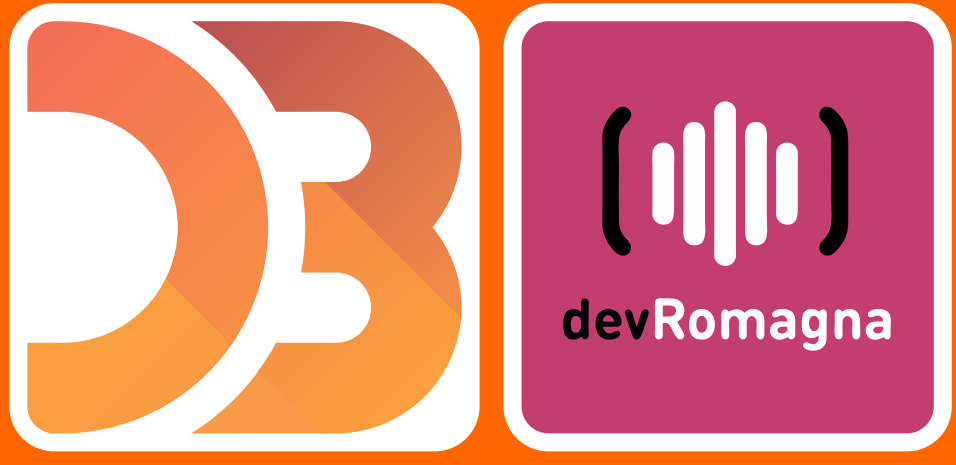
Data visualization with D3.js

Imola, 2019-01-17



Massimo Artizzu

Web dev & architect

at Antreem

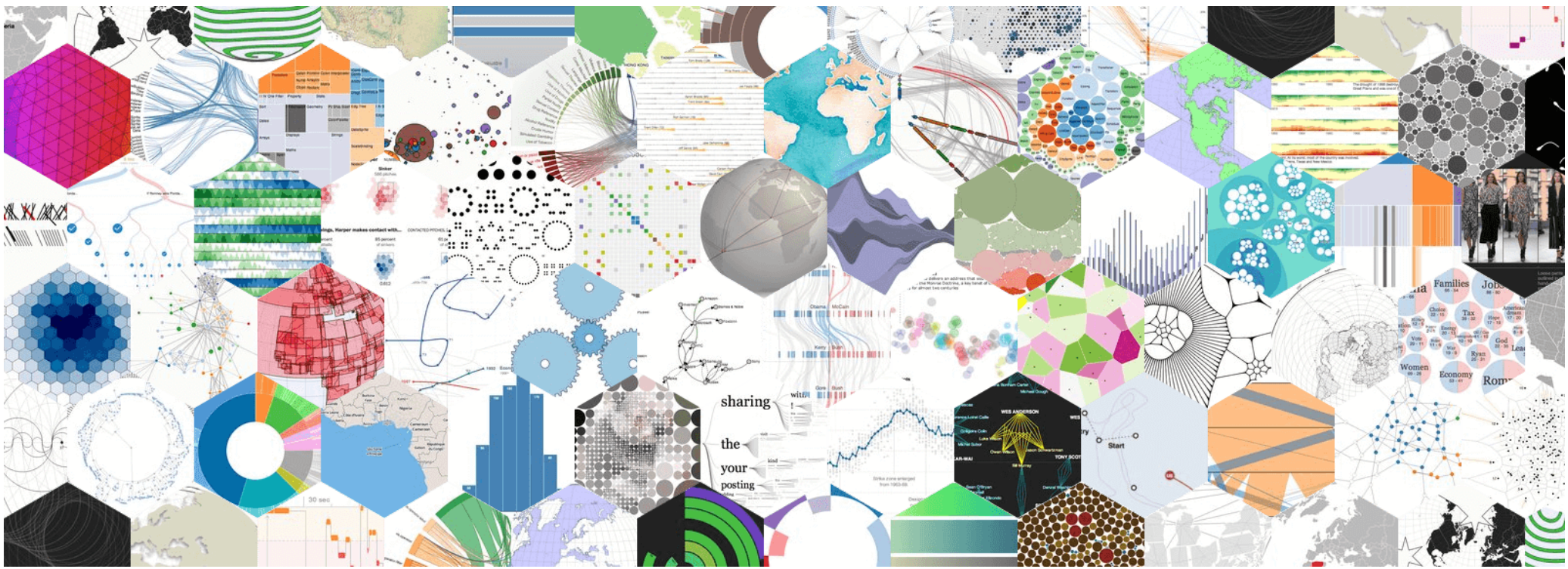
/

@MaxArt2501

You can find these slides at

maxart2501.github.io/d3-talk/devromagna/

What *is* D3.js? 🤔


sharing
with
the
your
posting
Families
Jobs
Tax
Women
Economy
Start

It's *not*

a chart library 😱

It's a *data visualization* library

You *can* draw charts...
... or not!

D3 = DDD = ...

Data
Driven
Document

Loading: the classic way

```
<script src="d3-array.min.js"></script>
<script src="d3-collection.min.js"></script>
<script src="d3-color.min.js"></script>
<script src="d3-format.min.js"></script>
<script src="d3-interpolate.min.js"></script>
<script src="d3-time.min.js"></script>
<script src="d3-time-format.min.js"></script>
<script src="d3-scale.min.js"></script>

d3.select('rect') ...
```

Loading: the module way

```
import { select } from 'd3-selection';
import { scaleLinear } from 'd3-scale';
import { axisLeft } from 'd3-axis';
import { arc, pie } from 'd3-shape';

const scale = scaleLinear();
select('rect') ...
```

The basics

`select` and friends

Think of jQuery…

… but it sucks 🤦

```
import { select } from 'd3-selection';

const app = select('#app')
    .text('Hello, world!')
    .attr('id', 'hello')
    .classed('done', true);

const chk = app.append('input')
    .attr('type', 'checkbox')
    .property('checked', true)
    .on('click', handleClick);
```

```
import { select, selectAll } from 'd3-selection';

selectAll('.power-of-2 li')
    .attr('data-index', (_, i) => i)
    .each(function(_, i) {
        const li = select(this);
        li.append('span').text(2 ** i);
    });
```

codesandbox.io/s/2p3o13jx5j

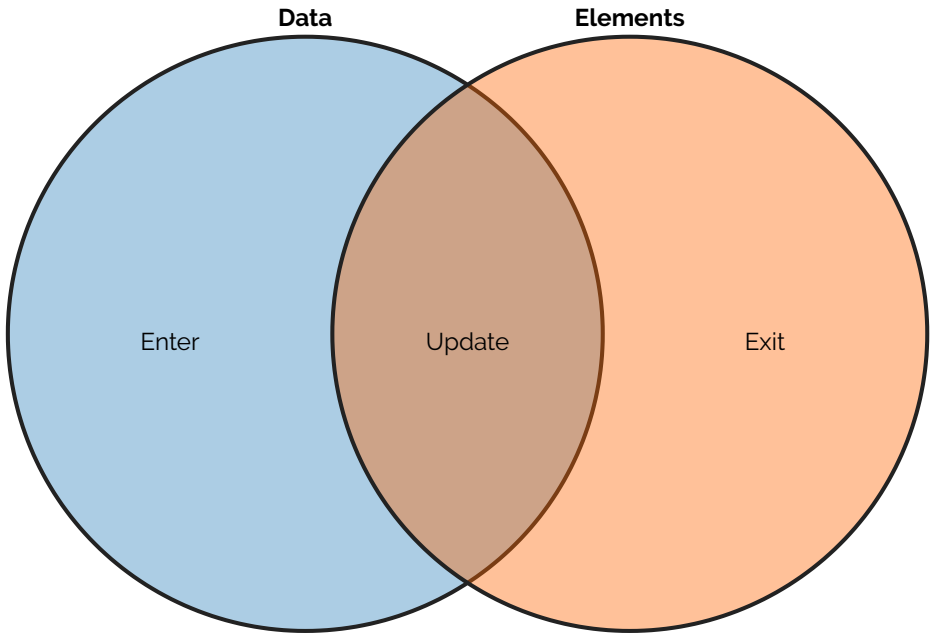
Data joins

```
<ul id="names"></ul>

const names = [
  'Alice',
  'Bruno',
  'Camilla'
];
```

- Alice
- Bruno
- Camilla

```
const items = select('ul#names')
  .selectAll('li.name')  // 😮
  .data(names);          // 🤔

items.enter()
  .append('li')
  .attr('class', 'name')
  .text((name, i) => name);
```

`items` is a so-called *data join*.



[Mike Bostock - Thinking with Joins]

Add missing elements

```
items.enter()
  .append('li')
  .attr('class', 'name')
  .text(function(name, index) {
    return name;
  });
```

- Appended to `ul#names`
- Must match `li.name`
- `name` = Alice, …
  `index` = 0, 1, …
  `this` = `<li>`

Update existing elements

```
items
  .text(function(name, index) {
    return name;
  });
```

Or in one pass…

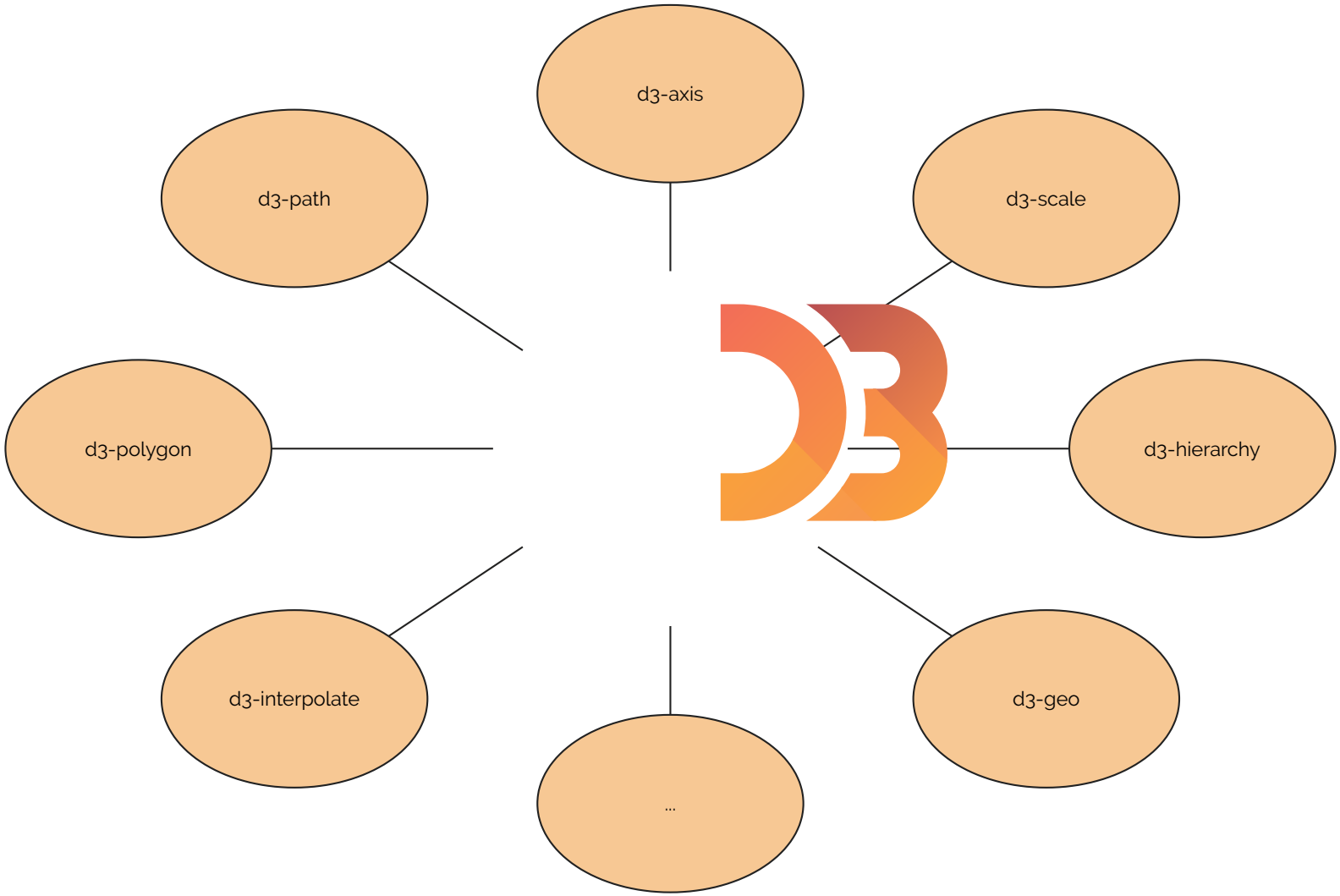
```
items
  .enter()
    .append('li')
    .attr('class', 'name')
  .merge(items)
    .text(name => name);
```

Remove unused elements

```
items
    .exit()
    .remove();
```

codesandbox.io/s/x2no0j1ryp

Ok, but... the charts?!


d3-axis
d3-path
d3-scale
d3-polygon
d3-hierarchy
d3-interpolate
...
d3-geo

Scales and axes

Types of scales

From the package `d3-scale`:

- `scaleLinear`
- `scalePow`, `scaleLog`
- `scaleTime`
- `scaleOrdinal`, `scaleBand`, `scalePoint`
- `scaleQuantize`, `scaleQuantile`, …

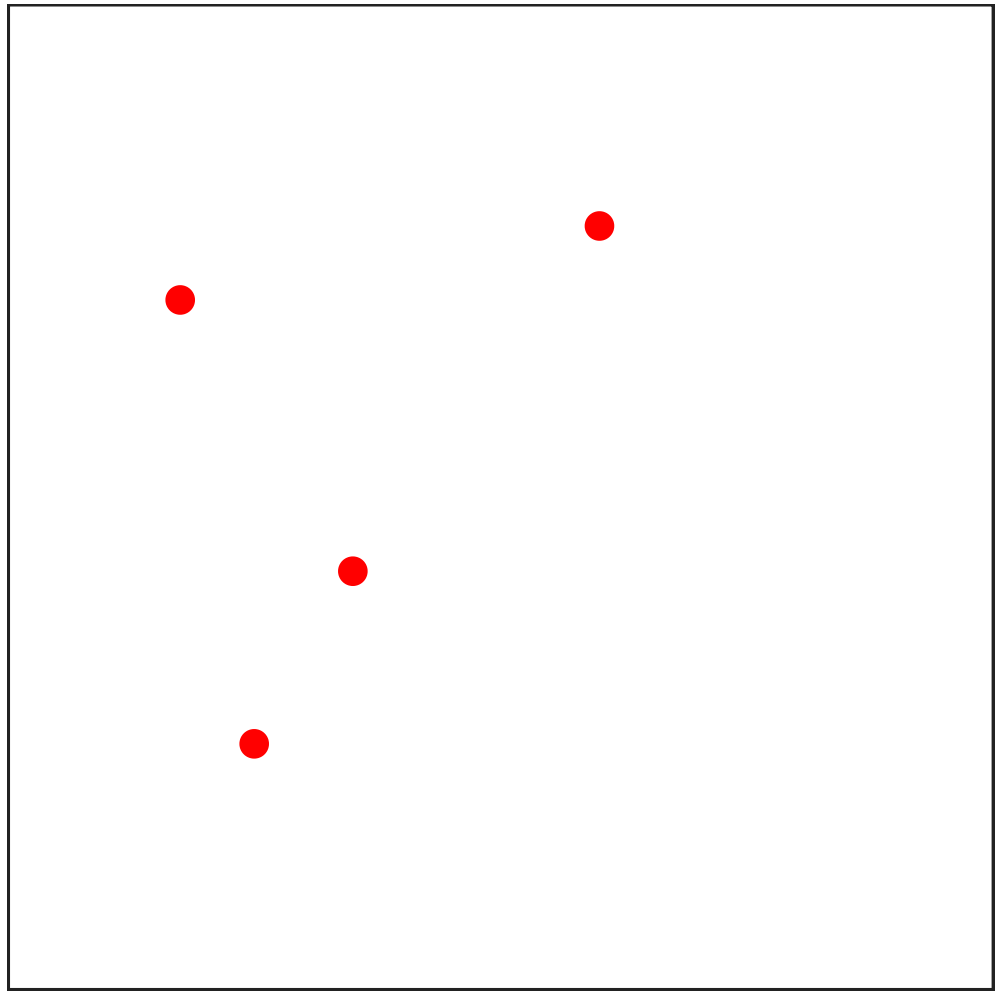
Scales are *functions*

They *all* need:

👉 a *domain*;
👉 a *range* ("codomain").

```
import { scaleLinear } from 'd3-scale';

const scale = scaleLinear()
  .domain([ 0, 100 ])
  .range([ 50, 450 ]);

scale(0);     // 50
scale(50);    // 250
scale(12.5);  // 100
scale(-100);  // -350
```

```
const points = [
  {x: 10, y: 30}, {x: 7, y: 12},
  {x: 24, y: 9}, {x: 14, y: 23}
];
const scaleX = scaleLinear()
  .domain([0, 40]).range([0, 200]);
const scaleY = scaleX.copy();

select('svg').selectAll('.point')
  .data(points).enter()
    .append('circle')
    .attr('class', 'point')
    .attr('cx', p => scaleX(p.x))
    .attr('cy', p => scaleY(p.y));
```



Axes

From the package `d3-axis`:

- `axisTop`
- `axisRight`
- `axisBottom`
- `axisLeft`

```
const axisY = axisLeft(scaleY);
```

```
<g fill="none" font-size="10" font-family="sans-serif" text-anchor="end">
  <path class="domain" stroke="currentColor" d="M-6,730.5H0.5V0.5H-6"></pat
  <g class="tick" opacity="1" transform="translate(0,730.5)">
    <line stroke="currentColor" x2="-6"></line>
    <text fill="currentColor" x="-9" dy="0.32em">0</text>
  </g>
  <g class="tick" opacity="1" transform="translate(0,669.667)">
    <line stroke="currentColor" x2="-6"></line>
    <text fill="currentColor" x="-9" dy="0.32em">5</text>
  </g>
  ...
</g>
```

Hot to draw a chart!

- 📈 retrive SVG sizes
- 📈 setup scales and axes
- 📈 create containers for the axes
- 📈 and for the chart
- 📈 get the data...
- 📈 ... and draw the chart!

Common charts

From the package `d3-shape`:

- 📈 line/area charts
- 📈 pie charts
- 📈 stacked bar charts
- 📈 link charts

Line charts

```
import { line } from 'd3-shape';
const data = [
  [ 19, 19, 24, 13, 20, 24, 14, 57, 16, 60 ],
  [ 13, 36, 14, 24, 31, 18, 26, 40, 10, 29 ],
  [ 43, 18, 15, 30, 21, 19, 65, 20, 18, 14 ]
];
const dataLine = line()
  .x((_, i) => scaleX(i))
  .y(d => scaleY(d));
const lines = select('g.chart')
  .selectAll('path').data(data);
lines.enter()
  .append('path')
  .attr('d', dataLine);
```

codesandbox.io/s/lyyrll692m

Pie charts

Disclaimer:

Don't make pie charts.

```
import { arc, pie } from 'd3-shape';
const data = [ 32, 140, 4, 13, 36, 6, 17, 24, 19, 37 ];
const pieData = pie()(data);
const sectorArc = arc()
  .innerRadius(80)
  .outerRadius(100)
  .padAngle(Math.PI / 360);
const sectors = select('g.chart')
  .selectAll('path').data(pieData);
sectors.enter()
  .append('path')
  .attr('d', sectorArc);
```

(But seriously, don't make pie charts.)

Stacked bar charts

Similarly to line charts, we have a data *matrix*:

```
const data = [
  [ 19, 19, 24, 13, 20, 24, 14, 57, 16, 60 ],
  [ 13, 36, 14, 24, 31, 18, 26, 40, 10, 29 ],
  [ 43, 18, 15, 30, 21, 19, 65, 20, 18, 14 ]
];
```

We can make a data join *from a data join*:

```
const join = chartRoot.selectAll('.group').data(data);
const group = join.enter()
  .append('g').attr('class', 'group')
  .merge(join);
join.exit().remove();

const subjoin = group.selectAll('.value').data(d => d);
subjoin.enter()
  .append('g').attr('class', 'value')
  .merge(subjoin)...
subjoin.exit().remove();
```

codesandbox.io/s/2znnw8q16n

(Just… don't abuse stacked charts either.)

Transitions

Disclaimer:

You *can* use CSS to animate your charts.

Even attributes like `x`, `r` or `<path>`'s `d`.

But

It works in

only 😕

`d3-transition` to the rescue!

```
import { transition } from 'd3-transition';

transition()
  .select('rect')
  .attr('fill', 'red')
  .style('stroke', 'black');
```

Transition of *what?*

- ⚡ Numbers: `0` → `100`
- ⚡ Colors: `#f60` → `rebeccapurple`
- ⚡ Lengths: `10px` → `120px`
- ⚡ Strings: `M0,20L10,5` → `M0,16L12,7`

Transition tuning

`.duration()`
`.delay()`
`.ease()` (see `d3-ease`)

Defining starting values

```
select('rect')
  .attr('fill', 'red');
.transition()
  .attr('fill', 'blue');
```

```
select('rect')
  .transition()
  .attrTween('fill', () =>
    interpolate('red', 'blue')
  );
```

See also `.styleTween()` and `.tween()`

codesandbox.io/s/z375662r0p

Chaining transitions

```
transition()
    .select('rect').attr('fill', 'red')
  .transition()
    .attr('fill', 'blue');
```

Transition events

```
select('rect')
  .transition()
  .attr('fill', 'red')
  .on('end', () => {
    console.log('All done!');
  });
```

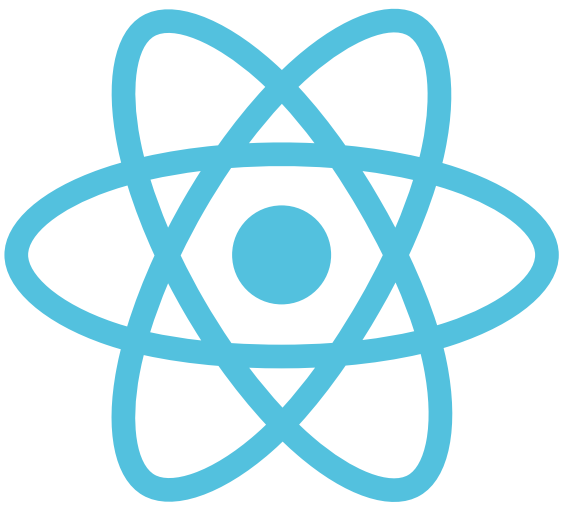
Other events: `start` and `interrupt`

codesandbox.io/s/m5qmlpqn88

D3 and modern frameworks

D3.js is *reactive*

… but so are these!



Solution #1: the black box

```
class BarChart extends React.Component {
  render() {
    return <svg ref={node => this.svgRoot = node}></svg>
  }
  drawChart() {
    const data = this.props.data;
    // Use D3.js here...
  }
  componentDidMount() { this.drawChart(); }
}
```

👍

☝ simple enough

✌ can reuse old code

👎

😕 feature-redundant

😕 "black box"

Solution #2: functional D3

```
class BarChart extends React.Component {
  constructor() {
    this.chartLine = line()...
  }
  render() {
    return <svg><g>
      {this.props.data.map(row =>
        <path d={this.chartLine(row)}/>
      )}
    </g></svg>
  }
}
```

👍

☝ better data flow

✌ more performant

👎

😕 unorthodox

😕 no axes, transitions

What's better?

¯_(ツ)_/¯

Server-side D3

☝ dynamic web charts
✌ batch generation
🤟 faster client side

D3 is a *DOM manipulation library*

How can it work on the server side?

☝ Functional D3 (again)
✌ *Tricked* into using a faux DOM

D3-Node

Two dependencies: `d3` and `jsdom`

```
const D3Node = require('d3-node');
const doc = new D3Node();
const root = doc.createSVG();

const scaleX = doc.d3.scaleLinear();
const lines = root.selectAll('path').data(data);
const output = doc.svgString(); // '<svg>...</svg>'
```

glitch.com/edit/#!/ssd3-pie-chart

That's all, folks!

maxart2501.github.io/d3-talk/devromagna/

```
for (const question of questions) {
  await answer(question)
}
```