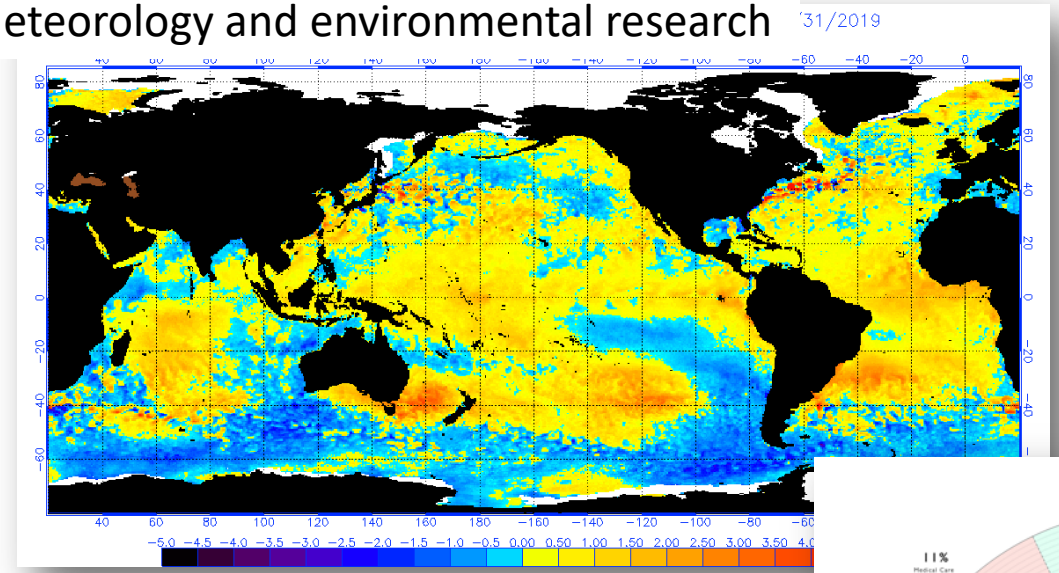
# xarray: A meaningful way of working with high-dimensional scientific data

**Diego Alonso-Álvarez and Mayeul d'Avezac de Castera**
Research Computing Service, Imperial College London

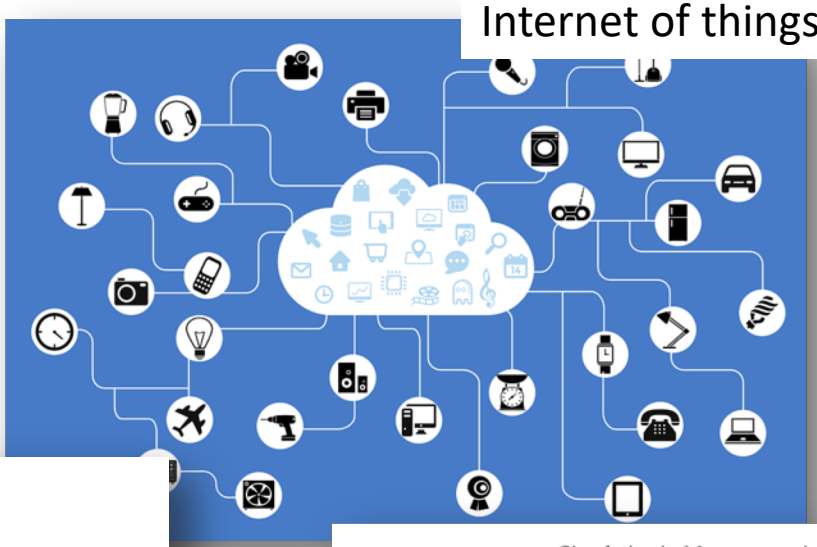
Imperial College London

RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

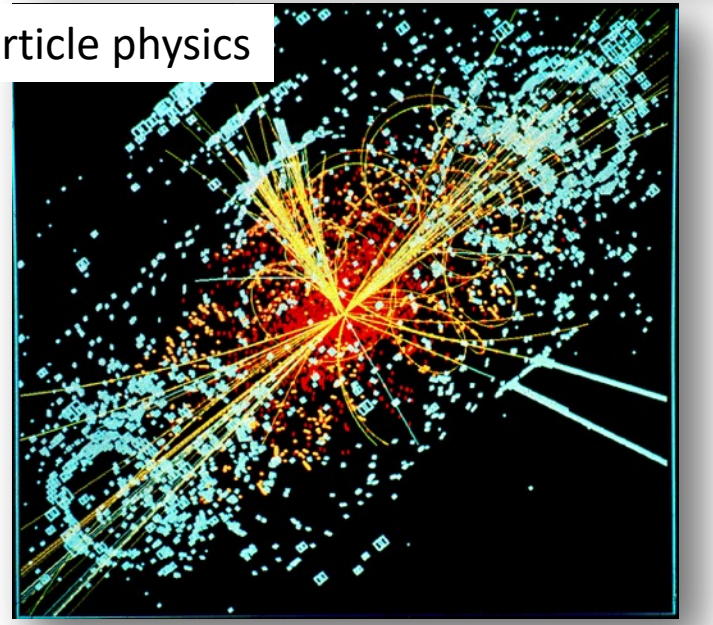
Meteorology and environmental research

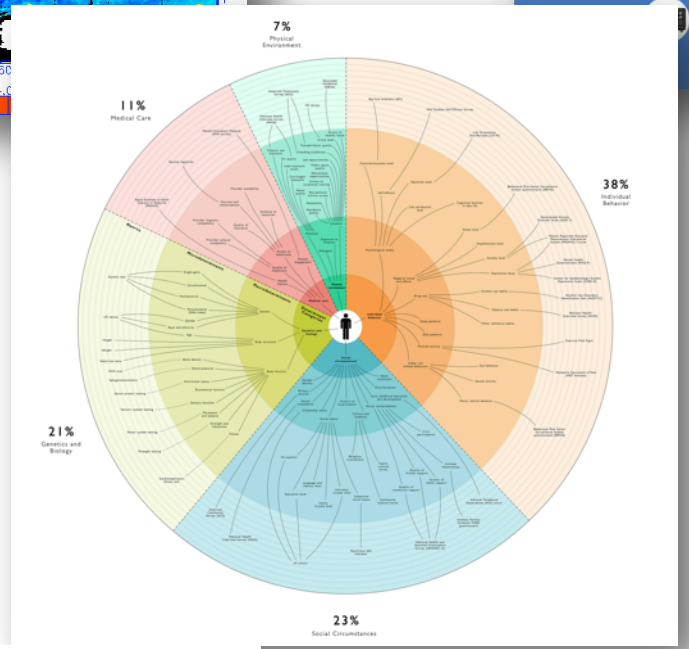



Internet of things

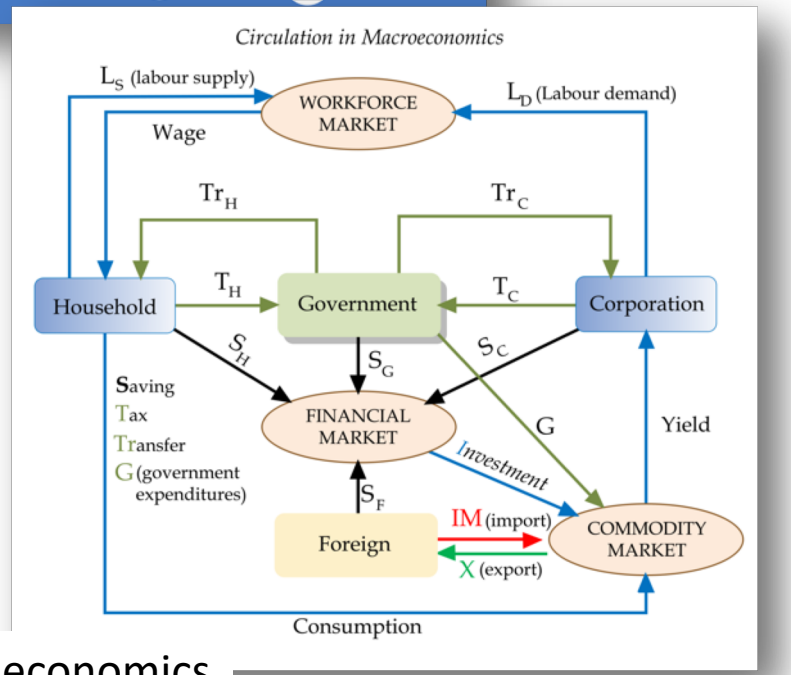


Particle physics






Social studies

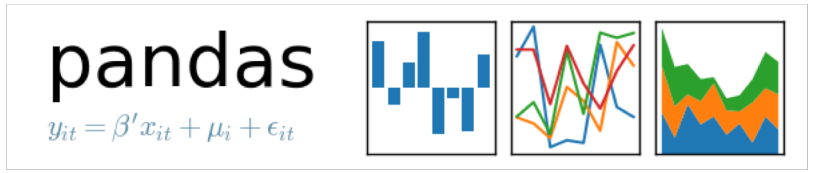



Macroeconomics

# What is it?

Python library for high-dimensional data manipulation

http://xarray.pydata.org



*Does not do anything new, but makes many tasks* ***<u>much easier to code and understand</u>***

# What is it?

- **DataArray** → Multi-dimensional array with label dimensions, coordinates and attributes (n-dimensional pandas Series)
- **Dataset** → Dictionary of DataArrays that share some dimensions (equivalent to pandas Dataframe)

  - Labelled dimensions, coordinates and data (arrays and attributes)
  - Accessing and selecting data using labels instead of integers
  - Array broadcasting for mathematical operations based on labels
  - Built-in array manipulation routines (eg. interpolation) across full datasets
  - Built-in statistical functions (eg. groupby, sdt, mean…)
  - Easy conversion pandas ←→ xarray
  - Parallel computation with Dask
  - Built-in serialization to multiple formats (eg. netCDF, Iris, rasterio, zarr, OPeNDAP…)

# What is it? - DataArrays

```
arr = numpy.random.random((3, 2, 2))
da = xarray.DataArray(arr,  dims=('x', 'y', 'time'),
                      coords={'x': [0, 0.1, 0.3], 'y': [ 0, 1],
                              'time': [datetime(2019, 1, 1), datetime(2019, 3, 1)]},
                      attrs={'user_name': 'Meriadoc Brandybuck',
                             'contry': 'The Shire' })
```

```
<xarray.DataArray (x: 3, y: 2, time: 2)>
array([[[0.928411, 0.785756],
        [0.422931, 0.643851]],

       [[0.50493 , 0.108378],
        [0.623028, 0.557061]],

       [[0.758997, 0.795663],
        [0.155889, 0.079399]]])
Coordinates:
  * x        (x) float64 0.0 0.1 0.3
  * y        (y) int64 0 1
  * time     (time) datetime64[ns] 2019-01-01 2019-03-01
Attributes:
    user_name:  Meriadoc Brandybuck
    contry:     The Shire
```

Labelled dimensions

Data

Labelled coordinates

Attributes

# What is it? - Datasets

```
ds = xarray.Dataset({"big_array": da1, "small_array" : da2})
```

```
<xarray.Dataset>
Dimensions:      (time: 2, x: 3, y: 2)
Coordinates:
  * x            (x) float64 0.0 0.1 0.3
  * y            (y) int64 0 1
  * time         (time) datetime64[ns] 2019-01-01 2019-03-01
Data variables:
    big_array    (x, y, time) float64 0.7248 0.5858 0.0194 ... 0.4288 0.3806
    small_array  (y, time) float64 0.04727 0.4738 0.7081 0.3678
```

# xarray vs numpy (1)

**Accessing and selecting data**: Using labels instead of integer locations

```
da[:, :, 1]
da.loc[:, :, '2019-03-01']
da.sel(time='2019-03-01')
```

```
<xarray.DataArray (x: 3, y: 2)>
array([[0.684902, 0.231581],
       [0.305218, 0.258243],
       [0.42951 , 0.161443]])
Coordinates:
  * x        (x) float64 0.0 0.1 0.3
  * y        (y) int64 0 1
    time     datetime64[ns] 2019-03-01
```

```
da[1, :, 1]
da.loc[0.1, :, '2019-03-01']
da.sel(time='2019-03-01', x=0.1)
```

```
<xarray.DataArray (y: 2)>
array([0.305218, 0.258243])
Coordinates:
    x        float64 0.1
  * y        (y) int64 0 1
    time     datetime64[ns] 2019-03-01
```

**Built-in multidimensional interpolation**

**… also works with full Datasets!**

```
da.interp(time='2019-02-01', x=[0.05, 0.15])
```

```
<xarray.DataArray (x: 2, y: 2)>
array([[0.572384, 0.398565],
       [0.426596, 0.577036]])
Coordinates:
  * y        (y) int64 0 1
    time     datetime64[ns] 2019-02-01
  * x        (x) float64 0.05 0.15
```

# xarray vs numpy (2)

**Array broadcasting**: Operations vectorized across multiple dimensions based on dimension names, not shape.

numpy

```
x = np.random.random((3, 2))
y = x.T
x*y
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-53-a08e63d571f2> in <module>
      1 x = np.random.random((3, 2))
      2 y = x.T
----> 3 x*y

ValueError: operands could not be broadcast together with shapes (3,2) (2,3)
```

xarray

```
xda = xr.DataArray(x, dims=['A', 'B'])
yda = xda.T
xda*yda
```

```
<xarray.DataArray (A: 3, B: 2)>
array([[1.485224e-01, 4.248256e-04],
       [1.653692e-01, 1.948700e-01],
       [6.352797e-01, 2.609216e-02]])
Dimensions without coordinates: A, B
```

Many dimensions:

```
x = np.random.random((3, 2, 5, 2, 7))
y = x.transpose((1, 2, 0, 3, 4))
x*y
```

```
xda = xr.DataArray(x, dims=['A', 'B', 'C', 'D', 'E'
yda = xda.transpose('B', 'C', 'A', 'D', 'E')
xda*yda
```

# xarray vs pandas

True n-dimensional data structures, indexing and manipulation

**pandas**

```
iterables = [['bar', 'baz', 'foo'], ['one', 'two'], ['cat', 'dog']]
index = pd.MultiIndex.from_product(iterables, names=['first', 'second', 'third'])
ds = pd.Series(np.random.randn(12), index=index)
```

```
first  second  third
bar    one     cat     -0.374588
               dog     -0.690487
       two     cat     -0.324122
               dog      0.851776
baz    one     cat     -1.832249
               dog     -0.620590
       two     cat     -0.662045
               dog     -1.232905
foo    one     cat      1.147498
               dog      0.106812
       two     cat     -1.902024
               dog      0.613718
dtype: float64
```

## Selecting data by label

**pandas**

```
ds.loc['bar', 'two', 'dog']    # Scalar value
```

**xarray**

```
da.loc['bar', 'two', 'dog']    # 0-D DataArray
```

## Selecting data by index

**pandas**

```
ds[3]              # Scalar value
```

**xarray**

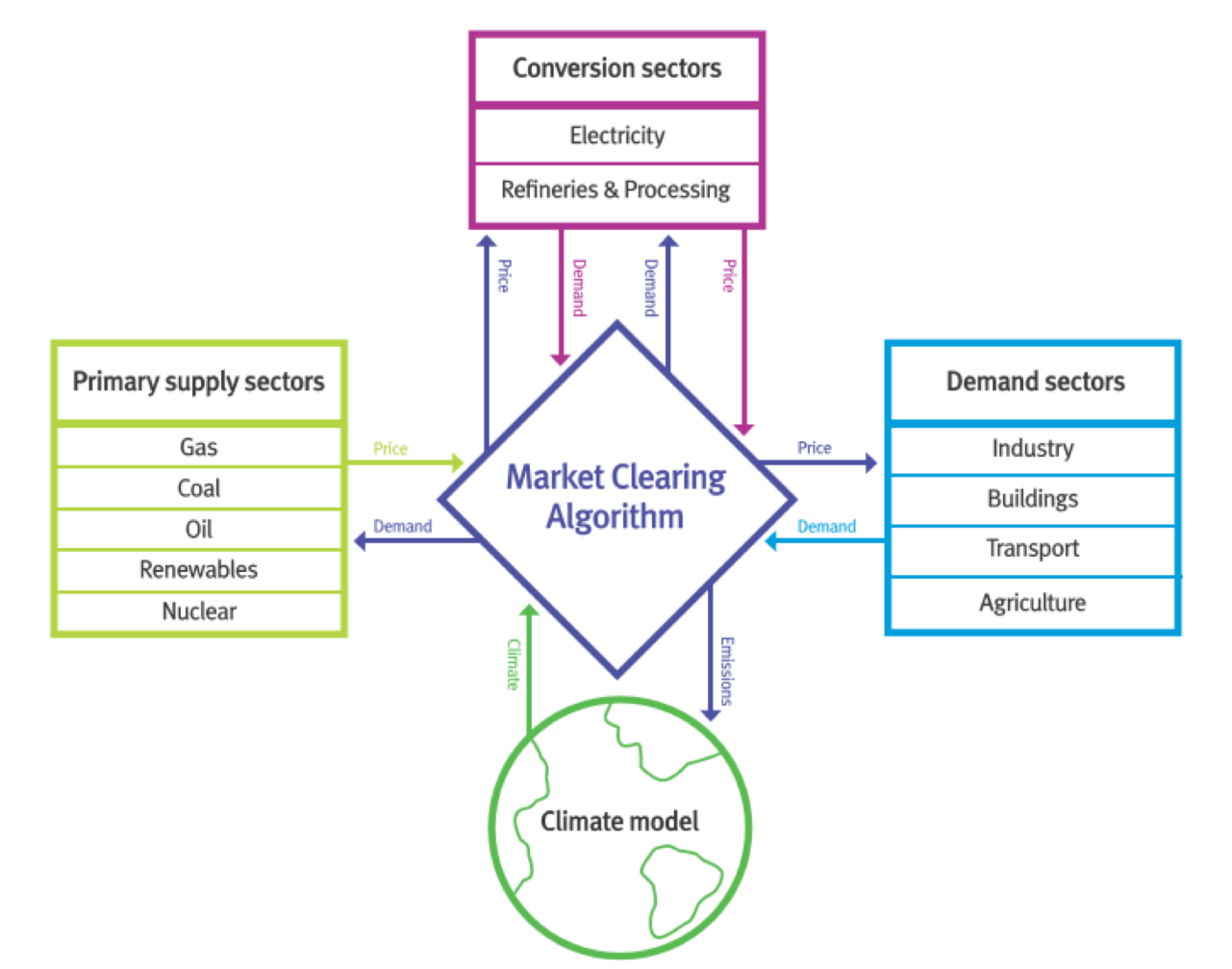
```
da[0, 1, 1]        # 0-D DataArray
```

Transpose, matrix multiplication…

**xarray**

```
arr = numpy.random.random((3, 2, 2))
da = xarray.DataArray(arr,  dims=('first', 'second', 'third'),
                      coords={'first': ['bar', 'baz', 'foo'],  'second': ['one', 'two'],
                              'third': ['cat', 'dog']})
ds = da.to_series()
```

# Example: MUSE - ModUlar energy system Simulation Environment




- Using a combination of numpy and pandas
- Multiple up to 5-D arrays
- Up to 7 nested for loops
- Need of extra 1-D arrays to keep track of the indices of interest in the n-D ones
- Slow to run, hard to maintain and expand

Vectorizing the code using **numpy arrays**

**Too complex and obscure!**

Vectorizing the code using **xarrays**

**Intuitive and self-describing**

www.sustainablegasinstitute.org/home/muse-energy-model/

Open source release planned for this year!

# Example: MUSE - ModUlar energy system Simulation Environment

4 dimensions

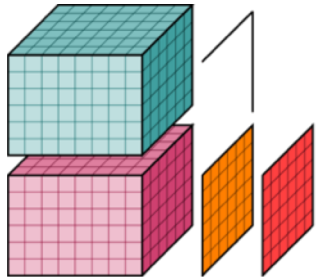
9 Coordinates

29 DataArrays

```
<xarray.Dataset>
Dimensions:                                (commodity: 77, region: 28, technology: 64, year: 2)
Coordinates:
  * technology                             (technology) object 'resAC' ... 'resNGSA'
  * region                                 (region) object 'ASEAN' 'ATE' 'AUS' ... 'USA' 'ZAF'
  * year                                   (year) int64 2010 2030
    tech_type                              (technology) object 'appl' 'heatp' ... 'standA'
    fuel                                   (technology) object 'elec' 'elec' ... 'ng' 'ng'
    enduse                                 (technology) object 'cspace' 'hspace' ... 'hspace'
  * commodity                              (commodity) <U11 'algae' 'agrires' ... 'marinef'
    comm_name                              (commodity) object 'Algae' ... 'Marine_freight'
    comm_type                              (commodity) object 'energy' 'energy' ... 'service'
Data variables:
    level                                  (technology, region, year) object 'fixed' ... 'fixed'
    cap_par                                (technology, region, year) float64 8.976 ... 9.756
    cap_exp                                (technology, region, year) int64 1 1 1 1 ... 1 1 1 1
    fix_par                                (technology, region, year) float64 0.4488 ... 0.9756
    fix_exp                                (technology, region, year) int64 1 1 1 1 ... 1 1 1 1
        …
    agent_share_1                          (technology, region) int64 0 0 0 0 0 0 ... 0 0 0 0 0
    agent_share_2                          (technology, region) int64 0 0 0 0 0 0 ... 0 0 0 0 0
    agent_share_3                          (technology, region) float64 0.15 0.1 ... 0.18 0.1
        …
    fixed_outputs                          (commodity, technology, region, year) float64 0.0 ... 0.0
    flexible_outputs                       (commodity, technology, region, year) float64 0.0 ... 0.0
        …
    emmission_factor                       (commodity) float64 112.0 112.0 100.0 ... 0.0 0.0 0.0
    heat_rate                              (commodity) int64 1 1 1 1 1 1 1 1 ... 1 1 1 1 1 1 1 1
```

# Conclusions

- xarray: Python package for working with high-dimensional data
- Simplify accessing and manipulating the data by using meaningful labels rather than indices, including array broadcasting and interpolation
- Fully integrated with the SciPy ecosystem (pandas, numpy, scipy…)
- Makes the code:
  - easier to write
  - easier to understand
  - easier to maintain



xarray

Many dimensions:

❌
```
x = np.random.random((3, 2, 5, 2, 7))
y = x.transpose((1, 2, 0, 3, 4))
x*y
```

✅
```
xda = xr.DataArray(x, dims=['A', 'B', 'C', 'D', 'E'
yda = xda.transpose('B', 'C', 'A', 'D', 'E')
xda*yda
```

```
<xarray.Dataset>
Dimensions:              (commodity: 77, region: 28, technology: 64, year: 2)
Coordinates:
  * technology           (technology) object 'resAC' ... 'resNGSA'
  * region               (region) object 'ASEAN' 'ATE' 'AUS' ... 'USA' 'ZAF'
  * year                 (year) int64 2010 2030
    tech_type            (technology) object 'appl' 'heatp' ... 'standA'
    fuel                 (technology) object 'elec' 'elec' ... 'ng' 'ng'
    enduse               (technology) object 'cspace' 'hspace' ... 'hspace'
  * commodity            (commodity) <U11 'algae' 'agrires' ... 'marinef'
    comm_name            (commodity) object 'Algae' ... 'Marine_freight'
    comm_type            (commodity) object 'energy' 'energy' ... 'service'
Data variables:
    level                (technology, region, year) object 'fixed' ... 'fixed'
    cap_par              (technology, region, year) float64 8.976 ... 9.756
    cap_exp              (technology, region, year) int64 1 1 1 1 ... 1 1 1 1
    fix_par              (technology, region, year) float64 0.4488 ... 0.9756
    fix_exp              (technology, region, year) int64 1 1 1 1 ... 1 1 1 1
        …
    agent_share_1        (technology, region) int64 0 0 0 0 0 0 ... 0 0 0 0 0
    agent_share_2        (technology, region) int64 0 0 0 0 0 0 ... 0 0 0 0 0
    agent_share_3        (technology, region) float64 0.15 0.1 ... 0.18 0.1
        …
    fixed_outputs        (commodity, technology, region, year) float64 0.0 ... 0.0
    flexible_outputs     (commodity, technology, region, year) float64 0.0 ... 0.0
        …
    emmission_factor     (commodity) float64 112.0 112.0 100.0 ... 0.0 0.0 0.0
    heat_rate            (commodity) int64 1 1 1 1 1 1 1 1 ... 1 1 1 1 1 1 1 1
```

# Any questions?

**Contact:**

d.alonso-alvarez@imperial.ac.uk
http://imperial.ac.uk/ict/rcs
@ImperialRSE

**The RSE team at Imperial:**

- Mark Woodbridge
- Mayeul d'Avezac de Castera
- Diego Alonso-Álvarez

**Acknowledgments:**

Imperial College London

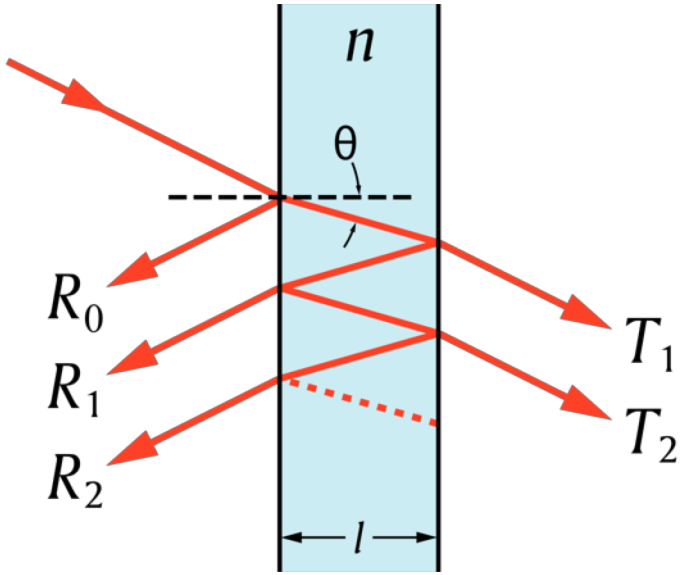
RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

SUSTAINABLE GAS INSTITUTE

NUMFOCUS
OPEN CODE = BETTER SCIENCE

# Example: TMM solver in Solcore

Transfer matrix method → light absorption in solar cells




4 dimensions (excluding angle)

2 dimensions (excluding angle)

3 dimensions (excluding angle)

```
M_list = zeros((num_layers, num_wl, 2, 2), dtype=complex)
for i in range(1, num_layers - 1):
    A = make_2x2_array(exp(-1j * delta[i]), np.zeros_like(delta[i]), np.zeros_like(delta[i]), exp(1j * delta[i]),
                       dtype=complex)
    B = make_2x2_array(np.ones_like(delta[i]), r_list[:, i, i + 1], r_list[:, i, i + 1], np.ones_like(delta[i]),
                       dtype=complex)
    d = (1 / t_list[:, i, i + 1])

    M_list[i] = np.transpose(d * np.transpose(np.matmul(A, B

Mtilde = make_2x2_array(np.ones_like(delta[i]), np.zeros_like(delta[i]), np.zeros_like(delta[i]),
                        np.ones_like(delta[i]), dtype=complex)
for i in range(1, num_layers - 1):
    Mtilde = np.matmul(Mtilde, M_list[i])

A = make_2x2_array(np.ones_like(delta[i]), r_list[:, 0, 1], r_list[:, 0, 1], np.ones_like(delta[i]),
                   dtype=complex)
d = 1 / t_list[:, 0, 1]
Mtilde = np.matmul(np.transpose(d * np.transpose(A, (1, 2, 0)), (2, 0, 1)), Mtilde)
```

Non-trivial transpose operations

# Example: TMM solver in Solcore

```
<xarray.Dataset>
Dimensions:           (Angle: 50, Depth: 1000, Interface: 4, Layer: 5, Wavelength: 50)
Coordinates:
  * Wavelength        (Wavelength) float64 300.0 314.3 328.6 ... 985.7 1e+03
  * Angle             (Angle) float64 0.0 1.837 3.673 5.51 ... 86.33 88.16 90.0
  * Interface         (Interface) <U11 'interface 0' ... 'interface 3'
  * Layer             (Layer) <U7 'layer 0' 'layer 1' ... 'layer 3' 'layer 4'
  * Depth             (Depth) float64 0.0 5.005 10.01 ... 4.995e+03 5e+03
Data variables:
    R                 (Wavelength, Angle, Interface) float64 0.2741 ... 0.3967
    T                 (Wavelength, Angle, Interface) float64 0.3838 ... 0.8027
    A                 (Wavelength, Angle, Layer) float64 0.1349 0.8721 ... 0.611
    AbsorptionDepth   (Wavelength, Angle, Depth) float64 0.6213 ... 0.07605
    E_s               (Wavelength, Angle, Depth) float64 0.8104 0.8837 ... 0.8792
    E_p               (Wavelength, Angle, Depth) float64 0.6544 0.8131 ... 0.4288
    R_total           (Wavelength, Angle) float64 0.2458 0.6138 ... 0.09802
    T_total           (Wavelength, Angle) float64 0.3013 0.04771 ... 0.8217
    A_total           (Wavelength, Angle) float64 0.6077 0.08734 ... 0.09917
    n                 (Wavelength, Layer) float64 0.9553 0.507 ... 0.6408 0.1651
```

- No need to keep track of the order/meaning of the indices
- No need to include complex transpose operations