A web based approach for deriving Hydrological Response Units (HRUs) using Open Source Software

Christian Schwartze
Friedrich Schiller University Jena, Germany

Background

- **HRU derivation so far:**
  - Use of mainly commercial SW (Arc*)
  - Applying GIS routines manually and guideline-orientated
- **Goals:**
  - Move to Open Source
  - Wizard-driven tool
  - Almost automated

WPS – Web Processing Service

- **OGC standard (1.0)**
- **Clients access GIS functionality across a network**
  - Spatial operations = **Processes**
  - Retrieving and execution through XML based protocol
  - WCS- and WMS-like requests
    - HTTP GET (key-value pairs)
    - HTTP POST (XML)
  - Response type: *Data* versus *Reference*

WPS sample requests

- ***GetCapabilities* request**

  http://127.0.0.1/cgi-bin/wps.py?service=wps&request=getCapabilities → show

- ***DescribeProcess* request**

  http://127.0.0.1/cgi-bin/wps.py?service=wps&version=1.0.0&request=describeProcess&identifier=waterflow

  → show

- ***Execute* request**

  http://localhost/cgi-bin/wps.py?service=wps&request=execute&version=1.0.0&identifier=outlets&datainputs=[drain=http://localhost/in/drain.tif;gaugesCoord=http://localhost/in/gauges_file]&responsedocument=[watersheds=@asreference=true]

  → show

What we use…

- **PyWPS 3.0.0**
  - Implements OGC WPS 1.0.0 in Python language
  - Native GRASS support, but not limited to
- **GRASS GIS 6.2.2**
  - Spatial processing
- **QuantumGIS 0.11 as client**
  - Embeds the HRU tool and serves as client for WPS on server
  - Visualization and user interaction

PyWPS 3.0.0 - Defining a process

```
from pywps.Process.Process import WPSProcess
class Process(WPSProcess):
    def __init__(self):
        # Process initialization:
        WPSProcess.__init__(self,
            identifier = "simpleProcess",
            title="A very simple process",
            version = "0.2",
            abstract="Adds a value to raster file, so simple...")
```

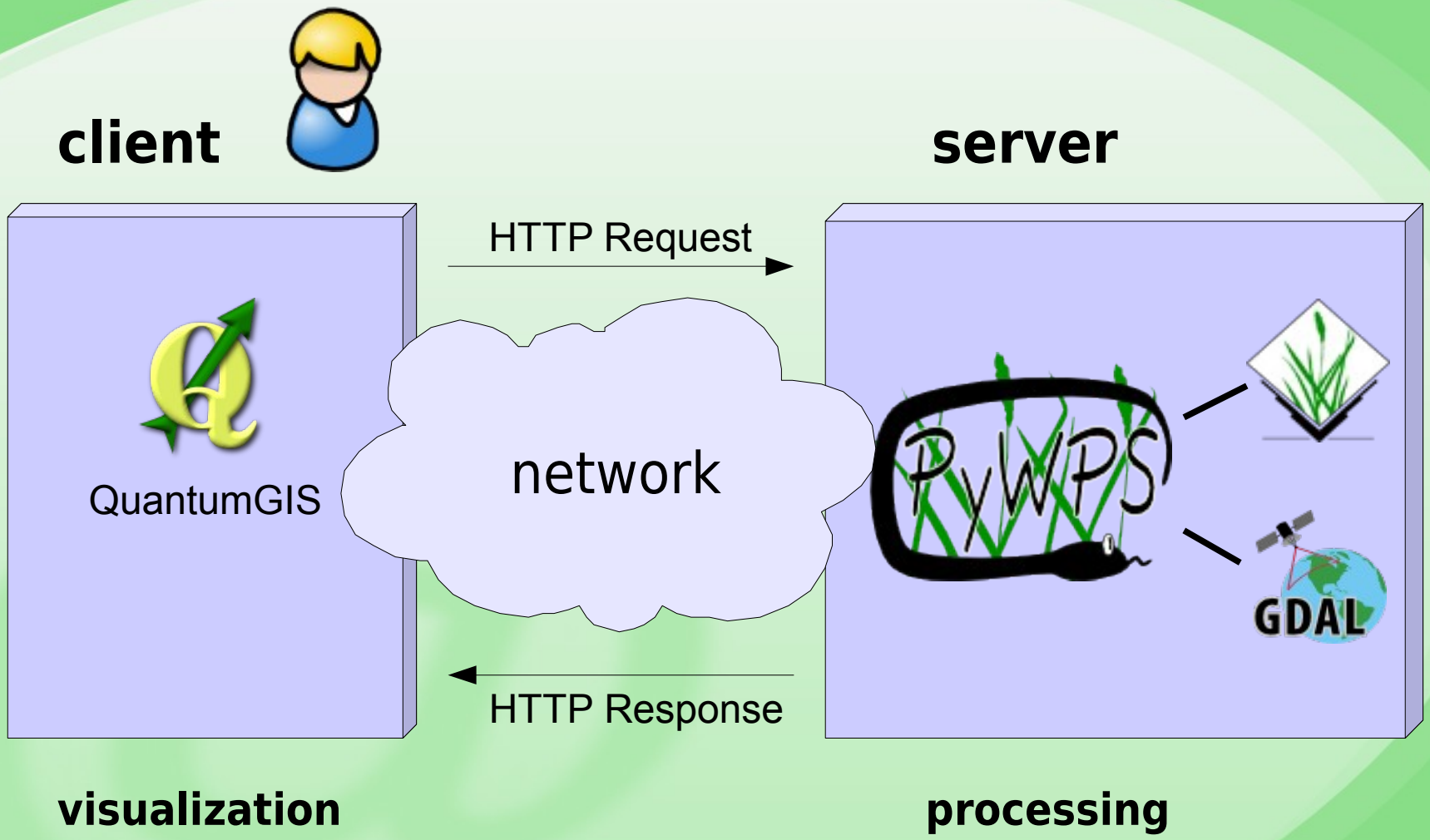
Process initialization

Process configuration

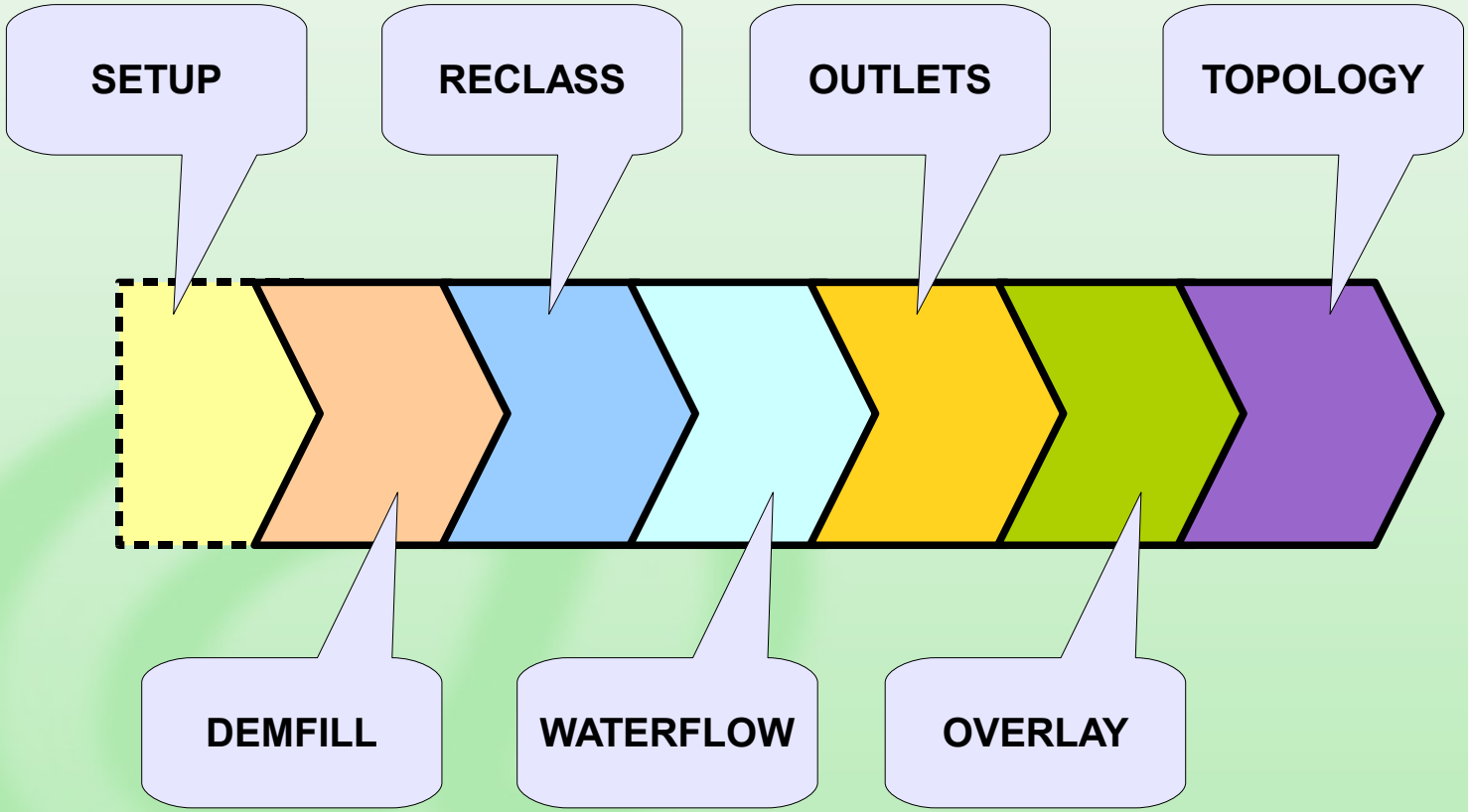
```
        self.someRaster = self.addComplexInput(identifier = "rasterIn",
                                title = "Raster input",
                                abstract = "Some raster we want to change",
                                maxmegabites = "100",
                                formats=[{"mimeType":"image/tiff"}])

        self.valueToAdd = self.addLiteralInput(identifier = "value",
                                title = "Just some value to add",
                                type = types.IntType)

        self.someNewRaster = self.addComplexOutput(identifier="rasterOut",
                                title="Output raster",
                                formats=[{"mimeType":"image/tiff"}])
    def execute(self):
        self.cmd("r.in.gdal input=%s output=test" % self.someRaster.value)
        self.cmd("r.mapcalc newRaster=test+%d" % self.valueToAdd.value)
        self.cmd("r.out.gdal input=newRaster output=newRaster.tif type=Byte")
        self.someNewRaster.setValue("newRaster.tif")
```

Process programming

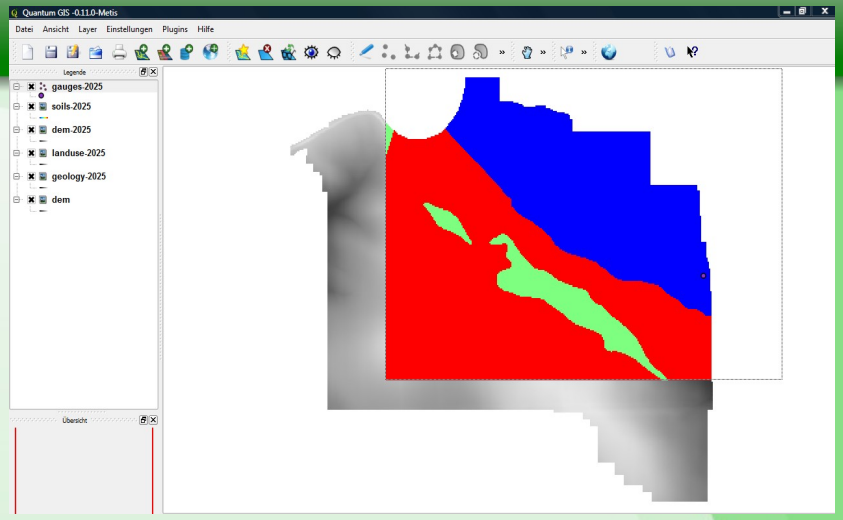
Architecture overview



The HRU WPS process chain

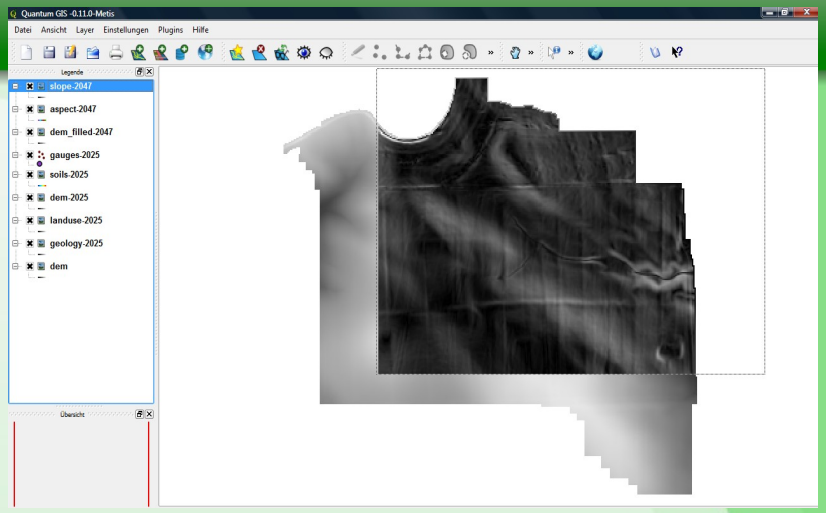


Substep 1 - SETUP.PY



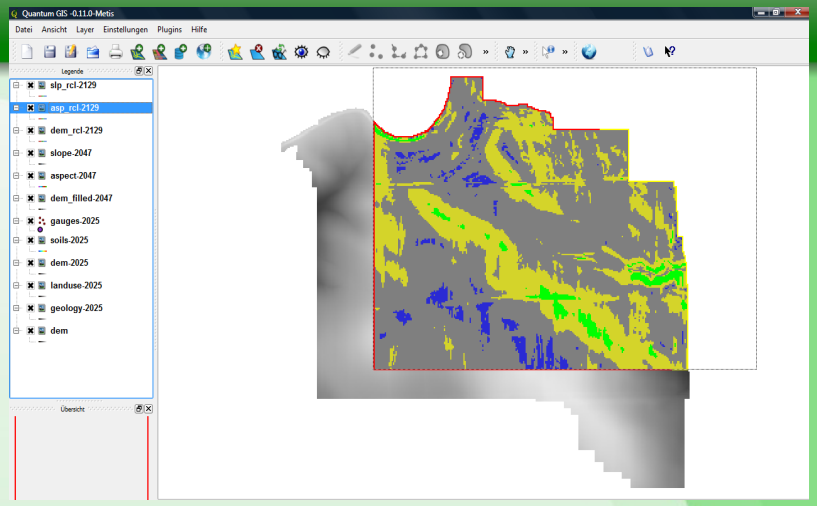
- **Inputs:**
  - DEM and several data layers
  - Bounding box of preferred area
- **Outputs:**
  - Subimages of all input layers (if necessary)
- **Methods:**
  - Geospatial Data Abstraction Library / Simple Feature Library (GDAL / OGR)

Substep 2 - DEMFILL.PY



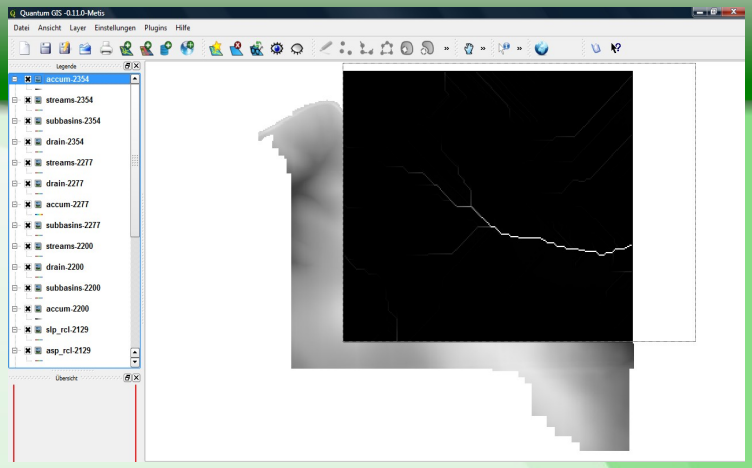
- **Inputs:**
  - DEM
- **Outputs:**
  - Sinkless DEM, aspect and slope
- **Methods:**
  - D8 flow algorithm
  - aspect represents the number degrees of east - 90 (N), 180 (W), 270 (S), 360 (E)
  - slope stated in degrees of inclination from the horizontal
  - i.a. *r.fill.dir*, *r.slope.aspect*

Substep 3 - RECLASSIFY.PY



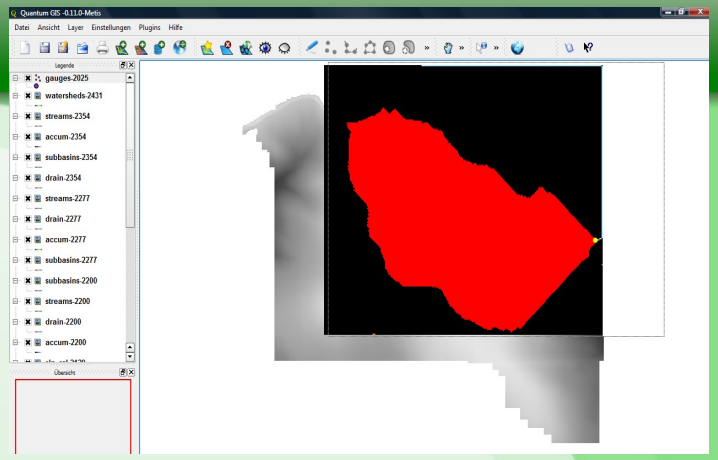
- **Inputs:**
  - DEM, slope and aspect
- **Outputs:**
  - Reclassed maps for DEM, aspect and slope
- **Methods:**
  - Creating reclassification rules based on user-defined ranges
  - Simple *r.reclass* / *r.recode*

Substep 4 - WATERFLOW.PY



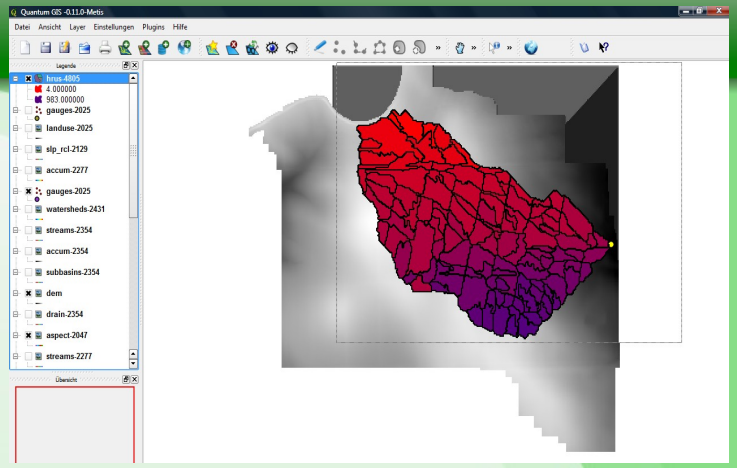
- **Inputs:**
  - Sinkless DEM
  - Minimum size of watershed basins
- **Outputs:**
  - Raster maps for accumulation, drainage direction, stream network and subbasins
- **Methods:**
  - Based on D8
  - Watershed basin analysis program *r.watershed*
  - Recursive upslope algorithm

Substep 5 - OUTLETS.PY



- **Inputs:**
  - Drainage direction map
  - Coordinates of (individual) outlet points
- **Outputs:**
  - Watershed basin map relating to outlets
- **Methods:**
  - Uses slope and aspect
  - i.a. *r.water.outlet*, *r.cross*

Substep 6 - OVERLAY.PY



- **Inputs:**
  - previously preset / calculated raster maps:
    - Watershed basins, subbasins, DEM, slope, aspect, soils, landuse, geology
  - Minimum area threshold
- **Outputs:**
  - HRU map
- **Methods:**
  - Layer intersection via cross product (*r.cross*)
  - Smallest neighbour merging approach (*r.mapcalc*)

Substep 7 - TOPOLOGY.PY

```
[...]
119 140;1.000
121 113;0.441 140;0.559
126 34;0.446 103;0.089 144;0.339 168;0.125
[...]
```

- **Inputs:**
  - HRU map
  - Drainage direction, accumulation, streams
  - Routing method { N:1 ; N:M }
- **Outputs:**
  - CSV file containing flow relations
- **Methods:**
  - Includes HRU → HRU and HRU → reach
  - Provides flow rates (N:M)
  - i.a. *r.mapcalc*, *r.statistics*, *r.cross*

Thank you!