

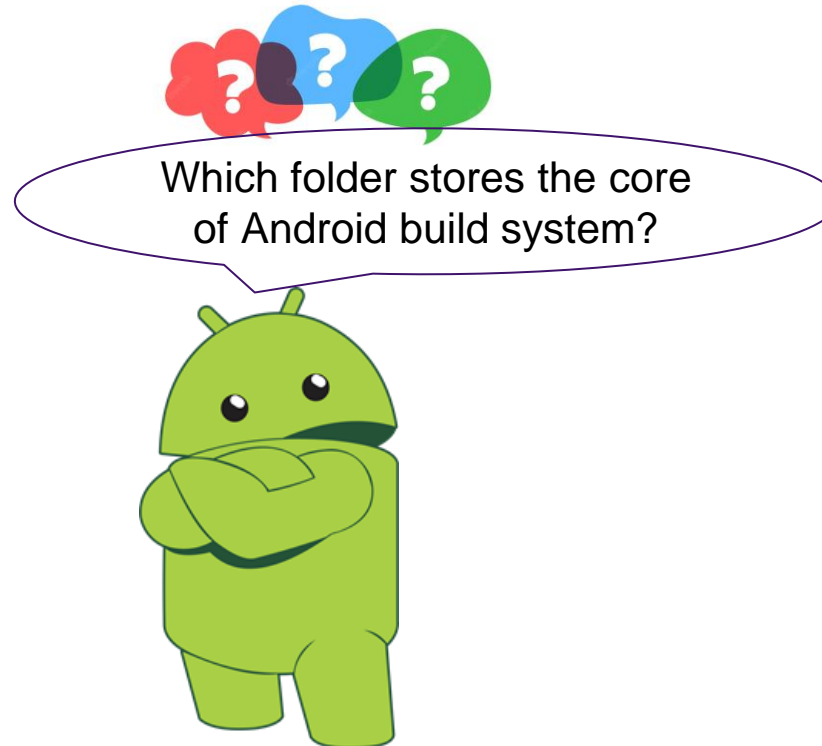
Android build system

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

Oct-2024

Day 2

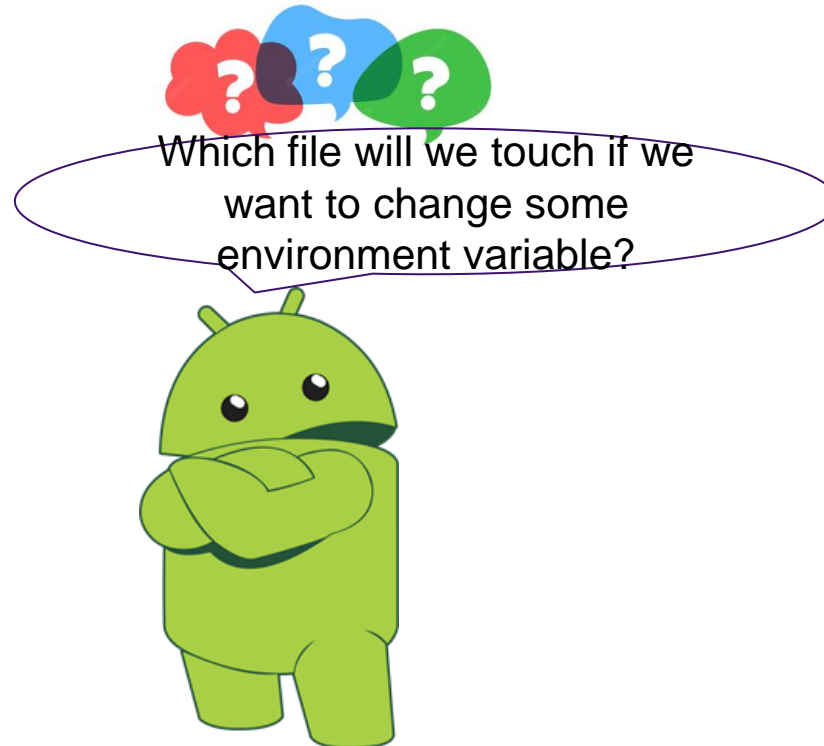
QUIZ (1/10)



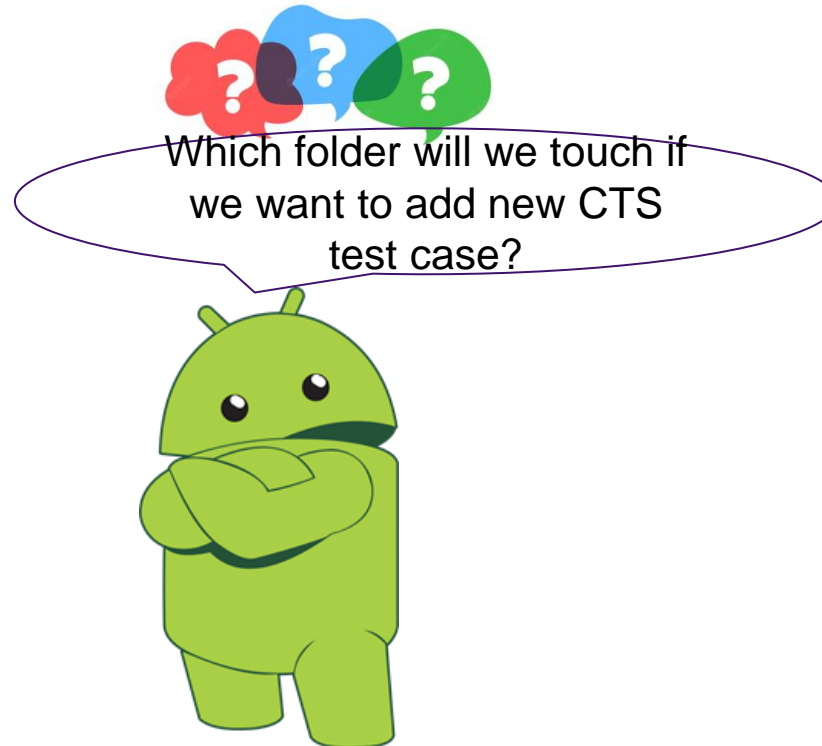
QUIZ (2/10)



QUIZ (3/10)



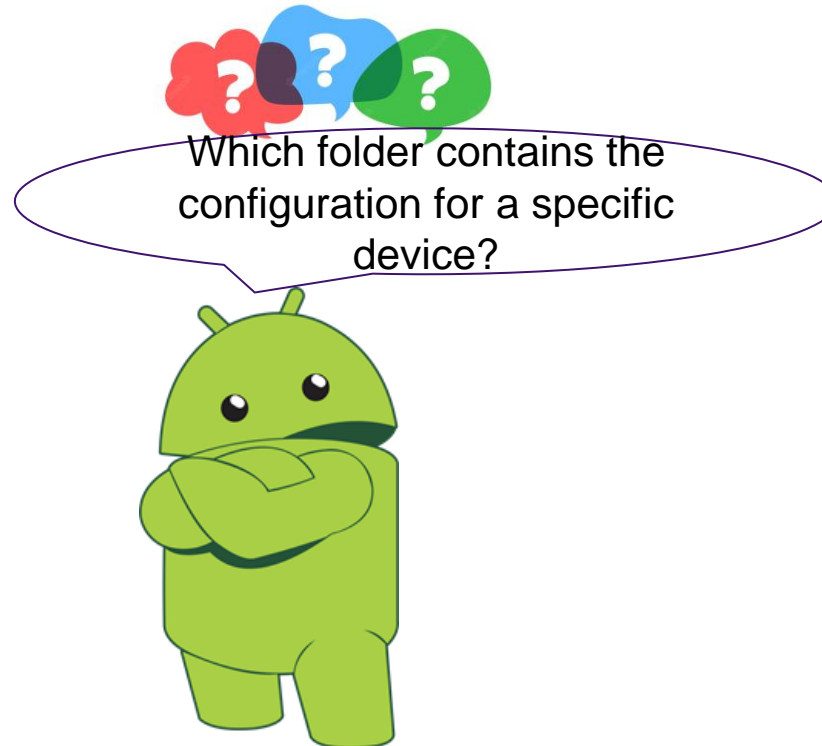
QUIZ (4/10)



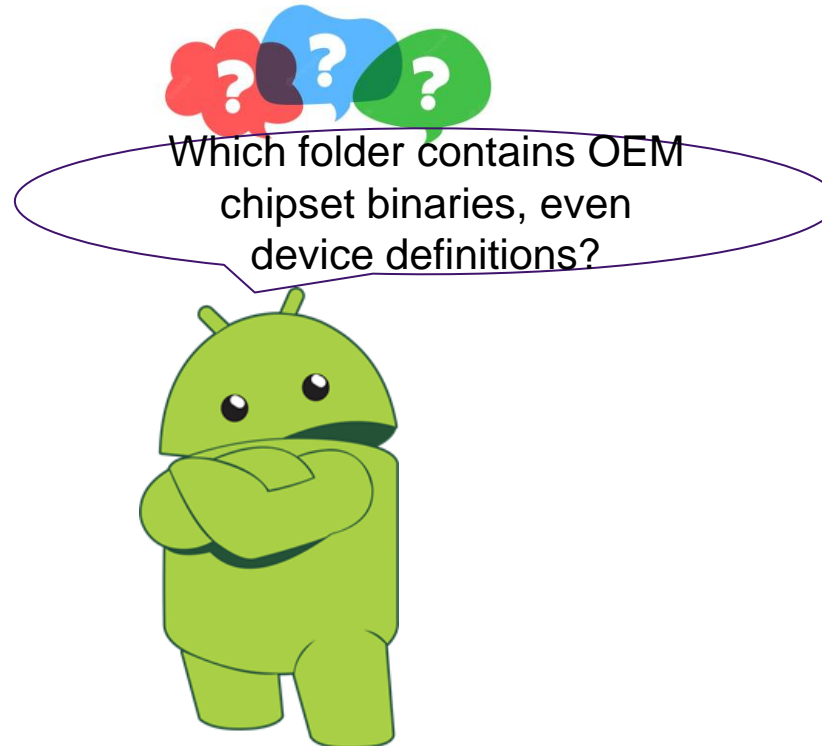
QUIZ (5/10)



QUIZ (6/10)



QUIZ (7/10)

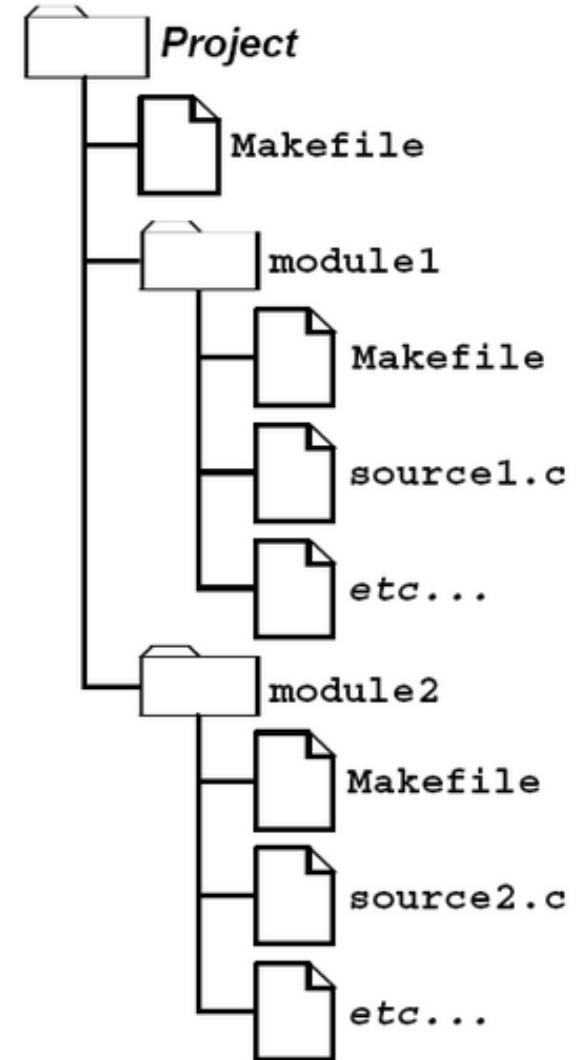


Group working



Traditional build system

- Traditionally, build systems rely on recursive *makefiles*.
- This refers to the use of a hierarchy of directories containing source files for the modules which make up the project, where each of the sub-directories contains a Makefile which describes the rules and instructions for the make program.
- The complete project build is done by arranging for the top-level Makefile to change directory into each of the sub-directories and recursively invoke make.

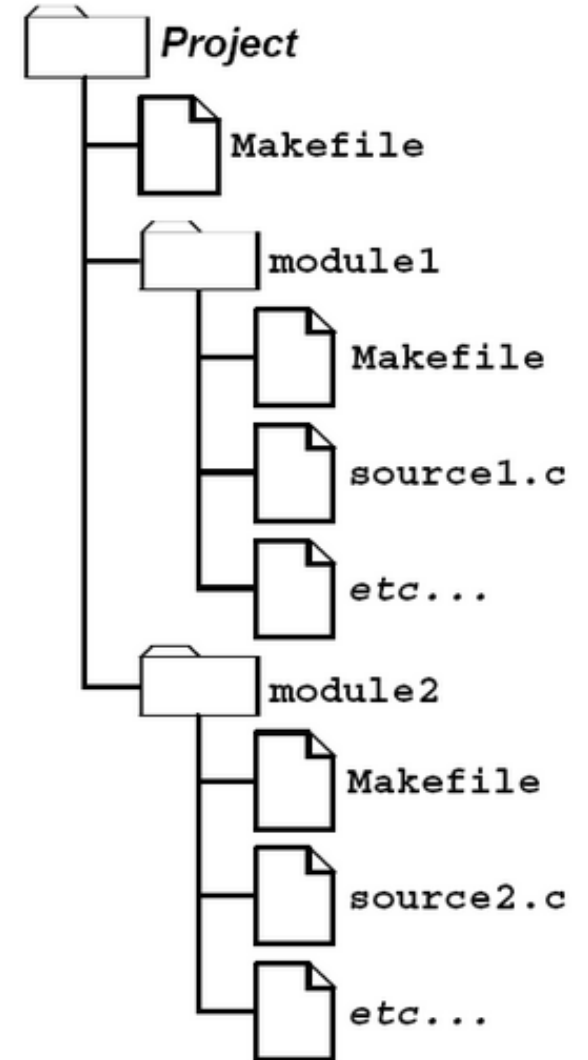


Group working



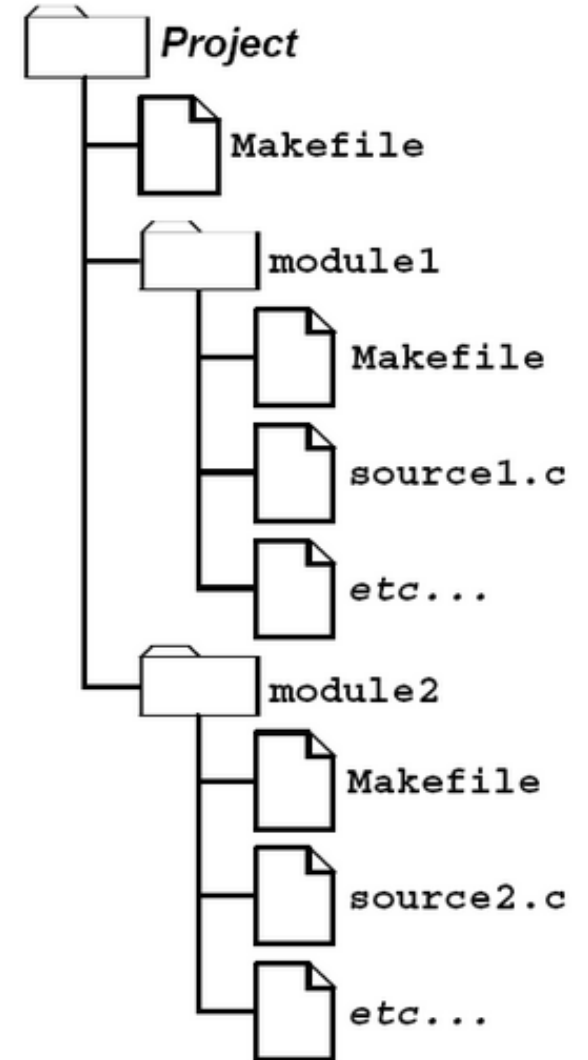
Traditional build system

- Normally, it is working fine. However, there are numerous problems with recursive make, and they are usually observed daily in practice:
 - It is very hard to get the order of the recursion into the subdirectories correct. This order is very unstable and frequently needs to be manually “tweaked.” Increasing the number of directories, or increasing the depth in the directory tree, cause this order to be increasingly unstable.
 - It is often necessary to do more than one pass over the subdirectories to build the whole system. This, naturally, leads to extended build times.



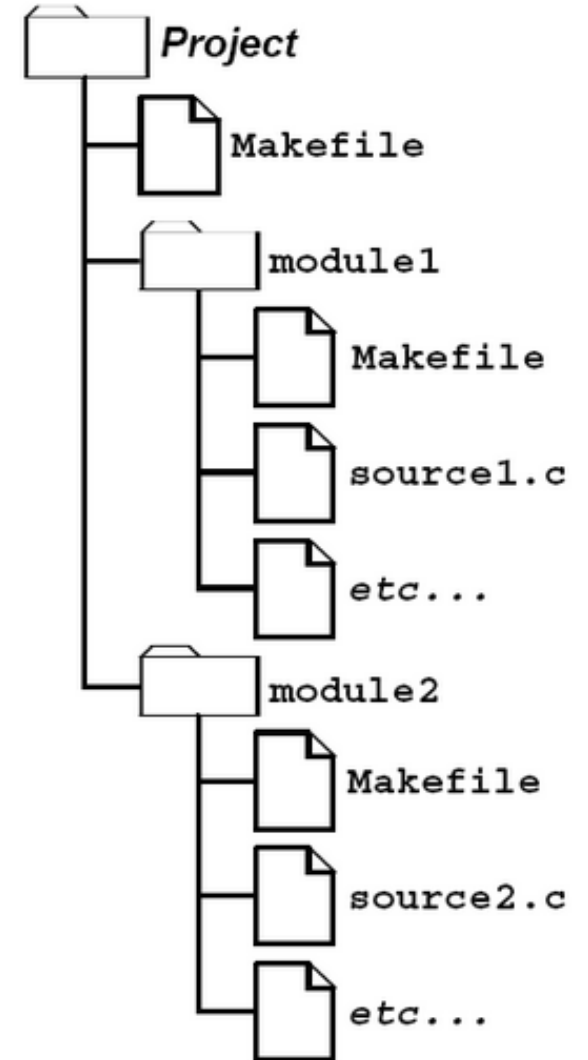
Traditional build system

- Normally, it is working fine. However, there are numerous problems with recursive make, and they are usually observed daily in practice:
 - Because the builds take so long, some dependency information is omitted, otherwise development builds take unreasonable lengths of time, and the developers are unproductive. This usually leads to things not being updated when they need to be, requiring frequent “clean” builds from scratch, to ensure everything has actually been built.
 - Because inter-directory dependencies are either omitted or too hard to express, the Makefiles are often written to build too much to ensure that nothing is left out.



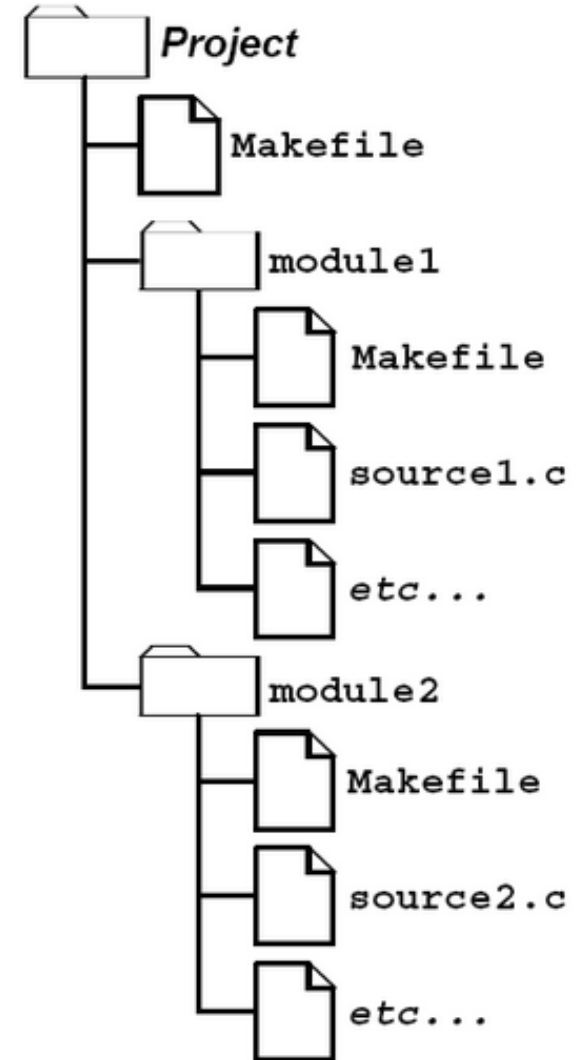
Traditional build system

- Normally, it is working fine. However, there are numerous problems with recursive make, and they are usually observed daily in practice:
 - The inaccuracy of the dependencies, or the simple lack of dependencies, can result in a product which is incapable of building cleanly, requiring the build process to be carefully watched by a human.
 - Related to the above, some projects are incapable of taking advantage of various “parallel make” implementations, because the build does patently silly things.

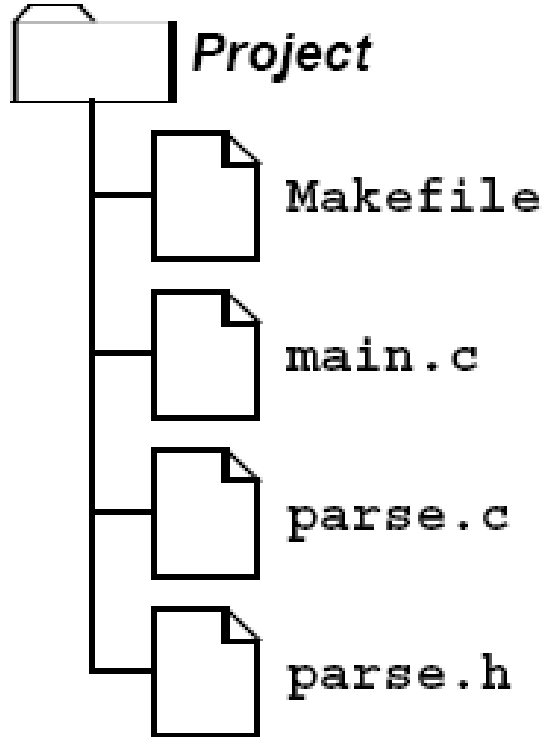


Traditional build system

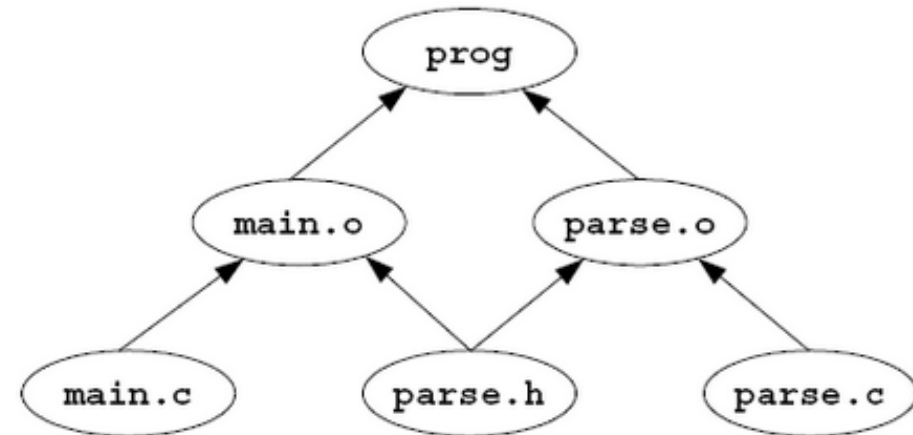
- Note:
 - Not all projects experience all of these problems.
 - The problem is not in make at all, but rather in the input given to make – the way make is being used.



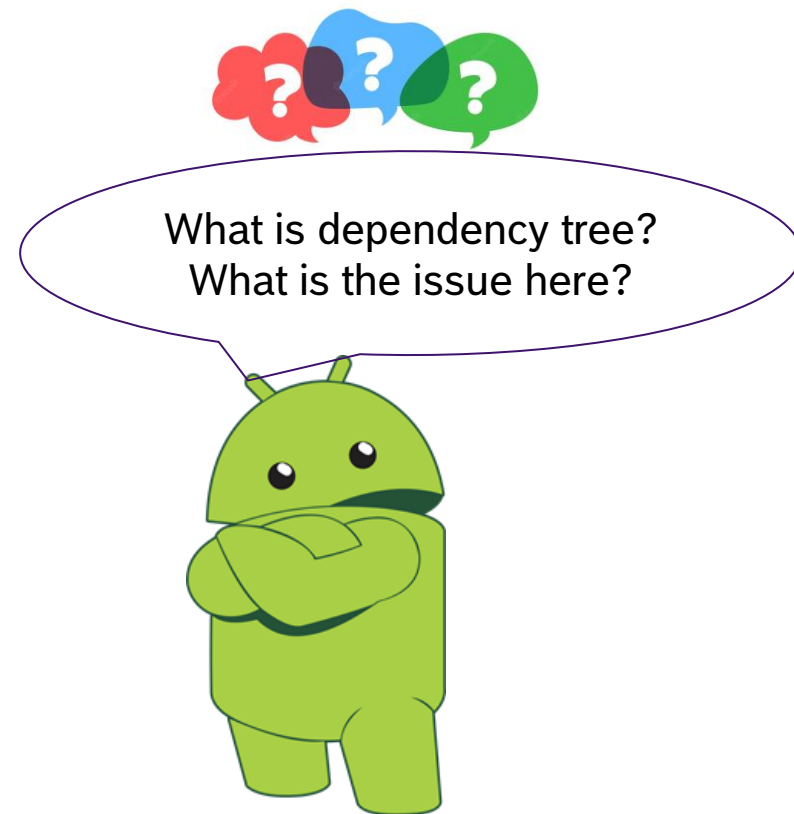
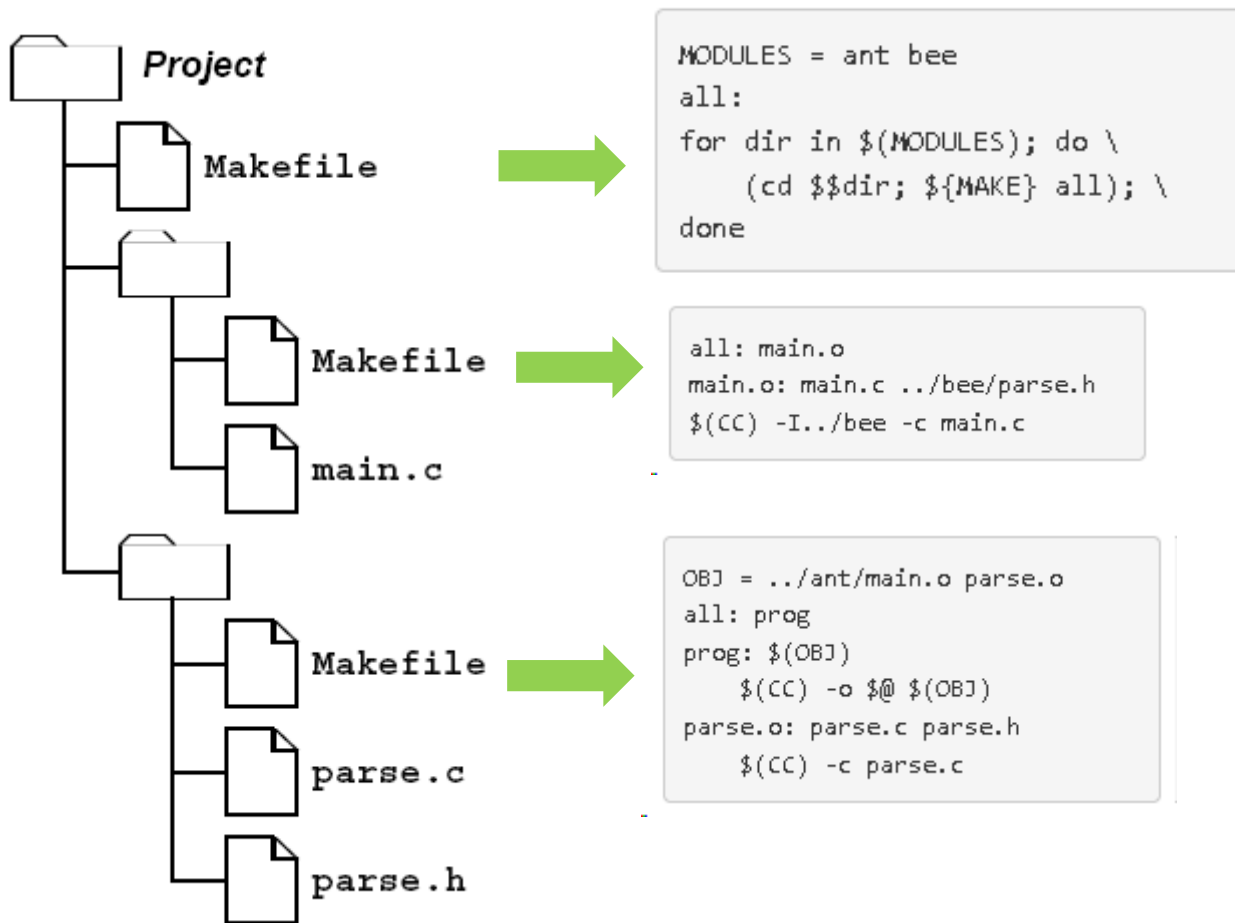
Traditional build system



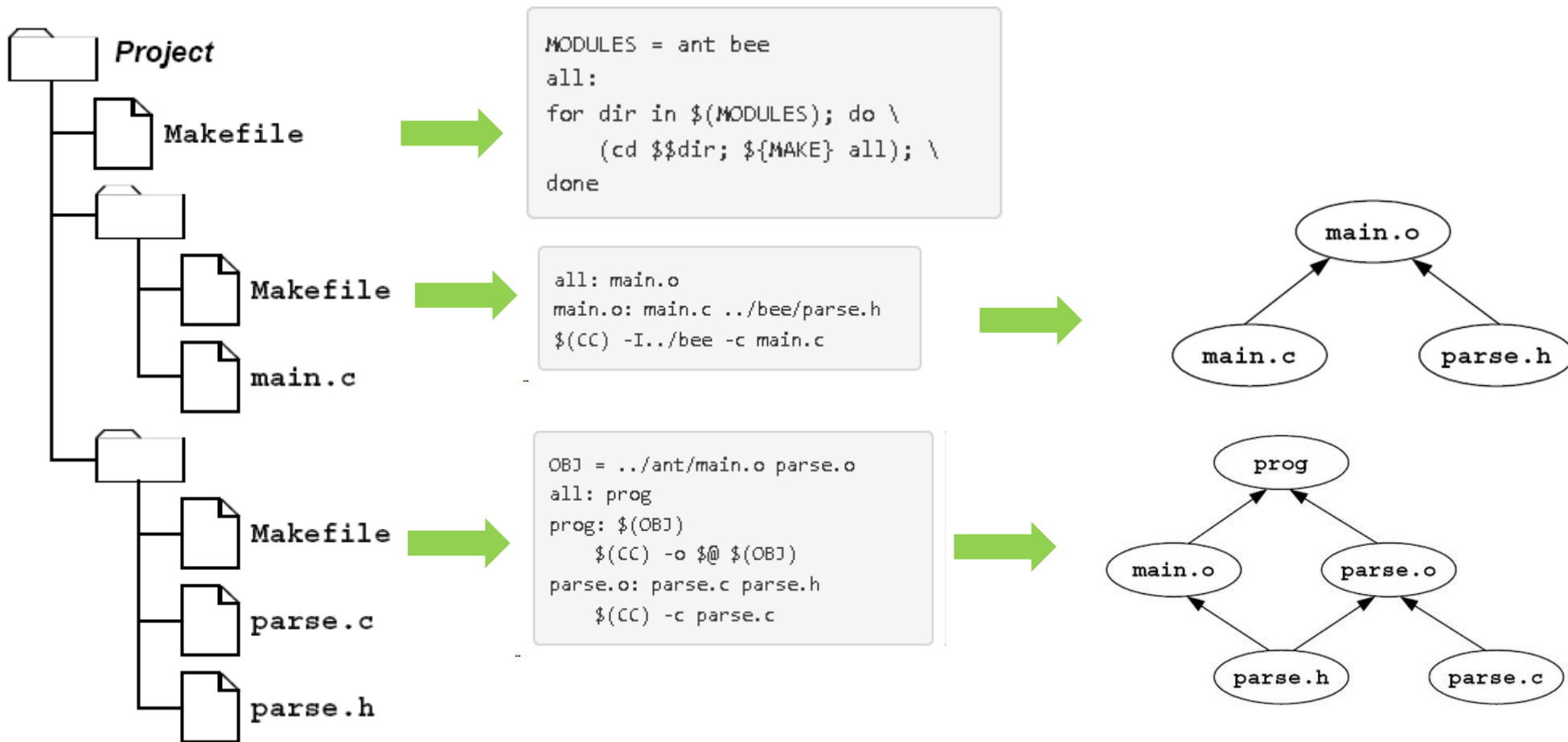
```
OBJ = main.o parse.o
prog: $(OBJ)
    $(CC) -o $@ $(OBJ)
main.o: main.c parse.h
    $(CC) -c main.c
parse.o: parse.c parse.h
    $(CC) -c parse.c
```



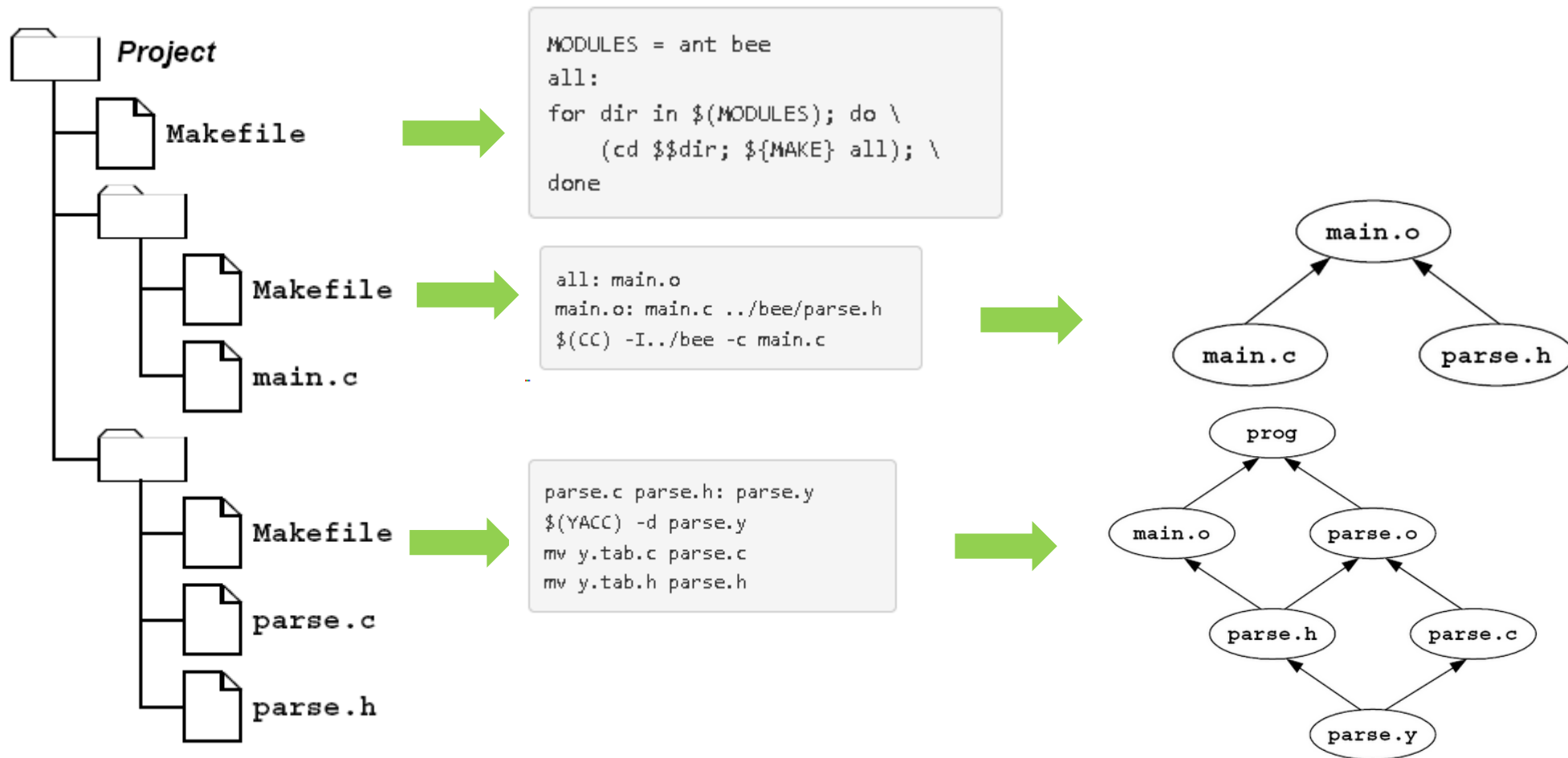
Traditional build system



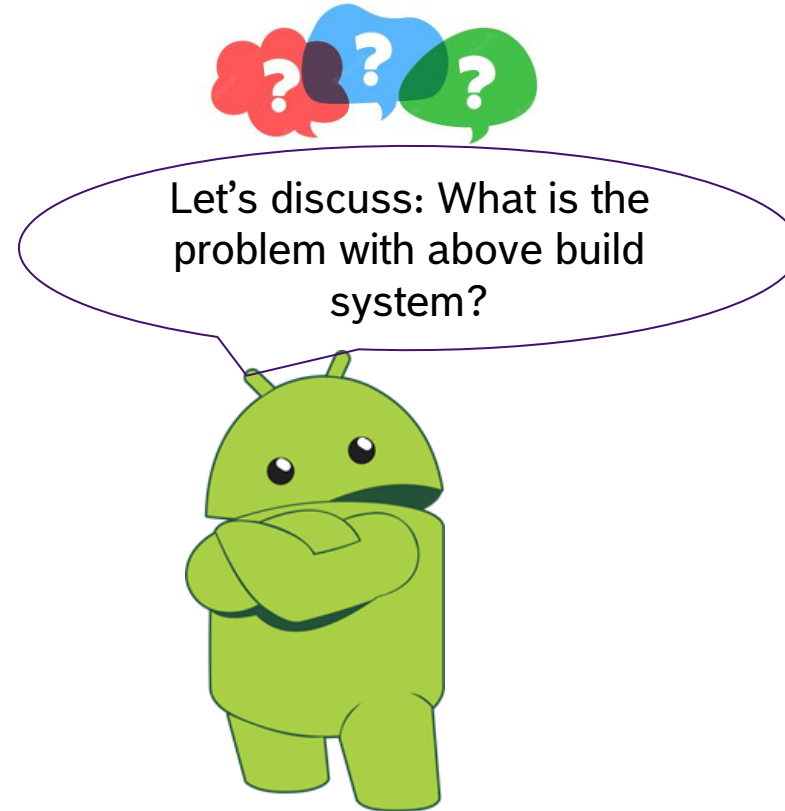
Traditional build system



Traditional build system



Group working



Android build system

- Unlike most make-based build systems, the Android build system **doesn't rely on recursive makefiles**.
- Instead, there is a script that **explores all directories and subdirectories** until it **finds an Android.mk or Android.bp** file.
- At that time, it **stops and doesn't explore the subdirectories underneath** that file's location—unless the **Android.mk or Android.bp** found **instructs the build system**.


Android build system

- Note:
 - Android doesn't rely on makefiles called Makefile. Instead, it's the Android.mk/Android.bp files that specify how the local “module” is built.
 - Android build “modules” have nothing to do with kernel “modules.”
 - a “module” is any component of the AOSP that needs to be built.
 - This might be a binary, an app package, a library, etc., and it might have to be built for the target or the host, but it's still a “module” with regards to the build system.

```
rag2hc@HC1VM0SD5:~/00_working$ find . -name Android.mk | wc -l
1179
rag2hc@HC1VM0SD5:~/00_working$ find . -name Android.bp | wc -l
7777
rag2hc@HC1VM0SD5:~/00_working$ |
```

Android build system

- Note:
 - Android doesn't use kernel-style menu-config or GNU autotools (i.e., autoconf, automake, etc.).
 - Instead, Android relies on a set of variables that are either set dynamically as part of the shell's environment by way of **envsetup.sh** and **lunch** or are defined statically ahead of time in a **buildspec.mk** file.
 - We can specify the properties of the target for which we want the AOSP to be built and, to a certain extent, which apps should be included by default in the resulting AOSP
 - However, there is **no way for us to enable or disable most features**, as in menu-config. For example, we don't want power management support or that you don't want the Location Service to start by default.



Thank for your listening!