
# Misc topics from Statoil

Joakim Hove

# Content

1. `ctypes` – a very convenient way to call C from Python.
2. `ert.ecl` - a ctypes based collection of Python classes to work with ECLIPSE binary files.
3. OPM – Eclipse Parser, an update from work in progress.
4. Writing ECLIPSE summary data from OPM; a discussion of what is needed.

# libffi – the basis of ctypes

libffi: A foreign function interface library for calling compiled (C) code.

- The information of the target function is determined/configured at *run time,* i.e. no need to create compiled custom wrappers like e.g. SWIG.

- Well suited for calling compiled C code from scripting languages. In addition to Python-ctypes the libffi library is employed in various packages in Java, Ruby, Lisp, JavaScript and Python.

# ctypes – getting the function handle

The basic use of ctypes is quite similar to a `dlopen()` and `dlsym()` based approach to loading modules:

```
#!/usr/bin/env python
import ctypes

libm = ctypes.CDLL('libm.so.6')

sin = getattr(libm , 'sin')

sin.argtypes = [ctypes.c_double]
sin.restype = ctypes.c_double

y = sin(1.57)
```

Load the shared library; on Posix platforms this is a thin wrapper

Look up the function (i.e. symbol) we are interested in. Functionally

'Decorate' the function object with result type and type of input arguments.

# ert – ‘OOP Pattern’

1. ‘Constructor‘ which allocates and initializes a structure.
2. ‘Destructor’ which frees all resources used by the structure.
3. Accesor functions to get and set (and calculate…) properties.
4. All functions take a pointer to an instance as first argument.

```
ecl_sum_type * ecl_sum_alloc(...) {
   ecl_sum_type * sum = malloc( sizeof * sum );
   // Initialize sum instance
   return sum;
}


void ecl_sum_free( ecl_sum_type * sum ) {
    // Free members of the sum struct.
    free( sum );
}


double ecl_sum_get_well_var( const ecl_sum_type * sum, const char * well, const char * var, time_t t) {
    ...
    return ;
}

bool ecl_sum_has_well_var( const ecl_sum_type * sum, const char * well, const char * var) {
     ...
     return ?;
}
```



Statoil

# Wrapping it in Python (simplified …)

```
class EclGrid:
   @classmethod
   def from_param(cls , obj):
       return obj.c_ptr

   def __init__(self , grid_file):
       self.c_ptr = cfunc.alloc_grid( grid_file )

   def dims(self):
       nx = cfunc.get_nx( self )
       ny = cfunc.get_ny( self )
       nz = cfunc.get_nz( self )
         return (nx,ny,nz)
   ...

lib = ctypes.CDLL( 'libecl.so' )
cfunc.alloc_grid = cwrap.prototype(' c_void_p ecl_grid_alloc( char* )')
cfunc.get_nx = cwrap.prototype('int ecl_grid_get_nx( ecl_grid )')
cfunc.get_ny = cwrap.prototype('int ecl_grid_get_ny( ecl_grid )')
```

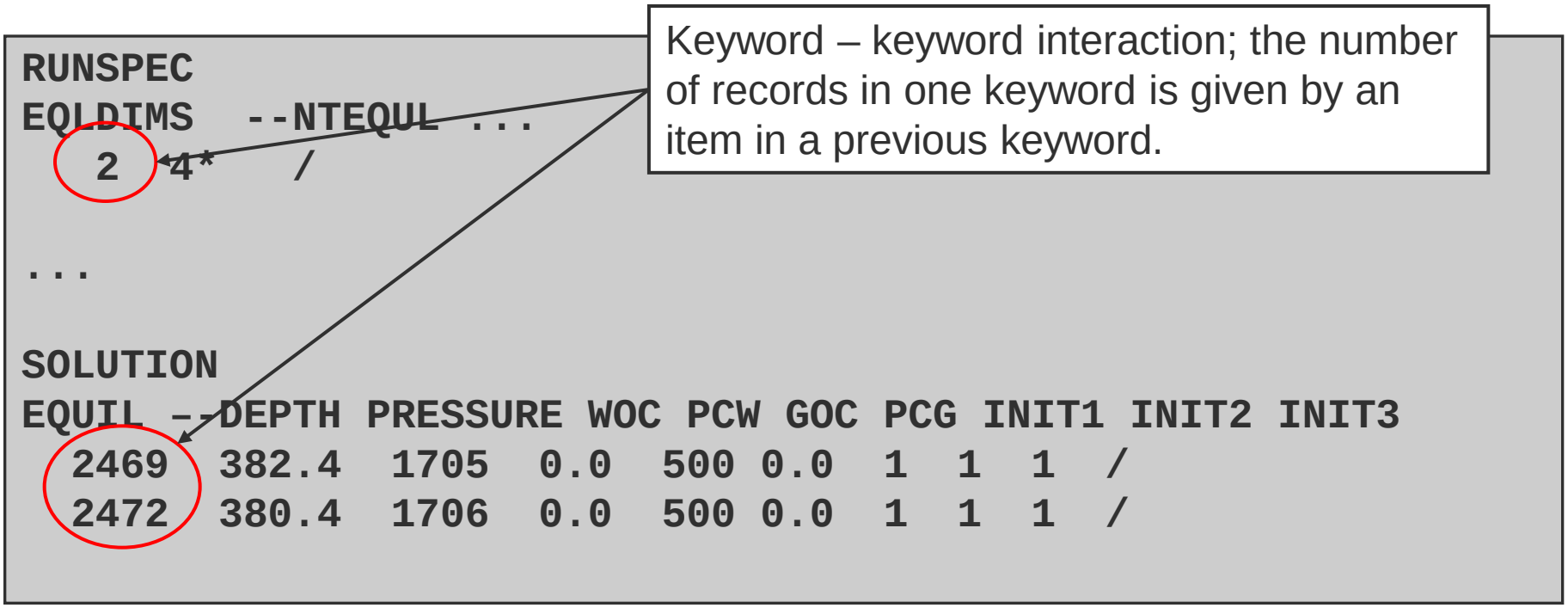
# Example I – grid, file, kw and region

```
#!/usr/bin/env python
import ert.ecl.ecl as ecl
import sys

case = sys.argv[1]
grid = ecl.EclGrid(case)
init = ecl.EclFile('%s.INIT' % case)
region = ecl.EclRegion( grid , False)

# Some grid properties
print 'Grid dimensions: (%d,%,%d) Active:%d' % (grid.nx , grid.ny , grid.nz , grid.nactive)


# Load properties from INIT file
permx = init['PERMX'][0]
poro = init['PORO'][0]

# Select all cells with porosity less than 0.025
region.select_less( poro , 0.025)

# Set PERMX -> 0 for all selected cells
permx.assign(0 , mask = region)

# Write updated permx input file:
fileH = open('permx.grdecl','w')
grid.write_grdecl( permx , fileH)
```

# Example II - summary

```
#!/usr/bin/env python
import ert.ecl.ecl as ecl
import sys

case = sys.argv[1]
sum = ecl.EclSum(case)

# Print all wells matching the glob: *O*:
for well in sum.wells( pattern='*O*'):
   print well

# Time direction
print 'Simulation timespan: %s - %s' % (sum.start_date , sum.end_date)

# Iterate through time direction for key WWCT:OP_1
for node in sum['WWCT:OP_1']:
   print '%8.3f  %g' % (node.days , node.value)


# Look up FOPT at a interpolated dates
date_list = []
for year in range(sum.start_date.year , sum.end_data.year):
    date_list.append( datetime.date( year , 1 , 1)
    date = datetime.date( year , 7 , 1)
    if sum.check_sim_time( date ):
        date_list.append( date )

for date in date_list:
    print '%s %g' % (date , sum.get_interp('FOPT' , date = date)
```

# OPM ECLIPSE Parser - keyword interactions





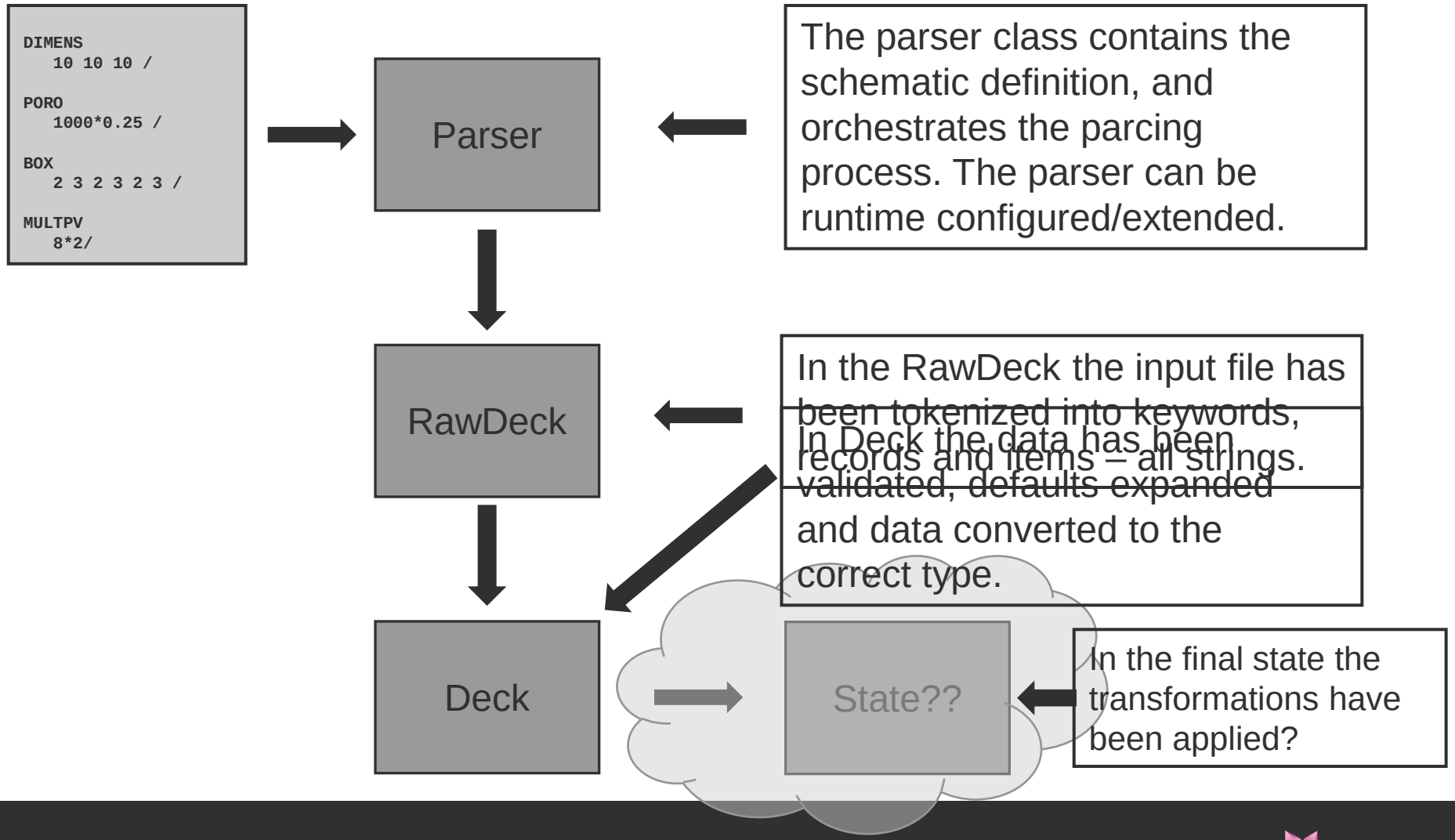

Statoil

# Masterdata?

```
-- Grid dimensions 10, 10, 10
DIMENS
   10 10 10 /

-- Porosity; 1000 float values in normal order.
PORO
   1000*0.25 /

-- Restrict the input box to the cube [2,3] x [2,3] x [2,3]
BOX
   2 3 2 3 2 3 /

-- Multiply the pore volume in the current input box with 2.
MULTPV
   8*2/
```

What is now the porosity in cell [2,2,2]?

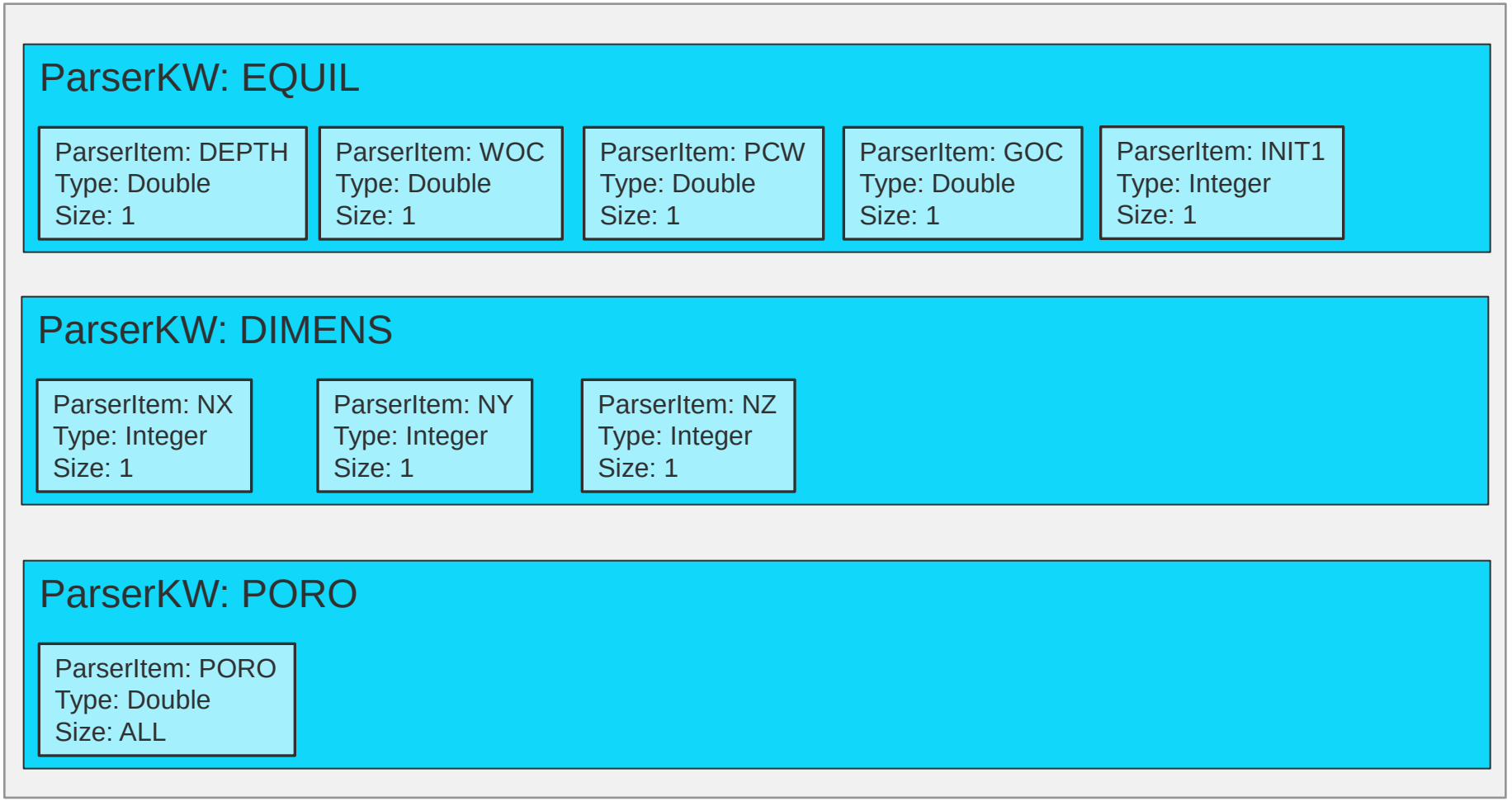
# OPM Parser - architecture






Statoil

# Parser






Statoil

# RawDeck

## RawKeyword: EQUIL

RawRecord: {‘**2469’, ‘382.4’, ‘1705’, ‘0.0’, ‘500’, ‘0.0’, ‘1’, ‘1’, ‘1’}**

RawRecord: {‘**2472’, ‘385.4’, ‘1706’, ‘0.0’, ‘500’, ‘0.0’, ‘1’, ‘1’, ‘1’}**

## RawKeyword: DIMENS

RawRecord: {’10’, ‘10’, ‘10’**}**

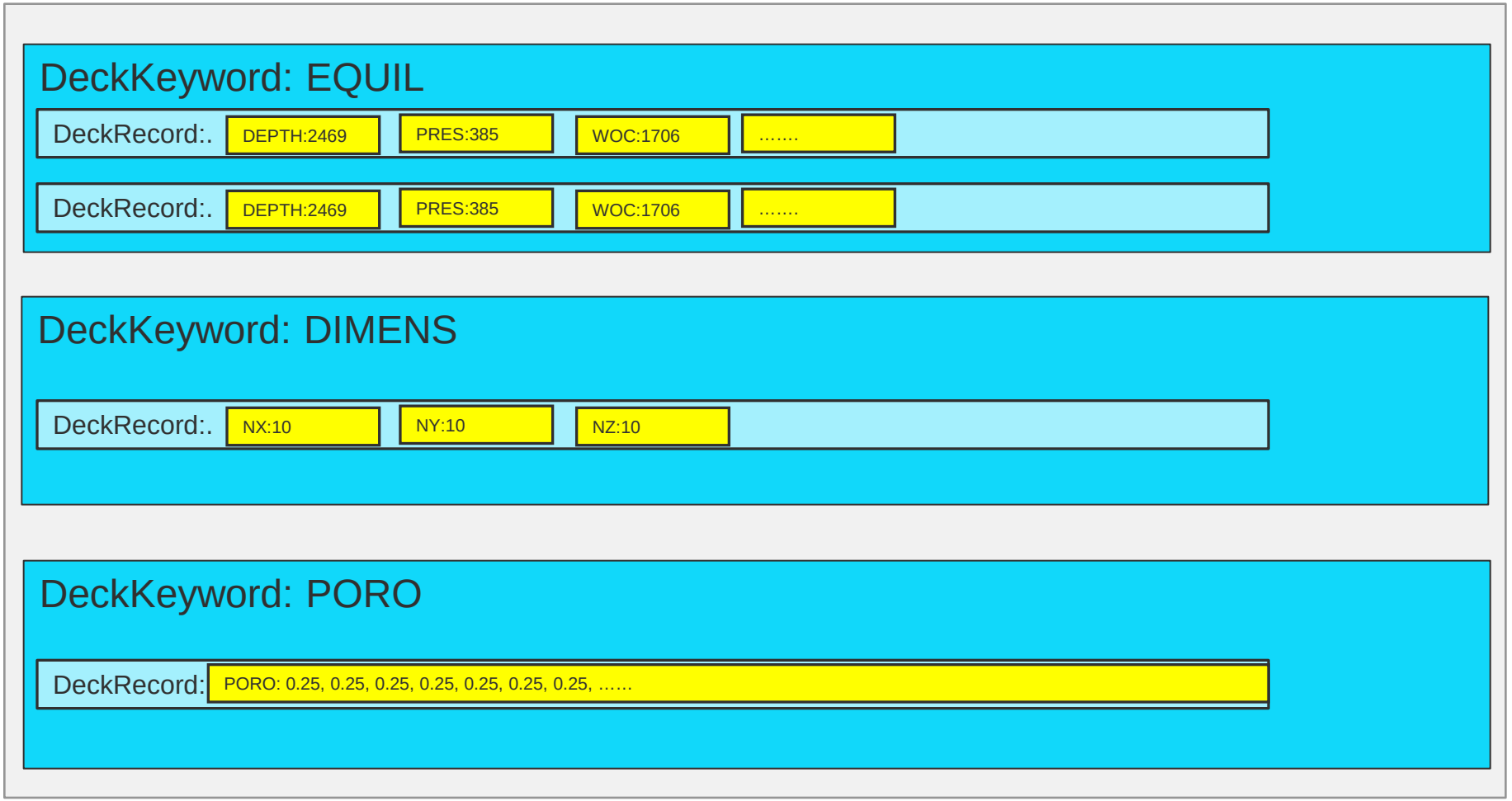
## RawKeyword: PORO

RawRecord: {’1000*0.25’**}**



Statoil

# Deck






Statoil

# Topics for discussion

1. How to relate to transformations like `MULTPV`?
2. Something completely different: How to support writing of ECLIPSE summary files from OPM?

There's never been a better time for good ideas

Presentation title

Presenters name
Presenters title
E-mail address ……@statoil.com
Tel: +4700000000

www.statoil.com



Statoil