Yocto project

Yocto Project Introduction

2015

issue.hsu@gmail.com

Outline

- What is Yocto Project
- Yocto Project Workflow
- References

What is Yocto Project

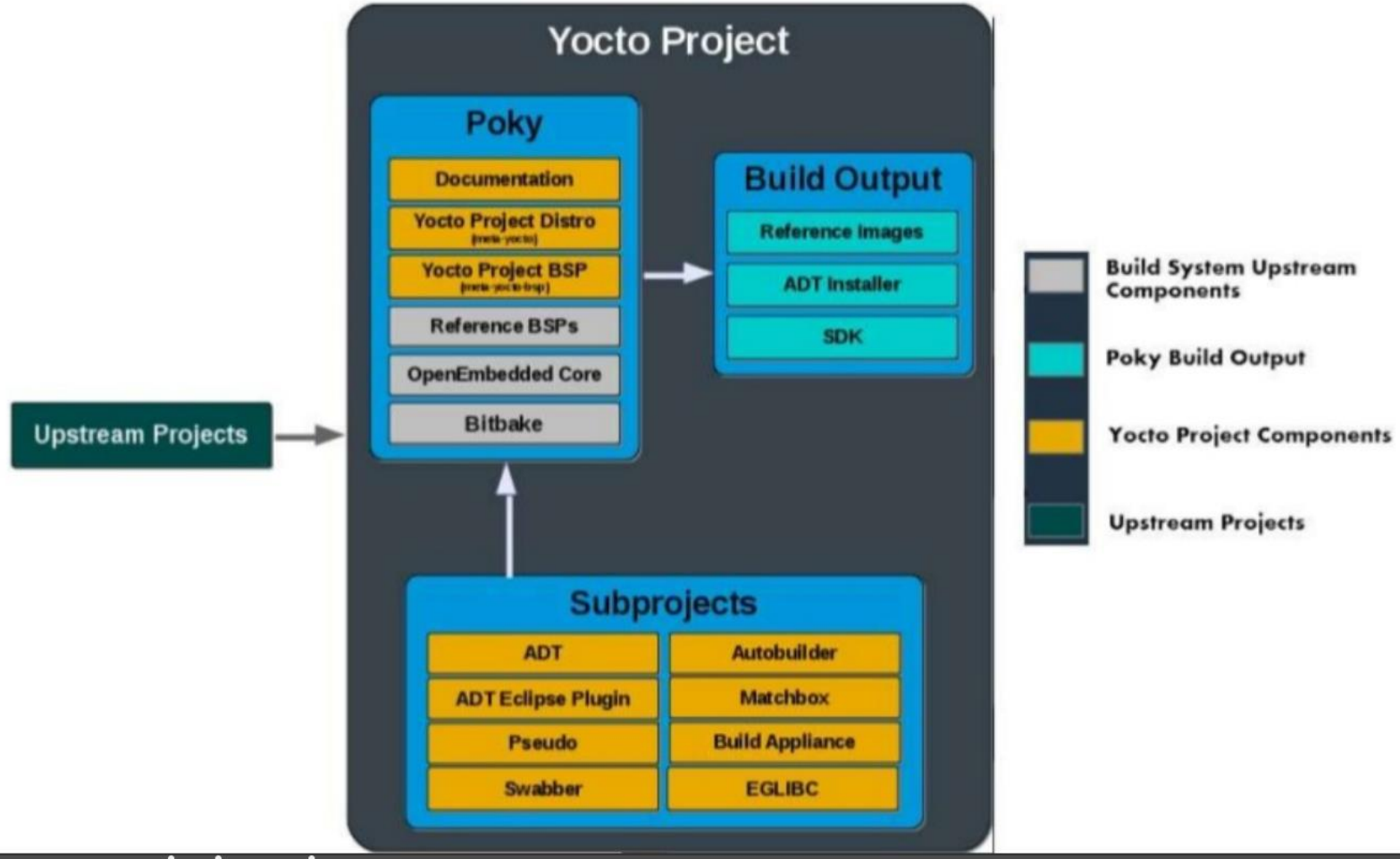
What is Yocto Project

- Overview
- Components
- Yocto Project VS OpenEmbedded
- Concept

Clip slide

Overview

- Yocto Project = Poky + Tools + Upstreams



Components

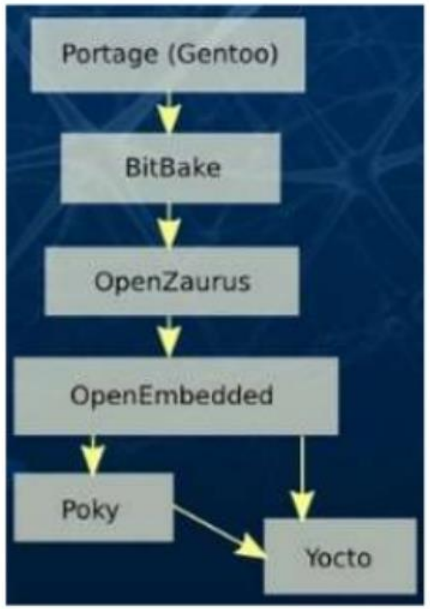
- **Poky = BitBake + metadata**
  - Poky: reference build system used by the Yocto Project. It is a collection of projects and tools, used to bootstrap a new distribution based on the Yocto Project
  - BitBake: the build engine. It is a task scheduler, like make. It interprets configuration files and recipes (also called metadata) to perform a set of tasks, to download, configure and build specified packages and filesystem images
  - Metadata: task configurations and definitions
- **Metadata = configuration (.conf) + classes (.bbclass) + recipes (.bb)**
- **OpenEmbedded Core**
  - a set of base layers. It is a set of recipes, layers and classes which are shared between all OpenEmbedded based systems

Components

- **Sub-Projects**
  - Hob: graphical user interface for BitBake
  - 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 VS 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
  - Large library of recipes to cross-compile over 1000 packages
- The Yocto Project is focused on enabling Commercial Product Development
  - Provides a reference distribution policy and root file system blueprints for building Linux OS stacks
  - 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

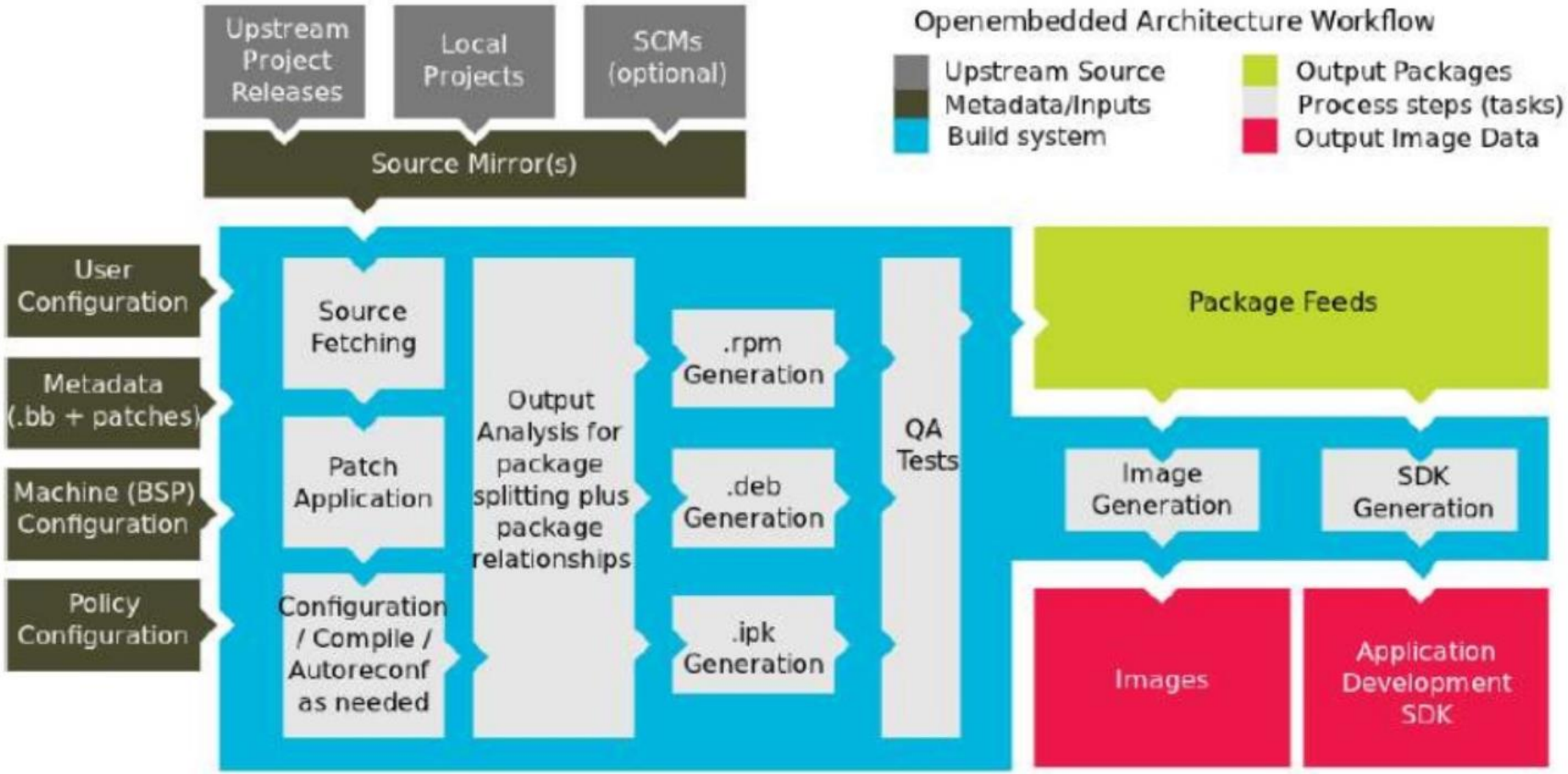


Key Concept

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

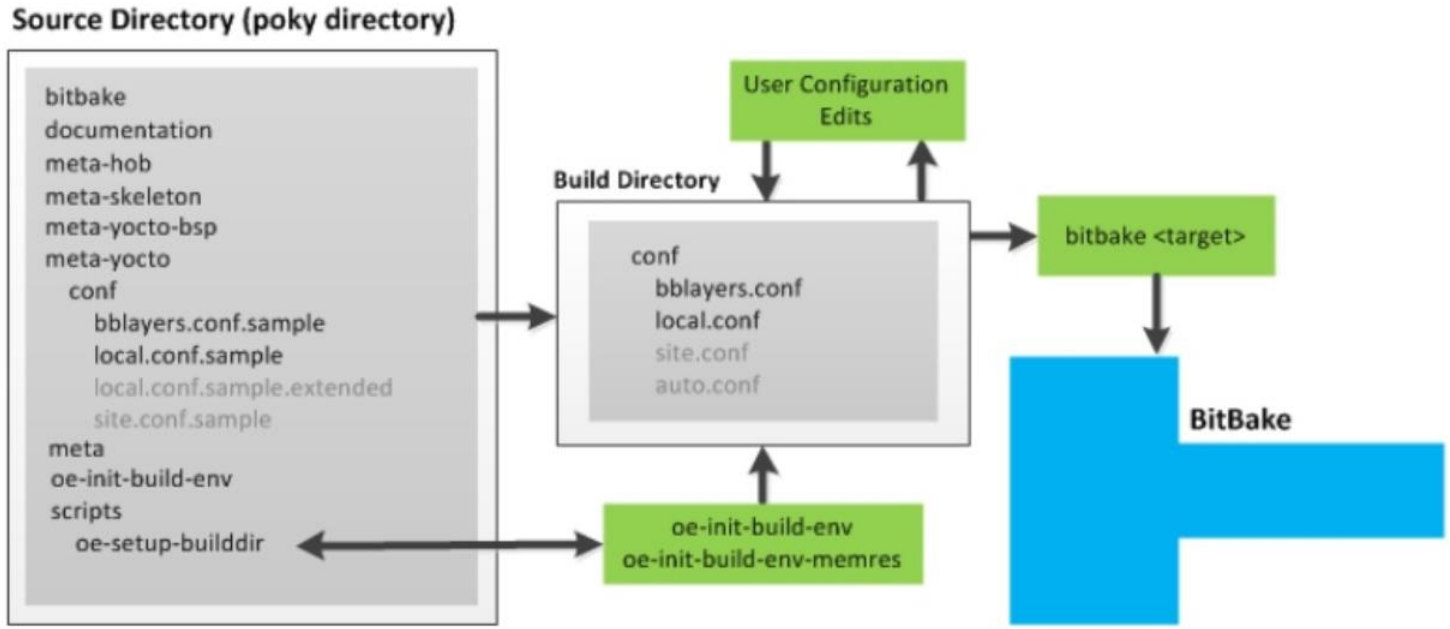
Yocto Project Workflow

Overview of Yocto Project Workflow



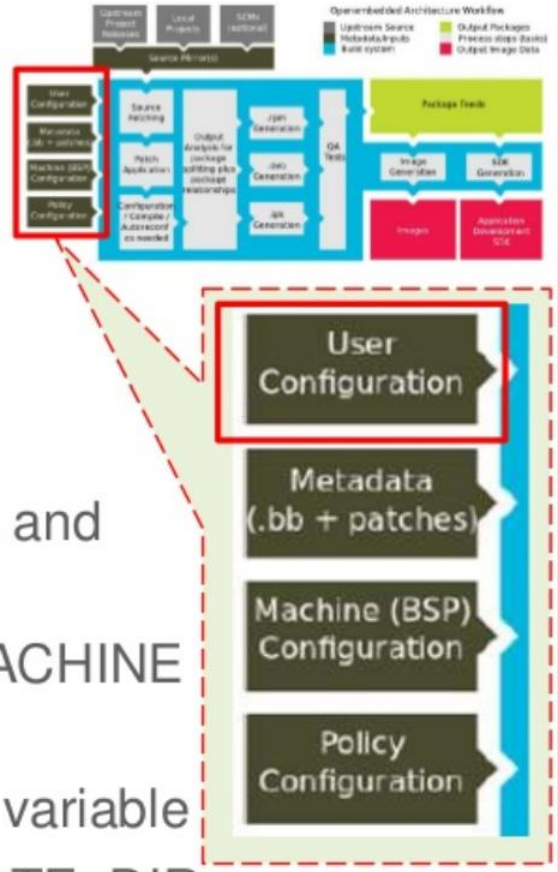
Yocto Project Workflow Configurations

- $ source oe-init-build-env [build_dir]
- $ build_dir> bitbake <target>



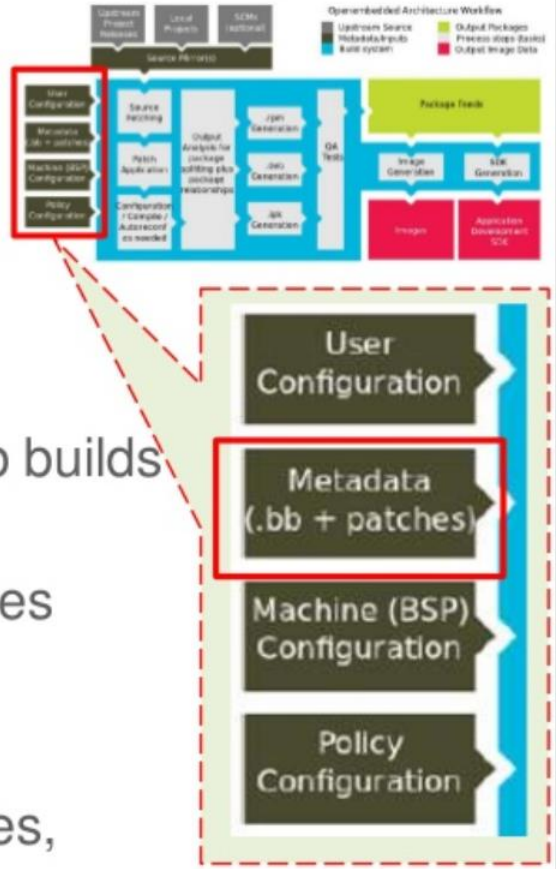
Configuration

- User Configuration
  - conf/local.conf – some things to set:
  - Parallelism Options: Controlled by the BB_NUMBER_THREADS, PARALLEL_MAKE, and BB_NUMBER_PARSE_THREADS variables
  - Target Machine Selection: Controlled by the MACHINE variable
  - Download Directory: Controlled by the DL_DIR variable
  - Shared State Directory: Controlled by the SSTATE_DIR variable
  - Build Output: Controlled by the TMPDIR variable



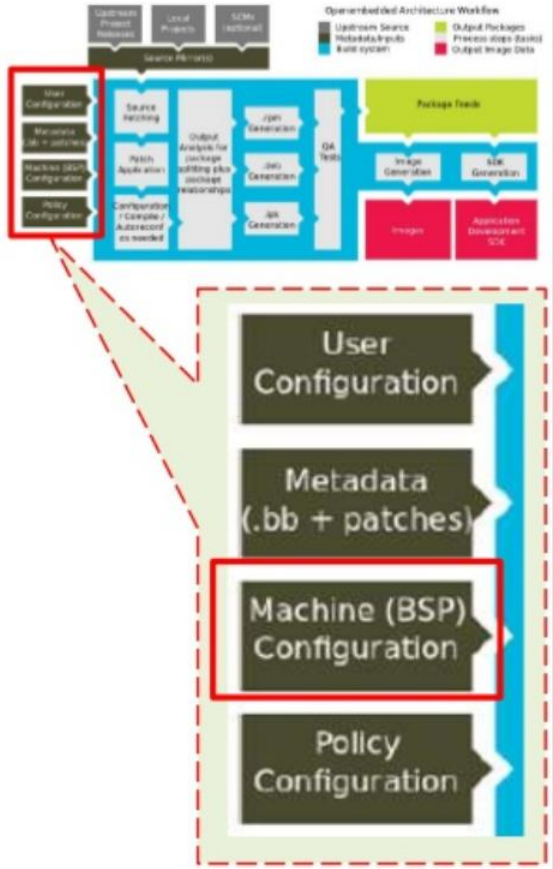
Configuration

- Metadata and Patches:
  - Recipes for building packages
  - Eg, meta/recipes-core/coreutils/coreutils_6.9.bb builds the core utilities (version 6.9) and installs them
  - meta/recipes-core/coreutils/coreutils-6.9/ includes patches, also could include extra files to install
  - Can be extended and enhanced via “layers”
  - Software layers contain user-supplied recipe files, patches, and append files



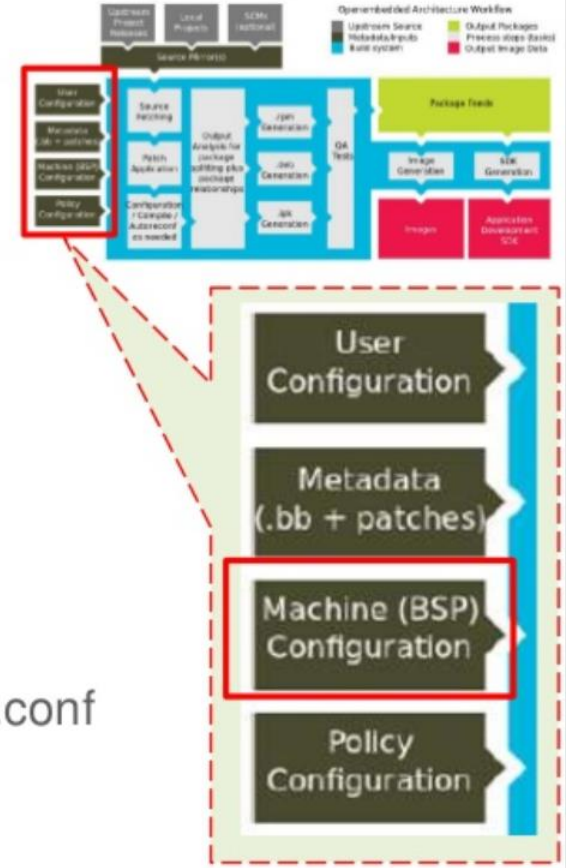
Configuration

- BSPs and Machine Configurations:
  - Configuration files to describe a machine
    - Processor/SOC tuning files
  - May include Linux kernel enhancements
  - Includes board specific kernel configuration
  - Machine specific package modifications
  - BSP layers provide machine configurations
    - conf/machine/machine.conf
    - conf/layer.conf
    - the layer is dedicated to specific recipes by function: recipes-bsp, recipes-core, recipes-graphics, and recipes-kernel



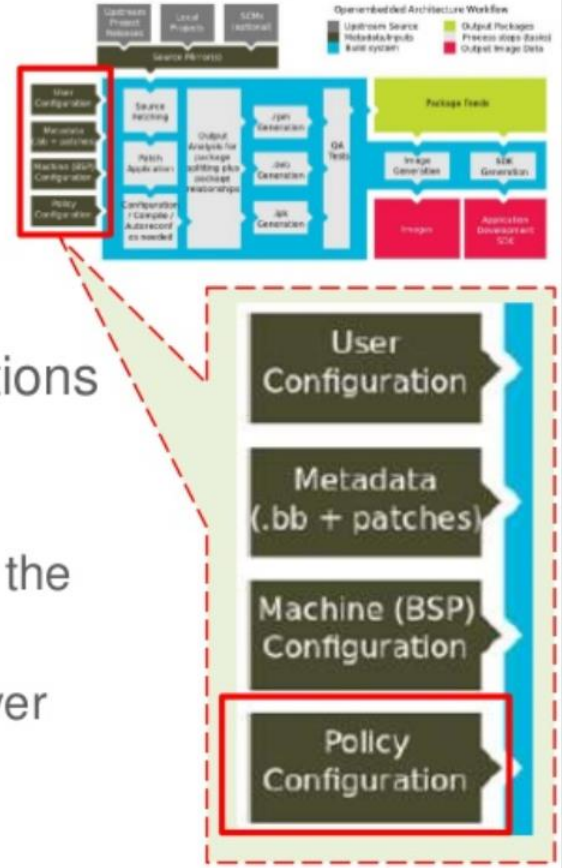
Configuration

- Machine settings are specified in a layer's conf/machine/xxx.conf file(s)
  - Sandy Bridge + Cougar Point
    - meta-intel/conf/meta-sugarbay/machine/sugarbay.conf
  - Routerstation Pro (MIPS)
    - meta-yocto-bsp/conf/machine/routerstationpro.conf



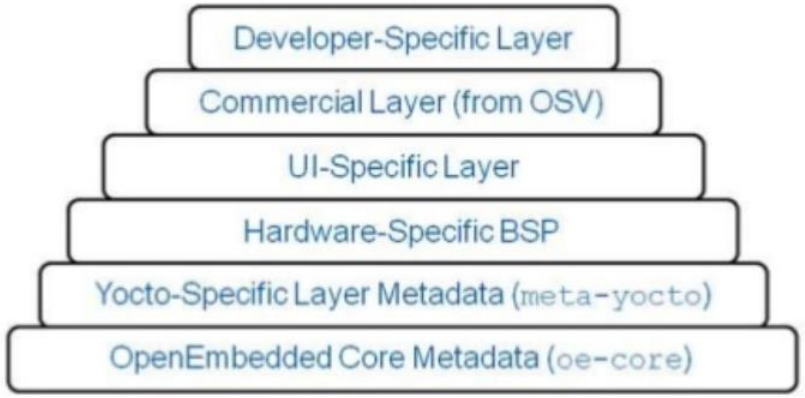
Configuration

- Policy Configurations:
  - The distribution layer provides policy configurations for your distribution
    - classes: Class files (.bbclass) hold common functionality that can be shared among recipes in the distribution
    - conf: This area holds configuration files for the layer (conf/layer.conf), the distribution (conf/distro/distro.conf), and any distribution-wide include files
    - recipes-*: Recipes and append files that affect common functionality across the distribution. This area could include recipes and append files to add distribution-specific configuration, initialization scripts, custom image recipes, and so forth

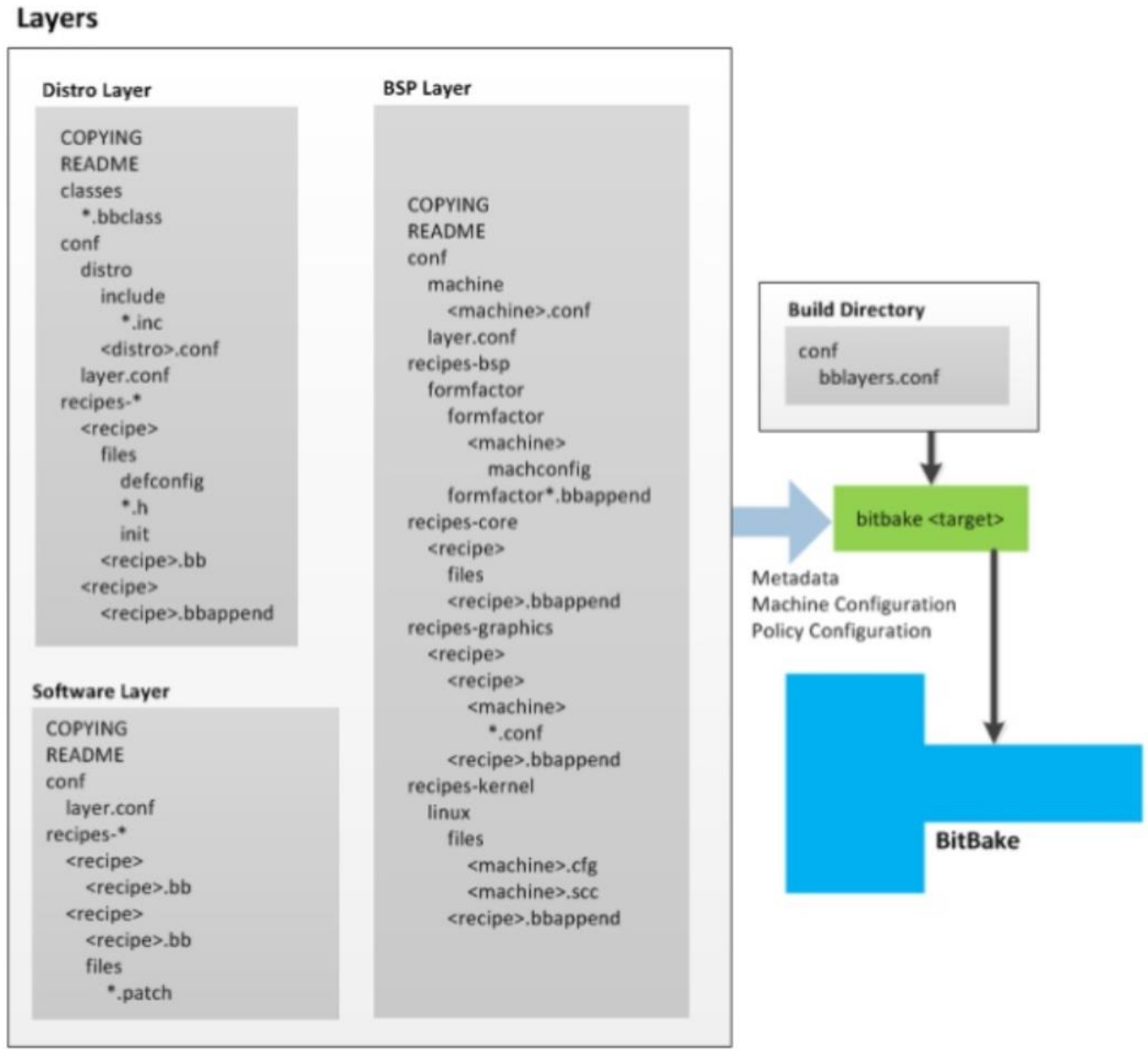


Yocto Project Layer

- Layers contain extensions and customizations to base system
  - Can include image customizations, additional recipes, modifying recipes, adding extra configuration
  - Really just another directory to look for recipes or recipe extensions
  - Added to the BBLAYERS variable in build/conf/bblayers.conf
- Layers are grouped by functional components
- Common Examples of Layers
  - Custom Toolchains (compilers, debuggers, profiling tools, etc)
  - Distribution specifications and customizatic
  - Functional areas (selinux, networking, etc)
  - OSV components
  - Project level changes
  - BSP/Machine specific components



Configuration Layers expanded Figure

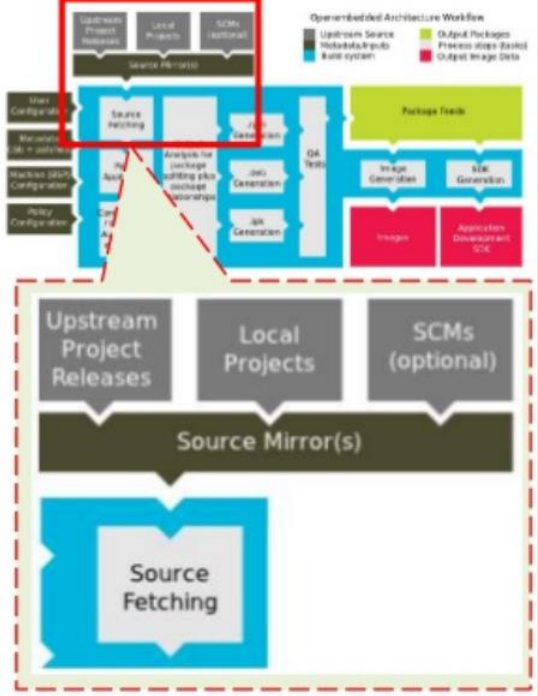


High-level Overview of Poky Task Execution

- BitBake supports both *_prepend* and *_append* as a method of extending task functionality by injecting code indicated by using prepend or append suffix into the beginning or end of a task.

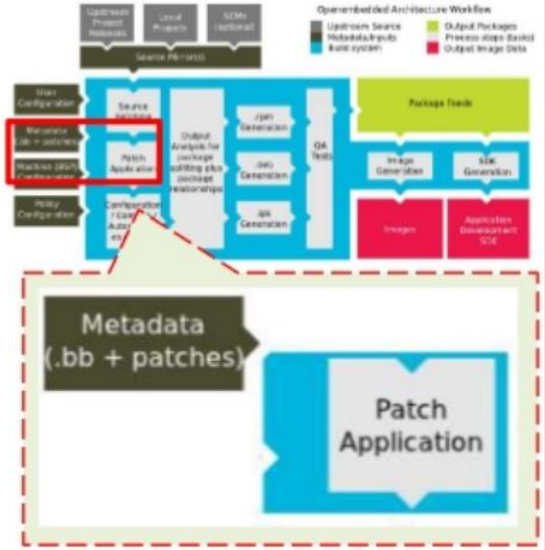
Source Fetching

- Recipes call out location of all sources, whether on the internet or local (Look for SRC_URI in *.bb files)
- Bitbake can get sources from git, svn, bzr, from tarballs, and many, many more
- Versions of packages can be fixed or updated automatically (Add SRCREV_pn- PN = "${AUTOREV}" to local.conf)
- Yocto Project sources mirror available as a fallback, if the sources move on the internet

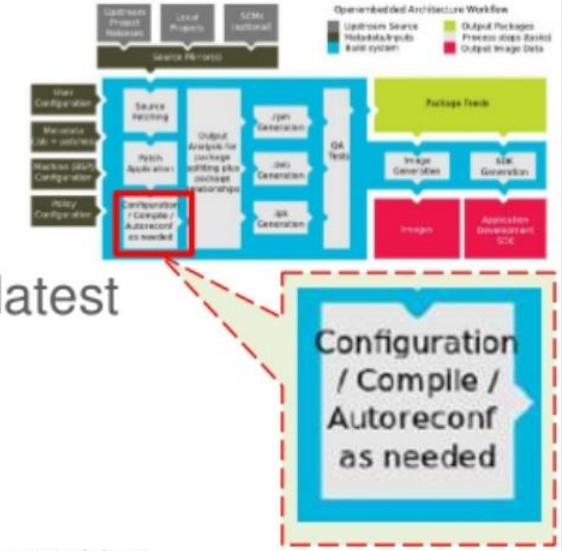


Patching

- Once sources are obtained, the patches are applied
- This is a good place to patch the software yourself
- However, we encourage you to contribute development upstream whenever possible



Configuration/Compile



- Autoconf can be triggered automatically to ensure latest libtool is used

```
DESCRIPTION = "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
```

- CFLAGS can be set

```
CFLAGS_prepend = "-I ${S}/include "
```

- Install task to set modes, permissions, target directories, done by “pseudo”

```
do_install () {
  oe_runmake install DESTDIR=${D} SBINDIR=${sbindir}
   MANDIR=${mandir} }
```

Packaging

- The most popular package formats are supported: RPM, Debian, and ipk
  - Set PACKAGE_CLASSES in conf/local.conf
- You can split into multiple packages using PACKAGES and FILES in a *.bb file
  - PACKAGES =+ "sxpm cxpm"
  - FILES_cxpm = "${bindir}/cxpm"
  - FILES_sxpm = "${bindir}/sxpm"

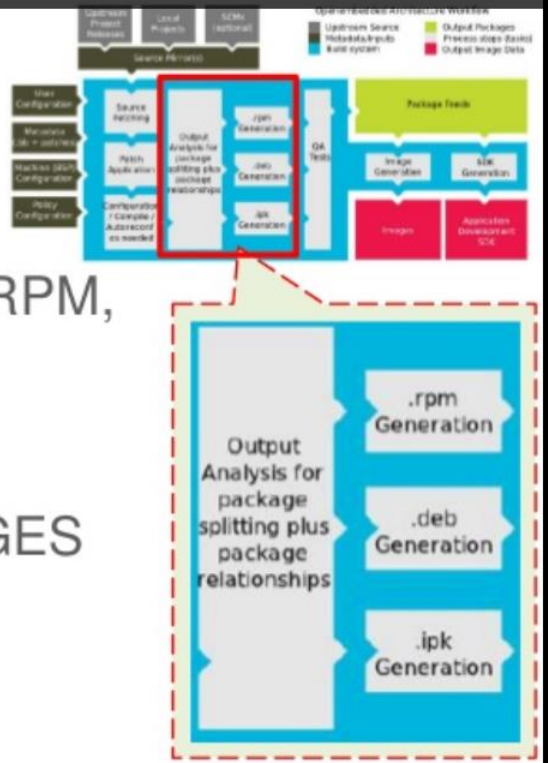
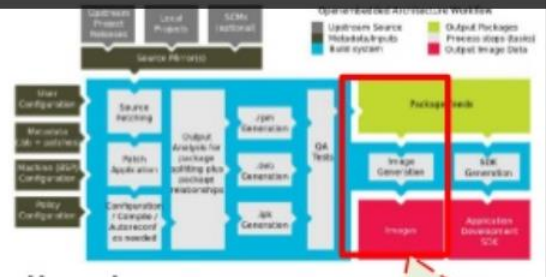
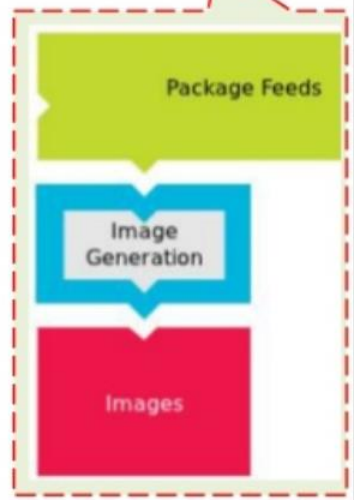


Image Generation



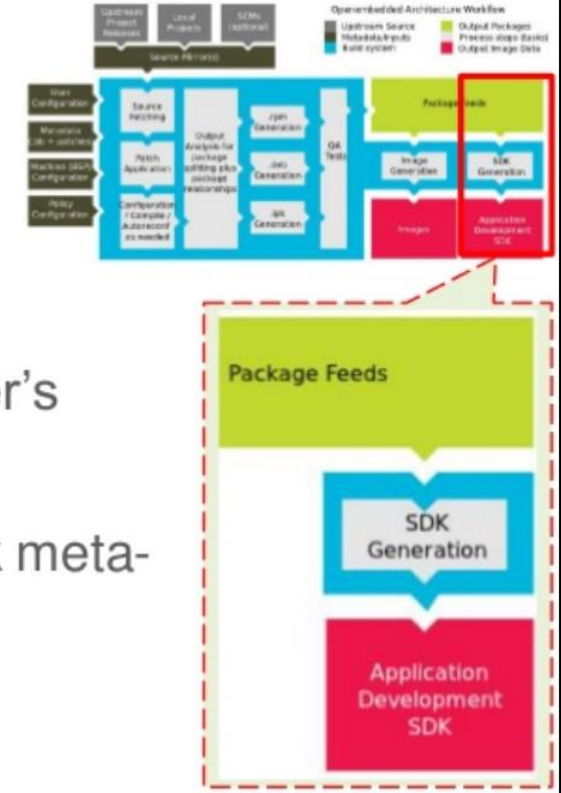
- Images are constructed using the packages built earlier in the process
- Uses for these images:
  - Live Image to boot a device
  - Root filesystem for QEMU emulator
  - Sysroot for App development



- Yocto Project lets you customize your embedded Linux OS

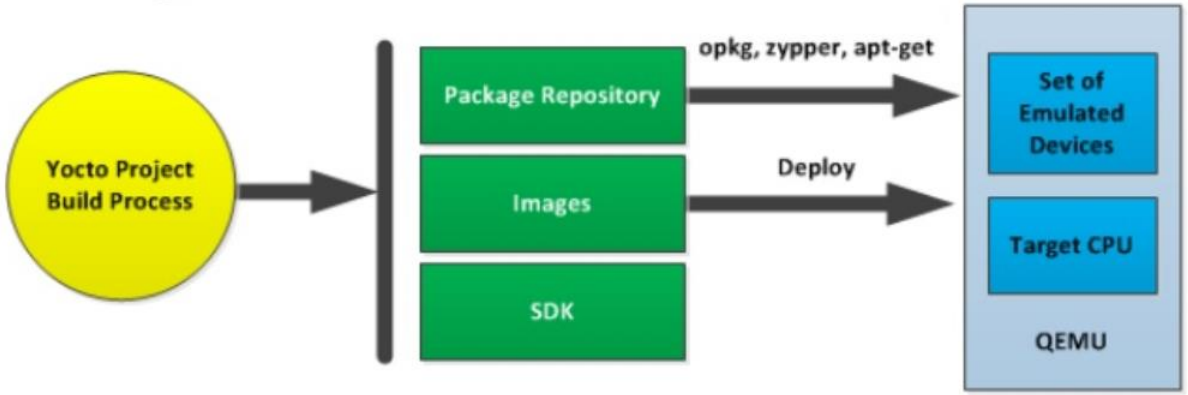
ADT Generation

- Cross toolchain and installation script generated
- This can be used to set up an application developer's cross development environment to create apps
- MACHINE=qemuarm bitbake poky-image-sato-sdk meta-toolchain package-index
- QEMU built for target architecture emulation



Preparing for Application Development

Running on QEMU



$ git clone http://git.yoctoproject.org/git/poky

$ cd poky

$ git checkout -b fido origin/fido

$ source oe-init-build-env

$ MACHINE=qemuarm bitbake core-image-sato

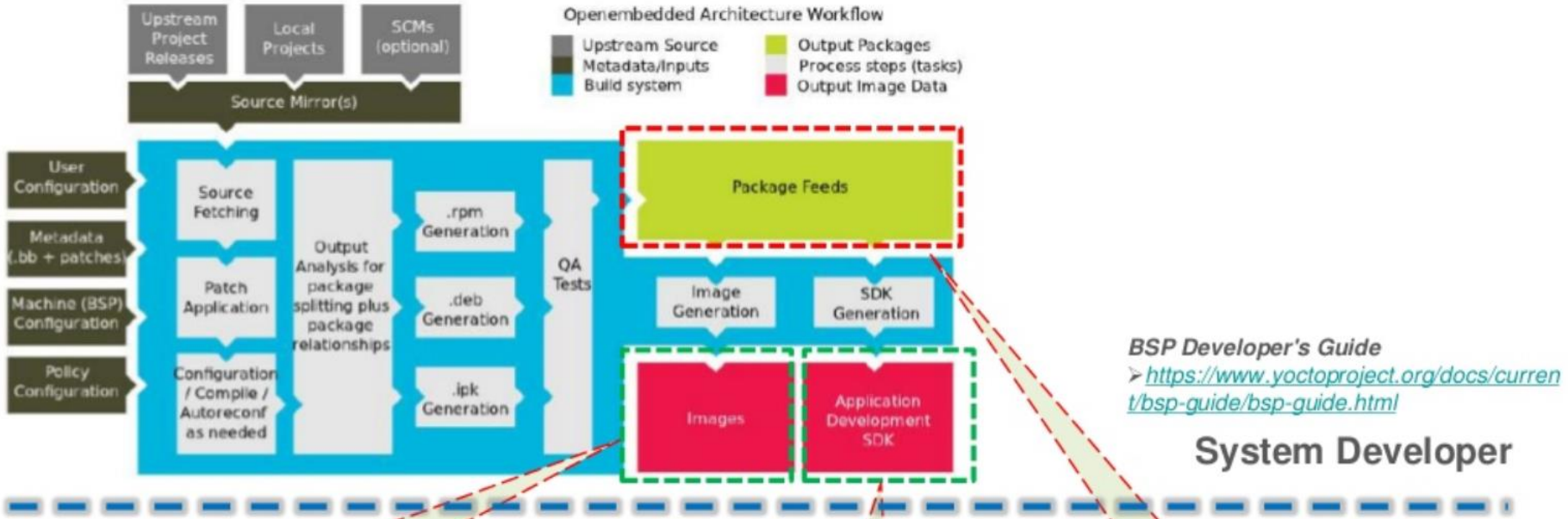
$ runqemu qemuarm core-image-sato

$ MACHINE=qemuarm64 runqemu /path/to/image.bin
/path/to/rootfs_image.ext4

Reference

➢ *https://www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.html*

Setting up the App Developer



BSP Developer's Guide
➢ *https://www.yoctoproject.org/docs/current/bsp-guide/bsp-guide.html*

System Developer

Application Developer

Application Developer's Guide
➢ *http://www.yoctoproject.org/docs/current/adt-manual/adt-manual.html*

- Sysroot (Bootable Linux filesystem tree with development headers)

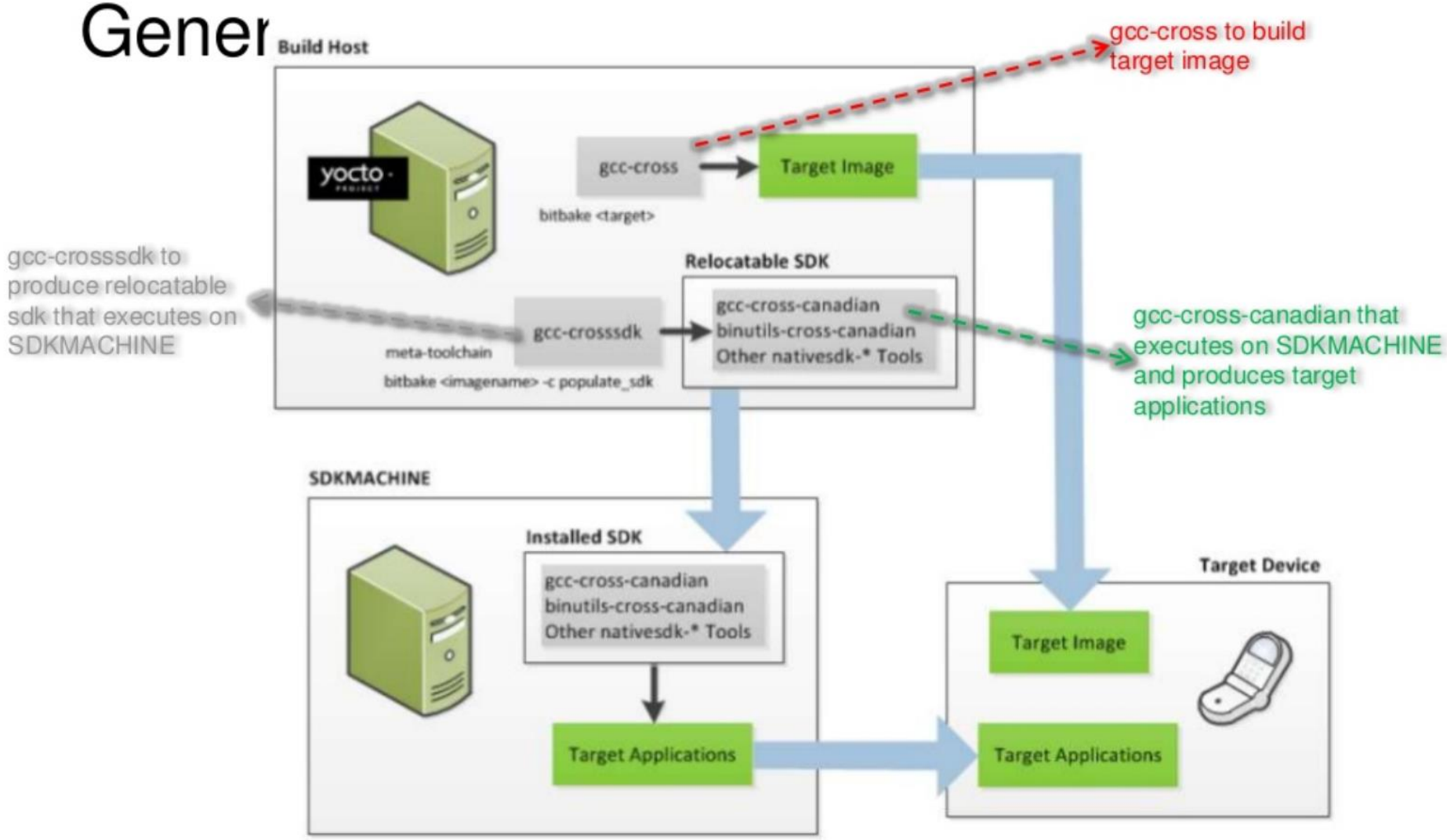
eclipse
Yocto plug-ins

- Cross toolchain installation (such as: /opt/poky)

- Package Repository (networked or local)

Yocto Project helps set up the embedded application development

Cross-Development Toolchain Gener



Building a Toolchain Installer

- *$ MACHINE=qemuarm bitbake meta-toolchain [-c populate_sdk]*
- When the bitbake command completes, the toolchain installer will be in tmp/deploy/sdk
- By default, this toolchain does not build static binaries
  - If you want to use the toolchain to build these types of libraries, you need to be sure your image has the appropriate static development libraries
  - Use the IMAGE_INSTALL variable inside your local.conf file to install the appropriate library packages
  - Following is an example using glibc static development libraries:
    - IMAGE_INSTALL_append = " glibc-staticdev"

Reference:
➢ *http://www.yoctoproject.org/docs/1.8/adt-manual/adt-manual.html#optionally-building-a-toolchain-installer*

Using a Cross-Toolchain Tarball

- Get your toolchain installer using one of the following methods
  - Get it from http://downloads.yoctoproject.org/releases/yocto/yocto-1.8/toolchain/
  - Build your own toolchain installer, like previous page
    - tmp/deploy/sdk/
- Once you have the installer, run it to install the toolchain
  - *$ tmp/deploy/sdk/poky-glibc-i686-meta-toolchain-armv5e-toolchain-1.8.sh*
  - The first thing the installer prompts you for is the directory into which you want to install the toolchain
    - The default directory used is /opt/poky/1.8

Reference:

➢ *http://www.yoctoproject.org/docs/1.8/adt-manual/adt-manual.html#using-an-existing-toolchain-tarball*

Setting Up the Cross-Development Environment

- *$ /opt/poky/1.8/environment-setup-x86_64-poky-linux*
- When you run the setup script, many environment variables are defined

- SDKTARGETSYSROOT - The path to the sysroot used for cross-compilation
- PKG_CONFIG_PATH - The path to the target pkg-config files
- CONFIG_SITE - A GNU autoconf site file preconfigured for the target
- CC - The minimal command and arguments to run the C compiler
- CXX - The minimal command and arguments to run the C++ compiler
- CPP - The minimal command and arguments to run the C preprocessor
- AS - The minimal command and arguments to run the assembler
- LD - The minimal command and arguments to run the linker
- GDB - The minimal command and arguments to run the GNU Debugger
- STRIP - The minimal command and arguments to run 'strip', which strips symbols
- RANLIB - The minimal command and arguments to run 'ranlib'
- OBJCOPY - The minimal command and arguments to run 'objcopy'
- OBJDUMP - The minimal command and arguments to run 'objdump'
- AR - The minimal command and arguments to run 'ar'
- NM - The minimal command and arguments to run 'nm'
- TARGET_PREFIX - The toolchain binary prefix for the target tools
- CROSS_COMPILE - The toolchain binary prefix for the target tools
- CONFIGURE_FLAGS - The minimal arguments for GNU configure
- CFLAGS - Suggested C flags
- CXXFLAGS - Suggested C++ flags
- LDFLAGS - Suggested linker flags when you use CC to link
- CPPFLAGS - Suggested preprocessor flags

Reference:

➢ *http://www.yoctoproject.org/docs/1.8/adt-manual/adt-manual.html#setting-up-the-cross-development-environment*

Using an External Toolchain

- The fundamental steps you need to accomplish are as follows:
  - Understand where the installed toolchain resides. For cases where you need to build the external toolchain, you would need to take separate steps to build and install the toolchain.
  - Make sure you add the layer that contains the toolchain to your bblayers.conf file through the BBLAYERS variable.
  - Set the EXTERNAL_TOOLCHAIN variable in your local.conf file to the location in which you installed the toolchain.
- A good example of an external toolchain used with the Yocto Project is Mentor Graphics® Sourcery G++ Toolchain
  - You can see information on how to use that particular layer in the README file at http://github.com/MentorEmbedded/meta-sourcery/
  - You can find further information by reading about the TCMODE variable in the Yocto Project Reference Manual's variable glossary

Reference:

➢ *http://www.yoctoproject.org/docs/1.8/adt-manual/adt-manual.html#optionally-using-an-external-toolchain*

Extracting the Root Filesystem

- Here are some cases where you need to extract the root filesystem:
  - You want to boot the image using NFS
  - You want to use the root filesystem as the target sysroot. For example, the Eclipse IDE environment with the Eclipse Yocto Plug-in installed allows you to use QEMU to boot under NFS
  - You want to develop your target application using the root filesystem as the target sysroot
- Following is an example
  - *$ source /opt/poky/1.8/environment-setup-i586-poky-linux*
  - *$ runqemu-extract-sdk core-image-sato-sdk-qemux86-20110914118 31.rootfs.tar.bz2 $HOME/qemux86-sato*

References

Reference Manuals

- Yocto Project Reference Manual
  - http://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html
- Yocto Project Development Manual
  - http://www.yoctoproject.org/docs/1.8/dev-manual/dev-manual.html
- Bitbake
  - http://www.yoctoproject.org/docs/1.8/bitbake-user-manual/bitbake-user-manual.html

Trainings and Tutorials

- Yocto Training from Free Electrons
  - http://free-electrons.com/doc/training/yocto/yocto-slides.pdf
  - http://free-electrons.com/doc/training/yocto/yocto-labs.pdf
- Developer Day 2014 Materials
  - http://www.yoctoproject.org/bulk/devday-eu-2014/ypdd14-hallinan-intro-lab.pdf
- The Yocto Project: Introducing devtool
  - https://drive.google.com/file/d/0B3KGzY5fW7IaTDVxUXo3UDRvd2s/view
  - http://events.linuxfoundation.org/sites/events/files/slides/yocto_project_dev_workflow_elc_2015_0.pdf
- OpenEmbedded and Yocto introduction
  - http://wiki.kaeilos.com/index.php/Setup_an_external_toolchain_with_Yocto
- http://10.19.133.152/wiki/doku.php?id=yocto
- http://www.crashcourse.ca/wiki/index.php/Yocto_FAQ
- Building your own recipes from first principles
  - https://wiki.yoctoproject.org/wiki/Building_your_own_recipes_from_first_principles

External Toolchain Reference

- https://wiki.linaro.org/KenWerner/Sandbox/OpenEmbedded-Core
- https://git.linaro.org/openembedded/meta-linaro.git
- http://www.rocketboards.org/foswiki/Documentation/YoctoDoraBuildWithMetaAltera
- http://wiki.kaeilos.com/index.php/Setup_an_external_toolchain_with_Yocto
- https://developer.ridgerun.com/wiki/index.php?title=Yocto_RR_SDK_Integration
- http://elinux.org/images/c/c8/ExternalToolchainsInYocto.pdf

Modify Toolchain Options

- Toolchain component versions
  - poky/meta/conf/distro/
    - TCMODE
    - TCLIBC
    - GCCVERSION
    - BINUVERSION
    - GDBVERSION
    - GLIBCVERSION
- Target machine options
  - poky/meta/conf/macine/
    - DEFAULTTUNE
    - TUNEVALID
    - AVAILTUNES
    - TUNE_FEATURES

Other Important Variables

- Other define variables
  - poky/meta/conf/documentation.conf
    - SELECTED_OPTIMIZATION
    - FULL_OPTIMIZATION
    - BUILD_OPTIMIZATION
    - OVERRIDES
      - http://www.yoctoproject.org/docs/latest/bitbake-user-manual/bitbake-user-manual.html#conditional-syntax-overrides
  - http://www.yoctoproject.org/docs/latest/ref-manual/ref-manual.html#ref-varlocality
  - http://www.yoctoproject.org/docs/latest/ref-manual/ref-manual.html#ref-variables-glos
  - http://www.embeddedlinux.org.cn/OEManual/recipes_directories.html

Built Toolchains

- Glibc @sanji
  - meta-toolchain @ branch master
    - armv5+gcc-4.9 / armv5+gcc-5.2 ok
    - arm64+gcc-5.2 ok
    - armv7a+gcc-5.2 ok
- Uclibc @yacc
  - meta-toolchain @ branch master
    - armv5+gcc-4.9 / armv5+gcc-5.2 ok
- Musl @usopp
  - Have to add meta-musl layer. git clone https://github.com/kraj/meta-musl
  - meta-toolchain @ master
    - armv5+gcc-4.9 / armv5+gcc-5.2 ok
    - arm64+gcc-5.2 ok
    - armv7a+gcc-5.2 ok

Built Images

- Glibc @sanji
  - core-image-sato @ branch master
    - armv5+gcc-4.9 / armv5+gcc-5.2 ok
    - arm64+gcc-5.2 ok
    - armv7a+gcc-5.2 ok
- Uclibc @yacc
  - core-image-minimal @ branch master
    - armv5+gcc-4.9 / armv5+gcc-5.2 ok
- Musl @usopp
  - Have to add meta-musl layer. git clone https://github.com/kraj/meta-musl
  - core-image-minimal @ master
    - armv5+gcc-5.2 ok
    - arm64+gcc-5.2 ok
    - armv7a+gcc-5.2 ok
- runqemu
  - Both armv5 and arm64 core-image-minimal build with glibc gcc-4.9/gcc-5.2 can boot with qemu
  - uclibc/musl and armv7 core-image-minimal are failed to boot the kernel

Speed up the build

- http://stackoverflow.com/questions/18074979/methods-for-speeding-up-build-time-in-a-project-using-bitbake
- http://www.yoctoproject.org/docs/latest/ref-manual/ref-manual.html#speeding-up-the-build