An Overview of ROMS Code

Kate Hedstrom, ARSC

January 2011

Outline

- Outline of the code
- cpp
- cppdefs.h
- Modules
- ocean.in
- Compiling ROMS

Is Trunk

Atmosphere/
Compilers/
Data/
Lib/
makefile
Master/
ROMS/
User/
Waves/

- I also have an Apps directory here for my applications

ls ROMS

```
Adjoint/        License_ROMS.txt
Bin/            Modules/          SeaIce/
Drivers/        Nonlinear/        Tangent/
External/       Obsolete/         Utility/
Functionals/    Programs/         Version
Include/        Representer/
```

Most Important

- **Drivers**
  - Various model main programs
- **Nonlinear**
  - The regular ocean physics (forward model)
- **Modules**
  - Ocean model data types, with allocation and initialization routines
- **Utility**
  - File reading and writing routines and other files common to the various models

Support

- **Include**
  - Include files, including cppdefs.h
- **Bin**
  - Perl and shell scripts
- **Compilers**
  - System-dependent parts of the makefile
- **Lib**
  - ARPACK and MCT libraries (optional)
- **External**
  - ASCII input files

Other

- **Data Assimilation**
  - Adjoint
  - Representer
  - Tangent
- **SeaIce**
- **Functionals**
  - Analytic expressions for initial conditions, etc.
- **Obsolete**
- **Programs**

Master/master.F

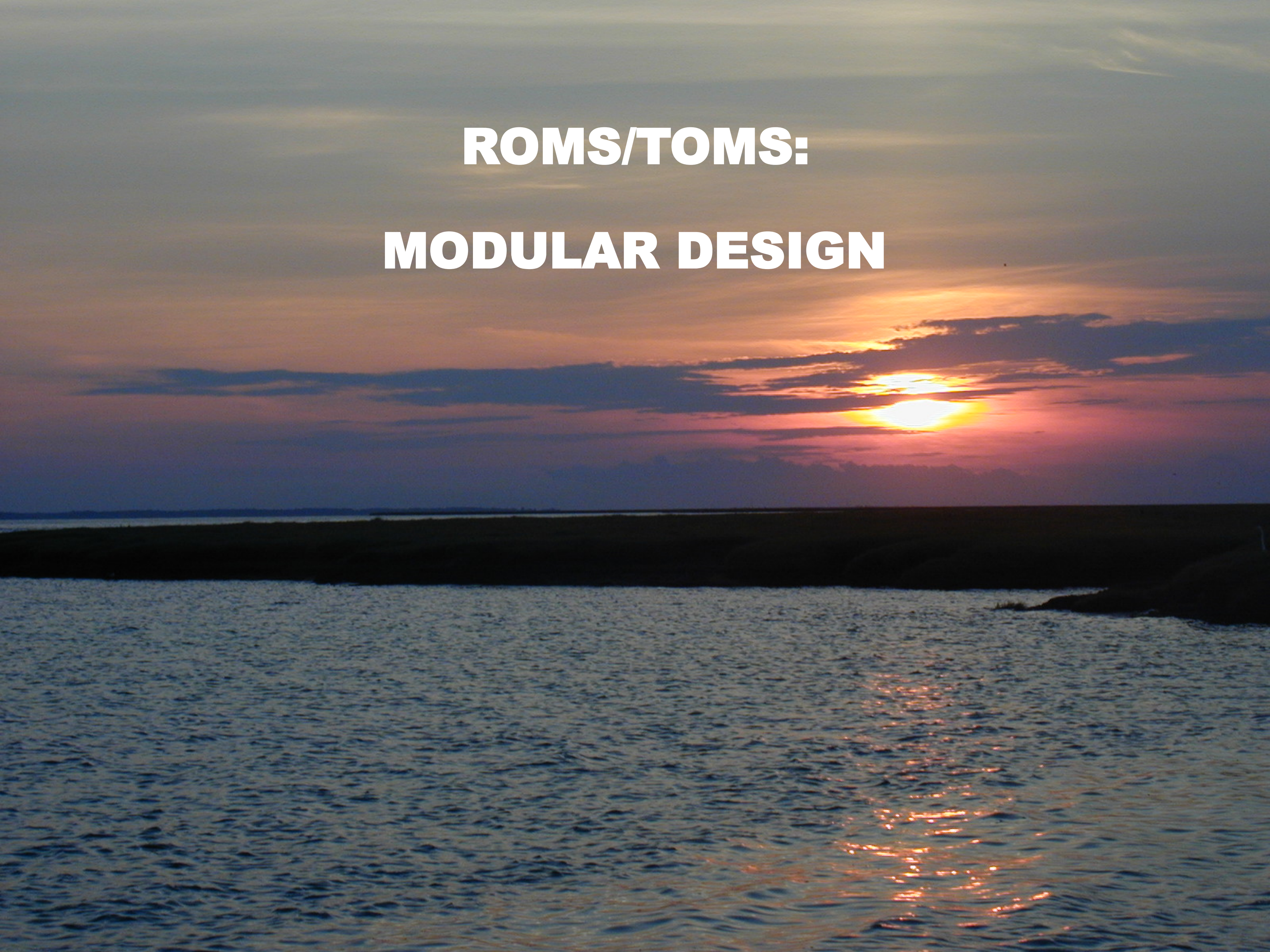
```
#include “cppdefs.h”
#if defined AIR_OCEAN
# include “air_ocean.h”
#elif defined WAVES_OCEAN
# include “waves_ocean.h”
#else
# include “ocean.h”
#endif
```

Master/ocean.h

```
#include “cppdefs.h”
    PROGRAM ocean
    USE …
#ifdef MPI
     CALL mpi_init
     CALL mpi_comm_rank(…)
#endif
     CALL initialize
     CALL run
     CALL finalize
#ifdef MPI
     CALL mpi_finalize
#endif
    END PROGRAM ocean
```

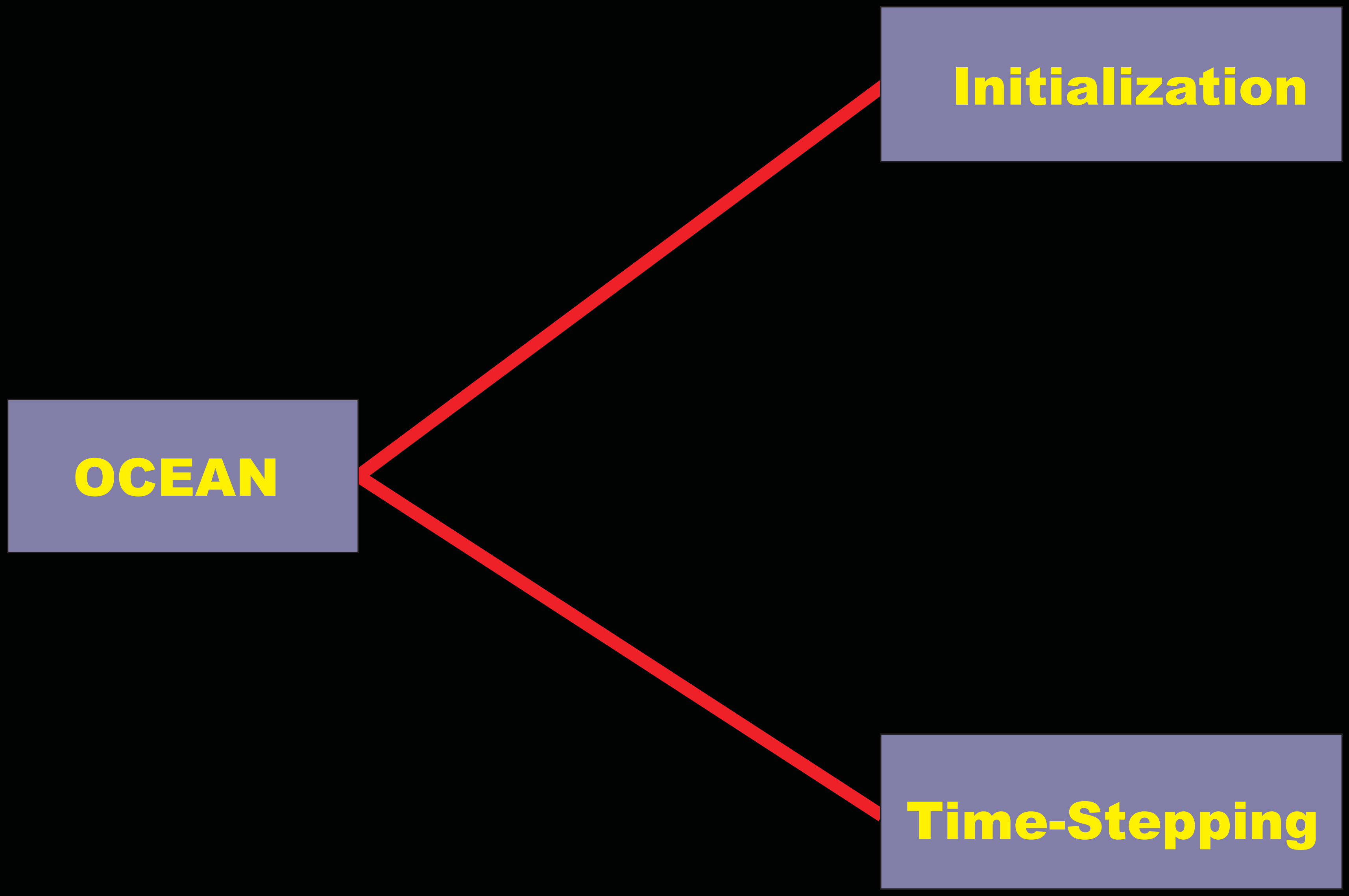
ROMS/Drivers/nl_ocean.h

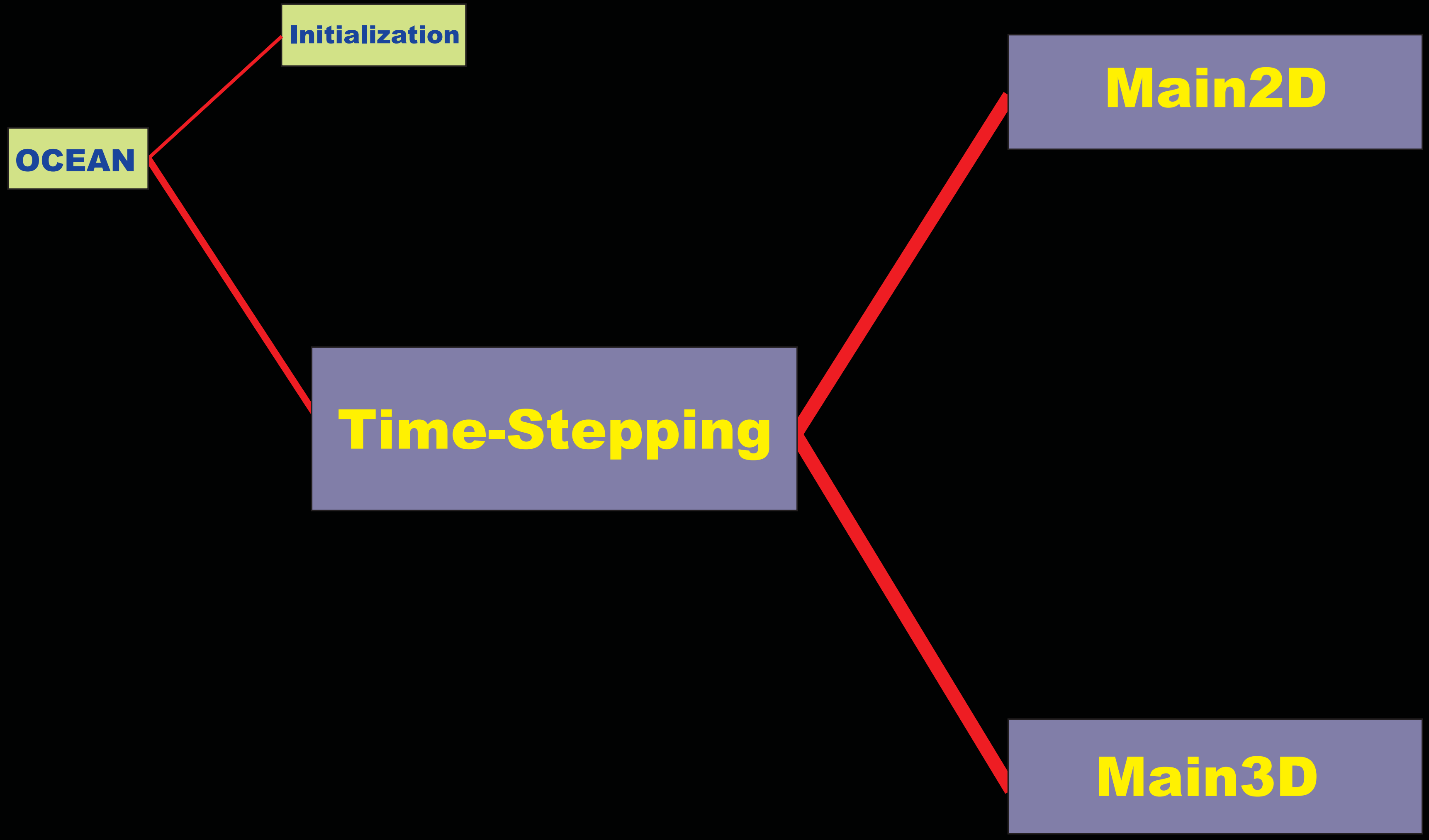
- Included by ocean_control.F
- Contains initialize, run, finalize for the nonlinear ocean model
- Run calls main3d or main2d inside the timestepping loop
- Many, many other include files for the other models

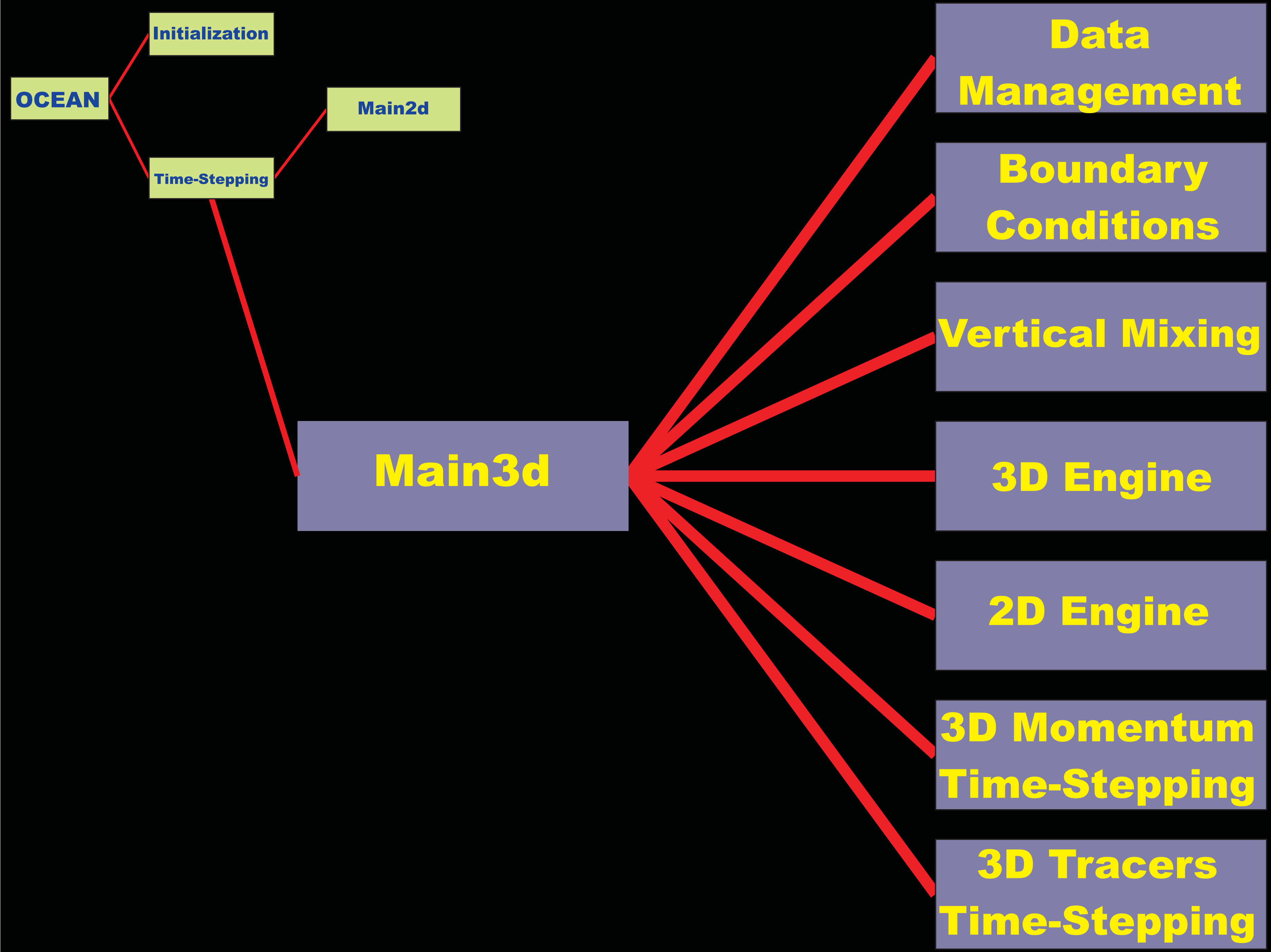


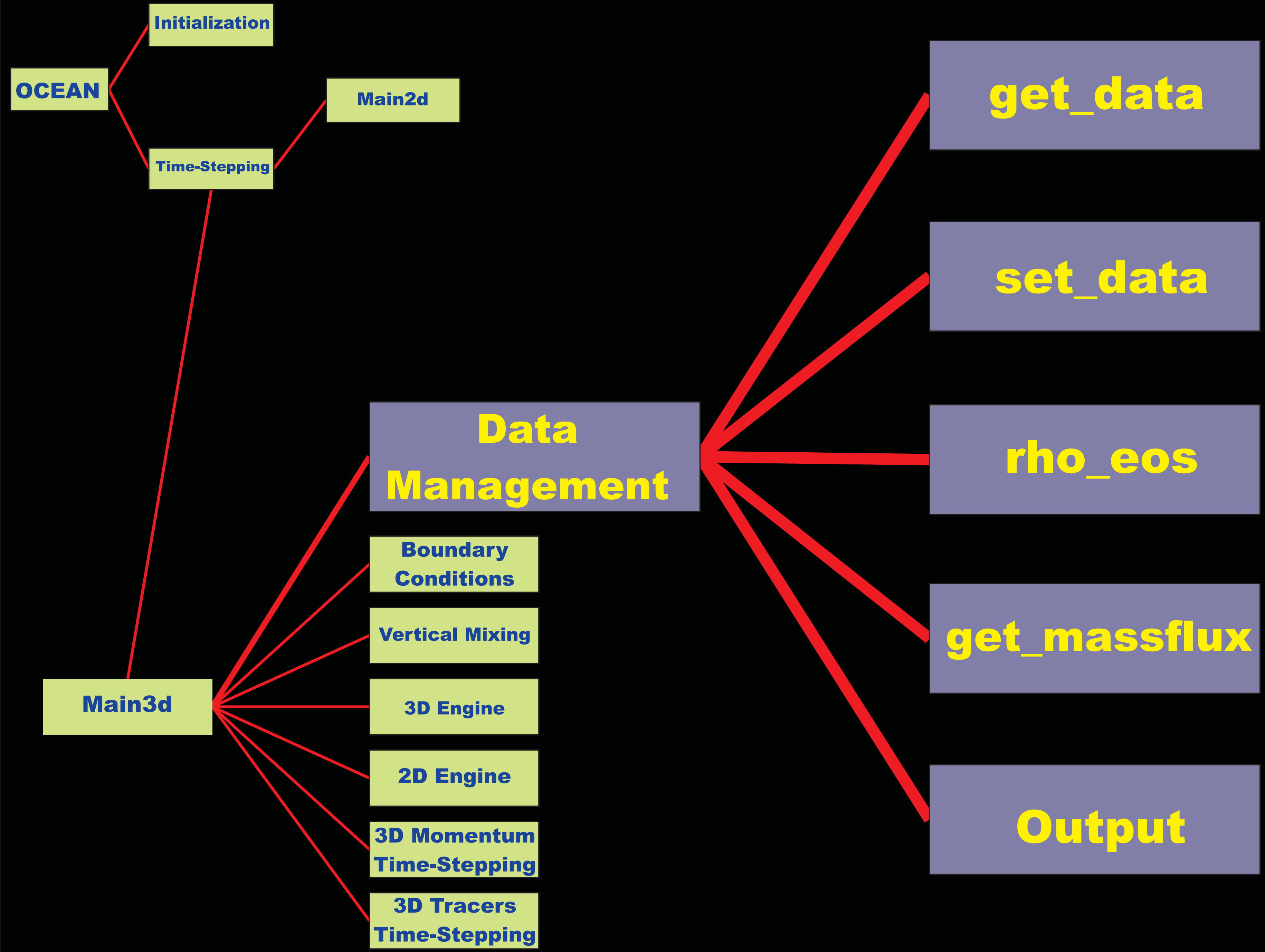
ROMS/TOMS:

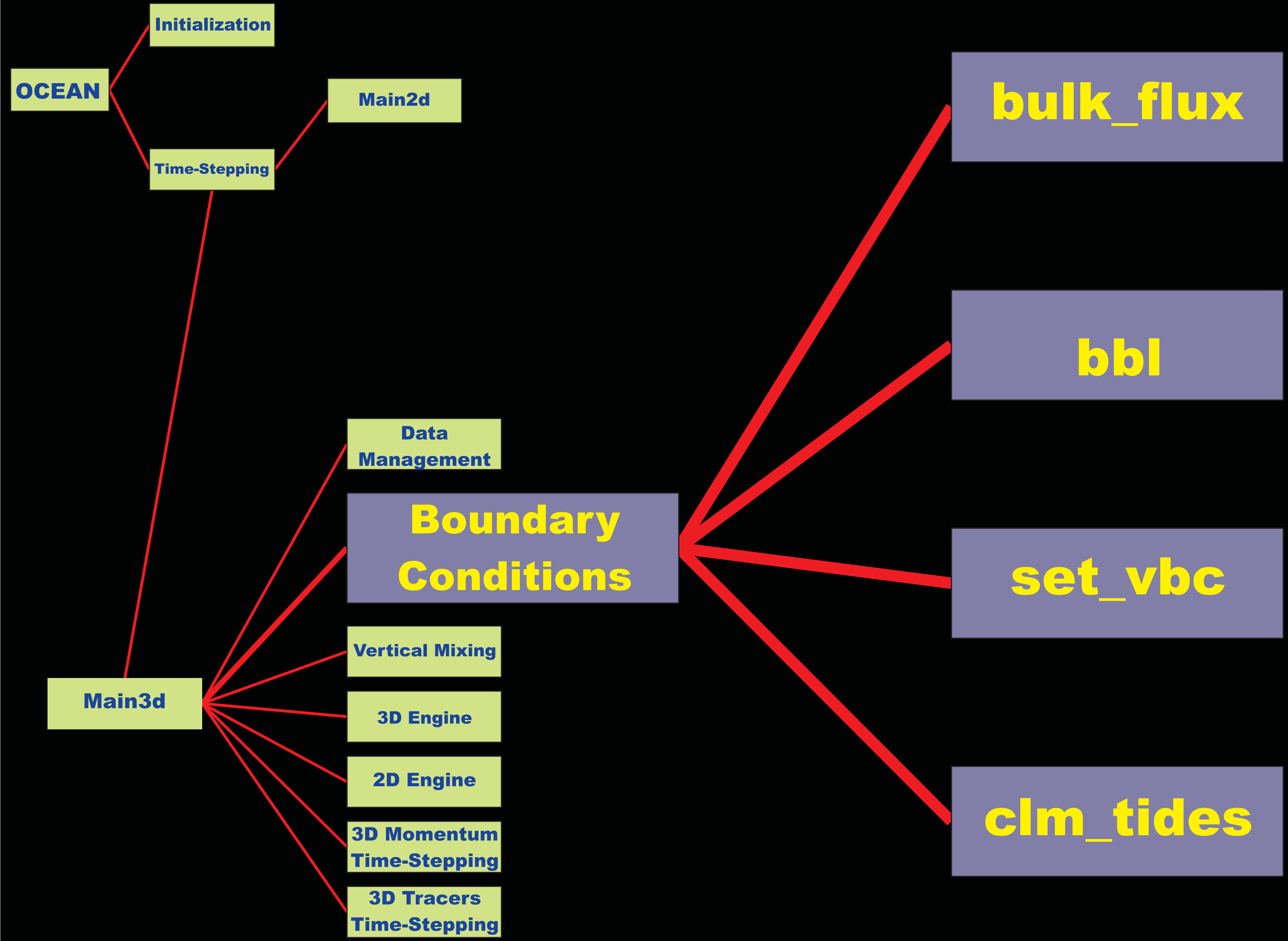
MODULAR DESIGN

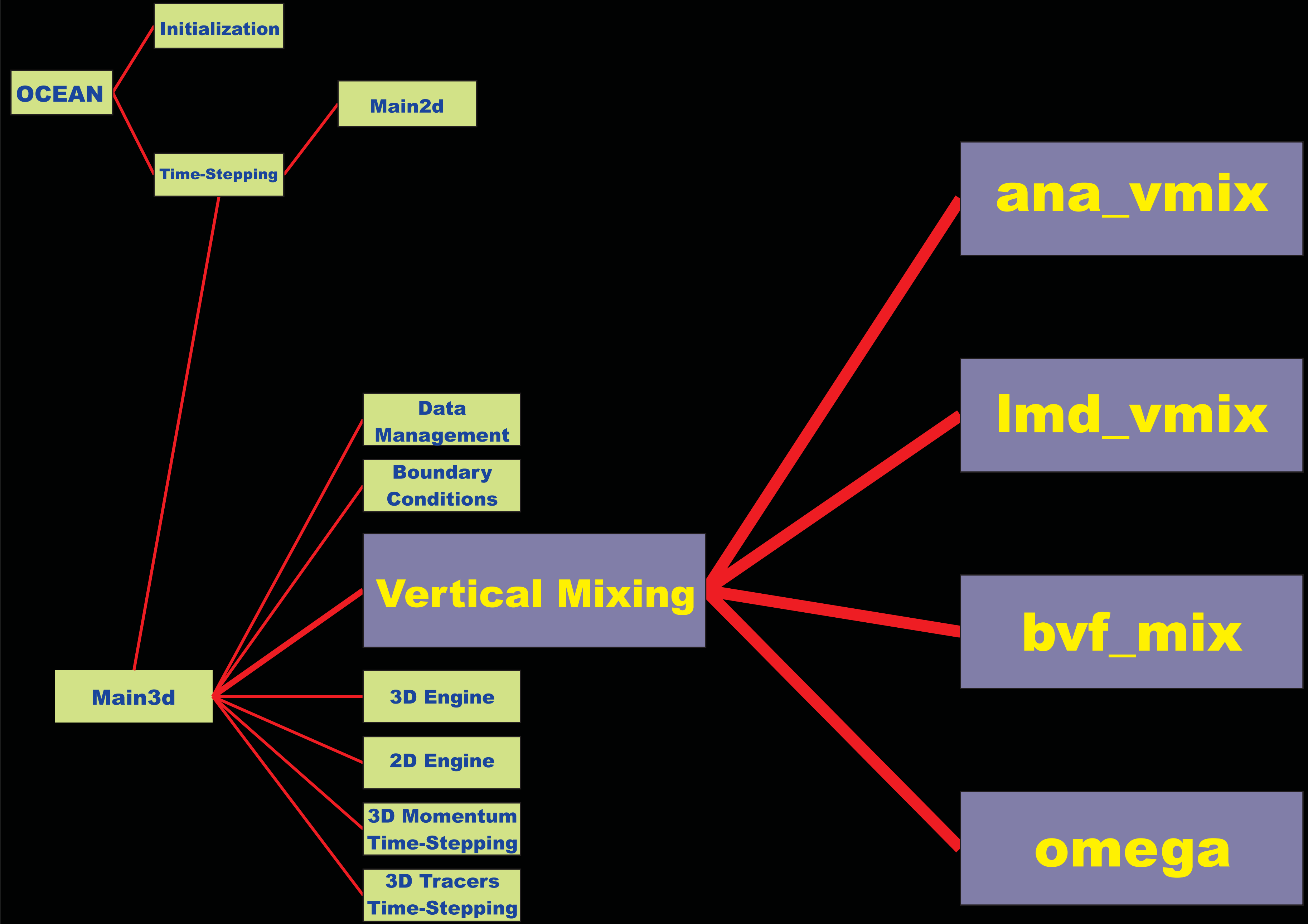

OCEAN
Initialization
Time-Stepping

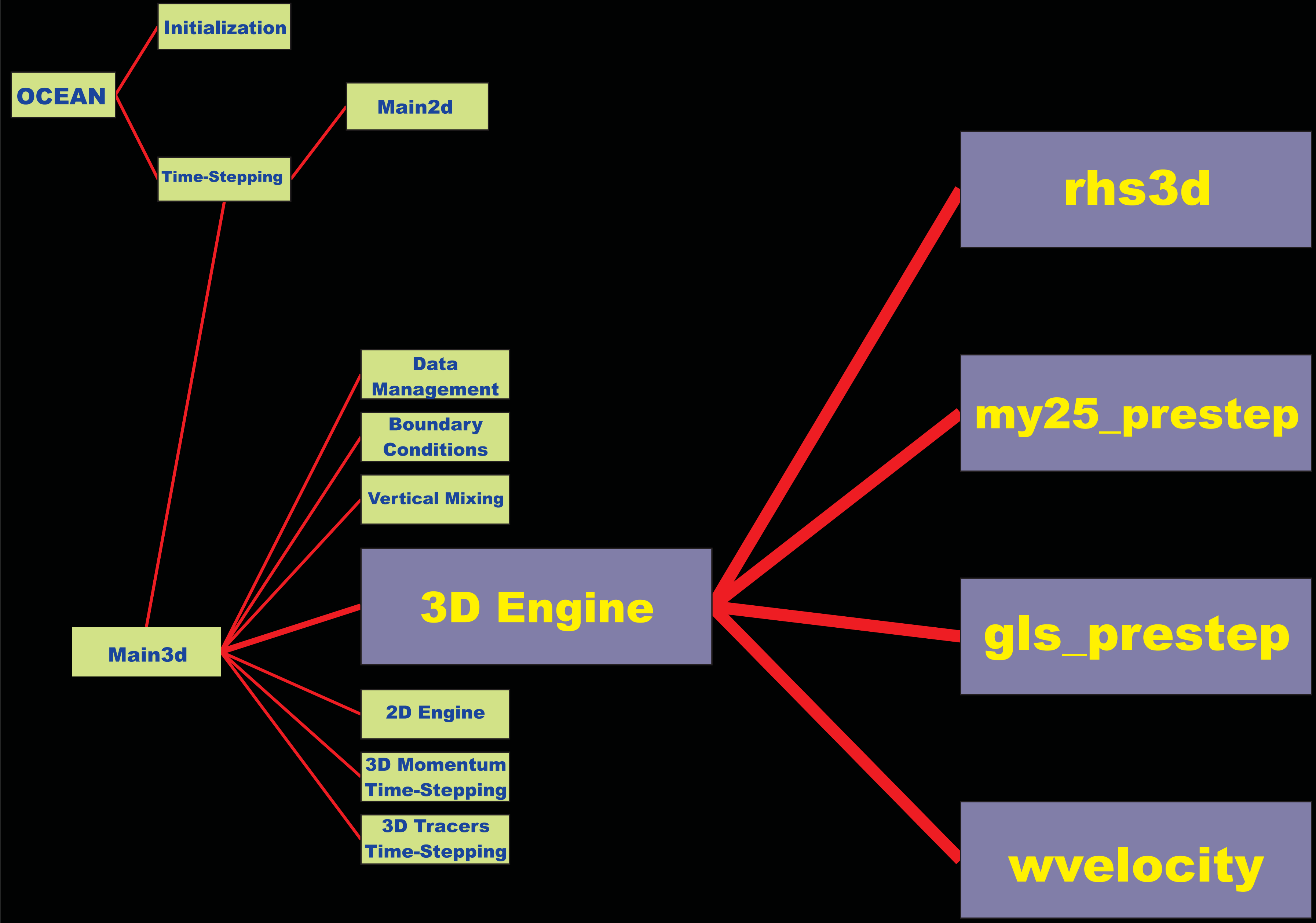

OCEAN
Initialization
Time-Stepping
Main2D
Main3D

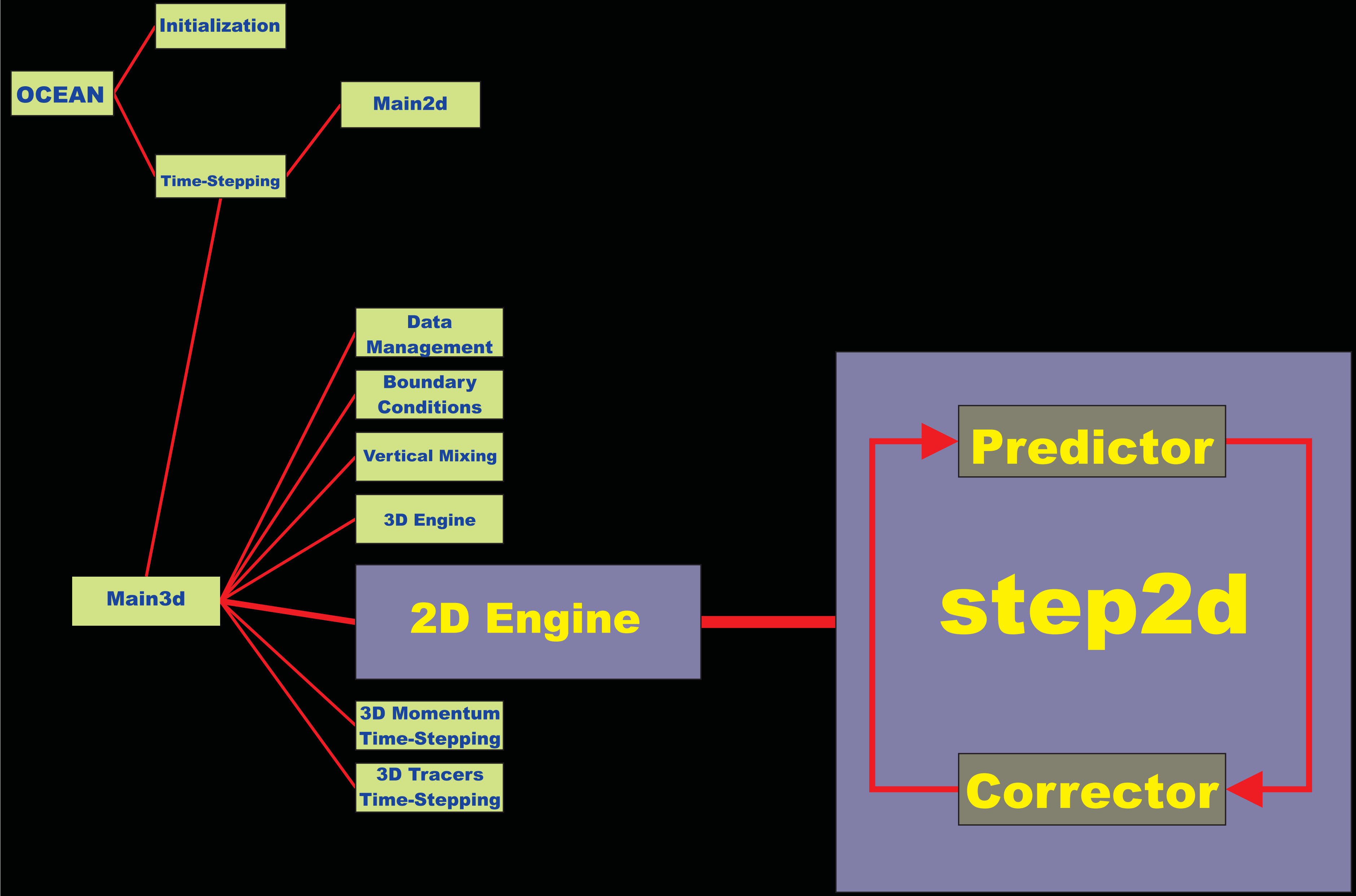

OCEAN
Initialization
Time-Stepping
Main2d
Main3d
Data Management
Boundary Conditions
Vertical Mixing
3D Engine
2D Engine
3D Momentum Time-Stepping
3D Tracers Time-Stepping


OCEAN
Initialization
Time-Stepping
Main2d
Main3d
Data Management
Boundary Conditions
Vertical Mixing
3D Engine
2D Engine
3D Momentum Time-Stepping
3D Tracers Time-Stepping
get_data
set_data
rho_eos
get_massflux
Output

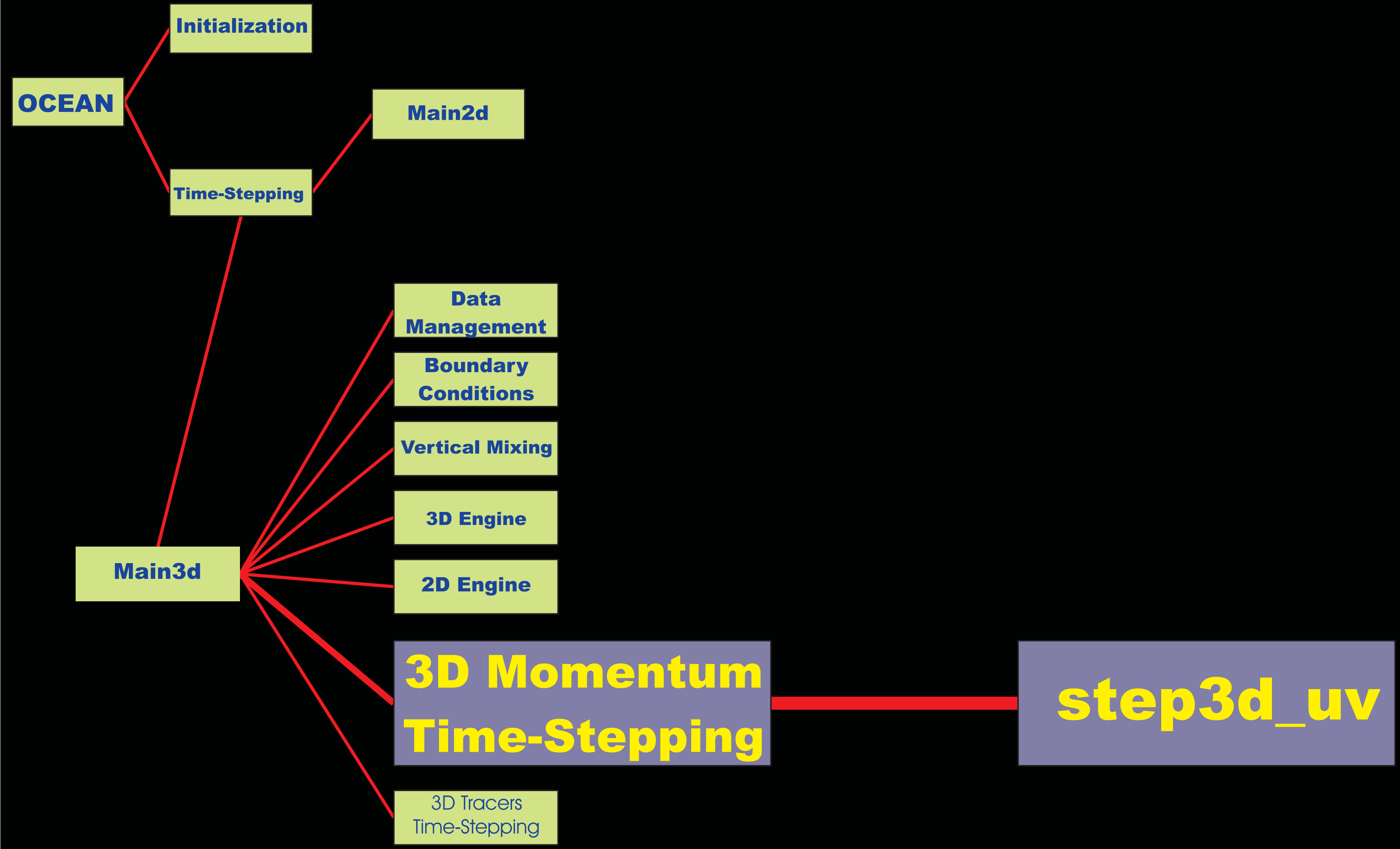

OCEAN
Initialization
Time-Stepping
Main2d
Main3d
Data Management
Boundary Conditions
Vertical Mixing
3D Engine
2D Engine
3D Momentum Time-Stepping
3D Tracers Time-Stepping
bulk_flux
bbl
set_vbc
clm_tides

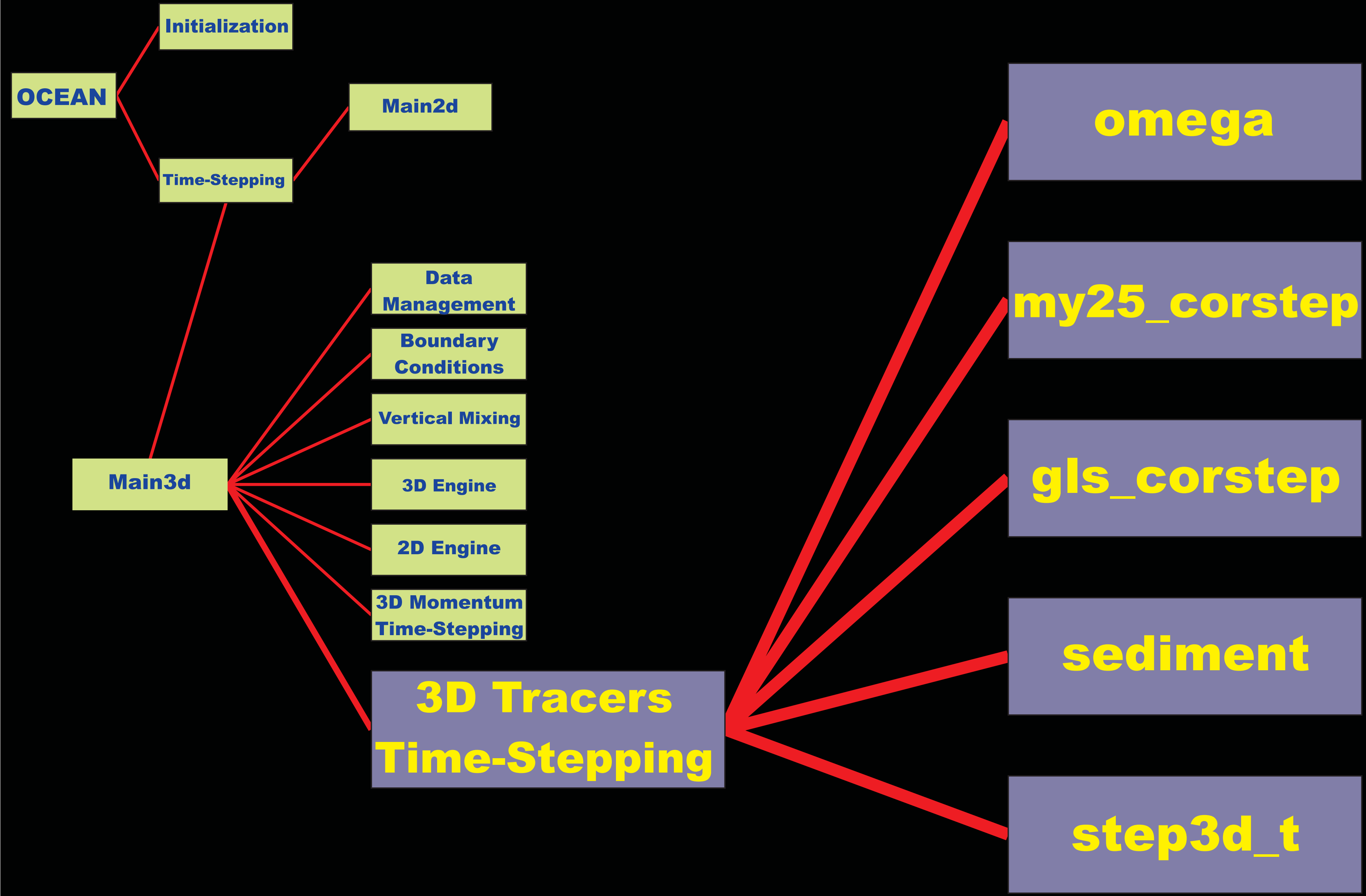

OCEAN
Initialization
Time-Stepping
Main2d
Main3d
Data Management
Boundary Conditions
Vertical Mixing
3D Engine
2D Engine
3D Momentum Time-Stepping
3D Tracers Time-Stepping
ana_vmix
lmd_vmix
bvf_mix
omega

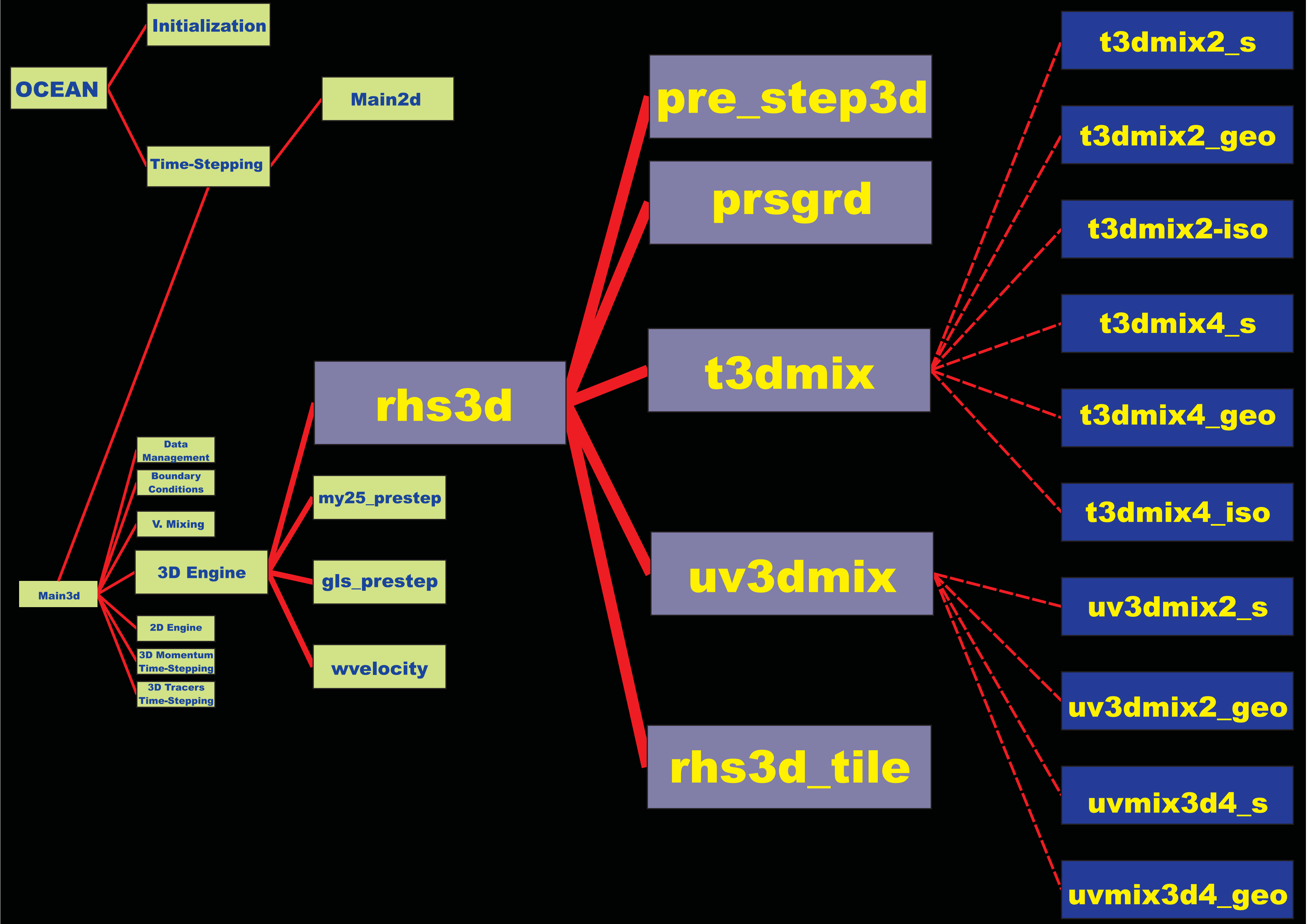

OCEAN
Initialization
Time-Stepping
Main2d
Main3d
Data Management
Boundary Conditions
Vertical Mixing
3D Engine
2D Engine
3D Momentum Time-Stepping
3D Tracers Time-Stepping
rhs3d
my25_prestep
gls_prestep
wvelocity



OCEAN
Initialization
Time-Stepping
Main2d
Main3d
Data Management
Boundary Conditions
Vertical Mixing
3D Engine
2D Engine
3D Momentum Time-Stepping
3D Tracers Time-Stepping
step2d
Predictor
Corrector


OCEAN
Initialization
Time-Stepping
Main2d
Main3d
Data Management
Boundary Conditions
Vertical Mixing
3D Engine
2D Engine
3D Momentum Time-Stepping
step3d_uv
3D Tracers Time-Stepping


OCEAN
Initialization
Time-Stepping
Main2d
Main3d
Data Management
Boundary Conditions
Vertical Mixing
3D Engine
2D Engine
3D Momentum Time-Stepping
3D Tracers Time-Stepping
omega
my25_corstep
gls_corstep
sediment
step3d_t


OCEAN
Initialization
Time-Stepping
Main2d
Main3d
Data Management
Boundary Conditions
V. Mixing
3D Engine
2D Engine
3D Momentum Time-Stepping
3D Tracers Time-Stepping
rhs3d
my25_prestep
gls_prestep
wvelocity
pre_step3d
prsgrd
t3dmix
uv3dmix
rhs3d_tile
t3dmix2_s
t3dmix2_geo
t3dmix2-iso
t3dmix4_s
t3dmix4_geo
t3dmix4_iso
uv3dmix2_s
uv3dmix2_geo
uvmix3d4_s
uvmix3d4_geo

cpp

- The C preprocessor, cpp, comes with some C compilers, or the functionality can be built into a C compiler
- Very simple macro processor
- Used in ROMS primarily for conditional compilation
- We probably won't switch to coco when it becomes widely available

cpp Versions

- People started using the C preprocessor before there was a C standard - the Standard cpp isn't quite the version we want
- Gnu "cpp -traditional" does the right thing for Fortran

File Inclusion

- In Fortran, you can include files with:

```
    include ‘file.h’
```

- In cpp, the equivalent is:

```
#include “file.h”
```

- We use the cpp version to make sure the #defines in the include files are seen

Macro Substitution

- **A macro definition has the form:**

```
#define text    replacement text
```

- **This is done in ROMS:**

```
#define WESTERN_EDGE Istr.eq.1
```

- **and used in:**

```
 if (WESTERN_EDGE) then ….
```

- **Safe as long as the replacement text is not much longer than the original**

More on Macros

- Another type of macro substitution is like statement functions in Fortran
- Statement functions and the more modern inlined functions are better because the compiler can do type checking

Logical Macros

- **A third kind of macro is something like:**

```
#define MASKING
```

- **or**

```
#define MASKING 1
```

- **These can be tested like:**

`#ifdef MASKING` **(first case)**

`#if MASKING` **(second case)**

- **We use the first style for historical reasons, gnu has officially gone to the second**

Conditional Compilation

- **ROMS uses conditional code *everywhere*.**

```
#ifdef ICE
 ! Stuff having to do with sea ice
 #endif
```

- **If you want to find out about sediment code, do a search (grep) on SEDIMENT**

More on Conditionals

- **When setting up a problem of your own, it's best to surround code you add with a unique cpp flag:**

```
#define LOMBOK_STRAIT
    :
 #ifdef LOMBOK_STRAIT
 ! My code
 #endif
```

Still More

- The ROMS Makefile will take our .F files and run them through cpp for us before passing them to the compiler
- The intermediate files have a .f90 extension
- The compiler errors will refer to line numbers in the .f90 file, not the original source file
- Fix the .F file, but feel free to look at the .f90 files to see what happened

cppdefs.h

- **Every ROMS source file starts with:**

```
#include "cppdefs.h"
```

- **This file has a list of the available options, then:**

```
#if defined ROMS_HEADER
# include ROMS_HEADER
#endif
```

- **The ROMS_HEADER variable comes from the makefile or build script**

Modules

- The model variables are stored in Fortran 90 modules defining specific types
- Many routines start with “use mod_kinds”, defining 64-bit reals, etc.
- Let’s look at a few modules…

Input file

- **ROMS has an ascii input file which it reads during initialization**
- **The file is not a namelist, but similar in intent**
- **It specifies things like:**
  - Number of timesteps
  - Number of gridpoints (Lm, Mm, N)
  - Parallel grid partitioning
  - Other input filenames, output options
  - Many others

Build System

- To compile ROMS, there is a build script, build.bash. Edit this, then run it. It invokes the “make” command.
- The “make” command uses the makefile to find its build rules.
- The makefile invokes cpp, then cpp_clean, then the Fortran compiler
- It also needs to know some things about your computer, especially where the NetCDF library is.

Build or Make?

- You can use the build script or you can use the makefile directly
- Either way, copy the standard one and edit the copy

Directories to Consider

- Where the sources are
- Where your current directory is (where the executable lands)
- Where the many intermediate files get created (.f90, .mod, .o)
- Where you put the problem dependent files (case.h, ana_grid.h, etc)
- Where to run the thing

My Preferences

- I keep the source under my home directory
- I use make and issue “make –f makefile.dujour” from my source directory
- $SCRATCH_DIR is either local Build or Build off in some scratch space
- I move the executable to scratch space visible to the compute nodes

Design Goals of Build

- Make it flexible enough that you can simultaneously build UPWELLING in one directory, CIRCLE in another
- I like to have one SCRATCH_DIR for debug, one for production
- Let’s get the build script working for UPWELLING

Build/MakeDepend

- **Automatically generated by a Perl script**
- **Has two purposes:**
  - Correct compilation order
  - Update a file and only recompile what's necessary
  - Second goal isn't quite met, hence "clean=1" in build.bash

Circle Problem

- The CIRCLE test problem comes in three flavors (so far)
- All need a C language Bessel function so the makefile is changed
- Put it in its own git branch to keep the rest clean
- Look at a makefile now…