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)))

Why do we need Web Locks API ?

Metarhia/NodeUA - Node.js Ukraine Community

Дописал базовую версию реализации Web Locks API для node.js, все для вас, 2 ночи сидел. Присоединяйтесь к альфа-тестированию, доработке, оптимизации, ставьбе звезды: https://github.com/metarhia/web-locks

> **GitHub**
> **metarhia/web-locks**
> Web Locks API. Contribute to metarhia/web-locks development by creating an account on GitHub.

5427 10:23 AM

Metarhia/NodeUA - Node.js Ukraine Community

Внезапно оказалось, что не все поняли, зачем нам Web Locks API, ну писали мы без него годами на JS и TS и ничего, все ж работало...

Anonymous Poll

70% Чем-то задним чую, что они нужны, нужно больше инфы

10% Та ладно, на моем компе все работает

10% В JavaScript и Node.js не может быть состояний гонки, все однопоточное же

10% Я с Java или C++ (или загадать свой вариант) ушел, чтоб с этим не иметь дело, а тут...

547 votes

3879 12:09 PM

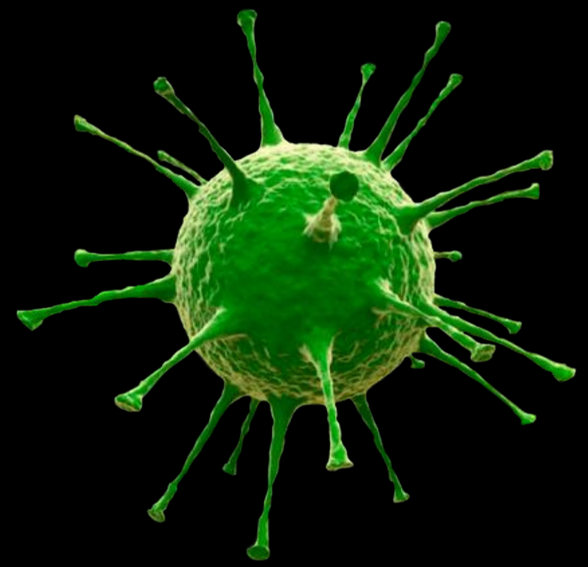
Who is at Risk?

Parallel programming

- Threads / Workers, SharedArrayBuffer, Atomics
- Mutex, Semaphore, Locks other primitives

Asynchronous programming

- Timers, I/O, events, DOM, fetch
- callbacks, Promise, async/await



Who is NOT at Risk?

Functional programming

- Pure functions, no side effects, immutability
- No transaction scripts, no imperative OOP

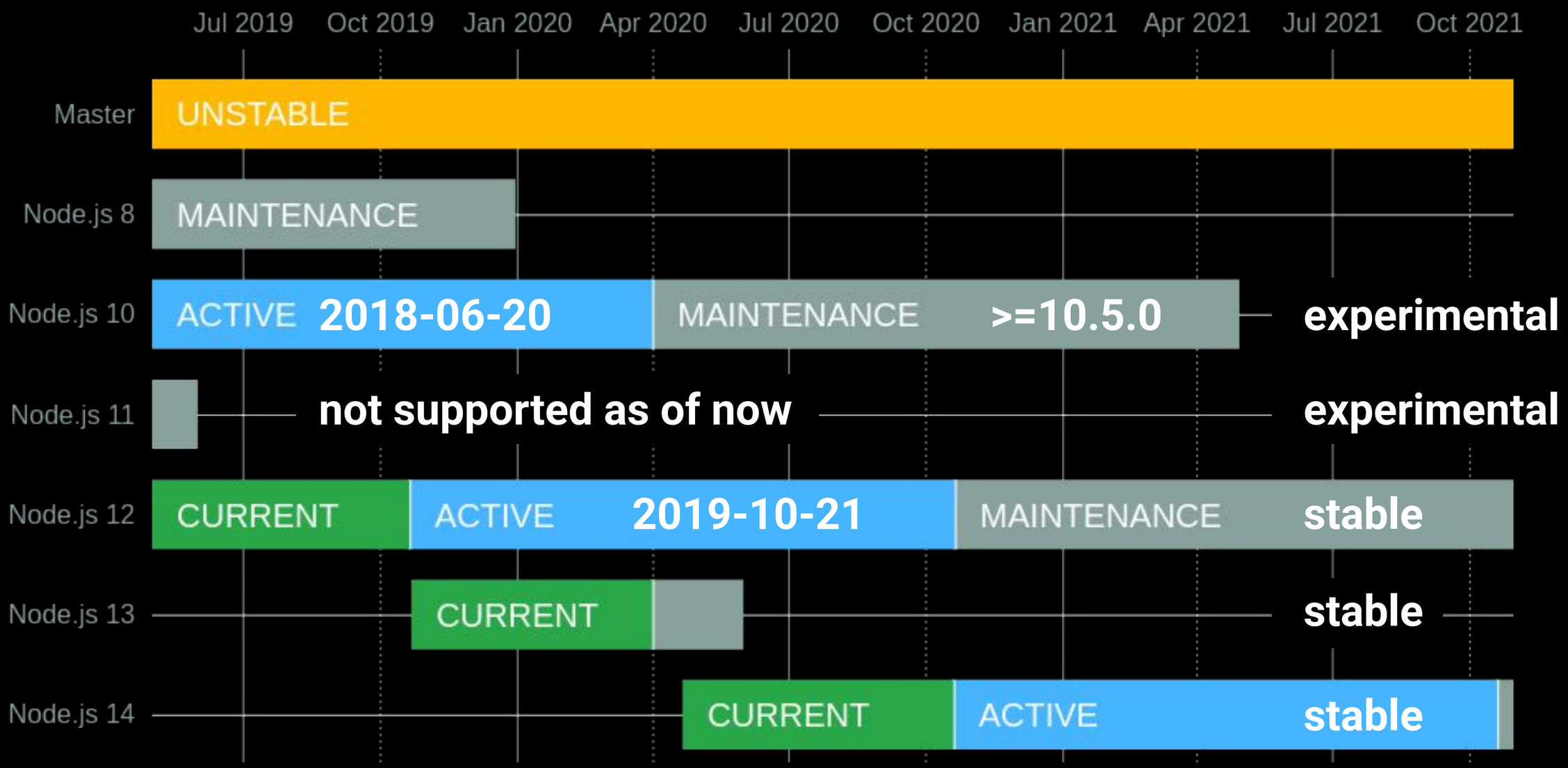
Specialized data structures

- Lock-free data structures, wait-free algorithms,
- Conflict-free data structures, immutable struct

Web Workers API in browsers

Safari 4	2009 Jan 8	(v14, iOS v13.3)
Firefox 3.5	2009 Jun 30	(v74, android v68)
Android 2.1	2009 Oct 26	(v2.2 - 4.3, v4.4 - 80)
Chrome 4	2010 Jan 29	(v80, android v80)
Opera 11.5	2011 Jan 28	(v66, android v46)
IE 10	2012 Sep 4	(v11)
Edge 12	2015 Jul 29	(v80)

Node.js releases with worker_threads



SharedArrayBuffer support

Safari	flag v10.1-13	(disabled by default)
Firefox	flag v57-76	(disabled by default)
Android and Chrome mob		(not supported)
Chrome	flag v60-67	supported v68-80
Opera	flag v47-63	supported v64-66
Edge	flag v16-18	supported v79-80

JavaScript Atomics support

Safari	v10.1-11, v11.1-13
Firefox	flag v46-54, v55-56, flag v57-74
Chrome	v60-62, v63-67, v68-80
Edge	v16, v17-18, v79-80

How Race Conditions in single threaded JS are possible?

Concurrency Problems

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

It works on my machine

javascruptissinglethreaded

Don't worry, everything will be random

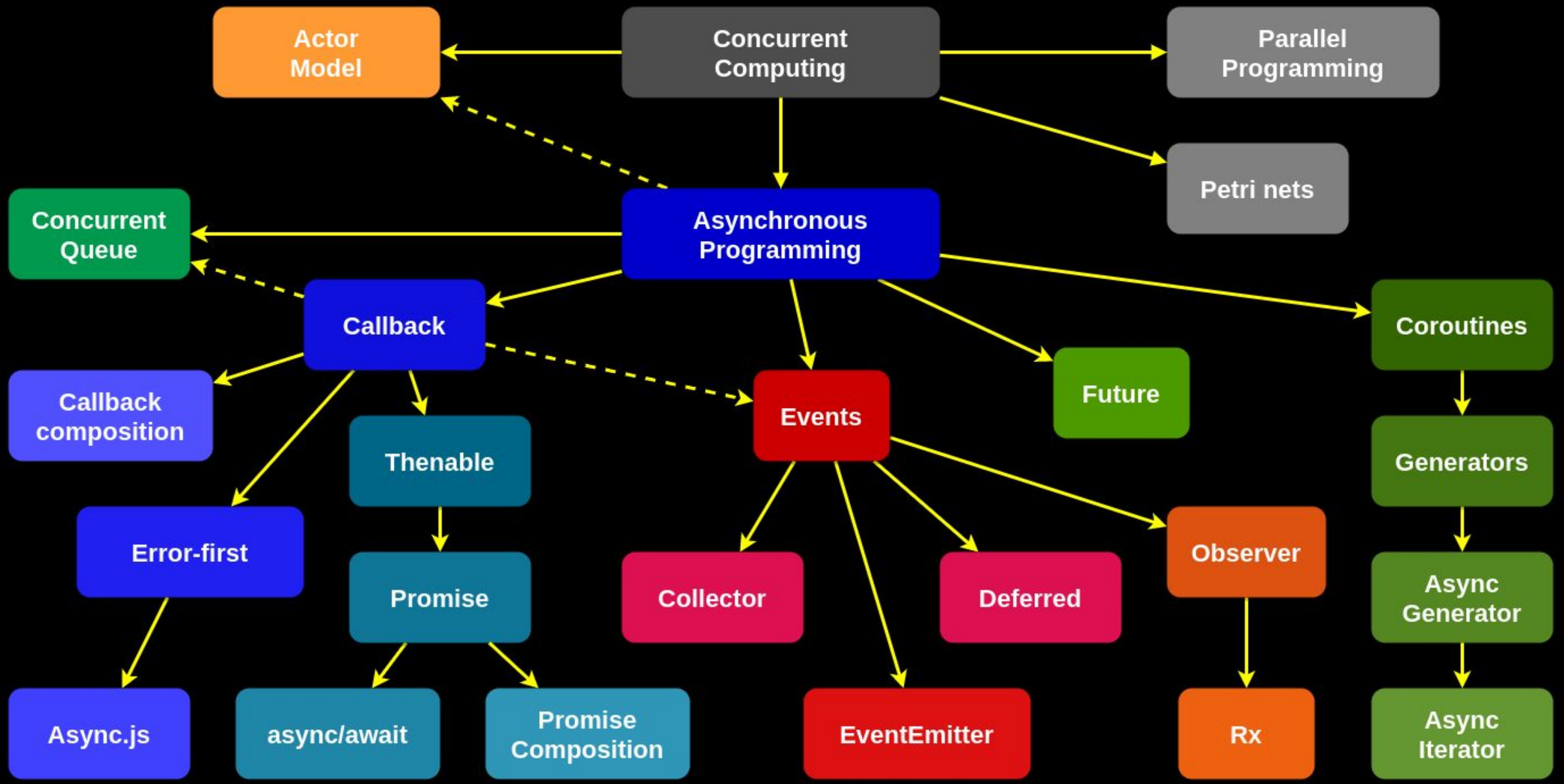
nodejsissinglethreaded

I have paws

Promises

async/await

Concurrent Computing



Race Condition

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  async move(dx, dy) {
    this.x = await add(this.x, dx);
    this.y = await add(this.y, dy);
  }
}
```

Race Condition

```
const random = (min, max) => Math
  .floor(Math.random() * (max - min + 1)) + min;

const add = (x, dx) => new Promise(resolve => {
  setTimeout(() => {
    resolve(x + dx);
  }, random(20, 100));
});
```

Race Condition

```
const p1 = new Point(10, 10);
console.log(p1);

p1.move(5, 5);
p1.move(6, 6);
p1.move(7, 7);
p1.move(8, 8);

setTimeout(() => {
  console.log(p1);
}, 1000);
```

Race Condition

Initial

```
Point { x: 10, y: 10 }
```

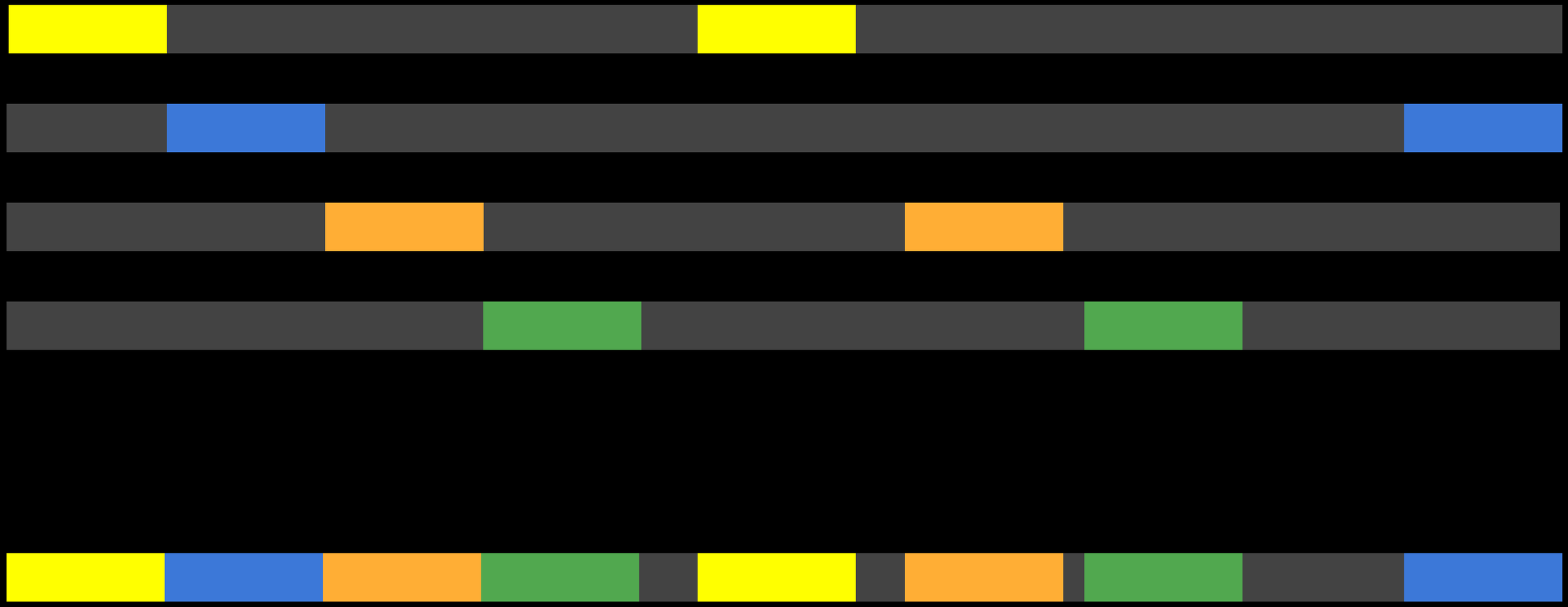
Expected

```
Point { x: 36, y: 36 }
```

Actual

```
Point { x: 18, y: 25 }
```

Race Condition



Possible Solutions

- Synchronization
  - Resource locking
- Special control flow organization
- Queuing theory
- Actor model
- Use DBMS transactions
- Specialized data structures

Synchronization Primitives

Semaphore
Binary semaphore
Counting semaphore
Condition variable
Spinlock

Mutex (and locks)
Timed mutex
Shared mutex
Recursive mutex
Monitor
Barrier

Semaphore

```
class Semaphore {
  constructor()
  enter(callback)
  leave()
}
semaphore.enter(() => {
  // do something
  semaphore.leave();
});
```

```
class Semaphore {
  constructor()
  async enter()
  leave()
}
await semaphore.enter();
// do something
semaphore.leave();
```

github.com/HowProgrammingWorks/Semaphore

Mutex

```
class Mutex {
  constructor()
  async enter()
  leave()
}

await mutex.enter();
// do something with shared resources
mutex.leave();
```

https://github.com/HowProgrammingWorks/Mutex

Resource Locking

```
class Lock {
  constructor() {
    this.active = false;
    this.queue = [];
  }

  leave() {
    if (!this.active) return;
    this.active = false;
    const next = this.queue.pop();
    if (next) next();
  }
}
```

```
enter() {
  return new Promise(resolve => {
    const start = () => {
      this.active = true;
      resolve();
    };
    if (!this.active) {
      start();
      return;
    }
    this.queue.push(start);
  });
}
```

Resource Locking

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.lock = new Lock();
  }

  async move(dx, dy) {
    await this.lock.enter();
    this.x = await add(this.x, dx);
    this.y = await add(this.y, dy);
    this.lock.leave();
  }
}
```

Resource Locking

```
const p1 = new Point(10, 10);
console.log(p1);

p1.move(5, 5);
p1.move(6, 6);
p1.move(7, 7);
p1.move(8, 8);

setTimeout(() => {
  console.log(p1);
}, 1000);
```

Resource Locking

Initial

```
Point { x: 10, y: 10 }
```

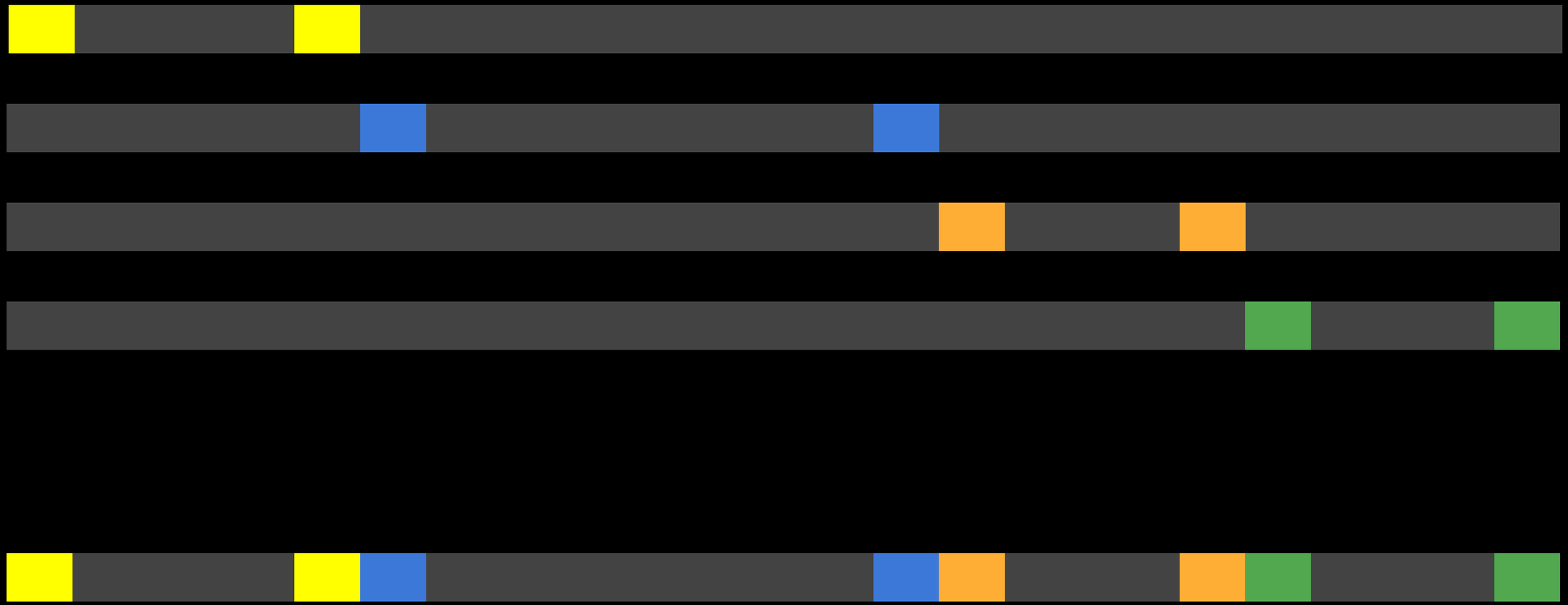
Expected

```
Point { x: 36, y: 36 }
```

Actual

```
Point { x: 36, y: 36 }
```

Resource Locking



Asynchronous Res Locks

Real-life Example

Warehouse API

- Check balances
- Ship goods
- Lock balances

github.com/HowProgrammingWorks/RaceCondition

Web Locks API

```
locks.request('resource', opt, async lock => {
  if (lock) {
    // critical section for `resource`
    // will be released after return
  }
});
```

https://wicg.github.io/web-locks/

Web Locks: await

```
(async () => {
  await something();
  await locks.request('resource', async lock => {
    // critical section for `resource`
  });
  await somethingElse();
})();
```

Web Locks: Promise

```
locks.request('resource', lock => new Promise(
  (resolve, reject) => {
    // you can store or pass
    // resolve and reject here
  }
));
```

Web Locks: Thenable

```
locks.request('resource', lock => ({
  then((resolve, reject) => {
    // critical section for `resource`
    // you can call resolve and reject here
  })
}));
```

Web Locks: Abort

```
const controller = new AbortController();
setTimeout(() => controller.abort(), 2000);

const { signal } = controller;

locks.request('resource', { signal }, async lock => {
  // lock is held
}).catch(err => {
  // err is AbortError
});
```

Web Locks: Timeout

```
locks.request(
  'resource', { timeout: 2000 }, async lock => {
    // lock is held
  }
).catch(err => {
  // err is TimeoutError
});
```

Web Locks for Node.js

github.com/nodejs/node/issues/22702
Open

github.com/nodejs/node/pull/22719
Closed

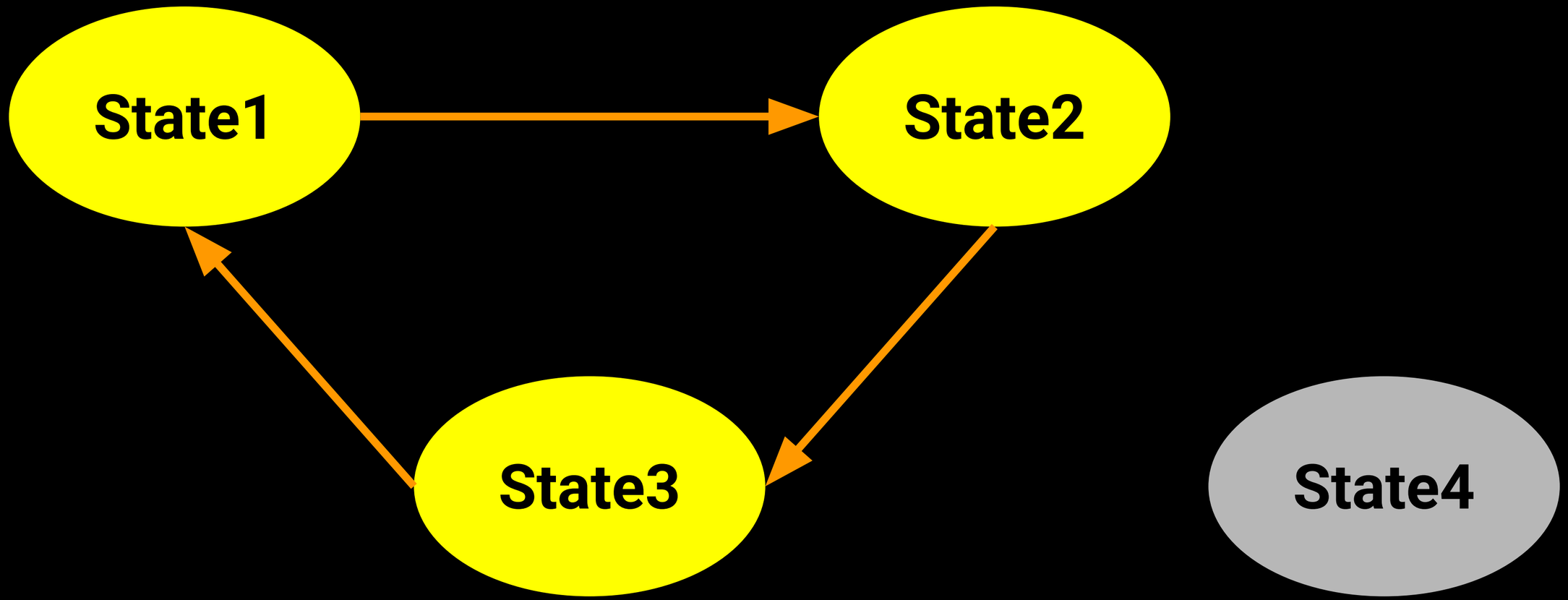
Safe data structures

- Low-level structures
  e.g. Register, Counter, Buffer, Array, Lists, etc.
- Abstract structures
  e.g. Queue, Graph, Polyline, etc.
- Subject-domain classes
  e.g. Sensors, Payment, Biometric data, etc.
- Resources and handles
  e.g. Sockets, Connections, Streams, etc.

Deadlock

```
(async () => {
  await locks.request('A', async lock => {
    await locks.request('B', async lock => {
    });
  });
})(); (async () => {
  await locks.request('B', async lock => {
    await locks.request('A', async lock => {
    });
  });
})();
```

Livelock



Alternative Solutions

- Thread safe data structures
- Lock-free data structures
- Wait-free algorithms
- Conflict-free data structures
- Immutable data structures

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?

timur.shemsedinov@gmail.com
github.com/tshemsedinov
github.com/HowProgrammingWorks/Index

youtube.com/TimurShemsedinov (>180 lectures, >50 h about async.prog, >35.5 h about Node.js)

meetup.com/HowProgrammingWorks
meetup.com/NodeUA
t.me/HowProgrammingWorks
t.me/NodeUA