# Compiling environment

## Working on Ecgate

Xavi Abellan

Xavier.Abellan@ecmwf.int

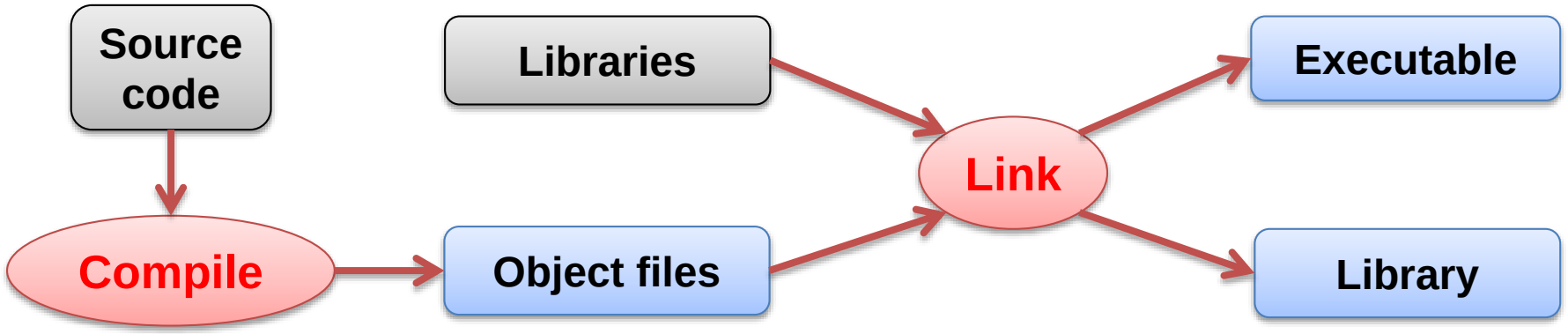
ECMWF

# Outline

- Introduction
- Fortran Compiler
- Linking
- Libraries
- Make
- Debugging
- Profiling
- Practical session

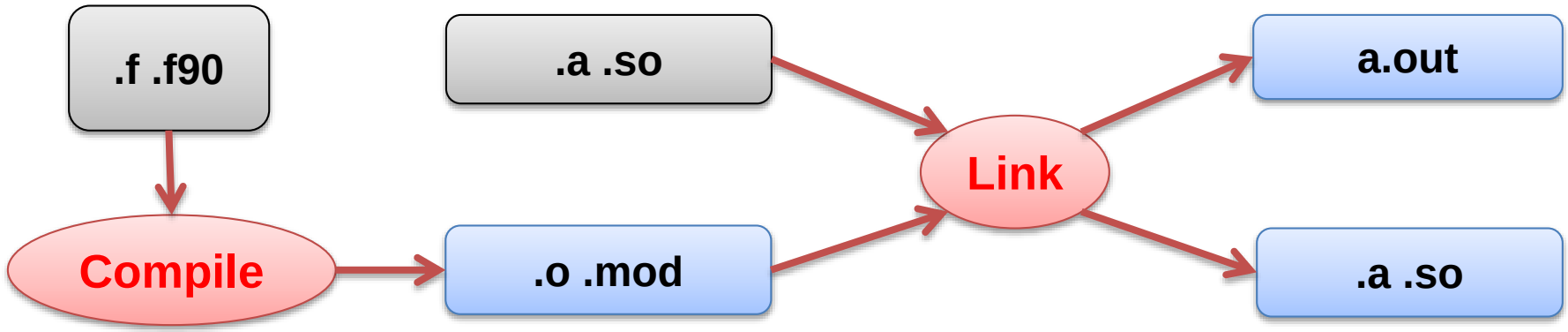
# Introduction

- Compiling
  - Objects
- Linking
  - Libraries
    - static libraries
    - shared libraries



# Introduction

- Common file suffixes and files
  - .f, .F, .f90, .F90 : source code
  - .o : object file
  - .a : archive file (library)
  - .so: share object (library)
  - .mod: Fortran 90 module files
  - a.out: default name of executable



# Introduction

- Why compiling at ECMWF?
  - decoding of (MARS) data
  - model runs
- Alternatives to compilation?
  - ecCodes tools or python interface, Metview, ...
  - Netcdf format generated from MARS
  - Wgrib, cdo, …
- Which platforms are available?
  - Linux server (ecgate)
  - Supercomputers (Cray: cca)

# Introduction

- Which compilers?
  - Fortran (77/90/95/2003)
  - C/C++
- Which platform to use?
  - High Performance Computing Facility (cca) for computing intensive work, including parallel work.
  - Linux server (ecgate) for serial decoding or I/O bound work.

# Compilers on ecgate

- GNU compilers:
  - gfortran
  - gcc
  - g++
- Which version do I use?

```
$ gcc --version
gcc (ECMWF build by usxa) 5.3.0
...
$ gfortran --version
GNU Fortran (ECMWF build by usxa) 5.3.0
...
```

# Compiling on the HPCF (cray)

- Three **programming environments**:
  - cray (default)
  - gnu
  - intel

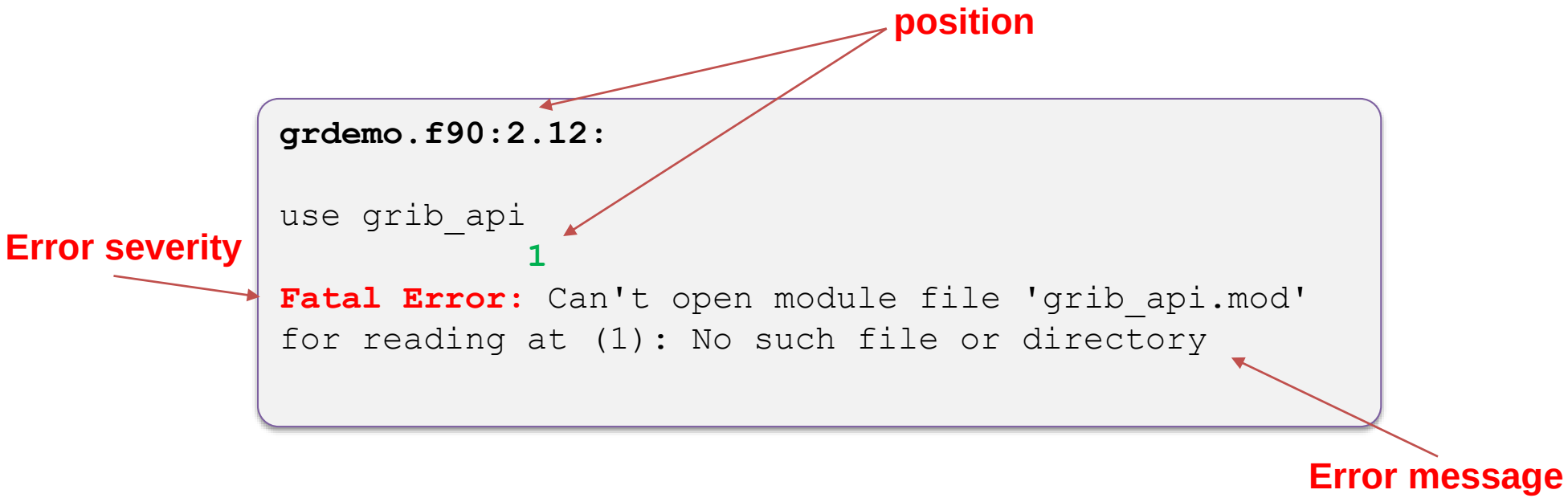
```
$ module avail PrgEnv

----------------- /opt/cray/modulefiles -----------------
PrgEnv-cray/5.2.14           PrgEnv-gnu/5.2.82(default)
PrgEnv-cray/5.2.82(default)  PrgEnv-intel/5.2.14
PrgEnv-gnu/5.2.14            PrgEnv-intel/5.2.82(default)
```

- Compilers are used via **wrappers**: **cc**, **CC**, **ftn**
  - They include all the necessary flags based on the modules loaded for most libraries
- Use the utility **prgenvswitchto** to change your programming environment
- Be aware that the compiler flags may be different for each compiler family
  - https://software.ecmwf.int/wiki/display/UDOC/Compilers

# Compilation – return codes

- Return code
  - Successful compilation: 0
  - Failure: (≠0) 1, …
  - gfortran messages:



# Fortran Compiler common options

- Fortran 77 / f90
  - -c compilation only, no linking
  - -fdefault-real-8 64bit real variables
    - -fdefault-double-8
  - -O[0-3] optimisation
  - -g debugging
  - -v verbose
  - --help display usage
- Many more options. See man page.

# Word lengths – precision

- 32bit real and integer variables by default.
- The option –fdefault-real-8 promotes real variables to 64bit entities.
  - NOTE: With GCC 5 it also promotes double variables to 128bit. Use -fdefault-double-8 to avoid it.
- When using a library, check its precision, e.g. for EMOSLIB, MAGICS.
- The ecCodes library is independent of the precision for floating points.

# Fortran I/O

- GRIB and BUFR formats are pure binary formats, accessible with ecCodes.
  - Platform independent
- IEEE format - big-endian on IBM systems (old supercomputer) , little endian on Linux systems (ecgate and Cray supercomputer)
  - real*4: 6 significant digits
  - real*8: 15 significant digits
  - Use ‘-fconvert=big-endian’ to read/write big-endian files.

# Linking

- Use gfortran to link, e.g.

```
$ gfortran -o prog prog.f $EMOSLIB
# equivalent to:
$ gfortran -o prog prog.f -L/usr/local/apps/libemos/000443/lib -lemos.R32.D64.I32
```

- Use "ar" to build static libraries, eg.

```
$ gfortran -c *.f
$ ar -vr libmy.a *.o
```

- Use gfortran -shared to build shared libraries, eg.

```
$ gfortran -c -fPIC *.f
$ gfortran -shared -fPIC -o libmy.so *.o
```

# Modules: managing your environment

- See what is loaded and what is available to load

```
$> module list
$> module avail
```

- Load and unload a module

```
$> module load package
$> module load package/version
$> module unload package
```

- Switch/swap an already loaded module by another one

```
$> module switch package/version1
$> module switch package/version1 package/version2
```

# Libraries

- ECMWF libraries
  - Graphics software library – MAGICS: $MAGPLUSLIB_SHARED (Magics++)
  - Meteorological Software - EMOS library - $EMOSLIB
  - ecCodes, for GRIB and BUFR formats – $ECCODES_LIB, $ECCODES_INCLUDE
    - The grib_api variables are still defined, but deprecated: $GRIB_API_LIB, $GRIB_API_INCLUDE
  - Locally produced software library - EC Library - $ECLIB
- Manufacturer/Public Domain Libraries
  - BLAS/LAPACK – public domain software
  - HDF/NetCDF available.

# Libraries (cont)

- Some of our locally produced libraries have both 32-bit and 64-bit floating point versions (REAL numbers) - different libraries.
- Do NOT make the confusion between the precision (32/64 bit REALS) and the ADDRESSING mode (32/64 bit) of a library:
  - You will get WRONG results when mixing libraries of different precision.
  - You will not be able to link your program if you mix libraries of different addressing mode.

# Make

- Easy to use utility to build a program or library.
- Suitable for different languages.
- **Makefile**: file containing rules on how to compile code and build library or executable.
- The ‘make’ command will read the Makefile and will figure out which code files (or libraries or executables) need to be rebuilt.
- make allows for compilations in parallel (make –j).

# Makefiles

- Contain rules that will be applied in cascade:
- The command(s) to run for each rule must be preceded by a tab

**No spaces!!!**

- Syntax:

```
target1: source1
    command_to_run target1 source1
```

- Example:

```
hello: hello.f
    $(FC) -o $@ -ffixed-form $(FFLAGS) $<
```

# Debugging

- checking:
  - array bounds checking: -fbounds-check

```
$ gfortran -fbound-check prog.f -o prog
$ ./prog
```

  - undefined reference checking

```
$ gfortran -finit-real=inf prog.f -o prog
```

**checkings done at runtime!**

- generating debug output:
  - Core file

```
$ ulimit -c unlimited
```

# Profiling - tuning

- time - command timer

```
$ time a.out
```

- -O and other options at compilation for faster execution. Try to use –O3

- other applications, like gprof

```
$ gfortran –O0 -g –pg –o prog prog.f
$ ./prog
$ gprof prog gmon.out
```

# Debugging – floating point exceptions

- Nothing generated on floating point exception.
- Floating point trapping

```
$ gfortran -ffpe-trap=overflow,invalid,zero [-g] [-O0] prog.f -o prog
$ ./prog
```

- interactive window based debugger: - totalview

```
$ module load totalview
$ totalview ./prog
```

- Core files – how to get a backtrace

```
$ gdb -c core ./prog
> where
```

# References

- GNU manuals (fortran, C, …):
  http://gcc.gnu.org/onlinedocs/

- User Documentation:
  https://software.ecmwf.int/wiki/display/UDOC/User+Documentation

- Compilers page:
  https://software.ecmwf.int/wiki/display/UDOC/Compilers

- Job examples:
  https://software.ecmwf.int/wiki/display/UDOC/Slurm+job+script+examples