Scalable, Spatiotemporal Tidy Arrays for R (`stars`)

Edzer Pebesma, Etienne Racine, Michael Sumner

ifgi
Institute for Geoinformatics
University of Münster

UseR! 2017, Jul 4-7, 2017, Brussels

stars

- is a follow-up project on "simple features for R"
- is, like `sf`, supported by the R consortium
- is anticipating the question: "When will `raster` support simple features?"
- is there because `setdiff(sp, sf) =` raster data
- will not easily *replace* `raster`, which has 45K lines of code
- has time frame: Sep '17 - Sep '18
- consists, right now, of design ideas (a good time to get involved!)

What is an array?

An array A is a mapping from dimensions $D \subset \{D_1 \times ... \times D_n\}$ to values $V \subset= \{V_1 \times ... \times V_m\}$, with D_i finite and totally ordered:

$$A : D \rightarrow V$$

Examples:

- climate: {lat, long, elev, time} $\rightarrow$ {temp, humidity}
- image: {row, col, color} $\rightarrow$ {energy}
- sound: {time, frequency_cls} $\rightarrow$ {level}
- Soc.Ec.: {state, time, age_cat} $\rightarrow$ {income, empl}
- Health: {region, time, age_cat, health_status} $\rightarrow$ {count}

- space and time are often among the dimensions
- space can be 2D array (raster), or a 1D sequence (points, roads, states)
- time can be 2D (time of forecast + time to forecast; week + weekday etc)

tables (`data.frame` etc) *can* hold one-dimensional arrays

What is an array?

An array A is a mapping from dimensions $D \subset \{D_1 \times ... \times D_n\}$ to values $V \subset= \{V_1 \times ... \times V_m\}$, with D_i finite and totally ordered:

$$A : D \rightarrow V$$

Examples:

- climate: {lat, long, elev, time} $\rightarrow$ {temp, humidity}
- image: {row, col, color} $\rightarrow$ {energy}
- sound: {time, frequency_cls} $\rightarrow$ {level}
- Soc.Ec.: {state, time, age_cat} $\rightarrow$ {income, empl}
- Health: {region, time, age_cat, health_status} $\rightarrow$ {count}

- space and time are often among the dimensions
- space can be 2D array (raster), or a 1D sequence (points, roads, states)
- time can be 2D (time of forecast + time to forecast; week + weekday etc)

tables (`data.frame` etc) *can* hold one-dimensional arrays

Why is there a need for a new project/package?

`base::array`
- in-memory arrays of single ("scalar") value type
- dimension reference would need to be in `dimnames` (`character`)

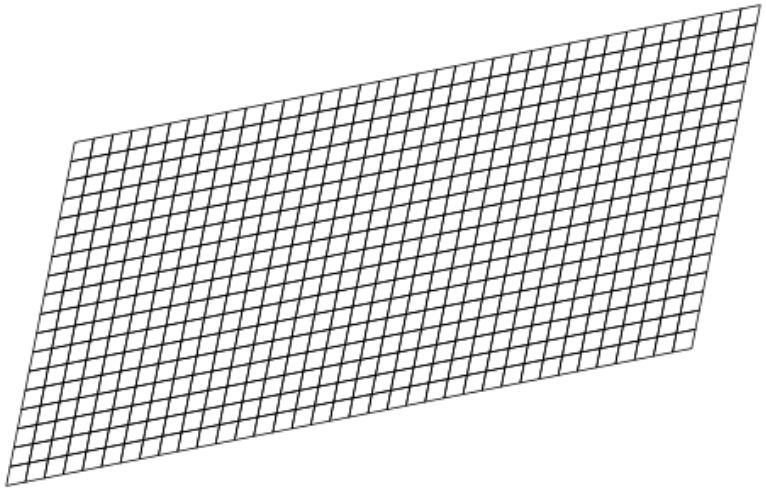
`dplyr::tbl_cube`
- in-memory, referenced dimensions
- mixed value type (list of `base::array`), dimensions are vectors
- not used much; still experimental?

`raster::raster`
- scales to disk-size data sets; uses proxies
- single type
- limited to 3D: raster x/y + 1D (time, or layer, not both)
- largely in C++, but I/O through other pkgs, non-linked
- used a lot

`spacetime::STFDF` in memory; see JStatSoft paper

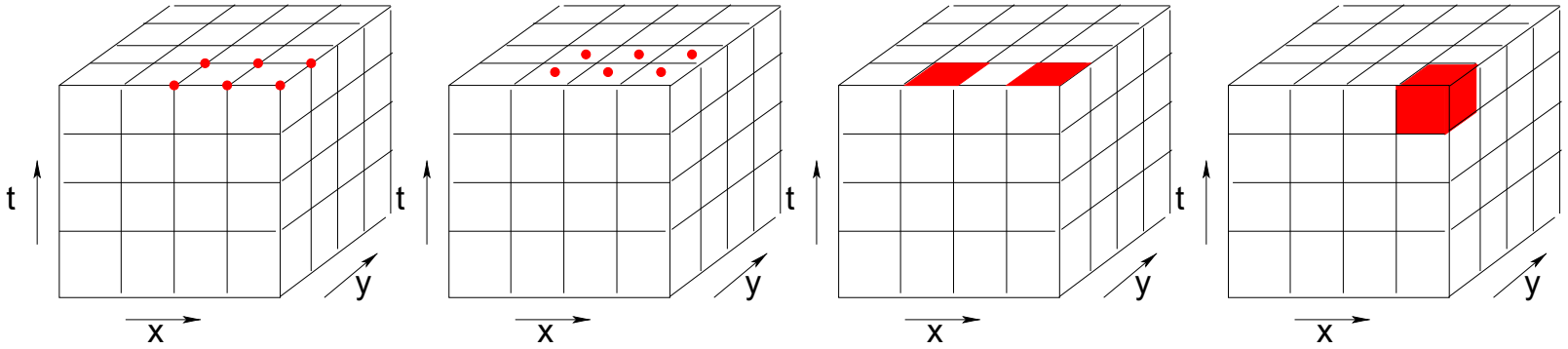
`stars`: 1. basic idea

1. `stars` objects extend `data.frame`, or are a list of `base::arrays`
2. only Values are retained, dimensions are implicit
3. dimensions information are kept in an attribute, the `Dimensions` list
4. each dimension has methods to convert from index (1,2,...,n) to dimension value, and back
5. examples:
   - type, start, end, n
   - list with (ordered) values, or simple features
   - (for sets of dimensions): parameters of transformation, e.g. affine:



stars: 2. Spatiotemporal

1. *use* the notion of cell size, and time intervals (spatial support)



2. strong support for spatial reference systems
3. regridding, warping, etc.
4. support for measurement units

`stars`: 3. Tidy

1. provide tidy verbs (methods)
2. extend with array things
   - slice/dice; filter (select) based on values
   - apply functions over array dimension; dimension reduction
   - downscale/upscale/aggregate
   - dimension to value, value to dimension conversions (dimension/attribute flattening)
   - filter operations

`stars`: 4. Scalable (1/2)

1. besides having complete arrays in memory, `stars` will also accomodate for proxy objects, having parts of the complete array in memory, other parts on local disk, **or on remote servers**
2. the proxy will have a reduced array, using strides (reading e.g. every 10-th row, 10-th column), so that visualisation is responsive, makes sense, and all dims covered
3. `dplyr`-style computation on small sets, fetching complete set triggers complete computation, possibly remotely

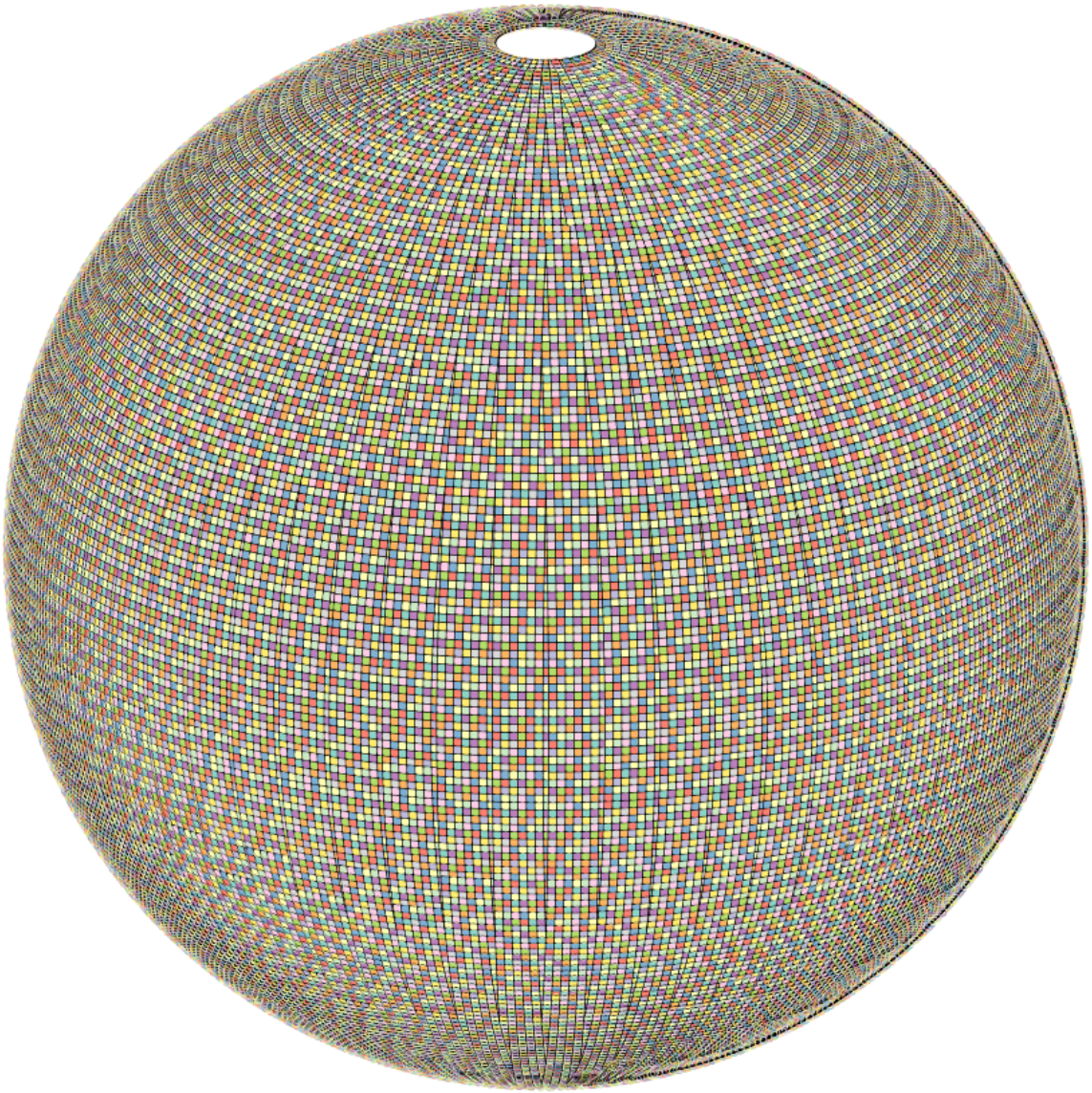
Q1: when proxying imagery on local disk or remote server, use the same interface?

`stars`: 4. Scalable (1/2)

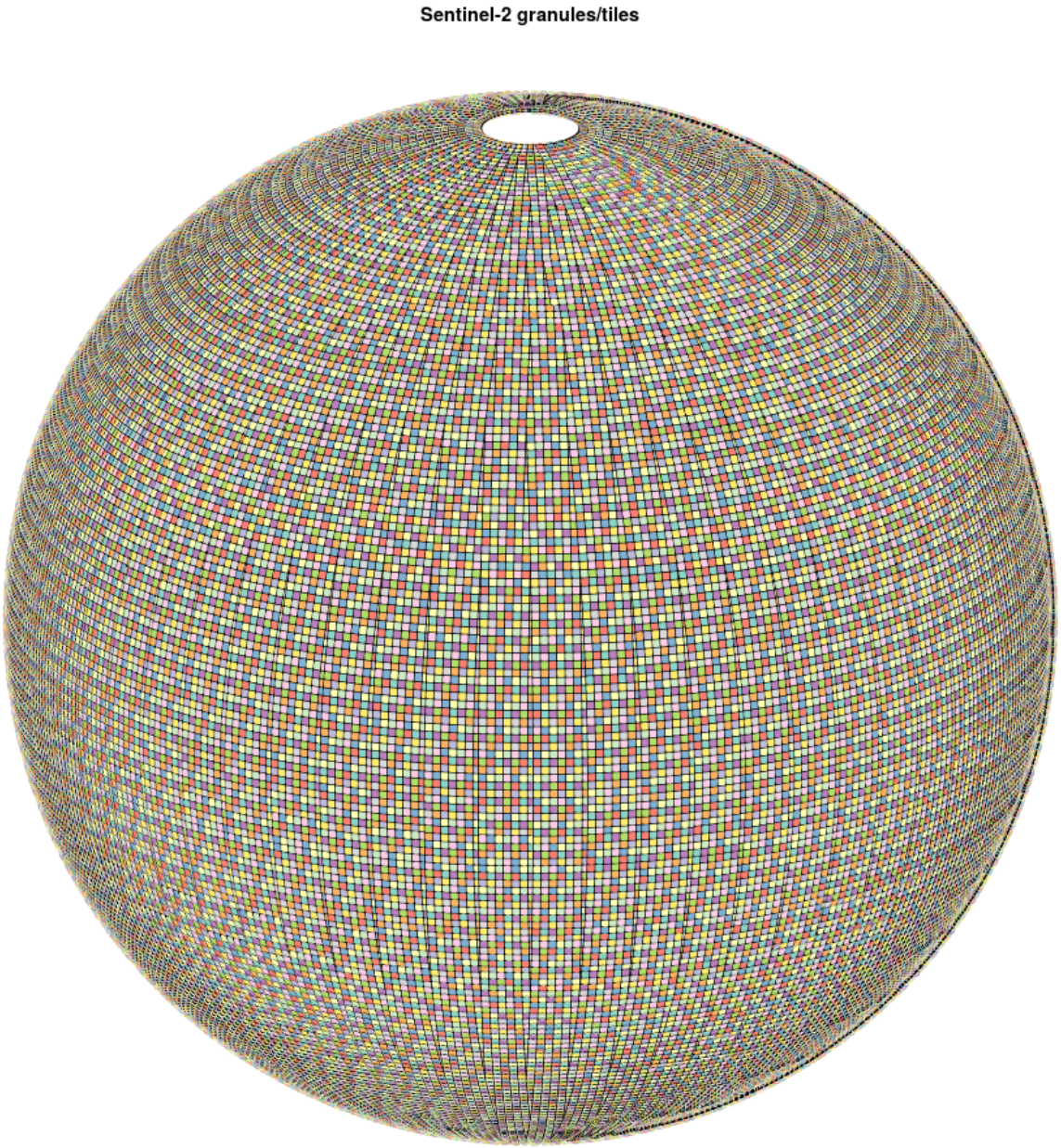
1. besides having complete arrays in memory, `stars` will also accomodate for proxy objects, having parts of the complete array in memory, other parts on local disk, **or on remote servers**
2. the proxy will have a reduced array, using strides (reading e.g. every 10-th row, 10-th column), so that visualisation is responsive, makes sense, and all dims covered
3. `dplyr`-style computation on small sets, fetching complete set triggers complete computation, possibly remotely

Q1: when proxying imagery on local disk or remote server, use the same interface?

Sentinel-2 granules/tiles

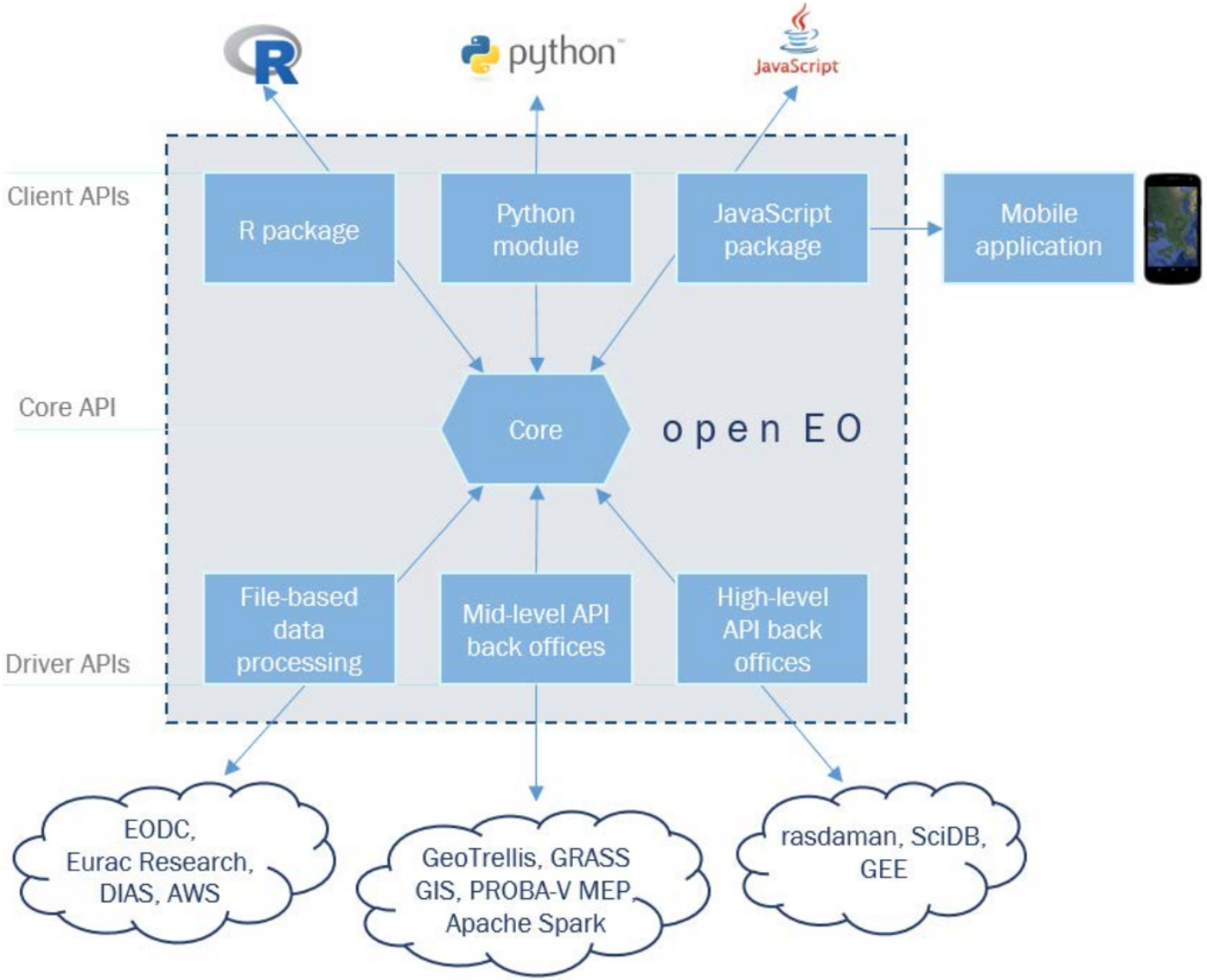


Q2: represent scene collections across different UTM zones?



Q2: represent scene collections across different UTM zones?

Larger context: openEO



Conclusions

1. downloading time series of EO data or model results is not a reasonable proposition (LS8, Sentinel 1-3, CMIP5/6)
2. `stars` will try to help smooth the transition from analysing arrays in-memory ⇒ on-disk ⇒ on-remote-server
3. it will use dplyr in-memory vs. in-database as a template, and follow the tidy manifesto
4. it will build upon `GDAL`, possibly `netcdf`, and reuse rpkg `sf` and `units`
5. development will start in September
6. feedback & involvement very welcome: `https://github.com/edzer/stars`