

**Тяжело в учении,
сложно при внедрении,
быстро в использовании**

Павел Дадыкин

ROGII



**Frontend
Conf 2024**

Обо мне

Павел Дадыкин

- 9 лет во фронтенде
- 2 года тимлид на проекте StarLite Web
- Первый раз на FrontendConf



О чём доклад

- История внедрения WebAssembly на реальном проекте
- Подводные камни и решение проблем
- Стоит ли вам использовать WebAssembly в своих проектах?



- Разработка продуктов для нефтегазовой индустрии
- Geoscience-решения
- Много математических расчётов
- Desktop, Web, iOS, Android, SDK, Public API

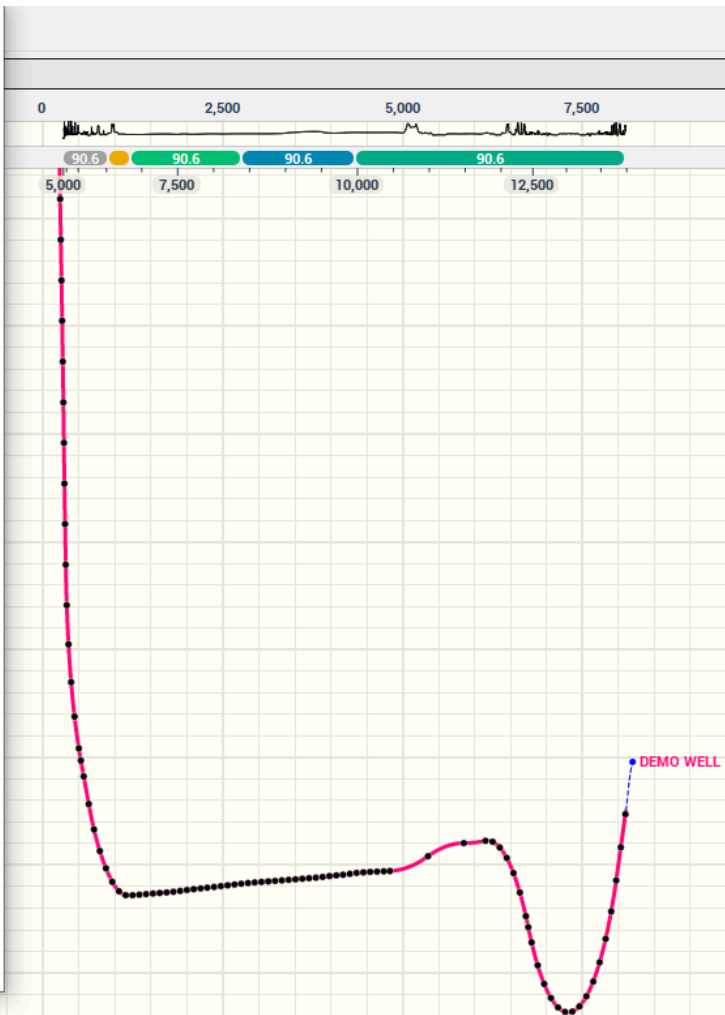
Well Trajectory - Demo Well

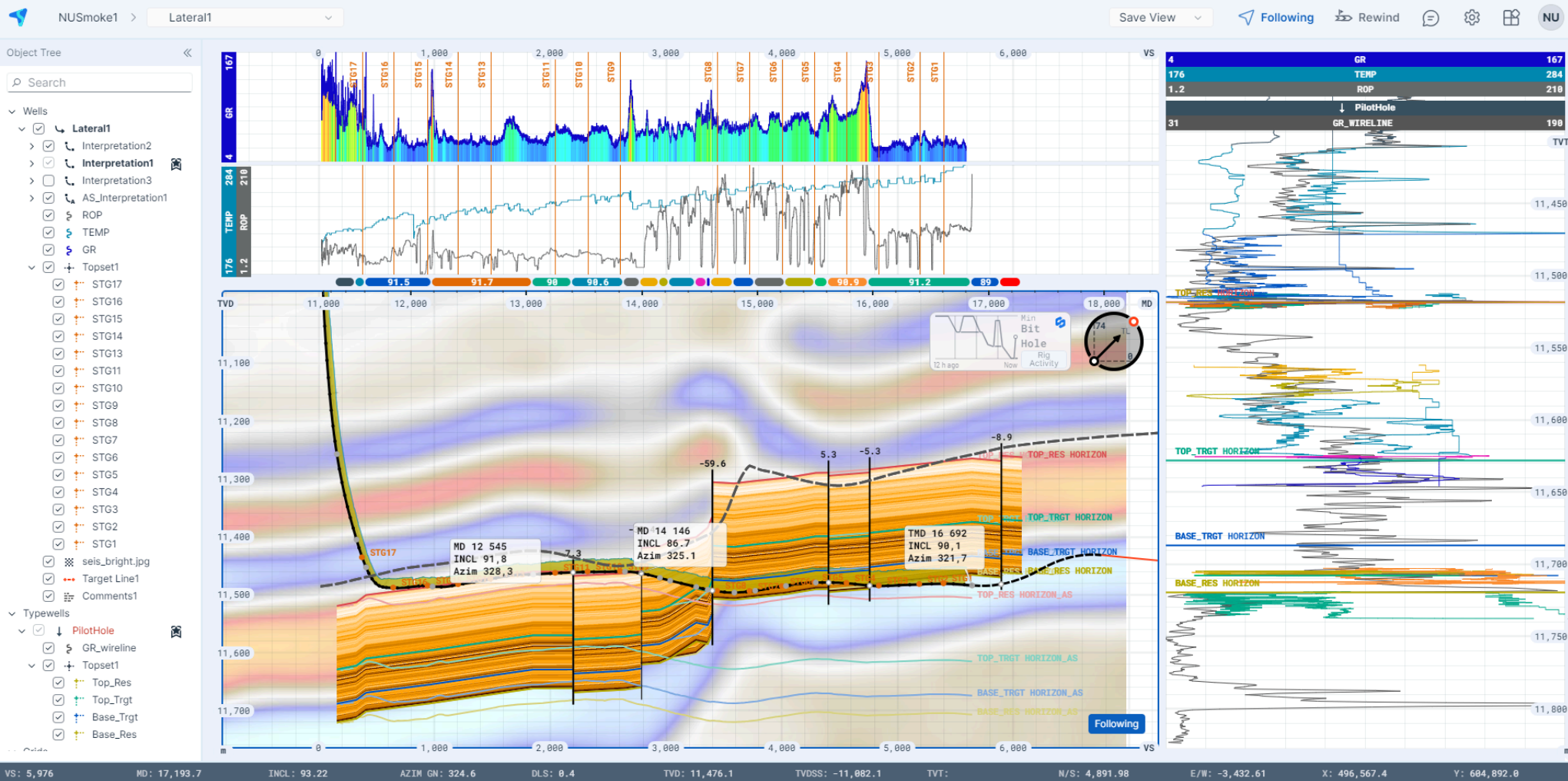
Well name: Demo Well Xsrf: 1,053,408.5 Ysrf: 772,770.4 KB Elevation: 4,799 Azimuth VS (GN): 342.00°

API: Operator: Grid Convergence: 90.00°

	MD	INCL	AZIM TN	AZIM GN	TVD	N/S (GN)	E/W (GN)	X (GN)	Y (GN)	TVDSS	VS	DLS
tie-in	0	0	0	270	0	0	0	1,053,408.5	772,770.4	4,799	0	0
1	26	0	0	270	26	0	0	1,053,408.5	772,770.4	4,773	0	0
2	147	0.43	307.8	217.8	147	-0.4	-0.3	1,053,408.2	772,770.1	4,652	-0.3	0.4
3	241	0.4	330.4	240.4	241	-0.8	-0.8	1,053,407.7	772,769.6	4,558	-0.5	0.2
4	335	0.38	357.5	267.5	335	-1	-1.4	1,053,407.1	772,769.5	4,464	-0.5	0.2
5	429	0.26	33.2	303.2	429	-0.9	-1.9	1,053,406.6	772,769.6	4,370	-0.3	0.2
6	523	0.81	91	1	523	-0.1	-2	1,053,406.5	772,770.3	4,276	0.5	0.8
7	618	0.92	87.3	357.3	618	1.3	-2.1	1,053,406.5	772,771.8	4,181	1.9	0.1
8	712	1.18	112	22	712	3	-1.7	1,053,406.8	772,773.4	4,087	3.4	0.5
9	807	1.2	121.9	31.8	806.9	4.7	-0.8	1,053,407.7	772,775.2	3,992.1	4.8	0.2
10	902	1.34	129.1	39.1	901.9	6.5	0.4	1,053,408.9	772,776.9	3,897.1	6	0.2
11	996	1.47	123	33	995.9	8.3	1.7	1,053,410.3	772,778.8	3,803.1	7.4	0.2
12	1,091	1.44	127.4	37.4	1,090.9	10.3	3.1	1,053,411.6	772,780.7	3,708.1	8.8	0.1
13	1,186	0.57	136.7	46.7	1,185.8	11.6	4.2	1,053,412.7	772,782	3,613.2	9.7	0.9
14	1,280	0.25	337.2	247.2	1,279.8	11.8	4.3	1,053,412.9	772,782.2	3,519.2	9.9	0.9
15	1,375	0.37	320.3	230.3	1,374.8	11.5	3.9	1,053,412.4	772,782	3,424.2	9.7	0.2
16	1,470	0.22	294.9	204.9	1,469.8	11.2	3.6	1,053,412.1	772,781.6	3,329.2	9.5	0.2
17	1,564	0.16	186.9	96.9	1,563.8	11	3.7	1,053,412.2	772,781.4	3,235.2	9.3	0.3
18	1,659	0.19	162.5	72.5	1,658.8	11	3.9	1,053,412.5	772,781.5	3,140.2	9.3	0.1
19	1,753	0.34	155.9	65.9	1,752.8	11.2	4.3	1,053,412.9	772,781.6	3,046.2	9.3	0.2
20	1,847	0.17	105.2	15.2	1,846.8	11.4	4.6	1,053,413.2	772,781.9	2,952.2	9.4	0.3
21	1,942	0.3	115.6	25.6	1,941.8	11.8	4.8	1,053,413.3	772,782.2	2,857.2	9.7	0.1
22	2,036	0.15	124.2	34.2	2,035.8	12.1	5	1,053,413.5	772,782.5	2,763.2	10	0.2
23	2,131	0.11	147.5	57.5	2,130.8	12.3	5.1	1,053,413.6	772,782.7	2,668.2	10.1	0.1
24	2,226	0.2	112.9	22.9	2,225.8	12.5	5.2	1,053,413.8	772,782.9	2,573.2	10.2	0.1
25	2,239	0.28	112.7	22.7	2,238.8	12.5	5.3	1,053,413.8	772,782.9	2,560.2	10.3	0.6

OK





Эволюция продуктов ROGII

1. Расчёты написаны для Desktop App (на C++)
2. Хранение данных в Cloud
3. Повторение расчётов в Web, iOS, Android

Проблемы

- Дублирование расчётов на разных платформах
- Низкая производительность при вычислениях
- Сложности добавления/изменения расчётов

Цель

- Внедрить в приложение расчёты, написанные на C++
- Сохранить при этом текущий стек технологий:
 - React
 - **Typescript**
 - **Webpack**
 - **Web workers**
 - Canvas
 - WebGL

WebAssembly (WASM)

- Формат байт-кода, исполняемого современными браузерами
- Может взаимодействовать с JavaScript



Плюсы WASM

- Обеспечивает высокую скорость исполнения
- Поддержка всеми браузерами
- Компиляция в WASM доступна для множества языков
(C, **C++**, C#, Rust, Elixir, Erlang, Go, TypeScript, D, Kotlin)

Минусы WASM

- Чёрный ящик
- Внешняя типизация
- Конвертация данных $JS \rightarrow WASM \rightarrow JS$

Доклады про WASM

- [Разработка под WebAssembly: реальные грабли и примеры](#)
[\(Андрей Нагих, FC 2018\)](#)
- [RUST + WEBASSEMBLY \(Илья Барышников, FC 2019\)](#)
- [WebAssembly SPA-фреймворки \(FC 2020\)](#)

Тяжело в учении

Примеры использования WASM в сети (конец 2021 года):

- A + B
- Factorial
- Fibonacci



Как же скомпилировать C++ в WASM?

Компиляция в WASM

- [Emscripten](#)
- [Binaryen](#)
- [Компиляция C в WebAssembly без Emscripten](#)

Emscripten

- Компиляция из C/C++/LLVM-language в WebAssembly
- Запуск этого кода в Web/Node.js
- Поддержка библиотек C/C++



Документация Emscripten

The screenshot shows the GitHub web interface for the file `emscripten / src / settings.js`. The left sidebar displays a file tree with `settings.js` selected. The main area shows the code with line numbers 2138 to 2177. The code contains various settings and comments related to debugging, library symbols, and memory management.

```
2138 // This setting is enabled by default if any of the following debugging settings
2139 // are enabled:
2140 // - PTHREADS_DEBUG
2141 // - DYLINK_DEBUG
2142 // - LIBRARY_DEBUG
2143 // - GL_DEBUG
2144 // - OPENAL_DEBUG
2145 // - EXCEPTION_DEBUG
2146 // - SYSCALL_DEBUG
2147 // - WEBSOCKET_DEBUG
2148 // - SOCKET_DEBUG
2149 // - FETCH_DEBUG
2150 // [link]
2151 var RUNTIME_DEBUG = false;
2152
2153 // Include JS library symbols that were previously part of the default runtime.
2154 // Without this, such symbols can be made available by adding them to
2155 // DEFAULT_LIBRARY_FUNCS_TO_INCLUDE, or via the dependencies of another JS
2156 // library symbol.
2157 var LEGACY_RUNTIME = false;
2158
2159 // User-defined functions to wrap with signature conversion, which take or return
2160 // pointer argument. Only affects MEMORY64=1 builds, see create_pointer_conversion_wrappers
2161 // in emscripten.py for details.
2162 // Use _ for non-pointer arguments, p for pointer/153 arguments, and P for optional pointer/153 values.
2163 // Example use -sSIGNATURE_CONVERSIONS=someFunction:_p,anotherFunction:p
2164 // [link]
2165 var SIGNATURE_CONVERSIONS = [];
2166
2167 //=====
2168 // Internal, used for testing only, from here
2169 //=====
2170
2171 // Internal (testing only): Disables the blitOffscreenFramebuffer VAO path.
2172 // [link]
2173 var OFFSCREEN_FRAMEBUFFER_FORBID_VAO_PATH = false;
2174
2175 // Internal (testing only): Forces memory growing to fail.
2176 // [link]
2177 var TEST_MEMORY_GROWTH_FAILS = false;
```

**Дефолтной конфигурации Emscripten
не существует!**

Флаги компиляции

01. `-s DYNAMIC_EXECUTION=0 -s ENVIRONMENT=worker`

02. `-s MODULARIZE=1 -s ALLOW_MEMORY_GROWTH=1 --bind`

- Выключаем использование eval
- **Задаем выполнение в веб-воркере**
- Оборачиваем в модульный код (без глобальных переменных)
- Разрешаем автоматический рост памяти
- Говорим привязать все функции к js

Где же взять описание функций?

Генерация документации

Doxygen

◆ calculateTHLProjectionFor3DPoint()

```
double calculateTHLProjectionFor3DPoint ( CGeomathCalculatedTrajectoryPointCollection calculatedTrajectory,  
                                          CGeomath3DPoint point,  
                                          CGeomathWellHead wellHead  
                                          )
```

calculates THL-projection for 3D point

Parameters

- [in] **calculatedTrajectory** Calculated points of the trajectory. Can be obtained by calling calculateTrajectory.
- [in] **point** Point which thl is needed.
- [in] **wellHead** Wellhead data of the lateral, in the projection of which calculations are performed.

Returns




Returns THL-projection.

Как сделать удобной отладку ошибок?

Разные сборки (через stake)




Debug

- 👎 Большой вес файла
- 👎 Работает медленнее
- 👍 Детальное описание ошибок

Name	Size
 package.json	182 bytes
 starsteermath_javascript.js	113.5 kB
 starsteermath_javascript.wasm	32.1 MB

Release

- 👍 Маленький вес файла
- 👍 Работает быстро
- 👎 Сложно отлаживать

Name	Size
 package.json	184 bytes
 starsteermath_javascript.js	37.8 kB
 starsteermath_javascript.wasm	387.7 kB

Утечки памяти

- Добавили дополнительный флаг в сборку
- Он передаёт флаг `-fsanitize=address -g2` компилятору
- Вызов функции для принудительного сбора статистики

Как реализовать версионирование?

Версионирование

Emscripten на выходе даёт 2 файла:

- Бинарный .wasm
- Обвязка в виде .js-файла



Версионирование

Добавить в начало JS-файла комментарий

01. /*

02. Version: 1.3.0

03. Commit: 681def4aefb99c9de5357617dc138b66e6ae1d0b

04. Pipeline: 103990

05. */

Версионирование

- Используем [Nexus](#)
- Собираем в npm-пакет
- Учитываем вариант сборки (Debug/Release)

01. "dependencies": {

02. "@nexus-npm/wasm": "0.9.0-Release",

03. }

Подключение WASM к проекту

Как настроить Webpack?

Webpack

[webassembly-loader](#)

[wasm-loader](#)

Repository

github.com/ballercat/wasm-loader

Homepage

github.com/ballercat/wasm-loader#rea...

Weekly Downloads

8,286



Version

1.3.0

License

MIT

Unpacked Size

24.3 kB

Total Files

11

Issues

0

Pull Requests

0

Last publish

6 years ago

Webpack 4 file-loader

```
01. {  
02.   test: /\.wasm$/,  
03.   type: "javascript/auto",  
04.   use: [{  
05.     loader: "file-loader",  
06.     options: { name: "wasm/[name].[hash].[ext]" }  
07.   }]  
08. }
```

Загрузка WASM в приложение

```
01. import wasm from 'math-js.wasm';  
02.  
03. const response = await client.get(wasm, {  
04.     responseType: 'arraybuffer'  
05. });  
06.  
07. const wasmBinary = response.data;
```

Webpack 5

Asset Modules на замену file-loader

syncWebAssembly / asyncWebAssembly

Как добавить типизацию?

Typescript

- [@types/emscripten](https://github.com/microsoft/typescript) для вспомогательных методов
- Описание структур и функций вручную, используя Doxygen
- Автоматическая генерация **.ts** из **.hpp**

CSP Error

“*Refused to compile or instantiate WebAssembly module because 'unsafe-eval' is not an allowed source of script in the following Content Security Policy directive: "script-src https:".*

Content-Security-Policy: script-src 'wasm-unsafe-eval'

Как конвертировать данные JS \leftrightarrow WASM?

JS

- Массивы объектов
- Автоматическое выделение памяти
- Автоматическая сборка мусора

C/C++

- `std::vector` / `TypedArray` /
Динамические массивы
- Требуется выделение памяти
- Нужно освобождать выделенную память

Конвертация JS ↔ WASM

```
01. const pointer = instance._malloc(size * Float64Array.BYTES_PER_ELEMENT);  
02.  
03. instance.HEAPF64.set(  
04.   new Float64Array(size),  
05.   pointer / Float64Array.BYTES_PER_ELEMENT,  
06. );  
07.  
08. instance._free(trajjectory.mdArray.pointer);
```

Добавление обёртки

- Добавили метаданные для функций конвертации и высвобождения памяти
- Emscripten сам конвертирует типы в C-структуры
- Передаём сразу массив объектов

Удобные функции

- 01. `convertToJSObject<WasmType, JSArrayType>(data)`
- 02. `convertFromJSObject<JSArrayType, WasmType>(data)`
- 03. `freeCollection(data)`

Undefined / NaN

В **TypeScript** мы можем написать

```
type X = number | undefined
```

В **C/C++** так нельзя!

Из функций расчёта в этом случае нам возвращается **NaN**

Undefined → NaN

```
01. // Преобразование для передачи в модуль расчёта
02. point.data ?? NaN
03.
04. // Проверка на наличие значения
05. if (!isNaN(point.data))
```

Batch запросов

- Большая часть времени тратится на преобразования из структур данных JS в WASM и обратно
- Лучше выполнить одну большую операцию, чем много маленьких

Batch запросов

```
01. for (const element of elements) {
```

```
02.   const point = calculatePoint(element.x);
```

```
03.   points.push(point);
```

```
04. }
```

```
01. const xArray = elements.map((element) => element.x);
```

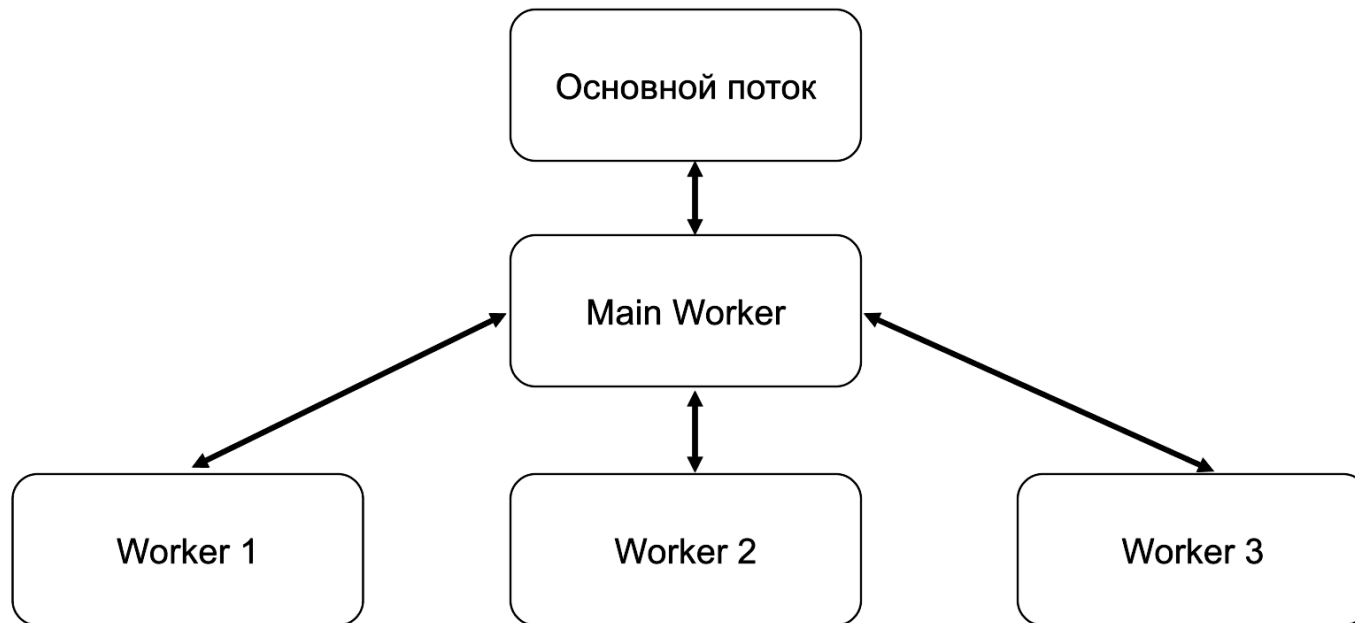
```
02. const points = wasm.calculatePoints(xArray);
```

Как всё подружить с веб-воркерами?

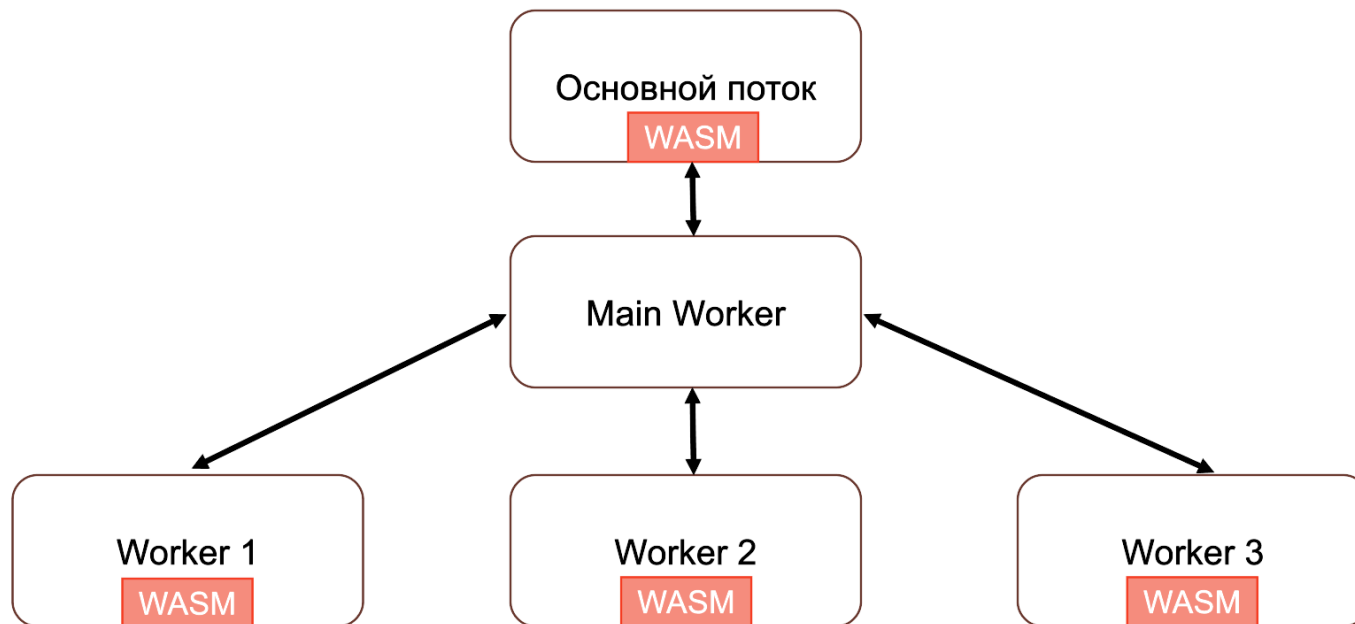
Web workers

- Механизм, который позволяет скрипту выполняться в фоновом потоке
- Основной поток без блокировки и замедления
- Worker'ы могут запускать другие worker'ы
- Они общаются друг с другом через `postMessage`

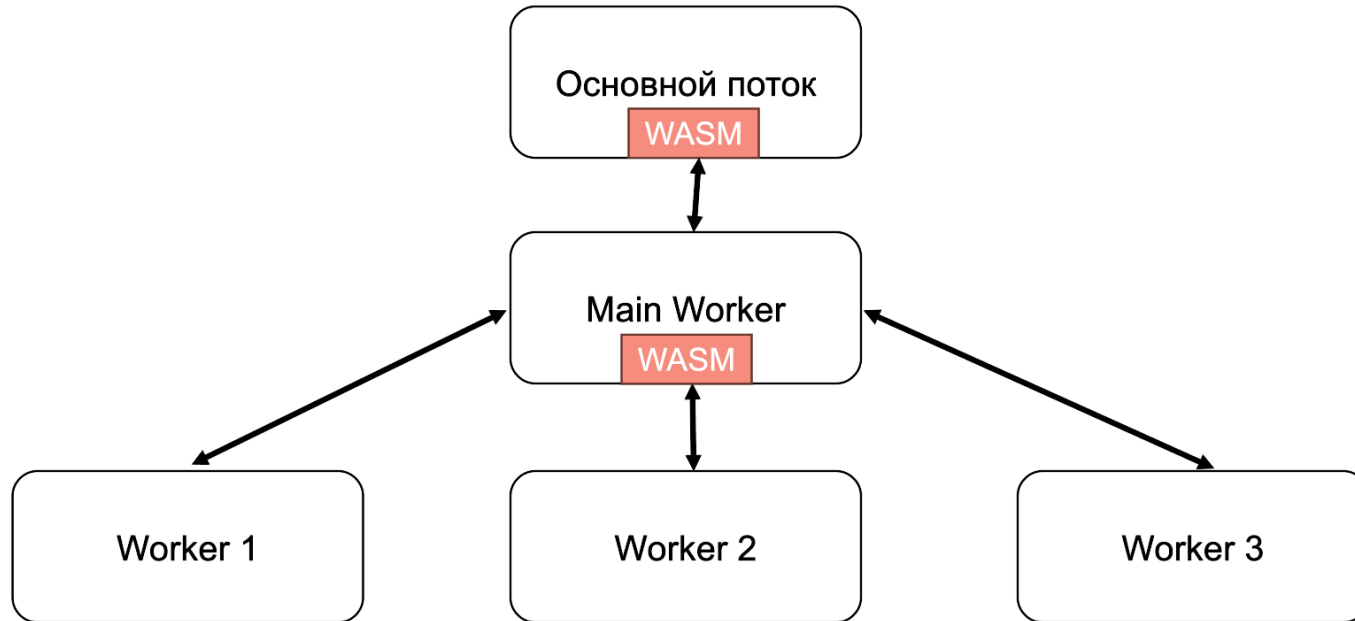
Web workers в StarLite



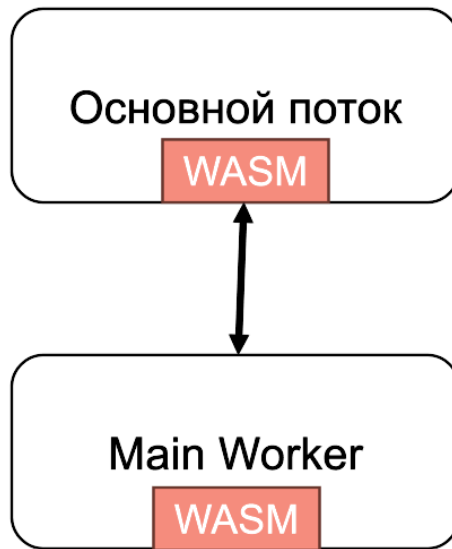
Web workers & WASM



Web workers & WASM



Web workers & WASM



Как тестировать WASM?

Тестирование

- Unit-тесты на стороне C++
- Интеграционные тесты QAA

Jest

```
01. import { readFileSync } from 'fs';  
02.  
03. const wasmBinary = readFileSync(  
04.   'node_modules/path-to-wasm/math.wasm',  
05. );  
06.  
07. instance = await createModule({ wasmBinary });
```


Результаты

Плюсы

- Скорость расчётов выросла (в 2 – 10 раз)
- Расчёты на разных платформах совпадают
- Нет дублирования математических расчётов

Минусы

- Сложность отладки
- Зависимость от другой команды
- Долгая итерация внедрения

Когда стоит использовать WASM?

- Нужные функции уже написаны на другом языке
- У вас большие объёмы данных, и вам нужен прирост в скорости
- У вас есть ресурсы на внедрение

Пожалуйста, оставьте СВОЙ ОТЗЫВ

Павел Дадыкин
ROGII

<https://meloman4eg.github.io/wasm-fc-2024/>

[@meloman4eg](https://github.com/meloman4eg)



FC

**Frontend
Conf 2024**