Android build system

Nguyen Tran (Nguyen.TranLeHoang@vn.bosch.com)

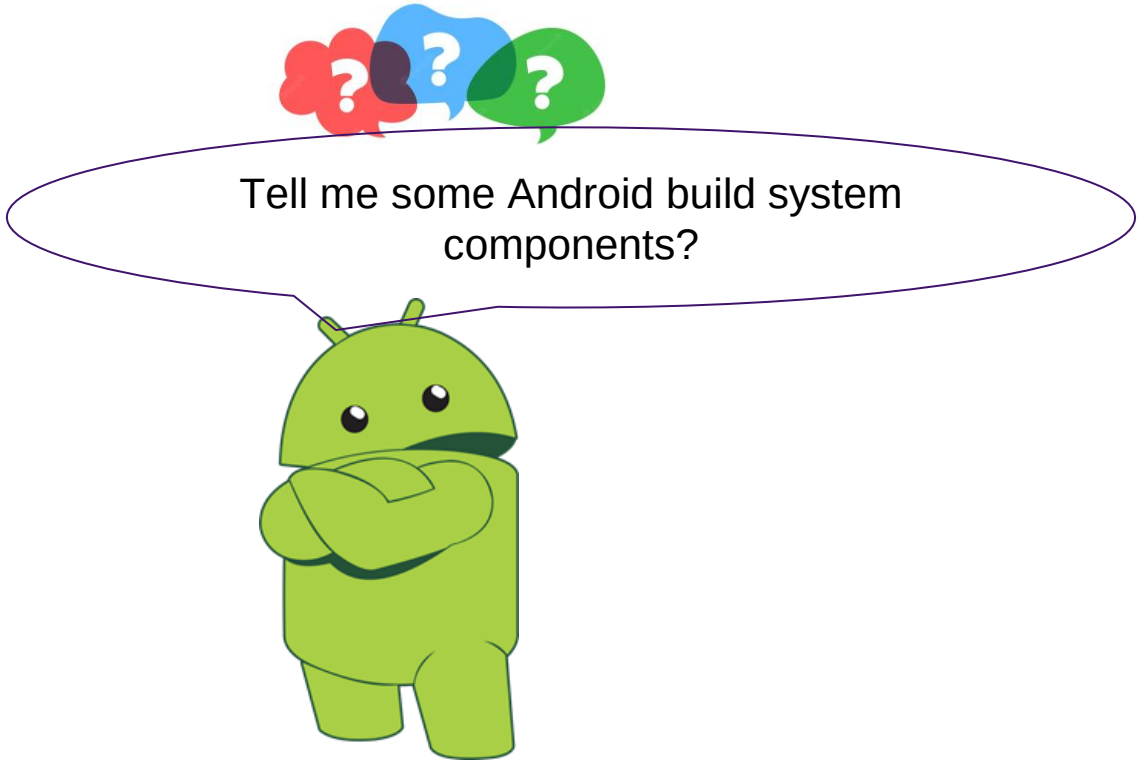
Oct-2024

BOSCH

Day 4

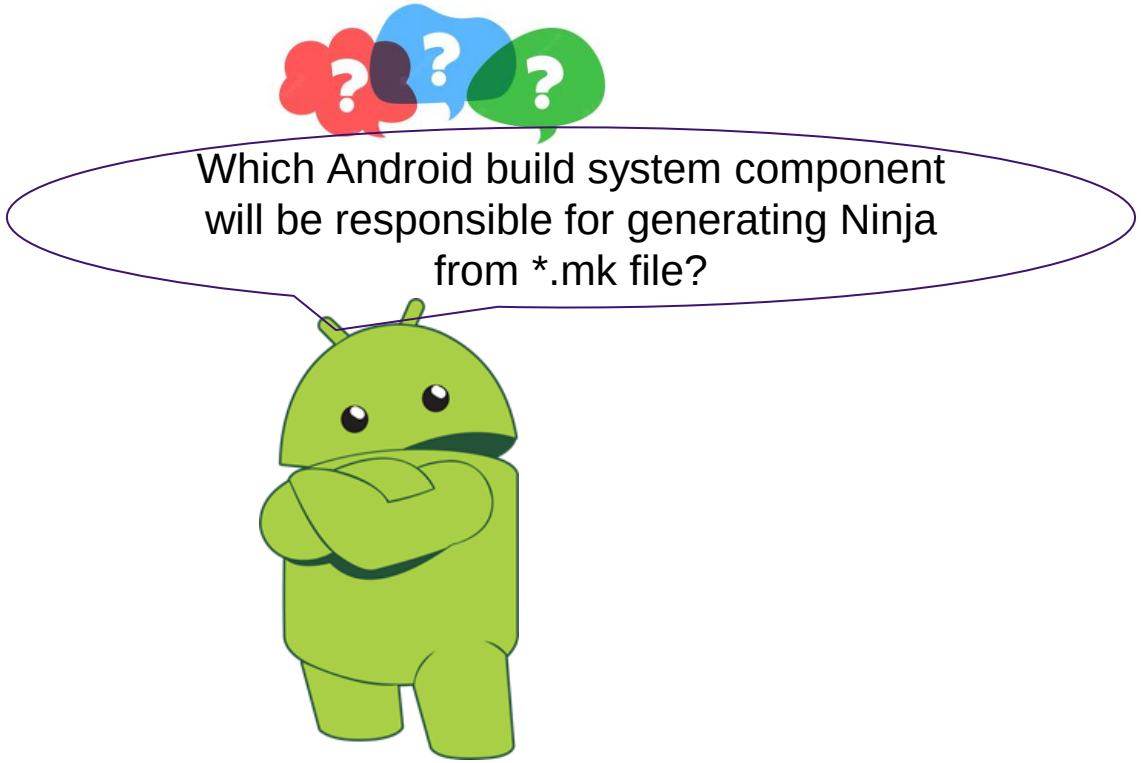
(Architecture)

QUIZ (1/10)



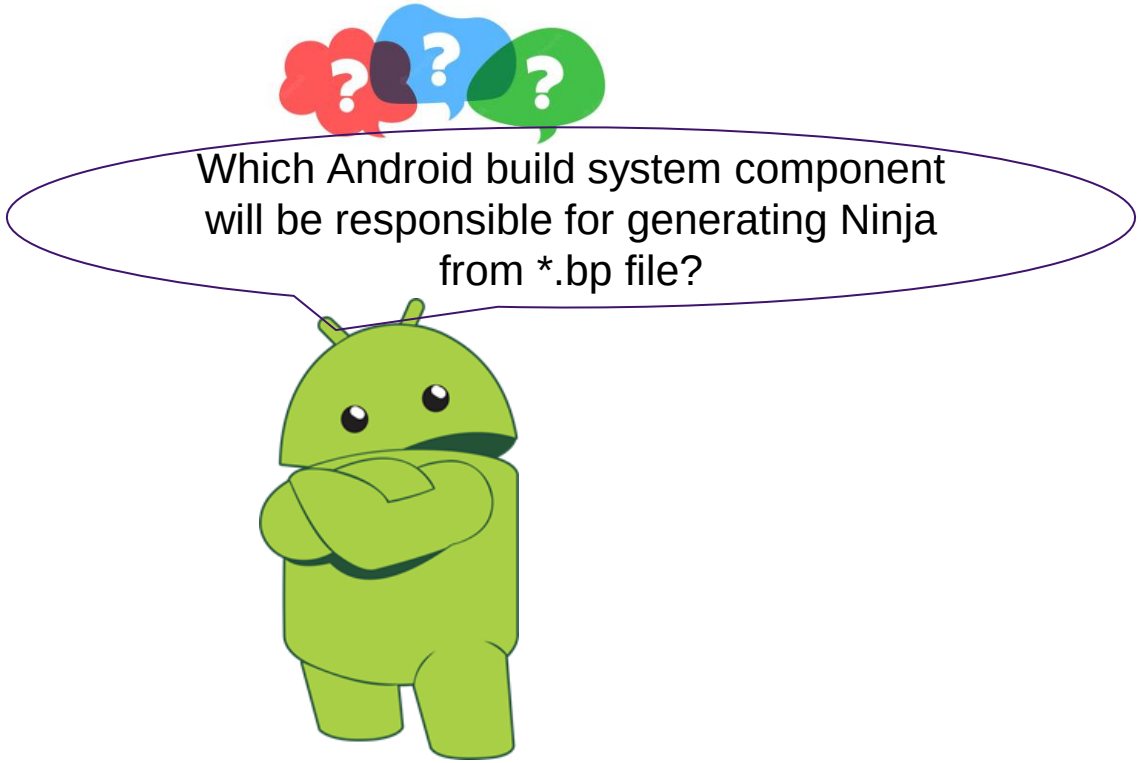
BOSCH

QUIZ (2/10)



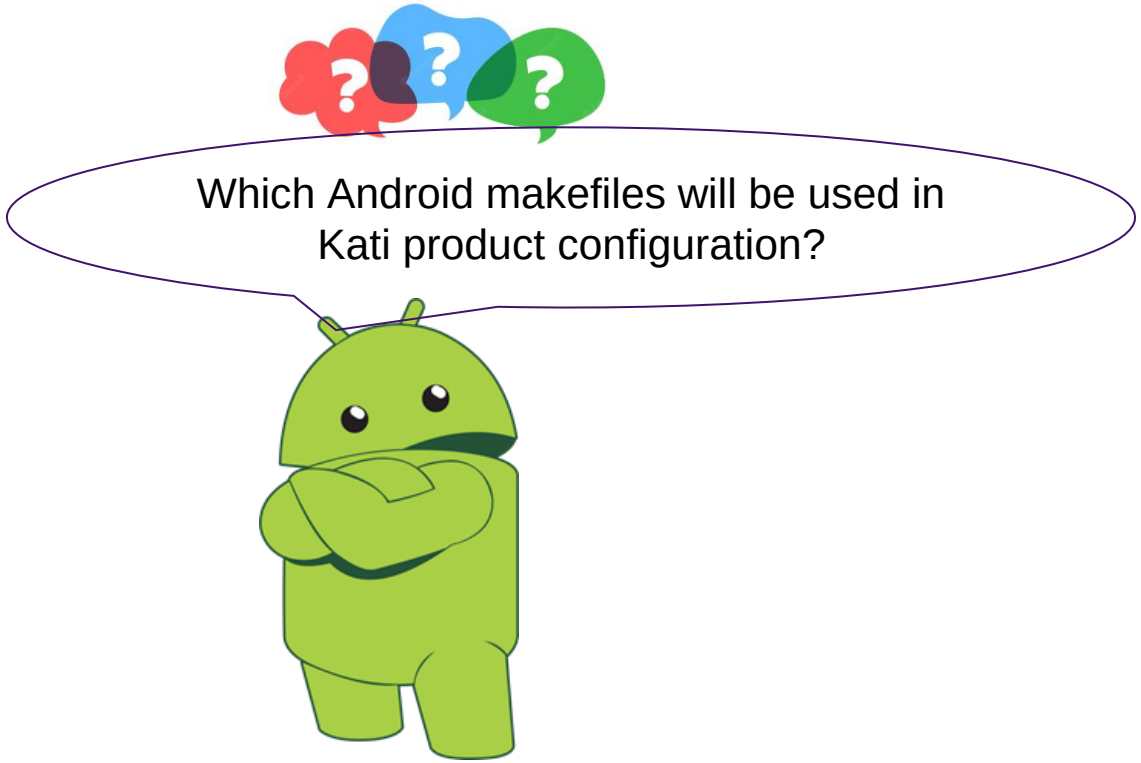
BOSCH

QUIZ (3/10)



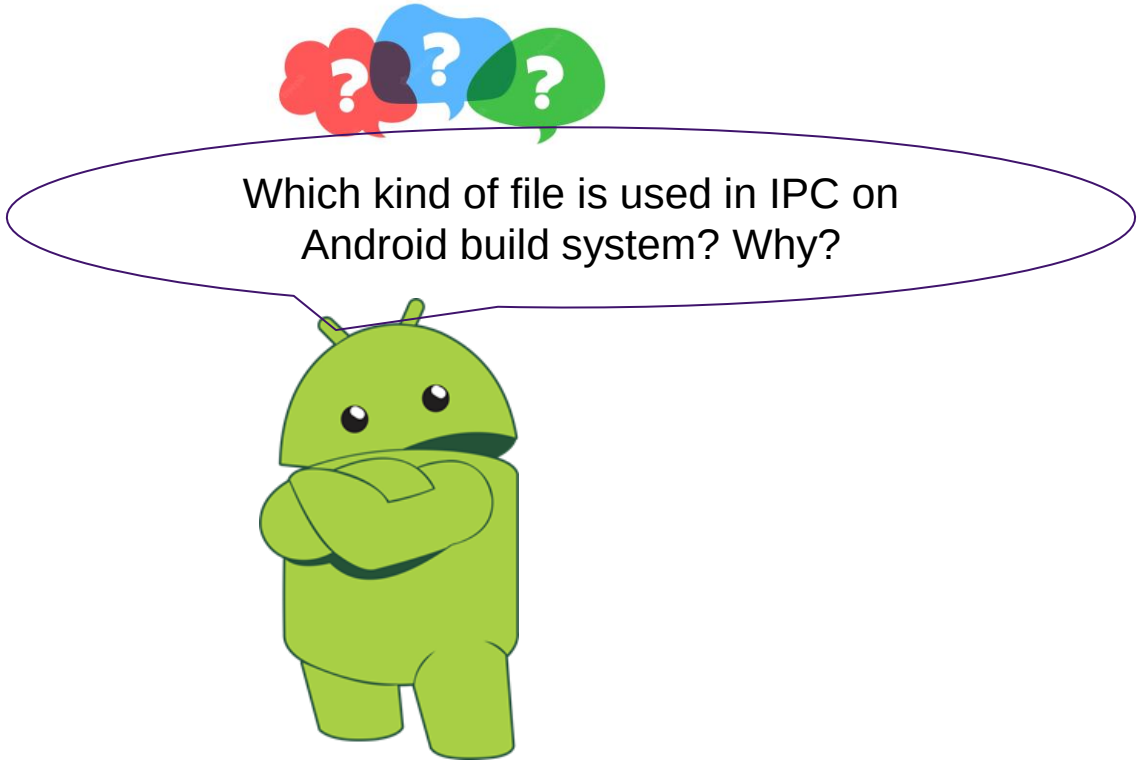
BOSCH

QUIZ (4/10)

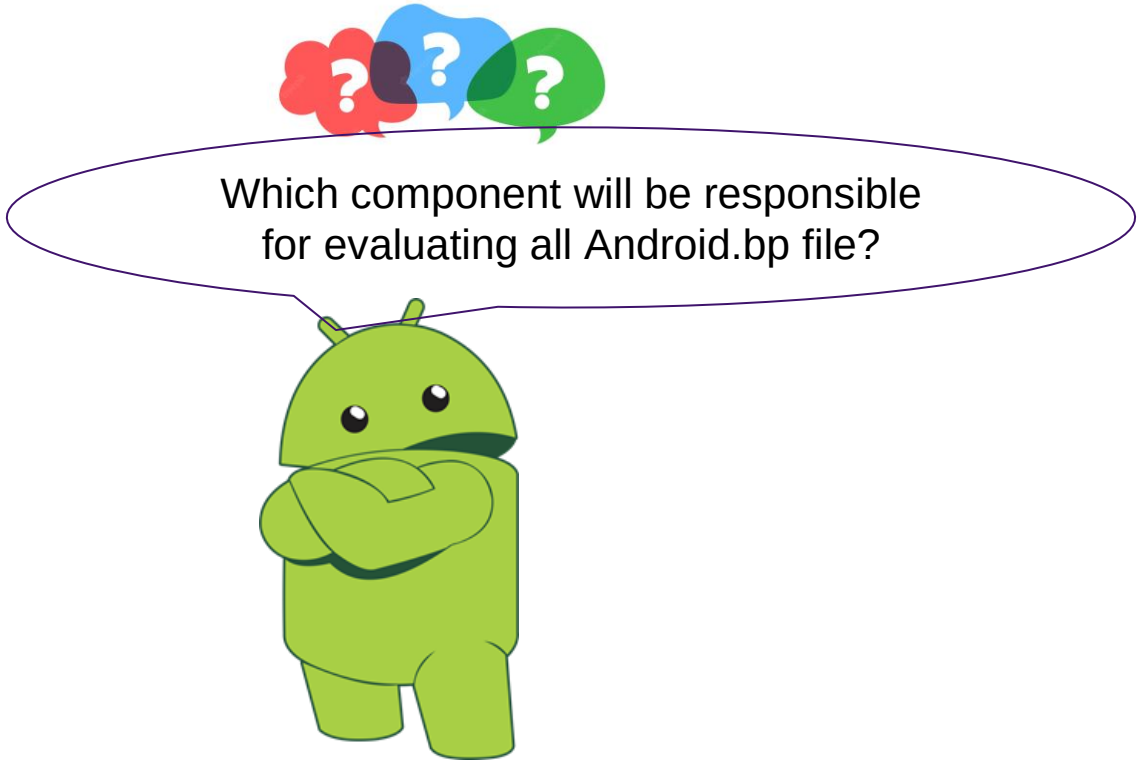


BOSCH

QUIZ (5/10)

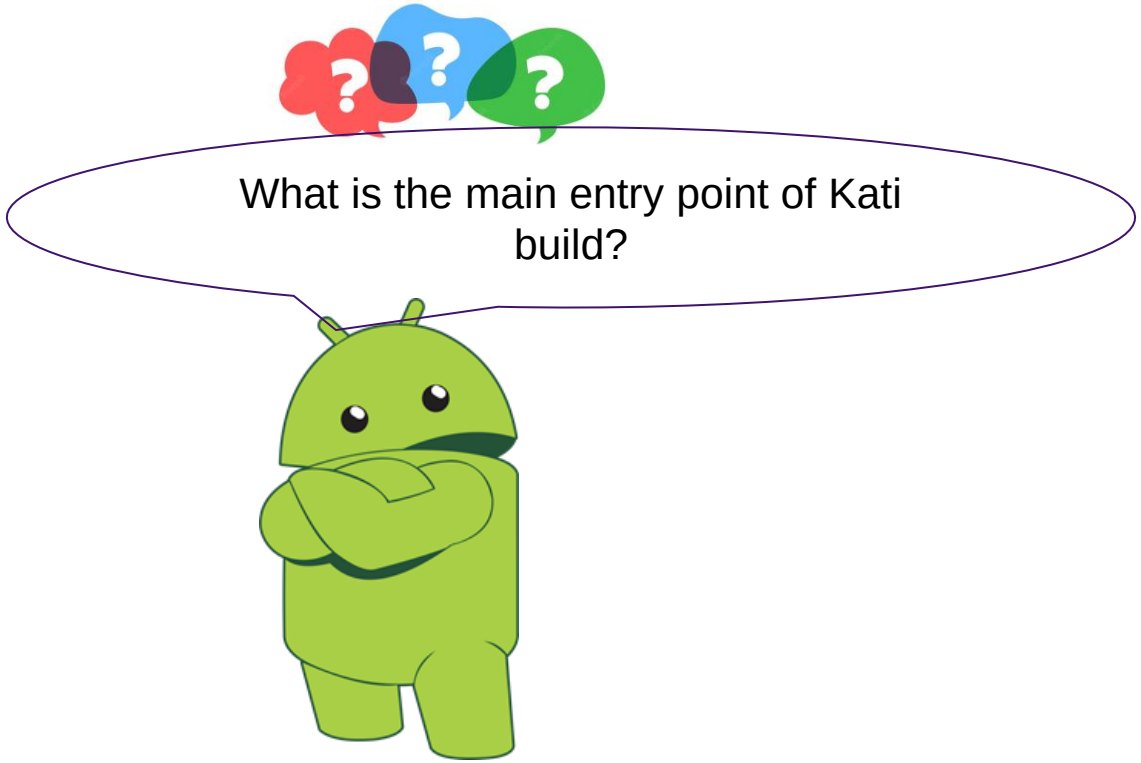


QUIZ (6/10)



BOSCH

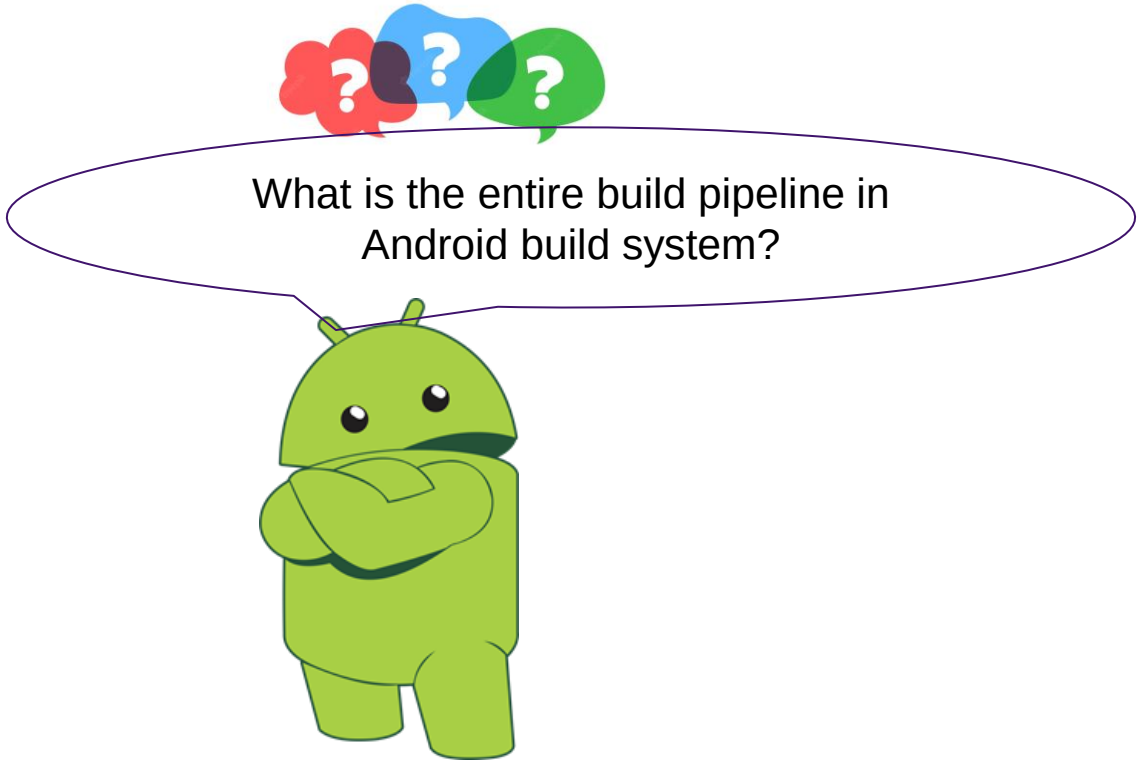
QUIZ (7/10)



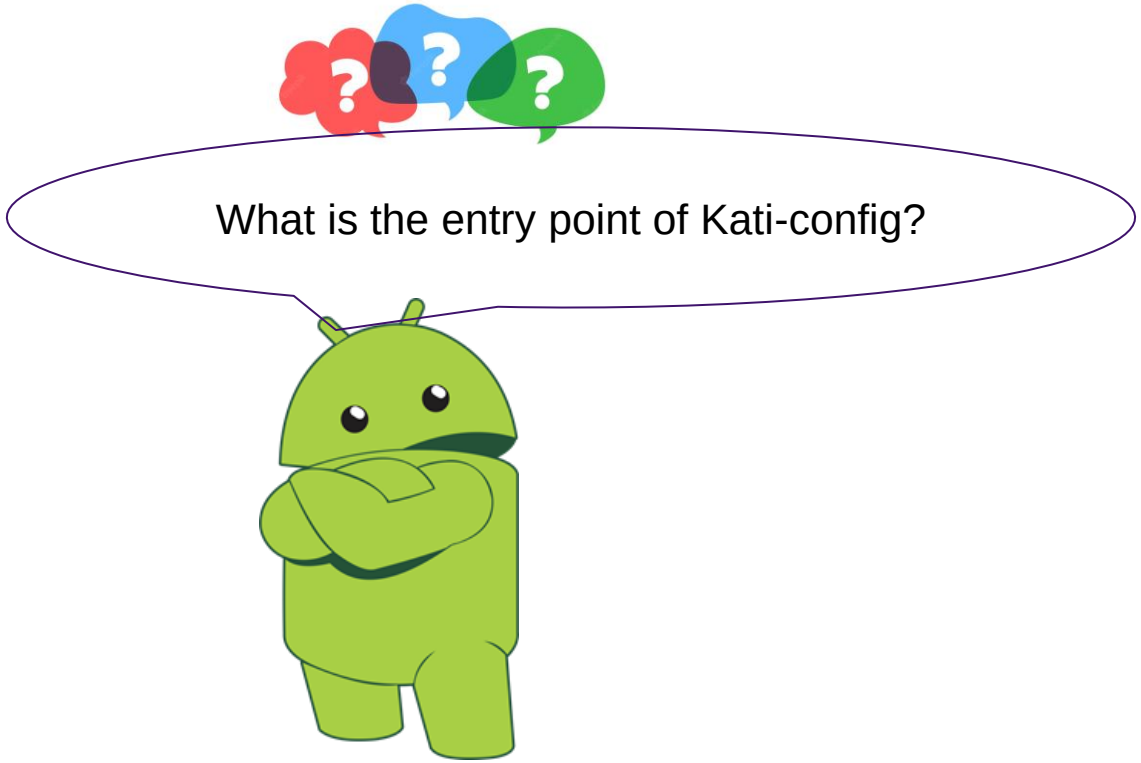
BOSCH

QUIZ (8/10)

QUIZ (9/10)

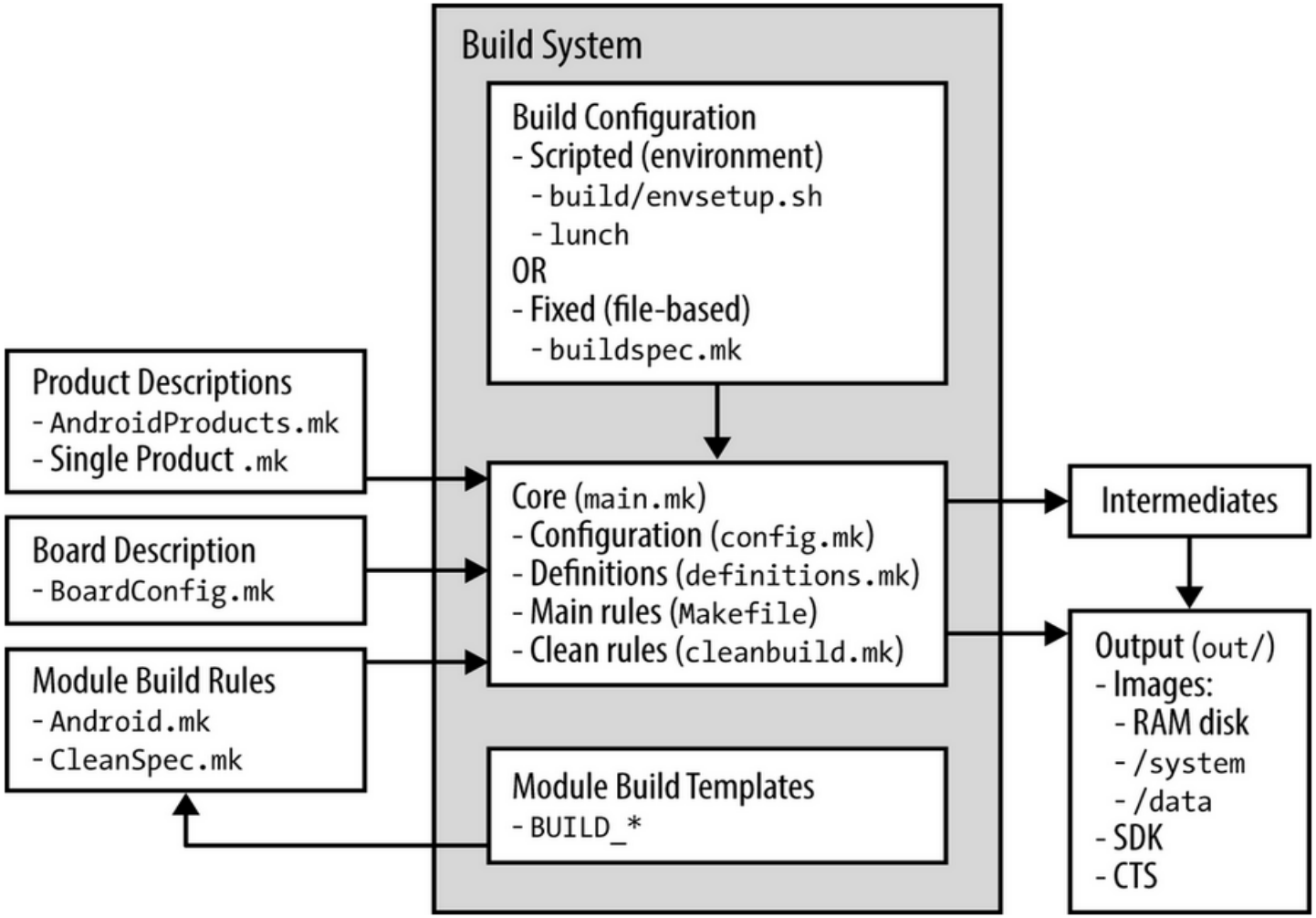


QUIZ (10/10)

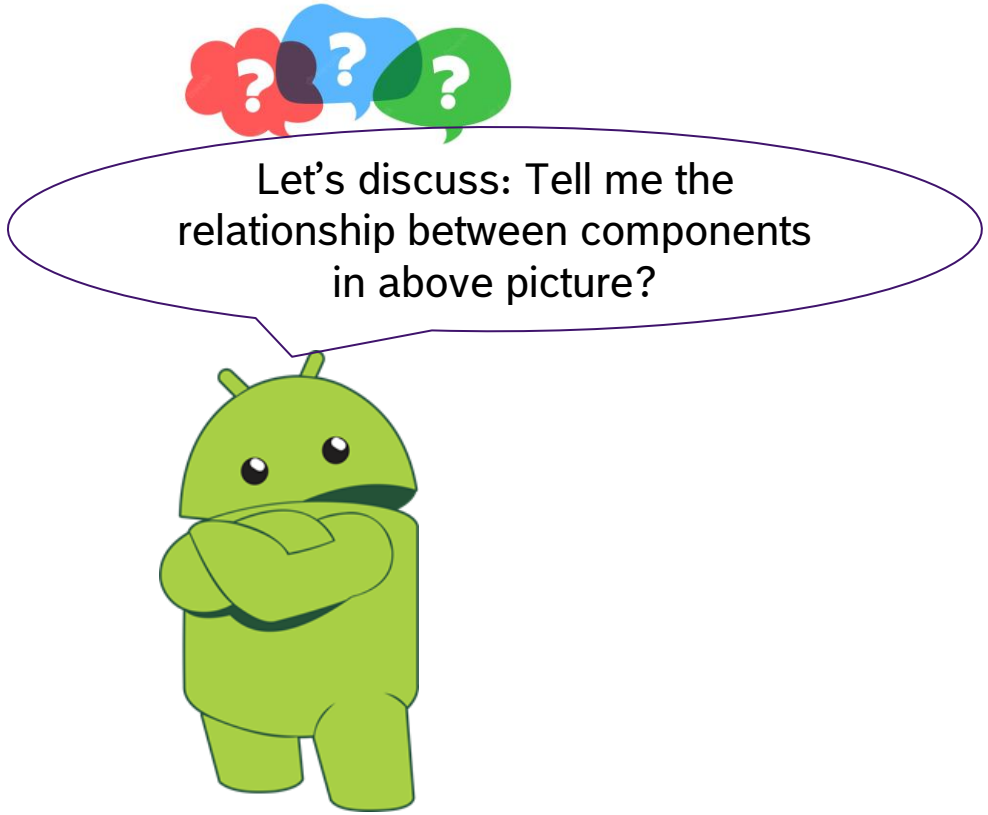


BOSCH

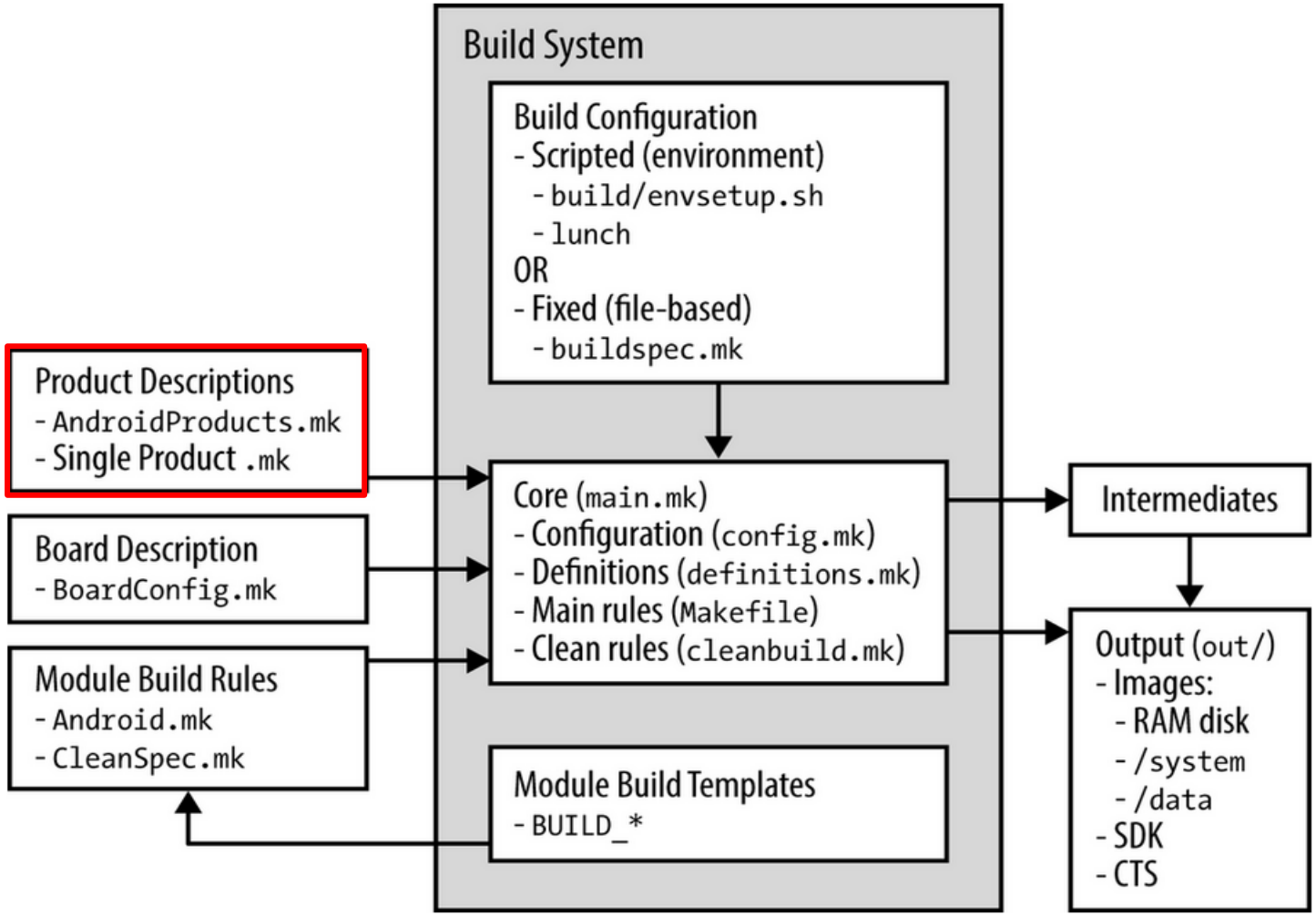
Architecture



Group working



Architecture



Architecture

- The entry point to make Android build system is the **main.mk** file found in the **build/core/** directory

```
rag2hc@HC1VMOSD5:~/00_working$ ls
Android.bp  bootable        BUILD         dalvik      device      hardware  libnativehelper  out       platform_testing  system     tools
art         bootstrap.bash  compatibility developers  external    kernel    log              packages  prebuilts         test       vendor
bionic      build           cts           development frameworks  libcore   Makefile         pdk       sdk               toolchain  WORKSPACE
rag2hc@HC1VMOSD5:~/00_working$ cat Makefile
### DO NOT EDIT THIS FILE ###
include build/make/core/main.mk
### DO NOT EDIT THIS FILE ###
rag2hc@HC1VMOSD5:~/00_working$ |
```

- Inside **main.mk**, the build system will call build configuration through the inclusion of **config.mk**

```
.PHONY: droid_targets
droid_targets:

# Set up various standard variables based on configuration
# and host information.
include build/make/core/config.mk
```

Architecture

- Inside **config.mk**, the build system will call user's build configuration through the inclusion of **envsetup.mk**

```
# ------------------------------------------------------------
# Define most of the global variables.  These are the ones that
# are specific to the user's build configuration.
include $(BUILD_SYSTEM)/envsetup.mk
```

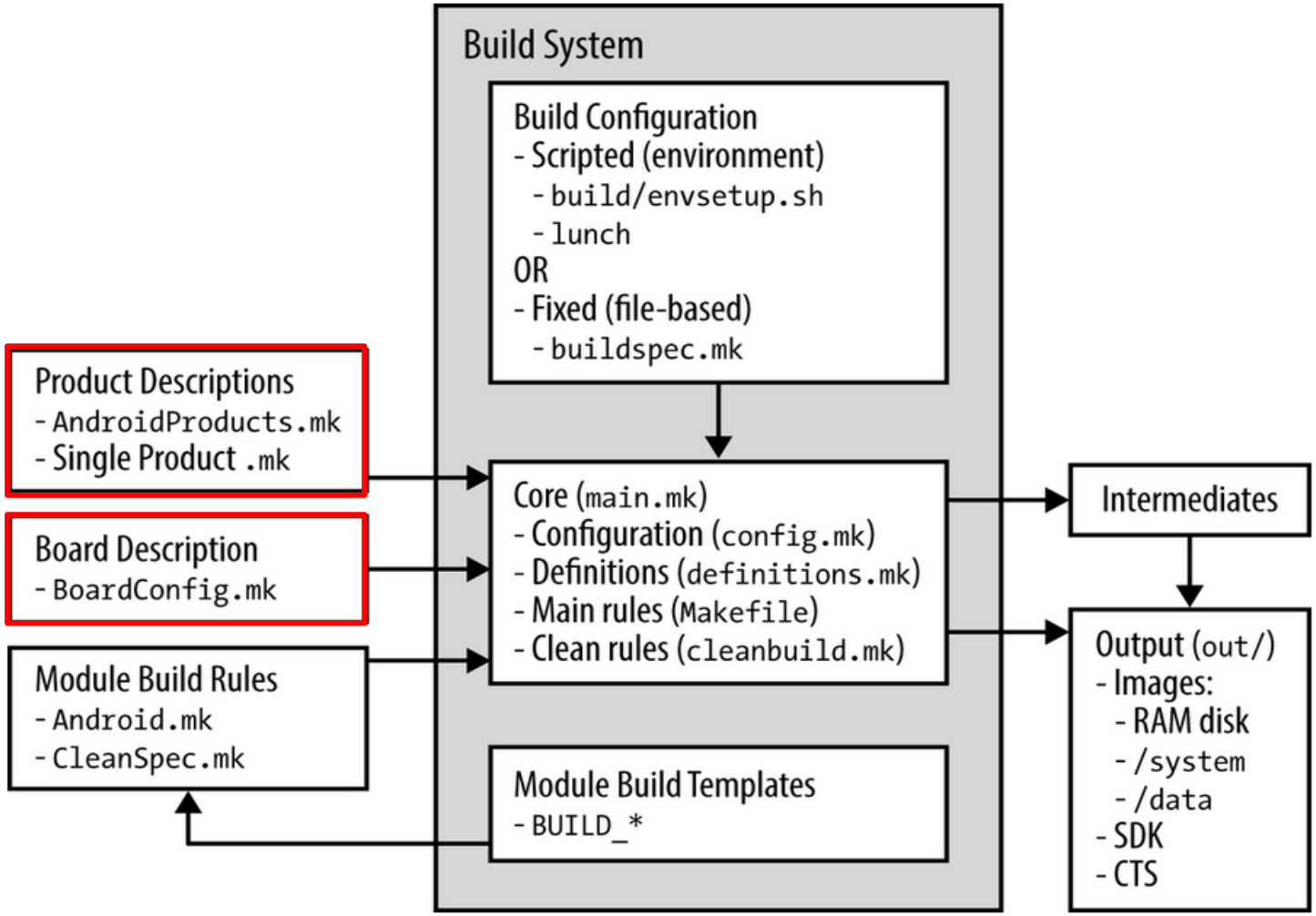
- Inside **envsetup.mk**, the build system will read the product specs and other variables through the inclusion of **product_config.mk**

```
# Read the product specs so we can get TARGET_DEVICE and other
# variables that we need in order to locate the output files.
include $(BUILD_SYSTEM)/product_config.mk
```

- Inside **product_config.mk**, the build system will read all product definitions specified by the **AndroidProducts.mk** file in the tree:

```
# Read in all of the product definitions specified by the AndroidProducts.mk
# files in the tree.
all_product_configs := $(get-all-product-makefiles)
```

Architecture



Architecture

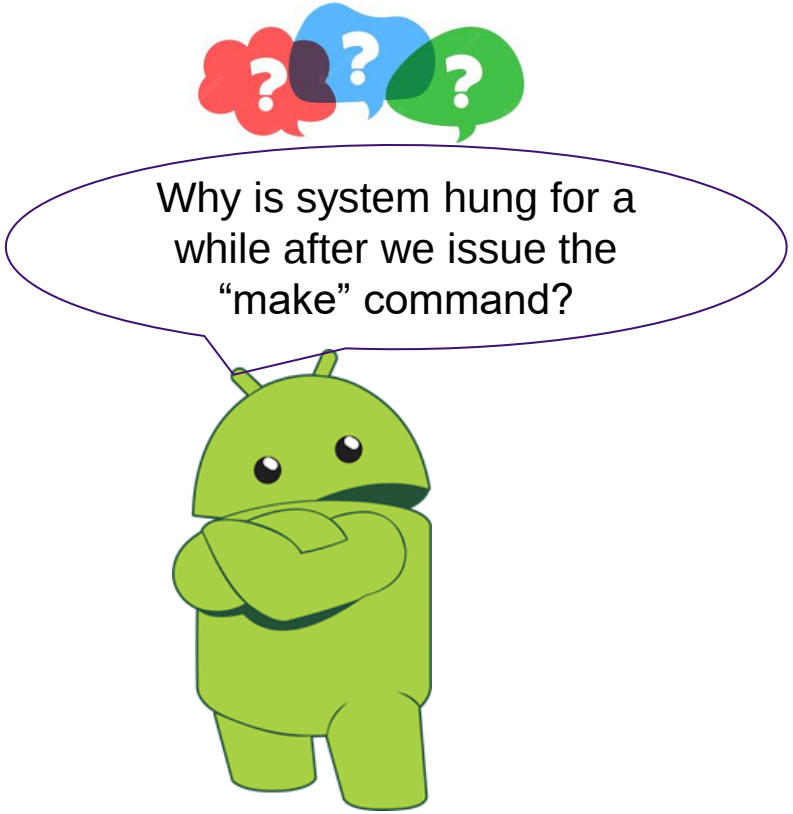
- Inside **envsetup.mk**, the build system will read board configuration and other variables through the inclusion of **board_config.mk**

```
include $(BUILD_SYSTEM)/board_config.mk
```

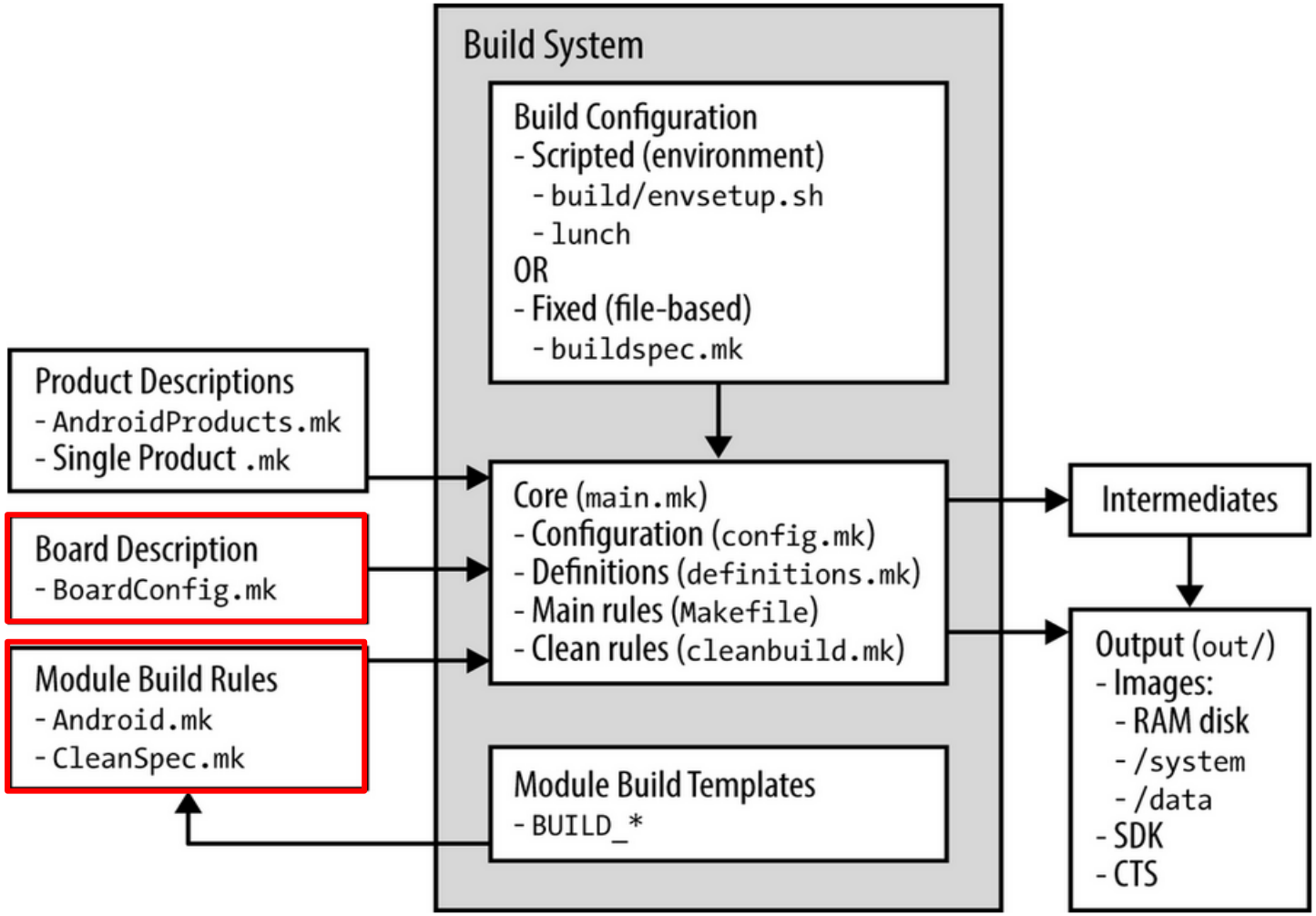
- Inside **board_config.mk**, the build system will search definition at **$(SRC_TARGET_DIR)/board/$(TARGET_DEVICE)** or **vendor/*/$(TARGET_DEVICE)**

```
ifdef TARGET_DEVICE_DIR
  ifneq ($(origin TARGET_DEVICE_DIR),command line)
    $(error TARGET_DEVICE_DIR may not be set manually)
  endif
  board_config_mk := $(TARGET_DEVICE_DIR)/BoardConfig.mk
else
  board_config_mk := \
    $(strip $(sort $(wildcard \
      $(SRC_TARGET_DIR)/board/$(TARGET_DEVICE)/BoardConfig.mk \
      $(shell test -d device && find -L device -maxdepth 4 -path '*/$(TARGET_DEVICE)/BoardConfig.mk') \
      $(shell test -d vendor && find -L vendor -maxdepth 4 -path '*/$(TARGET_DEVICE)/BoardConfig.mk') \
    )))
  ifeq ($(board_config_mk),)
    $(error No config file found for TARGET_DEVICE $(TARGET_DEVICE))
  endif
  ifneq ($(words $(board_config_mk)),1)
    $(error Multiple board config files for TARGET_DEVICE $(TARGET_DEVICE): $(board_config_mk))
  endif
  TARGET_DEVICE_DIR := $(patsubst %/,%,$(dir $(board_config_mk)))
  .KATI_READONLY := TARGET_DEVICE_DIR
endif

include $(board_config_mk)
```

Group working



Architecture



Architecture

- Inside **main.mk**, the build system will search all subdirs to find out all **Android.mk**/**Android.bp** files.

```
#
# Include all of the makefiles in the system
#

subdir_makefiles := $(SOONG_ANDROID_MK) $(file <$(OUT_DIR)/.module_paths/Android.mk.list) $(SOONG_OUT_DIR)/late-$(TARGET_PRODUCT).mk
subdir_makefiles_total := $(words int $(subdir_makefiles) post finish)
.KATI_READONLY := subdir_makefiles_total

$(foreach mk,$(subdir_makefiles),$(info [$(call inc_and_print,subdir_makefiles_inc)/$(subdir_makefiles_total)] including $(mk) ...)$(eval incl
ude $(mk)))
```

build/envsetup.sh

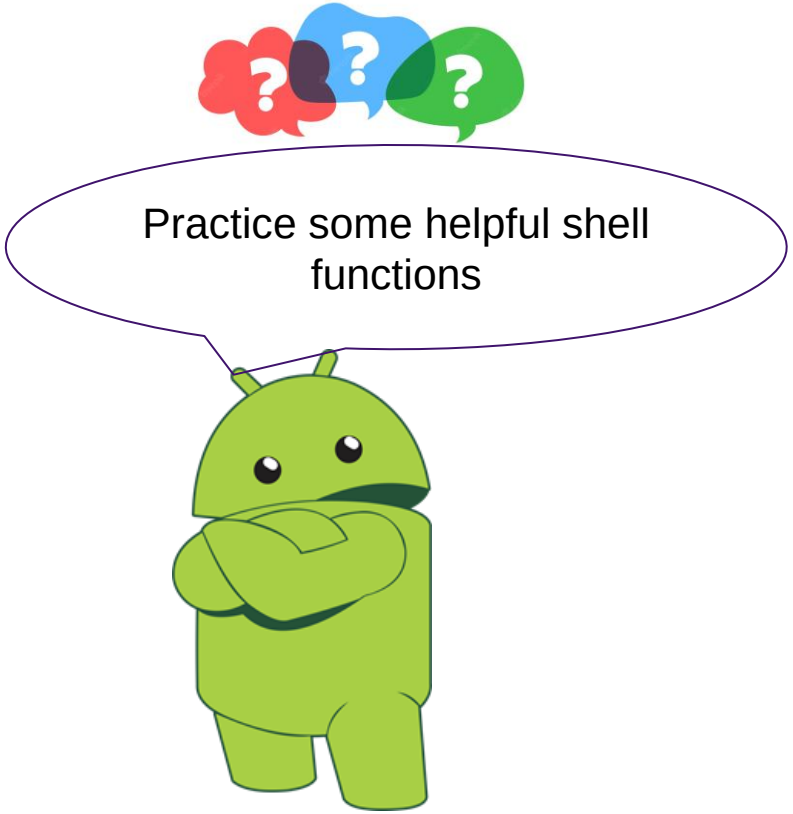
- This script is for setting up the **build env on the current shell** => that is the reason why we have to *source* it
- It added many useful macros (please type *hmm* to list all macros created).

```
rag2hc@VM-OSD5-RAG2HC:~/CCS2/00_prj/HA_aafw_master/android$ hmm

Run "m help" for help with the build system itself.

Invoke ". build/envsetup.sh" from your shell to add the following functions to your environment:
- lunch:      lunch <product_name>-<build_variant>
              Selects <product_name> as the product to build, and <build_variant> as the variant to
              build, and stores those selections in the environment to be read by subsequent
              invocations of 'm' etc.
- tapas:      tapas [<App1> <App2> ...] [arm|x86|mips|arm64|x86_64|mips64] [eng|userdebug|user]
- croot:      Changes directory to the top of the tree, or a subdirectory thereof.
- m:          Makes from the top of the tree.
- mm:         Builds all of the modules in the current directory, but not their dependencies.
- mmm:        Builds all of the modules in the supplied directories, but not their dependencies.
              To limit the modules being built use the syntax: mmm dir/:target1,target2.
- mma:        Builds all of the modules in the current directory, and their dependencies.
- mmma:       Builds all of the modules in the supplied directories, and their dependencies.
- provision:  Flash device with all required partitions. Options will be passed on to fastboot.
- cgrep:      Greps on all local C/C++ files.
- ggrep:      Greps on all local Gradle files.
- jgrep:      Greps on all local Java files.
- resgrep:    Greps on all local res/*.xml files.
- mangrep:    Greps on all local AndroidManifest.xml files.
- mgrep:      Greps on all local Makefiles files.
- sepgrep:    Greps on all local sepolicy files.
- sgrep:      Greps on all local source files.
- godir:      Go to the directory containing a file.
- allmod:     List all modules.
- gomod:      Go to the directory containing a module.
- pathmod:    Get the directory containing a module.
- refreshmod: Refresh list of modules for allmod/gomod.

Environment options:
- SANITIZE_HOST: Set to 'true' to use ASAN for all host modules. Note that
                 ASAN_OPTIONS=detect_leaks=0 will be set by default until the
                 build is leak-check clean.
- ANDROID_QUIET_BUILD: set to 'true' to display only the essential messages.

Look at the source to view more functions. The complete list is:
addcompletions add_lunch_combo allmod build_build_var_cache cgrep check_product check_type check_variant chooseco
 coredump_setup cproj croot _croot destroy_build_var_cache enable_zsh_completion findmakefile get_abs_build_var g
tsdcardpath gettargetarch gettop ggrep godir gomod hmm is isviewserverstarted jgrep key_back key_home key_menu lu
sion qpid rcgrep refreshmod resgrep runhat runtest sepgrep setpaths set_sequence_number set_stuff_for_environment
mstack tapas tracedmdump treegrep validate_current_shell _wrap_build
rag2hc@VM-OSD5-RAG2HC:~/CCS2/00_prj/HA_aafw_master/android$
```

build/envsetup.sh

- Some helpful shell functions:
  - **printconfig**: Prints the current configuration as set by the lunch and choose combo commands.
  - **m**: Runs make from the top of the tree. This is useful because you can run make from within subdirectories. If you have the TOP environment variable set, it uses that. If you don't, it looks up the tree from the current directory, trying to find the top of the tree.
  - **croot**: cd to the top of the tree.
  - **sgrep**: grep for the regex you provide in all .c, .cpp, .h, .java, and .xml files below the current directory.
  - **mgrep**: grep for the regex you provide in all Android makefile below the current directory.

Group working (10 mins)



lunch

- This is a shell function defined in ***build/envsetup.sh***
- It is the easiest way to configure a build. We can either launch it **without any argument** and it will ask to **choose among a list of known "combo"** or **launch it with the desired combos as argument**.

```
rag2hc@HC1VMOSD5:~/00_working$ source build/envsetup.sh
rag2hc@HC1VMOSD5:~/00_working$ lunch sdk_car_x86_64-userdebug
```

```
rag2hc@HC1VMOSD5:~/00_working$ source build/envsetup.sh
rag2hc@HC1VMOSD5:~/00_working$ lunch 73
```

- It sets the environment variables needed for the build and allows to start compiling at last.

```
rag2hc@HC1VMOSD5:~/00_working$ source build/envsetup.sh
rag2hc@HC1VMOSD5:~/00_working$ lunch

You're building on Linux

Lunch menu... pick a combo:
     1. aosp_arm-eng
     2. aosp_arm64-eng
     3. aosp_barbet-userdebug
     4. aosp_blueline-userdebug
     5. aosp_blueline_car-userdebug
     6. aosp_bonito-userdebug
     7. aosp_bonito_car-userdebug
     8. aosp_bramble-userdebug
      ................
     64. pixel3_mainline-userdebug
     65. poplar-eng
     66. poplar-user
     67. poplar-userdebug
     68. qemu_trusty_arm64-userdebug
     69. sdk_car_arm-userdebug
     70. sdk_car_arm64-userdebug
     71. sdk_car_portrait_x86_64-userdebug
     72. sdk_car_x86-userdebug
     73. sdk_car_x86_64-userdebug
     74. silvermont-eng
     75. uml-userdebug
     76. yukawa-userdebug
     77. yukawa_sei510-userdebug

Which would you like? [aosp_arm-eng] |
```

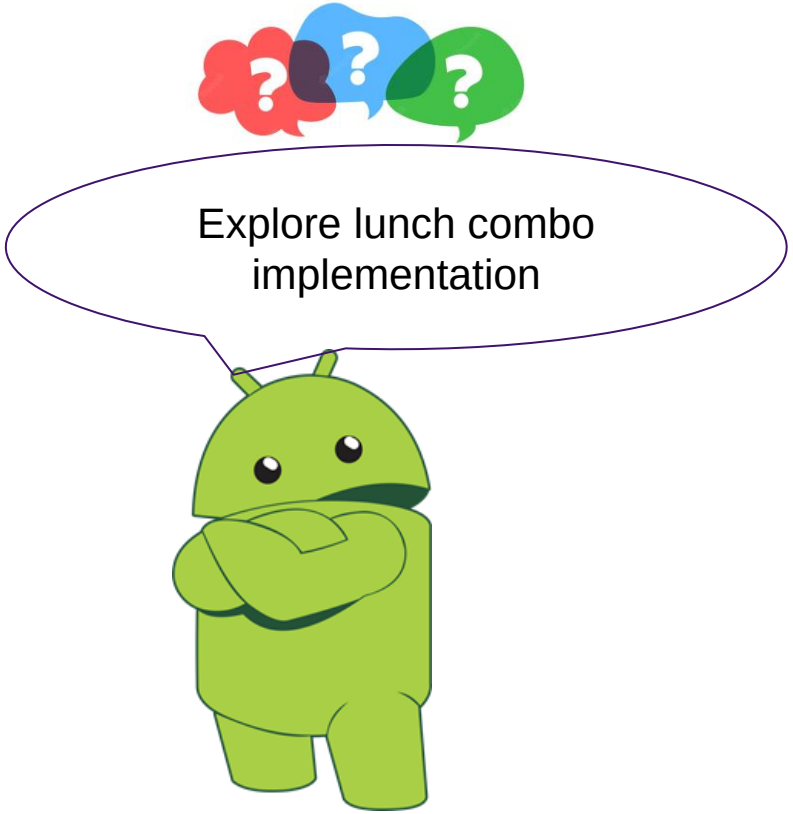
lunch

- In case, there is no argument, it will list all the combos available in current env:
  - By following all **vendor/*** and **device/*** folders, looking for either *vendorsetup.sh* files or *COMMON_LUNCH_CHOICES*

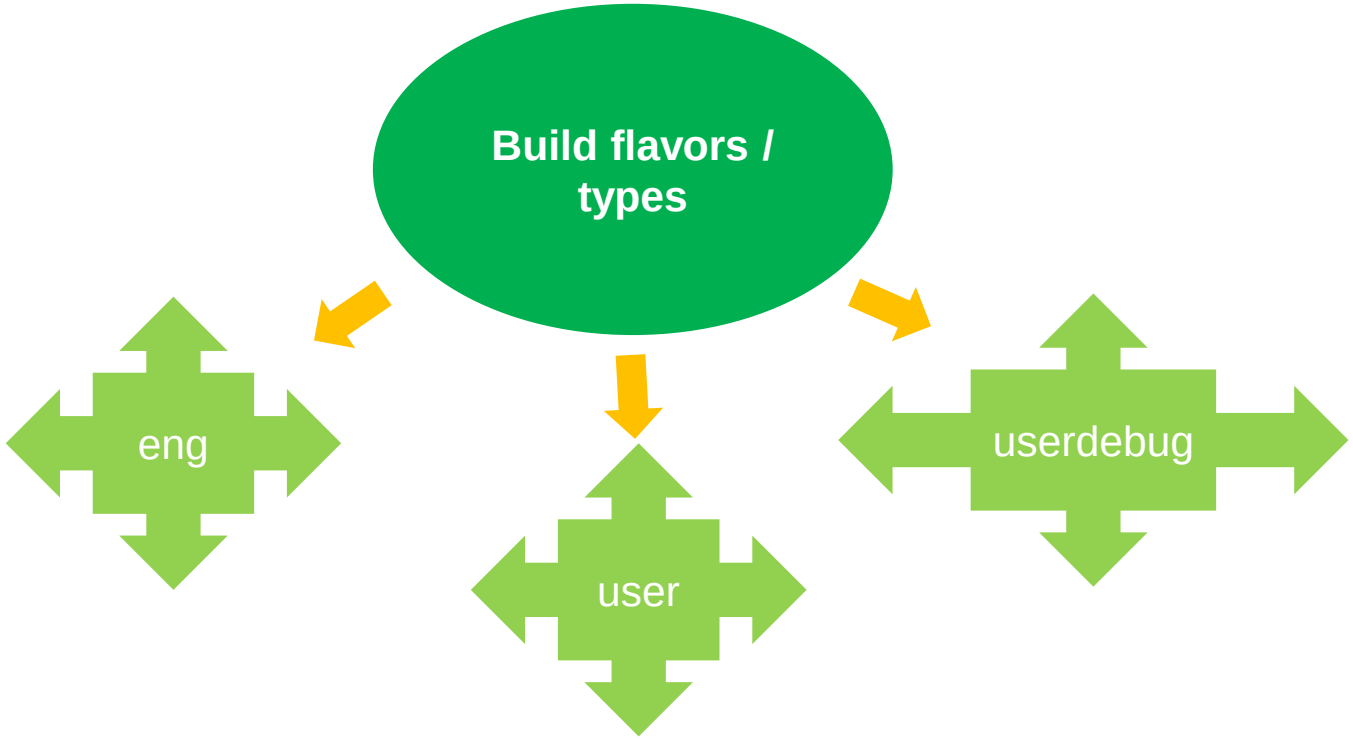
```
./renesas/salvator/vendorsetup.sh
./generic/uml/vendorsetup.sh
./generic/mini-emulator-x86_64/vendorsetup.sh
./generic/mini-emulator-mips/vendorsetup.sh
./generic/mini-emulator-x86/vendorsetup.sh
./generic/mini-emulator-arm64/vendorsetup.sh
./generic/mini-emulator-armv7-a-neon/vendorsetup.sh
./generic/mini-emulator-mips64/vendorsetup.sh
./generic/car/vendorsetup.sh
```

```
device/generic/car/AndroidProducts.mk:COMMON_LUNCH_CHOICES := \
device/generic/car/AndroidProducts.mk-    aosp_car_arm-userdebug \
device/generic/car/AndroidProducts.mk-    aosp_car_arm64-userdebug \
device/generic/car/AndroidProducts.mk-    aosp_car_x86-userdebug \
device/generic/car/AndroidProducts.mk-    aosp_car_x86_64-userdebug \
device/generic/car/AndroidProducts.mk-    car_x86_64-userdebug \
--
device/generic/mini-emulator-x86/AndroidProducts.mk:COMMON_LUNCH_CHOICES := \
device/generic/mini-emulator-x86/AndroidProducts.mk-    mini_emulator_x86-userdebug
--
device/generic/mini-emulator-x86_64/AndroidProducts.mk:COMMON_LUNCH_CHOICES := \
device/generic/mini-emulator-x86_64/AndroidProducts.mk-    mini_emulator_x86_64-userdebug
--
device/generic/mini-emulator-arm64/AndroidProducts.mk:COMMON_LUNCH_CHOICES := \
device/generic/mini-emulator-arm64/AndroidProducts.mk-    mini_emulator_arm64-userdebug
--
device/generic/mini-emulator-armv7-a-neon/AndroidProducts.mk:COMMON_LUNCH_CHOICES := \
device/generic/mini-emulator-armv7-a-neon/AndroidProducts.mk-    m_e_arm-userdebug
--
device/alliance/aasp/AndroidProducts.mk:COMMON_LUNCH_CHOICES := \
device/alliance/aasp/AndroidProducts.mk-    aasp_emulator-userdebug
--
device/google/cuttlefish/AndroidProducts.mk:COMMON_LUNCH_CHOICES := \
device/google/cuttlefish/AndroidProducts.mk-    aosp_cf_arm64_phone-userdebug \
device/google/cuttlefish/AndroidProducts.mk-    aosp_cf_x86_64_phone-userdebug \
device/google/cuttlefish/AndroidProducts.mk-    aosp_cf_x86_auto-userdebug \
device/google/cuttlefish/AndroidProducts.mk-    aosp_cf_x86_phone-userdebug \
device/google/cuttlefish/AndroidProducts.mk-    aosp_cf_x86_tv-userdebug
rag2hc@VM-OSD5-RAG2HC:~/CCS2/00_prj/HA_aafw_master/android$
```

Group working (5 mins)



Build flavors/types



Build flavors/types

- **eng**: This is the default flavor. A plain "make" is the same as "make eng". droid is an alias for eng. The build system will
  - Install modules tagged with: **eng**, **debug**, **user**, and/or **development**.
  - Install non-APK modules that have **no tags** specified.
  - Install APKs according to **the product definition** files, in addition to **tagged** APKs.
  - Set **ro.secure=0**
  - Set **ro.debuggable=1**
  - Set **ro.kernel.android.checkjni=1**
  - adb is **enabled** by default.

*Note: **eng** and **debug** are now obsolete. For the equivalent functionality, use PRODUCT_PACKAGES_ENG or PRODUCT_PACKAGES_DEBUG in the appropriate product makefiles.*

Build flavors/types

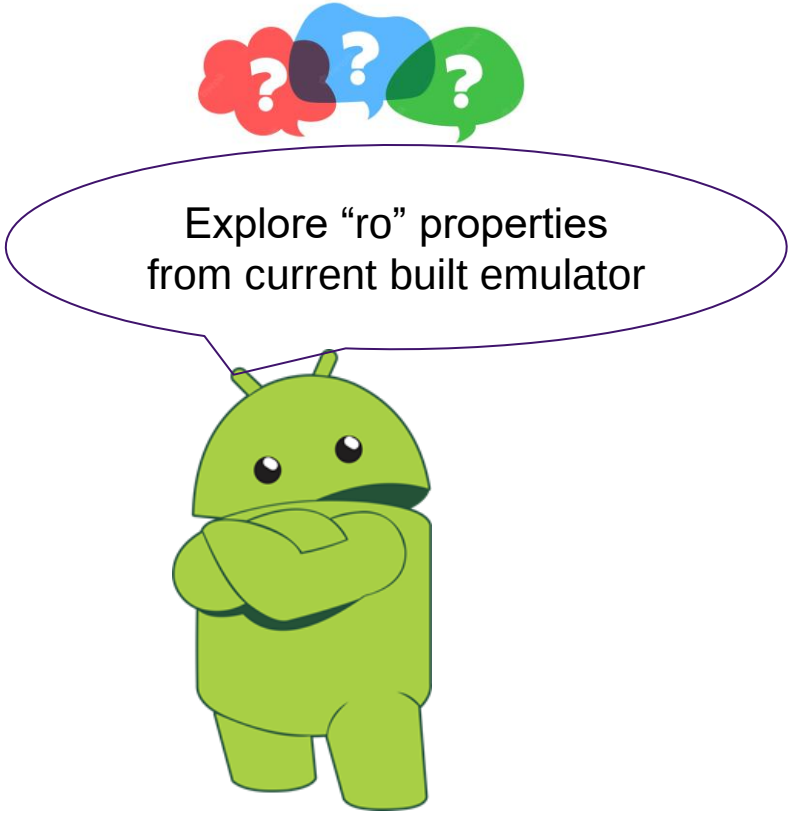
- **user**: This is the flavor intended to be the **final release** bits. The build system will
  - Installs modules tagged with **user**.
  - Installs non-APK modules that have **no tags** specified.
  - Installs APKs according to **the product definition** files; **tags are ignored for APK** modules.
  - Set **ro.adb.secure=1**
  - Set **ro.secure=1**
  - Set **ro.debuggable=0**
  - adb is **disabled** by default.

Build flavors/types

- **userdebug**: Same as user, except:
  - Install modules tagged with **debug**.
  - Set **ro.debuggable=1**
  - adb is **enabled** by default.

*Note: **eng** and **debug** are now obsolete. For the equivalent functionality, use PRODUCT_PACKAGES_ENG or PRODUCT_PACKAGES_DEBUG in the appropriate product makefiles.*

Group working (5 mins)



Combos

- A build combo are combination of a product to build and the variant to use:
  - **TARGET_PRODUCT**: a product defines how the final Android image is, selecting its services, initialization, application to be install

```
device/alliance/aasp/bootanimationex/bootanimationex.mk:ifeq ($(findstring _emulator,$(TARGET_PRODUCT)),)
device/alliance/aasp/emulator/earlydata/Android.mk:ifneq ($(findstring _emulator,$(TARGET_PRODUCT)),)
device/alliance/aasp/emulator/earlydata/earlydata.mk:ifneq ($(findstring _emulator,$(TARGET_PRODUCT)),)
device/alliance/aasp/emulator/audio/aasp_emulator.mk:ifeq ($(TARGET_PRODUCT),aasp_emulator)
device/alliance/aasp/emulator/audio/aasp_emulator.mk:OUT_INTERMEDIATES=$(OUT_DIR)/target/product/$(TARGET_PRODUCT)/obj
```

  - **TARGET_BUILD_VARIANT**: select the purpose of this build (**user**, **userdebug** or **eng**).

```
device/alliance/aasp/emulator/audio/aasp_emulator.mk:ifneq (,$(filter userdebug eng,$(TARGET_BUILD_VARIANT)))
device/alliance/aasp/emulator/audio/aasp_emulator.mk:ifneq (,$(filter userdebug eng,$(TARGET_BUILD_VARIANT)))
device/alliance/aasp/aasp.mk:ifneq (,$(filter userdebug eng, $(TARGET_BUILD_VARIANT)))
device/alliance/aasp/someiptcu/someiptcu.mk:ifneq (,$(filter userdebug eng, $(TARGET_BUILD_VARIANT)))
device/alliance/aasp/alliance_car/alliance_car_build.mk:ifneq (,$(filter userdebug eng, $(TARGET_BUILD_VARIANT)))
device/alliance/aasp/someip/someip.mk:ifneq (,$(filter userdebug eng, $(TARGET_BUILD_VARIANT)))
device/alliance/aasp/cluster/cluster.mk:ifneq (,$(filter userdebug, $(TARGET_BUILD_VARIANT)))
device/alliance/aasp/parking_aids/parking_aids.mk:ifneq (,$(filter userdebug eng, $(TARGET_BUILD_VARIANT)))
device/google/contexthub/firmware/build/common_config.mk:ifneq (,$(filter userdebug eng, $(TARGET_BUILD_VARIANT)))
device/google/atv/products/atv_base.mk:ifneq (,$(filter userdebug eng, $(TARGET_BUILD_VARIANT)))
rag2hc@VM-OSD5-RAG2HC:~/CCS2/00_prj/HA_aafw_master/android$
```

Combos

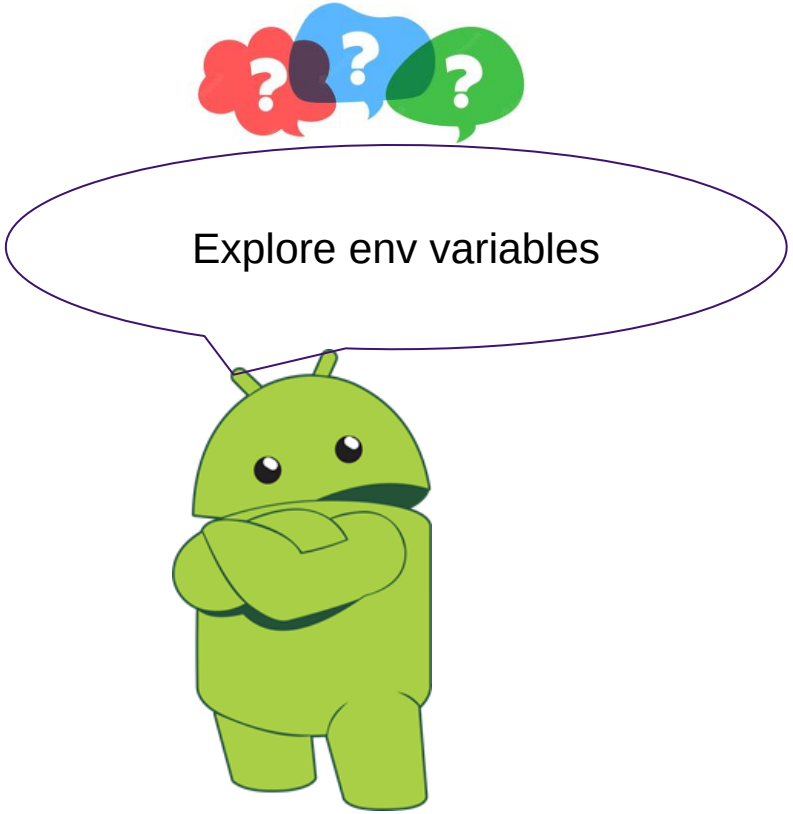
```
rag2hc@HC1VMOSD5:~/00_working$ source build/envsetup.sh
rag2hc@HC1VMOSD5:~/00_working$ lunch

You're building on Linux

Lunch menu... pick a combo:
     1. aosp_arm-eng
     2. aosp_arm64-eng
     3. aosp_barbet-userdebug
     4. aosp_blueline-userdebug
     5. aosp_blueline_car-userdebug
     6. aosp_bonito-userdebug
     7. aosp_bonito_car-userdebug
     8. aosp_bramble-userdebug
      ................
     64. pixel3_mainline-userdebug
     65. poplar-eng
     66. poplar-user
     67. poplar-userdebug
     68. qemu_trusty_arm64-userdebug
     69. sdk_car_arm-userdebug
     70. sdk_car_arm64-userdebug
     71. sdk_car_portrait_x86_64-userdebug
     72. sdk_car_x86-userdebug
     73. sdk_car_x86_64-userdebug
     74. silvermont-eng
     75. uml-userdebug
     76. yukawa-userdebug
     77. yukawa_sei510-userdebug

Which would you like? [aosp_arm-eng] |
```

Env variables

- **lunch** will set env variables used by the build:

Env variables	Values
PATH	*$ANDROID_JAVA_TOOLCHAIN:$ANDROID_BUILD_PATHS:$PATH*
ANDROID_TOOLCHAIN	*<build root>/prebuilts/gcc/linux-x86/x86/x86_64-linux-android-4.9/bin*
ANDROID_BUILD_TOP	*<build root>*
ANDROID_PRODUCT_OUT	*<build root>/out/target/product/*
TARGET_BUILD_VARIANT	*eng, user, userdebug*
TARGET_BUILD_TYPE	*debug or release*
TARGET_PRODUCT	*eg: aasp_emulator*
PLATFORM_VERSION	*Android version (eg: 10)*
TARGET_ARCH	*Target cpu architecture eg: x86_64, mips, arm, …*

Group working (5 mins)



buildspec.mk

- **lunch** is convenient to quickly switch from one configuration to another. However, if we have only **one product** or we want to do **more fine-grained configuration**, this is not really convenient.
- The file **buildspec.mk** is here for that.
- If we place it at the top of the sources, it will be used by the build system to get its configuration instead of relying on the environment variables.

buildspec.mk

- It offers more variables to modify, such as compiling a given module with debugging symbols, additional C compiler flags, change the output directory...
- A sample is available in **build/buildspec.mk.default**, with lots of comments on the various variables.

Module build

- To build a module from the top directory, just do **make <ModuleName>**
- The files generated will be put in **out/target/product/$TARGET_DEVICE/obj/<module_type>/<module_name>_intermediates**
- However, it **won't regenerate a new image**.
- This is just useful to make sure that the module is built.
- We will **have to do a full make to have an image that contains our module**

Module build

- Note that if a module tagged as **optional**, we **won't find it in our generated image**.
- To clean a single module, do **make clean-ModuleName**
- You can also get the list of the modules available in the build system with the **make modules** target.

Output

- All the output is generated in the **out/** directory, outside of the source code directory.
- This directory contains mostly two subdirectories: **host/** and **target/**
- These directories contain all the objects files compiled during the build process: **.o files** for **C/C++ code**, **.jar files** for **Java libraries**, etc.
- This is an interesting feature, since it keeps all the generated stuff separate from the source code, and we can easily clean without side effects

Output

- It also generates the system images in the **out/target/product/ <device_name>/** directory
- These images are:
  - **ramdisk.img**: Contains the root file system of Android, including:
    - **init.*** configuration files
    - **default.prop** containing the read only properties of this build
    - **/system** mounting point

Output

- It also generates the system images in the **out/target/product/ <device_name>/** directory
- These images are:
  - **system.img**: contains the components generated by the AOSP build, including:
    - Framework
    - Applications
    - daemons

Output

- It also generates the system images in the **out/target/product/ <device_name>/** directory
- These images are:
  - **userdata.img**:
    - Partition to hold generated content, usually empty after the build
  - **recovery.img, ramdisk-recovery.img**:
    - basic image partition used to recover user data or even the actual system if anything goes wrong
  - **vendor.img**:
    - A partition that will hold the vendor specific content.

Cleaning

- Cleaning is almost as easy as **rm -rf out/**
- **make clean** or **make clobber deletes all generated files**.
- **make installclean removes the installed files for the current combo**. It is useful when you work with several products to avoid doing a full rebuild each time you change from one to the other

Some useful build target

- **droid**: make droid is the normal build. This target is here because the default target has to have a name.
- **all**: make all builds everything make droid does, plus everything whose **LOCAL_MODULE_TAGS** do **not include the "droid" tag**. The build server runs this to make sure that everything that is in the tree and has an **Android.mk** builds.
- **clean-$(LOCAL_MODULE)** and **clean-$(LOCAL_PACKAGE_NAME)**: Let you selectively clean one target.
  - For example, we can type **make clean-libutils** and it will delete **libutils.so** and all of the **intermediate files**, or we can type **make clean-Home** and it will **clean just the Home app**.

Some useful build target

- **dataclean**: **make dataclean** deletes **contents of the data directory inside the current combo directory (out/target/product/<device>/data)**. This is especially **useful on the simulator and emulator**, where the persistent data remains present between builds.
- **LOCAL_MODULE**: Anything you specify as a **LOCAL_MODULE** in an **Android.**
  - **make runtime** might be shorthand for **make out/linux-x86-debug/system/bin/runtime**
  - **make libkjs** might be shorthand for **make out/linux-x86-debug/system/lib/libkjs.so**
- **targets**: **make targets** will **print a list of all of the LOCAL_MODULE names**

Exercises

Exercises

1) Connect with target device via USB. Then, open a new terminal to get following value:
   a. adb shell getprop
   b. abd shell getprop ro.secure
   c. abd shell getprop ro.debuggable
2) Be get familiar with **printconfig**, **m**, **mm**, **mmm**, **mma**, **croot**, **cgrep**, **jgrep**, **sgrep**, mgrep, **godir**
3) Be get familiar with **all make commands**

Note: please take screenshot or save to a log file for your work. Then upload to Practicing folder.

Thank for your listening!

BOSCH