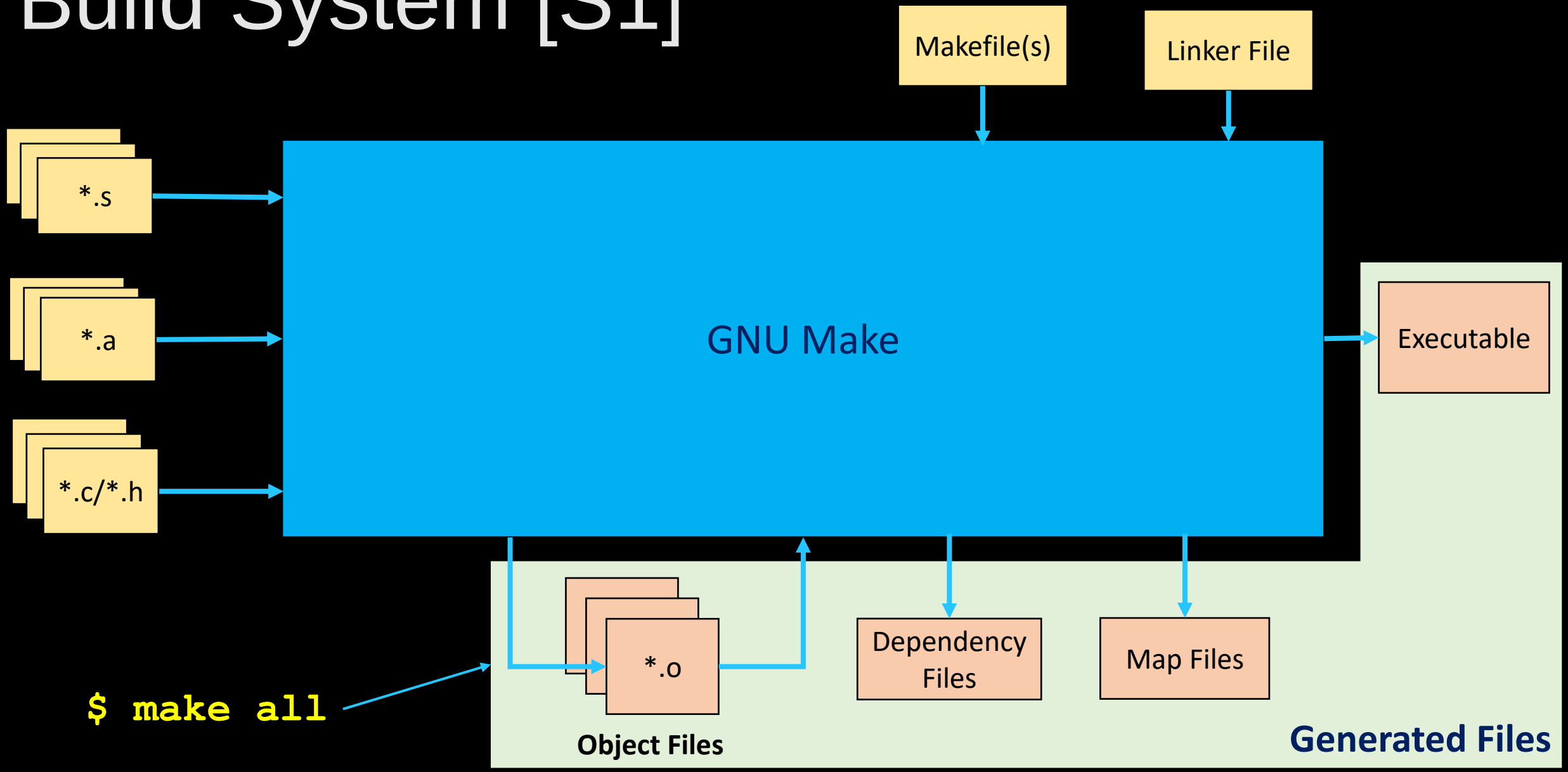
Embedded Software Essentials

Makefiles Part 1

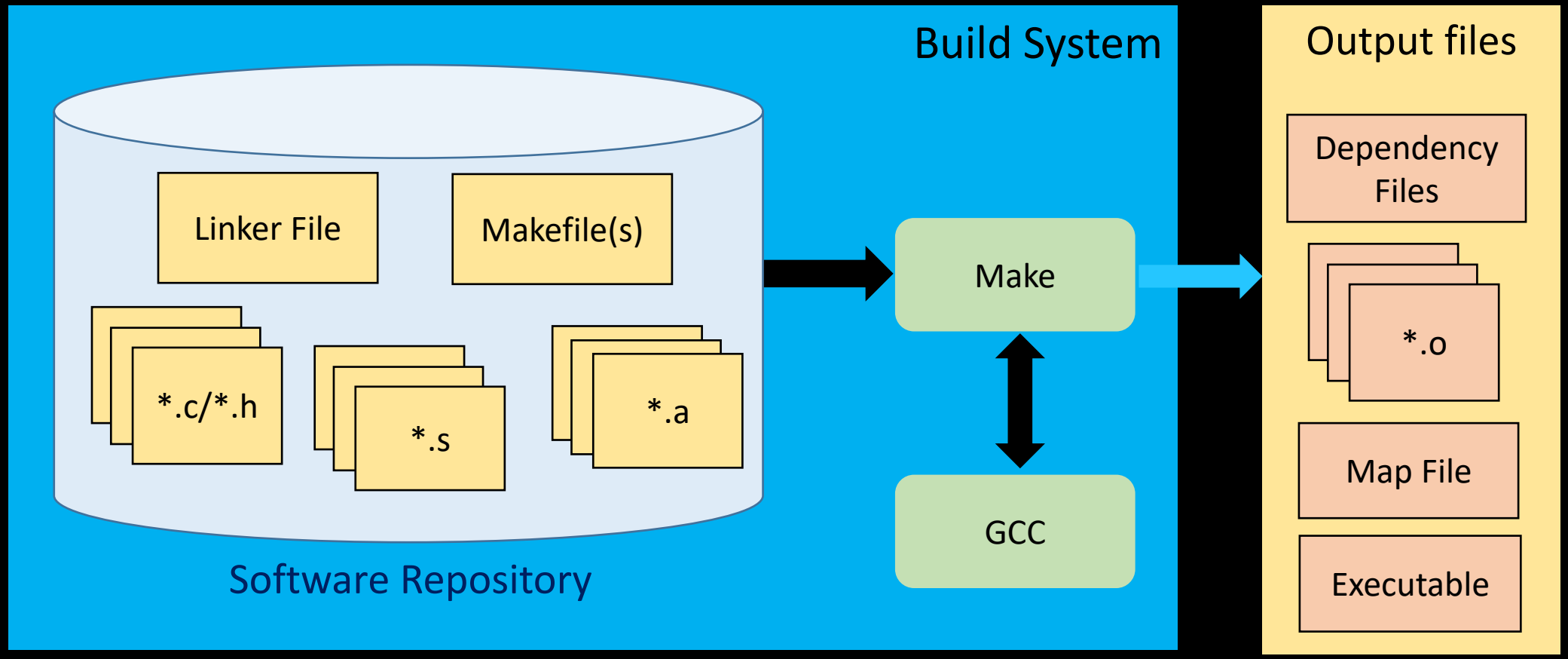
C1 M2 V7

Copyright

Build System [S1]



Version Controlled Build System [S2]



Makefiles [S3]

- One or more files used to tell **make** how to build a particular project

```
makefile, Makefile, sources.mk, includes.mk, …
```

- Makefiles have build **targets** or build **rules**

```
$ make all    $ make clean    $ make main.out    $ make main.o
```

- Targets can have **dependencies** or **prerequisites**

```
main.out: main.o my_file.o
        gcc -g -Wl,-Map=main.map -o main.out main.o my_file.o
```

Makefiles – Rules [S4]

- Build **rules** require a specific syntax of **target : prerequisites**, and **commands**
- These are **recipes** for how to build a particular executable or non-source file
- A recipe can only be executed if the **dependencies** are met

- Simple my_file.o build rule: Depends on my_file.c and my_file.h

```
$ make my_file.o
```

Generates the **my_file.o** target by executing the following rule

```
my_file.o: my_file.c my_file.h
        gcc -c my_file.c -o my_file.o
```

Makefile Syntax [S5]

- Comments start with a #
- Can include other makefiles
- Line continuation is done with a \
- Can create and use variables
- Can have multiple rules
- Command lines start with a tab
- Targets can depend on other targets

Makefile

```
# This is a comment

# This includes another file
include sources.mk

# Variable & Line Continuance
FLAGS = -g      \
        -Werror \
        -std=c99

# my_file.o target binary
my_file.o: my_file.h my_file.c
    gcc $(FLAGS) -c -o my_file.o my_file.c

# main.o target binary
main.o: my_file.h
    gcc $(FLAGS) -c -o main.o main.c

# Main Target Executable
main.out: main.o my_file.o
    gcc -o main.out main.o my_file.o
```

Makefile Syntax

- **Comments start with a #**
- Can include other makefiles
- Line continuation is done with a \
- Can create and use variables
- Can have multiple rules
- Command lines start with a tab
- Targets can depend on other targets

Makefile

```
# This is a comment

# This includes another file
include sources.mk

# Variable & Line Continuance
FLAGS = -g      \
        -Werror \
        -std=c99

# my_file.o target binary
my_file.o: my_file.h my_file.c
    gcc $(FLAGS) -c -o my_file.o my_file.c

# main.o target binary
main.o: my_file.h
    gcc $(FLAGS) -c -o main.o main.c

# Main Target Executable
main.out: main.o my_file.o
    gcc -o main.out main.o my_file.o
```

Makefile Syntax

- Comments start with a #
- **Can include other makefiles**
- Line continuation is done with a \
- Can create and use variables
- Can have multiple rules
- Command lines start with a tab
- Targets can depend on other targets

Makefile

```
# This is a comment

# This includes another file
include sources.mk

# Variable & Line Continuance
FLAGS = -g      \
        -Werror \
        -std=c99

# my_file.o target binary
my_file.o: my_file.h my_file.c
    gcc $(FLAGS) -c -o my_file.o my_file.c

# main.o target binary
main.o: my_file.h
    gcc $(FLAGS) -c -o main.o main.c

# Main Target Executable
main.out: main.o my_file.o
    gcc -o main.out main.o my_file.o
```

Makefile Syntax

- Comments start with a #
- Can include other makefiles
- **Line continuation is done with a **
- Can create and use variables
- Can have multiple rules
- Command lines start with a tab
- Targets can depend on other targets

Makefile

```
# This is a comment

# This includes another file
include sources.mk

# Variable & Line Continuance
FLAGS = -g      \
        -Werror \
        -std=c99

# my_file.o target binary
my_file.o: my_file.h my_file.c
    gcc $(FLAGS) -c -o my_file.o my_file.c

# main.o target binary
main.o: my_file.h
    gcc $(FLAGS) -c -o main.o main.c

# Main Target Executable
main.out: main.o my_file.o
    gcc -o main.out main.o my_file.o
```

Makefile Syntax

- Comments start with a #
- Can include other makefiles
- Line continuation is done with a \
- **Can create and use variables**
- Can have multiple rules
- Command lines start with a tab
- Targets can depend on other targets

Makefile

```
# This is a comment

# This includes another file
include sources.mk

# Variable & Line Continuance
FLAGS = -g      \
        -Werror \
        -std=c99

# my_file.o target binary
my_file.o: my_file.h my_file.c
    gcc $(FLAGS) -c -o my_file.o my_file.c

# main.o target binary
main.o: my_file.h
    gcc $(FLAGS) -c -o main.o main.c

# Main Target Executable
main.out: main.o my_file.o
    gcc -o main.out main.o my_file.o
```

Makefile Syntax

- Comments start with a #
- Can include other makefiles
- Line continuation is done with a \
- Can create and use variables
- **Can have multiple rules**
- Command lines start with a tab
- Targets can depend on other targets

Makefile

```
# This is a comment

# This includes another file
include sources.mk

# Variable & Line Continuance
FLAGS = -g      \
        -Werror \
        -std=c99

# my_file.o target binary
my_file.o: my_file.h my_file.c
    gcc $(FLAGS) -c -o my_file.o my_file.c

# main.o target binary
main.o: my_file.h
    gcc $(FLAGS) -c -o main.o main.c

# Main Target Executable
main.out: main.o my_file.o
    gcc -o main.out main.o my_file.o
```

Makefile Syntax

- Comments start with a #
- Can include other makefiles
- Line continuation is done with a \
- Can create and use variables
- Can have multiple rules
- **Command lines start with a tab**
- Targets can depend on other targets

Makefile

```
# This is a comment

# This includes another file
include sources.mk

# Variable & Line Continuance
FLAGS = -g      \
        -Werror \
        -std=c99

# my_file.o target binary
my_file.o: my_file.h my_file.c
	gcc $(FLAGS) -c -o my_file.o my_file.c

# main.o target binary
main.o: my_file.h
	gcc $(FLAGS) -c -o main.o main.c

# Main Target Executable
main.out: main.o my_file.o
	gcc -o main.out main.o my_file.o
```

Makefile Syntax

- Comments start with a #
- Can include other makefiles
- Line continuation is done with a \
- Can create and use variables
- Can have multiple rules
- Command lines start with a tab
- **Targets can depend on other targets**

Makefile

```
# This is a comment

# This includes another file
include sources.mk

# Variable & Line Continuance
FLAGS = -g      \
        -Werror \
        -std=c99

# my_file.o target binary
my_file.o: my_file.h my_file.c
    gcc $(FLAGS) -c -o my_file.o my_file.c

# main.o target binary
main.o: my_file.h
    gcc $(FLAGS) -c -o main.o main.c

# Main Target Executable
main.out: main.o my_file.o
    gcc -o main.out main.o my_file.o
```

Makefile Variables [S6]

- Variables can be set to strings of text and can include other variables
  - Variable access is done with the **$(***variable-name***)** syntax

Example Recursively Expanded Variables

```
CSTD=c89
CPU=cortex-m0plus
CC=arm-none-eabi-gcc
```

- Recursively Expanded Variables (=)
  - Expands **whenever** used
- Simply Expanded Variables (:=)
  - Expands **once** at the time of definition

Example Simply Expanded Variables

```
ARCH:=$(shell arch)
CWD:=$(shell pwd)
OS:=$(shell uname)
```

Special examples that run Linux commands to set data

- Use variables for things like
  - Compiler flags -> **CFLAGS**
  - Linker Flags -> **LDFLAGS**

C-Flags Example Variables

```
CFLAGS = -g -std=$(CSTD) -mcpu=$(CPU) -mthumb
```

Include Paths and Sources [S7]

- Can control what **directories** and **source files** are used for building
  - Includes provide path to code (absolute or relative paths)
  - Sources determine what needs to be built

```
INCLUDES=   \          SRCS=         \
  -I./libs  \            ./main.c    \
  -I./modem \            ./memory.c  \
  -I./uart  \            ./uart.c    \
  -I./arch               ./data.c
```

- Can reference a variable for include directories and sources files
  - Creates dynamic targets instead of statically defined targets

Building the Executable [S8]

- Use variables in your target rules

```
main.out: main.o my_file.o
    gcc -Wl,map=main.map -I./inc -o main.out main.o my_file.o


$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) $(INCLUDES) $(LDFLAGS) -o $(TARGET) $(OBJS)
```

- Automatic Variables - variables in a recipe with a scope

$@ - Target
$^ - All Prerequisites
$< - First Prerequisite

```
$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) $(LDFLAGS) $(INCLUDES) -o $@ $^
```