

Internetværk og Web-programmering

Kursus Introduktion

Forelæsning 1
Brian Nielsen

Distributed, Embedded, Intelligent Systems



The Team



Brian Nielsen,
bnielsen@cs.aau.dk



Michele Albano,
mialb@cs.aau.dk



Hjælpelærere

- Nicholas Duerlund Jørgensen
- Sebastian Lassen
- Rasmus B.V. Borup

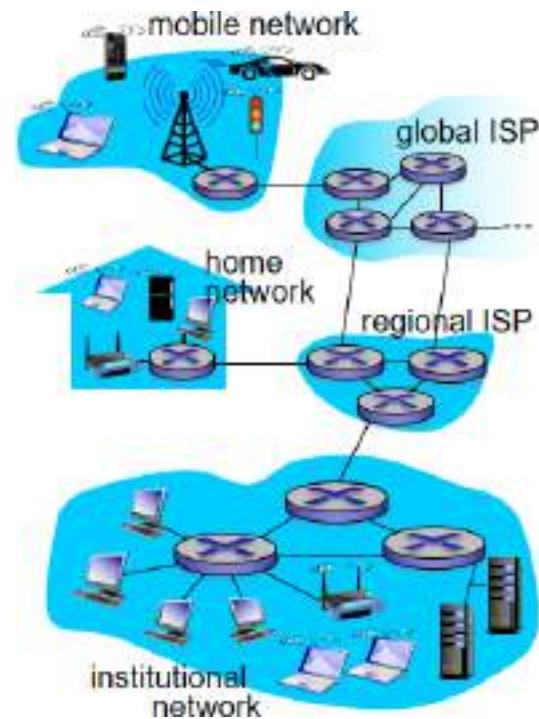
- *Ved kontakt, skriv til begge!*
- *Men begræns kommunikation til det nødvendige (200 mod 2)*

Idag

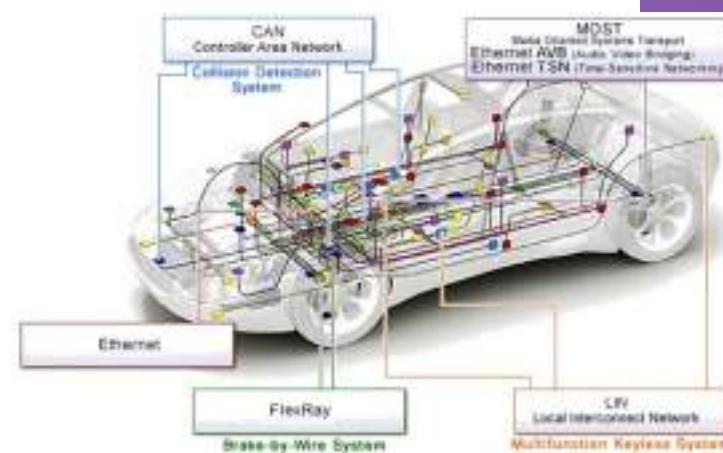
- Kursus indhold og mål
- Web-arkitekturen
- JavaScript Intro

Kursus indhold og mål

Computer Netværk: Et system af computere, som er forbundne af kommunikationskanaler, så de kan udveksle data [webster]



- Internet, Intranet
- Personal Area Networks
- I data centre
- Clouds: applikationer bruger virtualiserede netværk
- Telenettet
- Trådløse sensor netværk / IoT
- Biler, Fabrikker (SCADA),
- Smart Homes,



Betyder netværk noget for app-brug og -udvikling??

- Konstruktion af netværksbaserede applikationer kræver at de tager højde for:
 - Følge aftalte protokoller, og gøre brug af netværksservices
 - Netværk er LAAAAANGSOMME (og være meget varierende)!
 - Fejl i netværk og servere, som gør dem utilgængelige
 - Mange sikkerhedsproblematikker stammer fra netværk og trusler fra nettet.
 - Kommunikation er meget energi-krævende (batteridrevne enheder)
- Opdeling af applikation på flere maskiner er kvalitativt forskelligt fra enkelt-maskine applikationer:
 - Data ligger fjernt; Ingen delt hukommelse; ingen delte ure / ingen præcis fælles tid; del af app kan ”crashe” mens andet kører videre
 - Særlige teorier og metoder: **”distribuerede systemer”**

Protokoller

- Hvis 2 maskiner eller programmer skal kunne snakke sammen over et netværk, skal de følge bestemte aftalte regler.
 - Hvordan skal partnerne fortolke en bit-strøm som kommer på nettet?
 - Hvordan skal partnerne svare på hinandens beskeder?
- Et sådant **regelsæt kaldes en protokol**. Maskiner er ikke så medgivende som mennesker



<https://www.ibtimes.sg/trump-never-bowed-japan-emperor-no-protocol-blunder-either-194181>

Netværkshastighed

```
C:\Users\bniel>ping www.diku.dk
Pinging diku.dk [130.226.237.173] with 32 bytes of data
Reply from 130.226.237.173: bytes=32 time=33ms TTL=249
```

Latency ~ Roundtrip time/2:
Ping www.cs
Ping diku.dk
ping google.com
ping www.stanford.edu
ping www.doshisha.ac.jp
ping cnic.com.cn

Site	Round Trip Time (ms)	Download (Mbit/sec.)	Upload (Mbit/sec.)
DTU https://www.speedtest.net	8	366	938
Stockholm (Sweden) - EDIS.at *)	18	354	231
Frankfurt (Germany) - VULTR.net *)	27	293	53
New York (United States) - VULTR.net *)	110	56	2
Sydney (Australia) - VULTR.net *)	336	2.6	0.2

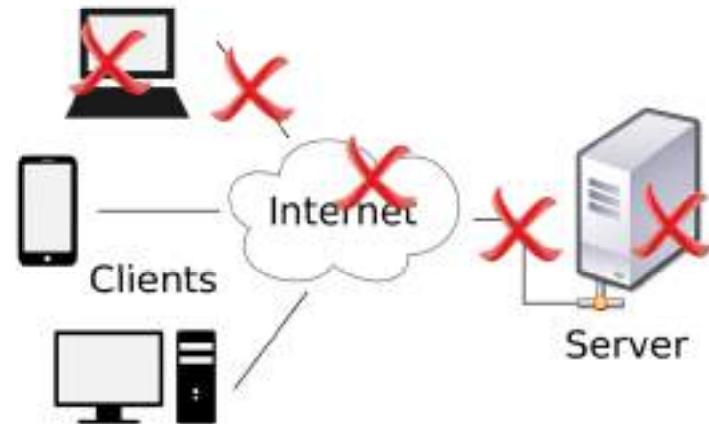
Kommunikation på netværk er LAAANGSOMME (ifht. internt i et program eller på maskinen)

Kald og retur af simpel C-funktion tager <<< 1 micro Sec. Fx 100ns, kan der foretages 10 Millioner kald pr. sekund!!!
Hvis funktionen er placeret på DTU kan der laves $1000\text{ms}/8\text{ms} = 125$ kald per sekund, og Sidney 3!!!!!!!

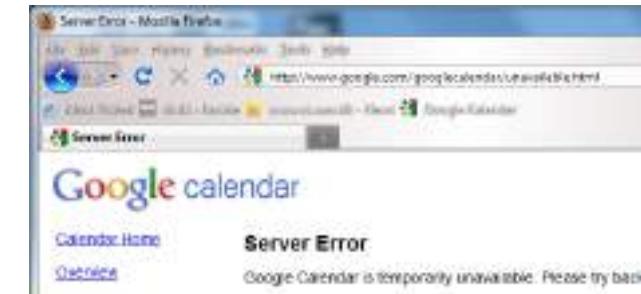
*) <https://www.meter.net> (Appears to use web-sockets) **) <https://perfops.net/ping-from-china>

Netværket: Fejl og fejl-tolerance

- Netværk kan blive (midlertidigt) afbrudt!
- Server(e) kan gå ned kortere eller længere tid!
- HÆNDELSER, DER SKER TILPAS OFTE TIL, AT DE IKKE KAN IGNORERES
 - Murphy's 1. lov. Kan det gå galt, så vil det gå galt!
 - Murphy's 2. lov: Det går altid galt på det mest kritiske tidspunkt
- **Net-Applikationer skal bygges til at tolerere disse hændelser**



- Rendegravere,
- Bugs i server SW el. i klient SW, i netværksudstyr
- Hardware fejl på klient/server/netværksudstyr
- Strømnedbrud
- Software fejl i datacenter,
- Fejl-konfiguration af HW/SW
- Planlagt nedetid til vedligehold
- ...



Sikkerhed!

- Med netværksforbindelse er vores klienter og servere åbne for ukendt trafik
- Giver adgang for brugere som bevidst udnytter fejl og sårbarheder til at
 - udrette skade
 - opnå gevinst
 - sjov
 - forbedre sikkerhed



Black-hat, grey-hat, white hat hackers

Motivation for kurset 1)

- ***Netværk (herunder Internettet og web) er så stor en del af IT-professionelles og udvikleres arbejde, at alle bør have en fundamental forståelse for dets virkemåde, styrker og svagheder***
 - Vi bruger det hele tiden: Intranet, web, soc.-medier, personlig netværk, online møder og samarbejde, ...
 - Mange moderne applikationer bruger Internet- og web-teknologi
 - Opdeling i del-systemer og kommunikation mellem disse, fx opdeling på Client/Server i Front-end og backend

Motivation for kurset 2)

- Faglighed: **Distribuerede systemer**
 - IoT, Cloud Computing, parallel og cluster beregning, sensor-netværk, Service Orienterede Arkitekturer, Block chains,...
 - Kurser: Computer arkitektur og operativ systemer, distribuerede systemer, cyber-physical systems.
 - Specialiserings-retning
 - Sammenhæng mellem applikation, system-software og hardware: Systemer der
dur: God effektivitet, Hastighed, Sikkerhed, og Robushed for fejl-situationer
- Viser **gode ”datalogiske” principper** for opbygning af komplekse systemer:
 - lagdeling,
 - protokoller,
 - client-server model

Kursusmål

- Forstå opbygning og virkemåde af computer netværk / Internet
- Forstå gængse Internet protokoller: HTTP, DNS, TCP, UDP, IP
- Forstå og konstruere *simple* programmer, der anvender Internettet
 - Simple Web-sider og Simple web-applikationer (front-end og back-end)
 - Lære *tilstrækkeligt* JavaScript til at lave disse
 - Fokus på grund-teknologier og metoder
 - Web-services og API'er
- **Baggrund:** Relevans for projekter ⇒ Gøre det praktisk muligt at bruge netværk i projekterne ved brug af "moderne" metoder: web-programmering

Udfordringer

- ”kun” 5 ECTS til både Internet og Web-programmering!
- Stor variation i jeres programmerings-erfaring
 - Nogle har lige (knap) lært programmering i C
 - Nogle er velbevandrede i moderne sprog (ex. python)
- Stor variation i jeres viden om web-teknologier
- Nogle fagligheder introduceres først senere!!
 - Placering allerede på 2. semester giver visse begrænsninger
 - OOP+OOAD og DGB kommer på 3. semester, DB på 5. semester.
- Netværksteori før eller efter web-programmering?
- **Begrænsninger**
 - Web programmering er inkluderet for at facilitere praktisk udvikling af netværksbaserede applikationer i projekterne: i disse tider ”web-apps”
 - JavaScript er et redskab hertil, ikke et selvstændigt mål
 - I bliver ikke ”færdigt” uddannede web-udviklere!
 - Et kæmpe teknologi- og metode-kompleks
 - Ej verdensmestre i JavaScript



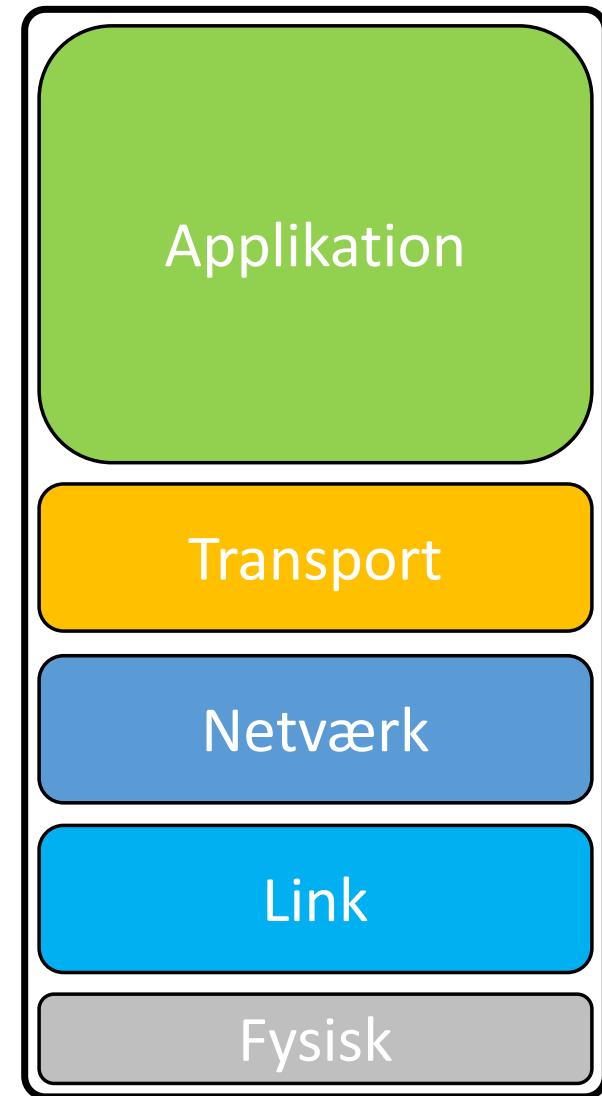
“Større Program udviklet af en gruppe” jfv. semester tema

- Fylder flere filer
- Flere udviklere - en gruppe
- Baseret på eksisterende kode, du ikke selv har skrevet!
 - Læse, forstå, anvende, udvide andres kode
- Afgang forud forudt ud på flere maskiner: klienter og server
- Flere sprog, teknologier, program biblioteker

Lektionsplan: Top-down tilgang

1. Introduktion, og teknologi-oversigt, Intro til JS
2. Funktioner og Objekter i JS
3. HTML, Inputs & forms, klassiske dynamiske web-sites, grundlæggende HTTP
4. Øvelsesgang
5. Klient-side programmering: DOM, events, fetch
6. Undtagelser og asynkrone programmer, fetch & promises
7. Server-side programmering: Node (HTTP, streams), sessions, cookies
8. Server-side programmering: Web-api's, REST
9. Introduktion til Inter-netværk
10. Applikationslagsprotokoller, HTTP, Programmering med Sockets
11. Transportlaget,, principper for pålidelig kommunikation)
12. TCP: Sliding windows, Congestion Control, IP-adresser
13. Sikkerhedsprotokoller (TLS/SSL/HTTPS)

5-lags model for
Internet protokol stakken



Lektionsplan: Top-down tilgang

1. Introduktion, og teknologi-oversigt, Intro til JS
2. Funktioner og Objekter i JS
3. HTML, Inputs & forms, klassiske dynamiske web-sites, grundlæggende HTTP
4. Øvelsesgang
5. Klient-side programmering
6. Undtagelser og array
7. Server-side programmering, cookies
8. Server-side programmering
9. Introduktion til Internet
10. Applikationslagsprotokoller (HTML, HTTP, Programmering med Sockets)
11. Transportlaget,, principper for pålidelig kommunikation)
12. TCP: Sliding windows, Congestion Control, IP-adresser
13. Sikkerhedsprotokoller (TLS/SSL/HTTPS)

Vi starter med JS og web af hensyn til **projekterne!**

For tidligt eller forsent dilemma.

5-lags model for
Internet protokol stakken

Applikation

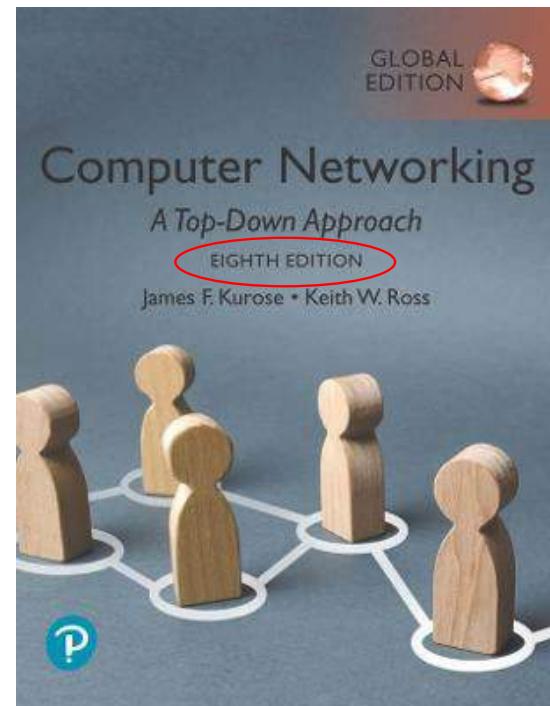
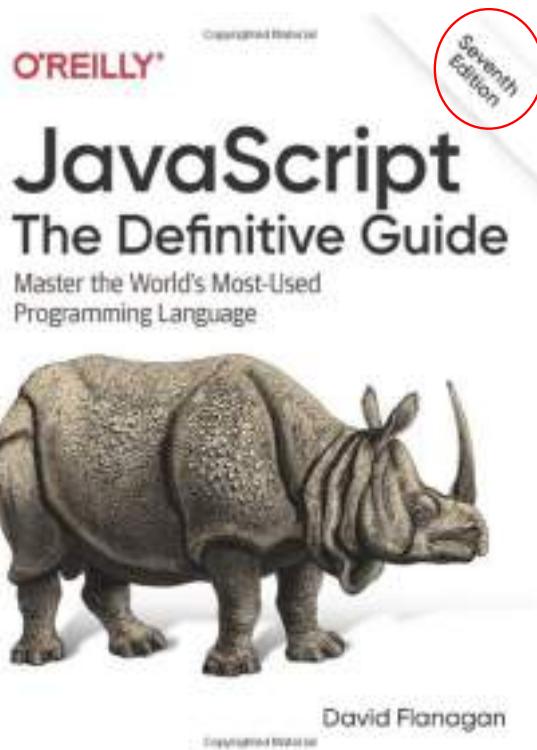
Transport

Netværk

Link

Fysisk

Primær Litteratur (Og ja, det er meningen at I skal læse det!)



Diverse Web ressourcer

- <https://www.oreilly.com/library/view/javascript-the-definitive/9781491952016/>
- <https://github.com/davidflanagan/jstdg7>

https://gaia.cs.umass.edu/kurose_ross/about.php

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

Tid I forventes at bruge

- $5 \text{ ECTS} =_{\text{def}} 5 * 30 \text{ timer} = 150 \text{ timer studieaktivitet}$
- 12 lektioner á 2 timer = 24 timer
- 12 øvelsesgange á 2 timer
1 øvelsesgang á 4 timer = 28 timer
- 4 timers læsning til hver lektion $\sim= 48$ timer
- Resterende øvelser hjemme = 20 timer
- Eksamens-repetition = 30 timer
- I alt 150 timer i alt

Når du ikke har lest til eksamen
men du møter opp og satser på det beste



Eksamensplan

Eksamensplan: Skriftlig

- I samme stil som øvelserne, og de større praktiske opgaver
 - Noget multiple-choice,
 - ”regne-opgaver”,
 - en større praktisk programmeringsopgave
- Intern censur
- Gennemføres som moodle quizz.
- Mere specifik info senere

Kurset kører primært fysisk!

- Deltag aktivt i øvelserne, i grupper
 - Viden deling
 - Fælles fokus
 - For høje (uindfriede) forventninger til egen arbejds-effektivitet hjemme)
- Hjælpelærere er til rådighed i opgaveregningstiden!
- Der bygges meget videre på hver lektion – hæng i!
- Optagelser: Inden eksamen kan I ikke nå at gense forelæsningerne alligevel.

Dagens lektion

Dagens opgaver

- Installation og grundlæggende brug af værktøjer:
 - Browser, VSCode editor, Node.js
- JS programmeringsøvelser
 - Grundlæggende variable og typer,
 - Basale kontrol strukturer (if-then-else; for; while)
 - Behandling af strenge

Internetværk og Web-programmering

Introduktion til Web Teknologier

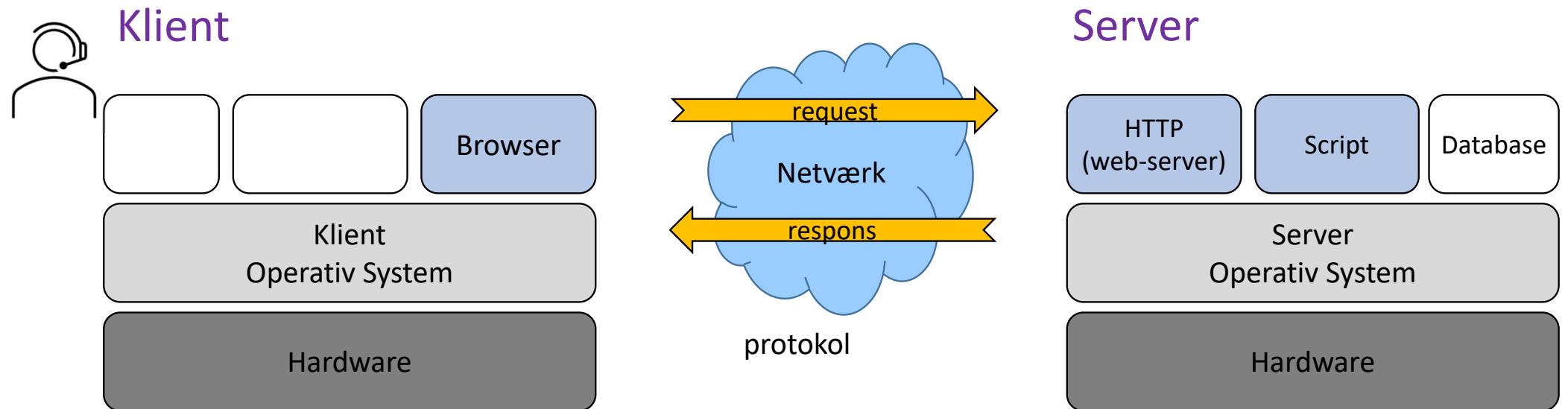
Klient-side og server-side teknologi

Forelæsning 1
Brian Nielsen

Distributed, Embedded, Intelligent Systems



Simpel web klient-server arkitektur

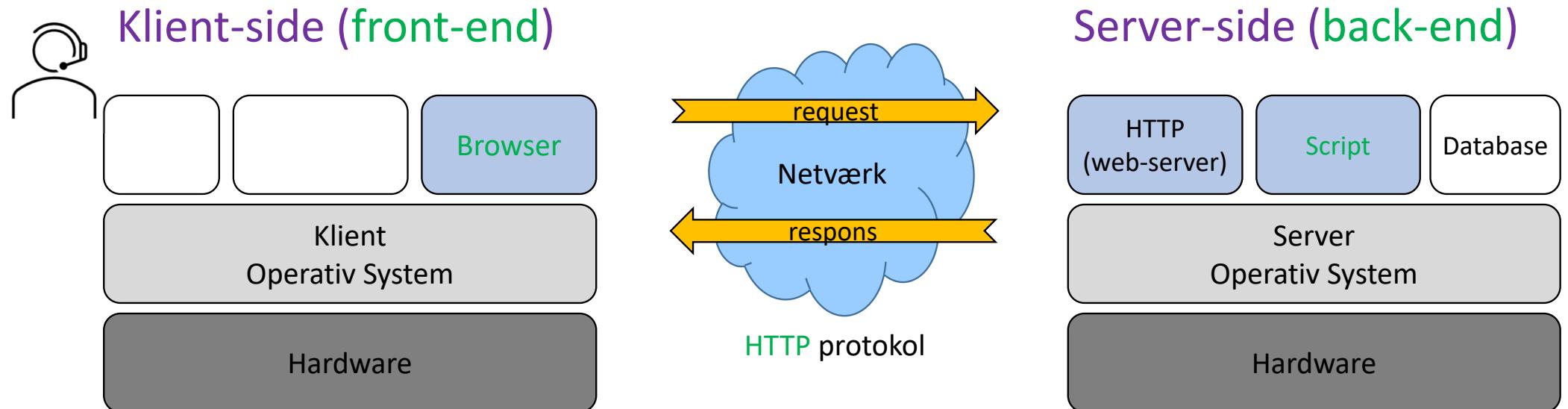


- Klient maskine: PC, mobil, tablet
 - Anvender service ved at sende anmodning (request) til server og præsentere svaret (respons).

- Server, særlig maskine
 - Always-on, offentlig IP, kraftig, ofte i datacenter
 - Klar til at modtage requests,
 - Normalt headless (uden gui), ssh terminal adgang!
 - Fx, Web-server, fil-server, print-server, ...
 - (Men kan også give adgang til data fra "smart devices", fx lys- og varme-styring)

Simpel web klient-server arkitektur

- WEB-applikation: opdelt i front-end + back-end vha. web-teknologier



- Afvikles typisk i en browser
- Indlæses via en web-side
 - (Men kan være andre programmer, som "snakker" HTTP)
- "UI" og nogle funktioner

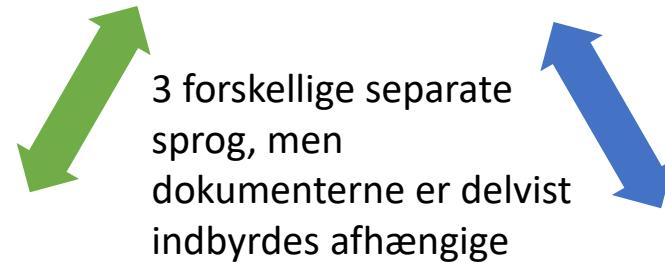
- Implementerer server-siden af en applikation
 - Gemmer tilstand, foretager optagninger og beregninger
- Server leverer information til klienten
 - HTML dokumenter: statiske filer, eller dynamisk genereret
 - Data-objekter (JSON)
 - Resultat af "beregninger"

Front-end tre-enigheden: HTML5, CSS, JavaScript

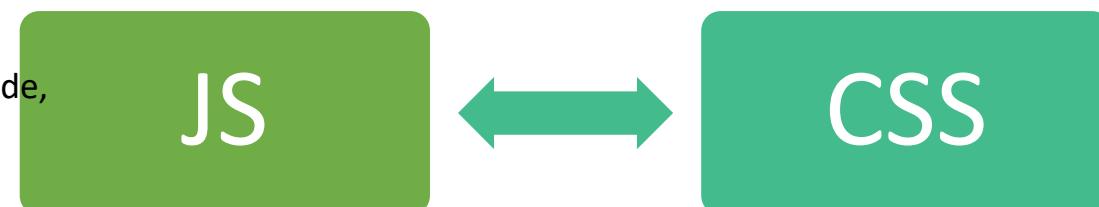


HTML beskriver
indhold og dets struktur, opbygning
i bestanddele

"semantic markup"



Javascript giver
Dynamisk optegning af side,
Dynamik, Interaktion,
Animation, UI,
input validering,
funktioner,
Dynamisk data-indlæsning fra server ...



Cascading Style Sheets
Definerer dokumentets præsentation: Udseende, layout, farver, fonts,...

Samme html dokument kan have flere udseender,
fx afhængigt af

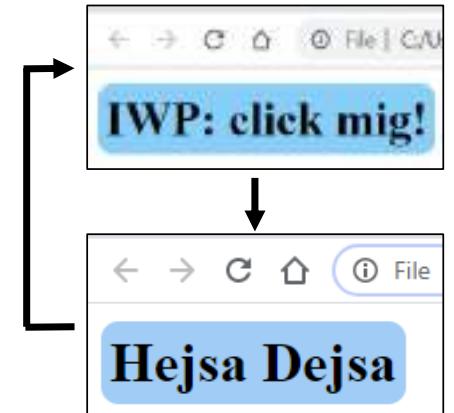
- Skærmstørrelser
- Brugeres/udvikleres præferencer

Front-end tre-enighed:

```
<!DOCTYPE html>
<html lang="da">
  <head>
    <title>IWP DEMO</title>
    <link rel="stylesheet" href="style.css">
    <meta charset="UTF-8">
  </head>
  <body>
    <h1 id="iwp_id"> IWP: click mig! </h1>
    <script src="demo.js"></script>
  </body>
</html>
```

HTML5

3 forskellige separate sprog, men dokumenterne er delvist indbyrdes afhængige



```
let toggleState=true;
let oldText="";
function doToggle(event){
  let elem=event.target;
  if(toggleState){
    oldText=elem.textContent;
    elem.textContent="Hejsa Dejsa";
    toggleState=false;
  } else {
    elem.textContent=oldText;
    toggleState=true;
  }
}
let h1Elem=document.querySelector("#iwp_id");
h1Elem.addEventListener("click", doToggle);
```

JavaScript

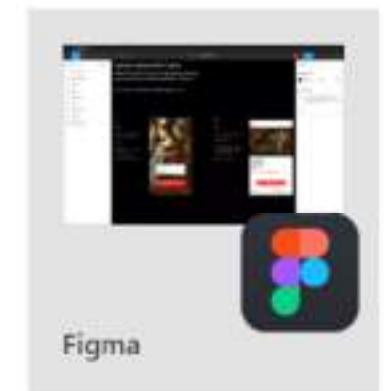
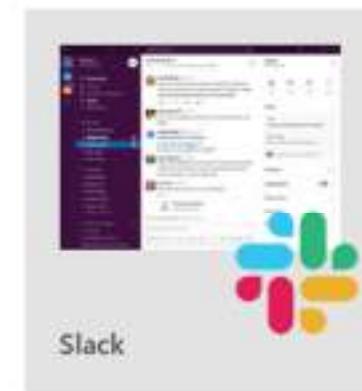
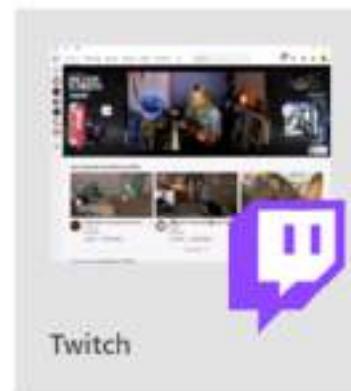
CSS

```
h1 {
  text-align: left;
  width: fit-content;
  height: auto;
  padding: 5px;
  margin-top: 10px;
  border: none;
  border-radius: 10px;
  background-color: #lightskyblue;
}
```

(JS til desktop Applikationer)



- Build cross-platform desktop apps with JavaScript, HTML, and CSS
 - <https://www.electronjs.org/>

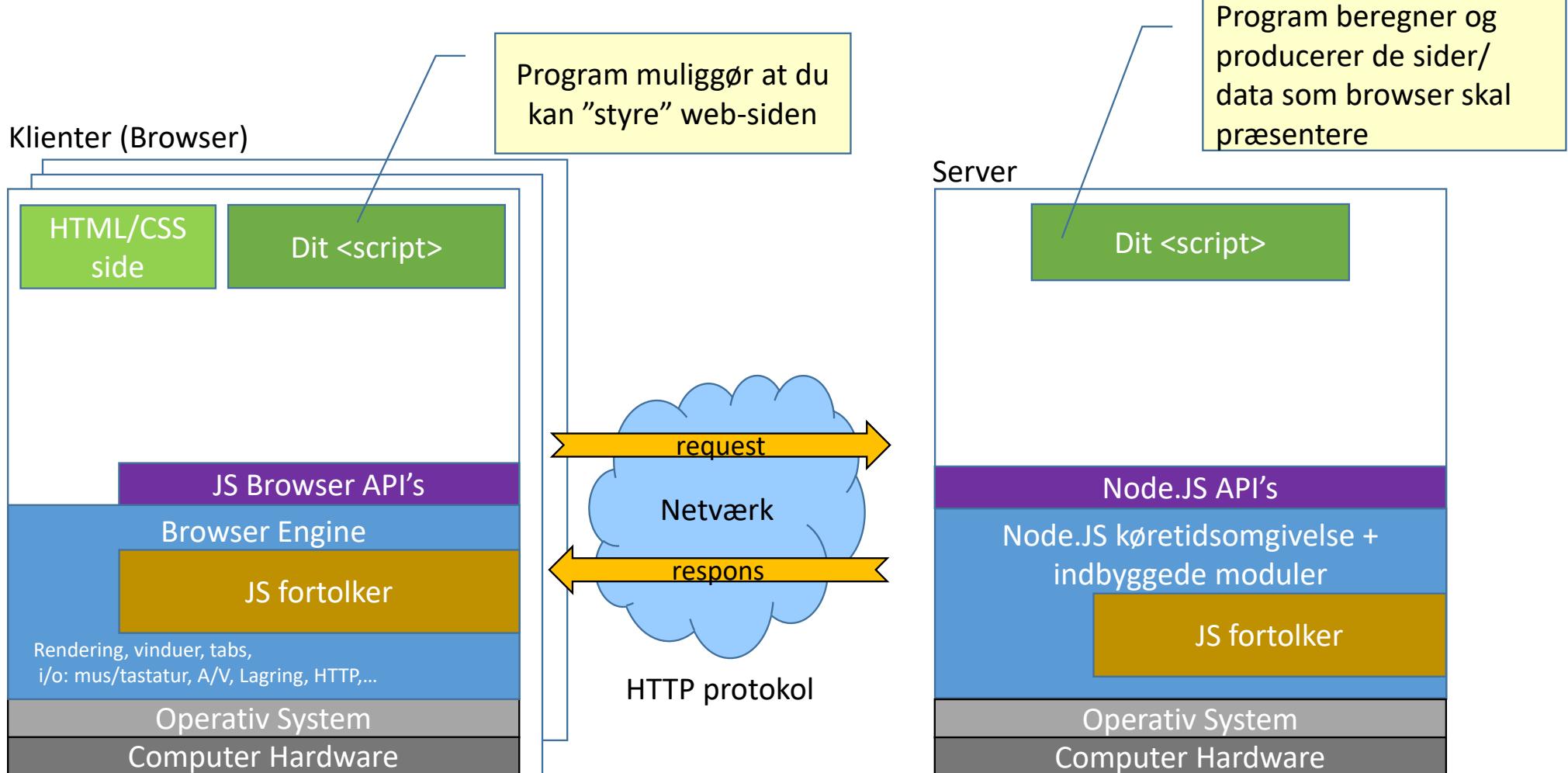


Back-end (server-side) teknologier

- ”Development stack”: Samling af sprog, biblioteker, værktøjer, databaser, designet til at arbejde sammen.
- Mange muligheder
 - LAMP (Linux-Apache-MySQL-PHP)
 - Den klassiske, velkendt, arbejdshesten bag mange web-sites.
 - WISA (Windows-IIS-Sql-ASP.net)
 - C# (.net) programmer på windows server
 - Django Stack: Python-Django-Apache-MySQL
 - Ruby-on-rails
 - Java-servlets: Java-Tomcat
 - **JavaScript/node.js**
 - MEAN (MongoDB-Express-Angular-Node) *)
 - MERN: ”React” framework i stedet for ”Angular”.
 - MEVN: ”Vue” framework i stedet for ”Angular”.
 - SvelteKit
 - ...

*) Angular, React, Vue, SvelteKit er front-end frameworks til udvikling af avancerede bruger grænseflader/apps.

Overordnet arkitektur i IWP



Internetværk og Web-programmering

Introduktion til Web Teknologier

JavaScript

Forelæsning 1
Brian Nielsen

Distributed, Embedded, Intelligent Systems



Javascript

- Oprindeligt tiltænkt mindre opgaver i web-sider, hvor Java var for tungt og klodset.
 - Introduceret i Netscape browser, 1996
- JavaScript, Mocha, LiveScript, JScript, **ECMAScript**,
- ECMA: standardiseringskommitté
 - 6th Edition - ECMAScript 2015 henviser til en bestemt standardiseret version
- **JavaScript ≠ Java**: Meget forskellige sprog
 - Fuldstændigt “rigtigt” programmeringssprog (*fortolket, dynamisk, svagt typet*)
 - The good, The Bad, The Ugly
 - Understøtter imperativ, funktionsorienteret, og tildels objekt-orienteret programmering
 - **Vi kan langt hen ad vejen klare os med en “imperativ” stil**
 - Under stadig forbedring og udvikling:
 - Især mange forbedringer fra ES6, 2016: “Modern JavaScript”
 - Nyeste “ECMAScript 2023” https://en.wikipedia.org/wiki/ECMAScript_version_history

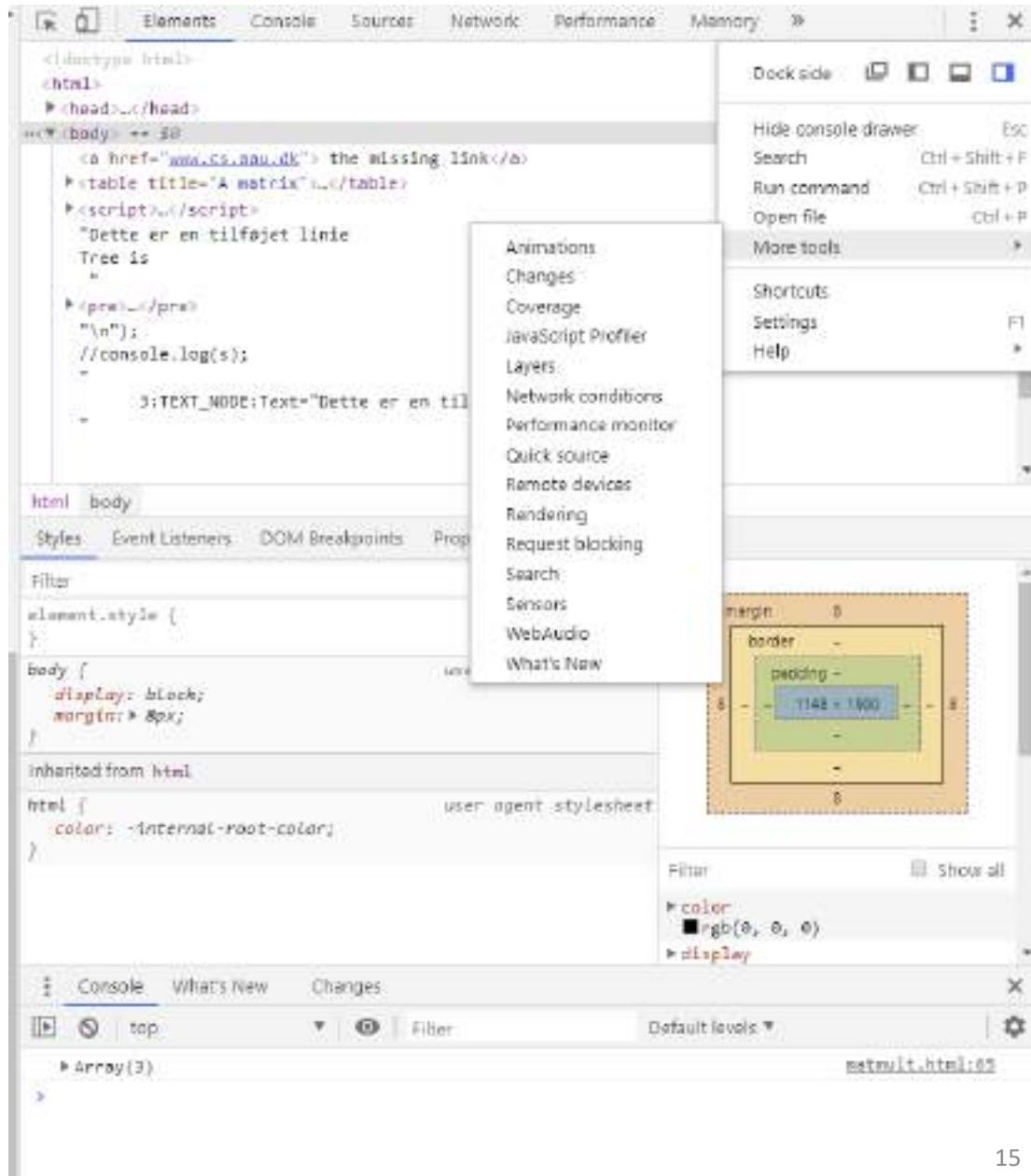
JS i browsere

Findes som "Ctrl+Shift+i" i chrome

→ Værktøjer → Developer Tools

- Inspektion og ændring af HTML og CSS
- Konsol
- Write-eval-print loop
 - Skrive og afvikle code-snips
- Debugger
- Optimering
 - Hastighed
 - Hukommelse
 - Brug af netværk
 - Profilers (flaskehalse)
- Dækningsmåling (Test)

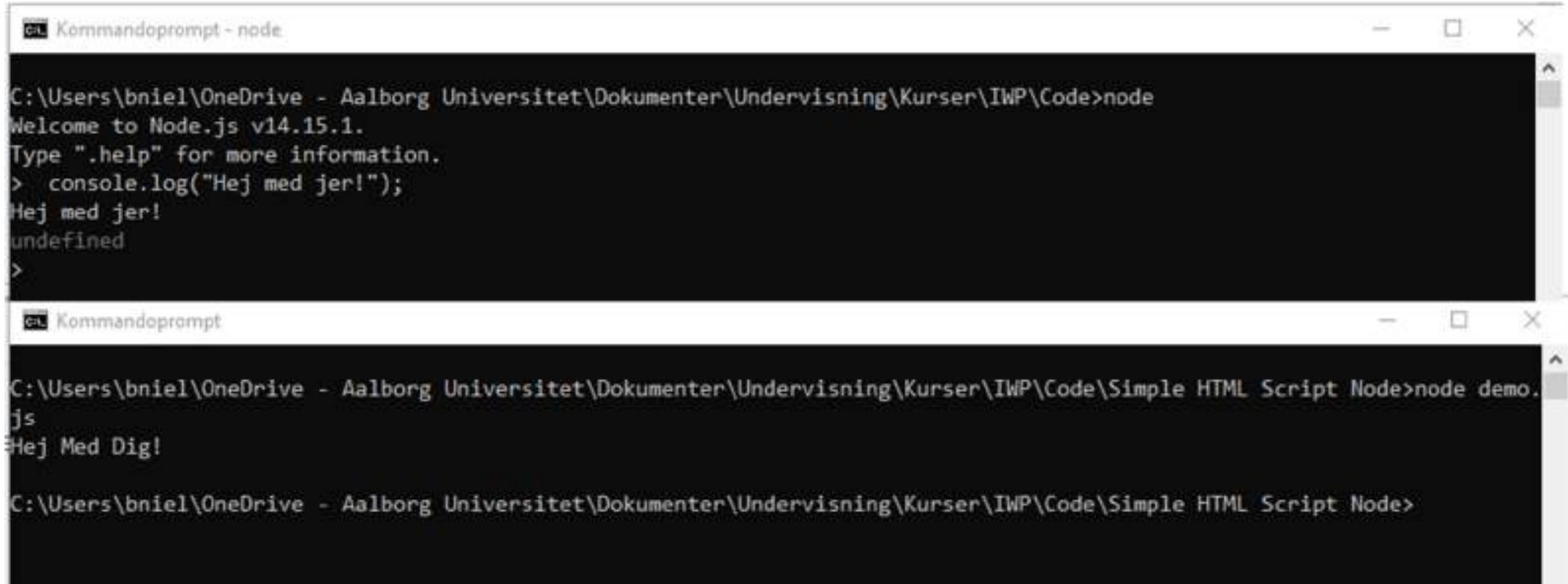
<https://developers.google.com/web/tools/chrome-devtools>
<https://javascript.info/debugging-chrome>



Hvad bruges JS til på front-end (klienter)

- 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 "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 "bekvemt" kan foretage lokalt
 - Funktioner den skal varetage, hvis server er "nede"

Back-end: Node.js



The image shows two separate Command Prompt windows. The top window is titled 'Kommandoprompt - node' and contains the following text:

```
C:\Users\bniel\OneDrive - Aalborg Universitet\Dokumenter\Undervisning\Kurser\IWP\Code>node
Welcome to Node.js v14.15.1.
Type ".help" for more information.
> console.log("Hej med jer!");
Hej med jer!
undefined
>
```

The bottom window is also titled 'Kommandoprompt' and contains the following text:

```
C:\Users\bniel\OneDrive - Aalborg Universitet\Dokumenter\Undervisning\Kurser\IWP\Code\Simple HTML Script Node>node demo.js
Hej Med Dig!
```

- Afvikling af JS programmer
- Kan sættes op til at fungere som **web-server**, og håndtere requests i JS

Hvad bruges JS til på back-end (server)

- Det samme som andre server scripting sprog
 - Modtagelse af HTTP requests fra klienter
 - Dekodning af parametre og formularer
 - Validering af modtagne data
 - Applikations funktioner og beregning (som indgår i svaret til klienter)
 - Opslag i og opdatering af databaser eller filer
 - Generering af dynamiske HTML sider
- Implementation af WEB-APIer
- Servering af filer
- Håndtering af data og funktioner, som skal være delte mellem forskellige klienter

Visual Studio Code - Intellisense

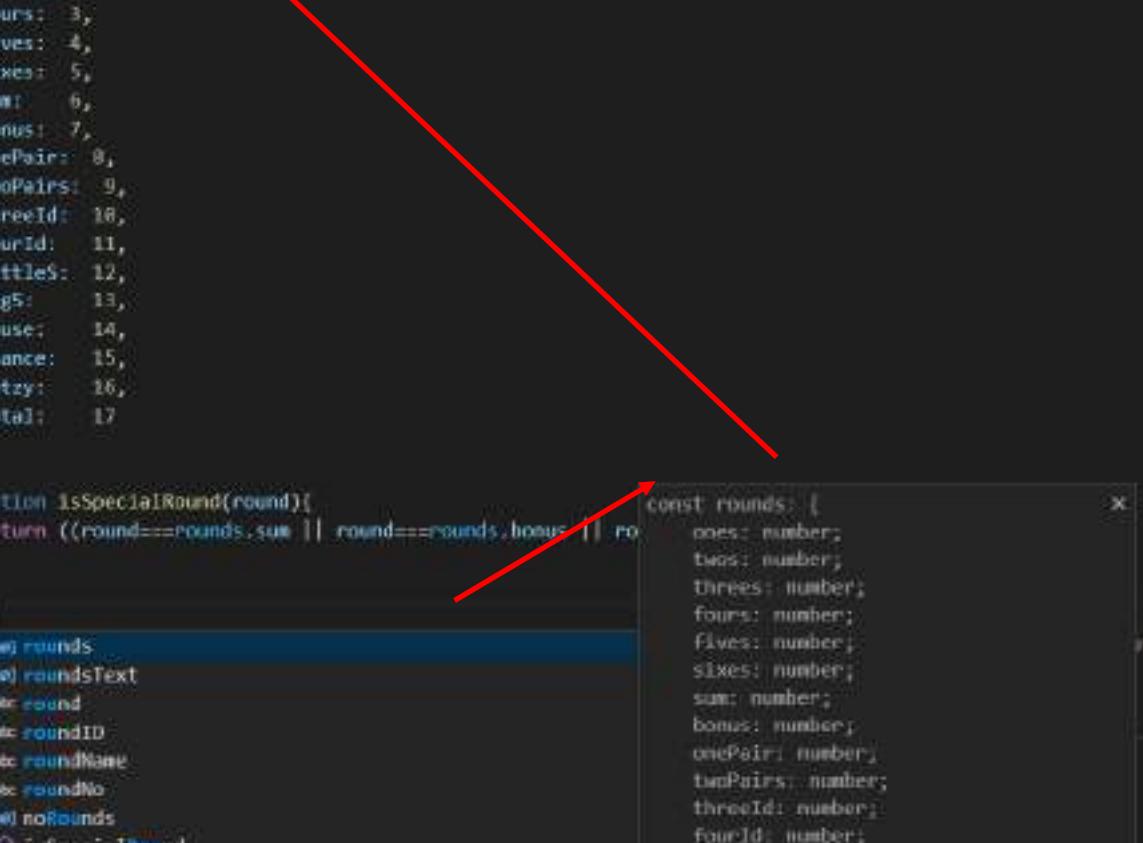
Avanceret editor med

- Syntax highlighting
 - Code-completion
 - Assistanse og hints
 - En stor hjælp til sprog som JS

Muligheder for fuldendelse af "rou"

Viser hints om erklæringen af den valgte fuldendelse

```
17 const rounds={ //as C-enum doesn't directly exist in JS, we simulate it using an object
18   ones: 0,
19   twos: 1,
20   threes: 2,
21   fours: 3,
22   fives: 4,
23   sixes: 5,
24   sum: 6,
25   bonus: 7,
26   onePair: 8,
27   twoPairs: 9,
28   threeId: 10,
29   fourId: 11,
30   littleS: 12,
31   bigS: 13,
32   house: 14,
33   chance: 15,
34   yatzy: 16,
35   total: 17
36 };
37
38 function isSpecialRound(round){
39   return ((round==rounds.sum) || (round==rounds.bonus) || no
40 }
41
42 round
43 const rounds:
44   *{0} roundsText
45   1; *{0} round
46   *{0} roundID
TERMINAL
Windows PowerShell
Copyright © Microsoft Corporation. All rights reserved. This product includes
Try the following:
    • isSpecialRound
    • playRound
    • runInContext
PS C:\Users\user> resourceUsage
PS C:\Users\user> runInNewContext
```



A screenshot of a terminal window showing a copy operation. The terminal shows a command-line interface with several options listed. A red arrow points from the word 'round' in the code editor above to the 'round' variable in the terminal command. Another red arrow points from the 'const rounds:' declaration in the code editor to the 'rounds' part of the terminal command.

Visual Studio Code - debugger

- Indsættelse af break-points, inspektion/ændring af variable, ...

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** demo.js - demo1 (Workspace) - Visual Studio Code.
- Left Sidebar:** Launch Program dropdown, Variables (Local), Watch, Call Stack (Paused on Break).
- Code Editor:** A simple Node.js script named demo.js:

```
'use strict';
//SEE: https://javascript.info/strict-mode
console.log("Hej Med Dig!");
```
- Bottom Status Bar:** Line 6, Column 1, Spaces: 4, UTF-8, CRLF, JavaScript.
- Bottom Right:** Page number 20.

The code editor shows a yellow bar highlighting the line `console.log("Hej Med Dig!");`. The terminal below shows the output of the script: `C:\Program Files\nodejs\node.exe ./demo.js` followed by `Hej Med Dig!` at line 4.

BMI-Tracker v1: som “konsol applikation”

```
eb-BMIV2 - CONSOLE - EXTENDED> node .\node\app.js.js
Enter Name Mickey Mouse
Enter Height 180
Enter Weight 114
Welcome New User
Output is in file bmiStatus.html
Enter Name Mickey Mouse
Enter Height 180
Enter Weight 125
Output is in file bmiStatus.html
Enter Name []
```

BMI Status of Mickey Mouse

Your BMI is 38.58. Since last, it has changed 3.39.

Weight	BMI	Delta
114	35.19	0
125	38.58	3.39

[Get Help](#)

BMI-Tracker v2: som “web-site”

The screenshot shows a web browser window with the URL 127.0.0.1:3000. The main title is "IWP BMI-recorder". Below it, there is a form for "Personal information" with fields for Name (Mickey Mouse), Height (cm) (180), Weight (kg) (118), and Age (years) (120). There are three radio button options for gender: Female, Male, and Indifferent, with "Indifferent" selected. A "Record" button is at the bottom right of the form.

Personal information:

Name Mickey Mouse

Height (cm): 180

Weight (kg): 118

Age (years): 120

Female

Male

Indifferent

Record

The screenshot shows a web browser window with the URL 127.0.0.1:3000. The main title is "IWP BMI-Statistics-tracker". Below it, there is a form for "Personal information" with fields for Name (Mickey Mouse) and a "Get Stats" button. At the bottom, there is a link labeled "Get Help".

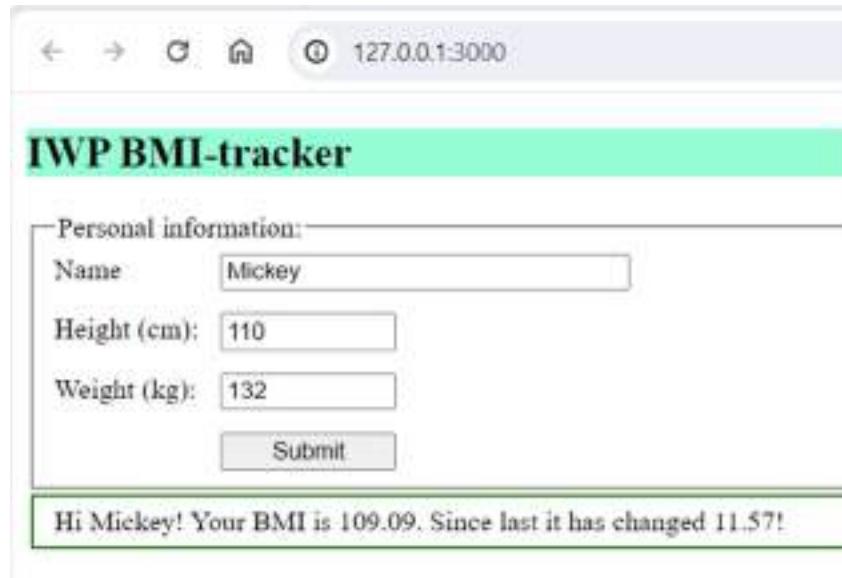
Personal information:

Name Mickey Mouse

Get Stats

[Get Help](#)

BMI-Tracker v3: som “web-app”



← → ⌛ ⌂ 127.0.0.1:3000

IWP BMI-tracker

Personal information:

Name: Mickey

Height (cm): 110

Weight (kg): 132

Submit

Hi Mickey! Your BMI is 109.09. Since last it has changed 11.57!

Lidt intro til JS sproget

Eksempel

```
/* Lidt JS INTRO */
let beer="Tuborg, tak!";

const buyBeer="Køb "+beer;

let menus=["Opret øl", "Slet øl", buyBeer];
for(let i = 0;i<menus.length;i++) console.log(menus[i]);

for(let menu of menus) console.log(menu);

function addButton(title){
    const b=document.createElement("button");
    b.append(title);
    document.body.append(b);
}

for(let menu of menus) addButton(menu);

addButton(sodavand); //ERROR

let sodavand; //Missing init

addButton(sodavand);

if("6"==6) console.log("true"); else console.log("false");
if("6"==6) console.log("true"); else console.log("false");
if(Number("6")===6) console.log("true"); else console.log("false");
```

Variabel Erklæringer - Bindinger

```
let x=0; // der laves en binding mellem navnet x og værdien 0
let y; //uden initialisering bindes til den specielle værdi undefined
console.log(y); //undefined

y=x+1; //y bindes til resultatet af udtrykket x+1
console.log(y); //1

const minDice=1; //en konstant
```

- Brug nøgleordet **let** til at definere en "binding" (oprette variabel)
- En **const** forbliver bundet til samme værdi
- Bemærk: ingen eksplisitte type erklæringer!!
- Typer udledes på køretid.
- Primitive typer: Tal, Booleans, Strenge

Lidt om Strenge

```
Let roundName = "Yatzy";
let cell1=<td class="left-text">" + roundName + '</td>';
let cell2= `<td class="left-text"> ${roundName}</td>`;
//<td class="left-text">Yatzy</td>
cell1[4]; // "c"
cell1.includes("text"); //true
let textStart=cell1.indexOf("text"); //16

let smiley="\u263A"; //☺
```

- En streng er indbygget type i JS (ikke som i C et array)
- Både " " og single tick '' angiver en streng værdi (bør anvende samme stil i samme program)
- Special tegn escapes med \
- + giver en konkatenering
- Back-tics `` angiver en **template litteral**
 - kan indeholde pladsholdere til program genereret tekst i `${expr}`
 - kan deles over flere linier, og bruge visse specialtegn uden escape af fx " (tegnene \${} skal dog)
- Tegn kodes i "UTF-16"-format
 - "Teknikalitet": nogle eksotiske tegn kræver 2 koder ('A\uD87E\uDC04Z' er 4 koder lang, men giver 3 tegn tekst A𠮷Z)
- Et enormt arsenal af metoder på streng værdier, se pensum

Typer og type konvertering

- Hvis operanders type ikke matcher, forsøger JS at lave en meningsfyldt type konvertering
 - Fx: `+` mellem tal og streng giver en `+ (konkatenering)` mellem tallet som streng
- Særlige funktioner findes til eksplisit konvertering
- !!Brug normalt `==` til sammenligning, checker for matchende type og værdi
- Operatoren `==` sammenligner værdier efter typekonvertering ("deprecated")

```
let x=0;
let s="Hej";
s=s+x;
console.log(s); // "Hej0"

let a;      // undefined
let b=a+1; // får den særlige værdi NaN ("not a number")

// eksplisit konvertering
x=Number("6"); // heltalsværdien 6
s=String(7);   // strengværdien "7"

if( 6 == "6" )    // true
  console.log("true");

if(6==="6")       // false: types doesn't match
  console.log("true");
else
  console.log("false");
```

Køretidsfejl

- Et JS program oversættes ikke, men fortolkes
- Kun grove syntax fejl findes inden programmet startes, fx manglende } eller "
- **Smart:**
 - hurtig udvikling,
 - kørsel af ufærdige programmer
 - Interaktive "skriv-og-evaluér"
- **Ulempe:**

```
let roundNo=0;  
let q=roundno;      //Køretidsfejl: Reference Error: roundno is not defined
```

- Programmet stopper med TRÆLSE fejl, fx som følge af simple stavefejl
- Eller kører videre med "undefined" værdier
 - IntelliSense hjælper dog lidt
 - [ESLint analyse værktøj](#) (kræver installation af værktøjet og evt. extension til VSCode)

Et fragment fra Multi Yatzy V1 (C-lign. JS)

- En terning er et tal 1..6
- Et ”kast” er et array af terning-værdier
- Bemærk funktions-erklæring
- Bemærk at arrays er dynamiske!
 - Elementer oprettes som de indsættes!
- JS har mange andre former for iteration ud over `for`, `while`

```
const minDice=1;
const maxDice=6; //min and max value of a normal dice

//returns an array that represents the outcome of rolling M dice
///e.g diceRoll [1,6,5,5,2]
function roll(M){
    let diceRoll=[]; //empty array
    for(let i=0;i<M;i++){
        diceRoll[i]= random(minDice,maxDice);
    }
    return diceRoll;
}
```

Internetværk og Web-programmering

JS Funktioner & Objekter

Forelæsning 2
Brian Nielsen

Distributed, Embedded, Intelligent Systems



Agenda: JavaScript Intro 2

- Lektion 1: Grundlæggende

- Bindinger
- Iteration, selection
- Basic Arrays, Strings



- Lektion 2

- Funktioner
- Funktions-parametre
- JavaScript Objekter
- Constructor funktioner
- Objekt serialisering
- Array metoder
- Videregående om objekter, prototype, og klasser

```
let diceRoll=[1,6,6,2,3,4,6];

function count6es(roll){
    let found6=0;
    for(let i=0;i<roll.length;i++){
        if(roll[i]===6) found6++;
    }
    console.log(`found ${found6} sixes!`);
    return found6;
}

console.log("Found6: "+count6es(diceRoll));

found 3 sixes!
Found6: 3
```

Evt.

“Found6”+String(count6es(diceRoll))

Intro til JS Funktioner

Funktioner i JS

Funktioner er vigtige og anvendes MEGET!

3 måder at definere dem på:

- Funktions-erklæringer
 - Hyppigst anvendte,
 - Nøgleordet **function**
 - Kan kaldes inden de erklæres ("hoistes");
- Funktions-udtryk
 - Skaber en funktions-værdi
 - Anvendes i udtryk eller sætninger hvor man har brug for en funktions-værdi
 - Kan bindes til et navn via `let` eller `const` som alle andre værdier
- Pile-funktions-udtryk
 - Kompakt alternative til alm. funktions udtryk
 - Anvendes typisk til at lave en "lille" funktion, der overføres som parameter
 - Relateret til "lambda-udtryk"
 - Har visse forskelle og begrænsninger ifbm. metoder og objekter

```
// funktions kald/applikation/anvendelse/invokation
//bmi er synlig her, da dens funktionserklæring "hoistes"
let b=bmi(1.80,90); //27.78

// funktions erklæring
function bmi(h,w){
  return w/(h*h);
}

//funktions udtryk (anonymt),
let bmi2= function(h,w){
  return w/(h*h);
};
b=bmi2(1.80,100); //30.86

//funktions udtryk (navngivet)
bmi2= function calcBMI(h,w){return w/(h*h);};
b=bmi2(1.80,110); //33.95

let bmix=bmi2;
b=bmix(1.80,120); //37.03

//pile-funktioner
const bmi3 = (h,w) =>{ return w/(h*h)};
b=bmi3(1.80,130); //40.12

//hvis kun 1 parameter: parentes i parameter liste unødvendig
//hvis kun 1 statement: fjern {}, og "return" er underforstået
const incr = a => a+1;
b=incr(5); //6
```

Funktions parametre

- Vi ønsker at udskrive nummererede overskrifter i et antal sproglige varianter
 - print_N_DK og print_N_UK er jo helt ens, bortset fra hvilken print funktion den kalder: så skal vi jo parameterisere
- funktioner kan overføres som argumenter til funktioner som alle andre værdier
- Den overførte funktion kan kaldes på alm vis.

```
function printHeadingDK(no){  
    console.log("Jeg er overskrift nummer " + no);  
}  
function printHeadingUK(no){  
    console.log("I am heading "+ no);  
}  
function print_N_DK(N){  
    for(let i=0;i<N;i++)  printHeadingDK(i);  
}  
function print_N_UK(N){  
    for(let i=0;i<N;i++)  printHeadingUK(i);  
}  
  
print_N_DK(5);  
print_N_UK(5);  
  
function repeatHeading(N,print){  
    console.log(`Will repeat ${N} time the function`);  
    for(let i=0;i<N;i++)  
        print(i);  
}  
repeatHeading(5,printHeadingDK);  
repeatHeading(5,printHeadingUK);  
repeatHeading(5,no => console.log("<H1> heading "+no+"</H1>"));
```



Jeg er overskrift nummer 0
Jeg er overskrift nummer 1
Jeg er overskrift nummer 2
Jeg er overskrift nummer 3
Jeg er overskrift nummer 4
I am heading 0
I am heading 1
I am heading 2
I am heading 3
I am heading 4

Funktions parametre

- Parameteriserede funktioner: mere genanvendelige, mere generiske
 - én funktion til sortering istedet for 100 varianter:
 - Sort_strings(..)
 - Sort_ints(..)
 - Sort_score(..)
 - ...
 - => sort(array, compare)
- Call-backs
 - Event-handlers:

```
function handleClick(){...}  
htmlElem.addEventListener("click", handleClick);
```

Funktions parametre i C.

- I kender qsort i C!

```
void qsort( void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))
```

C version

```
const char * array[] = { "eggs", "bacon", "cheese", "mushrooms",
"spinach", "potatoes", "spaghetti"};

/* Compare the strings.*/
static int compare (const void * a, const void * b) {
    /* The pointers point to offsets into "array", so we need to dereference
     them to get at the strings.*/
    return strcmp (*(const char **) a, *(const char **) b);
}
/* n_array is the number of elements in the array.*/
qsort (array, n_array, sizeof (const char *), compare);
```

JavaScript version

```
let arr = [ "eggs", "bacon", "cheese", "mushrooms", "spinach", "potatoes",
"spaghetti" ];

function compare(a,b) {
    if (a<b) return -1;
    if (a>b) return 1;
    return 0;
}
arr.sort(compare);
//eller bare arr.sort();
```

Function Parameter Quizz: Q1

Finish the program below to print out overweight actors:

```
let actorDB=[{name:"Stallone", bmi: 29}, {name:"Spacey", bmi: 28},  
{name:"Nicolson", bmi: 33}, {name:"Pachino", bmi: 17}, {name:"Jolie", bmi: 20}];  
  
function isHealthy(bmi){ return bmi>=18.5 && bmi <25; }  
function isUnderWeight(bmi){ return bmi <18.5; }  
function isOverWeight(bmi){ return bmi>=25 && bmi <30; }  
  
function printActorBMI_short(actor){  
    console.log(`Name: ${actor.name}, BMI: ${actor.bmi}`);  
}  
  
function printSelection(heading,which){  
    console.log(heading);  
    for (let i =0; i<actorDB.length;i++){  
        let actor=actorDB[i];  
        if(which(actor.bmi))  
            printActorBMI_short(actor);  
    }  
}  
  
printSelection("Over weight are: ", );
```

isOverWeight(27)

isOverWeight

bmi>=25

return bmi>=25 && bmi <30;

isOverWeight()

"Stallone"

Function Parameter Quizz: Q2

Complete the program below to allow printing actors in different formats:

```
let actorDB=[{name:"Stallone", bmi: 29}, {name:"Spacey", bmi: 28},  
{name:"Nicolson", bmi: 33}, {name:"Pachino", bmi: 17}, {name:"Jolie", bmi: 26}];  
  
function isHealthy(bmi){ return bmi>=18.5 && bmi <25; }  
function isUnderWeight(bmi){ return bmi <18.5; }  
function isOverWeight(bmi){ return bmi>=25 && bmi <30; }  
  
function printActorBMI_short(actor){  
    console.log(`Name: ${actor.name}, BMI: ${actor.bmi}`);  
}  
  
function printActorBMI_DK(actor) {  
    console.log(`Skuespiller ${actor.name} har et BMI-tal på ${actor.bmi}`);  
}  
function printActorBMI_HTML(actor) {  
    console.log(`<p>${actor.name} has an <em>BMI</em> of ${actor.bmi}</p>`);  
}  
  
function printSelection_v2(heading,which, printer){  
    console.log(heading);  
    for (let i =0; i<actorDB.length;i++){  
        let actor=actorDB[i]  
        if(which(actor.bmi))  
            [REDACTED] ;  
    }  
}  
  
printSelection_v2("Overweight",isOverWeight,printActorBMI_short);  
printSelection_v2("Overvægtige", isOverWeight,printActorBMI_DK);  
printSelection_v2("<h1> Overweight Actors</h1>", isOverWeight,printActorBMI_HTML);
```

printer()

printer(actor)

console.log(actor)

printf("%s\n", actor)

printer

Javascript Objekter

JavaScript Objekter: Litteraler

- En samling af "properties" / "egenskab"
 - Map fra nøgle (unik navn) til værdi ("key-value")
 - propertyName: PropertyValue
 - Et objekt litteral laves vha. { }
 - **Punktum notation** til adgang til nøglens værdi
 - Keys kan være strenge med mellemrum, men **Don't!** *)
 - Et objekt kan bindes til et navn via const og let.
 - Navnet "peger på" objektet.
- En egenskab kan oprettes dynamisk (og fjernes!)
- Mange indbyggede objekter: FX
 - Math, String, Array, Map, Set, JSON, ...
 - Function, Object, ...
- *) kan være behændigt, men misbruges ofte hvor typen "**Map**" er en bedre løsning; kan også have forudsete resultater med visse nøgle strenge!

```
let student1={  
    name: "Gynter",  
    studentID:12345678,  
    "Yndlings Sang": "Højt på træets grønne"  
}  
console.log(student1.name);  
console.log(student1["Yndlings Sang"]);  
  
student1.age=42;  
console.log(student1);
```

```
{  
    name: 'Gynter',  
    studentID: 12345678,  
    'Yndlings Sang': 'Højt på træets grønne top',  
    age: 42  
}
```

JavaScript Objekter: Metoder

- En egenskab kan også være en funktion
 - Funktioner er jo bare værdier!
 - Kaldes så ofte for "metoder"
- Kan læse/skrive/ på objektets andre egenskaber
- Kan kalde andre funktioner, som er medlem af objektet
- "**this**" er en implicit binding i en funktion til det objekt, som "metode-funktionen" tilhører
- Objekter anvendes til at kapsle relateret **data og de funktioner**, der arbejder på de data sammen i en logisk enhed

```
let hans={  
  name:"Hans",  
  sayHello: function(toName) {  
    console.log(`Hej ${toName}, jeg hedder ${this.name}`);  
  }  
}  
hans.sayHello("Brian");  
// Hej Brian, jeg hedder Hans
```

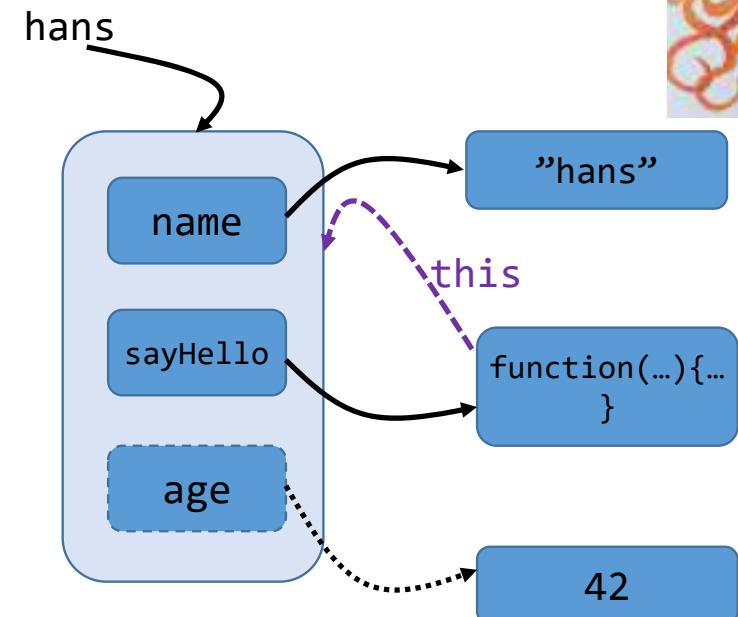
Tentacle model



Objekter: tentakel-modellen

- Objekter kan forstås som et "hovedet", der samler armene, som giber egenskabernes værdi.

```
let hans={  
    name:"Hans",  
    sayHello: function(toName) {  
        console.log(`Hej ${toName}, jeg hedder ${this.name}`);  
    }  
}  
hans.age=42;  
hans.sayHello("Brian");  
// Hej Brian, jeg hedder Hans
```



Objekter: Foranderlighed

- **Tal, Strenge, Booleans** er primitive typer som er "immutable"
 - Operationer på strenge genererer nye strenge.
- **Objekter, Arrays** er "mutable"
- Hvis en binding er erklæres `const`, kan *bindingen* ikke ændres
 - `hans={}` giver fejl, da hans er "konstant" bundet til andet objekt
 - Men det udpegede objekt **kan** ændres
- Som aktuelle parametre til funktioner
 - de primitive typer opfører sig som "value" parametre
 - Konstruerede typer opfører sig som referencer ("pointers")
- `Object.freeze(hans);`

```
let text1 = "Jeg er uforanderlig";
let text2 = text1.toUpperCase();
//text1[2] = 'c'; //run time error
console.log(text1); //Jeg er uforanderlig;

const hans={
  name:"Hans",
  sayHello: function(toName) {
    console.log(
      `Hej ${toName}, jeg hedder ${this.name}`);
  }
}
hans.sayHello("Brian");//Hej Brian, jeg hedder Hans

hans.name="Peter";
hans.sayHello("Brian");//Hej Brian, jeg hedder Peter
hans.name="Hans"; //ændr tilbage

let peter=hans; //nu bundet til same objekt
peter.sayHello("Brian");//Hej Brian, jeg hedder Hans
hans.sayHello("Brian"); // Hej Brian, jeg hedder Hans
```

Tentacle model



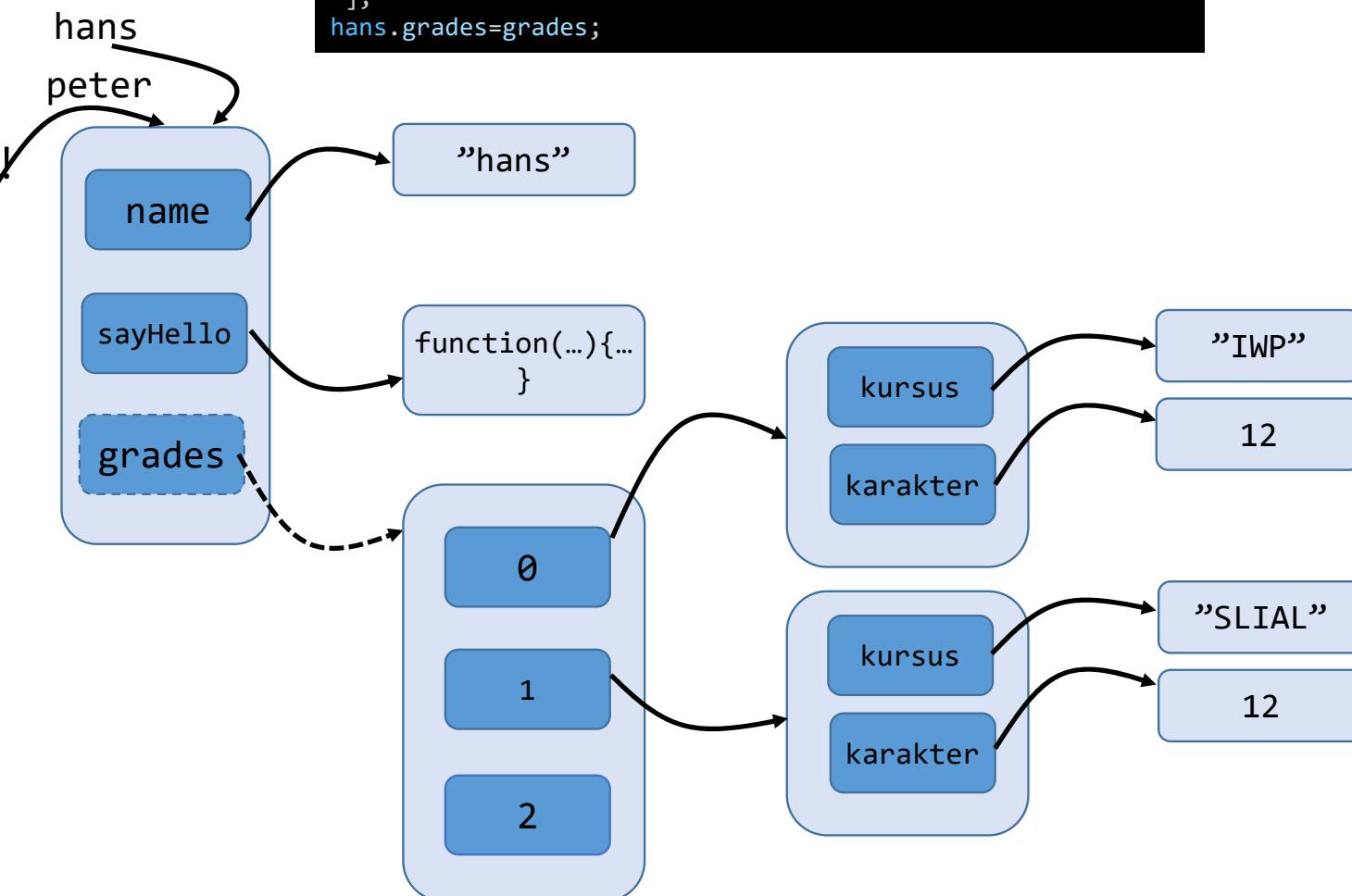
JavaScript Objekter

- En egenskab kan også være et objekt eller array
- Assignment til objekt skaber ny binding, ikke ny kopi af objektet!
 - `let peter=hans`
- Sammenligning bliver sand hvis operander er **bundet** til samme objekt
 - `peter==hans //true`
- NB!
 - `deepCopy, deepComparison, deepFreeze` finde ikke!

```
let kurt={};
Object.assign(kurt,hans);
```

- Kun egenskaber på øverste niveau kopieres
- Kan dog programmers selv – hvis du virkelig har behov

```
let grades=[
  {kursus: "IWP", karakter: 12},
  {kursus: "SLIAL", karakter: 4},
  {kursus: "Imperativ Programmering:", karakter: 7},
];
hans.grades=grades;
```



En objekt fabrik?

- Hvordan kan vi få mange studenter i vores program?
- En mulig "studenter" objekt fabrik?
 - Objekterne her fungerer som uafhængige objekter, som ikke deler arvemasse (se senere)
- En bedre og systematiseret løsning: Constructor funktioner

```
function studentFactory(studentName){  
    let template={  
        name:studentName,  
        sayHello: function(name){  
            console.log(  
                `Hello ${name}, my name is ${this.name}!`);  
        },  
        think: function() {  
            return this.name+ " is thinking hard ... ";  
        }  
    };  
    return template;  
}  
let hans=studentFactory("Hans");  
let peter=studentFactory("Peter");  
hans.sayHello("Brian");  
peter.sayHello("Brian");
```

Constructor Funktioner og ”new” operator

- **New** operatoren opretter en instans af et bruger-defineret (eller indbygget) objekt.
- Anvendes på en funktion.
 - En såkald konstruktor funktion, der opretter de nødvendige egenskaber i funktionen
- Konstruktor funktioner skrives pr. konvention med stort begyndelsesbogstav

```
function Student(name){  
    this.name=name;  
    this.sayHello=function(name){  
        console.log(  
            `Hello ${name}, my name is ${this.name}!`);  
    };  
}  
let hans=new Student("Hans");  
hans.sayHello("Brian");
```

Quizz Objects1: Q1

Actor "Stallone" is born in 1964.

- 1) Add a property to the "actorStallone" object that stores the year he is born.
- 2) Add functionality to the object that calculates his age, given a 'currentYear' as formal parameter.
- 3) Print to the console his age, given that the current year is 2024.

```
let actorStallone={  
  name: "Stallone",  
  [REDACTED]: [REDACTED],  
  [REDACTED]: [REDACTED] {  
    return currentYear - [REDACTED].born;  
  }  
};  
  
console.log([REDACTED]);
```

Quizz Objects1: Q2

Complete the constructor function below such that

- 1) actors have a property that stores the year they are born
- 2) actors have functionality to compute their age
- 3) create a "Spacey" actor object that is born in 1959

```
function Actor(actorName,birthYear){  
    this.name=actorName; //create property "name" with initial value birthYear  
    this.born=birthYear;  
    this.age=function(){  
        return currentYear-this.born;  
    }  
}  
  
let stallone = new Actor ("Stallone", "1946");  
console.log(stallone.age(2024));  
  
let spacey=  
    new Actor ("Spacey", "1959");  
console.log(spacey.age(2024));
```



Eksempel fra dagens øvelser: BMI-JS.JS

- Når bruger indsender nyt sæt BMI data oprettes et BMIEntry objekt

```
//Constructor Function to create BMI-objects
function BMIEntry(name,height,weight){
    console.log(`NEW ${name}, ${height}, ${weight}, `);
    this.userName=name;
    this.weight=weight;
    this.height=height;
    this.calcBMI=function(){... return bmi;};

    this.calcBMIChange=function (otherBMIEntry){
        return round2Decimals(this.calcBMI() -
            otherBMIEntry.calcBMI());
    };
}
```

```
//create a new unique ID for the game
//name and diceCount comes from user input

let entry=new BMIEntry("Mickey",180,98);
let usersBMI=entry.calcBMI();
...
```

Objekt Serialisering

- Serialisering er en betegnelse for at lave struktureret data om til en ”flad” (bit- eller karakter streng).

- Gem objekt i filer
 - Send over netværk

- JSON JavaScript Object Notation,

- Standardiseret data-udvekslingsformat for JS objekter
 - Kun "data" egenskaber inkluderes, ikke funktioner
 - Som læsbar tekst

- JSON objekt med metoderne

- `stringify()` // objekt til streng
 - `parse()` // streng til objekt

Også brugt i diverse konfiguration filer, fx til VisualStudio

```
let serializedHans=JSON.stringify(hans);
let hans2=JSON.parse(serializedHans);
console.log("Serialized object: "+serializedHans);
console.log(hans2);
```

```
Serialized object: {"name": "Hans", "grades": [{"kursus": "IWP", "karakter": -3}, {"kursus": "SLIAL", "karakter": 4}, {"kursus": "Imperativ Programmering:", "karakter": 7}]}{
```

```
{  
  name: 'Hans',  
  grades: [  
    { kursus: 'IWP', karakter: -3 },  
    { kursus: 'SLIAL', karakter: 4 },  
    { kursus: 'Imperativ Programmering:', karakter: 7 }  
  ]  
}
```

Mere om Arrays

JS Arrays

- Objekter, hvor properties/nøglerne er tal!
- Vi bruger [] til array literaler.
- Alternativt Array constructor
 - Mange og fleksible måder at skabe arrays på, se litt.
- Elementer indsættes/fjernes dynamisk
- UTALLELIGE metoder til at ændre på arrays!
 - Brug bog som opslagsværk, eller
 - web (fx Mozilla Developer Network)
- Kan være ”tyndt populeret” (sparse)
 - Afsættes kun plads til de elementer der bruges
- ”Itererbare”

```
let diceRoll=[1,6,6,2,3,4,6];
let dice2=new Array(1,6,6,2,3,4,6);
//prepared to hold 10000 elems.
let dice3=new Array(10000);

let sl=dice2.slice(2, 4); // [ 6, 2 ]
console.log(sl);          // [6,2]

sl.push(1);                // [6,2,1]
sl.push(2);                // [6,2,1,2]
sl.push(3);                // [6,2,1,2,3]
console.log(sl.pop());     // 3
console.log(sl);           // [6,2,1,2]

console.log("Index på første 6er: "+
diceRoll.indexOf(6)); // 1
```

JS Arrays: Itererbare

- Nogle objekter i JS repræsenterer serier af elementer som kan ”oplistes” og ”itereres”
 - Elementerne tages et efter et, og behandles
- Fx. **for (var of coll) {krop}**
 - Binder et element fra coll til ”var”, og udfører kroppen
 - Gentages serielt indtil alle elementer er behandlet
- Array-metoden **forEach** kalder en *callback funktion* for hvert element efter tur.
 - Call-back funktionen tager elementet som første parameter

```
arr.forEach(callback(currentValue[, index[, array]])) {}
```

```
let diceRoll=[1,6,6,2,3,4,6];
for(let d of diceRoll){
    console.log("Dice " +d);
}

let diceSum=0;
for(let d of diceRoll){
    diceSum+=d;
}
console.log(diceSum); //28

function printDiceHeading(dice){
    console.log(`<h3> Dice ${dice}</h3>\n`)
}
diceRoll.forEach(printDiceHeading);
```

```
printDiceHeading(diceRoll[0]);
printDiceHeading(diceRoll[1]);
...
printDiceHeading(diceRoll[6]);
```



```
<h3> Dice 1</h3>
<h3> Dice 6</h3>
<h3> Dice 6</h3>
<h3> Dice 2</h3>
<h3> Dice 3</h3>
...
...
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach

JS Arrays (sparse)

- Arrays kan være tyndt befolkede
 - De fleste JS implementationer afsætter kun plads til de elementer der bruges
 - Ubrugt plads kaldes et "hul"
- Vær dog obs på iteratorer
 - Inkluderer "huller" (undefined værdi)
 - Metoden forEach, skipper huller.

```
index 0 value undefined
index 1 value undefined
...
index 6 value undefined
index 7 value Hejsa

let sparse=[];
sparse[7]="Hejsa";
sparse[42]="med";
sparse[100]="dig!";
console.log(sparse.length); //101!

console.log(sparse[10]); //undefined
sparse.forEach(s=> console.log(`heading ${s}`));

//prints all values, incl holes.
for(let [i,d] of sparse.entries()){
    console.log("index "+i+" value "+d);
}

//iterates over non-holes.
let s="";
diceRoll.forEach(d=>s+=`heading ${d}\n`);
console.log(s);
```

heading Hejsa
heading med
heading dig!

Videregående om arrays og funktioner

- Map og reduce er velkendte indenfor funktions-orienteret programering
- (også kendte inden for cluster computing til big-data beregning)
 - Map: skaber et nyt array hvor hvert element er "behandlet" af map
 - Reduce: reducerer en serie af værdier til en (typisk)

```
let diceRoll=[1,6,6,2,3,4,6];

function sum(prevSum,newValue) {
    return prevSum+newValue;
}

let res=diceRoll.reduce(sum,0);
console.log("SUM="+res);           //=28!
```

```
sum(0,1)
sum(1,6)
sum(7,6)
sum(13,2)
sum(15,3)
sum(18,4)
sum(22,6)=28
```

```
let strings=["Hejsa", "med", "dig!"];
let avgStrLen=strings.map(s=>s.length).reduce(sum,0)/strings.length;
console.log("Avg Len="+avgStrLen); //=4!
```

```
[ "Hejsa", -map---> [5,
  "med",   -map---> 3,
  "dig!"]  -map---> ,4] -reduce ---> 12 / 3;
```

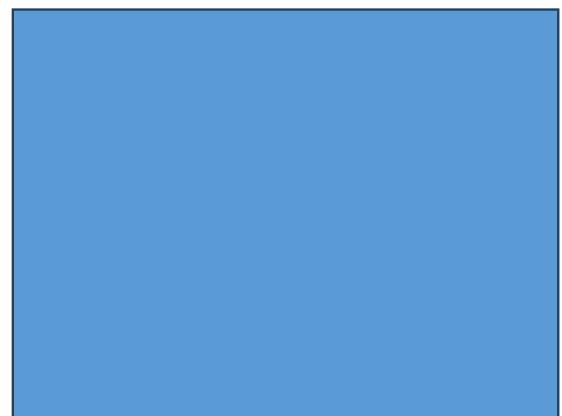
```
avgStrLen=strings.map(s=>s.length).reduce((s,v)=> s+v,0)/strings.length;
```

Function Parameter Quizz: Q3

What does the following program print to the console?

```
1  let actorDB=[{name:"Stallone", bmi: 29}, {name:"Spacey", bmi: 28},  
2    {name:"Nicolson", bmi: 33}, {name:"Pachino", bmi: 17}, {name:"Jolie", bmi: 20}];  
3  
4  function isHealthy(bmi){ return bmi>=18.5 && bmi <25; }  
5  function isUnderWeight(bmi){ return bmi <18.5; }  
6  function isOverWeight(bmi){ return bmi>=25 && bmi <30; }  
7  
8  ✓ function printActorBMI_short(actor){  
9    console.log(`Name: ${actor.name}, BMI: ${actor.bmi}`);  
10  }  
11  
12 ✓ function printSelection_v3(heading, which, printer){  
13  console.log(heading);  
14  const matches=actorDB.filter(b=>which(b.bmi));  
15  matches.forEach(m=>printer(m));  
16  }  
17  
18  printSelection_v3("Overskrift1: ", bmi=>bmi>=30,printActorBMI_short);  
19  const f = function(bmi){return bmi>=30;}  
20  printSelection_v3("Overskrift2: ", f,printActorBMI_short);  
21
```

Linie 1:	<input type="text"/>
Linie 2:	<input type="text"/>
Linie 3:	<input type="text"/>
Linie 4:	<input type="text"/>
Linie 5:	<input type="text"/>



Eks "BMI Database"

- Array af BMIEntry objekter
- Brug af **filter** metoden til at lave et nyt array med BMIEntries hvor entry.userName matcher userName argument.
- Funktionen til filter returnerer sand/falsk

```
function BMIDB() {  
    this.bmiData=[];  
    //Add test data  
    this.bmiData.push(new BMIEntry("Mickey",180, 90));  
  
    this.lookup= function(userName){... };  
    this.calcDelta= function (name){... };  
    this.recordBMI=function(bmiRecord){...};  
  
    this.selectUserEntries=function(userName){  
        return this.bmiData.filter(e=>e.userName==userName);  
    };  
}  
  
const theBMIDB=new BMIDB();  
let mickeyData=theBMIDB.selectUserEntries("Mickey");
```

Videregående / Perspektivering

Funktioner kan nestes

- En funktion danner sit eget synlighedsfelt (lexical scope);
- Bindinger indenfor "functionDemo"
er ikke synlige udenfor

```
function functionDemo(){  
    //funktioner kan nestes, synlighedsfelt  
  
    function printHeadingDK(no){  
        console.log("Jeg er overskrift nummer " + no);  
    }  
    function printHeadingUK(no){  
        console.log("I am heading "+ no);  
    }  
    function print_N_DK(N){  
        for(let i=0;i<N;i++) printHeadingDK(i);  
    }  
    function print_N_UK(N){  
        for(let i=0;i<N;i++) printHeadingUK(i);  
    }  
    ..  
}
```

Funktioner som retur-værdier!

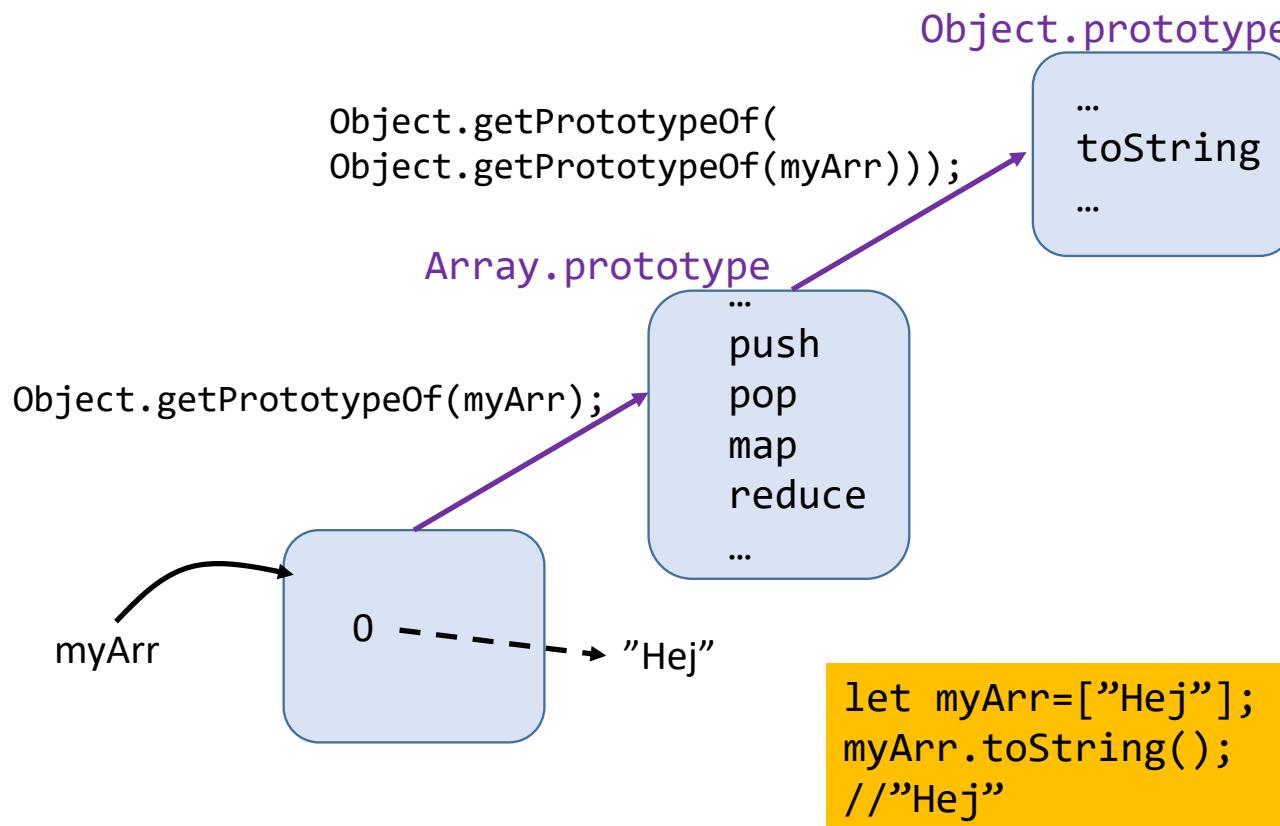
- Funktioner er værdier, derfor kan de returneres!
- nth-funktionen genererer en funktion der kalder f hver n'te gang
- Bemærk at nth's parametre anvendes i fn, og disse binder overlever i fn, efter den er returnerer!
 - "Closures"
- Nævnes her FYI, vi får nok ikke brug for det i web-programmering

```
function printHeadingDK(no){  
    console.log("Jeg er overskrift nummer " + no);  
}  
function repeatHeading(N,print){  
    for(let i=0;i<N;i++)  
        print(i);  
}  
repeatHeading(5,printHeadingDK);  
  
function nth(n,f){  
    let fn=function(no){  
        if(no%n==0)  
            return f(no);  
    }  
    return fn;  
}  
//function that only prints an even heading  
let evenHeading=nth(2,printHeadingDK);  
//even headings between 0..4  
repeatHeading (5, evenHeading);
```

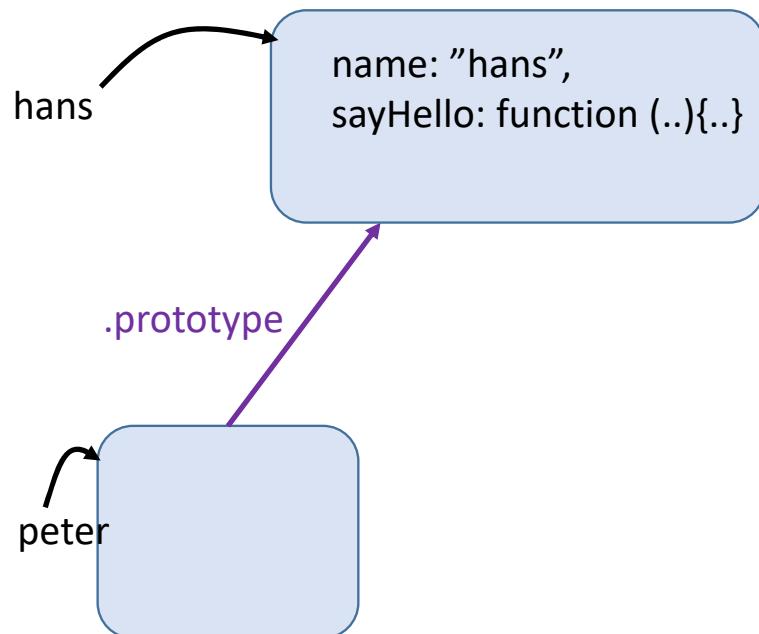
Jeg er overskrift nummer 0
Jeg er overskrift nummer 2
Jeg er overskrift nummer 4

JavaScript Objekter: Prototyper

- De fleste JS objekter er skabt ud fra en "prototype", og tilføjer egenskaber til denne
- Adgang til en "property" delegeres via prototype kæden til den mest specifikke erklæring



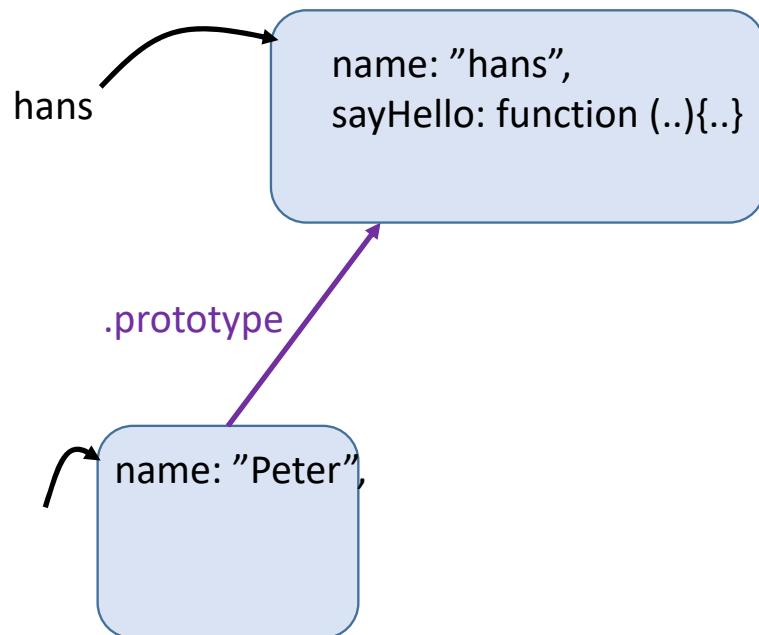
JavaScript: prototyper



Du kan lave dit eget objekt ud fra et andet brugt som prototype med
`Object.create()`

```
let hans={  
  name:"hans",  
  sayHello: function(nm) {  
    console.log("Hej "+nm+", jeg er "+this.name);  
  }  
}  
  
hans.sayHello("Brian"); // Hej Brian, jeg er hans  
  
let peter=Object.create(hans);  
  
peter.sayHello("Brian"); // Hej Brian, jeg er hans
```

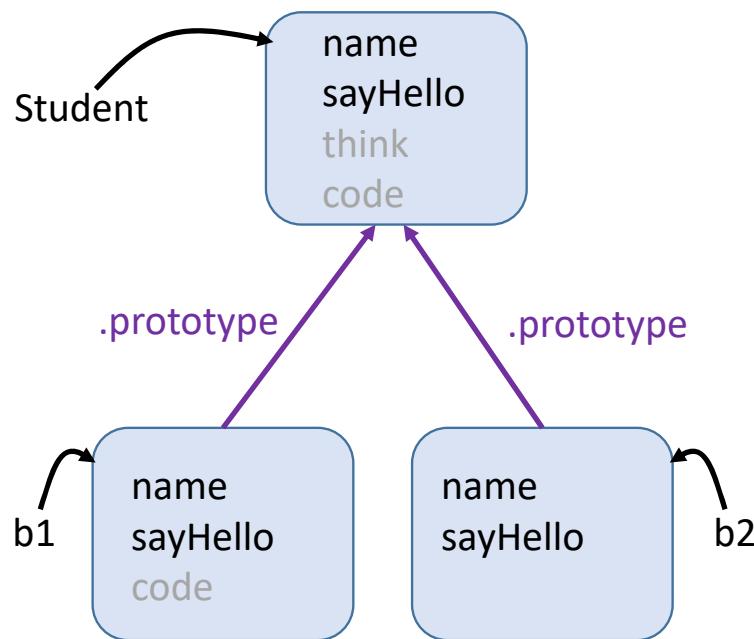
JavaScript: prototyper



Du kan lave dit eget objekt ud fra et andet brugt som prototype med
`Object.create()`

```
let hans={  
  name:"hans",  
  sayHello: function(nm) {  
    console.log("Hej "+nm+", jeg er "+this.name);  
  }  
}  
  
hans.sayHello("Brian"); // Hej Brian, jeg er hans  
  
let peter=Object.create(hans);  
  
peter.sayHello("Brian"); // Hej Brian, jeg er hans  
peter.name="Peter";      //Opret 'name' i peter objekt  
peter.sayHello("Brian"); //Hej Brian, jeg er Peter
```

JavaScript: prototyper



```
function Student(name){  
    this.name=name;  
    this.sayHello=function(name){  
        console.log(`Hello ${name}, my name is ${this.name}!`);  
    };  
}  
let b1=new Student("Brian");  
  
//Add properties to the prototype; now shared by all Students  
Student.prototype.think=function() {  
    return this.name+ " is thinking hard ... ";};  
Student.prototype.code=function() {  
    return this.think()+ " and is coding the pants off";}  
  
console.log(b1.code());  
let b2 = new Student("Birte");  
console.log(b2.code());  
  
b1.code=function(){  
    return this.think()+ " and is coding rather lazily";}  
console.log(b1.code());  
console.log(b2.code());
```

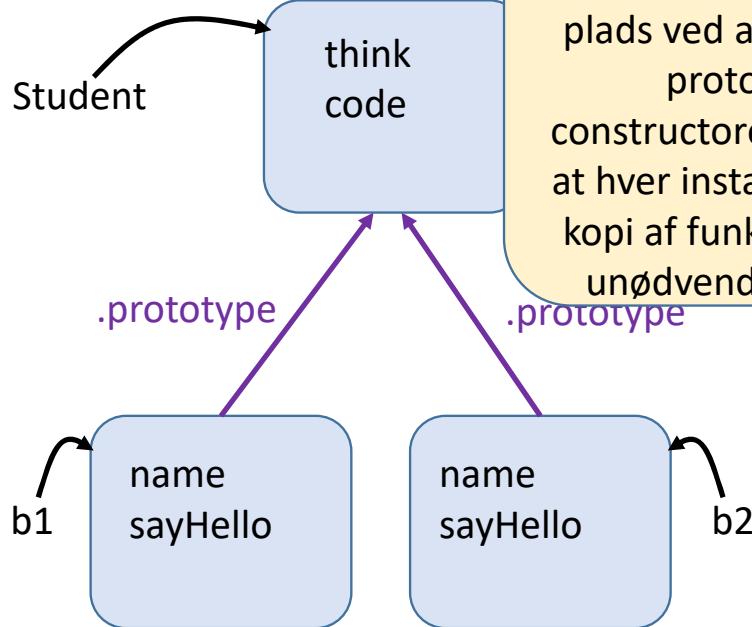
Objekter lavet af samme constructor function har **samme prototype**

Extending objekt at tilføje properties

Overriding metoder ved at erstatte den m. specialiseret version

*Brian is thinking hard ... and is coding the pants off
Birte is thinking hard ... and is coding the pants off
Brian is thinking hard ... and is coding rather lazily
Birte is thinking hard ... and is coding the pants off*

JavaScript: prototyper



NB: HVIS du skal lave *rigtig* mange instanser, kan du spare plads ved at tilføje metoderne til prototypen fremfor i constructoren. Herved undgås det at hver instans/objekt får sin egen kopi af funktions kodden (normalt unødvendigt og spild af plads)

```
function Student(name){  
    this.name=name;  
    this.sayHello=function(name){  
        console.log(`Hello ${name}, my name is ${this.name}!`);  
    }  
}  
  
Student("Brian");  
  
ties to the prototype; now shared by all Students  
Student.prototype.think=function() {  
    return this.name+ " is thinking hard ... ";}  
Student.prototype.code=function() {  
    return this.think()+ " and is coding the pants off";}  
  
console.log(b1.code());  
let b2 = new Student("Birte");  
console.log(b2.code());  
  
b1.code=function(){  
    return this.think()+ " and is coding rather lazily";}  
console.log(b1.code());  
console.log(b2.code());
```

Extending objekt at tilføje properties

Overriding metoder ved at erstatte den m. specialiseret version

*Brian is thinking hard ... and is coding the pants off
Birte is thinking hard ... and is coding the pants off
Brian is thinking hard ... and is coding rather lazily
Birte is thinking hard ... and is coding the pants off*

Lidt generelt om objekter

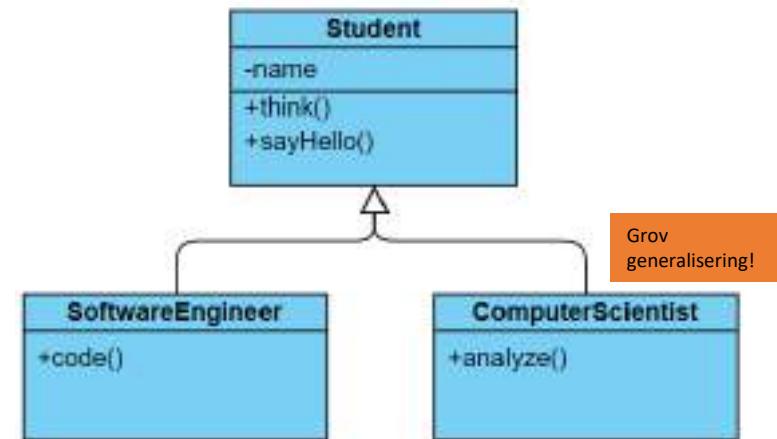
- Der er flere synspunkter på hvad objekter er (og især hvad OO er!) HER:
- ***En samling af tæt-relaterede variable og funktioner, der opererer derpå***
 - Fungerer som en selvstændig enhed i større programmer.
 - En "abstraktion":
 - kan repræsentere en "ting" fra en virkelig eller simuleret verden ("student", "bil", "betaling", "lønseddel")
 - eller en abstrakt datatype (fx "stak", "kø", "træ", "graf"):
- **Indkapsling:** Vi ønsker kun at "annoncere" information om, hvordan en bruger (programmøren/"kalderen") skal bruge objektet, ikke interne detaljer
 - Fx om bruger vi array eller liste til at lave "stak"?
 - Kalderen skal kun kende operationer "push", "pop"
 - => vi kan ændre intern repræsentation og algoritmer uden at ændre alle steder i kode objektet bliver brugt
 - => vi kan arbejde uafhængigt af hinanden
- **Interface:** De egenskaber (fortrinsvist funktioner) som må kendes udad til.
 - I nogle sprog kan funktioner og variable erklæres som "private" eller "public"
 - Nøgleord **private** tilføjet i JS2022

Lidt generelt om objekter

NB! I OO skelnes egenskaber (properties)

- variable ~ "attributter",
- funktioner ~ "metoder"

- **Class:** En beskrivelse af formen for lignende objekter
- **Objekt / "instans":** et konkret eksemplar af klassen
- Inheritance: Et objekt kan "arve" egenskaber fra et andet objekt.
 - Typisk: specialisering/generalisering
- Polymorfi: Evnen for et objekt at optræde under forskellig "type".
 - Hvis objektet mindst har "rette egenskaber" (interface) skal det kunne bruges alle steder i programmet, som forventer den egenskab (interface).
 - `University.examine(Student s) {s.think()...}`
`University.examine(michael);`



I et hypotetisk klasse-baseret OO sprog

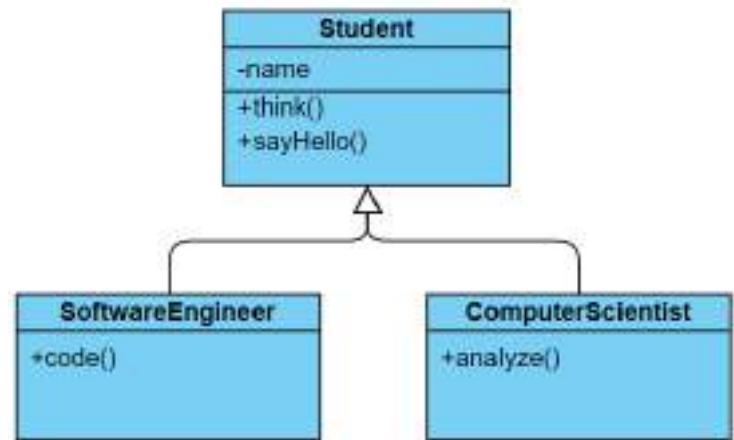
```
class Student(){
    String name;
    void think(){...};
    void sayHello();
}
class SoftwareEngineer inherits Student{
    void code() {doCode...};
}
```

```
Student hans, peter;
SoftwareEngineer michael;
ComputerScientist kurt;
micheal.think();
```

JavaScript "Classes"

- En overbygning på prototype begrebet, som **simulerer** et klasse-baseret oo-sprog

```
class Student{  
    constructor(name) {this.name=name}  
    sayHello (name){console.log(`Hello ${name}, my name is ${this.name}!`);}  
    think() {return this.name+ " is thinking hard ... ";}  
}  
  
class SoftwareStudent extends Student {  
    code() {return this.think()+ " and is coding the pants off";}  
}  
  
let b1 = new Student("Brian1");  
let b2 = new SoftwareStudent("Brian2");  
console.log(b2.code());  
console.log(b1.think());  
console.log(b1.code()); //Students don't generally code; so trying code()  
on a student fails.  
}
```



Brian2 is thinking hard ... and is coding the pants off

Brian1 is thinking hard ...

Error: b1.code is not a function

JavaScript og OO

- Constructor funktioner er nyttigt
- Brug OO og inheritance sparsommeligt:
Kommer på dat3/sw3 med fuld blæs
 - OOA
 - OOD
 - OOP
- Der er mange finurligheder af JS objekt model, som vi ikke kommer ind på
 - [DF chap 9] har yderligere info
 - For særligt interesserede: Se JS bøger under supplerende materiale, fx <http://speakingjs.com/es5/ch17.html>
 - [Understanding the four layers of JavaScript](#)
 - <https://javascript.info/prototype-inheritance>



Internetværk og Web-programmering

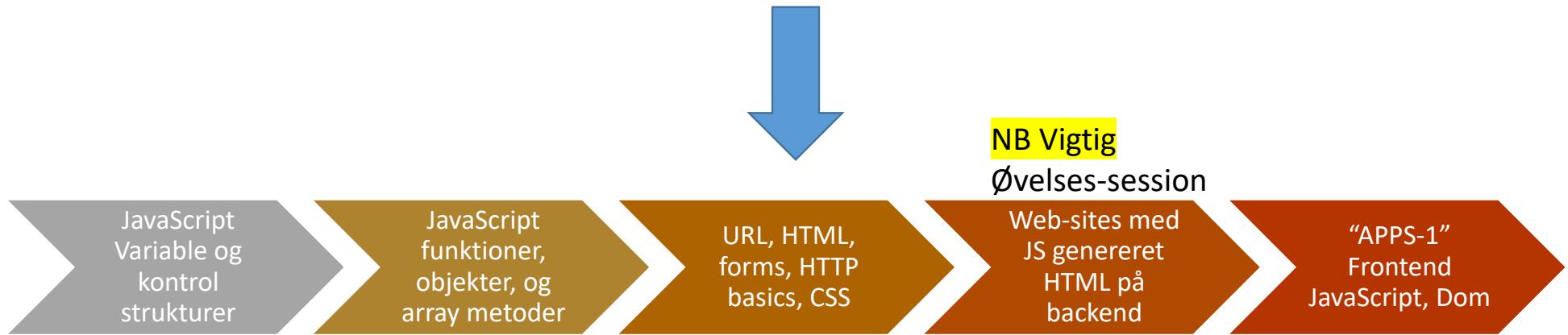
HTML, HTTP, Forms, CSS

Forelæsning 3
Brian Nielsen

Distributed, Embedded, Intelligent Systems



Lektions-status



URLs

Uniform Resource Locators

- En henvisning til en tilgængelig ressource aka **web-adresse**
- Ressource: den “ting” man ønsker at få/give adgang; fx en web-side, video, billede, applikation, fil, IoT Device, ...
- Angiver hvor ressourcen befinner sig, og hvordan den tilgås



Protokol (mere korrekt ”scheme”): Angiver den service (HTTP, FTP, file, mailto, tel, IRC,...) browseren skal tilgå (ofte svarende til en applikationsniveau protokol)

Domænet: DNS navn eller IP adresse, samt evt. port nummer "130.225.63.3:8080"

- Default: 443 (HTTPS) 80 (HTTP)

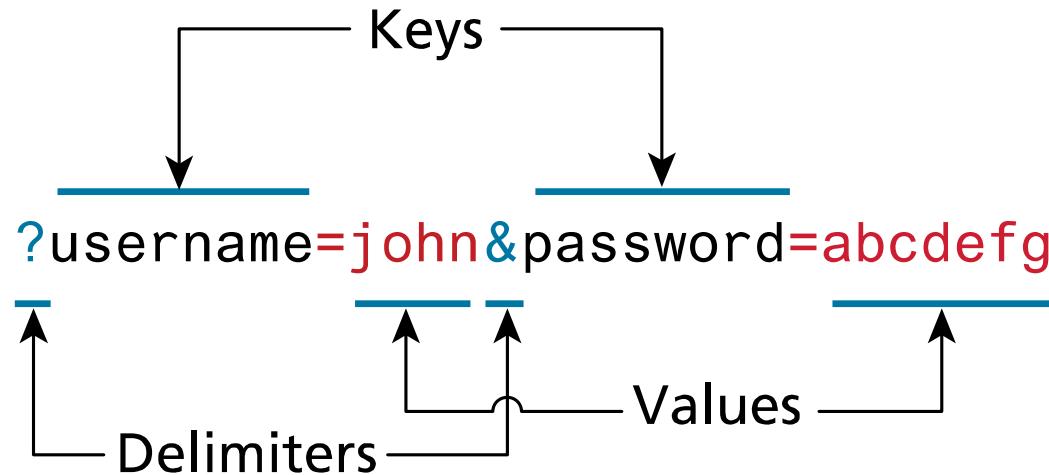
Sti-navn: Navn på ressource indenfor det pågældende domæne: oversættes sommetider til et filnavn

Søgestreng: en række ”key-value” par som server program kan bruge til at finde den ønskede (del af) ressourcen: fungerer som parameterliste til scripts

Fragment: en måde at bede om en portion af et dokument.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web

Uniform Resource Locators: Query String



Uniform Resource Locators: Encoding

- Problem 1: Hvad hvis vi ønsker at bruge de reserverede special tegn som / : ?=&=+#()[] i URL værdier (fx værdifeltet i forespørgsel / query) ?
- Problem 2: Hvad hvis vi vil bruge danske tegn i fx URL ressource navne?

- URL kan kun sendes på nettet som ASCII
- Special tegn og reserverede tegn, fx ?=&+()[]#=, skal "escapes"
 - Escape karakter i URL: %
- Derfor overføres de "URL-encodet"
- URL-encoding: https://www.w3schools.com/tags/ref_urlencode.ASP
 - Fx. Æsel? -> %C3%86sel
 - Skjules for det meste af nyere browsere
- [DF 11.9] beskriver URL ogSearchParams" APIs (Javascript objekter)



https://da.wikipedia.org/wiki/%C3%86sel

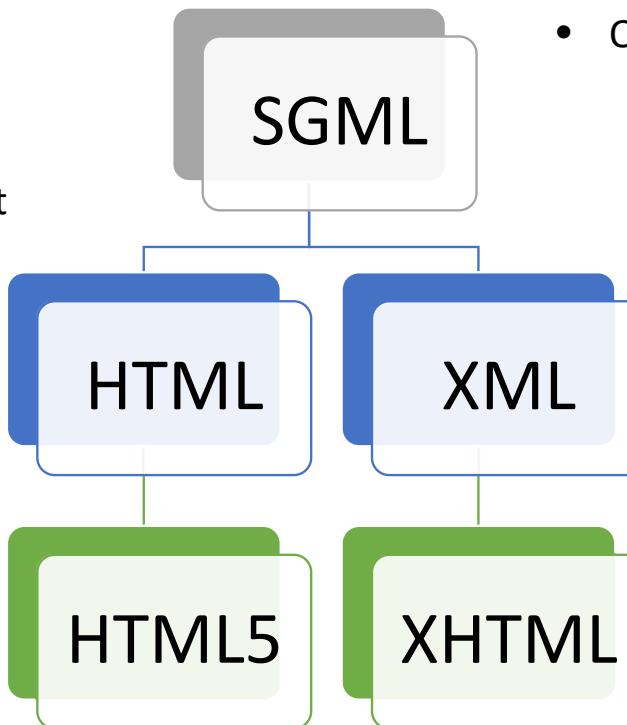
Markup og HTML

Hypertext & Markup-sprog

- **Hypertext:** Samle tekst information og forbinde disse via henvisninger, så informationen kan læses i bruger-defineret rækkefølge.
 - Gammel ide: '45,
 - '65: termen "hypertext" brugt af Ted Nelson i 1965 [wikipedia]
 - '86 Apples Hypercard system
 - Hypermedia (linked multi-medier)
 - '89 www draft: <https://home.cern/science/computing/birth-web/short-history-web>
- **"Markup"** er markeringer, tilføjelser, og annoteringer, der tilføjer ekstra information om et (tekst) dokument.
- **Markup sprog:** det computer sprog (syntax og semantik) der anvendes.
 - LaTeX tilføjer struktur og typesetting information.
 - **HTML** (Hyper Text Markup Language) udviklet af Tim Berners-lee 1991 til www hyper-tekst documenter
 - YAML: yet-another-markup-language
 - MD: Markdown
 -

Markup-sprog

- **HTML** (Hyper Text Markup Language) udviklet af Tim Berners-lee 1991, inspireret af SGML, til www hyper-tekst documenter
- Til visning af hypertext documenter
- HTML5 (2012) drives af Web Hypertext Application Technology Working Group ([WHATWG](#))



```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a heading (first one) </h1>
<p> The paragraph </p>

</body>
</html>
```

- **Standard Generalized Markup Language** (1986; [ISO](#) 8879) er en standard til at definere markup-sprog til elektroniske dokumenter.
- Omfangsrigt og komplext

- **eXtensible Markup Language** (W3C, 1998) **delmænge** af SGML til brug i WWW, som var nemmere at lave værktøjer til.
- Brugerne definerer selv deres tilladte elementer, "tags", nesting, etc.
- Mål: beskrive hvad data "består af"
- En instans af et XML dokument:

```
<<students>
  <<student id="100026">
    <name>Joe Average</name>
    <age>21</age>
    <major>Biology</major>
  <<results>
    <result course="Math 101" grade="C-"/>
    <result course="Biology 101" grade="C+/">
    <result course="Statistics 101" grade="D"/>
  <</results>
</student>
<<student id="100078">...</student>
</students>
```

Semantisk Markup

- Fokus på at definer **struktur og indhold af dokumentet**, ikke det **visuelle**
- Annotere et (tekst-) fragment med den **mest "sigende"** mærkat, som udtrykker hensigten bag teksten.
 - Overskrift? Kapitel? Sektion? Figur? Figurtekst? Tabel?
- Har en række fordele
 - Nemmere at **vedligehold**: farvelade og layout kan gives separat, og fælles for alle elementer, fx overskrifter.
 - **Performance**: Mindre dokumenter, der ikke indeholder en masse visuelle instruktioner.
 - Markering af strukturen kan **øge tilgængelighed** for svagtseende “Accessibility” (<http://www.w3.org/WAI>).
 - **Søgemaskine optimering**: Ord som er semantisk fremhævede, eller som fremtræder i titlen og overskrifter gives højere prioritering rangering af søgeresultater

VIGTIG

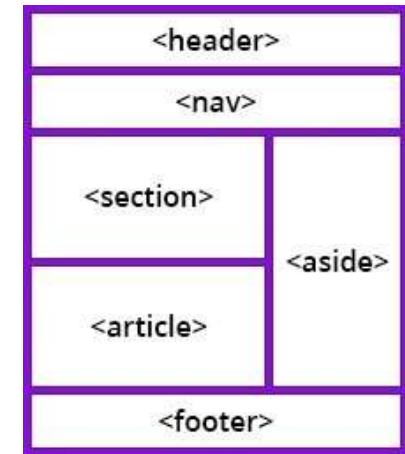
HTML Elementer

- Ca. 110 forskellige (32 nye i HTML5)

- Strukturelle og sektionering: Opdeling af dokument i separate logisk sammenhængende områder
- Opdeling i **tekst-blokke**: Opdeling af sektion i mindre, samhørende dele; laver linieskift
- **Inline tekst**: Definerer betydning/stil af et ord indenfor en tekst-linie; uden at lave linieskift.
- Billeder, 2D- og 3D grafik, multi-medier
- Tabeller
- Bruger-input, formularer ("Forms"), interaktive dialoger og menuer
- Scripting og templates
- Meta-data: Information om dokumentet (fx karaktersæt, forfatter., direktiver til søgemaskiner ...)

- Se fx oversigt på:

- <https://html.spec.whatwg.org/multipage/>
- <https://developer.mozilla.org/en-US/docs/Web/HTML/Element>
- <https://www.creativebloq.com/advice/html-tags>
- <https://www.atnyla.com/tutorial/introduction-to-html5/2/238>



God brug er **vigtig** for
Muligheder for at lave egen layout og styling
Brugbarhed for svagtseende
Optimering for søgemaskine

HTML5 Syntax: Elementer og Attributter

- Et **HTML document** består af tekstuelt indhold og **HTML elementer**
- **HTML element** består af
 - **Element navn** er omsluttet af <> parenteser (også kaldet **tag**)
 - **Attributer** (i åbningstag)
 - **Indhold** mellem start og slut tag.
- Fx **anchor element** (laver hyperlink)



- “**Tom element**” har ikke noget tekst indhold og intet slut tag.

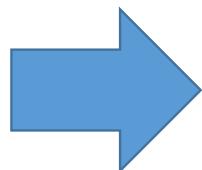
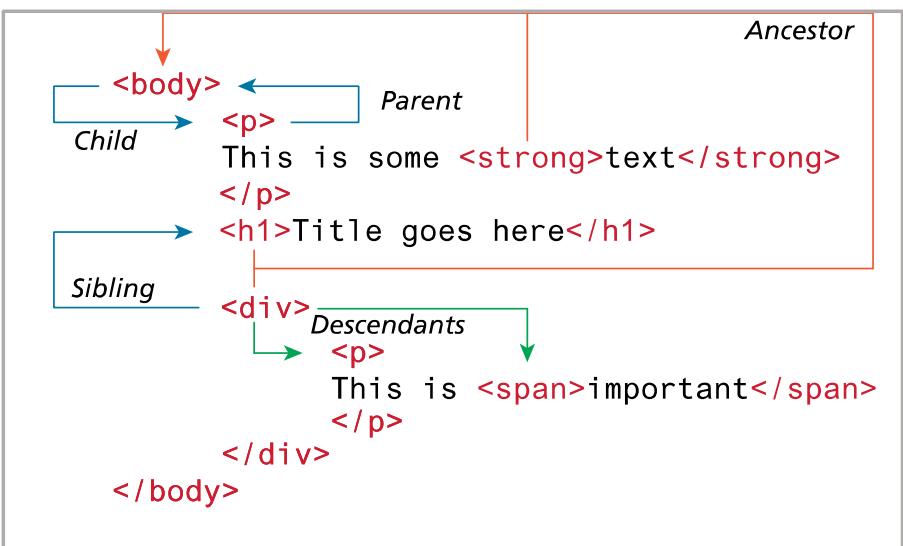
```

```

Element name

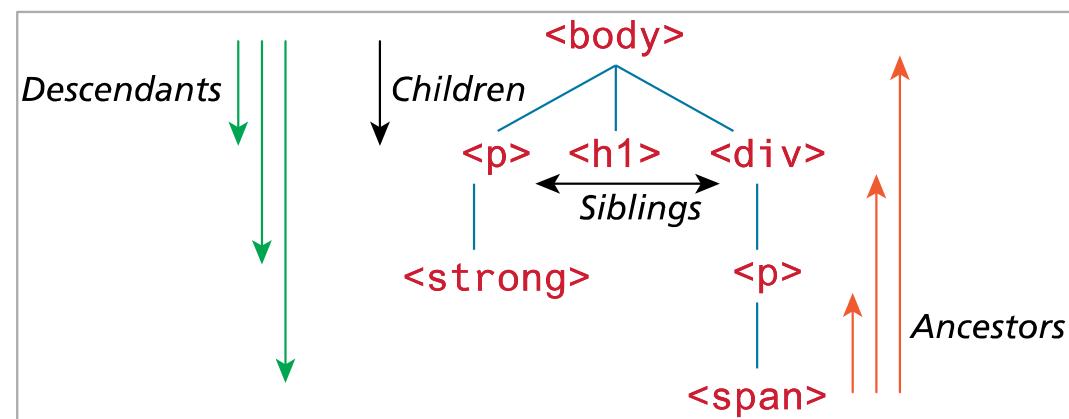
HTML Syntax: Nesting

Som parenteser, der skal balancere



NB Vigtigt:

Elementernes nesting kan vises som et træ:



Relationer som alm. familie træ:

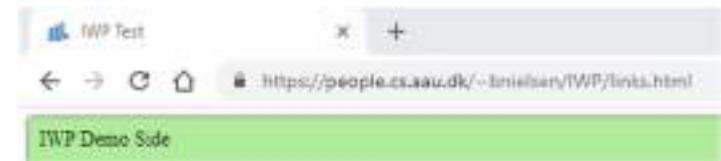
- Elementer på samme niveau: **Søskende**
- Elementer på underliggende niveauer: **efterfølgere**
- Elementer på overliggende niveauer: **forfædre**
- Elementer på umiddelbart overliggende niveau: **forældre**
- Elementer på umiddelbart underliggende niveau: **børn**

class og id attributter

- Alle html elementer kan gives en id attribut
 - Fx `<p id="id1" > Dette er en paragraf</p>`
 - Tekst strenge uden whitespace;
 - Id-attribut skal være unikt i dokumentet
 - Muligt at udpege det givne id som et "*anchor*"
 - JS/CSS kan udpege elementet ud fra id.
- Alle html elementer kan grupperes i bruger-definerede klasser
 - Fx `<p class="note" > Dette er en paragraph </p>`
 - Gør et muligt for JavaScript / CSS at udpege alle elementerne med det givne klasse-navn
 - Kan samtidigt have id.
- Se fx:
 - https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/class
 - https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/id
- Hvilke andre attributer hvilke elementer tillader: se documentation fx på MDN.

HTML header og body

```
<!DOCTYPE html>
<html lang="da">
  <head>
    <title>IWP Test</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="css/style.css">
  </head>
  <body>
    <header> IWP Demo Side</header>
  </body>
</html>
```



1. Doctype fortæller browser at det kommende tekster et et HTML5 dokument
2. <html> er rod-elementet, som markerer start og slut.
 - Det er god praksis at sætte sprog attributtet afh. skærm læsere.
 - Efterfølges af ét <head> og ét <body> element
3. <head> indeholder info om dokumentet. Kan indeholde mange ting! Typisk mindst:
 - Sætter titlen på siden; vises øverst i browseren vindue/tab
 - Definerer det karaktersæt der anvendes i dokumentet. UTF-8 tillader de fleste naturlige sprogs tegn – anbefalet.
 - Brugte eksterne stilarter
 - Brugte eksterne javascripts
4. <body>indeholder selve indholdet som skal vises på siden
 - <header> er et alm. element indeholder introducerende indhold

Meta-data: data som er information om andet data

https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started#Anatomy_of_an_HTML_document

IWP Demo Side is a crude HTML page made for demo purposes
Indholdsfortegnelse:

- [Introduktion](#)
 - [HTML](#)
 - [JavaScript](#)
- [to do](#)

Ingen bemærkninger

Resultat med browsers default stylesheet

Introduktion til Web-Programmering

Centrale elementer i web-teknologi er [HTML](#) og [JavaScript](#).

In bla bla.

Klient-side programmering kommer i [næste lektion](#).

Her er en god [reference til web-teknologi](#)

HTML

Med HTML kan du lave semantisk mark-up af et dokument, som vises i en web-browser. Etc. Etc. Etc.

HTML BODY

Indeholder vigtig information om dokumentet, som modtageren skal bruge for at forstå det.

JavaScript

JS kan bringes til at fungere sammen med en browser og interagere med en HTML side.



Fig. 1 - Sød Panda som browser bambus! © National Geographic

According to Goodreads, [Turing Award Winner Edsger Dijkstra](#), has stated that " If debugging is the process of removing software bugs, then programming must be the process of putting them in. "

Du skal dog først kende til [HTML](#).

[Forfatter af Brian Nielsen](#)

```
<body>
  <header> IWP Demo Side is a crude HTML page made for demo purposes </header>
  <nav>
    Indholdsfortegnelse:
    <ul>
      <li><a href="#htmlafsnit"> Introduktion </a>
      <ul>
        <li><a href="#htmlafsnit"> HTML </a></li>
        <li><a href="#htmlafsnit"> JavaScript. </a> </li>
      </ul></li>
      <li> to do </li>
    </ul>
  </nav>
  <aside>Ingen bemærkninger</aside>
  <main>
    <section id="introsection">
      <h1> Introduktion til Web-Programmering</h1>
      <em> Centrale</em> elementer i web-teknologi er <a href="#htmlafsnit"> HTML </a> og <a href="#htmlafsnit"> JavaScript. </a>
      <p>la bla bla.</p>
      <p> Klient-side programmering kommer i <a href="lecture5.html">næste lektion. </a> </p>
      Her er en god <a href="https://developer.mozilla.org/" title="God Web Refence Site">reference til web-teknologi </a>
    </section>
    <section id="htmlafsnit">
      <h2> HTML </h2>
      Med HTML kan du lave semantisk mark-up af et dokument, som vises i en web-browser. Etc. Etc.
      Etc.
      <h3>HTML BODY</h3> indeholder vigtig information om dokumentet, som modtageren skal bruge for at forstå det.
      </section>
      <!-- HER ER EN KOMMENTAR -->
      <section id="jsafsnit">
        <h2>JavaScript</h2>
        JS kan bringes til at fungere sammen med en browser og interagere med en HTML side.
        <figure>
          
          <figcaption>Fig. 1 - Sød Panda som browser bambus! &#169; National Geographic </figcaption>
        </figure>
        According to <cite>Goodreads</cite>, <a href="https://da.wikipedia.org/wiki/Turing-prisen">Turing Award Winner</a> <a href="https://en.wikipedia.org/wiki/Edsger_W._Dijkstra"> Edsger Dijkstra:</a> has stated that
        <q cite="https://www.goodreads.com/author/quotes/1013817. Edsger_W_Dijkstra" >
          If debugging is the process of removing software bugs,
          then programming must be the process of putting them in.
        </q>
        <p> Du skal dog først kende til <a href="#htmlafsnit"> HTML </a>. </p>
      </section>
    </main>
    <footer>
      <a href="https://www.cs.aau.dk/~bnielsen/" rel="author">Forfattet af Brian Nielsen</a>
    </footer>
  </body>
```

<http://people.cs.aau.dk/~bnielsen/IWP/links.html>

IWP Demo Side is a crude HTML page made for demo purposes
Indholdsfortegnelse:

- [Introduktion](#)
 - [HTML](#)
 - [JavaScript](#)
- [to do](#)

Ingen bemærkninger

Introduktion til Web-Programmering

Centrale elementer i web-teknologi er [HTML](#) og [JavaScript](#).

In bla bla.

Klient-side programmering kommer i [næste lektion](#).

Her er en god [reference til web-teknologi](#)

HTML

Med HTML kan du have sémantisk mark-up af et dokument, som vises i en web-browser. Etc. Etc. Etc.

HTML BODY

Indeholder vigtig information om dokumentet, som modtageren skal bruge for at forstå det.

JavaScript

JS kan bringes til at fungere sammen med en browser og interagere med en HTML side.



Fig. 1 - Sad Panda som browser bambus! © National Geographic

According to Goodwells, [Turing Award Winner Edsger Dijkstra](#), has stated that " If debugging is the process of removing software bugs, then programming must be the process of putting them in. "

Du skal dog først kende til [HTML](#).

Forfatter af Brian Nielsen

HTML Ex: struktur elementer

```
<body>
  <header> IWP Demo Side is a crude HTML page
    | made for demo purposes </header>
  <nav> ...
  </nav>
  <aside>Ingen bemærkninger</aside>
  <main>
    <section id="introsection">...
    </section>

    <section id="htmlafsnit">...
    </section>
    <!-- HER ER EN KOMMENTAR -->
    <section id="jsafsnit">...
    </section>
  </main>
  <footer>...
  </footer>
</body>
```

IWP Demo Side is a crude HTML page made for demo purposes
Indholdsfortegnelse:

- [Introduktion](#)
 - [HTML](#)
 - [JavaScript](#)
- [to do](#)

Ingen bemærkninger

Introduktion til Web-Programmering

Centrale elementer i web-teknologi er [HTML](#) og [JavaScript](#).

In bla bla.

Klient-side programmering kommer i [næste lektion](#).

Her er en god [reference til web-teknologi](#).

HTML

Med HTML kan du have sementisk mark-up af et dokument, som vises i en web-browser. Etc. Etc. Etc.

HTML BODY

Indeholder vigtig information om dokumentet, som modtageren skal bruge for at forstå det.

JavaScript

JS kan bringes til at fungere sammen med en browser og interagere med en HTML side.



Fig. 1 - Sod Panda som browser bambus! © National Geographic

According to Goodreads, [Turing Award Winner Edsger Dijkstra](#), has stated that " If debugging is the process of removing software bugs, then programming must be the process of putting them in. "

Du skal dog først kende til [HTML](#).

Forskriftet af Brian Nielsen

HTML Ex: headings og paragraffer

```
<section id="introsection">
  <h1> Introduktion til Web-Programmering</h1>
  <em> Centrale</em> elementer i web-teknologi er ...
  <p>la bla bla.</p>
  <p> Klient-side programmering kommer i
      <a href="lecture5.html"> næste lektion. </a> </p>
</section>
```

<http://people.cs.aau.dk/~bnielsen/IWP/links.html>

IWP Demo Side is a crude HTML page made for demo purposes

Indholdsfortegnelse:

- [Introduktion](#)
 - [HTML](#)
 - [JavaScript](#)
- [Til de](#)

Ingen bemærkninger

Introduktion til Web-Programmering

Centrale elementer i web-teknologi er [HTML](#) og [JavaScript](#).

Ja bla bla.

Klient-side programmering kommer i [næste lektion](#).

Her er en god [reference til web-teknologi](#).

HTML

Med HTML kan du lave semantisk mark-up af et dokument, som vises i en web-browser. Etc. Etc. Etc.

HTML BODY

Indeholder vigtig information om dokumentet, som modtageren skal bruge for at forstå det.

JavaScript

JS kan bringes til at fungere sammen med en browser og interagere med en HTML side.



Fig. 1 - Sød Panda som bærer bambus! © National Geographic

According to Goodreads, [Turing Award Winner Edsger Dijkstra](#), has stated that "If debugging is the process of removing software bugs, then programming must be the process of putting them in."

Du skal dog først kende til [HTML](#).

Forfatter af Brian Nielsen

HTML Ex: hyperlinks, fragmenter og relative URLs

<https://homes.cs.aau.dk/~bnielsen/IWP/links.html>

```
<section id="introduction">
  <h1> Introduktion til Web-Programmering</h1>
  <em> Centrale</em> elementer i web-teknologi er <a href="#htmlafsnit"> HTML
  </a> og <a href="#jsafsnit"> JavaScript. </a>
  <p>la bla bla.</p>
  <p> Klient-side programmering kommer i <a href="lecture5.html">næste
  lektion. </a> </p>
  Her er en god <a href="https://developer.mozilla.org" title="God Web
  Reference Site">reference til web-teknologi </a>
</section>
```

HTML AFSNIT

HTML

Med HTML kan du lave semantisk mark-up af et dokument, som vises i en web-browser. Etc. Etc. Etc.

```
<h3>HTML BODY</h3> indeholder vigtig information om dokumentet, som
modtageren skal bruge for at forstå det.
</section>
```

Relative URLs bliver fortolket relativt til omsluttende dokument:

- <https://homes.cs.aau.dk/~bnielsen/IWP/lecture5.html>
- <https://homes.cs.aau.dk/~bnielsen/IWP/links.html#htmlafsnit>

Der er flere attributter på anchor elementet: fx

- title: angiver formålet med linket (default css viser det som et tool-tip)
- "download"
- Se i reference manualen: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a>

- [Introduktion](#)
 - [HTML](#)
 - [JavaScript](#)
- [to do](#)

Ingen bemærkninger

Introduktion til Web-Programmering

Centrale elementer i web-teknologi er [HTML](#) og [JavaScript](#).

In bla bla.

Klient-side programmering kommer i [næste lektion](#).

Her er en god [reference til web-teknologi](#)

HTML

Med HTML kan du lave sémantisk mark-up af et dokument, som vises i en web-browser. Etc. Etc. Etc.

HTML BODY

Indeholder vigtig information om dokumentet, som modtageren skal bruge for at forstå det.

JavaScript

JS kan bringes til at fungere sammen med en browser og interagere med en HTML side.



Fig. 1 - Sød Panda som browser bambus! © National Geographic

According to Goodreads, [Turing Award Winner Edsger Dijkstra](#), has stated that " If debugging is the process of removing software bugs, then programming must be the process of putting them in. "

Du skal dog først kende til [HTML](#).

Forfatter af Brian Nielsen

HTML Ex: Figurer, billeder, special tegn

```
<figure>
    
    <figcaption>Fig. 1 -
        Sød Panda som browser bambus!    &#169; National Geographic
    </figcaption>
</figure>
```

Figurer har også en titel / figcaption

Billeder bør altid have en "alt" beskrivelse afh. skærmlæsere

- Reserverede tegn "<>&" kan skrives "Entity": &streng; fx "
- [1000 vis af andre special tegn](#)

IWP Demo Side is a crude HTML page made for demo purposes
Indholdsfortegnelse:

- [Introduktion](#)
 - [HTML](#)
 - [JavaScript](#)
- [to do](#)

Ingen bemærkninger

Introduktion til Web-Programmering

Centrale elementer i web-teknologi er [HTML](#) og [JavaScript](#).

In bla bla.

Klient-side programmering kommer i [næste lektion](#).

Her er en god [reference til web-teknologi](#).

HTML

Med HTML kan du have sémantisk mark-up af et dokument, som vises i en web-browser. Etc. Etc. Etc.

HTML BODY

Indeholder vigtig information om dokumentet, som modtageren skal bruge for at forstå det.

JavaScript

JS kan bringes til at fungere sammen med en browser og interagere med en HTML side.



Fig. 1 - Sod Panda som browser bambus! © National Geographic

According to Goodreads, [Turing Award Winner Edsger Dijkstra](#), has stated that " If debugging is the process of removing software bugs, then programming must be the process of putting them in. "

Du skal dog først kende til [HTML](#).

Erfattet af Brian Nielsen

HTML Ex: citationer

According to [`<cite>`Goodreads</code>](#),
[a href="https://da.wikipedia.org/wiki/Turing-prisen"](https://da.wikipedia.org/wiki/Turing-prisen)
[Turing Award Winner](#) [a href="https://en.wikipedia.org/wiki/Edsger_W._Dikstra"](https://en.wikipedia.org/wiki/Edsger_W._Dikstra) Edsger Dijkstra:[/a](#) has stated that
[`<q cite="https://www.goodreads.com/author/quotes/1013817.Edsger_W_Dikstra">`](#)
[`If debugging is the process of removing software bugs,`](#)
[`then programming must be the process of putting them in.`](#)
[`</q>`](#)

- `<cite>` angiver artiklen, der er citeret
- `<q>` er citatet; dets `cite` attribut udpeger citatets kilde

HTML Tabel Ex Scoreboard

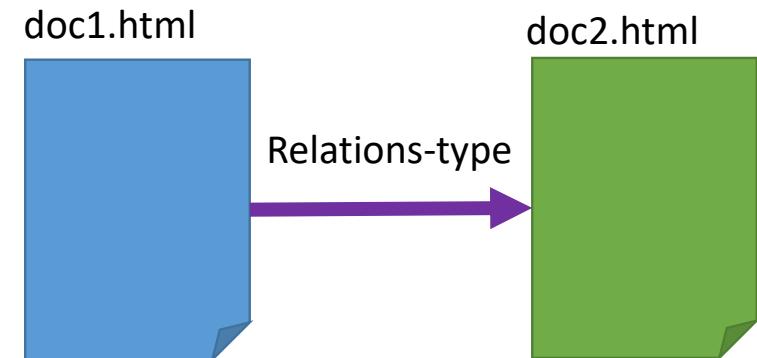
Yatzy Scores		
Peter		
Round Name	Dice	Score
1s	    	0
2s	    	0
3s		0
4s		0

- Tabel har en **caption**, **thead**, **tbody**
- Heading kolonner angives med "table head" **<th>**
- Rækker angives med "table row" **<tr>**
- Data kolonner angives med "table data" **<td>**
- **class** attribut bruges af stylesheet til at højre/venstre stille tekst, og give grøn baggrund til special runder
- Terninger er bare separate billeder
- Terninger samles i en "container" enhed: **span** (alt. **div**)

```
<table id="scoretable">
  <caption> Yatzy Scores </caption>
  <thead>
    <tr>
      <th colspan="3" > Peter </th>
    </tr>
    <tr>
      <th>Round Name </th>
      <th> Dice </th>
      <th>Score</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td class="left-text"> 1s </td>
      <td> <span>
          
          
          ...
        </span>
      </td>
      <td class="right-text"> 0 </td>
    </tr>
    ...
    <tr class="row-fill"> <td>Sum</td>...</tr>
    ...
  </tbody>
</table>
```

Lidt mere om hypertext "relationer"

- <link> eller rel attribut
 - rel="stylesheet"
 - Udpeger stil-regler
 - rel="author"
 - rel="alternate"
 - Fx en mobil udgave
 - rel="hreflang"
 - Forskellige sproglige versioner
 - rel="prev", rel="next"
 - Kapitlerne i en bog
 - ...
- Bruges af programmer som behandler hypertext dokumenter, fx søgemaskiner.
- Information i meta-elementet og links/relationer giver mulighed for søgemaskine optimering ("SOE")



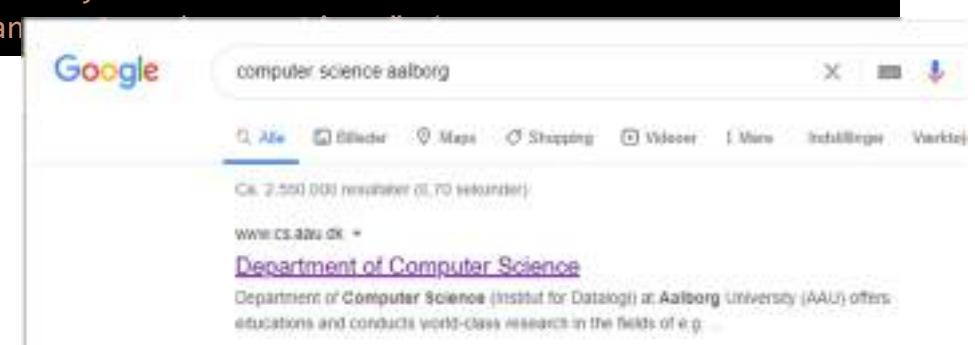
```
<a href="https://www.cs.aau.dk/~bnielsen" rel="author">  
Forfattet af Brian Nielsen</a>  
  
<link rel="alternate" hreflang="es"  
      href="https://www.example.es" >
```

Mere om HTML Head

```
<head>
  <meta charset="UTF-8">
  <title>IWP Test</title>
  <link rel="stylesheet" href="style.css">
  <link rel="author" href="http://www.cs.aau.dk/~bnielsen">
  <link rel="alternate" hreflang="es" href="https://www.example.es" >
  <meta name="copyright" content="Brian Nielsen" >
  <meta name="description" content="EN lille IWP Demo, som viser nogle interessante aspekter af HTML">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
...

```

```
<meta name="description" content="Department of Computer Science (Institut for Datalogi) at Aalborg University (AAU) offers educations and conducts world-class research in the fields of e.g. embedded software systems, data-intensive systems, artificial intelligence and human
```



Validering af HTML dokumenter og CSS

- Hensigten bag HTML elementerne er beskrevet i
 - På <abbr>MDN title=“Mozilla Developer Network“</abbr>
 - Uddybet og specificeret i <https://html.spec.whatwg.org/>
 - (The Web Hypertext Application Technology Working Group (WHATWG) is a community of people interested in evolving the web through standards and tests.)
- Der findes online validatorer, som kan checke for syntax og enkelte semantiske fejl i HTML dokumenter og CSS
 - <https://validator.w3.org/> TOP TIP!

CSS Ex. Layout og farvelader

IWP Demo Side is a simple HTML page made for demo purposes

Inneholdsforsmål:

- + [Introduktion](#)
 - * [HTML](#)
 - * [JavaScript](#)
- * [Index](#)

Introduktion til Web-Programmering

Centrale elementer i web-udvikling er [HTML](#) og [JavaScript](#).
Istedsat er der ikke et bestemt sprog til at udvikle med.
Klient-side programmering koncentrerer sig mest på [JavaScript](#).
Det er en god [referanse](#) til web-udvikling.

HTML

Med HTML kan du lave semantisk mark-op af et dokument, som vises i et web browser. Et. Ek. Ek.

HTML BODY

Indholder vigtig information om dokumentet, som modtageren skal høje for at fåret det.

JavaScript

JS kan bringe id til fulgere sammen med en højre- og venstre side i et HTML side:



Fig. 1 - Sæt Panda som beweist bambus! © National Geographic

According to Goodfellow, [Tanner Award Winner Editor Dr. Gupta](#), has stated that "If debugging is the process of removing software bugs, then programming must be the process of putting them in."

<https://www.goodfellow.com/india/quote/151111> (Citator: M. Balakrishna)

Du skal dog først kendte til [HTML](#).

Ingen
bemærkninger

```
body {  
    display: grid;  
    grid-template-areas:  
        "h h h"  
        "n m a"  
        "f f f";  
    grid-template-rows: 70px 1fr 60px;  
    grid-template-columns: 20% 1fr 15%;  
    grid-row-gap: 10px;  
    grid-column-gap: 10px;  
    height: 100vh;  
    margin: 0;  
}  
/*general coloring and layout */  
header, footer, main, nav, aside {  
    padding: 0.5em;  
    background: lightgray;  
    margin: 0.5em;  
    color: black; box-shadow: 0 0 0.25em black;  
}  
header { grid-area: h; }  
footer { grid-area: f; }  
main { grid-area: m; }  
nav { grid-area: n; }  
aside { grid-area: a; }  
header{  
    background-color: lightgreen;  
}  
  
section{  
    margin: 1em;  
    border: thin dashed; padding: 1em;  
    background: rgb(190, 190, 190);  
}  
#introduction{ background: rgb(248, 229, 55);}  
main{  
    /* try to uncomment this */  
    /*display: flex; */  
}
```

HTML Input og Forms

HTML Formularer

- HTML5 tilbyder et større antal elementer typer til bruger input:
 - <button> <datalist> <fieldset> <input> <form> <label> <legend> <meter> <output> <optgroup> <progress> <textarea> <select>
 - <input> undertyper: button, checkbox, radio, range, text, number, submit ,...
- Kan bruges enkeltvist til input til JS program
- Kan samles i en formular <form> </form> med en “submit” knap
 - Kan have en “action” URL og en “http metode” attribut, som normal får browser til at sende formular data i et http request, og som respons forventer en ny side med resultatet
- Alternativt, kan formularen kan også behandles af JS:
 - Validering og fejl-retning inden formularen indsendes
 - Kun JS behandler dataene

A screenshot of a web-based personal information form. The title is "Personal information:". It contains three text input fields: "Name" with value "Mickey", "Height (cm)", and "Weight (kg)". Below the fields is a "Record" button.

Eks. på elementer til bruger-input

Eksempler fra [MDN]

forslagsliste

<datalist>

Choose a flavor:

A screenshot of a dropdown menu. The placeholder text is "Choose a flavor:". Below it is a list of five items: Chocolate, Coconut, Mint, Strawberry, and Vanilla.

- Chocolate
- Coconut
- Mint
- Strawberry
- Vanilla

options

<input type="radio">

Select a maintenance drone:

- Huey
- Dewey
- Louie

menuer

<select>

A screenshot of a dropdown menu. The placeholder text is "Please choose an option...". Below it is a list of six items: Dog, Cat, Hamster, Parrot, Spider, and Goldfish. The first item, "Dog", is highlighted with a blue background.

- Please choose an option--
- Please choose an option
- Dog
- Cat
- Hamster
- Parrot
- Spider
- Goldfish

options

<input type="checkbox">

Choose your monster's features:

- Scales
- Horns

Tekst input/redigering

<textarea>

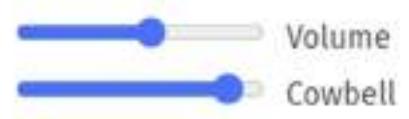
Tell us your story:

It was a dark and stormy
night...

ranges

<input type="range">

Audio settings:



Validering af inputs

VIGTIGT!

- Validering af input data på klient-side
 - Giver bruger hurtigt feedback (uden server kommunikation)
 - Sparer belastning på server, da den håndterer færre ugyldige anmodninger
 - Sparer båndbredde / data trafik: husk mobildata/roaming kan koste \$€
 - Sparer energi – især vigtigt på mobile / batteridrevne platforme. Kommunikation koster relativt meget energi. CPU typisk meget billigere
- Validering på server-side
 - **Server skal ALTID validere data fra kilder den ikke har 100% tillid fra**
 - Næsten alt eksternt
 - Klienter kan sende hvad som helst
 - være ondsindede
 - fejl-behæftede

HTML5 har faciliteter til basal validering, se https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation
Derudover foretages det vha. JS (kan også give målrettede fejlbeskeder).

Klient side Input Validering

- Kan laves i HTML5 ["forms validation"](#) med dens validerings attributter på input elementer
 - <label for="pauseInput">Indtast pauselænge i sekunder :</label>
<input id="pauseInput" type="number" min="1" max="1200" required name="pause">
- Kan også laves af JS i på klienten
 - Events: "input" og "submit" som fyrer når et felt er indtastet eller form submitted
 - Input felter har normalt en "value" property, som JS kan inspicere
- JS og HTML validering kan kombineres:
 - Fyrer "invalid" event, hvis input ikke matcher krav
pauseInputElement.addEventListener("invalid", invalid);

Formular til ny BMI registrering

IWP BMI-recorder

Personal information:

Name: Mickey

Height (cm):

Weight (kg):

```
<form id="bmiForm_id" action="bmi-records" method="post">
<fieldset>
    <legend>Personal information:</legend>

        <label for="name_id"> Name</label>
        <input type="text" name="name" id="name_id" value="Mickey" required minlength="1" maxlength="30">

        <label for="height_id"> Height (cm):</label>
        <input type="number" name="height" id="height_id" value="" min="1" max="300" required>

        <label for="weight_id"> Weight (kg):</label>
        <input type="number" name="weight" id="weight_id" value="" min="1" max="300" required>

        <input type="submit" id="submitBtn_id" value="Record">
</fieldset>
</form>
```

- Formularens action angiver hvilken URL formularen skal sendes til (nomalt relativt til den aktuelle side: <http://localhost:3000>)
- Formularens action angiver hvilken HTTP metode formularen skal sendes med
- I stedet for placeholder kunne vi have
`value="et default navn"`
- `<input>`'s name attribut og aktuelt input sendes som par (name,value) til server
- `<label>`s for attribut: binder label til `<input>`'s id attribut.
- Validering:
 - Required
 - Input type
 - Min og max (værdier / længder)

Forms: Method og Action

- Indsendelsesmetode:
 - ”Method” er typisk GET eller POST
 - Se lektion om HTTP
- Action:
 - Den ressource som forespørgslen skal læse/ændre.
 - En identifikation af den kode (handling), som server skal udføre når den modtager forespørgslen

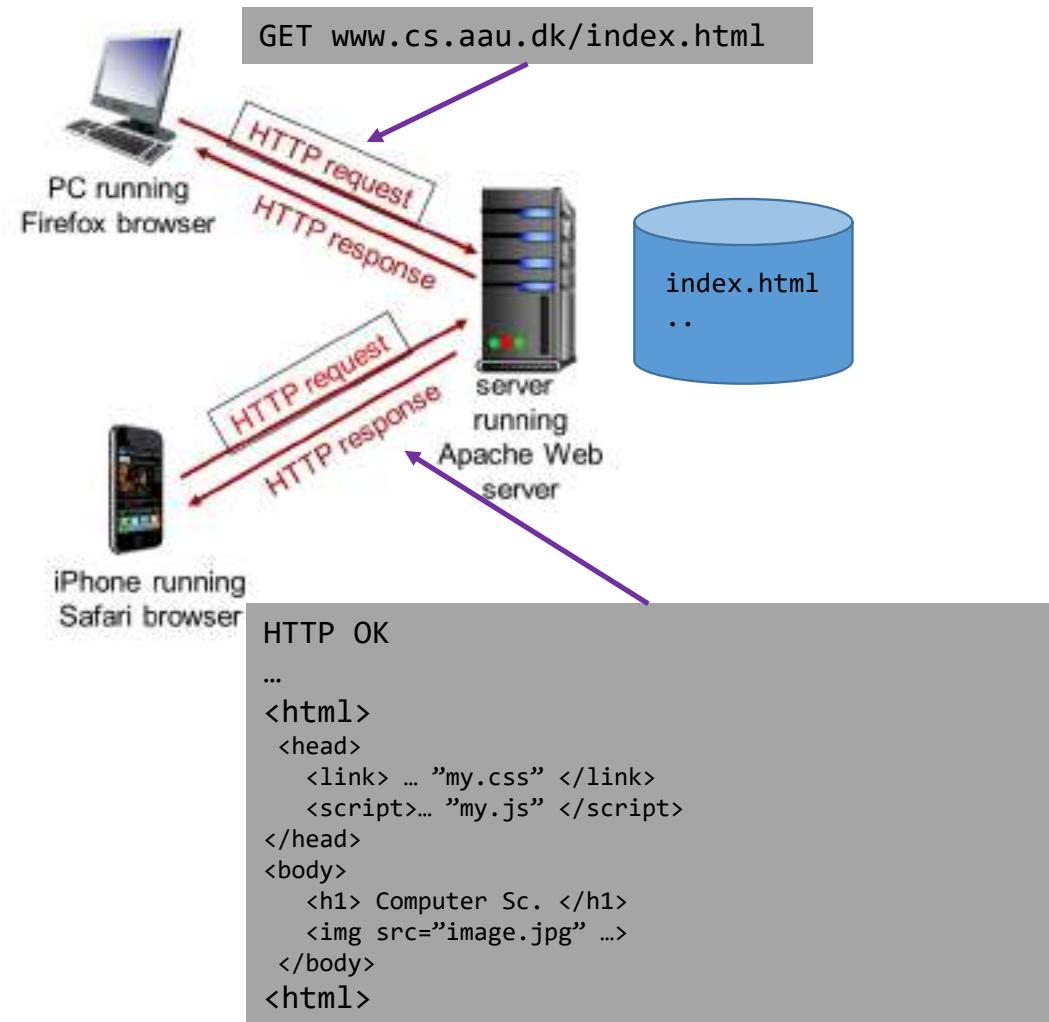
Grundlæggende HTTP

HyperText Transfer Protocol

Simpel HTTP Eksempel Scenarie: ressource er her statisk html fil

HTTP: hypertext transfer protocol

- applikationslags-protokol til web-trafik
- client/server model:
- **client:** program (typisk browser)
 1. Sender forespørgsel (vha. HTTP GET), om en web side
 2. Afventer respons
 3. Parser respons og optegner siden
 4. Kører evt. skridt 1-3 igen (sideløbende) for at henter nødvendige indlejrede ressourcer (billeder, js-scripts, style sheets)
- **server:**
 1. Afventer
 2. Modtager HTTP forespørgsel fra klient
 3. Behandler den, og læser filen,
 4. Sender HTTP respons med sender fil-indhold til klienten
- Server og ressourcen angives ved et Uniform Ressource Identifier, normalt URL



Det eksakte forløb afhænger af HTTP protokol version
(jfv. Forelæsning om applikationslagsprotokoller)

HTTP Generel Oversigt

HTTP: hypertext transfer protocol

- applikationslags-protokol til web-traffik
- client/server model
 - **client:** program (typisk browser) som anmoder (vha. HTTP), modtager svaret, og behandler svaret
 - **server:** Web server modtager anmodningen, beregner svaret, og sender (vha HTTP) resultatet til klienten
- Server er identificeret vha.
 - en IP-adresse (eller DNS navn) og et port nummer på serveren,
 - port nummer: 80 eller (443 for HTTPS)

Alternativt klient program: "CURL"



Server er værtsmaskine for et antal **ressourcer**:
en eller anden "ting" på serveren

- Information, data i db
- Web sider
- Filer
- Billeder
- Beregninger
- Services
- IoT Device / fx temperatur måling

En ressource har en eller flere **repræsentationer (dokument)** som afspejler dens aktuelle indhold/tilstand på en måde som kan sendes over netværk

- Metadata
- Sekvens af bytes
- Når server modtager en HTTP forespørgsel, kører den kode, som beregner svaret ("repræsentationen")
 - Dynamiske web sider, DB opslag
 - Program som server ejerne har lavet (fx jeres program)
 - REST APIs

HTTP Besked

- En besked (både request og respons) består af 2 dele:



- Header: formål med beskeden, og kontrol information om beskeden, protokol flag, optioner
- Body: de egentlige data, der udveksles
 - Data kan være hvad som helst (html, json, billeder, fil-data,)

HTTP Beskeder



HTTP/1.1 Metoder (a.k.a "verbs")

- **GET:**

- Anmoder om overførsel af en repræsentation af den ønskede ressource
- "Læsning"

- **POST:**

- Udfør en resource-specifik behandling på den ønskede ressource
- "Ændring"
 - Fx, tilføje data til ressourcen (fx indtastet i en "HTML form")

- **PUT:**

- Oprette (eller erstatte) tilstanden på den ønskede ressource) i sit hele så den svarer til den medsendte repræsentation

- **DELETE**

- Sletter ressourcen (eller fjerner forbindelsen imellem URL navn og ressourcen)
- HEAD, PATCH, CONNECT, OPTIONS, TRACE

Når serveren modtager et request med en given metode kalder den en **funktion** som "du" eller (web-server programmøren) har lavet!

"Spilleregler" for web-arkitekturen forudsætter at **funktionen** (som udføres på server siden) skal respektere hensigten bag HTTP metoderne

- Fx må GET ikke ændre ressourcen.
- Caching, forudindlæsning ("pre-fetching")

Sikre metoder: udførslen af **funktionen** må ikke "skade" ressourcen, eller give unormal stor belastning på server

- Klient kan gentage dem
- GET, HEAD, OPTIONS, TRACE skal programmeres så de er sikre
- Et PUT umiddelbart efterfulgt af et GET skal give den værdi der netop er oprettet
- Begrebet "**Idempotente**" operationer introduceres senere

DIN KODE SKAL RESPEKTERE DISSE

Content-Type header

- Indholdet af HTTP Body bestemmes af “Content-Type” feltet i HTTP headeren
 - Modtageren (fx browser) bruger den til at afgøre hvordan dataene skal behandles
 - Content-type skal der angives korrekt: MIME type Multi-purpose Internet Mail Extensions
 - Fil-endelser kan bruges af sender til at “gætte” en MIME-type

Generelt format:
type/subtype;parameter=value

| Fil Endelse | Dokument art | MIME Type | Eksempel |
|---------------|--|-----------------------------------|---------------------------|
| .bin | Any kind of binary data | application/octet-stream | text/plain; charset=UTF-8 |
| .css | Cascading Style Sheets (CSS) | text/css | |
| .htm
.html | HyperText Markup Language (HTML) | text/html | |
| | Html formular data | application/x-www-form-urlencoded | |
| .jpeg
.jpg | JPEG images | image/jpeg | |
| .js | JavaScript | text/javascript | |
| .json | JSON format | application/json | |
| .mjs | JavaScript module | text/javascript | |
| .mp3 | MP3 audio | audio/mpeg | |
| .mpeg | MPEG Video | video/mpeg | |
| .png | Portable Network Graphics | image/png | |
| .pdf | Adobe Portable Document Format (PDF) | application/pdf | |

HTTP Responskoder

1. 100-199: Informativt svar
2. 200-299: Succesfuld respons
3. 300-399: Omdirigering (Redirects)
4. 400-499: Client fejl
5. 500-599: Server fejl

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

102 Processing

- Server arbejder, tålmodighed, tak

200 OK

- Anmodning gik godt!
 - GET: resourcen er hentet og følger i beskedens “body”
 - POST: handling gik godt og resultatet følger i beskedens “body”

301 Moved Permanently

- Det forespurgte objekt er blevet flyttet; ny placering følger senere i beseden

400 Bad Request

- Forespørgslen kunne ikke forståes af server

404 Not Found

- Den forespurgte resource kunne ikke findes på denne server

505 HTTP Version Not Supported

HTTP og FORMS: GET

- **GET:** Kan bruges ved læsning af ressourcen
- Søge parametre kan sendes som en del af URL'en (i "query string")
 - kan ses direkte af bruger i browser
 - Indgår i browser historik, kan bogmærkes
 - Respons kan cache's
 - Query string kan kun være 2048 karakter lang
 - URL-encodes
 - Kan ikke indeholde binær data
- Form data sendes som i "query string"
 - Bemærk "name" attribut til identifikation af parametrenes navne

IWP Yatzy Game

Configure Game:

Name: Peter

Number of Dice: 6

New Game

```
<form action="http://httpbin.org/get" method="get">
  <input type="text" id="name_id" name="name" />
  <input type="number" id="diceCount_id" name="diceCount" />
  <input type="submit" value="New Game">
</form>
```

Not secure | http://httpbin.org/get?name=Peter&diceCount=6

GET /get?name=Peter&diceCount=6 HTTP/1.1
Host: httpbin.org

...

http header

(httpbin.org er et web-site til test af web-requests)

HTTP og FORMS: POST

- **POST:** Anvendes normalt til ændring af ressourcen (se mere specifikt slides http "verbs")
- Form data sendes som del af request-body
 - Url-encoded, jfv. Content-Type
 - Indgår ikke i historik, cache
 - Gensendes ikke af browser uden advarsel (js kan dette)
 - Kan (kun) under specielle omstændigheder anvendes til læsning
- (Fil-upload: skal overføres som content type "multipart/form-data")

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type>

IWP Yatzy Game

Configure Game:

Name: Peter

Number of Dice: 6

New Game

```
<form action="http://httpbin.org/post" method="POST">
  <input type="text" id="name_id" name="name" value="Peter">
  <input type="number" id="diceCount_id" name="diceCount" value="6">
  <input type="submit" value="New Game">
</form>
```

POST /post HTTP/1.1
Host: http://httpbin.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 22

name=Peter&diceCount=6

http header

http body

```
<form action="/" method="post" enctype="multipart/form-data">
  <input type="text" name="description" value="some text">
  <input type="file" name="myFile">
  <button type="submit">Submit</button>
</form>
```

Lidt om CSS

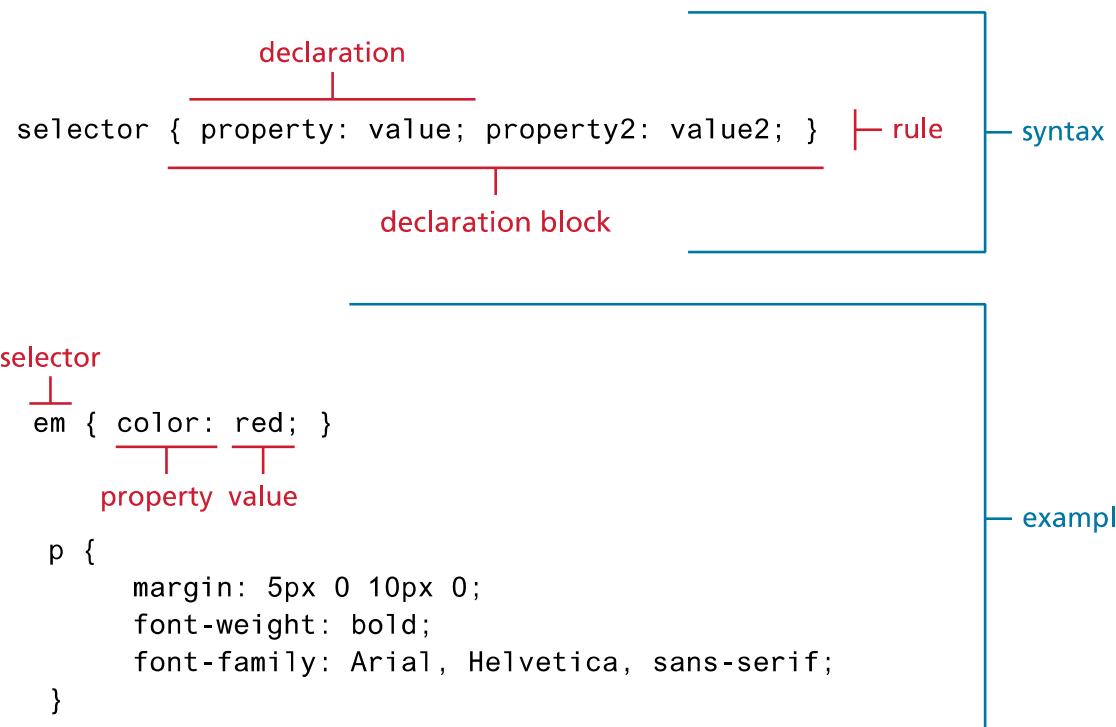
Her er rocker-udgaven! Ellers kan det blive flere virkelig lange forelæsninger

Cascading Style Sheets

- CSS er en W3C standard til at beskrive **udseendet** af HTML elementer
- Tildele egenskaber (properties) som font, farver, størrelser, kanter, baggrunde, og positionering til elementerne på en side
 - Forbedret **kontrol** over formattering.
 - Forbedret **vedligehold** af et site: ret kun ét sted (i css filer).
 - Forbedret **tilgængelighed**: tekst-til-tale for svagtseende forstyrres ikke af layout annotationer
 - Forbedret download **hastighed** (css filer gælder typisk for et helt site, ikke pr side: genbruges fra cache)
 - Forbedret **output fleksibilitet** (responsivt design): kan nemmere tilpasses forskellige klienters skærmstørrelse
- Kan være frustrerende (**historisk uensartet impl i browsere**).
- Layout delen ikke altid intuitiv

CSS regler

- CSS dokument = en liste af stil regler
- En stil regel består af
 - **selector** som udpeger de HTML elementer, som skal påvirkes.
 - En erkæringsblok med en liste af **property:value** par
- Selector kan skrives på flere måder
- Der er mange 100-vise af ”properties”



Fx Alle bliver røde og <p> elementer fede med Arial font

CSS properties

Ikke udømmende; CSS er et "rigt" sprog til at udtrykke sådanne egenskaber, og koble dem sammen

| Property Type | Property |
|----------------------|--|
| Fonts | font
font-family
font-size
font-style
font-weight
@font-face |
| Text | letter-spacing
line-height
text-align
text-decoration*
text-indent |
| Color and Background | background
background-color
background-image
background-position
background-repeat
box-shadow
color
opacity |
| Borders | border*
border-color
border-width
border-style
border-top, border-left, ...*
border-image*
border-radius |

| Property Type | Property |
|---------------|--|
| Spacing | padding
padding-bottom, padding-left, ...
margin
margin-bottom, margin-left, ... |
| Sizing | height
max-height
max-width
min-height
min-width
width |
| Layout | bottom, left, right, top
clear
display
float
overflow
position
visibility
z-index |
| Lists | list-style*
list-style-image
list-style-type |
| Effects | animation*
filter
perspective
transform*
transition* |

Selectors (VIGTIGT)

CSS har et rigt sprog til at udpege HTML elementer

- **Type:** udvælger elementer med et givet element navn
- **Class:** udvælger alle elementer med det givne klasse-attribut ":"
- **ID:** udvælger ud fra id-attribut "#"
- **Attribut:** udvælger [attribut] {...}
- **Pseudo:** udpeger på baggrund af et elements tilstand
`a:link, a:visited, :focus, :hover, :active ...`
- **Kontekstuel:** baseret på naborelation i HTML-træet.

| Selector | Example | Learn CSS tutorial |
|-----------------------------|--------------------------------|------------------------|
| Type selector | <code>h1 { }</code> | Type selectors |
| Universal selector | <code>* { }</code> | The universal selector |
| Class selector | <code>.box { }</code> | Class selectors |
| Id selector | <code>#unique { }</code> | ID selectors |
| Attribute selector | <code>a[title] { }</code> | Attribute selectors |
| Pseudo-class selectors | <code>p:first-child { }</code> | Pseudo-classes |
| Pseudo-element selectors | <code>p::first-line { }</code> | Pseudo-elements |
| Descendant combinator | <code>article p</code> | Descendant combinator |
| Child combinator | <code>article > p</code> | Child combinator |
| Adjacent sibling combinator | <code>h1 + p</code> | Adjacent sibling |
| General sibling combinator | <code>h1 ~ p</code> | General sibling |

[https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors#in this module](https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors#in_this_module)

CSS Ex. Tabel farvning

The screenshot shows a Yatzy scorecard titled "IWP Multi Yatzy". The table has columns for "Round Name", "Dice", and "Score". A red arrow points from the "Score" header cell to the text "id='scoretable'". Another red arrow points from the "Score" cell in the first row to the text "Header celler indenfor id='scoretable'". A third red arrow points from the "Score" cell in the last row to the text "class='rowfill'". A fourth red arrow points from the top of the table to the text "<h1>". A fifth red arrow points from the "Total Score" cell to the text "Mus svæver over rækken".

| Yatzy Scores | | |
|-----------------|-----------|-------|
| Peter | | |
| Round Name | Dice | Score |
| 1s | 1 1 1 1 1 | 1 |
| 2s | | 0 |
| 3s | | 0 |
| 4s | | 0 |
| 5s | | 0 |
| 6s | | 0 |
| Sum | | 1 |
| Bonus | | 0 |
| 1 Par | | 0 |
| 2 Pars | | 0 |
| Three Identical | | 0 |
| Four Identical | | 0 |
| Little Straight | | 0 |
| Big Straight | | 0 |
| House | | 0 |
| Chance | | 0 |
| Yatzy | | 0 |
| Total Score | | 1 |

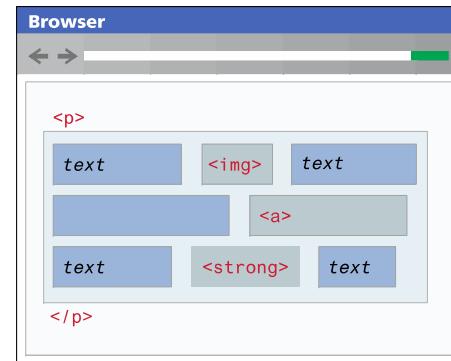
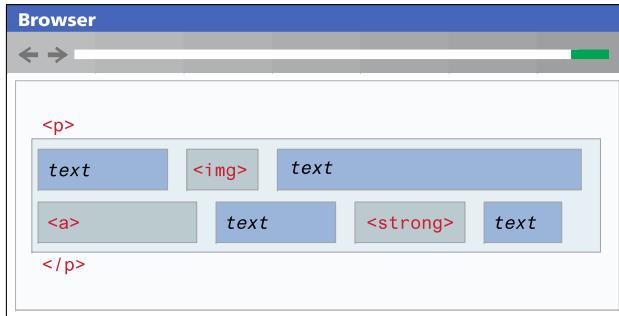
```
h1 {  
    background-color: lightgreen;  
}  
  
#scoretable {  
    font-family: Arial, Helvetica, sans-serif;  
    border-collapse: collapse;  
    border: 2px solid green;;  
    color: black;  
}  
  
#scoretable th {  
    padding-top: 12px;  
    padding-bottom: 12px;  
    text-align: center;  
    color: green;  
    border: 2px solid green;  
}  
  
#scoretable .row-fill {  
    background-color:#a1dfa3;  
    border: 2px solid green;;  
    padding: 8px;  
}  
#scoretable tr:hover {background-color: #ddd;}  
...
```

Kaskade-regler: Hvordan styles interagerer

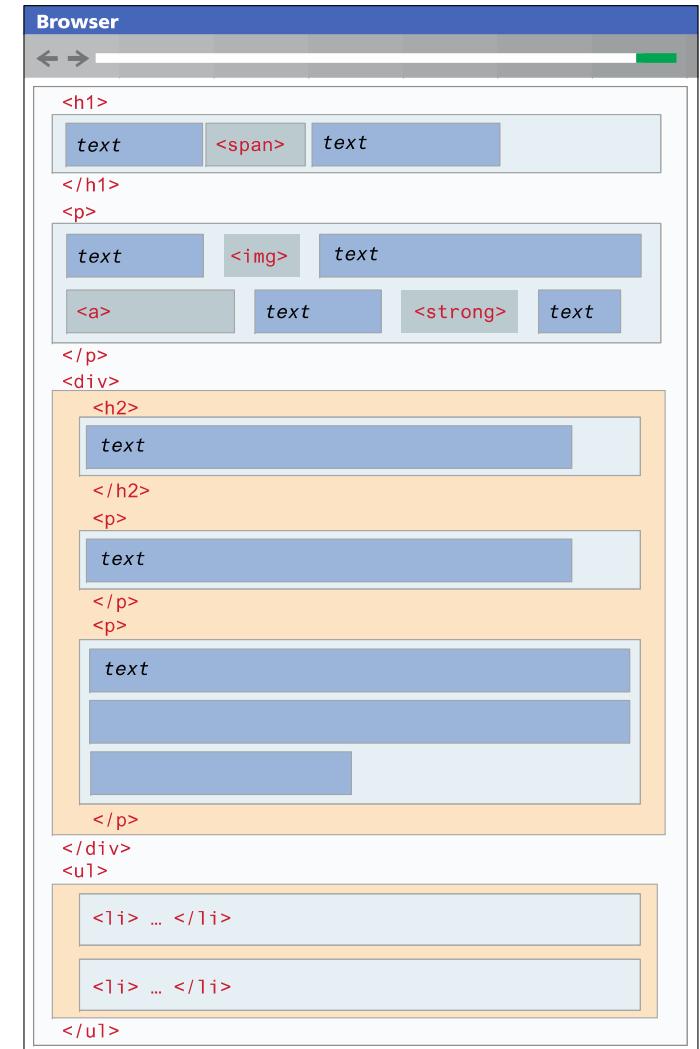
- “Cascade” i CSS henviser til algoritmen for hvordan stil-reglerne sættes sammen til en rendering af et element.
 - Definerer også hvordan overlappende/konfliktende regler håndteres. 1 er højst prioritet
 - 1. Oprindelse: Der kan være flere stylesheets i spil
 1. Slut-bruger kan have eget style-sheet (fx syns-handicappet)
 2. Web-forfatterens style-sheet
 3. Browser's indbyggede default;
 - 2. Specificitet: Des mere specifik selektor, des højere prioritet
 - 3. Regler med samme specificitet: erklæringsrækkefølge af regler er betydende så sidst erklærede vinder
 - 4. Nedarvning : ”arvelige” CSS egenskaber påført et HTML element videreføres automatisk også på elementets børn.
 - Arvelige CSS egenskaber: fx font, color, text
 - Ikke-arvelige egenskaber: fx layout, størrelser, kanter, baggrunde,
- https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Cascade_and_inheritance
- <https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade>
- <https://blog.logrocket.com/how-css-works-understanding-the-cascade-d181cd89a4d8/>

CSS positionering og layout: Normal flow

- HTML dokumentet læses fra top mod bund
- **Block-niveau elementer** (`<p>`, `<div>`, `<h2>`, ``, og `<table>`) får deres egen linie: top mod bund
 - Nogle kan nestes
 - I forældre elementets ramme
- **Inline elementer** (``, `` vises på samme linie venstre mod højre
 - Resize=>rewrap

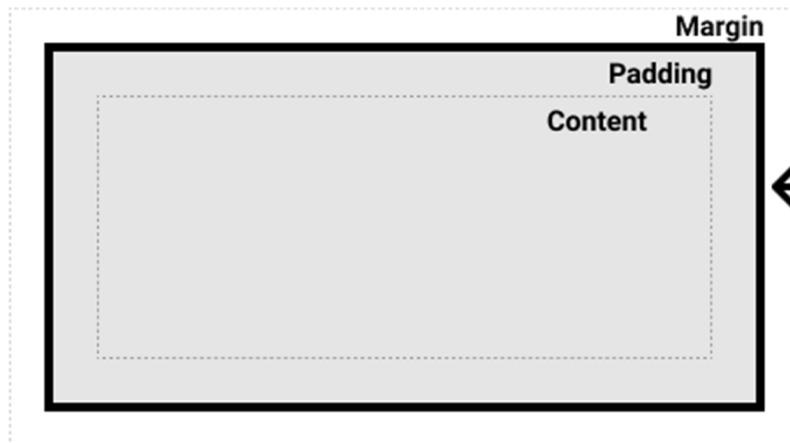


- `` og `<div>` anvendes hyppigt! Tillader meget fin-kornet kontrol af layoutet
- SEMANTISK MARKUP BØR ANVENDES HVOR MULIGT

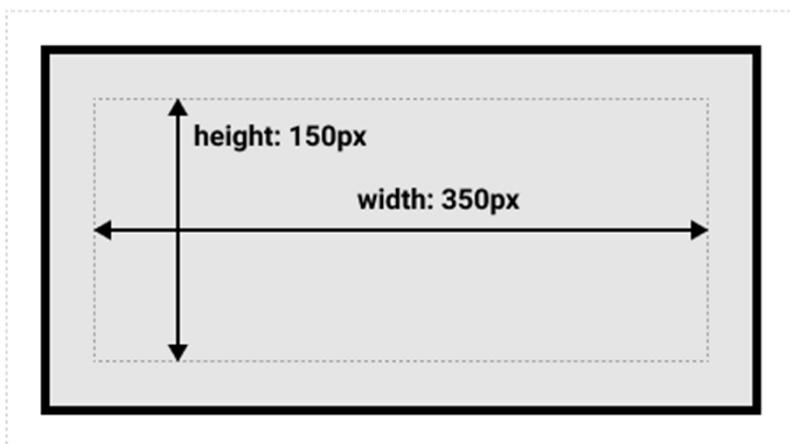


CSS positionering og layout: box-modellen

- Alle HTML elementer renderes i en "box"



```
1 .box {  
2   width: 350px;  
3   height: 150px;  
4   margin: 10px;  
5   padding: 25px;  
6   border: 5px solid black;  
7 }
```

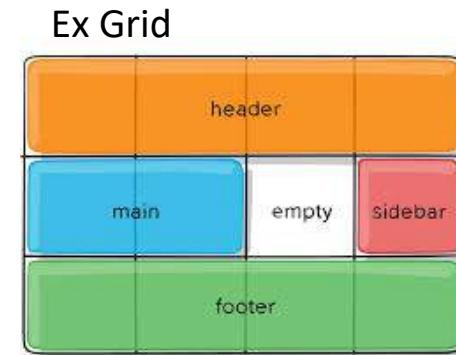


"Inspector" i "developer tools"



CSS positionering og layout: styring af bokse

- CSS indeholder en række mekanismer til layout af bokse
- Display egenskaben:
 - Vælger layout for elementets børn, fx
 - display: flow Normalt flow layout
 - display: grid Opstilling på basis af en matrice/"skelet"
 - display: flex Automatisk opstilling vertikalt eller horisontalt: alignment, justering af mellemrum, wrapping følger
 - Se også https://www.w3schools.com/css/css3_flexbox.asp
 - display:block, display:inline Vælger om elementet skal behandles som inline eller block
 - Om elementet skal vises eller ej (styres typisk via Javascript)
 - #div1{display: none}
 - #div1{visibility:hidden} (optager plads)
- Position egenskaben: (x,y) koordinater
 - Fin-justering af et elements position ifht. forventet placering
 - Adskillige modes(fixed, relativt, absolut,sticky..)



CSS Ex. Layout og farvelader

The screenshot shows a web page with the following structure:

- Header:** A light green header bar with the text "HTTP Demo Page".
- Content Outline:** A sidebar on the left containing a tree view of the page's structure:
 - Introduktion
 - HTML
 - JAVASCRIPT
 - to do
- Main Content Area:** A large yellow box titled "Introduktion til Web-Programmering". It contains text about the central elements of web technology being HTML or JavaScript, and links to CSS and CSS3.
- Section:** A grey box titled "HTML". It contains text about HTML being the semantic mark-up of a document, visible in a web browser.
- Section:** A grey box titled "HTML BODY". It contains text about the body element holding important information for the document.
- Section:** A grey box titled "JavaScript". It contains text about JavaScript being used to connect a browser and an image on a page.
- Image:** An image of a panda bear.
- Caption:** "Fig. 1 - Sad Panda som betorer bambu! © National Geographic"
- Text:** A quote from Goodwin: "According to Goodwin, [Timothy J. Goodwin](#) states that 'If debugging is the process of removing software bugs, then programming must be the process of putting them in.' http://www.goodwillassociates.com/2011/07/14/fig-1_djiguru/ Du skal dog først kende til [HTML](#).
- Footer:** A light green footer bar with the text "Exerciser af Brian Nielson".

```
body {  
    display: grid;  
    grid-template-areas:  
        "h h h"  
        "n m a"  
        "f f f";  
    grid-template-rows: 70px 1fr 60px;  
    grid-template-columns: 20% 1fr 15%;  
    grid-row-gap: 10px;  
    grid-column-gap: 10px;  
    height: 100vh;  
    margin: 0;  
}  
/*general coloring and layout */  
header, footer, main, nav, aside {  
    padding: 0.5em;  
    background:lightgray;  
    margin: 0.5em;  
    color: black; box-shadow: 0 0 0.25em black;  
}  
header { grid-area: h; }  
footer { grid-area: f; }  
main { grid-area: m; }  
nav { grid-area: n; }  
aside { grid-area: a; }  
header{  
    background-color: lightgreen;  
}  
section{  
    margin:1em;  
    border: thin dashed; padding:1em;  
    background: rgb(190, 190, 190);  
}  
#introduction{ background: rgb(248, 229, 55);}  
main{  
    /* try to uncomment this */  
    /*display: flex;*/  
}
```

Angivelse af stil-regler

1. Som ekstern .css text fil
 - den normale, anbefalede metode.
2. Indlejret metode: i HTML som style element i header
 - Ved kort style-sheet
3. Direkte metode via “style” attribut på HTML elementet).
 1. Anvendelse bør minimeres!
 2. Kan være handy til test af ny stil
 3. Gælder kun på det konkrete element

```
<head> ...
  <link rel="stylesheet" href="styles.css">
</head>
```

```
<head> ...
  <style>
    h1 { font-size: 24pt; }
    h2 {
      font-size: 18pt;
      font-weight: bold;
    }
  </style>
</head>#
<body>
```

```
<h2 style="font-size: 24pt; font-weight:bold;">
Karakterer</h2>
```

CSS Layout Strategier

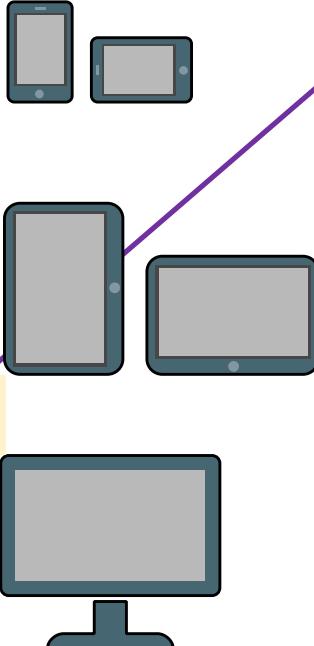
- **Fast layout:**
 - Web-designer vælger én ideal/typisk skærmstørrelser og tilpasser designet til den.
 - Simplere og mere forudsigtigt resultat (for den ene valgte størrelse)
- **Liquid-layout:**
 - Alle mål angives relativt i %, og browser renderer til den givne klient
 - Tricky at få pænt til alle tilfælde
- **Responsivt-web-design**
 - Layout tilpasser sig forskellige klienters (desktop, tablet, mobil, printere) vindue- og skærm-størrelser, og stående eller liggende (portrait/landscape)
 - Mere avanceret, mere arbejde
 - CSS har mekanismer til formålet
 1. Indstilling af klientens synsfelt (“viewports”) via `<meta>` element
 2. Lav versioner af CSS afhængigt af klientens “viewport” vha. forspørgsler på skærmstørrelse (“media queries”)
 3. Brug principper for liquid layouts, og skalér af billeder til størrelsen af “viewport”

Responsivt Web-design

- **Viewport:** området i en web-sider hvor indhold er synligt for brugeren

- For stor web-side: scroll-bars
- For lille: skærm udnyttes ikke
- Typisk optimeret indstilling:

```
<meta name="viewport" content="width=device-width,  
initial-scale=1.0">
```



styles.css

```
/* rules for phones */  
@media only screen and (max-width:480px)  
{  
    #slider-image { max-width: 100%; }  
    #flash-ad { display: none; }  
    ...  
}  
  
/* CSS rules for tablets */  
@media only screen and (min-width: 481px)  
and (max-width: 768px)  
{  
    ...  
}  
  
/* CSS rules for desktops */  
@media only screen and (min-width: 769px)  
{  
    ...  
}
```

- Medie forespørgsler

Instead of having all the rules in a single file, we can put them in separate files and add media queries to <link> elements.

```
<link rel="stylesheet" href="mobile.css" media="screen and (max-width:480px)" />  
<link rel="stylesheet" href="tablet.css" media="screen and (min-width:481px)  
and (max-width:768px)" />  
<link rel="stylesheet" href="desktop.css" media="screen and (min-width:769px)" />
```

CSS Resourcer, biblioteker, og værktøjer

- <https://css-tricks.com/>
- Der findes en række biblioteker (frameworks) med CSS des skabeloner
 - <https://learnlayout.com/frameworks.html>
- Der findes design værktøjer og CSS "generatorer"
 - <https://www.creativebloq.com/features/best-web-design-tools>



Råd om CSS til projektet

- KISS
 - Start med absolut minimal styling og layout, god funktionalitet
 - Start med fast layout til en "normal" desktop skærm
 - Ambitions-niveauet kan stige, men det kan være en tidsrøver
 - Start med vanilla JS/HTML/CSS, når i mestrer principperne i dette kan i overveje komponenter a la bootstrap.
 - Der er ikke specifikke læringsmål der går på fancy grafik og layout!
(men det må selvfølgeligt gerne se rimelig ud)

END

Internetværk og Web-programmering

Web-apps:

Multi-page vs. Single Page

Brian Nielsen

Distributed, Embedded, Intelligent Systems



Semantic markup?

```
<table id="scoretable">
    <caption>Table 1: BMI Classification for
Adults. From <cite>LEX <a
href="https://denstoredanske.lex.dk/body_mass_i
ndex">lex.dk</a></cite> </caption>
    <thead>
        <tr>
            <th> BMI range </th><th>Weight
Class</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>&lt;18.5
weight</td>
            <td>Under-

```



Table 1: BMI Classification for Adults. From LEX [lex.dk](#)

BMI range	Weight Class
<18.5	under-weight

```
<p><em>Table 1:</em> BMI Classification for</p>
<p> Adults. From <strong>LEX </strong></p>
<a href="https://denstoredanske.lex.dk/body_mass_index">lex.dk</a><br>

<table id="scoretatable">
<tr>
    <td> <b>BMI range</b> </td><td><b>Weight Class</b></td>
</tr>
<tr>
    <td>&lt;18.5</td><td>under-weight</td>
</tr>
...

```



Styling i stylesheets!!!

2 stil arter for web-applikationer

Klassiske "web-sider"

- Bruger udfylder og indsender "formular"
- Klient og server kommunikerer via HTML
- Begrenset 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 klient-side scripting til bruger-interaktion
- Klient programmers i Javascript
- 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

I dag: Klassiske Web-side applikationer

IWP BMI-tracker

Personal information:

Name: Mickey

Height (cm):

Weight (kg):

Submit

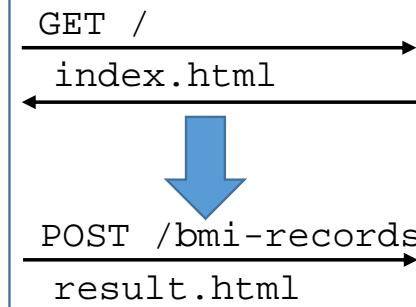


IWP BMI-tracker

Hi Mickey! Your BMI is 30.86. Since last it has changed 3.08!

Nyt selvstændigt html dokument
Evt. navigér tilbage og start forfra!

Klient (Browser)



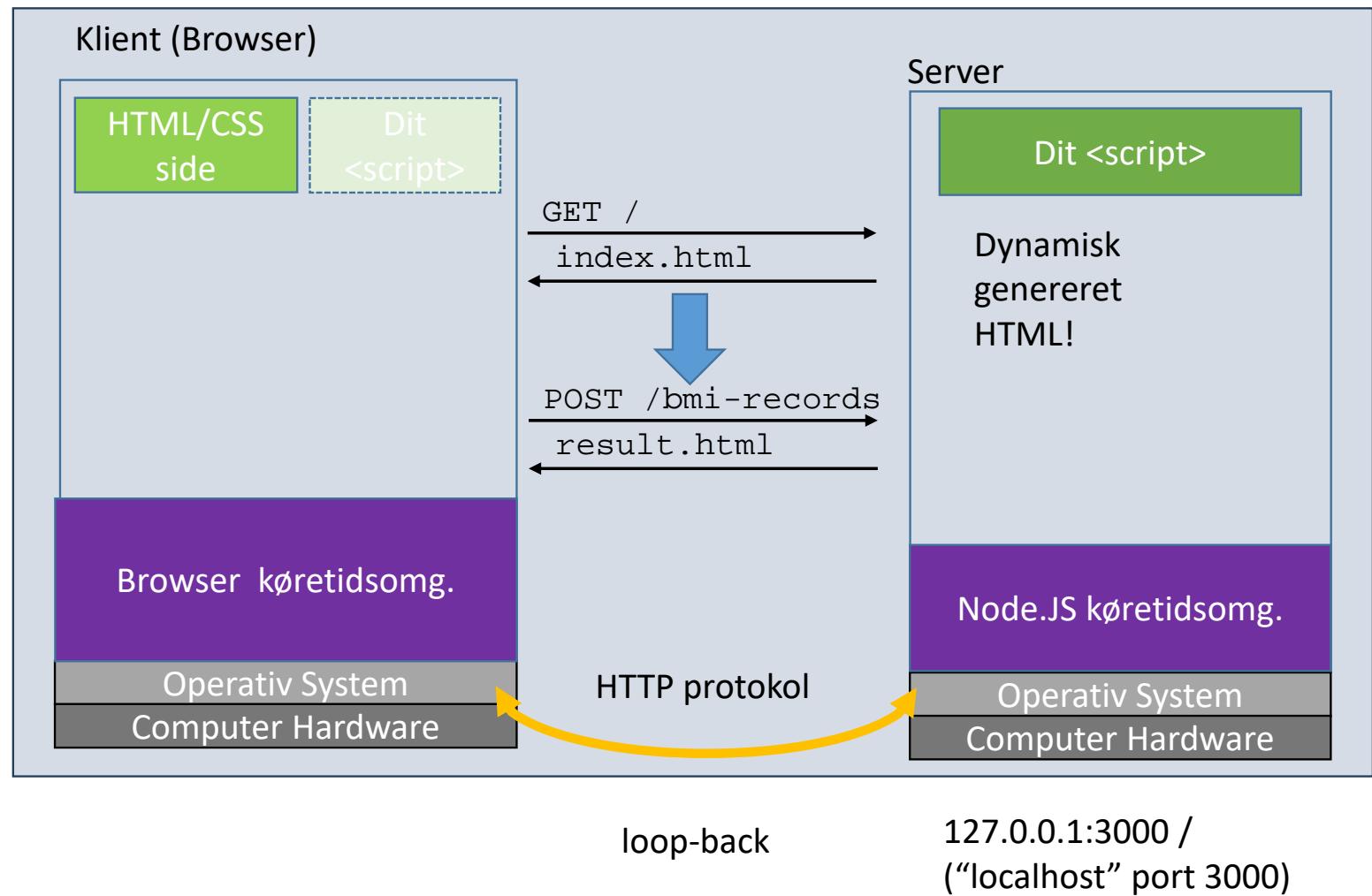
Server



I dag: Klassiske Web-side applikationer

Idag vil klient og server køre på same maskine!

- Bekvemt udviklingsmiljø.
- “loopback” forbindelse: virtuel netværksenhed, som maskinen bruger til at kommunikere med sig selv.
- Fungerer præcist som en (meget hurtig) netværksforbindelse
- Server identificeres med IP:Port
 - 127.0.0.1:3000
 - localhost:3000



Klient



Struktur af BMI-SITE applikation

GET /

Front page html

POST /bmi-records

Html page

Server/http-modul
(server.js)

Routing
(router.js)

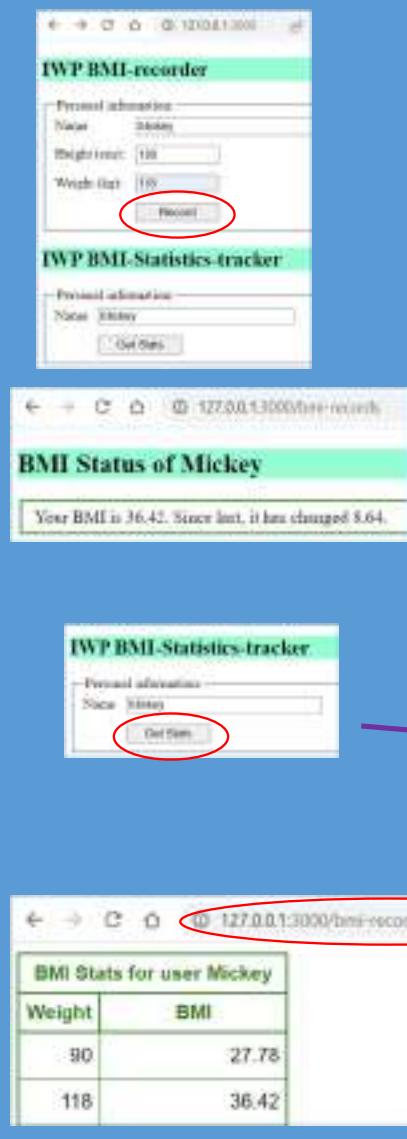
Applikations
funktionalitet
(app.js)

Node.js

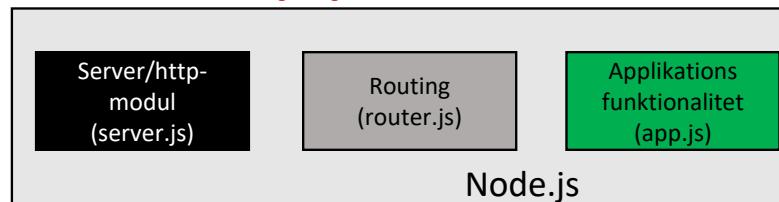
Overordnet logik:

- Browser udpeger applikation med URL: `127.0.0.1:3000 /`
- Server leverer forside (`bmi.html`)
- Bruger angiver navn, højde, og vægt
- Klient (Browser)
 - klient side HTML validering af formular
 - formular data indsendes i POST (data i http body)
- Server
 - validerer formular input,
 - gemmer data i "DB" (array)
 - Renderer side med resultat, og sender html tilbage i HTTP respons
- I statistik formular indsendes data via GET, `userName` i query string

Klient



Adfærd af BMI-SITE applikation



POST /bmi-records

html page

GET /bmi-records?userName=Mickey

html page

```
validatedData=validateBMIRecordForm  
          (formData)  
status=recordBMI(validatedData)  
html=renderHTMLBМИUpdatePage(status)
```

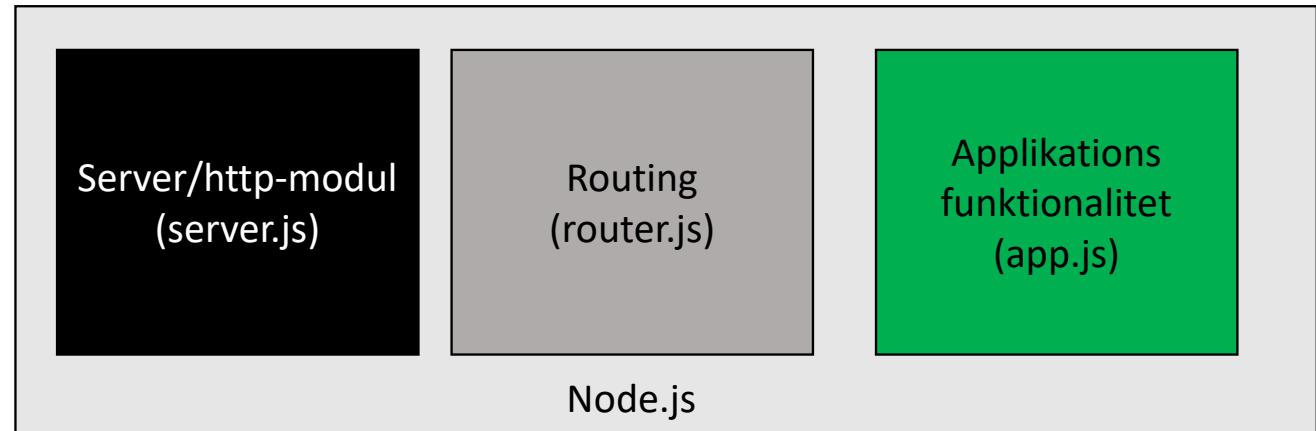
```
validatedFormData=validateBMIStatForm  
           (formData)  
Html=renderHTMLBМИStatPage(  
    validatedFormData)
```

server.js og router.js er BLACK BOXE ... indtil videre ☺

Fil-Struktur af applikationen

Fil-struktur

```
node/  
  server.js  
  app.js  
  router.js  
PublicResources/  
  css/  
    simple.css  
  js/  
  html/  
    help.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)

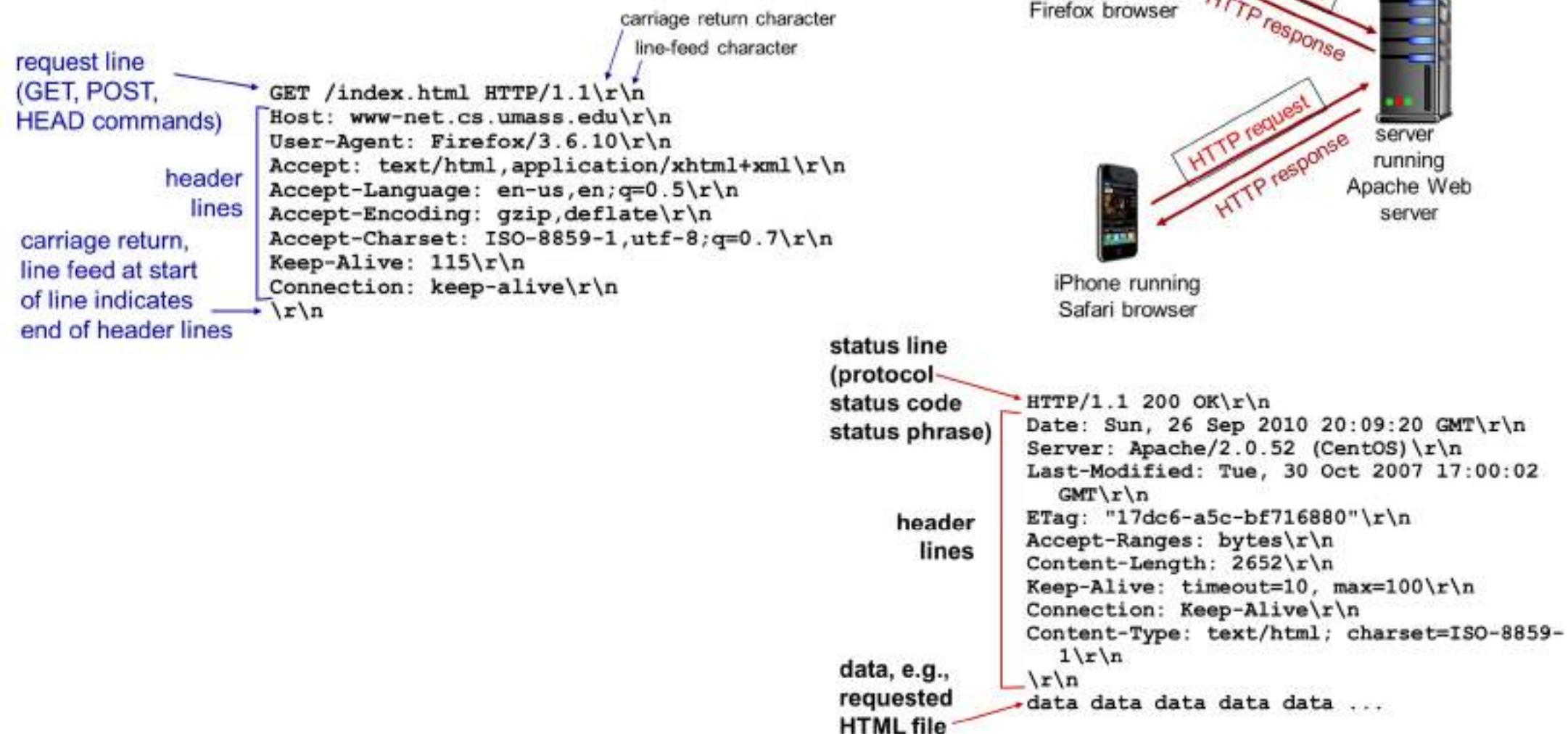
Dynamisk serverside generering af HTML

- HTML sider ligger ikke nødvendigvis (sjældent) som statiske filer på server
 - Dog sommetider som "skabeloner/templates"
- Script på serversiden genererer en streng som indeholder html-dokumentet; strengen sendes som svar i stedet for filen
 - Evt baseret på data udlæst fra en database.

Se eksempelkode

```
function renderHTMLBMIUpdatePage(bmiStatus){  
  let page=renderHTMLHdr("BMI Status",["/css/simple.css"]);  
  page+=`<body><section>  
    <h1> BMI Status of ${bmiStatus.userName} </h1>  
    <output>  
      Your BMI is ${bmiStatus.bmi}. Since last, it has changed ${bmiStatus.delta}.  
    </output>  
  
  </section></body>`;  
  return page;  
}
```

HTTP Beskeder



HTTP/1.1 Metoder (a.k.a "verbs")

- **GET:**

- Anmoder om overførsel af en repræsentation af den ønskede ressource
- "Læsning"

- **POST:**

- Udfør en resource-specifik behandling på den ønskede ressource
- "Ændring"
 - Fx, tilføje data til ressourcen (fx indtastet i en "HTML form")

- **PUT:**

- Oprette (eller erstatte) tilstanden på den ønskede ressource) i sit hele så den svarer til den medsendte repræsentation

- **DELETE**

- Sletter ressourcen (eller fjerner forbindelsen imellem URL navn og ressourcen)
- HEAD, PATCH, CONNECT, OPTIONS, TRACE

Når serveren modtager et request med en given metode kalder den en **funktion** som "du" eller (web-server programmøren) har lavet!

"Spilleregler" for web-arkitekturen forudsætter at **funktionen** (som udføres på server siden) skal respektere hensigten bag HTTP metoderne

- Fx må GET ikke ændre ressourcen.
- Caching, forudindlæsning ("pre-fetching")

Sikre metoder: udførslen af **funktionen** må ikke "skade" ressourcen, eller give unormal stor belastning på server

- Klient kan gentage dem
- GET, HEAD, OPTIONS, TRACE skal programmeres så de er sikre
- Et PUT umiddelbart efterfulgt af et GET skal give den værdi der netop er oprettet
- Begrebet "**Idempotente**" operationer introduceres senere

DIN KODE SKAL RESPEKTERE DISSE

URL og URLSearchParams Objekterne [DF11.9]



```
function SearchParamsDemo(){
  let url = new URL("http://example.com");
  url.pathname="bmi-records";
  console.log(url.toString());          //=>http://example.com/bmi-records
  let params = new URLSearchParams();
  params.append("userName", "Mickey");
  params.append("weight", "100");
  console.log(params.toString());        // =>userName=Mickey&weight=100
  url.search = params;
  console.log(url.href);                // =>http://example.com/bmi-records?userName=Mickey&weight=100

  let url2= new URL("http://example.com/bmi-records?userName=Mickey&weight=100");
  let sp=new URLSearchParams(url2.search); //or simply sp=url2.searchParams;

  console.log ( sp.has("weight") );//=> true
  console.log ( sp.has("sex") );   //=> false
  console.log ( sp.get("weight") );//=> 100
}
```

De næste gange: Moderne Web-applikationer

IWP BMI-tracker

Personal information:

Name: Mickey

Height (cm):

Weight (kg):

Submit



IWP BMI-tracker

Personal information:

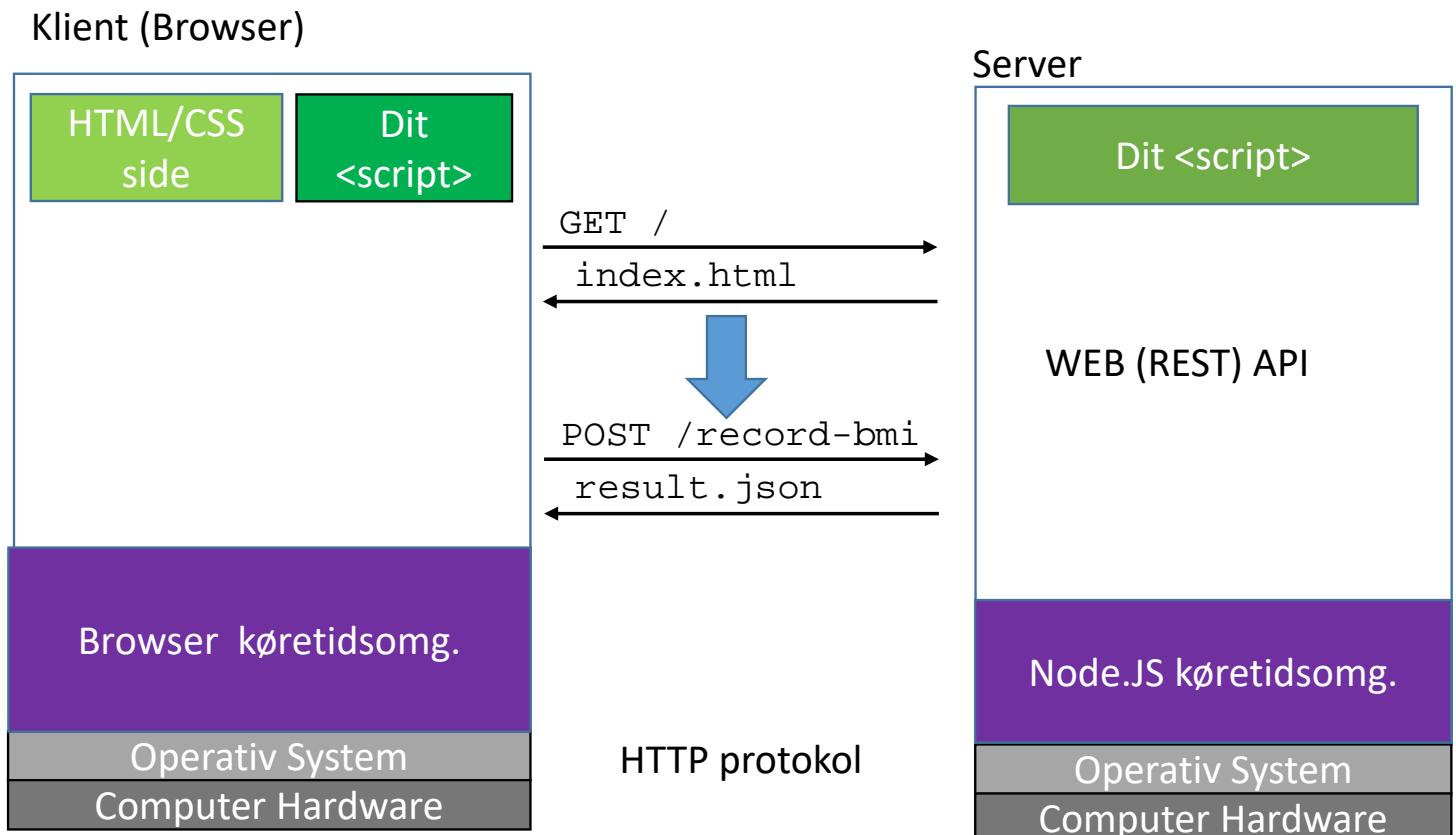
Name: Mickey

Height (cm): 180

Weight (kg): 100

Submit

Hi Mickey! Your BMI is 30.86. Since last it has changed -0.14!



Samme html dokument opdateres vha DOM og JS: Output synliggøres.

Internetværk og Web-programmering

Klient side scripting:

Browser som kørtidsomgivelse

Forelæsning 5
Brian Nielsen

Distributed, Embedded, Intelligent Systems



Agenda

- Single-page web-apps (BMI-demo)
- Browser APIs
 - DOM (Document Object Model)
 - Events and Callbacks (PauseTæller)
- Indlæsning af JS
 - Kode-injektionsangreb
 - CORS fejl
- Asynkron programmer
 - Fetch-API
 - Promises
 - BMI-app indlæsning af JSON

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!

← → C ⌂ ⌂ 127.0.0.1:3000

IWP Health App

Record BMI View BMI Stats Help

IWP BMI-recorder

Personal information:
Name Mickey
Height (cm):
Weight (kg):
Record

IWP BMI-Statistics-tracker

Personal information:
Name Mickey
Get Stats

IWP BMI-Statistics-tracker

Personal information:
Name Mickey
Get Stats

BMIStats for user Mickey	
Weight	Bmi
90	27.78
188	58.02

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

```
<html> -- 30
  <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.

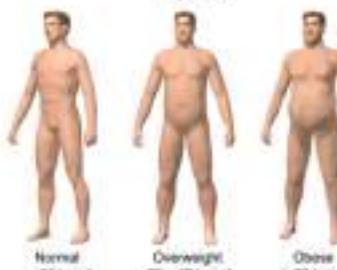
$$\text{BMI} = \frac{\text{weight (kg)}}{\text{height}^2(\text{m})}$$


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

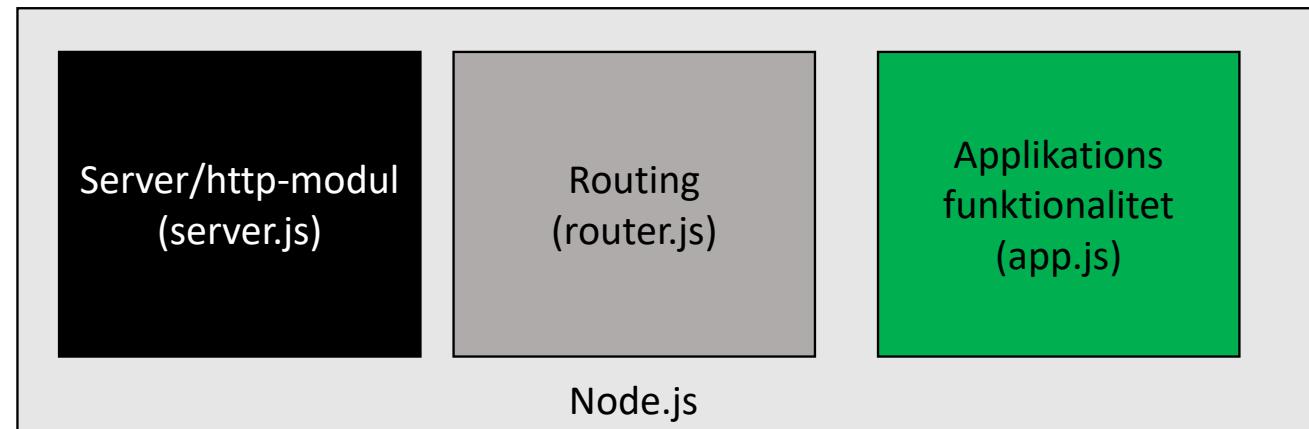
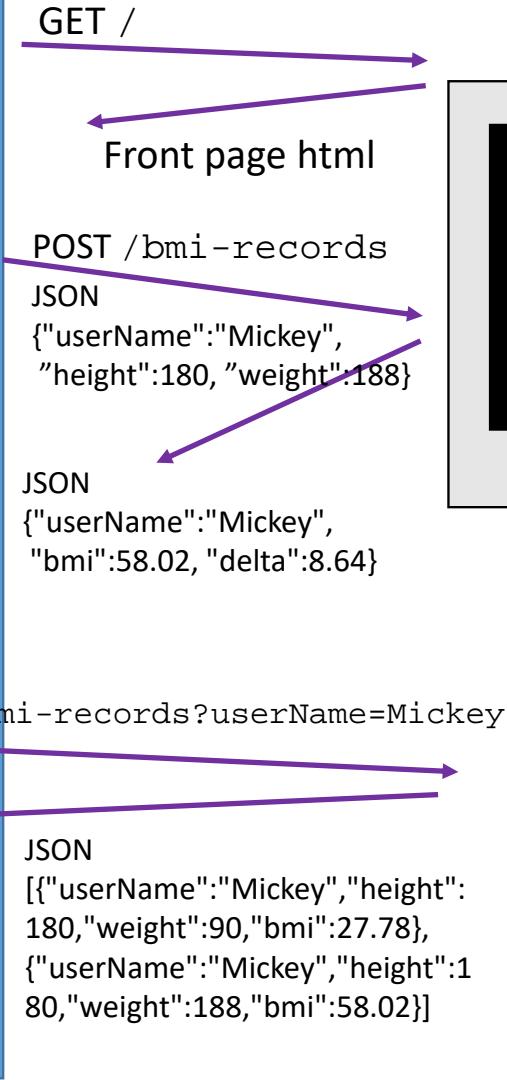
Background

Klient

The screenshot shows the IWP Health App interface with three main sections:

- IWP Health App:** Shows a "Record BMI" button, "View BMI Stats" button, and a "Help" link.
- IWP BMI-recorder:** A form for recording personal information (Name: Mickey, Height: 180, Weight: 188) and a "Record" button. Below the form, a message says "Hi Mickey! Your BMI is 58.02. Since last it has changed 8.64".
- IWP BMI-Statistics-tracker:** A form for entering personal information (Name: Mickey) and a "Get Stats" button. Below the form, a table titled "BMIStrats for user Mickey" shows two rows of data: [Weight: 90, Bmi: 27.78] and [Weight: 188, Bmi: 58.02].

Struktur af BMI-APP applikation



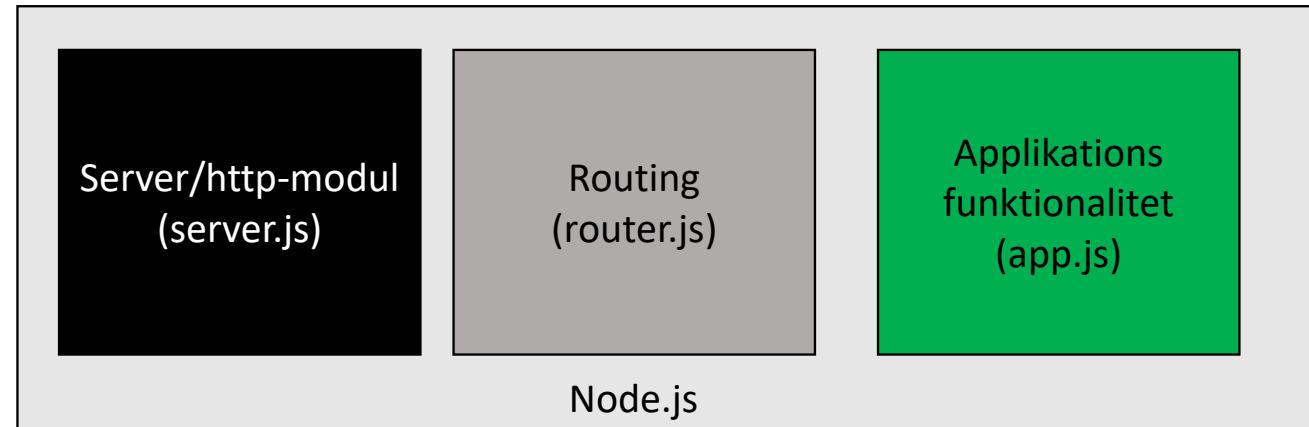
Overordnet logik:

- 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.

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:

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		
...				

JAVASCRIPT bmi-client.js

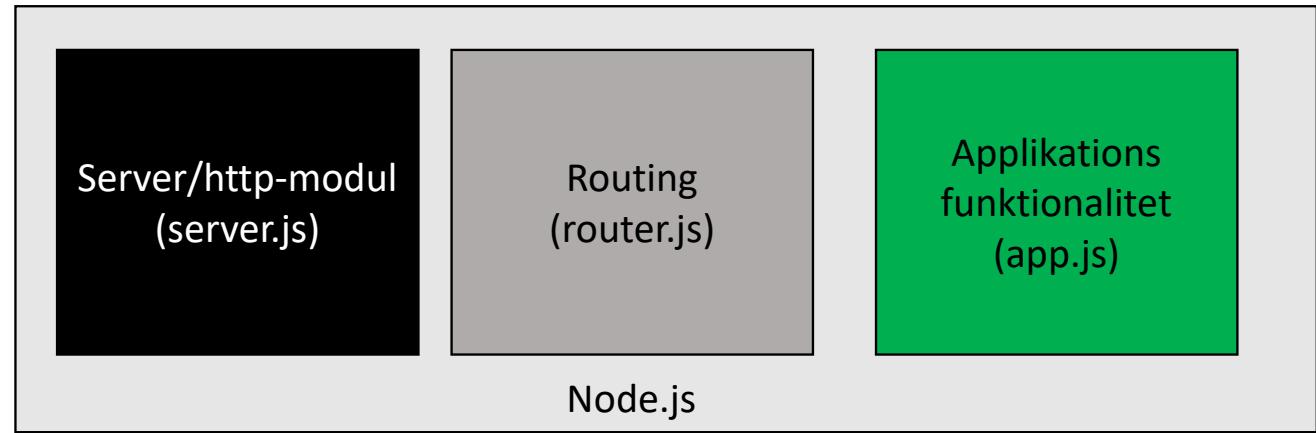
[username : Mickey, height : 180, weight : 188, bmi : 58.02]

- 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

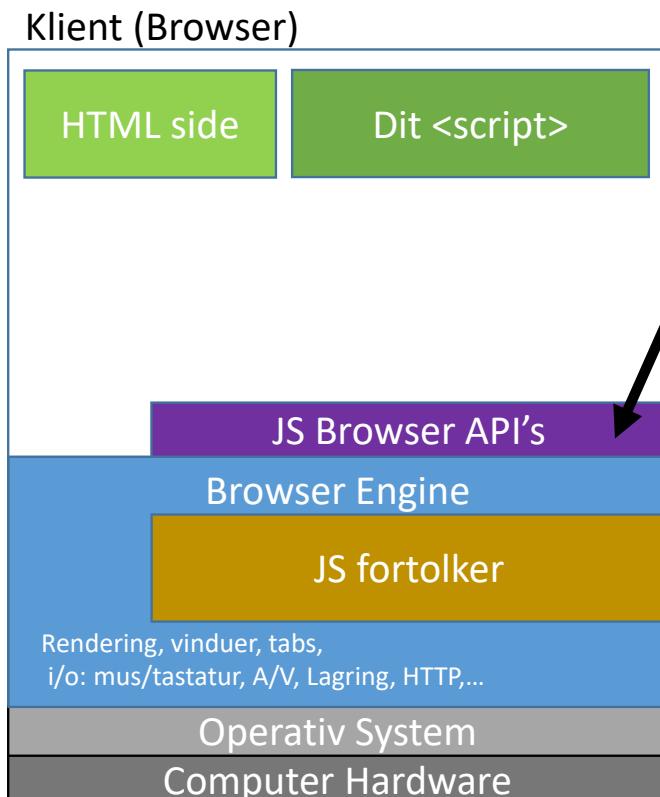
REMINDER!

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 "bekvemt" kan foretage lokalt
 - Funktioner den skal vare tage, hvis server er "nede"

Overordnet arkitektur: Indbyggede biblioteker

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



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 API'er ("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

Gamedep 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

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 **organizeret 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?

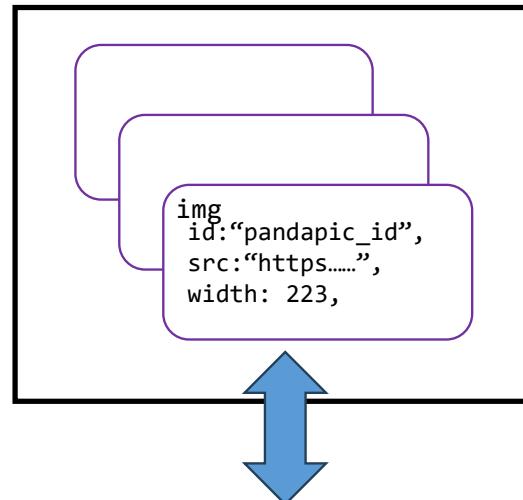
Document Object Model

dom-demo.html

Browser præsentation



DOM-repræsentation



Samling af javascript objekter som afspejler visning og HTML

Vist HTML repræsentation

```
<!DOCTYPE html>
<html lang="de">
  <head> ... </head>
  <body> ... $0
    
    <script> ... </script>
  </body>
</html>
```

```
let pandaElement=document.querySelector("#pandapic_id");
pandaElement.src="https://upload.wikimedia.org/wikipedia/commons/2/26/Bison_bonassus_%28Linnaeus_1758%29.jpg"

pandaElement.width=500
let newH1=document.createElement("h1");
newH1.append("Hej IWP");
pandaElement.before(newH1);
```

Hej IWP



```
<!DOCTYPE html>
<html lang="de">
  <head> ... </head>
  <body> ... $0
    <h1>Hej IWP</h1>
    
    <script> ... </script>
  </body>
</html>
```

Træ-struktur for HTML

Relationer som alm. familie træ:

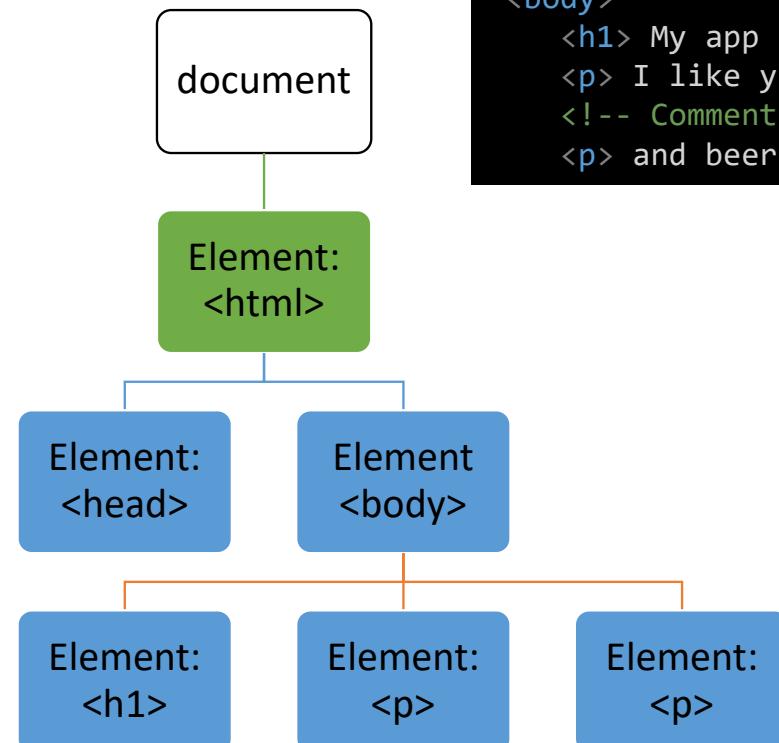
- Elementer på samme niveau: **Søskende**
- Elementer på underliggende niveauer: **efterfølgere**
- Elementer på overliggende niveauer: **forfædre**
- Elementer på umiddelbart overeau: **forældre**
- Elementer på umiddelbart underliggende niveau: **børn**

document

Repræsenterer en indlæst web-side i browser

Indbygget global variabel "**document**"

Udgør roden i DOM træet

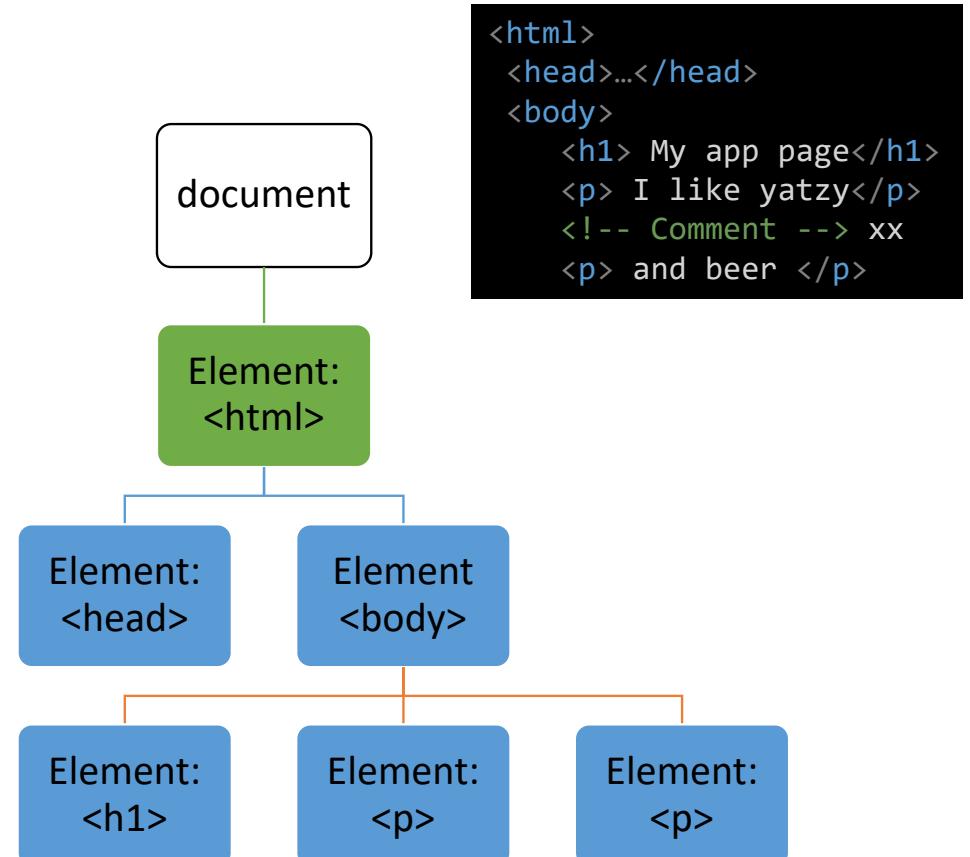


```
<html>
  <head>...</head>
  <body>
    <h1> My app page</h1>
    <p> I like yatzy</p>
    <!-- Comment --> xx
    <p> and beer </p>
```

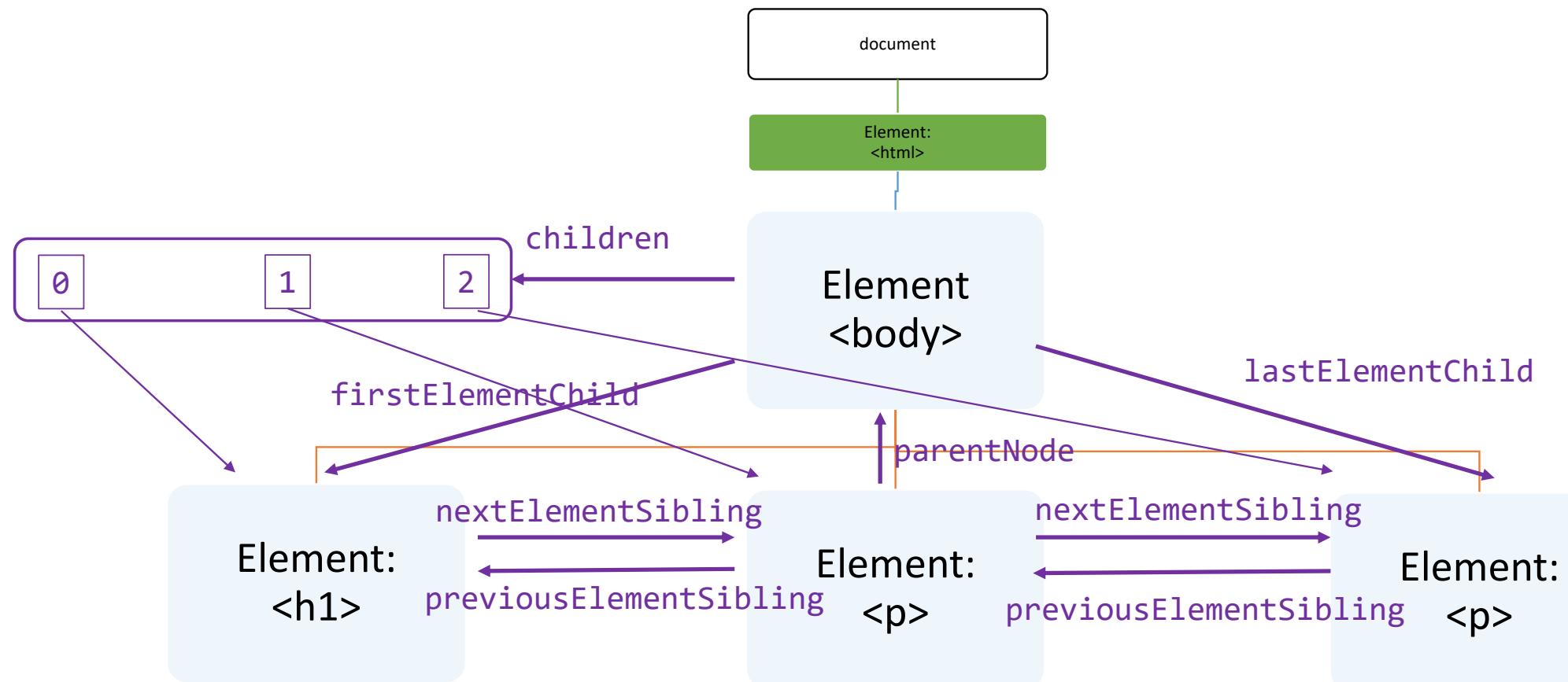
Træ-struktur for HTML: "HTML Element-træet"

Hvert HTML element har flg. egenskaber

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



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

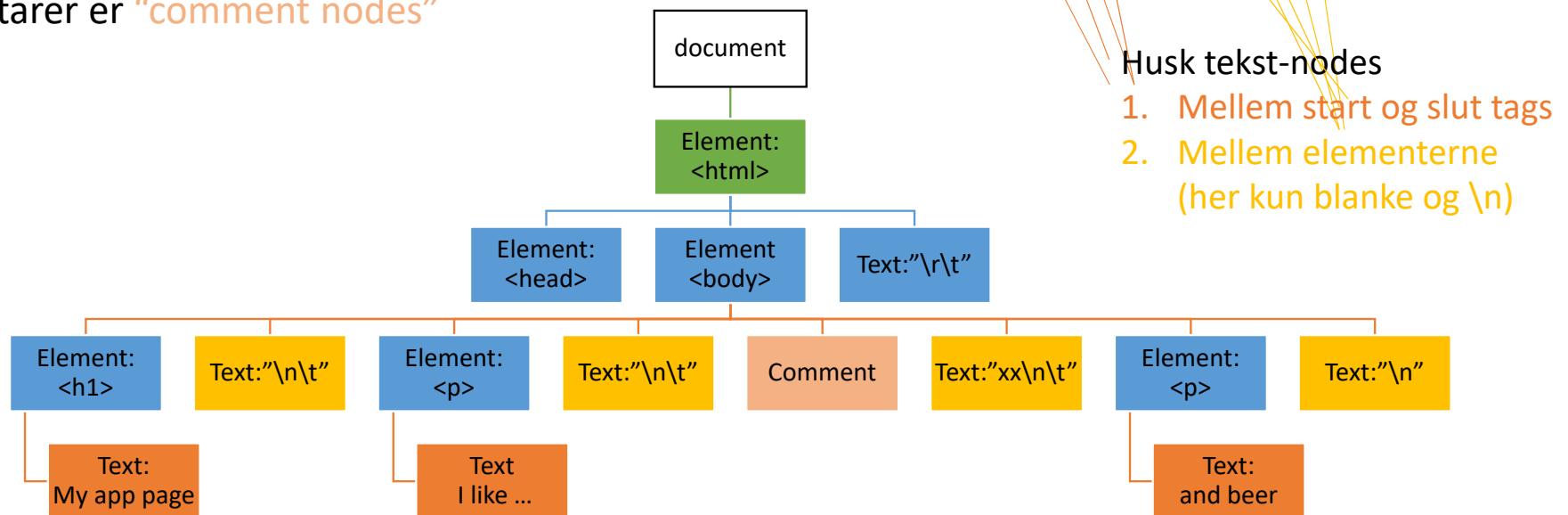


```
document.body.children[0].textContent;  
// "My app page"
```

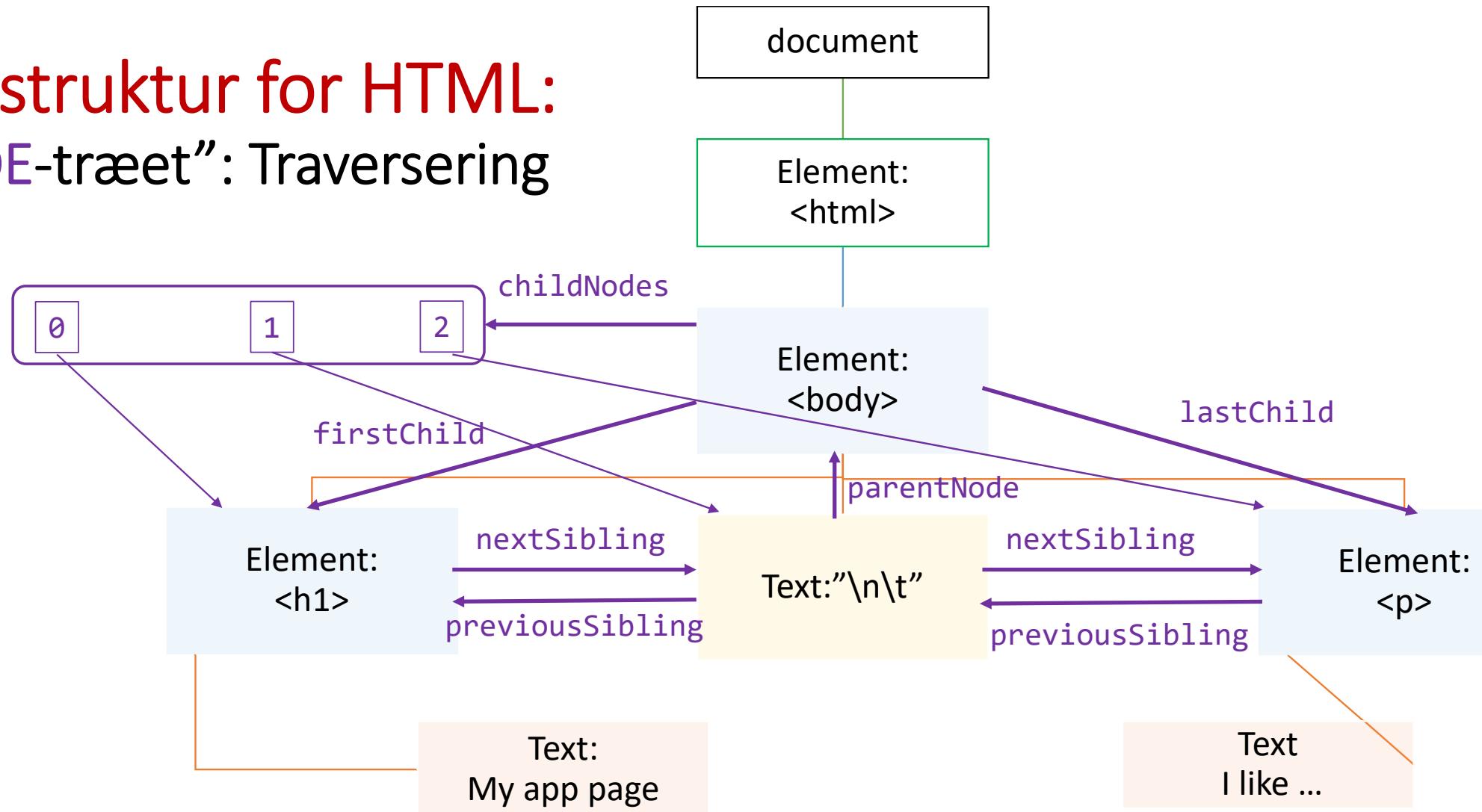
Træ-struktur for HTML: "NODE-træet"

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"



Træ-struktur for HTML: "NODE-træet": Traversering



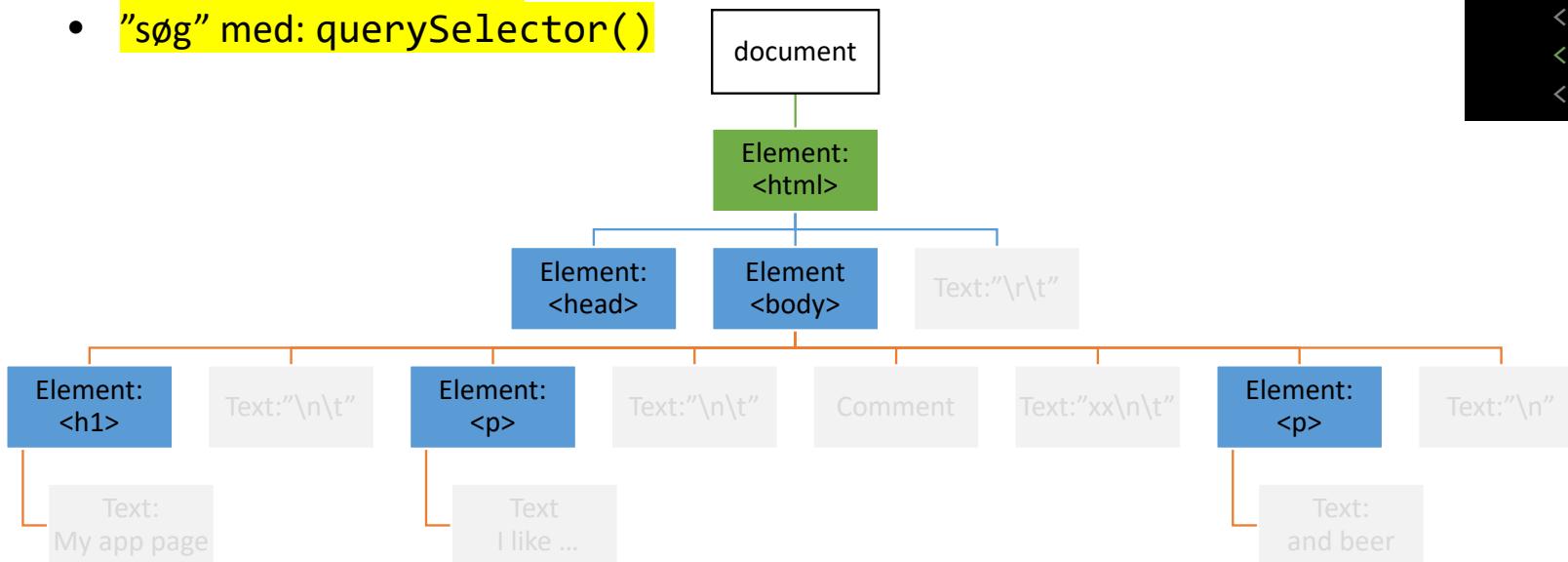
```
document.body.childNodes[0].firstChild.nodeValue;  
// "My app page"
```

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()`



HTML Element egenskaber

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

```
"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>
```

```
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!!!

- 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 søgestrenget
- Returnerer array lignende objekt (HTMLCollection) med alle matchende elementer

document.querySelector

Samme selector syntax som i CSS

querySelectorAll("nav ul a:link")

```
<body>
  <nav>
    <ul>
      <li><a href="#">Canada</a></li>
      <li><a href="#">Germany</a></li>
      <li><a href="#">United States</a></li>
    </ul>
  </nav>
```

querySelector("#main>time")

```
Comments as of
<time>November 15, 2012</time>
<div>
  <p>By Ricardo on <time>September 15, 2012</time></p>
  <p>Easy on the HDR buddy.</p>
</div>
```

```
<div>
  <p>By Susan on <time>October 1, 2012</time></p>
  <p>I love Central Park.</p>
</div>
```

querySelector("footer")

```
<footer>
  <ul>
    <li><a href="#">Home</a> | </li>
    <li><a href="#">Browse</a> | </li>
  </ul>
</footer>
</body>
```

HTML Elementers attributter i DOM

- HTML elementers attributter ”spejles” som egenskaber i DOM

```
<body>
  

  <label for="pauseInput">Indtast pauselænge 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);      //"billedet af et ..."
console.log(panda_pic.src);     //"https://media...."
console.log(panda_pic.alt);     //"panda_pic_id"
console.log(panda_pic.width);   //223

let pauseInputElement=document.getElementById("pauseInput");
let varighed=Number(pauseInputElement.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 (streg) for attributten med navnet name (streg)
- const attrValue = element.setAttribute(name, value);
 //Sætter/ændrer attributten med navnet name (streg) til value (streg)

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>
```

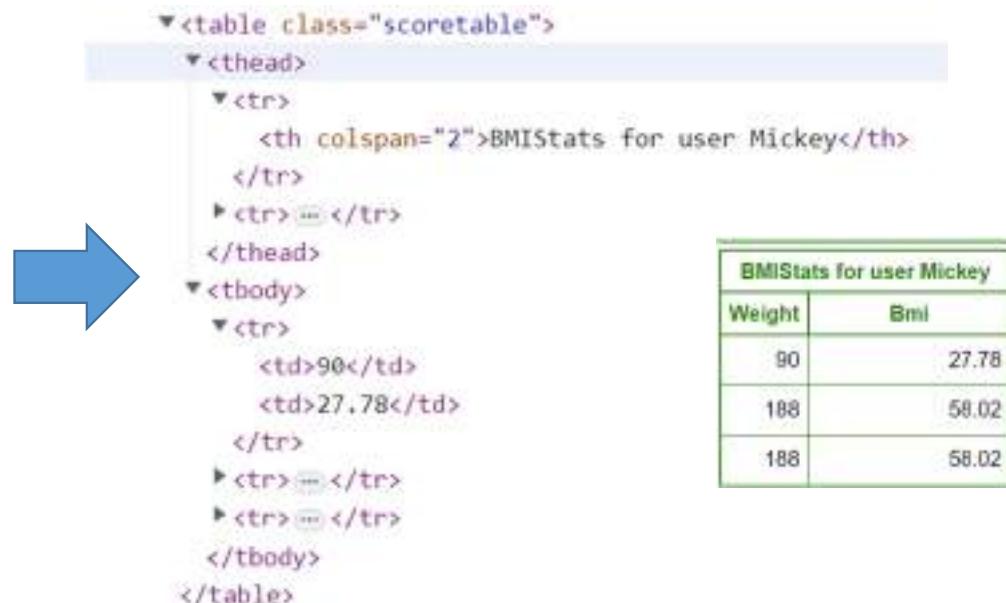
My app page
I like yatzy
xx
and Cola



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("BMISstats 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;  
}
```



Hændelser

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 utsat 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`);
});
```



```
<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

Pause Tælleren

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

```
<label for="pauseInput">Indtast pauselænge 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 pauseInputElem=document.getElementById("pauseInput");
let startElem=document.getElementById("start");

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

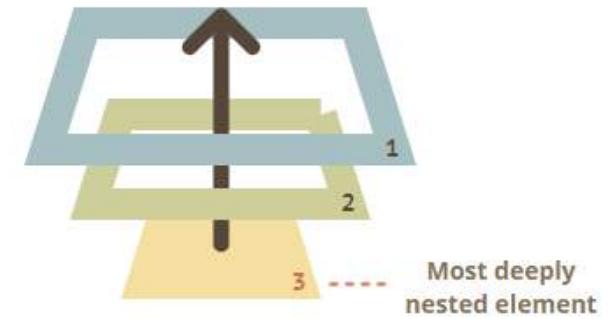
//eventhåndtering for tryk på start-knappen:
function start(event){
    //da vi ikke bruger forms, beder vi HTML om at validere
    //hvis invalid, fyrer checkValidity også en "invalid" Event
    if(pauseInputElem.checkValidity())
        resetTaeller(event);
}
function resetTaeller(event){
    let varighed=Number(pauseInputElem.value);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

IWP Pausetæller



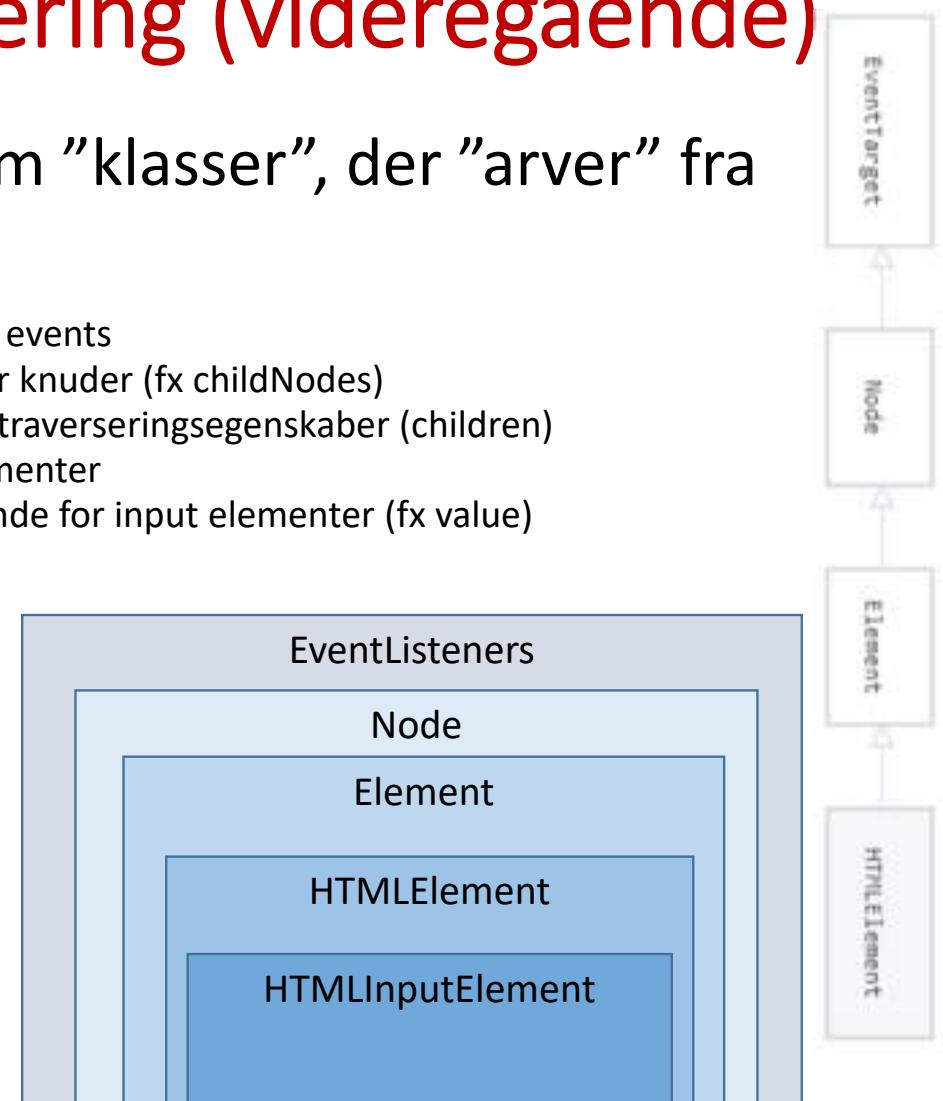
DOM API: EVENT Bubbling and capturing

- 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
 - ”**bobler**” 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](#)



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>

Indlæsning af Javascript

Indlæsning af JavaScript i klienten

- **Indlejret JavaScript metoden:**

- koden placeres indenfor et HTML <script> element.
- Til simplere 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 externe resourcer (CSS/Scripts/imgs) påbegyndes (separate HTTP kald)
2. Når browser mødet 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 kode sendes som kildetekst!

The screenshot shows a browser window with developer tools open. The address bar indicates the page is not secure (`Not secure | http://people.cs.aau.dk/~bnielsen/IWP/PauseTeller/pause.html`). The developer tools interface has tabs for Elements, Console, Sources, Network, Performance, Memory, Application, and Security. The Sources tab is active, showing the file structure and the content of the `pausetaeller.js` file.

File Structure:

- top
- people.cs.aau.dk
 - ~bnielsen/IWP/PauseTeller
 - js
 - pausetaeller.js
 - styles
 - pause.html

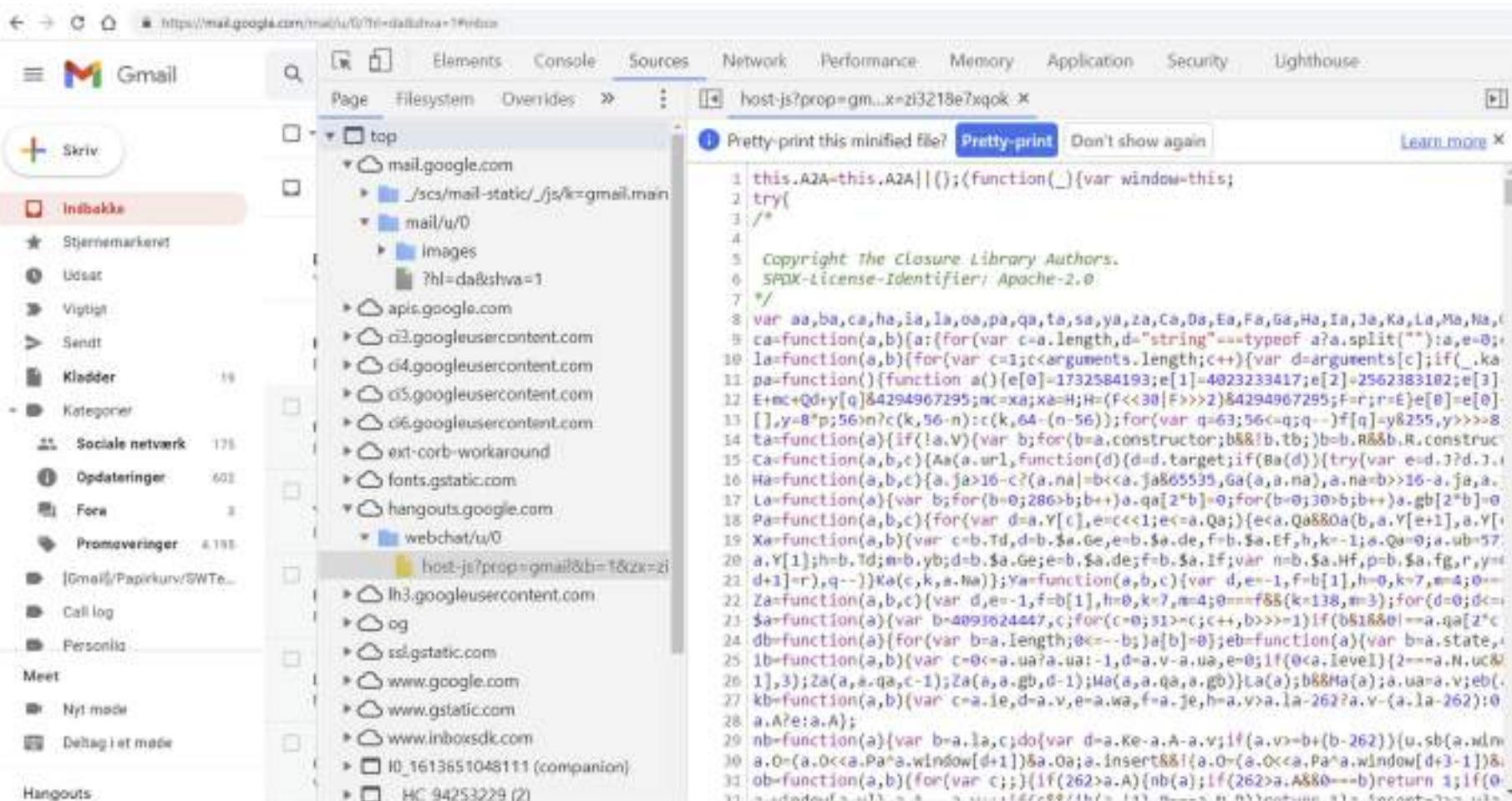
Code Content:

```
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 ticks 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 methode, 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;
29     }
30 }
```

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 "***obfusceres***", 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?



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 stripes)
 - 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 angrebs-strategi

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

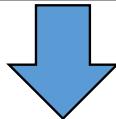
<script>
function processInput(text){
  console.log("Hej "+text);
  document.write("Hej "+text);
  // a bit more sophisticated would use:
  // document.write(document.head.innerHTML+document.body.innerHTML+"Hej "+text);
}
</script>
</body>
</html>
```

Kode injektionsangreb 1: via document.write

Normalt Brug

IWP Demo

Indtast dit navn:



Hej Brian

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

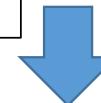
Ondsindet brug

Indtast flg. som navn

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

IWP Demo

Indtast dit navn:



This page says

You have been HACKED!!

OK

```
<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>
```

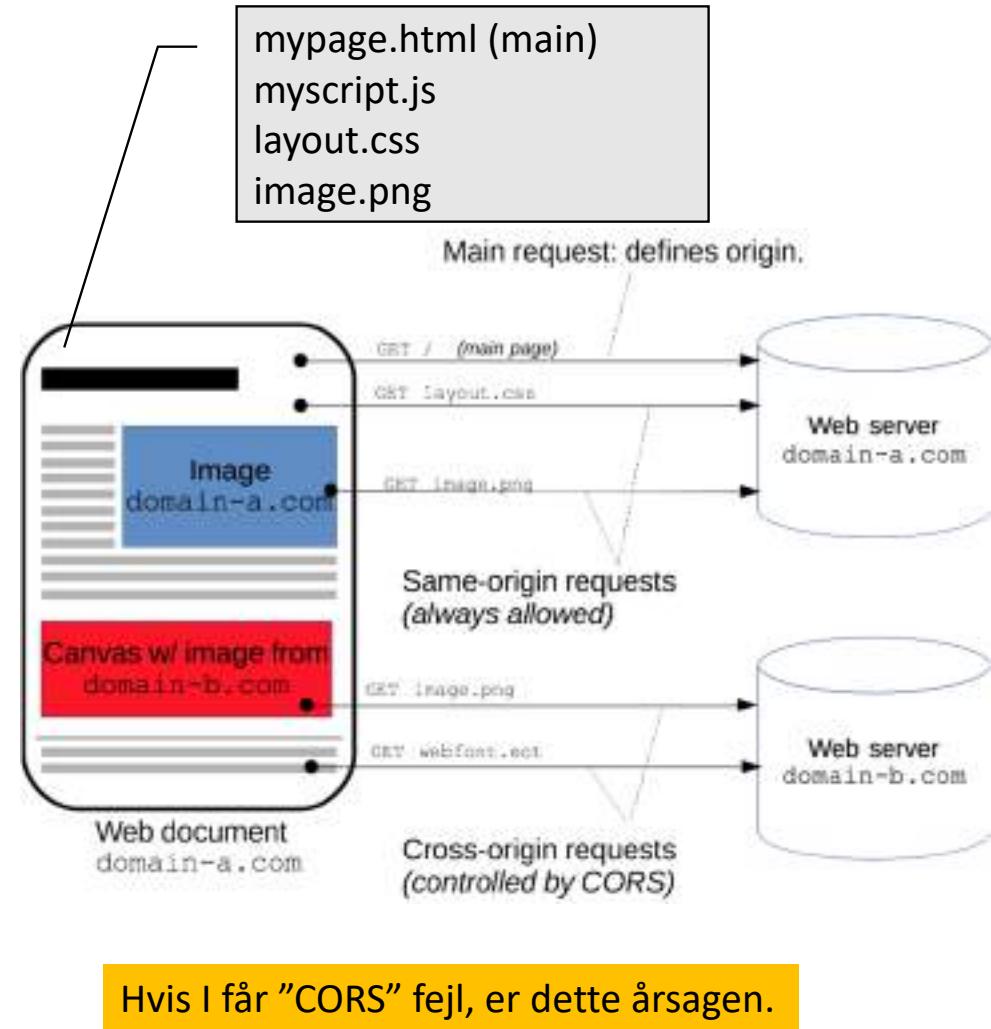


Kode injektionsangreb

- 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 tilgåes fra domain-a.com (**whitelisting**)



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

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

Bemærk:

- HTML sider, som indlæses fra filesystem 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 filesystem.

```
<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 bibliotekter

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

The screenshot shows a web browser window with the URL <https://www.cs.aau.dk>. The left side displays the university's logo and the text "AALBORG UNIVERSITY" and "DEPARTMENT OF COMPUTER SCIENCE". Below this are two images: one of students working on laptops and another of a research lab.

The right side of the screen shows the browser's developer tools with the "Sources" tab selected. The left pane lists the file structure of the page, including "digitalAssets" (containing files 917, 939, 947, 957, 979, and index), "fast.fonts.com", "fast.fonts.net", "platform.twitter.com" (with "widgets.js" highlighted), "www.design2013.aau.dk", "www.google-analytics.com" (with "analytics.js" highlighted), "plugins/ua" (with "linkid.js" and "analytics.js" sub-items), "www.googletagmanager.com", "www.resources.aau.dk", "aau-search-web-prod.azurewebsites.net/", and "rufous-sandbox (about:blank)".

The right pane shows the content of the "analytics.js" file. It includes a header for the Closure Library Authors and Apache-2.0 license, followed by a large block of minified JavaScript code. A tooltip at the top right of the pane asks "Pretty-print this minified file?" with options to "Pretty-print", "Don't show again", and "Learn more".

Asynkron program udførelse

Concurrency

Asynkrone Programmer

Pop quizzz

Hvad udskriver nedenstående program?

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



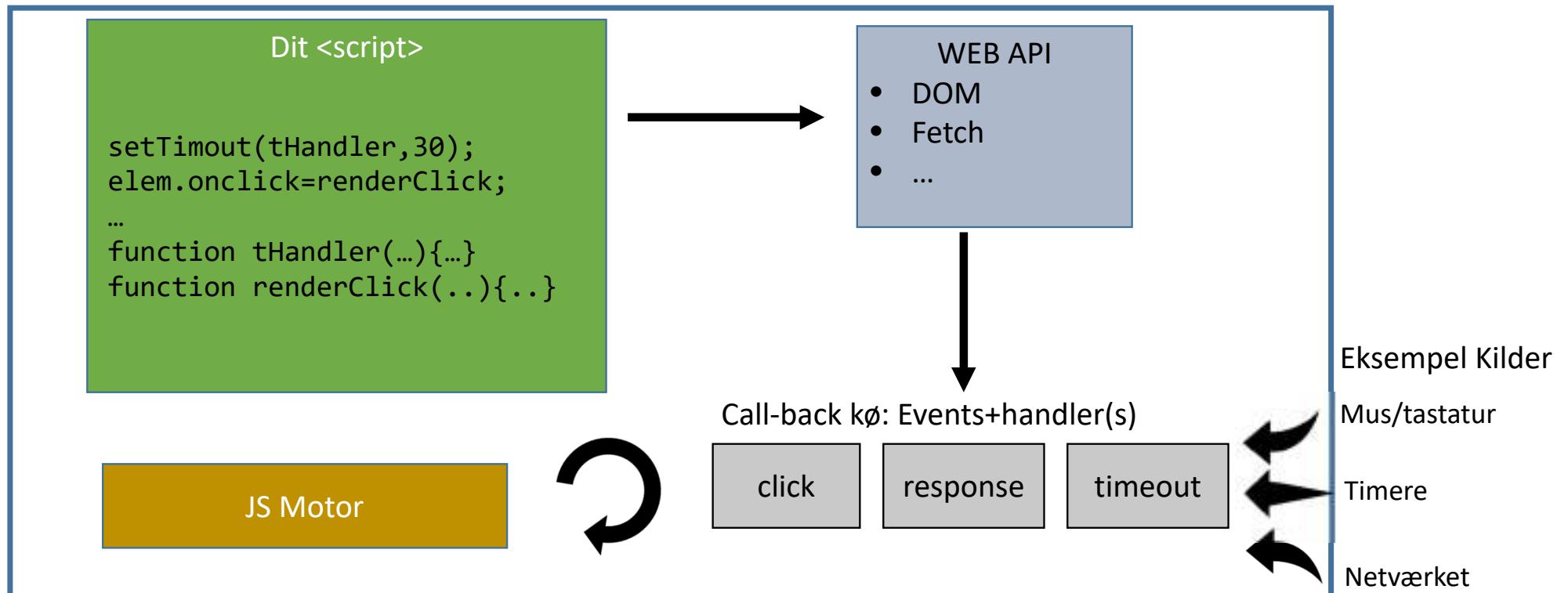
IWP:
med dig
Hej

- 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:
med dig
VM483:1
VM483:1
< undefined
Hej
VM483:1
```

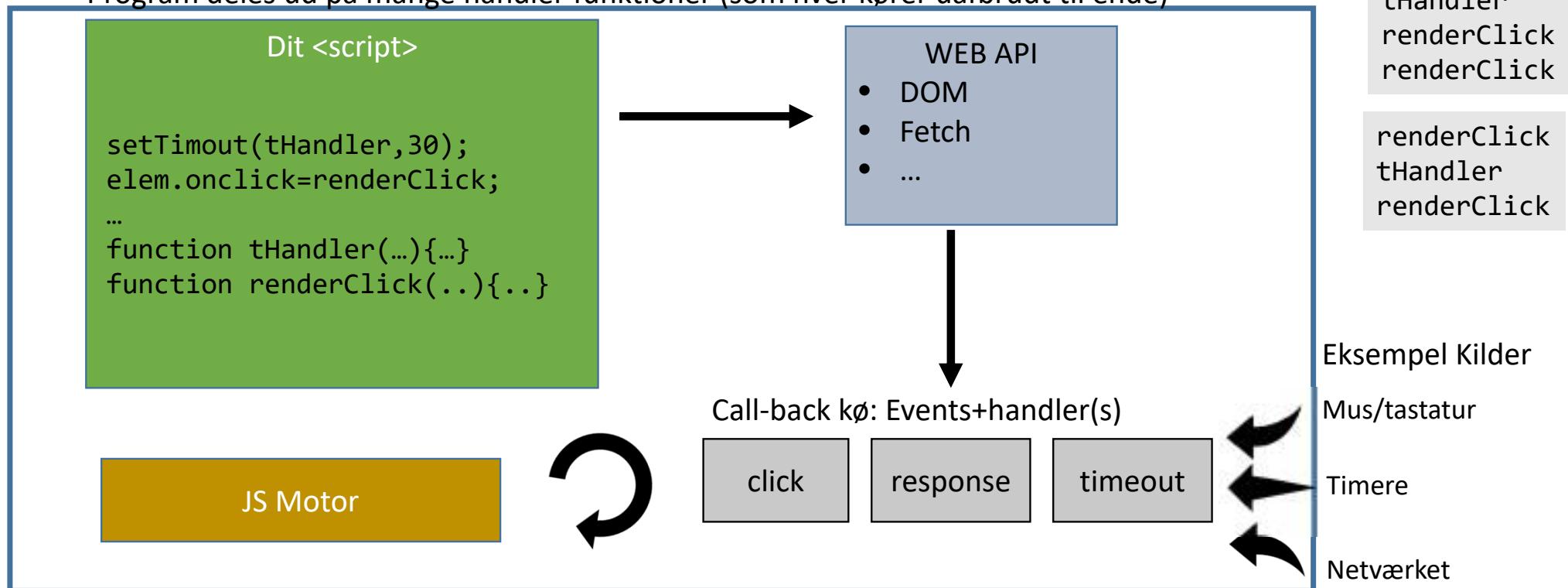
Event-loopet

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



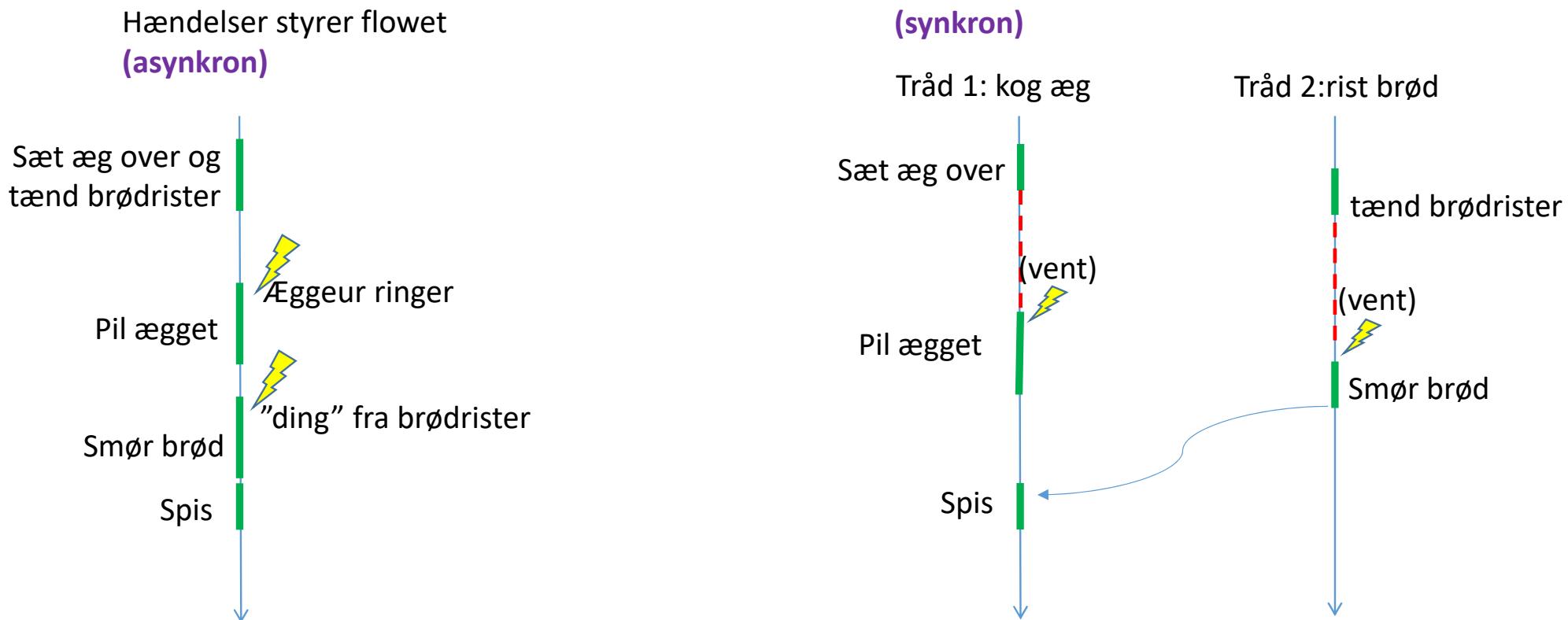
Event-loopet

- 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)



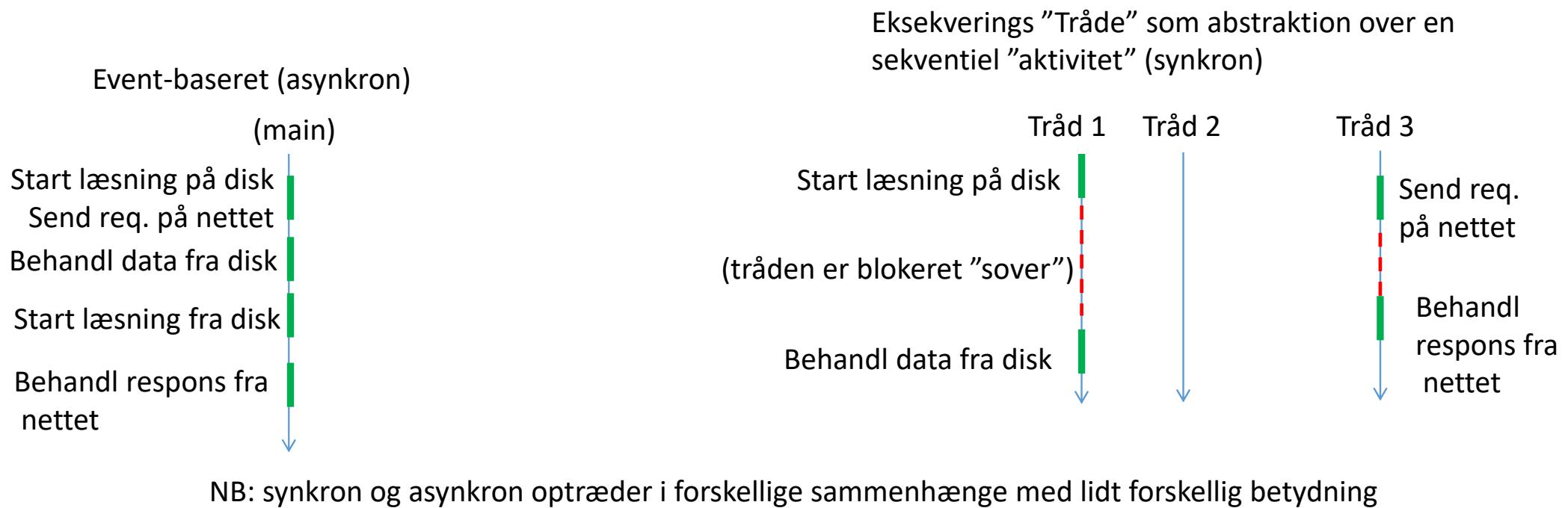
En analogi: Lav Morgenmad

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



Concurrency eller "Samtidighed"

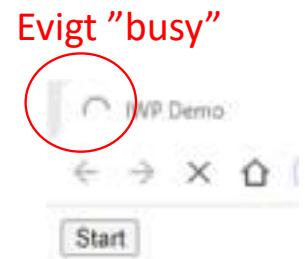
- En computer er i stand til at gøre mange ting samtidigt
 - Læse disk, holde øje tastatur, modtage fra netværket
 - Operativ system "multitasker" imellem flere programmer
 - Afvikle opgaver og beregninger ægte parallelt vha. flere processor kerner.



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); });
```



FETCH API

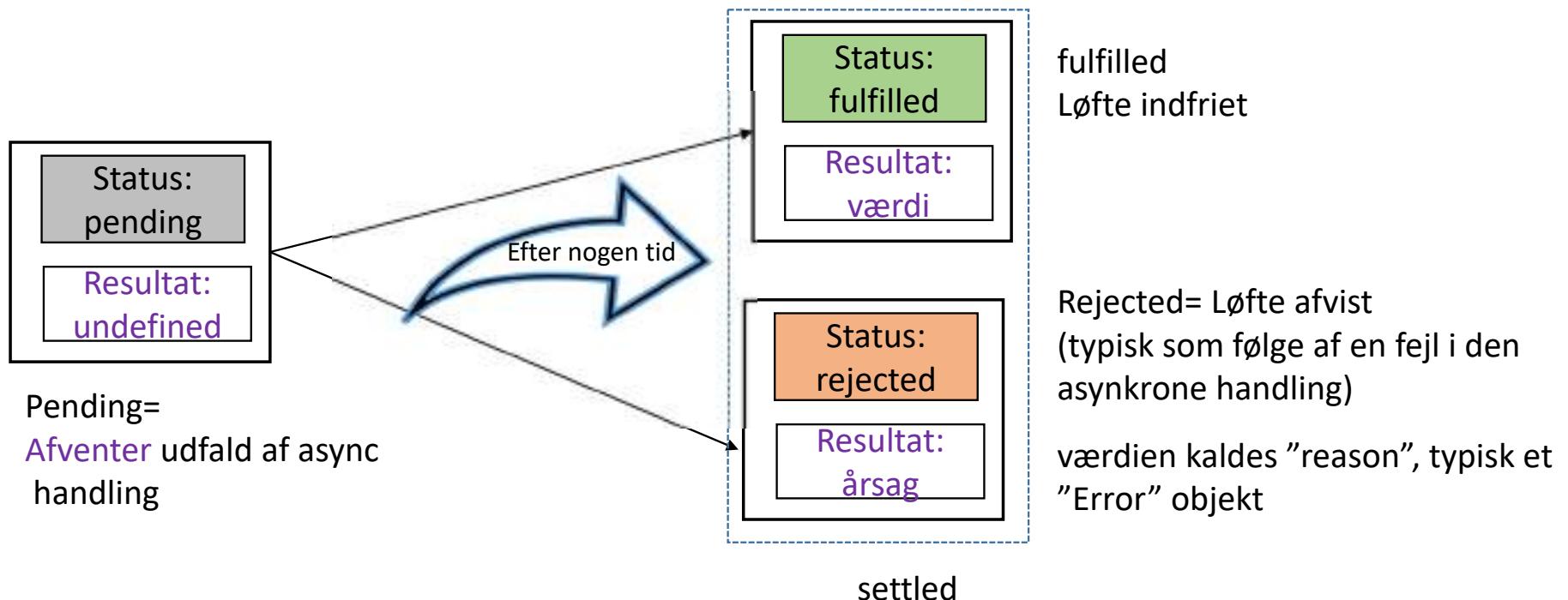
- Fetch er et promise-baseret API til hentning af web-ressourcer introduceret i ”moderne” JS
 - Ofte indlæser klienten javascript objekter fra server siden af applikationen
 - Serialiseret som en tekst streng i JSON format.
 - Alternativ i komplekse apps: ”Call-back hell”
 - Få asynkron kode til at ”ligne” 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

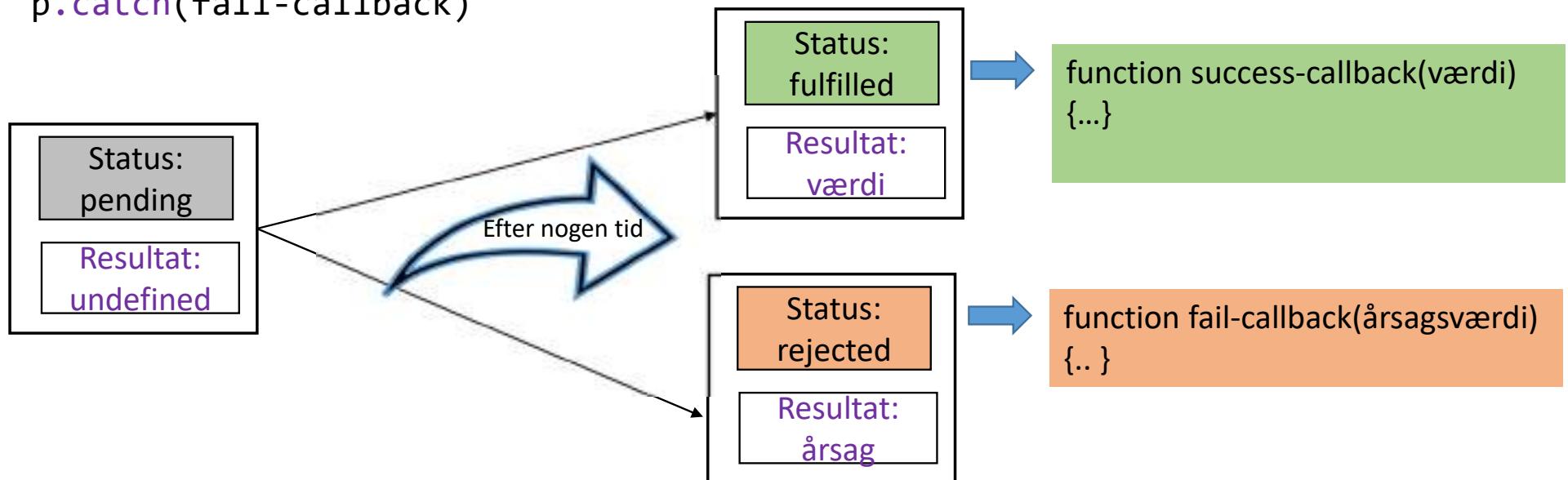
- Et **promise** er et objekt som er en placholder for en fremtidig værdi (resultat af en asynkron handling)
 - Fx afvent svaret på HTTP forespørgsler
- Mål: at forenkle struktur og overskuelighed af asynkrone programmer



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 returværdien 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)

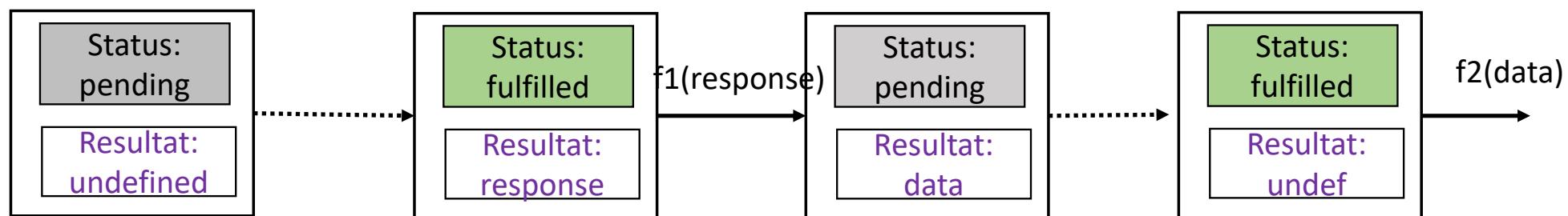


FETCH API - genbesøgt

- 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)
```

- `fetch` leverer et ”respons” objekt som argument til `f`.
- ”Respons” er et objekt giver adgang til HTTP responset.
- `respons.json()` returnerer et promise, som vil indeholde resultatet af at parse respons body som json.



*) bemerk: 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** (den indfrier sit promise, laver ikke reject) 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 http status kode.

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

Ex fra BMI -app

- Registrer eventhandler på submit (“record”)

IWP BMI-recorder

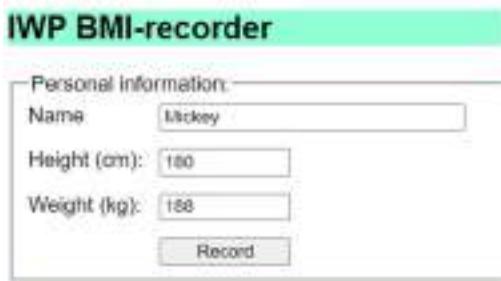
Personal information:

Name: Mickey

Height (cm): 180

Weight (kg): 168

Record



```
document.getElementById("bmiForm_id").addEventListener("submit", sendBMI);
```

Ex fra BMI-app

- `extractBMIData`: en hjælpefunktion, der udlæser indtastede værdier i formularen i et objekt
- `jsonPost`: 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 responsen kommer, parses respons body som json, som leverer `bmiStatus` objekt
- Grafikken opdateres, "submit" knappen tændes

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;
}
```

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);  
}
```

- fetch leverer et ”respons” objekt som argument til call-back jsonParse.
- ”Respons” er et objekt giver adgang til HTTP responset.
- respons.ok checker om HTTP respons=200?
- respons.json() returnerer et promise, som vil indeholde resultatet af at parse respons body som json.
- Derfor skal vi sikre at respons body har content-type json

Internetværk og Web-programmering

Asynkronitet, Promises, Fetch

Forelæsning 6
Michele Albano

Distributed, Embedded, Intelligent Systems



Agenda

- Exceptions
- Timers, callbacks and events
- Promises
- Async/await
- Fetch

Exceptions and Errors

- Synonyms in JavaScript
- Meaning: an exceptional condition or error has occurred

Terminology:

- An exception is *thrown* when the error or exceptional condition happens
- *Catching* an exception means handling the exception, to execute code e.g.: to recover from the exception.

The big picture

- The `try` statement lets you test a block of code for errors
- The `catch` statement lets you handle the error
- The `throw` statement lets you create custom errors
- The `finally` statement lets you execute code, after `try` and `catch` blocks, regardless of the result

Let us **throw** an exception

- The exception can be a JavaScript String, a Number, a Boolean or an Object

```
throw "Too big";      // throw a text  
throw 500;           // throw a number
```

- When the interpreter throws an error, it uses the `Error` class and its subclasses
- Properties: `name` (type of error) and `message` (holds the string passed to the constructor)
- JavaScript will actually create an `Error` object with two properties: `name` and `message`.

Example

```
function factorial(x) {  
    // If the input argument is invalid, throw an exception!  
    if (x < 0) throw new Error("x must not be negative");  
    // Otherwise, compute a value and return normally  
    let f;  
    for(f = 1; x > 1; f *= x, x--) /* empty */;  
    return f;  
}  
factorial(4)
```

Execution flow

- When an exception is thrown, JavaScript interpreter immediately stops normal program execution and jumps to the nearest exception handler
 - It checks against the current block of code
 - If no catch clause is found, next-highest enclosing block of code is considered
 - If no catch clause is found in the function, exception is propagated up to the code that invoked the function
 - If no catch clause is ever found, the exception is treated as an error and is reported to the user

How to catch an exception

- try/catch/finally statement:

```
try {  
    //Code where the exception could be thrown  
}  
  
catch(e) {  
    //Code to be executed if an exception is thrown. "e" is your exception  
}  
  
finally {  
    //Code executed at the end of the tryed code if no exception is thrown,  
    //and after the catched code if an exception is thrown  
}
```

- The finally code can throw an exception, which is propagated up; it can return to make the method return normally (no exception)

Error handling

- Two philosophies for error handling
 - One is the fail-silent approach where you ignore errors in the code
 - The other is the fail-fast and unwind approach where errors stop the world and rewind
- Better to stop code execution and let the user know, and possibly inform the developer

Input Validation

- Important use case for exceptions:
 - Examine the input. If it is wrong, an exception is thrown
 - The exception is caught by the catch statement and a custom error message is displayed

```
<!DOCTYPE html>
<html><body>

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="testIt()">Test It</button>
<p id="p01"></p>

<script>
function testIt() {
  var message, x;
  message = document.getElementById("p01");
  message.innerText = "";
  x = document.getElementById("demo").value;
  try {
    if(x == "") throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5) throw "too low";
    if(x > 10) throw "too high";
  }
  catch(err) {
    message.innerText = "Input is " + err;
  }
}
</script>

</body></html>
```

Agenda

- Exceptions
- Timers, callbacks and events
- Promises
- Async/await
- Fetch

Asynchronous execution

- A program is doing something
- The current action has to wait for something before continuing
- The program wants to do something else while waiting

Use cases:

- JavaScript programs in a web browser are typically event-driven, and must wait for user input (e.g.: clicks)
- JavaScript-based servers wait for client requests to arrive over the network before they do anything

JavaScript mechanisms for asynchronous execution

- Callbacks are registered, and called when an event happens
- Promises are objects that represent not-yet available result of an asynchronous operation
- `async` and `await` provide a syntax for asynchronous programming to simplify Promise-based code

Callbacks for timers

- You can register a function
- It gets called when the Timer is fired (see previous lecture)

```
timerId = setTimeout(checkForUpdates, 60000);
```

- The callback function is called one minute after `setTimeout` is executed
- `checkForUpdates` is called once at the correct time *with no arguments*, and nothing more
- Use instead `setInterval` to have periodic execution. You can later stop the periodic execution with `clearInterval(timerId)`. Don't forget to save `timerId`

Example for timers

```
timerId = setTimeout(() => {  
  console.log("hello later"); // runs after 2 seconds  
, 2000)
```

- I can pass it more parameters:

```
const myFunction = (firstParam, secondParam) => {  
  ...  
}  
setTimeout(myFunction, 2000, firstParam, secondParam)
```

- Similarly to the periodic execution, you can cancel the timer before the callback is called:

```
clearTimeout(timerId)
```

Callbacks for events

- You have seen this already
 - Event-driven JavaScript programs register event handler functions for specified types of events (e.g.: 'click') in specified contexts (e.g.: confirmUpdateDialog's button)
 - The web browser invokes those functions whenever the specified events occur

```
let okay = document.querySelector('#confirmUpdateDialog button.okay');
okay.addEventListener('click', applyUpdate);
```

- `applyUpdate` gets executed when the user clicks on the button of the dialog

Callbacks for file system events

- Node.js processes files, networking, etc asynchronously
 - These operations can be time consuming, and are mediated by the operating system. The Node.js program can do something else in the meantime

```
import fs from "node:fs";
let options = {};// Object to hold options, initialized with default values here

fs.readFile("config.json", "utf-8", (err, text) => {
    if (err) {
        console.warn("Could not read config file:", err);
    } else {
        Object.assign(options, JSON.parse(text));
    }
    startProgram(options);
});
```

Callbacks hell

- Imagine that you have to write callbacks for everything that can happen, and register and deregister them as needed
- Imagine also that a callback can define and register another callback, and so on and so forth
- Image having to read code from your colleagues where callbacks are indented on 5 different levels, since they are callbacks defined into callbacks that are defined into callbacks etc etc
- If it does not seem fun, you are right
 - **CALLBACKS HELL**
- Solution: mechanisms to write synchronous-like code that is executed asynchronously

Agenda

- Exceptions
- Timers, callbacks and events
- Promises
- Async/await
- Fetch

Promises

- A promise is a proxy for a value that will eventually become available
 - You create it “around” a function (also known as executor) that takes two functions as parameters. One is executed if the executor completes correctly, one if there was a failure
 - There is no way to synchronously get the value of a Promise; you can only ask the Promise to call a callback function when the value is ready
- The Promise represents a computation that will eventually produce a value or throw an exception
 - Not good for repeated computation

Benefits of Promises

- This solves two problems of “normal” callback-based programming:
 - I don’t have callbacks inside callbacks inside callbacks (**callbacks hell**). Instead, I have Promise chains
 - Error handling. It is impossible to just throw an exception, since there is no way for that exception to propagate back to the initiator of the asynchronous operation. Promises standardize how to handle errors

Using a Promise

- Image that `getJSON(url)` returns a Promise, you can call its `then` method to register two callbacks:

```
getJSON(url).then(  
  jsonData => {  
    // This is a callback function that will be asynchronously  
    // invoked with the parsed JSON value when it becomes available.  
  }, err => {  
    // This code is executed when an exception is raised  
  }  
);
```

- If the function in the first `then()` returns a promise, you can call the `then()` method multiple times, each of the functions will be called on the promise returned by the previous function

Idiomatic best practices

- Call the `then()` method directly on the function invocation that returns a Promise, without assigning the Promise to an object
- Name the functions returning Promises with verbs
- Instead of passing a “reject” function, `catch()` the exception outside the `then()` invocation
 - As we discussed, the exception is propagated outward until it finds a `catch()`
- `function displayUserProfile(profile) { /* implementation omitted */ }`
- `function handleProfileError (err) { /* implementation omitted */ }`
- *// Notice how this line of code reads almost like an English sentence:*
- `getJSON("/api/user/profile").then(displayUserProfile).catch(handleProfileError);`

More terminology: States

- A Promise can be *fulfilled* (correct computation of the final value, fist callback is called)
- A Promise can be *rejected* (failure with computing the value, Exception raised, second callback / `.catch()` is called)
- Before that time, the Promise is *pending*. As soon as it is either *fulfilled* or *rejected*, the Promise is *settled*
- After the Promise is *settled*, it provide its *result* (either the computed value, or the Error) as soon as the `.then().catch()` is applied to the Promise
 - Maybe you are executing another function and you cannot yet execute the `.then().catch()`
 - Maybe you already did the `.then().catch()` and the callback will be called as soon as possible

Executing Promises sequentially

- Later we will study the `fetch()` method
 - Used to perform a REST request (e.g.: GET of a web page)
- Its execution returns a `response` with the headers of the interaction, but not the full resource
 - Large data transfer can take a lot of time, and it is better to know immediately if the HTTP transfer will fail for sure (e.g.: wrong host name)
- Calling `json()` on the `response` returns a Promise for the JSON encoding of the data
- Example without chaining, similar to callbacks hell:

```
fetch("/api/user/profile").then(response => {
    response.json().then(profile => { // Ask for the JSON-parsed body
        // When the body of the response arrives, it will be parsed as JSON and passed to this function
        displayUserProfile(profile);
    });
});
```

Chaining Promises

- A natural way to express a sequence of asynchronous operations.
- Linear chain of `then()` method invocations, without having to nest each operation within the callback of the previous one
- Example:

```
fetch("/api/user/profile")
  .then(response => response.json()) // Ask for the JSON-parsed body
  .then(profile => { // When the body of the response arrives, it will be parsed
    displayUserProfile(profile); // as JSON and passed to this function
  });

```

This `.then()`
returns a Promise

- Same meaning of: `.then(response => { return response.json(); })`

More complex example

- This example has one more step and error handling
- Each invocation of the `then()` method returns a new Promise, which is not fulfilled until the function passed to its `then()` is complete

```
fetch(documentURL) // Make an HTTP request
  .then(response => response.json()) // Ask for the JSON body of the response
  .then(document => { // When we get the parsed JSON
    return render(document); // display the document to the user
  })
  .then(rendered => { // When we get the rendered document
    cacheInDatabase(rendered); // cache it in the local database.
  })
  .catch(error => handle(error)); // Handle any errors that occur
```

This `.then()` returns a Promise



More terminology: Fates

- As soon as the callbacks registered using `then()` / `then().catch()` returns, the Promise is *resolved*. We can *resolve* a promise with another promise
- A Promise is *unresolved* if it is not *resolved*

Relation States vs Fates:

- A Promise is not *fulfilled* if the callback returned a *pending* Promise (see chaining in next slide)
- A *fulfilled* Promise is *resolved*. A *rejected* Promise is *resolved*
- An *unresolved* Promise is *pending*

<https://stackoverflow.com/questions/35398365/js-promises-fulfill-vs-resolve>

<https://github.com/domenic/promises-unwrapping/blob/master/docs/states-and-fates.md>

Resolved but not Settled?

- As soon as the callbacks registered using `then()` / `then().catch()` returns, the Promise is *resolved*. We can *resolve* a promise with another promise. Can next `then()` method be called already?

```
function c1(response) { // callback 1
    let p4 = response.json(); return p4; } // returns promise 4
function c2(profile) {displayUserProfile(profile);} // callback 2
let p1 = fetch("/api/user/profile"); // promise 1, task 1
let p2 = p1.then(c1); // promise 2, task 2
let p3 = p2.then(c2); // promise 3, task 3
```

- When `fetch()` ends its asynchronous job task 1 (promise 1 is resolved), the output of task 1 can be sent as input to `c1` (promise 1 is fulfilled)
- If `p2` gets fulfilled, `c2` is invoked, and task 3 begins. Anyway, when `c1` returns, it returns Promise `p4`, which can still be rejected, maybe `p4` will never provide a value, and `c2` cannot be invoked yet. Promise `p2` is resolved to `p4`, but `p2` cannot settle until `p4` settles.

Definition of Resolved

- “resolved” Promise means: the Promise has become associated with, or “locked onto”, another Promise or a non-Promise value
- In some cases, we don’t know yet whether p will be fulfilled or rejected, but the callback has no control anymore over that
- Promise p is “resolved” in the sense that its fate now depends entirely on what happens to something else (the Promise it returned)

Error Handling

- We already discussed that the second callback is not usually provided
- Idiomatic approach similar to try/catch/finally:
- `getJSON("/api/user/profile").then(displayUserProfile).catch(handleProfileError).finally(cleanUp);`
- If `getJSON()` ends correctly, `then()` is invoked; if an Exception is raised (in `getJSON()` or in `then()`), the `catch()` is called. After `then()` end correctly or `catch()` ends, `finally()` is run
- One more example (`recoverFromStageTwoError()` returns a Promise):

```
startAsyncOperation()  
    .then(doStageTwo)  
    .catch(recoverFromStageTwoError)  
    .then(doStageThree)  
    .then(doStageFour)  
    .catch(logStageThreeAndFourErrors);
```

Promises in Parallel

- You can create an array of Promises

```
promises = urls.map(url => fetch(url).then(r => r.text()));
```

- Then you can execute all of them in parallel:

```
Promise.all(promises)  
  .then(bodies => { /* do something with the array of strings */ })  
  .catch(e => console.error(e));
```

- The Promise returned by `Promise.all()` rejects as soon as any of the input Promises is rejected

Making Promises from a Promise

- Creating a Promise that encapsulate another Promise:

```
functiongetJSON(url) {  
    return fetch(url).then(response => response.json());  
}
```

- When the internal Promise fulfills, the promise returned by `getJSON()` fulfills as well
- Error handling:
 - Checking `response.ok` and the Content-Type header?
 - No, in this case it is easier: we just allow the `json()` method to reject the Promise it returned with a `SyntaxError` if the response body cannot be parsed as JSON

Making Promises from synchronous value

- Creating a Promise based on synchronous computation:
 - Compute your value
 - Use the static methods `Promise.resolve()` and `Promise.reject()`

```
Promise.resolve('Success').then(function(value) {  
    console.log(value); // "Success"  
});
```

- No real asynchronicity
- The Promise will be settled as soon as the computation is done
 - They will fulfill or reject **after** the current synchronous chunk of code has finished running
 - `console.log(value)` outputs “Success” after the current chunk of code ends

Making Promises from scratch

- Creating a Promise to execute code asynchronously:
 - `function longFunction(resolve, reject) { ... }`
 - `new Promise(longFunction)`
 - The function can call `resolve / reject` whenever it wants
- It is possible to implement Promise-based APIs out of code using asynchronous callbacks and events

Example of making Promises (from page 364)

```
import http from "http";
functiongetJSON(url) { // Create and return a new Promise
  return new Promise((resolve, reject) => {
    // Start an HTTP GET request for the specified URL
    request = http.get(url, response => { // called when response starts
      if (response.statusCode !== 200) {
        reject(new Error(`HTTP status ${response.statusCode}`));
      } else if (response.headers["content-type"] !== "application/json") {
        reject(new Error("Invalid content-type"));
      } else { // GET was fine. Register events to read the body of the response
        let body = "";
        response.setEncoding("utf-8");
        response.on("data", chunk => { body += chunk; });
        response.on("end", () => {
          resolve(body);
        });
      }
    });
  });
}
```

Callback

Calling the *reject* of the Promise

Event-oriented programming

Example of making Promises (from page 364)

```
// When the response body is complete, try to parse it
try {
    let parsed = JSON.parse(body);
    resolve(parsed); // If it parsed successfully, resolve the Promise
} catch(e) {
    // If parsing failed, reject the Promise
    reject(e);
});
});

// Reject immediately if http.get fails
request.on("error", error => {
    reject(error);
});
};

}
```

Agenda

- Exceptions
- Timers, callbacks and events
- Promises
- **Async/await**
- Fetch

Async/await

- An `async` function is a function that implicitly returns a promise and that can, in its body, `await` other promises in a way that looks synchronous
- The `await` keyword takes a Promise and turns it back into a return value or a thrown exception (if you are into an `async` function)
- Here are three functions called `x`, `y` and `z` that return promises:

```
async function x() {return "one";}  
let y = async function () {return "two";}  
let z = async () => "three"
```

- Inside one of the promises, I can `await`:

```
async function do_it() {  
    console.log("message is "+await y());  
}
```

Benefits of async/await

- Useful to forget about Promises and their complexity
- Given a Promise object p, the expression await p waits until p settles
 - If p fulfills, then the value of await p is the fulfillment value of p
 - If p is rejected, then the await p expression throws the rejection value of p
- When inside an async function there is an await promise1, your program stops until promise1 settles
 - Or actually, the function is put to sleep and Javascript interpreter can do something else
 - *Any code that uses await is itself asynchronous*

Idiomatic `await`

- It is placed before the invocation of a function that returns a Promise:
 - `let response = await fetch("/api/user/profile");`
 - `let profile = await response.json();`
- It is uncommon to bind the Promise to an identifier
- What if I am into a non-async function, or in the top level?
 - It is a Promise ...
 - `getHighScore().then(displayHighScore).catch(console.error);`

Awaiting multiple promises

- Let us imagine we have an async function:

```
async function getJSON(url) {  
    let response = await fetch(url);  
    let body = await response.json();  
    return body;  
}
```

- This is very "sequential":

```
let value1 = await getJSON(url1);  
let value2 = await getJSON(url2);
```

- This is executed in parallel (thus probably faster):

```
let [value1, value2] = await Promise.all([getJSON(url1), getJSON(url2)]);
```

- `Promise.all` was discussed in slide 32

Asynchronous loops

- Promises do not work for sequences of asynchronous events
- We cannot use regular `async/await` statements
- Solution: `for/await`

```
import fs from "node:fs";

async function parseFile(filename) {
    let stream = fs.createReadStream(filename, { encoding: "utf-8"});
    for await (let chunk of stream) {
        parseChunk(chunk); // Assume parseChunk() is defined elsewhere
    }
}
```

For/await loops

- It is Promise-based:
 - The asynchronous iterator produces a Promise
 - The for/await loop waits for that Promise to fulfill
 - The fulfillment value is assigned to the loop variable
 - The body of the loop is executed
 - Another Promise from the iterator is created and the loop repeats

One more example

- `const urls = [url1, url2, url3];`
- `const promises = urls.map(url => fetch(url));`
- `for await (const response of promises) {`
- `handle(response);`
- }

meaning

```
for(const promise of promises) {  
    response = await promise;  
    handle(response);  
}
```

Agenda

- Exceptions
- Timers, callbacks and events
- Promises
- Async/await
- Fetch

Performing HTTP/HTTPS requests

- Different methods can be used to perform HTTP requests
 - Old approach: XMLHttpRequest. Let's forget about it
- Let us align on the “best” method we currently have: `fetch`
- `fetch()` defines a Promise-based API for making HTTP and HTTPS requests:
 - It accepts a URL and the kind of request as parameters
 - It returns a Promise that fulfills to the `response` (it was possible to contact the server) or it rejects if the HTTP connection could not be done
 - The `response` is a promise that resolves to a `body` (if the request went fine, e.g.: response code 200) containing the data

Fetch basic examples

- Fetch with `then()`

```
fetch("/api/users/current") // Make an HTTP (or HTTPS) GET request
  .then(response => response.json()) // Parse its body as a JSON object
  .then(currentUser => { // Then process that parsed object
    displayUserInfo(currentUser);
  });
}
```

- Fetch with `async / await`

```
async function isServiceReady() {
  let response = await fetch("/api/service/status");
  let body = await response.text();
  return body === "ready";
}
```

Fetch: better GET example

- Make an HTTPS GET request
 - When we get a response, first check it if for a success code (200 to 299) and return a Promise for the body
 - Or throw an error
 - When the response.text() Promise resolves let us print the body
 - Or if anything went wrong, just log the error. If the user's browser is offline, fetch() itself will reject. If the server returns a bad response then we throw an error above
- ```
async function let_s_fetch() {
 fetch("https://www.cs.aau.dk")
 .then(response => {
 if (response.ok) {
 return response.text();
 } else { throw new Error(
 `Unexpected response status ${response.status}`);
 }
 })
 .then(body => { console.log("result is " + body); })
 .catch(error => {
 console.log("Error while fetching data:", error);
 });
}
```

# Setting request parameters and headers

```
async function search(term) {
 let authHeaders = new Headers();
 authHeaders.set("Authorization", `Basic ${btoa(`${username}:${password}`)})`);
 let url = new URL("/api/search");
 url.searchParams.set("q", term);
 let response = await fetch(url);
 if (!response.ok) throw new
 Error(response.statusText);
 let resultsArray = await response.json();
 return resultsArray;
}
```

Setting an auth header

/api/search?q=\${term}

# Parsing the body

- What can I do with request (= the result of a fulfilled fetch Promise)?
- Two most common way of parsing the result of the GET:
  - `response.json().then( ... )` to get the result as a JSON object
  - `response.text().then( ... )` to get the result as text
- Other useful methods:
  - `response.formData().then(...)` to parse the result as a FormData object (“multipart/ form-data” format). Common to send data to a server, but not for the response
  - Streaming! See next slide

# Fetch Streaming API

- Do not get a Promise out of the response
  - If you access the data in any way (.json(), .text(), etc), you cannot re-access it
- The response has method `getReader()` to get a stream reader object:
  - `let reader = response.body.getReader();`
  - `while(true) { // Loop until we exit below`
  - `let {done, value} = await reader.read();`
  - // Verify value is not null. Process the “value”. Parse the data. Check for errors
  - `if (done) { // If this is the last chunk,`
  - `break; // exit the loop`
  - `}`
  - `}`

# Fetch: a POST

- `let user = { name: 'Michele', surname: 'Albano' };`
- `let response = await fetch('/article/fetch/post/user', {`
- `method: 'POST',`
- `headers:`
- `{ 'Content-Type': 'application/json; charset=utf-8' },`
- `body: JSON.stringify(user)`
- `});`
- `let body = await response.json();`
- `// do something with the response`
- 
- The Content-Type can be other things, such as FormData or blobs

# Security: CORS

- Web browsers generally disallow `fetch()` to servers different from the one the main HTML document comes from
  - Exceptions: images and scripts
- Cross-Origin Resource Sharing (CORS) aims to safe cross-origin requests
- The browser automatically adds an “Origin” header to the request
- The server has to **explicitly** answer with a “Access-Control-Allow-Origin” header, or the interaction is cancelled
  - Meaning, the Promise returned by `fetch()` is rejected
- The “Origin” header cannot be overridden via the `headers` property

# Aborting a Fetch request

- Need to create an AbortController PRIOR TO the fetch( )
- Pass the signal property of the AbortController as the signal property in the options of the fetch( )
- Call the abort( ) method of the controller to abort the request
- Let's not get a Promise out of the response
  - If you access the data in any way (.json(), .text(), etc), you cannot re-access it
- `let controller = new AbortController();`
- `options.signal = controller.signal;`
- `setTimeout(() => { controller.abort(); }, 10000); // 10 seconds before aborting`
- `fetch(url, options).then( ...`

# Exercises

Look on the moodle:

<https://www.moodle.aau.dk/mod/page/view.php?id=1695548>

for the exercises

# Internetværk og Web-programmering

## Server-side programming

Forelæsning 7  
Michele Albano

Distributed, Embedded, Intelligent Systems



# Agenda

- More on `fetch()`
- Client-side storage
- Non-client-side programming: Node.js
- Node.js as HTTP server

# Fetch Streaming API

- Do not get a Promise out of the response
  - If you access the data in any way (.json(), .text(), etc), you cannot re-access it
- The response has method `getReader()` to get a stream reader object:
  - `let reader = response.body.getReader();`
  - `while(true) { // Loop until we exit below`
  - `let {done, value} = await reader.read();`
  - // Verify value is not null. Process the “value”. Parse the data. Check for errors
  - `if (done) { // If this is the last chunk,`
  - `break; // exit the loop`
  - `}`
  - `}`

# Streaming Response bodies

- The body property of a Response object is a ReadableStream object.
  - Do not call text() or json() if you want to show a progress bar

```
fetch('big.json')
 .then(response => streamBody(response, updateProgress))
 .then(bodyText => JSON.parse(bodyText))
 .then(handleBigJSONObject);

async function streamBody(response, reportProgress,
processChunk) {
 let expectedBytes = parseInt(response.headers.get("Content-
Length"));
 let bytesRead = 0;
 let reader = response.body.getReader(); // Read bytes
 let decoder = new TextDecoder("utf-8"); // Bytes to text
 let body = ""; // Text read so far
```

```
 while(true) { // Loop until we exit below
 let {done, value} = await reader.read(); // Read a chunk
 if (value) { // If we got a byte array:
 if (processChunk) { // Process the bytes
 let processed = processChunk(value);
 if (processed) {body += processed;}
 } else { // Otherwise, convert bytes
 body += decoder.decode(value, {stream: true});}
 if (reportProgress) { // If a progress callback was
 bytesRead += value.length; // passed, then call it
 reportProgress(bytesRead, bytesRead / expectedBytes);}
 }
 if (done) { // If this is the last chunk,
 break; // exit the loop
 }
 }
 return body; // Return the body text we accumulated
}
```

# POST for fetch()

- Interactions: GET, PUT, DELETE, POST
- To perform a POST, you specify the method, and what you send in a body, for example a string:

```
fetch(url, {
 method: "POST",
 body: "hello world"
}) .then
```

- It can be JSON data:

```
fetch(url, {
 method: "POST",
 headers: new Headers({ "Content-Type": "application/json" }),
 body: JSON.stringify(requestBody)
}) .then
```

# FormData for fetch( )

- Instead of using the “submit” of a HTML form, it is possible to submit a form programmatically
- formData( ) returns an object that can be sent using fetch( )
- By using this, you lock your message on a *multipart/form-data* encoding
- Very useful for a POST

```
let formData = new FormData().append('username', 'abc123')
.append('avatar', 'thisismyavatar');

fetch('https://example.com/profile/avatar', {
 method: 'POST',
 body: formData
})
.then(response => response.json())
.catch(error => console.error('Error:', error))
.then(response => console.log('Success:', JSON.stringify(response)))
```

# Agenda

- More on `fetch()`
- **Client-side storage**
- Non-client-side programming: Node.js
- Node.js as HTTP server

# On-browser storage

- HTTP is a stateless protocol.
  - When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Three options to store something client-side (on the browser):

- The **Web Storage API** consists of the localStorage and sessionStorage objects
  - Persistent objects that map string keys to string values
  - Can store large (but not huge) amounts of data
  - Difference: sessionStorage is deleted when the browser window/tab is closed
- **IndexedDB** is an asynchronous API to an object database that supports indexing
- **Cookies** were designed to remember (little) information about the user
  - Cookies are saved in name-value pairs like:
    - username = John Doe
  - Data in the cookies is transmitted with every HTTP request, even if the data is only of interest to the client

# Cookie manipulation with Javascript

- JavaScript can create, read, and delete cookies with the `document.cookie` property

- Creation:

```
document.cookie = "username=John Doe";
```

- Creation with expiry date (by default, the cookie is deleted when the browser is closed):

- `document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";`

- Access to cookies:

```
let x = document.cookie;
```

- `document.cookie` will return all cookies in one string much like:  
`cookie1=value; cookie2=value; cookie3=value;`

# Checking a cookie

- Let us verify if a cookie is set.
  - If so, it will display a greeting.
  - If not, it will display a prompt box, asking for the name of the user, and stores the username cookie for 365 days, by calling the `setCookie` function:
- JavaScript code:

```
function checkCookie() {
 let username = getCookie("username");
 if (username != "") {
 alert("Welcome again " + username);
 } else {
 username = prompt("Please enter your name:", "");
 if (username != "" && username != null) {
 setCookie("username", username, 365);
 }
 }
}
```

# Agenda

- More on `fetch()`
- Client-side storage
- Non-client-side programming: Node.js
- Node.js as HTTP server

# Node.js

- Node.js is a program that allows you to apply your JavaScript skills outside of the browser.
- It provides the program node to execute javascript files:
  - `node hello.js`
- Or it can be used in REPL (read-eval-print-loop) mode:
  - `node`
- Then, an interpreter gets available to evaluate javascript interactively
- For example, it is possible to type
  - `global. [TAB][TAB]`
- And it will provide everything that can be typed after “`global.`”

# Event loop

- Javascript code runs on a single thread, which performs “event loops”:
- Pay attention to how you write your code and avoid anything that could block the thread, like synchronous network calls or infinite loops
- Every time the event loop takes a full trip, we call it a tick
- Example: a function can be passed to `process.nextTick()`. It will be executed on the current iteration of the event loop, after the current operation ends

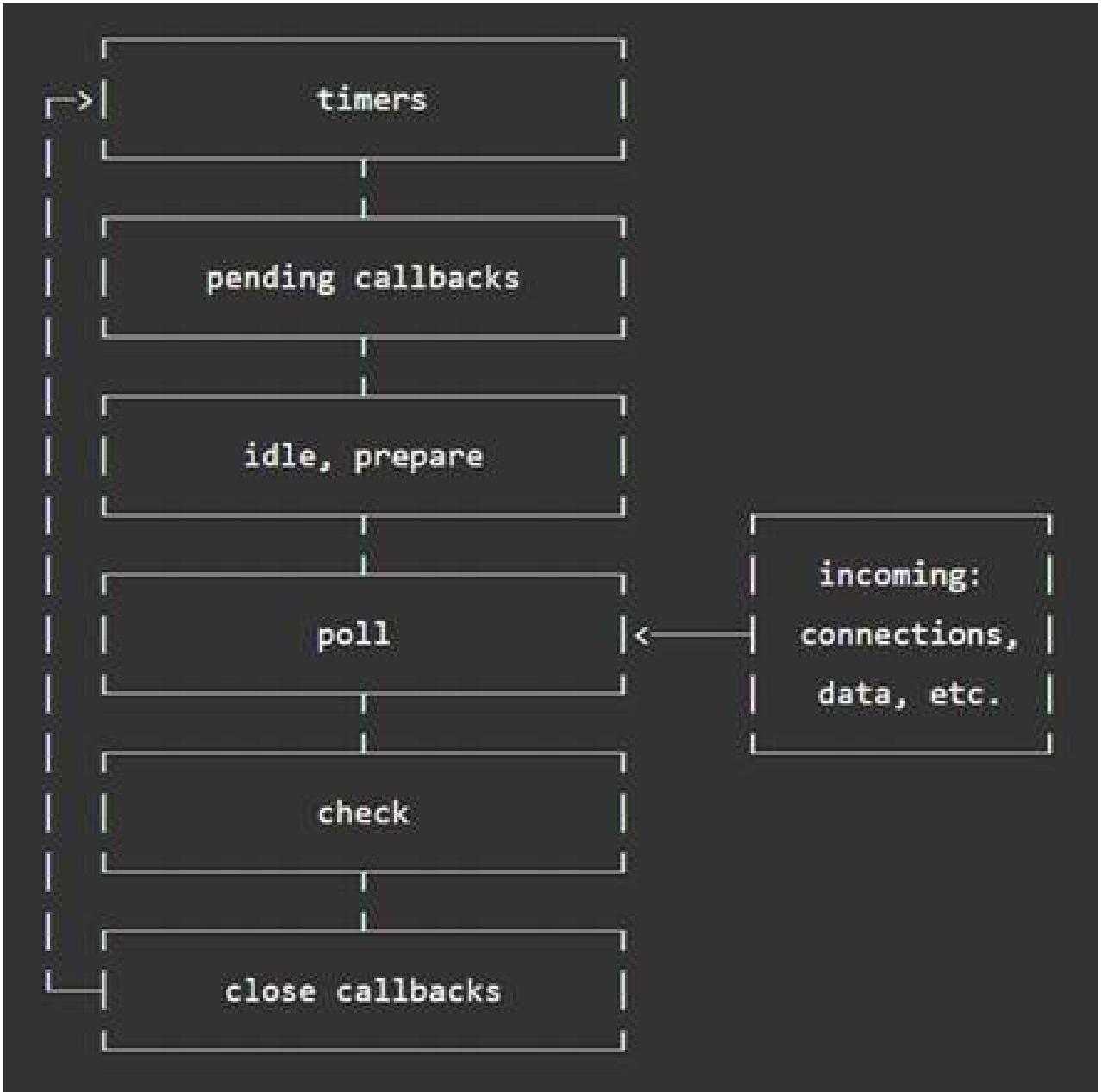
```
process.nextTick(() => {
 //do something
})
```

- Other options (both for next tick, or another tick in the future):
  - `setImmediate`: in a late phase of this tick
  - `setTimeout`: beginning of the proper tick

# Event loop of Node.js

JS: all code still runs on a single event loop.

- timers: this phase executes callbacks scheduled by `setTimeout()` and `setInterval()`.
- pending callbacks: executes I/O callbacks deferred to the next loop iteration.
- idle, prepare: only used internally.
- poll: retrieve new I/O events; execute I/O related callbacks (almost all with the exception of close callbacks, the ones scheduled by timers, and `setImmediate()`); node will block here when appropriate.
- check: `setImmediate()` callbacks are invoked here.
- close callbacks: some close callbacks, e.g. `socket.on('close', ...)`.



# Node.js: difference with browser's JS

- First of all, there are no *document*, *window* and all the other objects that are provided by the browser.

- There are a number of bindings that are provided by default, such as `process` (command line arguments) and `console`

```
process.argv.forEach((val, index) => {
 console.log(`#${index}: ${val}`)
})
process.exit(0)
```

- Environment variables:

```
console.log(process.env.PATH)
```

# Program life cycle

- Node programs do not exit until they are done running the initial file and until all callbacks have been called and there are no more pending events
- A Node-based server program that listens for incoming network connections will theoretically run forever because it will always be waiting for more events
- Forcing program to quit: `process.exit(0)`
- CTRL+C?
  - `process.on("SIGINT", ()=>{})`
- Exceptions in callbacks or event handlers must be handled locally or not handled at all
- Not to crash the program, register a global handler function:
  - `process.setUncaughtExceptionCaptureCallback(e => {`
  - `console.error("Uncaught exception:", e);`
  - `});`

# Packages and modules

- Code is shipped as packages and modules
- A package is a file or directory that is described by a package.json file, it can be installed with package managers (e.g.: npm), and it comprises one or more modules (libraries) grouped (or packaged) together.
- We will use the Standard ES6 modules:
  - `export` to make code available
  - `import` to access it
- Explicit file extension: `.mjs` ← for readability
- Package.json: type is “module” ← required

```
{ "main": "eventLoop.js", "type": "module" }
```
- ES6 modules can load CommonJS modules using the `import` keyword

# (Old) CommonJS packages and modules

- Old style
- You are inside a CommonJS file if you either:
  - Have a .cjs extension
  - Don't have "type" : "module" in the package.json file, for example you have "type" : "commonjs"
- Inside a CommonJS file, a module can be loaded by `require()`
  - It can be from a specified path:
    - `const config = require('/path/to/file');`
    - It can be looked up (`module.paths`):
      - `Let coolModule = require('find-me');`
  - The module's variables and functions are made available using `modules.export (...)`

# Npm package manager

- Example of an import:

```
const library = import('./library')
```

- Example of a package:

```
Shapes <- Package name
- Circle.js <----|
- Rectangle.js <--| Modules that belong to the Shapes package
- Square.js <----|
```

- You install the package (`npm install Shapes`), import it, and have access to the Circle, Rectangle, and Square modules.

# Promisifying: make promises easily

```
import {promisify} from 'util';
import {readFile} from 'node:fs';

const promisifiedReadFile = promisify(readFile);
promisifiedReadFile('fs_promisified.js', 'utf8')
 .then((data) => {console.log("received ", data)})
 .catch((err) => {
 console.log('Error', err);
});
```

# Streams

- Buffer class: a lot like a string
  - It is a sequence of bytes instead of a sequence of characters
  - Very common when reading data from files or from the network
  - Very common when manipulating binary data

```
let b = Buffer.from([0x41, 0x42, 0x43]); // <Buffer 41 42 43>
b.toString() // => "ABC"; default "utf8"
b.toString("hex") // => "414243"
let computer = Buffer.from("IBM3111", "ascii"); // Convert string to Buffer
for(let i = 0; i < computer.length; i++) { // Use Buffer as byte array
 computer[i]--; // Buffers are mutable
}
computer.toString("ascii") // => "HAL2000"
```

# Interaction with the file system

- Module “fs” provides methods to read and write files. Asynchronous mode:

```
import { readFile, writeFile } from 'node:fs';

readFile('fsExample.js', 'utf8', (err, data) => {
 if (err) throw err;
 console.log(data);
});

writeFile('message.txt', "hej world", 'utf8', (err) => {
 if (err) throw err;
 console.log('The file has been saved!');
});
```

# Synchronous read

- The “fs” module allows also for synchronous interactions
  - ```
import { readFileSync } from 'node:fs';
```
 - ```
const data = readFileSync('./message.txt',
```

```
{encoding:'utf8', flag:'r'});
```
  - 
  - ```
// Display the file data
```
 - ```
console.log(data);
```
- Not very JS-like: it could stop the JS thread for a long time!

# Promises and async/await file system interaction

```
// Promise-based asynchronous read
fs.promises
 .readFile("data.csv", "utf8")
 .then(processFileText)
 .catch(handleReadError);
```

```
// Or use the Promise API with await inside an async
function
async function processText(filename, encoding="utf8") {
 let text = await fs.promises.readFile(filename,
encoding);
 // ... process the text here...
}
```

# Agenda

- More on `fetch()`
- Client-side storage
- Non-client-side programming: Node.js
- Node.js as HTTP server

# Node.js as a webserver

- One of the most common usage of node.js is to implement a webserver
- First step:

```
import http from 'http';
```

- or:

```
import https from 'https';
```

# HTTP client

- `http.get()` / `https.get()`
  - It is a native API, there is no need to install third party modules
  - Quite low level
  - The response is a stream (“buffer” of bytes)
- However, please install the `node-fetch` module
  - support for Promises
  - same API as `window.fetch`
  - few dependencies

# The HTTP server

- Create a new Server object
- Call its listen() method to listen for requests on a specified port
- Register an event handler for “request” events
  - Read the client’s request (particularly the request.url property)
  - Write a response

# A touch of reality

- Node's built-in modules are all you need to write simple HTTP and HTTPS servers
- Production servers tend to use external libraries—such as the Express framework—that provide “middleware” and other higher-level utilities for backend web developers

# A simple webserver

- Let's now take a look at the three files in the BMI webserver

# Internetværk og Web-programmering

## Web-services and Web-APIs (in Node.js)

Forelæsning 8  
Michele Albano

Distributed, Embedded, Intelligent Systems



# Agenda

- SSE and Websockets
- Service Oriented Applications
  - basic notions (service producer, consumer, etc)
  - Stateless server, HATEOAS, idempotent operations
- Representational State Transfer (REST)
- Frameworks, and API Code generation
- Testing with Postman

# Server-Sent Events

- In HTTP (and HTTPS), the client initiates communication
  - Web browser
  - `fetch()`
- Some web apps need to receive data from the server *when the server wants*
- Solution: Server-Sent Events:
  - The client makes a request to the server
  - Connection is kept open
  - When the server has data to send, it writes them to the connection
- From the client's point of view, the server answers client's request in a slow and bursty way with pauses
- Network connections will close automatically after some time
  - The client reopens the connection (repeats the initial request) whenever it detects the connection was closed

# How is SSE?

- Pros:
  - Quite efficient (low delays)
- Cons:
  - Consumes resources on the server (TCP port etc for the active connection)
- Practical usage: EventSource API
- Client-side:
  - The client creates a EventSource around the URL of the server
  - The client receives events through the EventSource

```
let ticker = new EventSource("stockprices.html");
ticker.addEventListener("bid", (event) => {
 displayNewBid(event.data),
})
}
```

String describing the “type” of the event, as set by the server

String sent by the server

# SSE: server-side

- It must have an endpoint to receive the EventSource objects and save them:

```
server.on("request", (request, response) => {
 // Parse the requested URL
 let pathname = url.parse(request.url).pathname;
 if (pathname == "/chat" && request.method == "GET") {
 clients.push(response);
 }
}
```

- Whenever data must be sent back, use the `response` functions:

```
clients.forEach(client => client.write("event: chat\ndata: hallo\n\n"));
```



# SSE full example

- Let us now take a look at the example from the book

# Websockets

- Introduced with HTTP5 (created 2008, recommended 2014)
  - State-full, Full duplex peer to peer, Low latency, String and binary protocols.
- Keeps a connection open for two-way communication
- Allows the webserver to send content without first being requested

## • Lifecycle

- One peer (a client) initiates the connection by sending an HTTP handshake request.

**State of the WebSocket:**

WebSocket .CONNECTING  
This WebSocket is connecting.

WebSocket .OPEN  
This WebSocket is ready for communication

WebSocket .CLOSING  
This WebSocket connection is being closed

WebSocket .CLOSED  
Either the WebSocket has been closed (no further communication), or initial connection attempt failed
- The other peer (the server) replies with a handshake response.
- The connection is established. From now on, the connection is completely symmetrical.
- Both peers send and receive (binary) messages.
- One of the peers closes the connection.

# Practicality for websockets

Server-side, need to install the module:

```
npm install websocket
```

- The URL does not start with HTTP/HTTPS
  - It uses WS/WSS
  - Most browsers block WS and/or raise a warning for security reasons
- The browser first establishes an HTTP connection, then sends an Upgrade: websocket header requesting to switch to the WebSocket protocol
- Both client and server must now speak the WebSocket protocol

# Websockets small example

- Let us now take a look at some code

# Agenda

- SSE and Websockets
- Service Oriented Applications
  - basic notions (service producer, consumer, etc)
  - Stateless server, HATEOAS, idempotent operations
- Representational State Transfer (REST)
- Frameworks, and API Code generation
- Testing with Postman

# Service-oriented architectures 1/3

- SOA is a set of principles for the design, deployment and management of both applications and software infrastructure using sets of *loosely coupled* services that can be *dynamically discovered* and that allow a *producer* and a *consumer* to *communicate with each other* or are *coordinated* through *choreography* to provide *enhanced services*
- Four properties of a service:
  - It is a black box for its consumers
  - It is self-contained (loose-coupling between services)
  - It may consist of other underlying services
  - It should be stateless

# Service-oriented architectures 2/3

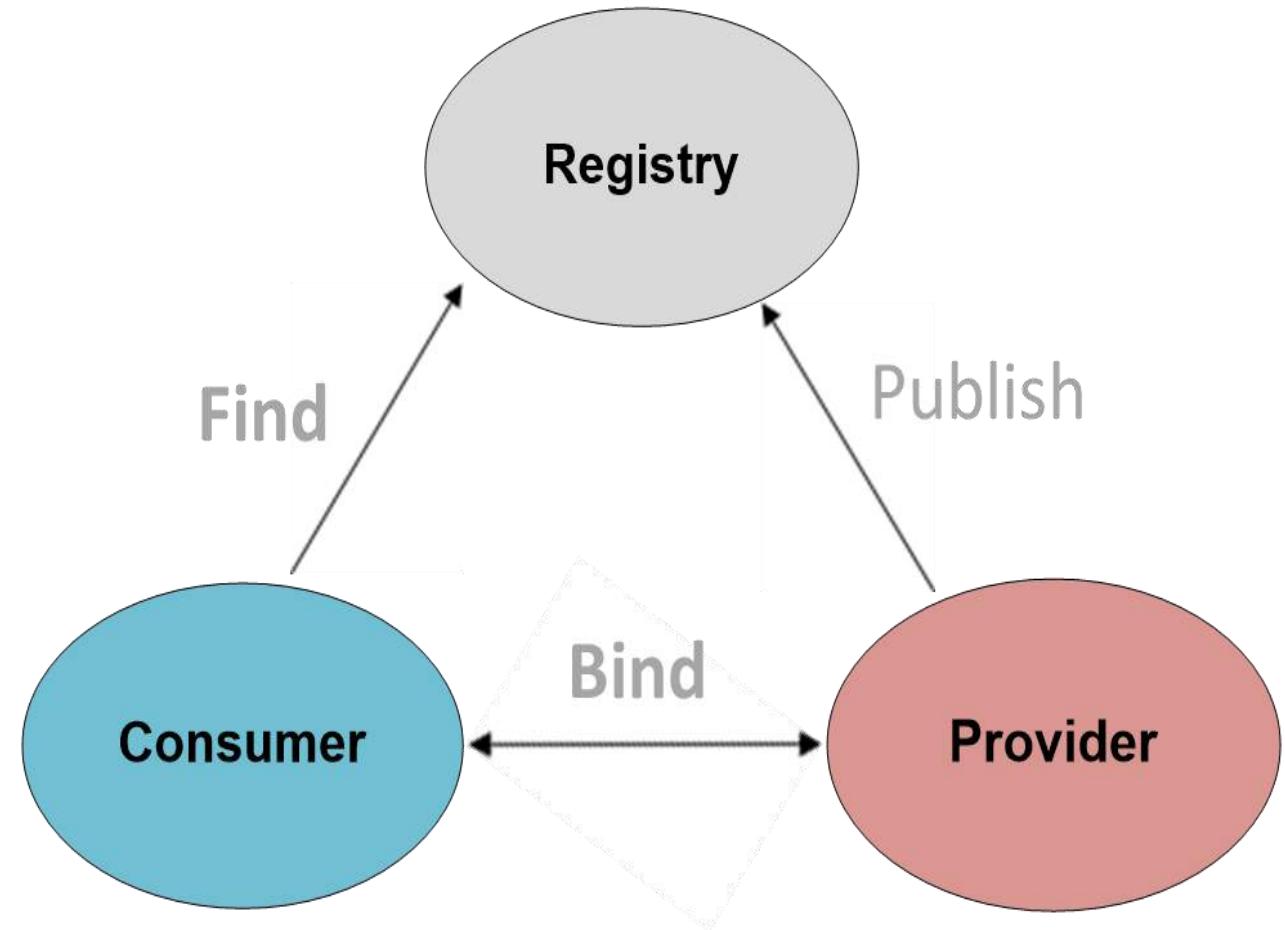
- It is a black box for its consumers
  - A service presents a simple interface articulated in endpoints to the requester that abstracts away the underlying complexity
  - The SOA infrastructure will provide standardized access mechanisms to *discover* services with service-level agreements
- It is self-contained (loose-coupling between services)
  - The consumer of the service is required to provide only the data stated on the interface definition, and to expect only the results specified on the interface definition
  - In the context of web services, loose coupling refers to minimizing the dependencies between services in order to have a flexible underlying architecture (reducing the risk that a change in one service will have a knock-on effect on other services)

# Service-oriented architectures 3/3

- It may consist of other underlying services
  - It allows users to combine and reuse them in the production of applications
  - It should be based on open standards. Open standards ensure the broadest integration compatibility opportunities
- It should be stateless
  - The service does not maintain state between invocations
  - If a transaction is involved, the transaction is committed and the data is saved somewhere
    - The id of the transaction can provide context (sessions)

# Design patterns

- 3 main roles:
  - Service provider / publisher
    - Offers the service, registers it in the service broker for consumers to find
  - Service consumer
    - Retrieves service providers from service broker, then uses the services
  - Service broker / registry / repository



# SOLID principles for SOA

- Single-responsibility Principle: each service should be specialized
- Open/Closed Principle: component should be open for extension but closed for modification
  - service orchestration
- Liskov Substitution Principle: if  $q(x)$  is a property provable about  $x$  of type  $T$ ,  $q(y)$  should be provable for objects  $y$  of type  $S$  where  $S$  is a subtype of  $T$ 
  - clients consume services, which consume lower granularity services
- Interface Segregation Principle: clients should not be forced to depend upon interfaces that they do not use (multiple small services better than one fat service interface)
- Dependency Inversion Principle: an interface is an abstraction between a higher and a lower level component
  - you can replace interface implementations without changing the interface

# Microservice Architecture

- MSA is a modern interpretation of SOA
- Services are still processes that communicate with each other over the network
- Emphasis on continuous deployment and other agile practices

# SOA vs MSA: more info 1/3

- Service Granularity:
  - MSA: generally single-purpose services that do one thing really, really well
  - SOA: service components can range in size. Coarse-grained to be useful to more applications
- Component Sharing:
  - one of the core tenets of SOA.
  - MSA tries to minimize on sharing through “bounded context”  
(coupling of a component and its data as a single unit with minimal dependencies)

# SOA vs MSA: more info 2/3

- Middleware vs API layer:
  - MSA provides an API layer
  - SOA has a messaging middleware component (provides mediation and routing, message enhancement, message, and protocol transformation)
- Remote services:
  - SOA architectures rely on messaging (AMQP, MSMQ)
  - Most MSAs rely on two protocols – REST and simple messaging (JMS, MSMQ), and the protocol found in MSA is usually homogeneous.

# SOA vs MSA: more info 3/3

- Heterogeneous interoperability:
  - SOA promotes the propagation of multiple heterogeneous protocols through its messaging middleware component: to integrate several systems using different protocols in a heterogeneous environment
  - MSA attempts to simplify the architecture pattern by reducing the number of choices for integration: all your services could be exposed and accessed through the same remote access protocol

# SOA vs MSA: bottom line

- SOA is better suited for large and complex business application environments that require integration with many heterogeneous applications using a middleware component
- Microservices are better suited for smaller and well-partitioned, web-based systems in which microservices give you much greater control as a developer

# Agenda

- SSE and Websockets
- Service Oriented Applications
  - basic notions (service producer, consumer, etc)
  - Stateless server, HATEOAS, idempotent operations
- **Representational State Transfer (REST)**
- Frameworks, and API Code generation
- Testing with Postman

# Representational State Transfer

- REST is an approach to services with a very constrained style of operation, applying “verbs” to “nouns”
  - Nouns are the URLs that identify web resources
  - Verbs are the HTTP operations *GET*, *PUT*, *DELETE* and *POST* (*and lately PATCH*) to manipulate resources
- GET to retrieve the representation of a resource
- POST to add a new resource
- PUT to update the representation of a resource using a new one
- PATCH to change part of the representation of a resource
- DELETE to discard a resource

# The tenets of REST

- Resources are identified by uniform resource identifiers (URIs)
- Resources are manipulated through their representations
- Messages are self-descriptive and stateless
- Multiple representations are accepted or sent
- Hypertext is the engine of application state

# Statelessness

- Two kinds of state
- **Application state** is the information necessary to understand the context of an interaction (e.g.: authentication, session)
  - In REST, all messages must include all *application* state as part of the content transferred from client to server back to client
- Changes in **resource state** are unavoidable
  - Someone has to POST new resources before others can GET them
  - REST is about avoiding implicit or unnamed state; resource state is named by URIs
- No application state means:
  - It prevents partial failures
  - It allows for substrate independence
    - Load-balancing
    - Service interruptions

# Hypermedia as the Engine of Application State

- Web application as a state machine:
  - State machines fit into REST when the states are expressed as resources with links indicating transitions
- HATEOAS:
  - hypermedia links in the response contents so that the client can dynamically navigate to the appropriate resource by traversing the hypermedia links
- A REST client hits an initial API URI and uses the server-provided links to dynamically discover available actions and access the resources it needs
  - The client need not have prior knowledge of the service or the different steps involved in a workflow
  - Thus, HATEOAS allows the server to make URI changes as the API evolves without breaking the clients

# Idempotent operations

- A request method is considered "idempotent" if the intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request.
- By this specification, GET, PUT and DELETE methods are idempotent.
- Do multiple requests return the same response every time they are called?
  - GET with Authentication: returns 401 (invalid authentication) vs with valid authentication (e.g.: 200).
  - PUT: either a 201 (resource is created) or 200/204 if the resource is updated
  - DELETE: 204 if the resource is deleted or a 404 if the resource was already deleted
- Due to this, idempotency in REST does not mean that consecutive calls to the same method and resource must return the same response, but rather that consecutive calls to the same method and resource **MUST** have the same intended effect on the server.

# More Best Practices for REST MSA 1/2

- Group together functionalities that are semantically related by means of paths to endpoints
  - They “are” the “same” service
  - Important also for maintainability: only a single reason to alter a service
  - Same endpoint only for same functionality (using different verbs if needed)
  - Separate also where data are stored (to have real isolation between services)
- Could somebody on his right mind deploy service A without service B?
  - Even though they could deploy service B without service A
- Dependencies: one endpoint for user authentication, another for payment processing
- Versioning: /megaservice/v1 and /megaservice/v2

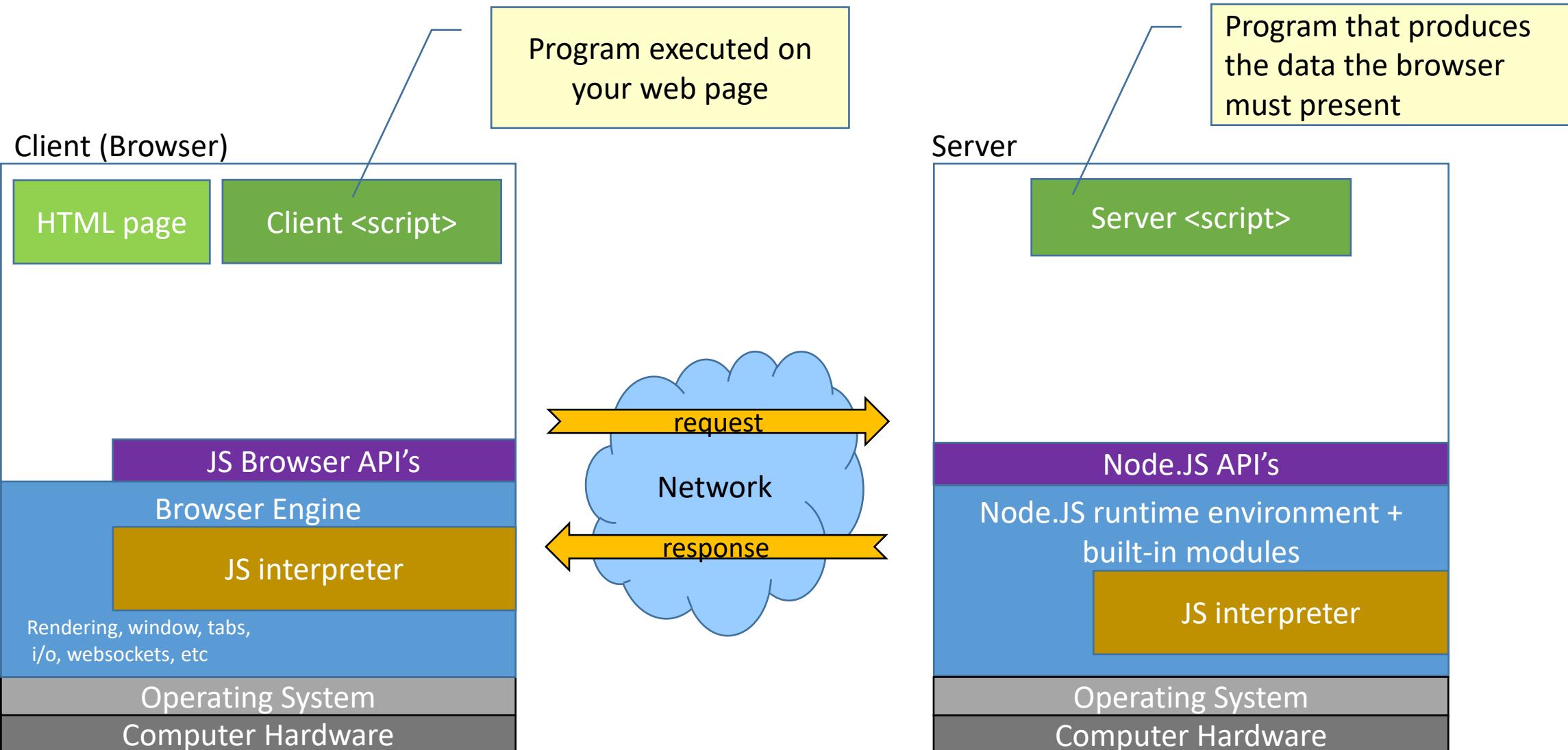
# More Best Practices for REST MSA 2/2

- REST API is good since no need not install any additional software or libraries
- Use asynchronous communication, for loose coupling (synchronization decoupling)
  - Publish/subscribe
  - Client exposing a service to receive responses
- Use an API Gateway as entry point for clients. Instead of calling services directly, clients call the API gateway, which forwards the call to the appropriate services on the back end
  - It decouples clients from services. Services can be versioned or refactored without needing to update all the clients
  - Services can use messaging protocols that are not web friendly, such as AMQP
  - The API Gateway can perform other cross-cutting functions such as authentication, logging, SSL termination, and load balancing

# Agenda

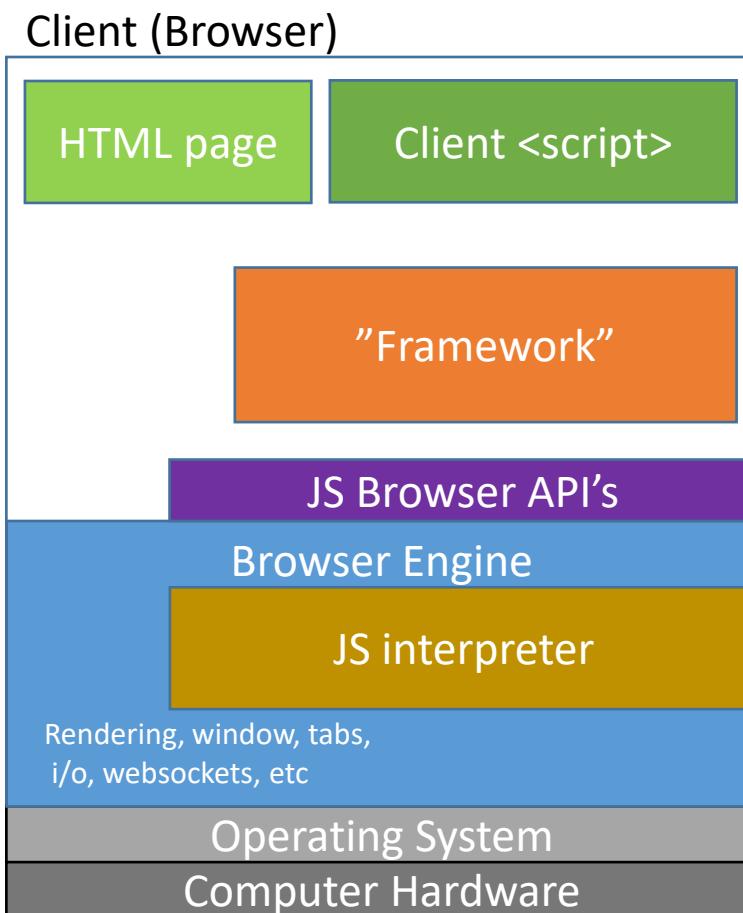
- SSE and Websockets
- Service Oriented Applications
  - basic notions (service producer, consumer, etc)
  - Stateless server, HATEOAS, idempotent operations
- Representational State Transfer (REST)
- Frameworks, and API Code generation
- Testing with Postman

# Overall architecture: "frameworks"



# Overall architecture: “client-side frameworks”

- **Framework:** JS libraries to simplify actions usually taken on most client-side scripts



Examples of frameworks:

- React,
- Vue,
- Angular,
- QueryJS,
- Bootstrap,
- ...

**Do not use them for the IWP exam!**

<https://www.freecodecamp.org/news/complete-guide-for-front-end-developers-javascript-frameworks-2019/>

<https://www.academind.com/learn/javascript/jquery-future-angular-react-vue/>

# Overall architecture: “server-side frameworks”

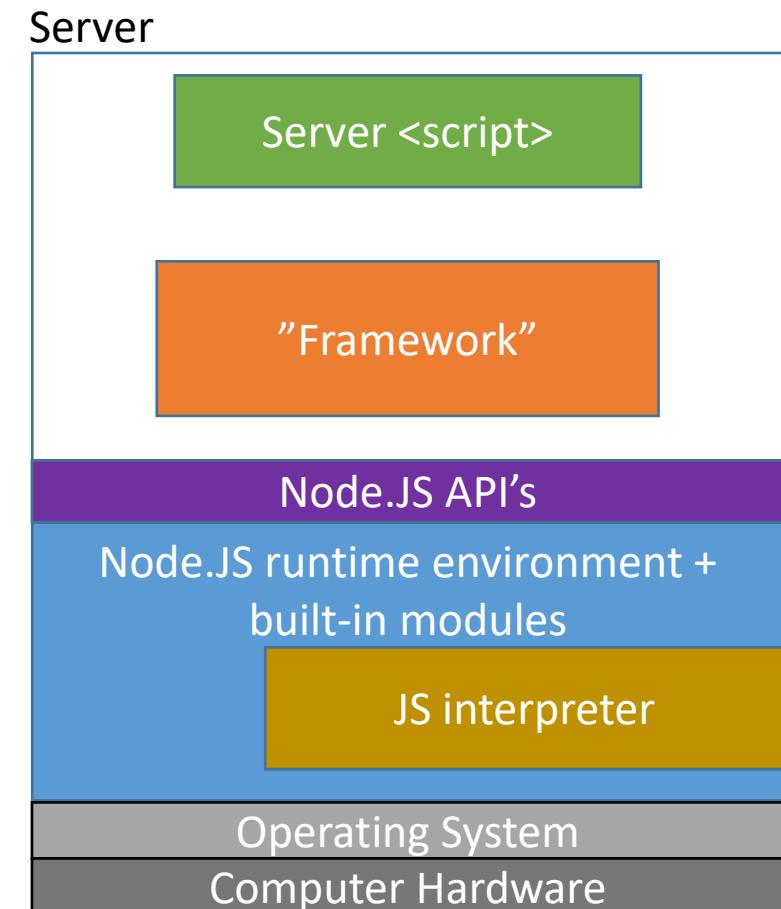
- **Framework:** JS libraries to simplify actions usually taken on most server-side scripts, especially when building a webserver

Examples of server-side frameworks:

- Express.js
- Meteor
- Loopback
- Backbone
- Koa
- ...

Do not use them for the IWP exam!

<https://scotch.io/bar-talk/10-node-frameworks-to-use-in-2019>



# Express.js

- Express.js <https://en.wikipedia.org/wiki/Express.js>
  - A very popular web application framework built to create Node.js Web based applications.
- Core features of Express framework:
  - Allows to set up middleware to respond to HTTP Requests.
  - Defines a routing table which is used to perform different action based on HTTP method and URL. See <https://expressjs.com/en/starter/basic-routing.html>
  - Allows to dynamically render HTML pages based on passing arguments to templates.

# Express.js Hello World

- Create a file named app.js and add the following codes:

```
import express from 'express';
let app = express();
app.get('/', function (req, res) {
 res.send('Hello World!');
});
app.listen(3000, function () {
 console.log('app.js listening to http://localhost:3000/');
});
```

- Run the app.js in the server:

```
node app.js
```

- Then, load <http://localhost:3000/> in a browser to see the output.

# Basic Routing

- Each route can have one or more handler functions, which are executed when the route is matched.
- Route definition takes the following structure:

```
app.METHOD(PATH, HANDLER);
```
- Where:
  - app is an instance of express.
  - METHOD is an HTTP request method, in lower case.
  - PATH is a path on the server
  - HANDLER is the function executed when the route is matched.
- Example: GET method route:

```
app.get('/', function (req, res) {
 res.send('Get request to the homepage');
});
```

# Response Methods

- Quite similar to “normal” Javascript
- You call methods on the response object (res)

| Method           | Description                                                                           |
|------------------|---------------------------------------------------------------------------------------|
| res.download()   | Prompt a file to be downloaded.                                                       |
| res.end()        | End the response process.                                                             |
| res.json()       | Send a JSON response.                                                                 |
| res.jsonp()      | Send a JSON response with JSONP support.                                              |
| res.redirect()   | Redirect a request.                                                                   |
| res.render()     | Render a review template.                                                             |
| res.send()       | Send a response of various types.                                                     |
| res.sendFile()   | Send a file as an octet stream.                                                       |
| res.sendStatus() | Set the response status code and send its string representation as the response body. |

# Designing and generating a REST API

- The service is provided at an URL
- The service allow access through a set of endpoints
- Each endpoint allows for REST verbs/operations (GET, POST, PUT, PATCH, DELETE)
- The input data can be JSON or XML
- The output is a HTTP status code, and sometimes data is JSON or XML

# OpenAPI

- OpenAPI Specification is an API description format for REST APIs, to describe:
  - Available endpoints and operations on each endpoint
  - Operation parameters Input and Output for each operation
  - Authentication methods
  - Contact information, license, terms of use and other information.
- An OpenAPI document itself is a JSON object, which may be represented and written either in JSON or YAML format



# OpenAPI 3.0

- We will use OpenAPI version 3 (the latest one)  
<https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.3.md>

- The OpenAPI document is composed by a number of parts
- Most important parts (except for the “headers” openapi, info and servers):
  - Paths
  - Components

# Paths

- It is a series of path objects
- It is the “path” part of the resource URL, followed by the operations it accepts
- For each operation:
  - The specific of the input data
  - A list of HTTP status code, and optionally the specific of return data for each case

# Components

- It contains a set of reusable objects
  - schemas
  - responses
  - parameters
  - etc
- All objects defined within the components object will have no effect on the API unless they are explicitly referenced

# Example

```
openapi: 3.0.3
info:
 title: ping test
 version: '1.0'
servers:
 - url: 'http://localhost:8000/'
paths:
 /ping:
 get:
 operationId: pingGet
 responses:
 '201':
 description: OK
```

# Trying OpenAPI out

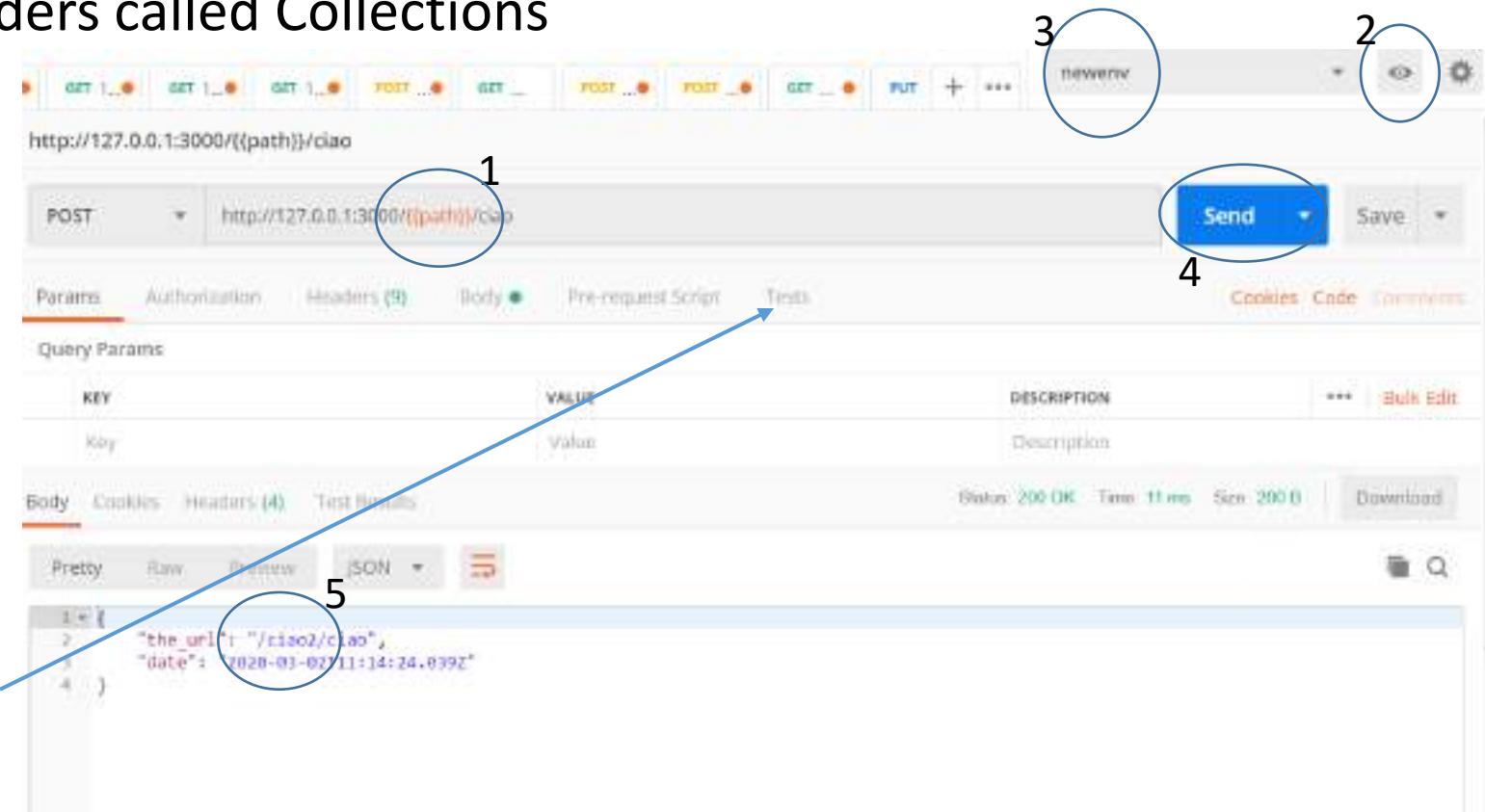
- Write an OpenAPI specification
- Use one of the generators to generate a server:
  - <https://editor.swagger.io/>
  - <http://api.openapi-generator.tech/index.html>
- The first one is easier to use, the second one must be installed but support many more languages
- Download the generated code for server and client
  - Or you can use Postman as client (next topic)
- Implement the business logic
- Try it out: Compile, execute, etc

# Agenda

- SSE and Websockets
- Service Oriented Applications
  - basic notions (service producer, consumer, etc)
  - Stateless server, HATEOAS, idempotent operations
- Representational State Transfer (REST)
- Frameworks, and API Code generation
- **Testing with Postman**

# Postman

- Tool useful to test APIs
  - Set up GET/PUT/POST/DELETE/etc requests and see responses
  - Save requests to repeat later (History)
  - Organize requests into folders called Collections
- Parametrization of requests → → → →
  1. Parametrize using {{..}}
  2. Set up an environment
  3. Apply it
  4. Execute the request
  5. See the result
- Possible to create “tests”
- Automate with CLI



**END**

# Internettværk og Web-programmering

## Introduktion til netværk

Forelæsning 9  
Brian Nielsen

Distributed, Embedded, Intelligent Systems



# Agenda

1. Struktur af Internettet
2. Packet switching princip
3. Forsinkelse, Gennemløbsrate (Throughput), og Flaskehalse
4. Internet Protokol stakken

# Struktur af Internettet

Hvilke komponenter består nettet af?

Hvordan er det opbygget i netværk af netværk?

# Hvad består Internettet af?



- Milliarder af forbundne “computere” :
  - **hosts = end systems**
  - Afvikler netværks-applikationer



- **Kommunikations-forbindelser (links)**
  - fiber, kobber, radio, satellite,...
  - Transporterer data med en vis transmissions rate:
  - (Mega) bits-per-sekund (Mbps)



- **packet switches:** enhed, der videresender data-pakker i nettet
  - **routers** og **Lag 2 -switches**



- **Netværk:** samling af hosts, routere, links, som administreres af en organisation

# Hvad består Internettet af? "Things"

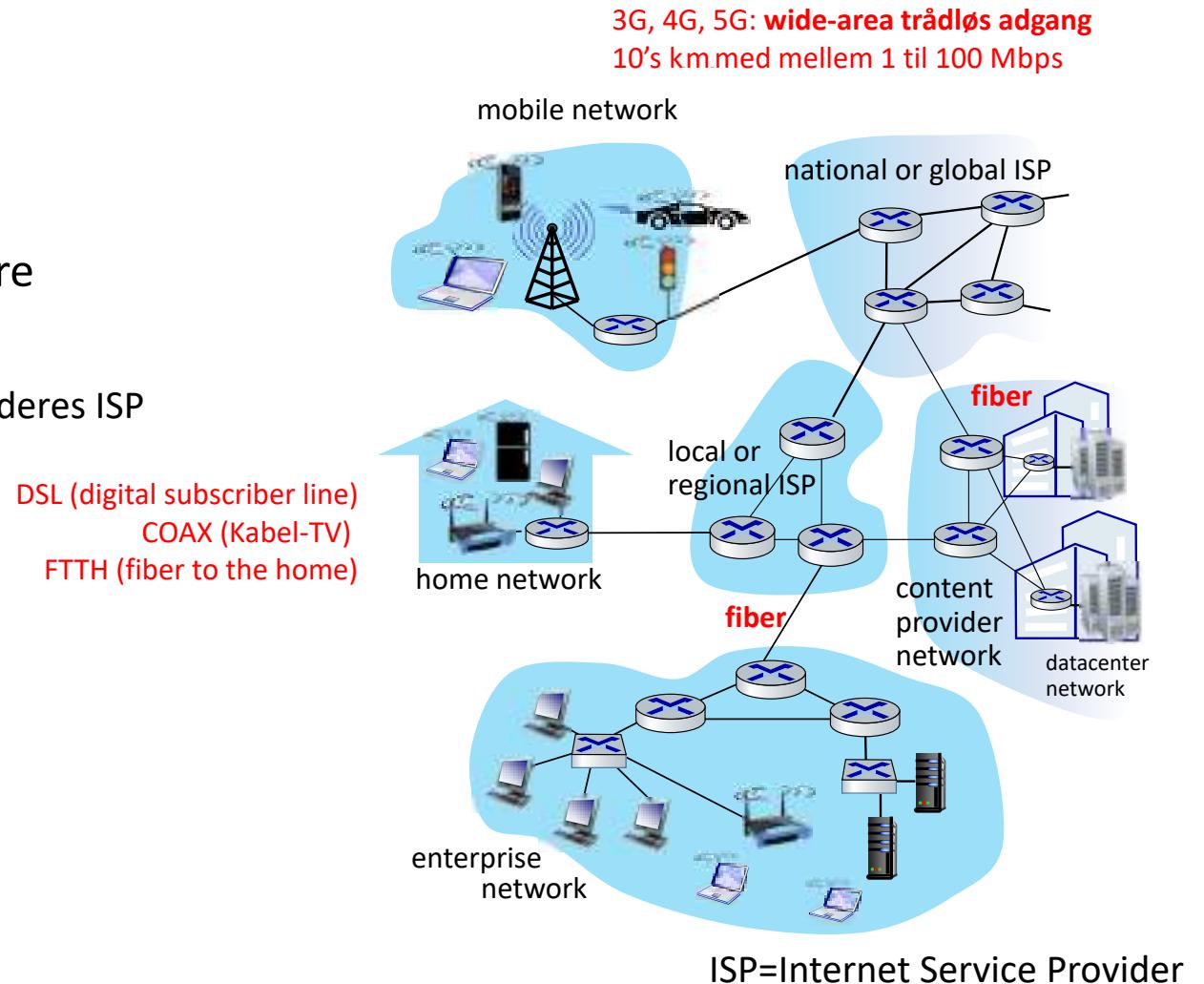


- Nu også end-systemer som sensorer, robotter, maskiner, TV, radiator termostater,...
  - Internet of things
  - Cyber-physical systems

<http://cityprobe.ciss.dk/>

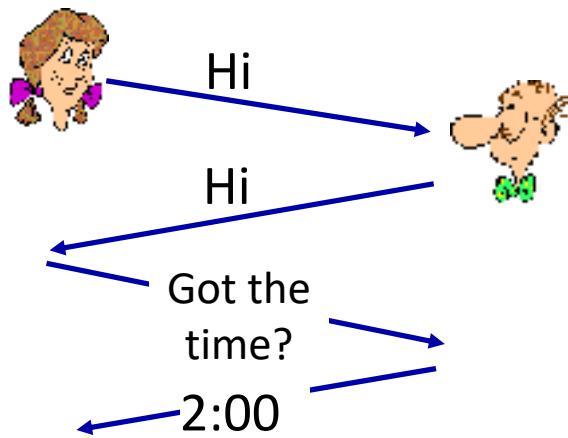
# Simpel model for struktur af Internettet

- Netværks-udkant (edge):
  - hosts = end systems
  - Klienter og servere
  - Servers, typisk placeret i datacentre
- Adgangs netværk (access)
  - Det yderste led, forbinder abonnenter til deres ISP
  - Lokal net + forbindelse til ISP
  - Trådede (wired) forbindelser
  - Trådløse (wireless) forbindelser
- Netværks kernen (core):
  - Sammenkoblede ISP routere
  - Hierarkier
  - **Netværk af netværk**
- Links: transmissionsmediet mellem to knudepunkter (hosts/router)

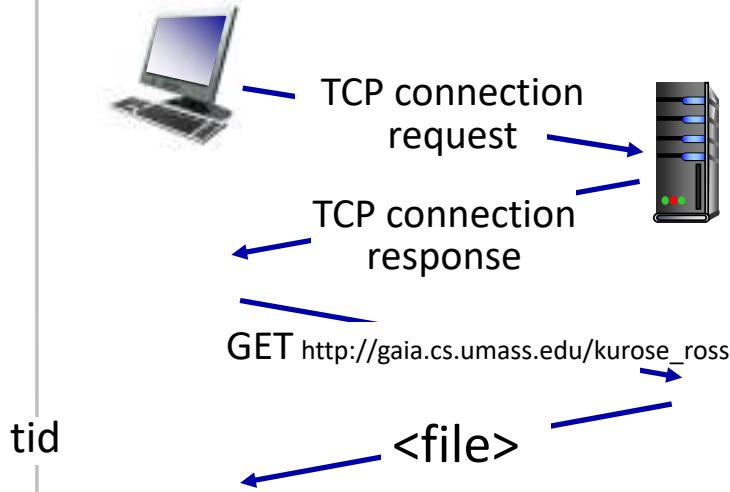


# Hvad består Internettet af?

## Mellem mennesker

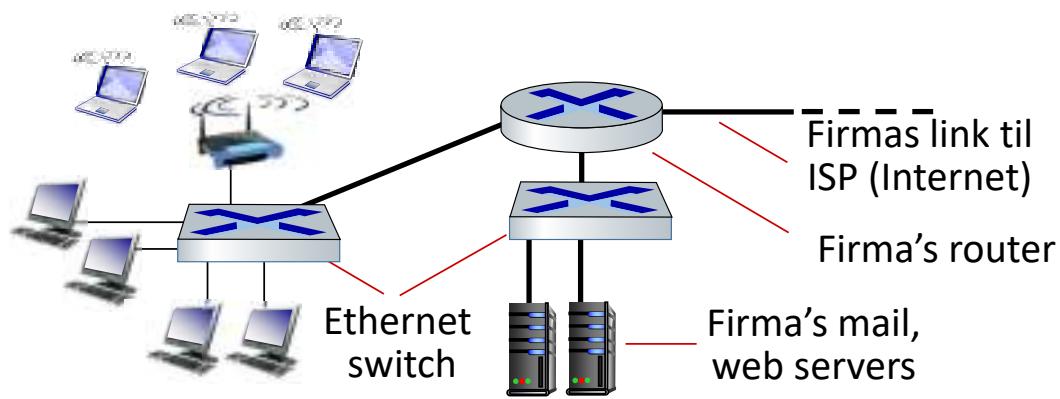


## Mellem computer



- **Protokoller:** Regler, der styrer afsendelse og modtagelsen af meddelelser
  - Dataindhold og format af meddelelser
  - Hvilke handlinger skal der ske ved modtagelse eller afsendelse af meddelelser?
  - e.g., TCP, IP, HTTP, Skype, 802.11
- **Internet standarder**
  - Beskrevet i "RFCs": Request for comments (<http://www.rfc-editor.org/standards>)
  - af IETF: Internet Engineering Task Force (<https://www.ietf.org/about/>)
- **Protokollerne er implementeret i en masse software!**

# Access Netværk i Firmaer (Ethernet)



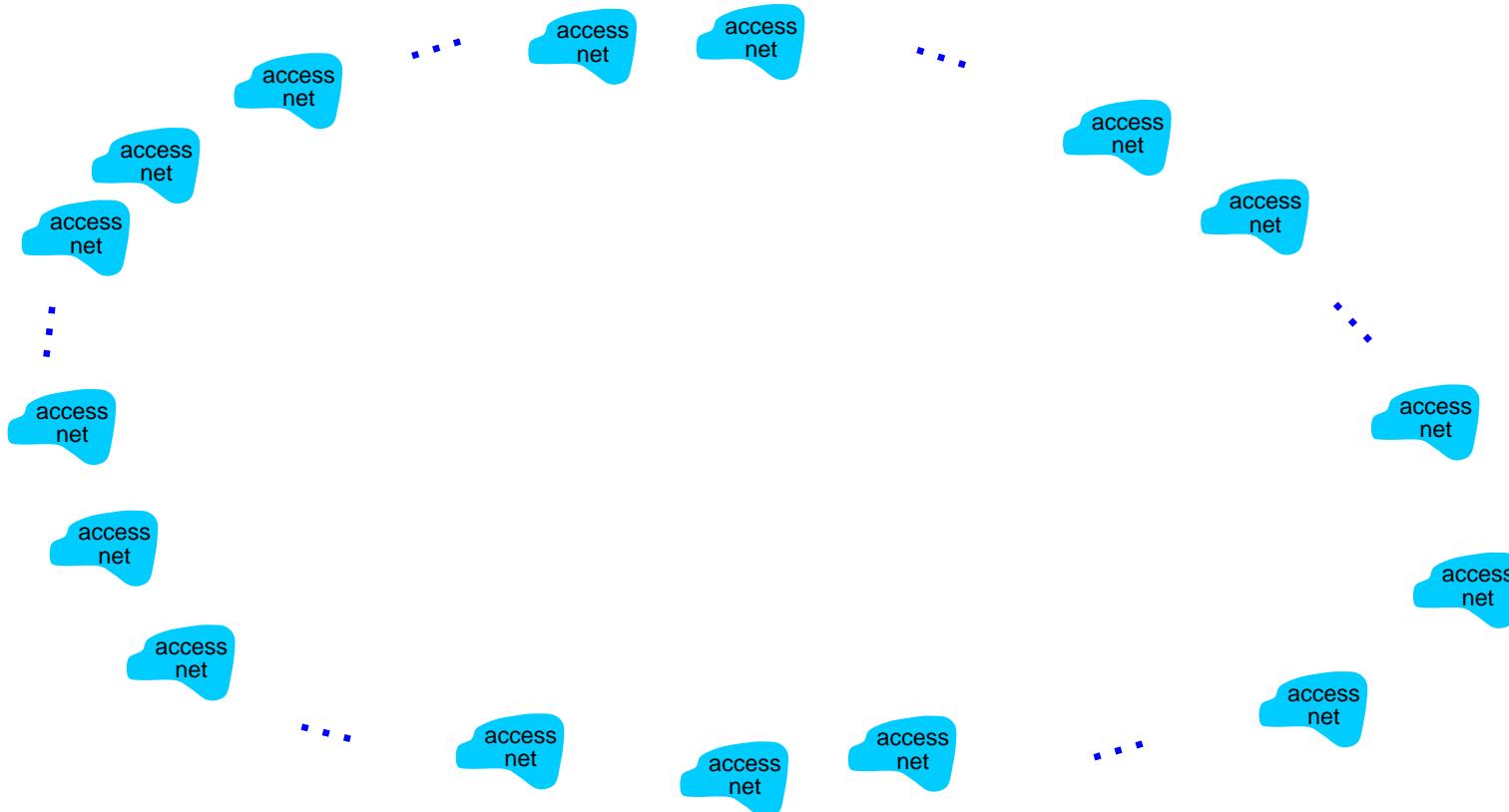
- Flere lokale netværk typisk brugt i firmaer, universiteter, etc.
- Nu til dags er end-systemer typisk forbundet til en Ethernet (Lag 2) switch
  - Den mest brugte netværksteknologi i adgangs netværket
  - Faste forbindelse med 10 M bps, 100 M bps, 1G bps, 10G bps transmissions rater
- WiFi: Trådløs, access-points 11, 54, 450 Mbps



Ethernet fysisk medium:  
Isolerede par af snoede kobber ledninger

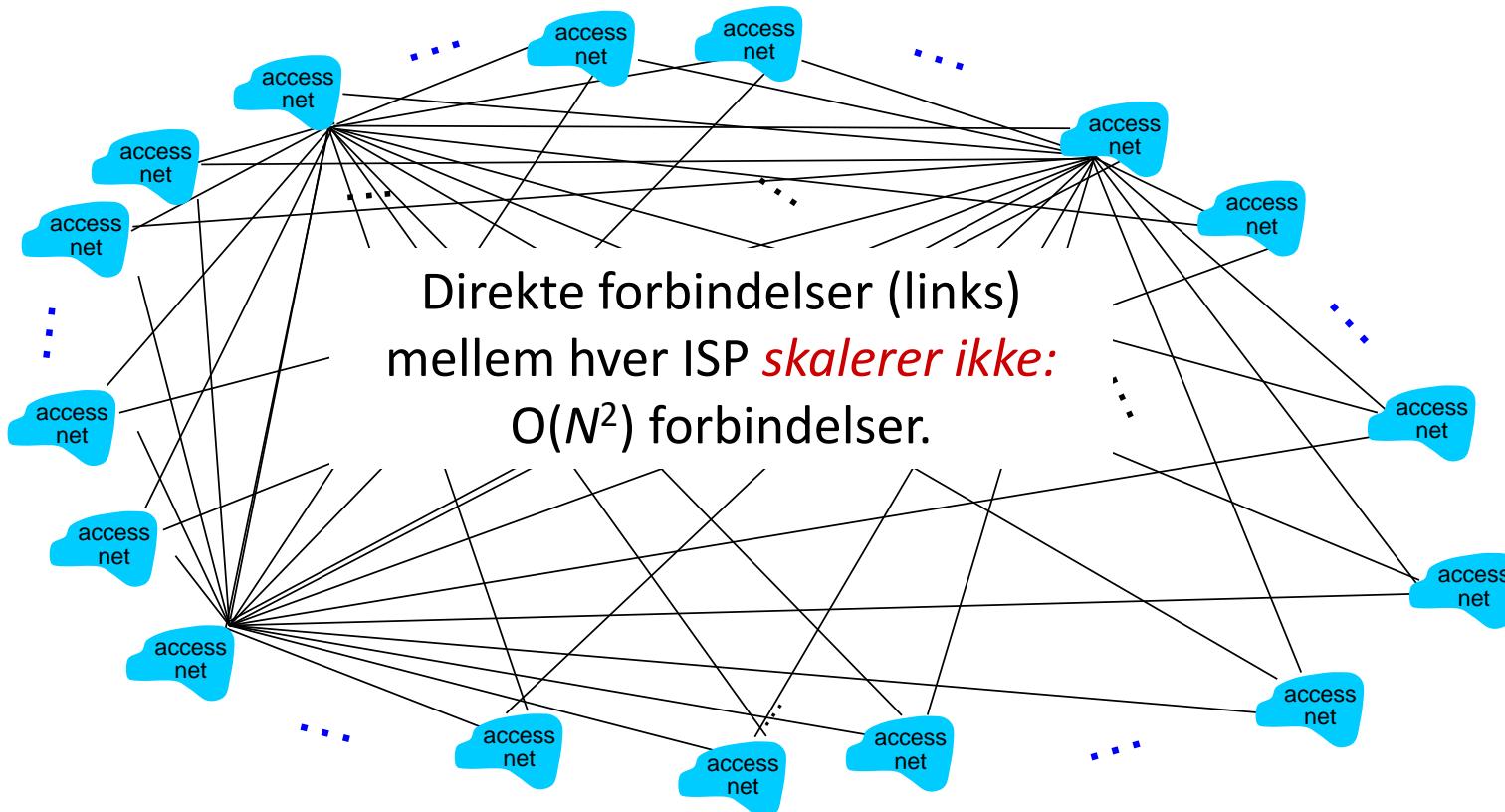
# Internet Struktur: Netværk af Netværk

*Spørgsmål:* Hvordan forbides *millioner* af ISP adgangs-netværk?



# Internet Struktur: Netværk af Netværk

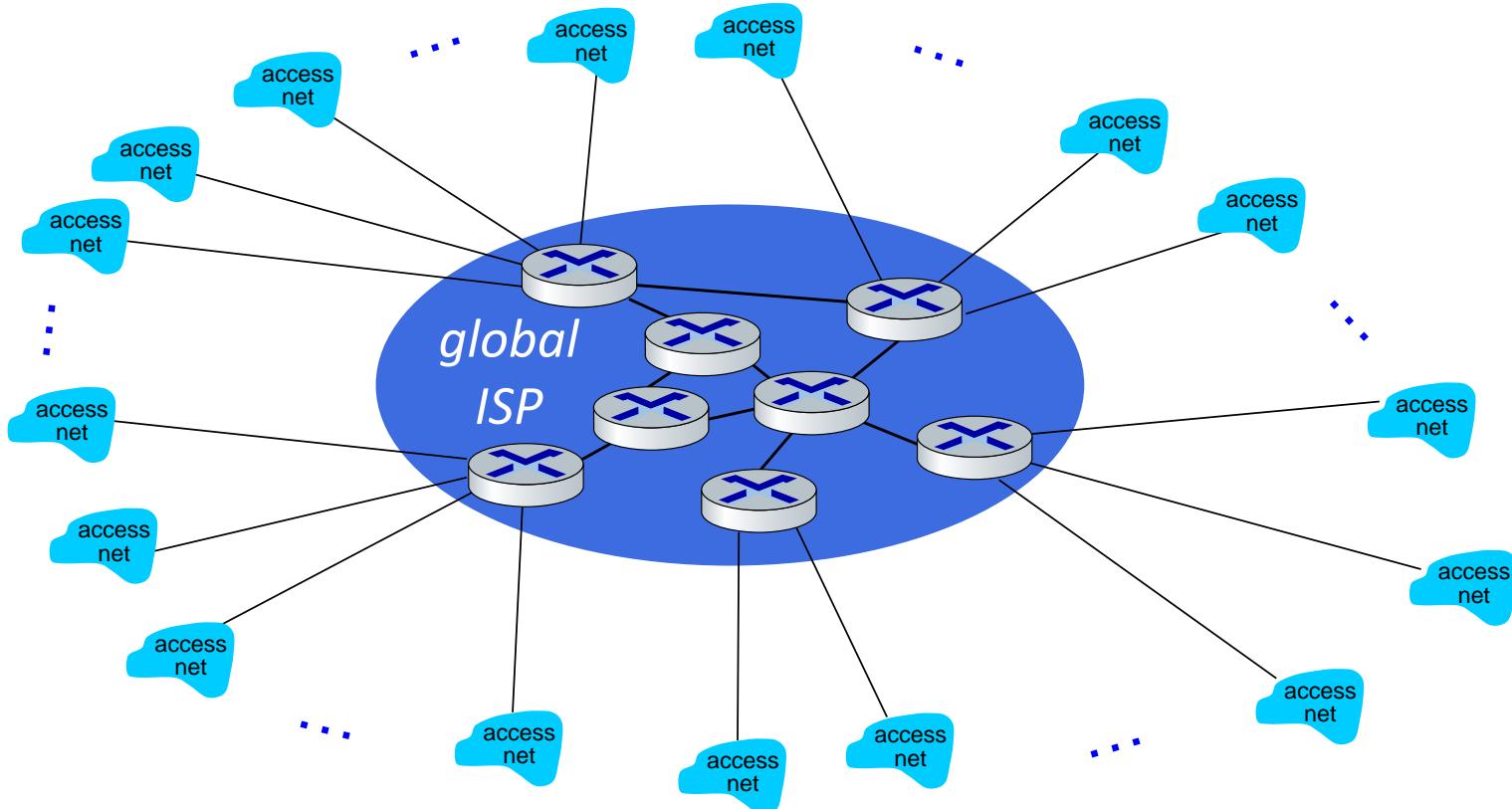
*Spørgsmål:* Hvordan forbides *millioner* af ISP adgangs-netværk?



# Internet Struktur: Netværk af Netværk

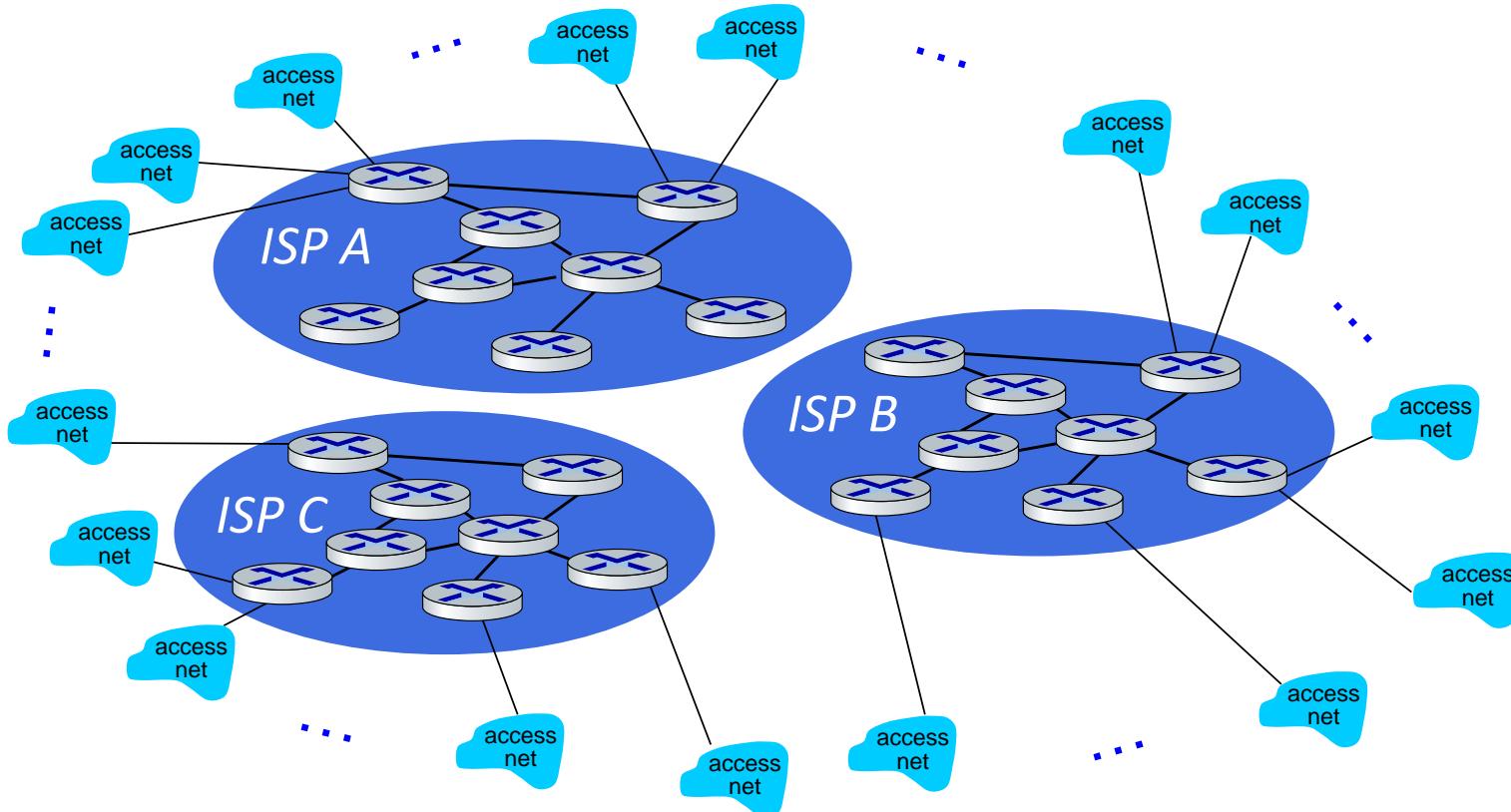
*En mulighed: forbind hver adgangs ISP til en global transit ISP?*

*Forretningsaftale imellem kunde- og leverandør ISPs*



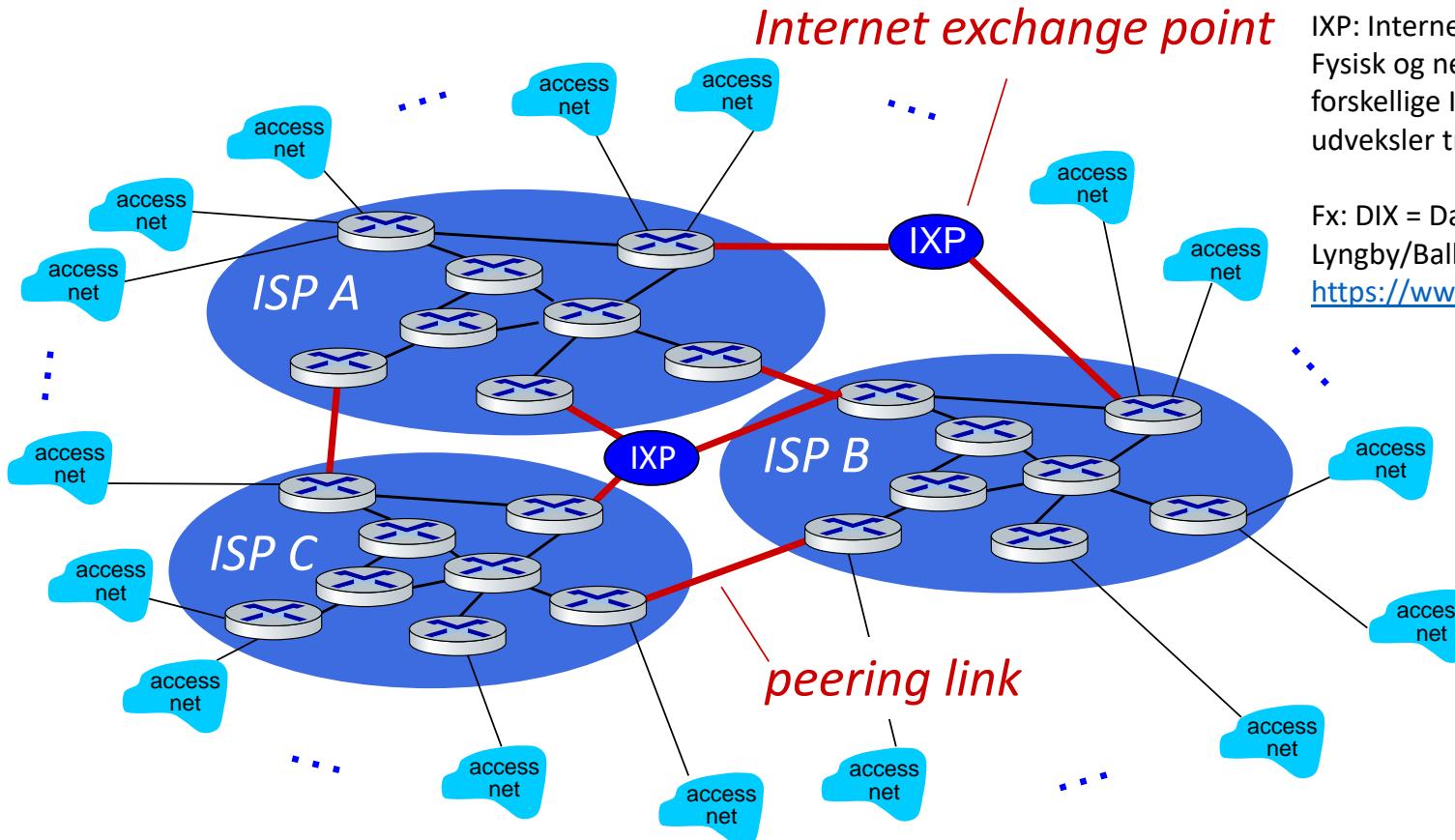
# Internet Struktur: Netværk af Netværk

Bedre med flere transit ISP'ere: Konkurrence, decentralisering, skalering, ...



# Internet Struktur: Netværk af Netværk"

Nogle Transit ISPere ønsker at forbinde med hinanden



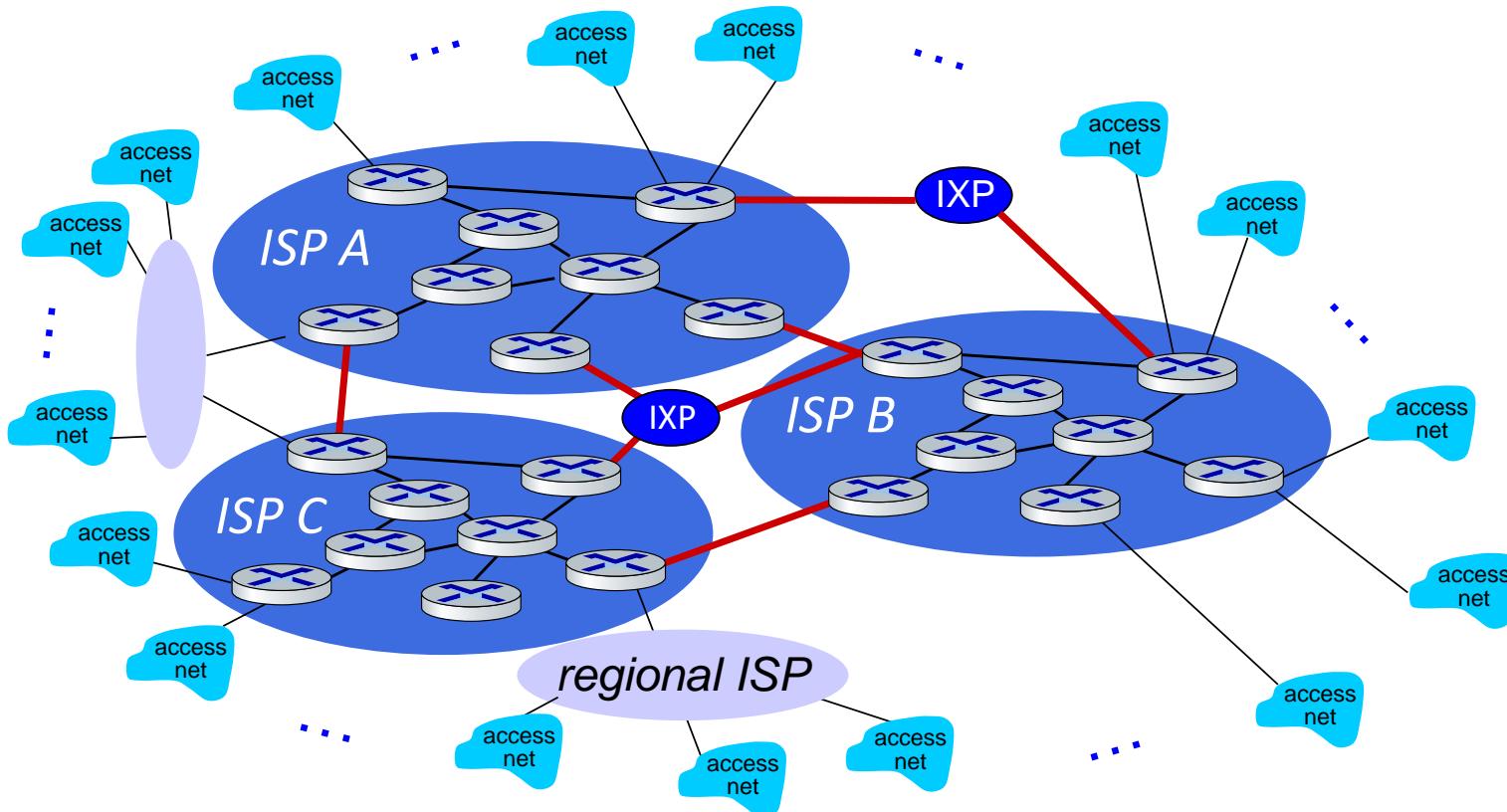
*Internet exchange point*

IXP: Internet Exchange Point (IXP) I  
Fysisk og neutral lokation, hvor  
forskellige ISP netværk mødes og  
udveksler traffik.

Fx: DIX = Danish Internet Exchange Point  
Lyngby/Ballerup/Skanderborg  
<https://www.dix.dk/>

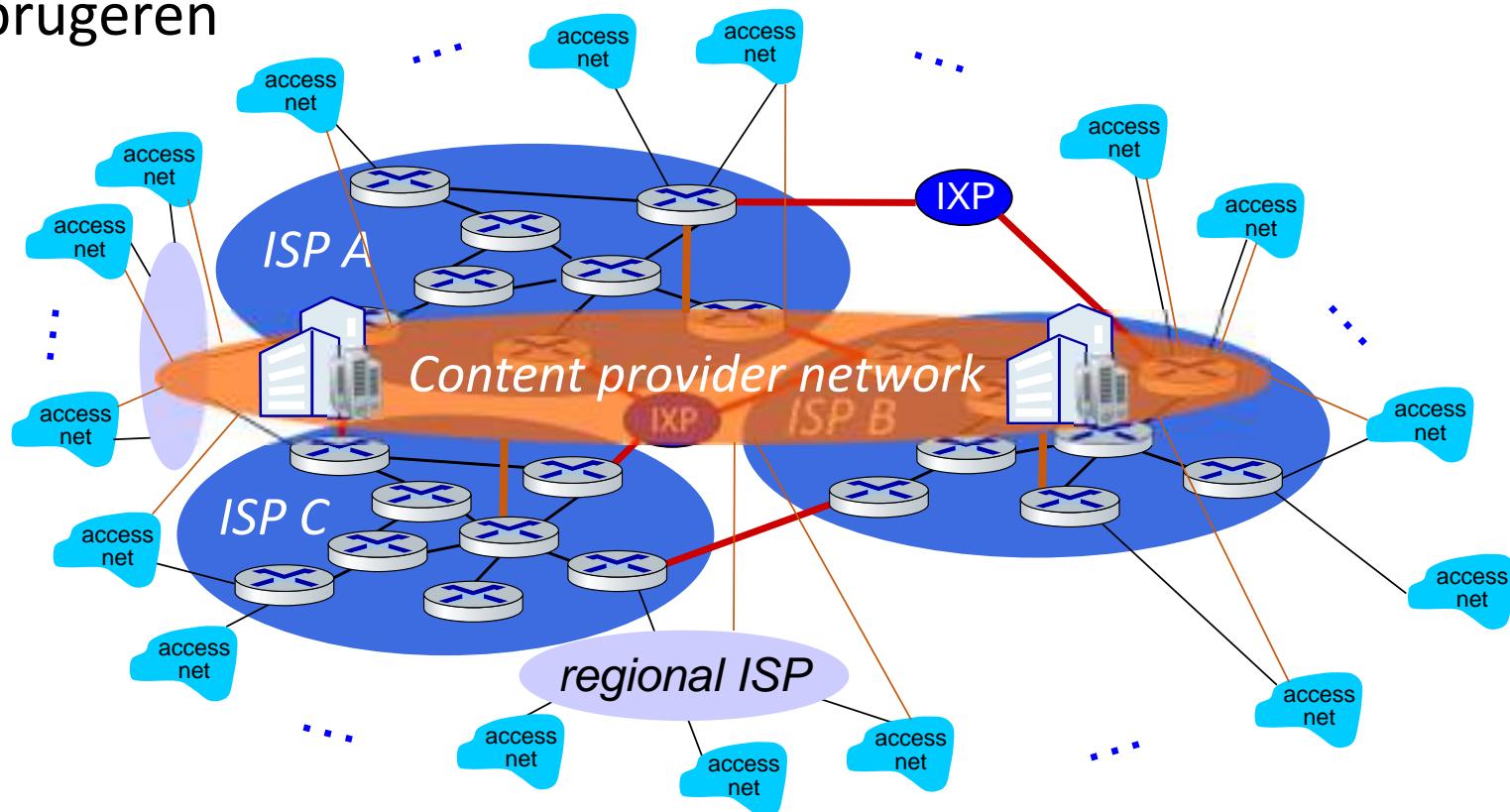
# Internet Struktur: Netværk af Netværk

... der kan opstå regionale netværk, der ønskes forbundet til det globale netværk

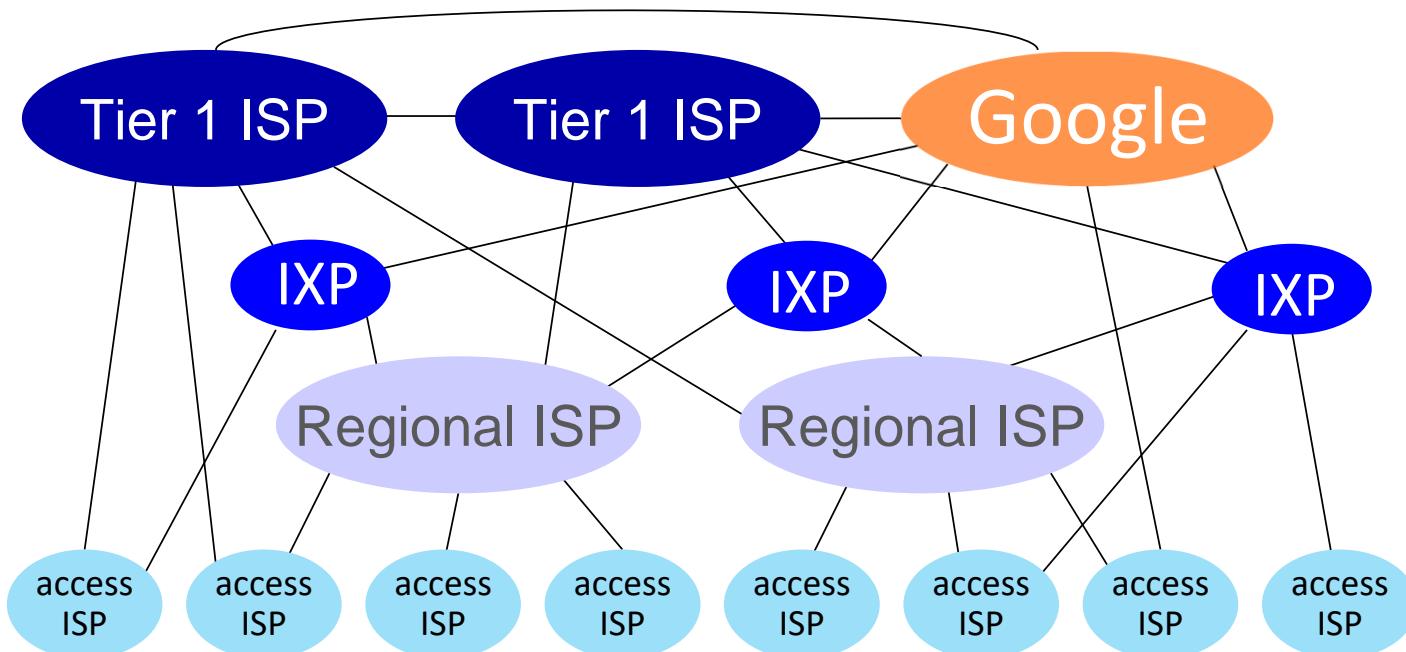


# Internet Struktur: Netværk af Netværk

... og større indholds-leverandører (e.g., Google, Microsoft, Akamai) driver deres egne netværk for at levere services og indhold tættere på slut-brugeren



# Internet Struktur: Netværk af Netværk

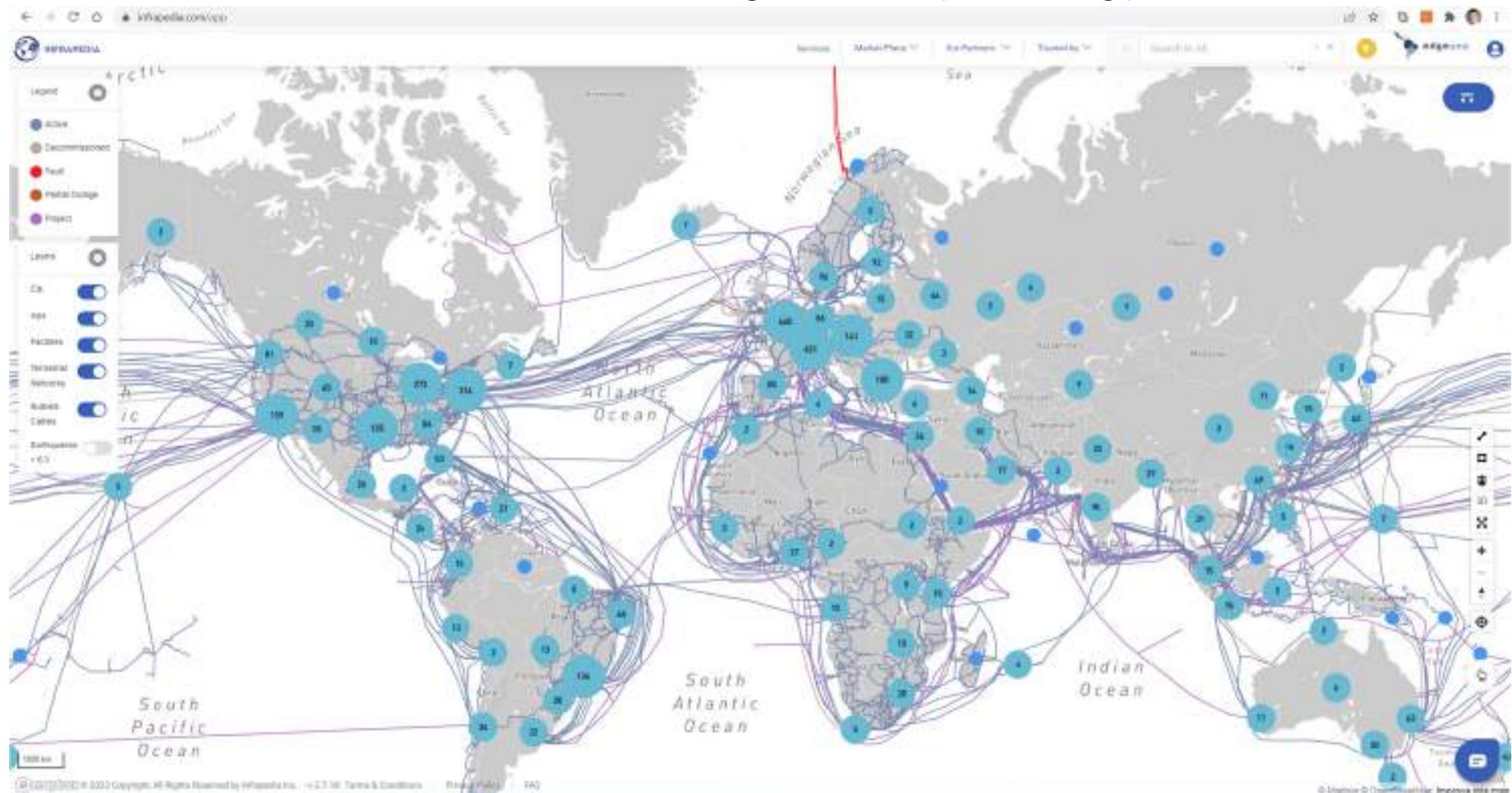


En kompleks struktur af *inter-forbundne netværk* blandt

- ISPs, Telekom udbydere, Indholdsudbyderes netværk (fx., Google, Microsoft, Akamai)
- I kernen: en mindre antal velforbundne store netværk, drevet af Større tele- og data-kommunikationsselskaber
- Data centre koncentrerer mange servers ("the cloud"), og indholdsleverandører koncentrerer meget trafik, ofte i egne netværk udenom lag-1 or regionale ISPs

# Infrapedia

Crowd-sourced information om netværks-forbindelser og infrastruktur (ufuldstændigt)





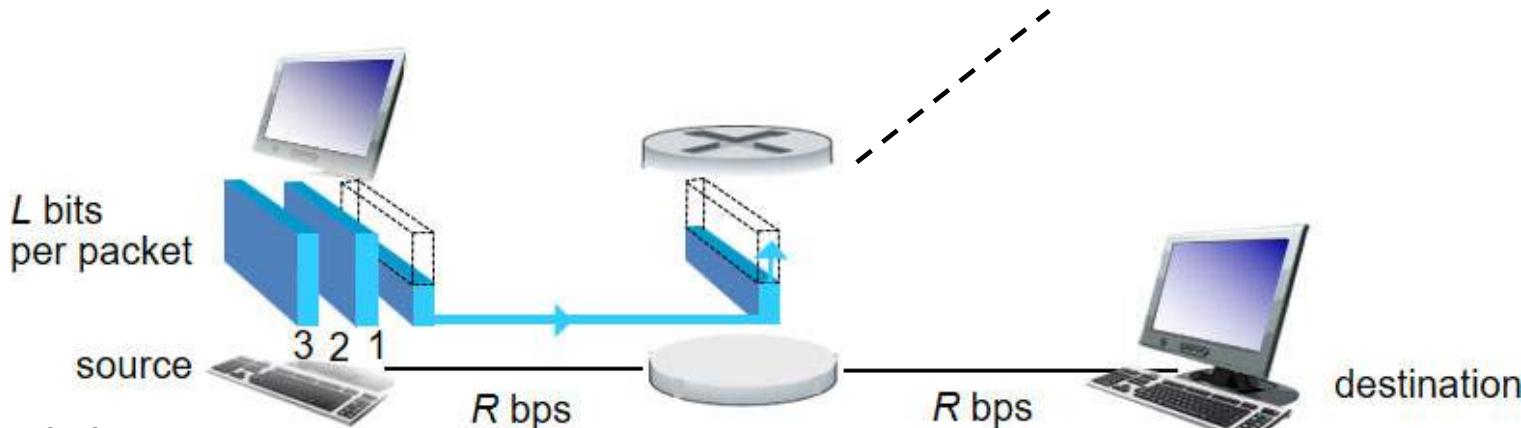
# Packet-switching

Hvad er packet switching?

Hvorfor og hvornår er det smart?

# Packet Switching

- **Pakke-kobling (packet-switching)**: sender host deler sine data op i mindre “data-pakker” \*)
  - Videresend pakker fra en router til den næste på stien af links fra source til destination
  - Hver pakke sendes med linkets (fulde) transmissionsrate
- **Gem-og-videresend (store and forward)**: hele pakken skal modtages af en router og gemmes i dennes hukommelse før den kan videresendes på den næste link

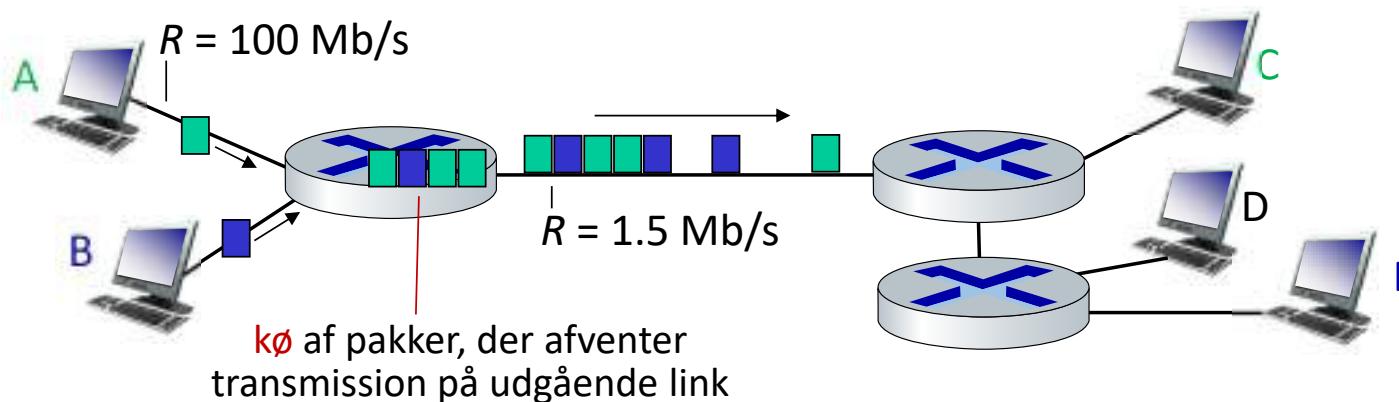


- **Forsinkelser**
  - En pakke der er  $L$ -bits lang, sendt med raten  $R$  bps, tager  $L/R$  sekunder at sende
  - Ex. Sende-forsinkelse:  $1000 \text{ bits}/1 \text{ Mbps} = 1\text{ms}$
  - Ex. 2 hop til destination: Samlet forsinkelse:  $2L/R$  (plus noget mere...)

\*) Fx, På Ethernet er maksimal pakkestørrelse  $L = 1522 \text{ bytes} = 12176 \text{ bits}$

# Packet Switching: Pakke-køer og pakke-tab

- Hvis ankomst raten til et link overstiger dets transmission rate over en kort periode:
  - Pakker bliver sat i **kø**, og afventer transmission på link

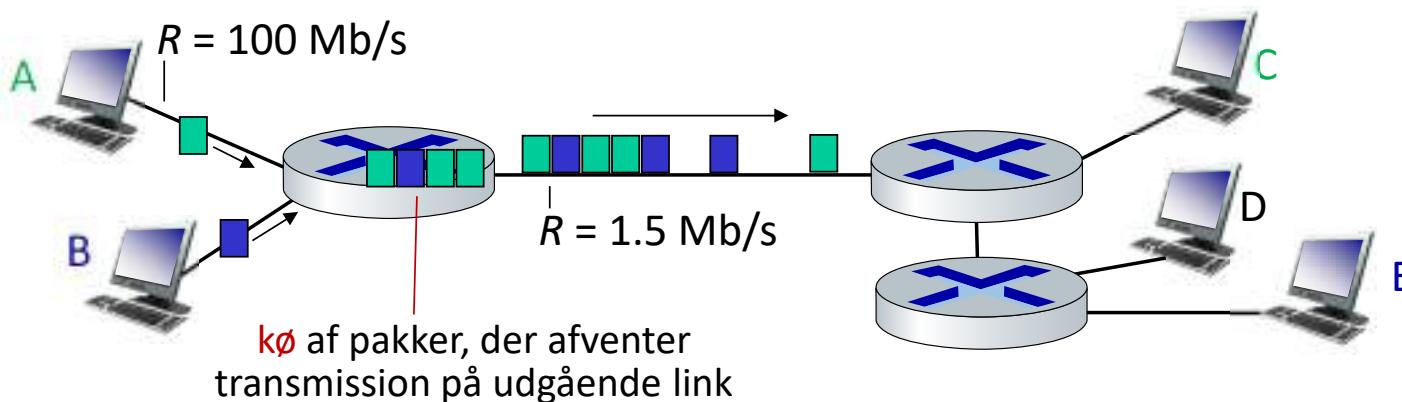


Coronatest kø?!

Der opstår køer når arbejde ankommer hurtigere end det kan behandles

# Packet Switching: Pakke-køer og pakke-tab

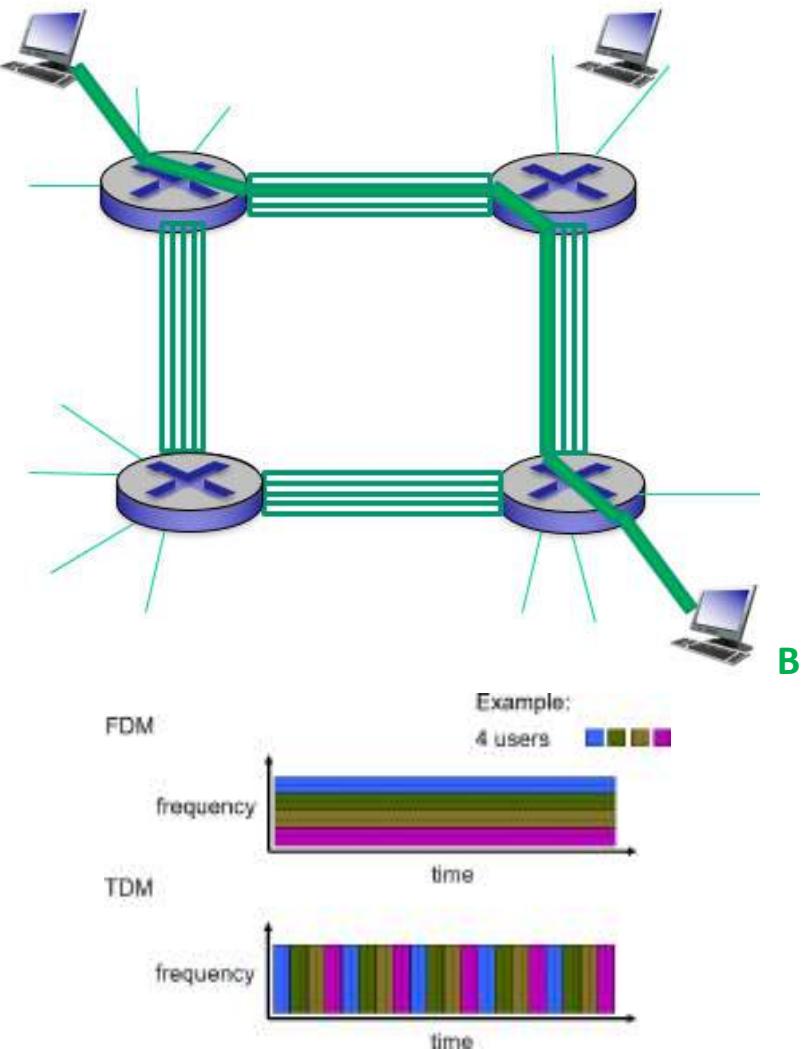
- Hvis ankomst raten til et link overstiger dets transmission rate over en kort periode:
  - Pakker bliver sat i **kø**, og afventer transmission på link
  - Pakker smides ud (droppes/tabes) hvis bufferen løber tør for plads



- **Godt til data-trafik som kommer i salver (bursts):** ressource deling ("statistical multi-plexing")
  - Hvis A og B sender samtidigt må de deles om output linkets kapacitet (fx 1.5 Mbps)
  - Det er mindre sandsynligt at A og B begge sender med R på samme tid langvarigt
    - **Hvis A sender "nu" og B ikke gør så bliver A's trafik videresendt med fuld 1.5 Mbps hastighed**
- **Uden kontrol:** uhæmmet grad af "forstoppelse" (congestion)
- Alternativt princip : circuit switching

# Kredsløbskobling (circuit switching)

- Før data sendes, oprettes en sti (kredsløb) A mellem sender og modtager, som alt data sendes over
- Hvert kredsløb får forud reserveret en fast transmissionsrate, fx 100 kbps
  - Til rådighed hele tiden, uanset større eller mindre behov.
  - Dermed virker det som om der er en dedikeret "fast" forbindelse mellem sender og modtager
- Teknikker til opdeling af links kapacitet
  - Frequency-Division Multiplexing
  - Time-Division Multiplexing



# Pakke kobling vs. kredsløbskobling

## eksempel:

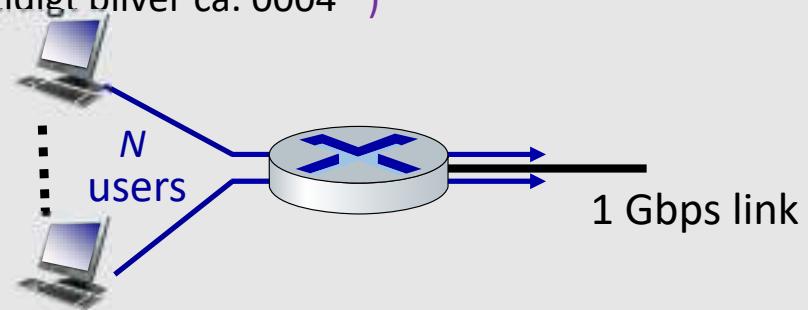
1 Gbps link, hvor hver bruger anvender 100 Mbps, men er kun “aktiv” 10% af tiden

### circuit-switching

- tillader i alt: 10 brugere
- **Dårlige ressource udnyttelse**
- **Godt til kritisk data, der helst ikke skal tabes**
- **Godt til tids-følsomt data**

### packet switching:

- $P(\text{en given bruger sender})=0.1$
- Ud af fx 35 (uafhængige) brugere, bliver sandsynligheden for at mere end 10 er aktivt samtidigt bliver ca. 0004 \*)



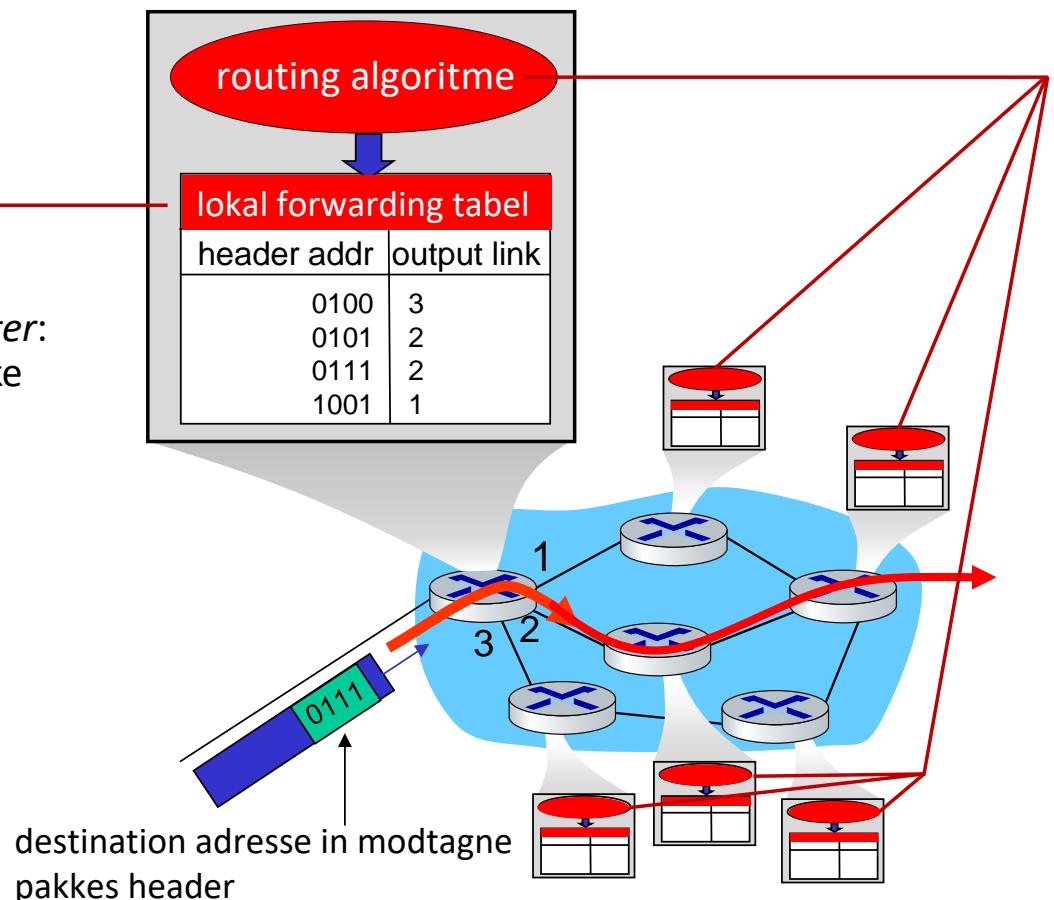
- **God ressource udnyttelse ved “bursty” trafik (kommer i “stød/salver”).**
- **Simplere, ingen opsætning af kredsløbet**
- **Forstoppelse (congestion) er muligt, giver pakketab**
  - => behov for mere avancerede protokoller til pålidelig data-overførsel og forhindring af forstoppelse

\*) Statistik med binomial fordelingen! SLIAL?

# To nøgle funktioner i en router

## Forwarding:

- aka "switching"
- *lokal handling i en router*: flyt ankommende pakke fra input link til rette output link



## Routing:

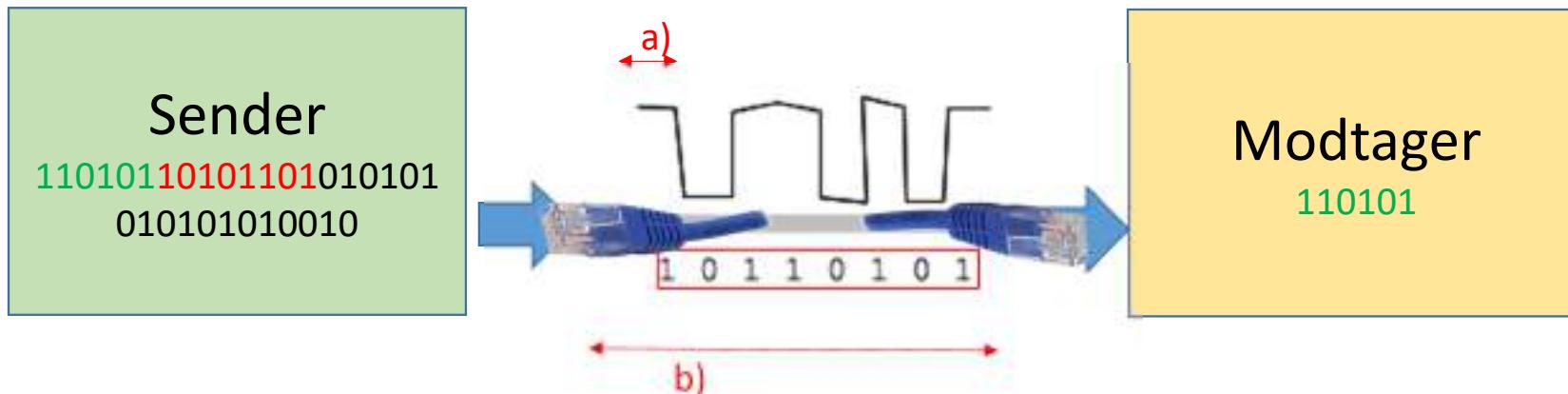
- *global* handling: beregn source-destination sti, som pakker skal følge
- routing algoritmer

# Forsinkelser, Throughput, Tab

Hvordan bestemmer man forsinkelse gennem netværket?

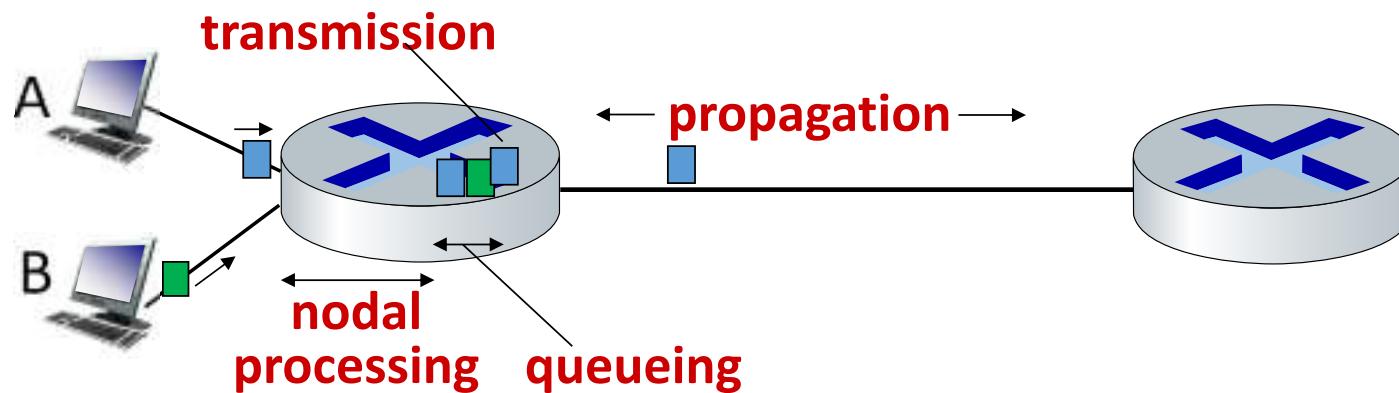
Hvordan bestemmes den mulige overførselshastighed?

# Udbredelse af et digital signal på et medie



- Det tager noget tid at sende (transmittere) en bit på mediet (a)
  - (for-)simplet eksempel
    - Et logisk 1 kodes som 3 volt i  $1 \mu\text{s}$
    - Et logisk 0 kodes om 0 volt i  $1 \mu\text{s}$
    - Giver transmissions hastighed på 1 mega bits per sekund
- Der er en signal-udbredelsestid mellem sender og modtager (b)
  - Afhænger af mediet og afstanden mellem sender og modtager
  - Knapt lysets hastighed (ca.  $2 \cdot 10^8 \text{ m/s}$ )
  - Elektrisk signal i kobber: ca. 70% af lysets hastighed

# Kilder til forsinkelse i en router



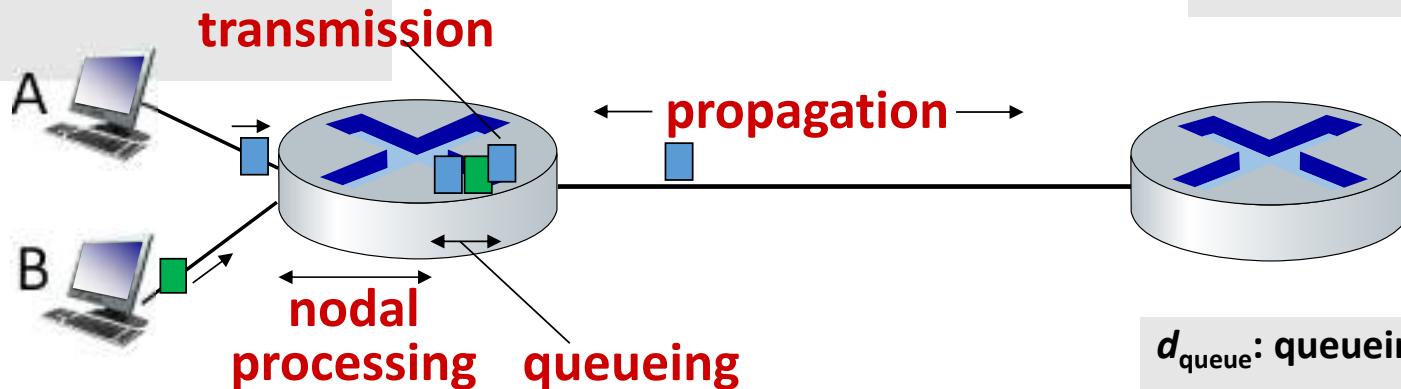
Tid fra en pakke netop er modtaget i én knude, til den netop er modtaget i næste

$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

# Kilder til forsinkelse i en router

## transmissions delay:

- Det tager tid at "kode en bit op" på mediet
- $L$ : længden af pakken (bits)
- $R$ : link transmissions rate (bps)
- $d_{trans} = \frac{L}{R}$



## $d_{proc}$ : nodal processering

- Knudepunktets data-behandlingstid
- check bit fejl
- bestemme output link
- Indsæt i kø
- typisk < msec

## Propagerings (udbredelse) delay:

- $d$ : distance (længde af fysiske link)
- $s$ : udbredelses hastighed (knappt lysets hastighed (ca  $2 \cdot 10^8$  m/s))
- $d_{prop} = \frac{d}{s}$ , (kun betydende når d stor)

## $d_{queue}$ : queueing delay

- Ventetid i kø, før pakken kan sendes på udgående link
- Afhænger af hvor antallet af ophobede pakker (congestion level)

$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

# Trafik Intensitet

Forholdet imellem indkommende og udgående trafik rate

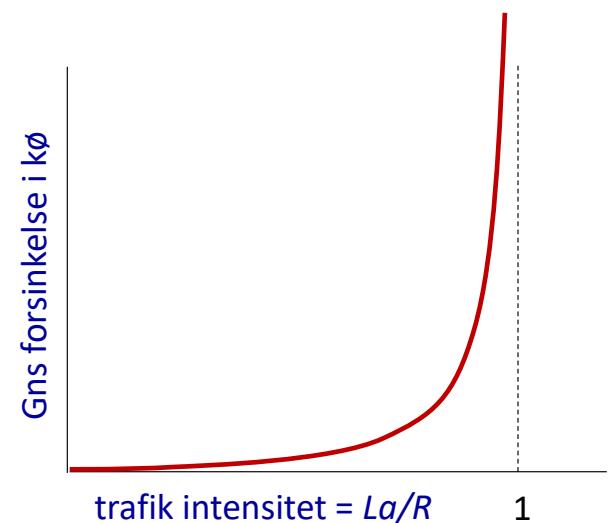
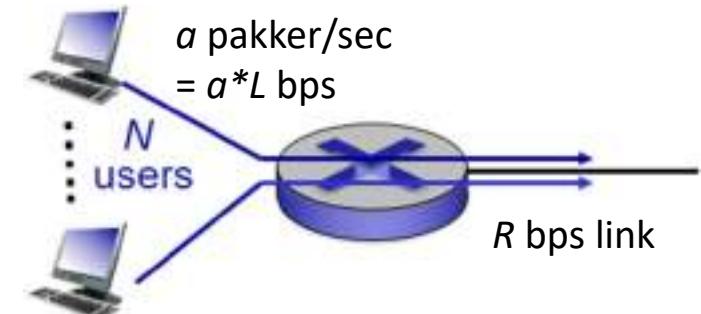
- $L$ : pakke længde (bits)
- $a$ : gennemsnitlig pakke ankomst rate (pakker er sec.)
- $L \times a$  = ankomst rate (bps)
- $R$ : link transmissions rate (bps)
- Trafik intensitet:  $\frac{L \times a}{R}$



- Hvis  $\frac{La}{R}$  overstiger 1: fortsat akkumulering af ventende pakker: Konsekvens=Pakketab!
- Hvis  $\frac{La}{R}$  er lille  $\sim 0$ : klar bane
- Hvis  $\frac{La}{R}$  nærmer sig 1: lang ventetid

(eksakt forsinkelse afhænger af klumpers fordeler sig i trafikken)

- Systemet skal designes så intensiteten bliver moderat



# Rigtige forsinkelser og router på Internettet

traceroute: gaia.cs.umass.edu to [www.eurecom.fr](http://www.eurecom.fr)

Udsender kontrol pakker for at måle en pakkes rute og forsinkelse igennem nettet:

| 3 målinger af forsinkelse fra gaia.cs.umass.edu til cs-gw.cs.umass.edu |                                                 |        |        |        |  |                                                               |
|------------------------------------------------------------------------|-------------------------------------------------|--------|--------|--------|--|---------------------------------------------------------------|
| 1                                                                      | cs-gw (128.119.240.254)                         | 1 ms   | 1 ms   | 2 ms   |  |                                                               |
| 2                                                                      | border1-rt-fa5-1-0.gw.umass.edu (128.119.3.145) | 1 ms   | 1 ms   | 2 ms   |  | 3 målinger af forsinkelse til border1-rt-fa5-1-0.gw.umass.edu |
| 3                                                                      | cht-vbns.gw.umass.edu (128.119.3.130)           | 6 ms   | 5 ms   | 5 ms   |  |                                                               |
| 4                                                                      | jin1-at1-0-0-19.wor.vbns.net (204.147.132.129)  | 16 ms  | 11 ms  | 13 ms  |  |                                                               |
| 5                                                                      | jin1-so7-0-0-0.wae.vbns.net (204.147.136.136)   | 21 ms  | 18 ms  | 18 ms  |  |                                                               |
| 6                                                                      | abilene-vbns.abilene.ucaid.edu (198.32.11.9)    | 22 ms  | 18 ms  | 22 ms  |  |                                                               |
| 7                                                                      | nycm-wash.abilene.ucaid.edu (198.32.8.46)       | 22 ms  | 22 ms  | 22 ms  |  | trans-atlantisk link                                          |
| 8                                                                      | 62.40.103.253 (62.40.103.253)                   | 104 ms | 109 ms | 106 ms |  |                                                               |
| 9                                                                      | de2-1.de1.de.geant.net (62.40.96.129)           | 109 ms | 102 ms | 104 ms |  |                                                               |
| 10                                                                     | de.fr1.fr.geant.net (62.40.96.50)               | 113 ms | 121 ms | 114 ms |  |                                                               |
| 11                                                                     | renater-gw.fr1.fr.geant.net (62.40.103.54)      | 112 ms | 114 ms | 112 ms |  | looks like delays decrease! Why?                              |
| 12                                                                     | nio-n2.cssi.renater.fr (193.51.206.13)          | 111 ms | 114 ms | 116 ms |  |                                                               |
| 13                                                                     | nice.cssi.renater.fr (195.220.98.102)           | 123 ms | 125 ms | 124 ms |  |                                                               |
| 14                                                                     | r3t2-nice.cssi.renater.fr (195.220.98.110)      | 126 ms | 126 ms | 124 ms |  |                                                               |
| 15                                                                     | eurecom-valbonne.r3t2.ft.net (193.48.50.54)     | 135 ms | 128 ms | 133 ms |  |                                                               |
| 16                                                                     | 194.214.211.25 (194.214.211.25)                 | 126 ms | 128 ms | 126 ms |  |                                                               |
| 17                                                                     | * * *                                           |        |        |        |  |                                                               |
| 18                                                                     | * * *                                           |        |        |        |  | * Angiver "intet svar" (probe-pakke tabt, router svarer ikke) |
| 19                                                                     | fantasia.eurecom.fr (193.55.113.142)            | 132 ms | 128 ms | 136 ms |  |                                                               |

\* Forsøg at lave traceroute til nogle eksotiske lande på [www.traceroute.org](http://www.traceroute.org)

# Demo (“tracert” program)

```
C:\ Kommandoprompt
-w timeout Wait timeout milliseconds for each reply.
-R Trace round-trip path (IPv6-only).
-S srcaddr Source address to use (IPv6-only).
-4 Force using IPv4.
-6 Force using IPv6.

C:\Users\bniel>tracert www.cs.aau.dk

Tracing route to www.cs.aau.dk [130.225.63.3]
over a maximum of 30 hops:

 1 2 ms 2 ms 2 ms 192.168.0.1
 2 3 ms 4 ms 3 ms 85.203.152.129
 3 4 ms 3 ms 3 ms stovr01ds01-ae24-0.eniig-net.dk [85.191.209.76]
 4 4 ms 4 ms 5 ms aarsx01cr01_ae11.em-net.dk [85.191.209.38]
 5 5 ms 5 ms 5 ms 10.10.1.53
 6 7 ms 6 ms 5 ms 10.10.0.1
 7 6 ms 6 ms 6 ms 87.116.38.121
 8 21 ms 8 ms 8 ms 93.176.93.8
 9 10 ms 11 ms 21 ms dk-uni.nordu.net [192.38.7.50]
10 12 ms 10 ms 10 ms lgb.core.fsknet.dk [109.105.102.159]
11 10 ms 11 ms 10 ms 100g-lgb.ore.core.fsknet.dk [130.225.245.154]
12 18 ms 18 ms 27 ms edge1.aau.dk [130.226.249.146]
13 17 ms 17 ms 17 ms Eth1-20.aau-core1.aau.dk [192.38.59.27]
14 18 ms 18 ms 17 ms Eth5-15.dc2-gw02.aau.dk [192.38.59.203]
15 28 ms 27 ms 18 ms vm-ig-www2.portal.aau.dk [130.225.63.3]

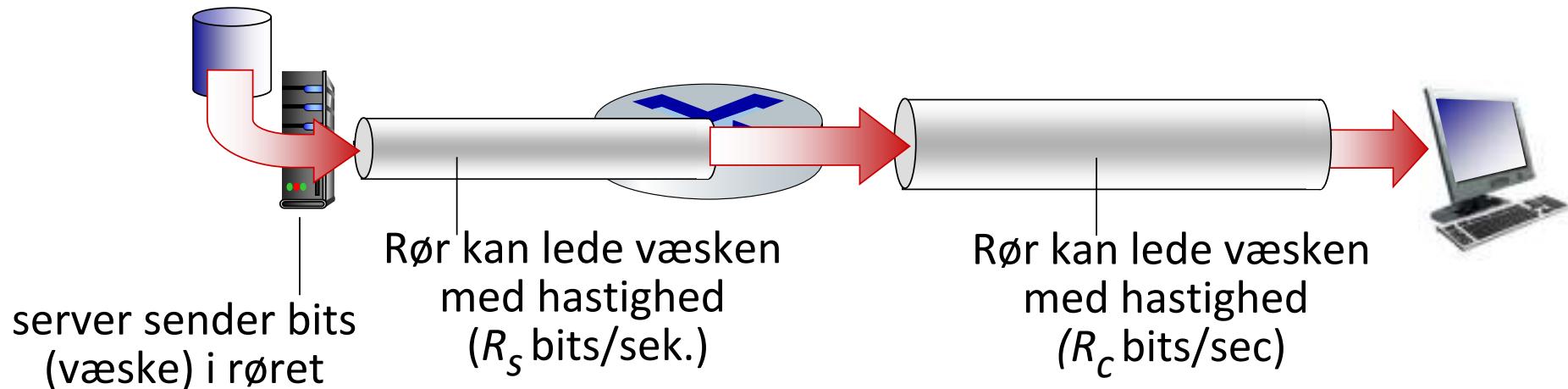
Trace complete.

C:\Users\bniel>
```

Hjemme fra.

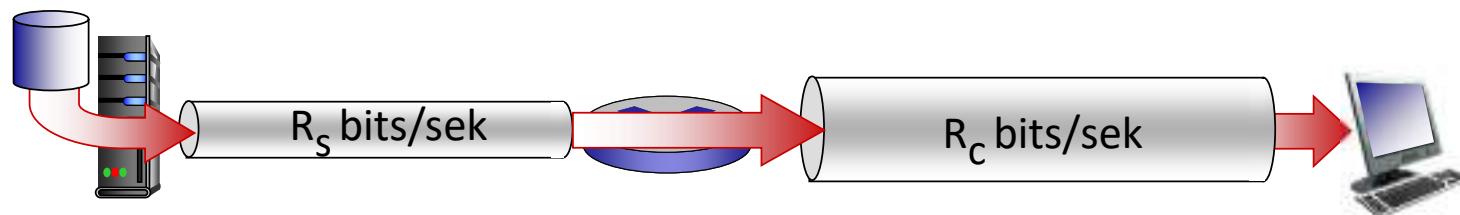
# Throughput

- **throughput:** (gennemløbsrate) hastighed (i bits/sekund) som kan overføres fra sender til modtager
  - **Øjeblikkelig:** hastighed på et givet tidspunkt
  - **Gennemsnitlig:** hastighed over en længere tidsperiode
- Rørlednings analogi

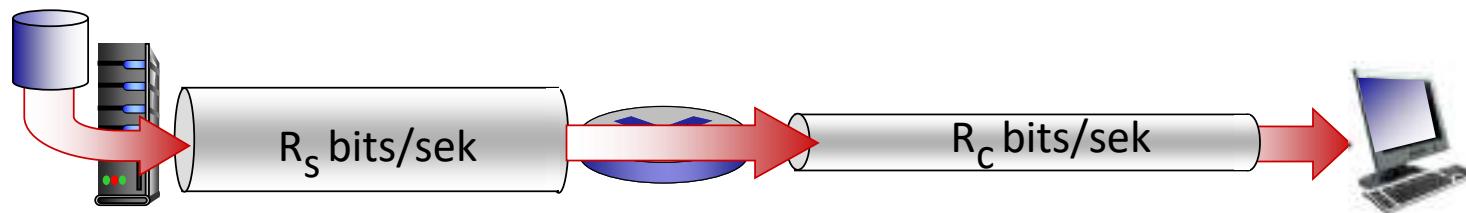


# Throughput

$R_s < R_c$  Hvad er de gennemsnitlige throughput mellem start og mål?



$R_s > R_c$  Hvad er de gennemsnitlige throughput mellem start og mål?

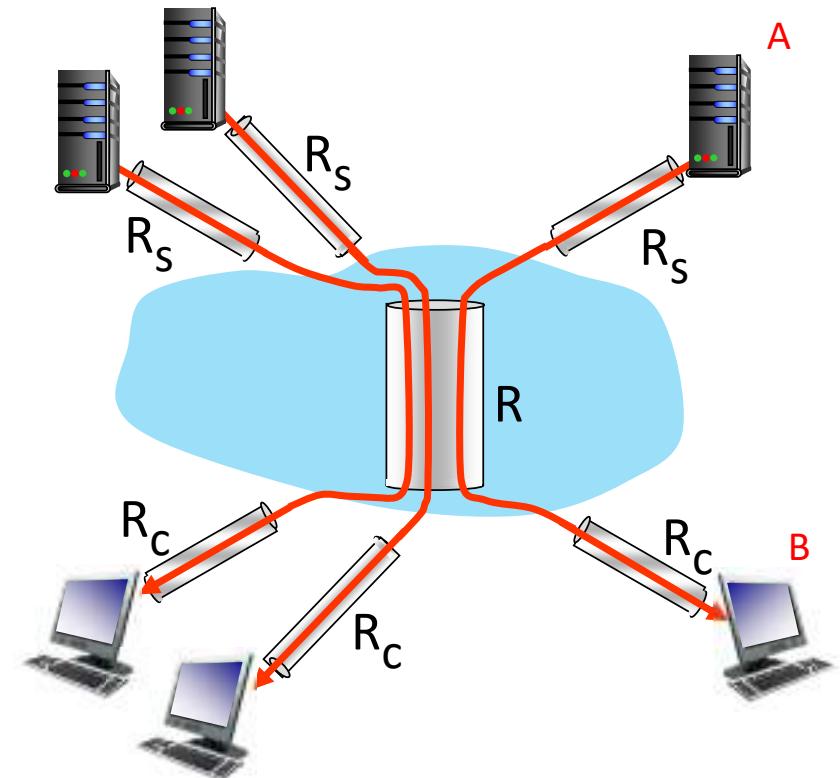


*flaskehals link*

Det link på "end-to-end" stien som begrænser "end-to-end" throughput

# Throughput: Internet Scenarie

- Bestemmes af link med mindst rate:
- Throughput  $\approx \min\left(R_c, R_s, \frac{R}{n}\right)$
- EX,
  - $R_c = 2 \text{ Mbps}$
  - $R_s = 5 \text{ Mbps}$
  - $R = 100 \text{ Mbps}$
  - $N = 10$  client server forbindelser
  - $\Rightarrow$  Throughput  $\approx 2 \text{ Mbps}$
- I praksis er  $R_c$  eller  $R_s$  ofte flaske-halsen



$N=10$  forbindelser mellem 10  
klient/server par, deler  $R$  bits/s fair

# TCP/IP protokol stakken

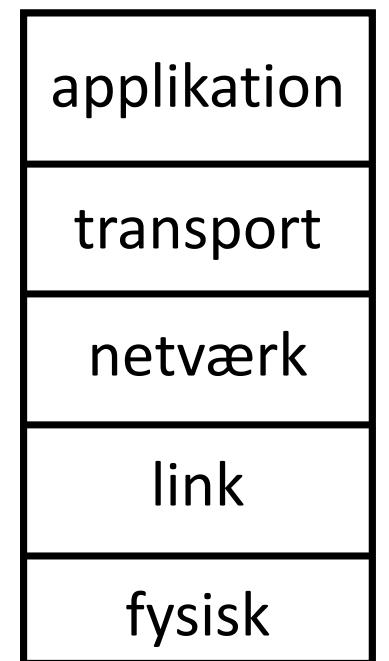
Hvad er en protokol stak?

Hvordan ser en model for lagdeling af internet protokollerne ud?

Hvordan behandles pakker i IP stakken?

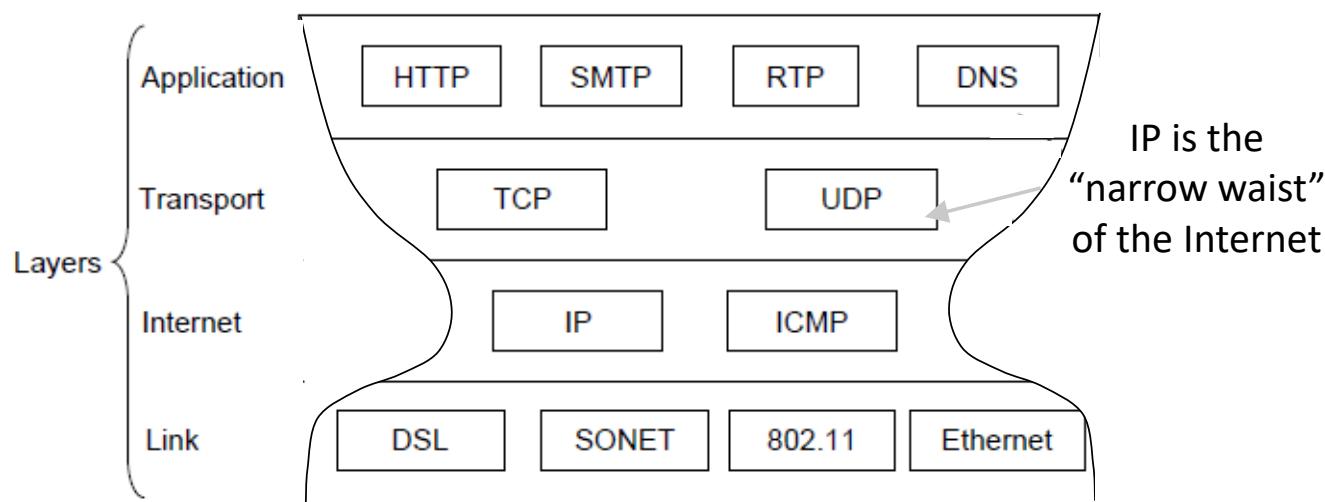
# TCP/IP Reference Model

- “**Protokol stak**” samling af relaterede protokoller, der er organiseret i lag
- **TCP/IP Reference Model:** En 5 lags model for Internet protokollerne , som er uddraget baseret på observationer af et konkret netværk
- **Applikations-lag:** Protokoller som understøtter afvikling af netværks-applikationer (mail, browser, fil-transport, ...)
  - FTP, SMTP, HTTP, DNS
- **Transport-lag:** Overfører data fra et kørende program (proces) på end-system A til modpart på end-system B
  - TCP, UDP
- **Netværks-lag:** routing og videresendelse af pakker fra source til destination
  - IP
- **Link-lag:** data overførsel mellem direkte nabo-knuder i netværket, dvs. koblet sammen med et enkelt link.
  - Ethernet, IEEE-802.11 (WiFi), PPP, SONET
- **Fysiske-lag:** bits ”på ledningen”



# TCP/IP Reference Model

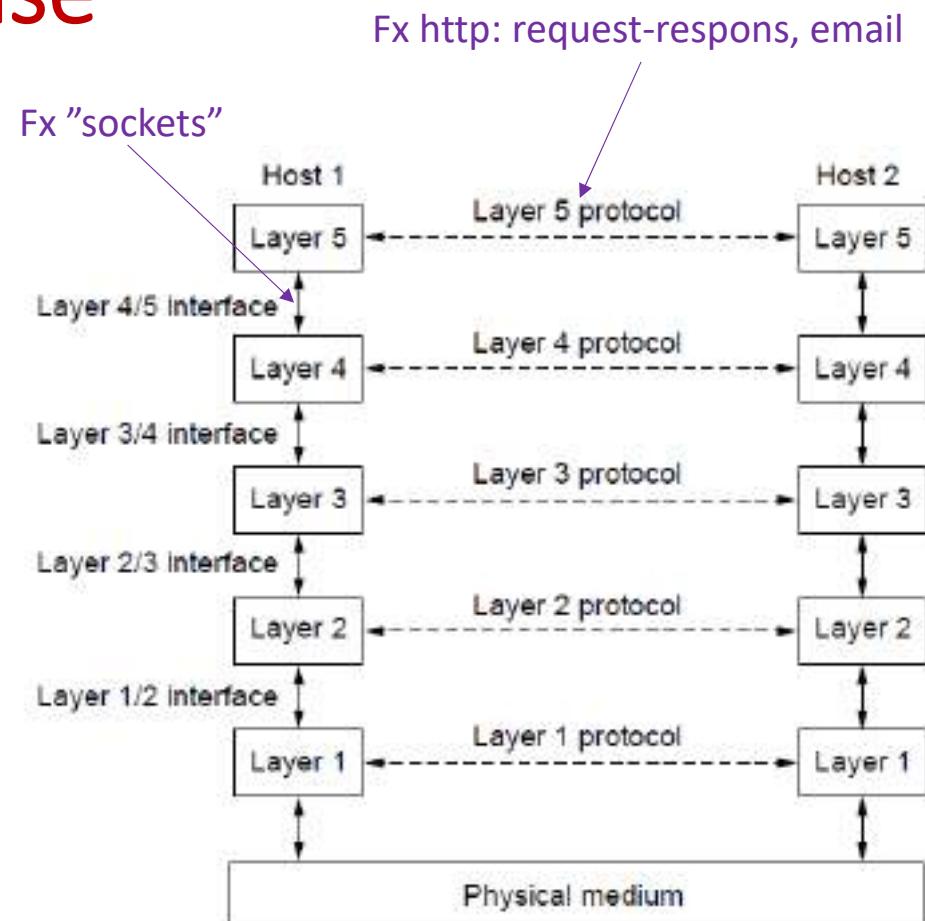
- Snæver-talje/timeglas-model: Hvis vi kan transportere IP på mediet X, virker alle transport og applikations-protokoller også.
- “Mindste fællesnævner”



Udvalgte Internet protokoller er vist på deres respektive lag

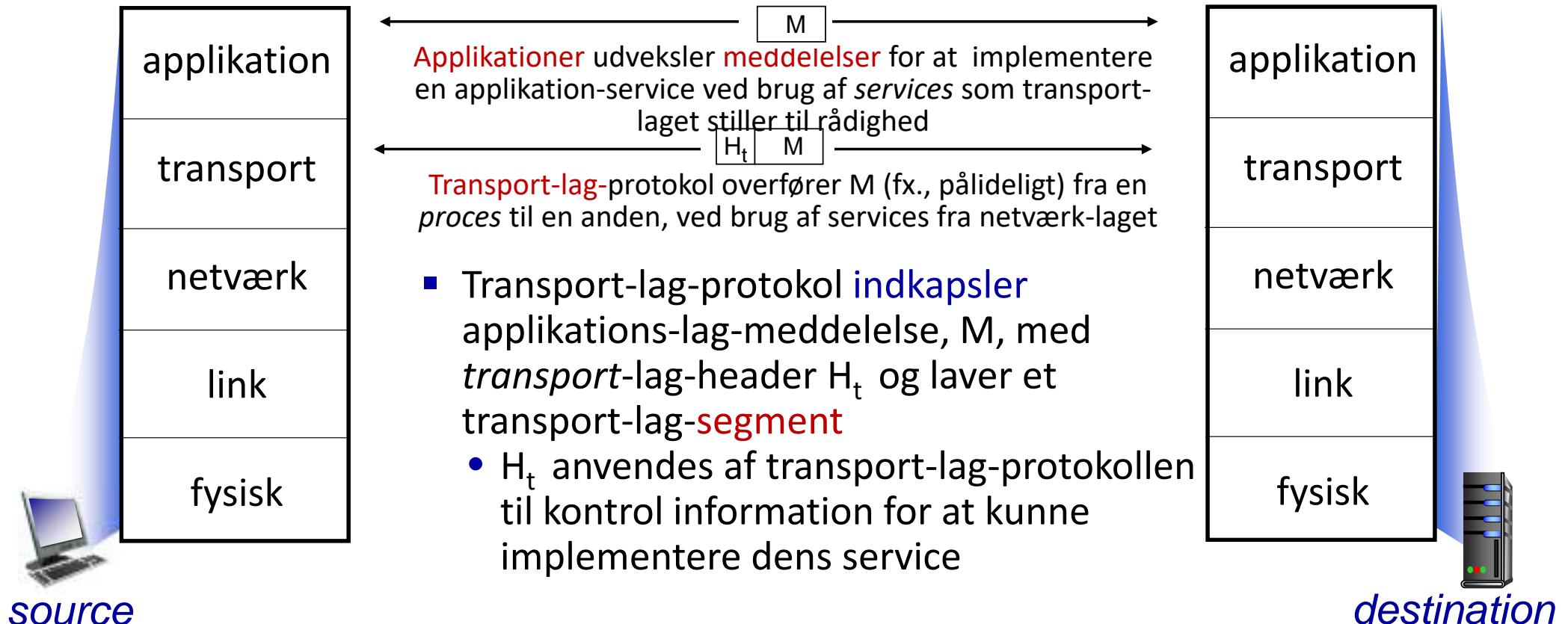
# Abstrakt service beskrivelse

- Hver protokol instans “snakker” **virtuelt** direkte med dens modpart (ligemand=peer)
- Hvert lag kommunikerer **kun** ved brug af laget under
- Services udbudt af et lavere lag kan tilgåes af overliggende lag via et interface
- I bunden overføres meddelelser på et fysisk medie

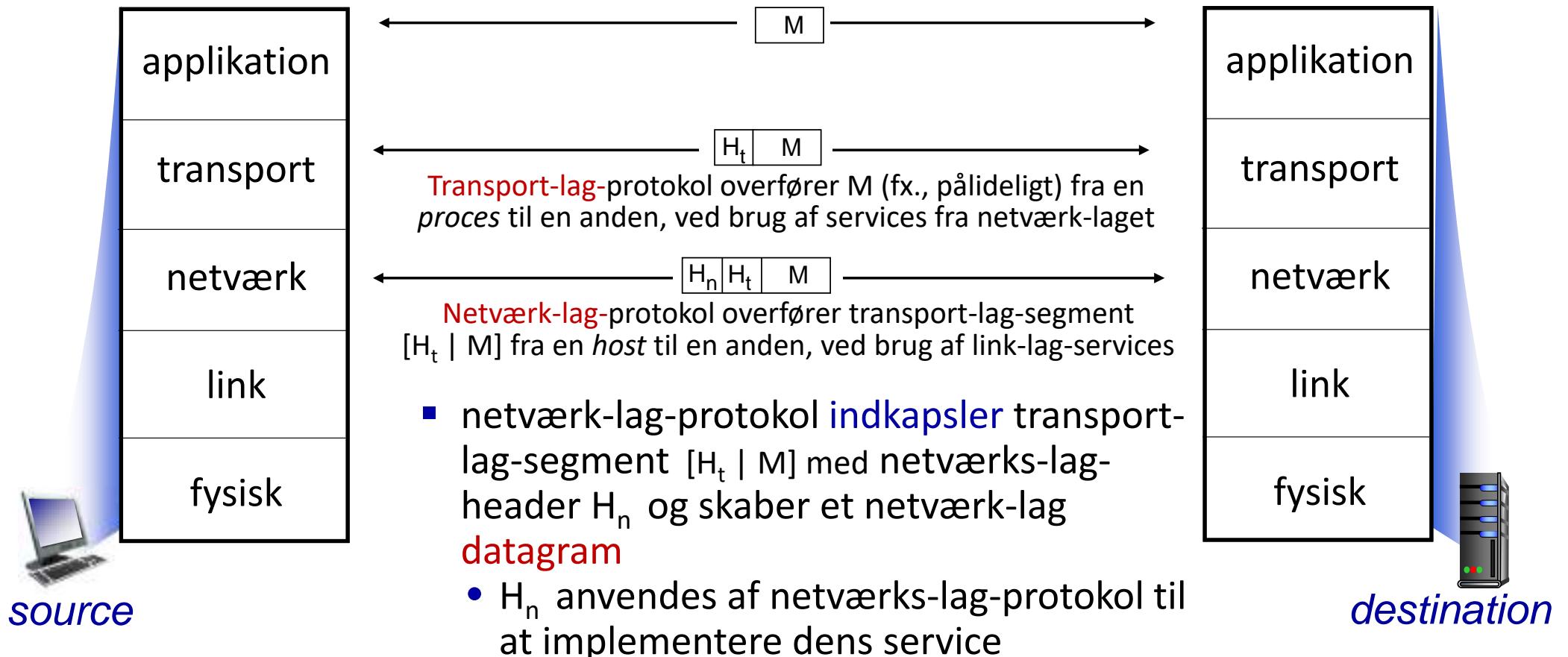


# Services, Lag-opdeling, og indkapsling

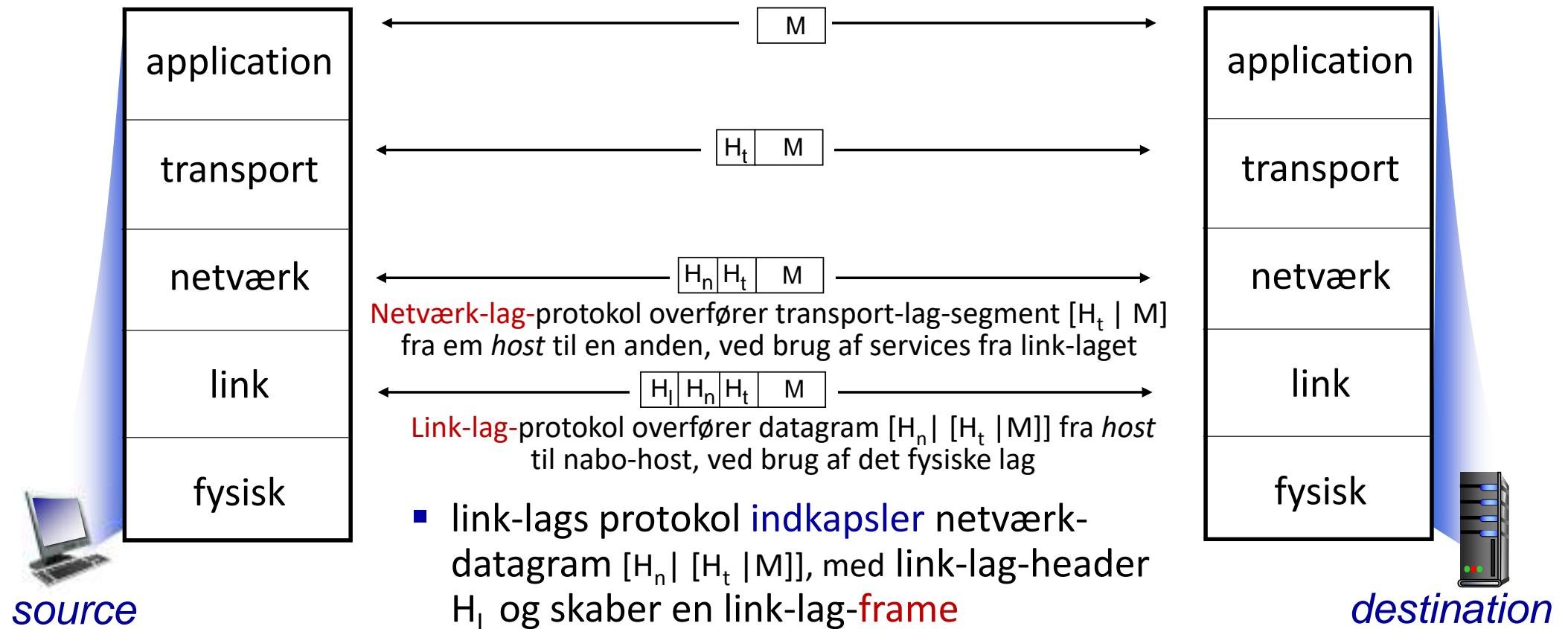
# Services, Lag-opdeling, og indkapsling



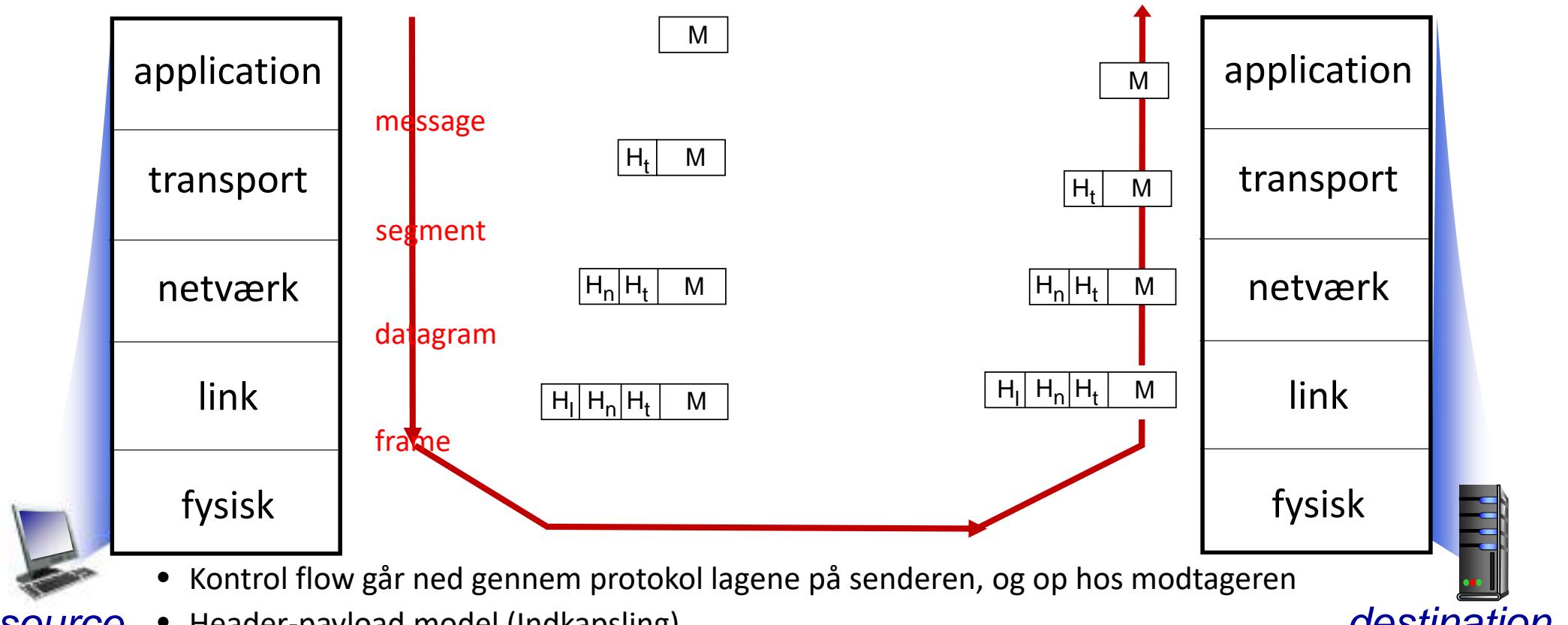
# Services, Lag-opdeling, og indkapsling



# Services, Lag-opdeling, og indkapsling

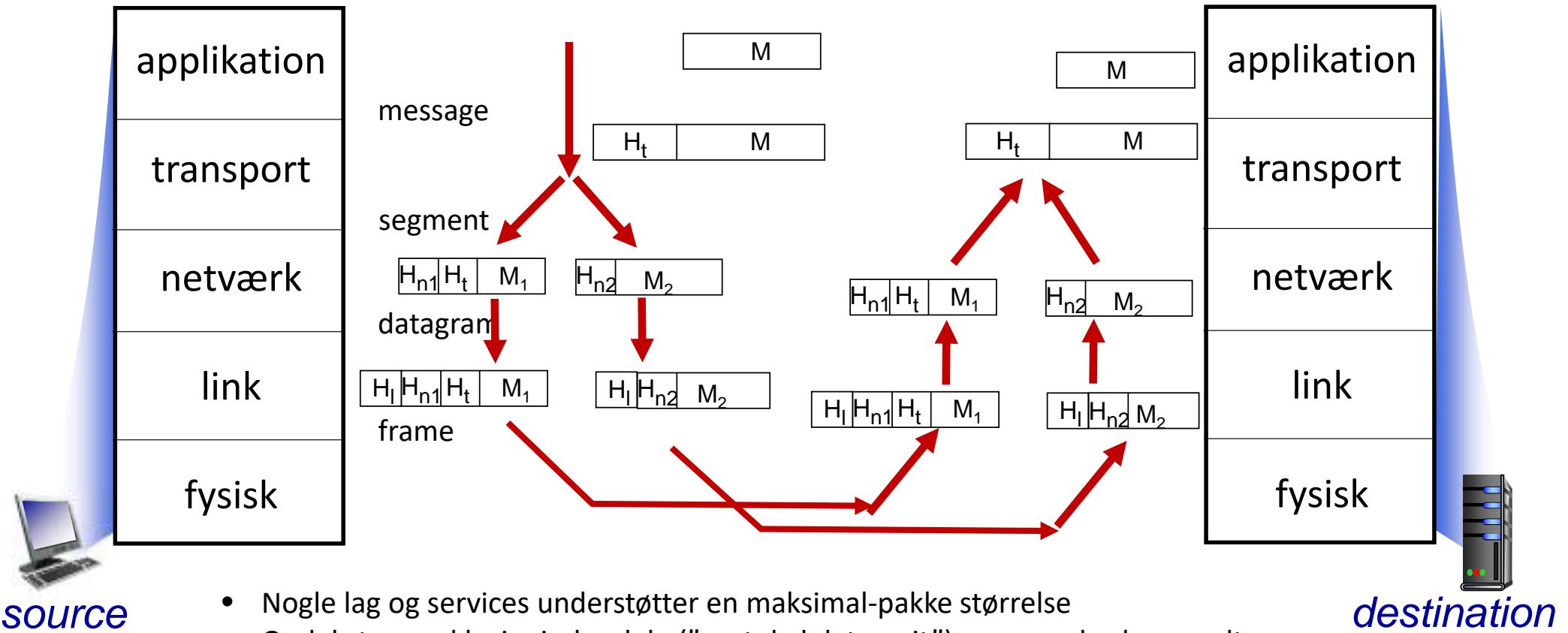


# Services, Lag-opdeling, og indkapsling



- Kontrol flow går ned gennem protokol lagene på senderen, og op hos modtageren
- Header-payload model (Indkapsling)
  - Hvert underliggende lag tilføjer sin egen **header** (med kontrol information) til indhold (**payload**) for at sende den, og fjerner den efter modtagelse
    - “kuvert” i “kuvert” princip
    - “Enveloping” / Encapsulation

# Segmentering og gen-samling



*source*

*destination*

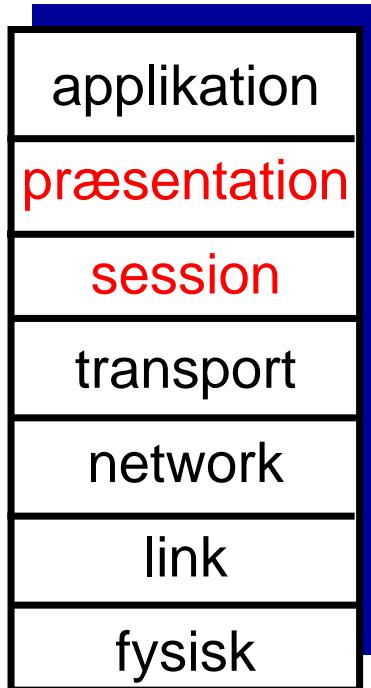
- Nogle lag og services understøtter en maksimal-pakke størrelse
- Opdel store pakke i mindre dele ("protocol data unit"); gen-samles hos modtager
- Opdeling i mindre pakker kan i princippet ske på alle lag

# Hvorfor lagdeling?

- Simplificerer opbygning af komplekse systemer:
  - Eksplisit struktur tillader at enkelt dele og deres sammenhæng kan identificeres og forståes i mindre bidder
  - Opbygning i simplere moduler letter vedligehold og opdatering af et system
    - Ændring af implementationen af en service kan foretages uden at ændre resten af systemet
- Ulemper ved lagdeling ?
  - Nogle funktioner kan blive besværlige eller performance mæssigt dyre at implementere
    - Statefull firewalls
    - NAT
- VIGTIGT GENERELT DATALOGISK PRINCIP:
  - Opdeling af kompleks funktionalitet i abstraktions-/virtualiserings-lag hver med vel-defineret funktionalitet, som kan være mere eller mindre strikt indkapslet

# OSI Reference Model

- En mere “principiel” model, standardiseret 7-lags model
  - Open Systems Interconnection Reference Model
  - International Organization for Standardization ("ISO 7498-2")



- Giver funktioner som brugere har behov for
- Konvertering mellem forskellige data-repræsentationer: gør at applikationer kan fortolke data ud fra deres betydning, f.x. kryptering, komprimering, maskine specifikke konventioner
- Styrer en dialog: synkronisering, checkpointing, gen-oprettelse af data udveksling
- Giver end-to-end levering
- Sender datagrammer over multiple links
- Sender frames med data
- Sender bits kodet som "signaler"

# Kritik af OSI & TCP/IP

## OSI:

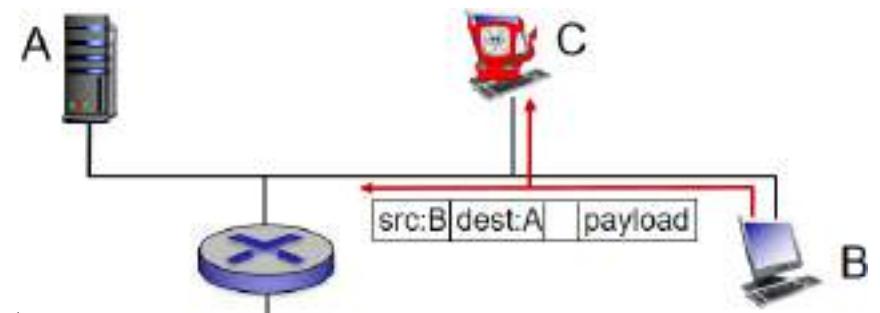
- + Meget indflydelsesrig model med klare begreber
- Modellen, protokollerne og adoption hæmmet af politik og kompleksitet

## TCP/IP:

- + Meget succesfulde protokoller, som virker godt og er meget udbredte
- Svag model, som er ***udledt efterfølgende*** ud fra de aktuelle protokoller

# Sikkerhed på nettet???

- Nettet blev oprindeligt lavet for en mindre lukket brugerskare med tillid til hinanden
  - ”sikkerhed” er ikke indbygget, men tilføjet ved nye protokoller og opsætning af begrænsninger på hvad videresendes
- Pakker kan ”sniffes” (wireshark)
  - Broadcast medier a la WiFi
  - Installeres på router
- Pakker kan ”spoofes” (opfind din egen header)
- Denial of service angreb, DOS,...
- ...



# Packet sniffing med Wireshark

The screenshot shows the Wireshark interface with several network packets captured. The packet list pane shows entries from 13 to 20, with details like source and destination IP addresses, protocols (TLSv1.2, TCP, SSOP), and lengths. The details pane provides a detailed breakdown of the selected packet (HTTP/1.1 200 OK) from the list. It includes fields such as Version (4), Header Length (20 bytes), Differentiated Services Field (DSCP: CS0, ECN: Not-ECT), Total Length (320), Identification (0x2a69), Flags (Don't fragment), Fragment Offset (0), Time to Live (64), Protocol (UDP), Header Checksum (0x8d1e), Source Address (192.168.0.182), Destination Address (192.168.0.111), and User Datagram Protocol details. The bytes pane at the bottom shows the raw hex and ASCII data of the selected packet.

Nb. Time Source Destination Protocol Length Info

13 0.232968 192.168.0.111 52.114.77.97 TLSv1.2 113 Application Data

14 0.273156 52.114.77.97 192.168.0.111 TLSv1.2 182 Application Data

15 0.288048 192.168.0.111 192.168.0.27 TCP 164 61595 + 8809 [PSH, ACK] Seq=1 Ack=1 Win=518 Len=118 [TCP segment of a reassembled PDU]

16 0.285715 192.168.0.27 192.168.0.111 TCP 164 8809 + 61595 [PSH, ACK] Seq=1 Ack=111 Win=279 Len=118 [TCP segment of a reassembled PDU]

17 0.288832 192.168.0.182 192.168.0.111 SSOP 334 HTTP/1.1 200 OK

18 0.326541 192.168.0.111 192.168.0.27 TCP 54 61595 + 8809 [ACK] Seq=111 Ack=111 Win=599 Len=8

...  
[Coloring Rule String: udp]  
Ethernet II, Src: Phillips\_78:a2:32 (00:17:88:78:a2:32), Dst: IntelCor\_97:47:4e (f4:4e:e3:97:47:4e)  
Internet Protocol Version 4, Src: 192.168.0.182, Dst: 192.168.0.111  
0100 .... = Version: 4  
.... 0101 = Header Length: 20 bytes (5)  
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)  
Total Length: 320  
Identification: 0x2a69 (18857)  
Flags: 0x40, Don't fragment  
...0 0000 0000 0000 = Fragment Offset: 0  
Time to Live: 64  
Protocol: UDP (17)  
Header Checksum: 0x8d1e [validation disabled]  
[Header checksum status: Unverified]  
Source Address: 192.168.0.182  
Destination Address: 192.168.0.111  
User Datagram Protocol, Src Port: 1980, Dst Port: 52625  
Source Port: 1980  
Destination Port: 52625  
Length: 300  
Checksum: 0x6887 [unverified]  
[Checksum Status: Unverified]  
[Stream Index: 0]  
[Timestamps]  
UDP payload (292 bytes)  
Simple Service Discovery Protocol  
HTTP/1.1 200 OK\r\nHOST: 239.255.255.250:1980\r\nEXT:\r\nCACHE-CONTROL: max-age=100\r\nLOCATION: http://192.168.0.182:88/description.xml\r\nSERVER: Hue/1.0 UPnP/1.0 IpBridge/1.50.0\r\nhue-bridgeid: 001780FFFFE70A232\r\nST: upnp:rootdevice\r\nUSN: uuid:2F482F80-da58-11e1-9b23-00178870a232::upnp:rootdevice\r\n\r\n

0000 f4 4e e3 97 47 4e 08 17 88 70 a2 32 08 08 45 08 N: ON: -p:2: E:  
0010 01 40 28 69 48 00 40 11 8d 1e c0 a8 00 66 c0 a8 #?#?#?----f:  
0020 00 6f 07 6c cd 91 01 2c 68 87 48 54 54 50 2f 31 .o.l..., h:HTTP/1:  
0030 2e 31 28 32 30 38 28 4f 4b ed 0a 48 4f 55 54 3a .1 200 O K: HOST:  
0040 28 32 33 39 3e 32 35 35 2e 32 35 28 239.255.255.250:  
0050 3a 31 39 38 39 0d 0a 45 t8 54 3a 0d 0a 41 41 43 :1980: E XT: CAC:

# Opgaverne idag

- Review: Har man forstået grundlæggende begreber
  - Packet switching, forsinkelser, flaskehalse, protokol stak
- Øvelser: Kan man anvende dem i nye eksempler?
  - Konkrete delay beregning
  - Flaskehalse
- Praktiske: Kan anvende netværksværktøjerne
  - Traceroute, start på Wireshark.

**SLUT**

# Internetværk og Web-programmering

## Applikationslaget

Forelæsning 10  
Brian Nielsen

Distributed, Embedded, Intelligent Systems



# Agenda

1. Applikationslagsprotokoller, og hvad er deres behov for data-transport
2. HTTP
  1. Struktur af Meddelelser
  2. Responstid og persistente forbindelser
  3. HTTP Caching
3. DNS
  1. Virkemåde
  2. DNS caching
  3. DNS værktøjer
4. (P2P)
  1. Bit-torrent

# Applikationslagsprotokoller

Hvad er netværksapplikationer og deres behov for kommunikation?

Hvordan kommunikerer applikationsprogrammer med hinanden?

Hvilke services stiller transport laget i TCP/IP modellen til rådighed for applikationer?

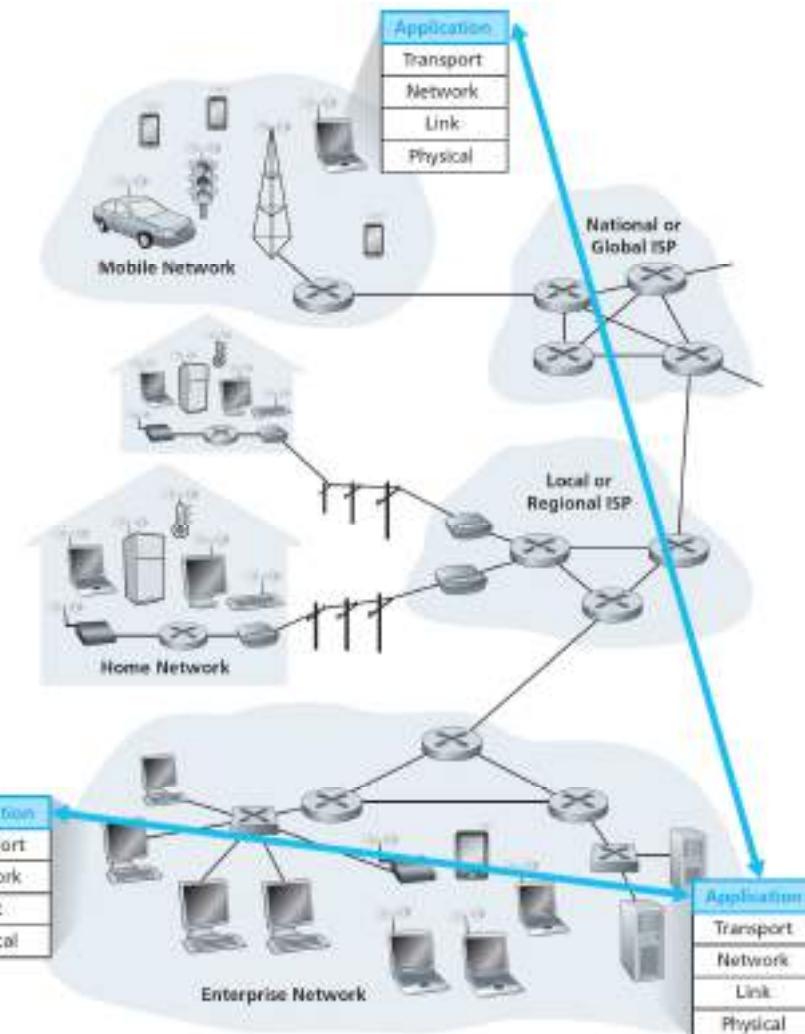
# Netværks applikationer

## Programmer som:

- Afvikles på (flere forskellige) **end systems**
- Kommunikerer over netværket
- Anvender applikations-niveau protokoller

## Eksempler

- Web-browser og web-server,
- Discord
- Mail,
- Skype,
- YouTube, Zoom, Teams,
- FTP,
- BitTorrent applikation,
- GoogleDocs, , GitHub, Dropbox,
- CityProbes data-opsamling+dashboard
- ...



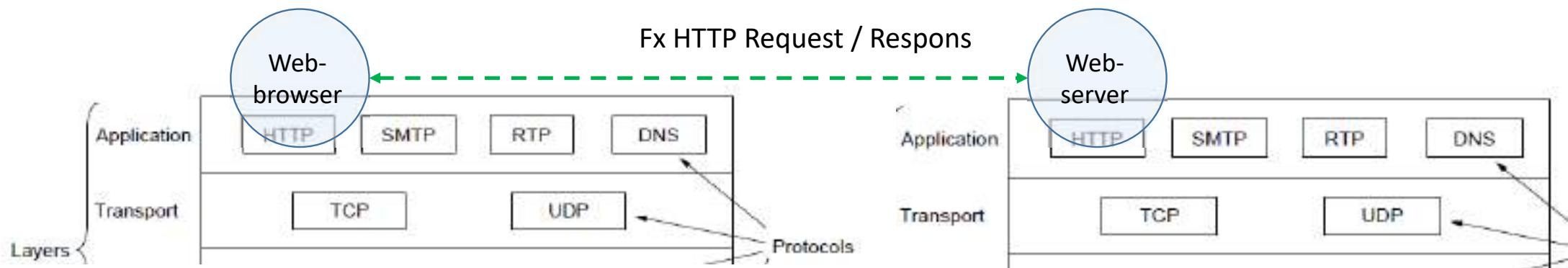
# Applikationskrav til kommunikation

| Applikation                       | Tilladt Data Tab | Throughput krav                          | Tids- og forsinkelsesfølsom |
|-----------------------------------|------------------|------------------------------------------|-----------------------------|
| Fil overførsel                    | Intet tab        | elastisk                                 | nej                         |
| e-mail                            | Intet tab        | elastisk                                 | nej                         |
| Web documenter                    | Intet tab        | elastisk                                 | nej                         |
| real-time/ interaktiv audio/video | tabs-tolerant    | audio: 5kbps-1Mbps<br>video:10kbps-5Mbps | ja, 100's msec              |
| stored audio/video                | Tabstolerant     | samme som ovenfor                        | ja, få secs                 |
| interaktive spil                  | tabs-tolerant    | få kbps og op                            | ja, 100's msec              |
| text messaging                    | Intet tab        | elastisk                                 | Ja og nej                   |

**De fleste applikationer har også behov for “sikkerhed” (security)**

- Kryptering, (hemmeligholdelse)
- Data integritet (ingen modificering),
- Authenticitet (Man er den, man giver sig ud for at være / vi ved hvem data kommer fra/skal til)

# Applikationslagsprotokoller



## En protokol definerer:

- Hvilken **type meddelelser** bliver udvekslet?
  - fx., request, response, ack,...
- Meddelelses **syntax**:
  - Hvilke felter er der i meddelelsen, og hvordan afgrenses de?
  - Dvs. definerer parametre+struktur
- Meddelelsens **betydning** (semantics)
  - Hvad betyder informationen i meddelelsen?
- **Regler** for hvordan meddelelser skal behandles og besvares

## Eksempler:

- Hypertext Transfer Protocol, Simple Mail Transfer Protocol
- Domain Name System Protocol, WebSockets
- Bit Torrent, Network Time Protocol,
- ...

## Åbne protokoller:

- defineret i RFCs
- tillader interoperabilitet
- e.g., HTTP, SMTP,

## Proprietære protokoller

- e.g., Skype

# Services fra Internet transport protokoller

## TCP (transmission control protocol) service:

- **pålidelig** transport af en sekvens af bytes (“rørledning” / byte stream) mellem sender og modtager proces
  - Hvis sender og modtager ikke crasher, og netværket ikke fejler permanent, bliver data leveret til modtager korrekt, uden tab, i afsendt rækkefølge
- **flow kontrol:** en hurtig sender overbebyrder ikke modtager
- **congestion kontrol:** sender (ned-)justerer sende raten når netværket er overbelastet (trafikprop i en router)
- **Forbindelses-orienteret** (connection-oriented): Der skal etableres en forbindelse mellem klient og server proces “handshake”

## GIVER IKKE

- tids/latens garantier,
- ingen mulighed for kapacitetsreservation (minimum throughput),
- ingen sikkerhed og kryptering
  - (kræver overbygning på applikationslaget “transport layer security”/ “Secure socket layer”)

## UDP (“User datagram protokol”) service:

- ” **best-effort** data transport
  - **Datagrammer**
  - **Connection less**
  - **Kan mistes, dublikeres, omordnes af netværket (fx pga forskellig rute)**

**GIVER IKKE** pålidelighed, flow control, congestion control, timing, throughput garanti, sikkerhed, eller forbindelser

- **SPM: Hva skal vi med den til?** Hvorfor findes den så?

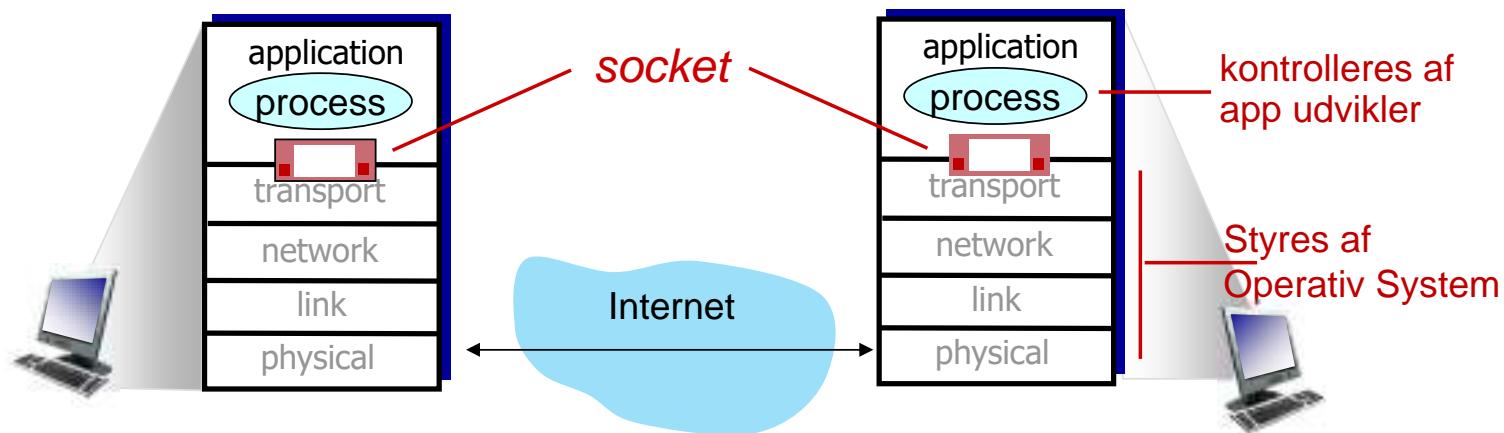
# Processer



- Applikationslaget kommunikerer mellem processer på forskellige hosts
- Applikationen afvikles i en "proces" = et kørende program på én host
  - klient proces: proces som initierer kommunikation
  - server proces: proces som afventer at blive kontaktet
  - En given proces kan have begge "roller"
- Processer kan fx vises med "task manager" eller "ps -aux"
- Processer på samme hosts kan kommunikere ved "inter-proces kommunikation"
  - `ls /usr/include | grep "stdio"`
- Mellem forskellige hosts (værtsmaskiner) vha. transport-laget

# Sockets

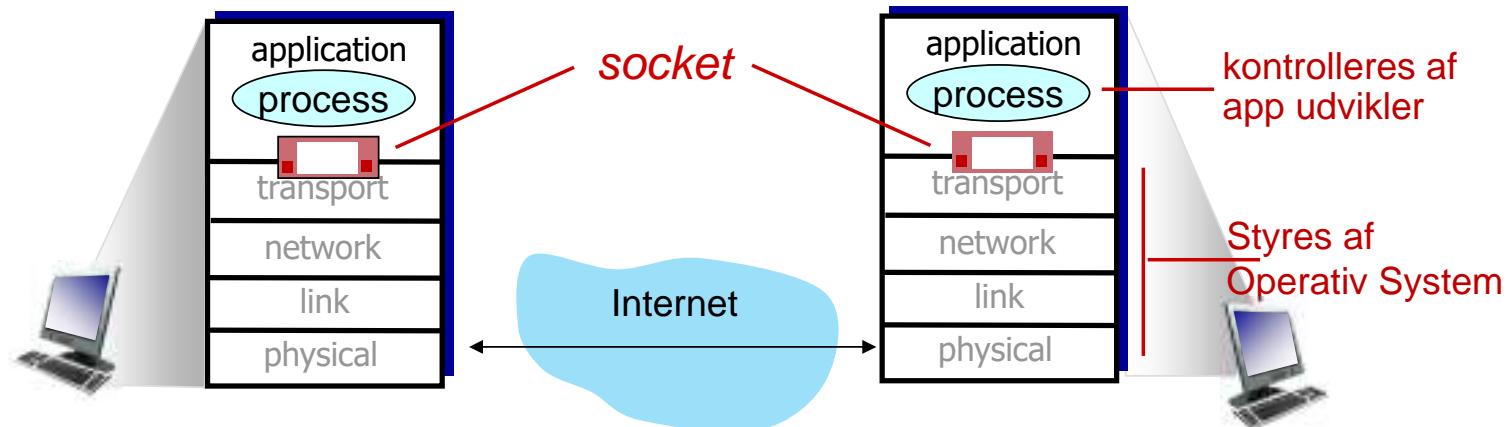
- Processer sender/modtager meddelelser via en ("fatning") **socket**,
- Bindeled/dør mellem applikationslag og transport lag
  - Kræver (mindst) 2-sockets: en på hver side
  - Oprettes af process ved kald til "socket" programmerings-interface



# Hvordan kontaktes en Proces?

Fx en server-proces

- Skal have en entydigt identifikation på nettet
- Proces adresseres vha.
  - Hosts **IP-nummer** fx:130.225.63.3
  - Et **port nummer** på host fx: 80
  - CS web-server kontaktes på: 130.225.63.3:80
  - Server process forventes at lave en socket til porten, som den bruge til at "lytte" på
- Mere præcist: Vi kontakter den proces på angivne host, der lytter på en "socket" som er bundet til den angivne port



# Velkendte Porte

- Anerkendte services har fast tildelte, reserverede porte
  - Vedligeholdes af [Internet Assigned Numbers Authority](#) (IANA)
  - Portnumre 0-1023 er alle reserverede

Fx,

| Port Number | Transport Protocol | Service Name                                                                                     | RFC                      |
|-------------|--------------------|--------------------------------------------------------------------------------------------------|--------------------------|
| 20, 21      | TCP                | File Transfer Protocol (FTP)                                                                     | RFC 959                  |
| 22          | TCP and UDP        | Secure Shell (SSH)                                                                               | RFC 4250-4256            |
| 23          | TCP                | Telnet                                                                                           | RFC 854                  |
| 25          | TCP                | Simple Mail Transfer Protocol (SMTP)                                                             | RFC 5321                 |
| 53          | TCP and UDP        | Domain Name Server (DNS)                                                                         | RFC 1034-1035            |
| 67, 68      | UDP                | Dynamic Host Configuration Protocol (DHCP)                                                       | RFC 2131                 |
| 69          | UDP                | Trivial File Transfer Protocol (TFTP)                                                            | RFC 1350                 |
| 80          | TCP                | HyperText Transfer Protocol (HTTP)                                                               | RFC 2616                 |
| 110         | TCP                | Post Office Protocol (POP3)                                                                      | RFC 1939                 |
| 119         | TCP                | Network News Transport Protocol (NNTP)                                                           | RFC 8977                 |
| 123         | UDP                | Network Time Protocol (NTP)                                                                      | RFC 5905                 |
| 135-139     | TCP and UDP        | NetBIOS                                                                                          | RFC 1001-1002            |
| 143         | TCP and UDP        | Internet Message Access Protocol (IMAP4)                                                         | RFC 3501                 |
| 161, 162    | TCP and UDP        | Simple Network Management Protocol (SNMP)                                                        | RFC 1901-1908, 3411-3418 |
| 179         | TCP                | Border Gateway Protocol (BGP)                                                                    | RFC 4271                 |
| 389         | TCP and UDP        | Lightweight Directory Access Protocol                                                            | RFC 4510                 |
| 443         | TCP and UDP        | HTTP with Secure Sockets Layer (SSL)                                                             | RFC 2818                 |
| 500         | UDP                | Internet Security Association and Key Management Protocol (ISAKMP) / Internet Key Exchange (IKE) | RFC 2408 - 2409          |
| 636         | TCP and UDP        | Lightweight Directory Access Protocol over TLS/SSL (LDAPS)                                       | RFC 4513                 |
| 989/990     | TCP                | FTP over TLS/SSL                                                                                 | RFC 4217                 |

# HTTP

Hvad bruges HTTP kontrol headers til?

Hvordan struktureres/formateres HTTP meddelelser?

Hvordan effektiviseres HTTP kommunikation?

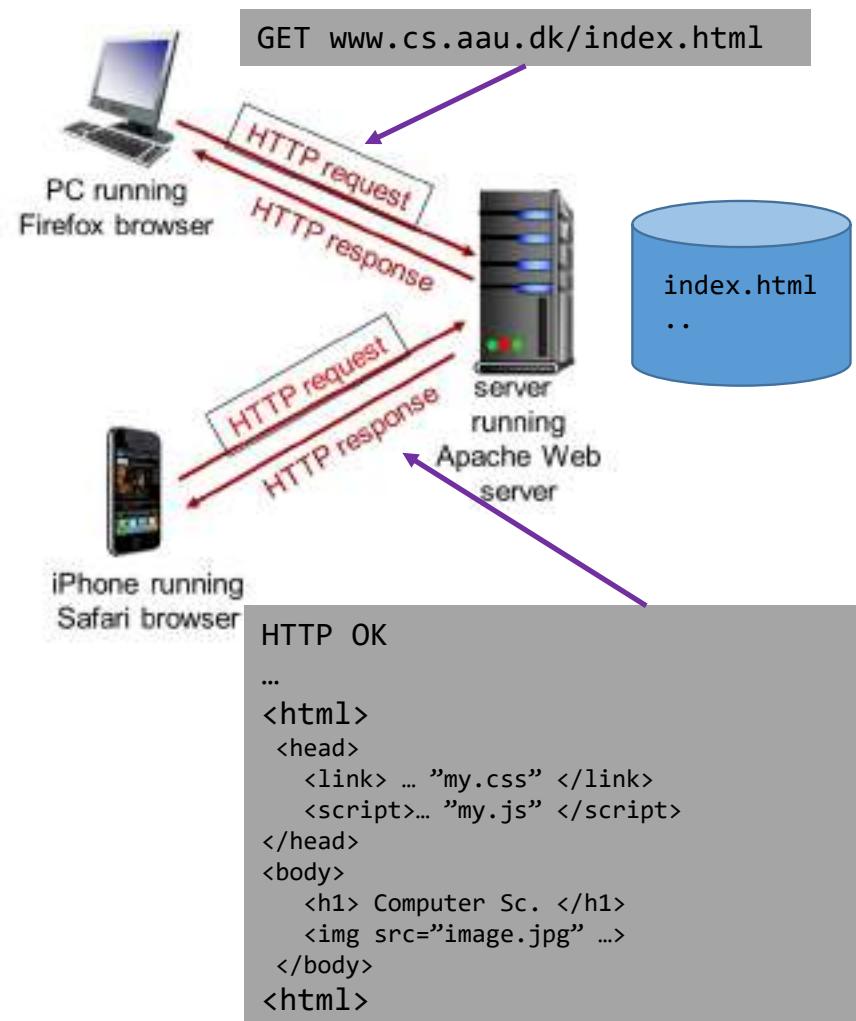
Hvordan håndteres forbindelser til serveren?

Hvorfor og hvordan "cacher" HTTP dokumenter?

# Simpel HTTP Scenarie: statisk html fil

## HTTP: hypertext transfer protocol

- applikationslags-protokol til web-trafik
- client/server model:
- **klient:** program (typisk browser)
  1. Klient opretter TCP forbindelse til server, typisk port 80
  2. Sender forespørgsel (vha. HTTP GET), om en web side
  3. Afventer respons
  4. Parser respons og optegner siden
  5. Kører evt. skridt 1-4 igen (sideløbende) for at henter nødvendige indlejrede ressourcer (billeder, js-scripts, style sheets)
- **server:**
  1. Afventer
  2. Modtager HTTP forespørgsel fra klient
  3. Behandler den, og beregner svar (fx indlæser filen),
  4. Sender HTTP respons ( fx med fil-indhold) til klienten
- Server og ressourcen angives ved et Uniform Resource Identifier, normalt URL  
Det eksakte forløb afhænger af HTTP protokol version



# HTTP Meddelelser (syntax)

- HTTP forespørgsel
  - Sendes som text (menneske-læsbar)
  - I protokol beskrivelser opstilles formatet som en tabel med felter

request line  
(GET, POST,  
HEAD commands)

header  
lines

carriage return,  
line feed at start  
of line indicates  
end of header lines

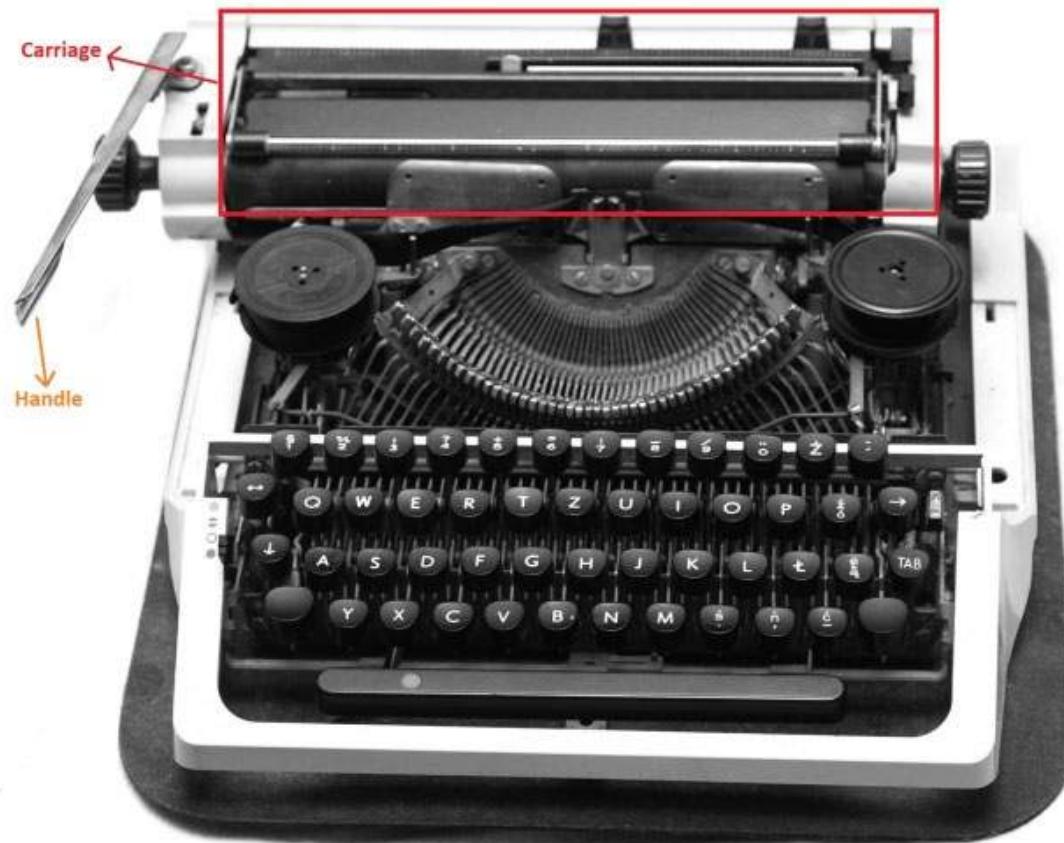
```
GET /index.html HTTP/1.1\r\nHost: www-net.cs.umass.edu\r\nUser-Agent: Firefox/3.6.10\r\nAccept: text/html,application/xhtml+xml\r\nAccept-Language: en-us,en;q=0.5\r\nAccept-Encoding: gzip,deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7\r\nKeep-Alive: 115\r\nConnection: keep-alive\r\n\r\n
```

Generelt format for HTTP forespørgsel

| method            | sp | URL   | sp | version | cr           | If | request line |  |  |
|-------------------|----|-------|----|---------|--------------|----|--------------|--|--|
| header field name | sp | value | cr | If      | header lines |    |              |  |  |
|                   |    |       |    |         |              |    |              |  |  |
| header field name | sp | value | cr | If      | entity body  |    |              |  |  |
| cr                | If |       |    |         |              |    |              |  |  |

HTTP respons har en tilsvarende format

# Carriage Return-Line Feed

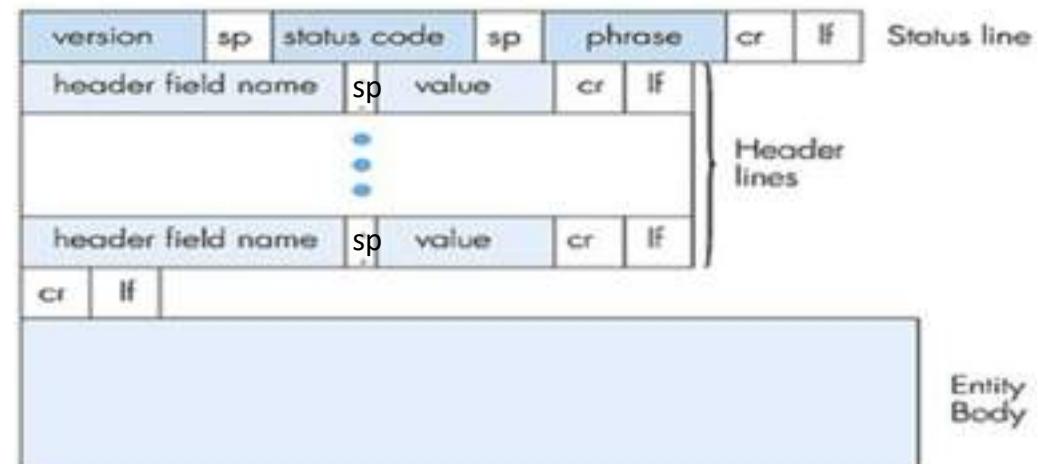


# HTTP Meddelelser: Respons

- HTTP Respons
    - Et lignende format

```
status line
(protocol
status code
status phrase) HTTP/1.1 200 OK\r\nDate: Sun, 26 Sep 2010 20:09:20 GMT\r\nServer: Apache/2.0.52 (CentOS)\r\nLast-Modified: Tue, 30 Oct 2007 17:00:02
GMT\r\nETag: "17dc6-a5c-bf716880"\r\nAccept-Ranges: bytes\r\nContent-Length: 2652\r\nKeep-Alive: timeout=10, max=100\r\nConnection: Keep-Alive\r\nContent-Type: text/html; charset=ISO-8859-
1\r\n\r\ndata, e.g.,
requested
HTML file data data data data data ...
```

## Generelt format for HTTP respons



# HTTP klient demo

- netcat værktøj: opretter TCP forbindelse
  - Udskriver hvad den modtager
  - Videresender hvad den får på input

```
root@AAU131963:~# nc -C -v www.cs.aau.dk 80
Connection to www.cs.aau.dk 80 port [tcp/http] succeeded!
GET / HTTP/1.1
Host: www.cs.aau.dk

HTTP/1.1 301 Moved Permanently
Date: Wed, 06 Apr 2022 08:19:24 GMT
Server: Apache
Location: https://www.cs.aau.dk/
Content-Length: 230
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved here.</p>
</body></html>
```

- åbner TCP forbindelse til port 80 (std. HTTP server port) på [www.cs.aau.dk](http://www.cs.aau.dk)
- Hvad du nu indtaster sendes til port 80 på [www.cs.aau.dk](http://www.cs.aau.dk)
- Minimalt valid HTTP GET:  
  
GET / HTTP/1.1  
Host: [www.cs.aau.dk](http://www.cs.aau.dk)  
  
+2\*Enter
- Inspicér responset

<https://blog.desdelinux.net/en/using-netcat-some-practical-commands/>  
<https://www.varonis.com/blog/netcat-commands>

# Netcat som server



The screenshot shows a web browser window with the address bar containing "localhost:3000/index.html". The main content area of the browser displays the raw HTTP request sent by a client to a netcat listener on port 3000. The request includes various headers such as Host, Connection, Cache-Control, and User-Agent, indicating the client is a modern web browser.

```
root@AAU131963:~# nc -l 3000
GET /index.html HTTP/1.1
Host: localhost:3000
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="100", "Google Chrome";v="100"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.75 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,da;q=0.7,nb;q=0.6,de;q=0.5,en-GB-oxendict;q=0.4,en-AU;q=0.3,en-CA;q=0.2,en-NZ;q=0.1,en-ZA;q=0.1

root@AAU131963:~#
```

# Egen protocol over TCP????

The image displays two terminal windows side-by-side, both titled "root@AAU131963: ~".

**Terminal 1 (Left):**

```
root@AAU131963:~# netcat -C -v localhost 3000
Connection to localhost 3000 port [tcp/*] succeeded!
hej
hej selv

dumme
selv
```

**Terminal 2 (Right):**

```
root@AAU131963:~# nc -l 3000
hej
hej selv

dumme
selv
```

The left terminal is running a netcat server on port 3000, listening for connections. It receives the message "hej" and replies with "hej selv". It then receives the message "dumme" and replies with "selv".

The right terminal is running a netcat client, connecting to the server at port 3000. It sends the message "hej" and receives the reply "hej selv". It then sends the message "dumme" and receives the reply "selv".

# HTTP Features

- Indholdsforhandling (foretrukne formation)
- **Forbindelseshåndtering**
- Cookies til sessionsstyring
- Cross Origin Ressource Sharing (CORS)
- Adgangskontrol (Authentication)
- **Caching**
- Data Kompression
- Omdirigering
- Portionslæsning (range requests)

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

# HTTP Forbindelseshåndtering

Hvordan effektiviseres HTTP client-server kommunikation?

Hvordan bruges TCP bedst?

# HTTP Respons Tid

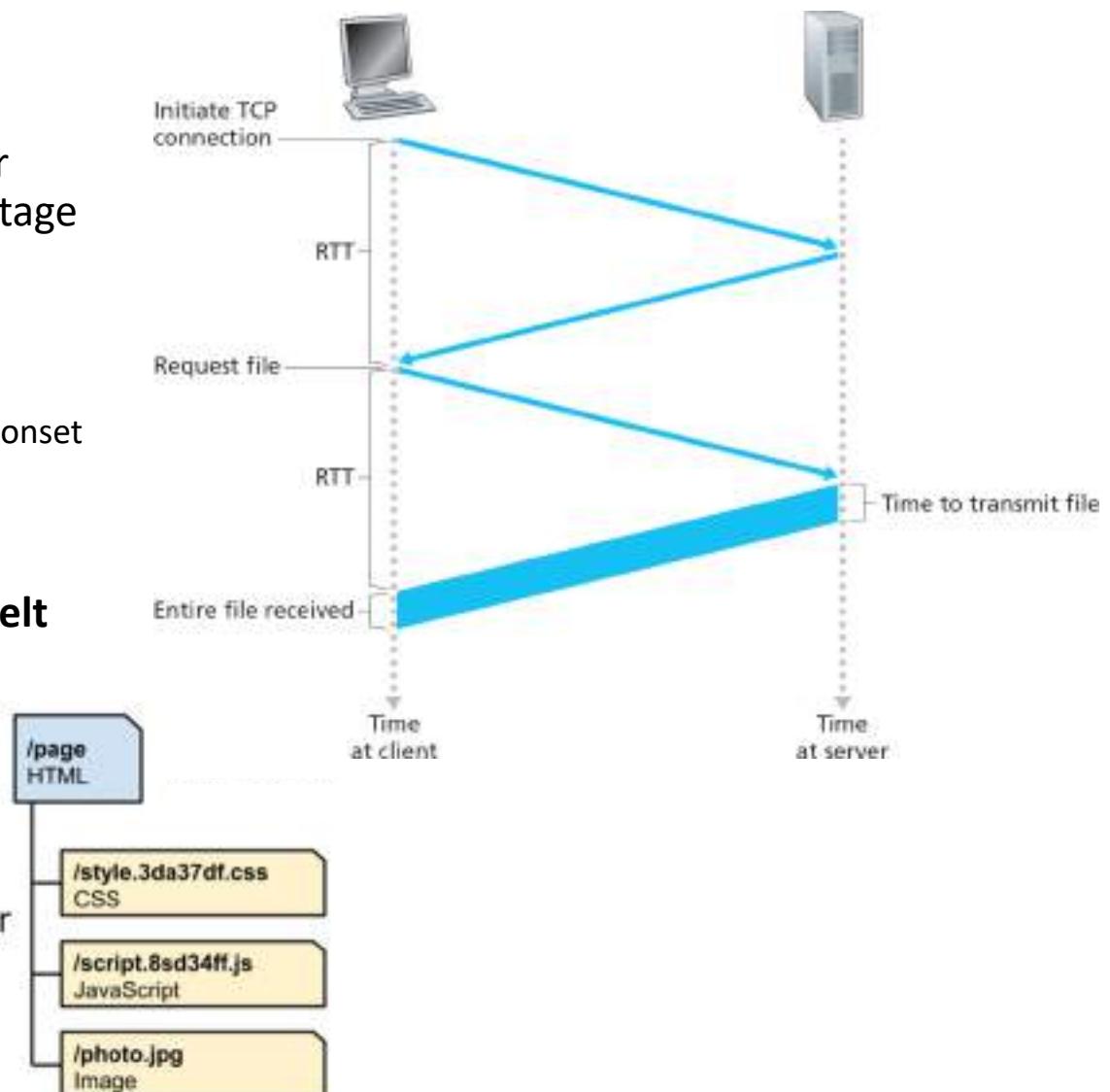
**RTT (definition): round-trip-tid:** tiden, det tager for klienten at sende en lille pakke til serveren og modtage svaret

## HTTP respons tid

- En RTT til at opsætte TCP forbindelsen (handshake)
  - En RTT for HTTP request og første få bytes af HTTP responset
  - Fil transmissionstid
- = 2RTT + file transmission time

## Simple browsere henter linkede objekter sekventielt

- Hvis proceduren gentages sekventielt for hvert objekt bliver det **langsamt** at indlæse en side
- I eksemplet: 4 \* HTTP Responstid
  - 1 \* HTTP Responstid til at hente siden /page.html
  - + 3 \* HTTP Responstid til at hente linkede objekter
  - >8 RTT

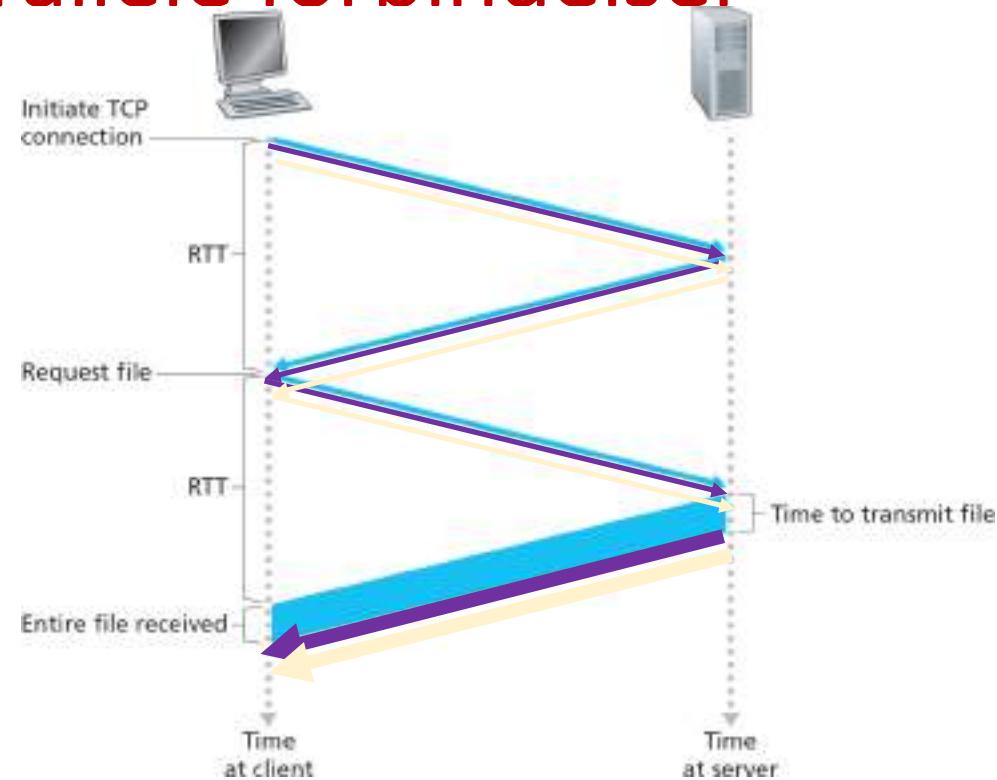
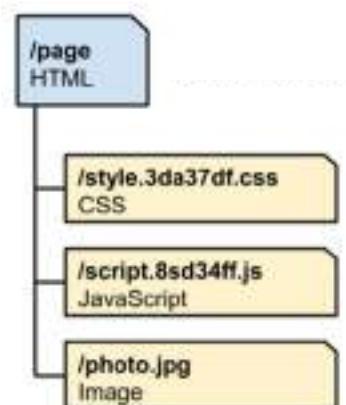


# HTTP Responstid med parrallele forbindelser

Klient kan oprette flere “parallelle” forbindelser til server

**HTTP respons tid:**

- $2 \text{RTT} + \text{file transmission time} + \text{server overhead}$  og evt.  
Kapacitetsbegrænsninger downlink til klient
- **Meget hurtigere for klienten**
- **Krævende for server**, da den skal reservere buffer kapacitet til hver  
forbindelse (gør dette for mange klinter)
- I eksemplet:  $>2 * \text{HTTP Responstid}$ 
  - $1 * \text{HTTP Responstid}$  til at hente siden (/page.html)
  - $+ 1 * \text{HTTP Responstid}$  til at hente linkede objekter
  - $+ \text{transmissionstid for 3 filer}$



3 parallele forbindelser

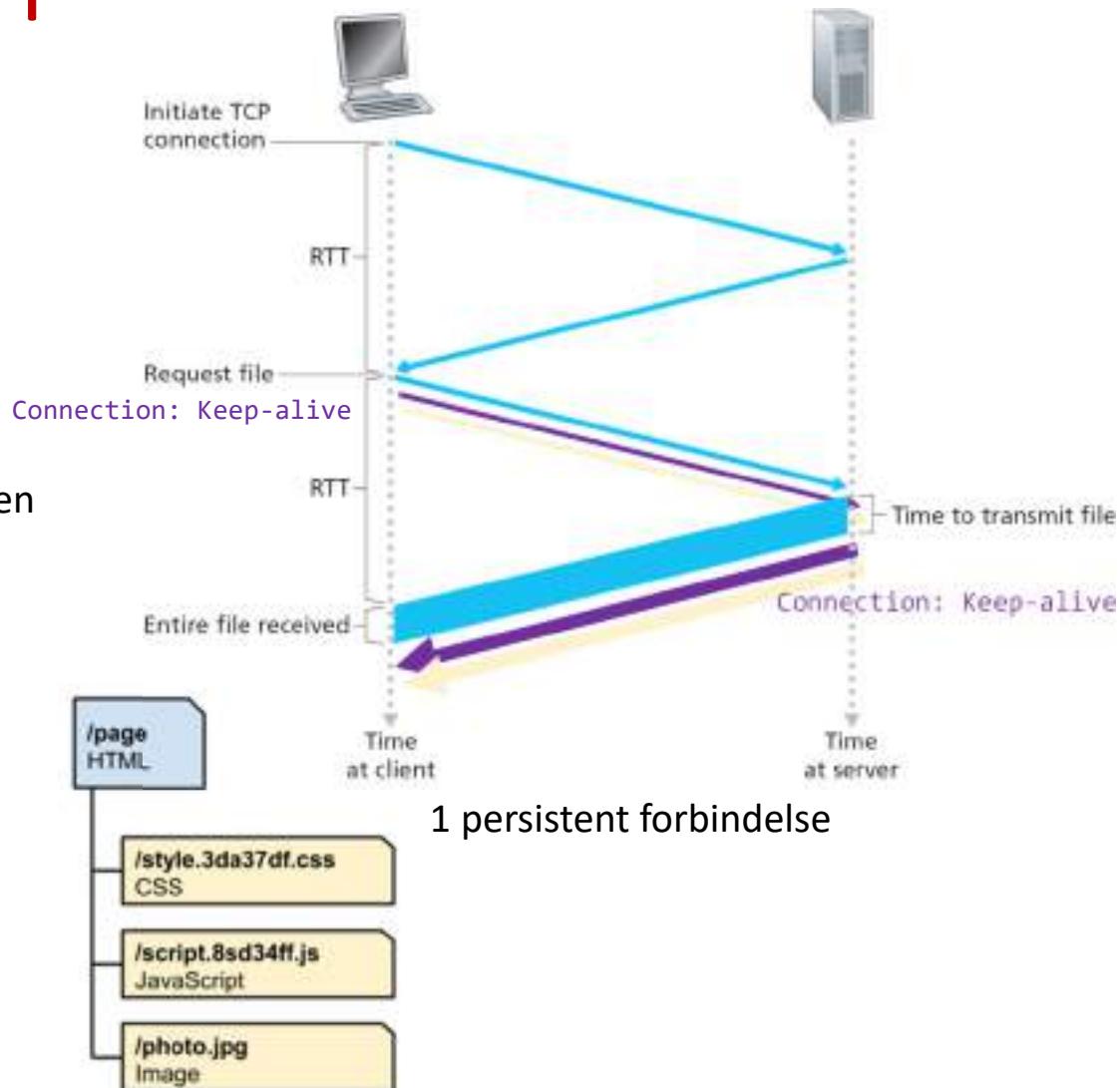
# Løsning: Persistent HTTP

## Persistent HTTP (default i HTTP/1.1):

- Server holder forbindelsen åben efter afsendelse af respons
- Efterfølgende HTTP forespørgsler og svar sendes på samme åbne forbindelse:
  - Sparer overhead ved oprettelse af TCP forbindelser

## HTTP Pipelining

- Forespørgsler kan “pipelines”: sendes efter-hinanden uden at afvente svar “samlebånds princip”



# Løsning: Persistent HTTP

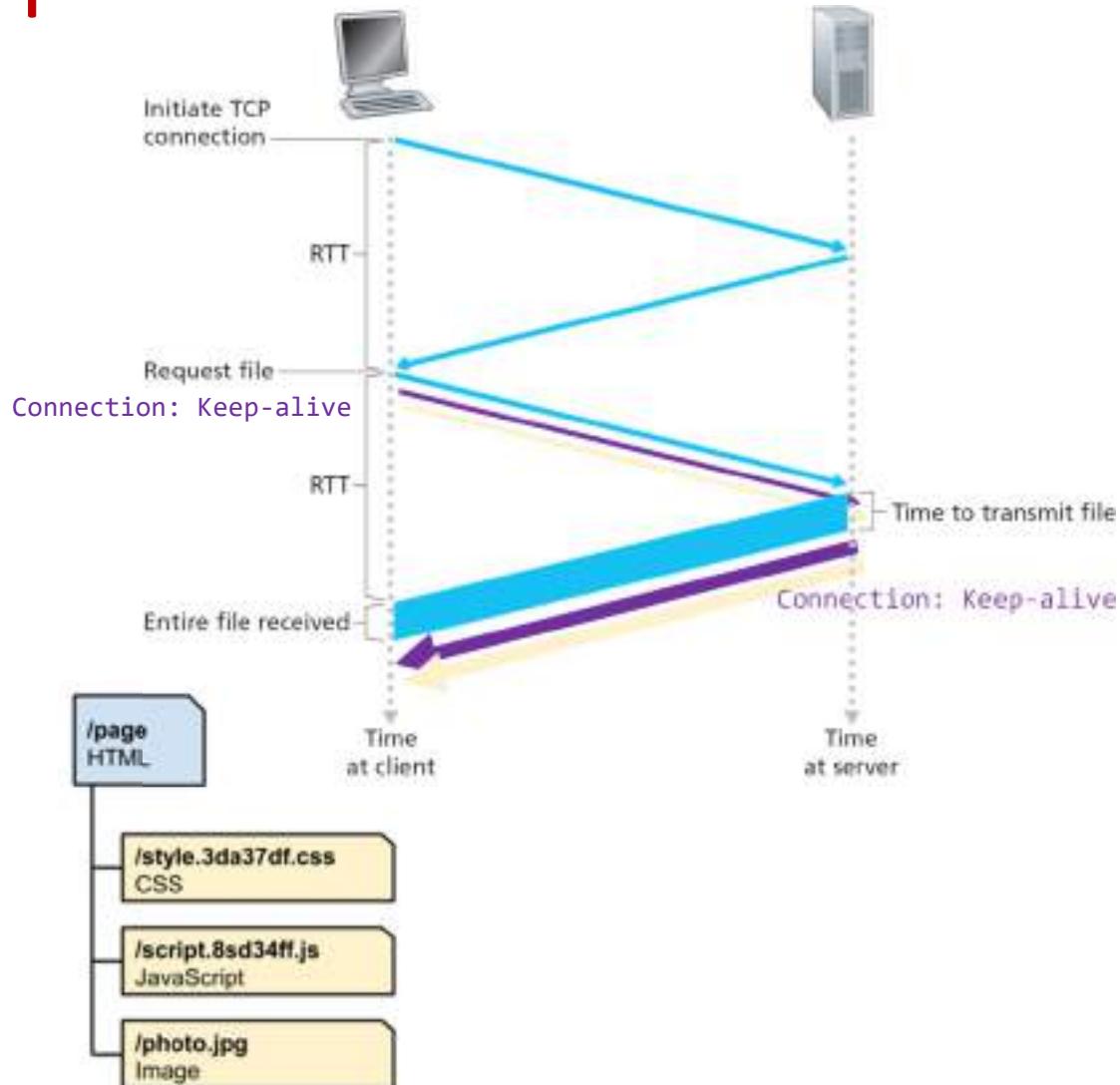
## HTTP response tid med pipelining:

- $2\text{RTT} + \text{file transmission time} + \text{server overhead}$  og evt. kapacitetsbegrænsninger downlink til klient
- **Meget hurtigere for klienten**
- **Mindre krævende for server**, da den kun skal vedligeholde én forbindelse pr klient, dog skal klienten lukke forbindelsen så snart den er færdig.
- I eksemplet:

- 1\* HTTP Responstid (GET /page.html)
- 1\*RTT+3\*fil transmissionstid

```
GET style.css
GET script.js
GET photo.jpg
```

- HTTP/2 tager denne idé videre



# HTTP/2

**Mål:** mindske forsinkelsen i multi-objekt HTTP requests

HTTP/2: [RFC 7540, 2015] øget fleksibilitet hos *server* ved afsending af web-objekter til klient:

- metoder, status koder, fleste header felter uforandret fra HTTP 1.1
- Transmissions rækkefølgen af forespurgte objekter baseres på en prioritet angivet af klienten (ikke nødvendigvis FCFS: First-Come-First-Serve)
- Server kan afsende (*push*) objekter til klienten, den (endnu) ikke har forespurgt
- Opdeling af større objekter i "frames", scheduler frames for at formindste HOL ("Head of Line") blokering

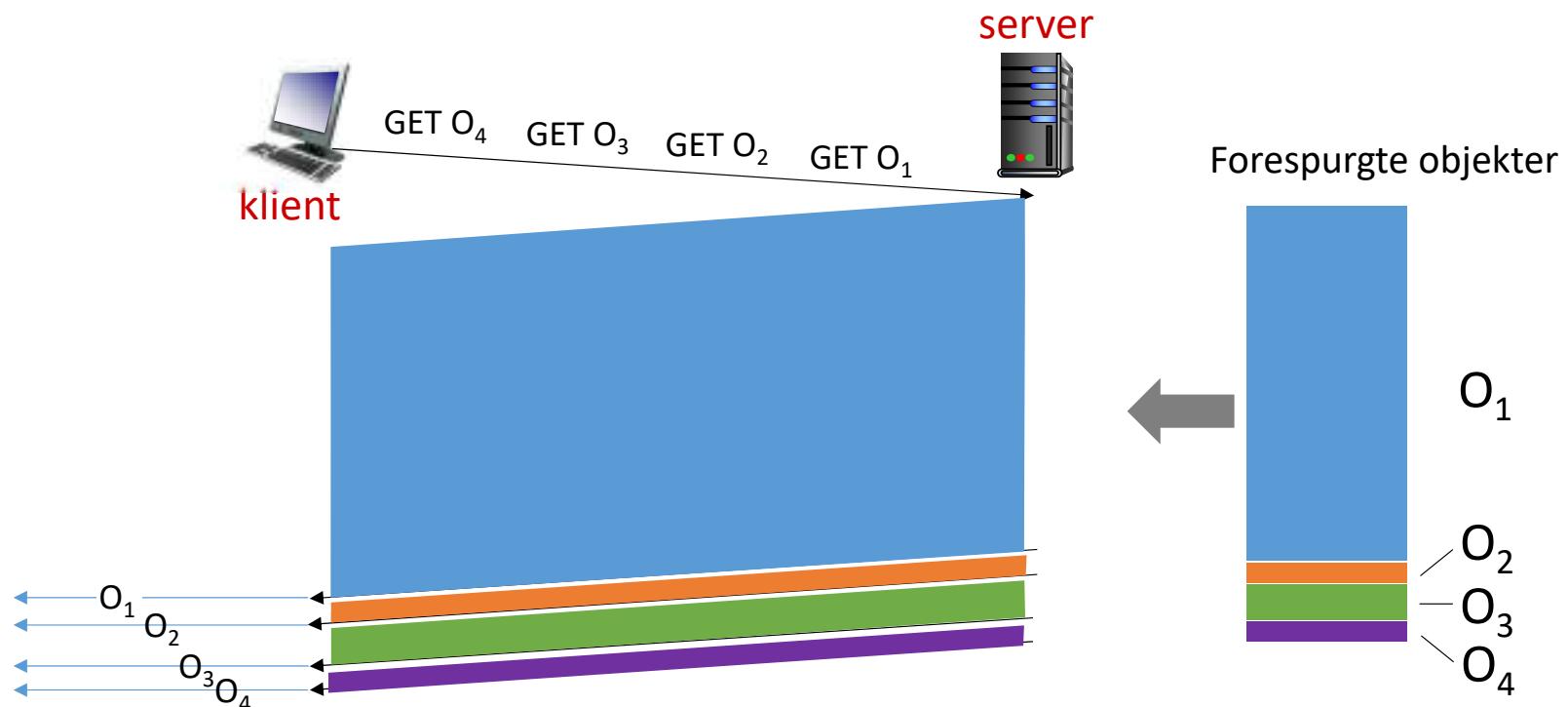
# Head-of-Line Blocking

Et stort, langvarigt job blokkerer for fremdrift af (flere) ventende små  
Fx i en supermarkeds kø



# HTTP/2: reduktion af HOL blokering

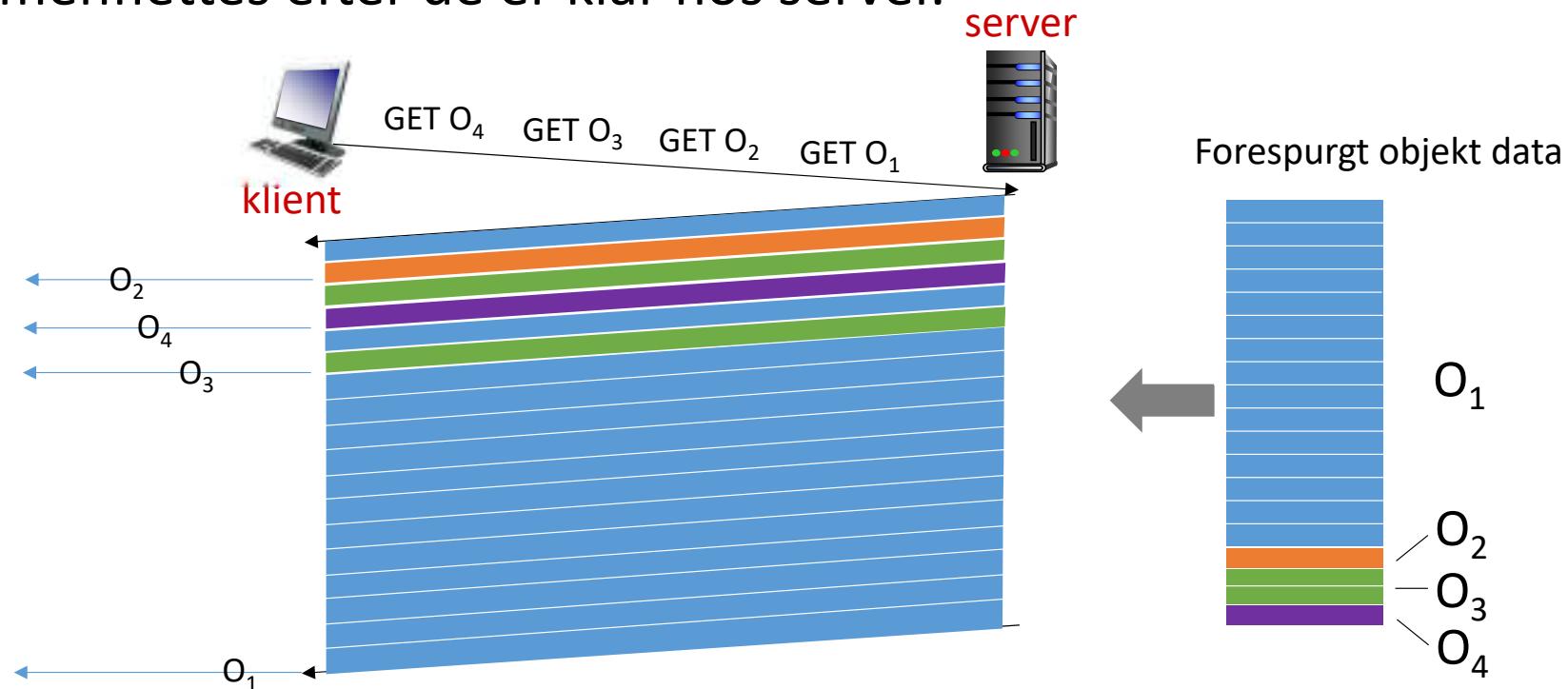
HTTP 1.1: klient beder om 1 stort objekt (e.g., video fil) and 3 mindre



I HTTP1.1 leveres svar objekter i den rækkefølge de efterspørges:  $O_2$ ,  $O_3$ ,  $O_4$  venter "bag"  $O_1$

# HTTP/2: reduktion af HOL blokering

HTTP/2: objekt deles op i bidder ("frames"); afsendelse af frame sammenflettes efter de er klar hos server.



O<sub>2</sub>, O<sub>3</sub>, O<sub>4</sub> leveres hurtigt, O<sub>1</sub> forsinkes en smule

# Fra HTTP/2 til HTTP/3

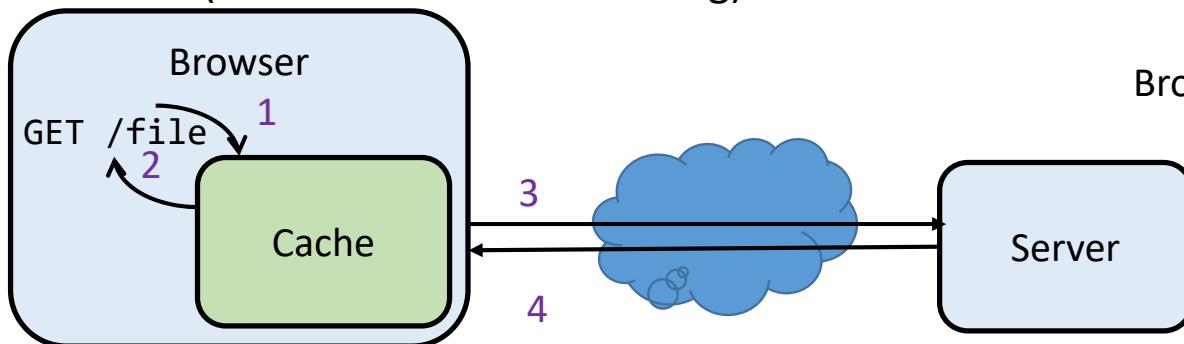
HTTP/2 over enkelt TCP forbindelse betyder

- Gentransmission ved pakketab får transmission af alle objekter til at gå i stå (“stalling) (TCP: ordnet, pålidelig byte-stream service)
  - Som i HTTP 1.1, får browser incitemment til at åbne multiple parallel TCP forbindelser to reducere stalling, og dermed øge samlet throughput
- Ingen sikkerhed over TCP connection
- **HTTP/3:** tilføjer sikkerhed, per objekt fejl- og congestion-kontrol over UDP (QUIC)
  - Lidt mere når vi kommer til transport laget

# HTTP Caching

# HTTP Caching

- En "cache" er et lokalt lager (forråd), der gemmer kopi af forespurgte objekter for at give ***hurtig*** adgang dertil
  - Optimeringstrick for programmer: gemmer og genbruger nyligt beregnede resultater
  - L1-L3 cache i computer arkitektur, disk/fil-cache,...
  - Web-cache
- I HTTP Web sammenhæng: netværket er "langsomt",
  - **Hurtigere svartid / visning af siden**
  - Mindsker trafikken til server (penge, strøm)
  - (Mindsker server belastning)



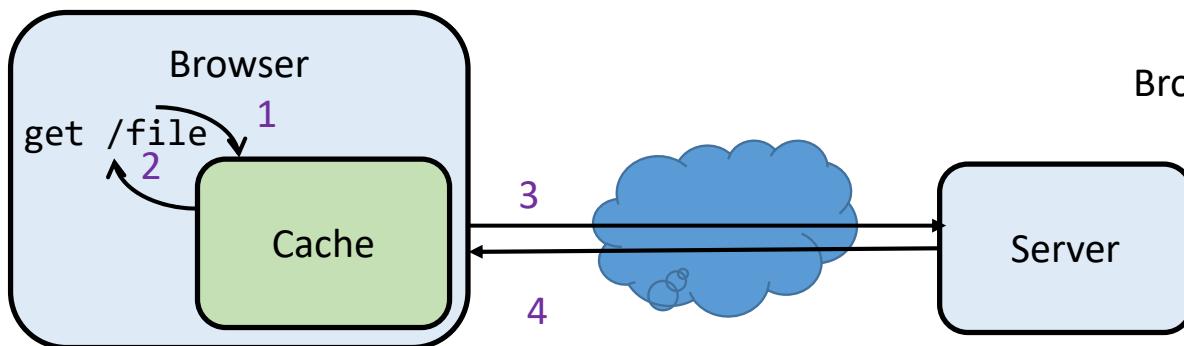
- Browser gemmer nyligt brugte objekter/filer på klient
1. Browser kigger først i cache om det efterspurgte objekt ("fil") findes der
  2. Hvis ja, hent objektet derfra
  3. Hvis nej, send forespørgsel til web-server
  4. Gem objektet i cache, og leverer objektet

- Hvad kan caches? Hvad hvis det ændres på server?

# HTTP Caching

- HTTP headers muliggør at *serveren* kan angive
  - Hvorvidt et objekt kan caches (om det er genbrugeligt)
  - Hvor længe kopien er gyldig, mm
- Kun serveren (og web-app programmør) kender til "foranderligheden" af objektet
  - Sjældent ændres: CSS filer, fleste billeder, iconer, varemærker, statiske html filer
  - Ofte ændres: dynamiske HTML baseret på DB opslag

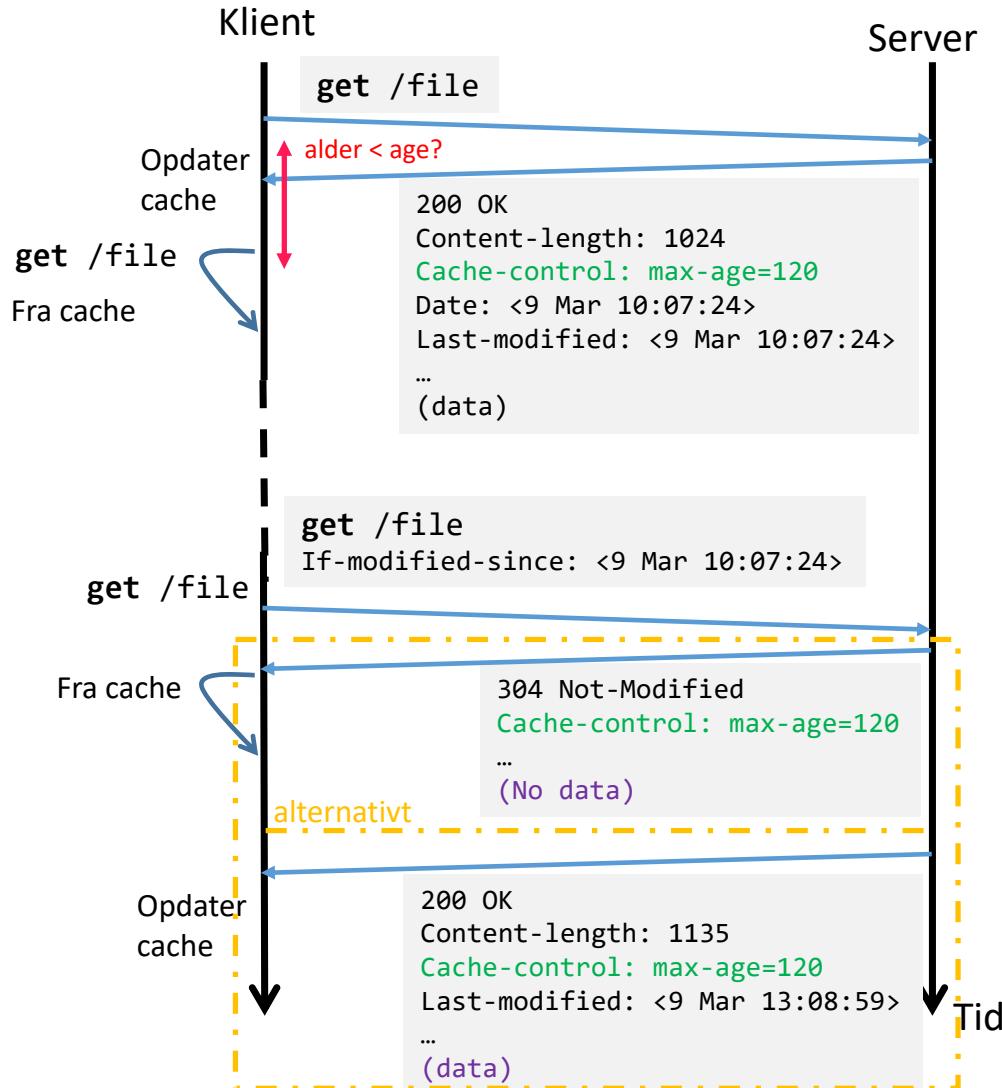
Cache-Control: max-age=<seconds>



- Browser gemmer nyligt brugte objekter/filer på klient
1. Browser kigger først i cache om det efterspurgte objekt ("fil") findes der
  2. Hvis ja, hent objektet derfra
  3. Hvis nej, send forespørgsel til web-server
  4. Gem objektet i cache, og leverer objektet

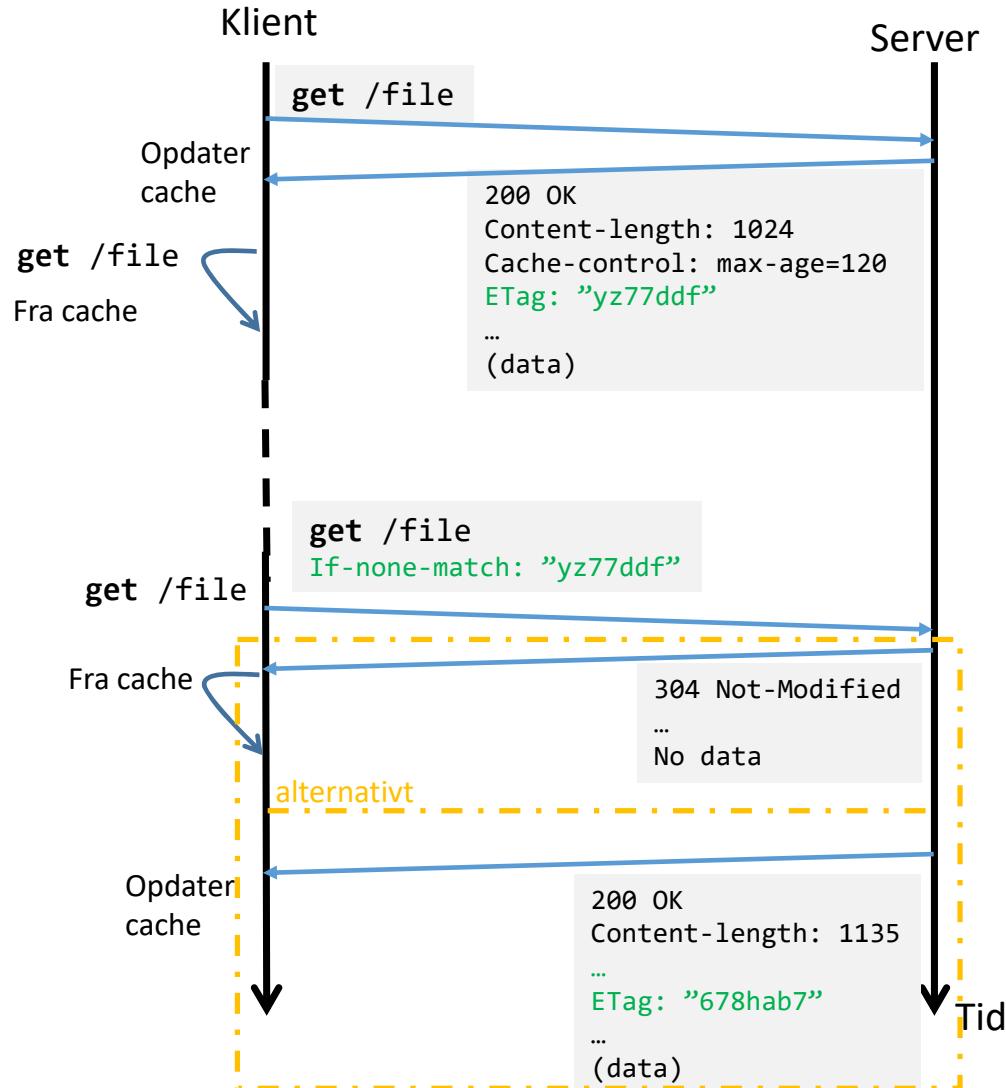
- Hvis objektet ændres, hvordan finder klienten så ud af den ikke bruger forældet info?

# HTTP Cache kontrol: Betinget forespørgsel



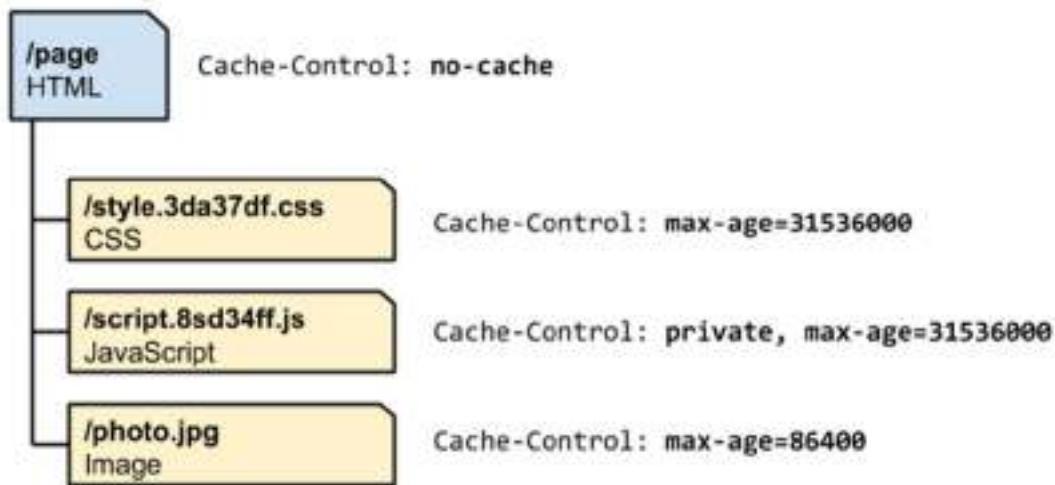
- **Max-age:** angiver i sekunder hvor lang tid objektet kan betragtes som "frisk" og dermed om klienten må bruge det cachede objekt
  - Fresh: Tidsinterval fra tidspunkt hvor server genererer svar + max-age
    - Typisk beregnet på basis af Date eller Last-modified header i respons
  - Stale: sidste anvendelses dato overskredet
- Hvis Stale: Klient anvender "Conditional Get" som checker med server om cachens objekt er up-to-date
  - Klienten gemmer det modtagne tids-stempel sammen med objektet, og medsender dette
  - Server sender ét af 2 mulige svar:
    - Objekt er ikke ændret siden <tidsstempel>:
      - INGEN data, evt ny forlænget levetid
    - Objektet er ændret siden <tidsstempel>+data

# HTTP Cache kontrol: Betinget forespørgsel, TAGS



- **ETAG**: en streng, der fungerer som versionsnummer, som klienten kan bruge til at validere om indholdet et objekt den har er "ens" med serverens
- fingeraftryk, teknisk set ofte beregnet som et "hash"
- Conditional Get: checker om cachens version stemmer overens med serverens version
  - Klienten medsender det modtagte versionsnummer
- Flere anvendelser:
  - Tidsopløsning på tidsstemplér er lav (1sek)
  - Undgå "lost-update": Hvis 2 klienter *samtidigt* forsøger med opdatering (put) (overskrivning) kan et afvises

# HTTP Cache kontrol



- **No-cache:** lokalt indhold skal altid valideres hos server først (sparer stadig data, hvis objektet må genbruges)
- **No-store:** Objektet må ikke gemmes i cache; skal hentes på ny for hver forespørgsel.
- **Private vs. public.** Privat på kun caches i klientens cache
- **Max-age , If-modified-since**
- ...
- **Hvad caches?**
  - Objekter hentet med GET + status kode (Læsning af ressource)
  - Objekter +status hentet med POST caches normal IKKE (ændring af ressource), skal eksplisit tillades
  - PUT / Delete respons må ikke caches
  - Objekter, der ikke er undtaget vha. cache-kontrol header
- Gennemtvivng reload i browser
- Tøm browser cache

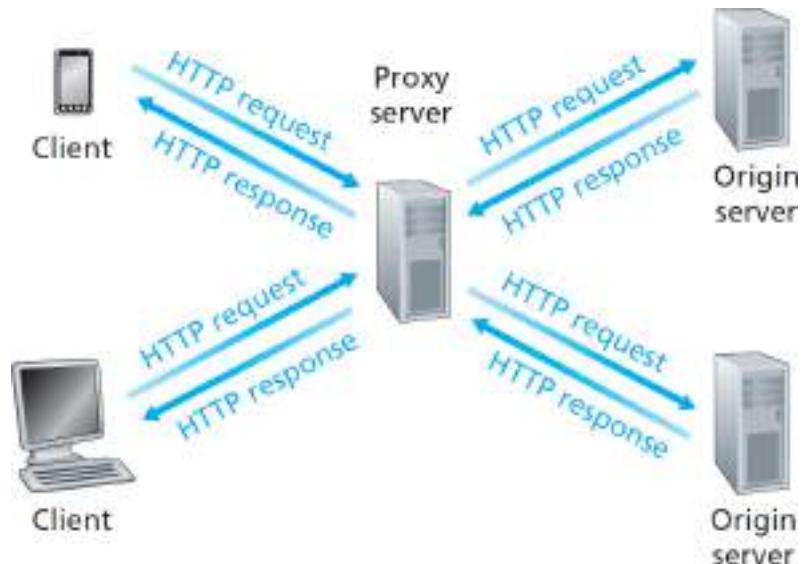
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching>

<https://developer.mozilla.org/en-US/docs/Glossary/cacheable>

# HTTP Proxies

HTTP er designet til at kunne fungere med proxy-servere (mellemliggende servere som er placeret imellem klient og oprindelses-server)

- Web-cache, delt med alle dens klienter
- Load-balancer til en server farm
- Indholds filter
- Anonymizer
- Compression/Encryption



Web-cache (placeret hos ISP)

- Reducerer svar tid til klients
- Reducerer belastning af servers
- Reducerer ekstern trafik brugt af store organisationer / ISP

Proxy og HTTPS

- En proxy er en "man-in-the-middle"
- Et web-fil, der er digitalt-signeret af origin kan ikke uden videre gemmes og sendes fra proxy
- En proxy som web-cache er derfor mindre effektiv ved HTTPS trafik

# Domain Name System

Hvad er formålet med DNS?

Hvordan er det struktureret?

Hvordan laver DNS forespørgsler?

Hvilke (slutbruger) værktøjer findes til DNS?

# Domain Name System

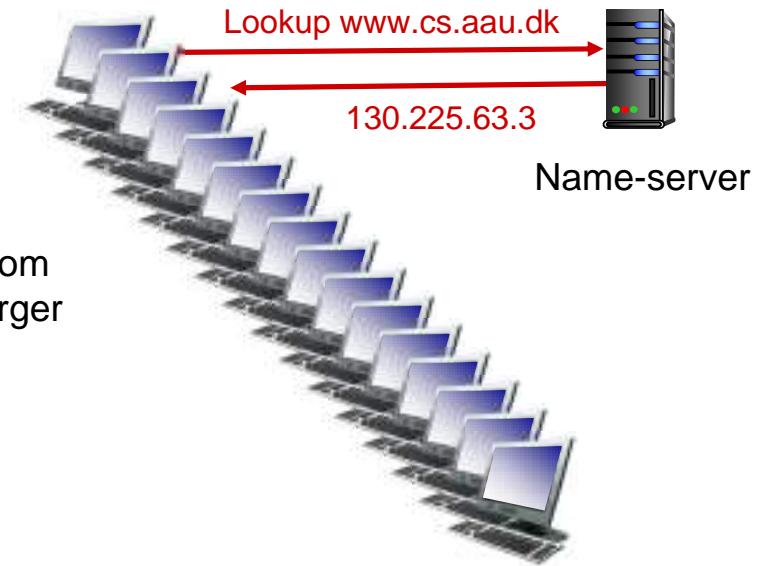
- Navngivning
  - Mennesker foretrækker meningsfylde navne "www.cs.aau.dk"
  - Hosts og routere fortrækker unikke numre (fx 32 bit IP adresse)
- Oversætter hostnavn → IP Adresse: www.cs.aau.dk → 130.225.63.3
- I Internettet barndom:
  - Oversættelserne blev gemt i hver host i en text fil: "/etc/hosts"
  - Ændringer sendt til central organisation på mail
  - Nye versioner blev hentet derfra med ftp.
  - Skalerede ikke, langsom opdatering, inkonsistens

# Idé 2: Én central navne-server?

- Vi sætter en fed server op som vedligeholder database med alle host navne og tilhørende IP adresser
- Nem at forespørge
  - Lookup([www.cs.aau.dk](http://www.cs.aau.dk))
- Kan let opdateres,
  - Update([www.cs.aau.dk=130.22.64.2](http://www.cs.aau.dk=130.22.64.2))
- Flaskehals
  - MANGE klienter (> 1 mia)
  - MANGE forespørgsler (>1 mia / sekund)
- Single-point-of-failure
  - Hvis server bryder ned stopper nettet!!
- Skalerer ikke; Håbløs dårlig ide

## => DNS: Distribueret Hierarkisk Database

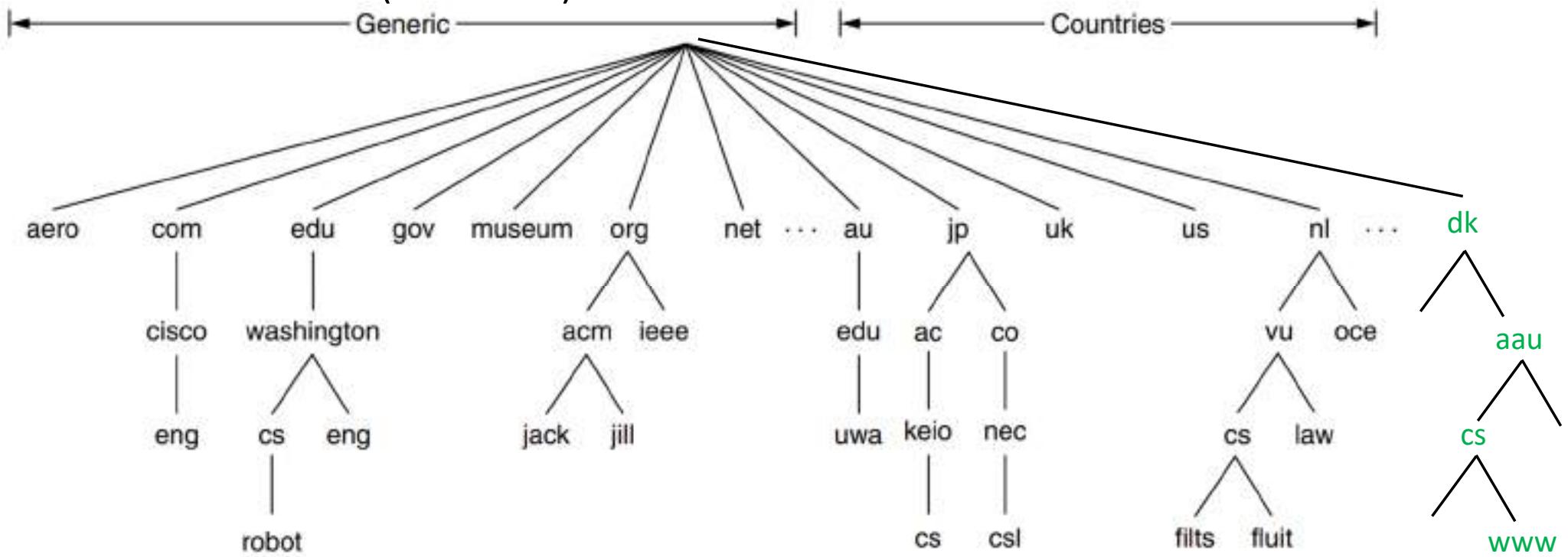
- Fordel belastning på mange servere, organiseret i et hierarki
- Undgå "single-point-of-failure"
- Applikationslagsprotokol: UDP (i enkelte særlige situationer TCP) port 53



navn	værdi
www.cs.aau.dk	130.225.63.3
www.google.com	172.217.4.228
...	
aau.dk	130.225.63.3
gaia.cs.umass.edu	128.119.245.12
dns01-bb.aau.dk	130.225.194.15
...	

Navnerum: [www.cs.aau.dk](http://www.cs.aau.dk)

- Navnene er også organiseret i hierarkier
    - Top domæner (fx .com, .dk) og underdomæner (aau.dk)
    - Et domæne (fx. aau.dk) kontrollerer navnene derunder



# DNS Dataposter

- Ressource Record  
(Navn, Type, Værdi, TTL)
- Type=A
  - Navn er et hostnavn
  - Værdi er en IP adresse
- Type=NS
  - Navn er et domæne (fx aau.dk)
  - Værdi er navnet på den host, som er "autoritative" navne server for domænet)
- Type=CNAME
  - Navn er et alias navn
  - Værdi er navnet på den "rigtige" ("kanoniske") host
- Type=MX
  - Navn er et alias navn
  - Værdi er mailserveren, der er tilknyttet navnet

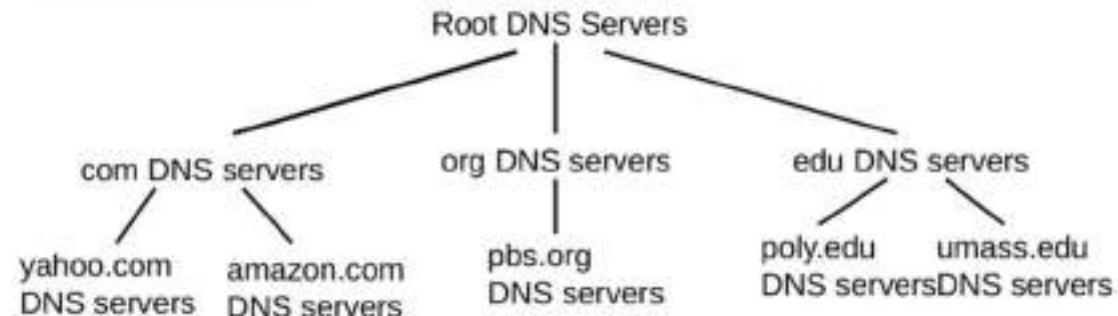
## Eksempler

navn	type	værdi	TTL
www.cs.aau.dk	A	130.225.63.3	3599
www.google.com	A	172.217.4.228	299
www.aau.dk	CNAME	aau.dk	3599
aau.dk	A	130.225.63.3	3599
cs.aau.dk	NS	dns01-bb.aau.dk	3600
dns01-bb.aau.dk	A	130.225.194.15	3600
aau.dk	MX	aau-dk.mail. protection.outlook.com.	3599

TTL=Time-to-live (i sekunder)  
3599 knap 1 time

# DNS Servers

- **Autoritative DNS server:** Hver organisation med offentlige servere har en DNS-server
  - som har til ansvar at hoste DNS poster for dens domæne.
  - Primær og sekundær
- **Top-level-domain DNS:** DNS-Server(e) for hvert TLD
  - Videreformidle forespørgsler til en autoritativ server
- **Root DNS Servers**
  - Videreformidle forespørgsler til en TLD server
- **Lokal DNS server:**
  - En server som klienter bruger for at foretager DNS forespørgsler
  - Opsat af ISP
  - Konfigureret i klient (ofte vha. DHCP)



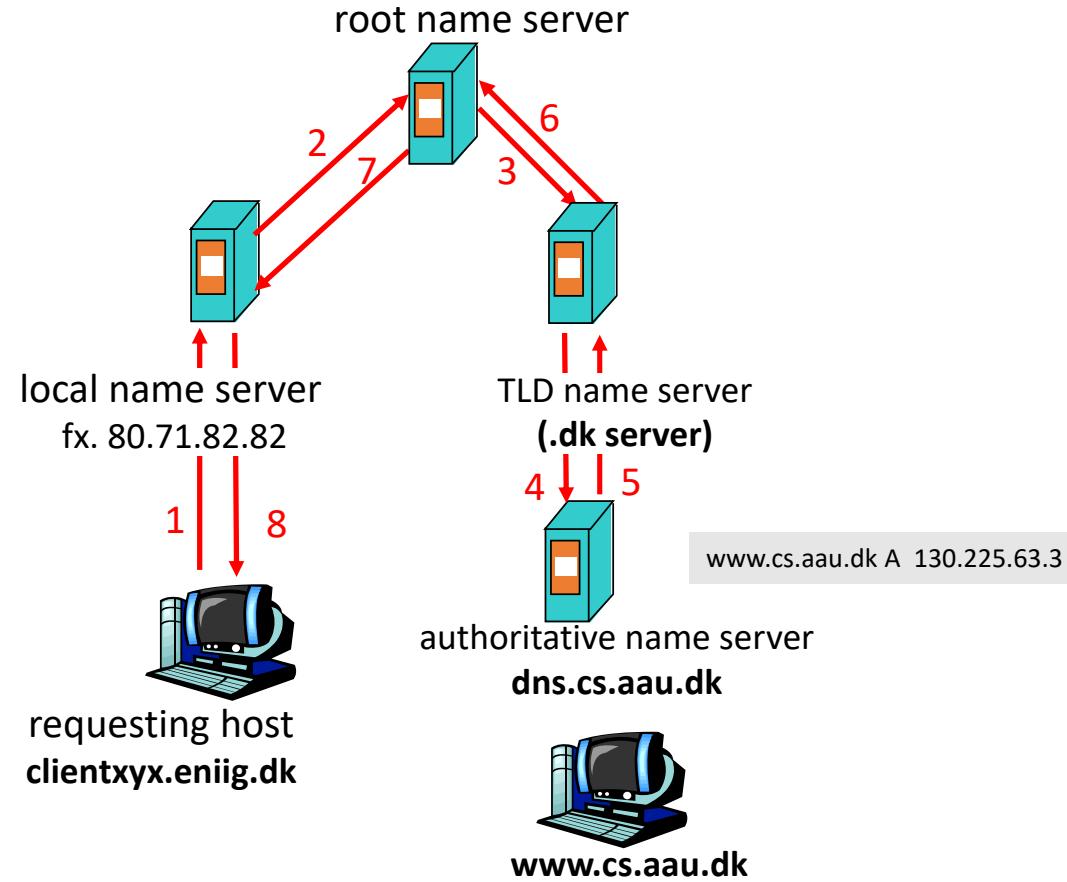
Princip skitse:

1. Klient kontakter lokal DNS server: IP for [www.cs.aau.dk](http://www.cs.aau.dk)?
2. Den lokal kontakter en Root DNS server
3. RootDNS videreformidler til en den ønskede TLD-server
4. TLD videreformidler til den autoritative server
5. Den autoritative server sender svaret tilbage mod klienten 130.225.63.3

# Rekursiv DNS Forespørgsel

Rekursiv DNS forespørgsel (query):

1. Klienten forespørger dens lokale NS
  2. Den lokale forespørger root
  3. Root NS forespørger TLD
  4. TLD forespørger en autoritative
  5. Resultat returneres til TLD
  6. Resultat returneres til root
  7. Resultat returneres til lokal
  8. Resultat returneres til klient
- Root name server:
    - Kender ikke nødvendigvis den autoritative, men henvender sig til en på lavere niveau (fx TLD) som den beder om at løse opgaven
  - Metafor: rekursivt procedurekald
  - Giver høj belastning på serverer på højere niveauer



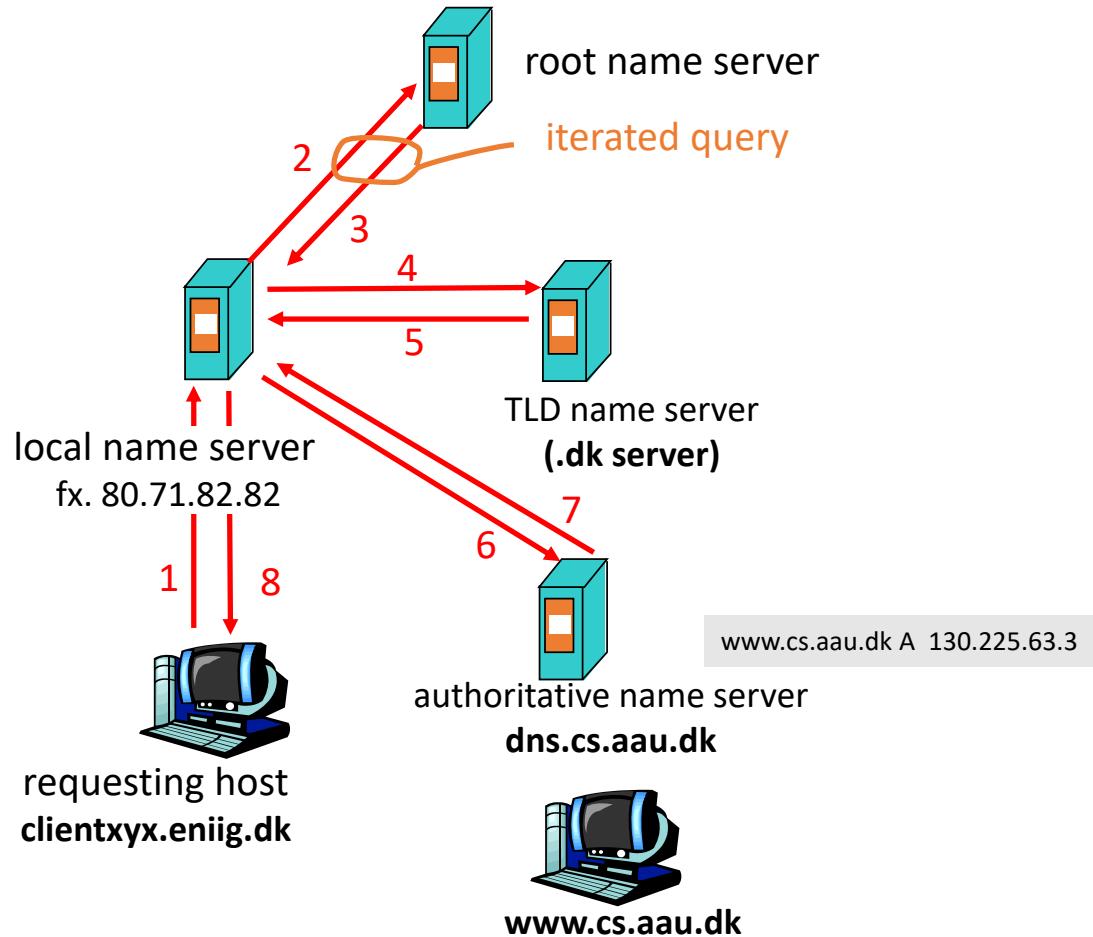
# Iterativ DNS Forespørgsel

Iterativ DNS forespørgsel (query):

1. Klienten forespørger dens lokale NS
2. Den lokale forespørger root
3. Root svare tilbage med navn på TLD
4. Den lokale forsørger TLD
5. TLD svarer med navn på autoritative
6. Den lokale forespørger den autoritative
7. Den autoritative svarer den lokale med resultatet
8. Resultat returneres til klient

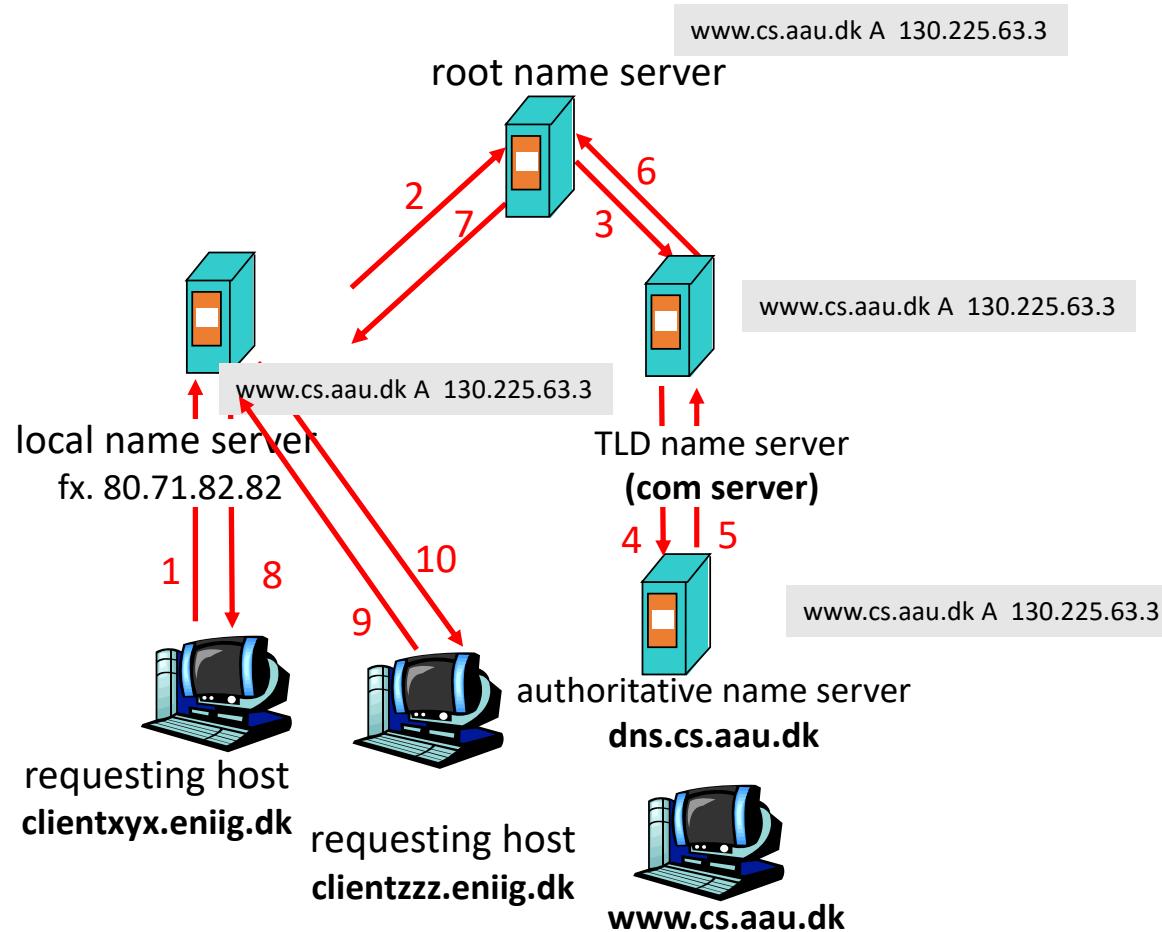
Skridt 3, 5:

- Den kontaktede server svarer med navnet på en server der kan kontaktes”
- “Jeg kender ikke navnet, men prøv denne server”
- Metafor: “while” løkke på den lokale



# DNS caching

- Når en name-server NS får kendskab til en oversættelse (DNS post) gemmes denne lokalt hos NS i en cache (5), (6), (7), (8)
- Når NS næste gang får en ny forespørgsel på en host (9), ser den først efter i cache om den allerede kender svaret selv.
- Hvis ja (10), returneres svaret mod klienten, og den rekursive/iterative forespørgsel stoppes
- Fjerner meget belastning fra Root / TLD servere
- Hvad hvis en host får ny IP-adresse (fx [www.cs.aau.dk](http://www.cs.aau.dk) flyttes til ny server) ??
  - Klinter bliver ved med at få gammel information fra caches
  - Derfor har hver DNS post en **TTL (time-to-live)**: et antal sekunder den må caches inden posten skal fjernes (udløbstid)
  - Det kan gå op mod TTL sekunder før den ny server bliver kendt

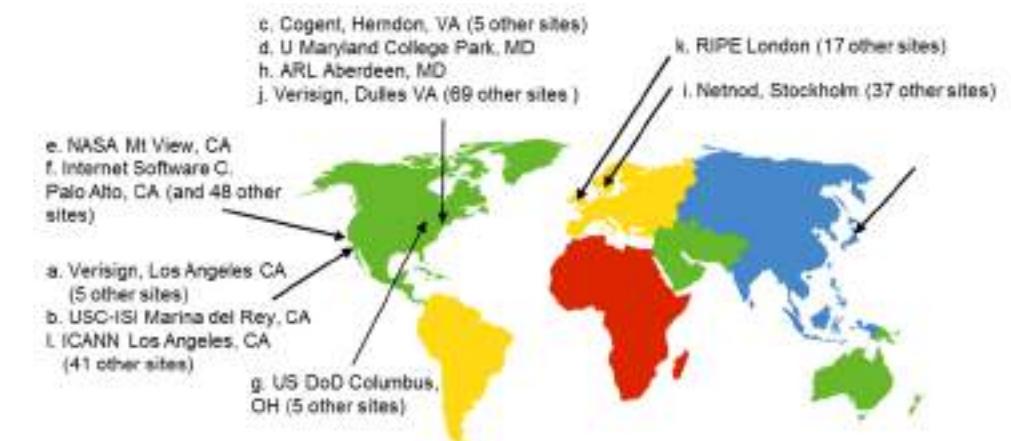


# Root Servers

- 12 globale uafh. organizationer, 1810 servere
  - a.root-server.net
- En kontaktes altid når den lokale ikke kan oversætte navnet
- Centraliseret funktion ⇒ Spørgsmål:
  - Flaskehals?
  - Enkelt kilde til fejl (single point of failure)?
  - Undgås ved replikering og "caching"

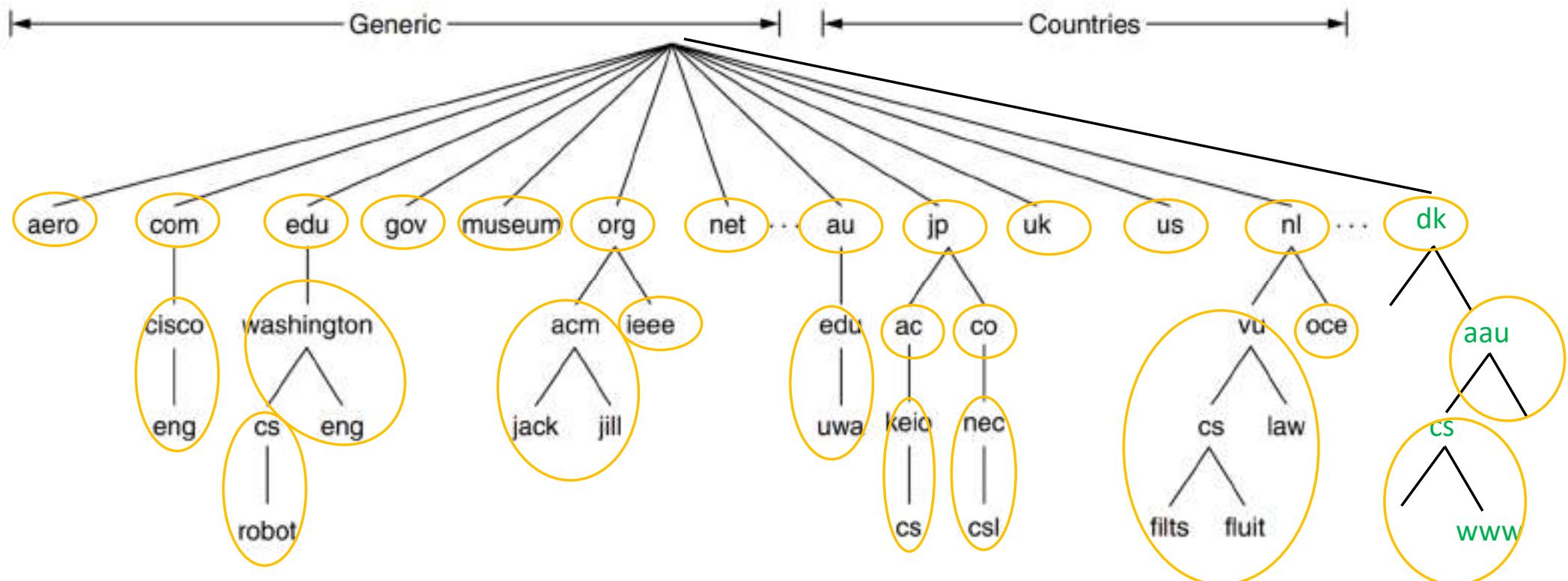
HOSTNAME	IP ADDRESSES	MANAGER
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	VeriSign, Inc.
b.root-servers.net	199.9.14.201, 2001:500:200::b	University of Southern California (ISI)
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10, 2001:500:a8::e	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4, 2001:500:12::d0d	US Department of Defense (NIC)
h.root-servers.net	198.97.190.53, 2001:500:1::53	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	VeriSign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:9f::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

Interaktivt kort: <https://root-servers.org/>



# Navnerum: Zoner

- Zone er en underopdeling med en (eller flere) DNS servere



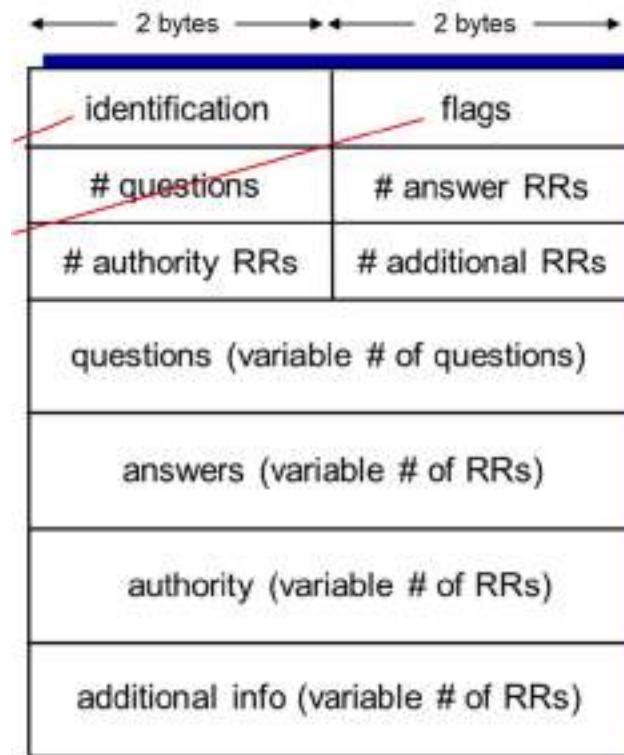
# Opdatering af DNS

- Registrering af nye domæne
  - nsmd.dk (Nørresundby Micro Devices)
  - Køb og registrer domænet hos registraror (<https://www.dk-hostmaster.dk/> , minimums information
    - <nsmd.dk , NS, dns1.nsmd.dk >  
<dns1.nsmd.dk, A , 123.123.123.123>
  - Info bliver sat ind i TLD server for .dk
- Opdatering af poster
  - Administrations-interface hos primær autoritære server
  - Dynamic-DNS: programmeringsmæssig / automatisk opdatering (fx. Ved oprettelse af ny virtuel maskine med en server)

# DNS meddelelser

# DNS meddelelses format:

- Samme format for forespørgsler og svar
  - Et flag (bit) bestemmer hvilket
- Identifikation: 16 bit tal til at matche forespørgsel med svar
- Flag, fx:
  - Forespørgsel eller svar
  - Rekursivt foretrukket
  - Rekursivt tilgængeligt
  - Svaret er autoritativt
- Antal forespørgsler / svar i meddelelsen
- Navn, type felt for forespørgsel
- Poster for authoritative servere
- Yderligere info
- Transporteres på UDP



# DNS meddelelses format: eksempel forespørgsel

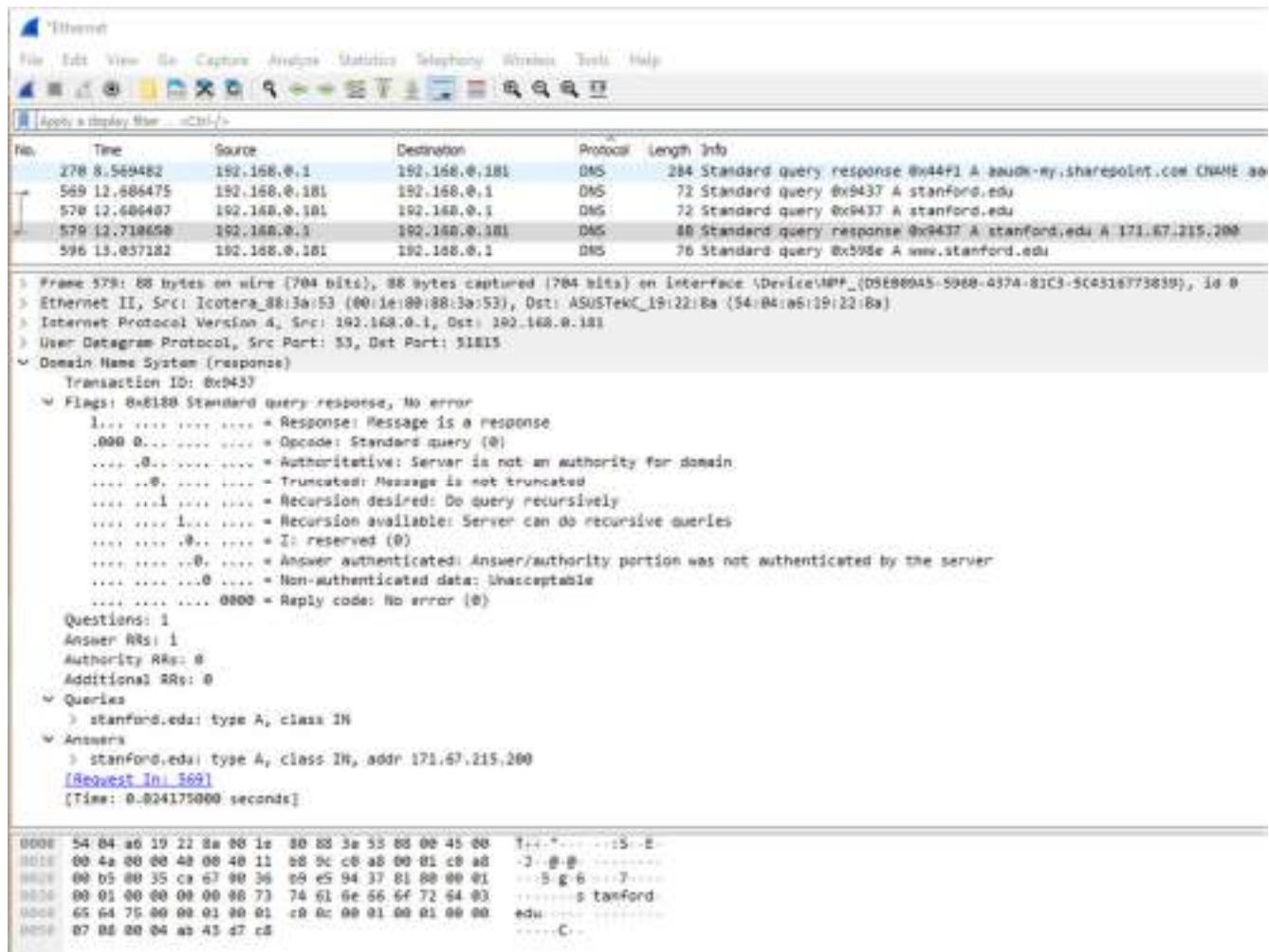
Liste med opsnappede pakker,  
sorteret efter protokoltype

Wireshark dekodet  
info fra UDP payload

De "rå" bits, vist som  
hexadecimale bytes og ascii tekst

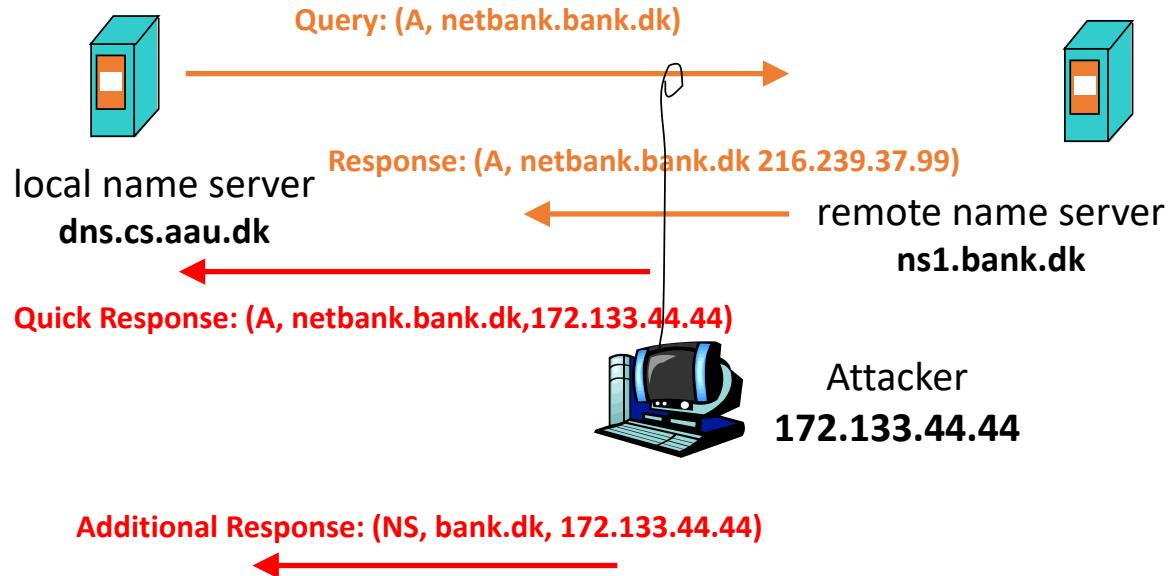
The screenshot shows a Wireshark capture of network traffic on an Ethernet interface. The list view shows several DNS requests and responses. The selected packet is a DNS query from 192.168.0.1 to 192.168.0.181, with a transaction ID of 0x9437. The Wireshark tree view deconstructs the DNS message into fields like Flags, Questions, and Answers. The detailed description pane provides a textual breakdown of the message structure. The bytes pane at the bottom shows the raw hex and ASCII representation of the DNS message, highlighting the query for 'stanford.edu'.

# DNS meddelelses format: eksempel svar



# DNS Sikkerhed

- Spoofing og Cache-forgiftning
  - Forfalske et svar og omdirigere klient til ønsket site
  - Lægge forfalskede oplysninger i en DNS cache
  - Teknisk svært (hastighed)
- Denial-of-service
  - Kan vi nedlægge en name-server ved at bombardere med forespørgsler
  - Nej, caching, og mange replikerede root server
- DNSSEC, en udvidelse baseret på kryptografi



# DNS Værktøjer

# DNS Værktøjer

- Hvad er min lokale name-server
  - Kommer når din klient maskine konfigureres på nettet, oftest via DHCP (Dynamic Host Configuration Protocol) som også giver maskinen en IP adresse
  - Se Kontrol-panel
  - Eller kommando-linie  
`>ipconfig /all`
- Hvilke DNS poster har min maskine cachet?
  - `>ipconfig /displaydns`

Linux / Mac har tilsvarende værktøjer

```
Administrator: Kommandoprompt
Ethernet adapter Ethernet:

Connection-specific DNS Suffix . : My_net
Description : Intel(R) 82579V Gigabit Network Connection
Physical Address. : 54-04-A6-19-22-8A
DHCP Enabled. : Yes
Autoconfiguration Enabled : Yes
Link-local IPv6 Address : fe80::c472:c696:ed6b:3564%17(Preferred)
IPv4 Address. : 192.168.0.181(Preferred)
Subnet Mask : 255.255.255.0
Lease Obtained : 12. marts 2020 19:43:19
Lease Expires : 21. marts 2020 17:11:38
Default Gateway : 192.168.0.1
DHCP Server : 192.168.0.1
DHCPv6 IAID : 257164454
DHCPv6 Client DUID. : 00-01-00-01-21-82-AE-09-54-04-A6-19-22-8A
DNS Servers : 192.168.0.1
NetBIOS over Tcpip. : Enabled
```

```
Administrator: Kommandoprompt
C:\Users\bniel>ipconfig /displaydns

Windows IP Configuration

www.ordbogen.com

Record Name : www.ordbogen.com
Record Type : 1
Time To Live : 11297
Data Length : 4
Section : Answer
A (Host) Record : 91.240.88.18

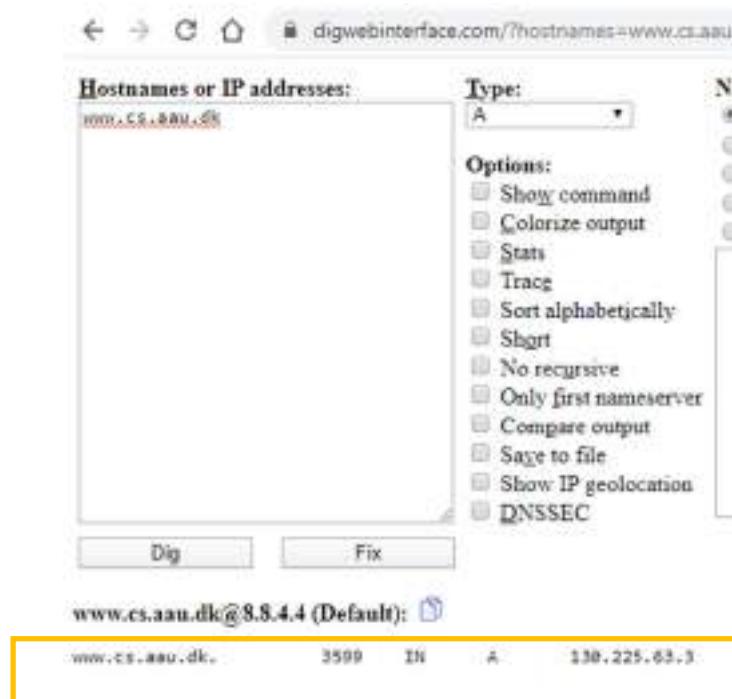
fcmconnection.googleapis.com

```

Normalt en lang liste!

# DNS Værktøjer

- Dig (domain information groper)
  - Kommando linie
  - Web-interface, fx  
<https://www.digwebinterface.com/>
- nslookup



```
C:\Users\bniel>
C:\Users\bniel>nslookup www.cs.aau.dk
Server: UnKnown
Address: 192.168.0.1

Non-authoritative answer:
Name: www.cs.aau.dk
Address: 130.225.63.3

C:\Users\bniel>nslookup 130.225.63.3
Server: UnKnown
Address: 192.168.0.1

Name: vm-ig-www2.portal.aau.dk
Address: 130.225.63.3

C:\Users\bniel>nslookup vm-ig-www2.portal.aau.dk
Server: UnKnown
Address: 192.168.0.1

Non-authoritative answer:
Name: vm-ig-www2.portal.aau.dk
Address: 130.225.63.3

C:\Users\bniel>
```

# DNS værktøjer: whois

- Information om hvem ejer et givet domæne
- En distribueret database vedligeholdt af registratorer
  - <https://whois.icann.org/en/about-whois>
- FX <https://www.whois.com/whois/aau.dk>

aau.dk

Updated 3 days ago

```
Copyright (c) 2002 - 2020 by DK Hostmaster A/S
#
Version: 4.0.2
#
The data in the DK Whois database is provided by DK Hostmaster A/S
For information purposes only, and to assist persons in obtaining
information about or related to a domain name registration record.
We do not guarantee its accuracy. We will reserve the right to remove
access for entities abusing the data, without notice.
#
Any use of this material to target advertising or similar activities
are explicitly forbidden and will be prosecuted. DK Hostmaster A/S
requests to be notified of any such activities or suspicions thereof.

Domain: aau.dk
DNS: aau.dk
Registered: 1997-10-31
Expires: 2023-12-31
Registration period: 5 years
VID: no
DNSSEC: Unsigned delegation, no records
Status: Active

Registrant
Handle: ***N/A***
Name: Aalborg Universitet
Address: Fredrik Bajers Vej 7K
Postalcode: 9220
City: Aalborg Øst
Country: DK
Phone: +4599409940

Nameservers
Hostname: nsns.aau.auc.dk
Hostname: noc.aau.auc.dk
```

# Peer-to-Peer Systemer

Hvad er P2P arkitekturen?

I modsætning til client-server

Hvilke fordele og ulemper er der ved den?

Hvordan virker BitTorrent i principippet?

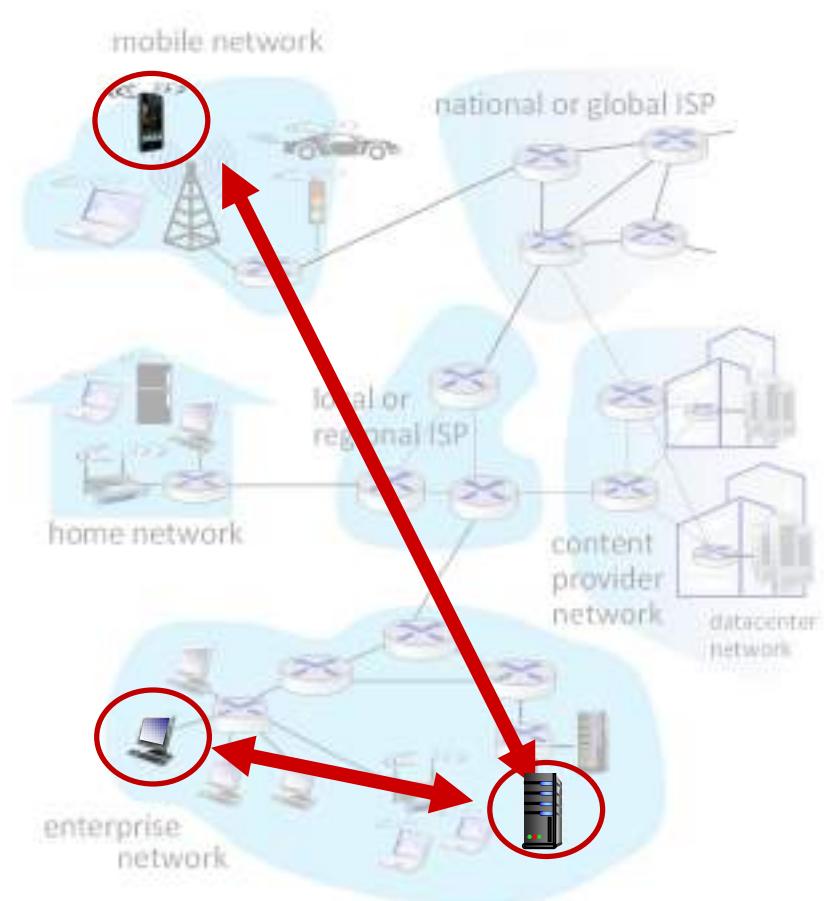
# Klient-server modellen

## server:

- Always-on host
- Permanent IP adresse
- Ofte i data centre, for skalering
- Tilbyder en speciel funktionalitet eller service
- Servicerer mange klienter

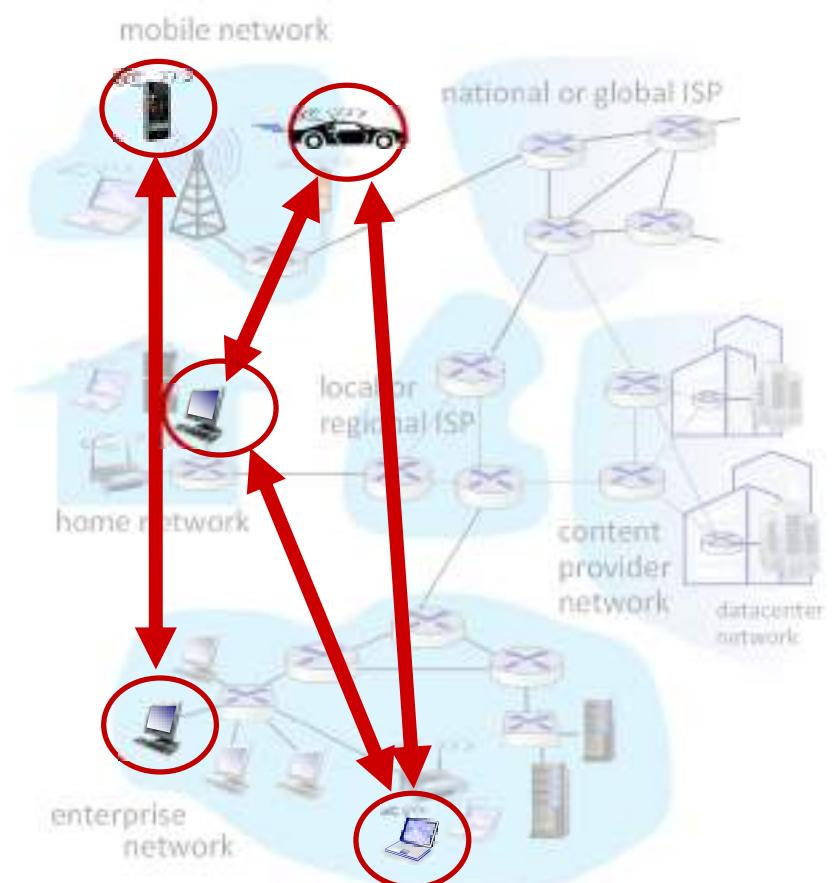
## klienter:

- Tager kontakt til server
- Kommer og går
- Kan have dynamiske IP addresser
- Kommunikerer *ikke* direkte indbyrdes
- Fx.: HTTP, IMAP, FTP



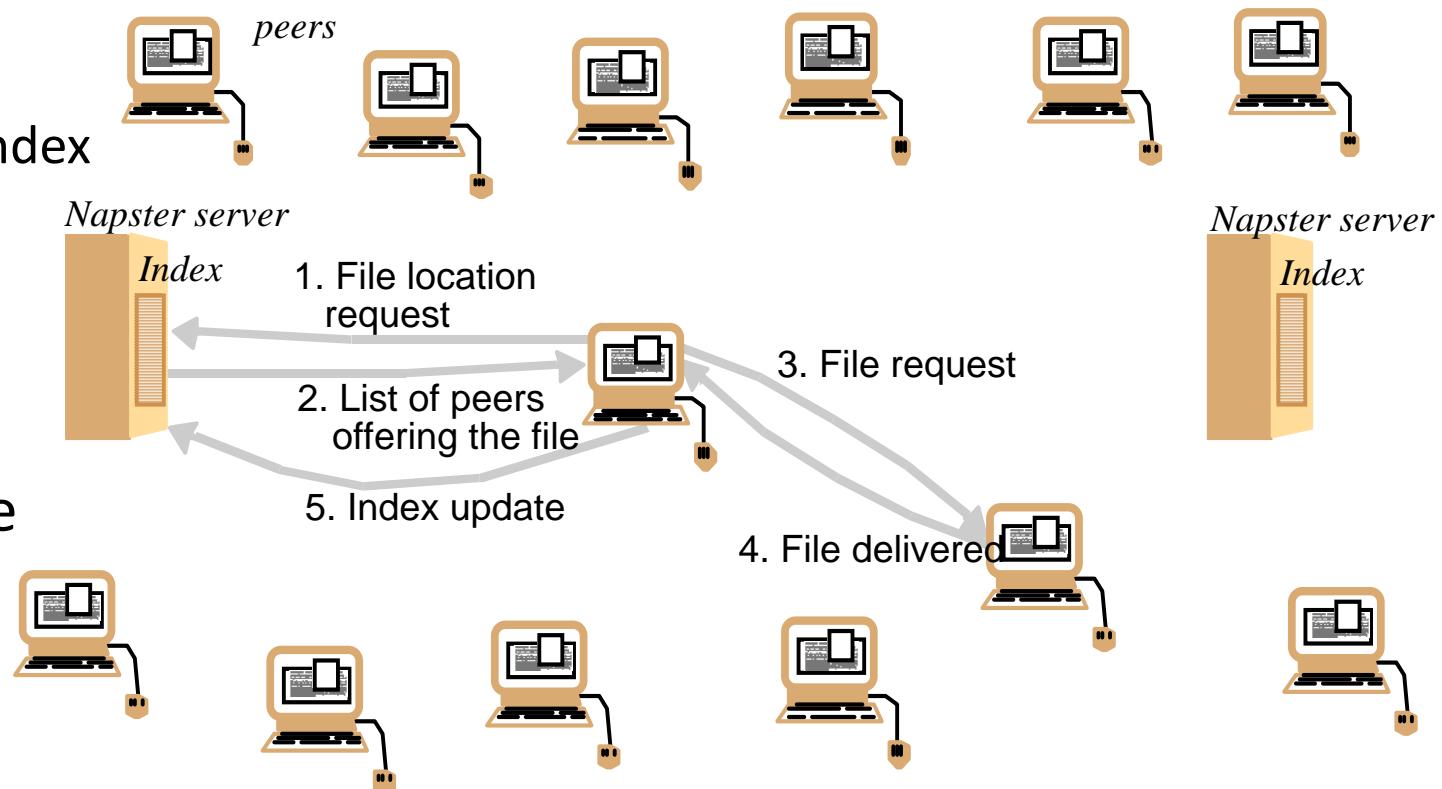
# Peer-to-peer Modellen

- Ingen permanent server
- arbitrære hosts kommunikerer direkte
- peers forespørger service direkte fra andre peers, og tilbyder selv servicen i bytte til andre
  - *Selv-skalerende* – nye peers bringer ny kapacitet til service så vel som efterspørgsel på service
  - Ingen enkelt ejer ⇒ Svært at lukke ned
  - **Oppetid:** ingen centraliseret server, men klienter kommer og går spontant
- Danner et ”overlejret” netværk
  - peers er midlertidigt og løst forbundne, udveksler IP adresser
  - Kompliceret at vedligeholde af dette
- Fx P2P fildeling: Fil distribution (BitTorrent)
  - Streaming (KanKan), VoIP (Skype), TOR, multicast, spil ...
- Varmt forskningsområde i 0’erne
  - Distribuerede Hash Tabeller til hurtig søgning efter data



# Pionererne

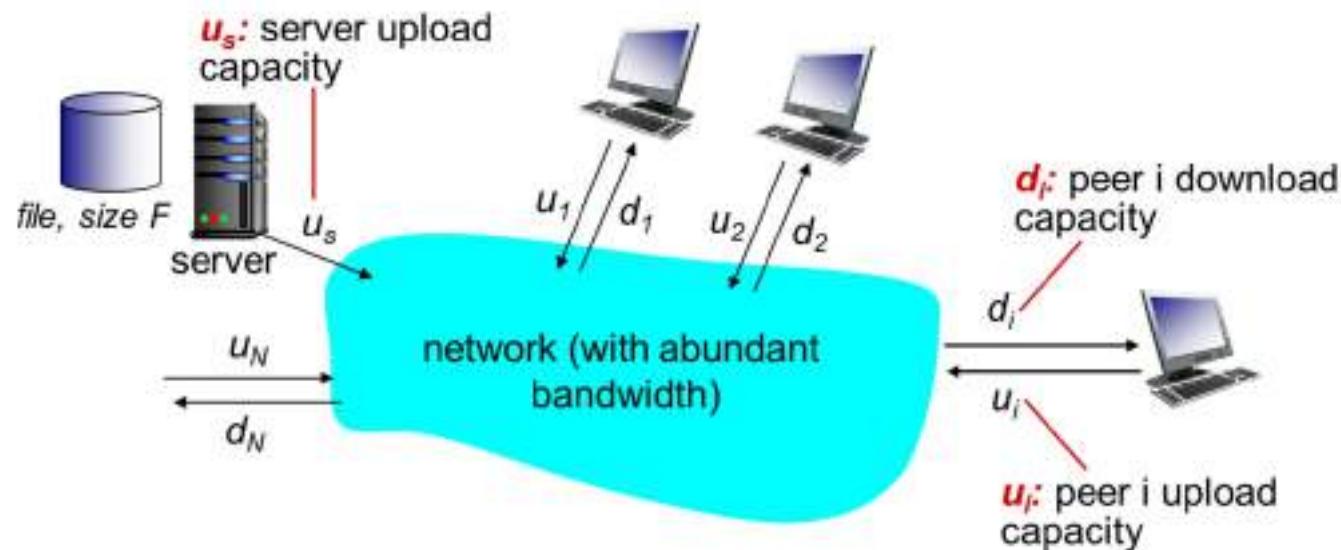
- Napster: 1999-2001
  - Deling af (.mp3) filer
  - Centraliseret, replikeret index
  - Lukket ned efter retskendelse,
  - 24 millioner brugere
- P2P blev synonym med ulovlig deling af materiale beskyttet af ophavsret
- Andre tidlige
  - Gnutella, Freenet



# File Distribution: Client-Server vs P2P

**Spørgsmål:** Hvor lang tid tager det at fordele en fil (af størrelsen  $F$  bits) fra én server til  $N$  peers?

- Vi sammenligner baseret på en overslagsberegning
- Antagelse: flaskehalse er i adgangsnetværket, men linkets fulde kapacitet er tilrådighed



# Fil Fordelingsstid: Client-Server

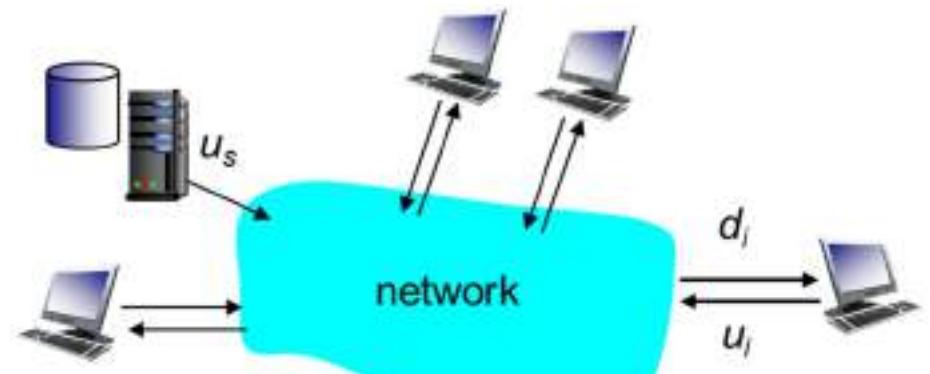
- **Server transmission:** skal sende (upload)  $N$  kopier af filen med  $F$  bits:

- Tid ialt:  $NF$  bits med  $u_s$  bps:  $\frac{NF}{u_s}$  sek.

- **Klient:** hver klient skal downloade en kopi

- $d_{min}$  = download-rate fra klienten med mindste hastighed (bps)

- Denne klient bestemmer samlet download tid:  $\frac{F}{d_{min}}$  sek.



- **Samlet fordelingstid**  $D_{cs} \geq \max \left\{ \frac{NF}{u_s}, \frac{F}{d_{min}} \right\}$

- Bemærk: behovet stiger lineært, med  $N$  men serverens kapacitet er konstant

- Med stort  $N$  bliver dette den dominerende faktor

# Fil Fordelingsstid : P2P

- **server transmission:** skal uploadé mindst en kopi

- Tid at sende en kopi:  $\frac{F}{u_s}$

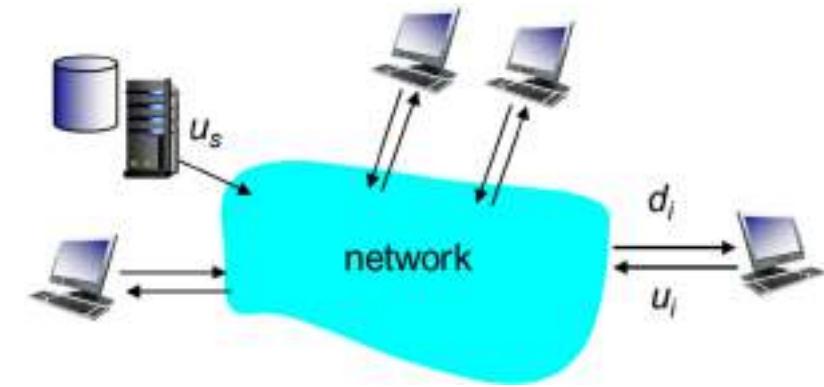
- Hver klient skal downloade en kopi

- Tid til at downloade en kopi hos langsomste:  $\frac{F}{d_{min}}$

- Systemet samlet

- Klienter har et download behov  $NF$  bits

- Upload kapacitet:  $u_s + \sum u_i$



- **Samlet** fordelingstid  $D_{p2p} \geq \max \left\{ \frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum u_i} \right\}$

- Bemærk at både behov og kapacitet stiger lineært i antallet af klienter

# Client-Server vs P2P: Eksempel

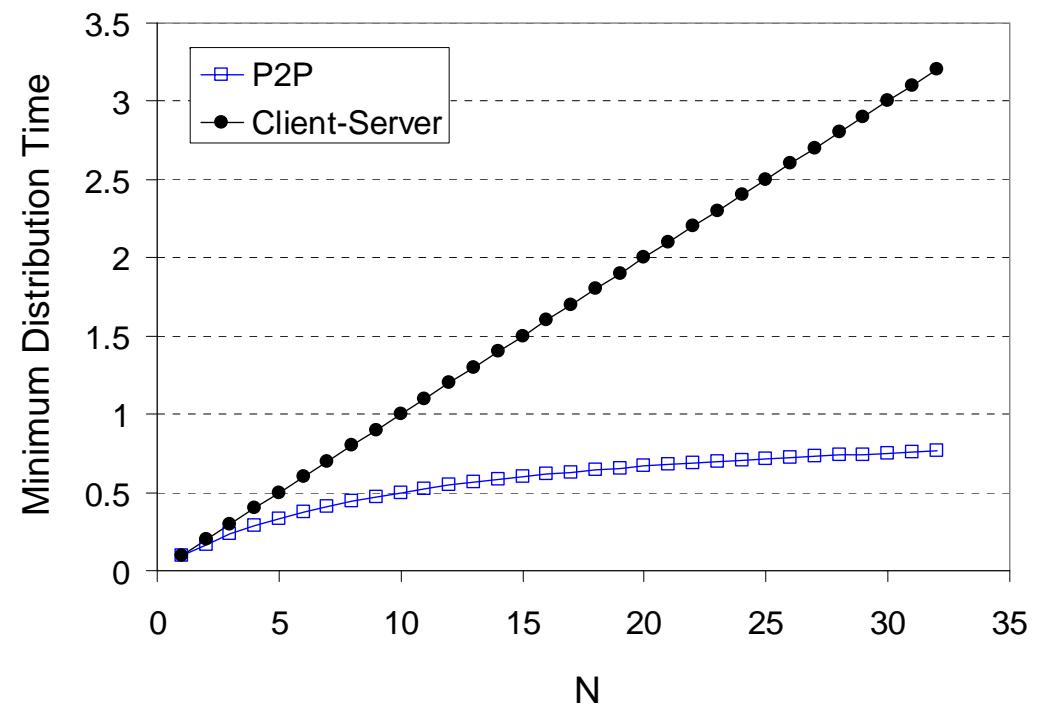
- Hvordan skalerer systemet?

- *Plot af afgørende termer*

$$\text{Server-baseret: } \frac{NF}{u_s}$$

vs.

$$\text{P2P baseret: } \frac{NF}{u_s + \sum u_i}$$



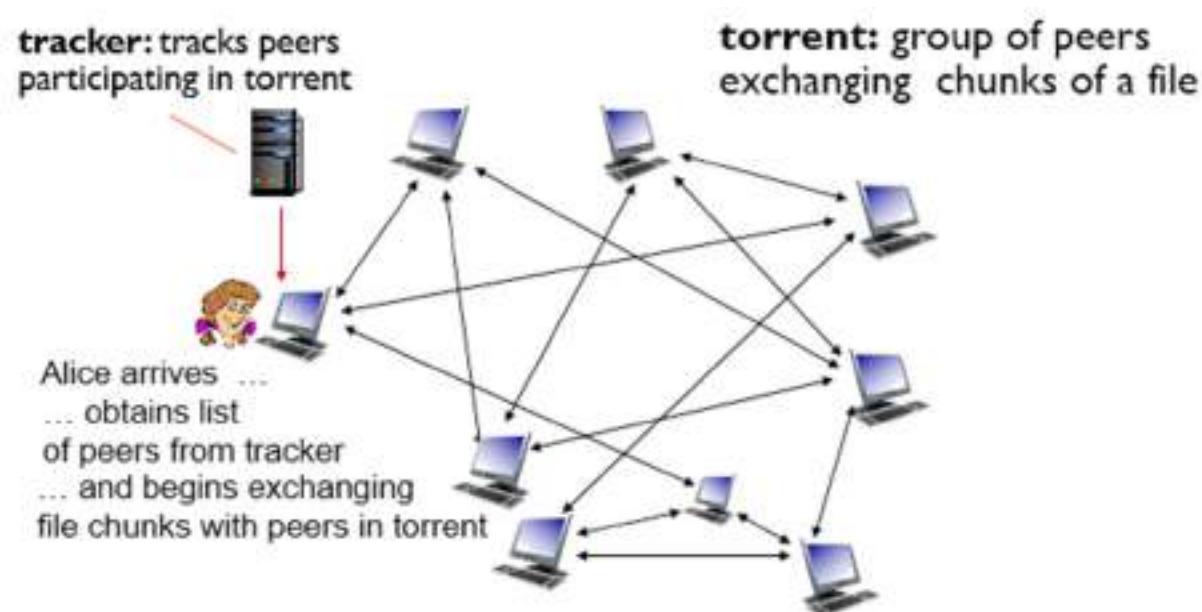
# Problemstillinger ved P2P netværk

- Klienter er meget dynamiske: melder sig ofte til og af systemet
  - Afmelding: Resourcer og data forsvinder fra systemet
  - Systemet skal stadig være nogenlunde velfungerende
- Klienter vil gerne nyde; i mindre grad yde
  - Hvordan laver vi en “fair” og jævn belastning af klienterne
- Hvordan kan klienter finde hinanden og “ønskede filer” uden servere?
  - Brug andre web-sider til at annoncere tilstedeværelse

Netværkstekniske problemstilling da fleste klienter ikke har en offentlig fast IP (“NAT”)

# P2P Fil Distribution: BitTorrent

- **Torrent** ("rivende strøm"): en gruppe af peers som udveksler en given fil
  - fil opdelt i portioner af 256 Kbytes: chunks
  - peers som deltager i en torrent sender og modtager chunks af filen
- **Tracker**: host, der vedligeholder liste med deltagende peers
- Ny "tom" peer melder sig hos tracker
  - Får tilsendt en liste med peers, den kan forbinde sig til (en delmængde af)
    - Peer kan vælge nye peers
  - Mens den downloader en chunk, uploads tidligere downloadede chunks
  - Vil gradvis (forhåbentligt) akkumulere alle chunks
- Mange lærerige aspekter, som vi ikke når



# Opgaverne idag

- Review: Har man forstået grundlæggende begreber
  - Klient og server ”roller”, cookies, DNS forespørgsler
- Øvelser: Kan man anvende dem i nye eksempler?
  - Undersøgelse af HTTP header
  - Tidsforskelt mellem forskellige HTTP forbindelses-strategier
  - HTTP cache (conditional get)
  - P2P download tider
- Praktiske: Kan anvende netværksværktøjerne
  - Wireshark: opsnappe og undersøge de udvekslede DNS meddelelser

**SLUT**

# Internettværk og Web-programmering

## Transport laget

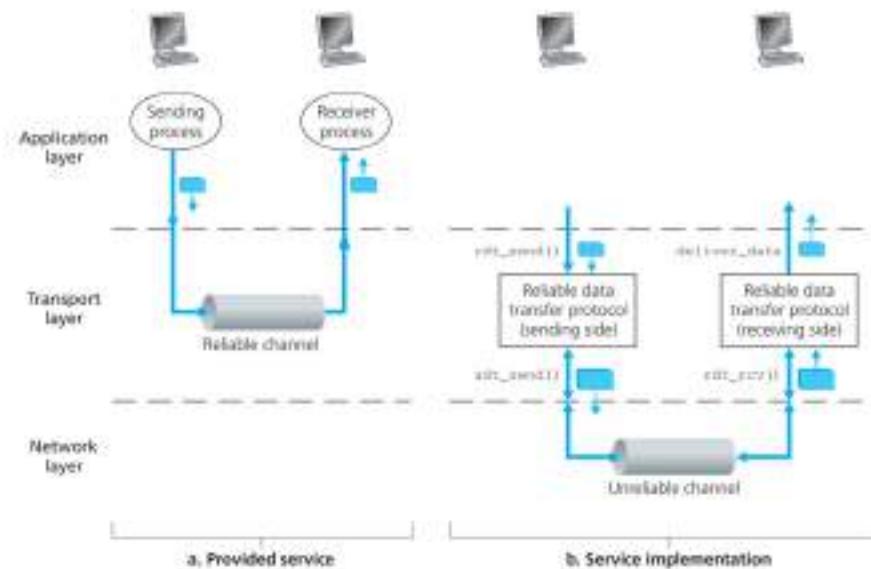
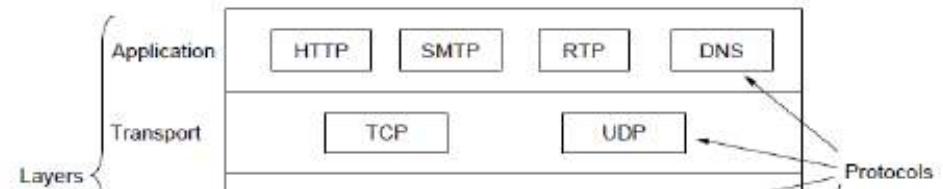
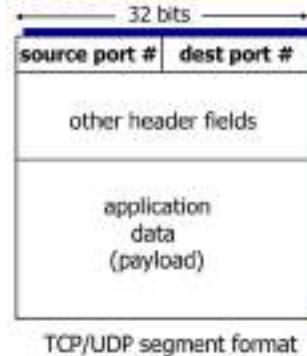
Forelæsning 11  
Brian Nielsen

Distributed, Embedded, Intelligent Systems



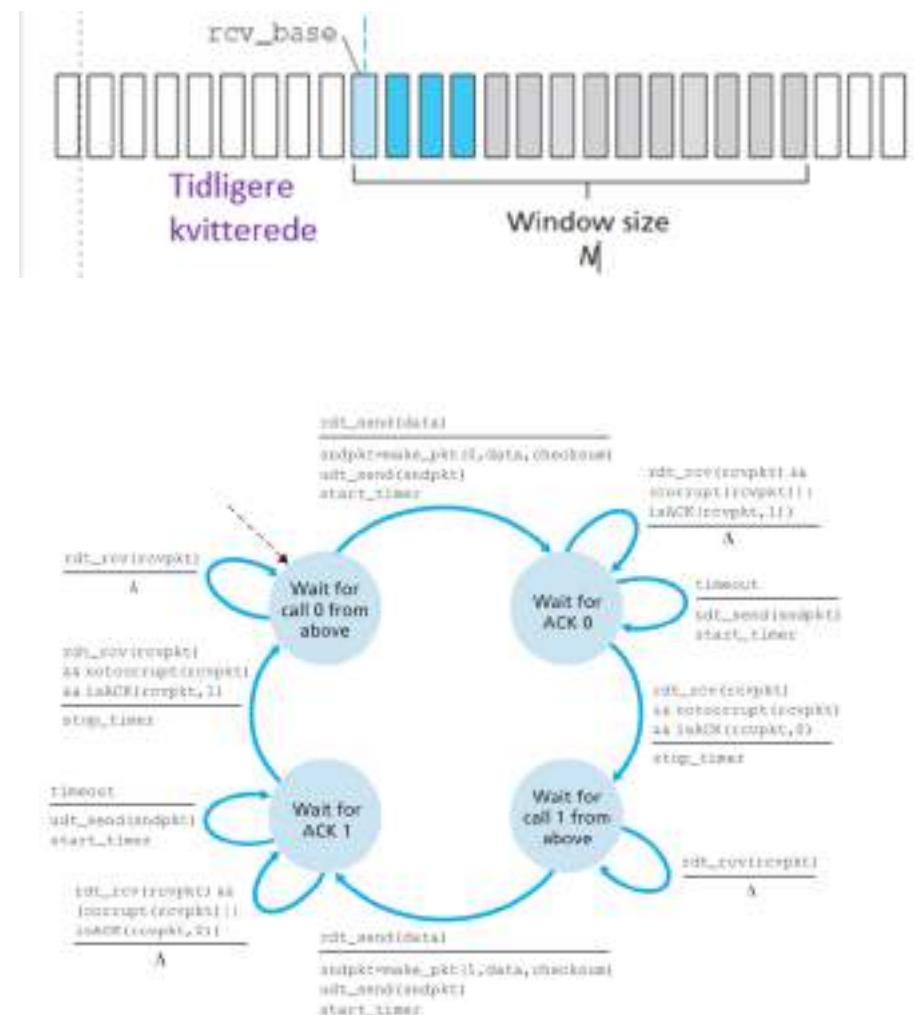
# Læringsmål 1

- Forstå opgaver, services, og protokoller på transport-laget
  - TCP og UDP
- Hvordan disse opgaver løses
  - Bindeleddet mellem applikationslaget og transportlaget:
    - "primitiver" (operationer)
    - "sockets" og "demultiplexing"
  - Oprettelse af TCP forbindelser



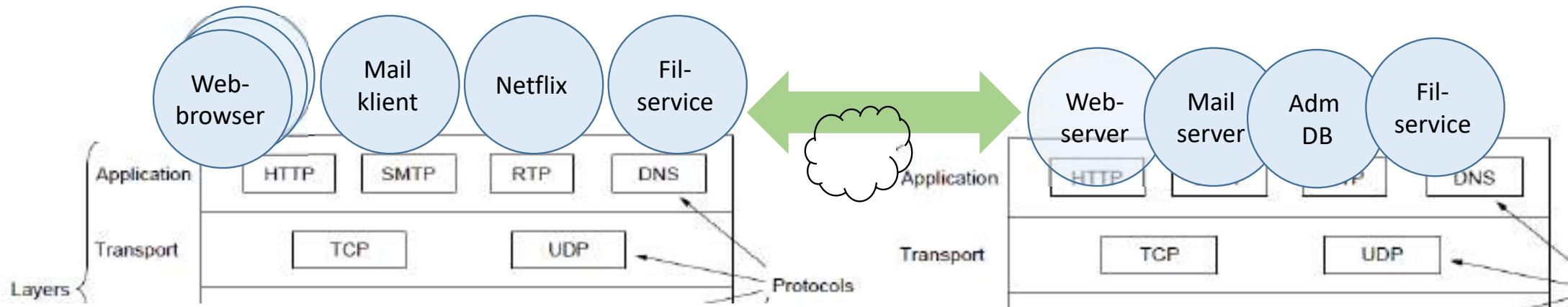
# Læringsmål 2

- Hvordan kan en Host A sende en strøm af data til Host B tabsfrit.
  - Hvordan laver man pålidelig end-to-end kommunikation?
  - Detektion af tabte beskeder?
  - Retransmission
  - Brug af sekvensnumre?
  - (go-Back-N, Selektiv Repeat)
  - **"Sliding windows"**
- Beskrivelse af protokol som tilstandsmaskine, (illustration af scenarier via sekvensdiagrammer)



# Transportlags protokoller

# Transportlaget i TCP/IP modellen



## Transportlags opgaver

- **END-TO-END** transport: logisk forbindelse imellem processer (proces=kørende program)
- Opdeling af data i mindre portioner (segmenter), der kan fremsendes af netværkslaget
  - Segmentation and re-assembly
- Fremsendelse af data til korrekte modtager proces
  - Multiplexing and demultiplexing
- **Fejl-korrektion**

## TCP (transmission control protocol)

- Forbindelses-orienteret, og
- Pålidelig, ordnet, byte-stream service
- Flow- og congestion-kontrol

## UDP (user datagram protocol)

- Forbindelsesløs, og
- Best-effort datagram service
  - Tabte- og omordnede segmenter

# En pålidelig transport protokol

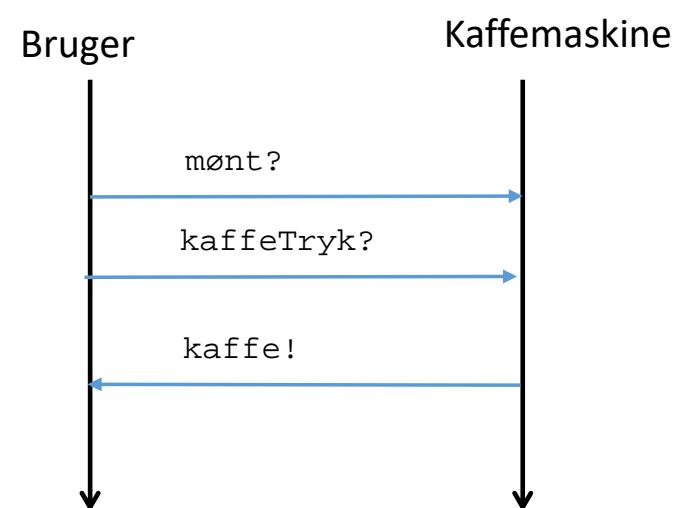
Hvordan sikrer man at data kan nå frem med pakketab?

Hvordan modelleres og specificeres en protokol?

Hvordan implementeres en protokol?

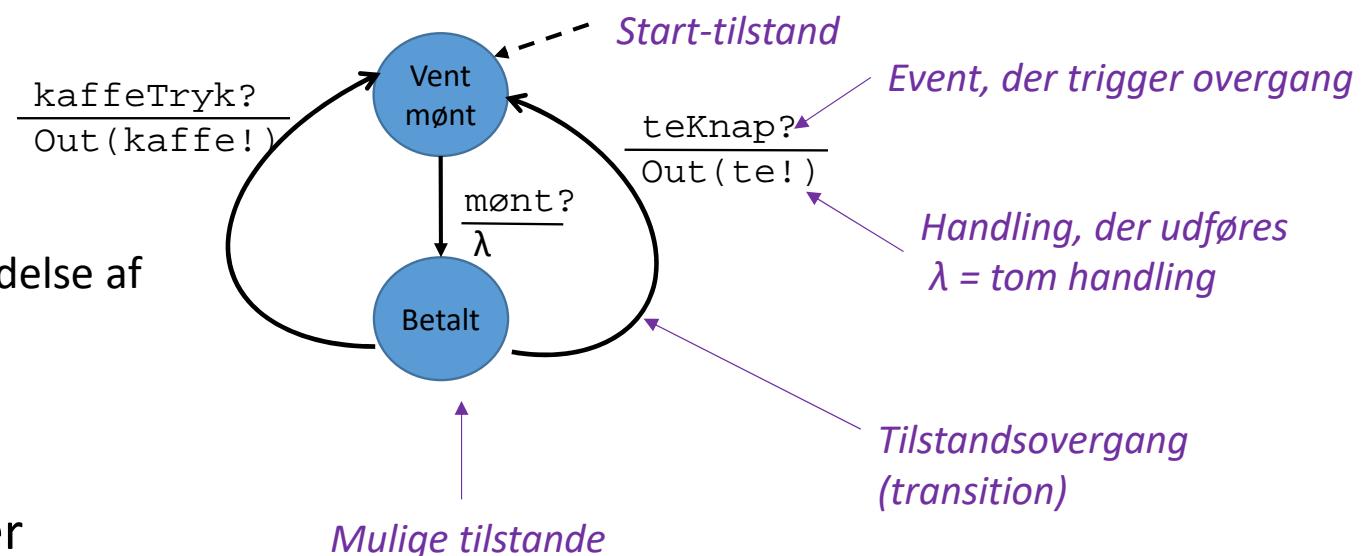
# Sekvens diagram

- Generel metode til at beskrive et systems ”opførsel”
  - Interaktion mellem enheder i systemet
    - Viser ét muligt (ud af flere) interaktions-scenarie mellem komponenter

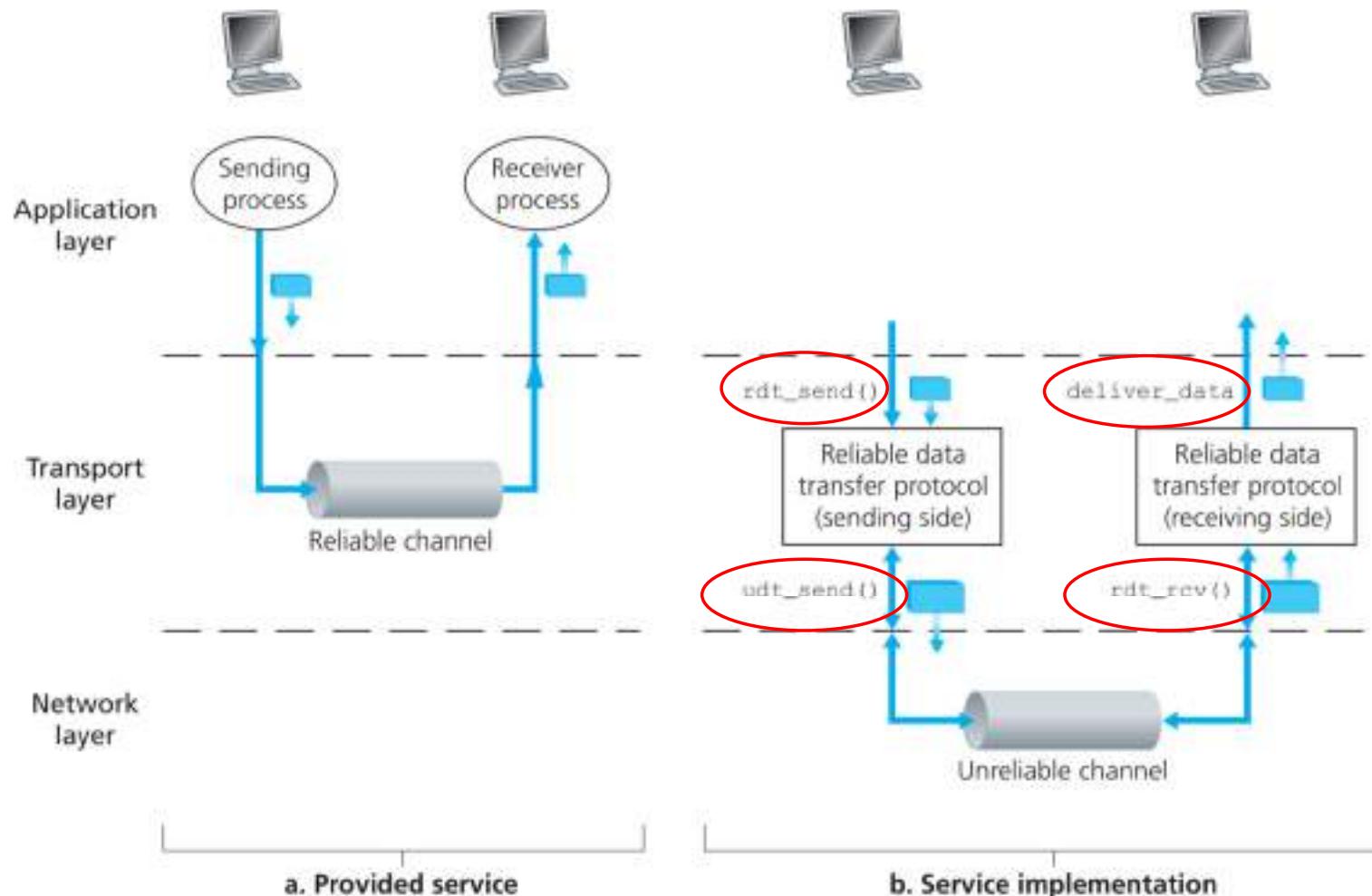


# Tilstandsmaskiner

- Generel metode til at modellere en system komponents "opførsel"
  - Dens tilstande og tilstandsskift
  - Dens interaktion m. andre komponenter
- Forskellige varianter
  - "Automater"
  - Moore-maskiner
  - Mealy maskiner
- Anwendelser i andre fag
  - OO-Design
  - Syntax & Semantik (fx, genkendelse af reg-exp)
  - Compilere
  - Her protokol modellering
  - Formel Verifikation
- En graf, hvor knuder og kanter tillægges en tilstandsfortolkning

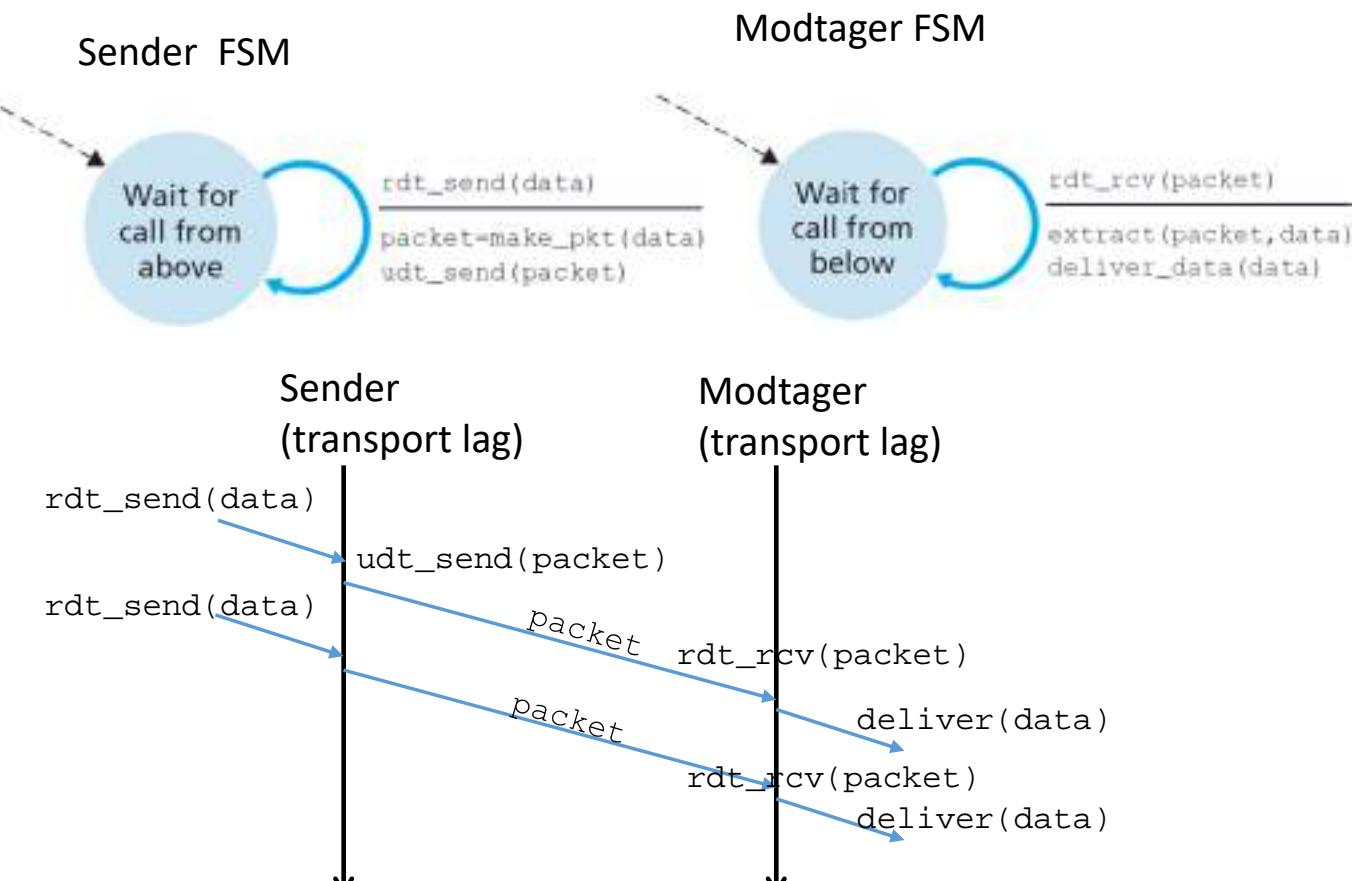


# Abstraktion og Implementation



# Pålidelig kanal: rdt1.0

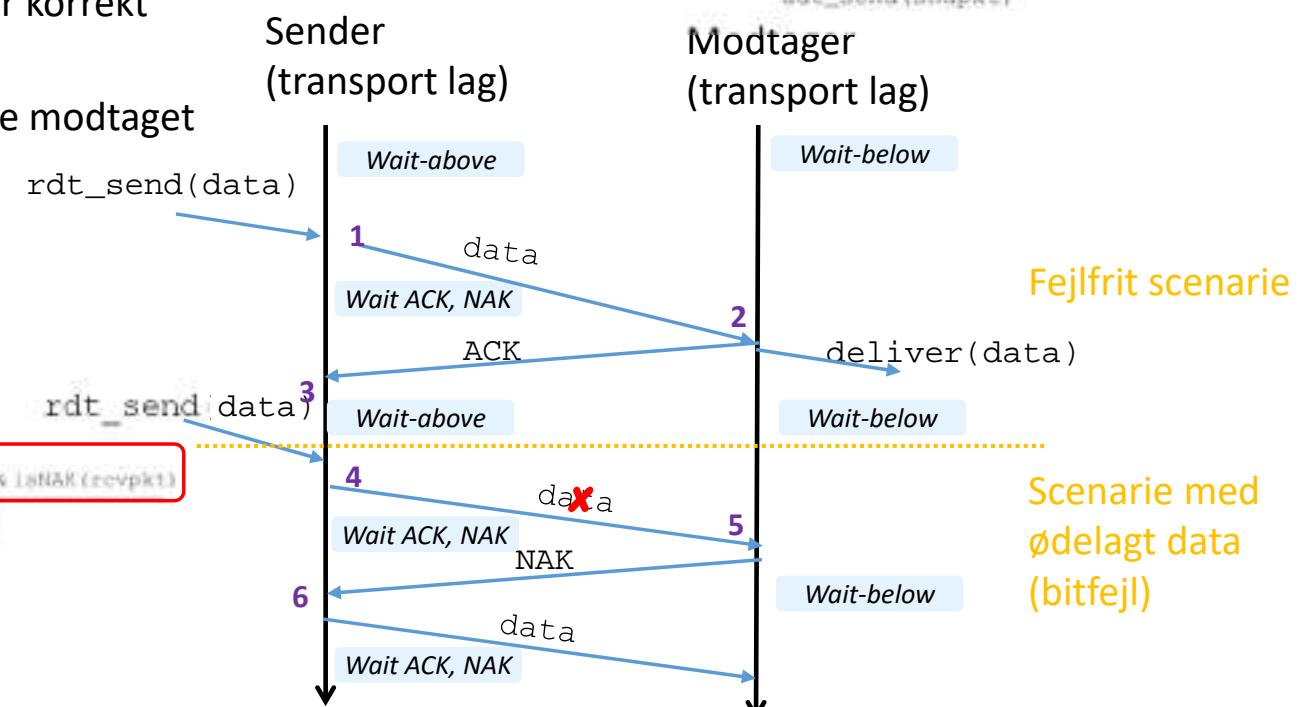
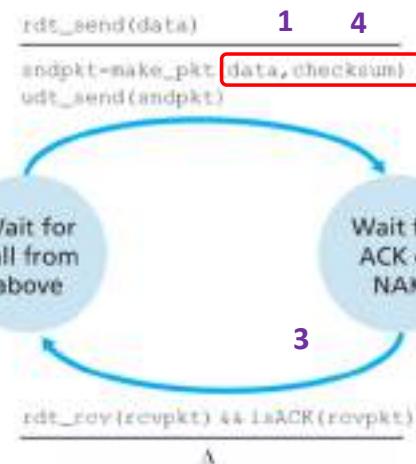
- Antagelser:
  - Kanalen er tabsfri
  - Pakker modtages fejlfri: Ingen bit-fejl
  - Bevarer rækkefølge
- Trivial!
- Sender og modtager følger hver sin tilstandsmaskine



Sekvensdiagram (message sequence chart) viser ét muligt scenarie

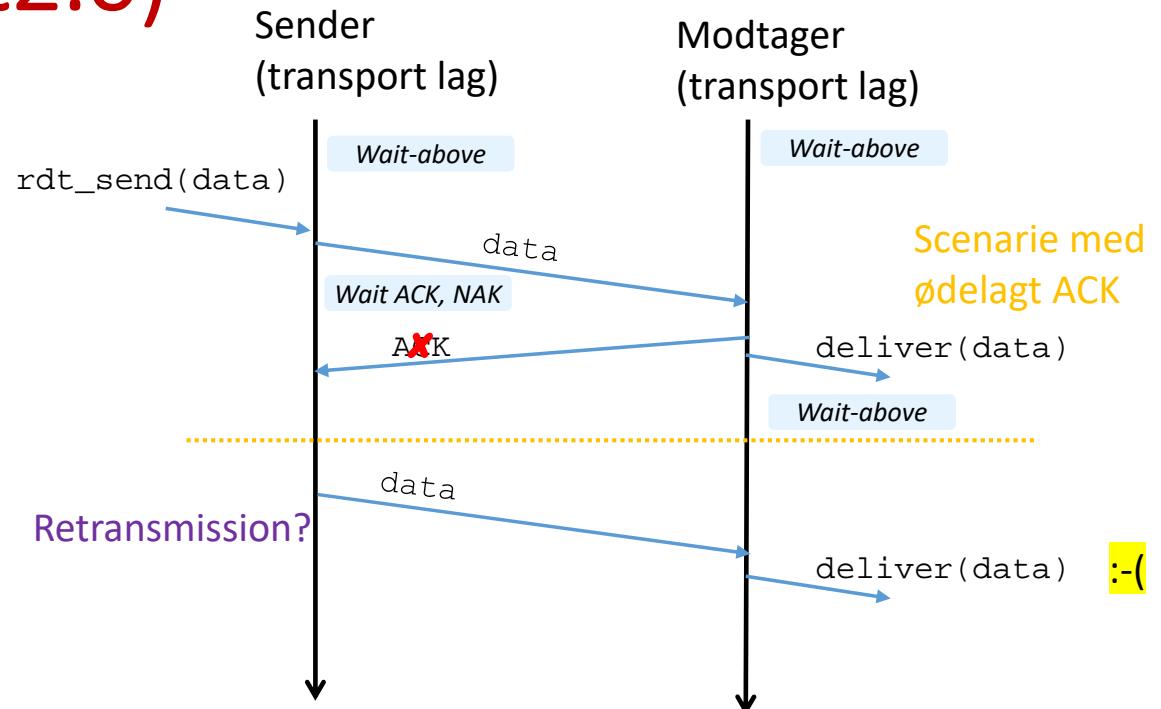
# Kanal med bitfejl (rdt2.0)

- Antagelser:
  - Bitfejl kan forekomme (pakken kan ikke forstås)
  - Kan detekteres vha. checksum
  - Bevarer rækkefølge
- **ACK** (acknowledge): meddelelsen kvitterer for korrekt modtagelse
- **NAK** (negativ acknowledge): meddelelsen ikke modtaget korrekt  $\Rightarrow$  retransmission

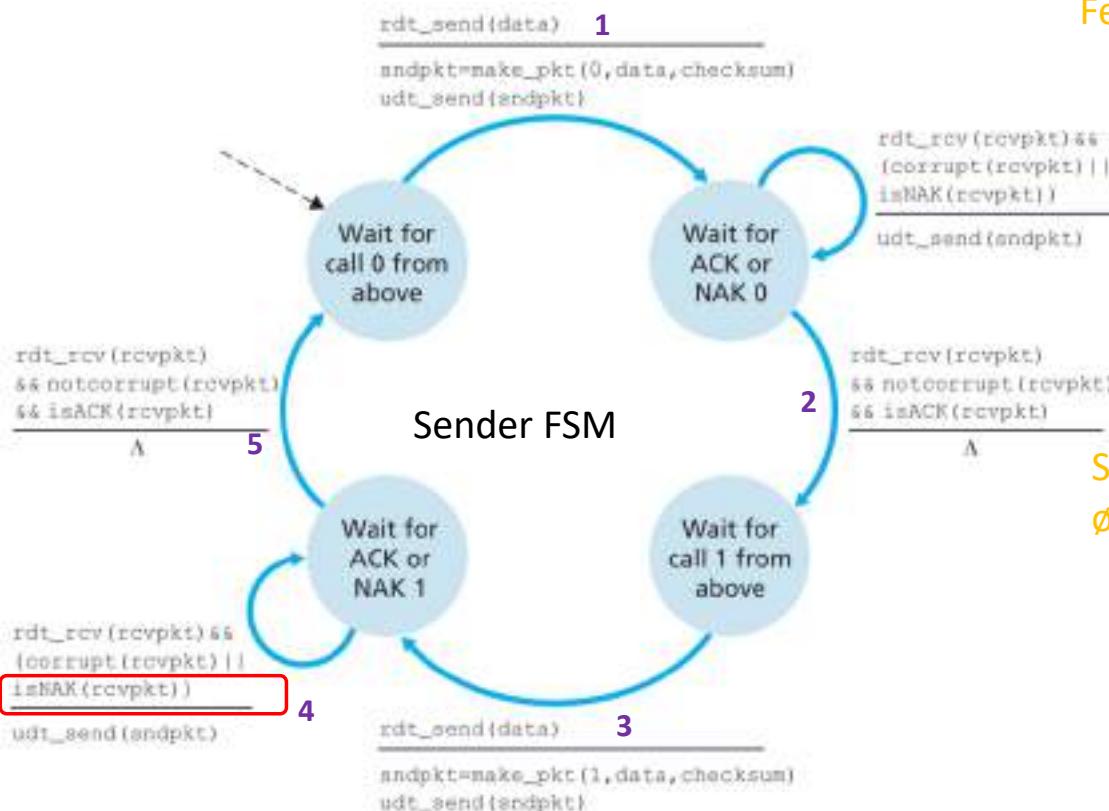


# Kanal med bitfejl (rdt2.0)

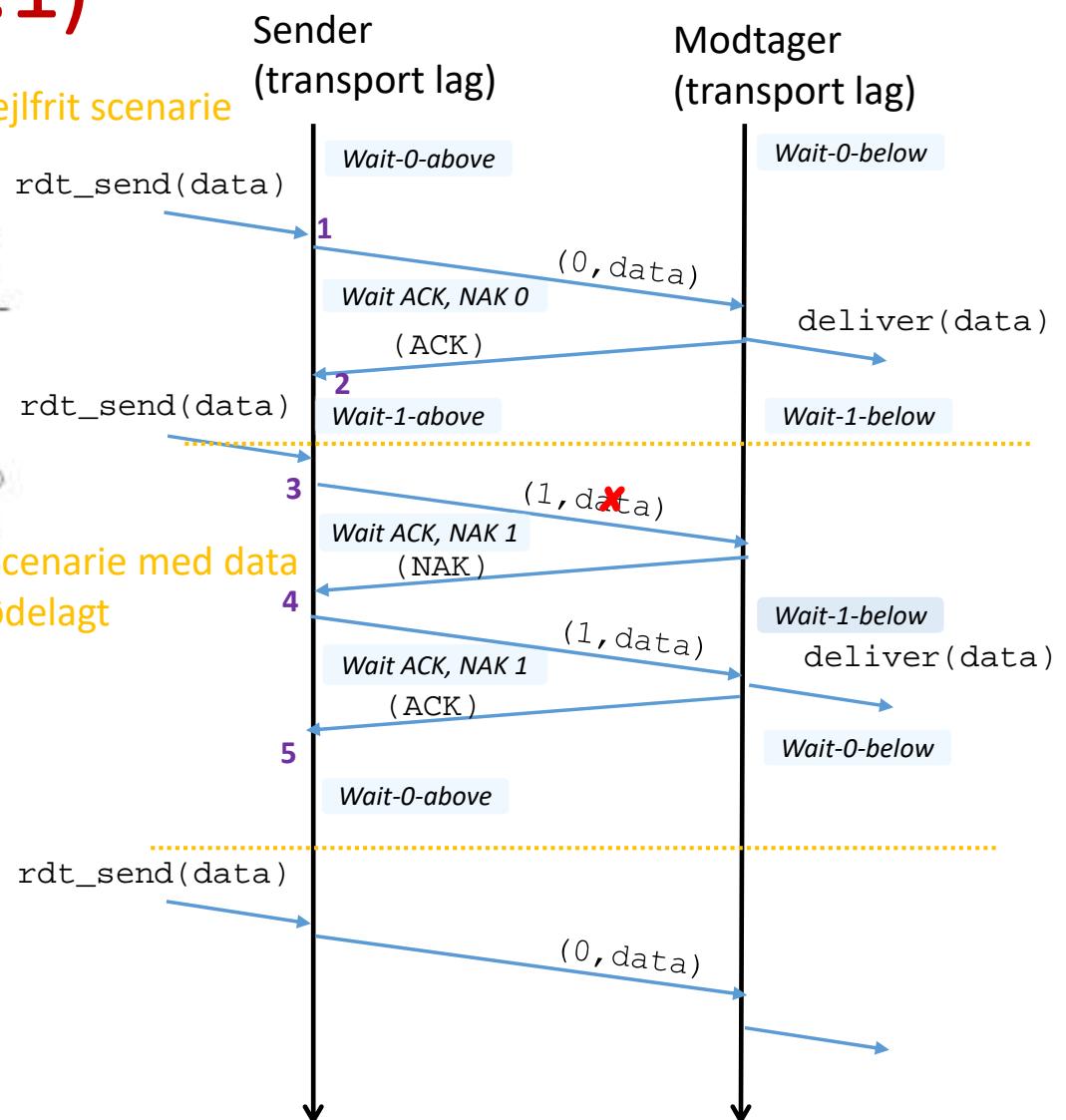
- Antagelser:
  - Bitfejl kan forekomme (pakken kan ikke forstås)
  - Kan detekteres vha. checksum
  - Bevarer rækkefølge
- **ACK/NAK** (kan selvfølgelig også rammes af bit-fejl og tabes)
  - ACK eller NAK?
  - Hænger fast i Wait ACK, NAK
  - Rtd2.0 duer ikke
- Hmm, sender må formode det værste (NAK) og gensende pakken?
  - Modtager leverer så samme data 2 gange (ved ikke om der er ny data eller gendsendt data).
  - ⇒ Brug sekvensnumre



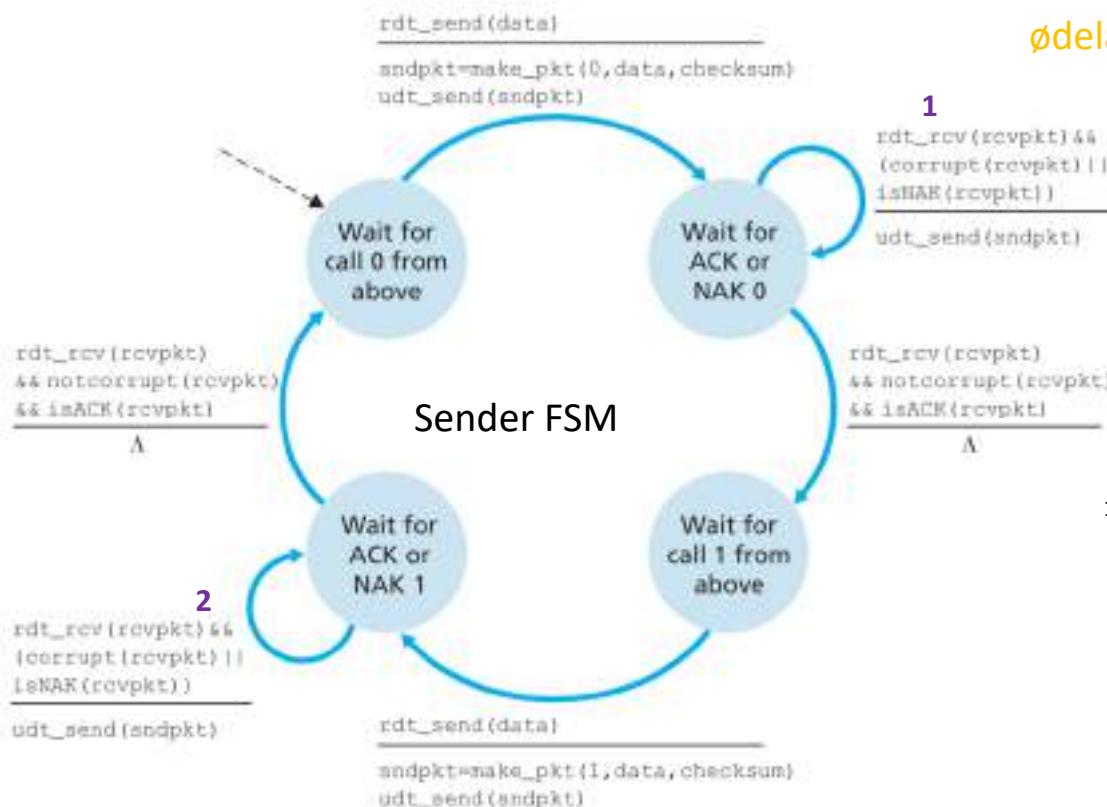
# Sekvens-numre 1 (rdt2.1)



Fejlfrigt scenarie



# Sekvens-numre 2 (rdt2.1)



Scenarie med ødelagt ack

rdt\_send(data)

1  
rdt\_recv(rcvpkt) &&  
(corrupt(rcvpkt) ||  
isNAK(rcvpkt))  
udt\_send(sndpkt)

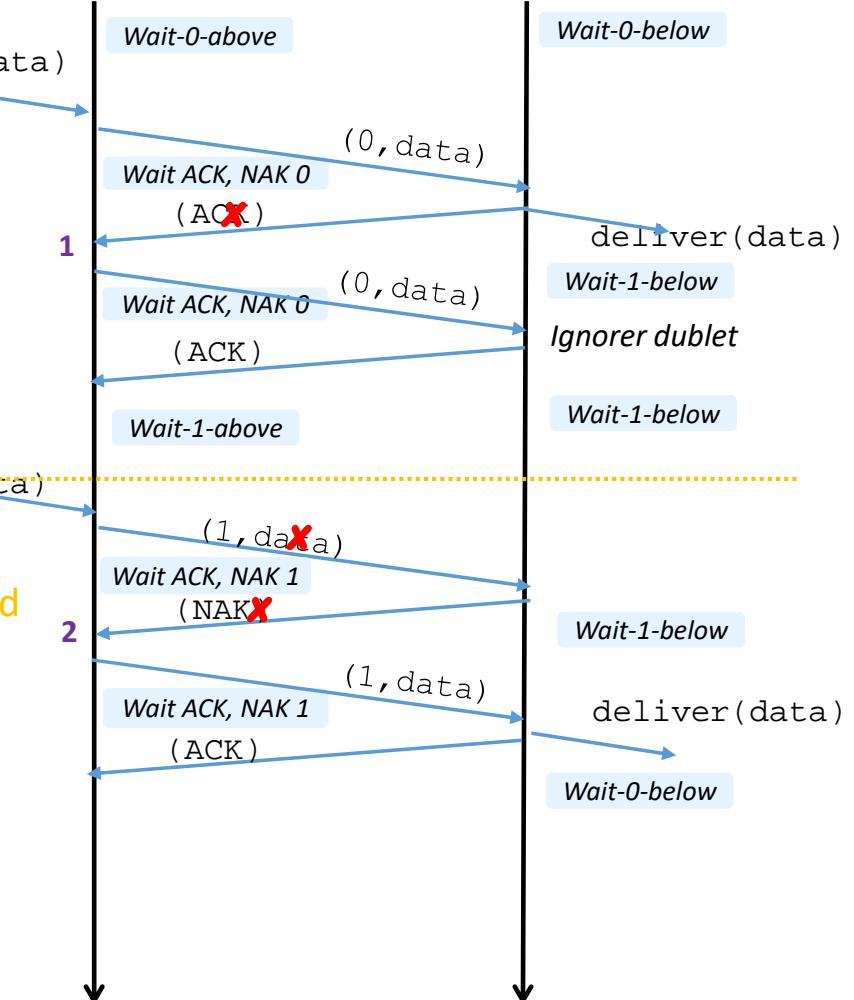
rdt\_recv(rcvpkt)  
&& !iscorrupt(rcvpkt)  
&& !isACK(rcvpkt)  
&& !isNAK(rcvpkt)  
udt\_send(sndpkt)

rdt\_send(data)

Scenarie med ødelagt NAK

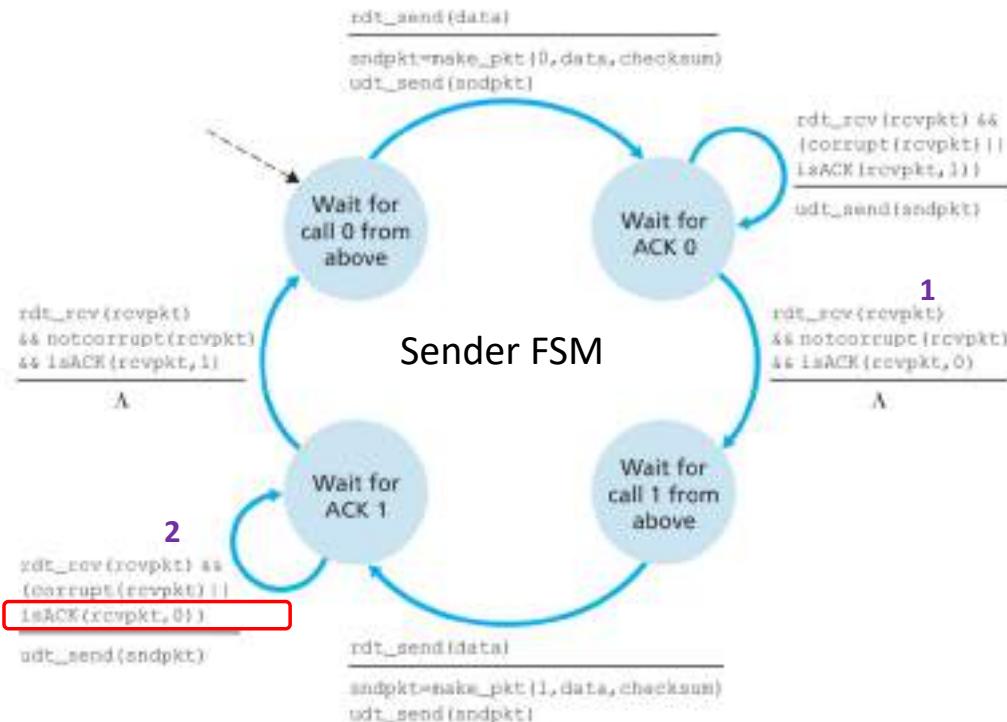
Sender  
(transport lag)

Modtager  
(transport lag)



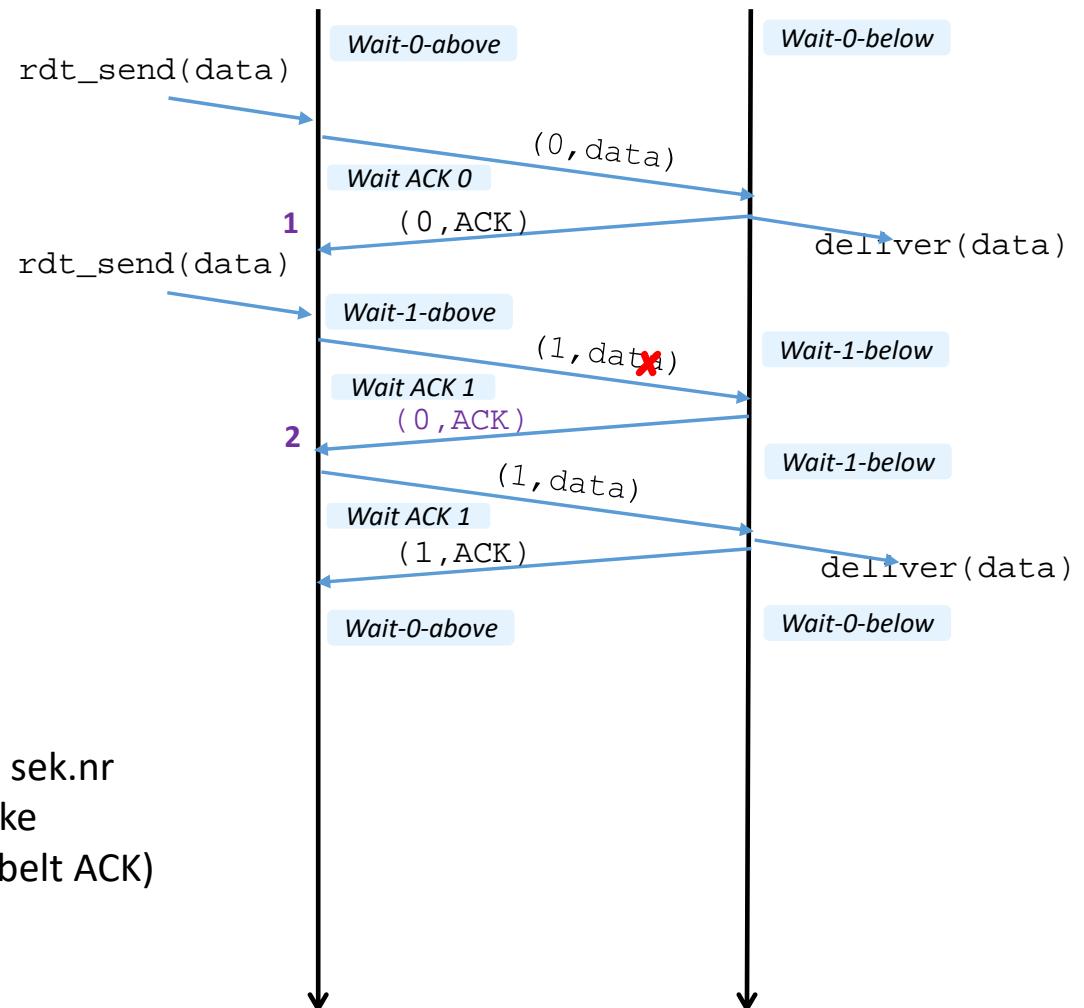
- Tabt ACK, NAK har samme effekt: retransmission
- Kun ACK med forventet sekvensnr, bringer protokollen frem
- ⇒Kan vi klare os med kun én af ACK, NAK?

# Sekvens-numre og kun ACK (rdt2.2)



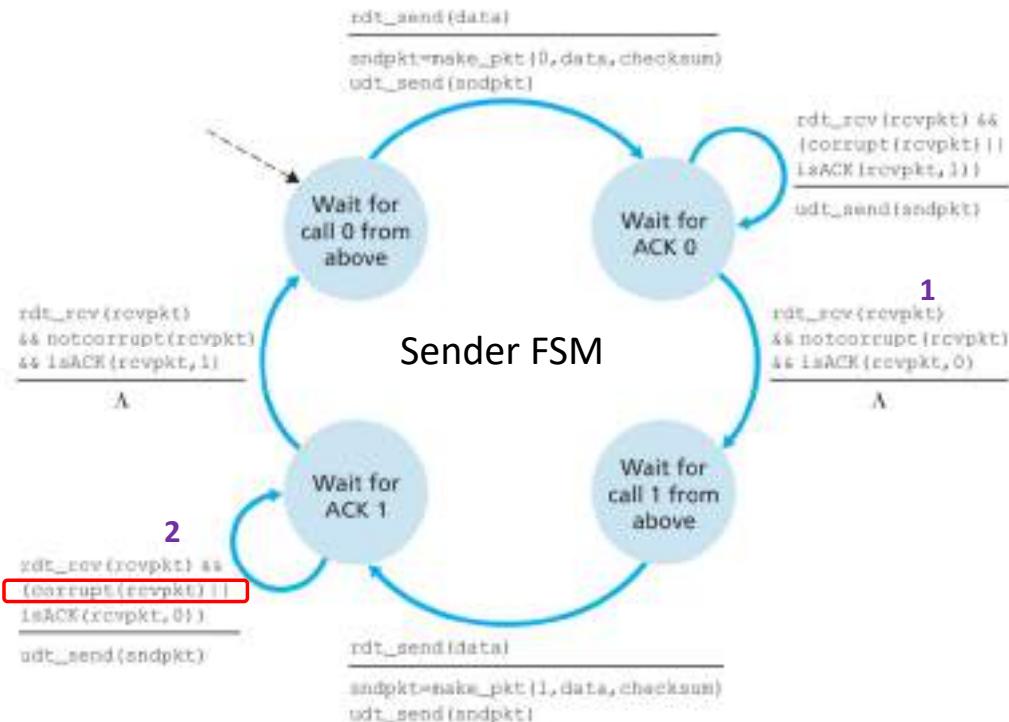
Scenarie med ødelagt data

Sender  
(transport lag)  
Modtager  
(transport lag)



- Ved tabt data, kvitterer modtager med seneste korrekte sek.nr
- ACK inkluderer sekvensnr på den korrekt modtagne pakke
- Sender, der modtager ACK med uventet sekvensnr (dobbelt ACK)
- ⇒ retransmission

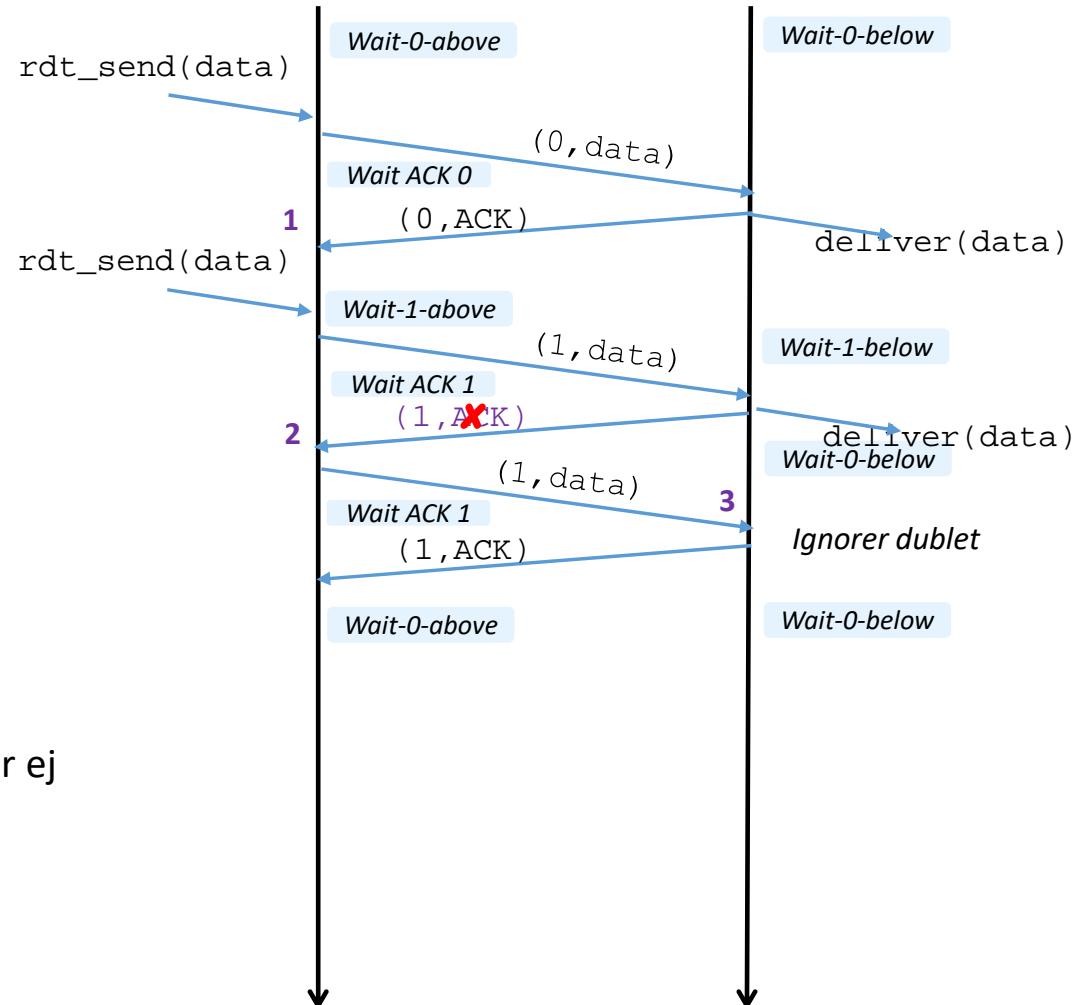
# Sekvens-numre og kun ACK (rdt2.2)



Scenarie med ødelagt ACK

Sender  
(transport lag)

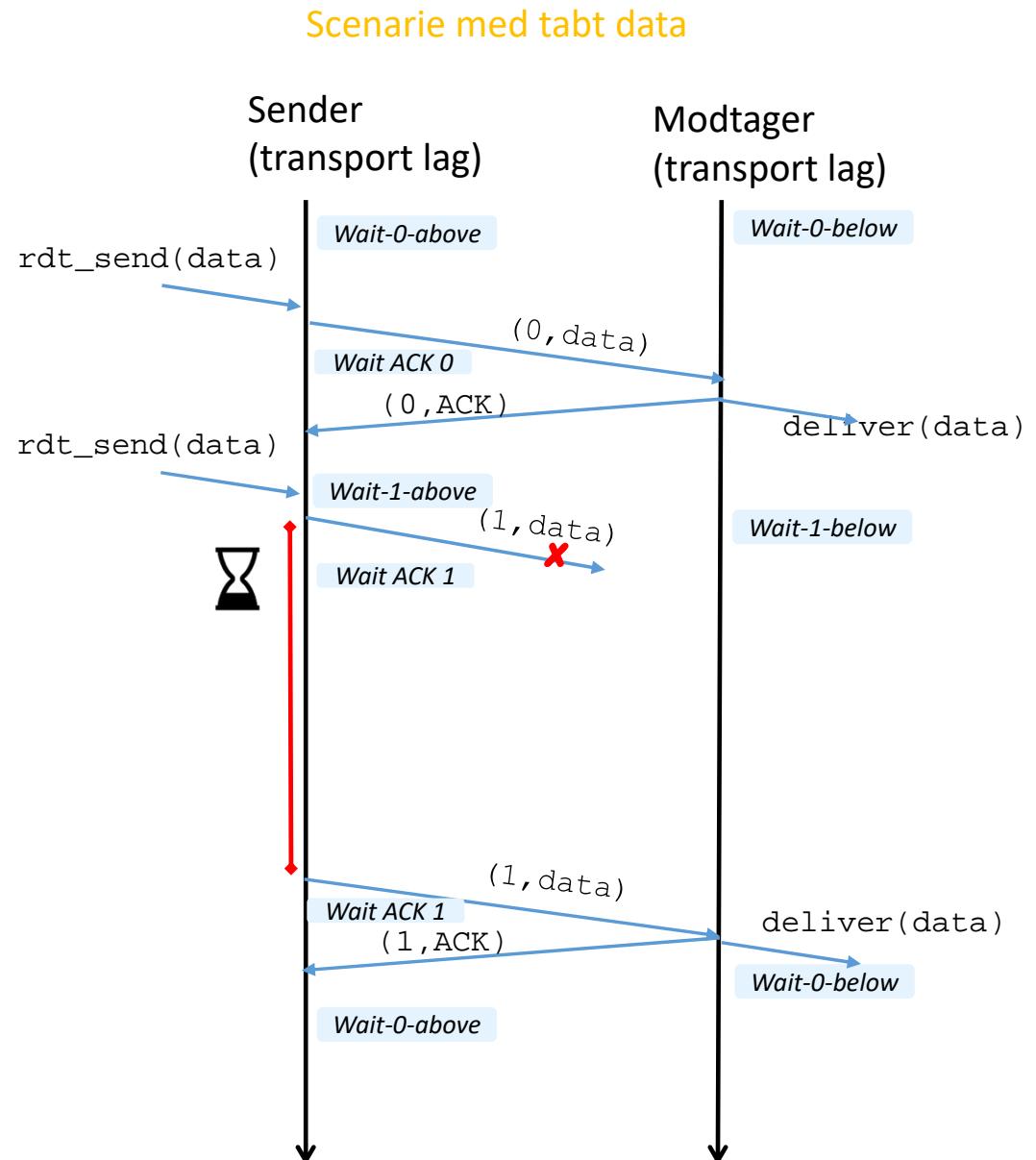
Modtager  
(transport lag)



- Ved tabt ACK, ved sender ikke om data er modtager eller ej
- ⇒ retransmission
- 3: modtager dublet: ignorerer data, men gensend ACK

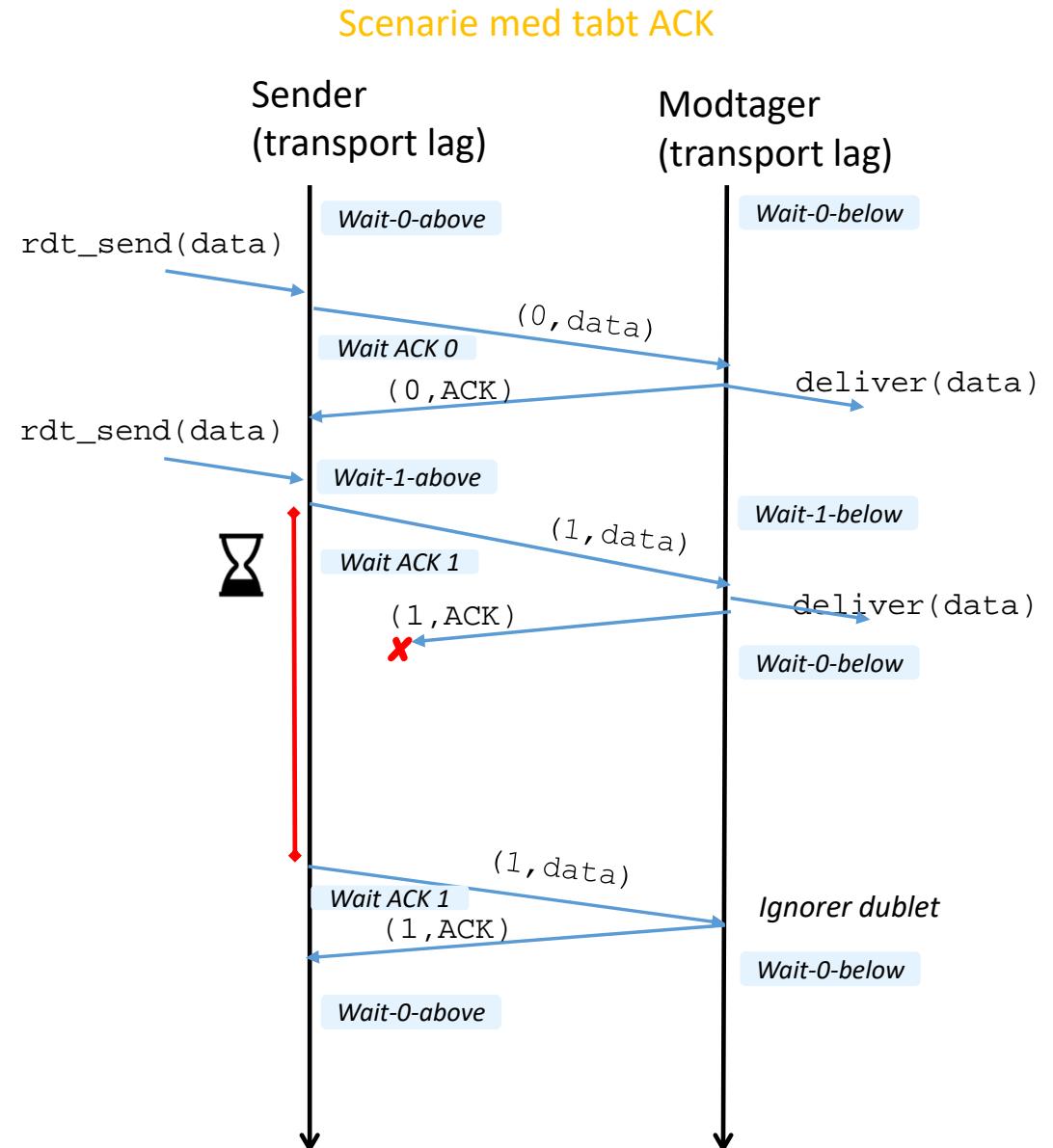
# Pakketab 1: rtd3.0

- Antagelser:
  - Kanalen kan **tabe pakker**
  - Mulighed for bit fejl
  - Bevarer rækkefølge
- Sender afventer "rimelig tid" på et ACK, ellers formodes data tabt.
- $\Rightarrow$  retransmission



# Pakketab 2: rtd3.0

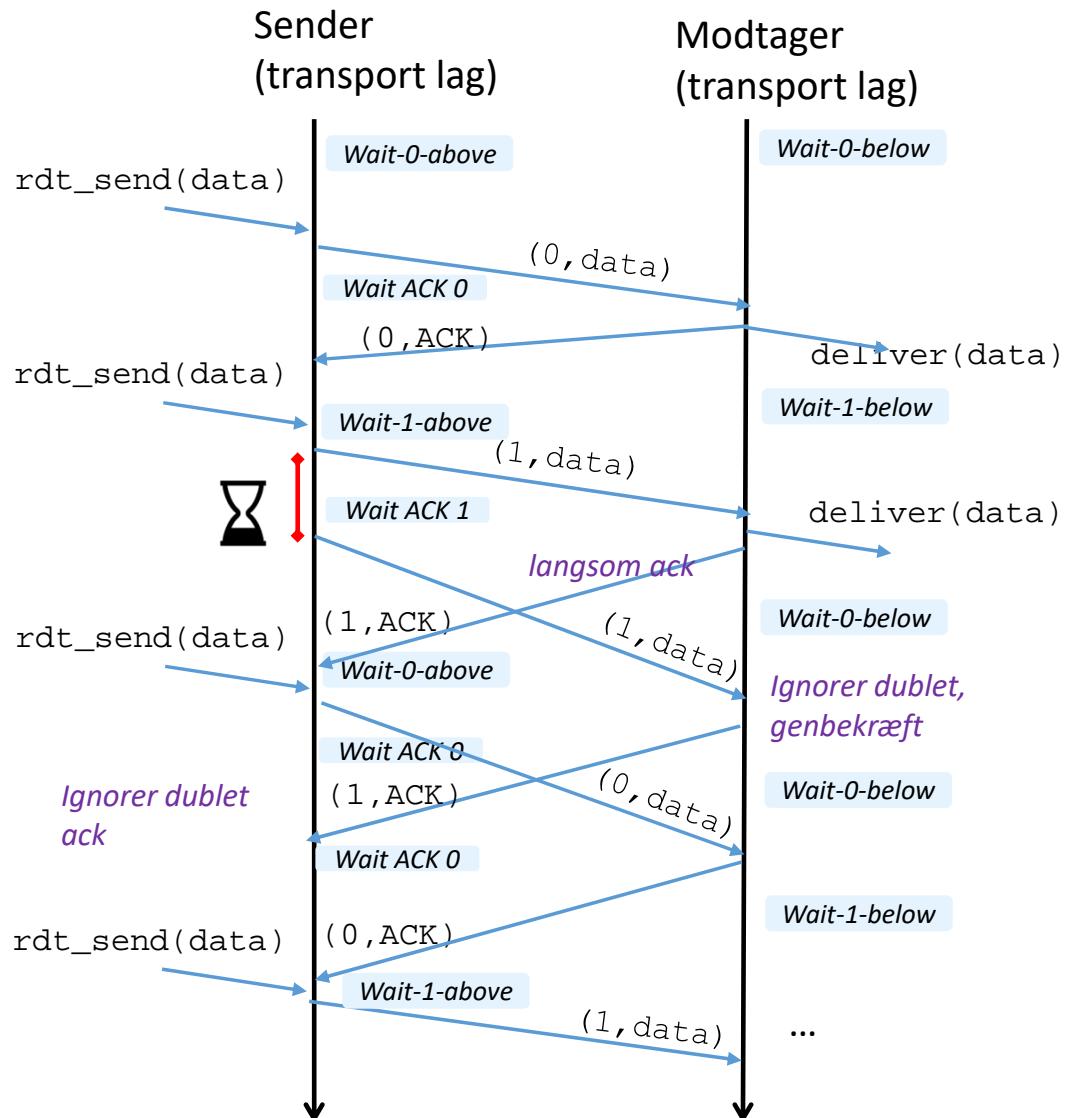
- Antagelser:
  - Kanalen kan tage pakker
  - Mulighed for bit fejl
  - Bevarer rækkefølge
- Sender afventer "rimelig tid" på et ACK, ellers formodes data tabt.
- $\Rightarrow$  retransmission



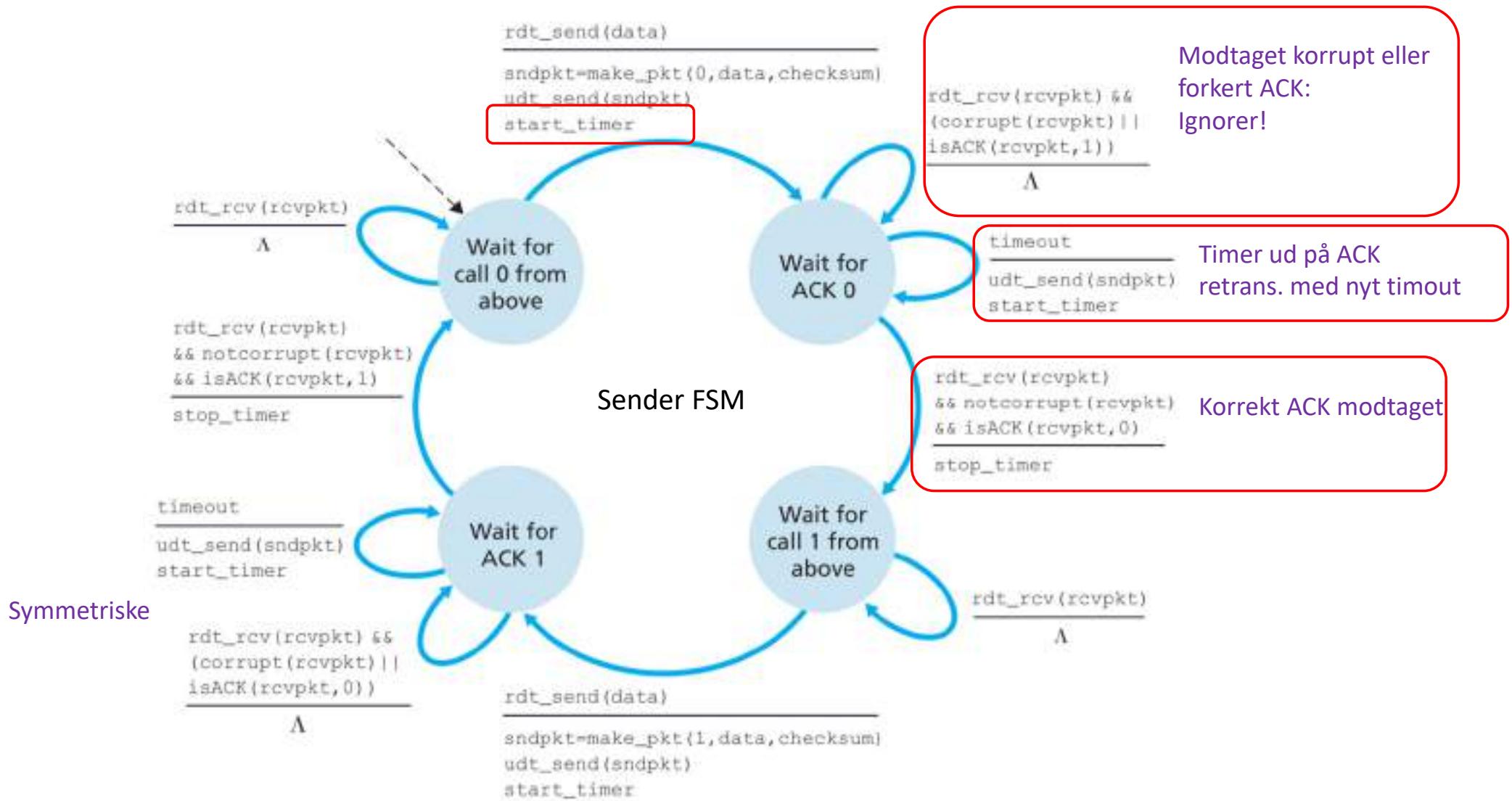
# Pakketab 3: rtd3.0

- Hvordan bestemmes "timeout" tid?
  - For lang: Unødvendig langsom
  - For kort: unødvendig retransmission af data og ACK
- Estimeres ud fra RTT
  - Varierer dynamisk efter netværksbelastning
  - Finde god timeout værdi, men kan aldrig undgå for tidlig / for langsom timeout

## Scenarie med for lille timeout

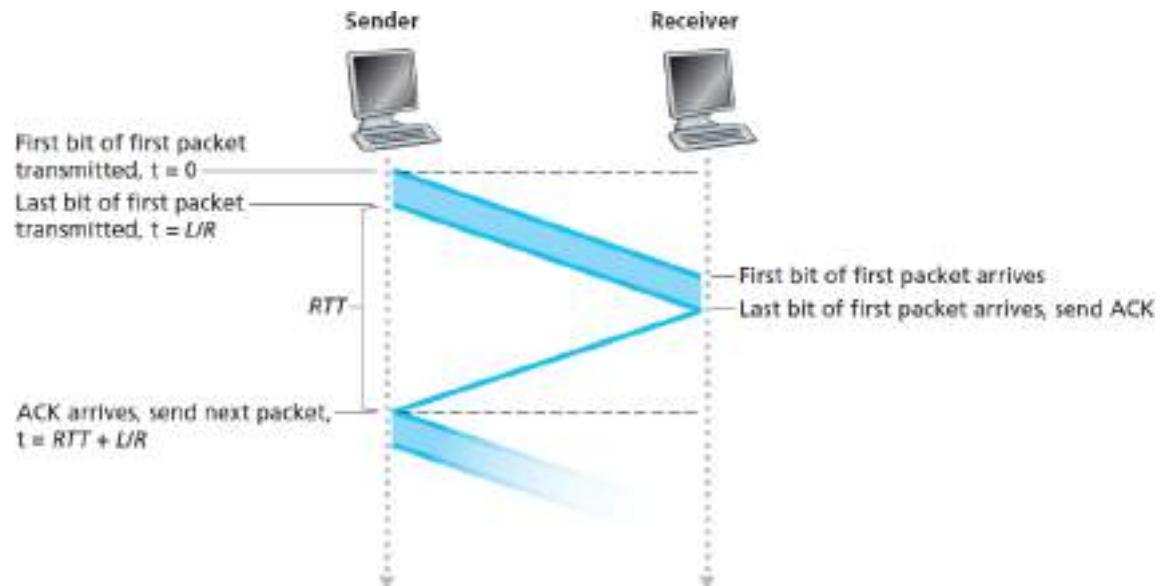


# Rtd 3.0



# Rtd3.0

- "Den alternerende bit-protokol"
- Mange scenarier!
  - Protokol test og verifikation?
- En passende detaljeret og præcis FSM kan "nemt" laves om til et program m. event-drevet programmering
- Stop&Wait
- Korrekt! Men Håbløs langsom!



Eksempel m. trans-US link:  
1 Gbps link, 15 ms prop. delay, 8000 bit pakke:

$$U_{\text{sender}} = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = \frac{.008}{30.008} = 0.00027$$

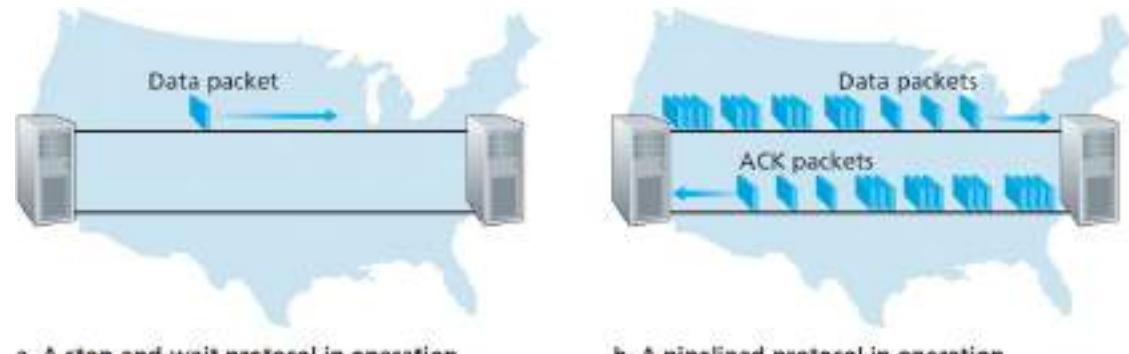
# Pipelinede protokoller

Hvordan får vi hurtigt transporteret en masse data korrekt til modtager?

Kan vi undgå en stop&wait ?

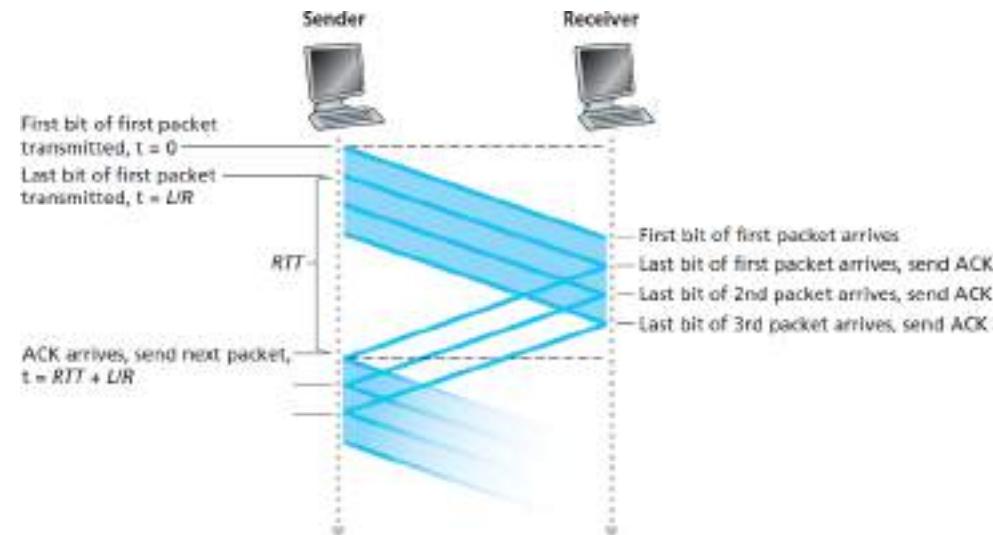
# Pipelinede protokoller

- Øg throughput (bps) ved at udsende flere pakker før vi afventer ACK
  - Udnytte kanalen
  - Uden at oversvømme modtager (flow-kontrol)
  - Uden at overbelaste netværket (congestion-kontrol)
- Pakker gemmes i buffere på sender og modtager til de er leveret
- 2 overordenede strategier for re-transmission
  - Go-Back-N
  - Selective-repeat



a. A stop-and-wait protocol in operation

b. A pipelined protocol in operation

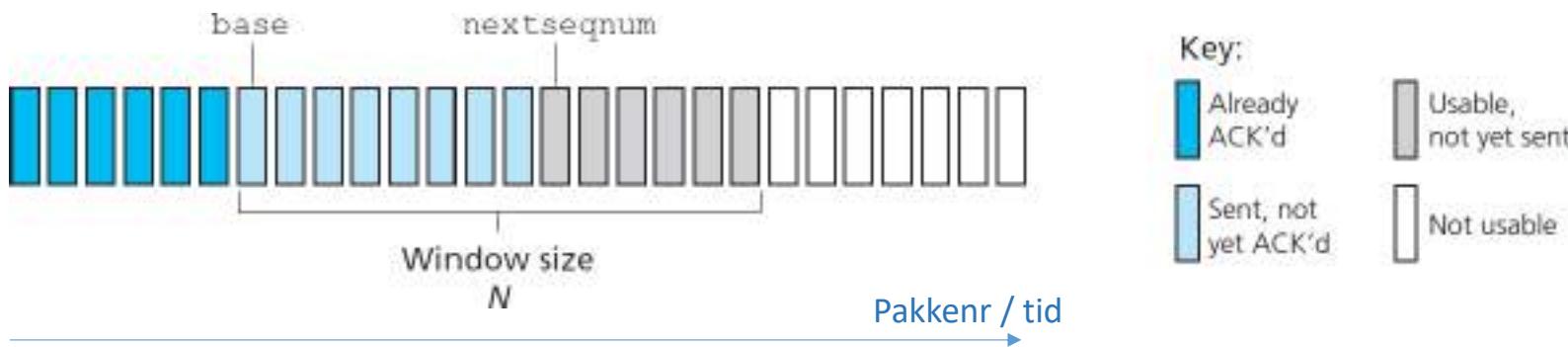


Eksempel m. trans-US link: 1 Gbps link, 15 ms prop. delay

Bits på link (BW-delay produkt) undervejs:  
 $(10^9 \text{ bps} * 15 * 10^{-3} \text{ s}) / 8 \text{ bits/byte} = 1.9 \text{ Mbyte}$

# Go-back-N

- Tillad at sender max har max  $N$  pakker undervejs i ”pipelinen”
  - Begrænse hvor meget data sender og modtager skal være klar til at bufferere
- Sekvensnummer felt i header med  $k$  bits:  $[0, 1, \dots, 2^k - 1]$  (fx  $k=16 \Rightarrow [0, 65535]$ )
- Senders sekvensnumre:
  - **base**: start på aktuelt vindue (ældste sendte ukvitterede pakke)
  - **nextSeqNum**: næste ledige sekvensnummer



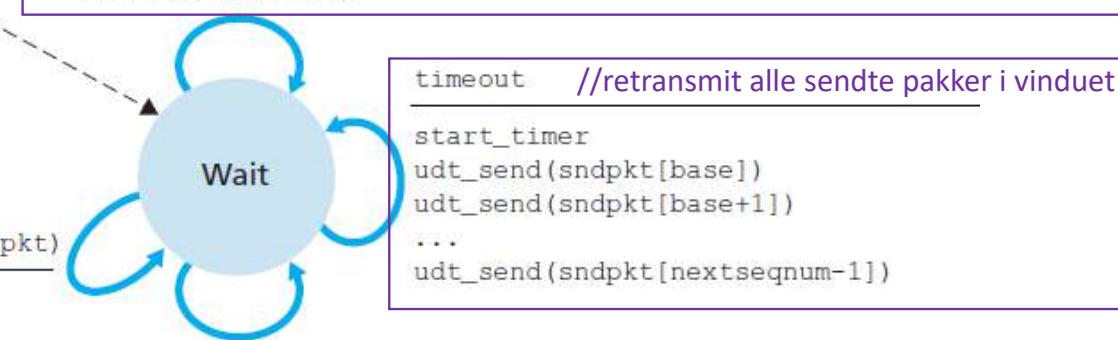
- Sender får ACK ( $n$ ): modtager har fået alle pakker med sekvens numre t.o.m  $n$  er korrekt (“kumulativt” ACK)
- Sætter timer for ældste ukvitterede pakke
- Timeout  $\Rightarrow$  gensender alle pakker i nuværende vindue, der er sendt efter  $n$
- Vinduet glider en tak frem, hver gang ældste pakke kvitteres

# Go-Back-N Sender

$\Lambda$   
base=1  
nextseqnum=1

```
rdt_send(data)
if(nextseqnum<base+N) { //har vi ledigt sekvensnr i vinduet?
 sndpkt[nextseqnum]=make_pkt(nextseqnum, data, checksum) //gem ny pakke i buffer
 udt_send(sndpkt[nextseqnum])
 if(base==nextseqnum)
 start_timer //Første pakke (ældst) i vinduet? Sæt timer
 nextseqnum++
}
else
 refuse_data(data) //app. laget må afvente (OS blokkerer process)
```

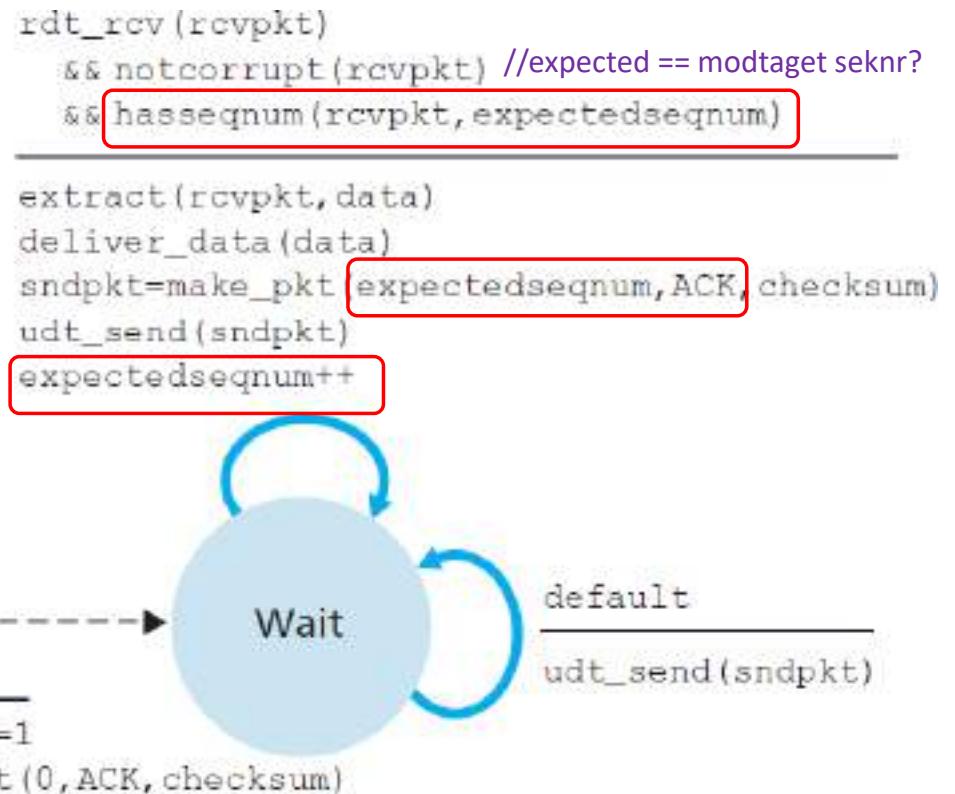
$\Lambda$   
rdt\_rcv(rcvpkt) && corrupt(rcvpkt)



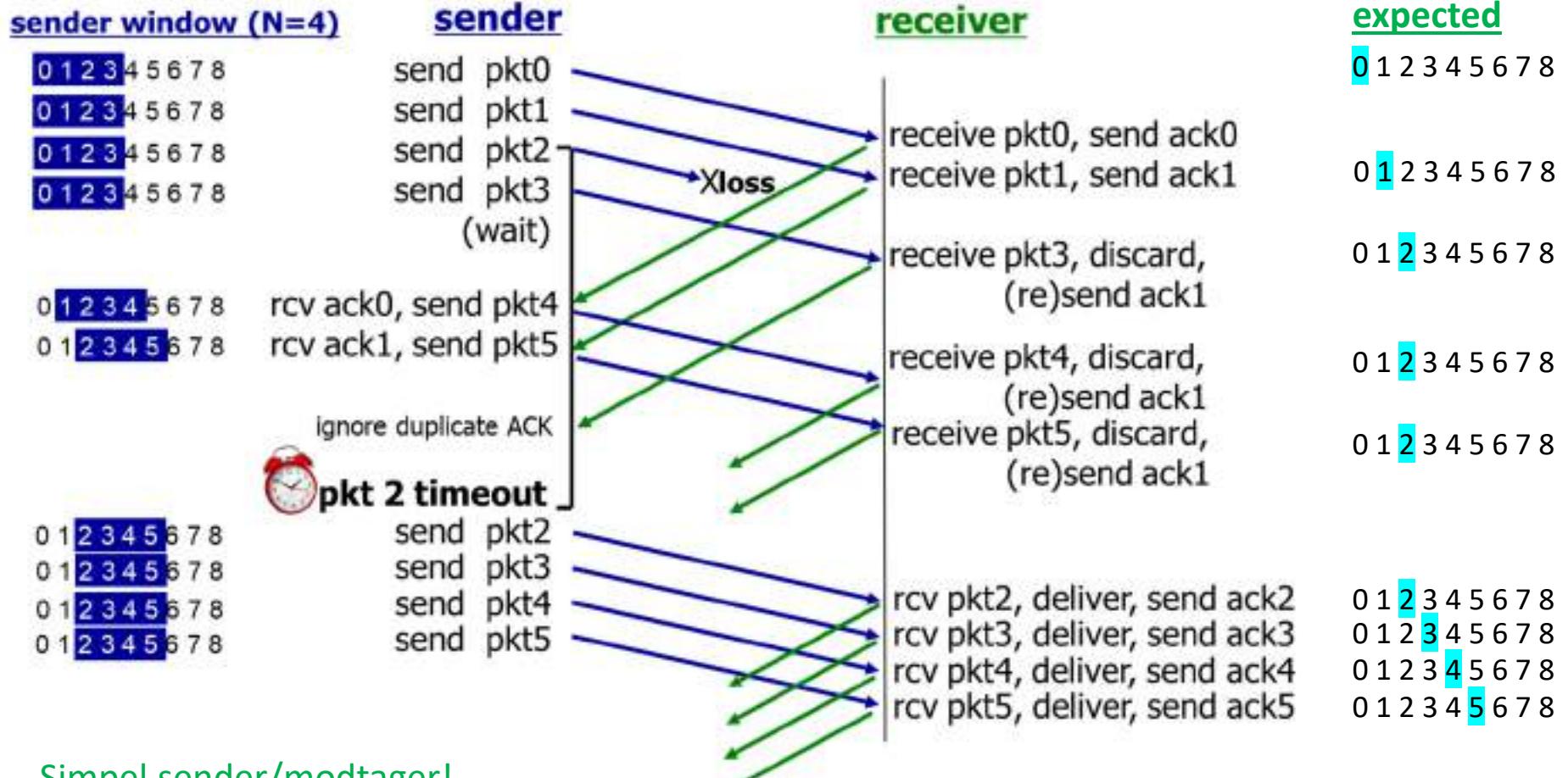
```
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
base=getacknum(rcvpkt)+1 //vi modtog (ack, n): alt t.o.m i modtager OK
If(base==nextseqnum) //glid vinduet frem til seq. nr. n
 stop_timer
else
 start_timer //start time for nu ældste pakke
```

# Go-Back-N Modtager

- Forventer at modtage data i stigende sekvens-orden, uden "huller": **in-order**
  - Tæller: **Expectedseqnum**
- Sender ACK for den korrekt modtagne pakke med størst **in-order** seknr.
  - Kan give dublerede ACKs
- Pakker der ankommer "out-of-order"
  - Bortkastes: ingen buffer på modtager siden
  - Gensende ACK med højeste korrekt modtaget sek nr.



# Go-Back-N Scenarie



Simpel sender/modtager!

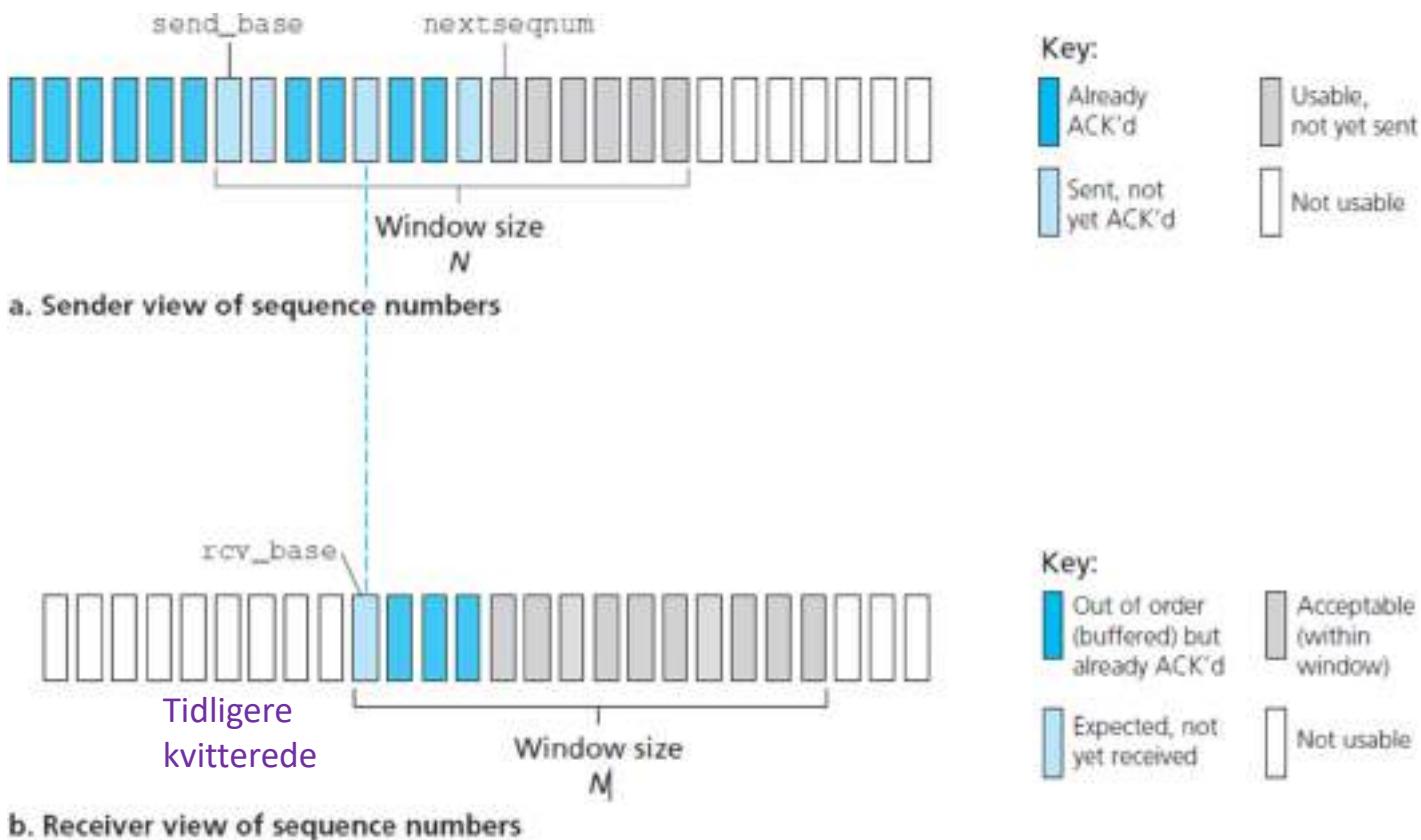
I værste fald skal vi gå N skrid tilbage!

# Selective Repeat

- Tillad at sender har max  $N$  pakker undervejs i ”pipelen”
- GBN bortkaster korrekte pakker der ankommer out-of-order
- I Selektive Repeat:
  - Modtager gemmer pakker, der ankommer out-of-order i en buffer
  - Modtager kvitterer *individuelt* for hver modtagne pakke
- Sender *retransmitterer kun de pakker der mangler kvittering*
- Sender sætter en timer for hver enkelt pakke den sender

# Selective Repeat

- Tillad at sender har max  $N$  udestående pakker
  - Bestemt fra ældste ukvitterede pakke
- Senders sekvensnumre:
  - **send\_base**: start på aktuelt vindue (ældste ukvitterede pakke)
  - **nextSeqNum**: næste ledige sekvensnummer
- Både sender og modtager vindue!
- Modtagers sekvens nr
  - **rcv\_base**: start på aktuelt vindue (ældste forventede pakke)
  - Accepterer at modtage  $N$  pakker frem



# Selective Repeat

## Sender

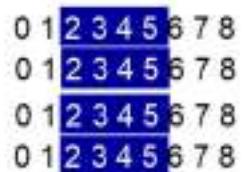
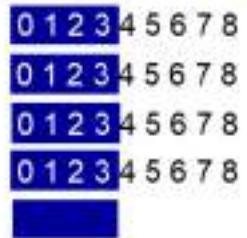
- Data klar fra app-lag?
  - Hvis ledigt seknr ( $n$ ) i vinduet, send pakke  $n$ .
  - Start timer for pakke  $n$
- Timeout ( $n$ ):
  - gensend pakke  $n$
  - Genstart timer for pakke  $n$
- ACK ( $n$ ) i  $[sendbase, sendbase+N-1]$ :
  - Marker  $n$  som modtaget
  - Hvis  $n$  var den første ukvitterede, forskyd vinduet frem til næste ukvitterede pakke

## Modtager

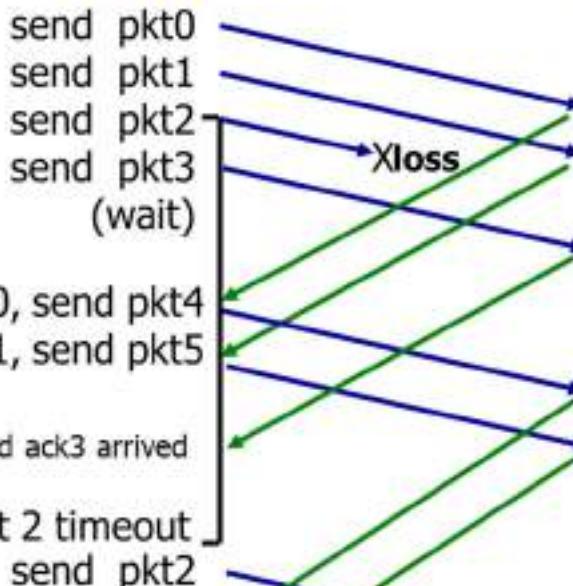
- Modtaget pakke  $n$  i  $[rcvbase, rcvbase+N-1]$ :?
  - (pakken er indenfor det forventede område)
  - Send ACK  $n$ .
  - Gem pakken  $n$  i buffer
  - Aflever alle in-order pakker til app laget
  - Skyd vinduet frem til næste forventede pakke
- Modtaget pakke  $n$  i  $[rcvbase-N, rcvbase-1]$ 
  - (pakken er tidligere kvitteret/leveret)
  - Gensend ACK  $n$ .
- ELLERS
  - Drop pakken

# Selective repeat Scenarie

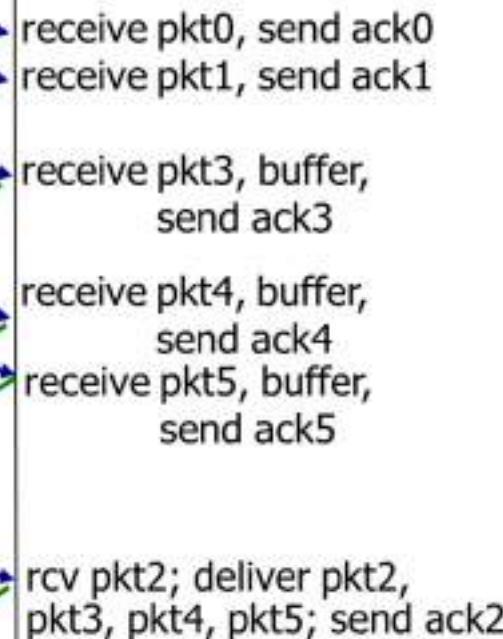
sender window (N=4)



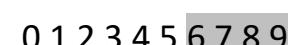
sender



receiver



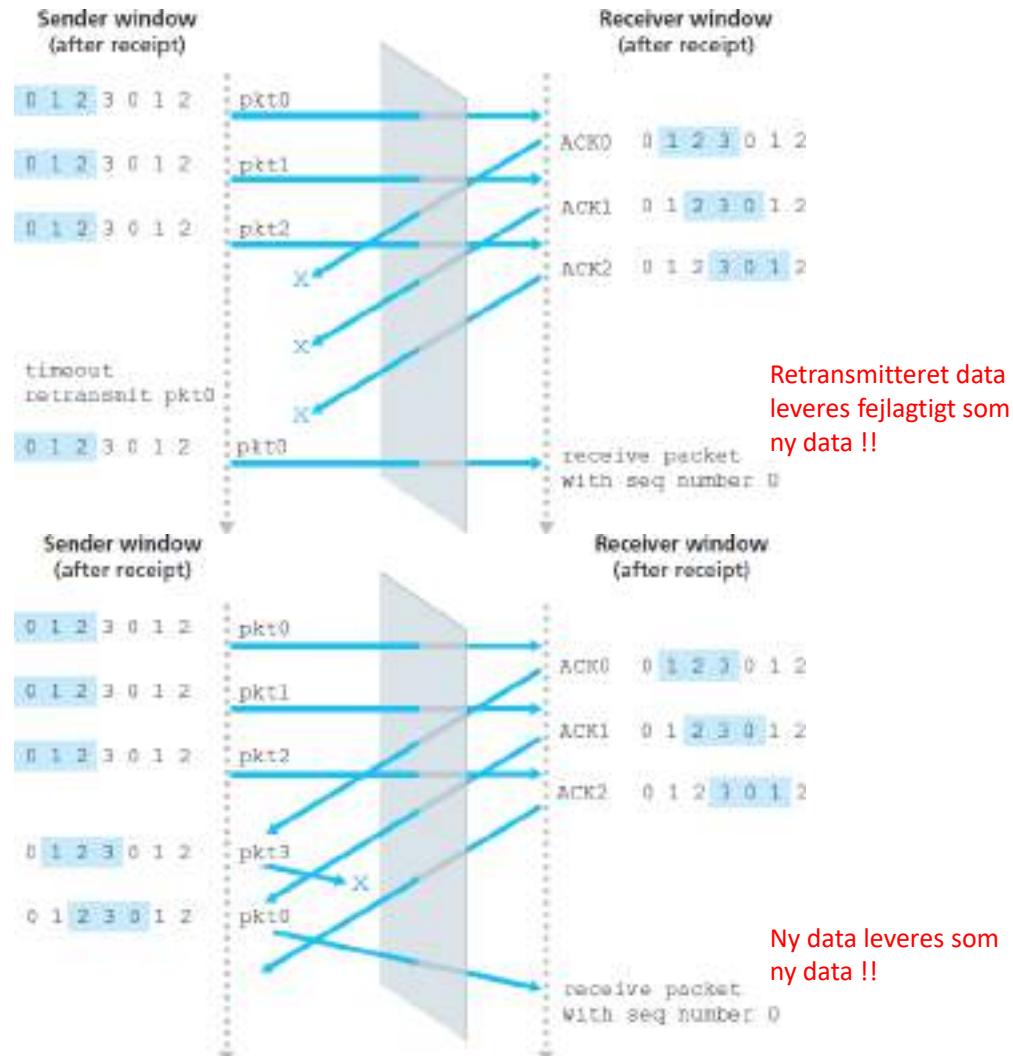
Receiver window expected



Q: what happens when ack2 arrives?

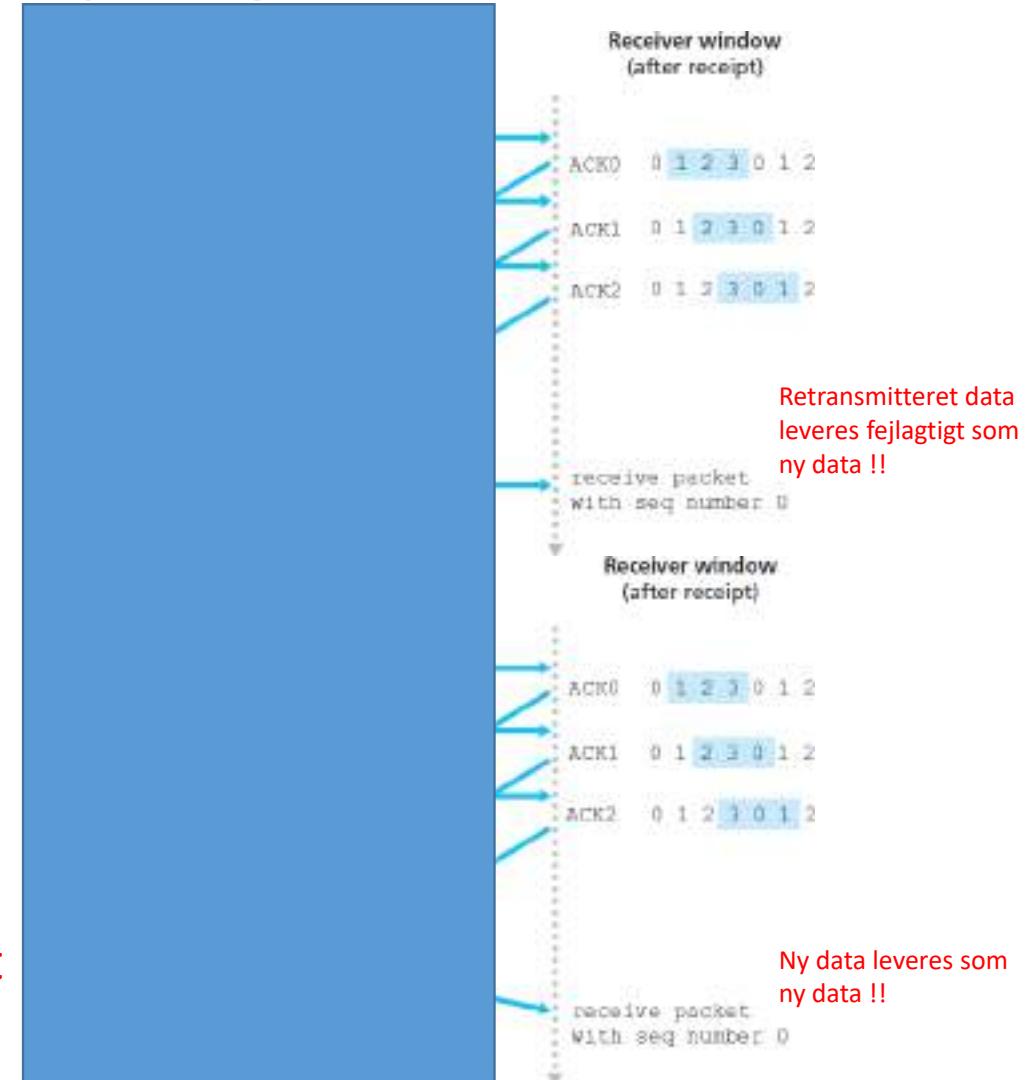
# Dilemma omkring sekvensnumre

- Header har kun plads til  $2^k - 1$  sekvensnumre
  - Wrap rundt, start forfra med 0
- Antag sekvens numre 0,1,2,3
- N=3
- 2 Scenarier
  - ACK tab
  - Data tab



# Dilemma omkring sekvensnumre

- Header har kun plads til  $2^k - 1$  sekvensnumre
  - Wrap rundt, start forfra med 0
- Antag sekvens numre 0,1,2,3
- N=3
- 2 Scenarier
  - ACK tab
  - Data tab
- Modtager kan ikke se forskel!
  - Retransmitteret data accepteres som nyt



# Udeståender

- Genbrug af sekvens numre?
  - Scenarie b indikerer at:  $2N < k^2 - 1$
- Hvad med en kanal som omordner pakker?
  - **Uproblematisch:** Hvis den ikke er for gammel, så fx buffererer selektiv repeat den, og leverer i rækkefølge
  - **Mere problematisk:** En *meget meget* gammel pakke kan have et sekvens nr, der passer ind i modtagers nye vindue: vi leverer forkert data?!
    - På internettet antages at pakker ikke lever ud over en max tid (3 min)
      - IP datagrammer har et "Time to live" (TTL / hop limit) felt som tælles ned hver gang de videresendes
    - NB: i et computer netværk er det problematisk at anvende klokken som tidsstempel, da alle computere har sit eget ur, der ikke kan synkroniseres (præcist.)
- Flow- og Congestion kontrol? Næste lektion!

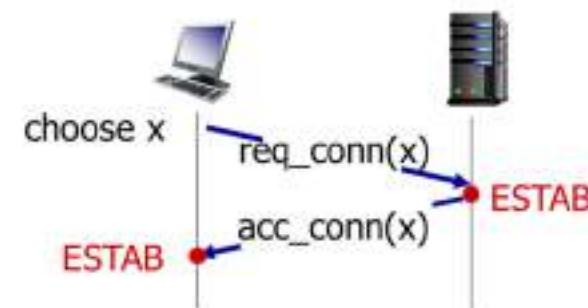
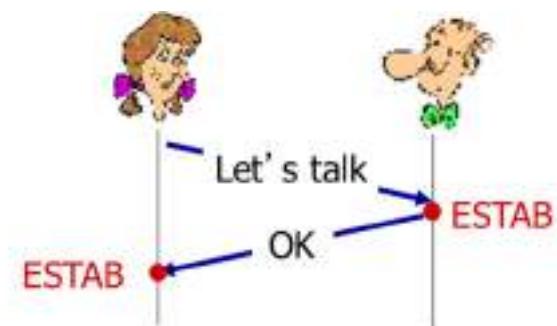
# Etablering af TCP forbindelser

Hvordan forbinder en klient sig til en server?

Hvordan stopper de kommunikationen og nedriver forbindelsen igen?

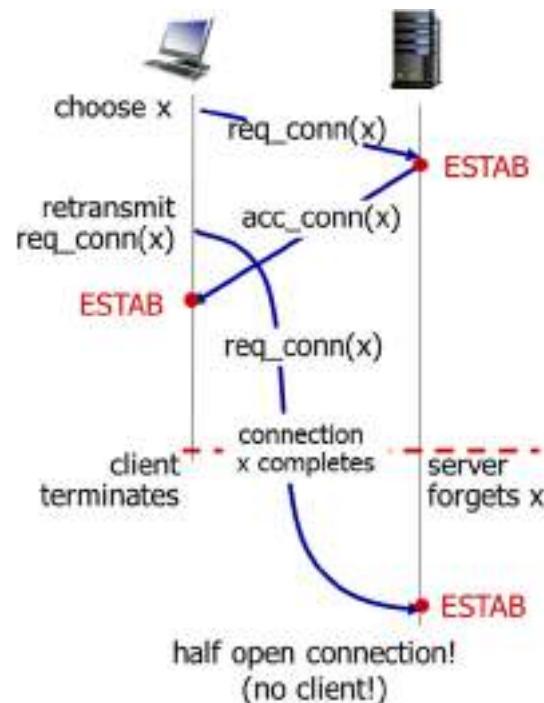
# Etablering og nedlægning af forbindelser

- Sender Klient og server skal blive enige om, at de har en forbindelse
  - Afsætte buffer-plads;
  - Initialisere "sliding window" parametre: sekvens nummer + vindues størrelse
  - I begge retninger da TCP er bi-direktionel
- Komplikationer
  - Gamle pakker (fx fra re-transmissioner på tidligere forbindelse) må ikke medgå i ny forbindelse
  - Ved nedlukning: afvente at alt sendt data er leveret til modtager, selv i tilfælde af, at retransmission er nødvendigt.
  - Special pakker til oprettelse og nedlukning af forbindelser kan gå tabt!

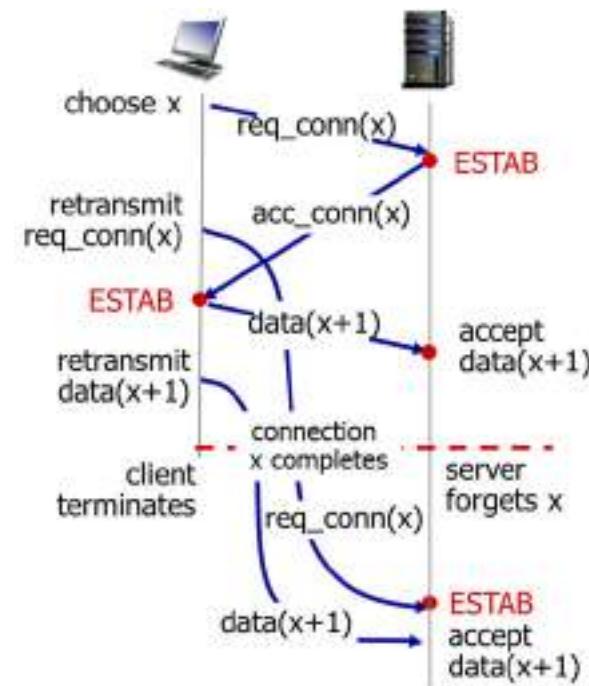


# 2-vejs handshake ?

- Eksempler på problematiske scenarier



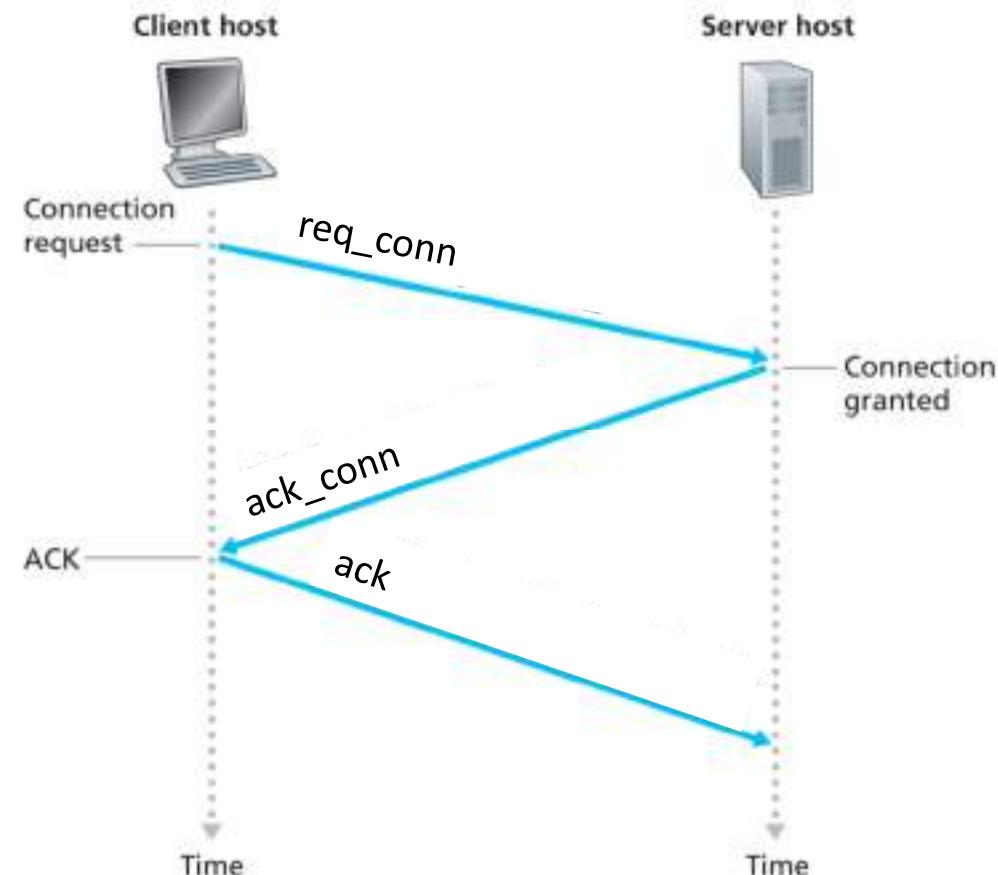
Server afsætter ressourcer til klient,  
der ikke findes



Server modtager gammel data.

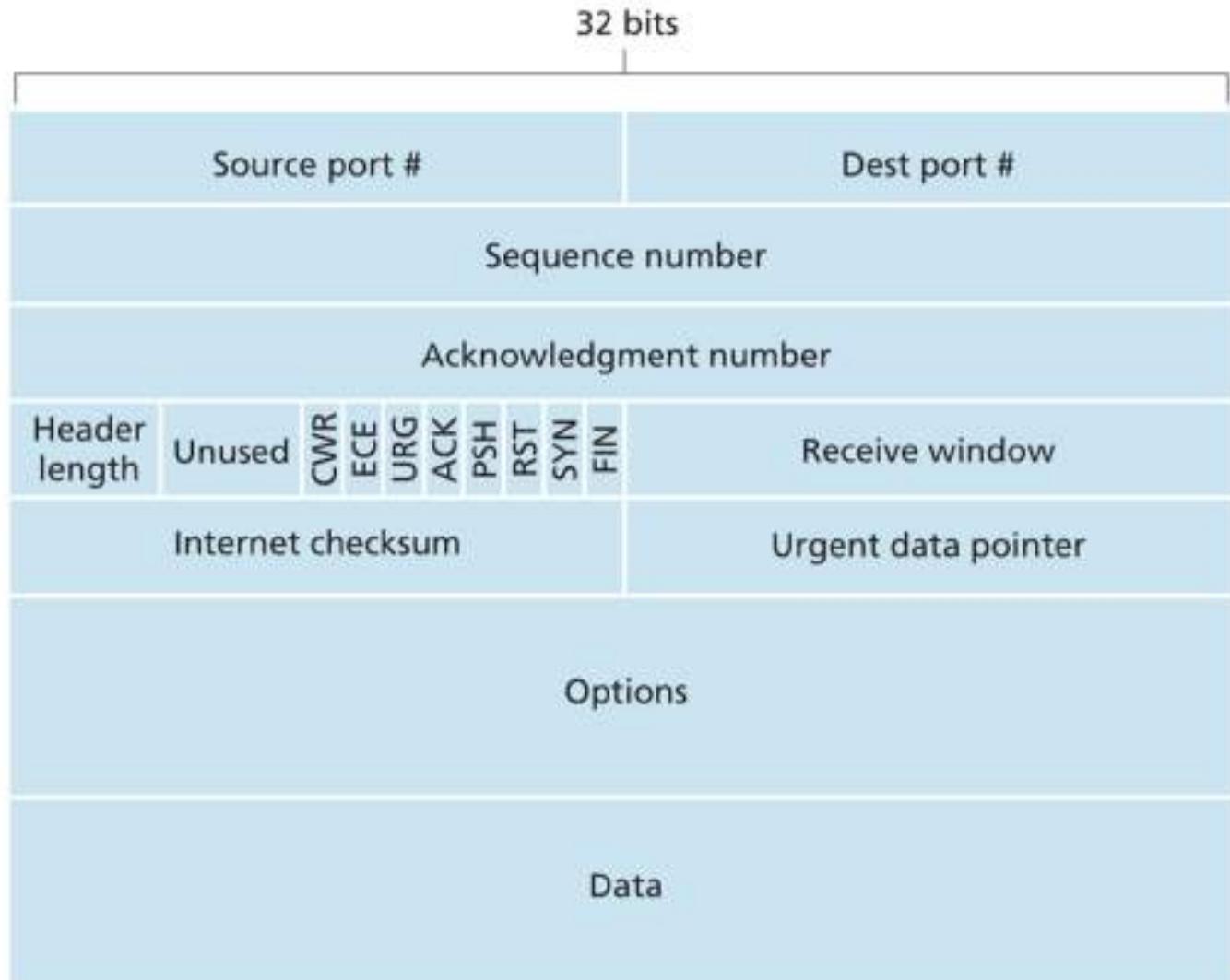
# 3-way-handshake

- Forudsætning:
  - Server-process lytter på socket (bundet til ønsket port)
  - Klient server process vil forbinde sig til server
- Transport laget foretager et 3-vejs handshake
  - Gensidig kvittering



# TCP header

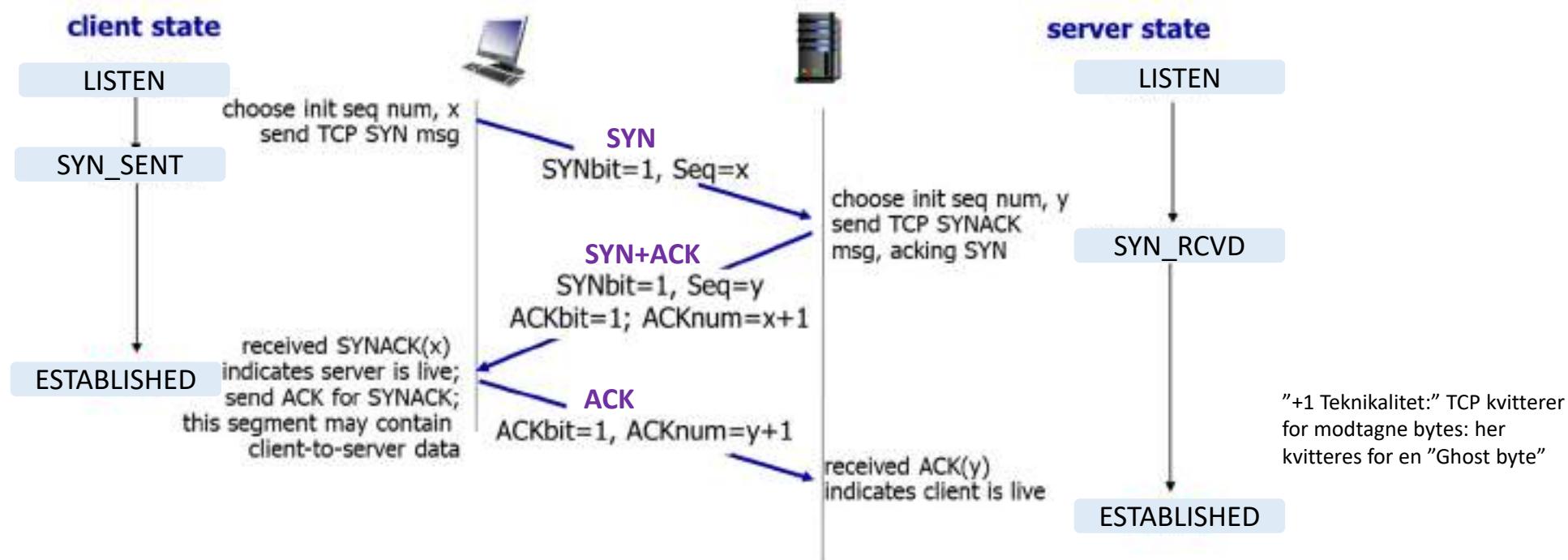
- TCP *Segment*
- Kontrol-bits til etablering og nedrivning af forbindelser:
  - SYNchronize
  - FINish
  - ACKnowledge (også til data)
    - ACK=1 ⇒ gyldig info i ack no feltet.
- Sekvens og ack. numre



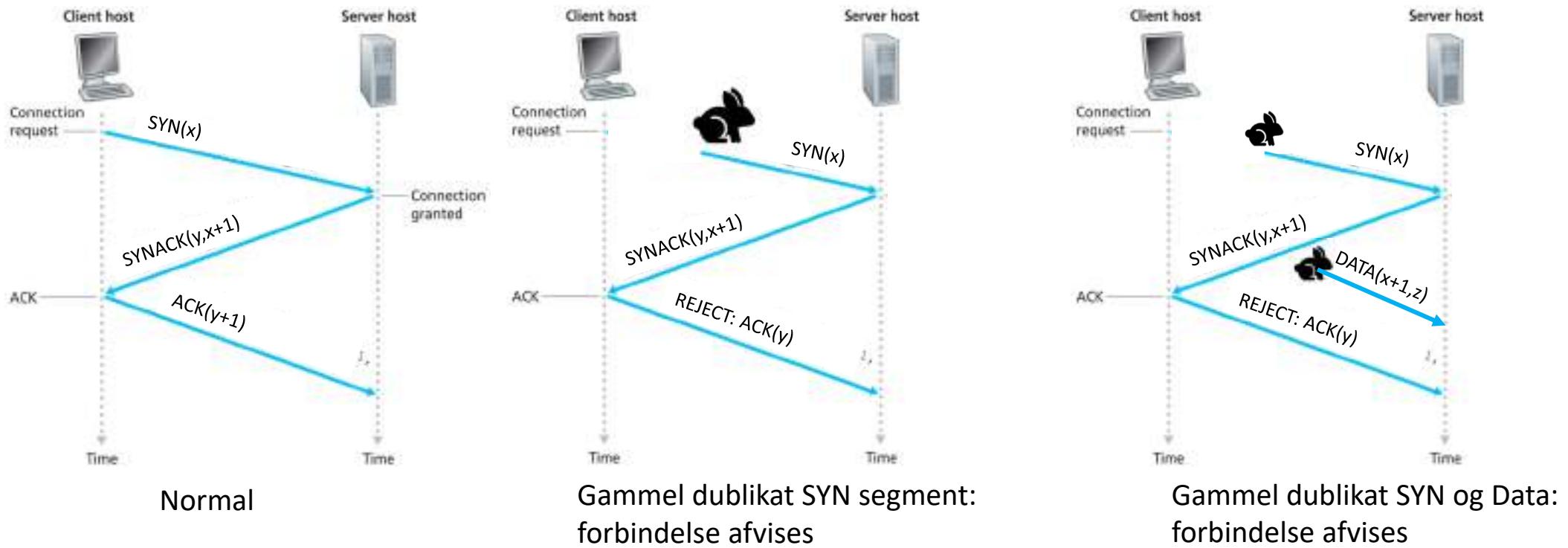
# 3-way-handshake

- **req\_conn:** TCP **SYN** segment (SYN=1, Seq=x):
  - x=klients (tilfældigt) valgte init sekvensnr
- **ack\_conn:** TCP **SYN+ACK** segment (SYN=1, ACK=1, Seq=y, AckNo=x+1):
  - y=servers (tilfældigt) valgte sekvensnummer
  - Server bekræfter; forventer at næste segment nr. har sekvens nr. x+1
- **ack :** TCP **ACK** (SYN=0, ACK=1, AckNo=y+1)
  - Klient bekræfter; forventer at næste segment nr. er sek. y+1

Se evt. denne video med  
[wireshark analyse af TCP 3-way handshake](#)



# 3-way-handshake

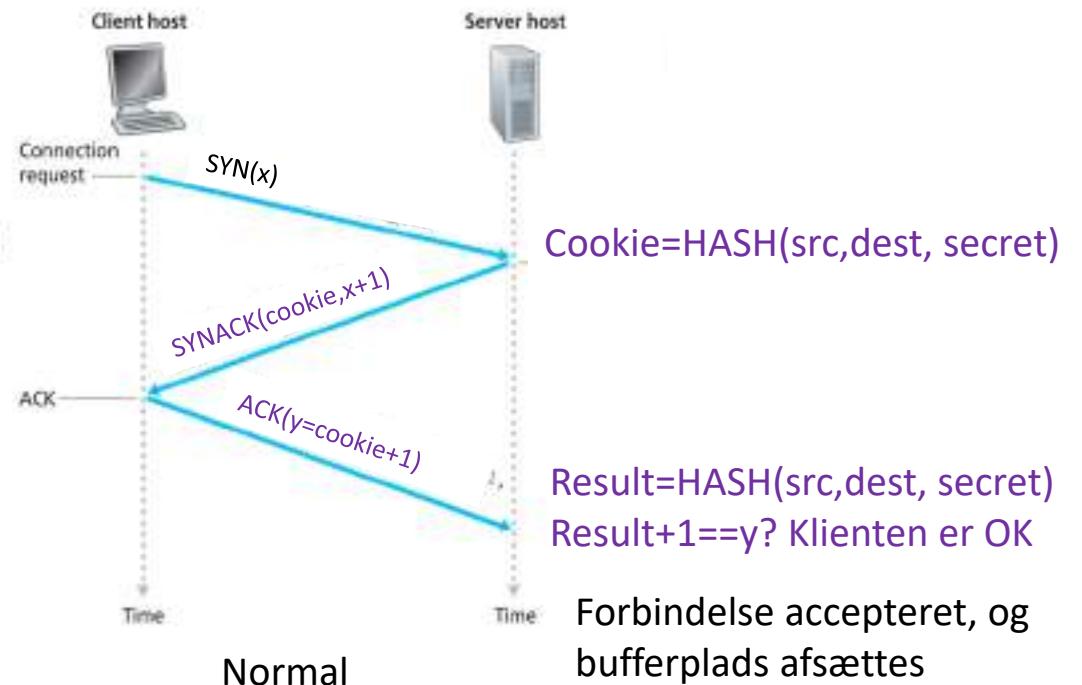


# SYN-flood angreb

- Angriber konstruerer falske SYN pakker
  - Modtagelse af SYN vil normalt få Server til at afsætte plads til buffere mv.
  - ”halv åben” forbindelse
  - Tilpas mange vil overvælde server

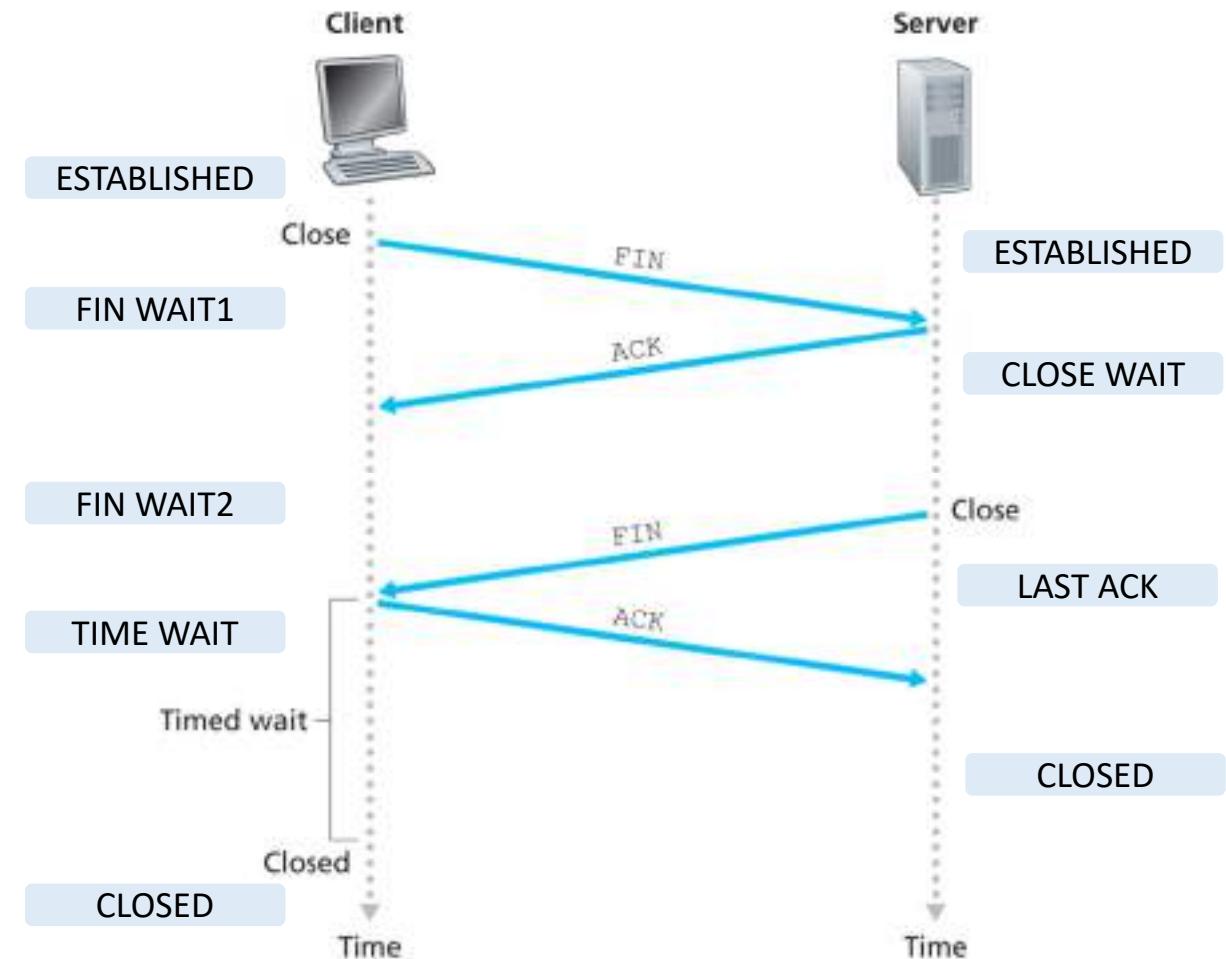
## • SYN Cookie Forsvar

- Afsæt kun ressourcer når der er en ”levende” / ”legitim” client bag forespørgslen



# Kontrolleret nedlægning

- Klient og server lukker begge forbindelsen
- I scenariet:
  - Klient starter
  - Server følger med sin egen
- TIME\_WAIT: giv tid (fx 30-120 sec.) til at gensende sidste ACK
- Mindst lige så mange "interessante" scenarier
- (faktisk teoretisk umuligt at de bliver "enige" om nedlukning: two-army problem")



# Multi-plexing/De-multiplexing

Hvordan skelnes mange forbindelser?

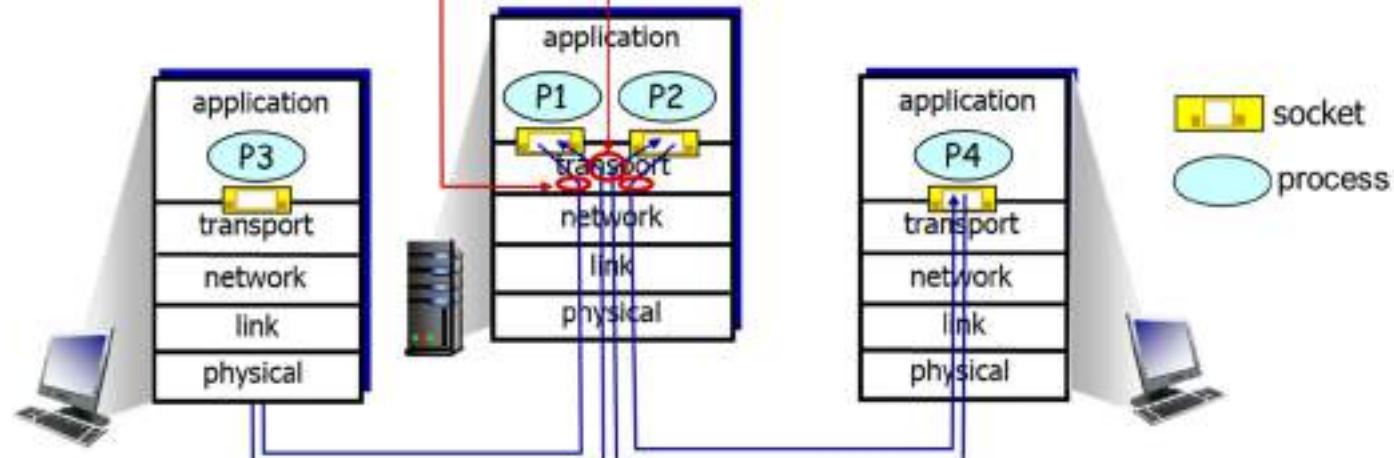
Hvordan får data til/fra rette socket?

# Multiplexing og demultiplexing

- Proces adresseres vha. hosts IP-nummer +port nummer indenfor en host
- Processer sender/modtager data via en **socket**, et bindeled mellem applikations- og transport-lag
- Der er mange processer på en host, dermed mange som transport laget skal betjene
  - En proces kan kommunikere med mange andre samtidigt (kan dermed have mange sockets)
- Hvordan leveres data til den rette socket?

## multiplexing at sender:

handle data from multiple  
sockets, add transport header  
(later used for demultiplexing)

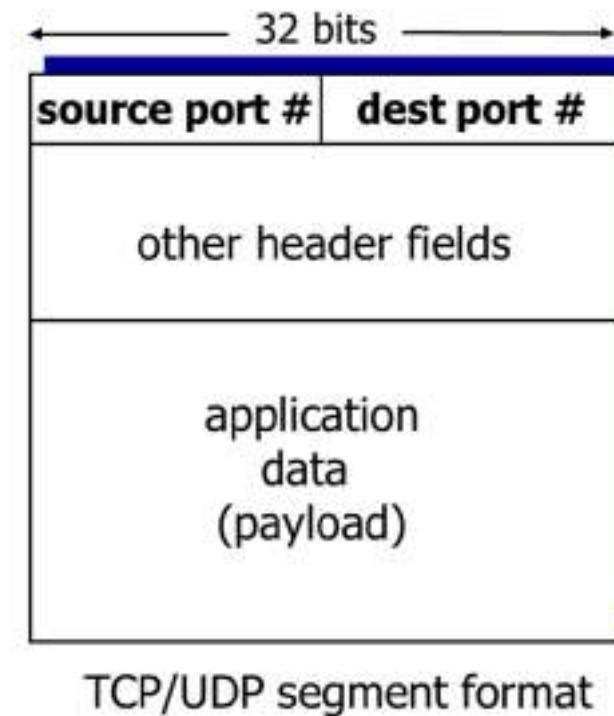


## demultiplexing at receiver:

use header info to deliver  
received segments to correct  
socket

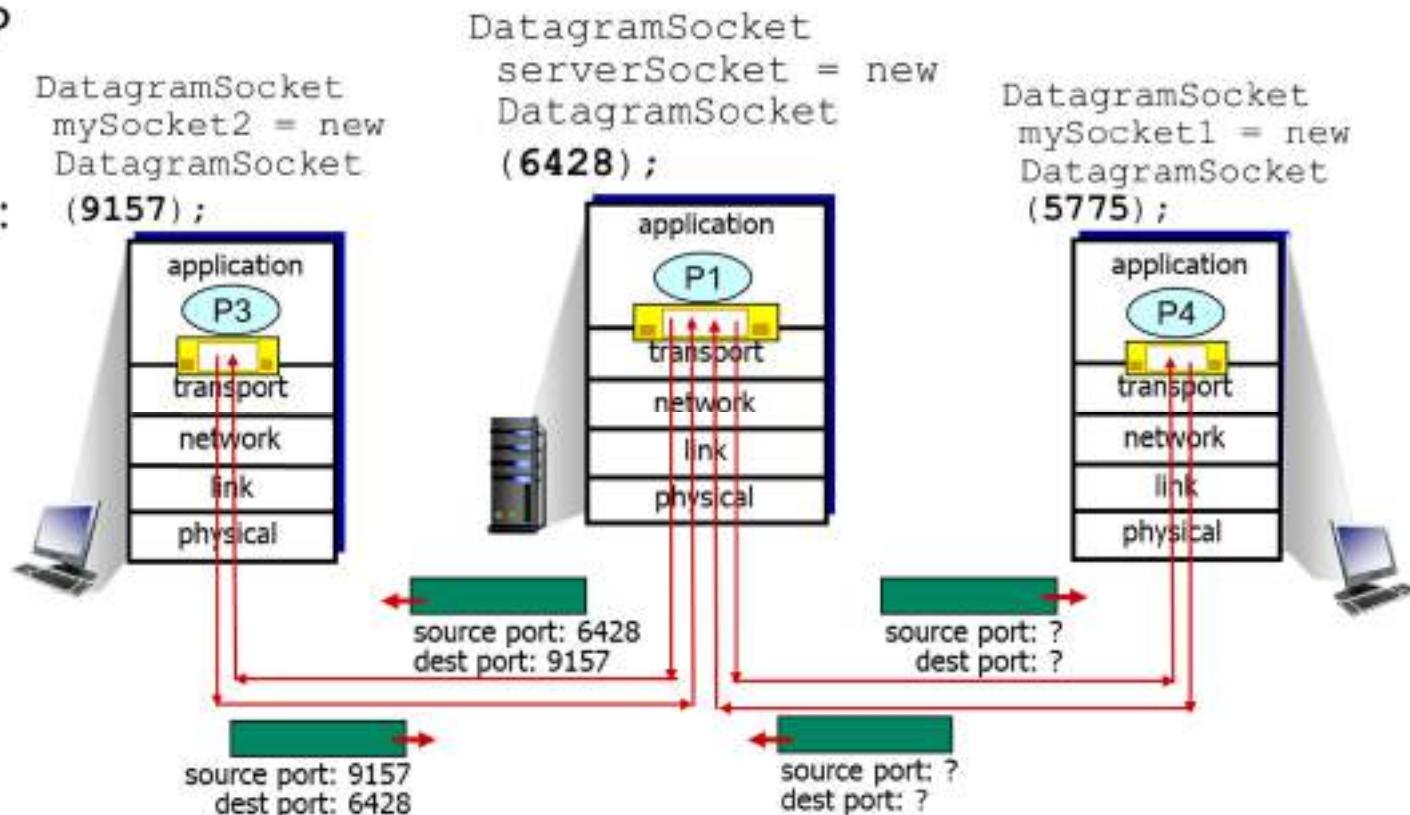
# Demultiplexing

- Host modtager et IP datagram
  - Hvert datagram har et source IP adresse, dest IP adresse
  - Hvert datagram bærer ét transport-lags segment
  - Hvert transport-lags segment har en source og destinations port
- Host bruger **IP-adresse og port numre** til at videreføre data til den tiltænkte socket



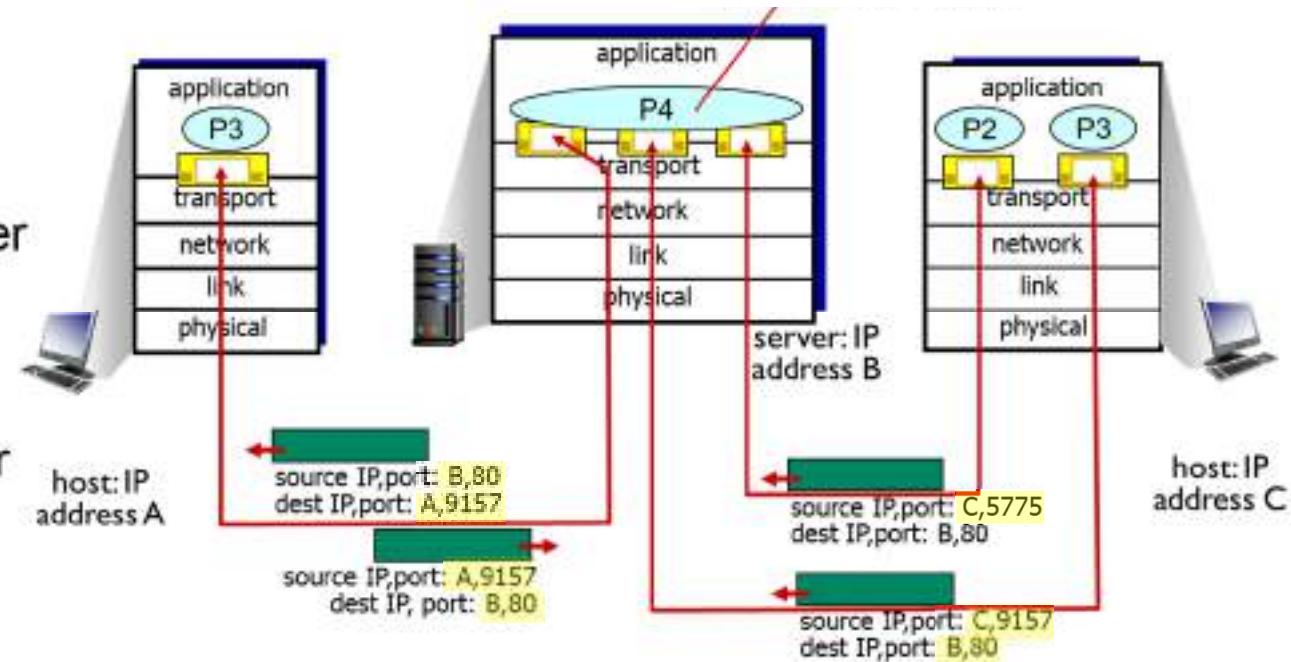
# De-multiplexing i UDP

- Modtager socket  
identificeres ved modtager IP  
og porte  
**(dest IP, dest port)**
- Modtagelse af UDP-segment:
  - Checker for om der findes en aktiv socket på dest-port
  - Videresender data til denne
- NB! 2 UDP segmenter med samme dest, men forskellig src leveres til samme socket
- NB! Source IP og port skal kendes hvis sender skal kunne besvare modtager



# Demultiplexing i TCP

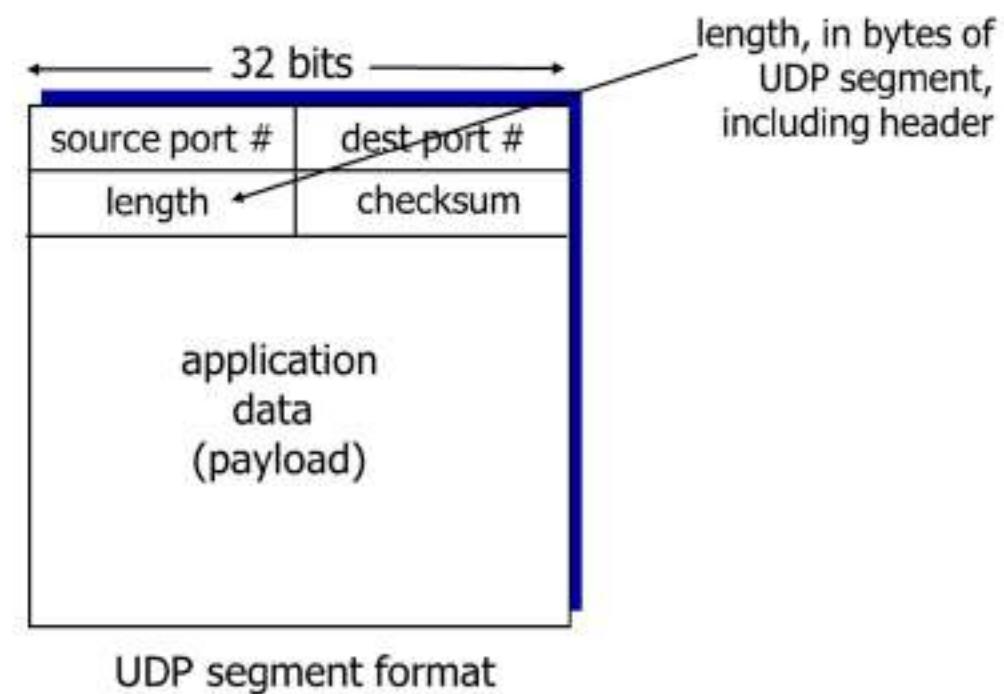
- TCP socket identificeres med 4-tupel \*):
  - **source IP adresse**
  - **source port nummer**
  - **dest IP adresse**
  - **dest port nummer**
- demux: modtager bruger alle 4 værdier for at videresende til den tiltænkte socket
- server proces kan have mange samtidige TCP forbindelser (fx, til hver web-klient):
  - Hver socket identificeres med sin egen 4-tupel



\*) når forbindelsen er etableret.

# UDP-transport

- Ingen ventetid på etablering af forbindelser
- Ingen congestion kontrol
- Meget simpel og lille header
  - Lavt overhead
- Checksum!
  - Støj på linien kan ”flippe” en eller flere bits
    - 01011 modtages som **11011**
  - Sender beregner en checksum på sendte data
  - Inkluderer dem i segmentet
  - Modtager beregner checksum på modtagne data
  - Pakken formodes at være korrekt modtaget hvis modtaget og beregnet checksum stemmer overens



# UDP Checksum

- UDP segment betragtes som en serie af 16 bit tal
  - Senderen:
    - Summerer serien af 16-bit tal efter one's complement metoden
    - Tager komplementet til denne sum; bruger resultatet som checksum
  - På modtageren
    - Summerer igen alle 16-bit tal, inkl. checksummen
    - Sum bør give 1111111111111111
    - Hvis ja: **sandsynlighed for at pakken er fejl-ramt er mindsket!**

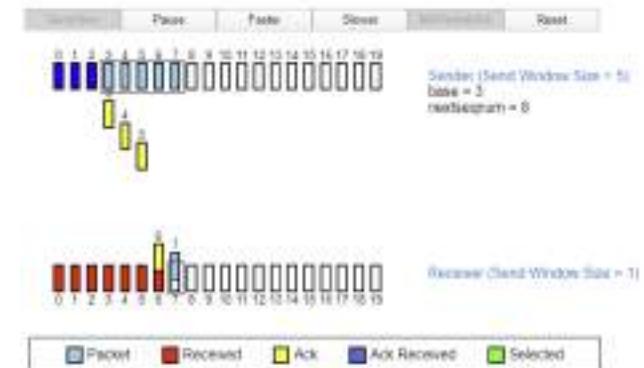
# Opgaverne idag

- Review: Har man forstået grundlæggende begreber
  - Demultiplexing,
  - Retransmissionsstrategier: Gå-N-tilbage, selektiv retransmission,
- Øvelser:
  - Styring af protokol-tilstandsmaskine: alt-bit
  - Sliding windows
- Praktiske: Kan anvende netværksværktøjerne
  - Portscan med NMAP: hvilke applikationer lytter på en given HOST?

## Prøv simulator

### Go-Back-N Protocol.

This interactive animation brings to life the Go-Back-N protocol. In this demo, the sending window is between sender and receiver. To simulate loss, select a moving data packet or ack, and then press "Retransmission" (i.e., timeout).



**SLUT**

# Internetværk og Web-programmering

## TCP and Socket Programming

Lecture 12  
Michele Albano

Distributed, Embedded, Intelligent Systems



# Last lecture you saw:

- Transport Layer Protocols:
  - Reliable Data Transfer 1.0 to 3.0
  - Pipelined protocols
  - Go-back-N vs Selective Repeat
- TCP / UDP
  - Protocol header
  - 3 ways handshake
  - Multiplexing / demultiplexing
- Don't forget to evaluate the course

# Agenda

## 3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control

## 3.6 principles of congestion control

## 3.7 TCP congestion control

## 2.7 Socket programming / [DF] 16.9 Javascript sockets

# Transmission Control Protocol:

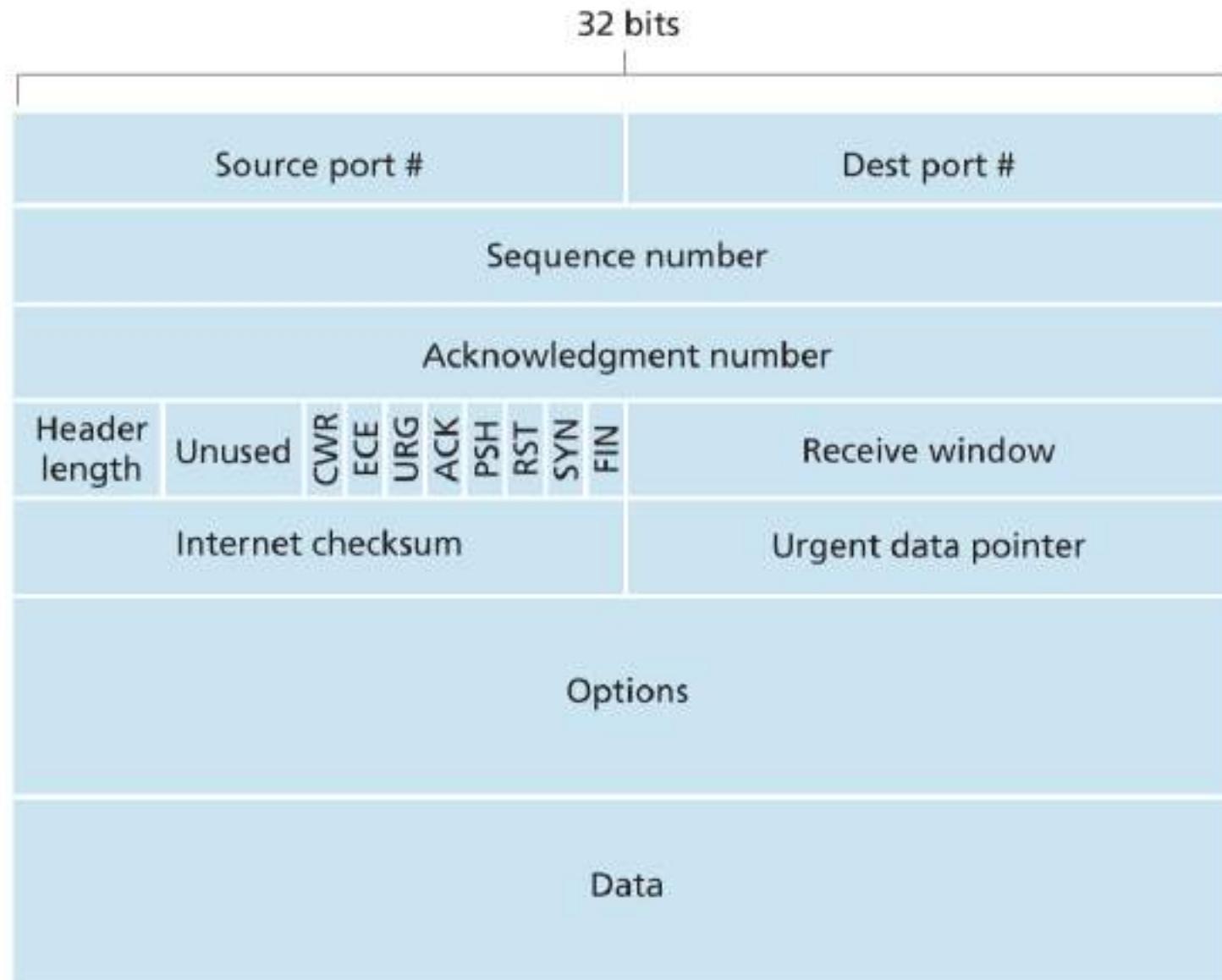
- point-to-point:
  - one sender, one receiver
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- reliable, in-order byte stream:
  - no “message boundaries”
- connection-oriented:
  - handshaking (exchange of control msgs) initializes sender, receiver state before data exchange
- pipelined:
  - TCP congestion and flow control set window size
- flow controlled:
  - sender will not overwhelm receiver

# TCP segment structure 1/2

- Src/dst ports
  - TCP and UDP have different namespaces
- Seq/ack numbers to the byte

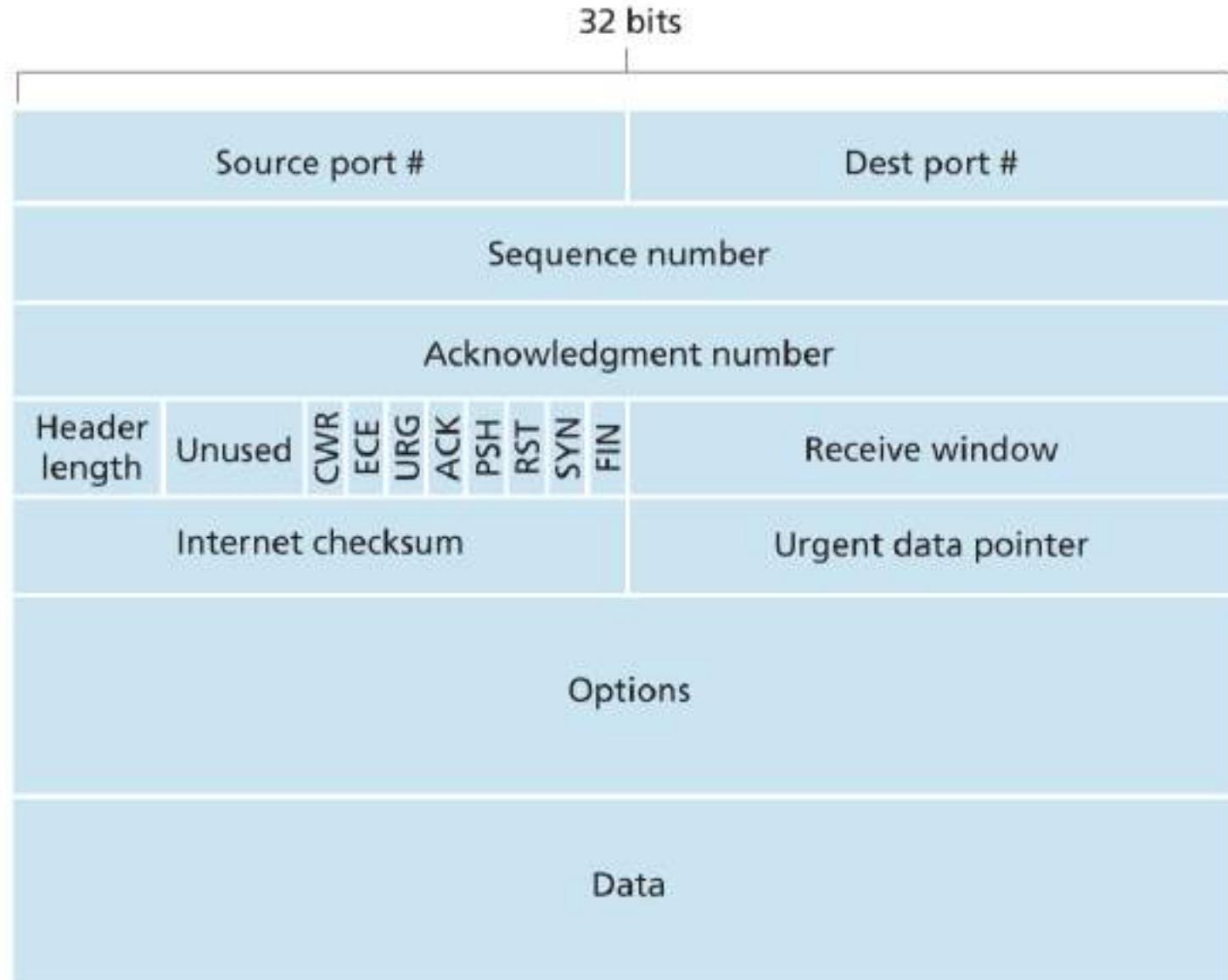
Flags:

- CWR and ECE – explicit congestion notification
- URG and PSH – urgent data
- SYN and FIN seen in lecture 11
- ACK is 1 if the ACK number is valid
- RST to kill the connection



# TCP segment structure 2/2

- Header length – needed since “Options” has variable length
- **Receive window: for flow control**
- Internet checksum: 16-bit checksum for TCP header, payload, and some data from IP (routing layer)
- Urgent data pointer: used with the URG flag
- Options: for example to set the maximum size of the segments, or timestamp



# TCP Sequence Numbers, ACKs 1/2

## sequence numbers:

- byte stream “number” of first byte in segment’s data

## acknowledgements:

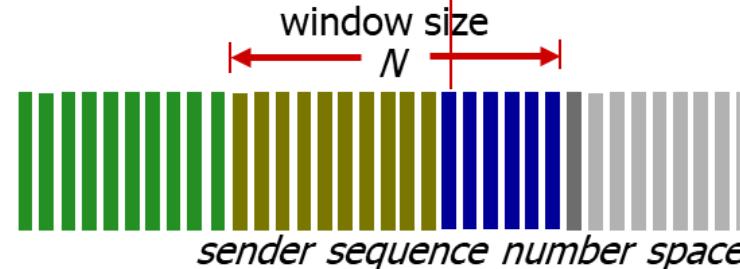
- seq # of next byte expected from other side
- cumulative ACK

How receiver handles out-of-order segments?

- TCP specs doesn’t say
- implementation dependent

outgoing segment from sender

source port #	dest port #
sequence number	
acknowledgement number	
	rwnd
checksum	urg pointer



sent  
ACKed

sent, not-  
yet ACKed  
("in-  
flight")

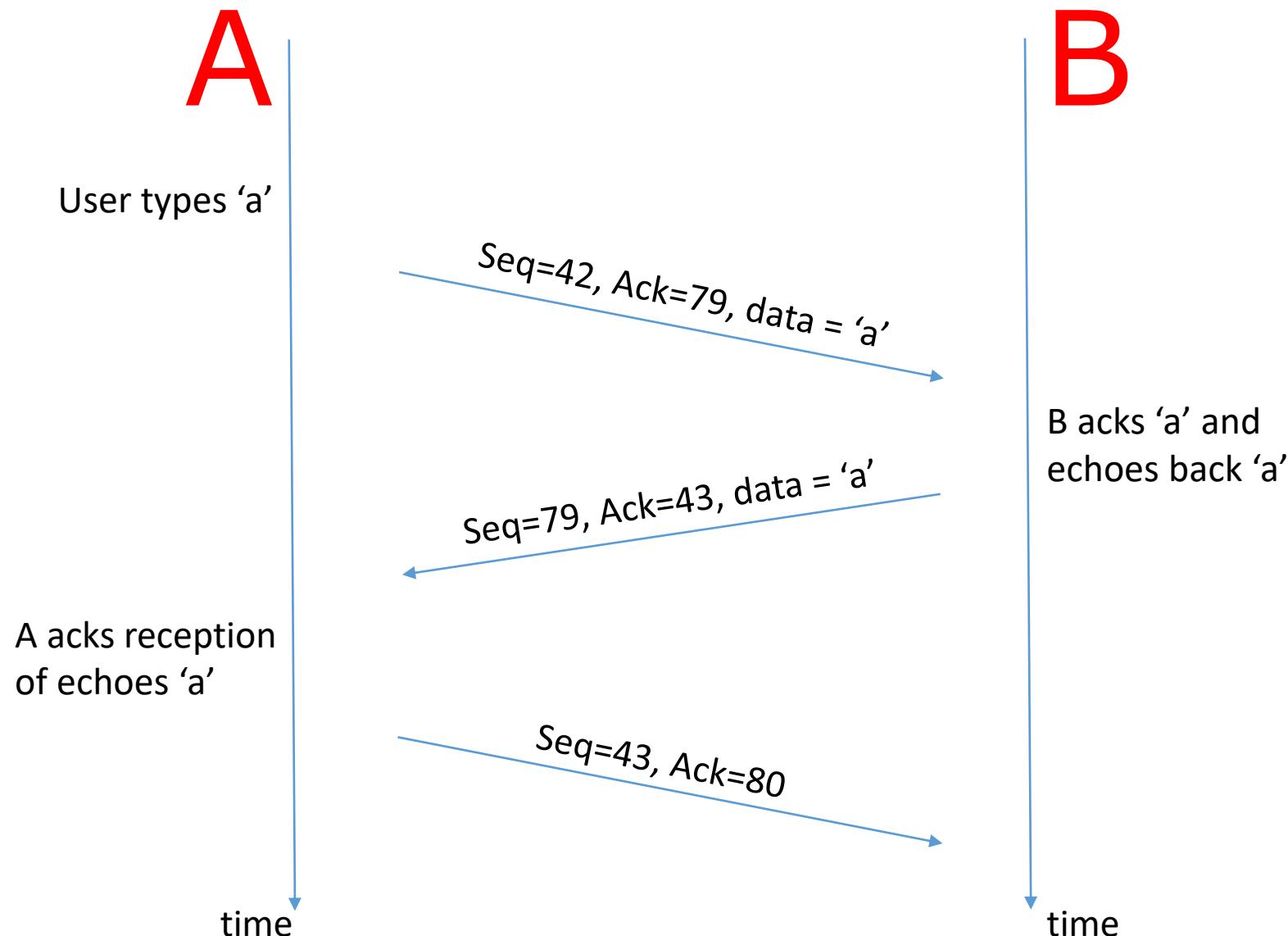
usable  
but not  
yet sent

not  
usable

incoming segment to sender

source port #	dest port #
sequence number	
acknowledgement number	
	A
checksum	rwnd
urg pointer	

# TCP Sequence Numbers, ACKs 2/2



# Agenda

## 3.5 connection-oriented transport: TCP

- segment structure
- **reliable data transfer**
- flow control

## 3.6 principles of congestion control

## 3.7 TCP congestion control

## 2.7 Socket programming / [DF] 16.9 Javascript sockets

# TCP Reliable Data Transfer

- TCP creates rdt service on top of IP's unreliable service
  - pipelined segments
  - cumulative acks
  - single retransmission timer
- retransmissions triggered by:
  - timeout events
  - duplicate acks
- let's initially consider simplified TCP sender:
  - ignore duplicate acks
  - ignore flow control, congestion control

## Timeout: how long?

- longer than RTT (but RTT varies)
- **too short**: premature timeout, unnecessary retransmissions
- **too long**: slow reaction to segment loss

# TCP Sender Events:

## **data received from app layer:**

- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unacked segment
  - expiration interval:  
**TimeOutInterval**

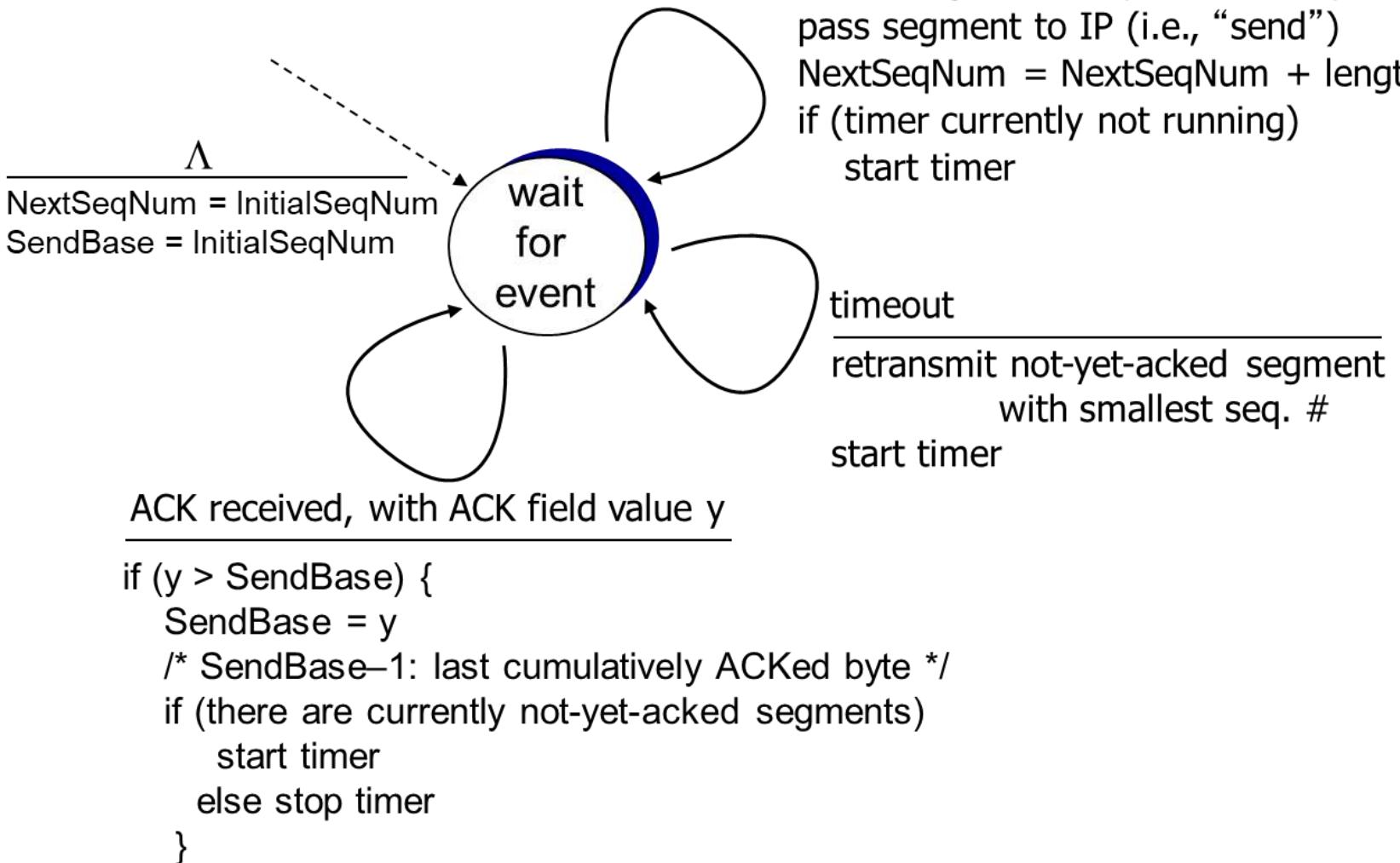
## **timeout:**

- retransmit segment that caused timeout
- restart timer

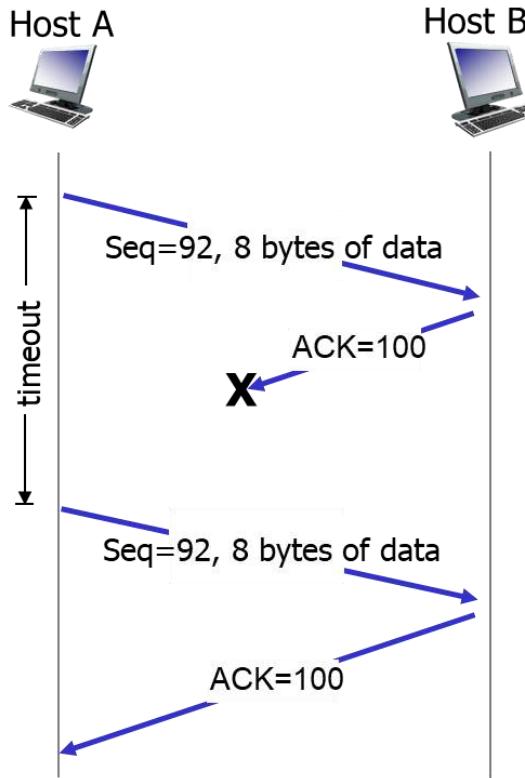
## **ack received:**

- if ack acknowledges previously unacked segments
  - update what is known to be ACKed
  - start timer if there are still unacked segments

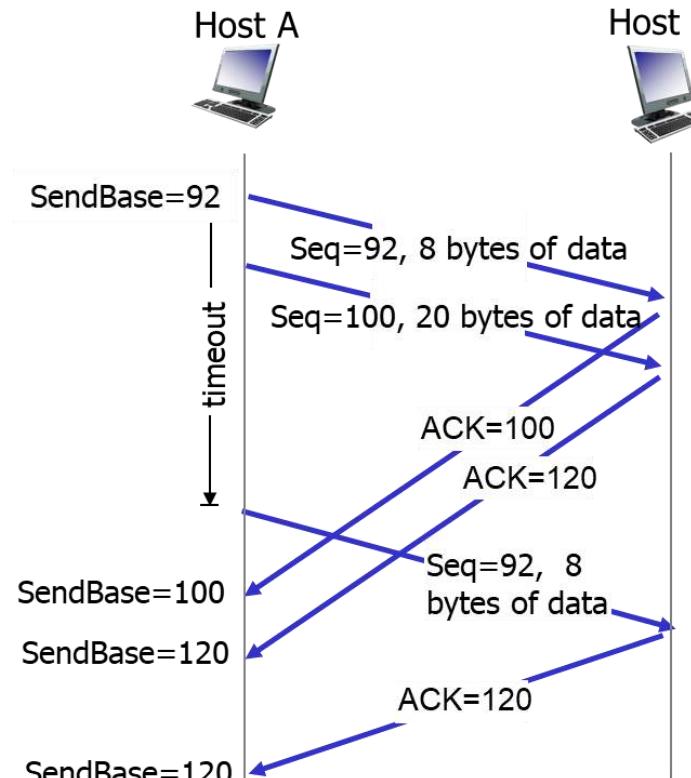
# TCP Sender (Simplified)



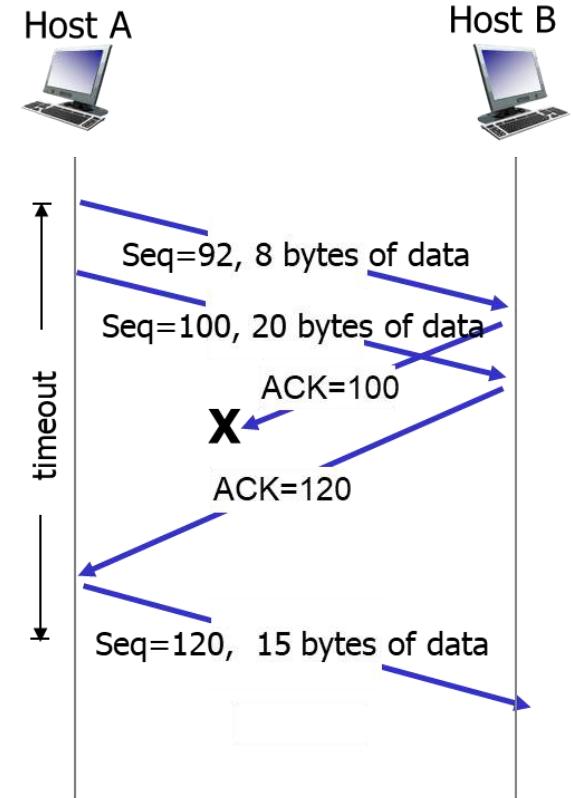
# TCP Retransmission Scenarios



lost ACK scenario



premature timeout



cumulative ACK

# TCP ACK Generation [RFC 1122, RFC2581]

event at receiver	TCP receiver action
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expect Sequence # . Gap detected	immediately send <b>duplicate ACK</b> , indicating Sequence # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap

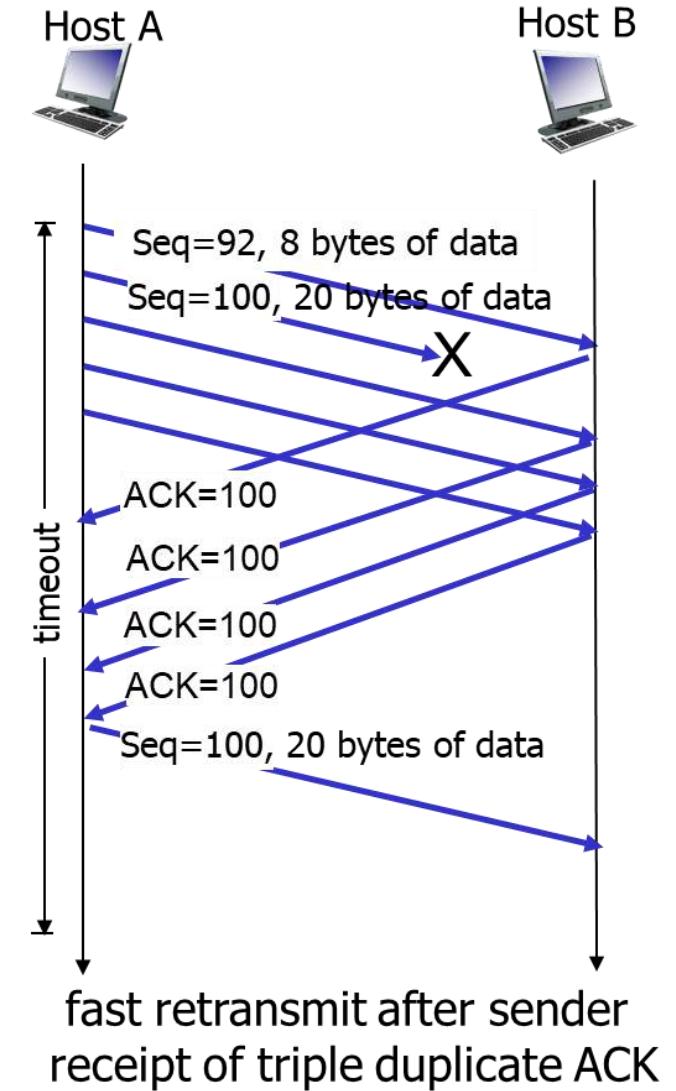
# TCP Fast Retransmit

- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs

## TCP fast retransmit

if sender receives 3 ACKs for same data (“triple duplicate ACKs”),  
resend unacked segment with smallest seq #

- likely that unacked segment lost, so don’t wait for timeout



# Agenda

## 3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- **flow control**

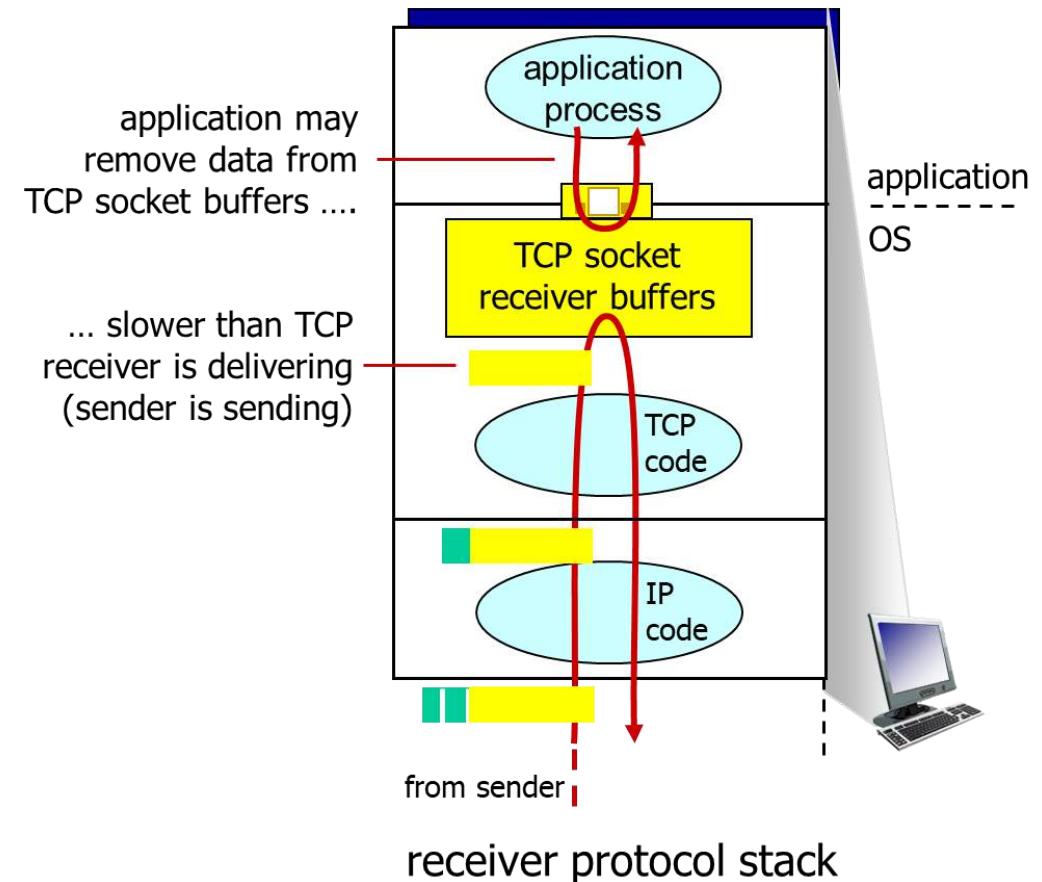
## 3.6 principles of congestion control

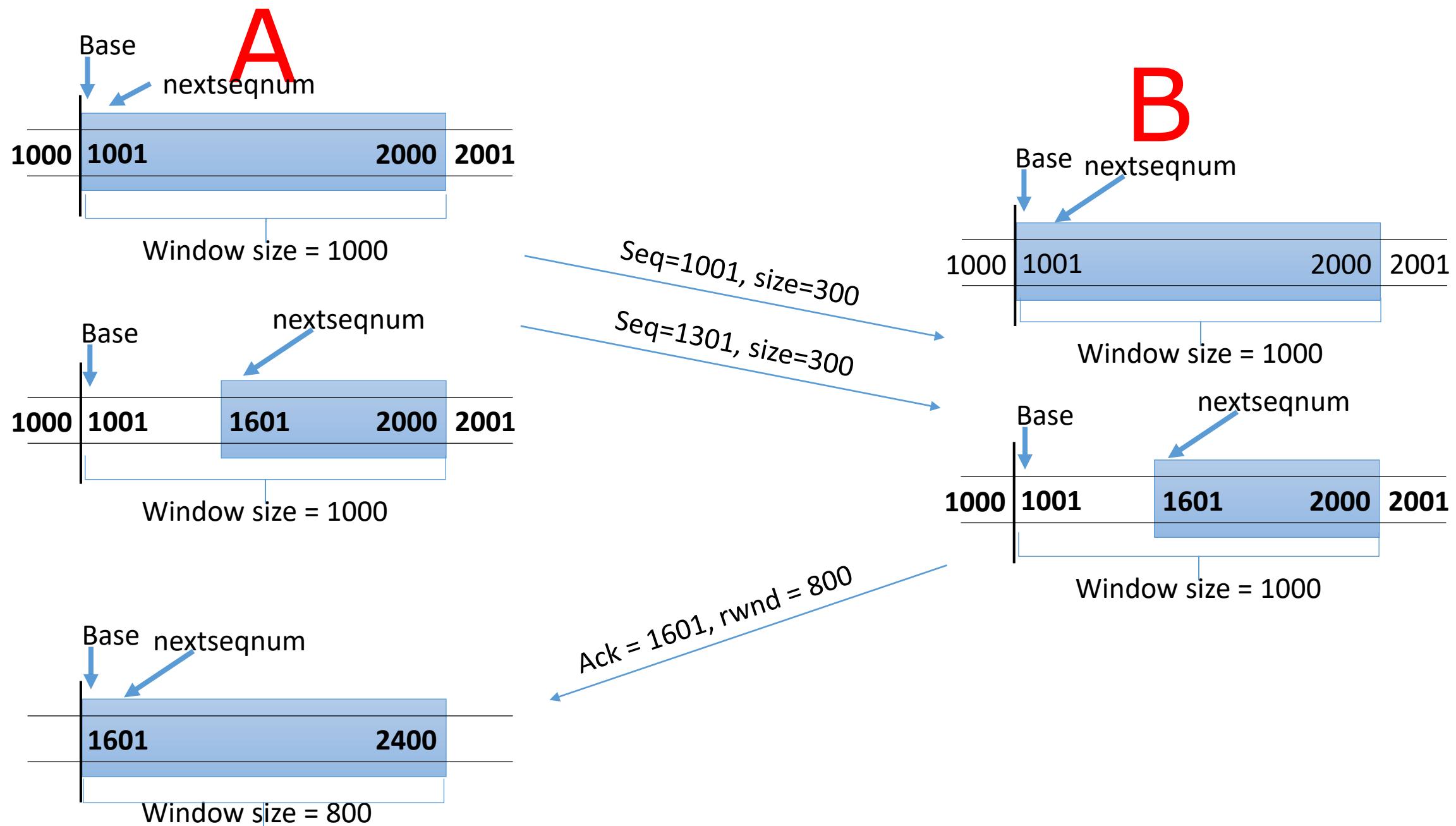
## 3.7 TCP congestion control

## 2.7 Socket programming / [DF] 16.9 Javascript sockets

# TCP Flow Control

- receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast
- receiver “advertises” free buffer space by including **rwnd** value in TCP header of receiver-to-sender segments
  - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **RcvBuffer**
- sender limits amount of unacked (“in-flight”) data to receiver's **rwnd** value
- guarantees receive buffer will not overflow





# Agenda

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control

3.6 principles of congestion control

3.7 TCP congestion control

2.7 Socket programming / [DF] 16.9 Javascript sockets

# Principles of Congestion Control

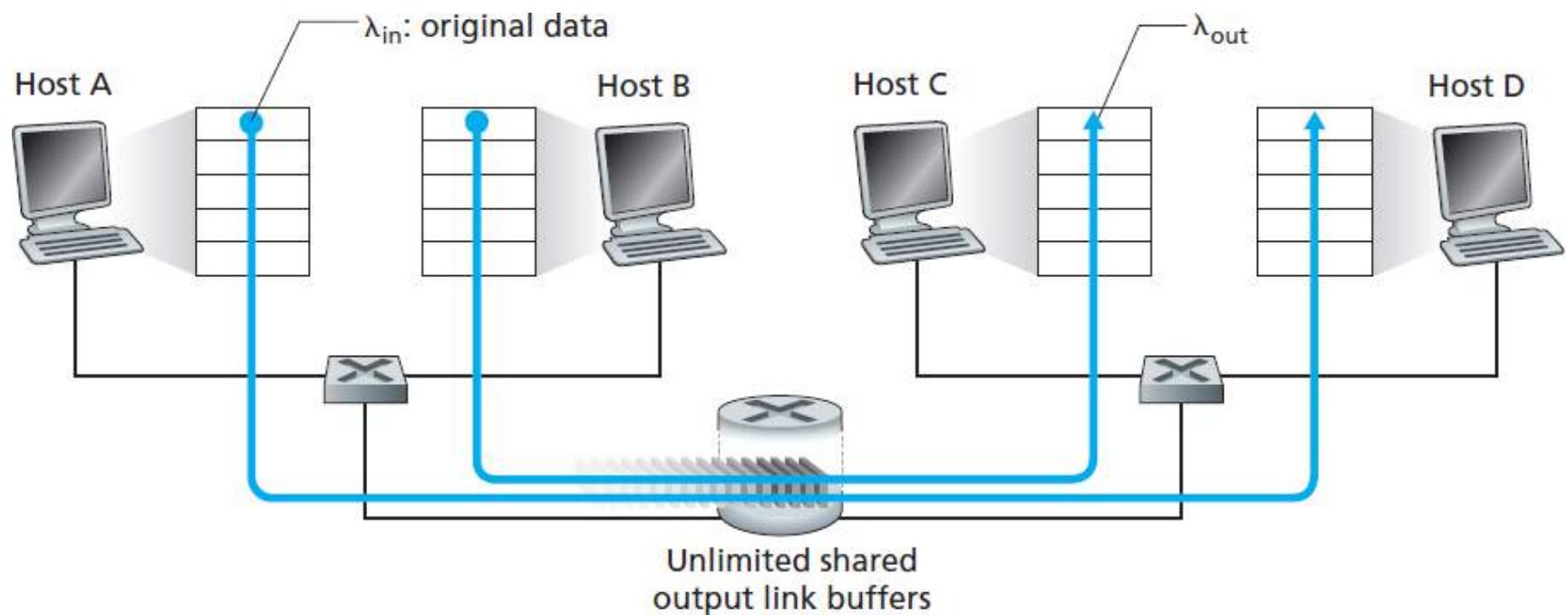
## congestion:

- informally: “too many sources sending too much data too fast for **network** to handle”
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- flow control vs congestion control
  - buffers of the receiver vs network resources
  - knowing the state of the receiving buffers vs estimating the state of the network

# Causes/Costs of Congestion: Scenario 1

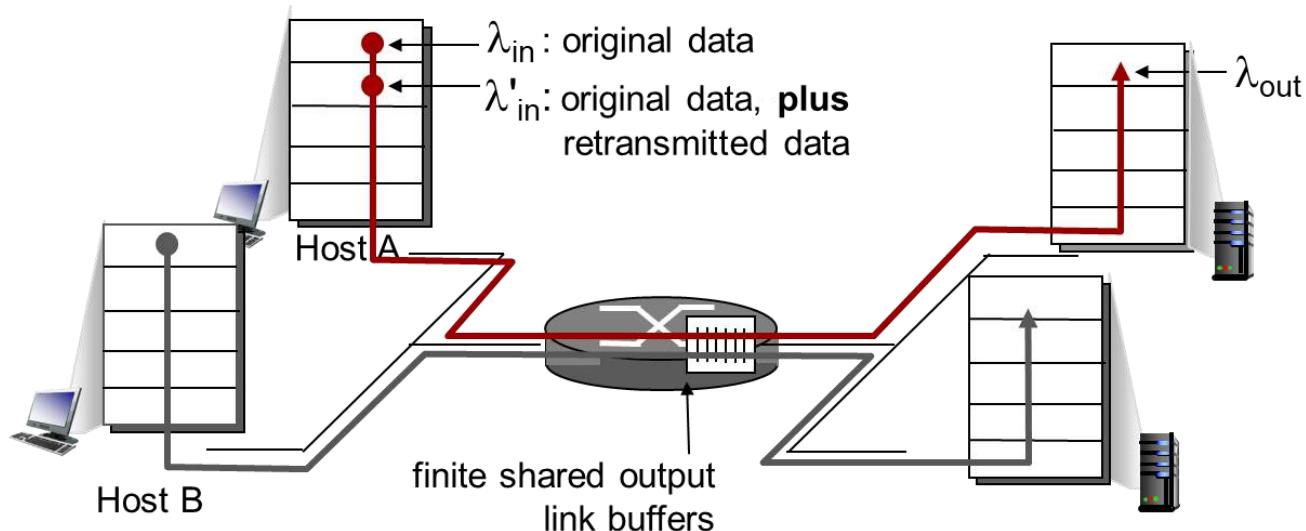
- two senders, two receivers
- one router, infinite buffers
- output link capacity:  $R$
- no retransmission

The capacity of the link must be split to all the data flows



# Causes/Costs of Congestion: Scenario 2 (1/2)

- one router, **finite** buffers
- sender retransmission of timed-out packet
  - application-layer input = application-layer output:  $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes **retransmissions** :  $\lambda'_{in} \geq \lambda_{in}$



## Idealizations:

- **perfect knowledge**
  - sender sends only when router buffers available
- **known loss** packets can be lost, dropped at router due to full buffers
  - sender only resends if packet **known** to be lost

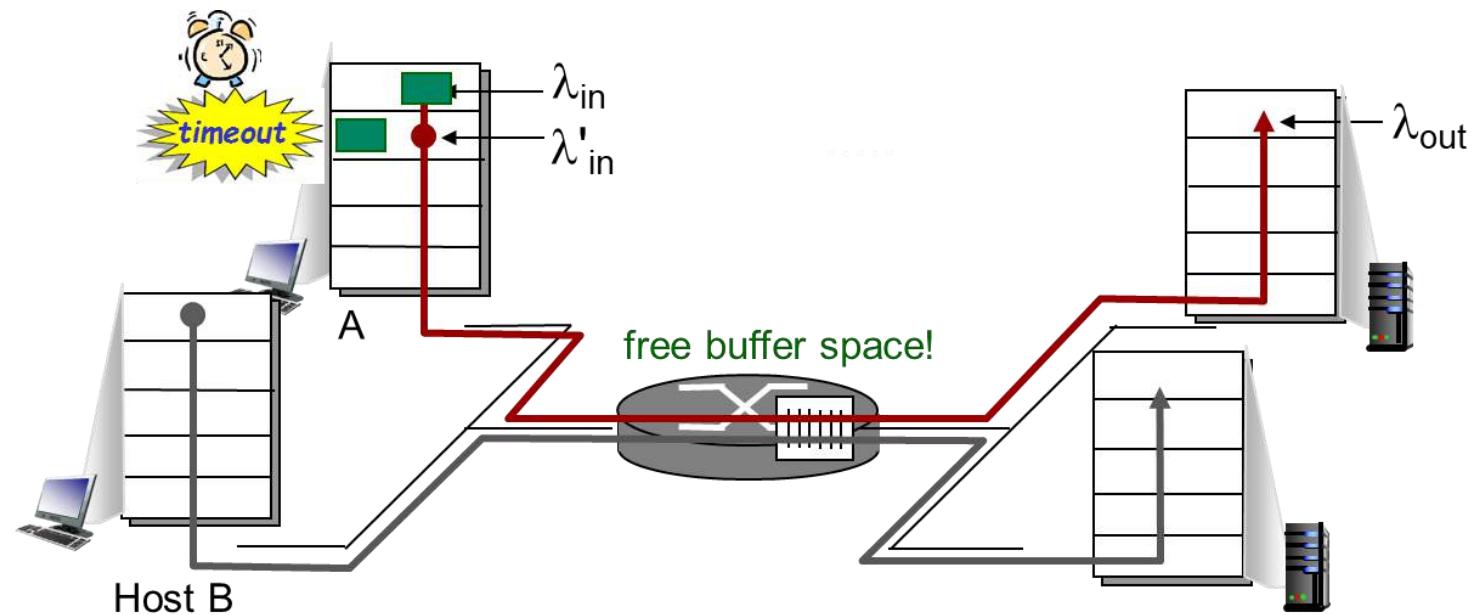
# Causes/Costs of Congestion: Scenario 2 (2/2)

## Realistic: duplicates

- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending **two** copies, both of which are delivered

## “costs” of congestion:

- more work (retrans) for given “goodput”
- unneeded retransmissions: link carries multiple copies of pkt
  - decreasing goodput

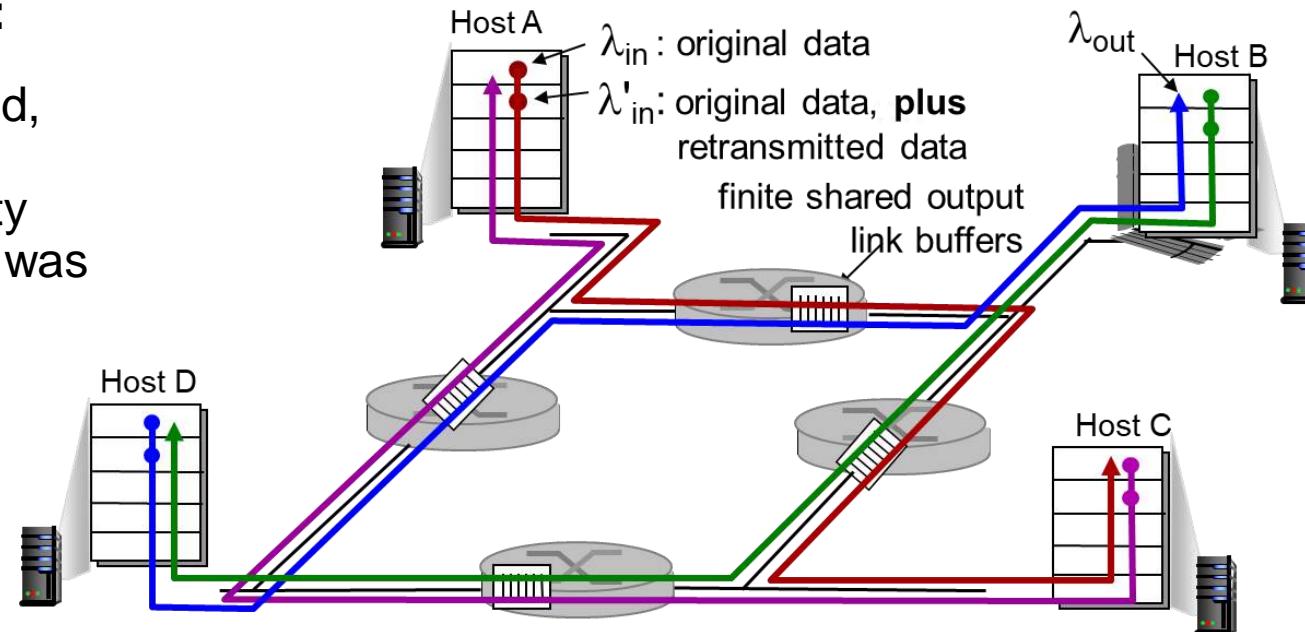


# Causes/Costs of Congestion: Scenario 3

- four senders
  - multihop paths
  - timeout/retransmit
- Q: what happens as  $\lambda_{in}$  and  $\lambda'_{in}$  increase ?
  - A: as red  $\lambda'_{in}$  increases, all arriving blue pkts at upper queue are dropped, blue throughput  $\rightarrow 0$

## “cost” of congestion:

- when packet dropped, any upstream transmission capacity used for that packet was wasted



# Agenda

## 3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control

## 3.6 principles of congestion control

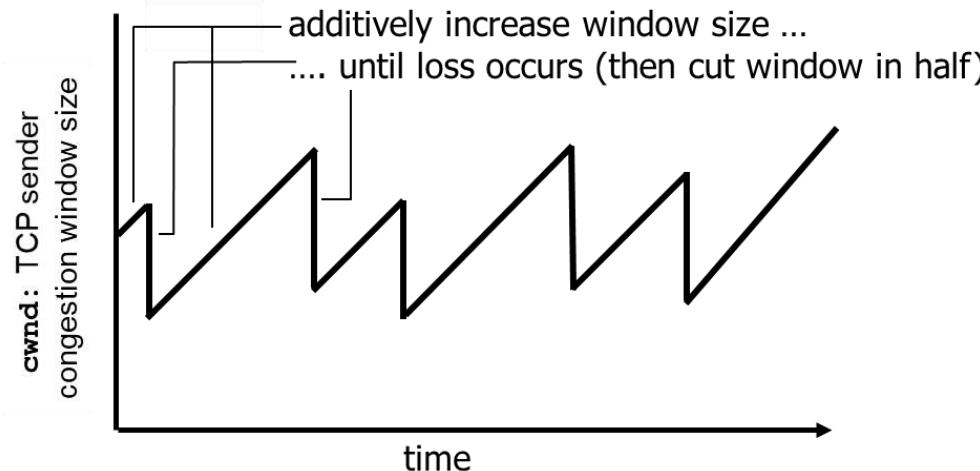
## 3.7 TCP congestion control

## 2.7 Socket programming / [DF] 16.9 Javascript sockets

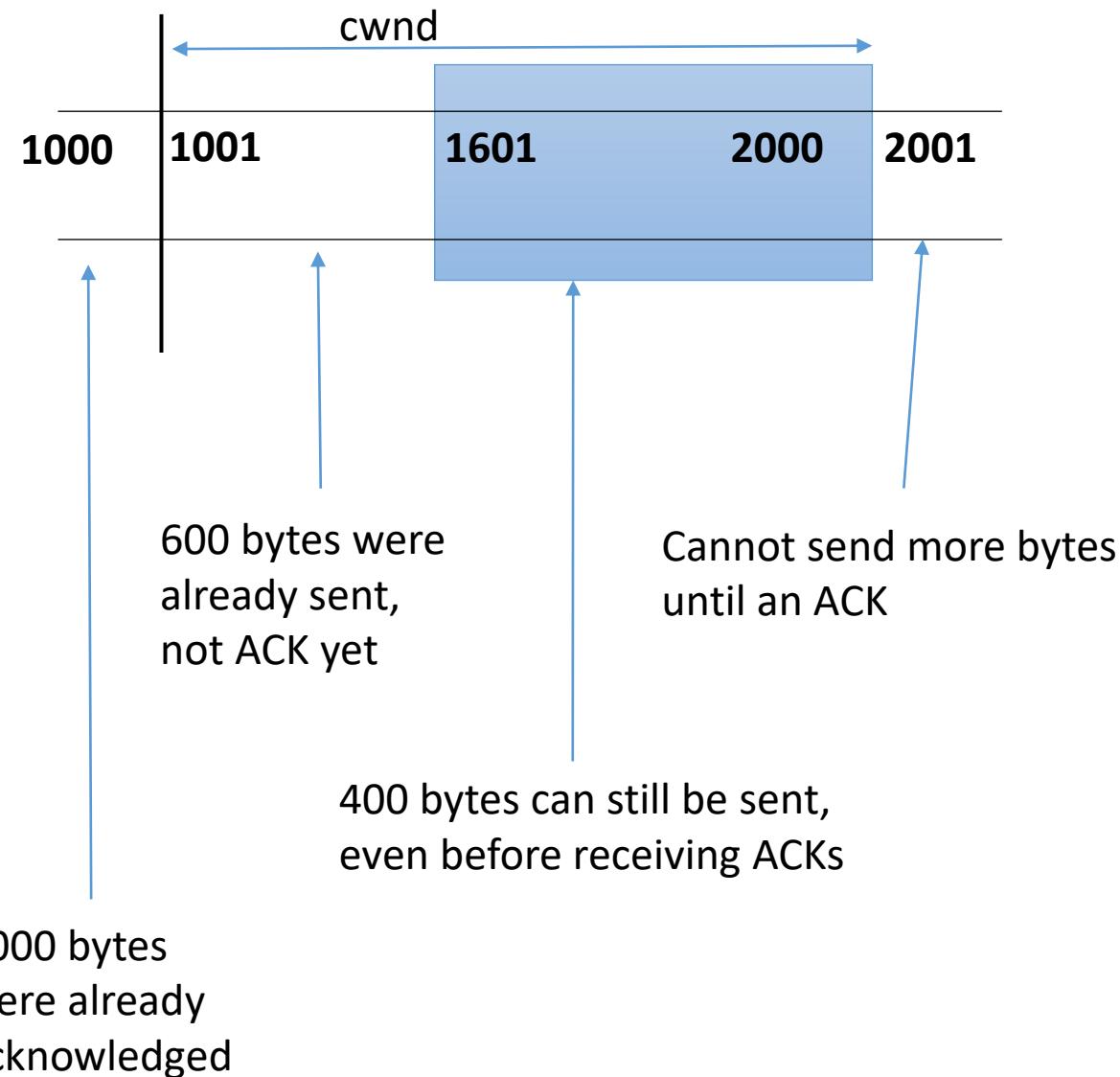
# TCP Congestion Control: Additive Increase Multiplicative Decrease

- **Mechanism:** changing  $cwnd$
- sender *increases* the window size, probing for usable bandwidth, until loss occurs
  - **additive increase:** increase  $cwnd$  by 1 MSS every RTT until loss detected
  - **multiplicative decrease:** cut  $cwnd$  in half after loss

AIMD saw tooth behavior: probing for bandwidth



# TCP Congestion Control: the sender



- sender limits transmission
- **cwnd** is dynamic, function of perceived network congestion

## TCP sending rate:

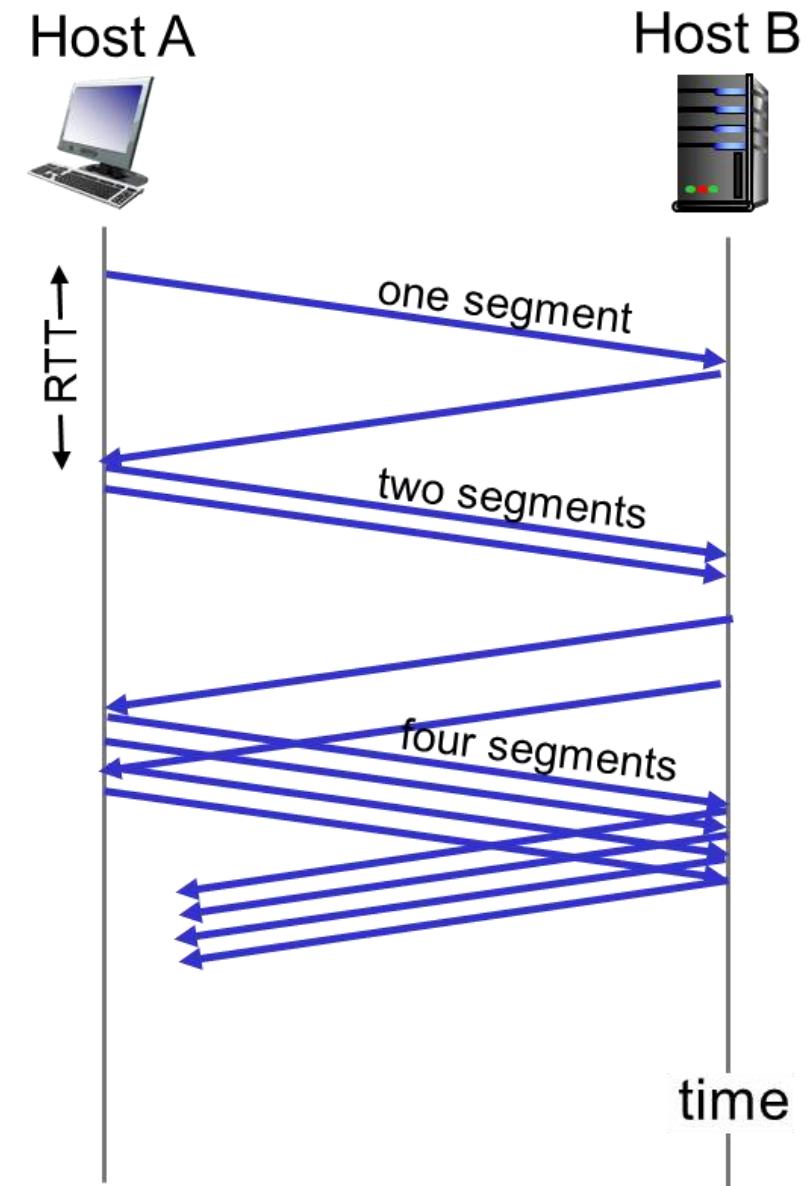
- **roughly:** send cwnd bytes, wait RTT for ACKS, then send more bytes

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

$$\frac{\text{LastByteSent} - \text{LastByteAcked}}{\text{RTT}} \leq \text{cwnd}$$

# TCP Slow Start

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received
- initial rate is slow but ramps up exponentially fast

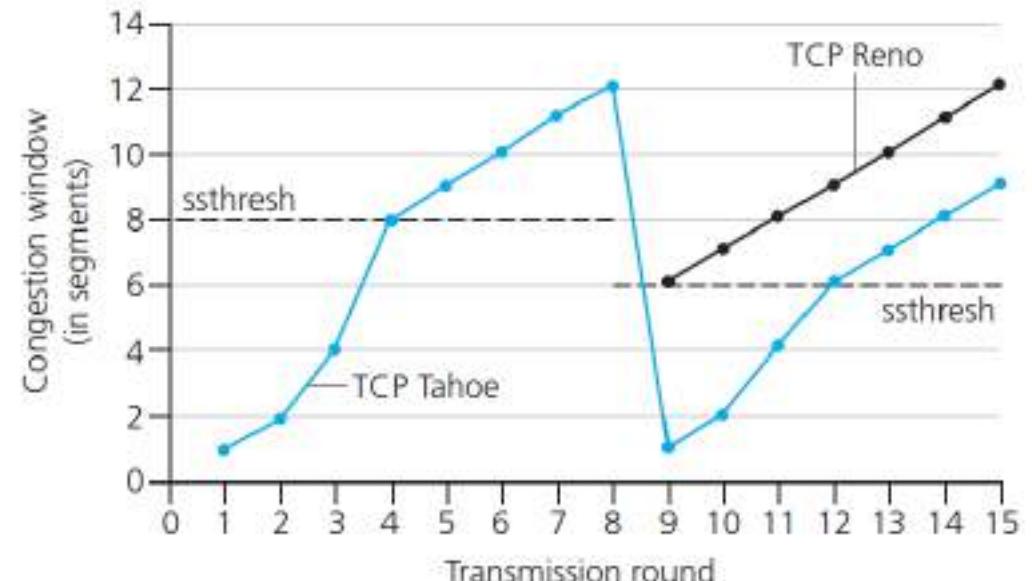


# TCP: Detecting, Reacting to Loss

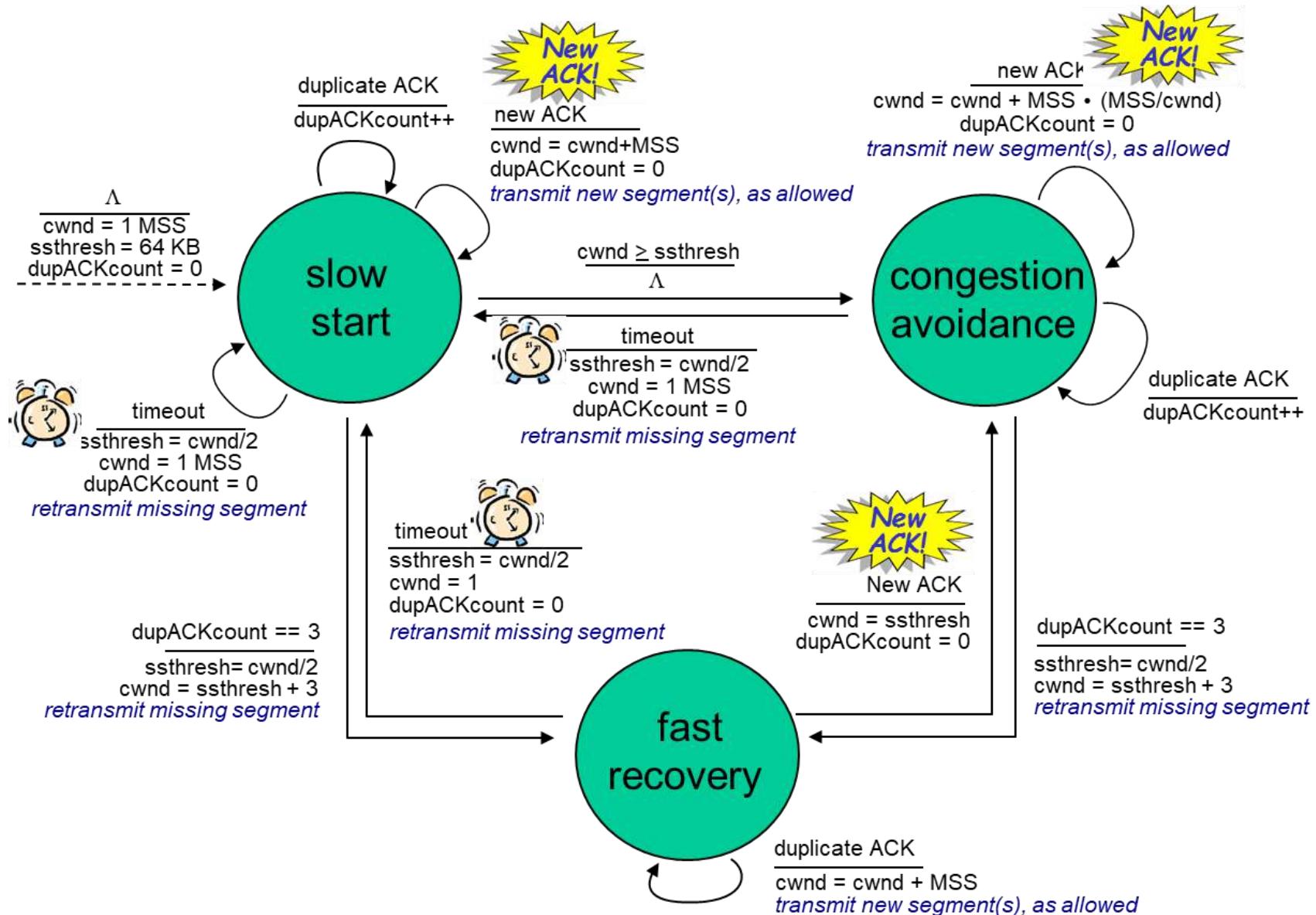
- loss indicated by timeout:
  - **cwnd** set to 1 MSS;
  - window then grows exponentially (as in slow start) to threshold (a configuration parameter, controlled by the sender\*), then grows linearly
- loss indicated by 3 duplicate ACKs: TCP RENO
  - dup ACKs indicate network capable of delivering some segments
  - **cwnd** is cut in half window then grows linearly
- TCP Tahoe always sets **cwnd** to 1 (timeout or 3 duplicate acks)

\* Check out the online interactive exercises for more examples:

[http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)



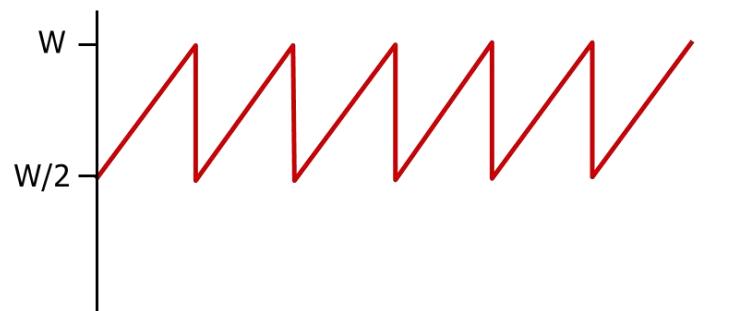
# Summary: TCP Congestion Control



# TCP Throughput

- average TCP throughput as function of window size, RTT?
  - ignore slow start, assume always data to send
- W: window size (measured in bytes) where loss occurs
  - average window size (# in-flight bytes) is
  - average throughput is  $\frac{3}{4}W \cdot \text{per RTT}$   $\frac{3}{4}W$

$$\text{avg TCP throughput} = \frac{3}{4} \frac{W}{\text{RTT}} \text{ bytes/sec}$$



# Fairness

**fairness goal:** if  $K$  TCP sessions share same bottleneck link of bandwidth  $R$ , each should have average rate of

## Fairness and UDP

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - send audio/video at constant rate, tolerate packet loss

## Fairness, parallel TCP connections

- application can open multiple parallel connections between two hosts
- web browsers do this
- example, link of rate  $R$  with 9 existing connections:
  - new app asks for 1 TCP, gets rate  $\frac{R}{10}$
  - new app asks for 11 TCPs, gets  $\frac{R}{2}$

# Agenda

## 3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control

## 3.6 principles of congestion control

## 3.7 TCP congestion control

## 2.7 Socket programming / [DF] 16.9 Javascript sockets

# The Socket

- Formed by the concatenation of a port value and an IP address
  - Unique throughout the Internet
- Used to define an API
  - Generic communication interface for writing programs that use TCP or UDP
- Stream sockets
  - All blocks of data sent between a pair of sockets are guaranteed for delivery and arrive in the order that they were sent
- Datagram sockets
  - Delivery is not guaranteed, nor is order necessarily preserved
- Raw sockets
  - Allow direct access to lower-layer protocols

# Defining a socket:

- Two main things to do
  - Addressing
    - Specifying a host and a service (IP + port)
    - It is a tuple ("www.cs.aau.dk", 80)
  - Data transport
    - Mainly TCP (SOCK\_STREAM) or UDP (SOCK\_DGRAM)

# Socket Programming with UDP

## **UDP: no “connection” between client & server**

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

## **UDP: transmitted data may be lost or received out-of-order**

### **Application viewpoint:**

- UDP provides **unreliable** transfer of groups of bytes (“datagrams”) between client and server

# Client/Server Socket Interaction: UDP

## server (running on serverIP)

```
create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)
```

read datagram from  
**serverSocket**

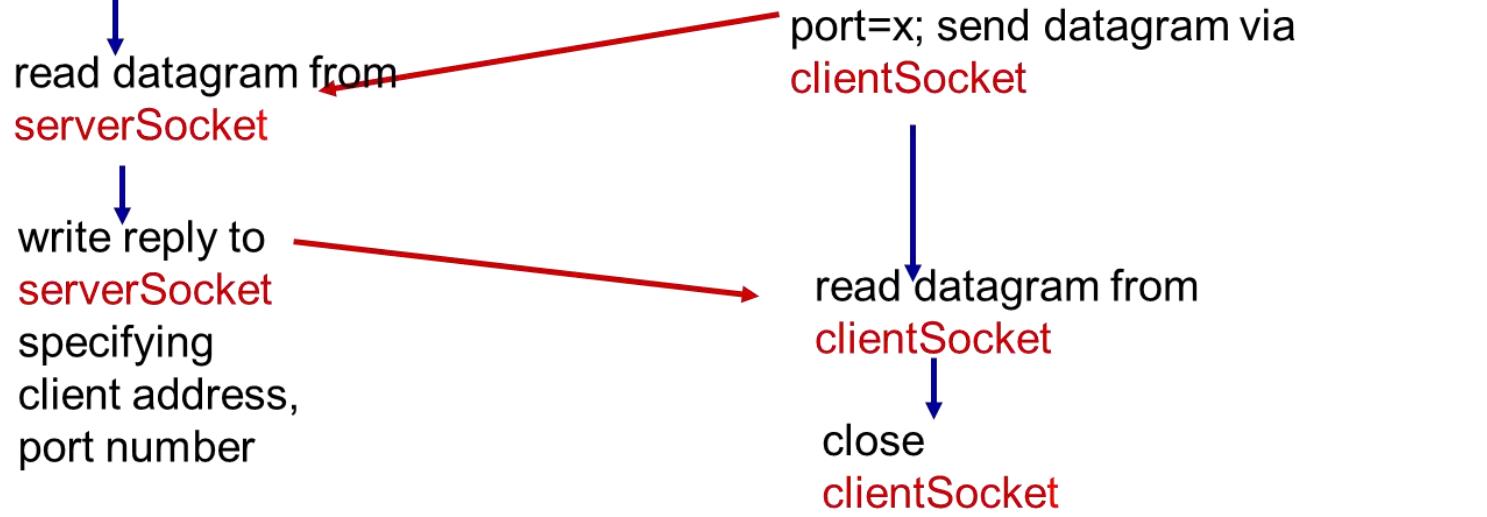
write reply to  
**serverSocket**  
specifying  
client address,  
port number

## client

```
create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)
```

Create datagram with server IP and  
port=x; send datagram via  
**clientSocket**

read datagram from  
**clientSocket**  
close  
**clientSocket**



# Socket Programming with TCP

## client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

## client contacts server by:

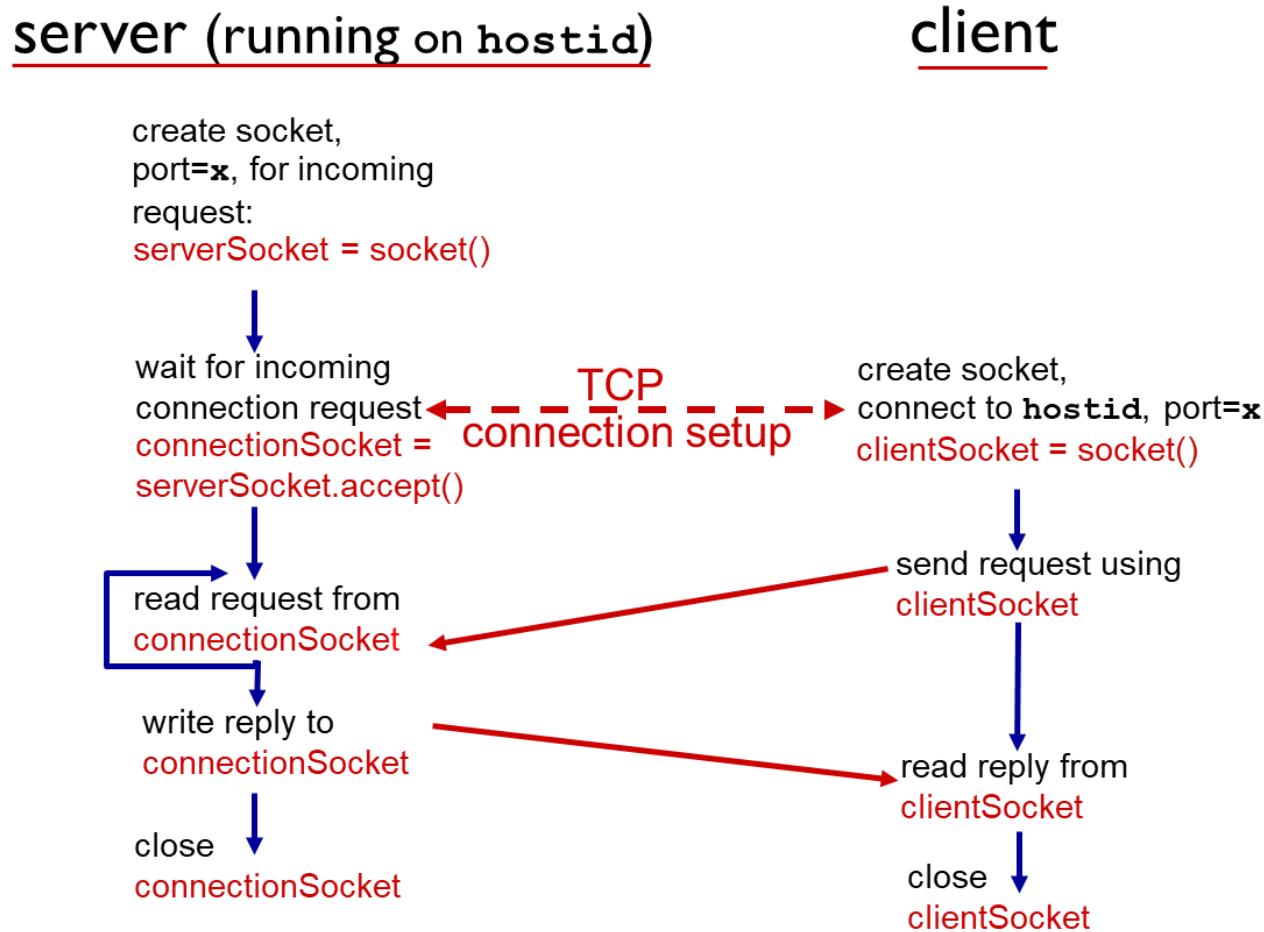
- Creating TCP socket, specifying IP address, port number of server process
- **when client creates socket:** client TCP establishes connection to server TCP

- when contacted by client, **server TCP creates new socket** for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

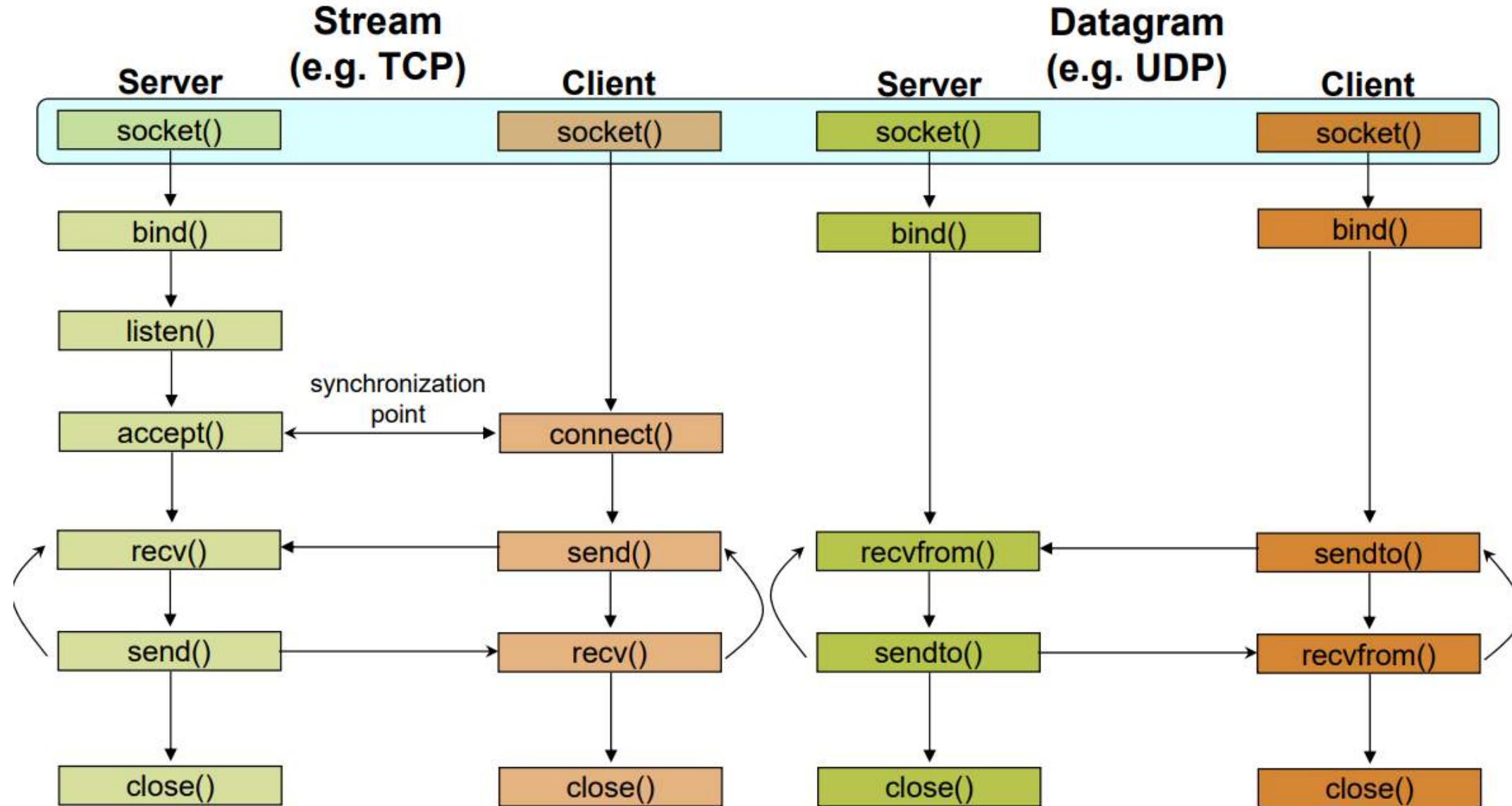
## application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

# Client/Server Socket Interaction: TCP



# The Socket API



Always check the return value of these calls !!!!

# C Server

1) Create a socket, bind it to a port, make it listen:

```
s = socket(AF_INET, SOCK_STREAM)
s.bind(my_address, 9000)
listen(s, 5) # up to 5 pending connections
```

2) Accept an incoming connection to have a connected socket:

```
c = accept(s, (struct sockaddr*)&client_addr,
&addrlen);
print("connected with %s", client_addr)
```

3) Communicate:

```
send(c, "you are connected\n", 19)
```

6) Close the Socket:

```
close(c);
```

7) Go to Step 2.

# C Client

1) Create a socket, connect it:

```
s = socket(AF_INET, SOCK_STREAM)
connect(s, destination)
```

3) Communicate (receive data):

```
recv(d, data, 1024)
printf("Received %s", data)
```

4) Close the socket:

```
close(s);
```

# Javascript Server

- 1) Import module “net”, create a socket, define what must be done all the time there is a connection:

```
const net = require('net');
const server = net.createServer((socket) => { ... });
```

- 2) The function is in an implicit infinite loop. Extract the address of the client, log, and answer to the client:

```
net.createServer((socket) => {
 addr = socket.address();
 console.log("%s:%d connected", addr.address, addr.port);
 socket.end("you are " + addr.address +
 " :" + addr.port + "\n");
}) ;
```

- 3) Bind and listen on the server socket:

```
server.listen(9999);
```

# Javascript Client

- 1) Import module “net”, create a socket:

```
const net = require('net');
const client = new net.Socket();
```

- 2) Connect the socket to the host provided by the command line:

```
client.connect({ port: 9999, host: process.argv[2] });
```

- 3) Communicate (receive data) and log:

```
client.on('data', (data) => {
 console.log("Received:\n" + data.toString('utf-8'));
});
```

# Windows programming

- Microsoft “version” of socket programming is called Winsock
- Same philosophy, implementation differences:
  - `WINSOCK_API_LINKAGE SOCKET WSAAPI socket( int af, int type, int protocol )`
    - It's not returning an integer!
- Need to initialize the Winsock subsystem:
  - `WSADATA wsaData;`
  - `iResult = WSASStartup( MAKEWORD( 2 , 2 ) , &wsaData );`
  - `if ( iResult != 0 ) { exit(-1); }`

# Issues:

- Reading/writing to a socket may involve partial data transfer
  - `send()` returns actual bytes sent
  - `recv()` length is only a maximum limit
- For TCP, the data stream is continuous---no concept of records, etc.
  - One `recv()` can return the strings sent in two `send()`
  - A `send()` can provide strings to two `recv()`s with a small maximum limit
- How to tell if there is no more data, in the sense that the connection has been (half-)closed?
  - `recv()` will return empty string
  - `ret = s.recv(1000)`
    - ... and `ret == ""`

# Issues with the recv()

- If a peer is doing a cycle like:

```
while (l = s.recv()) :
 do_something(l)
```

- There is the problem of termination:
  - Until `recv()` provides data, it doesn't end
  - When there is no more data, the program is stuck on the blocking `recv()` call
- Four solution:
  - Doing a `close()` on the other side
  - Send a *special* termination string
  - Use a timeout
  - Send the number of character that will be sent in the beginning

# Drawbacks of each solution

- Doing a `close()` on the other side
  - The `read()` will return (with a "" string)
  - But the socket cannot be used anymore
- Send a *special* termination string
  - But if it was part of the normal communication flow, it could quit the application too early
- Use a timeout
  - The application gets slow since it needs to wait until the timeout
  - Should the network be slow, the timeout could end the application too early
- Send the number of character that will be sent in the beginning
  - In some applications, you don't know the number of characters in advance

**SLUT**

# Internetværk og Web-programmering

## Security in computer networks

Lecture 13  
Michele Albano

Distributed, Embedded, Intelligent Systems



# Agenda

**8.1 What is network security?**

**8.2 Principles of cryptography**

**8.4 Authentication**

**8.3 Message integrity**

**8.6 Securing TCP connections: SSL**

**8.7 Network layer security: IPsec**

# What is Network Security?

**confidentiality:** only sender, intended receiver should “understand” message contents

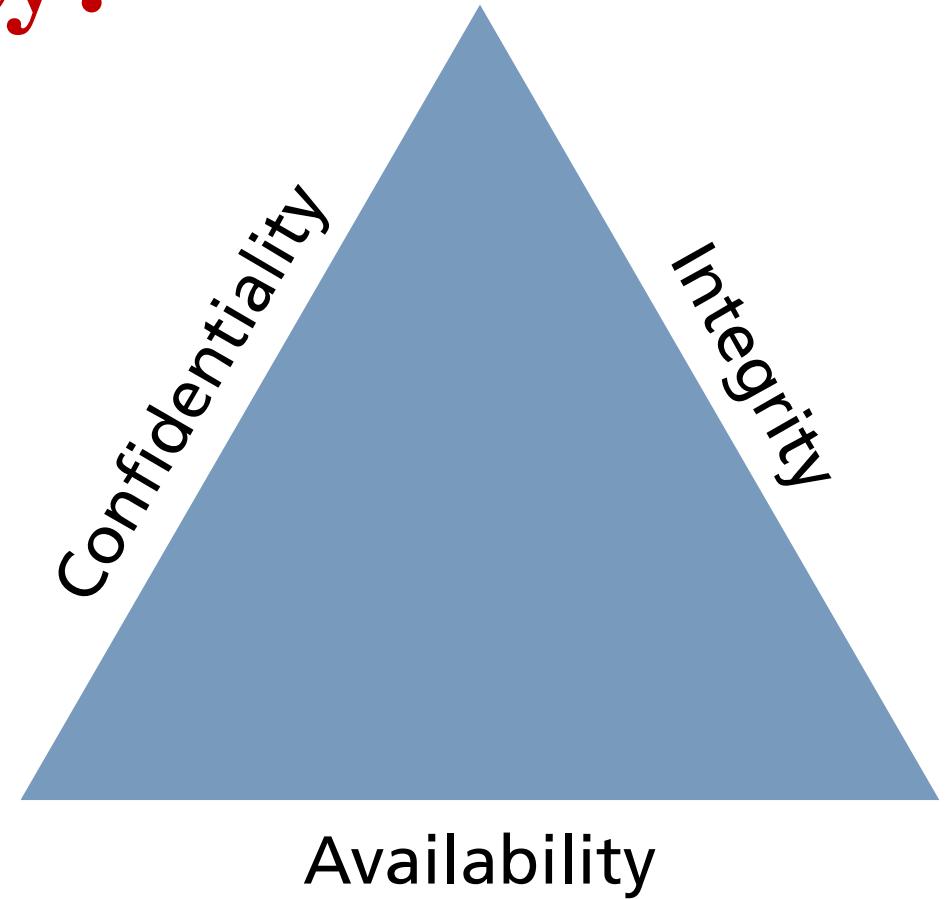
- sender encrypts message
- receiver decrypts message

**authentication:** sender, receiver want to confirm identity of each other

**authorization:** each authenticated user must be able to do a set of tasks

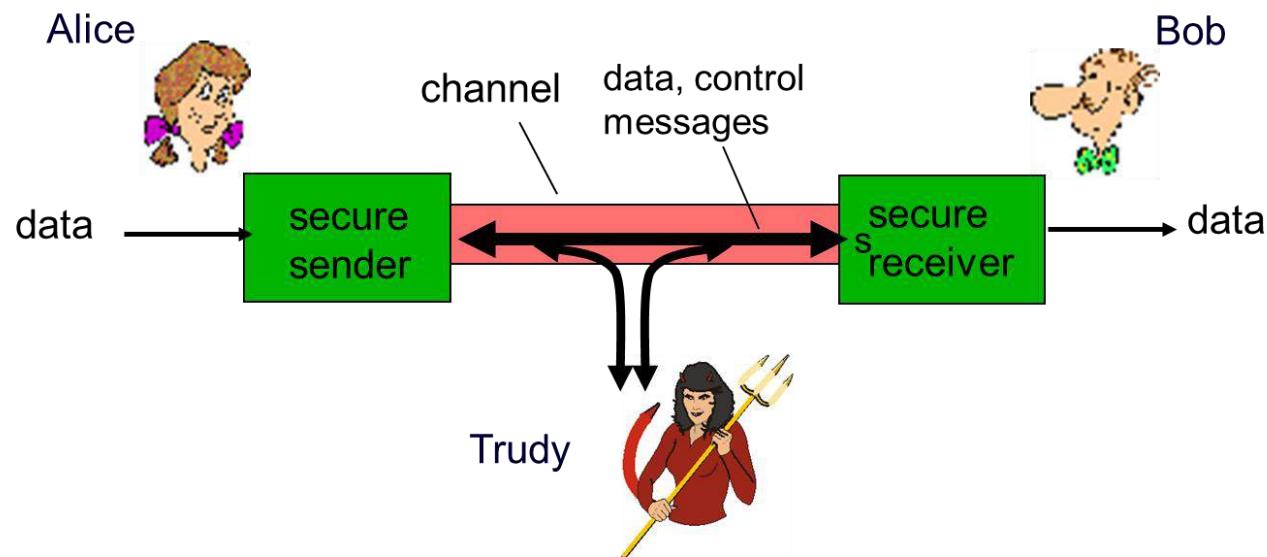
**message integrity:** sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

**access and availability:** services must be accessible and available to users



# Friends and Enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages



# What can a “bad person” do?

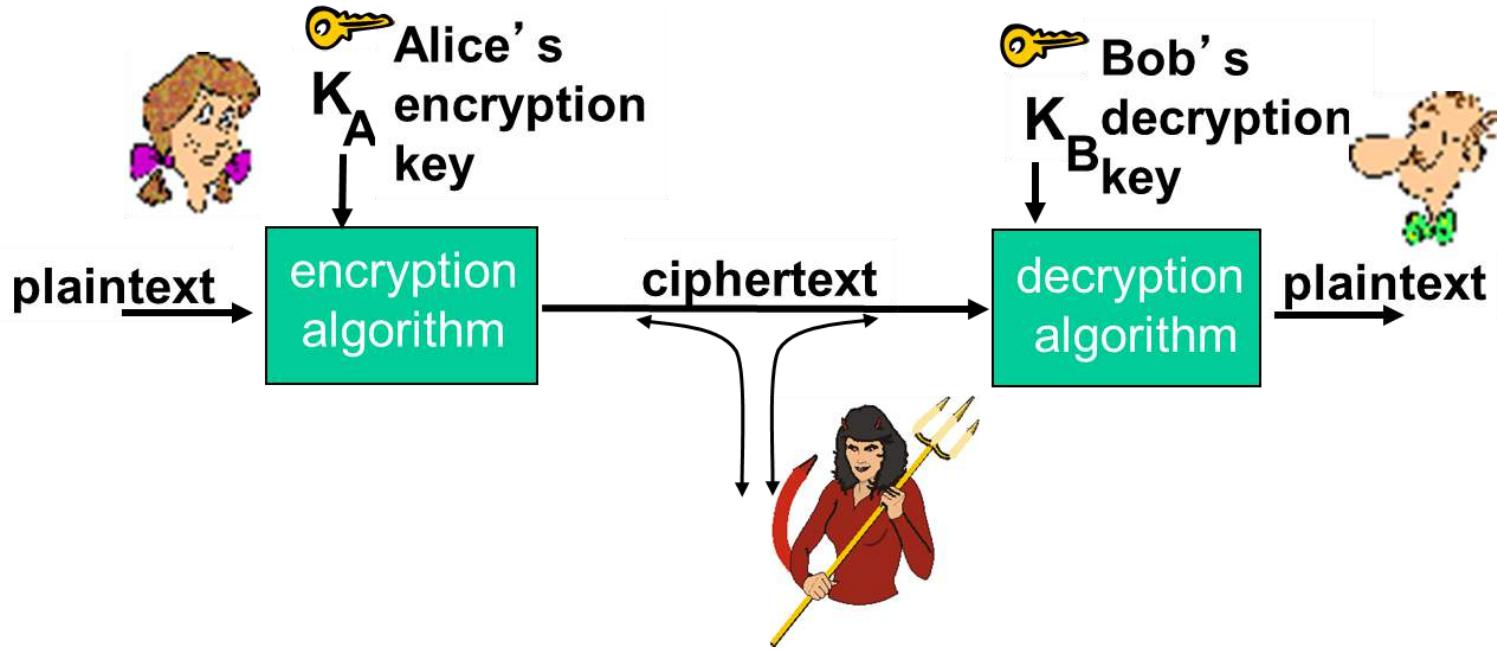
Broadly, threats can be categorized using the STRIDE mnemonic

- **Spoofing**—The attacker uses someone else’s information to access the system.
- **Tampering**—The attacker modifies some data in nonauthorized ways.
- **Repudiation**—The attacker removes all trace of their attack, so that they cannot be held accountable for other damages done.
- **Information disclosure**—The attacker accesses data they should not be able to.
- **Denial of service**—The attacker prevents real users from accessing the systems.
- **Elevation of privilege**—The attacker increases their privileges on the system thereby getting access to things they are not authorized to do.

# Agenda

- 8.1 What is network security?**
- 8.2 Principles of cryptography**
- 8.4 Authentication**
- 8.3 Message integrity**
- 8.6 Securing TCP connections: SSL**
- 8.7 Network layer security: IPsec**

# The Language of Cryptography



$m$  plaintext message

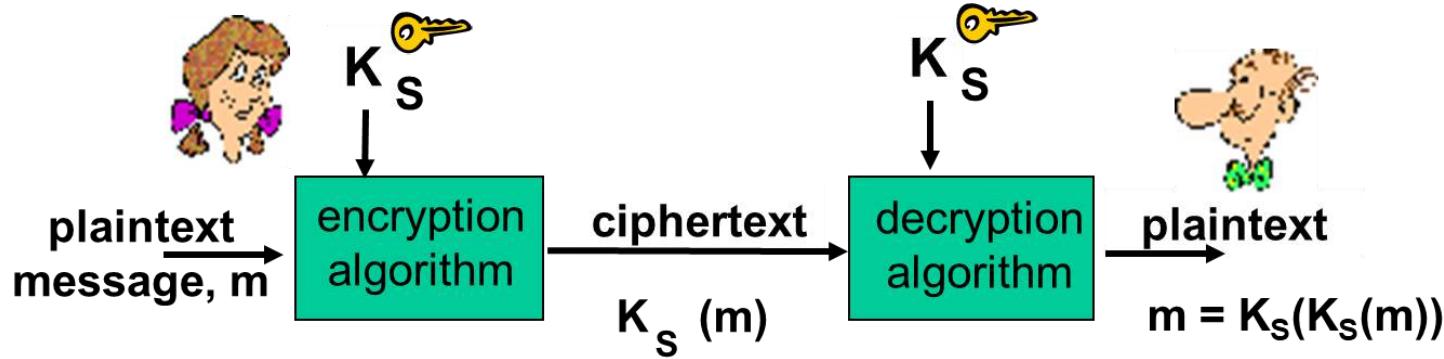
$K_A(m)$  ciphertext, encrypted with key  $K_A$

$$m = K_B(K_A(m))$$

# Breaking an Encryption Scheme

- **cipher-text only attack:**  
Trudy has ciphertext she can analyze
- **two approaches:**
  - brute force: search through all keys
  - statistical analysis
- **known-plaintext attack:**  
Trudy has plaintext corresponding to ciphertext
  - Trudy knows both  $m$  and  $K_A(m)$
- **chosen-plaintext attack:**  
Trudy can get ciphertext for chosen plaintext

# Symmetric Key Cryptography



**symmetric key crypto:** Bob and Alice share same (symmetric) key:  $K_s$

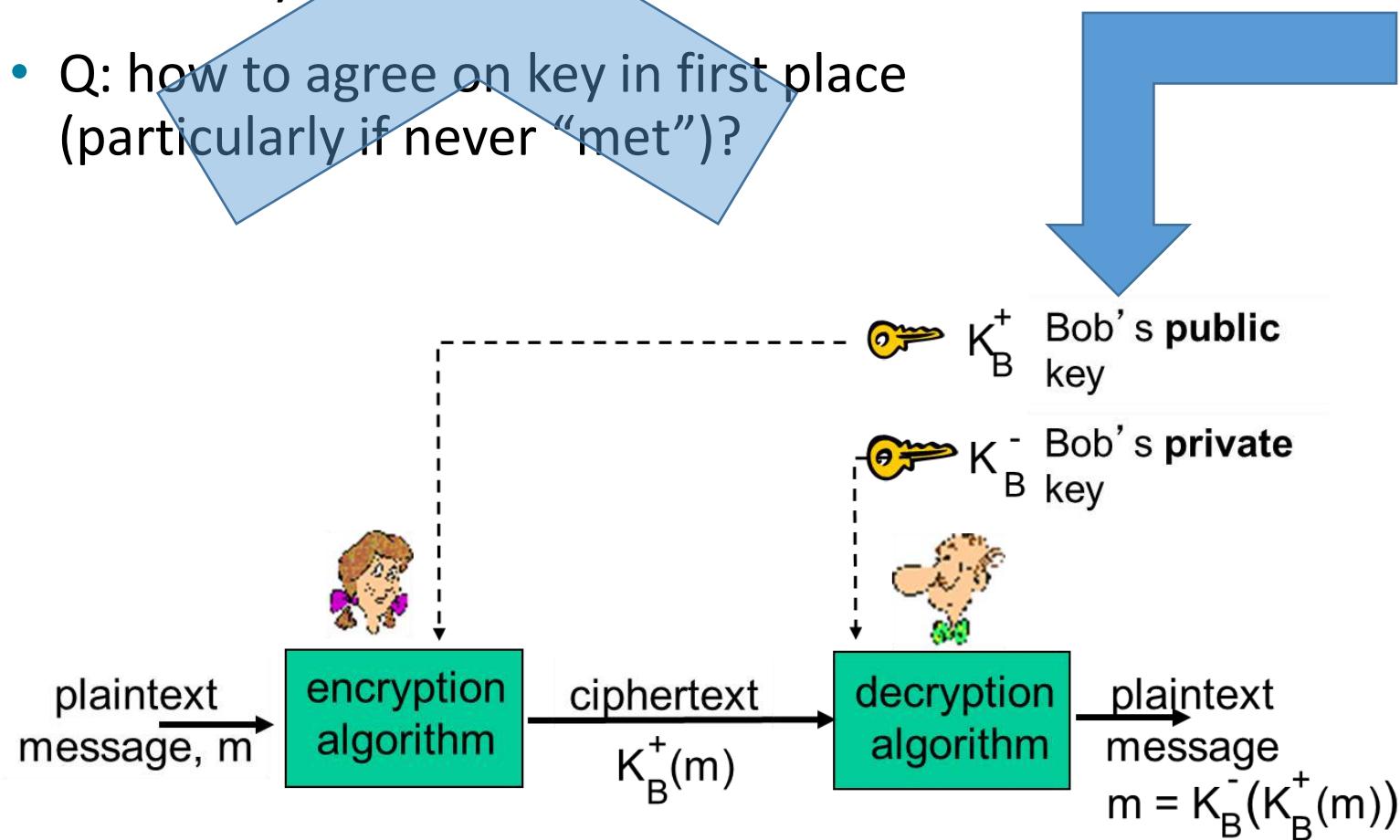
- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher, DES, AES

**Q:** how do Bob and Alice agree on key value?

# Public Key Cryptography

## ~~symmetric key crypto~~

- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never “met”)?



- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do **not** share secret key
- **public** encryption key known to all
- **private** decryption key known only to receiver

# Public Key Encryption Algorithms

requirements:

1. need  $k_B^+(.)$  and  $k_B^-(.)$  such that

$$k_B^-(k_B^+(m)) = m$$

2. given public key  $k_B^+$ , it should be impossible to compute private key  $k_B^-$

Important example:

- **RSA**: Rivest, Shamir, Adelson algorithm

# RSA: Another Important Property

The following property will be **very** useful later:

$$\underbrace{k_B^-(k_B^+(m))}_{\text{use public key first,}} = m = \underbrace{k_B^+(k_B^-(m))}_{\text{use private key first,}}$$

followed by private  
key

followed by public  
key

**result is the same!**

# RSA in Practice: Session Keys

- mathematical operations in RSA are computationally intensive
- DES is at least 100 times faster than RSA
- use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

**session key,  $K_s$**

- Bob and Alice use RSA to exchange a symmetric key  $K_s$
- once both have  $K_s$ , they use symmetric key cryptography

# Agenda

- 8.1 What is network security?**
- 8.2 Principles of cryptography**
- 8.4 Authentication**
- 8.3 Message integrity**
- 8.6 Securing TCP connections: SSL**
- 8.7 Network layer security: IPsec**

# Authentication (1 of 2)

**Goal:** Bob wants Alice to “prove” her identity to him

**Protocol ap1.0:** Alice says “I am Alice”



“I am Alice”

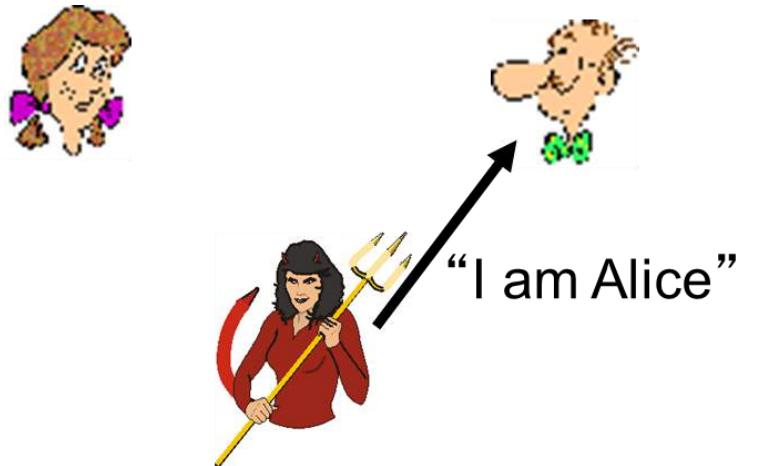
Failure scenario??



# Authentication (2 of 2)

**Goal:** Bob wants Alice to “prove” her identity to him

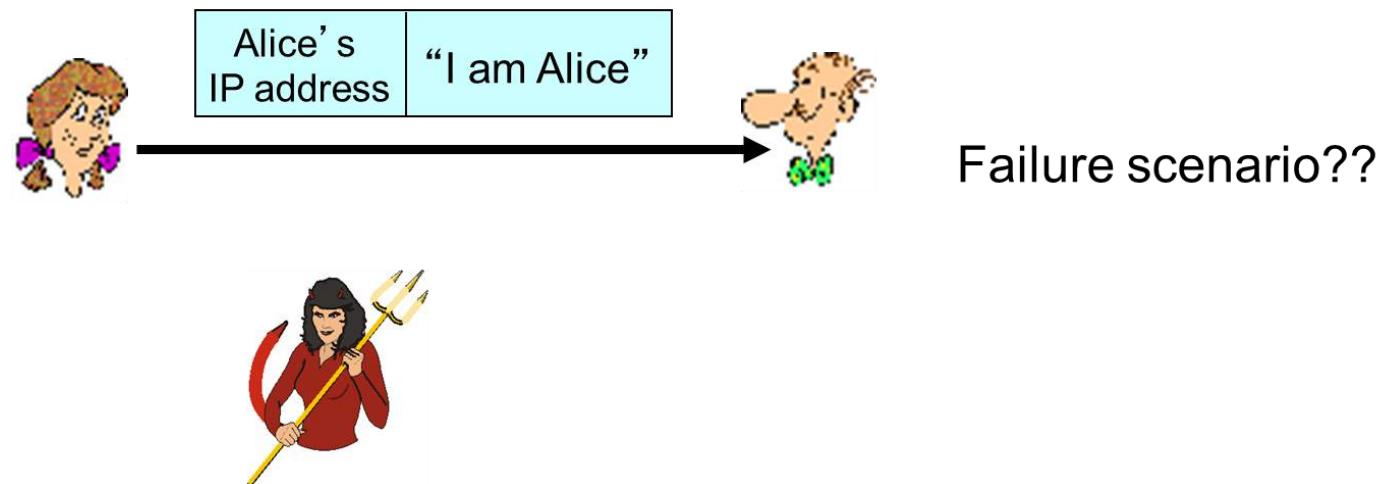
**Protocol ap1.0:** Alice says “I am Alice”



in a network,  
Bob can not “see” Alice,  
so Trudy simply declares  
herself to be Alice

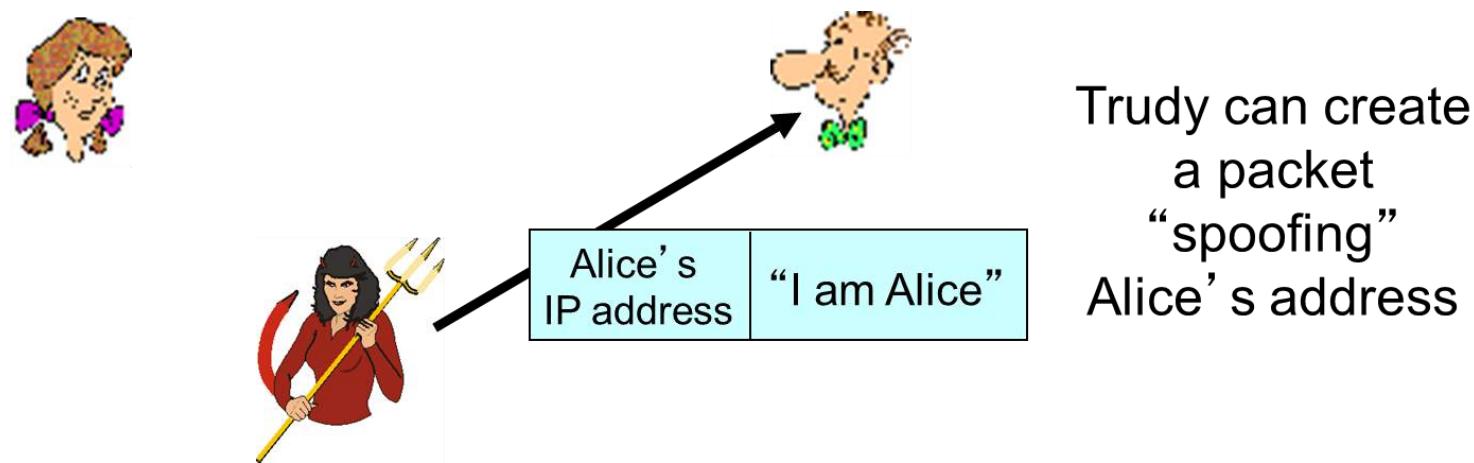
# Authentication: using IP address (1 of 2)

**Protocol ap2.0:** Alice says “I am Alice” in an IP packet containing her source IP address



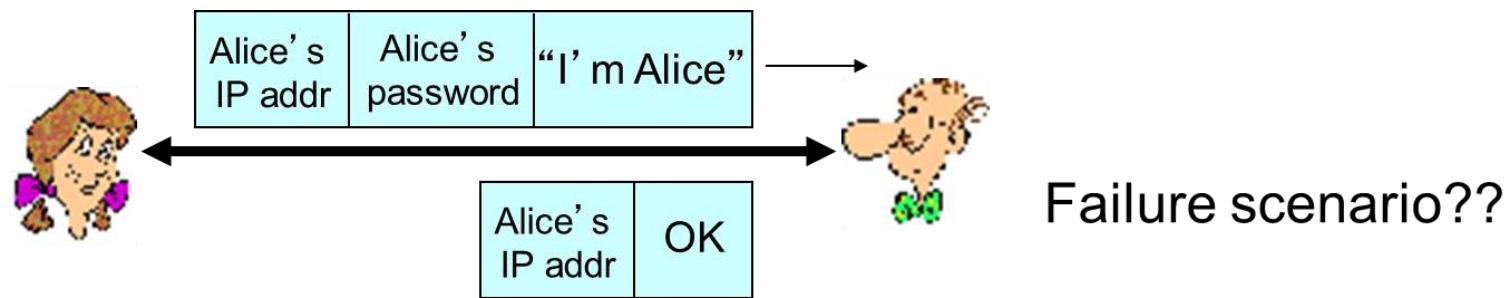
# Authentication: using IP address (2 of 2)

**Protocol ap2.0:** Alice says “I am Alice” in an IP packet containing her source IP address



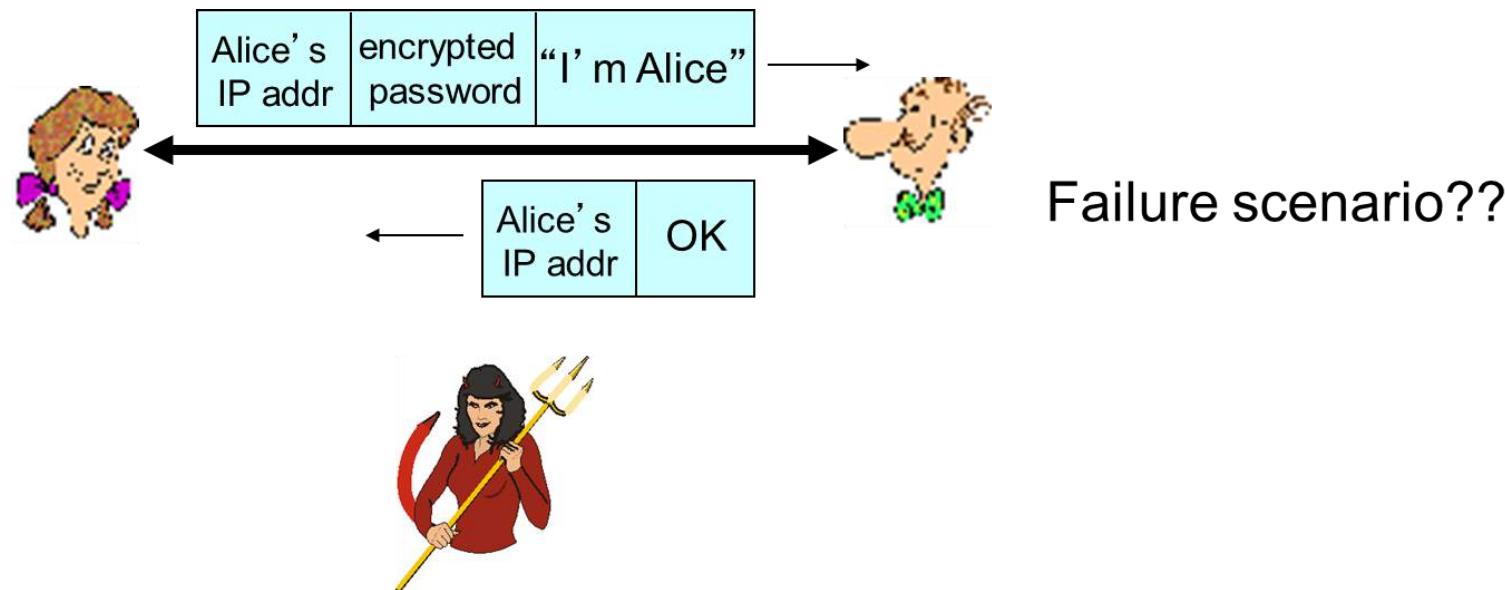
# Authentication: use a key

**Protocol ap3.0:** Alice says “I am Alice” and sends her secret password to “prove” it.



# Authentication: encrypt the password (1 of 2)

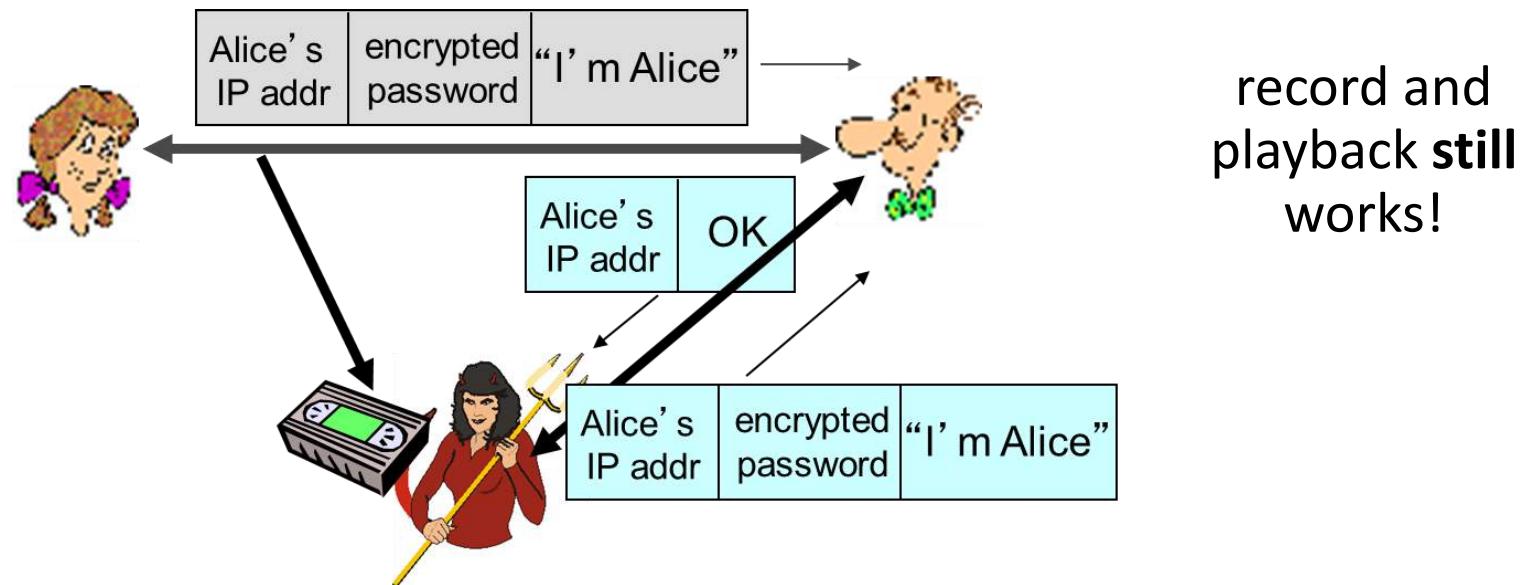
**Protocol ap3.1:** Alice says “I am Alice” and sends her **encrypted** secret password to “prove” it.



# Authentication: encrypt the password (2 of 2)

**Protocol ap3.1:** Alice says “I am Alice” and sends her **encrypted** secret password to “prove” it.

The attacker doesn’t learn Alice’s password, but it does not need it.

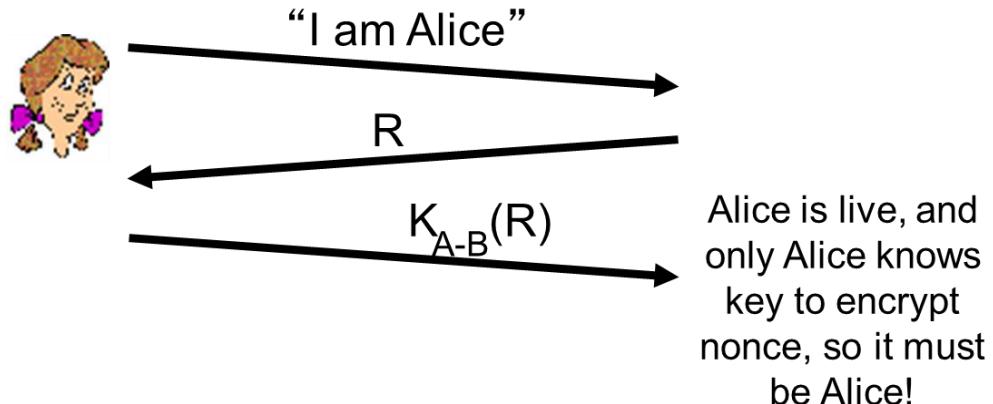


# Authentication: use a nonce

**Goal:** avoid playback attack

**nonce:** number ( $R$ ) used only **once-in-a-lifetime**

**ap4.0:** to prove Alice “live”, Bob sends Alice **nonce**,  $R$ . Alice must return  $R$ , encrypted with shared secret key



Failures, drawbacks?

Hint for failure: Trudy waits that Bob tries to authenticate with “Alice”

Hint for drawbacks: key management and dissemination

# Authentication: nonce + public key

ap4.0 required a shared symmetric key

- can we authenticate using public key techniques?

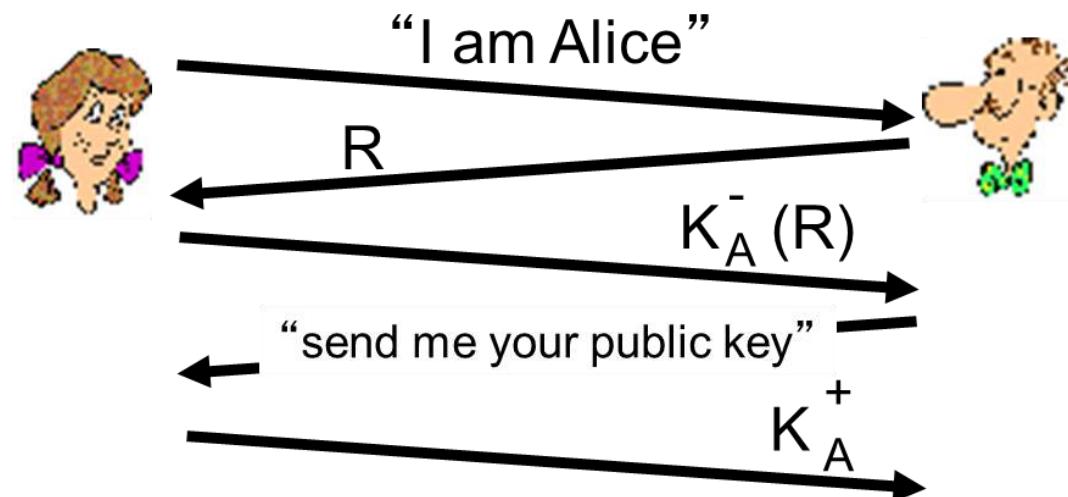
**ap5.0:** use nonce, public key cryptography

Bob computes

$$k_A^+ (k_A^- (R)) = R$$

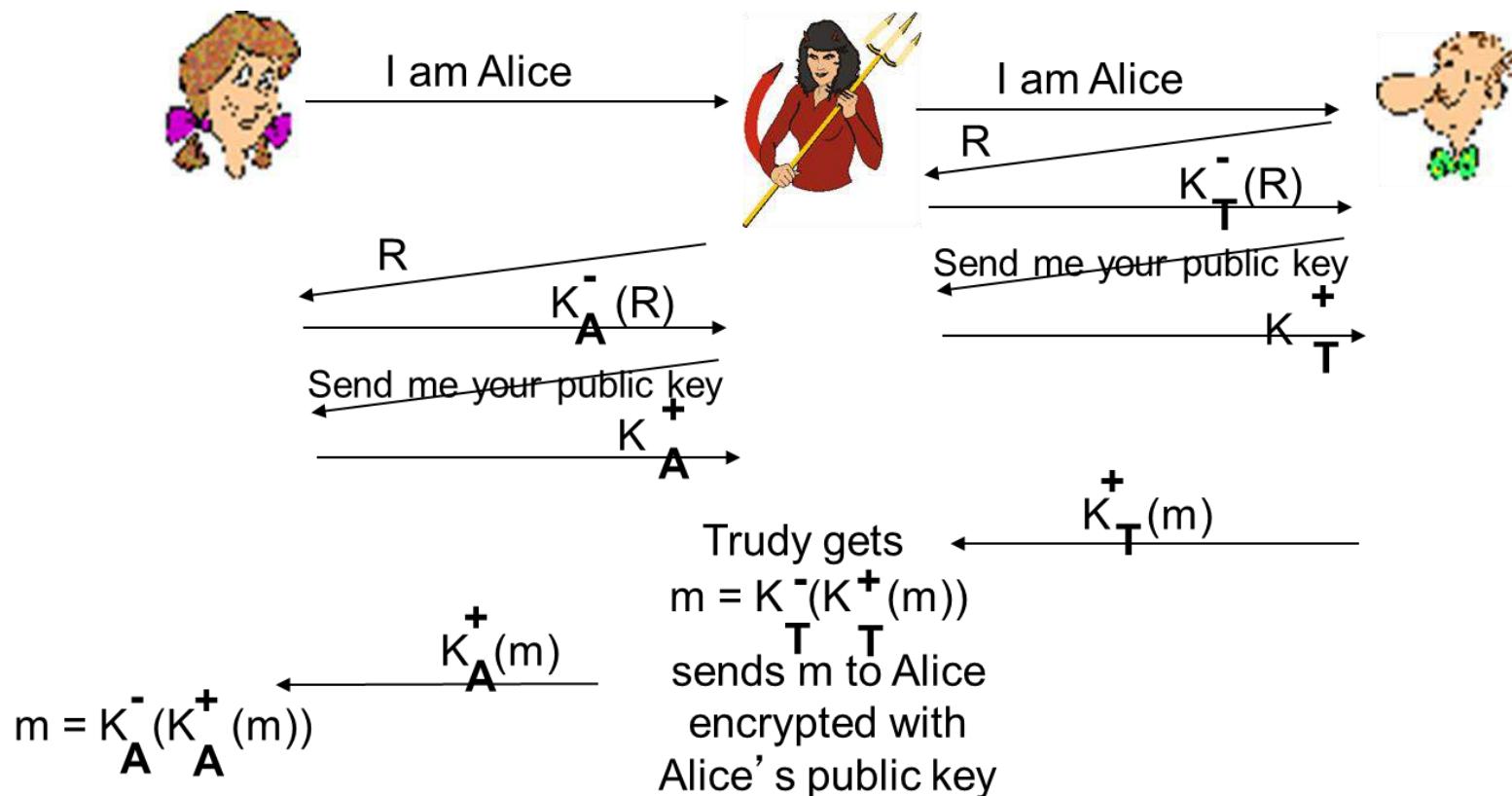
and knows only Alice could have the private key, that encrypted R such that

$$k_A^+ (k_A^- (R)) = R$$



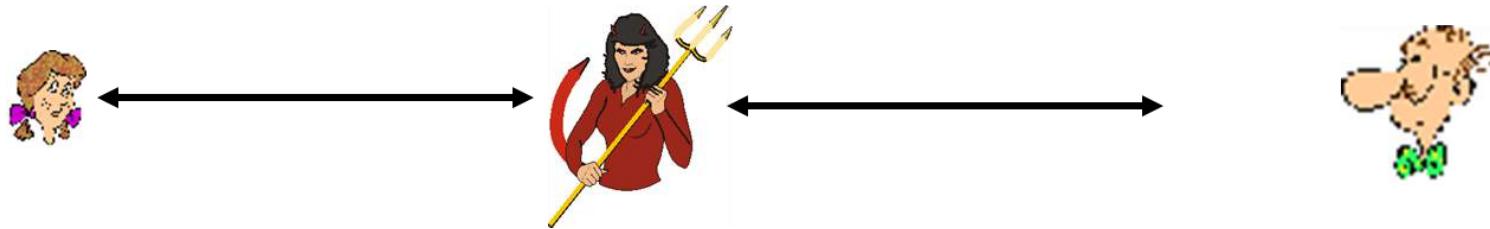
# ap5.0: Security Hole (1 of 2)

**man (or woman) in the middle attack:** Trudy poses as Alice (to Bob) and as Bob (to Alice)



# ap5.0: Security Hole (2 of 2)

**man (or woman) in the middle attack:** Trudy poses as Alice (to Bob) and as Bob (to Alice)



difficult to detect:

- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation!)
- problem is that Trudy receives all messages as well!

# Agenda

- 8.1 What is network security?**
- 8.2 Principles of cryptography**
- 8.4 Authentication**
- 8.3 Message integrity**
- 8.6 Securing TCP connections: SSL**
- 8.7 Network layer security: IPsec**

# Digital Signatures (1 of 3)

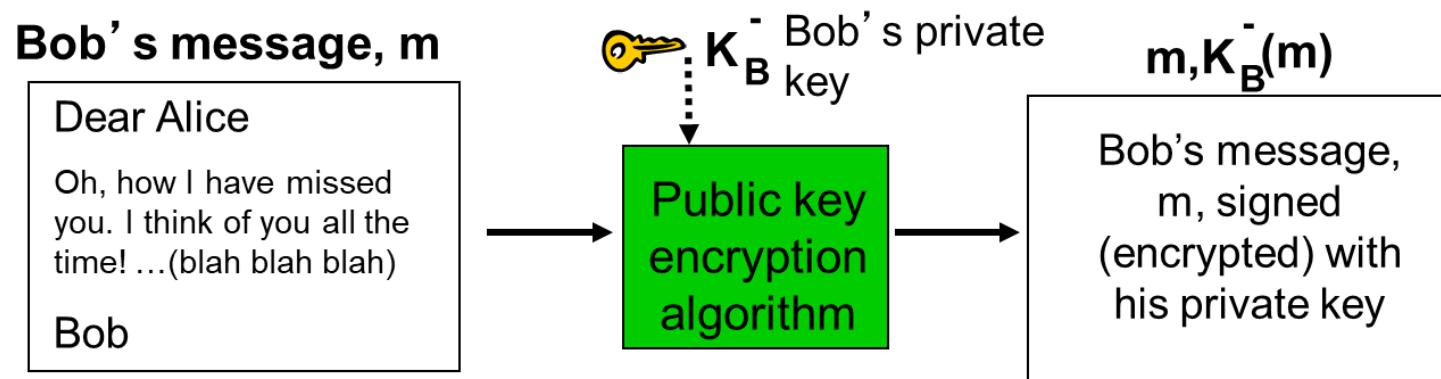
**cryptographic technique analogous to hand-written signatures:**

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- **verifiable, nonforgeable:** recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

# Digital Signatures (2 of 3)

**simple digital signature for message  $m$ :**

- Bob signs  $m$  by encrypting with his private key  $k_B^{-1}(m)$ , creating “signed” message,  $k_B^{-1}(m)$



# Digital Signatures (3 of 3)

- suppose Alice receives msg  $m$ , with signature:  $m, K_B^-(m)$
- Alice verifies  $m$  signed by Bob by applying Bob's public key  $k_B$  to  $K_B^-(m)$  then checks  $K_B(K_B^-(m)) = m$ .
- If  $K_B^+(K_B^-(m)) = m$ , whoever signed  $m$  must have used Bob's private key.

**Alice thus verifies that:**

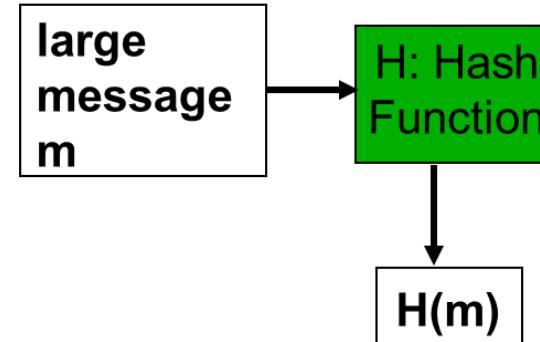
- It was Bob who signed  $m$
- Bob signed  $m$  and not  $m'$ 
  - By the way, Bob cannot repudiate  $m$

# Message Digests

computationally expensive to public-key-encrypt long messages

**goal:** fixed-length, easy- to- compute digital “fingerprint”

- apply hash function  $H$  to  $m$ , get fixed size message digest,  $H(m)$
- then, encrypt (sign) the hash  $H(m)$

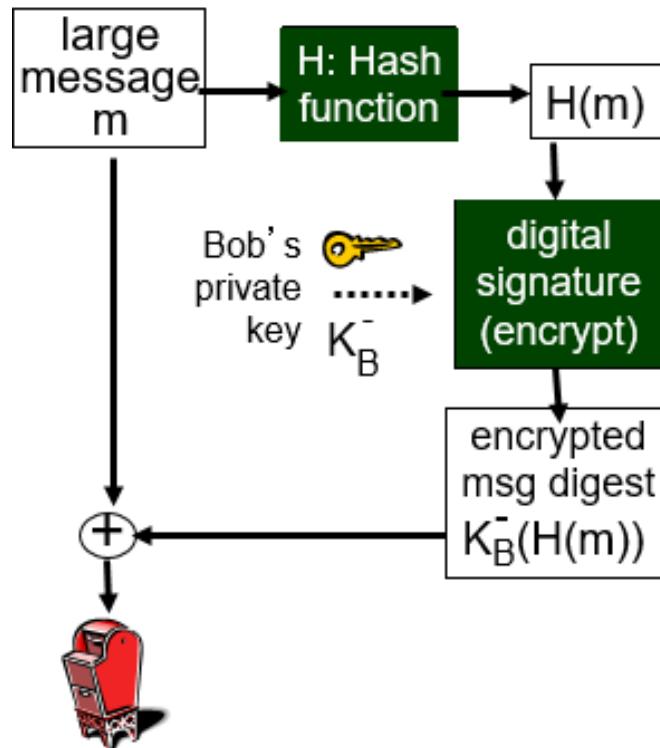


**Hash function properties:**

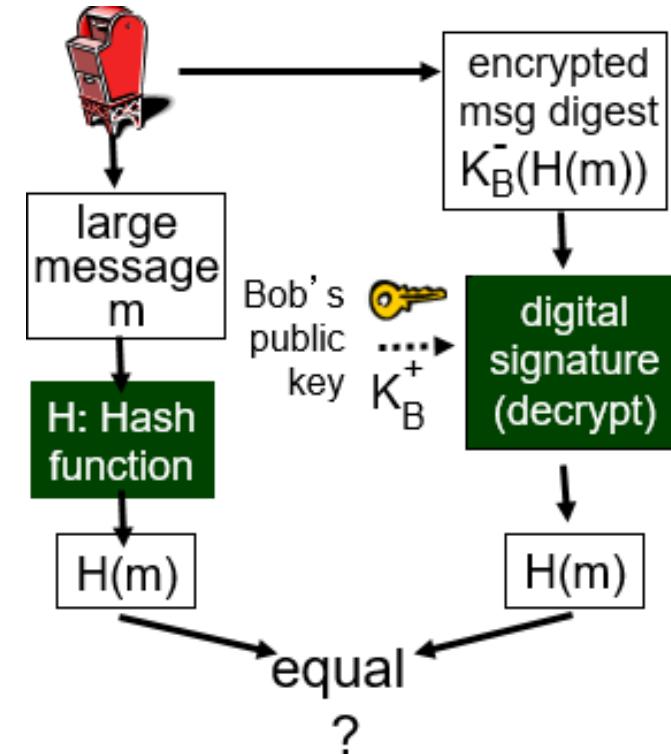
- many-to-1
- produces fixed-size msg digest (fingerprint)
- given message digest  $x$ , computationally infeasible to find  $m$  such that  $x = H(m)$

# Digital Signature = Signed Message Digest

Bob sends digitally signed message:



Alice verifies signature, **integrity** of digitally signed message:

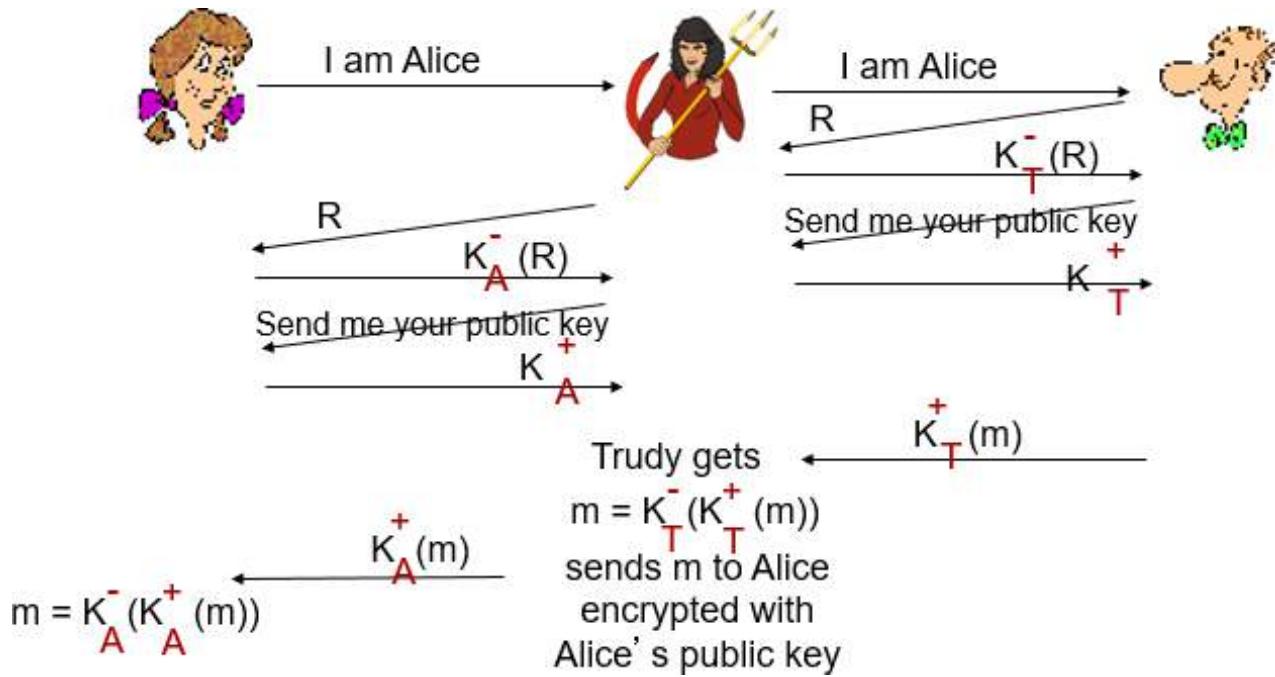


# Hash Function Algorithms

- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string  $x$ , appears difficult to construct msg  $m$  whose MD5 hash is equal to  $x$
- SHA-1 is also used
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

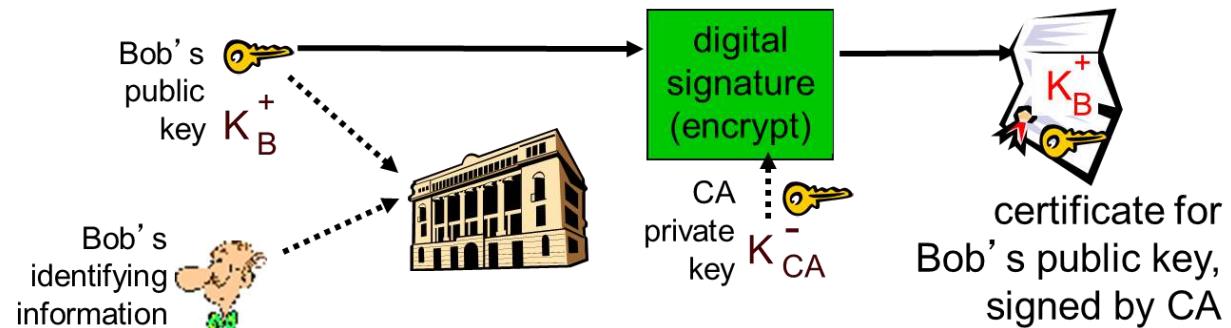
# Recall: ap5.0 Security Hole

**man (or woman) in the middle attack:** Trudy poses as Alice (to Bob) and as Bob (to Alice)



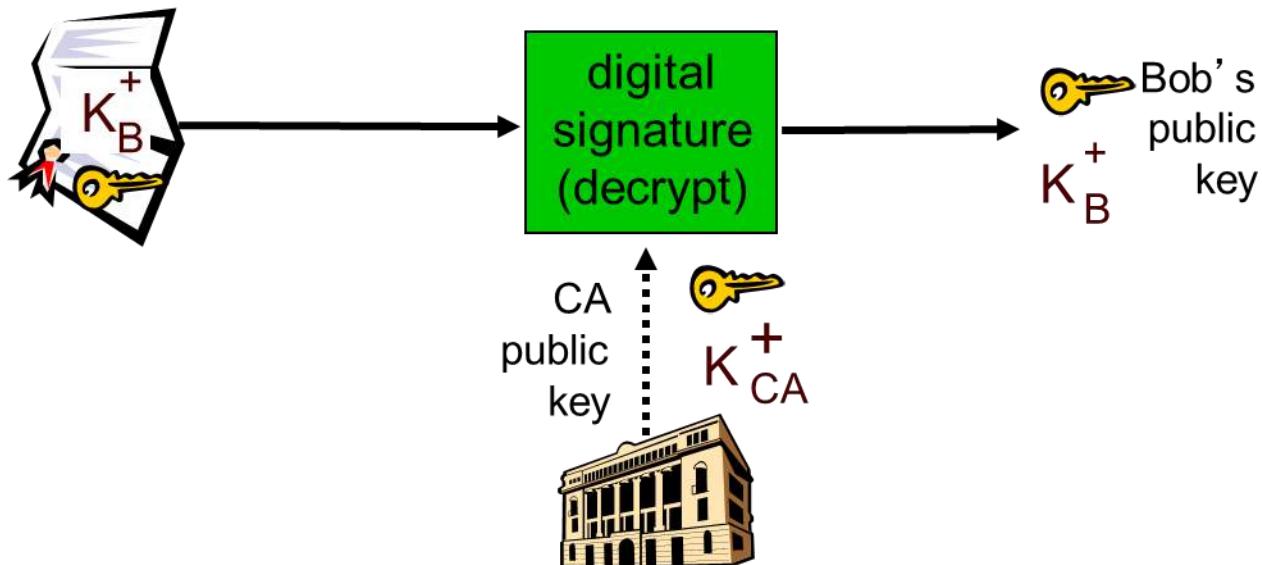
# Certification Authorities (1 of 2)

- **certification authority (CA):** binds public key to particular entity, E.
- E(person, router) registers its public key with CA.
  - E provides “proof of identity” to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E’s public key digitally signed by CA – CA says “this is E’s public key”



# Certification Authorities (2 of 2)

- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key
- Requirements for it to work:
  - Everybody must know  $K_{CA}^+$  (e.g.:pre-shipping CA's certificates with the browser)
  - Nobody can know  $K_{CA}^-$  (except the CA)



# Agenda

8.1 What is network security?

8.2 Principles of cryptography

8.4 Authentication

8.3 Message integrity

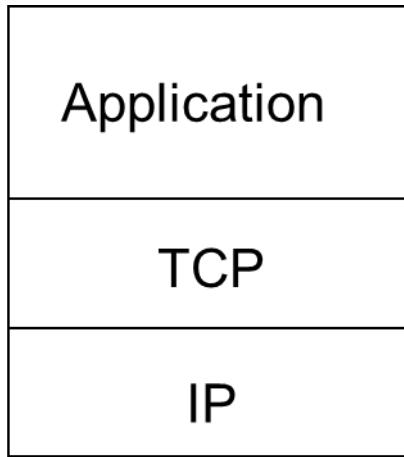
**8.6 Securing TCP connections: SSL**

8.7 Network layer security: IPsec

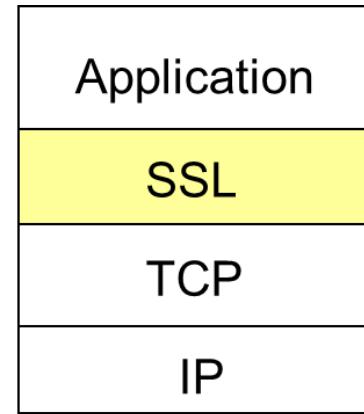
# SSL: Secure Sockets Layer

- widely deployed security protocol
  - supported by almost all browsers, web servers
  - https
  - billions \$/year over SSL
- mechanisms: [Woo 1994], implementation: Netscape
- variation TLS: transport layer security, RFC 2246
- provides
  - **confidentiality**
  - **integrity**
  - **authentication**
- original goals:
  - Web e-commerce transactions
  - encryption (especially credit-card numbers)
  - Web-server authentication
  - optional client authentication
  - minimum hassle in doing business with new merchant
- available to all TCP applications
  - secure socket interface

# SSL and TCP/IP



**normal application**



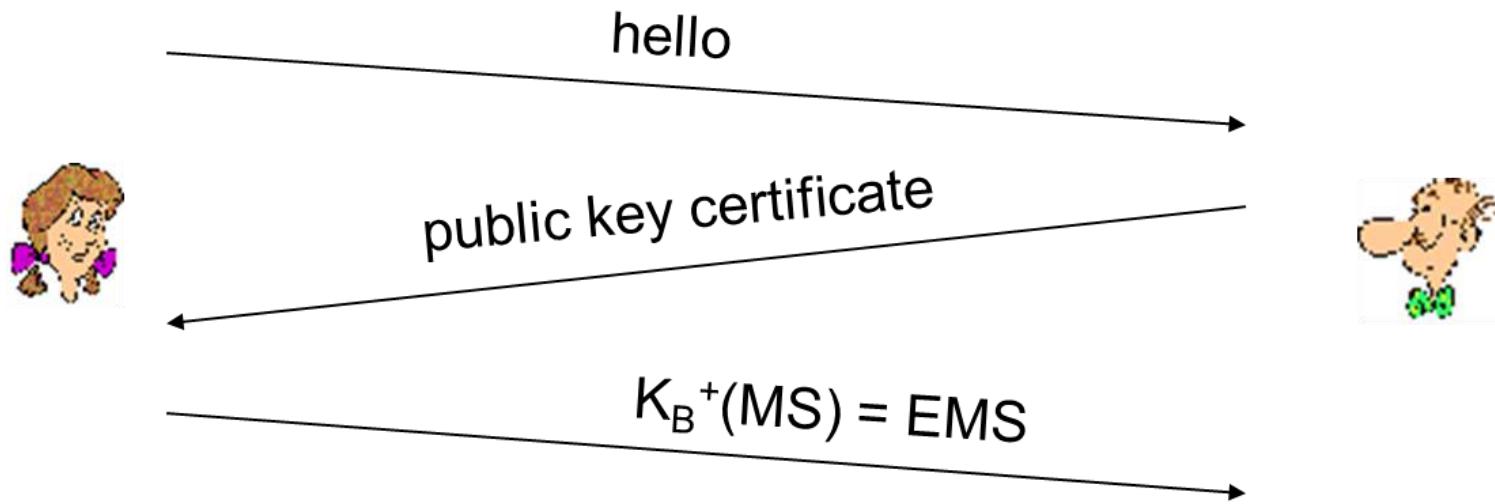
**application with  
SSL**

- SSL provides application programming interface (API) to applications
- C and Java SSL libraries/classes readily available

# Toy SSL: A Simple Secure Channel

- **handshake:** Alice and Bob use their certificates, private keys to authenticate each other and exchange shared secret
- **key derivation:** Alice and Bob use shared secret to derive set of keys
- **data transfer:** data to be transferred is broken up into series of records
- **connection closure:** special messages to securely close connection

# Toy: A Simple Handshake



**MS:** master secret

**EMS:** encrypted master secret

# Toy: Key Derivation

- considered bad to use same key for more than one cryptographic operation
  - use different keys for message authentication code (MAC) and encryption
- four keys:
  - $K_c$  = encryption key for data sent from client to server
  - $M_c$  = MAC key for data sent from client to server
  - $K_s$  = encryption key for data sent from server to client
  - $M_s$  = MAC key for data sent from server to client
- keys derived from key derivation function (KDF)
  - takes master secret and (possibly) some additional random data and creates the keys

# Toy: Data Records

- why not encrypt data in constant stream as we write it to TCP?
  - where would we put the MAC? If at end, no message integrity until all data processed.
  - e.g., with instant messaging, how can we do integrity check over all bytes sent before displaying?
- instead, break stream in series of records
  - each record carries a MAC
  - receiver can act on each record as it arrives
- issue: in record, receiver needs to distinguish MAC from data
  - want to use variable-length records



# Toy: Sequence Numbers

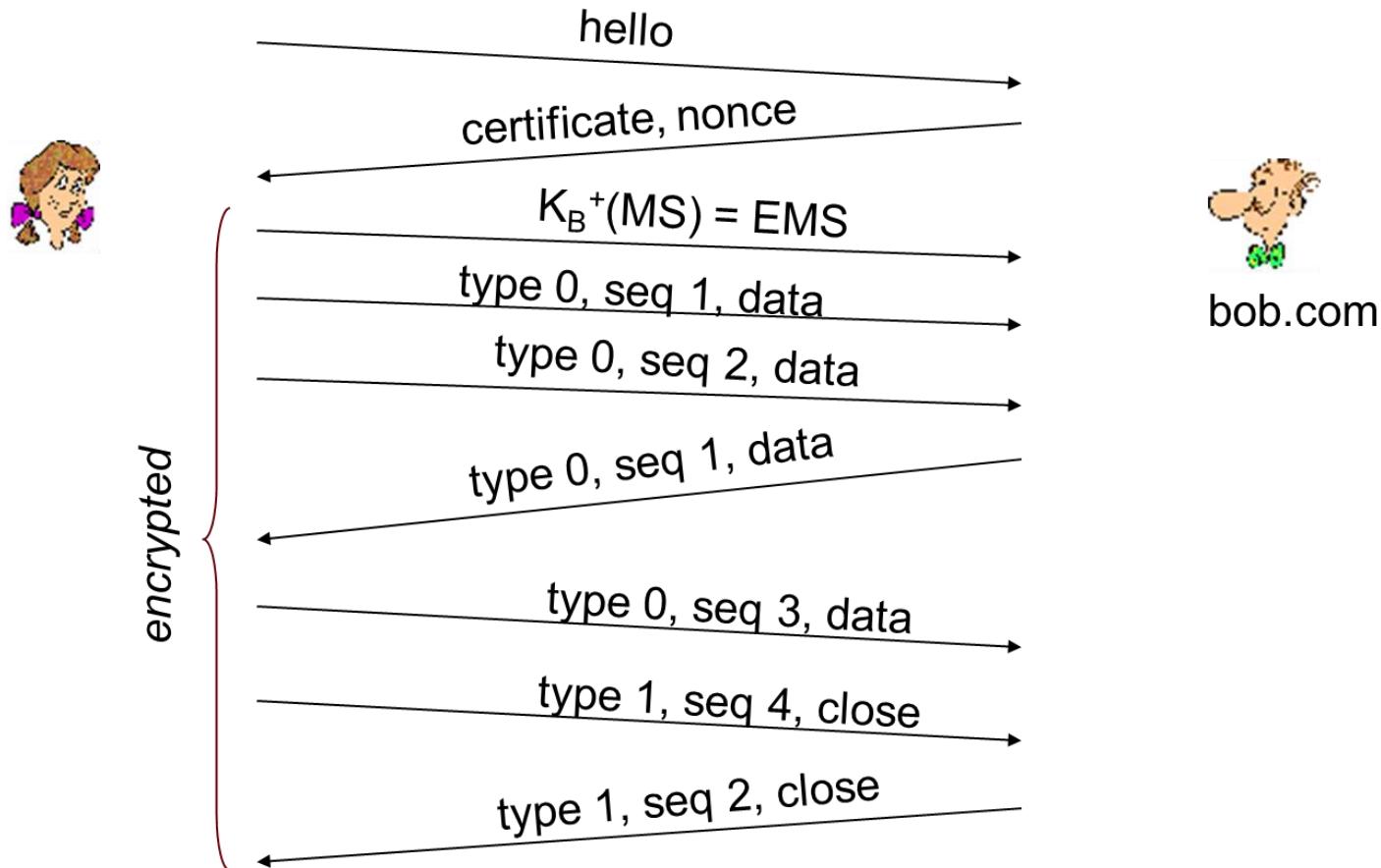
- **problem:** attacker can capture and replay record or re-order records
- **solution:** put sequence number into MAC:
  - $\text{MAC} = \text{MAC}(M_x, \text{sequence } || \text{data})$
  - note: no sequence number field
- **problem:** attacker could replay all records
- **solution:** use nonce

# Toy: Control Information

- **problem:** truncation attack:
  - attacker forges TCP connection close segment
  - one or both sides thinks there is less data than there actually is.
- **solution:** record types, with one type for closure
  - type 0 for data; type 1 for closure

$$\text{MAC} = \text{MAC}(M_x, \text{sequence} \parallel \text{type} \parallel \text{data})$$

# Toy SSL: Summary



# Toy SSL is not complete

- how long are fields?
- which encryption protocols?
- want negotiation?
  - allow client and server to support different encryption algorithms
  - allow client and server to choose together specific algorithm before data transfer

# SSL Cipher Suite

- cipher suite
  - public-key algorithm
  - symmetric encryption algorithm
  - MAC algorithm
- SSL supports several cipher suites
- negotiation: client, server agree on cipher suite
  - client offers choice
  - server picks one
- common SSL symmetric ciphers
  - DES – Data Encryption Standard: block
  - 3DES – Triple strength: block
  - RC2 – Rivest Cipher 2: block
  - RC4 – Rivest Cipher 4: stream
- SSL Public key encryption
  - RSA

# Real SSL: Handshake (1 of 4)

## Purpose

1. server authentication
2. negotiation: agree on crypto algorithms
3. establish keys
4. client authentication (optional)

# Real SSL: Handshake (2 of 4)

1. client sends list of algorithms it supports, along with client nonce
2. server chooses algorithms from list; sends back: choice + certificate + server nonce
3. client verifies certificate, extracts server's public key, generates pre\_master\_secret, encrypts with server's public key, sends to server
4. client and server independently compute encryption and MAC keys from pre\_master\_secret and nonces
5. client sends a MAC of all the handshake messages
6. server sends a MAC of all the handshake messages

# Real SSL: Handshake (3 of 4)

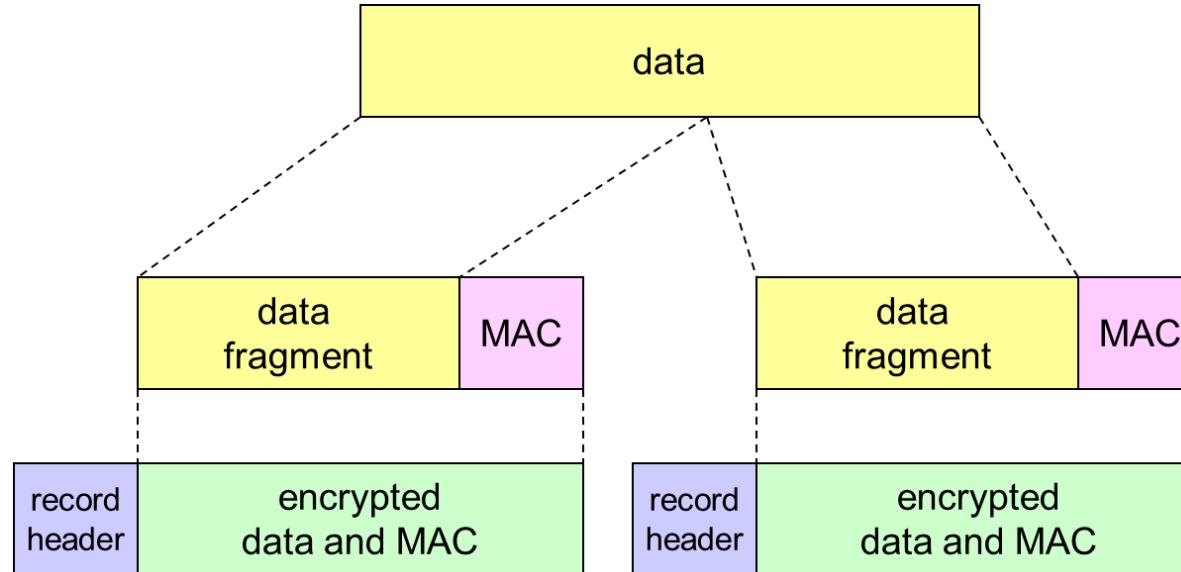
## last 2 steps protect handshake from tampering

- client typically offers range of algorithms, some strong, some weak
- man-in-the middle could delete stronger algorithms from list
- last 2 steps prevent this
  - last two messages are encrypted

# Real SSL: Handshake (4 of 4)

- why two random nonces?
- suppose Trudy sniffs all messages between Alice & Bob
- next day, Trudy sets up TCP connection with Bob, sends exact same sequence of records
  - Bob (Amazon) thinks Alice made two separate orders for the same thing
  - solution: Bob sends different random nonce for each connection. This causes encryption keys to be different on the two days
  - Trudy's messages will fail Bob's integrity check

# SSL Record Protocol

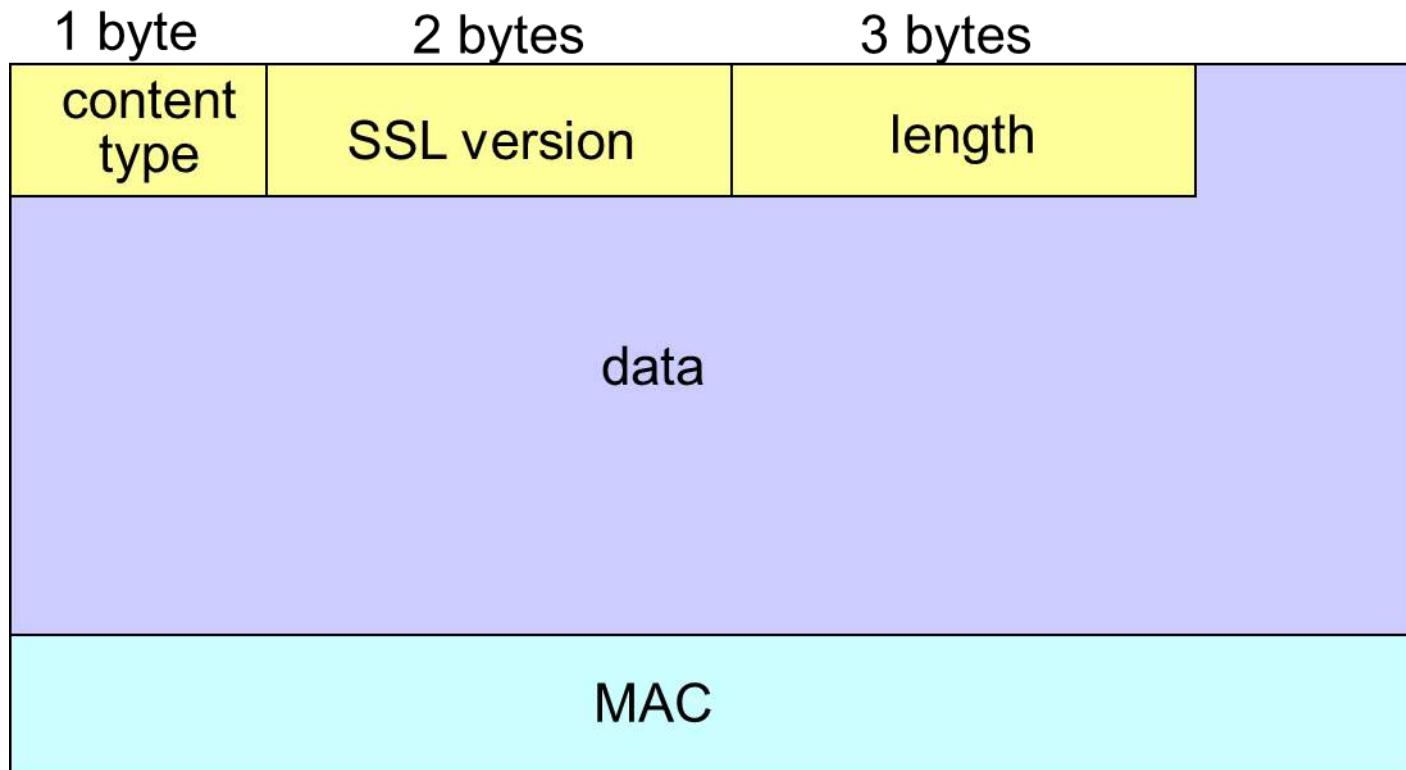


**record header:** content type; version; length

**MAC:** includes sequence number, MAC key  $M_x$

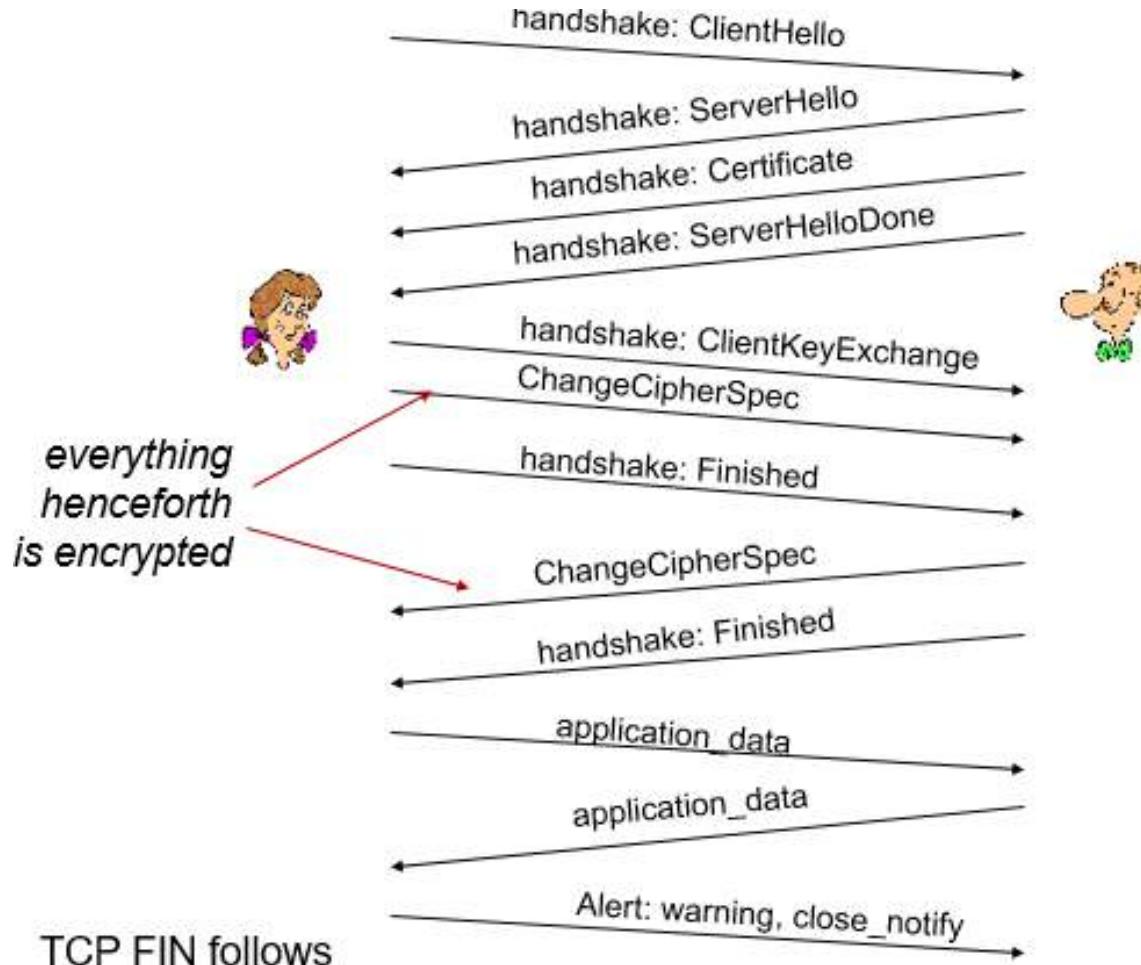
**fragment:** each SSL fragment:  $2^{14}$  bytes (~16 Kbytes)

# SSL Record Format



data and MAC encrypted (symmetric algorithm)

# Real SSL Connection



# Agenda

- 8.1 What is network security?**
- 8.2 Principles of cryptography**
- 8.4 Authentication**
- 8.3 Message integrity**
- 8.6 Securing TCP connections: SSL**
- 8.7 Network layer security: IPsec**

# What is Network-Layer Confidentiality ?

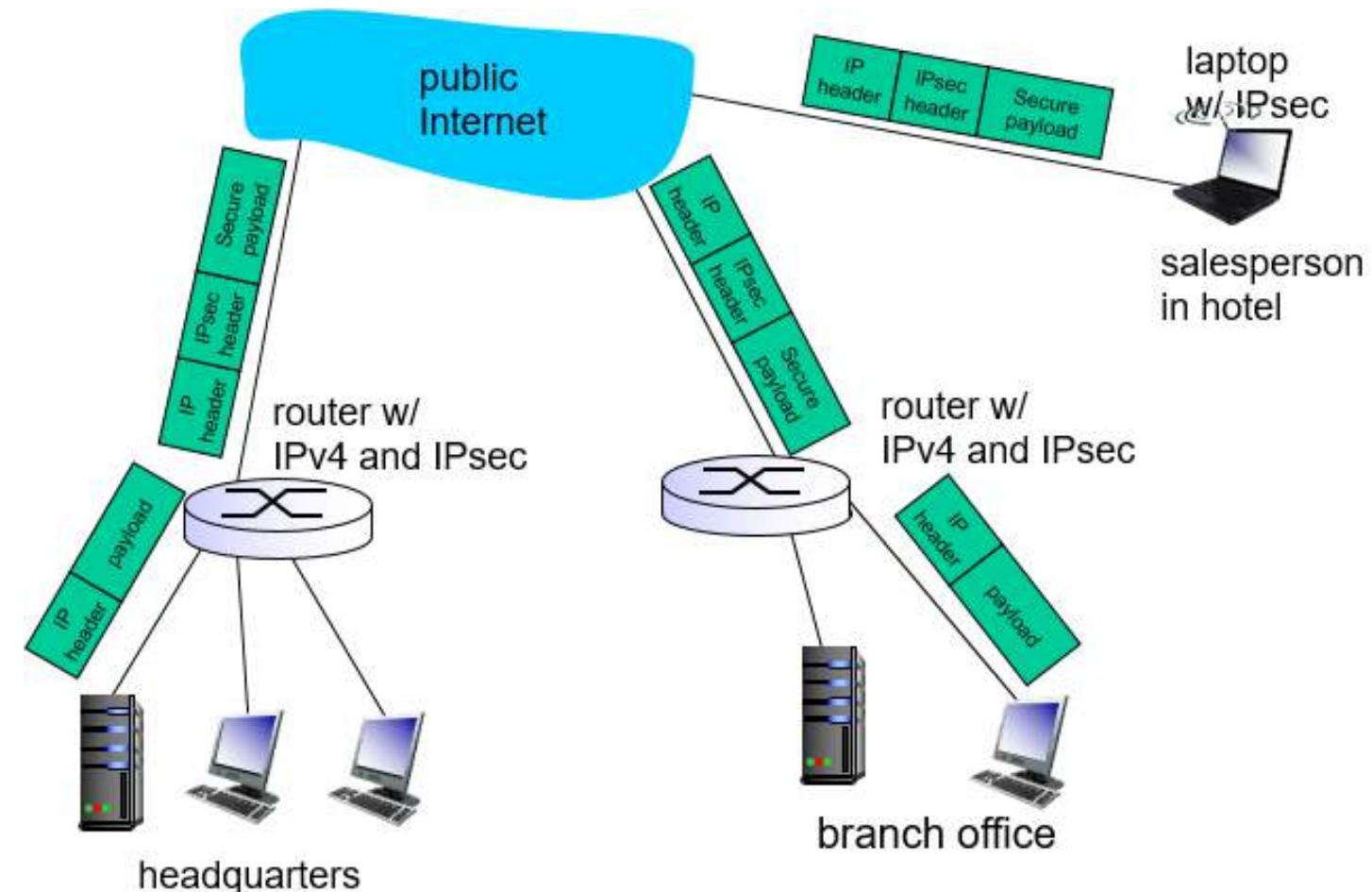
**between two network entities:**

- sending entity encrypts datagram payload, payload could be:
  - TCP or UDP segment, ICMP message, OSPF message ....
- all data sent from one entity to other would be hidden:
  - web pages, e-mail, P2P file transfers, TCP SYN packets ...
- “blanket coverage”

# Virtual Private Networks (VPNs)

## motivation:

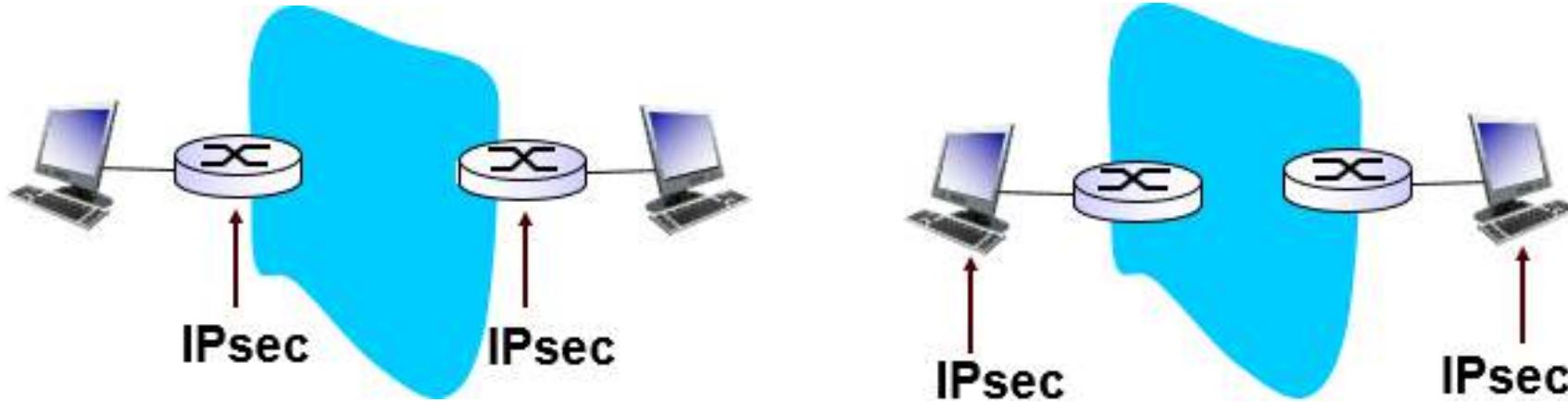
- institutions often want private networks for security
  - costly: separate routers, links, DNS infrastructure.
- VPN: institution's inter-office traffic is sent over public Internet
  - encrypted before entering public Internet
  - logically separate from other traffic



# IPsec Services

- data integrity
- origin authentication
- replay attack prevention
- confidentiality
- two protocols providing different service models:
  - AH
  - ESP

# IPsec – Transport Mode



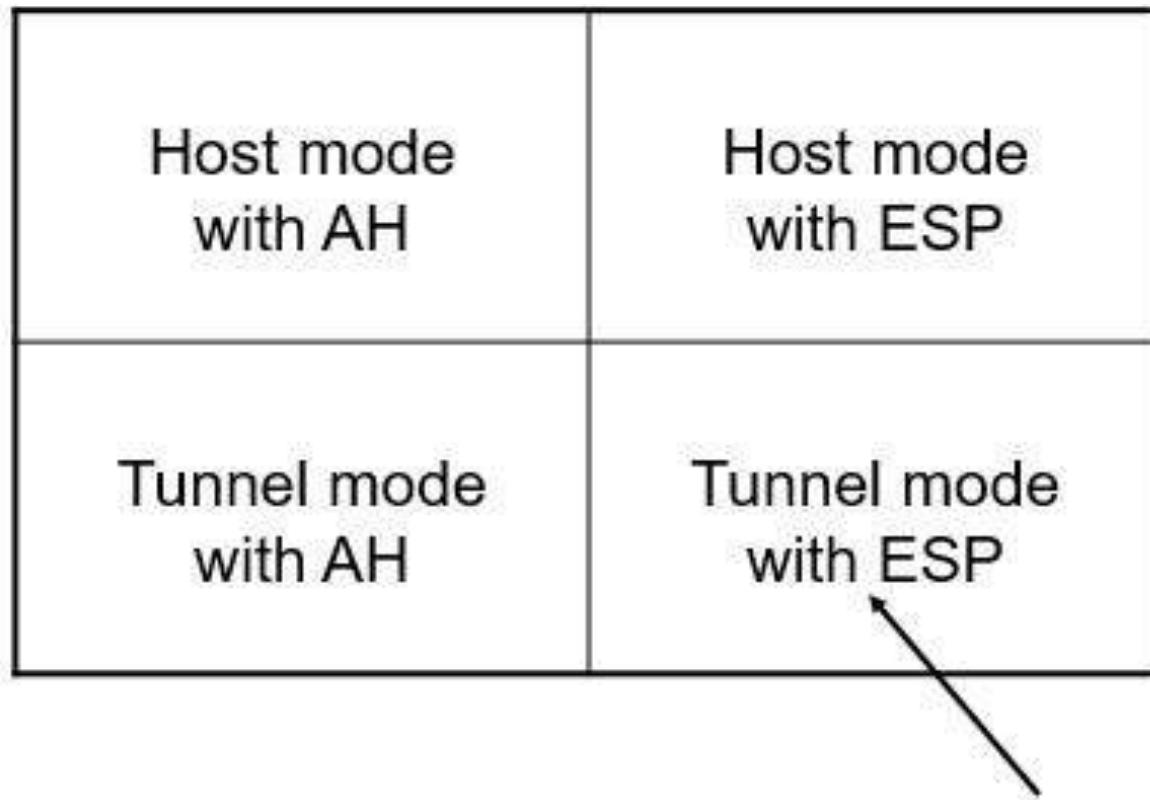
- Tunnel mode:  
edge routers IPsec-aware

- Host mode:  
hosts IPsec-aware

# Two IPsec Protocols

- Authentication Header (AH) protocol
  - provides source authentication & data integrity but **not** confidentiality
- Encapsulation Security Protocol (ESP)
  - provides source authentication, data integrity, **and** **confidentiality**
  - more widely used than AH

# Four Combinations are Possible!

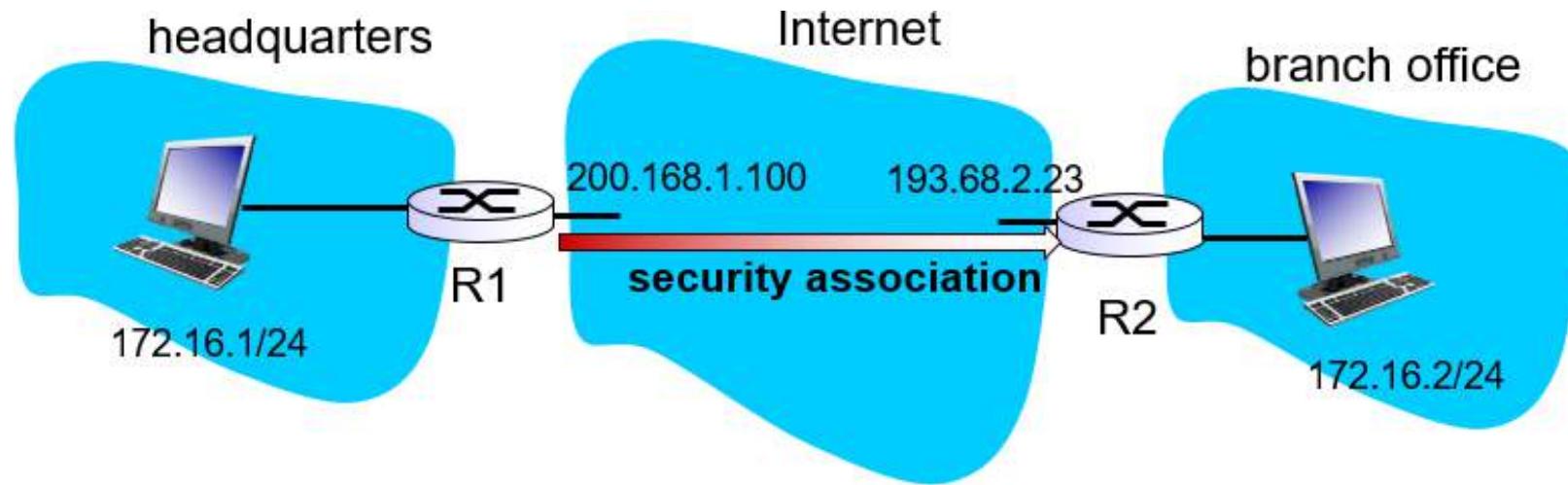


**most common and most  
important**

# Security Associations (SAs)

- before sending data, “**security association (SA)**” established from sending to receiving entity
  - SAs are simplex: for only one direction
- receiving entities maintain **state information** about SA
  - recall: TCP endpoints also maintain state info
  - IP is connectionless; IPsec is connection-oriented!
- how many SAs in VPN w/ headquarters, branch office, and n traveling salespeople?

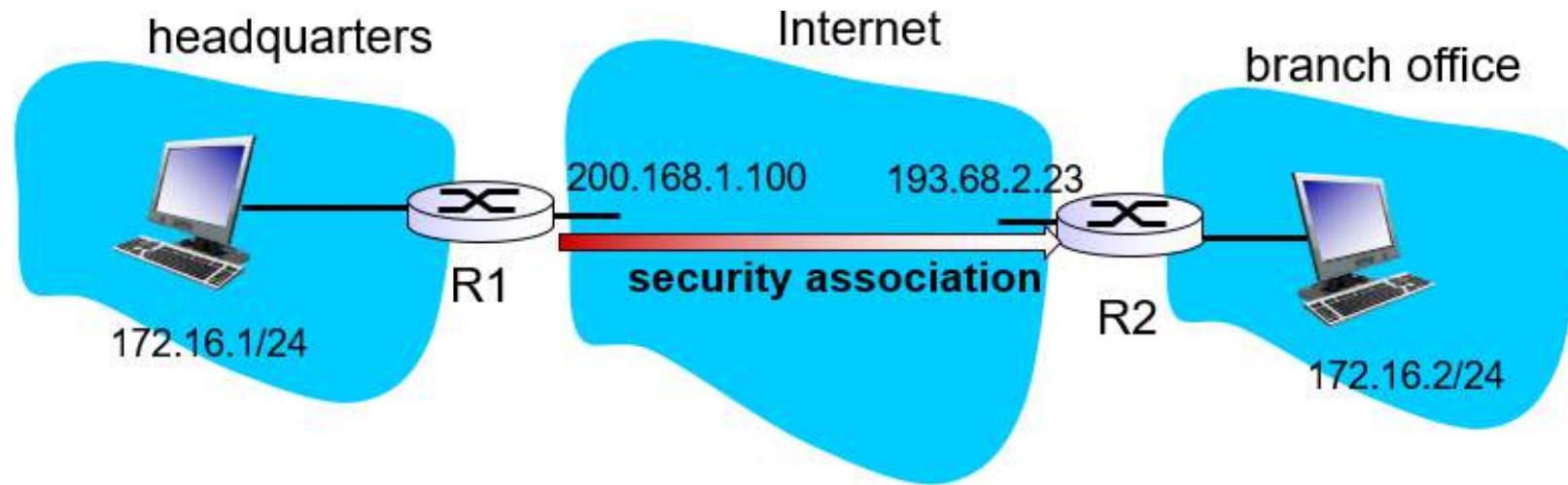
# Example SA from R1 to R2 (1 of 2)



**R1 stores for SA:**

- 32-bit SA identifier: **Security Parameter Index (SPI)**
- origin SA interface (200.168.1.100)
- destination SA interface (193.68.2.23)

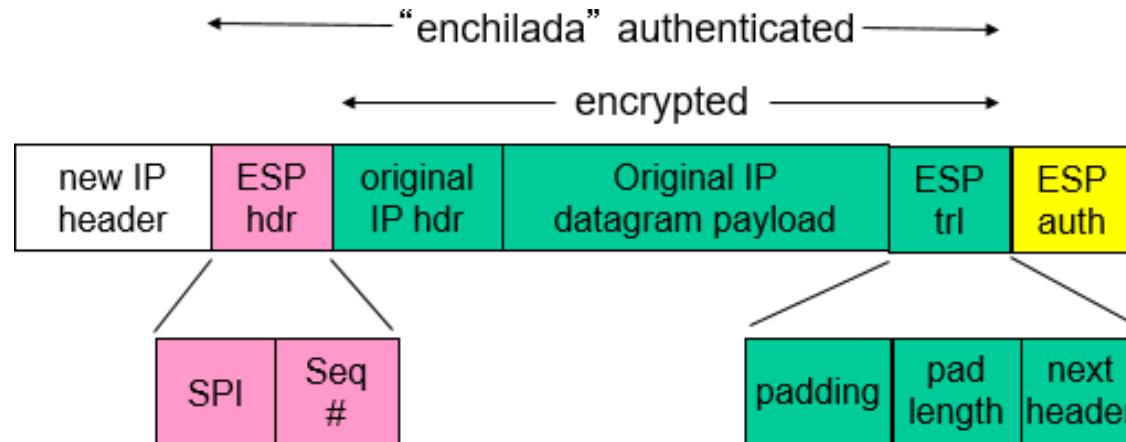
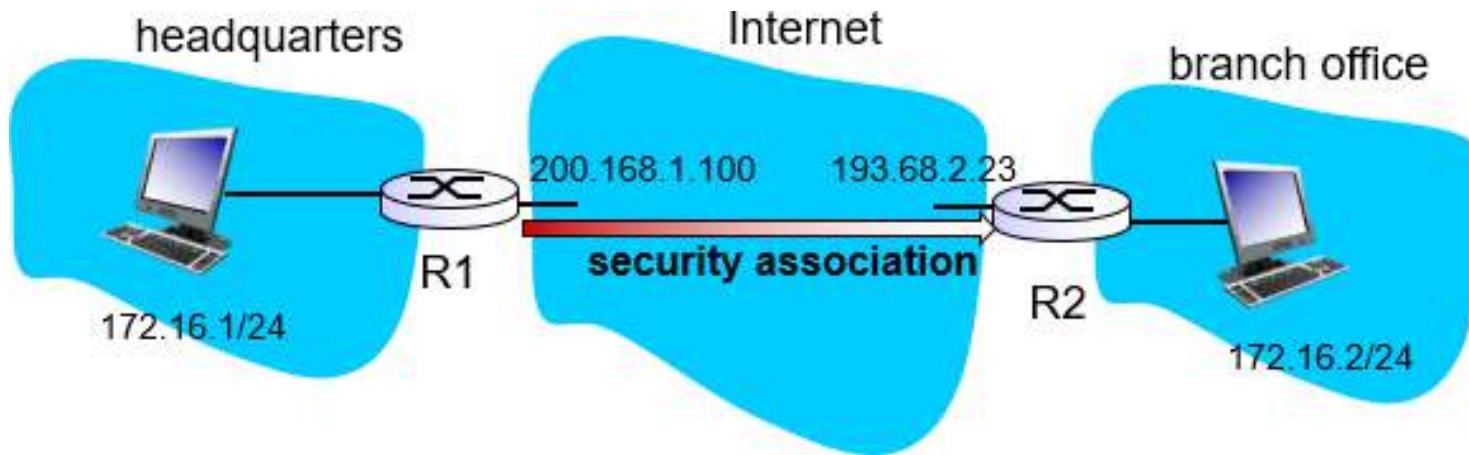
# Example SA from R1 to R2 (2 of 2)



## R1 stores for SA:

- type of encryption used (e.g., 3DES with CBC)
- encryption key
- type of integrity check used (e.g., HMAC with MD5)
- authentication key

# What Happens?



# IKE: Internet Key Exchange

- **previous examples:** manual establishment of IPsec SAs in IPsec endpoints:
  - **Example SA**
    - SPI: 12345
    - Source IP: 200.168.1.100
    - Dest IP: 193.68.2.23
    - Protocol: ESP
    - Encryption algorithm: 3DES-cbc
    - HMAC algorithm: MD5
    - Encryption key: 0x7aeaca...
    - HMAC key: 0xc0291f...
- manual keying is impractical for VPN with 100s of endpoints
- instead use **IPsec IKE (Internet Key Exchange)**

# IKE: PSK and PKI

- authentication (prove who you are) with either
  - pre-shared secret (PSK) or
  - with PKI (public/private keys and certificates).
- PSK: both sides start with secret
  - run IKE to authenticate each other and to generate IPsec SAs (one in each direction), including encryption, authentication keys
- PKI: both sides start with public/private key pair, certificate
  - run IKE to authenticate each other, obtain IPsec SAs (one in each direction).
  - similar with handshake in SSL.

# IPsec Summary

- IKE message exchange for algorithms, secret keys, SPI numbers
- either AH or ESP protocol (or both)
  - AH provides integrity, source authentication
  - ESP protocol (with AH) additionally provides encryption
- IPsec peers can be two end systems, two routers/firewalls, or a router/firewall and an end system

**SLUT**