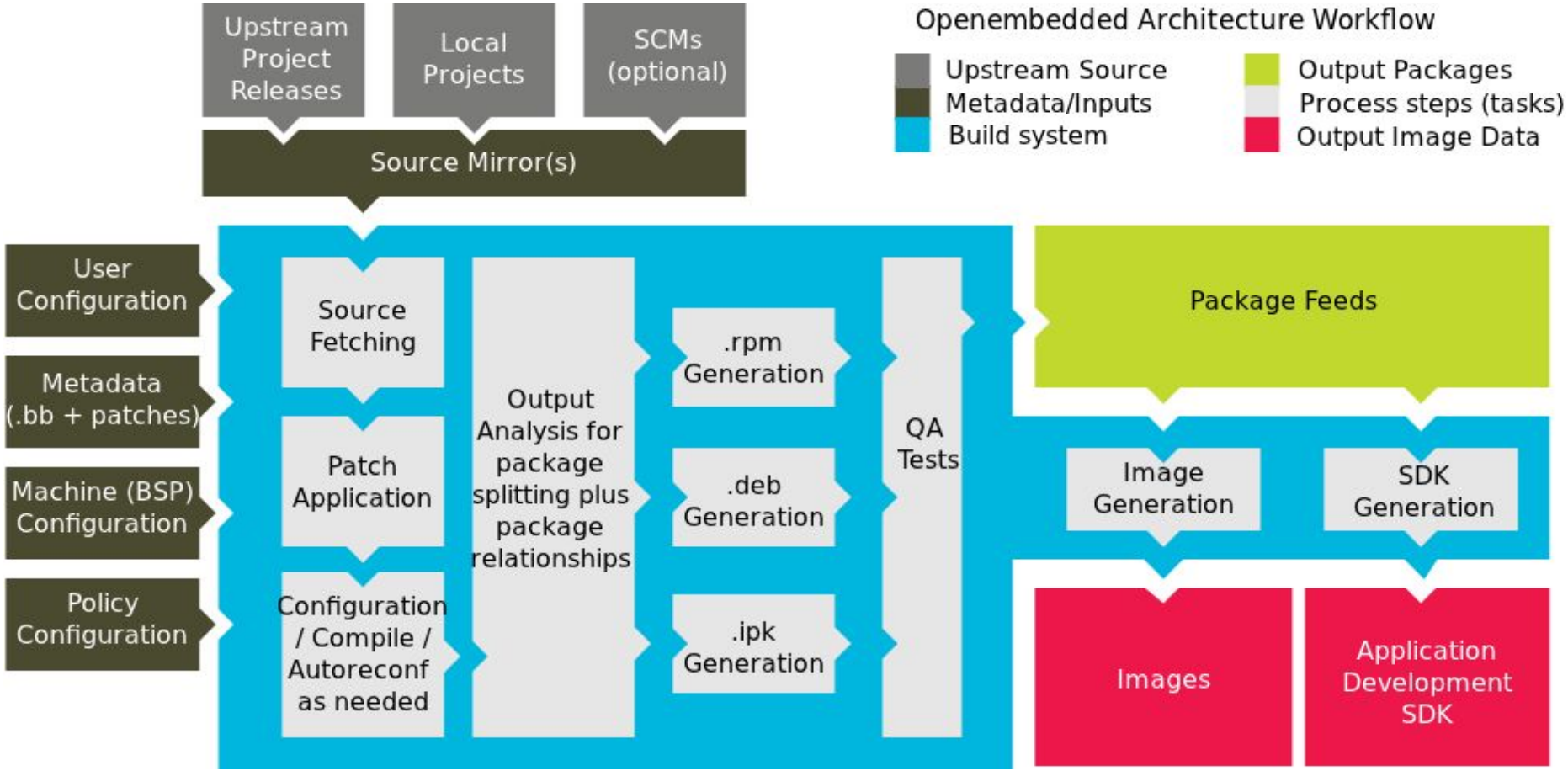
more about the Yocto Project

ESE3005

from the Quick Start guide...

“The Yocto Project through the OpenEmbedded build system provides an open source development environment targeting the ARM, MIPS, PowerPC, and x86 architectures for a variety of platforms including x86-64 and emulated ones. You can use components from the Yocto Project to design, develop, build, debug, simulate, and test the complete software stack using Linux, the X Window System, GTK+ frameworks, and Qt frameworks.”

- X Windows, GTK and Qt are all software libraries for creating graphical user interfaces!
- We will spend some time this semester learning about Qt in-depth


Openembedded Architecture Workflow
Upstream Source
Metadata/Inputs
Build system
Output Packages
Process steps (tasks)
Output Image Data
Upstream Project Releases
Local Projects
SCMs (optional)
Source Mirror(s)
User Configuration
Metadata (.bb + patches)
Machine (BSP) Configuration
Policy Configuration
Source Fetching
Patch Application
Configuration / Compile / Autoreconf as needed
Output Analysis for package splitting plus package relationships
.rpm Generation
.deb Generation
.ipk Generation
QA Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK

Here are some highlights for the Yocto Project:

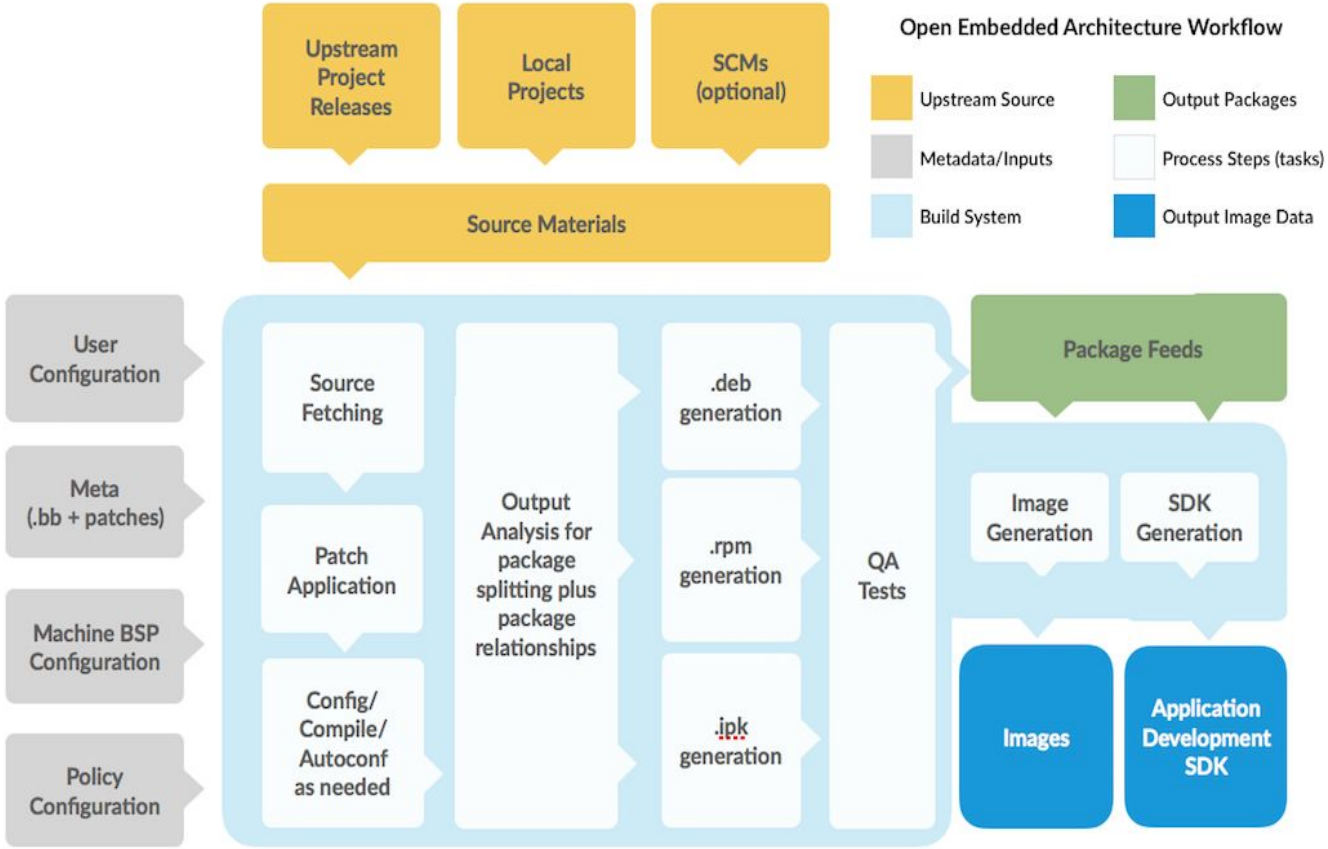
- Provides a recent Linux kernel along with a set of system commands and libraries suitable for the embedded environment.
- Makes available system components such as X11, GTK+, Qt, Clutter, and SDL (among others) so you can create a rich user experience on devices that have display hardware. For devices that do not have a display or where you wish to use alternative UI frameworks, these components need not be installed.
- Creates a focused and stable core compatible with the OpenEmbedded project with which you can easily and reliably build and develop.
- Fully supports a wide range of hardware and device emulation through the Quick EMUlator (QEMU).
- Provides a layer mechanism that allows you to easily extend the system, make customizations, and keep them organized.

You can use the Yocto Project to generate images for many kinds of devices. As mentioned earlier, the Yocto Project supports creation of reference images that you can boot within and emulate using QEMU

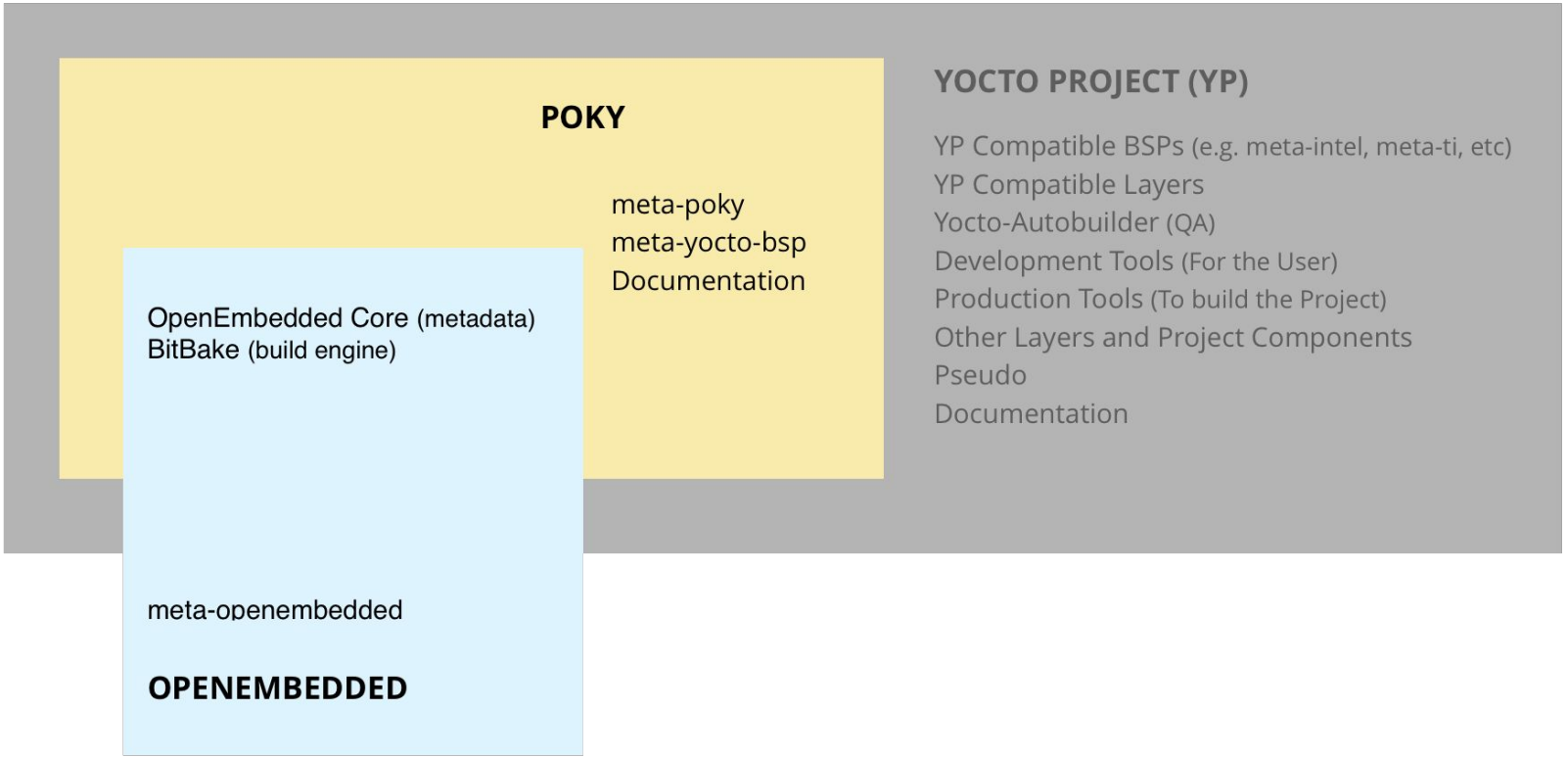
Understanding the Yocto Project workflow is important as it can help you both pinpoint where trouble is occurring and how the build is breaking. The workflow breaks down into the following steps:

1) **Fetch** – get the source code
2) **Extract** – unpack the sources
3) **Patch** – apply patches for bug fixes and new capability
4) **Configure** – set up your environment specifications
5) **Build** – compile and link
6) **Install** – copy files to target directories
7) **Package** – bundle files for installation.

During “fetch”, there may be an inability to find code. During “extract”, there is likely an invalid zip or something similar. In other words, the function of a particular part of the workflow gives you an idea of what might be going wrong.


Open Embedded Architecture Workflow
Upstream Source
Output Packages
Metadata/Inputs
Process Steps (tasks)
Build System
Output Image Data
Upstream Project Releases
Local Projects
SCMs (optional)
Source Materials
User Configuration
Meta (.bb + patches)
Machine BSP Configuration
Policy Configuration
Source Fetching
Patch Application
Config/ Compile/ Autoconf as needed
Output Analysis for package splitting plus package relationships
.deb generation
.rpm generation
.ipk generation
QA Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK

about Poky...


POKY
meta-poky
meta-yocto-bsp
Documentation
OpenEmbedded Core (metadata)
BitBake (build engine)
meta-openembedded
OPENEMBEDDED
YOCTO PROJECT (YP)
YP Compatible BSPs (e.g. meta-intel, meta-ti, etc)
YP Compatible Layers
Yocto-Autobuilder (QA)
Development Tools (For the User)
Production Tools (To build the Project)
Other Layers and Project Components
Pseudo
Documentation

more about Poky

- "Poky" is the name of the Yocto Project®'s reference distribution.
- You can use Poky to bootstrap your own distribution or as an example on how to customize your distribution. (Note that Poky does not contain binary files - it is a working example of how to build your own custom Linux distribution from source, complete with the build environment.)At the core of Poky is the **bitbake** task executor together with various types of configuration files.
    - Bitbake is the tool at the heart of Poky and is responsible for parsing the metadata, generating a list of tasks from it and then executing them
- The metadata or ".bb" files are usually referred to as "recipes". In general, a recipe contains information about a single piece of software such as where to download the source, any patches that are needed, any special configuration options, how to compile the source files and how to package the compiled output.
    - recipes produce packages through the build process; packages are usually .deb or .ipk files, like those you'd use to install software
- Class (.bbclass) files contain information which is useful to share between metadata files. An example is the autotools class which contains the common settings that any application using autotools would use. The classes reference section gives details on common classes and how to use them.
- The configuration (.conf) files define various configuration variables which govern what Poky does. These are split into several areas, such as machine configuration options, distribution configuration options, compiler tuning options, general common configuration and user configuration (local.conf).
- An important component integrated within Poky is Sato, a GNOME Mobile based user interface environment. It is designed to work well with screens at very high DPI and restricted size, such as those often found on smartphones and PDAs. It is coded with focus on efficiency and speed so that it works smoothly on hand-held and other embedded hardware. It will sit neatly on top of any device using the GNOME Mobile stack, providing a well defined user experience.