




yocto ·
PROJECT

THE
LINUX
FOUNDATION

Introduction to the Yocto Project

Accelerating Embedded Product Development

Rudolf J Streif



[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.**

Embedded Linux – Why is it Challenging?

- **DIY/Roll-Your-Own or Modify Mainstream Distro**
 - Long Term Maintenance is difficult
 - Upstream Changes are difficult to track
 - Not embedded friendly
 - Licensing issues
 - No commercial embedded support
 - Build System and Cross Toolchain Challenges
- **Commercial/Community Embedded Linux**
 - Too many competing systems
 - Incompatible distributions/build systems

Developers spend lots of time porting or making build systems

Leaves less time and resources to develop value-adding software features

Embedded Linux Landscape (1)

Android – <http://source.android.com>

- Great for systems with ARM-based SoCs and touch screens
- Build system and development tools



Ångström Distribution – <http://www.angstrom-distribution.org>

- Community distribution with a growing list of supported development boards
- Yocto Project build environment
- Online builder Narcissus



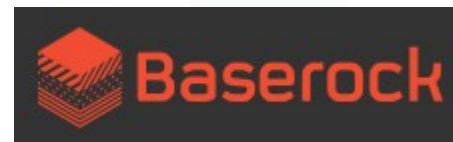
Buildroot – <http://buildroot.uclibc.org>

- GNU make-based build system
- uClibc target library
- BusyBox command line utility applications



Baserock – <http://wiki.baserock.org>

- Targeted for embedded appliances
- Native Builds for x86, x86_64, ARMv7



Embedded Linux Landscape (2)

OpenEmbedded – <http://www.openembedded.org>

- Created by merging OpenZaurus with contributions from Familiar Linux, OpenSIMPAd
- Based on BitBake build engine



OpenWrt – <http://www.openwrt.org>

- Debuted as open source OS for embedded devices routing network traffic
- Originally created from Linksys GPL sources for their WRT54G residential gateway
- Buildroot-based build environment
- Headless operation with web UI



The Yocto Project – <http://www.yoctoproject.org>

- Details to follow



Commercial Distributions

- Various solutions from different vendors for different applications
- Commercial support for toolchain and 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 build system
- **The benefit of doing so is:**
 - Designed for the long term
 - Designed for embedded
 - Transparent Upstream changes
 - Vibrant Developer Community
- ***Less time spent on things which don't make money (build system, core Linux components)***
- ***More time spent on things which do make money (app development, product development, ...)***

Who is 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)**

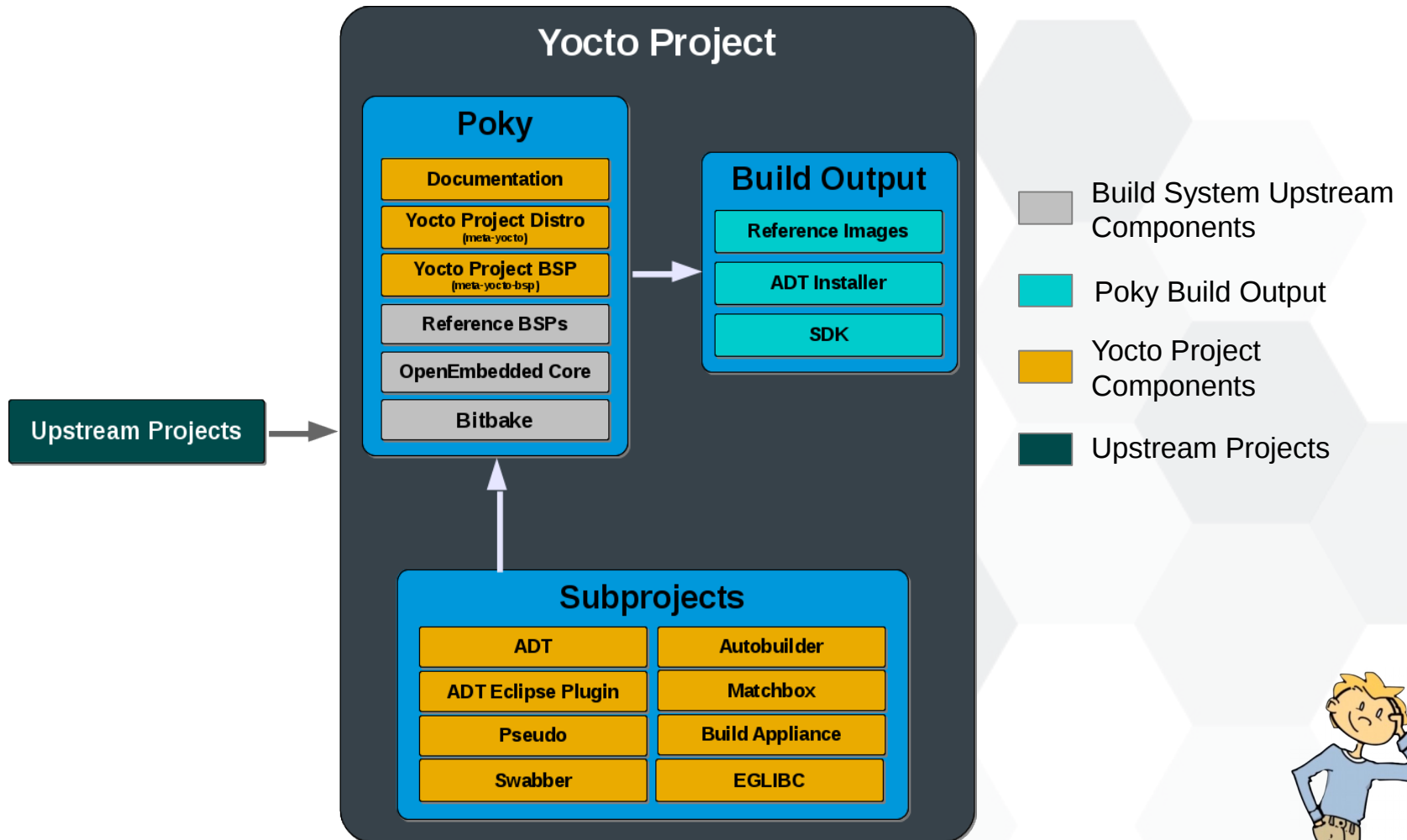


Why Should a Developer Care? (2)

- **Supports all major embedded architectures**
 - x86, x86-64, ARM, PPC, MIPS
 - Coming soon, MIPS64 and ARM Arch 64
- **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



The Yocto Project Family

- **Poky** – Yocto Project Reference Build System
- **BitBake** – Build Engine
- **Hob** – Graphical User Interface for BitBake
- **OpenEmbedded Core** – Shared Base Layer of Recipes and Classes
- **Application Development Toolkit (ADT)** – Development environment for user-space applications to run on OS stacks built by Poky
- **Eclipse IDE Plugin** – Integration of ADT into the Eclipse IDE
- **EGLIBC** – Embedded variant of the GNU C Library
- **Matchbox** – X Windows-based open source graphical UI for embedded devices
- **Autobuilder** – Automation for Yocto Project build tests and QA
- **Build Appliance** – Virtual machine image to try out the Yocto Project and Poky
- **Pseudo** – System administrator simulation environment
- **Swabber** – Host leakage detection tool

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 with two major releases 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 and QA Tests
 - Provides tools for application development such as ADT and Eclipse Plugin

The Yocto Project Ecosystem

Product Showcase

- Hardware Platforms
- Distributions – Open Source and Commercial
- Projects – Open Source Project using the Yocto Project

Participants

- Organizations who participate in the Yocto Project Compliance Program
- They also support the project through contributions and engineering resources

Member Organizations

- Organizations who provide the administrative leadership of the Yocto Project
- Their support includes membership dues for infrastructure etc. and engineering resources
- Members of the Yocto Project Advisory Board

Supporting Organizations

- Organizations who support the Yocto Project through contributions, product development, etc.

Yocto Project Branding and Compliance Program

Goals

- Strengthen the Yocto Project through a consistent branding.
- Provide recognition to participating organizations.
- Reduce fragmentation in the embedded Linux market by encouraging collaborative development of a common set of tools, standards, and practices and ensure that these tools, standards, and practices are architecturally independent as much as possible.

Yocto Project Participant

- Organizations and entities who use and support the Yocto Project publicly.
- Open to open source projects, non-profit organizations, small companies, and Yocto Project member organizations.

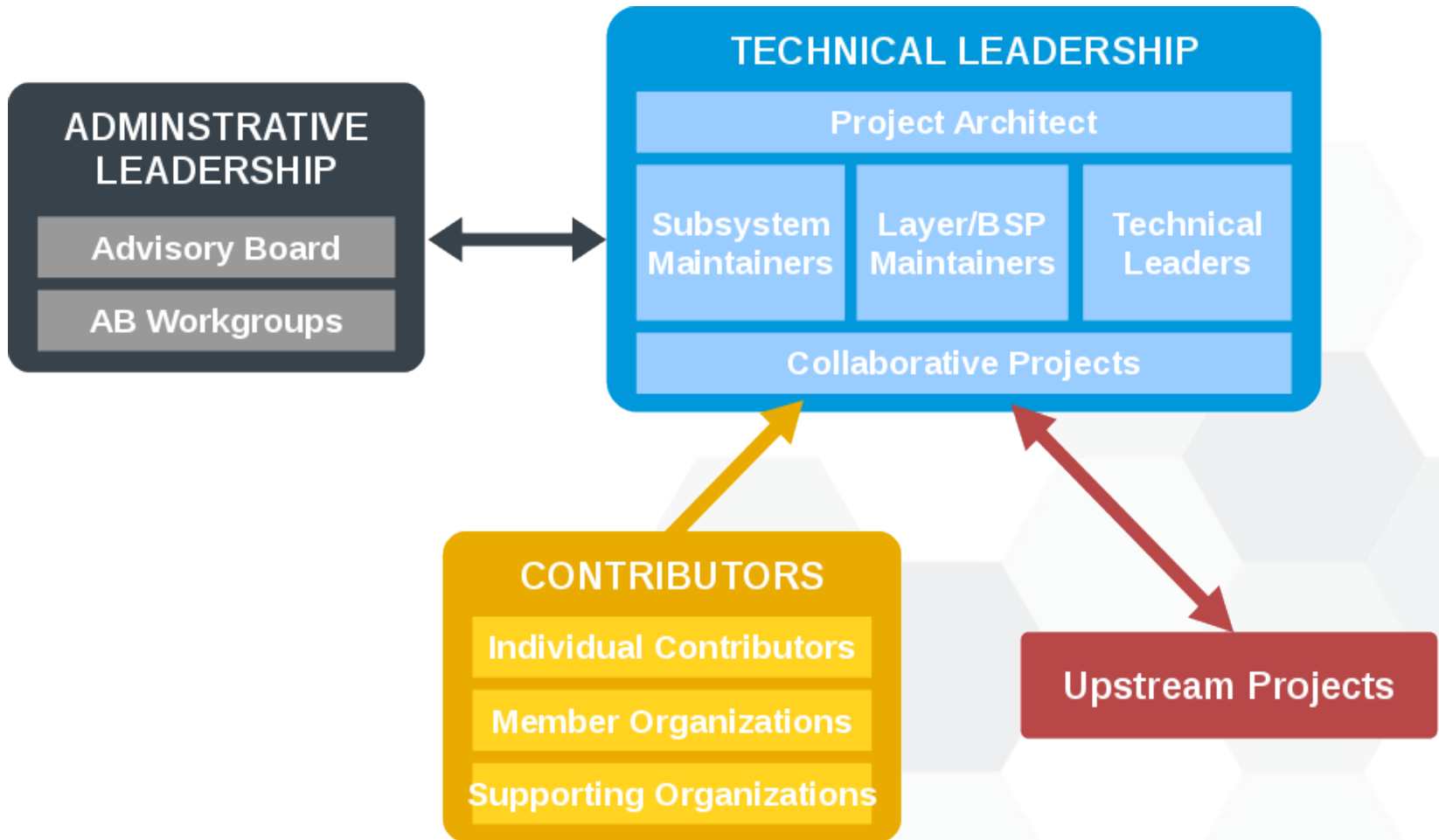


Yocto Project Compatible

- Products, BSPs, OpenEmbedded-compatible layers and other open source software projects that are built and work with the Yocto Project.
- These products and components must be submitted by open source projects, non-profit entities, or Yocto Project member organizations.



The Yocto Project Community



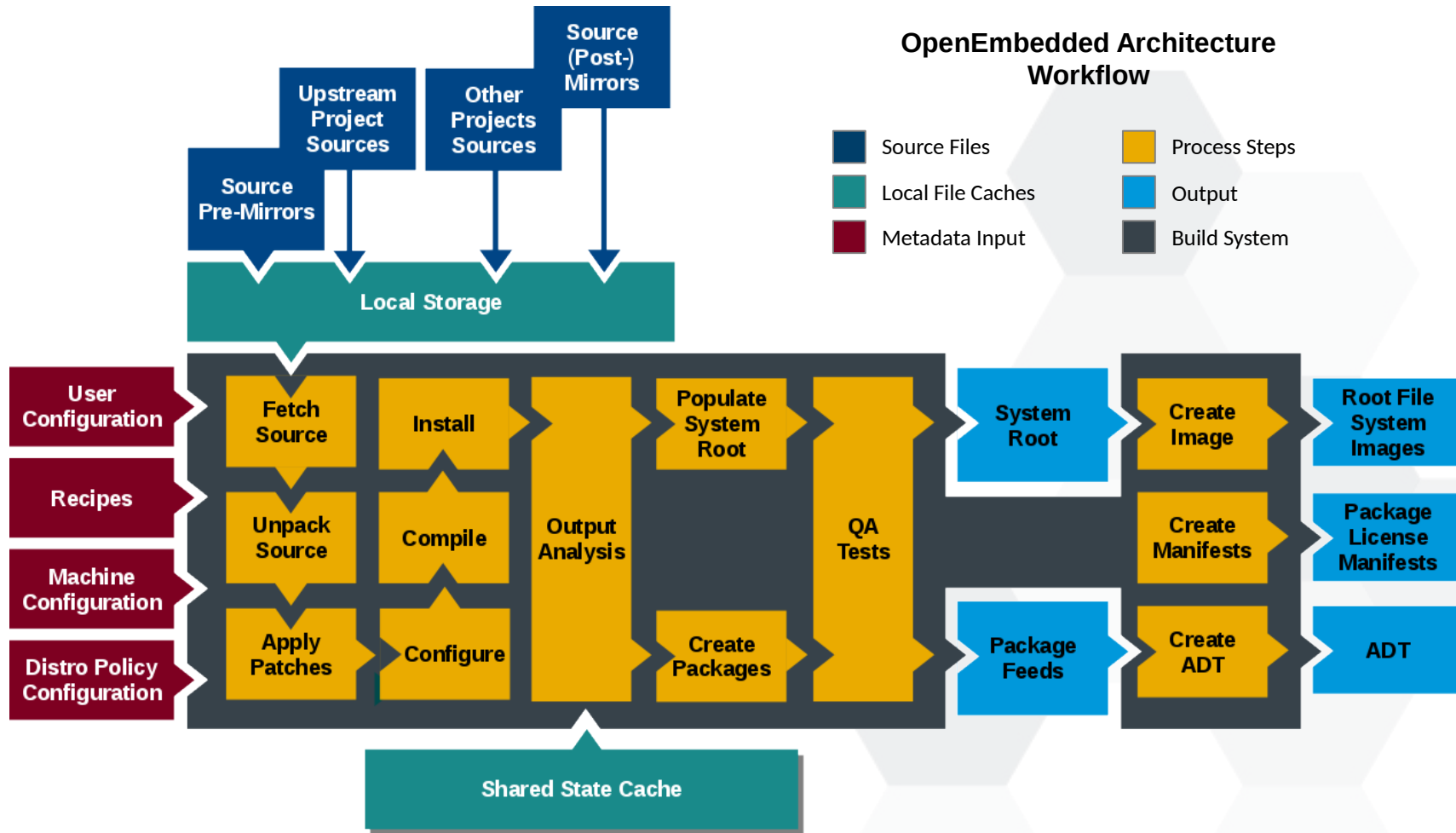
Getting Started

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

Quick Start

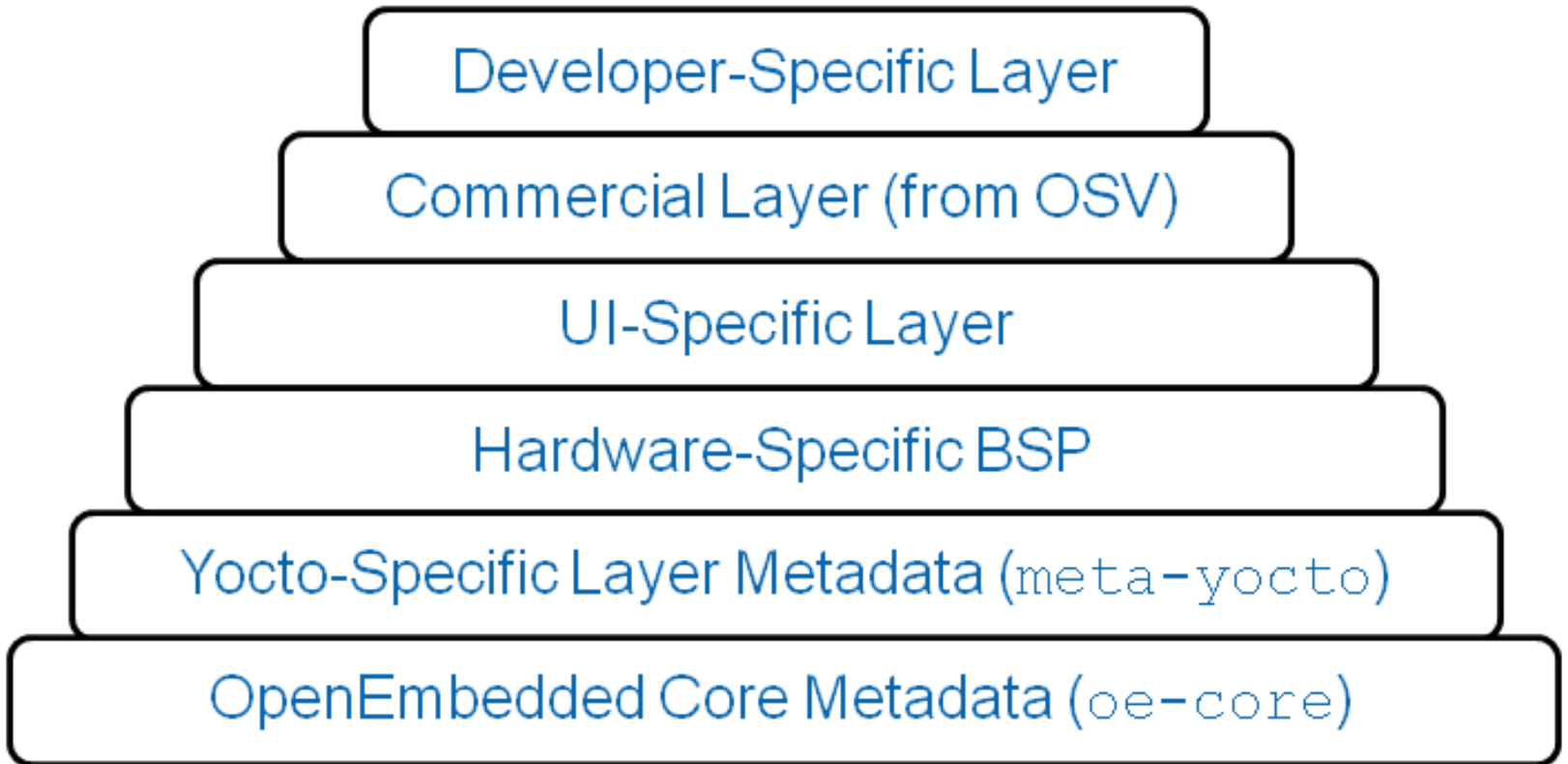
1. Go to <http://yoctoproject.org> click “Documentation” and read the Quick Start guide
2. Set up your Linux build system with the necessary packages (and firewall as needed)
3. Go to <http://www.yoctoproject.org> click “downloads” and download the latest stable release (Yocto Project 1.6.1 “Daisy” 11.0.1) – 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 qemux86` (if `MACHINE=qemux86`)

Build System Workflow



Layers (1)

- The build system is composed of layers



Layers (2)

- Layers are containers for the building blocks used to construct the system



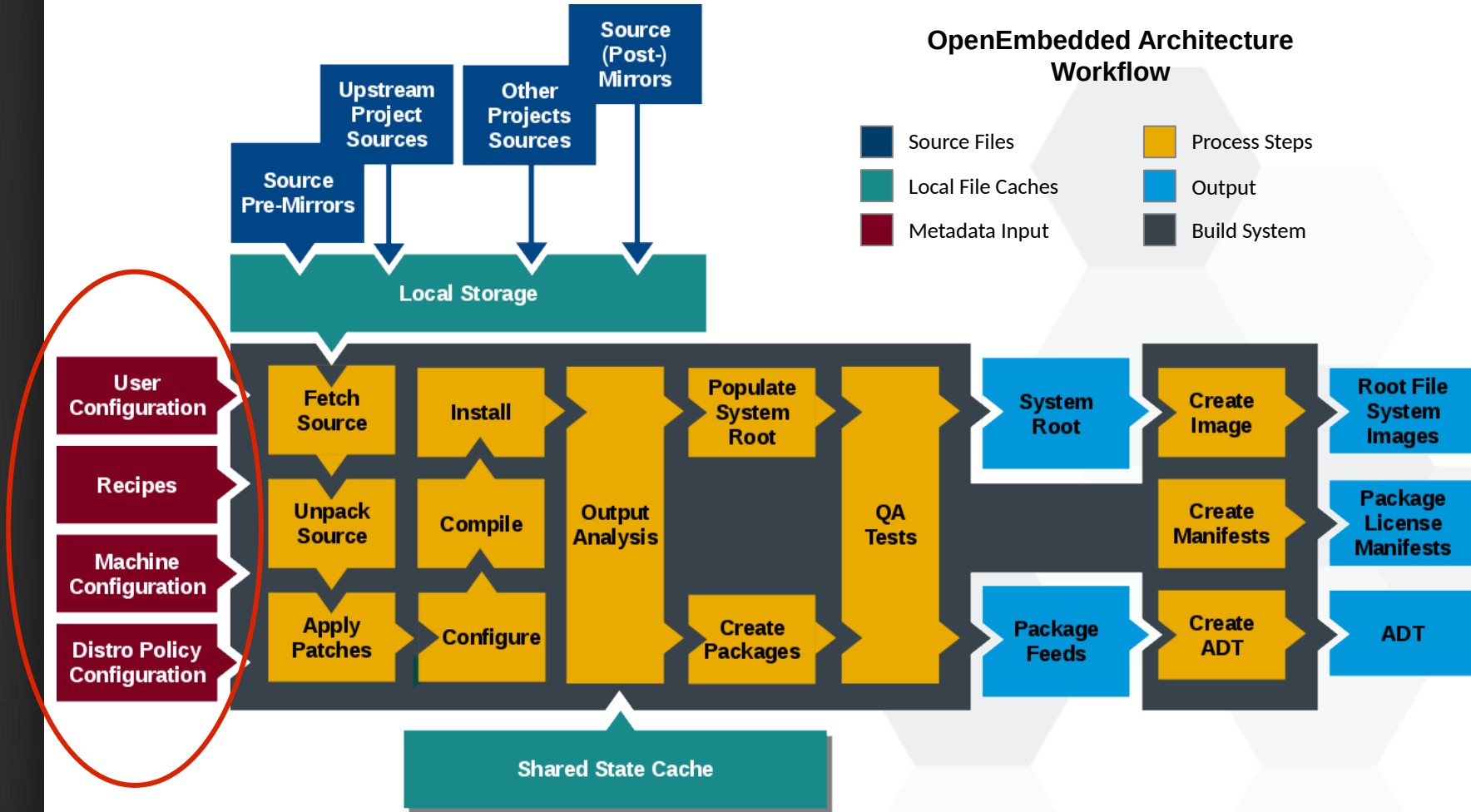
LEGO is a trademark of the LEGO Group

Layers (3)

- **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

User
Configuration

Recipes

Machine
Configuration

Distro Policy
Configuration

- **Configuration files (*.conf) – global build settings**
 - 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 Yocto Project Compatible layers to the 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)**
- **Eg, meta-yocto-bsp/conf/machine/beagleboard.conf**
- **Machine configuration refers to kernel sources and may influence some userspace software**
- **Compatible with OpenEmbedded**

Distribution Policy

User
Configuration

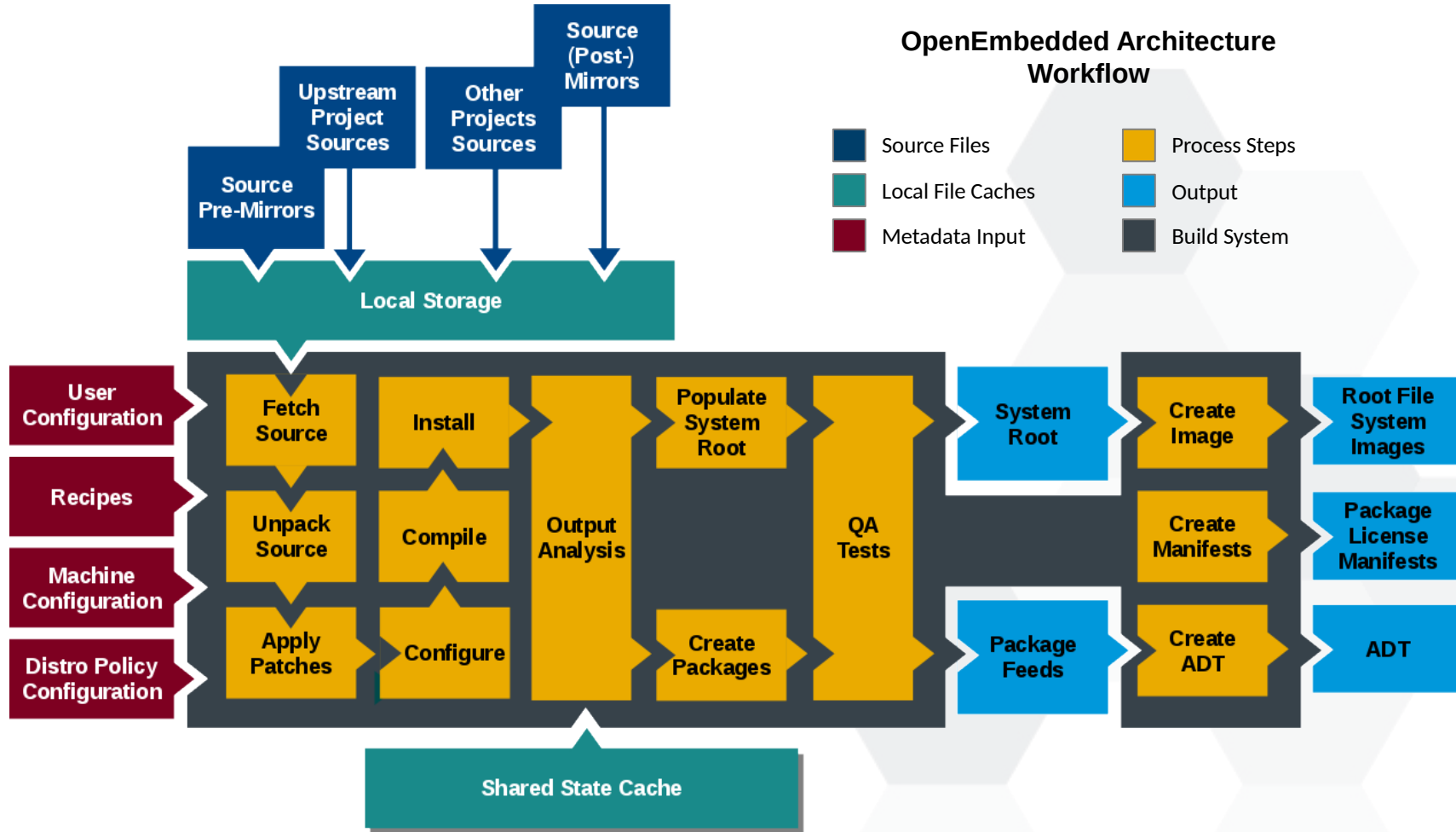
Recipes

Machine
Configuration

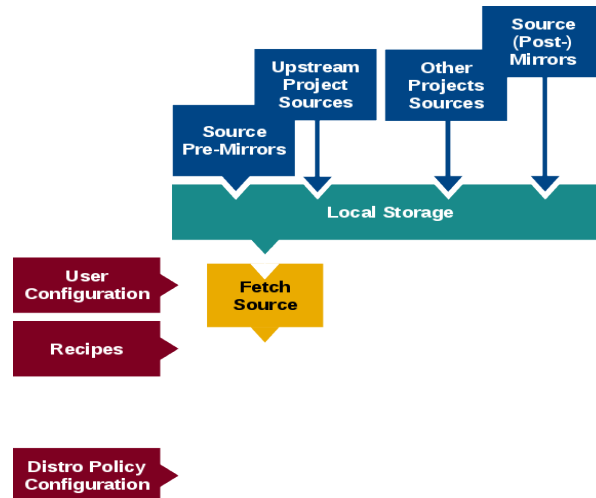
Distro Policy
Configuration

- **Defines distribution/system wide policies that affect the way individual recipes are built**
 - May set alternative preferred versions of recipes
 - May enable/disable LIBC functionality (i.e. i18n)
 - May enable/disable features (i.e. pam, selinux)
 - May configure specific package rules
 - May adjust image deployment settings
- **Enabled via the DISTRO setting**
- **Four predefined settings**
 - poky-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



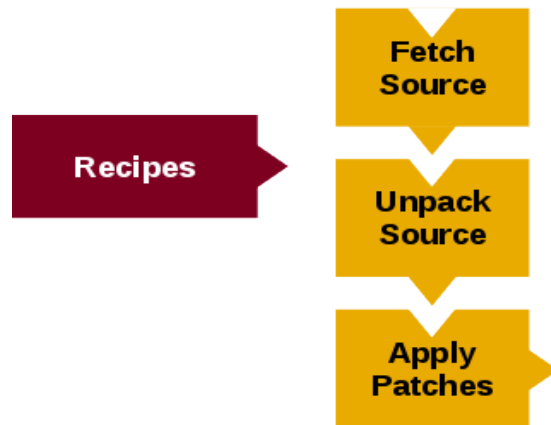
Source Fetching



- Recipes call out the location of all sources, patches and files. These may exist on the internet or be local. (See SRC_URI in the *.bb files)
- Bitbake can get the sources from git, svn, bazaar, 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

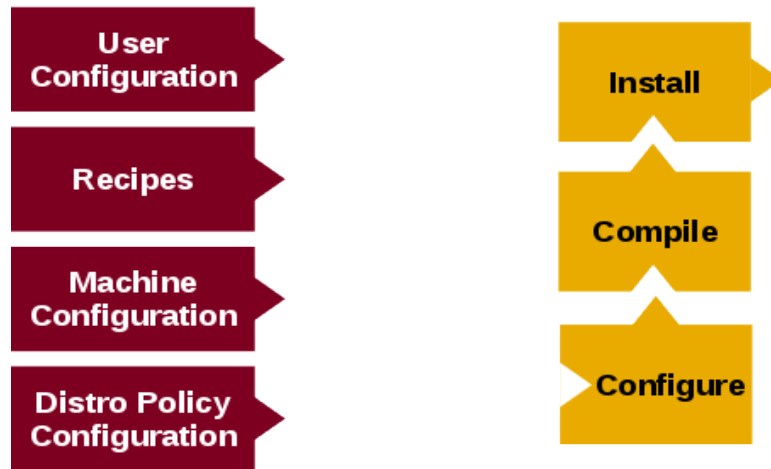
* Complete list includes: http, ftp, https, git, svn, perforce, mercurial, bazaar, cvs, osc, repo, ssh, and svk and the unpacker can cope with tarballs, zip, rar, xz, gz, bz2, and so on.

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

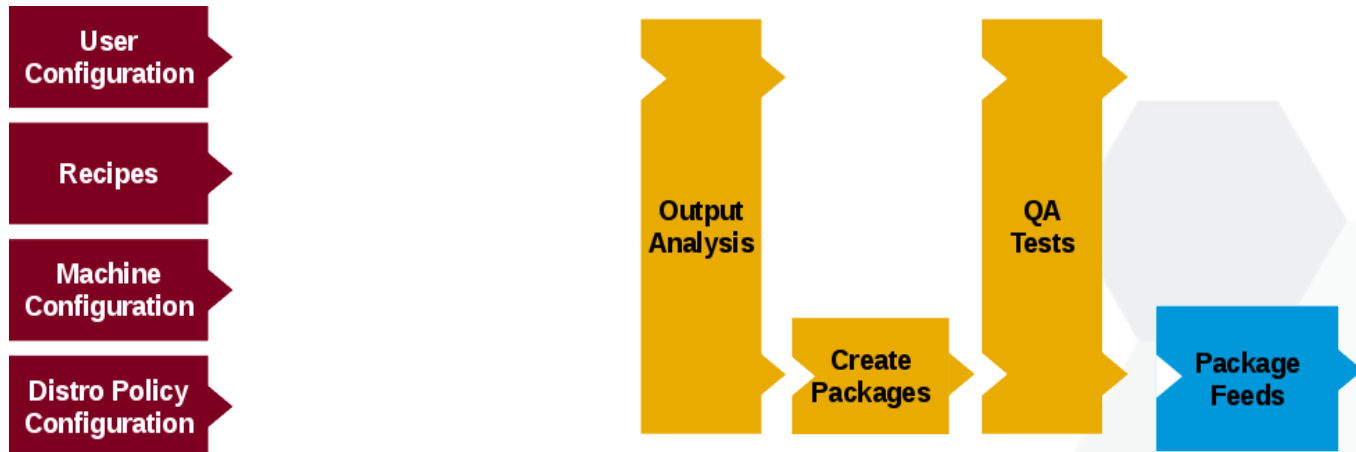


- **Recipe specifies configuration and compilation rules**
 - Various standard build rules are available, such as autotools and gettext
 - Standard ways to specify custom environment flags
 - Install step runs under 'pseudo', allows special files, permissions and owners/groups to be set

- **Recipe example**

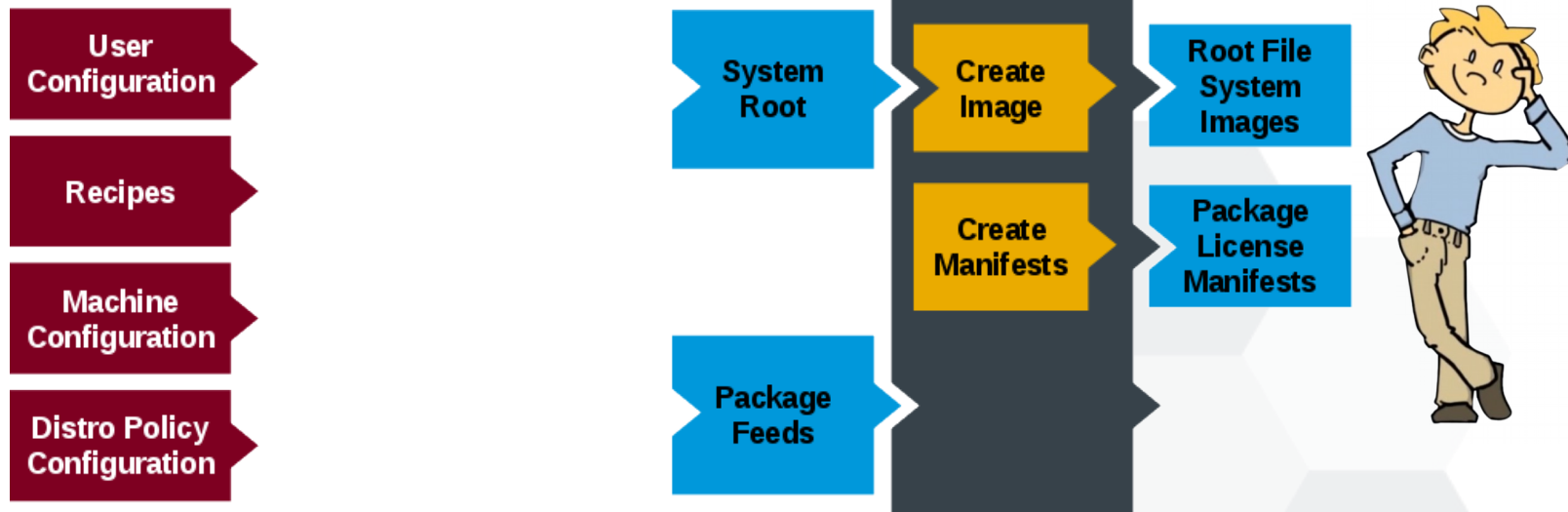
```
SUMMARY = "GNU Hello world Application"  
SECTION = "examples"  
  
LICENSE = "GPLv2+"  
LIC_FILES_CHKSUM = "file://COPYING;md5=751419260aa954499f7abaabaa882bbe"  
PR = "r0"  
SRC_URI = "${GNU_MIRROR}/hello/hello-${PV}.tar.gz"  
inherit autotools gettext
```

Output Analysis / Packaging



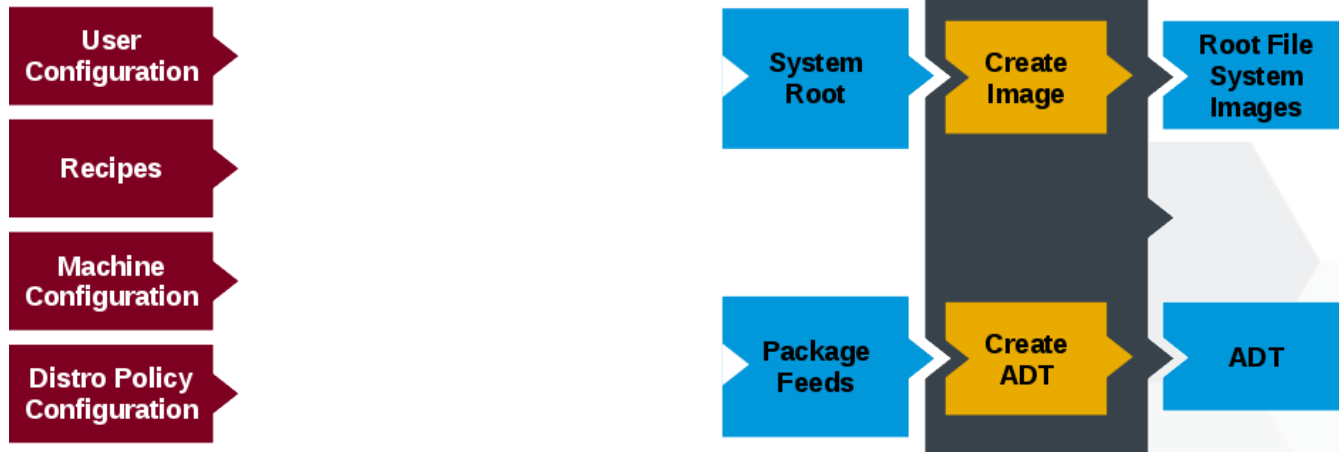
- **Output Analysis:**
 - Categorize generated software (debug, dev, docs, locales)
 - Split runtime and debug information
- **Perform QA tests (sanity checks)**
- **Package Generation:**
 - Support the popular formats, RPM, Debian, and ipk
 - Set preferred format using `PACKAGE_CLASSES` in `local.conf`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

Image Generation



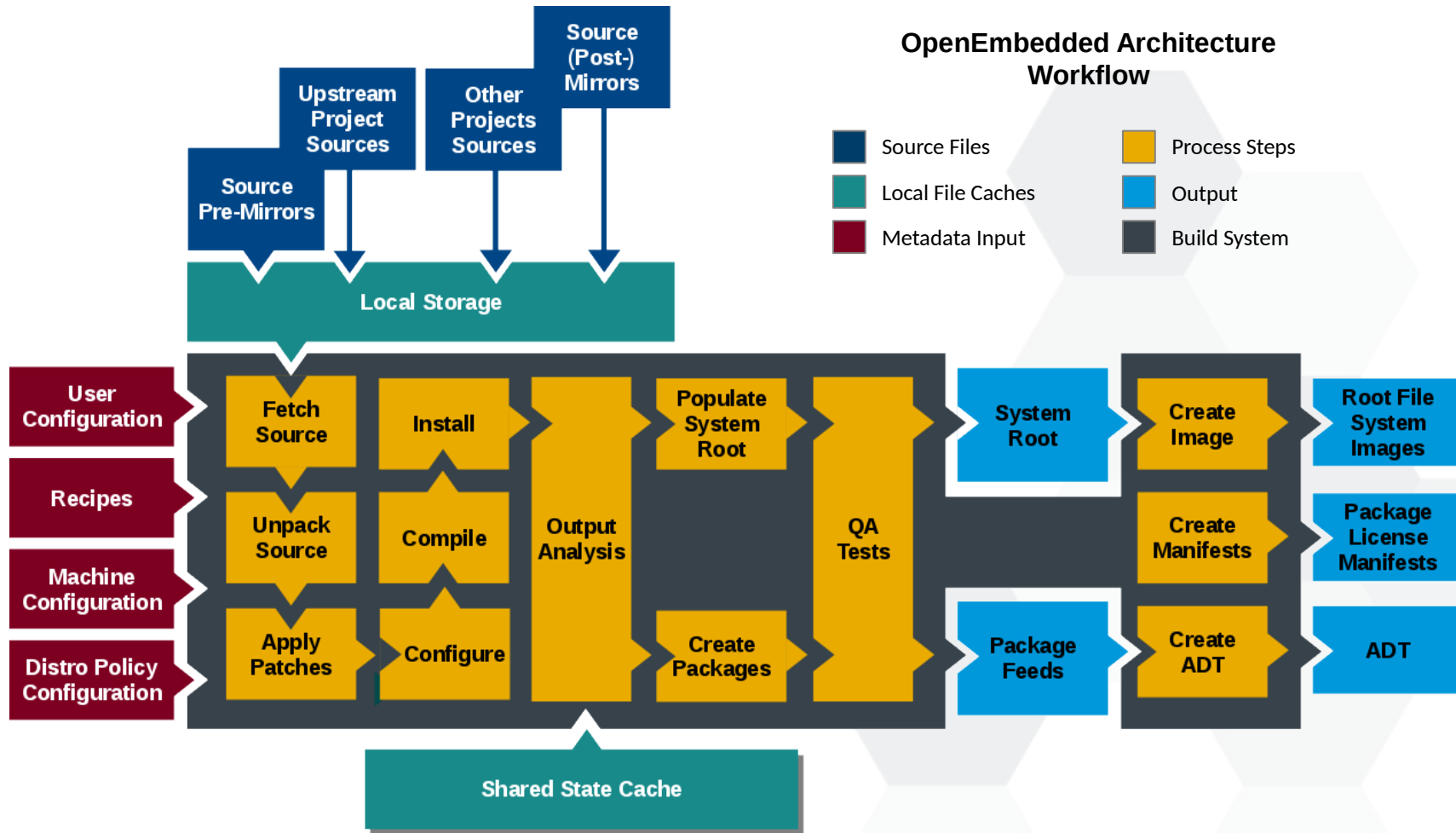
- Images are constructed using the packages built earlier and put into the Package Feeds
- Decisions of what to install on the image is based on the minimum defined set of required components in an image recipe. This minimum set is then expanded based on dependencies to produce a package solution.
-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

SDK Generation



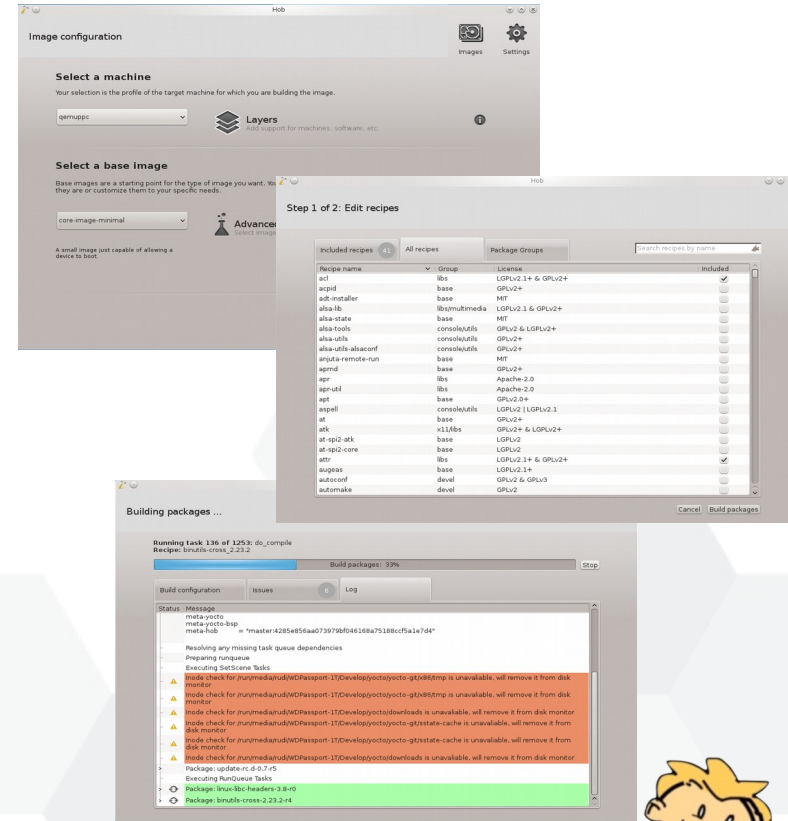
- A specific SDK recipe may be created. This allows someone to build an SDK with specific interfaces in it. (i.e. meta-toolchain-gmae)
- SDK may be based on the contents of the image generation
- SDK contains native applications, cross toolchain and installation scripts
- May be used by the Eclipse Application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

Build System Workflow



HOB – A GUI for BitBake

- **Configure Target Image**
 - Target Machine
 - Base Image
 - Package Selections
 - Root File Systems
 - Packaging Systems
- **Configure Build Environment**
 - Output Directories
 - Source Mirrors
 - Parallel Processing Options
- **Launch Build Process**
- **Review Logs and Build Results**
- **Deploy Build Results**



hob /häb/

1. Noun: A flat metal shelf at the side or back of a fireplace, having its surface level with the top of the grate.
2. A sprite of hobgoblin.



Toaster – A Web GUI for BitBake

● Build Analysis Tool

- Build Data Collection
- Database Storage
- Local and Remote Usage
- Requires BitBake Knotty or HOB to configure & launch builds
- Django Data Models/Web Interface

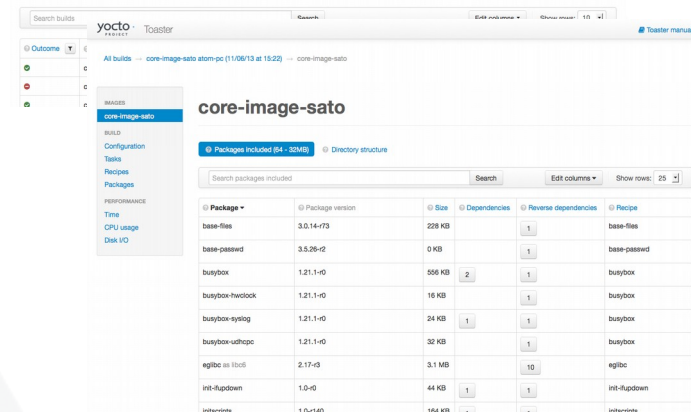
● Local Usage

- Install Django 1.6 and South 0.8.4
 - \$ pip install django==1.6
 - \$ pip install south==0.8.4
- Source Build Environment
 - \$ source poky/oe-init-build-env build
- Launch Toaster
 - \$ source toaster start
- Launch Build
 - \$ bitbake -k core-image-sato
- View Build Results
 - \$ xdg-open http://localhost:8000
- Stop Toaster
 - \$ source toaster stop



The screenshot shows the 'Recent builds' section of the Toaster web interface. It lists several build entries with their names, progress bars, and status indicators. The entries are:

- core-image-sato (+3 qemu66) - ETA: 16:34
- core-image-minimal qemuarm - ETA: 15:52
- core-image-sato atom-pc (15:22) - 4 warnings - Build time: 00:36:55
- core-image-x11 qemu66 (12:01) - 3 errors, 10 warnings - Build time: 00:27:45
- core-image-sato atom-pc (11:54) - 4 warnings - Build time: 00:36:55



The screenshot shows the 'All builds' section of the Toaster web interface, specifically the 'core-image-sato' build. It displays a table of packages included in the build, with columns for Package, Package version, Size, Dependencies, Reverse dependencies, and Recipe.

Package	Package version	Size	Dependencies	Reverse dependencies	Recipe
base-files	3.0.14-r73	228 KB	1		base-files
base-passwd	3.0.26-r2	0 KB	1		base-passwd
busybox	1.21.1-r0	568 KB	2	1	busybox
busybox-hwclock	1.21.1-r0	16 KB	1	1	busybox
busybox-syslog	1.21.1-r0	24 KB	1	1	busybox
busybox-udhcpd	1.21.1-r0	32 KB	1	1	busybox
eglibc-iss-ftsdl	2.17-r3	3.1 MB	10		eglibc
init-flutdown	1.0-r0	44 KB	1	1	init-flutdown
inetcron	1.0-r140	144 KB	-	-	inetcron



Application Development Toolkit

Core Components

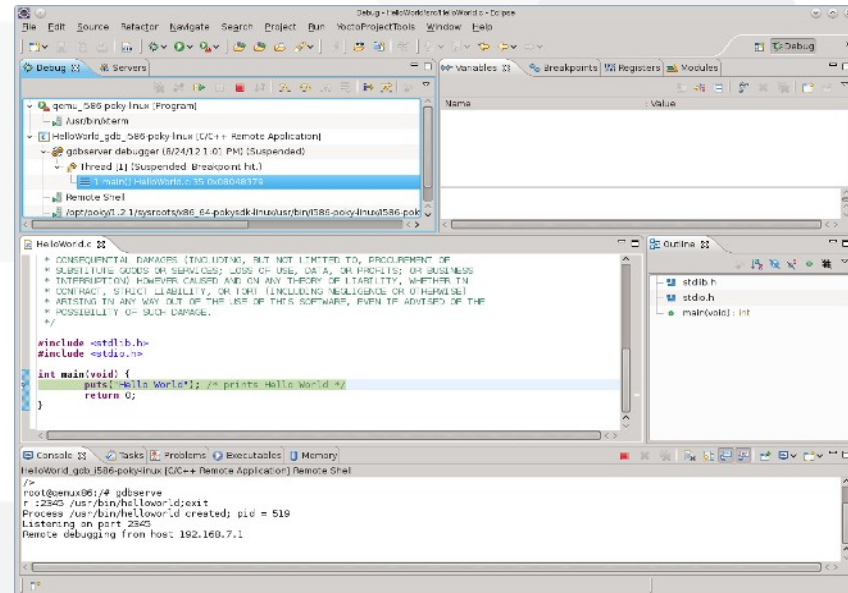
- Cross Development Toolchain
- System Root
- QEMU Emulator

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





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}"
```

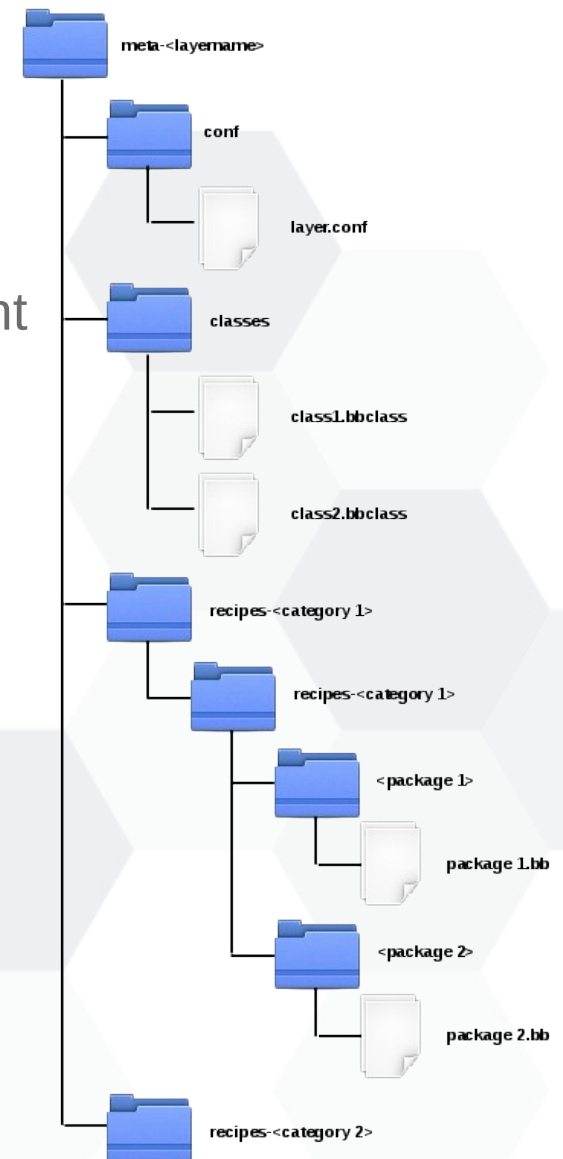
```
# We have recipes-* directories, add to BBFILES
BBFILES += "${LAYERDIR}/recipes-*/**/*.bb
           ${LAYERDIR}/recipes-*/**/*.bbappend"
```

```
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

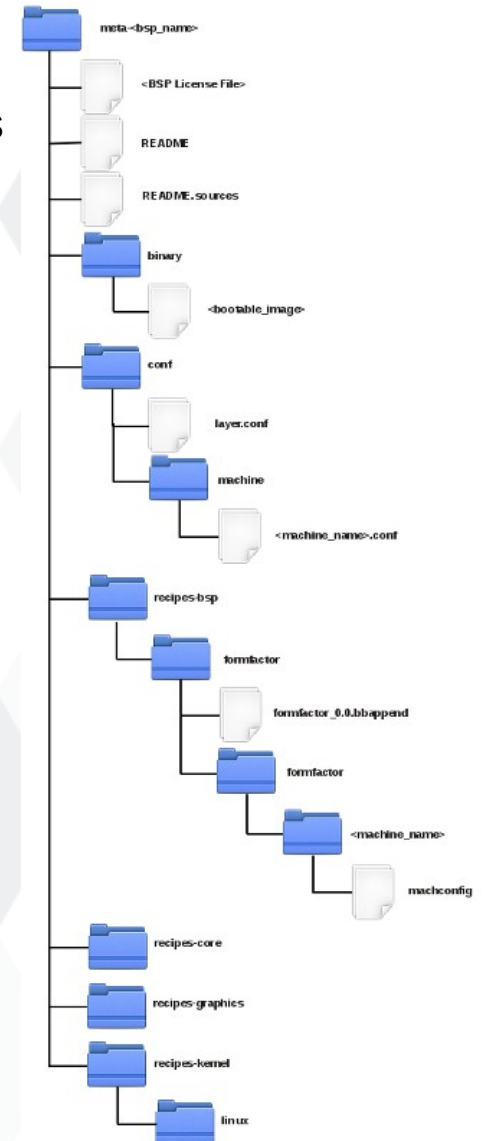
- Configuration and recipes for hardware platforms
- Dependent on core layers
- Extend core layer recipes and configuration
- Do not contain build system and/or tools

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

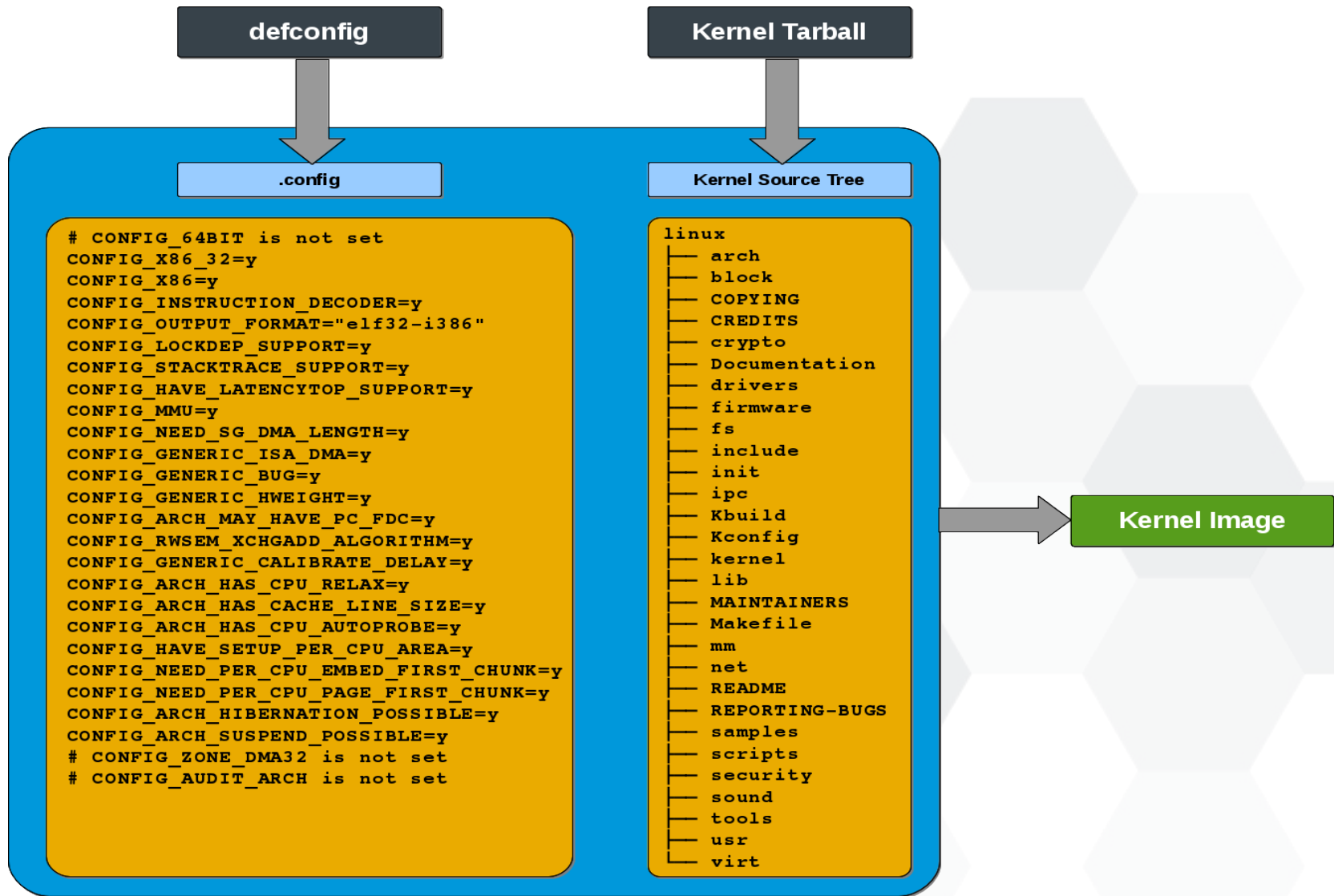
- **Three Approaches**

- 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"

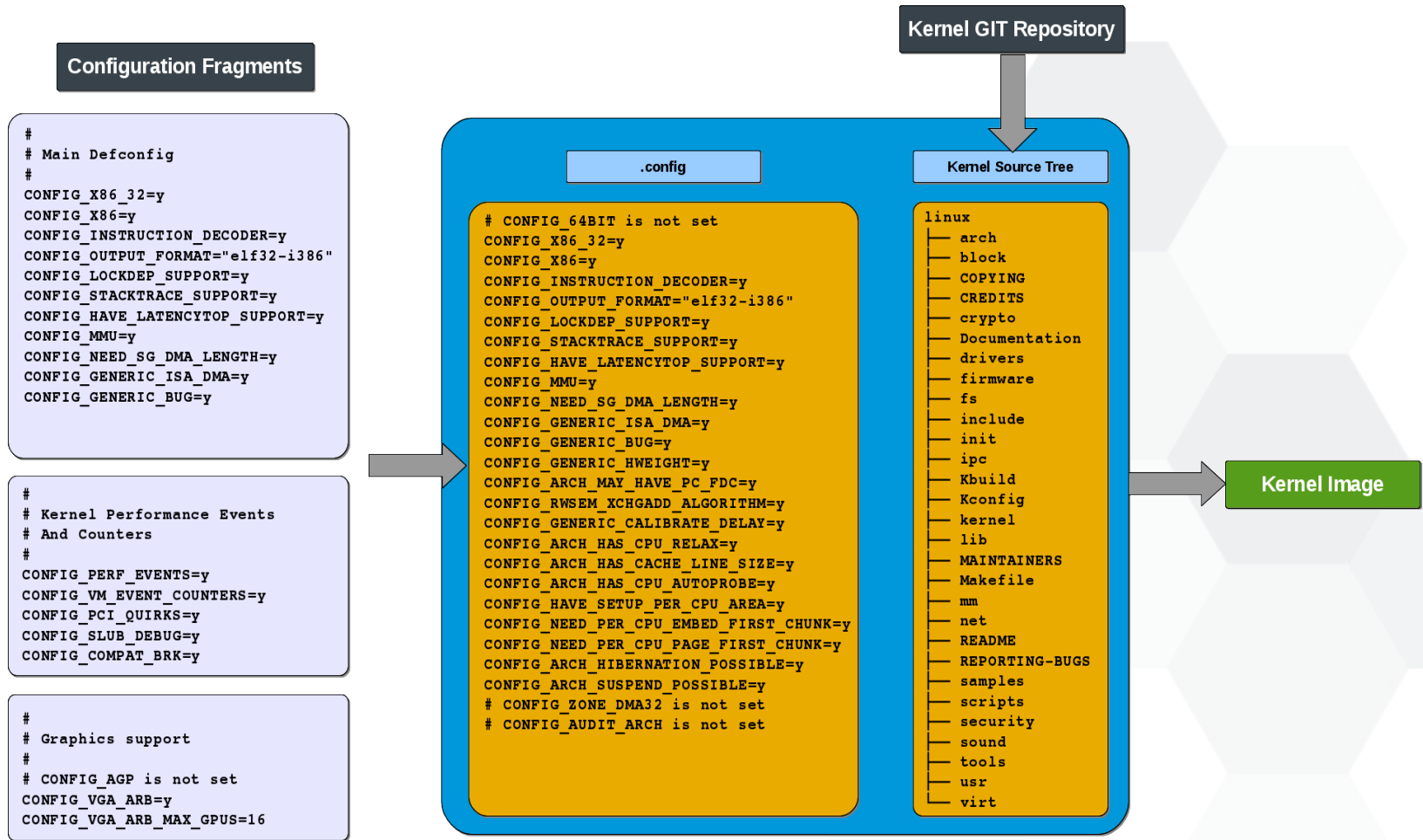
LINUX_VERSION ?= "3.11.0"
LINUX_VERSION_EXTENSION ?= "-custom"

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"

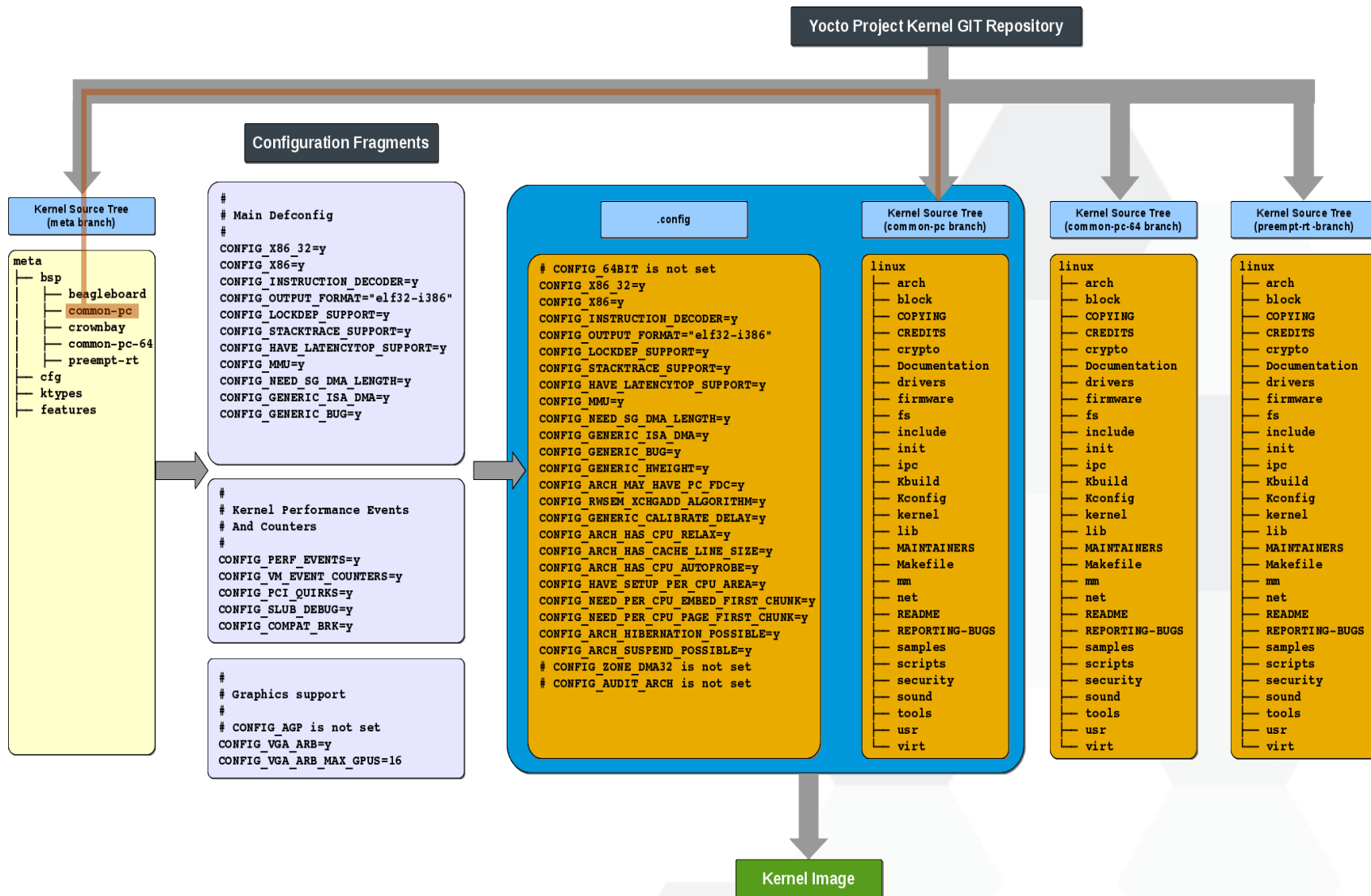
SRCREV = "720e07b4c1f687b61b147b31c698cb6816d72f01"

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 ?= "e4c12f12e61a29b6605c4fcbcf6db6e18bd7b4e4"
```

```
SRCREV_machine_qemux86 ?= "dd089cb5ba37ea14e8f90a884bf2a5be64ed817d"
```

```
SRCREV_machine_qemux86-64 ?= "dd089cb5ba37ea14e8f90a884bf2a5be64ed817d"
```

```
SRCREV_machine ?= "dd089cb5ba37ea14e8f90a884bf2a5be64ed817d"
```

```
SRCREV_meta ?= "8482dcd6f68f9f7501118f4c01fdcb8f851882997"
```

```
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

La Fin

Done for today but there is much more...

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!



A decorative graphic consisting of a cluster of semi-transparent, light gray hexagons arranged in a honeycomb pattern, located in the upper left quadrant of the slide.

**Thank you for your
participation!**

yocto ·
PROJECT

THE
LINUX
FOUNDATION