# Customizing the Yocto-Based Linux Distribution for Production

# Components of a Linux distribution

- Toolchain (gcc)
- Libraries (glibc, etc.)
- Bootloader (grub, u-boot, etc.)
- Kernel
- File system
  - Console utilities
  - Window system
  - Etc.

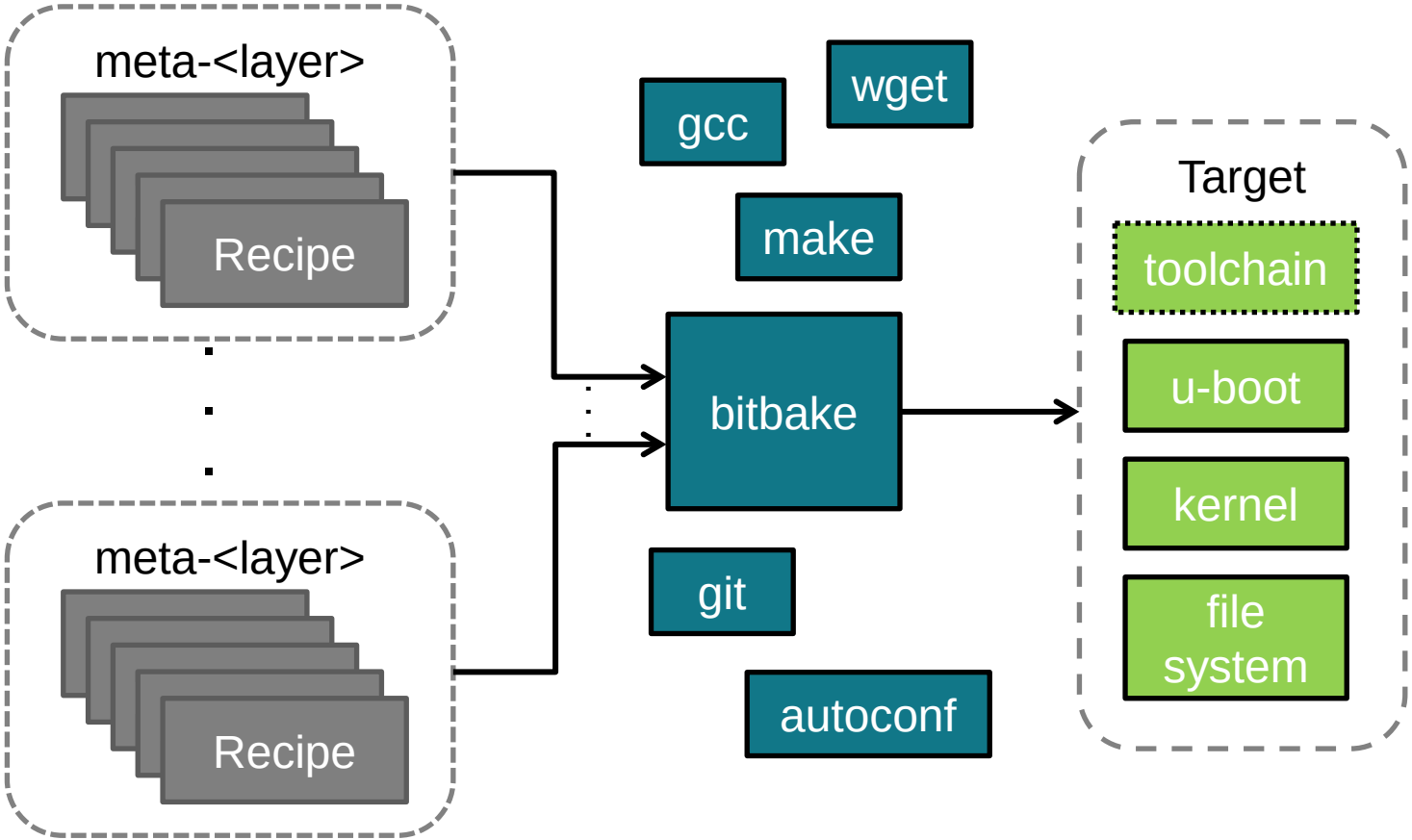
# Why choose Yocto?

- Cross compiling
- Toolchain flexibility (e.g. armv5fp)
- Scalability
- Licensing controls
- Accountability (precise knowledge of what’s in the distribution)

# Yocto Overview

- Yocto Project
- Poky (rhymes with “hockey”)
- Arago
- Recipes
- Layers
- `bitbake`
- Packages
- Package Groups

# Simplified Yocto Build Overview



# What’s in a recipe?

- Source code URL
- Git branch and commit ID
- Build dependencies
- Run-time dependencies
- License
- Configuration commands
- Compilation commands
- Installation commands

# Mapping of Proc SDK Linux to Yocto Branches

| Processor SDK Linux | Yocto Branch |
|---|---|
| 3.x | 2.1 (Krogoth) |
| 4.x | 2.2 (Morty) |
| | 2.3 (Pyro) |
| 5.x | 2.4 (Rocko) |
| | 2.5 (Sumo) |
| | 2.6 (Thud) |

# Building Processor SDK Linux

## Customizing the Yocto-Based Linux Distribution for Production

# Instructions


http://processors.wiki.ti.com/index.php/Processor_SDK_Building_The_SDK

# Build requirements

- Linux host machine
  - 64 bit Linux required
  - Native Linux machine strongly recommended!
- 16GB RAM, up to 200GB for a fully compiled file system
- Ubuntu 16.04 (64-bit) recommended

# Proxy configuration

Pay attention to:

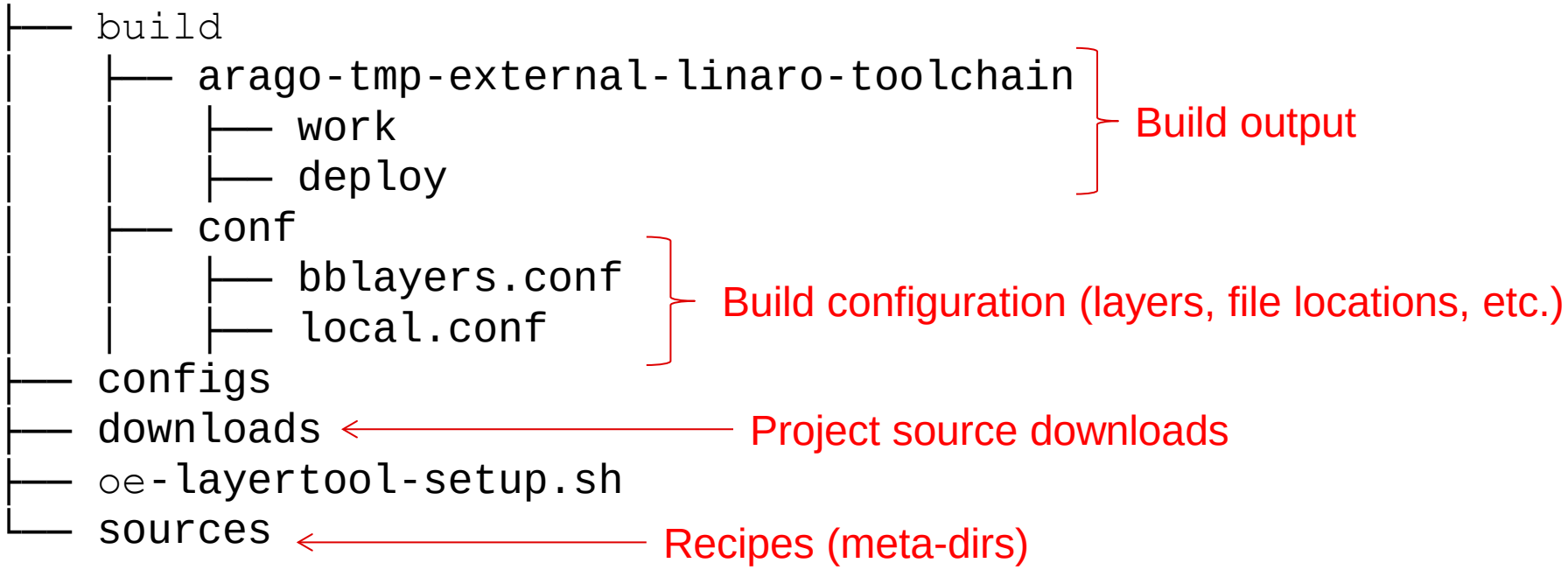
- git
- ssh
- wget
- apt

- Useful tips here:
  - https://wiki.yoctoproject.org/wiki/Working_Behind_a_Network_Proxy
  - http://processors.wiki.ti.com/index.php/Linux_Host_Configuration_-_Ubuntu#Working_with_Proxies

- -k option to speed things up!
  - Continues building packages if errors occur
  - Try re-running bitbake on Fetch failure before jumping into full debug mode (simple connectivity issues are common)
- Today `TI_MIRROR` must be added to `conf/local.conf` (Processor Wiki page)
- `dpkg-reconfigure dash` is required [strange environment errors otherwise]
- Configure threading and processor cores for best performance
  - `BB_NUMBER_THREADS, PARALLEL_MAKE` in `conf/local.conf`
- Currently need nearly 200GB to build Processor Linux SDK from latest snapshot!
- bitbake commands generally require the MACHINE environment variable
- List available targets
  - `MACHINE=am57xx-evm bitbake-layers show-recipes "*-image-*"`
    - assumes all images have "image" in their name
  - Recommend building Yocto-builtin target core-image-minimal as a first step
    - Smaller and has everything necessary to boot an EVM to a Linux prompt

- Debugging build issues (examples?)
- Where is the full output image?
- Where are the intermediate builds?

# Bird’s-eye view



# Deploying images to SD card (or NFS…)

- Built binaries: `build/arago-tmp-external-linaro-toolchain/deploy/images/<platform>`
- Assuming SD card has two partitions: boot (FAT32), rootfs (EXT4)
  - Copy MLO, u-boot.img to boot partition
  - Untar filesystem to rootfs partition
    - `sudo tar xf <filesystem target>.tar.xz –C /media/user/rootfs`

# Create a Custom Layer

**Customizing the Yocto-Based Linux Distribution for Production**

# What is a layer?

- Collection of related recipes
- Typically named meta-*something*
- Typically exists as a git repo (i.e. ownership implied)
- Hierarchical (upper layer modifies lower layer with bbappend)

# Create a new layer

- Create a new layer directory structure in `sources/`

```
sources/
|- meta-custom
|  |- licenses
|  |- recipes
|  layer.conf
|  README
```

- Edit `layer.conf` with boilerplate fields:

```
BBPATH .= ":${LAYERDIR}"
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \
            ${LAYERDIR}/recipes-*/*/*.bbappend"
BBFILE_COLLECTIONS += "meta-custom"
BBFILE_PATTERN_meta-custom = "^${LAYERDIR}/"
BBFILE_PRIORITY_meta-custom = "5"
LAYERVERSION_meta-custom = "1"
```

Note: set BBFILE_PRIORITY to appropriate hierarchy level

# Enable a new layer

- Edit `build/conf/bblayers.conf` to include new layer:

```
BBLAYERS += " \
    …
    <path to sdk>/sources/meta-custom \
    "
```

- Confirm layer is recognized:

```
$ MACHINE=<mach> bitbake-layers show-layers
layer                path                                      priority
…
meta-custom          <path-to-sdk>/sources/meta-custom  16
```

# Why do you want to create a new layer?

- Store recipes for your own software projects
- Consolidate patches/modifications to other people’s recipes
- Create your own images

# Add/Subtract a Package

**Customizing the Yocto-Based Linux Distribution for Production**

# Recipe basics – predefined variables

- my-recipe_1.0.bb


- PN – package name
- PV – package version
- P – package name and version
  - ❑ P = “${PN}-${PV}”

- Example
  - hello-world_1.0.bb
    - PN = “hello-world”
    - PV = “1.0”
    - P = “hello-world-1.0”

# Integrating proprietary elements overview

- Recipe generation tools (recipetool, devtool)
- Required project file structure for BitBake
- Verifying recipe structure
- Adding element to filesystem image

# Recipe generation tools

- Automates the process of creating a recipe

- Generate a starting point for a recipe
  - Specify Git URL (SRC_URI)

# Restrict package to specific machine(s)

- By default, package will be built for all machines
  - Package found in `build/$TOOLCHAIN/deploy/ipk/all`

# Append package to an image(s)

- Packages added to the image in layer.conf
  - E.g.: `sources/meta-custom/conf/layer.conf`
  - Include package in all images: `IMAGE_INSTALL += " hello-world"`
  - Selectively include package:

# Modifying an Existing Recipe

**Customizing the Yocto-Based Linux Distribution for Production**

# Common Reasons for Recipe Modification

- Update build options
- Change installation directory
- Source code patches

# .bbappend

- Example recipe
  - meta-somelayer/recipes-example/do-something_x.y.bb

- Appended version
  - meta-custom/recipes-example/do-something_x.y.bbappend

# Source audit and licensing

**Customizing the Yocto-Based Linux Distribution for Production**

# Yocto/ OpenEmbedded licensing overview

- Yocto/ OE licensing features and framework
- Audit and enforcement of usable licenses
- Workarounds for packages with incompatible licenses
- Software manifest

# Yocto/ OE licensing features and framework

- License file tracking – ensure changes to project license do not go unnoticed
- All packages must specify the following in their recipe:
  - License (e.g.: MIT, GPLv2, GPLv3, etc)
  - License file
  - License file checksum
- Restrict package licenses incompatible with business model
- Whitelist packages with restricted licenses

https://www.yoctoproject.org/docs/2.4.1/ref-manual/ref-manual.html#licenses

# Where to configure licensing

- Each package bitbake recipe requires the following license variable definitions:
  - `LICENSE`
  - `LIC_FILES_CHKSUM`
  - recipetool/ devtool will populate these fields automatically
- Global license configuration is handled in `build/conf/local.conf`
  - Restrictions, whitelisting

# Restricting licenses

- If GPLv3 is not compatible with a business model, consider building a filesystem “bottoms-up” (i.e. adding to a minimal image) rather than a “tops-down” approach (i.e. removing packages as dependency errors come up)
- License restrictions are applied on a global scale
  - In `build/conf/local.conf: INCOMPATIBLE_LICENSE = "GPLv3"`
- Packages with incompatible license dependencies will be broken
  - Finding `packagegroup` definitions can be time consuming at first. To speed things up, rename the recipe’s folder. Bitbake will output an error describing which `packagegroup` has unmet `RDEPENDS`. With the package and `packagegroup` names, removing the dependency looks like this:

In `meta-custom/layer.conf:`

```
RDEPENDS_packagegroup-arago-base-tisdk_remove = “ cifs-utils"
RDEPENDS_packagegroup-arago-test_remove = " perf"
```

# Final notes on removing GPLv3 components

- Becoming much more difficult to build an up-to-date, fully functional distribution/ filesystem without GPLv3 components in the Linux world
- Removing GPLv3 will may mean taking tradeoffs in features, or require investment .
- `meta-gplv2` contains packages with older GPLv2 versions that are either unmaintained or do not enjoy frequent updates due to limited community resources and **may contain unpatched security vulnerabilities**.
- Ultimately it is a business decision, however it bears repeating eliminating GPLv3 is not an easy task

# Whitelisting components

- Whitelist components with restricted license
  - In `build/conf/local.conf`:

```
LICENSE_FLAGS_WHITELIST = "commercial"
```

# Software manifest files

- Yocto generates a license manifest for all image packages
  - Major feature for embedded Linux products!
- Found here when the image completes:

**build/$TOOLCHAIN/deploy/licenses/$TARGET-$MACHINE-$TIMESTAMP/license.manifest**

```
PACKAGE NAME: alsa-conf
PACKAGE VERSION: 1.1.2
RECIPE NAME: alsa-lib
LICENSE: LGPLv2.1 & GPLv2+
```

Example variables:
$TOOLCHAIN: arago-tmp-external-linaro-toolchain
$TARGET: arago-base-tisdk-image
$MACHINE: am335x-evm

# For more information

- For questions regarding topics covered in this training, visit the support forums at the TI E2E Community website.