Sofiatech

Back To Basics

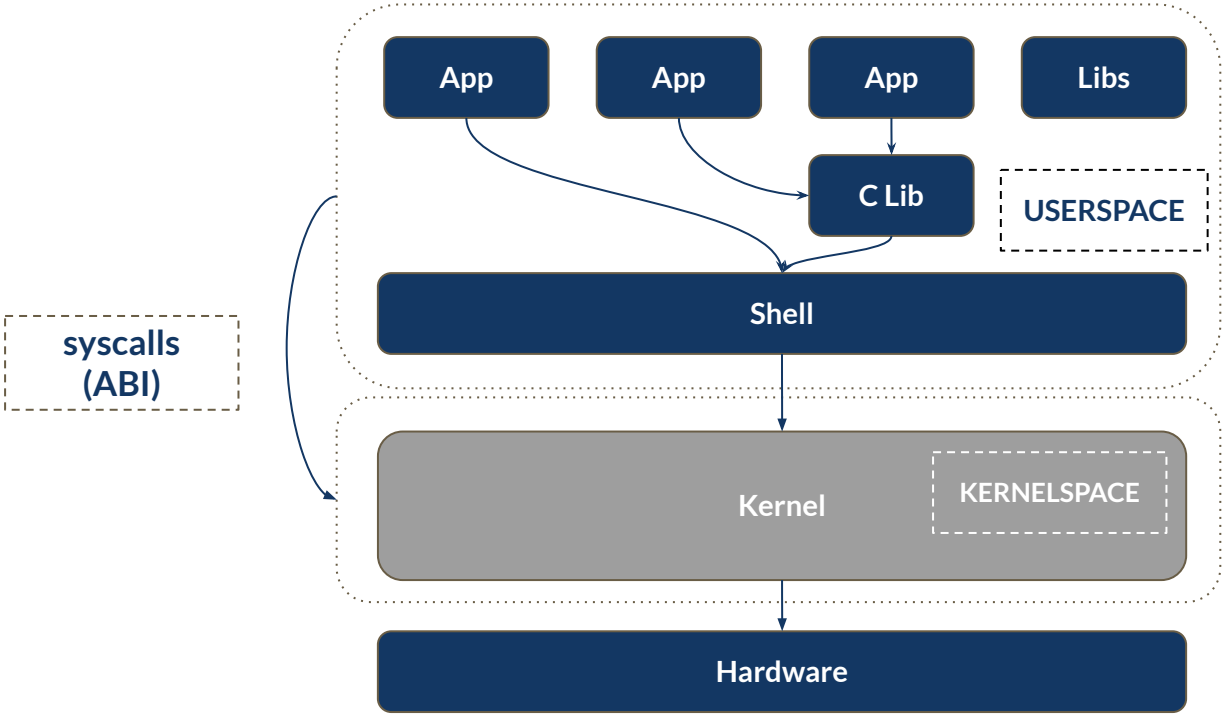
The ultimate Yocto introduction

```
summit.23:~$ bitbake -e | grep ^WHOAMI

WHOAMI=”Talel BELHAJSALEM”

WHOAMI_DESC=”Yocto Enthusiast and Trainer”

WHOAMI_WORK=”Embedded Linux Engineer at Sofia Technologies”

WHOAMI_LINKS=”linkedin email stackoverflow extra”

WHOAMI_LINKS[linkedin] = “https://www.linkedin.com/in/bhstalel/”

WHOAMI_LINKS[stackoverflow] = “https://stackoverflow.com/users/7553704/talel-belhadjsalem”

WHOAMI_LINKS[email] = “talel.hajsalem@sofia-technologies.com”

WHOAMI_LINKS[extra] = “https://yoctoleef.xyz/”
```

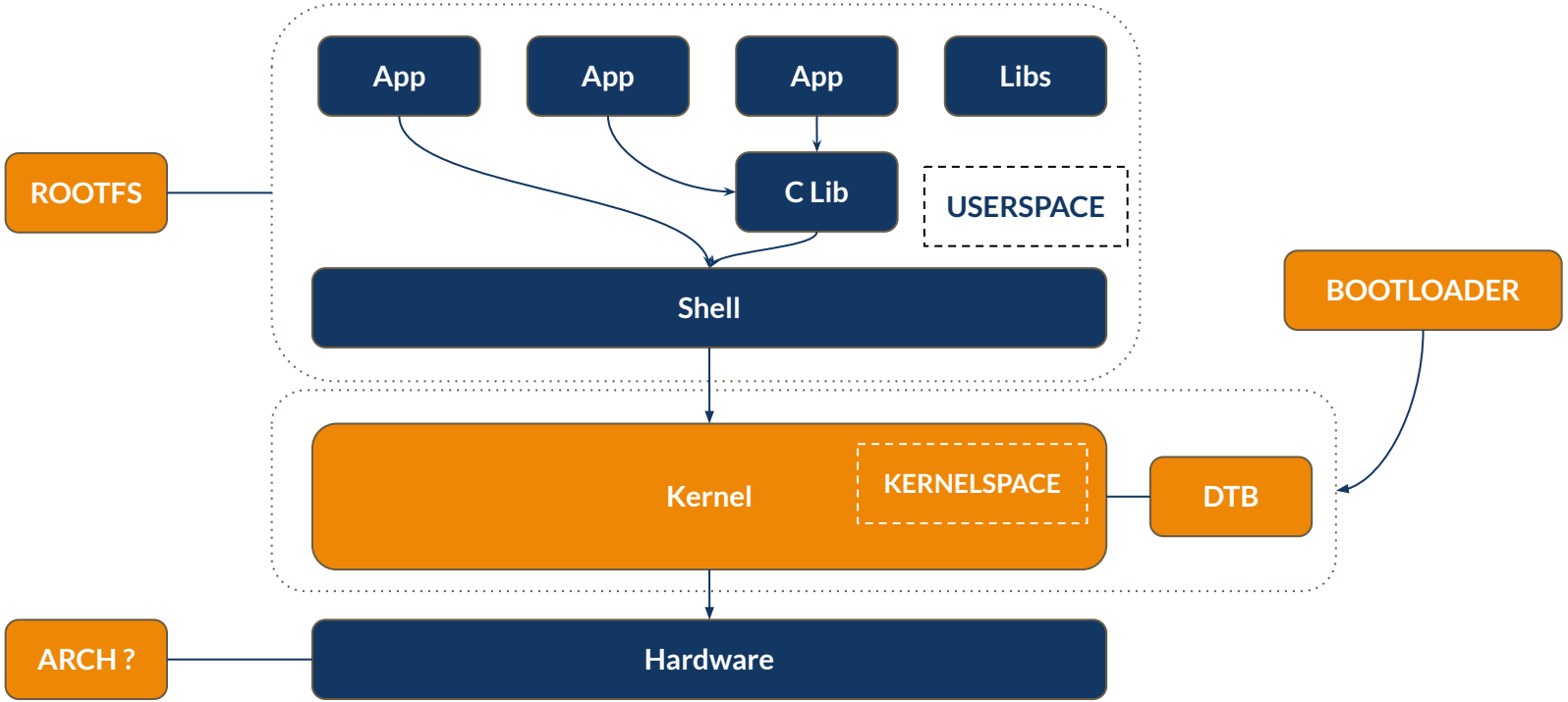
Summary

1. Linux Architecture
2. The Global Aim
3. Build System Philosophy
4. The Chef : Bitbake
5. The Story behind Metadata
6. Preparing The Kitchen : oe-init-build-env
7. The Cooking Process
8. Plating The Image
9. Useful Metadata
10. Board Support Package
11. Useful Cooking Techniques
12. What’s Next?

Linux Architecture : Basics

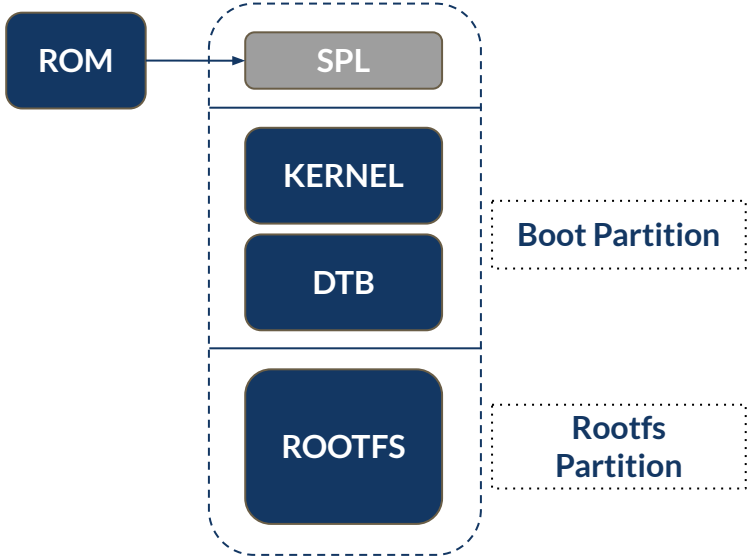


Linux Architecture : What we need?



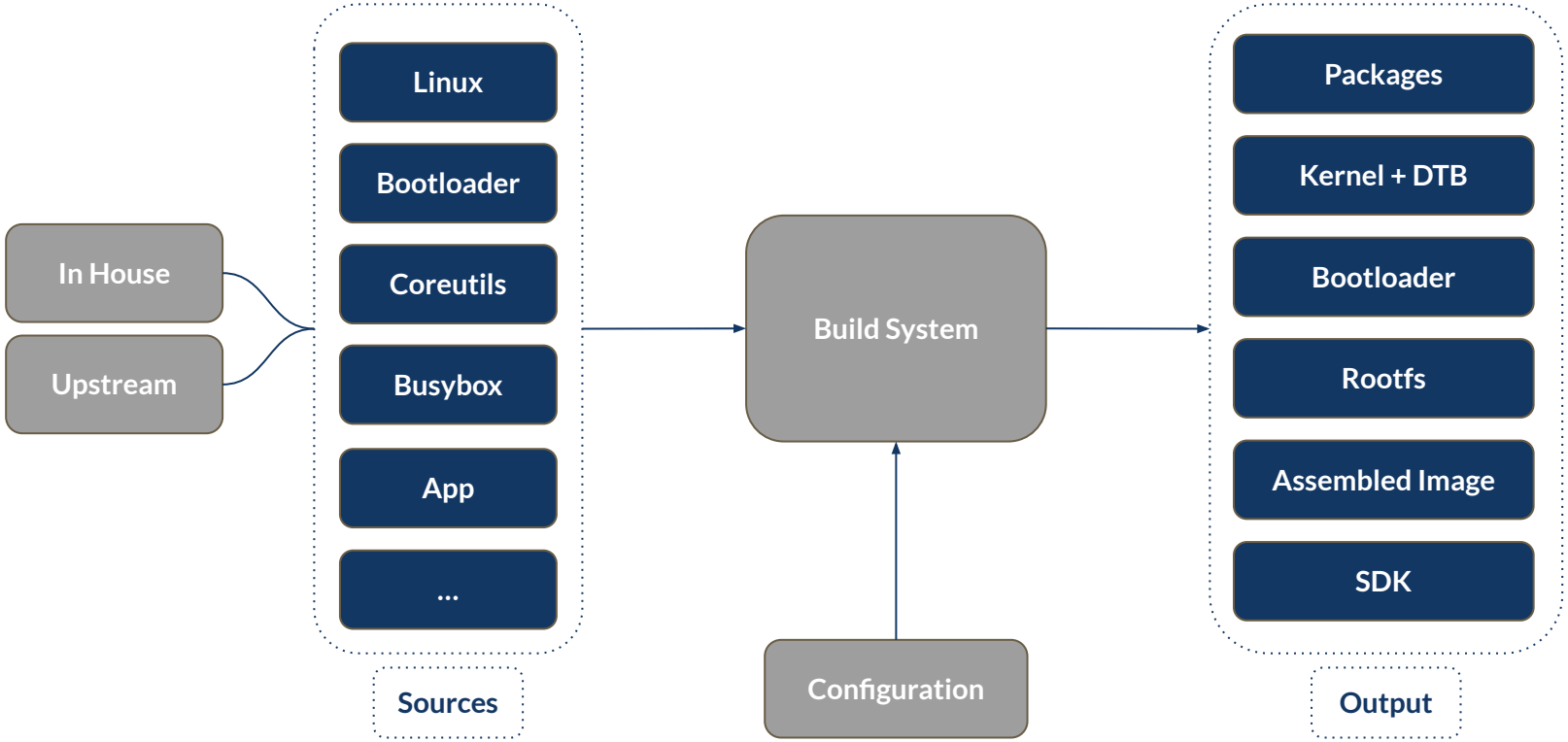
The Global Aim : Back to boot

- We need a way to:
  - Build **Kernel** and **DTB**
  - Assemble **Rootfs**
    - Handle packaging
    - Handle dependencies
  - Build **SPL** *(Vendor Specific)*
  - Assemble the final **image**
- We need a way to:
  - Build all of that for the target **ARCH**
  - Figure out the compilation type
  - Prepare an **SDK** (**Toolchain**)



The typical image structure (SD, eMMC, ...)

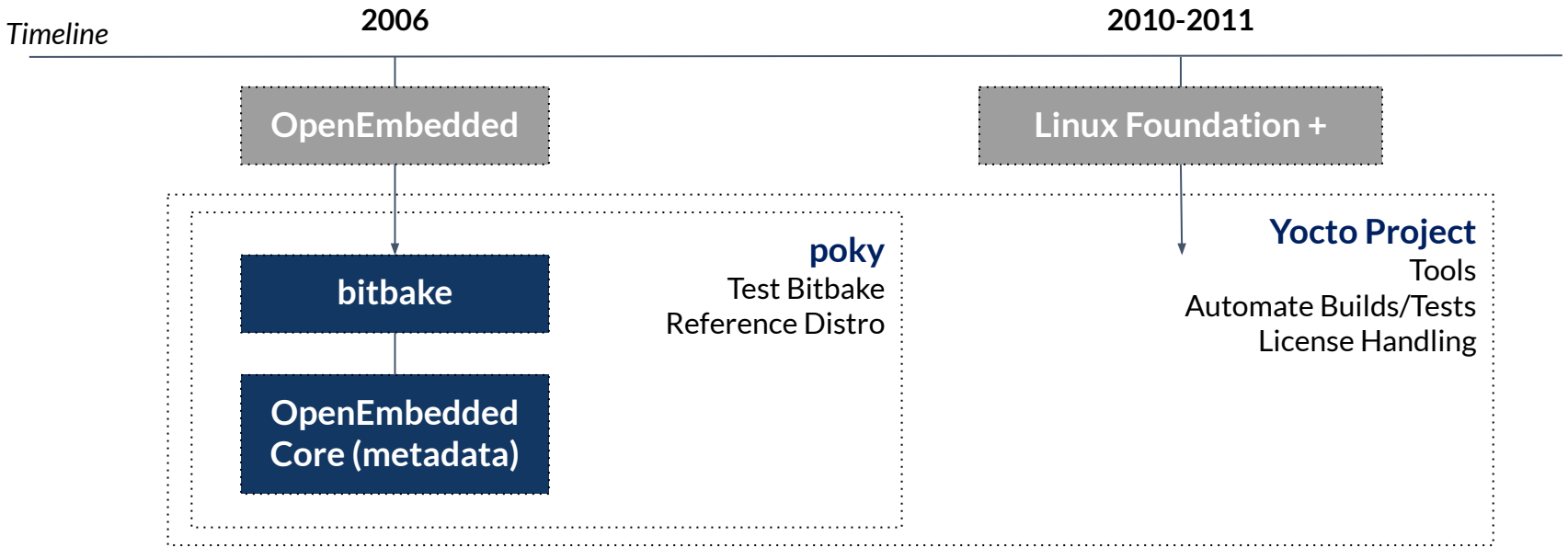
Build System Philosophy



Build System Philosophy : Configuration

- The build system needs some information in order to produce the output:
  - What **Linux Kernel** to use ?
  - What **Bootloader** to use ?
  - What **Device Tree Source** to compile from the Kernel dts sources ?
  - What is the Target **architecture** ?
    - This helps creating the right **toolchain** for the compilation process
  - What is the final **image file type** and structure ?
  - What **C Library** to use ?
  - What are **packages** to include in the final rootfs ?
  - What to use for the basic **utilities** (*busybox*, *coreutils*, ...) ?
  - What **init manager** to use (*systemd*, *sysvinit*, ...) ?
  - Other questions need to be answered

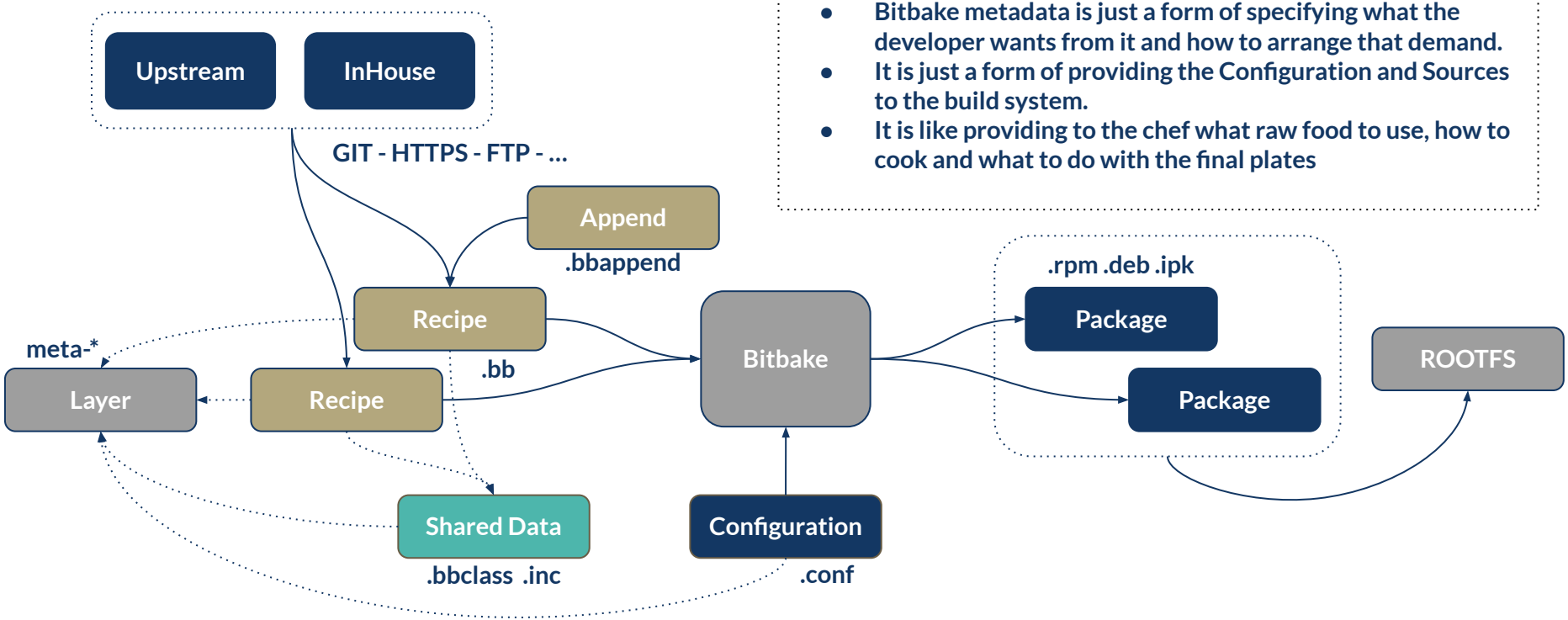
The Chef : BitBake



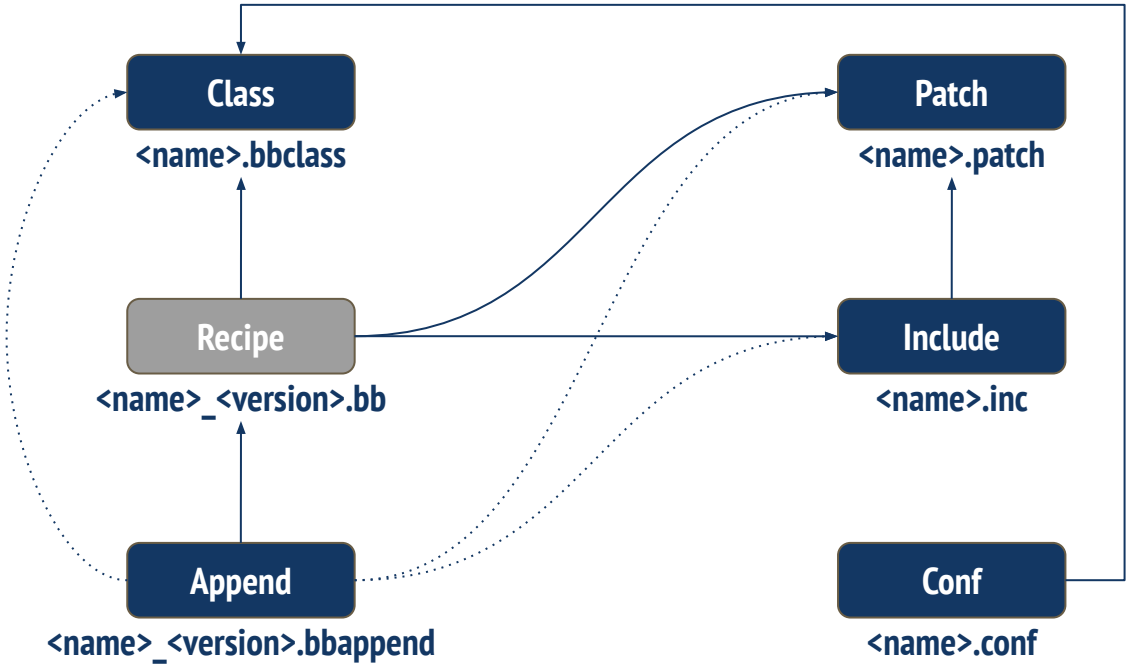
- The Build engine (Python)
- Called: The ***make*** of distributions

The Story Behind Metadata

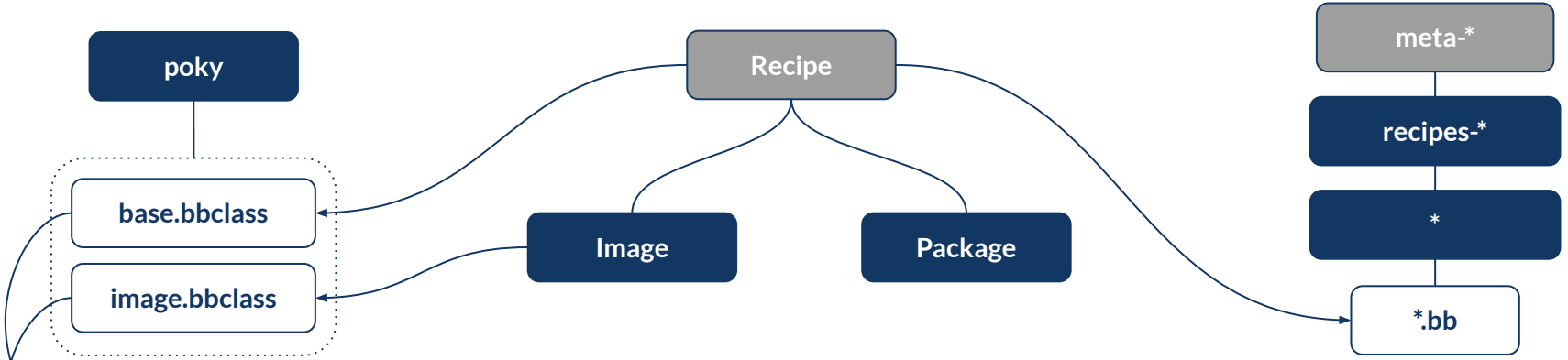
- Bitbake metadata is just a form of specifying what the developer wants from it and how to arrange that demand.
- It is just a form of providing the Configuration and Sources to the build system.
- It is like providing to the chef what raw food to use, how to cook and what to do with the final plates



The Story Behind Metadata : Interconnection



The Story Behind Metadata : Recipe



Contains all tasks needed to create a **Package** from fetching source to actual package creation.

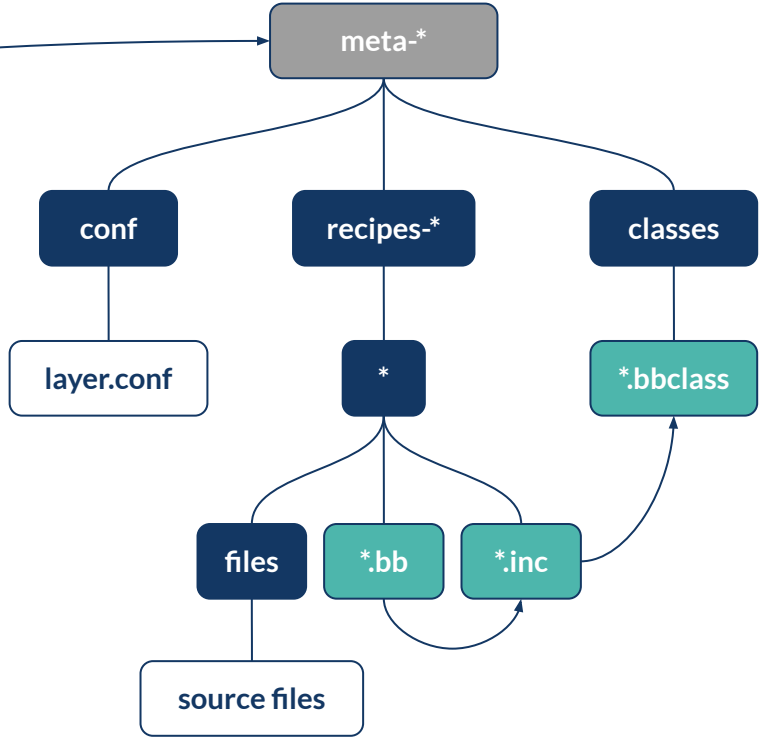
Disables all package tasks and create new tasks to create and assembly the final **Rootfs** using list of package recipes.

A package recipe defines (*Similar to a real world chef's recipe*):

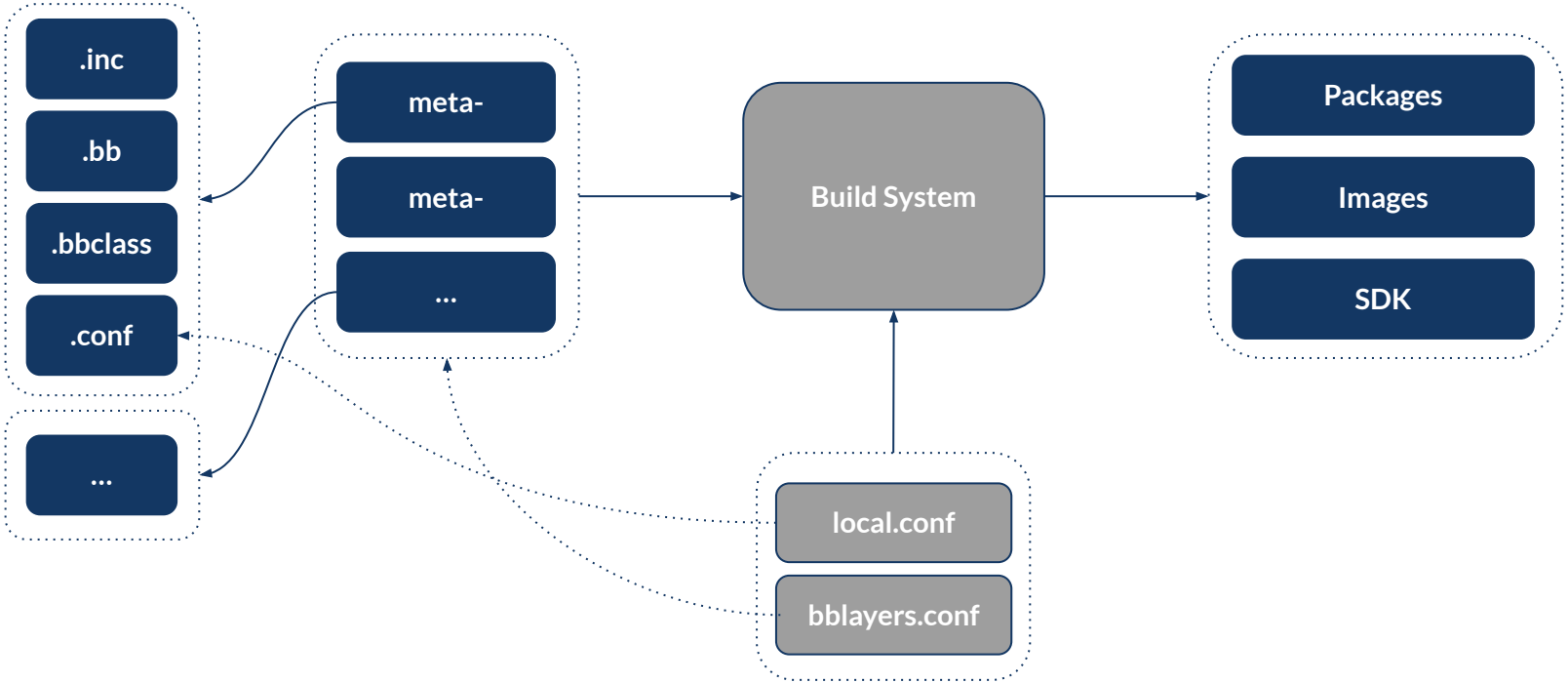
- ❏ What sources to work with: **SRC_URI**
- ❏ How to deal with them: **do_configure**, **do_compile**, …
- ❏ What to go into the final package: **do_install**, **FILES**, …

The Story Behind Metadata : Layer

```
# 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 += "skeleton"
BBFILE_PATTERN_skeleton = "^${LAYERDIR}/"

BBFILE_PRIORITY_skeleton = "1"

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

LAYERDEPENDS_skeleton = "core"
LAYERSERIES_COMPAT_skeleton = "kirkstone"
```



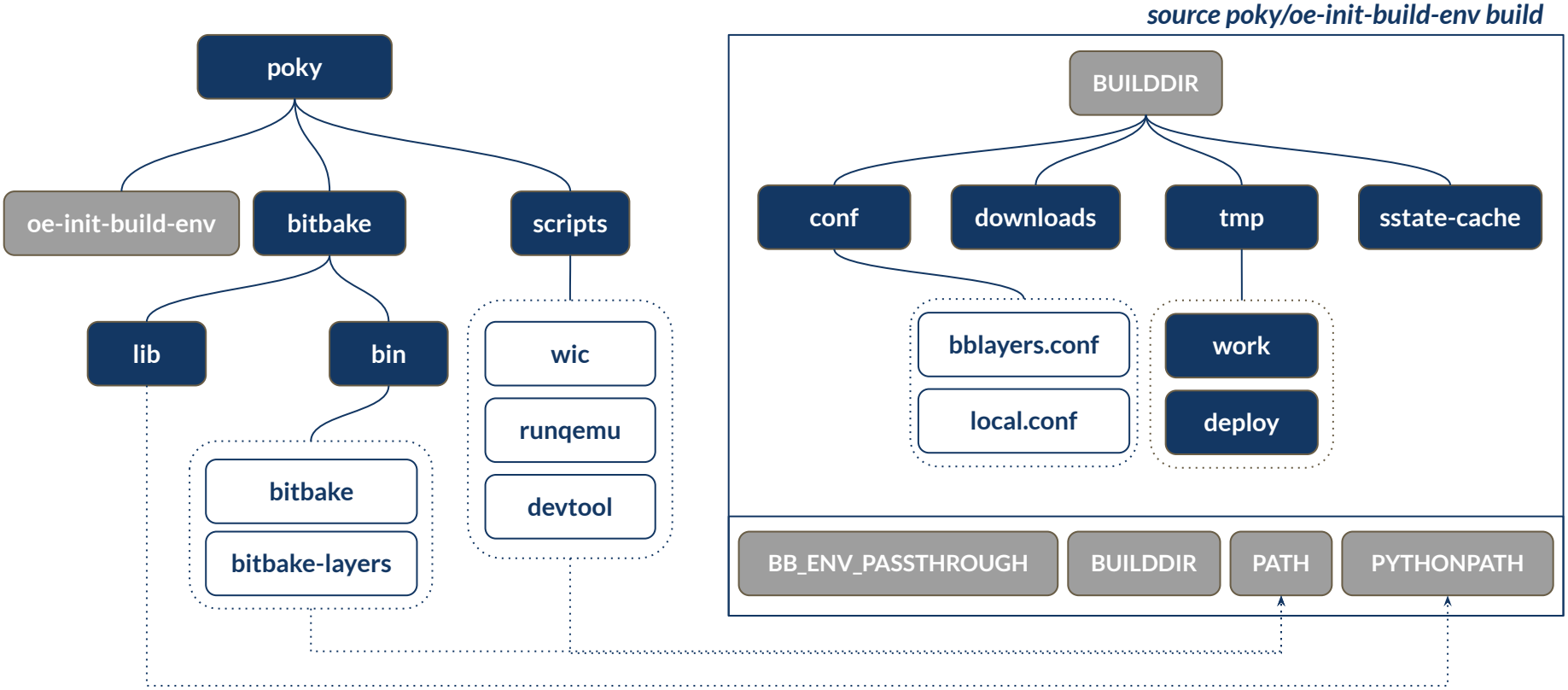
The Story Behind Metadata : The big picture



The chef (*BitBake*) needs to know:

Where to work (*The kitchen*) ?
How and where to find what to work with ?

Preparing The Kitchen : oe-init-build-env

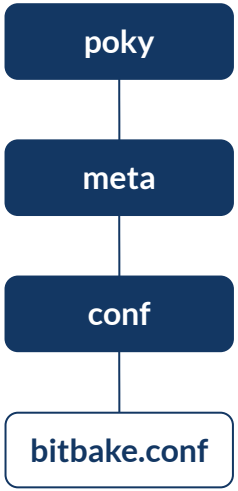


- Source the env in every new opened terminal
- Sourcing in new folder will create a new kitchen

The Cooking Process : Entry

- Introduce the chef to the kitchen (***source oe-init-build-env***)
- The chef analyses the environment and collects final information about:
  - What layers to consider: **BBFILE_COLLECTIONS**
  - Where recipes are: **BBFILES**
  - The rest of variables
- All configuration is set of variables used by the chef to know where to find things:
  - **MACHINE**
  - **DISTRO**
  - ...

The Cooking Process : Global configuration

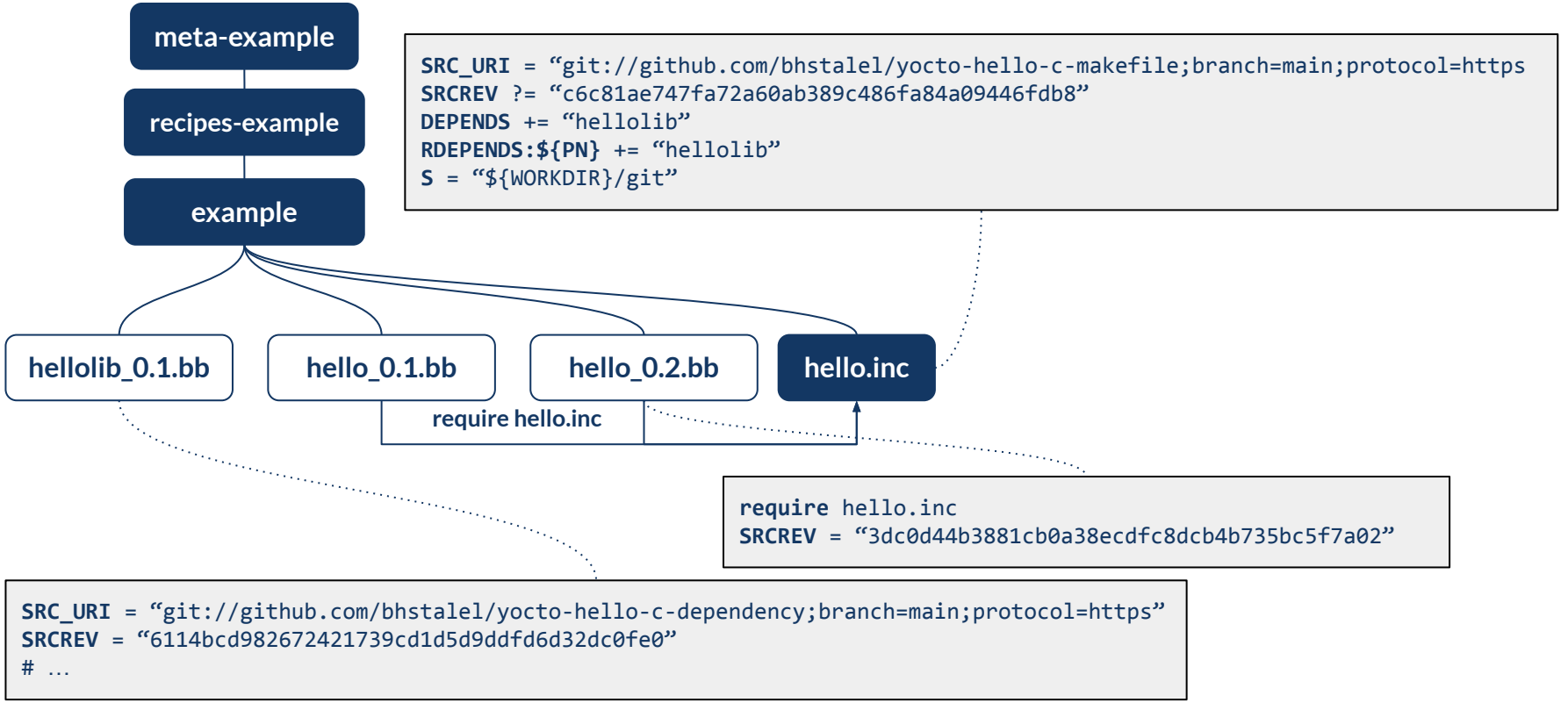



```
BUILD_ARCH := "${@os.uname()[4]}"
HOST_ARCH = "${TARGET_ARCH}"
TARGET_ARCH = "${TUNE_ARCH}"
```

```
TCLIBC ??= “glibc”
export BUILD_CC = "${CCACHE}${BUILD_PREFIX}gcc ${BUILD_CC_ARCH}"
export BUILD_CXX = "${CCACHE}${BUILD_PREFIX}g++ ${BUILD_CC_ARCH}"
export BUILD_CPP = "${BUILD_PREFIX}gcc ${BUILD_CC_ARCH} -E"
export BUILD_LD = "${BUILD_PREFIX}ld ${BUILD_LD_ARCH}"
export BUILD_AS = "${BUILD_PREFIX}as ${BUILD_AS_ARCH}"
export MAKE = "make"
```

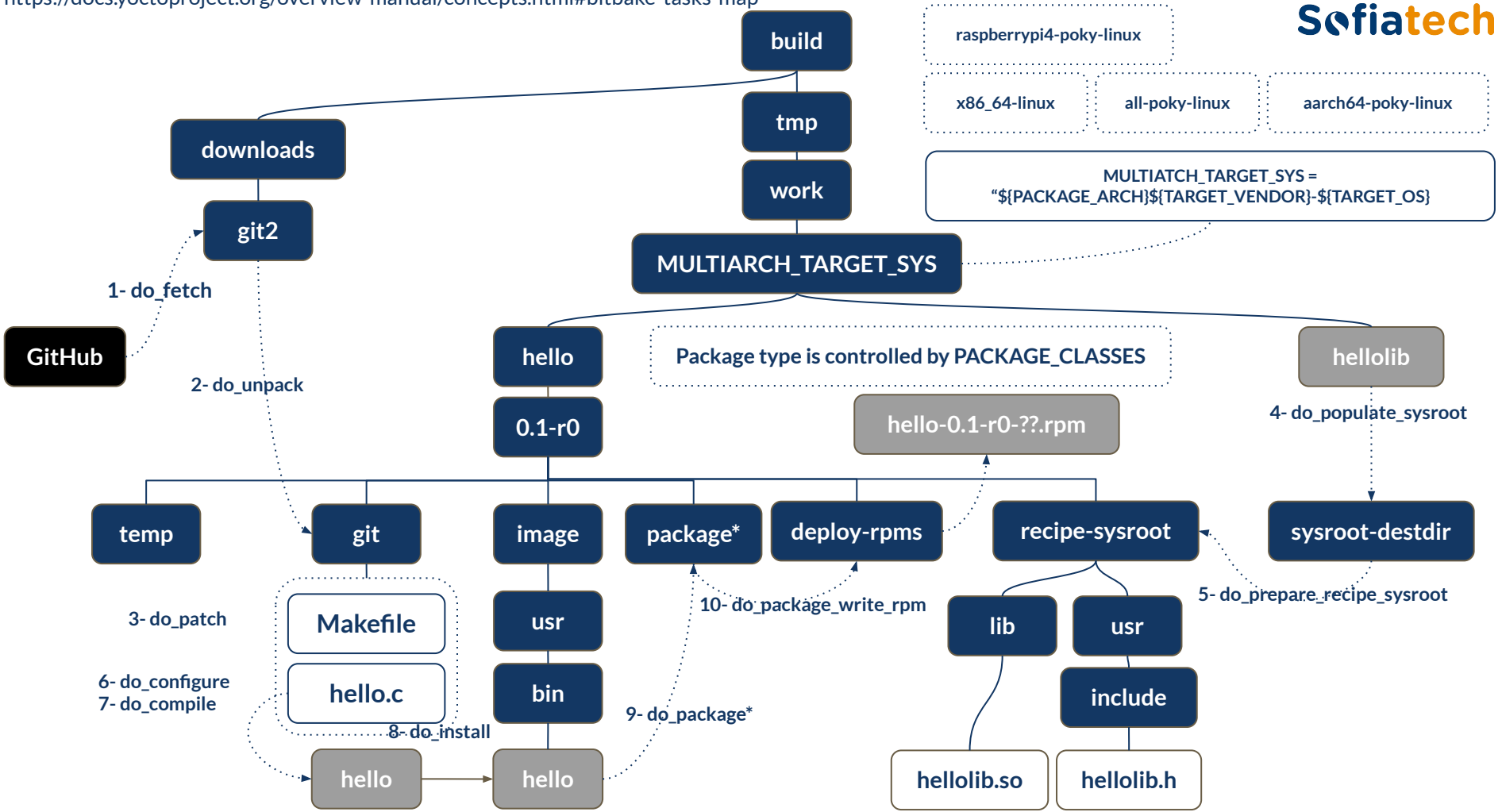
- This is the default configuration with lot of other variables as well.
- This is updated according to type of compilation
  - *Example*: for cross compilation: **HOST_ARCH** will be same as **BUILD_ARCH**
- This is automatically handled, you can choose not to bother digging into the configuration.

The Cooking Process : Recipe



https://docs.yoctoproject.org/overview-manual/concepts.html#bitbake-tasks-map

Sofiatech



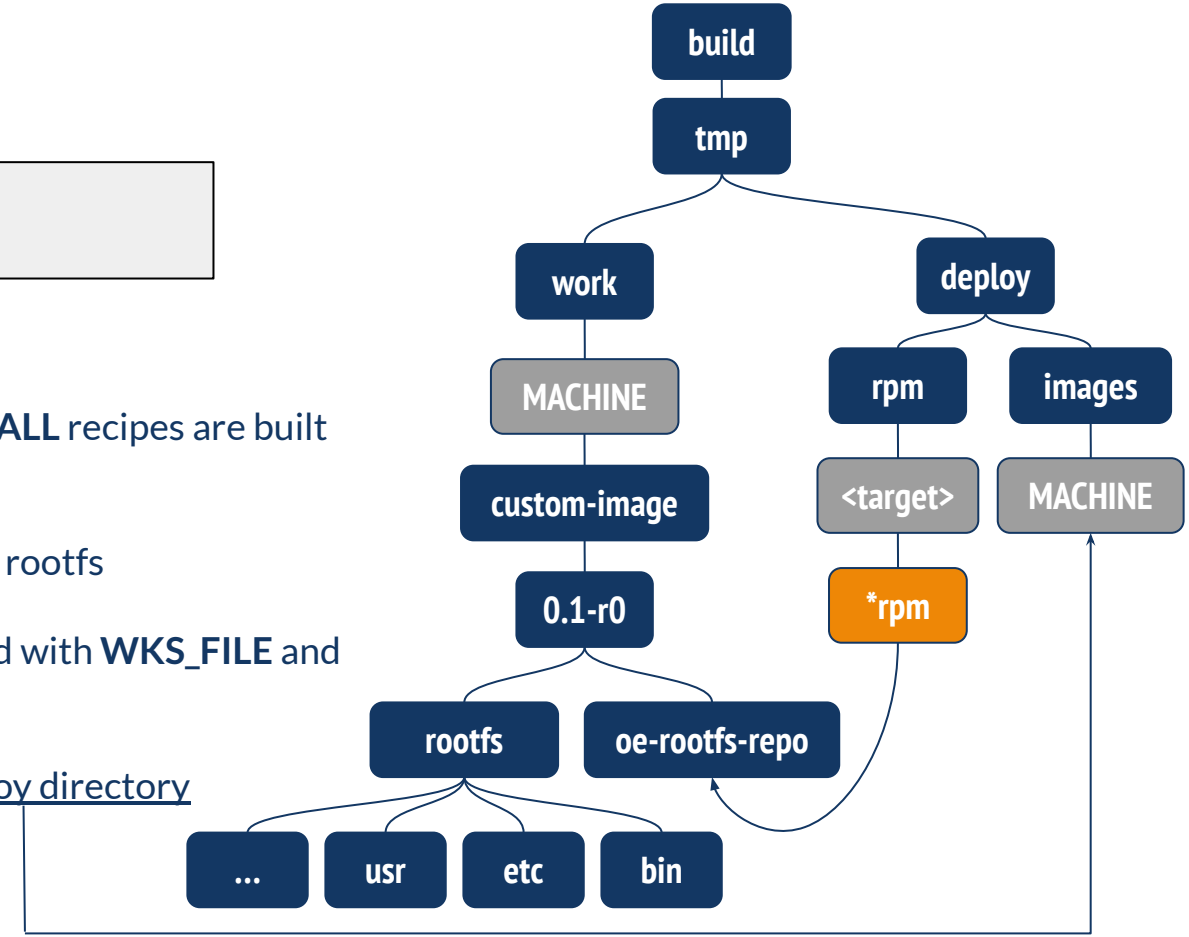
Plating The Image

custom-image.bb

```
inherit core-image
IMAGE_INSTALL:append = “ hello”
```

BitBake will:

- Make sure that all **IMAGE_INSTALL** recipes are built correctly
- Install all packages into the final rootfs
- Create image structure specified with **WKS_FILE** and other variables
- Place the final image in the deploy directory



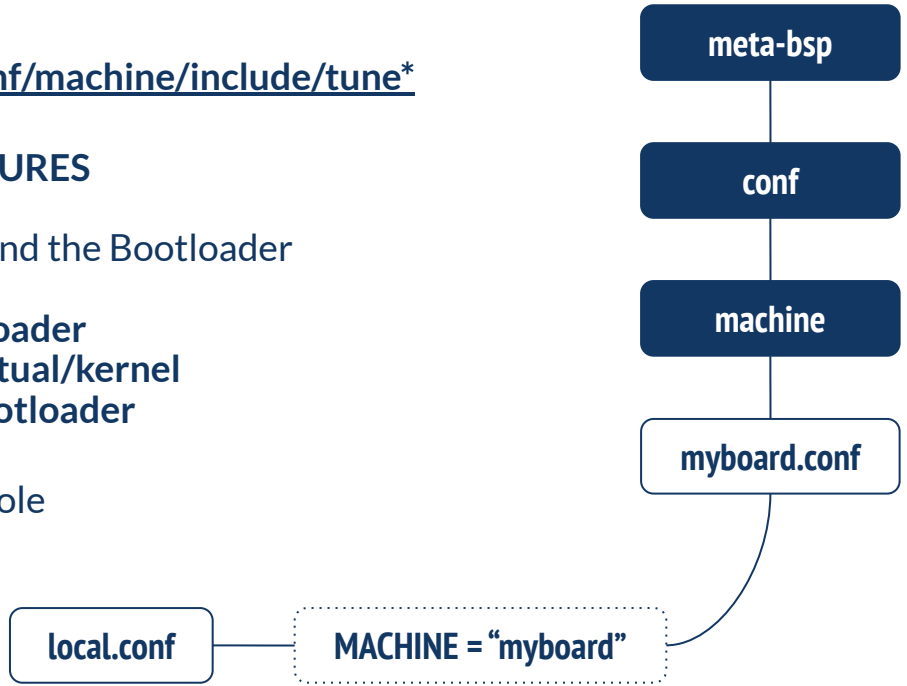
Useful Metadata

- Poky, the core OE layer and the community provides very useful metadata:
  - https://git.yoctoproject.org/

Layers		Classes
Board Support Package	Software	useradd.bbclass (poky)
meta-raspberrypi	meta-openembedded	systemd.bbclass (poky)
meta-intel	meta-browser	cmake.bbclass (poky)
meta-imx	meta-virtualization	qt6-qmake.bbclass (meta-qt6)
meta-st-stm32mp	meta-aws	packagegroup.bbclass (poky)
meta-ti	meta-qt6	module.bbclass (poky)
meta-xilinx	meta-java	setuptools3.bbclass (poky)

Board Support Package : Machine configuration

- Machine configuration contains:
  - Target architecture: **poky/meta/conf/machine/include/tune***
  - Machine features **MACHINE_FEATURES**
  - Variables to customize the Kernel and the Bootloader
  - The choice of the **Kernel** and **Bootloader**
    - **PREFERRED_PROVIDER_virtual/kernel**
    - **PREFERRED_PROVIDER_bootloader**
  - The configuration of the serial console
    - **SERIAL_CONSOLE**



Board Support Package : Machine configuration

- Kernel and Bootloader configurations:
  - Kernel DTB: **KERNEL_DEVICETREE**
  - Kernel Image name: **KERNEL_IMAGETYPE**
  - Uboot machine: **UBOOT_MACHINE**
  - Boot files: **IMAGE_BOOT_FILES**
  - Rootfs Image type: **IMAGE_FSTYPES** , **WKS_FILE**

- These variables are used by **virtual/kernel** and **virtual/bootloader** virtual recipes to do their internal compilation.

Board Support Package : raspberrypi4-64

```
IMAGE_FSTYPES = “tar.bz2 ext3 wic.bz2”
WKS_FILE ?= “sdimage-raspberrypi.wks”
MACHINE_FEATURES += “usbhost wifi ..”
KERNEL_IMAGETYPE_UBOOT ?= “Image”
KERNEL_IMAGETYPE_DIRECT ?= “Image”
UBOOT_MACHINE = “rpi_4_config”
SDIMG_KERNELIMAGE ?= “kernel8.img”
RPI_KERNEL_DEVICETREE = “broadcom/bcm2711-rpi-4-b.dtb”
PREFERRED_PROVIDER_virtual/kernel ?= “linux-raspberrypi”
IMAGE_CLASSES += “sdcard_image-rpi”
require conf/machine/include/arm/armv8a/tune-cortexa72.inc
```

This answers all the questions we asked in the beginning.

Now BitBake knows:

- What architecture to use, so CROSS
- What kernel to use
- What DTB to pass to Kernel
- What bootloader to use
- What Uboot machine to use
- What files to put in boot partition
- What image file structure to build

Advanced Cooking Techniques

Debugging and Useful techniques		Comment
bitbake -e	Expand the full environment	`bitbake -e <recipe> | grep ^VARIABLE`
bitbake -s	Show all recipes versions	Useful to check available versions and test layers priorities on them
devshell	Prepare dev shell in WORKDIR	Useful when you want to work manually on the source code after do_unpack: `bitbake <recipe> -c devshell`
pydevshell	Python dev interpreter	Same as devshell (more powerful) you have access to the *d* object: `d.getVar() d.getVarFlag()` …
Log Functions	Used inside tasks to log stuff	`bbwarn, bbinfo, …` `bb.warn, bb.info, …`
Anonymous Functions	Used to prepare some information or do some checks at parse-time.	`python __anonymous(){}`
Inline Python Variables	Used to do conditional assignment	`IMAGE_INSTALL:append = " ${@'m.service' if d.getVar('INIT_MANAGER') == 'systemd' else ''}"`

Useful Cooking Tools

Tool	Comment
bitbake-layers	Manage layers (create, add, remove, ...)
devtool	The most powerful utility after bitbake, it is used to add and modify recipes automatically. **Example:** `devtool add hello https://github.com/bhstalel/yocto-hello-c-makefile -B main`
recipetool	Used also to add and/or modify recipes and create bbappend files automatically
oe-pkgdata-util	Used to check information about packages, example: find which recipe provides a file

What Is Next?

- **PACKAGECONFIG**
- Create and test various **DISTRO**s
- Sharing **downloads** and **cache** and optimization
- Work with **Package Management**
- Work with auto wrappers like **repo** and **Kas**
- Populate and work with **SDK**
- Check Documentation: https://docs.yoctoproject.org/
- Welcome to IRC: https://web.libera.chat/?channels=#yocto
- Welcome to Mailing Lists: https://lists.yoctoproject.org/