



# Programmation Événementielle en JavaScript



Philippe Genoud

*Philippe.Genoud@univ-grenoble-alpes.fr*



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

- Événement = tout changement d'état du navigateur
  - interaction de l'utilisateur
    - clic sur un bouton ou sur un lien
    - survol de la souris au dessus d'un élément HTML
    - modification d'une zone de saisie
    - appui sur une touche de clavier
    - redimensionnement de la fenêtre
    - ...
  - autre
    - fin du chargement de la page,
    - erreur de connexion
    - ...
- JavaScript est un langage événementiel
  - détecte les événements
  - offre la possibilité d'y réagir en déclenchant des traitements spécifiques

<https://developer.mozilla.org/en-US/docs/Web/Events>

## Mouse events

Event Name	Fired When
<code>auxclick</code>	A pointing device button (ANY non-primary button) has been pressed and released on an element.
<code>click</code>	A pointing device button (ANY button; soon to be primary button only) has been pressed and released on an element.
<code>contextmenu</code>	The right button of the mouse is clicked (before the context menu is displayed).
<code>dblclick</code>	A pointing device button is clicked twice on an element.
<code>mousedown</code>	A pointing device button is pressed on an element.
<code>mouseenter</code>	A pointing device is moved onto the element that has the listener attached.
<code>mouseleave</code>	A pointing device is moved off the element that has the listener attached.
<code>mousemove</code>	A pointing device is moved over an element (fired continuously as the mouse moves).
<code>mouseover</code>	A pointing device is moved onto the element that has the listener attached or onto one of its children.
<code>mouseout</code>	A pointing device is moved off the element that has the listener attached or off one of its children.
<code>mouseup</code>	A pointing device button is released over an element.
<code>pointerlockchange</code>	The pointer was locked or released.
<code>pointerlockerror</code>	It was impossible to lock the pointer for technical reasons or because the permission was denied.
<code>select</code>	Some text is being selected.
<code>wheel</code>	A wheel button of a pointing device is rotated in any direction.

## Keyboard events

Event Name	Fired When
<code>keydown</code>	ANY key is pressed
<code>keypress</code>	ANY key (except <code>Shift</code> , <code>Fn</code> , or <code>CapsLock</code> ) is in a pressed position (fired continuously).
<code>keyup</code>	ANY key is released

## Form events

Event Name	Fired When
<code>reset</code>	The reset button is pressed
<code>submit</code>	The submit button is pressed

## Network events

Event Name	Fired When
<code>online</code>	The browser has gained access to the network.
<code>offline</code>	The browser has lost access to the network.

## Resource events

Event Name	Fired When
<code>error</code>	A resource failed to load.
<code>abort</code>	The loading of a resource has been aborted.
<code>load</code>	A resource and its dependent resources have finished loading.
<code>loadend</code>	The browser has finished downloading the document and its resources are about to be unloaded.

## Focus events

Event Name	Fired When
<code>focus</code>	An element has received focus (does not bubble).
<code>blur</code>	An element has lost focus (does not bubble).

## WebSocket events

Event Name	Fired When
<code>open</code>	A WebSocket connection has been established.

## CSS Animation events

Event Name	Fired When
<code>animationstart</code>	A CSS animation has started.
<code>animationend</code>	A CSS animation has finished.
<code>animationcancel</code>	A CSS animation has aborted.

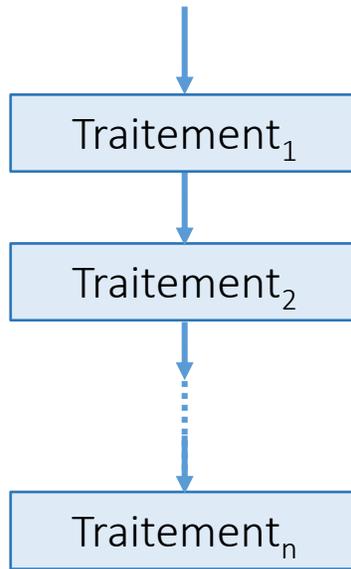
## Clipboard events

Event Name	Fired When
<code>cut</code>	The selection has been cut and copied to the clipboard
<code>copy</code>	The selection has been copied to the clipboard
<code>paste</code>	The item from the clipboard has been pasted

Événements web ne font pas partie du du noyau (core) JavaScript - ils font partie des APIs JavaScript intégrées aux navigateurs

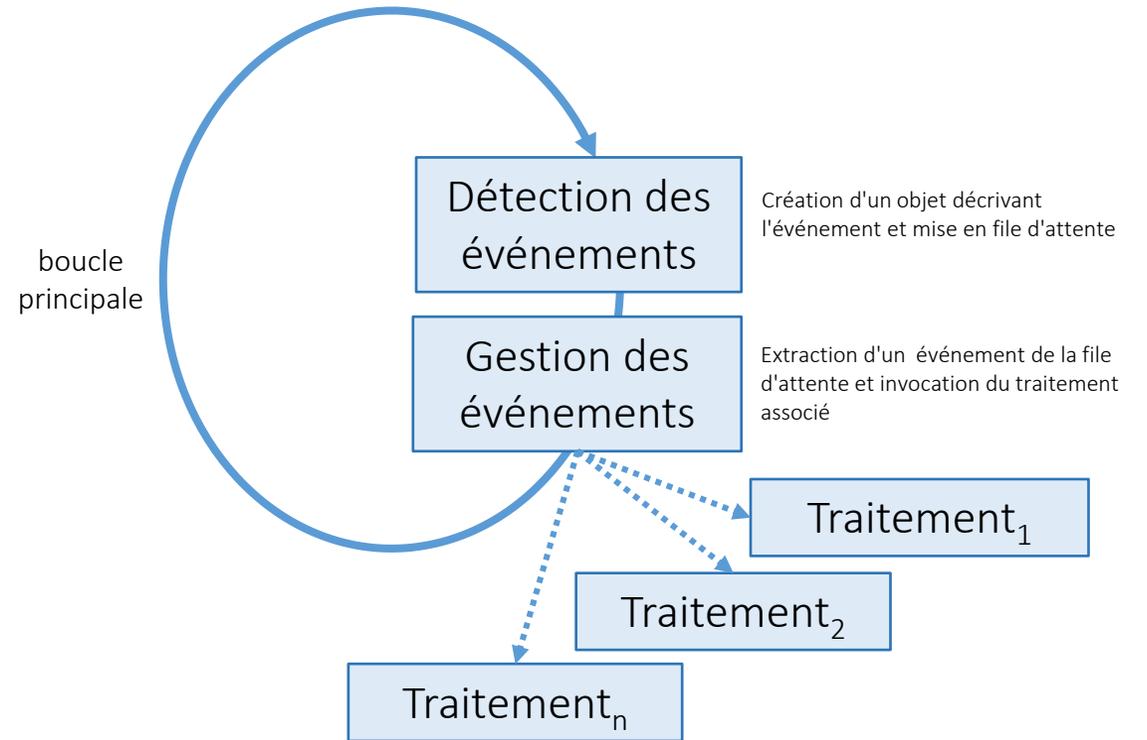
10/11/2023

Programmation séquentielle



Enchaînement des traitements est toujours le même

Programmation événementielle  
(event-driven programming)



Enchaînement des traitements déterminé par les événements

- ordre des événements inconnu
- survenue d'un événement non garantie

- JavaScript : langage événementiel
  - flux d'exécution du code, déterminé principalement par les interactions avec l'environnement (activation d'un lien, mouvement de la souris, chargement du contenu du document, ...)
  - le développeur a un contrôle limité sur celui-ci
- Spécifications gestion des événements dans le navigateur
  - W3C
    - DOM Level 2 Events Specification  
<http://www.w3.org/TR/DOM-Level-2-Events/> [nov. 2000]
      - auparavant chaque navigateur avait ses propres particularités pour gérer les événements
    - DOM level 3 UI Events <https://www.w3.org/TR/DOM-Level-3-Events> [nov. 2016]
  - WHATWG
    - DOM Living standard  
<https://dom.spec.whatwg.org/>



- A chaque événement disponible correspond un **gestionnaire d'événement** (*event handler* ou *event listener*) : bloc de code JavaScript exécuté lorsque l'événement est déclenché
- Trois manières de définir des gestionnaires d'événements



- Directement dans le code HTML

```
<button onclick="calculer()">=</button>
```

- Via les propriétés des éléments du DOM

```
<button id="btnCalculer">=</button>
```

```
let btnCalculer = document.querySelector("#btnCalculer");  
btnCalculer.onclick = calculer;
```

- Ajout/suppression d'un *écouteur* d'événements

```
<button id="btnCalculer">=</button>
```

```
let btnCalculer = document.querySelector("#btnCalculer");  
btnCalculer.addEventListener("click", calculer);
```

- Directement dans le code HTML

- Forme générale

`<element ontypeevenement="code javascript">`

Attribut définissant le type d'événement auquel on veut associer un traitement  
*onClick, onChange, onMouseenter, onMouseover, onkeyup...*

Code JavaScript exécuté lorsque l'événement intervient

- Exemples

```
<button onclick="calculer()">=</button>
```

- Possibilité de définir plusieurs instructions séparées par ;

```
<button onclick="console.log('lancement du calcul'); calculer()">=</button>
```



Mélange du traitement JavaScript avec contenu HTML

- Lisibilité --
- Réutilisation --
- Complexifie l'analyse du code HTML



[https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building\\_blocks/Events](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events)

**Inline event handlers — don't use these**

- Via les propriétés des éléments du DOM

- Forme générale
- Exemple

```
<html>
<head>
  ...
</head>
<body>
  ...
  <button id="btnCalculer"></button>
  ...
  <script src="./js/calculatrice.js"></script>
</body>
```

*target*.**ontypeevenement** = *fonctionJavascript* ;

Attribut définissant le type d'événement auquel on veut associer un traitement

Objet du DOM cible de l'événement (**Element, Window, Document**)

Identifiant de fonction ou fonction anonyme. La fonction est invoquée avec un paramètre **evt**, référence vers un objet **Event** représentant l'événement

Utilisation d'une fonction déclarée

```
function calculer() {
  let operande1 = parseFloat(document.querySelector("#op1").value);
  let operande2 = parseFloat(document.querySelector("#op2").value);
  let operateur = document.querySelector("#operateur").value;
  switch (operateur) {
    ...
  }
  document.querySelector("#resultat").innerHTML = res;
}

let btnCalculer = document.querySelector("#btnCalculer");
btnCalculer.onclick = calculer;
```

Utilisation d'une fonction anonyme

```
let btnCalculer = document.querySelector("#btnCalculer");

btnCalculer.onclick = function() {
  let operande1 = parseFloat(document.querySelector("#op1").value);
  let operande2 = parseFloat(document.querySelector("#op2").value);
  let operateur = document.querySelector("#operateur").value;
  switch (operateur) {
    ...
  }
  document.querySelector("#resultat").innerHTML = res;
};
```

 Bonne séparation entre données (code HTML) et traitement (code JavaScript)  
Permet de définir le traitement des événements au moment opportun (par exemple à la fin du chargement de la page)

 Peut interférer avec le traitement des événements définis par d'autres fonctionnalités

- Ajout/suppression d'un *écouteur* d'événements

- Forme générale `target.addEventListener(type, listener [, options]);`  
`target.addEventListener(type, listener [, useCapture]);`

Chaîne de caractères définissant le type d'événement concerné (**Attention** : sensible à la casse).

Gestionnaire de l'événement : identifiant de fonction ou fonction anonyme ou fléchée. La fonction est invoquée avec un paramètre `evt`, référence vers un objet `Event` représentant l'événement

`boolean`, si `true` le gestionnaire sera appelé lors de la phase de capture (1), si `false` le gestionnaire sera appelé lors de la phase cible (2) ou de bouillonnement (3). `false` par défaut.

Objet d'options spécifiant comportement du gestionnaire

```
{ capture: boolean;
  once: boolean;
  passive : boolean
}
```

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

- Exemple

```
<html>
<head>
  ...
</head>
<body>
  ...
  <button id="btnCalculer">=</button>
  ...
  <script src="./js/calculatrice.js"></script>
</body>
```

Utilisation d'une fonction anonyme

```
let btnCalculer = document.querySelector("#btnCalculer");

btnCalculer.addEventListener("click", function() {
  let operande1 = parseFloat(document.querySelector("#op1").value);
  let operande2 = parseFloat(document.querySelector("#op2").value);
  let operateur = document.querySelector("#operateur").value;
  switch (operateur) {
    ...
  }
  document.querySelector("#resultat").innerHTML = res;
});
```

 Bonne séparation entre données (code HTML) et traitement (code JavaScript) Permet de définir le traitement des événements au moment opportun (par exemple à la fin du chargement de la page)

 N'interfère pas avec les traitement des événements définis par d'autres fonctionnalités. **Une même cible peut avoir plusieurs gestionnaires (listeners)**

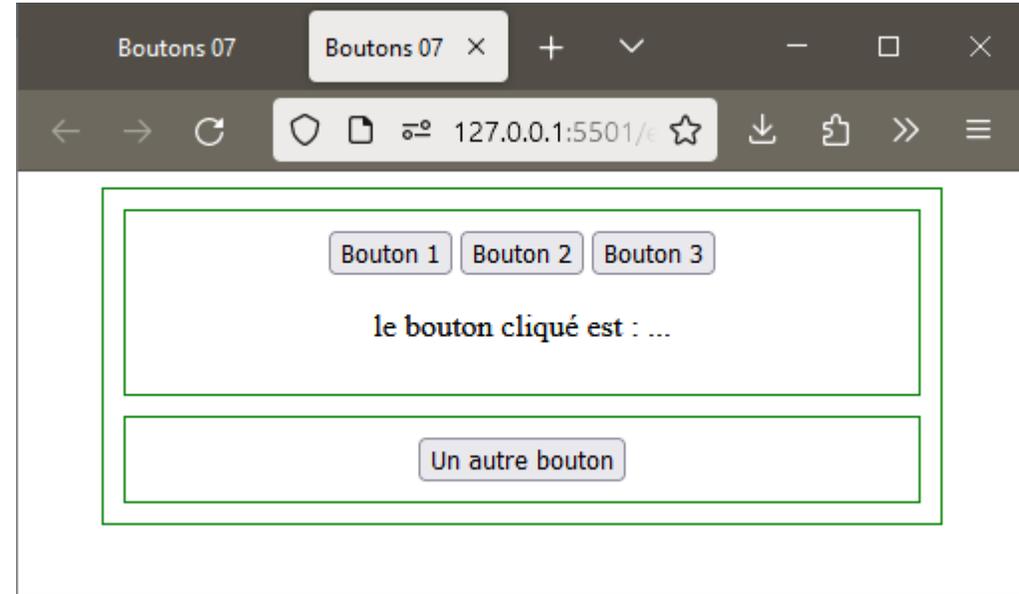
```
body {
  text-align: center;
}

div {
  border: solid 1px green;
  margin: auto;
  padding: 10px;
}

div + div {
  margin-top: 10px;
}

.container {
  width: 400px;
}

<body>
  <div class="container">
    <div>
      <button>Bouton 1</button>
      <button>Bouton 2</button>
      <button>Bouton 3</button>
      <p>le bouton cliqué est :...</p>
    </div>
    <div>
      <button>Un autre bouton</button>
    </div>
  </div>
</body>
```



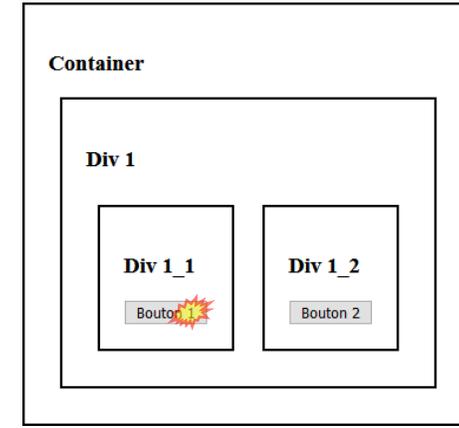
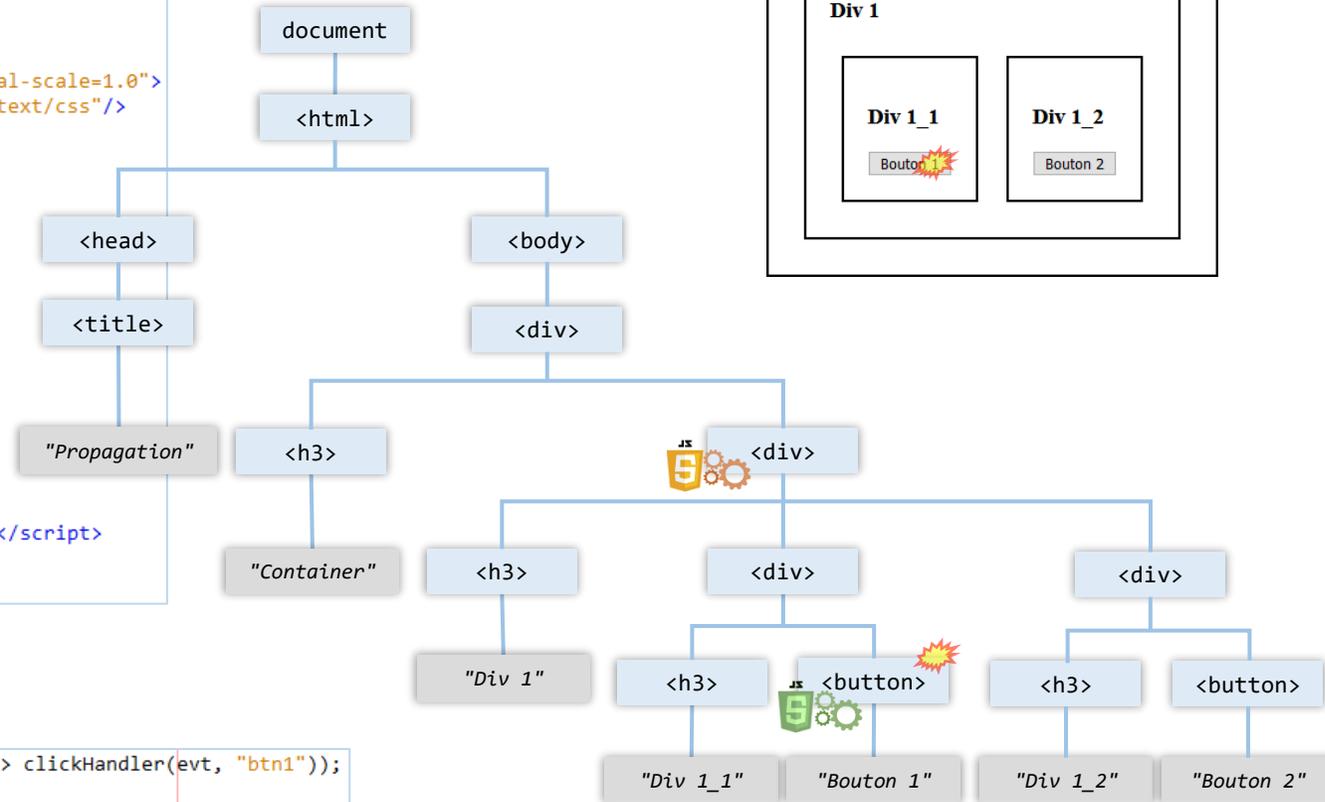
Que se passe-t-il lorsque l'utilisateur clique sur Bouton1 ?

- principes généraux

```

<html>
  <head>
    <title>Propagation</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="testpropagation.css" rel="stylesheet" type="text/css"/>
  </head>
  <body id="body">
    <div id="container">
      <h3>Container</h3>
      <div id="div1">
        <h3>Div 1</h3>
        <div id="div1_1">
          <h3>Div 1_1</h3>
          <button id="btn1">Bouton 1</button>
        </div>
        <div id="div1_2">
          <h3>Div 1_2</h3>
          <button id="btn2">Bouton 2</button>
        </div>
      </div>
    </div>
    <script src="testpropagation.js" type="text/javascript"></script>
  </body>
</html>

```



```

document.querySelector("#btn1").addEventListener("click", (evt) => clickHandler(evt, "btn1"));
document.querySelector("#div1").addEventListener("click", (evt) => clickHandler(evt, "div1"));

function clickHandler(evt, eltId) {
  console.log(`click handler de ${eltId}`);
  console.log(`  current target: ${evt.currentTarget.id}`);
  console.log(`  evt target    : ${evt.target.id}`);
}

```

**Solution 1**  
 click handler de btn1  
 current target: btn1  
 evt target : btn1

**Solution 2**  
 click handler de div1  
 current target: div1  
 evt target : btn1

**Solution 3**  
 click handler de btn1  
 current target: btn1  
 evt target : btn1  
 click handler de div1  
 current target: div1  
 evt target : btn1

**Solution 4**  
 click handler de div1  
 current target: div1  
 evt target : btn1  
 click handler de btn1  
 current target: btn1  
 evt target : btn1

**Solution 5**  
 Aucune des solutions proposées

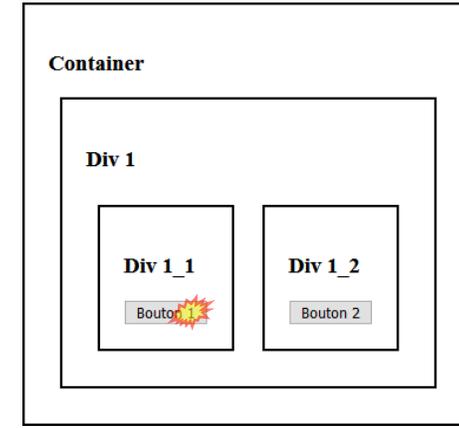
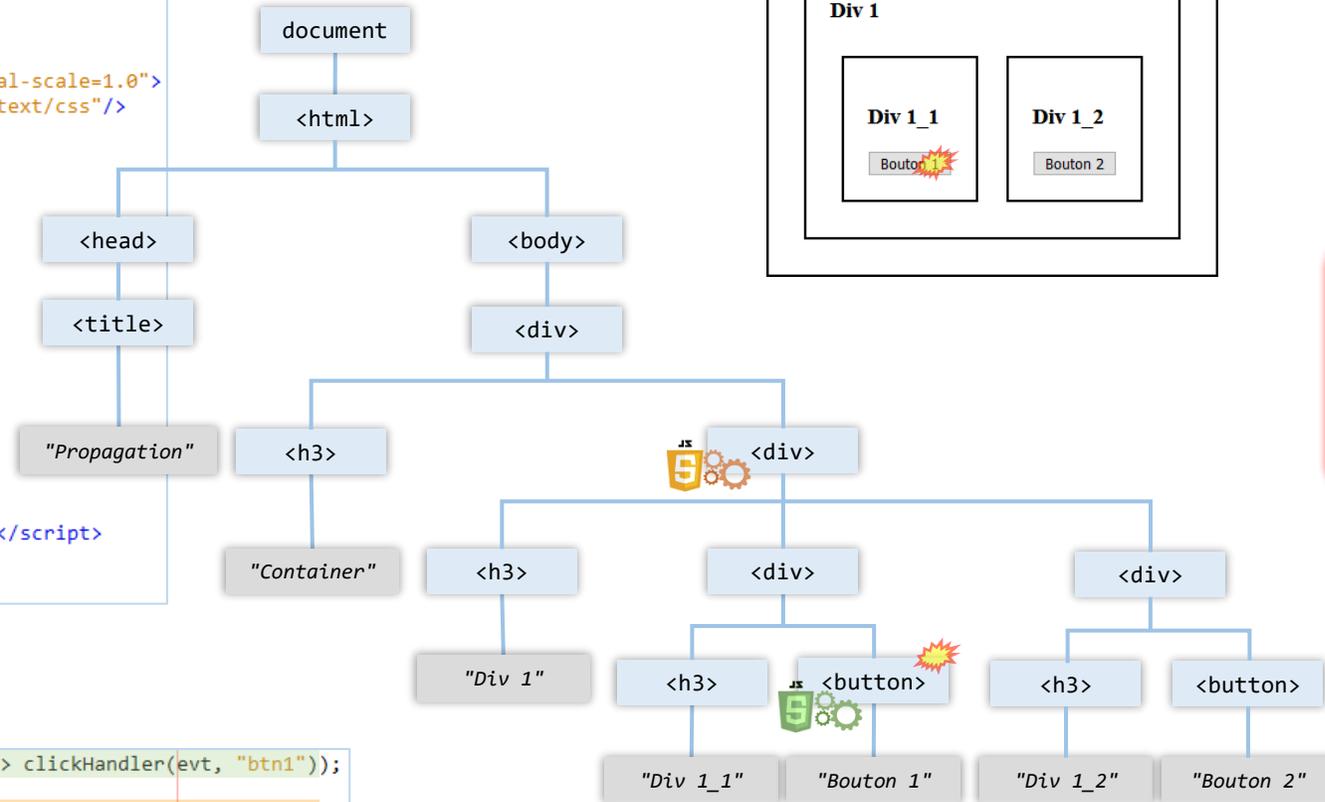


[Voir le code](#)

Que se passe-t-il lorsque l'utilisateur clique sur Bouton1 ?

- principes généraux

```
<html>
  <head>
    <title>Propagation</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="testpropagation.css" rel="stylesheet" type="text/css"/>
  </head>
  <body id="body">
    <div id="container">
      <h3>Container</h3>
      <div id="div1">
        <h3>Div 1</h3>
        <div id="div1_1">
          <h3>Div 1_1</h3>
          <button id="btn1">Bouton 1</button>
        </div>
        <div id="div1_2">
          <h3>Div 1_2</h3>
          <button id="btn2">Bouton 2</button>
        </div>
      </div>
    </div>
    <script src="testpropagation.js" type="text/javascript"></script>
  </body>
</html>
```



### Solution 1

```
click handler de btn1
current target: btn1
evt target : btn1
```

### Solution 2

```
click handler de div1
current target: div1
evt target : btn1
```

### Solution 3

```
click handler de btn1
current target: btn1
evt target : btn1
click handler de div1
current target: div1
evt target : btn1
```

### Solution 4

```
click handler de div1
current target: div1
evt target : btn1
click handler de btn1
current target: btn1
evt target : btn1
```

### Solution 5

Aucune des solutions proposées

```
document.querySelector("#btn1").addEventListener("click", (evt) => clickHandler(evt, "btn1"));
document.querySelector("#div1").addEventListener("click", (evt) => clickHandler(evt, "div1"));

function clickHandler(evt, eltId) {
  console.log(`click handler de ${eltId}`);
  console.log(`  current target: ${evt.currentTarget.id}`);
  console.log(`  evt target : ${evt.target.id}`);
}
```

[Voir le code](#)

## • principes généraux

```

<html>
  <head>
    <title>Propagation</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="testpropagation.css" rel="stylesheet" type="text/css"/>
  </head>
  <body id="body">
    <div id="container">
      <h3>Container</h3>
      <div id="div1">
        <h3>Div 1</h3>
        <div id="div1_1">
          <h3>Div 1_1</h3>
          <button id="btn1">Bouton 1</button>
        </div>
        <div id="div1_2">
          <h3>Div 1_2</h3>
          <button id="btn2">Bouton 2</button>
        </div>
      </div>
    </div>
    <script src="testpropagation.js" type="text/javascript"></script>
  </body>
</html>

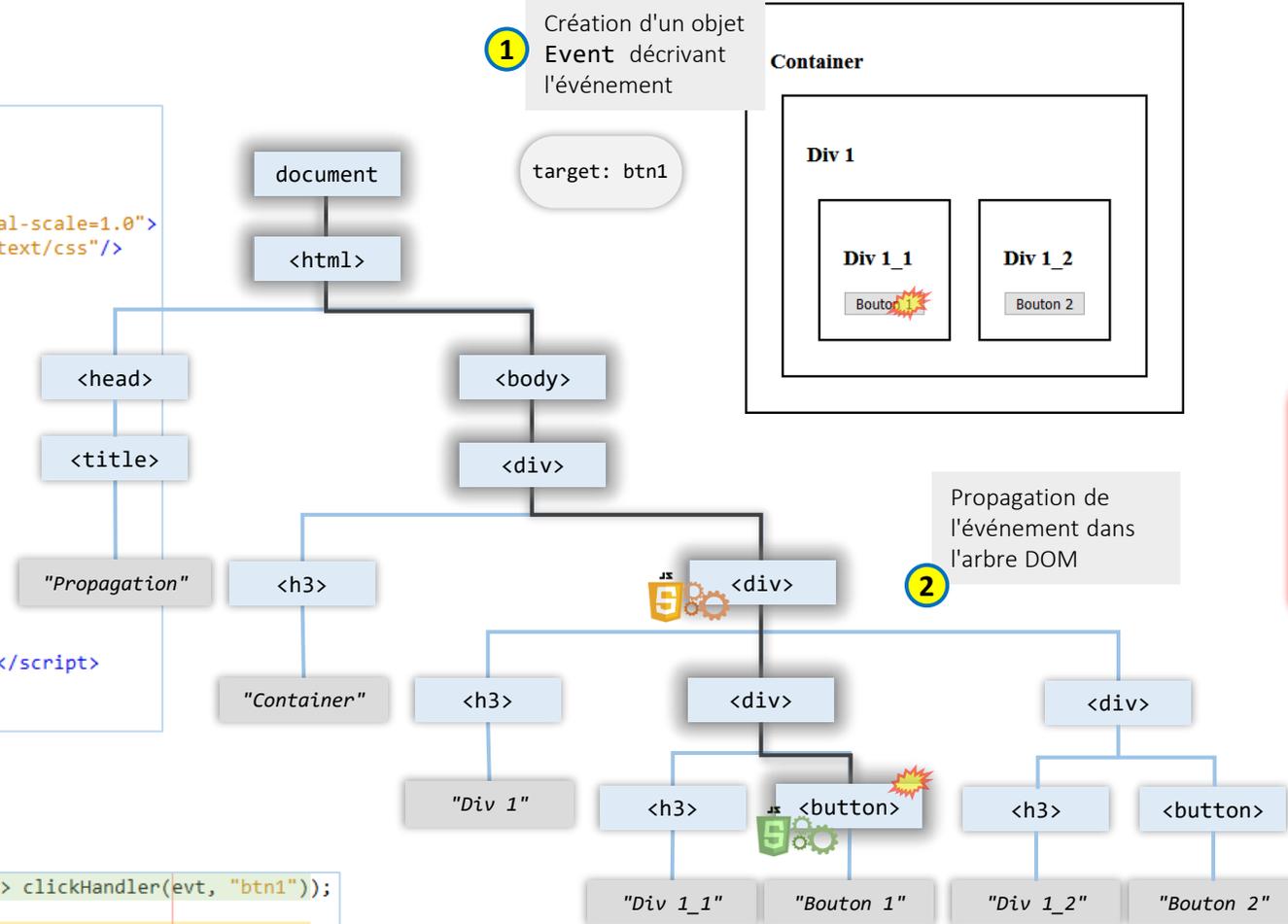
```

```

document.querySelector("#btn1").addEventListener("click", (evt) => clickHandler(evt, "btn1"));
document.querySelector("#div1").addEventListener("click", (evt) => clickHandler(evt, "div1"));

function clickHandler(evt, eltId) {
  console.log(`click handler de ${eltId}`);
  console.log(`  current target: ${evt.currentTarget.id}`);
  console.log(`  evt target   : ${evt.target.id}`);
}

```



Que se passe-t-il lorsque l'utilisateur clique sur Bouton1 ?

### Solution 1

```

click handler de btn1
current target: btn1
evt target   : btn1

```

### Solution 2

```

click handler de div1
current target: div1
evt target   : btn1

```

### Solution 3

```

click handler de btn1
current target: btn1
evt target   : btn1
click handler de div1
current target: div1
evt target   : btn1

```

### Solution 4

```

click handler de div1
current target: div1
evt target   : btn1
click handler de btn1
current target: btn1
evt target   : btn1

```

### Solution 5

Aucune des solutions proposées

[Voir le code](#)

## • principes généraux

```

<html>
  <head>
    <title>Propagation</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="testpropagation.css" rel="stylesheet" type="text/css"/>
  </head>
  <body id="body">
    <div id="container">
      <h3>Container</h3>
      <div id="div1">
        <h3>Div 1</h3>
        <div id="div1_1">
          <h3>Div 1_1</h3>
          <button id="btn1">Bouton 1</button>
        </div>
        <div id="div1_2">
          <h3>Div 1_2</h3>
          <button id="btn2">Bouton 2</button>
        </div>
      </div>
    </div>
    <script src="testpropagation.js" type="text/javascript"></script>
  </body>
</html>

```

```

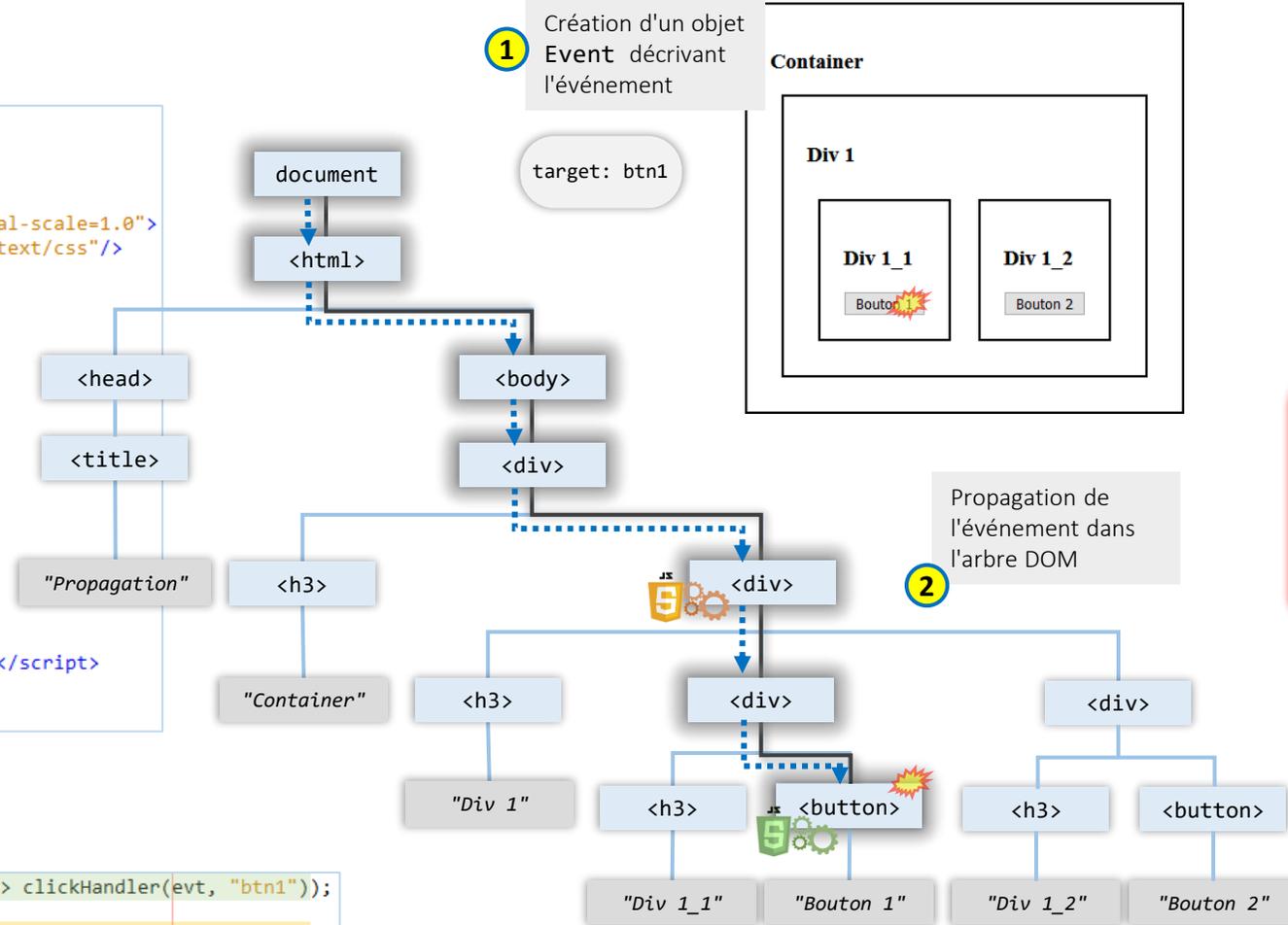
document.querySelector("#btn1").addEventListener("click", (evt) => clickHandler(evt, "btn1"));
document.querySelector("#div1").addEventListener("click", (evt) => clickHandler(evt, "div1"));

function clickHandler(evt, eltId) {
  console.log(`click handler de ${eltId}`);
  console.log(`  current target: ${evt.currentTarget.id}`);
  console.log(`  evt target   : ${evt.target.id}`);
}

```

[Voir le code](#)

Que se passe-t-il lorsque l'utilisateur clique sur Bouton1 ?



### Solution 1

```

click handler de btn1
current target: btn1
evt target   : btn1

```

### Solution 2

```

click handler de div1
current target: div1
evt target   : btn1

```

### Solution 3

```

click handler de btn1
current target: btn1
evt target   : btn1
click handler de div1
current target: div1
evt target   : btn1

```

### Solution 4

```

click handler de div1
current target: div1
evt target   : btn1
click handler de btn1
current target: btn1
evt target   : btn1

```

- Les événements se propagent selon un flux bien précis qui se décompose en 3 phases
  - phase 1 (capture) : l'événement se propage du nœud **document** (inclus) au nœud cible (*target*) exclu

## • principes généraux

```

<html>
  <head>
    <title>Propagation</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="testpropagation.css" rel="stylesheet" type="text/css"/>
  </head>
  <body id="body">
    <div id="container">
      <h3>Container</h3>
      <div id="div1">
        <h3>Div 1</h3>
        <div id="div1_1">
          <h3>Div 1_1</h3>
          <button id="btn1">Bouton 1</button>
        </div>
        <div id="div1_2">
          <h3>Div 1_2</h3>
          <button id="btn2">Bouton 2</button>
        </div>
      </div>
    </div>
    <script src="testpropagation.js" type="text/javascript"></script>
  </body>
</html>

```

```

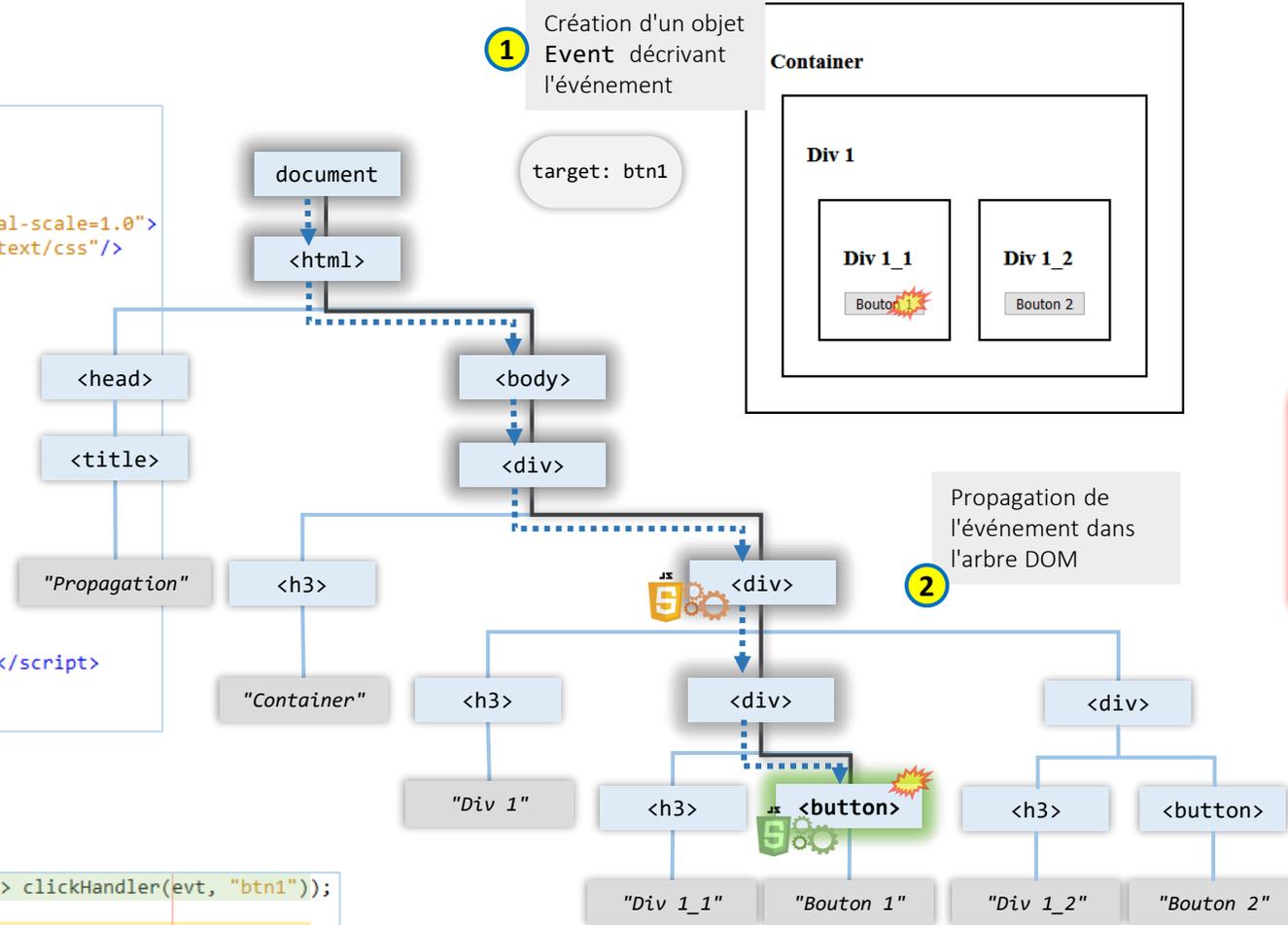
document.querySelector("#btn1").addEventListener("click", (evt) => clickHandler(evt, "btn1"));
document.querySelector("#div1").addEventListener("click", (evt) => clickHandler(evt, "div1"));

function clickHandler(evt, eltId) {
  console.log(`click handler de ${eltId}`);
  console.log(`  current target: ${evt.currentTarget.id}`);
  console.log(`  evt target   : ${evt.target.id}`);
}

```

[Voir le code](#)

Que se passe-t-il lorsque l'utilisateur clique sur Bouton1 ?



### Solution 1

```

click handler de btn1
current target: btn1
evt target   : btn1

```

### Solution 2

```

click handler de div1
current target: div1
evt target   : btn1

```

### Solution 3

```

click handler de btn1
current target: btn1
evt target   : btn1
click handler de div1
current target: div1
evt target   : btn1

```

### Solution 4

```

click handler de div1
current target: div1
evt target   : btn1
click handler de btn1
current target: btn1
evt target   : btn1

```

- Les événements se propagent selon un flux bien précis qui se décompose en 3 phases
  - 1 phase 1 (capture) : l'événement se propage du nœud document (inclus) au nœud cible (target) exclu
  - 2 phase 2 (cible target) : l'événement atteint le nœud cible

## • principes généraux

```

<html>
  <head>
    <title>Propagation</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="testpropagation.css" rel="stylesheet" type="text/css"/>
  </head>
  <body id="body">
    <div id="container">
      <h3>Container</h3>
      <div id="div1">
        <h3>Div 1</h3>
        <div id="div1_1">
          <h3>Div 1_1</h3>
          <button id="btn1">Bouton 1</button>
        </div>
        <div id="div1_2">
          <h3>Div 1_2</h3>
          <button id="btn2">Bouton 2</button>
        </div>
      </div>
    </div>
    <script src="testpropagation.js" type="text/javascript"></script>
  </body>
</html>

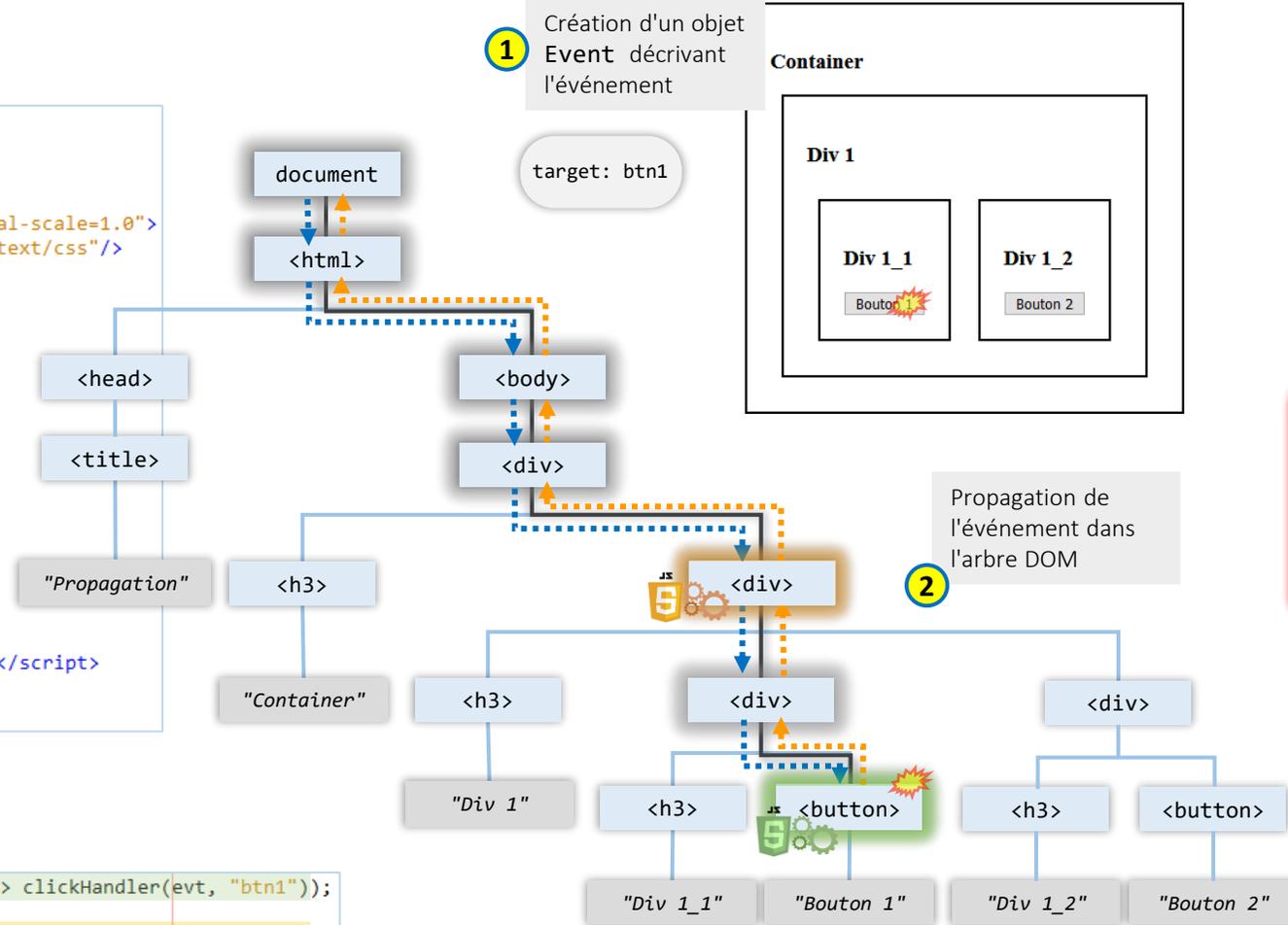
```

```

document.querySelector("#btn1").addEventListener("click", (evt) => clickHandler(evt, "btn1"));
document.querySelector("#div1").addEventListener("click", (evt) => clickHandler(evt, "div1"));

function clickHandler(evt, eltId) {
  console.log(`click handler de ${eltId}`);
  console.log(`  current target: ${evt.currentTarget.id}`);
  console.log(`  evt target   : ${evt.target.id}`);
}

```



Que se passe-t-il lorsque l'utilisateur clique sur Bouton1 ?

### Solution 1

```

click handler de btn1
current target: btn1
evt target   : btn1

```

### Solution 2

```

click handler de div1
current target: div1
evt target   : btn1

```

### Solution 3

```

click handler de btn1
current target: btn1
evt target   : btn1
click handler de div1
current target: div1
evt target   : btn1

```

### Solution 4

```

click handler de div1
current target: div1
evt target   : btn1
click handler de btn1
current target: btn1
evt target   : btn1

```

- Les événements se propagent selon un flux bien précis qui se décompose en 3 phases
  - 1 phase 1 (capture) : l'événement se propage du nœud document (inclus) au nœud cible (target) exclu
  - 2 phase 2 (cible *target*) : l'événement atteint le nœud cible
  - 3 phase 3 (bouillonnement *bubbling*) : événement se propage du nœud cible au nœud document inclus

Par défaut les fonctions de gestion des événements sont exécutées dans les phases 2 et 3

[Voir le code](#)

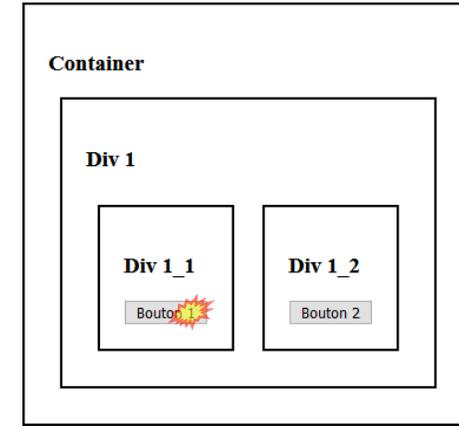
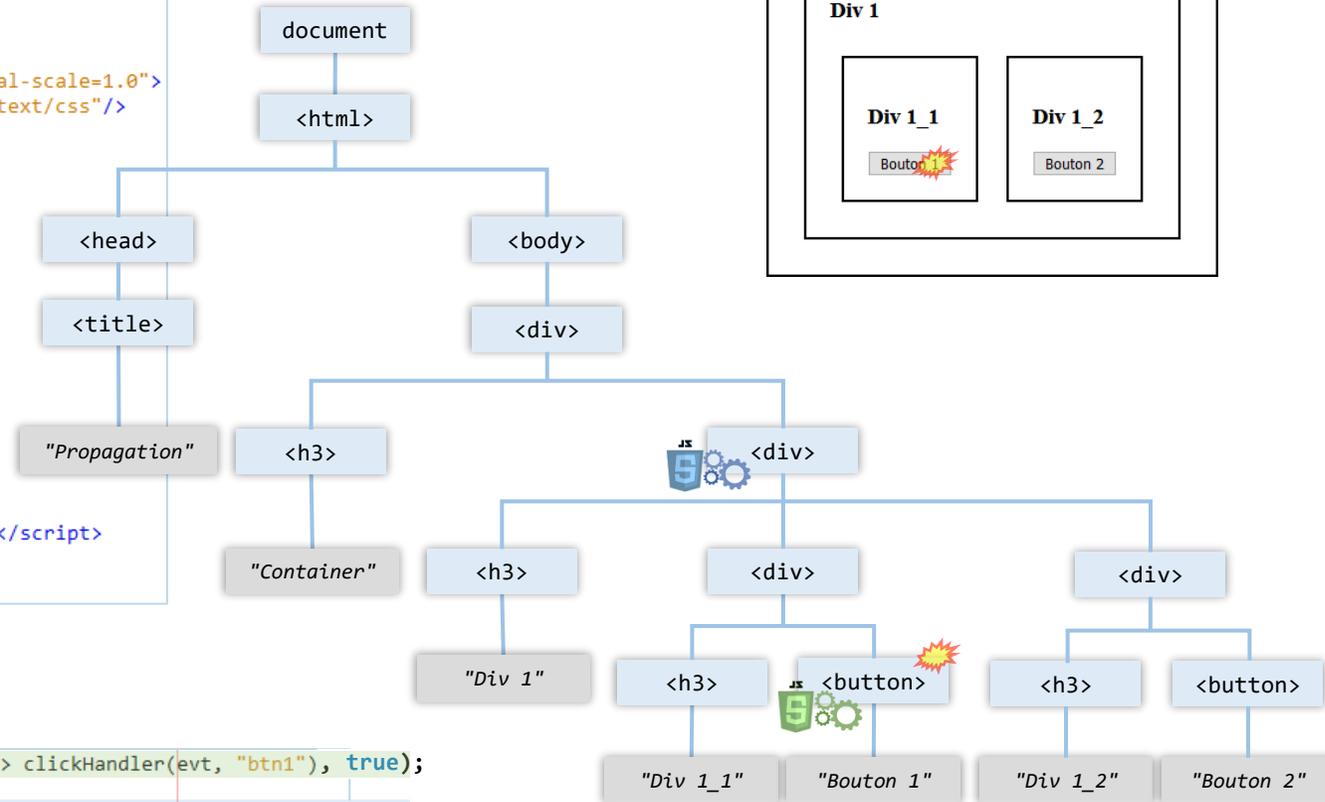
Que se passe-t-il lorsque l'utilisateur clique sur Bouton1 ?

- principes généraux

```

<html>
  <head>
    <title>Propagation</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="testpropagation.css" rel="stylesheet" type="text/css"/>
  </head>
  <body id="body">
    <div id="container">
      <h3>Container</h3>
      <div id="div1">
        <h3>Div 1</h3>
        <div id="div1_1">
          <h3>Div 1_1</h3>
          <button id="btn1">Bouton 1</button>
        </div>
        <div id="div1_2">
          <h3>Div 1_2</h3>
          <button id="btn2">Bouton 2</button>
        </div>
      </div>
    </div>
    <script src="testpropagation.js" type="text/javascript"></script>
  </body>
</html>

```



```

document.querySelector("#btn1").addEventListener("click", (evt) => clickHandler(evt, "btn1"), true);
document.querySelector("#div1").addEventListener("click", (evt) => clickHandler(evt, "div1"), true);

function clickHandler(evt, eltId) {
  console.log(`click handler de ${eltId}`);
  console.log(`  current target: ${evt.currentTarget.id}`);
  console.log(`  evt target    : ${evt.target.id}`);
}

```

**Solution 1**  
 click handler de btn1  
 current target: btn1  
 evt target : btn1

**Solution 2**  
 click handler de div1  
 current target: div1  
 evt target : btn1

**Solution 3**  
 click handler de btn1  
 current target: btn1  
 evt target : btn1  
 click handler de div1  
 current target: div1  
 evt target : btn1

**Solution 4**  
 click handler de div1  
 current target: div1  
 evt target : btn1  
 click handler de btn1  
 current target: btn1  
 evt target : btn1

**Solution 5**  
 Aucune des solutions proposées

[Voir le code](#)

## • principes généraux

```

<html>
  <head>
    <title>Propagation</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="testpropagation.css" rel="stylesheet" type="text/css"/>
  </head>
  <body id="body">
    <div id="container">
      <h3>Container</h3>
      <div id="div1">
        <h3>Div 1</h3>
        <div id="div1_1">
          <h3>Div 1_1</h3>
          <button id="btn1">Bouton 1</button>
        </div>
        <div id="div1_2">
          <h3>Div 1_2</h3>
          <button id="btn2">Bouton 2</button>
        </div>
      </div>
    </div>
    <script src="testpropagation.js" type="text/javascript"></script>
  </body>
</html>

```

```

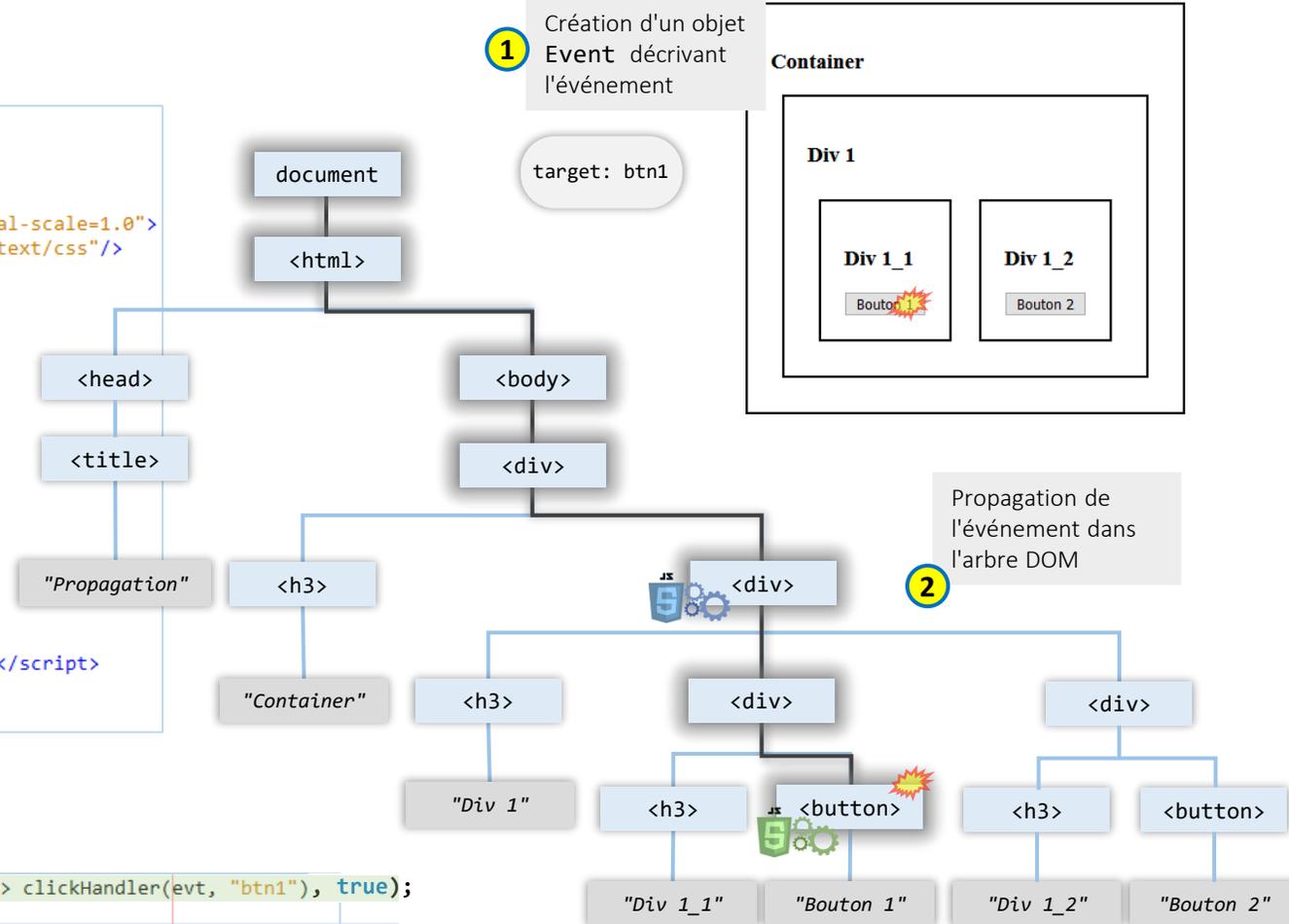
document.querySelector("#btn1").addEventListener("click", (evt) => clickHandler(evt, "btn1"), true);
document.querySelector("#div1").addEventListener("click", (evt) => clickHandler(evt, "div1"), true);

function clickHandler(evt, eltId) {
  console.log(`click handler de ${eltId}`);
  console.log(`  current target: ${evt.currentTarget.id}`);
  console.log(`  evt target   : ${evt.target.id}`);
}

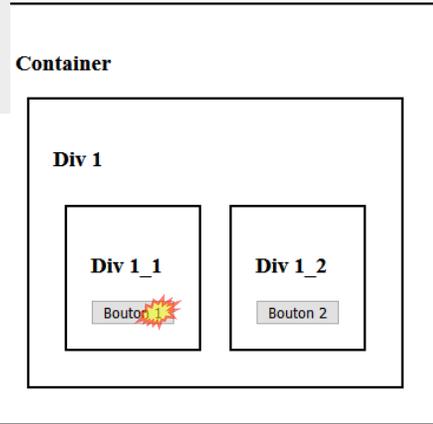
```

[Voir le code](#)

Que se passe-t-il lorsque l'utilisateur clique sur Bouton1 ?



1 Création d'un objet Event décrivant l'événement  
target: btn1



2 Propagation de l'événement dans l'arbre DOM

- 1 phase 1 (capture) (descente dans l'arbre DOM jusqu'à la cible)
- 2 phase 2 (cible target)
- 3 phase 3 (bouillonnement *bubbling*) (remontée dans l'arbre DOM)

### Solution 1

```

click handler de btn1
current target: btn1
evt target   : btn1

```

### Solution 2

```

click handler de div1
current target: div1
evt target   : btn1

```

### Solution 3

```

click handler de btn1
current target: btn1
evt target   : btn1
click handler de div1
current target: div1
evt target   : btn1

```

### Solution 4

```

click handler de div1
current target: div1
evt target   : btn1
click handler de btn1
current target: btn1
evt target   : btn1

```

### Solution 5

Aucune des solutions proposées

## • principes généraux

```

<html>
  <head>
    <title>Propagation</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="testpropagation.css" rel="stylesheet" type="text/css"/>
  </head>
  <body id="body">
    <div id="container">
      <h3>Container</h3>
      <div id="div1">
        <h3>Div 1</h3>
        <div id="div1_1">
          <h3>Div 1_1</h3>
          <button id="btn1">Bouton 1</button>
        </div>
        <div id="div1_2">
          <h3>Div 1_2</h3>
          <button id="btn2">Bouton 2</button>
        </div>
      </div>
    </div>
    <script src="testpropagation.js" type="text/javascript"></script>
  </body>
</html>

```

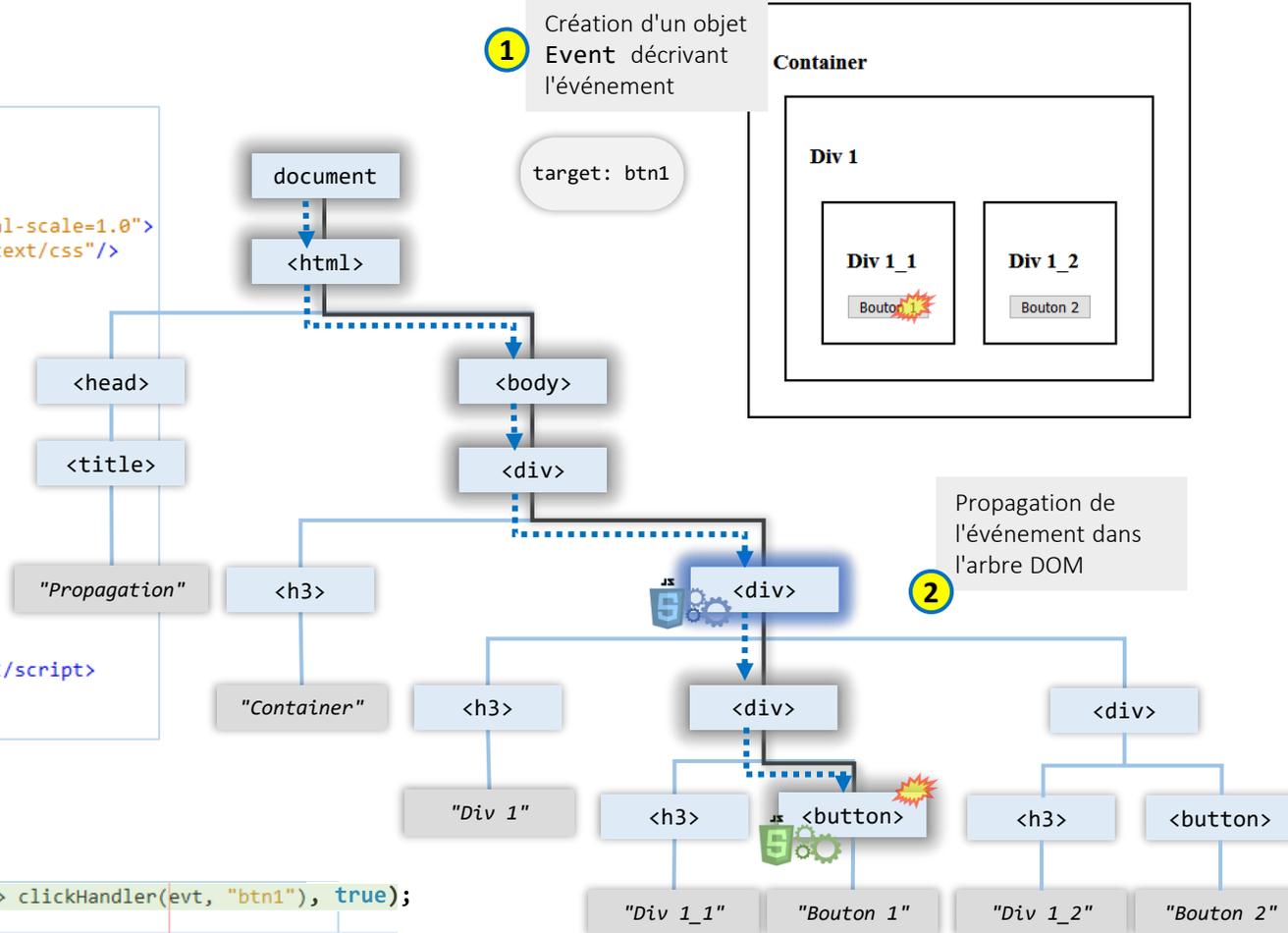
```

document.querySelector("#btn1").addEventListener("click", (evt) => clickHandler(evt, "btn1"), true);
document.querySelector("#div1").addEventListener("click", (evt) => clickHandler(evt, "div1"), true);

function clickHandler(evt, eltId) {
  console.log(`click handler de ${eltId}`);
  console.log(`  current target: ${evt.currentTarget.id}`);
  console.log(`  evt target   : ${evt.target.id}`);
}

```

exécuter dans la phase capture



1 phase 1 (capture) (descente dans l'arbre DOM jusqu'à la cible)

Que se passe-t-il lorsque l'utilisateur clique sur Bouton1 ?

- Solution 1**  
click handler de btn1  
current target: btn1  
evt target : btn1
- Solution 2**  
click handler de div1  
current target: div1  
evt target : btn1
- Solution 3**  
click handler de btn1  
current target: btn1  
evt target : btn1  
click handler de div1  
current target: div1  
evt target : btn1

**Solution 4**

```

click handler de div1
current target: div1
evt target : btn1
click handler de btn1
current target: btn1
evt target : btn1

```

**Solution 5**  
Aucune des solutions proposées

[Voir le code](#)

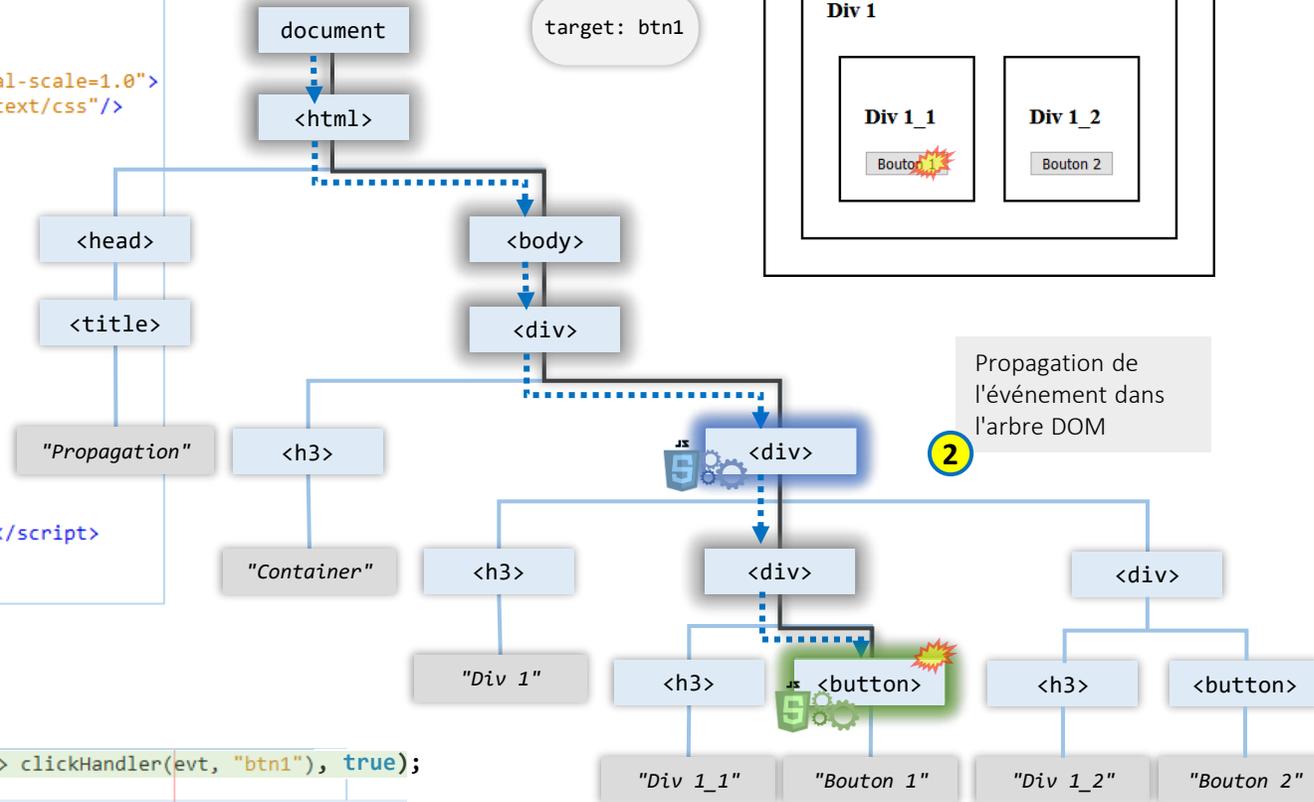
Que se passe-t-il lorsque l'utilisateur clique sur Bouton1 ?

## • principes généraux

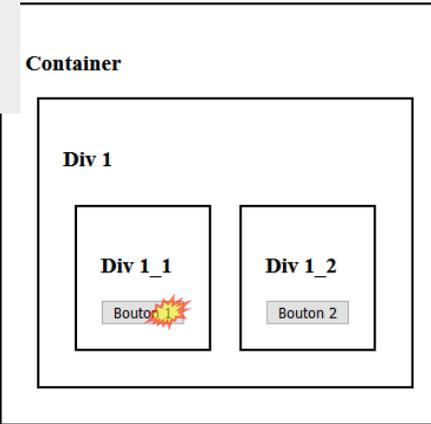
```

<html>
  <head>
    <title>Propagation</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="testpropagation.css" rel="stylesheet" type="text/css"/>
  </head>
  <body id="body">
    <div id="container">
      <h3>Container</h3>
      <div id="div1">
        <h3>Div 1</h3>
        <div id="div1_1">
          <h3>Div 1_1</h3>
          <button id="btn1">Bouton 1</button>
        </div>
        <div id="div1_2">
          <h3>Div 1_2</h3>
          <button id="btn2">Bouton 2</button>
        </div>
      </div>
    </div>
    <script src="testpropagation.js" type="text/javascript"></script>
  </body>
</html>

```



1 Création d'un objet Event décrivant l'événement  
target: btn1



2 Propagation de l'événement dans l'arbre DOM

```

document.querySelector("#btn1").addEventListener("click", (evt) => clickHandler(evt, "btn1"), true);
document.querySelector("#div1").addEventListener("click", (evt) => clickHandler(evt, "div1"), true);

function clickHandler(evt, eltId) {
  console.log(`click handler de ${eltId}`);
  console.log(`  current target: ${evt.currentTarget.id}`);
  console.log(`  evt target    : ${evt.target.id}`);
}

```

exécuter dans la phase capture

- 1 phase 1 (capture) (descente dans l'arbre DOM jusqu'à la cible)
- 2 phase 2 (cible target)

### Solution 1

```

click handler de btn1
current target: btn1
evt target    : btn1

```

### Solution 2

```

click handler de div1
current target: div1
evt target    : btn1

```

### Solution 3

```

click handler de btn1
current target: btn1
evt target    : btn1
click handler de div1
current target: div1
evt target    : btn1

```

### Solution 4

```

click handler de div1
current target: div1
evt target    : btn1
click handler de btn1
current target: btn1
evt target    : btn1

```

### Solution 5

Aucune des solutions proposées

[Voir le code](#)

- principes généraux

```

<html>
  <head>
    <title>Propagation</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="testpropagation.css" rel="stylesheet" type="text/css"/>
  </head>
  <body id="body">
    <div id="container">
      <h3>Container</h3>
      <div id="div1">
        <h3>Div 1</h3>
        <div id="div1_1">
          <h3>Div 1_1</h3>
          <button id="btn1">Bouton 1</button>
        </div>
        <div id="div1_2">
          <h3>Div 1_2</h3>
          <button id="btn2">Bouton 2</button>
        </div>
      </div>
    </div>
    <script src="testpropagation.js" type="text/javascript"></script>
  </body>
</html>
  
```

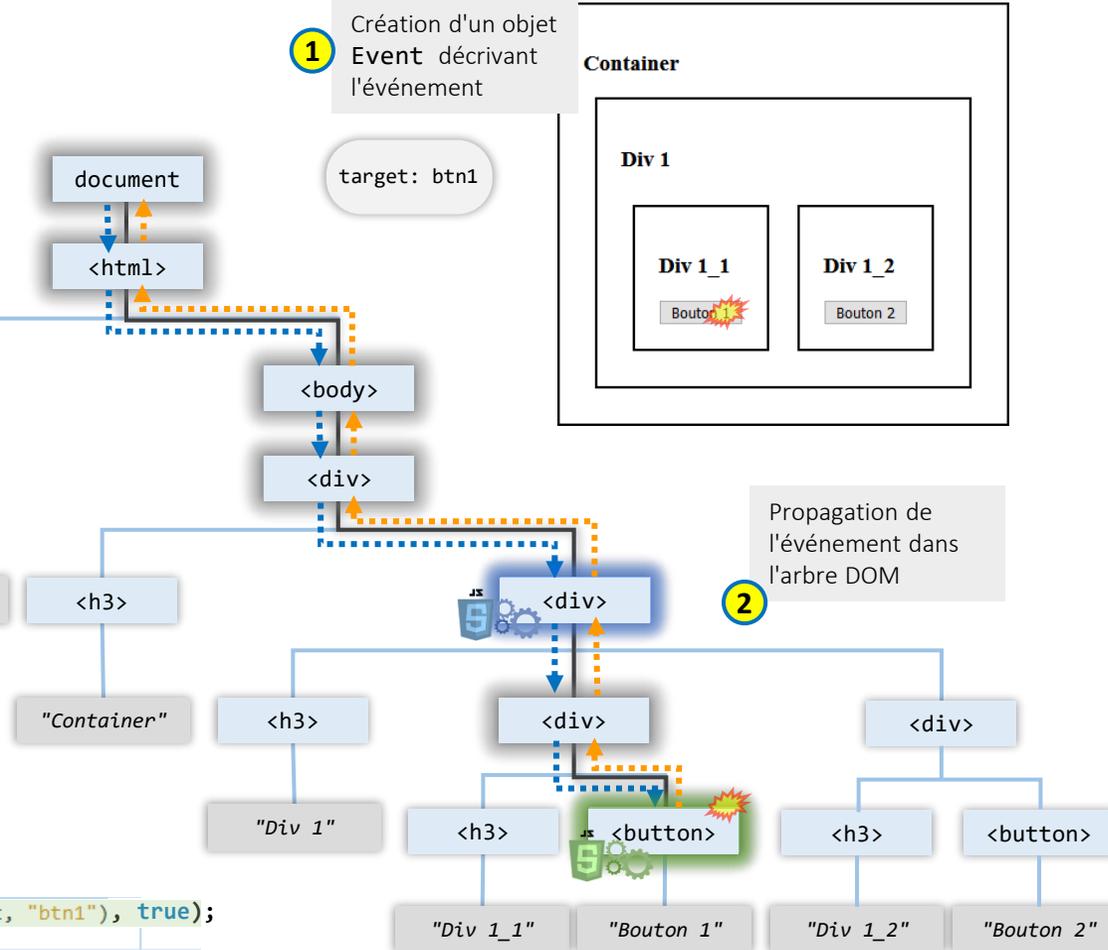
```

document.querySelector("#btn1").addEventListener("click", (evt) => clickHandler(evt, "btn1"), true);
document.querySelector("#div1").addEventListener("click", (evt) => clickHandler(evt, "div1"), true);

function clickHandler(evt, eltId) {
  console.log(`click handler de ${eltId}`);
  console.log(`  current target: ${evt.currentTarget.id}`);
  console.log(`  evt target    : ${evt.target.id}`);
}
  
```

exécuter dans la phase capture

Que se passe-t-il lorsque l'utilisateur clique sur Bouton1 ?



1 Création d'un objet Event décrivant l'événement  
target: btn1

2 Propagation de l'événement dans l'arbre DOM

- Solution 1**  
click handler de btn1  
current target: btn1  
evt target : btn1
- Solution 2**  
click handler de div1  
current target: div1  
evt target : btn1
- Solution 3**  
click handler de btn1  
current target: btn1  
evt target : btn1  
click handler de div1  
current target: div1  
evt target : btn1

**Solution 4**

click handler de div1
current target: div1
evt target : btn1
click handler de btn1
current target: btn1
evt target : btn1

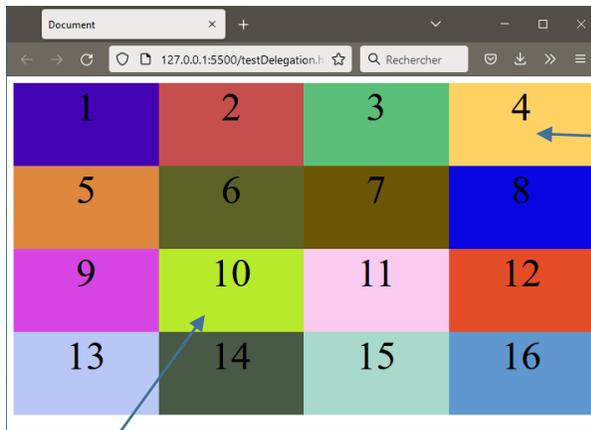
- 1 phase 1 (capture) (descente dans l'arbre DOM jusqu'à la cible)
- 2 phase 2 (cible target)
- 3 phase 3 (bouillonnement *bubbling*) (remontée dans l'arbre DOM)

**Solution 5**  
Aucune des solutions proposées

[Voir le code](#)

- la remontée des événements (phase 3 – bubbling) permet la **délégation des événements**
  - utilisé quand on veut exécuter un même code lorsque l'utilisateur interagit avec un des éléments d'un ensemble important d'éléments enfants
  - plutôt que d'associer un même gestionnaire d'événement à chacun des éléments enfants, on associe le gestionnaire à l'élément parent

[https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building\\_blocks/Events#event\\_delegation](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events#event_delegation)



chaque div à une couleur de fond tirée au hasard

```
<div id="container">
  <div class="tile">1</div>
  <div class="tile">2</div>
  <div class="tile">3</div>
  <div class="tile">4</div>
  <div class="tile">5</div>
  <div class="tile">6</div>
  <div class="tile">7</div>
  <div class="tile">8</div>
  <div class="tile">9</div>
  <div class="tile">10</div>
  <div class="tile">11</div>
  <div class="tile">12</div>
  <div class="tile">13</div>
  <div class="tile">14</div>
  <div class="tile">15</div>
  <div class="tile">16</div>
</div>
```

```
.tile {
  height: 100px;
  width: 25%;
  float: left;
  text-align: center;
  font-size: 3rem;
}

<script>
  function random(number) {
    return Math.floor(Math.random() * number);
  }

  function bgChange() {
    const rndCol = `rgb(${random(255)}, ${random(255)}, ${random(255)})`;
    return rndCol;
  }

  //initialiser les éléments avec une couleur au hasard
  for (let elt of document.querySelectorAll('.tile')) {
    elt.style.backgroundColor = bgChange();
  }
</script>
```

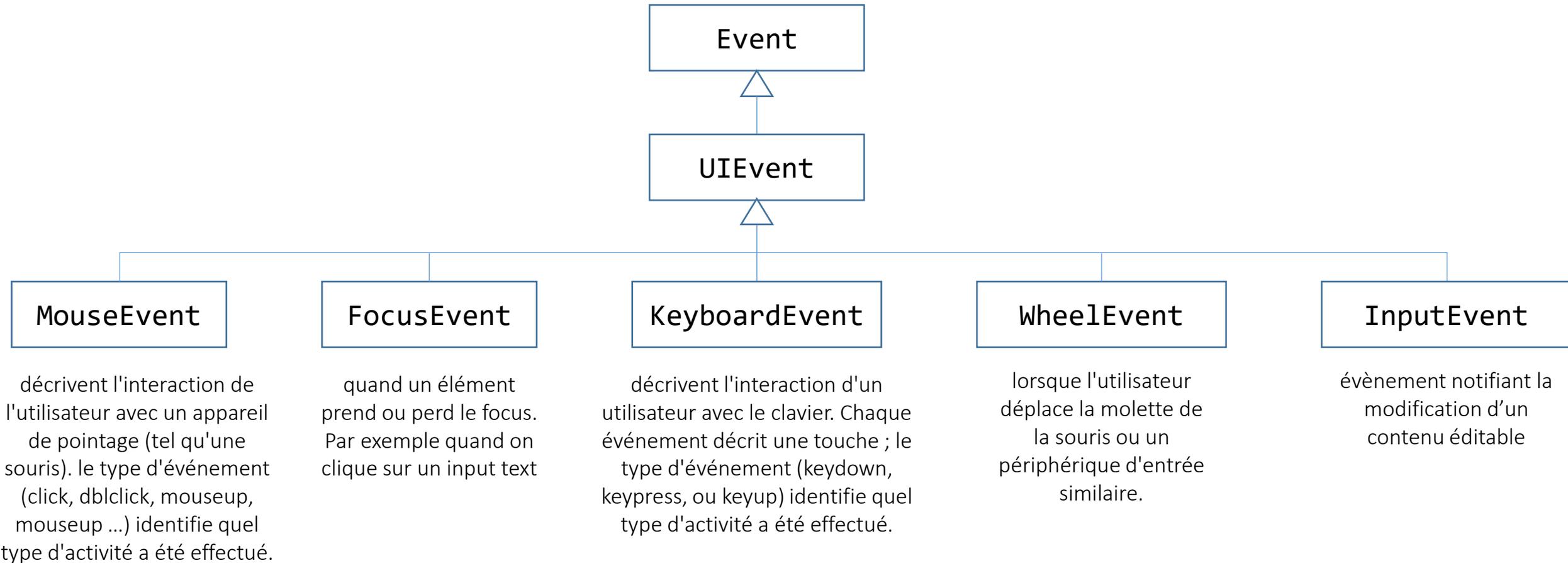
quand l'utilisateur clique sur l'une des div sa couleur change

```
const container = document.querySelector('#container');
container.addEventListener('click', (event) => event.target.style.backgroundColor = bgChange());
```

la gestion des événements click sur les div est déléguée au container

- objet **Event** décrit un événement (créé au déclenchement de l'événement)
- attributs
  - données spécifiques au type d'événement (ex touche pressée lors d'un événement `keydown`)
  - données communes à tous les types d'événements
    - **type** : le type de l'événement ("`load`", "`focus`", "`click`" ...)
    - **target** : nœud de l'arbre DOM cible de l'événement (par exemple l'élément le plus profond de l'arbre au-dessus duquel se trouve la souris pour un événement de souris)
    - **currentTarget** : nœud sur lequel l'événement se trouve actuellement lors de la propagation dans l'arbre DOM
    - **timestamp** : nombre de millisecondes écoulées depuis initialisation du document
    - ...
- méthodes
  - **stopPropagation()** : permet d'arrêter la propagation de l'événement dans l'arbre DOM
  - **preventDefault()** : Pour les types d'événements qui l'autorisent, permet d'annuler l'action implicite correspondante (exemples : envoi d'un formulaire après un **submit** , affichage du menu contextuel du navigateur après événement **contextmenu**(clic droit de la souris), ...)

- Les événements associés aux interactions avec la page Web sont des objets de classes dérivées (sous classes) de **Event**



- chaque type d'événement possède des propriétés qui lui sont spécifiques.

- exemple MouseEvent

chiffre représentant le bouton qui est pressé lorsque l'événement est lancé.

coordonnées x et y de la souris dans la fenêtre du document

renvoie `true` si la touche `ctrl` est pressée lorsque l'événement est lancé

...

