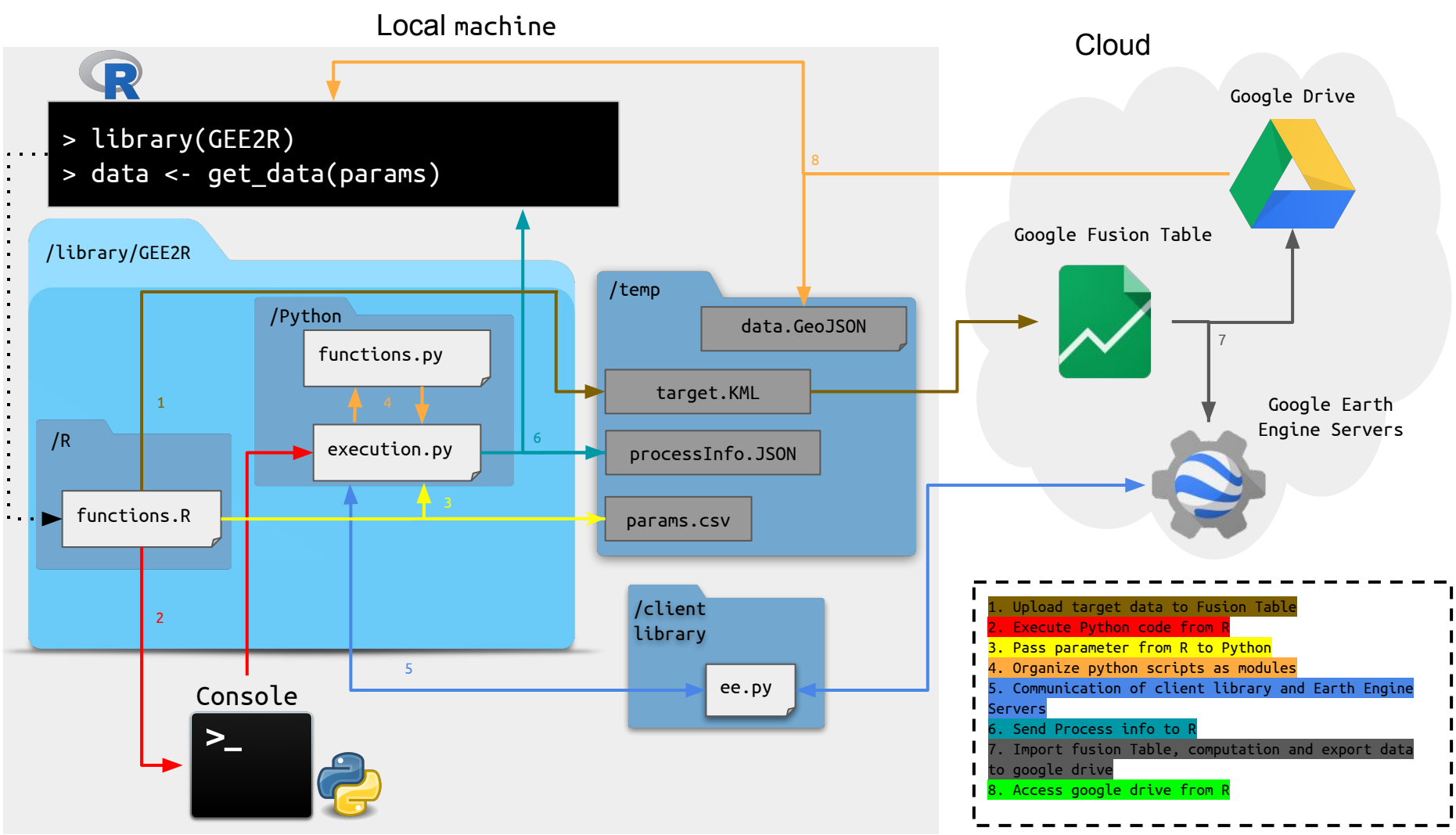
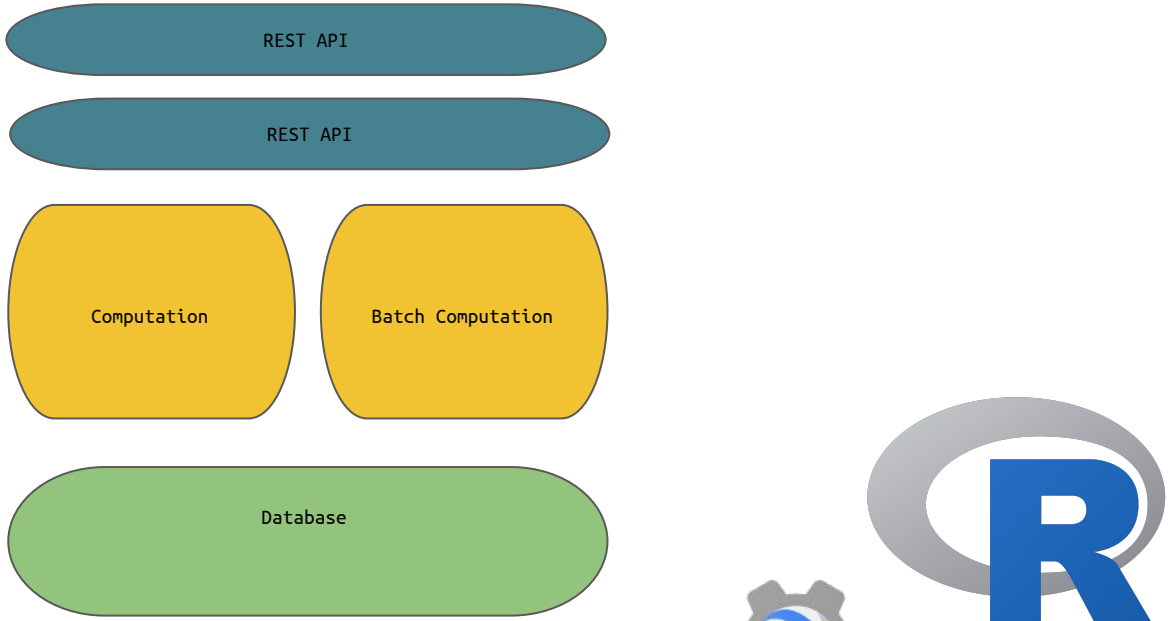

Local machine
Cloud
> library(GEE2R)
> data <- get_data(params)
/library/GEE2R
/Python
functions.py
execution.py
/R
functions.R
/temp
data.GeoJSON
target.KML
processInfo.JSON
params.csv
/client library
ee.py
Console
>_
Google Drive
Google Fusion Table
Google Earth Engine Servers
1
2
3
4
5
6
7
8
1. Upload target data to Fusion Table
2. Execute Python code from R
3. Pass parameter from R to Python
4. Organize python scripts as modules
5. Communication of client library and Earth Engine Servers
6. Send Process info to R
7. Import fusion Table, computation and export data to google drive
8. Access google drive from R


REST API
REST API
Computation
Batch Computation
Database
R
R

raw raster

final vector

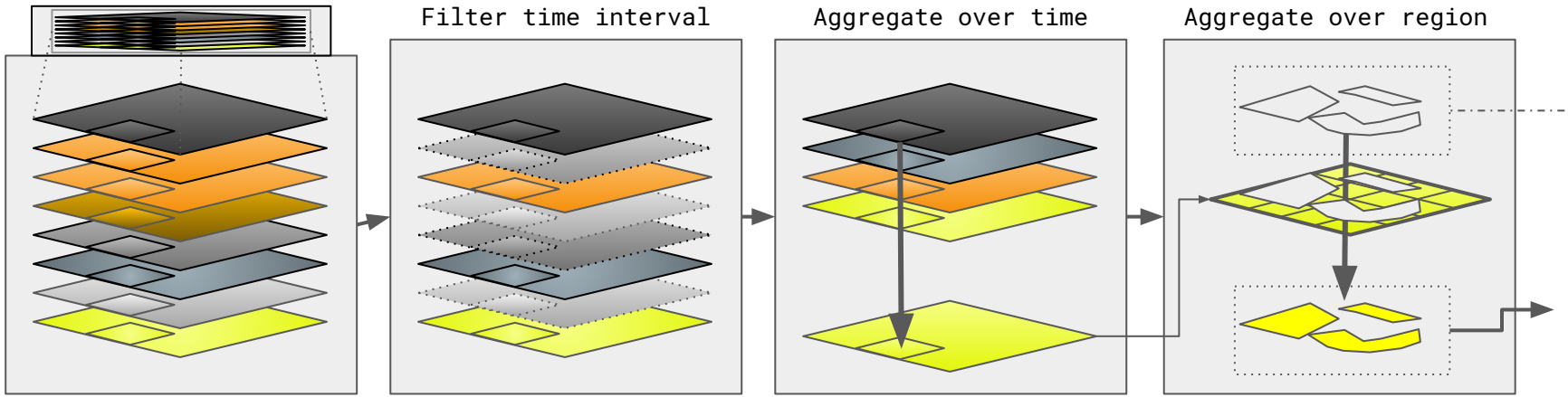
```
get_data(dataProduct, timeInterval, temporalReducer, spatialReducer, target)
```

Select data product

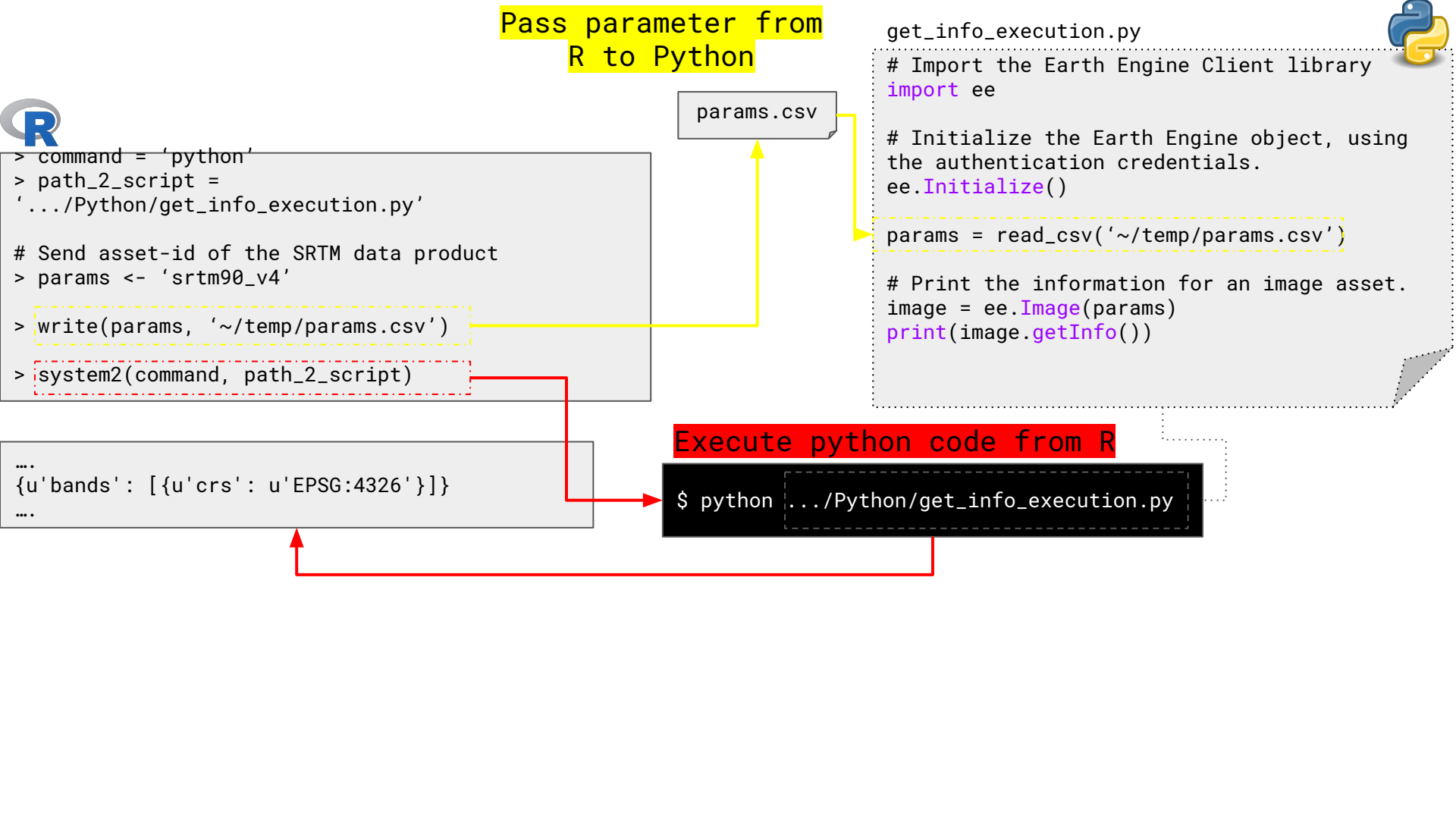
Filter time interval

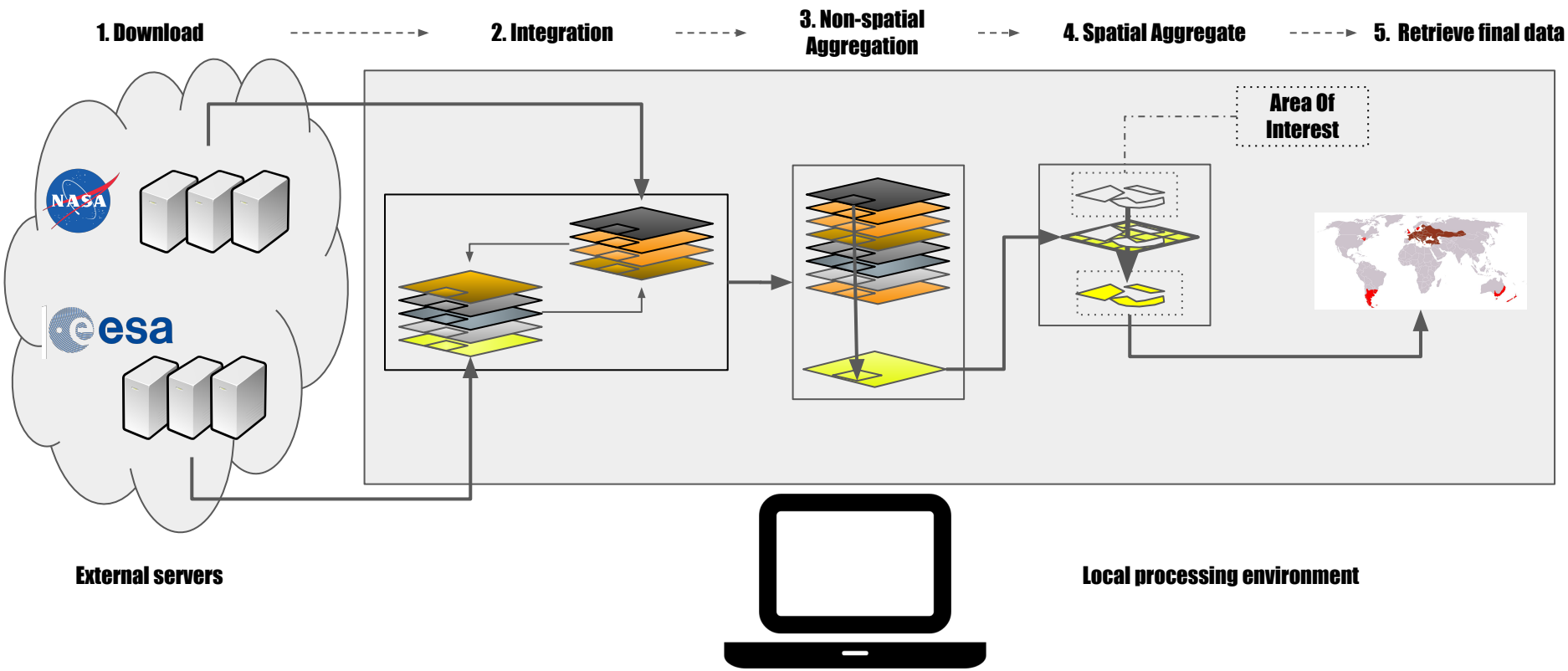
Aggregate over time

Aggregate over region



Output to export


Pass parameter from R to Python
params.csv
get_info_execution.py
# Import the Earth Engine Client library
import ee
# Initialize the Earth Engine object, using the authentication credentials.
ee.Initialize()
params = read_csv('~/temp/params.csv')
# Print the information for an image asset.
image = ee.Image(params)
print(image.getInfo())
> command = 'python'
> path_2_script = '.../Python/get_info_execution.py'
Send asset-id of the SRTM data product
> params <- 'srtm90_v4'
> write(params, '~/temp/params.csv')
> system2(command, path_2_script)
Execute python code from R
$ python .../Python/get_info_execution.py
....
{u'bands': [{u'crs': u'EPSG:4326'}]}
....


1. Download
2. Integration
3. Non-spatial Aggregation
4. Spatial Aggregate
5. Retrieve final data
Area Of Interest
NASA
esa
External servers
Local processing environment