DSCI 554 LECTURE

PRE-ATTENTIVE FEATURES, INTERACTIVE VISUALIZATIONS

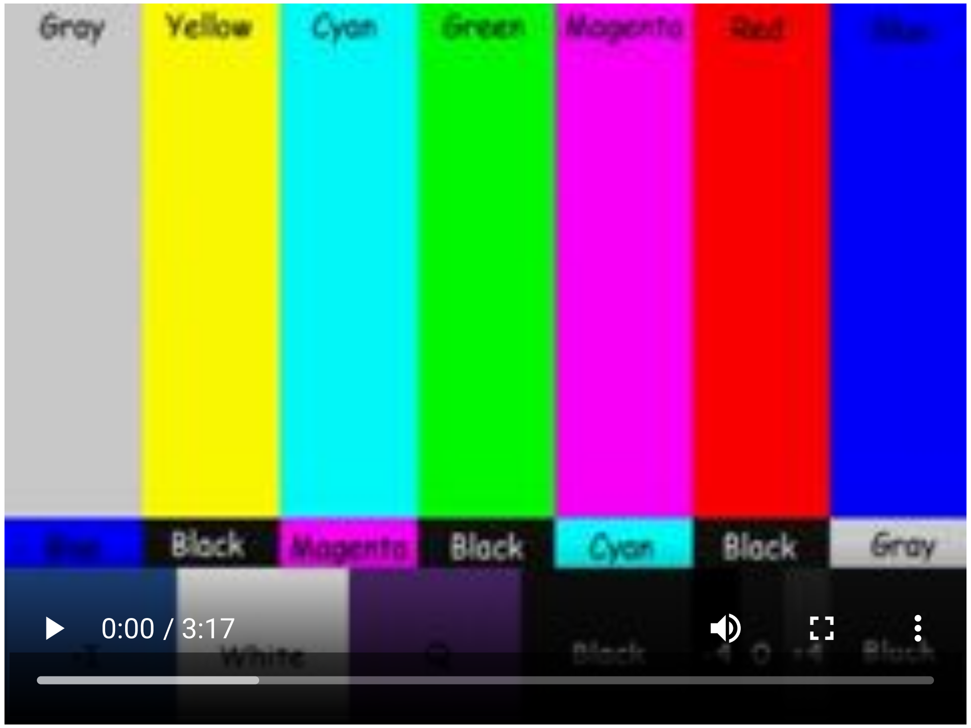
Dr. Luciano Nocera

USC Viterbi School of Engineering
Integrated Media Systems Center

OUTLINE

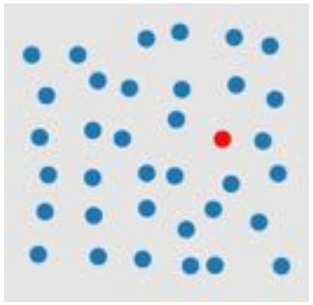
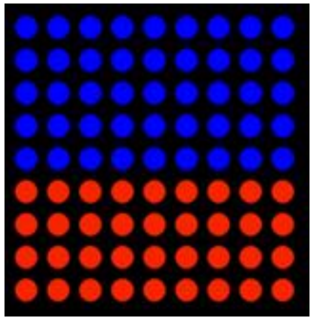
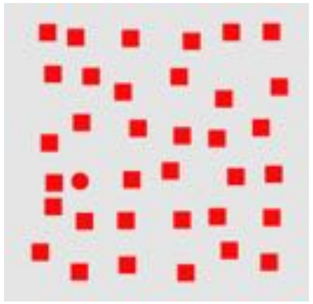
- Pre-attentive features
- Continuity of visual queries
- Interactive visualizations with D3

PREATTENTIVE FEATURES



Christopher G. Healey - Preattentive features and tasks

PREATTENTIVE PROCESSING*

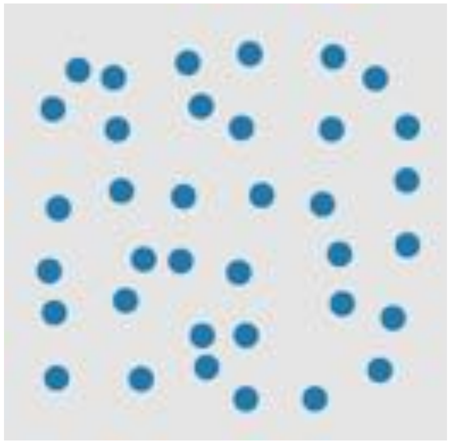




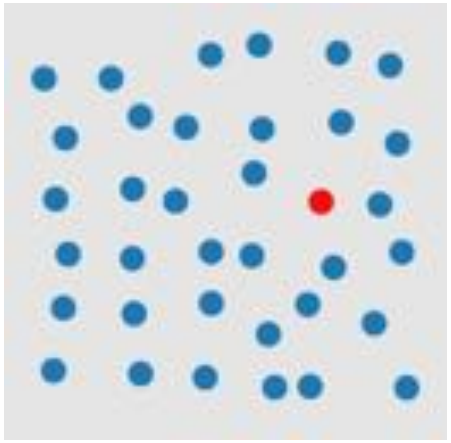
- Helps present information at a glance
- $< 200ms$ on large multi-element displays
- Uses information from a single glimpse
- Does not involve attention
- Independent of:
  - Practice
  - Familiarity with the features
  - Number of distractors
- Universal property

* Healey, C. G., Booth, K. S., & Enns, J. T. (1996). High-speed visual estimation using preattentive processing. ACM Transactions on Computer-Human Interaction (TOCHI), 3(2), 107-135.

TARGETS AND DISTRACTORS

● target ● distractor

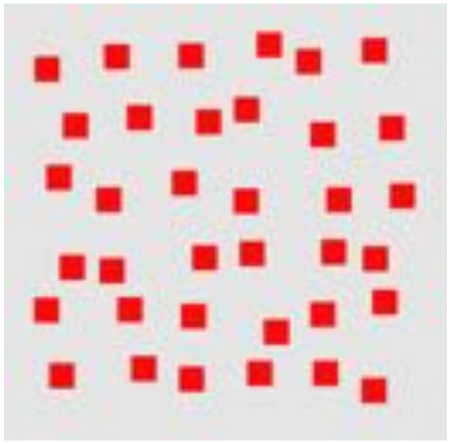

target absent

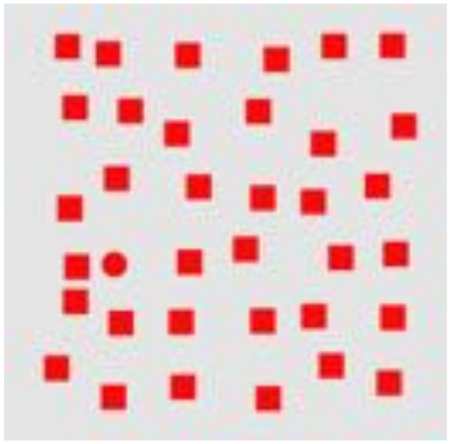

target present

	target	distractor
shape =	●	●
color ≠	🟥	🟦

FEATURE INTERFERENCE

● target 🟥 distractor


target absent

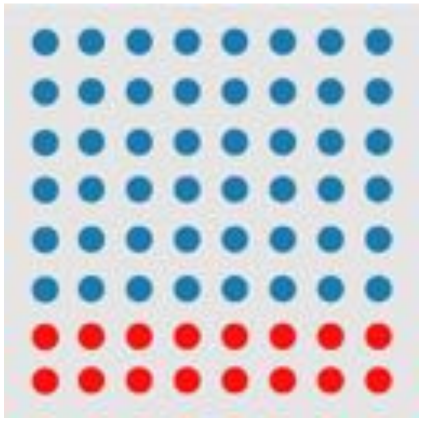

target present

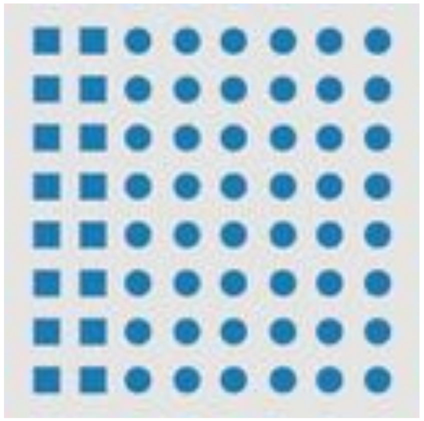
Visual features	target	distractor
shape ≠	●	■
color =	🟥	🟥

Occurs when visual features shared between targets and distractors interfere with the preattentive process

FEATURE HIERARCHY

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

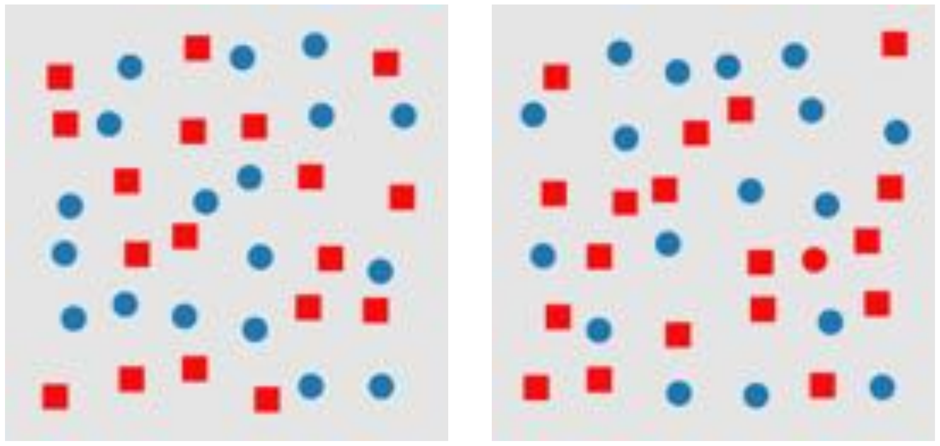




CONJUNCTION SEARCH

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

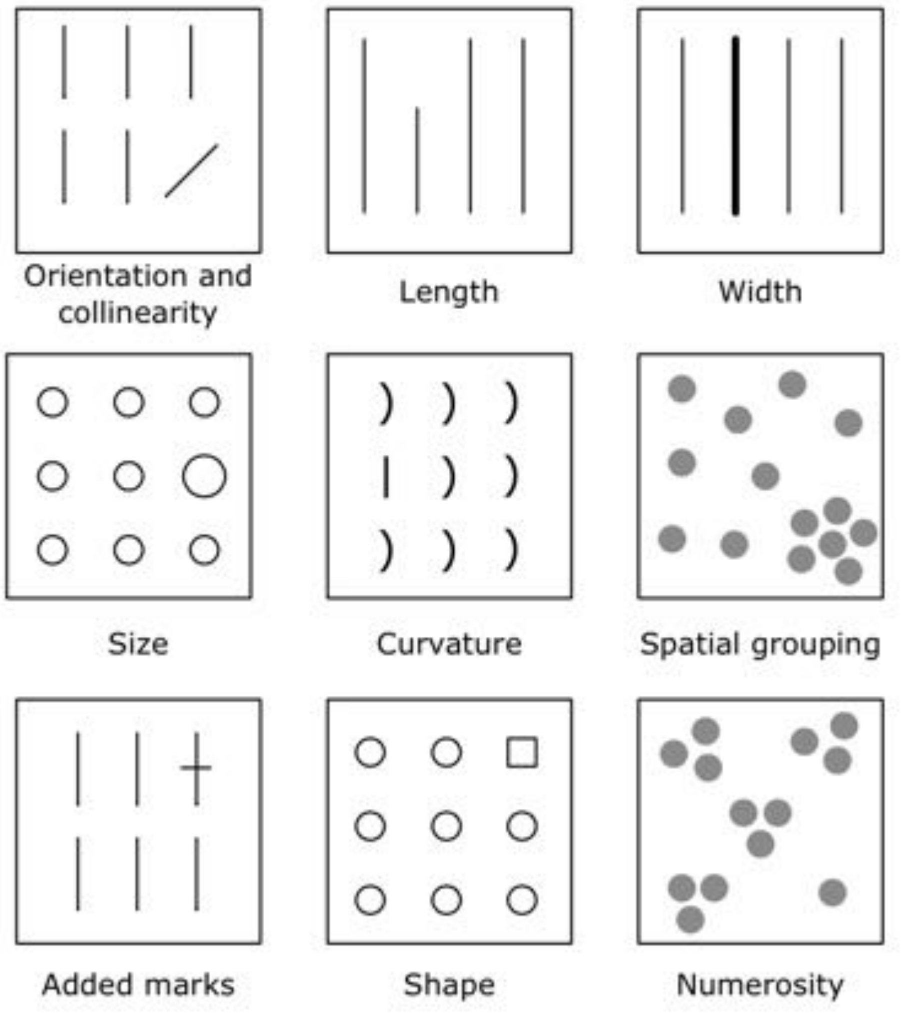
● target ● ■ distractors


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



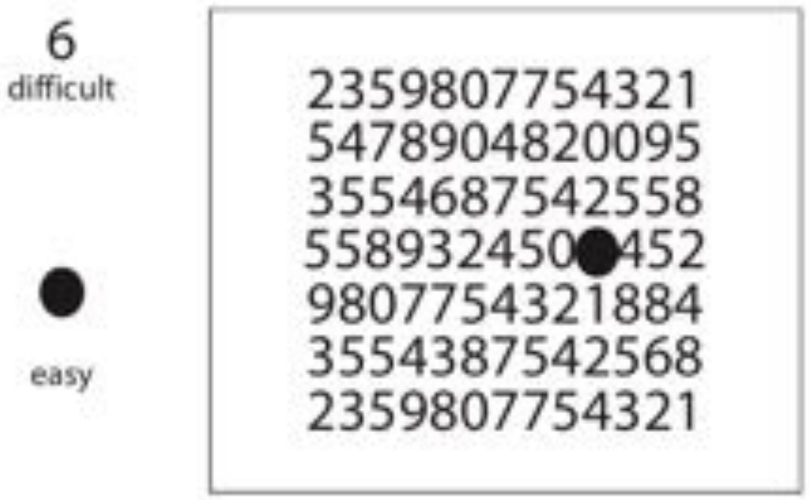
Mazza, Riccardo. Introduction to information visualization. Vol. 149. London: Springer, 2009.

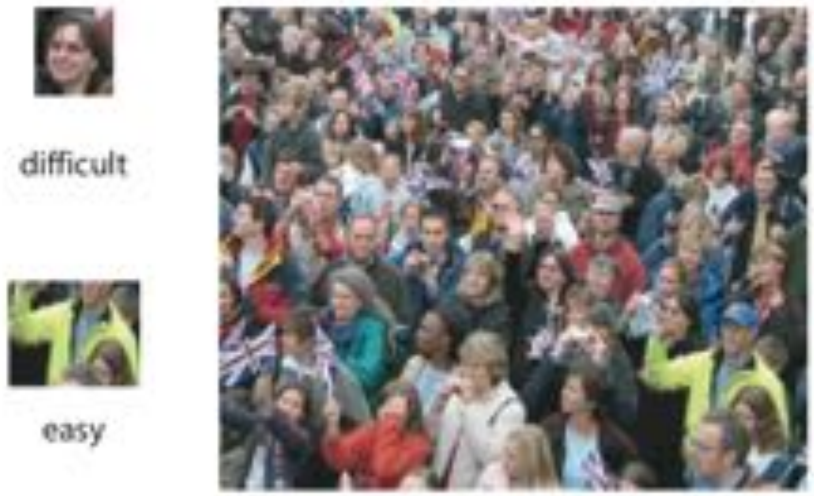
Visual features supporting preattentive processing:

- Color
- Orientation
- Size
- Motion
- Stereoscopic depth

POP-OUT EFFECT

Preattentive processing applied to visual queries

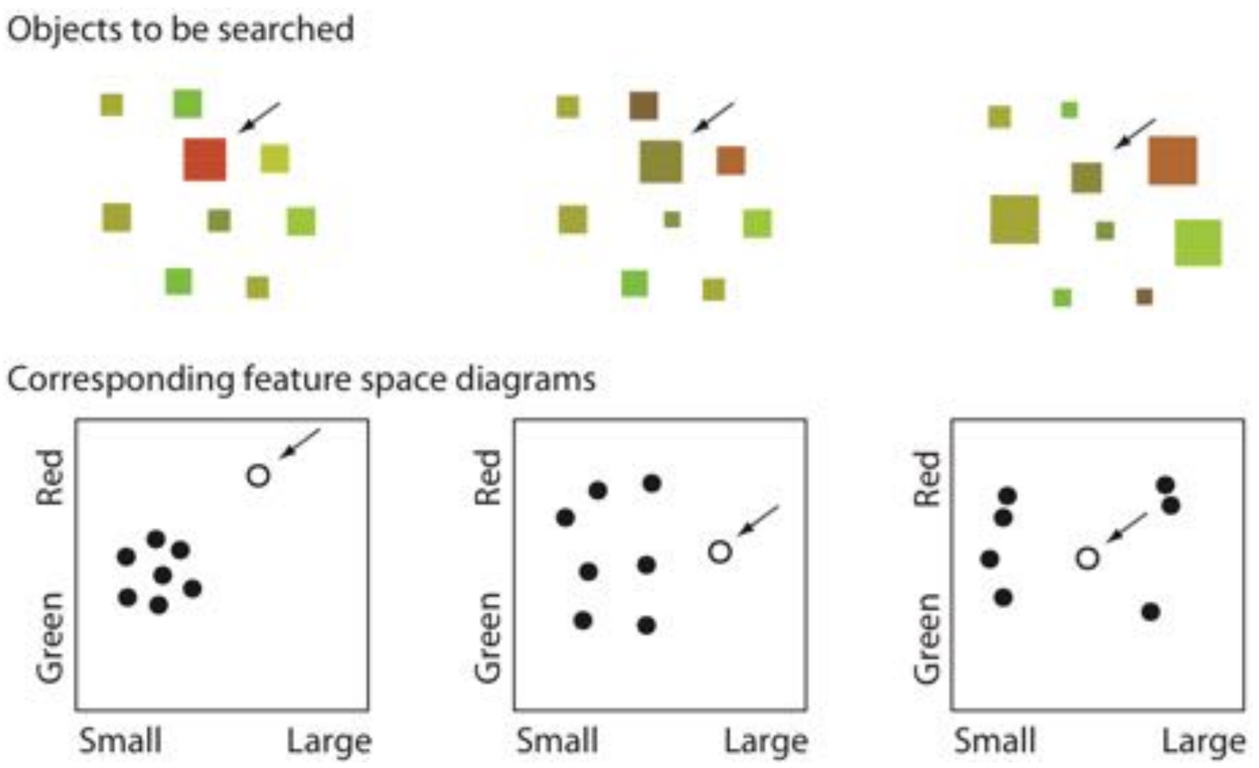




Ware, Colin. Visual thinking: For design. Morgan Kaufmann, 2010.

HOW TO MAXIMIZE THE POP-OUT EFFECT

In feature space: the greater the distance between the target and the 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

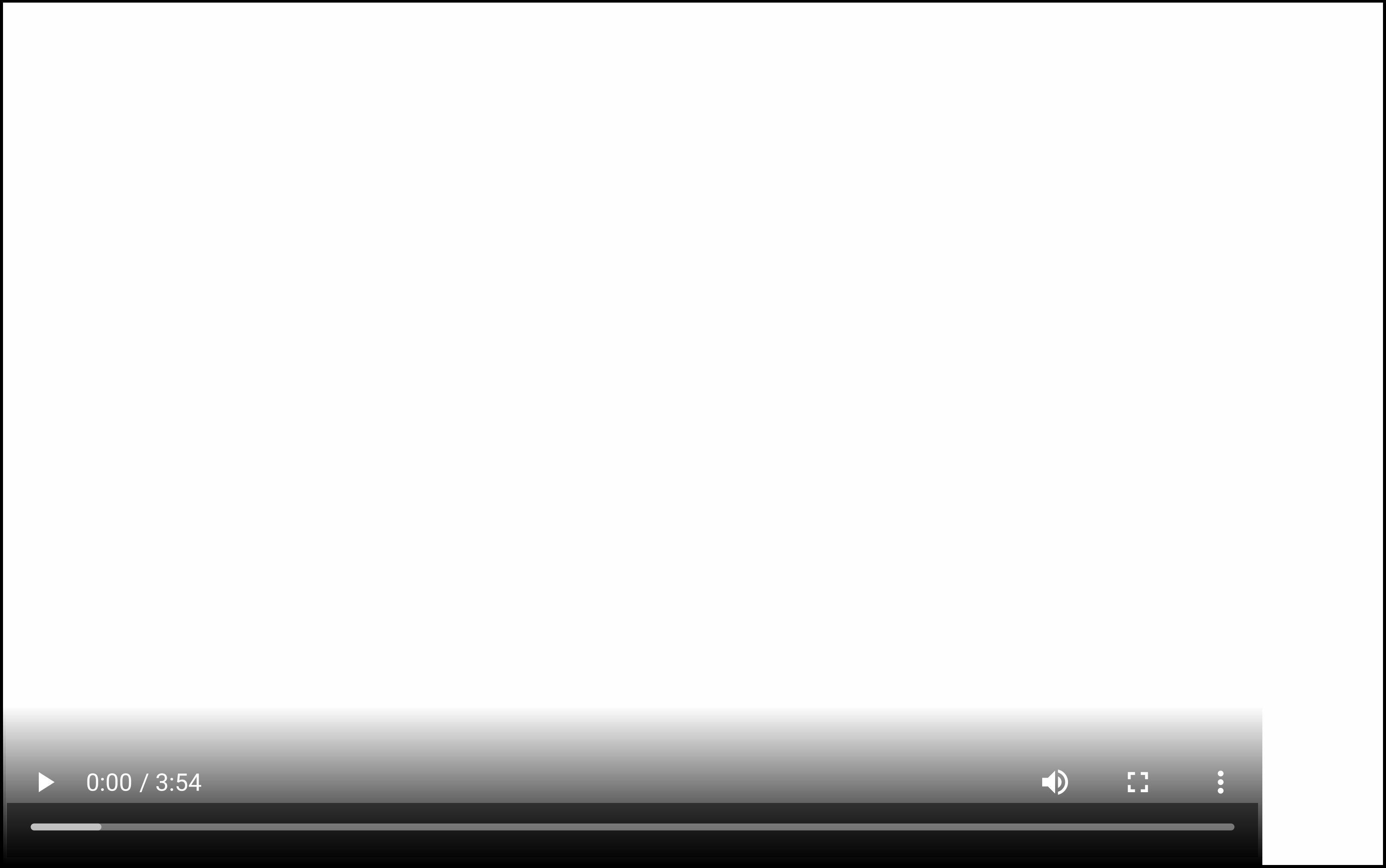
- *Failure to detect an unexpected stimulus that is fully visible*
- *Can usually only focus on one thing at the time*
- *The brain has to prioritize what to focus on*

$\neq$

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



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

ANIMATED TRANSITIONS

- Carefully crafted animations between initial and final states ensures object constancy
- Can be applied when we:
  - Filter the data
  - Reorder the data
  - Resize the the data
  - Change between forms

INTERACTIVITY TECHNIQUES

SINGLE-VIEW DISPLAYS

- Highlight selection, on hover
- Highlight with controls
- Highlight with brushing
- Interactive labels
- Tooltips, consolidated tooltips
- Pan & zoom, with controls and brushing
- Interactive legend
- Interactive stats
- Index chart

MULTI-VIEW DISPLAYS

- Pan and zoom
- Crossfilter
- Overview & detail
- Details on Hover
- Brushing for filtering data
- Brushing for zooming

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

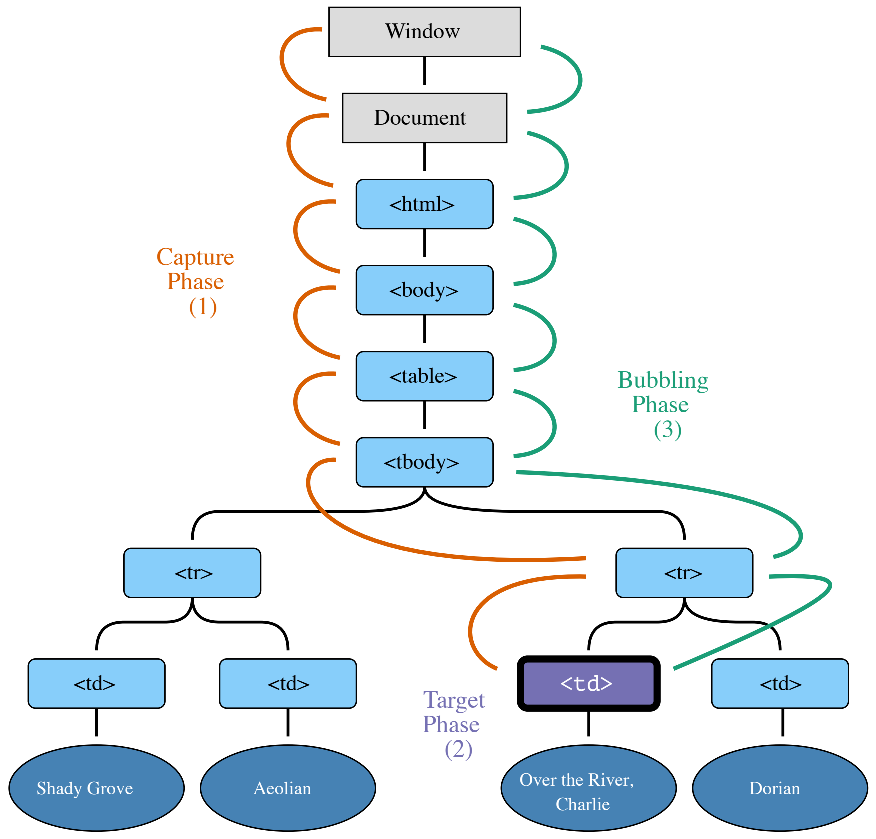
JAVASCRIPT `event`

- 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 relevant to visualization

EVENT PROPAGATION IN THE DOM TREE



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

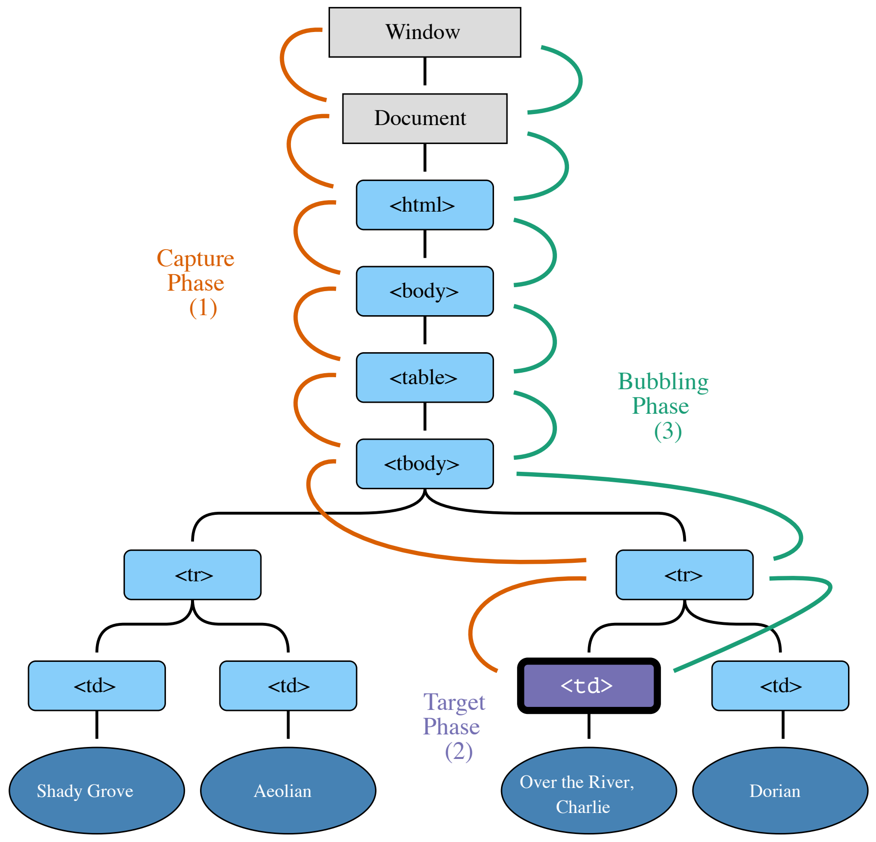
Three propagation phases:

- Capture Phase
- Target Phase
- Bubbling Phase

Event listeners:

- Are functions attached to elements that handle specific events
- Event handlers are called according to a specific propagation phase
- Multiple listeners can be specified

EVENT PROPAGATION PHASES

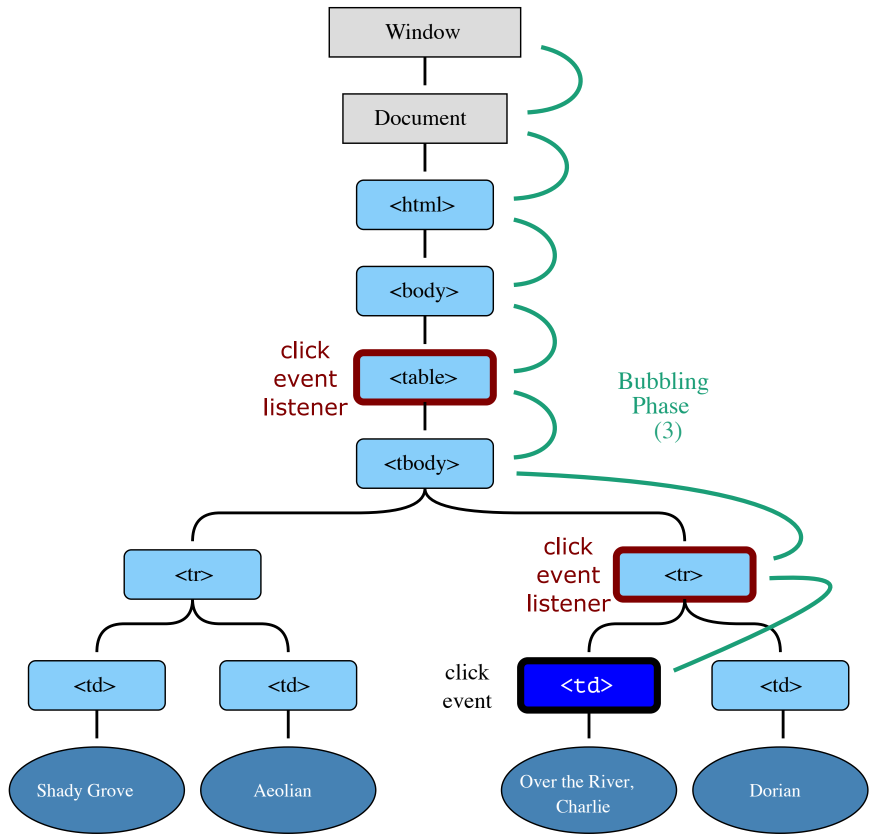


Bubbling phase
From element where the event originates to the Window. Default phase in event handlers and most used in code.

Capture phase
From the Window to the element where the event originates. Rarely used.

Target phase
At the element that originates the event (`event.target`)

EVENT PROPAGATION EXAMPLE

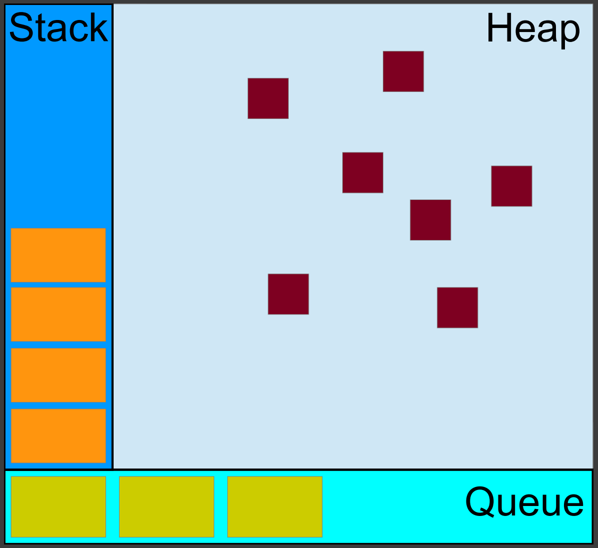


Two event listeners are set up on `tr` and `table` registered with the default bubbling phase.

Event propagation chronology:

1. The user clicks on the `td` with text "Over the river, Charlie"
2. The listener attached to the `tr` receives the event
3. The listener attached to the `table` receives the event

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.

EVENT HANDLING MODELS

DOM level 0 inline: only 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: only one event handler per element

```
var makeGreen = function () {
  document.getElementById('d1').style.color = 'green' };

document.getElementById('d1').onclick = makeGreen;  //assign an event handler
//document.getElementById('d1').onclick = null;  //remove event handler
```

DOM level 2: multiple event handlers per element

```
var setColor = function () {
  document.getElementById('d2').style.color = 'yellow';
  document.getElementById('pre2').style.fontSize = '14px';
};

var setBackground = function () { document.getElementById('#d2').style.backgroundColor = 'blue'; };

var div = document.getElementById('d2');
//target.addEventListener(type, listener[, useCapture]);
div.addEventListener("click", setColor, true);  //handler called during capture
div.addEventListener("click", setBackground);  //handler called during bubbling

//target.removeEventListener(type, listener[, useCapture]);
```

HOVER 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
#highlight-me:hover {
  background-color: blue;
  color: yellow;
}
```

HANDLING EVENTS WITH D3

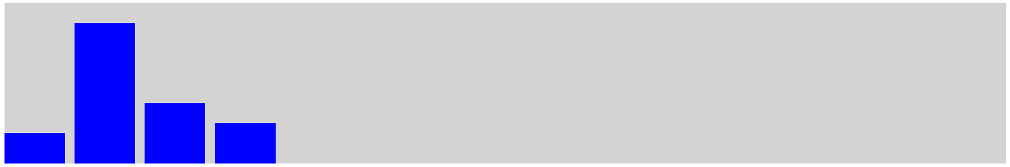
```
selection.on(typenames[, listener[, capture]]); //sets DOM level 2 event listeners
//typenames: event types, listener: handlers, capture: use bubbling (capture = undefined)
```

```
<div style="margin: 20px; border: 2px solid;" id="div00">div00</div>

<div style="display: inline-block; border: 2px solid; width: 350px;" id="div01"><b>div01 ⌘ + click</b></div>
<div style="display: inline-block; border: 2px solid; width: 350px;" id="div02"><b>div002 click</b></div>

<script>
  var flag = false;

  d3.select("#div01")
    .on("click", function (event) {
      console.log(event.target);  //event target is div with id div001 where the event originates
      if (event.metaKey) {  //check if event has meta key
        d3.select("#div00")  //select and modify div000
          .style("background-color", () => {
            flag = !flag; return flag ? "blue" : "darkgreen";
          });

        d3.select(this) //this == element for this event listener
          .html('<b>div01 ⌘ + click</b>'); //sets innerHTML (markup of the node's contents)
      }
    })

  d3.select("#div02")
    .on("mouseout click", function () {  //multiple typenames allowed separated with spaces
      d3.select(this) //this == element for this event listener
        .text('<b>div02 click</b>'); //sets text
        d3.select("#div00")  //select and modify div000
          .style("color", () => {
            flag = !flag; return flag ? "yellow" : "white";
          });
    })
    .on('mousedown', function(e) { e.preventDefault(); }); //prevent double-click selection
</script>
```

div00

div01 ⌘ + click | **div002 click**

EXAMPLE OF HANDLING EVENTS IN D3



```
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

Update the data with a combination of JavaScript array and d3 array methods

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

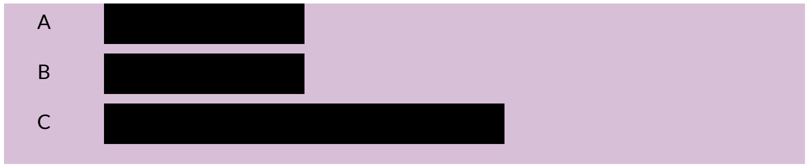
d3-array methods

- 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`

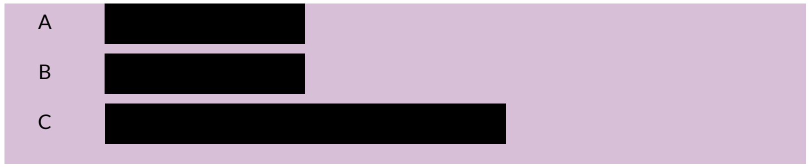
OUTLINE

- 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

ENTER SELECTION DATA JOIN LIMITATIONS



→



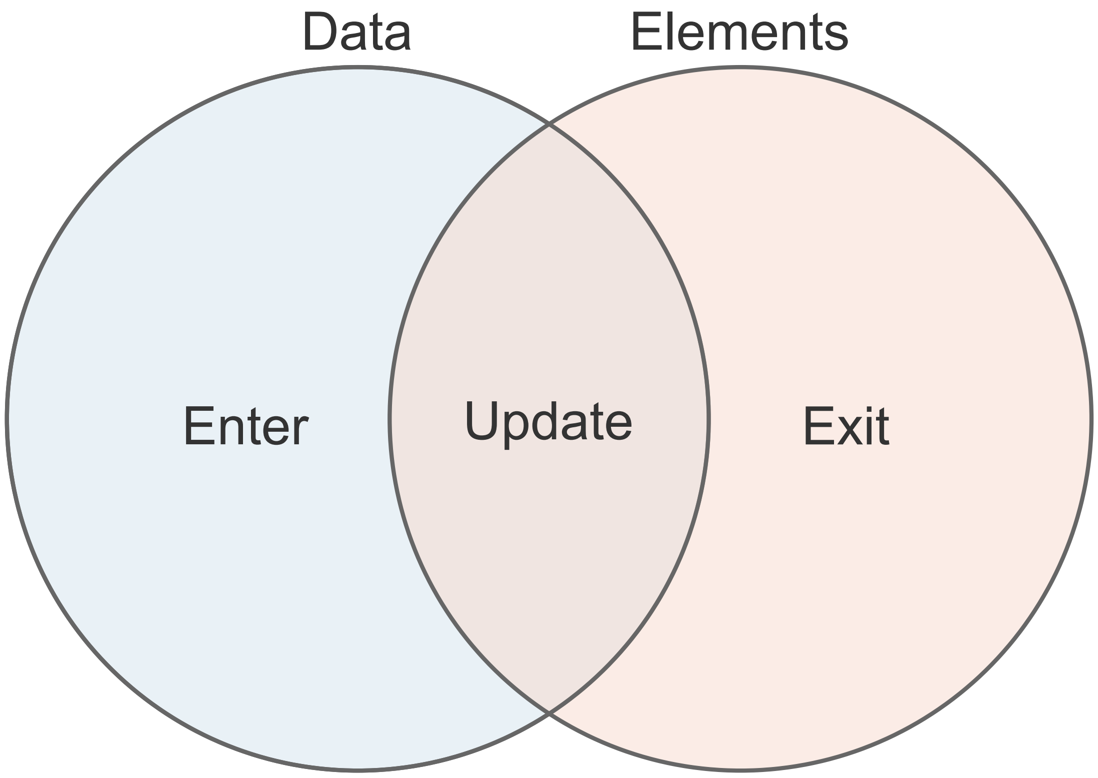
```
//We start with some data...
var data = [{item: 'A', value: 1},
            {item: 'B', value: 1},
            {item: 'C', value: 2}];

var svg = d3.select("#svg");

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

enter.append("rect")
  .attr('x', 50)
  .attr('y', (d, i) => i * 25)
  .attr('width', d => 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', (d, i) => 10 + i * 25)
  .text(d => d.item);
```

→

!?

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

var svg = d3.select("#svg");

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

enter.append("rect")
  .attr('x', 50)
  .attr('y', (d, i) => i * 25)
  .attr('width', d => d.value * 100)
  .attr('height', 20)

enter.append("text")
  .attr('font-size', '40px')
  .attr('text-anchor', 'middle')
  .attr('alignment-baseline', 'middle')
  .attr("x", 20)
  .attr("y", (d, i) => 10 + i * 25)
  .text(d => d.item);
```

DATA JOIN SELECTIONS



$$\text{Enter} : \text{size(data)} > \text{size(elements)}$$
$$\text{Update} : \text{size(data)} = \text{size(elements)}$$
$$\text{Exit} : \text{size(data)} < \text{size(elements)}$$

GENERAL UPDATE PATTERN

Select elements: `selectAll(selector)`
Selects all elements that match the specified selector string. *“update”* selection.

1. UPDATE: `selection.data([data[, key]])`
Binds the specified array of data with the selected elements, returning a new selection that represents the *“update selection”*.
Use the *“update selection”* to update old elements as needed.

2. ENTER: `selection.enter()`
Returns the *“enter selection”*: placeholder nodes for each datum that had no corresponding DOM element in the selection.
Use the *“enter selection”* to create new elements as needed.

3. ENTER + UPDATE: `selection.merge()`
Returns a new selection merging this selection with the specified other selection or transition.
Apply operations to the *“enter selection”* and *“update selection”* at once.

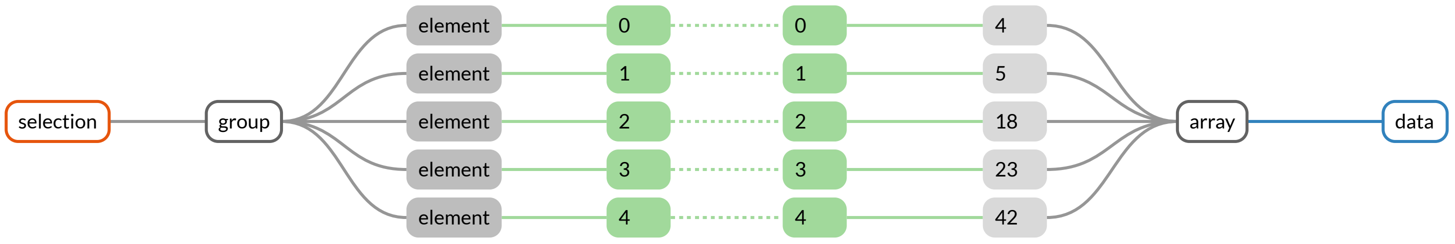
3. EXIT: `selection.exit()`
Returns the exit selection: existing DOM elements in the selection for which no new datum was found.
Use the *“exit selection”* to remove old elements.

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

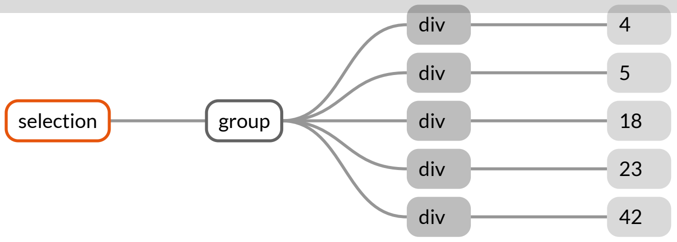
selection.data(data)

Maps data according to data and selection order

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



```
d3.selectAll('div').data(dataset);
```

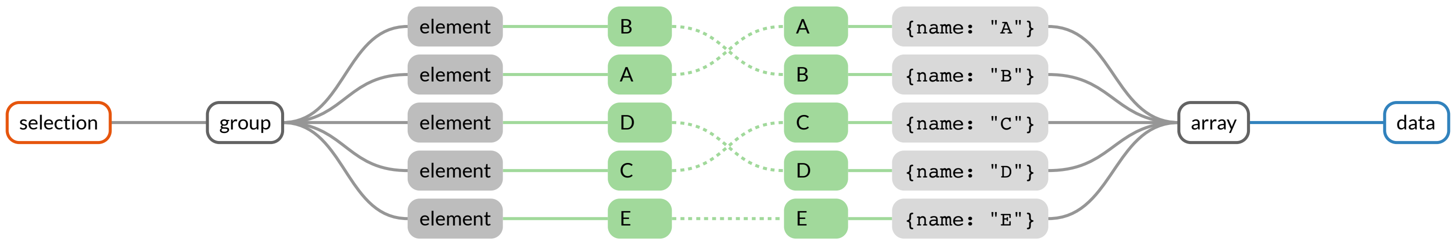


Mike Bostok's How Selections Work

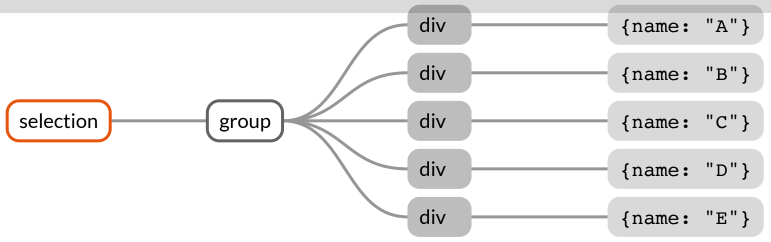
selection.data(data, key)

With keys updates can occur anywhere in the data array

```
var data = [{name: "A", value: 4},
  {name: "B", value: 5},
  {name: "C", value: 18},
  {name: "D", value: 23},
  {name: "E", value: 42}];
```



```
d3.selectAll("div").data(dataset, d => d.name);   //specify keys
```



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 domain to new data:

```
var data = [{name: "A", value: 4},
   {name: "B", value: 5},
   {name: "C", value: 18},
   {name: "D", value: 23},
   {name: "E", value: 42}];

//numerical scale
y.domain([d3.min(data, d => d.value),
          d3.max(data, d => d.value)])
 .range([0, 600]);

//ordinal scale
x.domain(data.map(d => d.name))
 .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

CSS ANIMATED TRANSITIONS WITH HTML

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>
```

CSS ANIMATED TRANSITIONS WITH SVG

```
<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.

```css
/* 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");
    });
</script>
```

Only one transition at the time per element is possible!