Earth Observation Data Cubes from Satellite Image Collections

The gdalcubes library

Marius Appel
July 5, 2019

GDAL
CUBES

Why?

- My problem: Combine global MODIS vegetation, MODIS land surface temperature and GPM precipitation data to model resistance of vegetation against heat / drought periods

- Problems of my colleagues, students (and others...):
    - How to combine Sentinel 2 and Landsat data to build long, dense time series
    - How can I process Sentinel 2 time series covering multiple tiles?
    - How can I develop my models on large datasets interactively?
    - ...

Satellite imagery is irregular!

- Differences in spatial resolution
- Overlapping spatial footprints
- Irregular time series for adjacent grid tiles
- Different spatial reference systems
- Variety of data formats
- ...

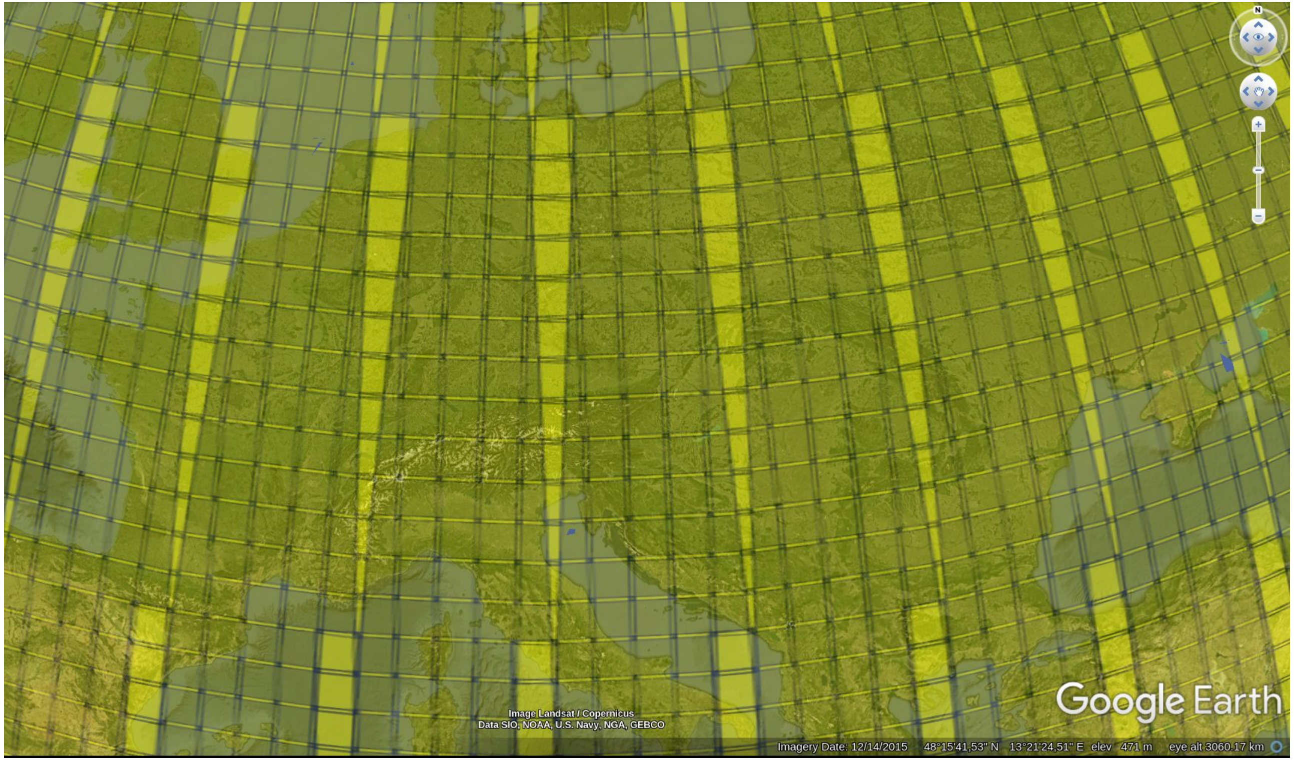
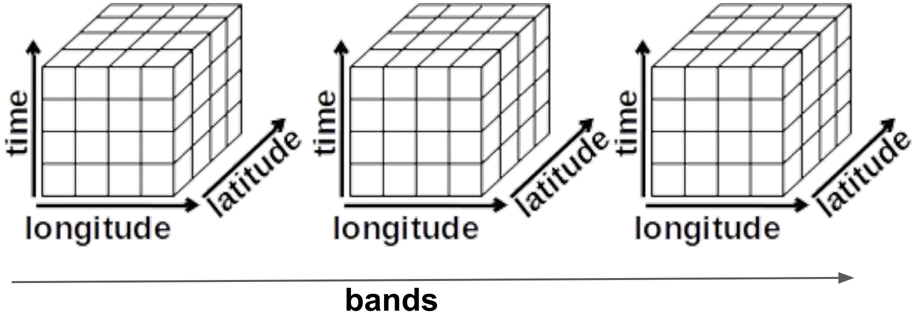


Image Landsat / Copernicus
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Google Earth
Imagery Date: 12/14/2015 48°15'41.53" N 13°21'24.51" E elev 471 m eye alt 3060.17 km

Wanted...

- A *regular dense raster data cube*:
  - 4d array (x, y, time, bands)
  - single spatial reference system (SRS)
  - Spatial axes aligned with SRS axes
  - Cells have constant temporal duration
  - b × t × y × x →real value
  -



- Creating data cubes from satellite imagery ⇒ Loosing information!
  - Reprojection, resampling in space
  - Aggregation in time
  -
- There is no “one and only” data cube, properties may or may not be appropriate for particular applications
- How to create data cubes from satellite imagery?

The gdalcubes library

- Open source, written in C++ (https://github.com/appelmar/gdalcubes)
- Core library builds on
  - GDAL to read and warp (reproject, crop, resample) images
  - NetCDF-4 to export data cubes as files
  - SQLite, CURL, and some header only libraries…
- R package on CRAN (https://cran.r-project.org/package=gdalcubes)
- Future: Python?

The gdalcubes library

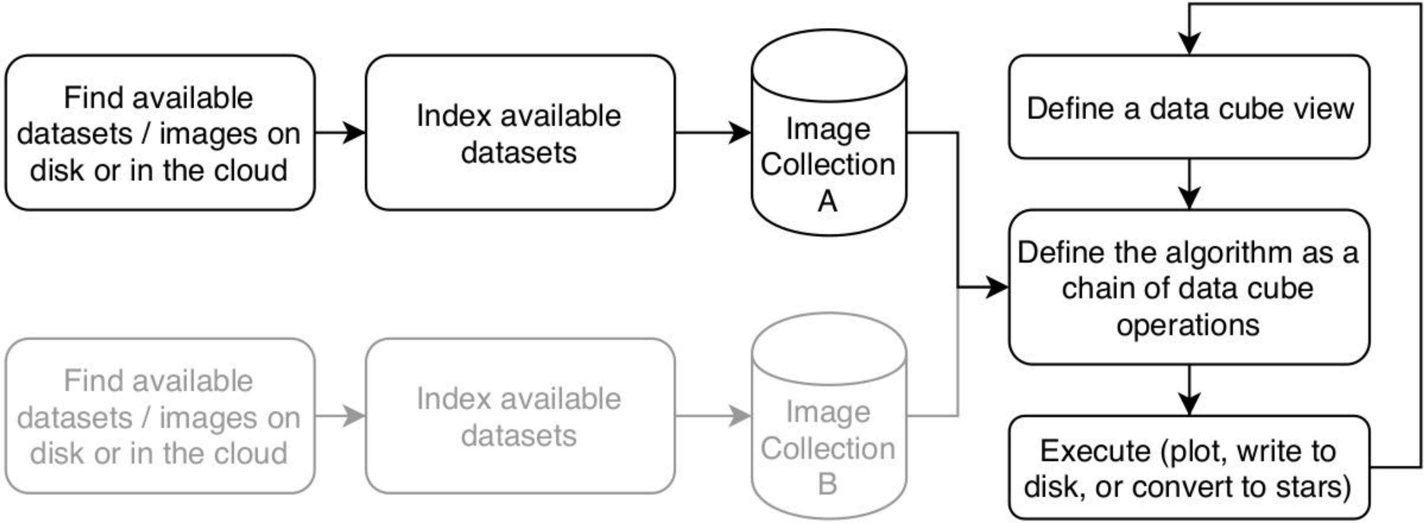
Builds and processes data cubes from image collections, where

- users define the shape of the target cube as a *data cube view* (st extent, resolution, srs)
- users may define a chain of operations to process the resulting cube
- image data is read on the fly, when needed (lazy evaluation)

Target users: data scientists, who

- do remote sensing time series analysis,
- apply methods on large areas,
- need to combine multiple EO data products

Using gdalcubes


Find available datasets / images on disk or in the cloud
Index available datasets
Image Collection A
Find available datasets / images on disk or in the cloud
Index available datasets
Image Collection B
Define a data cube view
Define the algorithm as a chain of data cube operations
Execute (plot, write to disk, or convert to stars)

Hello Sentinel-2...

```r
library(gdalcubes)
library(magrittr) # for %>%

gdalcubes_options(threads=8)

# 1. create an image collection from files on disk
files = list.files("/data/sentinel2_l2a_archive", ".zip", full.names = TRUE)
S2.col = create_image_collection(files, format = "Sentinel2_L2A")

# 2. create a data cube view for a coarse resolution overview
v = cube_view(srs="EPSG:3857", extent=S2.col, dx=300, dt="P5D",
              aggregation="median", resampling = "bilinear")

# 3. create a true color overview image
raster_cube(S2.col, v) %>%
  select_bands(c("B02", "B03", "B04")) %>%
  reduce_time("median(B02)", "median(B03)", "median(B04)") %>%
  plot(rgb=3:1,zlim=c(0,1200))
```

Hello Sentinel-2...

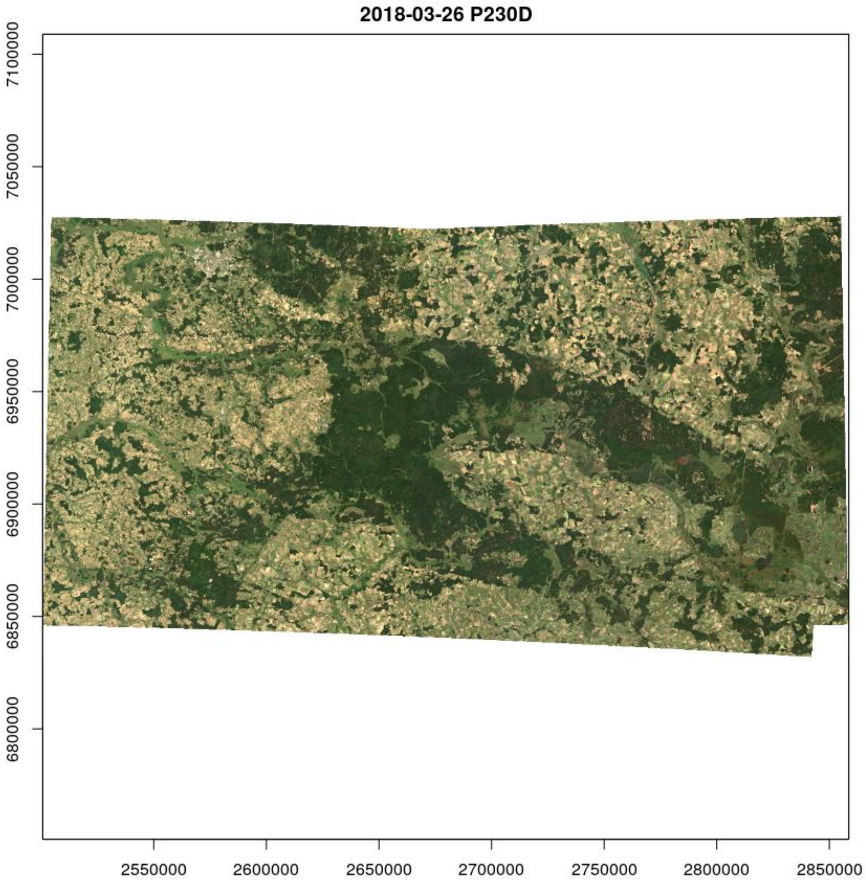
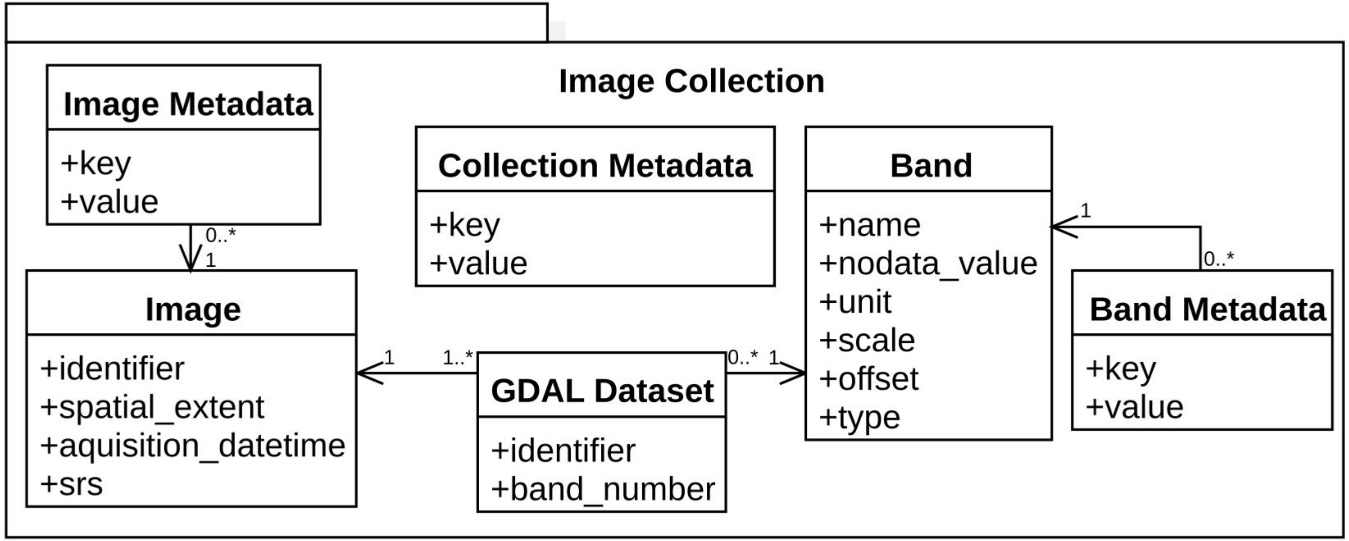


Image collections

- A simple SQLite database, indexing available images, their spatial extent, acquisition date/time, how bands relate to files, and other metadata
- Rules how to derive this information in predefined *collection formats*
- Typical size: a few kilobytes per image



Data cube operations

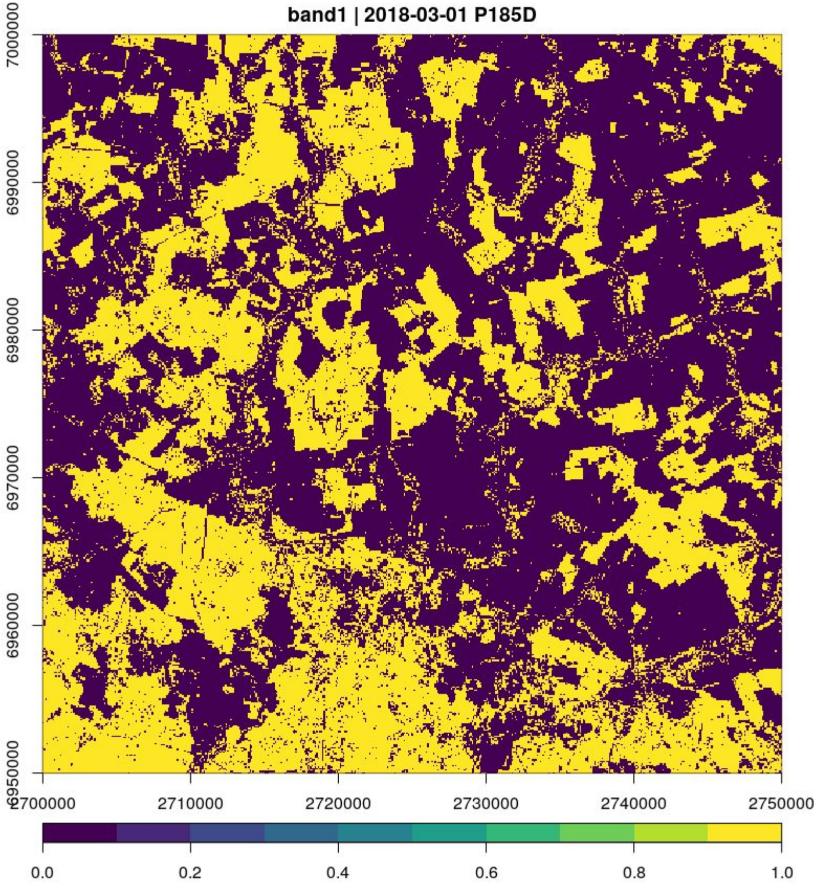
Operator	Description
`raster_cube`	Create a raster data cube from an image collection and a data cube view
`select_bands`	Subset available bands
`reduce_time`	Apply reducer functions over all pixel time series
`reduce_space`	Apply reducer functions over all spatial slices
`apply_pixel`	Apply an arithmetic expression on band values over all pixels
`filter_pixel`	Filter pixels by a logical expression on one or more band values
`join_bands`	Stack the bands of two identically shaped cubes in a single cube
`window_time`	Apply a reducer function or kernel filter over moving windows for all pixel time series
`write_ncdf`	Export a data cube as a netCDF file
`chunk_apply`	Apply a user-defined function over chunks of a data cube

More complex (with mask and user defined reducer)

```
v5 = cube_view(srs="EPSG:3857", extent=list(left=2700000, right=2750000,
               top=7000000,bottom=6950000,t0="2018-03-01",t1="2018-08-31"),
               dx=100, dt="P5D")

raster_cube(S2.col, v5, mask = image_mask("SCL", values=c(1,2,3,8,9))) %>%
  select_bands(c("B04", "B08")) %>%
  apply_pixel("(B08-B04)/(B08+B04)", "NDVI") %>%
  reduce_time(names="NDVI_GT_06", FUN = function(x) {
    return(sum(x["NDVI",] > 0.6, na.rm = TRUE) /
           sum(!is.na(x["NDVI",]), na.rm = TRUE))}) %>%
  apply_pixel("iif(NDVI_GT_06 > 0.7, 1, 0)") %>%
  plot(col=viridis::viridis, key.pos=1)
```

More complex (with mask and user defined reducer)



Interactivity

```
v.300m = cube_view(srs="EPSG:3857",extent=S2.col,dx=300,dt="P5D")
system.time(raster_cube(S2.col, v.300m) %>%
  select_bands(c("B02", "B03", "B04")) %>%
  reduce_time("median(B02)","median(B03)","median(B04)") %>%
  write_ncdf("300m.nc"))
```

```
v.50m = cube_view(srs="EPSG:3857",extent=S2.col,dx=50,dt="P5D")
system.time(raster_cube(S2.col, v.50m) %>%
  select_bands(c("B02", "B03", "B04")) %>%
  reduce_time("median(B02)","median(B03)","median(B04)") %>%
  write_ncdf("50m.nc"))
```

```
v.10m = cube_view(srs="EPSG:3857",extent=S2.col,dx=10,dt="P5D")
system.time(raster_cube(S2.col, v.10m) %>%
  select_bands(c("B02", "B03", "B04")) %>%
  reduce_time("median(B02)","median(B03)","median(B04)") %>%
  write_ncdf("10m.nc"))
```

Interactivity

```
v.300m = cube_view(srs="EPSG:3857",extent=S2.col,dx=300,dt="P5D")
system.time(raster_cube(S2.col, v.300m) %>%
  select_bands(c("B02", "B03", "B04")) %>%
  reduce_time("median(B02)","median(B03)","median(B04)") %>%
  write_ncdf("300m.nc"))
```

≈40 seconds

```
v.50m = cube_view(srs="EPSG:3857",extent=S2.col,dx=50,dt="P5D")
system.time(raster_cube(S2.col, v.50m) %>%
  select_bands(c("B02", "B03", "B04")) %>%
  reduce_time("median(B02)","median(B03)","median(B04)") %>%
  write_ncdf("50m.nc"))
```

≈ 26 minutes

```
v.10m = cube_view(srs="EPSG:3857",extent=S2.col,dx=10,dt="P5D")
system.time(raster_cube(S2.col, v.10m) %>%
  select_bands(c("B02", "B03", "B04")) %>%
  reduce_time("median(B02)","median(B03)","median(B04)") %>%
  write_ncdf("10m.nc"))
```

≈ 2 hours

Multi-sensor data cubes

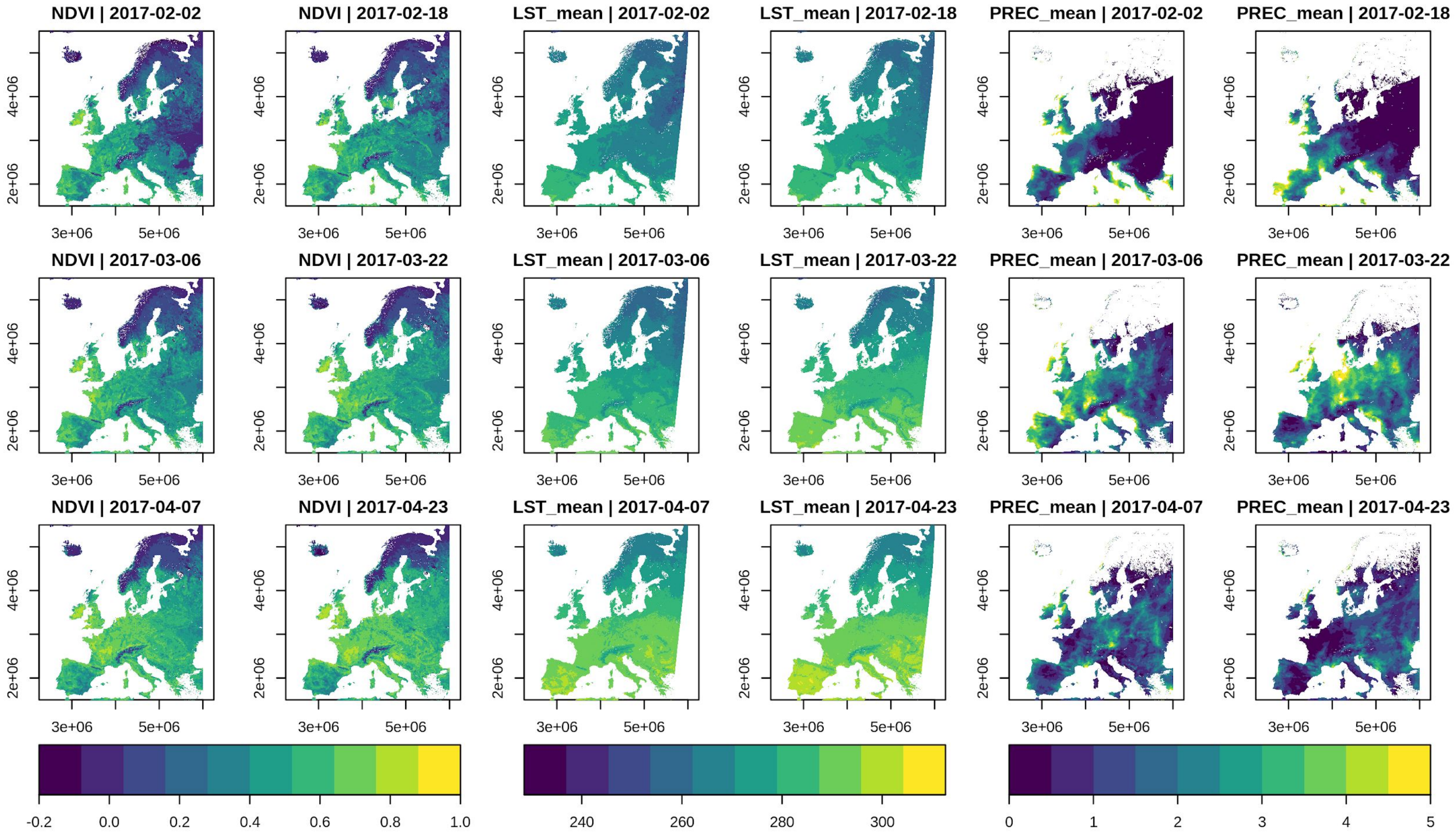
- Approach: use the same *data cube view* for data cubes from different image collections and combine bands with `join_bands()`

```
v.europe = cube_view(srs= "EPSG:3035", extent=list(left=2500000, right = 6000000, top = 5500000,
                     bottom = 1500000, t0 = "2014-01-01", t1 = "2018-12-31"), dx=1000, dt="P1D")

GPM.cube = raster_cube(image_collection("GPM.db"), v.europe) %>%
  select_bands("liquid_accum") %>%
  apply_pixel("liquid_accum / 10", names = "PREC") %>%
  window_time(expr = "mean(PREC)", window = c(30,0))
MOD13A2.cube =
  raster_cube(image_collection("MOD13A2_global_2014_2018.db"), v.europe) %>%
  select_bands("NDVI") %>%
  apply_pixel("NDVI / 1e4", names="NDVI")
MOD11A1.cube = raster_cube(image_collection("MOD11A1_2014_2018.db"), v.europe) %>%
  select_bands("LST_DAY") %>%
  apply_pixel("LST_DAY * 0.02", names="LST") %>%
  window_time(expr = "mean(LST_30D)", window = c(30,0))

join_bands(MOD13A2.cube, GPM.cube) %>%
  join_bands(MOD11A1.cube) %>% filter_pixel("iif(isnan(NDVI), 0, 1)") %>%
  write_ncdf("combined.nc")
```

Multi-sensor data cubes



In the cloud (experiments in progress)

- GDAL can directly read data from object storage on AWS, Google Cloud, and others (https://gdal.org/user/virtual_file_systems.html)
- gdalcubes can run as a simple service (in a container) to process chunks of a data cube (prototypical)
- Image collections point to image objects, and are shared between gdalcubes workers
- Chain of data cube operations serializable as JSON (process graph)
- one instance sends JSON graph to workers and tells them to compute chunks of the result

```
{
  "cube_type": "reduce_time",
  "in_cube": {
    "bands": [
      "B02",
      "B03",
      "B04"
    ],
    "cube_type": "select_bands",
    "in_cube": {
      "chunk_size": [
        16,
        256,
        256
      ],
      "cube_type": "image_collection",
      "file": "S2demo.db",
      "mask": {
        "bits": [],
        "invert": false,
        "mask_type": "value_mask",
        "values": [
          8.0,
          9.0,
          2.0,
          3.0,
          1.0
        ]
      },
      "mask_band": "SCL",
      "view": {
        "aggregation": "mean",
        "resampling": "near",
        "space": {
          "bottom": 6831917.799674884,
          "left": 2500790.1342810756,
          "nx": 700,
          "ny": 383,
          "right": 2858522.442392671,
          "srs": "EPSG:3857",
          "top": 7027880.729027874
        },
        "time": {
          "dt": "P5D",
          "t0": "2018-03-26",
          "t1": "2018-11-10"
        }
      },
      "warp_args": []
    }
  },
  "reducer_bands": [
    [
      "median",
      "B02"
    ],
    [
      "median",
      "B03"
    ],
    [
      "median",
      "B04"
    ]
  ]
}
```

Some limitations

- Data format is limited to 4d
- No vector data cubes
- Limited amount of built-in operations and pre-defined collection formats
- Images must be orthorectified / regularly gridded -> Sentinel-1, Sentinel-5P require preprocessing
- Speedup for low-resolution analysis depends strongly on data formats and availability of image pyramids
- Image collection index needs more scalability tests
- ...

Outlook

No schedule, but many ideas:

- Python interface
- Larger experiments in the cloud, currently AWS, DIASes?
- More data cube operations (including support for user-defined functions)
- Vector data cubes for zonal statistics
- OpenEO backend implementation
- Rewriting Documentation
- Extensions e.g. to interface Orfeo Toolbox
- Applications (with different data e.g. PlanetScope)

Thank you for listening

Contact: marius.appel@uni-muenster.de

Twitter: @appelmar

Further reading:

- Appel, M. and Pebesma, E., 2019. On-Demand Processing of Data Cubes from Satellite Image Collections with the gdalcubes Library. *Data*, *4(3)*, 92. (URL: https://www.mdpi.com/2306-5729/4/3/92)
- https://github.com/appelmar/gdalcubes
- https://appelmar.github.io/gdalcubes
- https://github.com/appelmar/gdalcubes_R
- https://cran.r-project.org/package=gdalcubes