

Yocto Project and Embedded OS

Kevin McGrath

- Using Yocto cross-compiler
- Running kernel via qemu
- Module installation, virtio, etc.
- Lessons learned, capabilities

Jeffrey Osier-Mixon

- What is the Yocto Project and why is it important?
- Working with an open source collaborative project & community
- Yocto Project concepts in a nutshell: environment, metadata, tools

July 28th 2015
11:00 PDT (GMT -7)



Yocto Project and Embedded OS

Our guests

Jeffrey Osier-Mixon:

Jeff "Jefro" Osier-Mixon works for Intel Corporation in Intel's Open Source Technology Center, where his current role is community manager for the Yocto Project.. Jefro also works as a community architect and consultant for a number of open source projects and speaks regularly at open source conferences worldwide. He has been deeply involved with open source since the early 1990s.

Kevin McGrath :

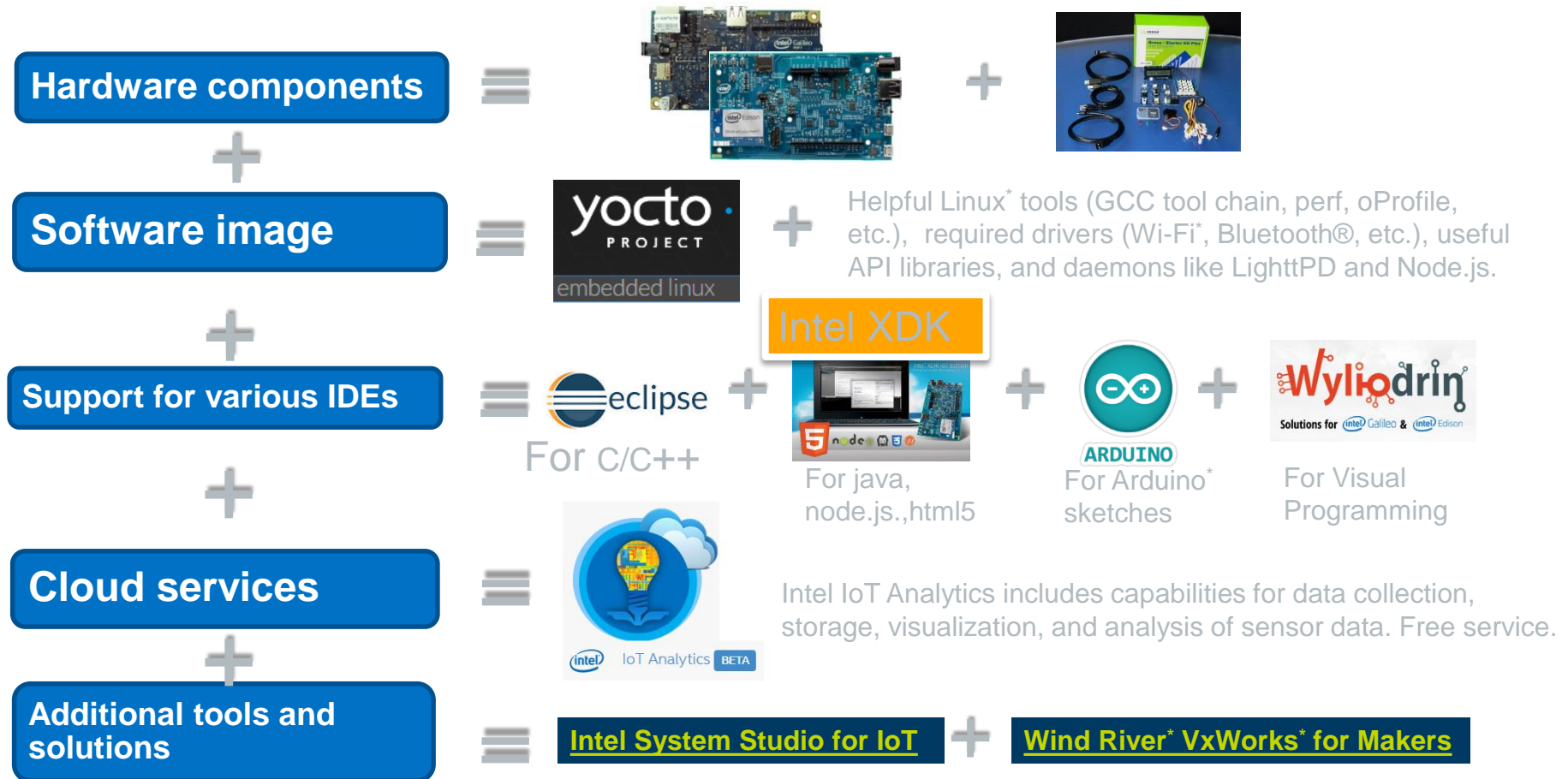
Instructor at Oregon State University. I primarily teach the operating systems sequence and the senior capstone project sequence, but have taught architecture, assembly programming, introductory programming classes, and just about anything else that needs someone to teach it. While my background is in network security and high performance computing (computational physics), today I mostly live in the embedded space, leading to the "ECE wannabe" title in my department.

Oleg Verge (Moderator):

Technical Program Manager Intel Higher Education, System Engineer MCSE,CCNA, VCP



Intel® IoT Developer Kit v1.0






Introduction to the Yocto Project

Accelerating Embedded Product Development

Jeff "Jefro" Osier-Mixon, Intel Corp
Yocto Project Community Manager



[yoc-to]

The smallest unit of measure,
equal to one septillionth (10^{-24}).



The Yocto Project Ecosystem

What it is, who we are, and why you should care...



**The Yocto Project is not an Embedded Linux Distribution.
It creates a custom one for You!**

**The Yocto Project is not Single Open Source Project.
It is an Ecosystem.**

**The Yocto Project combines the convenience of a ready-to-run
Linux Distribution with the flexibility of a custom Linux operating
system stack.**

What is the Yocto Project?

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

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



What the Yocto Project Provides

- The industry needed a common build system and core technology
 - Bitbake and OpenEmbedded Core = OE 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 & 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



Member Organizations



Supporting Organizations



<http://www.yoctoproject.org/ecosystem>

Why Should a Developer Care? (1)

- **Build a complete Linux system –from source– in about an hour (about 90 minutes with X)**
 - Multiple cores (i.e. quad i7)
 - Lots of RAM (i.e. 16 GB of ram or more)
 - Fast disk (RAID, SSD, etc...)
- **Start with a validated collection of software (toolchain, kernel, user space)**
- **Blueprints to get you started quickly and that you can customize for your own needs**
- **We distinguish app developers from system developers and we support both**
- **Access to a great collection of app developer tools (performance, debug, power analysis, Eclipse)**

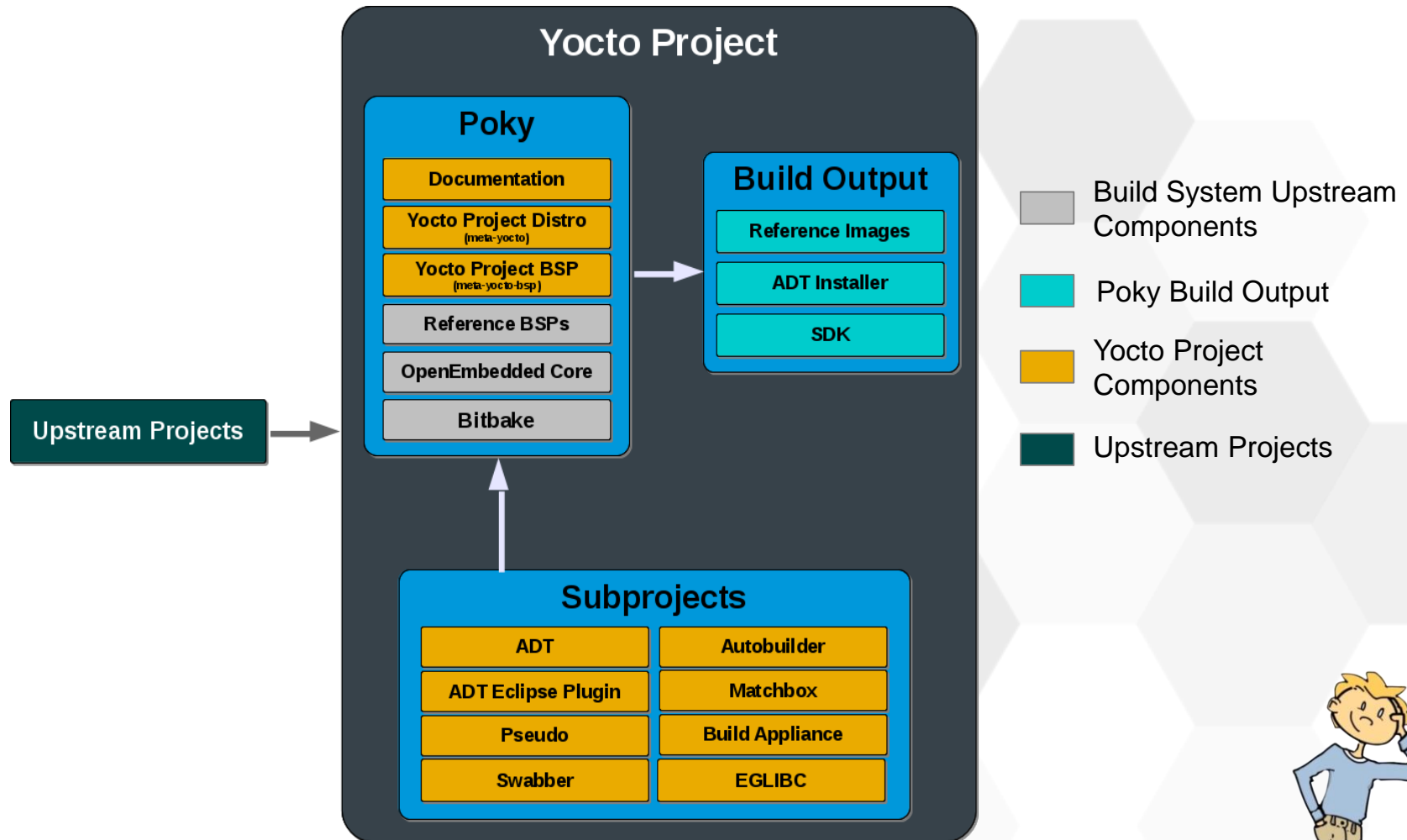


Why Should a Developer Care? (2)

- **Supports all major embedded architectures**
 - x86, x86-64, ARM, PPC, MIPS
 - Coming soon, MIPS64, ARM Arch 64, PPC64
- **Advanced kernel development tools**
- **Layer model encourages modular development, reuse, and easy customizations**
- **Compatibility program that is used to encourage interoperability and best practices**



Yocto Project Provides Embedded Tools, Best Practices, and Reference Implementation



Yocto Project and OpenEmbedded

- **OpenEmbedded**

- Created by merging the work of the OpenZaurus project with contributions from other projects such as Familiar Linux and OpenSIMpad into a common code base
- Community project focusing on broad hardware and architectures
- Large library of recipes to cross-compile over 1000 packages
- Switched from flat meta-data architecture (OpenEmbedded Classic) to layered architecture based on OpenEmbedded Core layer, which is in common with the Yocto Project and the Angstrom Distribution

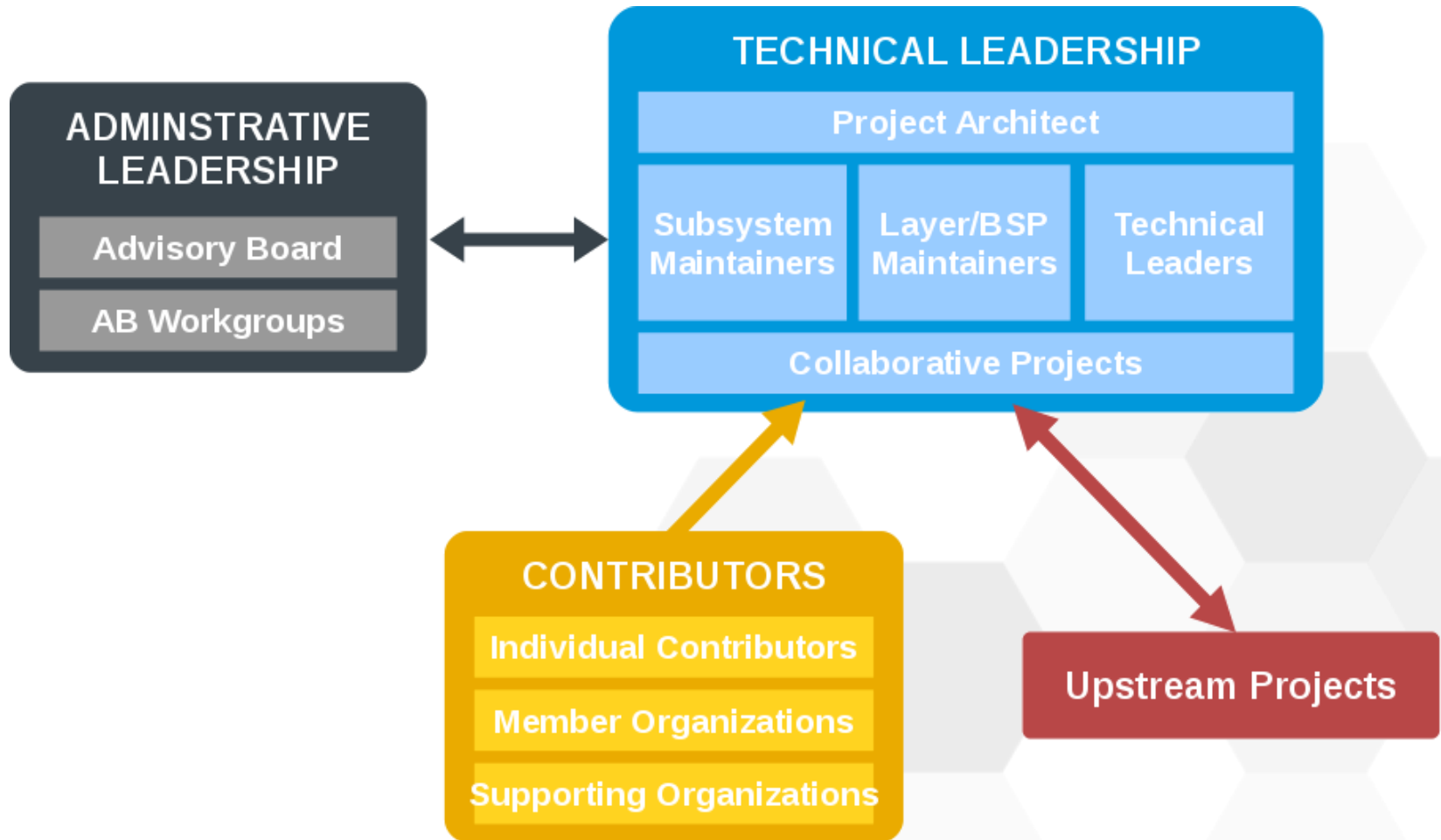
- **Yocto Project**

- Family of projects for developing Linux-based devices
- Self-contained build environment providing tools and blueprints for building Linux OS stacks
- Supported by silicon vendors, OSVs (also providing commercial support), open source projects for hardware and software, electronics companies
- Standardized components with compliance program
- Focused on tooling and maintenance, major release every 6 months

Why not just use OpenEmbedded?

- **OpenEmbedded is an Open Source Project providing a Build Framework for Embedded Linux Systems**
 - Not a reference distribution
 - Designed to be the foundation for others
 - Cutting-edge technologies and software packages
- **The Yocto Project is focused on enabling Commercial Product Development**
 - Provides a reference distribution policy and root file system blueprints
 - Co-maintains OpenEmbedded components and improves their quality
 - Provides additional tooling such as Autobuilder, QA Tests
 - Provides tools for application development such as ADT and Eclipse Plugin

The Yocto Project Community





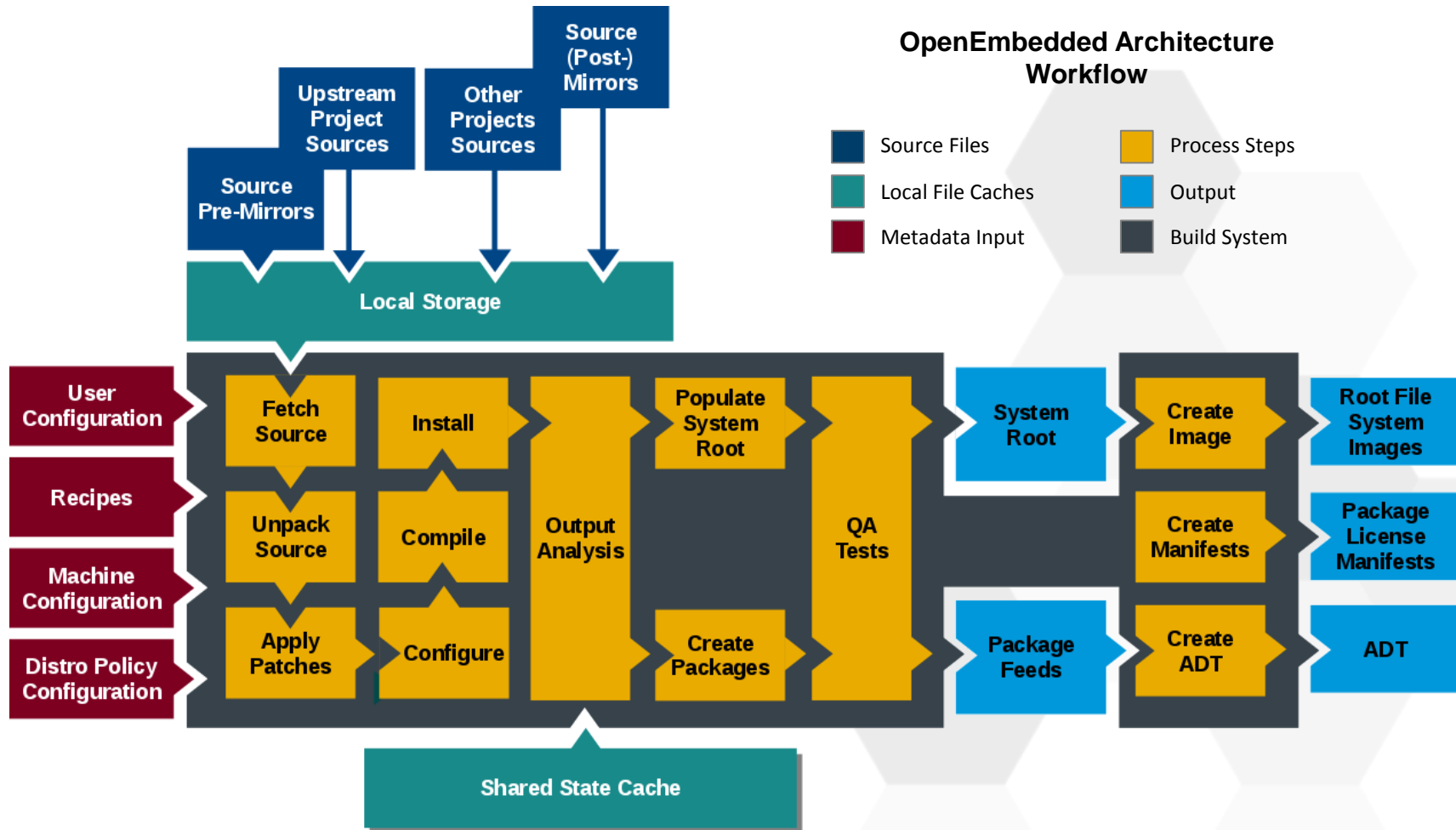
Getting Started

**All you need to know to get your feet wet and a little
beyond...**

Quick Start

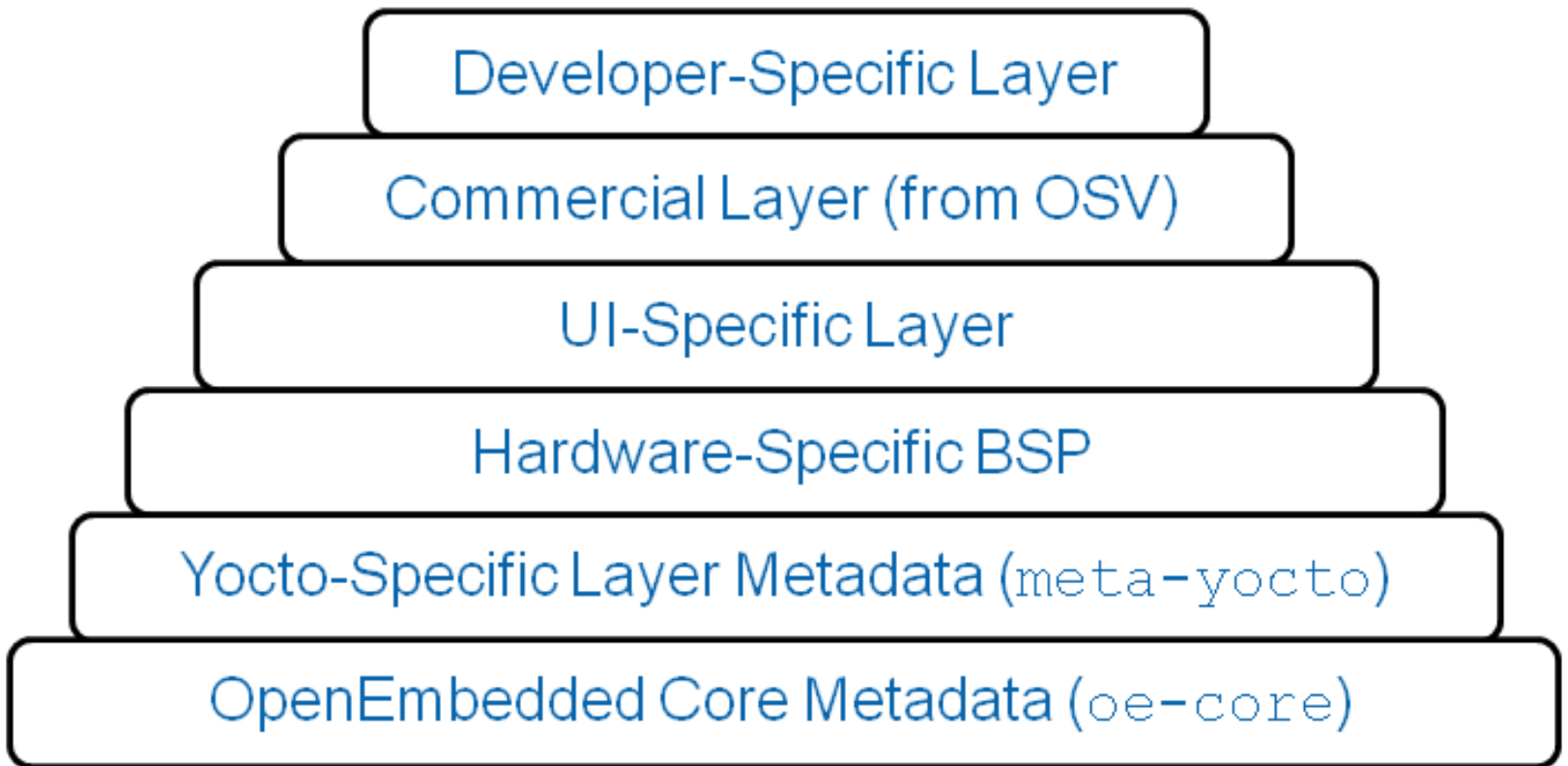
1. Go to <https://www.yoctoproject.org> – click “Documentation” and read the Quick Start guide
2. Set up your Linux build system with the necessary packages (and firewall as needed)
3. Go to <http://www.yoctoproject.org> click “downloads” and download the latest stable release (Yocto Project 1.6.1 “Daisy” 11.0.1) – 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 qemu86` (if `MACHINE=qemu86`)

Build System Workflow



Layers (1)

The build system is composed of layers

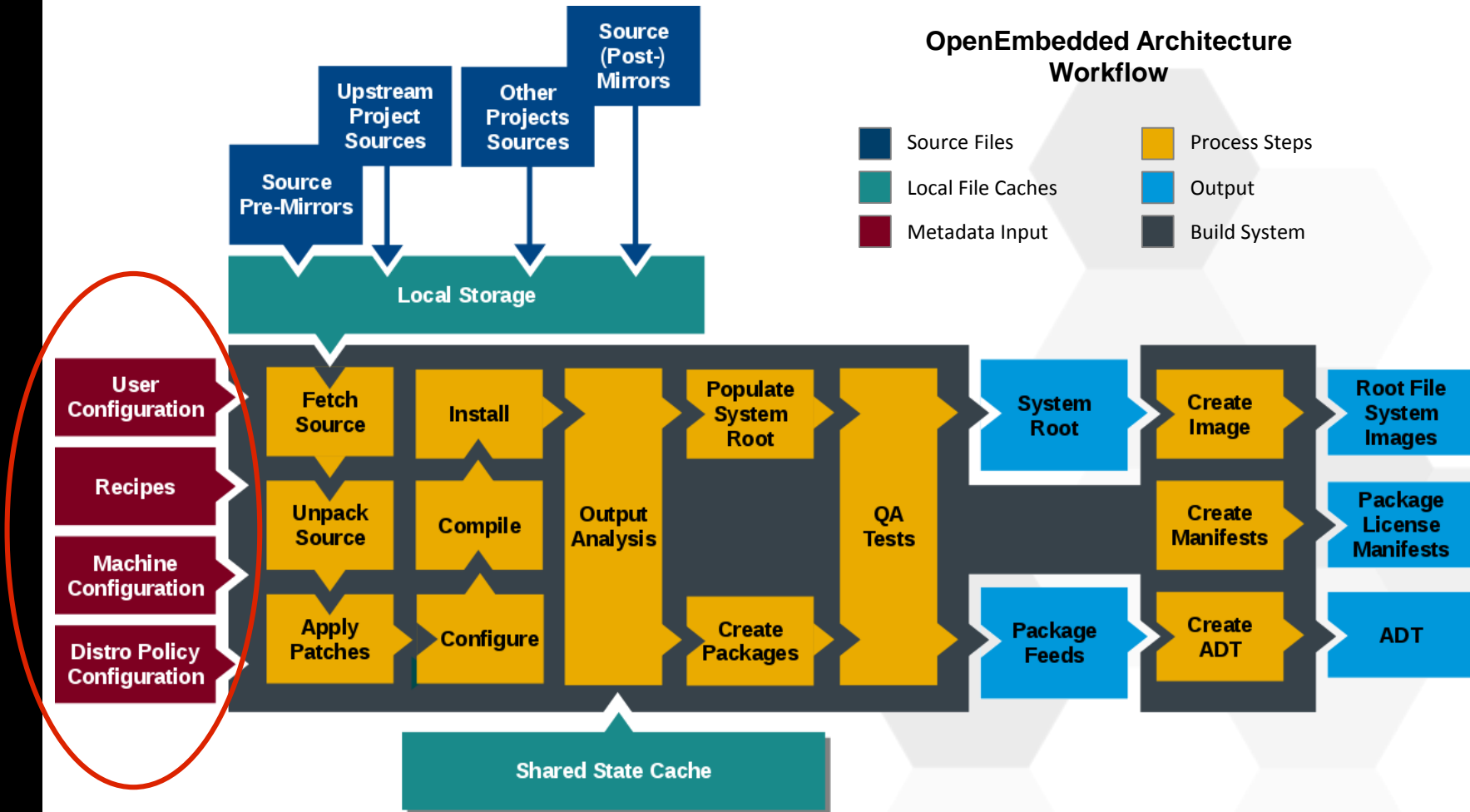


Layers (2)

- **Layers are a way to manage extensions, and customizations to the system**
 - Layers can extend, add, replace or modify recipes
 - Layers can add or replace bbclass files
 - Layers can add or modify configuration settings
 - Layers are added via BBLAYERS variable in build/conf/bblayers.conf
- **Best Practice: Layers should be grouped by functionality**
 - Custom Toolchains (compilers, debuggers, profiling tools)
 - Distribution specifications (i.e. meta-yocto)
 - BSP/Machine settings (i.e. meta-yocto-bsp)
 - Functional areas (selinux, networking, etc)
 - Project specific changes

All starts with the Configuration

OpenEmbedded Architecture Workflow



Configuration

Configuration files (*.conf) – global build settings

**User
Configuration**

Recipes

**Machine
Configuration**

**Distro Policy
Configuration**

- meta/conf/bitbake.conf (defaults)
- build/conf/bblayers.conf (layers)
- */conf/layers.conf (one per layer)
- build/conf/local.conf (local user-defined)
- meta-yocto/conf/distro/poky.conf (distribution policy)
- meta-yocto-bsp/conf/machine/beagleboard.conf (BSP)
- meta/conf/machine/include/tune-cortexa8.inc (CPU)
- Recipes (metadata)

User Configuration

User
Configuration

Recipes

Machine
Configuration

Distro Policy
Configuration

build/conf/local.conf is where you override and define what you are building

- BB_NUMBER_THREADS and PARALLEL_MAKE
- MACHINE settings
- DISTRO settings
- INCOMPATIBLE_LICENSE = "GPLv3"
- EXTRA_IMAGE_FEATURES

build/conf/bblayers.conf is where you configure with layers to use

- Add compatible layers to BBLAYERS
- Default: meta (oe-core), meta-yocto and meta-yocto-bsp

Recipes

User
Configuration

Recipes

Machine
Configuration

Distro Policy
Configuration

Build Instructions

- Recipes for building packages
- Recipe Files
 - meta/recipes-core/busybox_1.20.2.bb

Patches and Supplemental Files

- Location
 - meta/recipes-core/busybox/busybox-1.20.2

Recipes inherit the system configuration and adjust it to describe how to build and package the software

Recipes can be extended and enhanced through append-files from other layers

Yocto Project and OpenEmbedded recipes structures are compatible to each other

Machine Configuration

User
Configuration

Recipes

Machine
Configuration

Distro Policy
Configuration

Configuration files that describe a machine

- Define board specific kernel configuration
- Formfactor configurations
- Processor/SOC Tuning files

Hardware machines and emulated machines (QEMU)

For example:

`meta-yocto-bsp/conf/machine/beagleboard.conf`

Machine configuration refers to kernel sources and may influence some userspace software

Compatible with OpenEmbedded

Distribution Policy

Defines distribution/system wide policies that affect the way individual recipes are built

User
Configuration

Recipes

Machine
Configuration

Distro Policy
Configuration

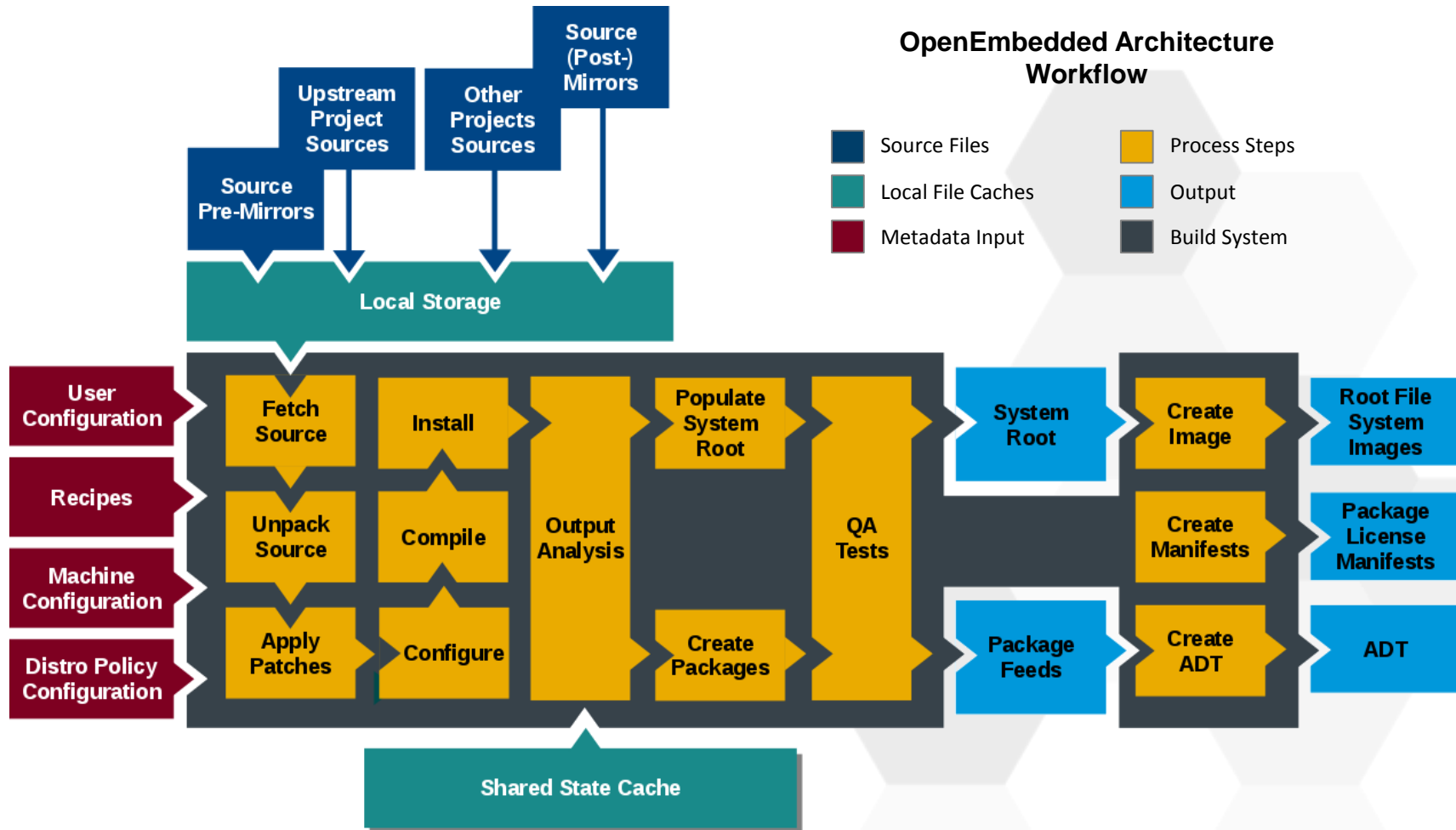
- May set alternative preferred versions of recipes
- May enable/disable LIBC functionality (i.e. i18n)
- May enable/disable features (i.e. pam, selinux)
- May configure specific package rules
- May adjust image deployment settings

Enabled via the DISTRO setting

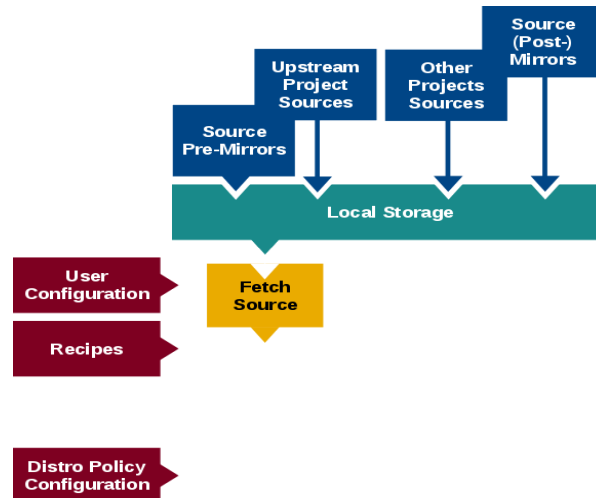
Four predefined settings

- poky-bleeding: Enable a bleeding edge packages
- poky: Core distribution definition, defines the base
- poky-lsb: enable items required for LSB support
- poky-tiny: construct a smaller then normal system

How does it work? In-depth build process



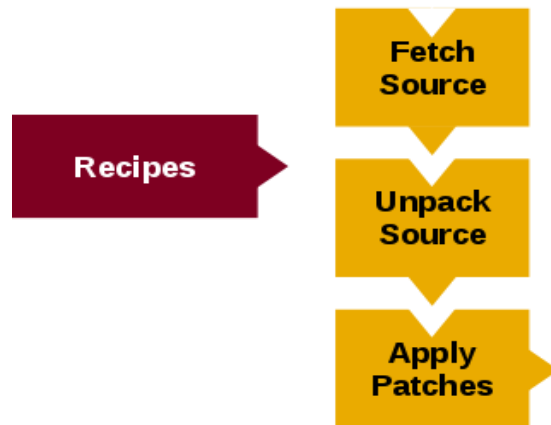
Source Fetching



- 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, bzip, tarballs, and many more*
- Versions of packages can be fixed or updated automatically (Add SRCREV_pn-PN = "\${AUTOREV}" to local.conf)
- The Yocto Project mirrors sources to ensure source reliability

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

Source Unpacking and 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
- Patches are documented according to the patch guidelines: http://www.openembedded.org/wiki/Commit_Patch_Message_Guidelines

Configure / Compile / Install

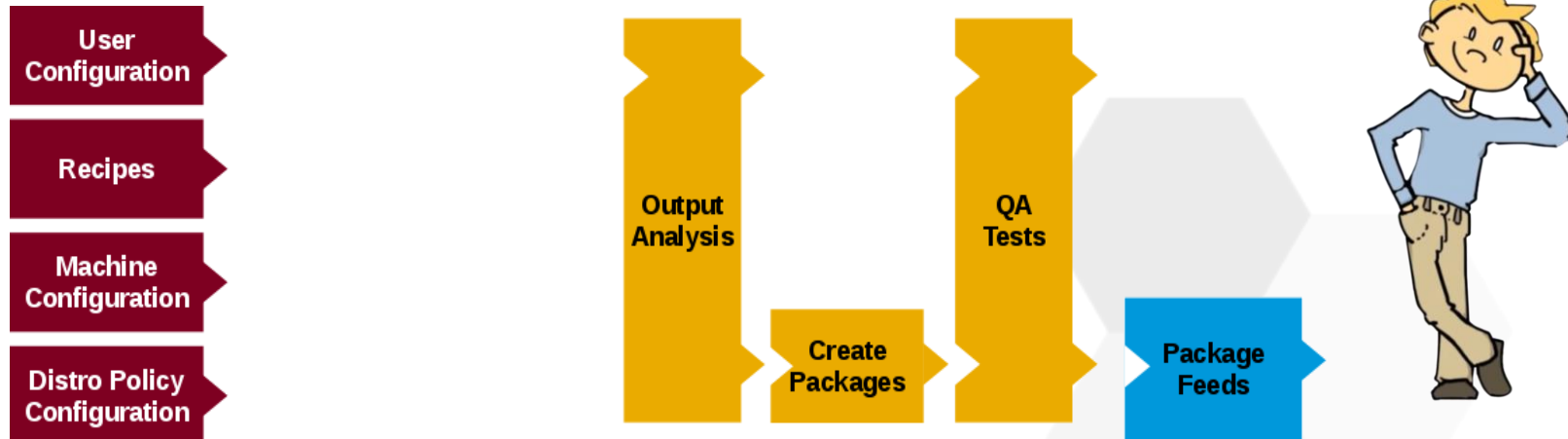


- **Recipe specifies configuration and compilation rules**
 - Various standard build rules are available, such as autotools and gettext
 - Standard ways to specify custom environment flags
 - Install step runs under 'pseudo', allows special files, permissions and owners/groups
- **Recipe example**

```
SUMMARY = "GNU Helloworld Application"
SECTION = "examples"

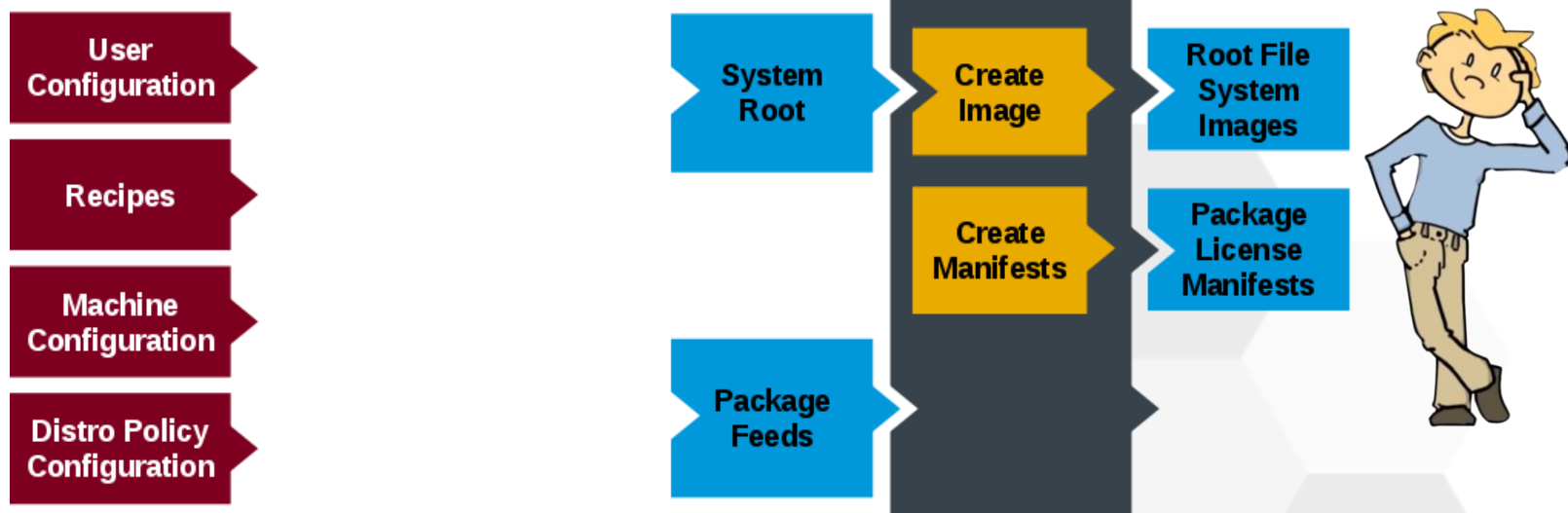
LICENSE = "GPLv2+"
LIC_FILES_CHKSUM = "file://COPYING;md5=751419260aa954499f7abaabaa882bbe"
PR = "r0"
SRC_URI = "${GNU_MIRROR}/hello/hello-${PV}.tar.gz"
inherit autotools gettext
```

Output Analysis / Packaging



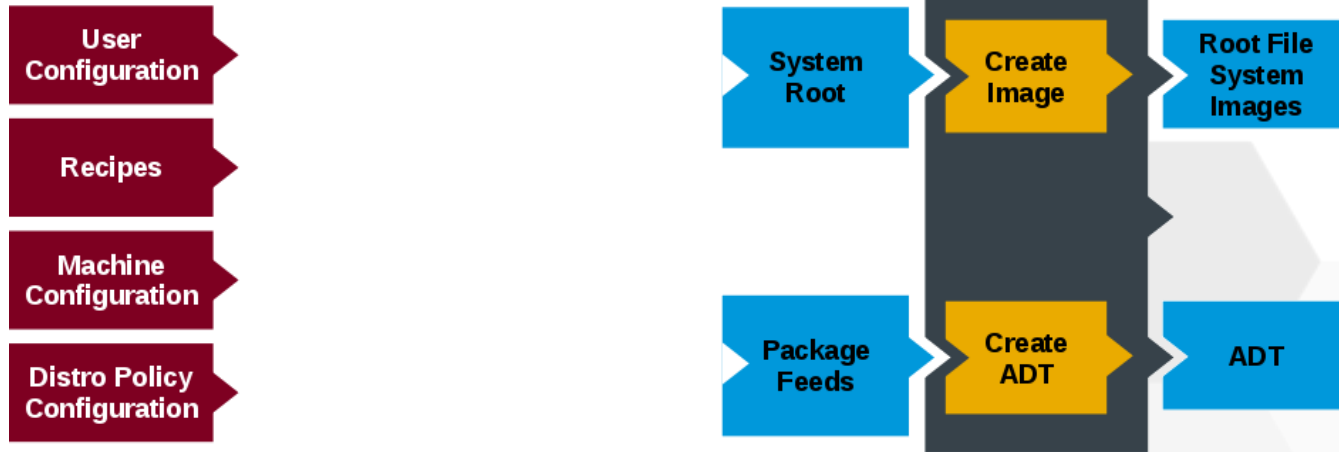
- **Output Analysis:**
 - Categorize generated software (debug, dev, docs, locales)
 - Split runtime and debug information
- **Perform QA tests (sanity checks)**
- **Package Generation:**
 - Support the popular formats, RPM, Debian, and ipk
 - Set preferred format using `PACKAGE_CLASSES` in `local.conf`
 - Package files can be manually defined to override automatic settings

Image Generation



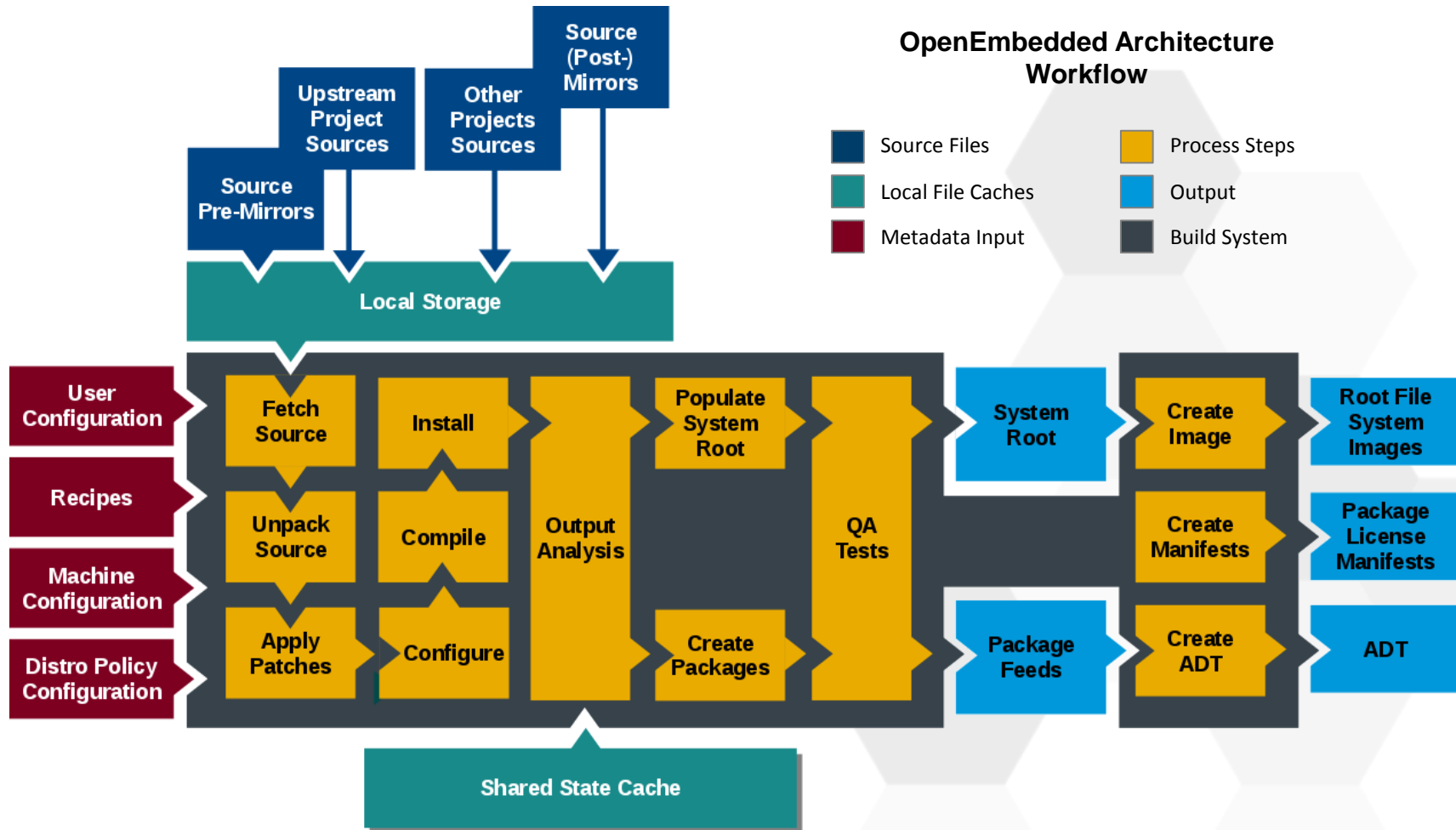
- Images are constructed using the packages built earlier and put into the Package Feeds
- Decisions of what to install on the image is based on the minimum defined set of required components in an image recipe. This minimum set is then expanded based on dependencies to produce a package solution.
-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

SDK Generation



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

Build System Workflow



Application Development Toolkit

Core Components

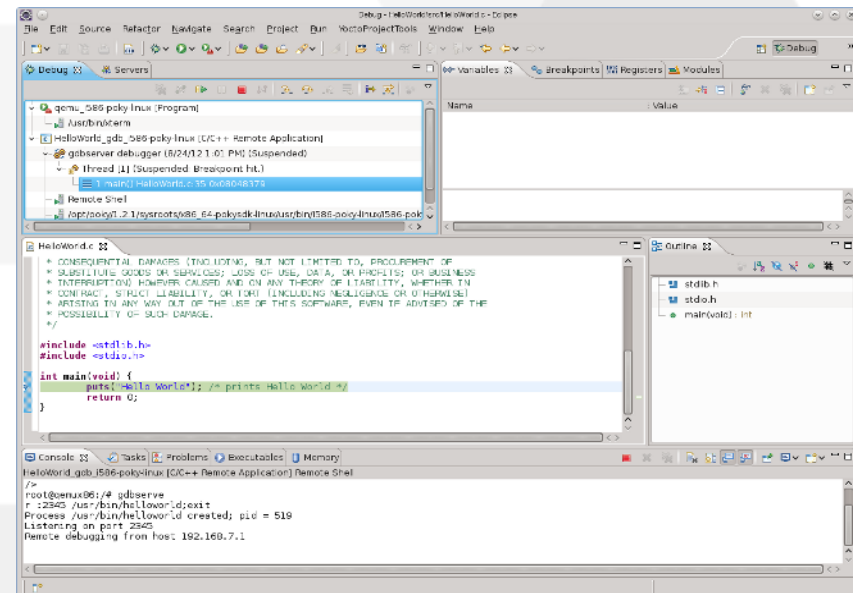
- Cross Development Toolchain
- System Root
- QEMU Emulator

Eclipse Plugin

- Roundtrip Application Development
- Toolchain/System Root Integration
- Emulated and Hardware Targets
- Application Templates
- On-target Debugging

- **Profiling Tools**

- LatencyTOP
- PowerTOP
- Oprofile
- Perf
- SystemTap
- Lttng-ust



The Takeaway

Embedded Systems are Diverse

- Unless you are using standard hardware you will have to adapt and build your own operating system stack.
- Building and maintaining every aspect of an OS stack requires a lot of expertise and resources.

The Yocto Project

- Provides a self-contained and rigorously tested build environment with tools, recipes and configuration data to build custom Linux OS stacks.
- Includes distribution blueprints (default configuration and policies) that enable quick ramp-up.
- Is supported and sustained by a growing community of contributors composed of silicon vendors, Linux OSVs, open source projects etc. providing BSPs, commercial and community support.
- Maintains stable Linux kernels with security and functionality patches.
- Provides standard format for BSPs and recipes to make them exchangeable.
- Allows you to draw from the expertise and experience of the Yocto developers while being able to easily customize, modify and extend to meet your own requirements.
- Scales from individual developer to engineering organizations.

Resources and References

Yocto Project

Website: <https://www.yoctoproject.org>

Wiki: https://wiki.yoctoproject.org/wiki/Main_Page

Downloads: <https://www.yoctoproject.org/downloads>

GIT Repository: <http://git.yoctoproject.org>

OpenEmbedded

Website/Wiki: http://www.openembedded.org/wiki/Main_Page

GIT Repository: <http://cgit.openembedded.org>

Publications

Yocto Project – Big in Embedded Linux:

<http://go.linuxfoundation.org/Yocto-Big-In-Embedded>

How Engineering Leaders Can Use The Yocto Project to Solve Common Embedded Linux Challenges:

<http://go.linuxfoundation.org/Yocto-Publication>

Collaboration is the key to success

Spend less time and resources to develop and maintain the commodity software.

Collaborate with other organizations instead and share the workload.

Be able to spend more time and use the resources you already have to create your products and value added components!



Thank you for your participation!

Please feel free to contact me with any questions:

jefro@jefro.net

Jefro on IRC (freenode.net)

@jefro_net on twitter

Jeffrey Osier-Mixon on Google+

Or through the Yocto Project website: <https://www.yoctoproject.org>



Special Topics

The Nitty Gritty in Fast Forward Mode

Troubleshooting

- **Task Run Files**

- BitBake creates a shell script for each task.
- Contains the environment variable settings and the shell and Python functions that are executed.

- **Task Log Files**

- Each task produces a log file that contains all the output from the commands run.

- **Running Specific Tasks for a Recipe**

- `bitbake <recipe> -c <task>`

- **Dependency Graphs / Dependency Explorer**

- `bitbake -g <target>`
- `bitbake -g -u depexp <target>`

- **Developer Shell**

- `bitbake <recipe> -c devshell`

Customizing Root File System Images

- **Extending a Pre-defined Image**

- Local Configuration Method
- EXTRA_IMAGE_INSTALL in conf/local.conf
- Recipe Method
- Write a recipe that includes another image recipe file

```
require recipes-core/images/core-image-base.bb  
IMAGE_INSTALL += "strace"
```

- **Inherit from Core-Image**

- Write a recipe that inherits from the core-image class

```
IMAGE_INSTALL = "packagegroup-core-boot packagegroup-base-extended"  
Inherit core-image
```

- **Package Groups**

- Write package group recipes that combine multiple packages into logical entities.
- Use the package group in IMAGE_INSTALL.

Package Groups

•Package Group Recipe

```
DESCRIPTION = "My Package Group"
LICENSE = "MIT"
LIC_FILES_CHECKSUM = "file://<licfile>;md5=<chksum>"

inherit packagegroup

PROVIDES = "${PACKAGES}"

PACKAGES = "packagegroup-mypkg-apps packagegroup-mypkg-tools"

RDEPENDS_packagegroup-mypkg-apps = "sqlite3 python-core python-sqlite3"
RDEPENDS_packagegroup-mypkg-tools = "sudo gzip tar"
```

•Image Recipe

```
IMAGE_INSTALL = "packagegroup-core-boot packagegroup-mypkg-apps"
inherit core-image
```

Layers Revisited - Conventions

- **Why layers?**

- Layers were not always supported by BitBake and OpenEmbedded Classic used a flat hierarchy for all of its meta data.
- Layers provide a mechanism to isolate meta data according to functionality, for instance BSPs, distribution configuration, etc.
- Layers allow to easily to add entire sets of meta data and/or replace sets with other sets.

- **Conventions and Best Practices for Layers**

- Use layers for your own projects
- Name your layer *meta-`<layername>`*
- Group your recipes and other meta data
- Append don't overlay
- Include don't duplicate

Layers Revisited – Creating a Layer

- Layers as easy as 1-2-3
- Create layer directory layout
- Add the layer configuration file
- Add the layer to your build environment
- **Template for layer.conf**

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

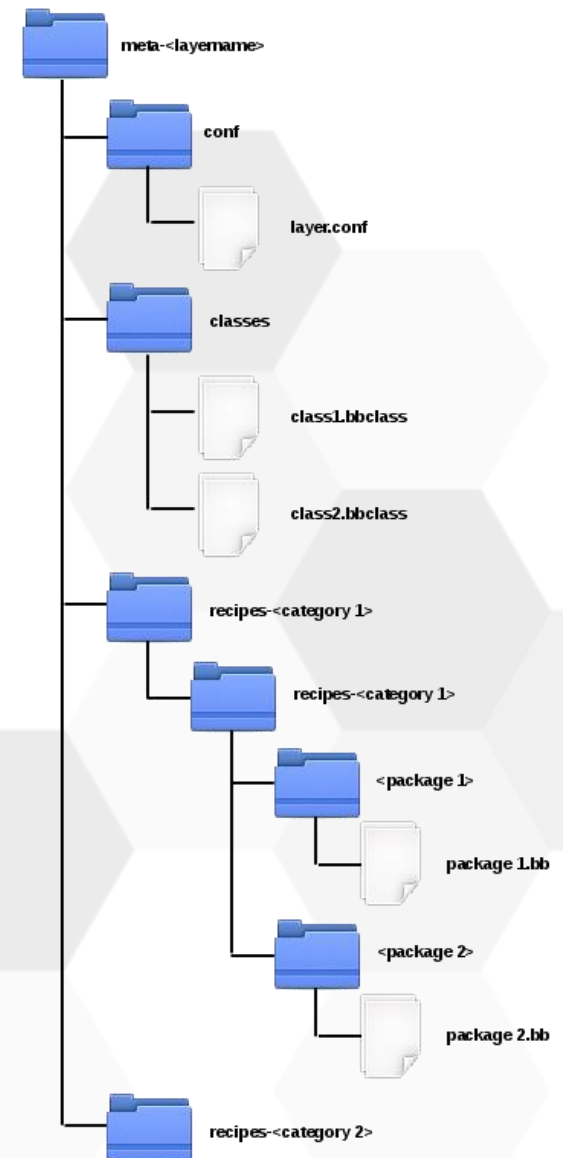
```
# We have recipes-* directories, add to BBFILES
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb
           ${LAYERDIR}/recipes-*/*/*.bbappend"
```

```
BBFILE_COLLECTIONS += "layername"
BBFILE_PATTERN_layername = "^${LAYERDIR}/"
BBFILE_PRIORITY_layername = "1"
```

```
# This should only be incremented on significant
# changes that will
# cause compatibility issues with other layers
LAYERVERSION_layername = "1"
```

```
LAYERDEPENDS_layername = "core"
```

- **Correct ordering of layers in BBLAYERS is important**



Yocto Project BSP - Architecture

Yocto Project BSP Anatomy

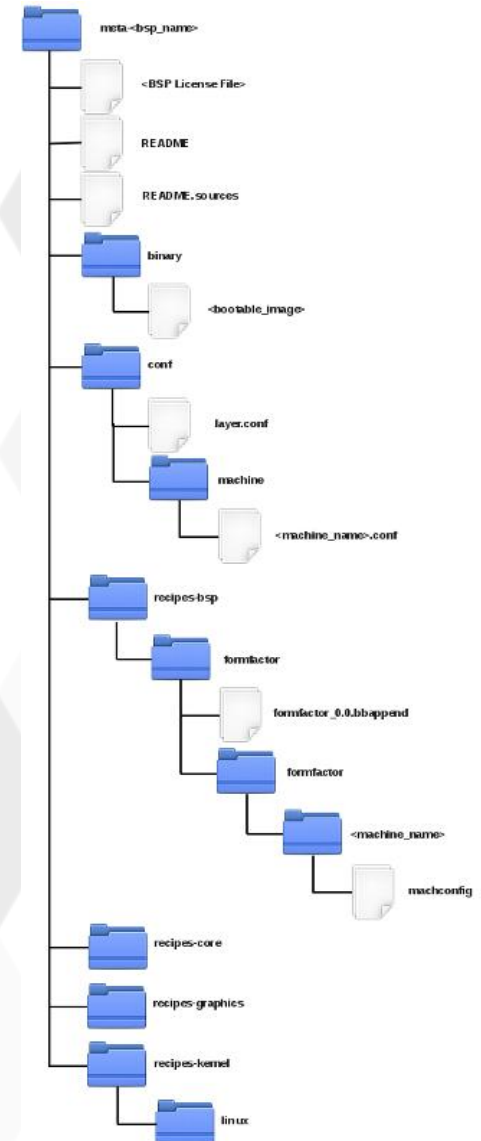
- Configuration and recipes for hardware platforms
- Dependent on core layers
- Extend core layer recipes and configuration
- Do not contain build system and/or tools

Standardized Layout

- Binary images
- Machine configuration
- Documentation
- Bootloader, kernel, graphics subsystem recipes
- Source patches
- License

BSP Tools

- Create BSP layers for various architectures and kernel configurations
- Kernel configuration and patch management



Consuming a Yocto Project BSP

- **Read the README – no kidding**
- BSP dependencies
- Build instructions
- **Create and Configure Build Environment**
- `oe-init-build-env mybuild`
- Add BSP layer to BBLAYERS variable in `mybuild/conf/bblayers.conf`
- Correct order of layers in BBLAYERS is of significance: applications, distribution, BSP, core
- Configure MACHINE in `mybuild/conf/local.conf`
- **Launch Build**
- `Bitbake -k <image-target>`

Building a Yocto Project BSP

- **Three Approaches**

- Manually from Scratch
- Most challenging
- Could make sense if no BSPs for similar hardware exist
- Copying and Modifying an Existing BSP Layer
- For similar hardware but it could make more sense to just extend the existing BSP
- Using the Yocto Project BSP Scripts
- Interactive scripts to build a BSP using the Yocto Project kernel infrastructure
- **A BSP is not required to use the Yocto Project kernel infrastructure and tooling**
- However, using it provides benefits such as maintenance.

Yocto Project Kernel Development

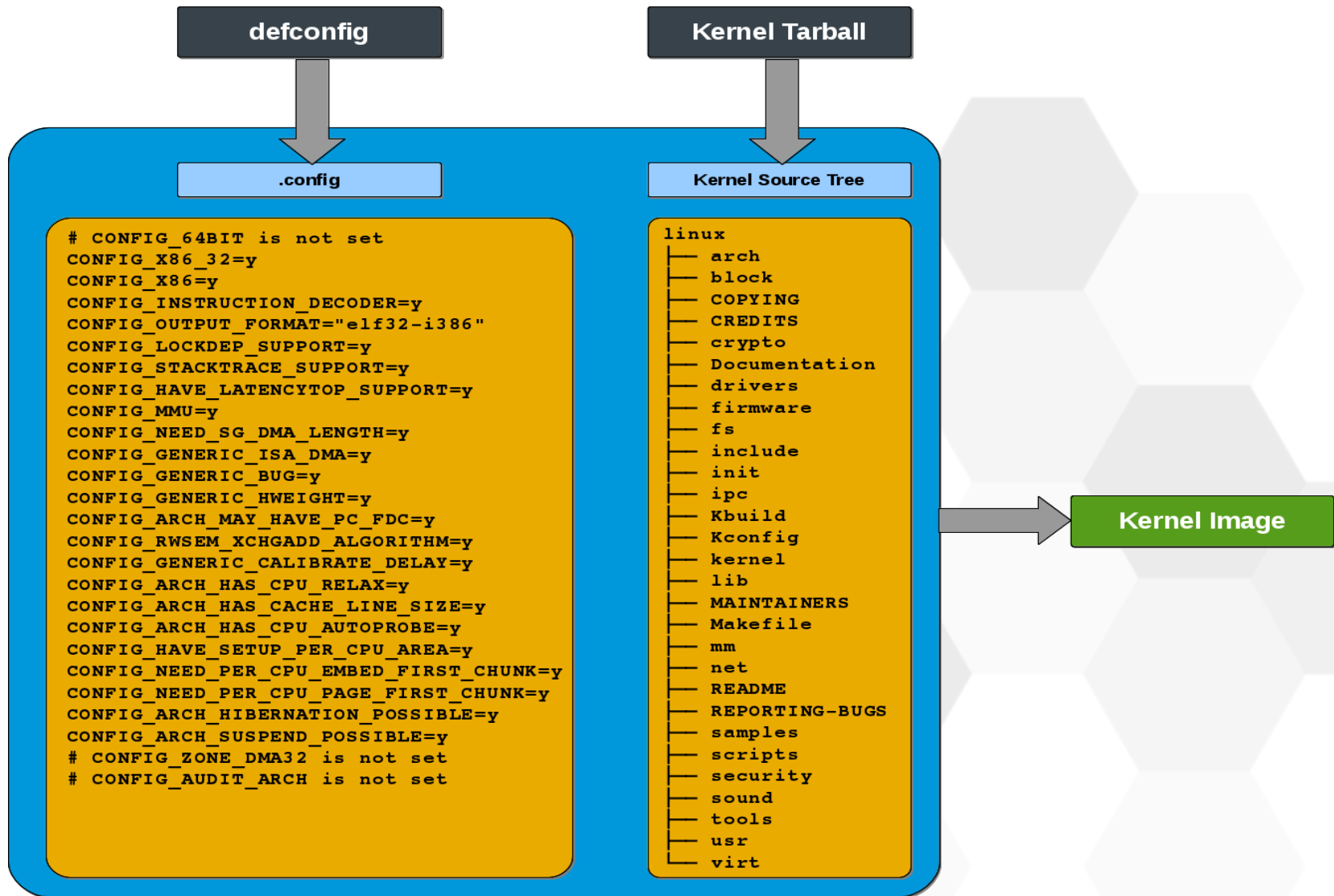
- **There is no Yocto Project Kernel**

- Uses upstream Linux kernels from kernel.org and clone them into Yocto Project kernel repositories
- Recipes and tooling point to the Yocto Project kernel repositories.
- Yocto Project adds machine meta data, configuration, patches on top.

- **Multiple ways of building the kernel**

- Traditional OpenEmbedded Kernel Recipes building from kernel tarball
- Custom Linux Yocto Kernel Recipes building from any kernel GIT repository
- Linux Yocto Kernel Infrastructure Recipes building from Yocto Project GIT kernel repository

Traditional OE Kernel Method - Overview



Traditional OE Kernel Method - Recipe

```
DESCRIPTION = "Bleeding Edge Linux Kernel"  
SECTION = "kernel"  
LICENSE = "GPLv2"
```

```
LIC_FILES_CHKSUM = "file://COPYING;md5=<chksum>"
```

```
inherit kernel
```

```
KVER = "${PV}-rc5"
```

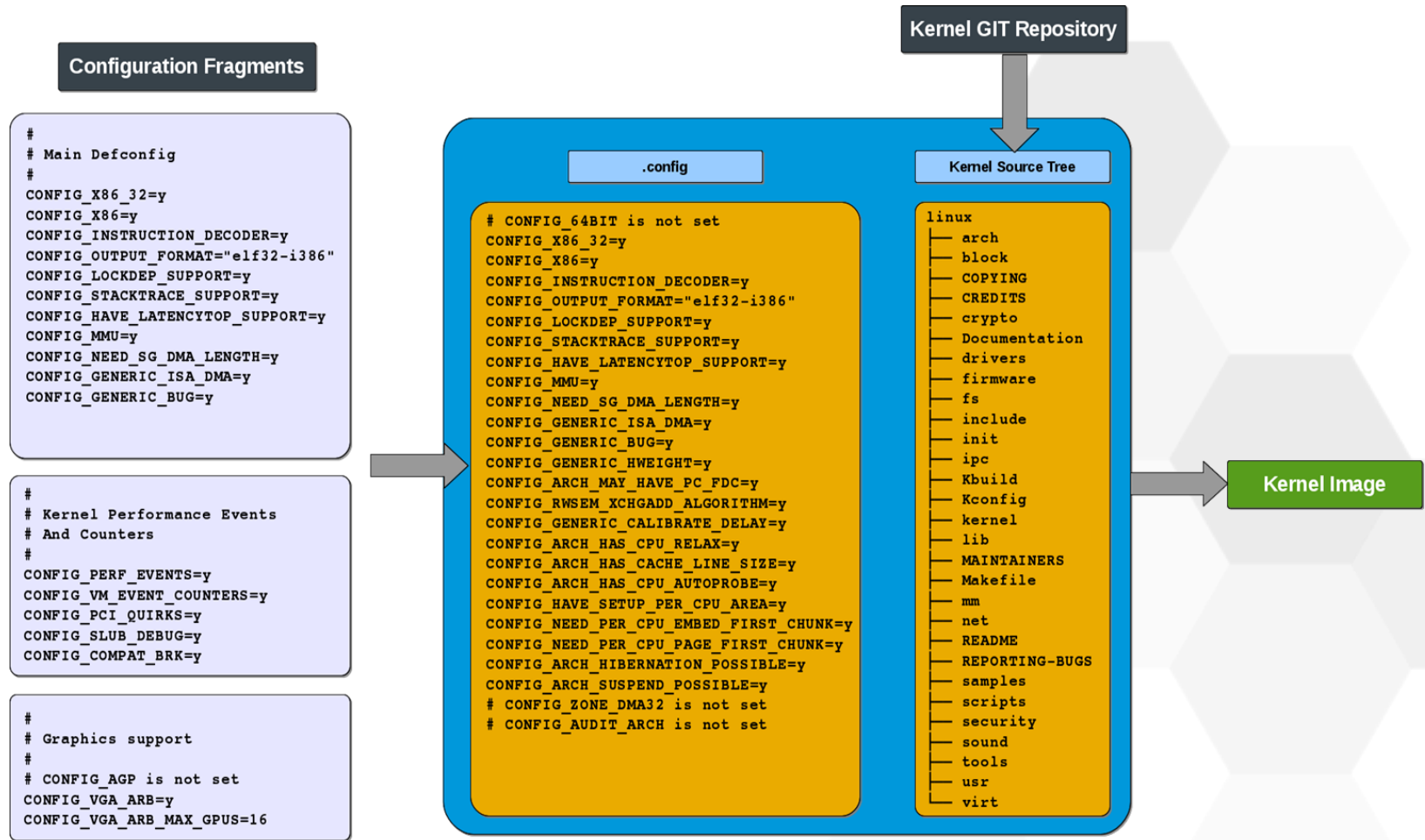
```
LINUX_VERSION ?= "3.11.0"  
LINUX_VERSION_EXTENSION ?= "-custom"
```

```
SRC_FILE = "${KERNELORG_MIRROR}/linux/kernel/v3.x/testing/linux-${KVER}.tar.xz"  
SRC_URI = "${SRC_FILE};name=kernel \  
    file://defconfig"
```

```
S = "${WORKDIR}/linux-${KVER}"
```

```
SRC_URI[kernel.md5sum] = "<chksum>"  
SRC_URI[kernel.sha256sum] = "<chksum>"
```

Linux Yocto Custom Method - Overview



Linux Yocto Custom Method - Recipe

inherit kernel

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

SRC_URI = "git://arago-project.org/git/projects/linux-am33x.git;protocol=git;bareclone=1"

SRC_URI += "file://defconfig"

SRC_URI += "file://am335x-pm-firmware.bin"

**SRC_URI += "file://beaglebone.scc \
file://beaglebone.cfg \
file://beaglebone-user-config.cfg \
file://beaglebone-user-patches.scc \
"**

KBRANCH = "v3.2-staging"

LINUX_VERSION ?= "3.2.31"

LINUX_VERSION_EXTENSION ?= "-bbone"

SRCREV = "720e07b4c1f687b61b147b31c698cb6816d72f01"

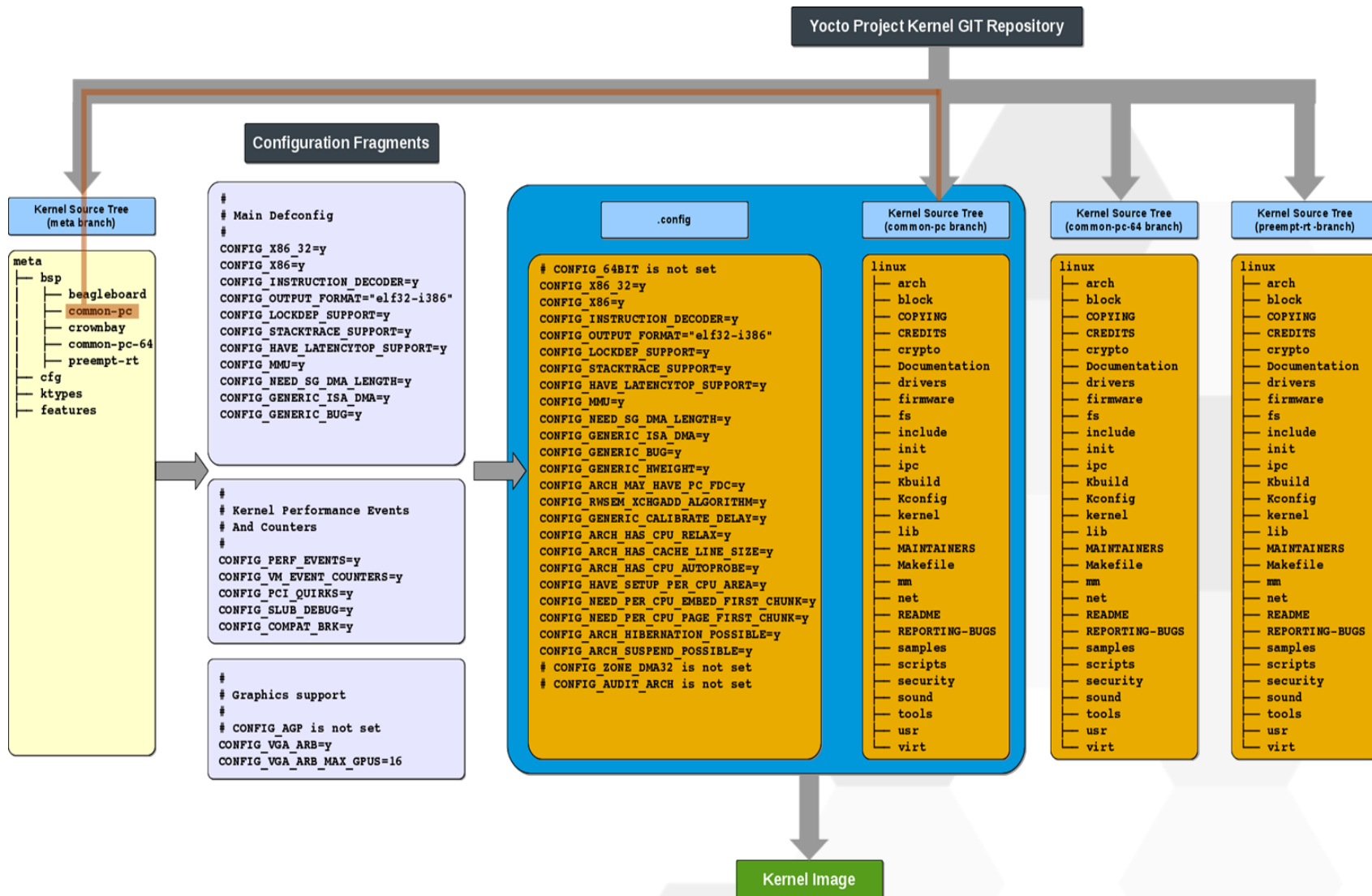
PR = "r1"

PV = "\${LINUX_VERSION}+git\${SRCPV}"

COMPATIBLE_MACHINE_beaglebone = "beaglebone"

**do_compile_prepend() {
cp \${WORKDIR}/am335x-pm-firmware.bin \${S}/firmware/
}**

Linux Yocto Kernel Method - Overview



Linux Yocto Kernel Method - Recipe

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

KBRANCH_DEFAULT = "standard/base"

KBRANCH = "\${KBRANCH_DEFAULT}"

SRCREV_machine_qemuarm ?= "8fb1a478c9a05362e2e4e62fc30f5ef5d6c21f49"

SRCREV_machine_qemumips ?= "b8870f2b11f4c948ae90a19886335fa8b7fca487"

SRCREV_machine_qemuppc ?= "e4c12f12e61a29b6605c4fcbcfd6dbe18bd7b4e4"

SRCREV_machine_qemux86 ?= "dd089cb5ba37ea14e8f90a884bf2a5be64ed817d"

SRCREV_machine_qemux86-64 ?= "dd089cb5ba37ea14e8f90a884bf2a5be64ed817d"

SRCREV_machine ?= "dd089cb5ba37ea14e8f90a884bf2a5be64ed817d"

SRCREV_meta ?= "8482dcdf68f9f7501118f4c01fdc8f851882997"

SRC_URI = "git://git.yoctoproject.org/linux-yocto-3.8.git;protocol=git;bareclone=1;\nbranch=\${KBRANCH},\${KMETA};name=machine,meta"

LINUX_VERSION ?= "3.8.11"

PR = "\${INC_PR}.1"

PV = "\${LINUX_VERSION}+git\${SRCPV}"

KMETA = "meta"

COMPATIBLE_MACHINE = "qemuarm|qemux86|qemuppc|qemumips|qemux86-64"

When to Use What Kernel Building Method

- **You have a kernel tarball and a defconfig**
 - Use *linux-yocto-custom* recipe template
 - Straightforward and easy to use
- **You have a GIT kernel repository and a defconfig**
 - Use *linux-yocto-custom* recipe template with GIT
 - Gives you the ability to add patches and configuration fragments using the Yocto Project kernel tooling
- **You are starting a new BSP project**
 - Consider using the Yocto Project kernel infrastructure, repositories and tooling
 - Get the advantage of an continuously updated and maintained kernel
 - Leverage the kernel types, feature and configuration pool of the *meta* kernel branch

Questions from webinar

What hardware is supported?

- A great deal of hardware is supported. Architectures include ARM, ARM64, PPC, PPC64, x86, x86-64, MIPS, and even some others.

Will my resulting OS have access to an app store?

- At the moment, YP does not provide binary package feeds of any kind, including access to app stores. It is very difficult to offer an app store for a build system that can build software on so many different architectures, and it is not often requested for deeply embedded systems that only do one job. That being said, this functionality is on our roadmap, though it is not currently scheduled for a particular release.

Where can I find other software layers & BSPs?

- You can find layers and BSPs on yoctoproject.org, the OE layer index at layers.openembedded.org, on silicon manufacturers' sites, on github and gerrithub, and in places even we don't know about!

Questions from webinar

What is the difference between the Yocto Project and OpenEmbedded? How about Buildroot?

- The Yocto Project is an open source project – an umbrella organization that provides support and business services to its component projects, which all have the common goal of enhancing the process for building embedded operating systems based on Linux. Some of the project managed within the Yocto Project are the OpenEmbedded build system, the opkg package manager, EGLIBC, and several other tools that help people build embedded Linux distributions. The Yocto Project is managed by 17 member organizations as a collaborative project under the Linux Foundation.
- OpenEmbedded is a build system comprised of the BitBake build tool and the OE Core metadata set. It is managed by the OpenEmbedded Project, which also manages a great number of metadata packages that are technically compatible with the OE build system (and thus with the Yocto Project).
- Poky is a reference system – a single instantiation of the Yocto Project build tools. In essence, when you “download the Yocto Project”, you download Poky, which contains the OE build system, the meta-yocto metadata layer that includes canonical BSPs, and a few other scripts that make it easy to build embedded Linux. You then customize this reference system to meet your own needs. Why not just download BitBake and OE separately? Because Poky is heavily tested as a system.
- The Buildroot build system is a different way of building embedded Linux that is based on build scripts and kconfig. Buildroot is unrelated to the Yocto Project, but the systems are not competitors – instead, we refer users to each other when the other system is more appropriate, in the true spirit of open source.

Questions from webinar

Is there a GUI or is it all command line?

- There is an older GUI called Hob that is now deprecated. There is a new HTML5-based GUI called Toaster that is still being designed and created. It has a manual as well as several videos on youtube to show how it works, and you can download it today. We welcome feedback on it.

Does it run on Windows/Mac or only Linux?

- The Yocto Project tools currently run only on Linux systems with the appropriate libraries and build tools available (see the Quick Start Guide). However, you can run the YP tools in a virtual machine on any system. Native Windows and Mac support is on our roadmap for a future release, but full support has not yet been scheduled or announced.

Is synopsys ARC HS38 supported?

- I don't immediately find an ARC HS38 BSP for YP, but I do note that the processor is supported in the Linux kernel. That means porting is just a few recipes away. You can see what is required to create your own BSP in the BSP Porting Guide - <https://www.yoctoproject.org/documentation>

What happens if you don't have a 32gb build machine?

- I frequently run YP builds on a core i7 laptop with 12gb RAM. I can build core-image-minimal in about 45 minutes, after downloading sources. YP is very good at capitalizing on available resources.

Questions from webinar

Does it work on Backtrack?

- I have never used Backtrack, but I don't see anything in Backtrack that would be incompatible. Remember, YP is a set of tools – it is not a platform, and it is not a distribution.

How hard is it to migrate platforms between different versions of the Yocto Project?

- You can lock down any part of your build, including the kernel or any package, so that it continues to build the same version despite updates to the build tools. That being said, YP (or rather, Poky) is tested with certain kernels, so some users find it easier to just stick with one version of the YP tools throughout a given project.

Do I need a specific bitbake to generate each version of the os (eg one for poky-dizzy and other distro) also, What about when new versions of YP are released?

- I think the above question should answer this, but if not, or with any other questions, please feel free to send me a note at jeffrey.osier-mixon@intel.com



Operating Systems

Using qemu and yocto in a classroom environment

D. Kevin McGrath – Oregon State University School of Electrical Engineering and Computer Science

Target audience and structure

Advanced undergraduate or entry level graduate course in operating systems concepts and implementation

- Use of Linux kernel as the case study
- Implementation projects
 - Scheduling
 - I/O
 - Kernel crypto API + block device
 - Userspace driver – think libusb
 - Memory management/slab
- Concurrency projects

Choice of target platform

There are several questions one must answer before picking a teaching platform:

- Which architecture to target?
- Physical or virtual hardware?
 - Physical options: Galileo, Edison, BeagleBone Black, Raspberry Pi 2, Terasic DE2i-150, MinnowBoard Max
 - Virtual options: qemu, VirtualBox, VMware
- What are the learning outcomes to target?

Physical targets

Physical targets work well in smaller classes

- Benefits
 - Real platforms, real issues
 - Can do real driver development with actual peripheral
 - Simplicity of on-host development
- Downsides
 - Debugging difficulty
 - Cost/attrition of hardware
 - Students don't like to carry things

Virtual targets

Virtual targets are ideal for larger class sizes, provided hardware is available

- Benefits
 - Easier choice of different platforms
 - Myriad debugging options
 - Flexible, students can't destroy it
 - OS development often done on VMs
- Downsides
 - Driver projects require more OOB thinking
 - Scaling to larger class sizes actually can be more expensive than physical hardware

Using yocto build system + qemu

Ability to target lots of platforms

- Yocto build system offers great flexibility
 - Cross compilers for all qemu virtualized targets
 - Simplicity of centrally managed build system
 - Same tools, same host for all students
- With proper hardware support, can scale to very large classes
 - Qemu+kvm offers significant performance advantages
- Cross compilation experience within a single host
- Can alternatively target SBCs like Galileo with only minor mods

Demonstration

- Toolchain installation
- Demo image and kernel
- Using qemu, command line options
- Cross compiling the kernel
- Running new kernel
- Adding a module

Steps:

- Download and install the toolchain:
 - `curl -O http://downloads.yoctoproject.org/releases/yocto/yocto-1.6.1/toolchain/x86_64/poky-eglibc-x86_64-core-image-sato-core2-64-toolchain-1.6.1.sh`
 - `sh poky-eglibc-x86_64-core-image-sato-core2-64-toolchain-1.6.1.sh -d ~/yocto -y`
- Download a kernel – match to toolchain:
 - `curl -O http://downloads.yoctoproject.org/releases/yocto/yocto-1.6.1/machines/qemu/qemux86-64/bzImage-qemux86-64.bin`
- Download a disk image (takes some time) – there are lots of options, feel free to switch
 - `curl -O http://downloads.yoctoproject.org/releases/yocto/yocto-1.6.1/machines/qemu/qemux86-64-lsb/core-image-lsb-sdk-qemux86-64.ext3`
- Download kernel source (and immediately expand) – match to that which image is running
 - `curl https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.14.tar.gz | tar xzvf -`
- Pull config from VM
 - Launch VM:
 - `source ~/yocto/environment-setup-core2-64-poky-linux`
 - `qemu-system-x86_64 -gdb tcp::5500 -S -smp 4 -m 2048 -nographic -kernel linux-3.14/arch/x86/boot/bzImage -drive file=core-image-lsb-sdk-qemux86-64.ext3,if=virtio -enable-kvm -net none -usb -localtime --no-reboot --append "root=/dev/vda rw console=ttyS0 debug"`
 - Log in to VM as root (empty password)
 - `zcat /proc/config.gz > ~/config.3.14.txt`
 - Pull config.3.14.txt out to host, copy to linux source tree as .config

Steps continued:

- Build the kernel with the standard make `-jN` (replace N with number of build threads)
- Replace kernel on qemu command line with the kernel you just built
- Use scp to load any modules as necessary
- Add/remove `-S` flag to qemu to require debugger or not
- All links from [Yocto Quick Start Guide](#), section 'Using Pre-Built Binaries and QEMU'

Lessons learned

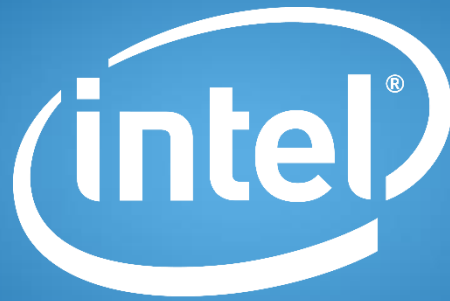
- Using virtio is great for performance – unless you want to modify the I/O schedulers
- Writing and using modules is important, so how to get the module to the guest?
- Remote debugging is awesome. Mostly.
- KVM support is really necessary. Only root can use it by default.
- Lots of root-only tools. Networking, for instance.
- Guest/host split does cause confusion.
- Whiners gonna whine. Haters gonna hate.
- Detailed setup instructions are a must. Thinking is hard.

Questions/answers

- Why use a cross compiler to build the kernel?
 - Not strictly necessary, but allows (for instance) to use a 32bit guest on a 64 bit host, or an ARM guest on an IA host. Further, allows for mismatched libraries and versions to be used on host and guest (eglibc vs. uclibc, for instance).
- Why use the yocto build system?
 - Vastly simplifies toolchain installation for cross-compilation.
- Are there benefits to using yocto+qemu over real hardware?
 - Yes. Simplifies course management, no broken hardware, etc. Please see slides 4 and 5.
- Can you adapt this system to using real hardware, such as the Intel Edison?
 - Quite easily, in fact. Instead of launching the new kernel with qemu, simply copy to the FAT partition on Edison/galileo. Just make sure you get the names right, and it should all just work.
- Can you add modules to the used file system?
 - Yes, you can either scp them onto the running VM, or use root-only tools to mount the rootfs image and do a direct make modules_install to that directory. scp is the easier option, in a classroom environment.
- How can networking be enabled?
 - Just remove the ' -net none' from the qemu command line

Questions/answers

- How can you load symbols for the kernel to the debugger?
 - Launch gdb with the vmlinuz file in the root of the kernel build tree.
- Can KVM/Qemu be useful if I want to pass-through a PCI Network card directly to the VM?
 - Qemu can use physical hardware, but I've never used it that way. I'd recommend consulting the qemu manual for details. That said, in a classroom environment, unless you had one NIC per student, I'm unsure how you'd do this in any meaningful way.
- Using a vm as the target system, how does one account for peripherals? For example on a Pi, how would one emulate the SD card, the camera, or the GPU?
 - This is very much a question for a qemu expert. I'm really unsure why one would want to in a classroom setting for an OS class, but each use case is different, after all. If part of your class involves programming those specific peripherals, this might not be the best approach for your particular class.



experience
what's inside™