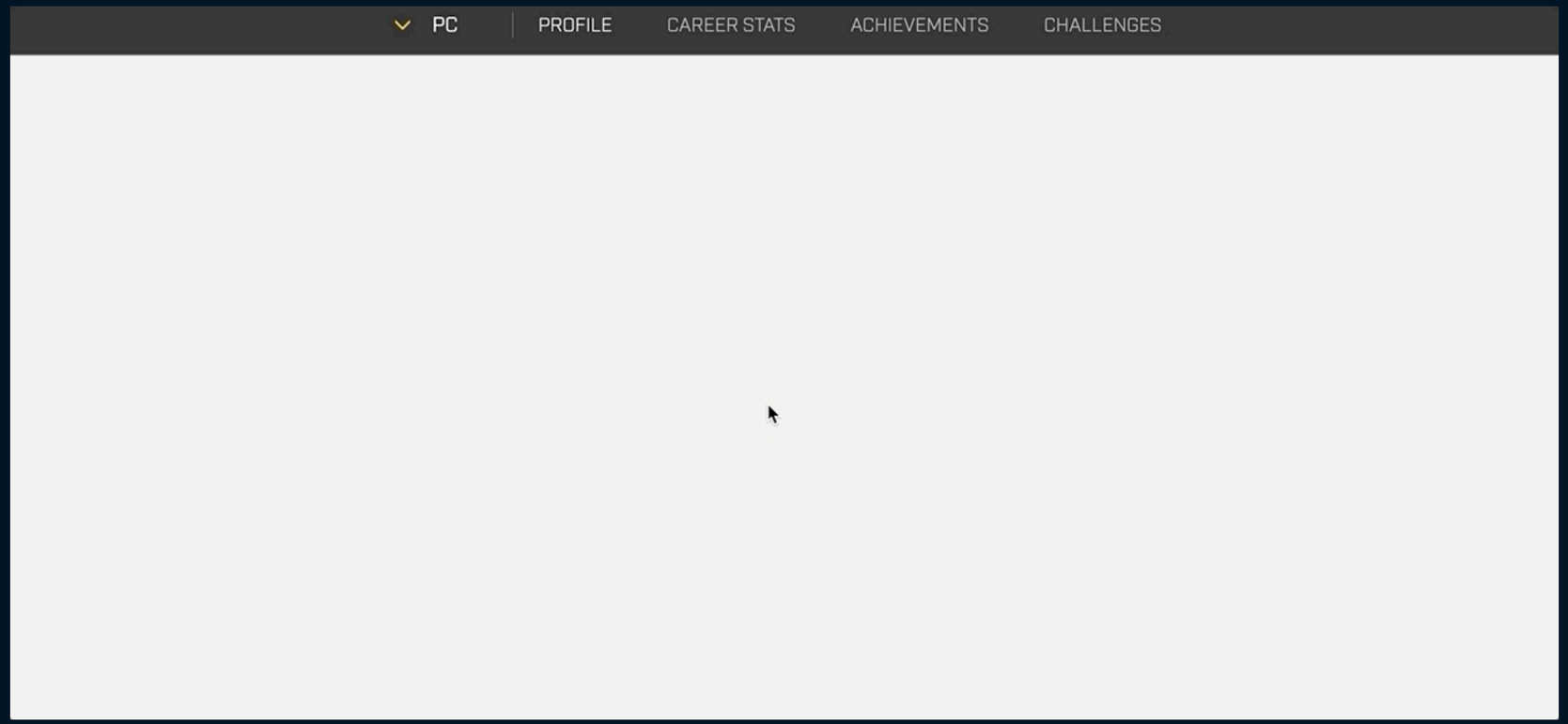
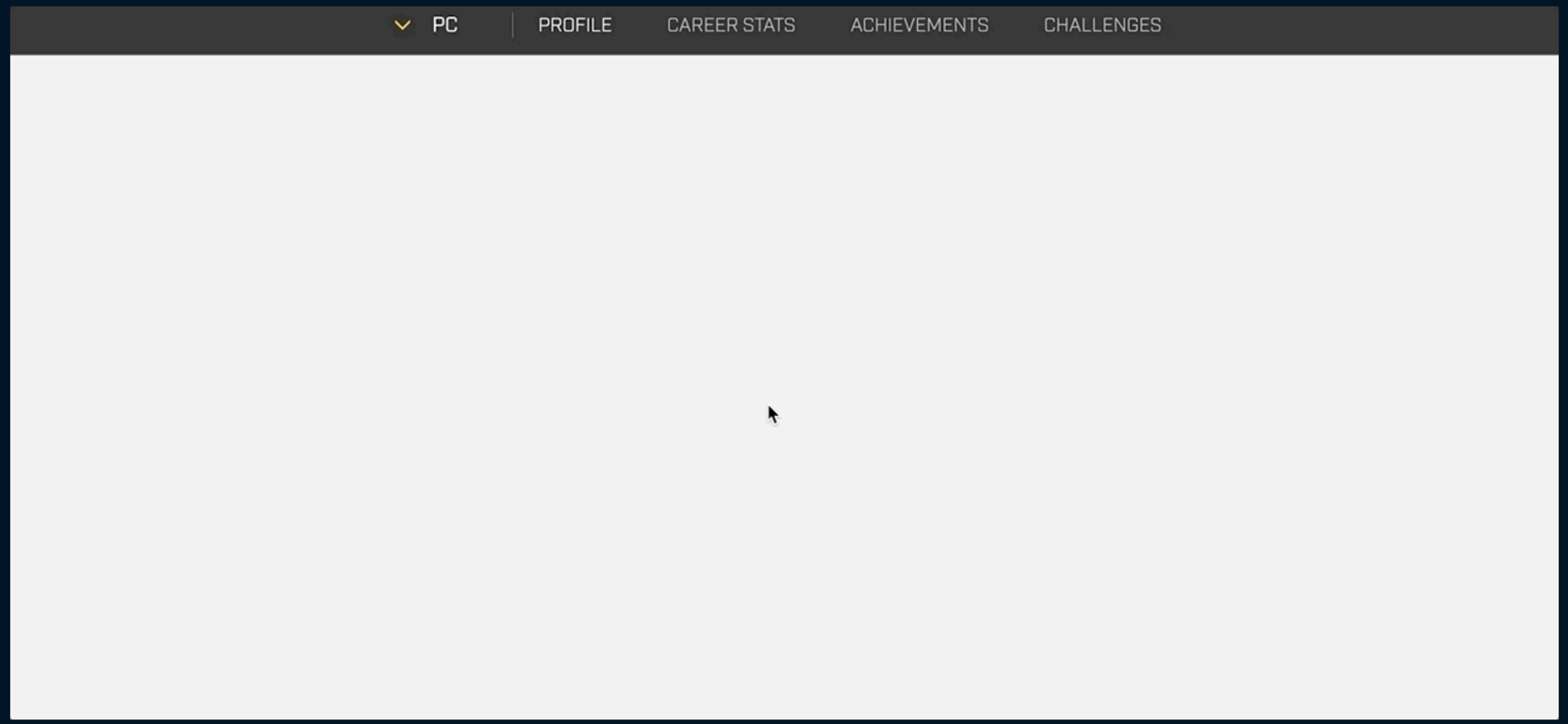
I really, really dislike chart libraries


PC
PROFILE
CAREER STATS
ACHIEVEMENTS
CHALLENGES


PC
PROFILE
CAREER STATS
ACHIEVEMENTS
CHALLENGES

Light Details

1 Alarms

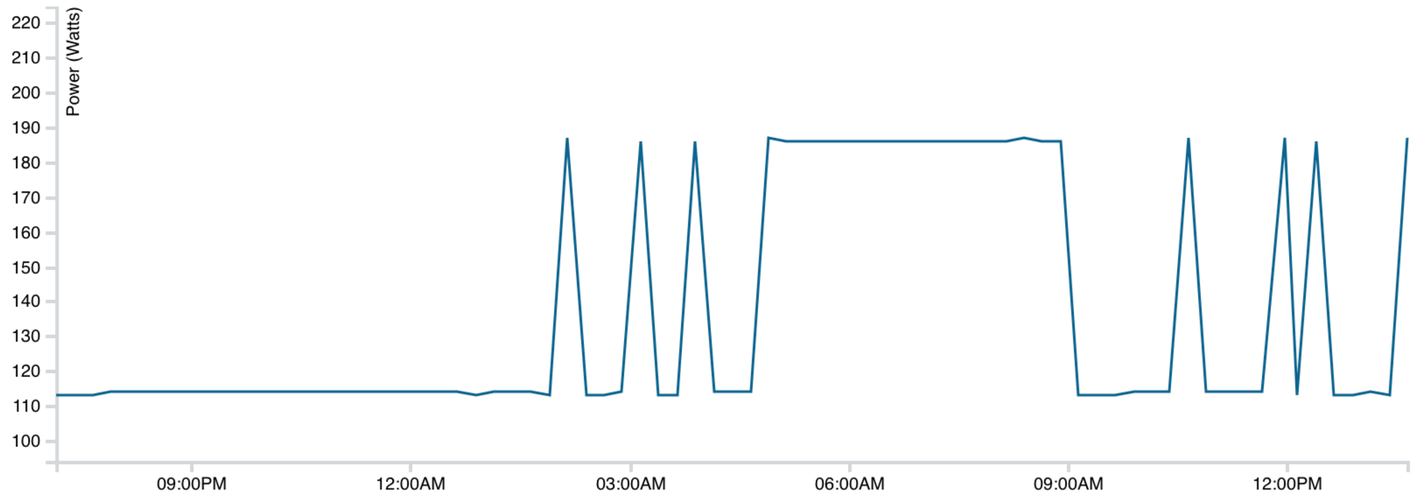
Light 65-T

EDIT LIGHT

100% OFF ON

Power Voltage

01/27/2020



GET CURRENT POWER DATA

The struggle with visualization libraries

The struggle with visualization libraries

- **Getting the design just right can be really painful**

The struggle with visualization libraries

Getting the design just right can be really painful

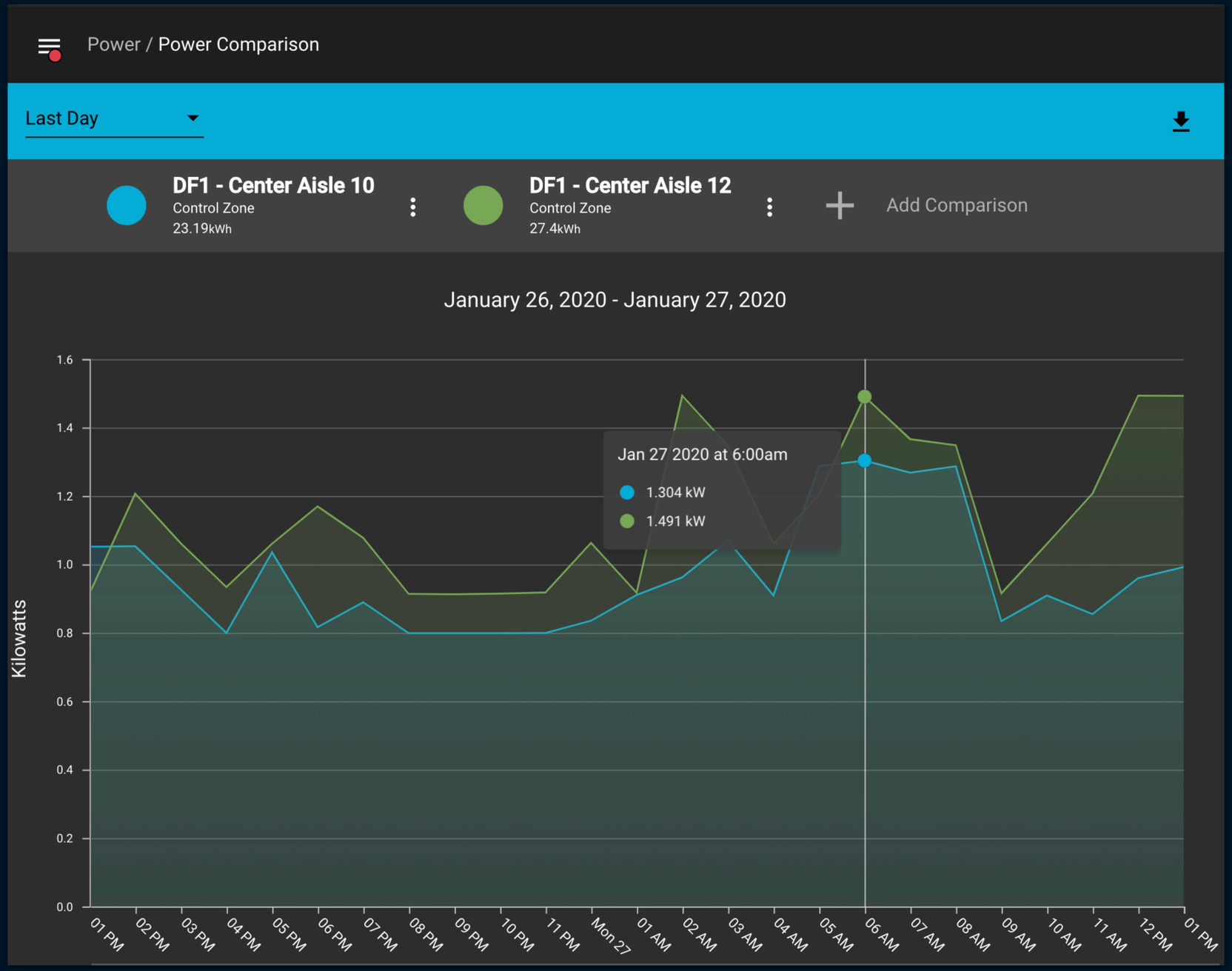
- **Hard to keep them looking & behaving right across upgrades**

The struggle with visualization libraries

Getting the design just right can be really painful

Hard to keep them looking & behaving right across upgrades

- Tend to be really config heavy


Power / Power Comparison
Last Day
DF1 - Center Aisle 10
Control Zone
23.19kWh
DF1 - Center Aisle 12
Control Zone
27.4kWh
Add Comparison
January 26, 2020 - January 27, 2020
Jan 27 2020 at 6:00am
1.304 kW
1.491 kW
Kilowatts
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
01 PM
02 PM
03 PM
04 PM
05 PM
06 PM
07 PM
08 PM
09 PM
10 PM
11 PM
Mon 27
01 AM
02 AM
03 AM
04 AM
05 AM
06 AM
07 AM
08 AM
09 AM
10 AM
11 AM
12 PM
01 PM

Making my own charts seemed terrifying

It was time to face my fears

Mike Ryan

@MikeRyanDev

Mike Ryan

@MikeRyanDev

Software Architect at Synapse 🏭 🌎 🌳

Mike Ryan

@MikeRyanDev

Software Architect at Synapse 🏭 🌎 🌳

Google Developer Expert

Mike Ryan

@MikeRyanDev

Software Architect at Synapse 🏭 🌎 🌳

Google Developer Expert

NgRx Co-Creator

Goals for improved visualizations

Goals for improved visualizations

- **Build on top of existing skill sets**

Goals for improved visualizations

Build on top of existing skill sets

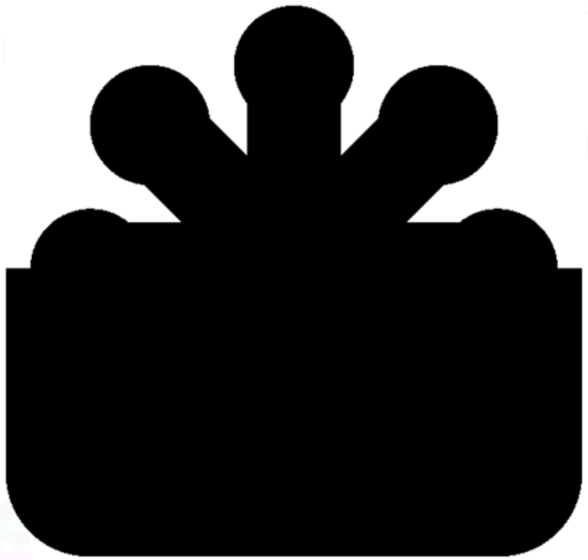
- **Endless amounts of customization**

Goals for improved visualizations

- Build on top of existing skill sets
- Endless amounts of customization
- Built to last well into the future

A new foe has appeared!

CHALLENGER APPROACHING

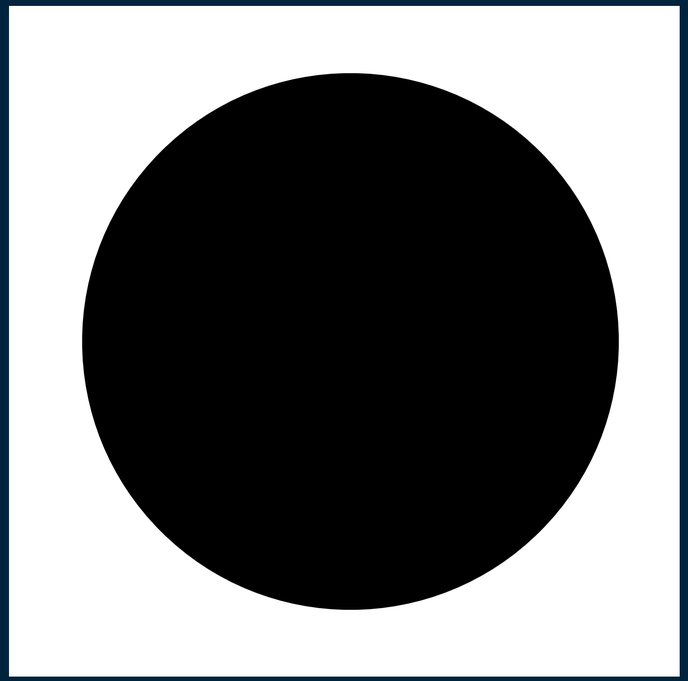


A new foe has appeared!

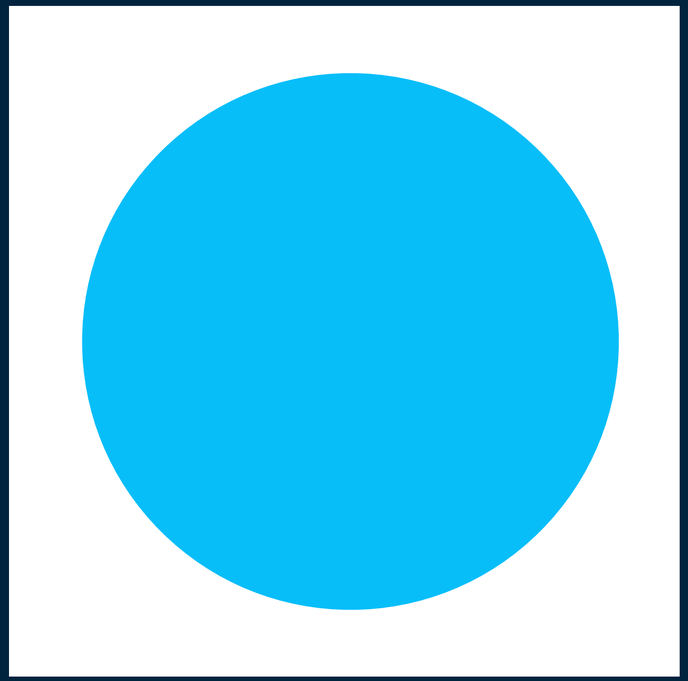
CHALLENGER APPROACHING

XML-based Vector Image Format

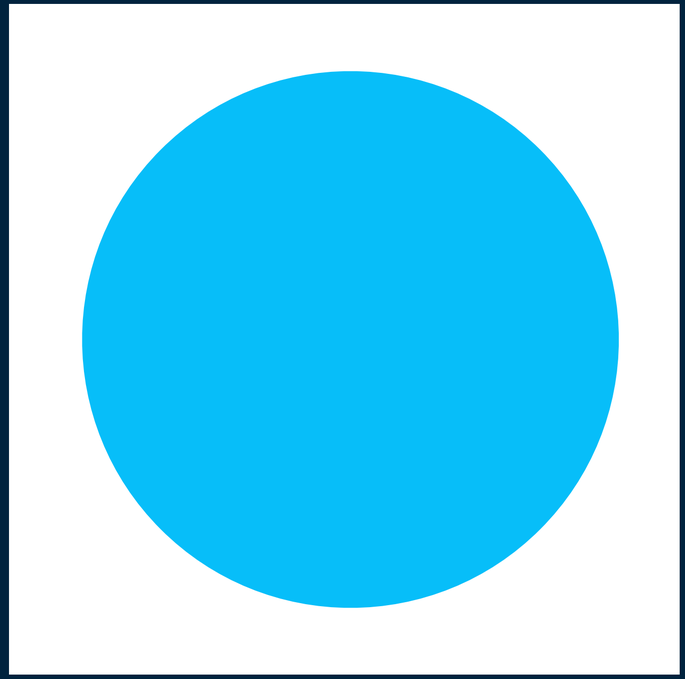
```
<svg width="500" height="500">
  <circle r="200" cx="250" cy="250" />
</svg>
```



```
<svg width="500" height="500">
  <circle r="200" cx="250" cy="250" />
</svg>
<style>
  circle {
    fill: blue;
  }
</style>
```



```
<svg width="500" height="500">
  <circle r="200" cx="250" cy="250" />
</svg>
<style>
  circle {
    fill: blue;
  }
</style>
<script>
  document
    .querySelector("circle")
    .addEventListener("click", () => {
      alert("Clicked!");
    });
</script>
```



We can use SVG in Angular

```
@Component({
  template: `
    <svg width="500" height="500">
      <svg:circle [attr.r]="radius" cx="250" cy="250" />
    </svg>
  `,
  styles: [`
    circle {
      fill: blue;
    }
  `]
})
export class SVGCircleComponent {
  radius = 200;
}
```

```
@Component({
  template: `
    <svg width="500" height="500">
      <svg:circle [attr.r]="radius" cx="250" cy="250" />
    </svg>
  `,
  styles: [`
    circle {
      fill: blue;
    }
  `]
})
export class SVGCircleComponent {
  radius = 200;
}
```

```
@Component({
  template: `
    <svg width="500" height="500">
      <svg:circle [attr.r]="radius" cx="250" cy="250" />
    </svg>
  `,
  styles: [`
    circle {
      fill: blue;
    }
  `]
})
export class SVGCircleComponent {
  radius = 200;
}
```

```
@Component({
  template: `
    <svg width="500" height="500">
      <svg:circle [attr.r]="radius" cx="250" cy="250" />
    </svg>
  `,
  styles: [`
    circle {
      fill: blue;
    }
  `]
})
export class SVGCircleComponent {
  radius = 200;
}
```

What about custom components?

SVG does not support custom element names

```
<svg width="500" height="500">
  <my-custom-circle />
</svg>
```

```
<svg width="500" height="500">
  <g></g>
</svg>
```

```
<svg width="500" height="500">
  <g my-custom-circle></g>
</svg>
```

```
@Component({
  selector: "svg:g[my-custom-circle]",
  template: `
    <svg:circle r="100" [attr.cx]="x" [attr.cy]="y" />
  `,
  styles: [`
    circle {
      fill: blue;
    }
  `]
})
export class CustomCircleComponent {
  @Input() x: number;
  @Input() y: number;
}
```

```
@Component({
  selector: "svg:g[my-custom-circle]",
  template: `
    <svg:circle r="100" [attr.cx]="x" [attr.cy]="y" />
  `,
  styles: [`
    circle {
      fill: blue;
    }
  `]
})
export class CustomCircleComponent {
  @Input() x: number;
  @Input() y: number;
}
```

```
@Component({
  selector: "svg:g[my-custom-circle]",
  template: `
    <svg:circle r="100" [attr.cx]="x" [attr.cy]="y" />
  `,
  styles: [`
    circle {
      fill: blue;
    }
  `]
})
export class CustomCircleComponent {
  @Input() x: number;
  @Input() y: number;
}
```

```
@Component({
  selector: "svg:g[my-custom-circle]",
  template: `
    <svg:circle r="100" [attr.cx]="x" [attr.cy]="y" />
  `,
  styles: [`
    circle {
      fill: blue;
    }
  `]
})
export class CustomCircleComponent {
  @Input() x: number;
  @Input() y: number;
}
```

```
@Component({
  selector: "svg:g[my-custom-circle]",
  template: `
    <svg:circle r="100" [attr.cx]="x" [attr.cy]="y" />
  `,
  styles: [`
    circle {
      fill: blue;
    }
  `]
})
export class CustomCircleComponent {
  @Input() x: number;
  @Input() y: number;
}
```

```
@Component({
  selector: "svg:g[my-custom-circle]",
  template: `
    <svg:circle r="100" [attr.cx]="x" [attr.cy]="y" />
  `,
  styles: [`
    circle {
      fill: blue;
    }
  `]
})
export class CustomCircleComponent {
  @Input() x: number;
  @Input() y: number;
}
```

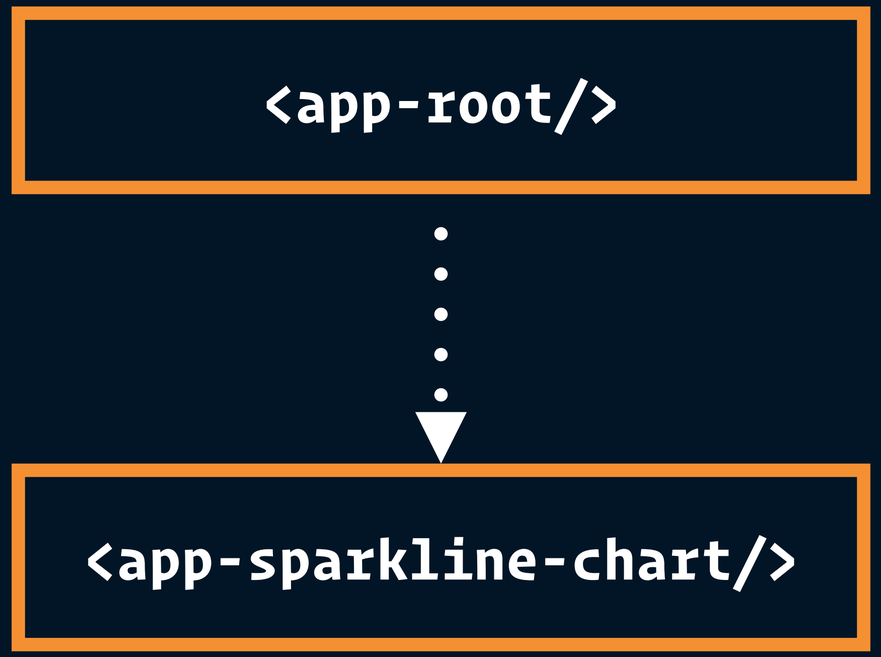
```
@Component({
  template: `
    <svg width="500" height="500">
      <svg:g my-custom-circle [x]="250" [y]="250"></svg:g>
    </svg>
  `
})
export class CustomSVGComponent {}
```

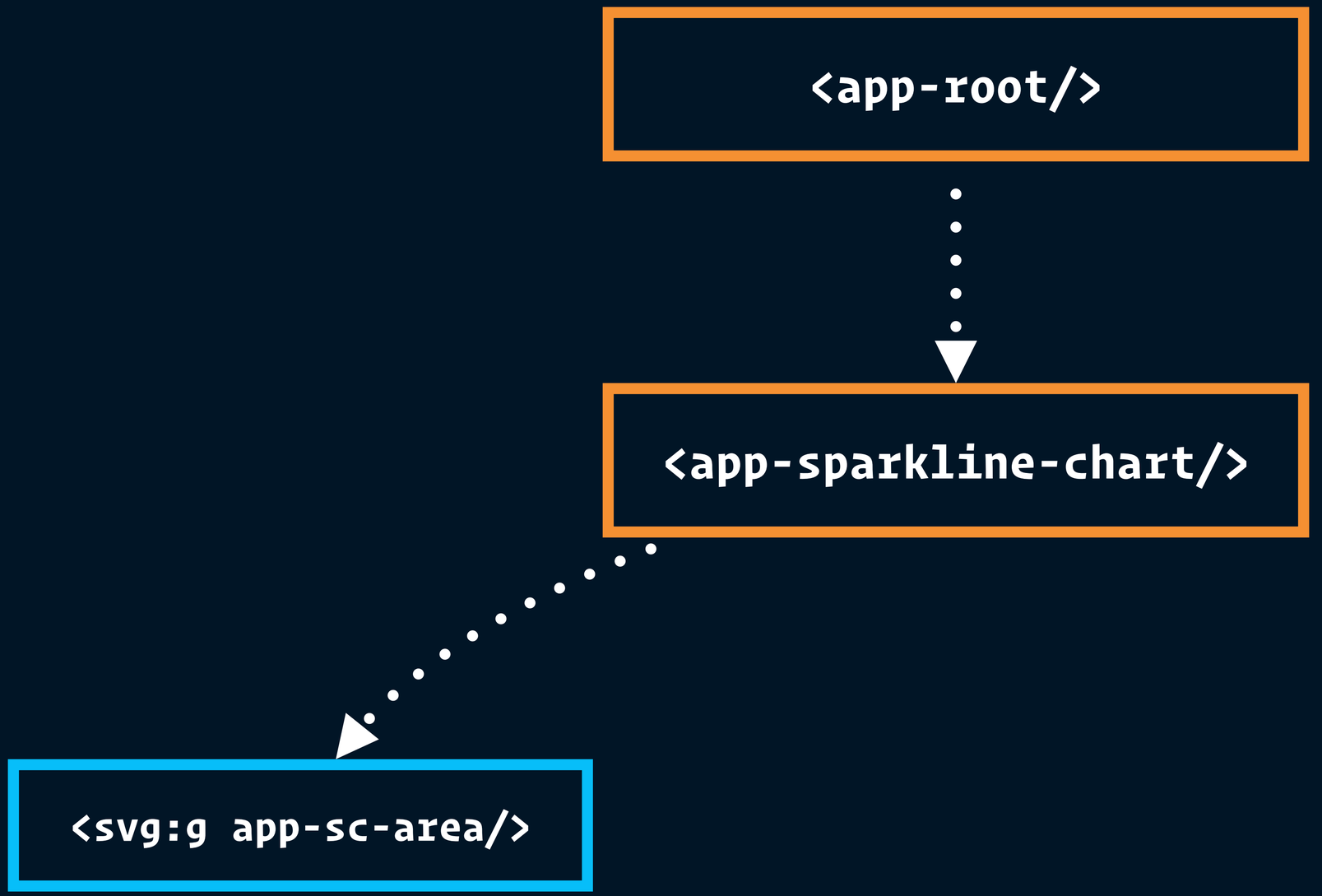
Building a Chart with SVG+Angular

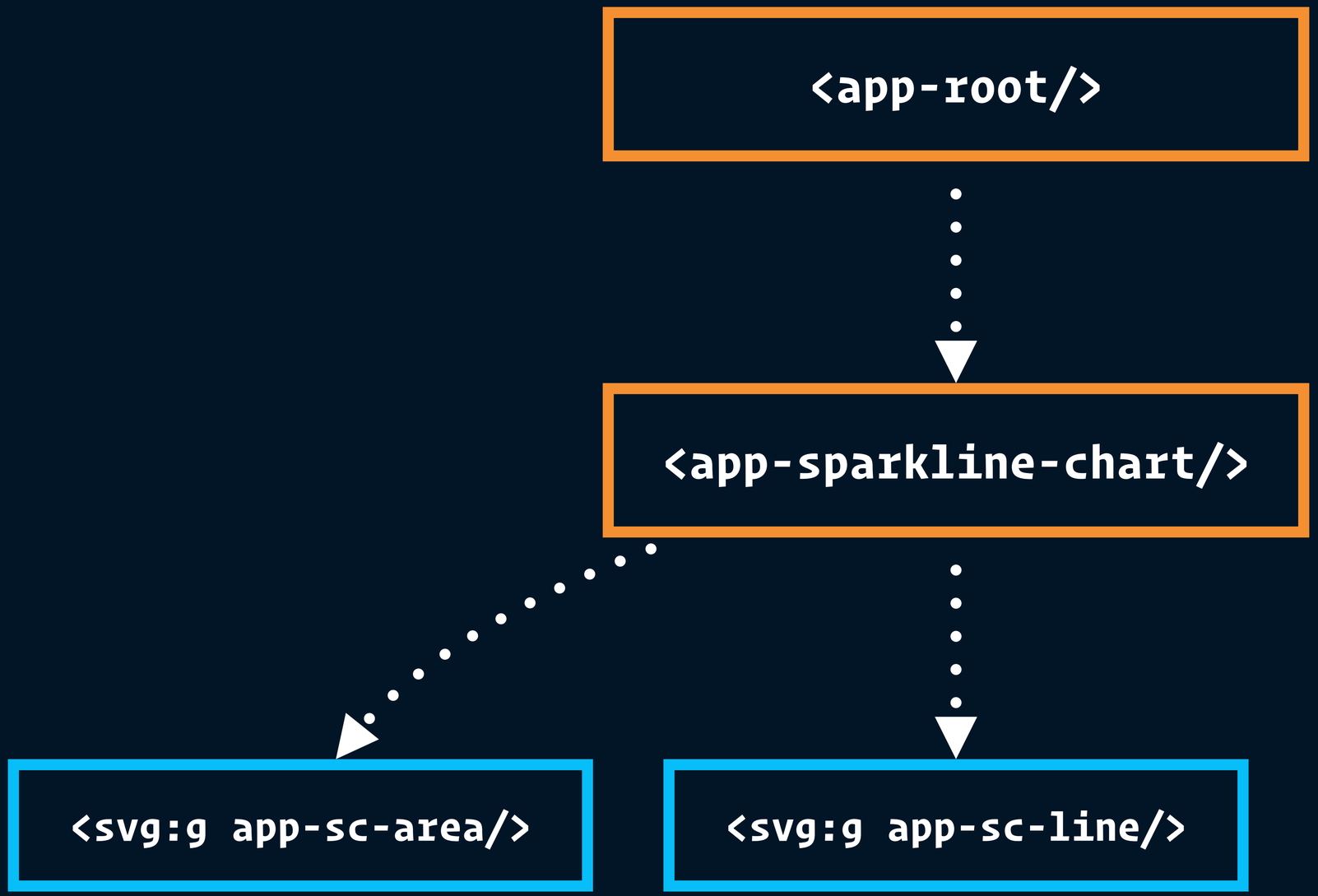

Movies By Year

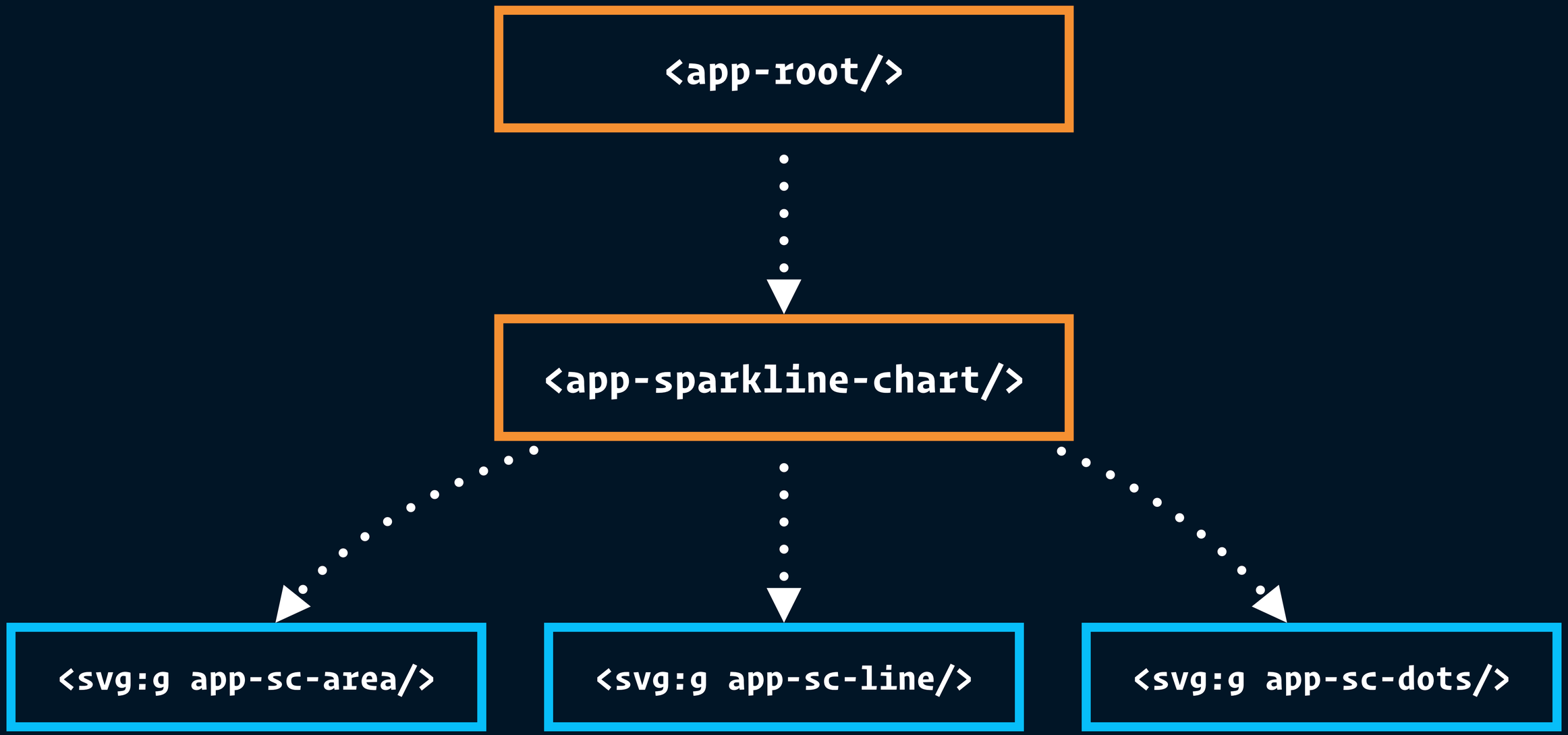

Movies By Year

<app-root/>


<app-root/>
<app-sparkline-chart/>


<app-root/>
<app-sparkline-chart/>
<svg:g app-sc-area/>


<app-root/>
<app-sparkline-chart/>
<svg:g app-sc-area/>
<svg:g app-sc-line/>


<app-root/>
<app-sparkline-chart/>
<svg:g app-sc-area/>
<svg:g app-sc-line/>
<svg:g app-sc-dots/>

```ts
export interface Point {
  x: number;
  y: number;
}
```

```
@Component({
  selector: "app-root",
  template: `
    <app-sparkline-chart [points]="points$ | async"></app-sparkline-chart>
  `
})
export class AppComponent {
  constructor(readonly movies: MoviesService) {}

  points$: Observable<Point[]> = this.movies.getMoviesByYear().pipe(
    map(movies => movies.map(({ year, movies }) => {
      return {
        x: year,
        y: movies
      };
    }))
  );
}
```

```
@Component({
  selector: "app-root",
  template: `
    <app-sparkline-chart [points]="points$ | async"></app-sparkline-chart>
  `
})
export class AppComponent {
  constructor(readonly movies: MoviesService) {}

  points$: Observable<Point[]> = this.movies.getMoviesByYear().pipe(
    map(movies => movies.map(({ year, movies }) => {
      return {
        x: year,
        y: movies
      };
    }))
  );
}
```

```
@Component({
  selector: "app-root",
  template: `
    <app-sparkline-chart [points]="points$ | async"></app-sparkline-chart>
  `
})
export class AppComponent {
  constructor(readonly movies: MoviesService) {}

  points$: Observable<Point[]> = this.movies.getMoviesByYear().pipe(
    map(movies => movies.map(({ year, movies }) => {
      return {
        x: year,
        y: movies
      };
    }))
  );
}
```

```
@Component({
  selector: "app-root",
  template: `
    <app-sparkline-chart [points]="points$ | async"></app-sparkline-chart>
  `
})
export class AppComponent {
  constructor(readonly movies: MoviesService) {}

  points$: Observable<Point[]> = this.movies.getMoviesByYear().pipe(
    map(movies => movies.map(({ year, movies }) => {
      return {
        x: year,
        y: movies
      };
    }))
  );
}
```

```
@Component({
  selector: "app-root",
  template: `
    <app-sparkline-chart [points]="points$ | async"></app-sparkline-chart>
  `
})
export class AppComponent {
  constructor(readonly movies: MoviesService) {}

  points$: Observable<Point[]> = this.movies.getMoviesByYear().pipe(
    map(movies => movies.map(({ year, movies }) => {
      return {
        x: year,
        y: movies
      };
    }))
  );
}
```

Plotting Data

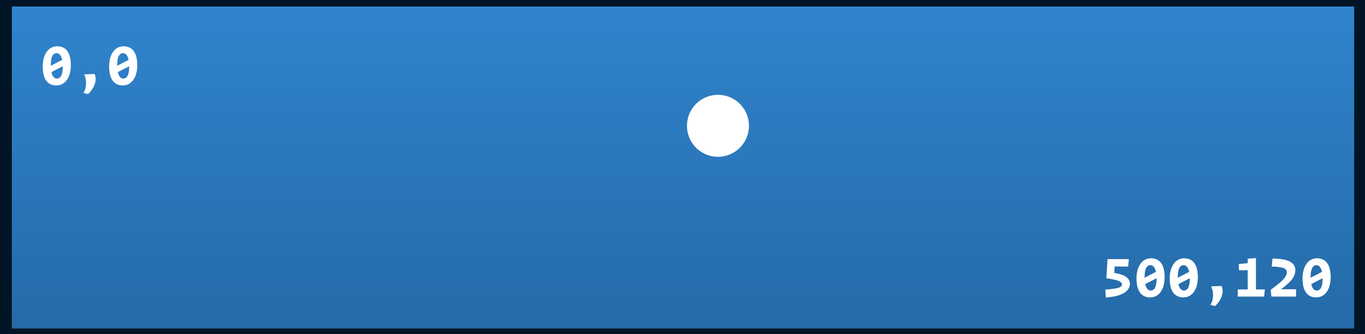

0,0
500,120

```
{ x: 2007, y: 538 }
```


{ x: 2007, y: 538 }
0,0
500,120


{ x: 2007, y: 538 }
0,0
500,120

```
{ x: 2007, y: 538 }
```



DOMAIN

[2002, 2003, 2004, ..., 2015, 2016, 2017]

RANGE

[0, 50, 100, ..., 400, 450, 500]

DOMAIN - DATA

[2002, 2003, 2004, ..., 2015, 2016, 2017]

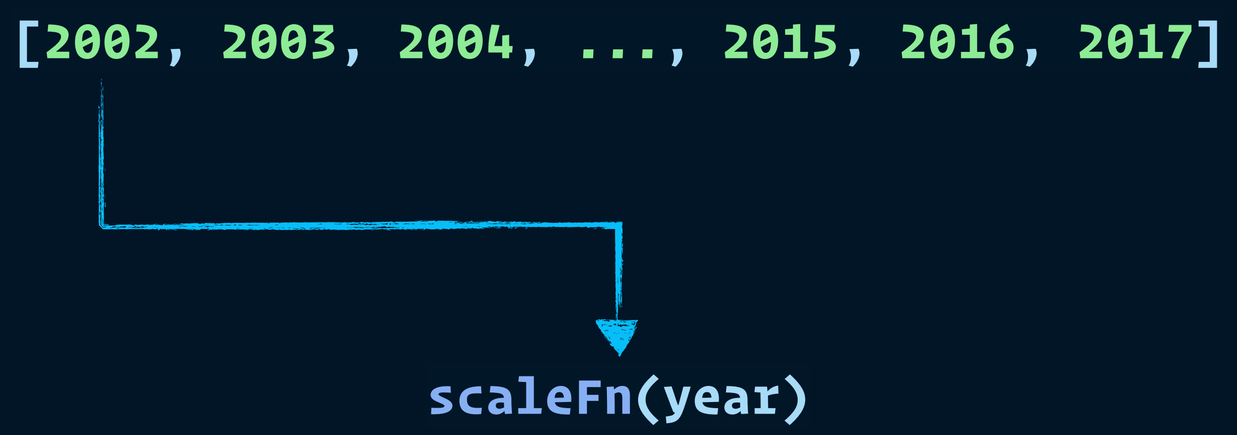
RANGE - RESULT

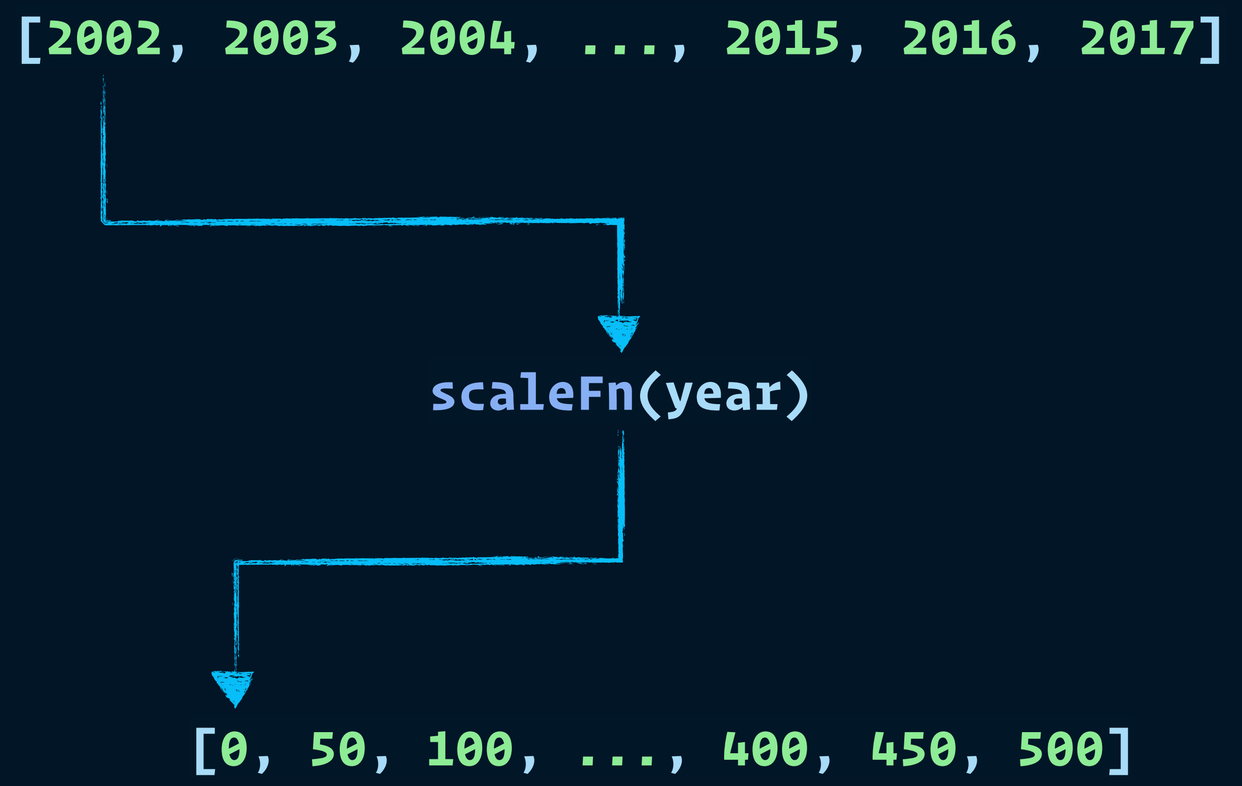
[0, 50, 100, ..., 400, 450, 500]

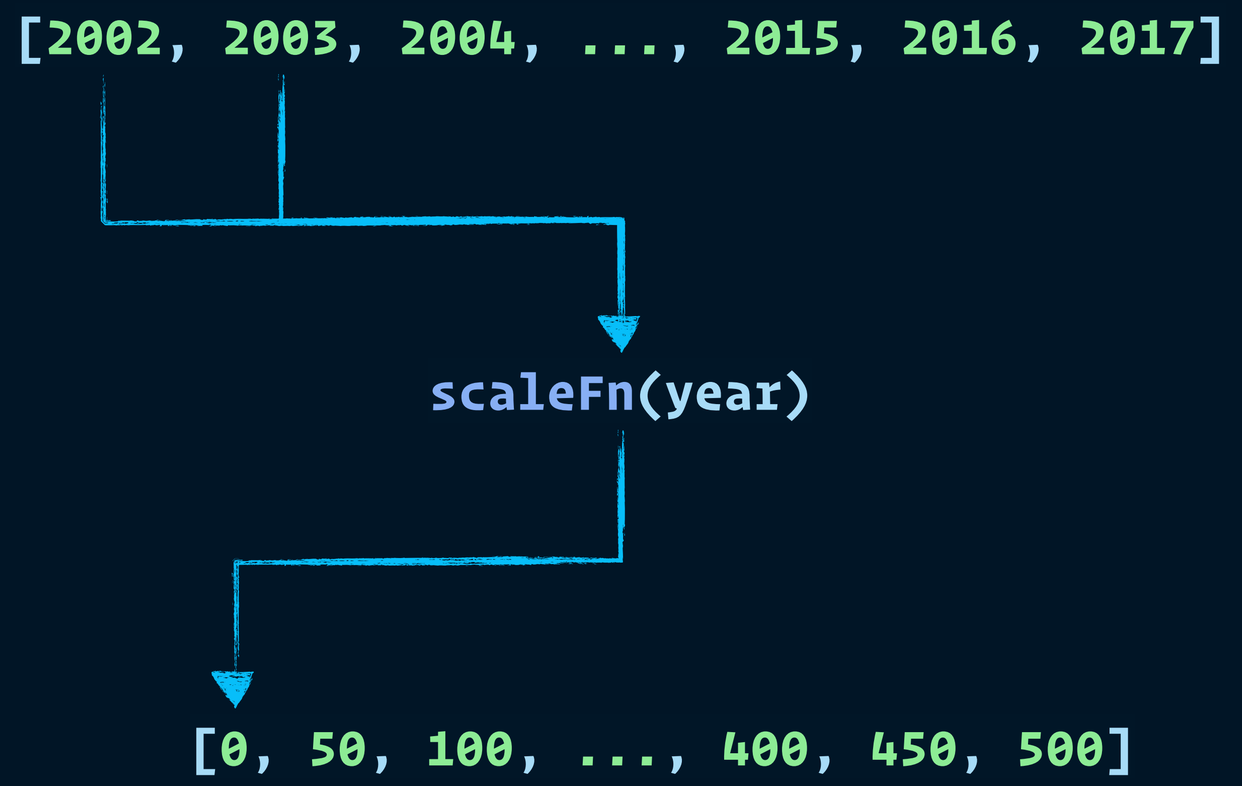
[2002, 2003, 2004, ..., 2015, 2016, 2017]

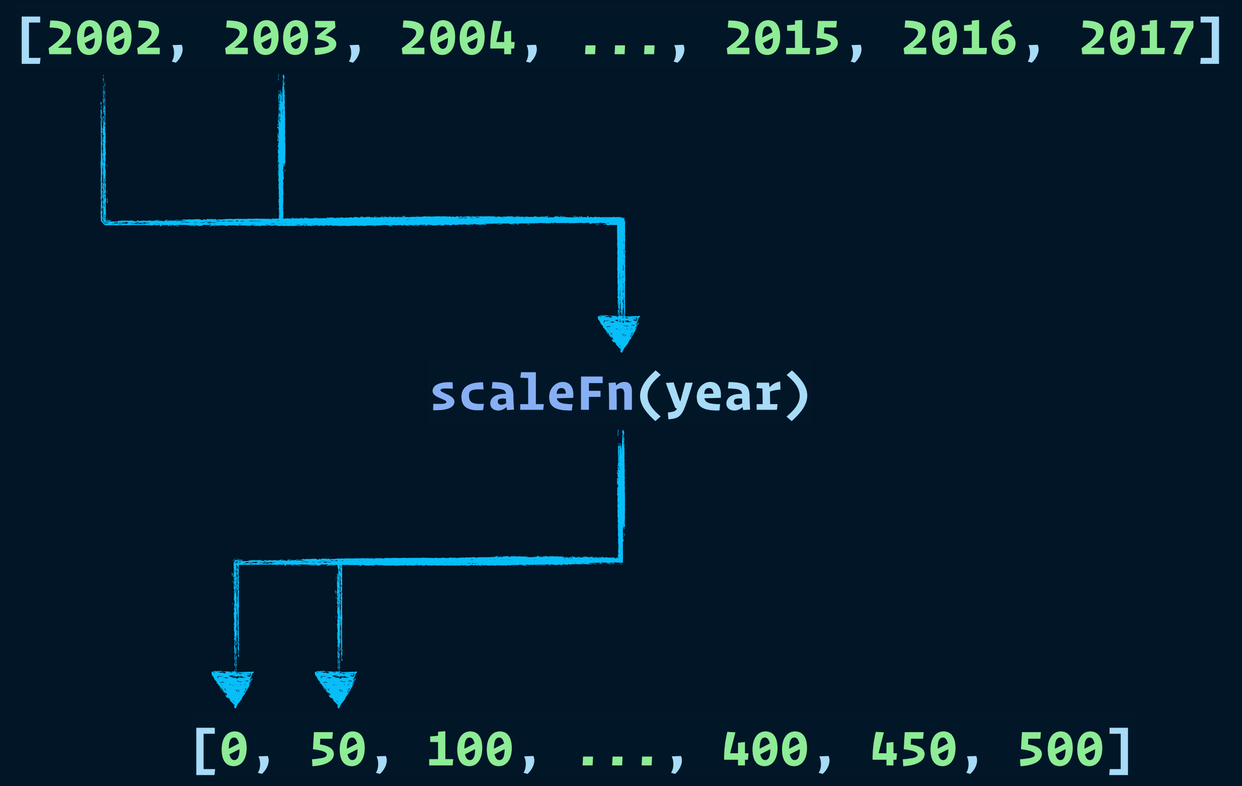
scaleFn(year)

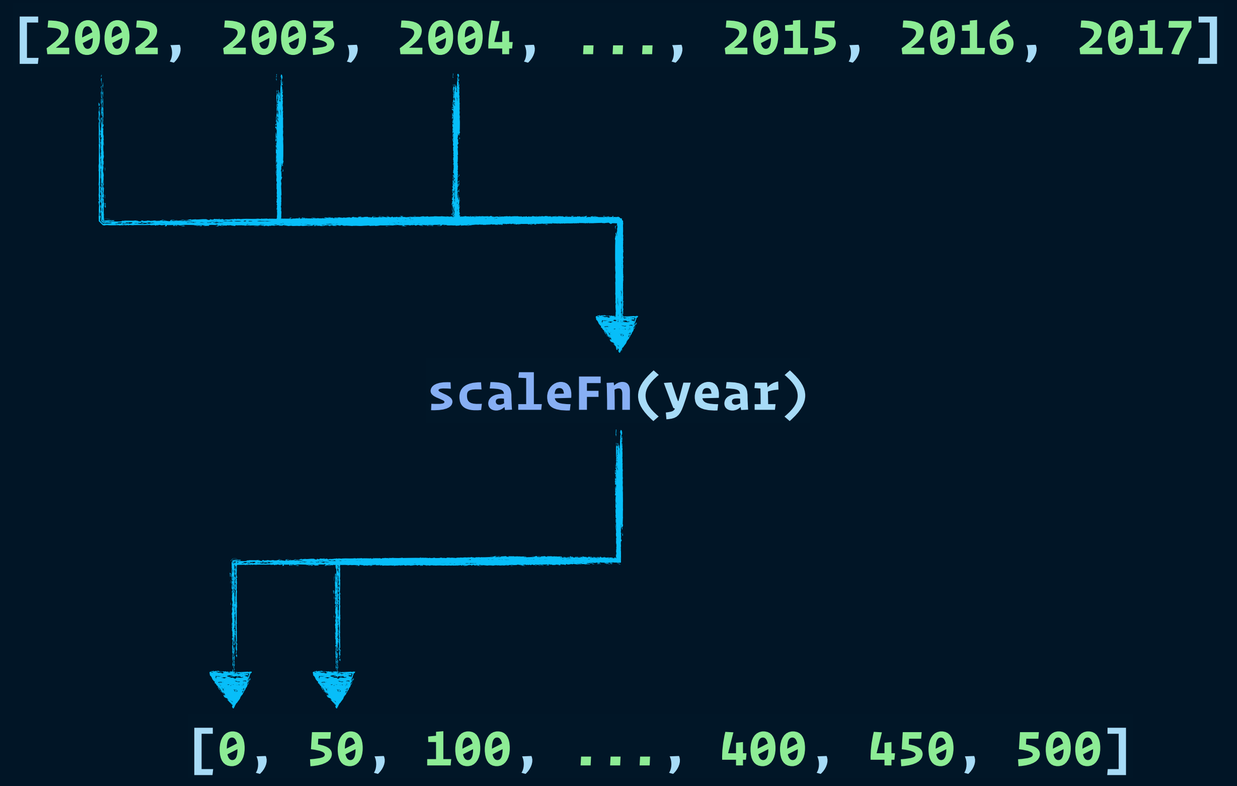
[0, 50, 100, ..., 400, 450, 500]

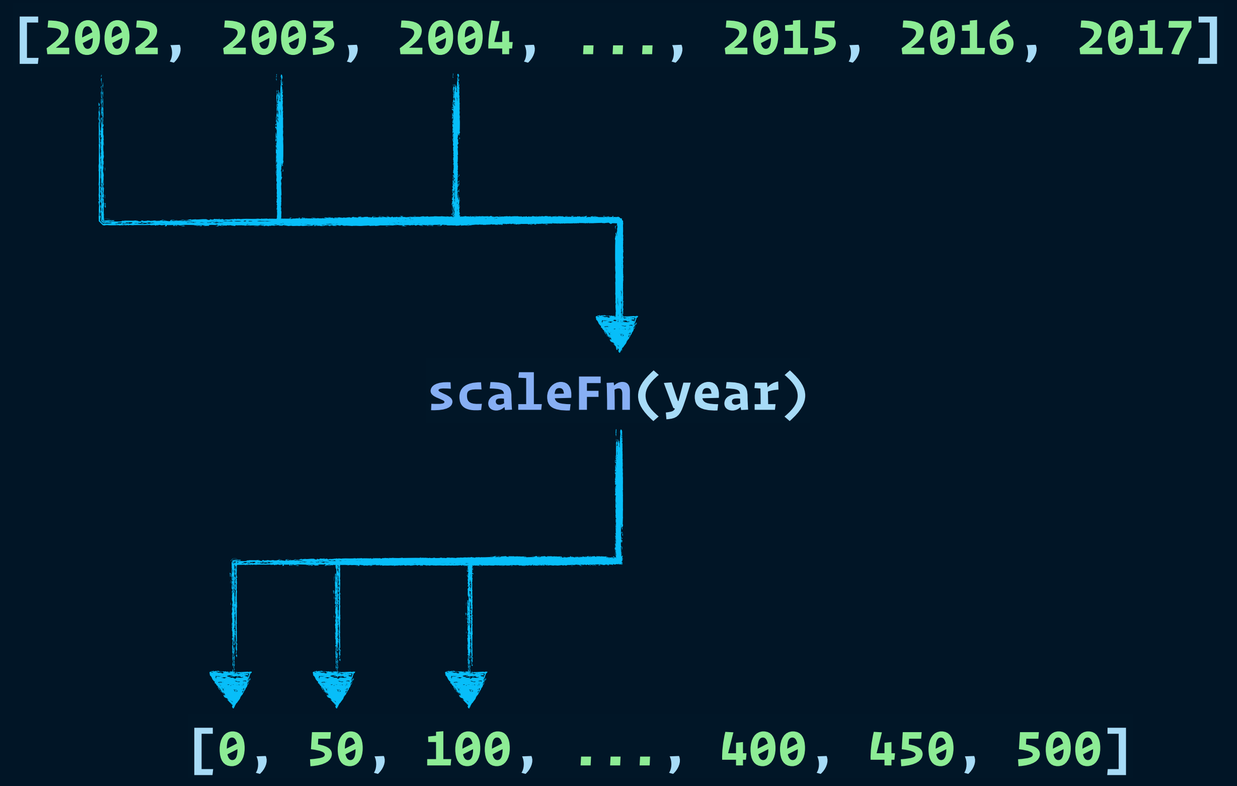

[2002, 2003, 2004, ..., 2015, 2016, 2017]
scaleFn(year)
[0, 50, 100, ..., 400, 450, 500]

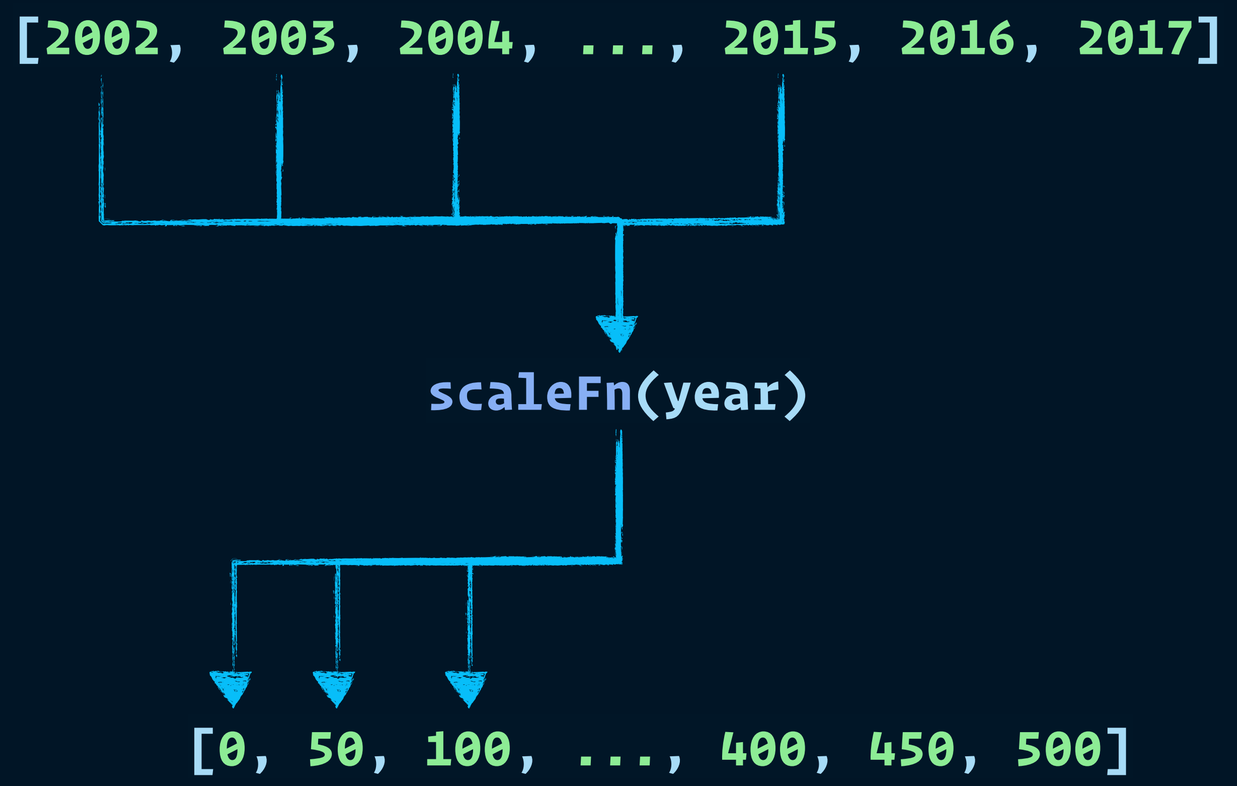

[2002, 2003, 2004, ..., 2015, 2016, 2017]
scaleFn(year)
[0, 50, 100, ..., 400, 450, 500]

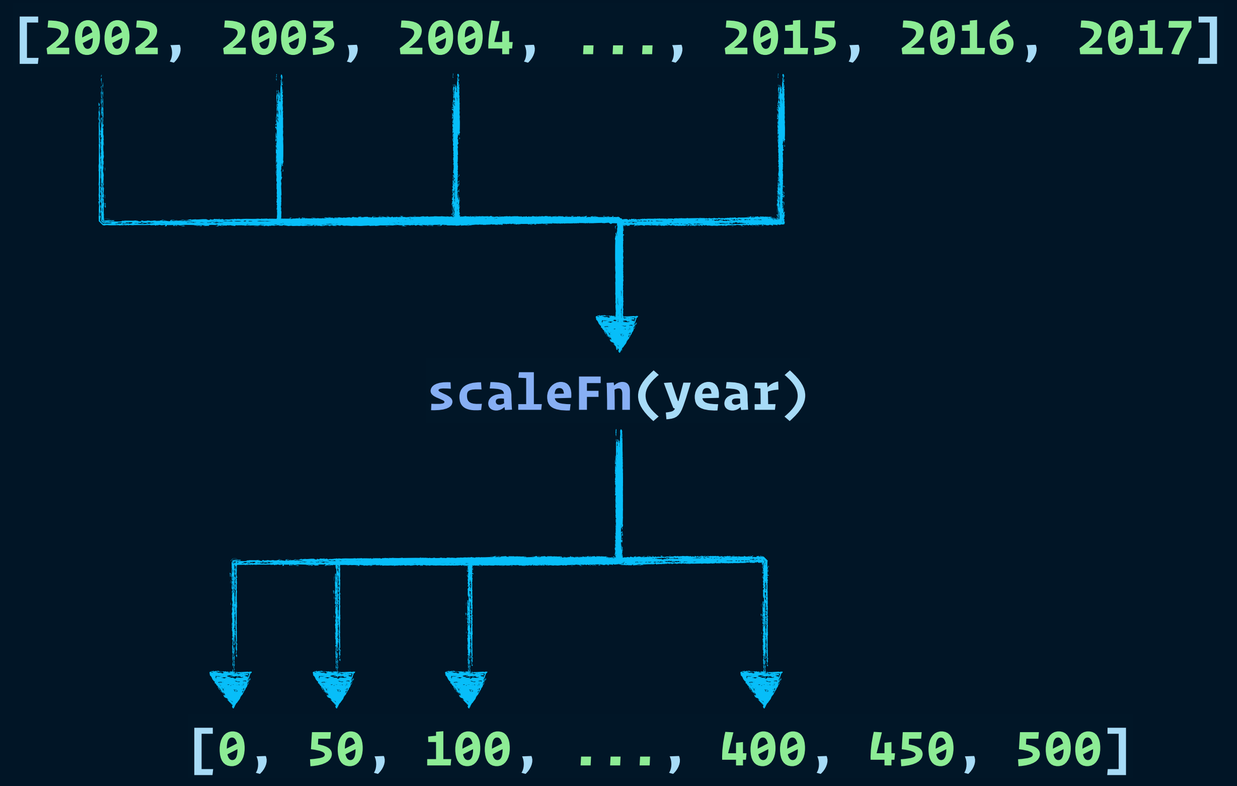

[2002, 2003, 2004, ..., 2015, 2016, 2017]
scaleFn(year)
[0, 50, 100, ..., 400, 450, 500]

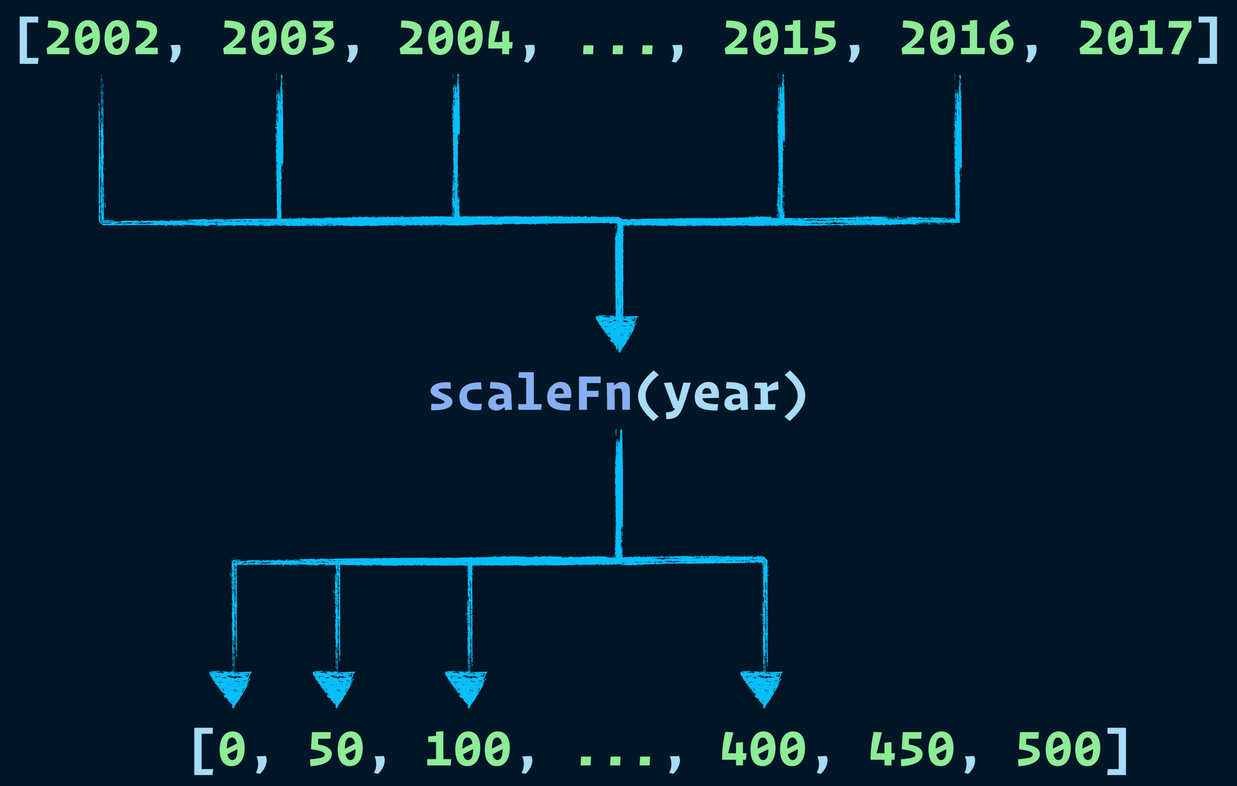

[2002, 2003, 2004, ..., 2015, 2016, 2017]
scaleFn(year)
[0, 50, 100, ..., 400, 450, 500]

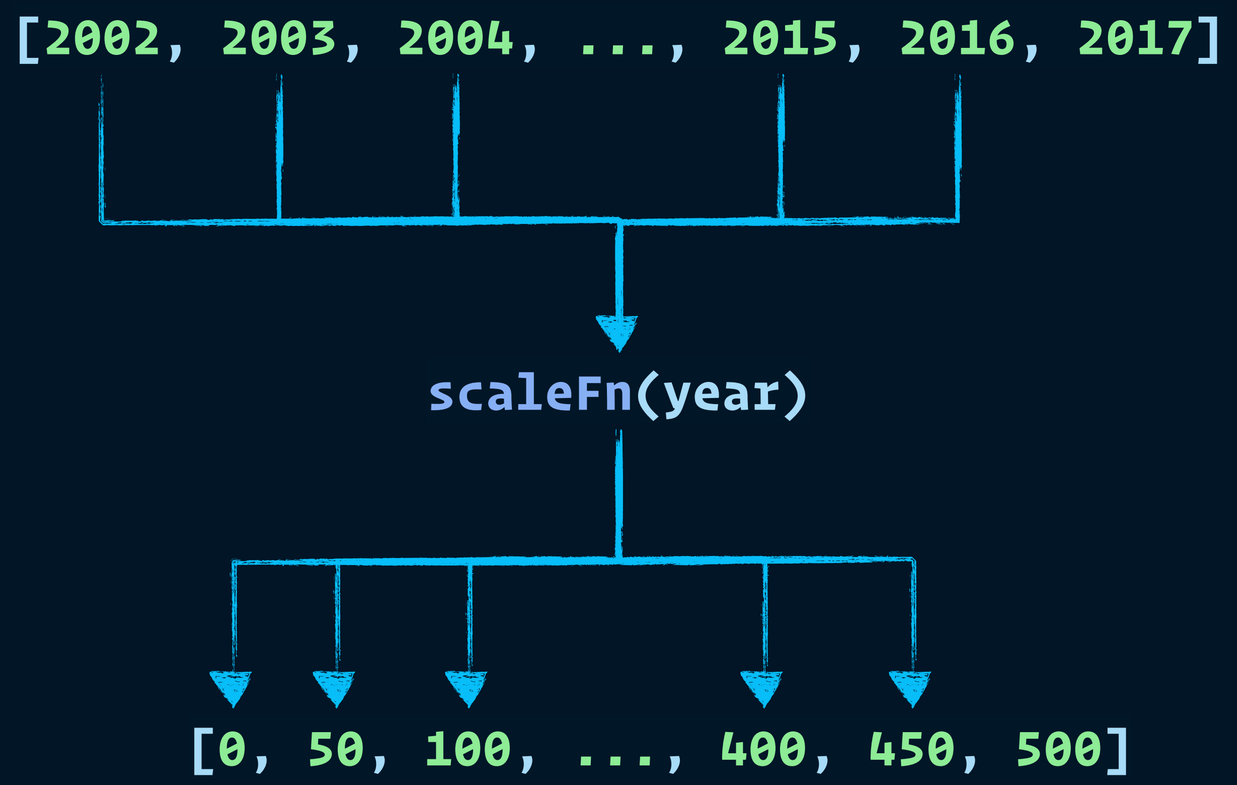

[2002, 2003, 2004, ..., 2015, 2016, 2017]
scaleFn(year)
[0, 50, 100, ..., 400, 450, 500]

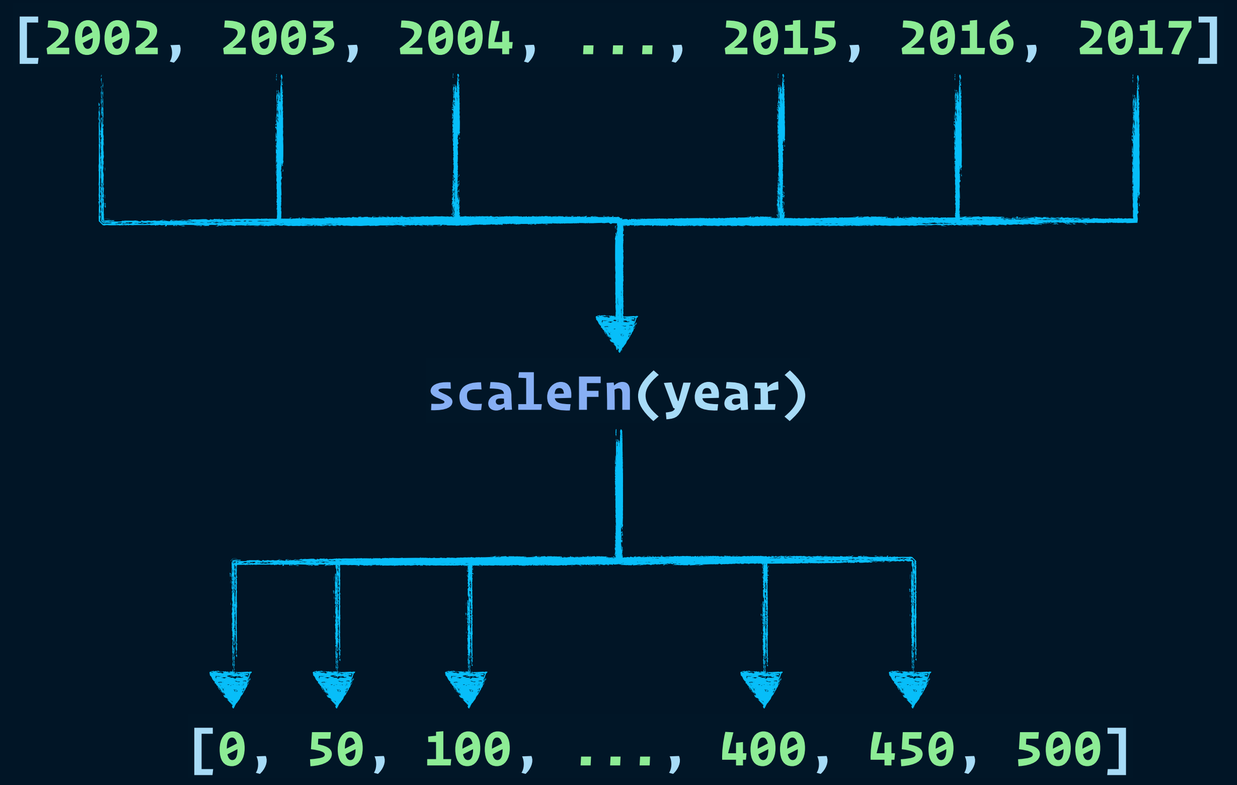

[2002, 2003, 2004, ..., 2015, 2016, 2017]
scaleFn(year)
[0, 50, 100, ..., 400, 450, 500]

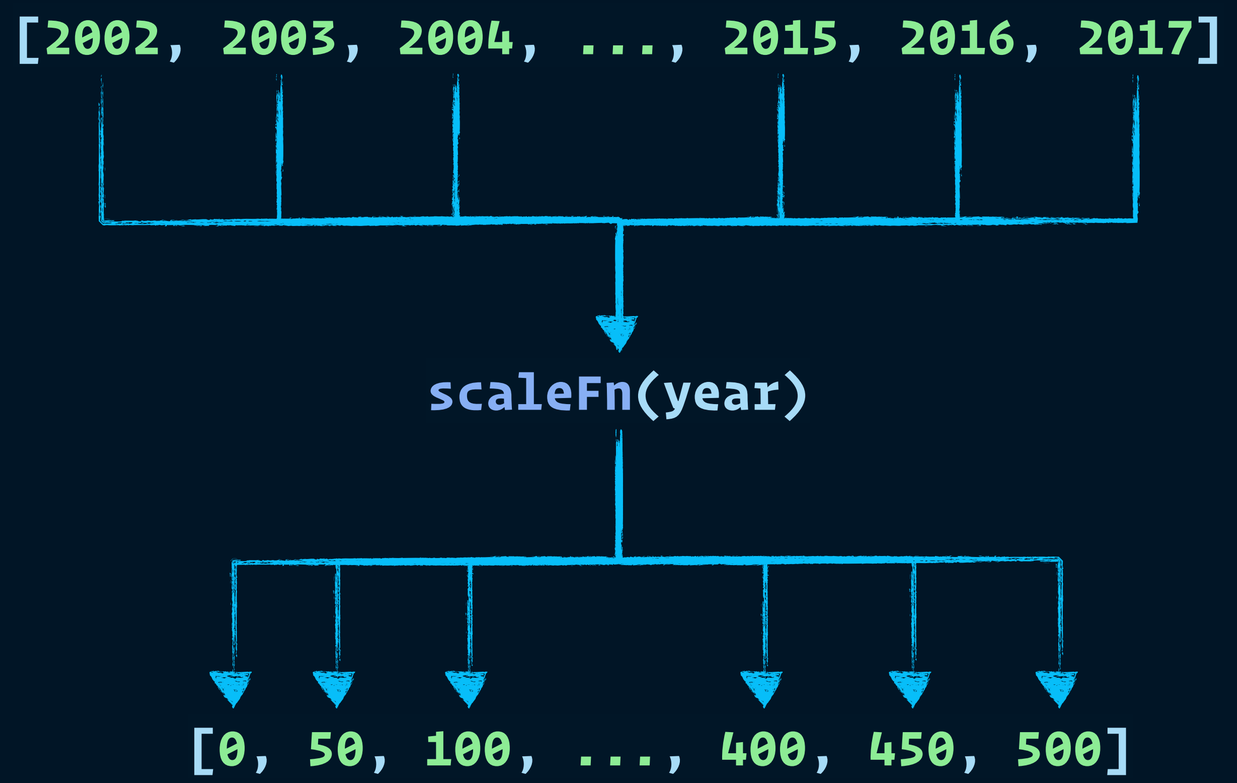

[2002, 2003, 2004, ..., 2015, 2016, 2017]
scaleFn(year)
[0, 50, 100, ..., 400, 450, 500]


[2002, 2003, 2004, ..., 2015, 2016, 2017]
scaleFn(year)
[0, 50, 100, ..., 400, 450, 500]


[2002, 2003, 2004, ..., 2015, 2016, 2017]
scaleFn(year)
[0, 50, 100, ..., 400, 450, 500]


[2002, 2003, 2004, ..., 2015, 2016, 2017]
scaleFn(year)
[0, 50, 100, ..., 400, 450, 500]


[2002, 2003, 2004, ..., 2015, 2016, 2017]
scaleFn(year)
[0, 50, 100, ..., 400, 450, 500]


[2002, 2003, 2004, ..., 2015, 2016, 2017]
scaleFn(year)
[0, 50, 100, ..., 400, 450, 500]

```ts
export function createLinearScale(
  domain: [number, number],
  range: [number, number]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],
  range: [number, number]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],
  range: [number, number]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],
  range: [number, number]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],
  range: [number, number]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],
  range: [number, number]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],
  range: [number, number]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],
  range: [number, number]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],
  range: [number, number]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],                    > [2002, 2017]
  range: [number, number]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],                    > [2002, 2017]
  range: [number, number]                      > [0, 500]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],                    > [2002, 2017]
  range: [number, number]                      > [0, 500]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;  > 15
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],                    > [2002, 2017]
  range: [number, number]                      > [0, 500]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;  > 15
  const sizeOfRange = maxRange - minRange;     > 500

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```
export function createLinearScale(
  domain: [number, number],                    > [2002, 2017]
  range: [number, number]                      > [0, 500]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;  > 15
  const sizeOfRange = maxRange - minRange;     > 500

  return function scale(value: number) {       > 2006
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],                                  > [2002, 2017]
  range: [number, number]                                    > [0, 500]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;                > 15
  const sizeOfRange = maxRange - minRange;                   > 500

  return function scale(value: number) {                     > 2006
    const positionInDomain = (value - minDomain) / sizeOfDomain; > 0.267

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```
export function createLinearScale(
  domain: [number, number],                                  > [2002, 2017]
  range: [number, number]                                    > [0, 500]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;                > 15
  const sizeOfRange = maxRange - minRange;                   > 500

  return function scale(value: number) {                     > 2006
    const positionInDomain = (value - minDomain) / sizeOfDomain; > 0.267

    return positionInDomain * sizeOfRange + minRange;        > 133.33
  };
}
```

```ts
export function createLinearScale(
  domain: [number, number],
  range: [number, number]
) {
  const [minDomain, maxDomain] = domain;
  const [minRange, maxRange] = range;
  const sizeOfDomain = maxDomain - minDomain;
  const sizeOfRange = maxRange - minRange;

  return function scale(value: number) {
    const positionInDomain = (value - minDomain) / sizeOfDomain;

    return positionInDomain * sizeOfRange + minRange;
  };
}
```

```
@Component({ selector: "app-sparkline-chart" })
export class SparklineChartComponent {
  computedPoints: Point[];

  @Input() set points(points: Point[]) {
    const bounds = getBounds(points);
    const scaleX = createLinearScale([bounds.minX, bounds.maxX], [10, 490]);
    const scaleY = createLinearScale([bounds.minY, bounds.maxY], [110, 10]);

    this.computedPoints = points.map(point => {
      return {
        x: scaleX(point.x),
        y: scaleY(point.y)
      };
    });
  }
}
```

```
@Component({ selector: "app-sparkline-chart" })
export class SparklineChartComponent {
  computedPoints: Point[];

  @Input() set points(points: Point[]) {
    const bounds = getBounds(points);
    const scaleX = createLinearScale([bounds.minX, bounds.maxX], [10, 490]);
    const scaleY = createLinearScale([bounds.minY, bounds.maxY], [110, 10]);

    this.computedPoints = points.map(point => {
      return {
        x: scaleX(point.x),
        y: scaleY(point.y)
      };
    });
  }
}
```

```
@Component({ selector: "app-sparkline-chart" })
export class SparklineChartComponent {
  computedPoints: Point[];

  @Input() set points(points: Point[]) {
    const bounds = getBounds(points);
    const scaleX = createLinearScale([bounds.minX, bounds.maxX], [10, 490]);
    const scaleY = createLinearScale([bounds.minY, bounds.maxY], [110, 10]);

    this.computedPoints = points.map(point => {
      return {
        x: scaleX(point.x),
        y: scaleY(point.y)
      };
    });
  }
}
```

```
@Component({ selector: "app-sparkline-chart" })
export class SparklineChartComponent {
  computedPoints: Point[];

  @Input() set points(points: Point[]) {
    const bounds = getBounds(points);
    const scaleX = createLinearScale([bounds.minX, bounds.maxX], [10, 490]);
    const scaleY = createLinearScale([bounds.minY, bounds.maxY], [110, 10]);

    this.computedPoints = points.map(point => {
      return {
        x: scaleX(point.x),
        y: scaleY(point.y)
      };
    });
  }
}
```

```
@Component({ selector: "app-sparkline-chart" })
export class SparklineChartComponent {
  computedPoints: Point[];

  @Input() set points(points: Point[]) {
    const bounds = getBounds(points);
    const scaleX = createLinearScale([bounds.minX, bounds.maxX], [10, 490]);
    const scaleY = createLinearScale([bounds.minY, bounds.maxY], [110, 10]);

    this.computedPoints = points.map(point => {
      return {
        x: scaleX(point.x),
        y: scaleY(point.y)
      };
    });
  }
}
```

Drawing the Dots

```
@Component({
  selector: "svg:g[app-sparkline-dots]",
  template: `
    <svg:circle
      *ngFor="let point of points"
      [attr.cx]="point.x"
      [attr.cy]="point.y"
    />
  `,
})
export class SparklineDotsComponent {
  @Input() points: Point[];
}
```

```
@Component({
  selector: "svg:g[app-sparkline-dots]",
  template: `
    <svg:circle
      *ngFor="let point of points"
      [attr.cx]="point.x"
      [attr.cy]="point.y"
    />
  `,
})
export class SparklineDotsComponent {
  @Input() points: Point[];
}
```

```
@Component({
  selector: "svg:g[app-sparkline-dots]",
  template: `
    <svg:circle
      *ngFor="let point of points"
      [attr.cx]="point.x"
      [attr.cy]="point.y"
    />
  `,
})
export class SparklineDotsComponent {
  @Input() points: Point[];
}
```

```
@Component({
  selector: "svg:g[app-sparkline-dots]",
  template: `
    <svg:circle
      *ngFor="let point of points"
      [attr.cx]="point.x"
      [attr.cy]="point.y"
    />
  `,
})
export class SparklineDotsComponent {
  @Input() points: Point[];
}
```

```
<svg width="500" height="120">
  <svg:g
    app-sparkline-dots
    [points]="computedPoints"
  ></svg:g>
</svg>
```


Movies By Year

Paths & Polygons with SVG

```
<svg:path
  d="M 72,0 L 144,25 L 135,120 L 72,154 L 10,120 L 0,25 Z"
></svg:path>
```

FN ...arguments FN ...arguments FN ...arguments

M 72,0 L 144,25 L 135,120 L 72,154 L 10,120 L 0,25 Z

```
M 72,0 L 144,25 L 135,120 L 72,154 L 10,120 L 0,25 Z
```

M 72,0 **L 144,25** L 135,120 L 72,154 L 10,120 L 0,25 Z

M 72,0 L 144,25 **L 135,120** L 72,154 L 10,120 L 0,25 Z

M 72,0 L 144,25 L 135,120 **L 72,154** L 10,120 L 0,25 Z

M 72,0 L 144,25 L 135,120 L 72,154 **L 10,120** L 0,25 Z

```
M 72,0 L 144,25 L 135,120 L 72,154 L 10,120 L 0,25 Z
```

```
M 72,0 L 144,25 L 135,120 L 72,154 L 10,120 L 0,25 Z
```

```
M 72,0 L 144,25 L 135,120 L 72,154 L 10,120 L 0,25 Z
```

```
function getPathCommandsForLine(points: Point[]) {
  const [firstPoint, ...restOfPoints] = points;
  const move = `M ${firstPoint.x},${firstPoint.y}`;
  const lines = restOfPoints.map(point => {
    return `L ${point.x},${point.y}`;
  });

  return `${move} ${lines.join(" ")}`;
}
```

```
function getPathCommandsForLine(points: Point[]) {
  const [firstPoint, ...restOfPoints] = points;
  const move = `M ${firstPoint.x},${firstPoint.y}`;
  const lines = restOfPoints.map(point => {
    return `L ${point.x},${point.y}`;
  });

  return `${move} ${lines.join(" ")}`;
}
```

```
function getPathCommandsForLine(points: Point[]) {
  const [firstPoint, ...restOfPoints] = points;
  const move = `M ${firstPoint.x},${firstPoint.y}`;
  const lines = restOfPoints.map(point => {
    return `L ${point.x},${point.y}`;
  });

  return `${move} ${lines.join(" ")}`;
}
```

```
function getPathCommandsForLine(points: Point[]) {
  const [firstPoint, ...restOfPoints] = points;
  const move = `M ${firstPoint.x},${firstPoint.y}`;
  const lines = restOfPoints.map(point => {
    return `L ${point.x},${point.y}`;
  });

  return `${move} ${lines.join(" ")}`;
}
```

```
function getPathCommandsForLine(points: Point[]) {
  const [firstPoint, ...restOfPoints] = points;
  const move = `M ${firstPoint.x},${firstPoint.y}`;
  const lines = restOfPoints.map(point => {
    return `L ${point.x},${point.y}`;
  });

  return `${move} ${lines.join(" ")}`;
}
```

```
function getPathCommandsForLine(points: Point[]) {
  const [firstPoint, ...restOfPoints] = points;
  const move = `M ${firstPoint.x},${firstPoint.y}`;
  const lines = restOfPoints.map(point => {
    return `L ${point.x},${point.y}`;
  });

  return `${move} ${lines.join(" ")}`;
}
```

```
@Component({
  selector: "svg:g[app-sparkline-line]",
  template: `<svg:path [attr.d]="d" />`,
})
export class SparklineLineComponent {
  d: string = "";

  @Input() set points(points: Point[]) {
    this.d = getPathCommandsForLine(points);
  }
}
```

```
@Component({
  selector: "svg:g[app-sparkline-line]",
  template: `<svg:path [attr.d]="d" />`,
})
export class SparklineLineComponent {
  d: string = "";

  @Input() set points(points: Point[]) {
    this.d = getPathCommandsForLine(points);
  }
}
```

```
@Component({
  selector: "svg:g[app-sparkline-line]",
  template: `<svg:path [attr.d]="d" />`,
})
export class SparklineLineComponent {
  d: string = "";

  @Input() set points(points: Point[]) {
    this.d = getPathCommandsForLine(points);
  }
}
```

```
@Component({
  selector: "svg:g[app-sparkline-line]",
  template: `<svg:path [attr.d]="d" />`,
})
export class SparklineLineComponent {
  d: string = "";

  @Input() set points(points: Point[]) {
    this.d = getPathCommandsForLine(points);
  }
}
```

```
<svg width="500" height="120">
  <svg:g
    app-sparkline-dots
    [points]="computedPoints"
  ></svg:g>
</svg>
```

```
<svg width="500" height="120">
  <svg:g
    app-sparkline-dots
    [points]="computedPoints"
  ></svg:g>
  <svg:g
    app-sparkline-line
    [points]="computedPoints"
  ></svg:g>
</svg>
```


Movies By Year

```
function getPathCommandsForPolygon(points: Point[]) {
  const commandsForLine = getPathCommandsForLine(points);
  const bounds = getBounds(points);
  const lineToBottomRight = `L ${bounds.maxX}, ${bounds.maxY}`;
  const lineToBottomLeft = `L ${bounds.minX}, ${bounds.maxY}`;

  return `${commandsForLine} ${lineToBottomRight} ${lineToBottomLeft} Z`;
}
```

```
function getPathCommandsForPolygon(points: Point[]) {
  const commandsForLine = getPathCommandsForLine(points);
  const bounds = getBounds(points);
  const lineToBottomRight = `L ${bounds.maxX}, ${bounds.maxY}`;
  const lineToBottomLeft = `L ${bounds.minX}, ${bounds.maxY}`;

  return `${commandsForLine} ${lineToBottomRight} ${lineToBottomLeft} Z`;
}
```

```
function getPathCommandsForPolygon(points: Point[]) {
  const commandsForLine = getPathCommandsForLine(points);
  const bounds = getBounds(points);
  const lineToBottomRight = `L ${bounds.maxX}, ${bounds.maxY}`;
  const lineToBottomLeft = `L ${bounds.minX}, ${bounds.maxY}`;

  return `${commandsForLine} ${lineToBottomRight} ${lineToBottomLeft} Z`;
}
```

```
function getPathCommandsForPolygon(points: Point[]) {
  const commandsForLine = getPathCommandsForLine(points);
  const bounds = getBounds(points);
  const lineToBottomRight = `L ${bounds.maxX}, ${bounds.maxY}`;
  const lineToBottomLeft = `L ${bounds.minX}, ${bounds.maxY}`;

  return `${commandsForLine} ${lineToBottomRight} ${lineToBottomLeft} Z`;
}
```

```
function getPathCommandsForPolygon(points: Point[]) {
  const commandsForLine = getPathCommandsForLine(points);
  const bounds = getBounds(points);
  const lineToBottomRight = `L ${bounds.maxX}, ${bounds.maxY}`;
  const lineToBottomLeft = `L ${bounds.minX}, ${bounds.maxY}`;

  return `${commandsForLine} ${lineToBottomRight} ${lineToBottomLeft} Z`;
}
```

```
function getPathCommandsForPolygon(points: Point[]) {
  const commandsForLine = getPathCommandsForLine(points);
  const bounds = getBounds(points);
  const lineToBottomRight = `L ${bounds.maxX}, ${bounds.maxY}`;
  const lineToBottomLeft = `L ${bounds.minX}, ${bounds.maxY}`;

  return `${commandsForLine} ${lineToBottomRight} ${lineToBottomLeft} Z`;
}
```

```
function getPathCommandsForPolygon(points: Point[]) {
  const commandsForLine = getPathCommandsForLine(points);
  const bounds = getBounds(points);
  const lineToBottomRight = `L ${bounds.maxX}, ${bounds.maxY}`;
  const lineToBottomLeft = `L ${bounds.minX}, ${bounds.maxY}`;

  return `${commandsForLine} ${lineToBottomRight} ${lineToBottomLeft} Z`;
}
```

```
@Component({
  selector: "svg:g[app-sparkline-area]",
  template: `
    <svg:path [attr.d]="d" />
  `
})
export class SparklineAreaComponent {
  d: string = "";

  @Input() set points(points: Point[]) {
    this.d = getPathCommandsForPolygon(points);
  }
}
```

```
@Component({
  selector: "svg:g[app-sparkline-area]",
  template: `
    <svg:path [attr.d]="d" />
  `
})
export class SparklineAreaComponent {
  d: string = "";

  @Input() set points(points: Point[]) {
    this.d = getPathCommandsForPolygon(points);
  }
}
```

```
@Component({
  selector: "svg:g[app-sparkline-area]",
  template: `
    <svg:path [attr.d]="d" />
  `
})
export class SparklineAreaComponent {
  d: string = "";

  @Input() set points(points: Point[]) {
    this.d = getPathCommandsForPolygon(points);
  }
}
```

```
@Component({
  selector: "svg:g[app-sparkline-area]",
  template: `
    <svg:path [attr.d]="d" />
  `
})
export class SparklineAreaComponent {
  d: string = "";

  @Input() set points(points: Point[]) {
    this.d = getPathCommandsForPolygon(points);
  }
}
```

```
<svg width="500" height="120">
  <svg:g
    app-sparkline-line
    [points]="computedPoints"
  ></svg:g>
  <svg:g
    app-sparkline-dots
    [points]="computedPoints"
  ></svg:g>
</svg>
```

```
<svg width="500" height="120">
  <svg:g
    app-sparkline-line
    [points]="computedPoints"
  ></svg:g>
  <svg:g
    app-sparkline-area
    [points]="computedPoints"
  ></svg:g>
  <svg:g
    app-sparkline-dots
    [points]="computedPoints"
  ></svg:g>
</svg>
```


Movies By Year


Movies By Year


Movies By Year

Tooltips

```
@Component({
  template: `
    <svg:circle
      *ngFor="let point of points"
      [attr.cx]="point.x"
      [attr.cy]="point.y"
    />
  `,
  styles: [`
    circle {
      r: 8px;
      fill: #51beff;
    }
  `]
})
export class SparklineDotsComponent { ... }
```

```
circle {
  r: 8px;
  fill: #51beff;
}
```

```
circle {
  r: 8px;
  fill: #51beff;
  opacity: 0;
  transition: opacity 350ms;
}

circle:hover {
  opacity: 1;
}
```


Movies By Year


Movies By Year

```ts
export interface Point {
  x: number;
  y: number;
}
```

```ts
export interface Point {
  x: number;
  y: number;
  tooltip: string;
}
```

```
this.movies.getMoviesByYear().pipe(
  map(movies =>
    movies.map(({ year, movies }) => {
      return {
        x: year,
        y: movies,
      };
    })
  )
);
```

```
this.movies.getMoviesByYear().pipe(
  map(movies =>
    movies.map(({ year, movies }) => {
      return {
        x: year,
        y: movies,
        tooltip: `${movies} movies in ${year}`
      };
    })
  )
);
```

```
<svg:circle
  *ngFor="let point of points"
  [attr.cx]="point.x"
  [attr.cy]="point.y"
/>
```

```
<svg:circle
  *ngFor="let point of points"
  [attr.cx]="point.x"
  [attr.cy]="point.y"
  [matTooltip]="point.tooltip"
/>
```


Movies By Year


Movies By Year

Goals for improved visualizations

Goals for improved visualizations

- **Build on top of existing skill sets**

Goals for improved visualizations

Build on top of existing skill sets

- **Endless amounts of customization**

Goals for improved visualizations

- Build on top of existing skill sets
- Endless amounts of customization
- Built to last well into the future

If you know Angular you can make visualizations

Download the slides & working example

github.com/MikeRyanDev/reactive-charts-angular-meetup-2021

Mike Ryan

@MikeRyanDev

Thank You!

github.com/MikeRyanDev/reactive-charts-angular-meetup-2021