

SWE 432 -Web Application Development

Spring 2023



George Mason
University

Dr. Kevin Moran

Week 15:

Information Visualization
&
Final Exam Review





Administrivia

- *HW Assignment 5* - Due 5/5 (Friday)!
- *Course Evaluations* - Open Until Sunday May 7th.
- *Optional Final Exam Review Session:* During Office Hours Monday and Thursday next week.
 - Tuesday - React Review
 - Thursday - User Centered Design Review

Overview of Information Visualization



Today

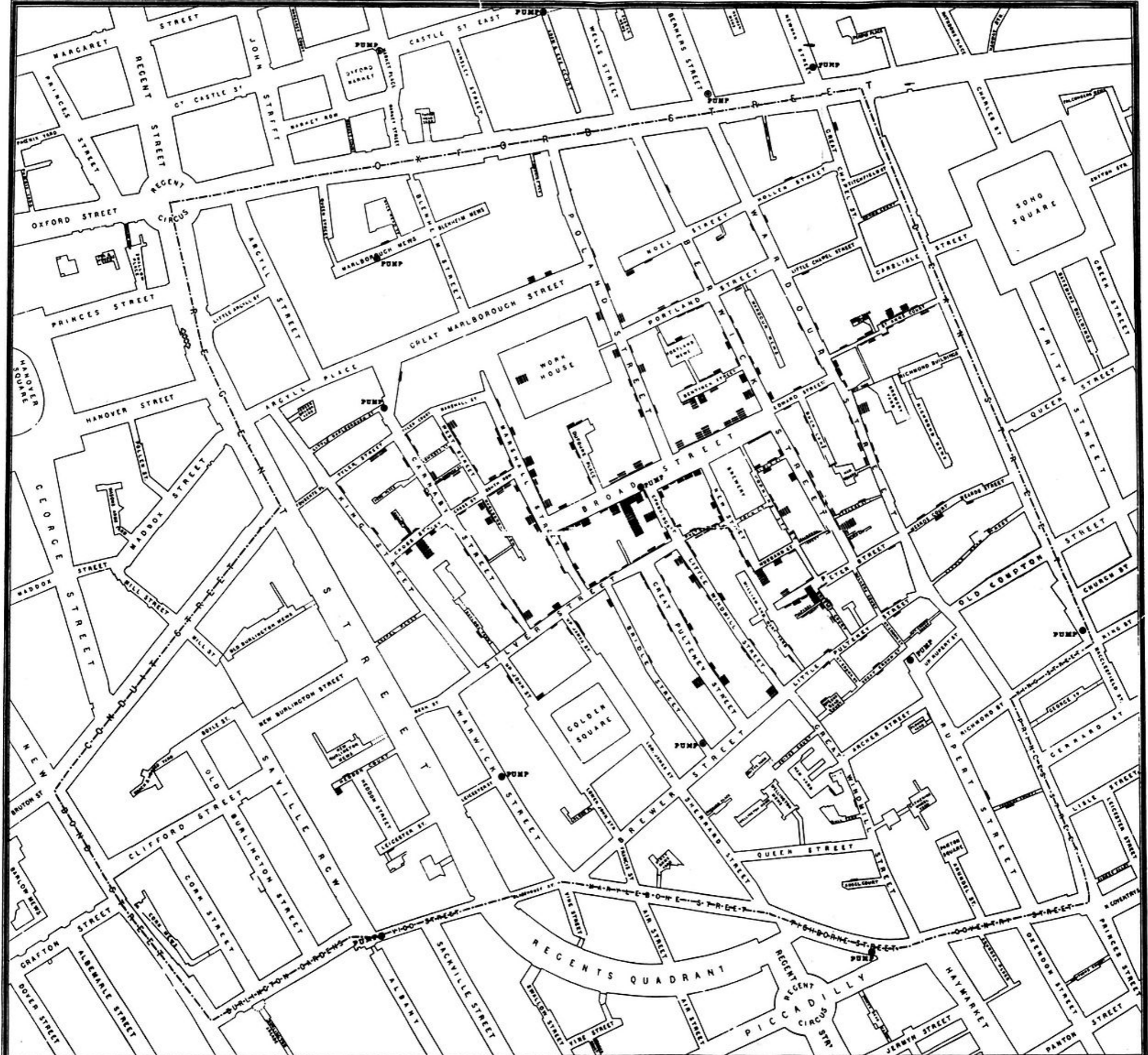


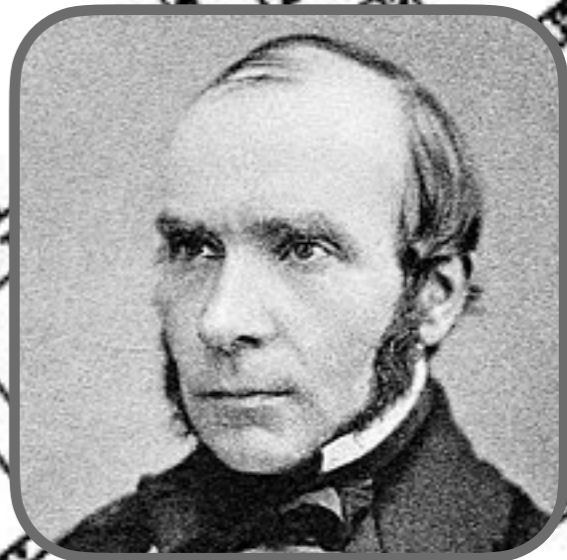
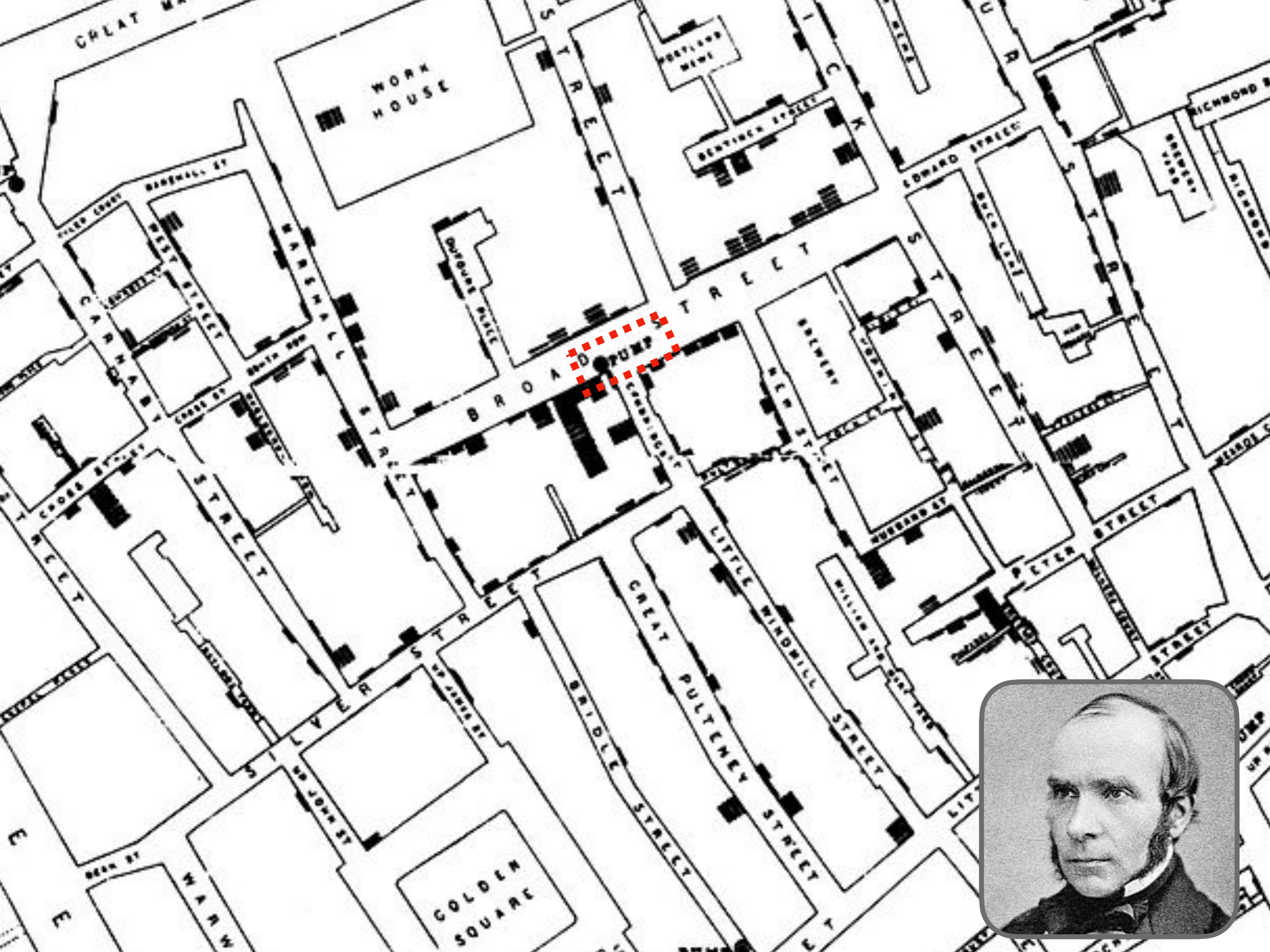
- What types of information visualization are there?
 - Which one should you choose?
- What principles and guidelines inform the design of information visualizations?
- How can interactivity be used to design better information visualizations?



Cholera Epidemic in London, 1854

- >500 fatal attacks of cholera in 10 days
 - Concentrated in Broad Street area of London
 - Many died in a few hours
- Dominant theory of disease: caused by noxious odors
- Afflicted streets deserted by >75% inhabitants







Investigation and Aftermath

- Based on visualization, did case by case investigation
- Found that 61 / 83 positive identified as using well water from Broad Street pump
- Board ordered pump-handle to be removed from well
- Epidemic soon ended
- Solved centuries old question of how cholera spread

Methods used by Snow

- Placed data in appropriate **context** for assessing cause & effect
 - Plotted on map, included well location
 - Reveals proximity as cause
- Made quantitative **comparisons**
 - Fewer deaths closer to brewery, could investigate cause
- Considered **alternative** explanations & contrary cases
 - Investigated cases not close to pump, often found connection to pump
- Assessment of possible **errors** in numbers

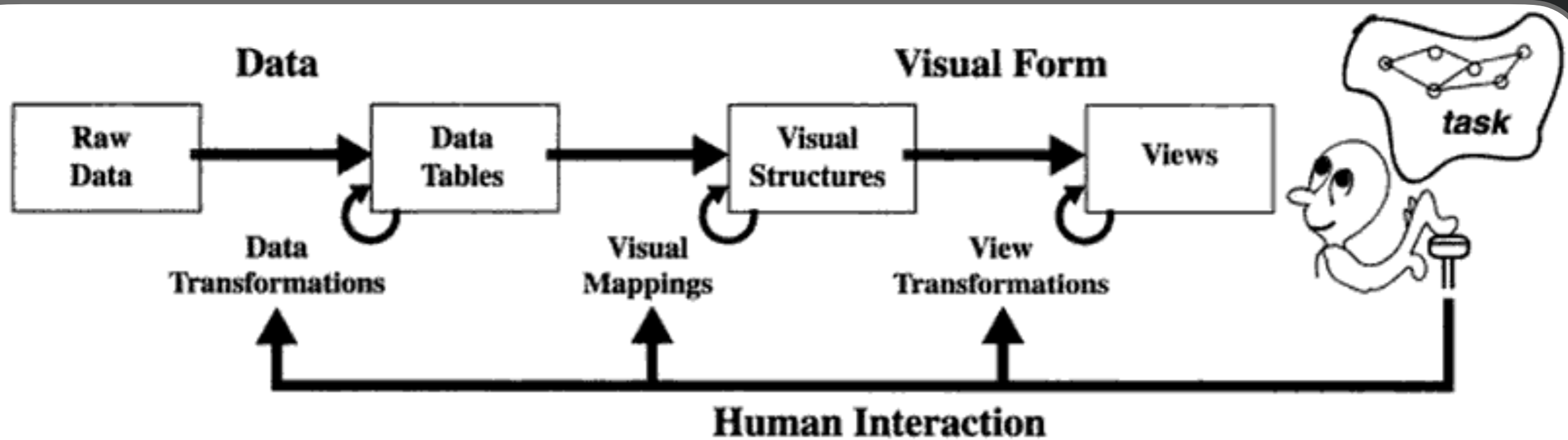
Amplifying Cognition

- Information Visualization can amplify cognition by:
 1. *Increasing the memory and processing resources available to users*
 2. *Reducing the search for information*
 3. *Using visual representations to enhance the detection of patterns*
 4. *Enabling perceptual inference*
 5. *Using perceptual attention mechanisms for monitoring*
 6. *Encoding Information in a manipulable medium*

Mapping Data to Visual Form



Designing an Information Visualization



Raw Data: idiosyncratic formats

Data Tables: relations (cases by variables) + metadata

Visual Structures: spatial substrates + marks + graphical properties

Views: graphical parameters (position, scaling, clipping, ...)

Types of Raw Data

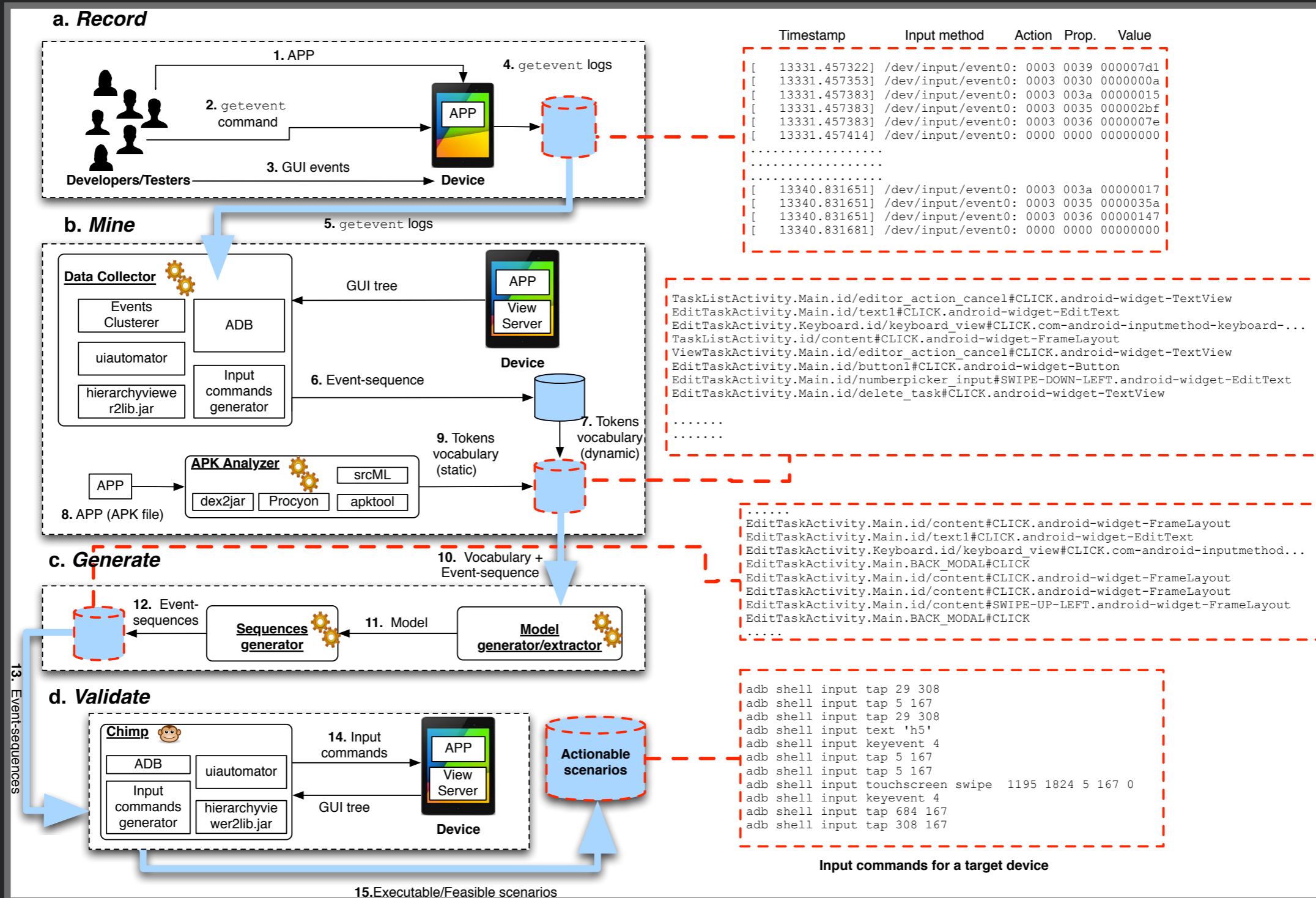
- Nominal - unordered set *without* a quantitative value
 - Gender: male, female
 - Hair color: brown, black, blonde, gray, orange, ...
- Ordinal - *ordered* set, with no meaning assigned to differences
 - How do you feel today: very unhappy, unhappy, ok, happy, very happy
 - Undefined how much better happy is than ok
- Quantitative - *numeric* value
 - Height, weight, distance, ...



Data Transformations

- Classing / binning: Quantitative \rightarrow ordinal
 - Maps ranges onto classes of variables
 - Can also count # of items in each class w/ histogram
- Sorting: Nominal \rightarrow ordinal
 - Add order between items in sets
- Descriptive statistics: mean, average, median, max, min, ...

Example Uses of a Data Transformation



Timestamp	Input method	Action	Prop.	Value
[13331.457322]	/dev/input/event0:	0003 0039	000007d1	
[13331.457353]	/dev/input/event0:	0003 0030	0000000a	
[13331.457383]	/dev/input/event0:	0003 003a	00000015	
[13331.457383]	/dev/input/event0:	0003 0035	000002bf	
[13331.457383]	/dev/input/event0:	0003 0036	0000007e	
[13331.457414]	/dev/input/event0:	0000 0000	00000000	
[13340.831651]	/dev/input/event0:	0003 003a	00000017	
[13340.831651]	/dev/input/event0:	0003 0035	0000035a	
[13340.831651]	/dev/input/event0:	0003 0036	00000147	
[13340.831681]	/dev/input/event0:	0000 0000	00000000	

```
TaskListActivity.Main.id/editor_action_cancel#CLICK.android-widget-TextView
EditTaskActivity.Main.id/text1#CLICK.android-widget-EditText
EditTaskActivity.Keyboard.id/keyboard_view#CLICK.com-android-inputmethod-keyboard-...
TaskListActivity.id/content#CLICK.android-widget-FrameLayout
ViewTaskActivity.Main.id/editor_action_cancel#CLICK.android-widget-TextView
EditTaskActivity.Main.id/button1#CLICK.android-widget-Button
EditTaskActivity.Main.id/numberpicker_input#SWIPE-DOWN-LEFT.android-widget-EditText
EditTaskActivity.Main.id/delete_task#CLICK.android-widget-TextView
```

```
EditTaskActivity.Main.id/content#CLICK.android-widget-FrameLayout
EditTaskActivity.Main.id/text1#CLICK.android-widget-EditText
EditTaskActivity.Keyboard.id/keyboard_view#CLICK.com-android-inputmethod-...
EditTaskActivity.Main.BACK_MODAL#CLICK
EditTaskActivity.Main.id/content#CLICK.android-widget-FrameLayout
EditTaskActivity.Main.id/content#CLICK.android-widget-FrameLayout
EditTaskActivity.Main.id/content#SWIPE-UP-LEFT.android-widget-FrameLayout
EditTaskActivity.Main.BACK_MODAL#CLICK
```

```
adb shell input tap 29 308
adb shell input tap 5 167
adb shell input tap 29 308
adb shell input text 'h5'
adb shell input keyevent 4
adb shell input tap 5 167
adb shell input tap 5 167
adb shell input touchscreen swipe 1195 1824 5 167 0
adb shell input keyevent 4
adb shell input tap 684 167
adb shell input tap 308 167
```

Input commands for a target device



Visual Structures

- 3 components
 - spatial substrate
 - marks
 - marks' graphical properties

Spatial Substrate

- Axes that divide space
- Types of axes - unstructured, nominal, ordinal, quantitative
- Composition - use of multiple orthogonal axes (e.g., 2D scatterplot, 3D)



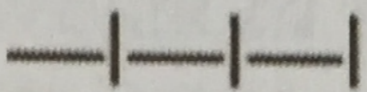
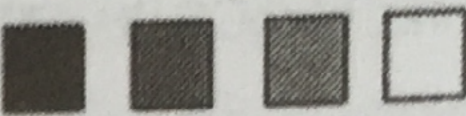
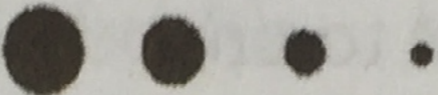
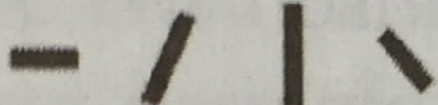


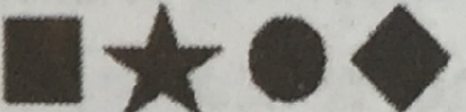


Marks

- Points (0D)
- Lines (1D)
- Areas (2D)
- Volumes (3D)

Marks' Graphical Properties

- Quantitative (Q), Ordinal (O), Nominal (N)
- Filled circle - good; open circle - bad

	Spatial	Object
Extent	(Position) 	Gray Scale 
	Size 	
Dif-feren-tial	Orientation 	Color 
		Texture 
		Shape 

Effectiveness of Graphical Properties

- Quantitative (Q), Ordinal (O), Nominal (N)
- Filled circle - good; open circle - bad

		Spatial	Q	O	N	Object	Q	O	N
Extent	(Position)		●	●	●	Grayscale	◐	●	○
	Size		●	●	●				
Differential			◐	◐	●	Color	◐	◐	●
	Orientation					Texture	◐	◐	●
						Shape	○	○	●

Animation

- Visualization can change over time
- Could be used to encode data as a function of time
 - But often not effective as makes direct comparisons hard
- Can be more effective to animate transition from before to after as user configures visualization

Examples of Visualizations

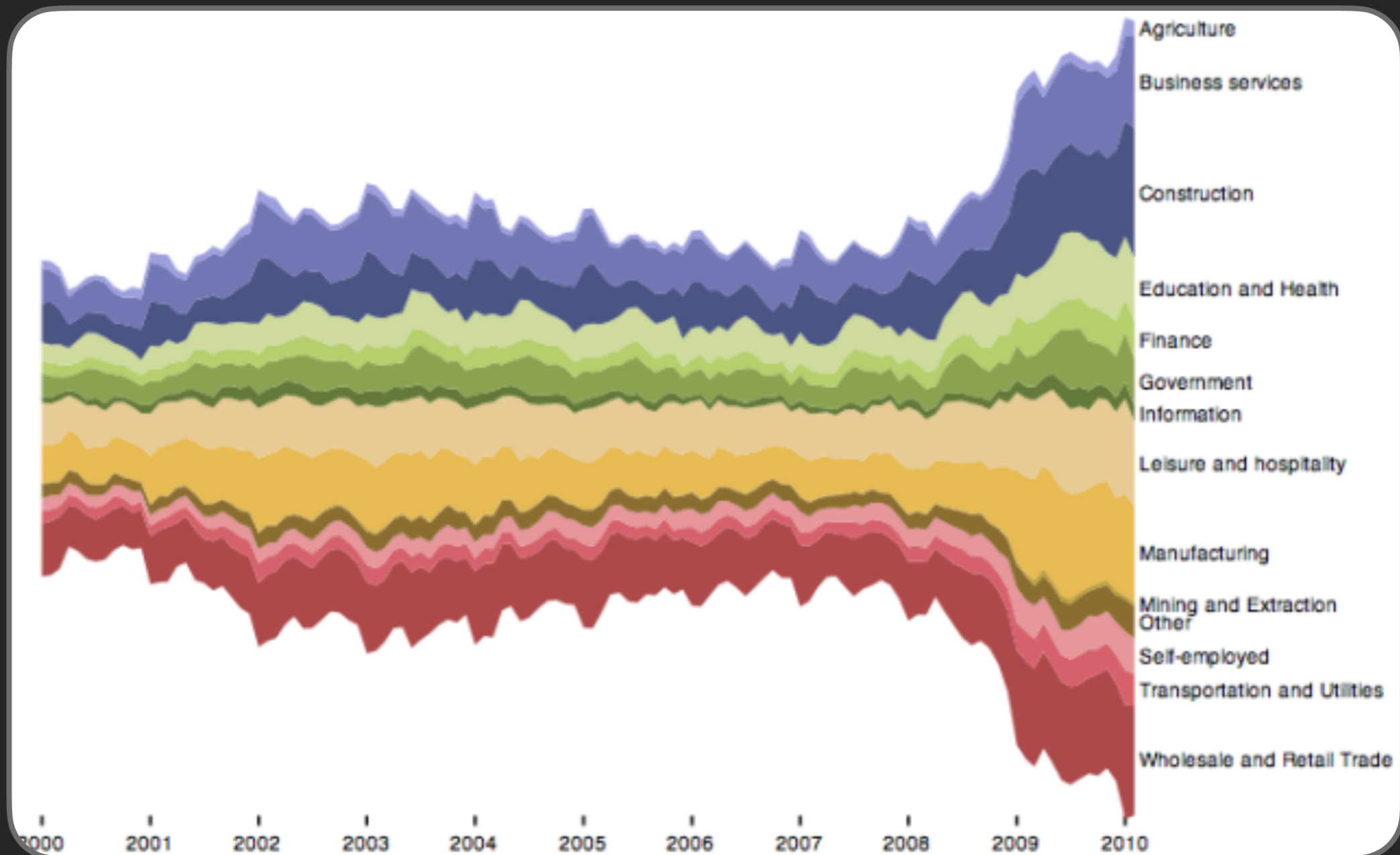


Time-series Data



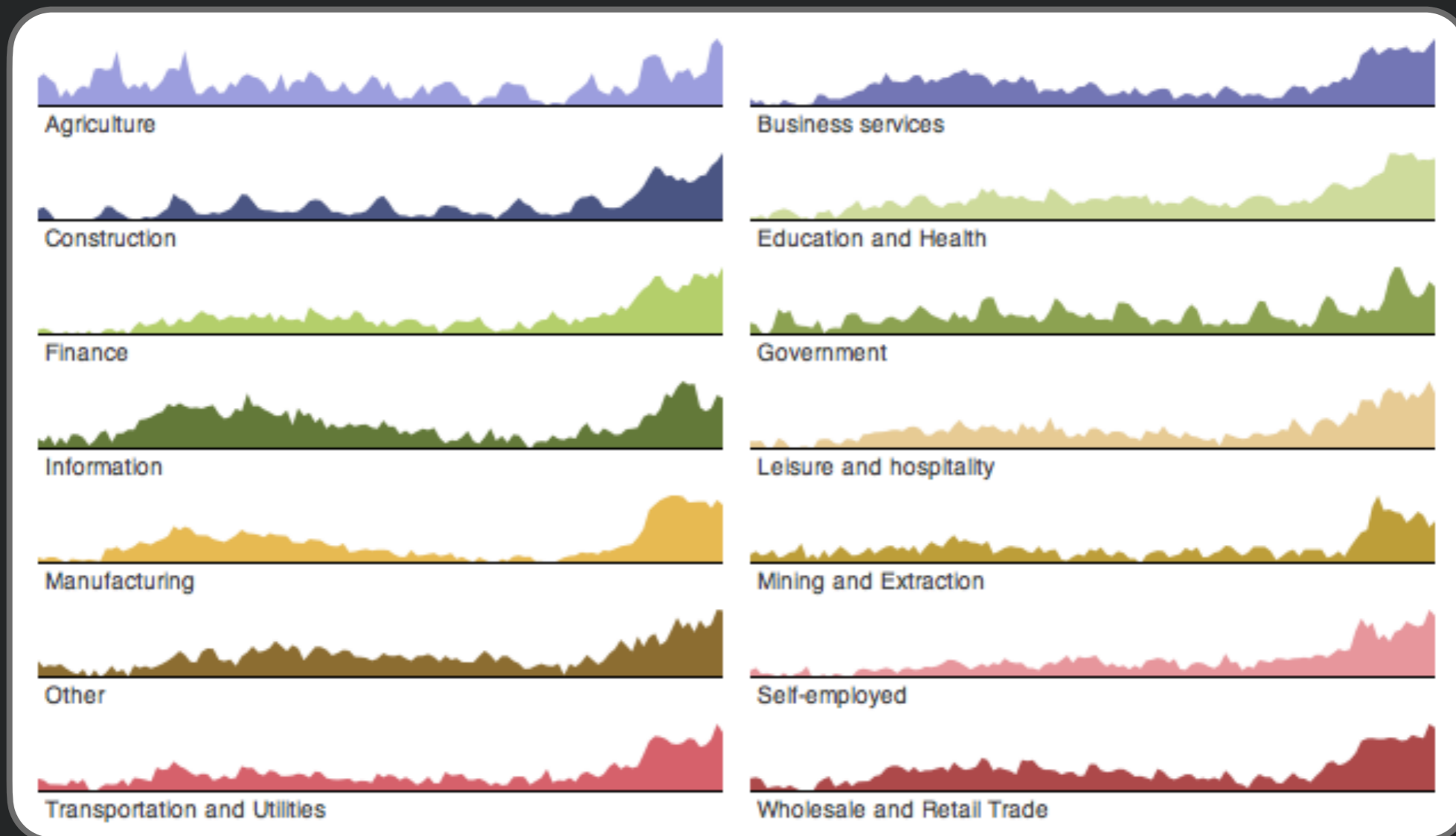
Stacked Graph

- Supports visual summation of multiple components



Small Multiples

- Supports separate comparison of data series
- May have better legibility than placing all in single plot

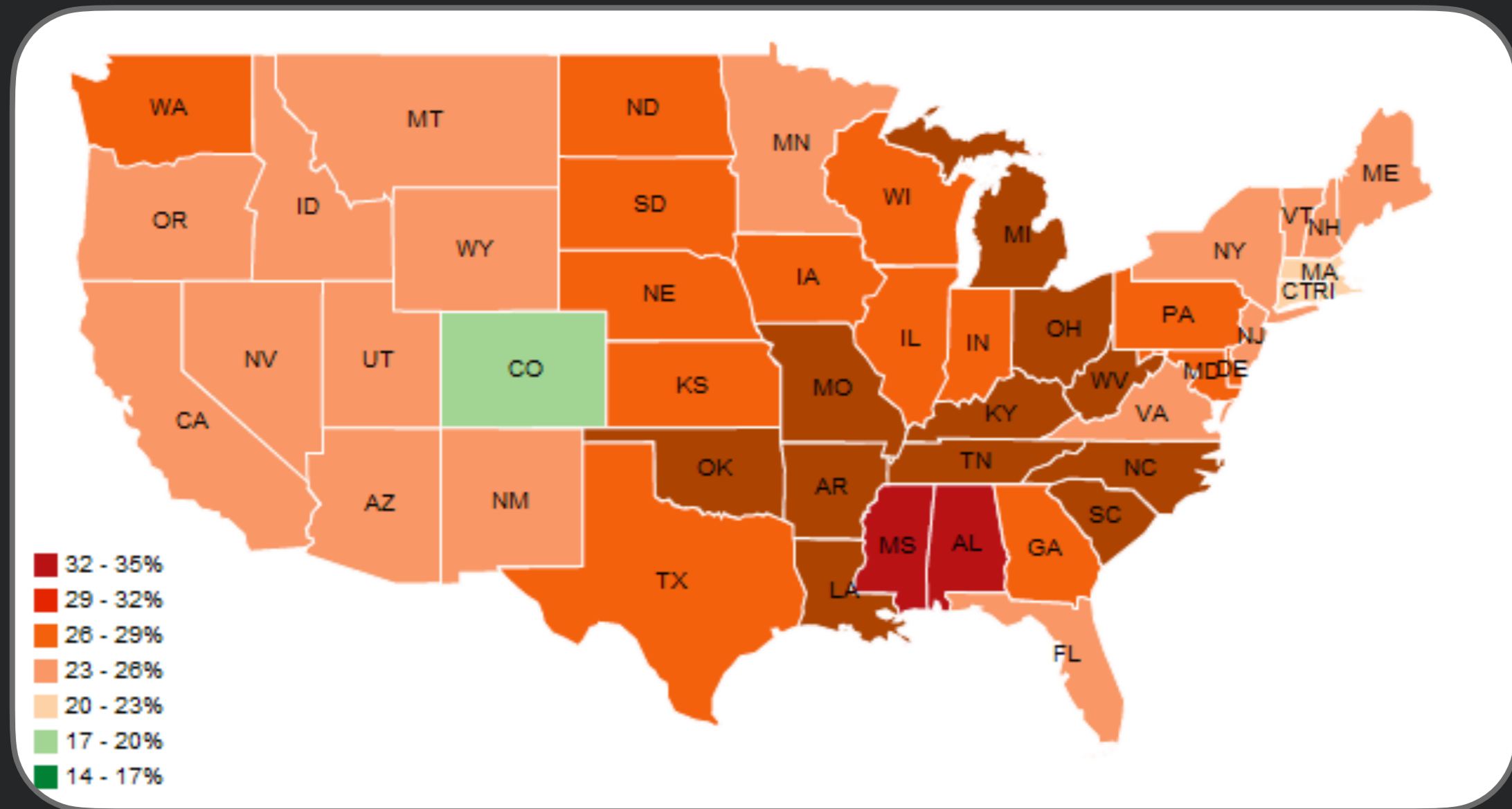


Maps



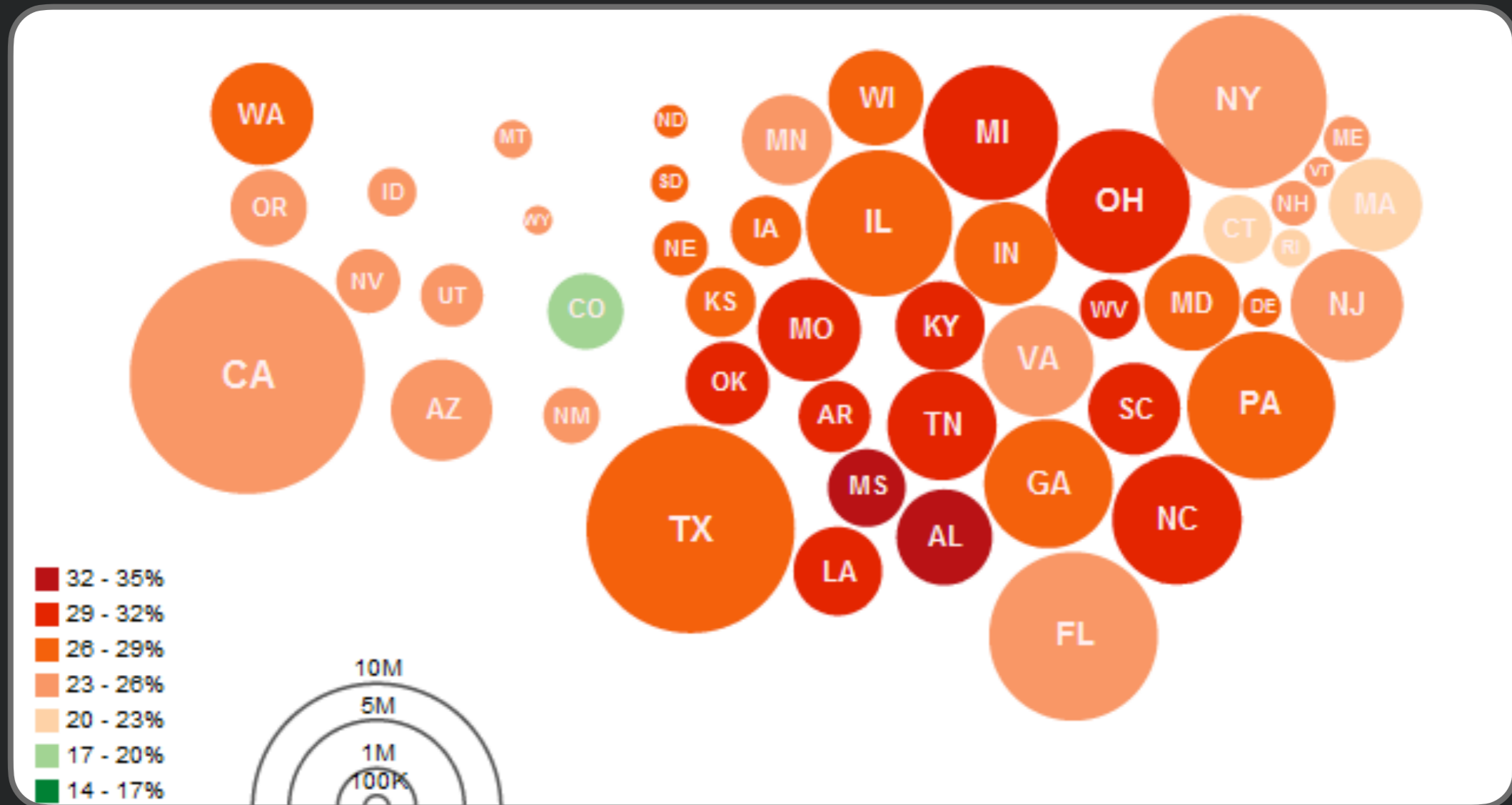
Choropleth Map

- Groups data by area, maps to color



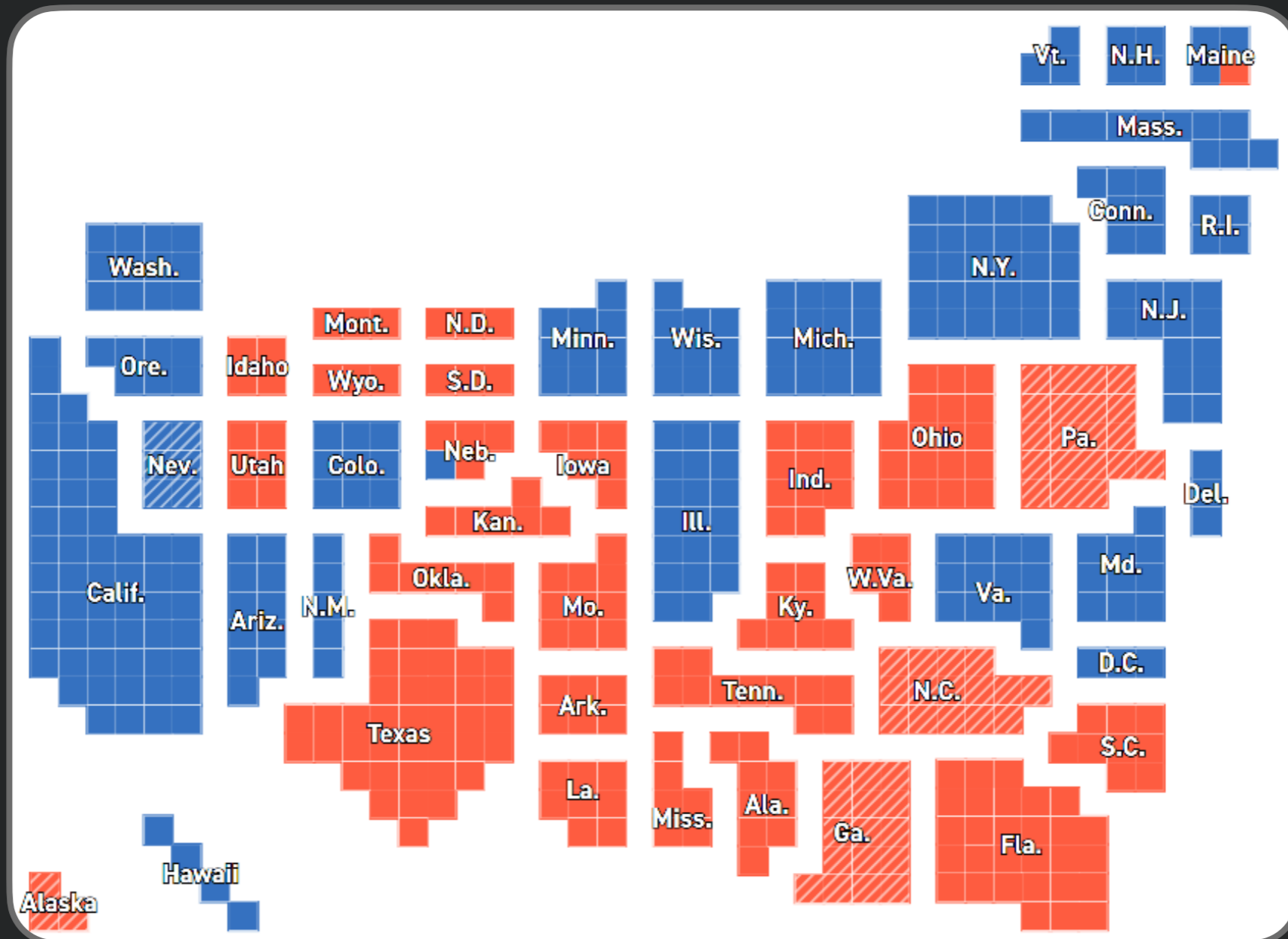
Cartograms

- Encodes two variables w/ size & color



Cartograms

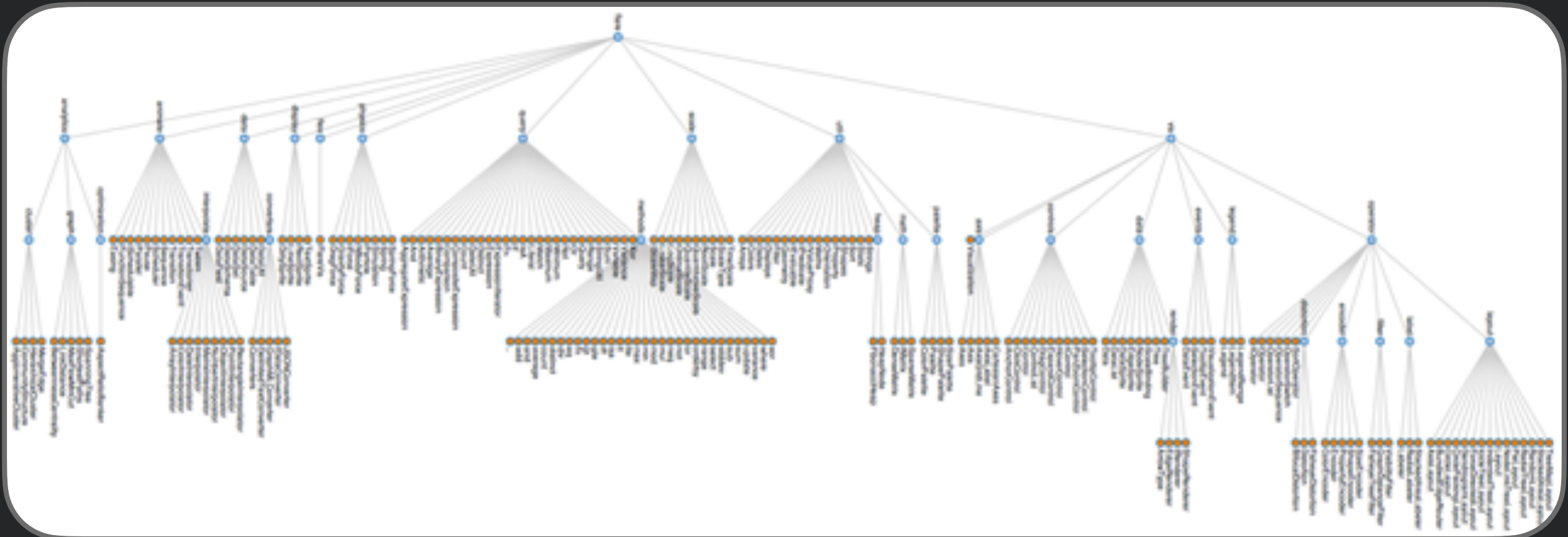
- Encodes two variables w/ size & color



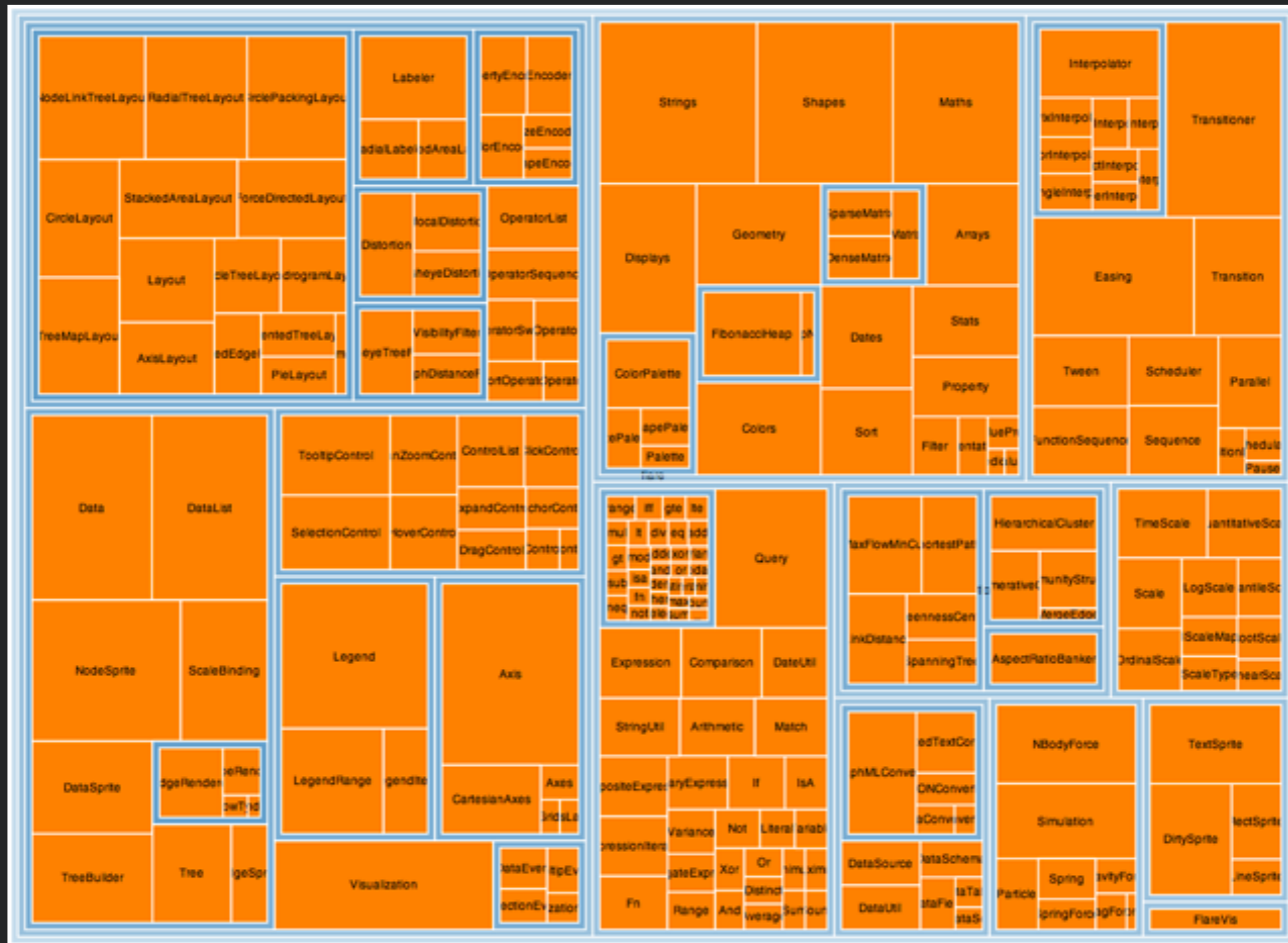
Hierarchies



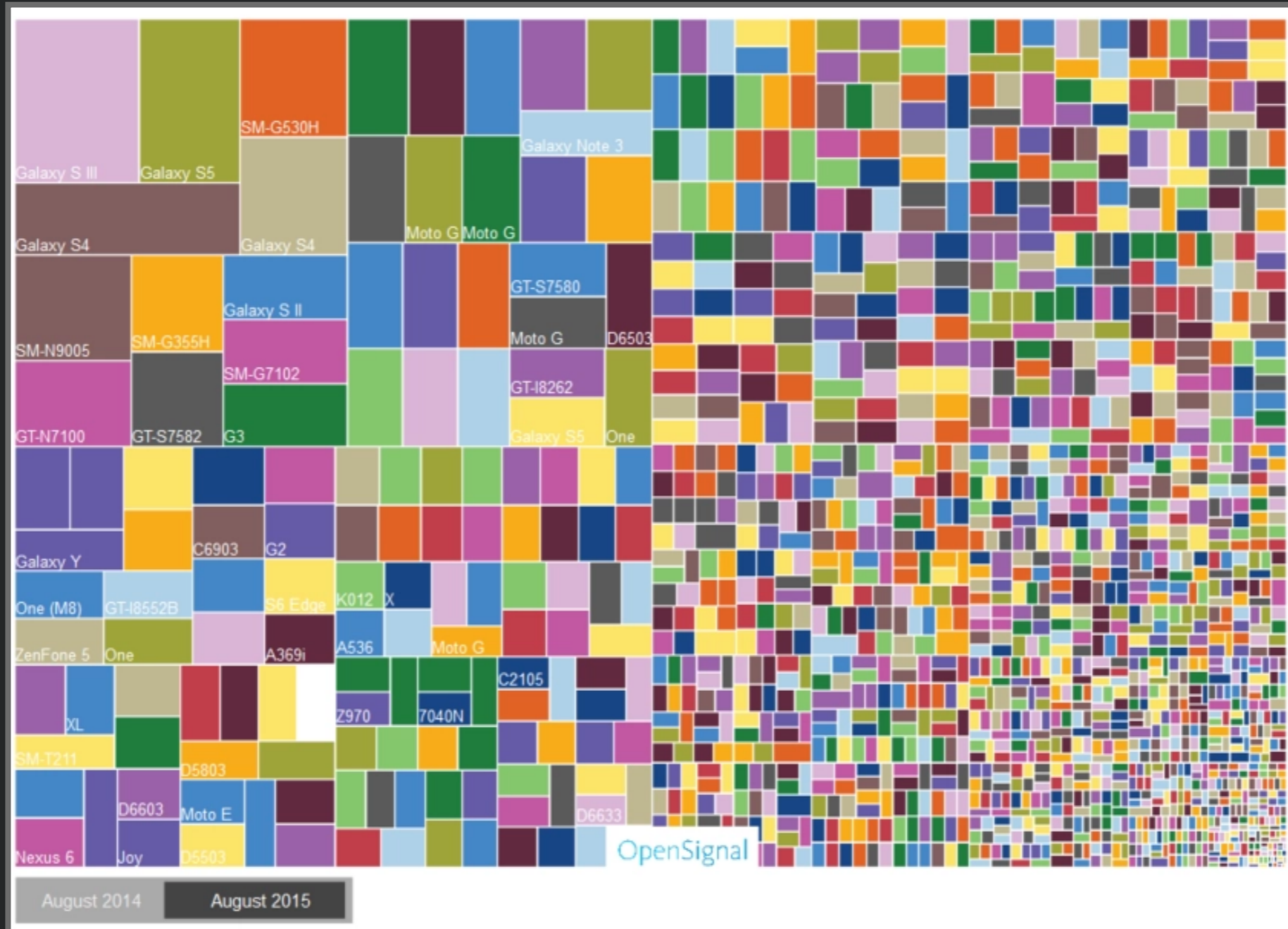
Node Link Diagram



Treemaps



Treemaps

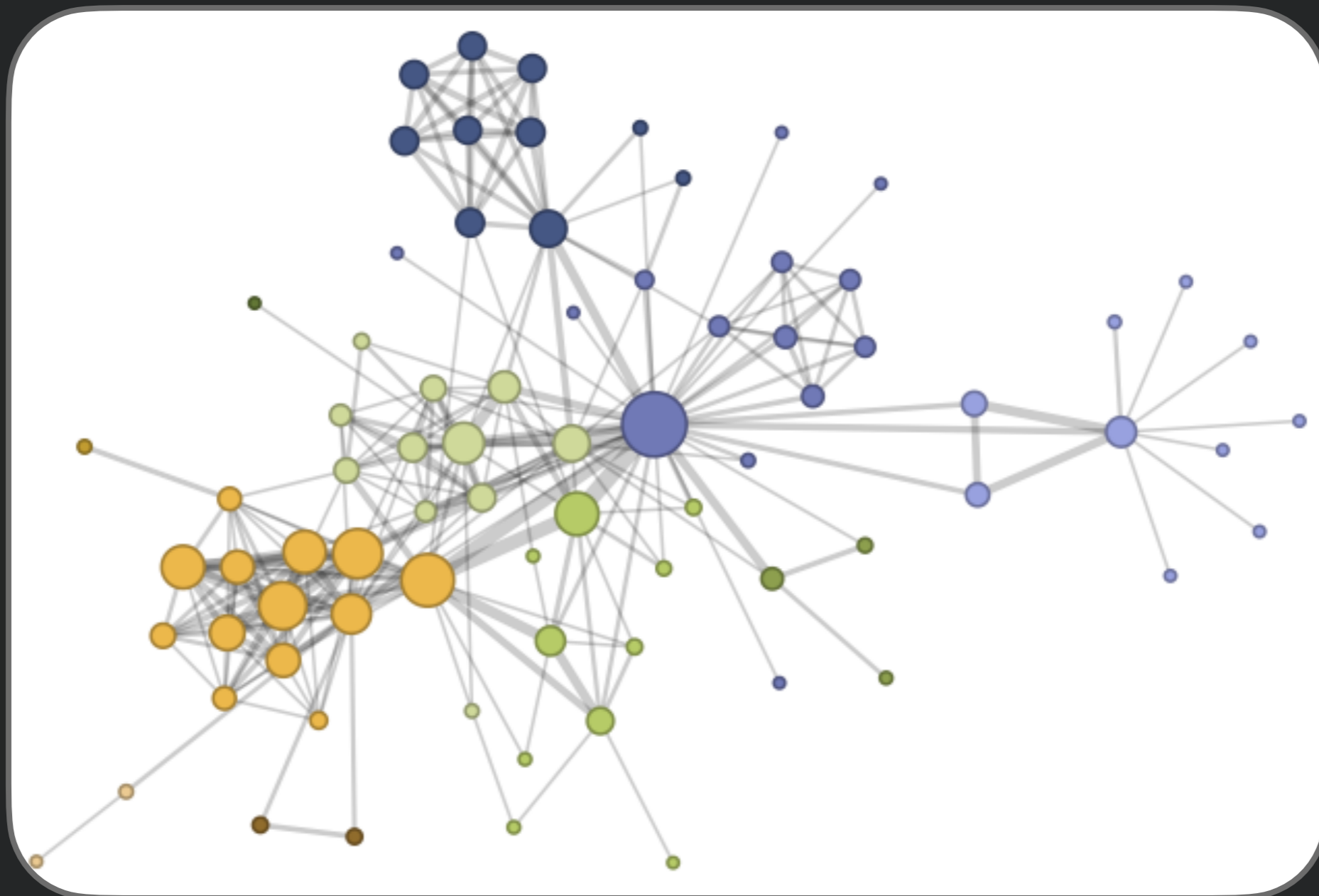


Networks



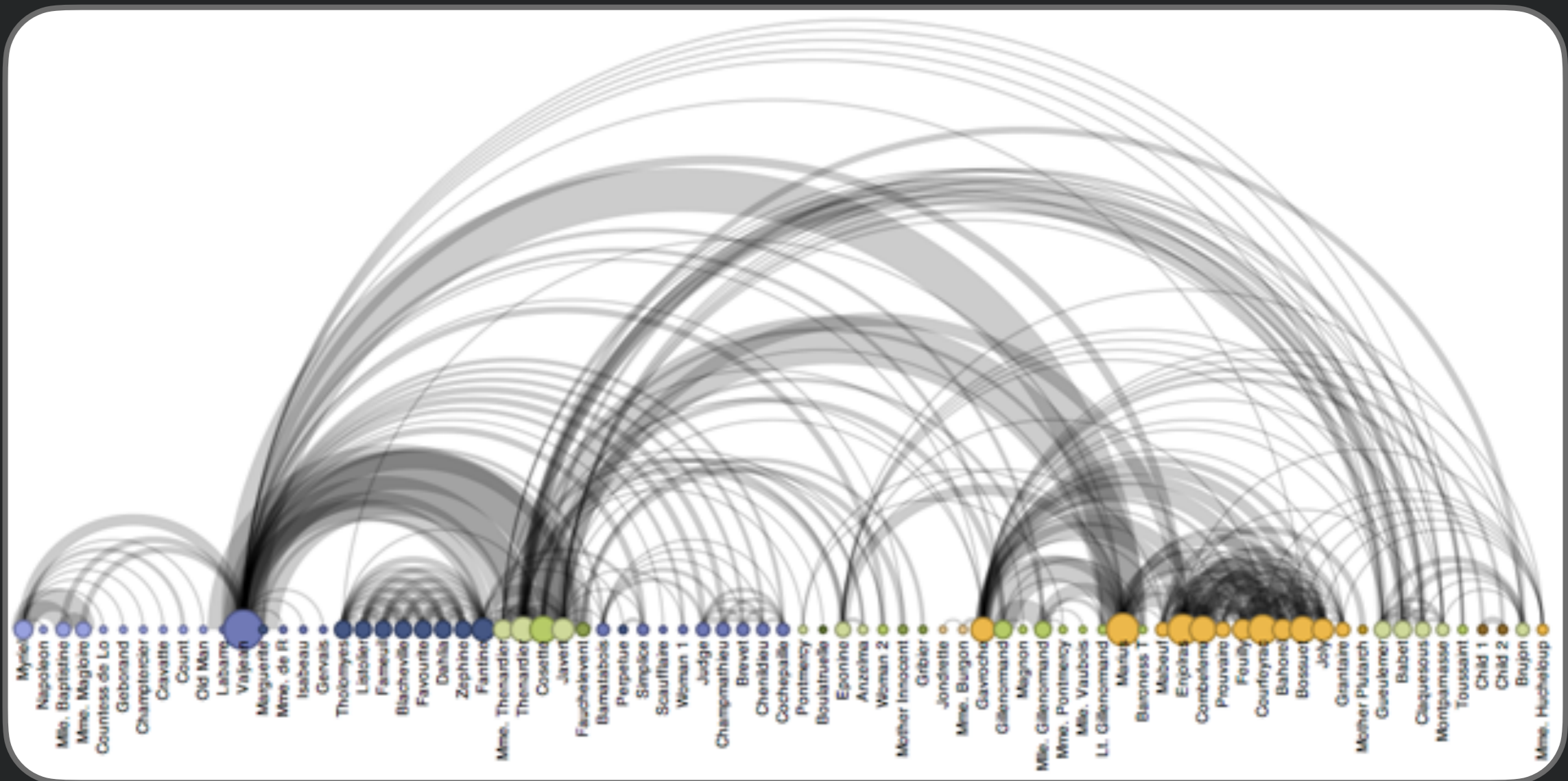
Force-directed Layout

- Edges function as springs, find least energy configuration



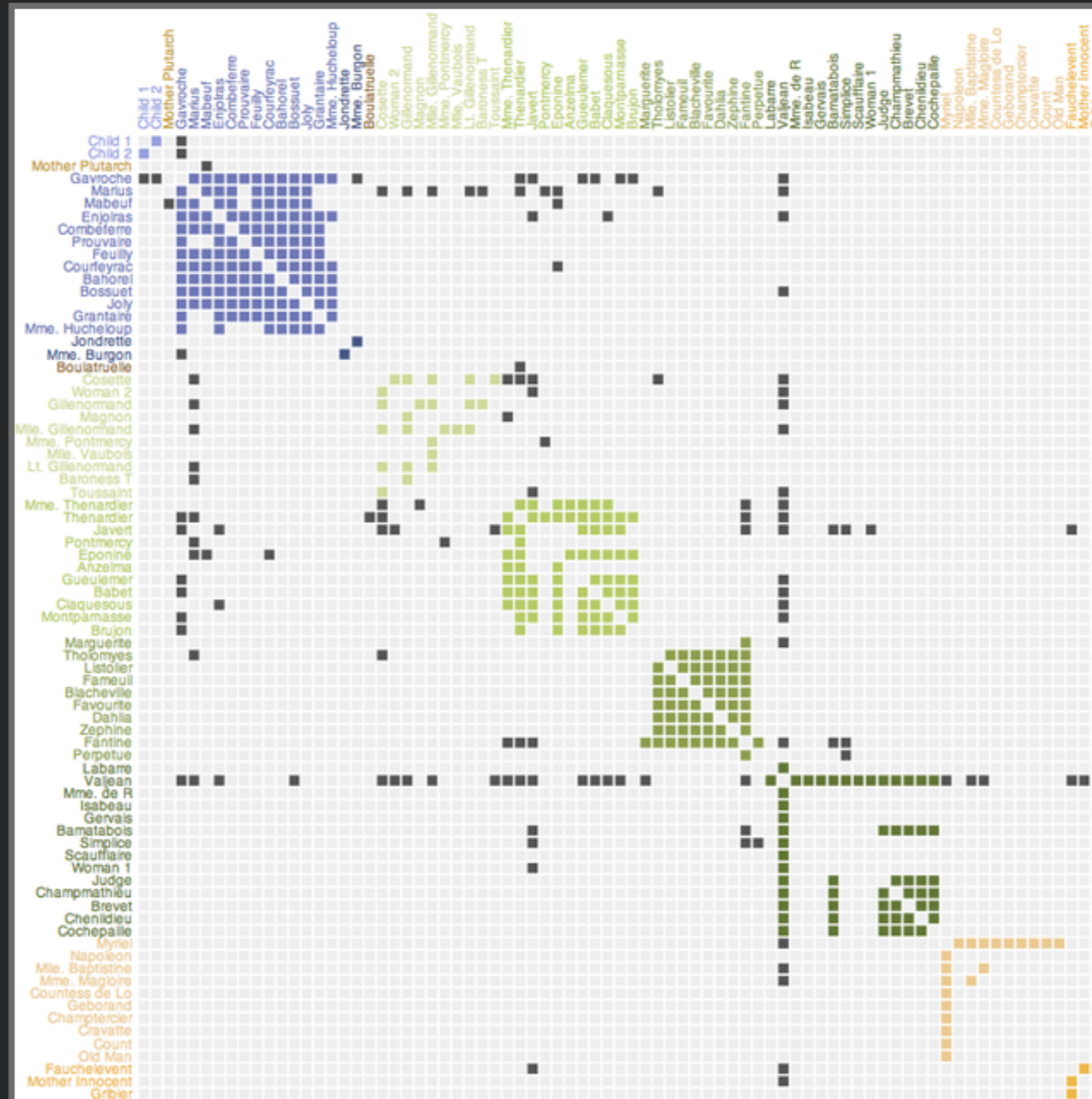
Arc Diagram

- Can support identifying cliques & bridges w/ right order





Adjacency Matrix



Design Considerations





Tufte's principles of graphical excellence

- Show the *data*
- Induce the viewer to think about the substance rather than the methodology
- Avoid distorting what the data have to say
- Present *many* numbers in a small space
- Make large data sets *coherent*
- Encourage the eye to *compare* different pieces of data
- Reveal data at several levels of detail, from overview to fine structure
- Serve reasonable clear *purpose*: description, exploration, tabulation, decoration



Distortions in Visualizations

- Visualizations may distort the underlying data, making it harder for reader to understand truth
- Use of *design* variation to try to falsely communicate *data* variation

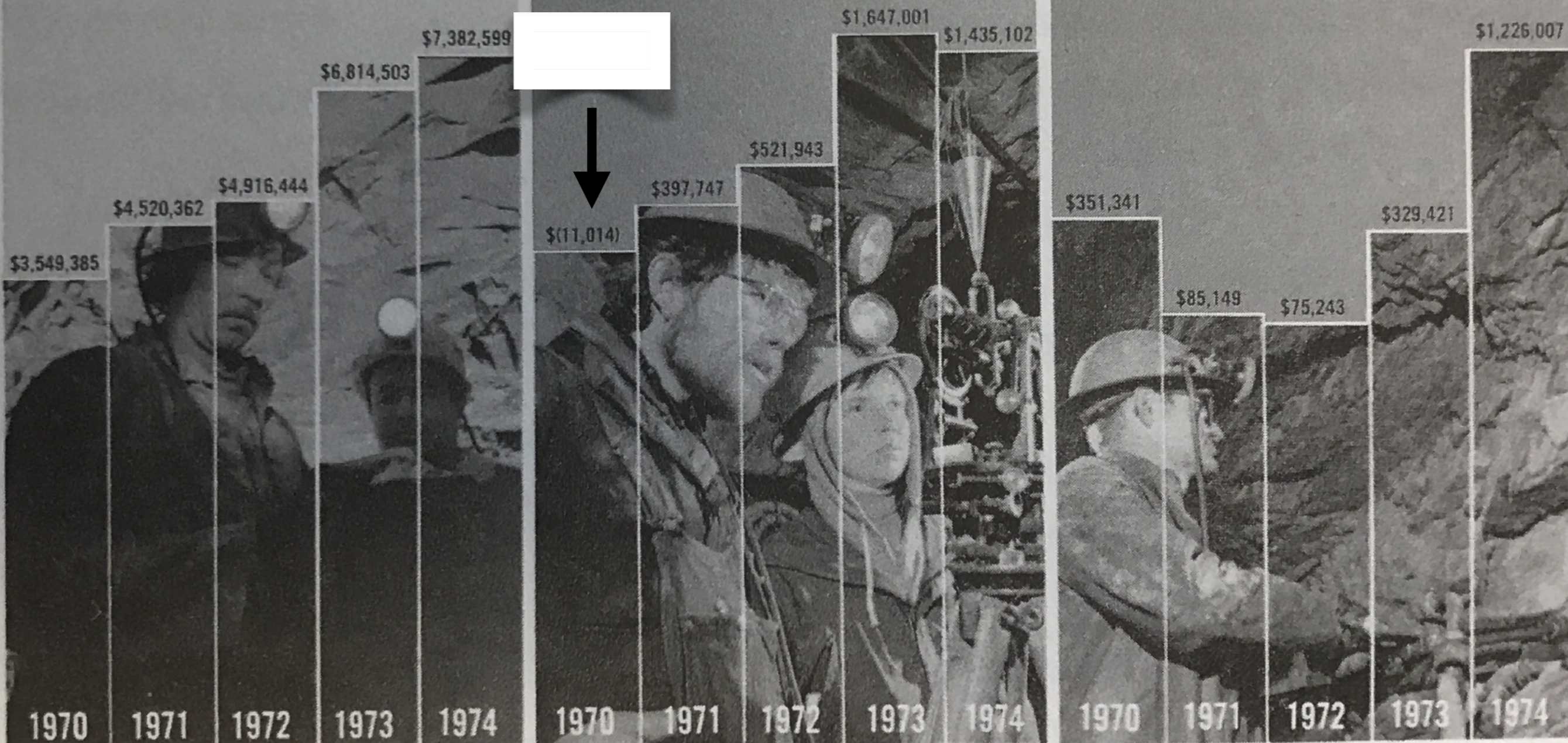
Example



OPERATING REVENUES

NET INCOME (LOSS)

EXPLORATION & DEVELOPMENT EXPENDITURES

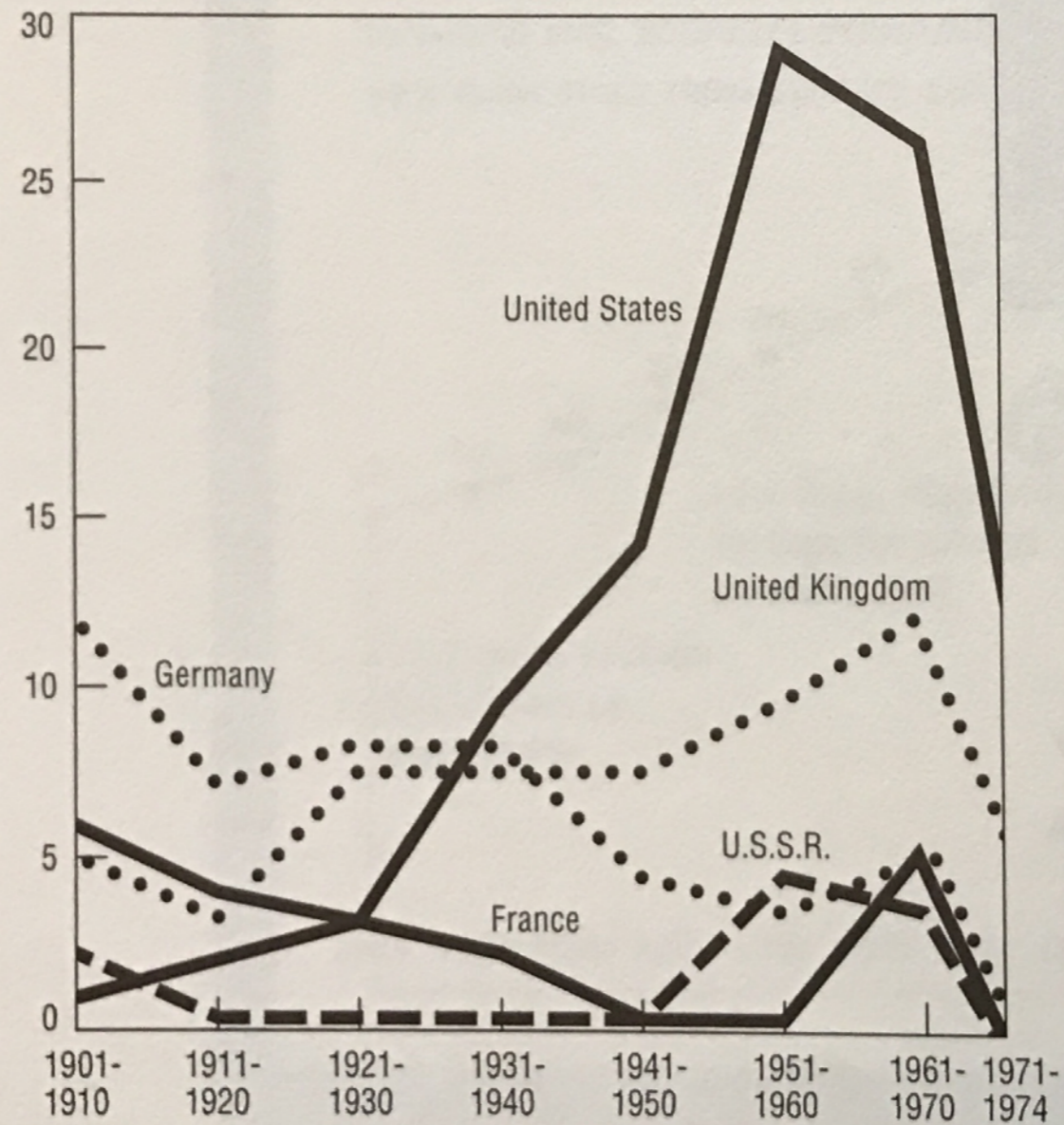


Example



**Nobel Prizes Awarded in Science,
for Selected Countries, 1901-1974**

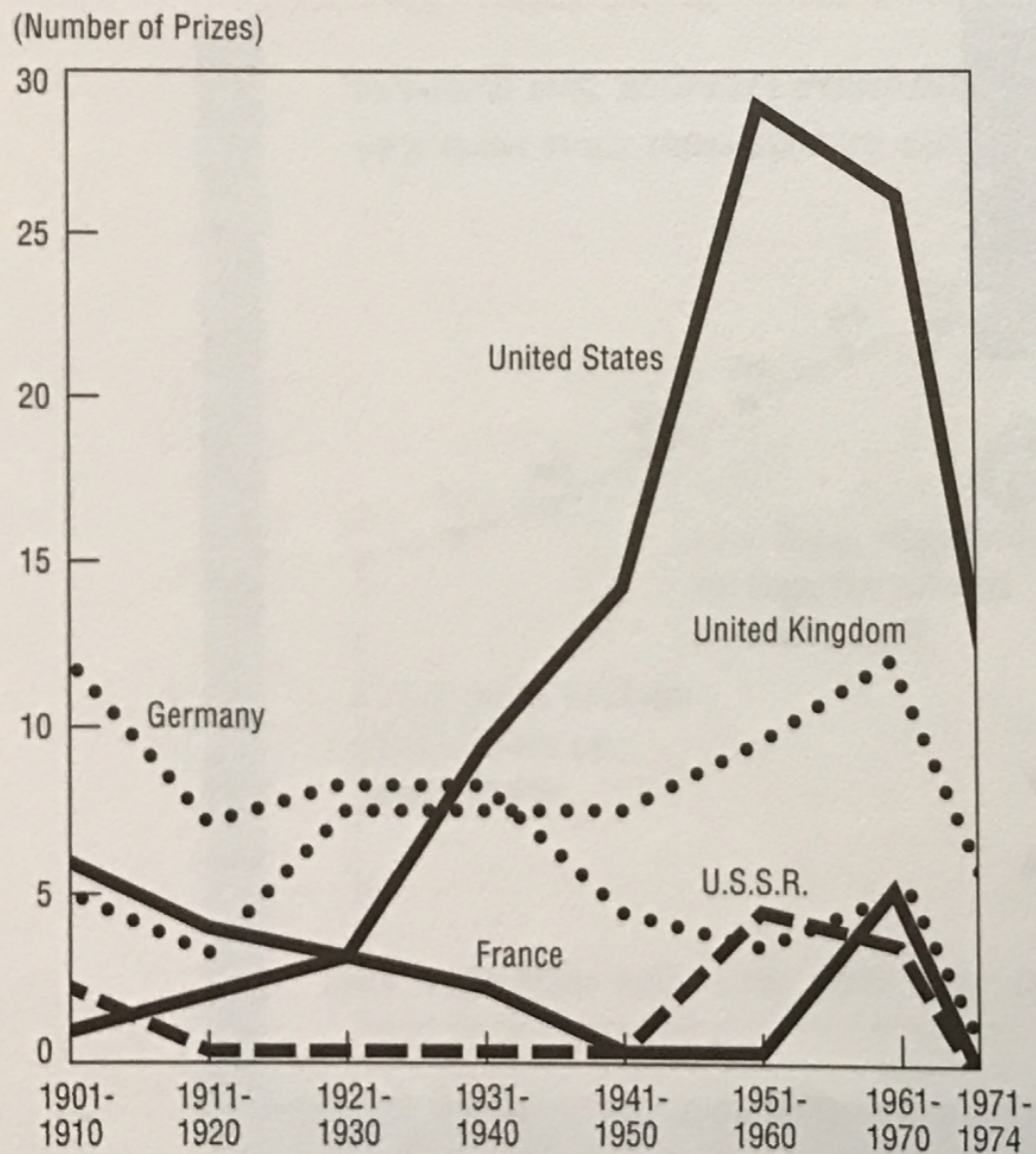
(Number of Prizes)



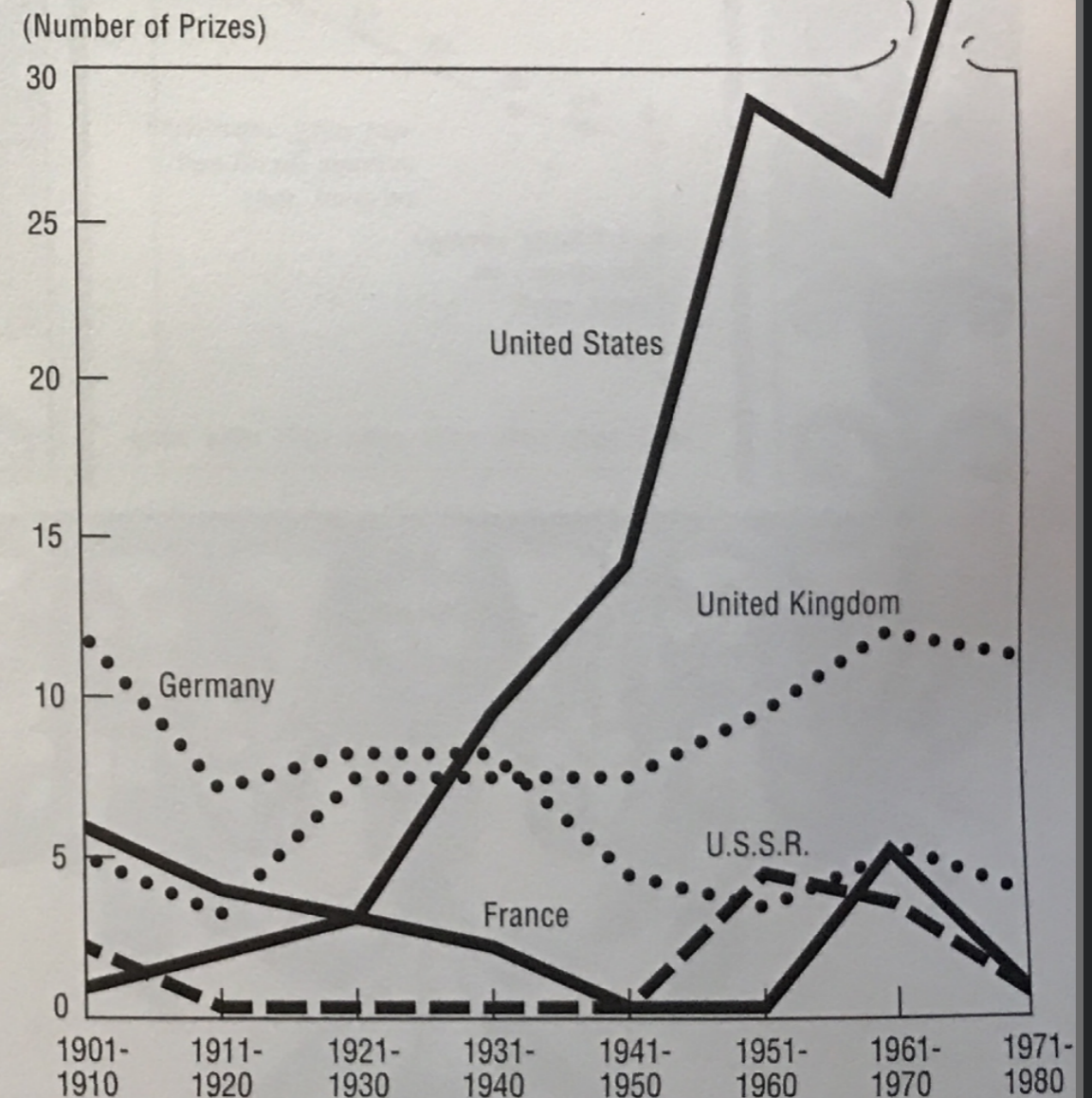
Example (corrected)



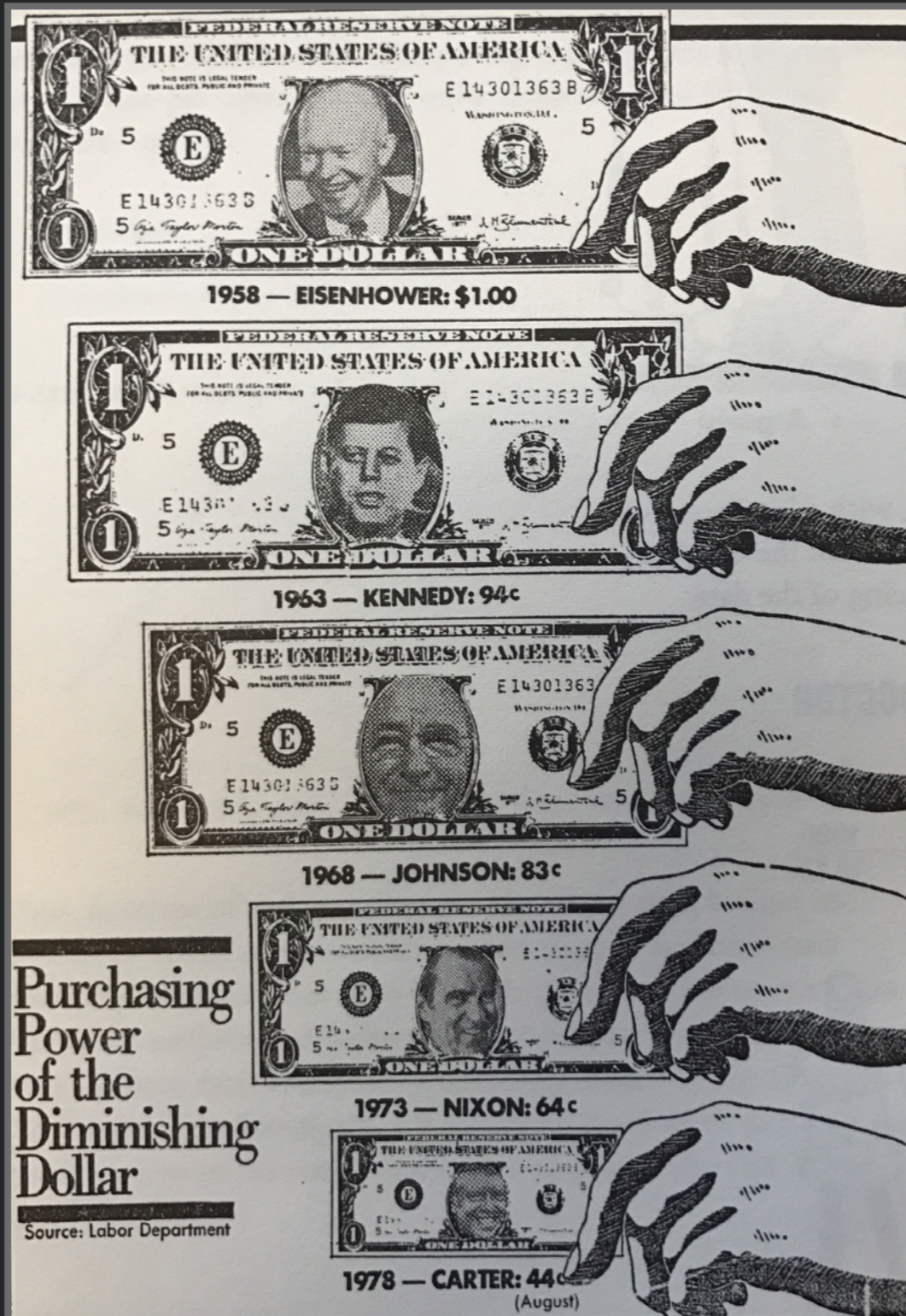
**Nobel Prizes Awarded in Science,
for Selected Countries, 1901-1974**



**Nobel Prizes Awarded in Science,
for Selected Countries, 1901-1980**

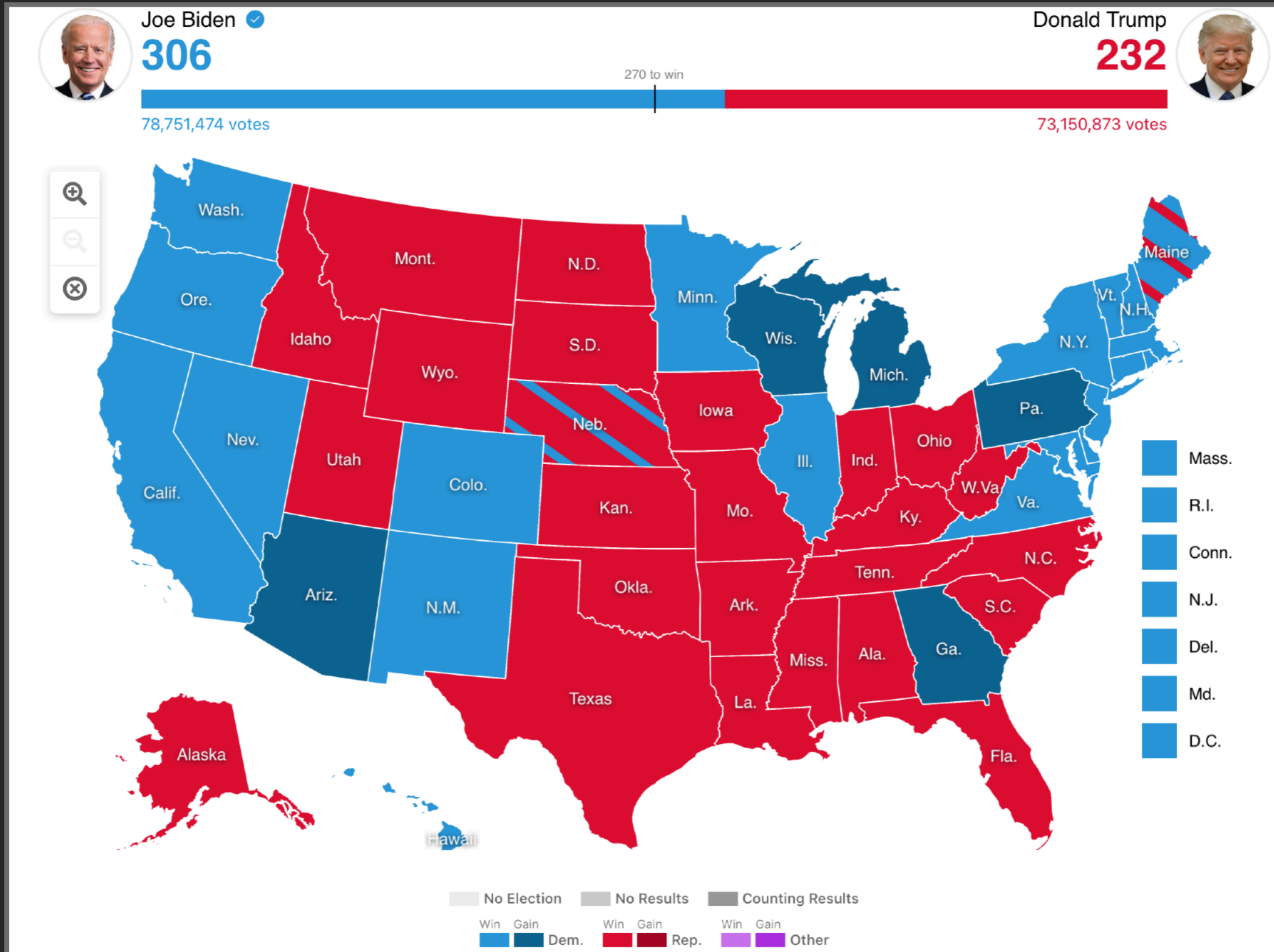


Example





Traditional Electoral Map



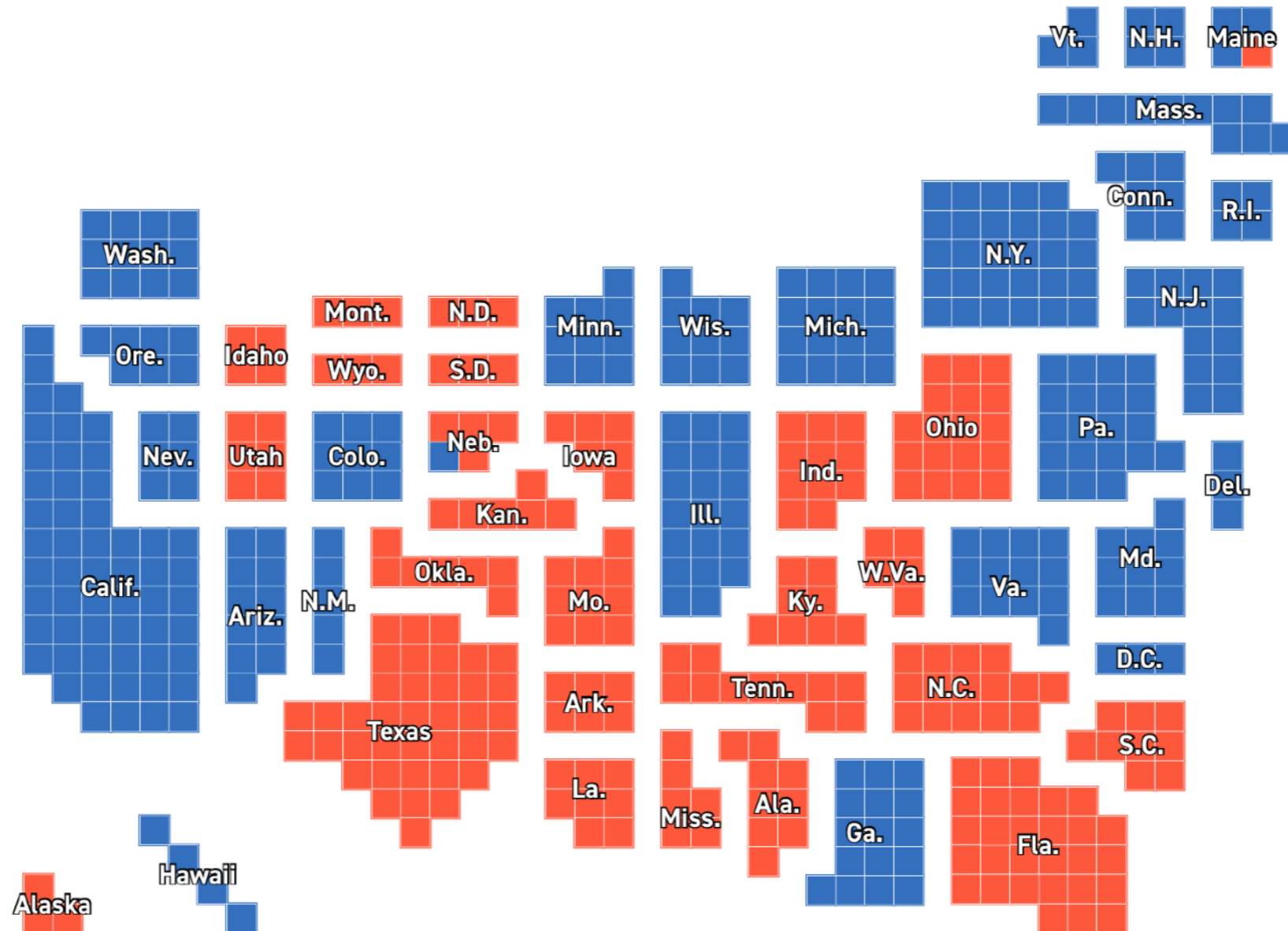


Weighted Electoral Map

✓ 306 BIDEN

270 to win

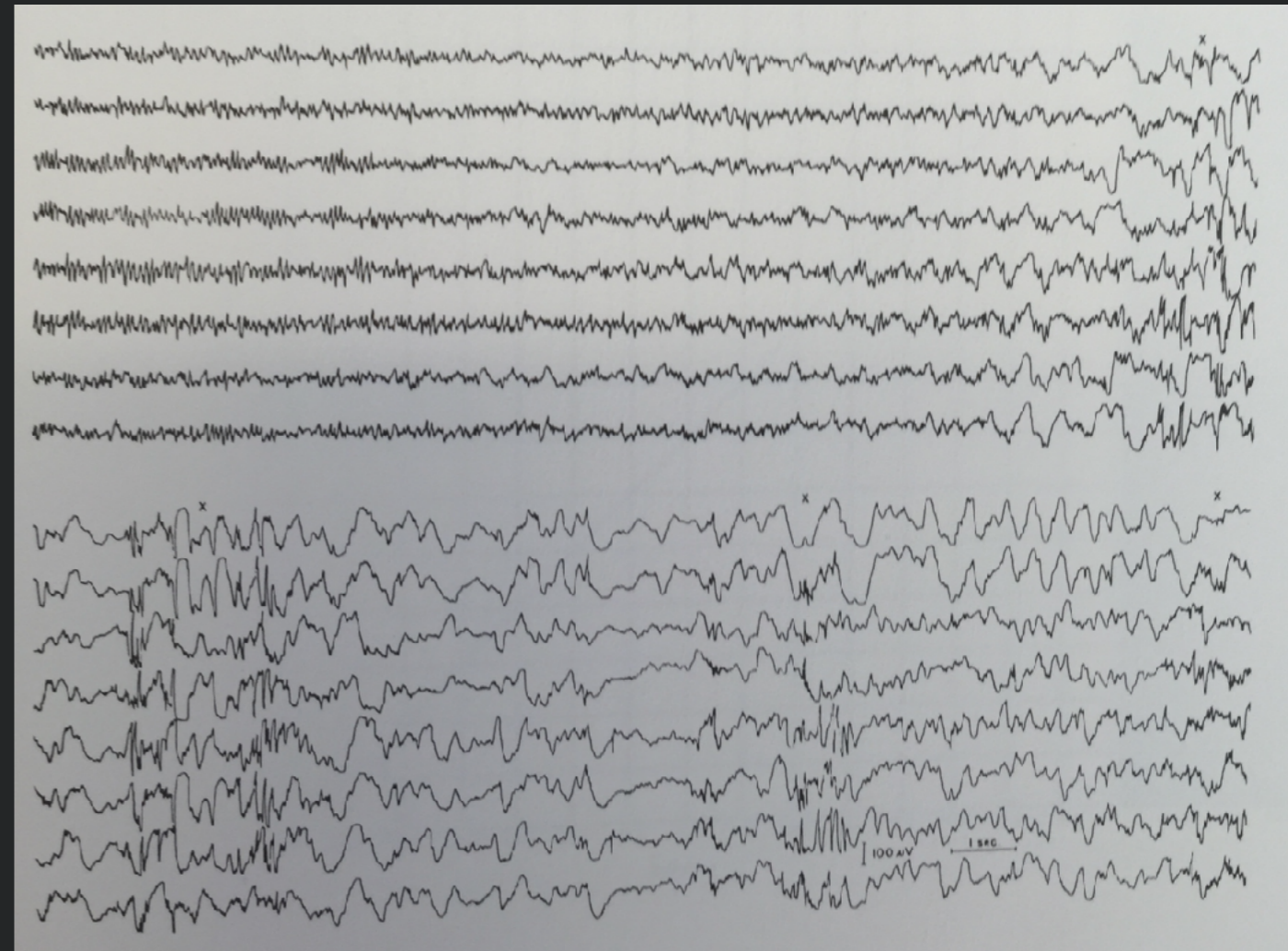
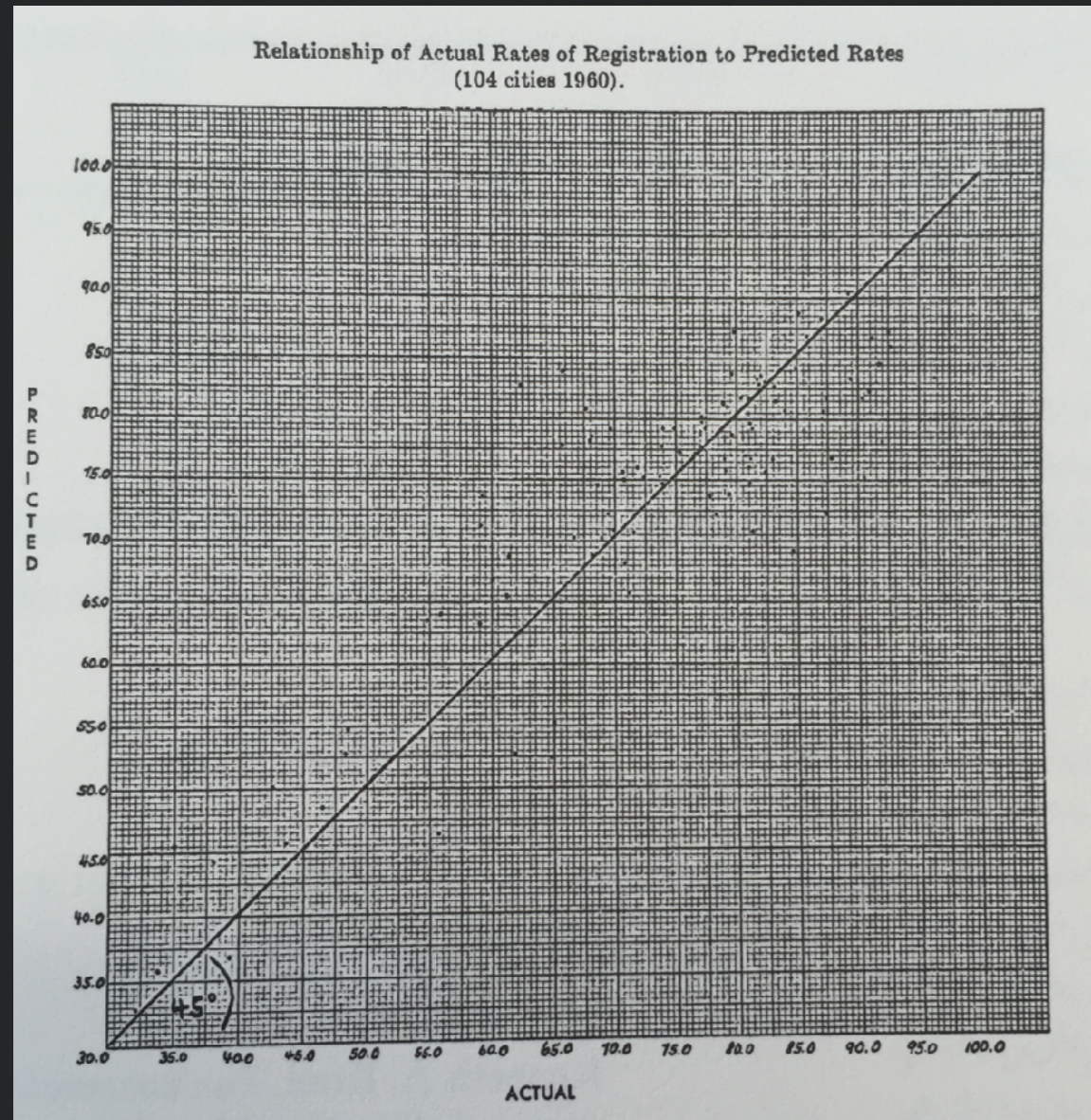
TRUMP 232



- Data-ink - non-redundant ink encoding data information

$$\begin{aligned} \text{Data-ink ratio} &= \frac{\text{Data-ink}}{\text{Total ink used to print the graphic}} \\ &= \text{proportion of a graphic's ink devoted to the} \\ &\quad \text{non-redundant display of data-information} \\ &= 1.0 - \text{proportion of a graphic that can be erased} \end{aligned}$$

Examples of Data-ink Ratio



~0

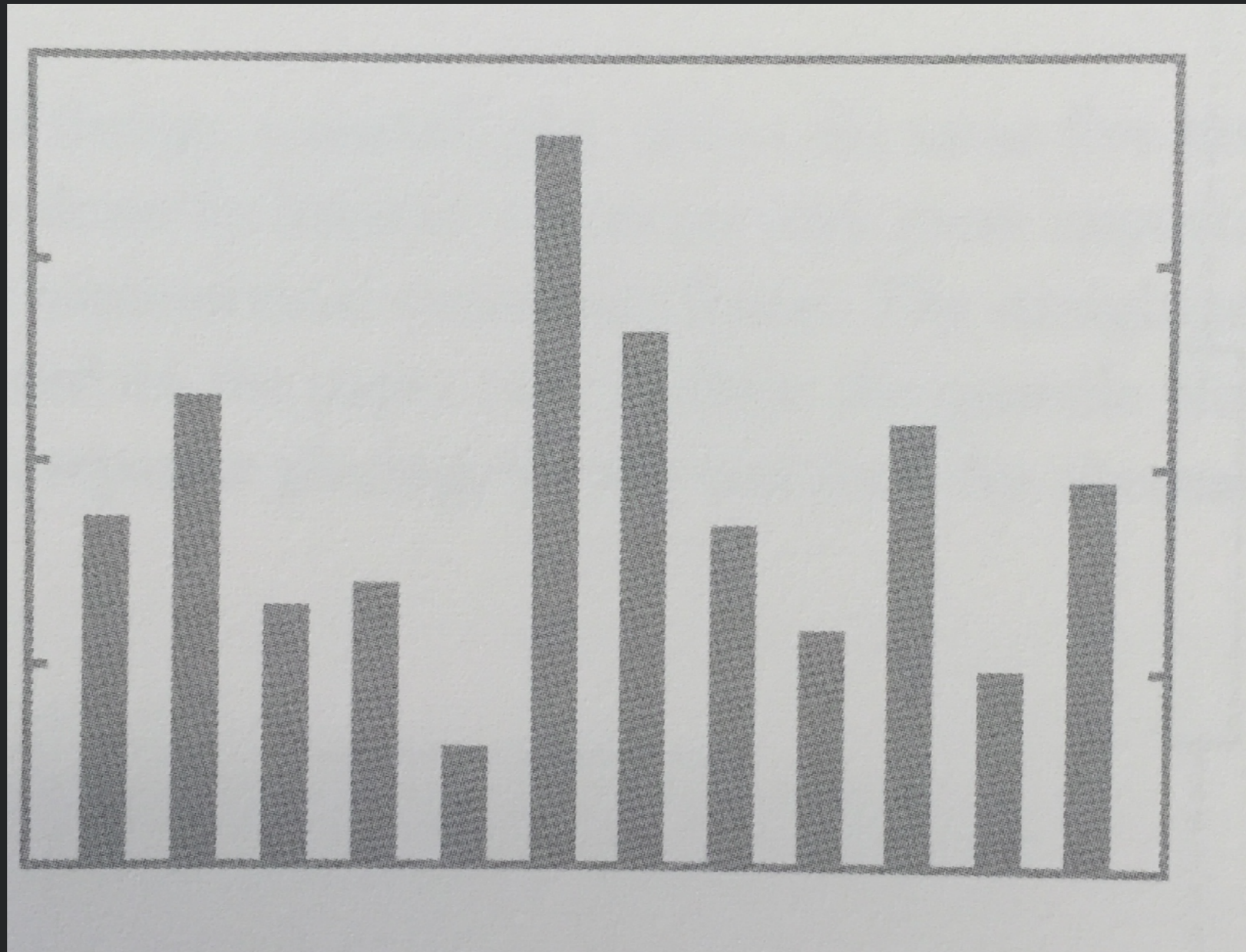
1.0



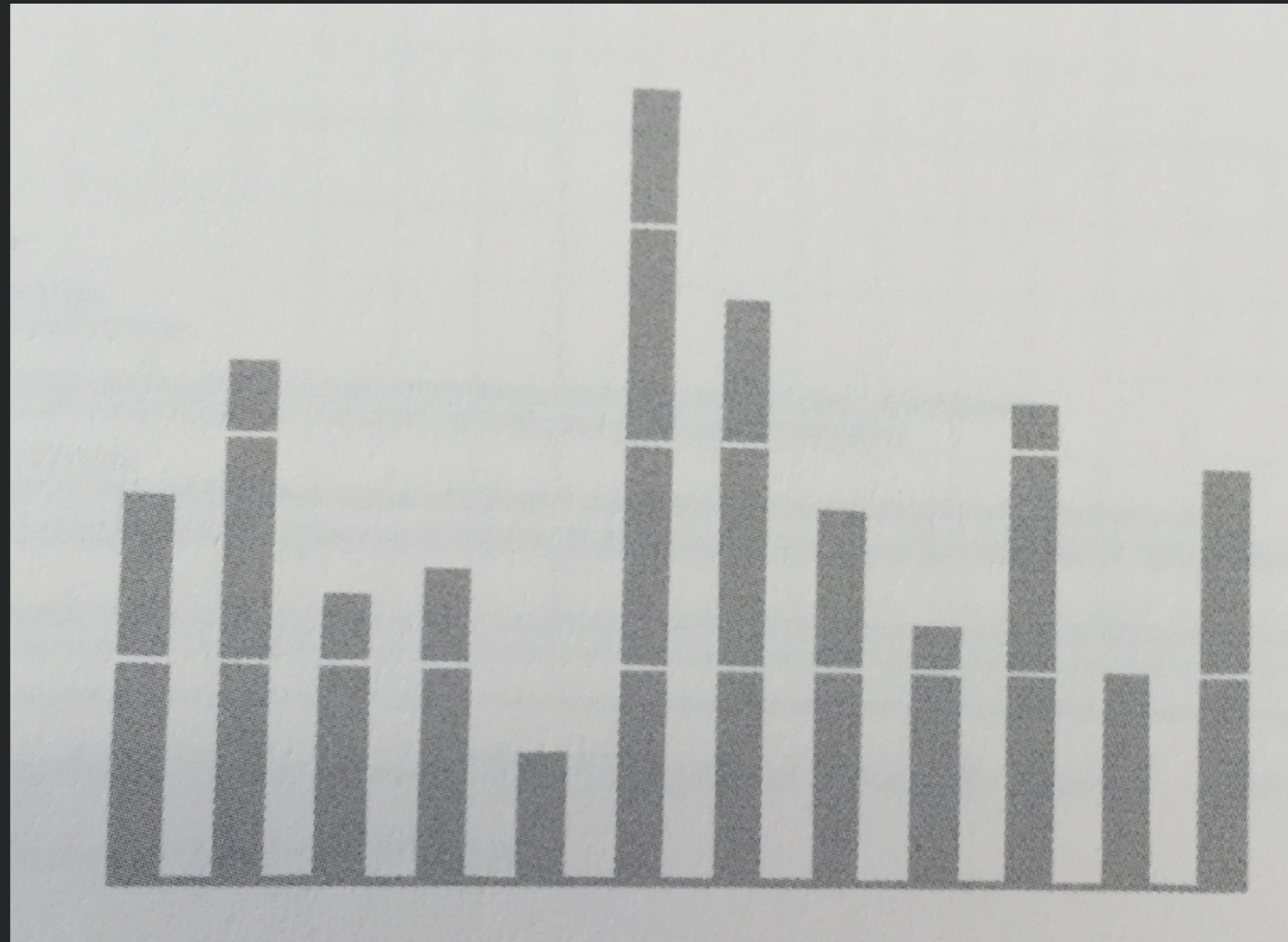
Design Principles for Data-ink

- (a.k.a. aesthetics & minimalism / elegance & simplicity)
- **Above all else show the data**
 - Erase non-data-ink, within reason
 - Often not valuable and distracting
 - Redundancy not usually useful

Example



Example (revised)



Interacting with Visualizations





Interactive Visualizations

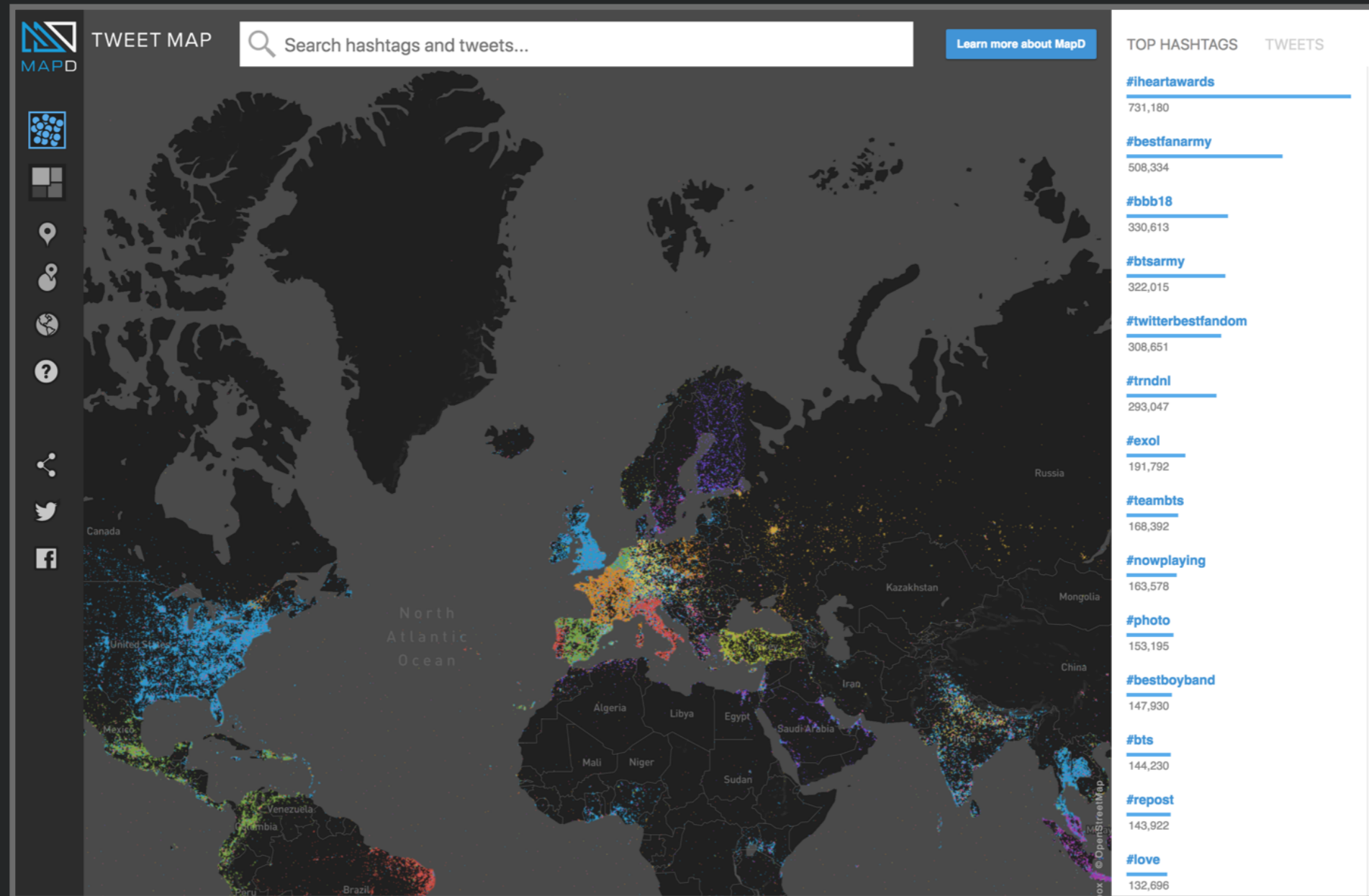
- Users often use iterative process of making sense of the data
 - Answers lead to new questions
- Interactivity helps user constantly change display of information to answer new questions
- Should offer visualization that offers best view of data moment to moment as desired view changes



Information Visualization Tasks

- **Overview:** gain an overview of entire collection
- **Zoom:** zoom in on items of interest
- **Filter:** filter out uninteresting items
- **Details on Demand:** select an item or group and get details
- **Relate:** view relationships between items
- **History:** support undo, replay, progressive refinement
- **Extract:** allow extraction of sub-collections through queries

Global Tweet Map



<https://www.mapd.com/demos/tweetmap/>

Renting vs. Buying Utility

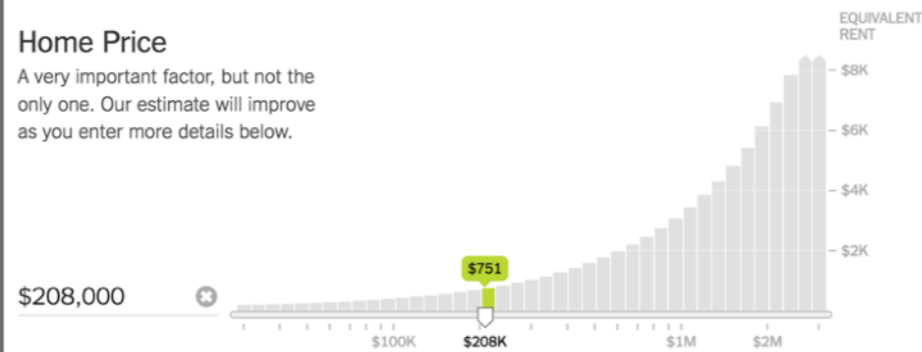
Is It Better to Rent or Buy?

By MIKE BOSTOCK, SHAN CARTER and ARCHIE TSE

The choice between buying a home and renting one is among the biggest financial decisions that many adults make. But the costs of buying are more varied and complicated than for renting, making it hard to tell which is a better deal. To help you answer this question, our calculator takes the most important costs associated with buying a house and computes the equivalent monthly rent. [RELATED ARTICLE](#)

Home Price

A very important factor, but not the only one. Our estimate will improve as you enter more details below.



If you can rent a similar home for less than ...

\$751 PER MONTH

... then renting is better.

Costs after 9 years	Rent	Buy
Initial costs	\$751	\$49,920
Recurring costs	\$90,937	\$137,912
Opportunity costs	\$13,092	\$37,376
Net proceeds	-\$751	-\$121,180
Total	\$104,029	\$104,029

How Long Do You Plan to Stay?

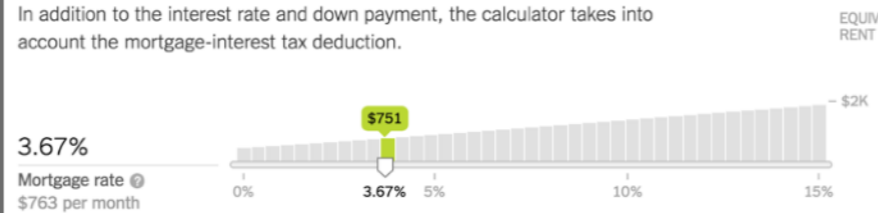
Buying tends to be better the longer you stay because the upfront fees are spread out over many years.



How to Read the Charts Charts that are relatively flat indicate factors that are not particularly important to the outcome. Conversely, the factors that have steep slopes have a large impact.

What Are Your Mortgage Details?

In addition to the interest rate and down payment, the calculator takes into account the mortgage-interest tax deduction.



SWE 432 - Web Application Development



George Mason
University

Instructor:
Dr. Kevin Moran

Teaching Assistant:
Oyindamola Oluyemo

Class will start in:
10:00



Final Exam

- Tuesday, May 16th, 4:30pm-7:10pm, This Room
- 3 Parts, In-class exam, closed book, 200 points total
 - **Part 1:** Multiple Choice Questions
 - **Part 2:** Short Answer
 - Either provide program output, or answer in a few short sentences
 - **Part 3:** Multi-Part Code Question (*implementing a simple Front-End in React*)
- Covers material from weeks 7-14, from both lectures and readings



Multiple Choice

- 75 Points -15 Questions
 - ~5 on React/CSS
 - ~2 on Deployment
 - ~8 on User-Centered Design
- Will Cover likely question material today



Short Answer

- 75 Points - 5 Questions
 - 1 Question on HTML/CSS (What is output)
 - 1 Question on Sketching and Prototyping
 - 1 Question on applying Heuristic Evaluation
 - 1 Question on Critiquing Visual Design
 - 1 Question on UI Evaluations
- Will Cover likely question material today



Coding Question

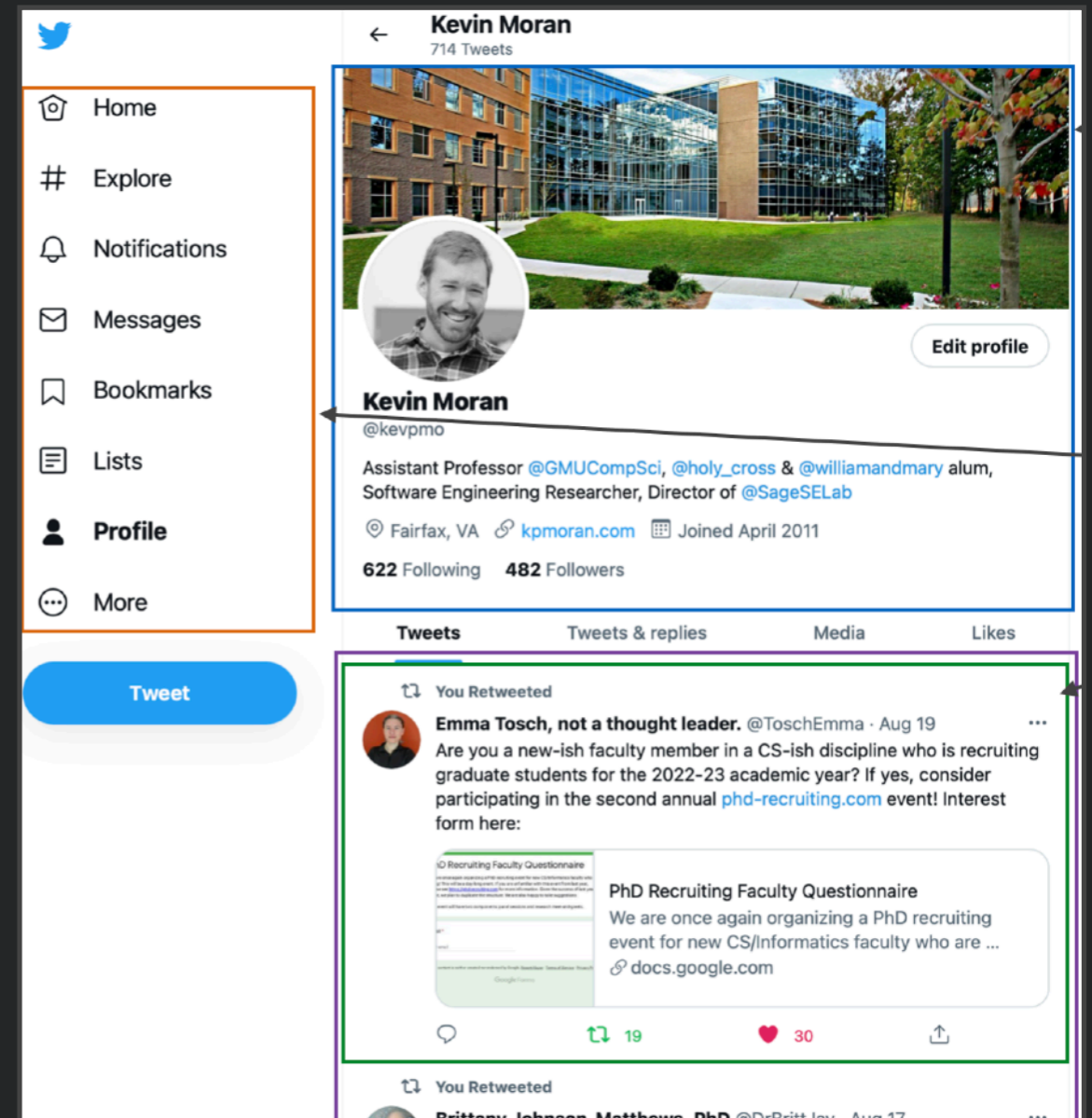
- 50 Points - 1 Question (Multiple Parts)
 - Implement a React Component
 - Will have interaction between two components
 - We will provide Example Code
 - You are free to use either Class-based or Functional Components (example code will be provided for both)
- I will provide an example question

Week 10 - React



Review: Components

- Web pages are complex, with lots of logic and presentation
- How can we organize web page to maximize modularity?
- Solution: Components
 - Templates that correspond to a specific widget
 - Encapsulates related logic & presentation using language construct (e.g., class)





Anatomy of a React Component

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }
```

```
  handleClick() {
    this.setState(prevState => ({ isToggleOn: !prevState.isToggleOn }));
  }
```

```
  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

```
ReactDOM.render(
  <Toggle />, document.getElementById('root')
);
```



What is state?

- All internal component data that, when changed, should trigger UI update
 - Stored as single JSON object `this.state`
- What isn't state?
 - Anything that could be computed from state (redundant)
 - Other components - should build them in render
 - Data duplicated from properties.



Properties vs. State

- Properties should be *immutable*.
 - Created through attributes when component is instantiated.
 - Should *never* update within component
 - Parent may create a new instance of component with new properties

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

- State *changes* to reflect the current state of the component.
 - Can (and should) change based on the current internal data of your component.



Working with State

- Constructor should initialize state of object

```
constructor(props) {  
  super(props);  
  this.state = {date: new Date()};  
}
```

- Use `this.setState` to update state

```
this.setState({  
  date: new Date()  
});
```

- Doing this (asynchronously) will eventually result in render being invoked
 - Multiple state updates may be batched together and result in a single render call (handled by the framework)



Handling Events

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(prevState => ({ isToggleOn: !prevState.isToggleOn }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}

ReactDOM.render(
  <Toggle />, document.getElementById('root')
);
```

Event Dispatching

- Each event target can have (0...n) listeners registered for any given event type, called in arbitrary order
- What happens with nested elements?

body

form

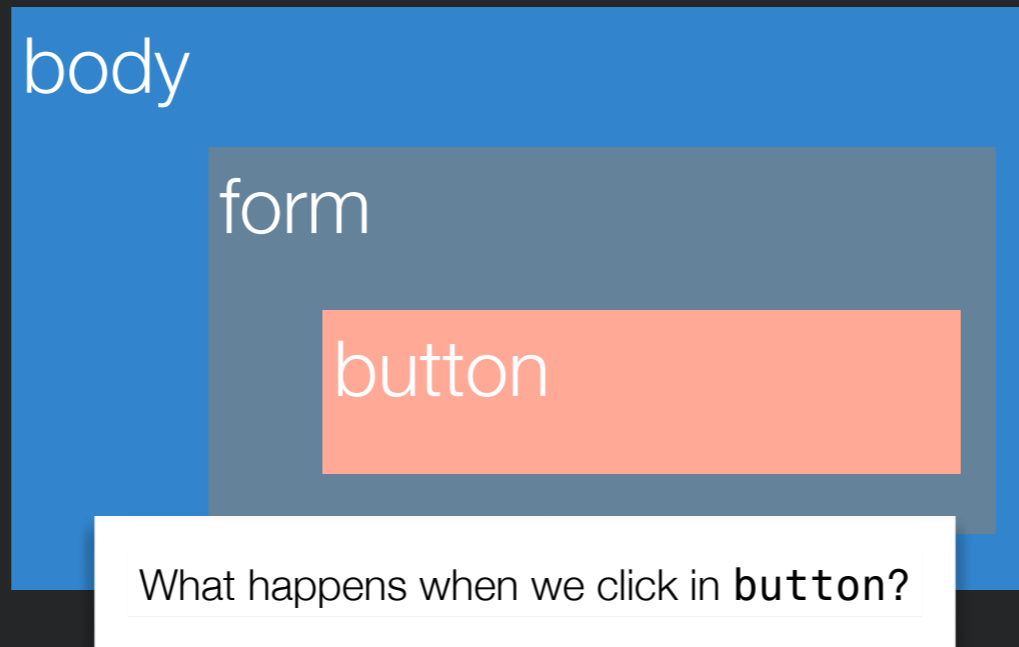
button

```
Listener1: body onClick  
Listener2: form onClick  
Listener3: button onClick
```

What happens when we click in `button`?



Event Bubbling



Called



```
Listener1: body onClick  
Listener2: form onClick  
Listener3: button onClick
```

This is the default behavior



Event Capturing

body

form

button

What happens when we click in **button**?

Called



Listener1: body onClick

Listener2: form onClick

Listener3: button onClick

Enable event capturing when you register your listener:
`element.addListener('click', myListener, true);`



Event Dispatching

- An individual listener can stop bubbling/capturing by calling
- `event.stopPropagation();`
 - Assuming that `event` is the name of your handler's parameter

body

form

button

```
Listener1: body onClick  
Listener2: form onClick  
Listener3: button onClick
```



The Data Flows Down

- State that is common to multiple components should be owned by a common ancestor
 - State can be passed into descendants as properties
- When this state can be manipulated by descendants (e.g., a control), change events should invoke a handler on common ancestor
 - Handler function should be passed to descendants

<https://reactjs.org/docs/state-and-lifecycle.html#the-data-flows-down>



The Data Flows Down

```
class Calculator extends React.Component {
  constructor(props) {
    super(props);
    this.handleCelsiusChange = this.handleCelsiusChange.bind(this);
    this.state = {temperature: '', scale: 'c'};
  }

  handleCelsiusChange(temperature) {
    this.setState({scale: 'c', temperature});
  }

  render() {
    const scale = this.state.scale;
    const temperature = this.state.temperature;
    const celsius = scale === 'f' ? tryConvert(temperature, toCelsius) : temperature;

    return (
      <div>
        <TemperatureInput
          scale="c"
          temperature={celsius}
          onTemperatureChange={this.handleCelsiusChange} />
      </div>
    );
  }
}
```



Nesting components

```
render() {  
  return (  
    <div>  
      <PagePic pagename={this.props.pagename} />  
      <PageLink pagename={this.props.pagename} />  
    </div>  
  );  
}
```

Establishes ownership by creating
in render function.

Sets pagename property of child to
value of pagename property of
parent



Controlled Components

```
class EssayForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: 'Please write an essay about your favorite DOM element.'
    };

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('An essay was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <textarea value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```




Controlled Components

- Single source of truth
- Whenever a control changes its value
 - React is notified
 - State is updated
- Whenever state is updated
 - If necessary, render function executes and generates control with new value



Reconciliation

```
<Card>  
  <p>Paragraph 1</p>  
  <p>Paragraph 2</p>  
</Card>
```

```
<Card>  
  <p>Paragraph 2</p>  
</Card>
```

- Process by which React updates the DOM with each new render pass
- Occurs based on order of components
 - Second child of Card is destroyed.
 - First child of Card has text mutated.

<https://reactjs.org/docs/reconciliation.html>



Reconciliation with Keys

- Problem: what if children are dynamically generated and have their own state that must be persisted across render passes?
 - Don't want children to be randomly transformed into other child with different state
- Solution: give children identity using keys
 - Children with keys will always keep identity, as updates will reorder them or destroy them if gone

Keys



```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```



Functional Components + Hooks

But what if we want state + clean functional components??

```
import React, { Component } from 'react';

class Counter extends Component {
  constructor(props) {
    super(props);

    this.state = {
      count: 0,
    };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button
          onClick={() =>
            this.setState({ count: this.state.count + 1 })
          }
        >
          Click me
        </button>
      </div>
    );
  }
}

export default Counter;
```




Functional Components + Hooks

Now we can have both with functional components + hooks!

```
import React from 'react';

// how to use the state hook in a React function component
function Counter() {
  const [count, setCount] = React.useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default Counter;
```

Week 11 - React & CSS



CSS: Cascading Style Sheets

- Language for styling documents

```
p {
  font-family: Arial;
}
```

Property Value



“Select all <p> elements”

Selector describes a *set* of HTML elements

“Use Arial font family”

Declaration indicates how selected elements should be styled.

- Separates *visual presentation* (CSS) from document structure (HTML)
 - Enables changes to one or the other.
 - Enables styles to be reused across sets of elements.

CSS Type Selectors

- What if we wanted more green?

```
h2, h3 {  
  color: LightGreen;  
}
```

“Select all <h2> and <h3> elements”

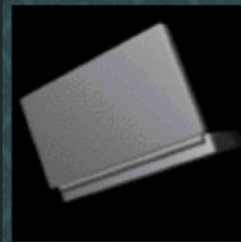
Type selector selects one or more element types.

```
* {  
  color: LightGreen;  
}
```

“Select all elements”

Universal selector selects all elements.

Prof Kevin Moran



This is Prof Moran's ACTUAL homepage from 1991

Welcome, students!

[See how to make this page](#)

Some funny links

- [Homestar Runner](#)
- [Hamster Dance](#)

About Prof Moran

Prof Moran's office is at 4442 Engineering Building. His email address is kpmoran@gmu.edu.

Last updated: September 28th, 1999

CSS Class Selectors

```

```

“Label element with imageLarge class”

```
.imageLarge {  
  width: 200px;  
  height: 200px;  
}
```

“Define class imageLarge.”

```

```

```
img.large {  
  width: 200px;  
  height: 200px;  
}
```

“Define large class that applies only to elements”

```
.transparent {  
  opacity: .50;  
}
```

“Define transparent class”

Classes enable the creation of sets of elements that can be styled in the same way.

CSS id Selectors

```
<div id="exampleElem">      #exampleElem {
    Some text                font-weight: bold;
</div>                      }
```

Some text

- Advantages

- Control presentation of individual elements

- Disadvantages

- Must write separate rule for *each* element



CSS Selectors

- **Key principles in designing effective styling rules:**
 - Use classes, semantic tags to create sets of elements that share a similar rules
 - Don't repeat yourself (DRY)
 - Rather than create many identical or similar rules, apply single rule to all similar elements
 - Match based on semantic properties, not styling
 - Matching elements based on their pre-existing styling is *fragile*



Cascading Selectors

- What happens if more than one rule applies?
- Most *specific* rule takes precedence
 - *p b* is more specific than *p*
 - *#maximizeButton* is more specific than *button*
- If otherwise the same, *last* rule wins
- Enables writing generic rules that apply to many elements that are overridden by specific rules applying to a few elements



CSS Inheritance

- When an element is contained inside another element, some styling properties are inherited
 - e.g., font-family, color
- Some properties are not inherited
 - e.g., background-color, border
- Can force many properties to inherit value from parent using the inherit value
 - e.g., padding: inherit;

Pseudo Classes

```
.invisible {  
  display: none;  
}
```

```
input:invalid {  
  border: 2px solid red;  
}
```

```
input:invalid + div {  
  display: block;  
}
```

```
input:focus + div {  
  display: none;  
}
```

```
<label>  
  Email: <input type="email" />  
  <div class="invisible">Please enter a valid email.</div>  
</label>
```

Email:

“Select elements with the invalid attribute.”

“Select elements that have focus.”

Classes that are automatically attached to elements based on their attributes.



Examples of Pseudo Classes

- :active - elements activated by user. For mouse clicks, occurs between mouse down and mouse up.
- :checked - radio, checkbox, option elements that are checked by user
- :disabled - elements that can't receive focus
- :empty - elements with no children
- :focus - element that currently has the focus
- :hover - elements that are currently hovered over by mouse
- :invalid - elements that are currently invalid
- :link - link element that has not yet been visited
- :visited - link element that has been visited

Color

- Can set text color (`color`) and background color (`background-color`)
- Several ways to describe color
 - six digit hex code (e.g., `#ee3e80`)
 - color names: 147 predefined names
 - `rgb(red, green, blue)`: amount of red, green, and blue
 - `hsla(hue, saturation, lightness, alpha)`: alternative scheme for describing colors
- Can set opacity (`opacity`) from 0.0 to 1.0

```
body {  
    color: Red;  
    background-color: rgb(200, 200, 200); }  
h1 {  
    background-color: DarkCyan; }  
h2 {  
    color: #ee3e80; }  
p {  
    color: hsla(0, 100%, 100%, 0.5); }  
div.overlay {  
    opacity: 0.5; }
```

Typefaces



im

Serif



im

Sans-Serif



im

Monospace



im

Cursive

font-family: Georgia, Times, serif;

“Use Georgia if available, otherwise Times, otherwise any serif font”.

font-family enables the typeface to be specified. The typeface must be installed. Lists of fonts enable a browser to select an alternative.

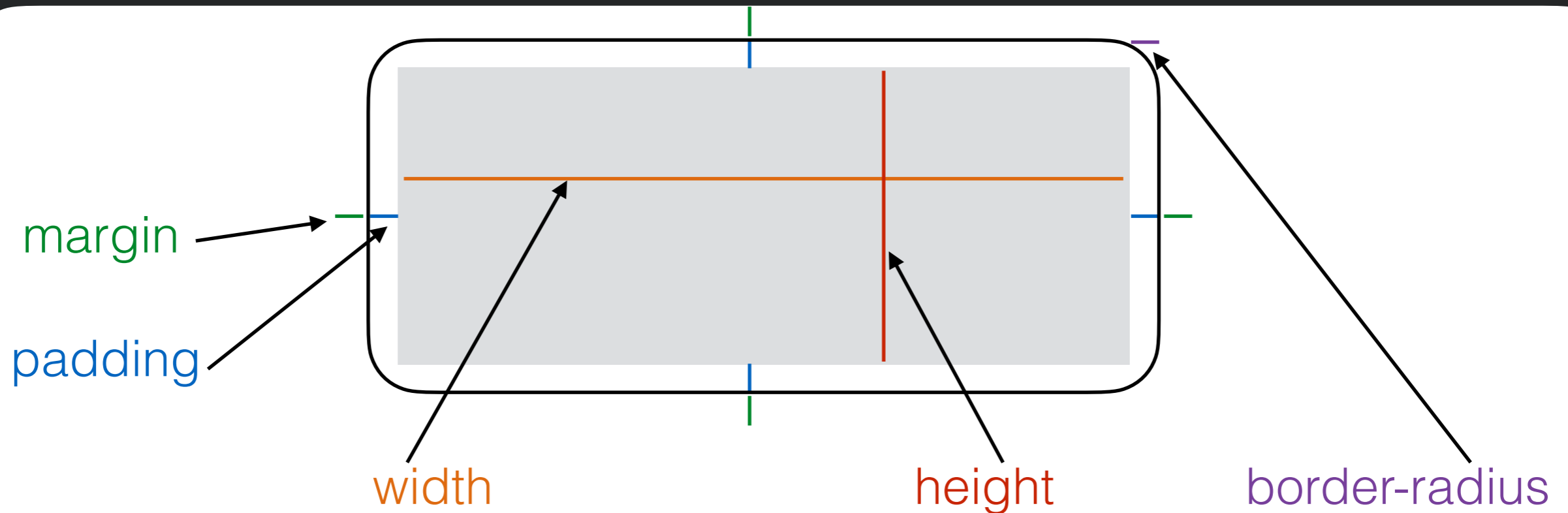
Styling text

```
h2 {  
  text-transform: uppercase;  
  text-decoration: underline;  
  letter-spacing: 0.2em;  
  text-align: center;  
  line-height: 2em;  
  vertical-align: middle;  
  text-shadow: 1px 1px 0 #666666;  
}
```

THIS TEXT IS IMPORTANT

- text-transform: uppercase, lowercase, capitalize
- text-decoration: none, underline, overline, line-through, blink
- letter-spacing: space between letters (kerning)
- text-align: left, right, center, justify
- line-height: total of font height and empty space between lines
- vertical-align: top, middle, bottom, ...
- text-shadow: [x offset][y offset][blur offset][color]

CSS "Box" Model



- Boxes, by default, are sized *just* large enough to fit their contents.
- Can specify sizes using `px` or `%`
 - `%` values are relative to the container dimensions
- `margin: 10px 5px 10px 5px;` (clockwise order - [top] [right] [bottom] [left])
- `border: 3px dotted #0088dd;` ([width] [style] [color])
 - style may be: solid, dotted, dashed, double, groove, ridge, inset, outset, hidden / none

Centering Content

```
.centered {  
  width: 300px;  
  margin: 10px auto 10px auto;  
  border: 2px solid #0088dd;  
}
```

This box is centered in its container.

- How do you center an element inside a container?
- Step 1: Must first ensure that element is *narrower* than container.
 - By default, element will expand to fill entire container.
 - So must usually explicitly set width for element.
- Step 2: Use *auto* value for left and right to create equal gaps

Visibility and layout

- Can force elements to be inline or block element.
 - display: inline
 - display: block
- Can cause element to not be laid out or take up any space
 - display: none
 - *Very* useful for content that is dynamically added and removed.
- Can cause boxes to be invisible, but still take up space
 - visibility: hidden;

```
<ul>
  <li>Home</li>
  <li>Products</li>
  <li class="coming-soon">Services</li>
  <li>About</li>
  <li>Contact</li>
</ul>
```

```
li {
  display: inline;
  margin-right: 10px; }
li.coming-soon {
  display: none; }
```

Home Products About Contact

```
li {
  display: inline;
  margin-right: 10px; }
li.coming-soon {
  visibility: hidden; }
```

Home Products About Contact

Transitions

```
.box {  
  width: 100px;  
  height: 100px;  
  background-color: #0000FF;  
  transition: width 2s, height 2s, background-color 2s, transform 2s;  
}
```



```
.box:hover {  
  background-color: #FFCCCC;  
  width: 200px;  
  height: 200px;  
  transform: rotate(180deg);  
}
```

```
<div class="box"></div>
```

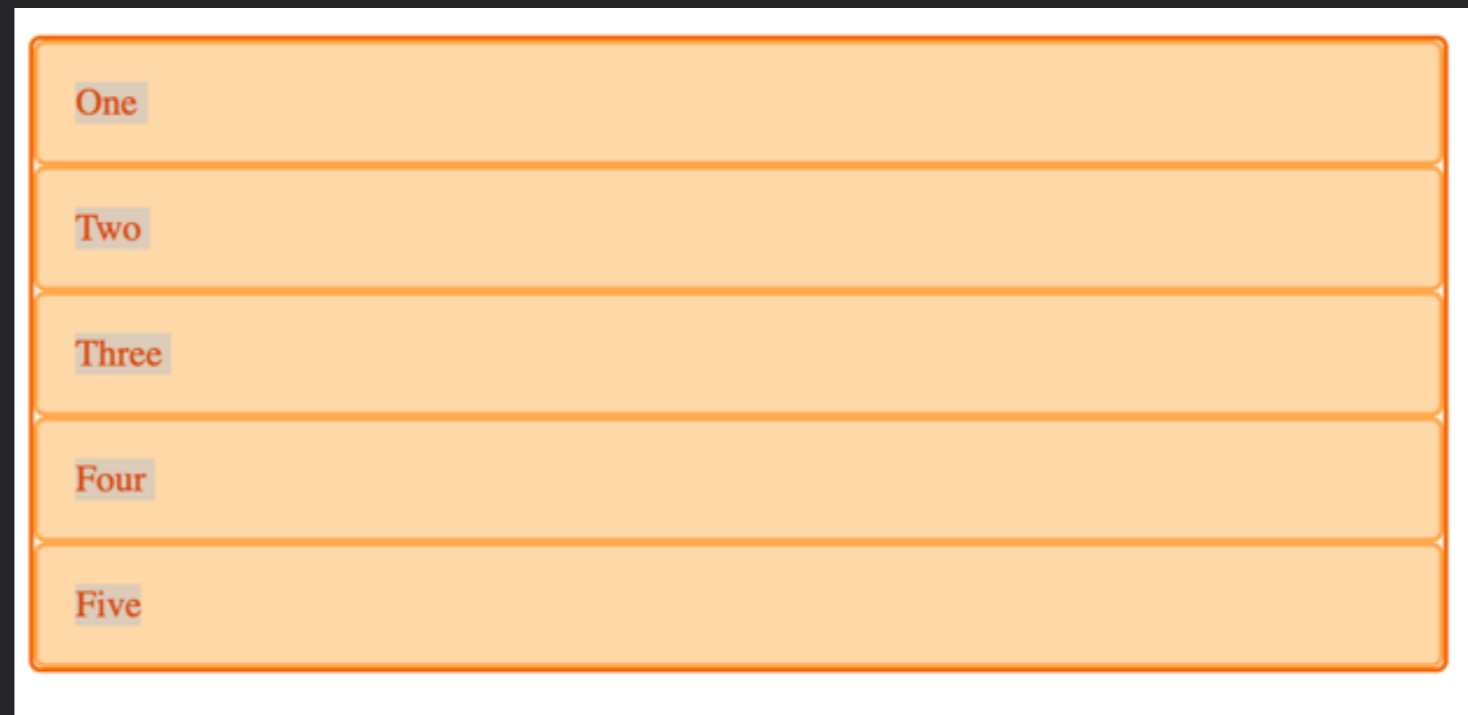
- transition: [property time], ..., [property time]
 - When new class is applied, specifies the time it will take for each property to change
 - Can use *all* to select all changed properties

Grid layout

- Create using display: grid or display: inline-grid

```
<div class="wrapper">  
  <div>One</div>  
  <div>Two</div>  
  <div>Three</div>  
  <div>Four</div>  
  <div>Five</div>  
</div>
```

```
.wrapper {  
  display: grid;  
}
```



Grid tracks

- Define rows and columns on grid with the *grid-template-columns* and *grid-template-rows* properties.
- Define grid tracks.
- A grid track is the space between any two lines on the grid.

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>

.wrapper {
  display: grid;
  grid-template-columns: 200px 200px 200px;
}
```



Liquid layouts

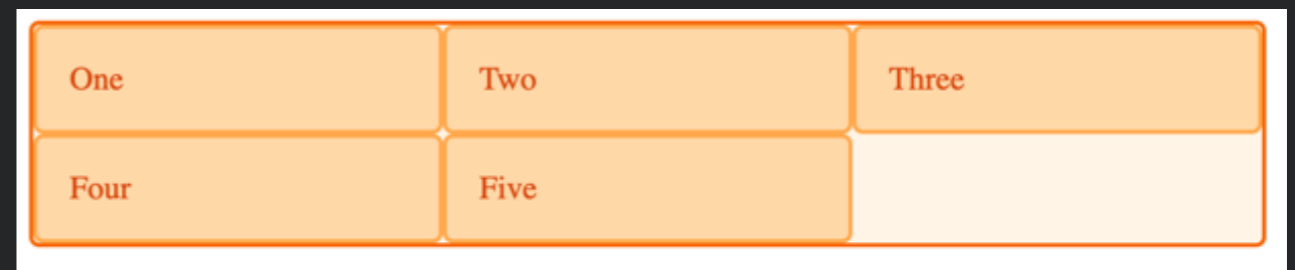
- `fr` represents a fraction of available space for grid container.
- Can mix absolute and flexible, where flexible occupies any remaining space after flexible is subtracted

```

<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>

.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}

```



```

.wrapper {
  display: grid;
  grid-template-columns: 500px 1fr 2fr;
}

```

Liquid layouts

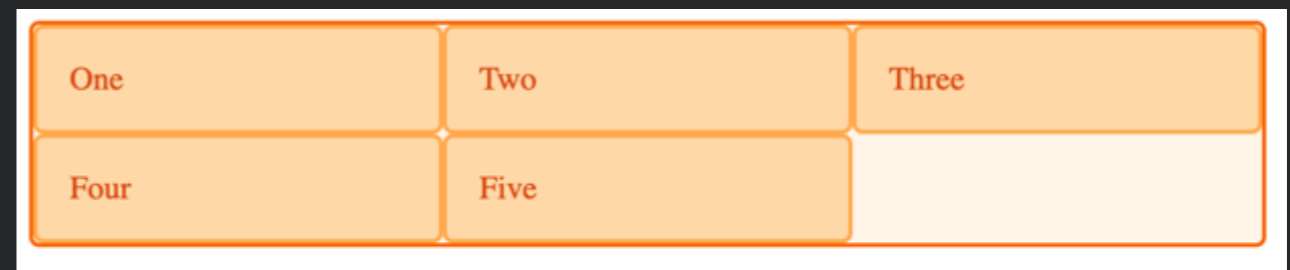
- fr represents a fraction of available space for grid container.
- Can mix absolute and flexible, where flexible occupies any remaining space after flexible is subtracted

```

<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>

.wrapper {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}

```



```

.wrapper {
  display: grid;
  grid-template-columns: 500px 1fr 2fr;
}

```


Positioning items

- Can explicitly place elements inside grid into grid areas

```
<div class="wrapper">
  <div class="box1">One</div>
  <div class="box2">Two</div>
  <div class="box3">Three</div>
  <div class="box4">Four</div>
  <div class="box5">Five</div>
</div>
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: 100px;
}
.box1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
}
.box2 {
  grid-column-start: 1;
  grid-row-start: 3;
  grid-row-end: 5;
}
```



- Can set gaps between columns and rows

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>

.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  column-gap: 10px;
  row-gap: 1em;
}
```

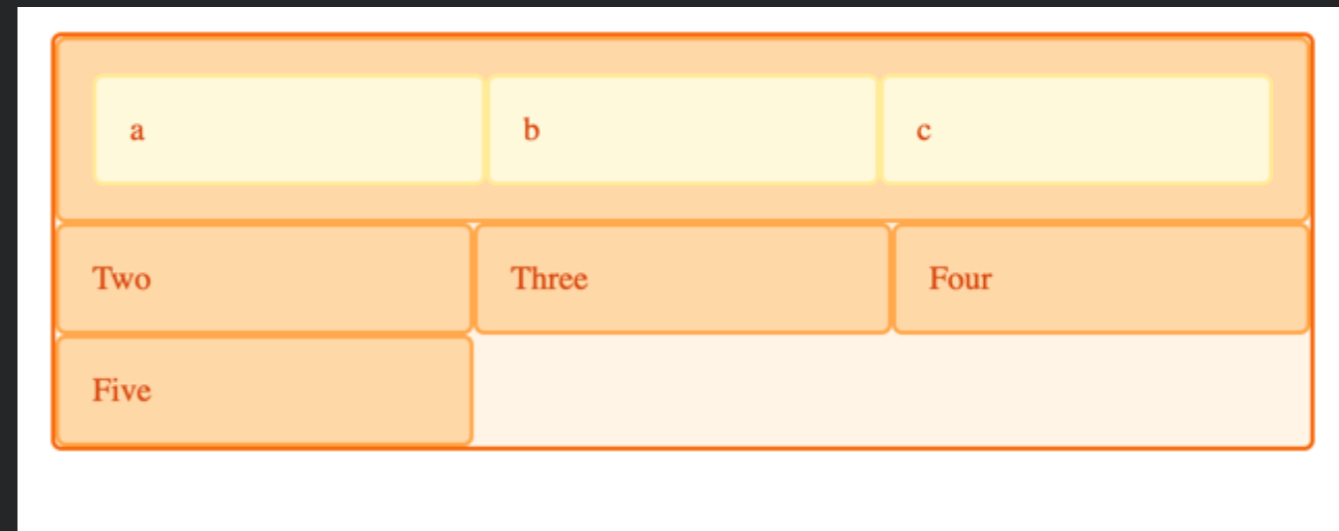


Nesting

- Can nest grids, which behave just like top-level

```
<div class="wrapper">
  <div class="box box1">
    <div class="nested">a</div>
    <div class="nested">b</div>
    <div class="nested">c</div>
  </div>
  <div class="box box2">Two</div>
  <div class="box box3">Three</div>
  <div class="box box4">Four</div>
  <div class="box box5">Five</div>
</div>

.box1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
```





CSS Exercise

- <https://replit.com/@kmoran/swe-432-react-example#src/App.jsx>
 - Center a component inside it's container
 - Use a display grid to create layout with multiple rows and columns
 - Override one of the Bootstrap selectors

Week 12 - User Centered Design & Sketching + Prototyping



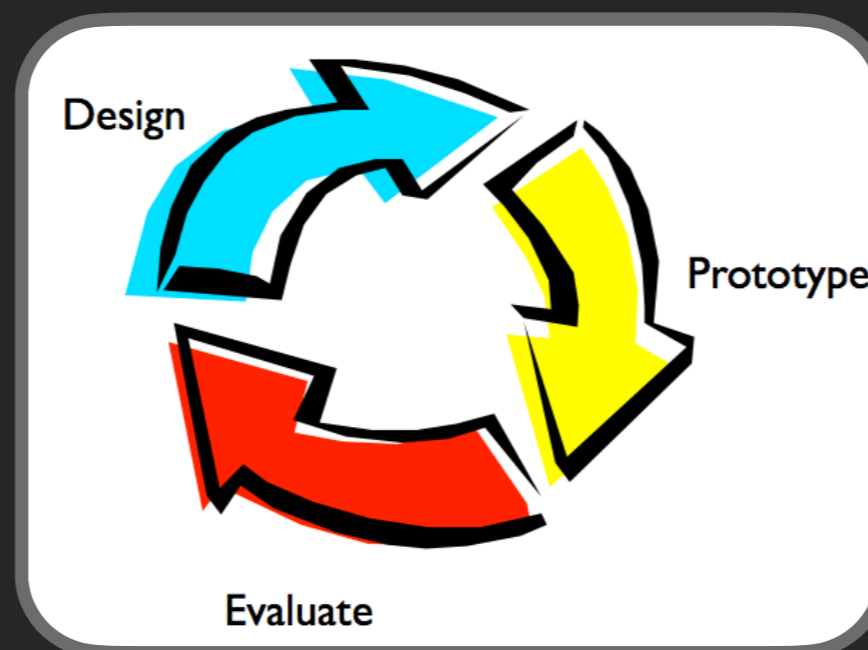


Usability

- A property of the relationship between
 - humans with goal-driven tasks
 - an artifact
- The speed and success with which the goals can be accomplished (task *performance*)

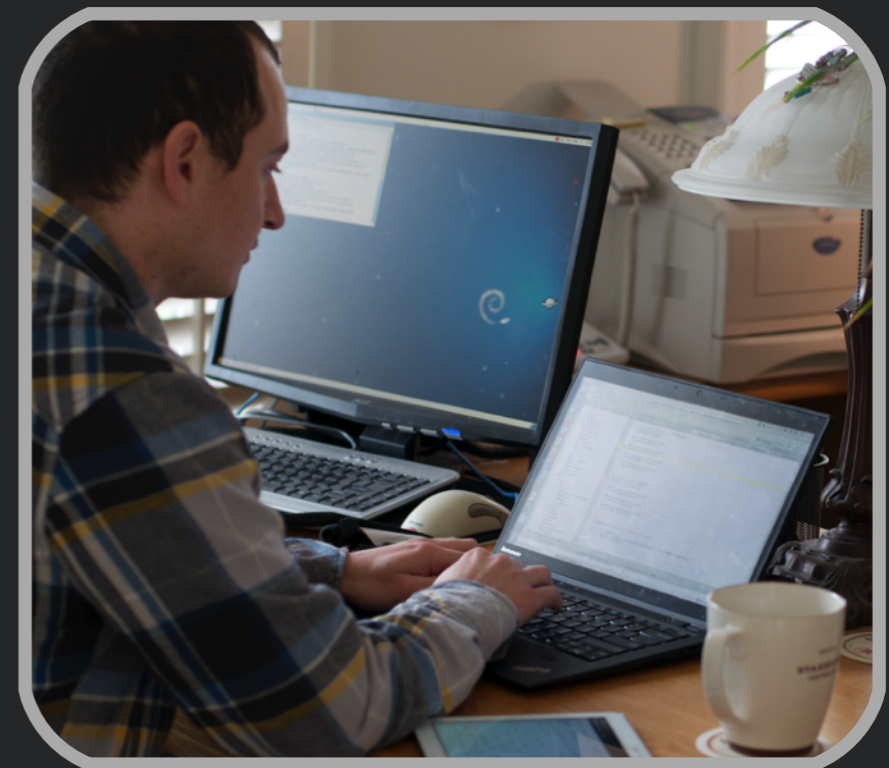
Iterative User-Centered Design

- Given humans with goals and tasks, redesign an existing artifact that helps to accomplish these tasks faster and more successfully

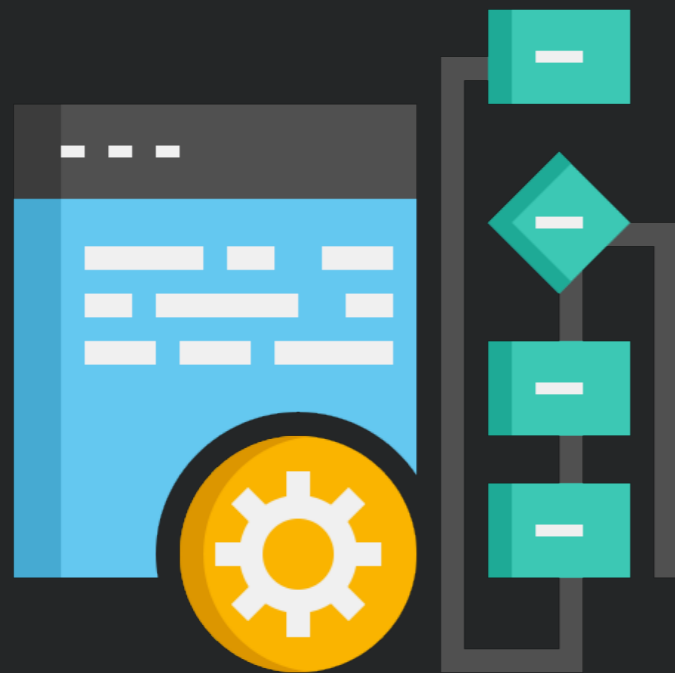


Empirical: Usability Study

- Given humans with goals and tasks an artifact, *observe humans* to identify usability issues that decrease task performance
- *“Ground Truth”*



Analytical: Usability Principles



- Given humans with goals and tasks and an artifact, *assess for conformance to UI principles* to identify usability issues that decrease task performance
- *Approximation of “ground truth”*

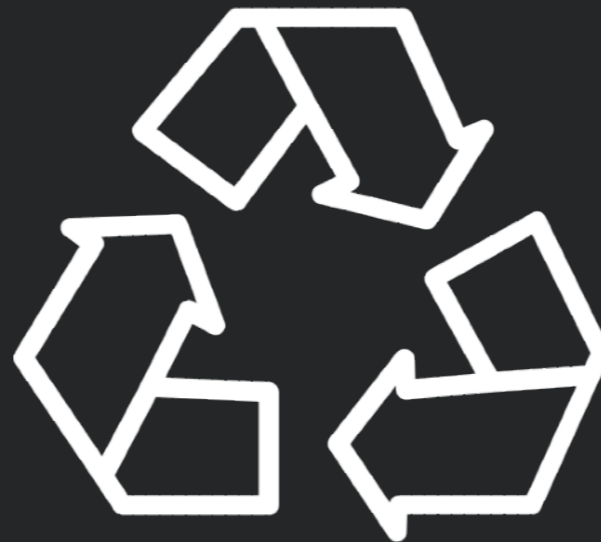
Iterative Model of User-Centered Design

Observation

(Re)Define the Problem
Understand User Needs

Test

Evaluate what
you have built



Idea Generation

Brainstorm
what to build

Prototype/ Implementation

Build



Heuristic Evaluation (Analytical)

- “*Discount* usability engineering methods” - Jakob Nielsen
- Involves a small team of evaluators to evaluate an interface based on recognized usability principles
- Heuristics – “rules of thumb”

Adapted from slides by Bonnie John and Jennifer Mankoff



Heuristic Evaluation

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition vs. recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, and recover from errors
10. Help and documentation



Advantages of Heuristic Evaluation

- “Discount usability engineering” - Intimidation low
- Don't need to identify tasks, activities
- Can identify some fairly obvious fixes
- Can expose problems user testing doesn't expose
- Provides a language for justifying usability recommendations



Disadvantages of Heuristic Evaluation

- Un-validated
- Do not employ real users
- Can be error prone
- Better to use usability experts
- Problems unconnected with tasks
- Heuristics may be hard to apply to new technology



Using Heuristic Evaluation

- Can be used informally to identify issues in a website
- Can be used as a more formal usability inspection method
- Evaluators each first separately identify issues
- Issues then combined from each evaluator

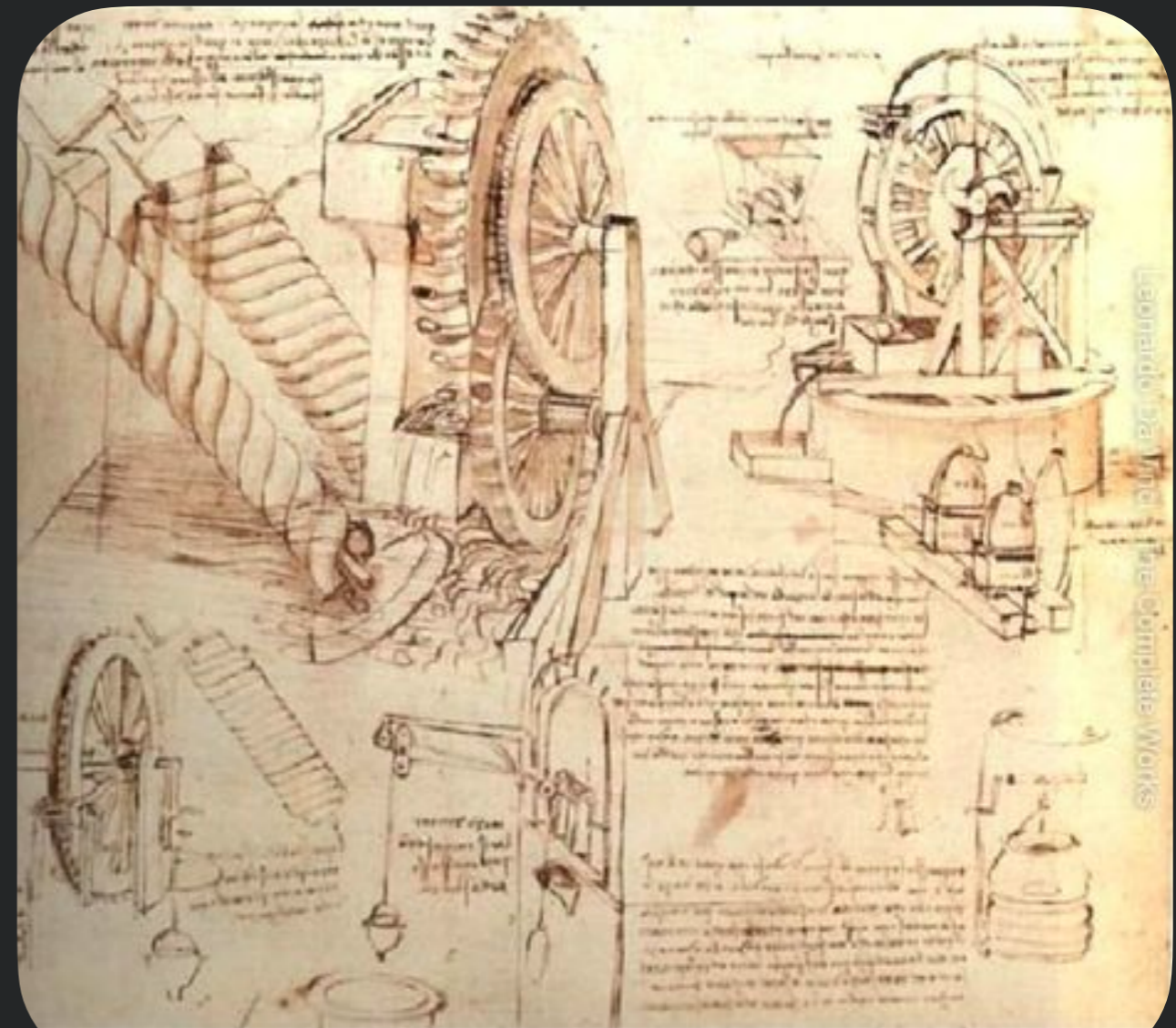


Ways to Use Heuristic Evaluation

- Early in design process to catch major issues
- When time or resources are not available for empirical usability evaluation

Why Sketch?

- Sketching offers visual medium for exploration, offering cognitive scaffolding to externalize cognition



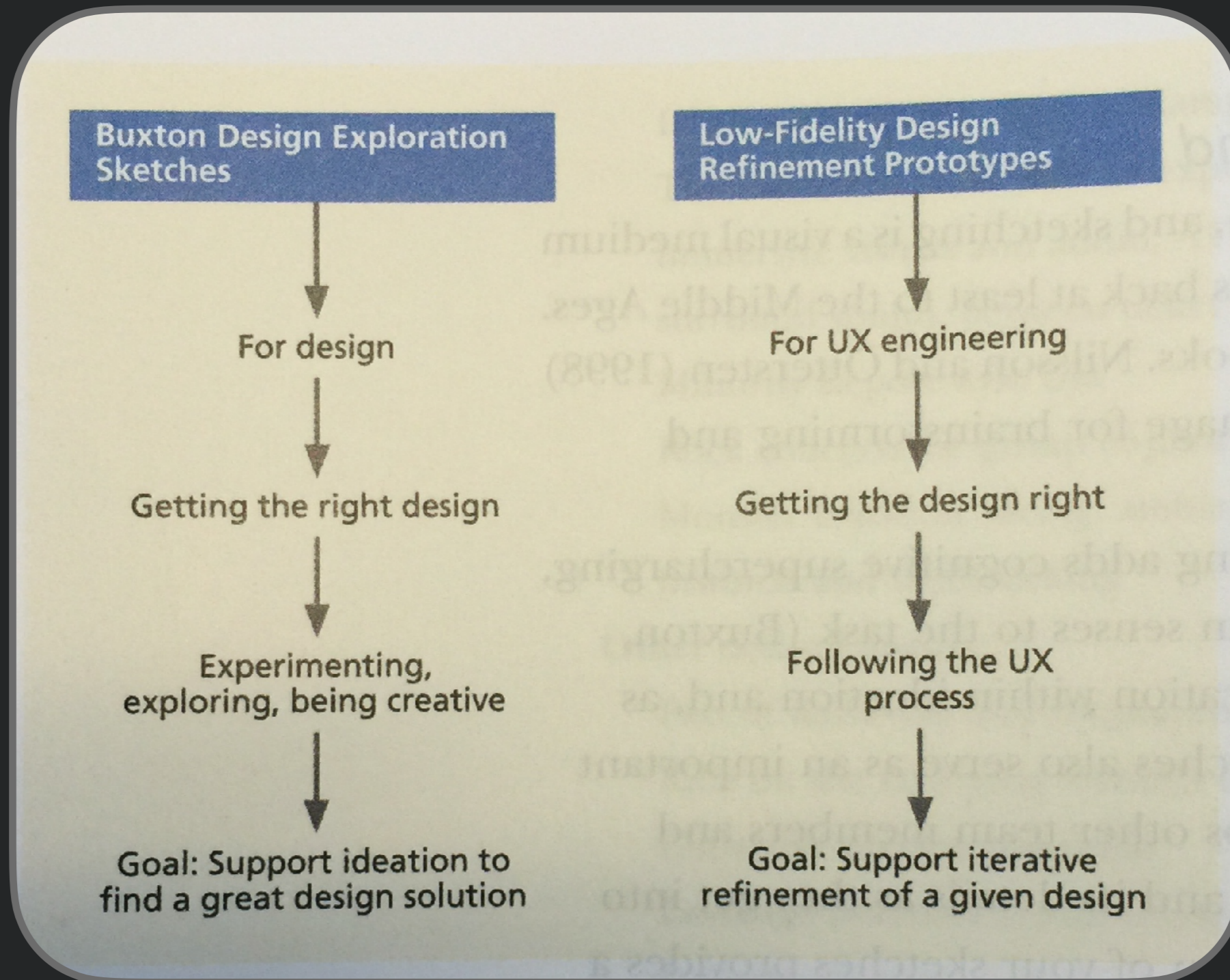
courtesy of www.leonardoda-vinci.org



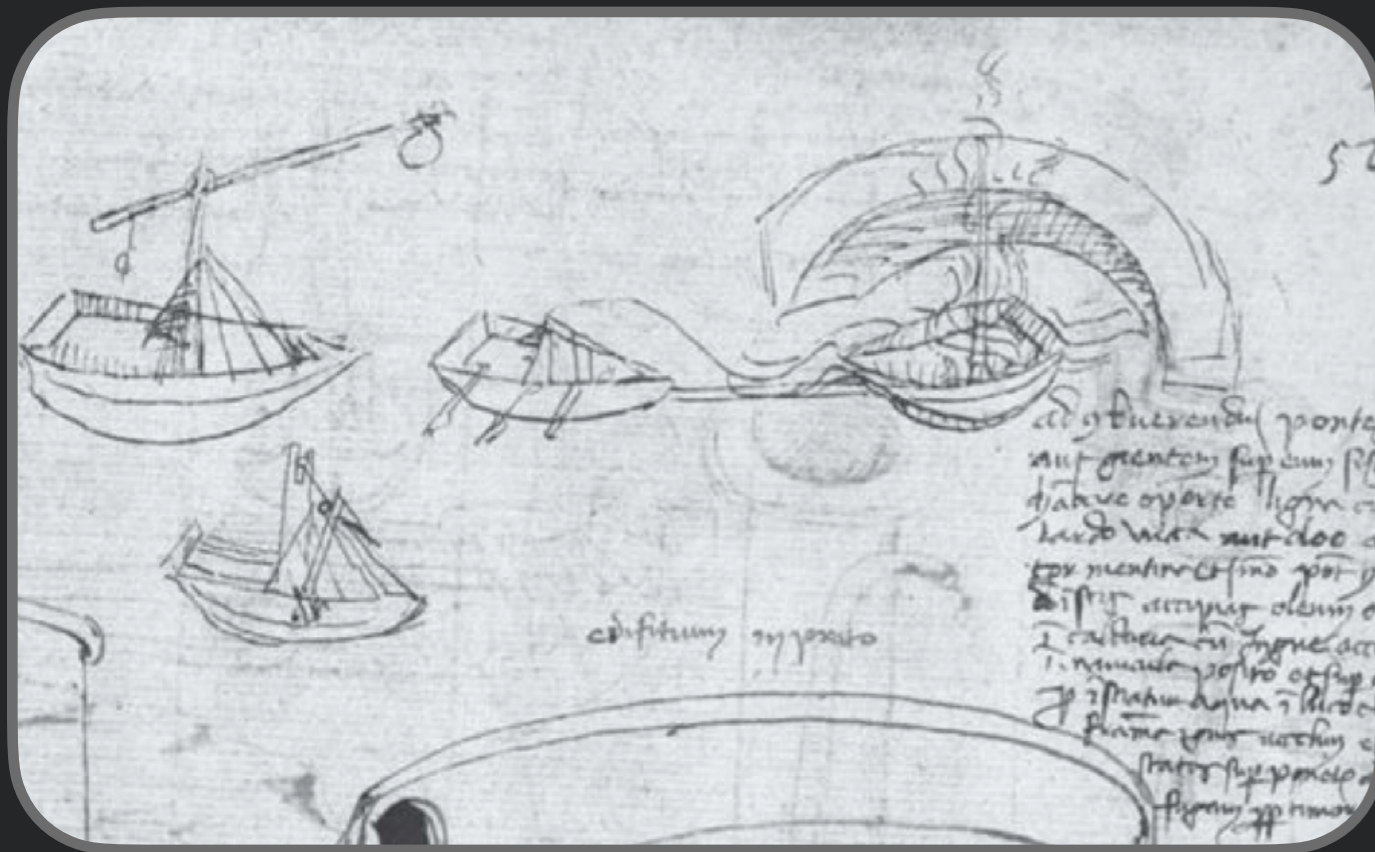
Being Creative with Sketches

- How do you come up with a great idea?
 - Generate lots of ideas
 - Work through ideas through externalization in sketch
 - Critique the ideas
 - Refine them to make them better
- Sketching offers a low-cost medium for working with early ideas **before** committing to one
- Design is process of creation & **exploration**

Sketching vs. Prototyping

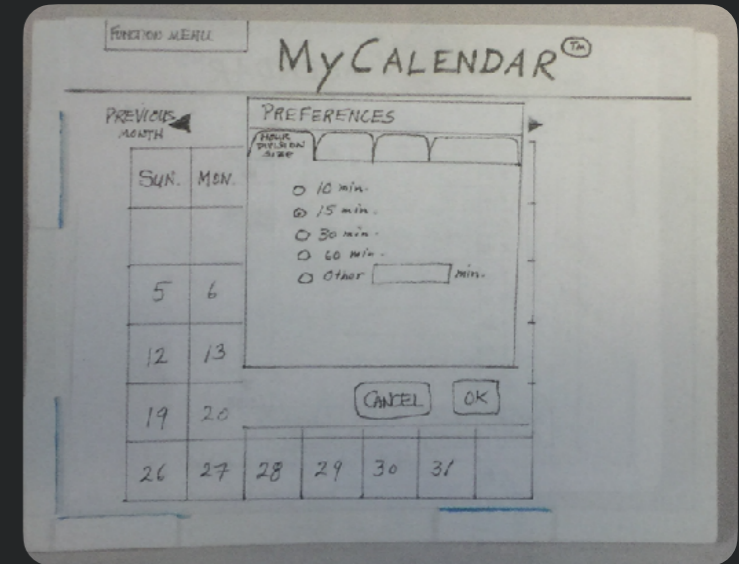
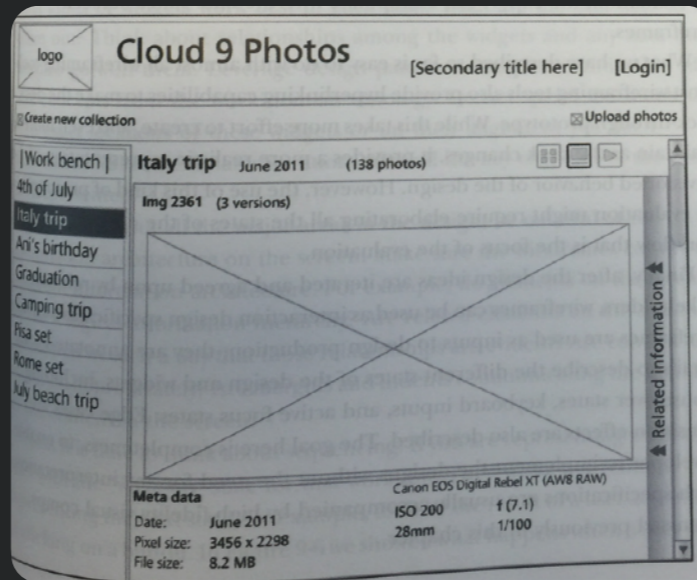


Sketches are Sketchy



- Not mechanically correct and perfectly straight lines
- **Freehand**, open gestures
- Strokes may miss connections
- Resolution & detail **low** enough to suggest is concept
- Deliberately **ambiguous** & abstract, leaving “holes” for imagination

Fidelity of Sketches & Mockups



Storyboard ————— Wireframe ————— Prototype

low

(many details left unspecified)

Fidelity

high

(more polished & detailed)



Storyboards for UI Design

- Sequence of visual “frames” illustrating *interplay* between user & envisioned system
- Explains how app fits into a larger *context* through a single scenario / story
- Bring design to *life* in graphical clips - freeze frame sketches of user interactions
- “Comic-book” style *illustration* of a scenario, with actors, screens, interaction, & dialog



Crafting a Storyboard

- Set the stage:
 - Who? What Where? Why? When?
- Show key interactions with application
- Show consequences of taking actions
- May also think about errors



Example Elements of a UI Storyboard

- Hand-sketched pictures annotated with a few words
- Sketch of user activity before or after interacting w/ system
- Sketches of devices & screens
- Connections with system (e.g., database connection)
- Physical user actions
- Cognitive user action in “thought balloons”



Frame Transitions

- Transitions between frames particularly important
- What users think, how users choose actions
- Many problems can occur here (e.g., gulfs of execution & evaluation) - we will talk more in a future class!
- Useful to think about how these work, can add thought bubbles to describe



Wireframes

- Lines & outlines (“wireframes”) of boxes & other shapes
- Capturing emerging interaction designs
- Schematic designs to define screen content & visual flow
- Illustrate approximate visual layout, behavior, transitions emerging from task flows
- Deliberate unfinished: do not contain finished graphics, colors, or fonts



Wireframes

- Can be used to step through a particular scenario
- Focus on key screens rather than every screen
- Tools can help
 - Can be made clickable
 - Can use stencils & templates; copy & edit similar screens

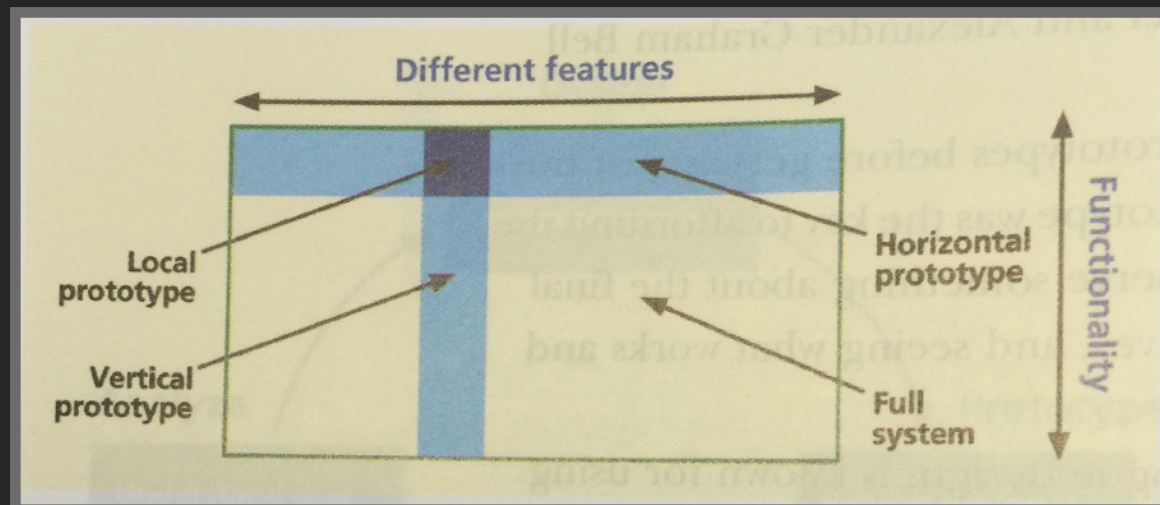


Prototyping

- How do you know your system design is right before you invest the time to build it?
- Answer: prototyping!
 - Evaluation performed **before** investing resources in building finished product
 - Early version of system constructed much **faster** & with less expense used to evaluate & **refine** design ideas

Types of Prototypes

- Which details do you leave out?
- **Horizontal**: *broad* in features, less depth
 - Explore overall concept of app, but not specific workflows
- **Vertical**: lots of *depth*, but only for a few features
 - Enables testing limited range of features w/ realistic user evals
- **T**: most of UI realized at low depth, few parts realized in depth
 - Combination of vertical & horizontal
- **Local**: focused prototype on *specific* interaction detail





Interactivity of Prototypes

- Scripted, click through prototypes
 - Prototype w/ **clickable** links to move between screens
 - Live action storyboard of screens
 - Simulates real **task flow**, but w/ static content
- Fully-implemented prototypes
 - Usually **expensive** to implement actual system
 - But can build key piece of system first to evaluate

Week 13: Think-aloud Usability Studies and Site Design



Why Conduct Usability Studies?

- Evaluate interaction design with real empirical data, gathering ground truth of user performance
- Identify usability issues





Think-aloud Usability Study

- Goal: observe users using app, identify usability issues
- Can use with
 - paper prototype
 - HTML prototype
 - Wizard of Oz study
 - actual app



Steps in a Usability Evaluation Study

- Formulate goals of study
- Design study protocol, tasks, materials, data collection, ...
 - Pilot study design
- Conduct study
- Analyze data to assess task performance and identify usability issues



Informed Consent

- Important for participants to be told up front what they will do and provide affirmative consent
- Helps allay potential participant fears
- Make clear purpose of study
- Make clear that you are evaluating your design, **not** the user



- What will users do?
- Goals for task design:
 - Provide specific goal: something that the user should accomplish
 - Comprehensive enough to exercise key features of your app
 - Short enough to minimize participant time commitments



Communicating Tasks

- Provide a scenario explaining the background of what users will be doing
- Provide a specific goal that the user should accomplish
 - But ***not*** how they should accomplish it
 - Don't give away how you hope users will accomplish goal
- Communicate ***end criterion*** for task - how do they know they're done?
- Provide maximum time limit after which they will be stopped



Training

- Goal: *avoid* unless really necessary
- Training necessary when
 - Participants require specialized knowledge to act as target users
 - Target users will have access to specialized training materials before they begin study



Interactions During the Task

- Goal: listen, not talk
- Prompt participants to think aloud when necessary
 - e.g., What are you trying to do? What did you expect to happen?
- If show signs of stress / fatigue, let them take a break
- Keep participants at ease
 - If participants frustrated, reassure & calm participants
 - If so frustrated they want to quit, let them



Giving Help

- If participants totally off track, small reminder of goal might help
- Should ***not*** give participants information about how to complete the task
- What if user asks for help?
 - Direct them to think through it or work it out for themselves



Collecting Critical Incidents

- *Any action that does not lead to progress in performing the desired task*
- Often related to a gulf of execution or gulf of evaluation
- Generally does not include
 - accessing help
 - random acts of curiosity or exploration



Understanding a Critical Incident

- Important to understand in the moment what users goal is and what actions they are taking
- When a critical incident occurs, jot down
 - The time
 - What user was trying to do
 - What user did



Reporting a Critical Incident

- Problem statement: summary of problem and effect on user (but not a solution!)
- User goals: what was user trying to do?
- Immediate intention: at the moment in time when problem occurred, what was the user trying to do
- Possible causes: speculate on what might have led user to take action they did



Critical Incidents → Usability Issues

- Group together similar incidents to form *usability issue*
 - Match similar critical incidents within and across study sessions
 - Identify underlying cause
- Brainstorm potential fixes



Challenges in Site Design

- Sometimes large space for users to navigate to find information.
- No spatial sense of scale. 50 pages? 500 pages? 50,000 pages?
- No sense of direction. Which way did I just go?
- No sense of location. No spatial anchoring of where I am now and how that relates to where I could go.
- No place to check if something is *not* present or supported.



Site Design

- Some key design dimensions
 - Organization of content into pages / screens
 - Organization of content within pages / screens
 - Ways in which users navigate between pages / screens
- Key design goals
 - Reduce the time / cost for users to reach content
 - Reduce the irrelevant information users must read



Planning

- Help users determine what they **can** do
 - Is this the right site for my goals? Is this the right page where I should spend my time?
- Support users in how they **determine** what to do
 - If this is the right place, how do I reach goal?

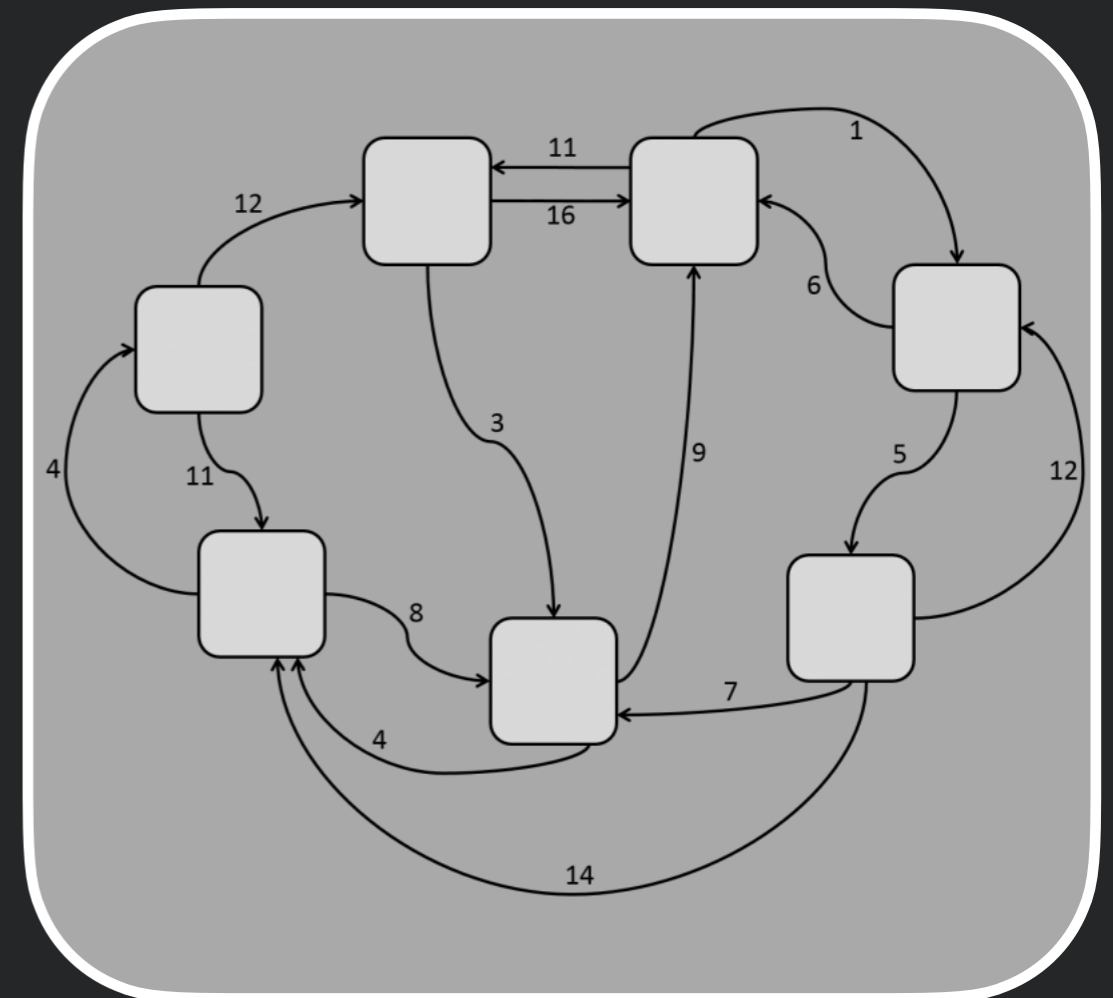


Information Foraging

- Mathematical model describing navigation
- Analogy: animals foraging for food
 - Can forage in different patches (locations)
 - Goal is to maximize chances of finding **prey** while minimizing time spent in hunt
- Information foraging: navigating through an information space (patches) in order to maximize chances of finding prey (information) in minimal time

Information environment

- Information environment represented as topology
 - Information patches connected by traversable links
- Examples
 - Web pages, connected by links
 - Menu options & dialogs connected by commands
 - Locations on map, connected by search, scroll, move interactions with map



Traversing Links

- Patch - a space in the environment where a user is located (e.g., a page, a dialog)
- Links - connection between patch offered by the information environment
- Cues - information features associated with outgoing links from patch
 - E.g., text label on a hyperlink
- User must choose which, of all possible links to traverse, has best chance of reaching prey

- User interprets cues on links by likelihood they will reach prey
- e.g., do I think that the “Advanced options” page is likely to have the option I’m looking for?





Design Implications of Information Foraging Theory

- Organize information into functionally **related** groups
 - If information required is already on same page, no need to go elsewhere
- Design effective **cues**, helping users predict what will be found by traversing links
 - Better cues --> better ability to navigate to correct pages
- Match **expectations** of user's mental model
 - Cues are interpreted relative to mental model
- Provide **search**
 - In large spaces, faster to search than traverse links



Web navigation conventions

Site ID

You are here

Local navigation

Utilities
Sections

NEW & INTERESTING FINDS ON AMAZON EXPLORE

amazon Prime LED & LCD TVs lg tv 4k

BLACK FRIDAY DEALS WEEK

Hello, Thomas Your Account Prime Lists Cart

Departments Browsing History Thomas's Amazon.com Today's Deals

Televitions & Video Deals Best Sellers Televitions Streaming Media Players Blu-ray Players Home Theater Systems AV Accessories

1-24 of 147 results for Electronics : Television & Video : Televitions : LED & LCD TVs : "lg tv 4k" Sort by Relevance

Show results for

- < Any Category
- < Electronics
- < Television & Video
- < Televitions
- LED & LCD TVs

Refine by

Delivery Day

- Get It by Tomorrow

Amazon Prime

- Prime

Television Feature

- Smart TV (132)
- 3D (53)

Television Resolution

- 4K Ultra HD (70)
- 1080p (16)
- 1080i
- 760p
- 760i
- 720p (1)
- 720i
- 480p

SPONSORED BY LG HOME ELECTRONICS

Save on LG Super UHD TVs

> Shop now

LG Electronics 65UH77... LG Electronics 55UH77...

Ad feedback

Showing most relevant results. See all results for lg tv 4k.

Television Feature: Smart TV | 3D

Sponsored

LG Electronics 55UH6550 55-Inch 4K Ultra HD Smart LED TV (2016 Model)

by LG Electronics

\$747.00 \$897.00 Prime

★★★★☆ 25

- Display Size: 55 inches
- Resolution: 4K Ultra HD
- Connectivity Technology: Built-in Wi-Fi
- Display Technology: LED
- Display Resolution Maximum: 4K Ultra HD

Sponsored

LG Electronics 60UH8500 60-Inch 4K Ultra HD Smart LED TV (2016 Model)

by LG Electronics

\$1,297.00 \$1,697.00 Prime

★★★★☆ 87

Electronics Gift Guide

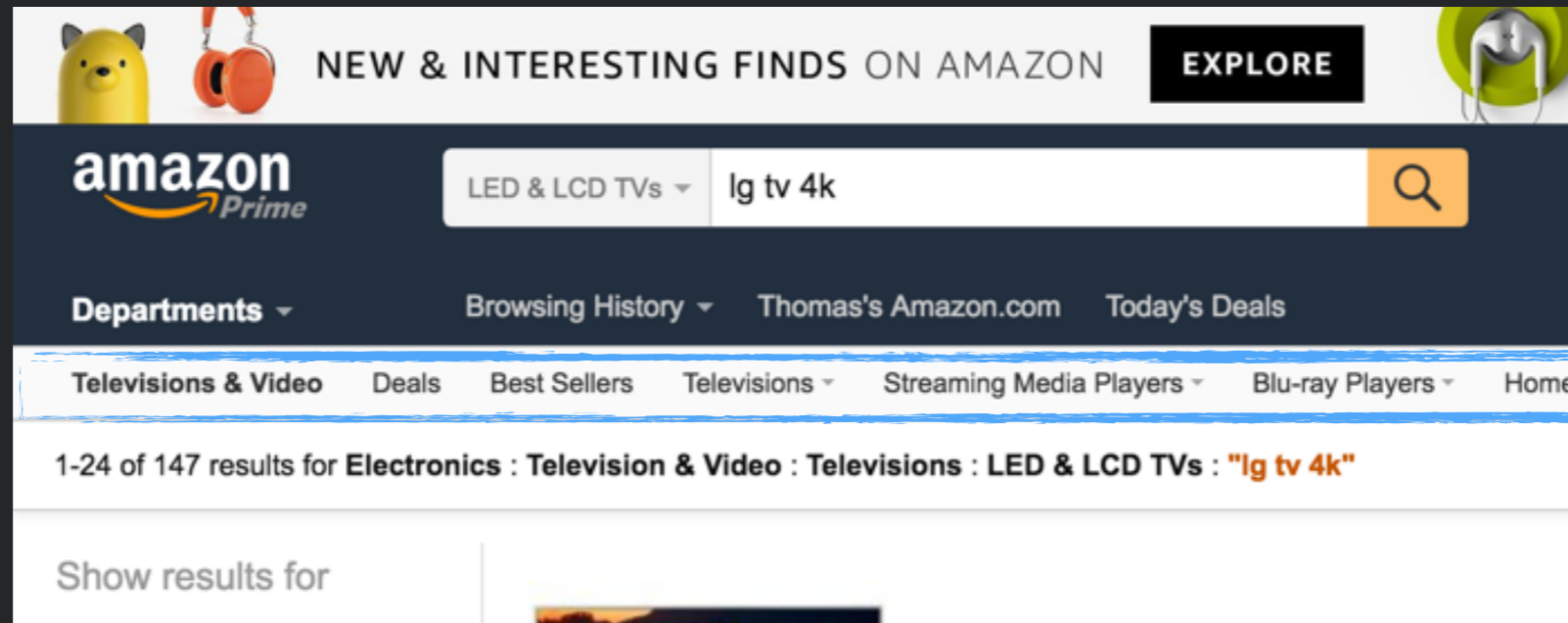
Footer
navigation



Persistent Navigation

- Forms a common idiom users already understand
- Gives instant confirmation that still on the same site
- Supports consistency and standards
 - If *all* of your pages function same way, users know how to do actions & what to expect
 - Ok for specialized page like forms that are clearly different to not follow conventions.

Tabs



- Example of a metaphor: tab dividers in a three ring binder or folders in a file drawer
- Partition into sections
- Advantages
 - Easily understood and self-evident
 - (Usually) hard to miss



Breadcrumbs

- Offer trail of where the user has been and how they got there
- Shows hierarchy of information space
- Shows current location

NEW & INTERESTING FINDS ON AMAZON EXPLORE

amazon Prime LED & LCD TVs lg tv 4k

BLACK FRIDAY DEALS WEEK

Hello, Thomas Your Account Prime Lists Cart

Departments Browsing History Thomas's Amazon.com Today's Deals

Televisions & Video Deals Best Sellers Televisions Streaming Media Players Blu-ray Players Home Theater Systems A/V Accessories

1-24 of 147 results for Electronics : Television & Video : Televisions : LED & LCD TVs : "lg tv 4k" Sort by Relevance

Show results for

- < Any Category
- < Electronics
- < Television & Video
- < Televisions
- LED & LCD TVs**

Refine by

Delivery Day

- Get It by Tomorrow

Amazon Prime

- Prime

Television Feature

- Smart TV (132)
- 3D (53)

Television Resolution

- 4K Ultra HD (70)
- 1080p (16)
- 1080i
- 760p
- 760i
- 720p (1)
- 720i
- 480p
- 480i

SPONSORED BY LG HOME ELECTRONICS

Save on LG Super UHD TVs

> Shop now

LG Electronics 65UH77... LG Electronics 55UH77...

Showing most relevant results. See all results for lg tv 4k.

Television Feature: Smart TV | 3D

Sponsored

LG Electronics 55UH6550 55-Inch 4K Ultra HD Smart LED TV (2016 Model)

by LG Electronics

\$747.00 \$897.00 Prime

★★★★☆ 25

- Display Size: 55 inches
- Resolution: 4K Ultra HD
- Connectivity Technology: Built-in Wi-Fi
- Display Technology: LED
- Display Resolution Maximum: 4K Ultra HD

Sponsored

LG Electronics 60UH8500 60-Inch 4K Ultra HD Smart LED TV (2016 Model)

by LG Electronics

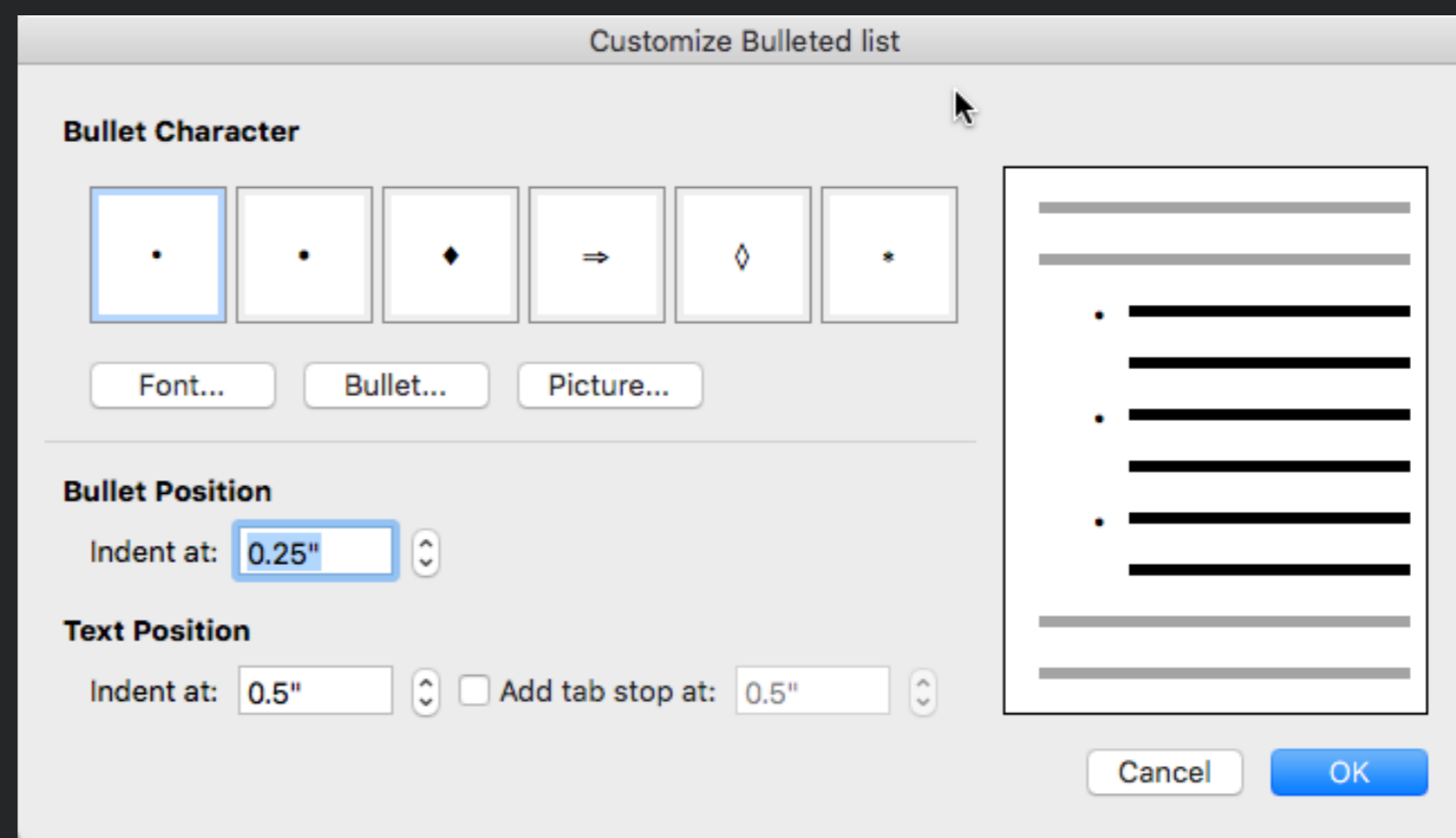
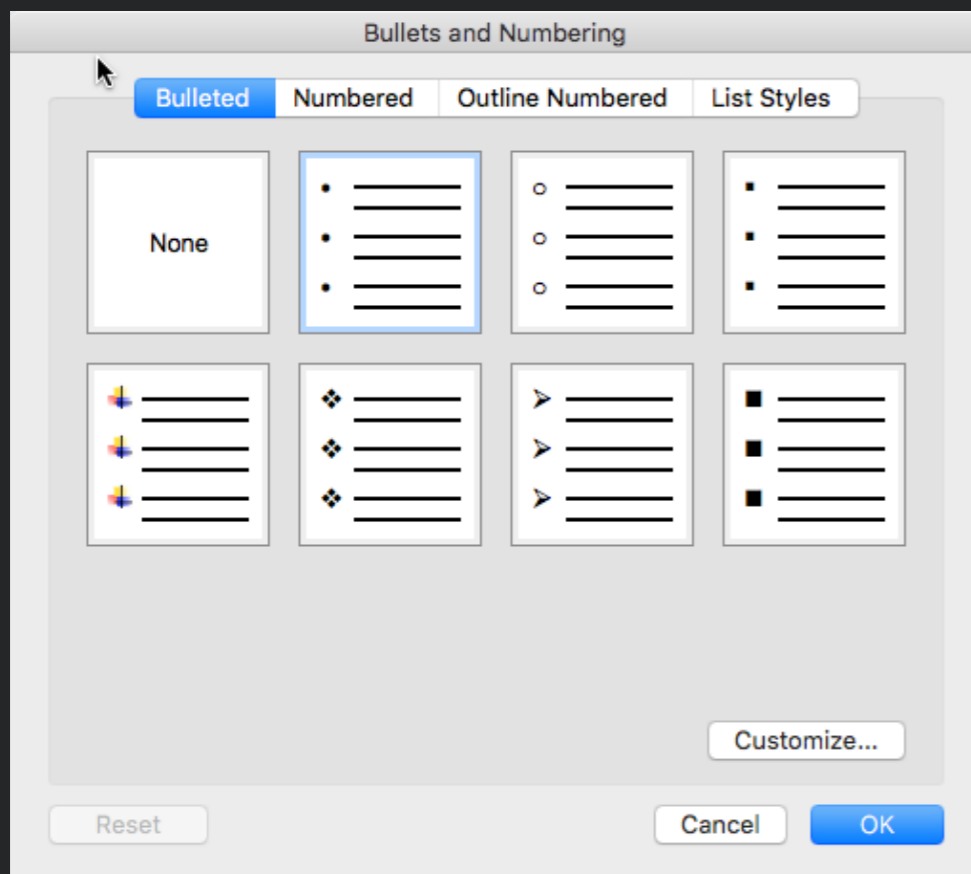
\$1,297.00 \$1,697.00 Prime

★★★★☆ 87

Electronics Gift Guide

Progressive Disclosure

- a.k.a. details on demand
- Separate information & commands into layers
- Present most frequently used information & commands first





Effective Site Design

- Answers to the following should be obvious for a good site design
 - What site is this? (Site ID)
 - What page am I on? (Page name)
 - What are the major sections of this site? (Sections)
 - What are my options at this level? (Local navigation)
 - Where am I in the site? (“You are here” indicators)
 - How can I search?



Metaphors - Advantages

- Leverages understanding of familiar objects & their functions
 - File cabinets, desks, telephones
- Provides *intuitive* understanding of possible affordances & eases mapping tasks to actions
 - Open a folder, throw file in trash, momentum scrolling

Metaphors - Disadvantages

- Tyranny of metaphor: ties interactions closely to workings of physical world
- Adds useless overhead in extra steps, wastes visual bandwidth
- Taken literally, becomes non-sensical
 - e.g., nesting folders 10 levels deep





Alternative - Idioms

- A consistent mental model of how something works
 - e.g., Files: open / close / save / save as
- Offers intuitive understanding of affordances & interactions
- Provides consistent vocabulary for describing interactions
- Only have to learn it **once**
- Might have originated in real world, but thought of in terms of mental model for UI interactions

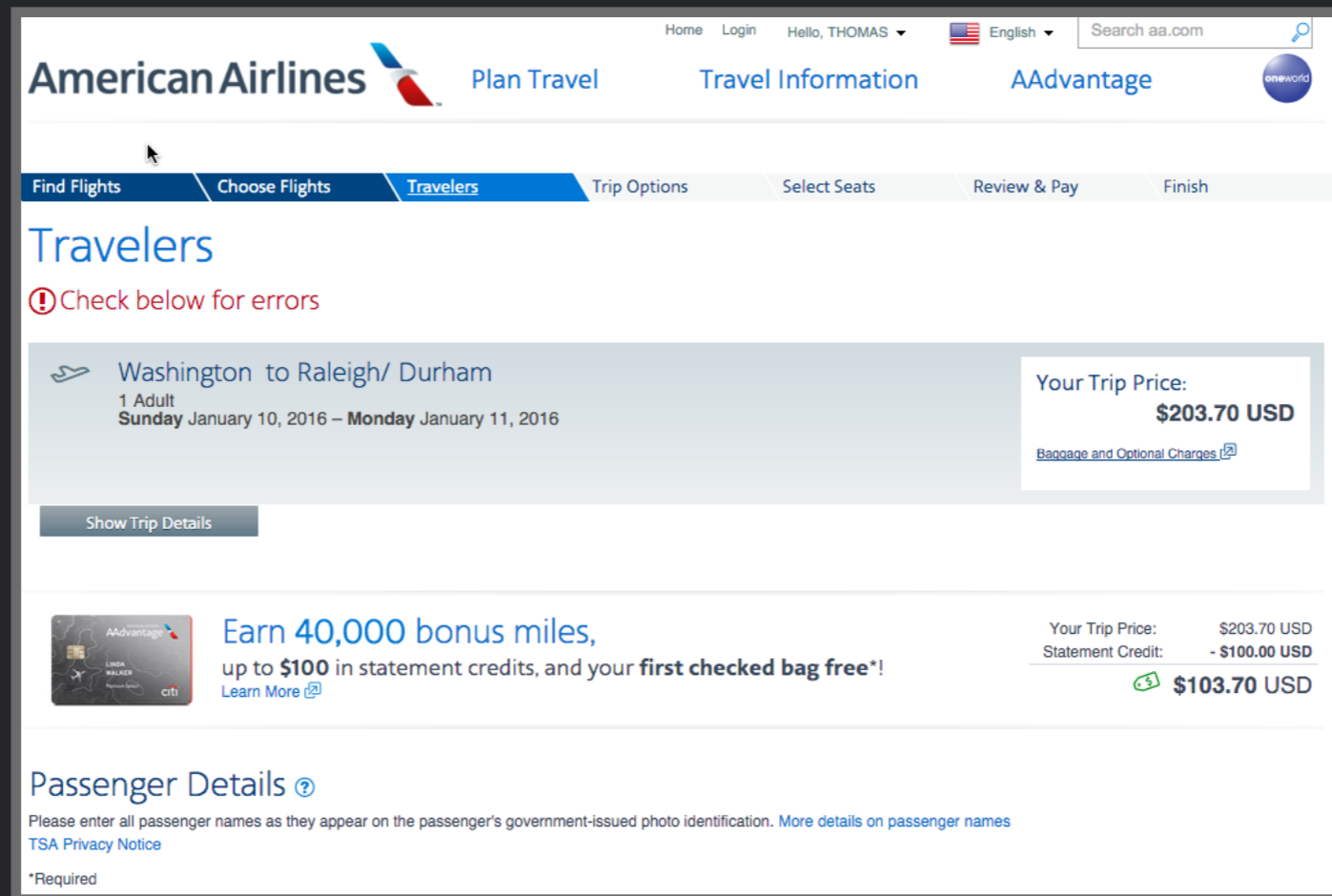


Task Structure

- In some cases, users must take actions in specific sequence
- Must input some information before being able to access subsequent information
 - e.g., must select a shipping method before seeing a final price
- To the extent possible, want to leave users in control of task (user control and freedom)
- But also do not want to distract users by making unrelated decisions in random order (flexibility and efficiency of use)
- And do not want to overwhelm users with too many options at a time (minimalist design)
- Good designs need to balance tradeoffs

Separate long tasks into sequences

- Reduce short term memory demands by having user only work on one aspect of larger task at a time
- Don't interrupt users in the middle with unrelated tasks
- Provide closure of each subtask at the end



Home Login Hello, THOMAS English Search aa.com

American Airlines Plan Travel Travel Information AAdvantage oneworld

Find Flights Choose Flights **Travelers** Trip Options Select Seats Review & Pay Finish


Travelers

⚠ Check below for errors

✈ Washington to Raleigh/ Durham
1 Adult
Sunday January 10, 2016 – Monday January 11, 2016

Your Trip Price: **\$203.70 USD**
[Baggage and Optional Charges](#)

Show Trip Details

 Earn 40,000 bonus miles,
up to \$100 in statement credits, and your **first checked bag free***
[Learn More](#)

Your Trip Price: \$203.70 USD
Statement Credit: - \$100.00 USD
\$103.70 USD

Passenger Details

Please enter all passenger names as they appear on the passenger's government-issued photo identification. [More details on passenger names](#)
[TSA Privacy Notice](#)

*Required



Interaction Flow Guidelines

- Don't use dialogs to report normal behavior
- Separate commands from configuration
- Don't ask questions, give users choices
 - Give users default input, show possible options
- Make dangerous choices hard to reach
- Design for the probable, provide for the possible

Week 14: Interaction Techniques & Visual Design



Signifiers

Is this a button?

Or a link?

- Goals
 - Show which UI elements can be manipulated
 - Show how they can be manipulated
 - Help users get started
 - Guide data entry
 - Suggest default choices
 - Support error recovery

Hinting

- Indicate which UI elements can be interacted with
- Possible visual indicators
 - **Static hinting** - distinctive look & feel
 - **Dynamic hinting** - rollover highlights
 - **Response hinting** - change visual design with click
 - **Cursor hinting** - change cursor display

Course Project

Course Project

Project Overview

The major assignments in the course will be in the form of a project, and will be distributed over the course of the semester as "Project Checkpoints". You will first design and implement a simple UI in the form of a web app. Throughout the semester, you will perform peer evaluations, identifying usability issues with the UI of apps built by other students in the course. Based on the reported usability issues you receive, you will then iteratively redesign and improve the usability of your web app to address these issues. Full details for each Project Checkpoint can be found in the Project Checkpoint descriptions below; the due dates are summarized in the course schedule.

What to Build?

You are given the freedom to build any type of web application that you would like for the semester project. However, there are some general guidelines that are important to follow:

- *The project should be something the group can implement in two weeks.* Because much of this project will be focused on evaluating and refining the UI, the premise of the app should be simple. Some successful projects in the past have been as short as 500 lines of code.
- **It must be implemented as a web application and be usable by visiting a URL.** Projects can be implemented entirely client-side, or with some back-end technologies, but the back-end should be kept to a minimum.
- *We will primarily be evaluating your project based on the UI you create, not the elegance or sophistication of your implementation.* Thus, we expect that the best projects will be those that involve a significant amount of user facing interactions.

Table of contents

- Course Project
- Project Overview
- What to Build?
- Project Collaboration
- Project Checkpoint Schedule and Assignment Instructions



Clarity of Wording

- Choose words carefully
- Speak the user's language
- Avoid vague, ambiguous terms
- Be as specific as possible
- Clearly represent domain concepts



Likely & Useful Defaults

- Default text, if relevant (e.g., date)
- Default cursor position
- Avoid requirements to retype & re-enter data



Modes

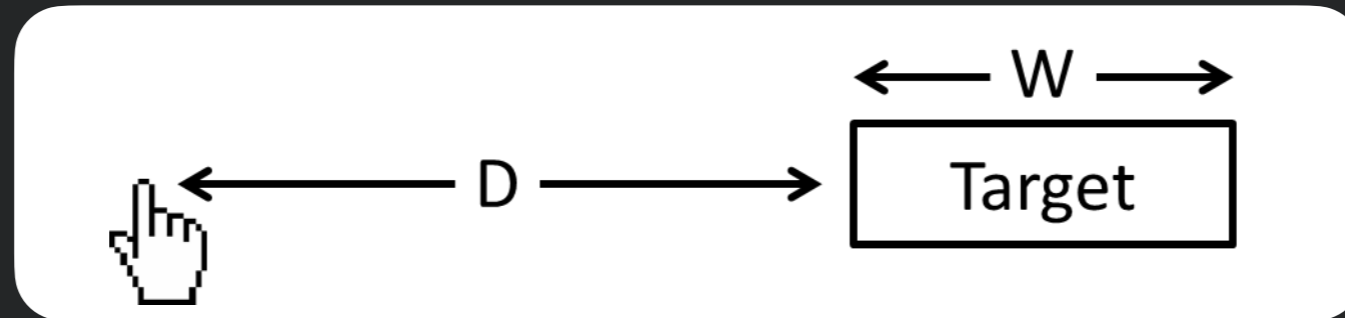
- Vary the effect of a command based on state of system
- Examples
 - caps lock
 - insert / overwrite mode
 - vi / emacs command modes
 - keyboard entry used for controlling game and chatting



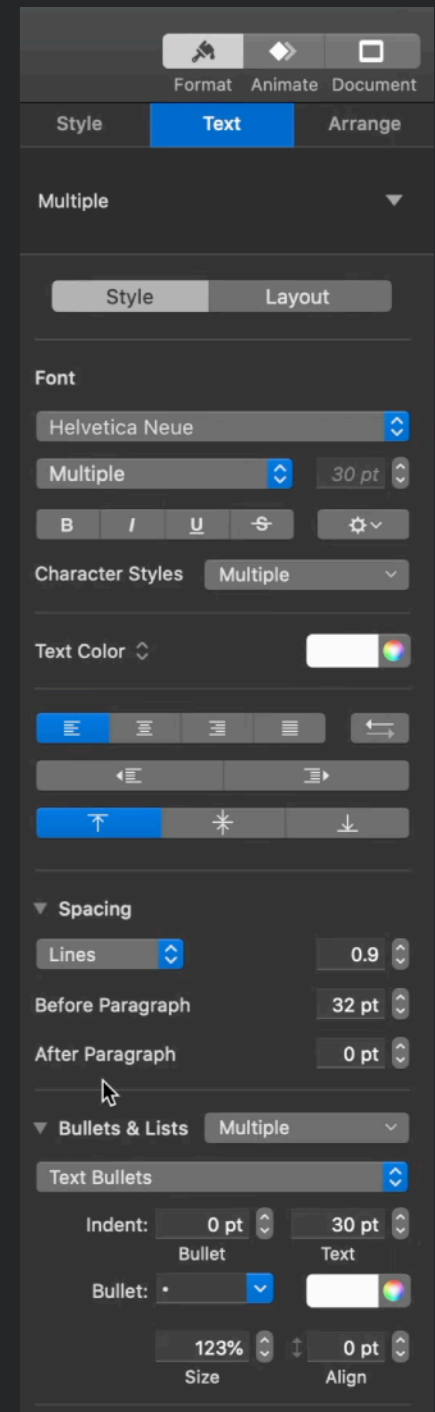
Challenges with Modes

- Modes create inconsistent mapping
 - E.g., control S sometimes saves, sometimes sends email
 - Especially dangerous for frequent interactions that become highly automatic System 1 actions
- Avoid when possible
- Clearly distinguish if necessary
 - Make clear to user which mode they are in and how to change

Fitt's Law



- Time required to move to a target decreases with target size & increases with distance to the target
- Movements typical consist of
 - one large quick movement to target (ballistic movement)
 - fine-adjustment movement (homing movements)
- Homing movements generally responsible for most of movement time & errors
- Applies to rapid pointing movements, not slow continuous movements





Design Implications of Fitt's Law

- **Constraining** movement to one dimension dramatically increases speed of actions
 - e.g., scroll bars are 1D

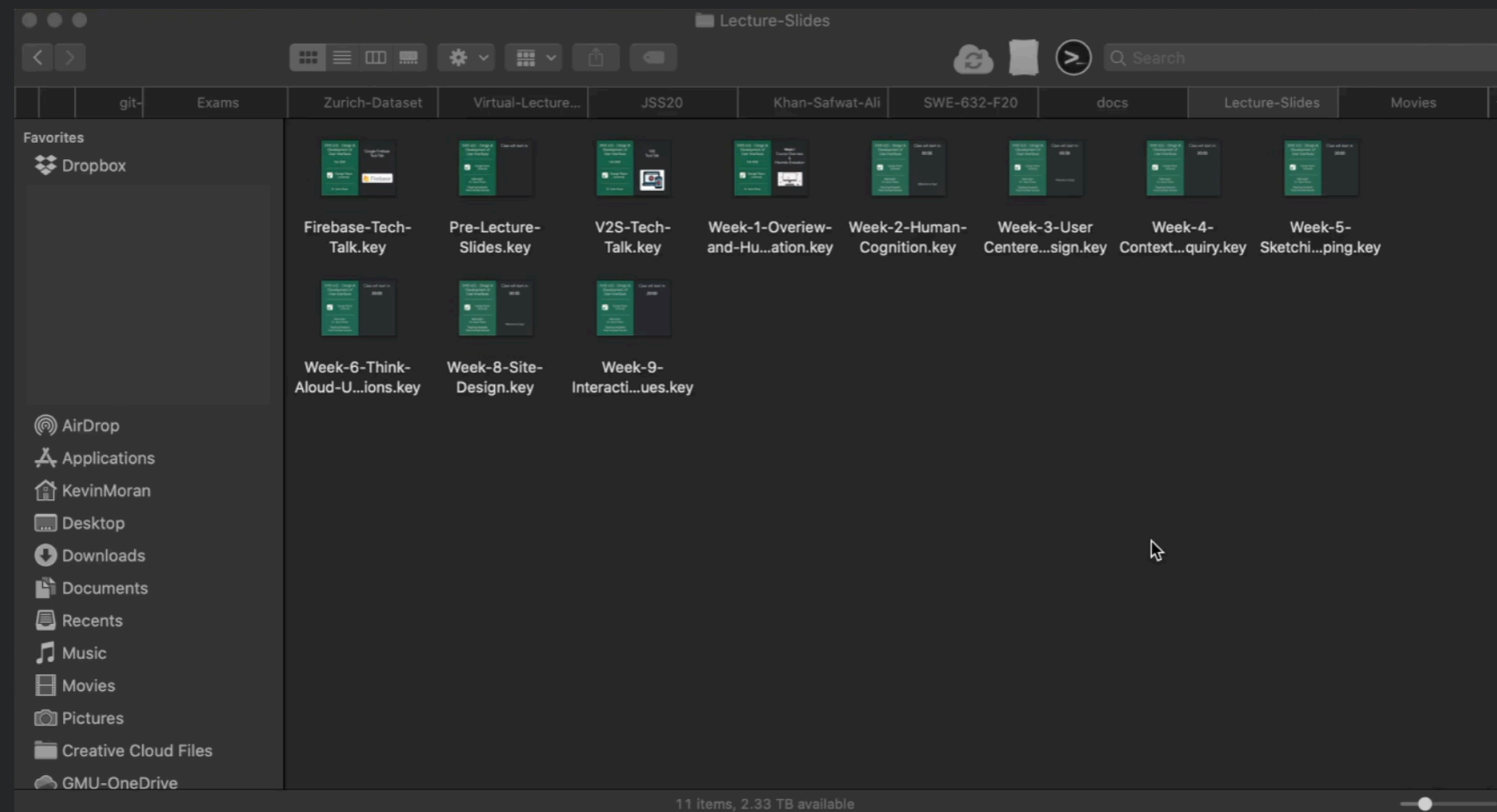
The screenshot shows a Beamer presentation slide titled "Design Implications of Fitt's Law". The slide is part of a larger presentation, as indicated by the navigation pane on the left. The slide content includes a list of bullet points:

- **Constraining** movement to one dimension dramatically increases speed of actions
- e.g., scroll bars are 1D

The slide also features a navigation pane on the left with 10 numbered slides, a status bar at the bottom showing "24", and a footer with the number "177".

Design implications of Fitt's law

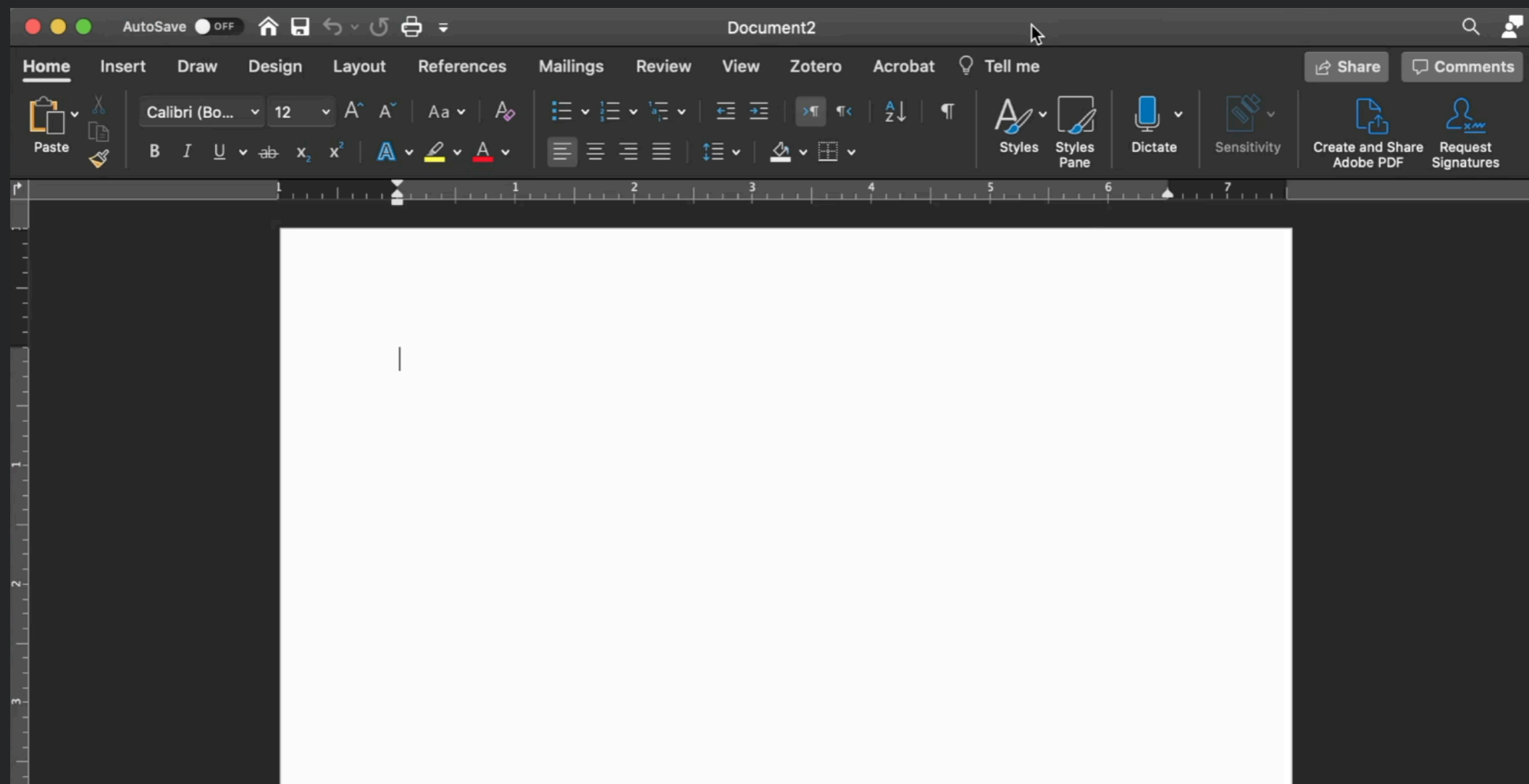
- Making controls ***larger*** reduces time to invoke actions
- Locating controls closer to user ***cursor*** reduces time
 - e.g., context menus





Design Implications of Fitt's Law

- Positioning button or control along edge of screen acts as barrier to movement, substantially reducing homing time & errors





Mobile Apps - Where's the Cursor?

- No cursor on many mobile devices
- Cannot use dynamic hinting to determine which elements can be interacted with
 - May require more use of static hinting
- Fitt's law still applies
 - Fingers are less sensitive, hard to select small buttons, occlude elements

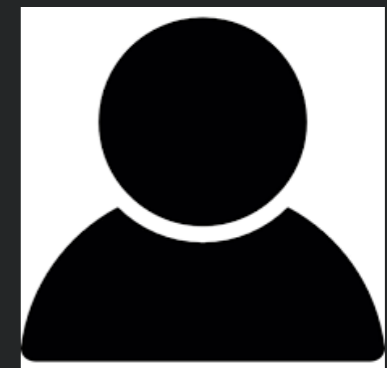


Supporting Users with Disabilities

- **Perception** - visual & auditory impairments
 - Blindness or visual impairments
 - Color blindness
 - Deafness & hearing limitations
- **Motion** - muscle control impairments
 - Difficulties with fine muscle control
 - Weakness & fatigue
- **Cognition** - difficulties with mental processes
 - Difficulties remembering
 - Difficulties with conceptualizing, planning, sequencing actions

Universal Design

- How can users with physical disabilities be supported in user interactions?
- Good: *assistive design* - offering equivalent actions for disabled users that cannot take normal actions
- Better: *universal design* - designing interactions so broadest set of users across age, ability, status in life can use normal actions



Example - Curb cut

- Initially designed for **accessibility** - support for disabled & wheel chairs
- But potentially benefits **all users** of public spaces - people w/ suitcases, hand carts, roller blades, bikes, ...



7 Principles of Universal Design

- **Equitable use:** The design is useful and marketable to people with diverse abilities
- **Flexibility in use:** The design accommodates a wide range of individual preferences and abilities
- **Simple and intuitive:** Use of the design is easy to understand, regardless of the user's experience, knowledge, language skills, or current concentration level
- **Perceptible information:** The design communicates necessary information effectively to the user, regardless of ambient conditions or the user's sensory abilities
- **Tolerance for error:** The design minimizes hazards and the adverse consequences of accidental or unintended actions
- **Low physical effort:** The design can be used efficiently and comfortably and with a minimum of fatigue
- **Size and space for approach and use:** Appropriate size and space is provided for approach, reach, manipulation, and use regardless of user's body size, posture, or mobility

Scale, Contrast, & Proportion



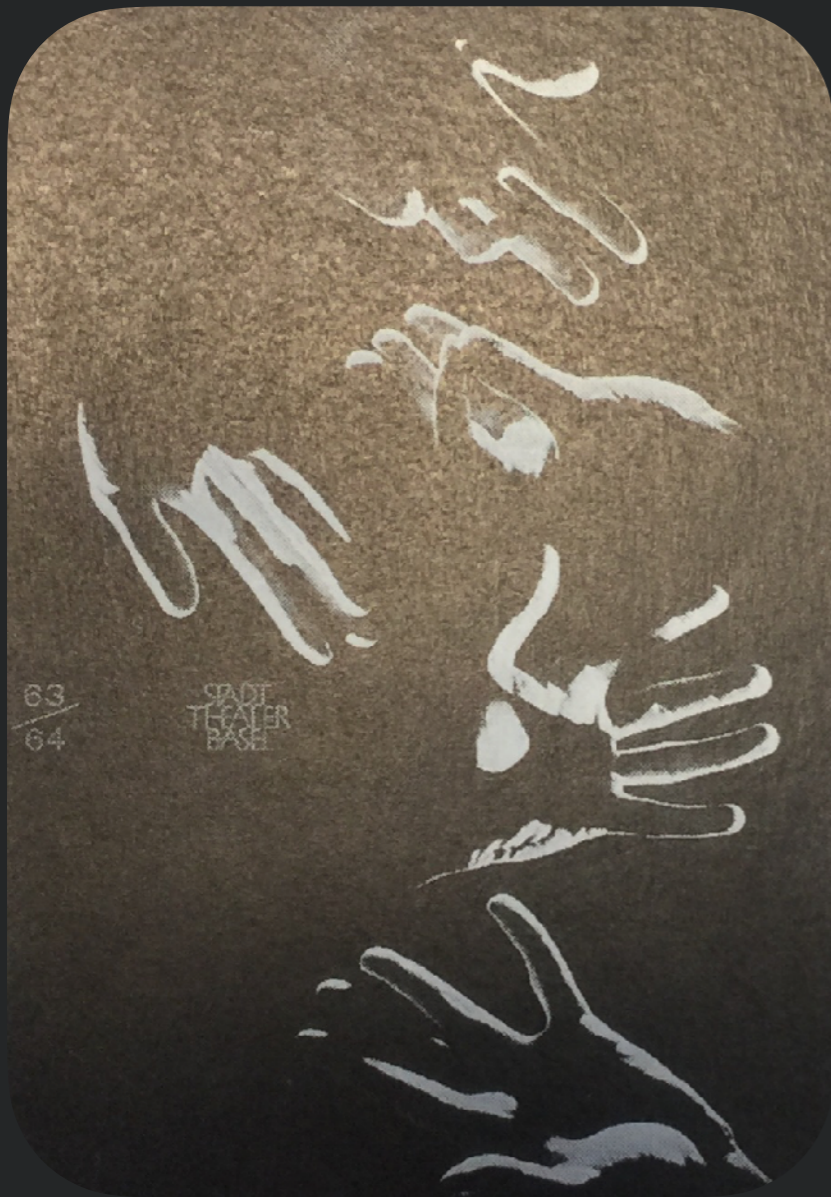


Scale, Contrast, & Proportion

Information consists of differences that make a difference.
(Edward Tufte, Envisioning Information)

Individual visual variables of design that encode information

Terminology



- Scale - *relative* size or magnitude of element in comparison to related elements
- Contrast - visually noticeable *distinctions* along a common visual dimension
- Proportion - ratio and *balance* between elements
- Emphasis - contrasts can emphasize important elements or areas & add visual *interest* by creating tension & drama

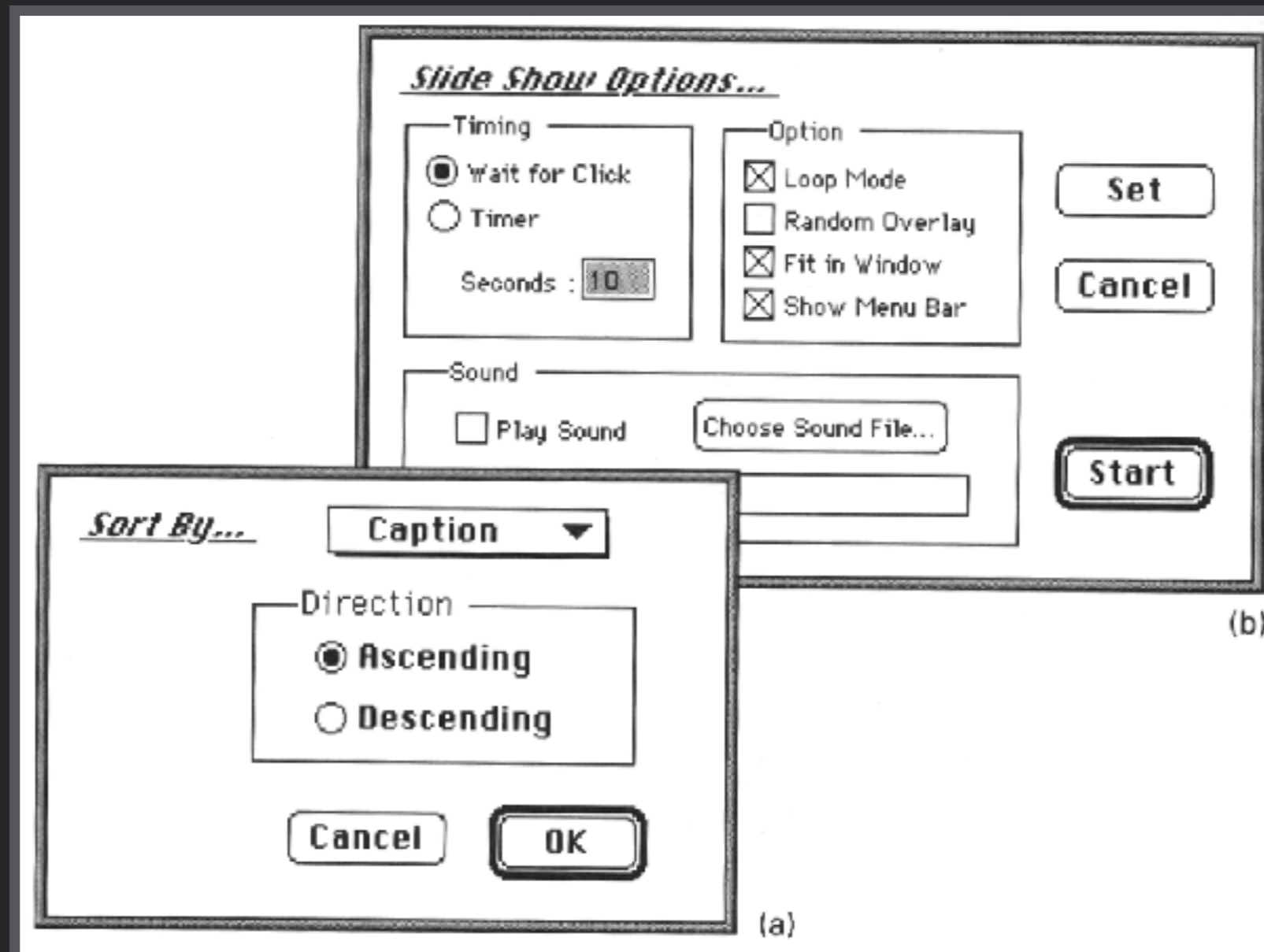


Principles

- **Clarity** - contrasts should be clear and easily differentiated, not slight and subtle
- **Harmony** - proportions and ratios should be harmonious
- **Activity** - use contrasts to maintain orientation & context within design
- **Restraint** - contrasts should be conscious, strong, few in number, and never overwhelming

Error - Excessive Typographic Contrasts

5 different types sizes in 3 different fonts (!!)



Layers



- Contrasting color, value, texture can segregate information into separate layers
- Supports overlapping information in displays, allowing selective processing of specific sets of elements
- Allows different layers to be read and interpreted separately





Creating Layers

1. Group items into categories based on intended use
2. Determine rank & importance of groups
3. Use perceptual variables (size, value, hue, etc.) to establish layering effect
4. Maximize differences between groups while minimizing differences within groups
5. Use squint test to ensure elements in group retain together but visually separated

Marshalling Signals



MARSHALLING SIGNALS



Organization & Structure



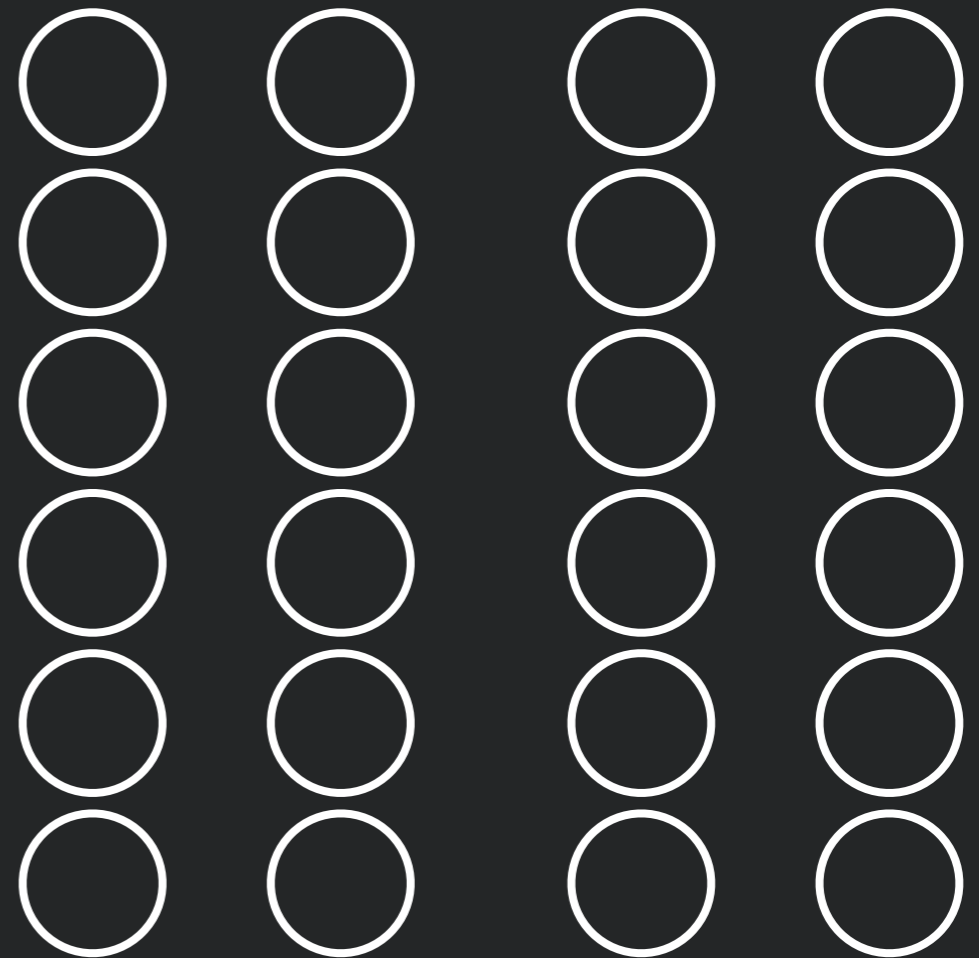
Organization & Structure

- Organization needs to be *designed*
- Benefits
 - Unity - ties together related elements so that they work **together**
 - Integrity & readability - offers structure that helps user to easily scan & make comparisons
 - Control - determines where user will focus **attention** in the design
- Gestalt -> psychology of perception

Gestalt Principle - Proximity

- Elements associated most strongly w/ nearby elements

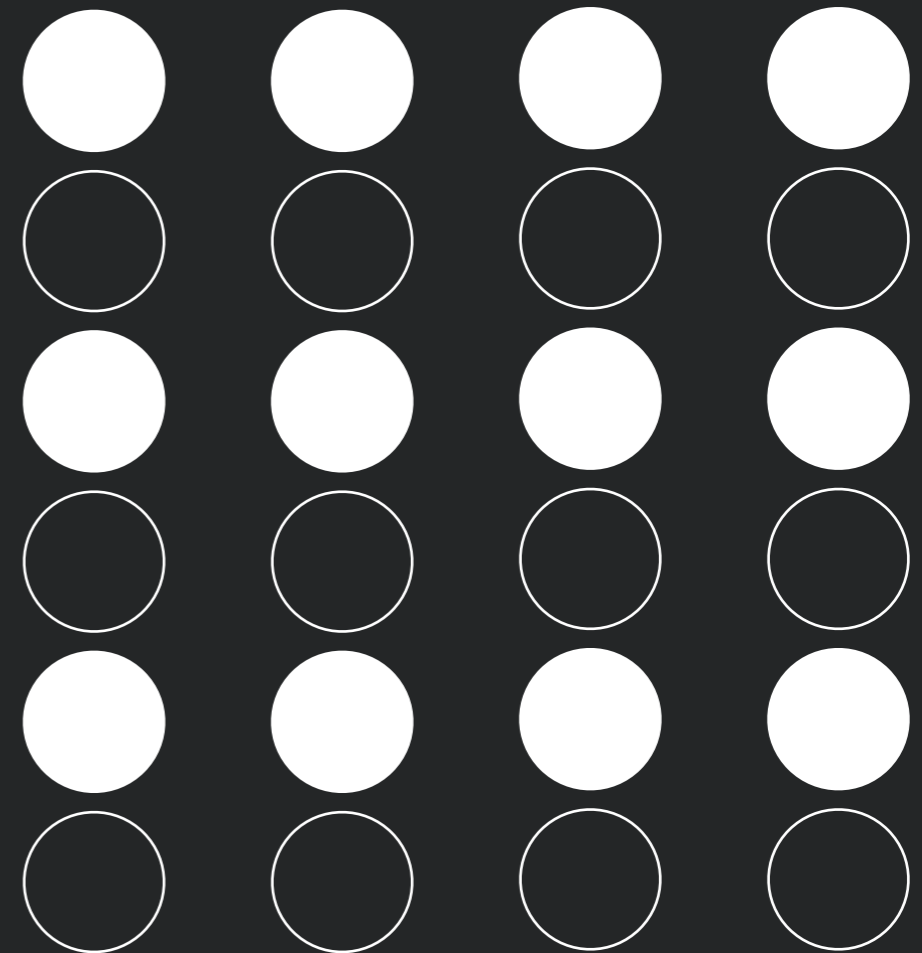
parsed as 4 columns based on close vertical spacing
then parsed as two sets of two columns based on spacing



Gestalt Principle - Similarity

- Elements associated more strongly when share common visual attributes than when they differ

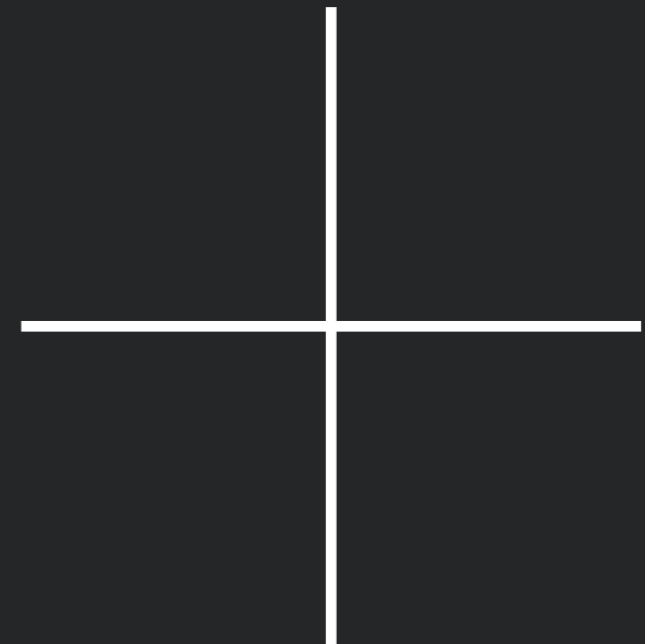
parsed as rows based on fill similarity,
despite closer column spacing



Gestalt Principle - Continuity

- Preference for *simplest* physical explanation of complex figure

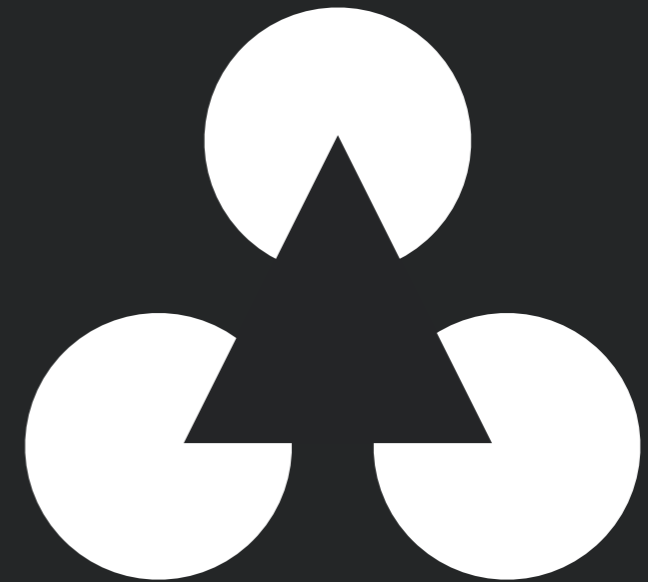
parsed as two lines, rather than 4
separate lines or 4 opposing angles



Gestalt Principle - Closure

- Preference to interpret figures as complete, even when missing information

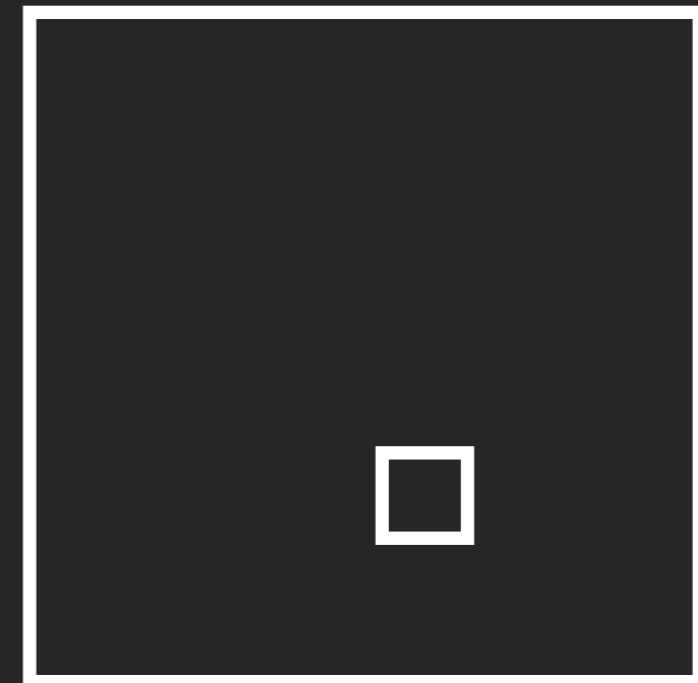
Parsed as triangle superimposed on 3 complete circles, even though none of these is actually present



Gestalt Principle - Area

- Preference to interpret smaller overlapping elements as figure, larger as ground

Small rectangle parsed as small rectangle on top of larger, rather than hole



Gestalt Principle - Symmetry

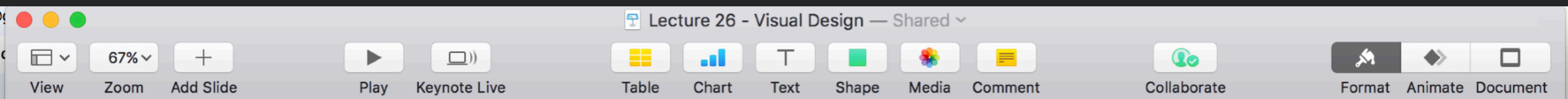
- Preference to interpret ambiguous form as multiple symmetric elements

Parsed as two overlapping objects rather than 3 separate shapes

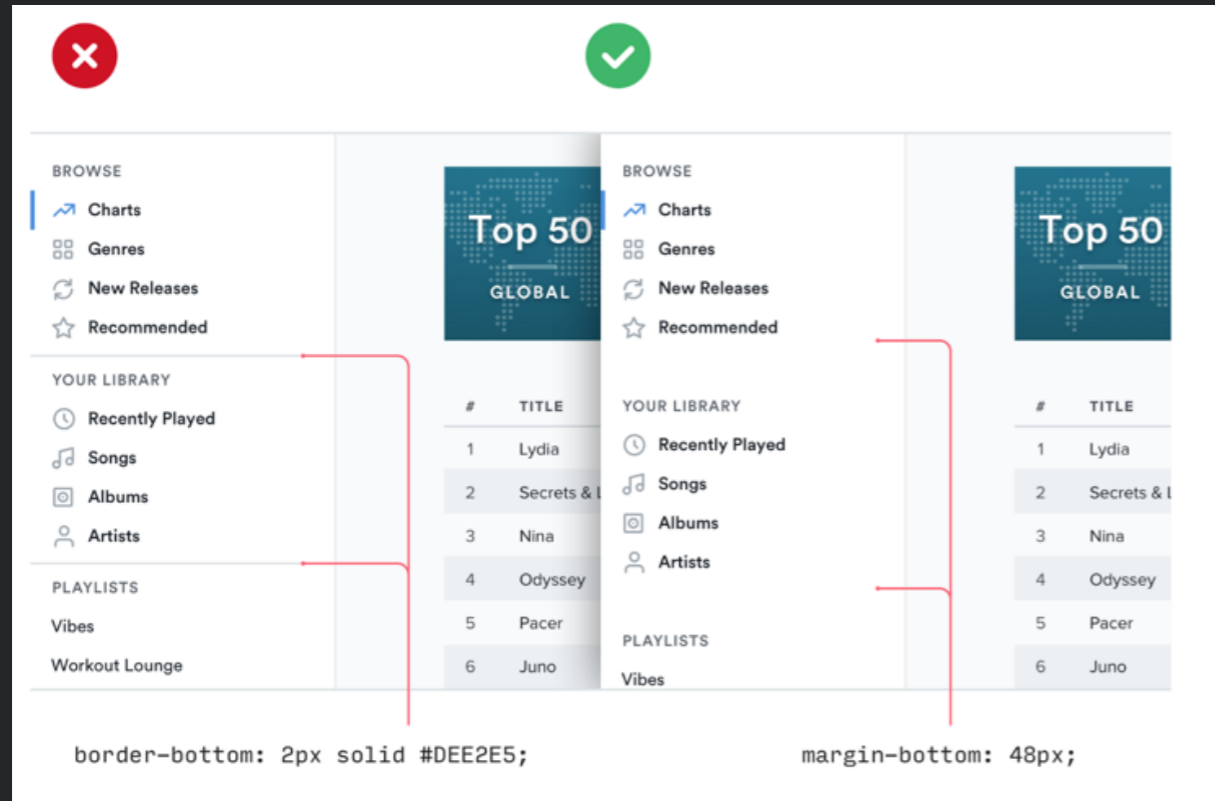


Grouping

- Binding UI elements tightly together while distinguishing them from surrounding controls
- “Showing” not “telling”
- Can be achieved through
 - Bounding boxes (not recommended)
 - Negative space & contrasts
 - Arrangement & alignment

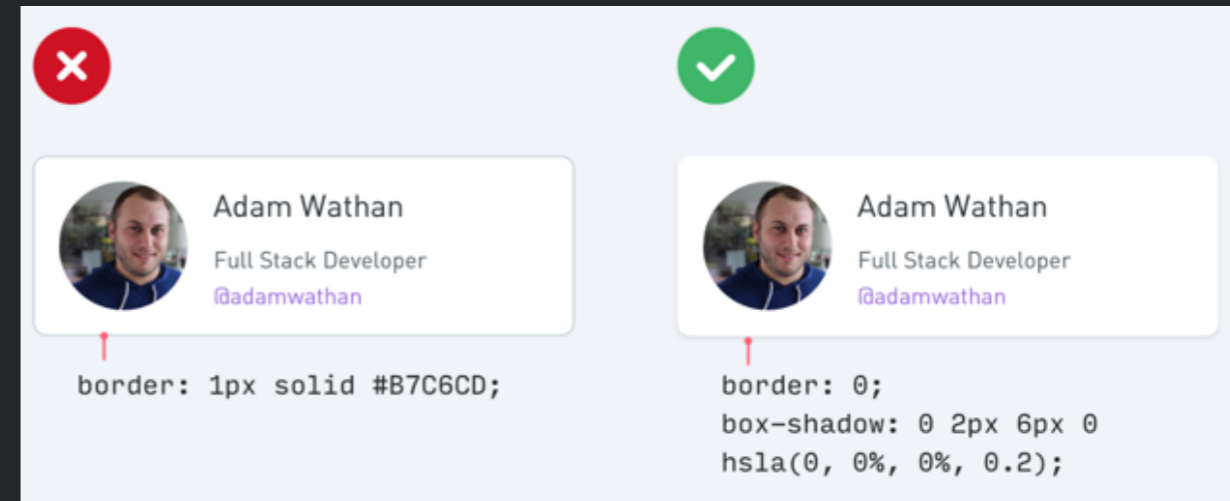


Use Fewer Borders

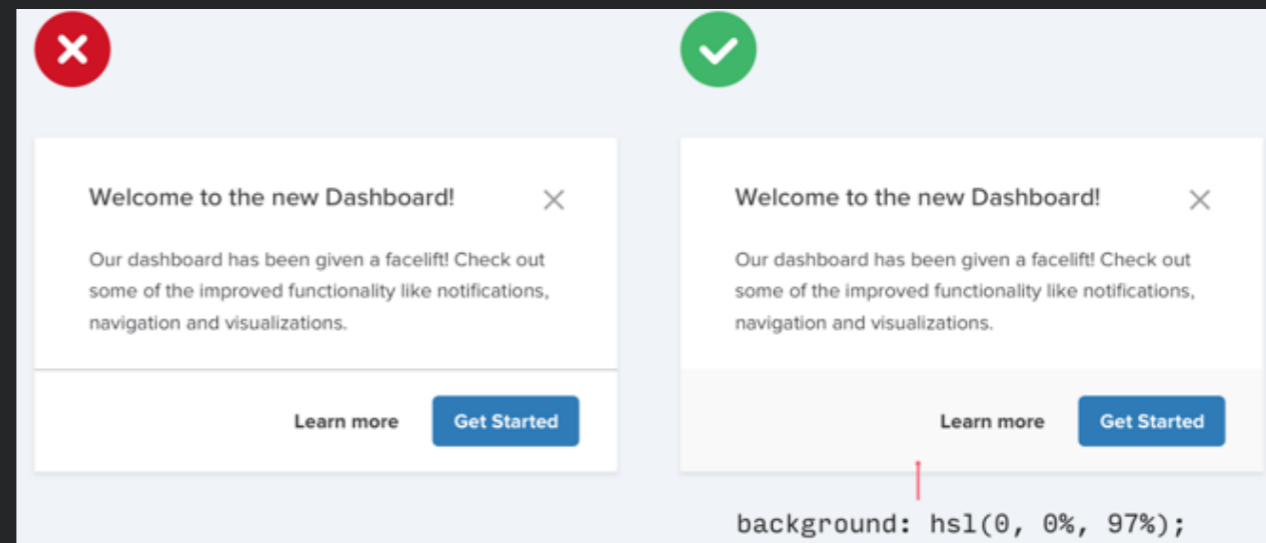


negative space

<https://medium.com/refactoring-ui/7-practical-tips-for-cheating-at-design-40c736799886>



box shadows



different backgrounds

Hierarchy

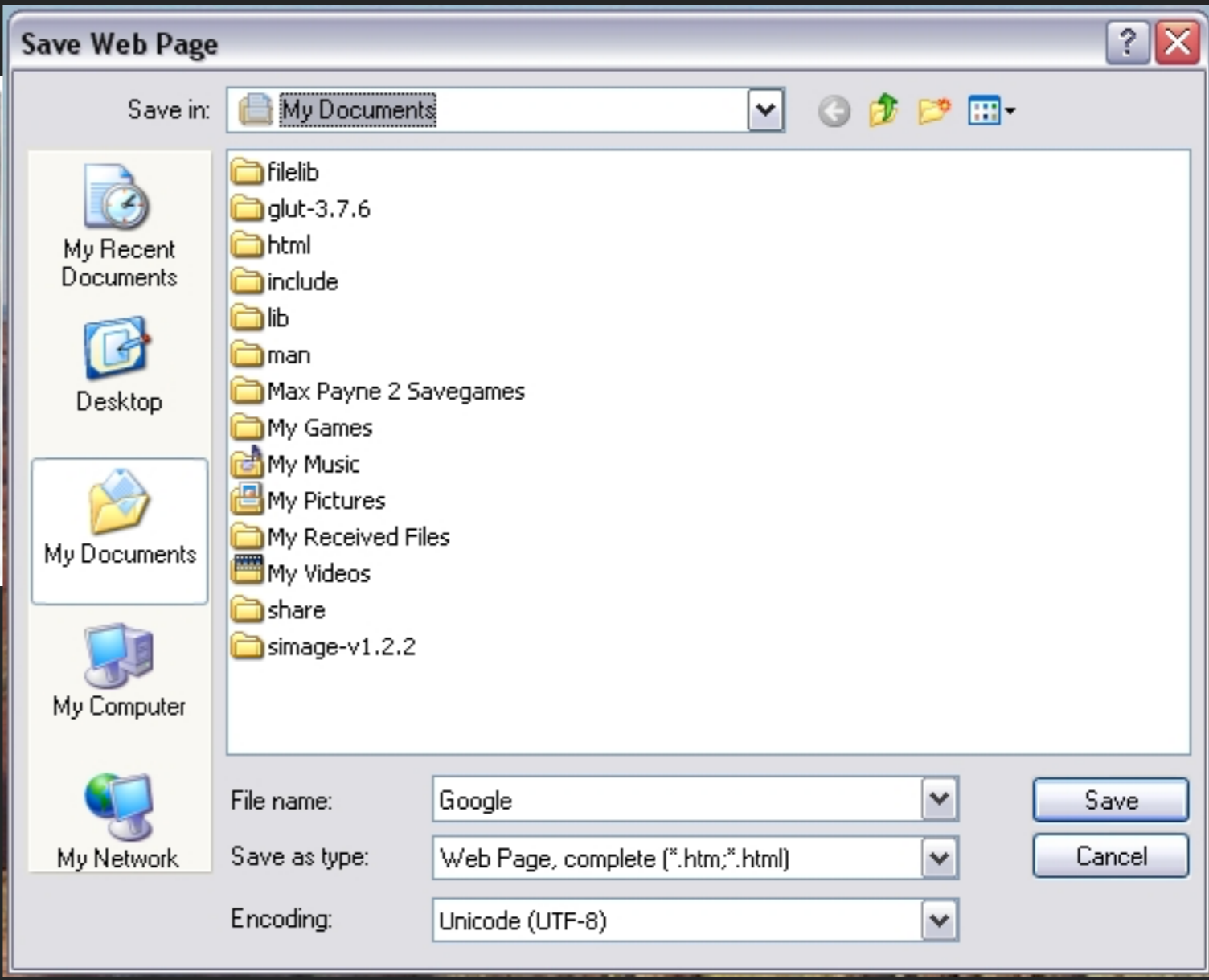
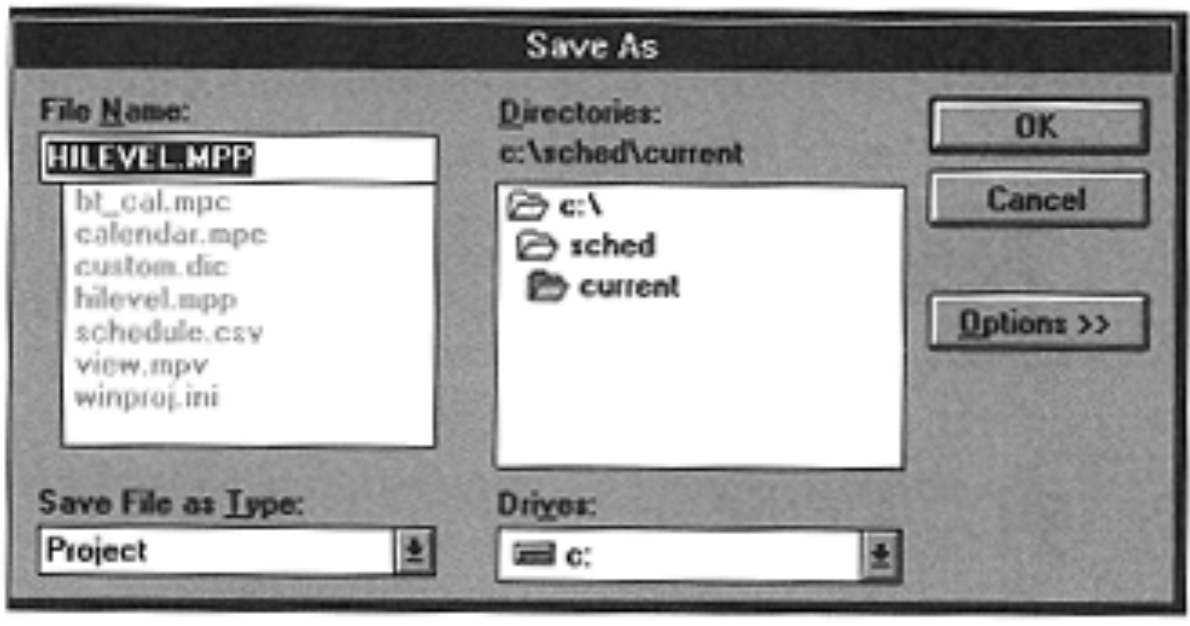
Order groups based on perceptual prominence corresponding to intended reading sequence

Can help solve “skimming” problems

Structure can help people focus attention on key parts

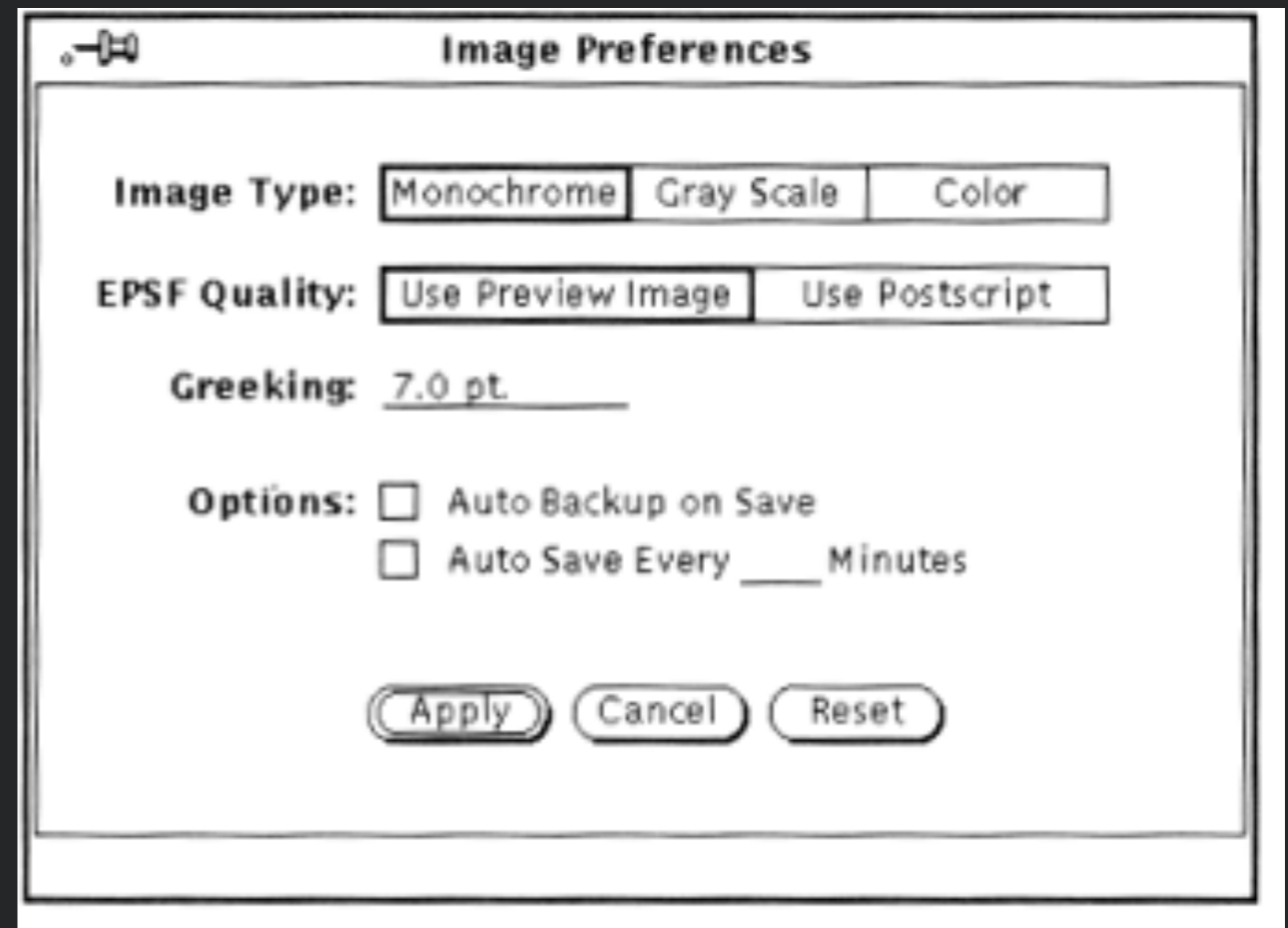
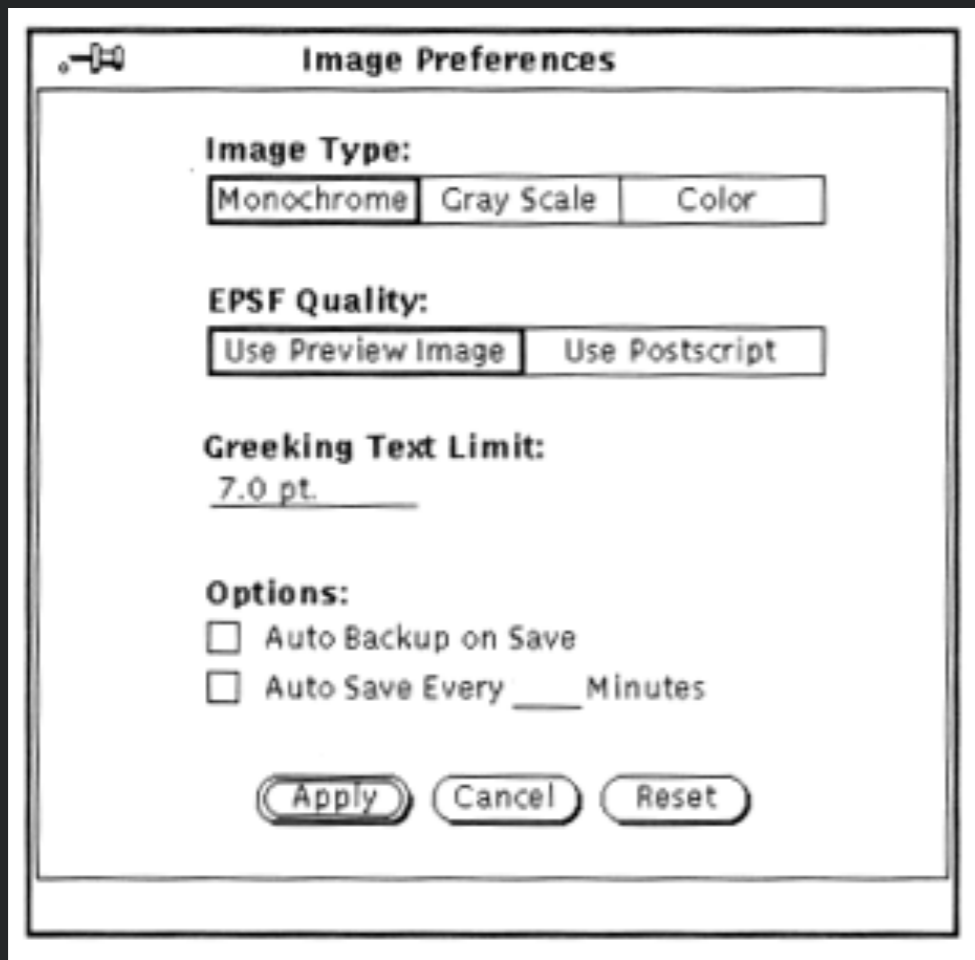
Key points might **get lost though.**

But bolding helps! Plus this obnoxious red arrow and text in a totally different font!



Use Negative Space

- Directs ***attention*** to critical regions of display
 1. Review design, prioritizing groups
 2. Add extra ***space*** to ensure spatial separation & emphasis, particularly for important elements



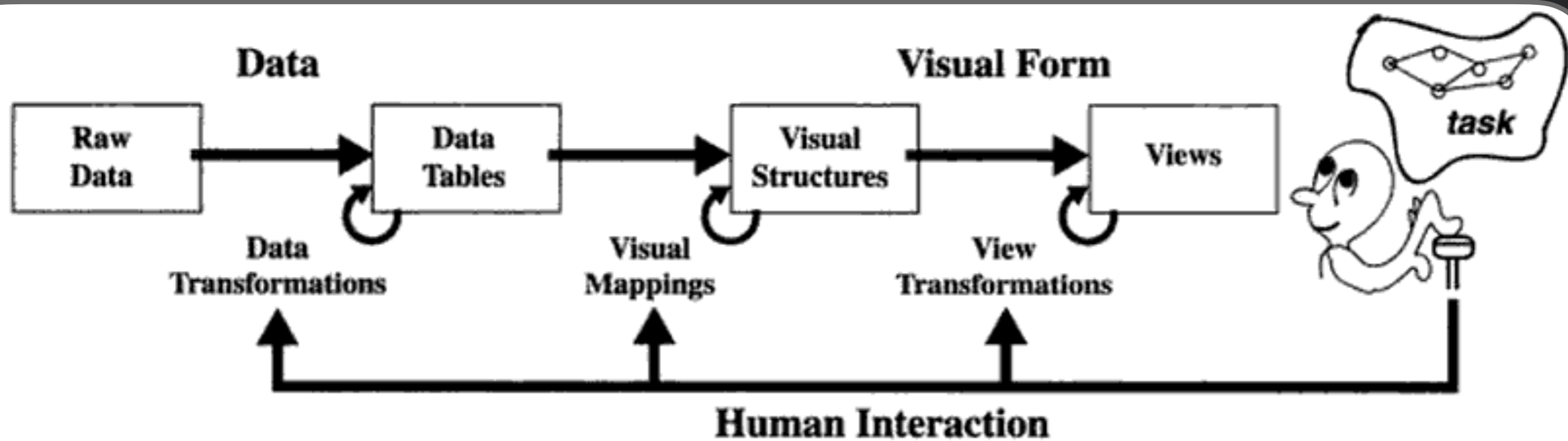
Week 15: Information Visualization



Amplifying Cognition

- Information Visualization can amplify cognition by:
 1. *Increasing the memory and processing resources available to users*
 2. *Reducing the search for information*
 3. *Using visual representations to enhance the detection of patterns*
 4. *Enabling perceptual inference*
 5. *Using perceptual attention mechanisms for monitoring*
 6. *Encoding Information in a manipulable medium*

Designing an Information Visualization



Raw Data: idiosyncratic formats

Data Tables: relations (cases by variables) + metadata

Visual Structures: spatial substrates + marks + graphical properties

Views: graphical parameters (position, scaling, clipping...)

Types of Raw Data

- Nominal - unordered set *without* a quantitative value
 - Gender: male, female
 - Hair color: brown, black, blonde, gray, orange, ...
- Ordinal - *ordered* set, with no meaning assigned to differences
 - How do you feel today: very unhappy, unhappy, ok, happy, very happy
 - Undefined how much better happy is than ok
- Quantitative - *numeric* value
 - Height, weight, distance, ...

Data Transformations

- Classing / binning: Quantitative \rightarrow ordinal
 - Maps ranges onto classes of variables
 - Can also count # of items in each class w/ histogram
- Sorting: Nominal \rightarrow ordinal
 - Add order between items in sets
- Descriptive statistics: mean, average, median, max, min, ...

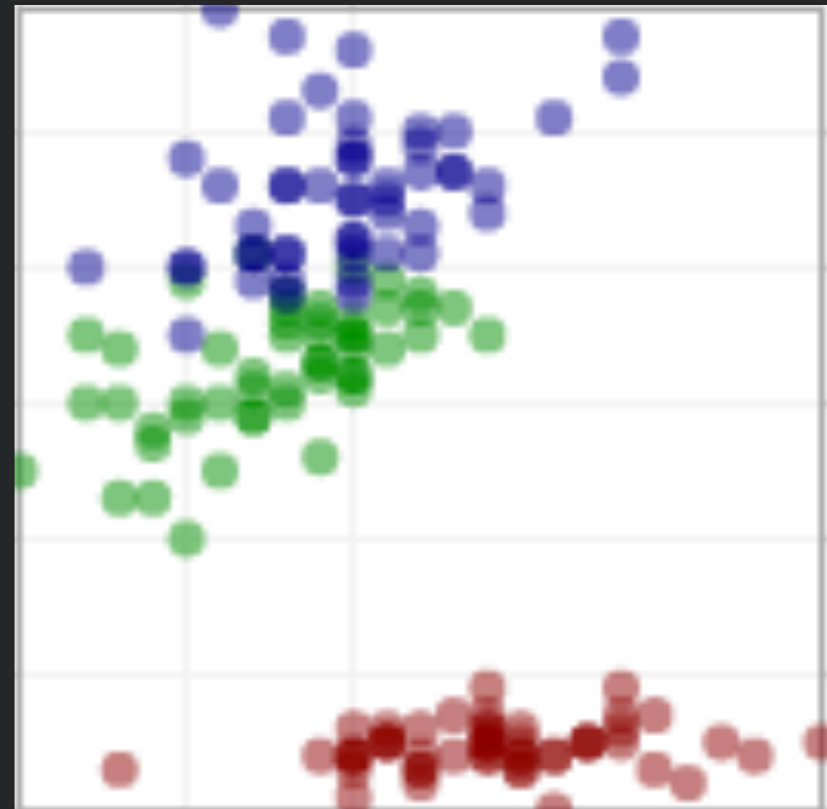


Visual Structures

- 3 components
 - spatial substrate
 - marks
 - marks' graphical properties

Spatial Substrate

- Axes that divide space
- Types of axes - unstructured, nominal, ordinal, quantitative
- Composition - use of multiple orthogonal axes (e.g., 2D scatterplot, 3D)





Marks

- Points (0D)
- Lines (1D)
- Areas (2D)
- Volumes (3D)

Marks' Graphical Properties

- Quantitative (Q), Ordinal (O), Nominal (N)
- Filled circle - good; open circle - bad

	Spatial	Object
Extent	(Position)	Gray Scale
	Size	
Dif-feren-tial	Orientation	Color
		Texture
		Shape

Effectiveness of Graphical Properties

- Quantitative (Q), Ordinal (O), Nominal (N)
- Filled circle - good; open circle - bad

		Spatial	Q	O	N	Object	Q	O	N
Extent	(Position)		●	●	●	Grayscale	◐	●	○
	Size		●	●	●				
Differential			◐	◐	●	Color	◐	◐	●
	Orientation					Texture	◐	◐	●
						Shape	○	○	●

Animation

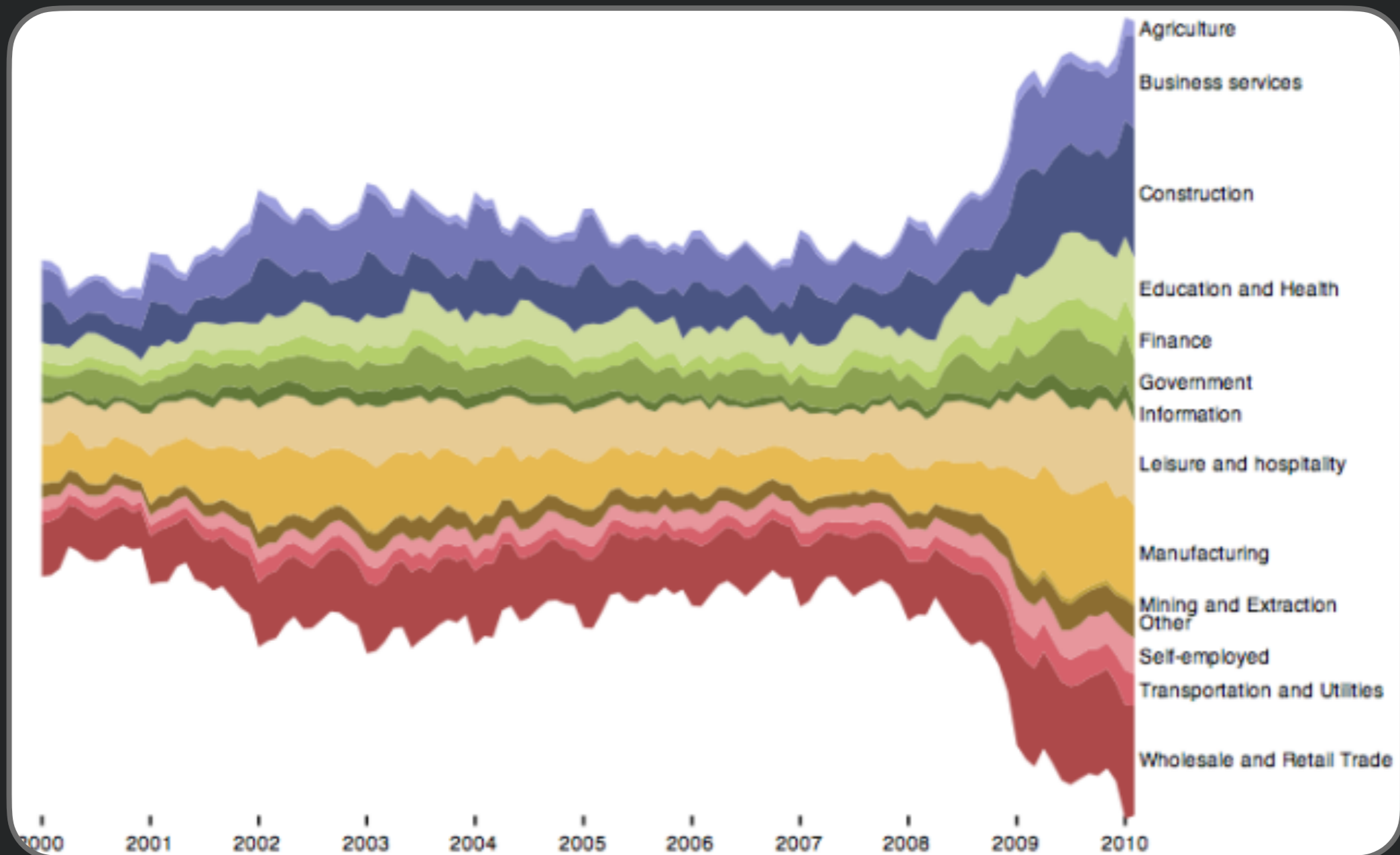
- Visualization can change over time
- Could be used to encode data as a function of time
 - But often not effective as makes direct comparisons hard
- Can be more effective to animate transition from before to after as user configures visualization

Time-series Data



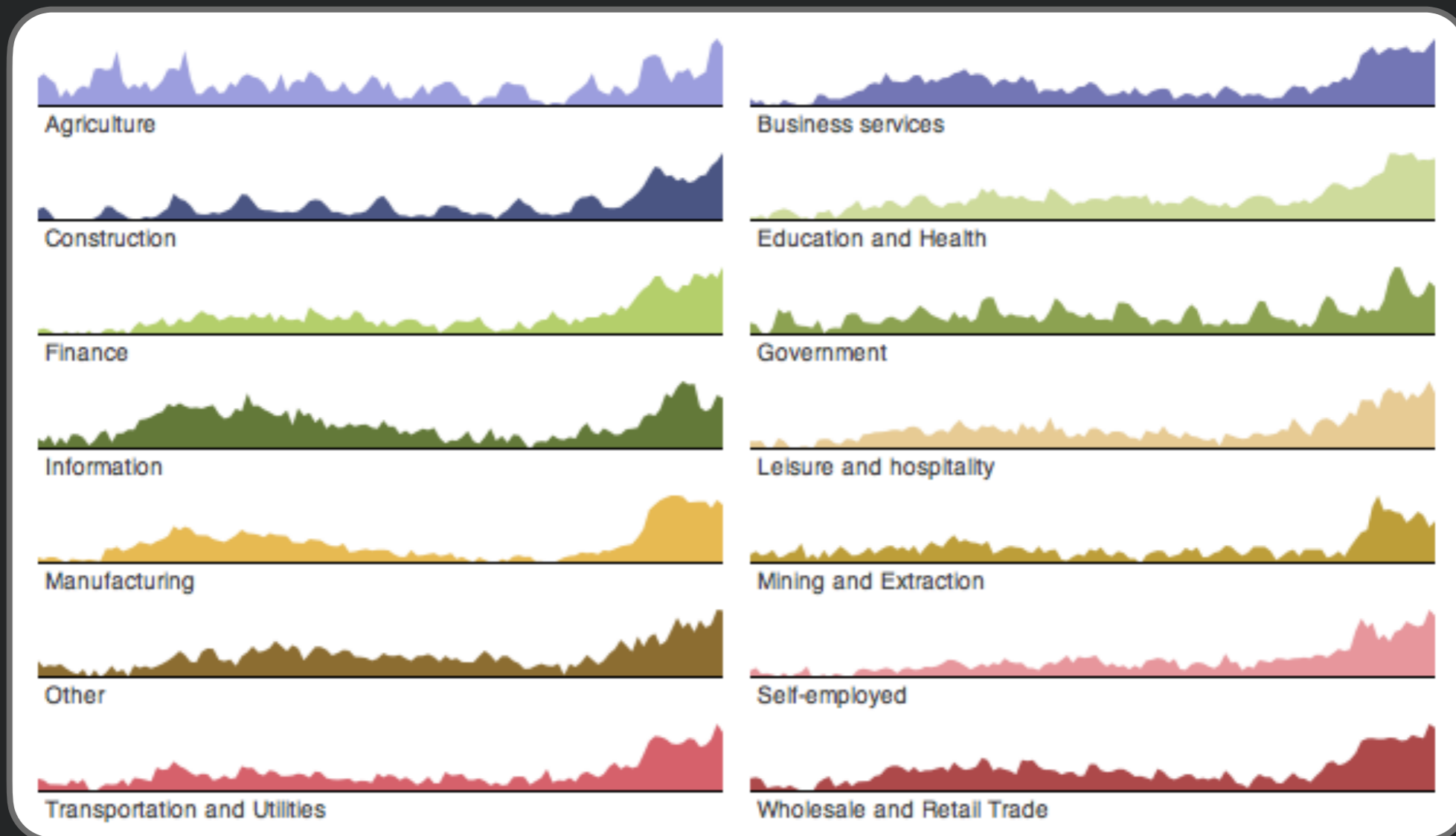
Stacked Graph

- Supports visual summation of multiple components



Small Multiples

- Supports separate comparison of data series
- May have better legibility than placing all in single plot

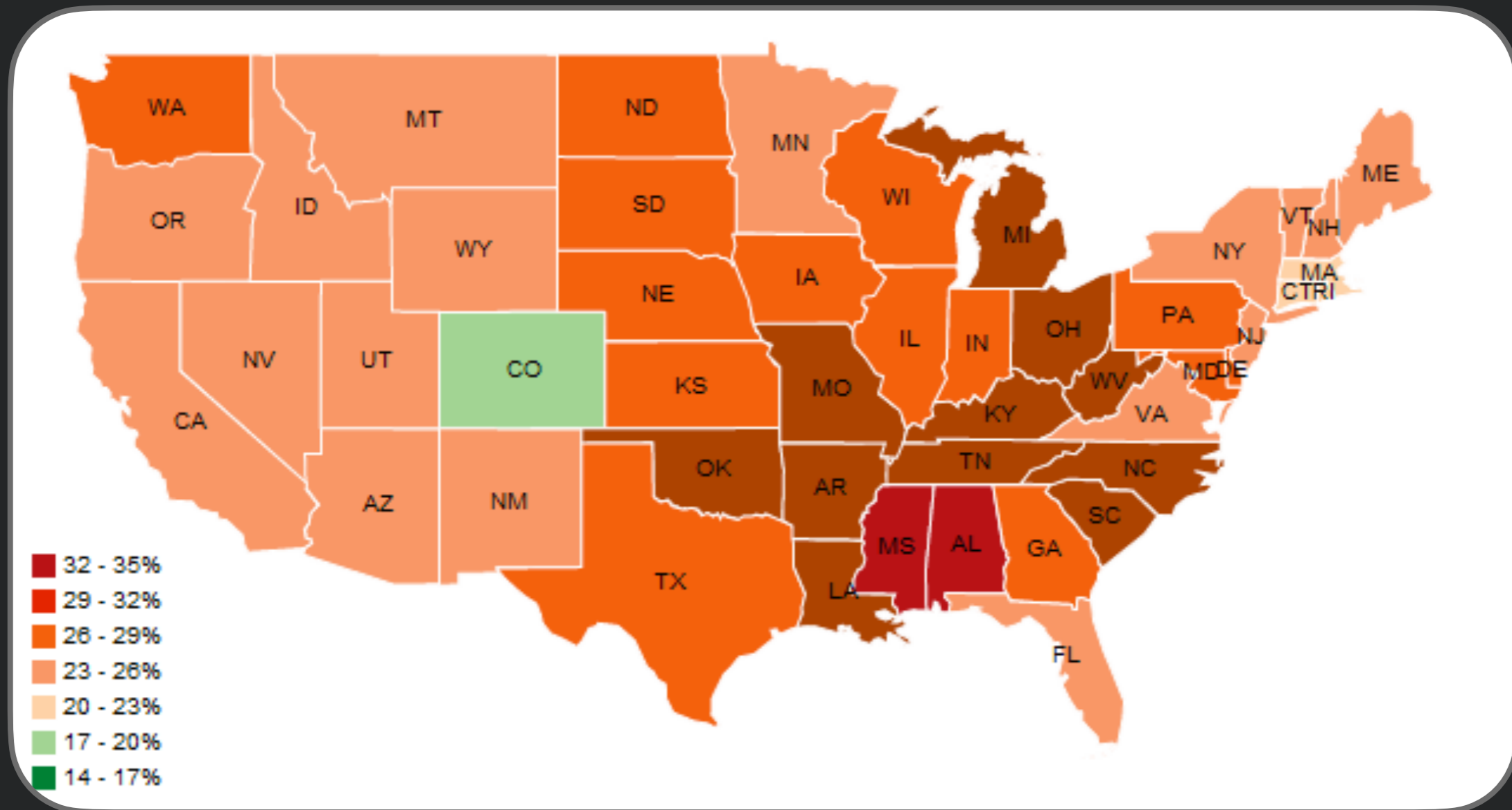


Maps



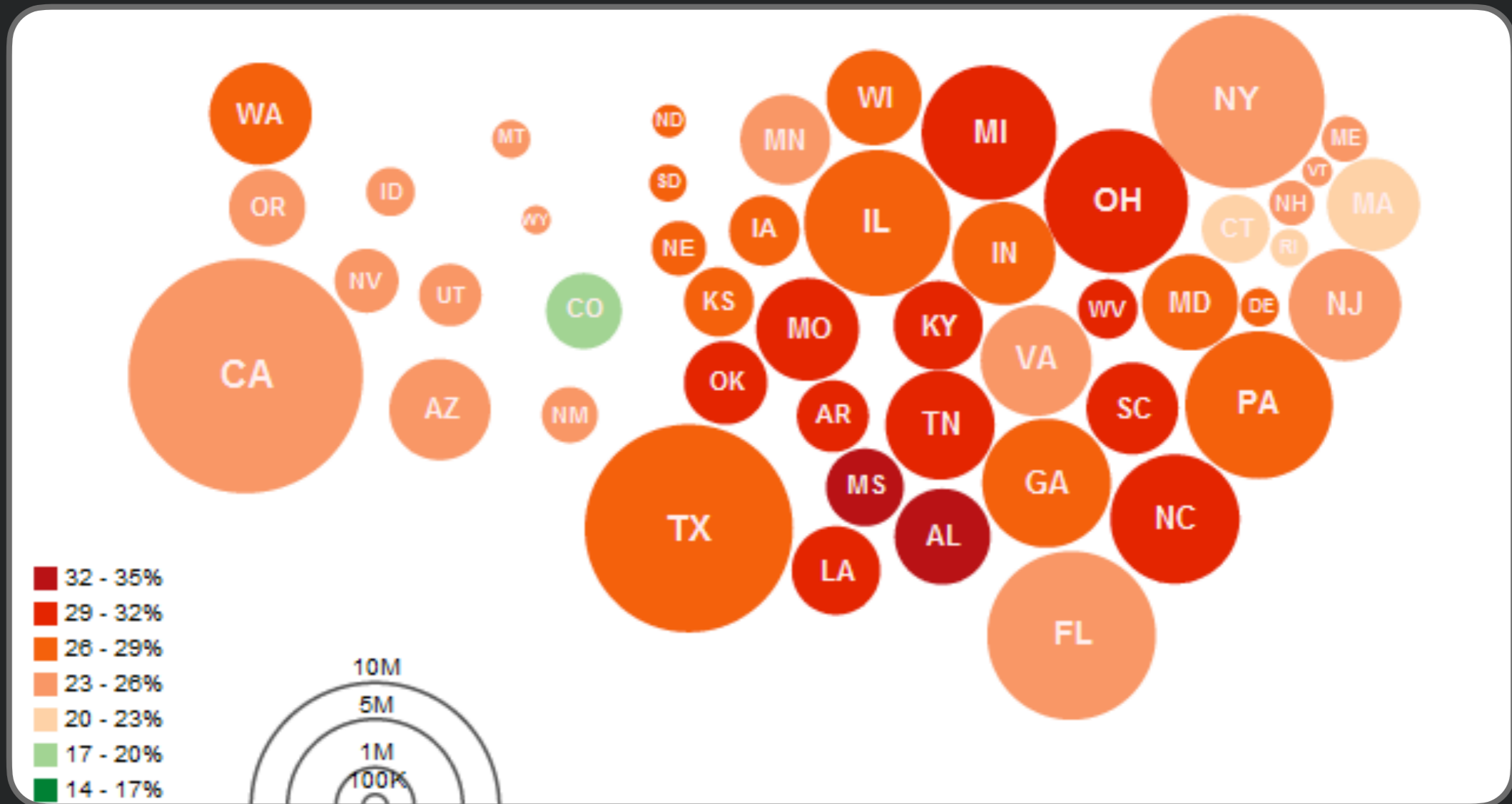
Choropleth Map

- Groups data by area, maps to color



Cartograms

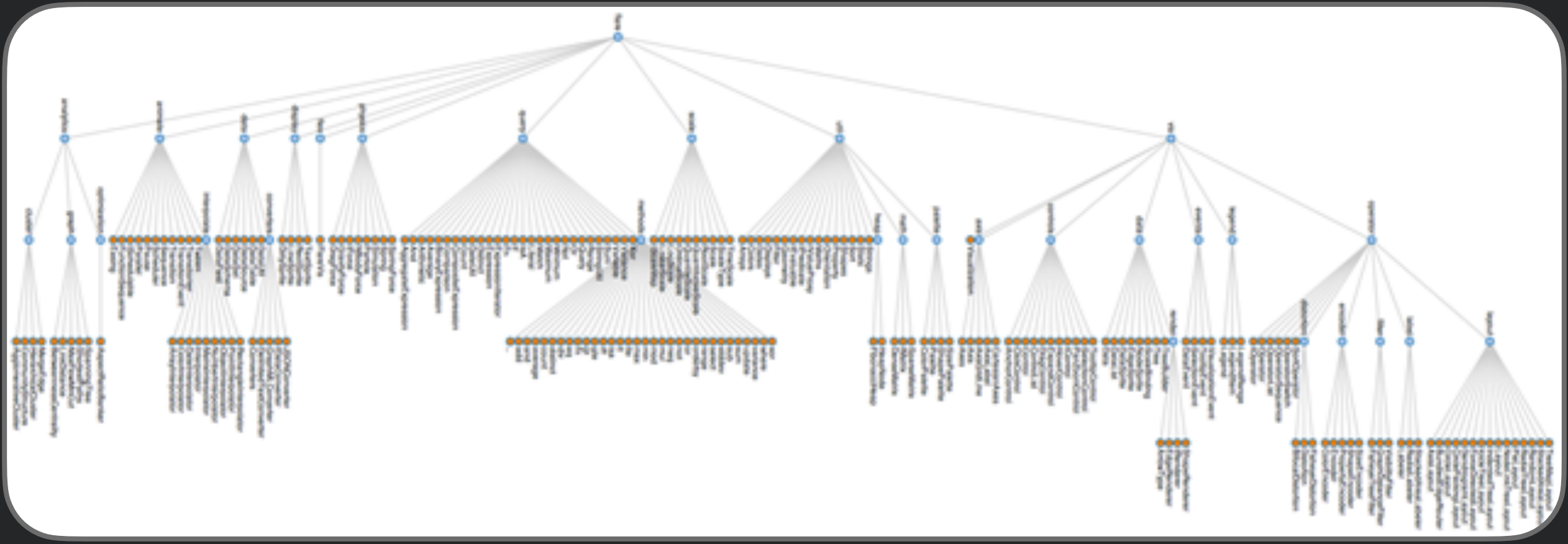
- Encodes two variables w/ size & color



Hierarchies



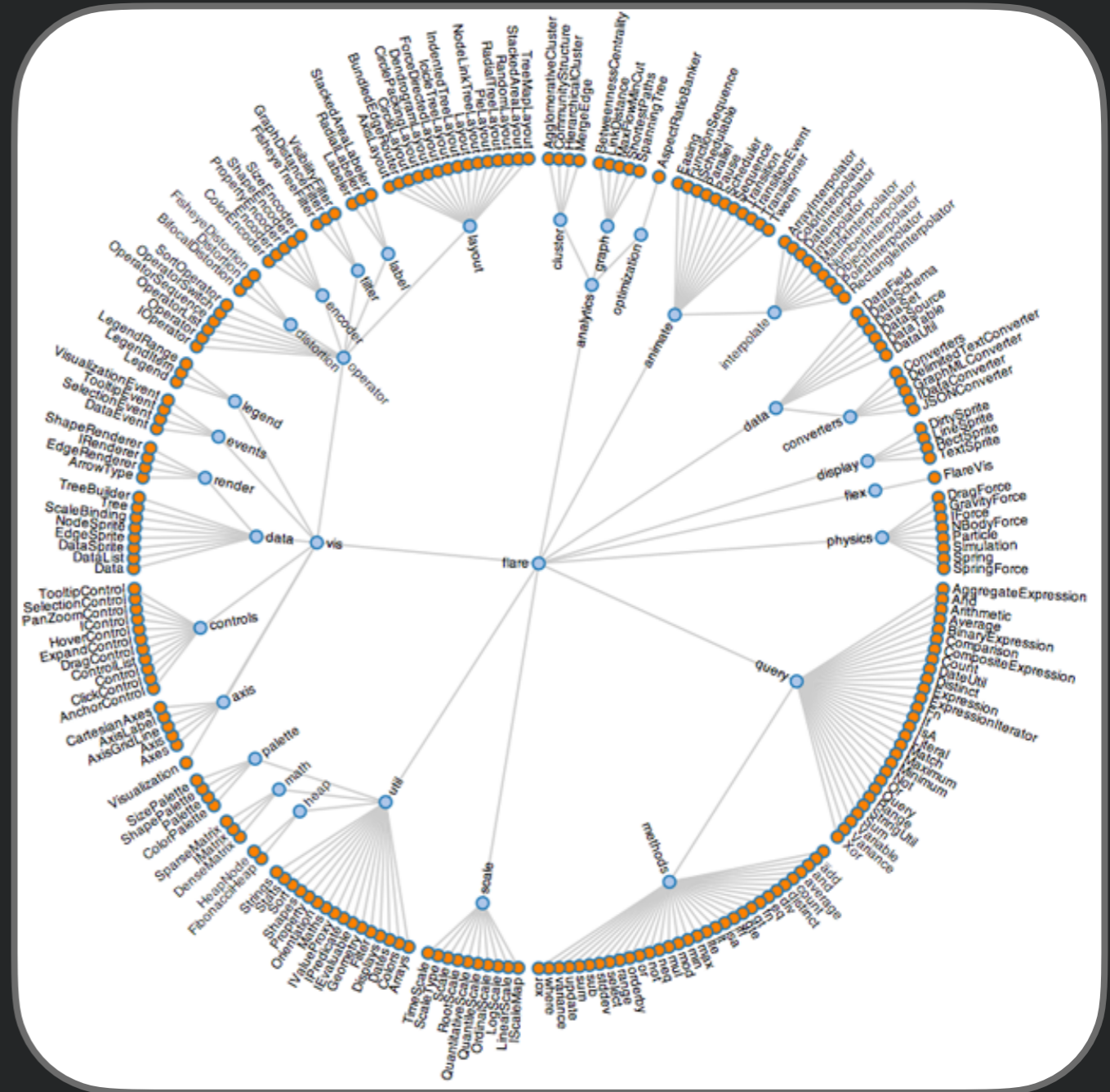
Node Link Diagram



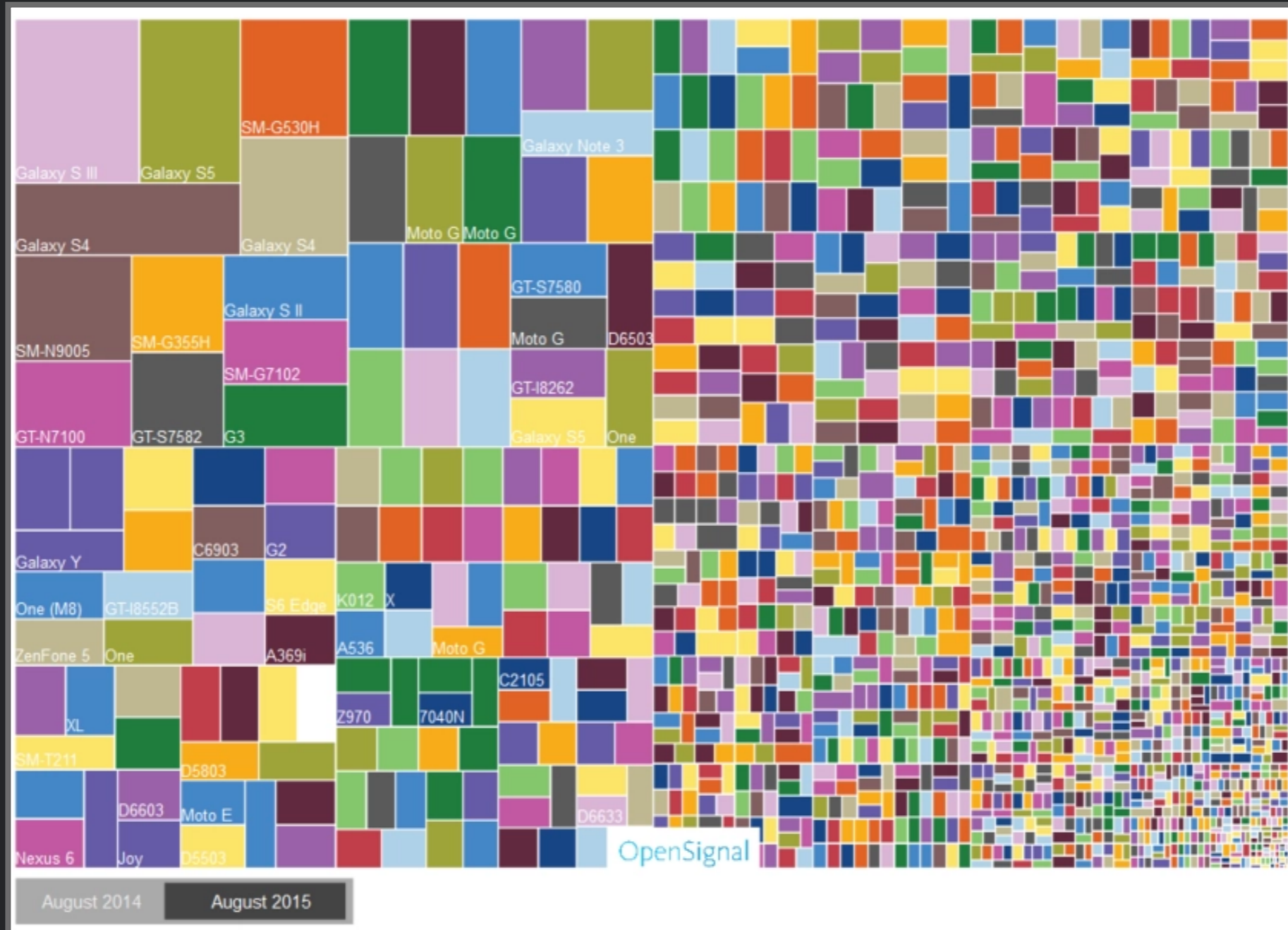


Dendrogram

- Leaf nodes of hierarchy on edges of circle



Treemaps

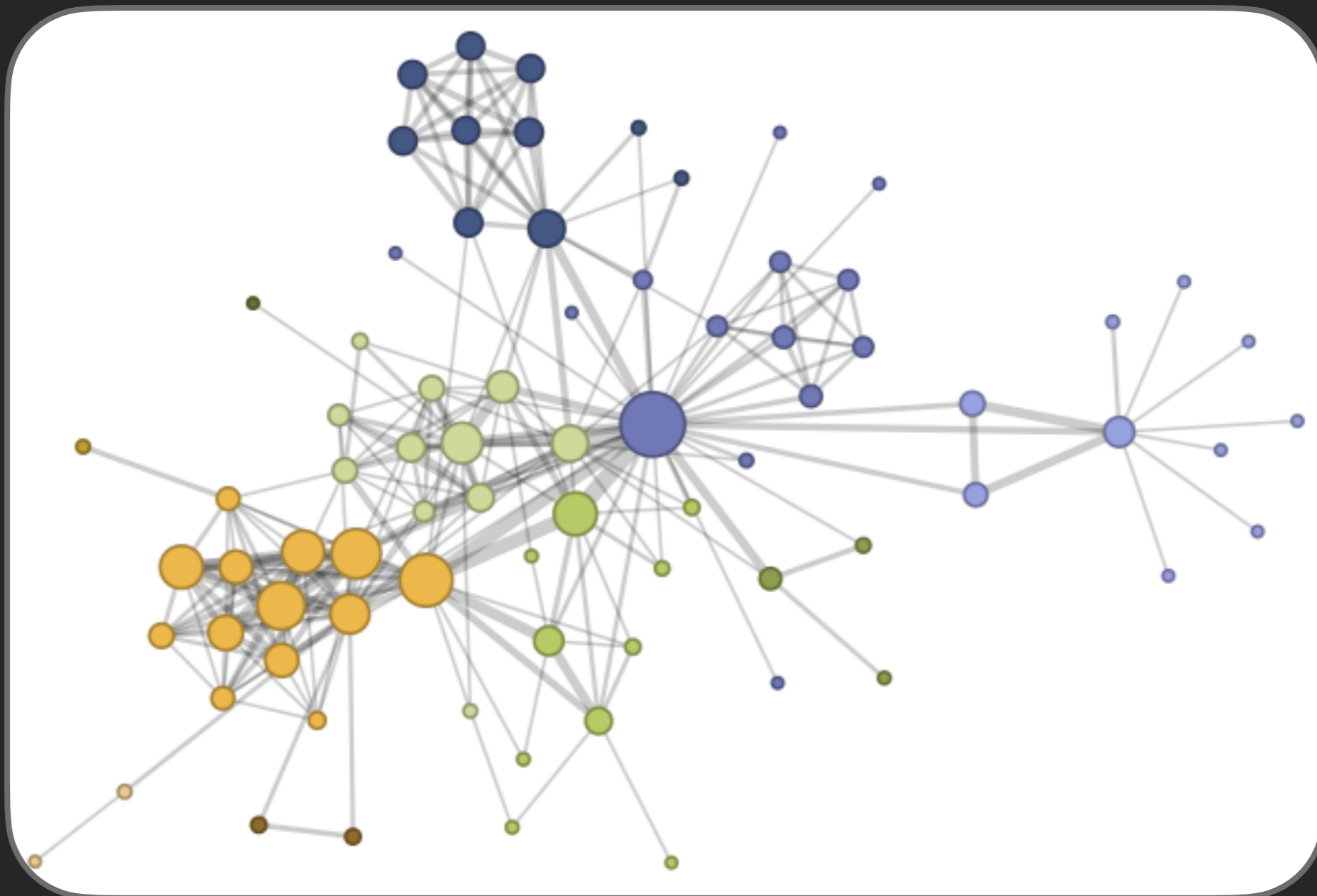


Networks



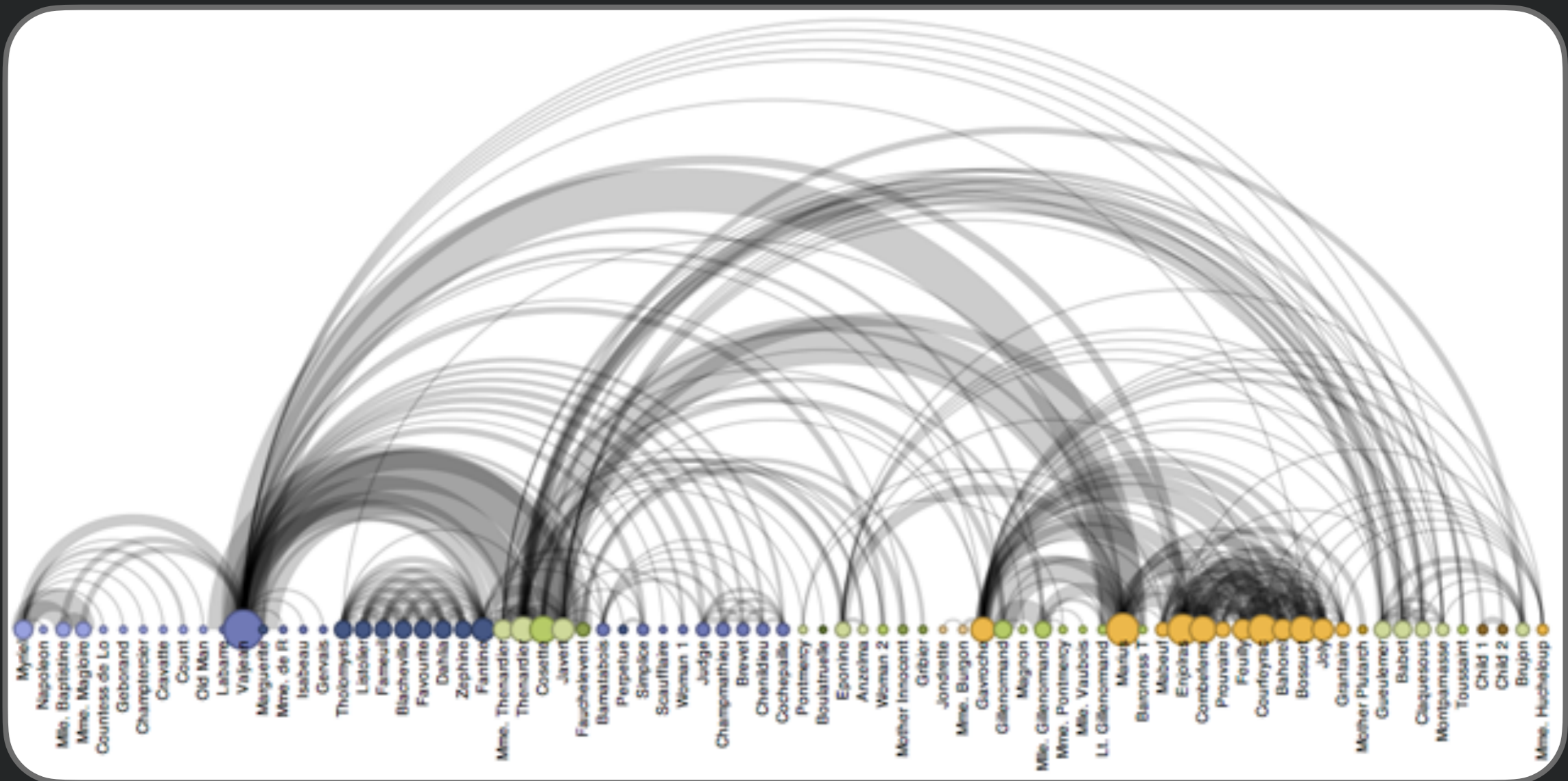
Force-directed Layout

- Edges function as springs, find least energy configuration



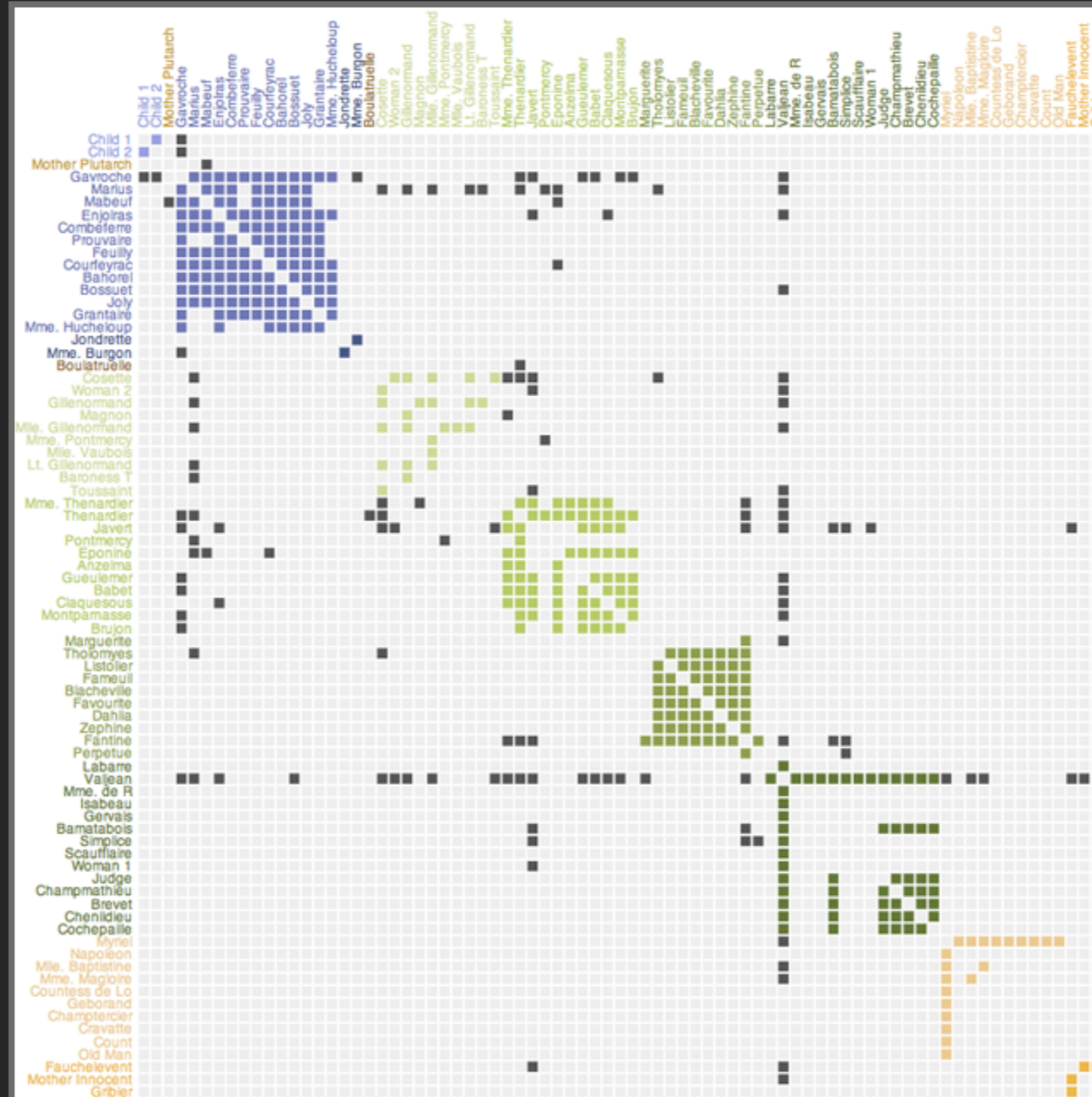
Arc Diagram

- Can support identifying cliques & bridges w/ right order





Adjacency Matrix





Tufte's principles of graphical excellence

- Show the *data*
- Induce the viewer to think about the substance rather than the methodology
- Avoid distorting what the data have to say
- Present *many* numbers in a small space
- Make large data sets *coherent*
- Encourage the eye to *compare* different pieces of data
- Reveal data at several levels of detail, from overview to fine structure
- Serve reasonable clear *purpose*: description, exploration, tabulation, decoration



Design Principles for Data-ink

- (a.k.a. aesthetics & minimalism / elegance & simplicity)
- **Above all else show the data**
 - Erase non-data-ink, within reason
 - Often not valuable and distracting
 - Redundancy not usually useful



Interactive Visualizations

- Users often use iterative process of making sense of the data
 - Answers lead to new questions
- Interactivity helps user constantly change display of information to answer new questions
- Should offer visualization that offers best view of data moment to moment as desired view changes



Information Visualization Tasks

- **Overview:** gain an overview of entire collection
- **Zoom:** zoom in on items of interest
- **Filter:** filter out uninteresting items
- **Details on Demand:** select an item or group and get details
- **Relate:** view relationships between items
- **History:** support undo, replay, progressive refinement
- **Extract:** allow extraction of sub-collections through queries

In-Class Coding Example Exam Problem





Activity: Final Exam Coding Practice

- Imagine you are building an app that implements a ToDo List. You've already implemented the logic adding ToDo Items. Now you'd like to do two things. You'd first like display added ToDo Items to the user. To do so, you decide to create a new child component, `ToDoItem`, which is initialized with text describing the task. Second, you want to add a button to the `ToDo` component which, when clicked, deletes the last added task. Implement the Functionality for each of these features.

Class-Based Example: <https://replit.com/@kmoran/SWE-432-Final-Exam-Practice-Class#src/App.jsx>