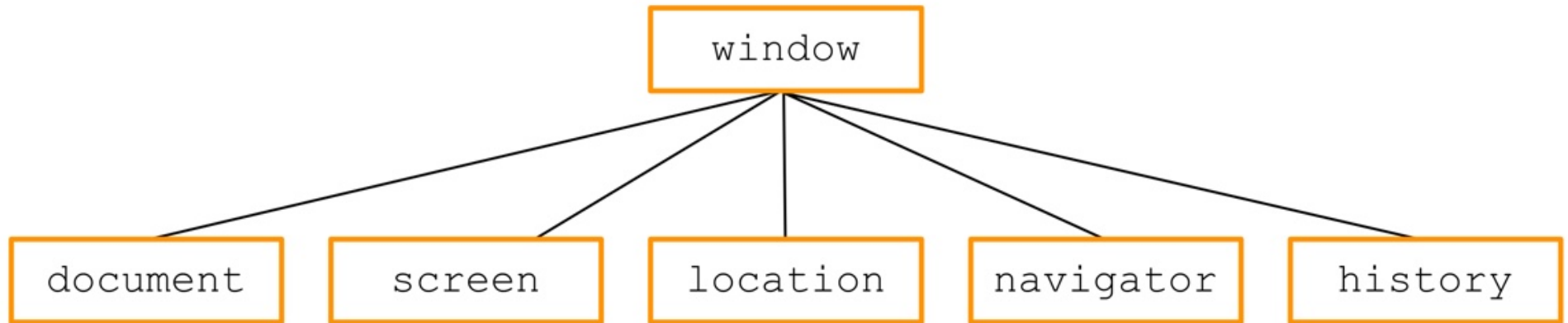


BROWSER OBJECT MODEL

- Das **Browser Object Model** (BOM) stellt Objekte zum Zugriff auf Eigenschaften des Browsers zur Verfügung
- Im Gegensatz zum DOM gibt es zum BOM keinen eigenen Standard
 - die Beschreibung der einzelnen Objekte ist über existierende Standards verteilt

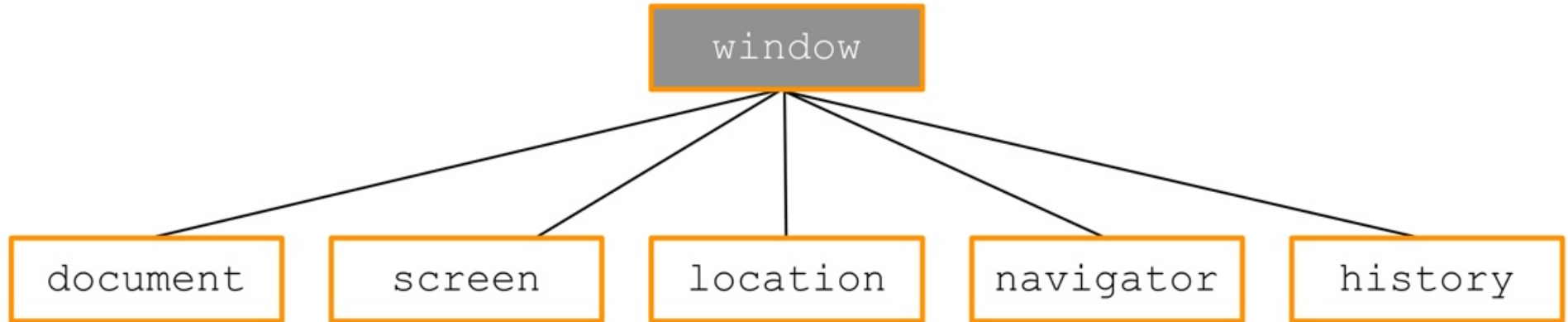
BOM: OBJEKTE



Spezifikationen der Objekte:

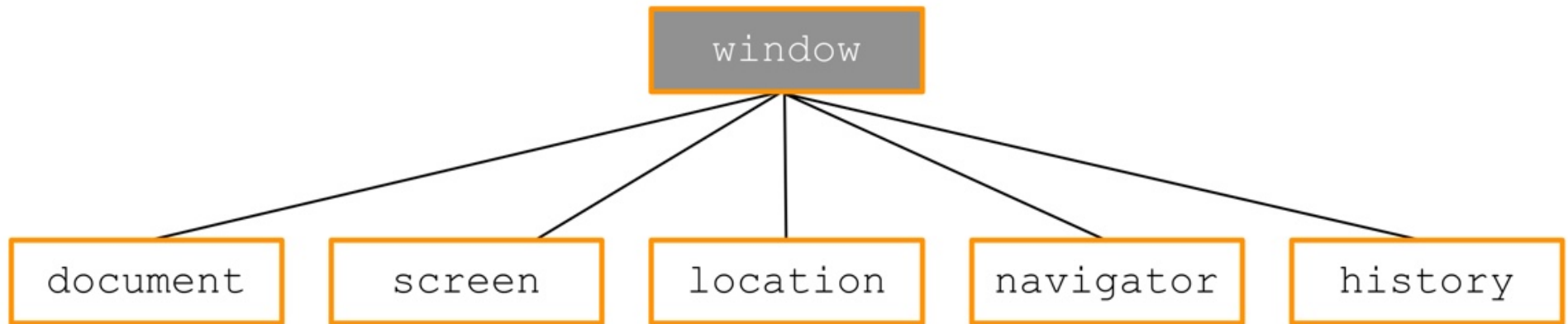
- window, location, navigator, history → [HTML-Standard](#)
- document → [DOM-Standard](#)
- screen → [CSSOM View Module](#)

BOM: `window`



- Das `window`-Objekt repräsentiert das Browser-Fenster
- `window` ist das globale (d.h. "oberste") Objekt im Browser - hier landen z.B. auch in JavaScript deklarierte globale Variablen
- Eigenschaften und Methoden des `window`-Objekts können direkt (d.h. ohne das Präfix `window.`) aufgerufen werden
- Die oben dargestellten Objekte (`screen`, `history`, etc.) sind Eigenschaften von `window`

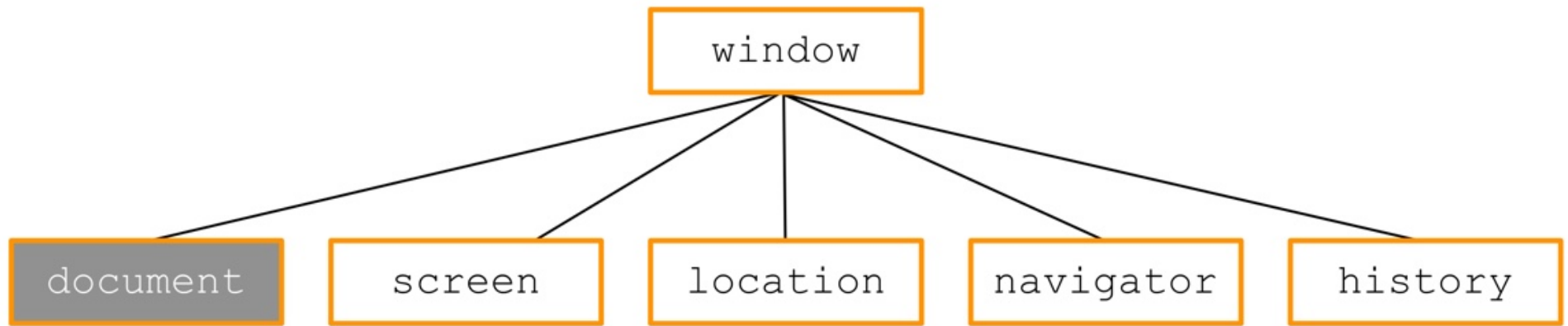
BOM: window (2)



Beispiele: Methoden von window

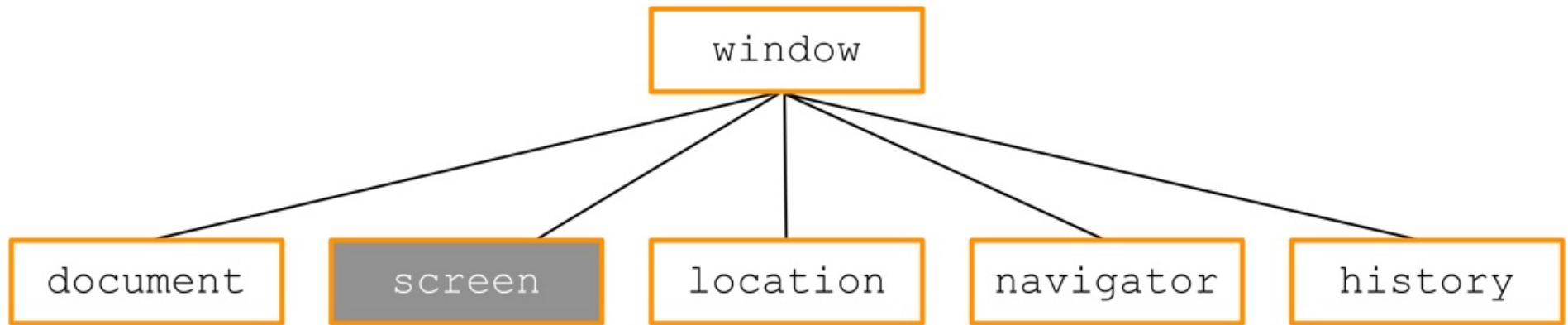
Methode	Bedeutung
<code>alert(message)</code>	Zeigt einen modalen Dialog mit der optionalen Nachricht <code>message</code>
<code>setInterval(function, delay)</code>	Ruft die Funktion <code>function</code> alle <code>delay</code> Millisekunden auf.
<code>open(file)</code>	Öffnet die Datei <code>file</code> in einem neuen Browser-Tab.

BOM: document



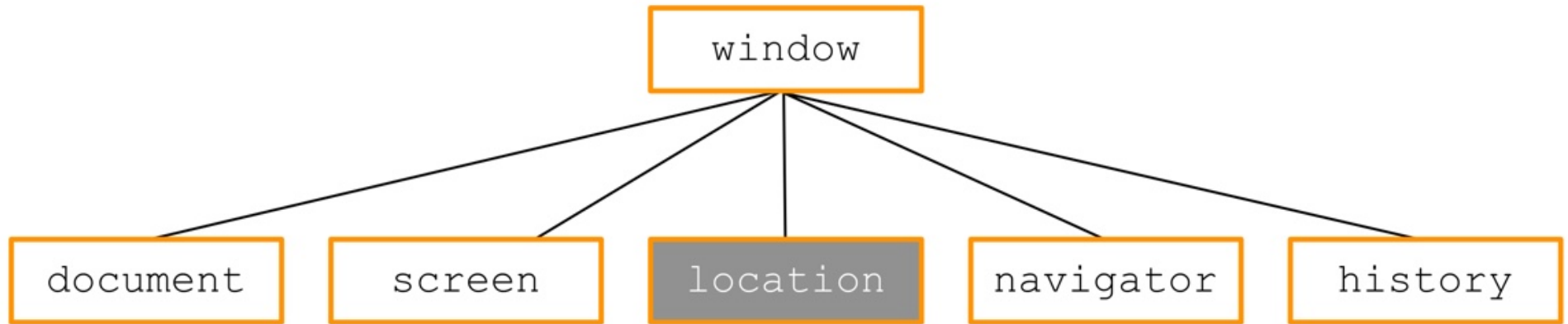
Das document-Objekt ermöglicht den Zugriff auf das DOM

BOM: screen



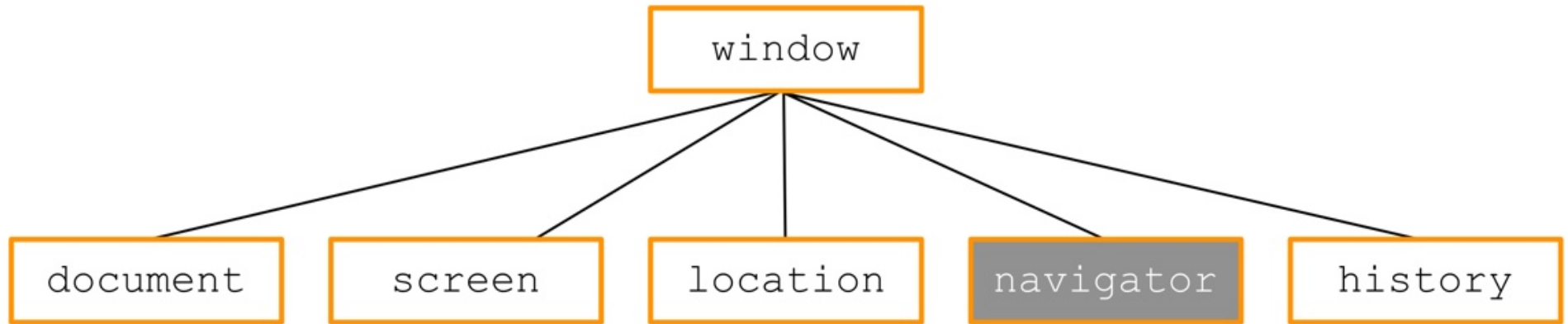
- Das screen-Objekt liefert Informationen zum Bildschirm
- Beispiele: `screen.width/screen.height` (Breite/Höhe des Bildschirms in Pixeln)

BOM: `location`



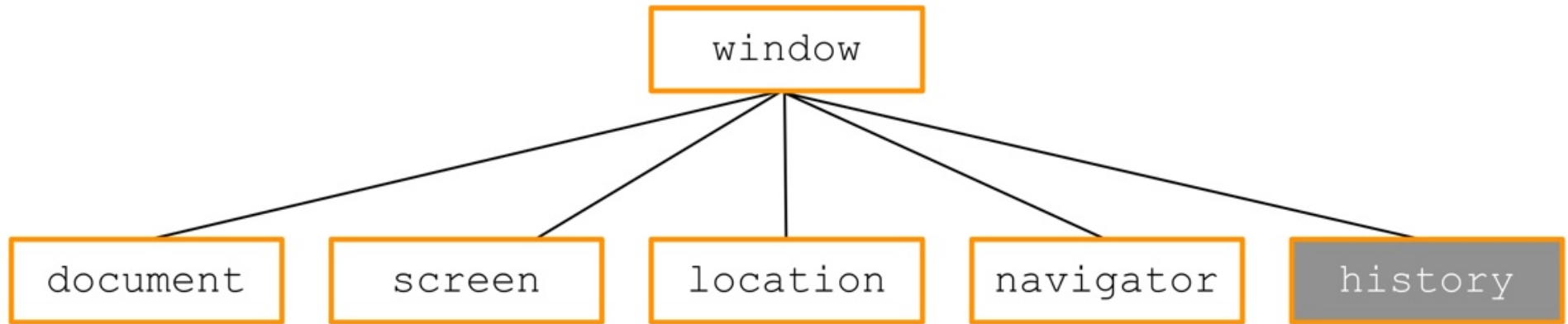
- Das `location`-Objekt ermöglicht Zugriff auf die `URI` des `window`-Objekts (meist die URL der Webseite) und deren Bestandteile
- Beispiele: `location.host`, `location.protocol`, `location.pathname`

BOM: navigator



- Das `navigator`-Objekt enthält Informationen zum Browser des Benutzers (allerdings oft nicht zuverlässig)
- Beispiele: `navigator.userAgent` (Name des Browsers), `navigator.language` (im Browser eingestellte Sprache)

BOM: history



- Das `history`-Objekt erlaubt eine (eingeschränkte) Navigation im Verlauf (besuchte Webseiten) des Benutzers
- Beispiele:

Methode	Bedeutung
<code>history.back()</code>	Lädt die vorherige Webseite im Verlauf (wie "Zurück"-Button im Browser)
<code>history.forward()</code>	Lädt die nächste Webseite im Verlauf (wie "Vorwärts"-Button im Browser)



DOM-EVENTS








Ereignisse mit JavaScript behandeln

DOM-EVENTS

- Der Browser erzeugt **Ereignisse** (*Events*) beim Auftreten bestimmter Aktionen, z.B.
 - Der Benutzer klickt etwas mit der Maus an
 - Der Benutzer drückt eine Taste auf der Tastatur
 - Der Wert in einem Formularfeld wird geändert
 - Die Webseite wurde komplett geladen
- Mittels JavaScript ist es möglich, auf solche Events zu reagieren

EVENT-TYPEN

Es existiert eine Vielzahl von **Event-Typen**, die in unterschiedlichen Standards spezifiziert sind, z.B.:

Event	Bedeutung	Spezifikation
blur	Ein Element hat den Fokus verloren	DOM Level 3 
change	Der Wert eines Eingabeelements hat sich geändert	DOM Level 2  , HTML5 
click	Eine Taste einer Maus (oder anderes "Zeigegerät") wurde gedrückt und losgelassen	DOM Level 3 
keydown	Eine Taste auf der Tastatur wurde gedrückt	DOM Level 3 
load/unload	Ein Dokument wurde vollständig geladen bzw. wird gerade verlassen	DOM Level 3 
mouseover/ mouseout	Die Maus (oder anderes "Zeigegerät") wird über ein Element bewegt bzw. davon fortbewegt	DOM Level 3 

! Mehr → [Event Reference auf MDN](#) 

EVENT-HANDLER

- Objekte bzw. Elemente, an denen ein Event auftritt, heißen **Event-Ziele** (*event targets*)
- Beispiel: Der Event-Typ `click` tritt an einem `button`-HTML-Element (=Event-Ziel) auf
- Um auf ein Event zu reagieren, wird für einen Event-Typ an einem Event-Ziel eine JavaScript-Funktion (*Event-Handler*) registriert, die beim Auftreten des Events durch den Browser aufgerufen wird
- Es gibt drei Möglichkeiten zur Registrierung von Event-Handlern

EVENT-HANDLER REGISTRIEREN

1. Registrierung direkt an einem HTML-Element

- HTML-Elemente, die bestimmte Events unterstützen, stellen Attribute zur Registrierung von Event-Handlern zur Verfügung
- Die Attribute haben Namen der Form `on${EVENTNAME}`, also z.B. `onclick`, `onblur`
- Als Wert wird JavaScript-Code notiert, z.B. der Aufruf einer Funktion

EVENT-HANDLER REGISTRIEREN

1. Registrierung direkt an einem HTML-Element: Beispiel

script.js

```
function showDateDialog() {  
    /* Date() gibt das aktuelle Datum inklusive  
       Uhrzeit als String zurück */  
    alert("Das Datum lautet: " + Date());  
}
```

seite.html

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Mein Titel</title>  
  <meta charset="utf-8">  
</head>  
<body>  
  <!-- Registrierung des Event-Handlers  
       für das click-Event -->  
  <button onclick="showDateDialog()">  
    Zeige Datum  
  </button>  
  <script src="script.js"></script>  
</body>  
</html>
```

Zeige Datum

EVENT-HANDLER REGISTRIEREN

1. Registrierung direkt an einem HTML-Element: Nachteile

- Vermischung von Zuständigkeiten (JavaScript in HTML eingebettet)
- Es kann nur ein Event-Handler pro HTML-Element für ein Event registriert werden

EVENT-HANDLER REGISTRIEREN

2. Registrierung als Eigenschaft des DOM-Elements per JavaScript

- Über das DOM selektierte Elemente bieten Eigenschaften zum Registrieren von Event-Handlern an
- Die Eigenschaften haben Namen der Form `on$ {EVENTNAME}`, also z.B. `onclick`, `onblur`
- Als Wert wird eine JavaScript-Funktion zugewiesen

EVENT-HANDLER REGISTRIEREN

2. Registrierung als Eigenschaft des DOM-Elements: Beispiel

script.js

```
/** Registrierung als Eigenschaft des selektierten
    DOM-Elements */
document.querySelector("#btn").onclick =
    function() {
        alert("Das Datum lautet: " + Date());
    };
```

seite.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Mein Titel</title>
    <meta charset="utf-8">
</head>
<body>
    <button id="btn">Zeige Datum</button>
    <script src="script.js"></script>
</body>
</html>
```

Zeige Datum

EVENT-HANDLER REGISTRIEREN

2. Registrierung als Eigenschaft des DOM-Elements: Vor-/Nachteile

- ➕ Zuständigkeiten sind besser getrennt (*unobtrusive JavaScript*)
- ➖ Es kann nur ein Event-Handler pro HTML-Element für ein Event registriert werden

EVENT-HANDLER REGISTRIEREN

3. Registrierung als Event-Listener per JavaScript

- Über das DOM selektierte Elemente stellen die Methode `addEventListener(name, handler)` zur Verfügung
- `name` ist der Name des Events, für das ein Event-Handler hinzugefügt werden soll, z.B. `click`, `blur` (ohne `on`-Präfix!)
- `handler` ist eine JavaScript-Funktion

EVENT-HANDLER REGISTRIEREN

3. Registrierung als Event-Listener per JavaScript: Beispiel

script.js

```
let button = document.querySelector("#btn");

// Registrierung eines Handlers für Mausklick
button.addEventListener("click", function() {
    alert("Das Datum lautet: " + Date());
});

// Registrierung eines weiteren Handlers für Mausklick
button.addEventListener("click", function() {
    button.innerHTML = "DATUM! JETZT!";
});
```

seite.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Mein Titel</title>
  <meta charset="utf-8">
</head>
<body>
  <button id="btn">Zeige Datum</button>
  <script src="script.js"></script>
</body>
</html>
```

Zeige Datum

EVENT-HANDLER REGISTRIEREN

3. Registrierung als Event-Listener per JavaScript: Vor-/Nachteile

- Zuständigkeiten sind besser getrennt (*unobtrusive JavaScript*)
- Es können mehrere Event-Handler pro HTML-Element für ein Event registriert werden

EVENT-FLUSS: BEISPIEL

script.js

```
let button = document.querySelector("#btn");
let paragraph = document.querySelector("#para");

button.addEventListener("click", function() {
    alert("button-Element wurde geklickt!");
});

paragraph.addEventListener("click", function() {
    alert("p-Element wurde geklickt!");
});
```

seite.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Mein Titel</title>
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css"
        href="style.css">
</head>
<body>
  <p id="para">
    <button id="btn">Klick</button>
  </p>
  <script src="script.js"></script>
</body>
</html>
```

style.css

```
#para {
  border: 1px solid;
  padding: 20px;
}
```



! Bei Klick auf den Button wird erst das `click`-Event des Buttons, und danach das `click`-Event des `p`-Elements ausgelöst → *bubbling*.

EVENT-FLUSS

Der Standard-Event-Fluss (*event propagation*) arbeitet in drei Phasen:

1. Capture-Phase:

- Das vom Browser erzeugte Event (z.B. nach Klick auf eine Schaltfläche) durchläuft den Objektbaum vom `window`-Objekt bis zum eigentlichen Event-Ziel
- Auf diese Weise kann ein Event z.B. abgefangen werden, bevor es sein Ziel erreicht

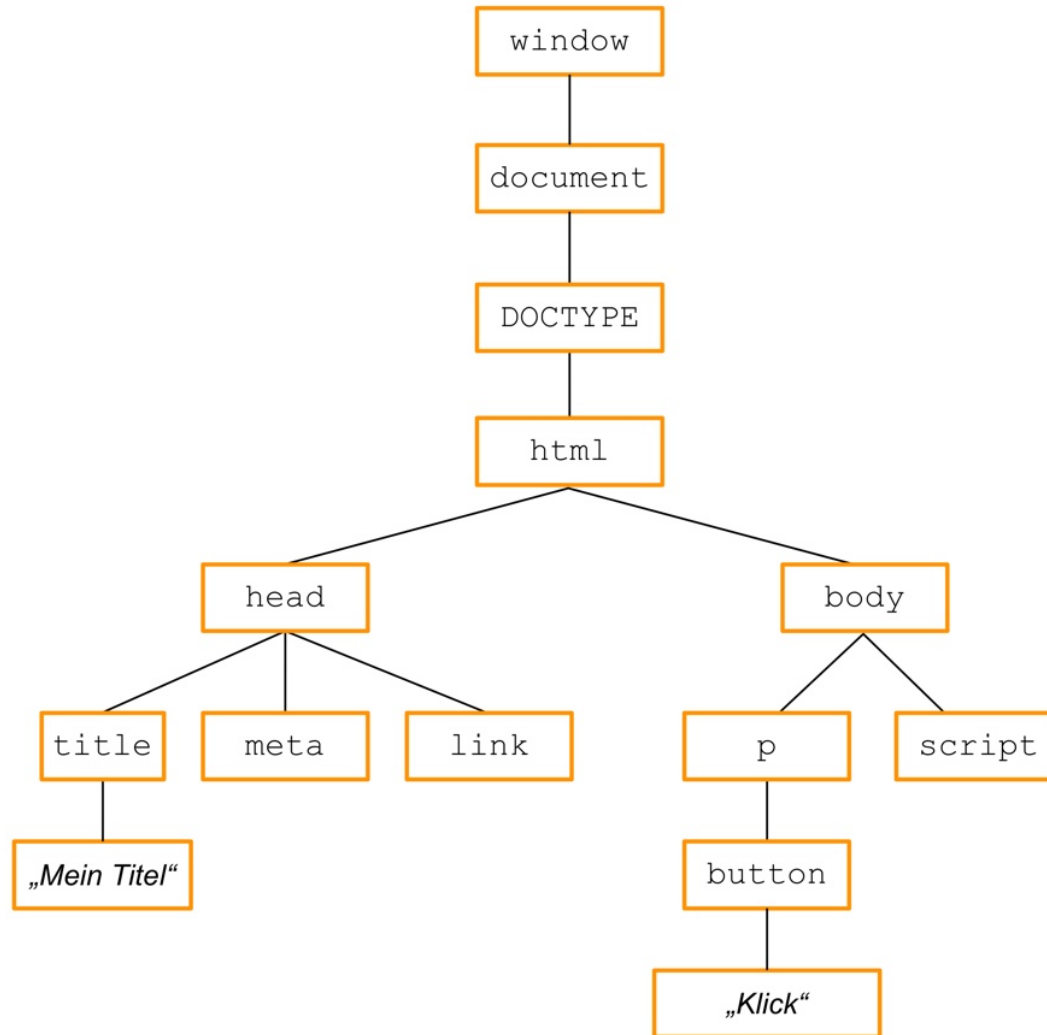
2. Target-Phase:

Die Event-Handler am Event-Ziel werden ausgelöst

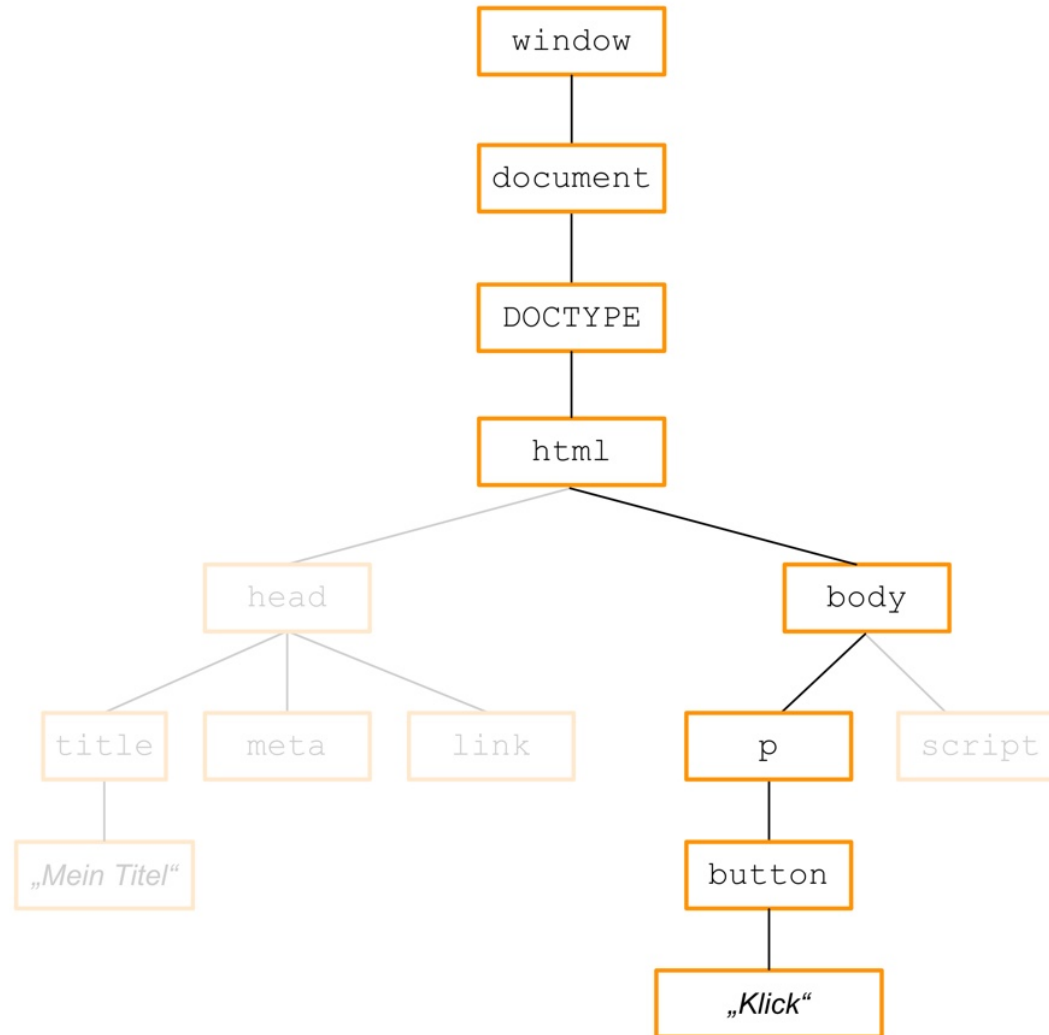
3. Bubbling-Phase:

- Das Event durchläuft den Objektbaum jetzt wieder zurück vom Event-Ziel bis zum `window`-Element
- An jedem Element werden dabei ggf. registrierte Event-Handler ausgelöst

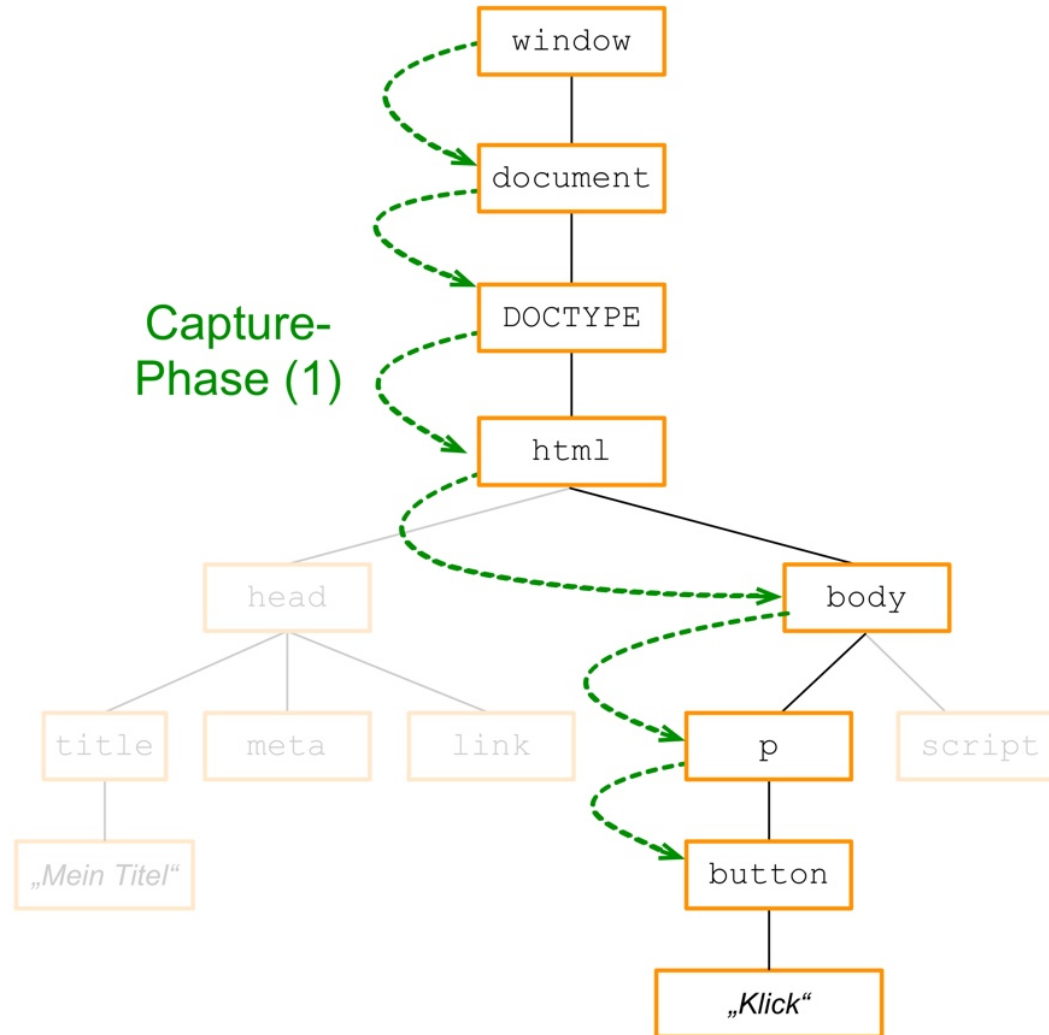
EVENT-FLUSS IM BEISPIEL



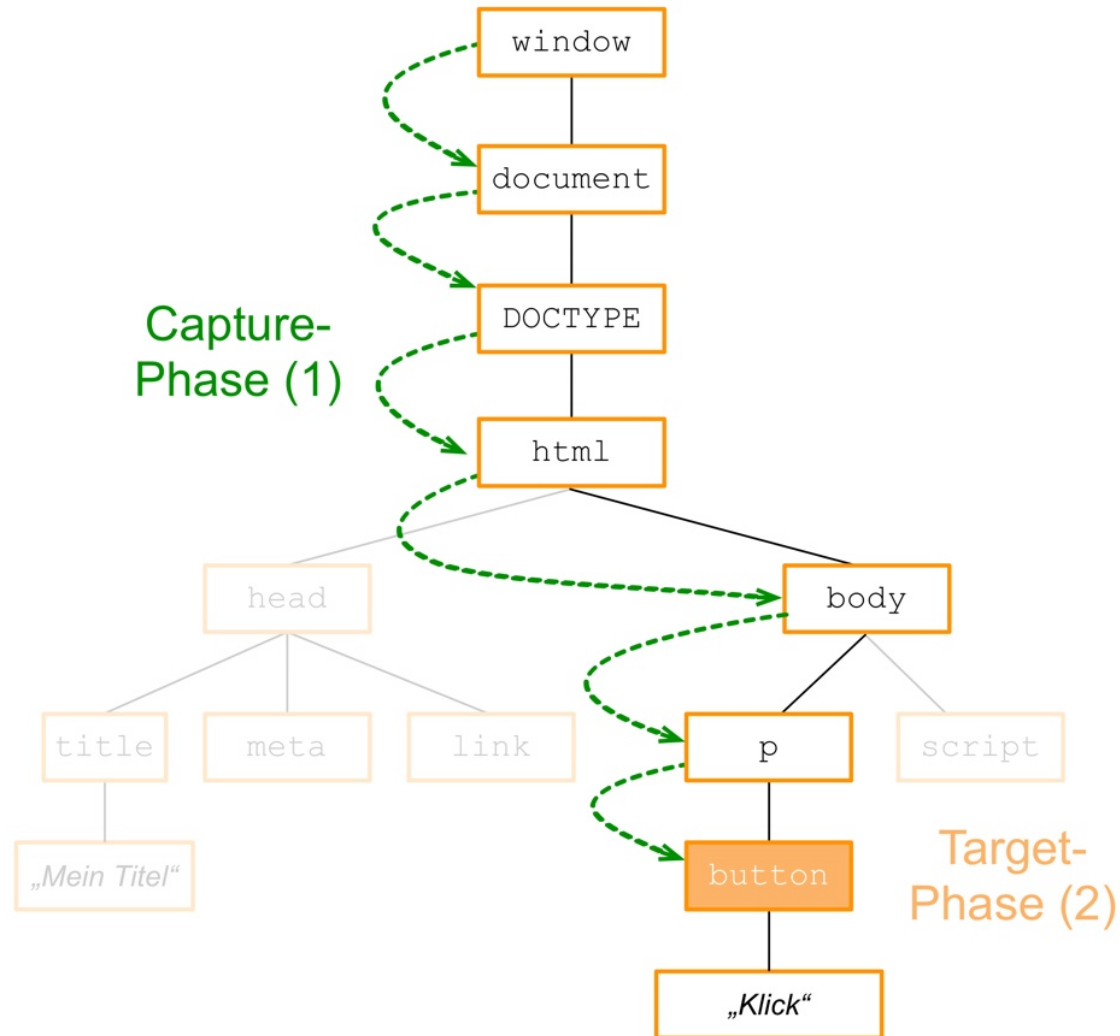
EVENT-FLUSS IM BEISPIEL



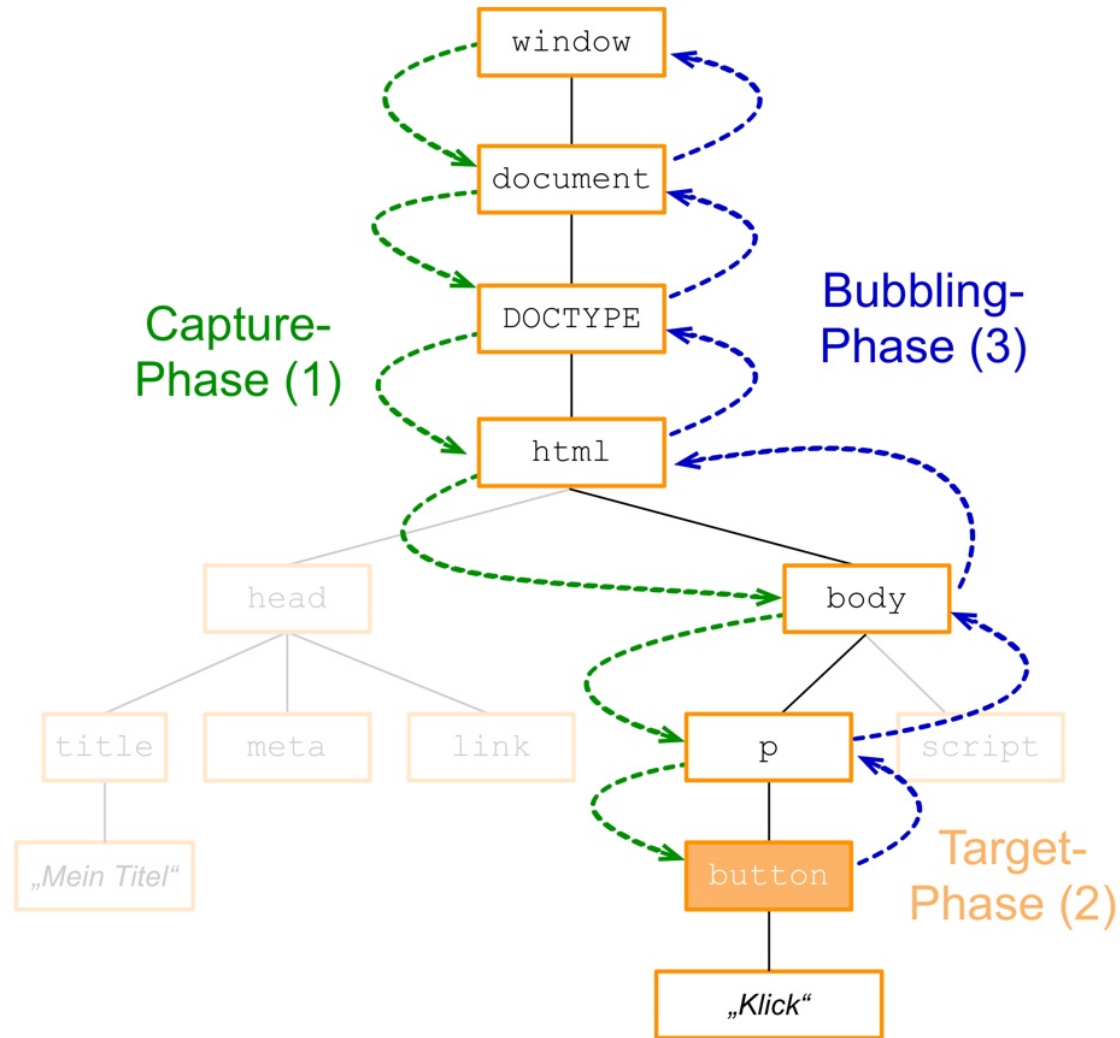
EVENT-FLUSS IM BEISPIEL



EVENT-FLUSS IM BEISPIEL



EVENT-FLUSS IM BEISPIEL



EVENT-FLUSS: BEISPIEL (2)

script.js

```
let button = document.querySelector("#btn");
let paragraph = document.querySelector("#para");

button.addEventListener("click", function() {
    alert("button-Element wurde geklickt!");
});

paragraph.addEventListener("click", function() {
    alert("p-Element wurde geklickt!");
});
```

seite.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Mein Titel</title>
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css"
        href="style.css">
</head>
<body>
  <p id="para">
    <button id="btn">Klick</button>
  </p>
  <script src="script.js"></script>
</body>
</html>
```

style.css

```
#para {
  border: 1px solid;
  padding: 20px;
}
```



! Das `click`-Event des Buttons wird in der Target-Phase ausgelöst, danach das `click`-Event des `p`-Elements in der Bubbling-Phase.

EVENT-FLUSS: HINWEISE

- Nicht alle Events steigen in der Bubbling-Phase wieder auf, z.B. `blur`, `load`
- Standardmäßig werden Event-Handler für die Bubbling-Phase registriert (bei allen drei Registrierungsvarianten)
- Eine Registrierung für die Capture-Phase kann mittels `addEventListener(name, handler, capture)` mit `capture=true` erfolgen

EVENT-FLUSS: BEISPIEL (3)

script.js

```
let button = document.querySelector("#btn");
let paragraph = document.querySelector("#para");

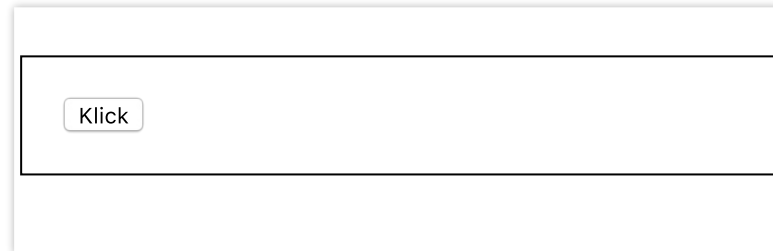
button.addEventListener("click", function() {
    alert("button-Element wurde geklickt!");
});
// Registrierung für Capture-Phase
paragraph.addEventListener("click", function() {
    alert("p-Element wurde geklickt!");
}, true);
```

seite.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Mein Titel</title>
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css"
        href="style.css">
</head>
<body>
  <p id="para">
    <button id="btn">Klick</button>
  </p>
  <script src="script.js"></script>
</body>
</html>
```

style.css

```
#para {
  border: 1px solid;
  padding: 20px;
}
```



! Erst wird das `click`-Event des `p`-Elements in der Capture-Phase ausgelöst, danach das `click`-Event des Buttons in der Target-Phase.



(VIEL!) MEHR ZU JAVASCRIPT

Buchserie: *You don't know JavaScript* von Kyle Simpson (lesbar auf [GitHub](#))