Charts in React

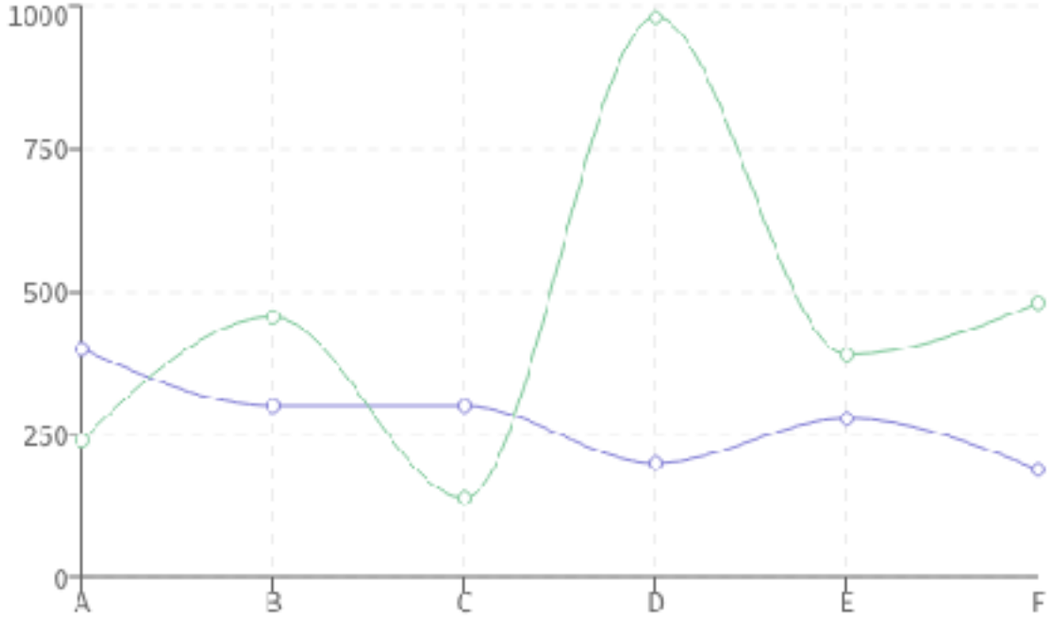
just use a library!

Recharts

A composable charting library built on React components

Install v1.0.0-beta.5

Star 7,320

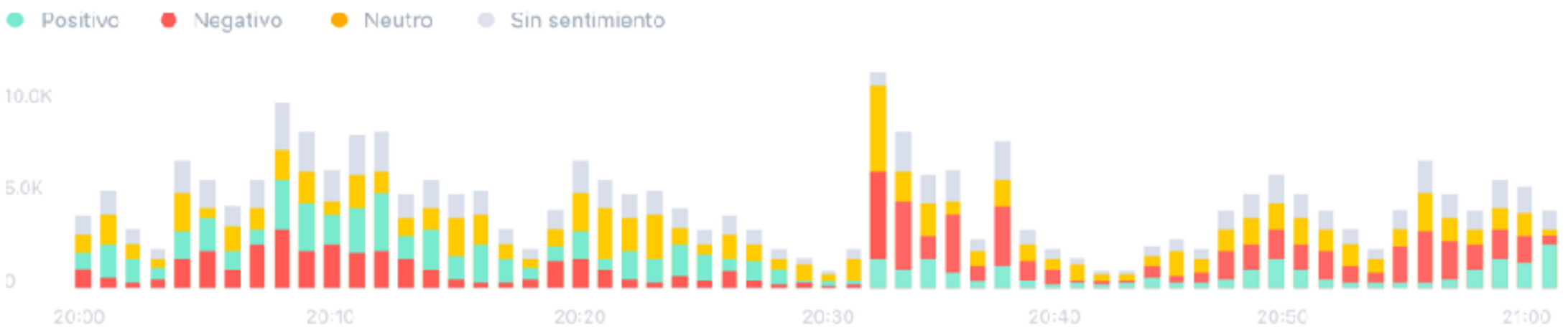



```
<LineChart width={500} height={300} data={data}>
  <XAxis dataKey="name"/>
  <YAxis/>
  <CartesianGrid stroke="#eee" strokeDasharray="5 5"/>
  <Line type="monotone" dataKey="uv" stroke="#8884d8" />
  <Line type="monotone" dataKey="pv" stroke="#82ca9d" />
</LineChart>
```

and maybe it's **good enough**
for you

but maybe you have a design

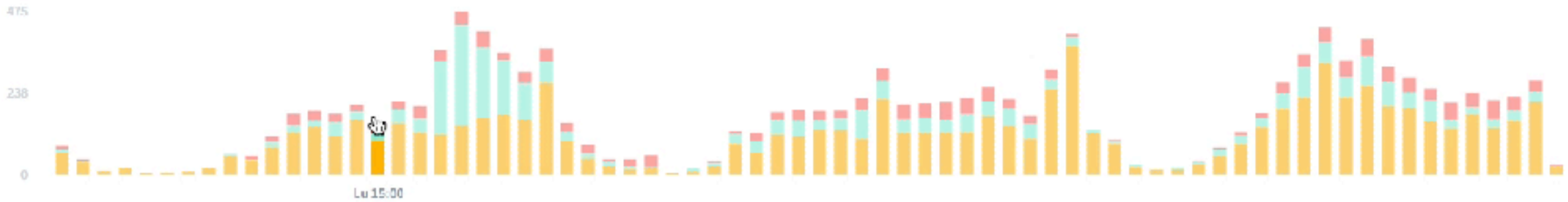
Evolución de la conversación



with complex interactions

Evolución de la conversación

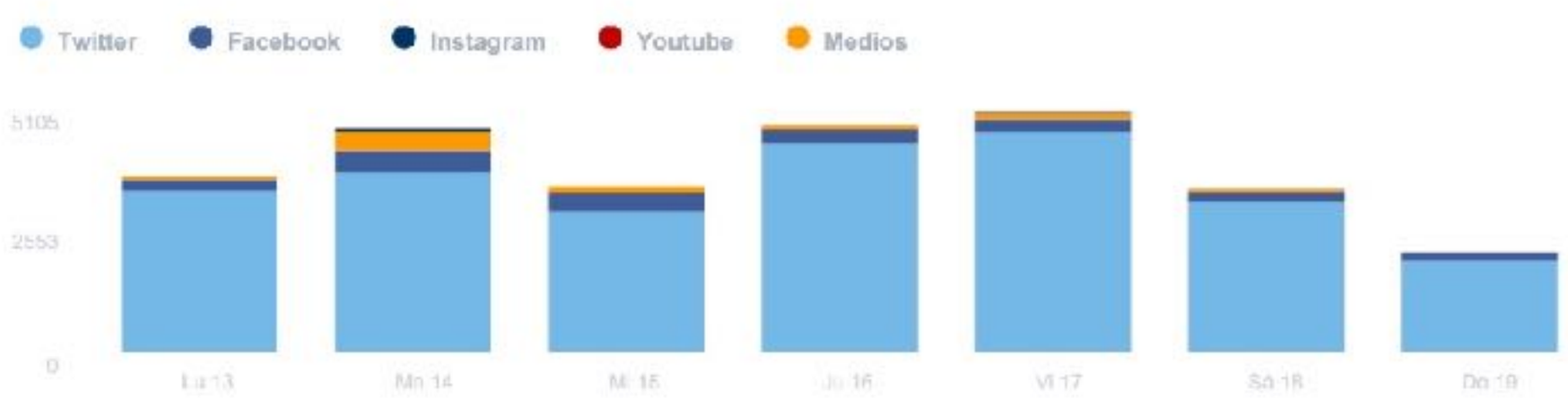
Positivo Negativo Neutro Sin sentimiento



and maybe it goes in a **PDF** too

Sentimiento

Conversación por fuente



or in an **email**

[Alerta Séntisis] [redacted]

Séntisis <info@sentisis.com> 11/4/16

para mí

That's built with tables

Alerta por Volumen

En la última hora se han producido más de **50 menciones negativas** sobre [redacted] en Twitter y Facebook.

Así ha evolucionado la conversación negativa sobre [redacted] **en las últimas 6 horas:**

Estas son las menciones con más impacto en la alerta:

so... **how?**

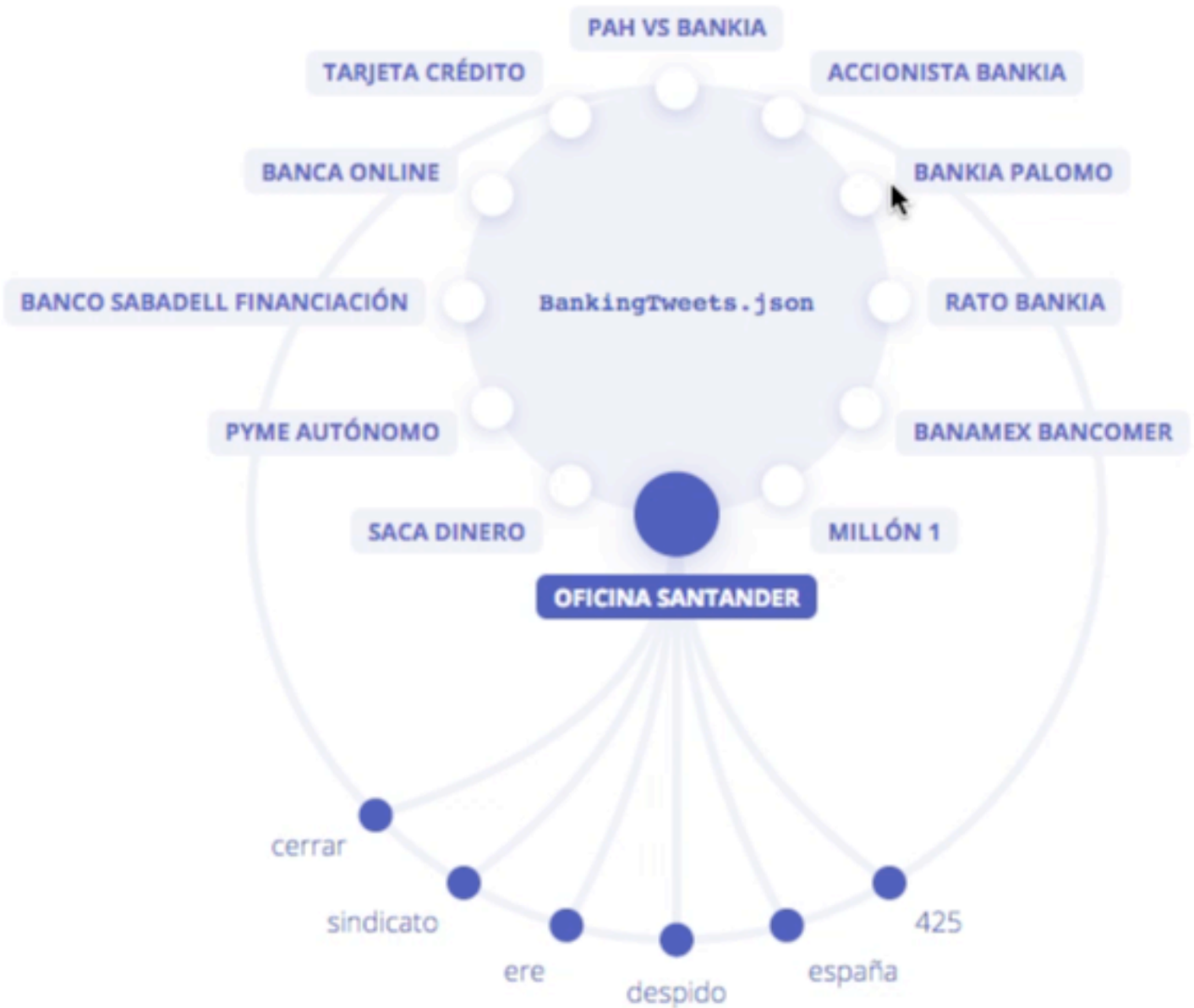
Alberto Restifo

Full Stack Developer

lang.ai

séntisis

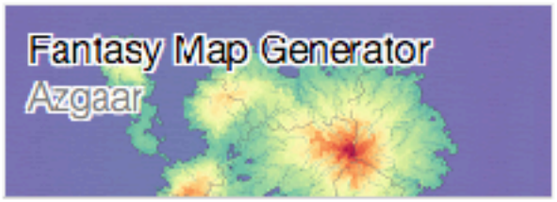
this year I built many charts

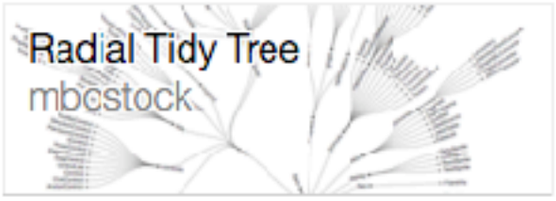

PAH VS BANKIA
TARJETA CRÉDITO
ACCIONISTA BANKIA
BANCA ONLINE
BANKIA PALOMO
BANCO SABADELL FINANCIACIÓN
BankingTweets.json
RATO BANKIA
PYME AUTÓNOMO
BANAMEX BANCOMER
SACA DINERO
MILLÓN 1
OFICINA SANTANDER
cerrar
sindicato
ere
despido
españa
425

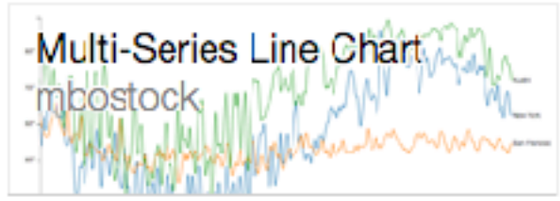

espera+tiempo
servicio+ao
dar_d+h_o
passar+és
plano+tarifa
els+rebuts
no_u+e_m
dia+teneis
si+tenia_n

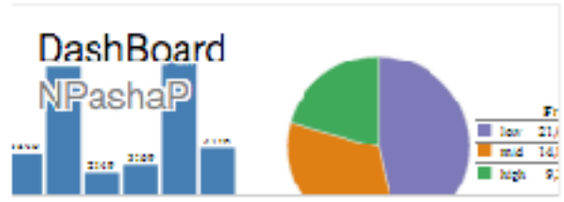
just use **refs**!

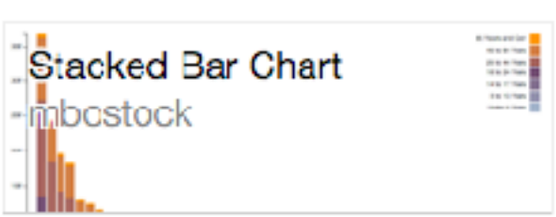
Popular Blocks
Updated November 30, 2017 1PM

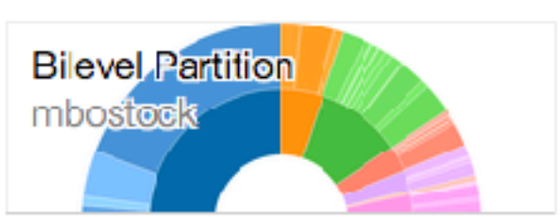


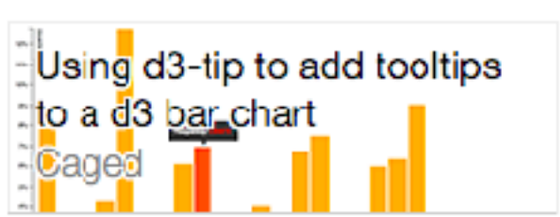


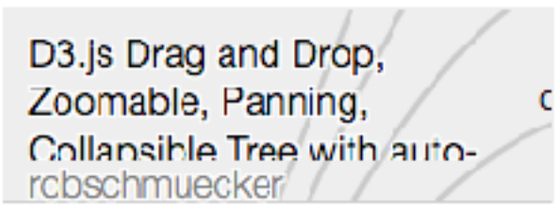


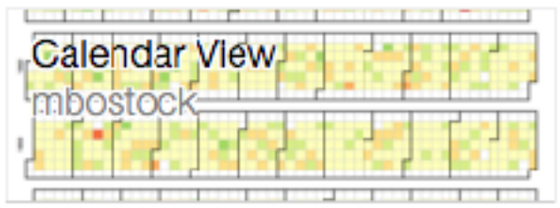


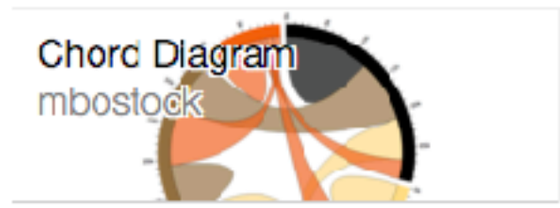


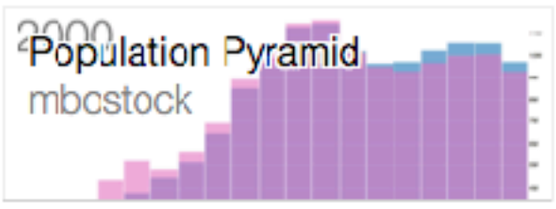


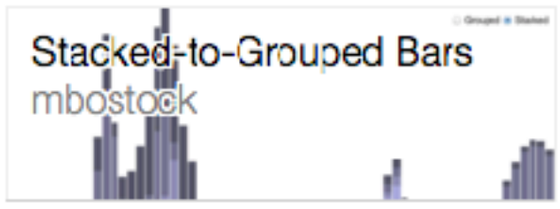


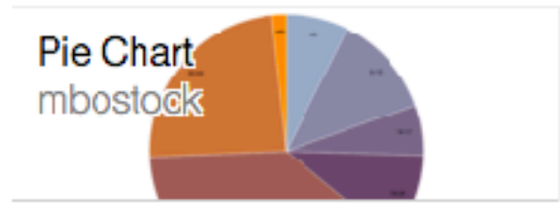


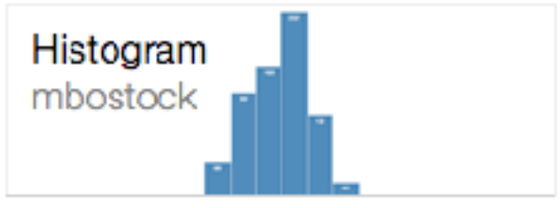


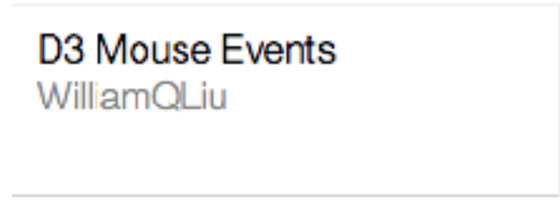


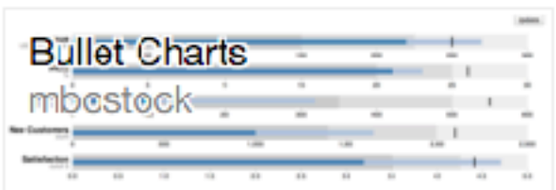


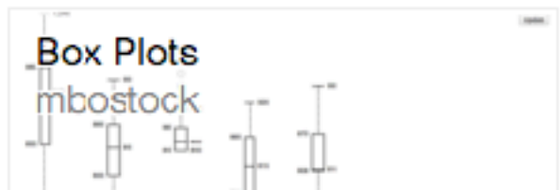




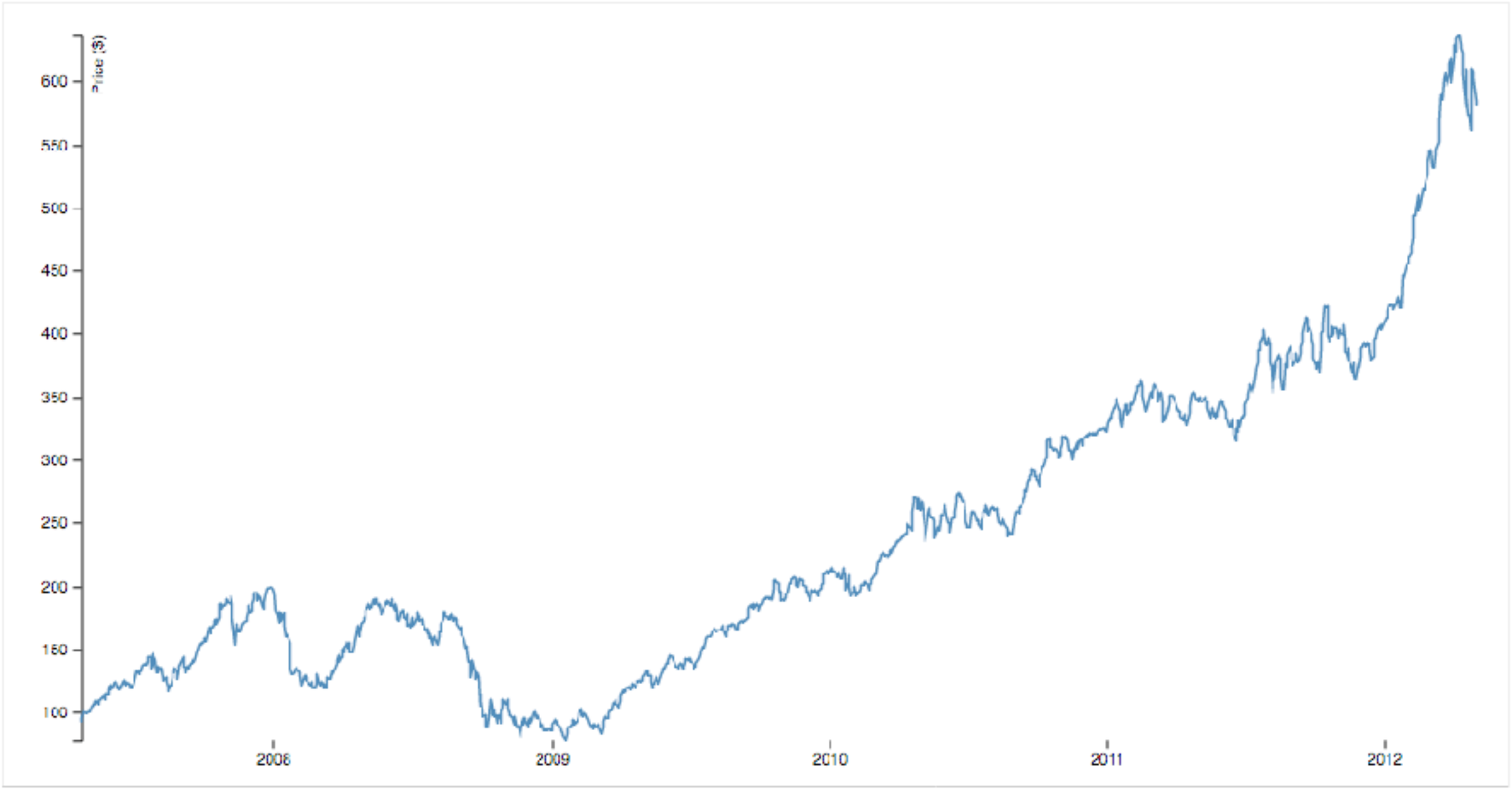








Line Chart



This simple line chart is constructed from a TSV file storing the closing value of AAPL stock over the last few years. The chart employs convertional margins and a number of D3 features:

Open

- d3-dsv - parse tab-separated values
- d3-time-format - date parsing and formatting
- d3-scale - position encodings
- d3-array - data processing

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

```
svgRef(el) {
  this.SvgEl = el;
}
```

```
componentDidMount() {
  const svg = d3.select(this.SvgEl)
  const g = svg.append('g')
  // ... more d3 logic
}
```

and it works!

Thank you.

EXAMPLE REPOSITORY:

goo.gl/H4L76g

what about **interactions**?

```
// componentDidMount, logic omitted
g.append('circle')
  .on('click', this.handleClick)
```

```
componentDidMount() {
  this.renderChart();
}
```

```
componentDidUpdate(prevProps) {
  if (
    this.props.data !== prevProps.data
  ) {
    this.renderChart();
  }
}
```

FOR THIS TO WORK:

- The `renderChart` function uses data from the props
- The props data is `immutable`

- The D3 logic inside renderChart can be called multiple times

I can deal with this.

what about **tooltips**?

```
const div = this.TooltipDiv;

g.selectAll('circle')
  // ...
 .on('mouseover', (d) => {
   div.html(`
     <span class="date">${formatTime(d.date)}</span>
     <span class="value">${d.value}</span>
   `)
   .style('opacity', 1)
   .style('top', `${event.pageX}px`)
   .style('left') `${event.pageY}px`)
})
.on('mouseout', (d) => {
  div.style('opacity', 0);
});
```

```
const div = this.TooltipDiv;

g.selectAll('circle')
  // ...
 .on('mouseover', (d) => {
   div.html(`
     <span class="date">${formatTime(d.date)}</span>
     <span class="value">${d.value}</span>
   `)
   .style('opacity', 1)
   .style('top', `${event.pageX}px`)
   .style('left') `${event.pageY}px`)
})
.on('mouseout', (d) => {
  div.style('opacity', 0);
});
```

```
const div = this.TooltipDiv;

g.selectAll('circle')
  // ...
 .on('mouseover', (d) => {
    div.html(`
      <span class="date">${formatTime(d.date)}</span>
      <span class="value">${d.value}</span>
    `)
    .style('opacity', 1)
    .style('top', `${event.pageX}px`)
    .style('left') `${event.pageY}px`)
})
.on('mouseout', (d) => {
  div.style('opacity', 0);
});
```

```
const div = this.TooltipDiv;

g.selectAll('circle')
  // ...
 .on('mouseover', (d) => {
   div.html(`
     <span class="date">${formatTime(d.date)}</span>
     <span class="value">${d.value}</span>
   `)
   .style('opacity', 1)
   .style('top', `${event.pageX}px`)
   .style('left') `${event.pageY}px`)
})
.on('mouseout', (d) => {
  div.style('opacity', 0);
});
```

```
const div = this.TooltipDiv;

g.selectAll('circle')
  // ...
 .on('mouseover', (d) => {
   div.html(`
     <span class="date">${formatTime(d.date)}</span>
     <span class="value">${d.value}</span>
   `)
   .style('opacity', 1)
   .style('top', `${event.pageX}px`)
   .style('left') `${event.pageY}px`)
})
.on('mouseout', (d) => {
  div.style('opacity', 0);
});
```

and it works!

```
const div = this.TooltipDiv;

g.selectAll('circle')
  // ...
 .on('mouseover', (d) => {
   div.html(`
     <span class="date">${formatTime(d.date)}</span>
     <span class="value">${d.value}</span>
   `)
   .style('opacity', 1)
   .style('top', `${event.pageX}px`)
   .style('left') `${event.pageY}px`)
})
.on('mouseout', (d) => {
  div.style('opacity', 0);
});
```

Charts in React

~~The Fast Way~~

The Right Way

what is **the right way**?

use the **math functions** of d3

to compute the chart

use React to **draw** the chart

let's see how it works

basic **svg** concepts

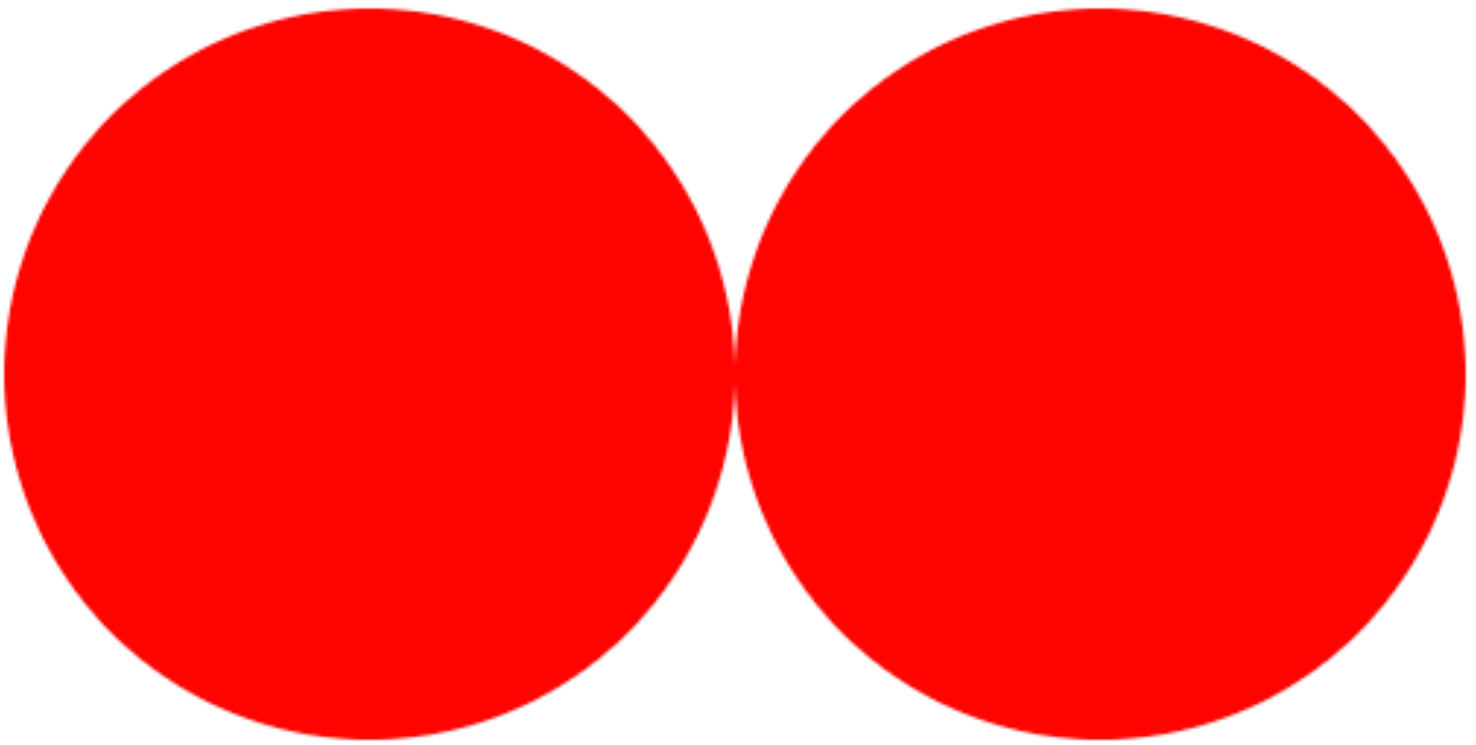

<svg width="500px" height="500px">
x=0
x=500
y=0
x=250, y=250
y=500

500 !== 500px

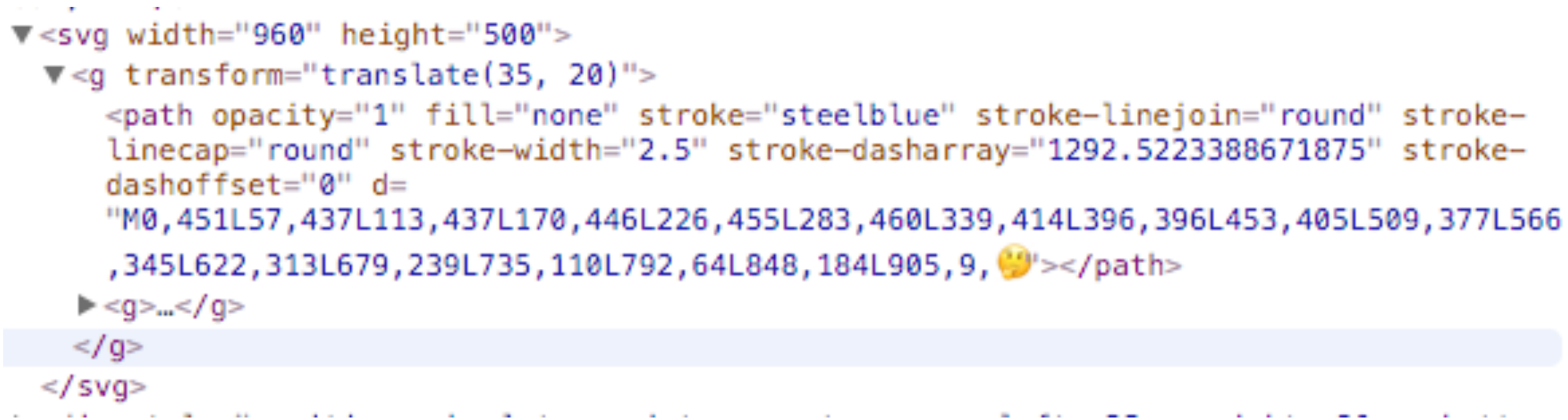
<svg width="250px" height="250px"
viewBox="0 0 500 500">

the <g> element

```
<svg width="500px" height="500px">
  <g fill="red">
    <circle cx="250" cy="125" r="125" />
    <circle cx="250" cy="375" r="125" />
  </g>
</svg>
```



and don't worry about the rest


▼<svg width="960" height="500">
▼<g transform="translate(35, 20)">
<path opacity="1" fill="none" stroke="steelblue" stroke-linejoin="round" stroke-linecap="round" stroke-width="2.5" stroke-dasharray="1292.5223388671875" stroke-dashoffset="0" d=
"M0,451L57,437L113,437L170,446L226,455L283,460L339,414L396,396L453,405L509,377L566,345L622,313L679,239L735,110L792,64L848,184L905,9,🤔"></path>
▶<g>…</g>
</g>
</svg>

create the **svg**

```
const svg = d3.select('svg');

const g = svg.append('g')
  .attr('transform',
    `translate(${margin.left}, ${margin.top})`);
```

```
render() {
  const { width, height, margin } = this.props;

  return (
    <svg width={width} height={height}>
      <g transform={`translate(...)`}>
      </g>
    </svg>
  );
}
```

```
// Create the drawing function
const line = d3.line()
  .x(d => x(d.date))
  .y(d => y(d.value));

// Append the path to the group
g.append('path')
  .datum(data)
  .attr('fill', 'none')
  .attr('stroke', 'steelblue')
  .attr('stroke-linejoin', 'round')
  .attr('stroke-linecap', 'round')
  .attr('stroke-width', 2.5)
  .attr('d', line);
```

D3-SHAPE

d3.line()

Constructs a new line generator with the default settings

line.x([x])

If x is specified, sets the x accessor to the specified function or number and returns this line generator.

line.y([y])

If y is specified, sets the y accessor to the specified function or number and returns this line generator.

D3-SHAPE

line(**data**)

Generates a line for the given array of data.

```
// Create the drawing function
const line = d3.line()
  .x(d => x(d.date))
  .y(d => y(d.value));

// Append the path to the group
g.append('path')
  .datum(data)
  .attr('fill', 'none')
  .attr('stroke', 'steelblue')
  .attr('stroke-linejoin', 'round')
  .attr('stroke-linecap', 'round')
  .attr('stroke-width', 2.5)
  .attr('d', line);
```

```
// Create the drawing function
const line = d3.line()
    .x(d => x(d.date))
    .y(d => y(d.value));

// Append the path to the group
g.append('path')
    .datum(data)
    .attr('fill', 'none')
    .attr('stroke', 'steelblue')
    .attr('stroke-linejoin', 'round')
    .attr('stroke-linecap', 'round')
    .attr('stroke-width', 2.5)
    .attr('d', line);
```

```
// Create the drawing function
const line = d3.line()
    .x(d => x(d.date))
    .y(d => y(d.value));

// Append the path to the group
g.append('path')
    .datum(data)
    .attr('fill', 'none')
    .attr('stroke', 'steelblue')
    .attr('stroke-linejoin', 'round')
    .attr('stroke-linecap', 'round')
    .attr('stroke-width', 2.5)
    .attr('d', line);
```

```
// Create the drawing function
const line = d3.line()
    .x(d => x(d.date))
    .y(d => y(d.value));

// Append the path to the group
g.append('path')
    .datum(data)
    .attr('fill', 'none')
    .attr('stroke', 'steelblue')
    .attr('stroke-linejoin', 'round')
    .attr('stroke-linecap', 'round')
    .attr('stroke-width', 2.5)
    .attr('d', line);
```

```
// ... define x and y scaling functions

const line = d3.line()
  .x(d => x(d.date))
  .y(d => y(d.value));

return (
  <svg width={width} height={height}>
    <g transform={`translate(...)`}>
      <path
        d={line(data)}
        fill="none"
        stroke="steelblue"
        strokeLinejoin="round"
        strokeLinecap="round"
        strokeWidth="2.5"
      />
    </g>
  </svg>
);
```

```
// ... define x and y scaling functions

const line = d3.line()
  .x(d => x(d.date))
  .y(d => y(d.value));

return (
  <svg width={width} height={height}>
    <g transform={`translate(...)`}>
      <path
        d={line(data)}
        fill="none"
        stroke="steelblue"
        strokeLinejoin="round"
        strokeLinecap="round"
        strokeWidth="2.5"
      />
    </g>
  </svg>
);
```

```
// ... define x and y scaling functions

const line = d3.line()
  .x(d => x(d.date))
  .y(d => y(d.value));

return (
  <svg width={width} height={height}>
    <g transform={`translate(...)`}>
      <path
        d={line(data)}
        fill="none"
        stroke="steelblue"
        strokeLinejoin="round"
        strokeLinecap="round"
        strokeWidth="2.5"
      />
    </g>
  </svg>
);
```

```
// ... define x and y scaling functions

const line = d3.line()
  .x(d => x(d.date))
  .y(d => y(d.value));

return (
  <svg width={width} height={height}>
    <g transform={`translate(...)`}>
      <path
        d={line(data)}
        fill="none"
        stroke="steelblue"
        strokeLinejoin="round"
        strokeLinecap="round"
        strokeWidth="2.5"
      />
    </g>
  </svg>
);
```

and **it works**!

```
// ... define x and y scaling functions

const line = d3.line()
  .x(d => x(d.date))
  .y(d => y(d.value));

return (
  <svg width={width} height={height}>
    <g transform={`translate(...)`}>
      <path
        d={line(data) + ',🤔'}
        fill="none"
        stroke="steelblue"
        strokeLinejoin="round"
        strokeLinecap="round"
        strokeWidth="2.5"
      />
    </g>
  </svg>
);
```

but there is a **problem**

```
// ... define x and y scaling functions

const line = d3.line()
  .x(d => x(d.date))
  .y(d => y(d.value));

return (
  <svg width={width} height={height}>
    <g transform={`translate(...)`}>
      <path
        d={line(data) + ',🤔'}
        fill="none"
        stroke="steelblue"
        strokeLinejoin="round"
        strokeLinecap="round"
        strokeWidth="2.5"
      />
    </g>
  </svg>
);
```

```
// ... define x and y scaling functions

const line = d3.line()
  .x(d => x(d.date))
  .y(d => y(d.value));

return (
  <svg width={width} height={height}>
    <g transform={`translate(...)`}>
      <path
        d={line(data) + ',🤔'}
        fill="none"
        stroke="steelblue"
        strokeLinejoin="round"
        strokeLinecap="round"
        strokeWidth="2.5"
      />
    </g>
  </svg>
);
```

save the data in the **state**

```
setData(props = this.props) {
  const { width, height, margin, data } = props;

  const x = scaleTime()
    .rangeRound([0, width - margin.left - margin.right])
    .domain(extent(data, d => d.date));

  const y = scaleLinear()
    .rangeRound([height - margin.top - margin.bottom, 0])
    .domain(extent(data, d => d.value));

  const graphData = data.map((d) => ({
    ...d,
    x: x(d.date),
    y: y(d.value),
  }));

  this.setState({ data: graphData });
}
```

```
setData(props = this.props) {
  const { width, height, margin, data } = props;

  const x = scaleTime()
    .rangeRound([0, width - margin.left - margin.right])
    .domain(extent(data, d => d.date));

  const y = scaleLinear()
    .rangeRound([height - margin.top - margin.bottom, 0])
    .domain(extent(data, d => d.value));

  const graphData = data.map((d) => ({
    ...d,
    x: x(d.date),
    y: y(d.value),
  }));

  this.setState({ data: graphData });
}
```

```
  setData(props = this.props) {
    const { width, height, margin, data } = props;

    const x = scaleTime()
      .rangeRound([0, width - margin.left - margin.right])
      .domain(extent(data, d => d.date));

    const y = scaleLinear()
      .rangeRound([height - margin.top - margin.bottom, 0])
      .domain(extent(data, d => d.value));

    const graphData = data.map((d) => ({
      ...d,
      x: x(d.date),
      y: y(d.value),
    }));

    this.setState({ data: graphData });
  }
```

```
setData(props = this.props) {
  const { width, height, margin, data } = props;

  const x = scaleTime()
    .rangeRound([0, width - margin.left - margin.right])
    .domain(extent(data, d => d.date));

  const y = scaleLinear()
    .rangeRound([height - margin.top - margin.bottom, 0])
    .domain(extent(data, d => d.value));

  const graphData = data.map((d) => ({
    ...d,
    x: x(d.date),
    y: y(d.value),
  }));

  this.setState({ data: graphData });
}
```

```
componentWillMount() {
  this.setData();
}

componentWillReceiveProps(nexProps) {
  if (
    this.props.data !== nextProps.data
  ) {
    this.setData(nextProps);
  }
}
```

```
 const line = d3.line()
-  .x(d => x(d.x))
-  .y(d => y(d.value));
+  .x(d => d.x)
+  .y(d => d.y);

 return (
   <svg width={width} height={height}>
     <g transform={`translate(...)`}>
       <path
-        d={line(data)}
+        d={line(this.state.data)}
         fill="none"
         stroke="steelblue"
         strokeLinejoin="round"
```

WHY IT'S AWESOME:

- No DOM required (server-rendering)
- React blazing-fast reconciliation algorithm
- Fast renders

let's add the **points**

```
g.selectAll('point')
  .data(data)
  .enter().append('circle')
    .attr('r', 4)
  .attr('cx', d => x(d.date))
  .attr('cy', d => y(d.value))
  .attr('fill', 'white')
  .attr('stroke', 'steelblue')
  .attr('stroke-width', 2.5);
```

```jsx
const Point = ({ x, y }) => (
  <circle
    cx={x}
    cy={y}
    r="4"
    fill="white"
    stroke="steelBlue"
    strokeWidth="2.5"
  />
);
```

```
const { data } = this.state;

return (
  <svg width={width} height={height}>
    <g transform={`translate(...)`}>
      <path d={line(data)} />
      {data.map((d, i) => (
        <Point key={i} x={d.x} y={d.y} />
      ))}

    </g>
  </svg>
);
```

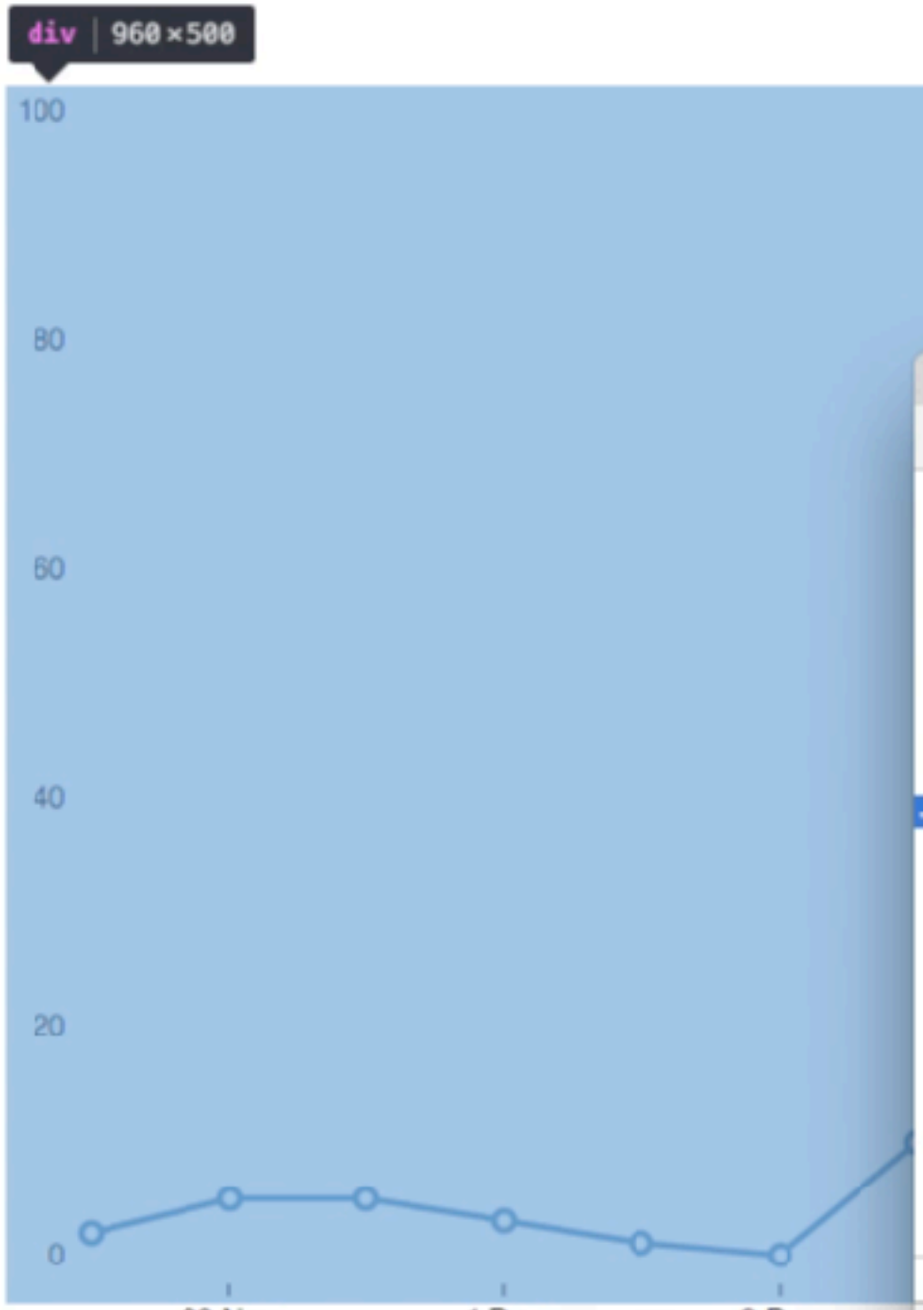
```
const { data } = this.state;

return (
  <svg width={width} height={height}>
    <g transform={`translate(...)`}>
      <path d={line(data)} />
      {data.map((d, i) => (
        <Point key={i} x={d.x} y={d.y} />
      ))}

    </g>
  </svg>
);
```

what about the **tooltip**?

```
import format from 'date-fns/format';

const Tooltip = ({ hidden, x, y, date, value }) => (
  <div
    style={{
      opacity: hidden ? 0 : 1
      transform: `translate(${x}px, ${y - 35}px)`,
    }}
  >
    <span className="date">
      {format(date, 'MMM DD')}
    </span>
    <span className="value">
      {value}
    </span>
  </div>
);
```

```
const { isTooltipVisible, tooltipDate } = this.state;

return (
  <div
    style={{
      width: `${width}px`,
      height: `${height}px`,
      position: 'relative',
    }}
  >
    <Tooltip
      hidden={!isTooltipVisible}
      {...tooltipData}
    />

    <svg>
      {/* ... */}
    </svg>
  </div>
);
```

```
const { isTooltipVisible, tooltipDate } = this.state;

return (
  <div
    style={{
      width: `${width}px`,
      height: `${height}px`,
      position: 'relative',
    }}
  >
    <Tooltip
      hidden={!isTooltipVisible}
      {...tooltipData}
    />

    <svg>
      {/* ... */}
    </svg>
  </div>
);
```



Developer Tools - http://localhost:3000/

Elements Console Sources Network Performance Memory Application »

```
<!DOCTYPE html>
<html lang="en">
▶<head>…</head>
▼<body>
  <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
  ▼<div id="root">
    ▼<div class="App">
      ▼<div style="width: 960px; height: 500px; position: relative;"> == $0
        ▶<div class="LineChart-tooltip" style="opacity: 0; transition: opacity 250ms ease-in-out;">…</div>
        ▼<svg width="960" height="500">
          ▶<g transform="translate(35, 20)">…</g>
          </svg>
        ▶<div style="position: absolute; pointer-events: none; left: 35px; right: 20px; bottom: 0px;">…</div>
        ▶<div style="width: 25px; position: absolute; pointer-events: none; top: 20px; left: 0px; bottom: 20px;">…</div>
        </div>
        <button>Start animation</button>
      </div>
```

html body div#root div.App div svg

Styles Event Listeners DOM Breakpoints Properties Accessibility Properties

Filter :hov .cls +

```
element.style {
  width: 960px;
  height: 500px;
  position: relative;
}
div {                                   user agent stylesheet
  display: block;
}
```

position 0
margin -
border -
padding -
960 × 500
0 0

```
const { isTooltipVisible, tooltipDate } = this.state;

return (
  <div
    style={{
      width: `${width}px`,
      height: `${height}px`,
      position: 'relative',
    }}
  >
    <Tooltip
      hidden={!isTooltipVisible}
      {...tooltipData}
    />

    <svg>
      {/* ... */}
    </svg>
  </div>
);
```

the **axis**

this time we're on our own

D3-ARRAY

d3.extent(*array[, accessor]***)**

Returns the minimum and maximum value in the given array using natural order.

d3.ticks(*start, stop, count***)**

Returns an array of approximately count + 1 uniformly-spaced, nicely-rounded values between start and stop (inclusive).

```
import { extent, ticks } from 'd3-array';

const [min, max] = extent(data, d => d.date);

const values = ticks(min, max, 5);
```

let's see an example

```
const { width, height, margin } = this.props;
const { data } = this.state;

return (
  <div
    style={{
      width: `${width}px`,
      height: `${height}px`,
      position: 'relative',
    }}
  >
    {/* Tooltip and chart omitted */}

    <XAxis data={data} width={width} margin={margin} />
    <YAxis data={data} height={height} margin={margin} />
  </div>
);
```

```
const { width, height, margin } = this.props;
const { data } = this.state;

return (
  <div
    style={{
      width: `${width}px`,
      height: `${height}px`,
      position: 'relative',
    }}
  >
    {/* Tooltip and chart omitted */}

    <XAxis data={data} width={width} margin={margin} />
    <YAxis data={data} height={height} margin={margin} />
  </div>
);
```

and our line chart is done!

EVEN MORE AWESOME:

- Small, isolated components
- Fully customizable
- No magic involved, it all simple React code

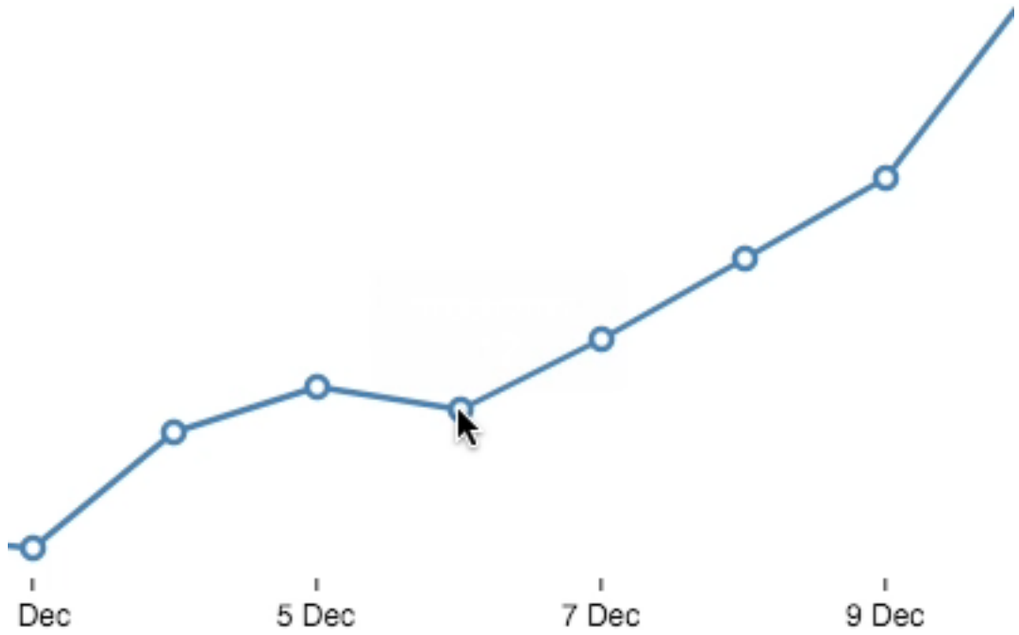
time to **animate** it

css traforms

simple is better.

```jsx
const Tooltip = ({ hidden, x, y, date, value }) => (
  <div
    style={{
      opacity: hidden ? 0 : 1,
      transition: 'opacity 250ms ease-in-out',
      transform: `translate(${x}px, ${y - 35}px)`,
    }}
  >
    <span className="date">
      {formatDate(date, 'D MMM YYYY')}
    </span>
    <span className="value">{value}</span>
  </div>
);
```

```
const Tooltip = ({ hidden, x, y, date, value }) => (
  <div
    style={{
      opacity: hidden ? 0 : 1,
      transition: 'opacity 250ms ease-in-out',
      transform: `translate(${x}px, ${y - 35}px)`,
    }}
  >
    <span className="date">
      {formatDate(date, 'D MMM YYYY')}
    </span>
    <span className="value">{value}</span>
  </div>
);
```


Dec
5 Dec
7 Dec
9 Dec

what about **complex**
animations?

d3 got your back

D3-TIMER

d3.timer(*callback[, delay[, time]***)**

Schedules a new timer, invoking the specified callback repeatedly until the timer is stopped.

timer.**stop()**

Stops this timer, preventing subsequent callbacks

```
const DURATION = 500;

const t = timer((elapsed) => {
  // Do something

  if (elapsed > DURATION) return t.stop();

}, 100);
```

```
const DURATION = 500;

const t = timer((elapsed) => {
  // Do something

  if (elapsed > DURATION) return t.stop();

}, 100);
```

```
const DURATION = 500;

const t = timer((elapsed) => {
  // Do something

  if (elapsed > DURATION) return t.stop();

}, 100);
```

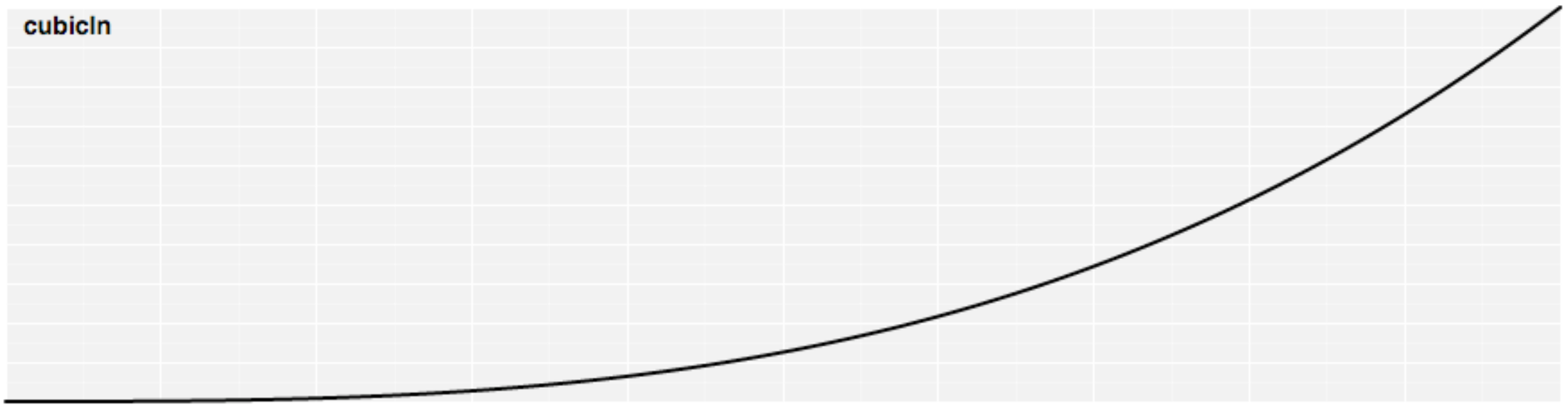
D3-EASE

ease(*t*)

Given the specified normalized time *t*, typically in the range [0,1], returns the “eased” time *t*, also typically in [0,1]

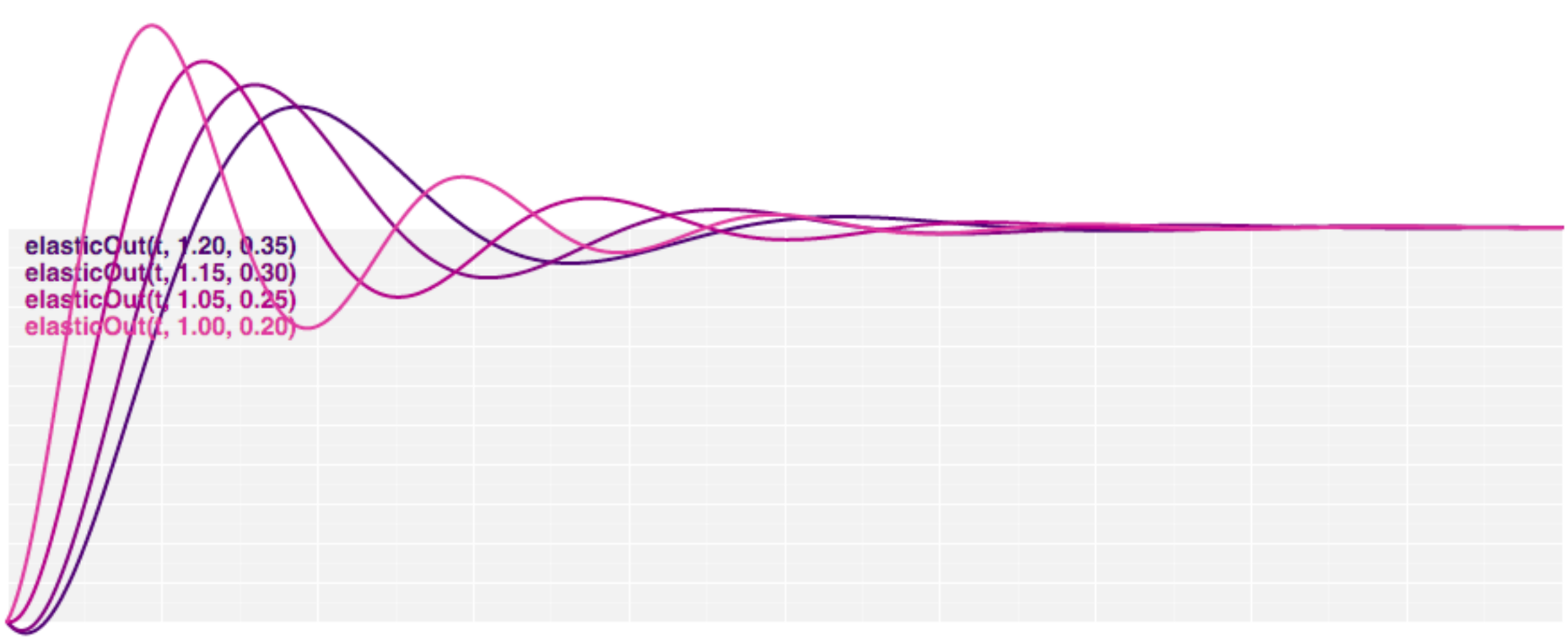
D3-EASE

`d3.easeCubicIn(`*t*`)`



D3-EASE

d3.easeElastic(*t[, amplitude[, period]]***)**



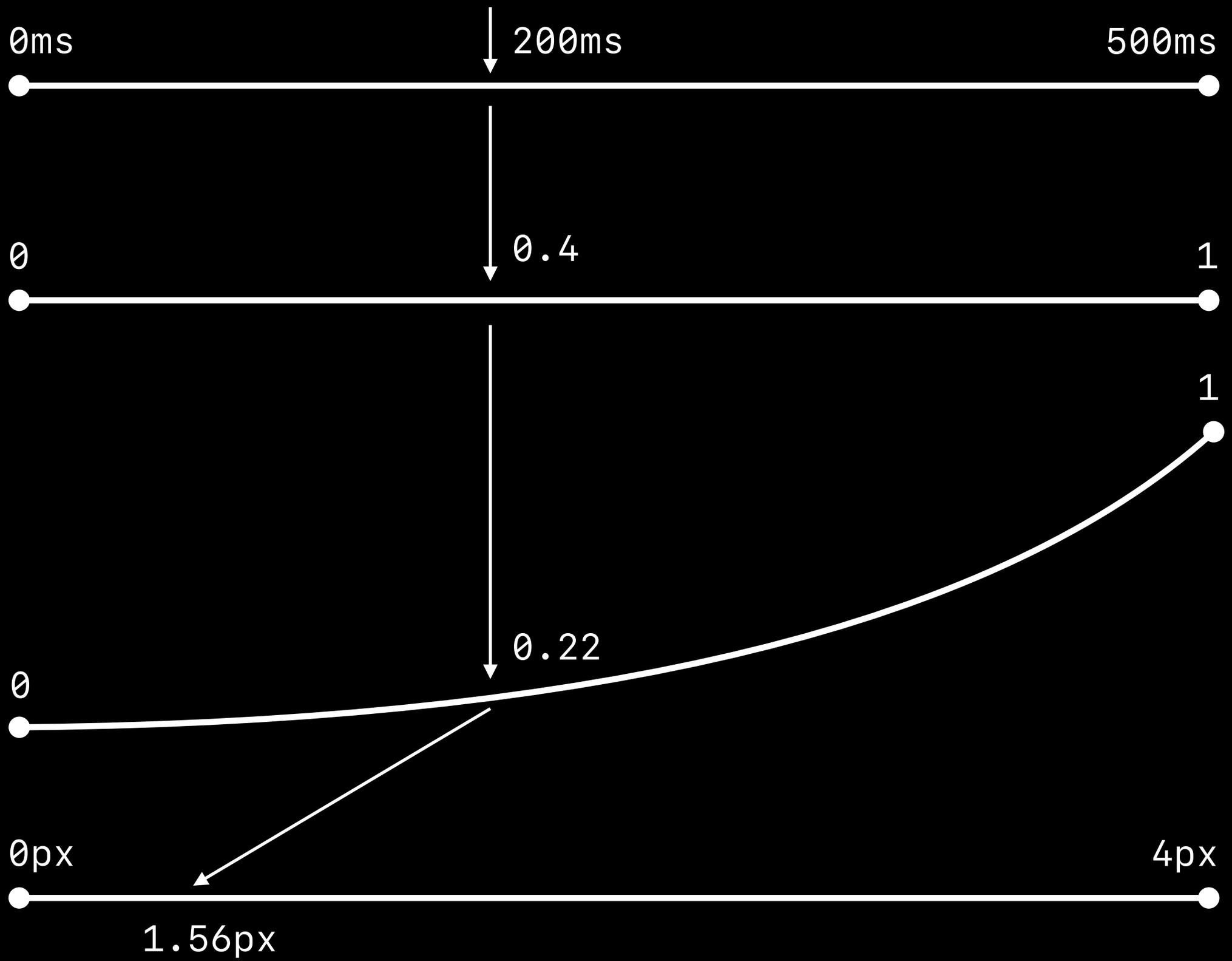
D3-INTERPOLATE

d3.interpolateNumber(*a*, *b*)

Returns an interpolator between the two numbers *a* and *b*.

A magical mathematical device that when passed a number between 0 and 1, returns the corresponding number between a and b

wat

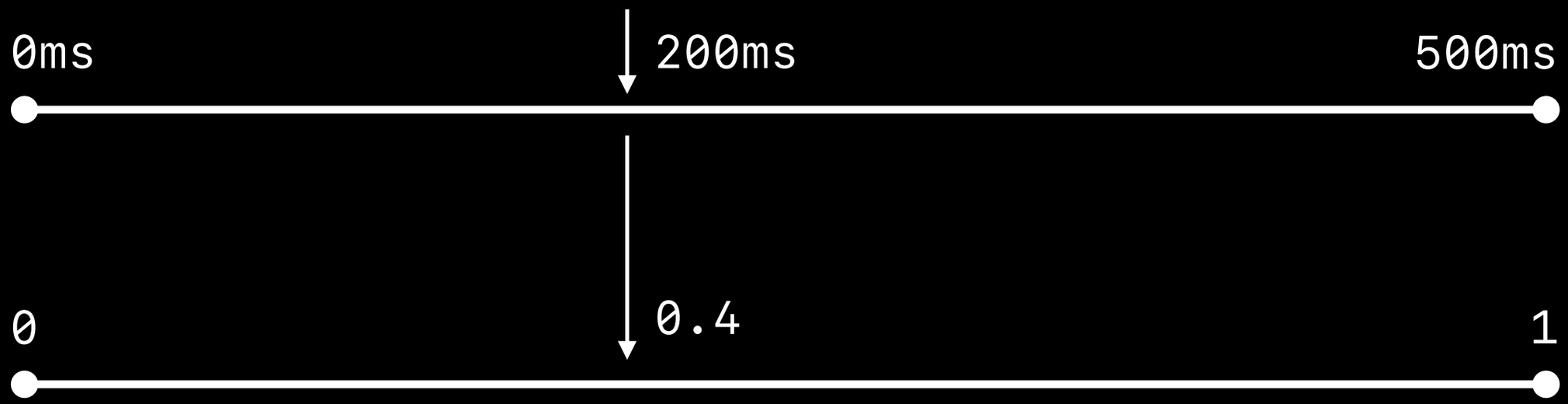

0ms
200ms
500ms
0
0.4
1
1
0
0.22
0px
4px
1.56px

```
const DURATION = 1000;

const timeScale = scaleLinear()
  .domain([0, DURATION])
  .clamp(true);

const i = interpolateNumber(0, 4);

const t = timer((elapsed) => {
  const te = easeElasticOut(timeScale(elapsed), 4, 0.5);

  const radius = i(te);
  this.setState({ radius });

  if (elapsed > DURATION) return t.stop();
});
```

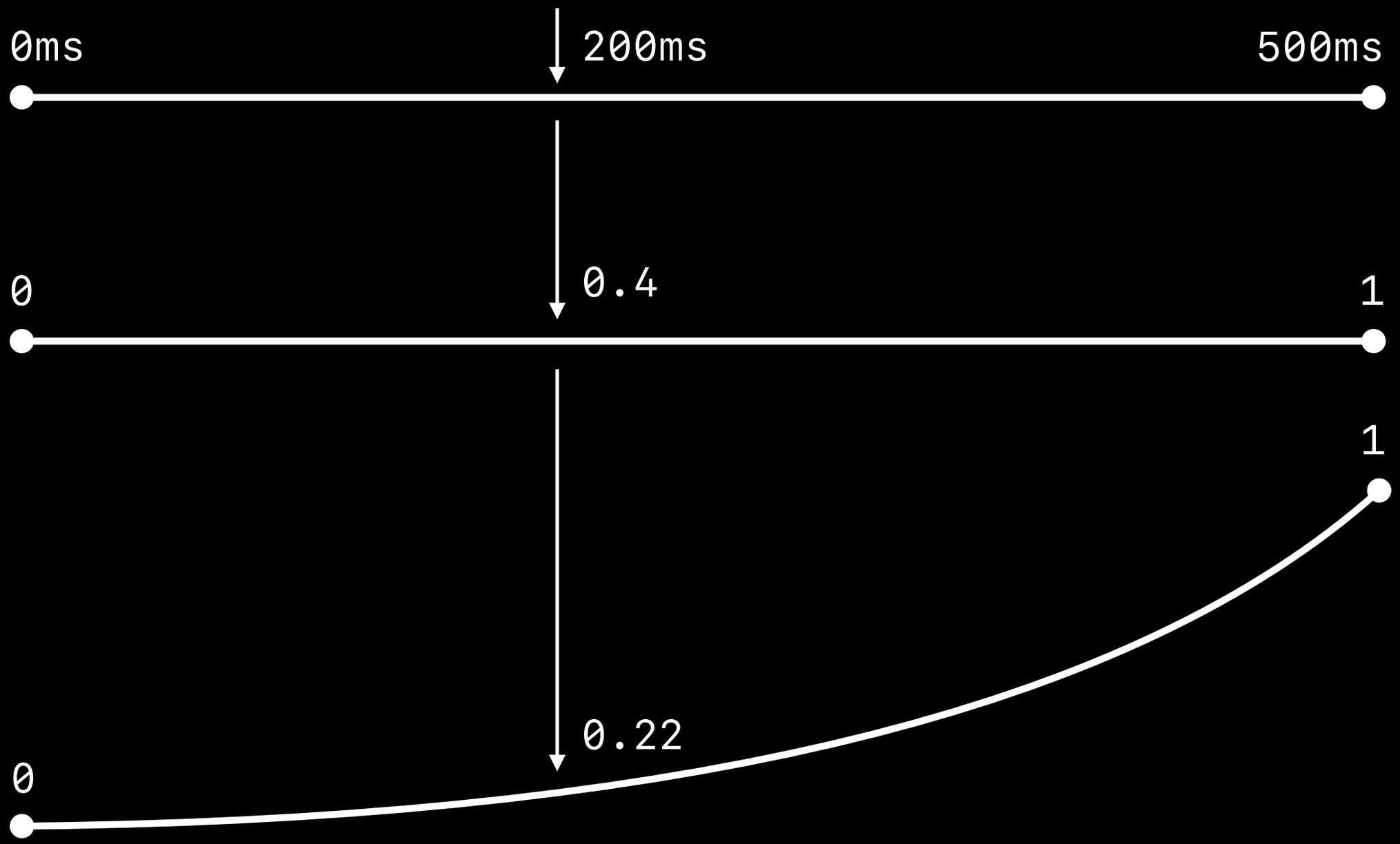
```
const DURATION = 1000;

const timeScale = scaleLinear()
  .domain([0, DURATION])
  .clamp(true);

const i = interpolateNumber(0, 4);

const t = timer((elapsed) => {
  const te = easeElasticOut(timeScale(elapsed), 4, 0.5);

  const radius = i(te);
  this.setState({ radius });

  if (elapsed > DURATION) return t.stop();
});
```


0ms
200ms
500ms

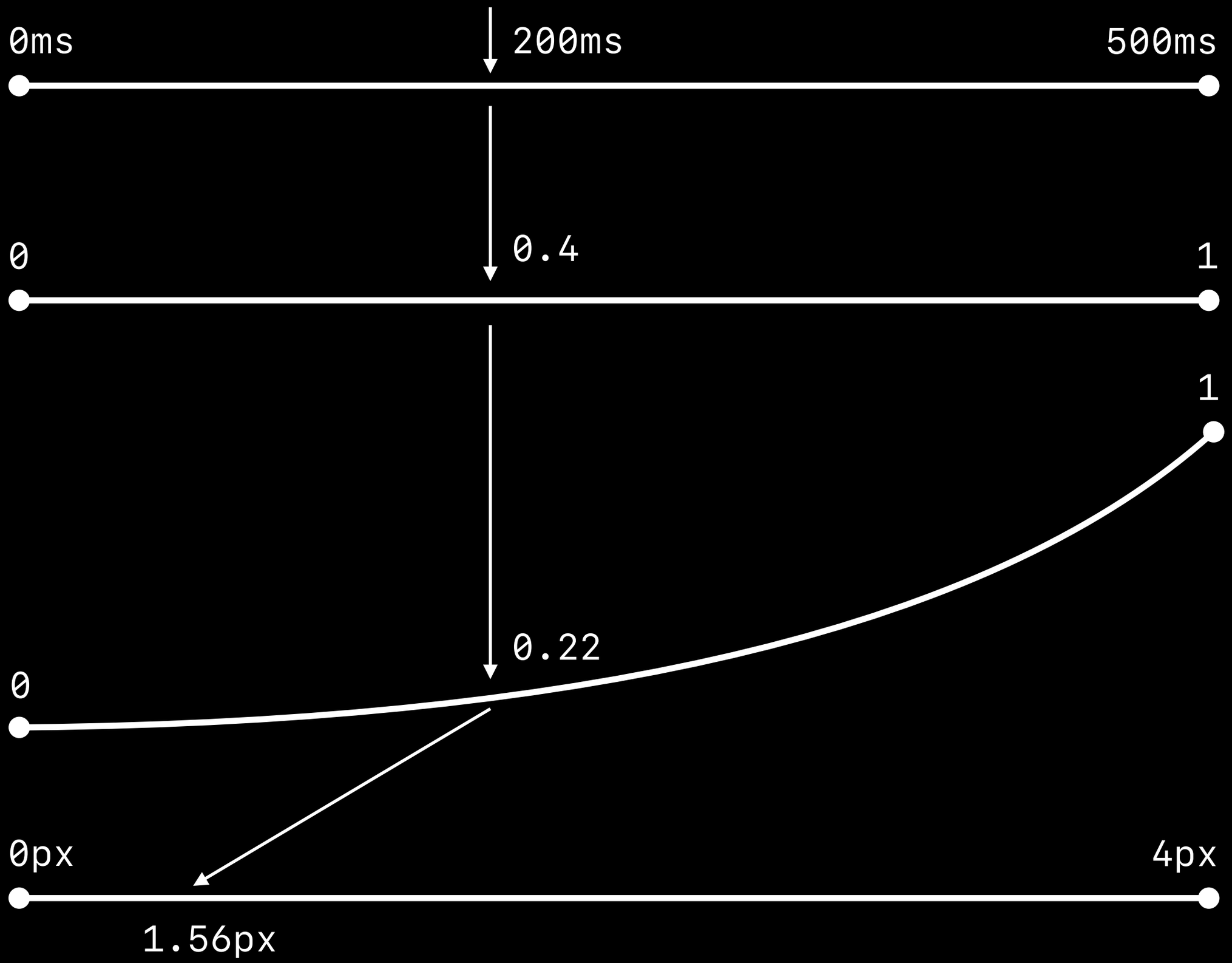
```
const DURATION = 1000;

const timeScale = scaleLinear()
  .domain([0, DURATION])
  .clamp(true);

const i = interpolateNumber(0, 4);

const t = timer((elapsed) => {
  const te = easeElasticOut(timeScale(elapsed), 4, 0.5);

  const radius = i(te);
  this.setState({ radius });

  if (elapsed > DURATION) return t.stop();
});
```


0ms
200ms
500ms
0
0.4
1

```
const DURATION = 1000;

const timeScale = scaleLinear()
  .domain([0, DURATION])
  .clamp(true);

const i = interpolateNumber(0, 4);

const t = timer((elapsed) => {
  const te = easeElasticOut(timeScale(elapsed), 4, 0.5);

  const radius = i(te);
  this.setState({ radius });

  if (elapsed > DURATION) return t.stop();
});
```


0ms
200ms
500ms
0
0.4
1
1
0.22
0

```
const DURATION = 1000;

const timeScale = scaleLinear()
  .domain([0, DURATION])
  .clamp(true);

const i = interpolateNumber(0, 4);

const t = timer((elapsed) => {
  const te = easeElasticOut(timeScale(elapsed), 4, 0.5);

  const radius = i(te);
  this.setState({ radius });

  if (elapsed > DURATION) return t.stop();
});
```


0ms
200ms
500ms
0
0.4
1
1
0
0.22
0px
4px
1.56px

plugging this into the chart

REACT-TRANSITION-GROUP

<Transition>

The Transition component lets you describe a transition from one component state to another over time with a simple declarative API

<TransitionGroup>

The TransitionGroup component manages a set of Transition components in a list.

```jsx
const Proint = ({ ... }) => (
  <Transition
    in
    timeout={DURATION}
    onEnter={this.handleEnter}
    {...others}
  >
    <circle {...} />
  </Transition>
);
```

```jsx
const Proint = ({ ... }) => (
  <Transition
    in
    timeout={DURATION}
    onEnter={this.handleEnter}
    {...others}
  >
    <circle {...} />
  </Transition>
);
```

```
<TransitionGroup component="g" appear>
  {data.map((d, i) => (
    <Point
      key={i}
      {...}
    />
  ))}
</TransitionGroup>
```

60 FPS has never been easier

but Alberto,

does it **scale**?

yes. yes it does.

there is a limit

quin+rebut
espera+tiempo
ajuda+els
passar+és
tots+estat
plano+tarifa
els+ara
els+rebuts
no_u+e_m
dia+teneis
si+tenia_n

this uses a **canvas**

```
drawGraph() {
  const { nodes } = this.props;
  const context = this.$canvas.getContext('2d');

  // Draw the nodes
  nodes.forEach(this.drawNode);
  // ...
}
```

```
containerRef(el) {
  this.$canvas = el;
}

render() {
  return (
    <canvas ref={this.containerRef} />
  );
}
```

```
drawGraph() {
  const { nodes } = this.props;
  const context = this.$canvas.getContext('2d');

  // Draw the nodes
  nodes.forEach(this.drawNode);
  // ...
}
```

```
containerRef(el) {
  this.$canvas = el;
}

render() {
  return (
    <canvas ref={this.containerRef} />
  );
}
```

```
drawGraph() {
  const { nodes } = this.props;
  const context = this.$canvas.getContext('2d');

  // Draw the nodes
  nodes.forEach(this.drawNode);
  // ...
}

containerRef(el) {
  this.$canvas = el;
}

render() {
  return (
    <canvas ref={this.containerRef} />
  );
}
```

```
drawGraph() {
  const { nodes } = this.props;
  const context = this.$canvas.getContext('2d');

  // Draw the nodes
  nodes.forEach(this.drawNode);
  // ...
}
```

```
containerRef(el) {
  this.$canvas = el;
}

render() {
  return (
    <canvas ref={this.containerRef} />
  );
}
```

```
handlePanAndZoom() {
  this.transform = {
    x: event.transform.x,
    y: event.transform.y,
    k: event.transform.k,
  };

  this.drawGraph();
}
```

PERFORMANCE CONSIDERATIONS:

- Always use PureComponent
- Be careful of shallow equality
- Keep the render function lean, avoid computations in render
- Prefer small components to leverage the React reconciliation algorithm

PERFORMANCE CONSIDERATIONS:

- Always use PureComponent
- Be careful of shallow equality
- Keep the render function lean, avoid computations in render
- Prefer small components to leverage the React reconciliation algorithm

```
data.map((d, i) => (
  <Point
    key={i}
    x={d.x}
    y={d.y}
    onMouseEnter={() => this.handleMouseEnterPoint(d)}
    onMouseLeave={() => this.handleMouseLeavePoint(d)}
  />
))
```

```
data.map((d, i) => (
  <Point
    key={i}
    x={d.x}
    y={d.y}
    onMouseEnter={() => this.handleMouseEnterPoint(d)}
    onMouseLeave={() => this.handleMouseLeavePoint(d)}
  />
))
```

```
data.map((d, i) => (
  <Point
    key={i}
    x={d.x}
    y={d.y}
    onMouseEnter={this.handleMouseEnterPoint}
    onMouseLeave={this.handleMouseLeavePoint}
  />
))
```

and that's it

Charts in React

The Right Way

Thank you.

EXAMPLE REPOSITORY:

goo.gl/H4L76g