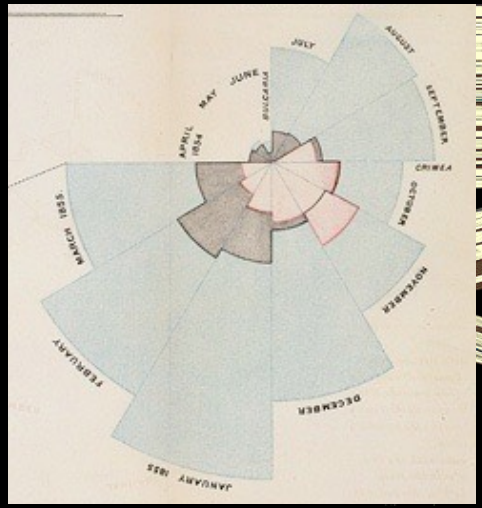
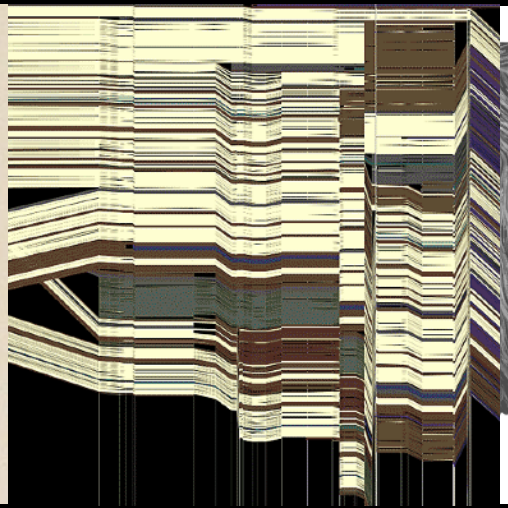
**CSE 442** - Data Visualization

# Mapping & Cartography



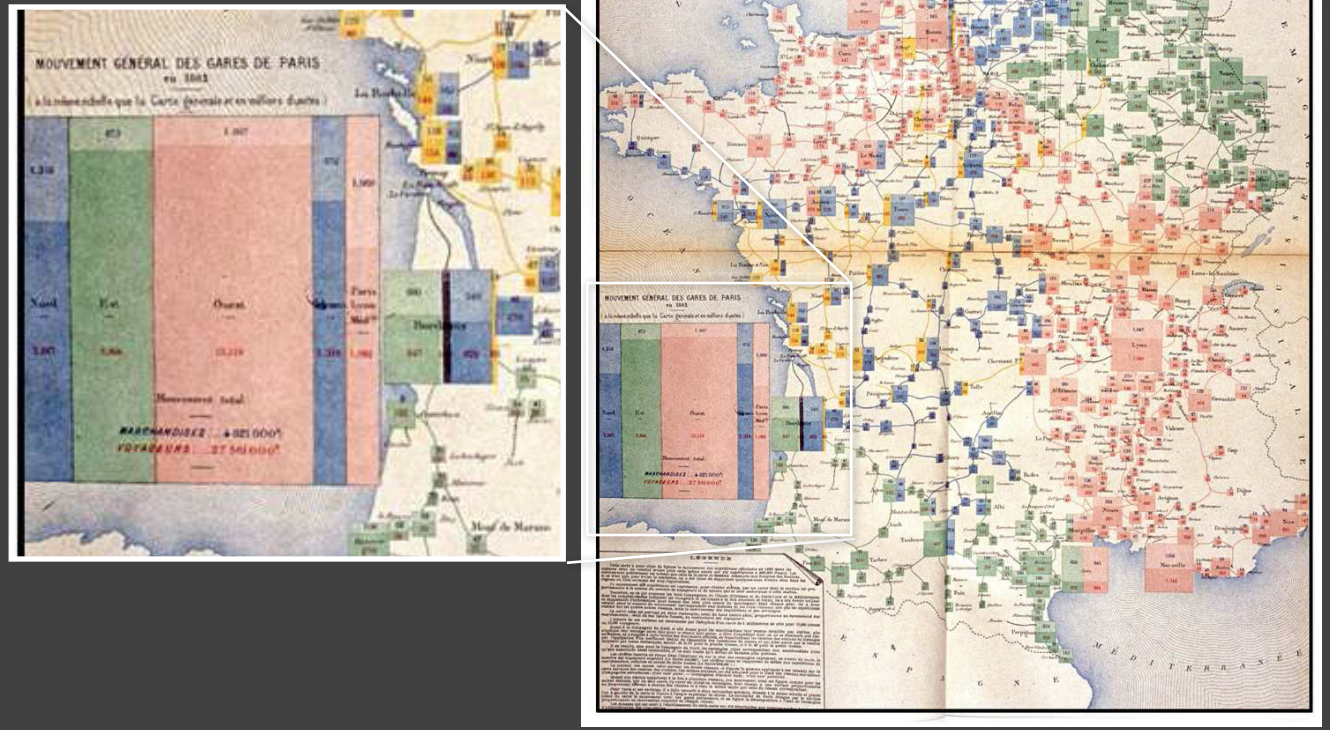

Jeffrey Heer University of Washington
(with significant material from Michal Migurski)

# Mapping

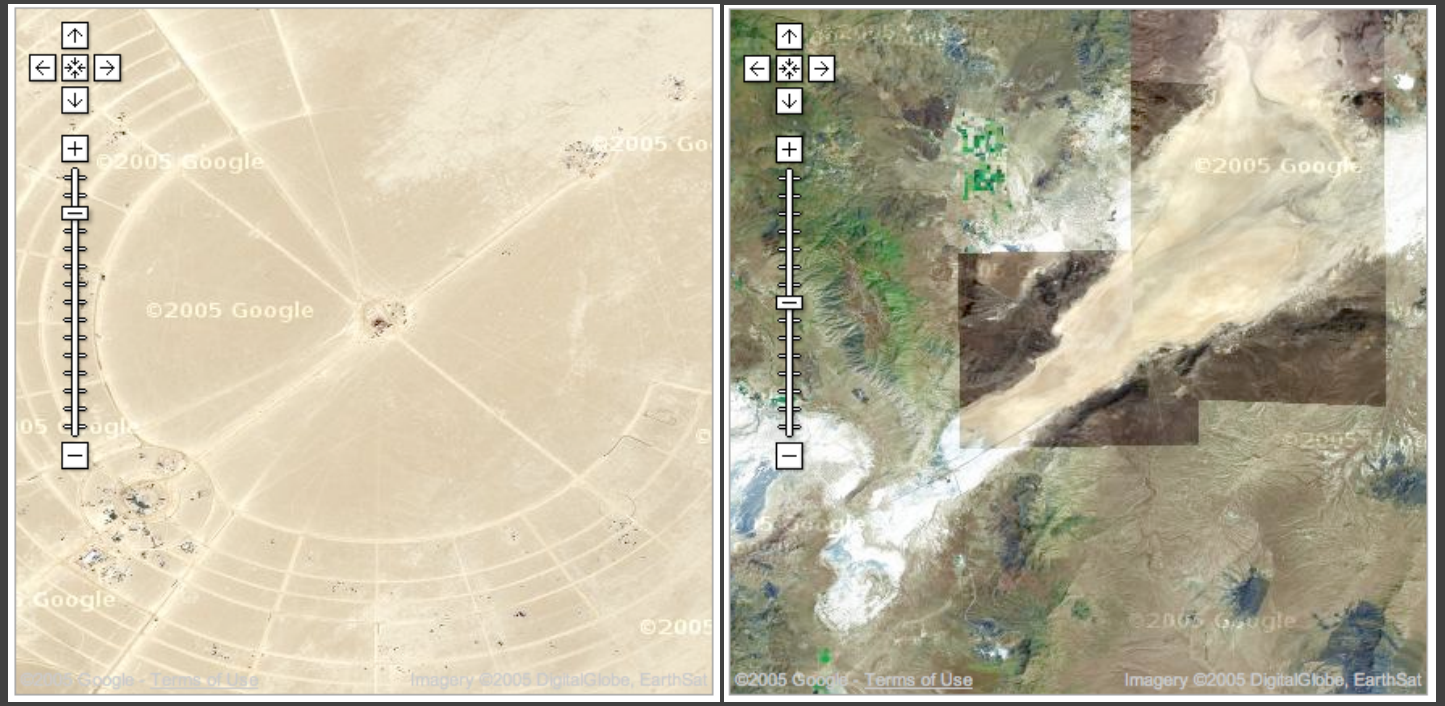
## Visualizing Geospatial Data


**Ptolemy's Geographica**

Original ~150AD, This Map ~1300AD

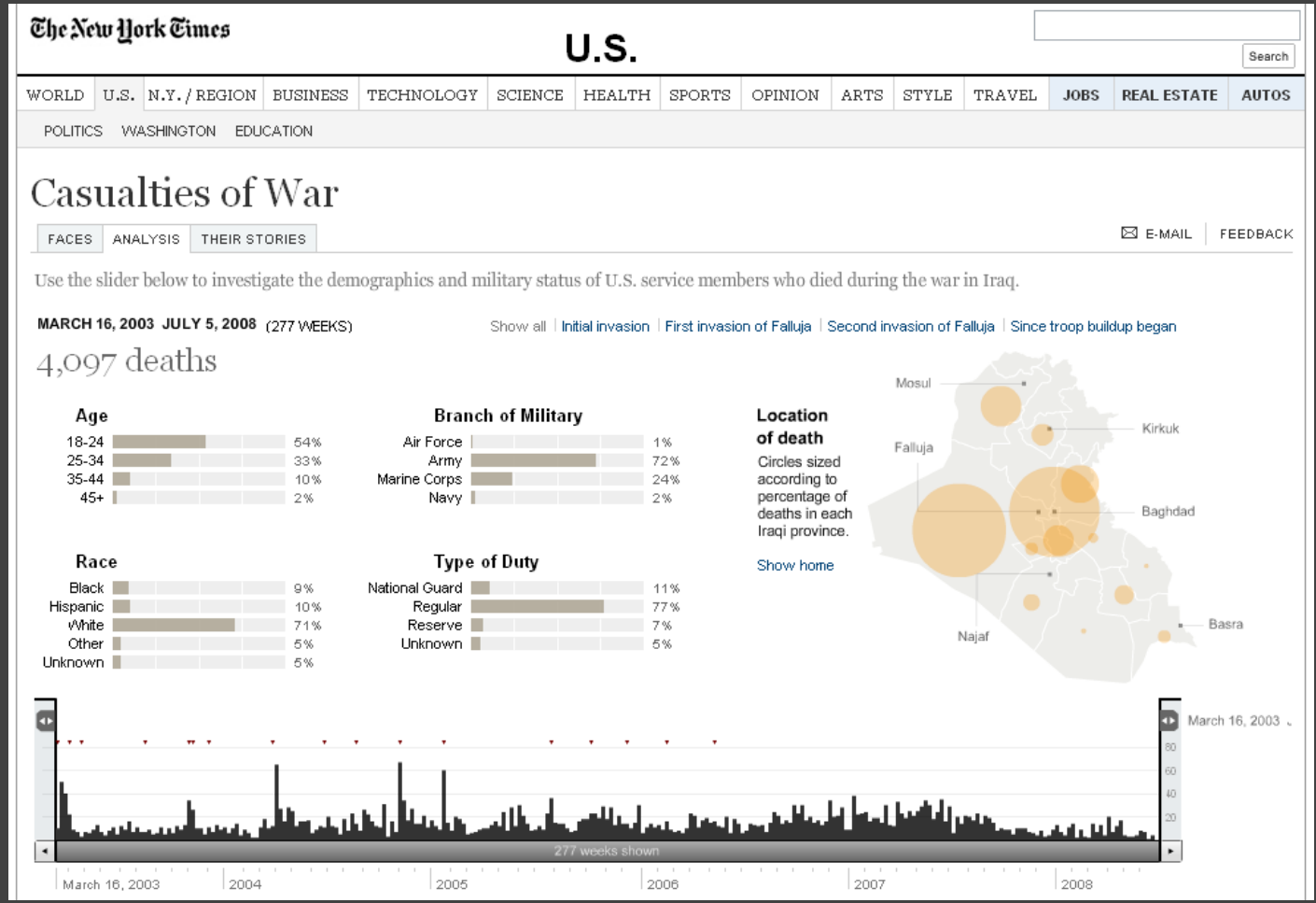



Rail Passengers and Freight from Paris 1884




Black Rock City, Nevada
(Burning Man)

Google Maps 2005




Casualties of War, New York Times 2006

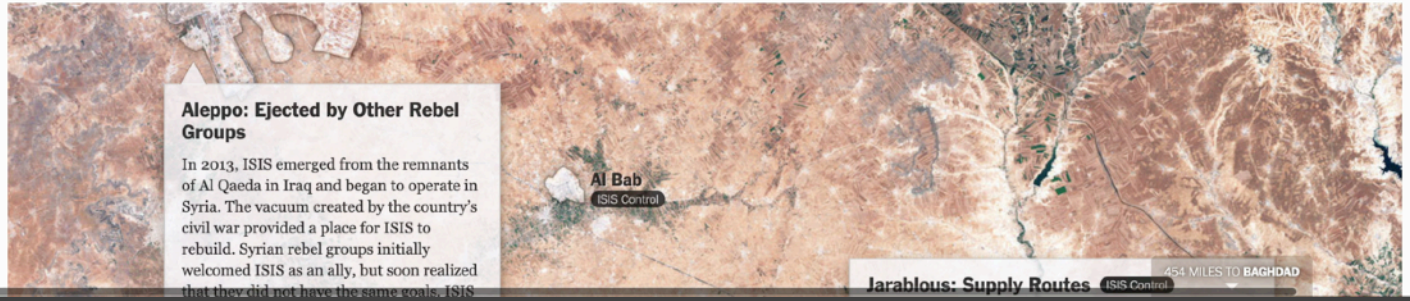

# A Rogue State Along Two Rivers

## How ISIS Came to Control Large Portions of Syria and Iraq

By JEREMY ASHKENAS, ARCHIE TSE, DEREK WATKINS and KAREN YOURISH July 3, 2014

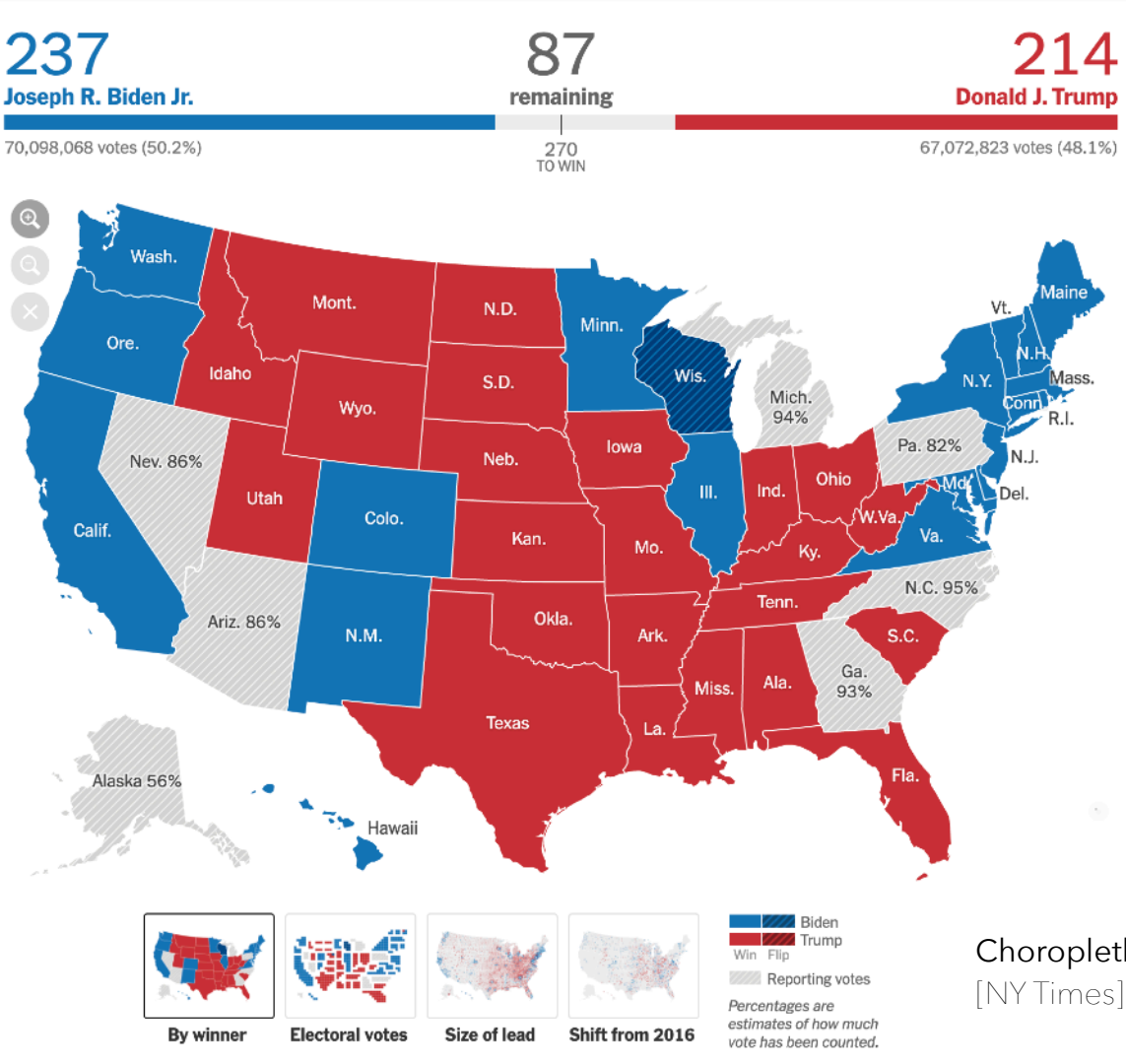
The militant group called the Islamic State in Iraq and Syria, or ISIS, seemed to surprise many American and Iraqi officials with the recent gains it made in its violent campaign to create a new religious state. But the rapid-fire victories achieved over a few weeks in June were built on months of maneuvering along the Tigris and Euphrates Rivers.

## The Euphrates

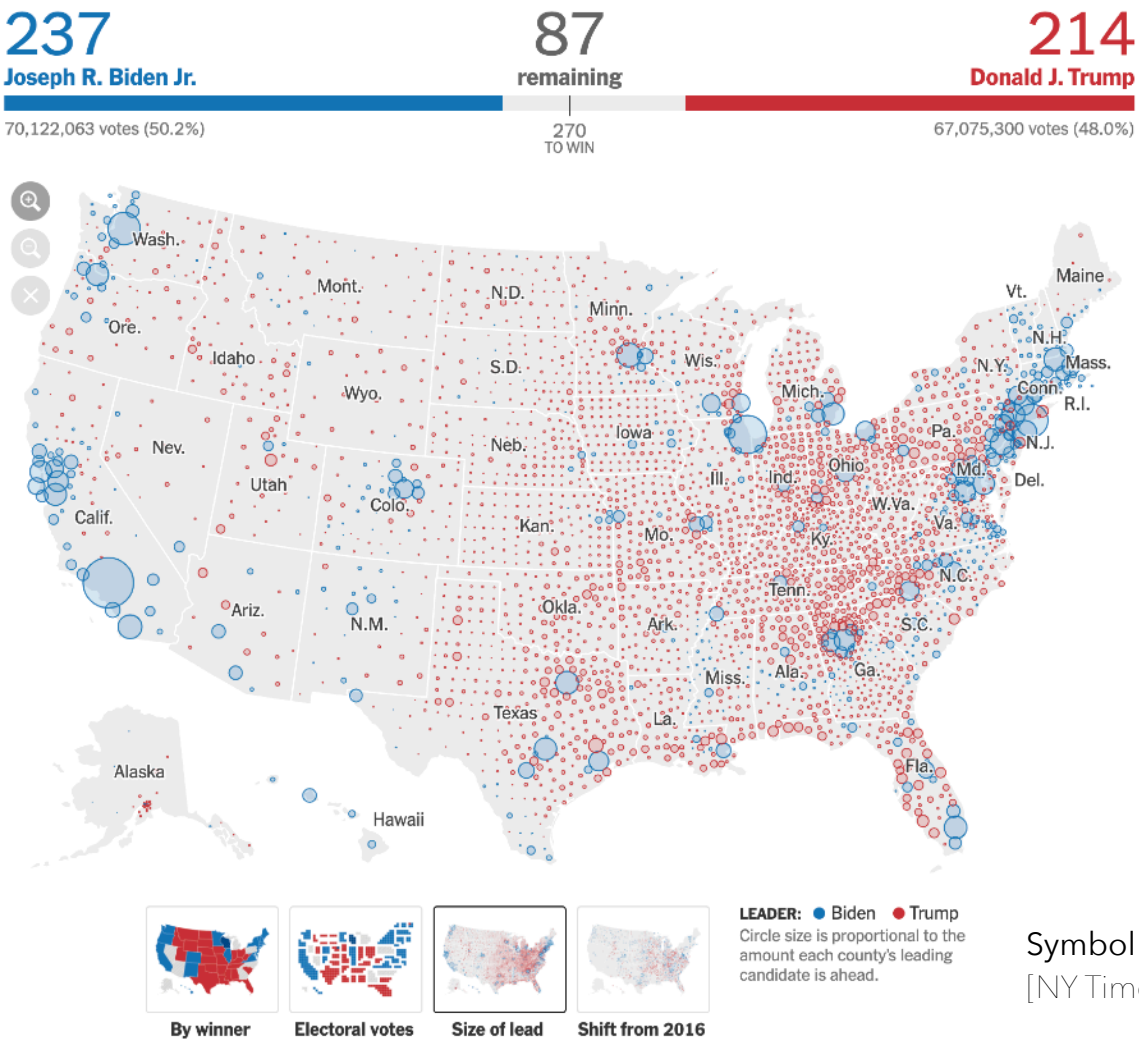



NY Times 2014

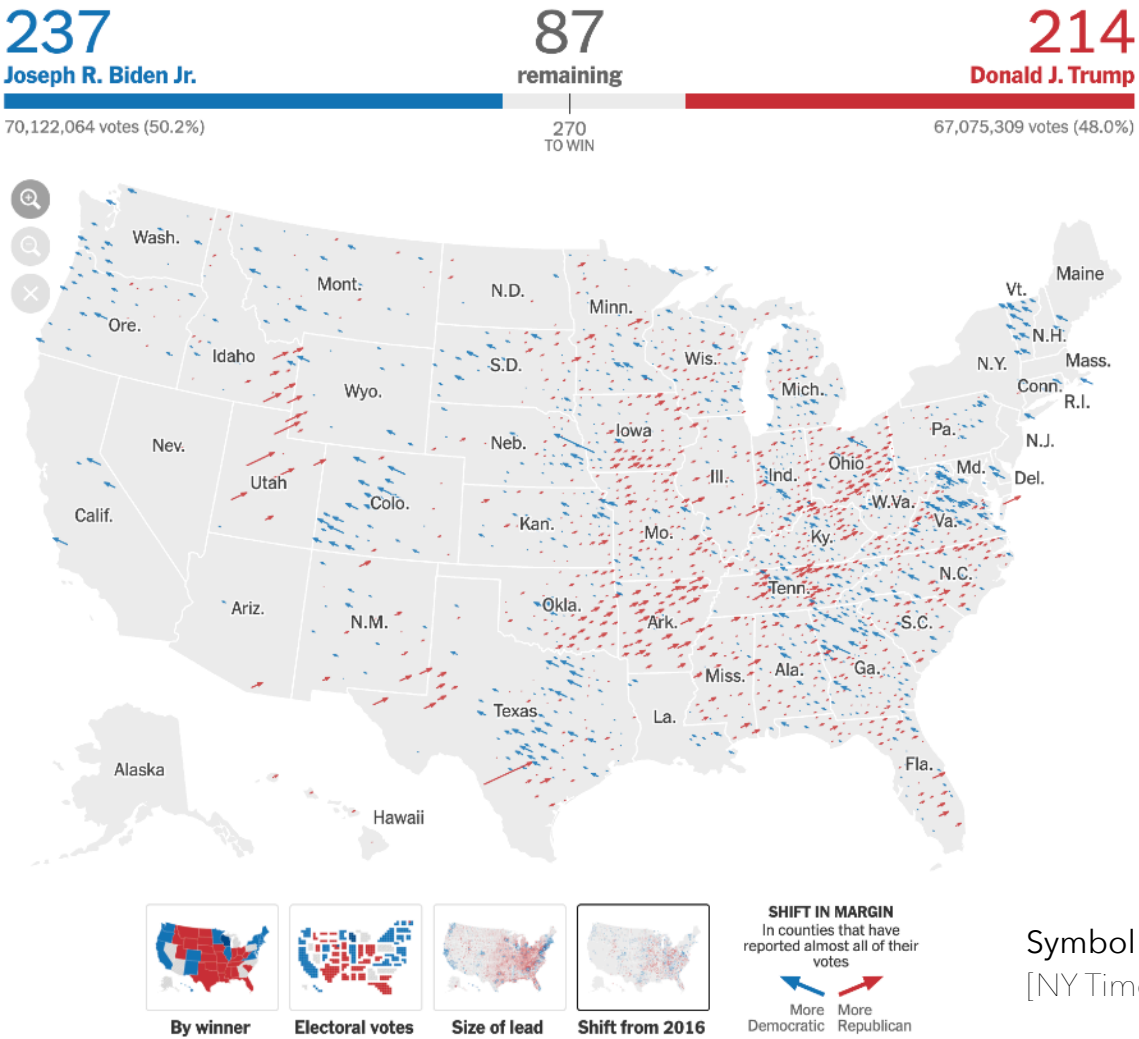

NY Times
2014



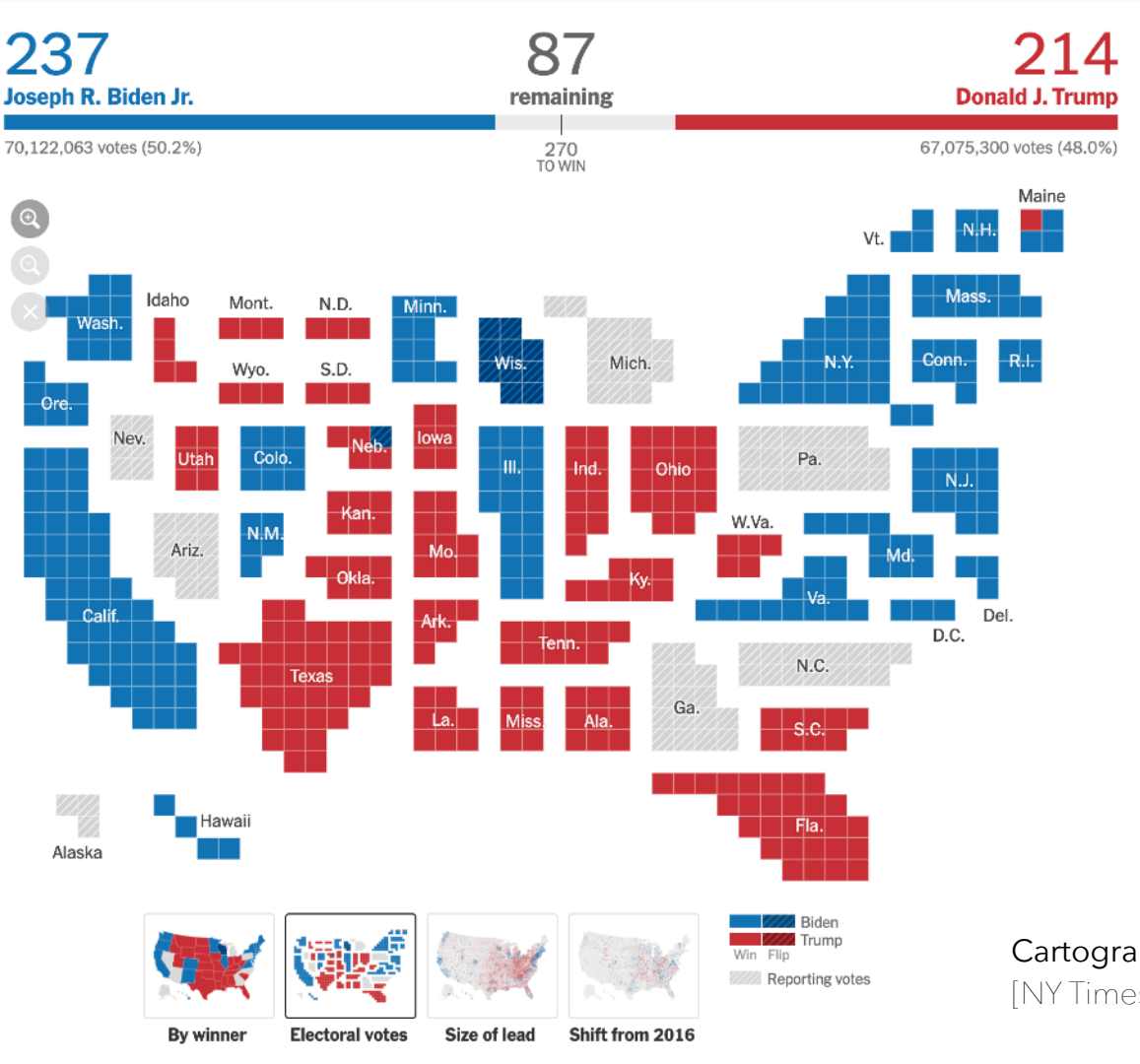

Choropleth Map
[NY Times]



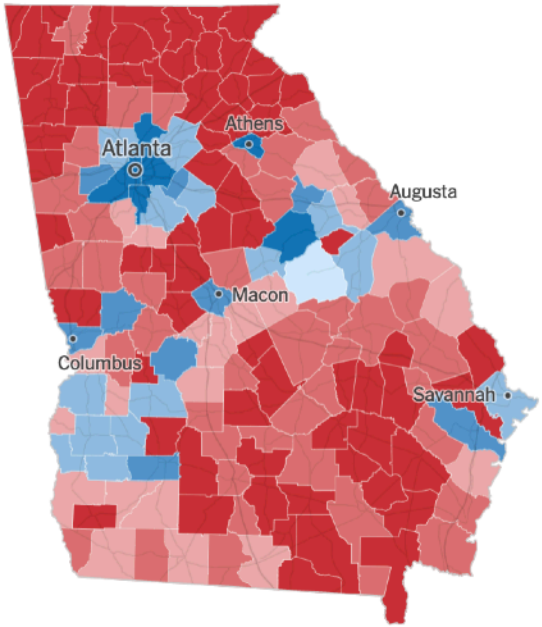

Symbol Map
[NY Times]



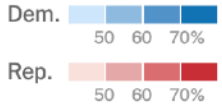

Symbol Map
[NY Times]

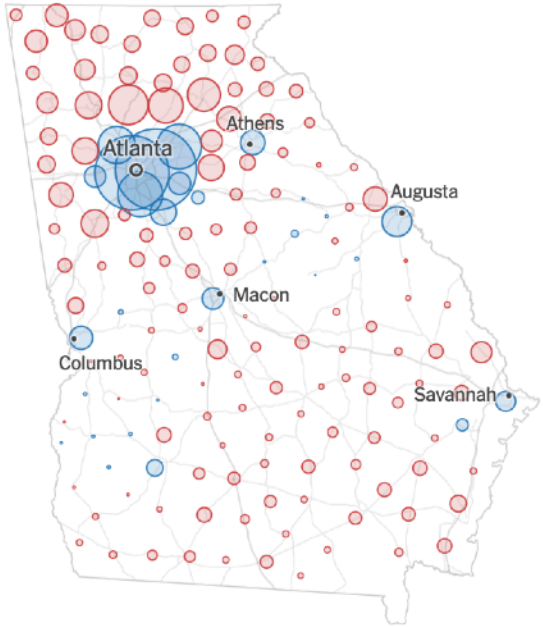



Cartogram
[NY Times]




Choropleth Map
[NY Times]




Symbol Map
[NY Times]

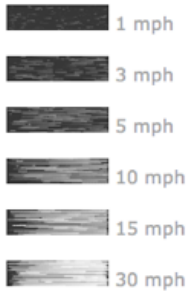
Circle size is proportional to the amount each county's leading candidate is ahead.

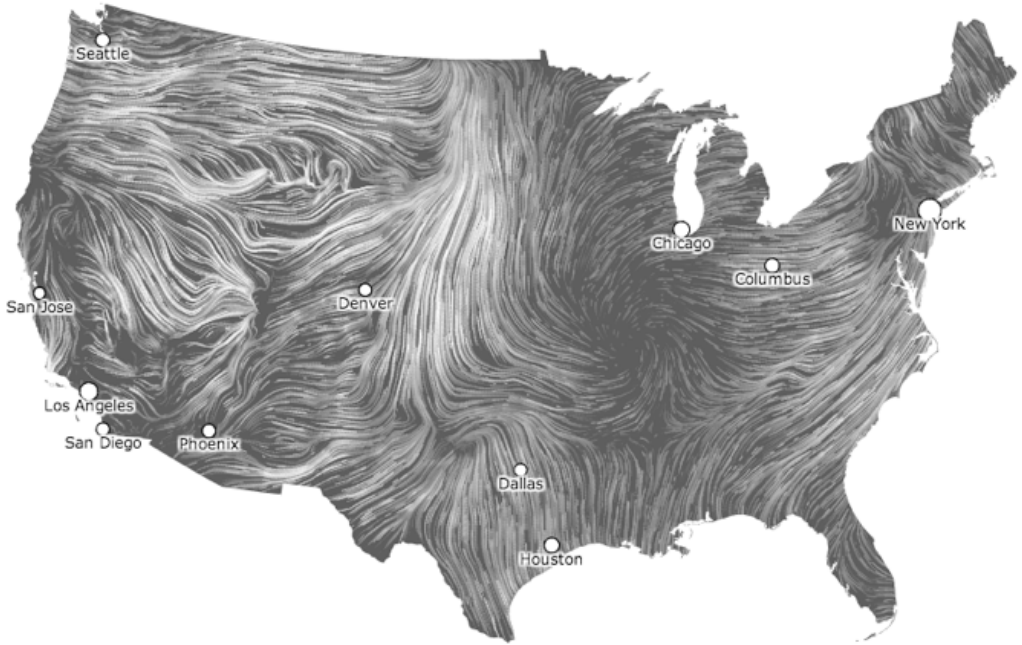
● Trump ● Biden

# wind map

**February 19, 201**
11:55 am EST
(time of forecast download)

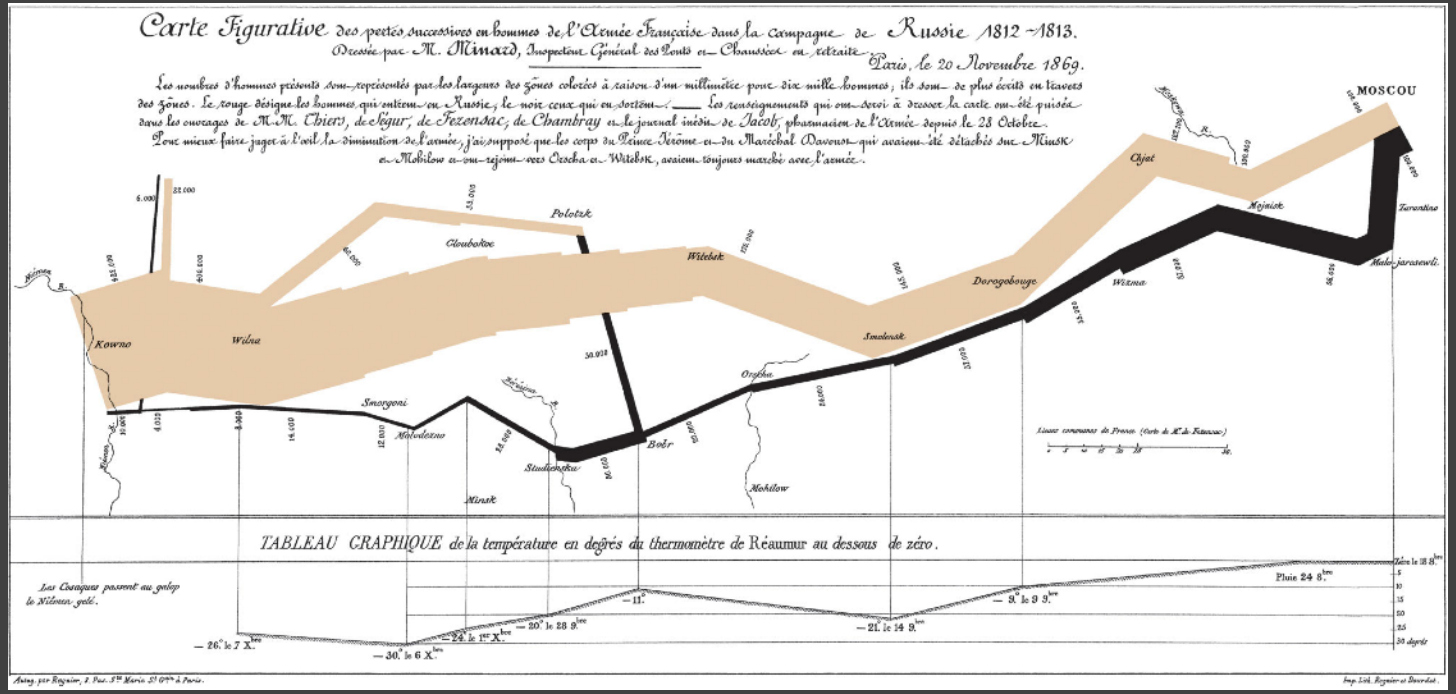
top speed: **35.3 mph**
average: **11.6 mph**

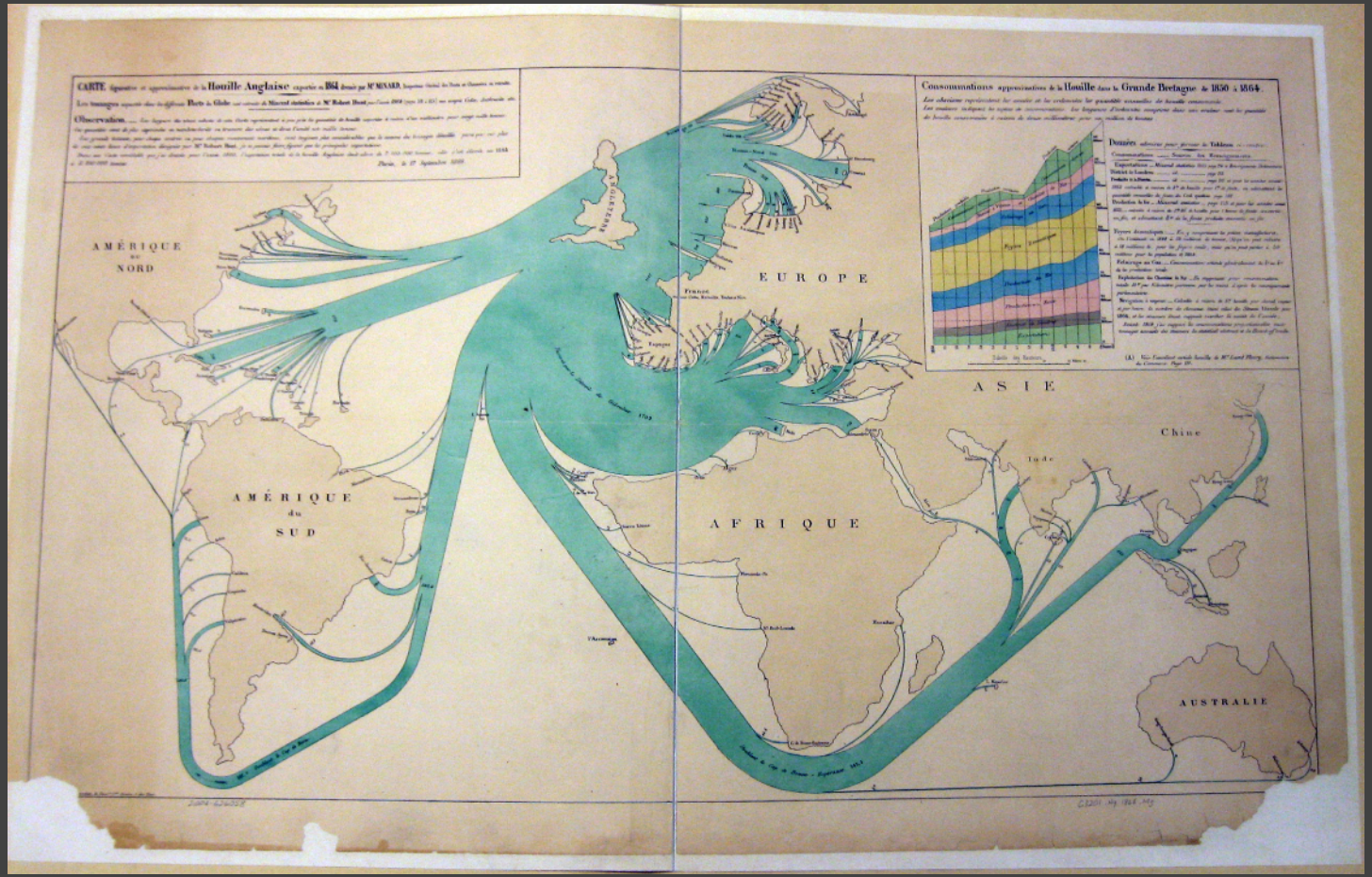



Wattenberg & Viegas

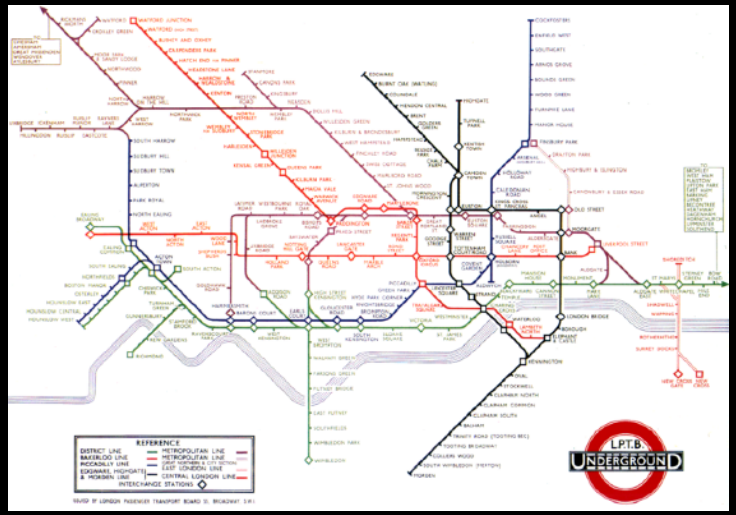
# Minard 1869: Napoleon's march



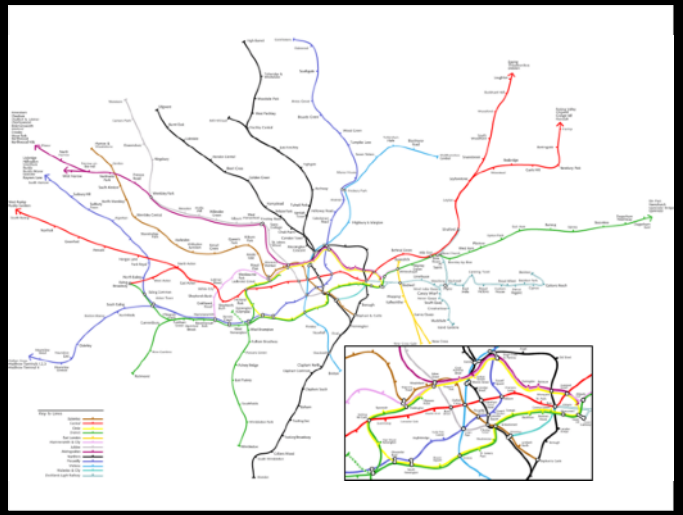



1864 British Coal Exports, Charles Minard


# Beck's London tube diagram




**London Underground** [Beck 33]



**Geographic version of map**

**Principle:** Straighten lines to emphasize stop sequence

Technique used to emphasize/de-emphasize information

# Approaches to Mapping Data

**Symbol Maps** → plot data over a map

**Choropleth Maps** → colored regions

**Heatmaps & Contours** → show densities

**Cartograms** → distort to show quantities

**Flow Maps** → flux across regions

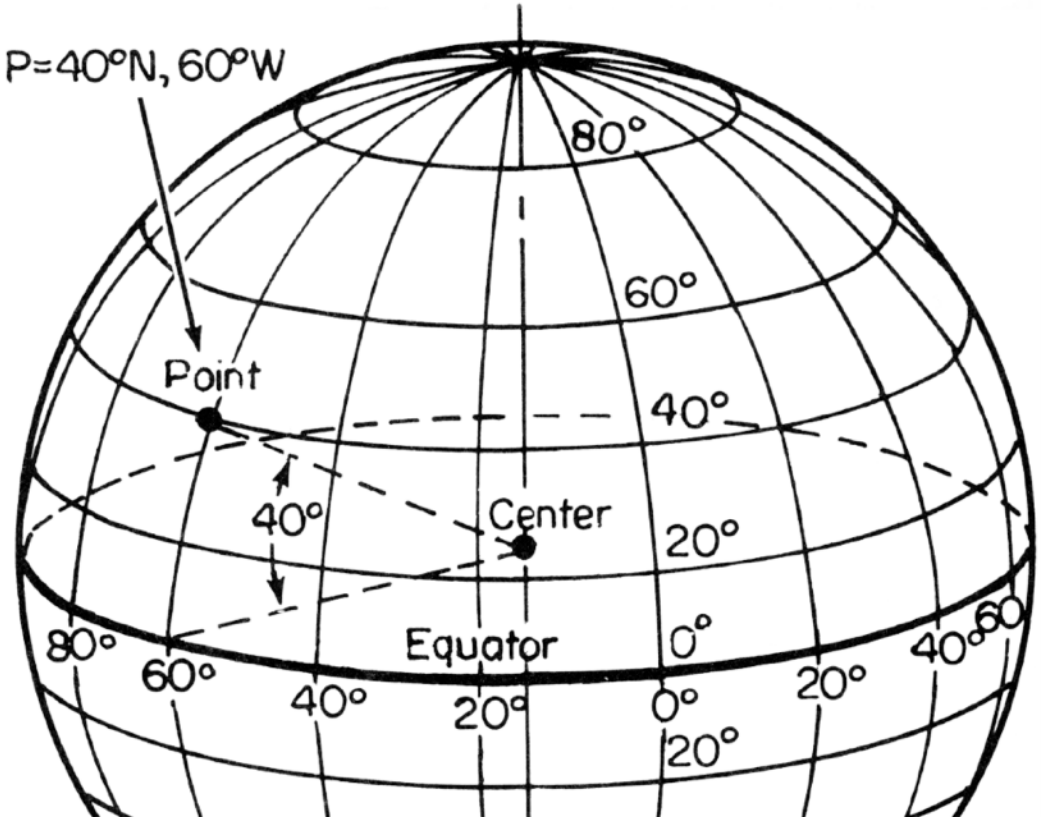
**Generalization** → distort/abstract to aid tasks

# Cartography

The Making of Maps

# Projections

# Latitude, Longitude




A sphere tears
when you flatten it

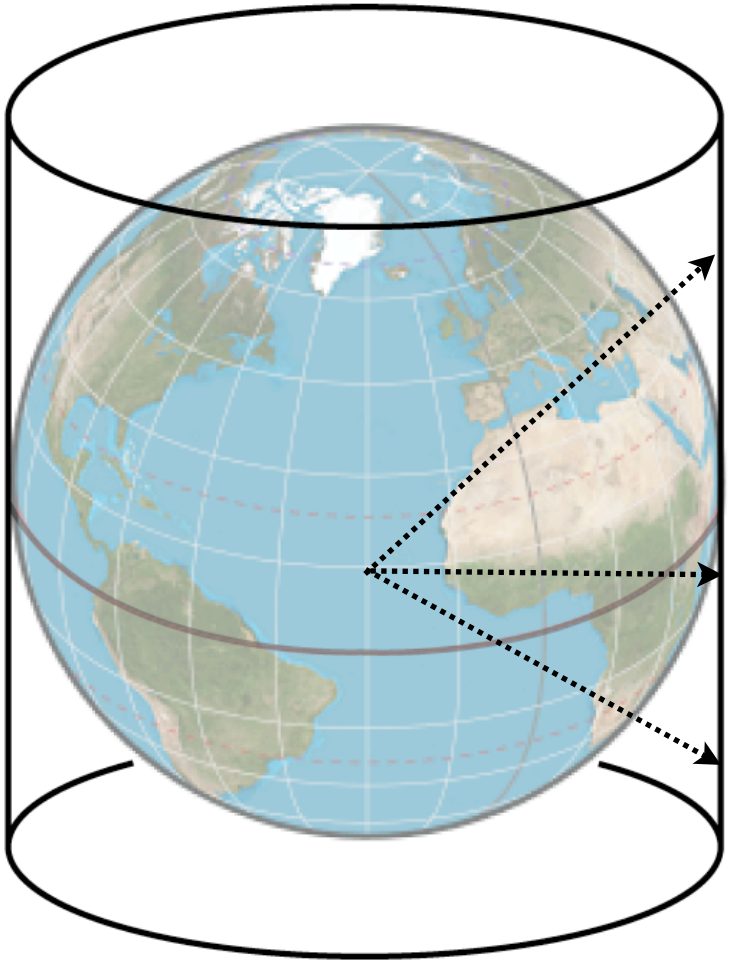
# Projections

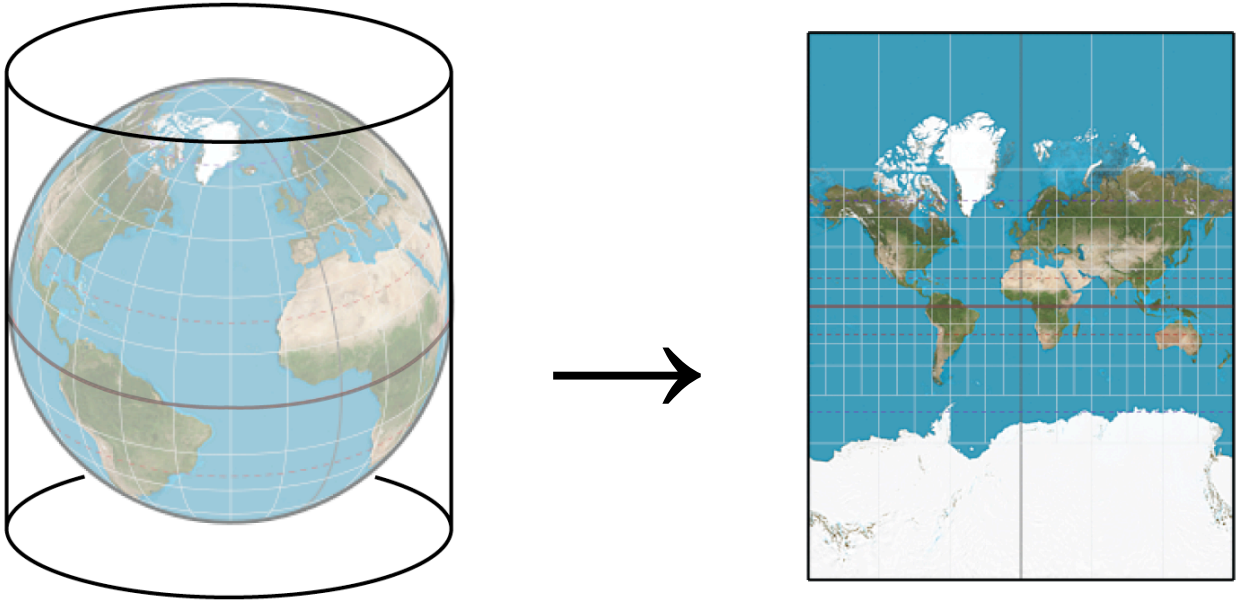
$$f(\phi, \lambda) \rightarrow (x, y)$$

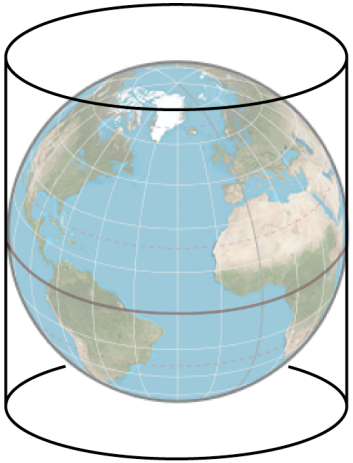
# Projections

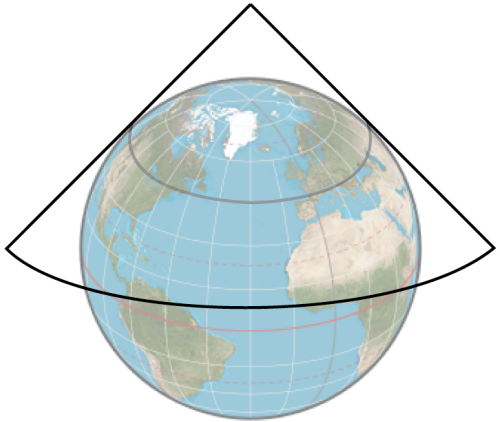
$$f(\phi, \lambda) \leftrightarrow (x, y)$$

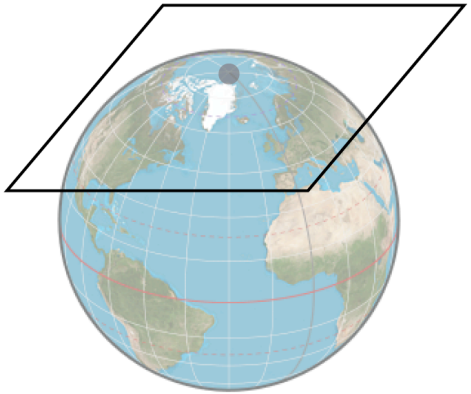
??

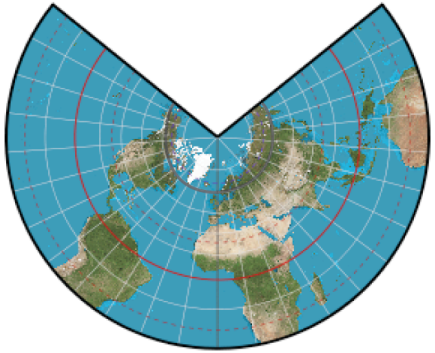


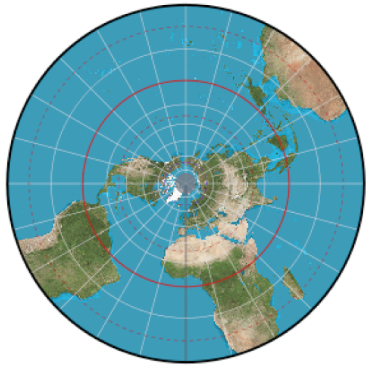












Cylindrical

Conical

Azimuthal

# Exploring Projections…

Type mercator

Scale 140

Yaw 0

Pitch 0

Roll 0

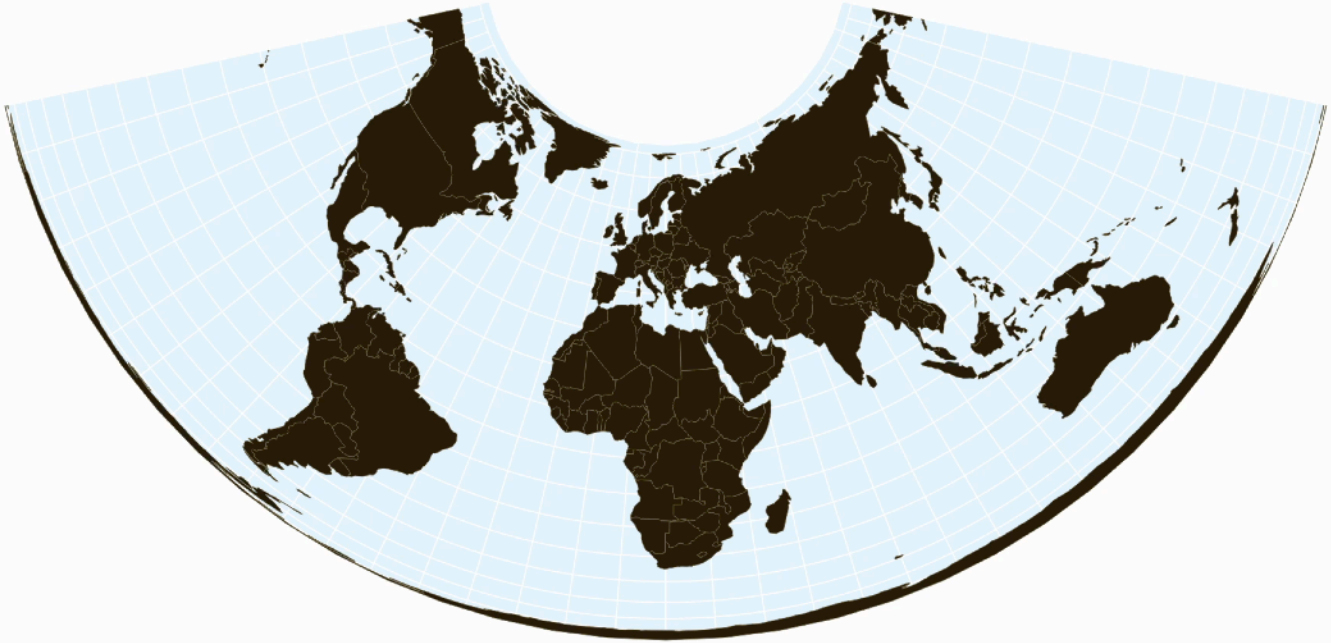
Type mercator

Scale 140

Yaw 0

Pitch 0

Roll 0



| | | |
|---|---|---|
| Type | conicEqualArea | |
| Scale | | 140 |
| Yaw | | -20 |
| Pitch | | 0 |
| Roll | | 0 |

Type orthographic

Scale 140

Yaw 0

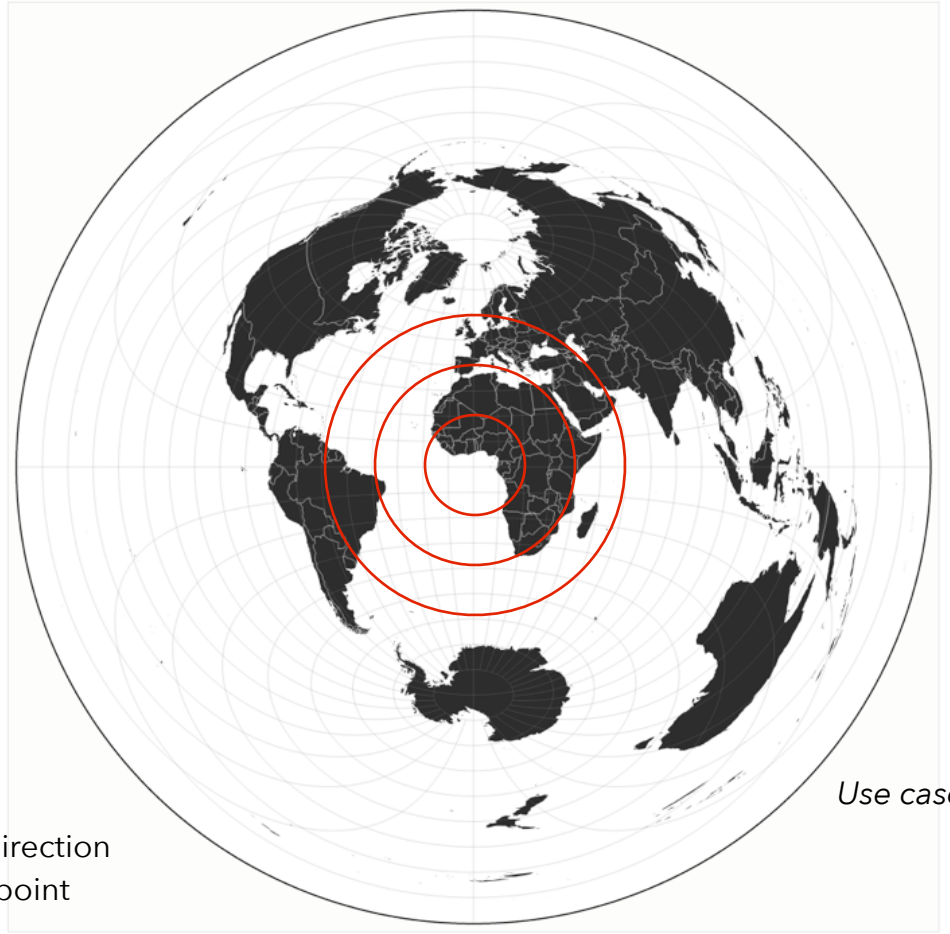
Pitch 0

Roll 0

We can categorize projections by what they preserve…


Distance
Preserve distance / direction from center

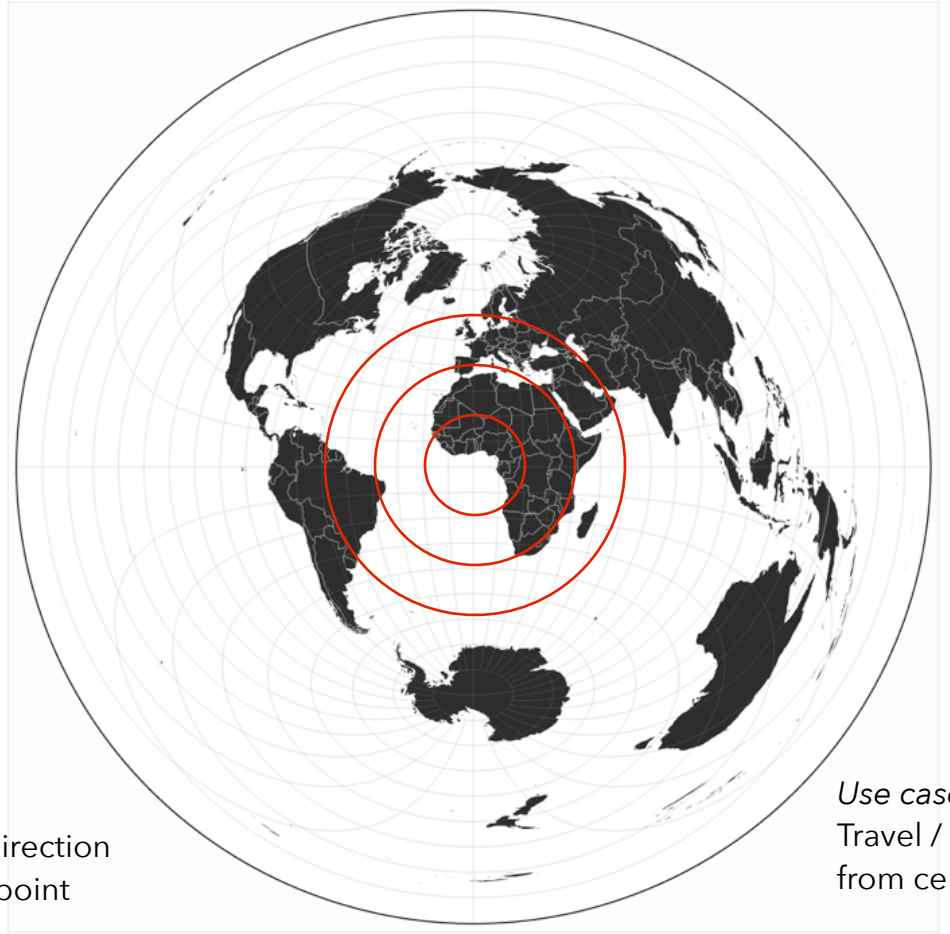
# Azimuthal Equidistant



*Preserves*:
Distance & direction
from center point

*Use cases?*

# Azimuthal Equidistant

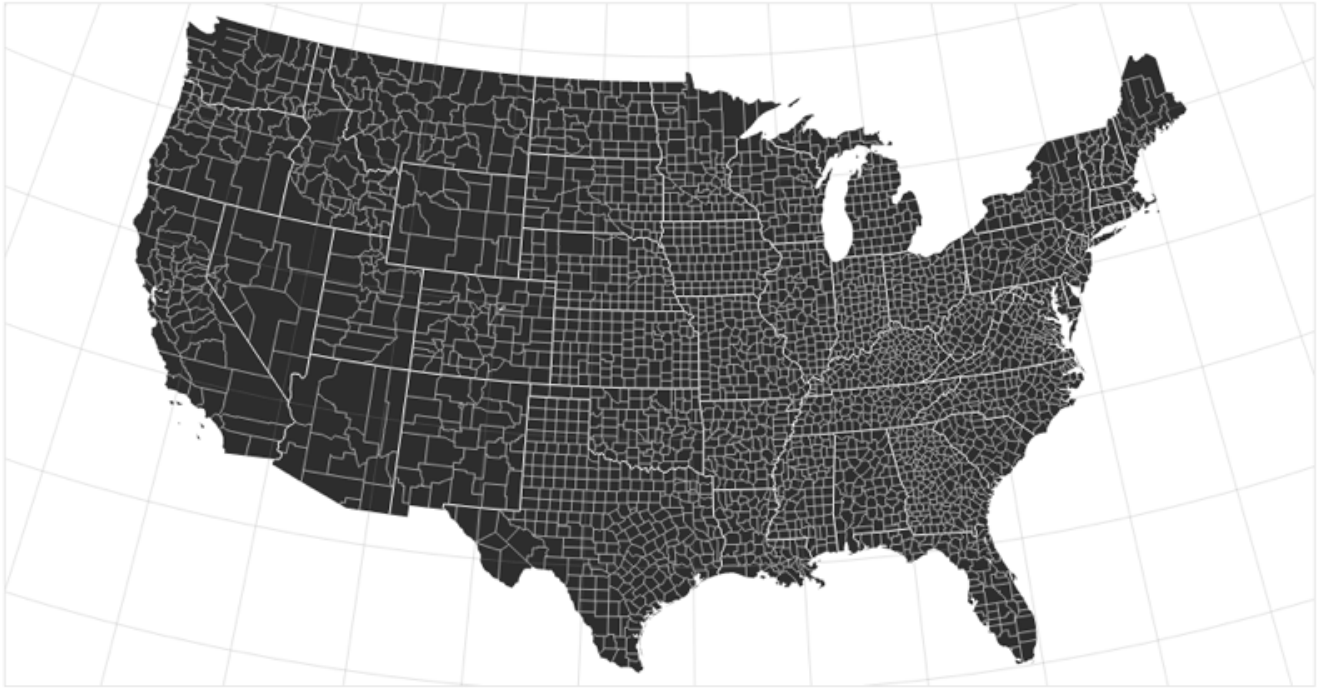


*Preserves*:
Distance & direction
from center point

*Use cases*:
Travel / propagation
from center point


Equal-Area
Preserve proportional areas

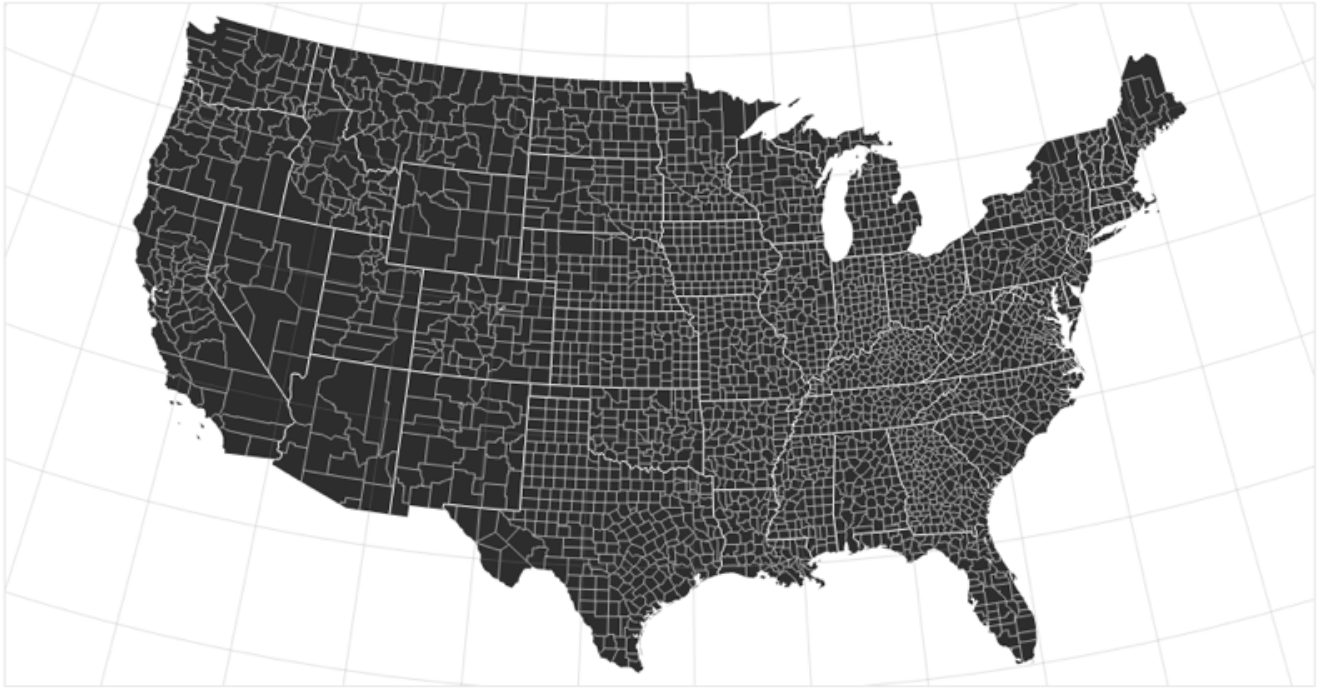
# Albers Equal-Area Conic



*Preserves*: Proportional area of geographic regions

*Use cases?*

# Albers Equal-Area Conic



*Preserves*: Proportional area of geographic regions

*Use cases*: Land surveys, choropleth (shaded) maps


Conformal
Preserve local angles (“shape”)

# Spherical Mercator


*Preserves*:
Compass bearing
as a straight line

*Use cases?*

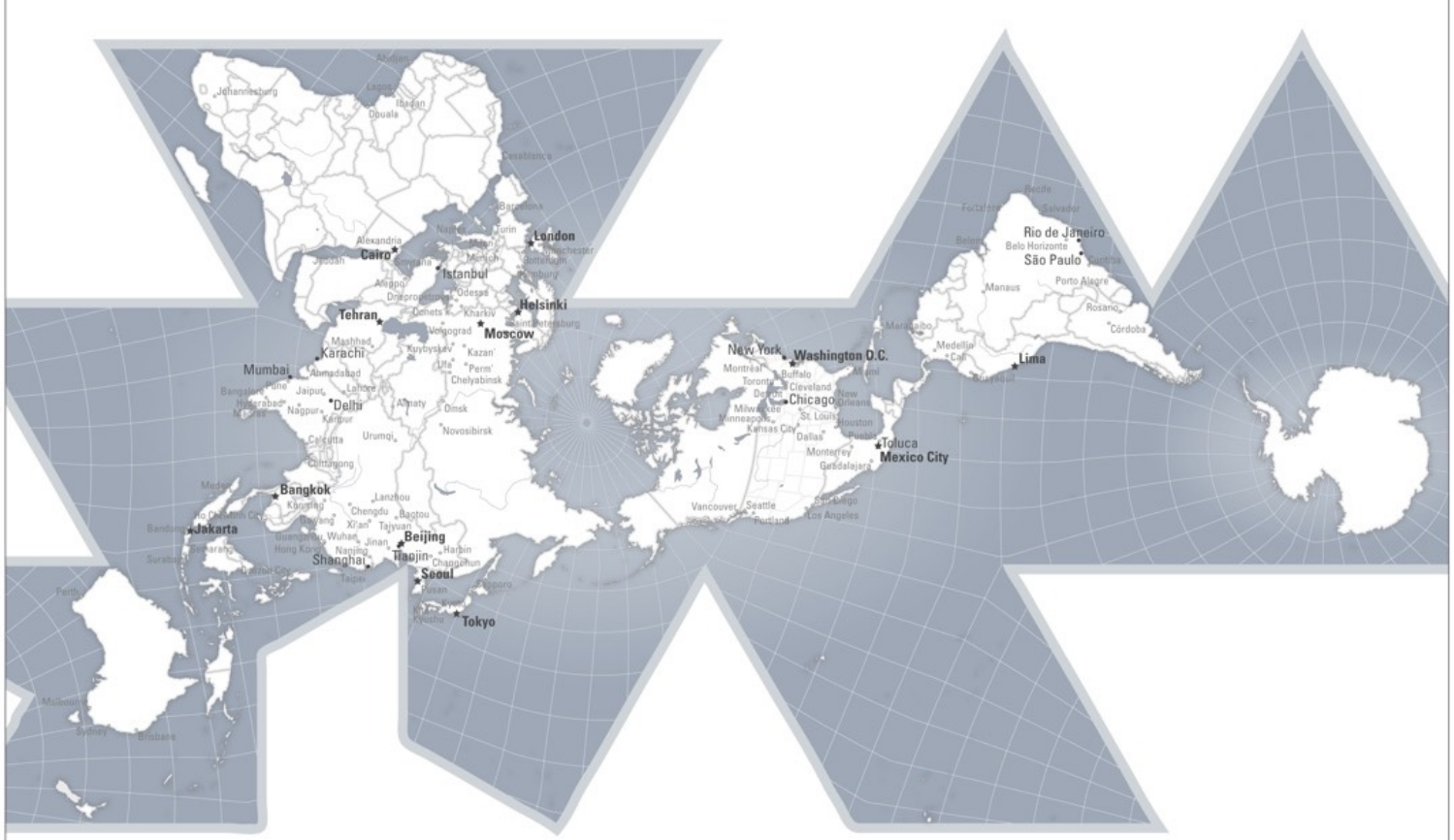
# Spherical Mercator


*Preserves*:
Compass bearing
as a straight line

*Use cases*:
Navigation

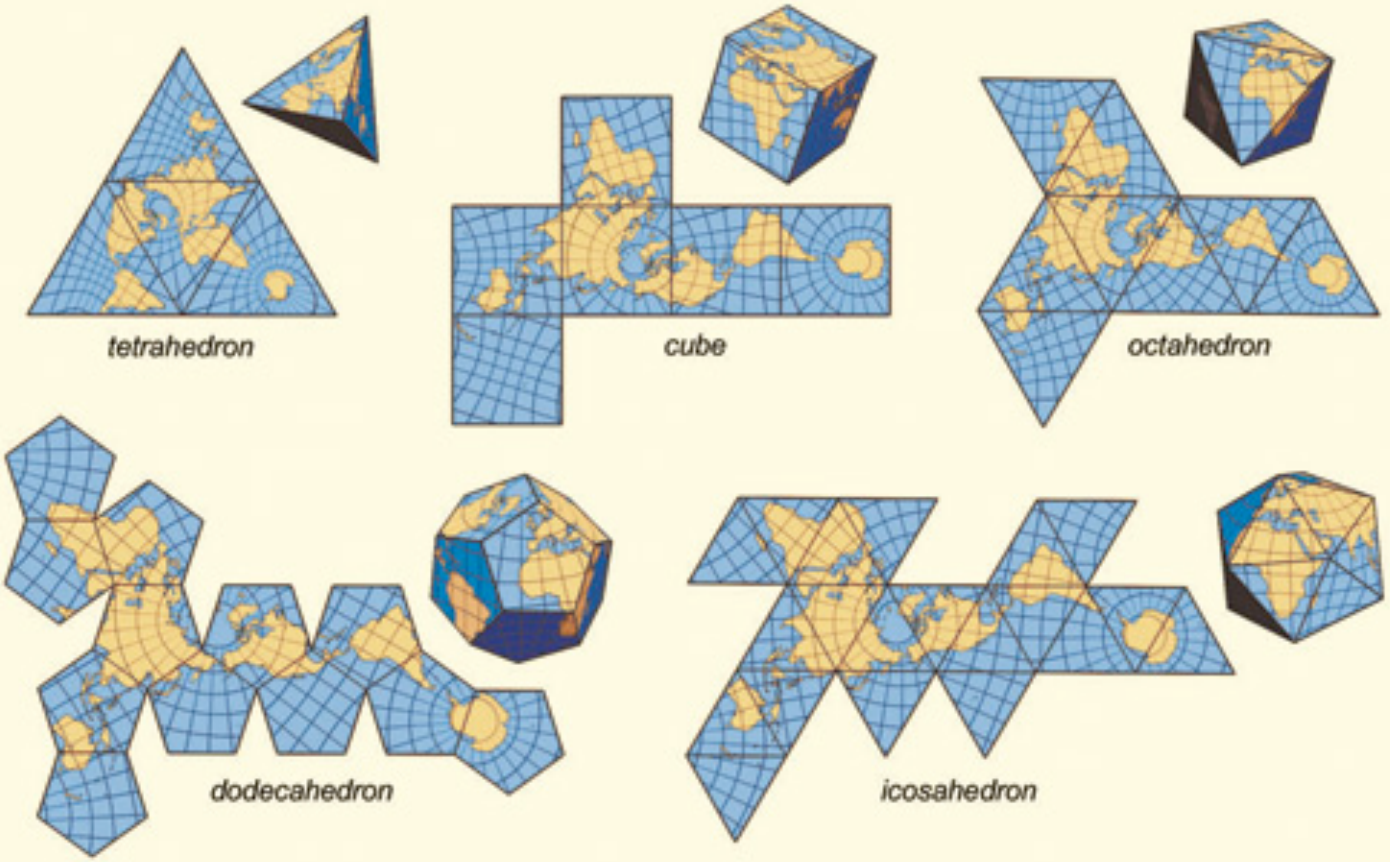
There are interesting ways to tear spheres

# One notable interesting way to tear a sphere




Balances preservation of area and shape.

*Provides different ways of thinking about the world!*

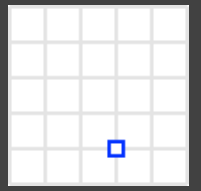

tetrahedron
cube
octahedron
dodecahedron
icosahedron

# Geographic Data Formats

# From Tables to Geometry: Basic Shapes

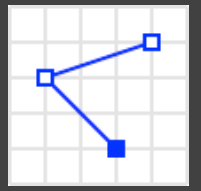
**Point**: An array containing 2D or 3D coords (e.g., `[lon, lat]`)

```
[125.6, 10.1]
```



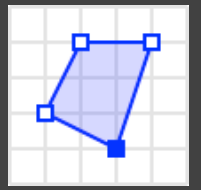
**LineString**: An array of points

```
[[30, 10], [10, 30], [40, 40]]
```



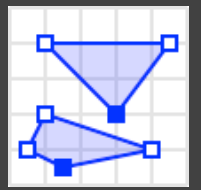
**Polygon**: One or more arrays of points

```
[ [[30, 10], [40, 40], [20, 40], [10, 20], [30, 10]] ]
```



**MultiPolygon**: An array of polygons

```
[
  [ [[30, 20], [45, 40], [10, 40], [30, 20]] ],
  [ [[15, 5], [40, 10], [10, 20], [5, 10], [15, 5]] ]
]
```



# GeoJSON Format

GeoJSON is a standardized JSON format for geometric data.

**Geometry**: An object with a type and a coordinates array

```
{“type”: “Point”, “coordinates”: [125.6, 10.1]}
```

```
{“type”: “Polygon”, “coordinates”: [[[30.0, 20.0], [45.0, 40.0],
[10.0, 40.0], [30.0, 20.0]]]}
```

**Feature**: An object with a geometry and optional named attributes

```
{
  “type”: “Feature”,
  “id”: “optional_id_string”,
  “geometry”: { “type”: “MultiPolygon”, …},
  “properties”: { “attr1”: “foo”, “attr2”: 12863 }
}
```

# GeoJSON Format, Continued

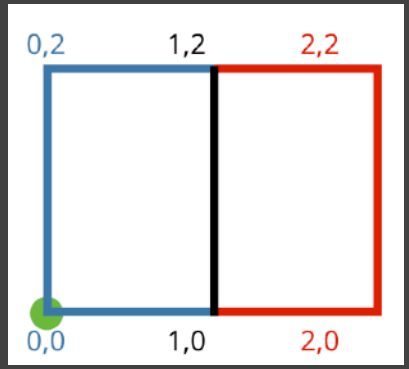
**FeatureCollection**: Top-level GeoJSON file object

```
{
  “type”: “FeatureCollection”,
  “features”: [
    { “type: “Feature”, “geometry”: … },
    { “type: “Feature”, “geometry”: … }
  ]
}
```

Tools like D3, Vega-Lite, and Observable Plot all use GeoJSON as the primary means of representing geographic data. Points values often use `[longitude, latitude]`, but are not required to. For example, pre-projected planar geometries (`[x, y]`) are also supported.

# TopoJSON Format

TopoJSON is a compressed form of GeoJSON that stores *topologies* rather than raw geometries. For example, the border between Colorado and Arizona can be stored only once, then be extracted into separate borders for each state.




Prior to use in visualization, TopoJSON data must be parsed and extracted to geometry data - typically GeoJSON. Vega-Lite includes support to perform this extraction internally.

Otherwise, one can use the topojson-client library:

```
topojson.feature(data, “states") // get feature collection
topojson.mesh(data) // get boundary mesh
```

# Resources

# Software Tools

## Web Tools

**d3-geo**: projections, paths and more
**GeoJSON**: JSON format for geo data
**TopoJSON**: topology -> compressed GeoJSON
**MapShaper**: online editor for map data
**Leaflet**: open-source, customizable map tile system

## Other

**PostGIS**: Postgres DB extensions for geo data
**Mapnik**: Render your own map tiles!

# Data Resources

**Natural Earth Data**
naturalearthdata.com

**OpenStreetMap**
openstreetmap.org

**U.S. Government**
nationalatlas.gov, census.gov, usgs.gov

# Tutorials

## Cartographic Visualization in Vega-Lite

https://observablehq.com/@uwdata/cartographic-visualization

## Command-Line Cartography

https://medium.com/@mbostock/command-line-cartography-part-1-897aa8f8ca2c

## How to Infer Topology

http://bost.ocks.org/mike/topology/

# Questions?