阿里云
奥运会全球指定云服务商

React 结合 D3
数据可视化最佳实践

高翔（云由）/ 阿里云 / @iWeb

GeekPlux
geekplux

Full-time Learner, good at data visualization and front-end development, interested in too much things.

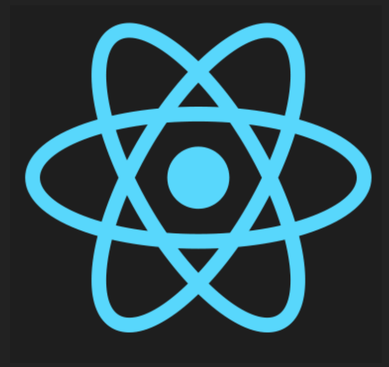
Web Developer

Data Visualization

https://geekplux.com/
https://github.com/geekplux

经过尴尬的自我介绍以后

我为什么讲这个话题



React

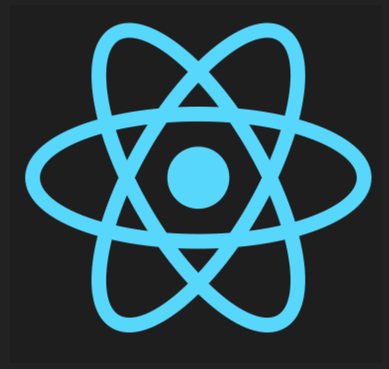
110k Stars

最主流的 Web 框架

D3

80k Stars

最知名可视化库



React

110k Stars

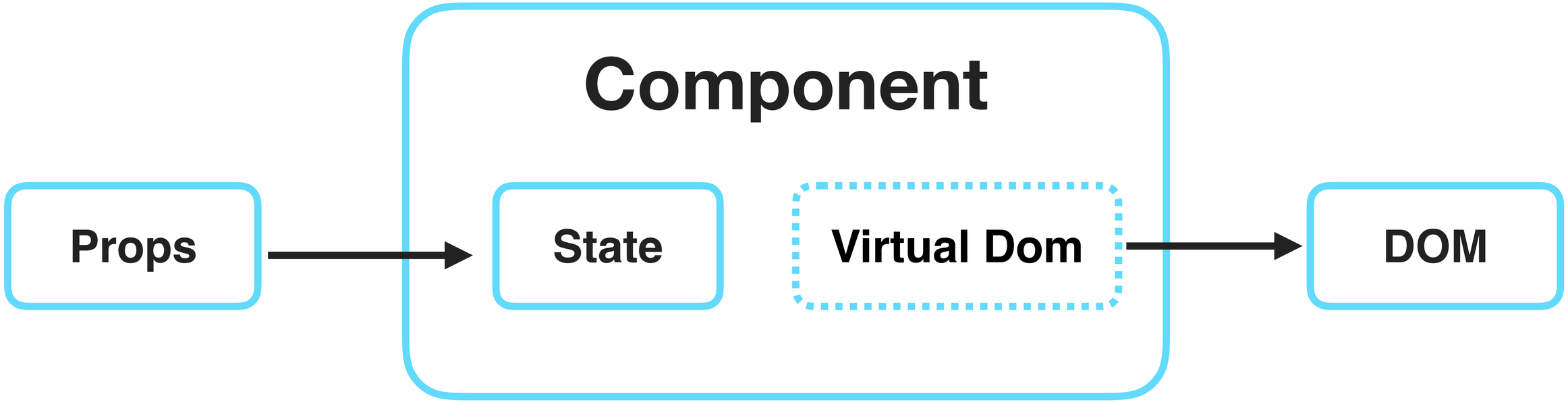
最主流的 Web 框架

D3

80k Stars

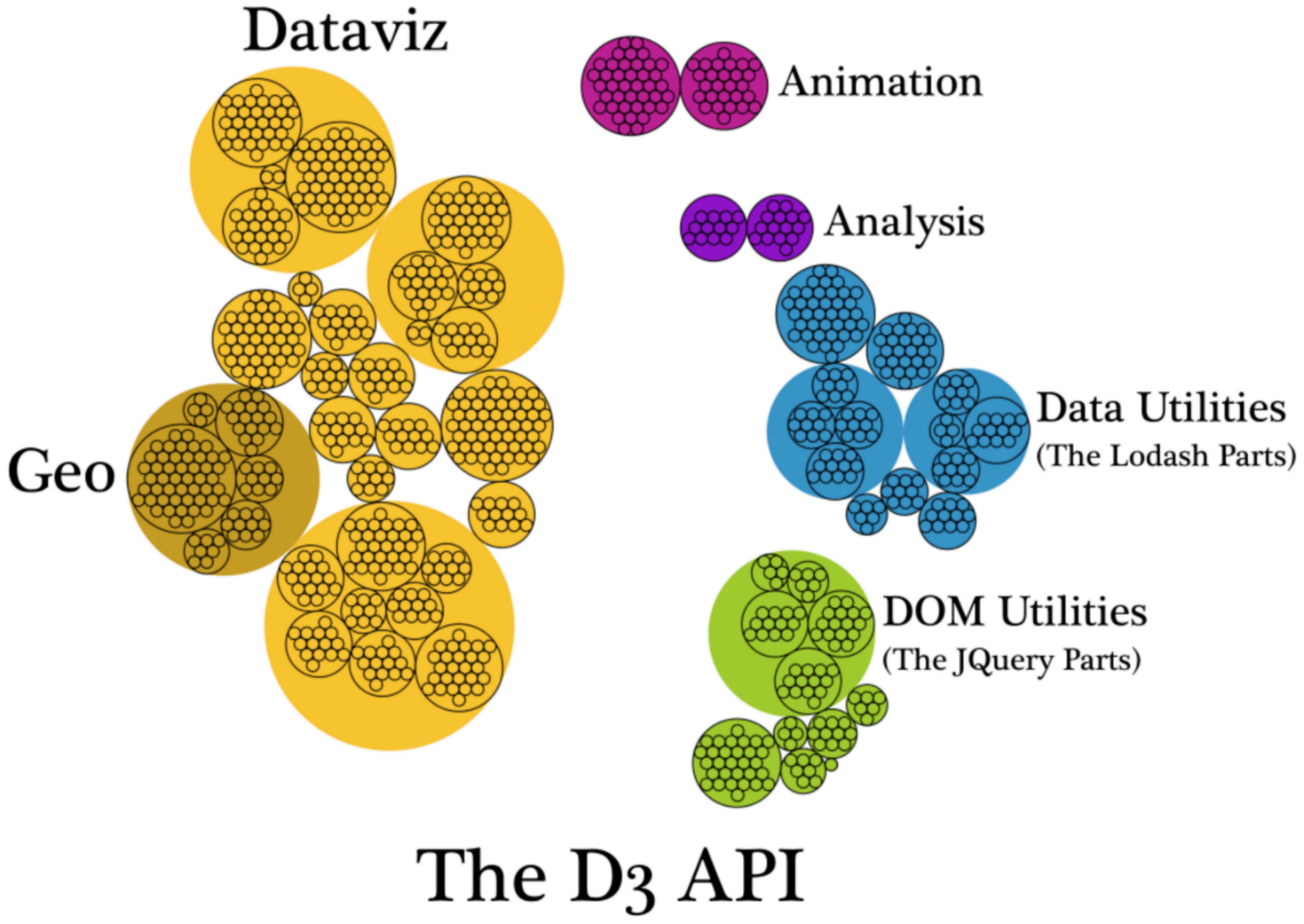
最知名可视化库

React



Data-Driven Documents

“Not just a data visualization library”



Use it like jQuery

```
const svg = d3.select('body')
    .append('svg')
    .attr('width', width)
    .attr('height', height)
    .append('g')
    .attr('transform', `translate(${width / 2}, ${height / 2})`);
```

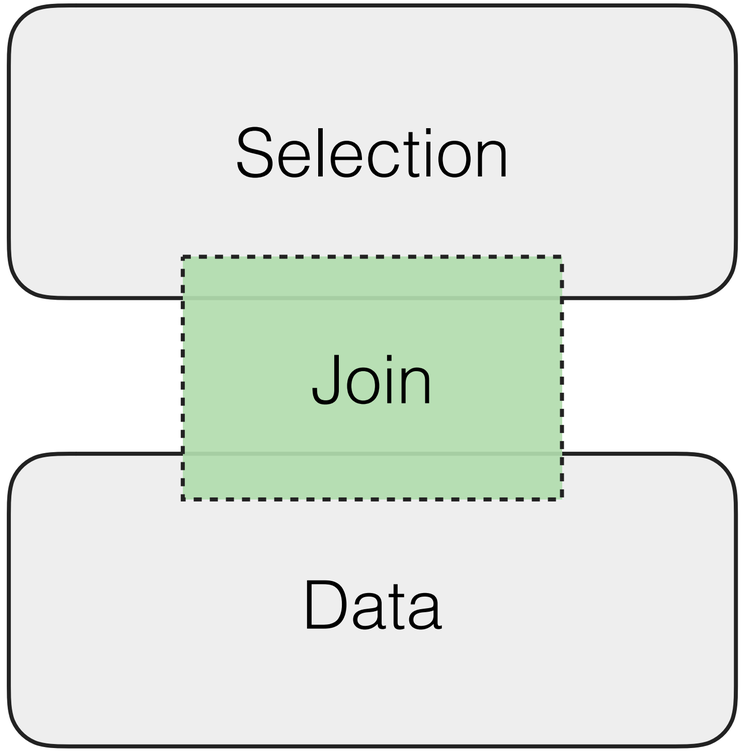
I have data binding, TOO !

Data Binding + Rendering

```
svg.selectAll("circle")
   .data(data)
   .enter().append("circle")
      .attr("cx", function(d) { return d.x; })
      .attr("cy", function(d) { return d.y; })
      .attr("r", 2.5);
```

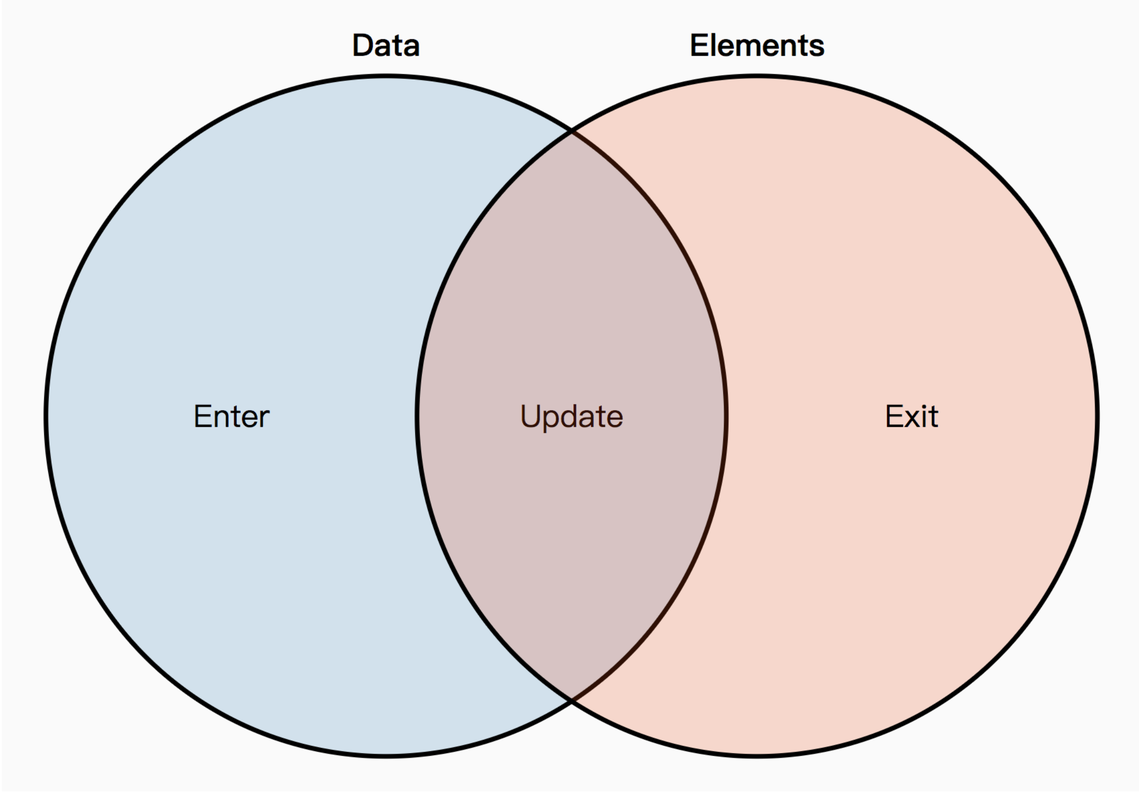
Selection 选择器

```
svg.selectAll("circle")
```



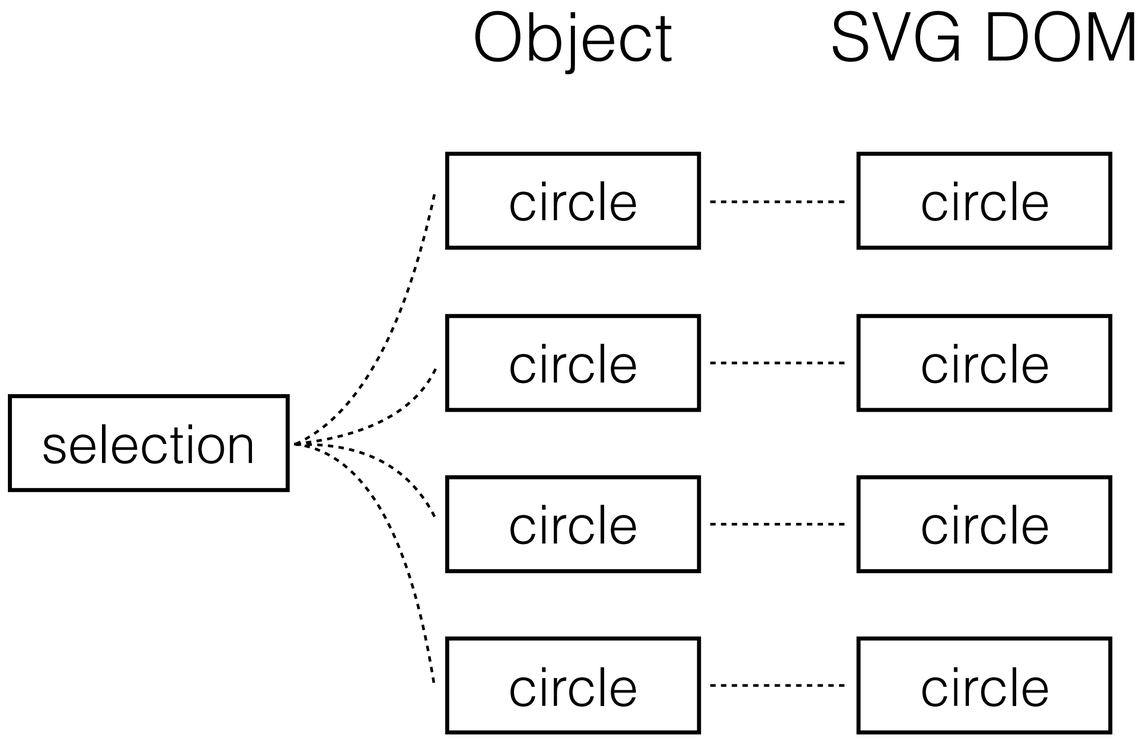
data Enter()

```
svg.selectAll("circle")
    .data(data)
    .enter()
```



Render by Append

```
svg.selectAll("circle")
    .data(data)
    .enter()
    .append("circle")
    .attr("r", 2.5);
```



React + D3

最主要的冲突

冲突	**数据**	**渲染**
React	单向数据流	Virtual DOM
D3	数据绑定 手动更新	JS Obejct 直接操作 DOM

阿里云
奥运会全球指定云服务商

终于说到

解决方案

方案竟然有三种

这就是为什么我能讲这么久的原因

方案一

D3 manipulate

+

React container

方案一 *D3 manipulate* + *React container*

- React 渲染一个空 div 作为容器
- 容器中的 DOM 数据/操作/渲染 都由 D3 负责

```
class MyComponent extends React.Component {
  setRef = ref ⇒ this.svgRef = ref;

  componentDidMount(){
    // use D3 to bind data and render DOMs here
    this.d3svg = d3.select(this.svgRef);
    //  ...  some d3 code
  }

  render() {
    return (
      <svg ref={this.setRef}></svg>
    );
  },
};
```

方案一

阿里云 奥运会全球指定云服务商

优势	限制
发挥 D3 所有 API	有两个数据/状态管理
直接复用 D3 示例代码	自己手动管理生命周期
动画方便	过程式的代码风格
性能	

方案一 总而言之

- 不符合 React 的代码风格，丧失 React 优势
- 适合希望手动精细操纵 DOM 的人

方案二

React structure

+

D3 compute

方案二 *React structure + D3 compute*

- DOM/SVG 结构都由 React 负责
- D3 负责数据/布局计算

```
class MyComponent extends React.Component {
  componentDidMount() {
    // some d3 computation
    const simulation = d3.forceSimulation()
        .force("link", d3.forceLink().id(function(d) { return d.id; }))
        .force("charge", d3.forceManyBody())
        .force("center", d3.forceCenter(width / 2, height / 2));
    simulation.nodes(graph.nodes).on("tick", ticked);
  }

  render() {
    return (
      <svg width={width} height={height}>
        <g className='node'>
          {this.nodes.map((node, i) => {
            return (<circle r={node.radius} x={node.x} y={node.y} key={i}/>);
          })}
        </g>
      </svg>
    );
  },
});
```

方案二

优势	限制
数据/状态管理靠 React	生命周期中的 D3 计算管理
DOM/SVG 结构清晰	动画
声明式的代码风格	无法直接复用 d3 示例
	D3 很多 API 无法使用
	性能
	组件只能在 React 生态中用

方案二 总而言之

- 充分发挥 React 及其生态优势
- 适合需要代码风格统一，且图表不太复杂的场景

方案三

react-faux-dom

“is a way to use existing D3 tooling but render it through React in the React ethos.”

– Oliver Caldwell react-faux-dom author

方案三 *react-faux-dom*

- 是一个 react 高阶组件
- 仿制一层虚拟 DOM 可供 D3 操纵
- 同时把这层 DOM 套上状态管理传给 React

```
class SomeChart extends React.Component {
  render () {
    // Create your element.
    var el = ReactFauxDOM.createElement('div')

    // Change stuff using actual DOM functions.
    // Even perform CSS selections!
    el.style.setProperty('color', 'red')
    el.setAttribute('class', 'box')

    // Render it to React elements.
    return el.toReact()
  }
}
```

```
import React from 'react'
import * as d3 from 'd3'
import {withFauxDOM} from 'react-faux-dom'

class MyReactComponent extends React.Component {
  componentDidMount () {
    const faux = this.props.connectFauxDOM('div', 'chart')
    d3.select(faux)
      .append('div')
      .html('Hello World!')
    this.props.animateFauxDOM(800)
  }

  render () {
    return (
      <div>
        <h2>Here is some fancy data:</h2>
        <div className='renderedD3'>
          {this.props.chart}
        </div>
      </div>
    )
  }
}

MyReactComponent.defaultProps = {
  chart: 'loading'
}

export default withFauxDOM(MyReactComponent)
```

方案三

优势	限制
D3 示例代码直接复用	学习/使用成本
(大部分) D3 API 可用	有的 DOM API 需要自己搬迁
数据/状态管理靠 React	实际还是需要关心生命周期
动画	过程式代码风格
服务端渲染	性能?
	复杂交互(brush,drag..)

方案三 总而言之

- 复杂图表/交互可行性不高（性能）
- 需要使用者对 react 和 d3 都精通

目前没有完美的方案

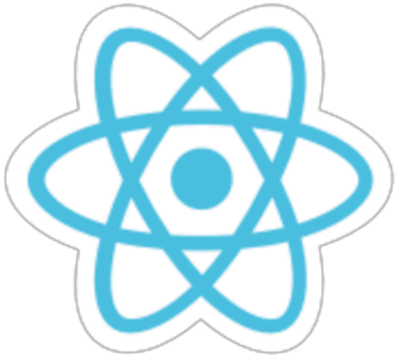
说了这么多废话

D3 within React

React Lifecycle Methods

React Faux DOM

D3 for the Math, React for the DOM



阿里云
奥运会全球指定云服务商

你可能不知道的

业界已有的方案

1

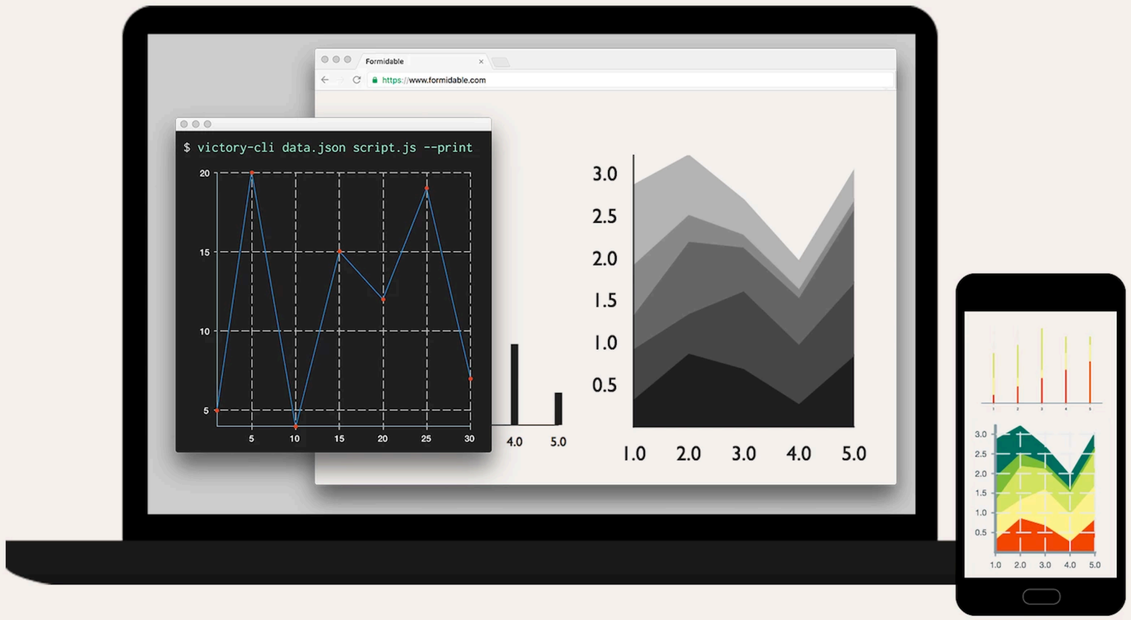
Victory

https://formidable.com/open-source/victory/

VICTORY™
by Formidable OPEN SOURCE

ABOUT DOCS FAQ GUIDES GALLERY SUPPORT GITHUB

React.js components *for* modular charting *and* data visualization




```
npm install victory
```

2

VX

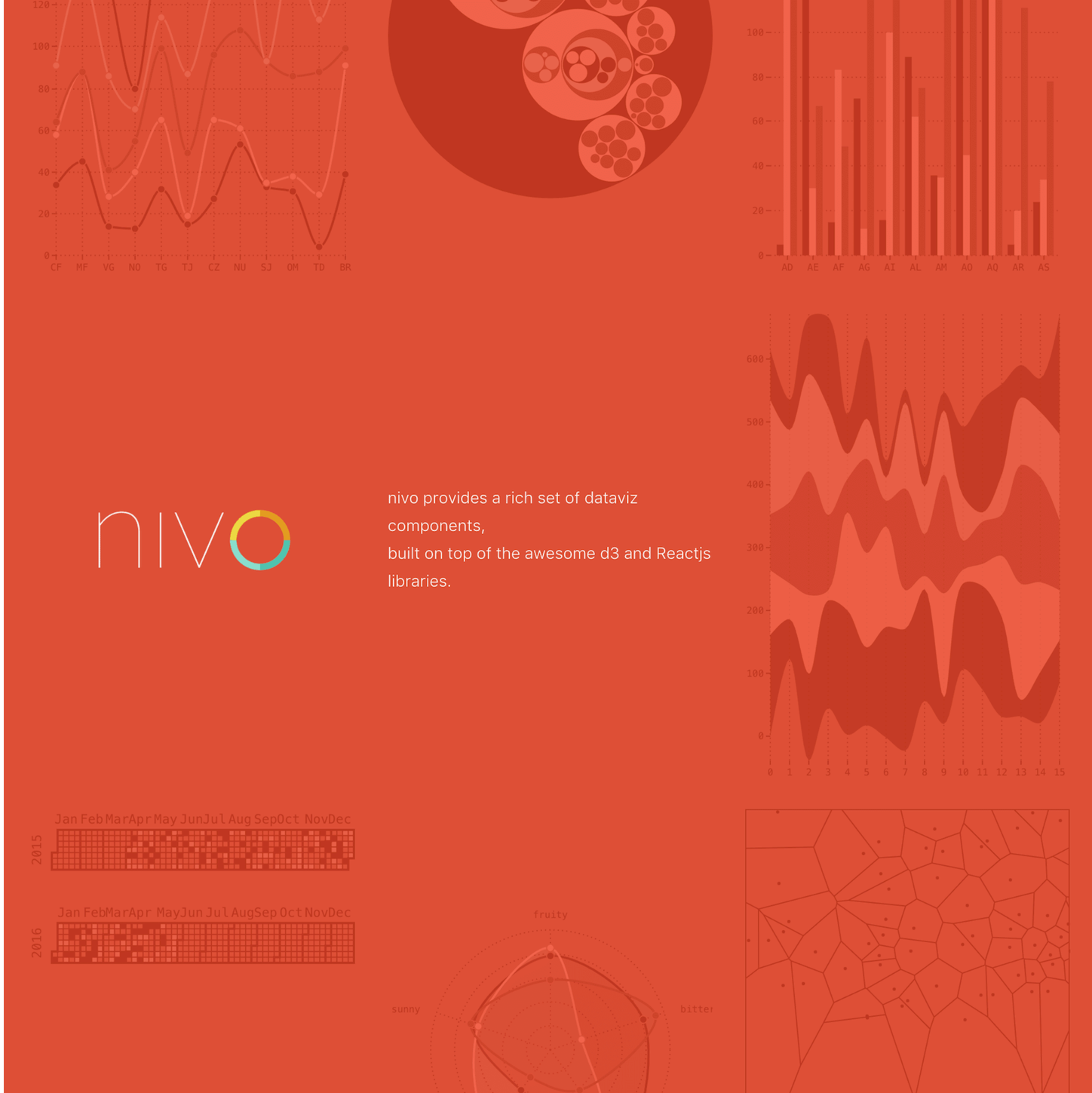
VIEW ON GITHUB

`vx` is collection of reusable low-level visualization components. `vx` combines the power of `d3` to generate your visualization with the benefits of `react` for updating the DOM.

https://vx-demo.now.sh/

3

nivo



http://nivo.rocks

4

Recharts

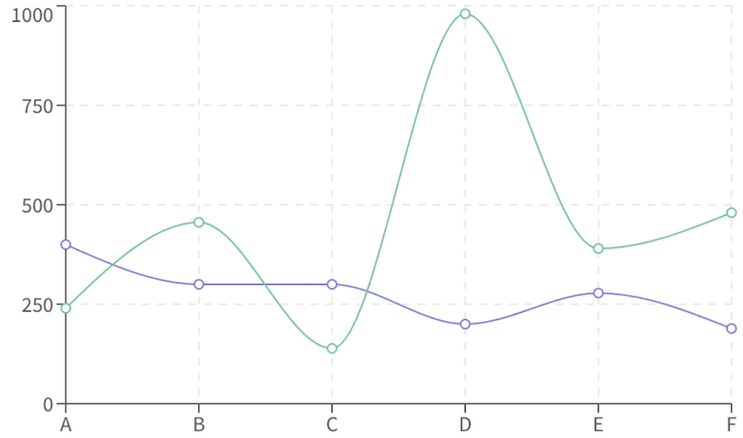
http://recharts.org

Recharts

A composable charting library built on React components

Install v1.0.0-beta.7

Star 7,573




```
<LineChart width={500} height={300} data={data}>
  <XAxis dataKey="name"/>
  <YAxis/>
  <CartesianGrid stroke="#eee" strokeDasharray="5 5"/>
  <Line type="monotone" dataKey="uv" stroke="#8884d8" />
  <Line type="monotone" dataKey="pv" stroke="#82ca9d" />
</LineChart>
```

也可以考虑

其他的可视化库

声明式/配置型 可视化库

- **Echarts** https://ecomfe.github.io/echarts-doc/public/en/index.html
- **BizCharts** http://bizcharts.net
- **Vega & Vega-lite** https://vega.github.io/

又到了尴尬的提问时间

Thank you.

https://geekplux.com/
https://github.com/geekplux

GeekPlux
geekplux

Full-time Learner, good at data visualization and front-end development, interested in too much things.