



enabling the adaptive enterprise

Wird JavaScript abgelöst? Einblicke in WebAssembly

Philipp Hartenfeller | MT AG

DOAG K+A 2021
17.11.2021

Unsere Hard Facts



Gründung
1994



Inhabergeführt



ca. 36 Mio. Euro
Umsatz in 2020



>100 Kunden



Hauptsitz
Ratingen



300
Beschäftigte



Ihr Partner für den digitalen Wandel

Individuelle IT-Lösungen aus einer Hand



Zertifizierter
Partner führender
Technologie-
hersteller



Herstellerneutral



Branchen-
übergreifend



Ausbildungsbetrieb,
Partner im
dualen Studium



Philipp Hartenfeller

Seit 2016 @ MT AG

APEX / DBs / Web / JavaScript

Aus Düsseldorf

Blog: <https://hartenfeller.dev/blog/>

 [@phartenfeller](https://twitter.com/phartenfeller)

Das Web - eine Plattform für alles



Fundraiser Donation

File Edit View Insert



100%



fx

	D	E
1	September	October
2	\$78.00	10/2/201
3	\$94.00	10/3/201
4	\$68.00	10/4/201
5	\$72.00	10/5/201
6	\$85.00	10/6/201
7	\$88.00	10/7/201
8	\$89.00	10/8/201
9	\$99.00	10/9/201
10	\$96.00	10/10/201
11	\$73.00	10/11/201
12	\$85.00	10/12/201
13	\$94.00	10/13/201
14	\$71.00	10/14/201
15	\$62.00	10/15/201
16	\$77.00	10/16/201
17	\$89.00	10/17/201
18	\$73.00	10/18/201
19	\$99.00	10/19/201
20	\$95.00	10/20/201
21	\$61.00	10/21/201

Mail

Inbox

Starred

Snoozed

Sent

Drafts

Chat

Jeffery Clark

Sounds great!

Shirley, Jeffery

Awesome, thanks.

Helen, Adam, Grego

Can we reschedule the

Meeting?

Rooms

Project Clover

Customer Success

Marketing updates

Project Skylight

My meetings

Meet

New meeting

My meetings

VanArsdel Journal

vanarsdel.sharepoint.com/teams/digitalmarketing/weeklyjournal

Word


VanArsdel Journal – Saved

File Home Insert Layout References Review View Help Tell me what you want to do Edit in App

Calibri 11

Heading 1

VanArsdel Digital Marketing



Bring your ideas to life.

Mission

Empowering a new era of personal productivity.

As we work to deliver on our company mission of empowering

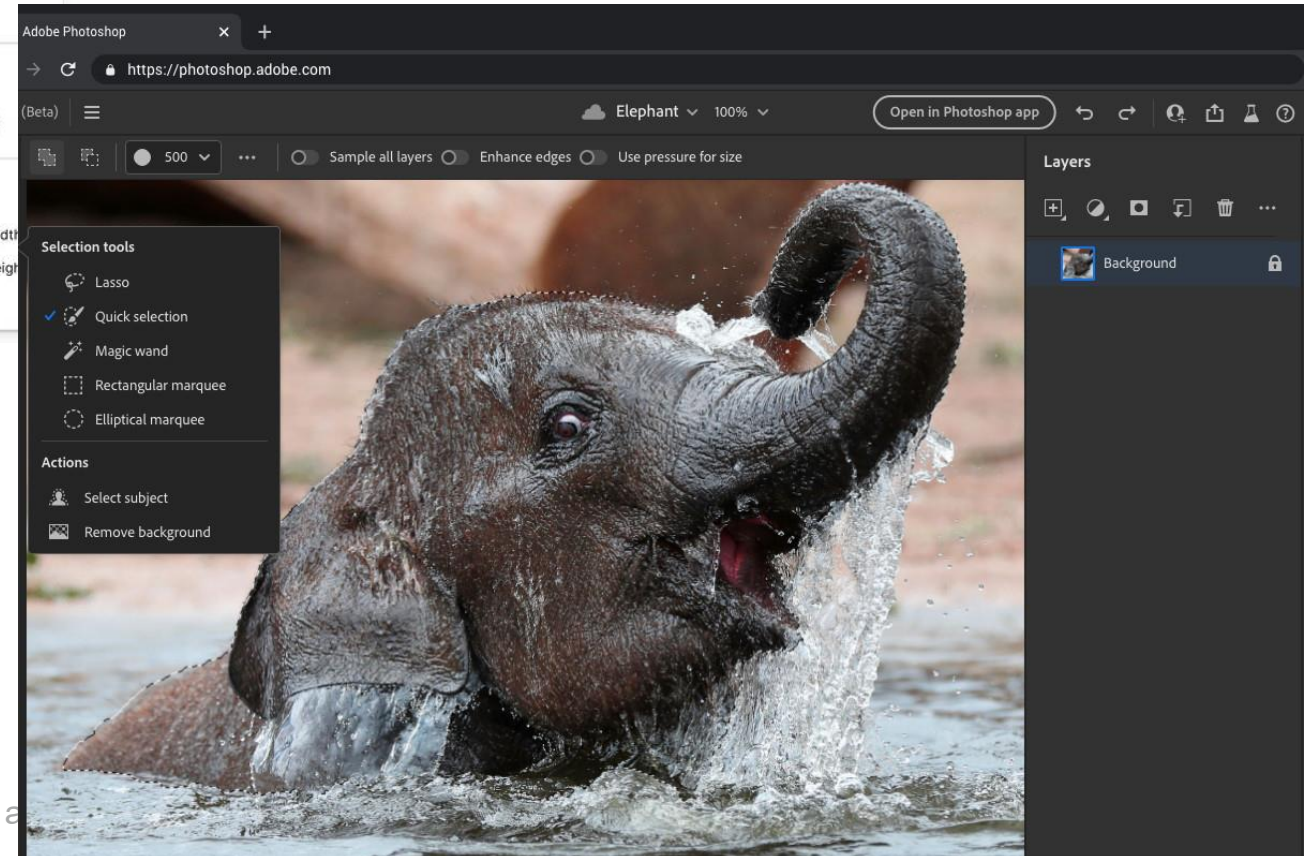
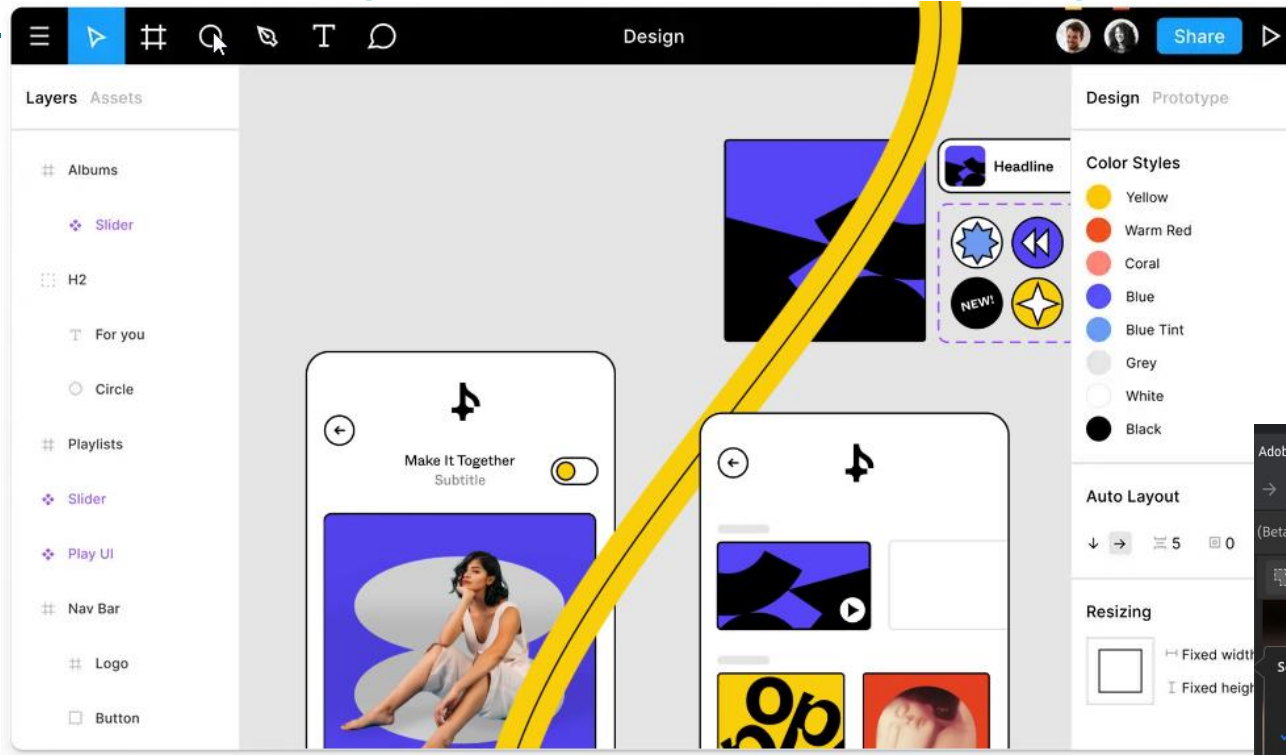
Kommunikation

The screenshot displays a Microsoft Teams interface. On the left, the navigation pane shows 'Fabrikam Designs' with a 'Stock Inventory' link. The main area shows a spreadsheet titled 'Fabrikam Designs_Inventory.xlsx' with the following data:

SKU	DESCRIPTION	BIN #
BESPOKE	Vintage Persian	FD49
CECIL	Cecil rugs	FD21
PADS	Rug backing pads	FD83
PLUSH_IN	Indigo plush	FD01
PLUSH_R	Red plush	FD72
PLUSH_WH	White plush	FD27
SHAG_BL	Black shag	FD33
SHAG_WH	White shag	FD52
WOOL_G	Grey wool	FD11

Below the spreadsheet, there are tabs for 'Inventory List', 'Inventory Pick List', and 'Bin Lookup'. Overlaid on the right is a FaceTime window with the text 'FaceTime' and 'Enter your name to join the conversation.' Below this is a 'Name' input field and a 'Continue' button. At the bottom of the FaceTime window, there is a 'Diagnostic Log' link and a 'Beta' label in the bottom right corner.

Design / Bildbearbeitung



Was steckt dahinter / der Browser als OS-Schnittstelle

Allgegenwärtigkeit:

- Fast jedes bedienbare Gerät hat einen Browser
- Responsive Design
- Verlinkbar, durchsuchbar, indiziert
- Keine Installation notwendig
- Barrierefrei

Sicherheit:

- Browser Sandboxing
- SSL / TLS -> HTTPS

Technologische Fortschritte

- Client-Side-Rendering
- PWAs:
 - Push-Benachrichtigungen
 - Dateisystem-API
 - Installierbarkeit
 - Offline-Apps

- Web Components
- Multithreading / Workers
- WebGPU / WebGL
- WebXR
- WebUSB / WebSerial
- WebMIDI
- **Just-in-Time Kompilierung**
- **WebAssembly**

Was ist WebAssembly?









WA

WebAssembly – ein Web Standard

- Low-Level + Assembly ähnliche Sprache, die in modernen Browsern läuft
- Versprechen: fast native Performanz
- Kann als Kompilierungsziel vieler bestehender Sprachen verwendet werden
- Standardisiert durch die W3C
- Entstand aus asm.js
- Erschienen in 2017
- Aktuelle Version 1.1 (Draft vom 03.11.2021)
 - <https://webassembly.github.io/spec/core/>



In allen modernen Browsern unterstützt

	Your browser	 Chrome ⁹⁵	 Firefox ⁹⁰	 Safari ^{15.0}	 Wasmtime ^{0.31}	 Wasmer ^{2.0}	 Node.js ^{16.4}
Standardized features							
JS BigInt to Wasm i64 integration	✓	✓	✓	✓	n/a	n/a	✓
Bulk memory operations	✓	✓	✓	✓	✓	✓	✓
Multi-value	✓	✓	✓	✓	✓	✓	✓
Import & export of mutable globals	✓	✓	✓	✓	✓	✓	✓
Reference types	✗	⌚	✓	✓	✓	✓	⌚
Non-trapping float-to-int conversions	✓	✓	✓	✓	✓	✓	✓
Sign-extension operations	✓	✓	✓	✓	✓	✓	✓
Fixed-width SIMD	✓	✓	✓	✗	⌚	✓	✓
In-progress proposals							
Exception handling	✓	✓	⌚	✗	✗	✗	⌚
Extended constant expressions	✗	✗	⌚	✗	✗	✗	✗
Memory64	✗	✗	⌚	✗	⌚	✗	✗
Module Linking	✗	✗	✗	✗	⌚	✗	✗
Tail calls	✗	⌚	✗	✗	✗	✗	⌚
Threads and atomics	✓	✓	✓	⌚	✗	⌚	✓

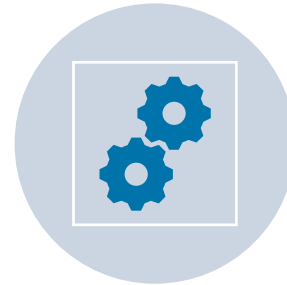
Quelle: Roadmap

4 Grundsätze von WASM



Safe

Code von unbekannten Webseiten wird ausgeführt



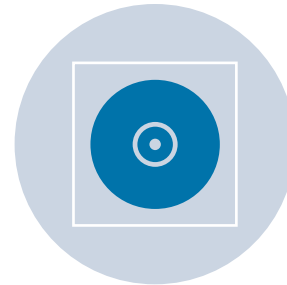
Fast

Maschinennah, vor der Ausführung optimiert



Portable

Plattform- und Hardwareübergreifend



Compact

Bandbreitenschonend

Quelle: Haas et. al. 2017 - 10.1145/3062341.3062363

WebAssembly und JavaScript

WA

JS

Codeausführung – Kompilierung vs. Interpretation

Kompilierung

- Vorab wird alles in Maschinencode umgewandelt,
dann erst ausgeführt

Nachteile für Browser

- Verzögerte Ausführung
- Architekturgebunden

Interpretation

- Zeile für Zeile wird übersetzt
- Dann sofort ausgeführt

Nachteile für Browser

- Code muss zunächst geparkt werden
- Nicht optimal bei wiederkehrenden Aufgaben wie Schleifen

→ JavaScript



Just-in-Time (JIT) Kompilierung

Vermischung von Interpretation und Kompilierung

- Zunächst wird alles interpretiert
- Öfters aufgerufene Teile (z. B. Schleifen) werden Kompiliert

Folgen:

- Ca. 10x mal schneller
(und seit 2008 dank zahlreicher Optimierungen noch 4x schneller)
- JS als Serversprache (Node.js, Deno) und in vielen Desktopanwendungen

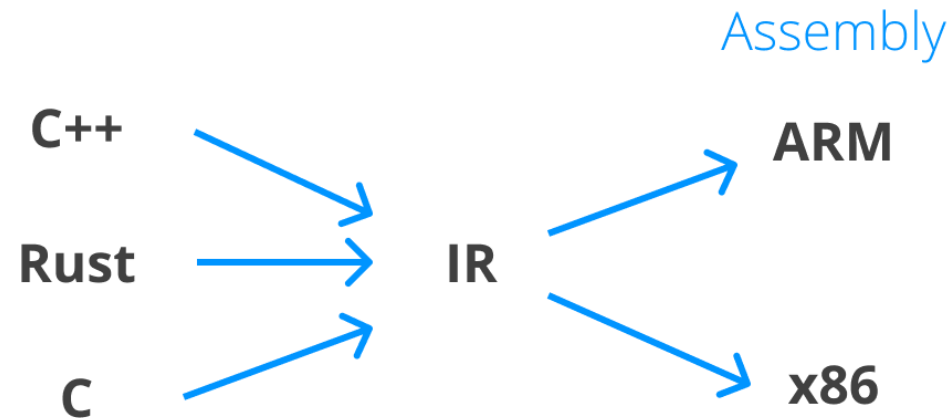
Justin Time?!



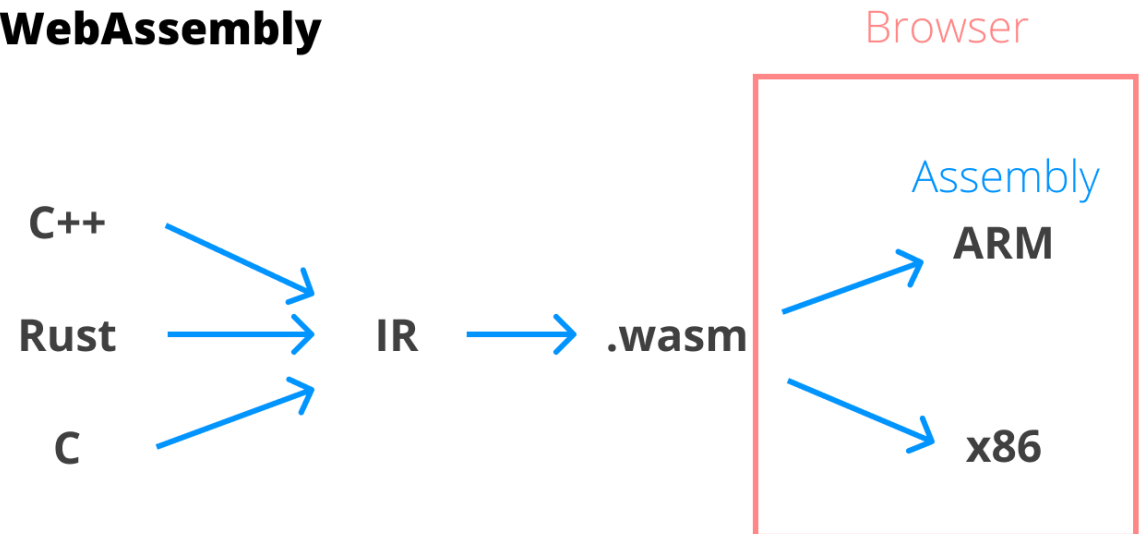
[Quelle: Pinterest](#)

WASM wird **V**orkompiliert

Standard



WebAssembly



IR = Intermediate Representation

Quelle: Lin Clark - Mozilla Hacks

Versprechen WASM gegenüber JS

- Kompilierung aus maschinennahem Code schneller als JIT-Methode
- Keine Nachoptimierung während der Laufzeit
- Keine „garbage collection“ weil Arbeitsspeicher manuell verwaltet werden muss
- Binärrepräsentation kleiner als JavaScript (in der Theorie)

WebAssembly in der Praxis

Importieren einer .wasm Datei im Browser



















```
WebAssembly.instantiateStreaming(fetch('simple.wasm'), importObject)
    .then(obj => obj.instance.exports.exported_func()
);
```

Sprachen die nach WASM kompilieren können

 .Net
 AssemblyScript
 Astro Unmaintained
 Brainfuck
 C
 C#
 C++
 Clean
 Co
 COBOL
 D
 Eel
 Elixir
 F#
 Faust
 Forest
 Forth
 Go

 Grain
 Haskell
 Java
 JavaScript
 Julia
 Idris Unmaintained
 Kotlin/Native
 Kou
 Lisp
 Lobster
 Lua
 Lys
 Never
 Nim
 Ocaml
 Perl
 PHP
 Plorth
 Poetry

 Python
 Prolog
 Ruby
 Rust
 Scheme
 Scopes
 Speedy.js Unmaintained
 Swift
 Turboscript Unmaintained
 TypeScript
 Wah Unmaintained
 Walt Unmaintained
 Wam Unmaintained
 WebAssembly
 Wracket Unmaintained
 Zig

Quelle: GitHub (appcypher)

Beispiel AssemblyScript

You, 18 hours ago | 1 author (You)

```
function isPrime(num: i32): bool {
  for (let i = 2; i <= Math.sqrt(num); i++) {
    if (num % i == 0) return false;
  }
  return true;
}

export function primeFactorization(num: i32): Int32Array {
  const primeArr = new Int32Array(10);
  let arrayI: i32 = 0;

  for (let i = 2; i < num; i++) {
    while (num % i == 0 && isPrime(i)) {
      primeArr[arrayI] = i;
      arrayI++;
      num /= i;
    }
  }
  if (num > 1) primeArr[arrayI] = num;
  return primeArr;
}
```

- TypeScript mit erweiterten Typen
- Inklusive Runtime: Memory-Management und Garbage-Collection
- Standardlibs wie „Math“ und „console“ sind verfügbar
- Minimale Unterschiede

```
asc assembly/index.ts --target release --exportRuntime
```

Beispiel C++

```
#include <cmath>
#include "primeFactor.h"

const unsigned int arrLen = 10;

bool isPrime(unsigned int num) {
    for (unsigned int i = 2; i < sqrt(num); i++) {
        if (num%i == 0) {
            return false;
        }
    }
    return true;
}

uint32_t* prime(unsigned int x) {
    unsigned int num = x;
    uint32_t primeArr[arrLen];
    unsigned int arrayI = 0;

    for (unsigned int i=2; i<x; i++) {
        while (num % i == 0 && isPrime(i)) {
            primeArr[arrayI] = i;
            arrayI++;
            num = num / i;
        }
    }

    if (num > 1) {
        primeArr[arrayI] = num;
    }

    // fill the rest of the array with 0's
    while (arrayI < arrLen) {
        primeArr[arrayI] = 0;
        arrayI++;
    }

    auto arrayPtr = &primeArr[0];
    return arrayPtr;
}
```

- Standard C / C++ Programmierung
- Compiler: Emscripten

```
em++ ../primeFactor.cpp -s WASM=1 -s STANDALONE_WASM -s EXPORT_ALL=1 -o hello.js
```

JS-Array aus zurückgegebenen Array-Pointer erstellen:

```
let result = Module.__Z5primej(num);
const arrayData = [];

for (let v = 0; v < MAX_ARRAY_LEN; v++) {
    arrayData.push(Module.HEAPU32[result / Uint32Array.BYTES_PER_ELEMENT + v]);
}
```



Demo

Surma - Is WebAssembly magic performance pixie dust?

„V8 is **really** good at executing JavaScript.
While WebAssembly can run faster than JavaScript, it is likely that you will have to hand-optimize your code to achieve that.
I could see this balance shift once there is wide-spread support for SIMD and Threads and a good developer experience around utilizing them.”

[Quelle](#)

Weitere Studien zur Performanz

- Jangda et. al. 2019 – DOI: 10.5555/3358807.3358817
 - Native C / C++ (Clang) vs. WASM Code (Emscripten)
 - WASM durchschnittlich 1.5x so langsam
 - Gründe: Compiler noch nicht so ausgereift und zusätzliche Sicherheitschecks
- Yan et. al. 2021 – DOI: 10.1145/3487552.3487827
 - Performanz von WASM variiert stark je nach Compiler
 - Unterschiede je nach Optimierung: Rust -> WASM 20 % langsamer wenn Kompiliert nach Größeneffizienz anstatt Performanz
 - Performanzunterschiede zwischen WASM und JS variieren stark je nach Szenario und Browser
 - WASM nutzt deutlich mehr Speicher als JS

Will WASM wirklich JS verdrängen?

Webseiten entwickeln für Backend-Entwickler ohne JS lernen zu müssen

Web frameworks-libraries

- `asm-dom` - A minimal WebAssembly virtual DOM to build C++ SPA
- `Blazor` - Microsoft's web UI framework using C#/Razor and HTML, running client-side via WebAssembly
- `Yew` - Rust framework for making client web apps
- `Perspective` - Streaming pivot visualization via WebAssembly
- `go-vdom-wasm` - Webassembly VDOM to create web application using Golang(experimental)
- `seed` - A Rust framework for creating web apps
- `Vugu` - A modern UI library for Go+WebAssembly
- `Vecty` - Lets you build responsive and dynamic web frontends in Go using WebAssembly

[Quelle: GitHub \(mbasso\)](#)

Schach-Engines

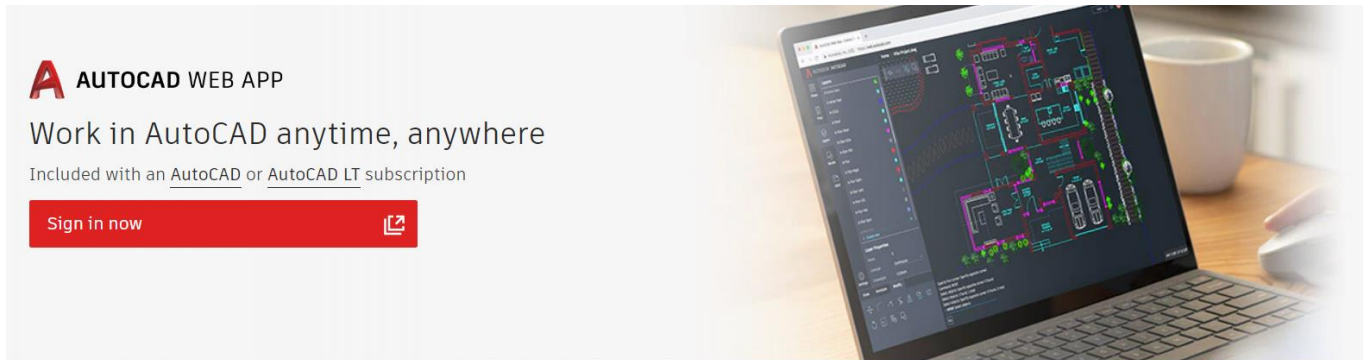


Schachengine Stockfish

C++ Port

<https://github.com/niklasf/stockfish.wasm>

Computer Aided Design (CAD)



Overview

Web app intro
Benefits
Partnerships
Features

Support & learning

Overview

Get quick, anytime access to CAD drawings with the AutoCAD web app

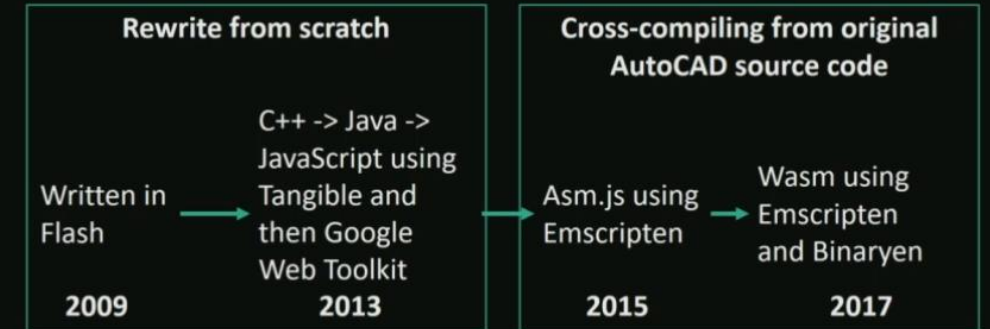
Edit, create, share, and view CAD drawings in a web browser on any computer. Just sign in and get to work –no software installation needed.

- Use familiar AutoCAD® drafting tools online in a simplified interface.
- Access and update DWG™ files from anywhere.
- Get the app included with an AutoCAD or AutoCAD LT subscription.

See also AutoCAD mobile app



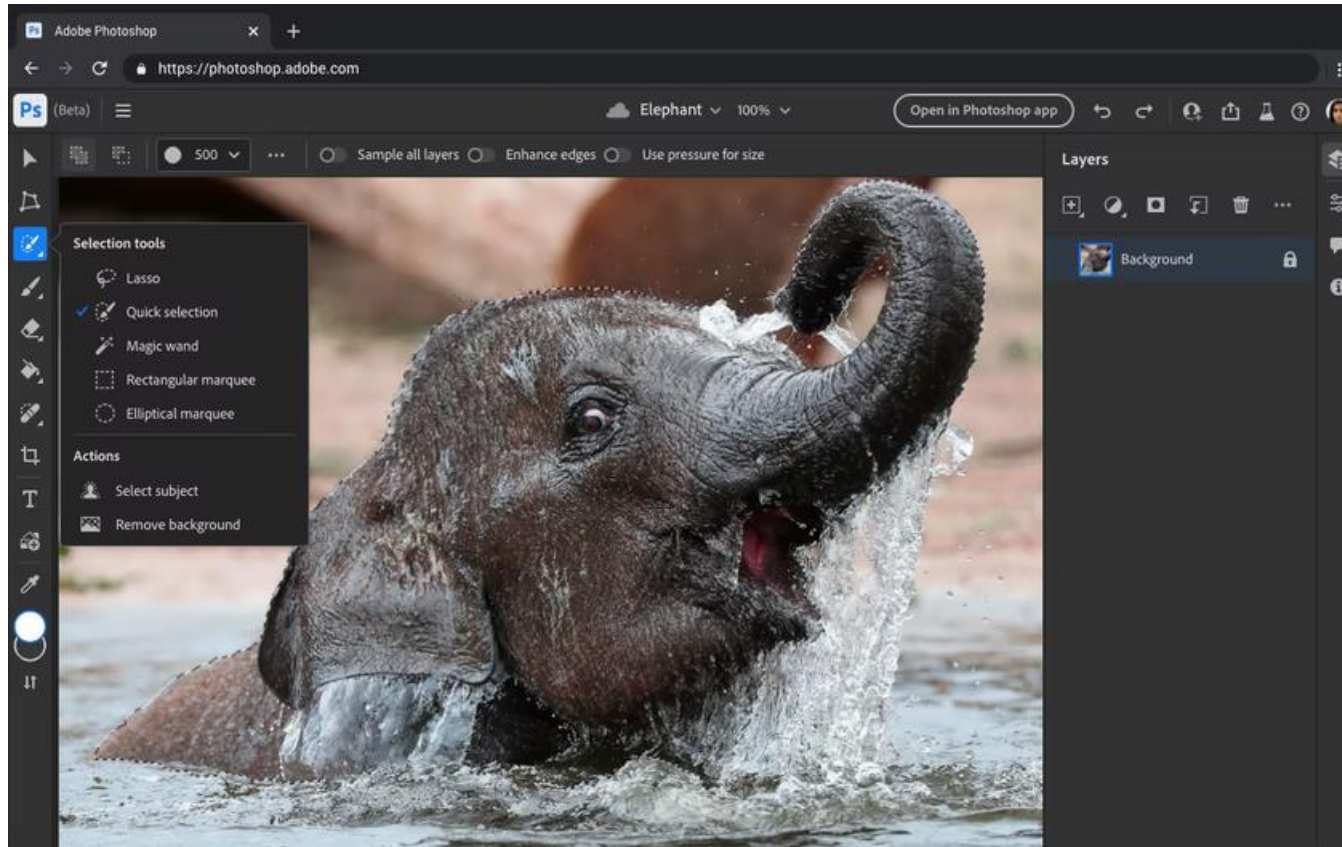
AutoCAD's Journey to the Web



WebAssembly ist nicht nur Berechnung sondern auch UI-Interaktion

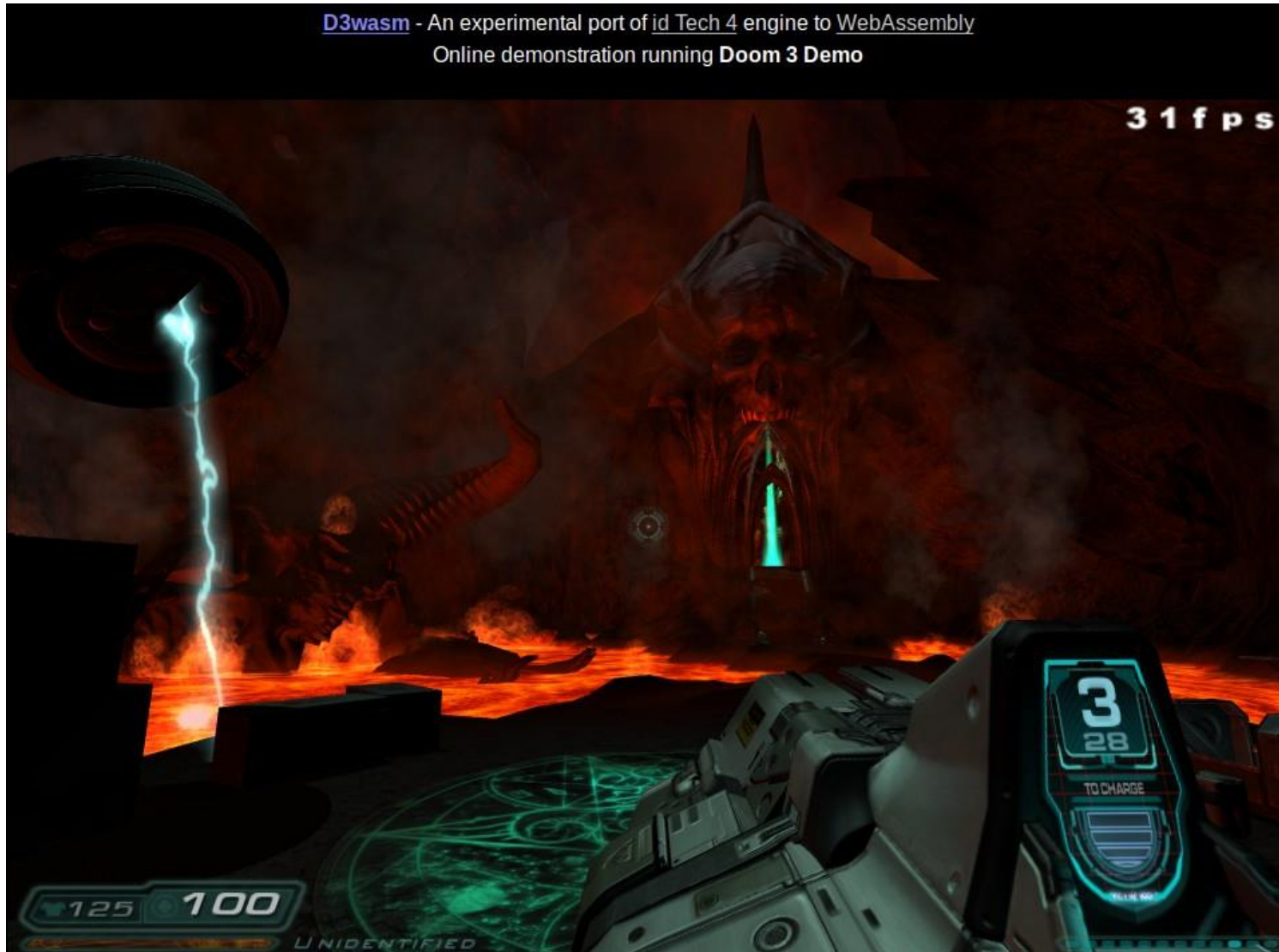
<https://www.infoq.com/presentations/autocad-webassembly/>

Photoshop



C++ Port mit Emscripten
UI mit WebComponents realisiert
<https://web.dev/ps-on-the-web/>

Computerspiele: Doom 3 Demo



380 MB Download nötig

C++ Port

<https://wasm.continuation-labs.com/d3demo/>

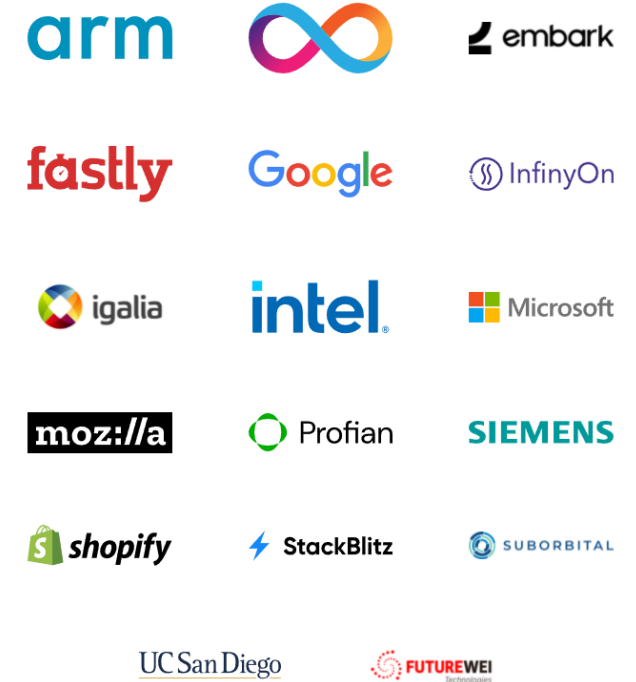
WASM als Laufzeitumgebung für unbekannten Code

- Problem: viel unbekannter Code durch Dependencies
- Lösung: Laufzeitumgebung für Dependencies in „Nanoprozessen“
 - Isoliert und Sandboxed: fein einstellbare Systemrechte pro Modul
 - Schnelle Schnittstelle und wenig Overhead durch WASM-Engine
 - Mehrere Programmiersprachen für eine Software verknüpfbar



Quelle: Bytecode Alliance

Members



Gefahren von WASM

- WASM Format erschwert Lesbarkeit von Code
 - → Gefahrenanalyse erschwert
- Studie 2019 (DOI: 10.1007/978-3-030-22038-9_2)
 - 55 % der Webseiten die WASM verwenden nutzten es für Crypto-Mining

- **WASM wird JS erst mal nicht ablösen**
- Aktuell primär verwendet für:
 - Experimente
 - Bestehenden Code ins Web zu bringen
 - „Ich möchte meine Website lieber in Rust als JS entwickeln“
- WASM steckt noch in den Kinderschuhen, neue Funktionen, bessere Compiler und besseres Tooling bieten hohes Potenzial

Vielen Dank für die Aufmerksamkeit



Philipp Hartenfeller
Philipp.hartenfeller@mt-ag.com
@phartenfeller
<https://hartenfeller.dev/blog>

Unsere Vorträge auf der DOAG Konferenz + Ausstellung 2021

Betriebsstabilisierung von Datenbanken

Ernst Leber, Principal
DB-Support
Dienstag, 12:00-12:40 Uhr
Raum: Tokio

Qualitätssicherung in Datenbankprojekten automatisieren

Oliver Lemm, Fachbereichsleiter
APEX Development II
Donnerstag, 15:00-15:40 Uhr
Raum: Tokio

APEX mit Excel zur Datenerhebung – eine bewährte Lösung im Praxiseinsatz

Timo Herwix, Berater
APEX Development
Mittwoch, 08:00-08:40 Uhr
Raum: Kiew

Wird JavaScript abgelöst? Einblicke in WebAssembly

Philipp Hartenfeller, Berater
APEX Development
Mittwoch, 16:00-16:40 Uhr
Raum: Seoul

Oracle ClusterWare Upgrade von 12c nach 19c

Amin Farvardin, Senior Berater
DB-Support
Donnerstag, 16:00-16:40 Uhr
Raum: St. Petersburg