NCAR/CISL/TDD

air • planet • people

Parallel Python Tools for Handling Big Climate Data

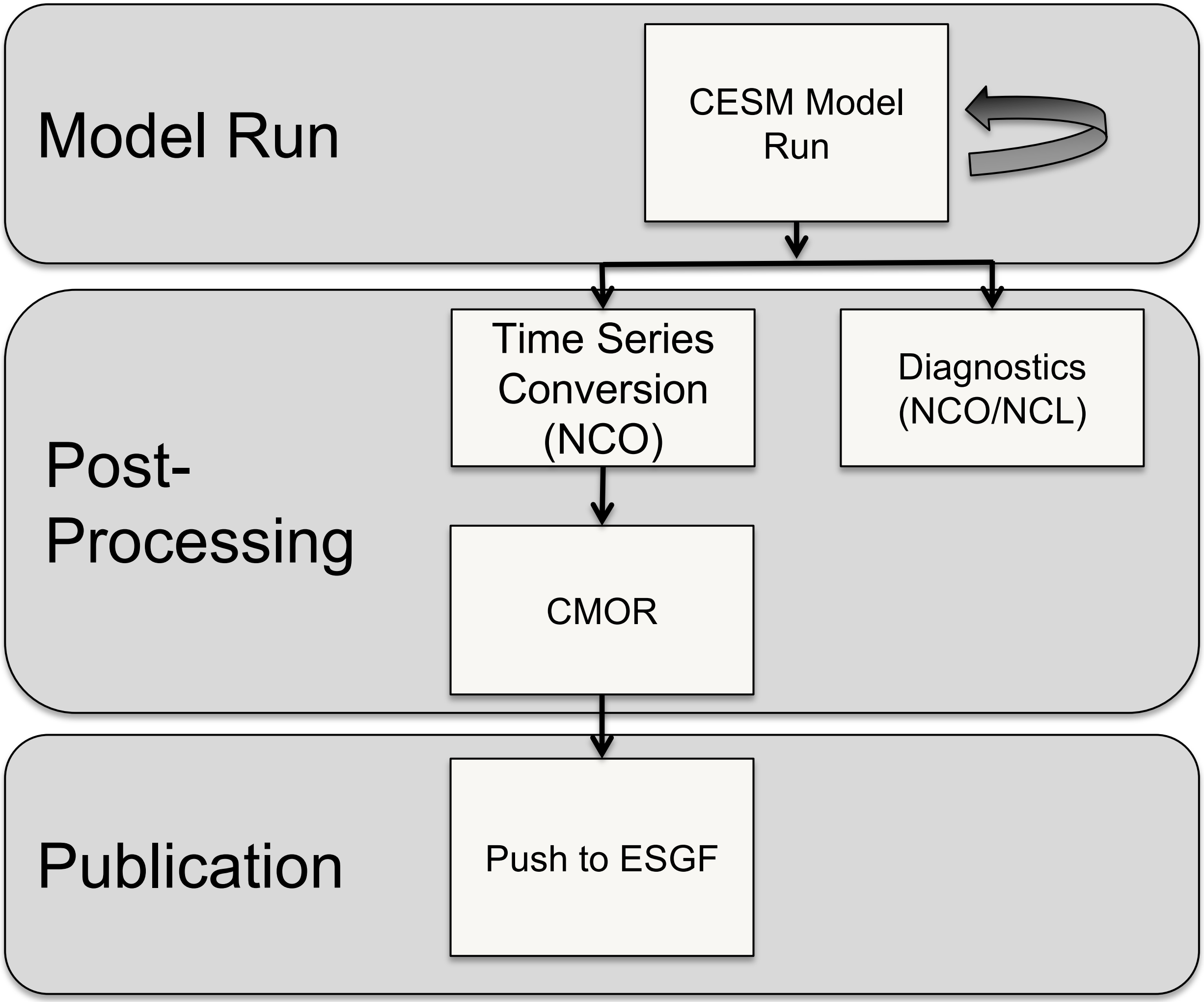
Sheri Mickelson

Kevin Paul

2018 Workshop on Developing Python Frameworks for Earth System Sciences

October 30, 2018

CESM's CMIP5 Workflow



Lessons We Learned From CMIP5

CESM was the first model to complete their simulations, but the last to complete publication.

Why?

- All of the post-processing was serial and it took a long time to run
- Workflow was error prone and was time consuming to debug
- Too much human intervention was needed between post-processing steps and time was wasted
- There was only one person who knew the status of all of the experiments

Motivating Factor
CMIP5 vs CMIP6

CMIP5

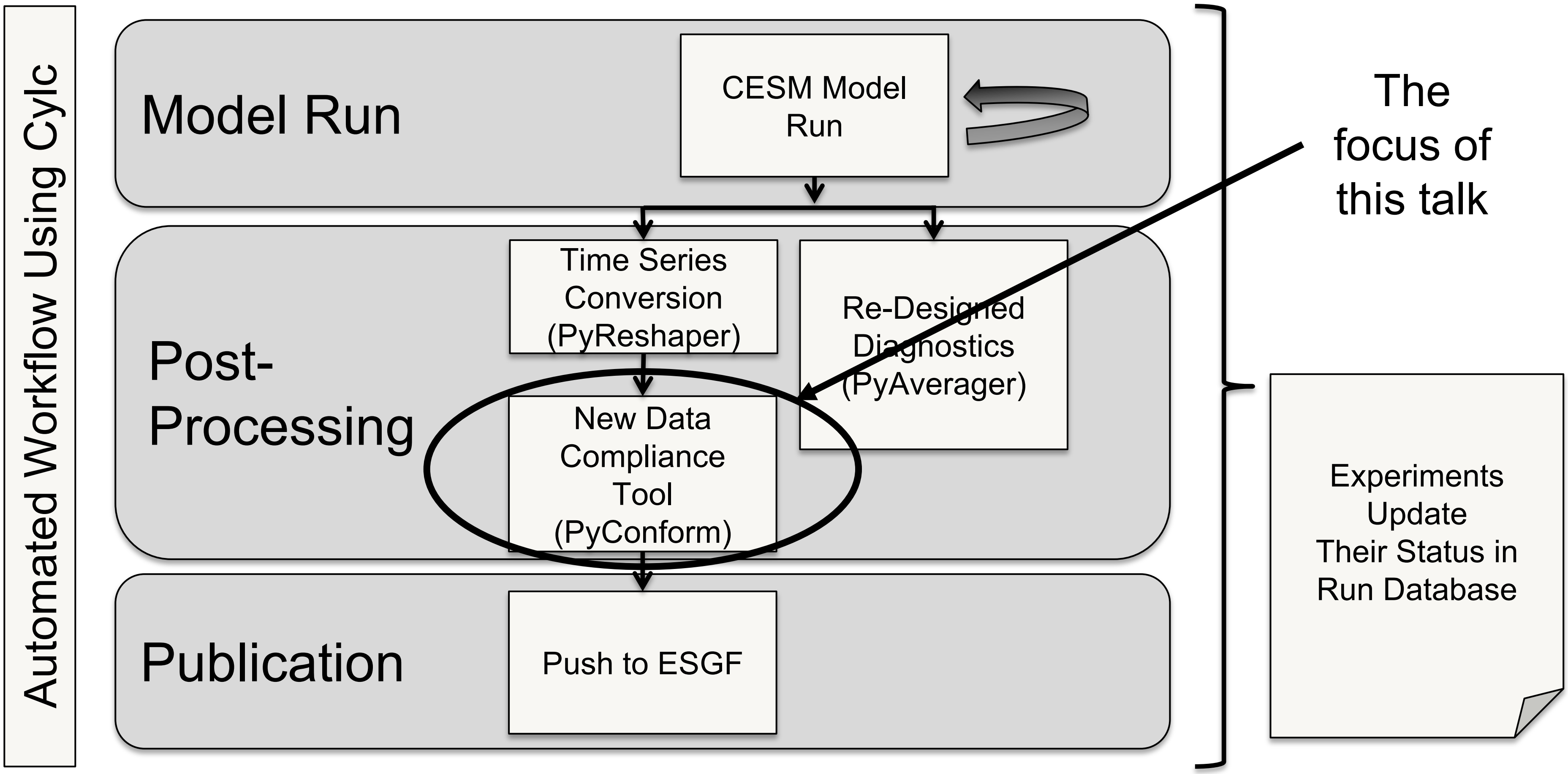
- 25 Experiments
- Timeline: 3 years
- Output size: 800TB
- Published size: 200TB

CMIP6

- 102 Experiments
- Timeline: 1 year
- Output size: 8PB (estimate)
- Published size: 2PB (estimate)

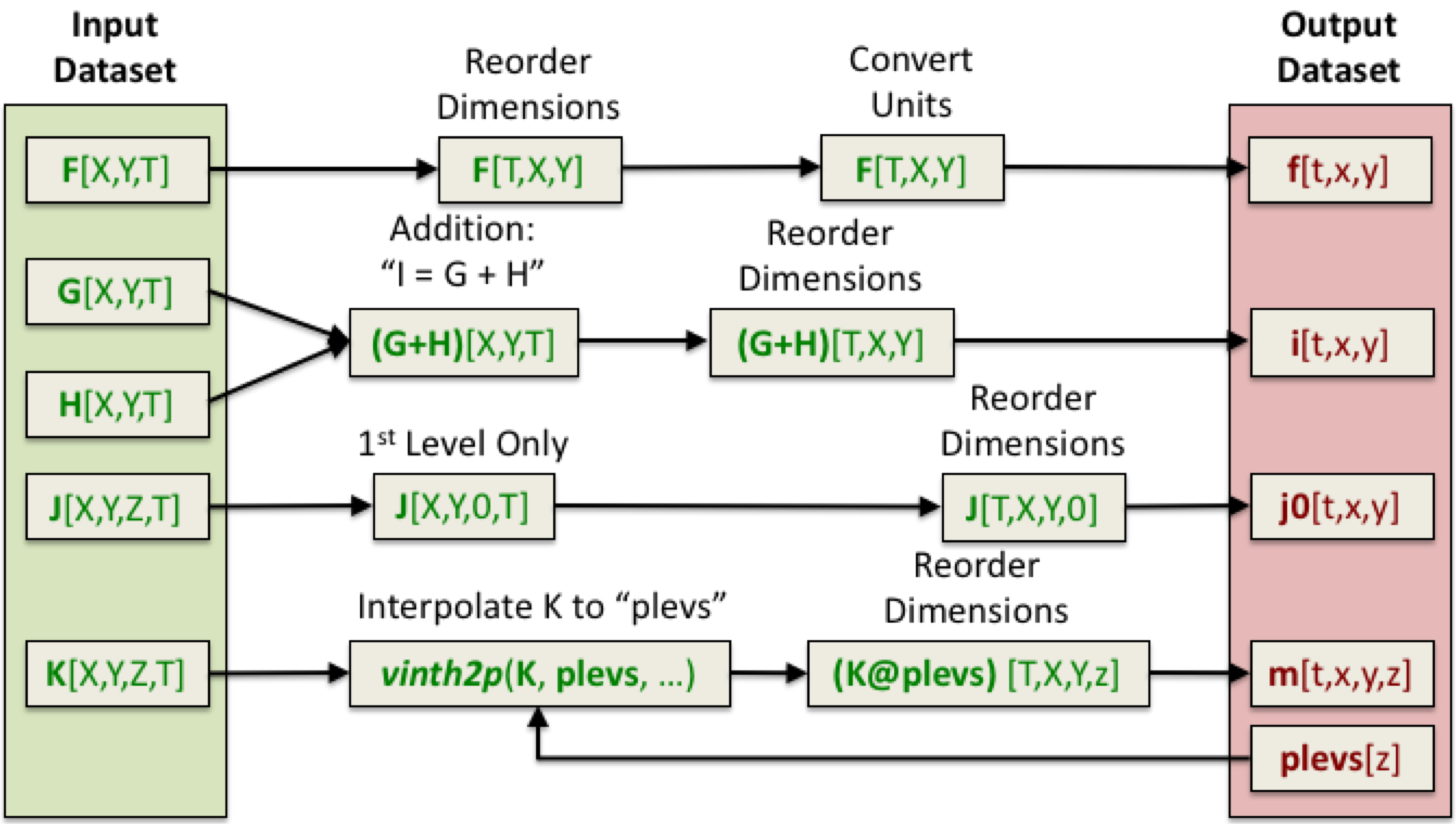
http://www.bbc.com/earth/story/20170510-terrifying-20m-tall-rogue-waves-are-actually-real

New CESM/CMIP6 Workflow



Data Compliance

Taking model output and processing it into experiment compliant data



Previous Version Used for CMIP5

- Used Fortran and CMOR to read in the raw output, do all conversions, and write out complaint files

- Serial, no parallelization

Slow
Rigid and Hard to Expand
Error Prone

PyConform: New Version Used for CMIP6

- Uses Python
    - netCDF4, numpy, dreqPy, cf_units, pyNGL
- Parallelization done with MPI4Py
- A three step process

Faster **(16x to 38x speedup over Fortran method)**
Flexible User Interface

First Step

Users need to create a text file with definitions that describe how to map model variables to requested variables

Examples:

```
cfc11global=f11vmr
ch4=vinth2p(CH4, hyam, hybm, plev, PS, P0)
mc=CMFMC+CMFMCDZM
siage=siage
```

Second Step

Then users run the iconform tool that matches the definitions to its variable information within the CMIP6 Data Request

The Data Request lists variable requirements:

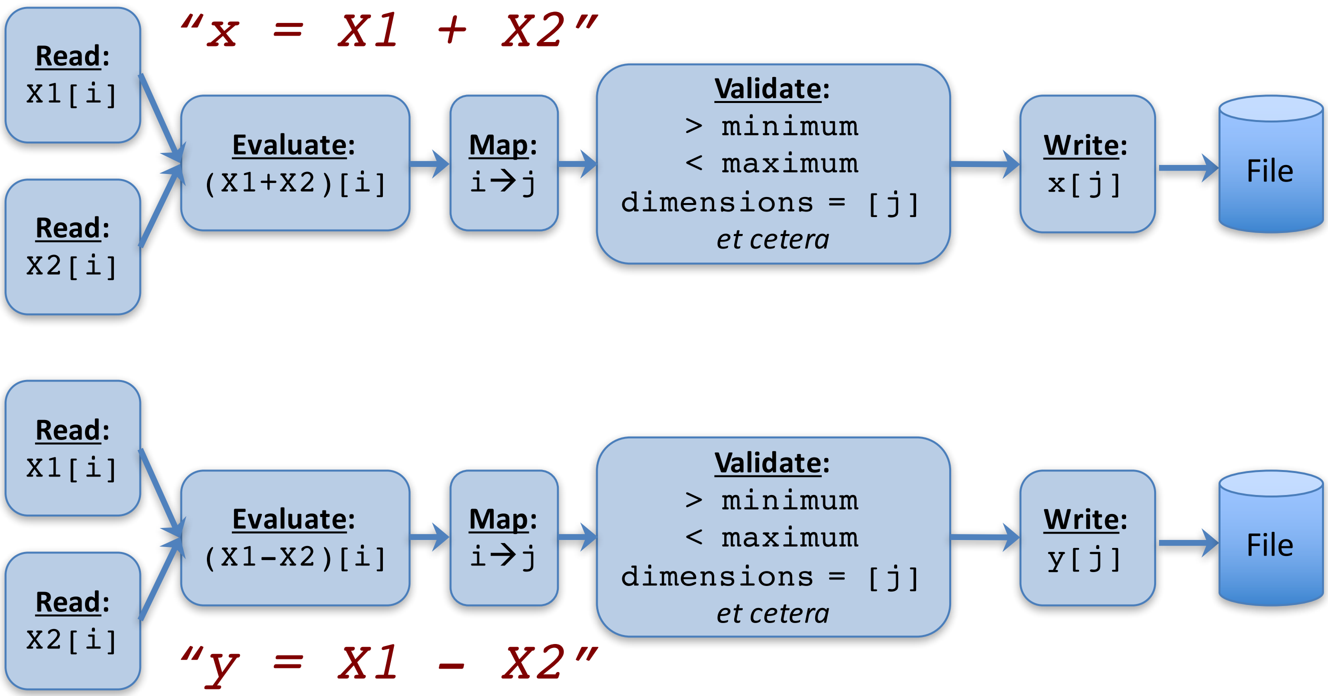
- Units
- Dimensions
- Descriptions
- Positive Attribute on Vertical Dimensions
- And a lot more …

Sample Portion of a PyConform Input File


```
"ua": {
    "attributes": {
        "_FillValue": "1e+20",

        "cell_measures": "area: areacella",

        "cell_methods": "time: mean",

        "comment": "\"Eastward\" indicates a
vector component which is positive when directed
eastward (negative westward). Wind is defined as a
two-dimensional (horizontal) air velocity vector,
with no vertical component. (Vertical motion in
the atmosphere has the standard name
upward_air_velocity.)",

        "description": "\"Eastward\" indicates
a vector component which is positive when directed
eastward (negative westward). Wind is defined as a
two-dimensional (horizontal) air velocity vector,
with no vertical component. (Vertical motion in
the atmosphere has the standard name
upward_air_velocity.)",

        "frequency": "mon",

        "id": "ua",

        "long_name": "Eastward Wind",

        "mipTable": "Amon",

        "out_name": "ua",

        "prov": "Amon ((isd.003))",

        "realm": "atmos",

        "standard_name": "eastward_wind",

        "time": "time",

        "time_label": "time-mean",

        "time_title": "Temporal mean",

        "title": "Eastward Wind",

        "type": "real",

        "units": "m s-1",

        "variable_id": "ua"
    },
    "datatype": "real",

    "definition": "vinth2p(U,hyam,hybm,plev,
                   PS,P0)",
```

Third Step

Then users run the xconform tool that generates requested variables based on the input specifications



Physarray Object

- Is a subclass of the maskedArray in NumPy
- Additional features that were needed above the masked array class:
  - Automatic Unit Conversion
  - Automatic Dimension Handling
  - Automatic Handling of the Positive Attribute

Physarray Object

Extra features we needed to generate the data correctly:
Automatic Unit Handling

"X = X1 + X2"

Units: K = Units: K + Units: K

"X" = Units: K

"X = X1 + X2"

Units: K = Units: K + Units: C

Convert to K before operation is performed

Units: K

"X" = Units: K + Units: K

"X = X1 * X2"

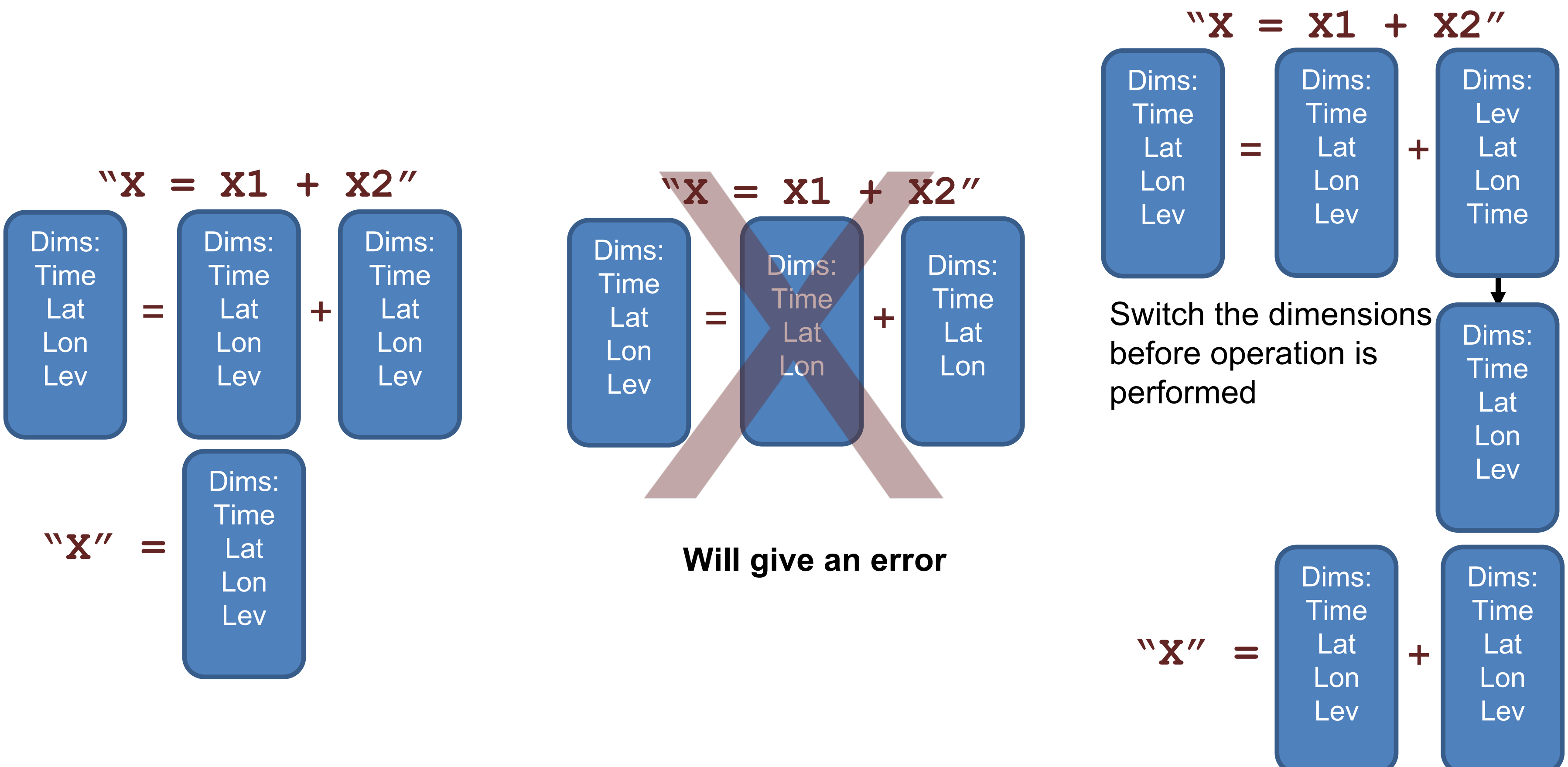
Units: kg m-2 = Units: kg * Units: m-2

"X" = Units: kg m-2

* Must be cf compliant units

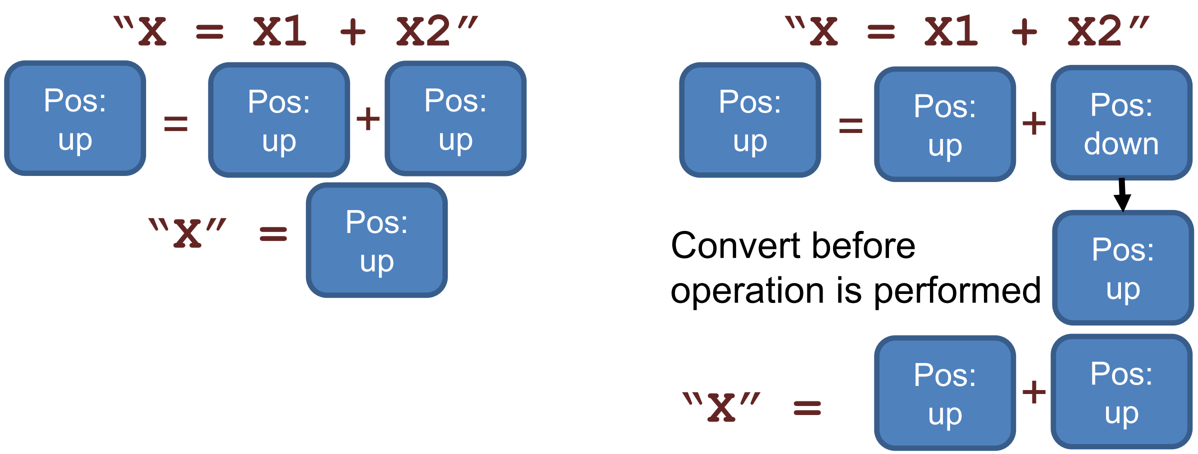
Physarray Object

Extra features we needed to generate the data correctly:
Automatic Dimension Handling



Physarray Object

Extra features we needed to generate the data correctly:
Automatic Handling of the Positive Attribute
(flipping the vertical dimension)



Switching to Xarray/Dask

- We are working on a new version that uses xarray
- While we no longer need the ability to handle dimension reordering, we still need functionality to handle the unit conversion and the flipping of the vertical dimension
- We will also need to evaluate the performance

Moving Forward ….

- We are currently using PyConform in its current form for our CMIP6 output

- We are looking at a redesign of the internal data structures to use new capabilities that didn't exist when we started the project

- Performance and usability are key for this tool and we will move in those directions

Questions

Contact: mickelso .at. ucar.edu
https://github.com/NCAR/PyConform

http://m.sweetclipart.com/halloween-pumpkin-pail-with-candy/