Data Visualization

INF552 - 2023 - Session 05 - exercices
Introduction to D3 animations

emmanuel.pietriga@inria.fr

olivier.gladin@inria.fr

D3 updates & transitions

Binding new values…

```
d3.selectAll(“circle")
  .data(arrayWithInitialValues);

//...

d3.selectAll("circle")
  .data(arrayWithNewValues);
```

… is necessary but not sufficient. Visual encodings that depend on those values need to be updated:

```
d3.selectAll("circle")
  .data(arrayWithInitialValues)
  .enter()
  .append(“circle")
  .attr(“cx", 200)
  .attr("cy", (d,j) => (j*42))
  .attr("r", 10)
  .attr("fill", (d) => (someScale(d)));

//...

d3.selectAll("circle")
  .data(arrayWithNewValues)
  .attr("cy", (d,j) => (j*42))
  .attr("fill", (d) => (someScale(d)));
```

D3 updates & transitions

Animating the above changes is as simple as calling `.transition()` using the chained syntax:

```
d3.selectAll("circle")
  .data(arrayWithNewValues)
  .transition()
  .attr("fill", (d) => (someScale(d)));
```

https://github.com/d3/d3-transition

D3 does all the interpolation work, and lets us adjust many parameters:

- duration:

```
d3.selectAll(“circle")
  .data(arrayWithNewValues)
  .transition()
  .duration(1000) // in milliseconds
  .attr("fill", (d) => (someScale(d)));
```

- pacing function:

```
d3.selectAll(“circle")
  .data(arrayWithNewValues)
  .transition()
  .ease(d3.easeExpInOut)
  .attr("fill", (d) => (someScale(d)));
```

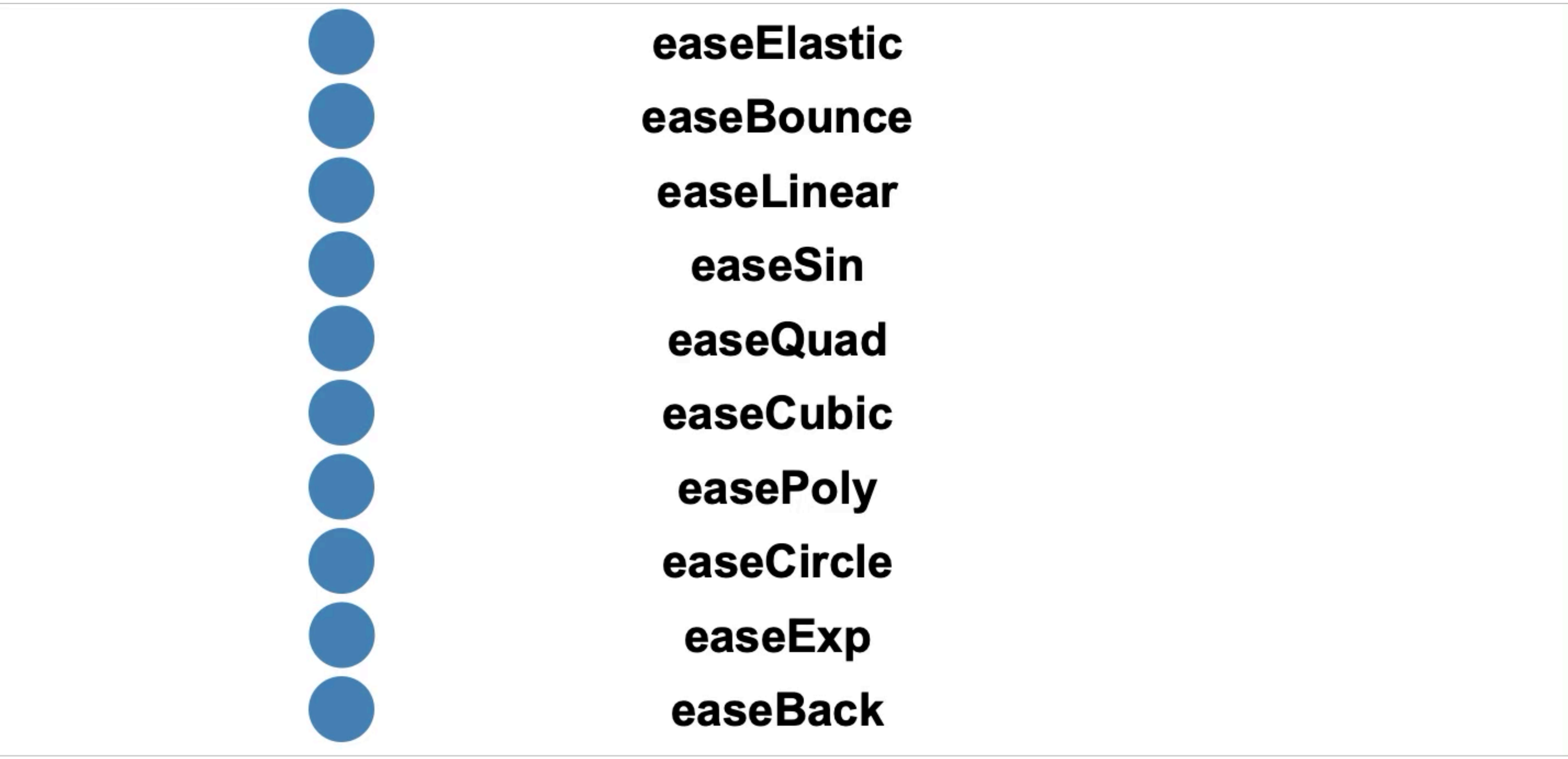
https://github.com/d3/d3-ease

- delay:

```
d3.selectAll(“circle")
  .data(arrayWithNewValues)
  .transition()
  .delay((d,i) => (i*20)) // in milliseconds
  .attr("fill", (d) => (someScale(d)));
```

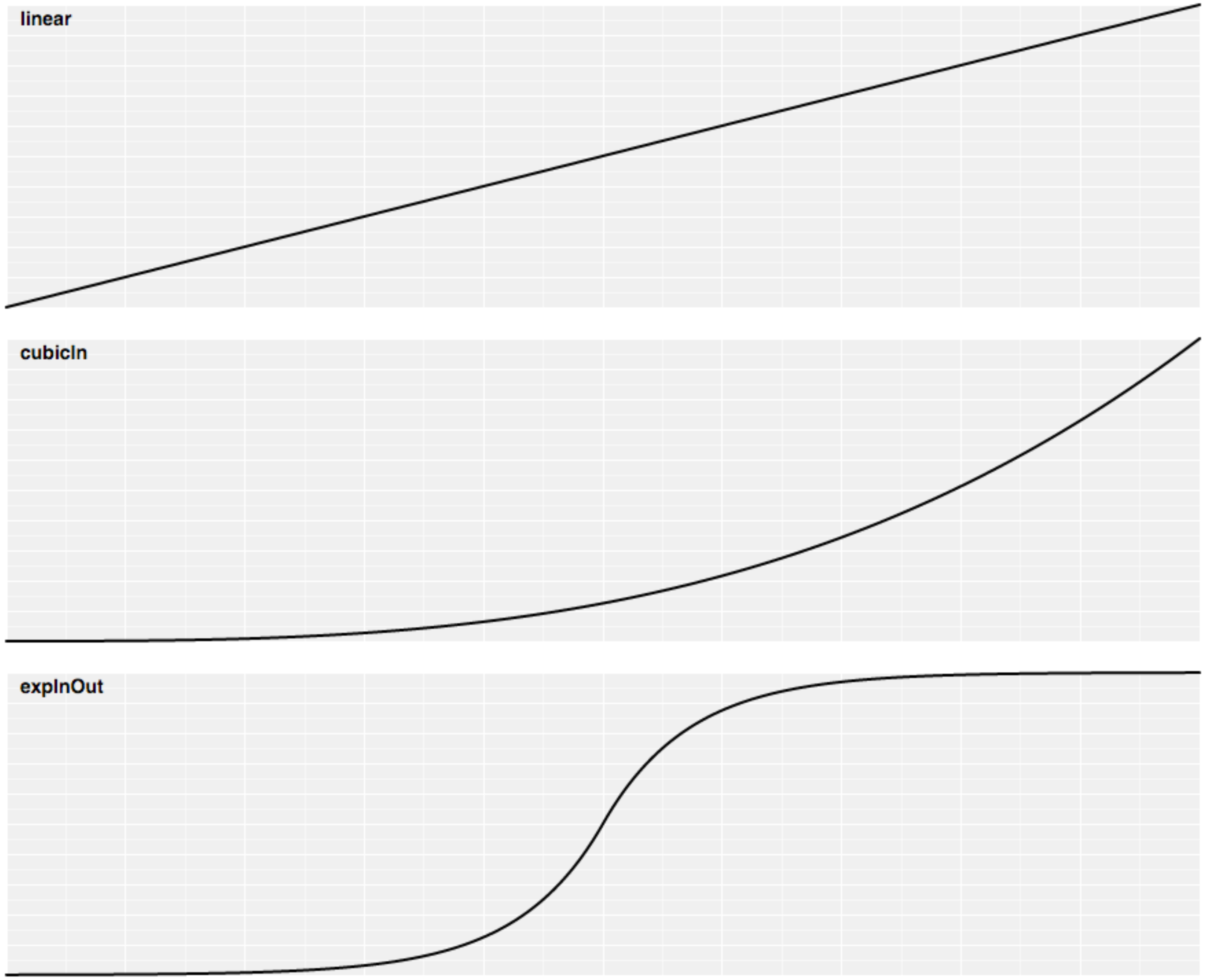
https://github.com/d3/d3-transition#transition_delay

Pacing functions



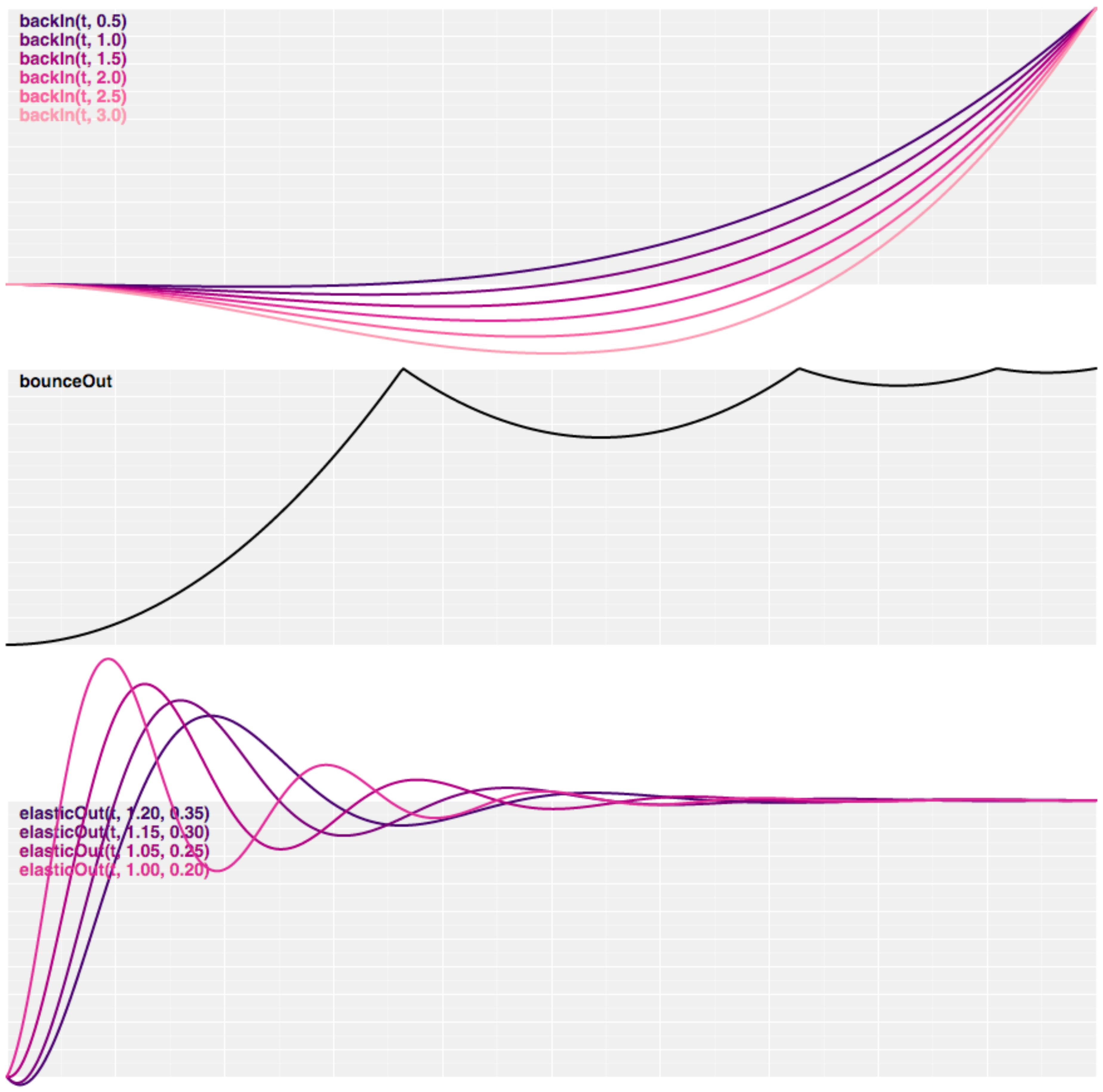
https://bl.ocks.org/d3noob/1ea51d03775b9650e8dfd03474e202fe

Pacing functions



[Image source: https://github.com/d3/d3-ease]

Pacing functions



[Image source: https://github.com/d3/d3-ease]

D3 updates & transitions

Animations do not necessarily depend on changes to the bound data.

It is possible to animate any elements in the DOM, even if those elements are not bound to data.

Example translating all circles in an SVG canvas out of the viewport, and fading them out in response to a simple mouse click event:

```
// make all circles move out of screen vertically,
// going outside the viewport in y=1600
d3.select("svg").on("mousedown", function(){
    d3.selectAll("circle")
      .transition()
      .duration(500)  // in 500ms
      .attr("transform", "translate(0,1600)")
      .attr("opacity", 0);
  });
```

D3 updates & transitions

D3 lets you chain transitions on the same graphical element.

Example:

```
d3.select("circle#foo")
  .transition()
  .duration(1000)
  .attr("cx", 200)
  .attr("cy", 100)
  .transition()
  .duration(500)
  .attr("fill", "red")
  .transition()
  .duration(2000)
  .attr("r", 40);
```

The above `<circle>`

- moves to `(200,100)` in 1 second;
- then changes color to `red` in 0.5 second;
- then gets resized to `40px` in 2 seconds.

(The whole animation lasts 3.5 seconds.)

D3 updates & transitions

D3 lets you execute code right before/after transitions:

- `on(…)` triggers code at the `start` or `end` of individual transitions (for each datum), or when the transition is interrupted (`interrupt`):

```
d3.selectAll("circle")
  .data(arrayWithNewValues)
  .transition()
  .on("start", function(d,i){
      console.log("About to start transition for datum "+d+" bound to mark:");
      console.log(d3.select(this));
  })
  .on(“end", function(d,i){
      console.log("Just finished transition for datum "+d+" bound to mark:");
      console.log(d3.select(this));
  })
  .attr("fill", (d) => (someScale(d)));
```

`d,i` return the usual suspects

`d3.select(this)` selects the graphical mark associated with `d`

https://github.com/d3/d3-transition#transition_on

D3 updates & transitions

Triggering a callback after *each* element has finished transitioning:

```
d3.selectAll("circle")
  .data(someData)
  .transition()
  .attr("cx", ...)
  .on("end", function(d,i){
    // triggered for each d in someData
  });
```

Triggering a callback after *all* elements have finished transitioning:

```
d3.selectAll("circle")
  .data(someData)
  .transition()
  .attr("cx", ...)
  .end().then(() => (/* triggered only once, after all d's in someData have finished 
transitioning*/));
```

https://d3js.org/d3-transition/control-flow

D3 updates & transitions

So far we have looked at the simple case of updating an existing collection of items.

In many cases, we want to also animate transitions where the collection varies in the count of items: some items may be added, some items may be removed.

D3 provides us with specific selectors to handle the particular elements:

- `d3.selectAll(…).data(…).enter()`: selects items that are not yet bound

```
d3.selectAll("circle")
  .data(arrayWithInitialValues)
  .enter()
  .append(“circle")
  .attr(“cx", 200)
  .attr("cy", (d,j) => (j*42))
  .attr("r", 10)
  .attr("fill", (d) => (someScale(d)));
```

- `d3.selectAll(…).data(…).exit()`: selects items that are no longer bound

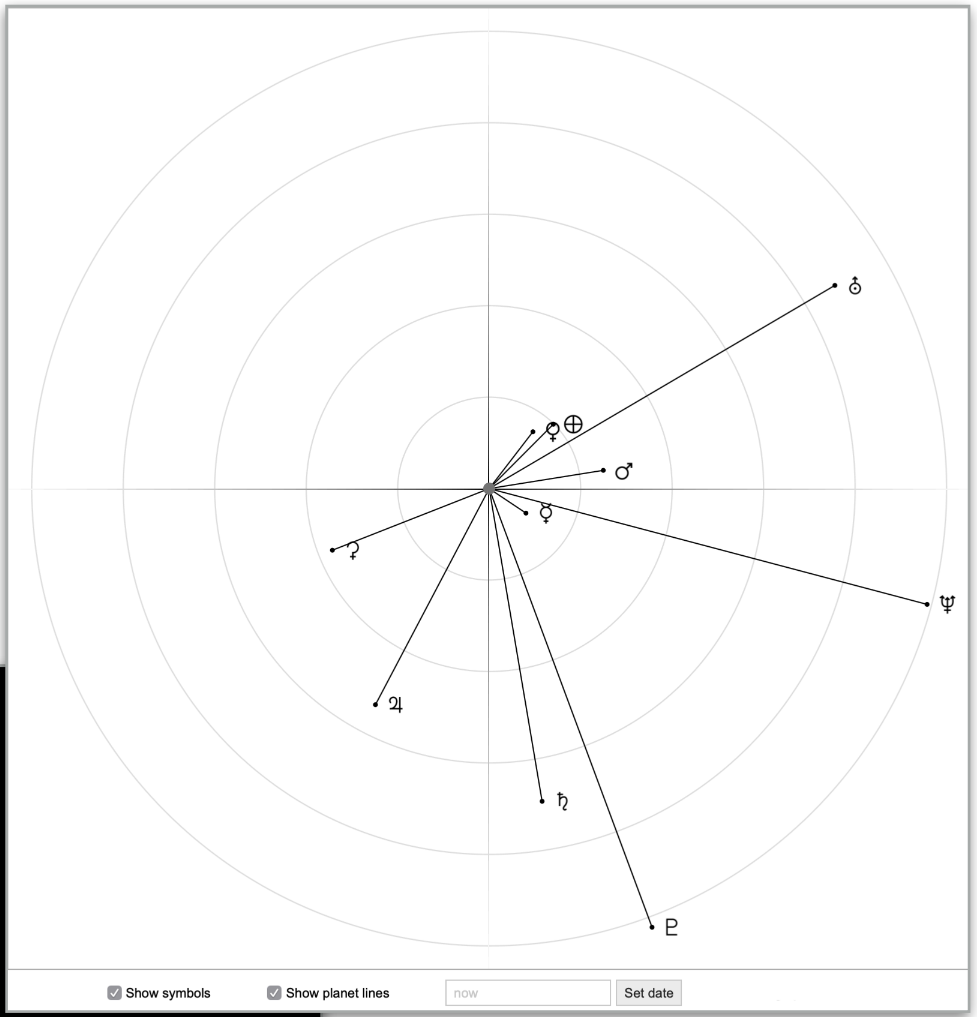
```
d3.selectAll(“circle")
  .data(smallerArray)
  .exit()
  .transition()
  .duration(1000)
  .attr("opacity", 0)
  .remove(); ←
```

actually removes the element from the DOM

D3 updates & transitions

D3 also lets you specify your own interpolation method:

http://www.sanctuaryproject.eu/calendar/



```
.attrTween("cx", function(d,i){
    return function(t){   ←――――――――――――――   t ∈ [0,1]
        var ir = prevCoords[d.sso.name][0];
        var ita = prevCoords[d.sso.name][1];
        var fr = getRadius(d.Hx, d.Hy);
        var fta = getAngle(d.Hx, d.Hy);
        var r = (fr-ir) * t + ir;
        // rotate by smallest angle between fta-ita and 2*PI-(fta-ita)
        var ta = Math.atan2(Math.sin(fta-ita), Math.cos(fta-ita)) * t + ita;
        return (ctx.cScale(ssScale(r))) * Math.cos(ta);
    }
})
.attrTween("cy", function(d,i){
    return function(t){   ←――――――――――――――   t ∈ [0,1]
        var ir = prevCoords[d.sso.name][0];
        var ita = prevCoords[d.sso.name][1];
        var fr = getRadius(d.Hx, d.Hy);
        var fta = getAngle(d.Hx, d.Hy);
        var r = (fr-ir) * t + ir;
        // rotate by smallest angle between fta-ita and 2*PI-(fta-ita)
        var ta = Math.atan2(Math.sin(fta-ita), Math.cos(fta-ita)) * t + ita;
        return (ctx.cScale(ssScale(r))) * Math.sin(ta);
    }
})
```