RASTER DATA HANDLING IN SPATIAL DATABASES: The Case for Images

Lúbia Vinhas
Ricardo Cartaxo
Gilberto Camara
Karine Ferreira
Antonio Miguel Vieira Monteiro

DPI/INPE
MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

An Outline of this Talk

- INPE's Motivation
- The Rationality for Having Images Stored in DBMS
- The Challenges
- Our Solution and Where We Are at this Stage
- Algorithm Development: API for Images Spatial Operations
- Conclusion and Future Works

INPE's Motivation

- Satellite Acquired Data is Everywhere !!
- Satellite Derived Observational Data
  - Large Mass of Highly Dimensional Spatio-Temporal Data
- 30 Years of lessons learned from dealing with *High Dimensional Spatio-Temporal* **Image Data** from Earth Remote Sensing Satellites and Airborne Sensors.
- INPE's Image Data Centre Project

The Rationality for Having Images Stored in DBMS

- A New Generation of Spatially Enabled DBMS;
- Huge Amount of Data that must be Dealt with, coming from a Variety of Sensors over a variety of plataforms;
- Make Data recovery and Integration a more easy Task;

The Challenges

- Technological Challenges:
  - Efficient Spatially Enabled DBMS
  - Provide spatial operations on spatial data types stored in different DBMS

- Scientific&Technological Challenges:
  - Methods and Techniques for Parameter/Pattern/Information-Content Extraction from High Dimensional Integrated Spatio-Temporal Datasets

The Challenges:
The Applications Needs driving the Technology Needs

- Run in a corporative environment
- Access data by internet and intranet
- Typical use of image data is visualization
- Integrates descriptive data stored in a conventional object-relational DBMS
- Integrates vector data

The Challenge:
The Basic Requirements

- The Image Data should be stored in the *existing object-relational database management system*
  - Data integrity and consistency
  - Independent and effective access by users of multiple applications

The Challenges:
The Research Needs driving the Scientific Needs

- Parameter/Pattern/ Information-Content Extraction:
  - Another Typical use of image data is getting *information* out of it:

    Needs: New Methods and Algorithms

Our Aim ...

- Provide a Research Testbed for Dealing with Large Raster Datasets that can help in:
  - Enabling Data Integration. Grid Data, Image Data, Observations Data and other Geographic Data types could be used together;
  - Enabling easy new algorithms development for *parameter* extraction from Satellite Image Datasets;
  - Enabling the test of new spatial-temporal statistics methods for "mining" high dimensional datasets

… and Where we are at this Stage

- Advances in database technology provide support for major advances in non-conventional database applications

- Spatial Data in Relational Databases
  - Integration of spatial data types in object-relational database management systems
  - Efficient handling of spatial data types
    - vector: polygons, lines and points
    - *Raster Data Structures: Images or any other Gridded data*
  - Tools for query and manipulation of spatial data

It is Time for Images…

- A special interest in the spatial databases community is the efficient handling of raster data

- An approach is to develop specialized image data servers
  - Main advantage: the capacity of performance improvements

Our Approach

- Include Building *Raster Data Management* capabilities into Object-Relational Database Management Systems

- Main advantages:
  - easy interface with existing user environments
  - To accommodate not only typical *Image* Data, but also Raster Data in general

Our Solution …

- Our Technological Solution:

TerraLib

(http://www.terralib.org)

TerraLib

- This work is part of the development of
  - **TerraLib** is an Open Source Licenced (LGPL) Geographic Library for providing support for the development of Geographic Applications powered by Spatially enabled DBMS

- Main features:
  - Geometry is stored and managed in the DBMS
  - Facilities supported in differents DBMS as ORACLE, PostgreSQL, MySQL, ORACLE Spatial, PostgreSQL/PostGIS, MS Databases through ADO

TerraLib

- Interface with DBMS

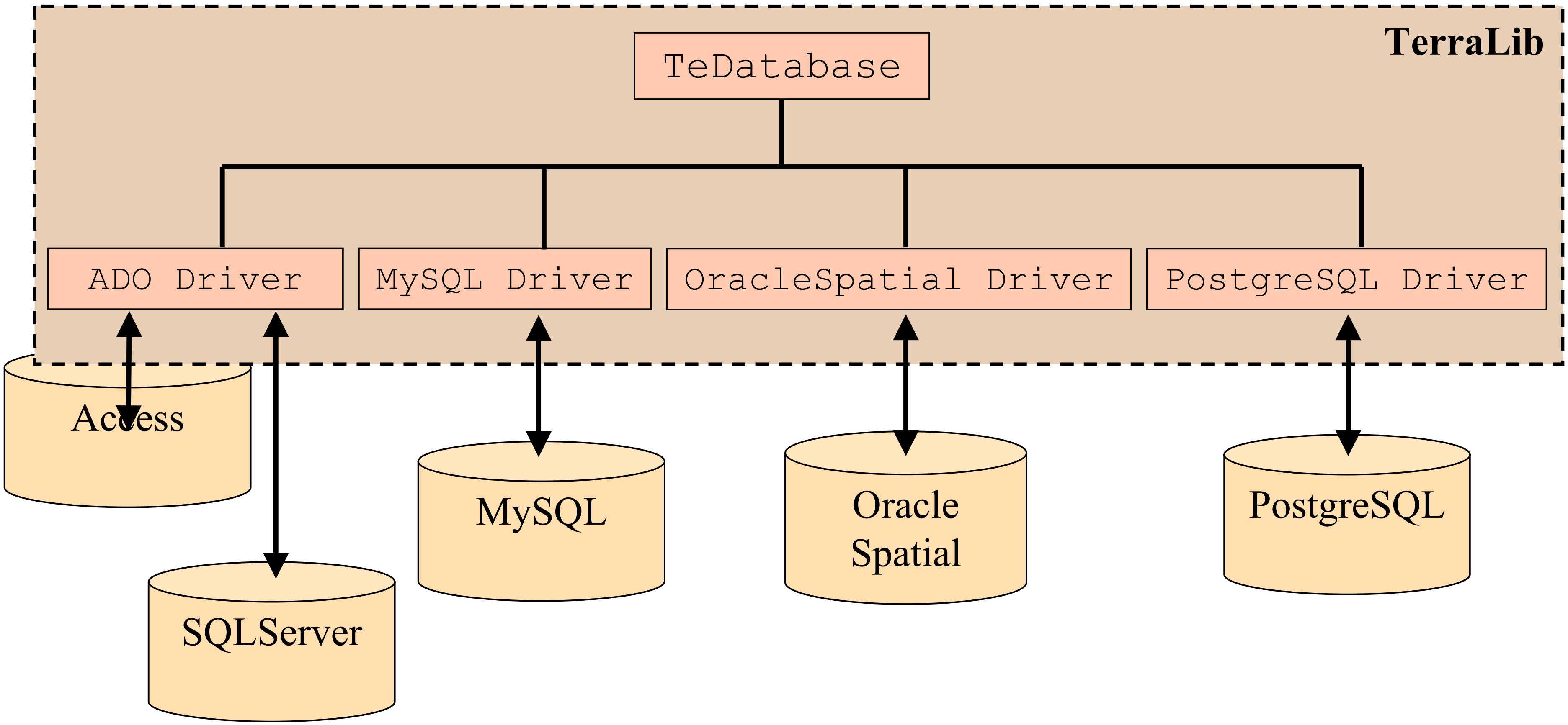


Image (Raster) Data Needs

- efficient storage and indexing mechanisms
- decoding of the different image data formats
- basic data manipulation functions
- convenient ways of accessing the image data by algorithms

Two Main Aspects

1. A DBMS Data Model
   - Tables schema
   - Spatial indexing
   - Support to compression
2. A set of C++ classes to allow applications to deal with Raster Data
   - Efficiency and flexibility to access the data

DBMS Data Model

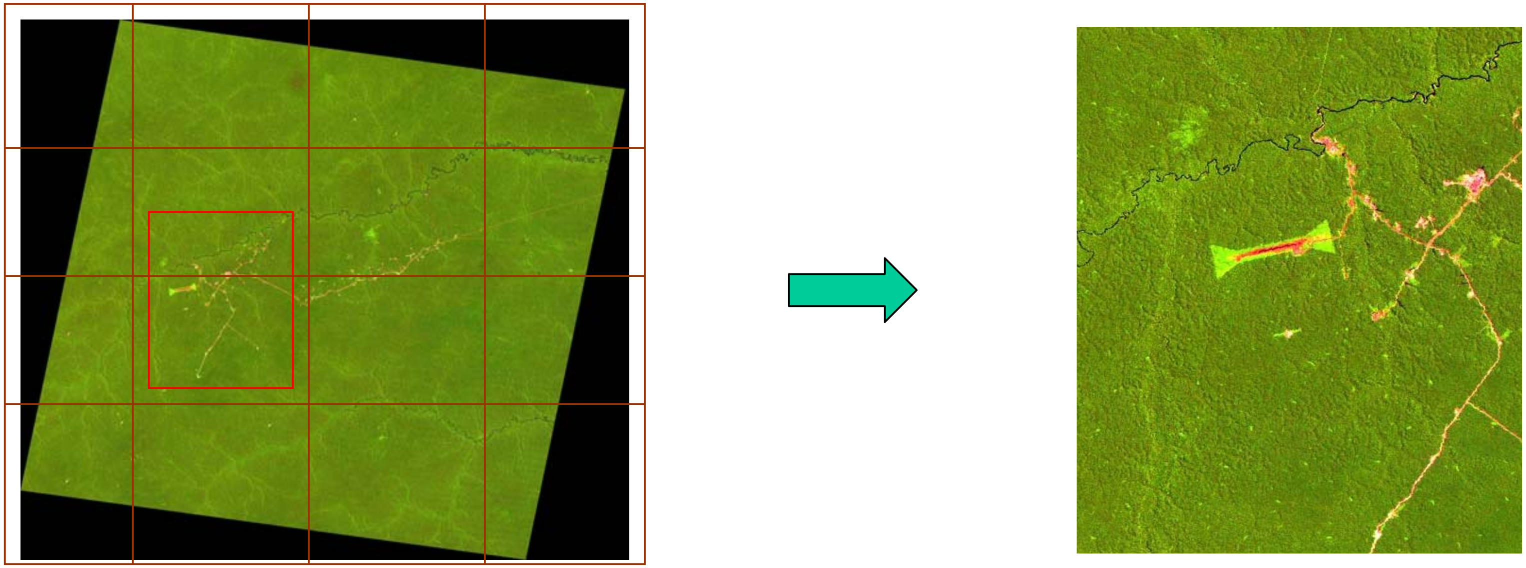
- Defines, at a physical level, how to store raster data in a object-relational database

- An ineffective approach:
  - Store each point of the image in a row of a table [x,y,z]

- Another approach:
  - The entire image is written to a <u>blob</u> and stored in a field of a table
- A variation of the second approach was adopted:
  - *Tiles* of image are written to a <u>blob</u> and stored in a field of a table

Tiling

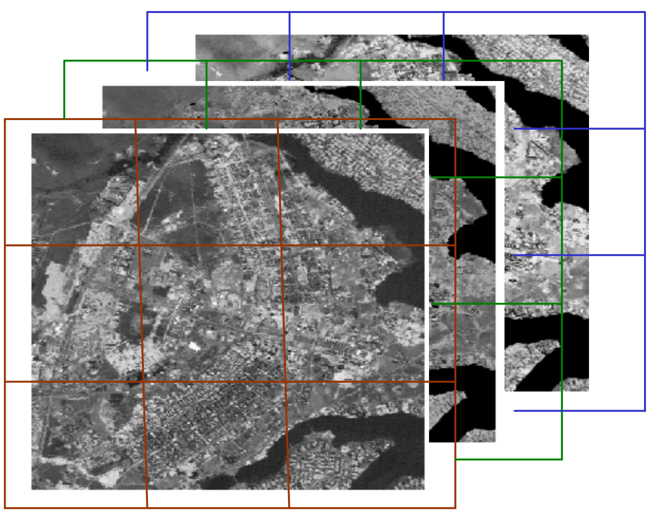
- Specific parts of the image can be retrieved and processing independently
- User control over the size of the tiles
- Example: zooming operation



MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

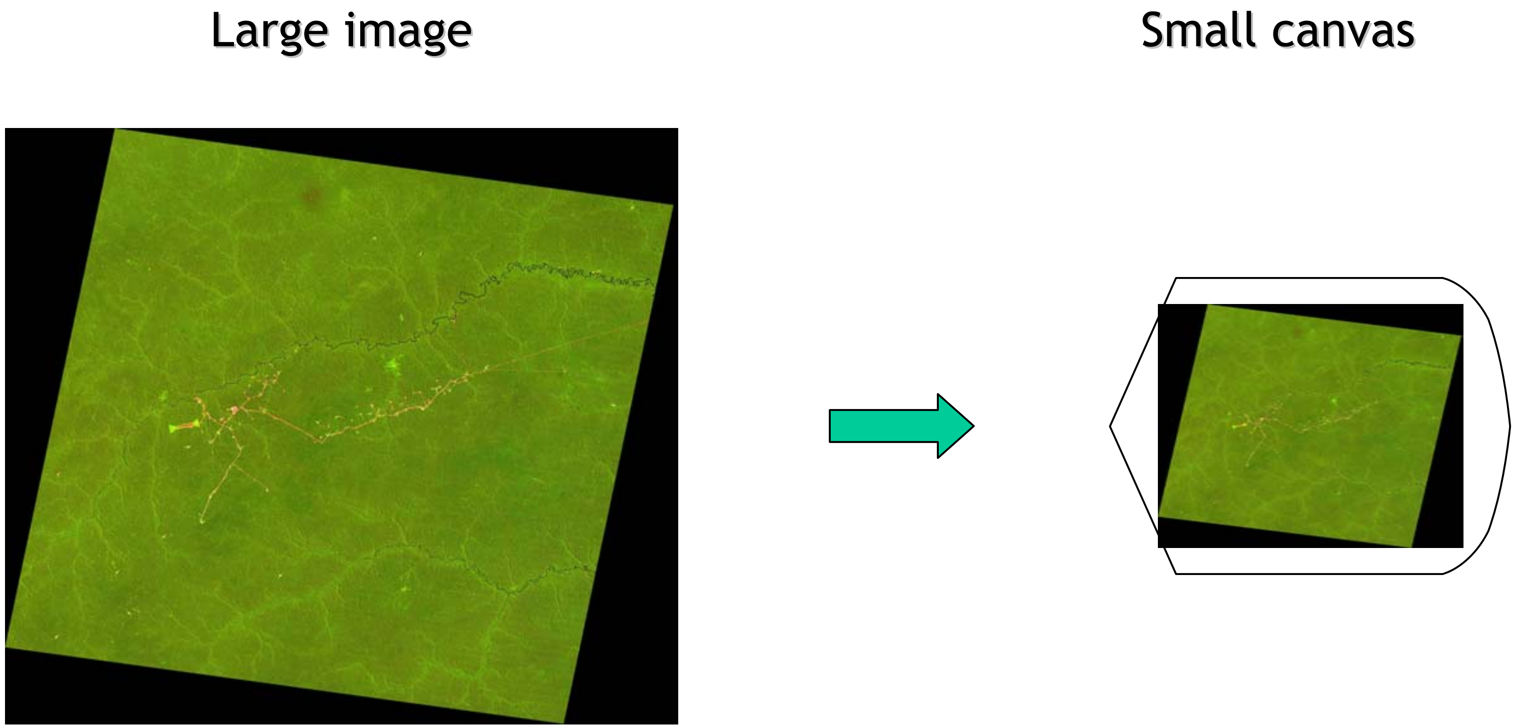
Tiling → DBMS Data Model

- Each raster data is stored in a table
- Each row stores a *tile* of a particular band



tile_id	band	blob
T1	1	...
T1	2	...
T1	3	...

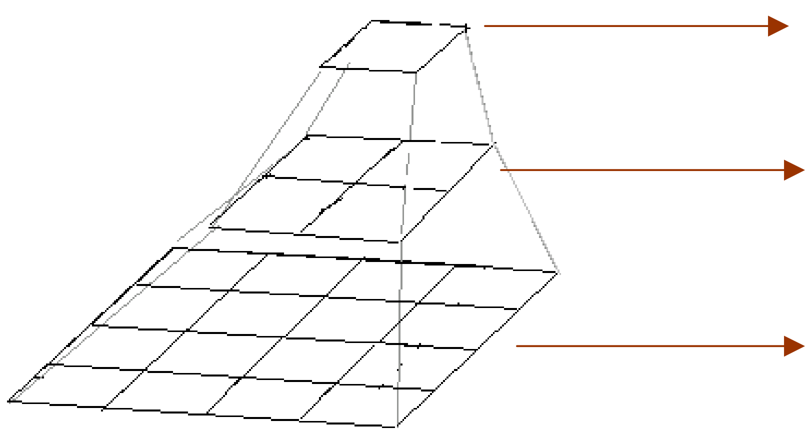
Multi-resolution



- Image is shown with a degraded resolution
- Much of the information retrieved is not used

Multi-resolution

- Lower resolution versions of the image are also stored in the database
- Application decides the best resolution level to be retrieved
- User control of the number of resolution levels



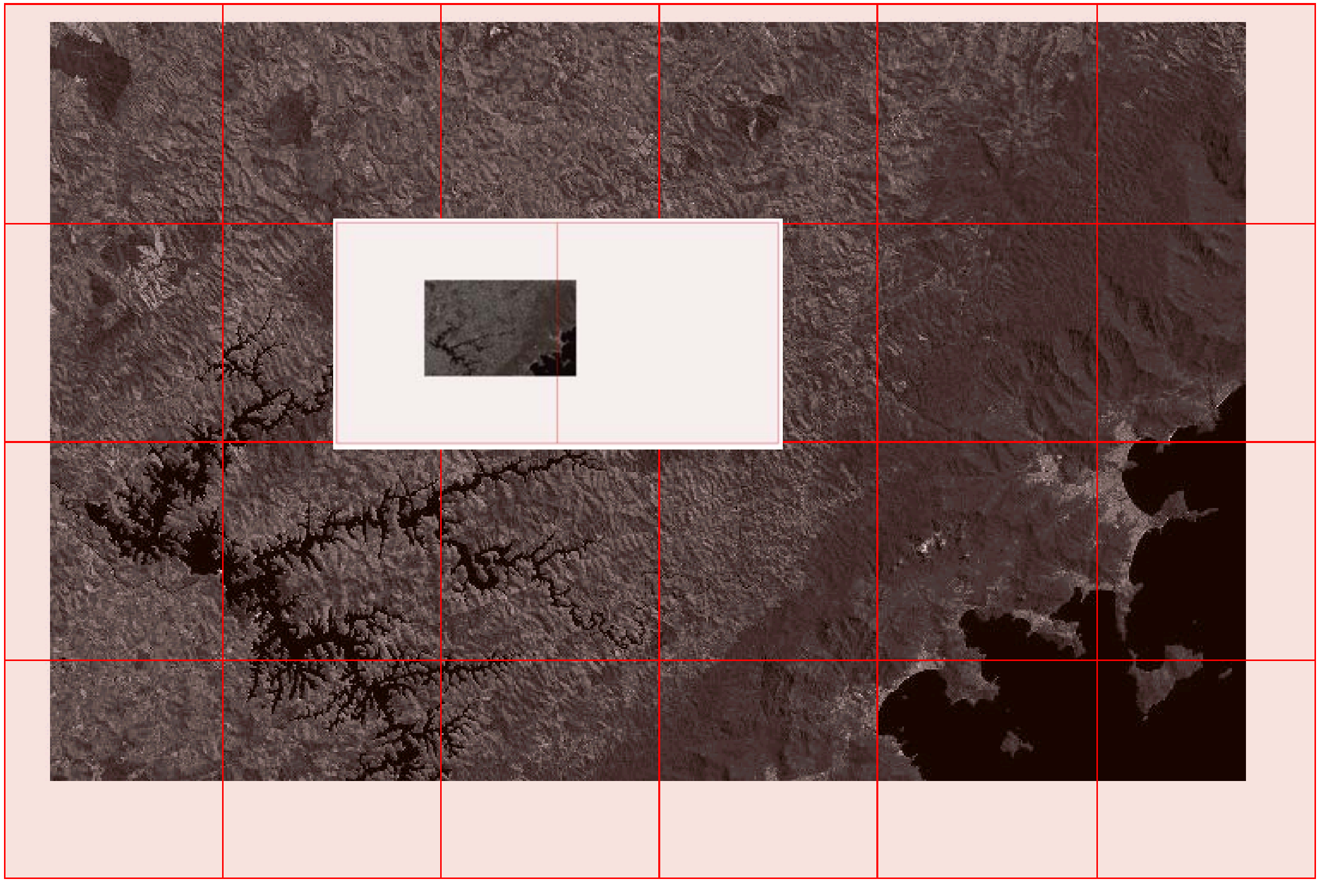
Level 2 → Resolution $R * 2^2$

Level 1 → Resolution $R * 2^1$

Level 0 → Resolution $R * 2^0$ (original)

Multi-resolution

- To store an image in a lower resolution less *tiles* are needed



30 m

60 m

120 m

240 m

Multi-resolution

- Each row of a Raster table contains information about the level of resolution of the *tile*

tile_id	band	resolution_factor	blob
T1	1	0	...
T1	1	1	...
T1	2	0	...
T1	2	1	...
T1	3	0	...
T1	3	1	...

Spatial Indexing

- For each *tile* the coordinates of its bounding box are stored
- Using a SQL statement an application can select the *tiles* that intercept a given area in a given resolution level

tile_id	band	resolution_factor	lower _x	lower_y	upper_x	upper_y	blob
T1	1	0					...
T1	1	1					...
T1	2	0					...
T1	2	1					...
T1	3	0					...
T1	3	1					...

```
SELECT * FROM raster_table
WHERE NOT (lower_x > 10 OR upper_x < 20 OR lower_y  > 10 OR  upperY  < 20 )
AND resolution_factor = 0
```

Accessing Pixels Individually

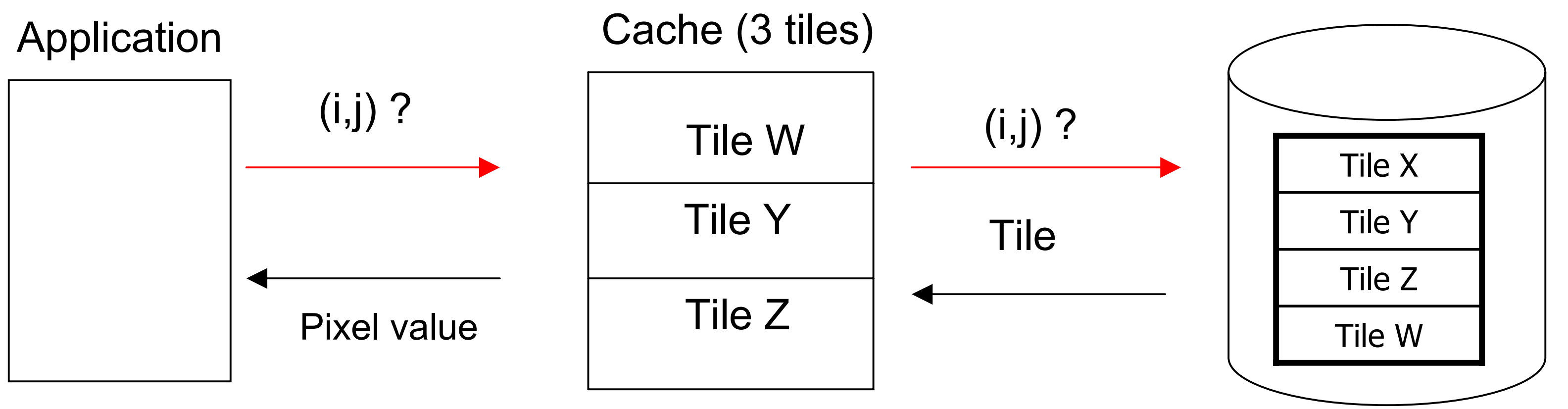
- Typical image processing algorithm:

```
for i=0 to num rows
  for j=0 to num cols
   process Image(i,j)
```

- To query the database for each pixel of image can be costly
- Solution: keep a cache of tiles in memory

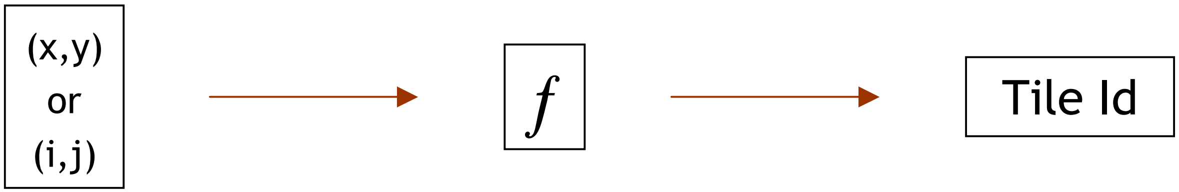
Virtual Memory

- Optimize the access of pixels of an image
- *Tiles* in memory have the same identification of the database



Tiles Identification

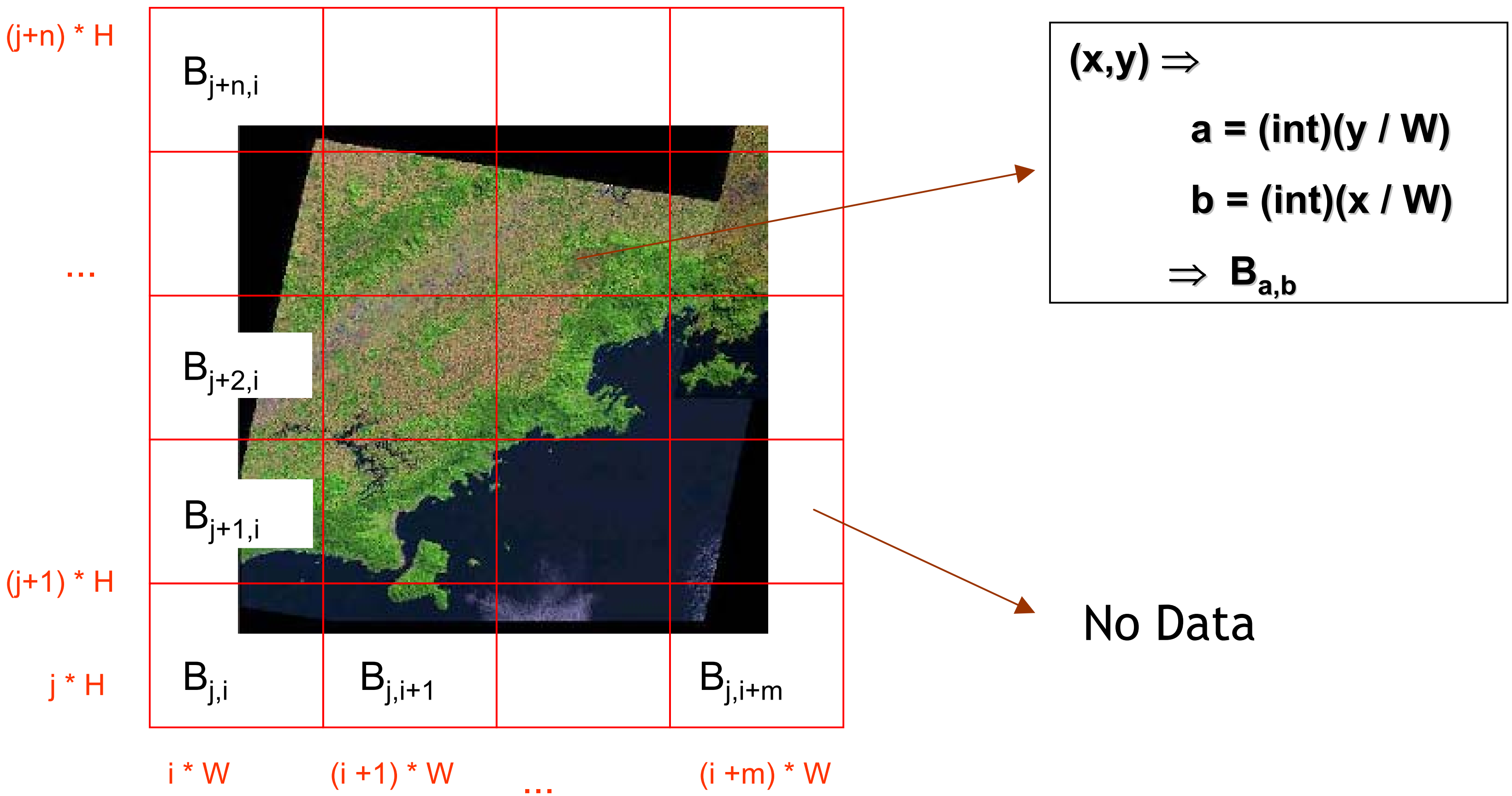
- A unique identification for each *tile*



- The function should return the same identification for every pixel that belongs to a *tile*
- The identification of *tiles* should remain consistent over mosaic operations

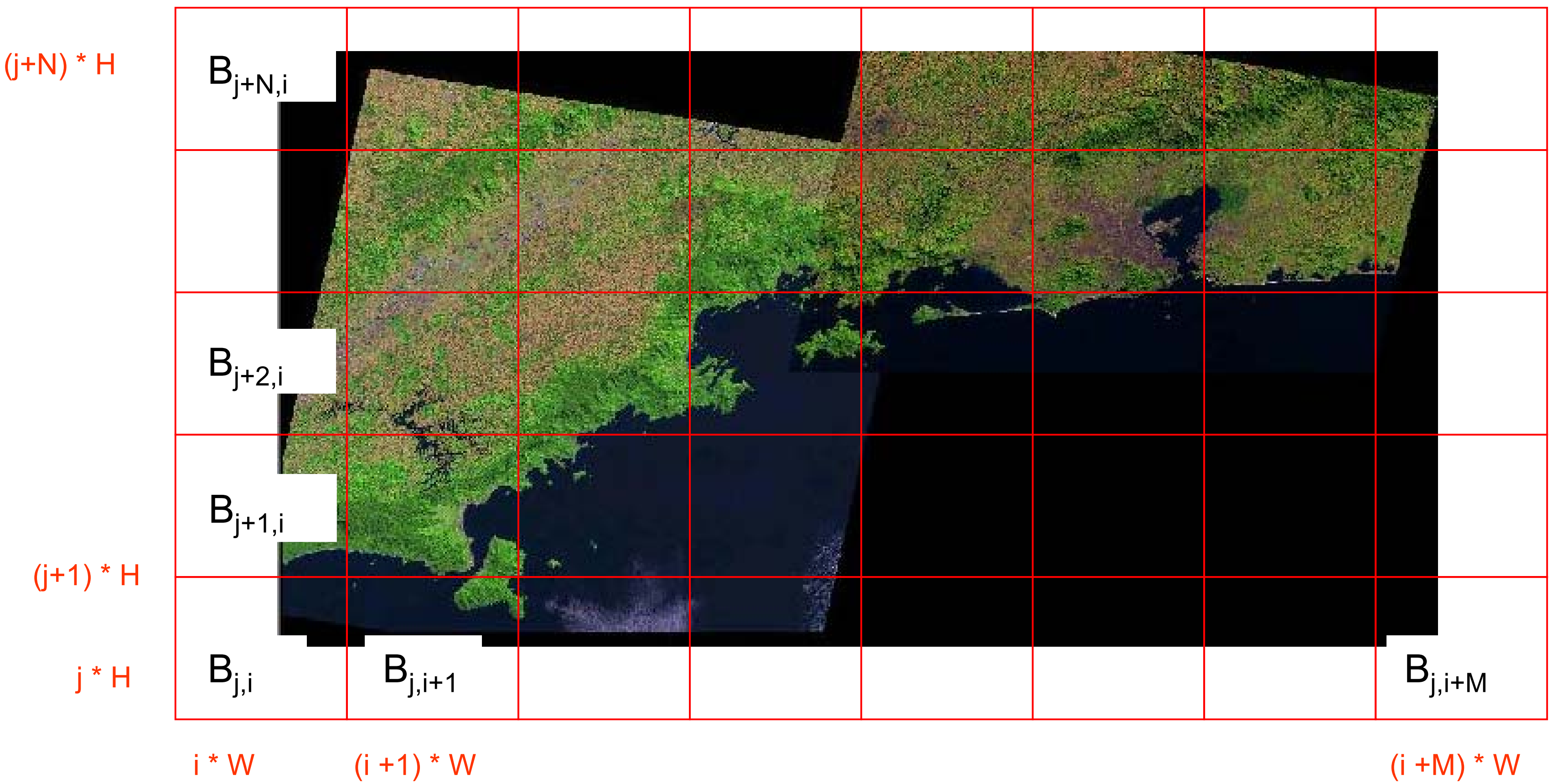
Tiles Identification

Tile size: **W** × **H** (in geographical units. I.e.: 1536m × 1536m)



Tiles Identification

Images can "grow" and identification of the *tiles* remains consistent

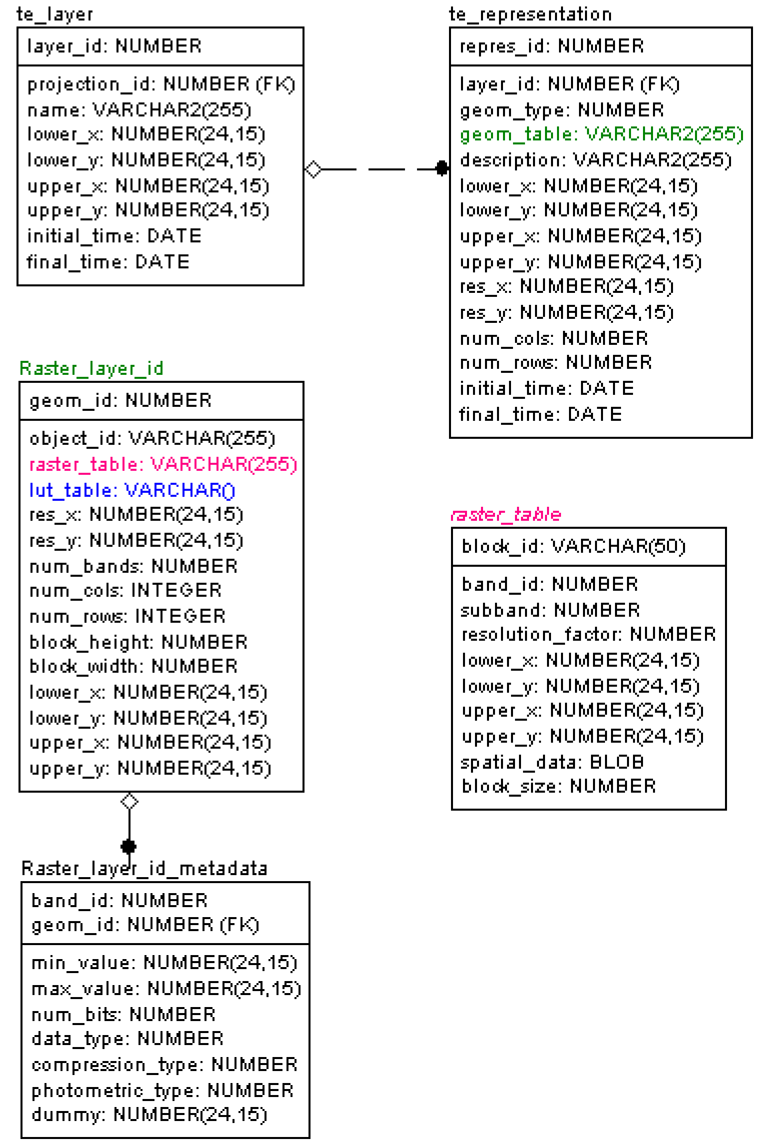


Compression

- *Tiles* can be compressed before stored in the database
- Compression techniques: Zlib, JPEG or wavelets
- An image of de 1778x2804 pixels (4985512 pixels), 1 band, X and Y resolution of 25m, stored in tiles of 512x512 pixels:
  - No compression - 6291456 bytes
  - ZLIB - 3746080 bytes (~59.0%)
  - JPEG 75% - 814694 bytes (~12.5%)

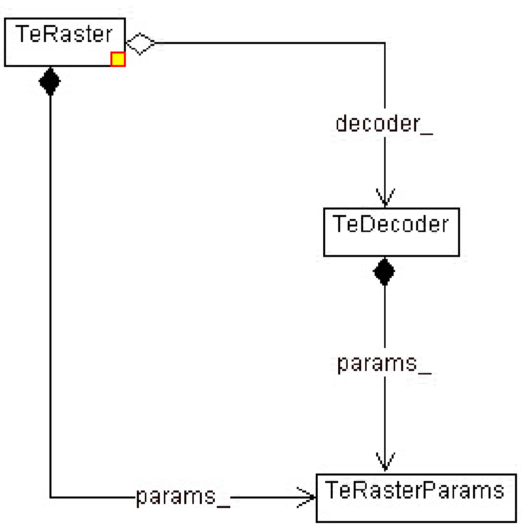
Metadata

- Database should also store <u>metadata</u> of the images in auxiliary tables



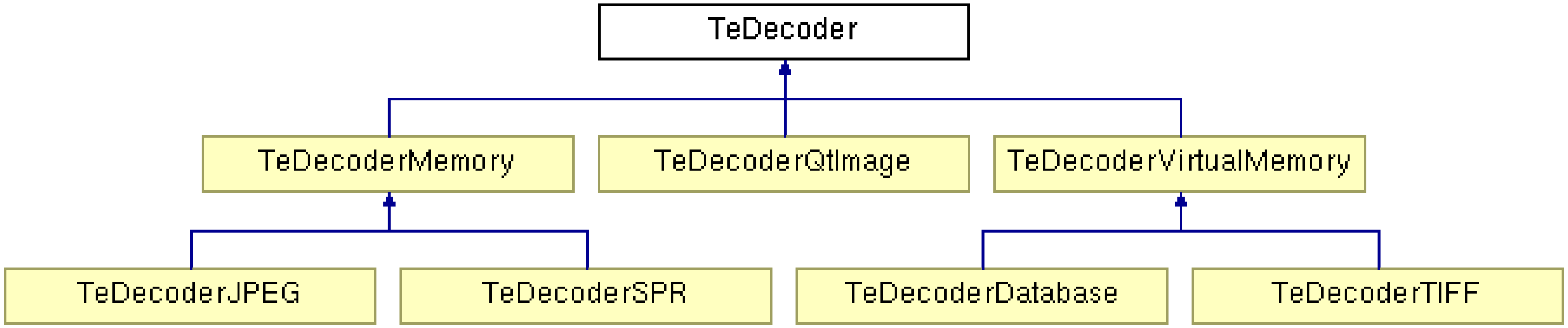
API Raster **TerraLib**

- **TerraLib** provides a set of C++ classes do deal with Raster Data
- Class `TeRaster`
  - Grid values are double
  - Methods `getElement` and `setElement` access elements of a Raster
- Class `TeRasterParams`
  - Information about a Raster representation
- Class `TeDecoder`
  - `Strategy Pattern`: allows the access to different formats and storage aspects



Decoders

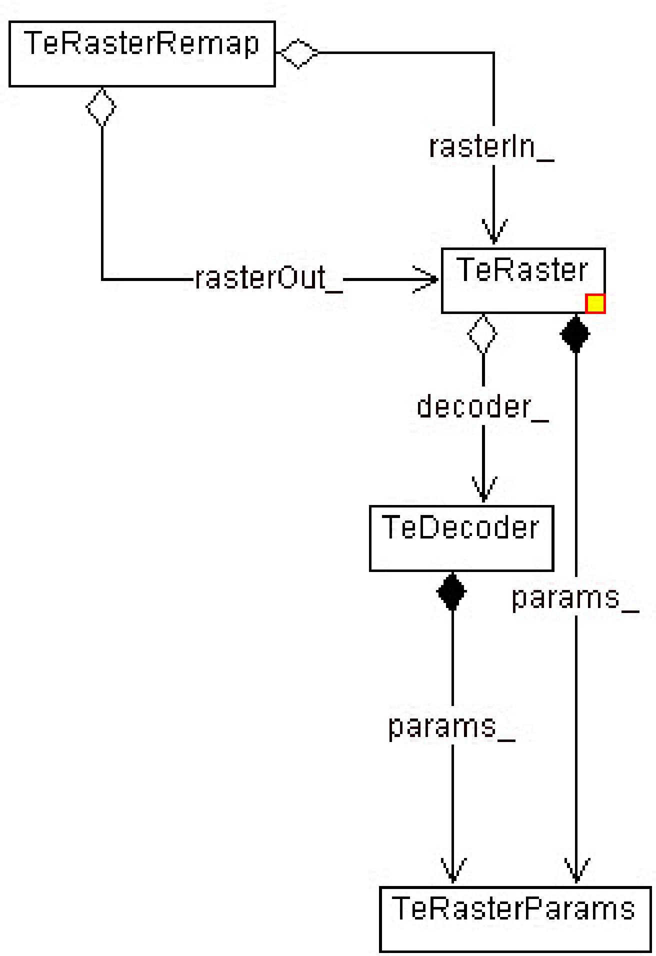
- Encapsulates the access to the elements of a Raster data
- Explicitly instantiated or defined from a file name for example
- Extensible



Manipulation

- Functions to import raster data into the database
- Class `TeRasterRemap` make a copy of a Raster Data solving differences in
  - projections
  - bounding boxes
  - resolutions

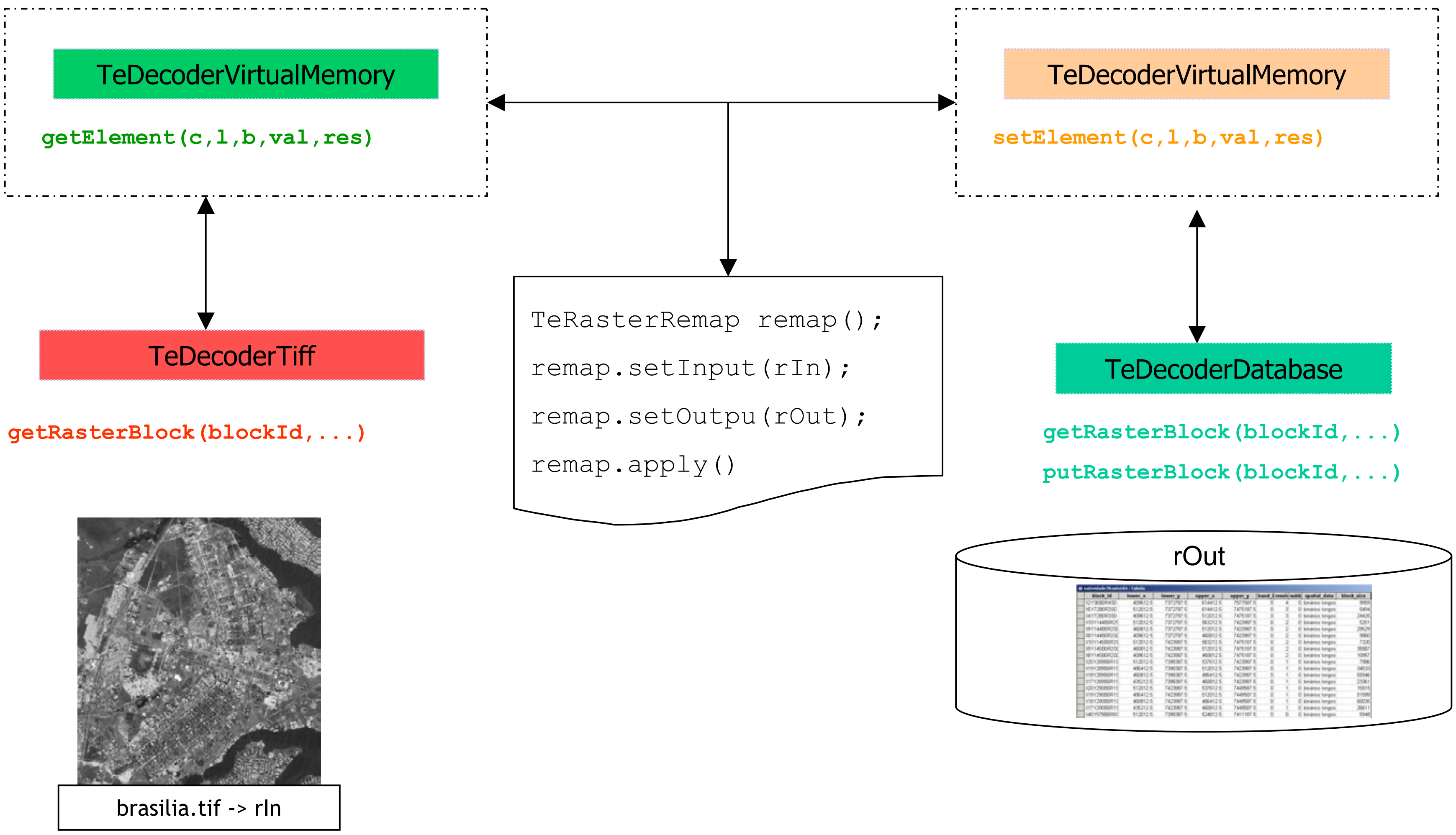
Manipulation



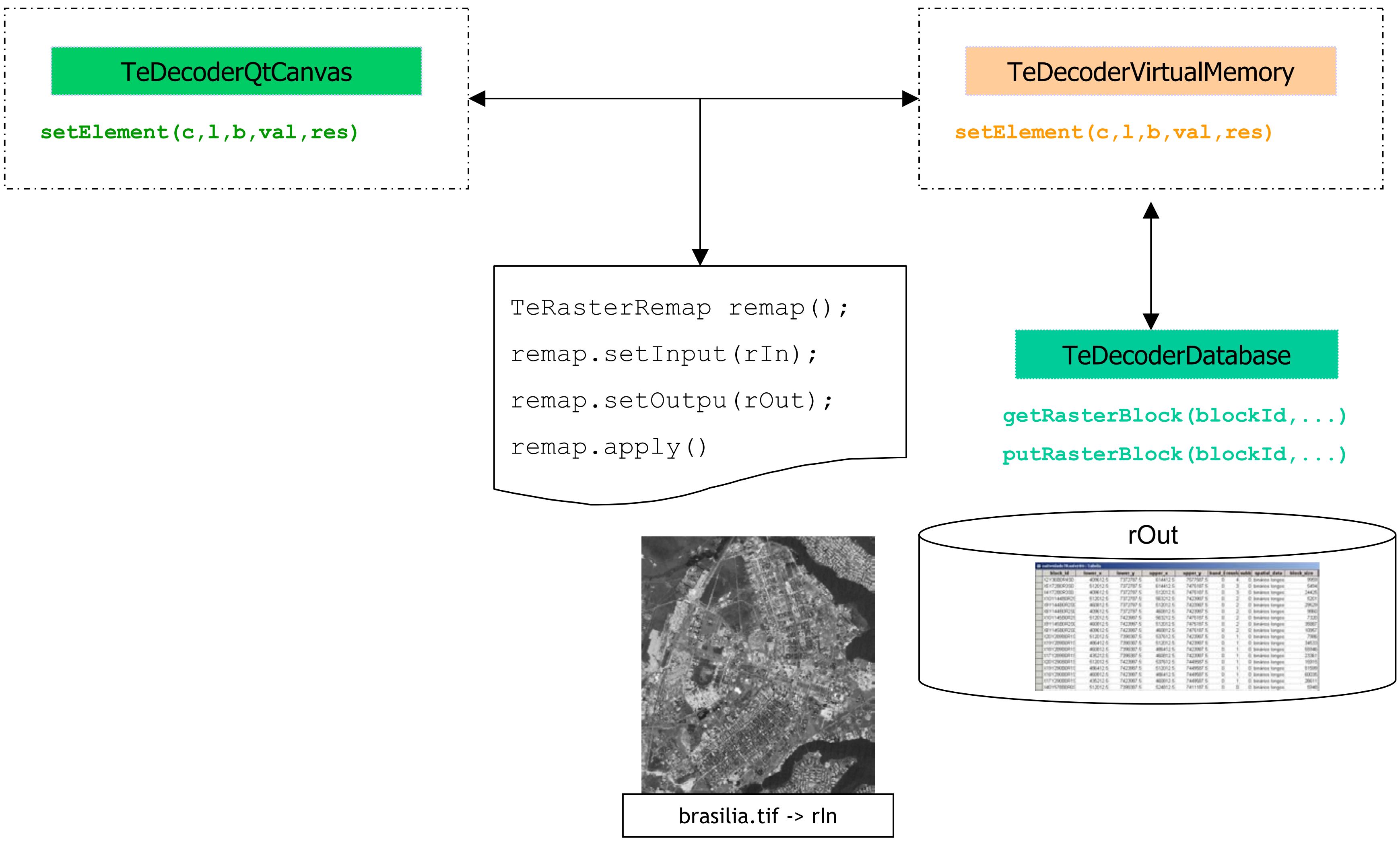
`TeRasterRemap :`

- Import from file to database
- Clipping
- Mosaic
- Visualization
- Reprojection

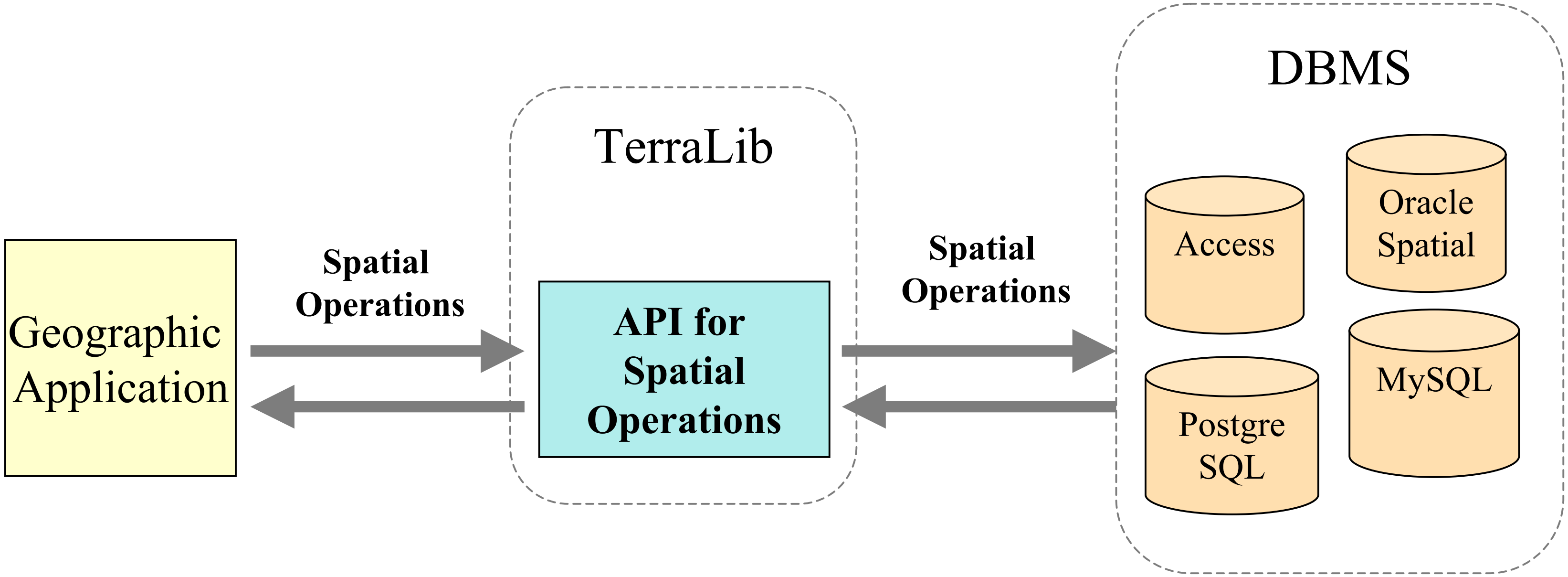
Importing


TeDecoderVirtualMemory
getElement(c,l,b,val,res)
TeDecoderVirtualMemory
setElement(c,l,b,val,res)
TeDecoderTiff
getRasterBlock(blockId,...)
TeRasterRemap remap();
remap.setInput(rIn);
remap.setOutpu(rOut);
remap.apply()
TeDecoderDatabase
getRasterBlock(blockId,...)
putRasterBlock(blockId,...)
rOut
brasilia.tif -> rIn

Visualization



API for spatial operations on Images



API – Zonal Operation

- Calculates statistics over a region or a zone of a Raster Data

Estatísticas do raster

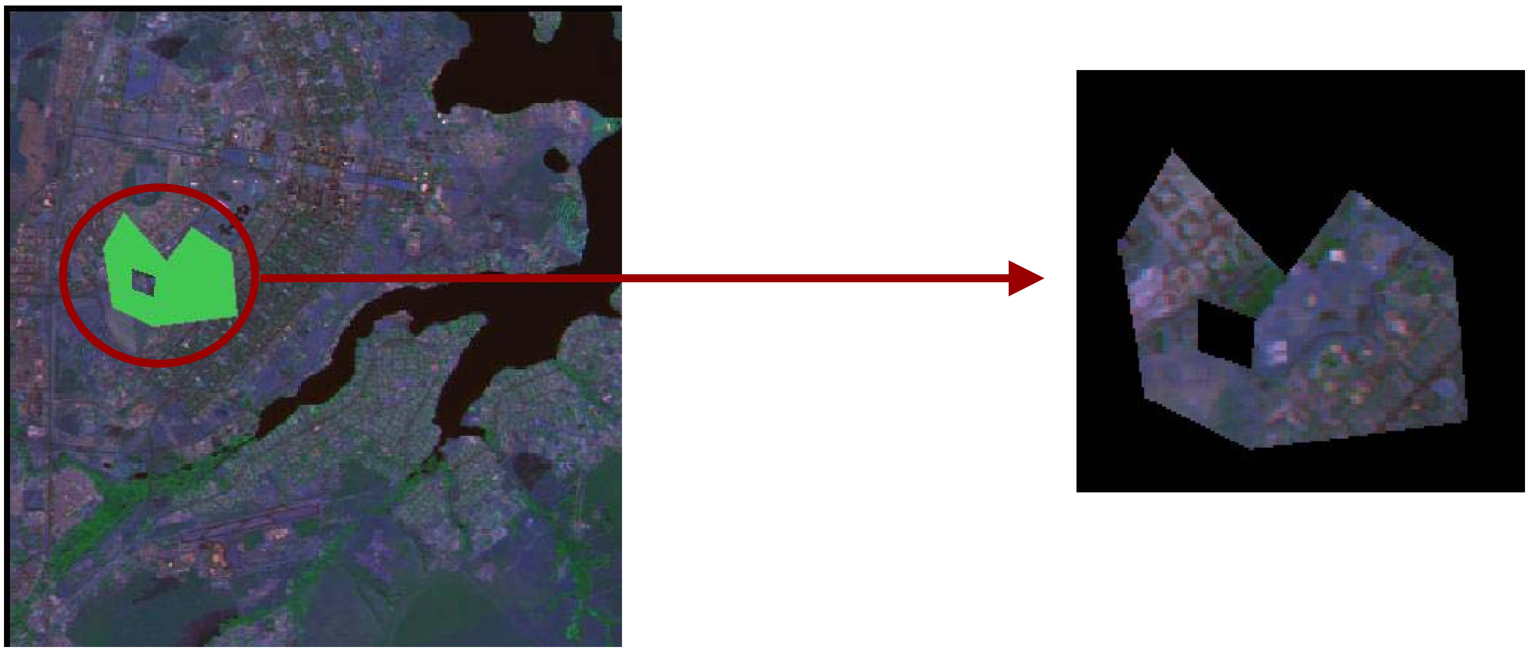
Estatísticas	Banda 0	Banda 1	Banda 2
soma	851164.000000	862173.000000	1091580.000000
valor máximo	205.000000	165.000000	206.000000
valor mínimo	30.000000	29.000000	28.000000
contagem	11365.000000	11365.000000	11365.000000
desvio padrão	18.811450	12.338327	24.338319
média	74.893445	75.862121	96.047514
variância	353.870652	152.234311	592.353748
assimetria	1.116281	1.030130	0.300146
curtose	5.929326	6.152706	3.588302
amplitude	175.000000	136.000000	178.000000
mediana	72.000000	74.000000	96.000000
coeficiennte de variação	25.117619	16.264147	25.339873
moda	70.000000	74.000000	97.000000

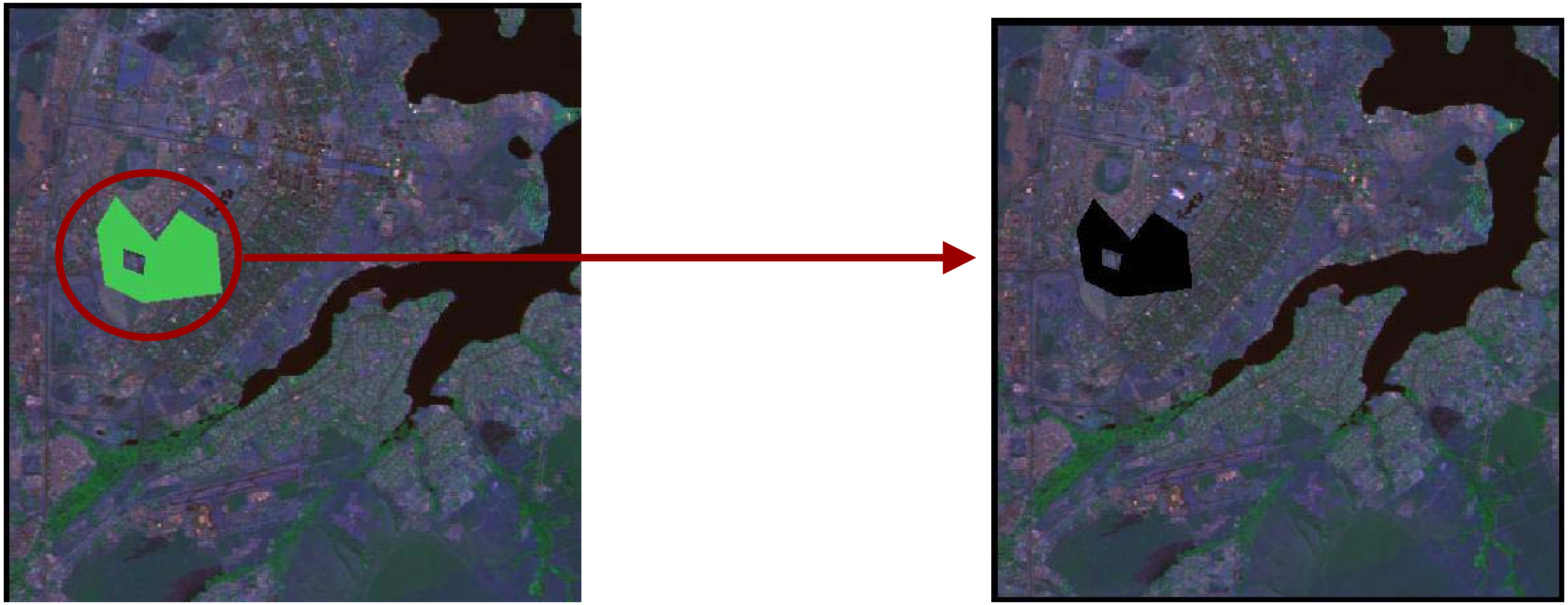
API – Raster Data

- **`Mask`** Operation
  - Clips a raster data using a mask

API – MASK Operation

– Clips a raster data using a mask





The Use of Iterators

- Mechanism to traverse a Raster Data only in a region **inside** or **outside** a specific polygon

- Developed:
  - Iterator concept on `TeRaster` structure
    - `IteratorPoly`
  - Route strategies

Algorithm Development made Easy

- *Iterator* is an abstraction of a pointer to a sequence

Algorithm

```
TeCalculateStatistics(itBegin, itEnd, stat)
```

↕

Iterator

```
TeRaster::iteratorPoly itBegin = raster->begin(poly, TeBoxPiIn)
TeRaster::iteratorPoly itEnd = raster->end(poly, TeBoxPixelIn)
```

↕

Data Structure

```
TeRaster* raster
```

Conclusions

- *Tiling* + Multi-resolution:
  - Efficient to visualization applications
- TeRaster provides an easy interface to algorithms
- TeDecoder provides flexibility to deal with different types of Raster data

Conclusions

- The developed API:
  - Provides spatial operations on a high level of abstraction for the developers of geographical application
  - Explores a new generation of object-relational DBMS that manage geographical data

Future Works

- Implement other operations on Raster Data:
  - Mathematical Operations
  - Reclassify
  - Slice
  - Weight
- Extend the API to support new spatial extensions
  - Spatial Extension in MySQL (release 4.1)
  -
- Use future resources of spatial extensions to treat Raster Data (ex. Oracle Spacial)