Geospatial API for the cloud

Martin Desruisseaux
Geomatys
martin.desruisseaux@geomatys.com

APACHECON North America

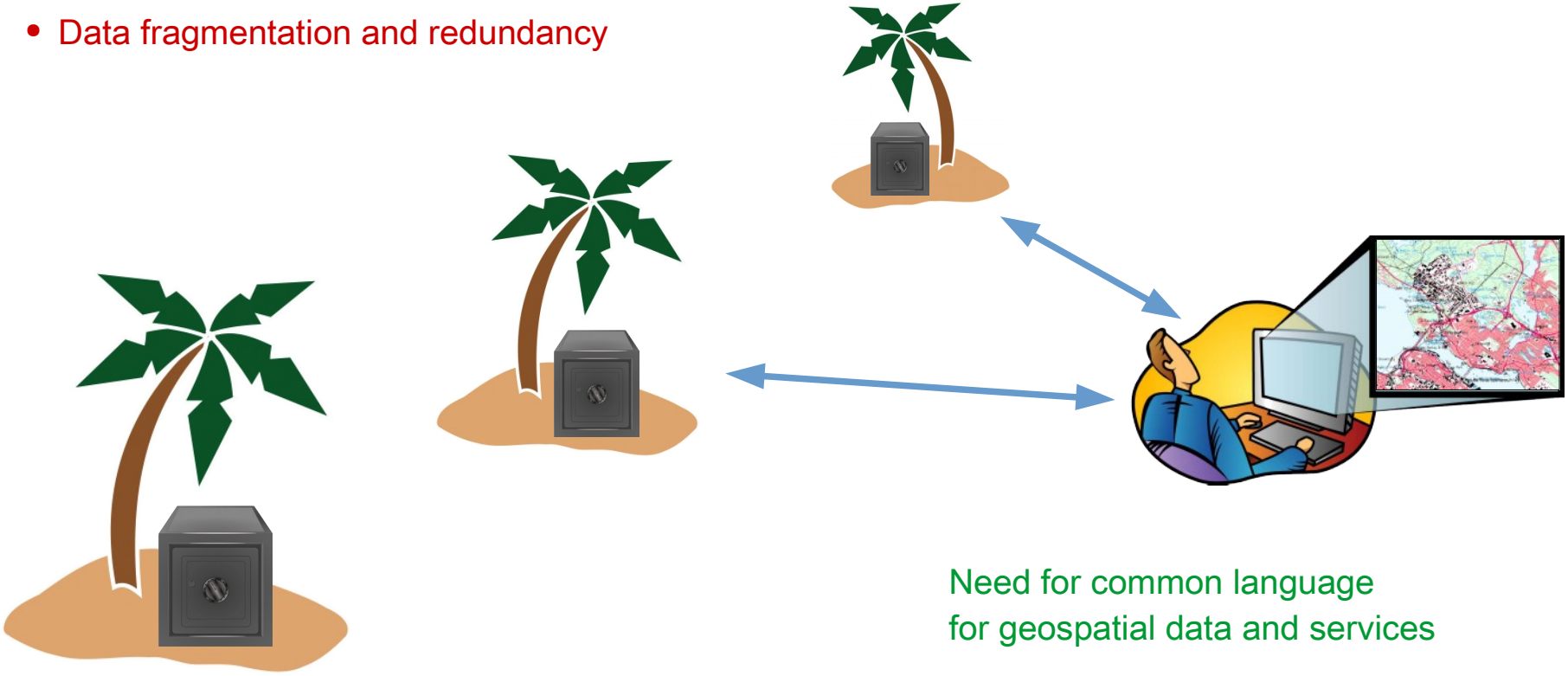
Sept. 24-27, 2018

Agenda

- Data formats and web services standards
- Need for bringing algorithms to data
- GeoAPI for implementation independence
- Apache Spatial Information System (SIS)

Barrier in data exchange

- Incompatible data and systems
- Data fragmentation and redundancy

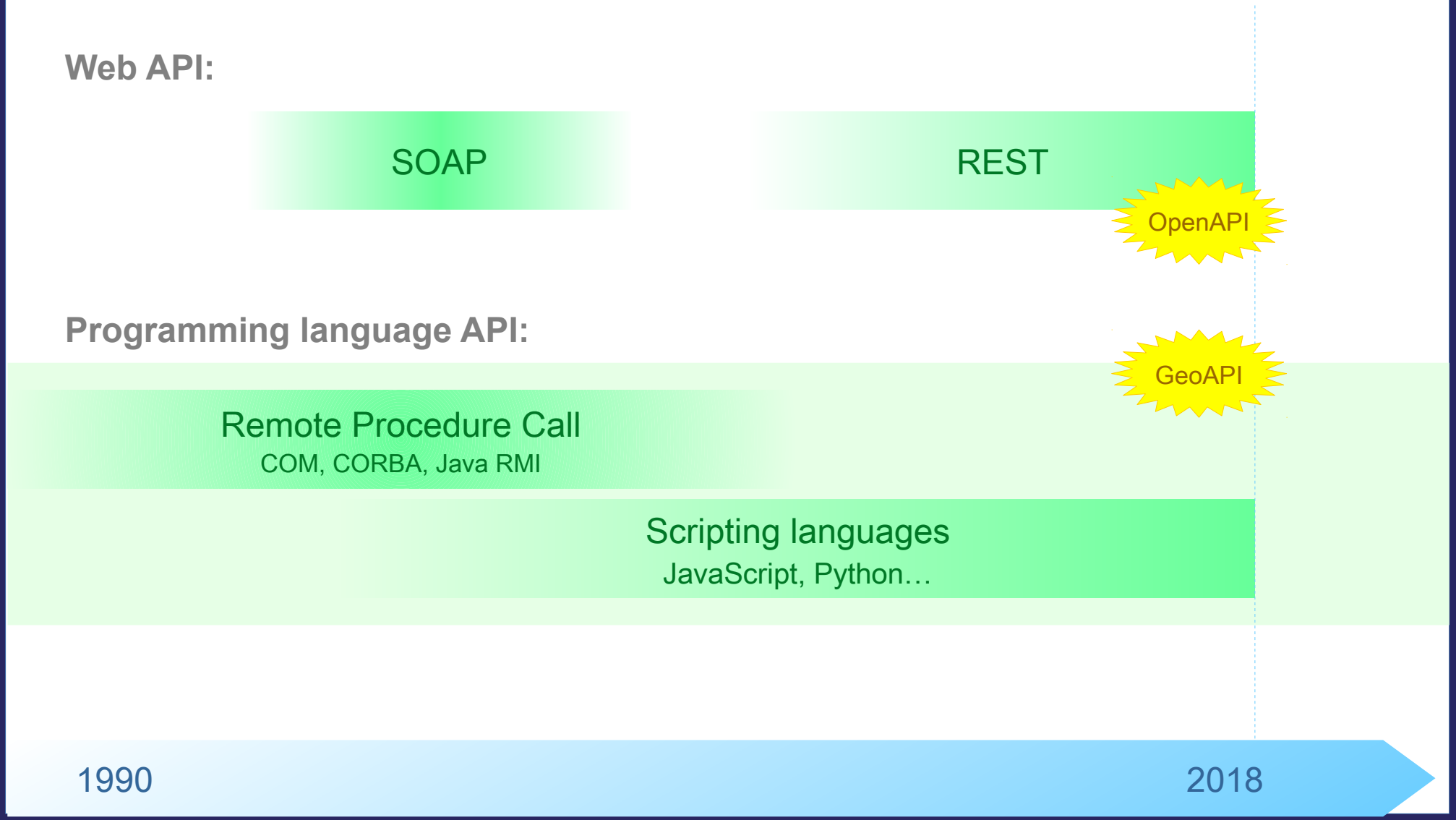


Need for common language
for geospatial data and services

An OGC standards benefit

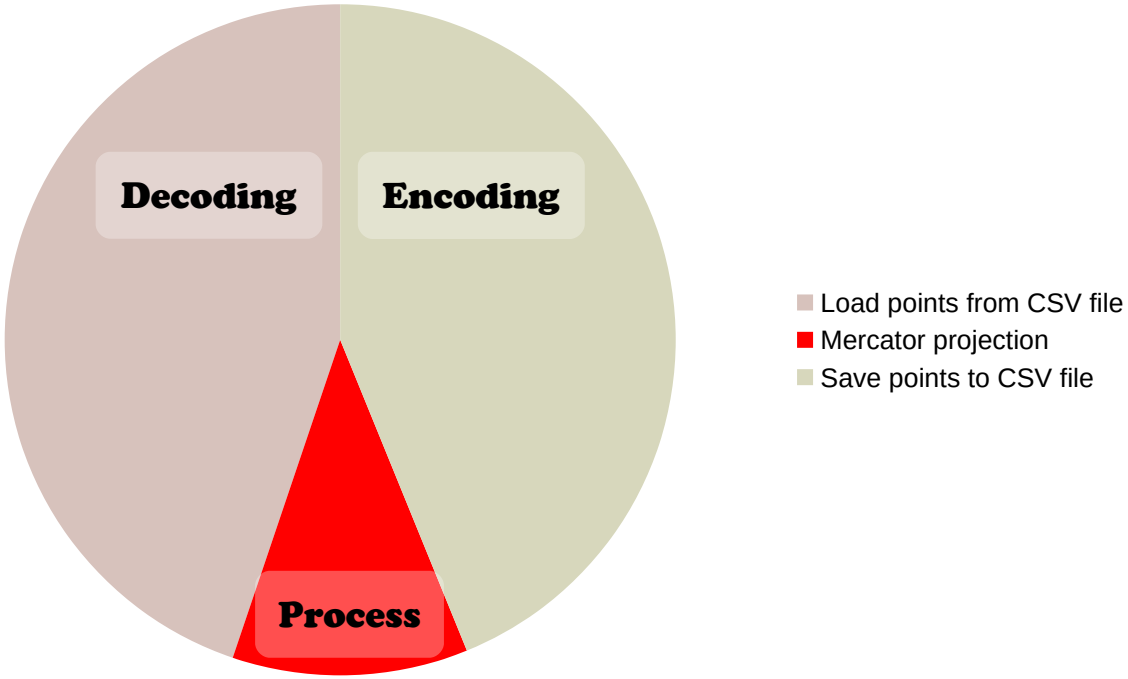

```
http://myserver/myservice?REQUEST=GetMap
    &SERVICE=WMS&VERSION=1.3.0
    &LAYERS=myLayer&FORMAT=image/png
    &CRS=EPSG:4326&BBOX=18,-161,23,-154
    &WIDTH=981&HEIGHT=826
```

API popularity



Web API implies data encoding

- May be large fraction of micro-services



- By contrast, API in programming languages often transfer only 4 or 8 bytes (a pointer)
  - More suited to fine-grain operations

Ways to process data

- Transferring data to algorithm

- Transferring algorithm to data

- WPS an an intermediate position
  - Transfer parameters for a process pre-existing on the server
  - Transfer SQL-like queries

Ways to bring algorithm to data

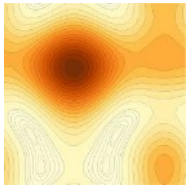
System Virtual Machine

- Include an operating system
- Costly to boot

Docker image

- More lightweight than system virtual machine
- Still relatively costly to boot
- Size is many Mb or Gb

User provides his own complete environment



Lambda function

- More lightweight than docker image
- Run in Process Virtual Machine (e.g. Java)
- Size can be a few kb

Google Earth Engine approach

Looping

Because the client doesn't know what's in server-side `ee.Thing` objects, JavaScript functionality such as conditionals and for-loops does not work with them. For that reason, and to avoid synchronous calls to `getInfo()`, use server functions to the extent possible. For example, consider the following two ways of creating a list:

Not recommended — client-side for-loop

```
var clientList = [];
for(var i = 0; i < 8; i++) {
  clientList.push(i + 1);
}
print(clientList);
```

Recommended — server-side mapping

```
var serverList = ee.List.sequence(0, 7);
serverList = serverList.map(function(n) {
  return ee.Number(n).add(1);
});
print(serverList);
```

The server-side mapping example is a little silly because you could make the same list simply with `ee.List.sequence(1, 8)`, but it illustrates some important concepts. The first concept is `map()` which simply applies the same function to everything in the list. Because this function is executed on the server, client-side functions such as `print()` won't work in a mapped function. For that reason, the `i + 1` code has to be replaced with the equivalent server-side code: `ee.Number(n).add(1)`. Importantly, `n` is an object that only exists on the server. Because the function doesn't know the type of its argument, it needs to be cast to an `ee.Number`.

OpenEO approach

http://openeo.org/openeo/news/2018/03/17/poc.html

- **Goal:** unified way to connect clients to Earth observation cloud back-ends
- High-level API (Python, Javascript, R)
- Various backends (Google Earth Engine, …)
- API not yet based on OGC standards

Another OGC standards benefit

Latitudes and longitudes without CRS are ambiguous

3 km error in some parts of the world

Credit: NASA/JPL/University of Texas

From standards to interfaces

Generated from XSD

```
CoordinateSystem cs;
if (crs instanceof GeodeticCRS) {
    GeodeticCRS geodeticCRS = (GeodeticCRS) crs;
    cs = geodeticCRS.getEllipsoidalCS();
    if (cs == null) {
        cs = geodeticCRS.getSphericalCS();
        if (cs == null) {
            cs = geodeticCRS.getCartesianCS();
        }
    }
} else if (crs instanceof VerticalCRS) {
    VerticalCRS verticalCRS = (VerticalCRS) crs;
    cs = verticalCRS.getVerticalCS();
} else if (crs instanceof EngineeringCRS) {
    EngineeringCRS engineeringCRS = (EngineeringCRS) crs;
    cs = engineeringCRS.getEllipsoidalCS();
    if (cs == null) {
        cs = engineeringCRS.getSphericalCS();
        if (cs == null) {
            cs = engineeringCRS.getCartesianCS();
            if (cs == null) {
                cs = engineeringCRS.getPolarCS();
                if (cs == null) {
                    cs = engineeringCRS.getCylindricalCS();
                }
            }
        }
    }
} else // etc.
```

From abstract model

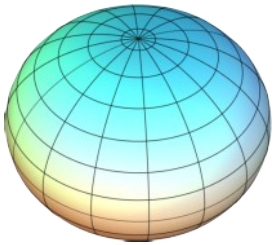
```
CoordinateSystem cs = crs.getCoordinateSystem();
```

W*S standards should not be the primary source for API in programming languages


“We just need latitude and longitude”
Simple, isn't it?

Coordinate Reference Systems

In EPSG dataset 9.5.2:

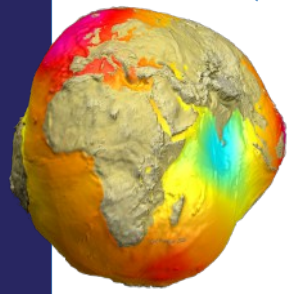


50 ellipsoids

487 two-dimensional geographic coordinate reference systems

Latitudes and longitudes without CRS are ambiguous

3 km error in some parts of the world



1 planet

514 geodetic datums

4675 projected coordinate reference systems

“I don't need all this complexity”

Let just use the WGS84 datum everywhere

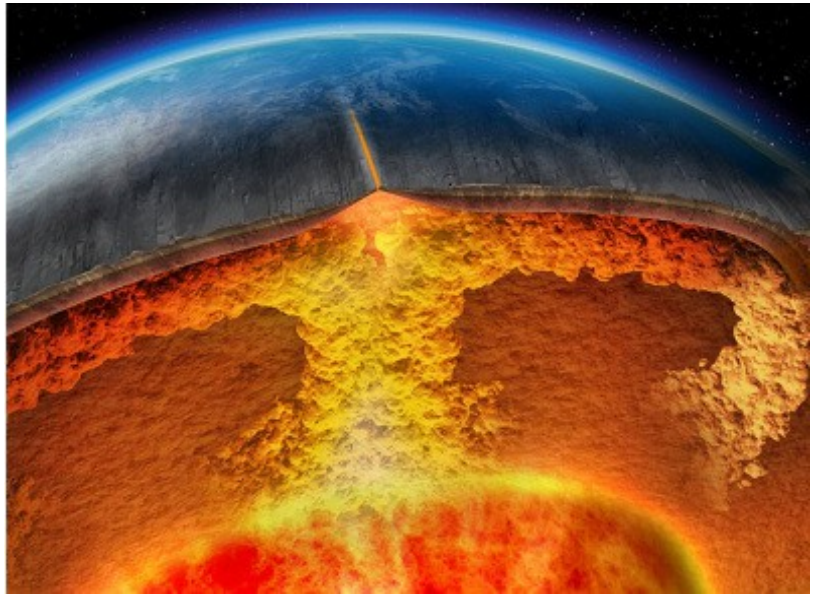
WGS84 = *World Geodetic System 1984*

This is the datum used by Global Positioning Systems (GPS)

Earth is changing

NAD83 is tied to North American Plate
WGS84 is averaged over the world

→ They do not move in the same way



30 years ago, NAD83 ≈ WGS84
Now, differ by about 1.5 metres

NAD83(86)	WGS 84(Transit)
NAD83(HPGN)	WGS 84(G730)
NAD83(CORS96)	WGS 84(G873)
NAD83(2007)	WGS 84(G1150)
NAD83(2011)	WGS 84(G1674)
NAD83(CSRS)	WGS 84(G1762)

Galileo uses a different reference frame

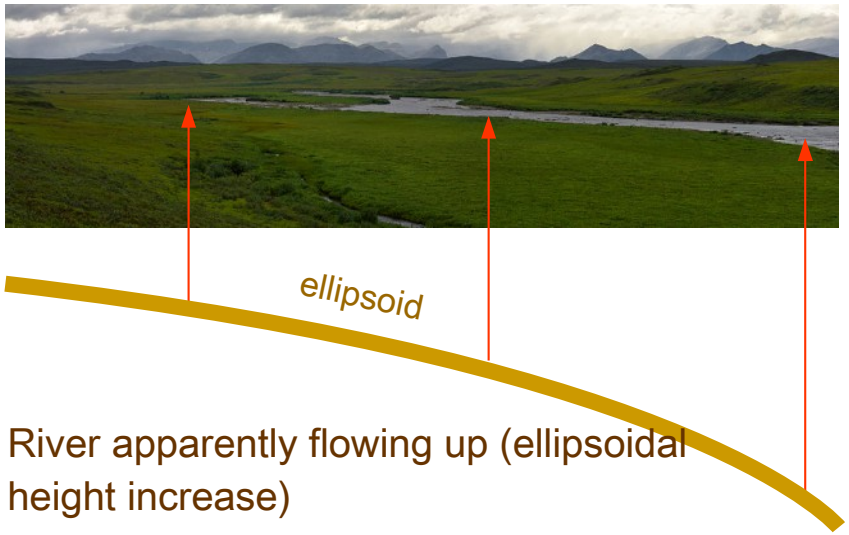
WGS84 not always suited

- Some boundaries in USA are still legally defined in NAD27

NAD27

NAD83

- Difference with geoidal height up to ±100 metres



River apparently flowing up (ellipsoidal height increase)

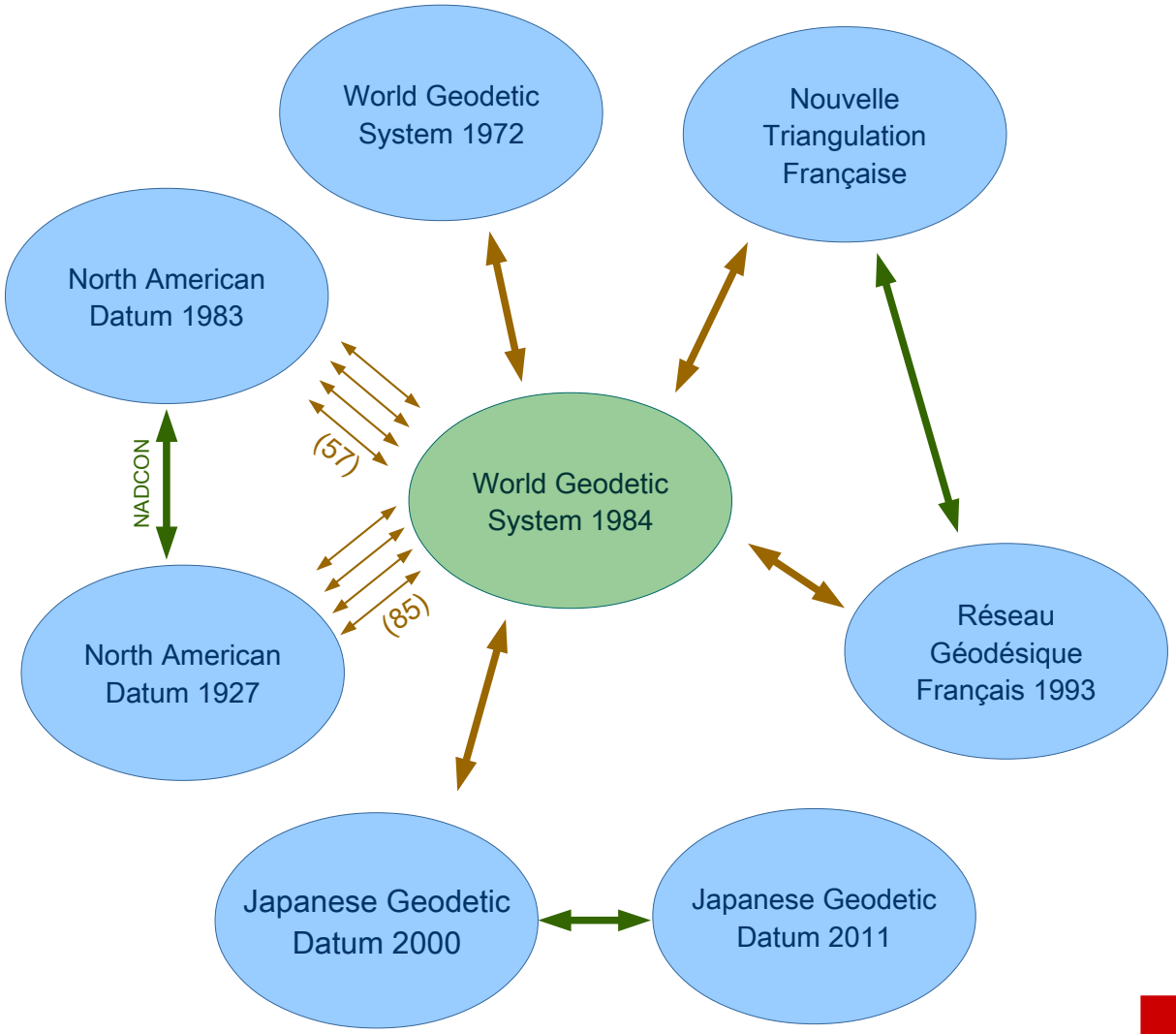
Recommendation

- Keep data in their original system and transform only when needed.
- Prefer the reference system defined by the mapping agency of the country where the data are located.

“Keep transformation engine simple”

Can we use WGS84 as the pivot system?

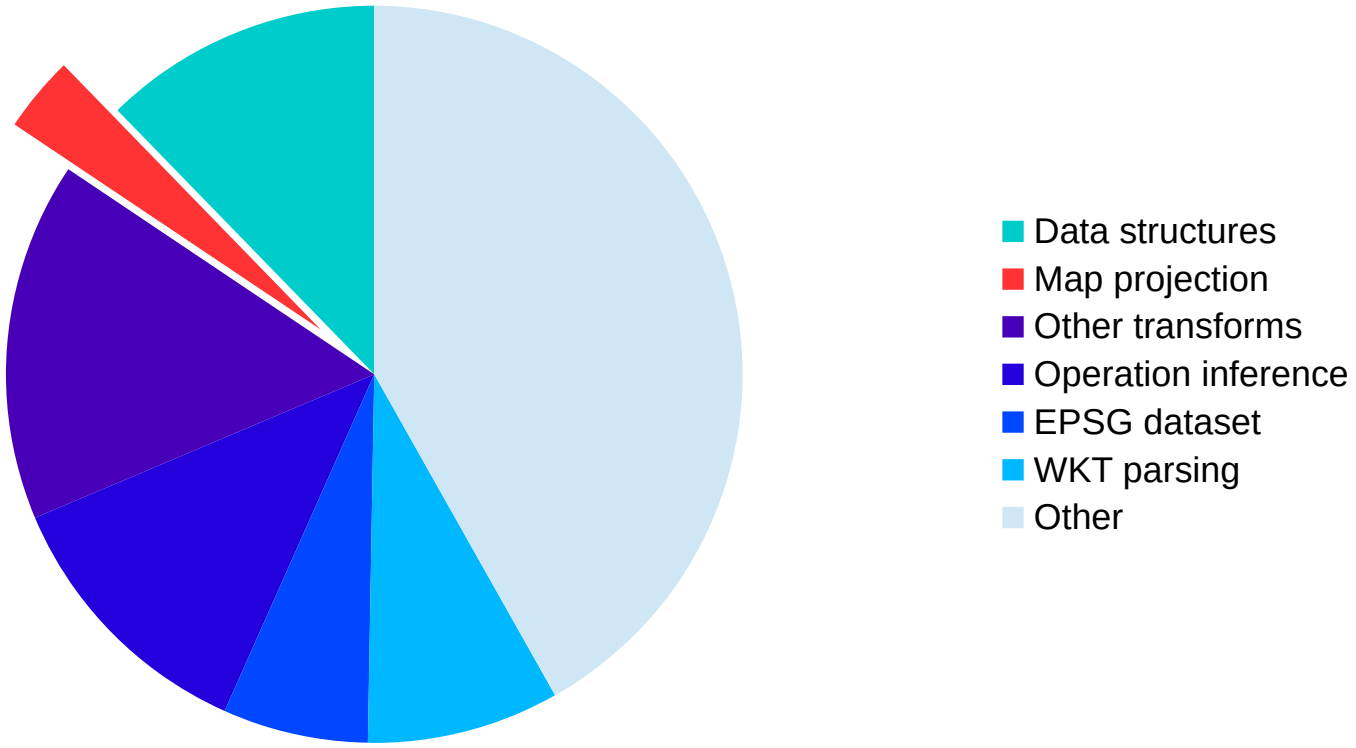
Pivot system can put errors



TOWGS84 is removed in WKT version 2

Map projections are only part of work

Proportion of map projection code in Apache SIS referencing module




“I just need a few metadata”
Coordinates and reference system, that’s all

Minimal metadata

- Popular library (GDAL) provide:
  - Title
  - Image size
  - Coordinate Reference System
  - Geographic or projected bounding box
  - Bands

ISO 19115: Geographic metadata

Metadata

Data identification

Citation
- Titles
- Authors (creator, contributor…)

Data format

Spatiotemporal extent
- Geographic bounding box
- Vertical and temporal ranges

Resolution

Content information
- Illumination elevation & azimuth angles
- Cloud cover percentage

Attribute (band) group
- Content type (physical measurement, …)

Attribute (band)
- Description (coastal aerosol, …)
- Peak response in nanometres
- Transfer function

More attributes (bands)…

ISO 19115: Geographic metadata

Metadata

Spatial representation

Acquisition

- Platform & instruments
- Operation (status, events, …)

Distribution

- Format
- Digital transfer options

Lineage

- Processors (organization, …)
- Process steps (inputs, algorithm, …)

Data quality

- Completeness
- Consistency (logical, thematic, …)

Maintenance

- Scope (dataset, software, …)
- Dates & update frequency

Constraints (legal, security, …)

GeoAPI

- Initiated in 2002
- Open Geospatial Consortium (OGC) working group
- Java interfaces derived from OGC/ISO conceptual models
- `org.opengis.*` packages
- **Versions:**
  - Latest release is GeoAPI 3.0.1 (September 2017)
  - New working group created at OGC for GeoAPI 3.1 and 4.0

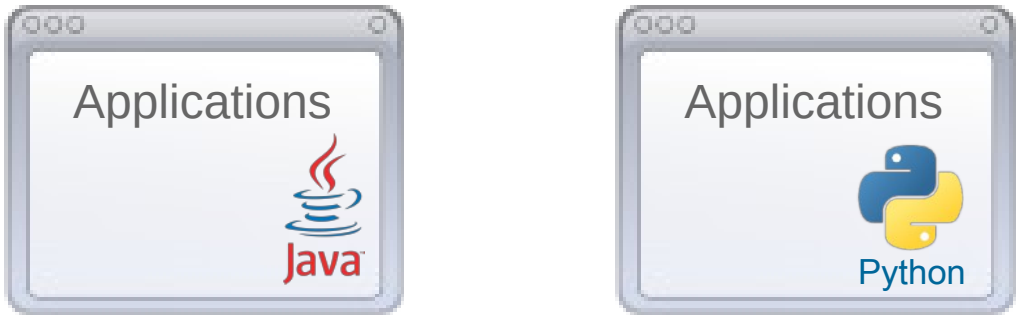
http://www.geoapi.org

Apache Spatial Information System (SIS)

- Initiated in 2010
- Top Level Apache project
- Strong focus on OGC/ISO standards (GeoAPI 3.0)
- **Versions:**
  - Latest release is Apache SIS 0.8 (November 2017)
  - Current development is Apache SIS 1.0-SNAPSHOT
- **Code:**
  - 227,000 lines of Java code
  - 262,000 lines of comments
  - Progressive transfer from Geotk project (800,000 lines) to Apache SIS

http://sis.apache.org

API layers



GeoAPI (temptative)

GDAL

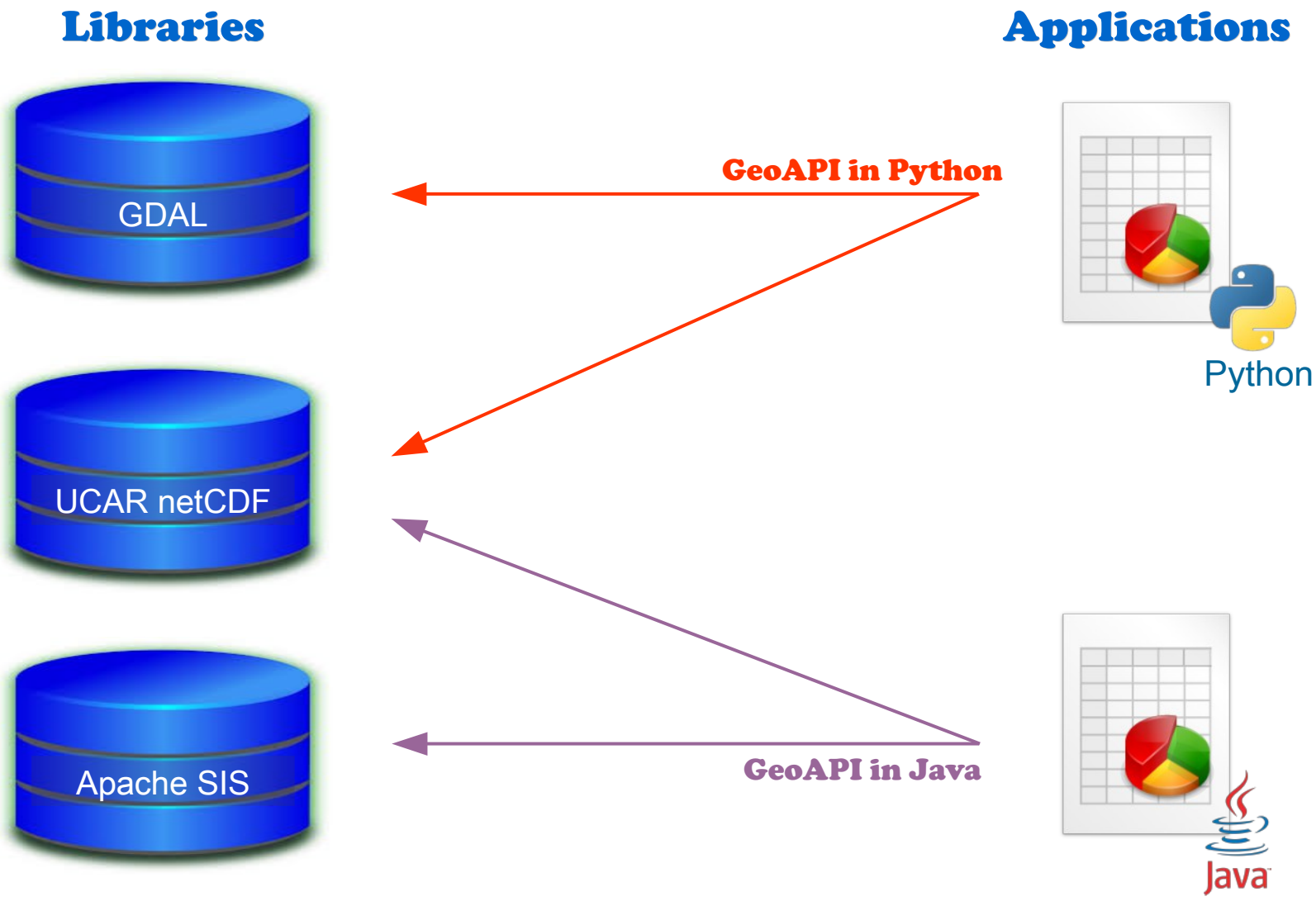
libpng | libtiff | …

Apache SIS

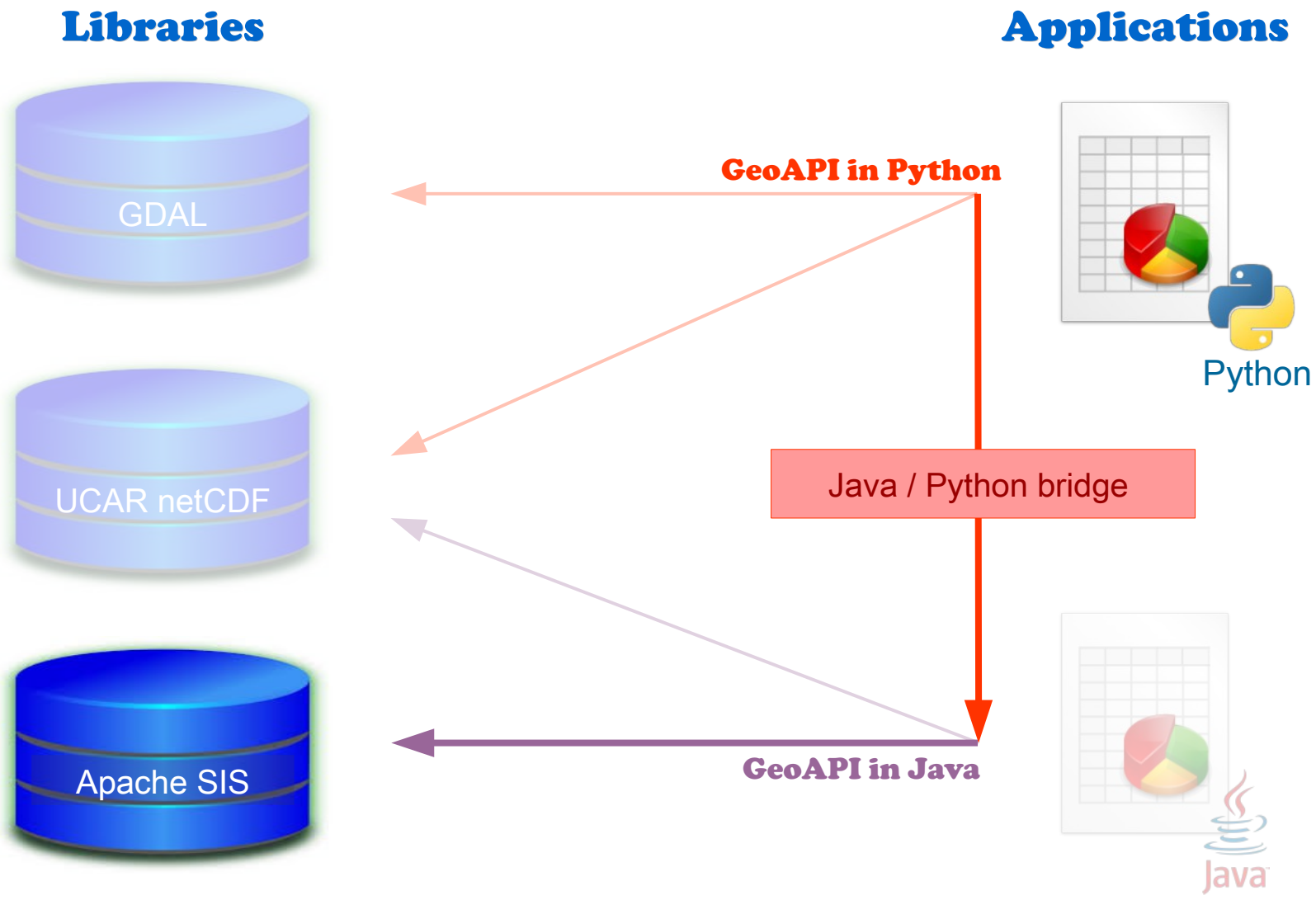
Demo

```
#from opengis.wrapper.gdal import DataSet
from apache.sis import DataSet

ds    = DataSet("myRaster.tif")
md    = ds.metadata()
axis0 = md.spatial_representation_info[0].axis_dimension_properties[0]
axis1 = md.spatial_representation_info[0].axis_dimension_properties[1]

print()
print("Resource title:      ", md.identification_info[0].citation.title)
print("Resource scope:      ", md.metadata_scope[0].resource_scope)
print("Name of first axis:  ", axis0.dimension_name)
print("Size of first axis:  ", axis0.dimension_size)
print("Name of second axis: ", axis1.dimension_name)
print("Size of second axis: ", axis1.dimension_size)
print()
print("Complete metadata as formatted by the implementation (non-normative):")
print(md)

ds.close()
```

Bridge between languages



Bridge between languages



Apache SIS usage example

- **Reference system by EPSG code**

```
CoordinateReferenceSystem myDataCRS = CRS.forCode("EPSG:3395");
```

Apache SIS uses the complete EPSG geodetic dataset

- **Reference System by Well Known Text (WKT)**

```
CoordinateReferenceSystem myDataCRS = CRS.fromWKT("PROJCRS[…]");
```

Apache SIS recognizes automatically both WKT 1 (OGC 01-009) and WKT 2

- **Reference System by Geographic Markup Language (GML)**

```
CoordinateReferenceSystem myDataCRS =
CRS.fromXML("<gml:ProjectedCRS>…");
```

Let user know about issues!

Log non-conform axis order:

```
CRS.fromWKT("GEOGCS[...definition with (lon,lat) axes..., AUTHORITY["EPSG", "4326"]]");
```

```
WARNING: The coordinate system axes in the given "WGS 84" description do not
conform to the expected axes according "EPSG:4326" authoritative description.
```

Log if prime meridian probably in wrong units (or other mismatches):

```
CRS.fromWKT("..., PRIMEM[...value in deg], ..., AUTHORITY["EPSG", "4807"]]");
```

```
WARNING: The given "NTF (Paris)" description does not conform to the "EPSG:4807"
authoritative description. Differences are found in prime meridian.
```

Log usage of deprecated EPSG code, with replacement proposal:

```
CRS.forCode("EPSG:26747");                    // NAD27 / California zone VII
```

```
WARNING: Code "EPSG:26747" is deprecated and superseded by 26799.
Reason is: Error in dependent projection record.
```

Find coordinate operation

1) Get two CRS (source and target)
2) Get a coordinate operation from source to target
3) Verify domain of validity and positional accuracy

```
import org.opengis.referencing.operation.CoordinateOperation;

// Class declaration omitted for brevety

CoordinateReferenceSystem sourceCRS = // any method shown in previous slides
CoordinateReferenceSystem targetCRS = // any method shown in previous slides
CoordinateOperation op = CRS.findOperation(sourceCRS, targetCRS, region);

// Verify domain of valididty and accuracy
System.out.println("Valid in " + CRS.getGeographicBoundingBox(op));
System.out.println("Accuracy " + CRS.getLinearAccuracy(op) + " m");
```

Apply coordinate operation

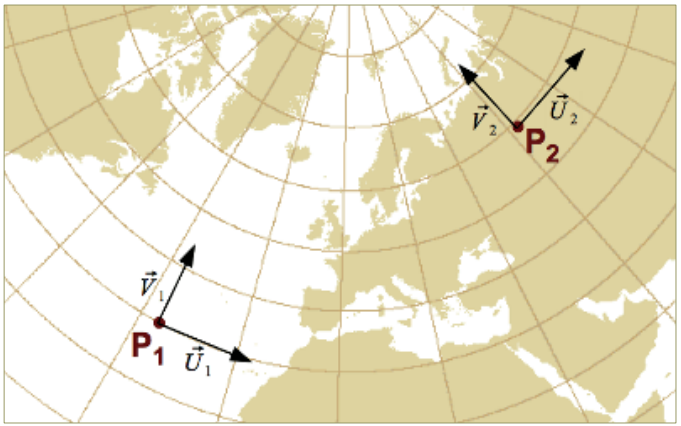
```
MathTransform mt = op.getMathTransform();
```

Example for a two-dimensional map projection:

(number of rows or columns depend on the number of dimensions)

`mt.transform(ϕ, λ)` :

$$\begin{pmatrix} x \\ y \end{pmatrix}$$



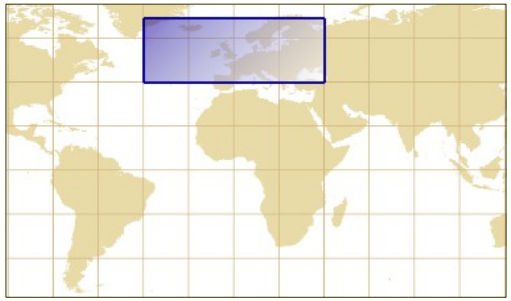
`mt.derivative(ϕ, λ)` :

$$\begin{pmatrix} \partial x/\partial \phi & \partial x/\partial \lambda \\ \partial y/\partial \phi & \partial y/\partial \lambda \end{pmatrix}$$

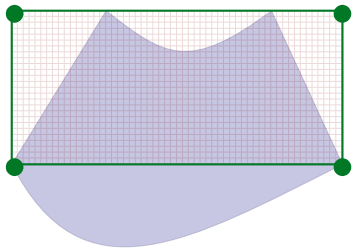
Reproject bounding boxes

```
Envelope transformed = Envelopes.transform(op, envelope);
```

Source

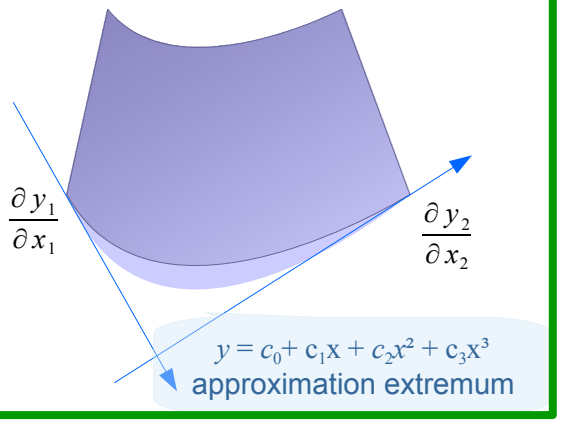


4 corners transformation

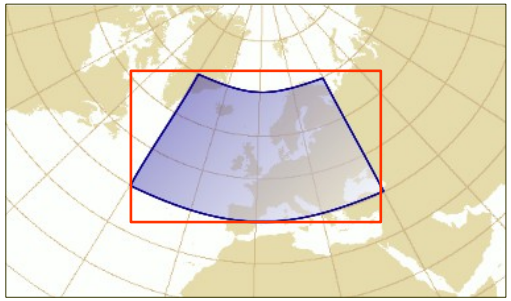


Not sufficient

Better



Target



Also have special handling for envelopes over a pole

Current state

ISO	OGC	Topic	GeoAPI	ApacheSIS
ISO 19103		Conceptual schema language	3.0.0	0.3
ISO 19115		Metadata (including imagery and gridded data extension)	3.0.0	0.3 (updated in 0.5)
ISO 19139		Metadata — XML schema		0.3, 1.0-SNAPSHOT
JSR-363		Units of Measurement API	3.0.1	0.8
ISO 19111	08-015	Spatial referencing by coordinates	3.0.0	0.4, 0.5, 0.6, 0.7, 0.8
ISO 19162	12-063	Well Known Text (WKT) representation of reference systems		0.6 (updated in 0.7)
ISO 19136	07-036	Geographic Markup Language (GML)		0.6 (updated in 0.7)
ISO 19109		Rules for application schema (Features)	3.1-SNAPSHOT	0.5
	14-084	Moving Features CSV encoding (read only)		0.7, 0.8
	16-114	Moving Features NetCDF encoding		0.8, 1.0-SNAPSHOT
ISO 19107		Feature geometry (1 to 3 dimensional)	pending	
ISO 19123	07-011	Coverage geometry and functions	pending	Pending port from the Geotk project.
ISO 19156	10-004	Observation and measurement	pending	
ISO 13249		SQL spatial		
	12-168	Catalog Services (CSW)		Google Summer of Code
ISO 19128	06-042	Web Map Service (WMS)		Pending port from the

The network is the computer

Remote Method Invocation (RMI) where introduced in Java 1.1, released in 1997.

OGC standards published in 2001 were RMI and CORBA ready.

THANK YOU

Martin Desruisseaux

martin.desruisseaux@geomatys.com