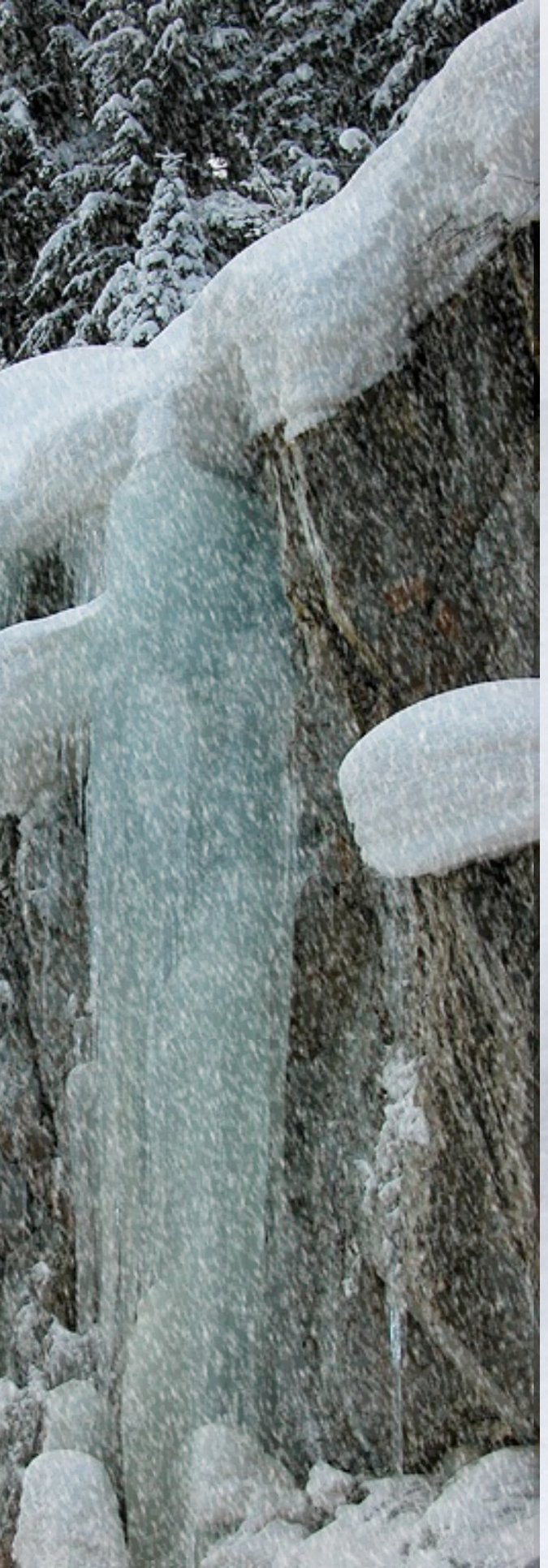
Lecture 7
Advanced JavaScript and DOM



Outline

- *Javascript Events*
- Event Types
- Event handling models
- Scope and Closure

Events

Events and event handling

- make web applications more responsive, dynamic and interactive
- programming by callbacks

JavaScript events

- allow scripts to respond to user's interactions with elements on a web page
- can initiate a modification of the page

Event-driven programming

- Event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs/threads.

- Event-driven programming is the dominant paradigm used in graphical user interfaces and other applications (e.g. JavaScript web applications) that are centered on performing certain actions in response to user input.

Event-driven programming

- In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected.
- Event-driven programs can be written in any programming language, although the task is easier in languages that provide high-level abstractions, such as closures.

Event Handlers

- Event handler
  - a callback subroutine that handles inputs received in a program (called a listener in Java and JavaScript)
  - function that is called in when an event occurs
  - typically associated with an XHTML element
  - must be registered
    - i.e., the association must be specified

What can JavaScript Do?

- Event handlers can be used to handle and verify user input, user actions, and browser actions:
  - Things that should be done every time a page loads
  - Things that should be done when the page is closed
  - Action that should be performed when a user clicks a button
  - Content that should be verified when a user inputs data
  - And more ...
- Many different methods can be used to let JavaScript work with events:
  - HTML event attributes can execute JavaScript code directly
  - HTML event attributes can call JavaScript functions
  - You can assign your own event handler functions to HTML elements
  - You can prevent events from being sent or being handled
  - And more ...

Syntax

- element.addEventListener(event, function, useCapture);
  - The first parameter is the type of the event (like "click" or "mousedown").
  - The second parameter is the function we want to call when the event occurs.
  - The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.
- The removeEventListener() method removes event handlers that have been attached with the addEventListener() method

```
element.addEventListener("click", myFunction);
element.addEventListener("click", mySecondFunction);
document.getElementById("myDiv").addEventListener("click", myFunction, true);
element.removeEventListener("mousemove", myFunction);
```

Observer pattern

- The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.
- It is mainly used to implement distributed event handling systems.
- The Observer pattern is also a key part in the familiar model–view–controller (MVC) architectural pattern.The observer pattern is implemented in numerous programming libraries and systems, including almost all GUI toolkits.
- a subset of the publish/subscribe pattern

Features

Benefits:

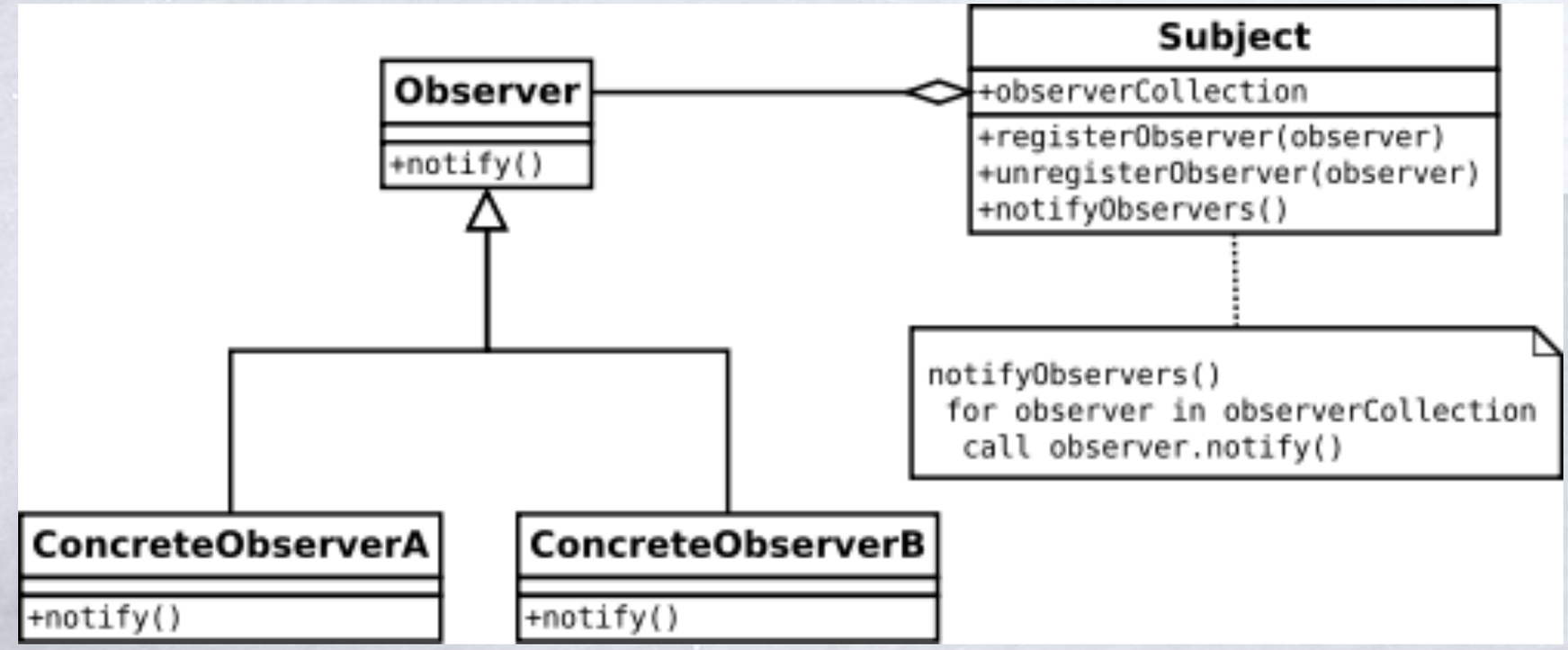
- Abstract coupling between subject and observer
- Support for broadcast communication

Notice:

- The observer pattern can cause memory leaks, known as the lapsed listener problem,

Observer pattern

- Events makes a subject may have multiple observer queues



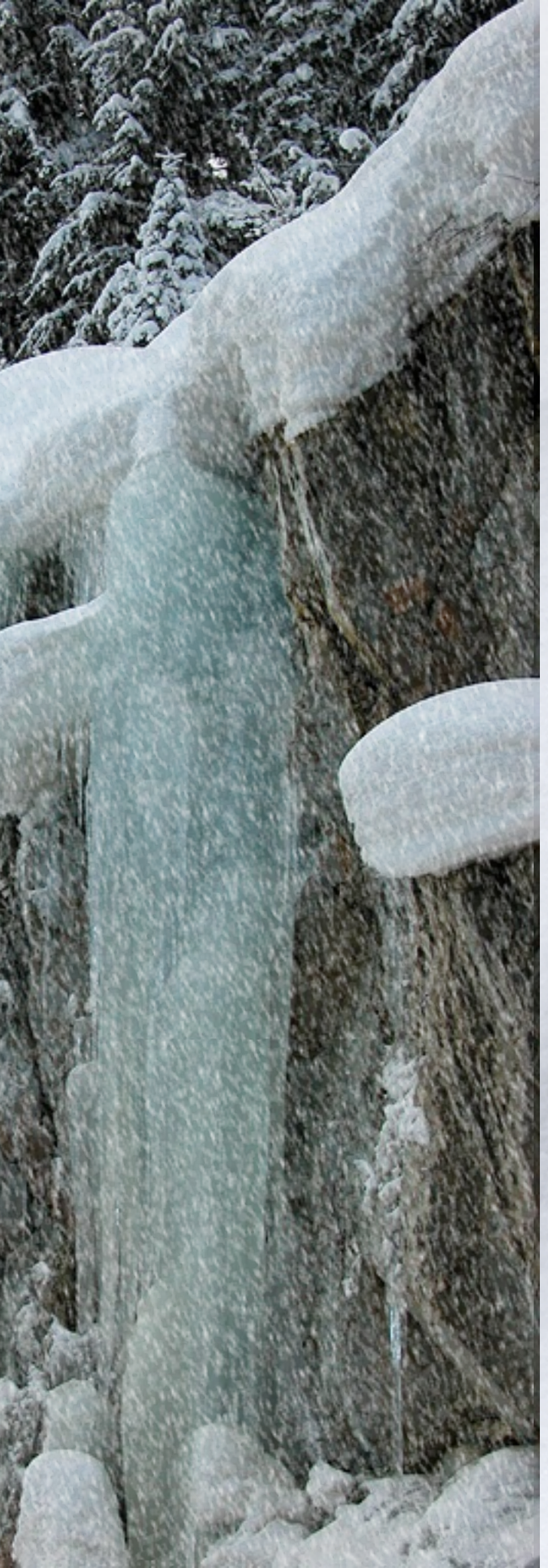
Attaching event handlers the Prototype way

- 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)

```
Event.observe(window,"load",myFunction);
```

Attaching event handlers the jQuery way

```
var hiddenBox = $( "#banner-message" );
$( "#button-container button" ).on( "click", function( event ) {
        hiddenBox.show();
       });
```



Outline

- Javascript Events
- Event Types
- Event handling models
- 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

HTML5 new events

- audio, video
  - canplay, playing, suspend,.....
- drag/drop
- history
- new form events
  - invalid
- offline,online.....
- message

Touchscreen and Mobile Events

- The widespread adoption of powerful mobile devices, particularly those with touchscreens, has required the creation of new categories of events.
- In many cases, touchscreen events are mapped to traditional event types such as click and scroll. But not every interaction with a touchscreen UI emulates a mouse, and not all touches can be treated as mouse events.
  - gesturestart, gestureend

Useful event types

- problem: events are tricky and have incompatibilities across browsers reasons:
  - fuzzy W3C event specs;
  - IE disobeying web standards;
  - etc.

The Event object

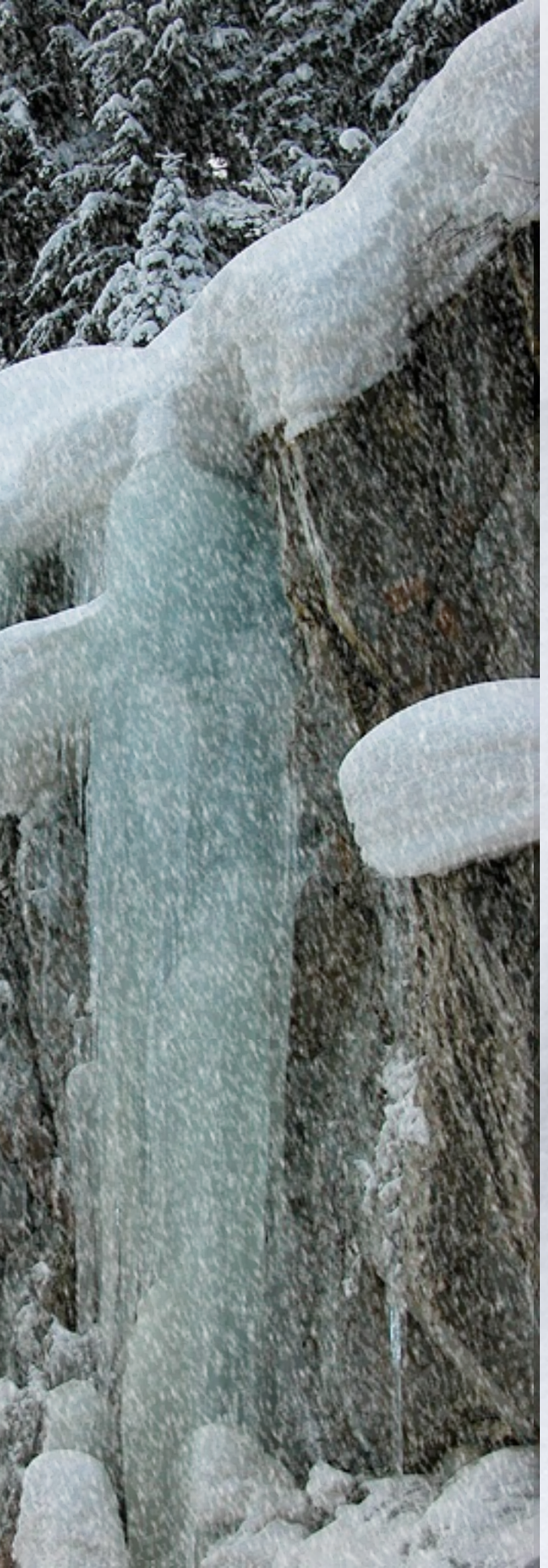
- Event handlers can accept an optional parameter to represent the event that is occurring. Event objects have the following properties / methods:

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

method/property name	description
type	Returns the name of the event, such as "click" or "mousedown"
currentTarget	Returns the element whose event listeners triggered the event
preventDefault()	Cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur
stopPropagation()	Prevents further propagation of an event during event flow

Mouse events

Event	Description	DOM
onclick	The event occurs when the user clicks on an element	2
ondblclick	The event occurs when the user double-clicks on an element	2
onmousedown	The event occurs when the user presses a mouse button over an element	2
onmouseenter	The event occurs when the pointer is moved onto an element	2
onmouseleave	The event occurs when the pointer is moved out of an element	2
onmousemove	The event occurs when the pointer is moving while it is over an element	2
onmouseover	The event occurs when the pointer is moved onto an element, or onto one of its children	2
onmouseout	The event occurs when a user moves the mouse pointer out of an element, or out of one of its children	2
onmouseup	The event occurs when a user releases a mouse button over an element	2



Outline

- Javascript Events
- Event Types
- **Event handling models**
- Scope and Closure

DOM Level 0

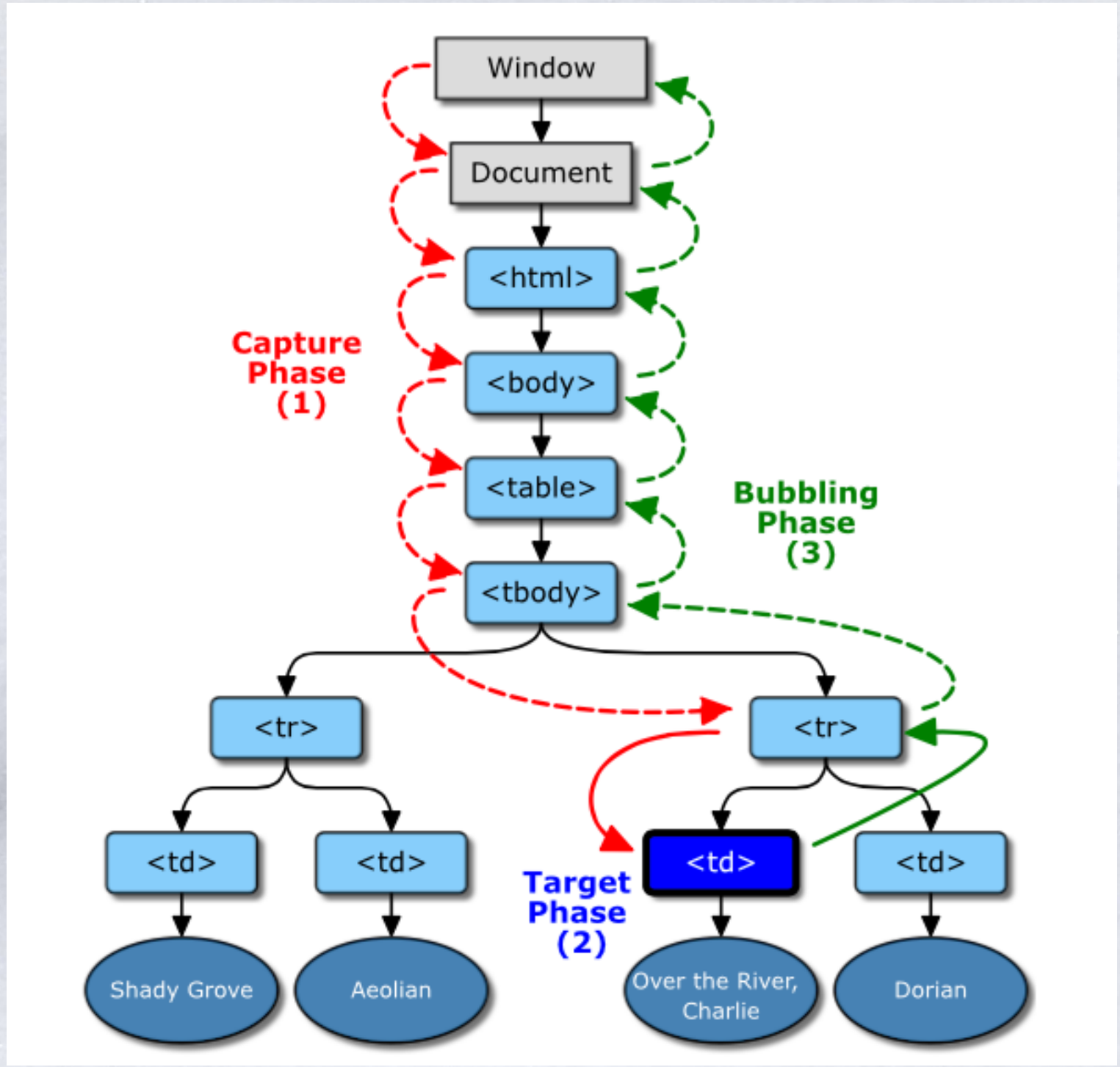
- This event handling model was introduced by Netscape Navigator, and remains the most cross-browser model as of 2005.There are two model types: inline model and traditional model.
  - Inline model: event handlers are added as attributes of elements.
  - Traditional model: event handlers can be added/removed by scripts.

  Like the inline model, each event can only have one event handler registered. The event is added by assigning the handler name to the event property of the element object. To remove an event handler, simply set the property to null:

```
<p>Hey <a href="http://
www.example.com"
onclick="triggerAlert('Joe'); return
false;">Joe</a>!</p>
<script>
    function triggerAlert(name) {
        window.alert("Hey " + name);
    }
</script>
```

```
<script>
    var triggerAlert = function () {
                        window.alert("Hey Joe”);};

    // Assign an event handler
     document.onclick = triggerAlert;
     // Assign another event handler
     window.onload = triggerAlert;
     // Remove the event handler that was just assigned
     window.onload = null;
</script>
```

DOM event flow



Event Phases

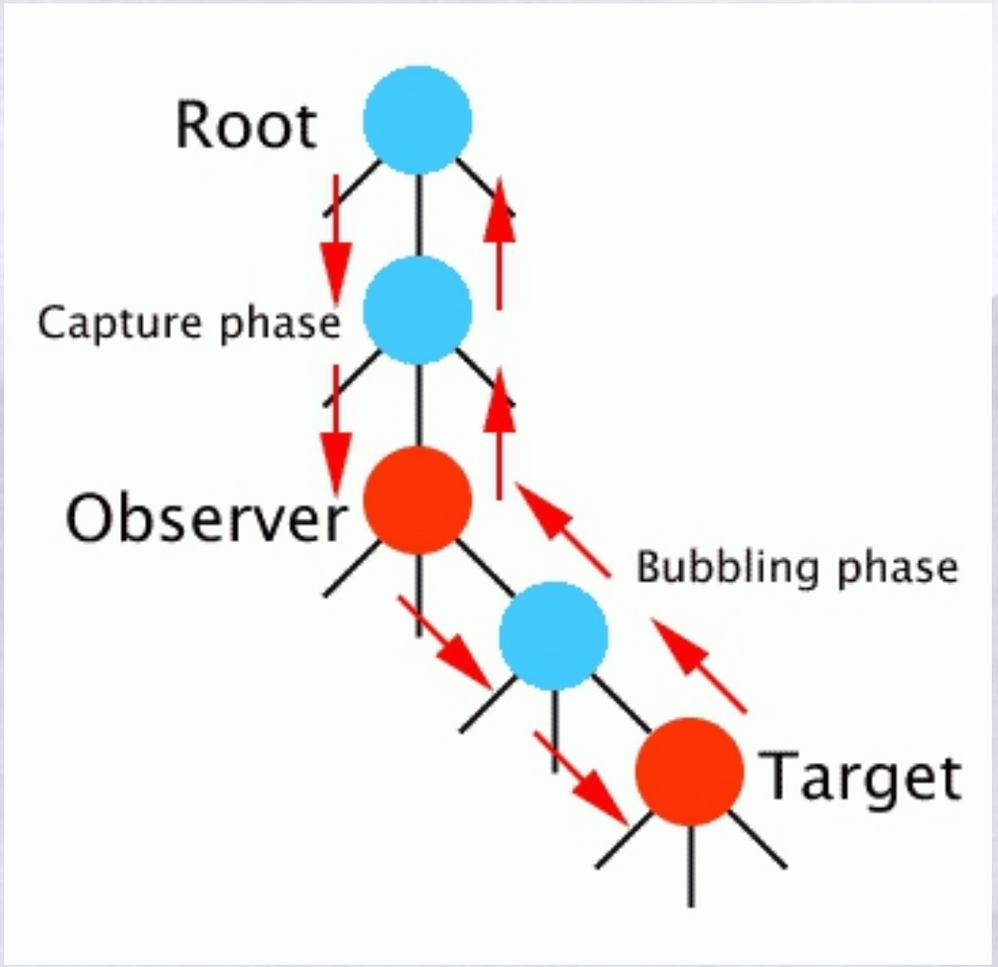
- The **capture** phase: The event object must propagate through the target's ancestors from the Window to the target's parent. This phase is also known as the capturing phase. Event listeners registered for this phase must handle the event before it reaches its target.
- The **target** phase: The event object must arrive at the event object's event target. This phase is also known as the at-target phase. Event listeners registered for this phase must handle the event once it has reached its target. If the event type indicates that the event must not bubble, the event object must halt after completion of this phase.
- The **bubble** phase: The event object propagates through the target's ancestors in reverse order, starting with the target's parent and ending with the Window. This phase is also known as the bubbling phase. Event listeners registered for this phase must handle the event after it has reached its target.

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
- 职责链模式



DOM Level 2

Name	Description	Argument type	Argument name
addEventListener	Allows the registration of event listeners on the event target.	DOMString	type
		EventListener	listener
		boolean	useCapture
removeEventListener	Allows the removal of event listeners from the event target.	DOMString	type
		EventListener	listener
		boolean	useCapture
dispatchEvent	Allows sending the event to the subscribed event	Event	evt

Browser Support

- The numbers in the table specifies the first browser version that fully supports these methods.

Method	Chrome	IE	Firefox	Safari	Opera
addEventListener()	1.0	9.0	1.0	1.0	7.0
removeEventListener()	1.0	9.0	1.0	1.0	7.0

Some useful things to know

- To prevent an event from bubbling, developers must call the "stopPropagation()" method of the event object.
  - cancelBubble=true
- To prevent the default action of the event to be called, developers must call the "preventDefault" method of the event object.
  - canceling default action (e.g. navigating to a new page when clicking on a hyperlink)
  -

A rewrite of the example used in the traditional model

```
<script>
var heyJoe = function () {
   window.alert( "Hey Joe!" );
}

// Add an event handler
document.addEventListener( "click", heyJoe, true );  // capture
phase

      // Add another event handler
      window.addEventListener( "load", heyJoe, false );  // bubbling
phase

      // Remove the event handler just added
       window.removeEventListener( "load", heyJoe, false );
      </script>
```

Microsoft-specific model

- Microsoft does not follow the W3C model up until Internet Explorer 8, as its own model was created prior to the ratification of the W3C standard. Internet Explorer 9 follows DOM level 3 events, and Internet Explorer 11 deletes its support for Microsoft-specific model.

Name	Description	Argument type	Argument name
attachEvent	Similar to W3C's addEventListener method.	String	sEvent
		Pointer	fpNotify
detachEvent	Similar to W3C's removeEventListener method.	String	sEvent
		Pointer	fpNotify
fireEvent	Similar to W3C's dispatchEvent method.	String	sEvent
		Event	oEventObject

Some useful things to know……

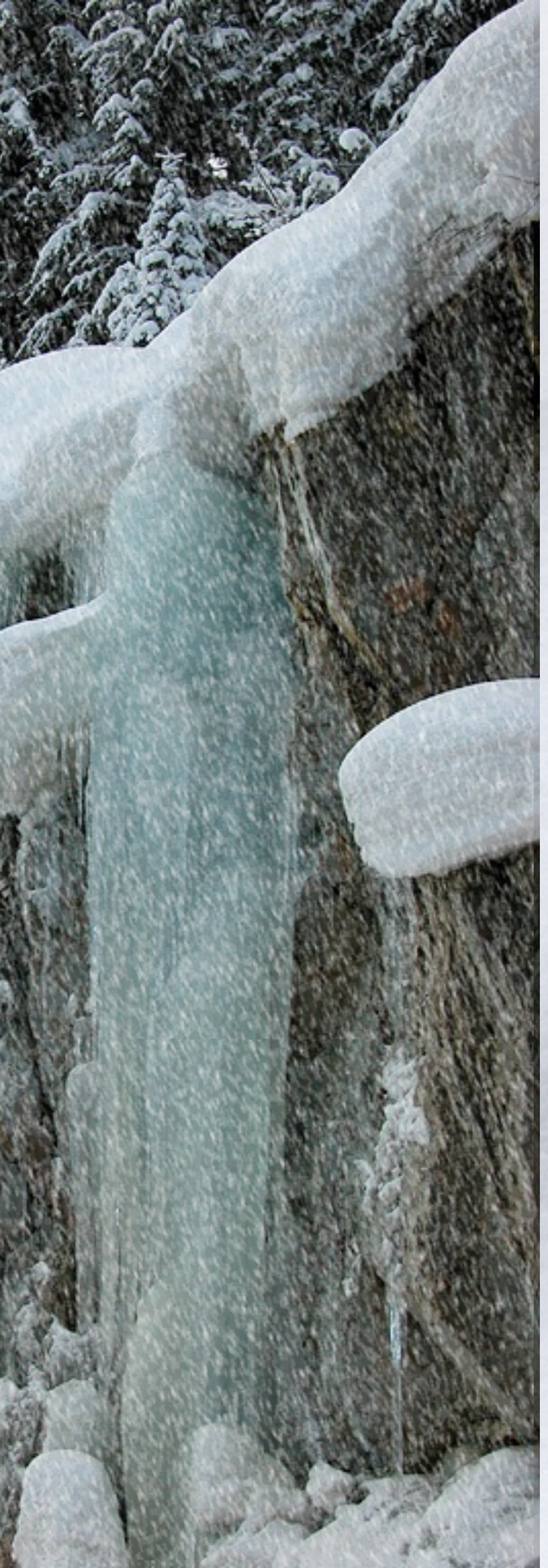
- To prevent an event bubbling, developers must set the event's cancelBubble property.
- To prevent the default action of the event to be called, developers must set the event's "returnValue" property.

Cross-browser solution

```
var x = document.getElementById("myBtn");
if (x.addEventListener) {
// For all major browsers, except IE 8 and earlier
    x.addEventListener("click", myFunction);
} else if (x.attachEvent) {// For IE 8 and earlier versions
    x.attachEvent("onclick", myFunction);
}
```

Event Handler Registration

- inline:
  - <a href="somewhere.html" onClick="myFunction()">
- traditional:
  - element.onclick = myFunction;
- DOM 2:
  - *element*.addEventListener(“click", myFunction);
- IE: (evil enough!)
  - element.attachEvent('onclick', myFunction);
- JQuery, Prototype...and so on:
  - jQuery.on()
  - Event.observe('target', 'click', myFunction);



Outline

- Javascript Events
- Event Types
- Event handling models
- Scope and Closure

JavaScript Scope

- Scope is the set of variables you have access to.

- JavaScript Scope
  - In JavaScript, objects and functions are also variables.
  - In JavaScript, scope is the set of variables, objects, and functions you have access to.
    - JavaScript has function scope: The scope changes inside functions.
- ES6之前，只有函数作用域和全局作用域
- ES6增加了通过let和const声明变量的块级作用域

Local JavaScript Variables

- Variables declared within a JavaScript function, become LOCAL to the function.
- Local variables have local scope: They can only be accessed within the function.
- Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.
- Local variables are created when a function starts, and deleted when the function is completed.

Example

```
// code here can not use carName

function myFunction() {
    var carName = "Volvo";

    // code here can use carName

}
```

Global JavaScript Variables

- A variable declared outside a function, becomes GLOBAL.
- A global variable has global scope: All scripts and functions on a web page can access it.

Example

```
var carName = " Volvo";

// code here can use carName

function myFunction() {
    // code here can use carName
}
```

Automatically Global

- If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable.
- This code example will declare carName as a global variable, even if it is executed inside a function.

Example

```
// code here can use carName

function myFunction() {
    carName = "Volvo";

    // code here can use carName

}
```

The Lifetime of JavaScript Variables

- The lifetime of a JavaScript variable starts when it is declared.
- Local variables have short lives. They are created when the function is invoked, and deleted when the function is finished.

Function Arguments

- Function arguments (parameters) work as local variables inside functions.

Global Variables in HTML

- Global variables live as long as your application (your window / your web page) lives.
- Global variables are deleted when you close the page.
- In HTML, the global scope is the window object: All global variables belong to the window object.

A Counter Dilemma

- Suppose you want to use a variable for counting something, and you want this counter to be available to all functions.
- You could use a global variable, and a function to increase the counter:

```
Example
var counter = 0;

function add() {
    counter += 1;
}

add();
add();
add();
```

But

- If I declare the counter inside the function, nobody will be able to change it without calling add():

Example

```
function add() {
    var counter = 0;
    counter += 1;
}

add();
add();
add();

// the counter should now be 3, but it does not work !
```

JavaScript Nested Functions

- All functions have access to the global scope.
- In fact, in JavaScript, all functions have access to the scope "above" them.
- JavaScript supports nested functions. Nested functions have access to the scope "above" them.
- In this example, the inner function plus() has access to the counter variable in the parent function:

Example

```
function add() {
    var counter = 0;
    function plus() {counter += 1;}
    plus();
    return counter;
}
```

JavaScript Closures

- The variable add is assigned the return value of a self-invoking function.
- The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression.
- This way add becomes a function. The "wonderful" part is that it can access the counter in the parent scope.
- This is called a JavaScript closure. It makes it possible for a function to have "private" variables.
- The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

- A closure is a function having access to the parent scope, even after the parent function has closed.

Example

```
var add = (function () {
    var counter = 0;
    return function () {return
counter += 1;}
 })();

add();
add();
add();
```

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
  - 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

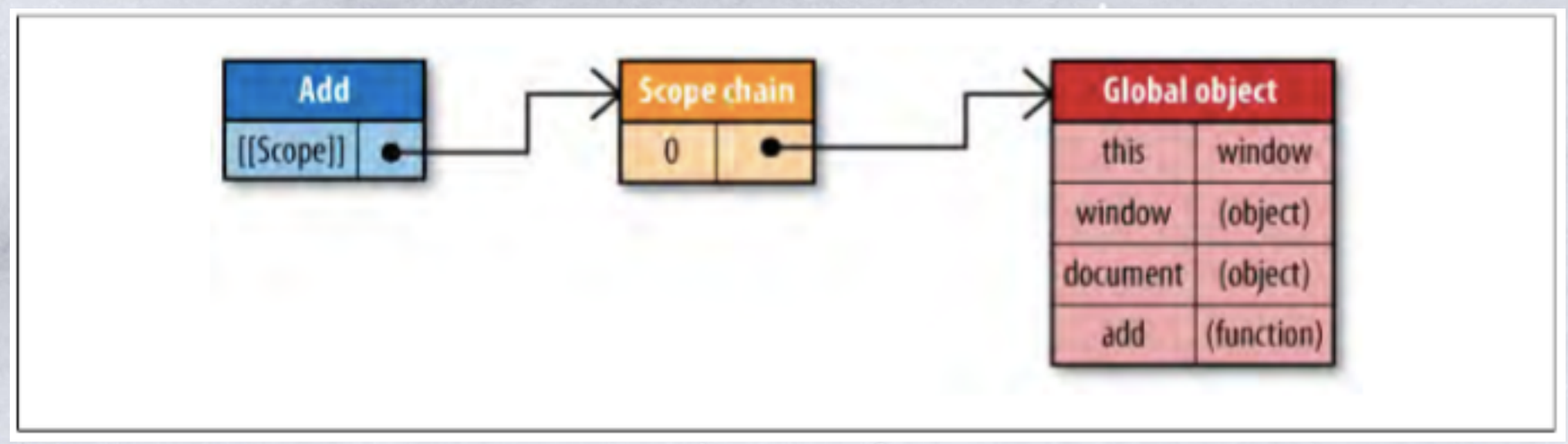
闭包应用场景

- 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();
```

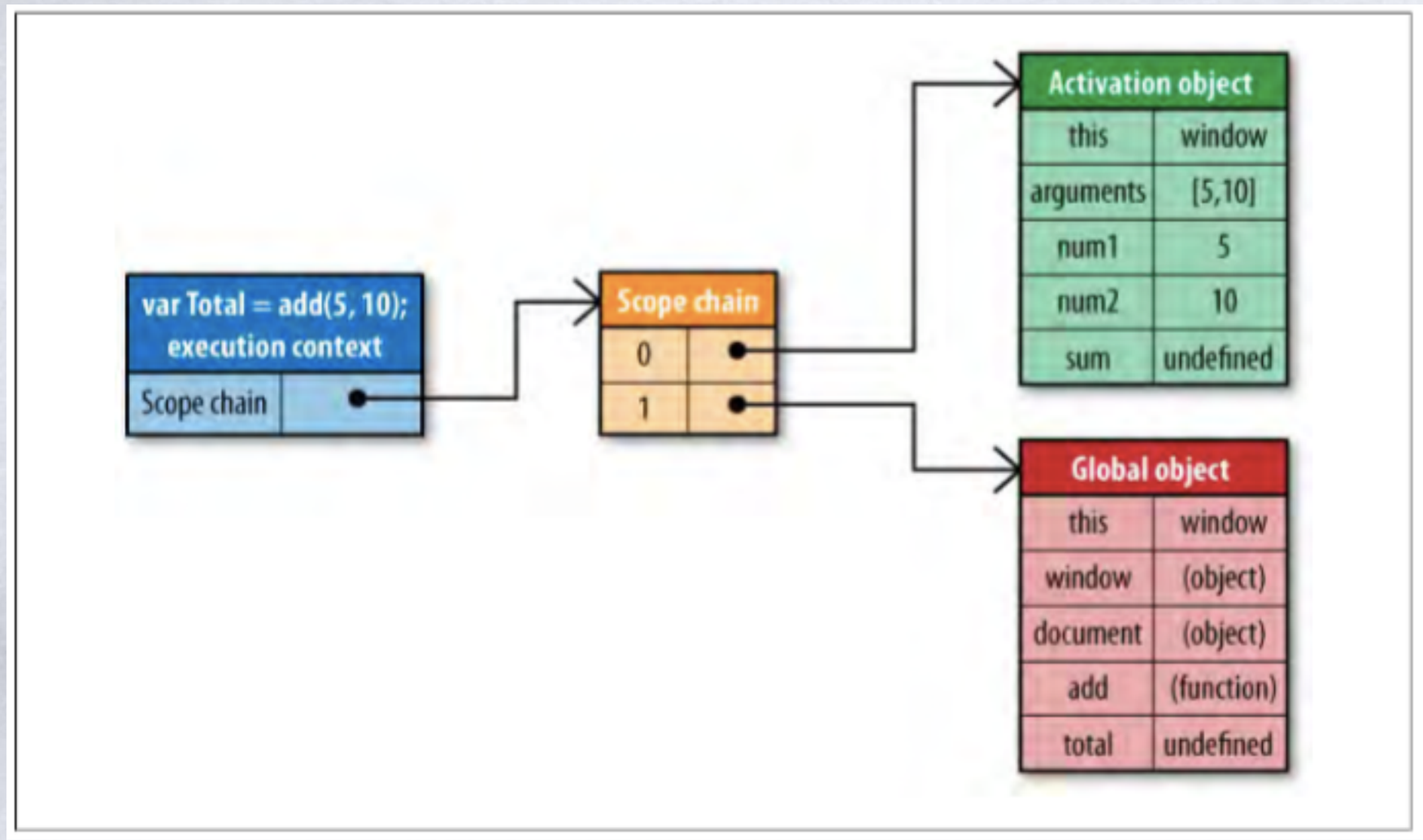
Scope chain

```
function add(num1, num2){
    var sum = num1 + num2;
    return sum;
}
```



Executing the add function

- execution context and its scope chain



For example

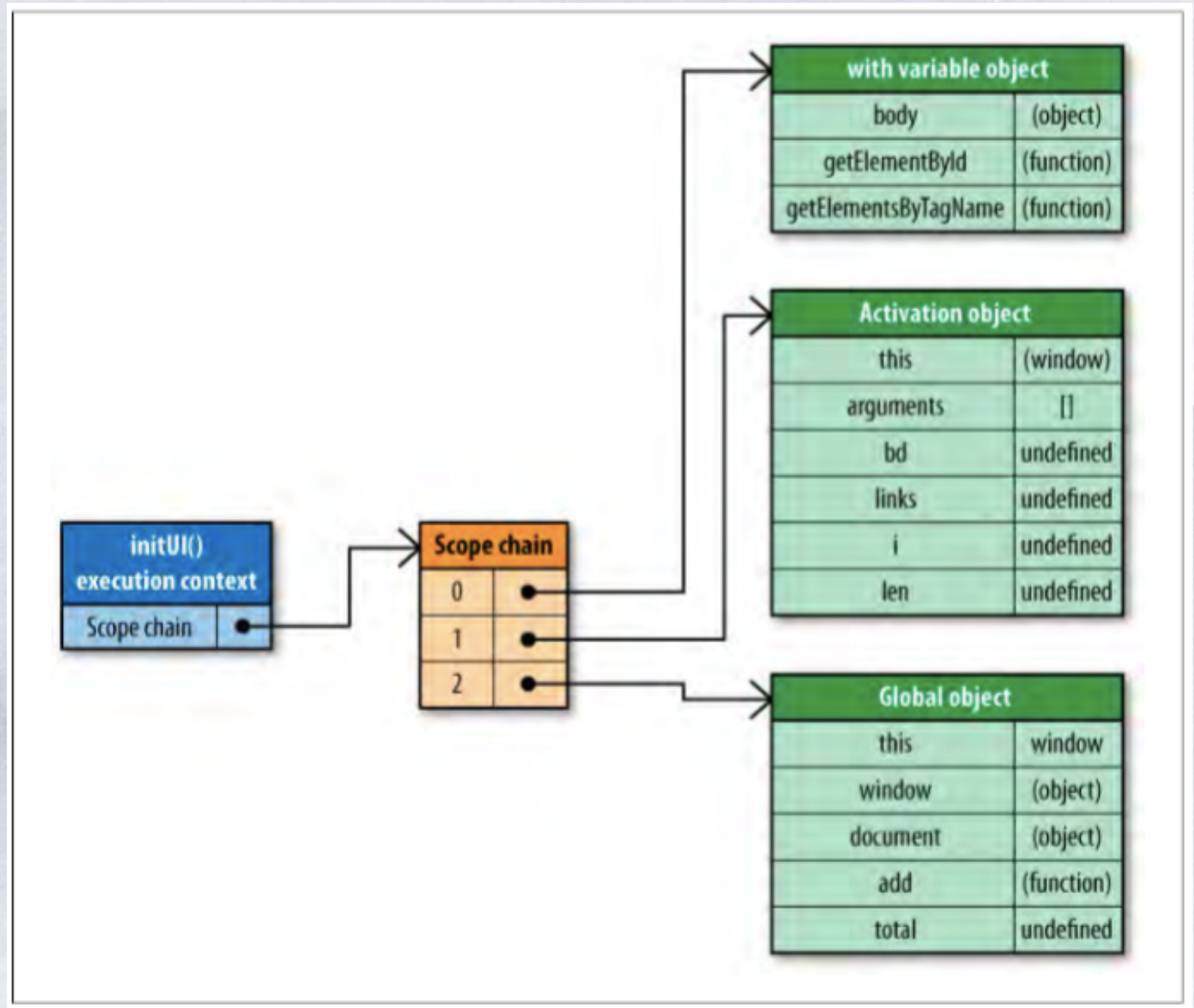
```
function initUI(){
  var bd = document.body,
      links = document.getElementsByTagName("a"),
      i= 0,
      len = links.length;

  while(i < len){
        update(links[i++]);
  }
  document.getElementById("go-btn").onclick = function(){
        start();
  };
  bd.className = "active";
}
```

```
function initUI(){
     var doc = document,
          bd = doc.body,
          links = doc.getElementsByTagName("a"),
          i= 0,len = links.length;
......}
```

Scope Chain Augmentation

```
function initUI(){
    with (document){ //avoid!
        var bd = body,
            links = getElementsByTagName("a"),
            i= 0,
            len = links.length;
        while(i < len){
            update(links[i++]);
        }
        getElementById("go-btn").onclick = function(){
            start();
        };
        bd.className = "active";
    }
}
```

Scope Chain Augmentation



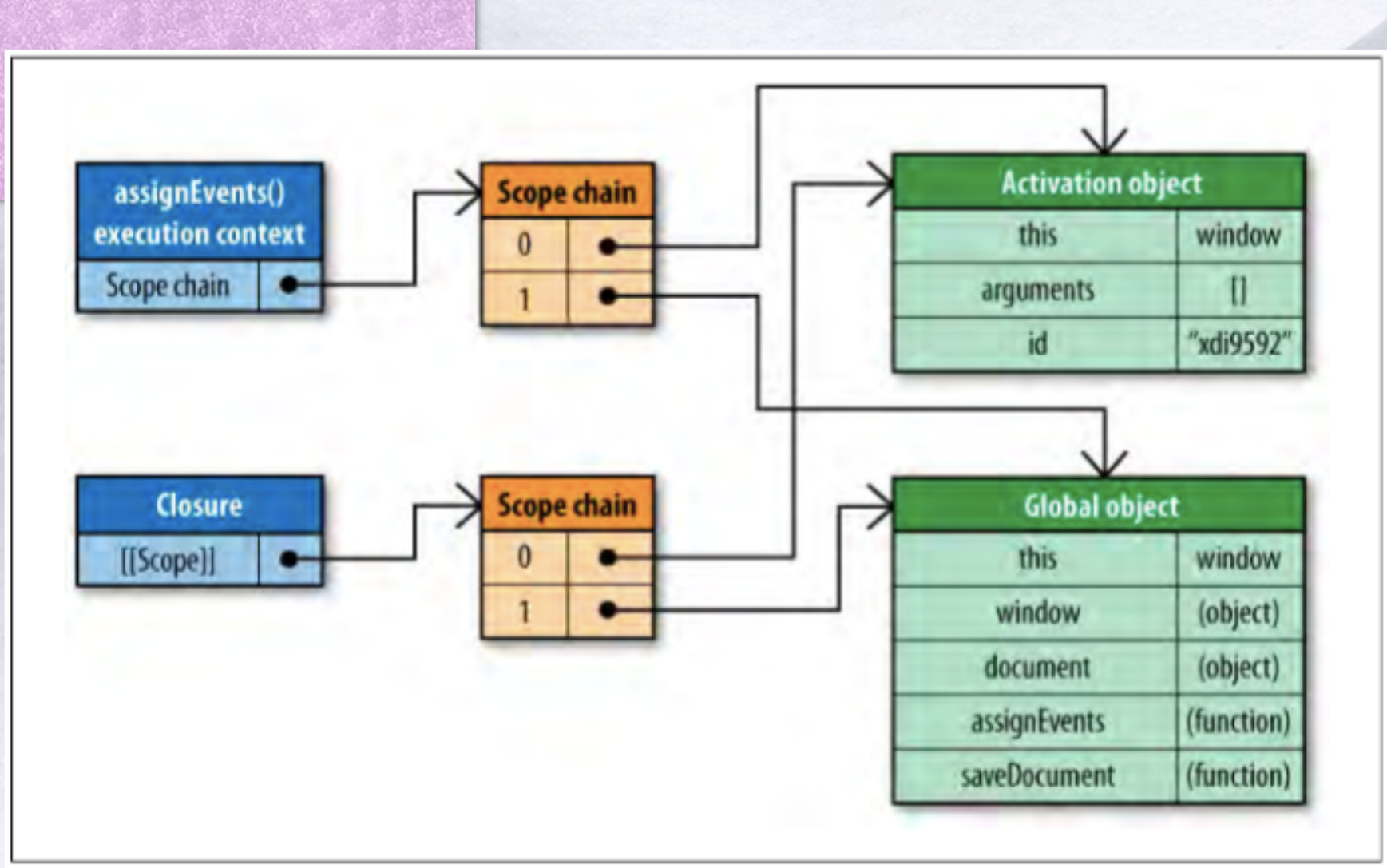
try-catch

```
try {
      methodThatMightCauseAnError(); }
catch (ex){
      alert(ex.message); //scope chain is augmented here
}
```

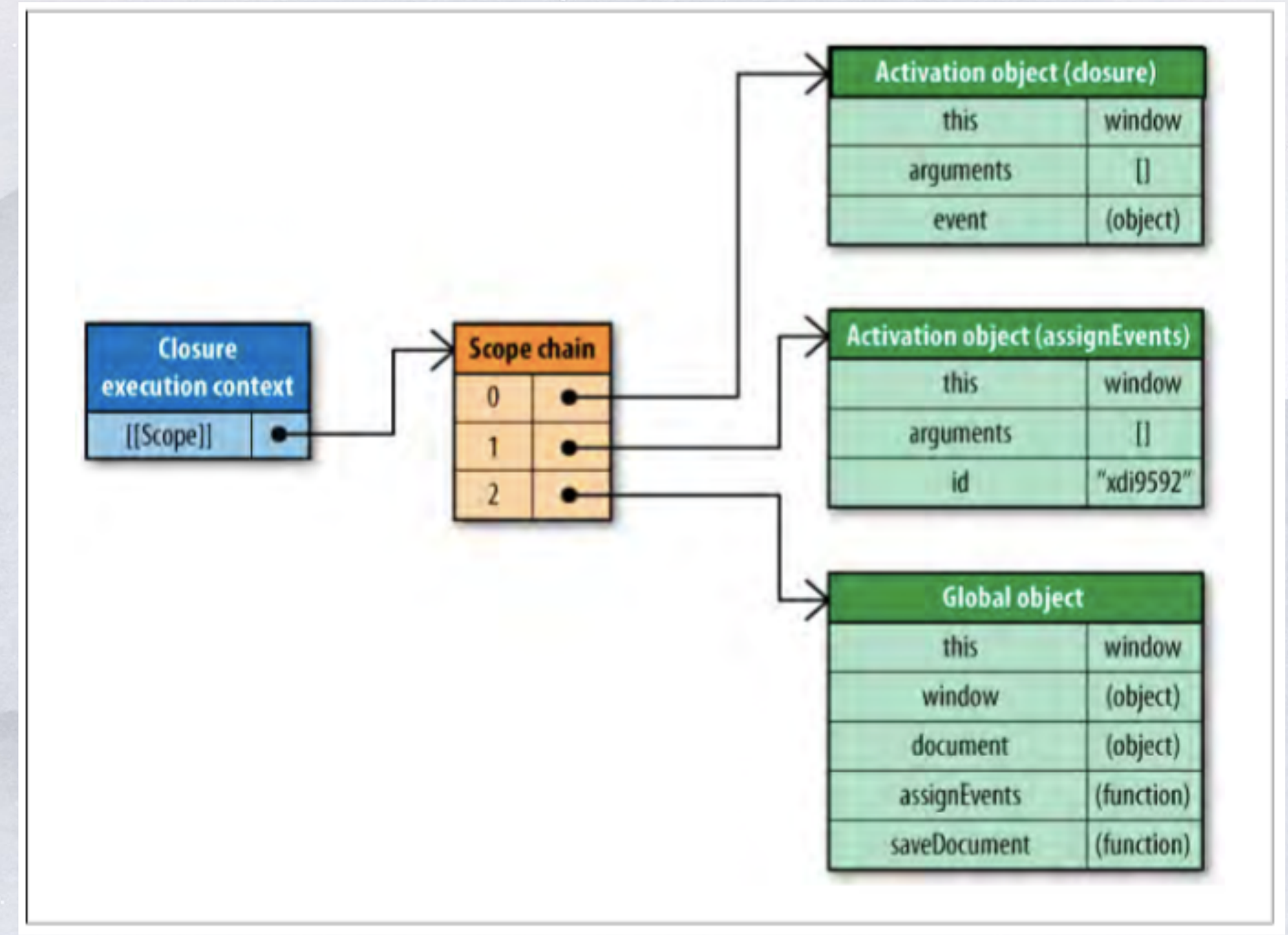
```
try {
      methodThatMightCauseAnError(); }
catch (ex){
      handleError(ex); //delegate to handler method
}
```

Closures, Scope, and Memory

```
function assignEvents(){
  var id = "xdi9592";
  document.getElementById(“save-btn”)
      .onclick = function(event){
  saveDocument(id); };
}
```



Executing the closure



Example 1

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

The Window

Example 2

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

My Object

推荐阅读

- JavaScript: The Good Parts, by Douglas Crockford.
- Programming Javascript Applications.
- JAVASCRIPT设计模式，Ross Harmes & Dustin Diaz. 人民邮电出版社.
- wikipedia: scope closure
- 阮一峰：学习Javascript闭包(Closure)
- 深入理解JavaScript闭包(Closure)
- http://w3techs.com
- http://kangax.github.io/es5-compat-table/es6/
- https://babeljs.io/docs/learn-es2015/
- https://github.com/lukehoban/es6features

Thanks!!!