

DOM

Michel Gagnon
École Polytechnique de Montréal



Ce qui se passe lorsqu'un document HTML est chargé

1. Déclenchement du processus de construction du DOM (Document Objet Model)
2. Quand une balise `<script>` est rencontrée, la construction du DOM est interrompue pour charger et exécuter le script (sauf s'il y a présence de l'attribut **async** ou **defer**)
(ex. `<script src='script.js' async></script>`)
3. Les scripts asynchrones sont téléchargés et exécutés dès que possible, mais en attendant on continue la construction du DOM
4. Quand le DOM est construit, les scripts déclarés avec l'attribut **defer** sont exécutés
(ex. `<script src='script.js' defer></script>`)
5. Il se peut, à ce stage, qu'il reste encore des éléments à télécharger, comme des images. Quand tout est téléchargé, le navigateur déclenche un événement **load**

DOM

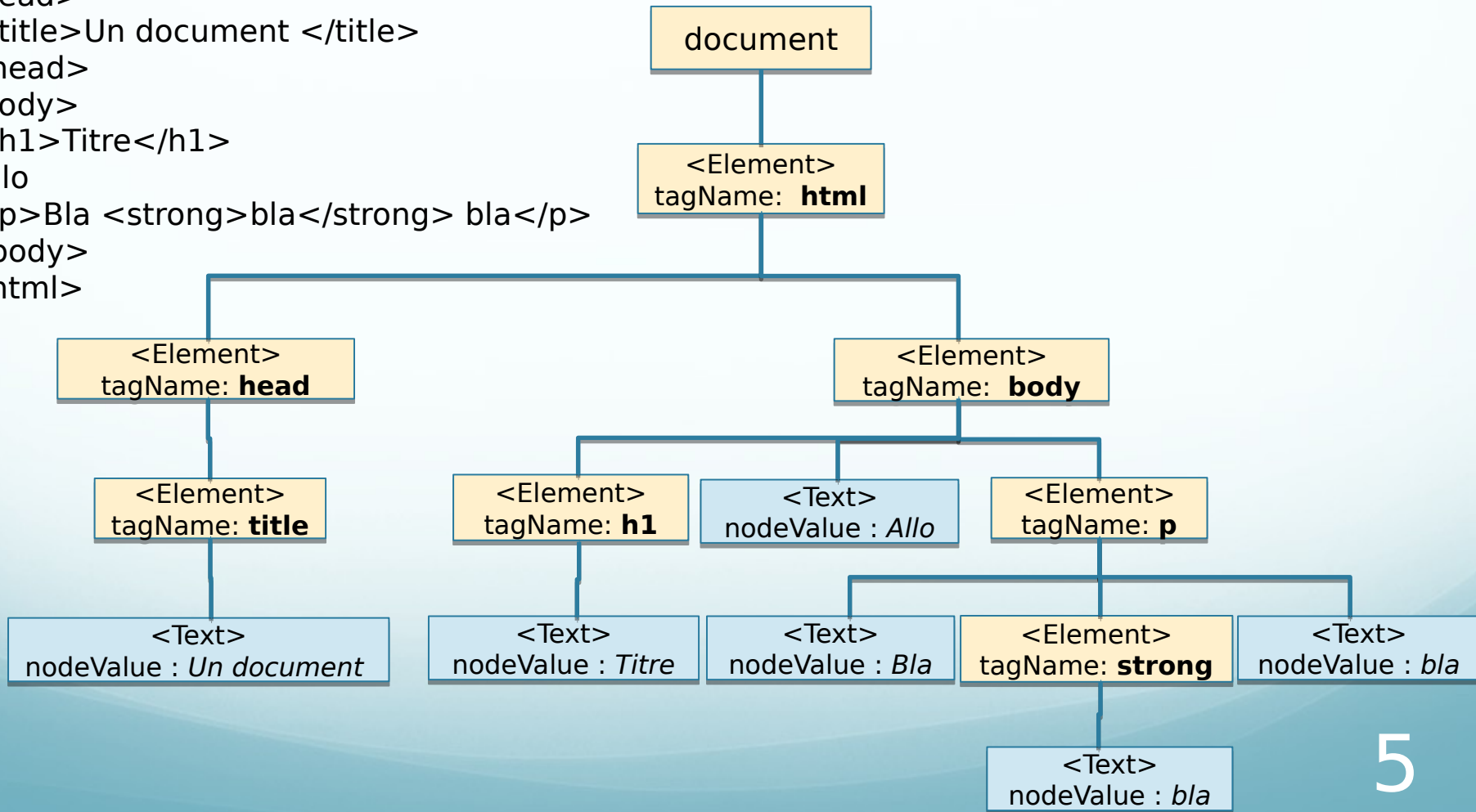
- Fournit une représentation structurée (sous forme d'objets) d'un document HTML
- Le DOM contient les items suivants:
 - Nœuds (classe Node): tous les items qui forment l'arborescence du document HTML (chaque nœud, sauf la racine, a donc un parent)
 - Événements (classe Event)

DOM

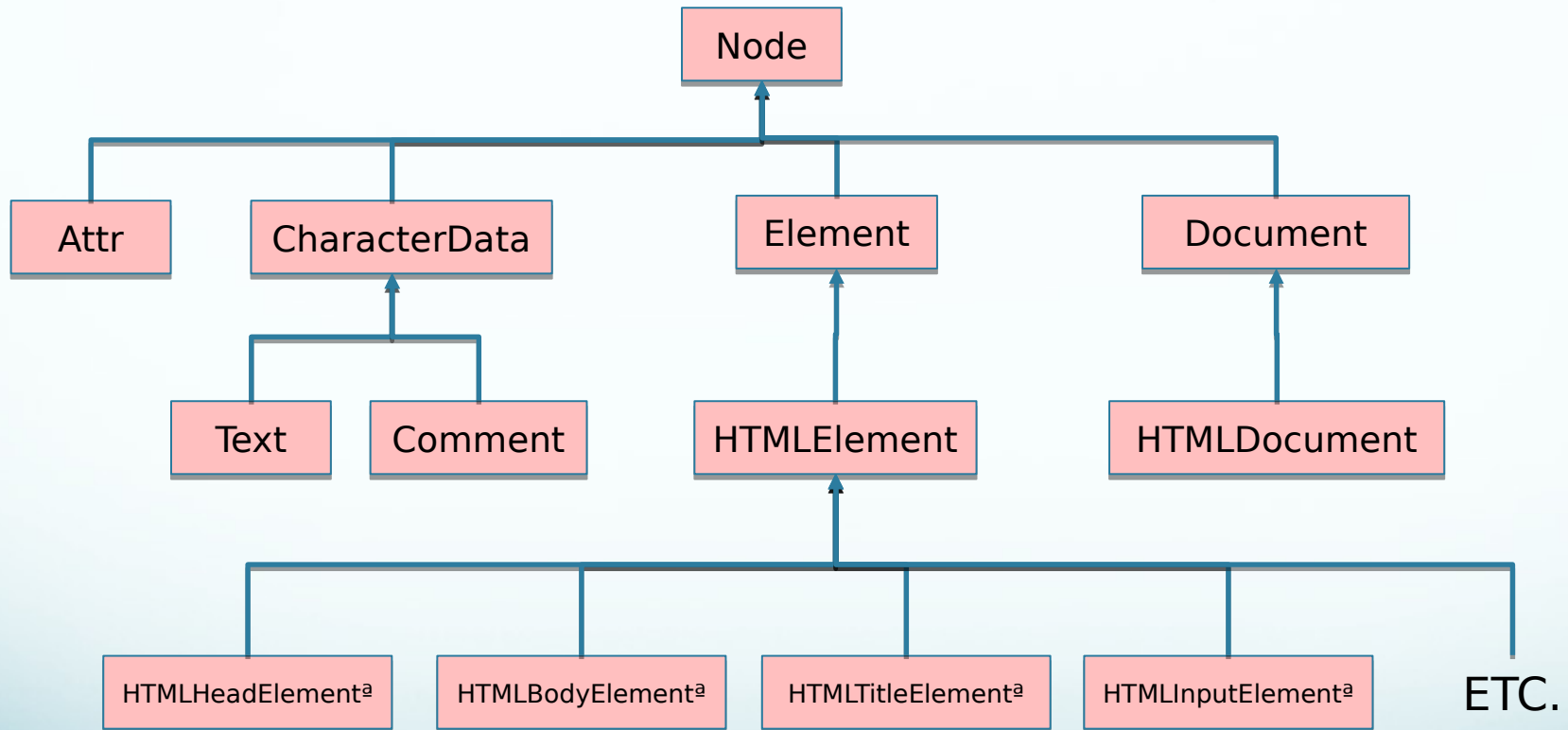
- Les nœuds qui composent le DOM implémentent une des interfaces suivantes:
 - **Element**: il s'agit d'un élément du document, correspondant à une balise HTML
 - **Text**: il s'agit du texte délimité par une balise
- Les nœuds forment un arbre dont la racine est l'objet **document**

Exemple d'arbre DOM

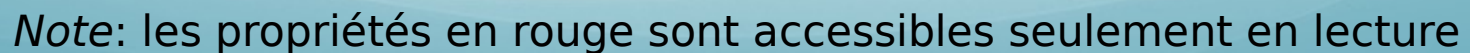
```
<html>
<head>
  <title>Un document </title>
</head>
<body>
  <h1>Titre</h1>
  Allo
  <p>Bla <strong>bla</strong> bla</p>
</body>
</html>
```



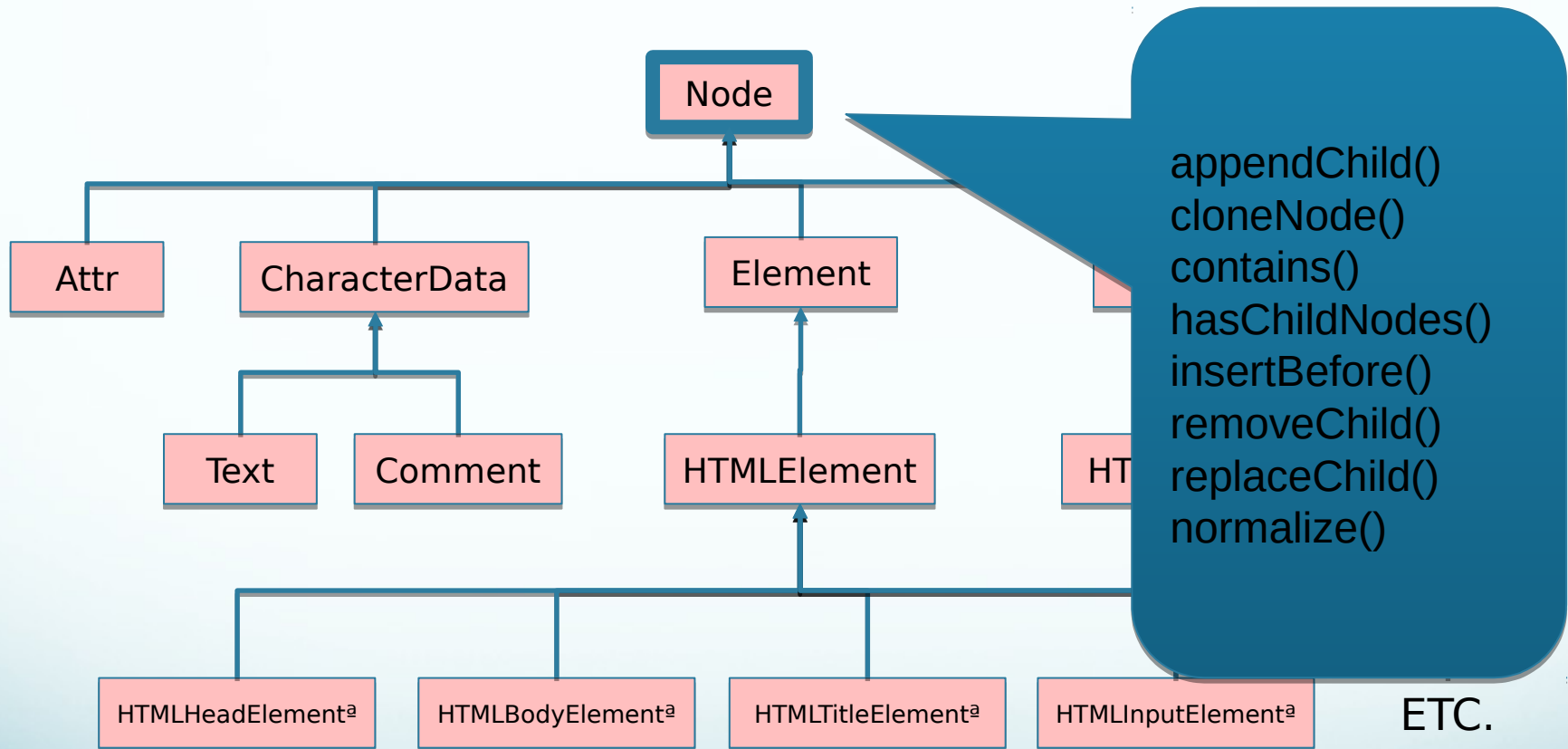
Hiérarchie des interfaces du DOM



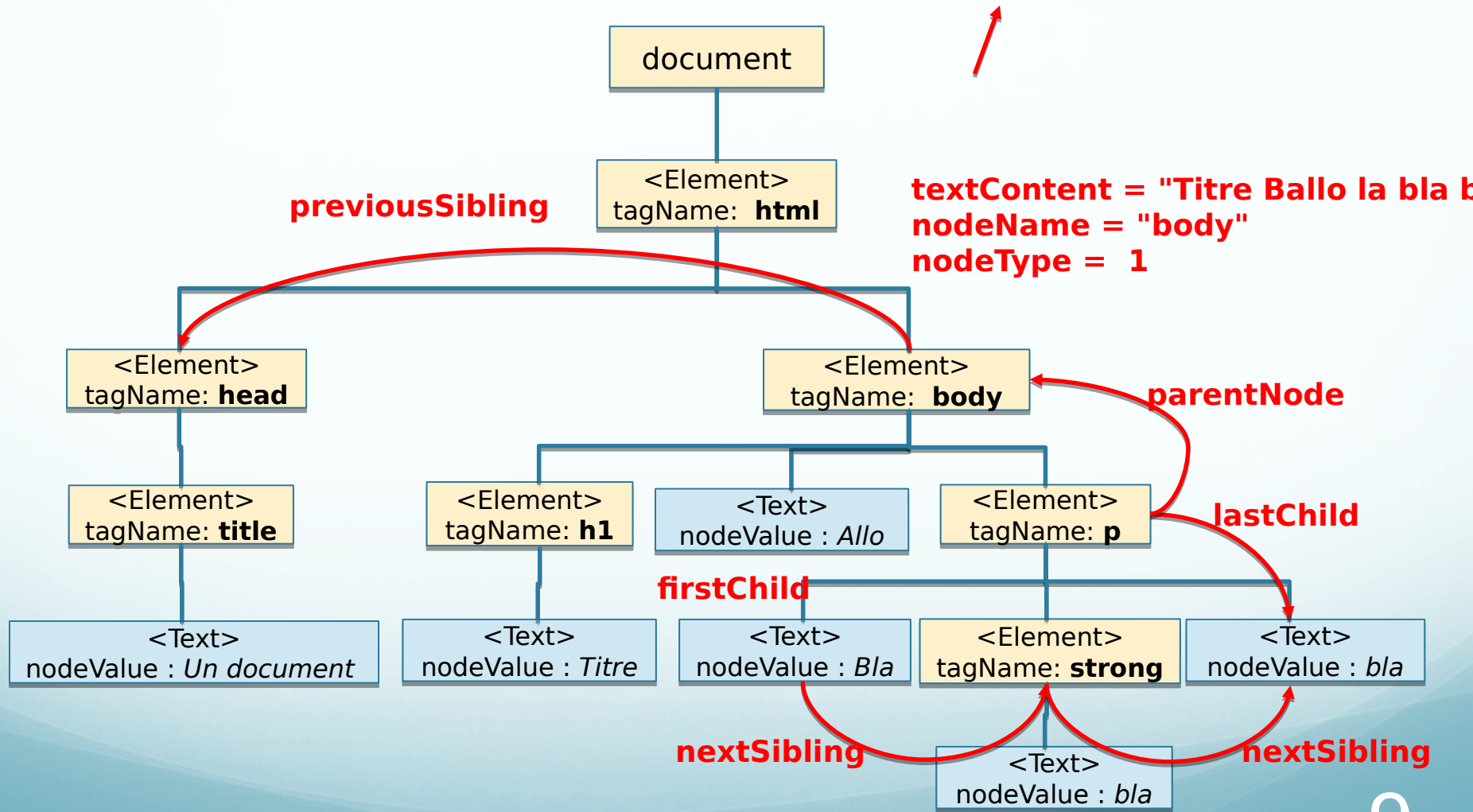
7



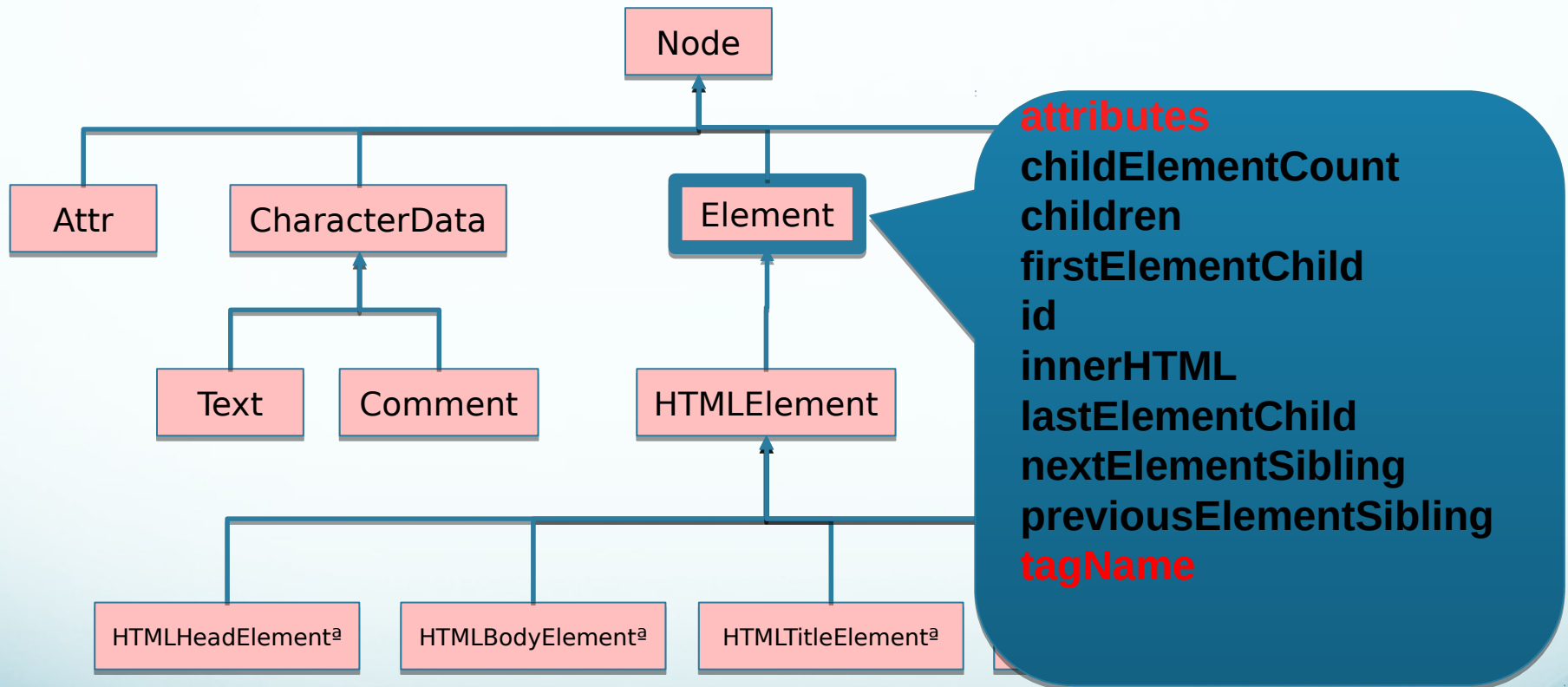
Hiérarchie des interfaces du DOM



DOM (API par nœuds)

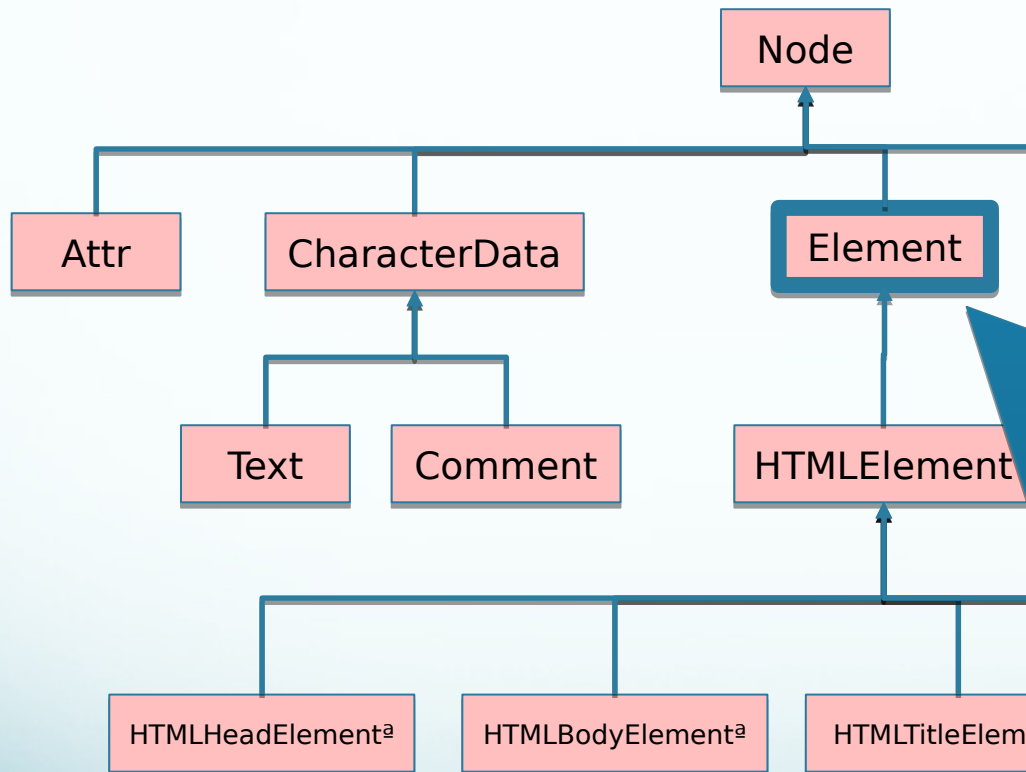


Hiérarchie des interfaces du DOM



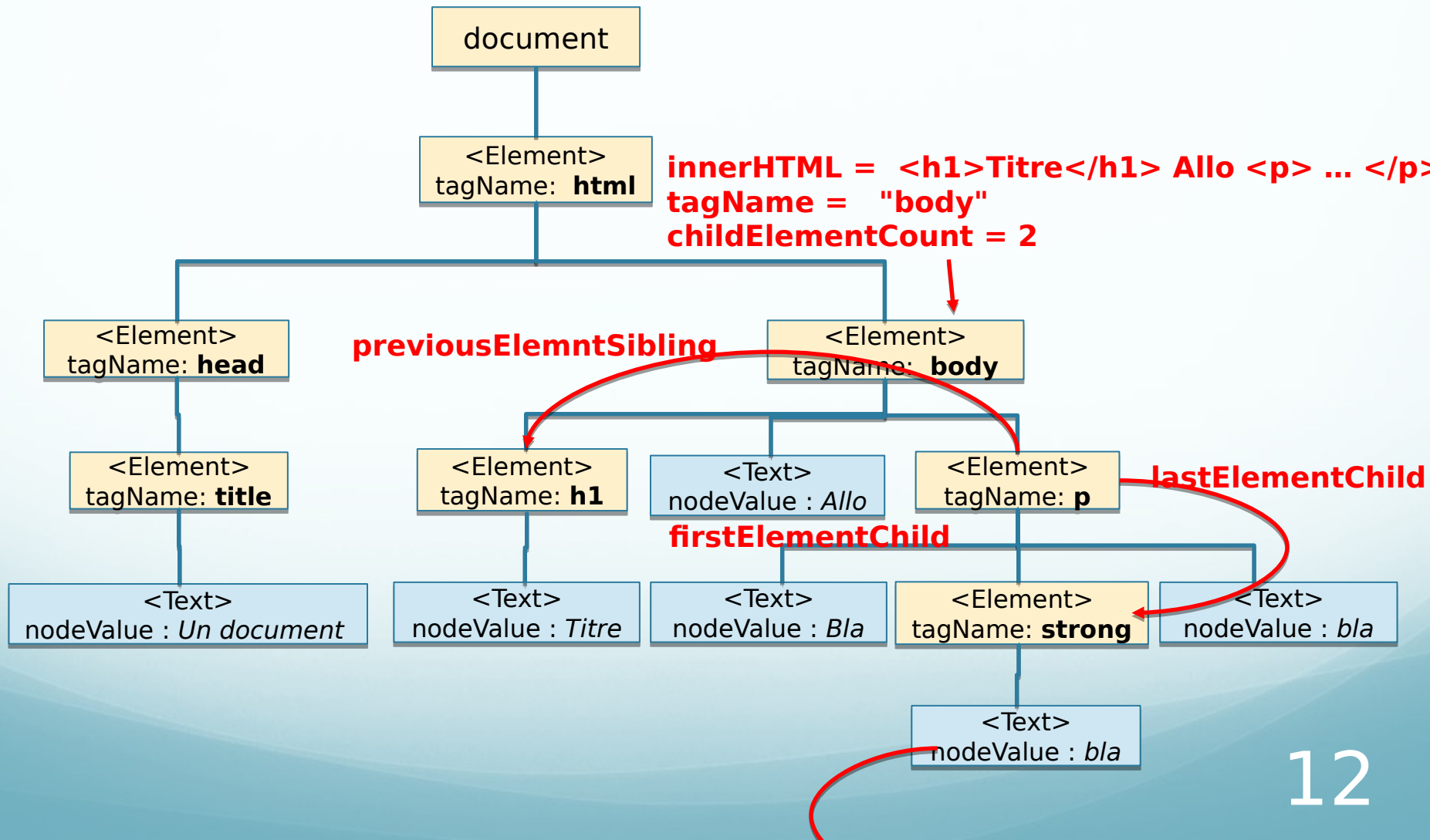
Note: les propriétés en rouge sont accessibles seulement en lecture

Hiérarchie des interfaces du DOM

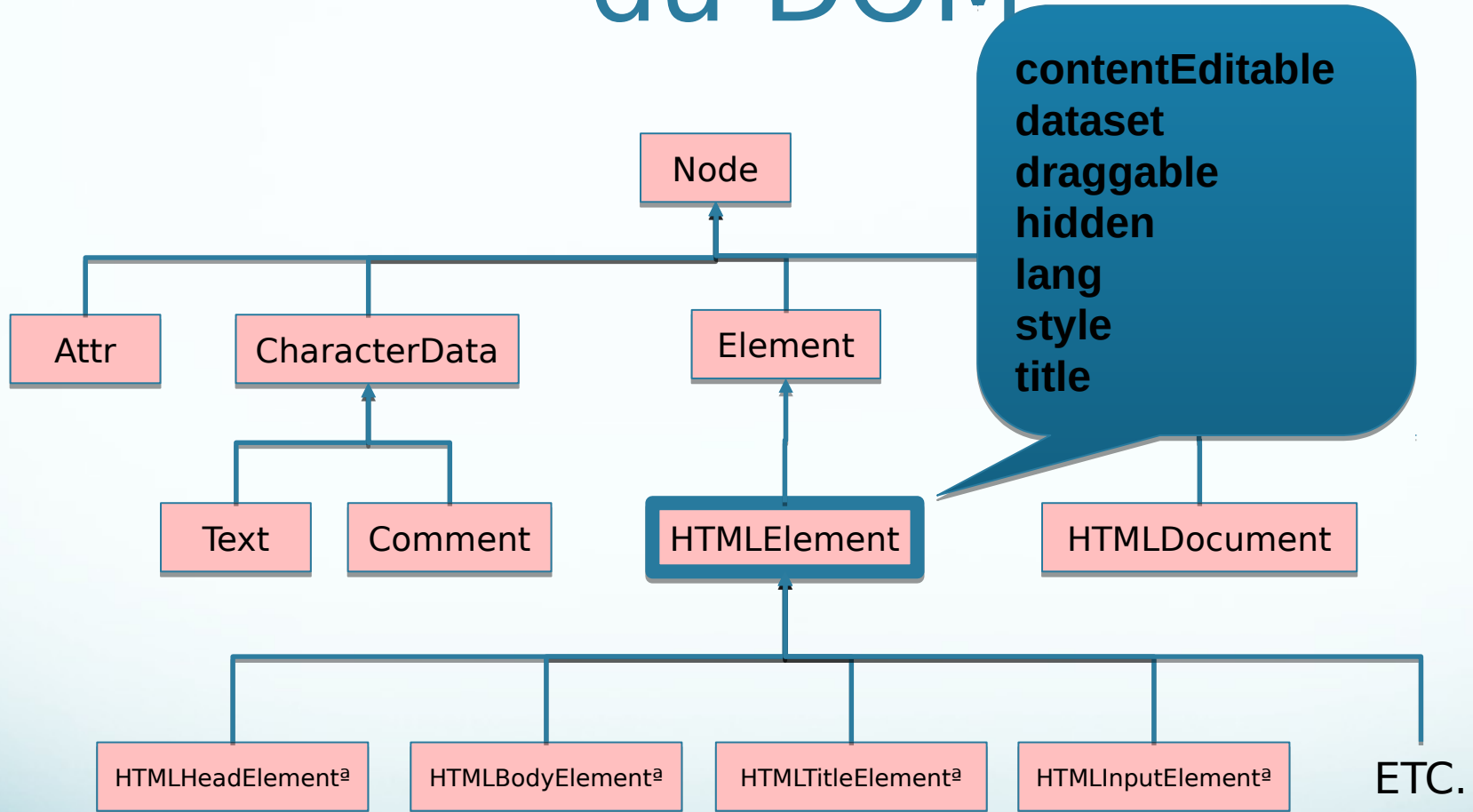


appendEventListener()
getAttribute()
getElementsByClassName()
getElementsByName()
hasAttribute()
querySelector()
querySelectorAll()
removeChild()
removeAttribute()
removeEventListener()
setAttribute()

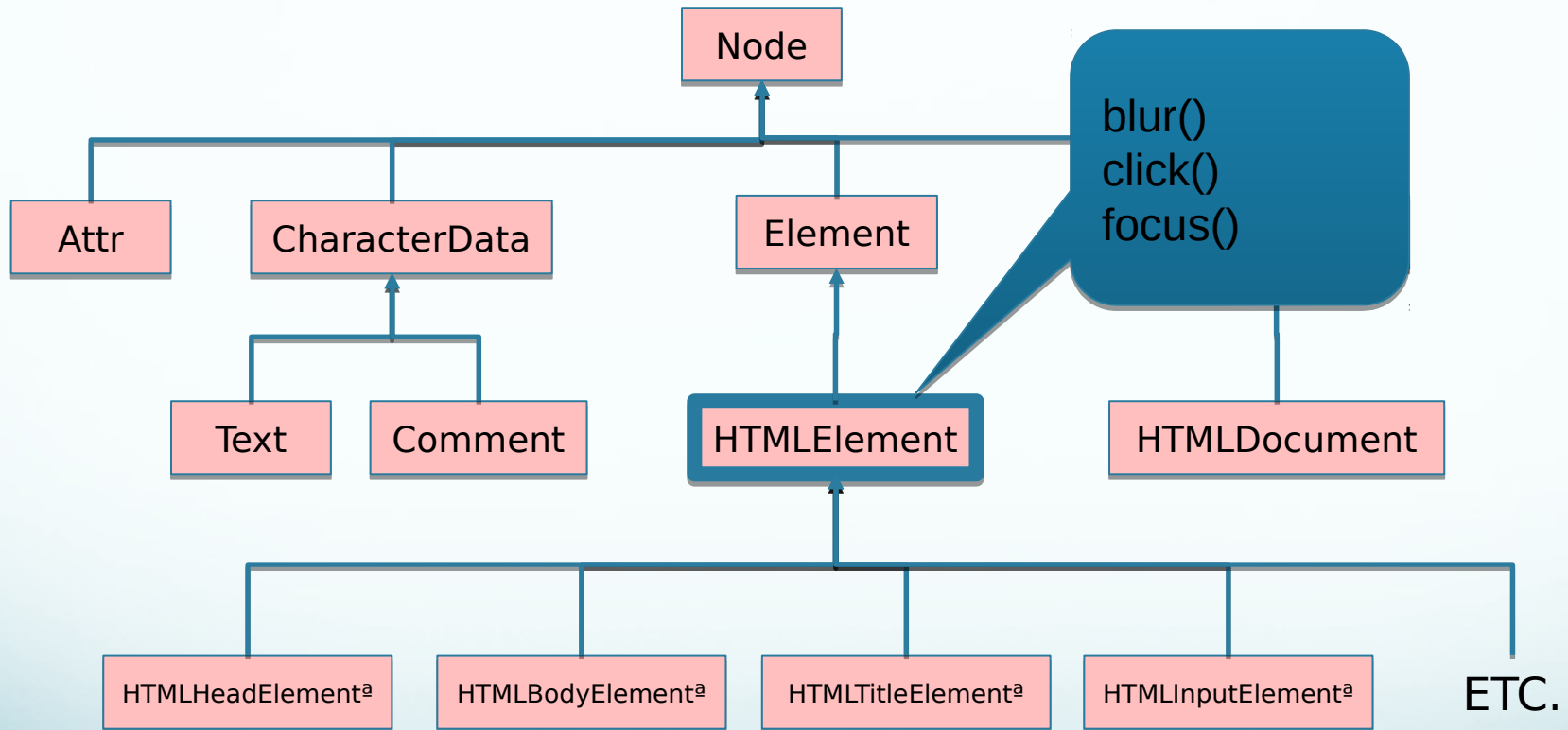
DOM (API par éléments)



Hiérarchie des interfaces du DOM



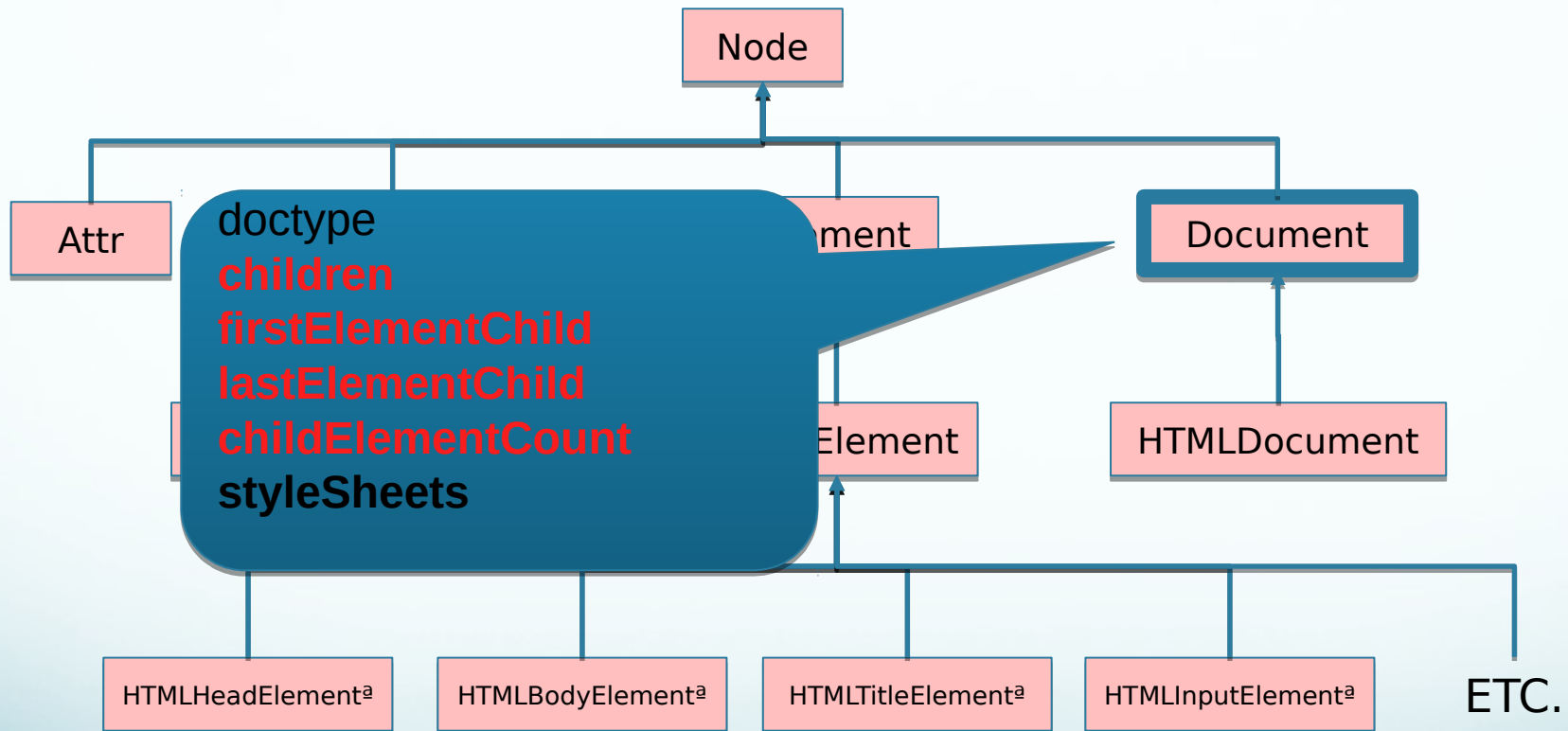
Hiérarchie des interfaces du DOM



Propriété **style**

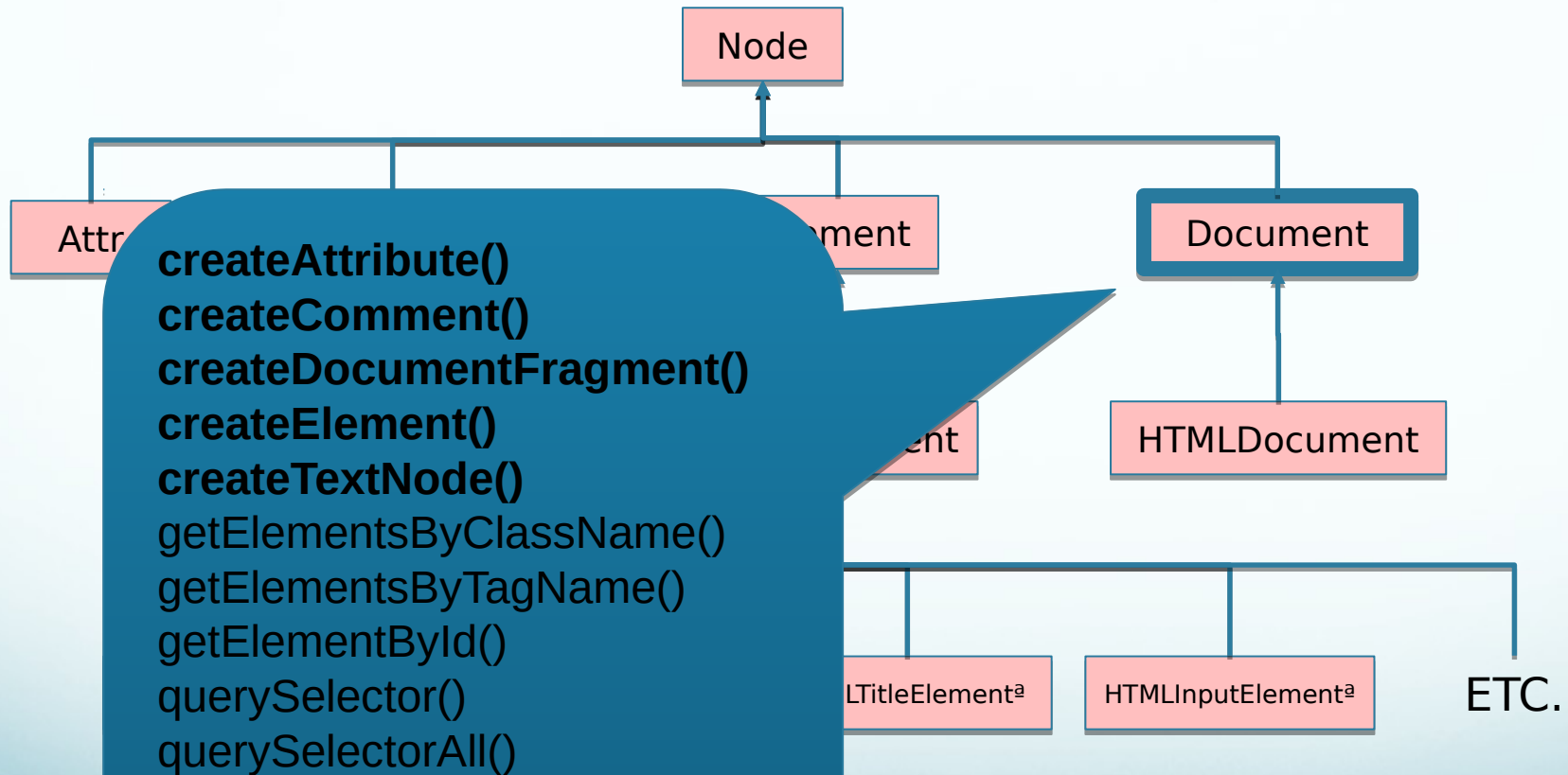
- Cette propriété est associée à des déclarations en-ligne CSS qui indiquent la présentation de l'élément
- Si on utilise cet attribut pour fixer des valeurs de propriété, elles seront au niveau « en-ligne » dans la cascade CSS
- Ce n'est pas cette propriété qu'il faut utiliser pour connaître le style d'un élément, puisqu'elle ne concerne que les déclarations en-ligne (utiliser plutôt la méthode **getComputedStyle()** de l'objet **window**)

Hiérarchie des interfaces du DOM

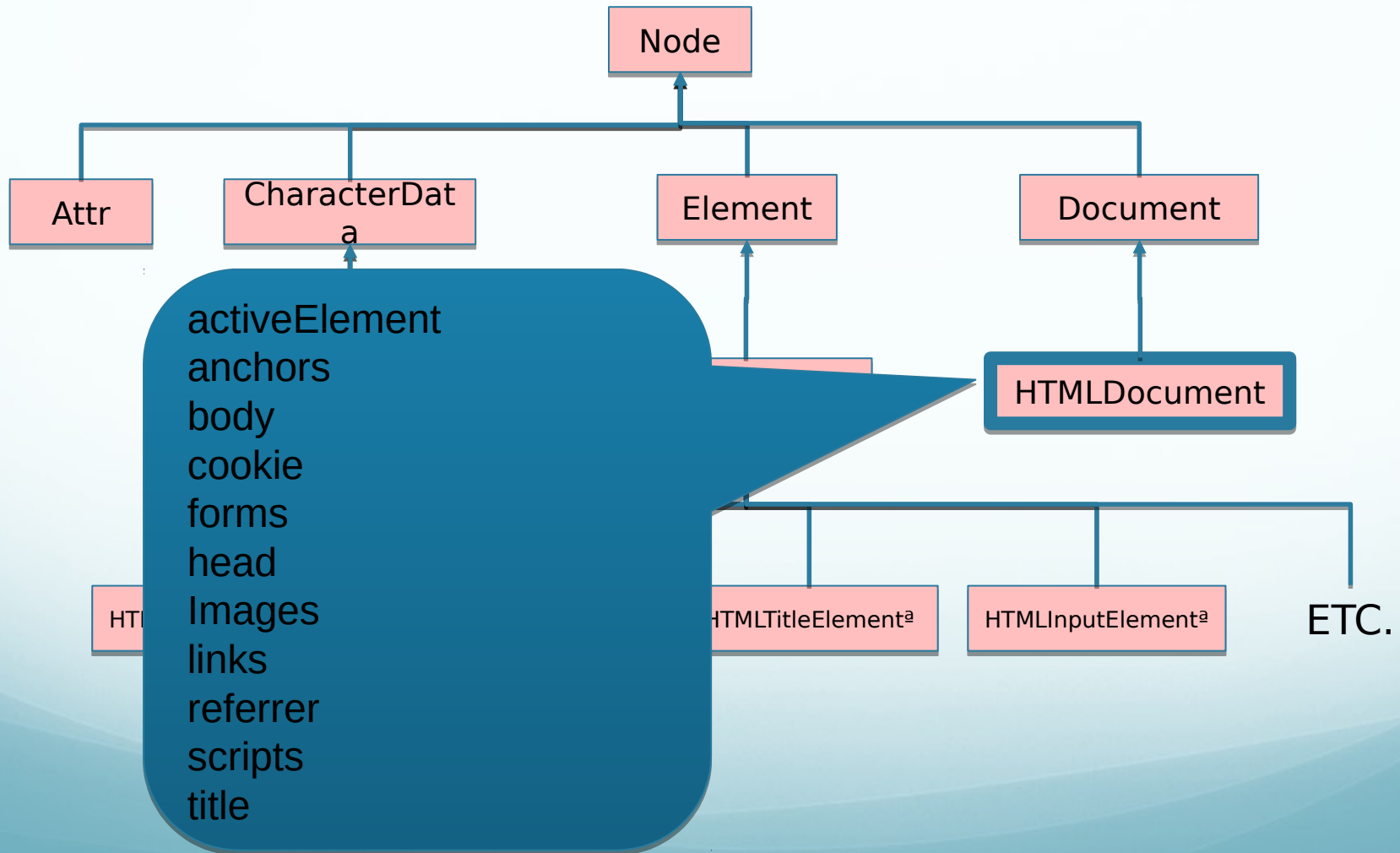


Note: les propriétés en rouge sont accessibles seulement en lecture

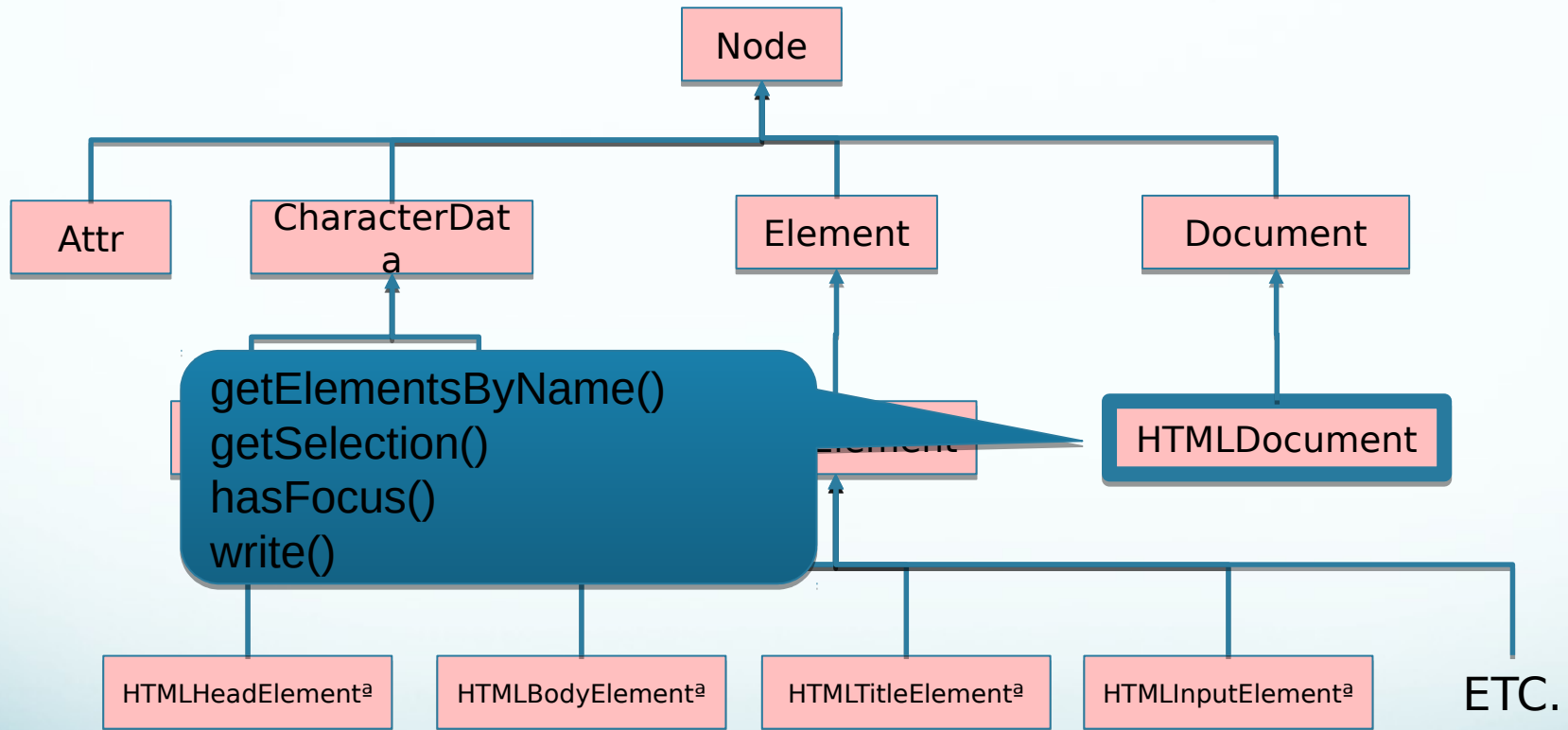
Hiérarchie des interfaces du DOM



Hiérarchie des interfaces du DOM



Hiérarchie des interfaces du DOM



Objet **window**

- Propriétés:
 - console
 - fullscreen
 - history
 - ...
- Méthodes:
 - alert()
 - close()
 - confirm()
 - getComputedStyle()
 - ...

Manipulation du DOM

Exemple

```
<html>
<script>
  // lancer cette fonction quand le document est chargé
  window.onload = function() {
    // crée quelques éléments dans
    // une page HTML pour l'instant vide
    heading = document.createElement("h1");
    heading_text = document.createTextNode("Grand titre");
    heading.appendChild(heading_text);
    document.body.appendChild(heading);
  }
</script>
</html>
```

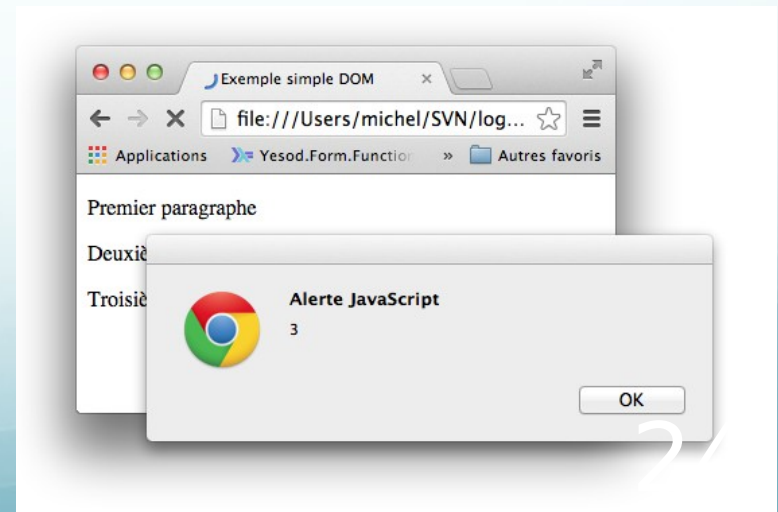


Interface NodeList

- Certaines méthodes, comme *getElementsByClass()*, retournent une collection d'éléments, qui sont contenus dans un objet de type *NodeList*
- On peut accéder à chaque élément contenu individuellement, avec l'opérateur [], comme si c'était un tableau
- Attention: très souvent ce conteneur est « vivant », c'est-à-dire que des changements dans le DOM seront reflétés dans la collection
- Le conteneur retourné par *querySelectorAll()* n'est pas « vivant »

Exemple de NodeList vivant

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Exemple simple DOM</title>
    <script>
      window.onload = function() {
        paragraphes = document.getElementsByTagName('p');
        alert(paragraphes.length);
        nouveauParagraphe = document.createElement('p');
        nouveauParagraphe.textContent = "Dernier paragraphe";
        document.body.appendChild(nouveauParagraphe);
        alert(paragraphes.length);
      }
    </script>
  </head>
  <body>
    <p> Premier paragraphe</p>
    <p> Deuxième paragraphe</p>
    <p> Troisième paragraphe</p>
  </body>
</html>
```



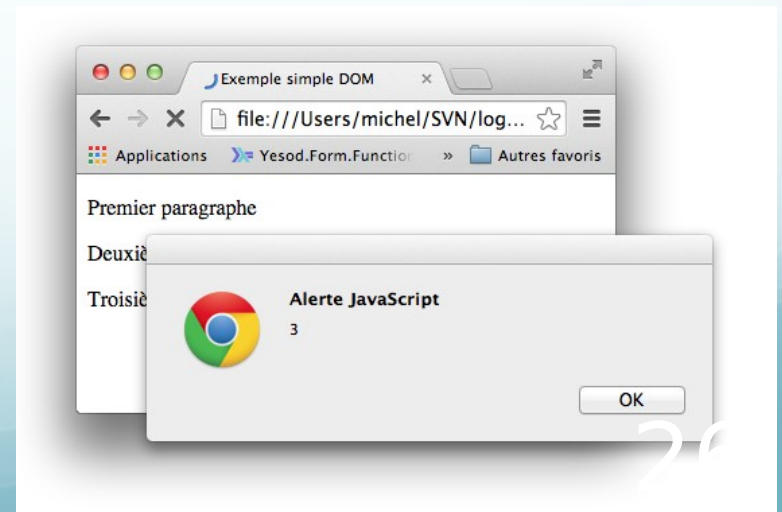
Exemple de NodeList vivant

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Exemple simple DOM</title>
    <script>
      window.onload = function() {
        paragraphes = document.getElementsByTagName('p');
        alert(paragraphes.length);
        nouveauParagraphe = document.createElement('p');
        nouveauParagraphe.textContent = "Dernier paragraphe";
        document.body.appendChild(nouveauParagraphe);
        alert(paragraphes.length);
      }
    </script>
  </head>
  <body>
    <p> Premier paragraphe</p>
    <p> Deuxième paragraphe</p>
    <p> Troisième paragraphe</p>
  </body>
</html>
```



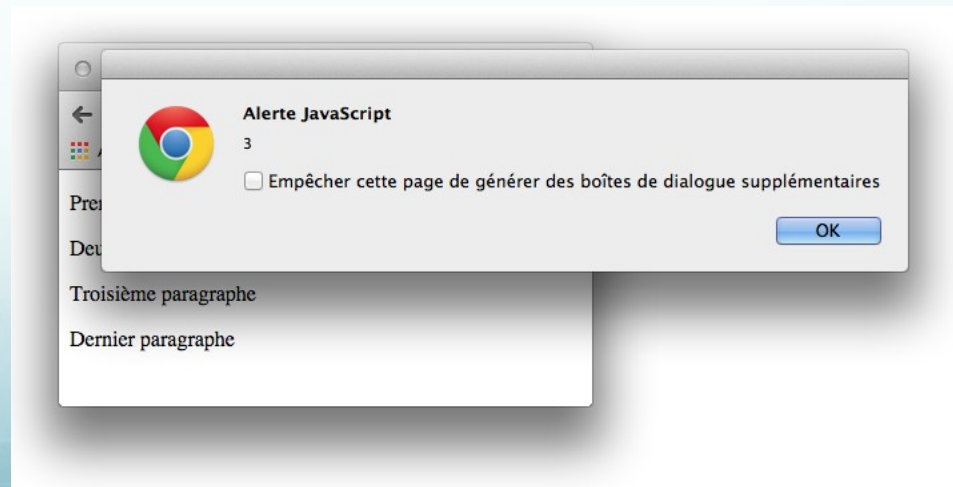
Exemple de NodeList vivant

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Exemple simple DOM</title>
    <script>
      window.onload = function() {
        paragraphes = document.querySelectorAll('p');
        alert(paragraphes.length);
        nouveauParagraphe = document.createElement('p');
        nouveauParagraphe.textContent = "Dernier paragraphe";
        document.body.appendChild(nouveauParagraphe);
        alert(paragraphes.length);
      }
    </script>
  </head>
  <body>
    <p> Premier paragraphe</p>
    <p> Deuxième paragraphe</p>
    <p> Troisième paragraphe</p>
  </body>
</html>
```



Exemple de NodeList vivant

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Exemple simple DOM</title>
    <script>
      window.onload = function() {
        paragraphes = document.querySelectorAll('p');
        alert(paragraphes.length);
        nouveauParagraphe = document.createElement('p');
        nouveauParagraphe.textContent = "Dernier paragraphe";
        document.body.appendChild(nouveauParagraphe);
        alert(paragraphes.length);
      }
    </script>
  </head>
  <body>
    <p> Premier paragraphe</p>
    <p> Deuxième paragraphe</p>
    <p> Troisième paragraphe</p>
  </body>
</html>
```



Propriété **dataset**

- Il s'agit d'un objet qui contient tous les attributs fixés par la forme **data-***, dans la balise correspondant à l'élément
- Pour chacune des occurrences, il y a une propriété associée dataset, dont le nom correspond à tout ce qui vient après « data- », sans les tirets, et où tout caractère qui succède à un tiret est mis en majuscule

Propriété dataset - exemple

```
<div id="user" data-ident="123456" data-user="mg" data-date-of-birth>  
  Michel Gagnon  
</div>
```

```
var el = document.querySelector('#user');
```

```
// Fixer la date de naissance
```

```
el.dataset.dateOfBirth = '1999-01-01';
```

```
// On ajoute une nouvelle propriété
```

```
el.dataset.autresDonnees = 'mesdonnees';
```

```
document.write('<ul>');
```

```
document.write('<li>' + el.dataset.ident + '</li>');
```

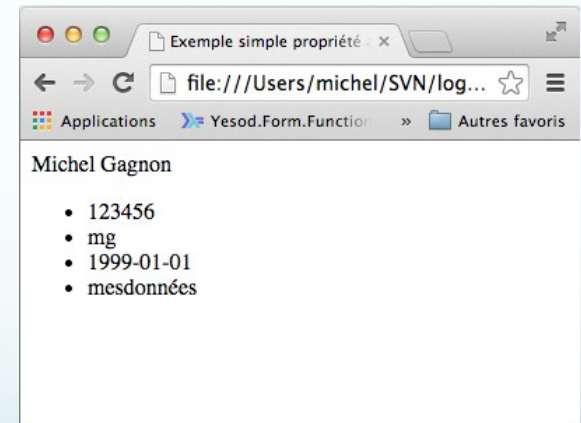
```
document.write('<li>' + el.dataset.user + '</li>');
```

```
document.write('<li>' + el.dataset.dateOfBirth + '</li>');
```

```
document.write('<li>' + el.dataset.autresDonnees + '</li>');
```

```
document.write('</ul>');
```

```
el.dataset.ident = '123456'  
el.dataset.user = 'mg'  
el.dataset.dateOfBirth = ''
```



Événements

- Chaque événement a un nom: *click*, *change*, *load*, *mouseover*, etc.
- Chaque événement a une cible (target), soit l'élément sur lequel il a été déclenché
- Il faut associer à chaque événement une fonction qui sera appelée lorsque celui-ci sera déclenché (gestionnaire d'événement)
- Trois manières d'associer un gestionnaire d'événement à un élément:
 - Le spécifier directement dans la balise (déconseillé)
 - L'associer à un attribut de l'élément (*onclick*, *onchange*, *onload*, *onmouseover*, etc) (pas très conseillé)
 - Utiliser la méthode *addEventListener()* qui, contrairement à la méthode précédente, permet d'ajouter plusieurs gestionnaires à un événement

Événements (suite)

- Le gestionnaire reçoit en paramètre un objet de type **Event** qui fournit de l'information sur l'événement (notamment, la cible)
- L'événement est propagé dans les éléments englobant celui qui l'a déclenché
- Certains événements ont une action par défaut qui leur est associée:
 - Clic sur un hyperlien: se déplacer à l'URL indiquée
 - Clic sur bouton de type **submit**: envoyer au serveur les données d'un formulaire

Interface Event

- Propriétés:
 - **bubbles** : indique si l'événement doit être propagé
 - **target** : l'élément sur lequel l'événement a été déclenché
 - **type** : le nom de l'événement
- Méthodes:
 - **preventDefault()**: annule l'événement
 - **stopPropagation()**: interrompt la propagation de l'événement
- Certains événements ont aussi une propriété **relatedTarget**, qui identifie un élément secondaire (par exemple, pour un événement *mouseover*, ce sera l'élément d'où provient la souris)
- Selon le type spécifique d'événement, d'autres propriétés peuvent apparaître

Événements de formulaires

- **submit** et **reset**
- Différents types d'événement selon le type d'input
- **inputEvent** lancé chaque fois que le contenu d'un input **textarea** est modifié

Événements de fenêtre

- Le plus important est **load**, lancé quand toute la page et ses ressources additionnelles sont chargées et affichées
- Autres événements:
 - **unload, beforeunload**
 - **focus, blur**
 - **resize, scroll**

Événements de souris

- Déclenchés par l'élément le plus imbriqué parmi ceux qui se trouvent sous la souris
- Contiennent des propriétés supplémentaires indiquant la position de la souris et l'état des boutons
- Exemples: **mousemove**, **mousedown**, **mouseup**, **click**, **dblclick**, **mouseover**, **mouseout**
- Remarques sur **mouseout** et **mouseover**:
 - L'événement a une propriété **relatedTarget** pour indiquer l'élément d'où on vient (ou celui où l'on va)
 - La propagation nous oblige à vérifier si l'élément en question est réellement celui qui nous intéresse

Événements de clavier

- Ils sont associés à l'élément qui a le focus et ils sont propagés aux objets **document** et **window**
- Événements de bas niveau: **keydown** et **keyup**
- Si la touche de clavier correspond à un caractère, on a en plus un événement **keypress**, qui spécifie le caractère en question

Événements du DOM (niveau 3)

- **focusin** et **focusout**, sont des alternatives à **focus** et **blur**, mais qui se propagent
- **mouseenter** et **mouseleave** sont des alternatives à **mouseover** et **mouseout**, mais qui ne se propagent pas
- **keypress** a les attributs supplémentaires **key** et **char**
- Événement **textInput**, qui contient les propriétés suivantes:
 - **data**, qui contient le texte entré
 - **inputMethod**, qui identifie la manière dont le texte a été entré (clavier, copier/coller, drop, etc.)

Événements de HTML5

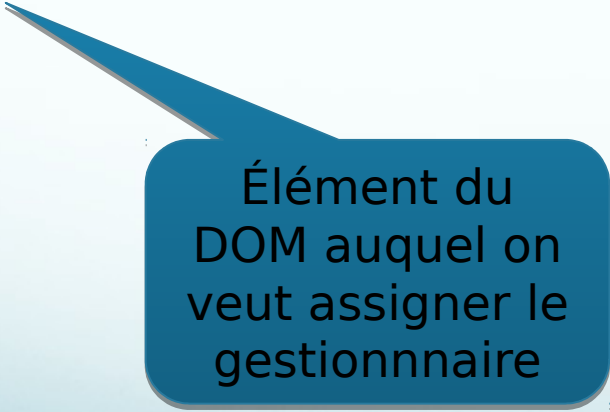
- Événements associés aux balises `<audio>` et `<video>`: **canplay**, **playing**, **pause**, etc.
- Événements associés à l'API drag and drop: **dragstart**, **drag**, **dragenter**, **dragend**, **drop**, **ended**, etc.
- Gestion de l'historique: **hashchange**, **popstate**
- Applications hors-ligne: **cached**, **checking**, **downloading**, **progress**, etc.
- Gestion du stockage local: **storage**

Assignation d'un gestionnaire d'événement

`element.addEventListener(événement, gestionnaire, capture)`

Assignation d'un gestionnaire d'événement

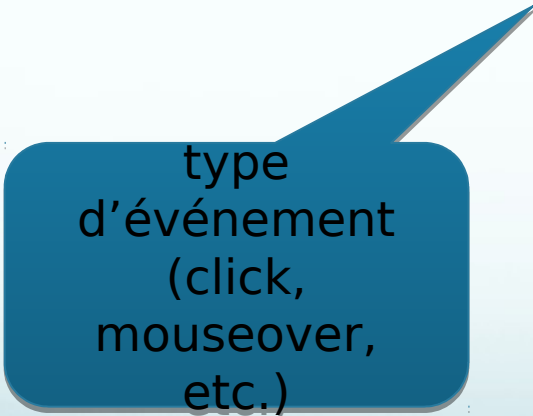
element.addEventListener(événement, gestionnaire, capture)



Élément du
DOM auquel on
veut assigner le
gestionnaire

Assignation d'un gestionnaire d'événement

```
element.addEventListener(événement, gestionnaire, capture)
```



type
d'événement
(click,
mouseover,
etc.)

Assignation d'un gestionnaire d'événement

`element.addEventListener(événement, gestionnaire, capture)`

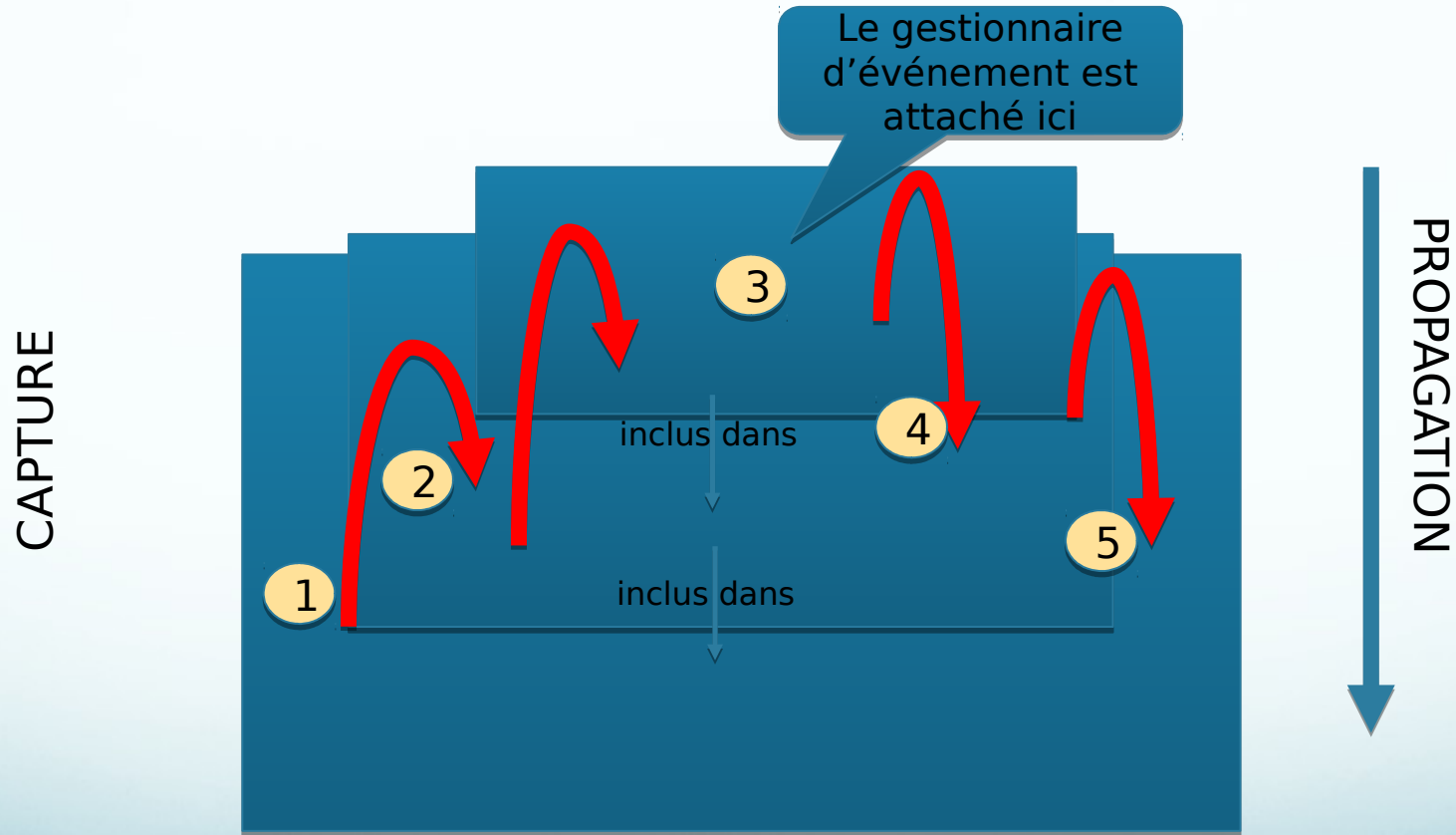
Une fonction qui représente le gestionnaire d'événement (qui sera exécutée chaque fois que l'événement sera déclenché)

Assignation d'un gestionnaire d'événement

`element.addEventListener(événement, gestionnaire, capture)`

true si on veut que le gestionnaire soit exécuté lors de la phase de capture, **false** s'il doit être exécuté lors de la phase de propagation

Propagation d'événements



Gestionnaire associé à un attribut **on***

- Un seul gestionnaire peut être associé à un élément par cette méthode
- Si le gestionnaire retourne **false**, quand il est appelé, le comportement par défaut sera annulé

Exemple - événement click

```
img {  
  border: solid red 1px;  
}  
  
function fixerLargeur(largeur) {  
  document.getElementById("img1").style.borderWidth = largeur + "px";  
}  
var boutonAgrandir = document.querySelector('#boutonAgrandir');  
var boutonReduire = document.querySelector('#boutonReduire');  
boutonAgrandir.addEventListener('click', function(event){ fixerLargeur(20); }, false);  
boutonReduire.addEventListener('click', function(event){ fixerLargeur(5); }, false);  
  
<body>  
<p>  
    
</p>  
<form name="UnFormulaire">  
  <input id="boutonAgrandir"  
    type="button"  
    value="Agrandir la bordure à 20px" />  
  <input id="boutonReduire ">  
    type="button ">  
    value="Réduire la bordure à 5px" />  
</form>  
</body>
```

Exemple - événement **click**

```
img {  
  border: solid red 1px;  
}  
  
function fixerLargeur(largeur) {  
  document.getElementById("img1").style.borderWidth = largeur + "px";  
}  
var boutonAgrandir = document.querySelector('#boutonAgrandir');  
var boutonReduire = document.querySelector('#boutonReduire');  
boutonAgrandir.addEventListener('click', function(event){ fixerLargeur(20); }, false);  
boutonReduire.addEventListener('click', function(event){ fixerLargeur(5); }, false);  
  
<body>  
<p>  
    
</p>  
<form name="UnFormulaire">  
  <input id="boutonAgrandir"  
    type="button"  
    value="Agrandir la bordure à 20px" />  
  <input id="boutonReduire"  
    type="button"  
    value="Réduire la bordure à 5px" />  
</form>  
</body>
```



Exemple -propagation

```
table {
  border: 1px solid red;
  font-size: xx-large;
}
#col1 { background-color: pink; }

function gestionnaire(e) {
  col2 = document.getElementById("col2");
  col2.innerHTML = "Allo";
  e.stopPropagation();
  alert("Propagation de l'événement interrompue");
}

window.onload = function () {
  elem = document.getElementById("ligne1");
  elem.addEventListener("click",gestionnaire, false);
  elem = document.getElementById("tableau");
  elem.addEventListener("click",function(e) {alert('Coucou!');}, false);
}

<body>
  <table id="tableau">
    <tr id="ligne1"><td id="col1">CLIQUEZ ICI</td></tr>
    <tr id="ligne2"><td id="col2">CLIQUEZ ICI AUSSI</td></tr>
  </table>
</body>
```

Exemple -propagation

```
table {  
  border: 1px solid red;  
  font-size: xx-large;  
}  
#col1 { background-color: pink; }  
  
function gestionnaire(e) {  
  col2 = document.getElementById("col2");  
  col2.innerHTML = "Allo";  
  e.stopPropagation();  
  alert("Propagation de l'événement interrompue");  
}  
  
window.onload = function () {  
  elem = document.getElementById("ligne1");  
  elem.addEventListener("click",gestionnaire, false);  
  elem = document.getElementById("tableau");  
  elem.addEventListener("click",function(e) {alert('Coucou!');}, false);  
}  
  
<body>  
  <table id="tableau">  
    <tr id="ligne1"><td id="col1">CLIQUEZ ICI</td></tr>  
    <tr id="ligne2"><td id="col2">CLIQUEZ ICI AUSSI</td></tr>  
  </table>  
</body>
```



Exemple -propagation

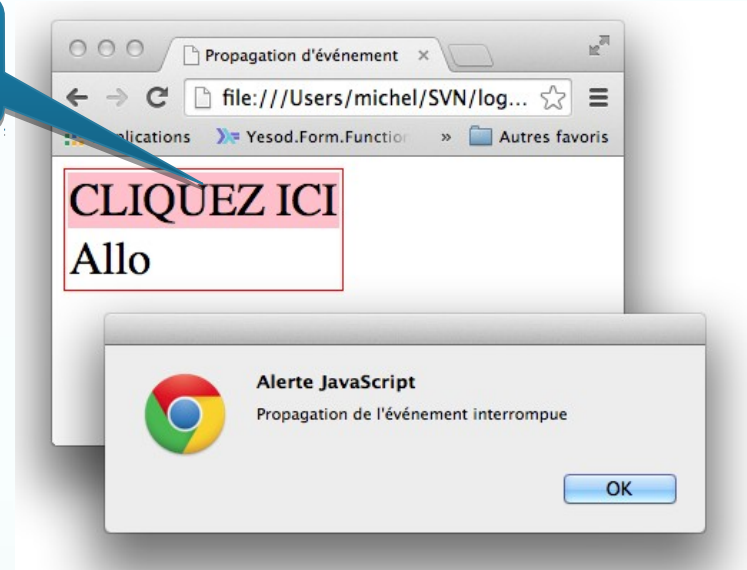
```
table {
  border: 1px solid red;
  font-size: xx-large;
}
#col1 { background-color: pink; }

function gestionnaire(e) {
  col2 = document.getElementById("col2");
  col2.innerHTML = "Allo";
  e.stopPropagation();
  alert("Propagation de l'événement interrompue");
}

window.onload = function () {
  elem = document.getElementById("ligne1");
  elem.addEventListener("click",gestionnaire, false);
  elem = document.getElementById("tableau");
  elem.addEventListener("click",function(e) {alert('Coucou!');}, false);
}

<body>
  <table id="tableau">
    <tr id="ligne1"><td id="col1">CLIQUEZ ICI</td></tr>
    <tr id="ligne2"><td id="col2">CLIQUEZ ICI AUSSI</td></tr>
  </table>
</body>
```

On a
cliqué ici



Exemple -propagation

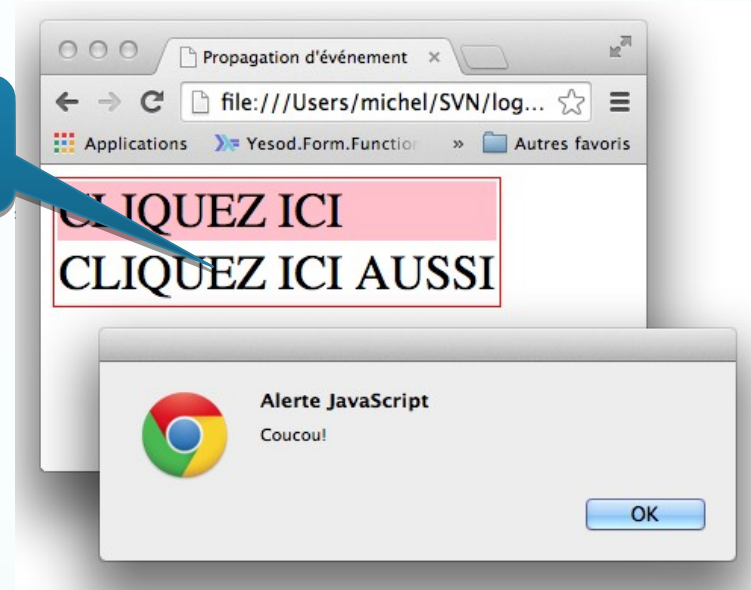
```
table {
  border: 1px solid red;
  font-size: xx-large;
}
#col1 { background-color: pink; }

function gestionnaire(e) {
  col2 = document.getElementById("col2");
  col2.innerHTML = "Allo";
  e.stopPropagation();
  alert("Propagation de l'événement interrompue");
}

window.onload = function () {
  elem = document.getElementById("ligne1");
  elem.addEventListener("click",gestionnaire, false);
  elem = document.getElementById("tableau");
  elem.addEventListener("click",function(e) {alert('Coucou!');}, false);
}

<body>
  <table id="tableau">
    <tr id="ligne1"><td id="col1">CLIQUEZ ICI</td></tr>
    <tr id="ligne2"><td id="col2">CLIQUEZ ICI AUSSI</td></tr>
  </table>
</body>
```

On a
cliqué ici



jQuery

- Bibliothèque Javascript pour simplifier la manipulation du DOM (entre autres)
- Grand avantage: le code devient automatiquement adapté à tous les navigateurs (IE8 inclus)
- Principe de base: des requêtes pour extraire les éléments qu'on désire manipuler
- Ajoute aussi certaines fonctionnalités qui n'existent pas dans le DOM

La fonction **jQuery()** ou **\$**

- Elle est fondamentale: elle prend un argument et retourne un *objet jQuery* sur lequel un grand nombre de manipulations peuvent être réalisées
- Très souvent, lorsque la requête correspond à plusieurs éléments, cet objet est une collection d'items, qui se manipule comme un tableau
- On peut aussi exécuter une méthode sur cette collection d'items, ce qui aura pour effet de l'appliquer à chaque item de cette collection
- Exemples:
 - Pour changer la couleur de fond de tous les paragraphes, on pourrait écrire ceci (ne faites pas ça dans votre application!):
`$('.p').css('background-color','ccc')`
 - Pour ajouter un gestionnaire de l'événement click à tous les éléments ayant la classe cliquerPourCacher:
`$('.cliquerPourCacher').click(function() { $(this).hide(); });`

Comment invoquer la fonction **jQuery()**

1. On lui passe un sélecteur CSS

- Si on lui passe un élément comme second argument, la recherche se fera seulement dans cet élément et ses descendants
(ex. `$('#p', document.getElementById('mondiv'))`)

2. On lui passe un élément, l'objet **document** ou l'objet **window**: dans ce cas, il retourne un objet jQuery qui englobe celui qui est passé en paramètre (ex. `$(document)`)

3. On lui passe du code HTML

- On peut lui passer en second argument un objet qui indique les valeurs de certaines propriétés

On peut aussi lui passer comme second argument un élément qui servira de contexte

(ex. `$('', { 'class': 'unLien' })` ou `$('')`)

4. On lui passe une fonction: celle-ci est exécutée lorsque la construction du DOM est terminée (on lui passe en argument le document) (ex. `$(function() { ... })`)

Comment itérer sur une collection d'items

- Méthode **each()**: reçoit en paramètre une fonction qui sera appelée pour chaque item de la collection
 - La fonction reçoit l'indice de l'item comme premier argument et l'élément concerné comme second argument
 - **this** représente aussi l'élément concerné
 - Si la fonction retourne **false**, l'itération s'arrête
- Itération implicite: beaucoup de méthodes itèrent de manière implicite sur tous les items:
 - **\$('#button').click(function() { ... })**

Méthode **map()**

- Comme **each()**, elle reçoit en paramètre une fonction qui sera appelée sur tous les items de la collection
- Retourne une nouvelle collection qui contient tous les résultats obtenus par l'application de la fonction à chaque item
- Exemple:
 - `$('#div').map(function() {
 return this.id;
}).toArray().sort()`

Méthode **index()**

- Si elle reçoit un élément en paramètre, elle retourne l'indice de cet élément
- Si elle reçoit un sélecteur CSS, elle retourne l'indice du premier élément qui correspond à ce sélecteur

Méthode **is()**

- Prend un sélecteur CSS comme argument
- Retourne **true** si l'élément correspond au sélecteur en paramètre

- Exemple:

```
$('#div').each(function() {  
  if ($(this).is(':hidden')) return;  
});
```

Manipulation des attributs

- La même méthode est utilisée pour lire la valeur courante et pour fixer une nouvelle valeur
- Lorsqu'il s'agit d'une collection, l'opération est appliquée à tous les items de la collection, s'il s'agit de fixer une nouvelle valeur
- S'il s'agit de lire la valeur courante, cela sera appliqué seulement sur le premier item
- On peut passer comme argument un objet pour fixer la valeur de plusieurs attributs
- Souvent, on peut aussi passer une fonction qui retournera la nouvelle valeur désirée: dans ce cas, elle sera appliquée à l'item et recevra deux arguments (l'indice de l'item et sa valeur courante)

Méthodes de manipulation des attributs

- `attr()`
- `css()`
- `addClass()`, `removeClass()`, `toggleClass()`
- `val()`
- `text()` et `html()`
- `width()`, `height()`, `innerWidth()`, `outerWidth()`, ...
- `data()`, `removeData()`

Modification de la structure du document

- Ajout d'éléments:
 - `append()`, `prepend()`, `before()`, `after()`, `replaceWith()`
 - `appendTo()`, `prependTo()`, `insertBefore()`, `insertAfter()`, `replaceAll()`
- Copie d'éléments:
 - `clone()`
 - Par défaut, ne copie pas les gestionnaires d'événements attachés
- Enrobage: `wrap()`, `wrapInner()`, `wrapAll()`
- Retrait d'éléments: `empty()`, `remove()`, `detach()`, `unwrap()`

Événements en jQuery

- Pour chaque événement du DOM, il y a une méthode (par exemple: click -> click())
- Ajouts:
 - hover()**: prend deux fonctions en argument, soit une pour **mouseenter** et une autre pour **mouseleave**
 - toggle()**: pour chaque clic, retire ou affiche l'item en alternance
 - toggleClass()**: pour chaque clic, retire ou ajoute la classe en alternance
- Le gestionnaire reçoit en paramètre un objet correspondant à l'événement (attributs de l'objet: target, relatedTarget, which, type, pageX, pageY etc. ; méthodes: preventDefault(), stopPropagation())
- Si un gestionnaire retourne **false**, la propagation et le comportement par défaut sont éliminés

Événements « vivant »

- Supposons que le code suivant s'exécute:
`$('a').click(gestionnaire)`
- Si d'autres éléments **`<a>`** s'ajoutent par la suite, le gestionnaire n'y sera pas attaché
- Pour obtenir ce comportement, il faut utiliser la méthode **`delegate()`**:
`$(document).delegate('a','click',gestionnaire)`
- Principe: lorsque la propagation atteint l'élément sur lequel **`delegate()`** est appelé (le document, dans l'exemple précédent), on détermine si la cible correspond au sélecteur et si c'est le cas on exécute le gestionnaire sur cette cible

Méthodes de sélection

- On peut utiliser les sélecteurs CSS
- Ajout de pseudo-classes:
:animated, :button, :checkbox, :contains(text), :eq(n)
,
:even, :file, :first (ne pas confondre avec :first-child)
:gt(n), :has(sel), :header, :hidden, :image, :input, :last,
:lt(n), :odd, :parent, :password, :radio, :reset, :selected
:submit, :text, :visible
- Exemple: `$('p:nth-child(3):not(:has(a))')`

Méthodes de sélection

- On peut aussi utiliser des méthodes de sélection définies par jQuery
- Méthode **slice(m,n)**: sélectionne les items des indices m à n
- Méthode **filter()**: sélectionne les items correspondant au critère passé en paramètre:
 - Un sélecteur CSS
 - Une autre collection jQuery
 - Une fonction prédicat (qui retourne une valeur booléenne)
(ex. `$('p').filter('.middle')`)
- Méthode **not()**
(ex. `$('p').not(':even')`)
- Méthode **has()**
(ex. `$('p').has('strong')`)

Méthodes de sélection

- Méthode **find()**: cherche parmi les descendants de l'item concerné

Exemple: `$('div').find('a')` pour obtenir tous les éléments `<a>` inclus dans un `<div>`

- Méthodes **children()**, **contents()**, **next()**, **prev()**, **nextAll()**, **prevAll()**, **siblings()**, **parents()**

Méthodes de sélection

- Méthode **end()**: pour revenir à niveau précédent dans la chaîne

- Exemple:

```
var divs = $('div');  
var paras = divs.find('p');  
paras.addClass('important');  
divs.css('border','solid black 1pt')
```

est équivalent à ceci:

```
$('div').find('p').addClass('important').end().css(...)
```

ou encore:

```
$('div').css(...).find('p').addClass('important')
```