Generic JavaScript-to-WebAssembly wrapper

Všeobecná obalová funkcia pre WebAssembly funkcie volané z JavaScriptu

Meno študenta: Eva Herencsárová
Vedúci: RNDr. Richard Ostertág, PhD.
Konzultant: Andreas Haas

Vznik práce

Spolupráca s WebAssembly Runtime Team, Google

Úvod, motivácia, cieľ

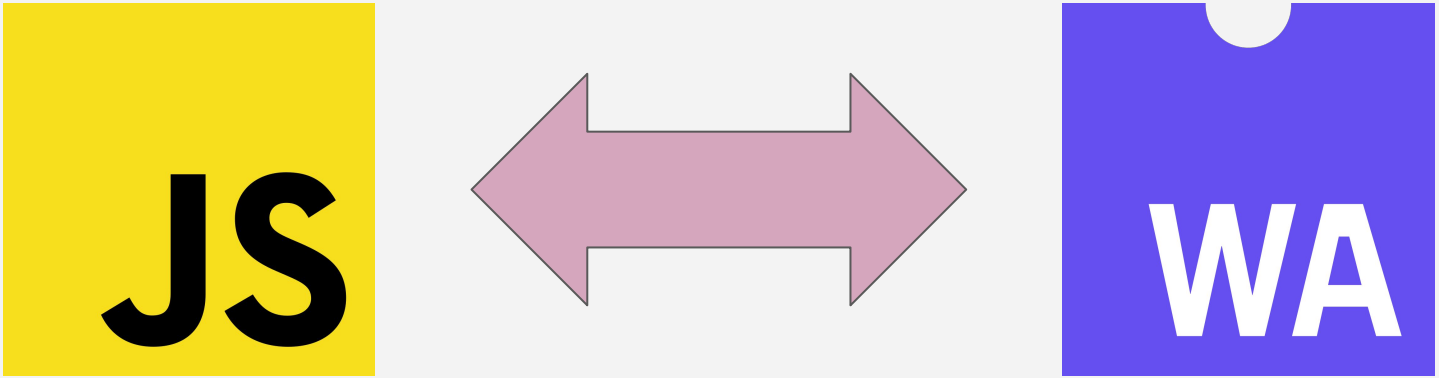
Úvod

- JavaScript:
  - skriptovací jazyk na tvorbu interaktívnych webstránok
- WebAssembly:
  - nový low-level programovací jazyk
  - formát pre lepší výkon
  - formát pre výsledok kompilácie high-level jazykov

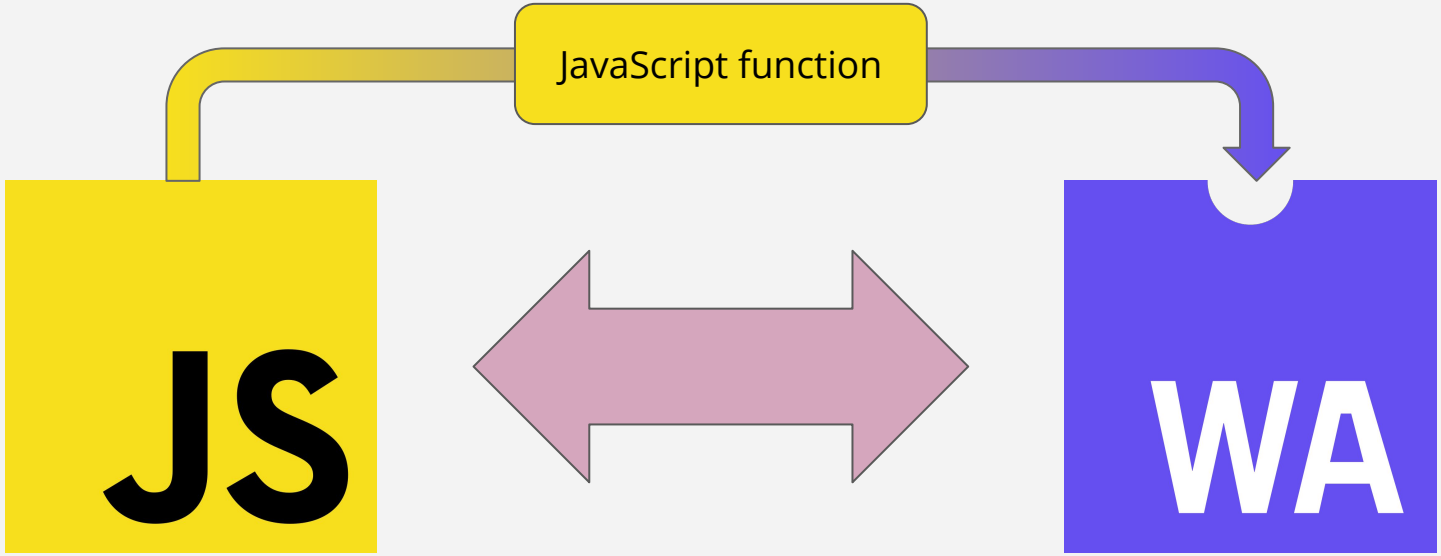
Úvod

WebAssembly ako doplnok k JavaScriptu

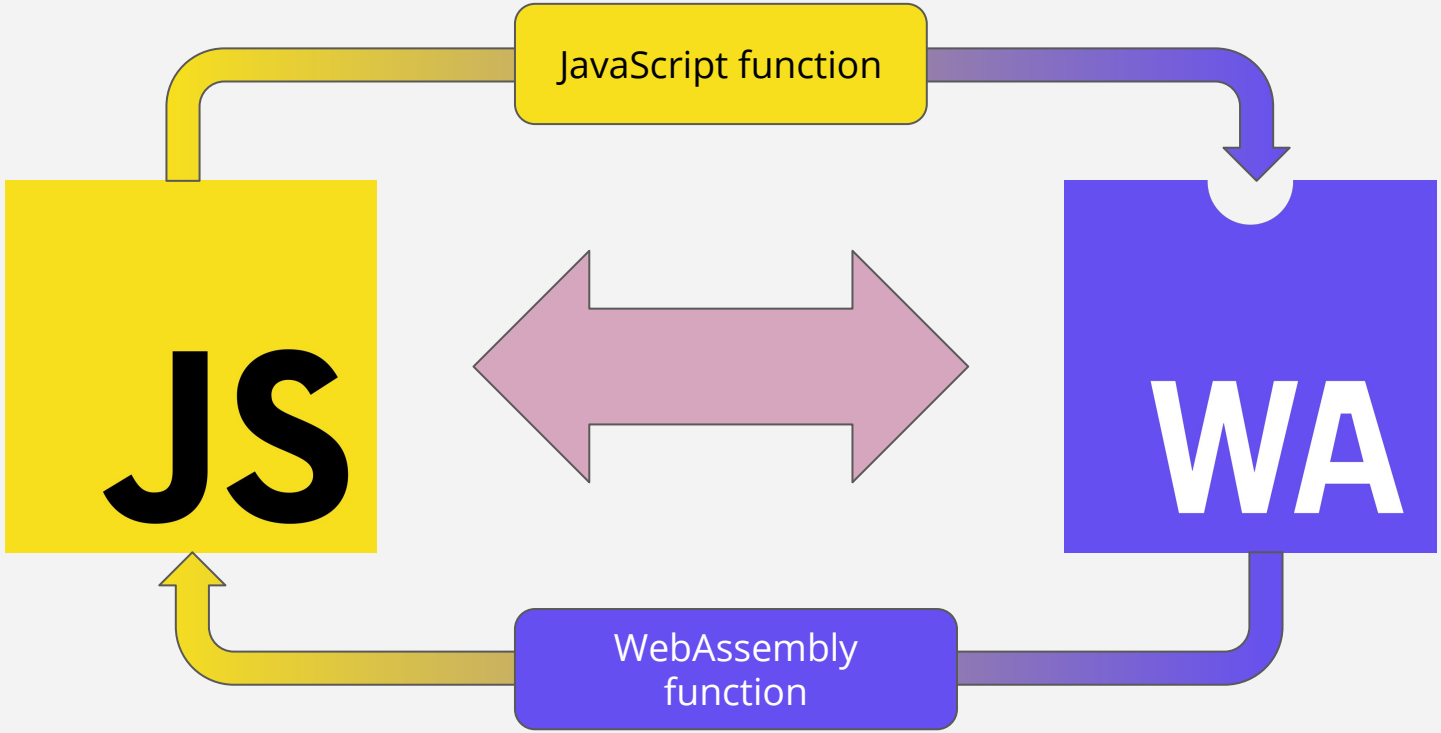
WebAssembly JavaScript API



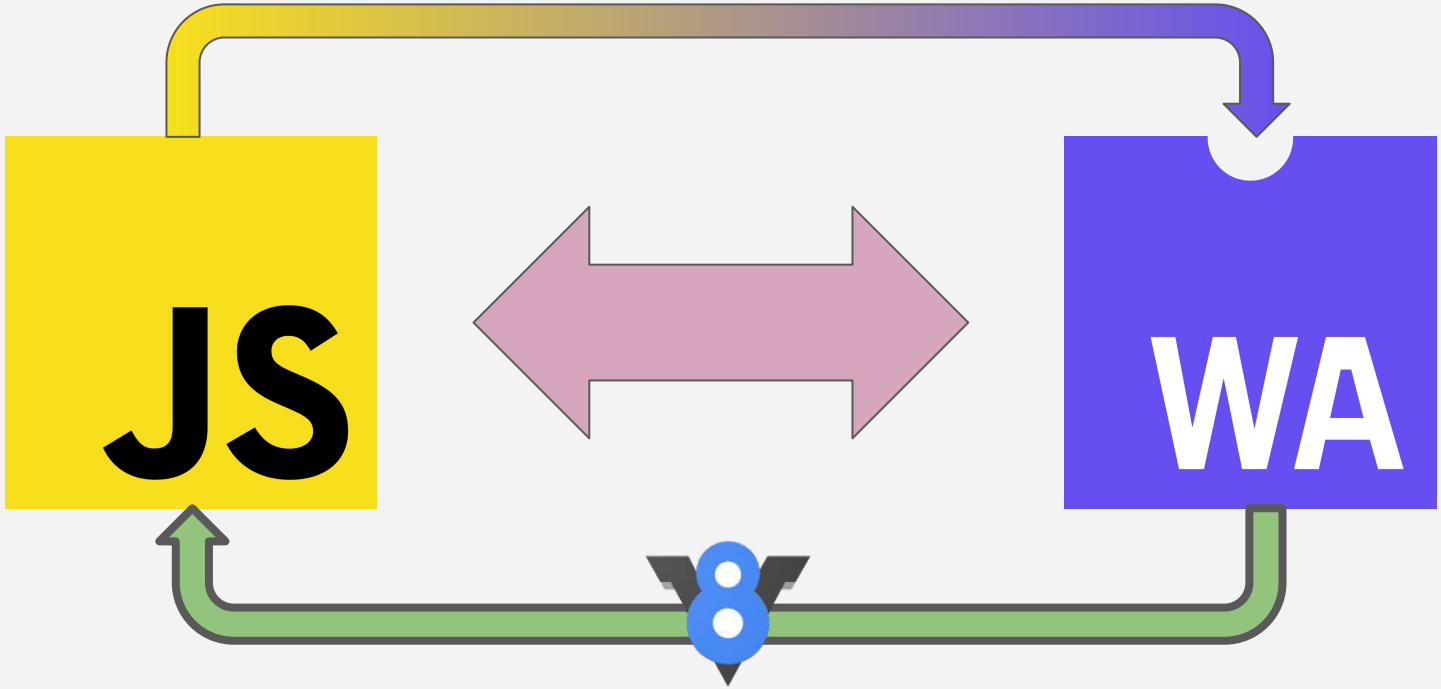
WebAssembly JavaScript API



WebAssembly JavaScript API



WebAssembly JavaScript API



V8

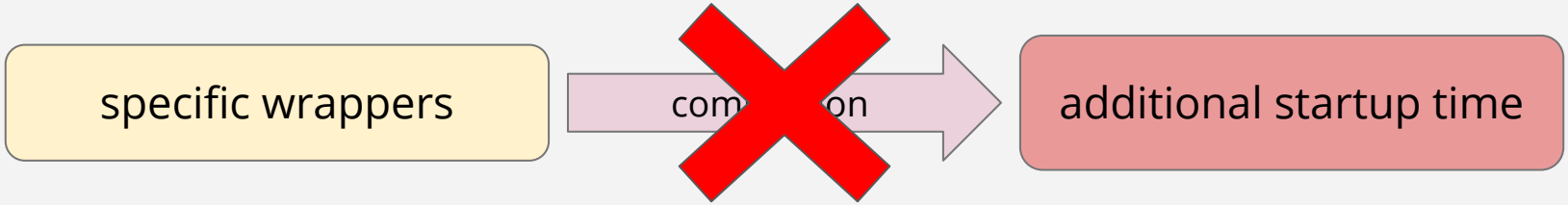
Interpreter/kompilátor pre WebAssembly a JavaScript
(napr. v Google Chrome)

V8

V8 skompiluje pre každú rôznu WebAssembly funkciu volanú z JavaScriptu špecifickú obalovú funkciu

Cieľ

Vyhnúť sa predkompilácii špecifických obalových funkcií



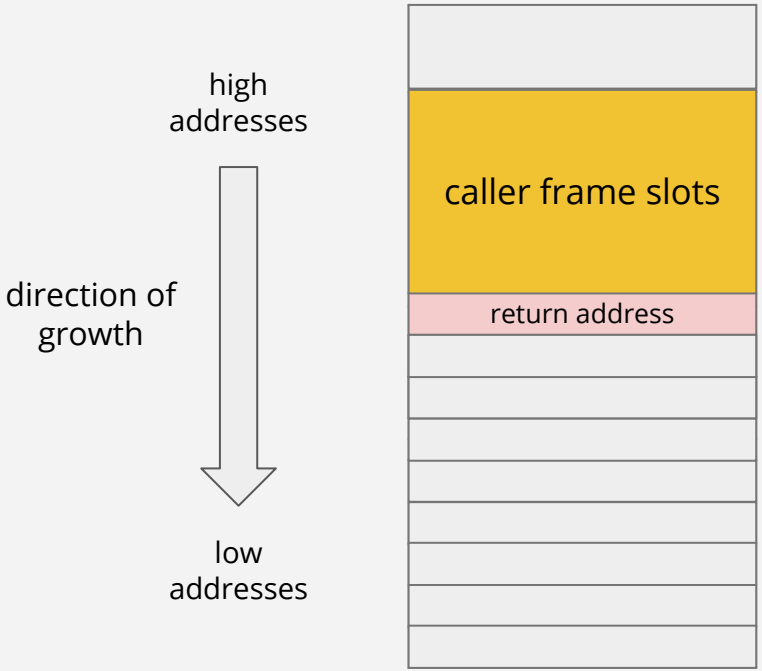
Cieľ

Navrhnúť a implementovať všeobecnú obalovú funkciu, ktorú by sme mohli použiť pre ľubovoľnú WebAssembly funkciu volanú z JavaScriptu

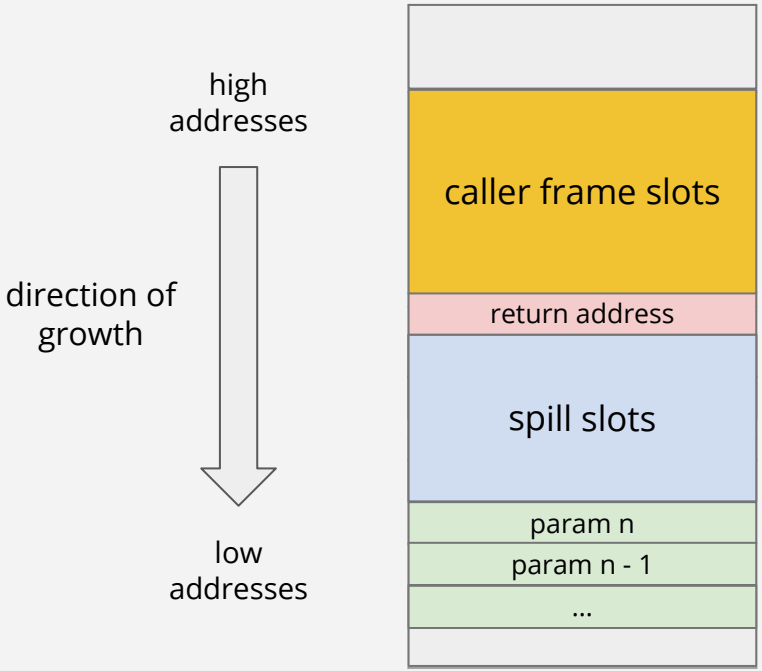
Creating one generic wrapper

Postup

Volanie WebAssembly funkcie z JavaScriptu



Volanie WebAssembly funkcie z JavaScriptu



Parameter registers

param 1

param 2

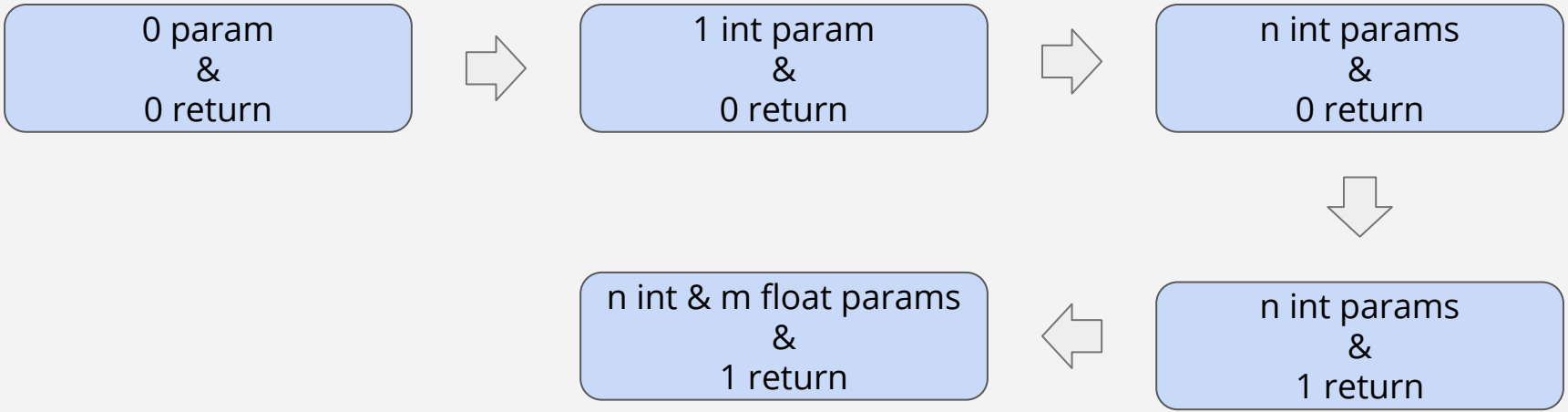
...

Return register

return val.

Všeobecná wrapper funkcia

- Jazyk: V8-špecifický macro assembler
- Postup: postupné rozširovanie množiny funkcií, pre ktorý sa dá použiť:

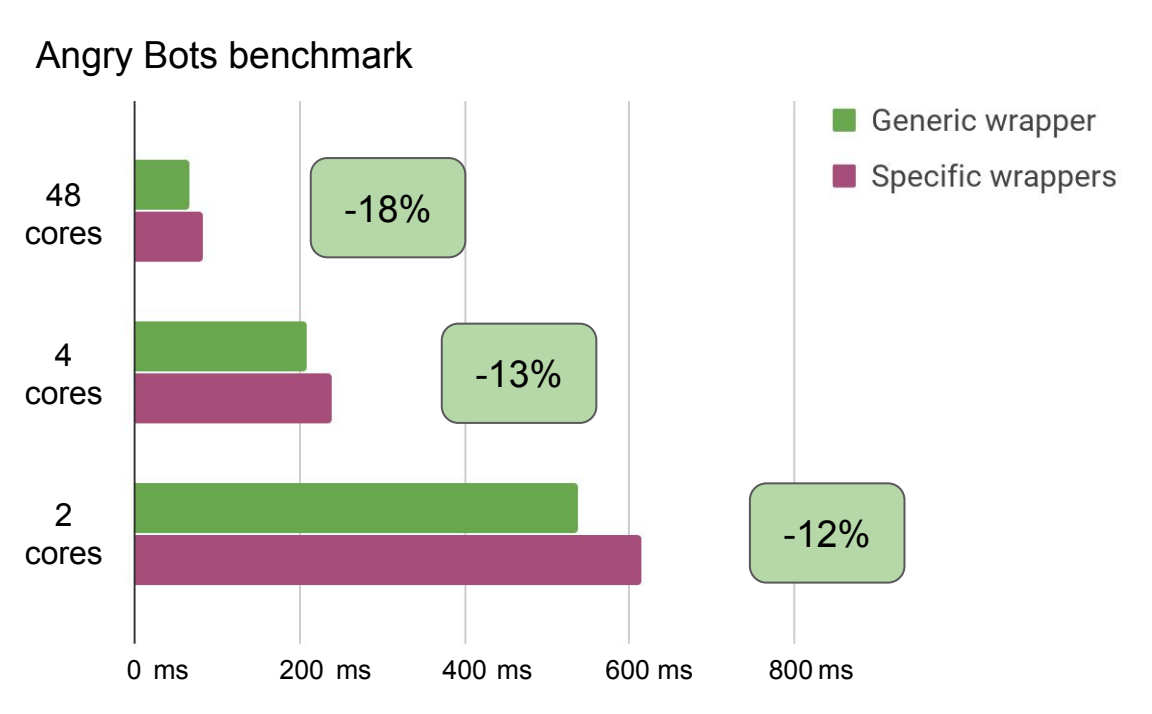


Všeobecná wrapper funkcia

- Získať informáciu o počte a typoch parametrov a return hodnôt
- Podporovať garbage collection
- Používať správne:
  - registre a časti registrov
  - inštrukcie

Výsledky

Doba kompilácie



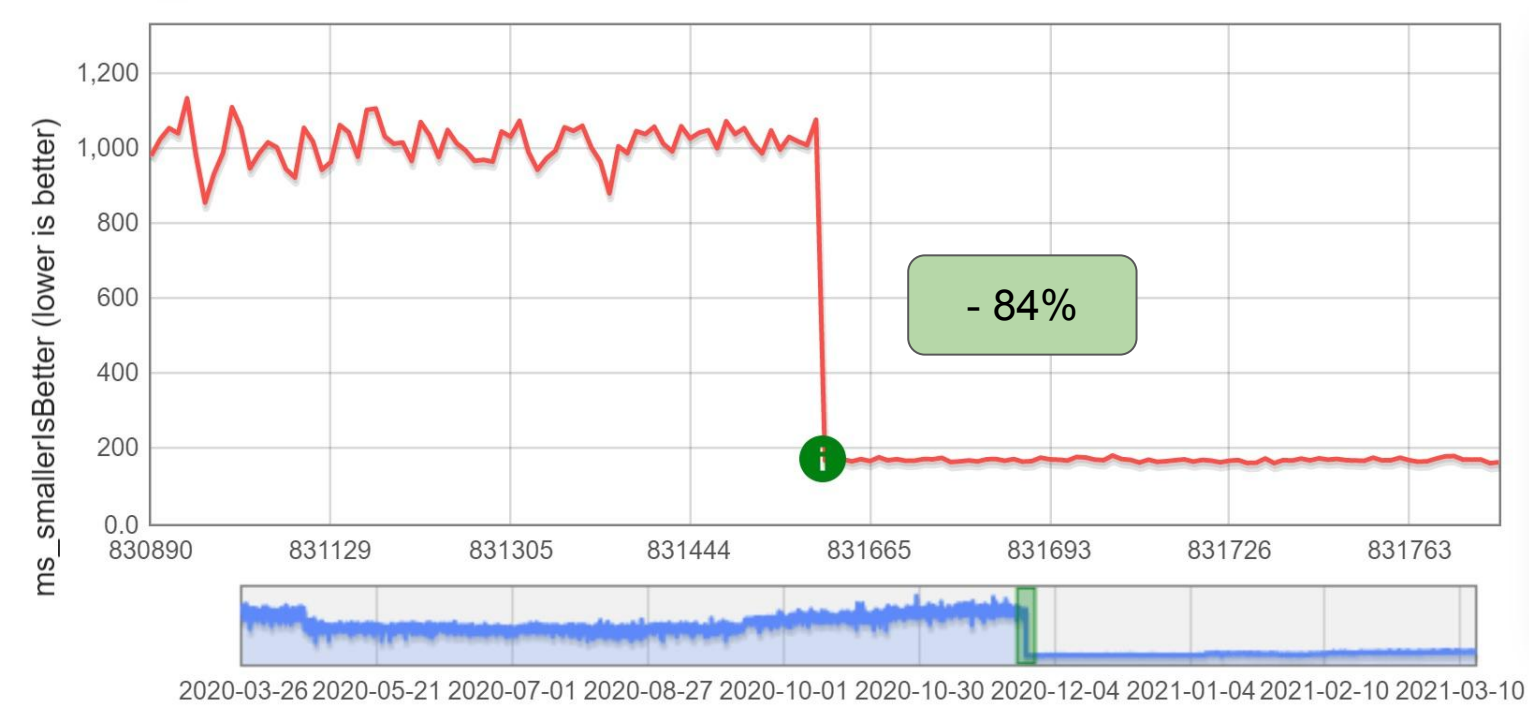
Runtime


Smi i32 parameters
Non-Smi i32 parameters
1 param
2 params
4 params
10 params
0
1000
2000
3000
0
2000
4000
6000
8000
Generic wrapper
Specific wrappers
Runtime
+25%
f32 parameters
f64 parameters
1 param
2 params
4 params
10 params
0
2000
4000
6000
8000
Generic wrapper
Specific wrappers

Následná práca - WebAssembly Runtime Team

- Použiť špecifické wrapper funkcie pre často volané funkcie
- Používať náš wrapper defaultne

Doba deserializácie pri Google Earth



Možné rozšírenia práce

- Implementácia aj na iných platformách
- Rozšíriť všeobecný wrapper aj pre neštandardné typy WebAssembly funkcií

Ďakujem za pozornosť!

Otázky oponenta

Môžete krátko zhrnúť budúcu prácu? Ktoré veci už boli implementované medzitým odkedy ste prácu zavŕšili, a čo stále zostáva otvorené?

Otázky oponenta

Môžete krátko zhrnúť budúcu prácu? Ktoré veci už boli implementované medzitým odkedy ste prácu zavŕšili, a čo stále zostáva otvorené?

Hotovo:

- Defaultne používané namiesto špecifických wrapperov
- Dynamické prepnutie k špecifickému wrapperu

Otvorené:

- Implementácia aj na iných platformách: riscv64, ppc, arm, arm64, ia32, s390, mips, mips64
- Rozšíriť všeobecný wrapper aj pre neštandardné typy WebAssembly funkcií: multi returns, JavaScript reference types

Otázky oponenta

Čo by si vyžadoval port na ďalšie architektúry okrem x64? Koľko z návrhu a koľko z implementácie sa bude dať zrecyklovať? Väčšina ostatných V8 built-inov podporuje všetky architektúry, takže vážne problémy asi nehrozia. Bol by tento port rovnako ľahký, alebo je na ňom niečo špeciálne, čo ho viac viaže ku konkrétnej architektúre?

Otázky oponenta

Čo by si vyžadoval port na ďalšie architektúry okrem x64? Koľko z návrhu a koľko z implementácie sa bude dať zrecyklovať? Väčšina ostatných V8 built-inov podporuje všetky architektúry, takže vážne problémy asi nehrozia. Bol by tento port rovnako ľahký, alebo je na ňom niečo špeciálne, čo ho viac viaže ku konkrétnej architektúre?

- Iný počet registrov
- Iné všeobecné/dátové a špeciálne registre

Otázky oponenta

Generický wrapper vyhráva v štartovacom čase a špecifický wrapper v cene za jedno volanie. V kapitole 4.2 sa spomína tier-up stratégia, ktorá medzi nimi dynamicky prepína. Aj keď už nie je súčasťou práce, zaujímalo by ma, ako zhruba funguje, a prípadne, či sú od súčasnej tier-up implementácie možné ďalšie vylepšenia.

Otázky oponenta

Generický wrapper vyhráva v štartovacom čase a špecifický wrapper v cene za jedno volanie. V kapitole 4.2 sa spomína tier-up stratégia, ktorá medzi nimi dynamicky prepína. Aj keď už nie je súčasťou práce, zaujímalo by ma, ako zhruba funguje, a prípadne, či sú od súčasnej tier-up implementácie možné ďalšie vylepšenia.

- Jednoduchá tier-up stratégia:
  - ak počet volaní funkcie prekročí nejakú prahovú hodnotu, tak sa skompiluje pre ňu špecifický wrapper, ktorý nahradí všeobecný wrapper pri tomto typu signatúry

Otázky oponenta

Ako to, že v štartovacom čase pre Google Earth nastalo také neuveriteľné zrýchlenie (o 84%)? V bežnej aplikácii by som čakal, že iba malé percento všetkých WebAssembly funkcií bude exportované, a aj pre ne by wrapper mal byť len zlomok oproti samotnému telu funkcie. V kapitole 4.2 sa píše, že špecifické wrappery treba kompilovať pre každý thread zvlášť - je toto zodpovedné za celý efekt? A prečo to vlastne treba - ak sú výstupom v oboch prípadoch len podgrafy pre TurboFan, prečo sa iba tieto nedajú zdieľať?

(Táto otázka je značne nad rámec práce. Privítam aj špekuláciu.)

Zdroje

Chrome Performance Dashboard

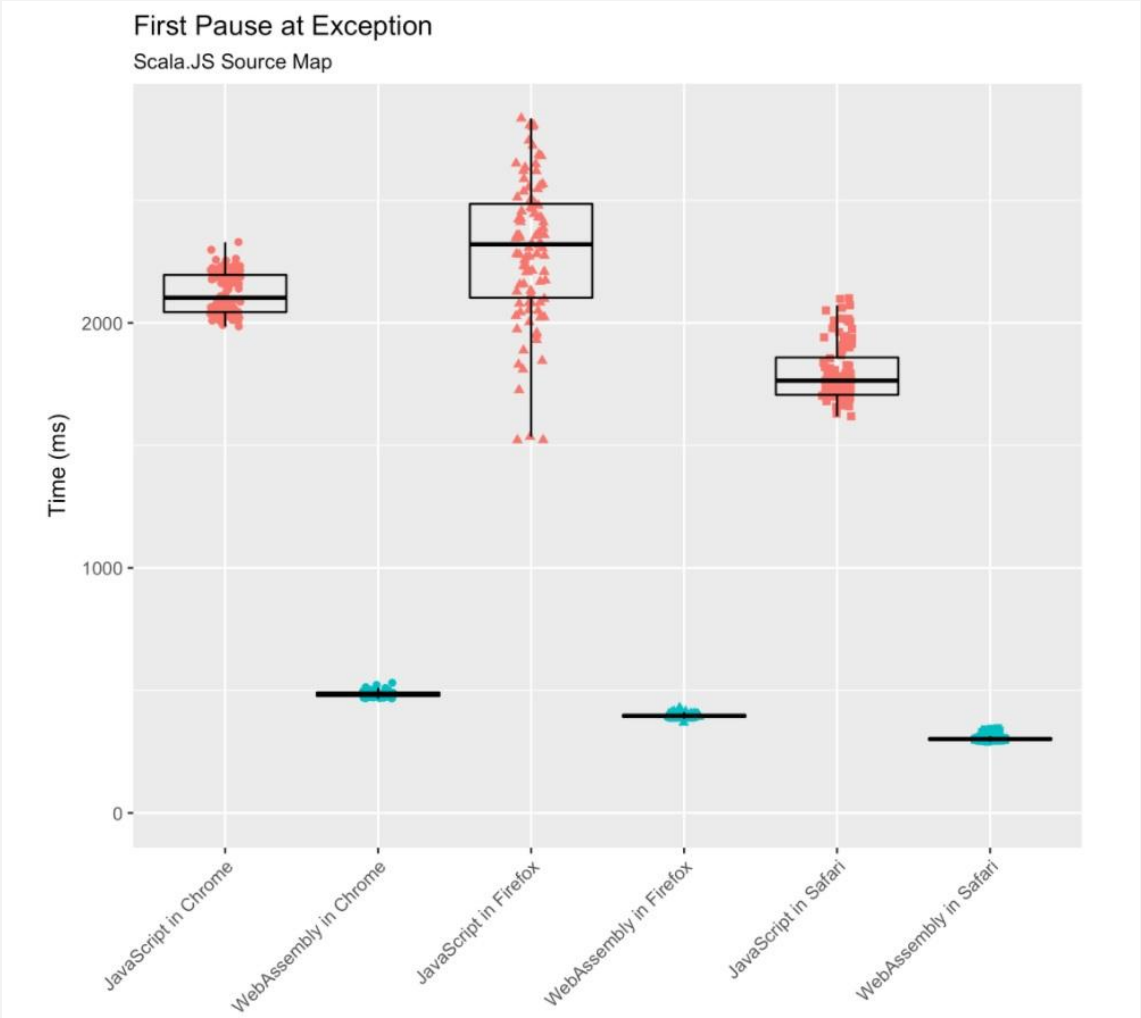
File:WebAssembly Logo.svg

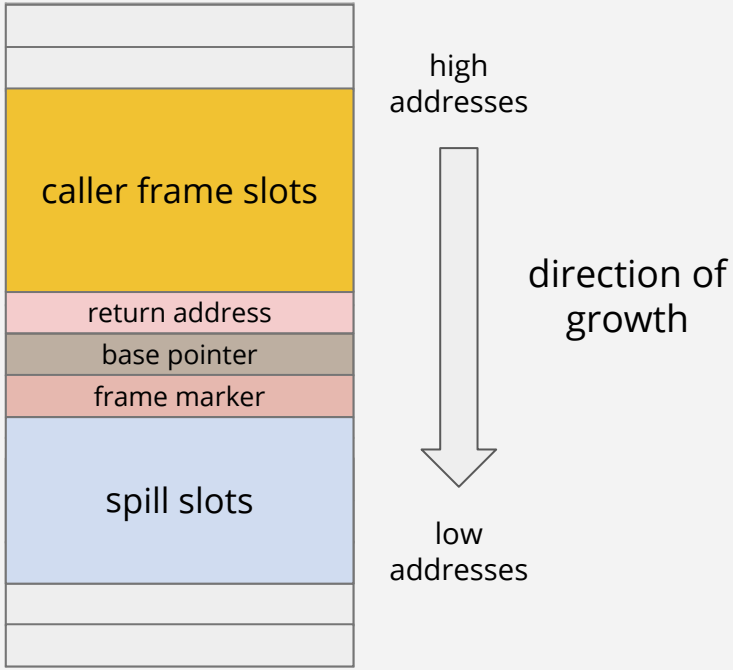
V8 logos · V8

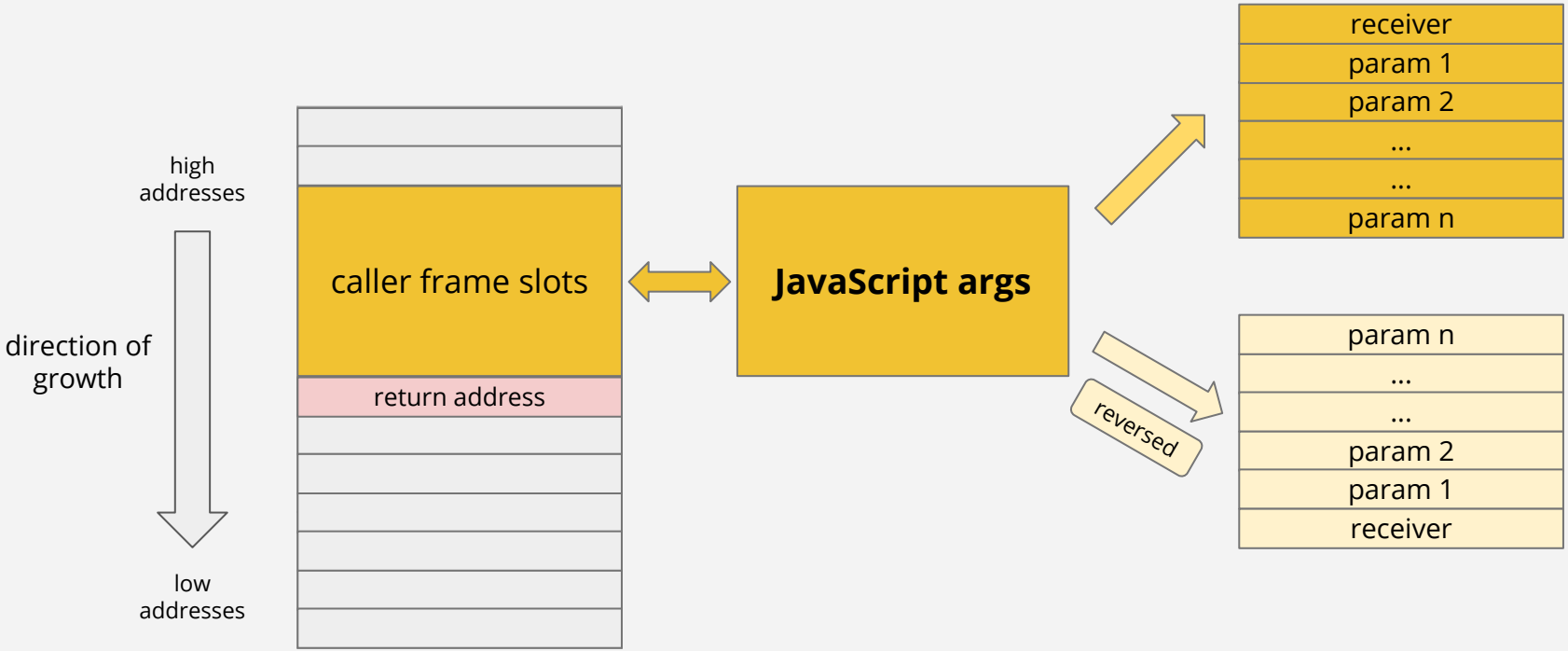
File:Unofficial JavaScript logo 2.svg

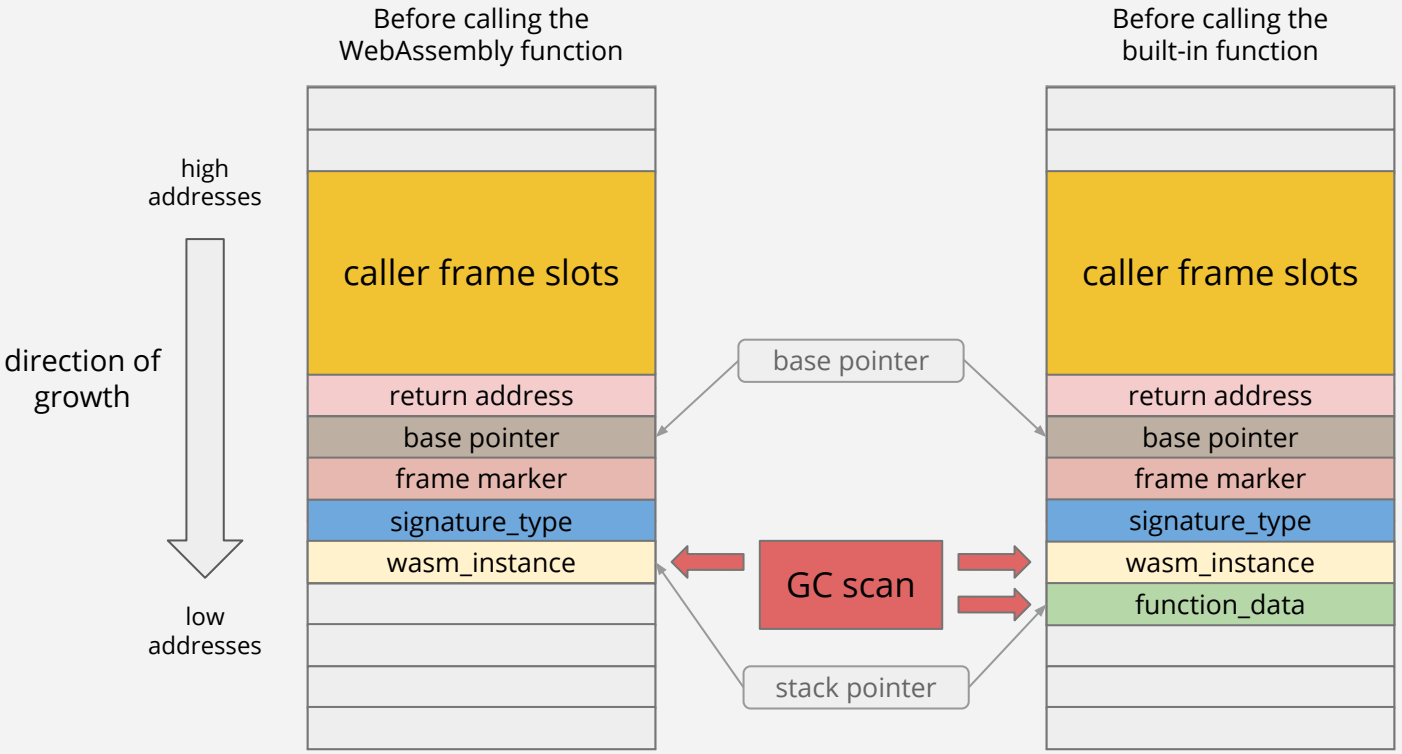
Runer silhouette running fast - Free sports

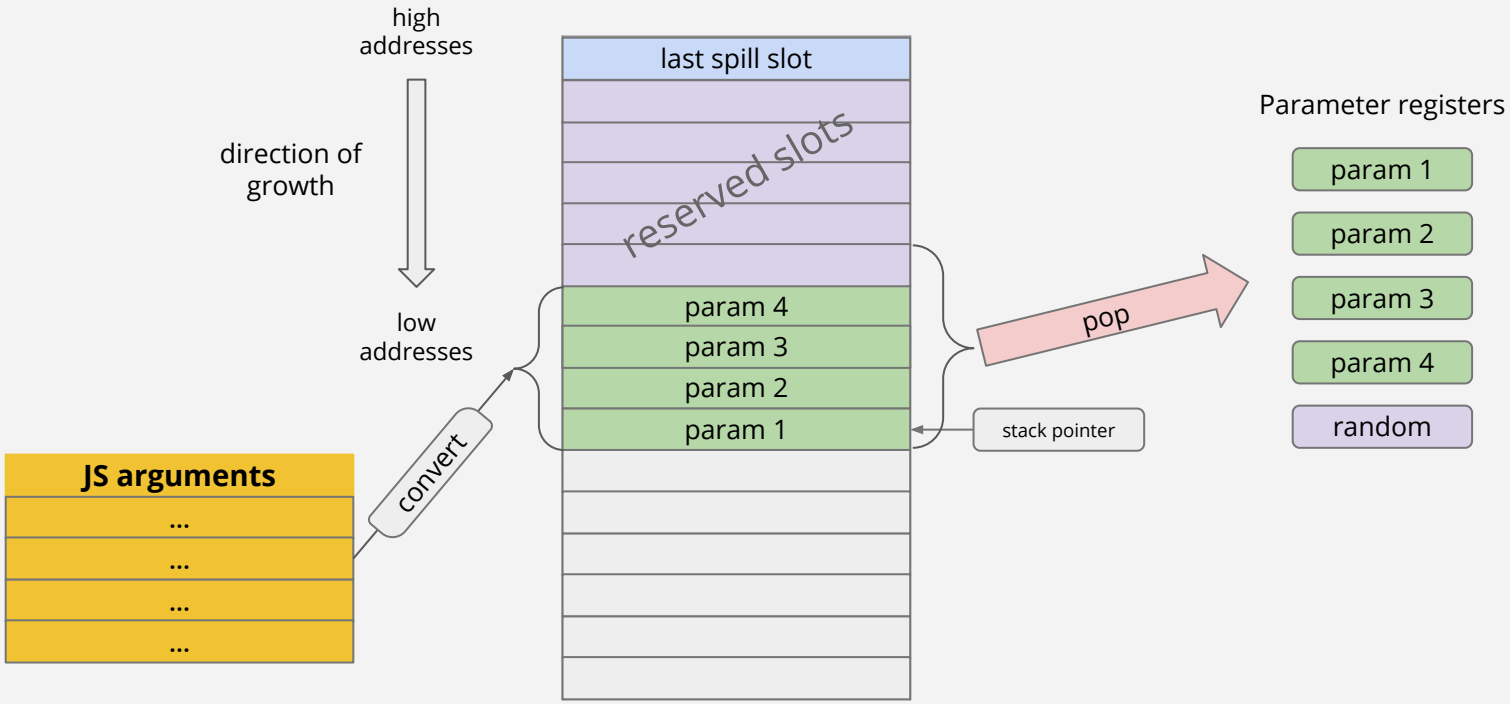
Dodatok

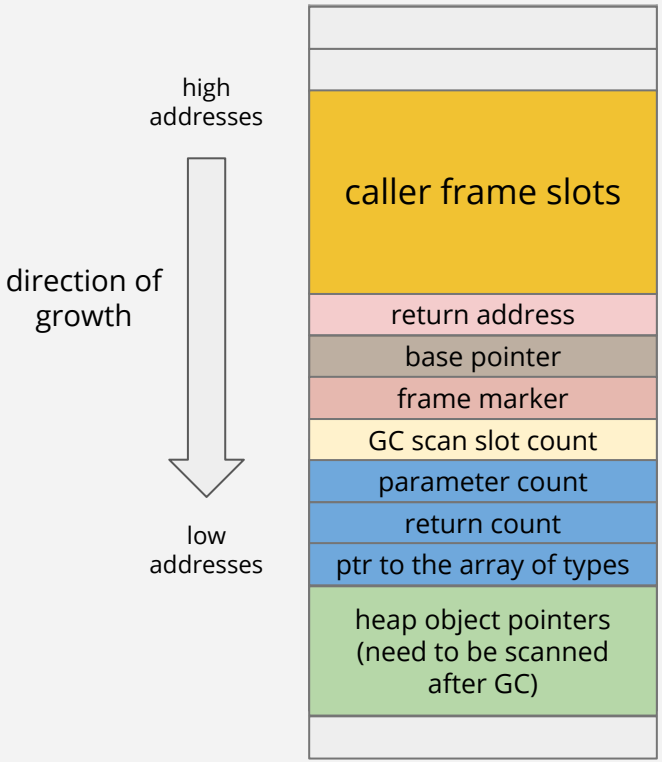

First Pause at Exception
Scala.JS Source Map
Time (ms)
2000
1000
0
JavaScript in Chrome
WebAssembly in Chrome
JavaScript in Firefox
WebAssembly in Firefox
JavaScript in Safari
WebAssembly in Safari

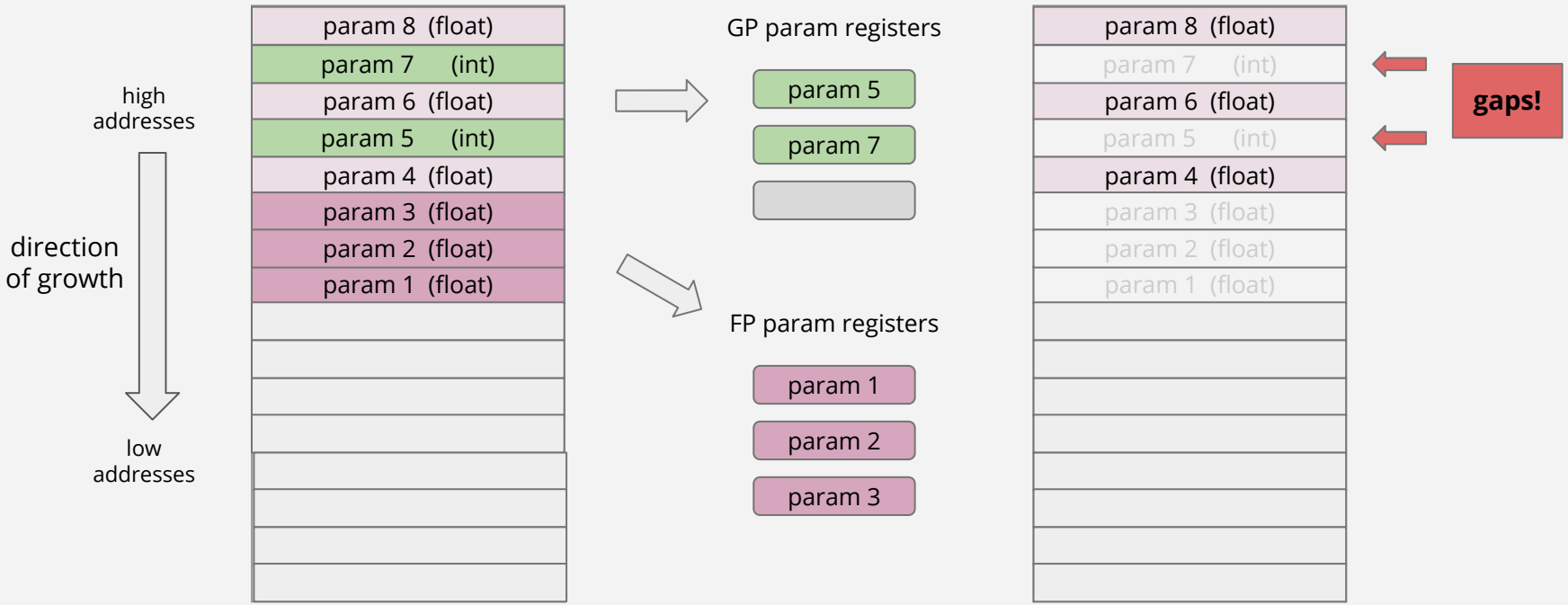

caller frame slots
return address
base pointer
frame marker
spill slots
high addresses
low addresses
direction of growth

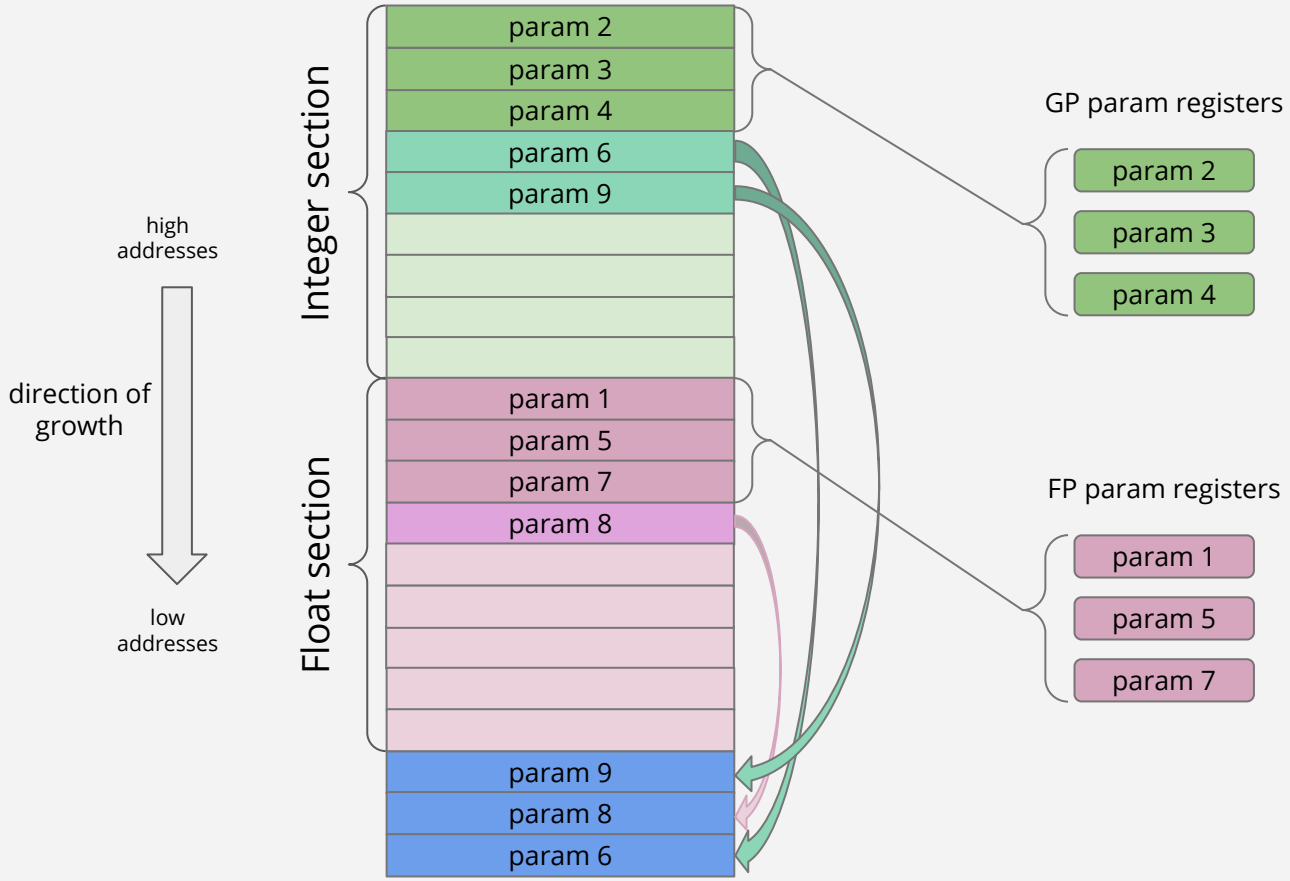

high addresses
direction of growth
low addresses
caller frame slots
return address
JavaScript args
receiver
param 1
param 2
...
...
param n
reversed
param n
...
...
param 2
param 1
receiver


Before calling the WebAssembly function
Before calling the built-in function
high addresses
direction of growth
low addresses
caller frame slots
return address
base pointer
frame marker
signature_type
wasm_instance
caller frame slots
return address
base pointer
frame marker
signature_type
wasm_instance
function_data
base pointer
GC scan
stack pointer


high addresses
direction of growth
low addresses
last spill slot
reserved slots
param 4
param 3
param 2
param 1
JS arguments
...
...
...
...
convert
pop
stack pointer
Parameter registers
param 1
param 2
param 3
param 4
random


high addresses
direction of growth
low addresses
caller frame slots
return address
base pointer
frame marker
GC scan slot count
parameter count
return count
ptr to the array of types
heap object pointers (need to be scanned after GC)


high addresses
direction of growth
low addresses
param 8 (float)
param 7 (int)
param 6 (float)
param 5 (int)
param 4 (float)
param 3 (float)
param 2 (float)
param 1 (float)
GP param registers
param 5
param 7
FP param registers
param 1
param 2
param 3
param 8 (float)
param 7 (int)
param 6 (float)
param 5 (int)
param 4 (float)
param 3 (float)
param 2 (float)
param 1 (float)
gaps!


high addresses
direction of growth
low addresses
Integer section
param 2
param 3
param 4
param 6
param 9
Float section
param 1
param 5
param 7
param 8
param 9
param 8
param 6
GP param registers
param 2
param 3
param 4
FP param registers
param 1
param 5
param 7