System

Objectives

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

Log into your lab cloud-based host: Instructor will supply URL/credentials

Host System Layout

```
/scratch
  |---downloads (DL_DIR)
      Source cache (downloaded for each recipe)
  |---sandbox
      Working area for your software and mods
  |---sstate-cache  (SSTATE_DIR)
      Common binary cache
  |---working
      Project build directory
  |---yocto
    Yocto sources and caches. Do Not Modify
```

Note: This is your instructor's preferred setup – others may have their own favorite working layout

Host System Layout

```
/scratch/yocto
  |---binary-images
      Pre-built binary boot images for
    convenience
  |---poky
      poky build system Do Not Modifiy
```

Note: This is your instructor's preferred setup – others may have their own favorite working layout

Host System Layout

- **/scratch/yocto/poky**
  - **|---poky** (Yocto baseline dist)

Note: This is your instructor's preferred setup – others may have their own favorite working layout

Poky Layout

- /scratch/yocto/poky
  - |---bitbake (The BitBake application)
  - |---documentation
  - |---meta (OE-CORE)
  - |---meta-yocto (Yocto distro policy)
  - |---meta-yocto-bsp (Yocto reference BSPs)
  - |---oe-init-build-env (project setup utility)
  - |---README
  - |---README.hardware
  - |---scripts (Various helper scripts)

Note: some items omitted for simplicity

Exercise 2: Examining Recipes

- **Objectives:**
  - Familiarize yourself with recipe basics, and where they are found in the build system hierarchy
  - Examine representative techniques in different recipes

Exercise 2: Examining Recipes

- **Look at 'bc' recipe:**
- **Found in `/scratch/yocto/poky/meta/recipes-extended/bc/bc_1.06.bb`**
  - Uses LIC_FILES_CHKSUM and SRC_URI checksums
  - Note the DEPENDS build dependency declaration indicating that this package depends on `flex` to build

Exercise 2: Examining Recipes

- **Look at 'flac' recipe**
- **Found in `/scratch/yocto/poky/meta/recipes-multimedia/flac/flac_1.3.0.bb`**
  - Includes custom source patches to apply to the sources
  - Customizes autoconf configure options (`EXTRA_OECONF`) based on "TUNE" features
  - Breaks up output into multiple binary packages

Exercise 2: Examining Recipes

- **Look at 'ofono' recipe(s):**
- **Found in `/scratch/yocto/poky/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

Pause here

- **Review concepts**

Exercise 3: Building a Linux Image

- **Objectives:**
  - Create a project directory using poky scripts
  - Configure the build
    - Select appropriate MACHINE
    - Setup paths to sstate and source cache

Exercise 3: Building a Linux Image

- **General Procedure:**
  - Create a project directory using poky scripts
  - Configure the build by editing '`local.conf`'
    - Select appropriate MACHINE
    - Setup paths to sstate and source cache

Exercise 3: Building a Linux Image

- **Execute the following commands to create the project directory:**

```
$ cd /scratch/working
```

Don't miss the 'dot', a shorthand for the 'source' command

```
$ . /scratch/yocto/poky/oe-init-build-env build_qemuarm
```

(This script sets up path to bitbake, builds default project directory)

Exercise 3: Building a Linux Image

- **Edit the main project configuration file: `local.conf`**

- **Set `MACHINE = "qemuarm"` in `conf/local.conf`**
  - Specifies that we're building for the qemuarm target
  - Set up sstate-cache and source cache (downloads)
  - `DL_DIR = "/scratch/downloads"`
  - `SSTATE_DIR = "/scratch/sstate-cache"`

Example in next slide

Exercise 3: Building a Linux Image

- Your `local.conf` should look like this:

chris — sleep — 117×32 — ⌘5

```
#
# Qemu configuration
#
# By default qemu will build with a builtin VNC server where graphical output can be
# seen. The two lines below enable the SDL backend too. This assumes there is a
# libsdl library available on your build system.
PACKAGECONFIG_append_pn-qemu-native = " sdl"
PACKAGECONFIG_append_pn-nativesdk-qemu = " sdl"
ASSUME_PROVIDED += "libsdl-native"


# CONF_VERSION is increased each time build/conf/ changes incompatibly and is used to
# track the version of this file when it was generated. This can safely be ignored if
# this doesn't mean anything to you.
CONF_VERSION = "1"

MACHINE = "qemuarm"
DL_DIR = "/scratch/downloads"
SSTATE_DIR = "/scratch/sstate-cache"
```

Add these lines at end of file

Exercise 3: Building a Linux Image

- **Now build the image (builds an entire embedded Linux distribution!)**

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

  - Builds a reference image for the qemuarm target

- **If everything is configured correctly, your build should take less than 5 minutes**

Exercise 4: Booting Your Board

- **Objectives:**
  - Familiarize your self with running QEMUunder Yocto
  - Get a bootable QEMU instance using the kernel and root file system you build in the previous exercise.

Exercise 4: Booting Your Image with QEMU

- **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 qemuarm [nograpic]`

  - Use `nographic` if using a non-graphical session, do not type the square brackets

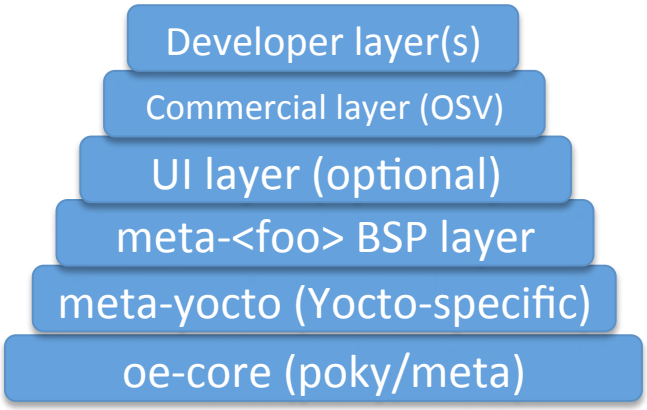
- **Your QEMU instance should boot**
- **Kill it using another terminal: `killall qemu-system-arm`**

LAYERS AND MORE

This section will introduce the concept of layers and how important they are in the overall Yocto and Openembedded architecture

Layers Agenda

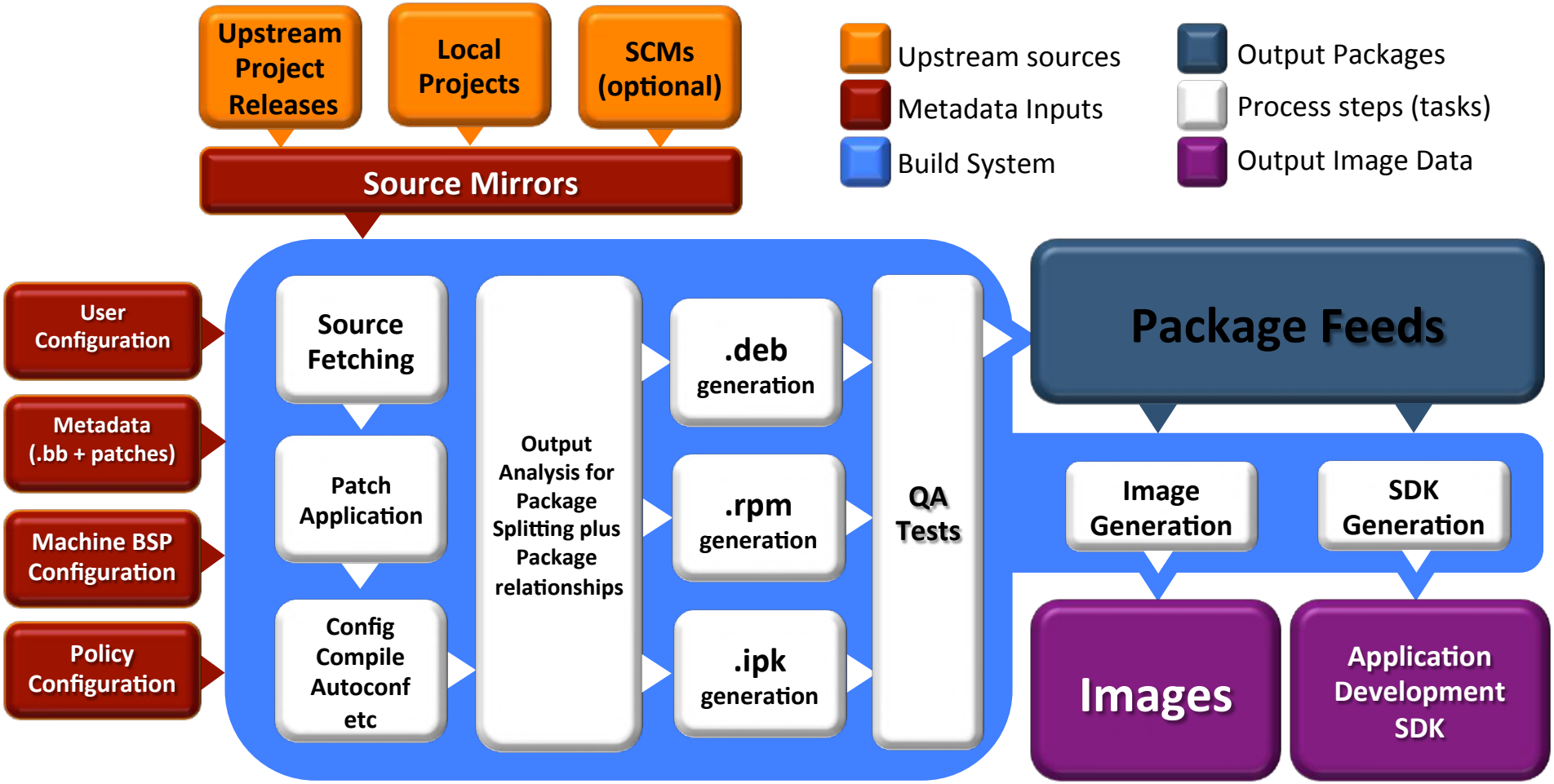
- Introduction to Layers
- Stacking Customizations
- Adding Layers
- Board Support Packages



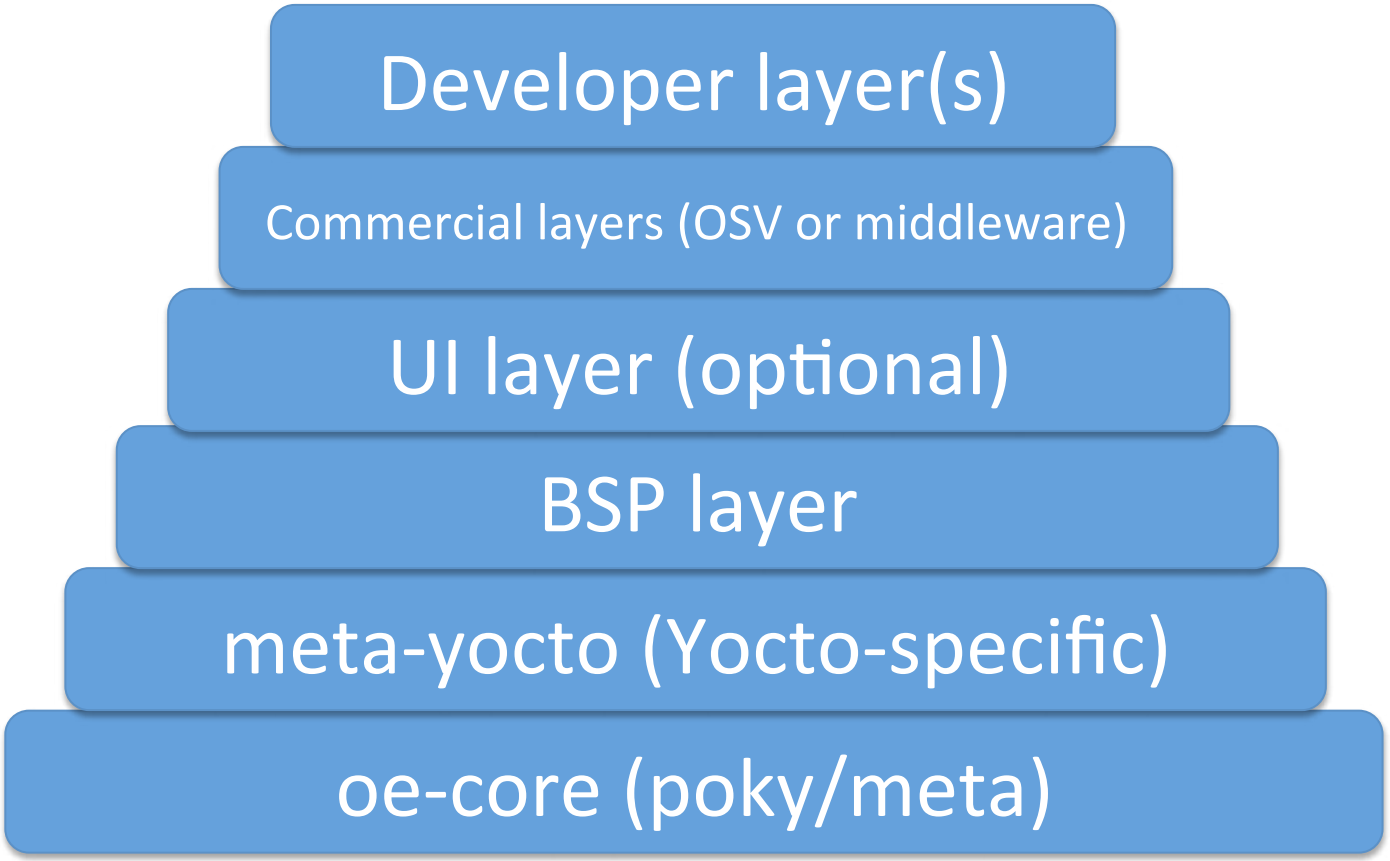
Layers

- 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 with a lesser priority

Build System Workflow



Layer Hierarchy



Using Layers

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

```
BBLAYERS ?= " \
      /scratch/yocto/poky/meta \
      /scratch/yocto/poky/meta-yocto \
      /scratch/yocto/meta-yocto-bsp \
      "
```

Board Support Packages

- **BSPs are layers to enable support for specific hardware platforms**
- **Defines machine configuration variables for the board (machine)**
- **Adds machine-specific recipes and customizations**
  - Kernel config
  - Graphics drivers (e.g, Xorg)
  - Additional recipes to support hardware features

Notes on using Layers

- When doing development with Yocto, do not edit files within the Poky source tree – use a custom layer for modularity and maintainability
- Create a new layer or layers to contain your customizations

EXERCISE 5:
CREATE CUSTOM LAYER

Exercise 5: Create a Custom Layer

- **Objectives:**
  - Become familiar with OE layer structure
  - Create a custom layer and integrate it into your build
  - Using the layer to add your own custom image recipe

Exercise 5: Create a Custom Layer

- **Create a new layer to contain your customizations in `/scratch/sandbox`**
  - Let's call this layer `meta-ypdd`
- **This layer must include:**
  - `layer.conf` file which tells bitbake what kind of files are contained in the layer and defines other layer attributes
  - At least one `recipes-*` directory
  - A `README` file (basic documentation for the layer, including maintainer info, dependencies)

Exercise 5: Create a Custom Layer

- Create a new layer called `meta-ypdd`

```
$ mkdir -p /scratch/sandbox/meta-ypdd/conf
$ mkdir /scratch/sandbox/meta-ypdd/recipes-core
```

- **Create a layer.conf for your new layer**
  - Contents in next slide

meta-ypdd/conf/layer.conf

```
$ vi /scratch/sandbox/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:
CREATE A CUSTOM IMAGE RECIPE

Exercise 6: Custom Image Recipe

- **Objective:**
  - Become familiar with image recipe basics
  - Create custom image recipe

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-core/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:

- **Create an `images` directory:**

```
$ mkdir /scratch/sandbox/meta-ypdd/
recipes-core/images
```

Exercise 6: Creating a Custom Image Recipe

```
$ vi /scratch/sandbox/meta-ypdd/recipes-core/images/ypdd-image.bb
```

chris — ilab01@devday00: /scratch/working/build-qemuarm — ssh — 73×21 — ⌘1

```
DESCRIPTION = "A core image for YPDD"
LICENSE = "MIT"

# Core files for basic console boot
IMAGE_INSTALL = "packagegroup-core-boot"

# Add our desired extra files
IMAGE_INSTALL += "psplash dropbear"

inherit core-image

IMAGE_ROOTFS_SIZE ?= "8192"
~
~
~
~
~
~
~
~
~
                                                    7,29          All
```

- *This is your image recipe*

EXERCISE 7:
BUILD AND BOOT YOUR NEW IMAGE

Exercise 7: Build/Boot Image

- **Objectives:**
  - Become familiar with how to add layers to your build
  - Build your new image recipe
  - Boot QEMU using your new image recipe created in the previous exercise

Exercise 7: Build and Boot Your Custom Image

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

  (example in the next slide)

Add your layer to bblayer.conf

```
$ vi /scratch/working/build-qemuarm/conf/bblayers.conf
```

chris — ilab01@devday00: /scratch/working/build-qemuarm — ssh — 73×21 — ⌘1

```
# changes incompatibly
LCONF_VERSION = "6"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
  /scratch/sandbox/meta-ypdd \
  /scratch/yocto/poky/meta \
  /scratch/yocto/poky/meta-yocto \
  /scratch/yocto/poky/meta-yocto-bsp \
  "
BBLAYERS_NON_REMOVABLE ?= " \
  /scratch/yocto/poky/meta \
  /scratch/yocto/poky/meta-yocto \
  "
~
~
~
~
-- INSERT --                                   9,31          Bot
```

Add this line

Exercise 7: Build and Boot Your Custom Image

- **Build your custom image:**

```
$ bitbake ypdd-image
```

(If everything is configured correctly, this should take less than 5 minutes)

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

Exercise 7: Build/Boot Custom Image

- Verify that dropbear ssh server is present

```
$ which dropbear
```

- If you used the graphical invocation of QEMU using VNC viewer, you will see the splash screen on boot.

EXERCISE 8: DEVELOP AND INTEGRATE CUSTOM APPLICATION

Use "hello world" example to add application to your new image

Exercise 8: Add Application

- **Objectives:**
  - Gain familiarity with process for integrating a new recipe (application) into your custom image
  - Develop hello world recipe and application
  - Deploy to your image containing your new 'hello' application

Exercise 8: Add Application

- **General procedure:**
  - Write hello world application (hello.c)
  - Create recipe for hello world application
  - Modify image recipe to add hello world application to your image

Exercise 8: Add App

- Create hello app, place it into a directory called "files" under your hello directory

```
$ mkdir -p /scratch/sandbox/meta-ypdd/recipes-core/hello/files
```

```
$ vi /scratch/sandbox/meta-ypdd/recipes-core/hello/files/hello.c
```

Exercise 8: Add app

```
$ vi /scratch/sandbox/meta-ypdd/recipes-core/hello/files/hello.c
```

chris — sleep — 117×32 — ⌘5

```
#include <stdio.h>

int main(int argc, char **argv) {
        printf("Hello World\n");
        return 0;
}
```

Exercise 8: Add Application

- Write hello world recipe
- Create directory to hold the recipe and associated files

```
$ mkdir -p /scratch/sandbox/meta-ypdd/recipes-core/hello/files
```

(already done in previous step)

- Generate hello.bb (next slide)

```
$ vi /scratch/sandbox/meta-ypdd/recipes-core/hello/hello_1.0.0.bb
```

Exercise 8: Add App

```
$ vi /scratch/sandbox/meta-ypdd/recipes-core/hello/hello_1.0.0.bb
```

chris — sleep — 117×32 — ⌘5

```
DESCRIPTION = "Hello World example"
LICENSE = "MIT"

LIC_FILES_CHKSUM = "file://${COREBASE}/meta/
COPYING.MIT;md5=3da9cfbcb788c80a0384361b4de20420"

SRC_URI = "file://hello.c"

do_compile() {
        ${CC} hello.c -o hello
}

do_install() {
        install -d -m 0755 ${D}/${bindir}
        install -m 0755 hello ${D}/${bindir}/hello
}
```

Exercise 8: Add Application

- Modify image recipe to add hello world application to your image
- See example on next slide

Exercise 8: Add app

```
$ vi /scratch/sandbox/meta-ypdd/recipes-core/images/ypdd-image.bb
```

chris — sleep — 117×32 — ⌘5

```
DESCRIPTION = "A core image for YPDD"
LICENSE = "MIT"
IMAGE_INSTALL = "packagegroup-core-boot"
IMAGE_INSTALL += "psplash dropbear hello"

inherit core-image
IMAGE_ROOTFS_SIE ?= "8192"
```

Add the package 'hello' to your image recipe

Exercise 8: Add app

- **Now build your image recipe**

```
$ bitbake ypdd-image
```

  - `hello_1.0.0.bb` will be processed because it is in your custom layer, and referenced in your image recipe.

- **Boot your image using runqemu, as before:**

```
$ runqemu qemuarm tmp/deploy/images/qemuarm/ypdd-image-qemuarm.ext3 nographic
```

- **You should be able to type "hello" at the command line and see "Hello World"**

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, it's results are cumulative 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.org
- Bug reports and feature requests
- http://bugzilla.yoctoproject.org

Tip: ack-grep

- **Much faster than grep for the relevant use cases**
- **Designed for code search**
- **Searches only relevant files**
  - Knows about many types: C, asm, perl
  - By default, skips .git, .svn, etc.
  - Can be taught arbitrary types
- **Perfect for searching metadata**
- **`$ bback` (it's on your vm)**

Tip: ack-grep

chris@speedy: ~ — ssh — 69×20

```
chris@speedy 11:34 AM /build/intro-lab/poky-dylan-9.0.2
$ bback "SRC_URI ="
documentation/ref-manual/examples/h
6:SRC_URI = "${GNU_MIRROR}/hello/he

documentation/ref-manual/examples/hello-single/hello.bb
6:SRC_URI = "file://helloworld.c"

documentation/ref-manual/examples/mtd-makefile/mtd-utils_1.0.0.bb
9:SRC_URI = "ftp://ftp.infradead.org/pub/mtd-utils/mtd-utils-${PV}.ta
r.gz"

meta/classes/bin_package.bbclass
15:# SRC_URI = "http://foo.com/foo-1.0-r1.i586.rpm;subdir=foo-1.0"

meta/classes/externalsrc.bbclass
20:SRC_URI = ""

meta/classes/gnomebase.bbclass
8:SRC_URI = "${GNOME_MIRROR}/${BPN}/${@gnome_verdir("${PV}")}/${BPN}-
```

```
alias bback='ack-grep --type bitbake'
```

TIP: VIM Syntax Highlighting

- https://github.com/openembedded/bitbake/tree/master/contrib/vim
- Install files from the above repo in ~/.vim/
- Add "syntax on" in ~/.vimrc

- $ tree ~/.vim/
- /Users/chris/.vim/
- ├── ftdetect
- │ └── bitbake.vim
- ├── ftplugin
- │ └── bitbake.vim
- ├── plugin
- │ └── newbb.vim
- └── syntax
- └── bitbake.vim
- You do use VI, right? ;-)

It's on your VM!

TIP: VIM Syntax Highlighting

chris@speedy: ~ — ssh — 80×24

```
SUMMARY = "The basic file, shell and text manipulation utilities."
DESCRIPTION = "The GNU Core Utilities provide the basic file, shell andd
 text \
manipulation utilities. These are the core utilities which are expectedd
 to exist on \
every system."
HOMEPAGE = "http://www.gnu.org/software/coreutils/"
BUGTRACKER = "http://debbugs.gnu.org/coreutils"
LICENSE = "GPLv3+"
LIC_FILES_CHKSUM = "file://COPYING;md5=d32239bcb673463ab874e80d47fae5044
\
                    file://src/ls.c;beginline=5;endline=16;md5=38b797855
ca88537b75871782a2a3c6b8"
PR = "r0"
DEPENDS = "gmp libcap"
DEPENDS_class-native = ""

inherit autotools gettext

SRC_URI = "${GNU_MIRROR}/coreutils/${BP}.tar.xz \
           file://remove-usr-local-lib-from-m4.patch \
           file://coreutils-build-with-acl.patch \
           file://dummy_help2man.patch \
                                                  1,1           Top
```

It's not an embedded Linux distribution

It creates a custom one for you