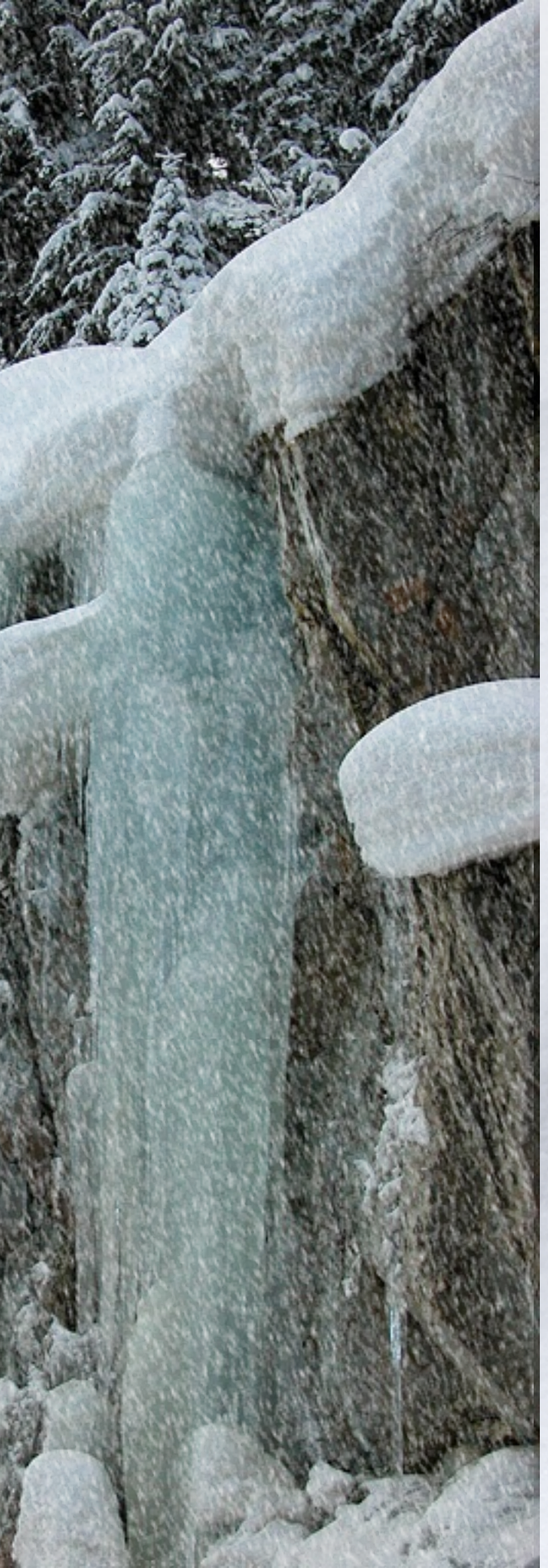
Lecture 6
Advanced JavaScript and DOM

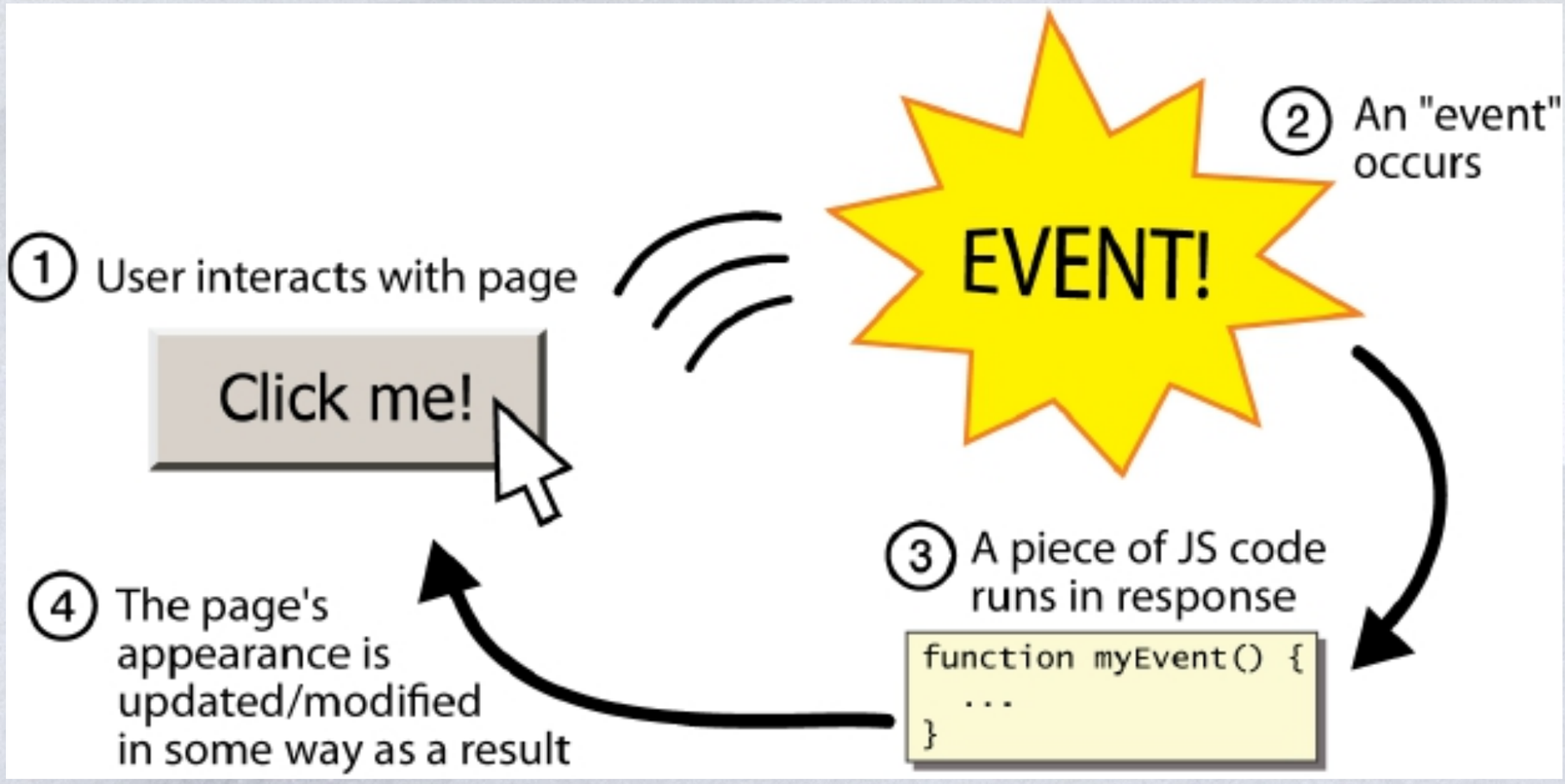
Play



Outline

- Observer pattern
- DOM 2 event flow
- Event handling
- Global DOM Objects
- Unobtrusive JavaScript
- Client-side validation
- Scope and Closure

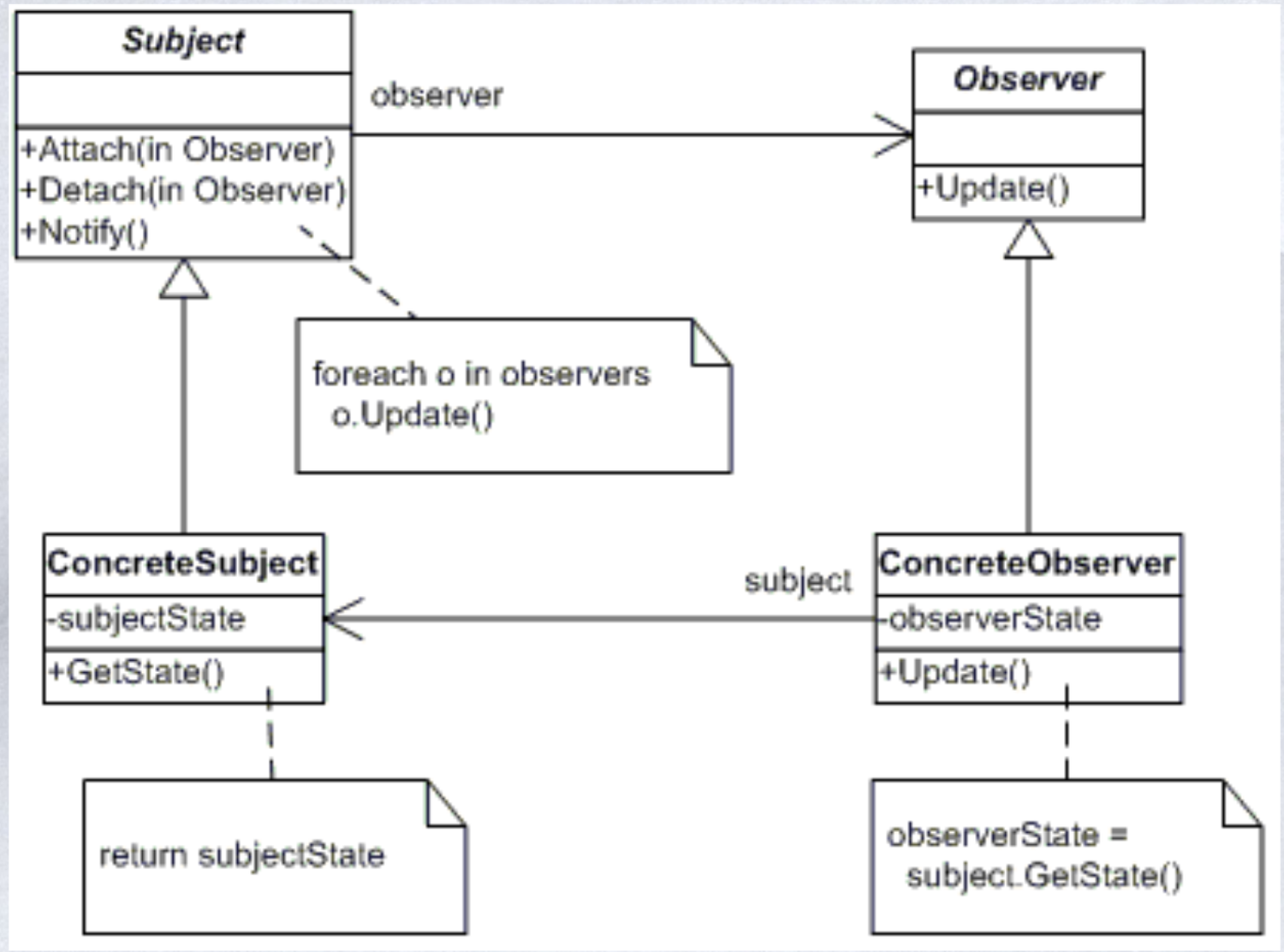
Event-driven programming



- event-driven programming: writing programs driven by user events

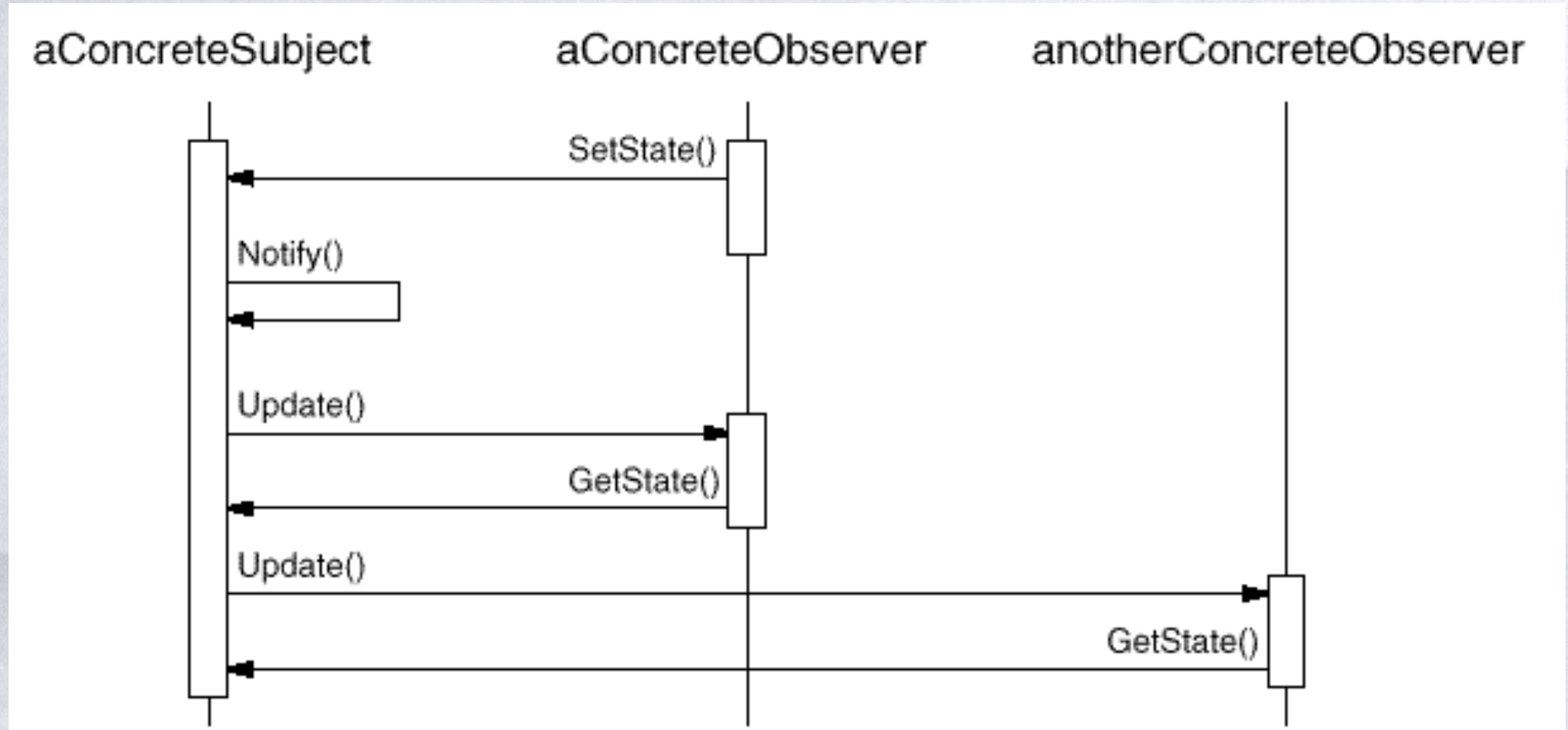
Observer pattern

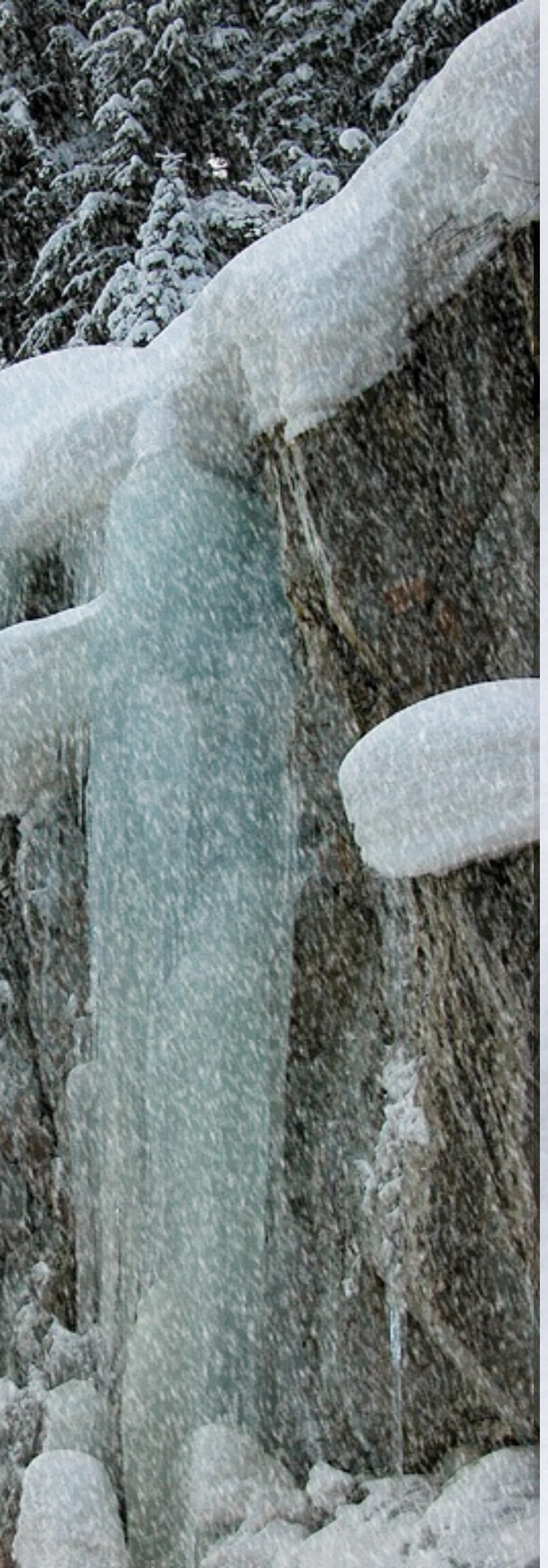
- Events makes a subject may have multiple observer queues



Observer pattern

- Observer pattern is the base for event-driven programming
- How can we apply observer pattern on the complex DOM tree?





Outline

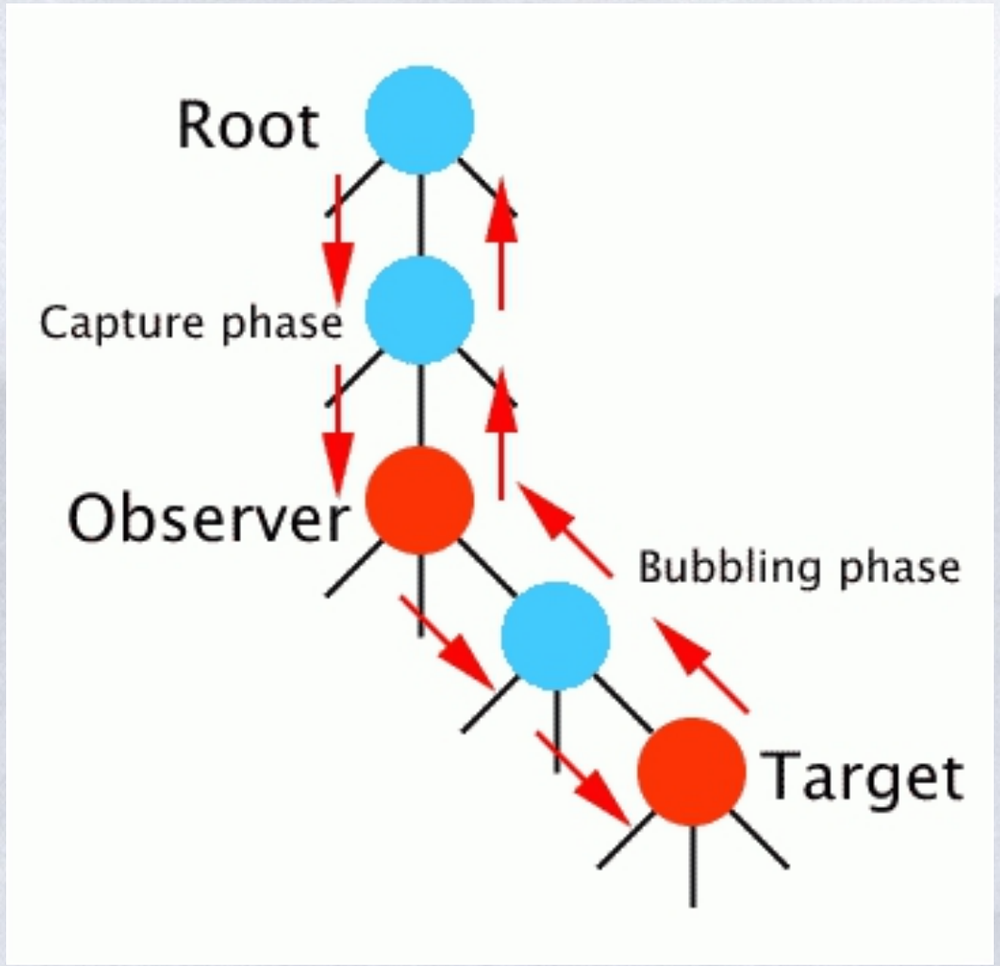
- Observer pattern
- DOM 2 event flow
- Event handling
- Global DOM Objects
- Unobtrusive JavaScript
- Client-side validation
- Scope and Closure

Event flow

- each event has a target, which can be accessed via event

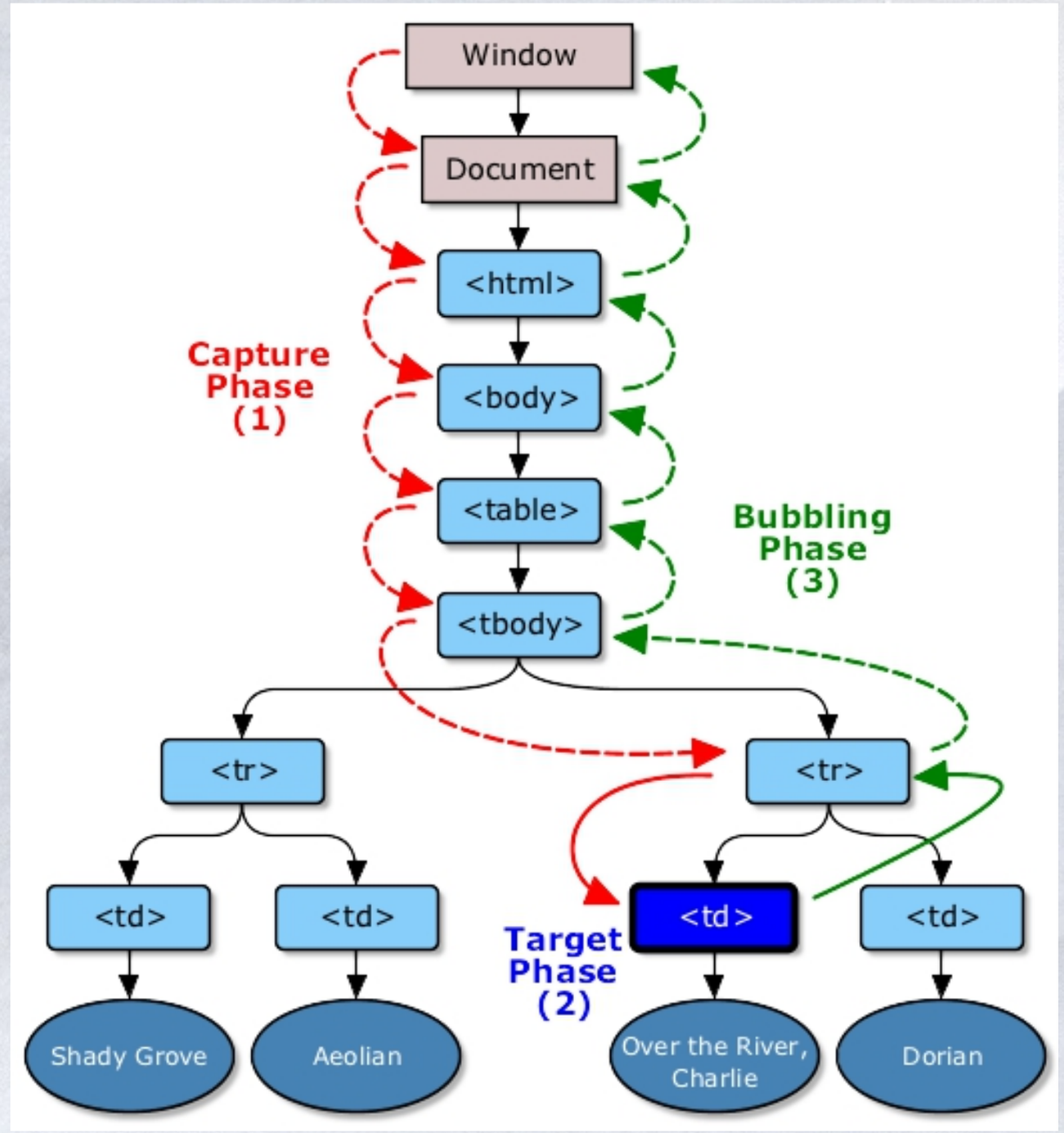
```
element.onclick = handler(e);
function handler(e){
        if(!e) var e = window.event;
        // e refers to the event
        // see detail of event
        var original = e.eventTarget;
}
```

- each event originates from the browser, and is passed to the DOM
- DOM propagates the event in 3 phases:
  - capture phase, target phase, bubbling phase (some events have no bubbling phase, i.e. load event of document element.)
  - register a capture phase handler: (IE can't do this)
    element.addEventListener('click',handler,true);



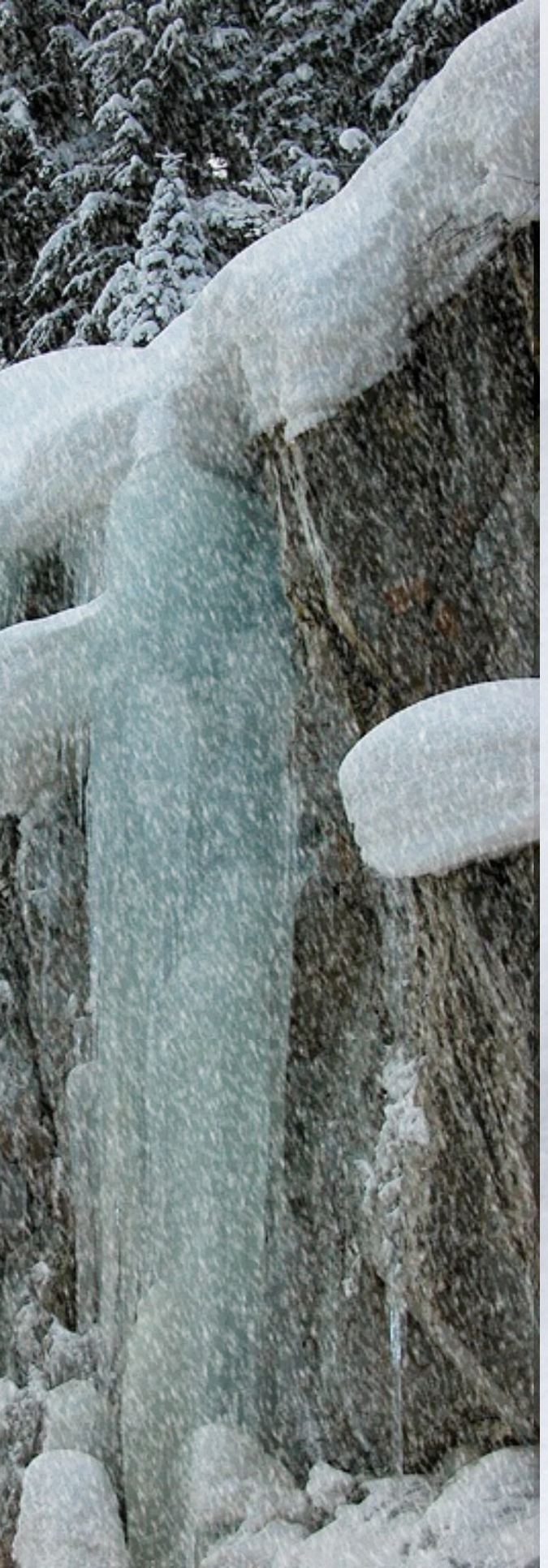
Event flow

- stopping event propagation
  - by throwing any exception inside an event handler
  - by calling event.stopPropagation(); inside a handler
- event cancelation
  - canceling default action (i.e. navigating to a new page when clicking on a hyperlink):
  - event.preventDefault();



Event handler registration

- inline:
  - <a href="somewhere.html" onClick="doSomething()">
- traditional:
  - element.onclick = doSomething;
- DOM 2:
  - element.addEventListener('click', doSomething, false);
- IE: (evil enough!)
  - element.attachEvent('onclick', doSomething);
- Prototype: (prefered)
  - Event.observe('target', 'click', doSomething);
  - document.observe('dom:loaded', doSomething);



Outline

- Observer pattern
- DOM 2 event flow
- Event handling
- Global DOM Objects
- Unobtrusive JavaScript
- Client-side validation
- Scope and Closure

DOM 2 Event Types

- UI event types:
  - DOMFocusIn, DOMFocusOut, DOMActivate
- Mouse event types:
  - click, mousedown, mouseup, mouseover, mousemove, mouseout
- Key event types: (not in DOM 2, but will in DOM 3)
- Mutation events:
  - DOMSubtreeModified, DOMNodeInserted, ...
- HTML event types:
  - load, unload, abort, error, select, change, submit, reset, focus, blur, resize, scroll

Useful event types

- problem: events are tricky and have incompatibilities across browsers reasons:
  - fuzzy W3C event specs;
  - IE disobeying web standards;
  - etc.
- solution: Prototype includes many event-related features and fixes

abort	blur	change	click	dbclick	error
keydown	keypress	keyup	load	mousedown	mousemove
mouseover	mouseup	reset	resize	select	submit
focus	mouseout	unload			

Attaching event handlers the Prototype way

- to use Prototype's event features, you must attach the handler using the DOM element object's observe method (added by Prototype)
- pass the event of interest and the function to use as the handler
- handlers must be attached this way for Prototype's event features to work
- observe substitutes for addEventListener and attachEvent (IE)

```
element.onevent = function;
element.observe("event", "function");
```
JS

```
// call the playNewGame function when the Play button is clicked
$("play").observe("click", playNewGame);
```
JS

Attaching multiple event handlers with $$

- you can use $$ and other DOM walking methods to unobtrusively attach event handlers to a group of related elements in your window.onload code

```
// listen to clicks on all buttons with class "control" that
// are directly inside the section with ID "game"
window.onload = function() {
  var gameButtons = $$("#game > button.control");
  for (var i = 0; i < gameButtons.length; i++) {
    gameButtons[i].observe("click", gameButtonClick);
  }
};

function gameButtonClick() { ... }
```
JS

The Event object

- Event handlers can accept an optional parameter to represent the event that is occurring. Event objects have the following properties / methods:

```
function name(event) {
  // an event handler function ...
}
```
JS

method/property name	description
type	what kind of event, such as "click" or "mousedown"
element()*	the element on which the event occurred
stop()**	cancels an event
stopObserving()	removes an event handler

* replaces non-standard srcElement and which properties
** replaces non-standard return false, stopPropagation

Mouse events

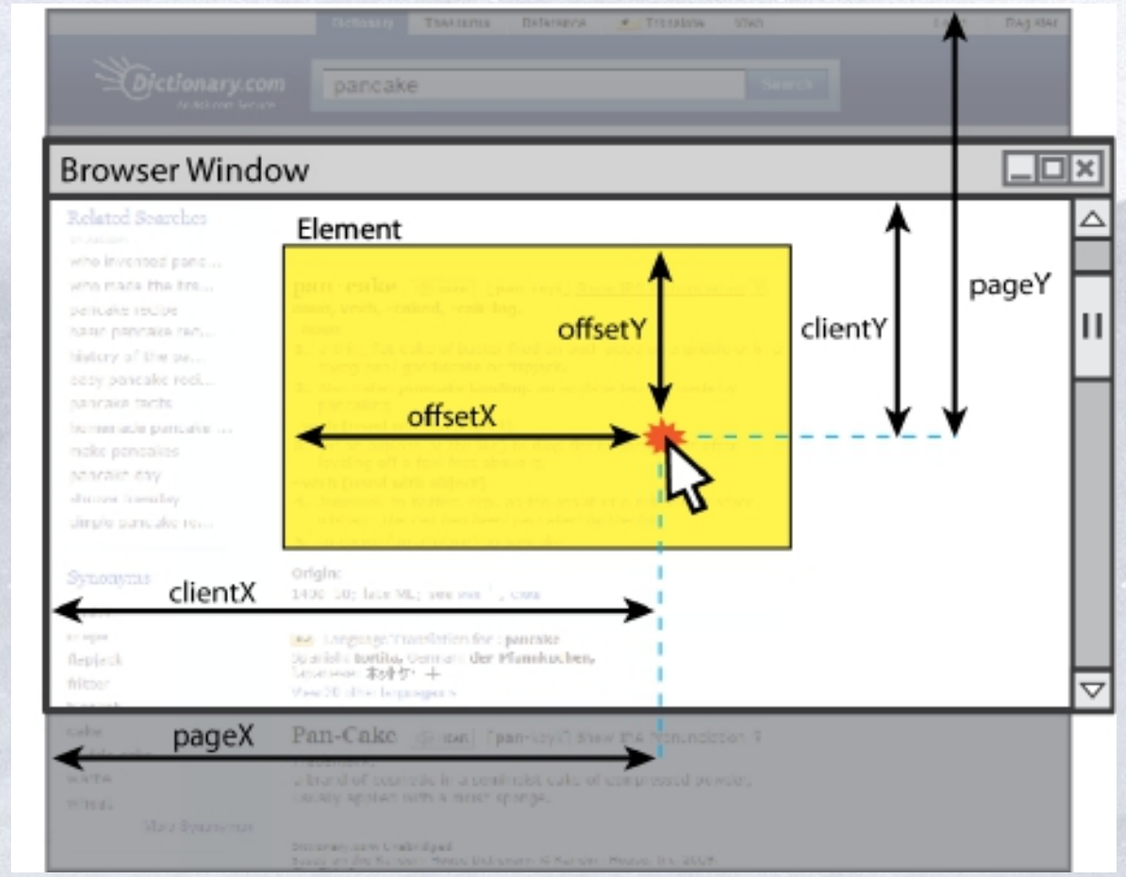
click	
click	user presses/releases mouse button on this element
dbclick	user presses/releases mouse button twice on this element
mousedown	user presses down mouse button on this element
mouseup	user releases mouse button on this element

movement	
mouseover	mouse cursor enters this element's box
mouseout	mouse cursor exits this element's box
mousemove	mouse cursor moves around within this element's box

Mouse event objects

- The event parameter passed to a mouse event handler has the following properties:

property/method	description
clientX, clientY	coordinates in browser window
screenX, screenY	coordinates in screen
offsetX, offsetY	coordinates in element
pointerX(), pointerY()*	coordinates in entire web page
isLeftClick( ) **	true if left button was pressed



* replaces non-standard properties pageX and pageY
** replaces non-standard properties button and which

Mouse event example

```HTML
<pre id="target">Move the mouse over me!</pre>
```

```JS
window.onload = function() {
  $("target").observe("mousemove", showCoords);
};

function showCoords(event) {
  this.innerHTML =
      "pointer: (" + event.pointerX() + ", " + event.pointerY() + ")\n"
    + "screen : (" + event.screenX + ", " + event.screenY + ")\n"
    + "client : (" + event.clientX + ", " + event.clientY + ")";
}
```

```output
pointer: (1333, 471)
screen : (-100, 734)
client : (1333, 471)
```

Page/window events

name	description
load	the browser loads the page
unload	the browser exits the page
resize	the browser window is resized
contextmenu	the user right-clicks to pop up a context menu
error	an error occurs when loading a document or an image

- The above events can be handled on the global window object. Also:

```
// best way to attach event handlers on page load
window.onload = function() { ... };
document.observe("dom:loaded", function() {
  $("orderform").observe("submit", verify);
});
```

Form events

event name	description
submit	form is being submitted
reset	form is being reset
change	the text or state of a form control has changed

```
window.observe("load", function() {
  $("orderform").observe("submit", verify);
});

function verify(event) {
  if ($("zipcode").value.length < 5) {
    event.stop();          // cancel form submission unless
  }                        // zip code is 5 chars long
}
```

Prototype and forms

```
$("id")["name"]
```

- gets parameter with given name from form with given id

```
$F("id")
```

- $F returns the value of a form control with the given id

```
var name = $F("username");
if (name.length < 4) {
  $("username").clear();
  $("login").disable();
}
```

- other form control methods:

activate	clear	disable	enable
focus	getValue	present	select

Keyboard/text events

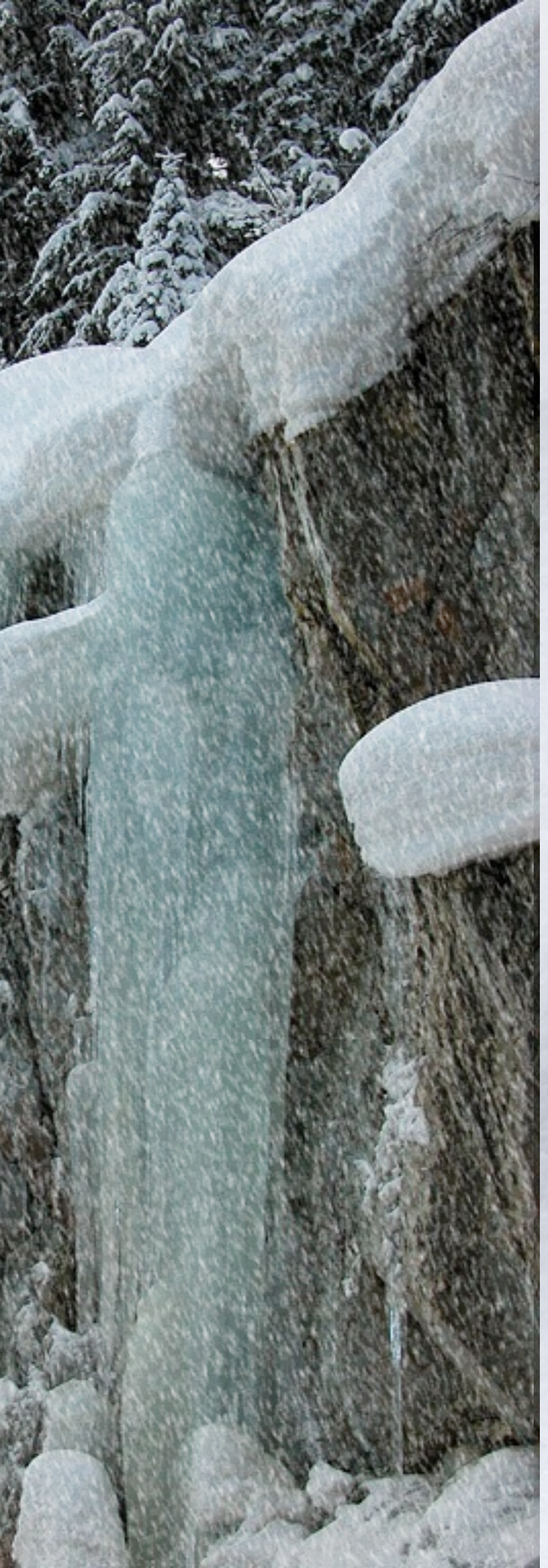
name	description
keydown	user presses a key while this element has keyboard focus
keyup	user releases a key while this element has keyboard focus
keypress	user presses and releases a key while this element has keyboard focus
focus	this element gains keyboard focus
blur	this element loses keyboard focus
select	this element's text is selected or deselected)

- focus: the attention of the user's keyboard (given to one element at a time)

Key event objects

property name	description
keycode	ASCII integer value of key that was pressed (convert to char with String.fromCharCode)
altKey, ctrlkey,shiftKey	true if Alt/Ctrl/Shift key is being held

Prototype's key code constants			
Event.KEY_BACKSPACE	Event.KEY_DELETE	Event.KEY_DOWN	Event.KEY_END
Event.KEY_ESC	Event.KEY_HOME	Event.KEY_LEFT	Event.KEY_PAGEDOWN
Event.KEY_PAGEUP	Event.KEY_RETURN	Event.KEY_RIGHT	Event.KEY_TAB
Event.KEY_UP			



Outline

- Observer pattern
- DOM 2 event flow
- Event handling
- Global DOM Objects
- Unobtrusive JavaScript
- Client-side validation
- Scope and Closure

Global DOM objects

- Every browsers' Javascript program can refer to the following global objects:

name	description
document	current HTML page and its content
history	list of pages the user has visited
location	URL of the current HTML page
navigator	info about the web browser you are using
screen	info about the screen area occupied by the browser
window	the browser window

The window object

- the entire browser window; the top-level object in DOM hierarchy
- technically, all global code and variables become part of the window object
- properties:
  - document, history, location, name
- methods:
  - alert, confirm, prompt (popup boxes)
  - setInterval, setTimeout clearInterval, clearTimeout (timers)
  - open, close (popping up new browser windows)
  - blur, focus, moveBy, moveTo, print, resizeBy, resizeTo, scrollBy, scrollTo

The document object

- the current web page and the elements inside it
- properties:
  - anchors, body, cookie, domain, forms, images, links, referrer, title, URL
- methods:
  - getElementById
  - getElementsByName
  - getElementsByTagName
  - close, open, write, writeln

The location object

- the URL of the current web page
- properties:
  - host, hostname, href, pathname, port, protocol, search
- methods:
  - assign, reload, replace

The navigator object

- information about the web browser application
- properties:
  - appName, appVersion, browserLanguage, cookieEnabled, platform, userAgent
- Some web programmers examine the navigator object to see what browser is being used, and write browser- specific scripts and hacks:

```
if (navigator.appName === "Microsoft Internet Explorer") { ...
```
JS

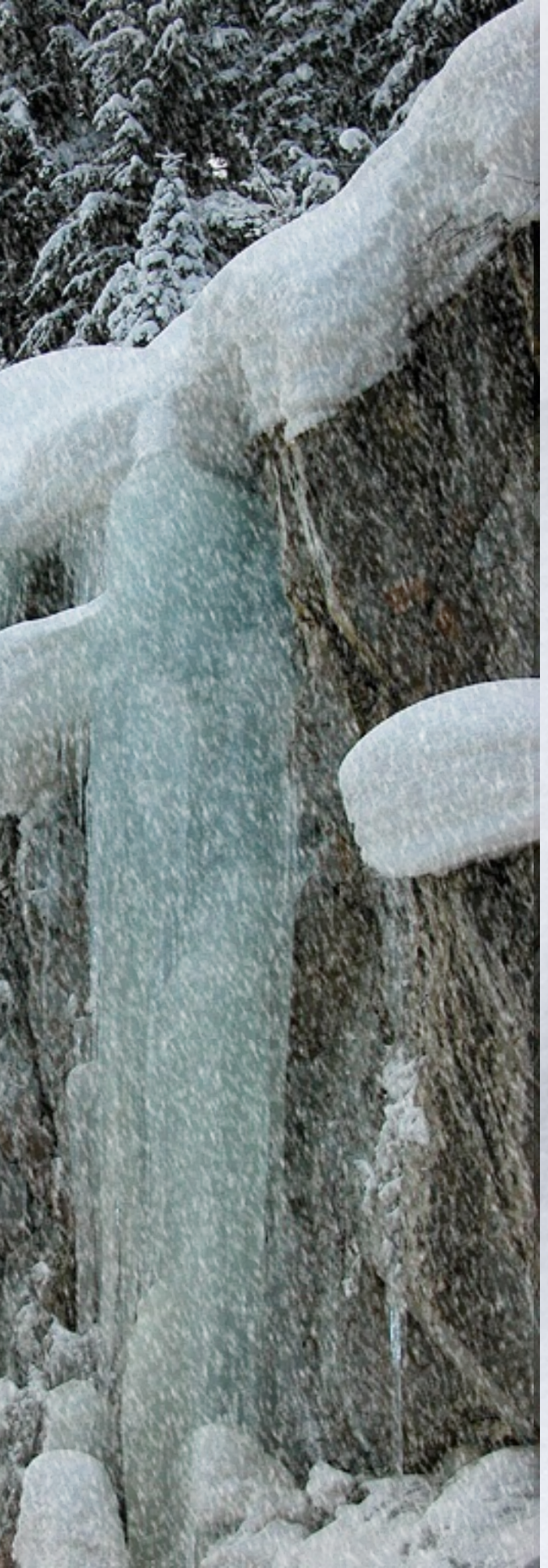
  - (this is poor style; you should not need to do this)

The screen object

- information about the client's display screen
- properties:
  - availHeight, availWidth, colorDepth, height, pixelDepth, width

The history object

- the list of sites the browser has visited in this window
- properties:
  - length
- methods:
  - back, forward, go
- sometimes the browser won't let scripts view history properties, for security



Outline

- Observer pattern
- DOM 2 event flow
- Event handling
- Global DOM Objects
- Unobtrusive JavaScript
- Client-side validation
- Scope and Closure

Unobtrusive JavaScript

- JavaScript event code seen previously was obtrusive, in the HTML; this is bad style
- now we'll see how to write unobtrusive JavaScript code
  - HTML with minimal JavaScript inside
  - uses the DOM to attach and execute all JavaScript functions
- allows separation of web site into 3 major categories:
  - content (HTML) - what is it?
  - presentation (CSS) - how does it look?
  - behavior (JavaScript) - how does it respond to user interaction?

Obtrusive event handlers (bad)

- this is bad style (HTML is cluttered with JS code)
- goal: remove all JavaScript code from the HTML body

```
<button id="ok" onclick="okayClick();">OK</button>
```
HTML

```
// called when OK button is clicked
function okayClick() {
  alert("booyah");
}
```
JS

OK

output

Attaching an event handler in JavaScript code

```
// where element is a DOM element object
element.event = function;
```
JS

```
$("ok").onclick = okayClick;
```
JS

OK

output

- it is legal to attach event handlers to elements' DOM objects in your JavaScript code
  - notice that you do not put parentheses after the function's name
- this is better style than attaching them in the HTML
- Where should we put the above code?

When does my code run?

```
  <head>
    <script src="myfile.js" type="text/javascript"></script>
  </head>

  <body> ... </body>
```
HTML

```
// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);
```
JS

- your file's JS code runs the moment the browser loads the script tag
  - any variables are declared immediately
  - any functions are declared but not called, unless your global code explicitly calls them

A failed attempt at being unobtrusive

```HTML
<head>
  <script src="myfile.js" type="text/javascript"></script>
</head>

<body>
  <div><button id="ok">OK</button></div>
```

```JS
// global code
$("ok").onclick = okayClick;   // error: $("ok") is null
```

- problem: global JS code runs the moment the script is loaded
- script in head is processed before page's body has loaded
  - no elements are available yet or can be accessed yet via the DOM
- we need a way to attach the handler after the page has loaded...

The window.onload event

```
// this will run once the page has finished loading
function functionName() {
   element.event = functionName;
   element.event = functionName;
   ...
}

window.onload = functionName;    // global code
```
JS

- we want to attach our event handlers right after the page is done loading
  - there is a global event called window.onload event that occurs at that moment
- in window.onload handler we attach all the other handlers to run when events occur

An unobtrusive event handler

```html
<!-- look Ma, no JavaScript! -->
<button id="ok">OK</button>
```
HTML

```js
// called when page loads; sets up event handlers
function pageLoad() {
  $("ok").onclick = okayClick;
}

function okayClick() {
  alert("booyah");
}

window.onload = pageLoad;  // global code
```
JS

output

Common unobtrusive JS errors

- many students mistakenly write () when attaching the handler

```
window.onload = pageLoad();
window.onload = pageLoad;

okButton.onclick = okayClick();
okButton.onclick = okayClick;
```
JS

- event names are all lowercase, not capitalized like most variables

```
window.onLoad = pageLoad;
window.onload = pageLoad;
```
JS

The keyword this

```
this.fieldName                          // access field
this.fieldName = value;                 // modify field

this.methodName(parameters);        // call method
```
JS

- all JavaScript code actually runs inside of an object
- by default, code runs inside the global window object
  - all global variables and functions you declare become part of window
- the this keyword refers to the current object

Event handler binding

- event handlers attached unobtrusively are bound to the element
- inside the handler, that element becomes this (rather than the window)

```
function pageLoad() {
  $("ok").onclick = okayClick;    // bound to okButton here
}

function okayClick() {            // okayClick knows what DOM object
  this.innerHTML = "booyah";      // it was called on
}

window.onload = pageLoad;
```

JS

OK

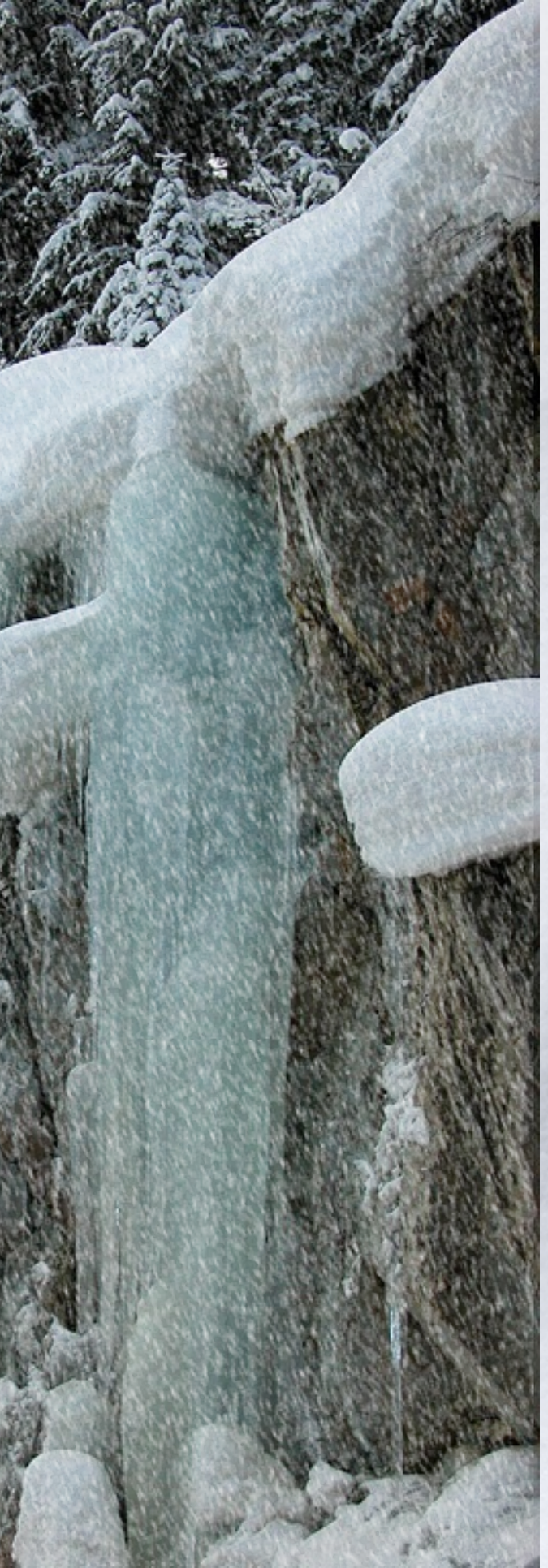
output

Fixing redundant code with this

```HTML
<fieldset>
  <label><input type="radio" name="ducks" value="Huey"  /> Huey</label>
  <label><input type="radio" name="ducks" value="Dewey" /> Dewey</label>
  <label><input type="radio" name="ducks" value="Louie" /> Louie</label>
</fieldset>
```

```JS
function processDucks() {
  if ($("huey").checked) {
    alert("Huey is checked!");
  } else if ($("dewey").checked) {
    alert("Dewey is checked!");
  } else {
    alert("Louie is checked!");
  }
  alert(this.value + " is checked!");
}
```

- if the same function is assigned to multiple elements, each gets its own bound copy



Outline

- Observer pattern
- DOM 2 event flow
- Event handling
- Global DOM Objects
- Unobtrusive JavaScript
- **Client-side validation**
- Scope and Closure

Client-side validation

- forms expose onsubmit and onreset events
- to abort a form submission, call Prototype's Event.stop on the event

```HTML
<form id="exampleform" action="http://foo.com/foo.php">
```

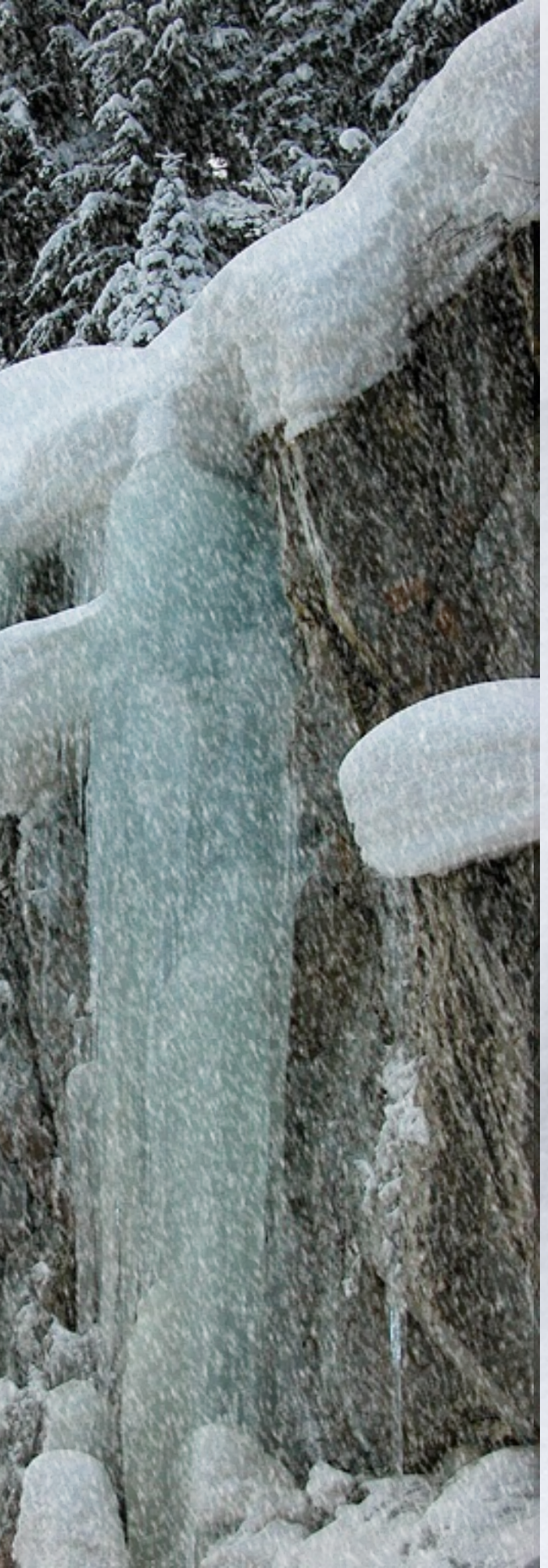
```JS
window.onload = function() {
  $("exampleform").onsubmit = checkData;
};

function checkData(event) {
  if ($("city").value == "" || $("state").value.length != 2) {
    Event.stop(event);
    alert("Error, invalid city/state.");  // show error message
  }
}
```

Replacing text with regular expressions

- string.replace(regex, "text")
  - replaces the first occurrence of given pattern with the given text
  - var str = "Qing Zang"; str.replace(/[a-z]/, "x"); //returns “Qxng Zang"
  - returns the modified string as its result; must be stored str = str.replace(/[a-z]/, "x")
- a g can be placed after the regex for a global match (replace all occurrences)
  - str.replace(/[a-z]/g, "x"); //returns “Qxxx Zxxx"
- using a regex as a filter
  - str = str.replace(/[^A-Z]+/g, ""); // turns str into "QZ"

Regular expression in JavaScript

- string.match(regex)
  - if string fits the pattern, returns the matching text; else returns null
  - can be used as a Boolean truthy/falsey test: var name = $("name").value; if (name.match(/[a-z]+/)) { ... }
- an i can be placed after the regex for a case-insensitive match
  - name.match(/Eric/i) will match “eric", “ERic", ...



Outline

- Observer pattern
- DOM 2 event flow
- Event handling
- Global DOM Objects
- Unobtrusive JavaScript
- Client-side validation
- Scope and Closure

Anonymous functions

- JavaScript allows you to declare anonymous functions
- quickly creates a function without giving it a name
- can be stored as a variable, attached as an event handler, etc.

```
function(parameters) {
  statements;
}
```
JS

Anonymous function example

```
window.onload = function() {
  var okButton = document.getElementById("ok");
  okButton.onclick = okayClick;
};

function okayClick() {
  alert("booyah");
}
```

JS

OK

output

- or the following is also legal (though harder to read and bad style):

```
window.onload = function() {
  var okButton = document.getElementById("ok");
  okButton.onclick = function() {
    alert("booyah");
  };
};
```

JS

闭包的定义

- 闭包是函数和执行它的作用域组成的综合体——《**JavaScript**权威指南》
  - 所有的函数都是闭包
- 函数可以访问它被创建时的上下文环境，称为闭包——《**JavaScript**语言精粹》
  - 内部函数比它的外部函数具有更长的生命周期
- 更简单的定义　闭包是引用了自由变量的函数

闭包 in action

```
function closure(name){
        var status = 1;
        return {
         getName:function(){ return name; },
         getStatus:function(){ return status++; }
        }
}
var a = closure('w3ctech');
alert(a.getName());
alert(a.getStatus());
alert(a.getStatus());
```

作用域

函数作用域

- 变量和参数在函数外不可见
- 变量可以在函数内任何位置定义，并在函数内任何地方可见
- 嵌套函数可以访问外部函数的参数和变量

作用域示例一

```
function func_scope(){
    var x = 1;
    if (true) {
        var x = 2;
        alert(x);
    }
    alert(x);
}
func_scope();
```

2
2

作用域示例二

```
function a(){
   var name = 'w3ctech';
   return b();
   function b(){ return name; }
}
alert(a());
```

w3ctech

自由变量

- 自由变量是作用域可以导出到外部作用域的变量
  - 函数内部变量和函数参数都可以是自由变量
  - 函数参数不包含**this**和**arguments**

自由变量示例一

```
var name = "The Window";
var object = {
    name : "My Object",
    getNameFunc : function(){
        return function(){
            return this.name;
        };
    }
};
alert(object.getNameFunc()());
```

The Window

自由变量示例二

```
var name = "The Window";
var object = {
        name : "My Object",
        getNameFunc : function(){
         var that = this;
         return function(){
             return that.name;
         };
        }
};
alert(object.getNameFunc()());
```

My Object

闭包应用场景

- 实现私有成员
- 保护命名空间
- 避免污染全局变量
- 变量需要长期驻留在内存

```
function a() {
    var i = 0;
    function b() {
        alert(++i);
    }
    return b;
}
var c = a();
c();
```

闭包与匿名函数的关系

- 闭包和匿名函数不是互斥的关系
- 有自由变量的匿名函数是闭包

推荐阅读

- **JAVASCRIPT**设计模式，**Ross Harmes & Dustin Diaz.** 人民邮电出版社.
- 维基百科：闭包 (计算机科学)
- 阮一峰：学习**Javascript**闭包**(Closure)**
- 深入理解**JavaScript**闭包**(Closure)**

Thanks!!!