Браузер как платформа для “тяжелых” приложений

О авторах

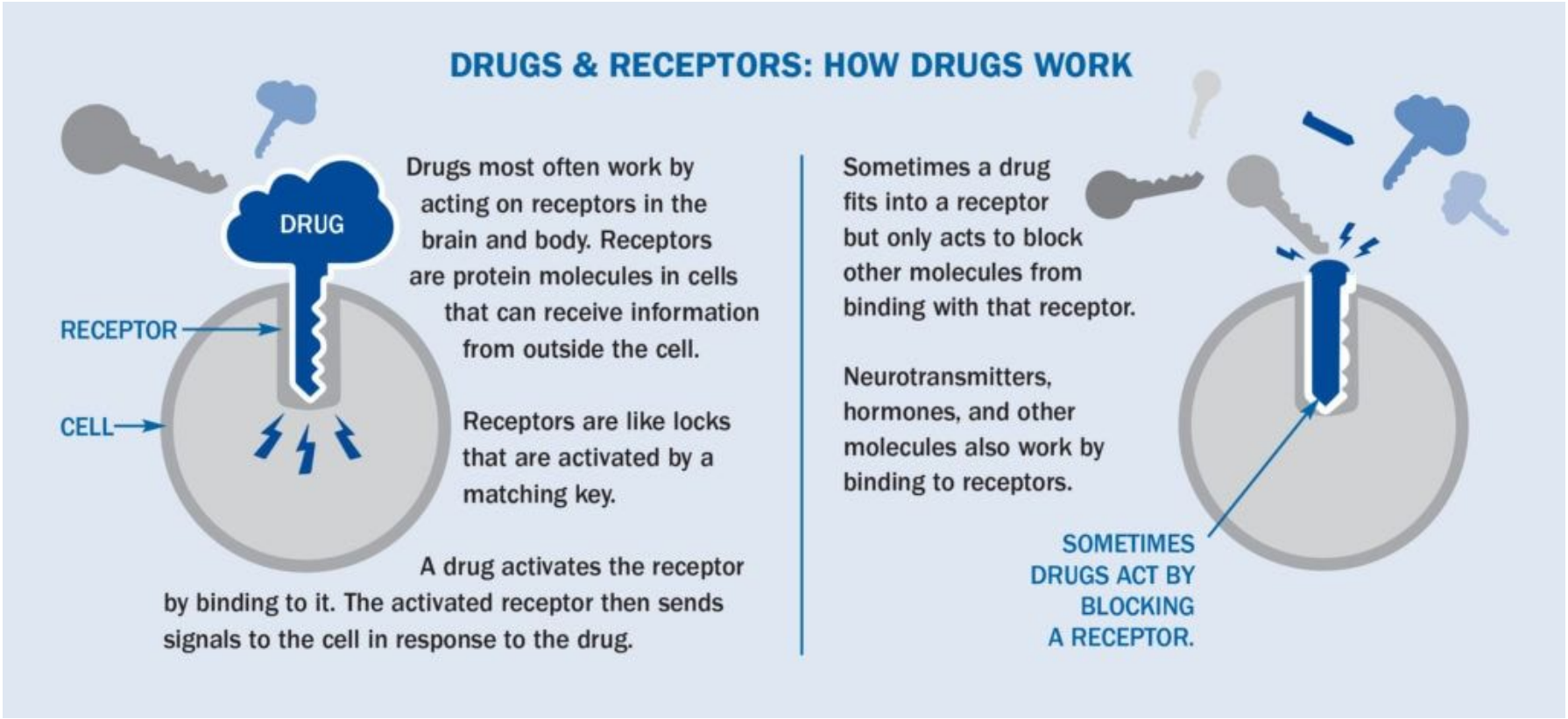
Иван Затравкин

Тех лид, любитель поделиться знаниями,

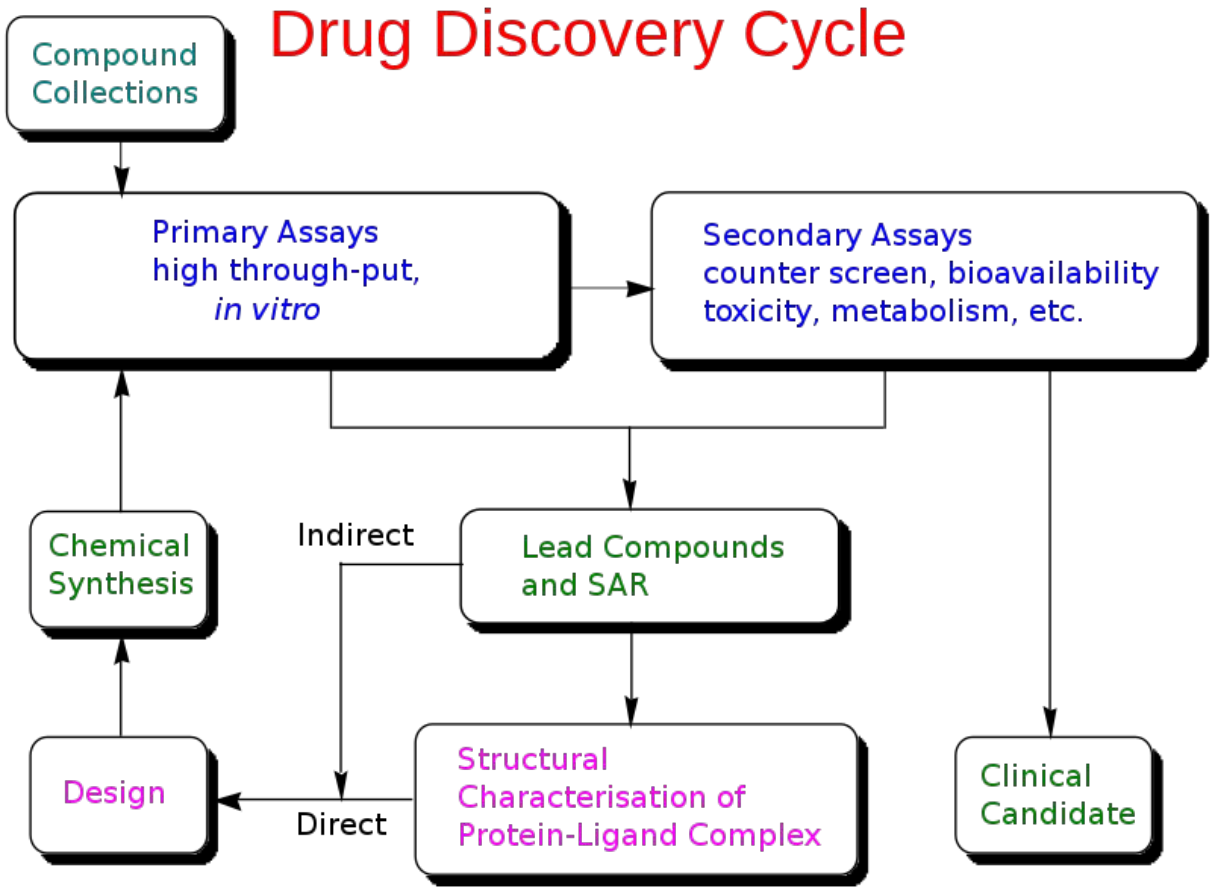
в свободное время пишу игры.

О чем будем говорить?

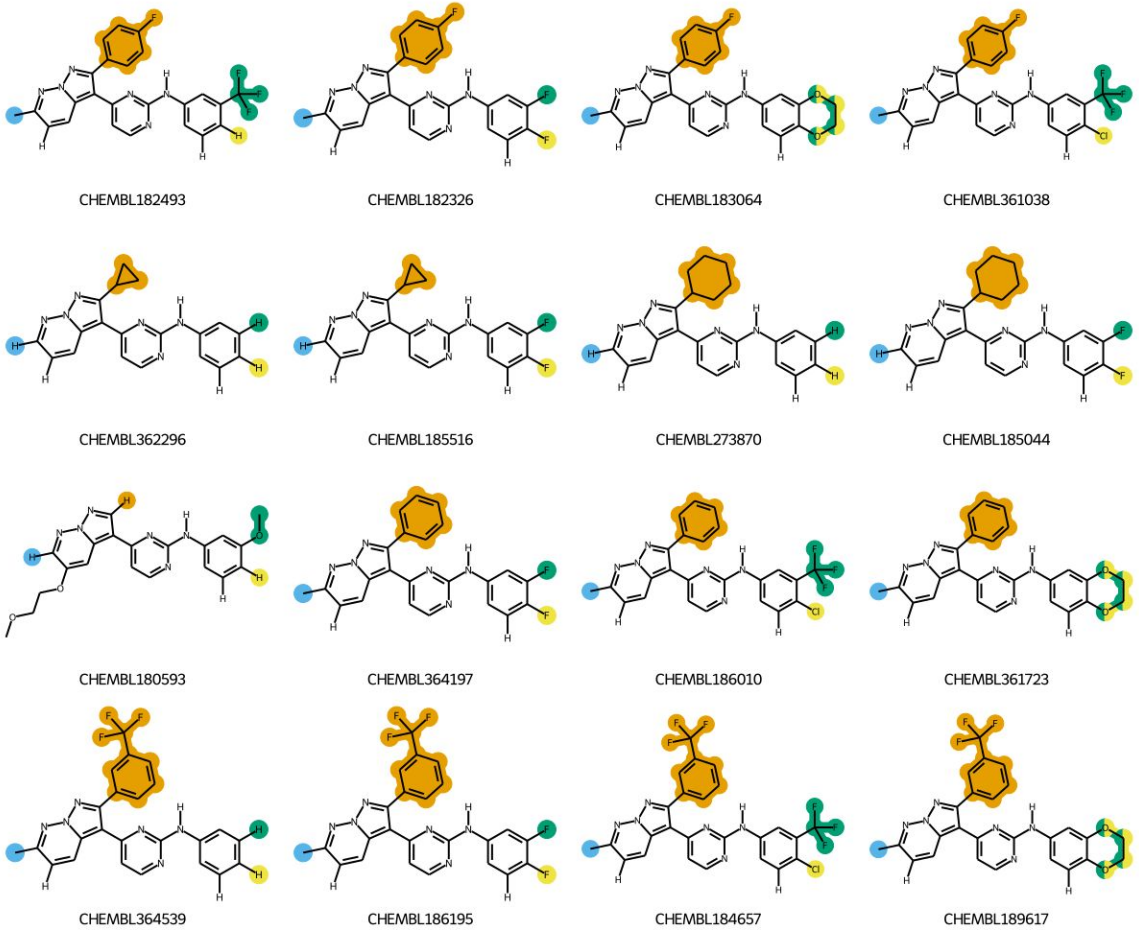
- Формат: case-study
- История проекта
- Эволюция решений
- Что не взлетело
- Итоги



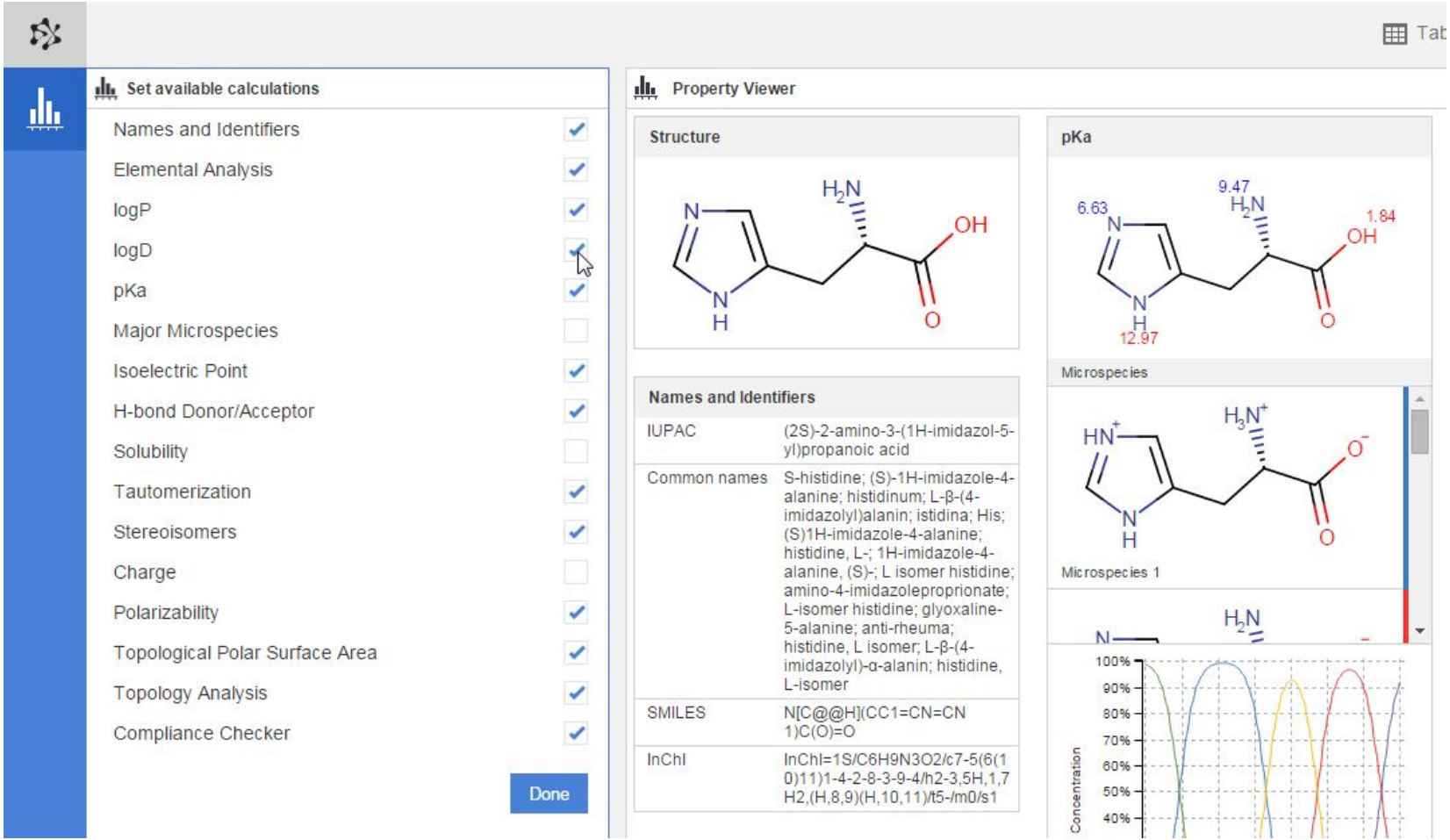
Как действуют лекарства https://dystonia-foundation.org/on-target-drug-discovery/



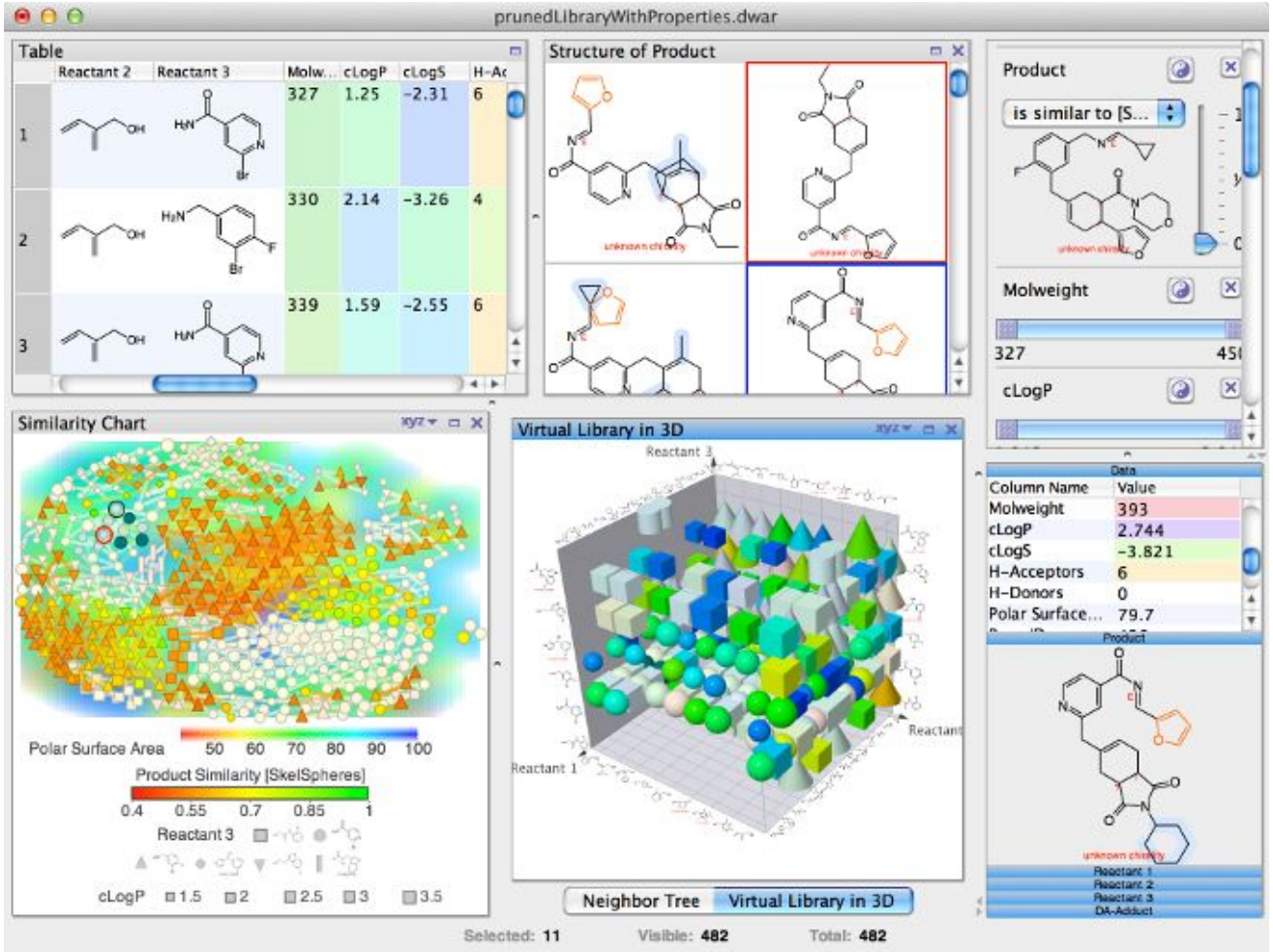
Цикл разработки лекарств https://en.wikipedia.org/wiki/Drug_discovery



Структура может быть связана с активностью. Иллюстрация - RGroup decomposition
https://greglandrum.github.io/rdkit-blog/posts/2021-08-07-rgd-and-highlighting.html



Физические свойства могут быть рассчитаны на основе структуры. Иллюстрация ChemAxon
https://docs.chemaxon.com/display/docs/discontinued_plexus-suite_calculating-molecular-properties-for-single-compounds.md



DataWarrior - Open Source решение https://openmolecules.org/datawarrior/

О проекте - с чего начинаем

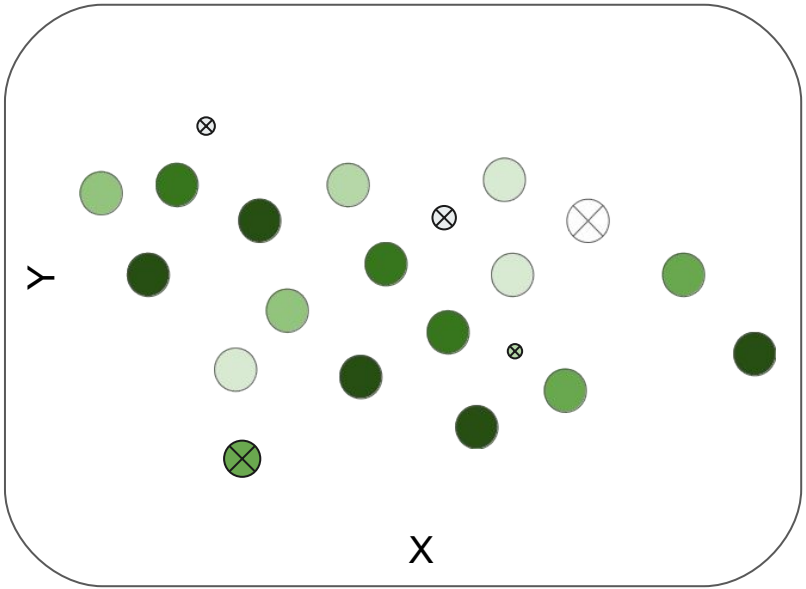
- JS образца 2014 года
- Bower - менеджер пакетов
- Gulp для сборки
- Vanilla JS для интерфейса
- Pixi.js для графиков
- D3 для расчёта координат

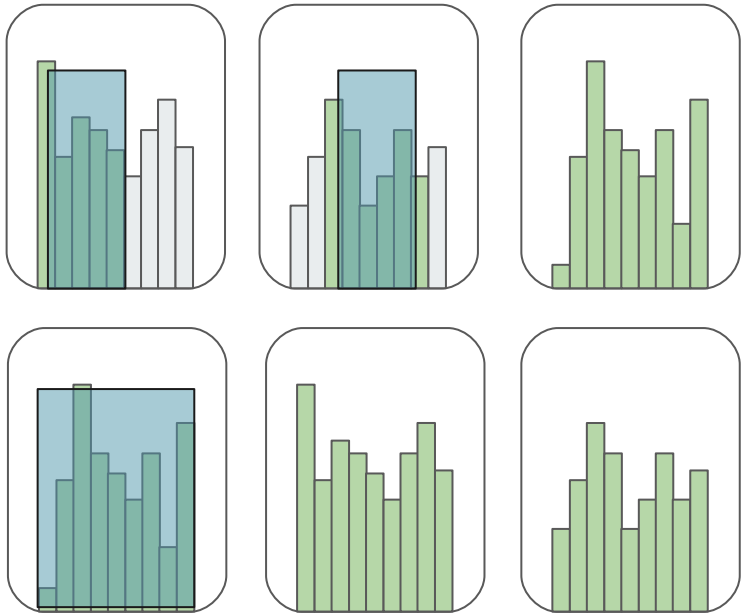
Прежде чем начать рефакторинг

- Разбиваем приложение на функциональные части
  - Система фильтрации
  - Визуализация
  - Хим-информатика
- Описываем интерфейсы их взаимодействия

Система фильтрации

Система фильтрации





Система фильтрации - задача

- Производительность на большом количестве измерений
- Производительность на большом количестве точек
- Больше форматов данных: строки, химические структуры, булины
- Поддержка записи, не только чтения

Система фильтрации - можно ли быстрее

- Начальная точка - 0.1 FPS на 100к молекул
- 1Ghz = 1e9 операций в секунду
- Количество операций постоянно с числом элементов
- 100 операций на один элемент
- 100 000 молекул
- 1e9/1e2/1e5 = 1e2 = 100 FPS
- 4 фильтра сразу -> 100 fps / 4 = 25

Fermi Estimate

https://brilliant.org/wiki/fermi-estimate/

Интерфейс

```
type Data = {
  value: number;
};
```

```
type Row = readonly [{ id: number },...Data[]];
```

```
type Column = {
  id: number;
  name: string;
  idx: number;
};
```

```
type Dataset = {
  columns: Column[];
  rows: Row[];
};
```

```
type Filter = {
  column: number;
  min: number;
  max: number;
  includeMin: boolean;
  includeMax: boolean;
};
```

Система фильтрации - SQL

Используем стандарт фильтрации данных: SQL

```
type Filter = {
  column: number;
  min: number;
  max: number;
  includeMin: boolean;
  includeMax: boolean;
};
```

```
SELECT * FROM data WHERE `${column}` > `${min}`
```

Система фильтрации - SQL

1 FPS

Система фильтрации - JS

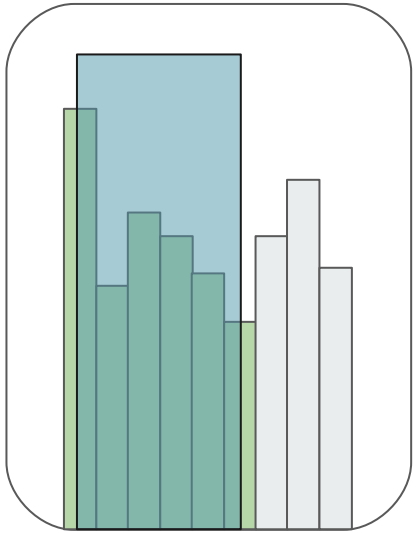
```
const filterData = (dataset: Dataset, filters: Filter[]) => {
  return dataset.rows.filter((row) => {
    return filters.every((filter) => {
      const value = (row[filter.column]).value;
      return (
        (filter.includeMin ? value >= filter.min : value > filter.min) &&
        (filter.includeMax ? value <= filter.max : value < filter.max)
      );
    });
  });
};
```

Система фильтрации - JS

10 FPS

Система фильтрации - большие датасеты

Можно пересчитывать только границы.

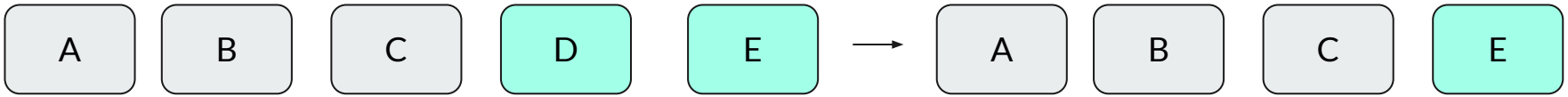


Система фильтрации - большие датасеты

10 FPS -> 60 FPS

Система фильтрации - добавление колонок

```
type Row = readonly [{ id: number },...Data[]];
```



Система фильтрации - колоночная БД

```
type AllData = Row[]
```



```
type AllData = {
    [column: string]: Value[]
}
```

Система фильтрации

- Как уведомлять React о мутациях?
- Создавать новый массив данных - дорого
- Proxy!

```
private originalData

public data = new Proxy(originalData, {})
```

Система фильтрации - Итог

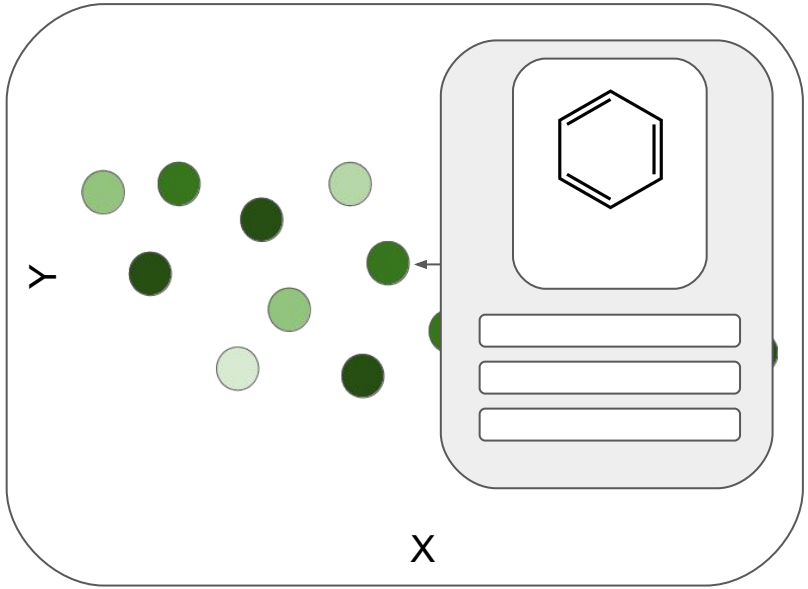
- Попробовали SQL: 0.1fps -> 1 fps
- Простое решение на JS: 1 fps -> 10 fps
- Пересчёт границ: 10 fps -> 60 fps
- Перешли на колоночную структуру для лёгких мутаций
- Переиспользуем память через Proxy

Визуализация

Визуализация

Задача:

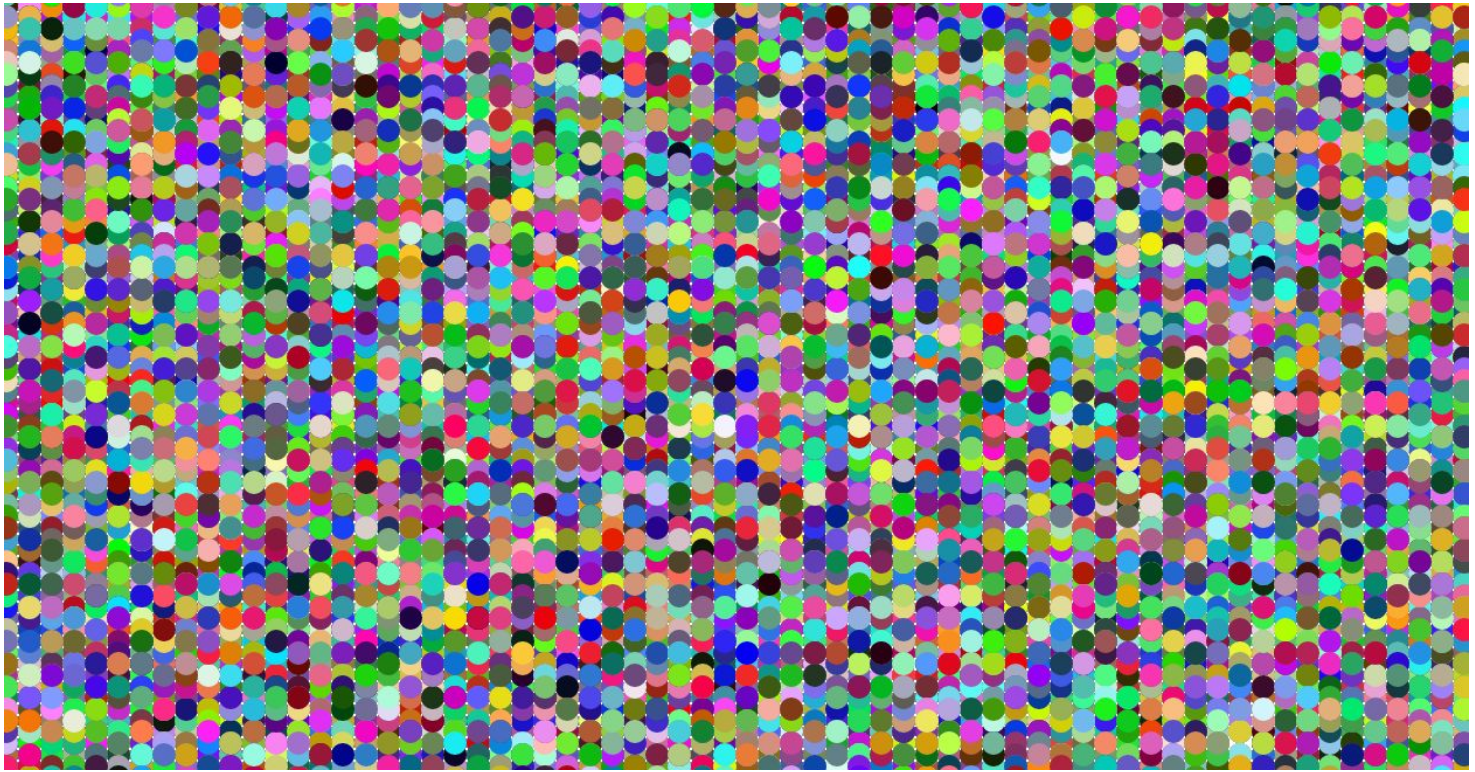
- Отобразить интерактивный график с точками
- Точки могут исчезать/видоизменяться во время фильтрации



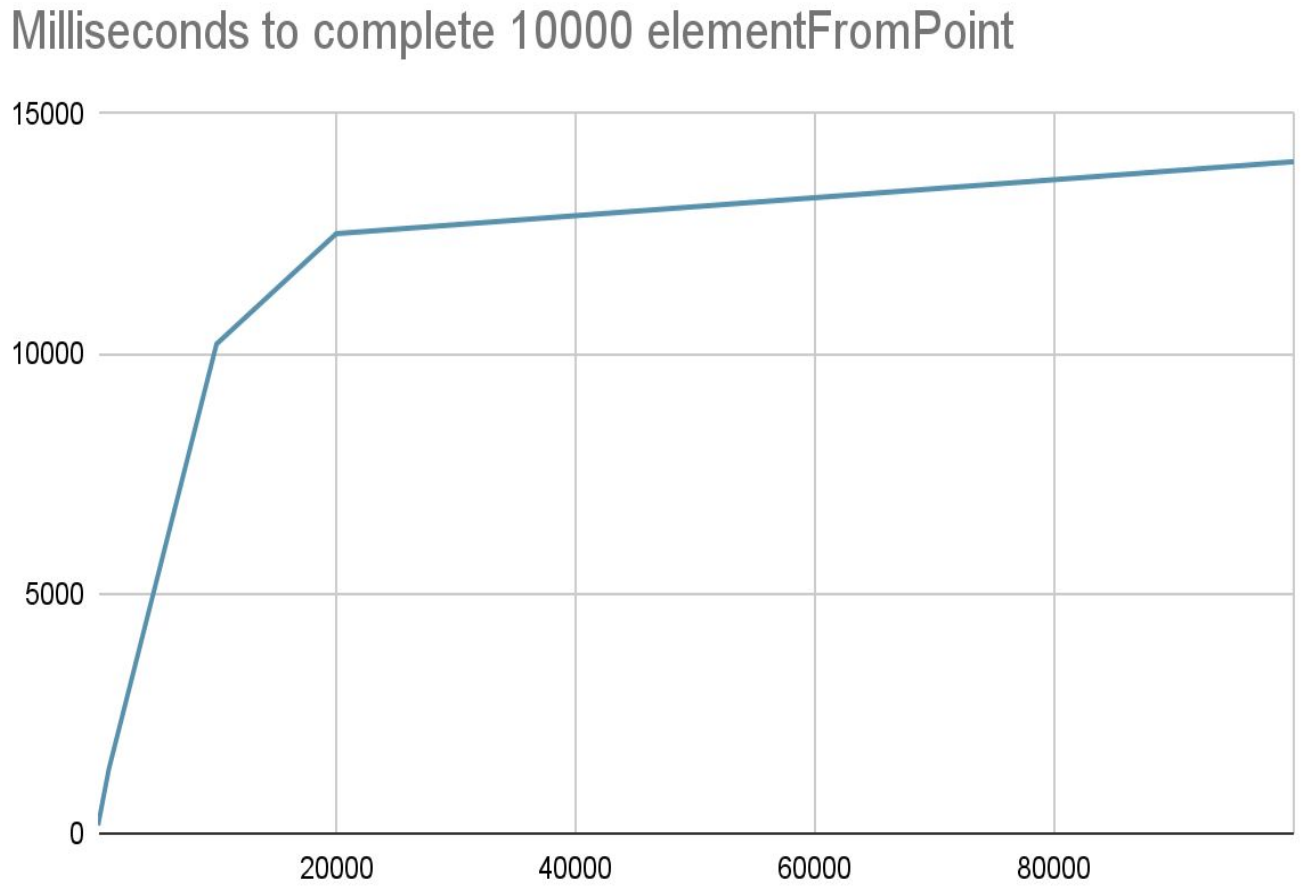
Визуализация - что есть изначально

- Решение на PixiJS
- Sprite для каждой точки
- В Sprite рисуется Graphics с кругом
- На 100к точек 100% загрузка CPU и 10 FPS при интерактиве
- 1 ГБ потребление памяти на 100к точек при 4 графиках
- 5 секунд на добавление нового графика при 100к точек

Визуализация - интерактив



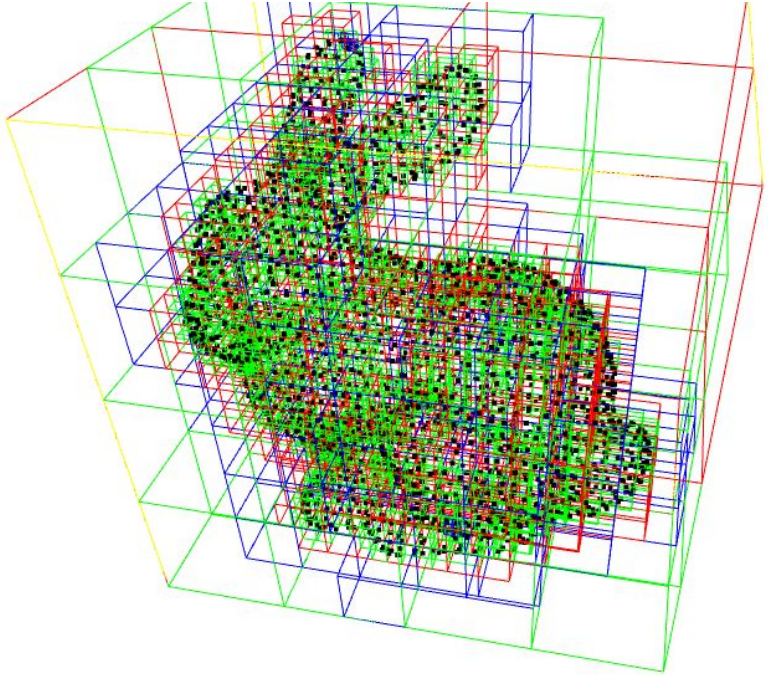
На что наведена мышка?



Браузеру тоже сложно

Визуализация - интерактив

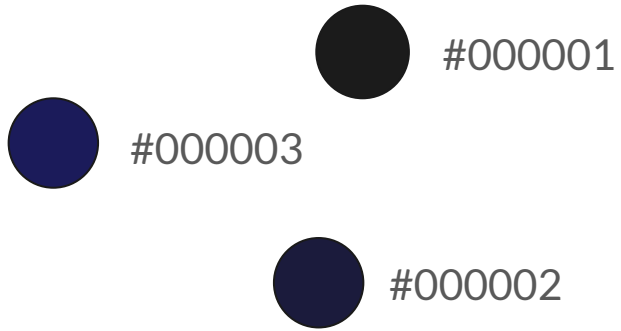
Пространственные структуры данных



https://observablehq.com/@2talltim/spatial-data-structures-octrees-bsp-and-k-d-trees

Визуализация - интерактив

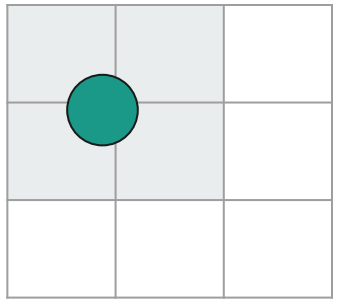
Рендер ID в текстуру



https://webglfundamentals.org/webgl/lessons/webgl-picking.html

Визуализация - интерактив

- Сетка
- Простота реализации
- Эффективно по памяти - 10к массивов для графика 1000x1000
- 60 FPS при 1% загрузке CPU



Визуализация - медленное добавление графиков

- Добавление графика на 100к точек занимает 5 секунд
- Основное время идёт на инициализацию объектов
- Мы можем переиспользовать объекты - Object Pool

Визуализация - потребление памяти

- 1 GB на 4 графика по 100к точек
- 250 MB на график
- 1e5 точек
- 2 float на координаты, 1 на цвет, 1 на размер
- 1 float = 4 byte
- 1e5*6*4 ~= 2.4e6 ~= 2.4 MB

Визуализация - WebGL

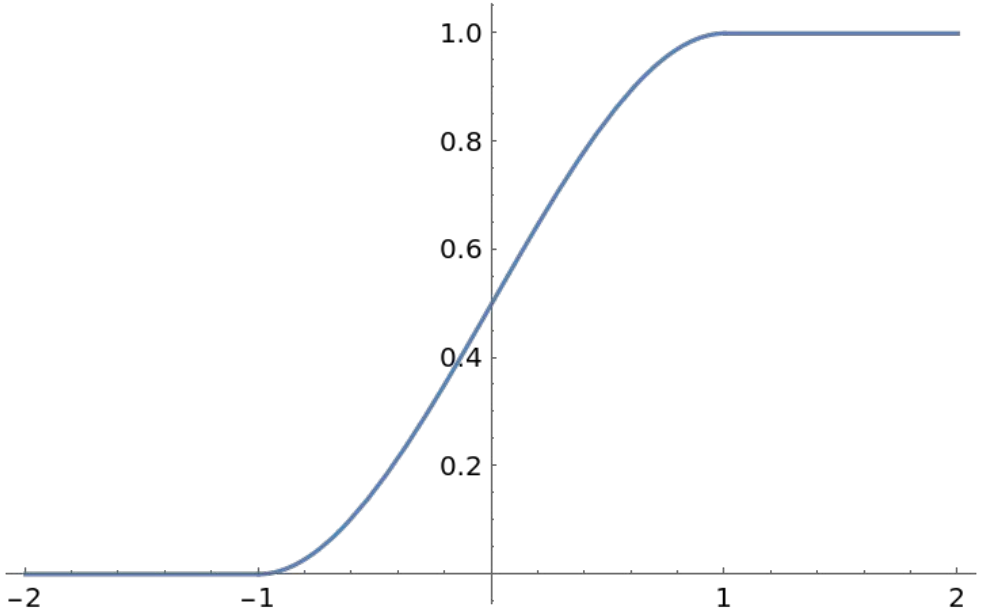
- Считаем координаты точек с помощью d3.js на CPU
- Передаем координаты на GPU
- В фрагментном шейдере рисуем круг - step(radius, distance(point, center))

Визуализация - WebGL

Раньше было лучше

Визуализация - WebGL

```
smoothstep(radius, radius + 0.005, len);
```



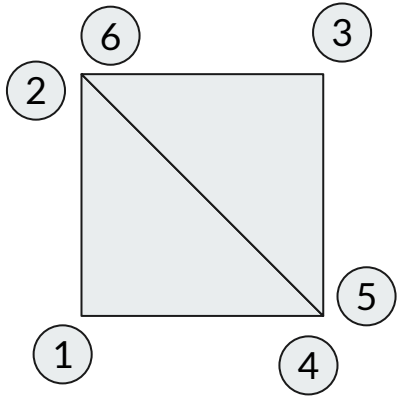
https://resources.wolframcloud.com/FunctionRepository/resources/SmoothStep

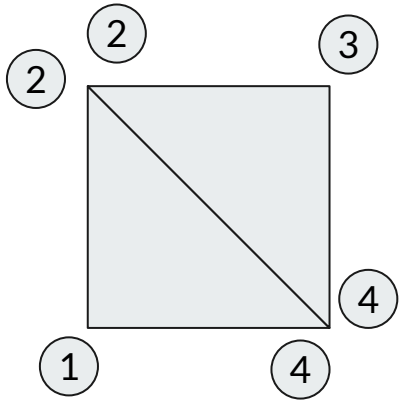
Визуализация - WebGL

Большое количество точек работает медленно

Визуализация

drawElements позволяет переиспользовать общие вершины

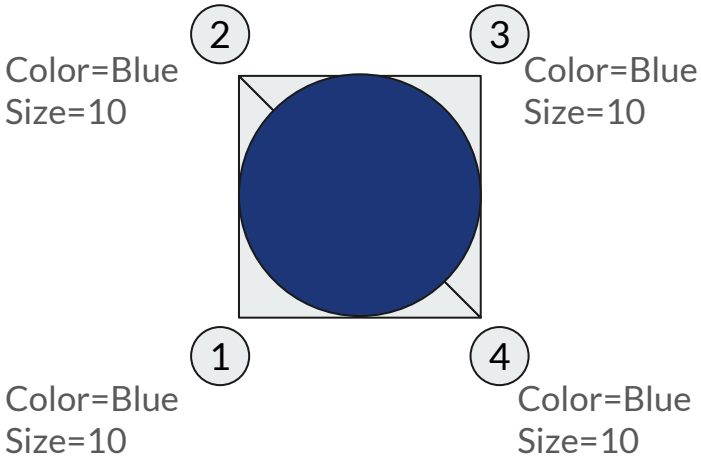




https://webglfundamentals.org/webgl/lessons/webgl-indexed-vertices.html

Визуализация

Дублирование данных

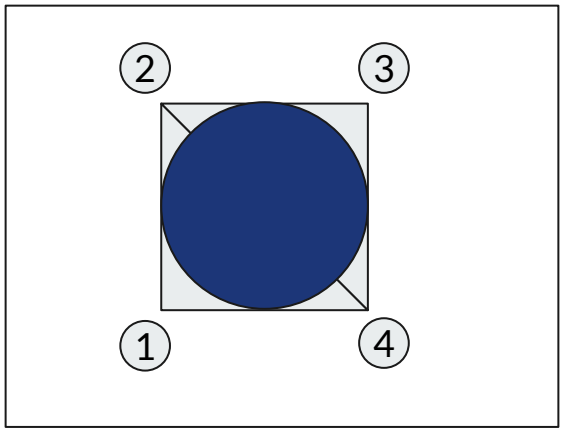


Визуализация

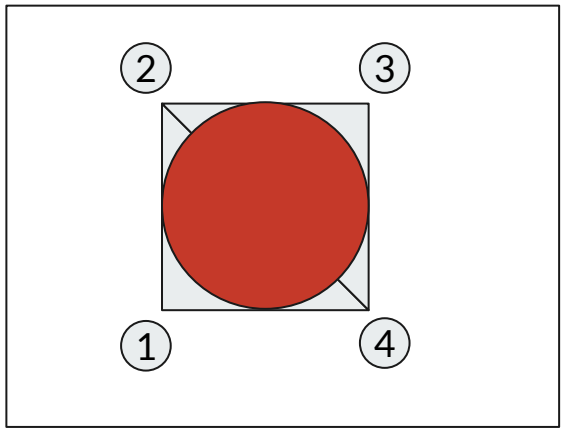
drawElementsInstanced

40 FPS

Color=Blue
Size=10



Color=Red
Size=10




IT WORKS ON MY MACHINE
THEN WE'LL SHIP YOUR MACHINE

Визуализация

WARNING: Too many active WebGL contexts. Oldest context will be lost.

Ограниченное количество WebGL контекстов

Использование общего контекста для всех графиков.

Визуализация

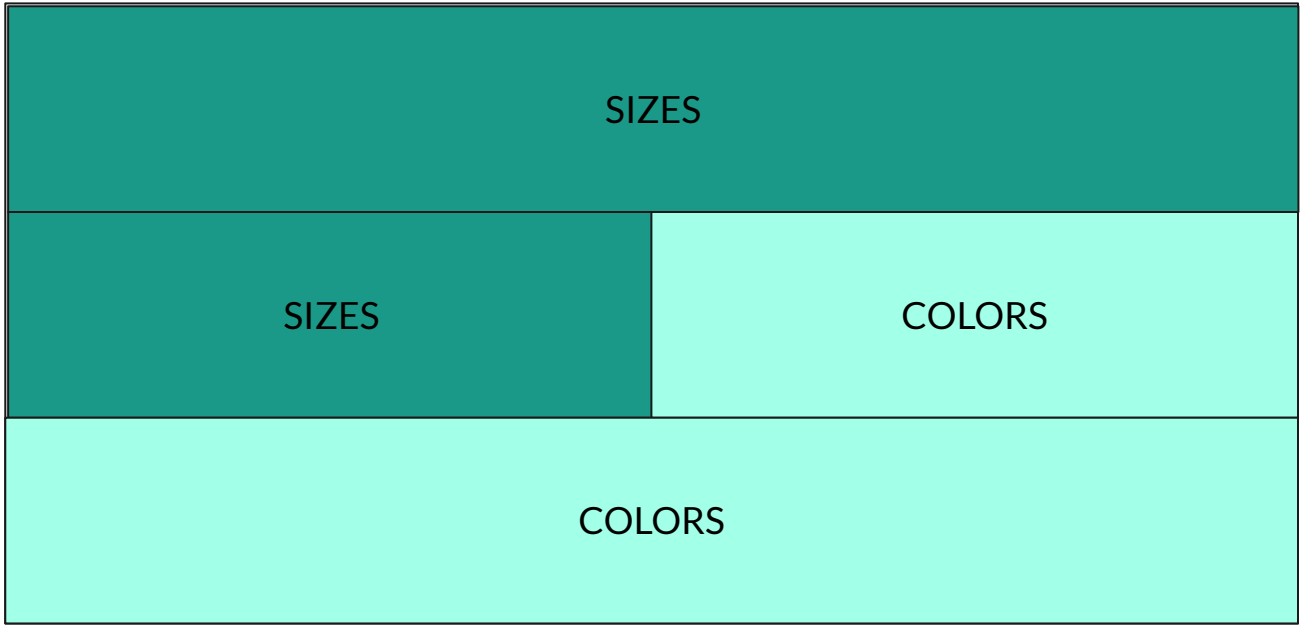
ERROR: Too many attributes

gl.getParameter(gl.MAX_VERTEX_ATTRIBS) -> 8

Packing data in textures!

Визуализация - Texture Packing

TIP: use 2^N size



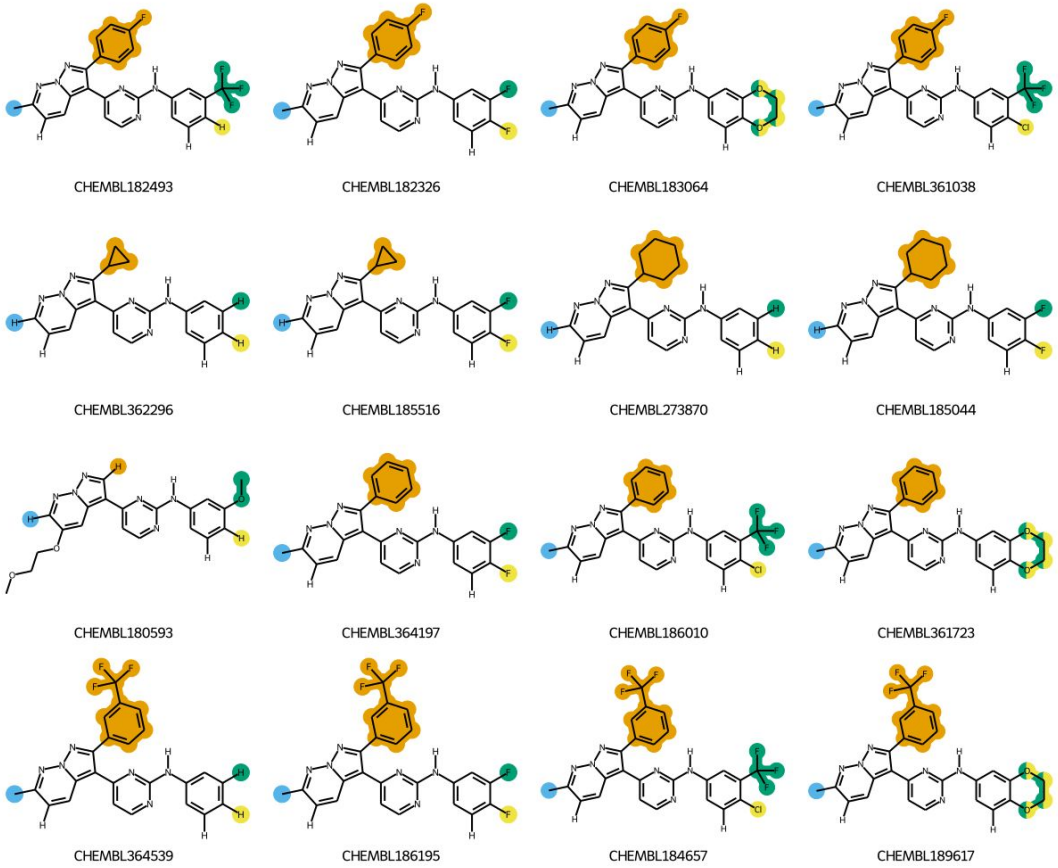
Визуализация - Итог

- Реализовали интерактив с помощью сетки
- Написали рендер на WebGL
- Получилось быстро
- После оптимизаций - быстро и на больших наборах данных
- Проблемы с драйверами видеокарты теперь и в браузере
- Safari - боль
- Обзавелись парком устройств для тестирования
- Интеграционное тестирование - сложно

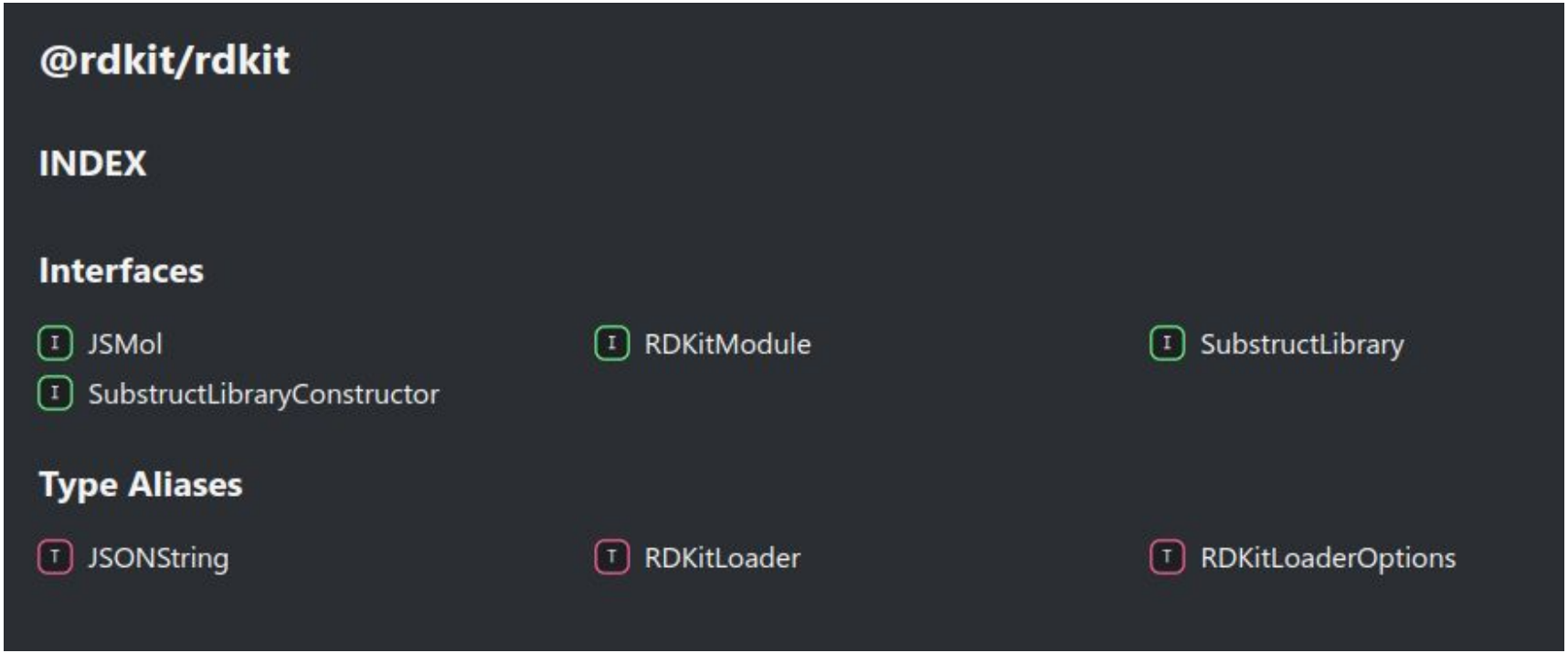
Химинформатика

Химинформатика

- Фильтрация по структуре
- Подсветка фрагментов
- Расчёт свойств
- Декомпозиция



Химинформатика - RDKit



Химинформатика

```
using namespace emscripten;
EMSCRIPTEN_BINDINGS(RDKit_minimal) {
  register_vector<std::string>("StringList");

  class_<JSMol>("Mol")
      .function("is_valid", &JSMol::is_valid)
      .function("has_coords", &JSMol::has_coords)
      .function("get_smiles", &JSMol::get_smiles)
      .function("get_cxsmiles", &JSMol::get_cxsmiles)
      .function("get_smarts", &JSMol::get_smarts)
      .function("get_cxsmarts", &JSMol::get_cxsmarts)
      .function("get_molblock",
                select_overload<std::string() const>(&JSMol::get_molblock))
      .function("get_molblock",
                select_overload<std::string(const std::string &) const>(
                    &JSMol::get_molblock))
```

Химинформатика - emscripten

Доступ к функциям из C++ в браузере

```
#include <emscripten/bind.h>
using namespace emscripten;

float add(float a, float b) {
   return a + b;
}

EMSCRIPTEN_BINDINGS(my_module) {
   function("add", &add);
}
```

Химинформатика

Сборка

```
./emcc app.cpp
```


That's all Folks!

Химинформатика - обмен данными

- Обмен JSON между JS и WASM
- Мы используем https://rapidjson.org/ на стороне C++
- Это неэффективно, но просто и быстро реализовать
- Лучший вариант возможен с protobuf

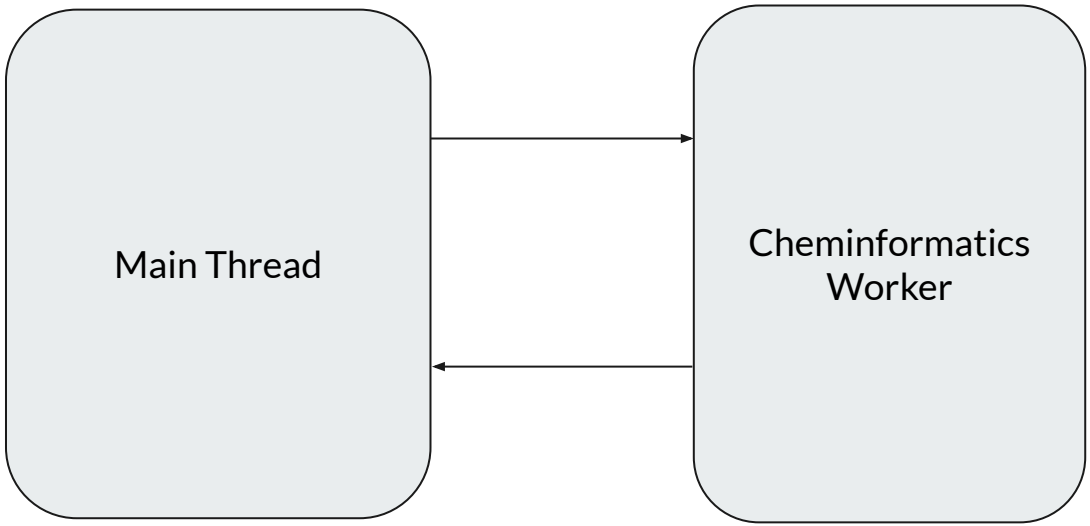
Химинформатика - проблемы сборки

- Сборка занимает 15 минут
- Хранение артефакта в коде
- Параллельные пры ломают артефакт
- Браузерный кэш ломает артефакт

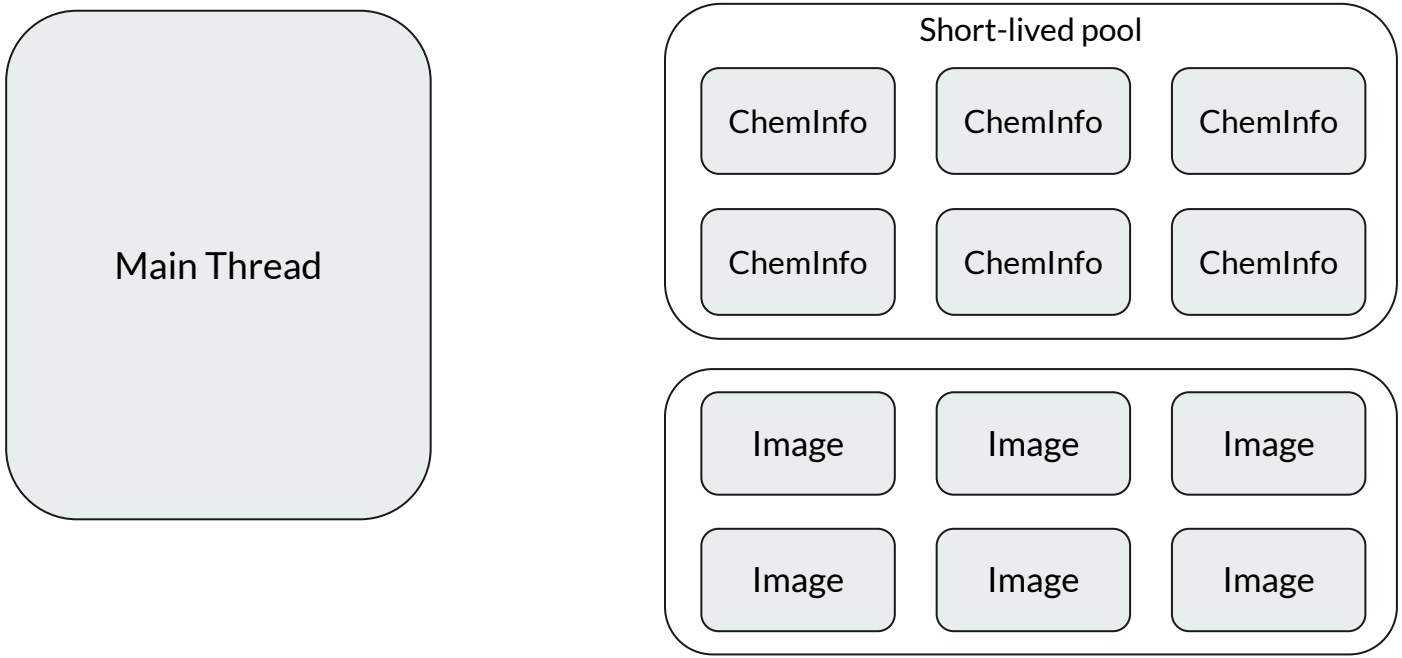
Химинформатика - производительность

- Расчёт химических свойств занимает до 1 минуты
- Изображения молекул появляются с задержкой
- R-Group decomposition фризит интерфейс

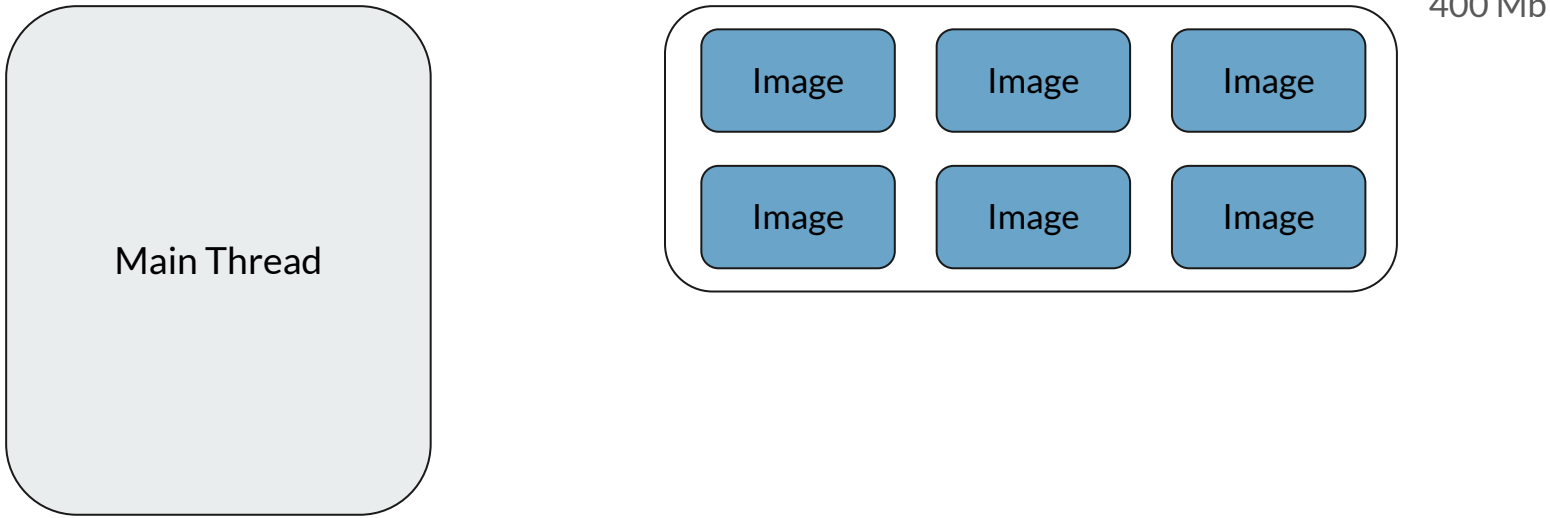
Химинформатика - WebWorkers



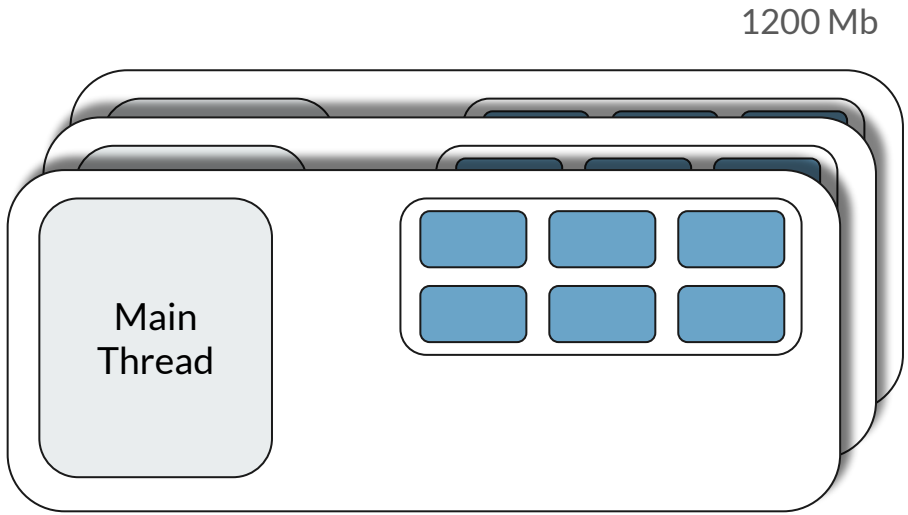
Химинформатика - Worker Pools



Химинформатика - Worker Pools



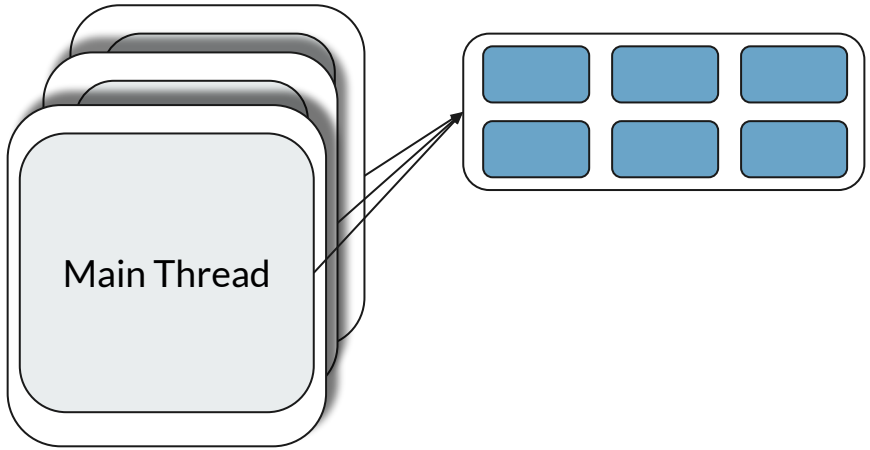
Химинформатика - Worker Pools



Химинформатика - Shared Workers

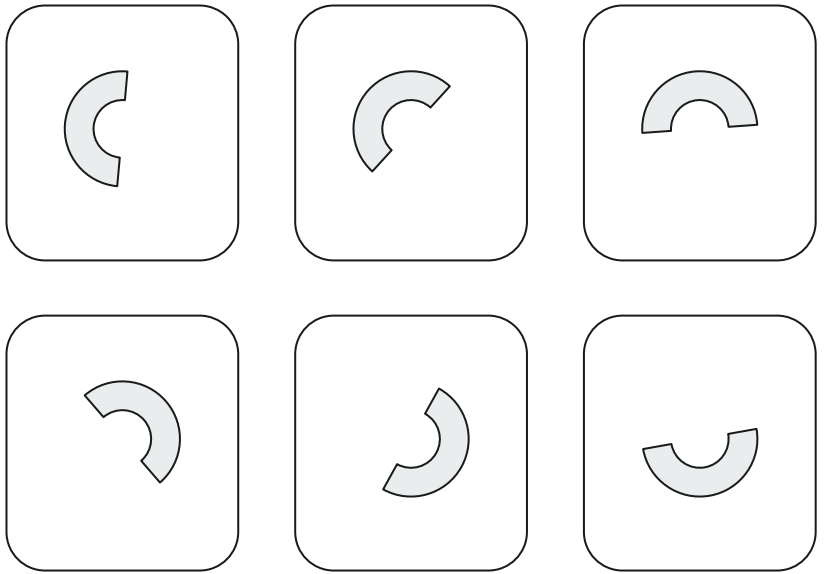
```
const imageWorker = new SharedWorker("worker.js?id=N");
```

400 Mb



Химинформатика - Image Worker

Зависшие спинеры из-за бэкграунд задач

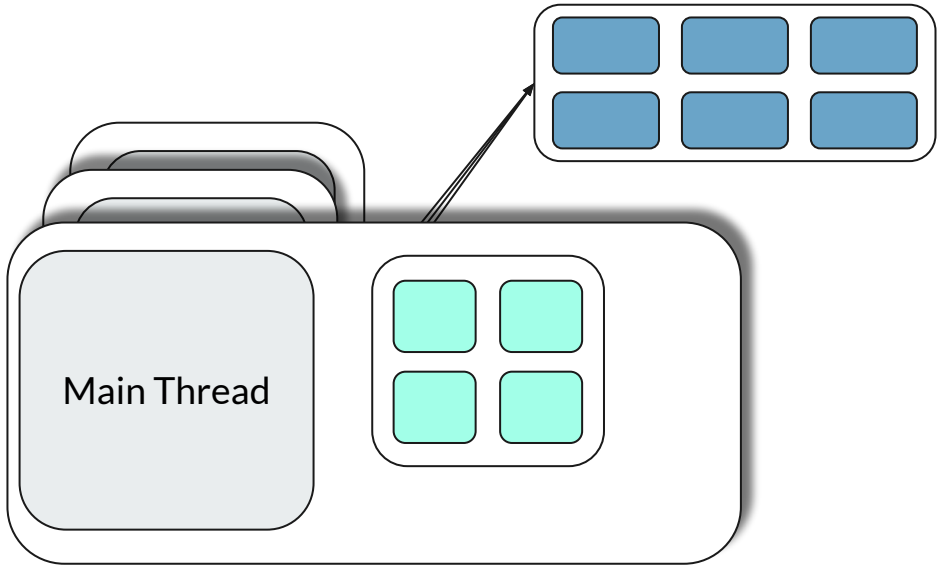


Химинформатика - Read-Only replicas

- Для ускорения доступа к данным мы хотим создать Read-Only копии
- В emscripten прозрачно работает SharedArrayBuffer
- SharedArrayBuffer заблокирован CSP политиками
- Можно копировать память модуля руками

Химинформатика - итоги

- WASM работает
- DX страдает
- Фичи
  - Рендер молекул
  - Поиск
  - Декомпозиция
  - Расчёт свойств



Что не получилось

Фильтрация на SQL

- WebAssembly - виртуальная машина
- WebAssembly работает параллельно с JS (register pressur
- Медленный обмен данными с JS

https://www.usenix.org/system/files/atc19-jangda.pdf

Фильтрация данных в WebWorker

- Неподходящие интерфейсы (они синхронные)
- Дублирование данных на каждый воркер -> высокое потребление памяти
- Нужен общий кэш на все воркеры

Фильтрация данных в WebGL

- Это быстро
- Но считать данные обратно медленно
- Считать данные - O(N), фильтрация тоже O(N) и константа не сильно отличается
- Возможно в WebGPU будет лучше

Итоги

Итоги

- Эффективно работаем с большими объемами данных
- Реализовали химинформатику внутри браузера
- Многопоточное приложение с сервисной архитектурой на клиенте в браузере

Полезные материалы

Полезные материалы

- WebGL
  - Branchless programming: https://en.algorithmica.org/hpc/pipelining/branchless/
  - Введение в WebGL: https://webglfundamentals.org/
  - Shadertoy: https://www.shadertoy.com/
  - Glsify: https://github.com/glslify/glslify
  - React Three Fiber: https://docs.pmnd.rs/react-three-fiber
- WebAssembly
  - Showcase: https://madewithwebassembly.com/
  - WebAssembly intro: https://hacks.mozilla.org/category/code-cartoons/a-cartoon-intro-to-webassembly/
  - WebAssembly performance: https://www.usenix.org/system/files/atc19-jangda.pdf
  - SIMD: https://en.algorithmica.org/hpc/simd/
  - UI: egui, blazer