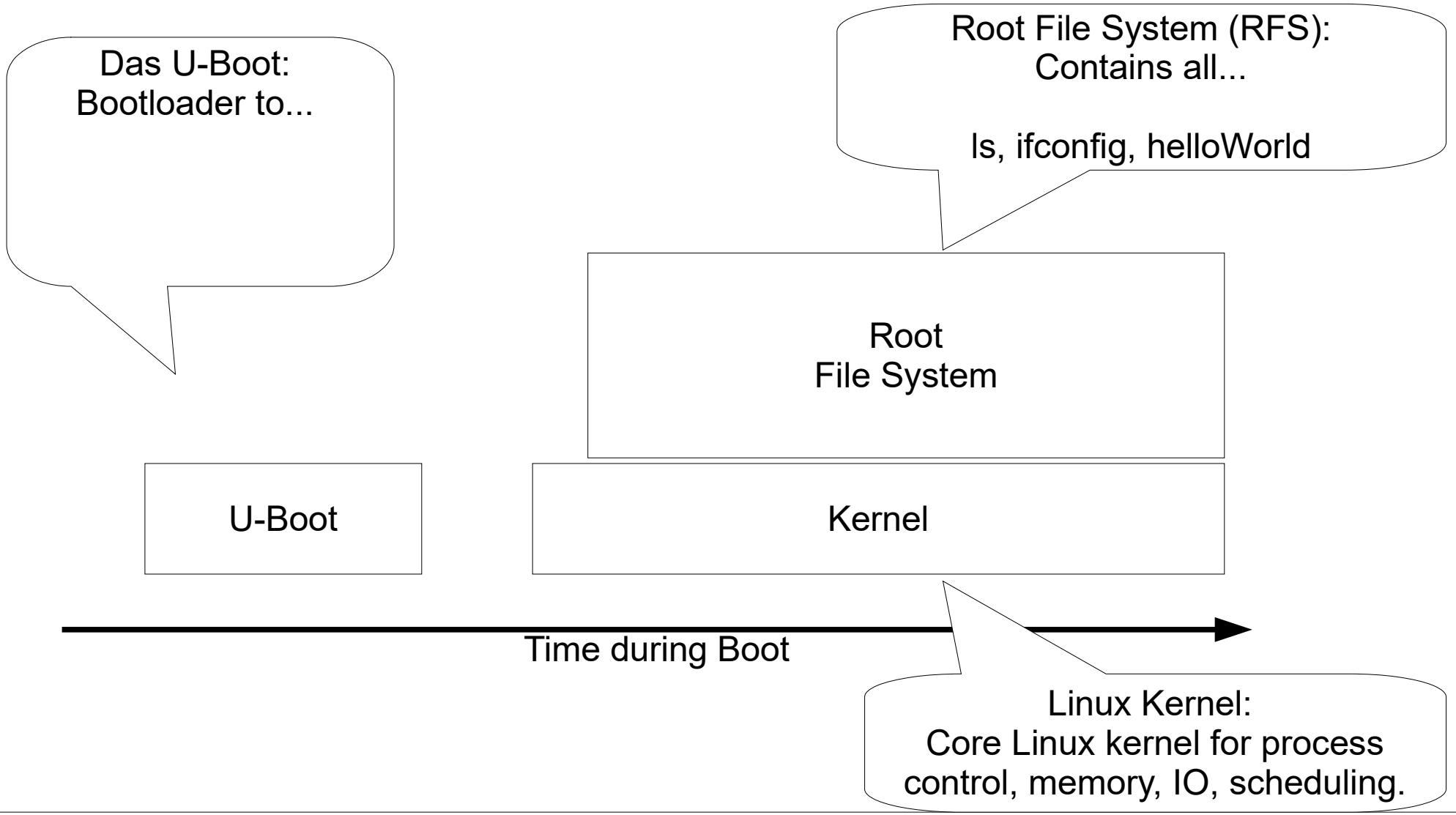
Launching & Building Embedded Software

U-Boot,
Cross Compiling,
Make, CMake
& Editors

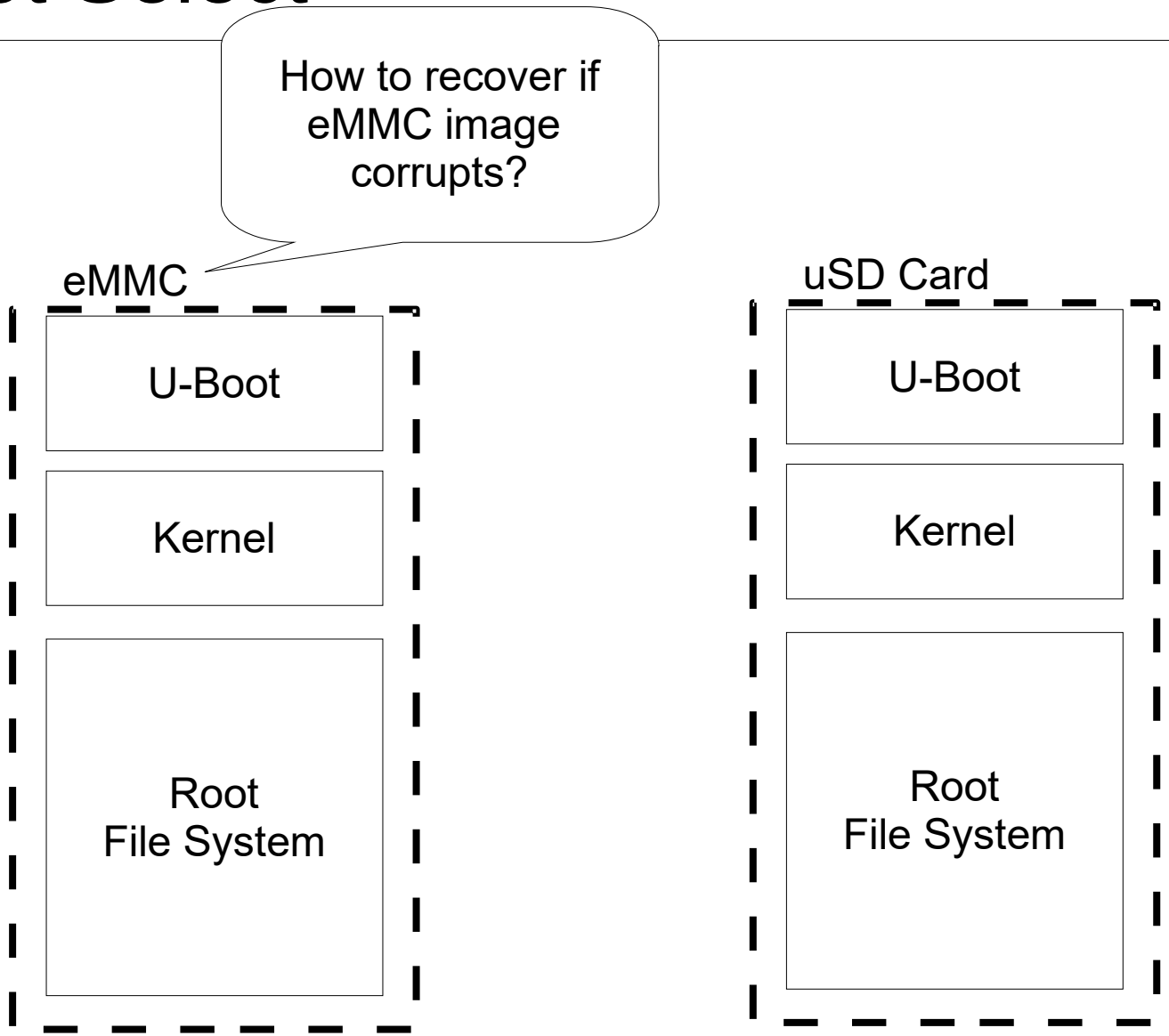
Topics

1) What software components run on the board?
2) How can we build our software?
3) How can we edit files via just text console?

Software Components



Boot Select


How to recover if eMMC image corrupts?
eMMC
U-Boot
Kernel
Root File System
uSD Card
U-Boot
Kernel
Root File System

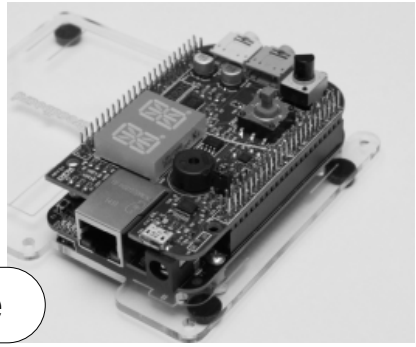
Servers & Directories

- Work (private) Directory
  - Ex: .c, .h, filelists.txt, makefile
- Public Directory
  - Holds files to...
  - Unprotected by passwords!
    Only for compiled code.

$HOME/cmpt433/work

$HOME/cmpt433/public

Host



/mnt/remote

Target

Cross-compile demo

- Compile on host for target

  (host)$ arm-linux-gnueabihf-gcc hello.c -o hello

- Check compiled file

  (host)$ readelf -h hello

- Run on board via NFS (one line each)

  (bbg)$ busybox mount -o tcp -t nfs -o nolock \
  192.168.7.1:/home/brian/cmpt433/public \
  /mnt/remote

  (bbg)$ cd /mnt/remote/
  (bbg)$ ./hello

Building Software With

Make & CMake

Makefile Basics

- Makefiles are

  ..

  - Name your script Makefile
  - Build a specific make-target with:..

    ```
    (host)$
    ```
  - Build default make-target with:

    ```
    (host)$ make
    ```
- Examples

  ```
  (host)$ make clean
  (host)$ make all
  ```

Simple Makefile

```
# Simple Makefile for building Hello world!

CC_C = arm-linux-gnueabihf-gcc
CFLAGS = -Wall -g -std=c11 -D _POSIX_C_SOURCE=200809L -Werror

app:
	$(CC_C) $(CFLAGS) helloWorld.c -o hello
	cp hello ~/cmpt433/public/myapps/
   ...

clean:
	rm hello
```

Define custom variables for later use.

Targets of form targetName:

Command(s) for this target.

clean a common target to remove all build files.

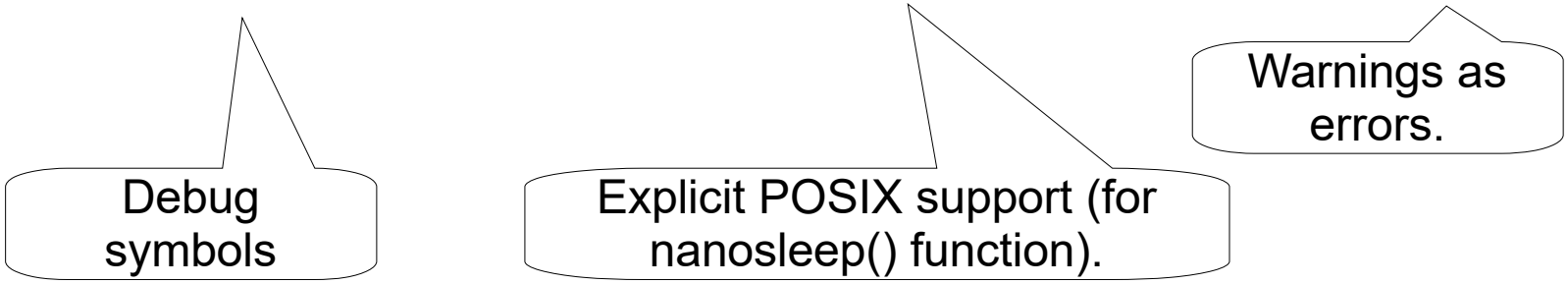
More Makefile

```
OUTFILE = helloWorld
OUTDIR = $(HOME)/cmpt433/public/myApps

CROSS_COMPILE = arm-linux-gnueabihf-
CC_C = $(CROSS_COMPILE)gcc
CFLAGS = -Wall -g -std=c11 -D _POSIX_C_SOURCE=200809L -Werror

help:
	@echo "Build Hello World program for BeagleBone"
	@echo "Targets include all, app, and clean."

all: app nestedDir done

app:
	$(CC_C) $(CFLAGS) helloWorld.c -o $(OUTDIR)/$(OUTFILE)
	ls -l $(OUTDIR)/$(OUTFILE)

nestedDir:
	make --directory=myNestedFolder

done:
	@echo "Finished building application."

clean:
	rm $(OUTDIR)/$(OUTFILE)
```

Setup output info once, used twice.

Compiler Flags

```
OUTFILE = factorial
OUTDIR = $(HOME)/cmpt433/public/myApps

CROSS_COMPILE = arm-linux-gnueabihf-
CC_C = $(CROSS_COMPILE)gcc
CFLAGS = -Wall -g -std=c11 -D _POSIX_C_SOURCE=200809L -Werror
  ..
```



..... rest of makefile omitted...

CMake

- CMake =..
  - Manage software build process
    ..
  - Supports intelligently recompiling only the files that changed
  - CMake Scripts:
    Describe the build process: CMakeLists.txt

    Can have multiple scripts:
    one to build each part, one to combine, etc.
- CMake is a Meta Build System
  1) CMake processes CMakeLists.txt files to..
  2) Use GNU Make to build the software using those Makefiles

Anatomy of CMakeLists.txt

CMakeLists.txt

```
# Minimum version. Run on the host.
cmake_minimum_required(VERSION 3.18)

# Project info
project(
    SimpleCMakePrj
    VERSION 1.0
    DESCRIPTION "Simple demo of CMake"
    LANGUAGES C
)

# Compiler options
set(CMAKE_C_STANDARD 11)
add_compile_options(-Wall -Werror -Wpedantic -Wextra)

add_executable( simple_cmake
    src/main.c src/funstuff.c
)
```

Required Elements

Lowest CMake version that will build our system (on host).

Many commands take key-value pair: VERSION 2.80

Info about project: name, version, necessary compilers, etc.

Generate this executable (1st arg) using these source files

Running CMake - Terminal

- Regenerate build/ folder and makefiles:
  `(host)$` **`cmake -S . -B build`**
- Build (compile & link) the project
  `(host)$` **`cmake --build build/`**
- Clean up temporary build folder (when needed)
  `(host)$` **`rm -rf build/`**

```
brian@PC-debian:~/all-my-code/CMPT433-Code/04-Building$ cmake -S . -B build
-- The C compiler identification is GNU 10.2.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/brian/all-my-code/CMPT433-Code/04-Building/build
brian@PC-debian:~/all-my-code/CMPT433-Code/04-Building$ cmake --build build/
Scanning dependencies of target simple_cmake
[ 33%] Building C object CMakeFiles/simple_cmake.dir/src/main.c.o
[ 66%] Building C object CMakeFiles/simple_cmake.dir/src/funstuff.c.o
[100%] Linking C executable simple_cmake
[100%] Built target simple_cmake
brian@PC-debian:~/all-my-code/CMPT433-Code/04-Building$ ls build/simple_cmake
build/simple_cmake
brian@PC-debian:~/all-my-code/CMPT433-Code/04-Building$ rm -rf build
```

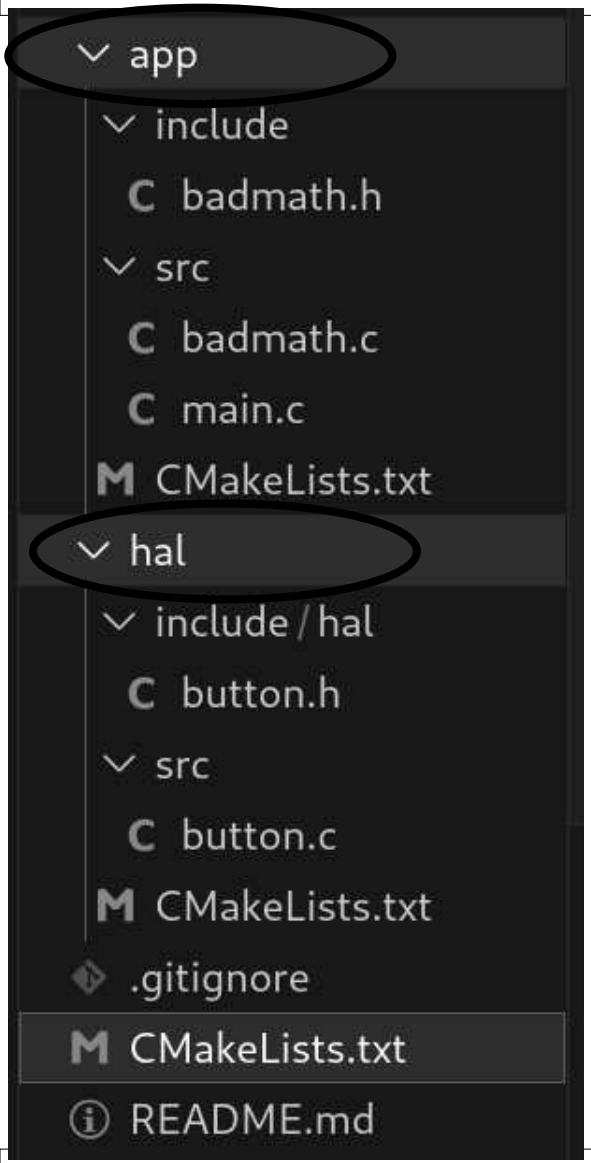
Running CMake - VS Code's Addon

- CMake Tool addon loaded with project with a CMakeLists.txt
- Select a Toolchain via different..
  "A kit encompasses project-agnostic and configuration-agnostic information about how to build code." [1]
  - Specifies compiler toolchain and version
  - We'll have one for native, one for cross-compile
    (Use "unspecified" to build natively)
  - Addon scans host system for available toolchains
- Building
  - Generate then run makefiles:

  - Run makefiles: Ctrl + Shift + B
    Terminal > Configure Default Build Task... > CMake:Build

1. https://vector-of-bool.github.io/docs/vscode-cmake-tools/kits.html

CMake Starter Project



- hal/ ..
  - Low-level *modules* with hardware specific details.
- app/ ..
  - Organized into *modules* for better organization and encapsulation
- build/
  - Created by CMake; *temporary*
- 3 CMakeLists.txt
  - One in root to control full build
  - One in each of hal/ and app/

Nano

- Nano is a somewhat easier to use text editor.
  $ nano myfileToEdit.txt
  - Just type and edit text as you might expect.
- Commands
  - : Displays help. Ctrl+x to quit help.
  - : Quit, asks you if you want to save.

Simple create/view a file

- Redirect text to a file

  $ echo “Overwrite file with text” test.txt
  $ “Adding this to end of file” test.txt

- View a file

  $ cat daFile
  concatenate the file, outputs to stdout (terminal)

  $ less daLongFile
  shows page-by-page view of long file

  $ tail -20 daLongFile
  Shows last 20 lines of the file.

- Pipe output from one tool to another

  $ dmesg
  displays kernel messages

  - $ dmesg | less
    $ dmesg | tail -20

Summary

- Boot sequence
  - UBoot --> Kernel --> Root File System
- Makefiles automate building software.
  - Create targets for different products/actions.
- CMake: cross-platfrom meta build system
  - Process defined in CMakeLists.txt
- Text-based Editors
  - Nano