# Race conditions, Web Locks & Shared Memory in Node.js

github.com/HowProgrammingWorks


## Timur Shemsedinov

Chief Technology Architect at Metarhia
Lecturer at Kiev Polytechnic Institute

github.com/tshemsedinov

# Why do we need this problems?

- 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 ))))

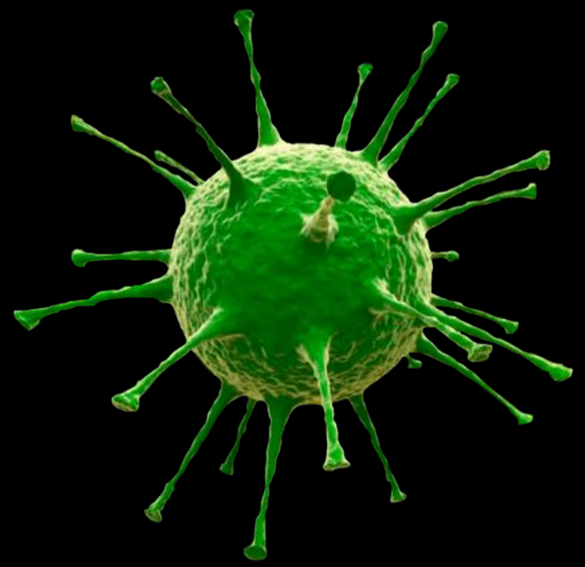
# Who is at Risk?

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

# How to use workers_threads

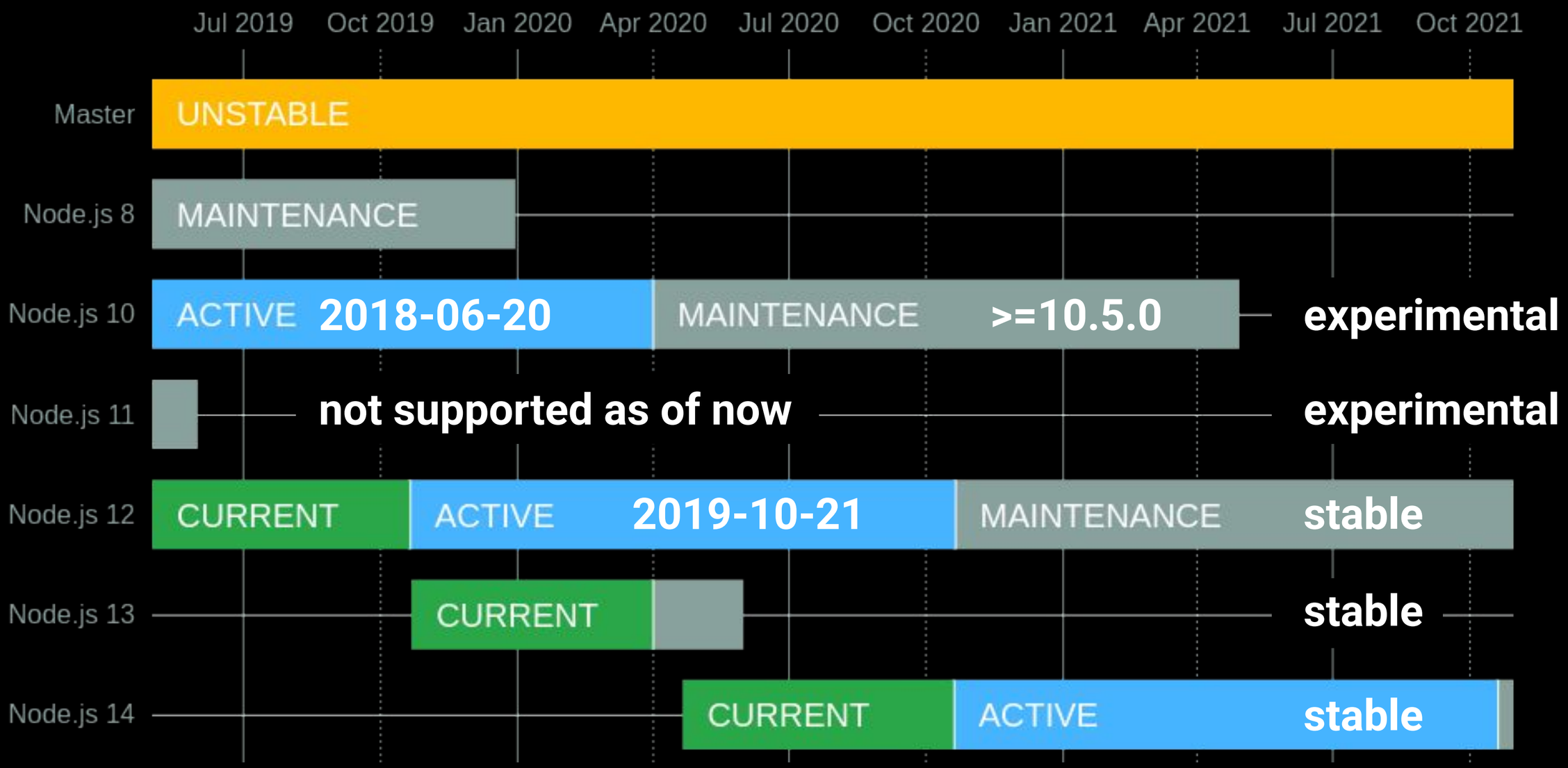
Node.js: The Road to Workers
Anna Henningsen
https://youtu.be/pO5a10YPQG4

A Crash Course on Worker Threads
Rich Trott
https://youtu.be/GRb-XQ5JRA8

# We are ready for Parallel programming

- Stable worker_threads and messaging API
  https://nodejs.org/api/worker_threads.html
- Atomics for Compare-and-Swap operations
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Atomics
- SharedArrayBuffer to share memory
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer

# Node.js releases with worker_threads



# 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) |

# 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

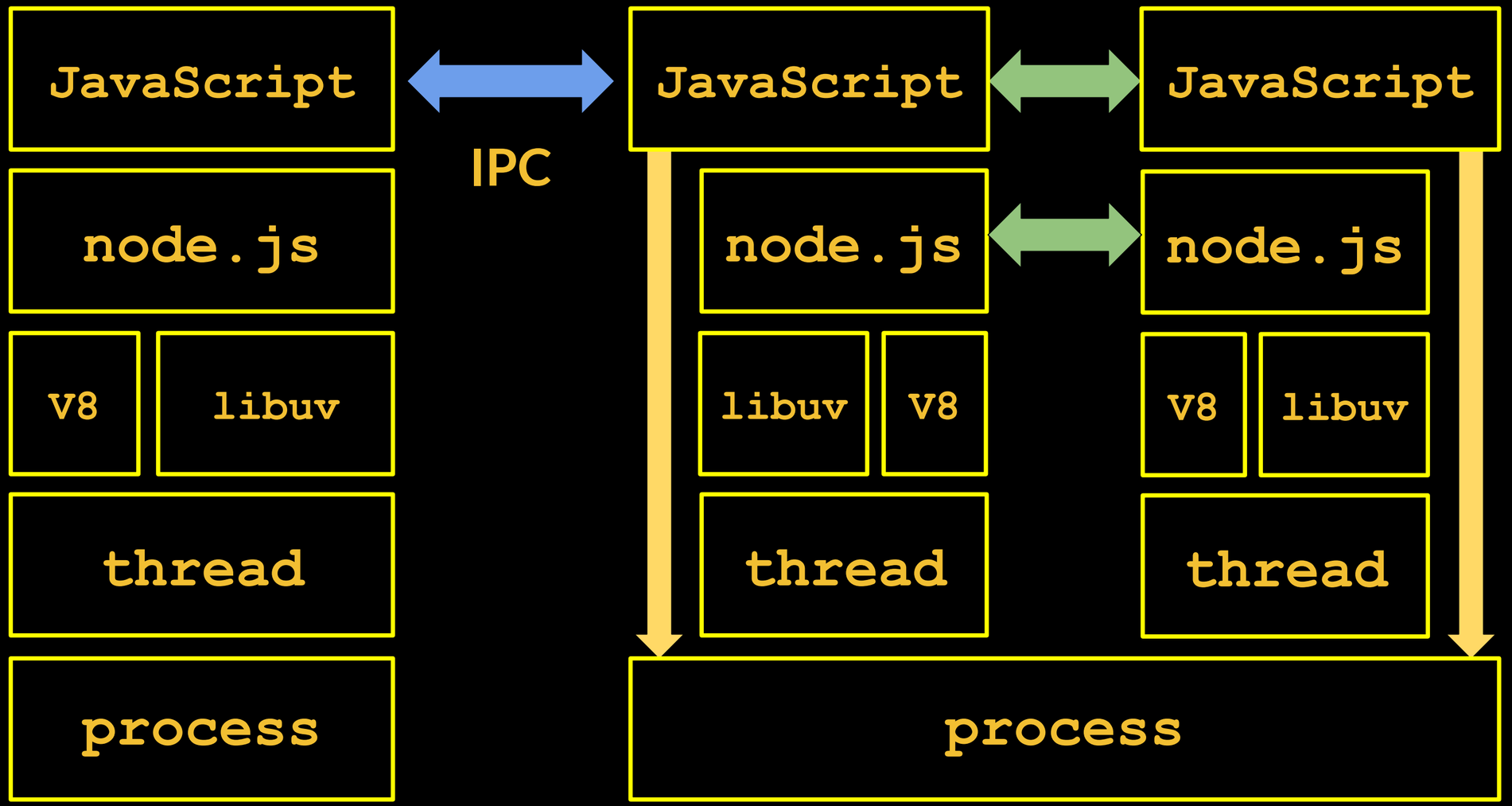
Supported started from Node.js 9

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

# 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 |

# Threads vs Processes



# We use threads for...

- CPU-intensive operations
  - Calculations and blocking business-logic
  - Cryptography, parsing, rendering, etc.
- Scaling applications to utilize CPU cores for I/O
  - For example thread per port
- Code isolation related to I/O operations
  - I/O Event loop and thread pool isolation

# Don’t use threads for...

Don’t run separate threads for:

- database queries
- external API calls
- timers and waiting for events, like schedulers
- everything with low intensive CPU and I/O

Don’t run large number of thread to prevent context switching

# Why Isolation?

- Unhandled exceptions
- Memory leaks and other resources leaks
- In-memory data structures
- Database and other connections
- Environment access, OS, IPC...
- Network descriptors, sockets, ports

# Let's combine worker threads With JS Atomics and SharedArrayBuffer

# Node.js worker_threads API

```
const threads = require('worker_threads');
const { Worker, isMainThread } = threads;

if (isMainThread) {
  const worker = new Worker(
    __filename,
    { workerData: {} }
  );
} else {
  const { parentPort } = threads;
}
```

# MessagePort API

```
worker.on('message', (...args) => {});
worker.on('error', err => {});
worker.on('exit', code => {});
worker.postMessage('Hello there!');

parentPort.postMessage('Hello there!');
parentPort.on('message', (...args) => {});

// MDN Specification
// https://developer.mozilla.org/en-US/docs/Web/API/MessagePort
```

# Wrap Shared Memory with OOP

```
class Point {
  constructor(buffer, offset) {
    this.data = new Int8Array(buffer, offset, 2);
  }
  get x() {
    return this.data[0];
  }
  set x(value) {
    this.data[0] = value;
  }
  // ...
```

# Wrap Shared Memory with OOP

```
// Shared Point instantiation
const buffer = new SharedArrayBuffer(64);
const point = new Point(buffer, 4);

// Now we can read/write fields an instance
point.x += 10;
point.y = 7;
const { x } = point;
```

# Atomics and SharedArrayBuffer

```
Int8Array,  Uint8Array,
Int16Array, Uint16Array,
Int32Array, Uint32Array
```

# ES2017 Atomics

```
prev = Atomics.add(array, index, value)
prev = Atomics.sub(array, index, value)
prev = Atomics.and(array, index, value)
prev = Atomics.or(array, index, value)
prev = Atomics.xor(array, index, value)
```

# Atomics.add(array, index, value)

```
   function add(array, index, value) {
1    const prev = array[index];
2    const sum = prev + value;
3    array[index] = sum;
4    return prev;
   }
```

# ES2017 Atomics (CAS)

```
stored = Atomics.store(array, index, value)
value = Atomics.load(array, index)
prev = Atomics.exchange(array, index, value)
prev = Atomics.compareExchange(
  array, index, expected, replacement
)
```

# ES2017 Atomics (notify/wait)

```
woken = Atomics.notify(array, index, count)
res = Atomics.wait(array, index, value, [timeout])

res <"ok" | "not-equal" | "timed-out">
array <Int32Array>
```

# Mutex

```
class Mutex {
  constructor(shared, offset = 0)
  enter()
  leave()
}

// https://github.com/HowProgrammingWorks/Mutex
```

# Sync Mutex contract

```
{
  mutex.enter();
  // do something
  // with shared resources
  // or data structures
  mutex.leave();
}
```

# Mutex constructor

```
constructor(shared, offset = 0) {
  this.lock = new Int32Array(shared, offset, 1);
  this.owner = false;
}

const threads = require('worker_threads');
const { workerData } = threads;
const mutex1 = new Mutex(workerData, offset);
```

# Mutex.enter with Atomics.wait

```
enter() {
  let prev = Atomics.exchange(this.lock, 0, LOCKED);
  while (prev !== UNLOCKED) {
    Atomics.wait(this.lock, 0, LOCKED);
    prev = Atomics.exchange(this.lock, 0, LOCKED);
  }
  this.owner = true;
}

// Example: 6-blocking.js
```

# Mutex.leave with Atomics.notify

```
leave() {
  if (!this.owner) return;
  Atomics.store(this.lock, 0, UNLOCKED);
  Atomics.notify(this.lock, 0, 1);
  this.owner = false;
}
```

# Async Mutex with callback contract

```
mutex.enter(() => {
  // do something
  // with shared resources
  // or data structures
  mutex.leave();
});
```

# Async Mutex with async/await contract

```
(async () => {
  await mutex.enter();
  // do something
  // with shared resources
  // or data structures
  mutex.leave();
})();
```

# Spinlock

```
enter() {
  return new Promise(resolve => {
    while (true) {
      let prev = Atomics.exchange(this.lock, 0, LOCKED);
      if (prev === UNLOCKED) break;
    }
    this.owner = true;
    resolve();
  });
}
```

# Spinlock with setTimeout

```
enter() {
  return new Promise(resolve => {
    const tryEnter = () => {
      let prev = Atomics.exchange(this.lock, 0, LOCKED);
      if (prev === UNLOCKED) {
        this.owner = true;
        resolve();
      } else {
        setTimeout(tryEnter, 0);
      }
    };
    tryEnter();
  });
}
```

# Asynchronous Mutex with messaging

```
constructor(messagePort, shared, offset = 0) {
  this.port = messagePort;
  this.lock = new Int32Array(shared, offset, 1);
  this.owner = false;
  this.trying = false;
  this.resolve = null;
  if (messagePort) {
    messagePort.on('message', kind => {
      if (kind === 'leave' && this.trying) this.tryEnter();
    });
  }                                  // Example: 8-async.js
}
```

# Asynchronous Mutex

```
enter() {
  return new Promise(resolve => {
    this.resolve = resolve;
    this.trying = true;
    this.tryEnter();
  });
}
```

# Asynchronous Mutex

```
tryEnter() {
  if (!this.resolve) return;
  let prev = Atomics.exchange(this.lock, 0, LOCKED);
  if (prev === UNLOCKED) {
    this.owner = true;
    this.trying = false;
    this.resolve();
    this.resolve = null;
  }
}
```

# Thread safe data structures

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

# 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

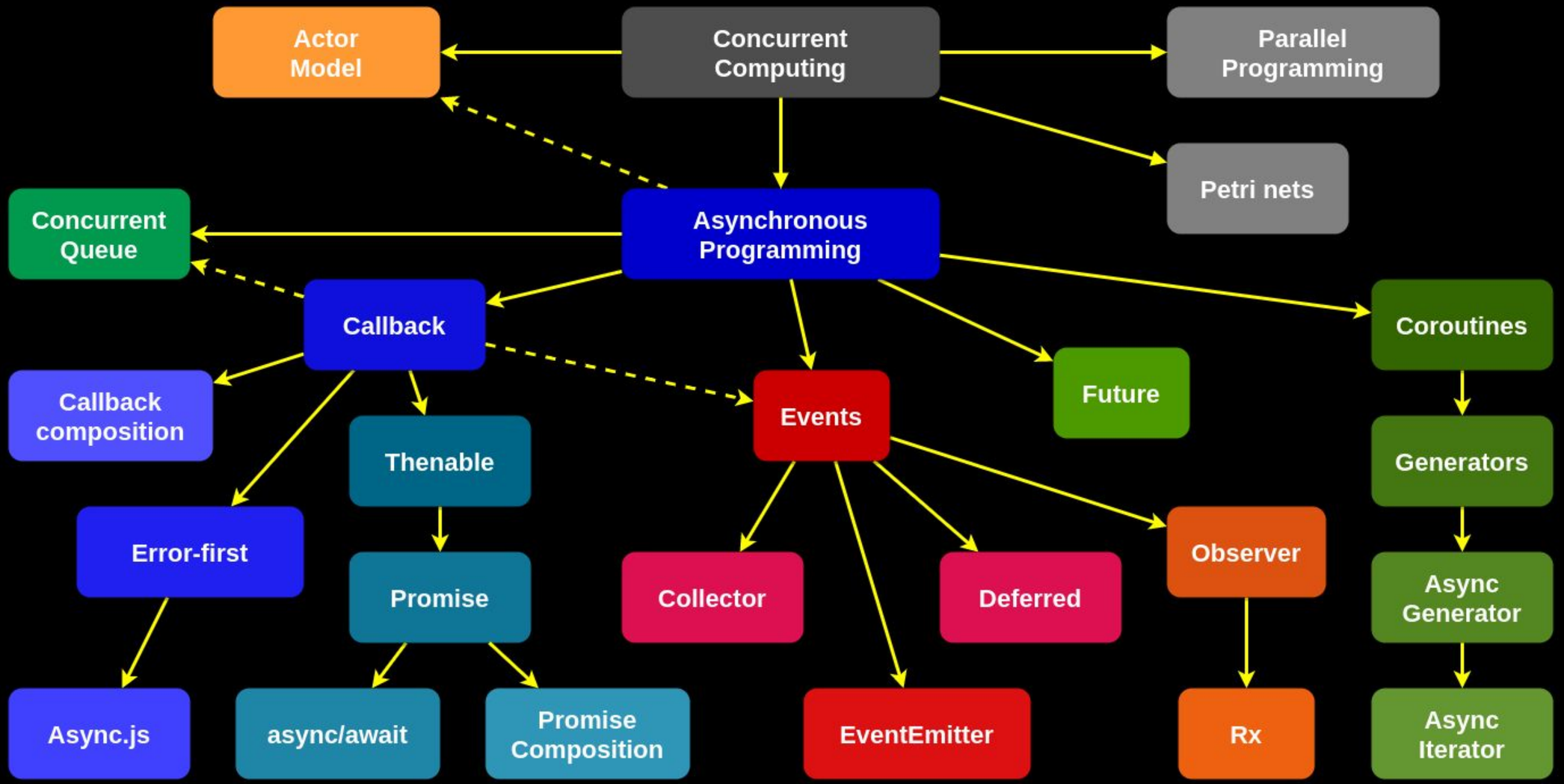
nodejsissinglethreaded

# 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 }
```

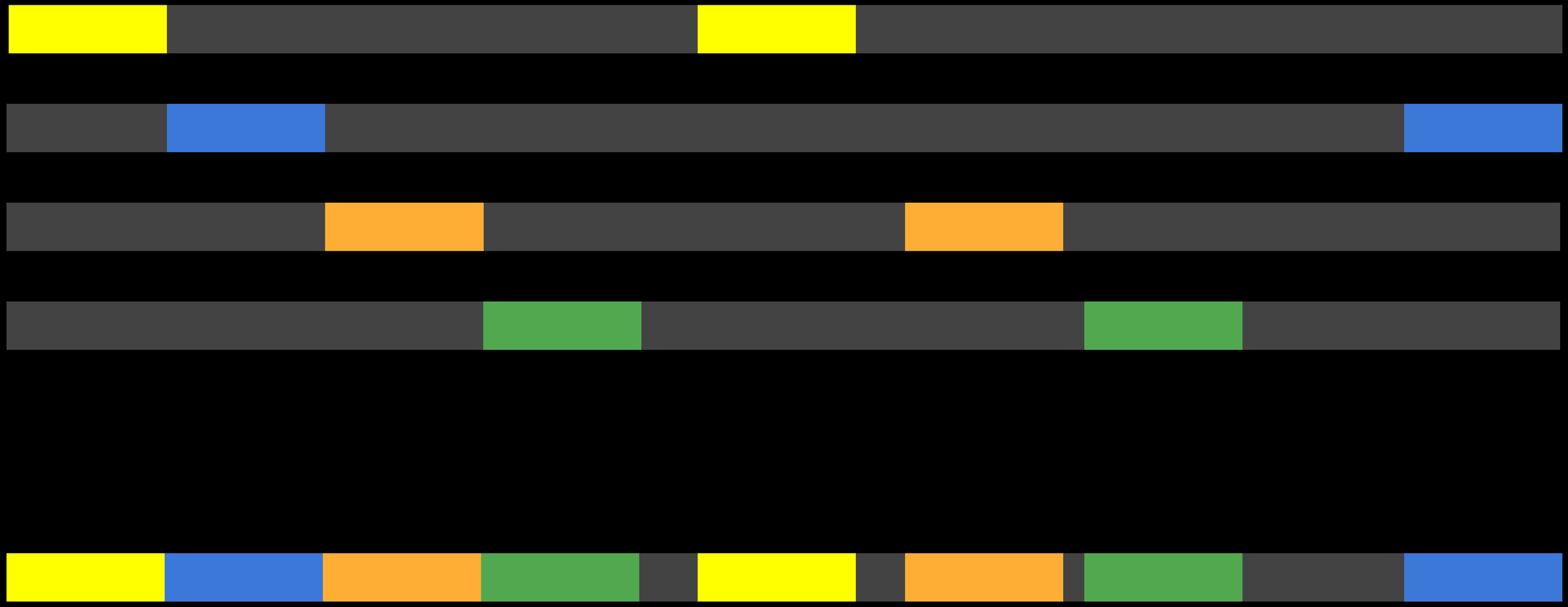
Expected

```
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 }
```

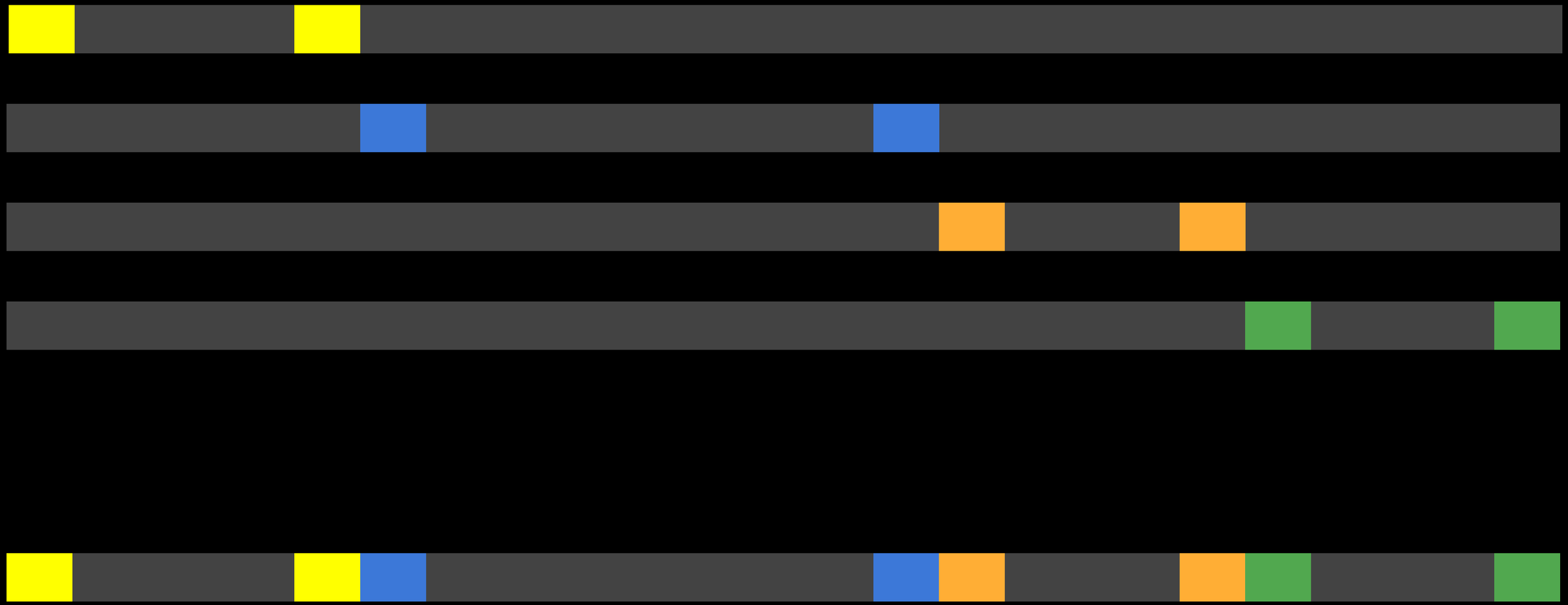
Expected

```
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

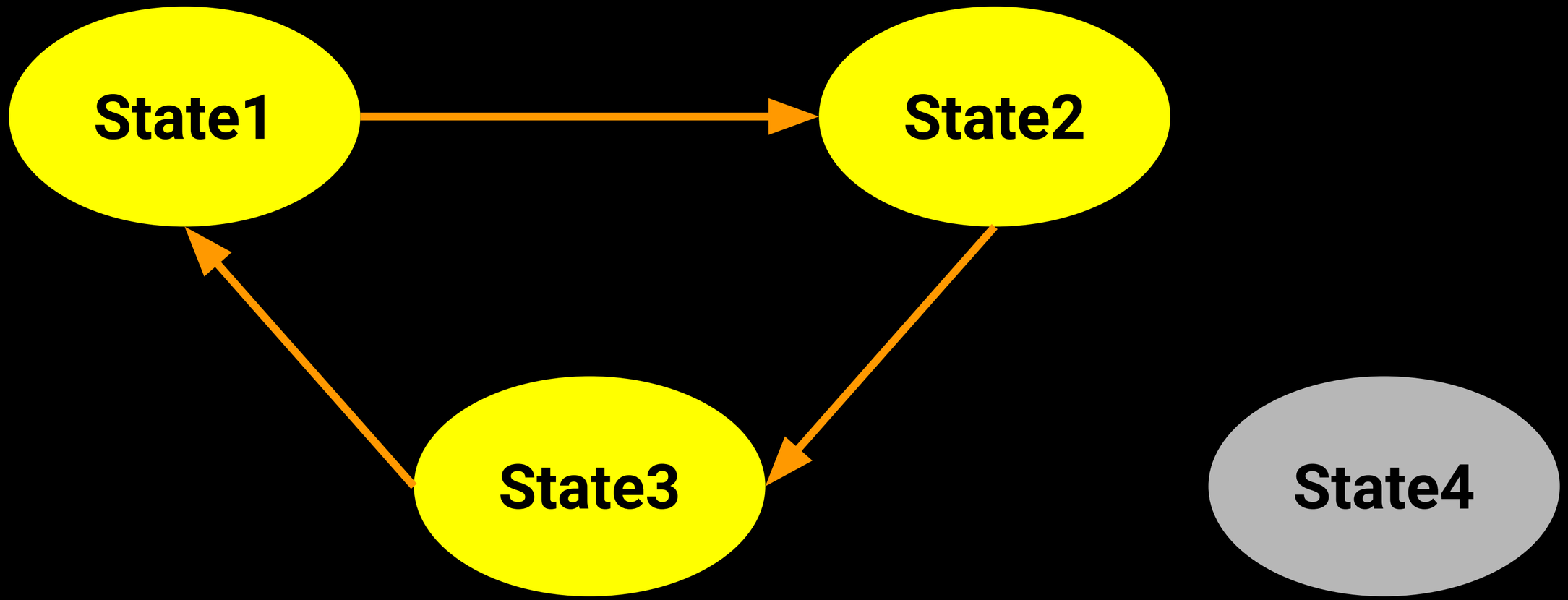
# Safe data structures

- 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/

# Contacts

github.com/tshemsedinov
youtube.com/TimurShemsedinov
github.com/HowProgrammingWorks
patreon.com/tshemsedinov
t.me/HowProgrammingWorks
t.me/NodeUA

timur.shemsedinov@gmail.com