A C and Matlab interface to LAPACK

by Rémi Delmas

supervised by Julien Langou

Innovative Computing Laboratory

COMPUTER SCIENCE DEPARTMENT

UNIVERSITY OF TENNESSEE


ORANGINA
ARRIVÉE
ORANGINA
ORANGINA
DÉPART
TROISEL
MAIRIE DE TOULOUSE
BIO-TRADE
210

A C Interface to LAPACK

by Rémi Delmas

supervised by Julien Langou

Innovative Computing Laboratory

COMPUTER SCIENCE DEPARTMENT

UNIVERSITY OF TENNESSEE

Summary

1. Why a wrapper for C to LAPACK?
2. Design of the package
3. Testing!!!

1.a Why a wrapper? Intro

```
/* Example on how to use LAPACK from C */
/* This code solve the symmetric positive definite system Ax=b */
/* using Cholesky factorization. One RHS*/
int lda, ldb, n, nrhs;
double *A, *b;
/* Allocate and Affect values in A and b (column-major) */
```

```
/* access to LAPACK FORTRAN library */
nrhs=1;
dposv_( "U", &n, &nrhs, A, &lda, b, &ldb, &info );
```

```
#include "lapack.h"
/* access to LAPACK FORTRAN library through a wrapper */
lapack_dposv( lapack_uplo_U, n, 1, A, lda, b, ldb, &info );
```

Problem with direct access:

- Name mangling: dposv_, dposv, DPOSV, dposv__ ??
- Need to declare nrhs
- Assume types ok between Fortran and C
- Enums enables clearer code and useful debugging information
- Proto provided in the .h file

Convertion of types, example.

```
#ifdef CRAY
  #include <fortran.h>
  #define F77_CHAR _fcd
  #define C2F_CHAR(a) ( _cptofcd( (a), 1 ) )
  #define C2F_STR(a, i) ( _cptofcd( (a), (i) ) )
  #define F77_STRLEN(a) (_fcdlen)
#endif

#ifdef WeirdNEC
  #define F77_INT long
#endif
```

1.b Goals of a wrapper

1. Implement a “nice” user interface, for C this means:
   a) Avoid adress passing where it is not mandatorry.
   b) Enums
   c) Prototype
   d) Transparent convertion from C type to Fortran (and vice-versa) for logical/integer/character
   e) Takes care of the name mangling
   f) Double-layer for functions
2. Be portable
   a) It has to work on all machines where LAPACK runs with any combination C/Fortran compiler.
   b) Compatible with any LAPACK distribution (netlib, vendors,…).
3. Be as efficient as possible
4. Have a test suite
5. Be automatically generated (as much as possible, at least)
   a) To avoid maintainenance
   b) Be useable by other Fortran packages

⇒ Portability of the user application code

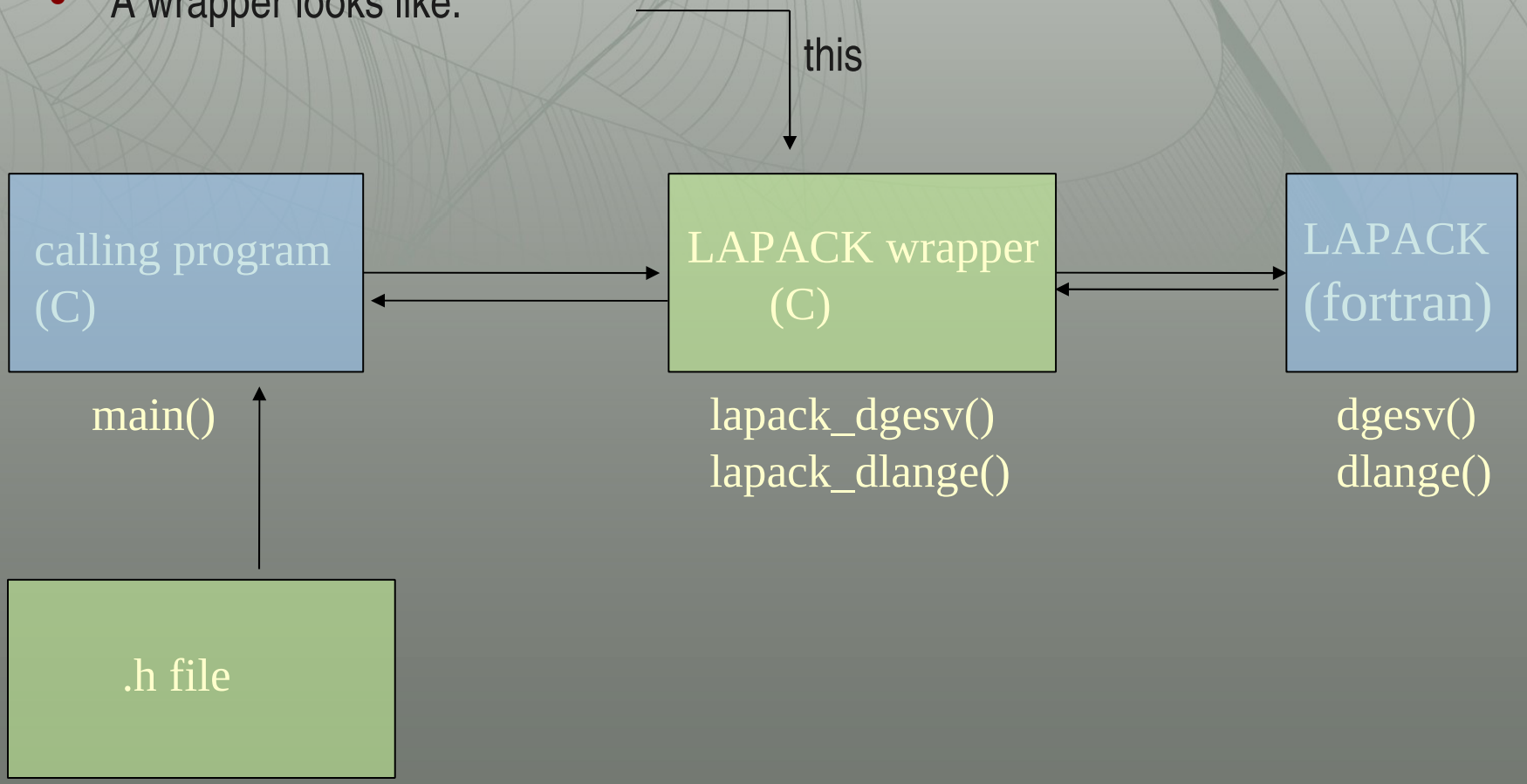
⇒ The C user should not know that LAPACK is written in Fortran*

*ok, column major remains

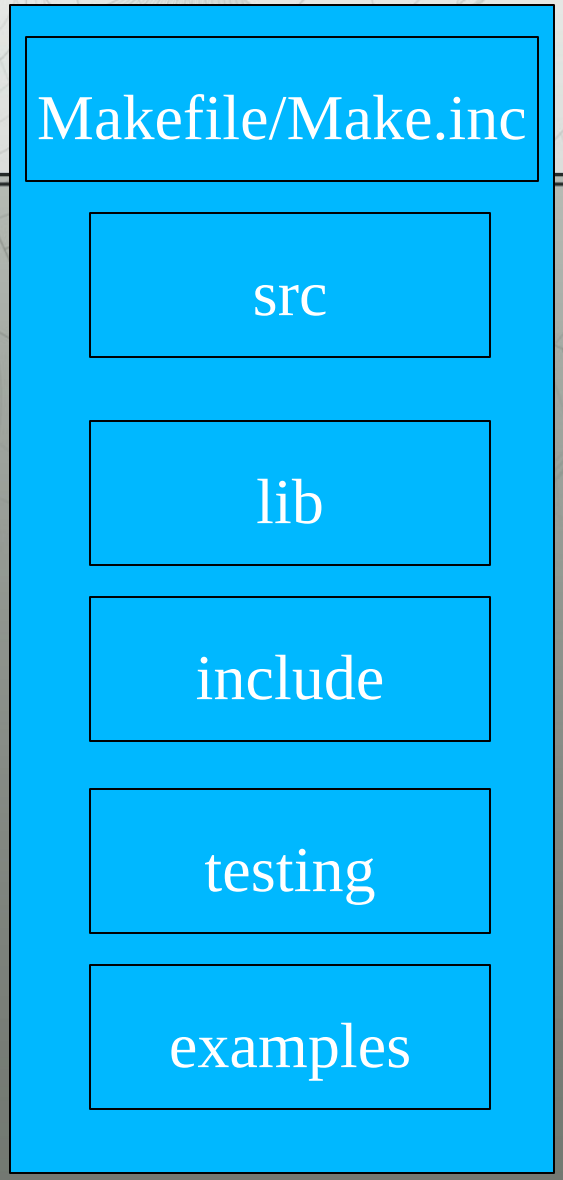
Package	Name mangling	Convertion integer character logical	Enum	Avoid adress passing if possible	Prototype	Double layer for function	Comments
LAPACK Fortran							LAPACK netlib great for Fortran
CBLAS wrapper	**OK**	**OK**	**OK**	**OK**	**OK**	**OK**	For BLAS only
CLAPACK	**OK**	**OK**			**OK**		F2C version of LAPACK (slow?, what about other dist.)
CLAPACK + clapackwrap	**OK**	**OK**		**OK***	**OK**		*Only drivers
Vendors, ISV LAPACK	**OK**	**OK**	**??**	**OK**	**OK**		$!
LAPACK ATLAS	**OK**	**OK**	**OK**	**OK**	**OK**		- Only DGESV/DPOSV + Row major/Column major
LPP (LAPACK PLUS PLUS)	**OK**	**OK**		**OK**	**OK**		For C++, automatic allocation in option
Babel/SIDL	Ask Piotr why bad – Ask Victor why great : discussion on the LAPACK developer forum						
LAPACK C wrapper	**OK**	**OK**	**OK**	**OK**	**OK**	**OK**	What I am speaking of today (+ test)

3. Design

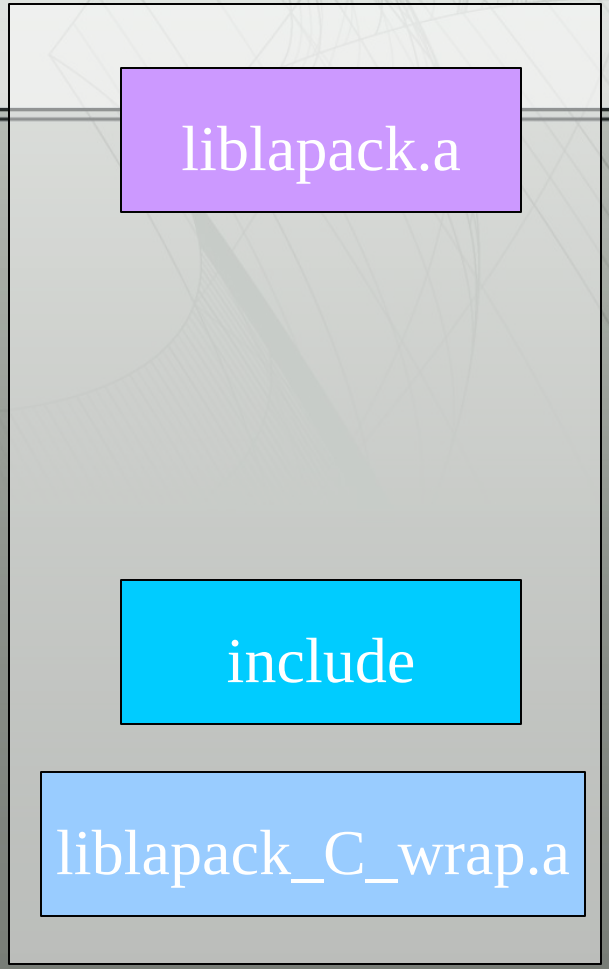
- A wrapper looks like:



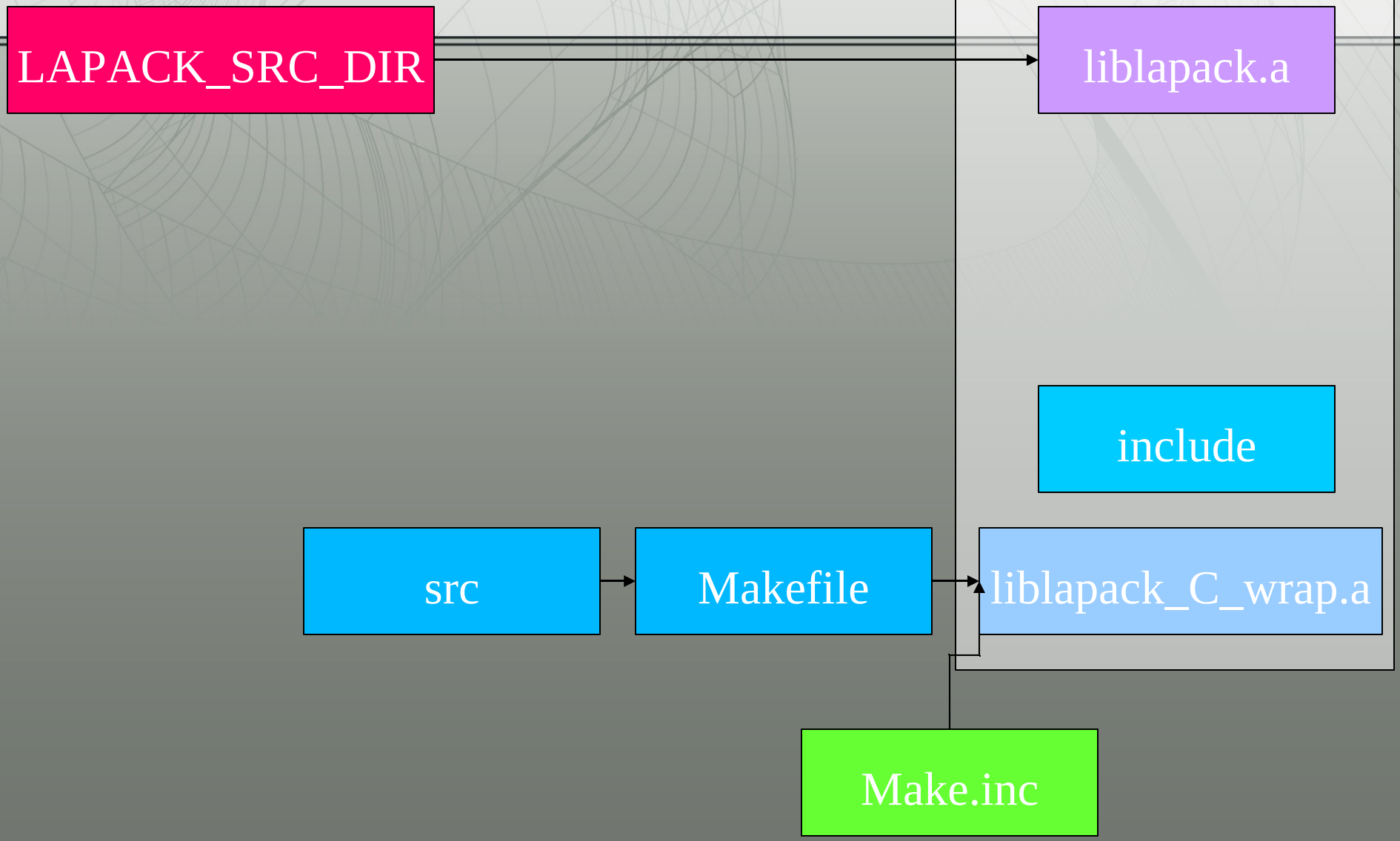
3. Design of clapack_C_wrappper



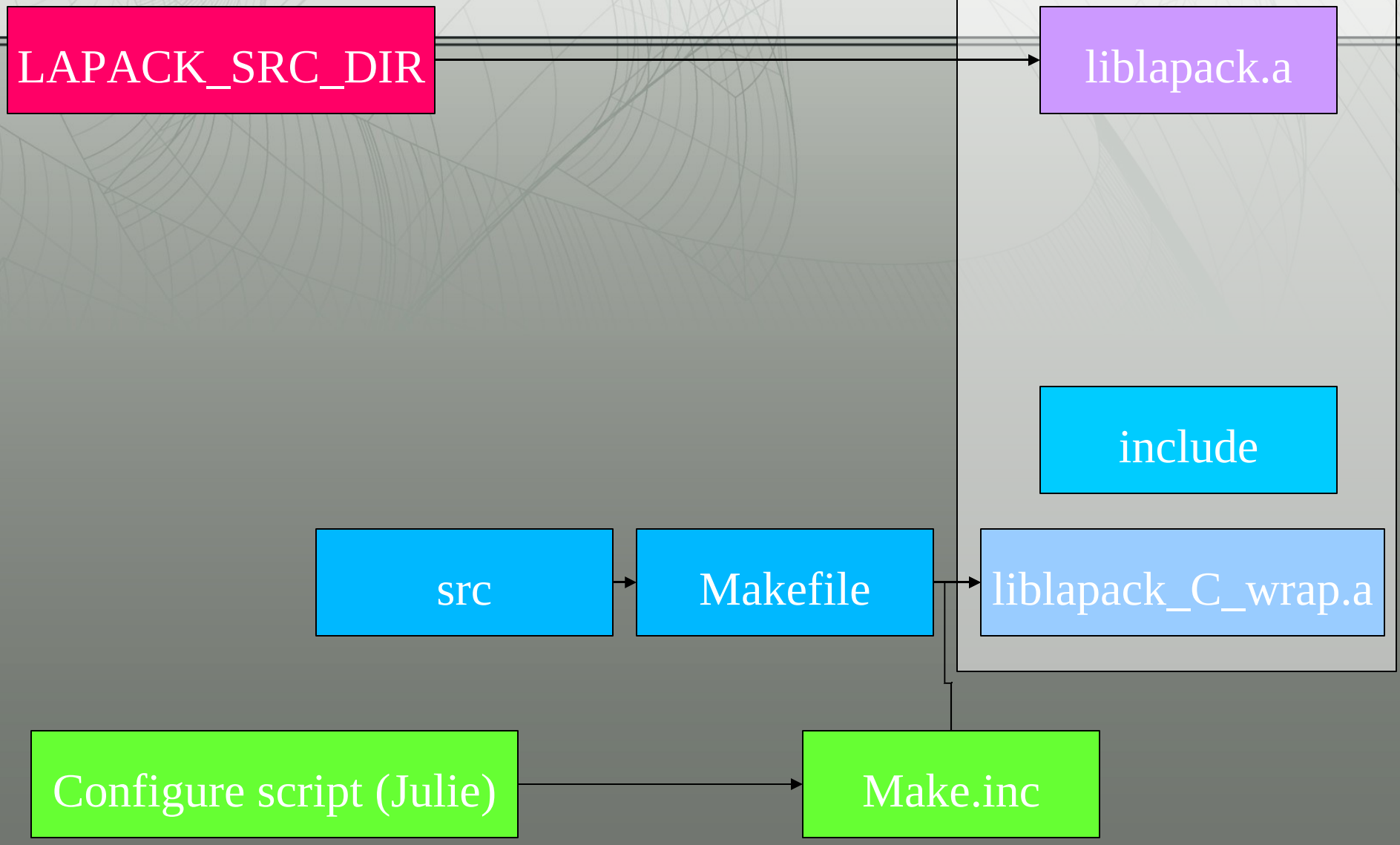
3. Design



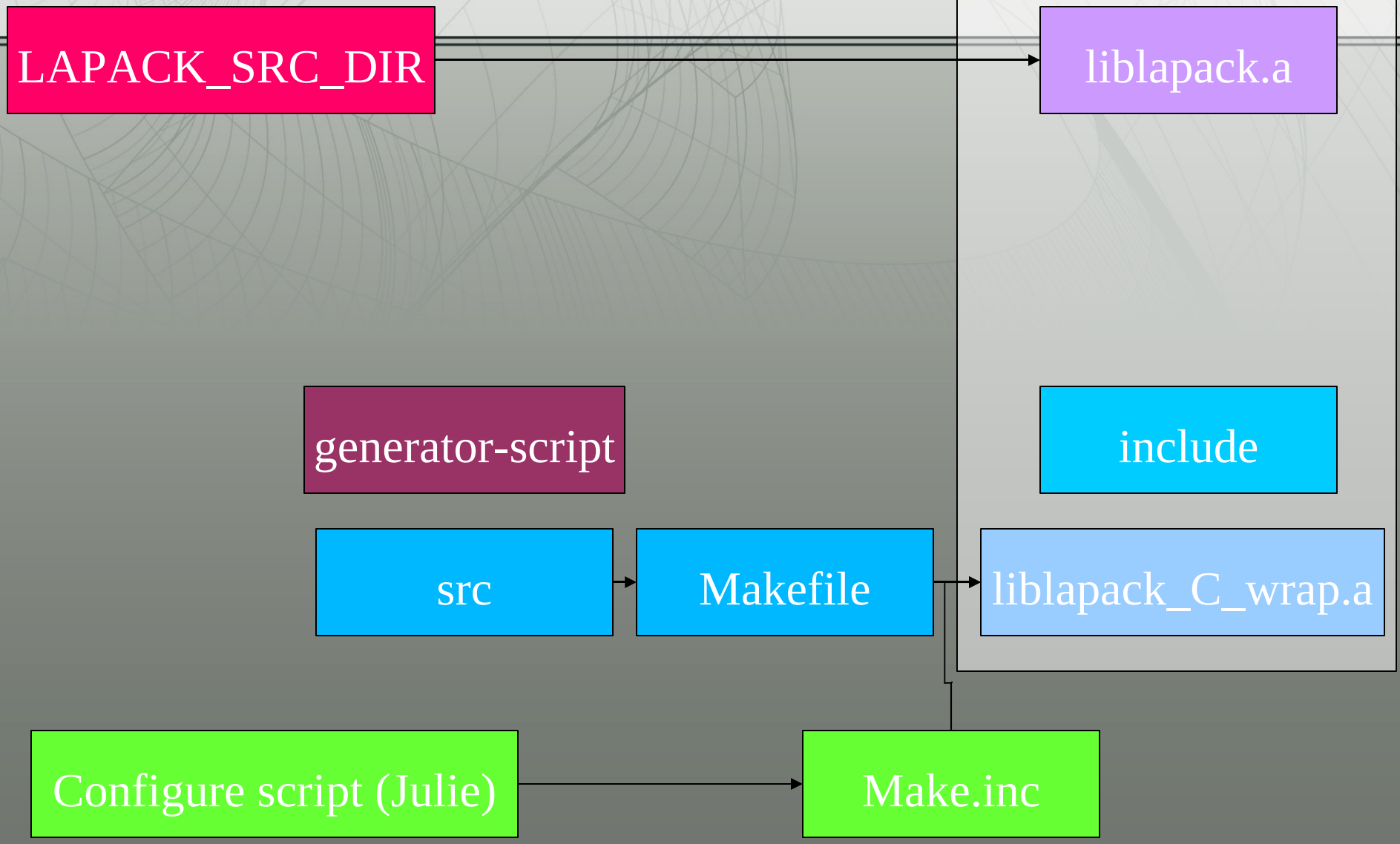
3. Design


LAPACK_SRC_DIR
liblapack.a
include
src
Makefile
liblapack_C_wrap.a
Make.inc

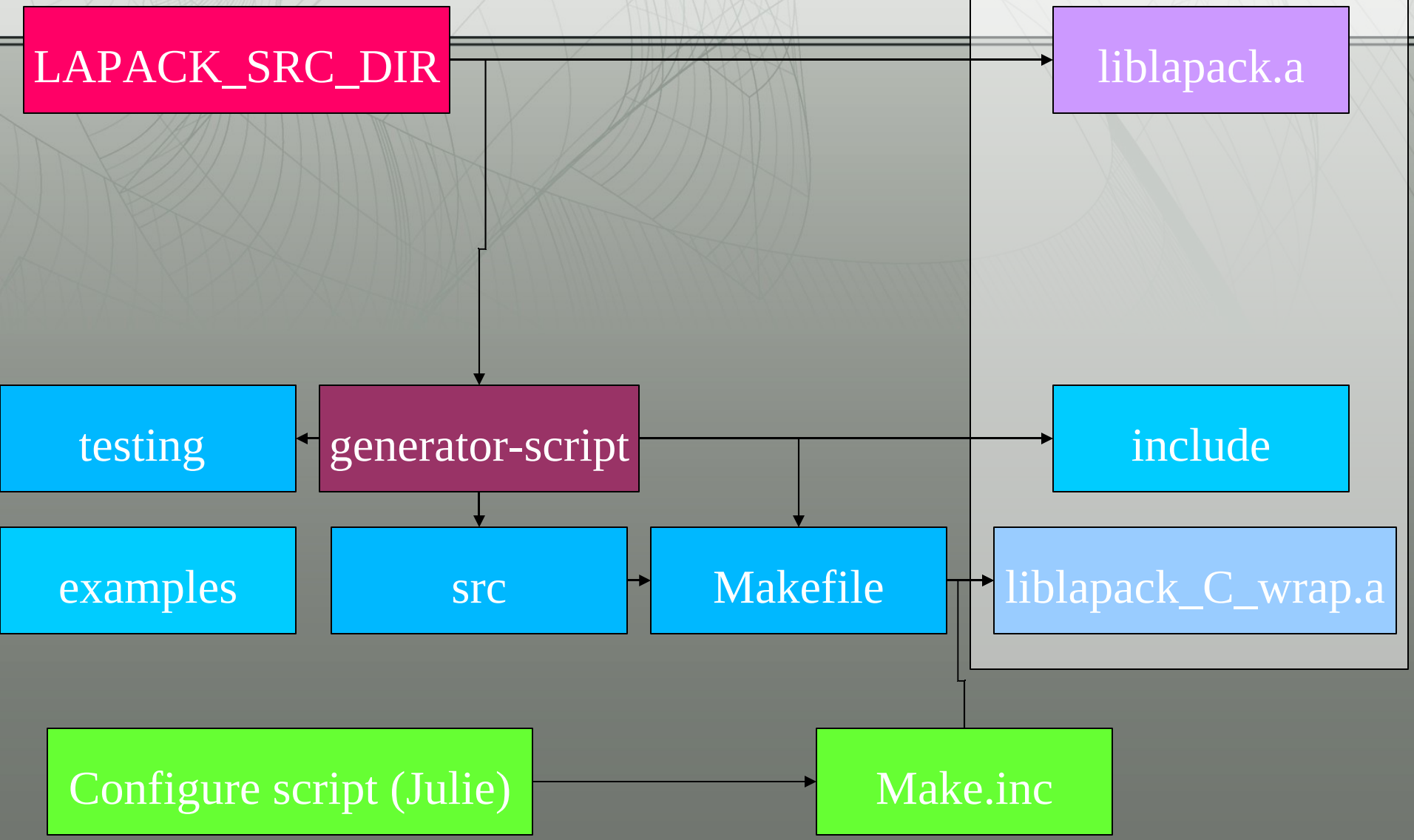
3. Design


LAPACK_SRC_DIR
liblapack.a
include
src
Makefile
liblapack_C_wrap.a
Configure script (Julie)
Make.inc

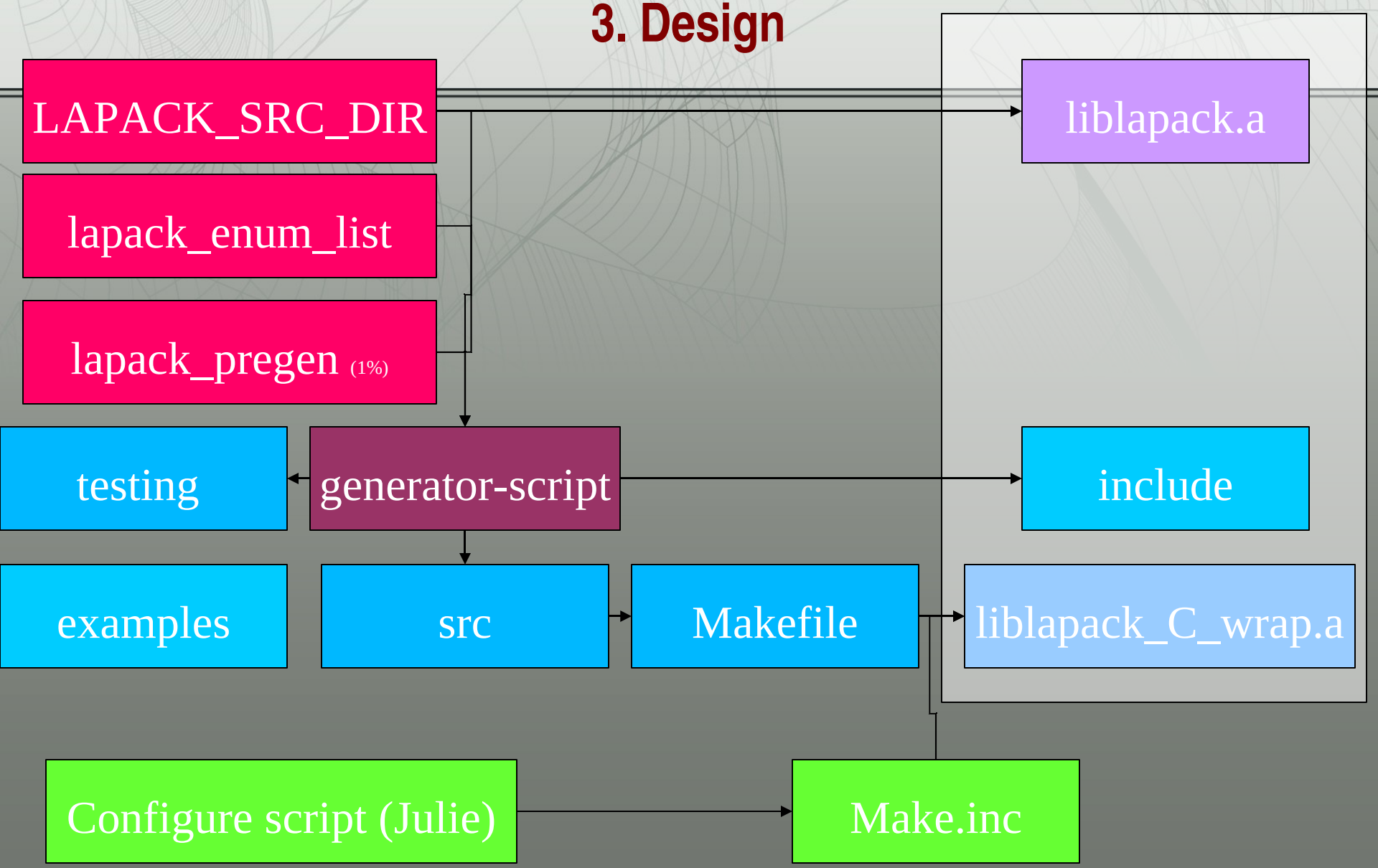
3. Design


LAPACK_SRC_DIR
liblapack.a
generator-script
include
src
Makefile
liblapack_C_wrap.a
Configure script (Julie)
Make.inc

3. Design


LAPACK_SRC_DIR
liblapack.a
testing
generator-script
include
examples
src
Makefile
liblapack_C_wrap.a
Configure script (Julie)
Make.inc

3. Design


LAPACK_SRC_DIR
liblapack.a
lapack_enum_list
lapack_pregen (1%)
testing
generator-script
include
examples
src
Makefile
liblapack_C_wrap.a
Configure script (Julie)
Make.inc

```
void lapack_cgesv(const int n, const int nrhs, void * a, const int lda,
                  int * ipiv, void * b, const int ldb, int * info )
{
#ifdef F77_INT
      int i_local;
      F77_INT F77_ipiv[n*1];
      for(i_local=0;i_local<n*1;i_local++) {
            F77_ipiv[i_local]=(F77_INT) ipiv[i_local]; }
#else
   #define F77_n n
      ...
#endif
f77_cgesv(&F77_n, &F77_nrhs, a, &F77_lda, F77_ipiv, b, &F77_ldb, F77_info);
...
}
```

3. Design :
3.a SRC
3.a.3 name mangling

```
/* lapack_enum.h */
enum lapack_uplo_type  {
        lapack_upper      = 121,
        lapack_lower      = 122,
        lapack_upper_lower = 123 };
```

```
/* lapack_f77.h */
#if defined(ADD_)
   #define f77_cgesv cgesv_
#elif defined(UPCASE)
   #define f77_cgesv CGESV
...
```

3. Design:
3.b: SCRIPT

Automatic generation

a) Based on the LAPACK comments section

c) Two steps

i. Used David Bindel's awk script to parse the LAPACK comments to extract useful information on the functions and their parameters. This information is written to a file as pseudo-code.

ii. Then a perl-script parses this file to generate the actual wrapper source.

d) Need correct and standardize LAPACK comment. If comments are wrong or not in the standard format, the parser fails.
=> Many many LAPACK comments were changed mostly for bugs (580 lines in 359 routines changed over 1293 routines).

e) The 2-level parsing allows for easily writing wrappers in other langages.

3. SCRIPT = standard comments

Enforcing LAPACK standardization in comments :

- If the norm is not respected, the script will fail, or worse, give false results.
- order of appearance of the variable in the comments must be the same than in the prototype
- lines begin with the variable name
- (input) (output) (input/output) (input or output) mandatorry
- type in upper case
- if array, put the word “array” somewhere on the line, along with the dimension between ().
- everything must be be on one line
- character of size 1 are noted character*1

3. SCRIPT = standard comments

Enforcing LAPACK standardization in comments :

- If the norm is not respected, the script will fail, or worse, give false results.
- order of appearance of the variable in the comments must be the same than in the prototype
- lines begin with the variable name
- (input) (output) (input/output) (input or output) mandatorry
- type in upper case
- if array, put the word “array” somewhere on the line, along with the dimension between ().
- everything must be be on one line
- character of size 1 are noted character*1

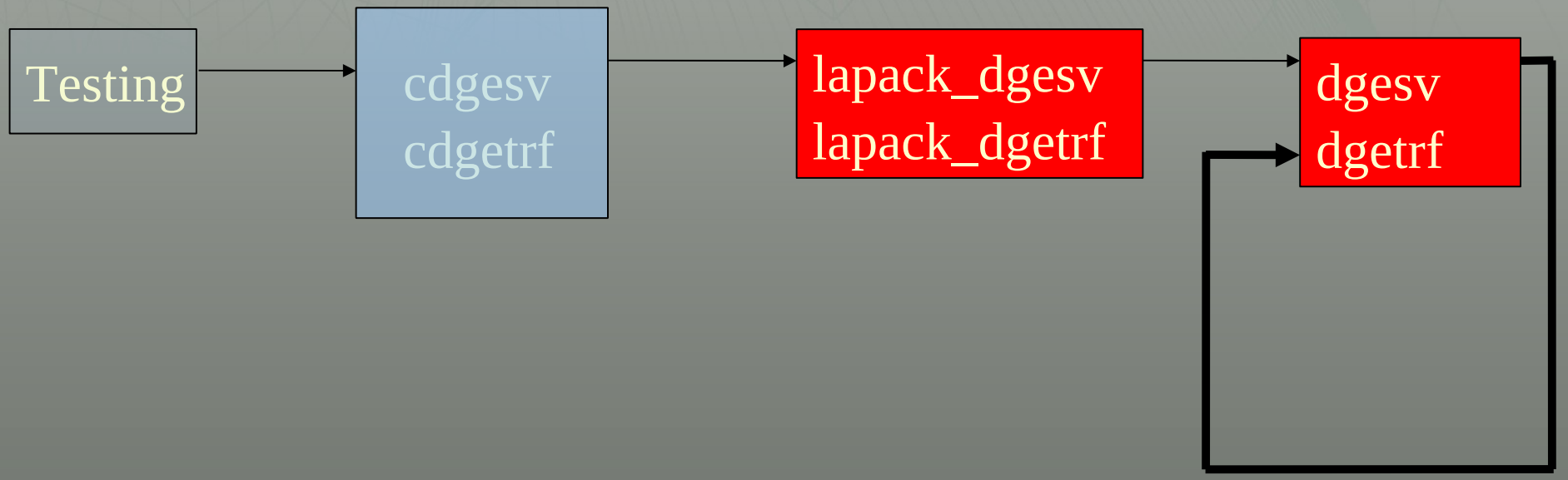
But SCRIPT = correct comments !!!!

4. Testing

- Two ways to test.
  - The easiest one : unwrap the wrapper ;) to allow for the testing suite provided in the LAPACK package to call the wrapper.
    - Problem : when a LAPACK function calls a LAPACK function, that call does not go through the wrapper (eg, dgesv will be tested but not dgetrf or dgetrs)
    - => Only the drivers are tested.
  - The “hard” one :
    - Change all LAPACK functions so that the internal calls are done through the wrapper.

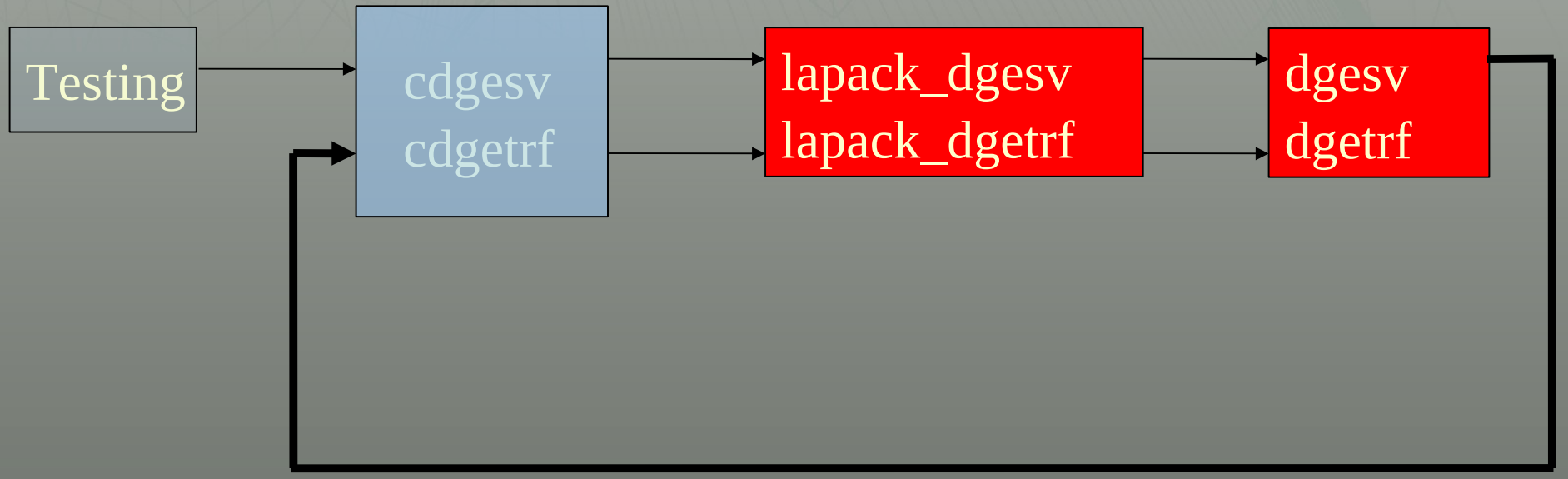
4. Testing

- Problem with first testing : only drivers are tested.



4. Testing

- Solution :hack lapack so that call to aux routines go through the wrapper.



4. Testing

- Some functions were awful. Best of :
  - Size of array that depend on other parameters (like a char)...
  - Size of array that depends on the result of some other function (how I am supposed to know that size?)...
  - Some inconsistensies in the CHARACTER*1 parameters :
    - In *ggsvd, JOBU = 'U' or 'N'
    - JOBV = 'V' or 'N'
    - JOBQ = 'Q' or 'N'
    - .... but they all have the same role !
  - Function pointers...
  - Differenciation of enums is based only on the parameter name. But JOB may have many meanings... Lots of hacks in the script...
  - Places were a character would be needed, but an integer is used.

4. Testing

? Wrapper = Performance issue ?

- Timing was not done, but for default settings the execution time should be more or less the same (function call)

- On the testing:
  - Our testing calls two wrappers (instead of one)
  - Small matrices are used (sometime no computation)
  - Recursive levels of wrapping (instead of only the driver in the general case)

  we are 1% more expensive than pure LAPACK

- The copy is a problem (cpu- and memory-wise), but we found no way to correct that.

Conclusion

- Complete automatic generation is an utopia, but we came close enough (thanks to some hacks...). Only 18 functions had to be done "*by hand*".
- We now have a working, portable, complete and tested C wrapper.
- Webpage : http://icl.cs.utk.edu/~delmas