

Internetwork og Web-programmering

Klient side:

Browser som kørtidsomgivelse

Forelæsning 5
Brian Nielsen

Distributed, Embedded, Intelligent Systems



2 stil arter for web-applikationer

Klassiske "web-sider"

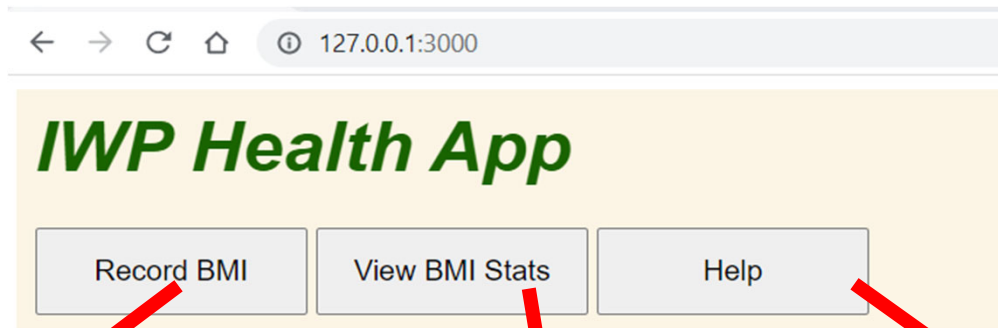
- Bruger udfylder og indsender "formular"
- Klient og server kommunikerer via HTML
- Begrænset klient-side scripting
- Ingen interaktion med server uden et click på en "submit" knap
- Server-side scripting (klassisk PHP) genererer dynamisk (beregninger og DB-opslag) en respons web-side, som nyt, helt, og selvstændigt HTML dokument
 - Flere "tunge" dokumenter transporteres til/fra server
- "Old-school" (men simpel og stadig arbejdshesten bag mange applikationer),
- God til søgemaskine optimering, bookmarks
- "Fler-sidede applikationer"

Moderne web-applikationer

- En oplevelse af at arbejde med en "rigtig" applikation
- Meget client-side scripting til bruger-interaktion
- Server interaktion foregår ofte i baggrunden via HTTP (REST) API og JSON (AJAX)
 - Flere, hurtigere kald
- Dynamisk omskrivelse på klient-side af applikationens HTML side vha DOM og events.
 - Kun den opdaterede del ændres.
- I det ekstreme "Single Page Web-application"

Kan blandes til "Hybride" varianter

Single page app (demo) – ét html document!



IWP BMI-recorder

Personal information:

Name

Height (cm):

Weight (kg):

IWP BMI-recorder

Personal information:

Name

Height (cm):

Weight (kg):

Hi Mickey! Your BMI is 58.02. Since last it has chan

IWP BMI-Statistics-tracker

Personal information:

Name

IWP BMI-Statistics-tracker

Personal information:

Name

| BMIStats for user Mickey | |
|--------------------------|-------|
| Weight | Bmi |
| 90 | 27.78 |
| 188 | 58.02 |

```
...<body> == $0
<header> IWP Health App</header>
<!-- MENU -->
<nav>...</nav>
<!-- FEATURE: BMI RECORDER -->
<section id="record_Section_id" style="display: none;">...</section>
<!-- END OF RECORD BMI-->
<!-- FEATURE: BMI STATS TRACKER -->
<section id="stats_Section_id" style="display: block;">...</section>
<!-- END OF STAT TRACKER-->
<!-- FEATURE: BMI HELP -->
<section id="help_Section_id" style="display: none;">...</section>
<!-- END OF HELP-->
<script src="js/bmi-client.js"></script>
</body>
</html>
```

Help on Body Mass Index

BMI (Body Mass Index) is a widely used and simple, method for estimating whether you are over- or under- normal weight. It can give an indication of possible health problems.

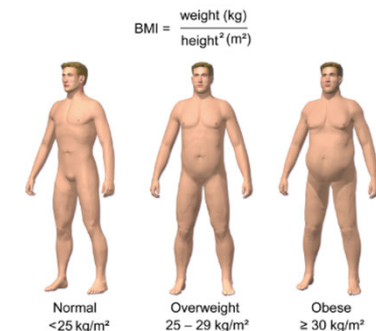


Figure 1: The figure illustrates the meaning of BMI-numbers. *Img. from wikipedia*

Background

Klient

127.0.0.1:3000

IWP Health App

Record BMI View BMI Stats Help

IWP BMI-recorder

Personal information:

Name

Height (cm):

Weight (kg):

Record

Hi Mickey! Your BMI is 58.02. Since last it has char

IWP BMI-Statistics-tracker

Personal information:

Name

Get Stats

| BMIStats for user Mickey | |
|--------------------------|-------|
| Weight | Bmi |
| 90 | 27.78 |
| 188 | 58.02 |

JAVASCRIPT bmi-client.js

Struktur af BMI-APP applikation

GET /

Front page html

POST /bmi-records

JSON

```
{"userName":"Mickey",
"height":180, "weight":188}
```

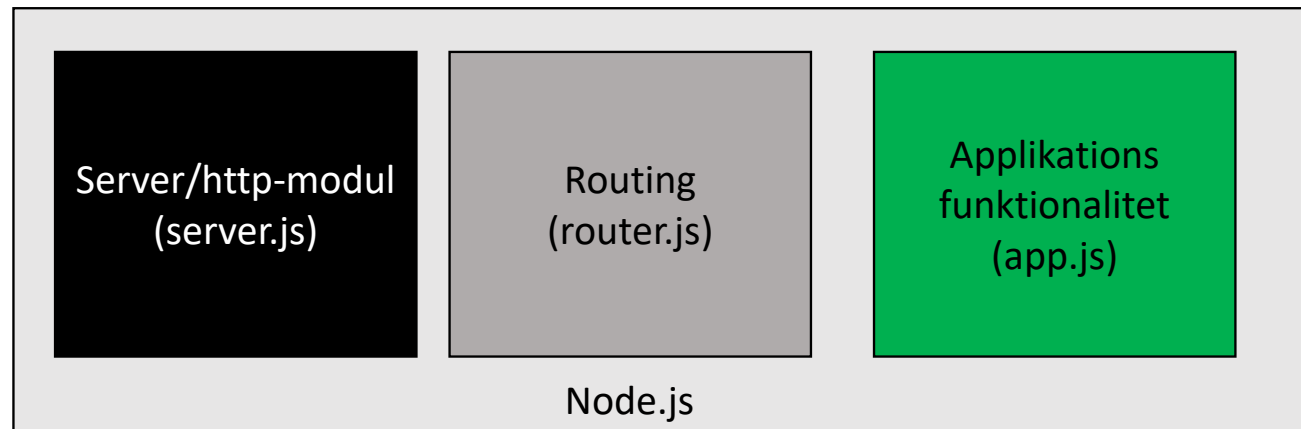
JSON

```
{"userName":"Mickey",
"bmi":58.02, "delta":8.64}
```

GET /bmi-records?userName=Mickey

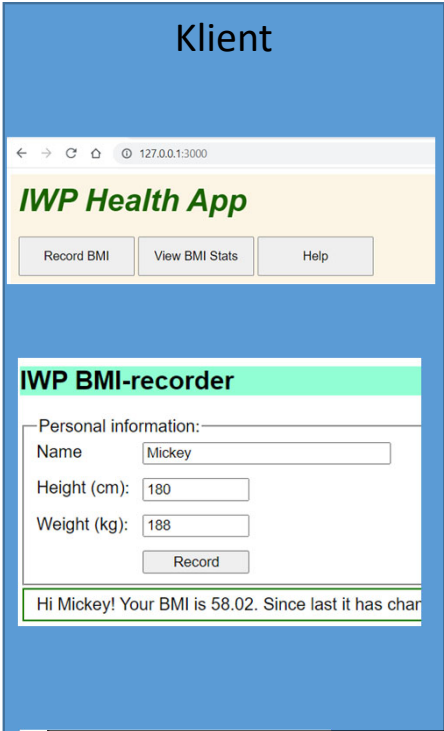
JSON

```
[{"userName":"Mickey","height":180,"weight":90,"bmi":27.78},
{"userName":"Mickey","height":180,"weight":188,"bmi":58.02}]
```



Overordnet logik:

- Browser udpeger applikation med URL: 127.0.0.1:3000 /
- Server leverer forside (bmi.html)
- Forsiden linker til bmi-client.js fil,
- Server leverer **bmi-client.js**, som fortolkes af browser
- Bruger vælger funktionalitet, den valgte del af siden vises
- Klient (Browser)
 - klient side HTML validering af formular
 - formular udlæses af JS
 - data indsendes af JS (hhv POST / GET) som JSON objekt
 - modtager svar som JSON
 - Optegner del af siden baseret på modtagne JSON objekter.



IV

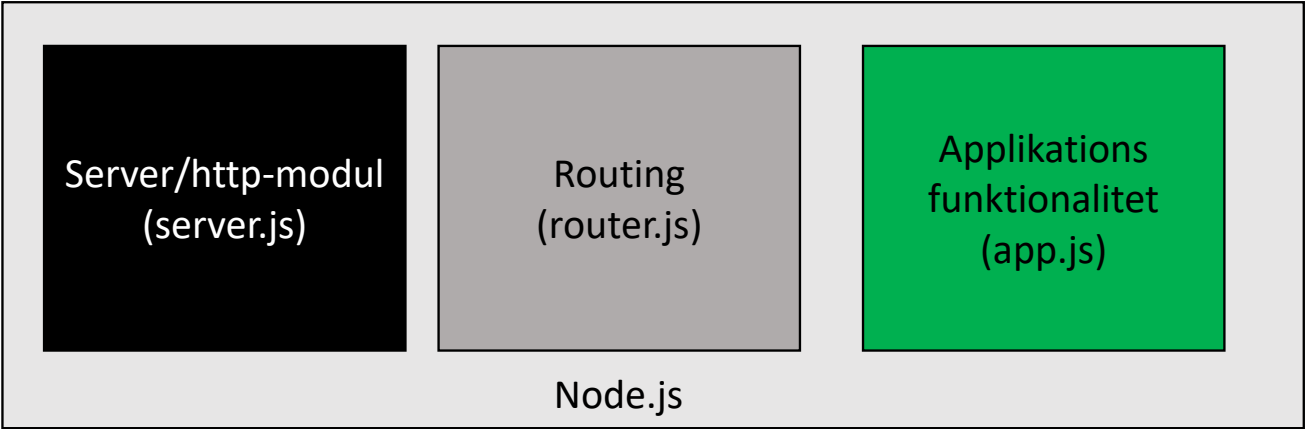
JAVASCRIPT bmi-client.js

BMI-APP "API"

GET /
Front page html

POST /bmi-records
JSON
{ "userName": "Mickey",
 "height": 180, "weight": 188 }

JSON
{ "userName": "Mickey",
 "bmi": 58.02, "delta": 8.64 }



Overordnet logik:

• Browser udlever applikation med URL: 127.0.0.1:3000 /

| Route | POST | GET | PUT | DEL |
|----------------------------|------------------------|--|-----|-----|
| /bmi-records | Tilføjer nyt BMI entry | Returnerer all records | | |
| /bmi-records?userName=name | | returnerer BMI stats for userName | | |
| /bmi-records/userName/ | | returnerer BMI stats for userName, samme som ovenfor, for eksemplets skyld | | |
| ... | | | | |

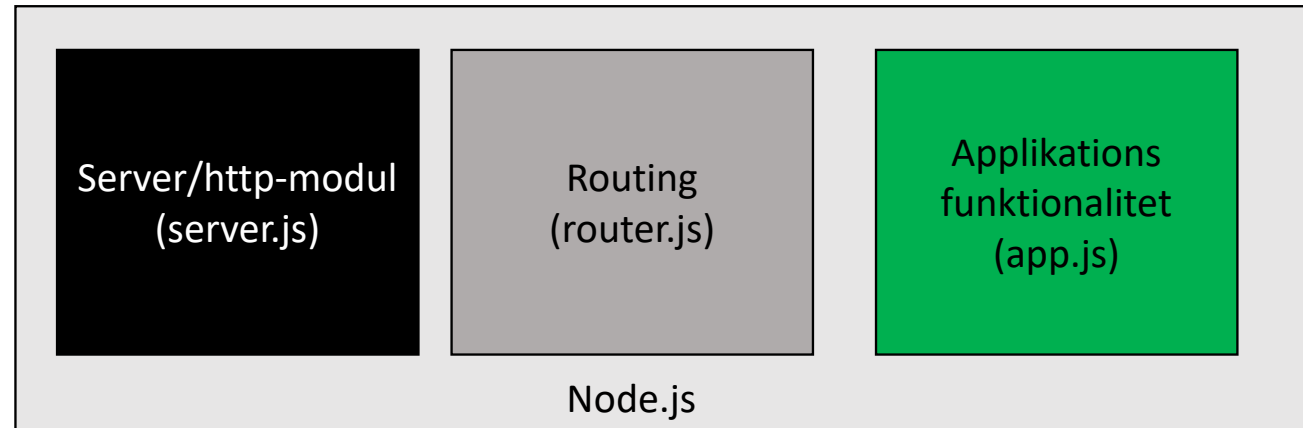
{ "userName": "Mickey", "height": 180, "weight": 188, "bmi": 58.02 }

- modtager svar som JSON
- Optegner del af siden baseret på modtagne JSON objekter.

Fil-Struktur af applikationen

Fil-struktur

```
node/  
  server.js  
  app.js  
  router.js  
  PublicResources/  
    css/  
      simple.css  
    js/  
      bmi-client.js  
  html/  
    bmi.html  
  img/  
    bmi.png ...
```



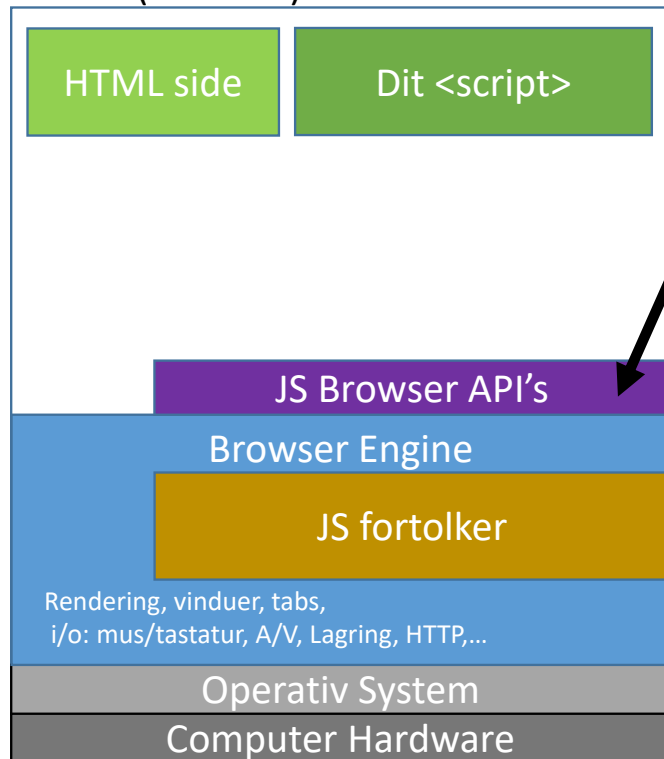
- Server har adgang til de scripts og filer den skal bruge i node kataloget.
- Klienter har kun adgang til filer i "PublicResources" (håndhæves af server-modul)
 - ondsindede brugere kunne forsøge at lave requests til filer udenfor dette område! Fx `PublicResources/../../../../../passwords.txt`
 - (credentials er self. lagret i krypteret form)

Browser som køretidsomgivelse

Overordnet arkitektur: Indbyggede biblioteker

- **API Application Programming Interface:** samling funktioner, der giver adgang til funktionalitet i eksternt program modul / service

Klient (Browser)



Klient Side

- **DOM: (Document Object Model):** Læsning og ændring af HTML/CSS dokumenter
- **Fetch:** foretager HTTP requests til server
- **BOM (Browser Object Model):** Adgang til browser information
- **2D,3D grafik:** I HTML canvas-element
- **Multi-medie:** visning af lyd/video
- **Device:** fx notifikationer
- **Client-side lagring:** Gemme data persistent på klienten

- Et script kan kun tilgå klientens ressourcer via disse kontrollerede APler ("en sandkasse")
- Scripts fra nettet kan være ondsindede.

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs

API dokumentation:

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

- Vigtige
 - DOM
 - Fetch
- Andre nyttige
 - Console
 - Storage
 - Web storage
 - WebSockets

Specifications

This is a list of all the APIs that are available.

A

Ambient Light Events

B

Background Tasks

Battery API 


Beacon

Bluetooth API

Broadcast Channel API

C

CSS Counter Styles

CSS Font Loading API 

CSSOM

Canvas API

Channel Messaging API

Console API

Credential Management API

D

DOM

E

Encoding API

Encrypted Media Extensions

F

Fetch API 

File System API 

Frame Timing API

Fullscreen API

G

Gamepad API 

Geolocation API

H

HTML Drag and Drop API

High Resolution Time

History API

I

Image Capture API


IndexedDB

Intersection Observer API

L

Long Tasks API 

M

Media Capabilities API 

Media Capture and Streams


Media Session API

Media Source Extensions 

MediaStream Recording

N

Navigation Timing

Network Information API 

P

Page Visibility API

Payment Request API

Performance API

Performance Timeline API

Permissions API

Pointer Events

Pointer Lock API

Proximity Events 

Push API 

R

Resize Observer API

Resource Timing API


S

Server Sent Events

Service Workers API

Storage

Storage Access API

Streams 

T

Touch Events

V

Vibration API

W

Web Animations 

Web Audio API

Web Authentication API

Web Crypto API

Web Notifications

Web Storage API

Web Workers API

WebGL

WebRTC

WebVR API 

WebVTT

WebXR Device API

Websockets API

REMINDER!

Hvad bruges JS til på front-end (clients)?

- Interaktivitet
 - Validering af bruger input og meningsfyldte fejlmeddelelser
 - Dynamisk fremstilling af web-side, afhængigt af brugers valg
 - Styring af GUI-elementer: sliders, menuer, pop-ups,...
 - Applikations funktionalitet
- Server-kommunikation
 - Dynamisk indlæsning af data fra server, filtreret visning
 - Opdatering af web-side uden explicit "submit" eller "reload"
 - Minimere server kommunikation: en del data behandling kan ske lokalt uden at bruge netværk og server.
- Program funktioner
 - Lettere beregninger og program dele, som klienten "bekvem" kan foretage lokalt
 - Funktioner den skal vare tage, hvis server er "nede"

Indlæsning af JavaScript i klienten

- **Indlejret JavaScript metoden:**

- koden placeres indenfor et HTML `<script>` element.
- Til simple kode for dokumentet.

```
<script>
  console.log("Hello IWP");
</script>
```

- **Ekstern JavaScript metoden:**

- Normalt anbefalet
- koden angives som en ekstern fil i HTML script elementet.
- Mindre HTML dokumenter, deling af scriptet fra flere sider
- Indeholder typisk definitioner af komplekse funktioner og data, og hele biblioteker
- Placeres i HTML-body (normalt nederst, da siden så er optegnet og DOM er klar)
- `type="module"` attribut erklærer at det er en EC6 modul: import og export statements kan bruges
- Kan også linkes i header (typisk hvis det er en generelt bibliotek)
 - Sæt evt defer-attribut

```
<script src="arrays.js"></script>
```

```
<script type="module"src="arrays.js"></script>
```

- **Inline JavaScript metoden:**

- kode placeres direkte indenfor visse HTML attributer. Typisk til event-håndteringsfunktioner for elementet.
- Der er bedre metoder (`addEventListener`)

```
<a href="JavaScript:alert('hej');">info</a>
```

```
<input type="button" value="Send"
  onclick="alert('Hej!');" >
```

Indlæsning af HTML/JavaScript

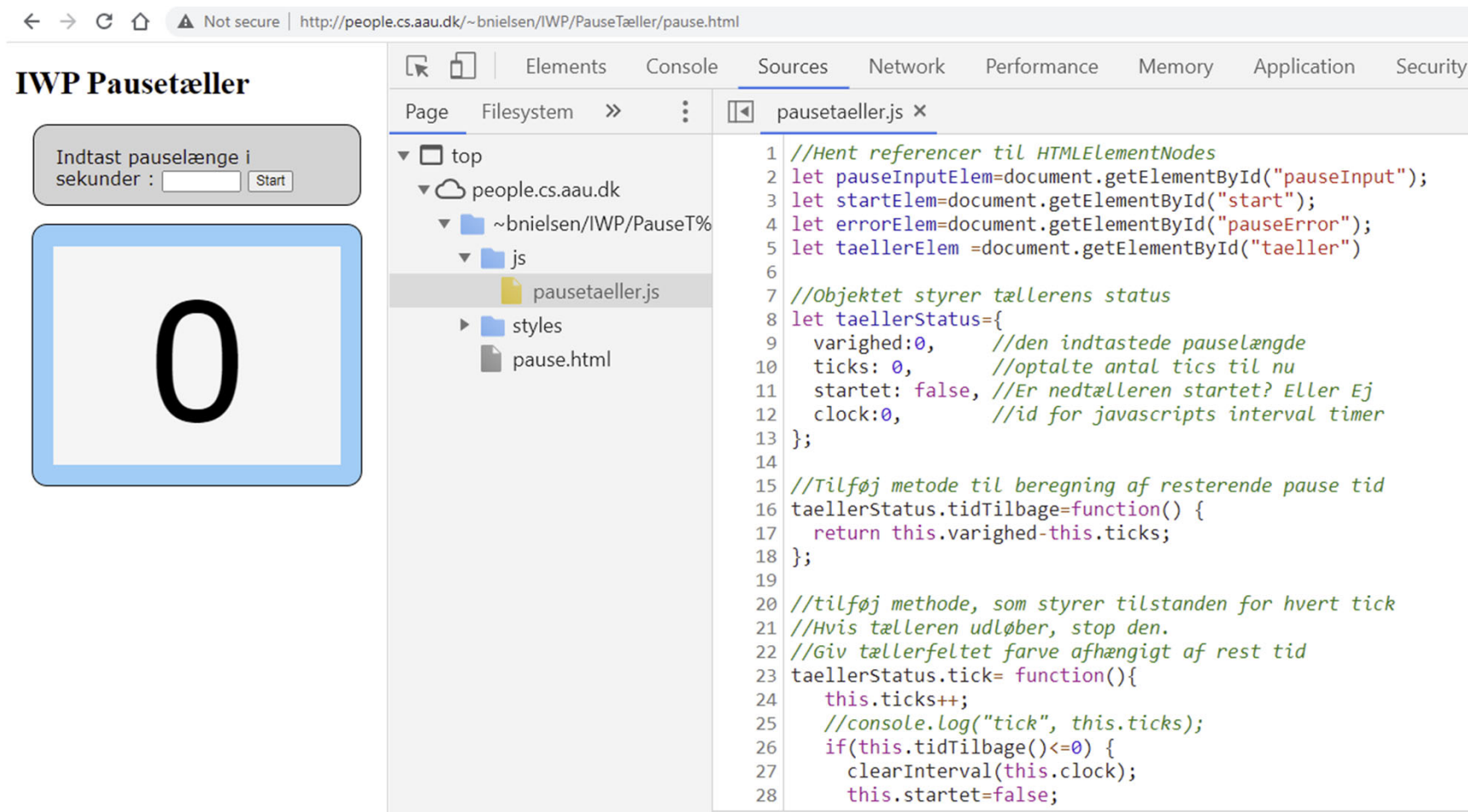
1. Indlæsningsfase

1. HTML dokumentet indlæses, parses/fortolkes
 - Browser opbygger sidens struktur som et familie-træ af HTML objekter
 - Nedlæsning af eksterne ressourcer (CSS/Scripts/imgs) påbegyndes (separate HTTP kald)
2. Når browser møder et <script> element, udføres scriptet
 - Med mindre special-tilfælde: defer, async, eller modul-scripts
 - Typisk defineres en masse funktioner og egenskaber.
3. Flere scripts udføres i den **rækkefølge** de er erklæret i
 - Med mindre special-tilfælde: defer, async, eller EC6 modul-scripts
4. Resterende ressourcer, fx billeder hentes, fortolkes, og vises
5. Siden er færdighentet, optegnet: status "load"et

2. Event-fase

- Browseren lytter efter "events" fra bruger og system, og kalder javascript funktioner til event håndtering (event-handlers)

JavaScript code sendes som kildetekst!



The screenshot displays a web browser window with the URL `http://people.cs.aau.dk/~bnielsen/IWP/PauseTæller/pause.html`. The page title is "IWP Pausetæller". The application interface includes a text input field labeled "Indtast pauselængde i sekunder :", a "Start" button, and a large digital display showing the number "0".

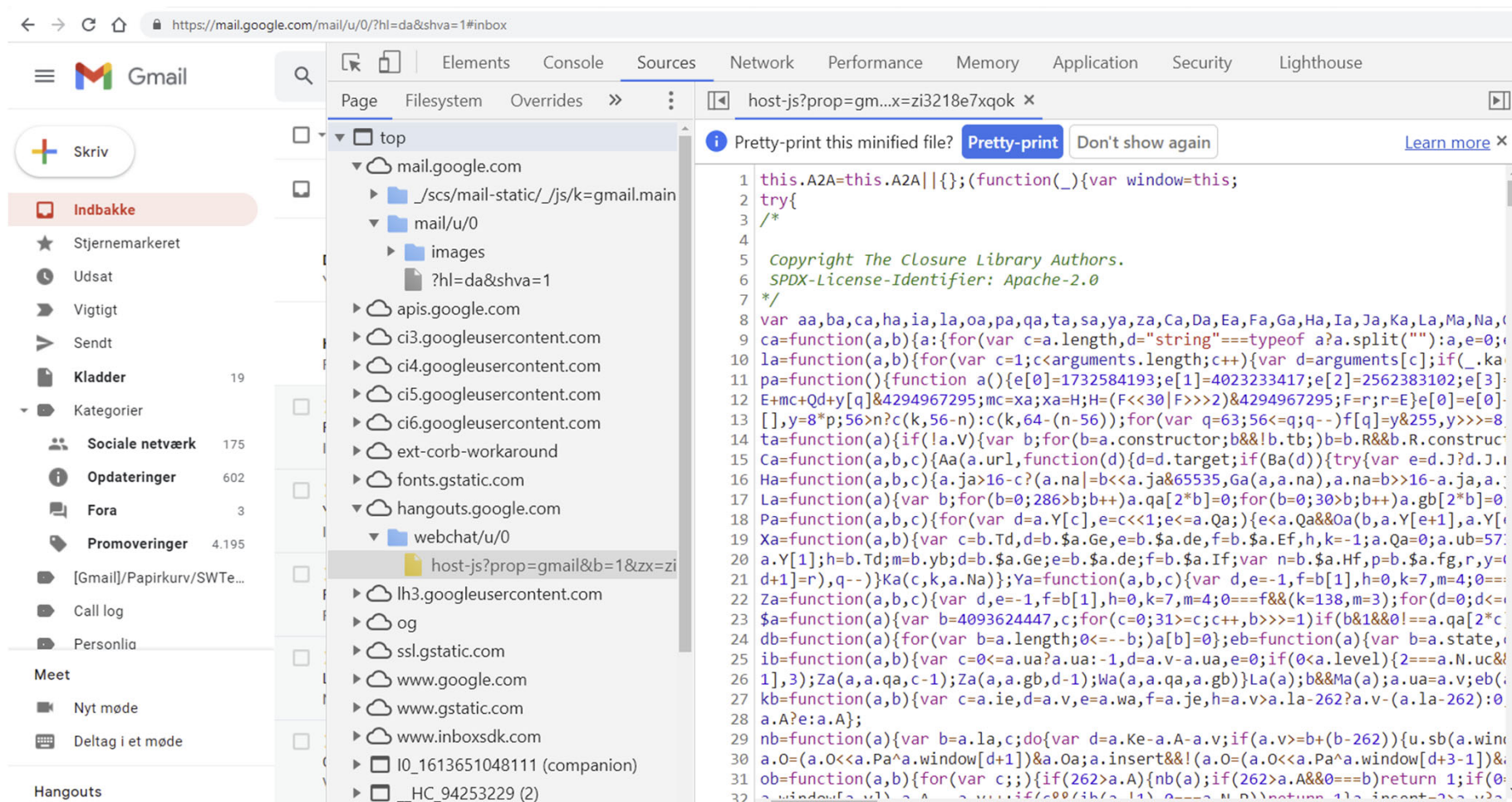
The browser's developer tools are open, showing the "Sources" panel. The file tree on the left indicates the JavaScript file `pausetaeller.js` is loaded. The right pane shows the source code of `pausetaeller.js`, which is as follows:

```
1 //Hent referencer til HTMLElementNodes
2 let pauseInputElem=document.getElementById("pauseInput");
3 let startElem=document.getElementById("start");
4 let errorElem=document.getElementById("pauseError");
5 let taellerElem =document.getElementById("taeller")
6
7 //Objektet styrer tællerens status
8 let taellerStatus={
9   varighed:0,      //den indtastede pauselængde
10  ticks: 0,        //optalte antal tics til nu
11  startet: false,  //Er nedtælleren startet? Eller Ej
12  clock:0,         //id for javascripts interval timer
13 };
14
15 //Tilføj metode til beregning af resterende pause tid
16 taellerStatus.tidTilbage=function() {
17   return this.varighed-this.ticks;
18 };
19
20 //tilføj metode, som styrer tilstanden for hvert tick
21 //Hvis tælleren udløber, stop den.
22 //Giv tællerfeltet farve afhængigt af rest tid
23 taellerStatus.tick= function(){
24   this.ticks++;
25   //console.log("tick", this.ticks);
26   if(this.tidTilbage()<=0) {
27     clearInterval(this.clock);
28     this.startet=false;
```

Javascript kode

- **Distribueres som kildetekst** (source-code)!
 - Da fortolkeren direkte arbejder med kildeteksten.
 - Alle kan læse, inspicere/debugge, kopiere koden, udlæse variable under køretid, fx vha. browserens "developer mode"!
 - Kan "**obfuskeres**", som gør det svært for mennesker at finde mening med koden. Skjuler stadig ikke følsomme oplysning.
 - Kan "**minimeres**" så det fylder mindre (hurtigere at loade)
- Kode og information, som skal holdes hemmelig for brugere,
 - bør slet ikke sendes til klienter, men holdes på server-siden
 - eller sendes og forblives krypteret
 - følsomme oplysninger (passwords, api-keys) som kan misbruges, hvis de udlæses, må ikke sendes til klienten: holdes på server-siden.

Vil du have klient kilde-tekst til Gmail el. Hangouts?



The screenshot shows a web browser window with the Gmail interface. The address bar displays the URL: `https://mail.google.com/mail/u/0/?hl=da&shva=1#inbox`. The Gmail sidebar on the left shows the 'Indbakke' (Inbox) and various filters. The 'Sources' tab is selected in the browser's developer tools, showing a list of domains and files. The file `host-js?prop=gm...x=zi3218e7xqok` is selected, and its minified JavaScript code is displayed in the right pane. The code is a single large function that initializes the Gmail client, including various utility functions and the main application logic. The code is minified and obfuscated, making it difficult to read. The browser's developer tools also show the 'Elements' and 'Console' tabs.

Page | Filesystem | Overrides | **Sources** | Network | Performance | Memory | Application | Security | Lighthouse

host-js?prop=gm...x=zi3218e7xqok

Pretty-print this minified file? [Pretty-print](#) [Don't show again](#) [Learn more](#)

```
1 this.A2A=this.A2A||{};(function(_){var window=this;
2 try{
3 /*
4
5 Copyright The Closure Library Authors.
6 SPDX-License-Identifier: Apache-2.0
7 */
8 var aa,ba,ca,ha,ia,la,oa,pa,qa,ta,sa,ya,za,Ca,Da,Ea,Fa,Ga,Ha,Ia,Ja,Ka,La,Ma,Na,
9 ca=function(a,b){a:{for(var c=a.length,d="string"===typeof a?a.split(""):a,e=0;
10 la=function(a,b){for(var c=1;c<arguments.length;c++){var d=arguments[c];if(_
11 pa=function(){function a(){e[0]=1732584193;e[1]=4023233417;e[2]=2562383102;e[3]
12 E+mc+Qd+y[q]&4294967295;mc=xa;xa=H;H=(F<<30|F>>>2)&4294967295;F=r;r=E[e[0]
13 [],y=8*p;56>n?c(k,56-n):c(k,64-(n-56));for(var q=63;56<=q;q--)f[q]=y&255,y>>>8
14 ta=function(a){if(!a.V){var b;for(b=a.constructor;b&&!b.tb;)b=b.R&&b.R.construc
15 Ca=function(a,b,c){Aa(a.url,function(d){d=d.target;if(Ba(d)){try{var e=d.J?d.J.I
16 Ha=function(a,b,c){a.ja>16-c?(a.na|=b<<a.ja&65535,Ga(a,a.na),a.na=b>>16-a.ja,a.
17 La=function(a){var b;for(b=0;286>b;b++)a.qa[2*b]=0;for(b=0;30>b;b++)a.gb[2*b]=0
18 Pa=function(a,b,c){for(var d=a.Y[c],e=c<<1;e<=a.Qa;){e<a.Qa&&0a(b,a.Y[e+1],a.Y[
19 Xa=function(a,b){var c=b.Td,d=b.$a.Ge,e=b.$a.de,f=b.$a.Ef,h,k=-1;a.Qa=0;a.ub=57
20 a.Y[1];h=b.Td;m=b.yb;d=b.$a.Ge;e=b.$a.de;f=b.$a.If;var n=b.$a.Hf,p=b.$a.fg,r,y=
21 d+1=r),q--}}Ka(c,k,a.Na));Ya=function(a,b,c){var d,e=-1,f=b[1],h=0,k=7,m=4;0==
22 Za=function(a,b,c){var d,e=-1,f=b[1],h=0,k=7,m=4;0==f&&(k=138,m=3);for(d=0;d<=
23 $a=function(a){var b=4093624447,c;for(c=0;31>=c;c++,b>>>=1)if(b&1&&0!==(a.qa[2*c
24 db=function(a){for(var b=a.length;0<--b;)a[b]=0;eb=function(a){var b=a.state,
25 ib=function(a,b){var c=0<=a.ua?a.ua:-1,d=a.v-a.ua,e=0;if(0<a.level){2===a.N.uc&
26 1},3);Za(a,a.qa,c-1);Za(a,a.gb,d-1);Wa(a,a.qa,a.gb)}La(a);b&&Ma(a);a.ua=a.v;eb(
27 kb=function(a,b){var c=a.ie,d=a.v,e=a.wa,f=a.je,h=a.v>a.la-262?a.v-(a.la-262):0
28 a.A?e:a.A};
29 nb=function(a){var b=a.la,c;do{var d=a.Ke-a.A-a.v;if(a.v>=b+(b-262)){u.sb(a.winc
30 a.O=(a.O<<a.Pa^a.window[d+1])&a.Oa;a.insert&&!(a.O<<a.Pa^a.window[d+3-1])&
31 ob=function(a,b){for(var c;){if(262>a.A){nb(a);if(262>a.A&&0===b)return 1;if(0
32 a.window[a.v])a.A=a.v;if(c%!(b(a.la)-0===a.N.O))return 1;a.insert=262-a.v;}
```

Document Object Model

DOM-API

- **Document Object Model (DOM)** er en W3 standardiseret, platforms- og sprog neutralt API, der tillader scripts at dynamisk tilgå og opdatere indhold, struktur, og stil i et HTML dokument.
- **HTML DOM**
 - Hvert HTML element er spejlet som et (JS) **objekt** med tilhørende **egenskaber**
 - Element objekterne er organiseret i et "familietræ", som følger nesting af elementerne
- Hvordan forespørger eller *udvælger* man (JavaScript) et specifikt element i et dokument?
- Hvordan *traverser man* et dokument og finder HTML søskende, efterfølgere, forældre elementer?
- Hvordan forespørger og ændre man attributter på HTML elementer?
- Hvordan ændrer man indhold og layout af et dokument?
- Hvordan ændre men strukturen af et dokument ved at oprette, indsætte, og slette HTML elementer?

Træ-struktur for HTML

Relationer som alm. familie træ:

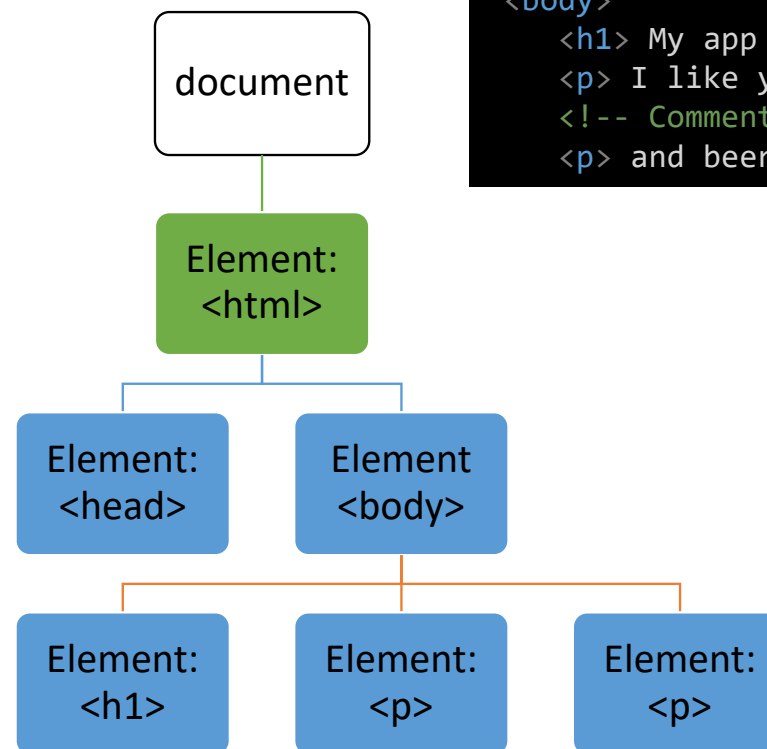
- Elementer på samme niveau: *Søskende*
- Elementer på underliggende niveauer: *efterfølgere*
- Elementer på overliggende niveauer: *forældre*
- Elementer på umiddelbart overeau: *forældre*
- Elementer på umiddelbart underliggende niveau: *børn*

document

Repræsenterer en indlæst web-side i browser

Indbygget global variabel "*document*"

Udgør roden i DOM træet

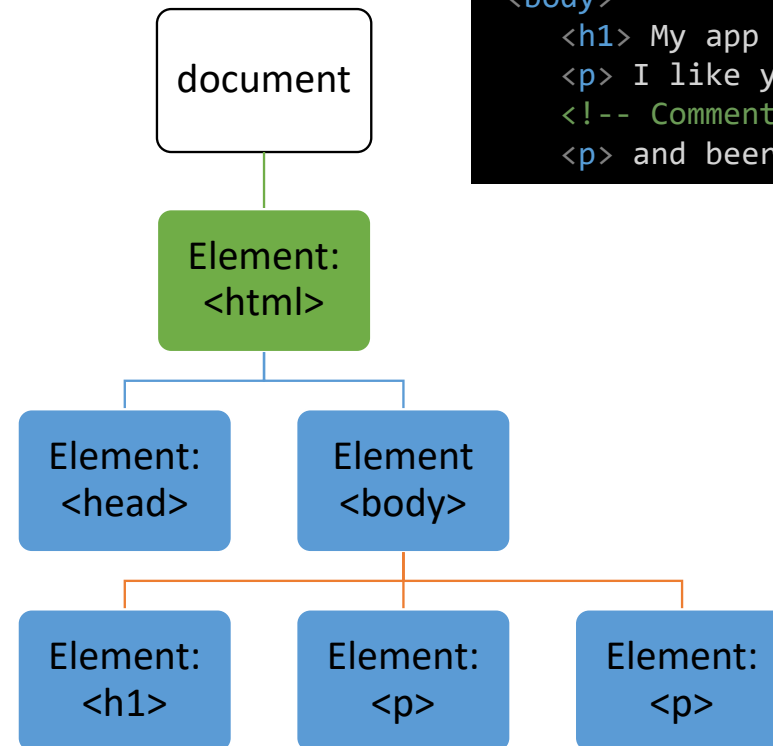


```
<html>
  <head>...</head>
  <body>
    <h1> My app page</h1>
    <p> I like yatzy</p>
    <!-- Comment --> xx
    <p> and beer </p>
```

Træ-struktur for HTML: "HTML Element-træet"

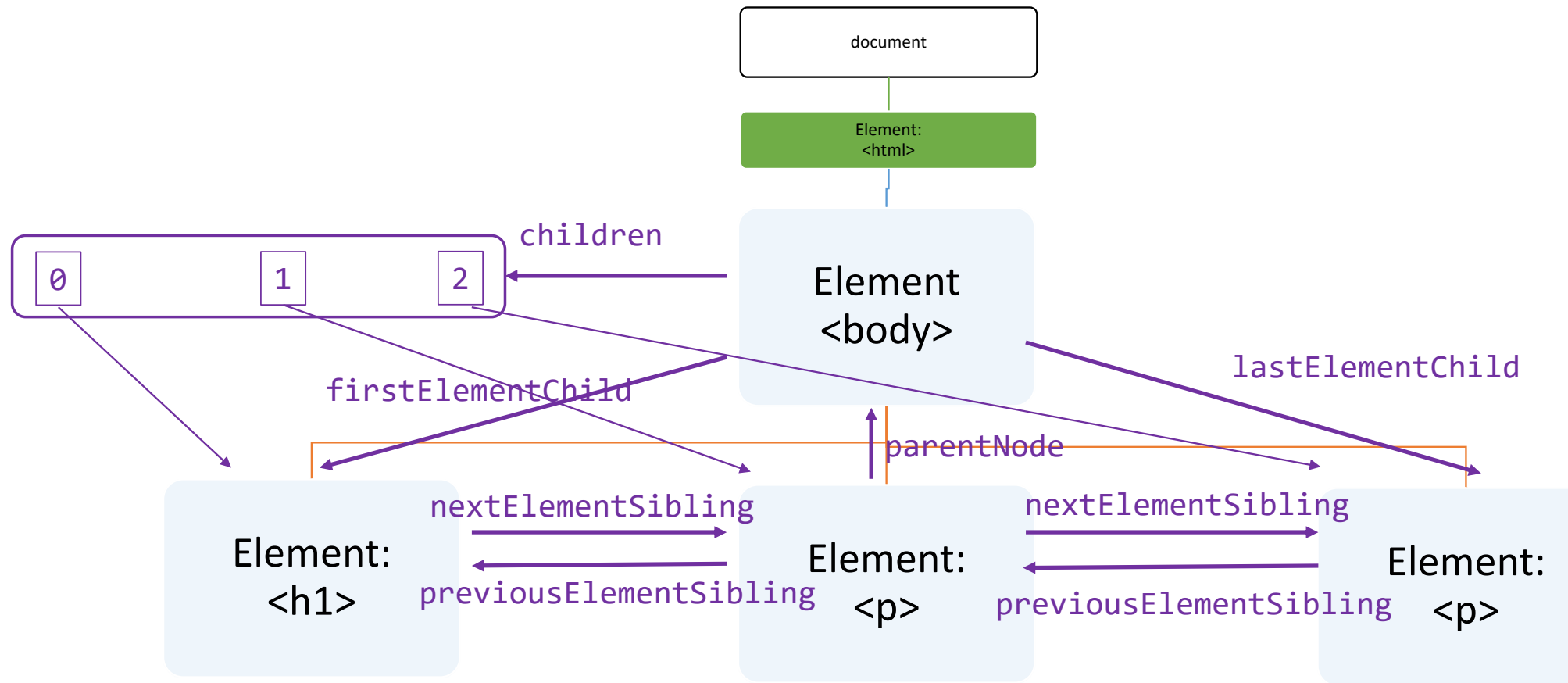
Hvert HTML elementer har flg. egenskaber

- `children`
et array lignende objekt med børnene
- `firstElementChild`, `lastElementChild`
udpeger første og sidste barn
- `nextElementSibling`, `previousElementSibling`
hæfter søskende sammen i en dobbelt-kædet liste
- `parentNode`
forældre elementet



```
<html>
  <head>...</head>
  <body>
    <h1> My app page</h1>
    <p> I like yatzy</p>
    <!-- Comment --> xx
    <p> and beer </p>
```

Træ-struktur for HTML: "Element-træet": traversering



```
document.body.children[0].textContent;  
// "My app page"
```

Træ-struktur for HTML: "NODE-træet"

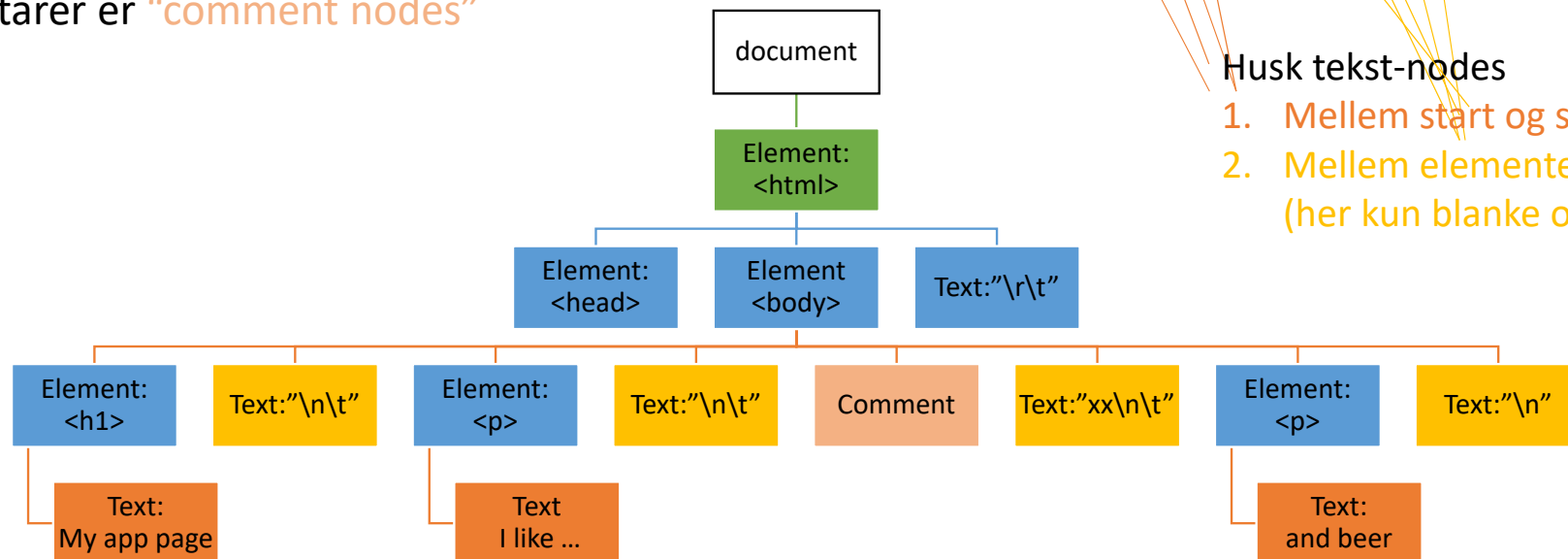
Alt i et HTML dokument er en knude/"node" objekt:

- Dokumentet selv er en node
- Alle HTML elementer er "element nodes"
- Text indenfor HTML elementer "text nodes"
- Kommentarer er "comment nodes"

```
<body>
  <h1> My app page</h1>
  <p> I like yatzy</p>
  <!-- Comment --> xx
  <p> and beer </p>
```

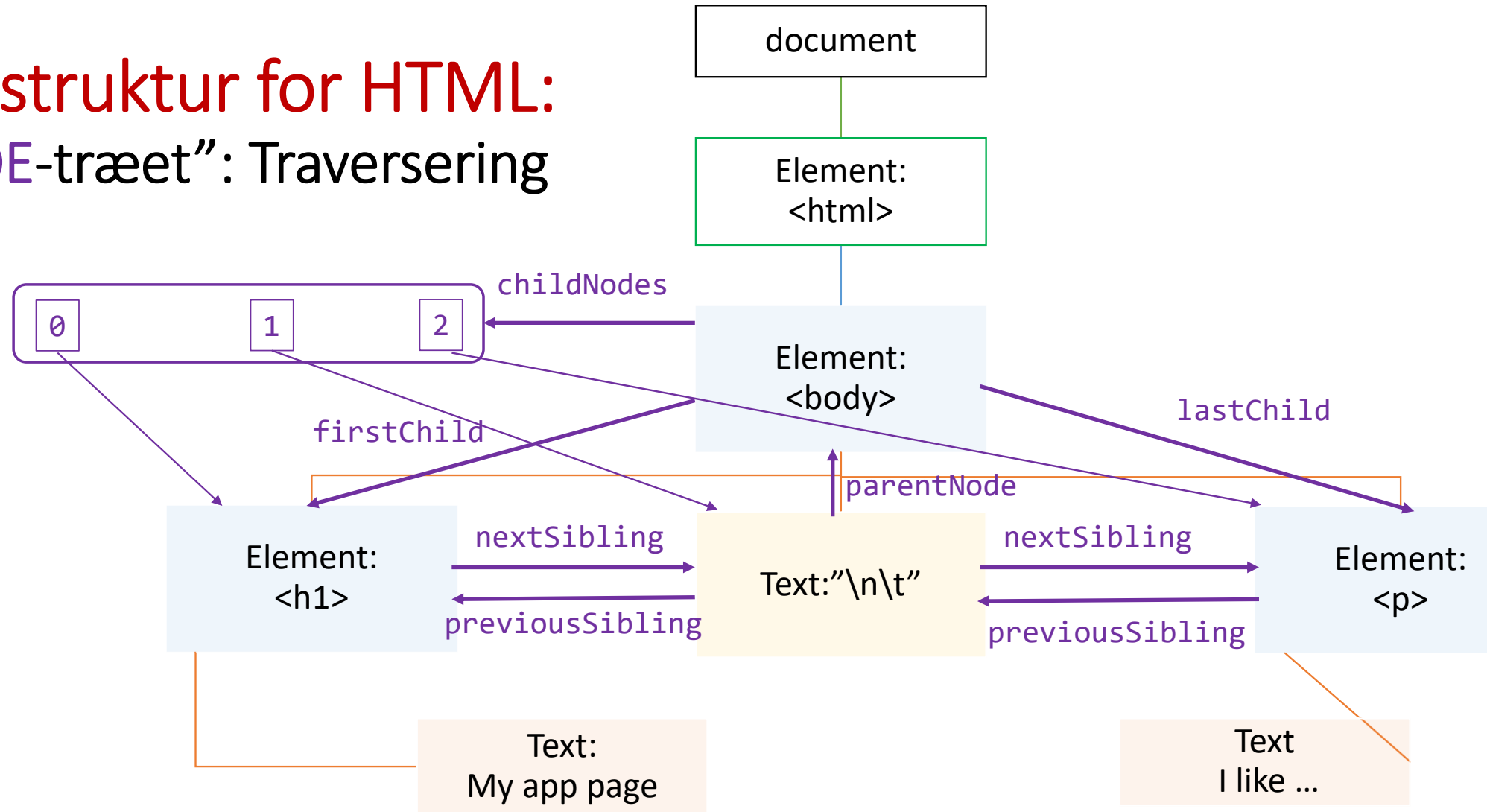
Husk tekst-nodes

1. Mellem start og slut tags
2. Mellem elementerne (her kun blanke og \n)



Træ-struktur for HTML:

"NODE-træet": Traversering



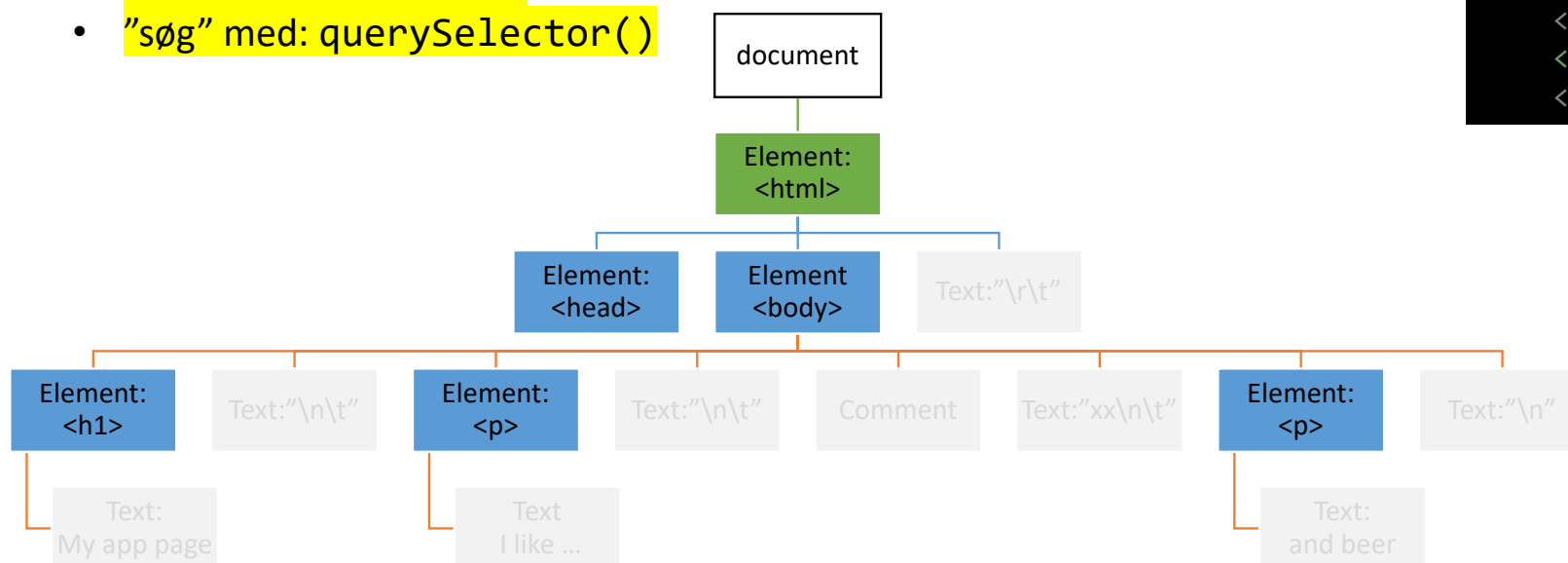
```
document.body.childNodes[0].firstChild.nodeValue;  
// "My app page"
```

Traversering af NODE-træet er **skrøbeligt** pga "tomme text felter" og ændringer i HTML strukturen

Træ-struktur for HTML: Lidt mere præcist "Element-træet"

Browser vedligeholder 2 træer

- Node-træet
- Element træet: A la Node træ, hvor KUN HTML elementer er medtaget
- **BRUG ELEMENT-TRÆET!**
- "søg" med: `querySelector()`



```
<body>
  <h1> My app page</h1>
  <p> I like yatzy</p>
  <!-- Comment --> xx
  <p> and beer </p>
```

HTML Element egenskaber

- Hvert HTML element har flg. egenskaber
- `textContent`
returnerer teksten i elementet og dets efterfølgere
- `innerHTML`
HTML koden for "indmaden" i et element, og dets efterfølgere

```
document.body.textContent;  
document.body.children[2].textContent = "and Cola";  
  
document.body.children[2].innerHTML = "and <em> beer </em>"
```

Ændrer html koden for sidste <p> element, og indsætter dermed undertræ med elementet

Skrivning til `.innerHTML` er oftest en dårlig ide! Sikkerhedshul!
DON'T!!!

"My app page
I like yatzy
xx
and beer "

```
<html>  
<head>...</head>  
<body>  
  <h1> My app page</h1>  
  <p> I like yatzy</p>  
  <!-- Comment --> xx  
  <p> and beer </p>
```

- HTML elementers attributter og styles kan ændres på tilsvarende måde!!! => Dynamisk ændring af siden!!!

Søgning efter HTML nodes

Den simple, klassiske

```
let elem = document.getElementById(string);
```

Søger dokumentet igennem efter det (ene) element hvis id attribut == string

```
<button id="minKnap"> Start </button>

...
<script>
let knapElem = document.getElementById("minKnap");
knapElem = document.querySelector("#minKnap");
</script>
```

Den moderne [selector-baserede](#): nemt, universelt, og fleksibelt

- `document.querySelector(selektorString);`
 - `document.querySelector("#elem_id")`: samme som `getElementById("elem_id")`
- `document.querySelectorAll(selektorString);`

• De gamle

```
let elems = document.getElementsByTagName(string);
```

- Søger dokumentet igennem efter de elementer hvis tag == string
- Returnerer array lignende objekt (HTMLCollection) med alle matchende elementer

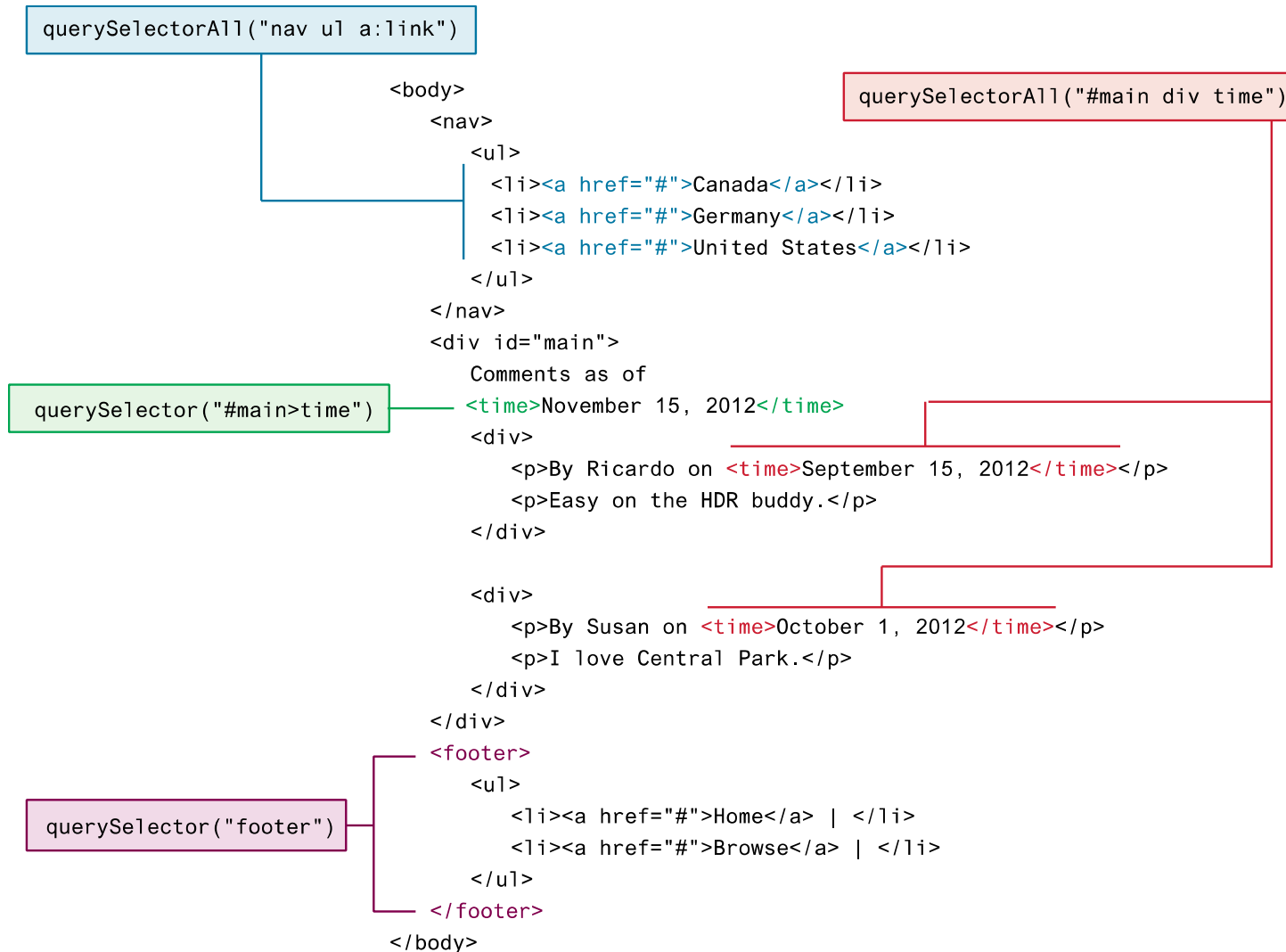
```
let elems = elem.getElementsByTagName(string);
```

- Bruger `elem` som rod-element i søgningen

```
let elems = elem.getElementsByClassName(nameString);
```

- "nameString" er en streng af navne sepereret med mellemrum
- Søger dokumentet igennem efter alle de elementer hvis class attribut == et af klasse navnene i søgestrengen
- Returnerer array lignende objekt (HTMLCollection) med alle matchende elementer

document.querySelector



HTML Elementers attributter i DOM

- HTML elementers attributter "spejles" som egenskaber i DOM

```
<body>
  

  <label for="pauseInput">Indtast pauselængde i sekunder :</label>
  <input type="number" id="pauseInput" min="1" max="1200" value="60" required >
```

```
let panda_pic=document.querySelector("#panda_pic_id");
console.log(panda_pic.id);    //"billede af et ..."
console.log(panda_pic.src);   //"https://media..."
console.log(panda_pic.alt);   //"panda_pic_id"
console.log(panda_pic.width); //"223"

let pauseInputElem=document.getElementById("pauseInput");
let varighed=Number(pauseInputElem.value); //60
```



Kan også tildeles nye værdier

class attributter findes som `classList`

Alternativ adgang til elementets attributer

- `const names = element.getAttributeNames();` //Giver et array af strenge med navnene på de definerede attributer
- `const attrValue = element.getAttribute(name);` //Giver et værdien (streng) for attributten med navnet *name* (streng)
- `const attrValue = element.setAttribute(name, value);`
//Sætter/ændrer attributten med navnet *name* (streng) til *value* (streng)

Ændring af DOM træet

- Meget fleksible metoder: append, prepend, after, before
- Oprettelse af ny element: createElement(elementNavn);
- Sletning/erstatning af nodes: replaceWith, remove

```
let title=document.querySelector("body>h1");
let subTitle=document.createElement("h2");
subTitle.append("Yatzy Fun");
title.replaceWith(subTitle);
document.body.append(title);

let img=document.createElement("img");
img.src="https://media.nationalgeographic.org/assets/photos/000/222/22252.jpg";
img.width=233;
subTitle.after(img);
```

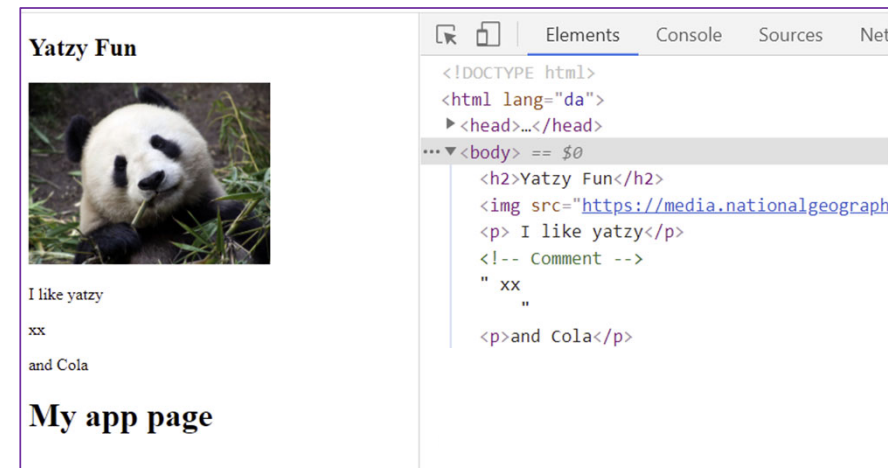
```
<html>
<head>...</head>
<body>
  <h1> My app page</h1>
  <p> I like yatzy</p>
  <!-- Comment --> xx
  <p> and beer </p>
```

My app page

I like yatzy

xx

and Cola



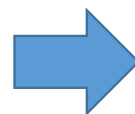
Constructing a table

```
function renderBMITable(userName,records){
  const tableElem=document.createElement("table");
  const theadElem=document.createElement("thead");
  const theadRowElem=document.createElement("tr");
  const theadColumnElem=document.createElement("th")
  const tbodyElem=document.createElement("tbody");
  theadColumnElem.append("BMIStats for user "+userName);
  theadRowElem.append(theadColumnElem);
  theadColumnElem.setAttribute("colspan",2);

  theadElem.append(theadRowElem);
  theadElem.append(createRow("th",["Weight","Bmi"]));
  tableElem.append(theadElem);

  for(let entry of records){
    const row=createRow("td",[entry.weight,entry.bmi]);
    tbodyElem.append(row);
  }
  tableElem.append(tbodyElem);
  tableElem.setAttribute("class","scoretable");
  console.log(tableElem);
  let output=document.querySelector("#stats_result_id");
  output.textContent=""; //clear existing output
  output.append(tableElem);
}
```

```
function createRow(elemType,clmnsText){
  const row=document.createElement("tr");
  for(let clmnText of clmnsText) {
    const c1=document.createElement(elemType);
    c1.append(clmnText);
    row.append(c1);
  }
  return row;
}
```



```
▼ <table class="scoretable">
  ▼ <thead>
    ▼ <tr>
      <th colspan="2">BMIStats for user Mickey</th>
    </tr>
    ▶ <tr> ... </tr>
  </thead>
  ▼ <tbody>
    ▼ <tr>
      <td>90</td>
      <td>27.78</td>
    </tr>
    ▶ <tr> ... </tr>
    ▶ <tr> ... </tr>
  </tbody>
</table>
```

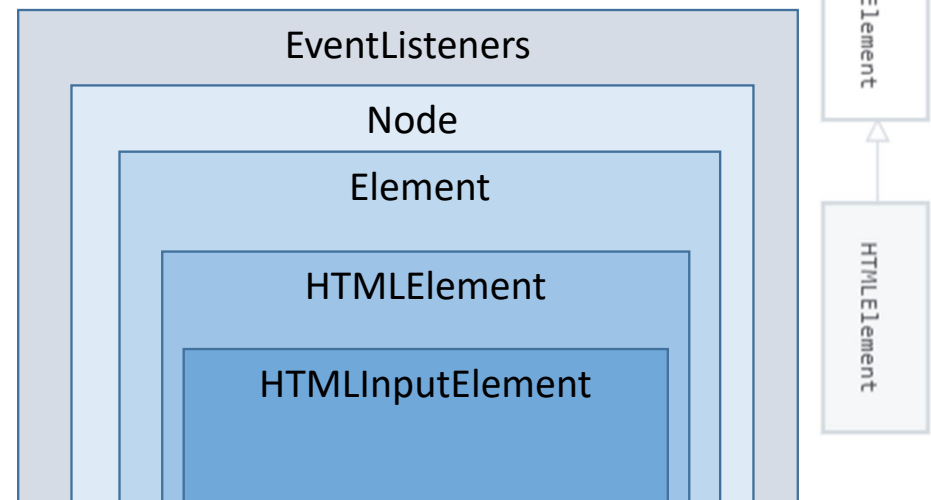
| BMIStats for user Mickey | |
|--------------------------|-------|
| Weight | Bmi |
| 90 | 27.78 |
| 188 | 58.02 |
| 188 | 58.02 |

HTML DOM Objekt-Orientering (videregående)

HTML DOM Objekt-Orientering (videregående)

- Knudetyper i DOM repræsenteres som "klasser", der "arver" fra hinanden
 - (i JavaScript: prototype kæde)
 - EventTarget: Objekter, som kan modtage og håndtere DOM events
 - Node: Knude i DOM træet, tilføjer egenskaber gældende for knuder (fx childNodes)
 - Element: Et element i træet (basis for HTML elementer) fx traverseringsegenskaber (children)
 - HTML Element: Tilføjer yderligere særkender for HTML elementer
 - HTML Input element: Tilføjer særkende egenskaber gældende for input elementer (fx value)
- HTML Element har altså alle egenskaberne fra Node, plus flere
- Et HTML Input Element er et specielt HTML Element
- Alle HTML Elementer er således EventListeners

<https://javascript.info/basic-dom-node-properties>



Hændelser

Demo

Call-back funktioner

- En funktion, der kaldes når en "interessant" hændelse er sket
- Registreres i et objekt, som detekterer hændelsen
 - Jfv messageBoard øvelse
- Her: JavaScript Timere
 - setInterval, clearInterval
 - setTimeout, clearTimeout

```
let clock=setInterval(f,1000); //1 tick per second
let ticks=10;

function f(){
  console.log(ticks);
  ticks--;
  if(ticks===0) clearInterval(clock);
}

//Eksempel fra pauseTæller; NB pilefunktion
taellerStatus.tick= function(){...}

taellerStatus.start=function(varighed){ ...
  this.clock=setInterval(() => this.tick(),1000);
```

Events

- Browser genererer en event ("hændelse") når noget "interessant" sker
 - Bruger input, mus-hændelse
 - HTML dokumentet, eller elementer heri.
- JavaScript kan abonnere på disse events og udføre en funktion når eventet indtræffer
 - Hændelse-håndterings funktion / "event-handler-funktion" / "call-back-funktion"
- Skabe interaktivitet og dynamik på en HTML side;

Device-dependent input events

- Tastatur: "keydown" and "keyup."
- Mus: "mousedown", "mousemove", "mouseup", "mouseover"
- Touch: "touchstart", "touchmove", "touchend", "keydown", and "keyup."

Device-independent input events

- "click", "input",
- "pointerdown", "pointermove", and "pointerup" event

User interface events

- HTML form element: "focus", "change", "submit"

State-change events

- Dokument tilstand: "load"

API-specific events:

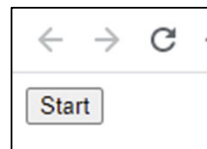
- Timers: timeout
- AV playback "waiting" "playing" "seeking" "volumechange"

Afht. [Tilgængelighed](#) bør device independent events bruges når muligt

Event og event-håndtering er typisk for GUI programmering, ikke kun browser JS.

Event begreber

- **Hændelses type:** (Event type). Ex et *click*
- **Målet for hændelsen** (event-target): Det objekt som er udsat for hændelsen, fx en specifik HTML *"button"*
- **Hændelse-håndterings funktion** ("event-handler"): en registreret funktion, som kaldes når hændelses indtræffer
- **Hændelses-objektet** ("event-objekt"): information om hændelsen
 - Overføres som argument til event-handler funktionen
 - Inkluderer "event-target", timestamp
 - For mus-events: x,y position
- **Registrerings-mekanismen:** hvordan en event-handler registreres
- **Udbredelse af hændelsen:** ("event propagation") hvordan hændelsen indfanges og udbredes i et HTML dokument med nestede elementer



```
<body>
  <button id="minKnap"> Start </button> ...
```

```
let knapElem = document.querySelector("#minKnap");

knapElem.onclick = function(event) {
  console.log(`Starter! (${event.x},${event.y})`);
};

knapElem.addEventListener("click", (event) => {
  console.log(`Starter! (${event.x},${event.y}) Igen`);
});
```

```
Starter! (39,24)
```

```
Starter! (39,24) Igen
```

```
>
```

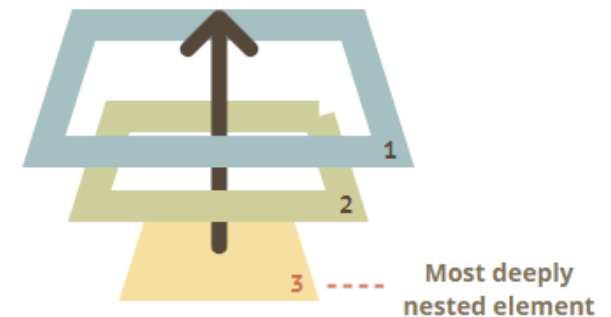
```
<input type="button" value="Send"
  onclick="alert('Hej!');" />
```

Default event-håndtering

- Browseren har en række standart event handlers:
 - Click-på et hyper-link: får browseren til at indlæse den nye web-side
 - Submit på form: browseren indsender formularen som angivet i formularens action og metode.
 - Click på et html element: normalt ingen ting.
- Du kan forhindre browseren i at udføre standart handlingen ved at indsætte "preventDefault()" i din event-handler

DOM API: EVENT Bubbling and capturing

- Der kan være flere handlers registreret på forskellige niveauer i DOM træet:
 - En event håndteres først af handlerne på det dybeste nesting niveau
 - `event.target` elementet
 - **"bubler"** derefter op til handlers på overliggende niveauer (forfædre)
 - `event.currentTarget` elementet
 - 3 faser i DOM
 1. Indfangelse (capturing): hændelsen går ned mod target.
 2. Målfasen (target): hændelsen har nået målet
 3. Boble-fasen (bubbling): hændelse håndteres fra mål mod forfædre (på overliggende niveauer)
- Der er funktioner til at håndtere events i disse 3 faser og stoppe udbredelsen (i avancerede GUI'er, ikke kritisk for IWP)
- [Mere om detektering, indfangning og håndtering](#)



Pause Tælleren

<http://people.cs.aau.dk/~bnielsen/IWP/PauseT%c3%a6ller/pause.html>

```
<label for="pauseInput">Indtast pauselänge i sekunder :</label>
<input type="number" id="pauseInput" min="1" max="1200" required list="pauseList">
<button type="button" id="start">Start</button>
```

```
//Hent referencer til HTMLElementNodes
let pauseInputElement=document.getElementById("pauseInput");
let startElem=document.getElementById("start");

//Registrer event håndteringsfunktion, der kaldes når bruger trykker start
startElem.addEventListener("click",start);

//eventhåndtering for tryk på start-knappen:
function start(event){
    //da vi ikke bruger forms, beder vi HTML om at validere
    //hvis invalid, fyrer checkValidity også en "invalid" Event
    if(pauseInputElement.checkValidity())
        resetTaeller(event);
}
function resetTaeller(event){
    let varighed=Number(pauseInputElement.value);
    taeller.textContent=String(varighed);
    taellerStatus.start(varighed); //starter interval timer;
}
```

IWP Pausetæller

Indtast pauselänge i sekunder :

Start

0

Lidt om sikkerhed

Kode injektionsangreb

- Via bruger-input at smugle kode ind, som en angriber ønsker udført
- Velkendte eksempler
 - SQL-code injection
 - [XSS: cross site scripting](#)
- Bruger input
 - skal valideres og saniteres (special tegn skal escapes eller strippes)
 - Også alt input fra nettet på serversiden!!!

Kode injektionsangreb 1: via document.write

- `document.write()` som skriver direkte til sidens HTML repræsentation
- Heldigvis bruges `document.write` mest til test formål, og bruges ikke (og I bør ikke!) længere til operationelle sites
 - og afvises af browsere under specifikke betingelser
 - ikke nogen effektiv angrebsstrategi

```
<body>
  <h1> IWP Demo</h1>
  <p>Indtast dit navn:</p>
  <input type="text" onchange="processInput(this.value)">

  <script>
  function processInput(text){
    console.log("Hej "+text);
    document.write("Hej "+text);

    // a bit more sophisticated would use:
    // document.write(document.head.innerHTML+document.body.innerHTML+"Hej "+text);
  }
</script>
</body>
</html>
```

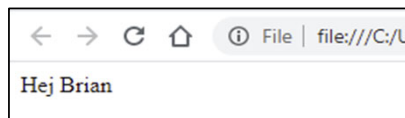
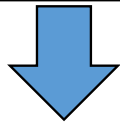
Kode injektionsangreb 1: via document.write

Normalt Brug

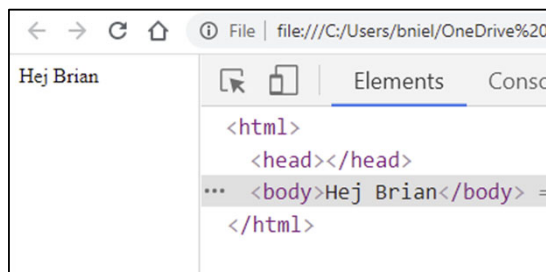


IWP Demo

Indtast dit navn:



Hej Brian



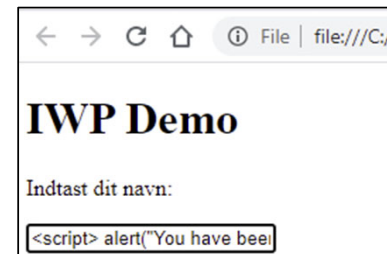
Hej Brian

```
<html>
<head></head>
... <body>Hej Brian</body> =
</html>
```

Ondsindet brug

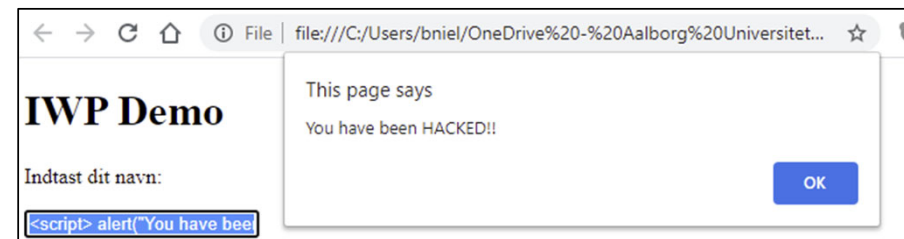
Indtast flg. som navn

`<script> alert("You have been HACKED!!");</script>`



IWP Demo

Indtast dit navn:

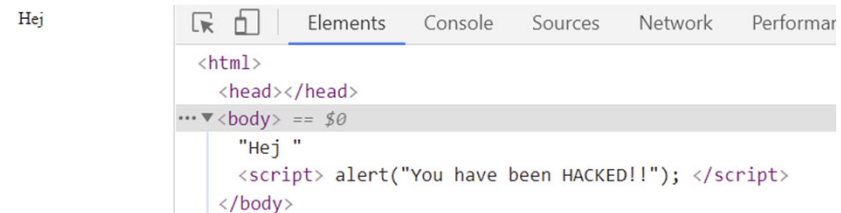


IWP Demo

Indtast dit navn:

This page says
You have been HACKED!!

OK



Hej Brian

```
<html>
<head></head>
... <body> == $0
    "Hej "
    <script> alert("You have been HACKED!!"); </script>
</body>
```

Kode injektionsangreb 2: via innerHTML

- Skrivning til et HTML elements innerHTML attribut er var en hyppigt anvendt form for DOM omskrivning
- Heldigvis specificerer HTML5 at `<script>` element som indsættes via innerHTML ikke må udføres
 - Forsøg med indtastning af `<script>alert("You have been HACKED!!");</script>` som navn har ikke den ønskede effekt
 - Vi må så håbe at browser-udviklere har implementeret dette check korrekt!
 - I må ikke bruge innerHTML til at tilføje HTML kode som I ikke har total kontrol over (fx fra bruger input)

```
<body>
  <h1> IWP Demo</h1>
  <p>Indtast dit navn:</p>
  <div>
    <input type="text" onchange="processInput(this.value)">
  </div>
  <script>
    function processInput(text){
      document.body.innerHTML += "Hej " + text;
    }
  </script>
```



Kode injektionsangreb 3: via innerHTML

- Skrivning til et HTML elements innerHTML attribut er en hyppigt anvendt form for DOM omskrivning
- Men der er mange andre måder at få JavaScript kode in på!
 - Forsøg med "navnet"

 - Når img elementet sættes via innerHTML følger event-handleren med.
 - Kaldes når billedet ikke kan indlæses
 - I må ikke bruge innerHTML til at tilføje HTML kode som I ikke har total kontrol over (fx fra bruger input)

```
<body>
  <h1> IWP Demo</h1>
  <p>Indtast dit navn:</p>
  <div>
    <input type="text" onchange="processInput(this.value)">
  </div>
  <script>
    function processInput(text){
      document.body.innerHTML += "Hej " + text;
    }
  </script>
```



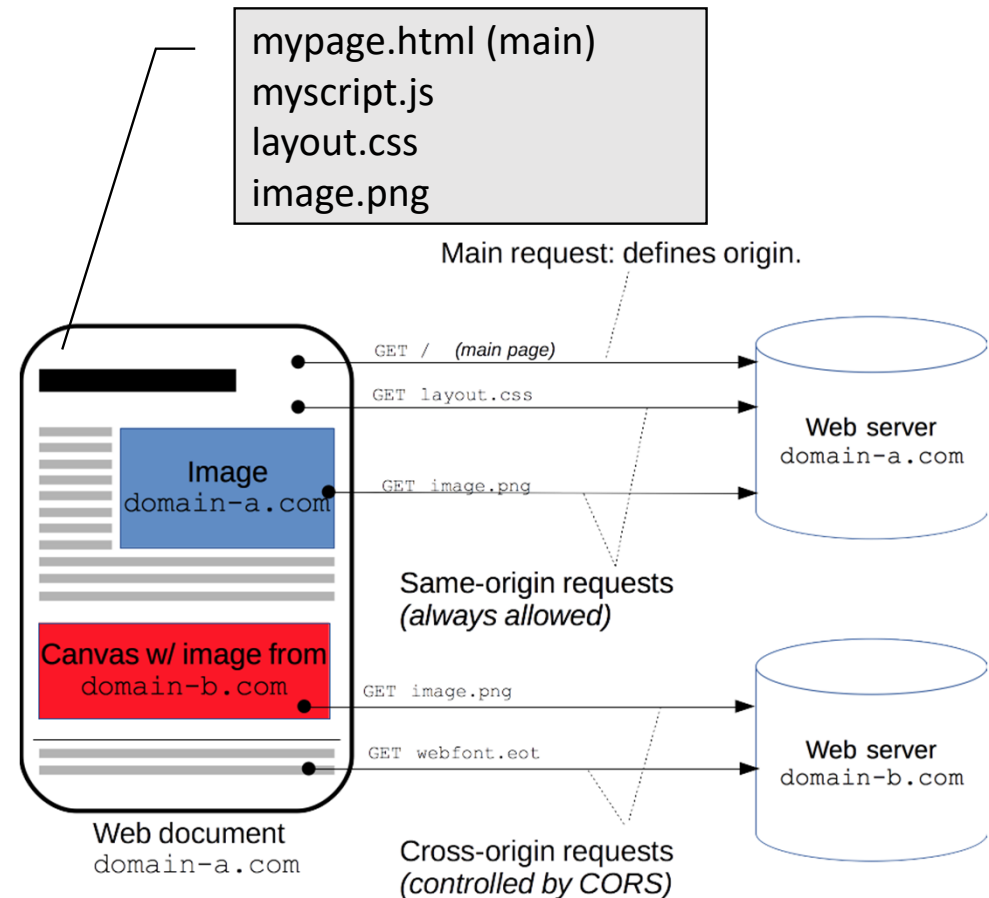
Kode injektionsangreb

- Hackere er på evig jagt efter smut-huller!
- Snedige!
- Ikke kun ej JavaScript fænomen
- Huller på klient-siden
 - ALT INPUT SKAL VALIDERES, SANITERES, OG BEHANDLES OMHYGGELIGT
- Huller på server siden:
 - ALT INPUT FRA NETTET SKAL VALIDERES, SANITERES, OG BEHANDLES OMHYGGELIGT

JS i browsere ("CORS fejl")

- Når JS afvikles i en browser (sandkasse) begrænser den (i samarbejde med server) hvad fetch må hente hvorfra
- Et script afvikles med "same-origin" sikkerhedspolitik: Et JS script må tilgå (kalde http metoder på) ressourcer fra samme oprindelse hvor det selv kommer fra
- **Origin/Oprindelse** : hostnavn + portnr + protokol fx <https://domain-a.com:80>
- "Cross origin": et script's forsøg på at hente visse typer data fra et andet sted, fx domain-b.com: **afvises**
- **CORS: Cross-origin-ressource sharing**: en http mekanisme, der gør det muligt for server admin (fx domain-b.com) at give tilladelse til at (visse af) dets ressourcer må tilgås fra domain-a.com (whitelisting)

Eksempel på angreb: https://en.wikipedia.org/wiki/Cross-site_request_forgery



Hvis I får "CORS" fejl, er dette årsagen.

Bemærk:

- HTML sider, som indlæses fra filsystem har et "NULL" Origin
 - fetch herfra virker normalt ikke på de fleste servere
 - Udvikling af web-applikationer kræver normalt en lille server som udviklings host, så origin bliver denne, fx `http://127.0.0.1:3000`

✖ Access to fetch at '<http://people.cs.aau.dk/~bnielsen/IWP/scores.json>' from origin 'null' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.

- **Scripts** og billeder kan godt loades fra non-origin:
- Lav en test.html fil, indsæt nedenstående element, og indlæs den i browseren fra filsystem.

```
<script src="http://people.cs.aau.dk/~bnielsen/IWP/test.js"></script>
```

- I stoler altså på origin af det inkluderende html dokumentet,
- JS biblioteker kan deles på denne måde

Deling af JS biblioteker

- www.cs.aau.dk henter JS kode fra google og twitter!

The screenshot shows a web browser with the URL <https://www.cs.aau.dk>. The website has an orange header with the Aalborg University logo and the text "DEPARTMENT OF COMPUTER SCIENCE". Below the header, there are two sections: "EDUCATION" with a photo of three students looking at a laptop, and "RESEARCH" with a photo of a person using a smartphone. The browser's developer tools are open, showing the "Sources" tab. The file tree on the left lists various assets, including "digitalAssets" (folders 917, 939, 947, 957, 979), "fast.fonts.com", "fast.fonts.net", "platform.twitter.com" (widgets.js), "www.design2013.aau.dk", "www.google-analytics.com" (plugins/ua, linkid.js, analytics.js), "www.googletagmanager.com", "www.resources.aau.dk", "aau-search-web-prod.azurewebsites.net/", and "rufous-sandbox (about:blank)". The "analytics.js" file is selected, and its source code is displayed in the right pane. The code is minified JavaScript, starting with a function definition and including copyright information for The Closure Library Authors under the Apache-2.0 license. A tooltip points to the file path "top/www.cs.aau.dk/digitalAssets/947".

Asynkron program udførelse

Concurrency

Asynkrone Programmer

Pop quizzz

Hvad udskriver nedenstående program?

```
console.Log("IWP:");  
setTimeout(()=>console.log("Hej"),1000);  
console.log("med dig");
```



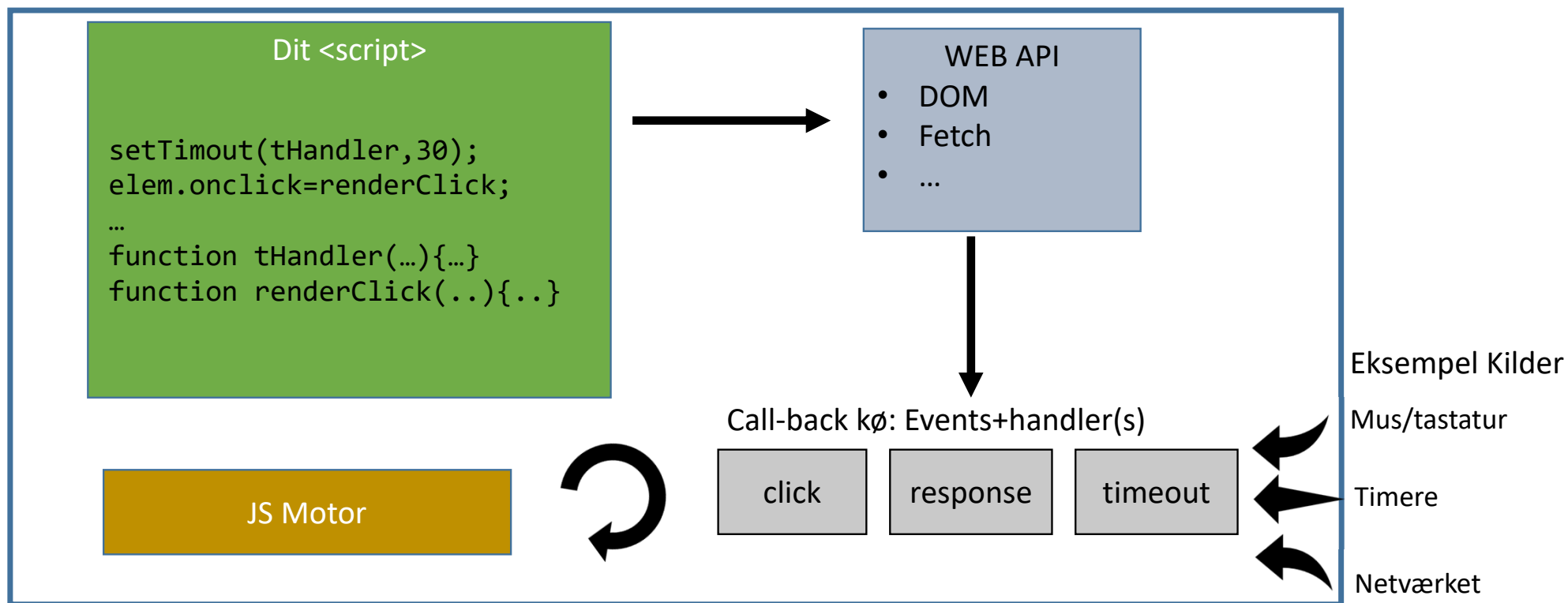
IWP:
med dig
Hej

- setTimeout tager en call-funktion som argument, registrerer denne, og returnerer umiddelbart.
- JS Systemet udfører call-back funktionen efter det angivne antal milli-sekunder
- Forårsager en aktivitet "ude af trit" med program teksten (asynkront)

```
> console.log("IWP:");setTimeout(()=>console.log("Hej"),1000); console.log("med dig");  
IWP: VM483:1  
med dig VM483:1  
< undefined  
Hej VM483:1  
.
```

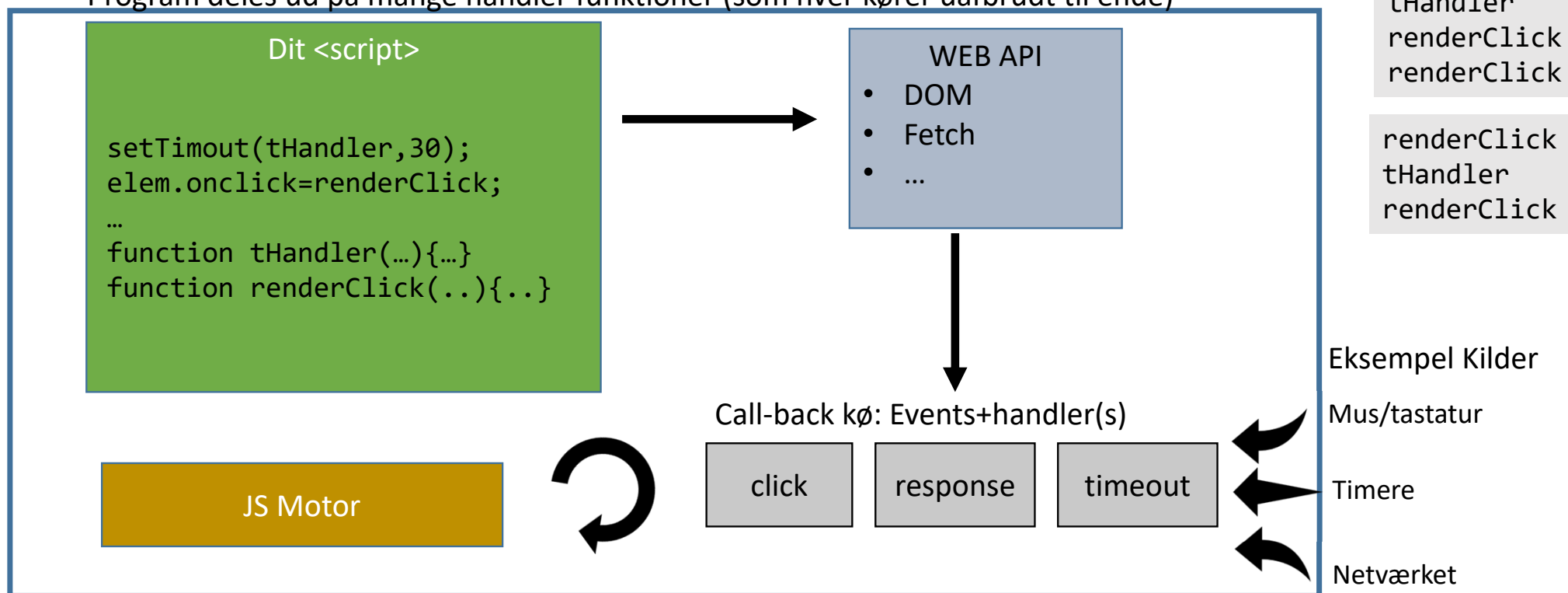
Event-loopet

- JS udfører sekventielt én handler funktion ad gangen og kører den til "slut"
- Både node.js og browsere kører på denne måde.



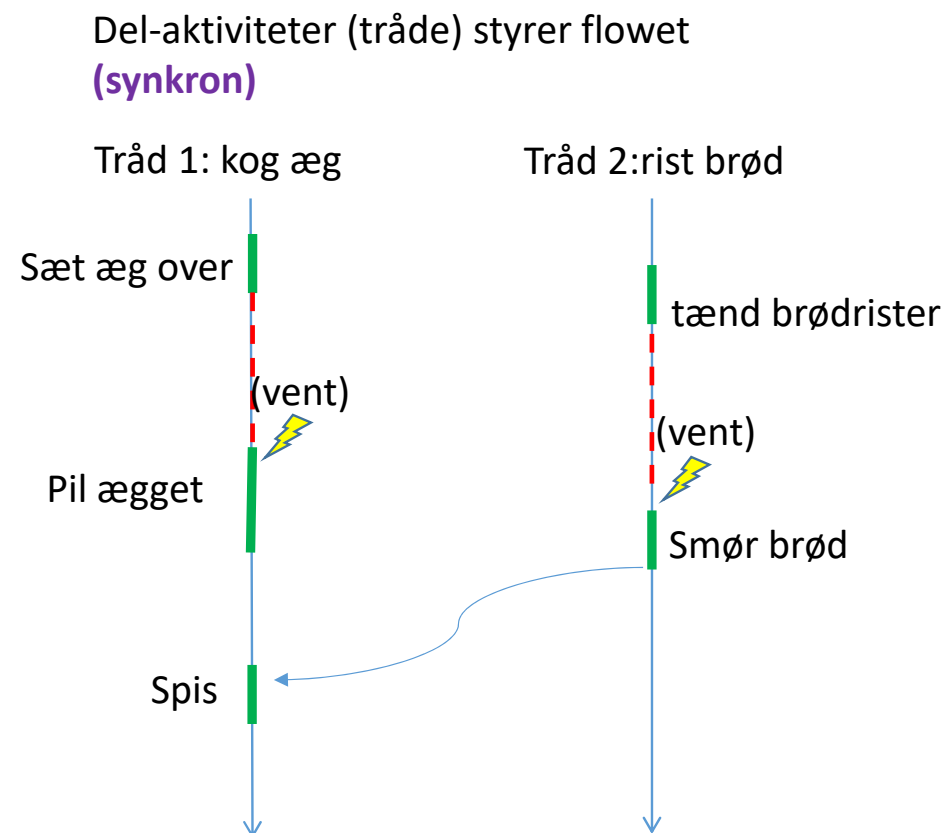
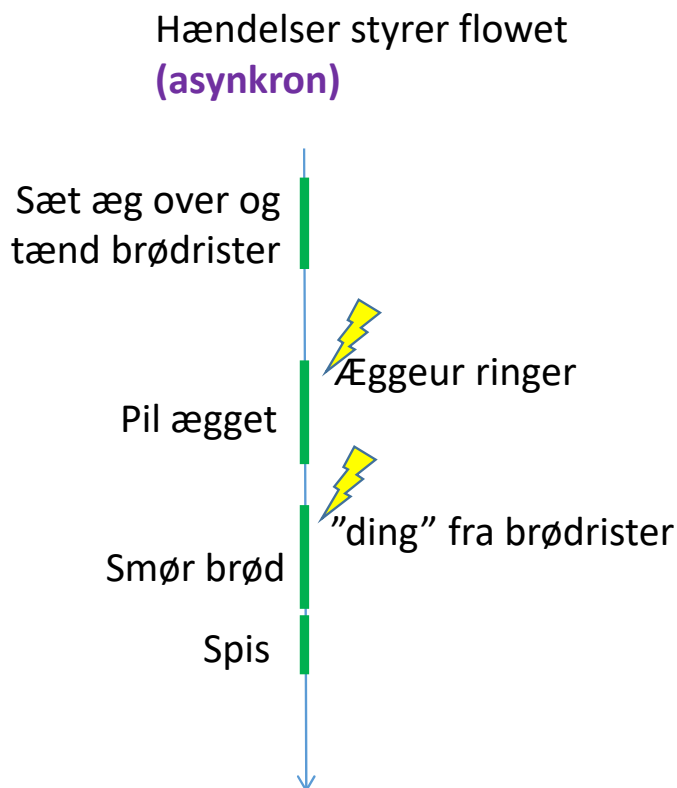
Event-loopet

- Asynkron afvikling: længerevarende operationer brydes op i 2 dele: Igangsætning og afslutning
 - FX: bestilling af en timer; kørsel af handleren
 - FX: registrering af en "click" handler; kørsel af handleren når der er click'et
 - FX: Send HTTP request; kørsel af handler til håndtering af svaret.
- I den mellemliggende periode afvikles andre events!
- Program deles ud på mange handler funktioner (som hver kører uafbrudt til ende)



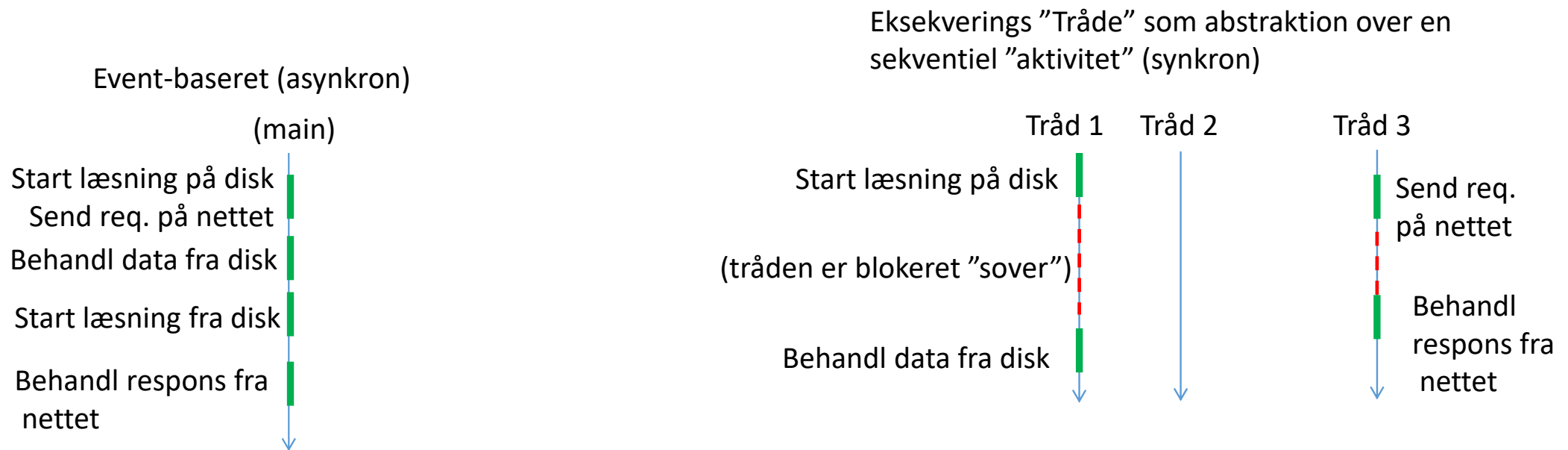
En analogi: Lav Morgenmad

- Håndtering af samtidige aktiviteter: Kog æg og rist brød
- Håndtering via hændelser (asynkron) eller aktivitets-tråde (synkron)



Concurrency eller "Samtidighed"

- En computer er i stand til at gøre mange ting samtidigt
 - Læse disk, holde øje tastatur, modtage fra netværket
 - Operativ system "multitasker" imellem flere programmer
 - Afvikle opgaver og beregninger ægte parallelt vha. flere processor kerner.



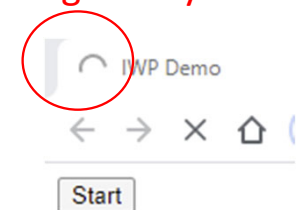
NB: synkron og asynkron optræder i forskellige sammenhænge med lidt forskellig betydning

Indlæsning af web-ressource fra browser

- Nedlæsning fra nettet er en tidskrævende aktivitet, behandles normalt asynkront!
 1. Browser igangsætter indlæsning; afsender request
 2. Lytter efter og behandler events som normalt
 3. Når responset ankommer, behandles det
- Ellers vil siden "hænge" og ikke føles interaktiv
 - Gælder også lang behandlingstid fra event-handlers

```
knapElem.addEventListener("click", (event) => { while(1); });
```

Evigt "busy"



FETCH API

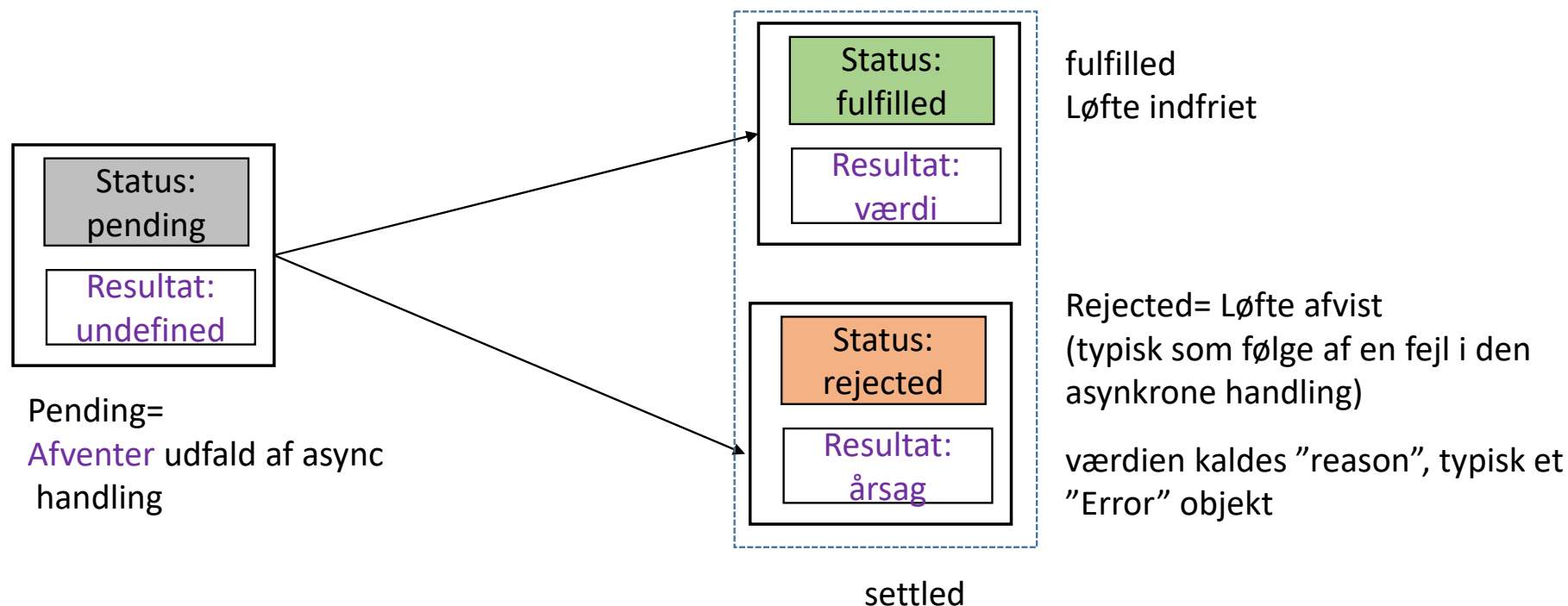
- [Fetch](#) er et **promise-baseret** API til hentning af web-ressourcer introduceret i "moderne" JS
 - Ofte indlæser klienten javascript objekter fra server siden af applikationen
 - Serialiseret som en tekst streng i JSON format.
 - Alternativ i komplekse apps: "Call-back hell"
 - Få asynkron kode til at "ligne" synkron kode som en sammenhængende sekvens af handlinger
- `let promise = fetch(url, [options])`
- `promise.then(f)`: Afvikler funktionen f, når resultatet er klart.
- Fx Indlæsning af et JSON dokument fra server

```
fetch("scores.json")
  .then(response=> {return response.json()})
  .then(data=>{console.log(data);})
  .catch(reportError)
```

- Fetch bruger GET som default, men kan tage et yderligere options argument til opsætning af metode (fx POST) ,header, evt. body.

JavaScript Promises

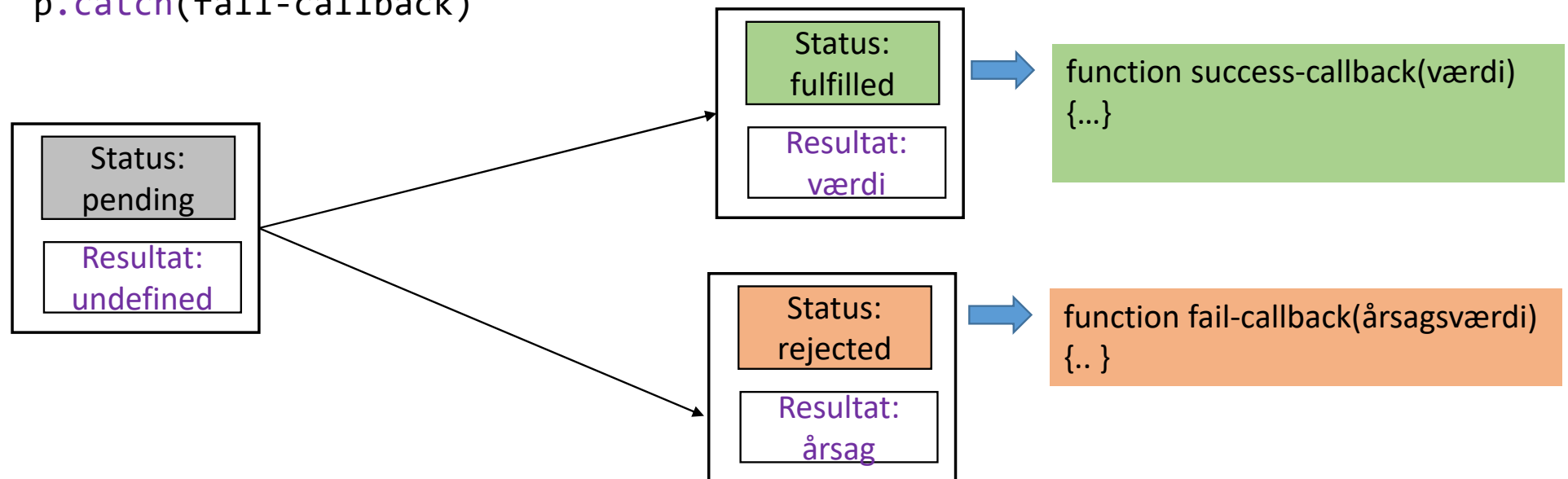
- Et **promise** er et objekt som er en pladsholder for en fremtidig værdi (resultat af en asynkron handling)
 - Fx afvent svaret på HTTP forespørgsler
- Mål: at forenkle struktur og overskuelighed af asynkrone programmer



Promises .THEN

`p.then(success-callback, fail-callback)`

- Registrerer 2 call-back funktioner, der udføres når løftet hhv. er indfriet eller afvist
- Afventer at løftet bliver afgjort
- Returnerer et nyt promise-objekt baseret på returnværdien af call-back funktionen
 - => then sætninger kan kædes i sekvens
- Normalt bruges .then kun med et argument: success-callback, og fail-call back registreres med .catch():
`p.catch(fail-callback)`



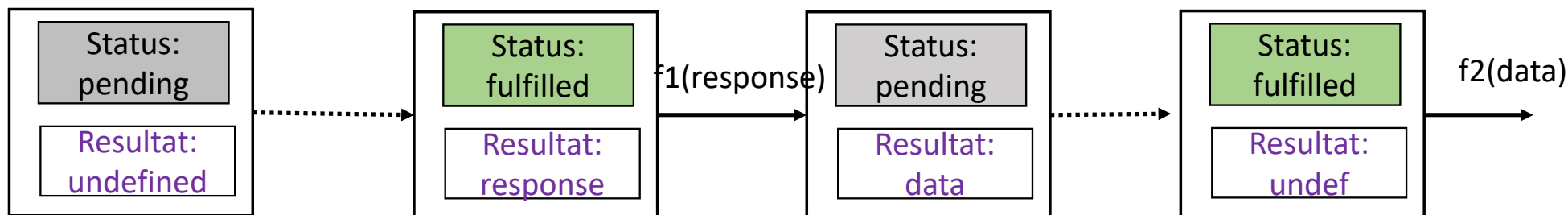
FETCH API - genbesøgt

- [Fetch](#) er et **promise-baseret** API til hentning af web-ressourcer introduceret i "moderne" JS
 - Ofte indlæser klienten JavaScript objekter fra server siden af applikationen
 - Serialiseret som en tekst streng i JSON format.
- `let promise = fetch(url, [options])`

```
fetch("scores.json")  
  .then(response=> {return response.json()})  
  .then(data=>{console.log(data);})  
  .catch(reportError)
```

f1

f2



*) bemærk: lidt forenklet, mere næste lektion!

Fetch API - genbesøgt

- Fetch promise afvises (reject) når
 - Serveren er utilgængelig (nede eller længerevarende netværksfejl)
 - Bryder browserens sikkerhedspolitik (se CORS)
- Fetch fejler **IKKE** (indfrier sit promise) hvis vi har gyldigt HTTP respons fra server, selv med en HTTP fejlkode (fx 404, "siden findes ikke").
 - Status på respons kan checkes med

```
if(response.ok) {} else { //error}
```

- [Response.status](#) — Et heltal (default værdi 200) som indeholder respons status kode.

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

Ex fra BMI -app

- bmiData er et objekt med spil konfigureringsdata, der er fisket ud fra input elementernes værdier

```
function extractBMIData(){  
  let bmiData={};  
  bmiData.userName=document.getElementById("name_id").value;  
  bmiData.height=document.getElementById("height_id").value;  
  bmiData.weight=document.getElementById("weight_id").value;  
  console.log("Extracted"); console.log(bmiData);  
  return bmiData;  
}
```

Ex fra BMI-app

- jsonPost er en selv-skrevet funktion, der opsætter fetch til at lave et POST
 - med et objekt (bmiData) som json-serialiseret indsættes i request-body

```
function sendBMI(event) {
  event.preventDefault(); //we handle the interaction with the server rather than browsers form
  submission
  document.getElementById("submitBtn_id").disabled=true; //prevent double submission
  let bmiData=extractBMIData();

  jsonPost(document.getElementById("bmiForm_id").action,bmiData)
  .then(bmiStatus=>{
    let resultElem=document.getElementById("result_id");
    resultElem.textContent=`Hi ${bmiData.userName}! Your BMI is ${bmiStatus.bmi}. Since last it
has changed ${bmiStatus.delta}!`;
    showElem(resultElem);
    document.getElementById("submitBtn_id").disabled=false; //prevent double submission
  }).catch(e=>{
    alert("Encountered Error: " +e.message + "\nPlease retry!");
    document.getElementById("submitBtn_id").disabled=false;
  });
}
```

- Når responset kommer, optegnes resultat teksten
- Grafikken opdateres, "submit" knappen tændes

Helper functions: jsonPost and jsonParse

```
function jsonParse(response){
  if(response.ok)
    if(response.headers.get("Content-Type") === "application/json")
      return response.json();
    else throw new Error("Wrong Content Type");
  else
    throw new Error("Non HTTP OK response");
}

function jsonPost(url = '', data={}){
  const options={
    method: 'POST', // *GET, POST, PUT, DELETE, etc.
    cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(data) // body data type must match "Content-Type" header
  };
  return fetch(url,options).then(jsonParse);
}
```