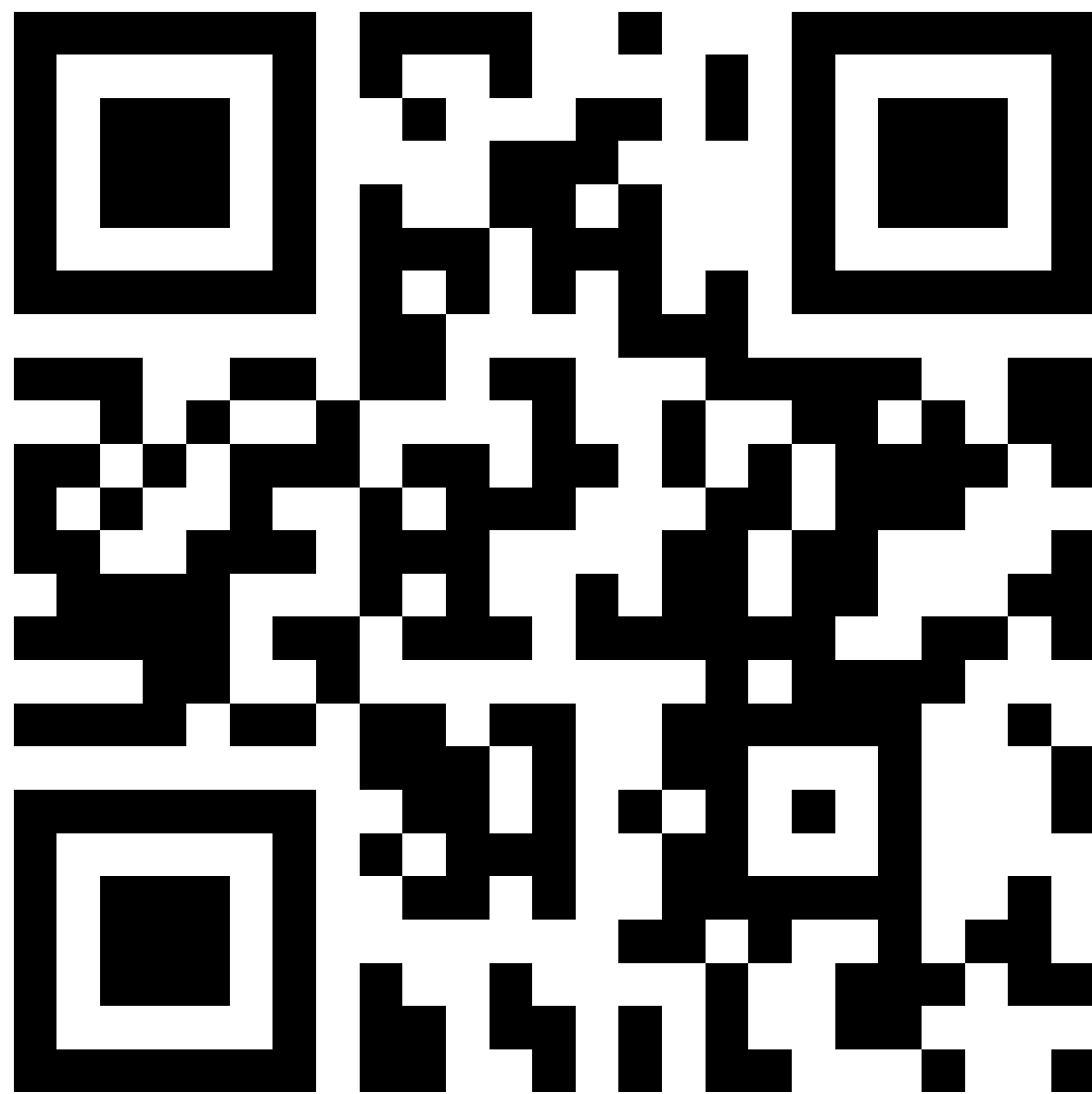


# Web Engineering

**DHBW Stuttgart** · Informatik / Wirtschaftsinformatik

**Sommersemester 2026**

<https://librete.ch/dhbw/>



# Ressourcen zum Selbstlernen

- **CODE CRISPIES** – [codecrispi.es](https://codecrispi.es)
- **Online Code-Editor** – [codepen.io](https://codepen.io)
- **MDN** (Mozilla Developer Network) – [developer.mozilla.org/de/](https://developer.mozilla.org/de/)
- **Flexbox-Spiel** – [flexboxfroggy.com](https://flexboxfroggy.com)
- **Grid-Spiel** – [cssgridgarden.com](https://cssgridgarden.com)

**Herzlich Willkommen!**

**1. Sitzung: 08.05.2026**

# Über mich

## Michael Werner Czechowski

- Systems and Platform Engineer
- Schwerpunkte:
  - Web-Technologien, Barriere-Armut, Open Source
- Hintergrund:
  - Philosophie (Uni Stuttgart)
  - Wirtschaftsinformatik (Leibniz-FH Hannover)
- Honorar-dozent (DHBW Stuttgart, HdM Stuttgart, Leibniz-FH)
- Kontakt: [michael.czechowski@lehre.dhbw-stuttgart.de](mailto:michael.czechowski@lehre.dhbw-stuttgart.de)



# Vorstellungsrunde

## Wer bist du, was machst du?

“ Ich bin \_\_\_, mache bei \_\_\_ \_\_\_ und ein Fun Fact über mich ist, dass ich ...

”

- ... sämtliche One Piece Bände gelesen habe.
- ... alle XKCD Comics mit Titel kenne.
- ... irgendwann bei \_\_\_ arbeiten möchte.
- ... Paulaner Spezi im Blindtest herausschmecken kann.

# Vorlesungsplan

|                                                      |               |                                            |               |                                                   |               |                                                     |               |
|------------------------------------------------------|---------------|--------------------------------------------|---------------|---------------------------------------------------|---------------|-----------------------------------------------------|---------------|
| <b>1</b><br>Intro · Projektgruppen<br>· Internet 101 | <b>08.05.</b> | <b>2</b><br>HTML und CSS<br>(Frameworks)   | <b>15.05.</b> | <b>3</b><br>JS (Frameworks) und<br>npm            | <b>22.05.</b> | <b>4</b><br>nodeJS: Scripting,<br>Running, Building | <b>29.05.</b> |
| <b>5</b><br>Express API, CRUD,<br>Middlewares        | <b>05.06.</b> | <b>6</b><br><b>Projektwerkstattbericht</b> | <b>12.06.</b> | <b>7</b><br>Testing (Unit ·<br>Integration · E2E) | <b>19.06.</b> | <b>8</b><br>TypeScript                              | <b>26.06.</b> |
| <b>9</b><br>Docker · Proxies · DBs                   | <b>03.07.</b> | <b>10</b><br>Wrap-up &<br>Projektsupport   | <b>10.07.</b> | <b>11</b><br><b>Präsentation</b>                  | <b>17.07.</b> |                                                     |               |

# Erwartungen und Wünsche

- Was haben wir bisher gelernt?
- Was kennen wir bereits aus dem Arbeitskontext?
- Welche Technologien nutzen wir bereits?
- Was möchte ich hier lernen: Frontend, Backend, DevOps?
- Eigene FOSS entwickeln?
- CLI-Skills verbessern?
- Cloud, Cloud, Cloud?

# Prüfungsleistung – Übersicht

Eigenes Projekt: **Web-Applikation XOR Backend** (API/CLI).

| Punkte     | Bereich                                               |
|------------|-------------------------------------------------------|
| 30         | Grundanforderungen (Projektgrundlagen + Präsentation) |
| 25         | Clean Code Prinzipien                                 |
| 25         | Accessibility & UX                                    |
| 20         | Technische Umsetzung                                  |
| <b>100</b> | <b>Basispunkte</b>                                    |
| +15        | Bonuspunkte (max. 3 Features a 5 P.)                  |

# Prüfungsleistung – Grundanforderungen (30 P.)

## Projektgrundlagen (15 P.)

- Funktionsfähige Anwendung (8 P.)
- Repository mit Versionskontrolle (4 P.)
- Projektdokumentation / README (3 P.)

## Präsentation (15 P.)

- Elevator Pitch (5 P.)
- Technische Präsentation (5 P.)
- Projektevolution: wie hat sich das Projekt vs. Ursprungsidee verändert? (5 P.)

# Prüfungsleistung – Clean Code (25 P.)

## Projektstruktur (10 P.)

- Ordnerstruktur (5 P.)
- Datei-Organisation (5 P.)

## Code-Qualität (10 P.)

- KISS – Keep It Simple, Stupid (3 P.)
- Beschreibende Namen (3 P.)
- Funktionsverantwortlichkeiten (4 P.)

## Kommentierung & Dokumentation (5 P.)

# Prüfungsleistung – Accessibility & UX (25 P.)

## Web Accessibility (15 P.)

- Semantisches HTML (5 P.)
- ARIA Labels & Alt-Texte (5 P.)
- Tastaturnavigation (5 P.)

## Responsive Design (10 P.)

- Mobile First Approach (5 P.)
- Flexbox/Grid Layout (5 P.)

# Prüfungsleistung – Technische Umsetzung (20 P.)

- Plattformunabhängigkeit (5 P.)
- Package Management (5 P.)
- Build Tools & Optimierung (5 P.)
- Live-Demo (5 P.)

# Prüfungsleistung – Bonuspunkte (max. 15 P.)

Mindestens 2 Bereiche wählen, je 5 Punkte:

- API Integration
- Local Storage / State Management
- Progressive Web App Features
- Testing
- Erweiterte Responsive Features (3+ Breakpoints)
- Security Features

# Prüfungsleistung – Abgabe & Termine

- **Code-Upload:** bis 27.07.
- **Präsentation:** 17.07. (Gruppen, ~10 Min.)

Details: <https://git.librete.ch/DHBW/pruefungsleistung>.

# Was ist (k)eine Web Applikation?

## Exkurs

# Was ist (k)eine Web Applikation?

## **Keine Web Applikation ist:**

- Betriebssystem (GNU/Linux, Windows, macOS, iOS, Android)
- Desktop-Applikation
- Mobile-Applikation
- Embedded System (Microcontroller, IoT)

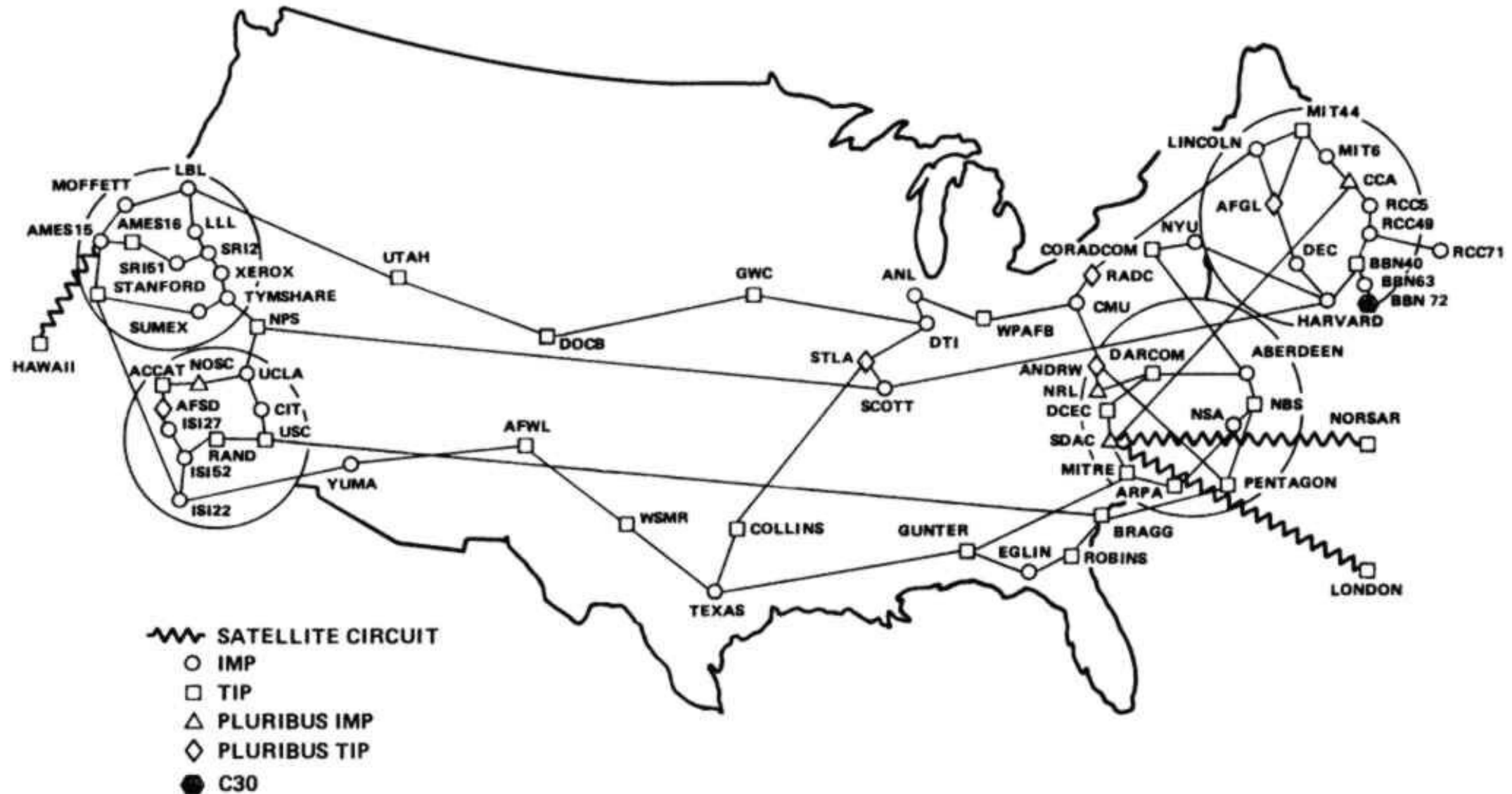
## **Eine Web Applikation ist:**

- plattformunabhängig (läuft im Browser)
- nutzt in der Regel HTTP-Requests
- keine Installation nötig – serverseitig aktualisiert
- gebaut in Web-Technologien (HTML, CSS, JS/TS)
- skalierbar durch verteilte Architektur

# Internet-Timeline

| Jahr | Meilenstein                                 |
|------|---------------------------------------------|
| 1966 | ARPANET-Projekt startet (DARPA)             |
| 1969 | Erste Verbindung zwischen UCLA und Stanford |
| 1974 | TCP/IP-Protokoll entwickelt                 |
| 1989 | Tim Berners-Lee erfindet das WWW (CERN)     |
| 1993 | Mosaic – erster grafischer Browser          |
| 1994 | W3C gegründet (HTML, CSS, SVG, WCAG)        |
| 1995 | ECMAScript / JavaScript wird Standard       |
| 2008 | V8 JavaScript Engine (Chrome / Node.js)     |

# ARPANET GEOGRAPHIC MAP, OCTOBER 1980



(NOTE: THIS MAP DOES NOT SHOW ARPA'S EXPERIMENTAL SATELLITE CONNECTIONS)  
 NAMES SHOWN ARE IMP NAMES, NOT (NECESSARILY) HOST NAMES

# Das WWW (1989)

**Tim Berners-Lee** am **CERN** (Genf)

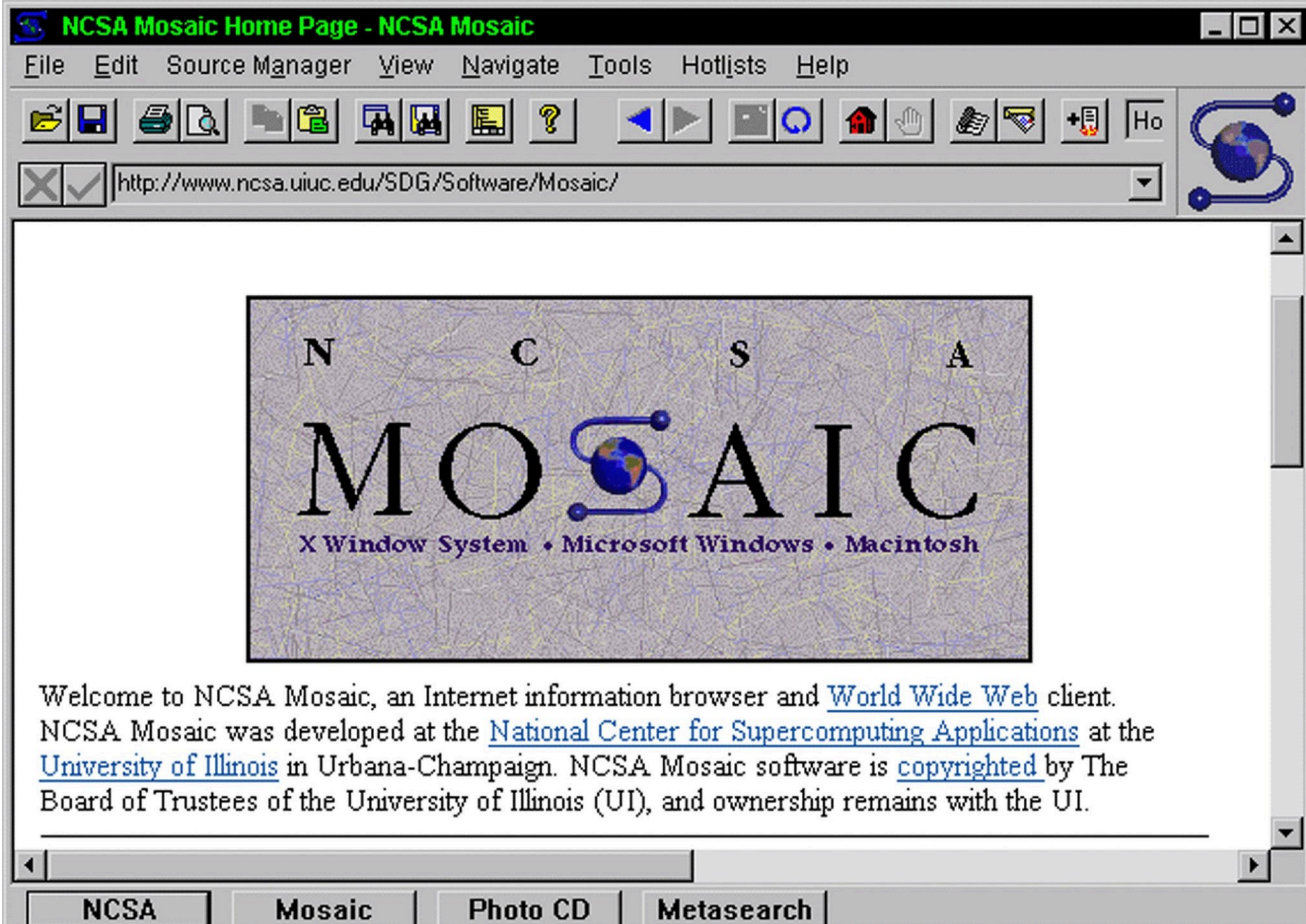
**Problem:** Wissenschaftler:innen können Forschungsdokumente nicht verlinken – Inkompatible Systeme, kein gemeinsames Format.

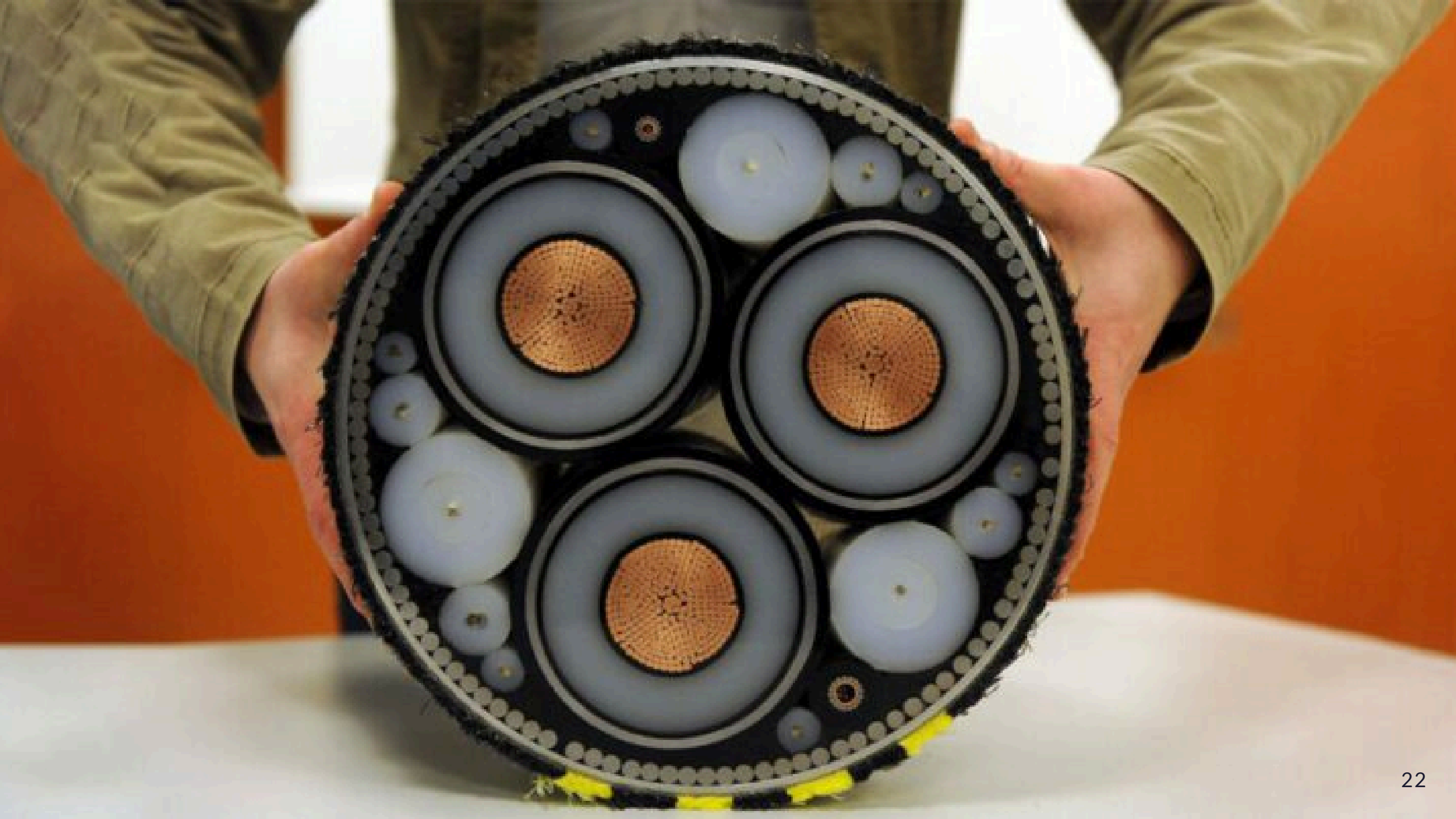
**Lösung – drei Bausteine:**

- **HTML** – Auszeichnungssprache für Dokumente
- **HTTP** – Protokoll zur Übertragung
- **URL** – eindeutige Adressen

**1991:** erste Webseite geht online • 1993 vom CERN gemeinfrei freigegeben.

→ WWW ≠ Internet. Das WWW ist eine **Anwendung auf** dem Internet.





# Heute: Physische Infrastruktur

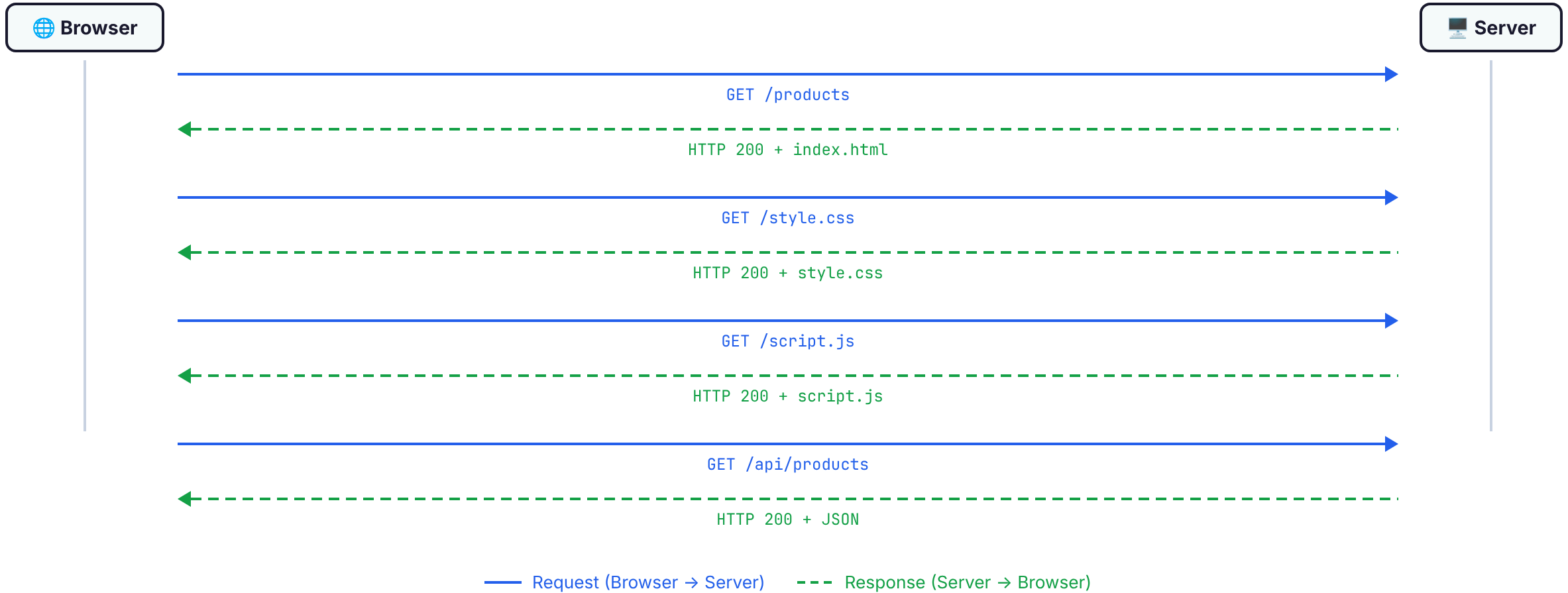
- **5,5 Milliarden** Menschen online
- **> 400 Unterseekabel**, ca. **1,3 Mio. km** Glasfaser
- Daten reisen mit **Lichtgeschwindigkeit**
- Latenz Frankfurt – New York: ~80 ms

→ Keine "Cloud" ohne Kabel. Das Internet ist **physische** Infrastruktur – und damit angreifbar.

**Geopolitik:** Five-Eyes-Allianz (USA, UK, CA, AU, NZ) zapft Unterseekabel ab (Programm "Tempora", aufgedeckt 2013 durch Snowden). DE ist Teil der Fourteen Eyes.

# Browser-Server: HTTP Kommunikation

Jede Ressource = eigener Request — Browser blockiert nicht, lädt parallel



# HTTP – Hypertext Transfer Protocol

## Request:

```
GET /products HTTP/1.1  
Host: localhost:8080
```

## Response:

```
HTTP/1.1 200 OK  
Content-Type: application/json  
  
[{"id": 1, "name": "Produkt A"}, ...]
```

# Onboarding

## Tools, Runtimes, Workflows

# Werkzeuge & Betriebssystem

## Erforderliche Werkzeuge:

- **Terminal** (zsh / bash / fish / pwsh) – Kommandozeile
- **Editor** (siehe folgende Folie)
- **JavaScript-Runtime** (Node.js / Bun / Deno)
- **Browser mit DevTools** (Chromium / Firefox / Safari)
- **Git** – verteiltes Versionskontrollsystem (Stand der Industrie)

## OS-Empfehlung:

| OS                         | Status                                 |
|----------------------------|----------------------------------------|
| macOS                      | First-class, viele Tools "just work"   |
| Linux                      | First-class, beste Server-Parität      |
| Windows + WSL2             | Empfohlen – Windows-CLI hat Edge Cases |
| Windows nativ (PowerShell) | Funktioniert, aber holpriger Pfad      |

# Node.js installieren – macOS

Empfohlen: **LTS** (aktuell **Node.js 24** – Stand 2026).

Paketmanager: Homebrew

```
# Homebrew installieren (falls nicht vorhanden)
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Node.js LTS installieren
brew install node@24

# Version prüfen
node --version
npm --version
```

Alternative: Installer von nodejs.org.

# Node.js installieren – Linux

Empfohlen: **LTS** (aktuell **Node.js 24**).

**Debian / Ubuntu** ([NodeSource](https://nodejs.org/en/download/package-manager/#debian-ubuntu)):

```
curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -  
sudo apt install -y nodejs
```

**Arch / Manjaro:**

```
sudo pacman -S nodejs npm
```

**Fedora / RHEL:**

```
sudo dnf module install nodejs:24/common
```

→ Distro-Pakete sind oft **veraltet**. Versionsmanager (folgende Folie) bevorzugen.

# Node.js installieren – Windows

Empfohlen: **LTS** (aktuell **Node.js 24**).

**Option 1: winget** (empfohlen, Win 10/11):

```
winget install OpenJS.NodeJS.LTS
```

**Option 2: Chocolatey** ([chocolatey.org](https://chocolatey.org)):

```
choco install nodejs-lts
```

**Option 3: Scoop** ([scoop.sh](https://scoop.sh)):

```
scoop install nodejs-lts
```

**Option 4:** Installer von [nodejs.org](https://nodejs.org).

→ **Empfehlung:** WSL2 + Linux-Workflow nutzen.

# Direkt-Install: Wenn es eng wird

**Typisches Szenario** – drei parallele Projekte mit unterschiedlichen Anforderungen:

| Projekt               | Stack            | Node-Version             |
|-----------------------|------------------|--------------------------|
| Praktikums-API        | Express (Legacy) | 18.x (LTS Hydrogen)      |
| Studienarbeit         | Next.js 15       | 22.x (LTS Jod)           |
| Aktuelles Lab-Projekt | neueste Features | 24.x (Iron, current LTS) |

System-Install kennt nur **eine** Node-Version global → Konflikt.

# Symptome im Alltag

- `npm install` bricht mit `engines`-Warning ab
- native Module (z. B. `node-sass`) bauen nicht
- CI-Pipeline ist grün, lokal rot – oder umgekehrt
- erheblicher Zeitverlust beim Debuggen von Versions-Konflikten

→ Lösung: **Versionsmanager** und Node-Version pro Projekt pinnen.

# Node-Versionsmanager

Verschiedene Projekte brauchen oft verschiedene Node-Versionen. **Lösung:** Versionsmanager.

| Tool         | Sprache | Geschwindigkeit                                | Plattform             |
|--------------|---------|------------------------------------------------|-----------------------|
| <b>nvm</b>   | Bash    | langsam (Shell-Funktion)                       | macOS, Linux          |
| <b>fnm</b>   | Rust    | sehr schnell                                   | macOS, Linux, Windows |
| <b>asdf</b>  | Bash    | mittel, multi-language                         | macOS, Linux          |
| <b>volta</b> | Rust    | schnell, pinning per <code>package.json</code> | alle                  |

```
# fnm
curl -fsSL https://fnm.vercel.app/install | bash
fnm install --lts          # aktuell Node 24
fnm use lts-latest

# .nvmrc / .node-version automatisch
echo "lts/*" > .nvmrc
fnm use # liest .nvmrc
```

# Alternative Runtimes

| Runtime        | Erschienen | Engine               | USP                                   |
|----------------|------------|----------------------|---------------------------------------|
| <b>Node.js</b> | 2009       | V8 (C++)             | Standard, größtes Ökosystem           |
| <b>Deno</b>    | 2020       | V8 (Rust)            | Sicherheit, TypeScript nativ, Std-Lib |
| <b>Bun</b>     | 2022       | JavaScriptCore (Zig) | Geschwindigkeit, all-in-one           |

```
# Deno
curl -fsSL https://deno.land/install.sh | sh
deno run main.ts

# Bun
curl -fsSL https://bun.sh/install | bash
bun install
bun run dev
```

→ Im Kurs primär **Node.js**. Bun und Deno sind ebenfalls zulässig.

# Editor (IDE)

- **VS Code** (empfohlen) – kostenlos, große Plugin-Auswahl
- **JetBrains WebStorm** – sehr mächtig, kostenpflichtig
  - kostenlos für Studierende via [JetBrains Education](#)
- **Vim / Neovim / Emacs** – für CLI-Profis
- **Zed** – modern, schnell

# Browser DevTools

Chromium / Firefox / Safari — **F12** oder **Cmd+Opt+I**

Chromium = Open-Source-Basis von Chrome, Edge, Brave, Opera, Vivaldi etc.

## Wichtigste Panels:

- **Network** — HTTP-Requests, Responses, Timing
- **Console** — **console.log**, JS-Errors, REPL
- **Elements** — DOM-Inspektor, Live-CSS-Editor
- **Sources** — Debugger, Breakpoints, Source-Maps
- **Performance** — Profiling, Frame-Rate, FPS
- **Application** — LocalStorage, Cookies, Service Workers

# HTTP-Clients

- **Bruno** ([usebruno.com](https://usebruno.com)) – open source, git-friendly
- **Hoppscotch** ([hoppscotch.io](https://hoppscotch.io)) – open source, browser-basiert (vormals Postwoman)
- **Insomnia** / **Postman** – etabliert, freemium
- **curl** / **httpie** – CLI (Command-Line Interface, scriptbar)

# Git Setup & Workflow

## Einmalige Konfiguration:

```
git config --global user.name "Vorname Nachname"
git config --global user.email "vorname.nachname@example.com"
git config --global init.defaultBranch main
git config --global pull.rebase true
```

## SSH-Key für GitHub/GitLab:

```
ssh-keygen -t ed25519 -C "vorname.nachname@example.com"
cat ~/.ssh/id_ed25519.pub # → in GitHub-Settings einfügen
```

## Tägliche Befehle:

```
git status           # was hat sich geändert?
git add <file>       # für Commit vormerken
git commit -m "feat: ..." # Conventional Commits
git push             # hochladen
git pull --rebase    # neueste Änderungen holen
```

# Projekt-Skeleton

```
mkdir mein-projekt && cd mein-projekt
git init
npm init -y           # erzeugt package.json
echo "node_modules" > .gitignore
echo ".env" >> .gitignore
npm install express
```

**package.json** minimal:

```
{
  "name": "mein-projekt",
  "version": "0.1.0",
  "type": "module",
  "scripts": {
    "start": "node index.js",
    "dev": "node --watch index.js"
  }
}
```

**.env** für Secrets (nie committen!), **.env.example** mit Dummy-Werten committen.

