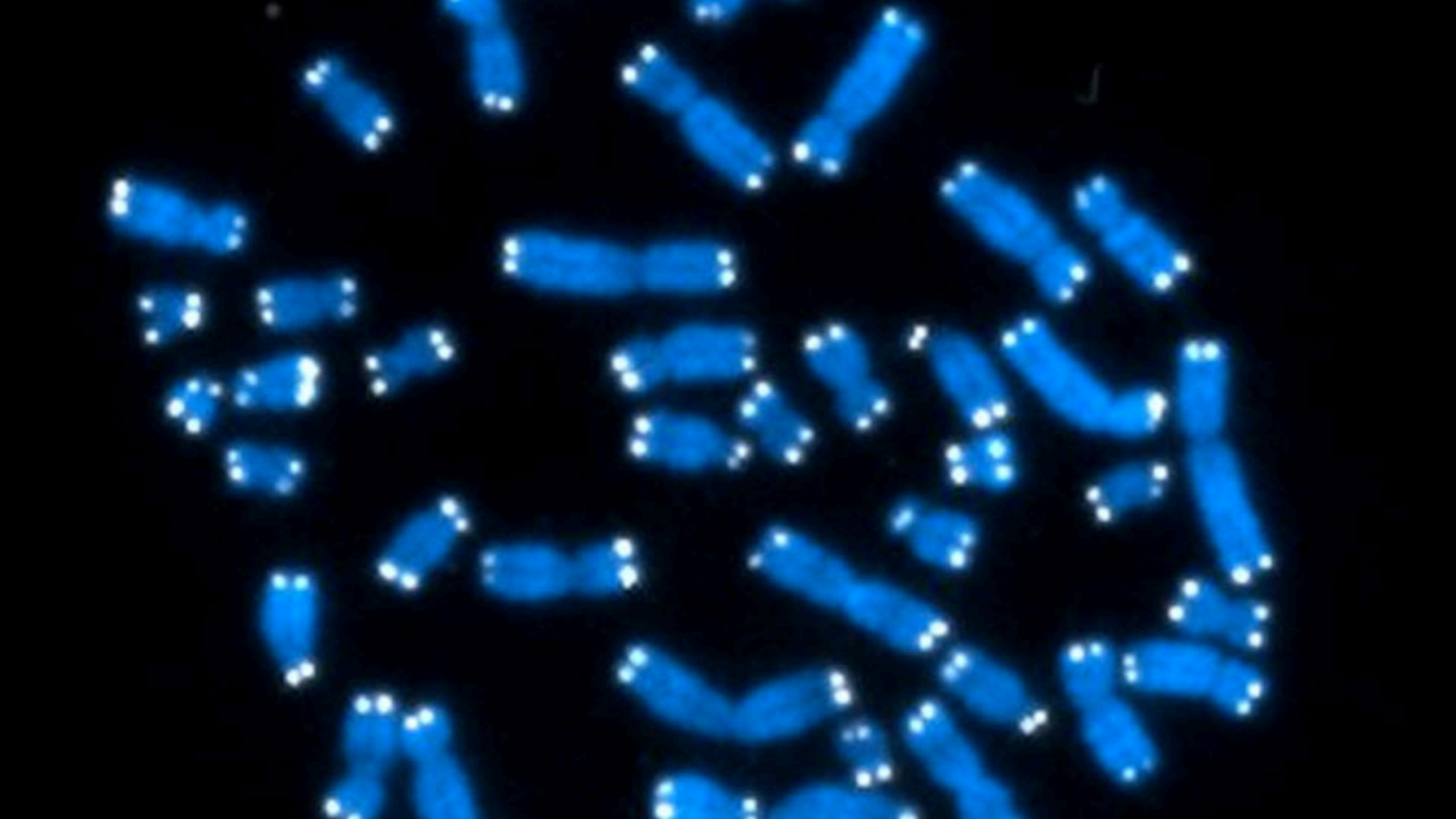

node
JS

Threads in Node

Cześć!

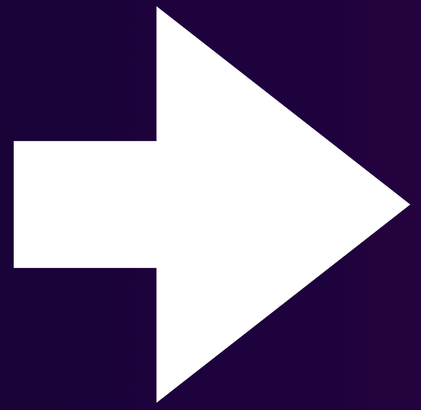
OLEKSANDR TRYSHCHENKO
TRYSHCHENKO.COM
OTRY.EU

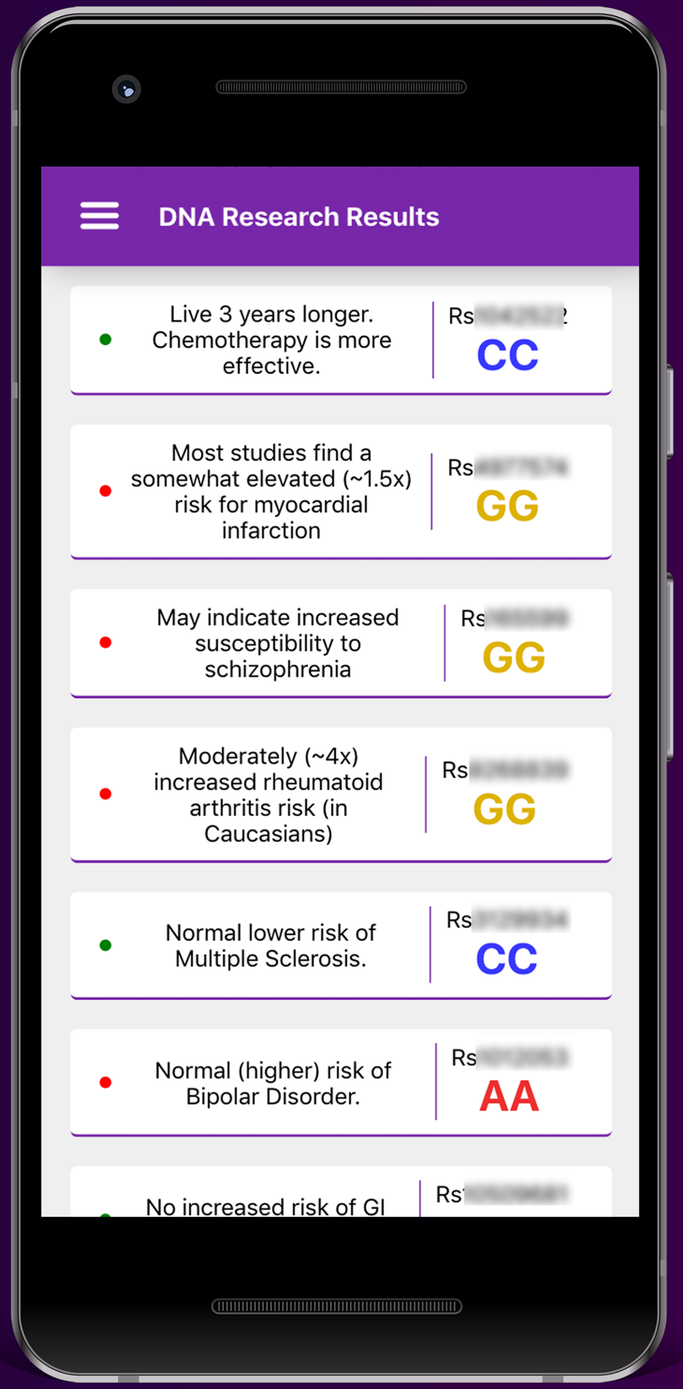



397-387


MiSeq™
MiSeq Reporter 2.1.43
illumina

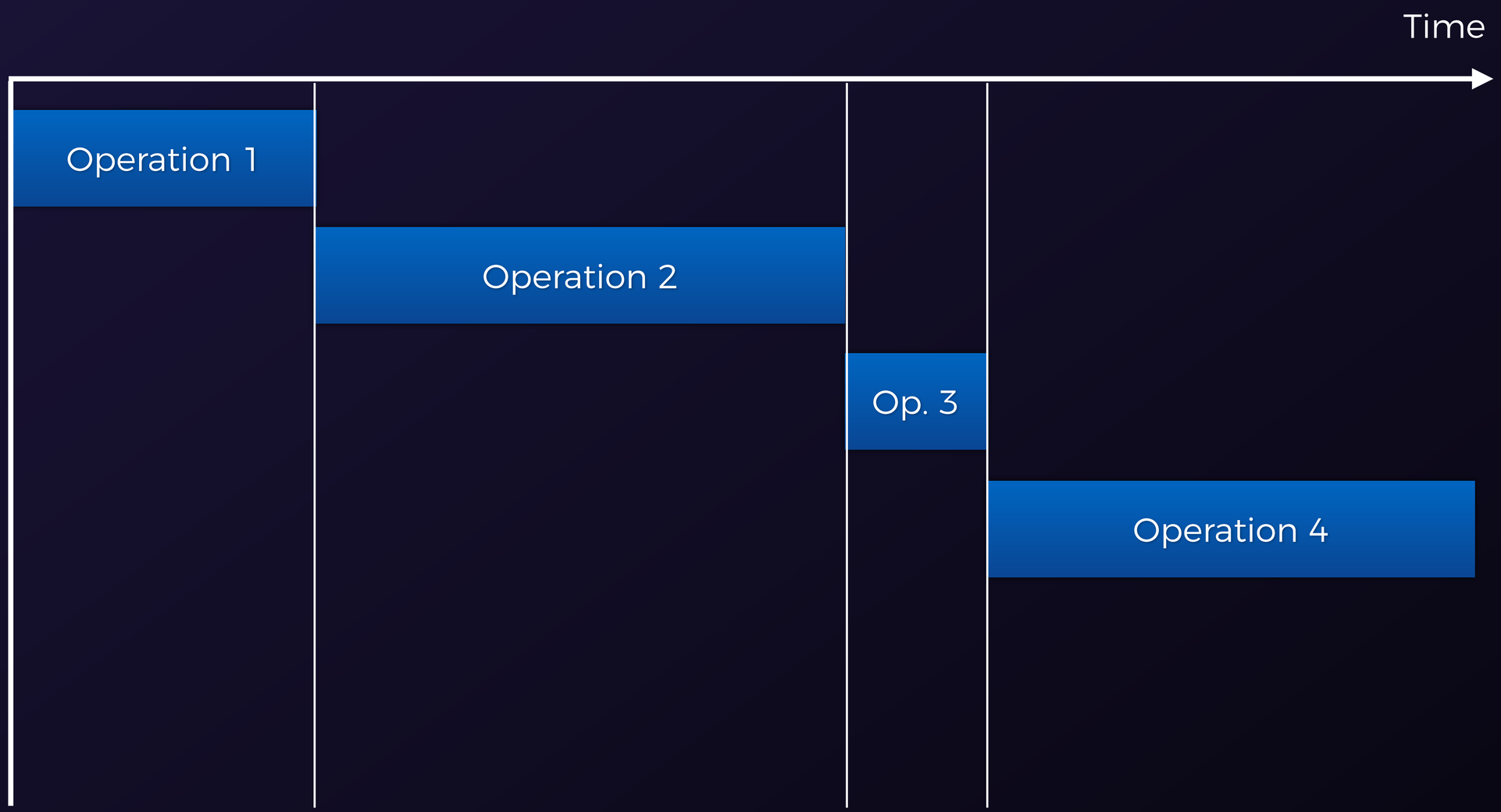
rs28508199	1	1852484	AG
rs2144687	1	1853394	TT
rs2250833	1	1854321	GG
rs2803316	1	1865298	AG
rs12758705	1	1873625	AG
rs2803329	1	1874581	GG
rs2262190	1	1875267	GG
rs2748975	1	1886519	CC
rs3820011	1	1888193	AC
rs2803291	1	1892325	CC
rs16824588	1	1900232	CT
rs3795283	1	1916587	GT
rs6661597	1	1926087	CC
rs6605081	1	1930754	CT
rs6656398	1	1941929	AG
rs13303016	1	1946591	AG
rs28603108	1	1950621	AG
rs28669111	1	1954066	CC
rs28437090	1	1959022	CC
rs41307846	1	1959699	--
rs2376803	1	1967954	CT
rs28706891	1	1970444	AA
rs3795277	1	1981118	AA
rs2803309	1	1987803	GG
rs1878745	1	1991014	AA
rs6605074	1	1992748	CC
rs2678939	1	2000592	AC
rs6663158	1	2004098	GG
rs2803310	1	2017297	CT
rs2459994	1	2024064	CC
rs12755035	1	2026361	AG
rs3128326	1	2026403	CT
rs884080	1	2026749	AG
rs7513222	1	2027901	GG
rs908742	1	2033256	GG
rs10910030	1	2035684	CT
rs12410859	1	2038732	CC
rs4648807	1	2040898	CT
rs4648808	1	2040936	CT





What are threads?

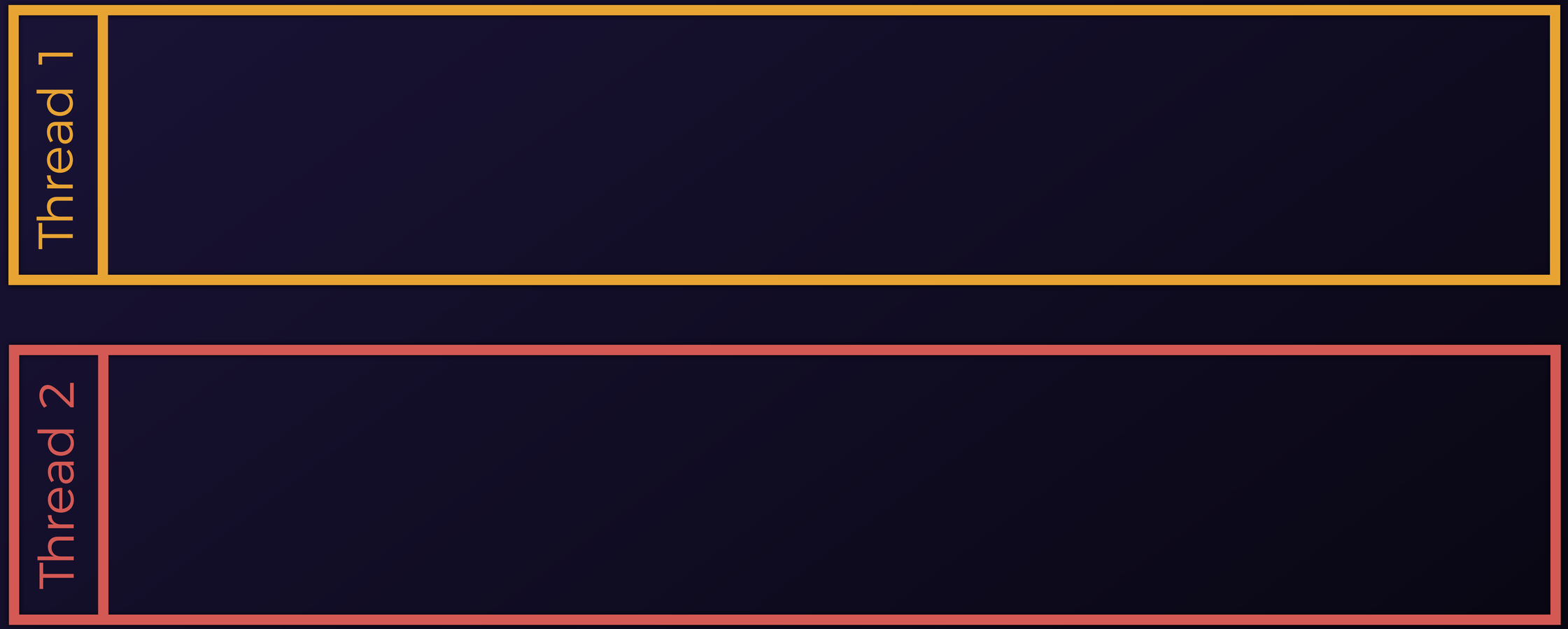
One thread



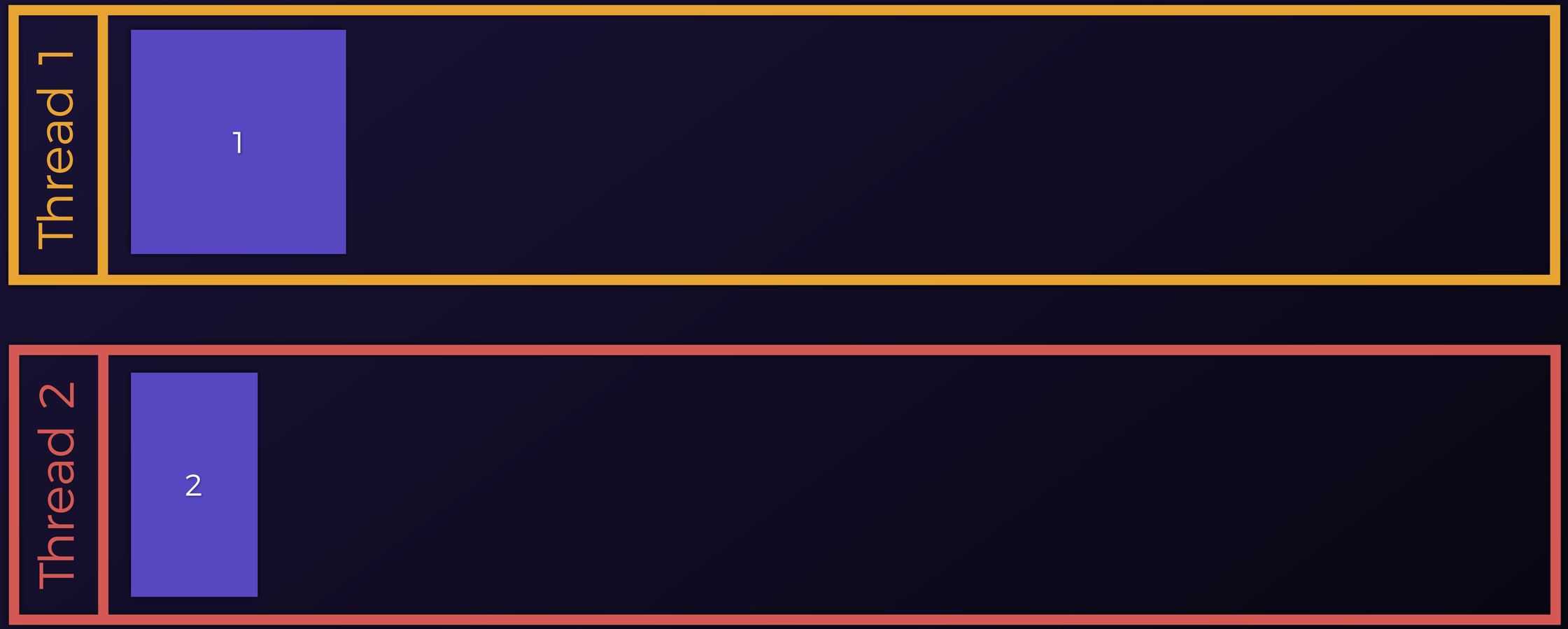
Two threads

Thread Pool

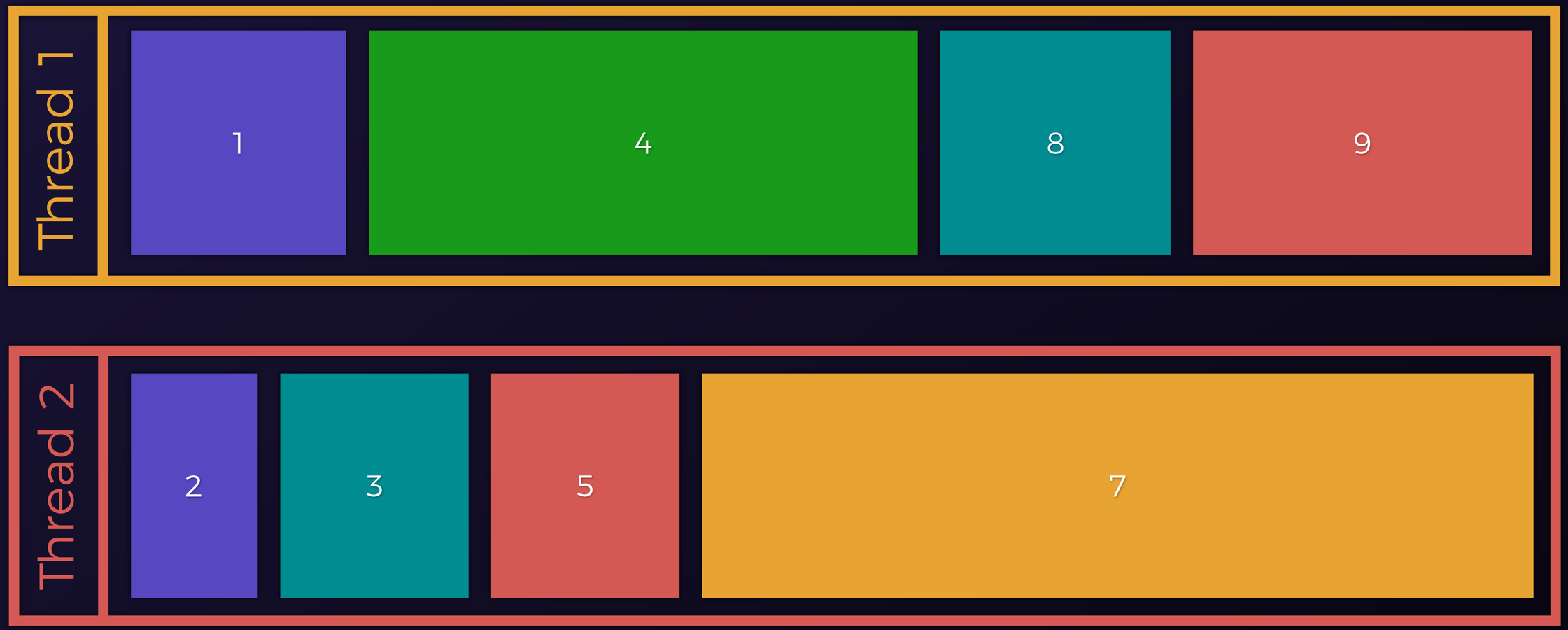
Thread pool

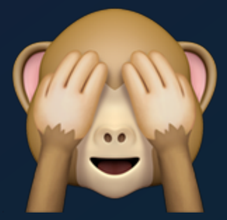


Thread pool

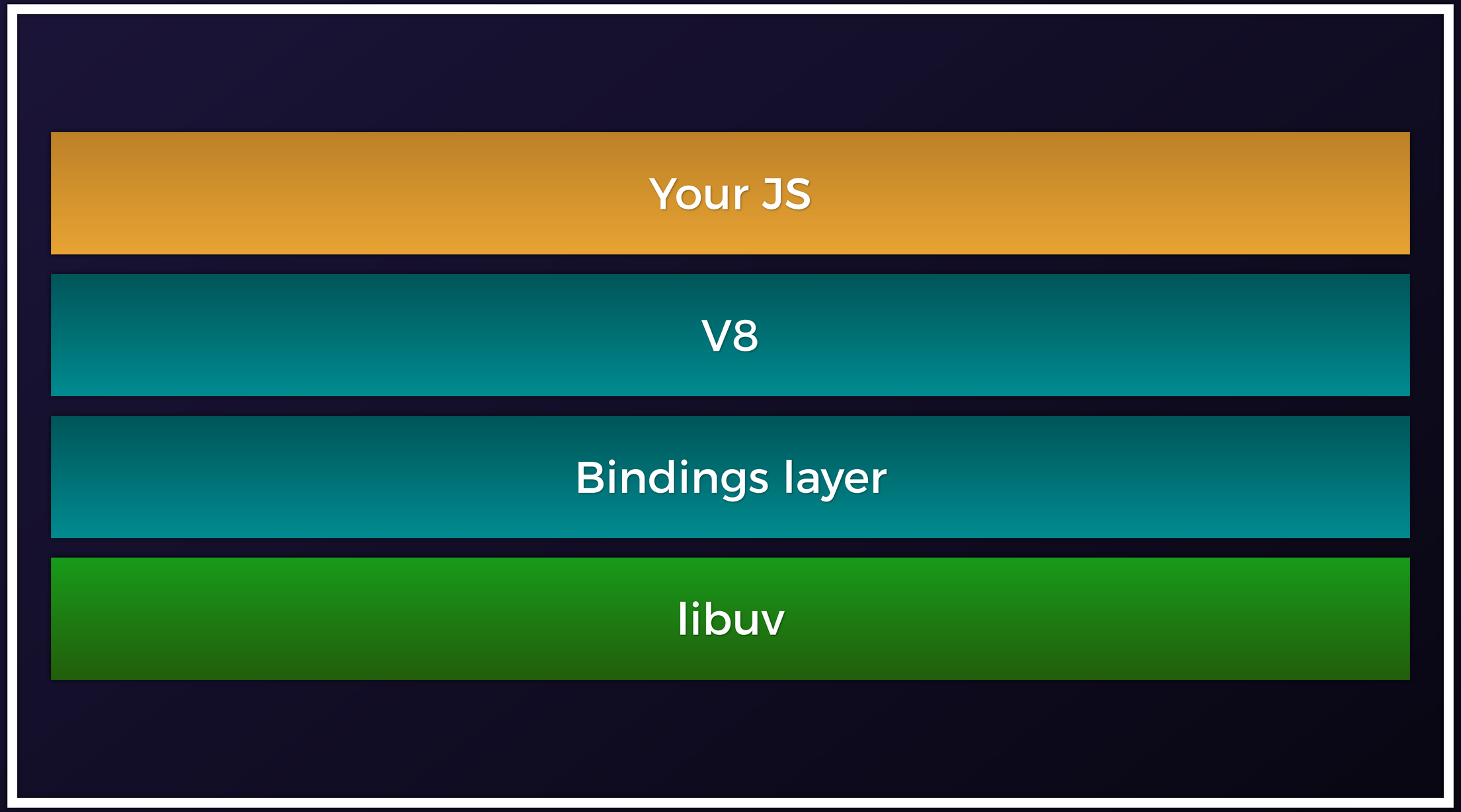


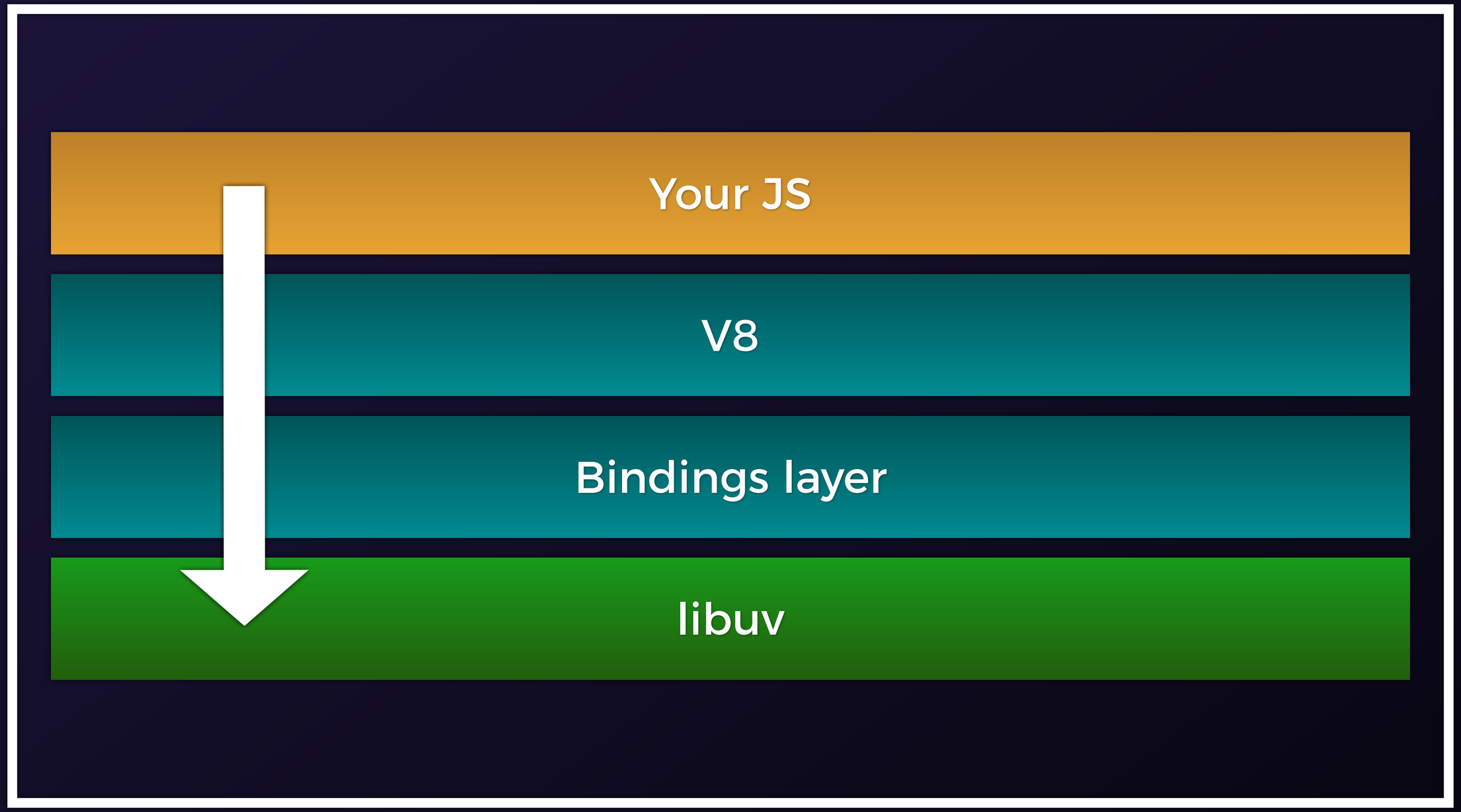
Thread pool



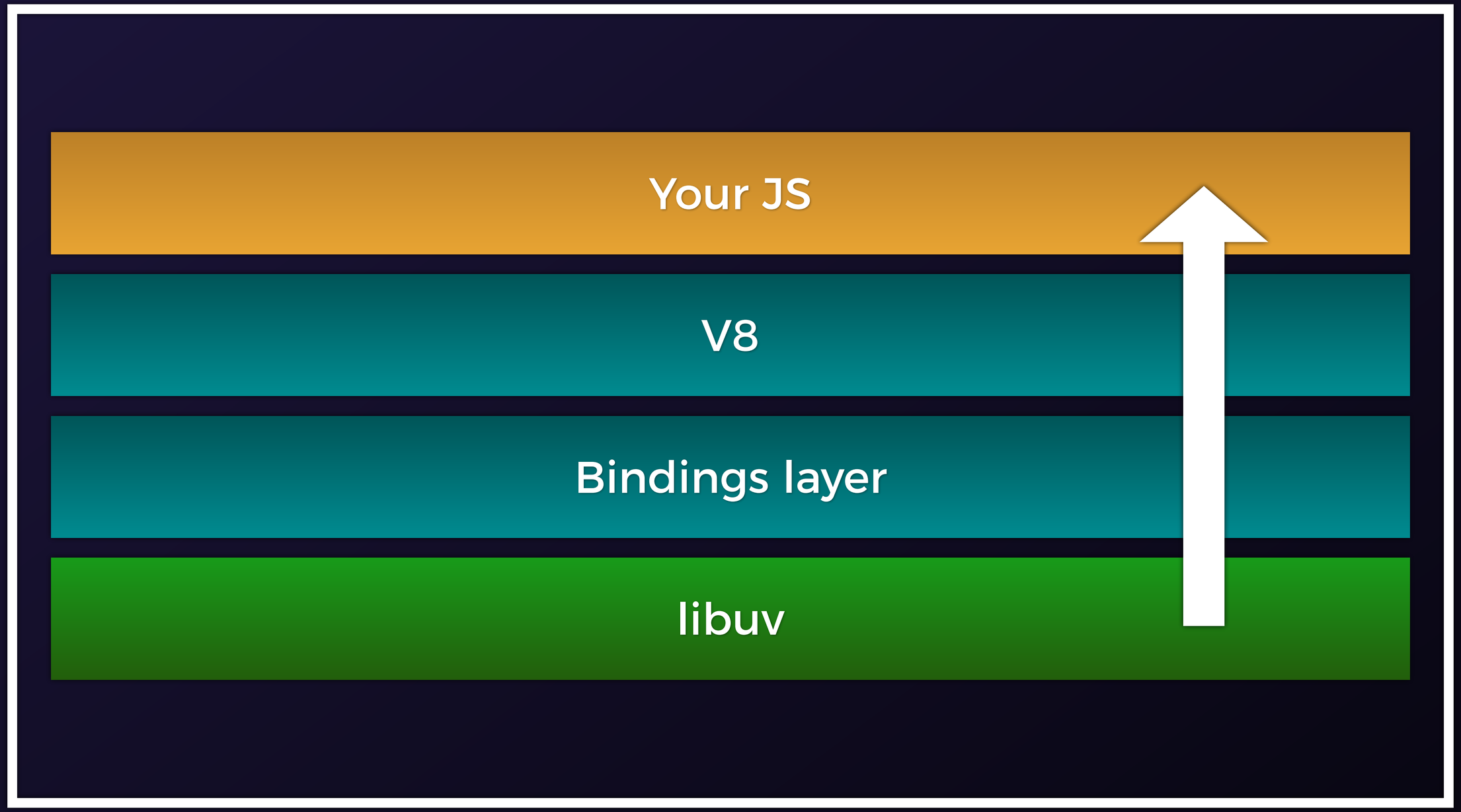


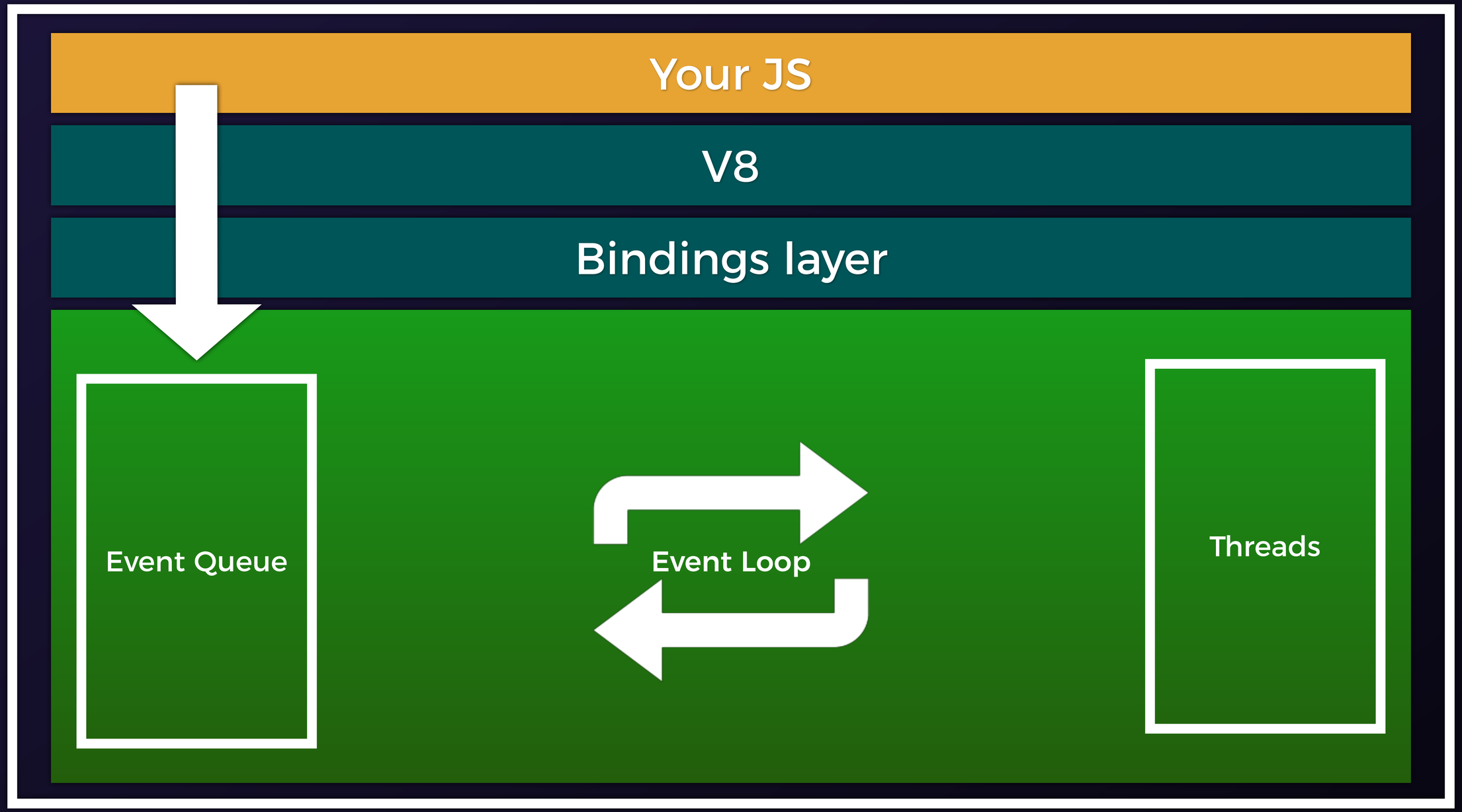
What is Node.js?

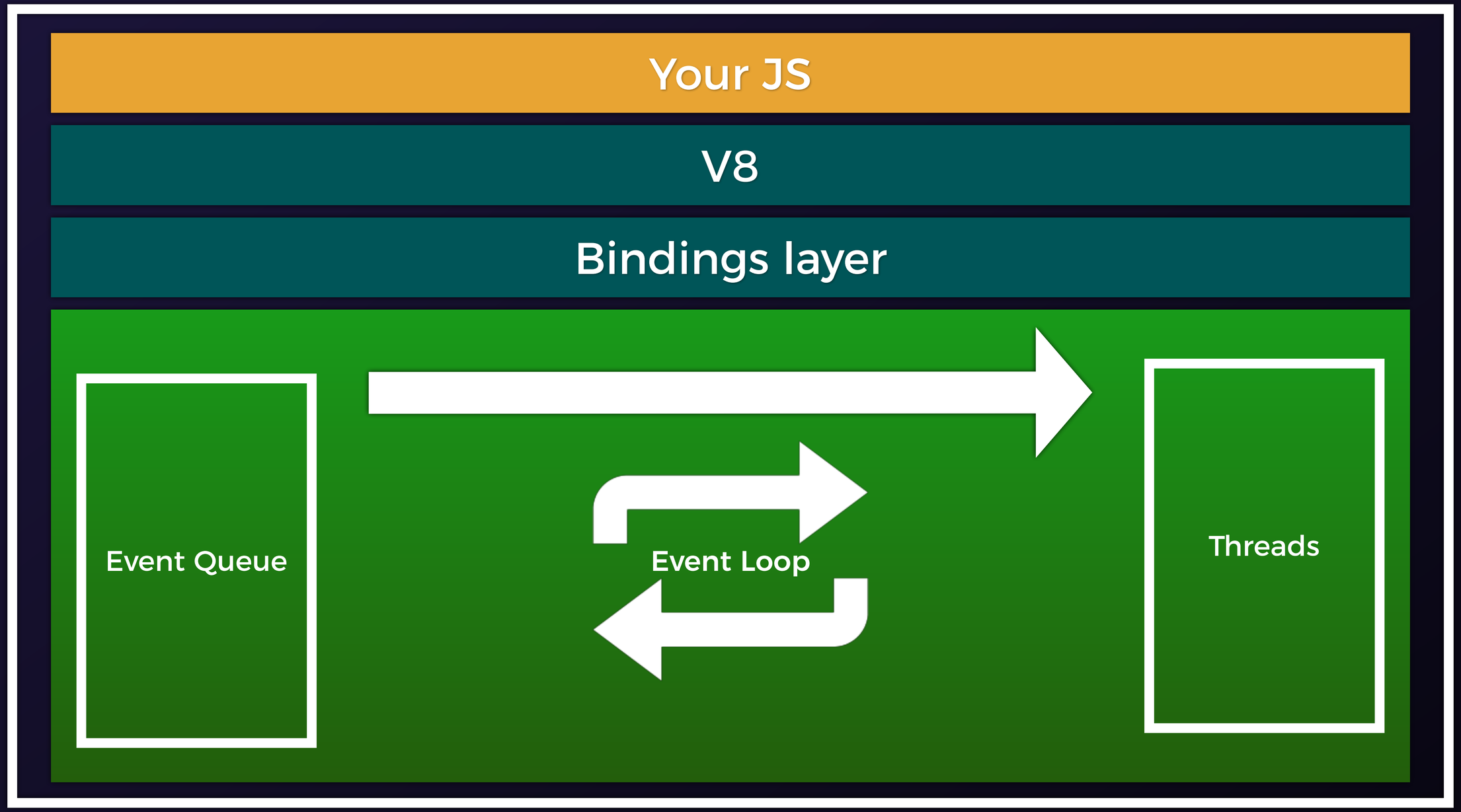

Your JS
V8
Bindings layer
libuv

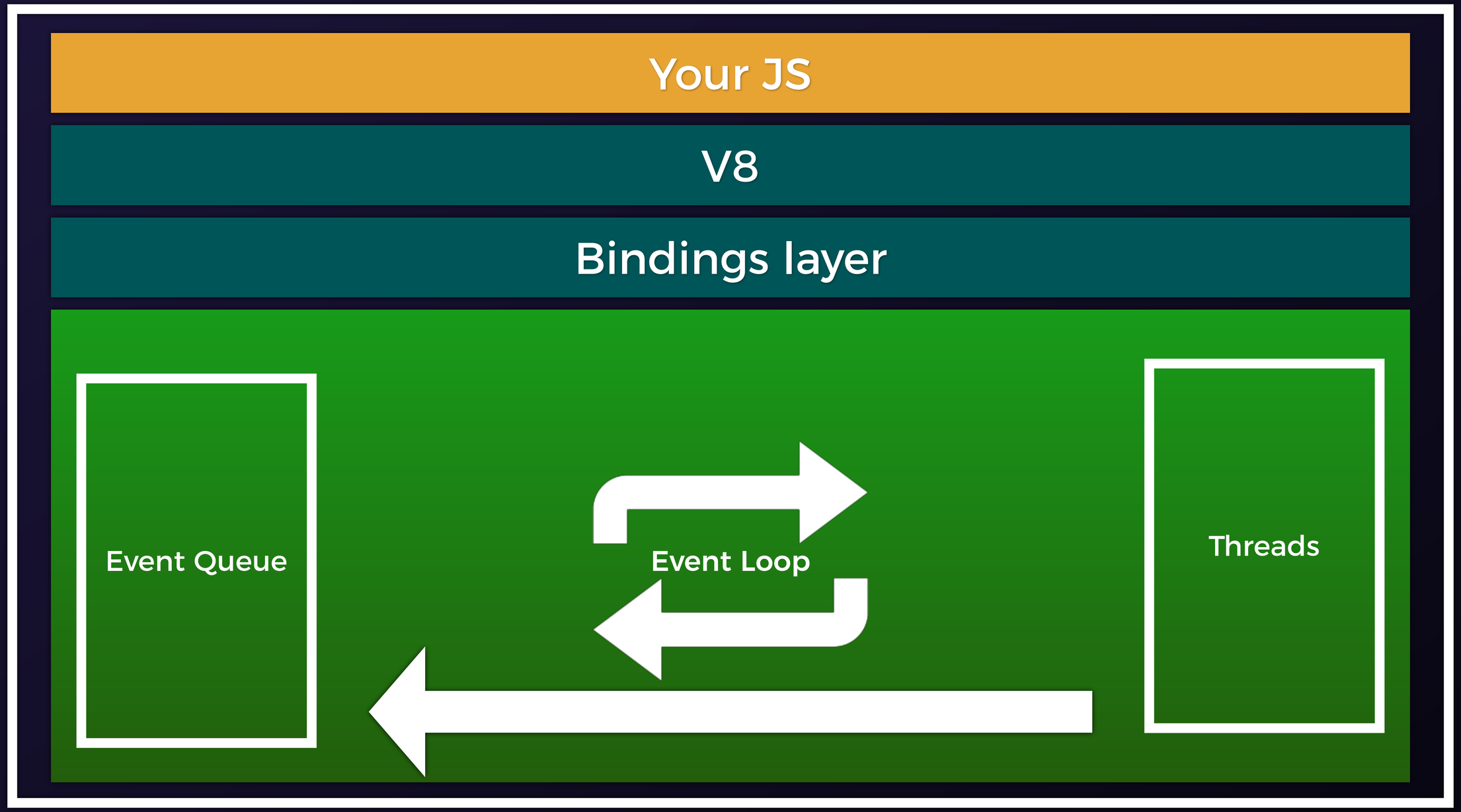

Your JS
V8
Bindings layer
libuv

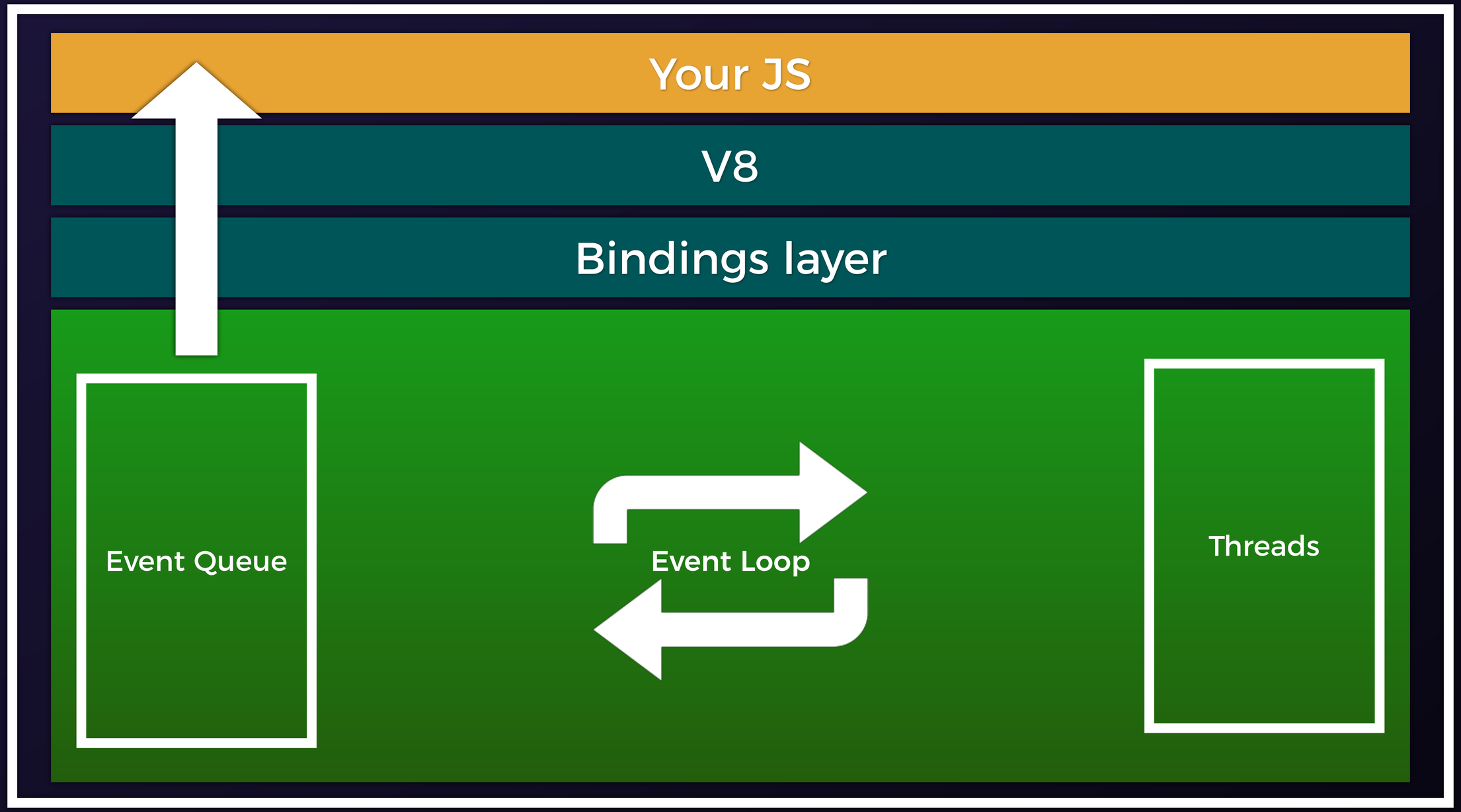
Your JS


Your JS
V8
Bindings layer
libuv


Your JS
V8
Bindings layer
Event Queue
Event Loop
Threads


Your JS
V8
Bindings layer
Event Queue
Event Loop
Threads


Your JS
V8
Bindings layer
Event Queue
Event Loop
Threads


Your JS
V8
Bindings layer
Event Queue
Event Loop
Threads

Thread 1

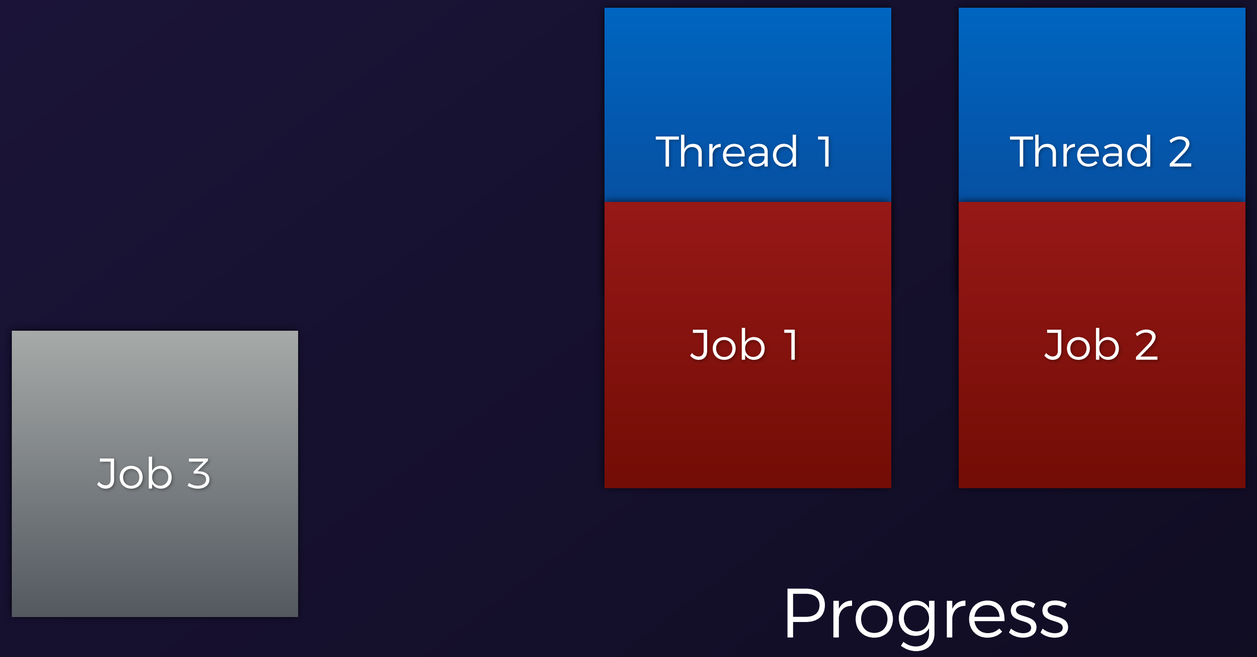
Thread 2

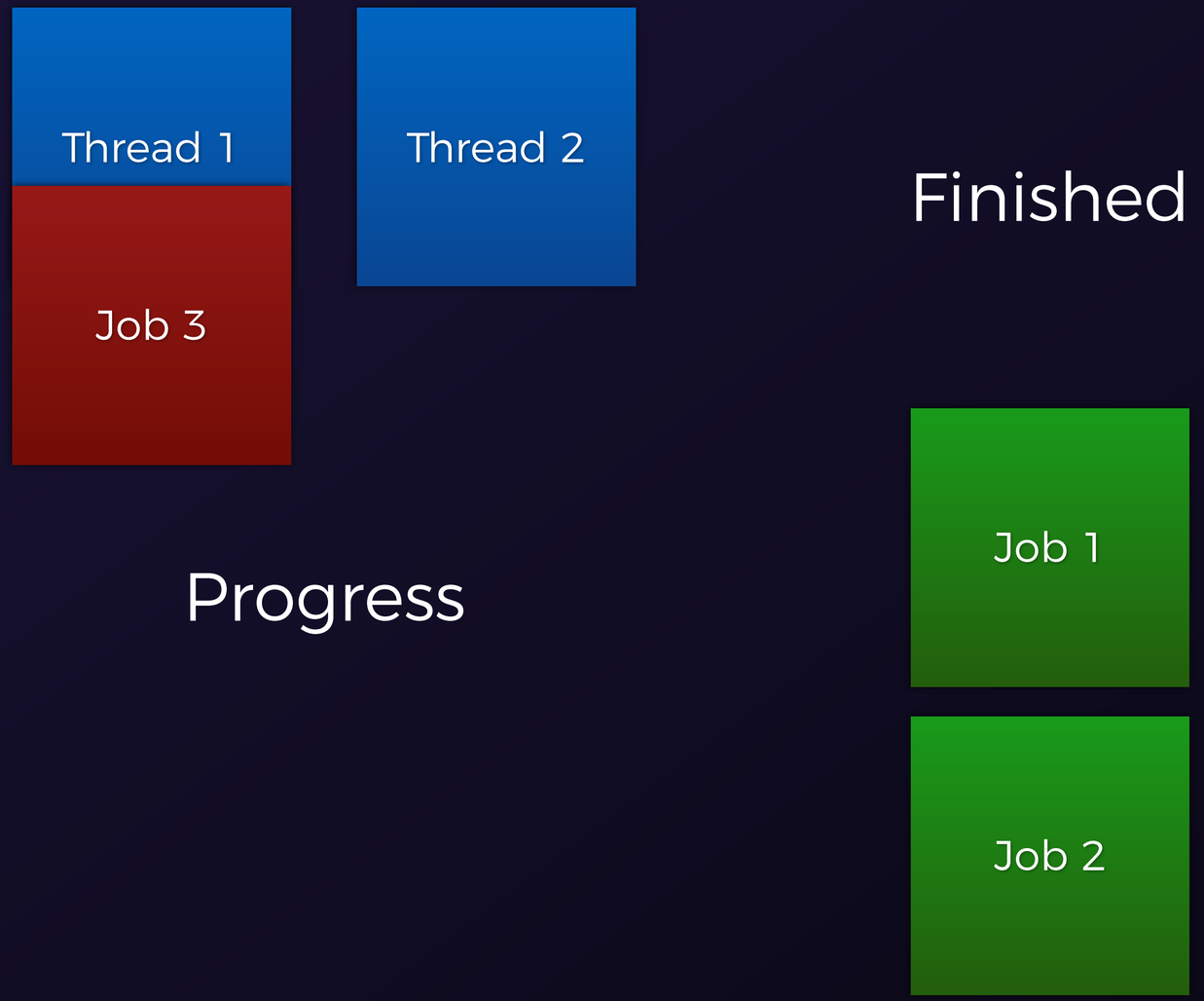
Queue

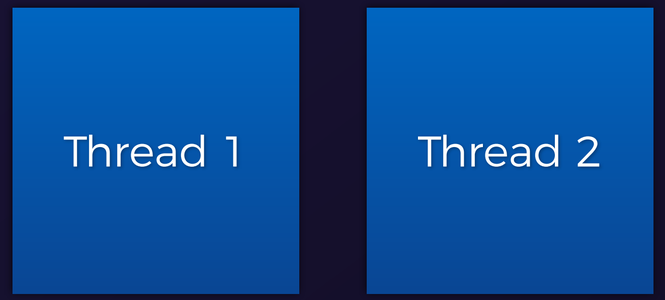
Job 1

Job 2

Job 3


Thread 1
Thread 2
Job 1
Job 2
Job 3
Progress


Thread 1
Thread 2
Job 3
Progress
Finished
Job 1
Job 2



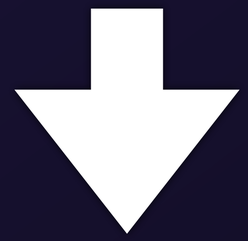
Finished

Job 1

Job 2

Job 3

Cluster

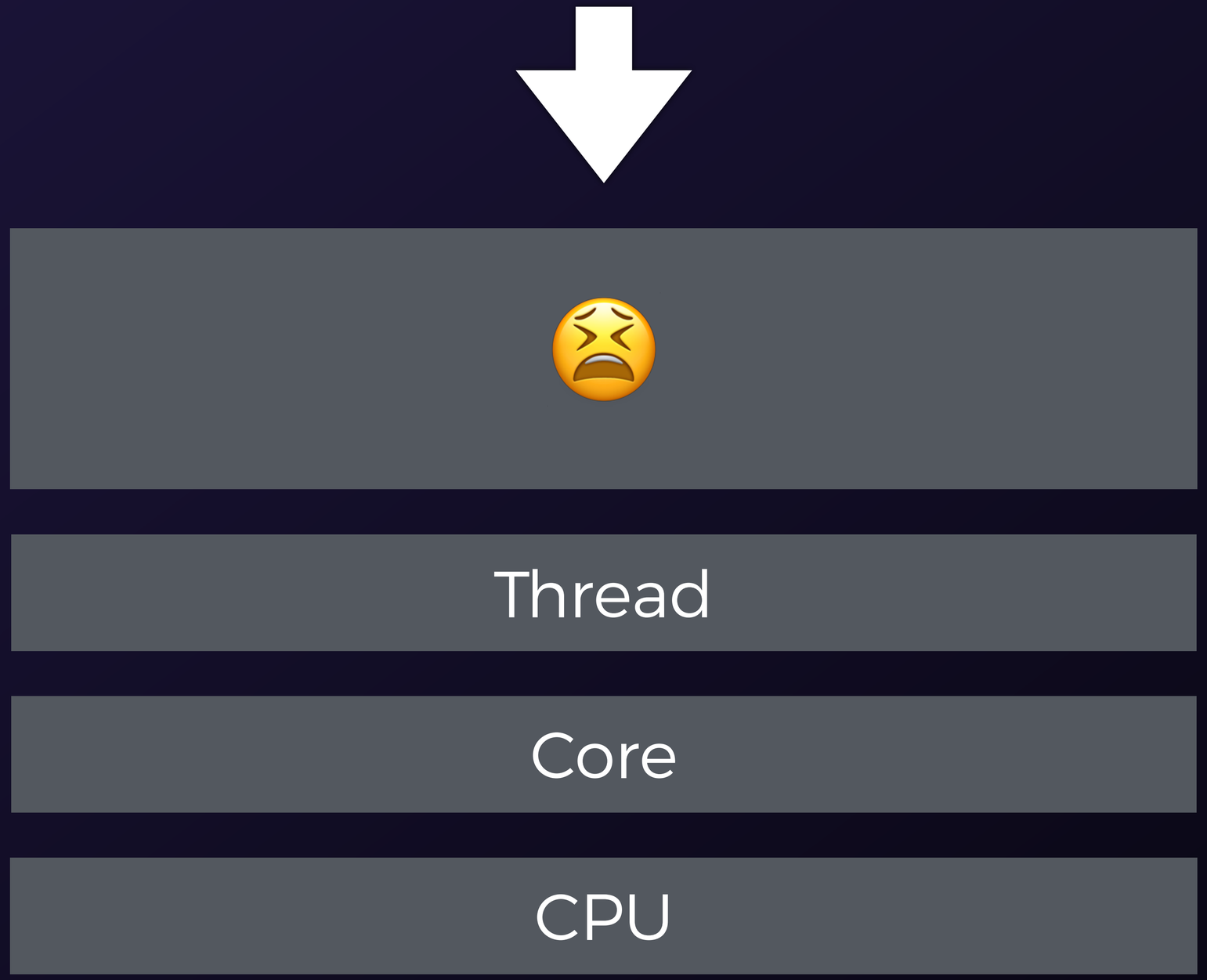


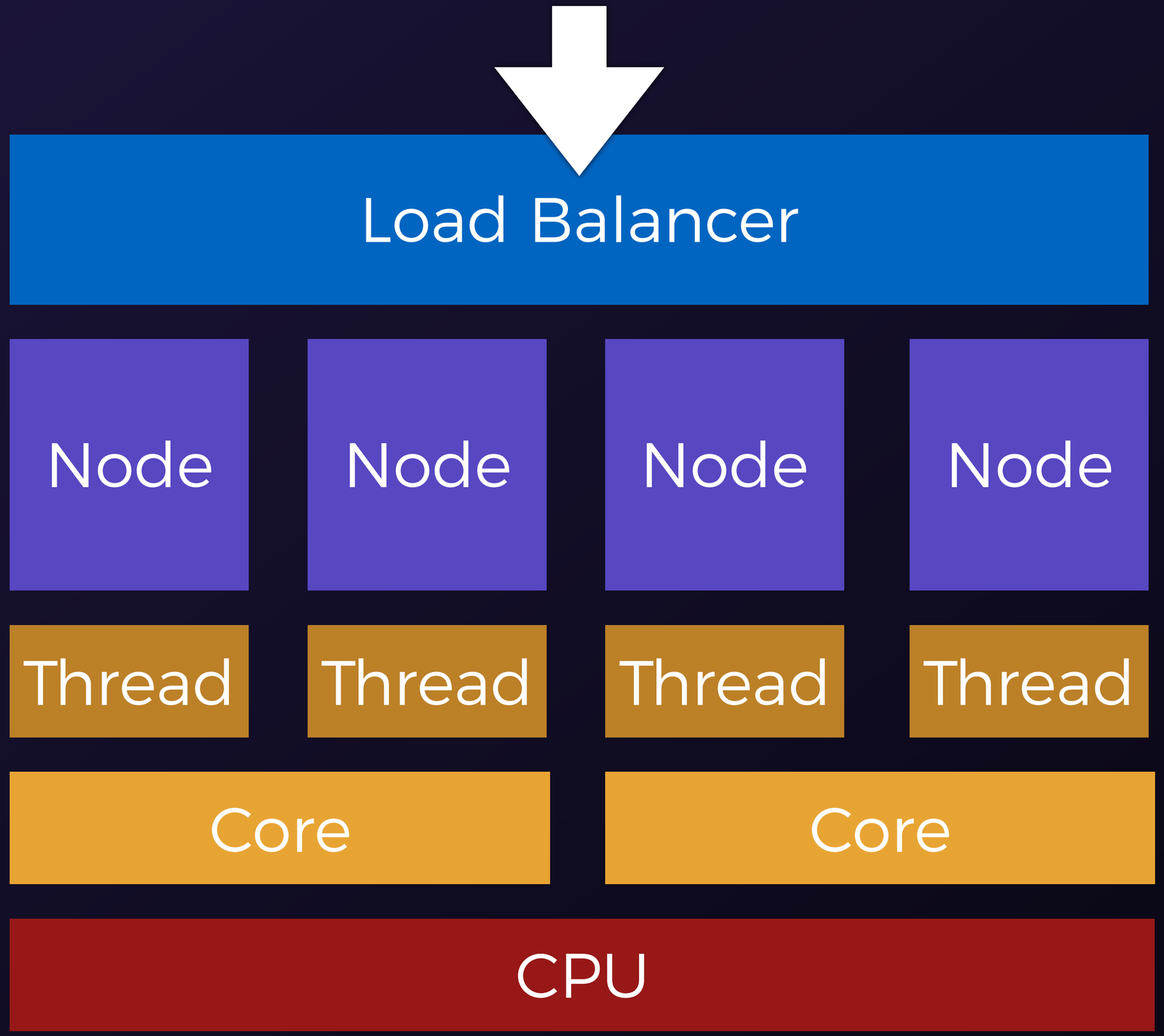
Node

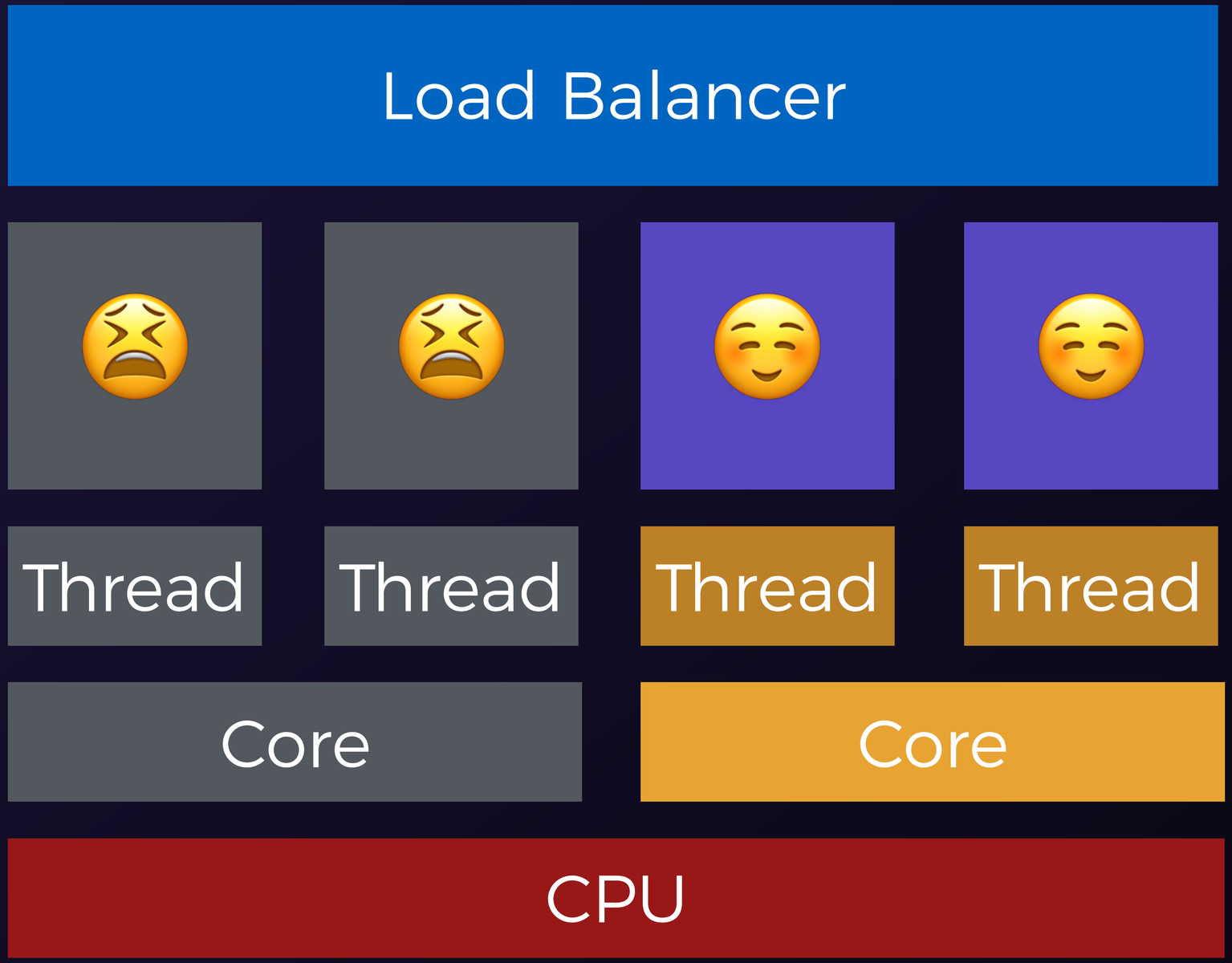
Thread

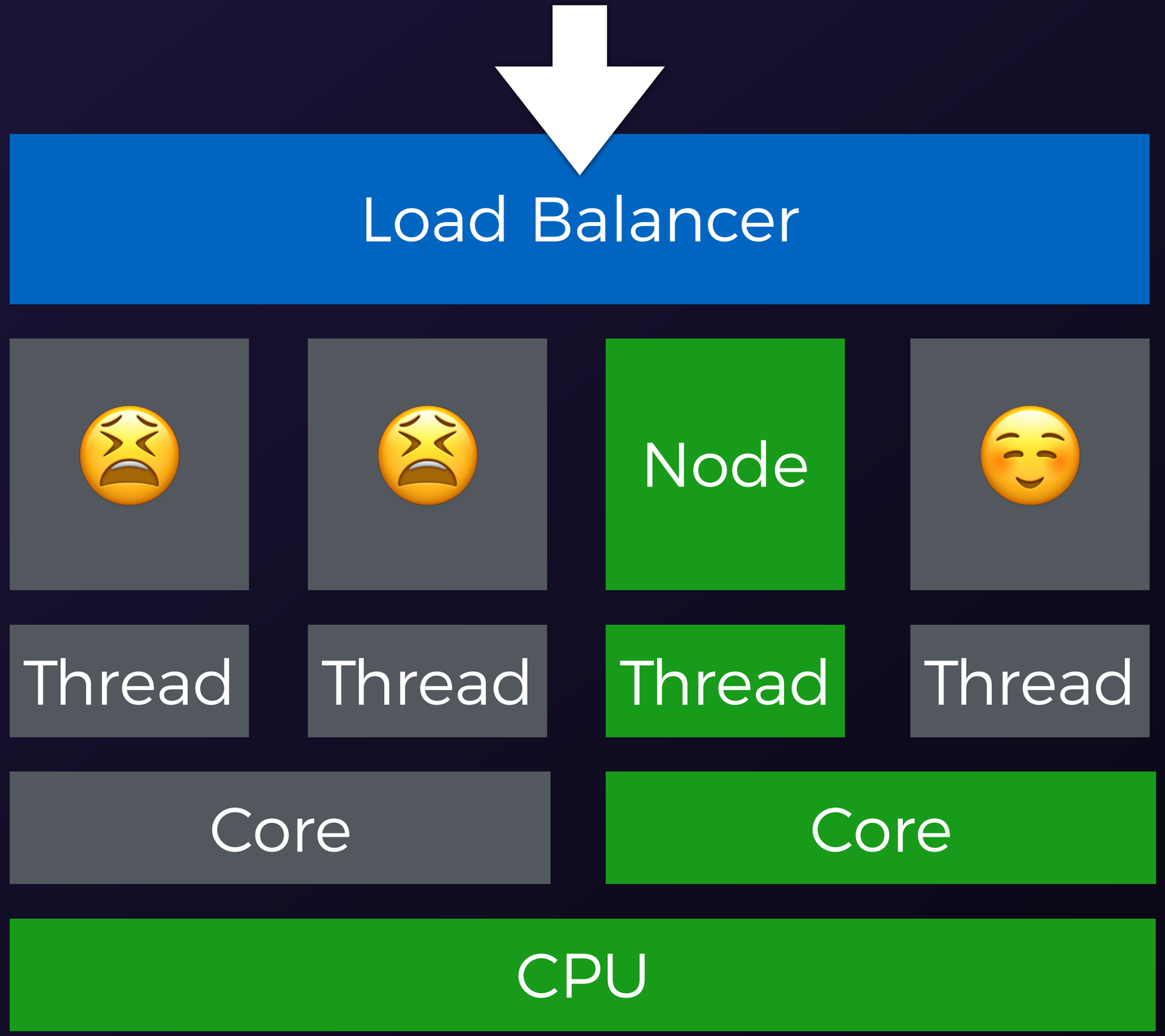
Core

CPU


😫
Thread
Core
CPU


Load Balancer
Node
Node
Node
Node
Thread
Thread
Thread
Thread
Core
Core
CPU


Load Balancer
Thread
Thread
Thread
Thread
Core
Core
CPU


Load Balancer
Node
Thread
Thread
Thread
Thread
Core
Core
CPU

PM2

```
artillery quick --count 10 -n 20
```

One Process

```
Summary report @ 22:01:45(+0100) 2018-11-25
  Scenarios launched:  10
  Scenarios completed: 10
  Requests completed:  200
  RPS sent: 13.91
  Request latency:
    min: 101.9
    max: 1272
    median: 606.4
    p95: 1229.8
    p99: 1267.5
  Scenario counts:
    0: 10 (100%)
  Codes:
    200: 200
```

Cluster with 4 processes

```
Summary report @ 22:03:05(+0100) 2018-11-25
  Scenarios launched:  10
  Scenarios completed: 10
  Requests completed:  200
  RPS sent: 25.35
  Request latency:
    min: 70.7
    max: 719.4
    median: 270.9
    p95: 557.2
    p99: 711.5
  Scenario counts:
    0: 10 (100%)
  Codes:
    200: 200
```

Worker Threads

Worker Threads are still experimental

Use with a flag:

node --experimental-worker [filename]

Worker Threads API

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

Limitations of WT

- You couldn't access Node API's
- You need to manager how to exchange the data between threads

- Worker per available thread may work
- Too many workers are bad
- Creating workers isn't time-free
- Exchanging data with workers isn't time free

```
~/Developmnet/playground/node-threads 41s
❯ node --experimental-worker threads-refactor.js
Finished at: 9651 ms. With 1 threads.

~/Developmnet/playground/node-threads 10s
❯ node --experimental-worker threads-refactor.js
Finished at: 9784 ms. With 1 thread(s).

~/Developmnet/playground/node-threads 10s
❯ node --experimental-worker threads-refactor.js
Finished at: 6163 ms. With 2 thread(s).

~/Developmnet/playground/node-threads 7s
❯ node --experimental-worker threads-refactor.js
Finished at: 4913 ms. With 3 thread(s).

~/Developmnet/playground/node-threads
❯ node --experimental-worker threads-refactor.js
Finished at: 4722 ms. With 4 thread(s).

~/Developmnet/playground/node-threads 6s
❯ node --experimental-worker threads-refactor.js
Finished at: 4888 ms. With 5 thread(s).

~/Developmnet/playground/node-threads
❯ node --experimental-worker threads-refactor.js
Finished at: 5133 ms. With 6 thread(s).

~/Developmnet/playground/node-threads 6s
❯ node --experimental-worker threads-refactor.js
Finished at: 5403 ms. With 7 thread(s).
```

What I can use WT for?

- Heavy data computations, ML
- Multiple file manipulations
- Compression/conversion
- Video streaming
- Bundling pipe for front-end frameworks
- ...

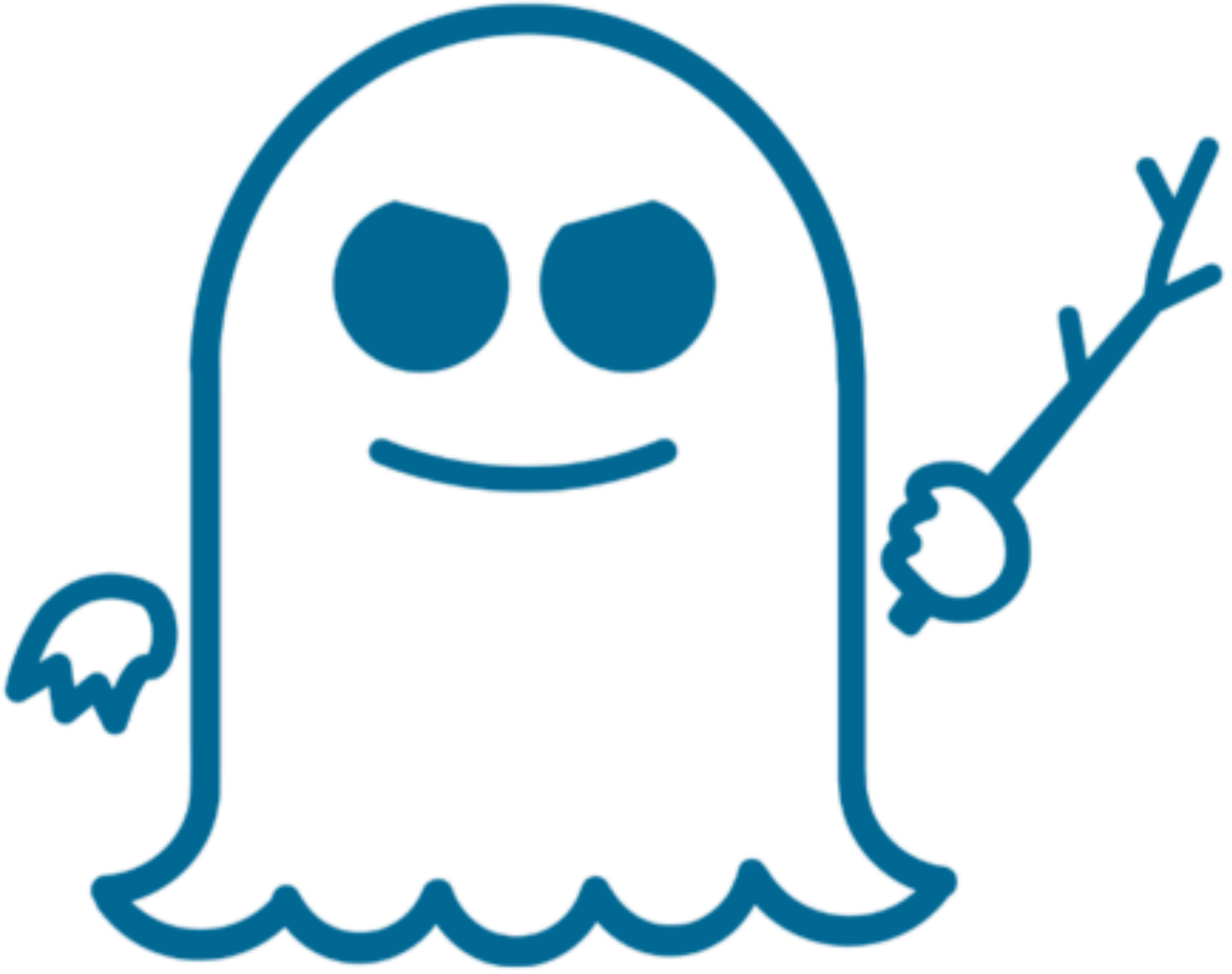
References

What should I think before using Worker Threads?

- Make sure that increased complexity of code worth it
- Calculate if performance price of workers isn't bigger than price of performance benefit
- Choose classical clustering solution if it's about API optimizations

Shared Memory


pikabu.ru



Array Buffer

- The need to specify the length in bytes
- Possibility use only strict type defined typeArray's views

Shared Array Buffer 👻🛡️

- **Was** available in Web Workers
- **Available** in Worker Threads

Atomics  

- ECMAScript 2017.
- Is supported by **some** browsers. Damaged by "Spectre" and "Meltdown"
- Global singleton for every thread
- Hides the shared memory implementation behind
-

References

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer
- https://www.udemy.com/advanced-node-for-developers
- My demos: https://github.com/tryshchenko/nodejs-worker-thread-simple-demos

Thank You!

Would you send me a pigeon?
TRYSHCHENKO.COM
OTRY.EU