

Data Visualization (CSCI 627/490)

Data

Dr. David Koop

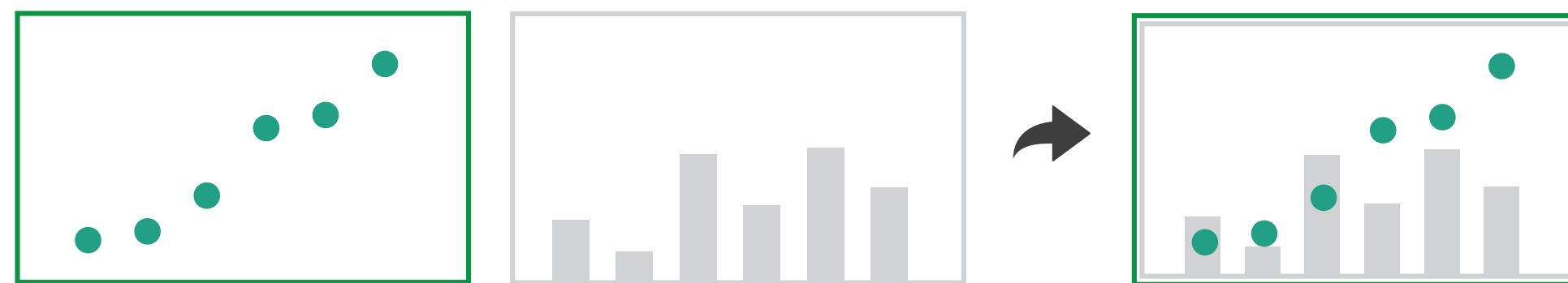
Focus+Context Overview

➔ Embed

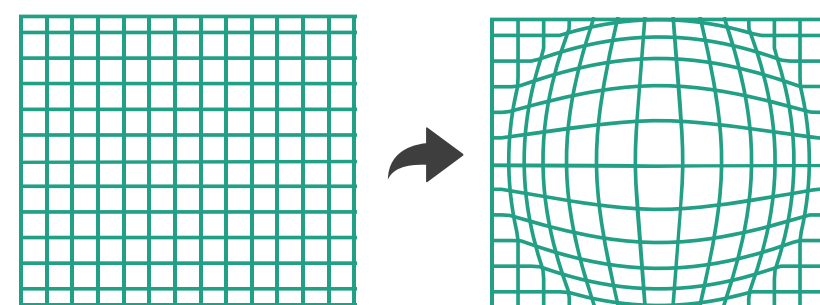
➔ Elide Data



➔ Superimpose Layer



➔ Distort Geometry



Reduce

➔ Filter



➔ Aggregate

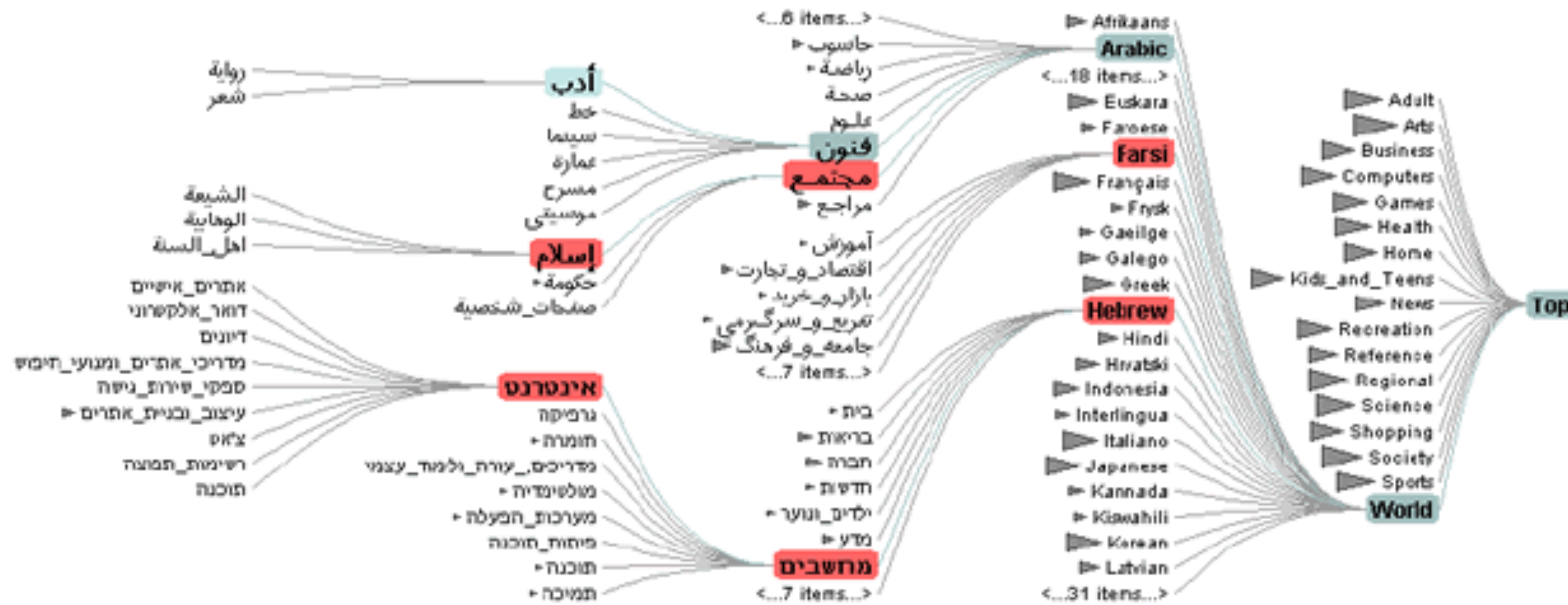


➔ Embed



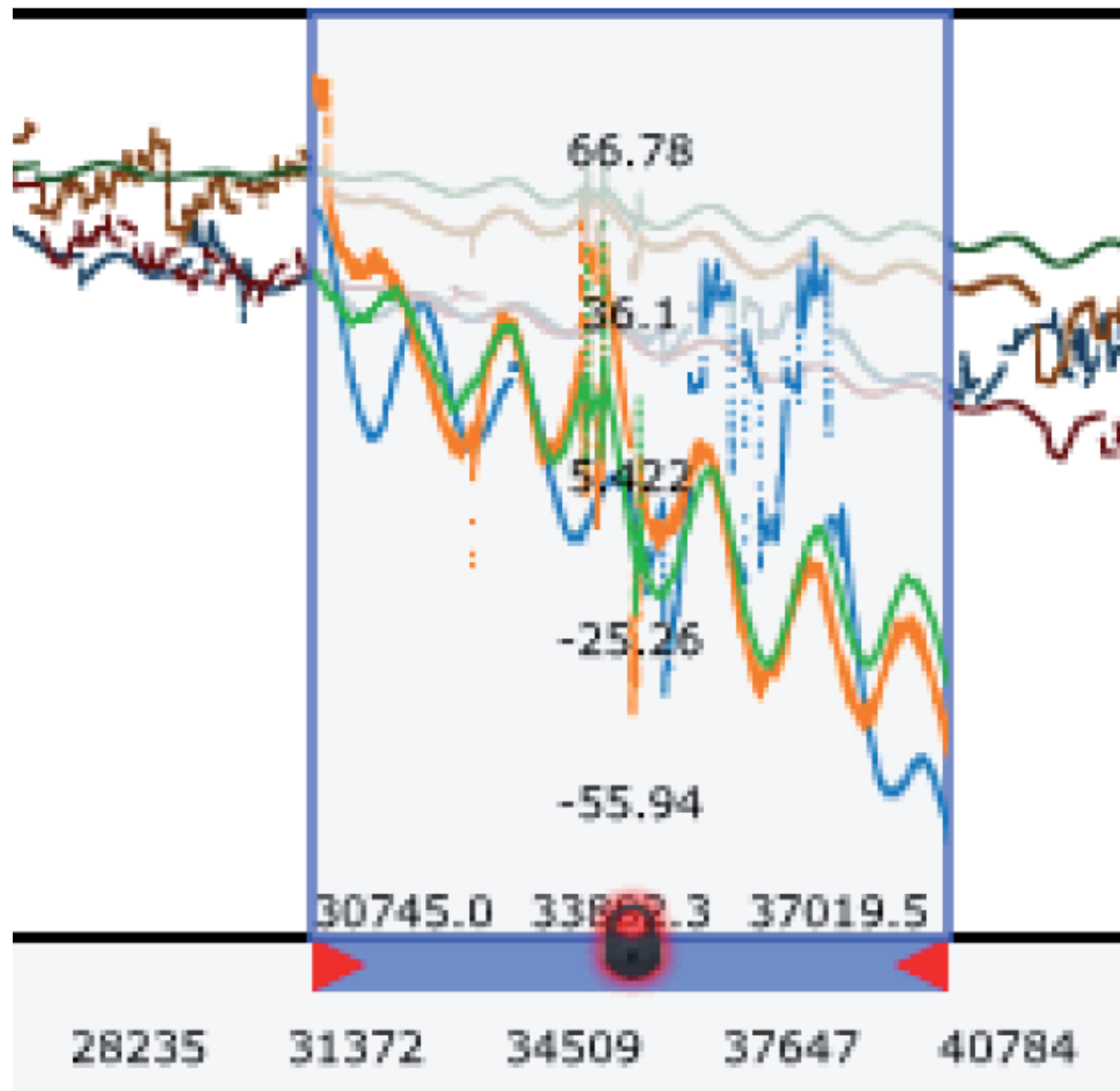
Elision: DOTrees

- Example: 600,000 node tree
 - Multiple foci (from search results or via user selection)
 - Distance computed topologically (levels, not geometric)

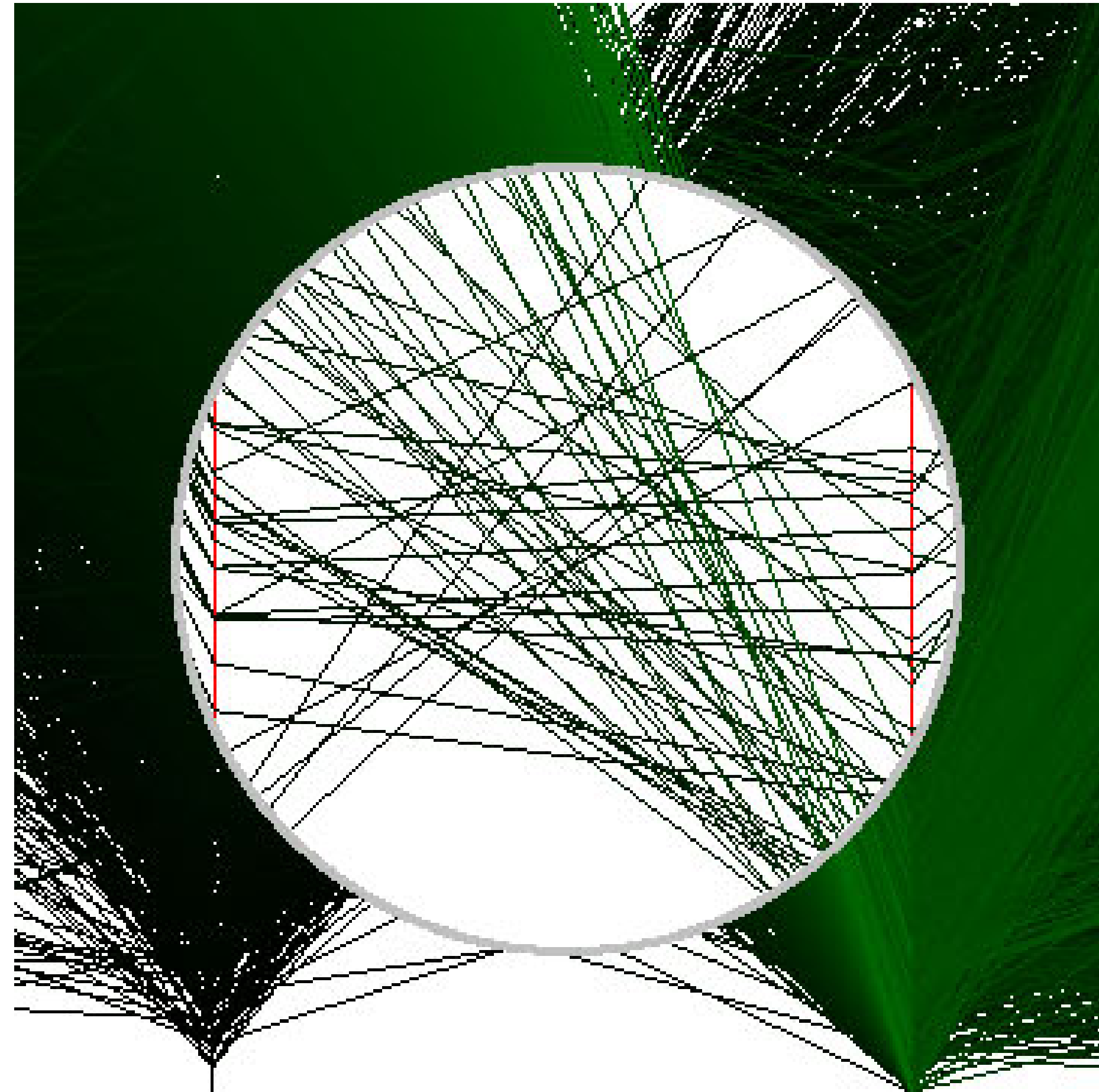


[Heer and Card, 2004]

Superimposition with Interactive Lenses



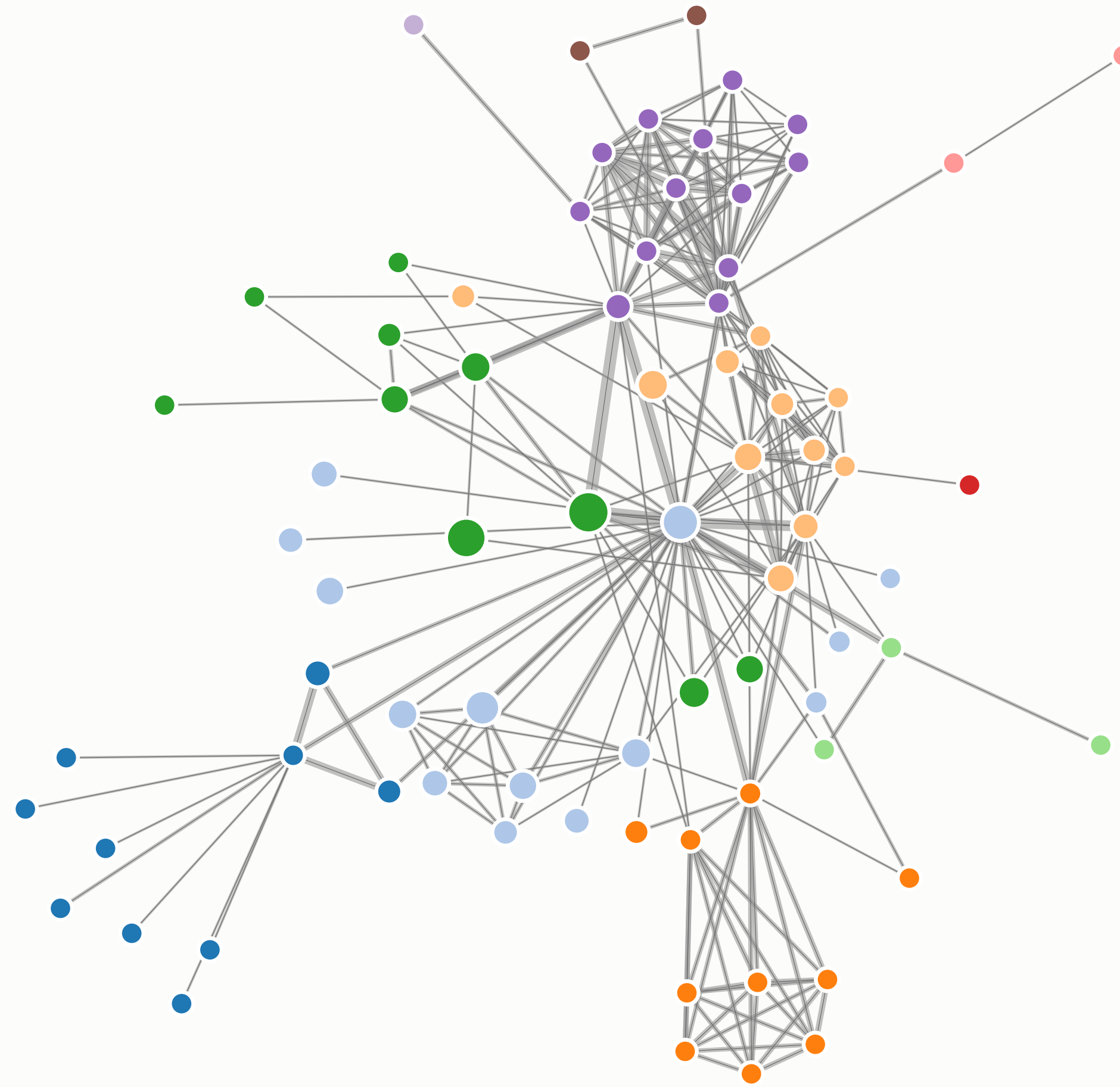
(a) Alteration



(b) Suppression

[ChronoLenses and Sampling Lens in Tominski et al., 2014]

Distortion

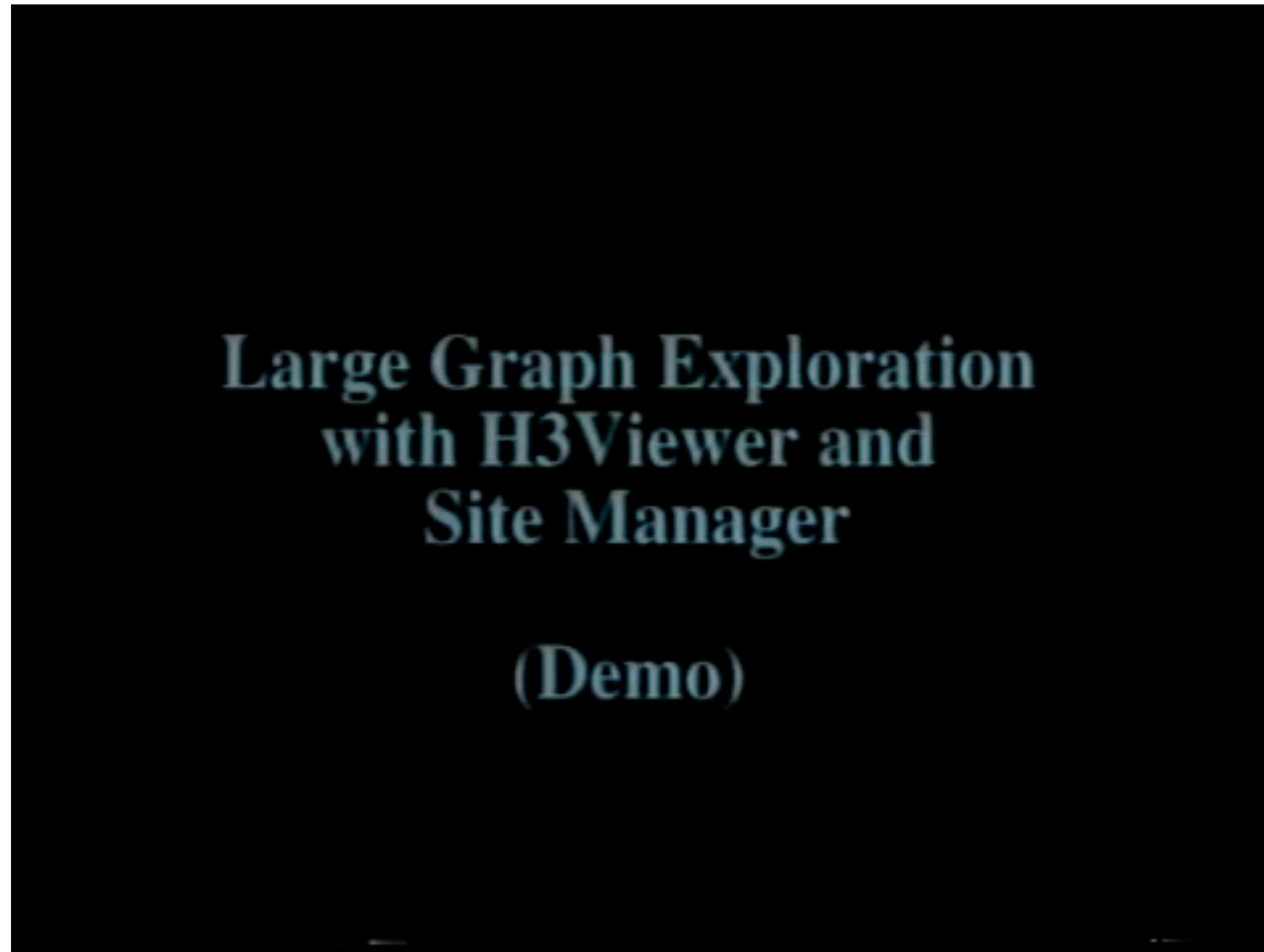


[M. Bostock]

Distortion Concerns

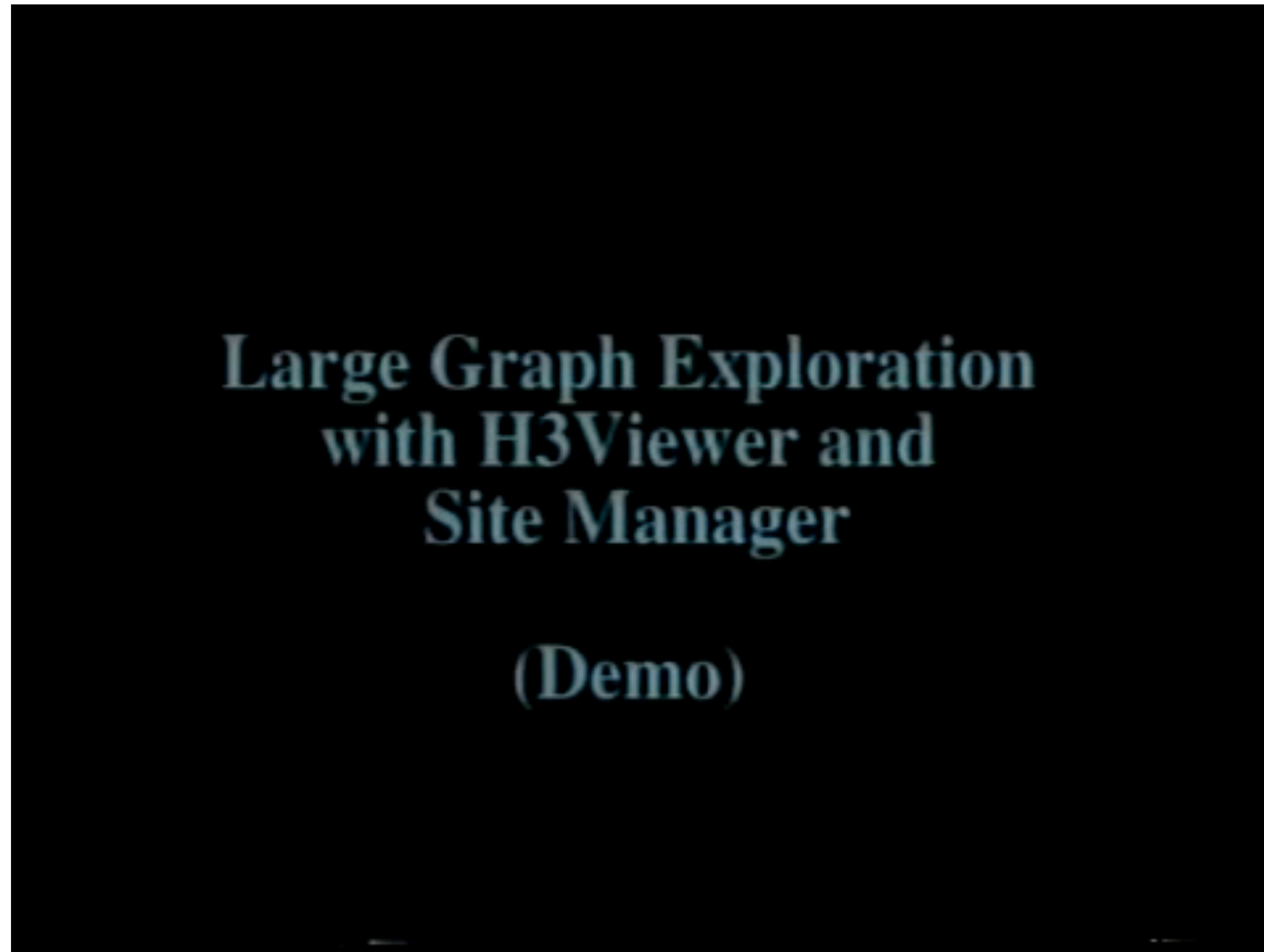
- Distance and length judgments are **harder**
 - Example: Mac OS X Dock with Magnification
 - Spatial position of items changes as the focus changes
- Node-link diagrams not an issue... why?
- Users have to be made aware of distortion
 - Back to scatterplot with distortion example
 - Lenses or shading give clues to users
- **Object constancy**: understanding when two views show the same object
 - What happens under distortion?
 - 3D Perspective is distortion... but we are well-trained for that
- Think about **what** is being shown (filtering) and method (fisheye)

H3 Layout



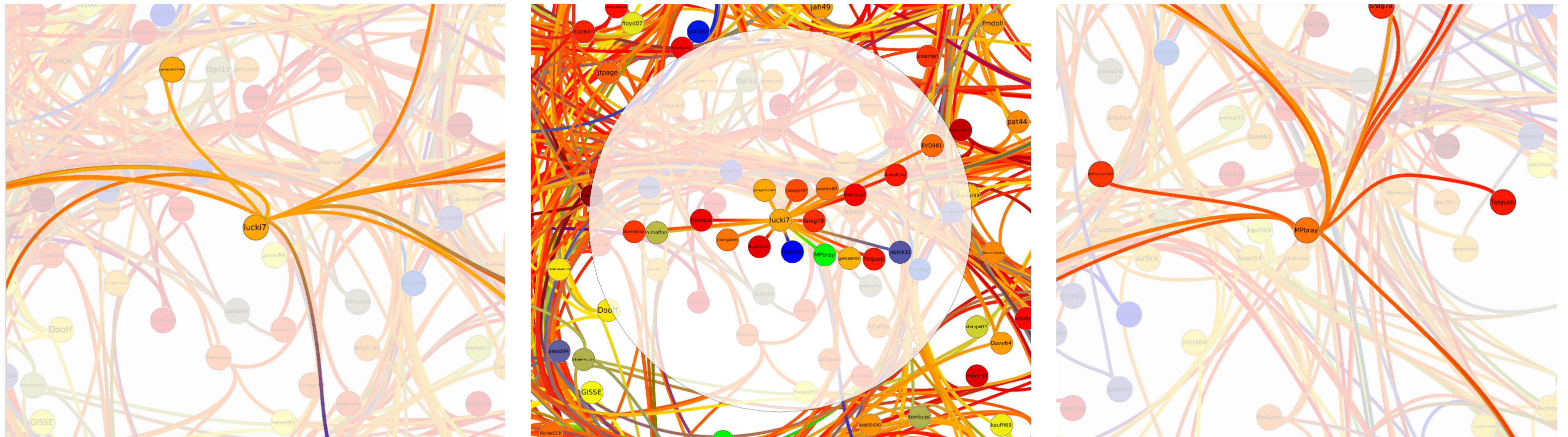
[T. Munzner, 1998]

H3 Layout



[T. Munzner, 1998]

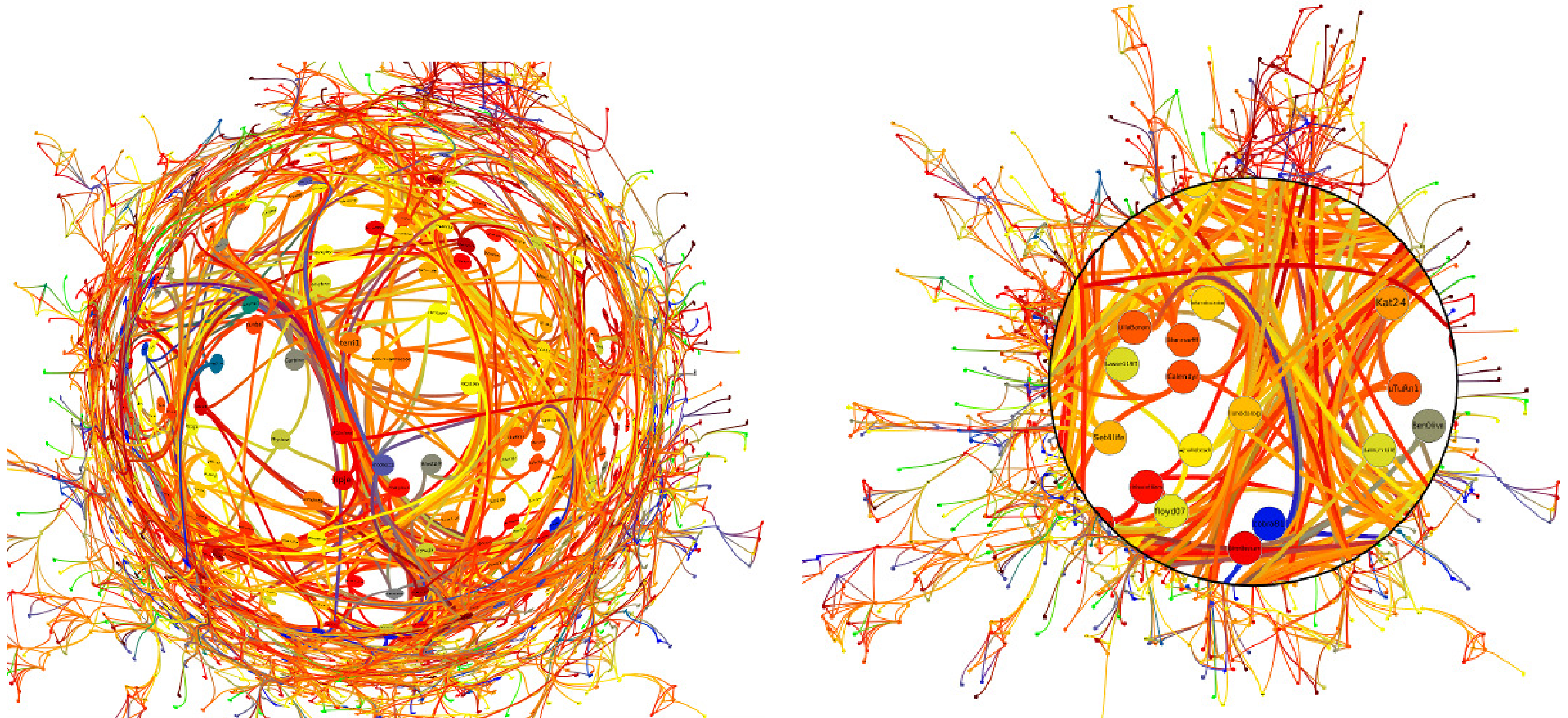
Focus+Context in Network Exploration



(a) Bring (step 1) – Selecting a node fades out all graph elements but the node neighborhood. (b) Bring (step 2) – Neighbor nodes are pulled close to the selected node. (c) Go – After selecting a neighbor (the green node in Fig. 4(b)), a short animation brings the focus towards a new neighborhood.

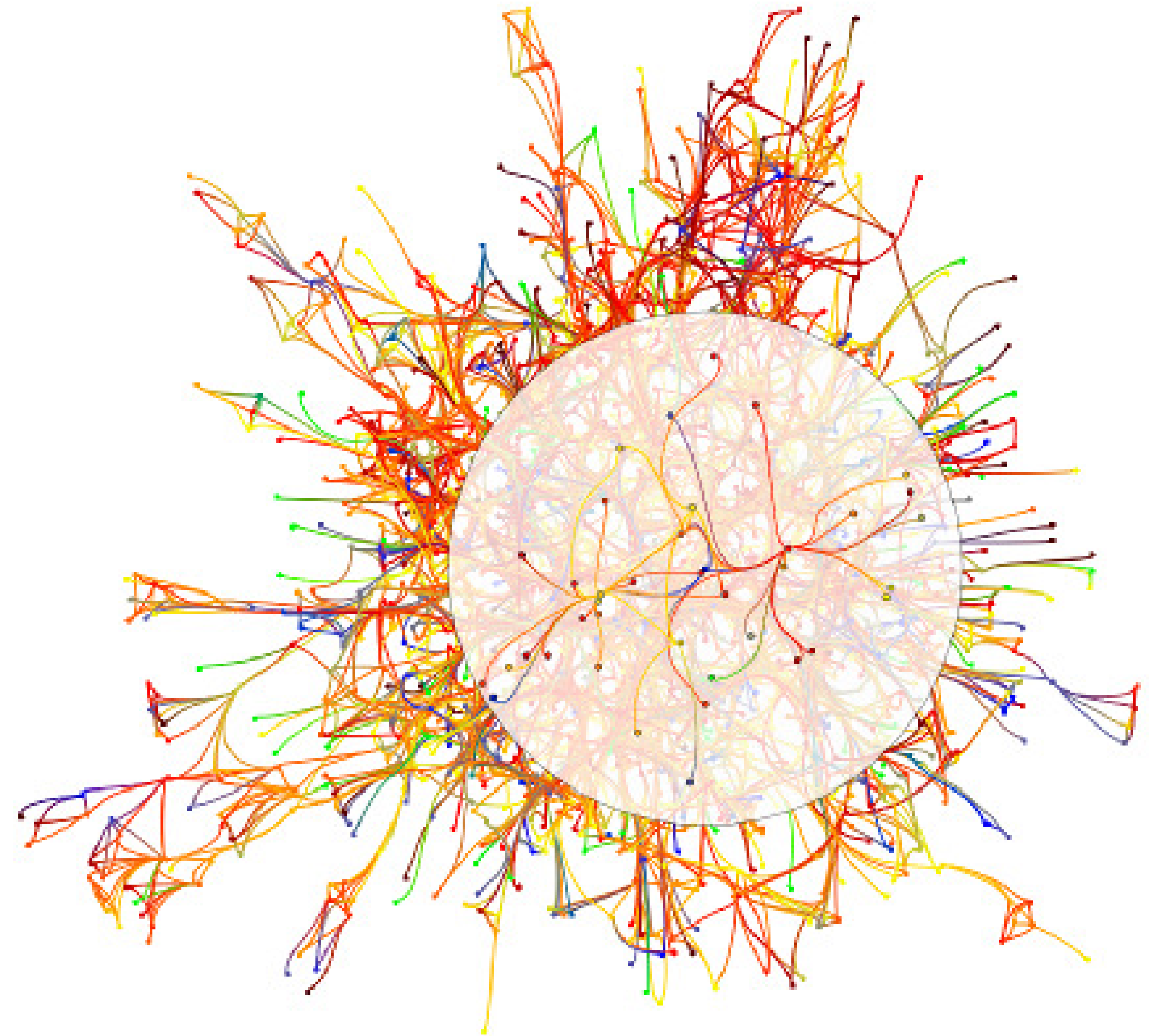
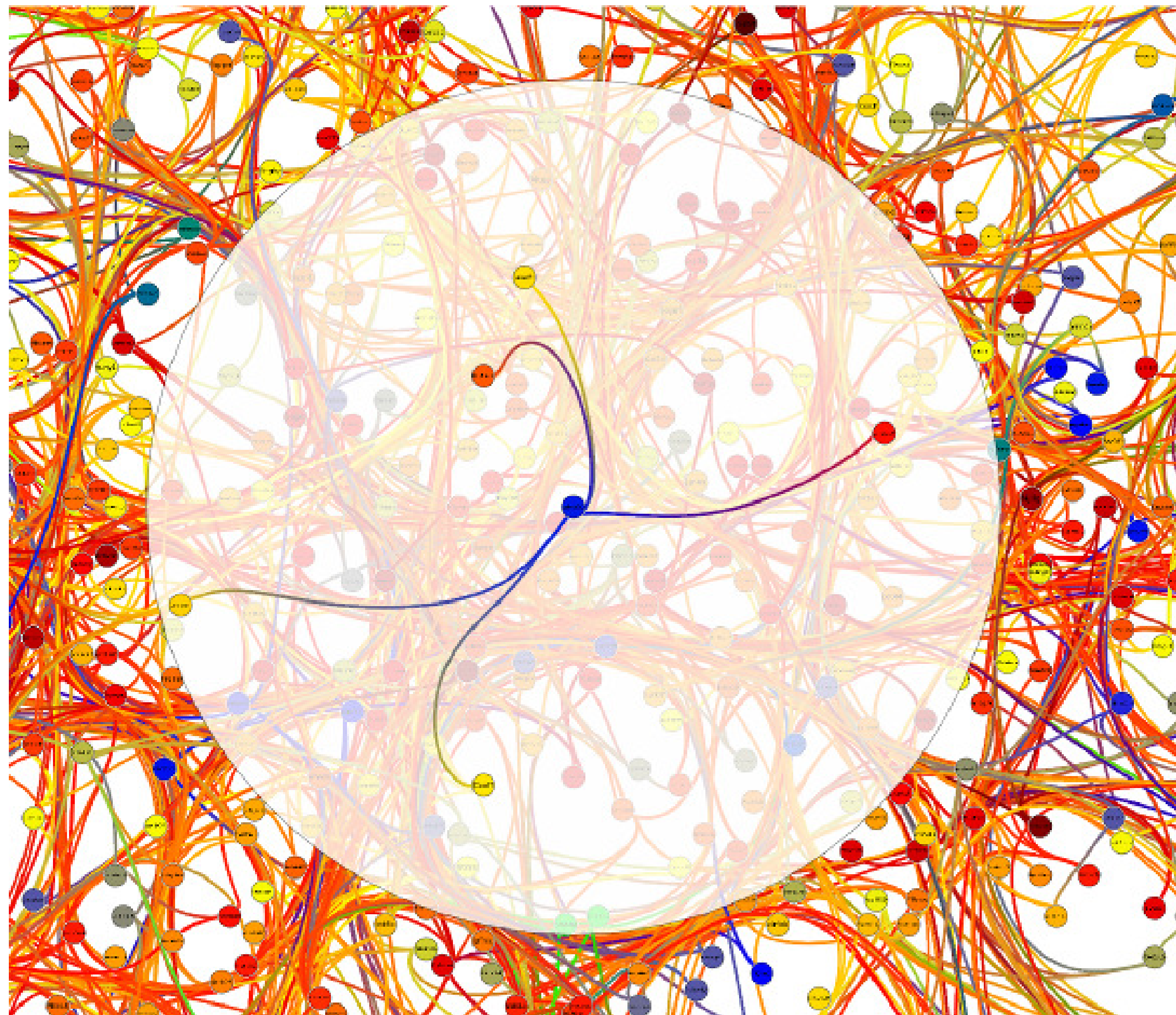
[Lambert et al., 2010]

Focus+Context in Network Exploration



[Lambert et al., 2010]

Focus+Context in Network Exploration



[Lambert et al., 2010]

Upcoming Work

- Assignment 5: Last Assignment
- Project: Continue working on projects
 - Reading Data: [Link](#)
 - Presentations: last week of class
 - Final Report: due at the end of the semester
- Final Exam

Data

Data Wrangling

- Problem 1: Visualizations need data
- Solution: The Web!
- Problem 2: Data has extra information I don't need
- Solution: Filter it
- Problem 3: Data is dirty
- Solution: Clean it up
- Problem 4: Data isn't in the same place
- Solution: Combine data from different sources
- Problem 5: Data isn't structured correctly
- Solution: Reorder, map, and nest it

Hosting data

- github.com
- gist.github.com
- figshare.com
- myjson.com
- observablehq.com
- Other services

Cross-origin resource sharing (CORS)

- Restricts where data can be loaded from
- If developing locally, can
 - Run a web server locally (`python -m http.server` or npm's `http-server`)
 - Put the data on a website (like github), make sure to use **raw** URLs
- If loading JavaScript, this sometimes requires more help
 - <https://www.jsdelivr.com/?docs=gh>

Why JavaScript?

- Python and R have great support for this sort of processing
- Data comes from the Web, want to put visualizations on the Web
- Sometimes unnecessary to download, process, and upload!
- More tools are helping JavaScript become a better language

JavaScript Data Wrangling Resources

- Latest version: <https://observablehq.com/@berkeleyvis/learn-js-data>
- My old version: <https://observablehq.com/@dakoop/learn-js-data>
- Based on <http://learnjsdata.com/>
- Good coverage of data wrangling using JavaScript

Comma Separated Values (CSV)

- File structure:

```
cities.csv:
```

```
city,state,population,land area
seattle,WA,652405,83.9
new york,NY,8405837,302.6
boston,MA,645966,48.3
kansas city,MO,467007,315.0
```

- Loading using D3:

```
d3.csv("/data/cities.csv").then(function(data) {
  console.log(data[0]);
});
```

- Result:

```
=> {city: "seattle", state: "WA", population: 652405, land area: 83.9}
```

- Values are strings! Convert to numbers via the unary + operator:

- `d.population => "652405"`
- `+d.population => 652405`

[<http://learnjsdata.com>]

Tab Separated Values (TSV)

- File structure:

`animals.tsv:`

<code>name</code>	<code>type</code>	<code>avg_weight</code>	
<code>tiger</code>	<code>mammal</code>	<code>260</code>	
<code>hippo</code>	<code>mammal</code>	<code>3400</code>	
<code>komodo</code>	<code>dragon</code>	<code>reptile</code>	<code>150</code>

- Loading using D3:

```
d3.tsv("/data/animals.tsv").then(function(data) {  
  console.log(data[0]);  
});
```

- Result:

```
=> {name: "tiger", type: "mammal", avg_weight: "260"}
```

- Can also have other delimiters (e.g. '|', ':')

[<http://learnjsdata.com>]

JavaScript Object Notation (JSON)

- File Structure:

```
employees.json:  
[  
  {"name": "Andy Hunt",  
   "title": "Big Boss",  
   "age": 68,  
   "bonus": true  
},  
  {"name": "Charles Mack",  
   "title": "Jr Dev",  
   "age": 24,  
   "bonus": false  
}  
]
```

- Loading using D3:

```
d3.json("/data/employees.json").then(function(data) {  
  console.log(data[0]);  
});
```

- Result:

```
=> {name: "Andy Hunt", title: "Big Boss", age: 68, bonus: true}
```

[<http://learnjsdata.com>]

Parquet Files

- More efficient, well-typed storage, but not human-readable
- How to read:
 - Via Apache Arrow and parquet-wasm
 - Using DuckDB-Wasm (easier)
 - ```
client = {
 const client = await DuckDBClient.of();
 await client.sql`CREATE VIEW details AS SELECT * FROM
read_parquet("https://raw.githubusercontent.com/dakoop/
tmdb-tv-shows/refs/heads/main/tv-details.parquet");`;
 return client;
}
```

# Loading Multiple Files

---

- Use Promise.all to load multiple files and then process them all

```
Promise.all([d3.csv("/data/cities.csv"),
 d3.tsv("/data/animals.tsv")])
 .then(analyze);

function analyze(data) {
 cities = data[0]; animals = data[1];

 console.log(cities[0]);
 console.log(animals[0]);
}
=> {city: "seattle", state: "WA", population: "652405", land area: "83.9"}
{name: "tiger", type: "mammal", avg_weight: "260"}
```

[<http://learnjsdata.com>]

# Filtering Data

---

- Often useful to filter data before loading into a visualization
- Already seen this with array functions
- ```
data = [{name: "Two-Egg Breakfast", price: 12.39},  
        {name: "Hash Brown", price: 1.99},  
        {name: "Eggs Benedict", price: 11.29},  
        {name: "Coffee", price: 2.49}]  
data.filter(d => d.price < 5)
```

-

Combining Data

- Suppose given products and brands
- Brands have an `id` and products have a `brand_id` that matches a brand
- Want to join these two datasets together
 - `Product.brand_id => Brand.id`
- Use a nested `forEach/filter`
- Use a native join command

[<http://learnjsdata.com>]

Summarizing Data

- d3 has min, max, and extent functions of the form
 - 1st argument: dataset
 - 2nd argument: accessor function

- Example:

```
var landExtent = d3.extent(data, function(d) { return d.land_area; });  
console.log(landExtent);  
=> [48.3, 315]
```

- Summary statistics, e.g. mean, median, deviation → same format

- Median Example:

```
var landMed = d3.median(data, function(d) { return d.land_area; });  
console.log(landMed);  
=> 193.25
```

[<http://learnjsdata.com>]

Grouping Data

- Take a flat structure and turn it into a (potentially nested) map
- Similar to a groupby in databases
- Data

```
var expenses = [{"name": "jim", "amount": 34, "date": "11/12/2015"},  
  {"name": "carl", "amount": 120.11, "date": "11/12/2015"},  
  {"name": "jim", "amount": 45, "date": "12/01/2015"},  
  {"name": "stacy", "amount": 12.00, "date": "01/04/2016"},  
  {"name": "stacy", "amount": 34.10, "date": "01/04/2016"},  
  {"name": "stacy", "amount": 44.80, "date": "01/05/2016"}  
];
```

- Grouping:

```
expensesByName = d3.group(expenses, d => d.name)
```

- Results:

```
Map(3) { "jim" => Array(2) [Object, Object]  
  "carl" => Array(1) [Object]  
  "stacy" => Array(3) [Object, Object, Object] }
```

Rollup Data

- Data

```
var expenses = [{"name": "jim", "amount": 34, "date": "11/12/2015"},  
  {"name": "carl", "amount": 120.11, "date": "11/12/2015"},  
  {"name": "jim", "amount": 45, "date": "12/01/2015"},  
  {"name": "stacy", "amount": 12.00, "date": "01/04/2016"},  
  {"name": "stacy", "amount": 34.10, "date": "01/04/2016"},  
  {"name": "stacy", "amount": 44.80, "date": "01/05/2016"}  
];
```

- Using d3.rollup:

```
expensesAvgAmount = d3.rollup(  
  expenses,  
  v => d3.mean(v, d => d.amount), // aggregate by the mean of amount  
  d => d.name // group by name  
)
```

← the aggregation function
(difference from group)

- Result:

```
Map(3) {  
  "jim" => 39.5  
  "carl" => 120.11  
  "stacy" => 30.3  
}
```

[<http://learnjsdata.com>]

groups and rollups

- Both group and rollup return Map objects
- groups and rollups are the same functions but return nested arrays
- More examples: <https://observablehq.com/@d3/d3-group>

SQL

- DuckDB can be used in the browser to load databases
- External databases can be queried directly

arquero

- Library for query processing and transformation of array-backed data tables
- Similar to database and/or dataframe frameworks
- Integrates with Apache Arrow
- Documentation
- Illustrated Guide to Arquero's Verbs