VERWENDUNG VON POSTGRESQL UND GRASS-GIS IN EINER „VIRTUAL APPLIANCE“ FÜR DATENBANKBASIERTE RASTERHALTUNG IN ILMS

PGDay.eu 2010
06.12.2010 – 08.12.2010
Stuttgart

Christian Schwartze

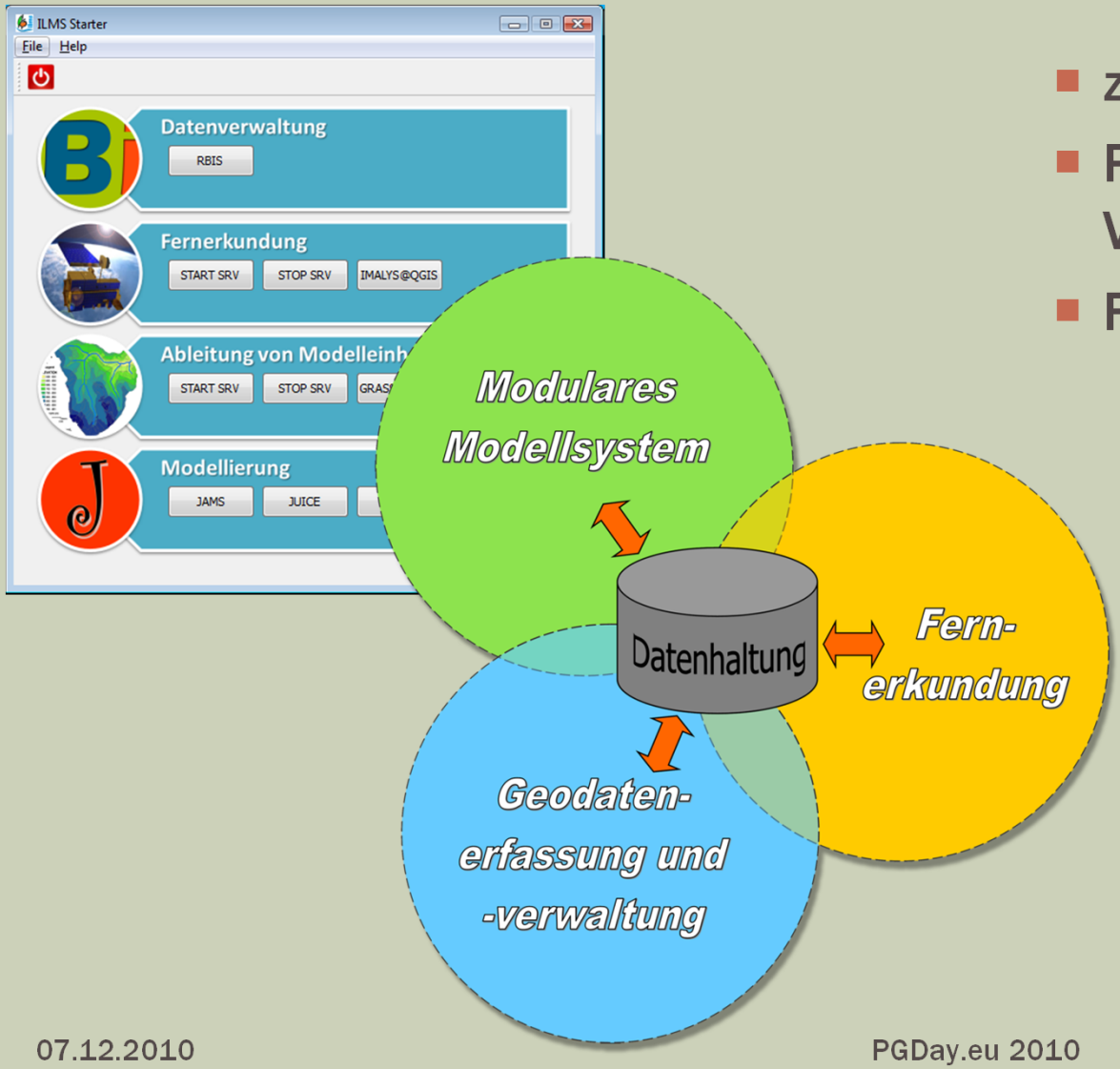
Lehrstuhl für Geoinformatik, Geohydrologie und Modellierung

Universität Jena

PROJEKT ILMS
Integriertes Landschafts-Managementsystem

- Problem
  - Nachhaltiges Landschaftsmanagement
- ILMS-Lösung: Integriertes Softwaresystem zur
  - Erhebung und Verwaltung von Geodaten
  - qualitativen und quantitativen Landschaftsbewertung
  - prognostischen Modellierung von Management-Szenarien
  - Entscheidungsunterstützung für eine nachhaltige Landschaftsbewirtschaftung
- Nutzer
  - Ingenieure, Behörden, Planungsbüros
- Anwendungsbereiche
  - Wasserwirtschaft, Kommunal- und Regionalplanung, Landwirtschaft, Forstwirtschaft, Umweltschutz, Katastrophenschutz, Forschung

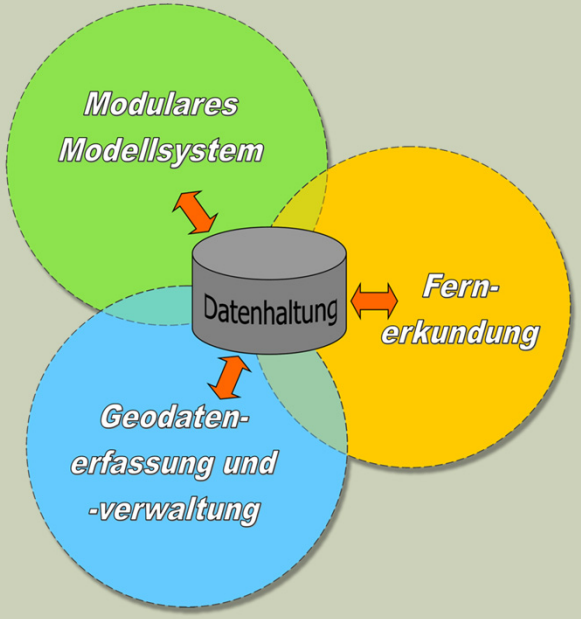
ILMS
Datenbankbezogene Schwerpunkte



- zentrale Datenhaltung
- PostgreSQL/PostGIS für Vektordaten
- Fernerkundung
  - objektorientierte Methoden zur Informationsextraktion aus Satellitendaten (IMALYS)
  - Datenspeicherung in PostgreSQL/PostGIS

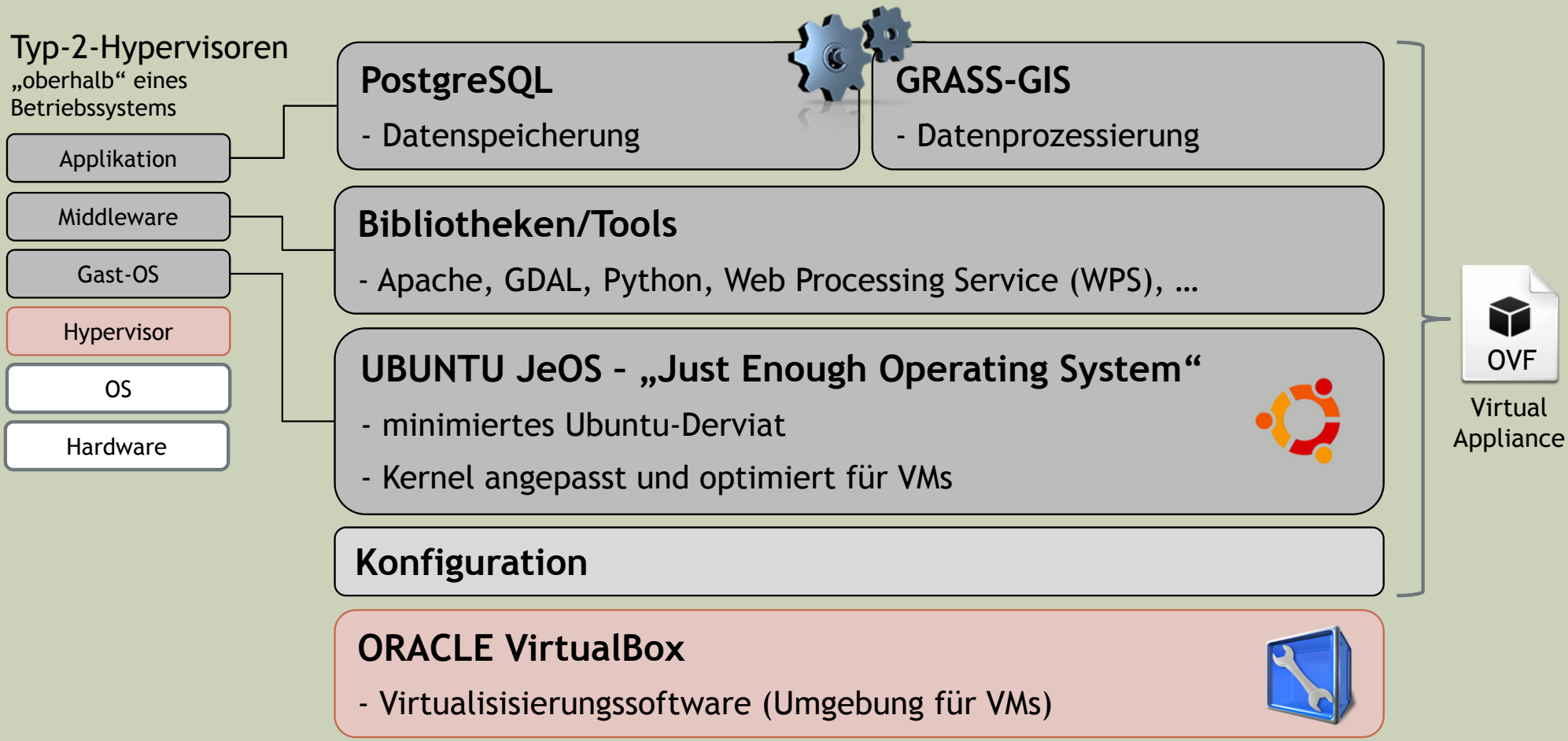
ILMS
Teilkomponenten FE ↔ Rasterspeicherung

- Softwareumgebung als „Virtual Appliance“ verpackt (ILMS*raster*)
  - Desktop- oder Server-Applikation
  - einfache Verbreitung und kein Installationsaufwand, da vorkonfiguriert
  - Virtualisierungssoftware
    - z.B. VirtualBox
  - schnell einsatzbereit durch Standardisierung
- QGIS-Client



- Auswahl an Appliances, z.B. unter
  - http://www.turnkeylinux.org (*TurnKey Linux Virtual Appliance Library*)
  - http://bitnami.org/stacks (*BitNami Application Stacks*)

APPLICATION STACK ILMS*raster*



UMSETZUNG ILMS*raster* (1)
Zielsetzung

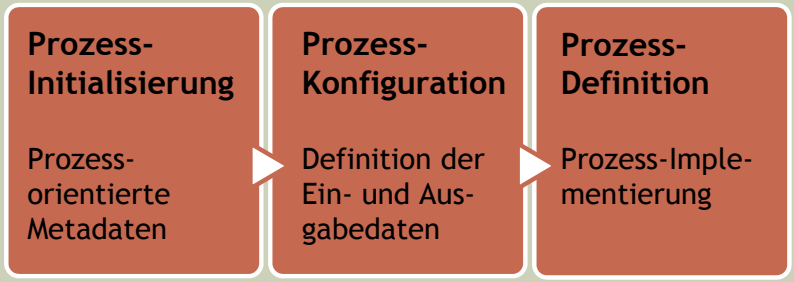
- Vorbearbeitung der Rasterdaten durch GRASS-GIS
  - Kachelung der Rasterszene
  - Erzeugen eines „Grid-Templates“
  - Quicklook-Generierung
- Leichtgewichtige Erweiterung der Datenbank ohne Ergänzung des Typensystems
  - Abbildung von Rasterszenen auf **Datenbanktabellen**
  - Abbildung von Kacheln auf **Tabellenzeilen**
  - eine Python-Datenbankfunktion
    - *StoreImageData(xoff, yoff, xsize, ysize, xsize, ysize, grass_map)*
    - flexibel einsetzbar: einzelne Rasterkacheln, Quicklooks
  - Binary Large Objects (BLOB’s)
- …und WKTraster?

GRASS-GIS

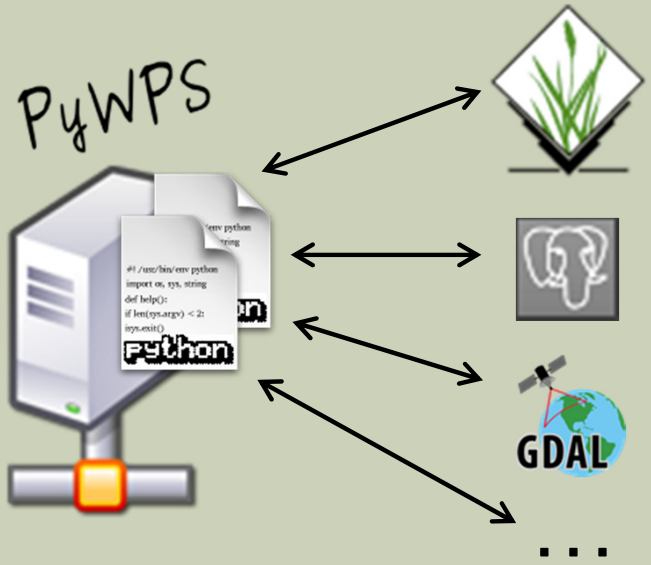
PostgreSQL

UMSETZUNG ILMS*raster* (2)
Web Processing Service

- Verwendung von PyWPS – Implementierung des OGC WPS 1.0.0 in Python
  - standardisiert die Beschreibung und Ausführung von entfernten Prozessen mit Fokus auf räumlichen Daten/Operationen
  - native GRASS-Schnittstelle
  - Python als Sprache zur Implementierung der GIS-Prozesse
- Prozesse



  - zu implementieren
    - Export: QGIS → ILMS-DB
    - Import: ILMS-DB → QGIS



UMSETZUNG ILMS*raster* (3)
PyWPS-Prozess

```
class Process(WPSProcess):
   def __init__(self):
      WPSProcess.__init__(self, identifier = "import_img", title= "Grid Calculation", version = "0.1", statusSupported = True)

      self.img = self.addComplexInput(identifier = "img", title = "Raster image", abstract = „Raster file for import",
                    maxmegabites = "500",
                    formats=[{"mimeType":"image/tiff"}])

      self.pyramids = self.addLiteralInput(identifier = "pyramids", title = "Number of pyramid levels", type = types.IntType)

      self.gridsize = self.addLiteralInput(identifier = "gridsize", title = "Block size in cells", type = types.IntType)
      [...]
      self.debugOut = self.addLiteralOutput(identifier = "debug", title = "Debug Output", type = types.StringType)

      def execute(self):
         self.cmd("r.in.gdal input=%s output=img location=newloc" %(self.img.value))
         self.cmd("g.gisenv set=LOCATION_NAME=newloc")
         [...]
         self.cmd("r.to.vect input=img_mask output=img_mask_v feature=area")
         [...]
         self.cmd("db.execute 'ALTER TABLE ObereGera ADD COLUMN ax DOUBLE PRECISION'")
         [...]
         self.cmd("awk ...")
         [...]
         self.debugOut.setValue("INFO: " + self.cmd("g.gisenv LOCATION_NAME"))
         return
```

Prozess-Initialisierung

Prozess-Konfiguration

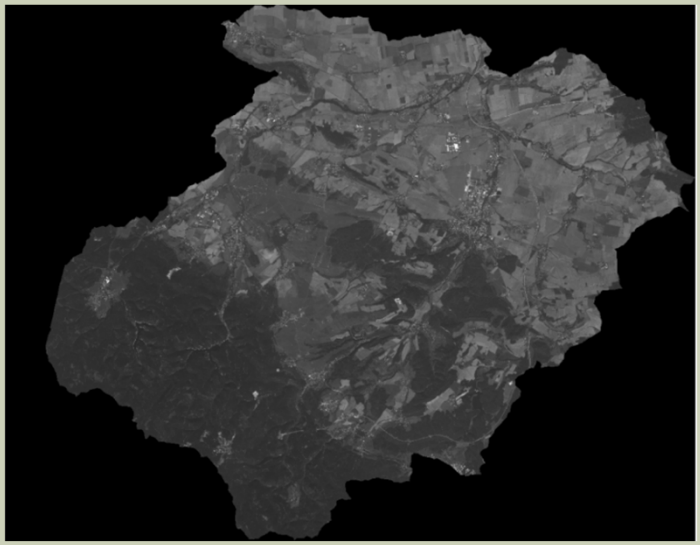
Prozess-Implementierung

UMSETZUNG ILMS*raster* (4)
Prozessanforderungen/Ziele

- Raster-Export:
  - Kachelung
  - verlustfreie Speicherung mit Komprimierung
  - Auflösungsstufen (Pyramiden)
  - Metadaten
  - QGIS-Plugin
- Raster-Import
  - Originalauflösung und Resampling
  - Bounding-Box-Anfragen (BBox)
  - QGIS-Plugin
  - noch wünschenswert:
    - Import von berechneten Karten (GRASS-Funktionen)
    - flexible Ausschnitte (z.B. Einzugsgebiet, räuml. Modelleinheiten)

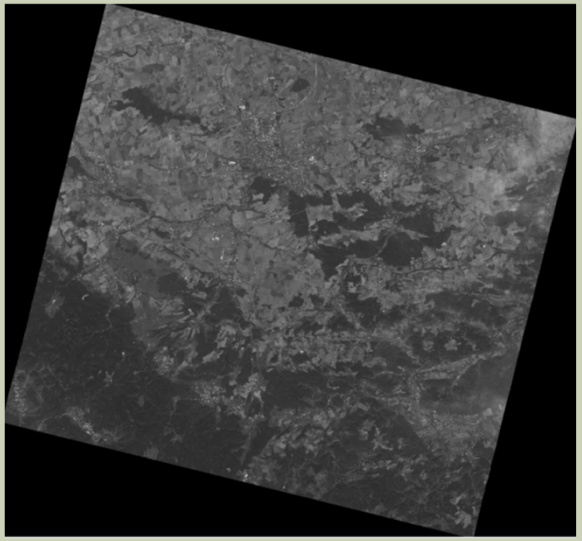
UMSETZUNG ILMS*raster* (5)
Testdatensätze

- Obere Gera (GeoTiff)



  - hydrolog. Testgebiet
  - 58.8 MB
  - 8904 x 6914 Pixel
  - 5m Auflösung

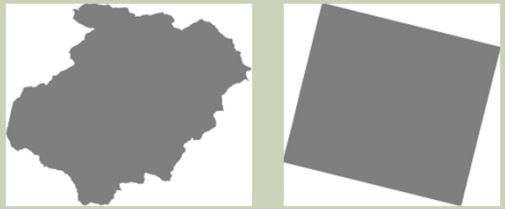
- Thüringen (GeoTiff)



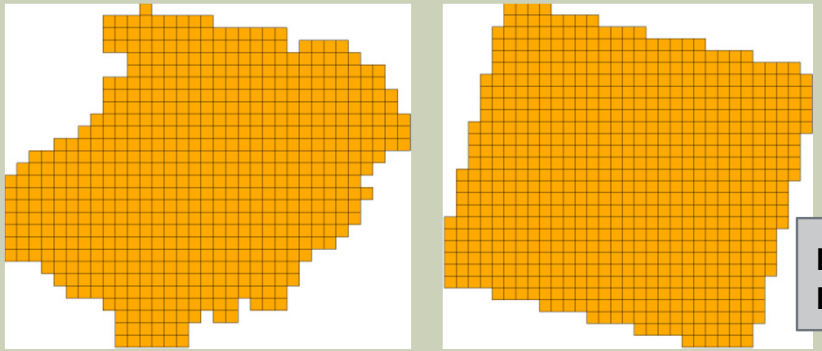
  - 221 MB
  - 15802 x 14740 Pixel
  - 5m Auflösung

UMSETZUNG ILMS*raster* (6)
Raster-Export

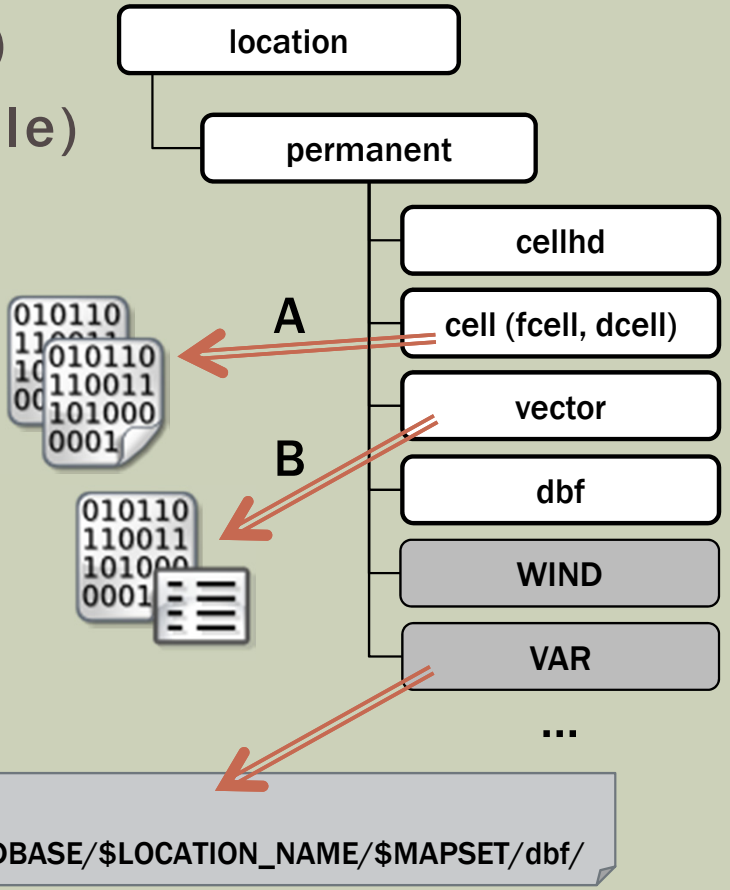
- Export in temporäre GRASS-Location (A)
- Erzeugen der Maske (→ PG-Export-Tabelle)
  - Ausblenden von NULL (Raster, A)



  - erweitertes Overlay mit regelm. Grid (Vektor, B)



  - Konvertierung/Export nach PostgreSQL/PostGIS




```
v.out.ogr input=<grid> type=area olayer=ObereGera
dsn="PG:host=localhost dbname=ilmsraster[...]" format=PostgreSQL
```

UMSETZUNG ILMS*raster* (7)
Raster-Export

- GRASS SQL-Interface (externes Attributmanagement)

DBF	PG	SQLITE	MSYSQL	ODBC
Dbase-Dateien	PostgreSQL	SQLite	MySQL	UnixODBC-Datenquellen
Keine Aggregate, keine mathematischen Funktionen	Kompletter SQL-Support und UDFs			

  - Wechsel zwischen Vektor-Treibern möglich, ggf. Login

```
db.connect driver=pg database="host=localhost,dbname=ilmsraster"
db.login user=postgres pass=postgres
```

DB_DRIVER: pg
DB_DATABASE: host=localhost dbname=ilmsraster

  - Zugriff auf PostGIS-Funktionen
    - Bearbeitung/Erweiterung des exportierten Grids:

CAT	WKB_GEOMETRY
86	POLYGON((4415220.5 5616233.5, 4415220.5 5617513.5, ...))

UMSETZUNG ILMS*raster* (8)
Raster-Export

- Erweiterungen der Grid-Tabelle
  - Informationen über Extent einer Kachel
  - nötig für Binärdaten-Export

Grid-Tabelle „obere_gera“

CAT	WKB_GEOMETRY	AX	AY	BX	BY	XOFF	YOFF	XSIZE	YSIZE
86	POLYGON((...))	4415220.5	5617513.5	4416500.5	5616233.5	3328	6400	256	256

DATA_0	...	DATA_N
\000\231\...	...	\222\233\...

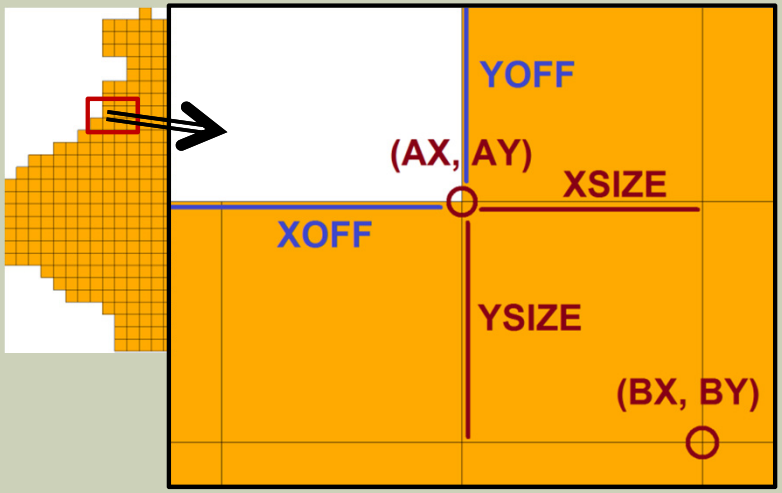
  - Spalten anlegen

```
db.execute "ALTER TABLE ObereGera ADD
  COLUMN ax DOUBLE PRECISION"
```

  - Spalten füllen

```
db.execute "UPDATE ObereGera SET ax =
  X(POINTN(EXTERIORRING(wkb_geometry),2))
```

  - Bilddaten als BLOB in DATA_[0,..,N]



UMSETZUNG ILMS*raster* (9)
Einfügen von Binärdaten

- Realisierung über UDF
  - nutzt GDAL-Treiber für GRASS-Raster (ReadOnly)
  - Definition der Raster über Pfad innerhalb GRASS-Location
    - /tmp/**pywps-2382**/**PERMANENT**/**cellhd**/**ObereGera**

      GRASS-Location **GRASS-Mapset** **Raster-Headerzeilen** **Rasterkarte**
  - Aufruf in SQL Update:

```
db.execute ”UPDATE ObereGera SET data_0 =
  ilms.STOREIMAGEDATA(xoff, yoff, xsize, ysize, xsize, ysize,
  'tmp/pywps-2382/PERMANENT/cellhd/ObereGera')”
```

    - data_0: Daten in Originalauflösung *PX* (z.B.: 5m)
    - data_n: Daten mit Auflösung (2^n*PX), über Kachelgröße (XSIZE, YSIZE) gesteuert:

```
db.execute ”UPDATE ObereGera SET data_n =
  ilms.STOREIMAGEDATA(xoff, yoff, xsize, ysize, (xsize/2^n), (ysize/2^n),
  'tmp/pywps-2382/PERMANENT/cellhd/ObereGera')”
```

UMSETZUNG ILMS*raster* (10)
StoreImageData()-Funktion

- UDF in PL/Python („untrusted“)
  - Keine Einschränkungen in Implementierung
    - IMPORT GDAL, PSYCOPG2 (PostgreSQL-Treiber)
    - liefert BYTEA-Daten pro Kachel

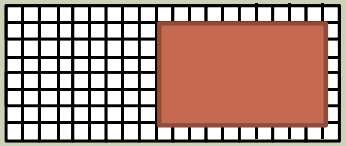
```
[...]
dataset = gdal.Open(raster, GA_ReadOnly)
band1 = dataset.GetRasterBand(1)
scanline = band1.ReadRaster(a, b, c, d, c, d, GDT_Byte)
[...]
```

  - GDAL-ReadRaster(xoff, yoff, xsize, ysize, buf_xsize, buf_ysize) liest Rasterausschnitt flexibel in Buffer
    - xsize(ysize) > buf_xsize(buf_ysize) → Resampling
    - einsetzbar für Originalauflösung, Auflösungspyramiden, Quicklooks

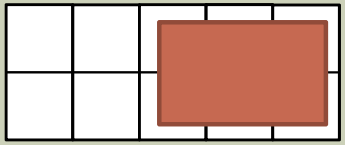
UMSETZUNG ILMS*raster* (11)
Kachelgröße des Grids

- Problem
  - ideale Kachelgröße (128, 256, 512 … ?)
  - Ergebnismenge einer BBox-Anfrage: überlappende Kacheln
  - GDAL Merge
    - große Kacheln → geringere Anzahl zum Laden, hohes Datenaufkommen pro Kachel
    - kleine Kacheln →höhere Anzahl zum Laden, kleinere Datenaufkommen pro Kachel

- Auflösungsstufe (Raster-Import)
  - abhängig von Anzahl der Kacheln in Ergebnismenge (?)



37.5 %



60 %

UMSETZUNG ILMS*raster* (12)
Metadaten-Tabelle und Vorschaubilder

- Metadaten
  - SQL-Update im Export-Skript

Tabelle „ilms_raster“

RASTER_ID	DESC	QUICKLOOK	PYRAMIDS	DX	DY	TYPE	ORIG_X	ORIG_Y	WIDTH	HEIGHT
obere_gera	EZG Obere Gera	\203\226\252\210sh	2	5	5	Byte	4398580.5	5649513.5	8448	7168

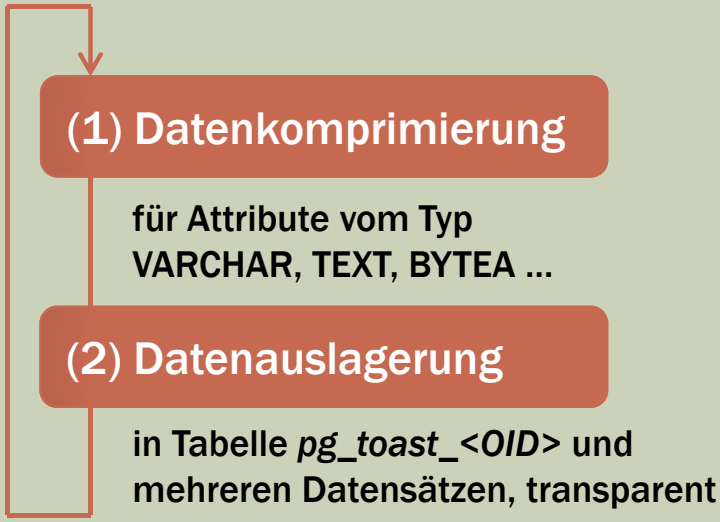

Grid-Tabelle

- Previews/Quicklooks (im QGIS-Client)
  - nötig für Auswahl des Rasterausschnitts (BBox)
  - erstellt in GRASS-GIS als PNG (zuvor Resampling der Location)
  - SQL-Update über *StoreImageData()* mit konstanten Ziel-Buffer
    - PNG-Bildbreite: 250 Pixel

```
db.execute ”UPDATE ilms_raster SET quicklook =
  ilms.STOREIMAGEDATA(0, 0, 250, 212, 250, 212,
  'tmp/pywps-2382/PERMANENT/cellhd/ObereGera')”
```

UMSETZUNG ILMS*raster* (13)
Binärdatenverwaltung in PostgreSQL

- TOAST
  - PostgreSQL >= 7.1
  - „The Oversized-Attribute Storage Technique“
  - Tabellenzeile > Blockgröße / 4



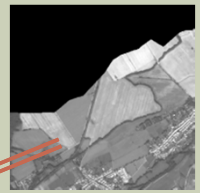
- LargeObject-Schnittstelle (LO)
  - PostgreSQL < 7.1
  - Verwendung serverseitiger LO-Funktionen zum Erzeugen, Einfügen, Lesen, Schreiben und Entfernen großer Datenobjekte

Datenzerlegung

  - auf System-Tabelle *pg_largeobject*

UMSETZUNG ILMS*raster* (14)
TOAST-Daten in PostgreSQL

Grid-Tabelle „obere_gera“



CAT	...	DATA_0	DATA_1	DATA_2
51	...	\222\233\...	...	...

8 KB – 32 KB

LZW/LZW2-Komprimierung

TOAST-Tabelle PG_TOAST_XXXXX

CHUNK_ID	CHUNK_SEQ	CHUNK_DATA
...	...	...

- 1 Zeile ~ 2000 Bytes (TOAST_MAX_CHUNK_SIZE)

Beispiel „Obere Gera“ (~ 59 MB):
(+ 2 Auflösungsstufen)

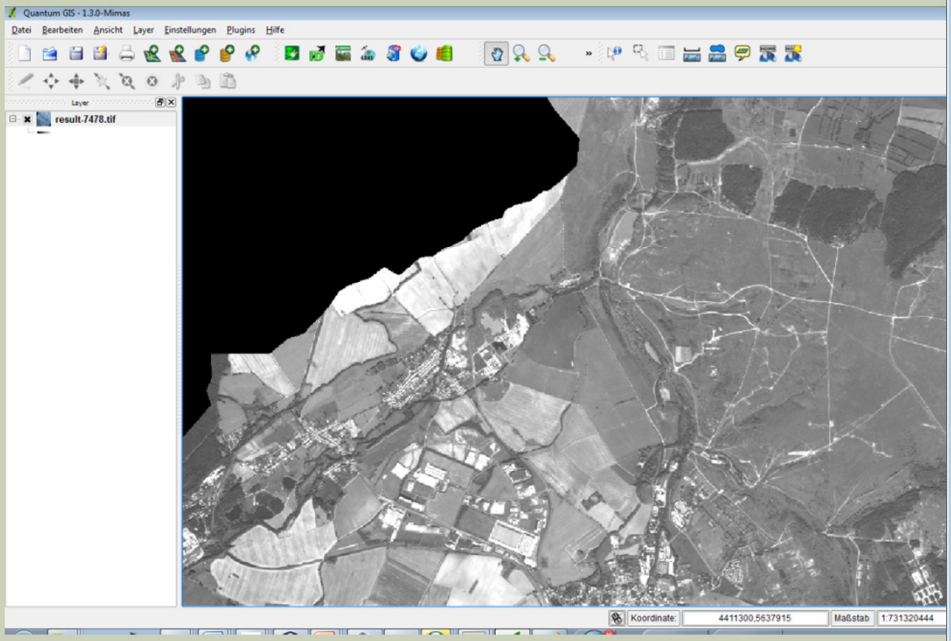
```
SELECT relfilenode FROM pg_class
WHERE relname = 'obere_gera';
 relfilenode
-------------
      340901
```

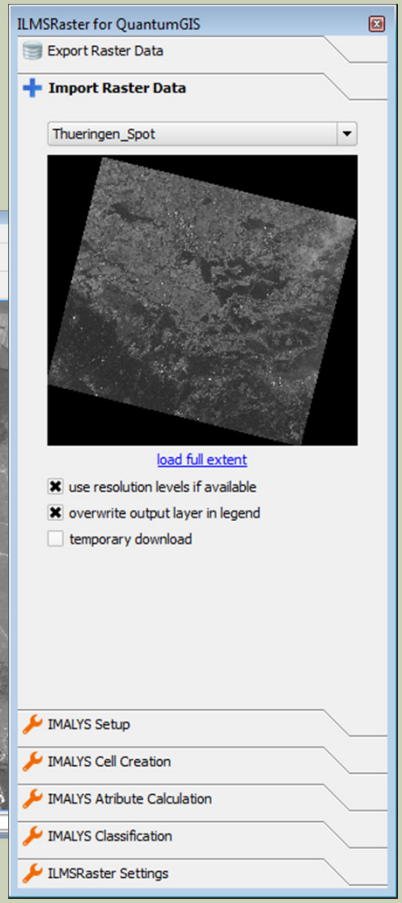
```
SELECT
pg_size_pretty(pg_total_relation_size('obere_gera'));
 pg_size_pretty
----------------
 36 MB
```

```
SELECT relname, relpages FROM pg_class
WHERE relname = 'pg_toast_340901' OR relname = 'pg_toast_340901_index' OR
relname = 'obere_gera';
      relname          | relpages
-----------------------+----------
 pg_toast_340901       |     4549
 pg_toast_340901_index |       52
 obere_gera            |       34
```

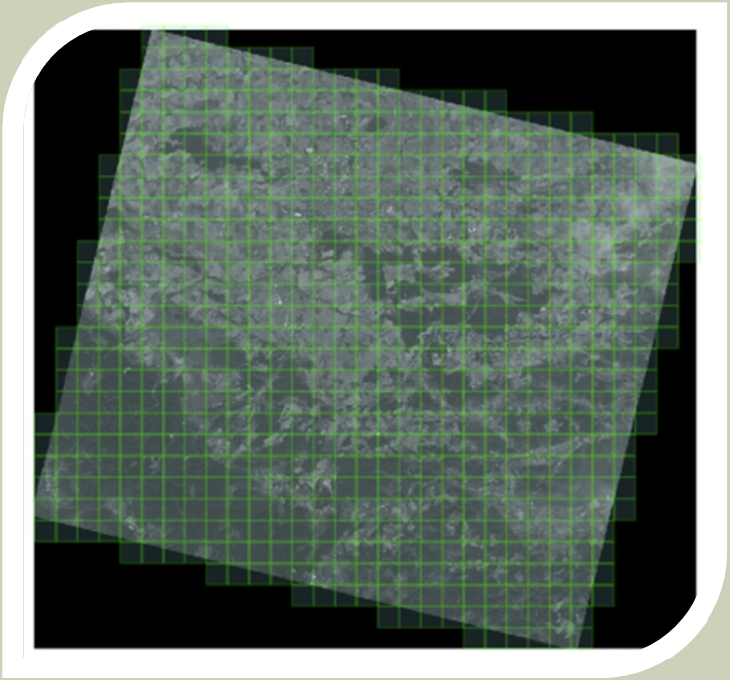
UMSETZUNG ILMS*raster* (15)
QGIS

- QGIS-Client
  - umfasst IMALYS-Schnittstelle (FE-Software) und ILMS*raster* (Raster-Schnittstelle)
  - Plugin in Python und PyQt
  - DB-Unabhängigkeit durch SQLObject





VIELEN DANK!



DEMO...