bw | HPC – C5

bwHPC course – Tutorial: Compiling, Makefile

Hartmut Häfner

Funding: Baden-Württemberg MINISTERIUM FÜR WISSENSCHAFT, FORSCHUNG UND KUNST

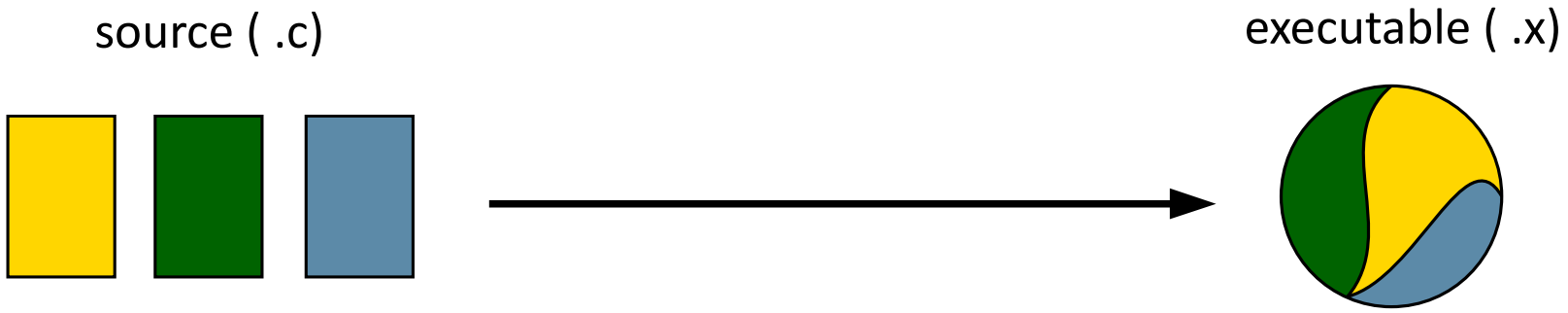
www.bwhpc-c5.de

Outline

- Compiler + Numerical Libraries
  - compiling
  - linking
- Makefile
  - Intro, Syntax (Explicit + Implicit Rules ...)

1. Compilation

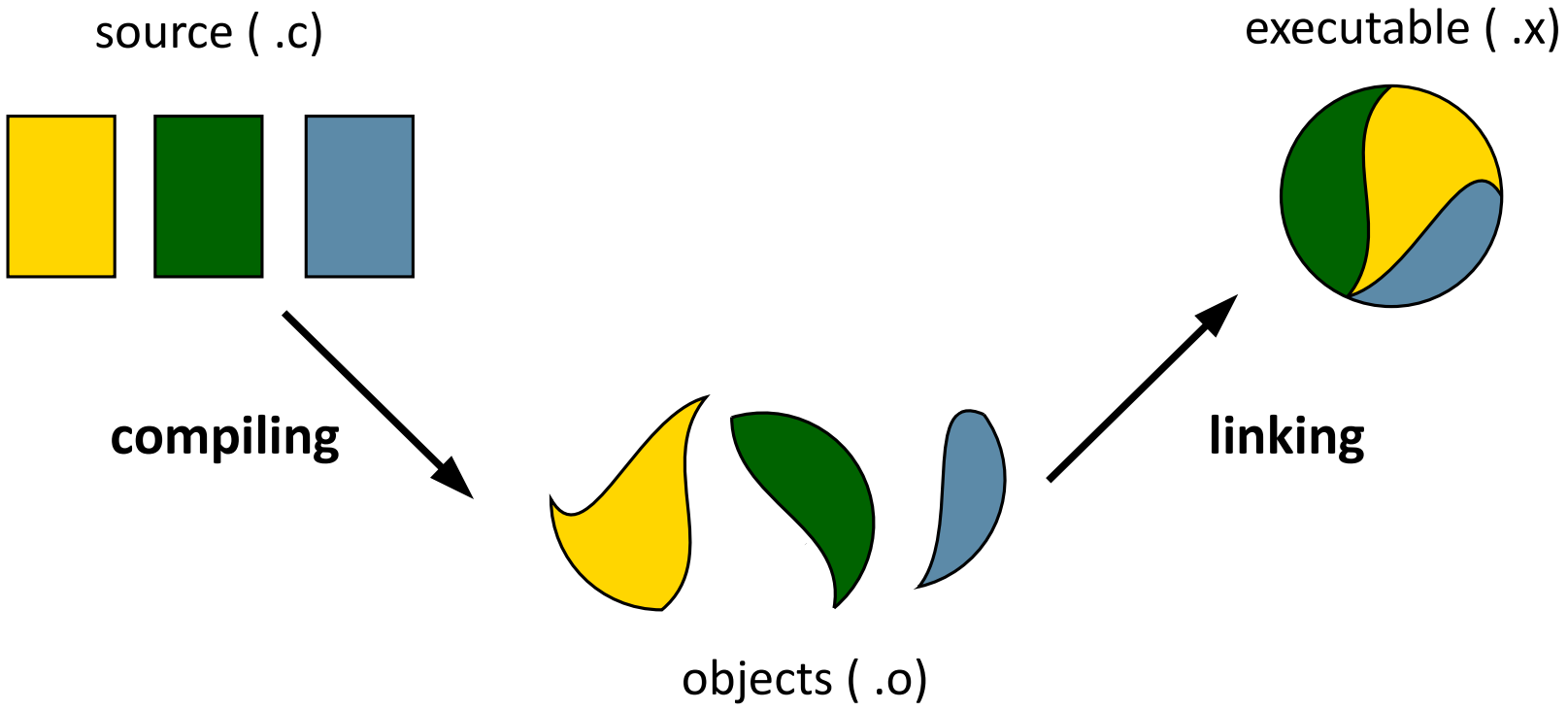
Object files



- Example:

```
$ gcc -o exec.x src1.c src2.c src3.c
$ ./exec.x
```

Object files




```
$ gcc -c src1.c; gcc -c src2.c; gcc -c src3.c
$ gcc -o exec.x src1.o src2.o src3.o
```

- Changes in a single file do not afford the compilation of all source code.

Include files

- Header files (.h)
  - Declaration of variables
  - Definition of static variables
  - Declaration of functions/subroutines
  - ..

- Example: include header file /home/myincs/header.h
  - Preprocessor directive in source code:

```
#include "header.h"
…
```
src1.c

'#' does **not** initiate command lines but preprocessor directives in C/C++ code!

  - Add header directory `-I<include_directory>`

```
$ gcc -I/home/myincs -c src1.c; gcc -c src2.c
$ gcc -o exec.x src1.o src2.o
$ ./exec.x
```

Example: Hello

Main Program

```
#include "hello.h"

int main(void){

  print_hello();

  return 0;
}
```

hello.c

Header (Declarations)

```
#ifndef _HELLO_H_
#define _HELLO_H_

int print_hello(void);

#endif
```

hello.h

Functions (Definitons)

```
#include <stdio.h>

int print_hello(void){

  printf(„hello!\n“);

  return 0;
}
```

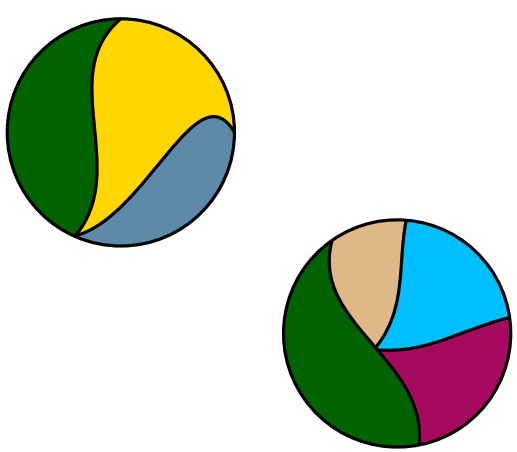
hello_fct.c

- Exercise: *hello*
  - Build objects hello.o hello_fct.o
  - Build executable by linking objects
  - `$ ./hello`

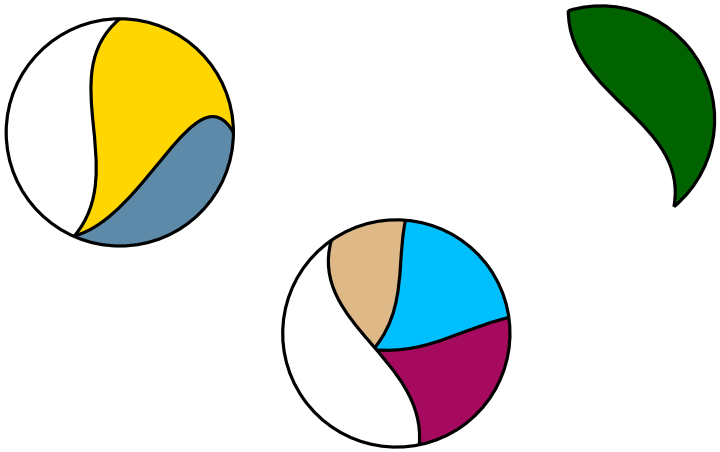
Shared object files and Libraries

- Objects can be used by different executables.
- A **library** contains program parts (subroutines, classes, type definitions, …) that can be used by different executables.

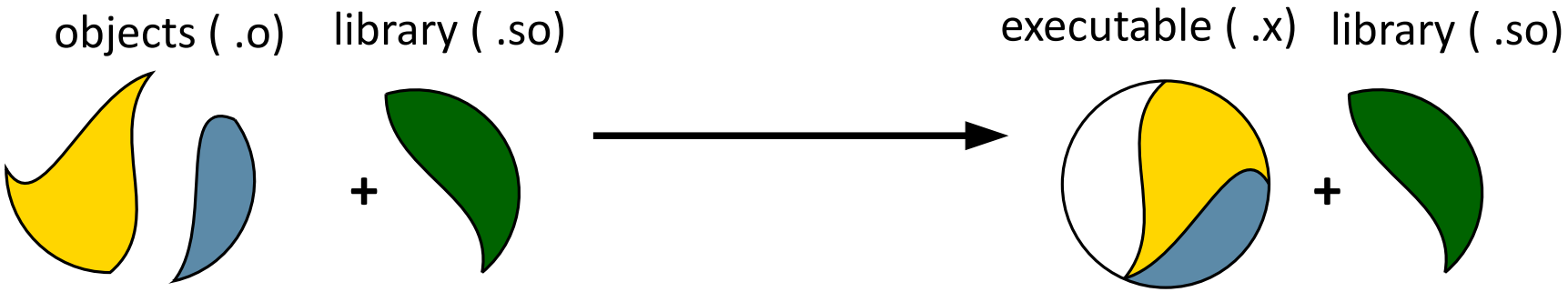
- Static library
  - Linked during building executable



- Shared library
  - Loaded during runtime



Shared Object files and Libraries



- Example: link library */home/mylibs/libexample.so*
  - Build executable:
    - Add library directory `-L<library_directory>`
    - load library `-l<library_name>` **after** refering source/object files

```
$ gcc -o exec.x src1.o src2.o -L/home/mylibs -lexample
```

  - Run executable:
    - Add `<library_directory>` to list of library directories `${LD_LIBRARY_PATH}`

```
$ export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/home/mylibs
$ ./exec.x
```

Module files

- Module files set/prepare following environment variables amongst others:
  - **`*_LIB_DIR = <library_directory>`**
  - **`*_INC_DIR = <include_directory>`**
  - **`${LD_LIBRARY_PATH}`**

- Show module file setup with **`$ module show <module_file>`**

- Example: link NETCDF library
  - Build executable:

```
$ module load compiler/intel
$ module load lib/netcdf
$ icc -I${NETCDF_INC_DIR} -c src1.c; gcc -c src2.c
$ icc -o exec.x src1.o src2.o -L${NETCDF_LIB_DIR} -lnetcdf
```

  - Run executable:

```
$ module load lib/netcdf
$ ./exec.x
```

2. Makefile

Motivation

- Interactively
  - `$ gcc -o hello -I. hello.c hello_fct.c`
  - Works as long as command history is active
- Shell script
  - `$ ./compile.sh`
  - Does always recompile the whole code
- Makefile
  - `$ make`
  - better organisation of code compilation
  - recompiles only updated files,

```
make: `hello' is up to date.
```

Makefile

- `$ make [-f <Makefile_name>] [<target>]`
- executes script named *Makefile* or *makefile*
  - without argument first rule in *Makefile* is executed

- Rule definition (format):

```
target: prerequisites
<TAB>command
```

Rule has to be applied, if any of these files is changed

To apply the rule, command has to be executed.

Only works with beginning tab stop!

```
hello: hello.h hello.c hello_fct.c
	gcc -o hello -I. hello.c hello_fct.c
```

Makefile.1

- Exercise: *Makefile.1*
  - define a second rule named `clean` to remove the executable

Rules - Content

- Explicit rules
  - `hello.o:` rule to build target *hello.o*
- Wildcards
  - `hello: *.c` *hello* depends on all files with suffix *.c* in this directory
- Pattern rules
  - `%.o:` rule for all files with suffix *.o*
  - `%.o: %.c` `%` in prerequisites substitutes the same as `%` in the target
- Phony Targets
  - `.PHONY: clean` target *`clean`* is nothing to build

```
clean:
```

Variables

- Variable assignment
  - `=` recursively expanded (referenced by reference)
  - `:=` simply expanded (referenced by value)
  - `?=` only if variable is not defined yet (no overwrite)
  - `+=` add item to variable array

```
CC      ?= gcc
CFLAGS  = -I.
INC     := hello.h
OBJ     := hello.o
OBJ     += hello_fct.o
EXE     := hello

${EXE}: ${INC} ${OBJ}
        ${CC} -o ${EXE} ${CFLAGS} ${OBJ}

.PHONY: clean
clean:
        rm -f ${OBJ} ${EXE}
```

Makefile.2

- Exercise: *Makefile.2*
  - write an appropriate rule for target hello.o

Automatic Variables

- Automatic variables change from rule to rule

  `$@` = target

  `$<` = first item of prerequisites

  `$^` = all items of prerequisites separated by ' '

- Exercise: *Makefile.3*
  - Use automatic variables in rule to build *hello*

```
CC      ?= gcc
CFLAGS  = -I.
INC     := hello.h
OBJ     := hello.o
OBJ     += hello_fct.o
EXE     := hello

%.o: %.c ${INC}
        ${CC} -o $@ ${CFLAGS} -c $<

hello: hello.o hello_fct.o
        ${CC} -o ${EXE} ${CFLAGS} ${OBJ}

.PHONY: clean
clean:
        rm -f ${OBJ} ${EXE}
```

Makefile.3

Directives

- Conditions can be expressed by directives
  - if VAR is (not) defined

```
ifdef/ifndef VAR
  ..
else
  ..
endif
```

  - if A and B are (not) equal

```
ifeq/ifneq (A,B)
  ..
else
  ..
endif
```

- Example:
  - Conditional assignment `CC ?= gcc` is equivalent to

```
ifndef CC
  CC = gcc
endif
```

Include

- Parts of *Makefile* can be outsourced
  - e.g. platform specific statements
- External makefile code, e.g. file *make.inc*, can be loaded in *Makefile* via

```
include make.inc
```

- Example: *hello*
  - *make.inc.gcc* and *make.inc.icc* contain compiler specific makefile statements
  - include *make.inc* depending on `${CC}`
  -
```
$ module load compiler/gnu
$ make -f Makefile.5
```
  -
```
$ module load compiler/intel
$ make -f Makefile.5
```

```
CC      = gcc
CFLAGS = -I. -O
```

make.inc.gcc

```
CC      = icc
CFLAGS = -I. -O
```

make.inc.icc

```
include make.inc.${¢¢}
...
hello: ${OBJ}
        ${CC} -o $@ ${CFLAGS} $<
```

Makefile.5

3. MPI-Parallelization

(only 1 slide)

MPI-Course and -Information

- MPI-course
  - 7/12/17 (one day) from 11.30 to 16.30 (SCC-building, CS)

- Informations on MPI
  - MPI tutorial from Livermore Computing Center (https://computing.llnl.gov/tutorials/mpi/)
  - MPI Standards on http://www.mpi-forum.org