
ANIMOVE
P1.y=diag(c(0, 0.001, 0.001))
splayPar(mov.model=~1, err.model=list(x=~errX, y=~errY),
data=nfsNew, fixPar=c(NA, 1, NA, 1, NA, NA, NA, NA))
t <- crwMLE(mov.model=~1, err.model=list(x=~errX, y=~errY),
data=nfsNew, coord=c("longitude", "latitude"),
Time.name="Time", initial.state=initial.drift,
fixPar=c(NA, 1, NA, 1, NA, NA, NA, NA),
control=list(maxit=2000,trace=1, REPORT=10),

June 2024

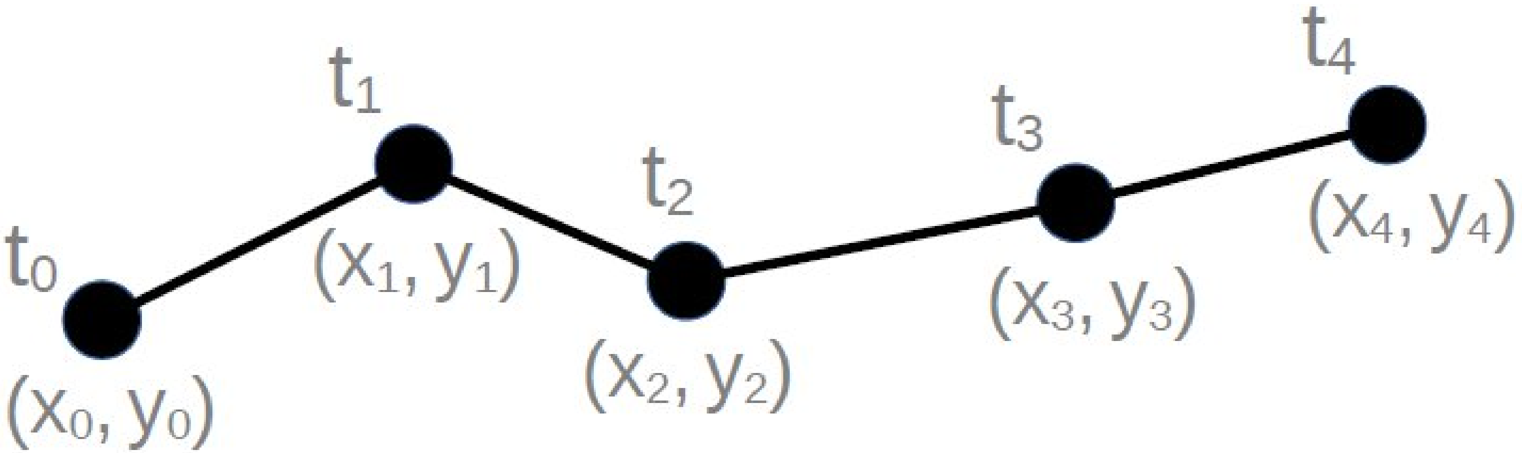
Movement Data

Defining movement, space and time

Definition of Movement

Movement is defined as **location(s) through time**.

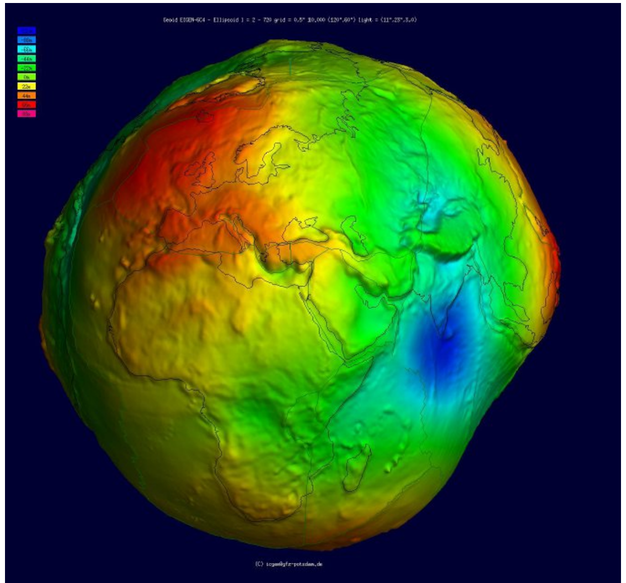
Movement is a spatio-temporal object, therefore **correct definition of space and time** is fundamental to manipulate the data in an appropriate way.



Defining SPACE

How do we agree on where things are on Earth?
We need to place them on a **reference system**!

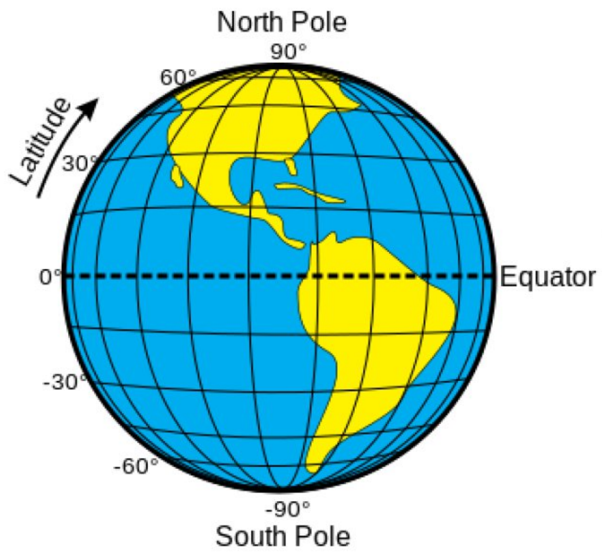
- **Geoid**: geometric shape of the Earth → approximated as a rotating sphere (ellipsoid)


The Geoid (scale factor: 1000) – *Wikimedia Commons, 2022*

Geographical coordinate system

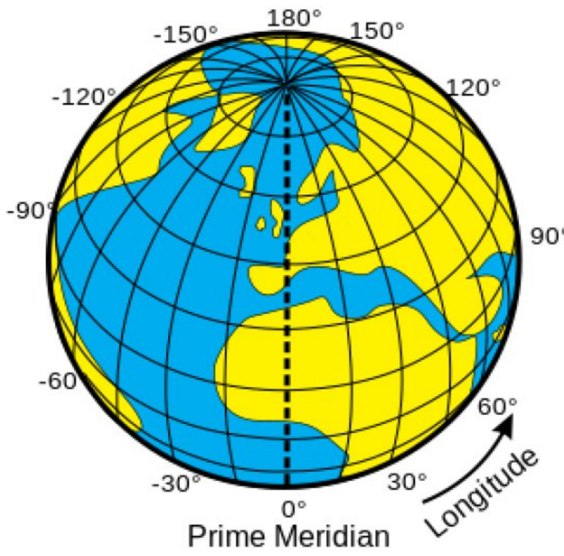
The **Geographic Coordinate System (GCS) is spherical and uses angles** to define a location on Earth:

- **Equatorial plane**: simply derivable from the axis of rotation (natural)
- **Prime Meridian**: Artificially defined, Eastern and Western hemisphere
- **Latitude**: angle between equatorial plane and plane perpendicular of rotating sphere at the point
- **Longitude**: angle between Prime Meridian and a plane through North and South pole at point



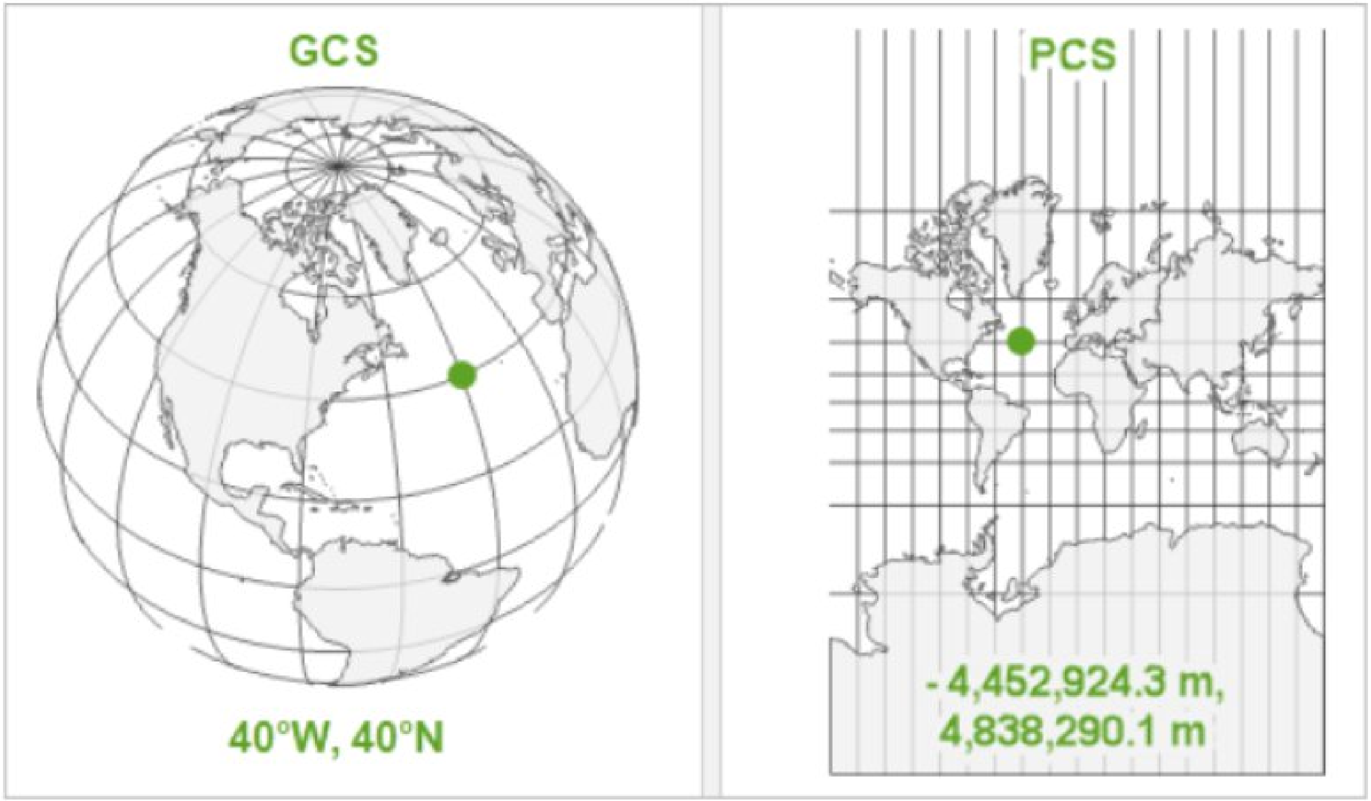
The Geographic Coordinate System is using **360 longitudes and 180 latitudes** and uses the following systems:

- DD Decimal Degrees (29.1000°,-113.3000°)
- DM Degrees Decimal Minutes (39°33.0', -125°31.0')
- DMS Degrees Minutes Seconds (e.g. 39°23'05" N, 113°27'01" W)



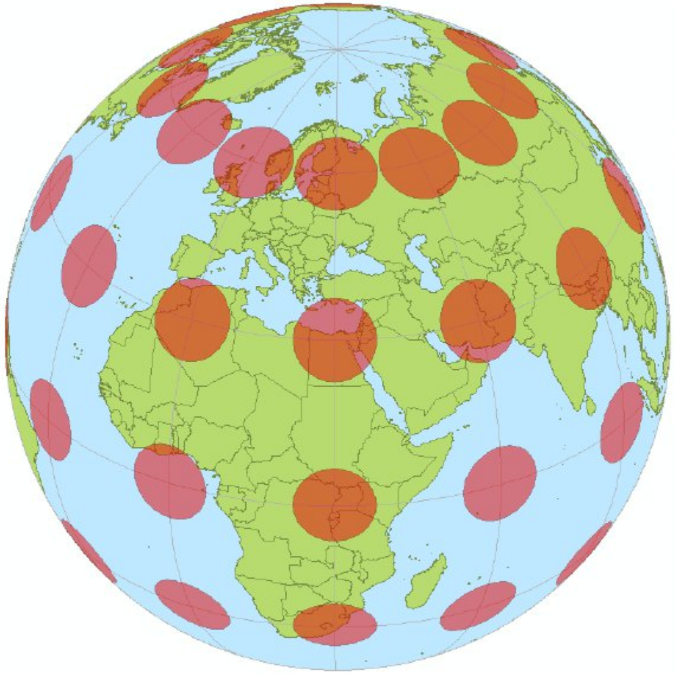
Projected coordinate system

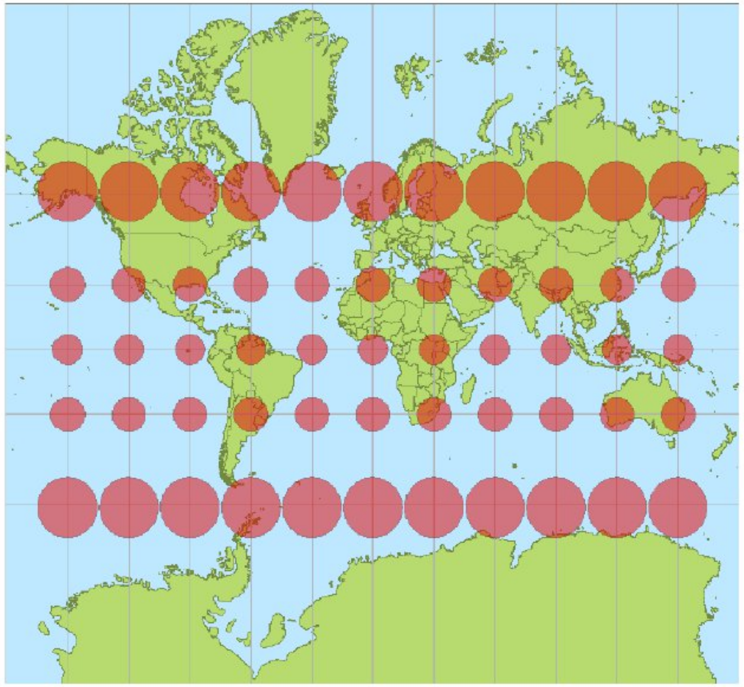
The Earth is approximately a sphere, but paper maps and computer screens are flat.
A **projected coordinate system (PCS)** tells us how data located on this sphere can be drawn in 2D (metres).



Projections and distortions

It is not possible to flatten the skin of a peeled orange on the table without crinkles and cuts, meaning..

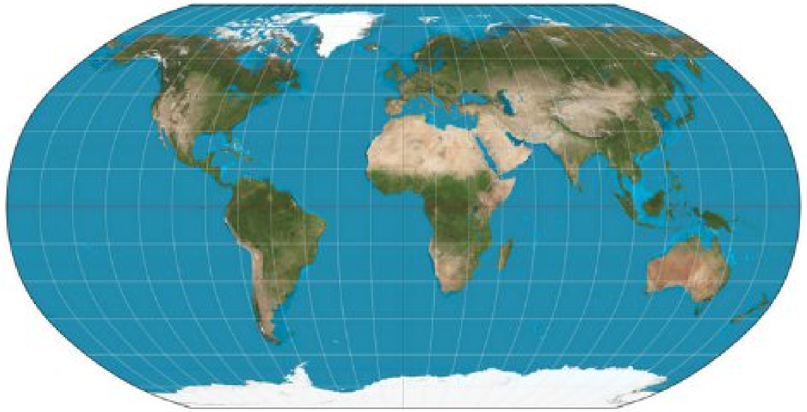


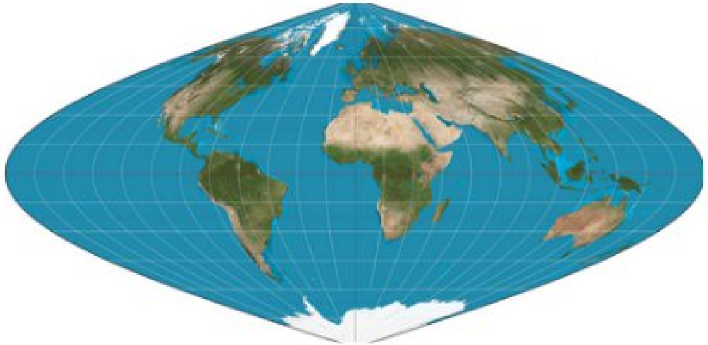


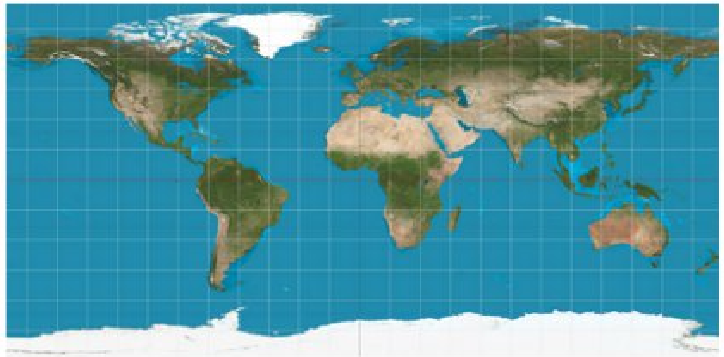
Distortions visualized for the often-used Mercator projection: Direction and shapes are correct, but area and distance are distorted – Wikimedia Commons, 2022

..we cannot map a geometric feature on a sphere in 2D without some degree of distortion in at least 2 of four fundamental geometric properties:
shape, area, distance and direction.

Shapes of continents in different projections

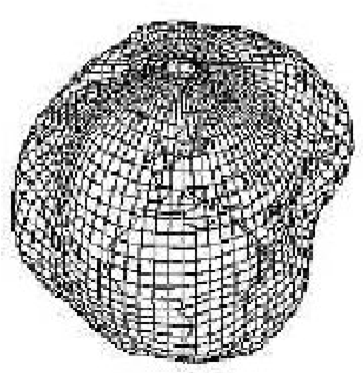




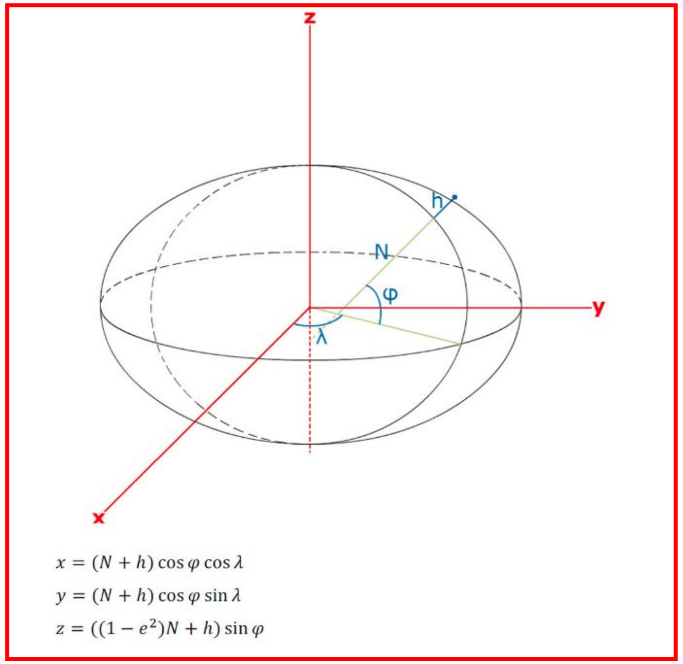


Coordinate Reference System (CRS)

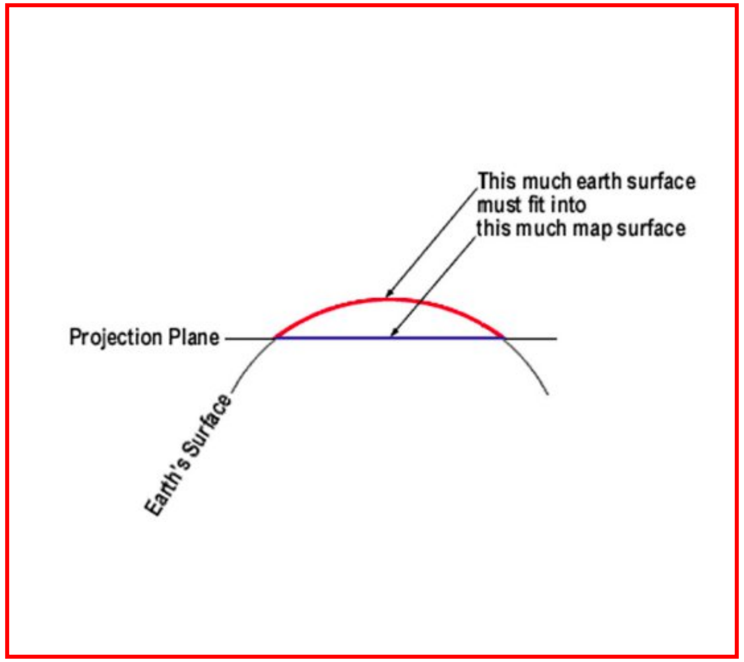
A CRS describes a projection, mathematical warping/transformation of the spherical data (3D) into a planar system (2D).



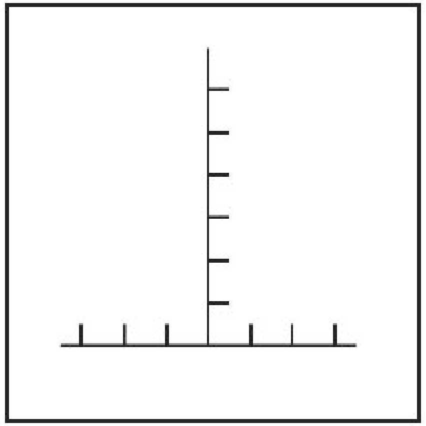
Geoid



Datum (ellipsoid type)



Coordinate system/Projection



Planar map with coordinate system

Coordinate Reference System (CRS)

Defining the Coordinate Reference System (CRS)

A CRS describes a projection, mathematical warping/transformation of the spherical data (3D) into a planar system (2D). It consists of:

- *Datum*: defines the type of ellipsoid approximating the sphere and how it is placed (e.g. the popular WGS84)
- *Coordinate System*: projecting a coordinate grid of some sort onto the geodetic datum
- *Units*: spatial measure used by the CRS, e.g. meters, feets, degrees.

Human-readble CRS designation:
Universal Transverse Mercator (UTM) World Geodetic System 1984 (WGS84) Zone 32N

Authority ID:
EPSG:32632

Proj4 string:

```
+proj=utm +zone=32 +datum=WGS84 +units=m +no_defs
```

Well-known text (WKT)

```
PROJCRS["WGS 84 / UTM zone 32N",
    BASEGEOGCRS["WGS 84",
        DATUM["World Geodetic System 1984",
            ELLIPSOID["WGS 84",6378137,298.257223563,
                LENGTHUNIT["metre",1]]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["degree",0.0174532925199433]],
        ID["EPSG",4326]],
    CONVERSION["UTM zone 32N",
        METHOD["Transverse Mercator",
            ID["EPSG",9807]],
        PARAMETER["Latitude of natural origin",0,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8801]],
        PARAMETER["Longitude of natural origin",9,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8802]],
        PARAMETER["Scale factor at natural origin",0.9996,
            SCALEUNIT["unity",1],
            ID["EPSG",8805]],
```

Helpful webpages to find **projections definitions** for R:
https://epsg.io/ and *http://spatialreference.org/*

Which projection to choose

It depends on **scale**, **purpose** of analysis, CRS of **other spatial data** that we need to work with. Correctly defining the CRS is fundamental to **minimise the distortion** in the measures we are interested in (not more than 2), and to be able to **use data from different sources** and reproject them to a uniform projection.

Conformal
map projections preserving angles locally and thereby shape

Equal-Area
map projections preserving area

Equidistant
map projections preserving distance

True direction
map projections preserving direction

→

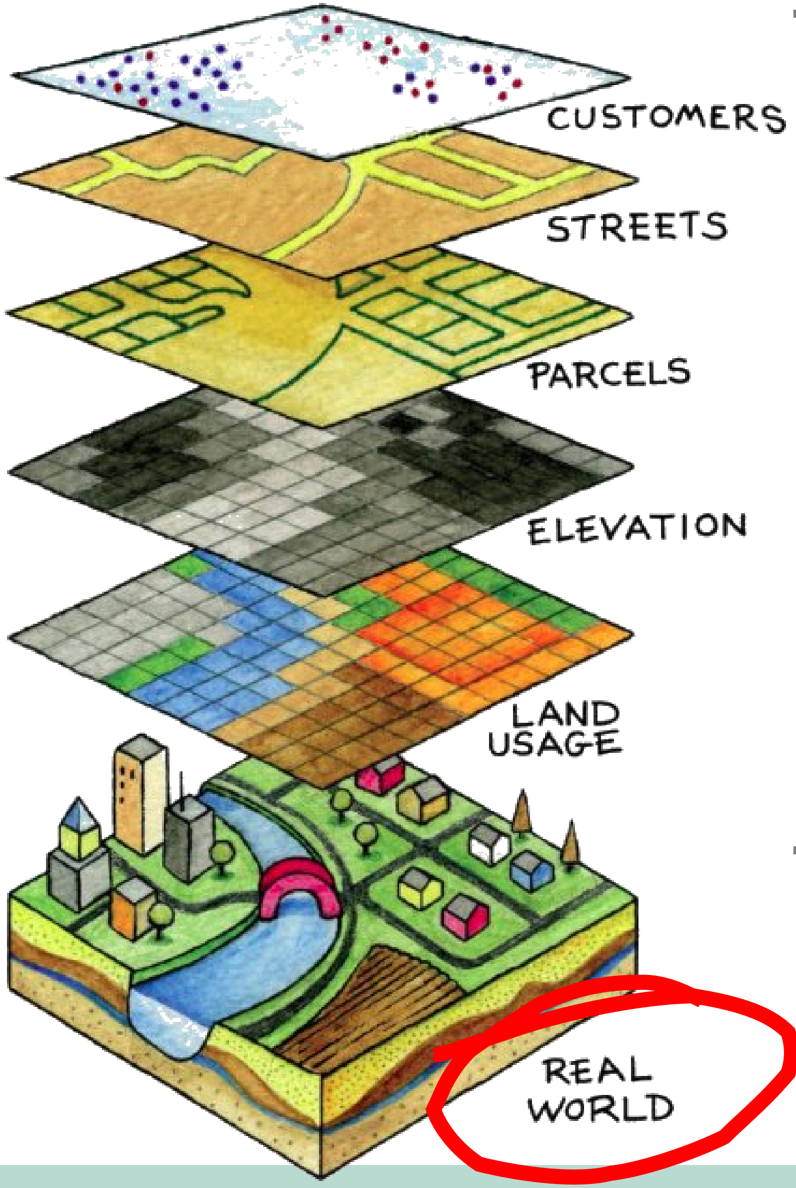
Global Scale
e.g. Sinusoidal Equal Area

Continental Scale
e.g. Lambert Azimuthal Equal Area

Small Scale
e.g. Universal Transverse Mercator (UTM) with WGS84, Lambert Conformal Conic

Helpful webpages to understand and choose projections:
https://www.geo-projections.com/
https://www.esri.com/arcgis-blog/products/arcgis-pro/mapping/gcs_vs_pcs/
https://learn.arcgis.com/en/projects/choose-the-right-projection/

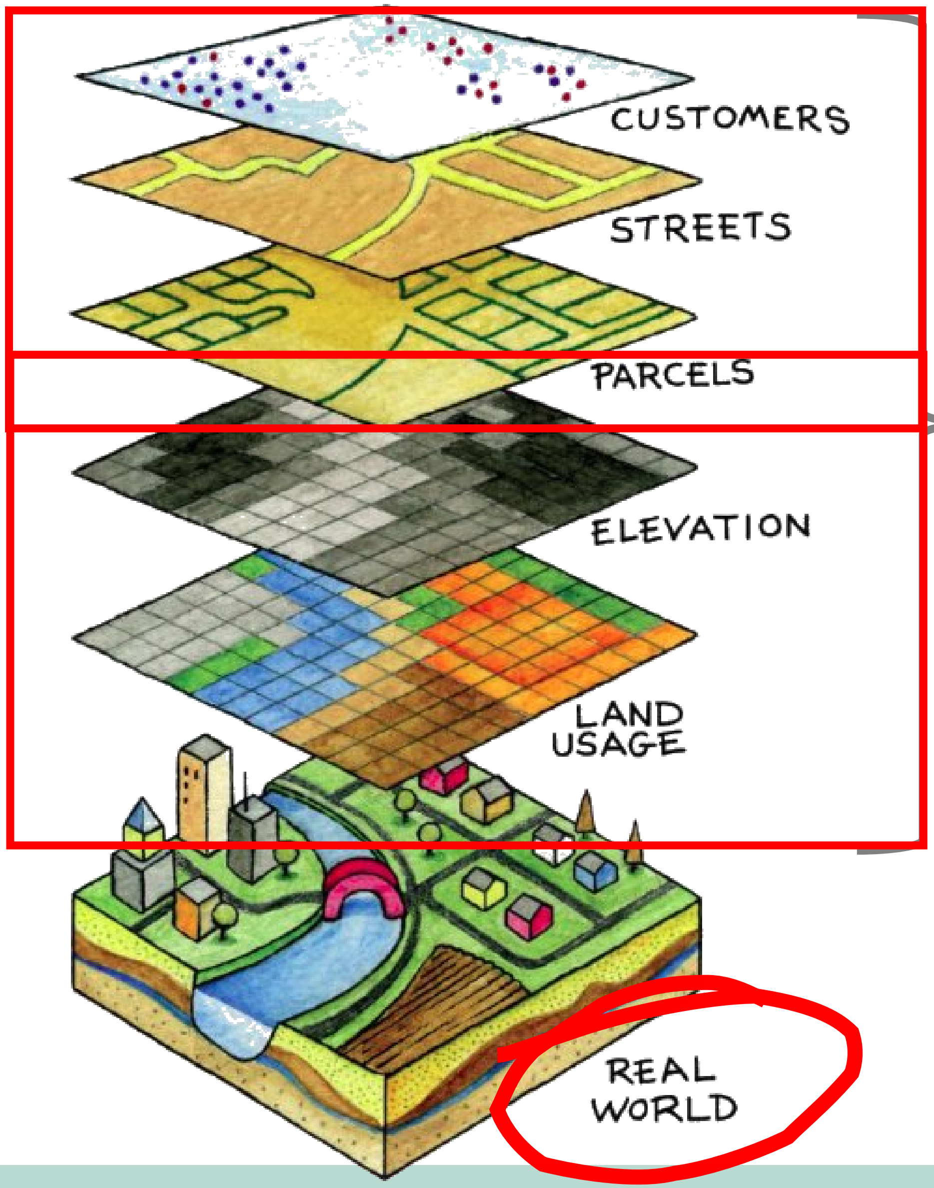
Spatial data models



DATA MODELS

Representations of the real world

Spatial data models

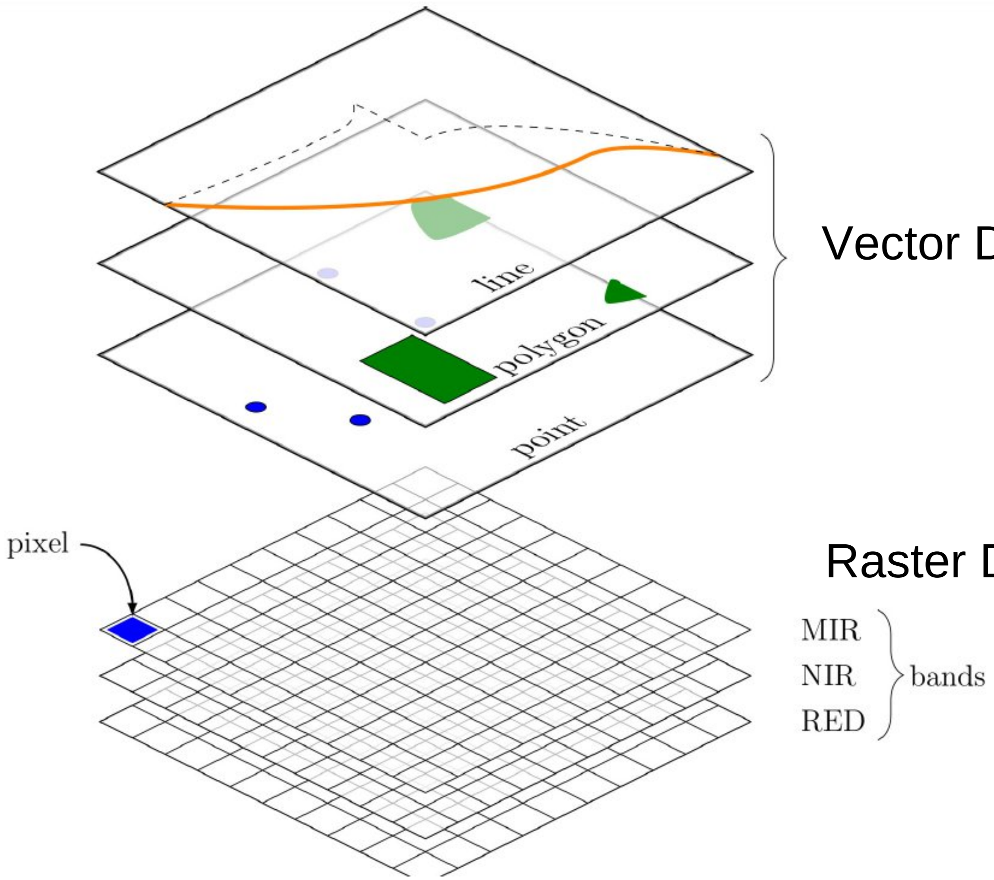


Vector data

DATA MODELS

Raster data

Spatial data in R



Vector Data → sf

Raster Data → Terra

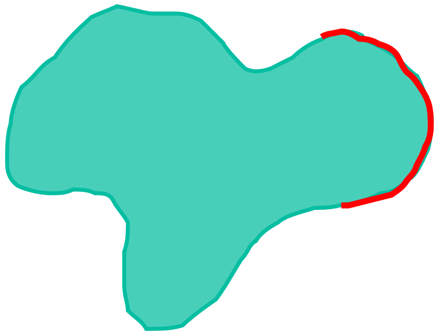
(but also **`raster`** package)

You will hear about this on Thursday!

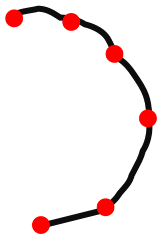
Learning sources:
Tutorial about the use of the `sf` package (for vectors): https://r-spatial.github.io/sf/articles/sf1.html
Tutorial about the use of the `terra` package (for rasters): https://rspatial.org/pkg/index.html

Vector data

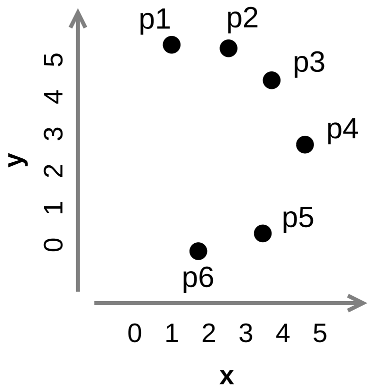
- **Points** – A coordinate pair
- **Lines** – Ordered set of coordinate pairs that begin and end with a node (point).
- **Polygons** (feature with area) – Ordered set of connected lines.



An AREA consists of...



LINES which consist of...



POINTS which consist of...

p1 = 0,5
p2 = 1,4.5
p3 = 3,4
p4 = 5,2.5
p5 = 3,1
p6 = 2,0

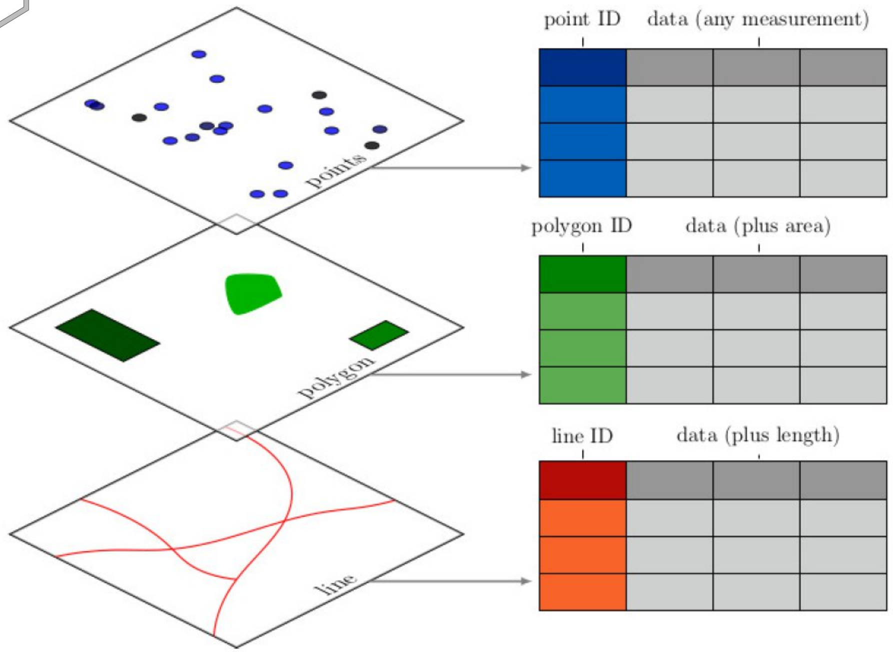
COORDINATES

Existing vector formats:
- *.GeoPackage (easier handling)
- *.gpx
- *.shp (widely used)
- *.kml

Now you know that these will be different depending on the CRS

Vector data in R

Simple feature (sf)



COORDINATES Geometry/Spatial attribute (R class `sf`)

Non-spatial attribute (optional) (R class `data.frame/tibble`)

A **feature** is an object (a tree, a building), and a feature can be made of many features:

type	description
`POINT`	zero-dimensional geometry containing a single point
`LINESTRING`	sequence of points connected by straight, non-self intersecting line pieces; one-dimensional geometry
`POLYGON`	geometry with a positive area (two-dimensional); sequence of points form a closed, non-self intersecting ring; the first ring denotes the exterior ring, zero or more subsequent rings denote holes in this exterior ring
`MULTIPOINT`	set of points; a MULTIPOINT is simple if no two Points in the MULTIPOINT are equal
`MULTILINESTRING`	set of linestrings
`MULTIPOLYGON`	set of polygons
`GEOMETRYCOLLECTION`	set of geometries of any type except GEOMETRYCOLLECTION

Vector data in R

Declaring a CRS vs reprojecting:

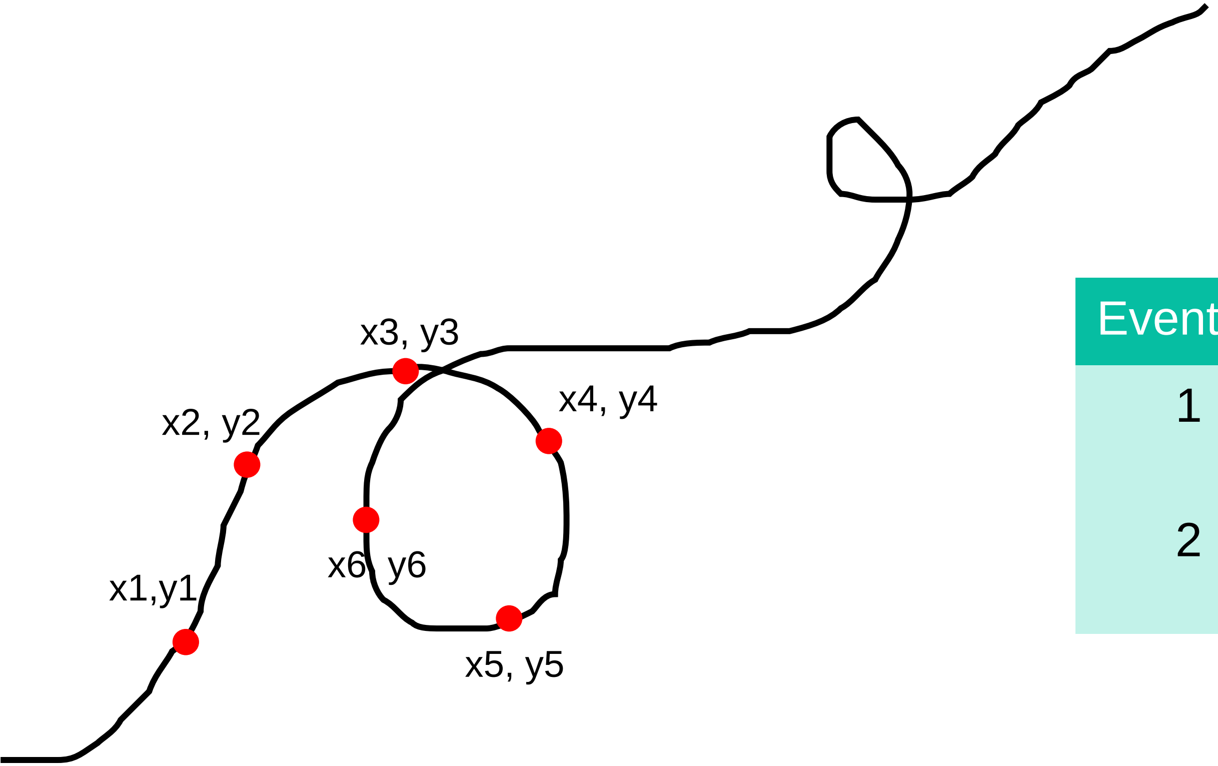
In R we can do that by working with proper spatial objects. But careful, **declaring** a projection:

```
> library(sf)
> YourData <- st_as_sf(YourData,
coords = c("location.long", "location.lat"), crs = "EPSG:4326")
> class(YourData)
[1] "sf" "data.frame"
> st_crs(YourData)
Coordinate Reference System:
User input: EPSG:4326
```

Is different from **reprojecting**, i.e. changing the coordinate system:

```
> YourDataProj <- st_transform(YourData, crs = "EPSG:32622")
```

Movement data are vector data... + TIME!



Event ID	Animal ID	Species
1	Leo	Turkey vulture
2	Leo	Turkey vulture

TIMESTAMP
2024-06-17 09:00:00
2024-06-17 09:10:00

COORDINATES
Geometry/Spatial attribute
(R class `sf`)

Non-spatial attribute
(R class `data.frame/tibble`)

TIME!

Defining Time

Time zones and light saving switches

Time settings are crucial. You have to know what your device used as time.
Many GPS devices use and report local time, some report UTC!

Helpful questions you have to ask yourself before exploring/analysing your data are:

- In which **time zone** does your device report time?
- Did you track over the **day light saving** switches? Does your animal cross time zones?
- Did you **set/declare** the time zone in R?

Setting vs converting timezones

Time format in R is not a real problem, but the names of the time zones are specific to each operating system (check the names recognised by your OS typing *OlsonNames()*).
The default time zone of your OS will also differ between computers, you can check it by typing *Sys.timezone()*
Basically time zones are likely to be a pain in the neck, no matter how experienced you are..

Before doing anything with times, remember to always **define the time zone in which your data were collected**. **POSIXct** is our object of choice to deal with time in R:

```
>(t <- as.POSIXct("2024-06-17 14:00:00", "%Y-%m-%d %H:%M:%S", tz="UTC"))
[1] "2024-06-17 14:00:00 UTC"
```

(Universal Time Coordinated)

Only **after** the time zone is set, time can be **converted to different time zones**:

```
> format(t, tz="Europe/Berlin") # or:
> lubridate::with_tz(t, tz="Europe/Berlin")
[1] "2024-06-17 16:00:00 CEST"
```

The `move2` package

The **`move2` package** provides a series of functions to **import, visualize and analyse** animal movement data and offers **direct connection to Movebank**.
The move object extends the properties of the **`sf` spatial object** by making **time a mandatory component** (**movement = location + time**).

sf + TIME = move2

The **`move2` package** is designed as a successor of the `move` package (based on the spatial packages `sp` and `rgdal`, now deprecated). `move2`, based on sf, also improves in speed and functionality, but for some functions we will still rely on its predecessor `move`.

For detailed information about these packages see:
https://bartk.gitlab.io/move/articles/move.html (for move)
https://bartk.gitlab.io/move2/ (for move2)

Hands on

How to do this in R in the script:
“1_LoadExportData_introMove2.R”