Node.js in 2020 #1

Выйди и зайди нормально

github.com/HowProgrammingWorks

Timur Shemsedinov

Chief Technology Architect at Metarhia
Lecturer at Kiev Polytechnic Institute

github.com/tshemsedinov

github.com/tshemsedinov
https://youtube.com/TimurShemsedinov
github.com/HowProgrammingWorks/Index

Весь курс по ноде (>35.5 часов)
https://habr.com/ru/post/485294/

t.me/HowProgrammingWorks
t.me/NodeUA

timur.shemsedinov@gmail.com

Node.js Starter Kit

no dependencies: 15kb size, with PG drivers + 1.2mb

Starter Kit Key Ideas

- Минимум кода и зависимостей
- Минимизация I/O, отдача всего из памяти
- Безопасность и изоляция контекстов
- Структура и архитектура приложения
- Разделение системного и прикладного слоя
- Все на контрактах (interface)
- Балансировка, таймауты и очередь запросов

Starter Kit Feature List

- Автороутинг API и поддержка HTTP(S), WS(S)
- Подгрузка изменений на лету через fs.watch
- Загрузчик конфигурации и Graceful shutdown
- Утилизация CPU, кластеризация через потоки
- Слой доступа к данным DAL: Postgresql
- Сессии с сохраняемым состоянием
- Песочницы, потоки, масштабирование, тесты

Node.js in 2020

State of the platform and future

Node.js уже 10 лет: v0.0.1 — 27 мая 2019

0.10.x и 0.12.x — (2013 - 2016)
Io.js 1.x, 2.x, 3.x — (2014 - 2015)

Argon 4.x (2015 - 2018), 5.x (2015 - 2016),
Boron 6.x (2016 - 2019), 7.x (2016 - 2017),
Carbon 8.x (2017 - 2019), 9.x (2017 - 2018),
Dubnium 10.x (2018 - 2021), 11.x (2018 - 2019),
Erbium 12.x (2019 - 2022), 13.x (to June 2020)
14.x (April 2020 - April 2023), 15.x(2020-)

Node.js features

8.x V8 6.0, async/await, TurboFan and Ignition

10.x V8 6.6, HTTP/2, fs.promises, BigInt, npm 6

12.x V8 7.8, TLS 1.3, OpenSSL 1.1.1c, npm 6.10.3

js: #, static, async/await, async stack, динамическая куча, llhttp и llparser, threads, DOS в HTTP/2, startup, fs.rmdir & fs.Dir, process.resourceUsage()

Node.js features

13.x V8 7.8, npm 6.13.6, libuv 1.34.1
WASI, worker.resourceLimits, vm.Module
Source map, Advanced Serialization API

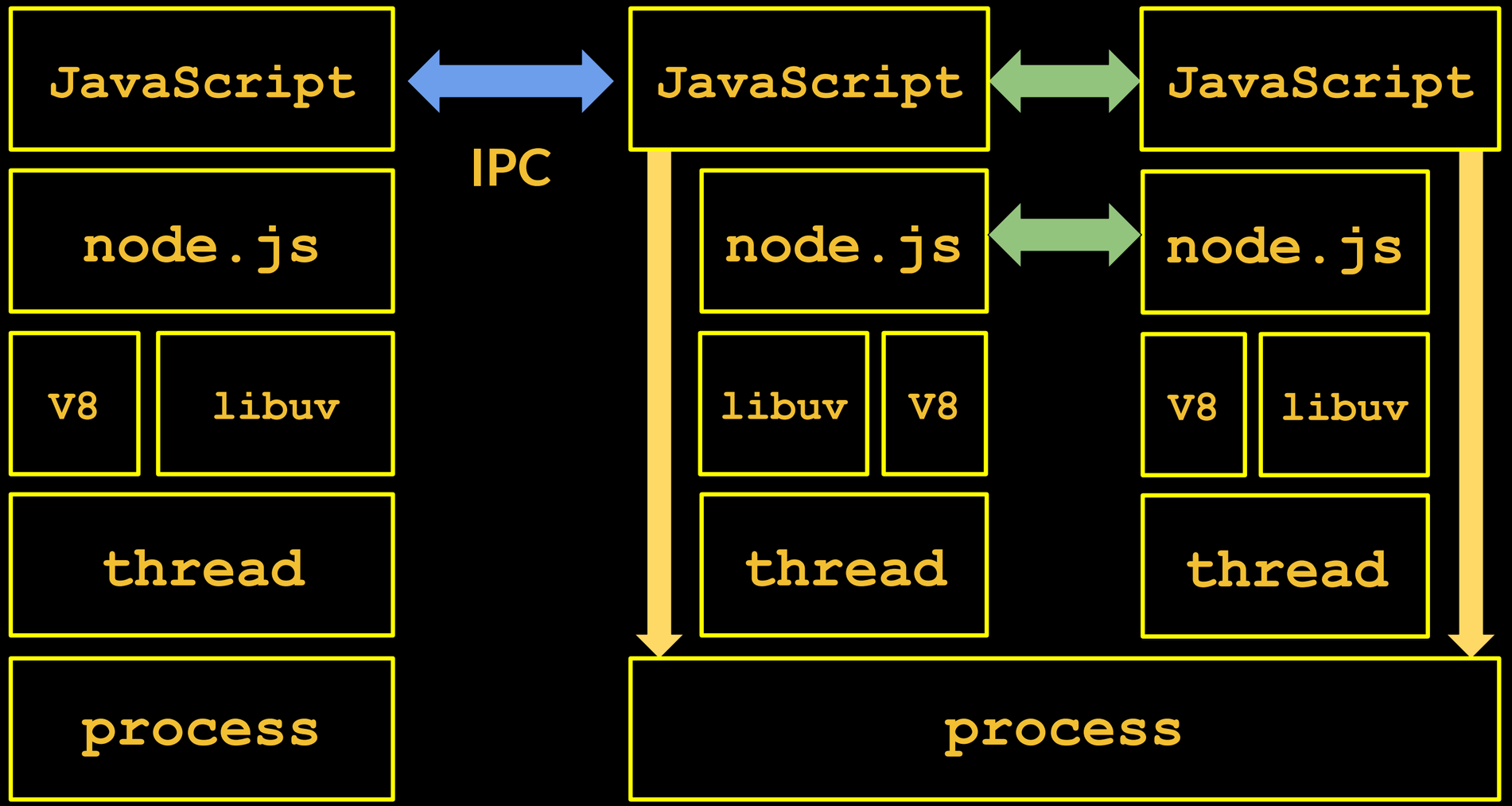
14.x Ожидания: V8 8.x: больше доступа к v8 api
HTTP/3 (HTTP over IETF QUIC)
улучшения в WASI, N-API, async_hooks
streams, в работе с workers и esm

Shared memory and Atomics

Появилось в Node.js 9

- SharedArrayBuffer
- Atomics
  - add, sub, and, or, xor
  - store, load, exchange, compareExchange
  - notify, wait, ~~wake~~ (deprecated)

Threads vs Processes



Why Isolation?

- Ошибки
- Утечки памяти и других ресурсов
- Приложение: данные, соединения с БД
- Файловая система и корневой каталог
- Окружение ОС, PID, IPC
- Безопасность ОС: пользователи, группы
- Сеть: дескрипторы сокетов, порты, хосты

Execution Strategy Problems

- Недостаточная изоляция исполнения запросов к серверу друг от друга
- Один неудачный запрос может убить все параллельно исполняемые
- В асинхронной среде сложно найти и связать ошибку с запросом
- Изоляция приложений и организаций в SaaS

Execution Isolation

- VPS (виртуальная машина)
- Контейнер (Docker)
- Провесс (node)
- Поток (встроенный модуль worker_threads)
- Песочница (vm.createContext, vm.Script)
- Программная абстракция (объект или замыкание)

Links

Node https://nodejs.org/en/about/releases/
https://nodejs.org/en/blog/
https://node.green/

Deno https://github.com/denoland/deno
https://youtu.be/z6JRlx5NC9E

Deno

- Безопасность:
  файловая система, сеть, окружение
- V8, TypeScript
- Rust вместо C++
- Tokio (event loop, I/O scheduler)
- Встроенный менеджер пакетов

Node.js

Готовность ноды для серьезных систем

Проблемы ноды

- Безопасность, заражения, зависимости
- Потерянные ошибки, утечки, перезапуски
- Асинхронность и стектрейс

Перспективы платформы

v8 Serialization API

v8.serialize, v8.deserialize

v8.Serializer, v8.Deserializer

Race Condition

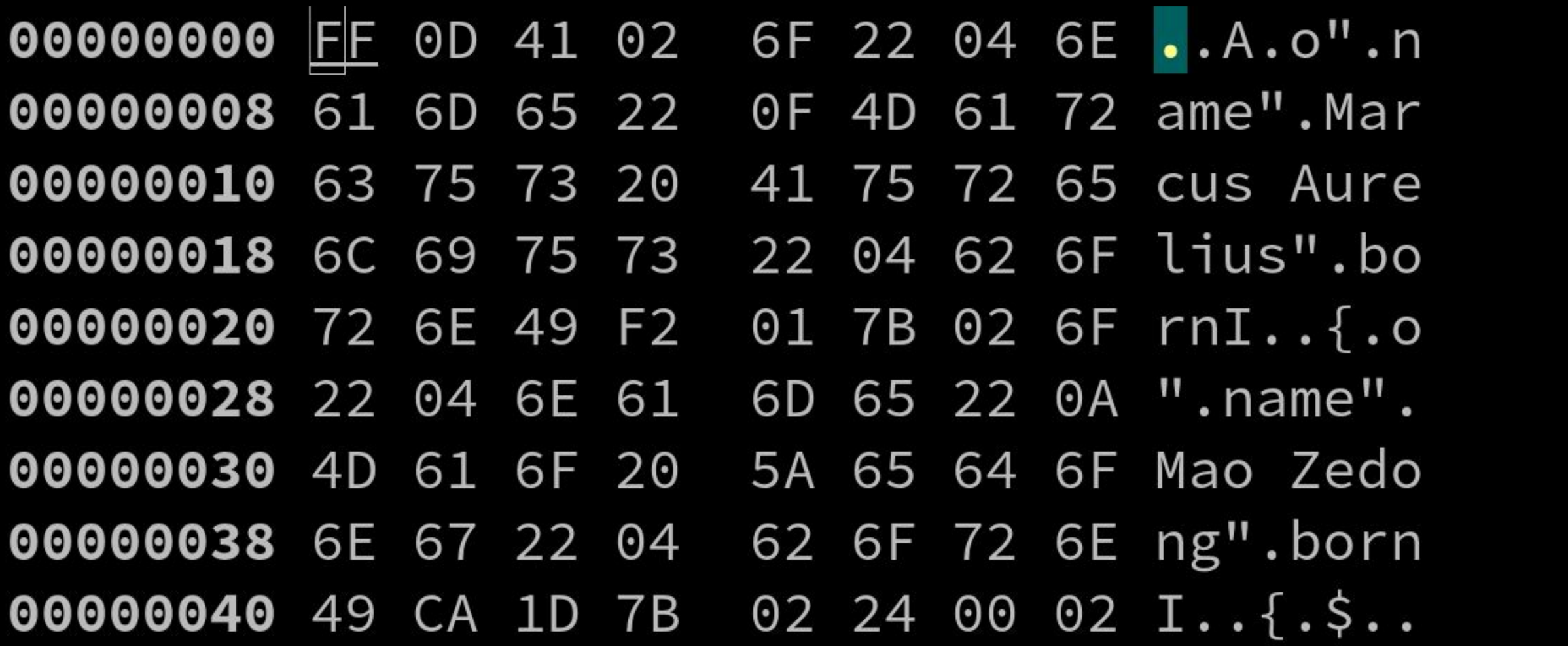
```
const v8 = require('v8');

const dataset = [
  { name: 'Marcus Aurelius', born: 121 },
  { name: 'Mao Zedong', born: 1893 },
];

const v8Data = v8.serialize(dataset);

const obj = v8.deserialize(v8Data);
```

v8 Serialization



vm Sandboxing

vm.Script, vm.runInContext and v8::Context

vm.createContext

```
const vm = require('vm');

const sandbox = {
  console: new Logger(),
  require: wrap(require),
  application: new Application(),
};

sandbox.global = sandbox;
vm.createContext(sandbox);
```

vm.Script

```
const vm = require('vm');
const fs = require('fs');

const code = await fs.readFile(fileName, 'utf8');
const src = `'use strict';\n context => ${code}`;

const script = new vm.Script(src);
script.runInContext(sandbox, { timeout: 5000 });
```

vm.Script

```
const options = {
  timeout: 5000,
  displayErrors: false, // default: true
  breakOnSigint: true,  // default: false, CTRL+C
};

script.runInContext(sandbox, options);
const object = { /* global */ };
script.runInNewContext(object, options);
script.runInThisContext(options);
```

ES.Next
ECMAScript 2020

ES.Next features

- Приватные и статические поля
- Изменения в Array, Object, String
- Atomics и SheredArrayBuffer
- Коллекции Set, Map, WeakSet, WeakMap
- globalThis
- Math

Operators

Rest

```
const f = (a, b, ...array) => {};
const g = ({ a, b, ...array }) => {};
const { name, ...rest } = obj;
```

Spread

```
f(a, b, ...array);
const obj2 = { name, ...obj1 };
const clone = { ...obj };
```

Operators

Exponentiation

```
Math.pow(x, y)    x ** y    x **= y    x = x ** y
```

Optional chaining

```
const spqr = {
  emperor: { name: 'Marcus' }
};
console.log(spqr.emperor?.name);
console.log(spqr.president?.name);
```

Operators

Exponentiation

```
Math.pow(x, y)    x ** y    x **= y    x = x ** y
```

Optional chaining (still waiting v8 8.x in Node.js)

```
const spqr = {
  emperor: { name: 'Marcus' }
};
console.log(spqr.emperor?.name);
console.log(spqr.president?.name);
```

Asynchronous iterable contract

Symbol.iterator

```
iterable[Symbol.iterator]()
```

Symbol.asyncIterator

```
asyncIterable[Symbol.asyncIterator]()
```

Try...catch

```
try {
  throw new Error('message');
} catch {
  console.log('no arguments catched');
}
```

Function

```
((a, b) => {
    const c = a + b; // hello there
    return c;
}).toString()
"(a, b) => {
    const c = a + b; // hello there
    return c;
}"
```

Promise.finally

```
new Promise(executor)
  .then(onFulfilled[, onRejected])
  .catch(onRejected)
  .finally(onFinally);
```

Promise.allSettled

```
const p1 = Promise.resolve('p1');
const p2 = new Promise((resolve, reject) => {
  setTimeout(resolve, 1000, 'p2');
});
const p3 = new Promise((resolve, reject) => {
 setTimeout(reject, 100, 'p3');
});

Promise.all([p1, p2, p3]).then(values => {
 console.log(values);
});
```

Promise.allSettled

```
Promise.all([p1, p2, p3]).then(values => {
 console.log(values);
});
```

```
node:26549) UnhandledPromiseRejectionWarning: p3
(node:26549) UnhandledPromiseRejectionWarning: Unhandled promise
rejection. This error originated either by throwing inside of an
async function without a catch block, or by rejecting a promise
which was not handled with .catch(). (rejection id: 1)
(node:26549) [DEP0018] DeprecationWarning: Unhandled promise
rejections are deprecated. In the future, promise rejections that
are not handled will terminate the Node.js process with
a non-zero exit code.
```

Promise.allSettled

```
const p1 = Promise.resolve('p1');
const p2 = new Promise((resolve, reject) => {
  setTimeout(resolve, 1000, 'p2');
});
const p3 = new Promise((resolve, reject) => {
  setTimeout(reject, 100, 'p3');
});

Promise.allSettled([p1, p2, p3]).then(values => {
  console.log(values);
});
```

Promise.allSettled

```
Promise.allSettled([p1, p2, p3]).then(values => {
  console.log(values);
});

[
  { status: 'fulfilled', value: 'p1' },
  { status: 'fulfilled', value: 'p2' },
  { status: 'rejected', reason: 'p3' }
]
```

Состояние гонки

В параллельном и асинхронном программировании

JavaScript
fwdays

Web Locks API in node.js & browser

Timur Shemsedinov

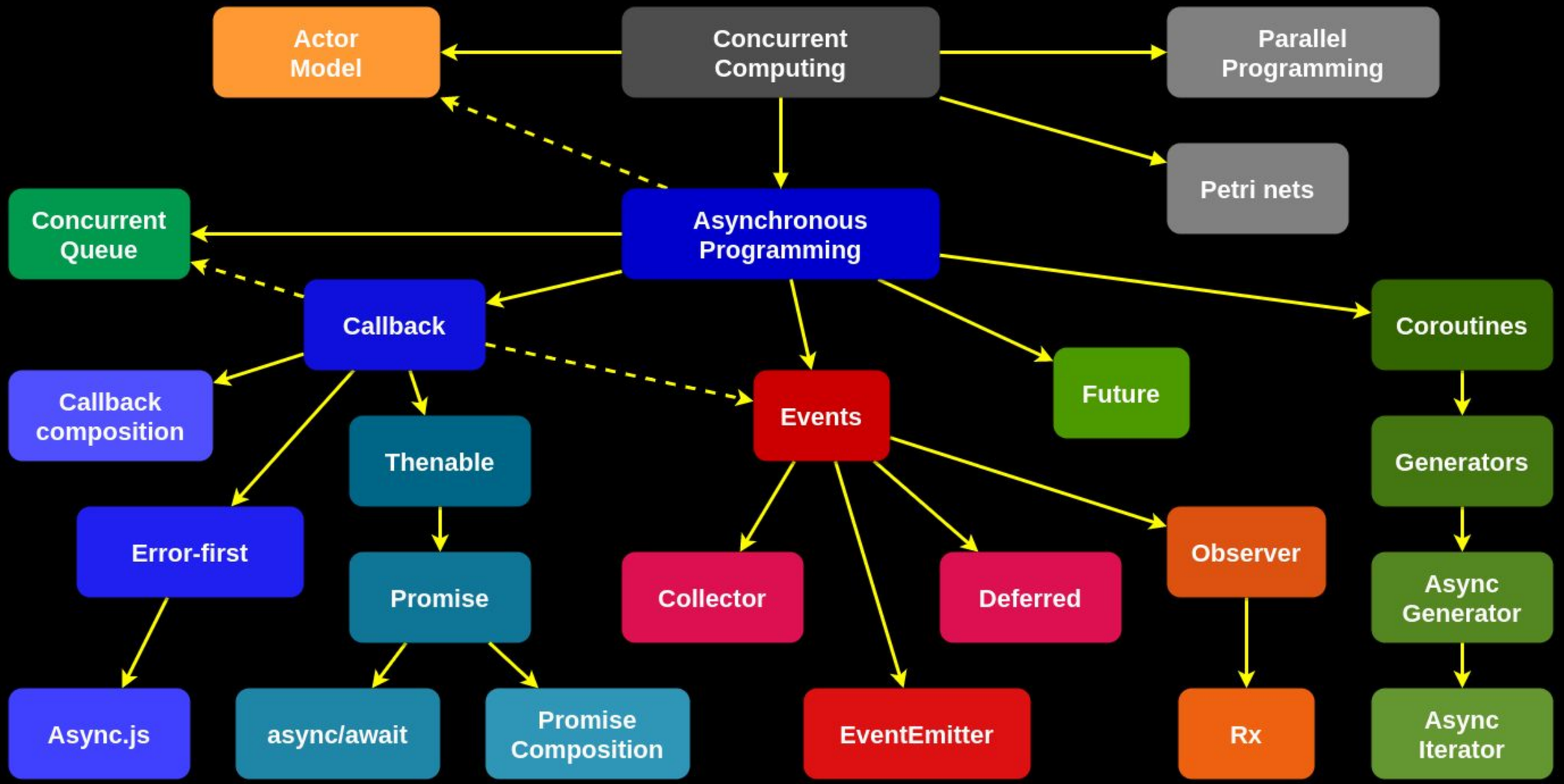
Chief Technology Architect at Metarhia
Lecturer at KPI

Why do we need Web Locks API ?

- Do you know what is
  mutex, locks, critical section, race condition,
  parallel programming at all?

- Congrats!
  It’s is very likely that
  all your JavaScript code broken)))

Concurrent Computing



Concurrency Problems

- Race condition
- Deadlock
- Livelock
- Resource starvation
- Resource leaks

Synchronization Primitives

Semaphore
Binary semaphore
Counting semaphore
Condition variable
Spinlock

Mutex
Timed mutex
Shared mutex
Recursive mutex
Monitor
Barrier

Links

Spec: wicg.github.io/web-locks
MDN: developer.mozilla.org/en-US/docs/Web/API/Web_Locks_API
Implementation: github.com/metarhia/web-locks
Examples:
  github.com/HowProgrammingWorks/RaceCondition
  github.com/HowProgrammingWorks/Semaphore
  github.com/HowProgrammingWorks/Mutex
Async prog: habr.com/ru/post/452974/

Questions?

github.com/tshemsedinov
https://youtube.com/TimurShemsedinov
github.com/HowProgrammingWorks/Index

Весь курс по ноде (>35.5 часов)
https://habr.com/ru/post/485294/

t.me/HowProgrammingWorks
t.me/NodeUA

timur.shemsedinov@gmail.com