Can we simplify charts?

Experience from a year on @mui/x-charts

Alexandre Fauquette
@alexfauquette

Background story

MUI Core

@mui/material-ui

Background story

MUI Core

@mui/material-ui

What are you looking for?

Background story

MUI Core

@mui/material-ui
@mui/base
@mui/joy

Background story

MUI Core

@mui/material-ui
@mui/base
@mui/joy

MUI Toolpad

mui.com/toolpad

Background story

MUI Core

@mui/material-ui
@mui/base
@mui/joy

MUI Toolpad

mui.com/toolpad

MUI X

@mui/x-data-grid
@mui/x-date-pickers
@mui/x-tree-view
@mui/x-charts

Background story

MUI Core

@mui/material-ui
@mui/base
@mui/joy

MUI Toolpad

mui.com/toolpad

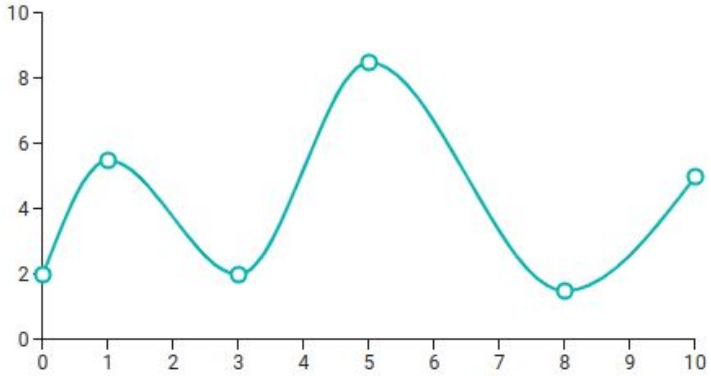
MUI X

@mui/x-data-grid
@mui/x-date-pickers
@mui/x-tree-view
@mui/x-charts

Good **docs** & similar **DevExp**

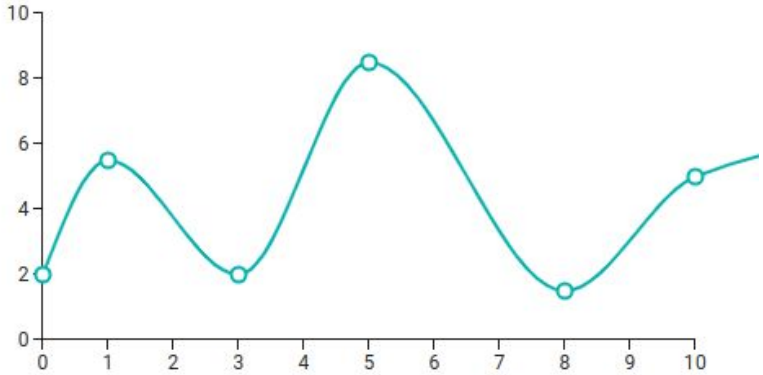
I have a dream

```
<ChartContainer width={300} height={200}>
  <XAxis min={0} max={10} />
  <YAxis />
  <Line data={yourData} />
</ChartContainer>;
```



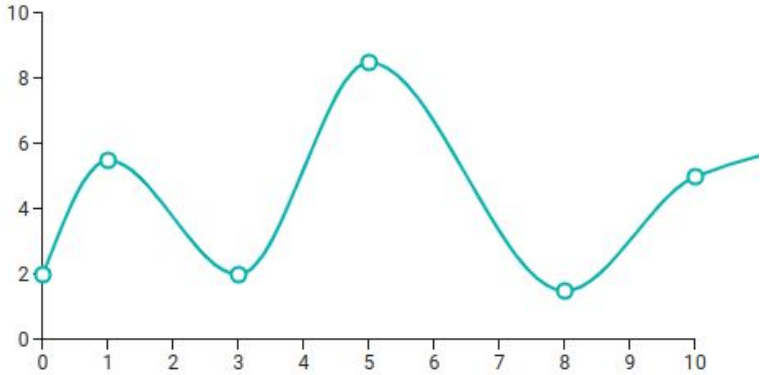
I have a dream

```
<ChartContainer width={300} height={200}>
  <XAxis min={0} max={10} />
  <YAxis />
  <Line data={moreData} />
</ChartContainer>;
```



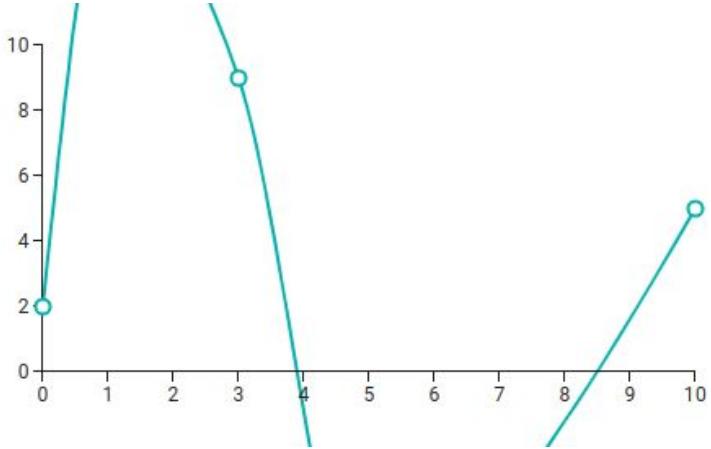
I have a dream

```
<ChartContainer width={300} height={200}>
  <XAxis min={0} max={10} />
  <YAxis />
  <Line data={moreData} />
</ChartContainer>;
```



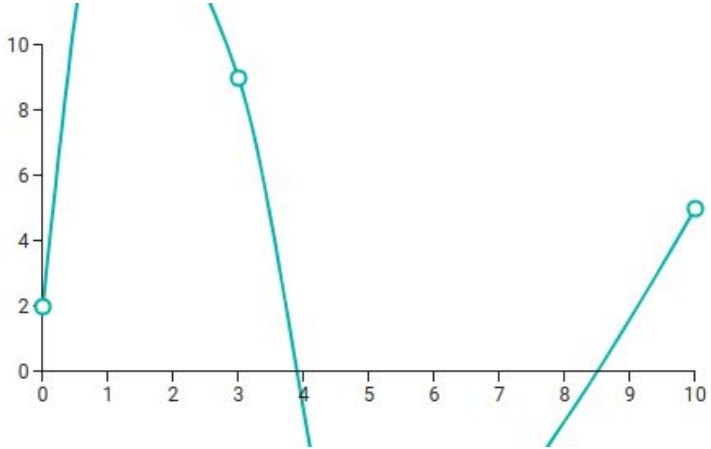
I have a dream

```
<ChartContainer width={300} height={200}>
  <XAxis min={0} max={10} />
  <YAxis />
  <Line data={largerData} />
</ChartContainer>;
```



I have a dream

```
<ChartContainer width={300} height={200}>
  <XAxis min={0} max={10} />
  <YAxis />
  <Line data={largerData} />
</ChartContainer>;
```



I have a ~~dream~~ an issue

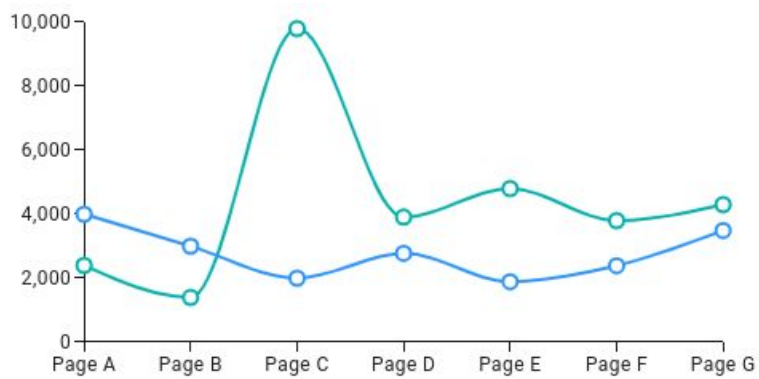
<Axis /> extremes  <Line /> data

Data format

```
yourData = [
  {id: 'id1', x: 0, y1: 5, y2: 7},
  {id: 'id2', x: 1, y1: 6, y2: 9},
  ...
]
```

Data format

```
yourData = [
  {id: 'id1', x: 0, y1: 5, y2: 7},
  {id: 'id2', x: 1, y1: 6, y2: 9},
  ...
]
```

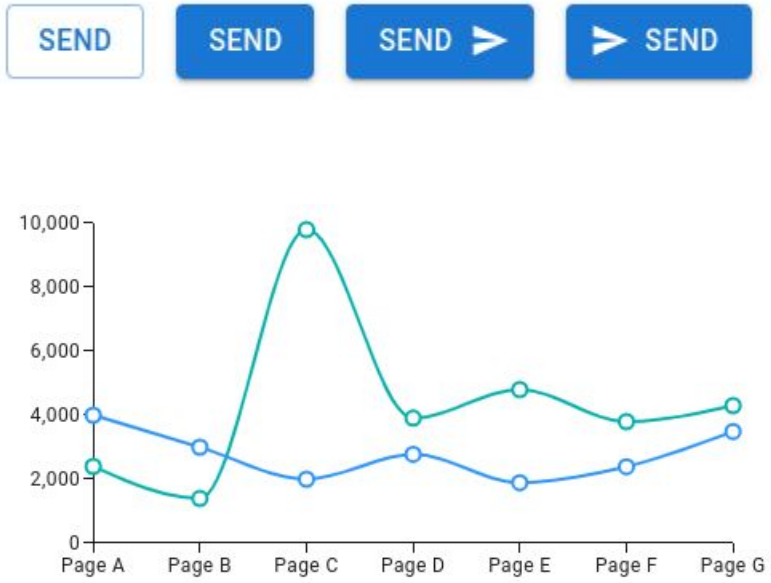


Data format

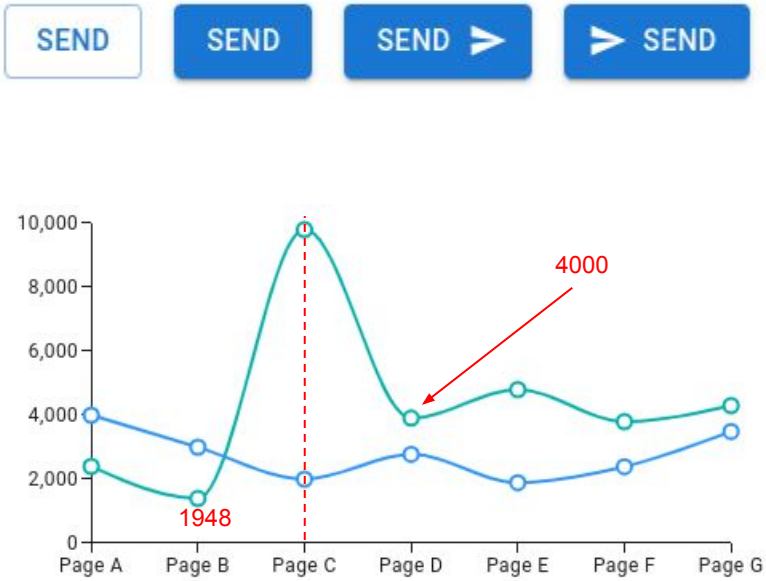
```
yourData = [
  {id: 'id1', pie1: 5,    pie2: 2 },
  {id: 'id2', pie1: 10,   pie2: 3 },
  {id: 'id3', pie1: 15,   pie2: 6 },
  {id: 'id4', pie1: null, pie2: 8 },
  {id: 'id5', pie1: null, pie2: 6 },
  {id: 'id6', pie1: null, pie2: 5 },
  {id: 'id7', pie1: null, pie2: 9 },
  ...
]
```

Customization

Customization



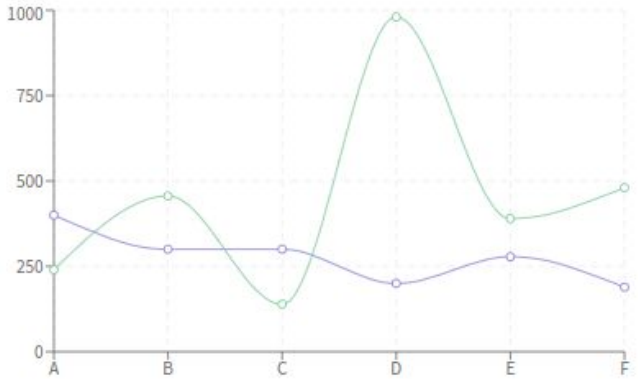
Customization



State of the art

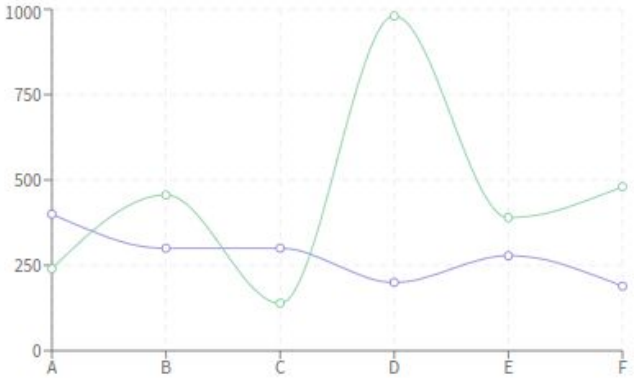
- Recharts: composition
- Nivo: single component
- Echarts: configuration object

Recharts




```
<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>
```

Recharts

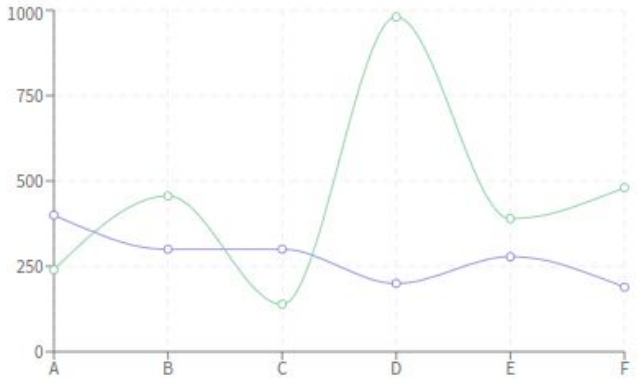



```
<LineChart width={500} height={300} data={data}>
  <XAxis dataKey="name"/>
  <YAxis/>
  <CartesianGrid stroke="#eee" strokeDasharray="5 5"/>
  <Line type="monotone" dataKey="uv" stroke="#8884d8" />
  <Line type="monotone" dataKey="pv" stroke="#82ca9d" />
</LineChart>
```

<LineChart>

What are the props of my children axes?

Recharts



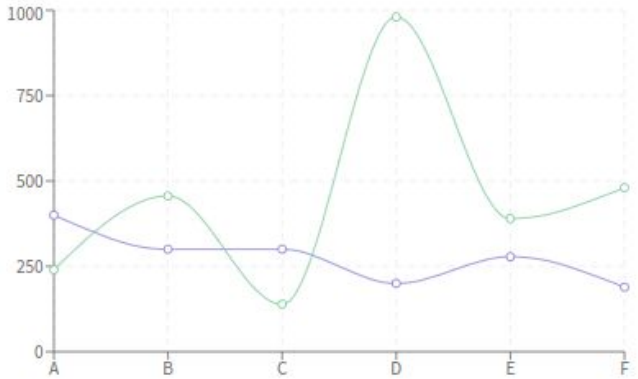

```
<LineChart width={500} height={300} data={data}>
  <XAxis dataKey="name"/>
  <YAxis/>
  <CartesianGrid stroke="#eee" strokeDasharray="5 5"/>
  <Line type="monotone" dataKey="uv" stroke="#8884d8" />
  <Line type="monotone" dataKey="pv" stroke="#82ca9d" />
</LineChart>
```

<LineChart>

What are the props of my children axes?

Which line are plotted?

Recharts




```
<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>
```

<LineChart>

What are the props of my children axes?

Which line are plotted?

Agregation

Nivo

```
<ResponsiveLine
  data={data}
  xScale={{
    type: 'linear',
    min: 0,
    max: 10
  }}
/>;
```

Nivo

1 chart type = 1 component + a set of option

Nivo

1 chart type = 1 component + a set of option

❌ mixing line and bar

❌ adding/modifying features

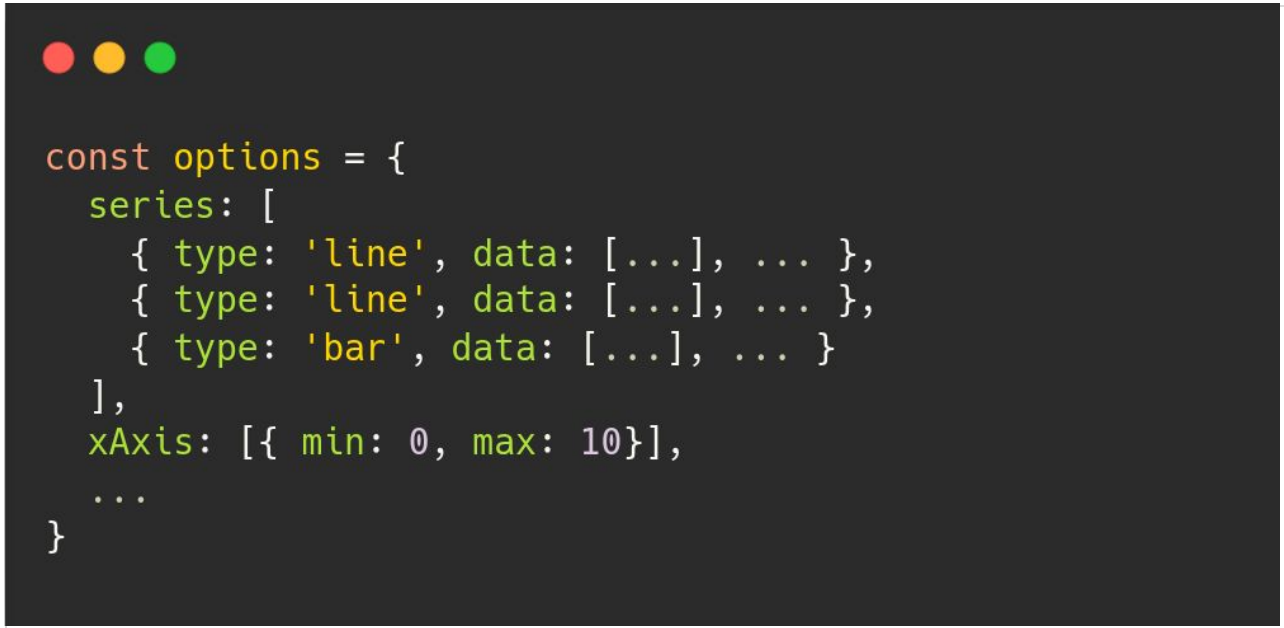
Echarts

```
import * as echarts from 'echarts';

var chartDom = document.getElementById('main');
var myChart = echarts.init(chartDom);

myChart.setOption(option)
```

Echarts

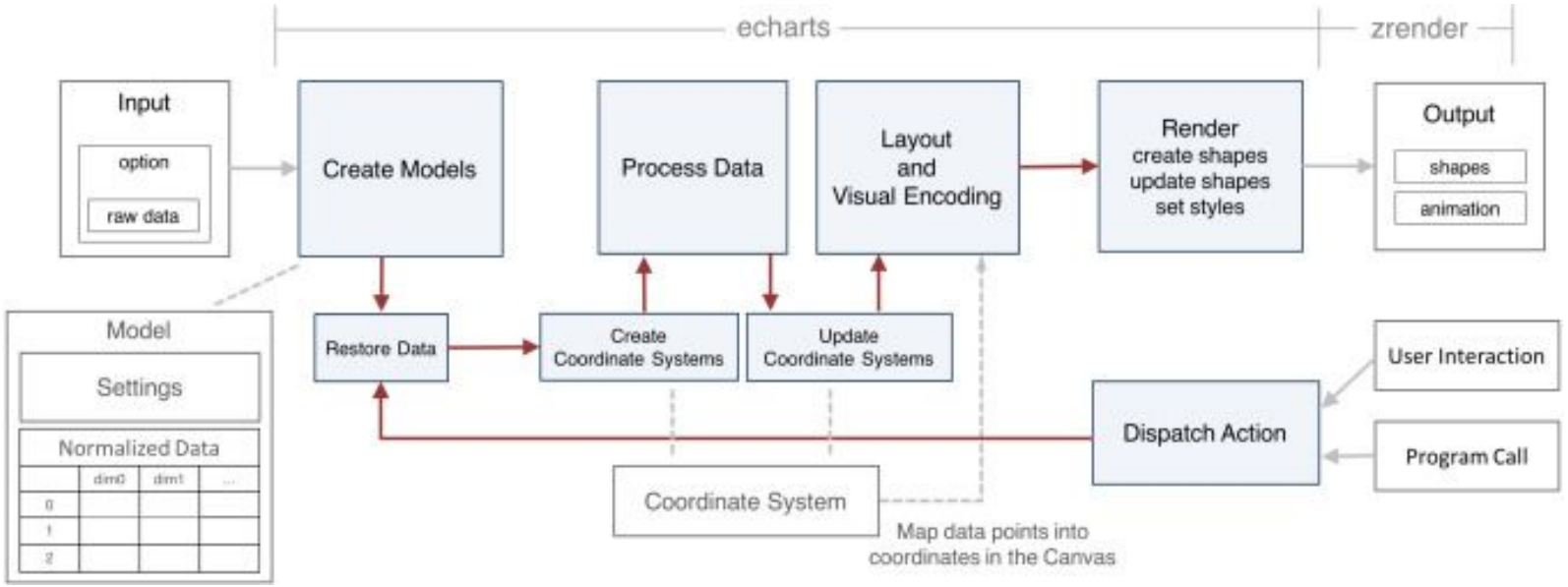
```
import * as echarts from 'echarts';

var chartDom = document.getElementById('main');
var myChart = echarts.init(chartDom);

myChart.setOption(option)
```

Echarts

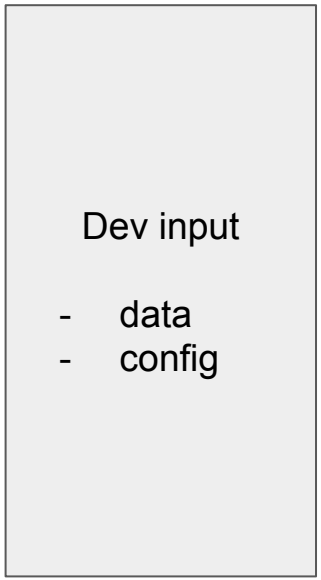


```
const options = {
  series: [
    { type: 'line', data: [...], ... },
    { type: 'line', data: [...], ... },
    { type: 'bar', data: [...], ... }
  ],
  xAxis: [{ min: 0, max: 10}],
  ...
}
```

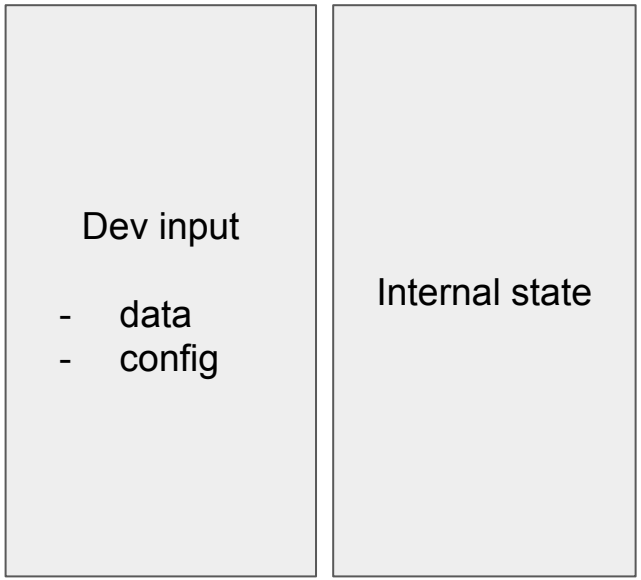
Echarts


echarts
zrender
Input
option
raw data
Create Models
Process Data
Layout
and
Visual Encoding
Render
create shapes
update shapes
set styles
Output
shapes
animation
Model
Settings
Normalized Data
dim0
dim1
...
0
1
2
Restore Data
Create
Coordinate Systems
Update
Coordinate Systems
User Interaction
Dispatch Action
Program Call
Coordinate System
Map data points into
coordinates in the Canvas

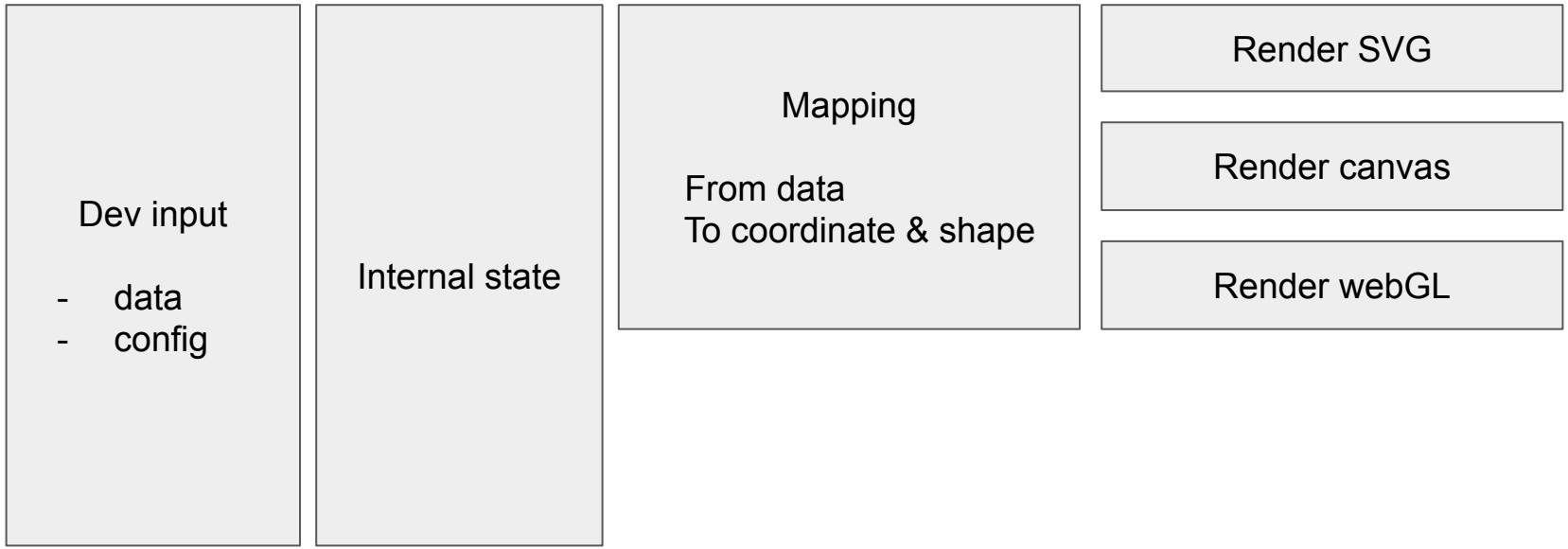
Echarts



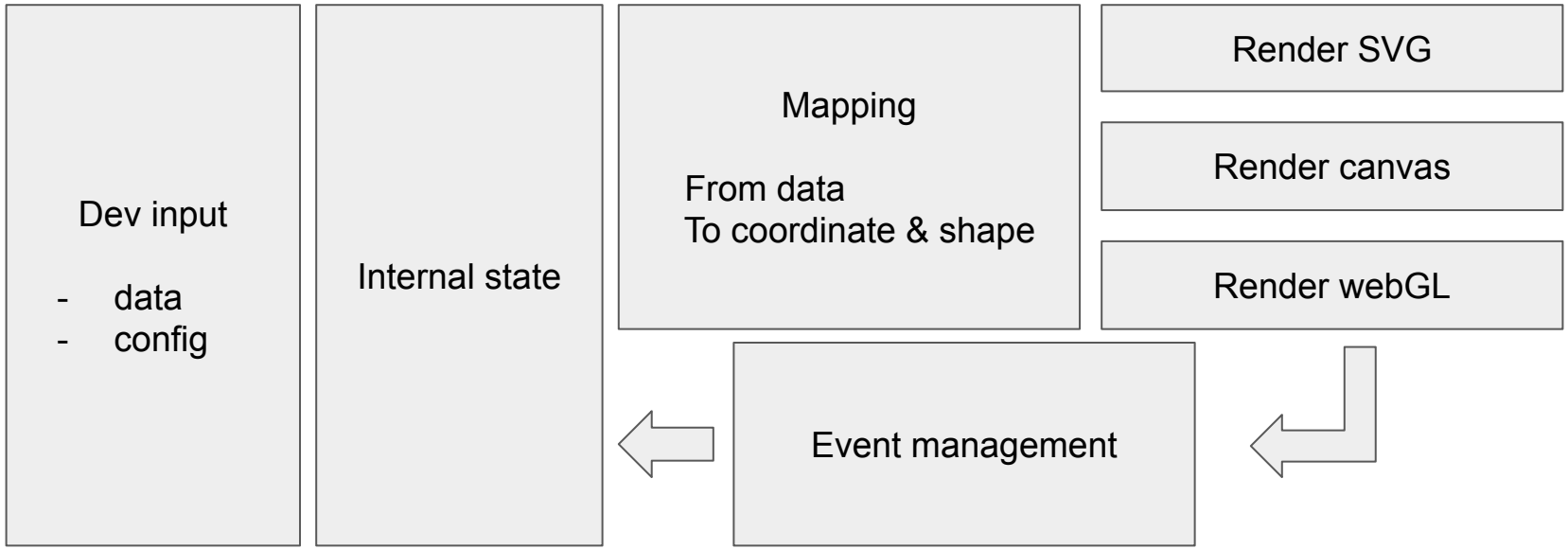
Echarts



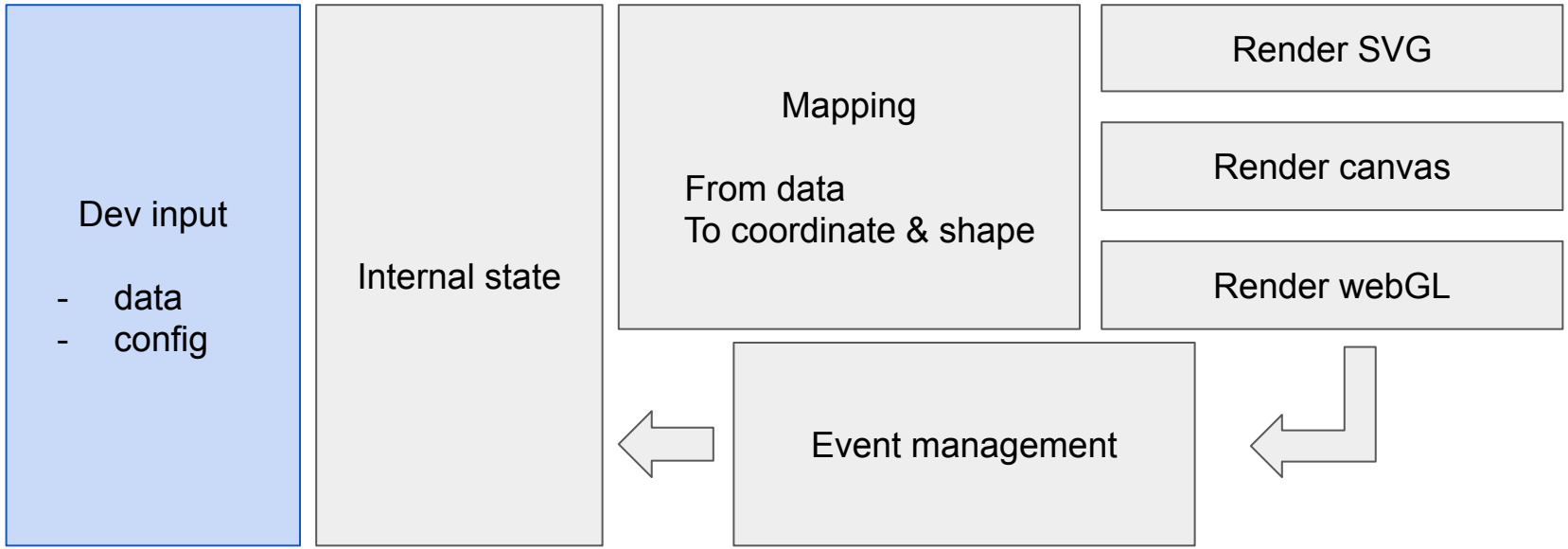
Echarts


Dev input
- data
- config
Internal state
Mapping
From data
To coordinate & shape
Render SVG
Render canvas
Render webGL

Echarts


Dev input
- data
- config
Internal state
Mapping
From data
To coordinate & shape
Render SVG
Render canvas
Render webGL
Event management

Echarts


Dev input
- data
- config
Internal state
Mapping
From data
To coordinate & shape
Render SVG
Render canvas
Render webGL
Event management

Dev Experience

	Composition	Single Component
Data sharing		
Adding Elements		

Dev Experience

	Composition	Single Component
Data sharing	❌	✅
Adding Elements		

Dev Experience

	Composition	Single Component
Data sharing	❌	✅
Adding Elements	✅	❌

Dev Experience

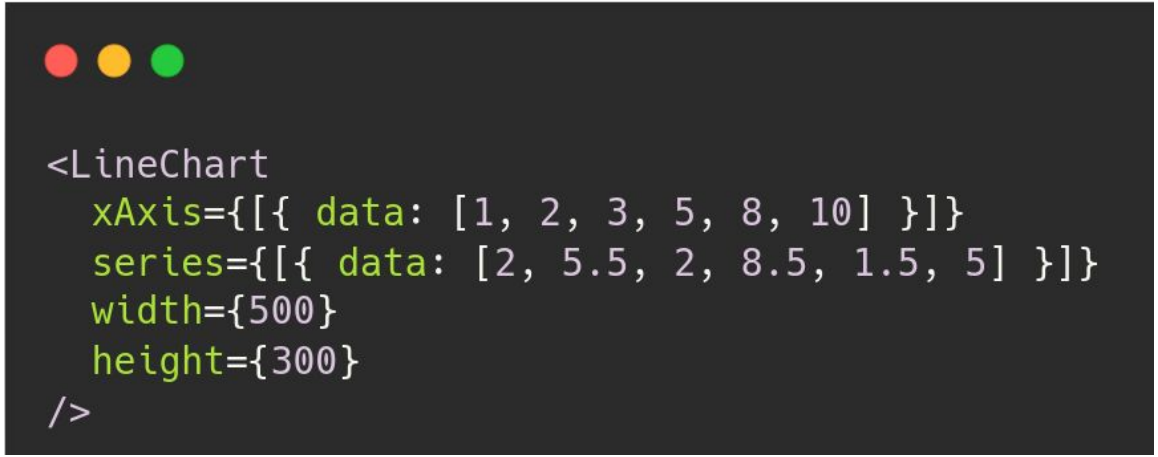
	Composition	Single Component
Data sharing	❌	✅
Data per series	✅	❌
Adding Elements	✅	❌

Conclusion

- Single components can do a lot 👍
- Lot can be done with config 👍
- Composition for customization 😬

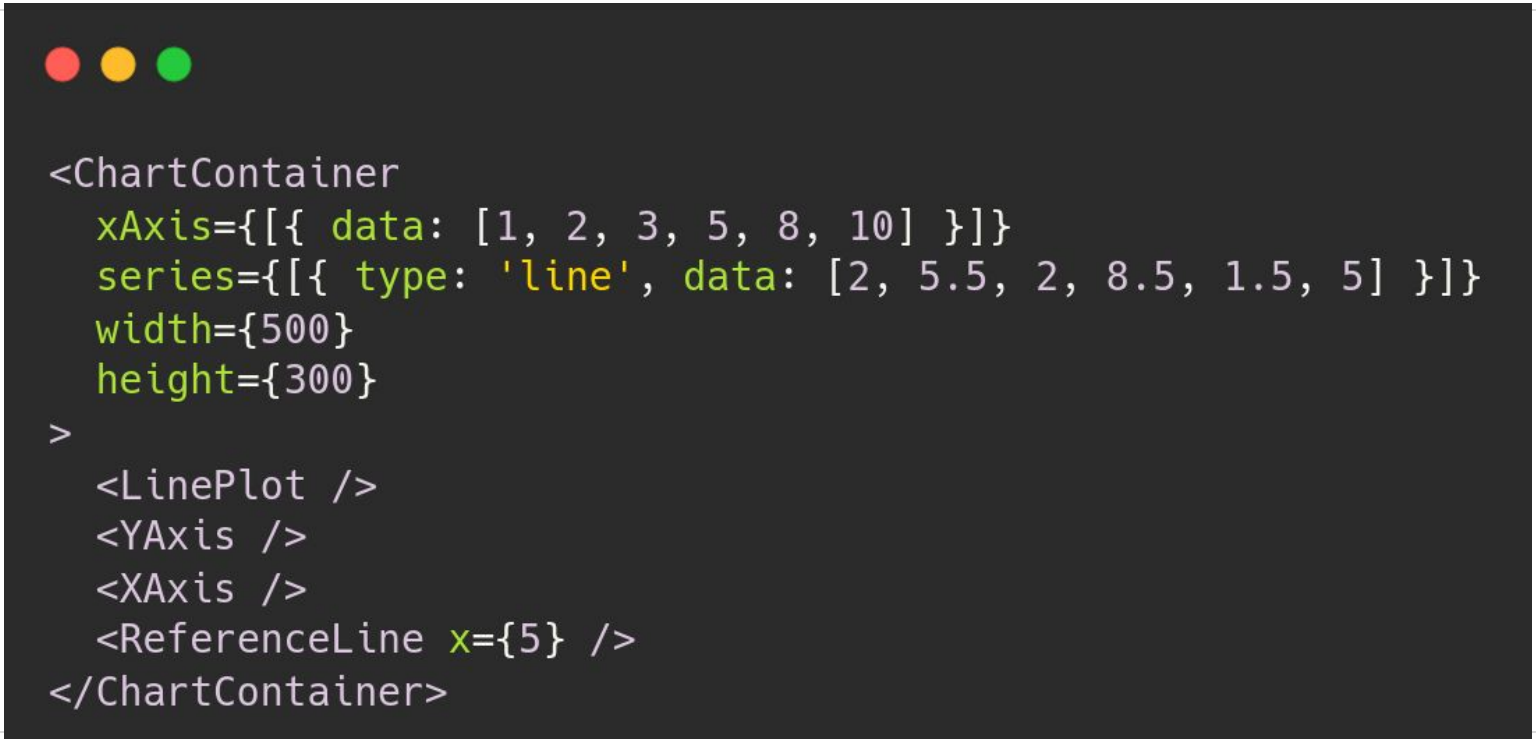
Proposal

Proposal



```
<LineChart
  xAxis={[{ data: [1, 2, 3, 5, 8, 10] }]}
  series={[{ data: [2, 5.5, 2, 8.5, 1.5, 5] }]}
  width={500}
  height={300}
/>
```

Proposal



```
<ChartContainer
  xAxis={[{ data: [1, 2, 3, 5, 8, 10] }]}
  series={[{ type: 'line', data: [2, 5.5, 2, 8.5, 1.5, 5] }]}
  width={500}
  height={300}
>
  <LinePlot />
  <YAxis />
  <XAxis />
  <ReferenceLine x={5} />
</ChartContainer>
```

Proposal

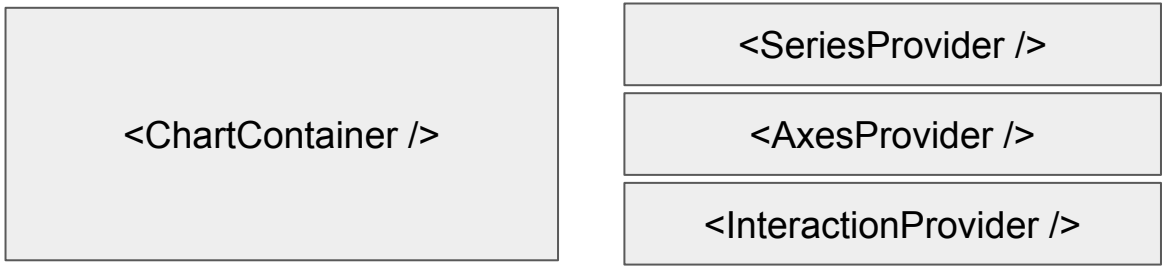
axes, series, event,
size, …

Proposal

axes, series, event,
size, …

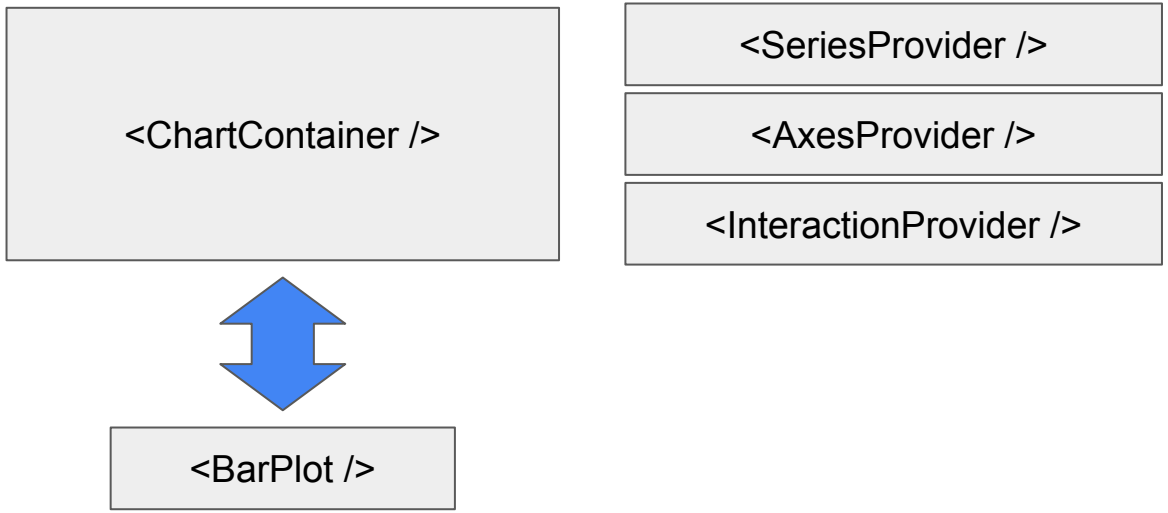
Proposal

axes, series, event, size, …



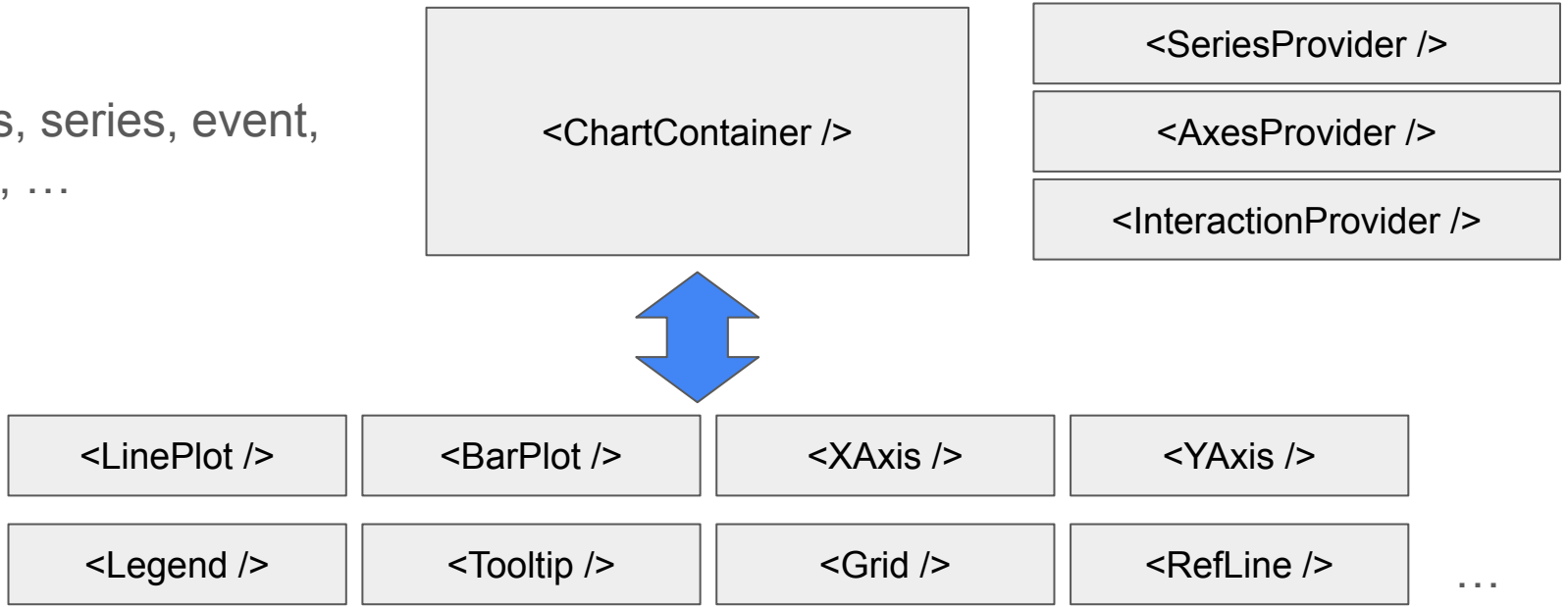
Proposal

axes, series, event, size, …



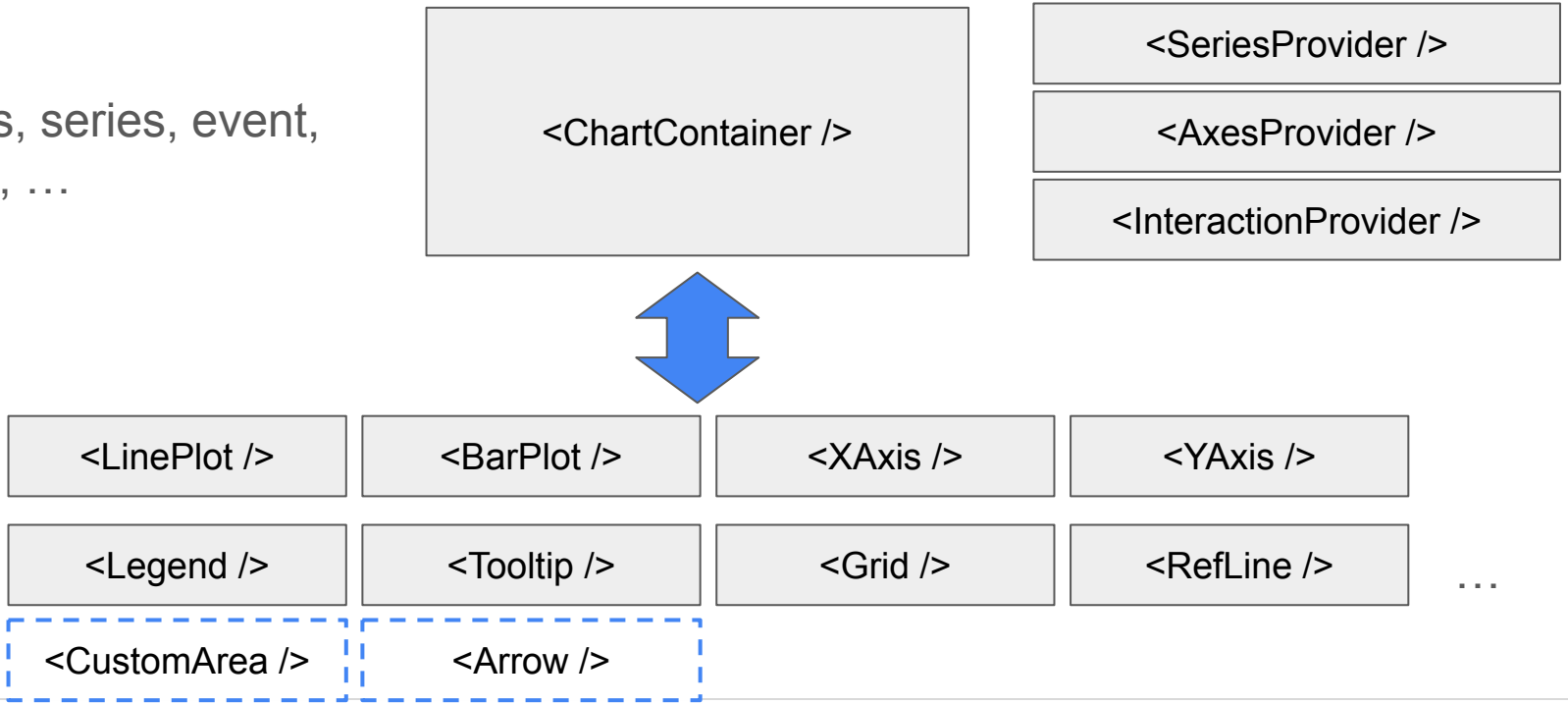
Proposal

axes, series, event, size, …



Proposal

axes, series, event, size, …



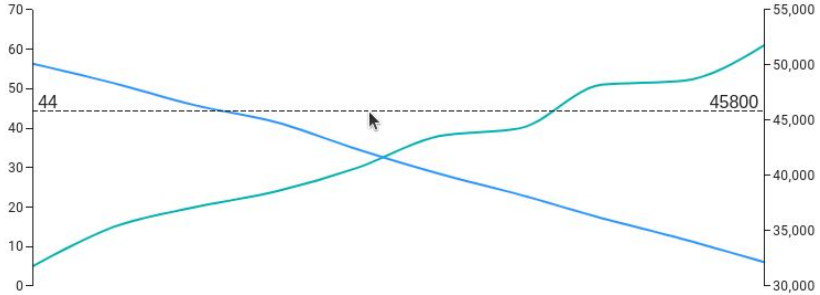
Conclusion

- single component for basics
- composition fallback for complex/custom
- config feeling is mandatory
- succeed to empower developers 🚀

Demo

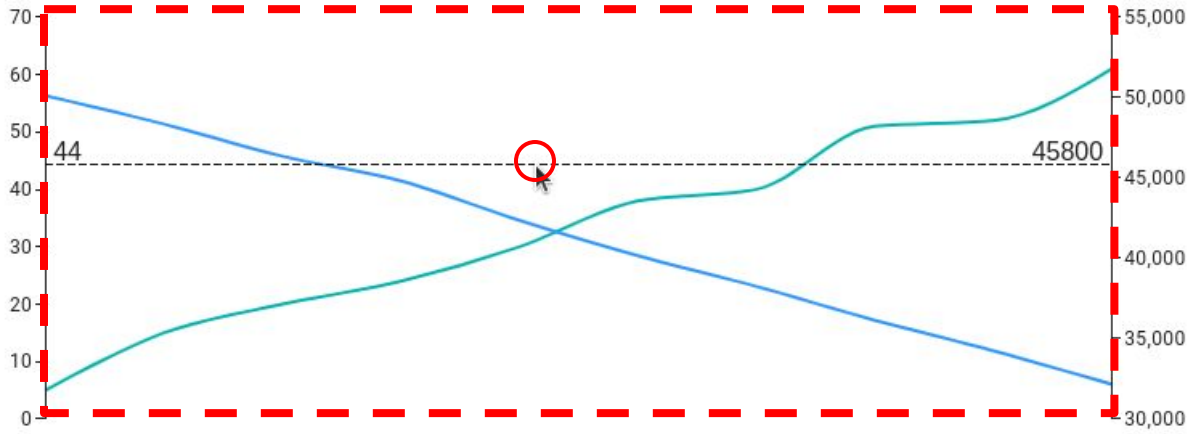
```
<ChartContainer
  ref={svgRef}
>
  ...
  <ValueHighlight svgRef={svgRef} />
</ChartContainer>
```

Demo

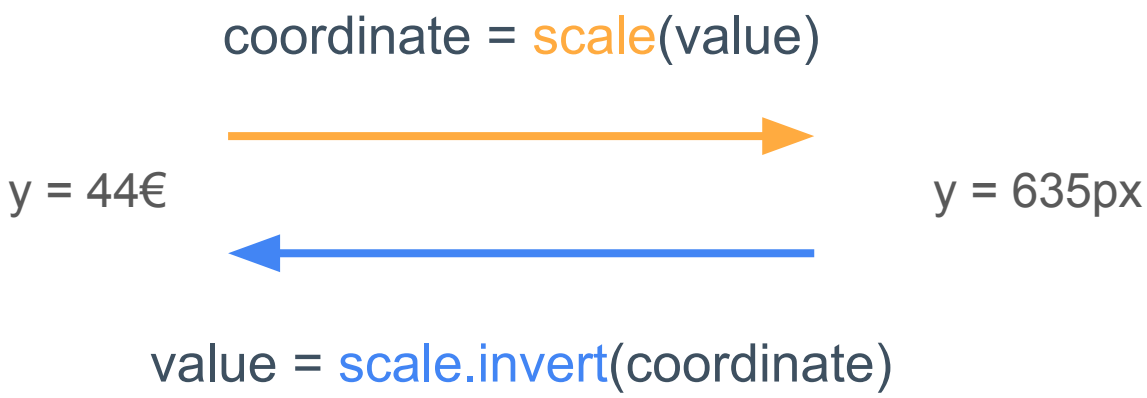



```
<ChartContainer
  ref={svgRef}
>
  ...
  <ValueHighlight svgRef={svgRef} />
</ChartContainer>
```

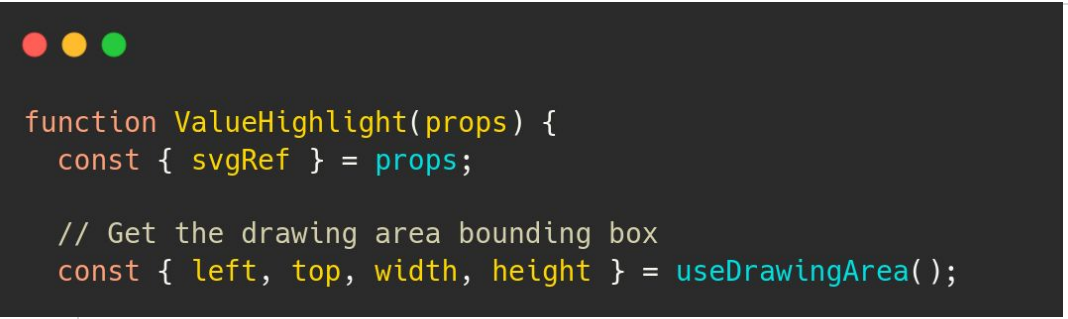
Demo


70
60
50
40
30
20
10
0
55,000
50,000
45,000
40,000
35,000
30,000
44
45800

Demo

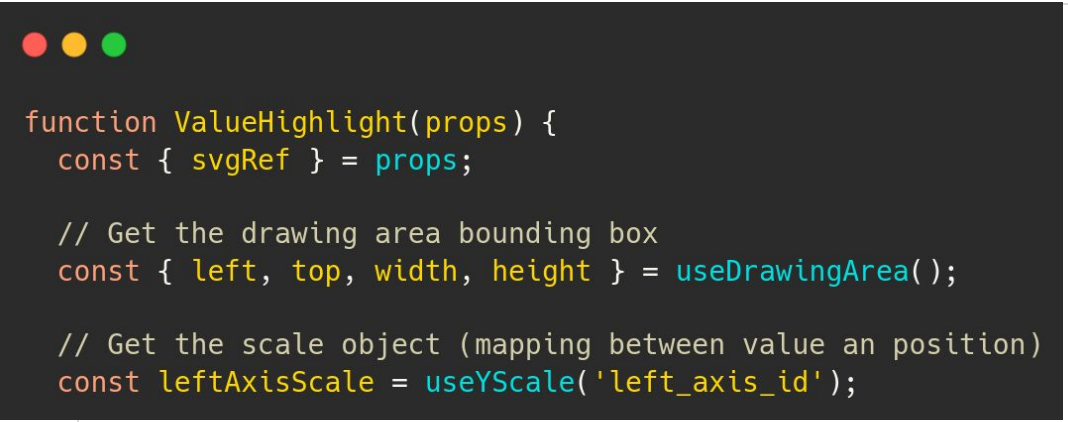

coordinate = scale(value)
y = 44€
y = 635px
value = scale.invert(coordinate)

Demo



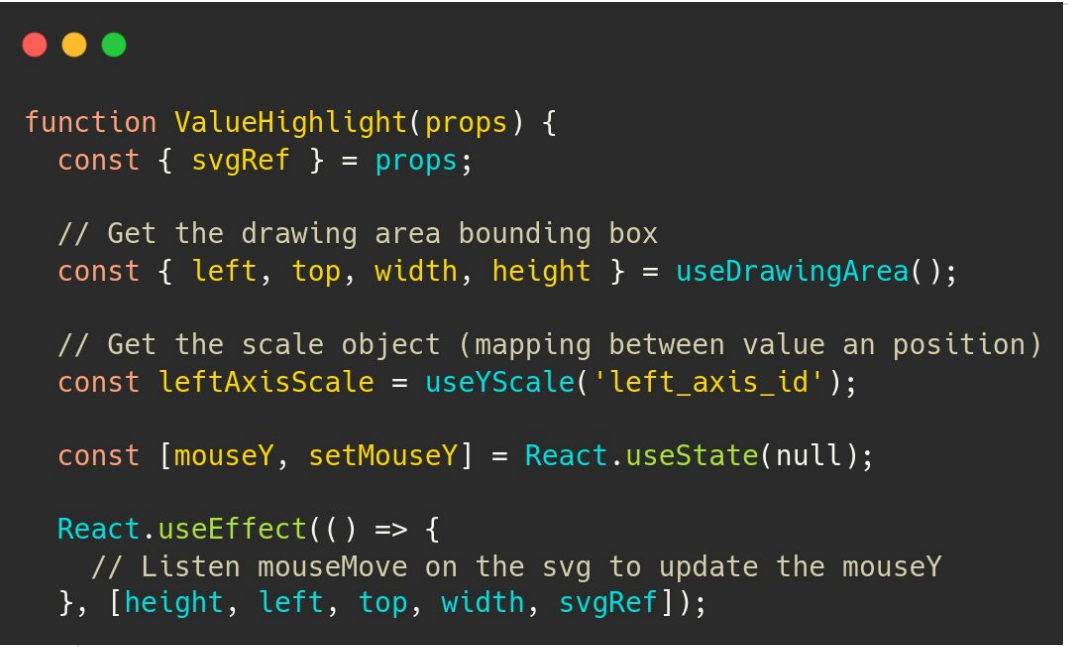
```
function ValueHighlight(props) {
  const { svgRef } = props;

  // Get the drawing area bounding box
  const { left, top, width, height } = useDrawingArea();
```

Demo



```
function ValueHighlight(props) {
  const { svgRef } = props;

  // Get the drawing area bounding box
  const { left, top, width, height } = useDrawingArea();

  // Get the scale object (mapping between value an position)
  const leftAxisScale = useYScale('left_axis_id');
```

Demo



```
function ValueHighlight(props) {
  const { svgRef } = props;

  // Get the drawing area bounding box
  const { left, top, width, height } = useDrawingArea();

  // Get the scale object (mapping between value an position)
  const leftAxisScale = useYScale('left_axis_id');

  const [mouseY, setMouseY] = React.useState(null);

  React.useEffect(() => {
    // Listen mouseMove on the svg to update the mouseY
  }, [height, left, top, width, svgRef]);
```

Demo

```
function ValueHighlight(props) {
  const { svgRef } = props;

  // Get the drawing area bounding box
  const { left, top, width, height } = useDrawingArea();

  // Get the scale object (mapping between value an position)
  const leftAxisScale = useYScale('left_axis_id');

  const [mouseY, setMouseY] = React.useState(null);

  React.useEffect(() => {
    // Listen mouseMove on the svg to update the mouseY
  }, [height, left, top, width, svgRef]);
```

```
  if(mouseY === null){
    return null
  }
```

Demo

```
function ValueHighlight(props) {
  const { svgRef } = props;

  // Get the drawing area bounding box
  const { left, top, width, height } = useDrawingArea();

  // Get the scale object (mapping between value an position)
  const leftAxisScale = useYScale('left_axis_id');

  const [mouseY, setMouseY] = React.useState(null);

  React.useEffect(() => {
    // Listen mouseMove on the svg to update the mouseY
  }, [height, left, top, width, svgRef]);
```

```
  if(mouseY === null){
    return null
  }
  return <g>
    <path d={`M ${left} ${mouseY} l ${width} 0`} />
    <text
      x={left + 5}
      y={mouseY}
      textAnchor="start"
      dominantBaseline="text-after-edge"
    >
      {leftAxisScale.invert(mouseY).toFixed(0)}
    </text>
  </g>
}
```

Demo

Questions?

Now

alexandre@mui.com

Alexandre Fauquette

Don’t hesitate to send feedback :)