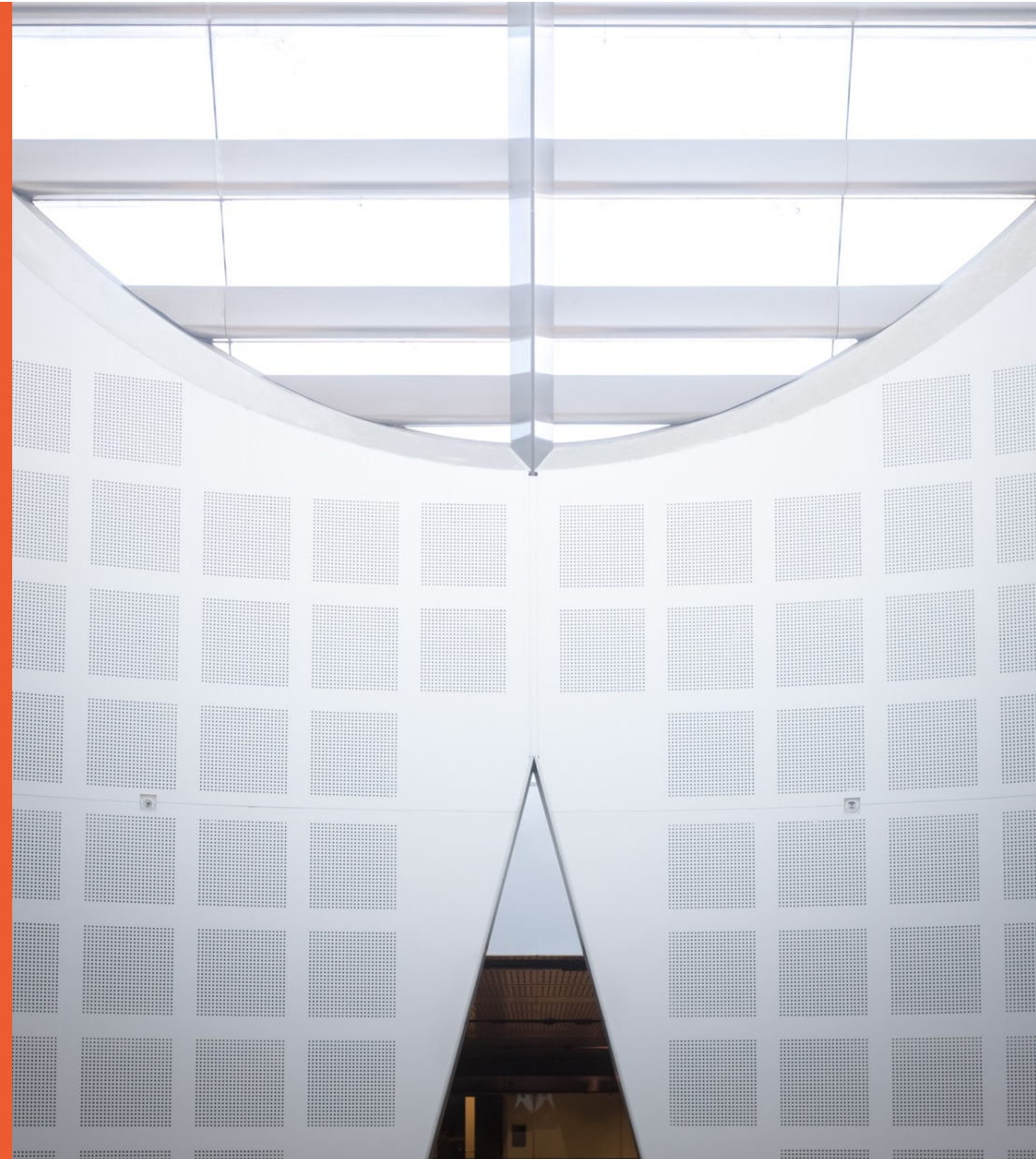


DATA2001 – Data Science, Big Data and Data Diversity Week 09: (Geo-)Spatial Data

Presented by

A/Prof Uwe Roehm

School of Computer Science



Learning Objectives

- Spatial Data Types
 - Point/Raster Data and Spatial Objects
- Querying Spatial Data with Spatial Operations
- Representing Spatial Data in Databases
 - Spatial Indexes
- GeoPandas
- Spatial Data Exchange with JSON and XML

[many slides based on S. Shekhar and S.Chawla]

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the **University of Sydney** pursuant to Part VB of the Copyright Act 1968 (the Act).

The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice

Spatial Data

Spatial Data

- Spatial data is about objects and entities which have a location and/or a geometry
- A special form is **geospatial data** which refers to data or information that identifies the geographic location of features and boundaries on Earth (such as localities, cities, suburbs etc)

Motivation

- Large amount of spatial data:
 - NASA satellites will generate one terabyte of data in one day
 - Census Data; Weather and Climate Data
 - Mobile, location-aware applications
 - Rivers, Farms, ecological impact
 - Medical Imaging
- Non-spatial queries (examples)
 - List the names of all bookstore with more than ten thousand titles.
 - List the names of ten customers, in terms of sales, in the year 2018
- Spatial Queries (examples)
 - List the names of all bookstores within ten kilometres of your current location
 - List all customers who live in Victoria and its adjoining states

Example Applications of Spatial Data

- Location-aware services
 - Location-aware mobile phone apps (eg. food finder, ride-/car-/bike-sharing apps, etc.)
 - web APIs for geo-data (eg. location APIs)
- Geographic Information Systems (GIS)
 - E.g., ESRI's ArcInfo; OpenGIS Consortium
 - Geospatial information
 - All classes of spatial queries and data are common
- Multimedia Databases
 - Images, video, text, etc. stored and retrieved by content
 - First converted to feature vector form; high dimensionality
 - Nearest-neighbor queries are the most common
- ...

Example: OpenStreetMap API

<https://nominatim.openstreetmap.org/search?city=Sydney&country=Australia&format=json>

```
[ {
  "place_id": 198993911,
  "licence": "Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright",
  "osm_type": "relation", "osm_id": 5750005,
  "boundingbox": [
    "-34.1732416",
    "-33.3641481",
    "150.260825",
    "151.3427428"
  ],
  "lat": "-33.8548157",
  "lon": "151.2164539",
  "display_name": "Sydney, New South Wales, Australia",
  "class": "place",
  "type": "city",
  "importance": 0.9623536703831199,
  "icon": "https://nominatim.openstreetmap.org/images/mapicons/poi_place_city.p.20.png"
},
{
  "place_id": 198840448,
  "licence": "Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright",
  "osm_type": "relation", "osm_id": 5729534,
  "boundingbox": [
    "-33.8797564",
    "-33.8561096",
    "151.196999",
    "151.223011"
  ],
  "lat": "-33.8679574", "lon": "151.210047",
  "display_name": "Sydney, Council of the City of Sydney, New South Wales, 2000, Australia",
  "class": "boundary",
  "type": "administrative",
  "importance": 0.688850631413584,
  "icon": "https://nominatim.openstreetmap.org/images/mapicons/poi_boundary_administrative.p.20.png"
}
```

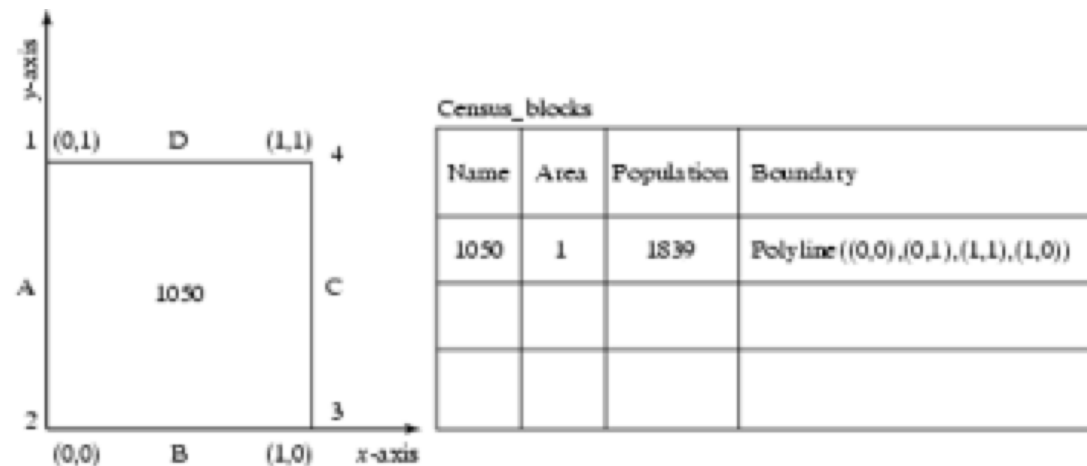
SDBMS vs GIS

- Spatial Database Management System (SDBMS)
 - Handle large amount of spatial data stored in secondary storage.
 - Spatial semantics built into query language
 - Specialized index structure to access spatial data
- Geographic Information System (GIS)
 - SDBMS Client
 - Characterized by a rich set of geographic analysis functions
 - SDBMS allows GIS to scale to large databases, which are now becoming the norm
 - Information in a GIS is typically organized in “layers”.
 - For example a map will have a layer of “roads”, “train stations”, “suburbs” and “water bodies”.
 - GIS allows data exploration and integration across layers.

Example: Census Block Database

```
CREATE TABLE census_blocks (  
    name          string,  
    area          float,  
    population    number,  
    boundary      polyline );
```

- Note: Polyline is not a built-in datatype in a non-spatial databases



Example (cont'd): Mapping Spatial Data into Relations (if no spatial data type available)

Census_blocks				Polygon	
Name	Area	Population	boundary-ID	boundary-ID	edge-name
340	1	1839	1050	1050	A
				1050	B
				1050	C
				1050	D

Edge		Point		
edge-name	endpoint	endpoint	x-coor	y-coor
A	1	1	0	1
A	2	2	0	0
B	2	3	1	0
B	3	4	1	1
C	3			
C	4			
D	4			
D	1			

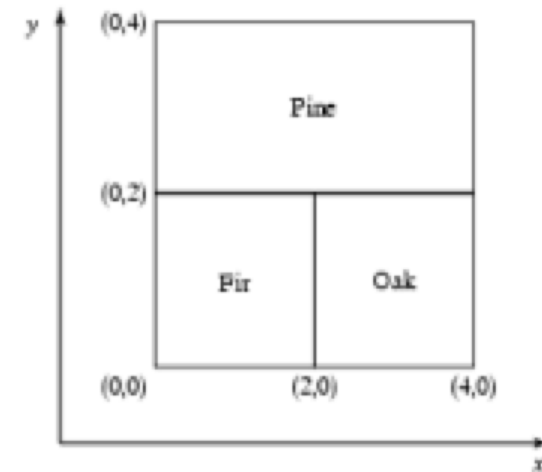
Possible, but not ideal: Even simple queries now involve many joins.
Hence: SDBMS features very important when dealing with spatial data.

Types of Spatial Data

- Point Data
 - Points in a multidimensional space
 - E.g. geo-location of some data entities from location-aware apps
 - As time-series: **trajectory data** of moving objects (such as cars)
 - E.g., **raster data** such as satellite imagery, where each pixel stores a measured value
 - E.g., Feature vectors extracted from text
- Region Data
 - Objects have spatial extent with location and boundary
 - DB typically uses geometric approximations constructed using line segments, polygons, etc., called **vector data**.

Models of Spatial Information

- Two common models
 - Field based (*also: space-based*)
 - Object based (*also: entity-based*)
- Example: Forest stands
 - (a) forest stand map
 - (b) Object view has 3 polygons
 - (c) Field view has a function
- We will mainly use object-based models in DATA2001



(a)

Area-ID	Dominant Tree Species	Area/Boundary
F51	Pine	[(0,2),(4,2),(4,4),(0,4)]
F52	Fir	[(0,0),(2,0),(2,2),(0,2)]
F53	Oak	[(2,0),(4,0),(4,2),(2,2)]

(b)

$$f(x,y) = \begin{cases} \text{"Pine,"} & 2 \leq x \leq 4; 2 < y \leq 4 \\ \text{"Fir,"} & 0 \leq x \leq 2; 0 \leq y \leq 2 \\ \text{"Oak,"} & 2 < x \leq 4; 0 \leq y \leq 2 \end{cases}$$

(c)

Object Model

- Object model concepts
 - **Objects:** distinct identifiable things relevant to an application
 - Objects have attributes and operations
 - **Attribute:** a simple (e.g. numeric, string) property of an object
 - **Operations:** function maps object attributes to other objects
- Example from a roadmap
 - Objects: roads, landmarks, ...
 - Attributes of road objects:
 - spatial: location, e.g. polygon boundary of land-parcel
 - non-spatial: name (e.g. Route 66), type (e.g. motorway, residential street), number of lanes, speed limit, ...
 - Operations on road objects: determine center line, determine length, determine intersection with other roads, ...

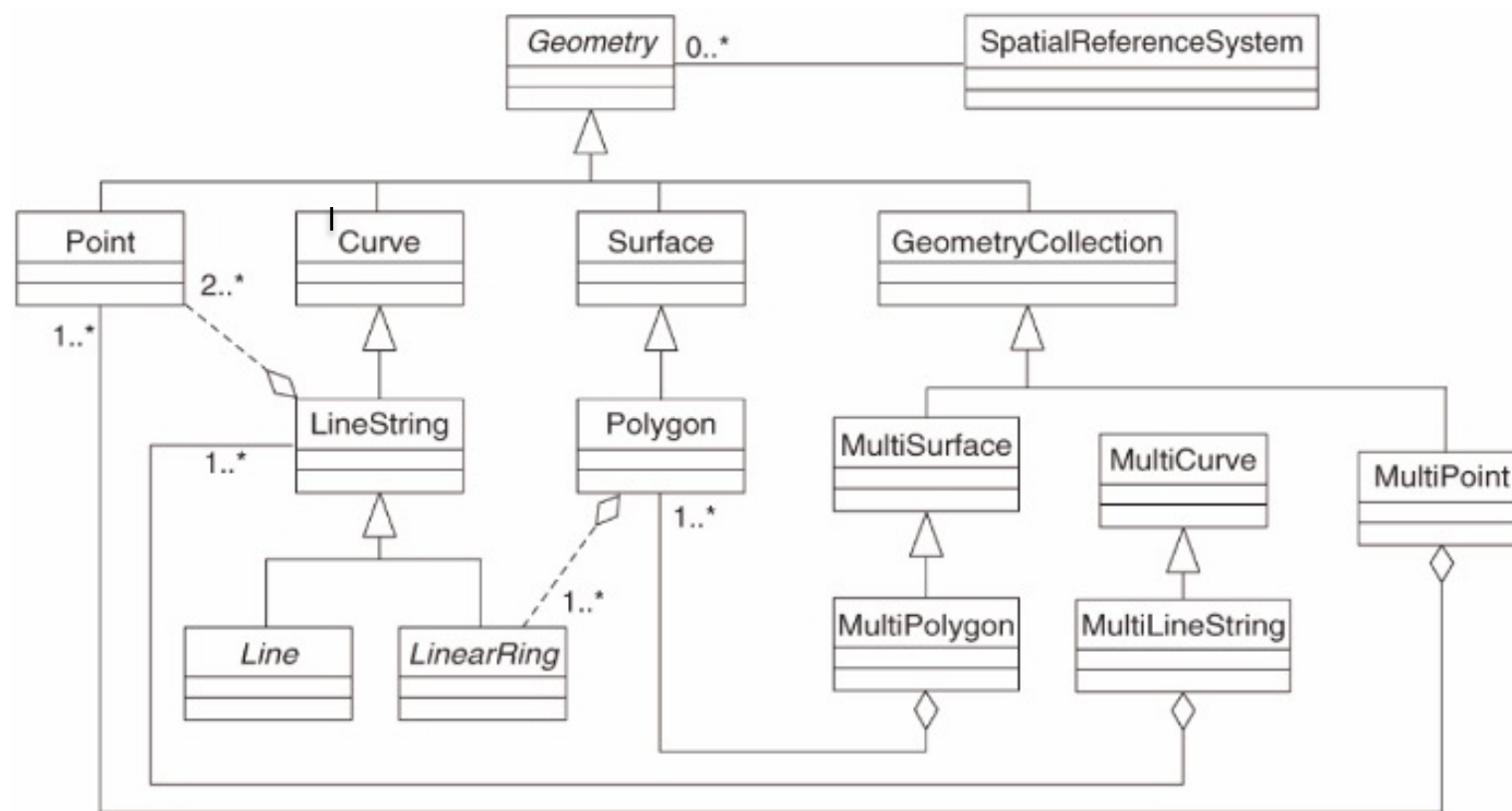
Classifying Spatial Objects

- Spatial objects are spatial attributes of general objects
- Spatial objects are of many types
 - Simple
 - 0- dimensional (points), 1 dimensional (curves), 2 dimensional (surfaces)

Spatial Object Types	Example Object	Dimension
Point	City	0
Curve	River	1
Surface	Country	2

- Collections
 - Polygon collection (e.g. boundary of Japan or Hawaii), ...

OpenGIS Consortium (OGC) Data Model

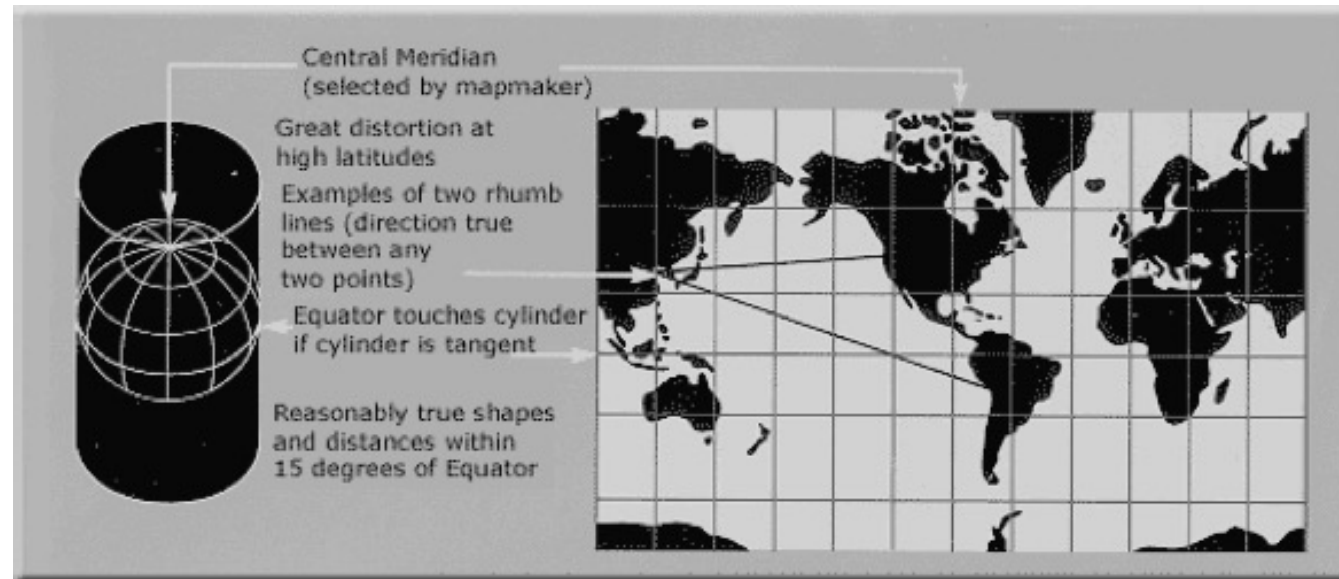


[Source: OGC Simple Features, 2016]

Coordinate Systems

- Each instance of a spatial type has a *spatial reference identifier* (SRID)
 - The SRID is an integer that identifies the coordinate system in which the type is described.
 - Two instances with different SRIDs are incompatible.
 - Different SRID means different results for some operations, such as area.
- List typically according to prevalent standards (EPSG/OGP).
 - Currently about 390 distinct coordinate systems.
- Types
 - **Cartesian Coordinates**
 - point positions from a defined origin along axes on a plane
 - **Geodetic or Geocentric Coordinates (Geographic Coordinates)**
 - angular coordinates (longitude and latitude)
 - **Projected Coordinates**
 - are planar Cartesian coordinates that result from performing a mathematical mapping from a point on the Earth surface to a plane

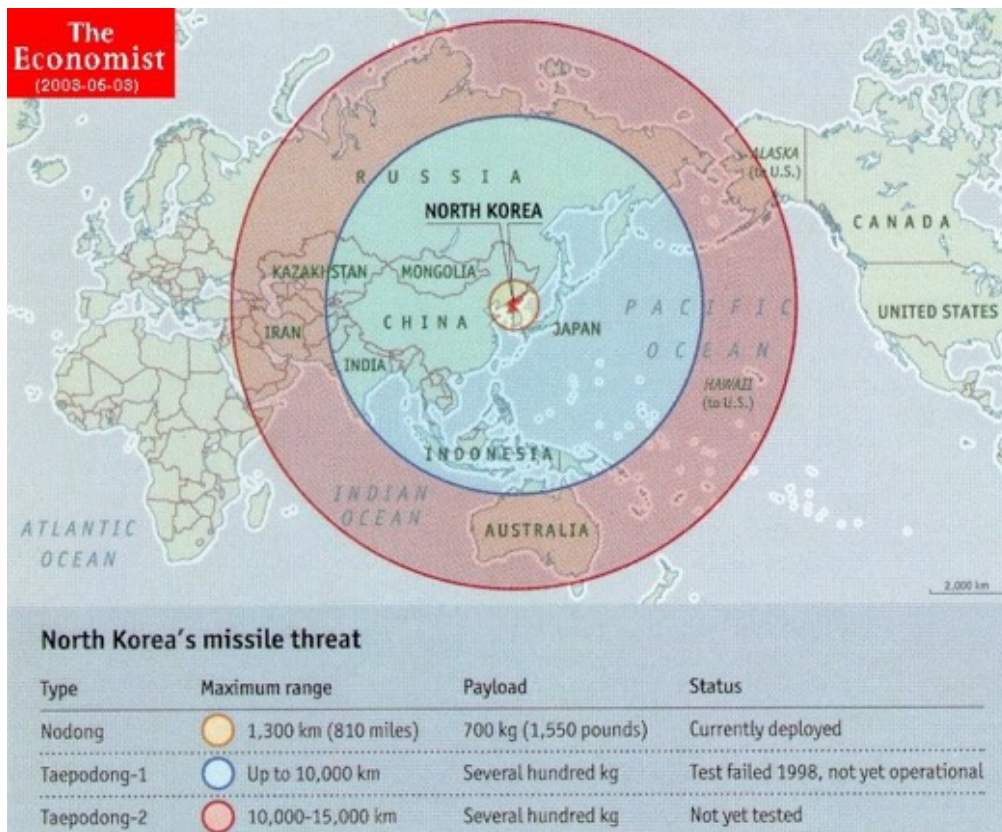
Mercator Projection



- ▶ Projections are trying to fit something round, like the earth, to something flat, like a map.
- ▶ This process introduces distortion
 - ▶ For *mercator projection*, most of the distortion happens at high latitude relative to the equator
 - ▶ E.g., in reality Greenland is 14 times smaller than Africa

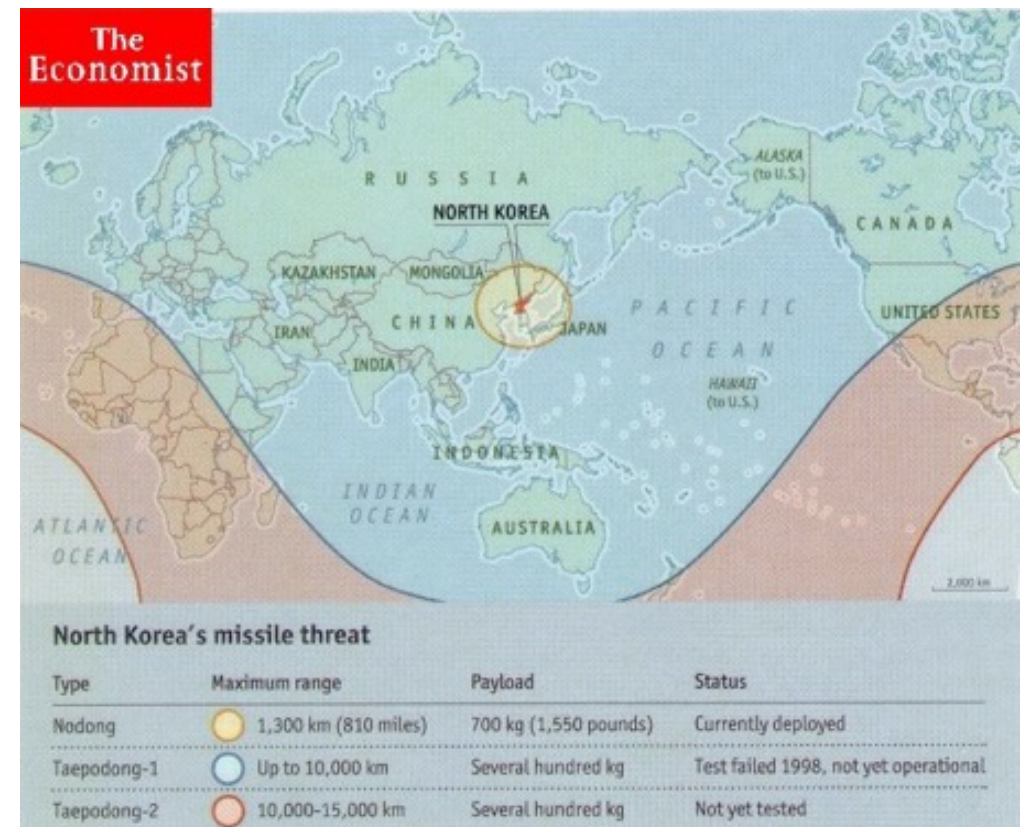
Example: Coordinate Systems & Projections Do Matter

Planar Computation Model



► Published on May 3rd 2003 - **wrong!**

Correct Spherical Computation Model



► Corrected on May 15th, 2003

WGS84 versus Australian GDA94

- WGS84 is used by the official GPS system
- The official geodetic datum (coordinate system) for Australia is GDA94 ("Geocentric Datum of Australia")
 - Based on IERS Terrestrial Reference Frame (ITRF), but fixed to a number of reference points in Australia.
- Difference between WGS84 and GDA94:
 - "The spheroids used for WGS84 and GDA84 are also almost identical, and both systems are geocentric. Thus for most mapping, exploration and GIS uses, WGS84 and GDA94 coordinates will be the same. [...] For precise surveys, however, the difference between WGS84 and GDA94 may be significant, and changes slowly over time. [...] The difference between GDA94 and WGS84 is approximately 45cms in 2000."

[<http://www.geoproject.com.au/gda.faq.html>]

Database Support for (Geo-)Spatial Data: PostGIS

Support for Spatial Data in PostgreSQL

- PostgreSQL supports **spatial data** out of the box
- Geometric Types
 - Point, line, box, path, polygon, circle
 - <https://www.postgresql.org/docs/current/static/datatype-geometric.html>
- Geometric Functions and Operators
 - Such as intersect, contains, overlaps, distance etc
 - <https://www.postgresql.org/docs/current/static/functions-geometry.html>
- Indexing of Geometric Types
 - GiST
 - <https://www.postgresql.org/docs/current/static/gist.html>
`CREATE INDEX census_idx ON CensusData USING GIST(geom);`
- For **geographic data** : PostGIS extension (open source – <https://postgis.net>)

PostGIS

[<http://postgis.net/documentation/>]

- Spatial database extension for PostgreSQL supporting geographic objects (OGC)
 - **Geometry types** for Points, LineStrings, Polygons, MultiPoints, etc.
 - including import/export from standard formats such as GeoJSON or KML
 - **Geography type** for geodetic data
 - Support for **spatial reference systems** and transformations between
 - **Spatial predicates** on geometries using the nine-intersection model
 - Spatial operators for determining **geospatial measurements** like area, distance, length and perimeter, and **geospatial set operations**, like union, difference etc.
 - **R-Tree indexing** (over GiST)
- Example:

```
INSERT INTO superhero VALUES ( 'Catwoman', ST_SetSRID(ST_MakePoint(41.87,-87.634), 4326));  
SELECT superhero.name  
FROM city, superhero  
WHERE ST_Contains(city.geom, superhero.location)  
AND city.name = 'Gotham';
```

using **WGS84**



PostGIS: Geometry vs. Geography Type

- **Geometry** type:
 - shapes on a *plane*; shortest path between two points is a straight line
- **Geography** type
 - Basis is a *sphere*; shortest path between two points is a circle arc
- This difference affects all calculations of geometries/geographies (areas, distances, lengths, intersections, ...)
 - Because of more complex computations, less operations defined in PostGIS on **geographies** than on geometries
 - ST_Intersects(), ST_Area(), ST_Distance(), ST_Perimeter(), ST_CoveredBy()
 - those functions, that are defined, take more CPU time to execute than for geometries
 - Large-scale geodetic data (spanning countries or even continents) ideally represented and compared as geographies, in smaller scale geometry is fine
 - If needed, a geometry type can be converted on demand using `::geometry`
- **See also:** https://postgis.net/docs/using_postgis_dbmanagement.html#PostGIS_Geography

Querying Spatial Data

Classifying Operations on Spatial Objects in the Object Model

- Classifying operations
 - **Set based:** 2-dimensional spatial objects (e.g. polygons) are sets of points
 - a set operation (e.g. intersection) of 2 polygons produce another polygon
 - **Topological operations:** Boundary of USA touches boundary of Canada
 - **Directional:** Brisbane is to north of Sydney
 - **Metric:** Brisbane is about 730 km from Sydney (by plane).

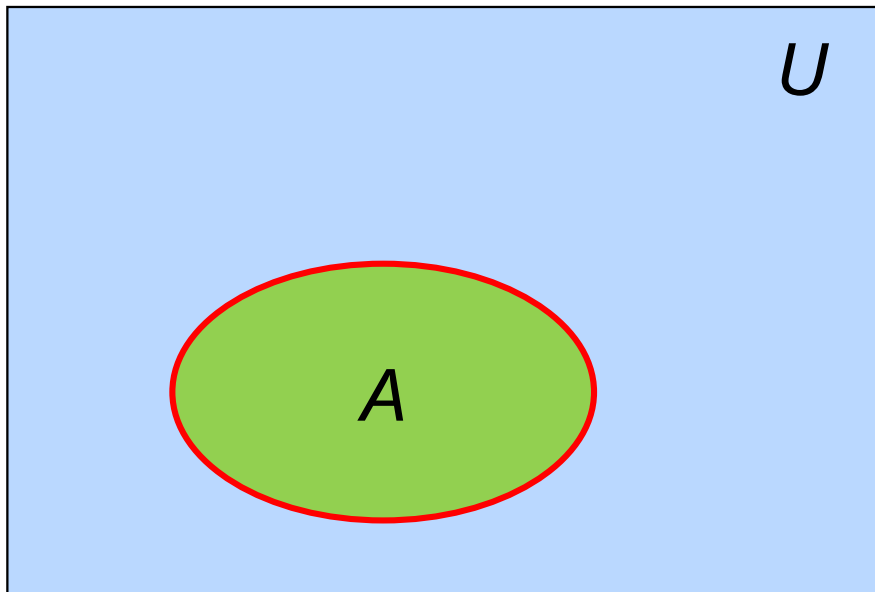
Set theory based	Union, Intersection, Containment
Topological	Touches, Disjoint, Overlap, etc.
Directional	East,North-West, etc.
Metric	Distance

Topological Relationships

- Topological Relationships
 - invariant under elastic deformation (without tear, merge).
 - Two countries which touch each other in a planar paper map will continue to do so in spherical globe maps.
- Topology is the study of topological relationships
- Example queries with topological operations
 - What is the topological relationship between two objects A and B ?
 - Find all objects which have a given topological relationship to object A ?

Topological Concepts

- Interior, boundary, exterior
 - Let A be an object in a “Universe” U .



Green is A interior (A°)

Red is boundary of A (∂A)

Blue – (**Green** + **Red**) (A^c)
is A exterior

- exterior also referred to as the *complement* of an object

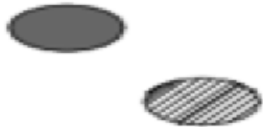







Nine-Intersection Model of Topological Relationships

- Many topological relationships between A and B can be specified using the **9-Intersection Model**
 - Eight possible 2D topological relationships for objects without holes; Examples on next slide
- Nine intersections
 - intersections between interior, boundary, exterior of A, B
 - A and B are spatial objects in a two dimensional plane.
 - Can be arranged as a 3 by 3 matrix
 - Matrix element take a value of 0 (false) or 1 (true).

$$\Gamma_9(A, B) = \begin{pmatrix} A^\circ \cap B^\circ & A^\circ \cap \partial B & A^\circ \cap B^- \\ \partial A \cap B^\circ & \partial A \cap \partial B & \partial A \cap B^- \\ A^- \cap B^\circ & A^- \cap \partial B & A^- \cap B^- \end{pmatrix}$$

Specifying Topological Operations using the 9-Intersection Model

Fig 2.3: 9 intersection matrices for the eight possible topological operations on 2D objects without holes

			
$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ disjoint	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ contains	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ inside	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ equal
			
$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ meet	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ covers	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ coveredBy	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ overlap

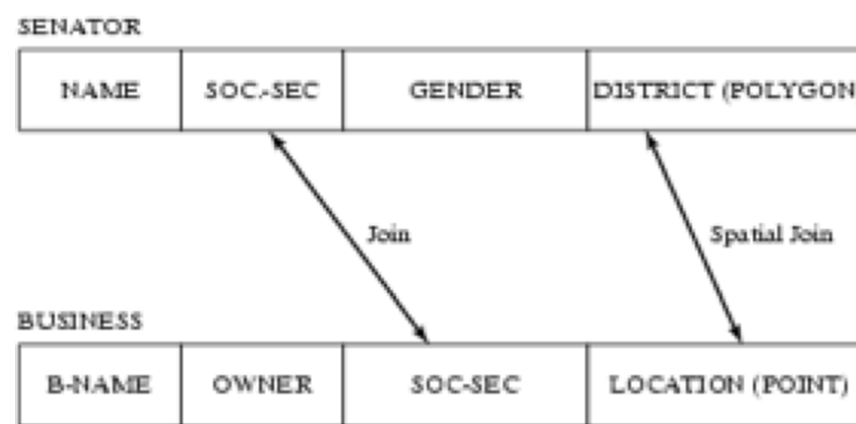
0 corresponds to $= \emptyset$, and 1 to $\neq \emptyset$

Types of Spatial Queries

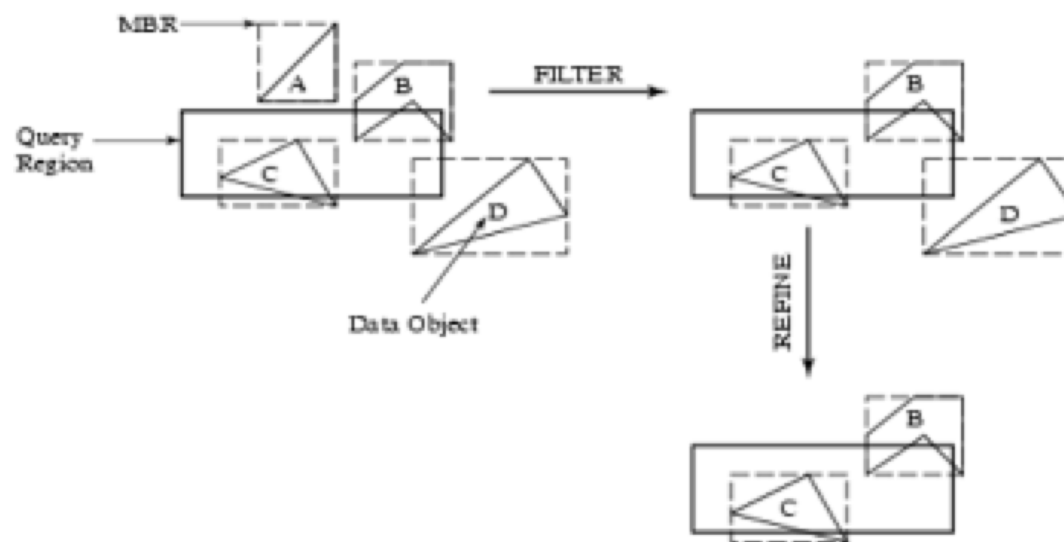
- Spatial Range Queries
 - *Find all cities within 100 kilometers of Sydney*
 - Query has associated region (location, boundary)
 - Answer includes overlapping or contained data regions
- Nearest-Neighbor Queries
 - *Find the 3 cities nearest to Sydney*
 - Results must be ordered by proximity
- Spatial Join Queries
 - *Find all cities near an ocean*
 - Expensive, join condition involves regions and proximity

Spatial Query Processing: Spatial Joins

- Non-spatial Join and Spatial Join



Spatial Query Processing: Filter-Refine Strategy

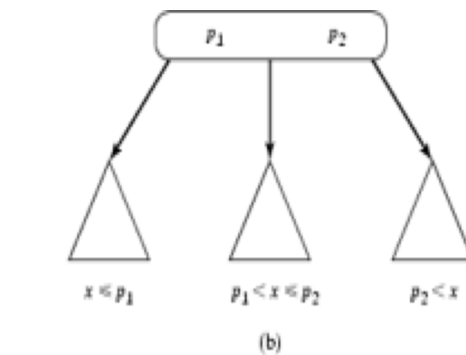


Because precise spatial query processing is complex, typically a two step '**filter-refine**' strategy is used for querying spatial data

- Filter Step: Objects with *minimum bounding box* (MBR) intersecting query regions
- Refine Step: Query region really intersecting only with B and C

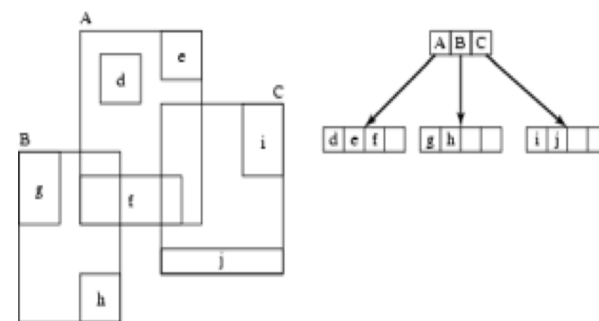
Indexing of Spatial Data

For 1-dimensional point data:



Classical B+ tree

For n-dimensional spatial data:

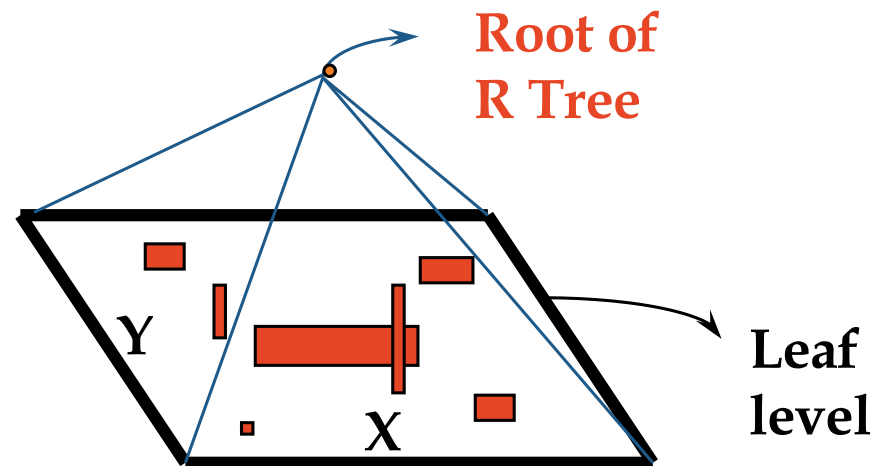


Spatial R+ tree

(PostgreSQL: GiST)

The R-Tree

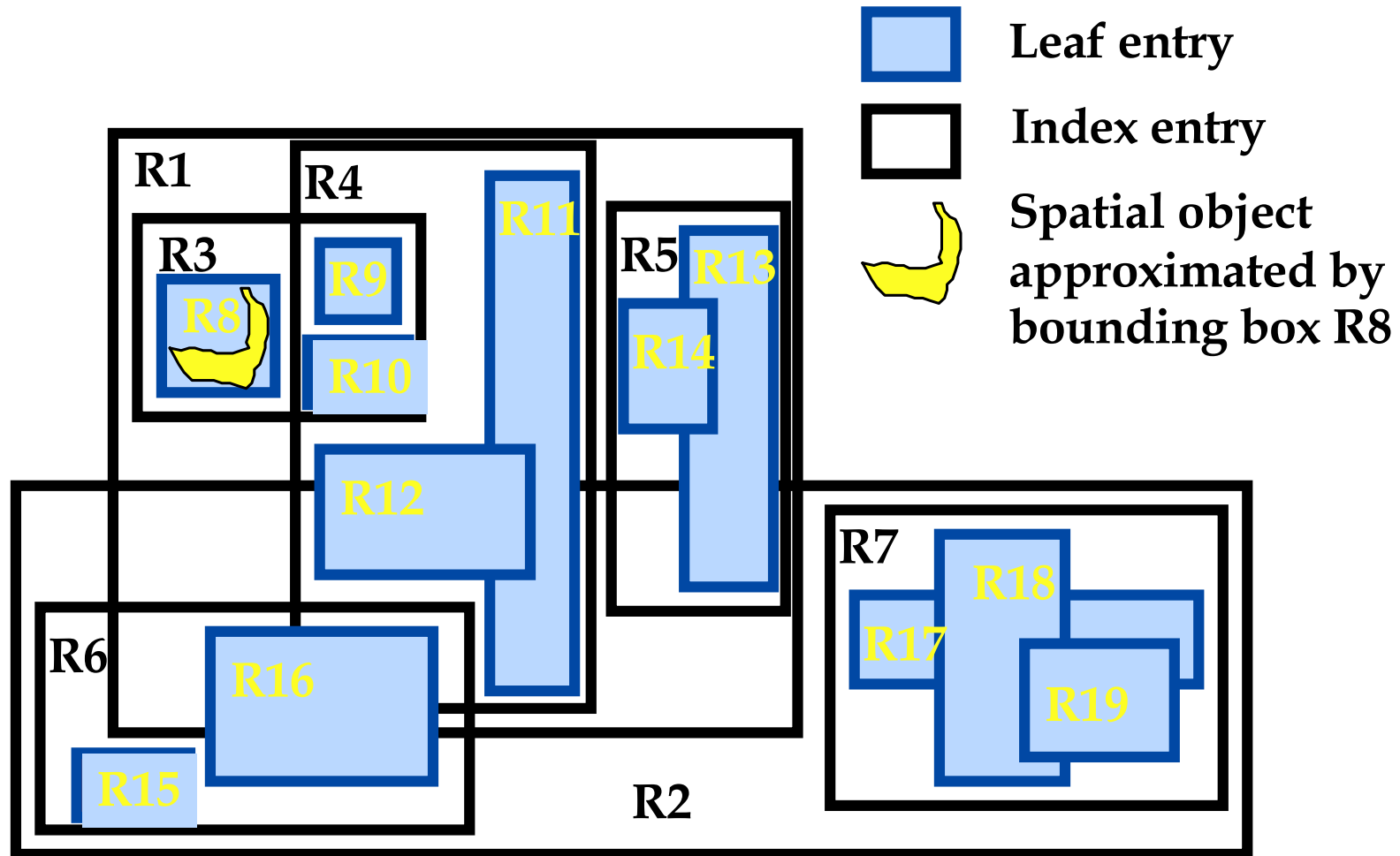
- The R-tree is a tree-structured index that remains balanced on inserts and deletes.
- Each key stored in a leaf entry is intuitively a box, or collection of intervals, with one interval per dimension.
- Example in 2-D:



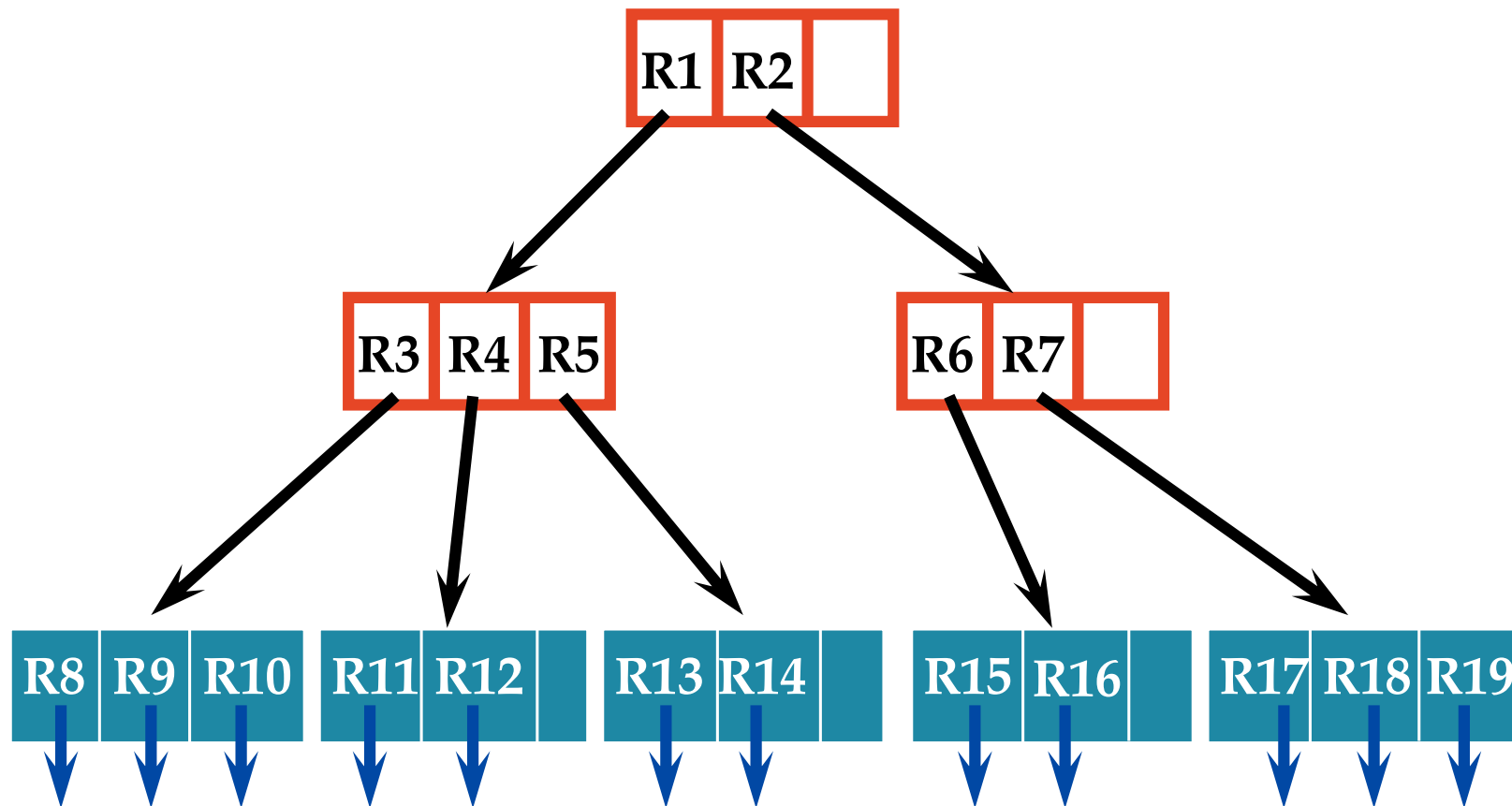
R-Tree Properties

- Leaf entry = $\langle \text{n-dimensional box, rid} \rangle$
 - This is Alternative (2), with *key value* being a box.
 - Box is the tightest bounding box for a data object.
- Non-leaf entry = $\langle \text{n-dim box, ptr to child node} \rangle$
 - Box covers all boxes in child node (in fact, subtree).
- All leaves at same distance from root.
- Nodes can be kept 50% full (except root).
 - Can choose a parameter m that is $\leq 50\%$, and ensure that every node is at least $m\%$ full.

Example of a R-Tree



Example of a R-Tree (cont'd)



GeoPandas

Processing of GeoData in Python

- GeoPandas
 - <http://geopandas.org>
- Builds on and internally uses a number of other Python frameworks
 - **Shapely**: Manipulation and analysis of geometric objects in the Cartesian plane.
 - **GeoPandas**: geographic data
 - **Geos**: geospatial measures, functions and predicates (same core as PostGIS)
 - **Fiona**: library for reading&writing geo-formats, such as ESRI Shapefiles, Mapinfo TAB files, ...
 - Libspatialite
 - Hdf5
 - ...
- Might need post-installation with Anaconda: `conda install geopandas`

GeoPandas Data Types

- Provides two main geo data types
 - **GeoSeries** and **GeoDataFrame**
- **GeoSeries**
 - Main geometry building block on which GeoDataFrame is build
 - A GeoPandas geometry is an extension of a shapely.geometry object
 - Six classes, closely resembling the OGC Data Model
 - Single entity (core, basic types):
 - Point
 - Line
 - Polygon
 - Homogeneous entity collections:
 - Multi-Point, Multi-Line, Multi-Polygon

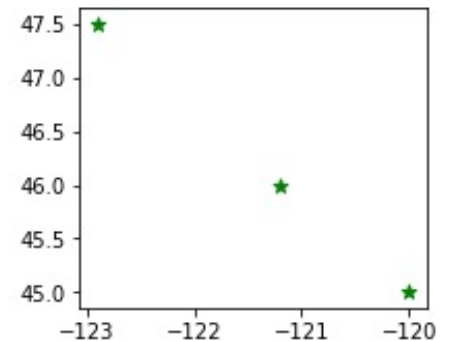
Example GeoSeries

```
// create a GeoSeries object from 3 Points
gs = GeoSeries([Point(-120,45), Point(-121.2,46), Point(-122.9,47.5)])

// check new object
gs
type(gs), len(gs)

// set WGS84 reference system
gs.crs = {'init': 'epsg:4326'}

// visualise using Matplotlib.PyPlot
gs.plot(marker='*', color='red', markersize=100, figsize=(4, 4))
plt.xlim([-123, -119.8]) plt.ylim([44.8, 47.7]);
```



GeoSeries Attributes and Methods (selection)

Attributes

- **area**: shape area (in units of current CRS projection)
- **bounds**: tuple of max and min coordinates on each axis for each shape
- **total_bounds**: tuple of max and min coordinates on each axis for entire GeoSeries
- **geom_type**: type of geometry.

Basic Methods

- **distance**(other): returns Series with minimum distance from each entry to other
- **centroid**: returns GeoSeries of centroids
- **to_crs()**: change coordinate reference system
- **plot()**: plot GeoSeries

Relationship Tests

- **contains**(other): is shape contained within other
- **intersects**(other): does shape intersect other

CRS – Coordinate Reference System

- Similar to the SRID from OGC, GeoPandas uses a CRS to tell Python how coordinates relate to places on the Earth.
 - codes for most commonly used projections: www.spatialreference.org
 - one of the most commonly used CRS is WGS84
 - can be referred to by EPSG codes, e.g., for WGS84: "+init=epsg:4326"
- Setting a Projection:
 - Can manually set a projection by assigning the correct codes to **crs** attribute:

```
my_geoseries.crs = {'init' : 'epsg:4326'}
```
 - You can see an objects current CRS through the crs attribute: **my_geoseries.crs**
 - Note: Data loaded from a reputable source (using the `from_file()` command) *should* always include projection information.
- Re-Projecting: geo-objects support a `to_crs(...)` function
Example:

```
my_geoseries.to_crs({'init' : 'epsg:3395'})
```

GeoDataFrame

- GeoDataFrame allows to combine normal data features and values with spatial data coming from GeoSeries
- The most important property of a GeoDataFrame is that it always has one GeoSeries column that holds a special status.
- This GeoSeries is referred to as the GeoDataFrame's “**geometry**”.
- When a spatial method is applied to a GeoDataFrame (or a spatial attribute like **area** is called), this commands will always act on the “**geometry**” column.

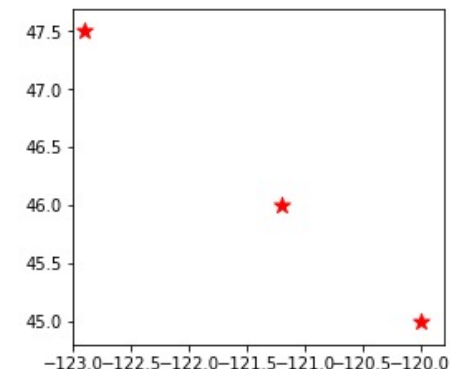
Example GeoDataFrame

```
// create a standard DataFrame from some data including coordinates
data = {'name': ['a', 'b', 'c'],
        'lat': [45, 46, 47.5],
        'lon': [-120, -121.2, -122.9]}
df = pd.DataFrame(data)
df
```

	name	lat	lon
0	a	45.0	-120.0
1	b	46.0	-121.2
2	c	47.5	-122.9

```
// use above's data to create a GeoDataFrame with 'list-of-shapes'
geometry = [Point(xy) for xy in zip(df['lon'], df['lat'])]
gdf = GeoDataFrame(df, geometry=geometry)
```

```
// visualise using built-in plot functionality
gdf.plot(marker='*', color='green',
          markersize=50, figsize=(3, 3));
```



Plotting & Mapping

- *geopandas* provides a high-level interface to matplotlib library for making maps.
- Mapping shapes is as easy as using the **plot()** method on a *GeoSeries* or *GeoDataFrame*.
 - Without parameters gives already simple plot with random colors
 - Can be customized to liking
 - E.g. colors, legend, titles...
 - Maps (with the same CRS!) can be overlayed in one plot

Reading GeoSpatial Data From a Shape File + Plotting

GeoPandas.read_file() uses internally the Fiona library for reading and writing all kinds of geo-formats, such as ESRI Shapefiles, Mapinfo TAB files, ...

```
# read some standard example dataset that come with geopandas
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
cities= gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))

world.head()
world.crs
world.plot()

# make sure cities are in same CRS than world data
cities.head()
cities = cities.to_crs(world.crs)

# plot cities on top of base world map (2-layer plot)
base = world.plot(color='white', edgecolor='black')
cities.plot(ax=base, marker='o', color='red', markersize=5)
```

Reading GeoSpatial Data From PostGIS

- GeoPandas can read geometry data from PostGIS using **read_postgis()** command

- Example:

```
import json
import psycopg2
conn = psycopg2.connect(db_conn_dict)
...
seas = gpd.read_postgis("SELECT * FROM world_seas", conn, coerce_float=False)
conn.close()

seas.crs
seas.head()
seas.plot(column='oceans', categorical=True, legend=True, figsize=(14, 6));
```


Reading GeoSpatial Data From GeoJSON

Cf. Jupyter notebook demo:

```
import requests
import geojson
headers_dict = {'accept': 'application/text', 'authorization': '*****'}
opendata_url = "https://api.transport.nsw.gov.au/v1/life/hazards/incident/open"
r = requests.get(opendata_url, headers=headers_dict)

response = r.text.replace('POINT', 'Point') # quick hack to fix data for geojson
incidents_geo = geojson.loads(response)

print(type(incidents_geo))
print(incidents_geo.keys())

incidents_gdf = GeoDataFrame.from_features(incidents_geo)
incidents_gdf.head()
incidents_gdf.plot(ax=world.plot(cmap='Set3',figsize=(10,6)), marker='o', color='red', markersize=15);
bounds=incidents_gdf.geometry.bounds
plt.xlim([bounds.minx.min()-5, boiunds.maxx.max()+5]);
plt.ylim([bounds.miny.min()-5, boiunds.maxy.max()+5]);
```

Joining GeoPandas Data

– Attribute Joins

- Two GeoDataFrames can be joined on matching attribute values using the **merge()** function
- Example: add country names to country_shapes using ISO codes
`country_shapes = country_shapes.merge(country_names, on='iso_a3')`

– Spatial Joins

- **sjoin()**: Merge two geometry objects based on their spatial relationship
- Example:
`cities_with_country = gpd.sjoin(cities, countries,
 how="inner", op='intersects')`

Data-Exchange of Spatial Data using XML and JSON (spatial web APIs)

Motivation: Geo-aware Web API

- Web APIs are very important for modern/mobile applications
 - RESTful APIs allow for low-latency dynamic applications
 - return results as JSON or XML
- Location-aware services nowadays very popular
- Need to be able to represent spatial data in JSON or XML formats
 - **GeoJSON**
 - **KML**
 - many more...

Web Service Standards

- RESTful APIs (also: RESTful web services)
 - REST: Representational State Transfer
 - Stateless operations performed on a simple, uniform API using standard HTTP requests (eg. GET or PUT)
 - requests made to a resource's URL responded with HTML, XML, or JSON
- XML Web Services
 - XML-message based (both ways: request and response)
 - Based on the “Simple Object Access Protocol” (SOAP), but actually the much more complex (but also more powerful) standard
 - Not further covered in this lecture
- If we refer to web service in DATA2001, we mean REST APIs

GeoJSON

- GeoJSON is a format for encoding of geographic data structures as JSON
 - Define spatial data as features of standard JSON objects
 - supports various geometry types
 - Point, LineString, Polygon, MultiPoint, MultiLineString and MultiPolygon
 - and additional properties

For examples, see also: <https://en.wikipedia.org/wiki/GeoJSON>

- Example:

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

[source: geojson.org]

- GeoJSON Specification: RFC7946 (Aug 2016)

GeoJSON Examples

- <https://www.data.gov.au/dataset/street-food>
- <https://data.nsw.gov.au/data/dataset/nsw-rural-fire-service-current-incidents-feed>
- <https://opendata.transport.nsw.gov.au/node/342/exploreapi>
- <http://geojson.io> - website for authoring and sharing GeoJSON files
- geojsonlint.com - website that checks validity of GeoJSON data
- Python support library: geopandas (note: not installed in labs)
- Utility service:
<http://jsonprettyprint.com>

GeoJSON as Output Format

- You can also use GeoJSON to display geometries on maps
- Example: Google Maps API
 - <https://developers.google.com/maps/documentation/javascript/datalayer>

```
var map;  
function initMap() {  
  map = new google.maps.Map(document.getElementById('map'), {  
    zoom: 4,  
    center: {lat: -28, lng: 137}  
  });  
  
  // NOTE: This uses cross-domain XHR, and may not work on older browsers.  
  map.data.loadGeoJson('https://storage.googleapis.com/mapsdevsite/json/google.json');  
}
```

- Alternatively on OpenStreetMap

KML – the Keyhole Markup Language

- Spatial file format originally developed for Google Earth
 - Since 2008 international standard of the Open Geospatial Consortium
- XML variant, more widely used now
 - KML files specify a set of features (place marks, images, polygons, 3D models, textual descriptions, etc.) for display in spatial software
- Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <Placemark>
      <name>New York City</name>
      <description>New York City</description>
      <Point>
        <coordinates>-74.006393,40.714172,0</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```
- Documentation:
 - https://developers.google.com/kml/documentation/kml_tut

Example Usages

- <http://www.ga.gov.au/earthquakes/searchQuake.do>
- https://www.data.gov.au/dataset?q=&res_format=KML
 - Eg. “City of Gold Coast Road Closures”
 - or “Ipswich City Boundary”
 - => also available as GeoJSON
 - => Possible to visualise with eg., <http://geojson.io>
(there is also a Python library for this called **geojsonio**)
- There are currently already hundreds of open spatial data sets available via data.gov.au which use either KML or GeoJSON

Use Cases

1. Ingest Data with spatial features
 - Such as geographic location (point data) or a geometry (spatial object)
 - Store and index in database for later processing
 - **Spatial indexing** important
 - make sure coordinate systems (SRID) are compatible
 - Be aware of **spatial query types** and **spatial joins**
2. Use web APIs with support for GeoJSON to lookup further data
 - Eg. Boundaries of localities
 - Parsing GeoJSON required => semi-structured data techniques of last week
3. Use GeoJSON or KML to export or visualize spatial data

Other Spatial Data Exchange formats

- **OSM XML**
 - Open Street Map XML exchange format
 - https://wiki.openstreetmap.org/wiki/OSM_XML
- **WSM**
 - **Web Map Service (WMS)** by the [Open Geospatial Consortium](#) (OGC)
 - Other OGC standards for web geo-APIs:
Web Feature Service (WFS) and Web Coverage Service (WCS) data
- **MapInfo TAB** format
- **Esri Shapefile** Format
- **Esri Rest API**
 - ArcGIS REST API from commercial ArcGIS application; returns JSON

Trajectories (not examinable)

- A very active research topic, is what are the best data structures, index structures, algorithms for trajectories
 - This means a track of a moving object, which has a different location at various times
 - Typically assume straight-line notion between the observed points
- Example: cars moving around a city

Lessons Learned

- **Spatial Data Models**

- *Field based and Object based*
- Object based for modelling discrete entities, like country
- OGC Spatial Data Model; coordinate systems; Nine-Intersection Model

- **Spatial Querying**

- *Range Queries; Nearest Neighbour Queries; Spatial Join Queries*
- *2-Phase Filter-Refine Strategy*
- Spatial Index Structures: *B-Tree* on 1-dimensional data, else *R-Tree* or *GiST*

- **PostGIS and GeoPandas**

- **Spatial Data Exchange Format for Spatial Web APIs**

- GeoJSON, KML, ...
- Use cases for spatial data exchange or data visualisation

References

- S. Shekhar and S.Chawla: *Spatial Databases: A Tour*. Prentice Hall, 2002. (Chap. 2 – 5)
- P. Rigaux, M. Scholl and A. Voisard: *Spatial Databases - With Application to GIS*. Morgan Kaufmann, 2002. (especially Section 2.1, Chapter 3, and Chapter 6)
- Ramakrishnan and Gehrke: *Database Management Systems*. 3rd Edition, McGraw-Hill, 2003. (Chapter 28)
- Python Libraries including GeoPandas:
 - <https://geopandas.org>
 - <https://pypi.org/project/Shapely>
 - Have also a look at **geojsonio**
 - Great tutorial from the 2018 GeoHackWeek by the University of Washington (UW):
 - <https://geohackweek.github.io/vector/04-geopandas-intro/>
- PostgreSQL Online documentation
 - <https://www.postgresql.org/docs/current/static/datatype-geometric.html>
 - <https://www.postgresql.org/docs/current/static/functions-geometry.html>
 - <https://www.postgresql.org/docs/current/static/gist.html>
 - PostGIS documentation: <http://postgis.net/documentation>