



An extended 'getting started' guide

Torsten Bringmann

www.darksusy.org



UiO : University of Oslo

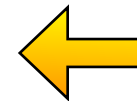
Installation

- Download the latest version

- <https://darksusy.hepforge.org/download.html>

- Configure: `darksusy-6.4.0> ./configure`

- As usual, you can provide options (see, e.g., `./conf.gfortran`)



- Make: `darksusy-6.4.0> make` (for full installation)

- More experienced users can use autocompleate to see various make targets.
NB: 'make coffee' still work in progress...

- Test installation: `darksusy-6.4.0/examples/test> ./dstest`

- After a little while you should see something like the following:

```
=====
Summary of performed DarkSUSY tests
=====
(An 'X' indicates that the corresponding module has not been built
- probably because you installed with 'make darksusy_light')

Number of errors reported by dstest_mssm:      1  (no worries...)
Number of errors reported by dstest_genWIMP:    0
Number of errors reported by dstest_genFIMP:    0
Number of errors reported by dstest_vdSIDM:     0
Number of errors reported by dstest_silveira_zee: 0

Total time needed to run dstest: about 23 s
=====
```

**Anyone did
not see this?**

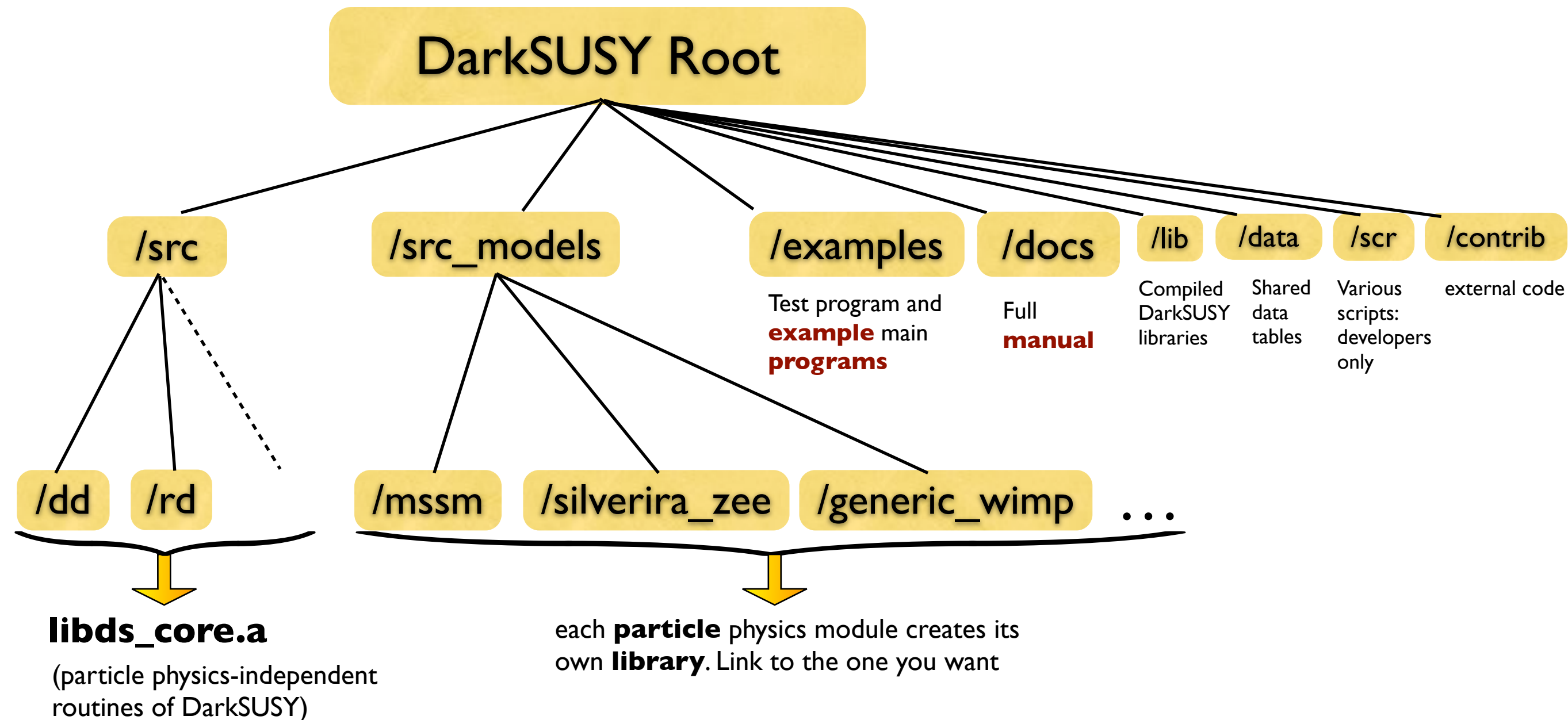


Try bypassing compilation problems
with contributed code:

`make darksusy_light`

Directory structure

- After the installation, you will see the following:



Documentation

- A **manual** (not fully up to date yet & does not cover everything) is distributed with DarkSUSY. Create with

make manual

default version(s)

make pdf_manual

only creates shorter version,
skipping subroutine headers

- Also see **headers** of various subroutines for instructions (and which author to blame 😊).

Main programs

- DarkSUSY consists mainly of **FORTRAN 77**, with some FORTRAN 95/03 additions
 - *FORTRAN is rather basic, but results in fast code*
 - *Furthermore, it is straight-forward to follow (and write)*
- DarkSUSY is essentially a **library** of routines and functions
- You, as a **user**, have to provide a suitable **main program** *(by linking to these libraries)*
- Some examples of main programs exist in the /examples folder and are good starting points

Typical Fortran program

6 chars
↔

declarations
↑
↓

program
↑
↓

```
program myprogram  
implicit none
```

includes files, typically global
variables (common blocks)

```
include 'dsver.h'  
real*8 oh2,xf  
integer unphys,war,ierr,iwar,nfc  
real*8 dsrdomega
```

arguments

(must match declaration)

subroutine call

```
call dsinit  
call dsgive_model(500.0d0,1000.d0,300.d0,10.d0,  
& 3000.d0,0.d0,0.d0)
```

exponent for
double precision
(real*8) number

```
call dsmodelsetup(unphys,war)
```

function call

```
oh2=dsrdomega(1,1,xf,ierr,iwar,nfc)  
write(*,*) 'Relic density, omega h^2 = ',oh2
```

```
end
```

continuation character (col 6)

See example programs or tutorials on the web for more Fortran examples

dsmain WIMP

- The dsmain_wimp program in examples/ essentially does the same thing, but in a more user-friendly way.

TASK: run this program!

- You will be asked what model to pick

```
What kind of SUSY model do you want to look at?  
1 = MSSM-7  
2 = cMSSM  
3 = as read from an SLHA2 file
```

**Pick MSSM-7
and enter (e.g.)**

mu: 1000 m0: 3000
M2: 1000 At/m0: 0
MA: 400 Ab/m0: 0
tan(β): 10

**TASK: Inspect output and try to identify the
corresponding part in the code!
Repeat for another model point if you like...**

dsmain_WIMP (2)

- At the end of the dsmain_wimp run you got

```
-----  
The DarkSUSY example program has finished successfully.  
Particle module that was used: MSSM  
-----  
  
[simply call 'make -B dsmain_wimp DS_MODULE=<MY_MODULE>' if you want to try  
with a different module <MY_MODULE>]
```

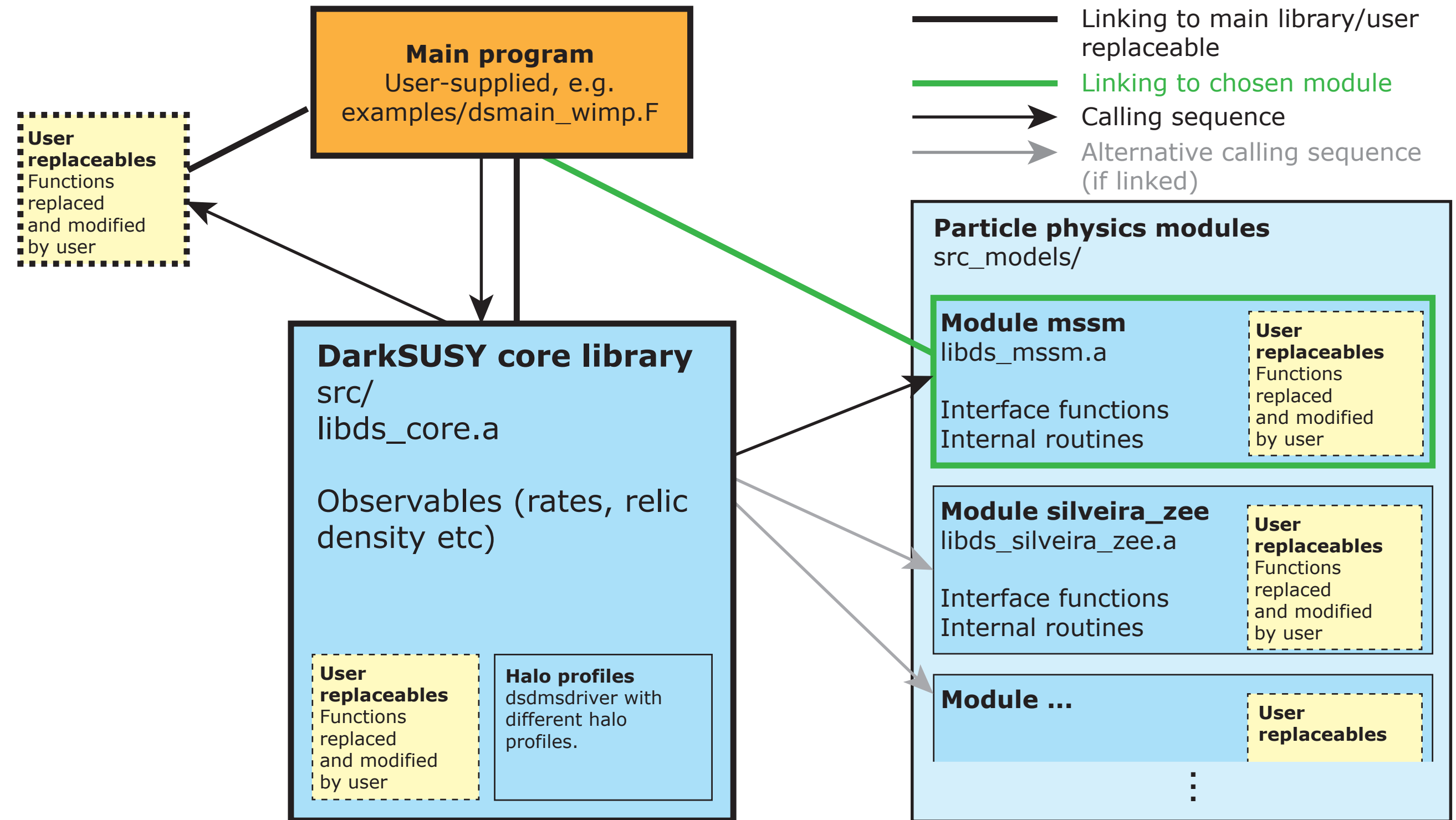
TASK: Compile and then run the same program again, now for a generic WIMP (use `<MY_MODULE> = generic_wimp`) !

use e.g. the following parameters

mass: 100
self-conjugate: 0
asymmetry: 0
annihilation cross section: $3e-26$
PDG: 15 (τ leptons)
scattering cross section: $1e-42$

*See docs/pdg_codes.pdf for
a full list of particle codes*

DarkSUSY 6 structure



Makefiles

- Which particle module is used is decided when you build your main program:

→ `gfortran -o dsmain_wimp dsmain_wimp.f -lds_core -lds_mssm` (for MSSM)
`gfortran -o dsmain_wimp dsmain_wimp.f -lds_core -lds_generic_wimp` (for generic Wimp)

- This can be made much more flexible, and system-independent, with makefiles:

```
dscheckmod :
    @test `ls ../lib/ | grep libds_${DS_MODULE}.a` || { echo ERROR: Module $
{DS_MODULE} does not exist, or is not compiled; exit 1;}

dsmain_wimp : DS_MODULE = $(shell sed -n '1p' dsmain_wimp.driver)

dsmain_wimp : dscheckmod makefile dsmain_wimp.F
    printf "#define MODULE_CONFIG MODULE_"$(DS_MODULE)"\n" > module_compile.F
    printf "$(LIB)/libds_core_user.a\n"$(LIB)/libds_core.a\n"$(LIB)/libds_"$
(DS_MODULE)"_user.a\n"$(LIB)/libds_"$(DS_MODULE)".a" > module_link.txt
    $(ADD_SCR) libds_tmp.a module_link.txt
    → $(FF) $(FOPT) $(INC) $(INC_MSSM) -L$(LIB) -o dsmain_wimp dsmain_wimp.F \
libds_tmp.a $(shell if [ "x$(DS_MODULE)" = "xmssm" ]; then printf "%s" " $
(AUX_LIB_MSSM)"; fi)
    rm -f module_compile.F
    rm -f module_link.txt
    rm -f libds_tmp.a
```

Makefiles (2)

- In `dsmain_wimp.F`, we have blocks of the type

```
#if MODULE_CONFIG == MODULE_generic_wimp
  subroutine dspmenterparameters
    [more code for this module]
#endif
```

- This is how the program performs, e.g., model-specific setup and initialisation tasks
- Alternative: separate main program for each particle physics module $\rightsquigarrow$ simpler makefiles (as in `/examples/aux/`).

Makefiles (3)

Let us compare the targets for `dsmain_wimp` and `oh2_generic_wimp`:

`/examples/makefile:`

```
dscheckmod :
    @test `ls $(LIB) | grep libds_${DS_MODULE}.a` || { echo ERROR: Module ${DS_MODULE} does not exist, or is not compiled; exit 1;}

dsmain_wimp : DS_MODULE = $(shell sed -n '1p' dsmain_wimp.driver)
dsmain_wimp : dscheckmod dscheck_libs_mssm makefile dsmain_wimp.F
dsmain_wimp : $(LIB)/libds_core.a $(LIB)/libds_core_user.a
    printf "#define MODULE_CONFIG MODULE \"$(DS_MODULE)\"\\n" > module_compile.F
    printf "$(LIB)/libds_\"$(DS_MODULE)\"_user.a\\n\"$(LIB)/libds_core_user.a\\n\"$(LIB)/libds_\"$(DS_MODULE)\".a\\n\"$
(LIB)/libds_core.a\" > module_link.txt
    $(ADD_SCR) libds_tmp.a module_link.txt
    $(FF) $(FOPT) $(INC) $(INC_MSSM) -L$(LIB) -o dsmain_wimp dsmain_wimp.F \
libds_tmp.a $(shell if [ "x$(DS_MODULE)" = "xmssm" ]; then cat mssm_libs.txt; fi)
    rm -f module_compile.F
    rm -f module_link.txt
    rm -f libds_tmp.a
    rm -f mssm_libs.txt
```

`/examples/aux/makefile:`

```
oh2_generic_wimp: DS_MODULE = generic_wimp
oh2_generic_wimp: INC_MODULE = $(INC_GENERIC)
oh2_generic_wimp: oh2_generic_wimp.f
oh2_generic_wimp: $(LIB)/libds_core.a $(LIB)/libds_core_user.a
    $(ADD_SCR) libds_tmp.a $(LIB)/libds_$(DS_MODULE)_user.a $(LIB)/libds_core_user.a $(LIB)/libds_$(DS_MODULE).a $(LIB)/libds_core.a
    $(FF) $(FOPT) $(INC) $(INC_MODULE) -L$(LIB) -o oh2_generic_wimp oh2_generic_wimp.f \
libds_tmp.a
    rm -f libds_tmp.a
```

variable describing particle
module simply set by hand

additional, module-specific libraries would
be added at these places



Don't be afraid of makefiles!

[‘Everything’ can be done by copy&paste + simple replacements]

Writing your own programs

• Typical program layout

```
call dsinit  
[make general settings]  
[determine your model parameters your way]  
call dsgive_model...  
call dsmodelsetup  
[calculate what you want]
```

• Step-by-step guide:

1. Go to your own **private** folder
[i.e. **not** a subfolder of the DS release]

2. Create a .f file with the above structure

(Or **copy** one of the files in examples [/aux] to this folder. These often provide good starting points)

3. copy examples/aux/makefile to your folder
+ modify as desired

4. 'make' in your **private** folder and then run the code

‘My’ 1st DarkSUSY program

TASK: Copy `oh2_generic_wimp.f` to some private directory, rename it to `my_first_DS_program.F`. Then compile and run it there!



There are other ways to use DarkSUSY, but this is the cleanest and by far preferred one. It keeps your routines separate from DS and makes updates easier!

‘My’ 1st DarkSUSY program

TASK: Copy `oh2_generic_wimp.f` to some private directory, rename it to `my_first_DS_program.f`. Then compile and run it there!

● Solution

1. Copy *all* files needed by `oh2_generic_wimp.f`:

```
/tutorial> cd my_code  
/tutorial/my_code> cp ../darksusy-6.4.0/examples/aux/oh2_generic_wimp.f my_first_DS_program.f  
/tutorial/my_code> cp ../darksusy-6.4.0/examples/aux/makefile .
```

2. Modify makefile **only** where **necessary**:

- replace all `oh2_generic_wimp` → `my_first_DS_program` (8 places)
- [delete blocks for unneeded targets, e.g.]

```
flxconv: DS_MODULE = generic_wimp  
flxconv: INC_MODULE = $(INC_GENERIC)  
flxconv: flxconv.f  
flxconv: $(LIB)/libds_core.a $(LIB)/libds_core_user.a  
$(ADD_SCR) libds_tmp.a $(LIB)/libds_core.a $(LIB)/libds_core_user.a $(LIB)/libds_$(DS_MODULE).a $(LIB)/libds_core.a  
$(FF) $(FOPT) $(INC) $(INC_MODULE) -L$(LIB) -o flxconv flxconv.f \br/>libds_tmp.a  
rm -f libds_tmp.a
```

WARNING:
makefiles
distinguish
<TAB> and
<SPACE> !!!

- **delete** removed (obsolete) targets from target ‘all’:
`all: my_first_DS_program dsmain_decay`

My(!) 1st DarkSUSY program

- OK, this was cheating a bit... so let's try a bit harder:

TASK:

Create your own program (in your private directory) to

1. set up a Scalar Singlet model (with some pre-defined values for coupling and DM mass) and
2. calculate the relic density and write it to screen.
3. Make sure it links to the correct libraries and compiles
4. run it!

Hints:

- Think of an example earlier seen in this tutorial — you should use **less than 15 lines** of Fortran code!
- The makefile target for oh2_ScalarSinglet probably does the same as you want to do...

My(!) 1st DarkSUSY program

● Solution

1. Create Fortran file:

```
program myprogram
implicit none

real*8 oh2,xf
integer unphys,war,ierr,iwar,nfc
real*8 dsrdomega

call dsinit
call dsgivemodel_silveira_zee(0.3d0,1000.d0) ! lambda = 0.3, mdm = 1000
call dsmodelsetup(unphys,iwar)
oh2=dsrdomega(1,1,xf,ierr,iwar,nfc)
write(*,*) 'Relic density, omega h^2 = ',oh2

end
```

2. Add following block to makefile (**copy&paste** just replace names):

```
Scalar_Singlet_example: DS_MODULE = silveira_zee
Scalar_Singlet_example: INC_MODULE = $(INC_SILVEIRAZEE)
Scalar_Singlet_example: Scalar_Singlet_example.f
Scalar_Singlet_example: $(LIB)/libds_core.a $(LIB)/libds_core_user.a
$(ADD_SCR) libds_tmp.a $(LIB)/libds_$(DS_MODULE)_user.a $(LIB)/libds_core_user.a
$(LIB)/libds_$(DS_MODULE).a $(LIB)/libds_core.a
$(FF) $(FOPT) $(INC) $(INC_MODULE) -L$(LIB) -o Scalar_Singlet_example
Scalar_Singlet_example.f \
libds_tmp.a
rm -f libds_tmp.a
```

**Make sure to not
convert <TAB>s
to <SPACE>s
without noticing!**



3.[Make sure to keep the definition (at the beginning)]

```
INC_SILVEIRAZEE=-I$(DS_INSTALL)/src_models/silveira_zee/include
```

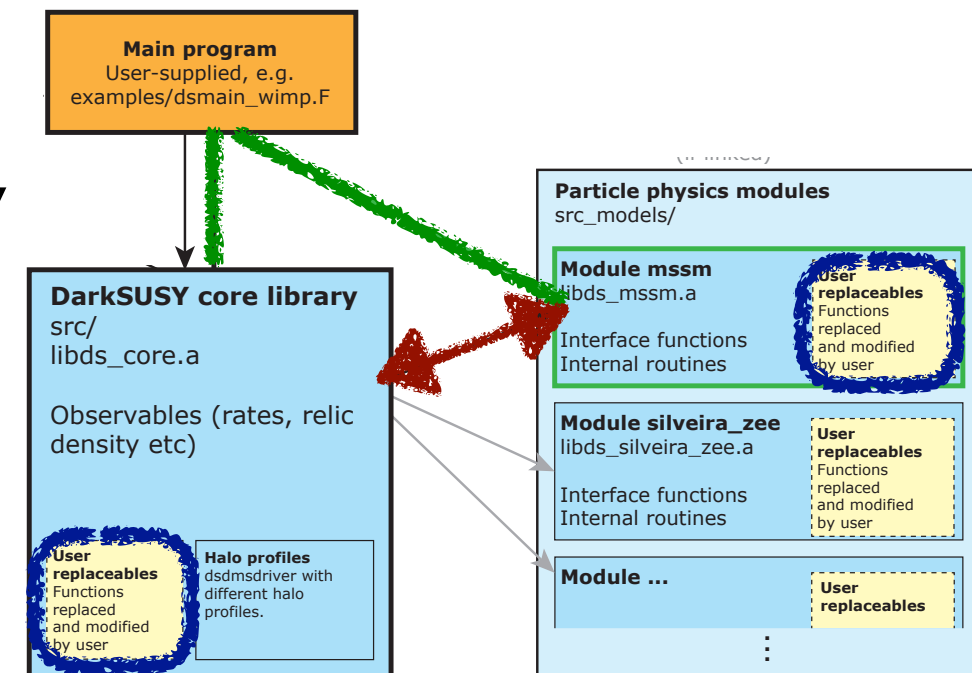
TASK: DISCUSS (or try...) — what do you have to change to do the same for the generic_WIMP model?

DarkSUSY modularity: key concepts

- **Main program** always **links** to DS_core and *one* particle module

- **Interface functions** communicate model-dependent input to core library

- ‘Set of interface functions **defines** particle module’
- No further exchange between core and modules
- Minimal: about a dozen in total
- A particle module can provide less — this only restricts possible applications in main program [error at linking stage points to missing interface function]



- Most functions are **replaceable functions**

- Can be individually replaced **at linking stage** (when building the main program)
- DarkSUSY **installation** remains **unchanged**
- User-supplied function will still be consistently used in rest of code
- Examples: external annihilation rate for relic density calculation; different yields for indirect detection routines, etc...

Using a replaceable function

- Let's assume you don't trust DarkSUSY — purely hypothetically of course 😊
- So you want to **replace** e.g. the invariant rate W_{eff} (a typical example of an **interface function**) for the scalar singlet model, provided by the code, with a result that you obtained yourself.

Using a replaceable function (2)

- Let's say — for the sake of the argument — that you are 'really sure' that there is a missing factor of $E_{cm}^2/(4m_S^2)$ in the code



If you now (or at any other time...) think about changing a function or subroutine in the installed DarkSUSY folder: **DON'T !!!** ⚡ ⚡ ⚡

TASK:

Correct this 'mistake' in the function `src_models/silveira_zee/an/dsanwx.f`, **by 'replacing' it, and test the result !**

Hint: • Have a look at the `generic_wimp_oh2[_threshold]` case in `examples/aux/makefile ...`

Using a replaceable function (3)

Solution

1. Copy [DS]/src_models/silveira_zee/an/dsanwx.f to [my local directory]/my_replaceables, and add the following lines at the end:

```
c... TB: here we add the additional factor required in the tutorial
      s = 4.*(dsmwimp()**2 + p**2) ! Ecm**2
      dsanwx = dsanwx*s/dsmwimp()**2/4.d0
c      write(*,*) 'replaceable test'

      return
      end
```

2. Add following block to makefile (copy&paste, **changes** as indicated)

```
ScalarSinglet_mod DS_MODULE = silveira_zee
ScalarSinglet_mod INC_MODULE = $(INC_SILVEIRA_ZEE)
ScalarSinglet_mod: ScalarSinglet_example.f my_replaceables/dsanwx.f
ScalarSinglet_mod $(LIB)/libds_core.a $(LIB)/libds_core_user.a
$(ADD_SCR) libds_tmp.a $(LIB)/libds_$(DS_MODULE)_user.a $(LIB)/libds_$(DS_MODULE).a $(LIB)/libds_core.a
$(FF) $(FOPT) $(INC) $(INC_MODULE) -L$(LIB) -o ScalarSinglet_example ScalarSinglet_example.f
my_replaceables/dsanwx.f \
libds_tmp.a
rm -f libds_tmp.a
```

Using 'dsanwx.f' → '*.f' allows you to replace many functions at the same time (and switch such changes on/off simply by dragging them into/out of your my_replaceables/ folder !)



Example programs

- Our ‘minimal’ application examples are quite instructive, and a very good next step to explore further:

```
/examples/aux> ls *.f
```

```
DDCR_flux.f  
DDCR_limits.f  
DD_example.f  
DMhalo_bypass.f  
DMhalo_bypass_prep.f  
DMhalo_los.f  
DMhalo_new.f  
DMhalo_predef.f  
DMhalo_table.f
```

```
FreezeIn_ScalarSinglet.f  
FreezeIn_generic_fimp.f  
ScalarSinglet_thermal_averages.f  
caprates.f  
caprates_ff.f  
flxconv.f  
flxconvplot.f  
neutrinospectra.f  
neutrinoysields.f
```

```
oh2_ScalarSinglet.f  
oh2_aDM.f  
oh2_cBE_ScalarSinglet.f  
oh2_dark_sector.f  
oh2_generic_wimp.f  
oh2_vdSIDM.f  
sucmh_test.f  
wimpyields.f
```

+self-interactions!

direct detection examples

indirect detection

usage of halo model database

relic density [+ kinetic decoupling]

Ultra-compact minihalos

- Any **questions** ?

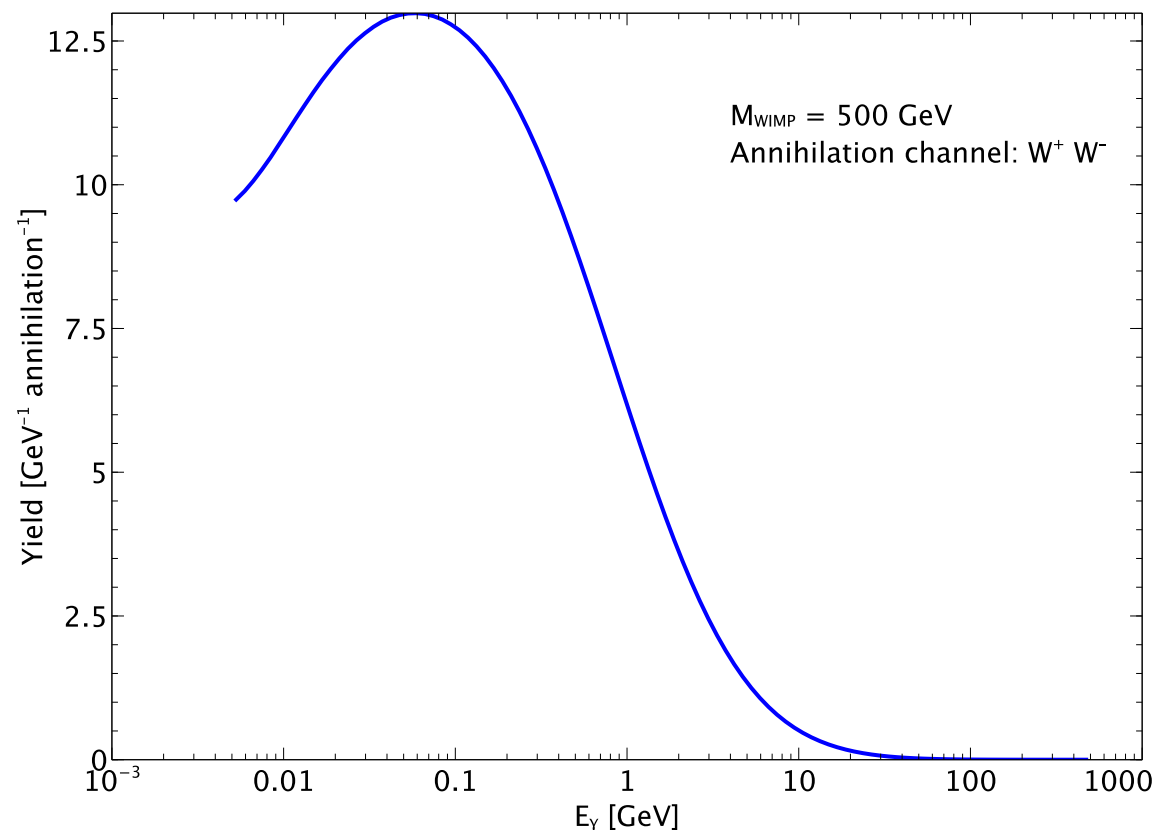


- *During or after the school*
- *Concerning these example programs or other DarkSUSY-related issues*
- ***Just get in touch, also after the workshop !***

WIMP yields

- E.g. `wimpyields.f` shows how to access the yield tables calculated with PYTHIA

TASK: Calculate the differential gamma-ray flux for a WIMP of mass 500 GeV annihilating to W^+W^- !



- If you want to explore this further, you could for example
 - look at integrated yields
 - look at different channels
 - load different tables, e.g. the improved spectra for sub-GeV DM annihilation derived by *Plehn, Reimitz & Richardson [1911.11147]*