DSCI 554 LECTURE 6

PRE-ATTENTIVE FEATURES, INTERACTIVE VISUALIZATIONS WITH D3

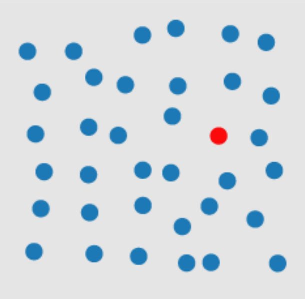
Dr. Luciano Nocera

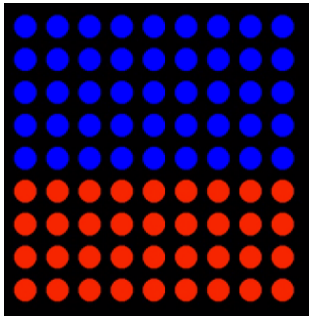
USC Viterbi
School of Engineering
Integrated Media Systems Center

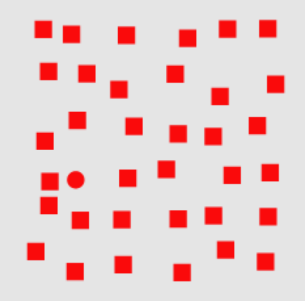
OUTLINE

- Pre-attentive features
- Continuity of visual queries
- Interactive visualizations with D3

PREATTENTIVE PROCESSING [HEALEY 1996]

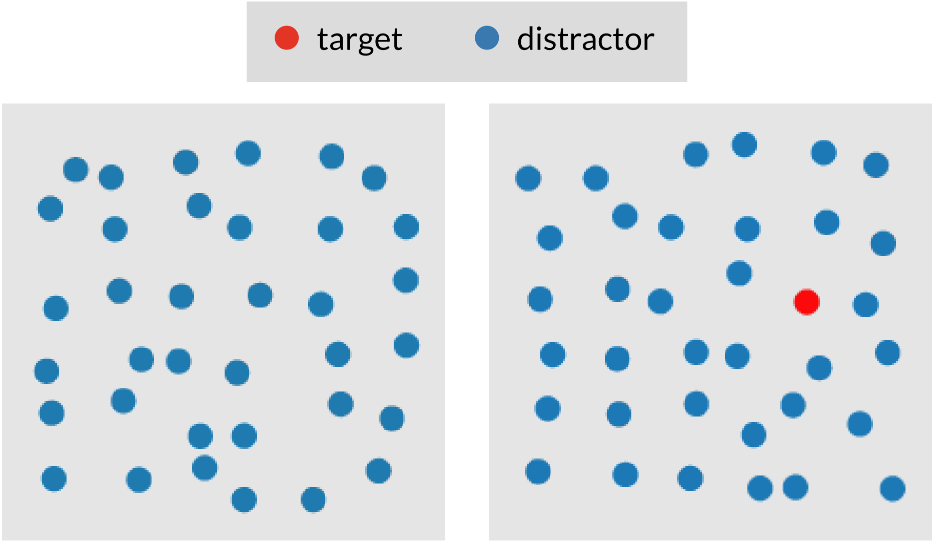






- Universal property
- Uses information from a single glimpse
- $< 200ms$ on large multi-element displays
- Does not involve attention
- Independent of:
  - Practice
  - Familiarity with the features
  - Number of distractors
- Can help present information at a glance

TARGETS, DISTRACTORS & VISUAL FEATURES

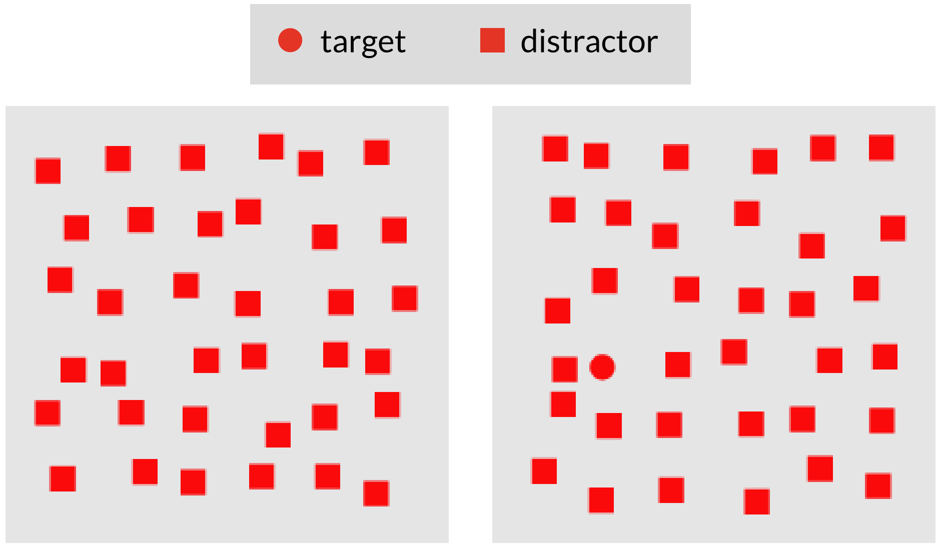


target absent

target present

Visual features	target	distractor
shape =	●	●
color ≠	🟥	🟦

FEATURE INTERFERENCE



target absent

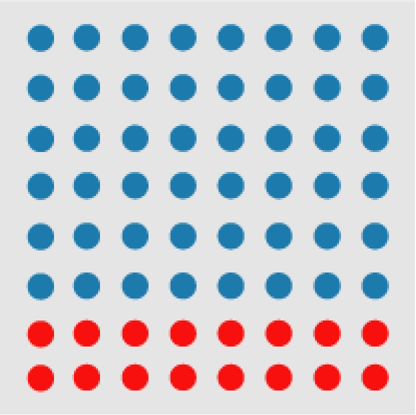
target present

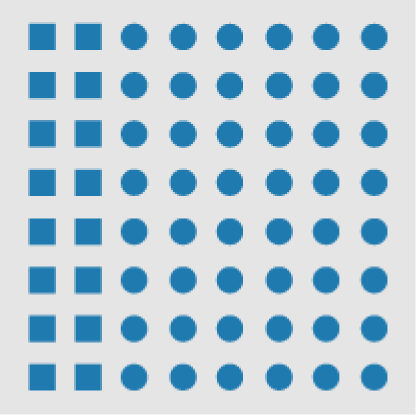
Visual features	target	distractor
shape ≠	●	■
color =	🟥	

Feature interference occurs when a visual feature that is shared between the target and distractors interferes with the preattentive process

FEATURE HIERARCHY

Some visual features interfere more than others, e.g., "color" is easier to detect than "shape"

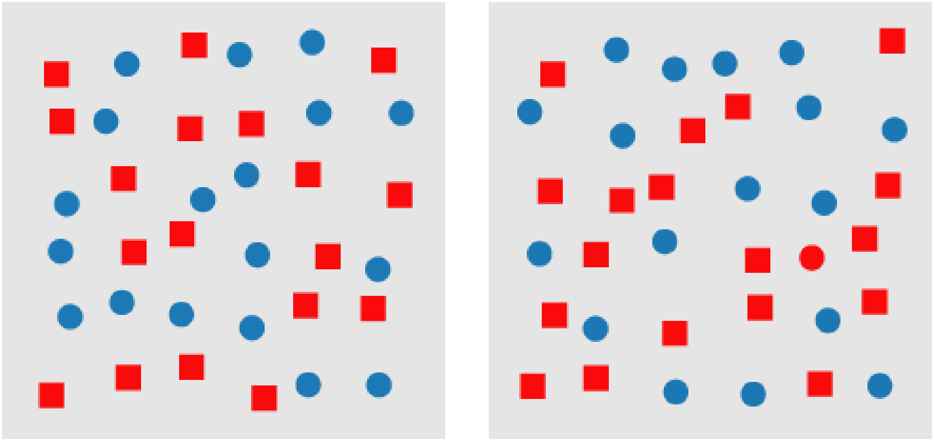




CONJUNCTION SEARCH

> *A conjunction search is a visual search involving a combination of non-unique features. In general a conjunction search cannot be done preattentively.*

● target ● ■ distractors

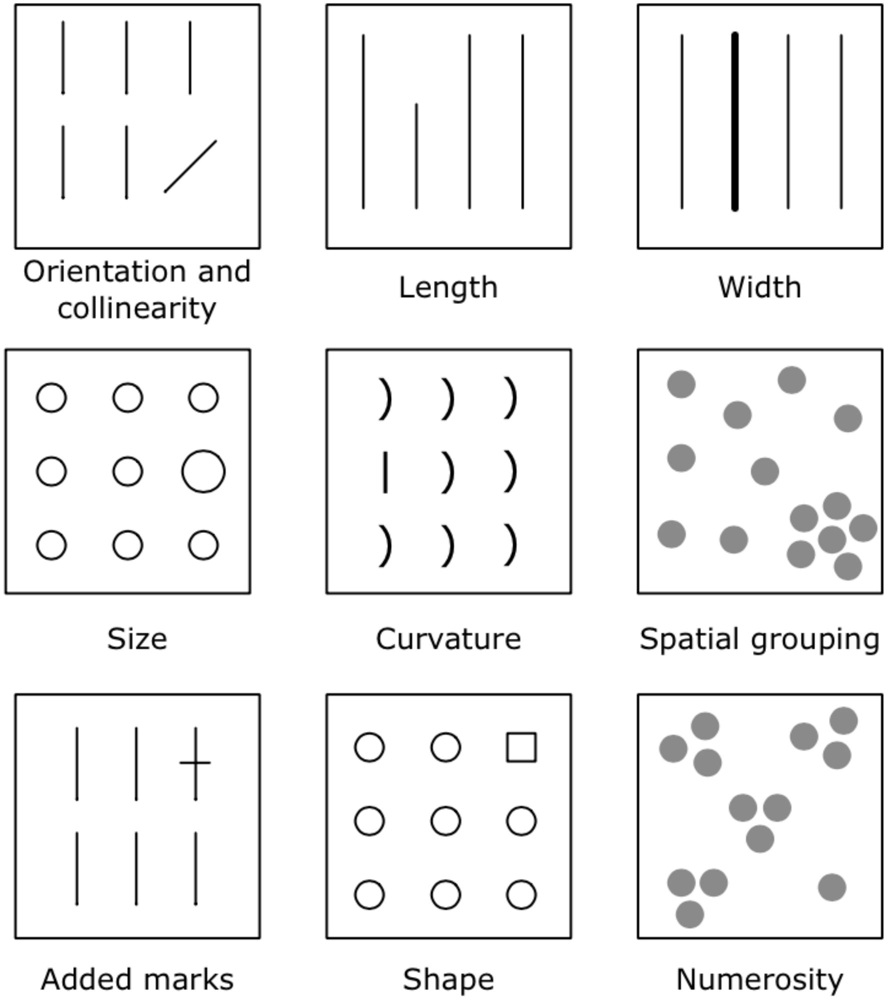


target absent target present

PREATTENTIVE TASKS

Target detection	Detect the presence or absence of a target element with a unique visual feature within a field of distractor elements
Boundary detection	Detect a texture boundary between two groups of elements, where all of the elements in each group have a common visual property
Region tracking	Track one or more elements with a unique visual feature as they move in time and space
Counting & estimation	Count or estimate the number of elements with a unique visual feature

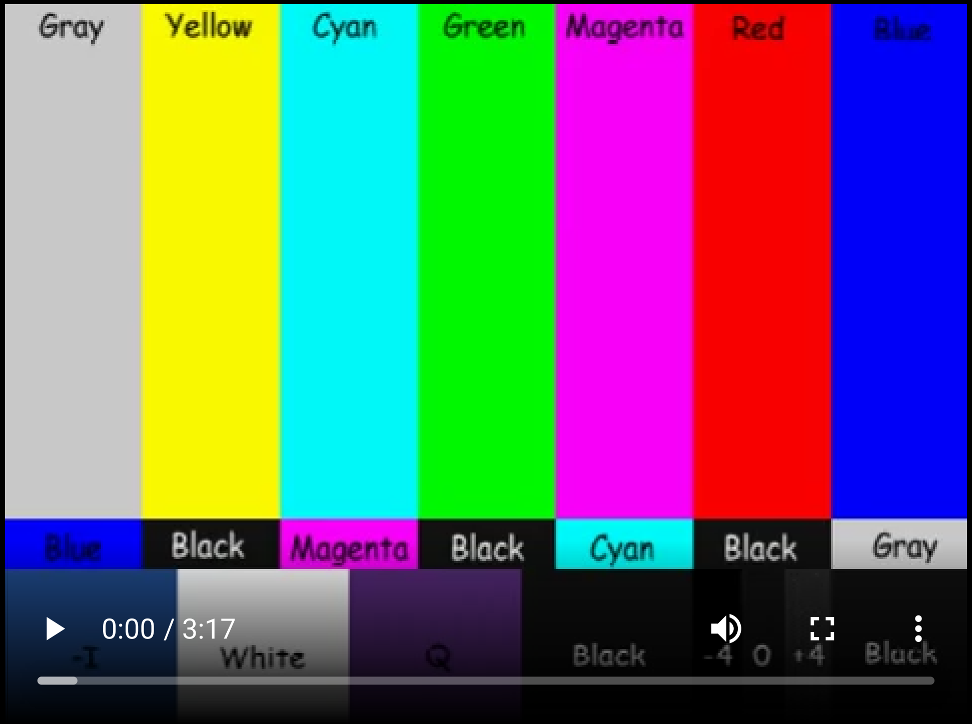
PREATTENTIVE FEATURES



Mazza, Riccardo. Introduction to information visualization. Vol. 149. London: Springer, 2009.

Visual features supporting preattentive processing:

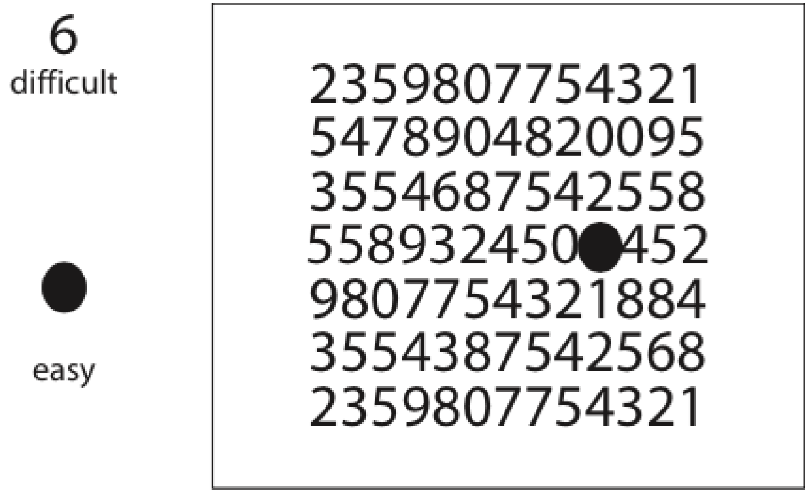
- Color
- Orientation
- Size
- Motion
- Stereoscopic depth

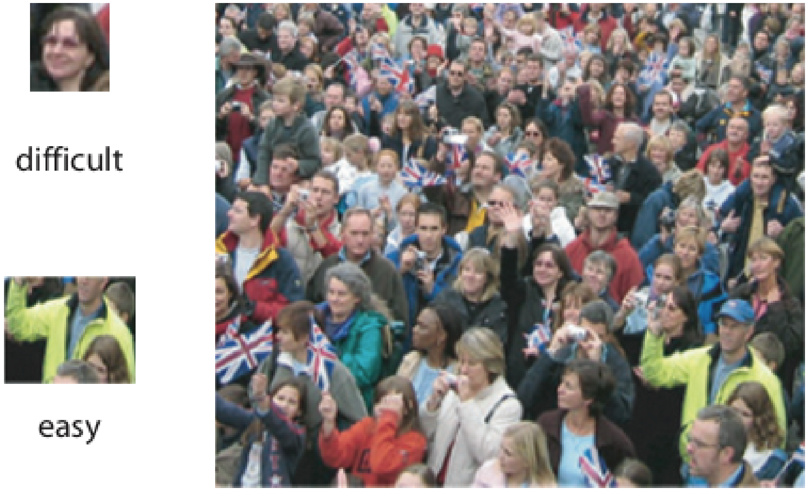


Christopher G. Healey - Preattentive features and tasks

POP-OUT EFFECT

Pre-attentive tasks applied to visual queries

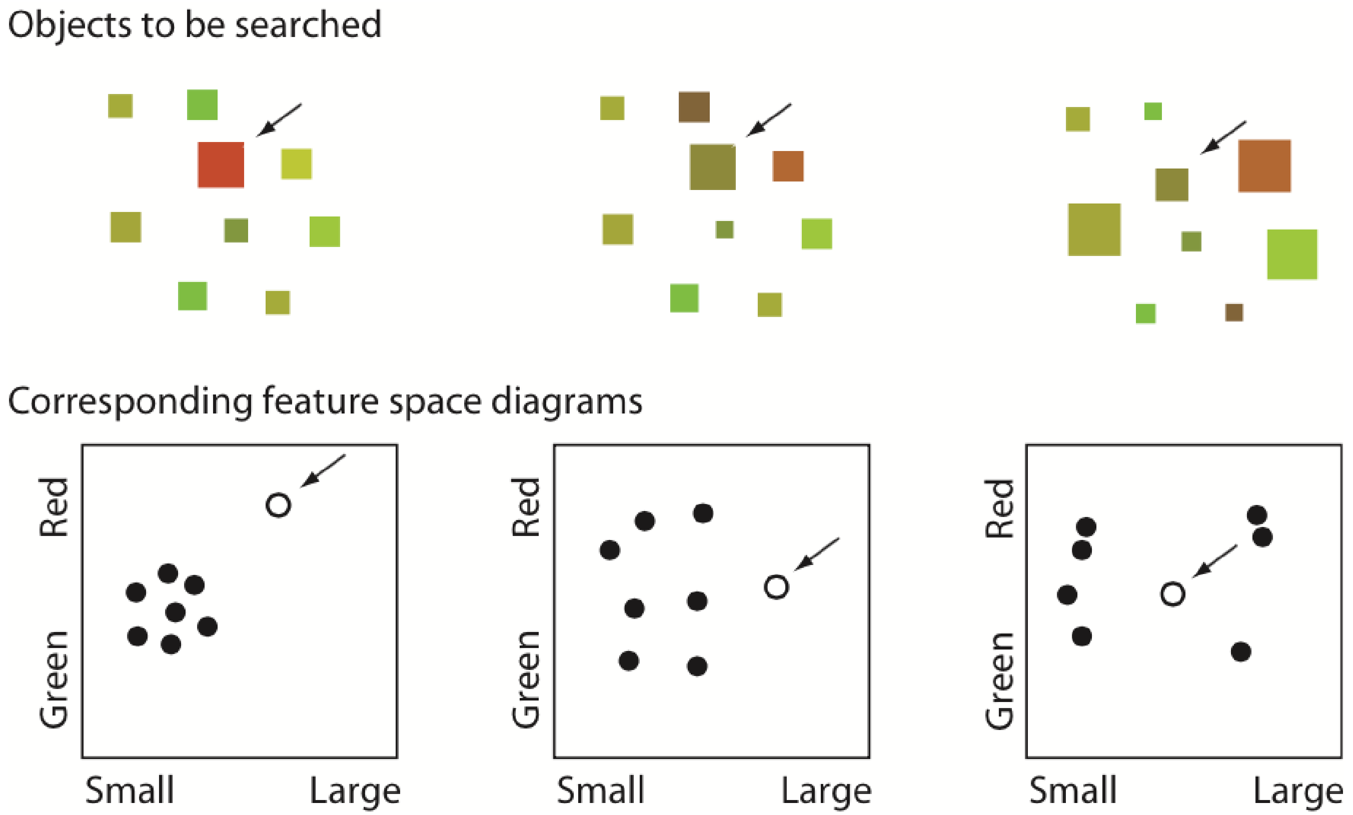




Ware, Colin. Visual thinking: For design. Morgan Kaufmann, 2010.

MAXIMIZING THE POP-OUT EFFECT

IN FEATURE SPACE, THE GREATER THE DISTANCE BETWEEN TARGET AND DISTRACTORS THE GREATER THE POP-OUT EFFECT



DISTINCTIVENESS IN FEATURE SPACE - WARE, COLIN. VISUAL THINKING: FOR DESIGN. MORGAN KAUFMANN, 2010.

OUTLINE

- Pre-attentive features
- Continuity of visual queries
- Interactive visualizations with D3

The new ŠKODA Fabia attention test: https://youtu.be/qpPYdMs97eE

INATTENTIONAL BLINDNESS

- Failure to detect an unexpected stimulus that is fully visible
- Can usually only focus on one thing at the time
- The brain has to prioritize what to focus on

INATTENTIONAL BLINDNESS

- Failure to detect an unexpected stimulus that is fully visible
- Can usually only focus on one thing at the time
- The brain has to prioritize what to focus on

≠

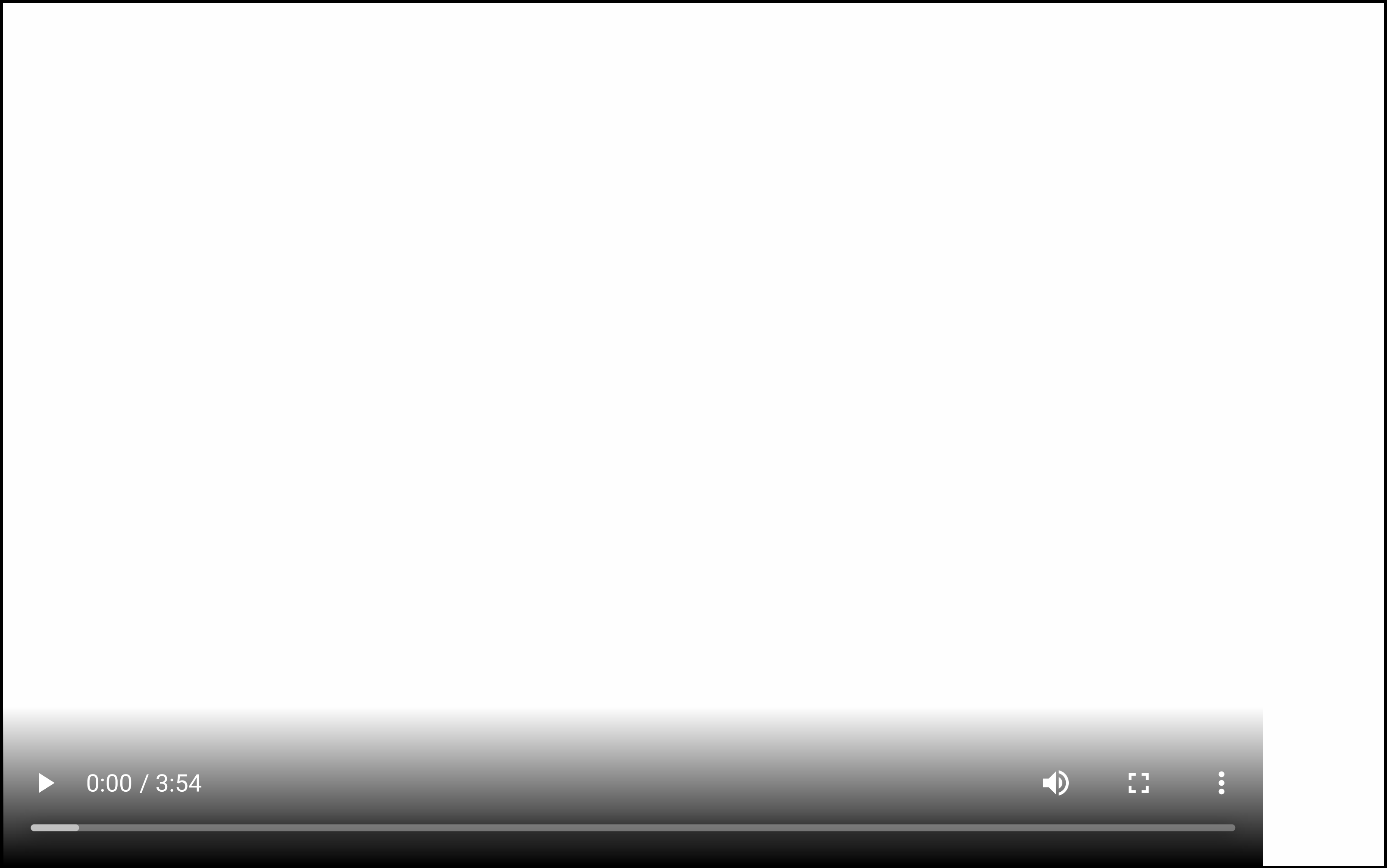
CHANGE BLINDNESS

- Not detecting a brief transitory event occurring in the visual field
- Happens when we blink or move our eyes quickly
- Involves very short term (iconic) memory

ANIMATED TRANSITIONS

- Animations between an initial and a final state
- Ensures object constancy (let's us track objects) during:
    - Filtering data
    - Reordering data
    - Resizing operations
    - Changing forms...

OBJECT CONSTANCY THROUGH ANIMATED TRANSITIONS



Animated Transitions in Statistical Data Graphics [Heer 2007]: https://youtu.be/qpPYdMs97eE

OUTLINE

- Pre-attentive features
- Continuity of visual queries
- Interactive visualizations with D3
    - Events and handling events
    - Updating visualizations
        - Updating the data
        - Data join selections
        - Updating scales and axes
        - Animated transitions

JAVASCRIPT `event`

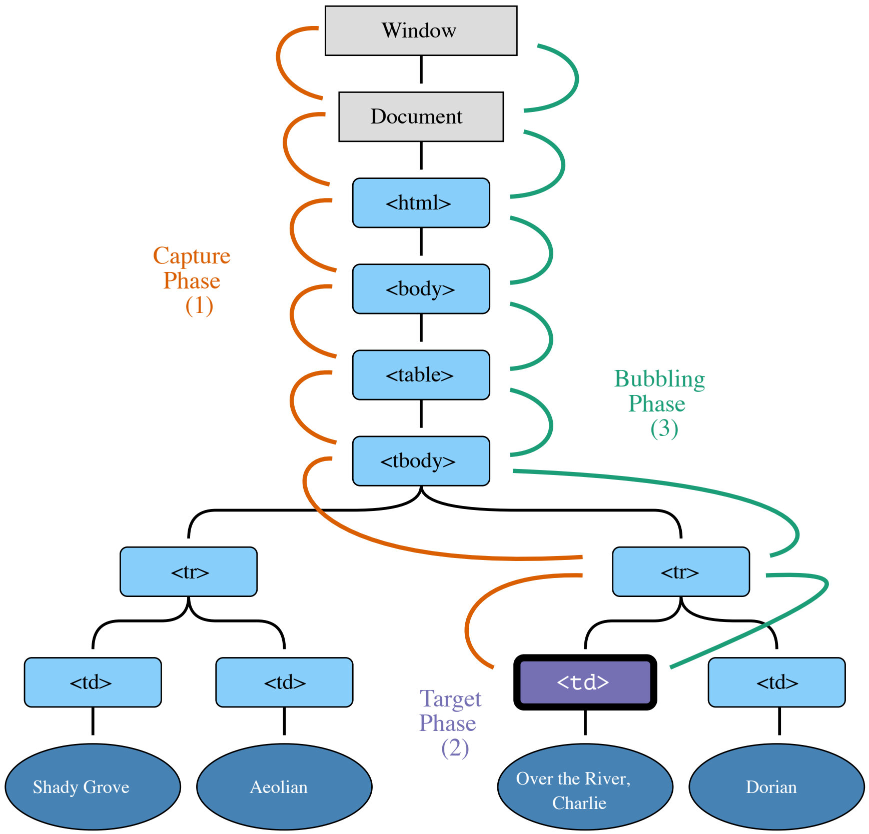
- Resource Events (e.g., load)
- Focus Events (e.g., focus)
- Form Events (submit)
- View Events (e.g., resize)
- Keyboard Events (e.g., keypress)
- Mouse Events (e.g., click)
- Drag & Drop Events
- Media Events (e.g., play)
- SVG events (SVGResize)
- Document events
- Popup events
- Touch events

- Network Events
- Websocket Events
- Session History Events
- CSS Animation Events (animationstart)
- Printing Events
- Clipboard Events
- Text Composition Events
- CSS events
- Script events
- Window events
- DOM mutation events
- ...

(most used in visualizations)

EVENT HANDLING

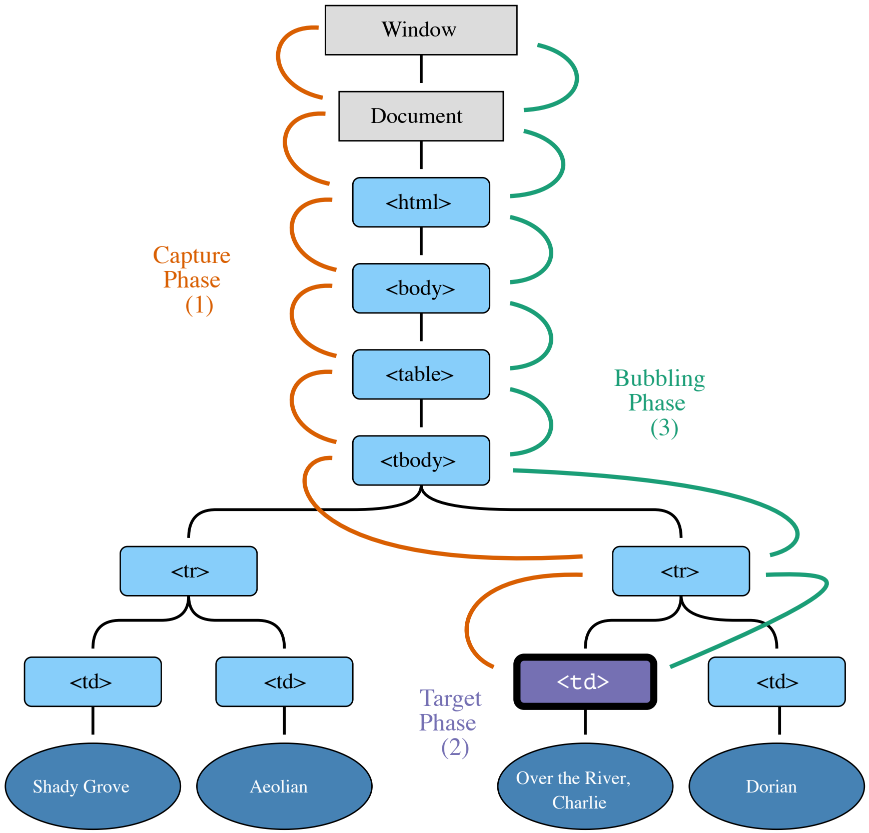
Once emitted, events are propagated in the DOM tree and processed by elements through event listeners



Event listener:

- Function attached to elements used to handle specific events
- The handler is called according to a specific propagation phase

EVENT PROPAGATION PHASES

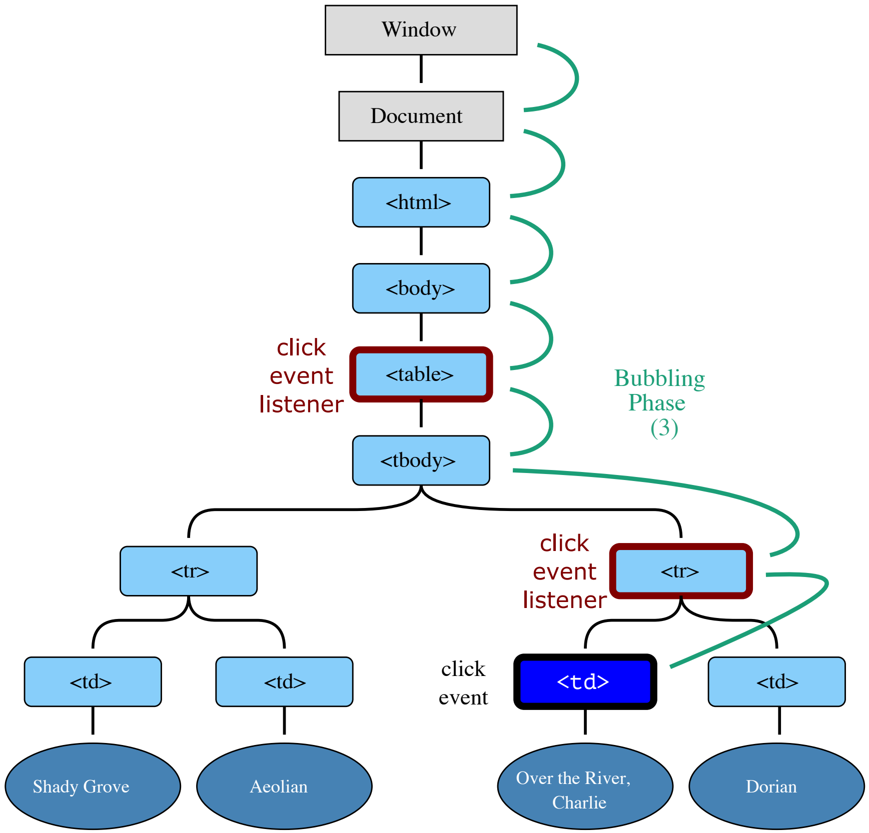


Phases:

1. ↓ Capture phase: rarely used in code
2. → Target phase: element that originated the event (i.e., `event.target`)
3. ↑ Bubbling phase: default used in code

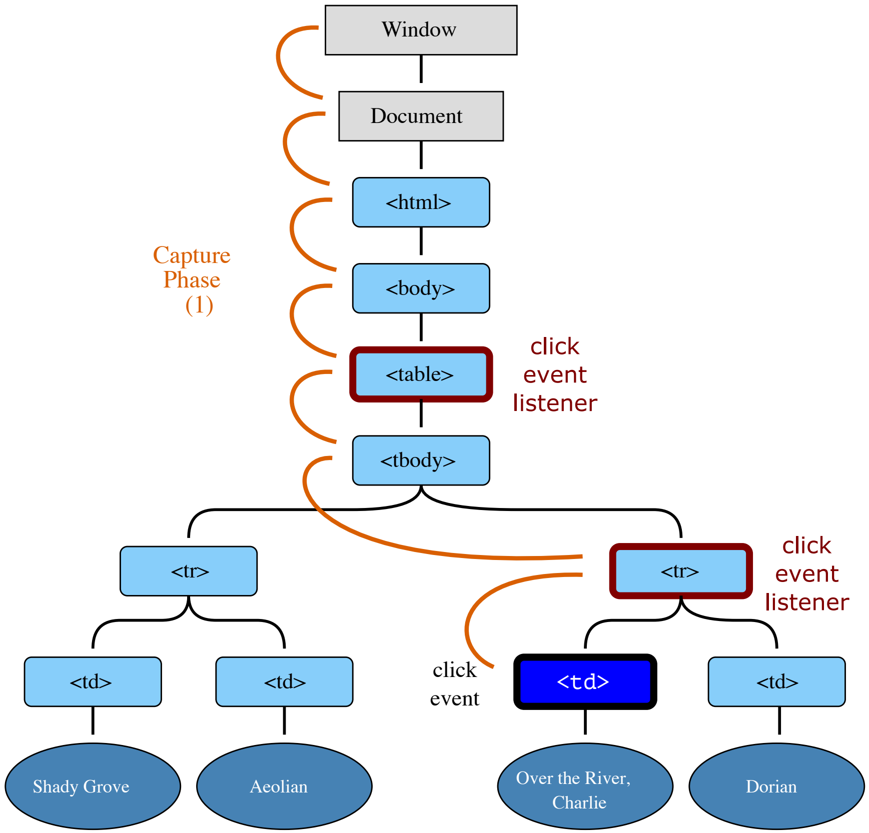
- Bubbling phase is the default phase
- Multiple listeners can be specified
- Phases are useful to respond to events when animating hierarchical data

EXAMPLE USING BUBBLING



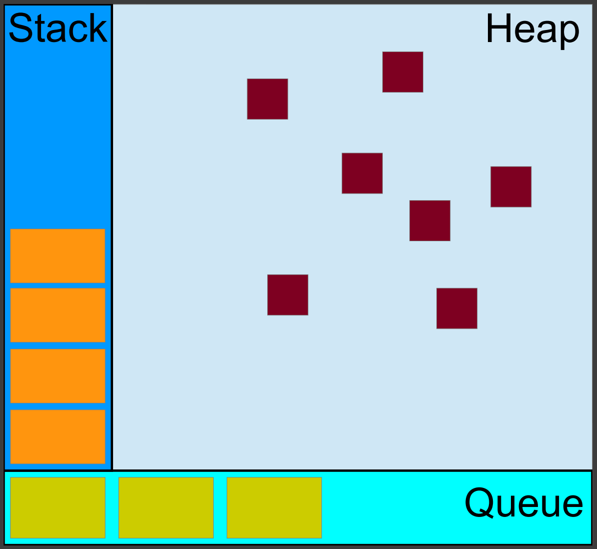
- The listener attached to the <tr> is called first
- The listener attached to the <table> si called last

EXAMPLE USING CAPTURE



- The listener attached to the <table> is called first
- The listener attached to the <tr> is called last

JAVASCRIPT EVENT LOOP



Concurrency model and the event loop

- Stack: stack of frames for function calls to be executed.
- Heap: memory used for JS object.
- Queue: list of messages to be processed.

- Messages are added anytime an event occurs to the queue and there is an event listener attached to it.
- Messages on the queue are handled in turn starting with the oldest one by:
  1. Removing the message from the queue
  2. Calling the message function passing the message as parameter
- Called functions are added as a new stack frame.
- Functions are processed until the stack is empty, then the event loop will process the next message in the queue.

DOM level 0 inline (one event handler per element)

```
<div id="#d0" onclick="document.getElementById('#d0').style.color = 'red'; return false;">
  DOM level 0 inline (one event handler per element)
</div>
```

DOM level 0 traditional (one event handler per element)

```
var makeGreen = function () {
  document.getElementById('#div01').style.color = 'green' };

document.getElementById('#div01').onclick = makeGreen;  //assign an event handler
//document.getElementById('#div01').onclick = null;  //remove event handler
```

DOM level 2 (multiple event handlers per element)

```
var makeBlue = function () {
  document.getElementById('#d2').style.color = 'blue' };

var makeOrange = function () {
  document.getElementById('#d2').style.color = 'orange' };

var div = document.getElementById('#d2');
//target.addEventListener(type, listener[, useCapture]);
div.addEventListener("click", makeBlue, true);  //useCapture=true, capture phase
div.addEventListener("click", makeOrange);  //useCapture=false default, use bubbling phase

//target.removeEventListener(type, listener[, useCapture]);
//div.removeEventListener("click", makeOrange, false);  //remove handler
```

HANDLING EVENTS WITH CSS HOVER PSEUDO-CLASS

> *A CSS pseudo-class is a keyword added to a selector that specifies a special state of the selected element(s).*

```css
blockquote:hover {
  background-color: orangered;
  color: white;
}
```

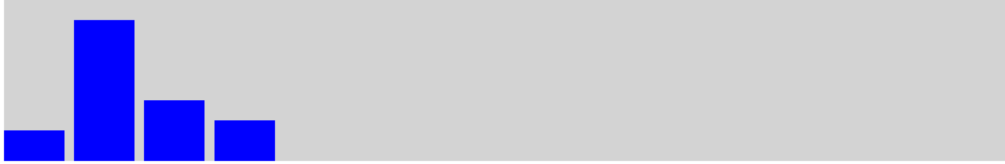
HANDLING EVENTS WITH D3

```
//selection.on(typenames[, listener[, capture]]);
//Sets DOM level 2 event listeners
//typenames: event type string
//listener: call-back
//capture: useCapture flag

var flag = false;
d3.select("#b0")
  .on("click", function(event) {
    console.log(event);  //event object
    if (event.metaKey) {  //check if event has meta key
      d3.select("#b1")  //select and modify d01
        .text('b1: b0 event')
        .style("background-color", function() {
          flag = !flag;
          return flag ? "red" : "lightseagreen";
        });
    }
})

d3.select("#b1")
  .on("mouseout click", function() {  //multiple typenames allowed separated with spaces
    d3.select(this)  //this == element for this event listener
      .text('b1: b1 event')
      .style("background-color", function() {
        flag = !flag;
        return flag ? "orange" : "lightseagreen";
      })
})
```

b0 (⌘ click)

b1

HANDLING EVENTS IN D3 DATA JOIN



```
var data = [15, 70, 30, 20];
var svg = d3.select('#c00');
svg.append('text')
  .attr('id', 'v00')
  .attr('x', '20')
  .attr('y', '20')
  .style('font-size', '18px');

  svg.selectAll('rect')
  .data(data)
  .enter()
  .append('rect')
  .attr('x', function (d, i) { return i * 35; })
  .attr('y', function (d) { return 80 - d; })
  .attr('width', function (d) { return 30; })
  .attr('height', function (d) { return d; })
  .attr('fill', function (d) { return 'blue'; })
  .on("mouseover", function (event, d) {
    d3.select(this).style('fill', 'orangered');
    console.log(event)
    d3.select('#v00').text('event=' + event + ' d=' + d);
  })
  .on("mouseout", function (event, d) {
  d3.select(this).style('fill', 'blue');
  d3.select('#v00').text('');
  });
```

OUTLINE

- Pre-attentive features
- Continuity of visual queries
- Interactive visualizations with D3
  - Events and handling events
  - Updating visualizations
    - Updating the data
    - Data join selections
    - Updating scales and axes
    - Animated transitions

UPDATING THE DATA

JAVASCRIPT ARRAY METHODS

- array.map returns array where a function is called on every element
- array.sort sorts in place the elements of the array.
- array.slice shallow copy of a portion of an array into a new array object
- array.shift remove the first element from the array
- array.splice changes array by removing existing and/or adding new elements

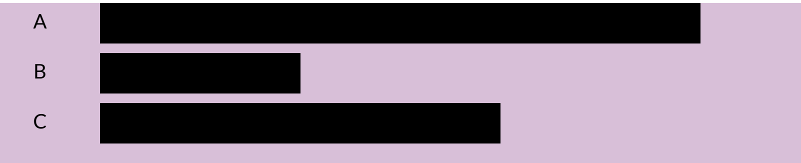
D3-ARRAY METHODS

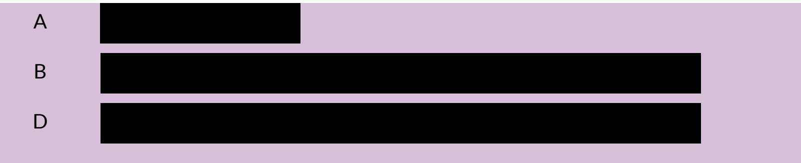
- d3.min compute the minimum value in an array
- d3.max compute the maximum value in an array
- d3.ascending comparator function to use with `array.sort`
- d3.descending comparator function to use with `array.sort`

OUTLINE

- Pre-attentive features
- Continuity of visual queries
- Interactive visualizations with D3
  - Events and handling events
  - Updating visualizations
    - Updating the data
    - Data join selections
    - Updating scales and axes
    - Animated transitions

DATA JOIN WITH ENTER SELECTION



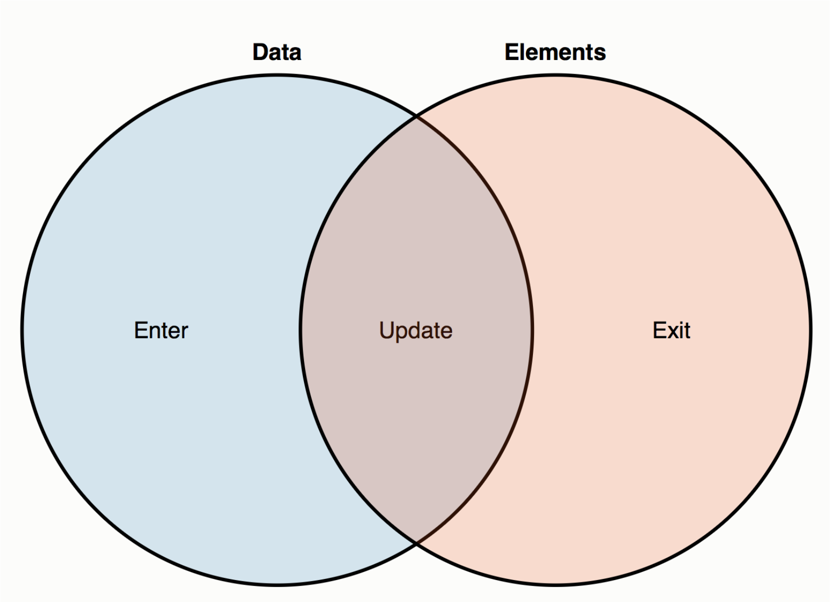


```
//We start with some data...
var data = [{item: 'A', value: 3},
            {item: 'B', value: 1},
            {item: 'C', value: 2}];


enter = svg.selectAll("rect")
  .data(data)
  .enter();

enter.append("rect")
  .attr("x", 50)
  .attr("y", function(d, i) { return i * 25; })
  .attr("width", function(d) { return d.value * 100; })
  .attr("height", 20)

enter.append("text")
  .attr('font-size', '18px')
  .attr('text-anchor', 'middle')
  .attr('alignment-baseline', 'middle')
  .attr("x", 20)
  .attr("y", function(d, i) { return 10 + i * 25; })
  .text((d) => { return d.item; });
```

```
//... and at some point we change the data
data = [{item: 'A', value: 1},  //update
        {item: 'B', value: 3}, //update
        //{item: 'C', value: 2}, //remove
        {item: 'D', value: 3}];  //add

enter = svg.selectAll("rect")  //data join
  .data(data)
  .enter();

enter.append("rect")  //enter selection is empty!
  .attr("x", 50)
  .attr("y", function(d, i) { return i * 25; })
  .attr("width", function(d) { return d.value * 100; })
  .attr("height", 20)

enter.append("text")  //enter selection is empty!
  .attr('font-size', '18px')
  .attr('text-anchor', 'middle')
  .attr('alignment-baseline', 'middle')
  .attr("x", 20)
  .attr("y", function(d, i) { return 10 + i * 25; })
  .text((d) => { return d.item; });
```

NEED DATA JOIN ON UPDATE AND EXIT SELECTIONS!

DATA JOIN SELECTIONS



$$\begin{aligned} \text{Enter} &: \text{data array size} > \text{selection size} \\ \text{Update} &: \text{data array size} = \text{selection size} \\ \text{Exit} &: \text{data array size} < \text{selection size} \end{aligned}$$

THE GENERAL UPDATE PATTERN

1. DATA *"JOIN"*: `selection.data(...)`
Join new data with old elements, if any.
Returns a reference to the *"update"* selection.

2. UPDATE
Update old elements as needed.

3. ENTER: `selection.enter()`
Create new elements as needed.

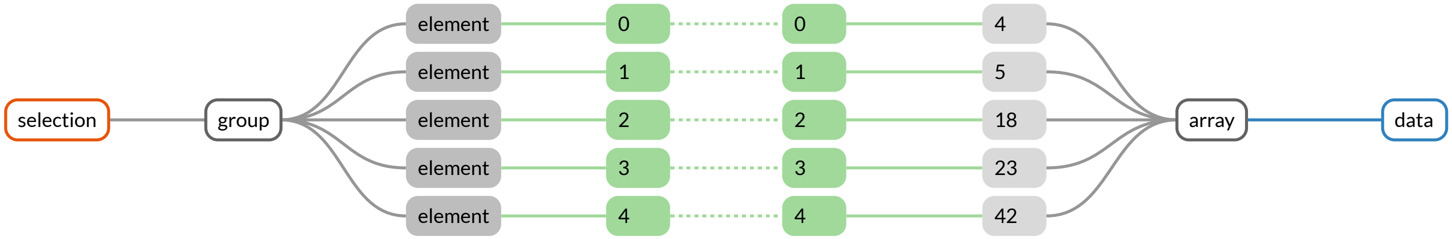
4. ENTER + UPDATE: `selection.merge(...)`
Merge the entered elements with the update selection and apply operations to both.

5. EXIT: `selection.exit()`
Remove old elements as needed.

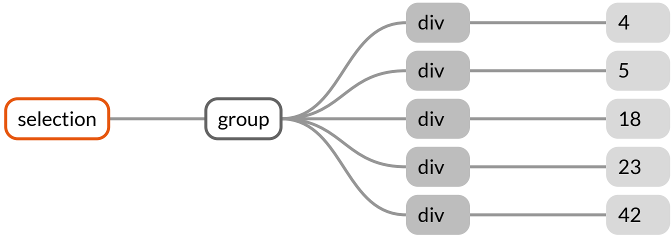
See Mike Bostok's General Update Pattern I, General Update Pattern II and General Update Pattern III

selection.data(dataset)

```
var dataset = [4, 5, 18, 23, 42];
```



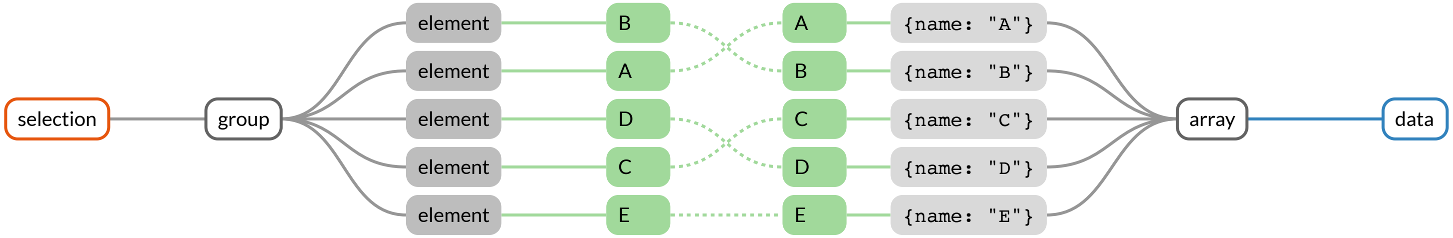
```
//maps data according to data and selection order, the first datum
//in values is assigned to the first element in the selection and so on
d3.selectAll('div').data(dataset);
```



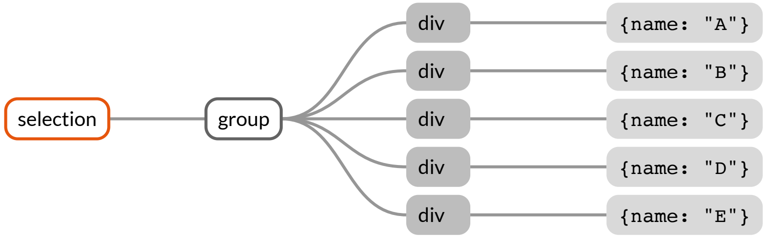
Mike Bostok's How Selections Work

selection.data(dataset, key)

```
var dataset = [{name: "A", frequency: .08167},
  {name: "B", frequency: .01492},
  {name: "C", frequency: .02780},
  {name: "D", frequency: .04253},
  {name: "E", frequency: .12702}];
```



```
//updates can occur anywhere in the data array, depending
//on the overlap between the old and new values
function name(d) { return d.name; }  //returns the key for a datum
d3.selectAll("div").data(dataset, name);
```



OUTLINE

- Pre-attentive features
- Continuity of visual queries
- Interactive visualizations with D3
  - Events and handling events
  - Updating visualizations
    - Updating the data
    - Data join selections
    - Updating scales and axes
    - Animated transitions

UPDATING SCALES AND AXES

1. Adjust the scale properties:

```
var dataset = [{name: "A", frequency: .08167},
  {name: "B", frequency: .01492},
  {name: "C", frequency: .02780},
  {name: "D", frequency: .04253},
  {name: "E", frequency: .12702}];

y.domain([d3.min(dataset, function (d) { return d.frequency; }),
          d3.max(dataset, function (d) { return d.frequency; })])
 .range([0, 600]);

x.domain(dataset.map((d) => { return d.name; })) //ordinal scale
 .range([0, width]);
```

2. Redraw the axes:

```
svg.select('.axis')
  .call(xAxis);
```

OUTLINE

- Pre-attentive features
- Continuity of visual queries
- Interactive visualizations with D3
  - Events and handling events
  - Updating visualizations
    - Updating the data
    - Data join selections
    - Updating scales and axes
    - Animated transitions

TRANSITION BETWEEN STATES (HTML)

Done with CSS

Transition width and background-color

```
<div id="delay1">Transition width and background-color</div>


<style>
#delay1 { /* initial state */
  transition-property: width, background-color;
  transition-duration: 3000ms;
  transition-delay: 1000ms;
  transition-timing-function: ease-in-out;
  width: 480px;
  background-color: darkorange; }

#delay1:hover {  /* final state */
  transition-property: width, background-color;
  transition-duration: 3000ms;
  transition-delay: 1000ms;
  transition-timing-function: ease-in-out;
  width: 960px;
  background-color: cornflowerblue; }
</style>
```

TRANSITION BETWEEN STATES (SVG)

Done with CSS: same as HTML with SVG attributes!

```
<svg style="background-color: mistyrose" width="100%" height="30px">
  <rect id="delay2" x="0" y="5" width="450" height="20"></rect>
</svg>
<style>
  #delay2 {
    transition-property: width, fill;
    transition-duration: 3000ms;
    transition-delay: 1000ms;
    transition-timing-function: ease-in-out;
    width: 480px;
    fill: darkorange; }

  #delay2:hover {
    transition-property: width, fill;
    transition-duration: 3000ms;
    transition-delay: 1000ms;
    transition-timing-function: ease-in-out;
    width: 960px;
    fill: cornflowerblue; }
</style>
```

EASING: CSS TIMING FUNCTIONS

Method of distorting time to control apparent motion in animation. The timing function defines an acceleration curve controlling the speed of the transition over its duration.

```
/* Keyword values */
transition-timing-function: ease;
transition-timing-function: ease-in;
transition-timing-function: ease-out;
transition-timing-function: ease-in-out;
transition-timing-function: linear;
transition-timing-function: step-start;
transition-timing-function: step-end;

/* Function values */
transition-timing-function: steps(4, end);
transition-timing-function: cubic-bezier(0.1, 0.7, 1.0, 0.1);
transition-timing-function: frames(10);

/* Multiple timing functions */
transition-timing-function: ease, step-start, cubic-bezier(0.1, 0.7, 1.0, 0.1);

/* Global values */
transition-timing-function: inherit;
transition-timing-function: initial;
transition-timing-function: unset;
```

See the Easing Functions Cheat Sheet and MDN web docs transition-timing-function

TRANSITIONS WITH D3

```
<svg id="svg20" style="background-color: mistyrose" width="100%" height="30px"></svg>
<script>
  d3.select("#svg20")
    .append("rect")
    .attr("y", 5)
    .attr("width", 480)
    .attr("height", 20)
    .attr("fill", "darkorange")
    .on("mouseover", function () {
     d3.select(this)
       .transition()                     //selection.transition() works on the selection
       .delay(1000)                      //transition delay in ms
       .duration(3000)                   //transition duration in ms
       .ease(d3.easeBounce)              //specify easing function, defaults to d3.easeCubic
       .attr("width", 960)               //final transition state
       .attr("fill", "cornflowerblue");  //final transition state
    })
    .on("mouseout", function () {
      d3.select(this)
        .transition()
        .delay(1000)
        .duration(3000)
        .attr("width", 480)
        .attr("fill", "darkorange");
    });

  //Only one transition at the time per element!

  //d3.transition() works also on data joins where
  //function(d, i) {...} can be used with .delay() and .duration()
  //to implement staggered transitions
</script>
```