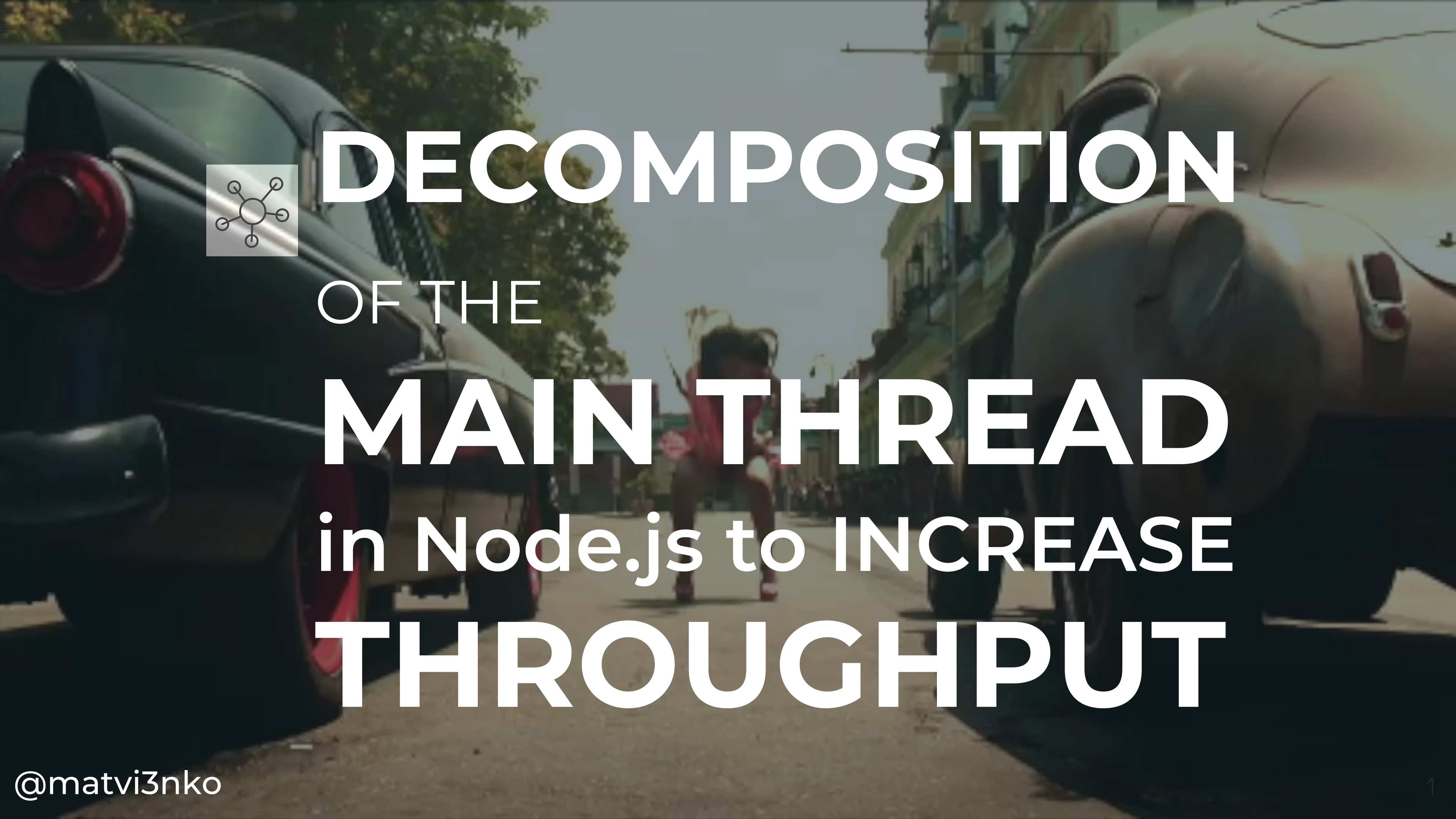


DECOMPOSITION OF THE MAIN THREAD in Node.js to INCREASE THROUGHPUT

@matvi3nko

HELLO!

I am Nikolay Matvienko

JS Developer at Grid Dynamics

You can find me at twitter.com/**matvi3nko**

github.com/**nickkooper**

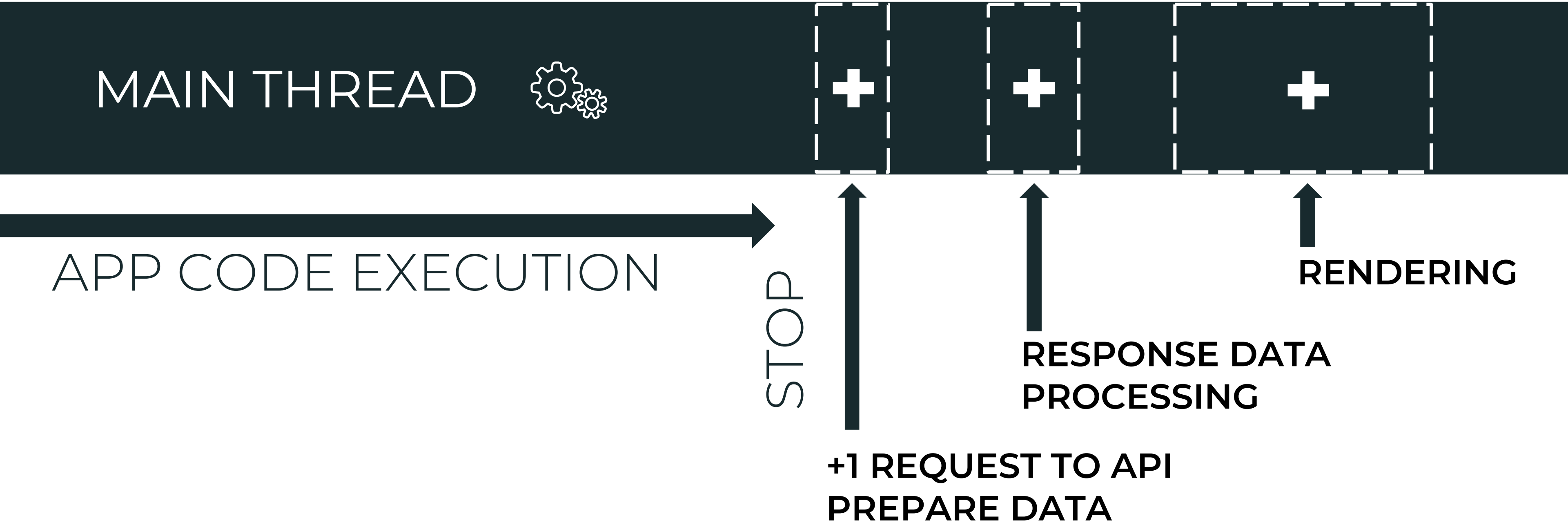
WEB UI BACKEND

ORCHESTRATION LAYER

MICROSERVICES

Node.js

ADDING NEW FEATURE



MAIN THREAD

APP CODE EXECUTION

STOP

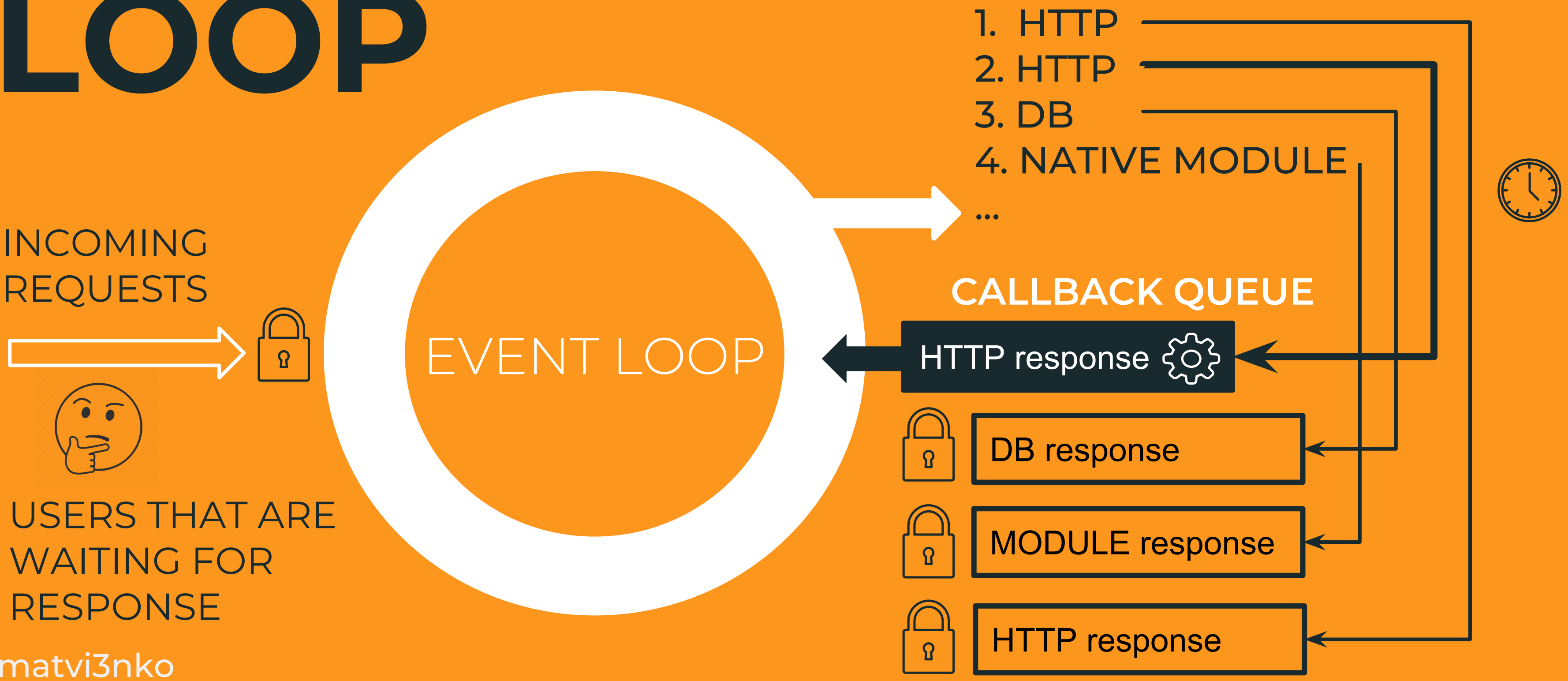
+1 REQUEST TO API
PREPARE DATA

RESPONSE DATA
PROCESSING

RENDERING

WHATEVER IS FAST TODAY IS SLOW TOMORROW AS DEMANDS CAN ONLY GO UP

BLOCKED EVENT LOOP



INCOMING REQUESTS

USERS THAT ARE WAITING FOR RESPONSE

EVENT LOOP

1. HTTP
2. HTTP
3. DB
4. NATIVE MODULE

...

CALLBACK QUEUE

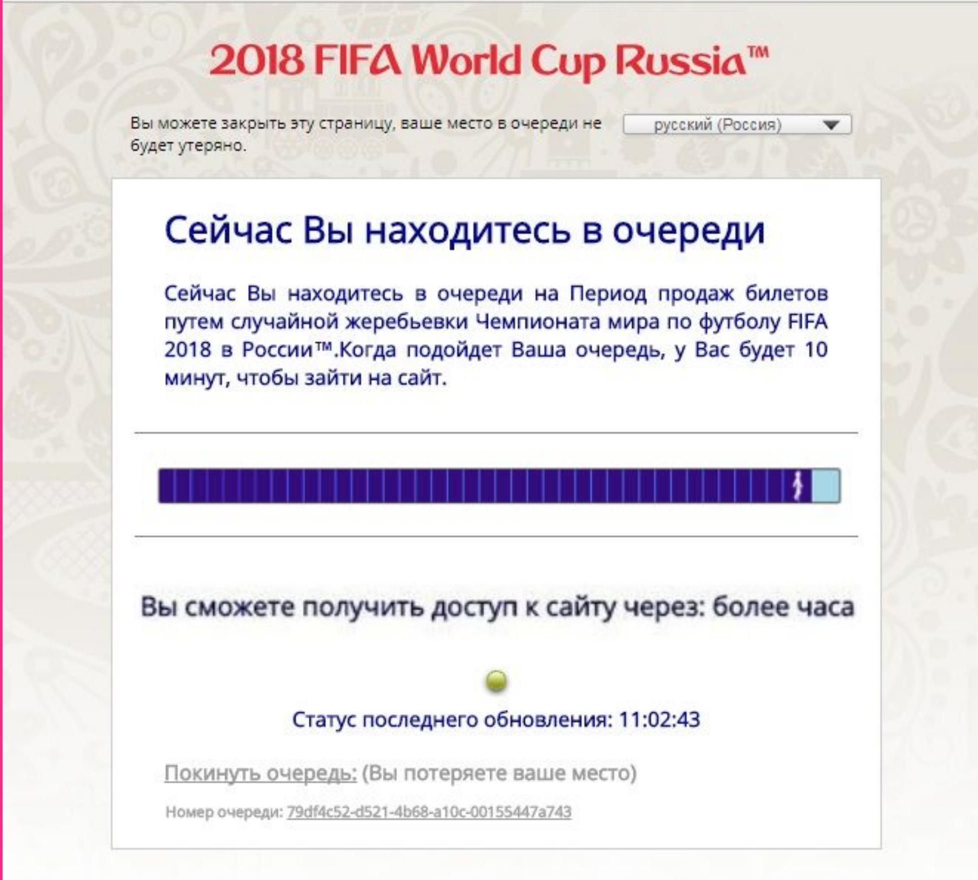
HTTP response

DB response

MODULE response

HTTP response

IN THE QUEUE



2018 FIFA World Cup Russia™

Вы можете закрыть эту страницу, ваше место в очереди не будет утеряно.

Сейчас Вы находитесь в очереди

Сейчас Вы находитесь в очереди на Период продаж билетов путем случайной жеребьевки Чемпионата мира по футболу FIFA 2018 в России™.Когда подойдет Ваша очередь, у Вас будет 10 минут, чтобы зайти на сайт.

Вы сможете получить доступ к сайту через: более часа

Статус последнего обновления: 11:02:43

Покинуть очередь: (Вы потеряете ваше место)

Номер очереди: 79df4c52-d521-4b68-a10c-00155447a743

BIG LATENCY

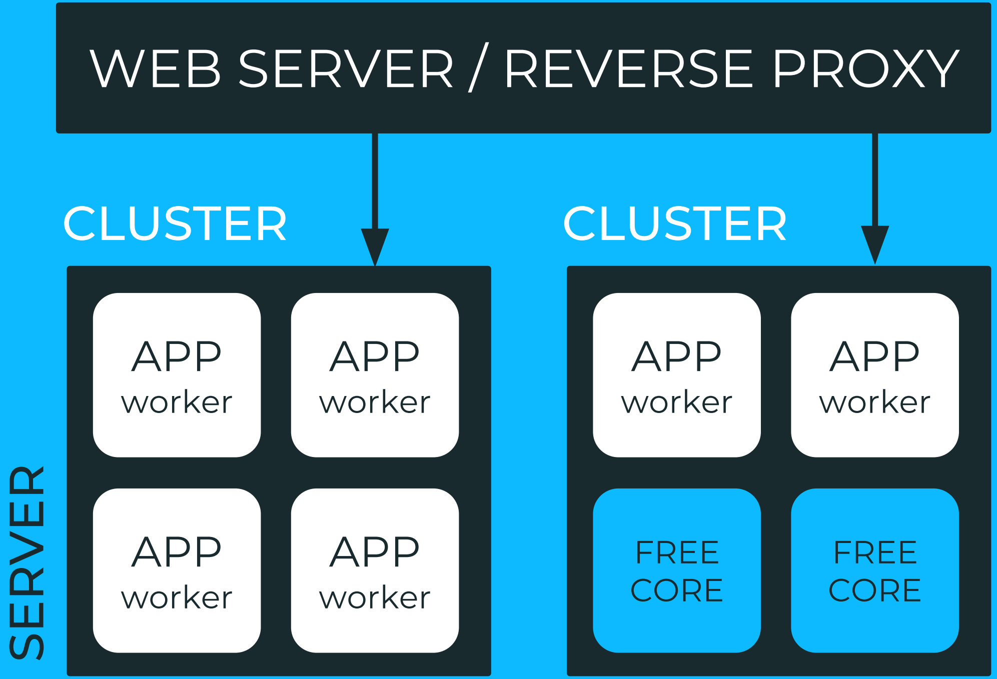
milliseconds

THROUGHPUT

request/second

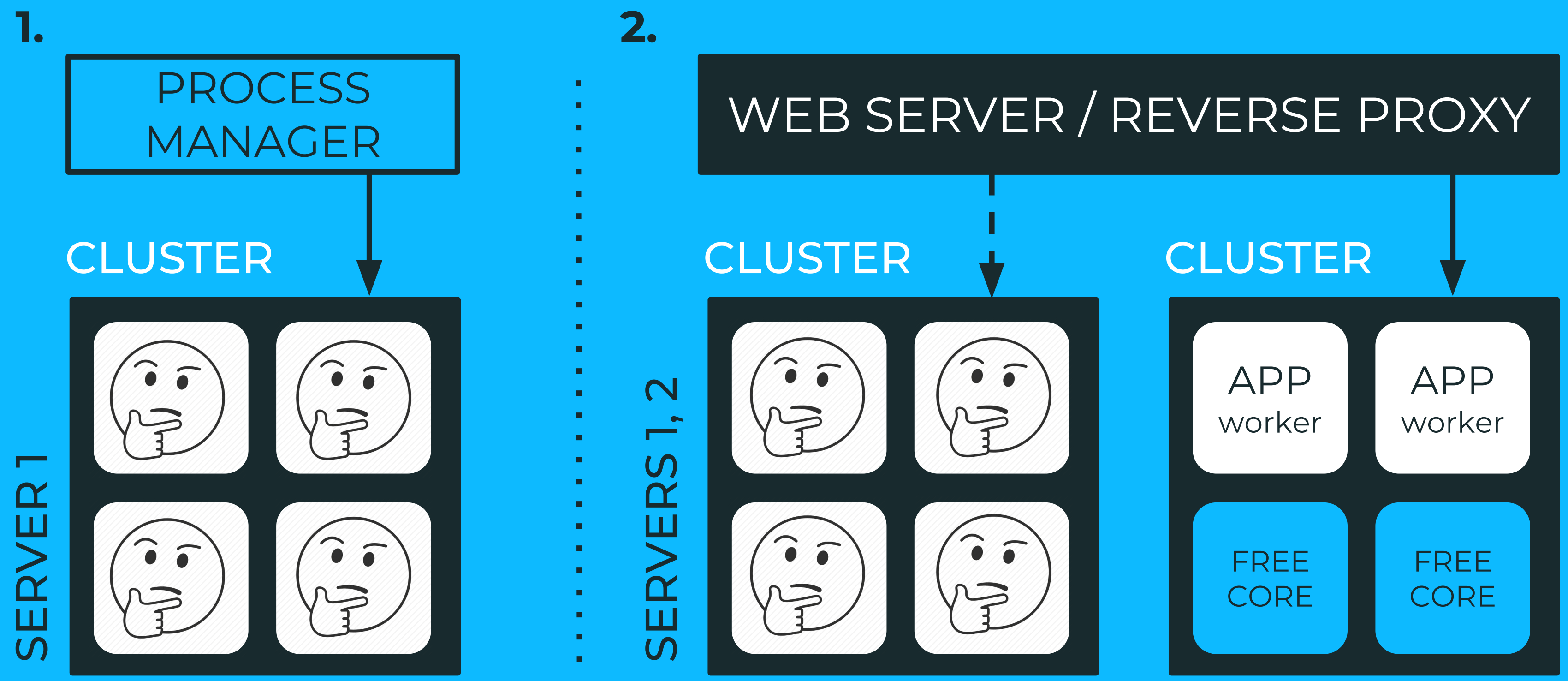
SCALING

1. **MULTIPLE PROCESSES**
   - CLUSTER module
   - PM2
2. **MULTIPLE SERVERS**
   WEB SERVER
   / REVERSE PROXY
   - PHUSION PASSENGER
   - NGINX



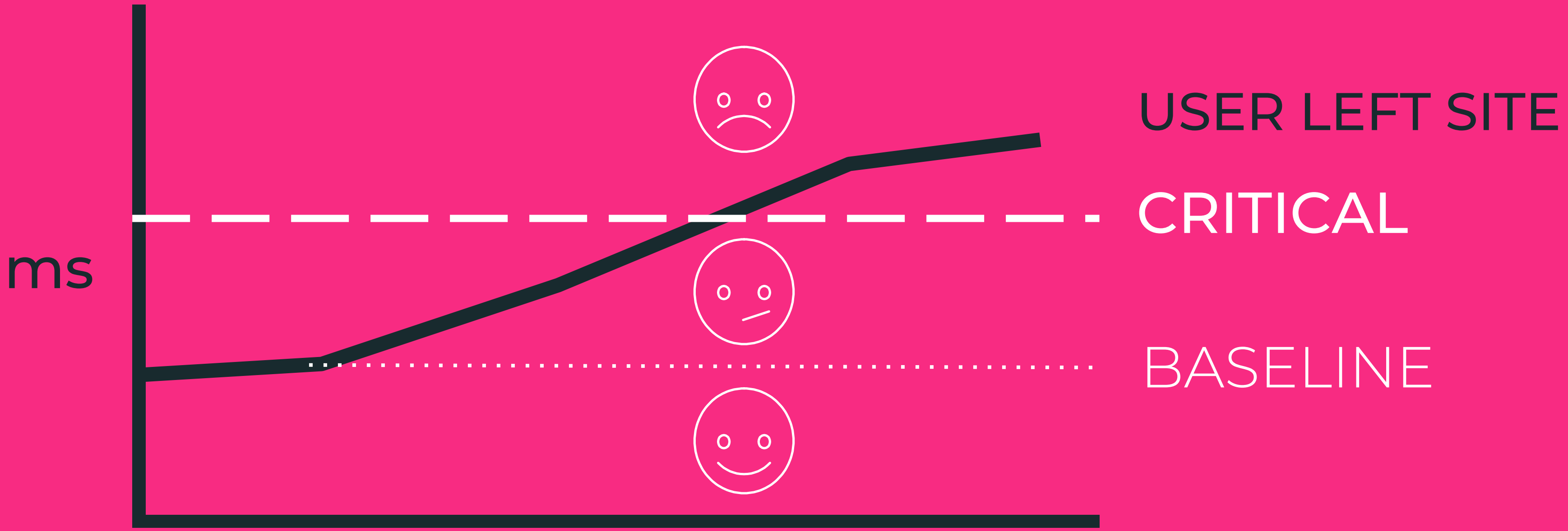
@matvi3nko

LOAD BALANCING

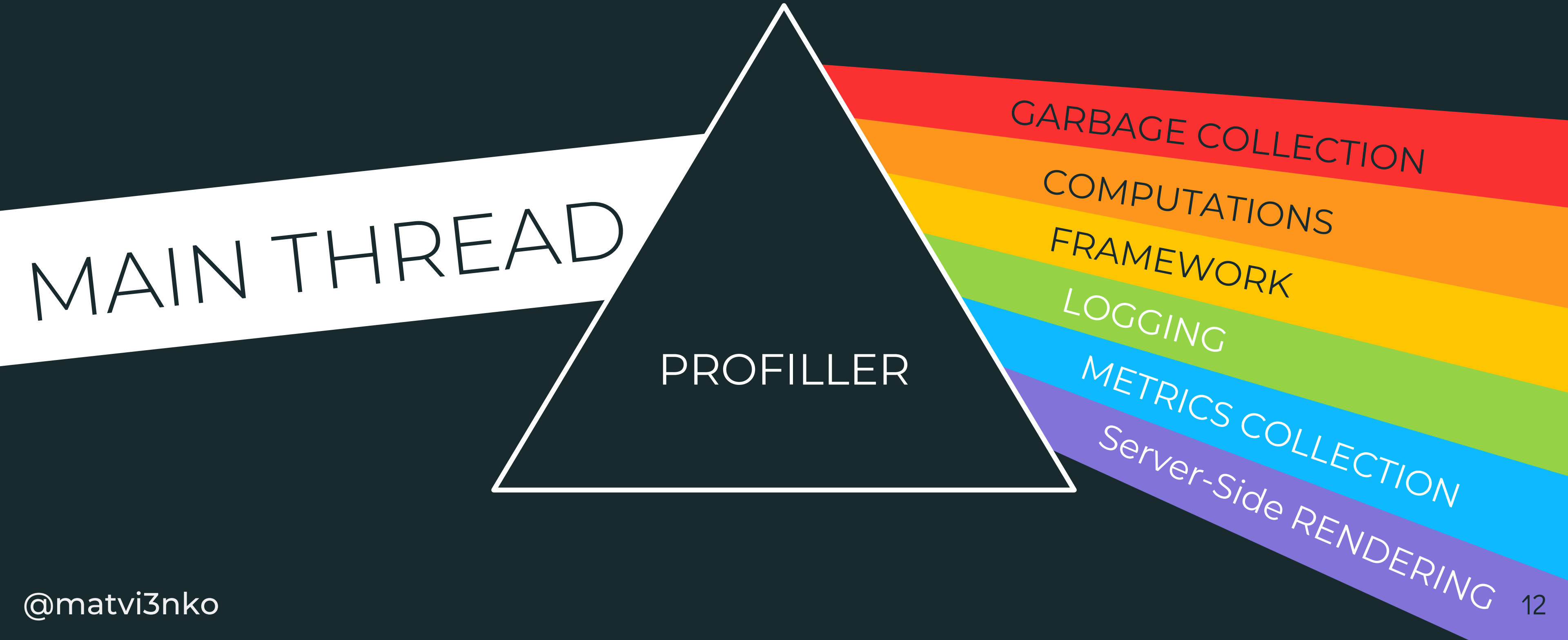


@matvi3nko

RESPONSE TIME

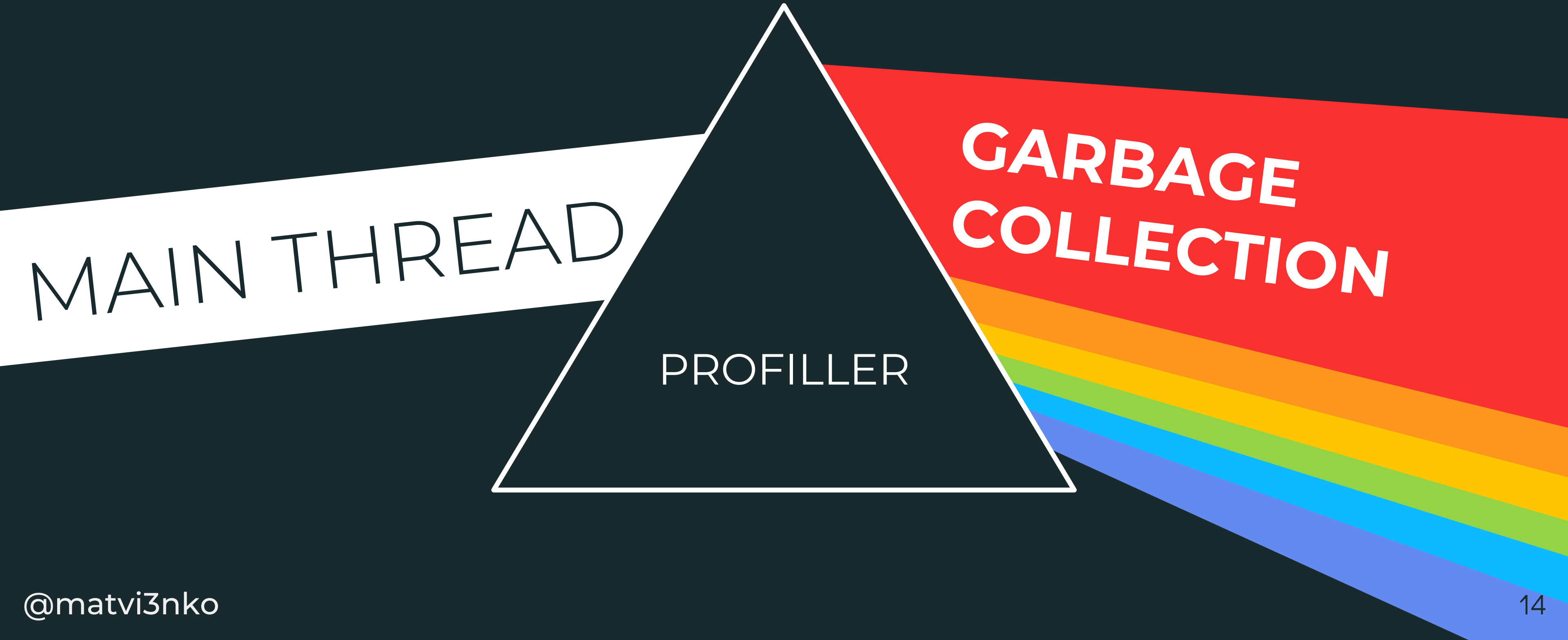


DISPERSION


MAIN THREAD
PROFILLER
GARBAGE COLLECTION
COMPUTATIONS
FRAMEWORK
LOGGING
METRICS COLLECTION
Server-Side RENDERING

.... TOO MANY REQUESTS ARE HANDLED IN NODE.JS

@matvi3nko



MAIN THREAD

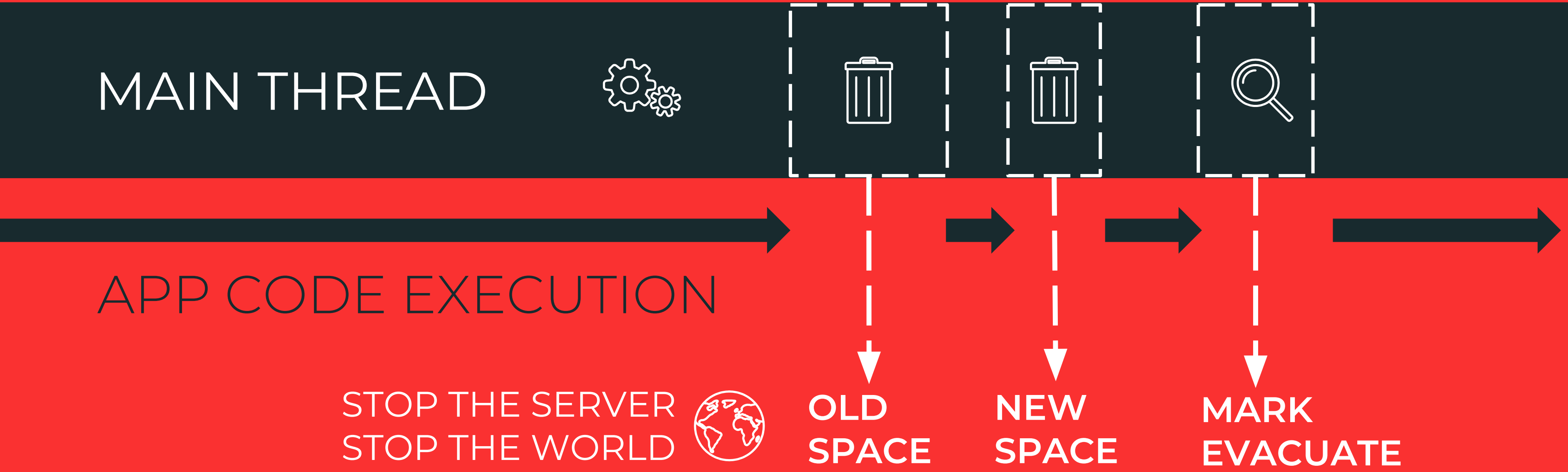
PROFILLER

GARBAGE COLLECTION

“THE WORLD 🌍 IS MINE”

© GARBAGE COLLECTOR
1959

GARBAGE COLLECTION



MAIN THREAD

APP CODE EXECUTION

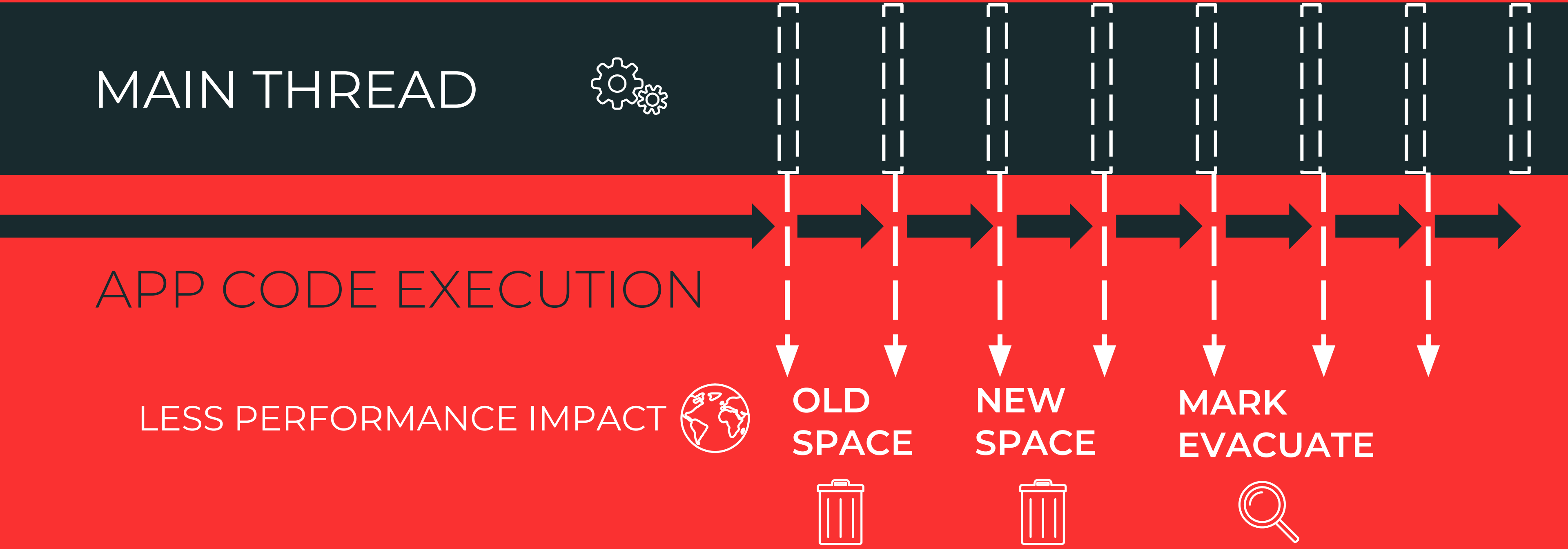
STOP THE SERVER
STOP THE WORLD

OLD SPACE

NEW SPACE

MARK EVACUATE

INCREMENTAL COLLECTION



MAIN THREAD

APP CODE EXECUTION

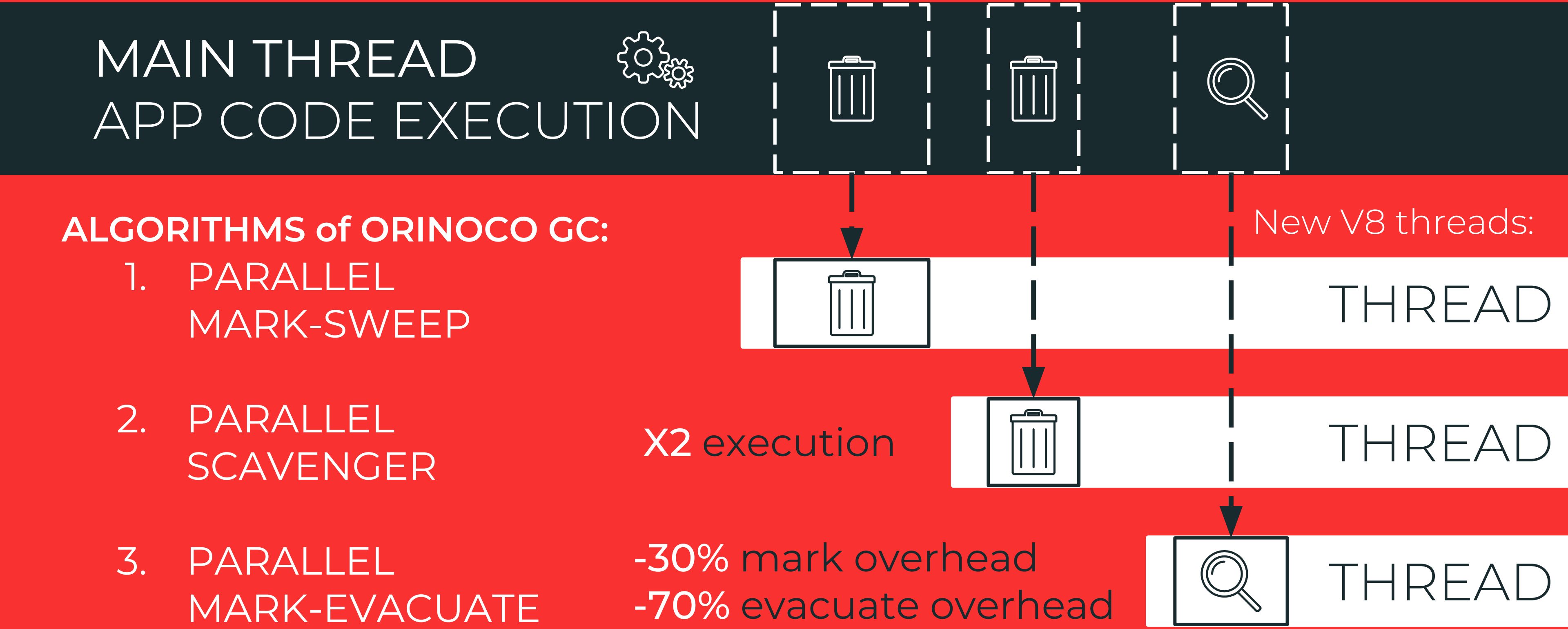
LESS PERFORMANCE IMPACT

OLD SPACE

NEW SPACE

MARK EVACUATE

GC DECOMPOSITION



ALGORITHMS of ORINOCO GC:

1. PARALLEL MARK-SWEEP
2. PARALLEL SCAVENGER
3. PARALLEL MARK-EVACUATE

X2 execution

-30% mark overhead
-70% evacuate overhead

@matvi3nko

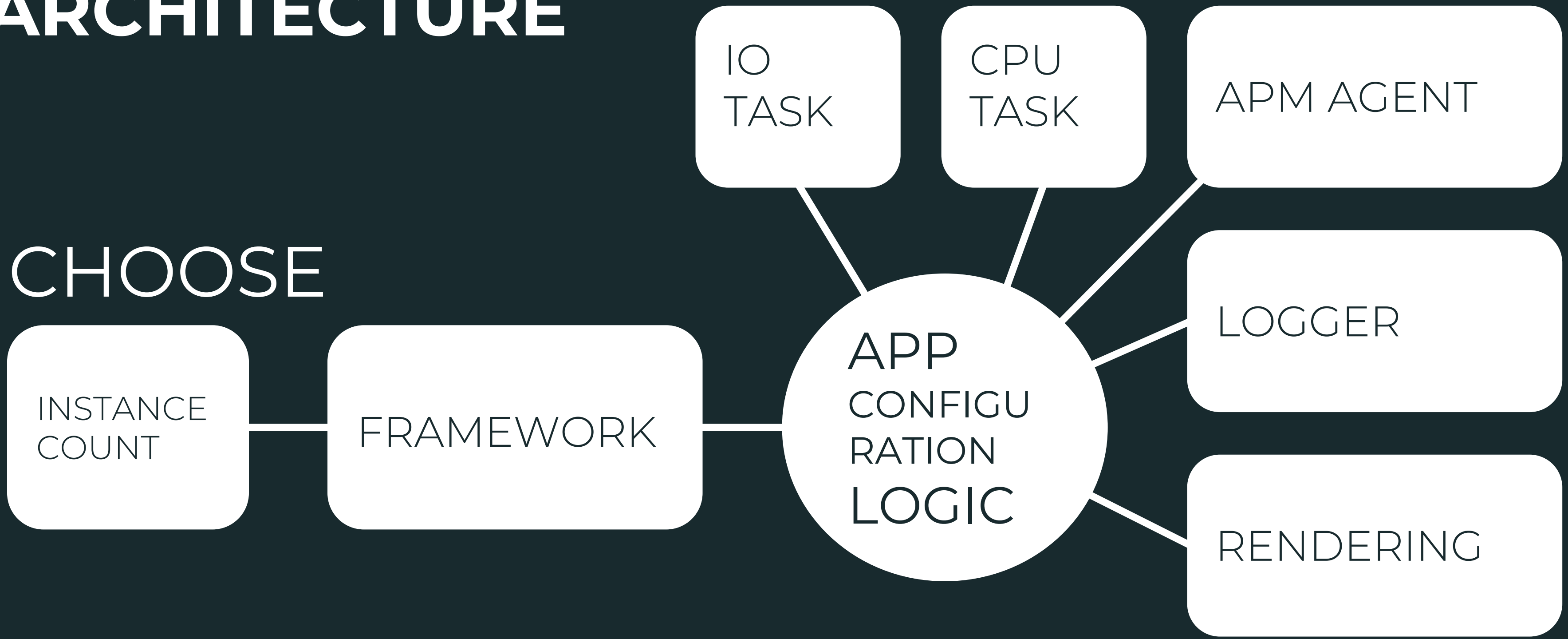
… 50 years later

“THE WORLD 🌍 IS MINE NOW”

© JS COMPUTING OPERATION IN NODE.JS
2009

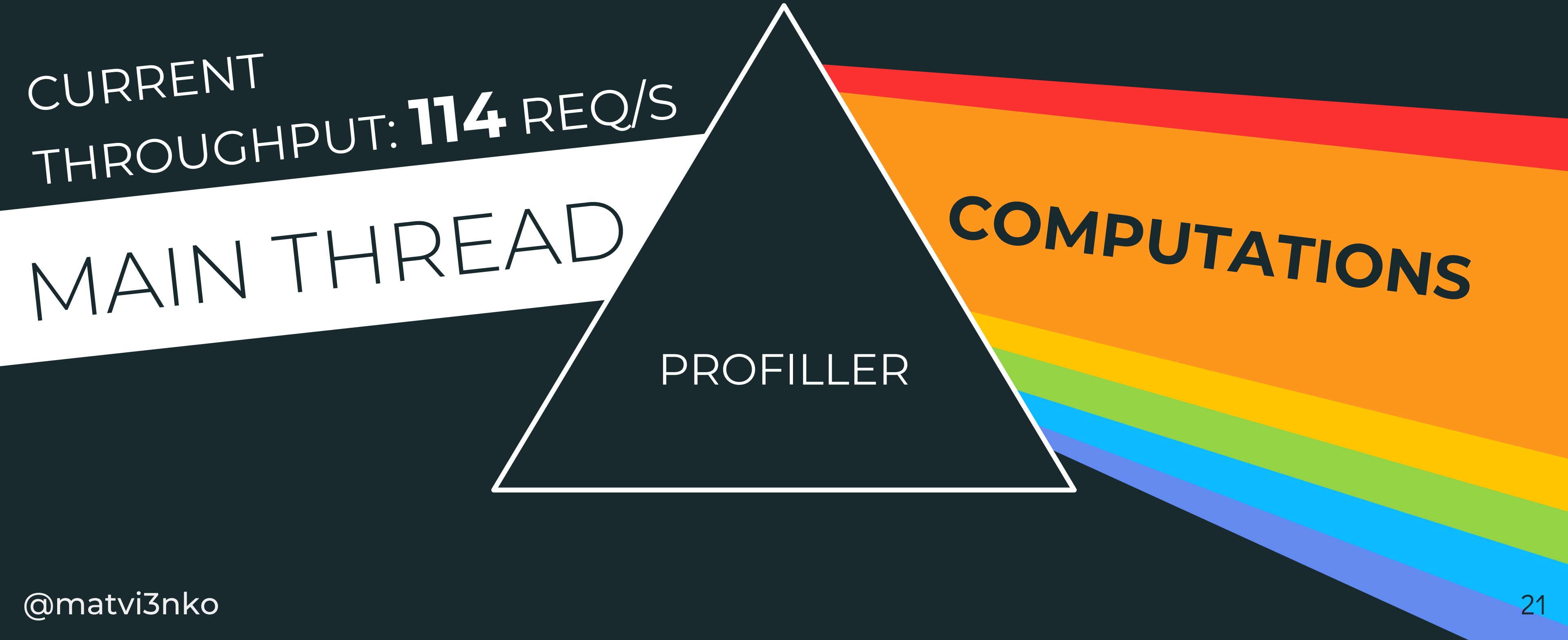
DEMO APP

ARCHITECTURE



@matvi3nko

CURRENT THROUGHPUT: **114** REQ/S



MAIN THREAD

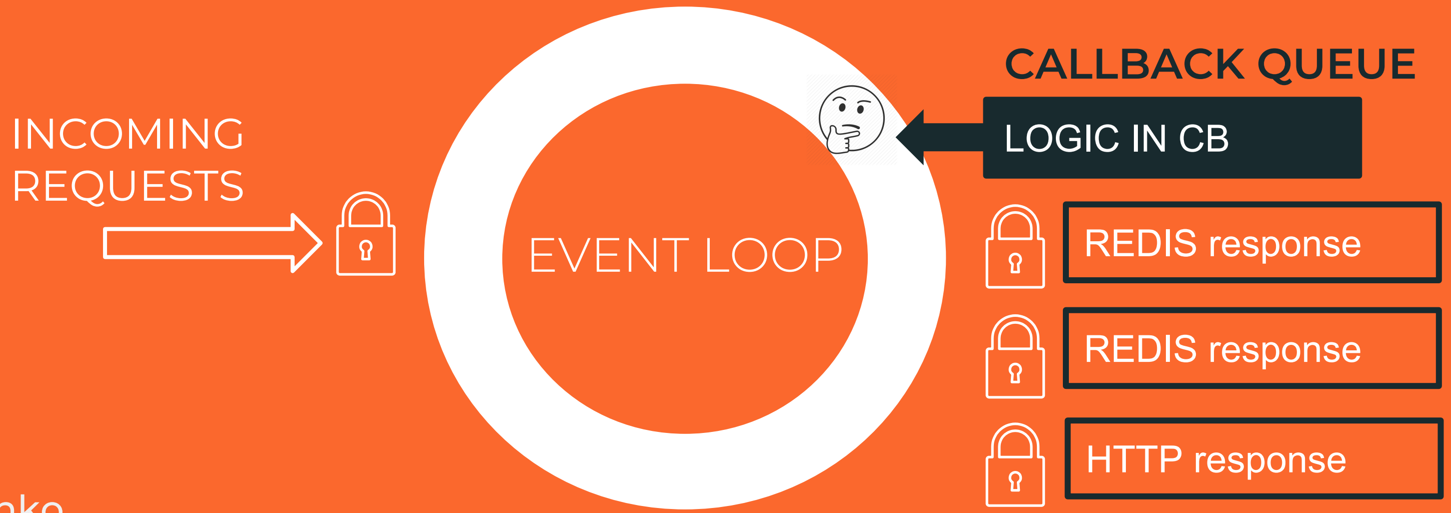
PROFILLER

COMPUTATIONS

CPU-BOUND TASKS

MAIN THREAD
APP CODE EXECUTION

CPU-bound processing



@matvi3nko

HOW TO PERFORM?

1. **Child Process fork**

Node.js core module

Create Process/Worker Pool

2. **Threads/Workers libraries**

Napa.js https://github.com/Microsoft/napajs

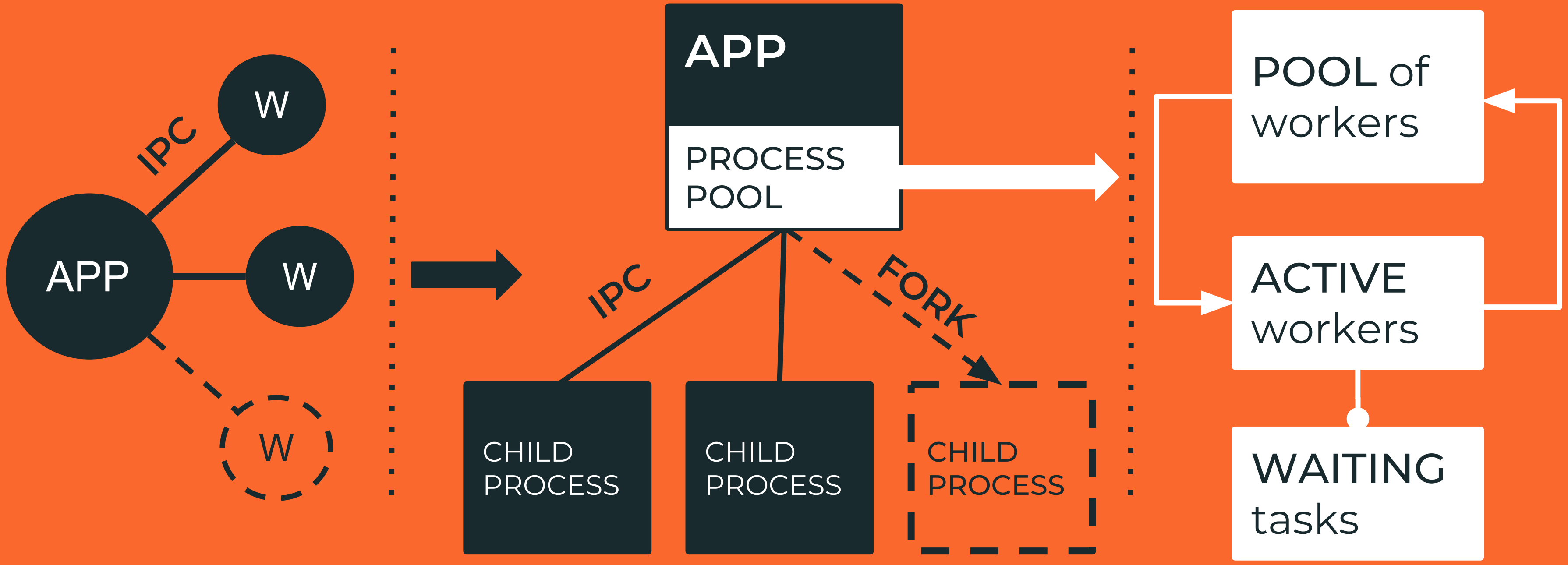
WebWorker Threads https://www.npmjs.com/package/webworker-threads

List of other https://github.com/SyntheticSemantics/List-of-Parallel-JS-Projects

3. **Microservices**

4. **Native Modules**

PROCESS POOL



APP

W

W

W

IPC

APP

PROCESS POOL

IPC

FORK

CHILD PROCESS

CHILD PROCESS

CHILD PROCESS

POOL of workers

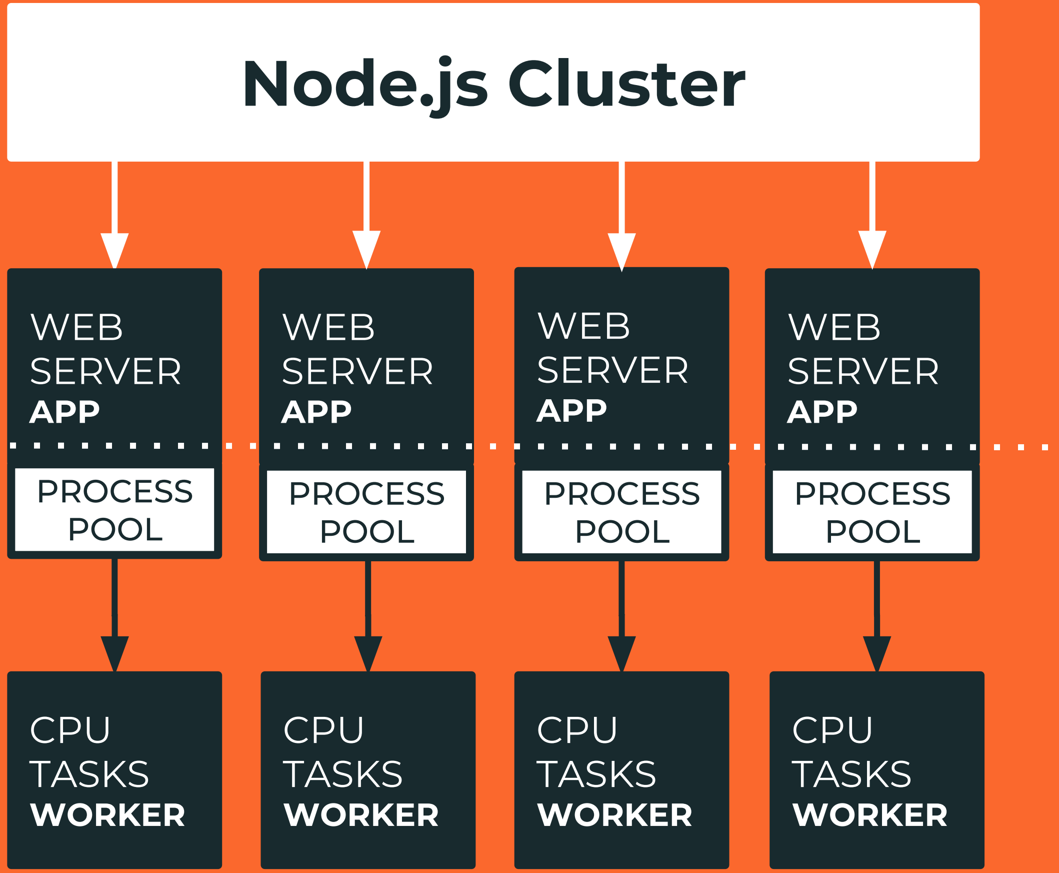
ACTIVE workers

WAITING tasks

CLUSTER

IN-PROCESS COMPUTING

PARALLEL OFF-PROCESS COMPUTING



Node.js Cluster

WEB SERVER **APP**

PROCESS POOL

CPU TASKS **WORKER**

WEB SERVER **APP**

PROCESS POOL

CPU TASKS **WORKER**

WEB SERVER **APP**

PROCESS POOL

CPU TASKS **WORKER**

WEB SERVER **APP**

PROCESS POOL

CPU TASKS **WORKER**

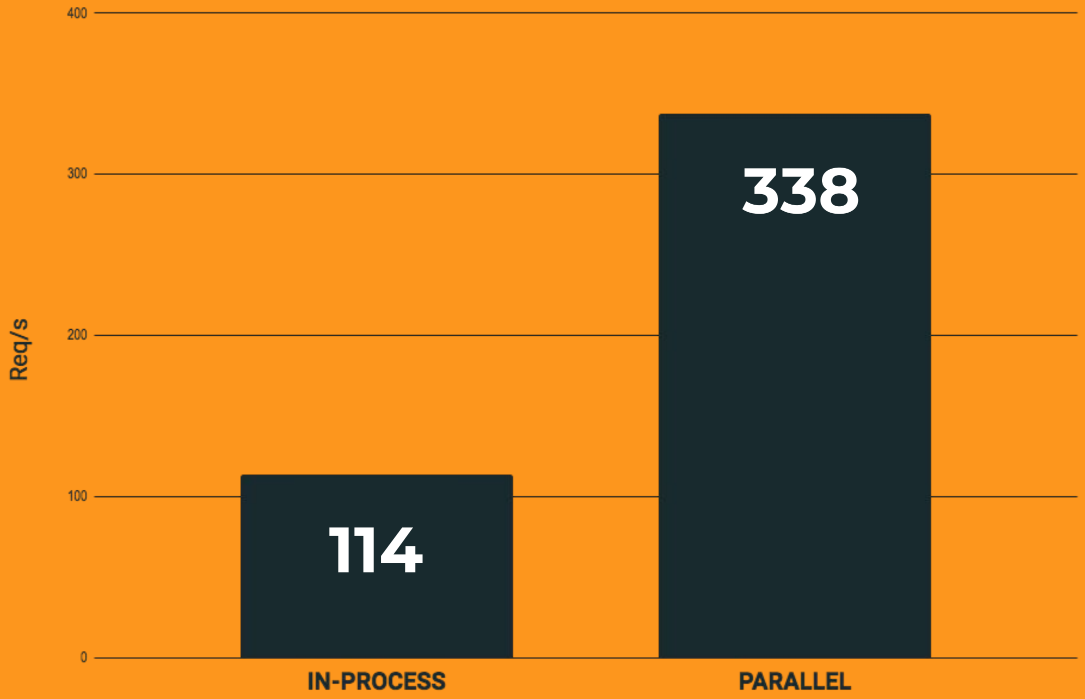
CPU TASKS PARALLELIZATION

RESULT

3X

MORE REQ/S

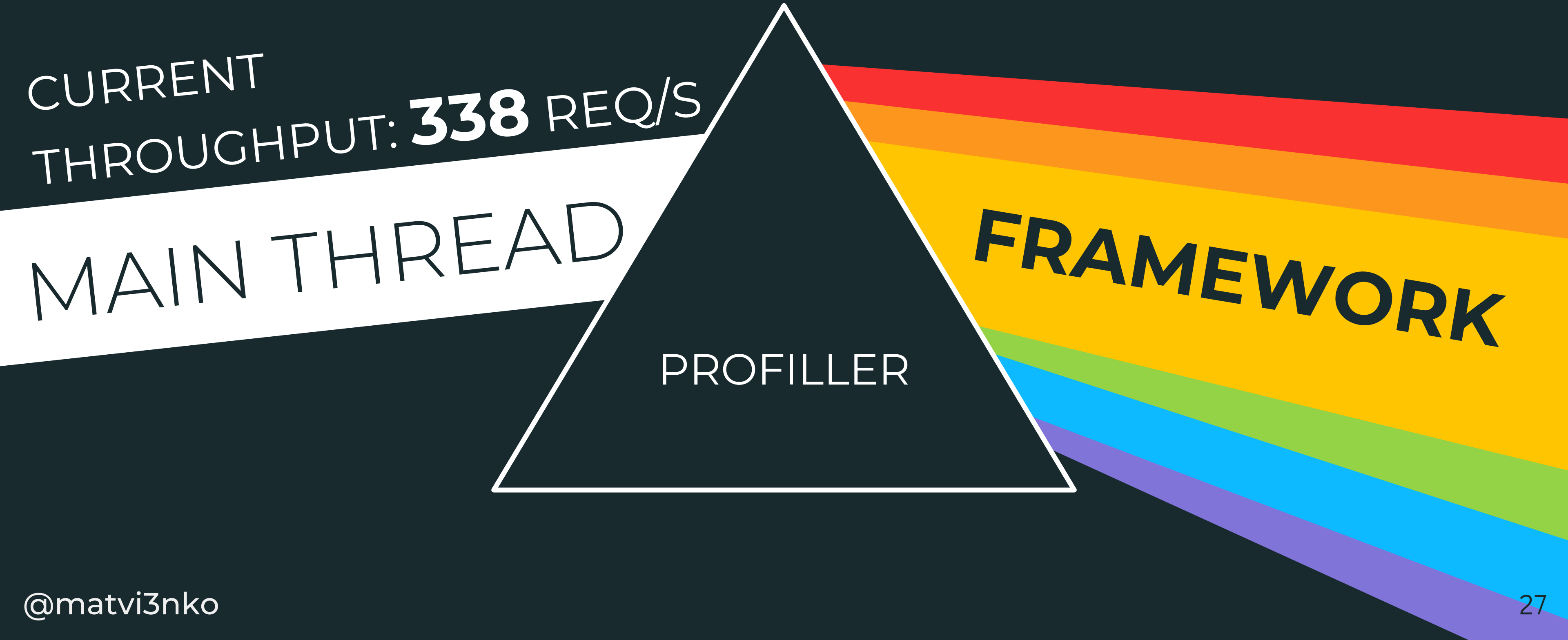
WITH OFF-PROCESS (PARALLEL) JS COMPUTATION



@matvi3nko

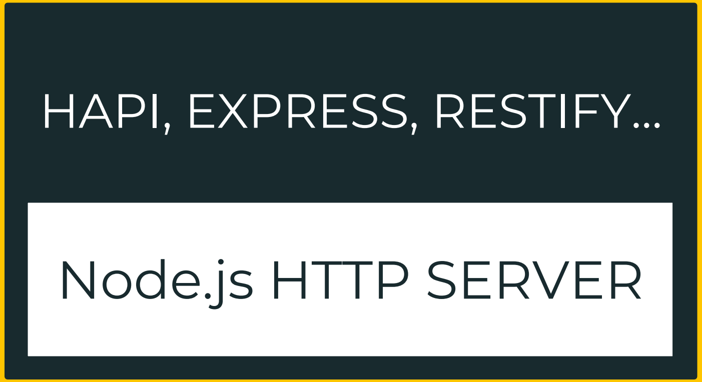
FRAMEWORK

CURRENT THROUGHPUT: **338** REQ/S



FRAMEWORK

HELLO WORLD



HAPI, EXPRESS, RESTIFY...

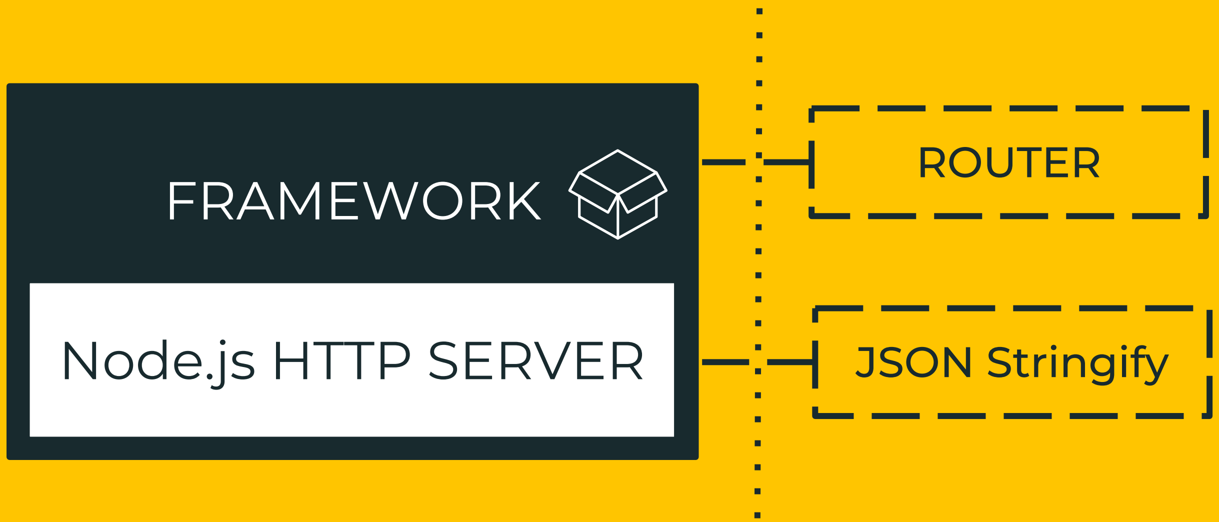
Node.js HTTP SERVER

~ 1.5 – 2X SLOWLY
than http.createServer
https://github.com/fastify/fast-json-stringify

OPTIMIZATIONS

1. Express
https://github.com/expressjs/express

2. Fastify
https://github.com/fastify/fastify



X5 faster
https://github.com/delvedor/router-benchmark

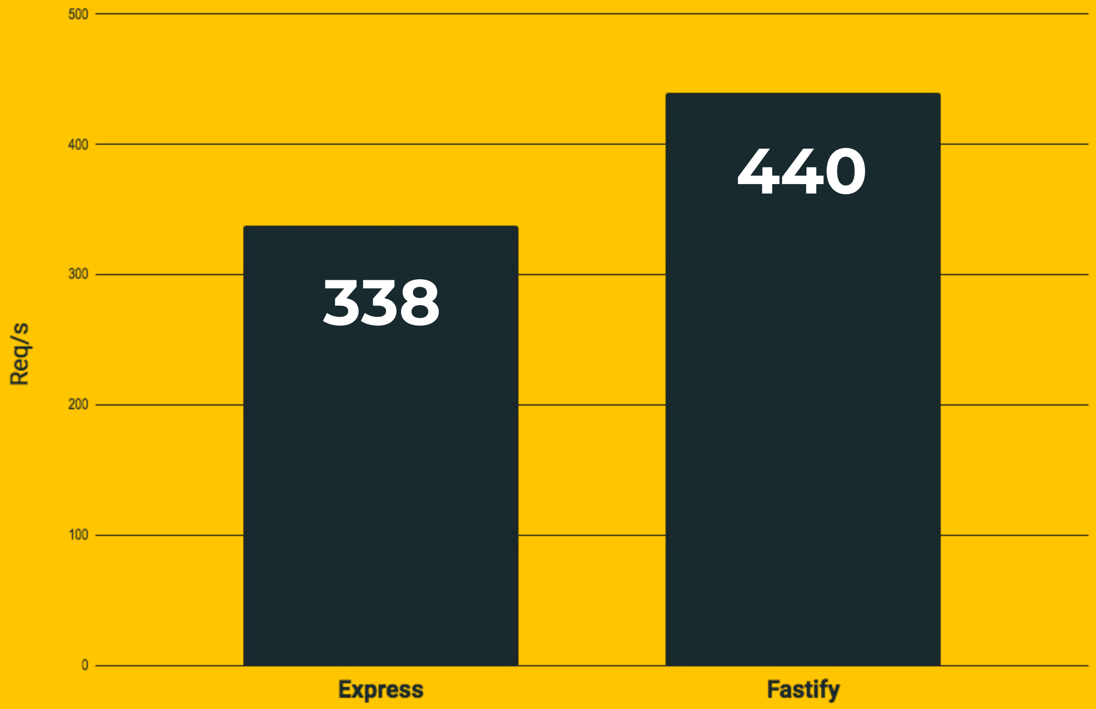
X2-3 faster
https://github.com/fastify/fast-json-stringify

LONG SERIALIZATION

FRAMEWORK CHANGE RESULT

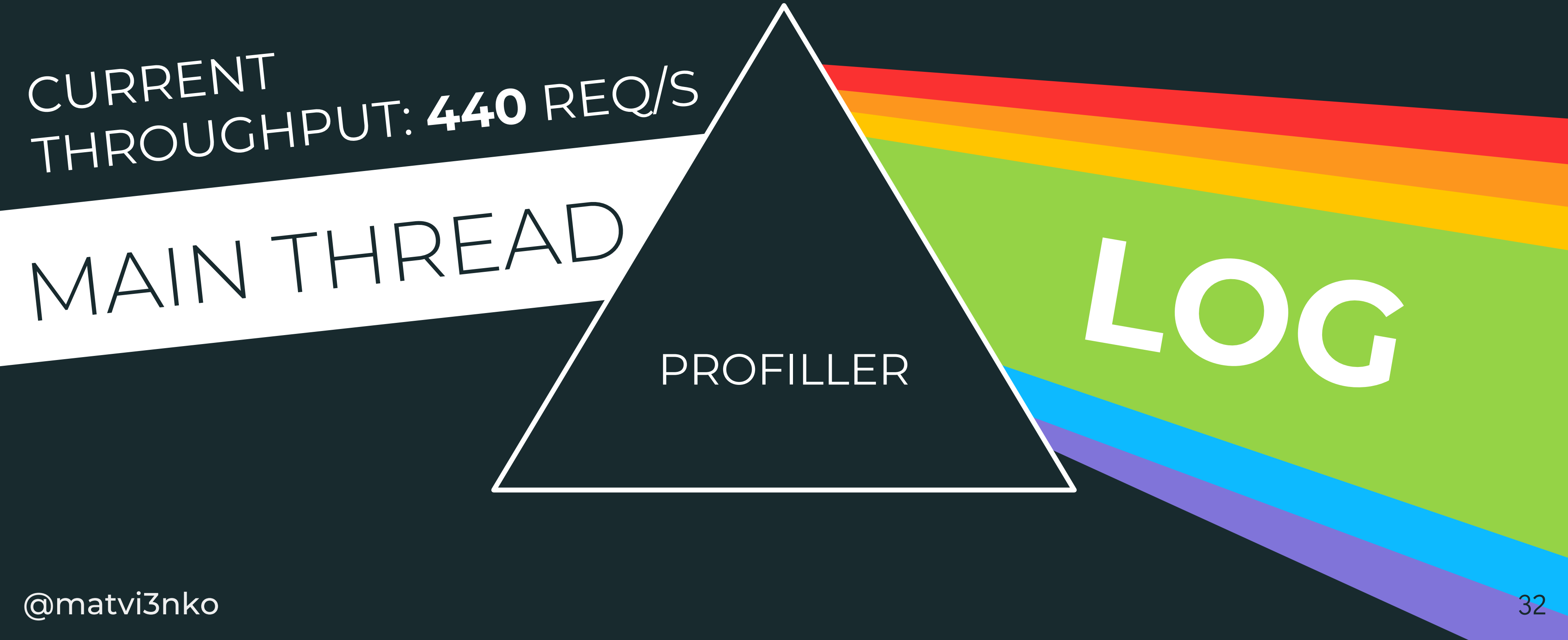
+30%

MORE REQ/S
WITH FASTIFY

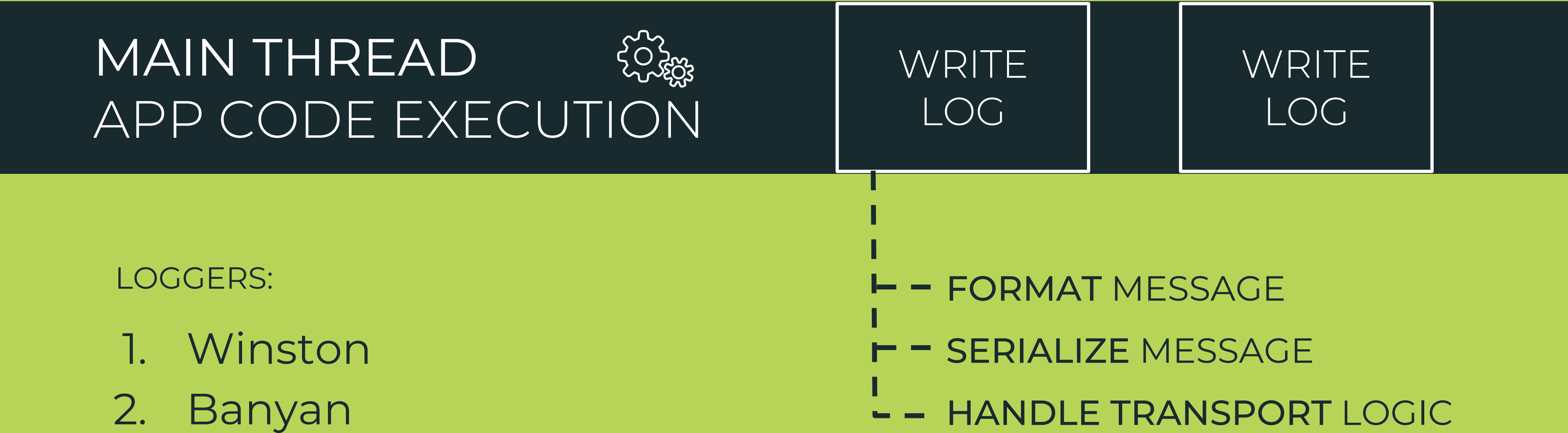


LOGGING

CURRENT THROUGHPUT: **440** REQ/S



LOGGING



MAIN THREAD
APP CODE EXECUTION

WRITE LOG

WRITE LOG

FORMAT MESSAGE

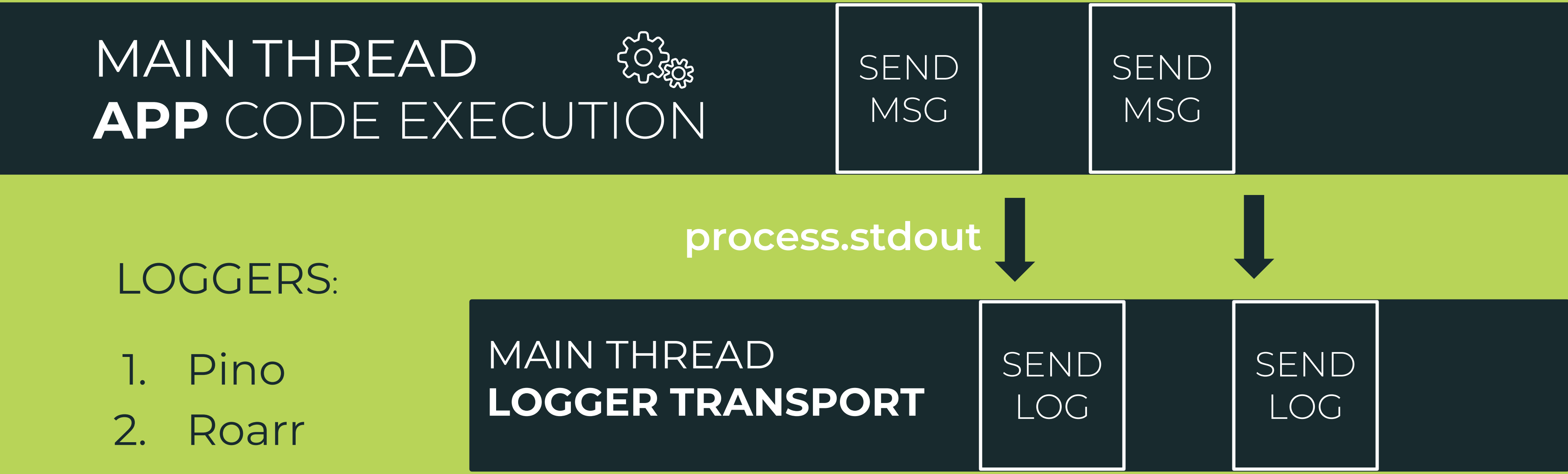
SERIALIZE MESSAGE

HANDLE TRANSPORT LOGIC

LOGGERS:

1. Winston
2. Banyan
3. Morgan and others

OFF-PROCESS LOGGER TRANSPORT



MAIN THREAD
APP CODE EXECUTION

SEND MSG

SEND MSG

process.stdout

MAIN THREAD
LOGGER TRANSPORT

SEND LOG

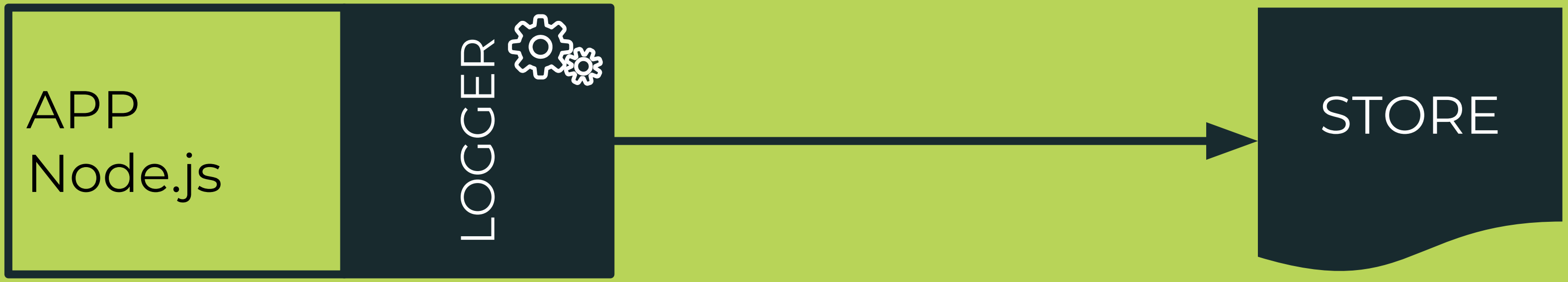
SEND LOG

LOGGERS:

1. Pino
2. Roarr

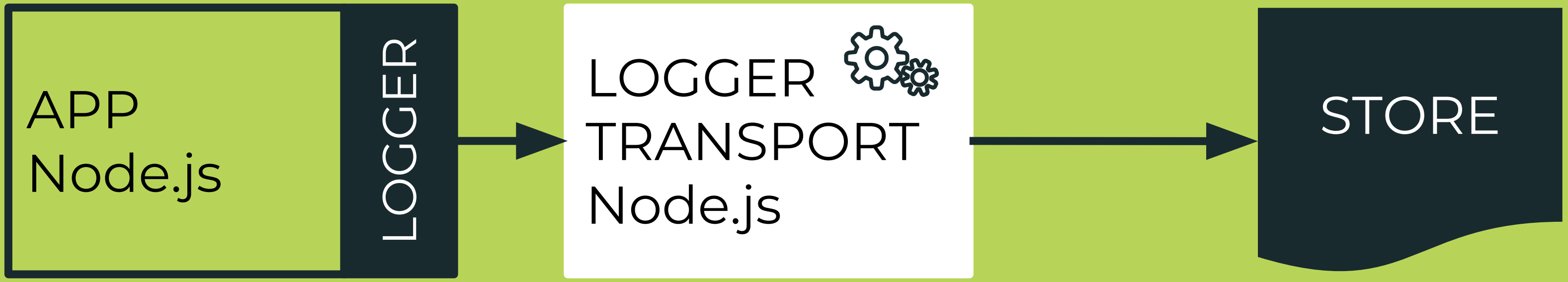
IN-PROCESS LOGGING

PERFORMANCE OVERHEAD: **27%**

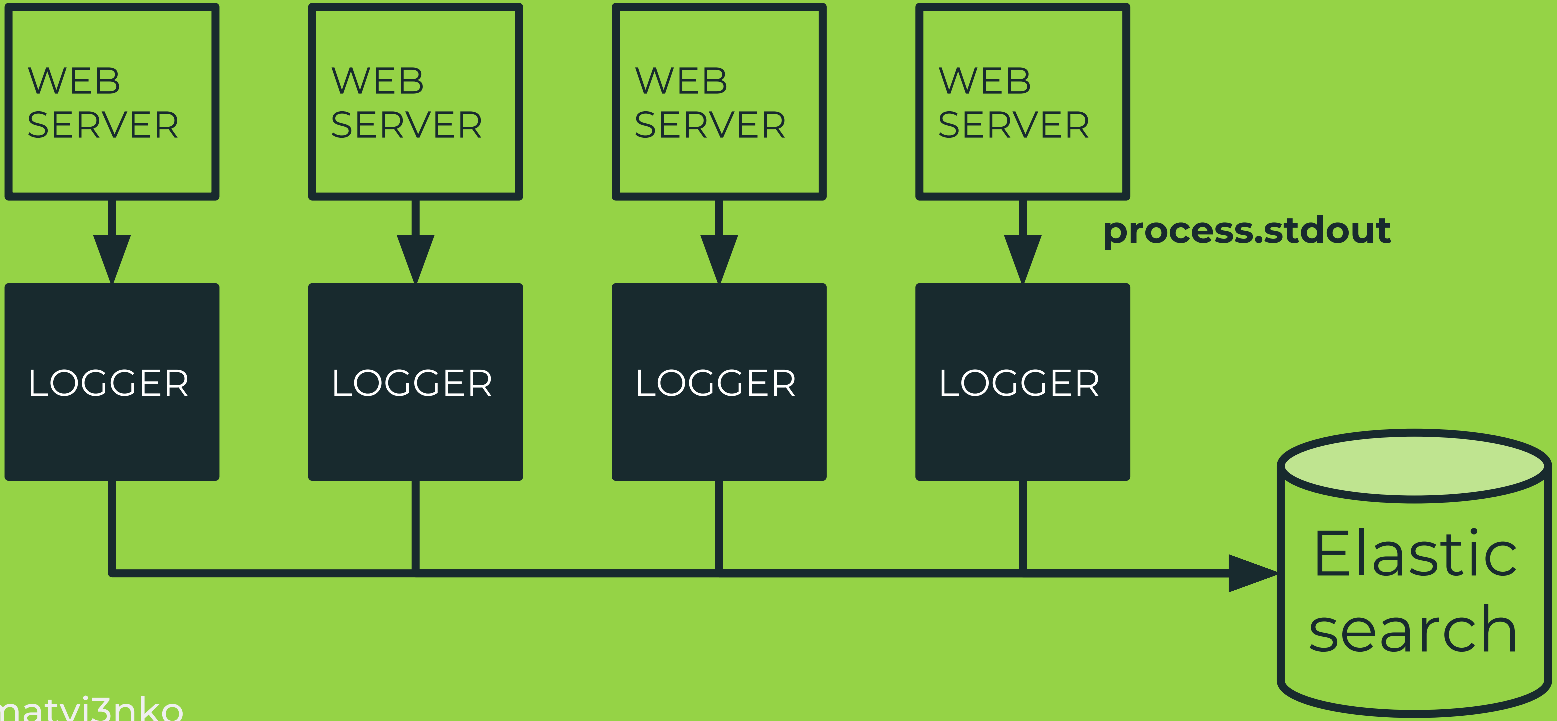


OFF-PROCESS LOGGING

PERFORMANCE OVERHEAD: **3%**



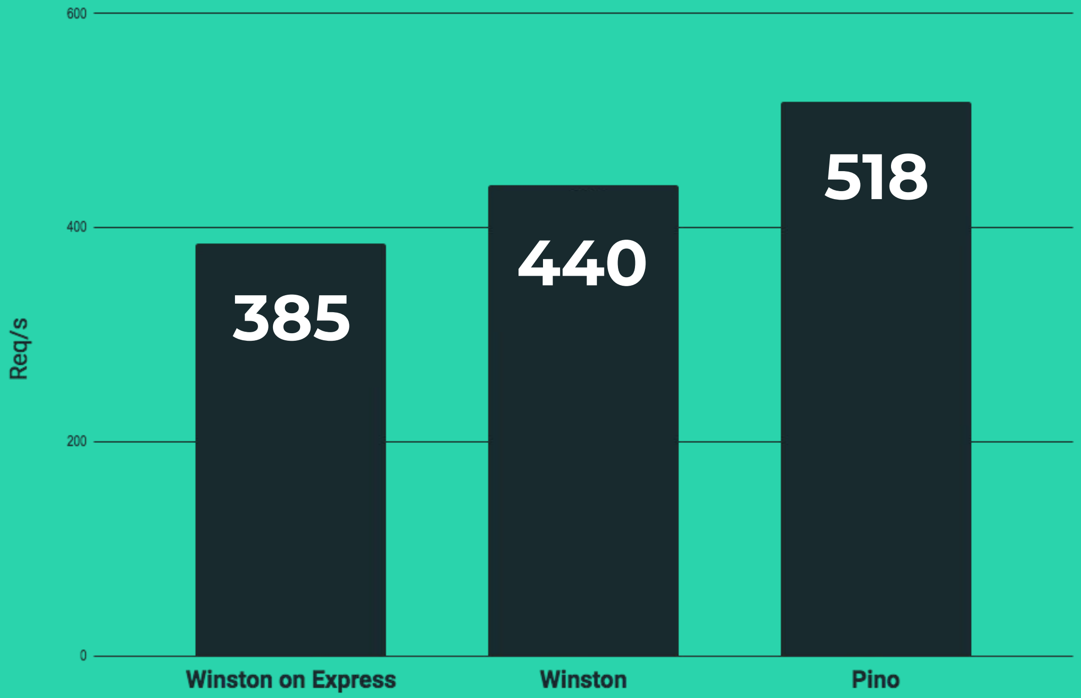
CLUSTER


WEB SERVER
WEB SERVER
WEB SERVER
WEB SERVER
process.stdout
LOGGER
LOGGER
LOGGER
LOGGER
Elastic search

OFF-PROCESS LOGGING RESULT

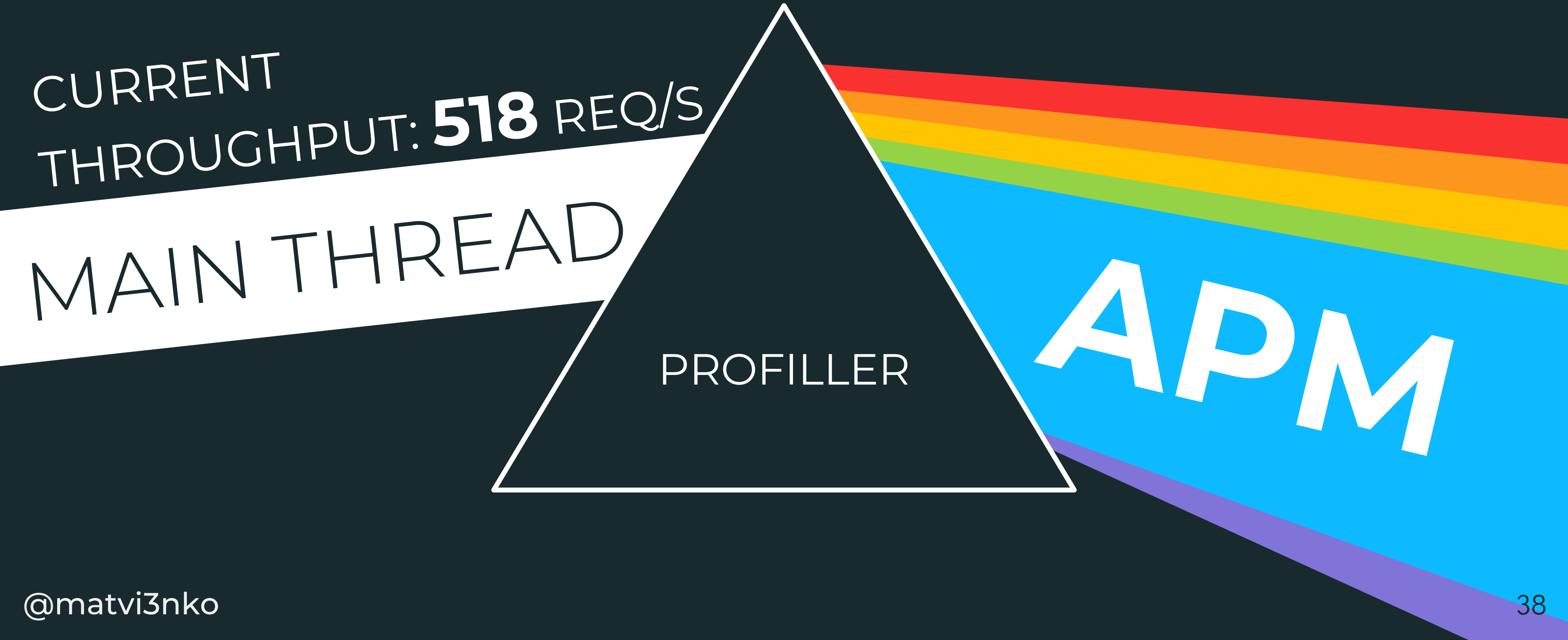
+17%

MORE REQ/S WITH OFF-PROCESS LOGGER TRANSPORT



@matvi3nko

APPLICATION PERFORMANCE MONITORING



CURRENT THROUGHPUT: **518** REQ/S

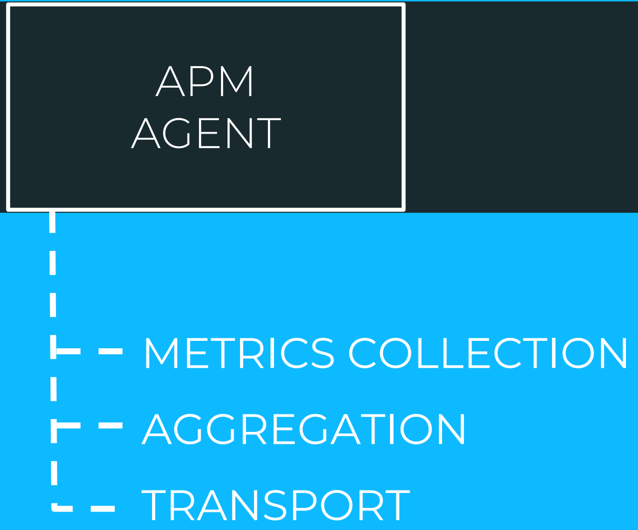
MAIN THREAD

PROFILLER

APM

APPLICATION PERFORMANCE MONITORING

MAIN THREAD
APP CODE EXECUTION



APM
AGENT

METRICS COLLECTION

AGGREGATION

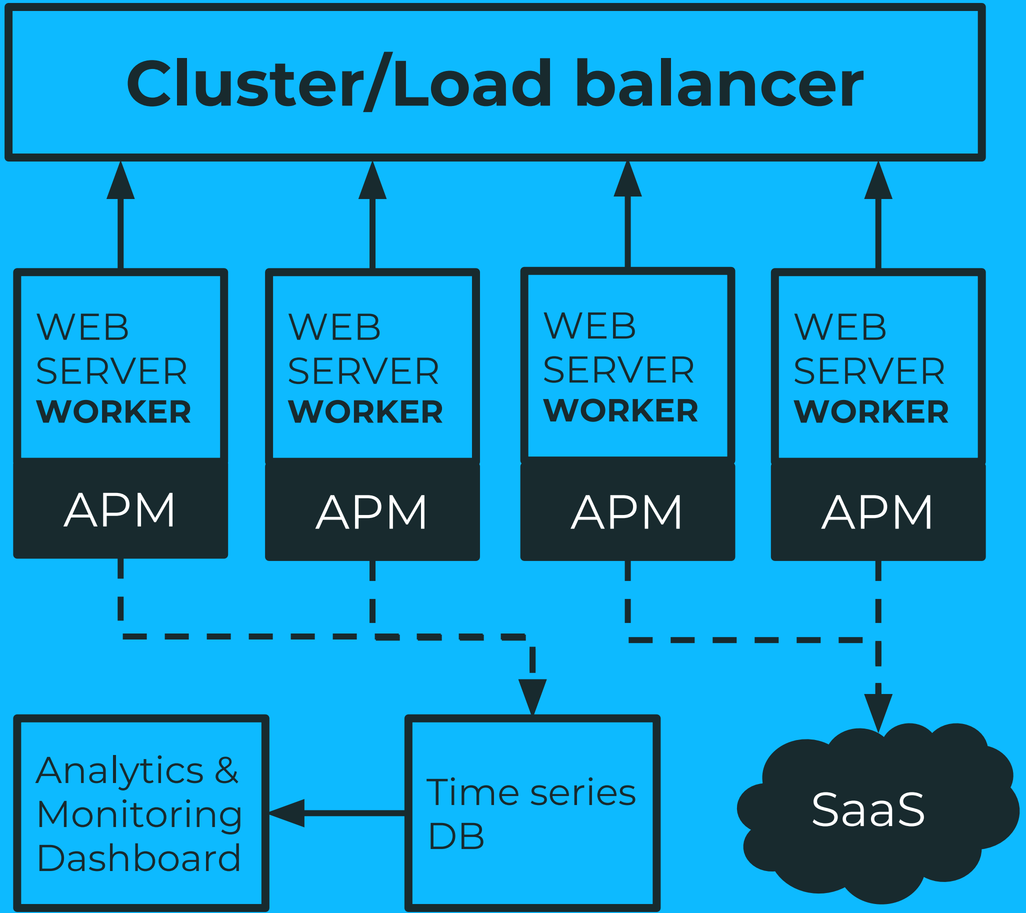
TRANSPORT

APM vendors/agents:

1. NewRelic
2. Dynatrace
3. OpenTracing
4. node-measured

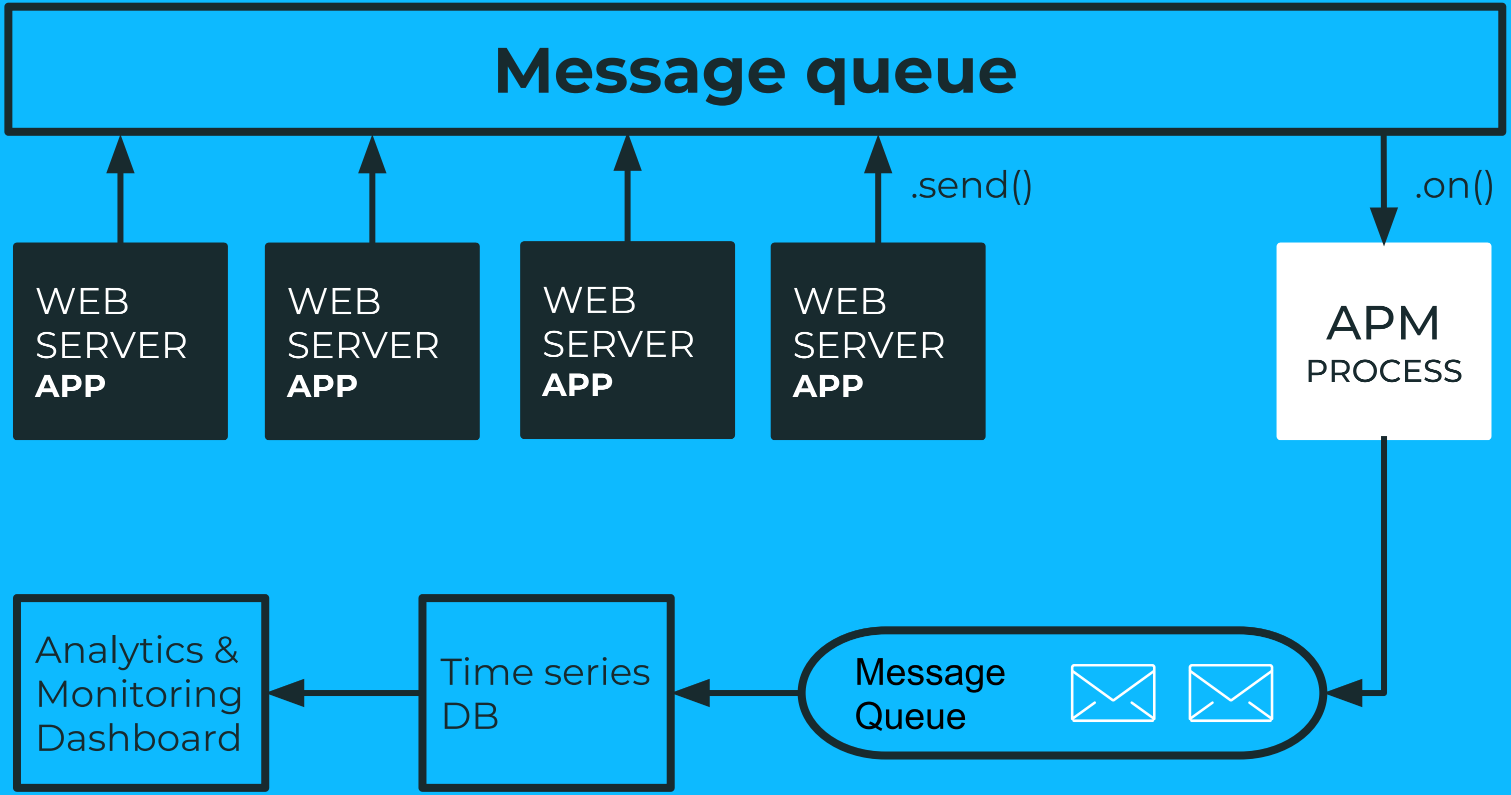
IN-PROCESS APM AGENT

THE APM AGENT PROBLEMS ARE APPLICATION PROBLEMS



@matvi3nko

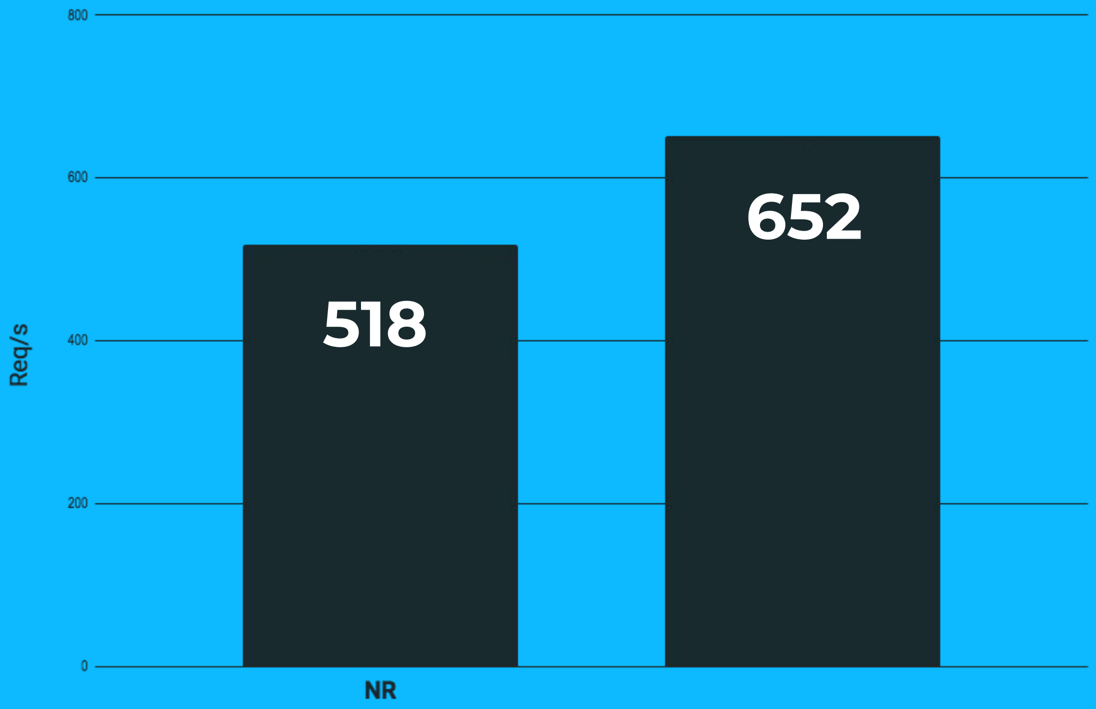
OFF-PROCESS APM AGENT



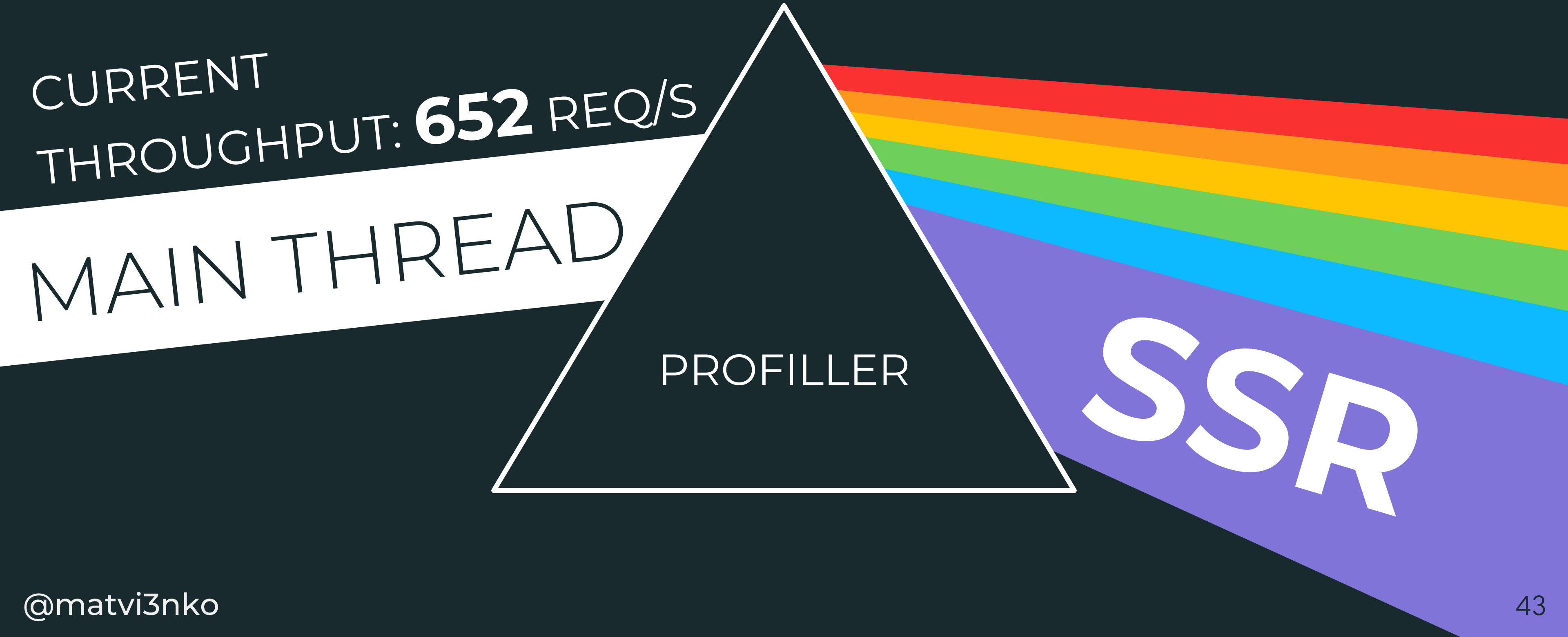
OFF-PROCESS MONITORING RESULT

+25%

MORE REQ/S
WITH
OFF-PROCESS
METRIC AGENT



CURRENT THROUGHPUT: **652** REQ/S



MAIN THREAD

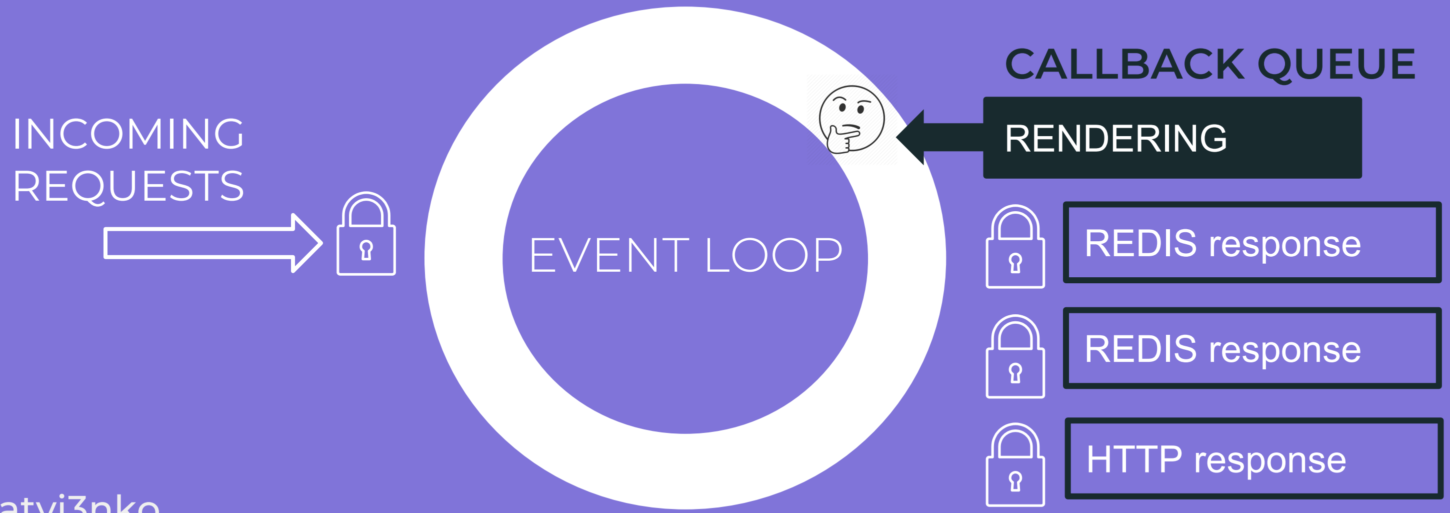
PROFILLER

SSR

SERVER-SIDE RENDERING

MAIN THREAD
APP CODE EXECUTION

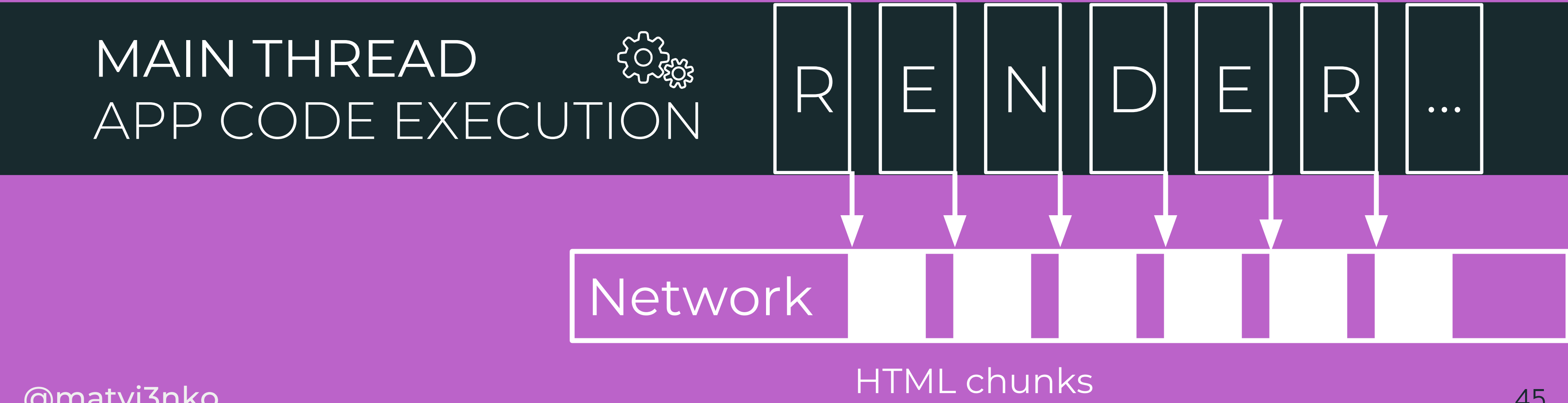
RENDERING



STREAMING SERVER-SIDE RENDERING

```
renderStream.pipe(res, { end: 'false' });
renderStream.on('end', () =>
{response.end('</div></body></html>'); });
```

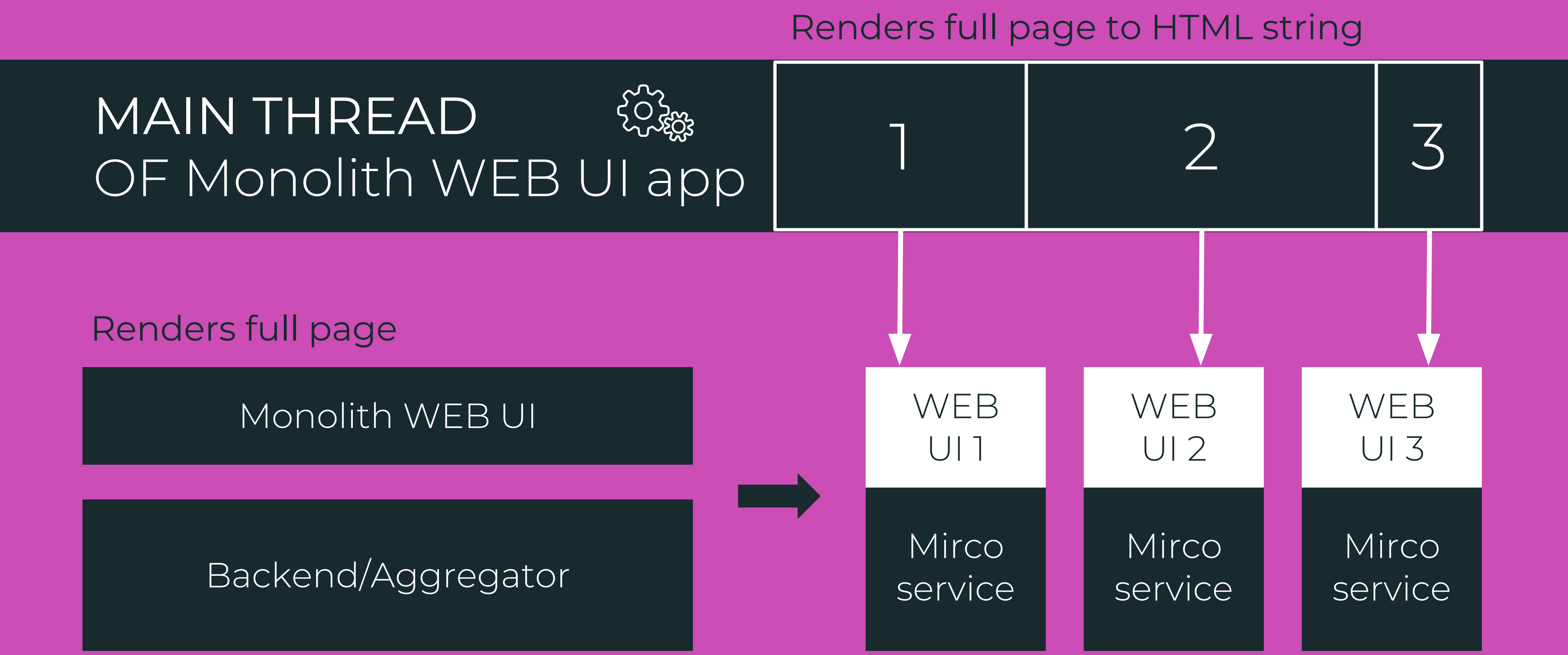
Asynchronous execution in STREAM with **REACT 16**



MAIN THREAD
APP CODE EXECUTION

HTML chunks

MICRO FRONTENDS



Renders full page to HTML string

MAIN THREAD
OF Monolith WEB UI app

1

2

3

Renders full page

Monolith WEB UI

Backend/Aggregator

WEB
UI 1

Mirco
service

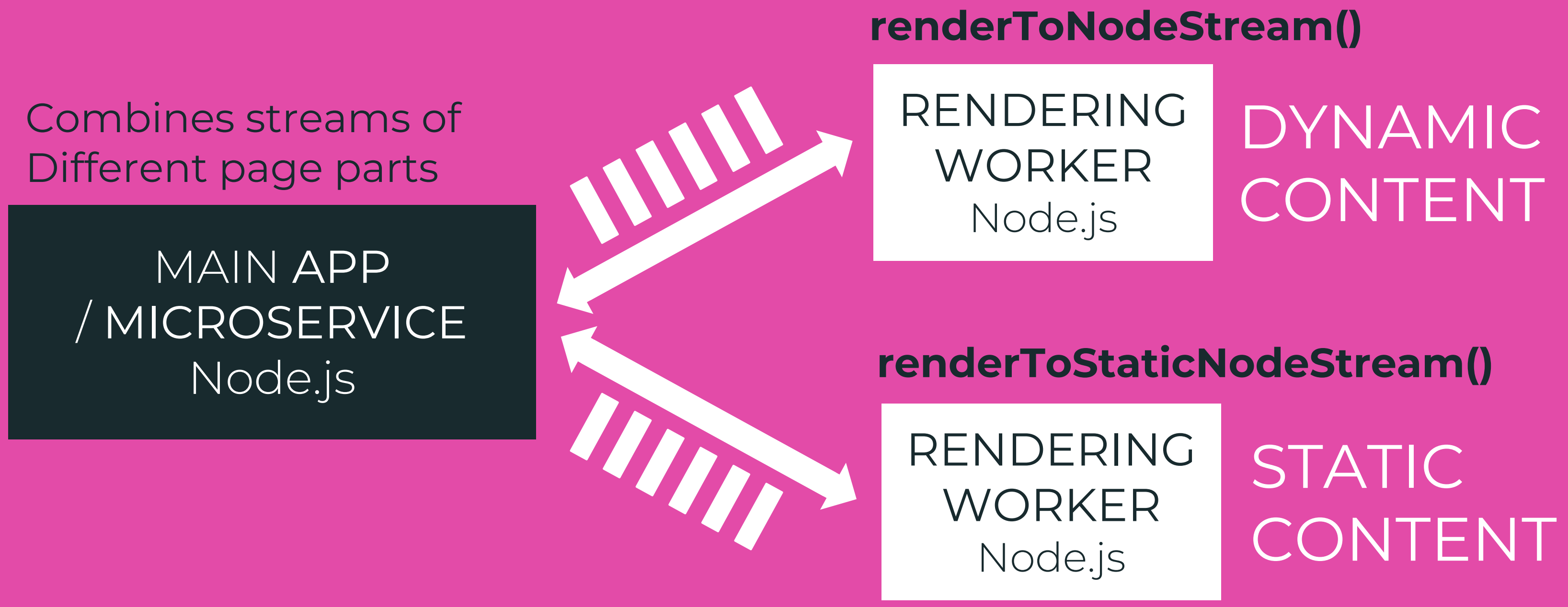
WEB
UI 2

Mirco
service

WEB
UI 3

Mirco
service

PARALLEL RENDERING WITH WORKERS



Combines streams of Different page parts

MAIN APP / MICROSERVICE Node.js

renderToNodeStream()

RENDERING WORKER Node.js

DYNAMIC CONTENT

renderToStaticNodeStream()

RENDERING WORKER Node.js

STATIC CONTENT

REACT 16

2X

* Average value.

THROUGHPUT

FROM

114 REQ/S

TO 652

ORDER?

MULTY

THREADING

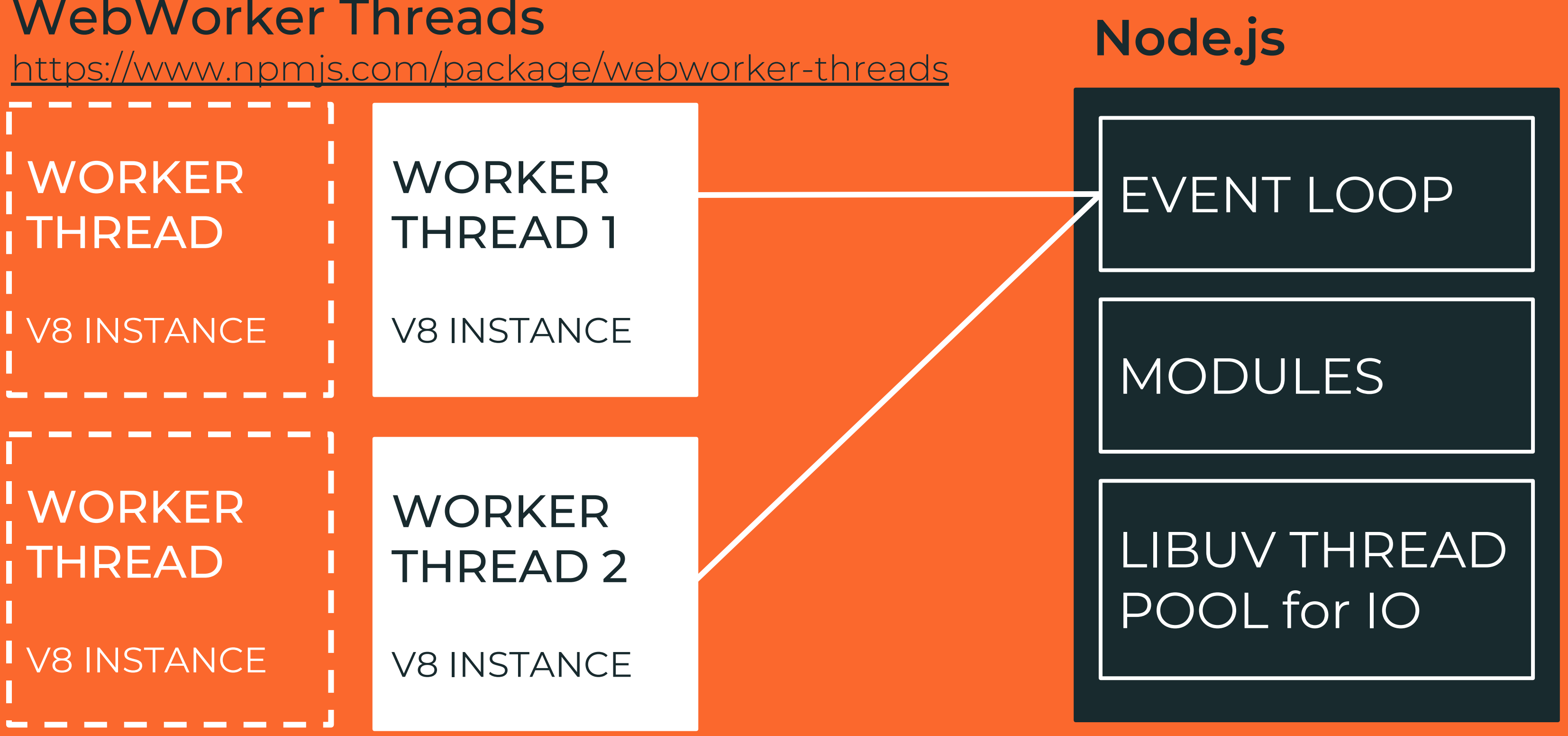
IN NODE.JS

Hhheeellllooo
Wwoorrlldd

WEBWORKER THREADS

WebWorker Threads

https://www.npmjs.com/package/webworker-threads



MICROSOFT NAPA.JS

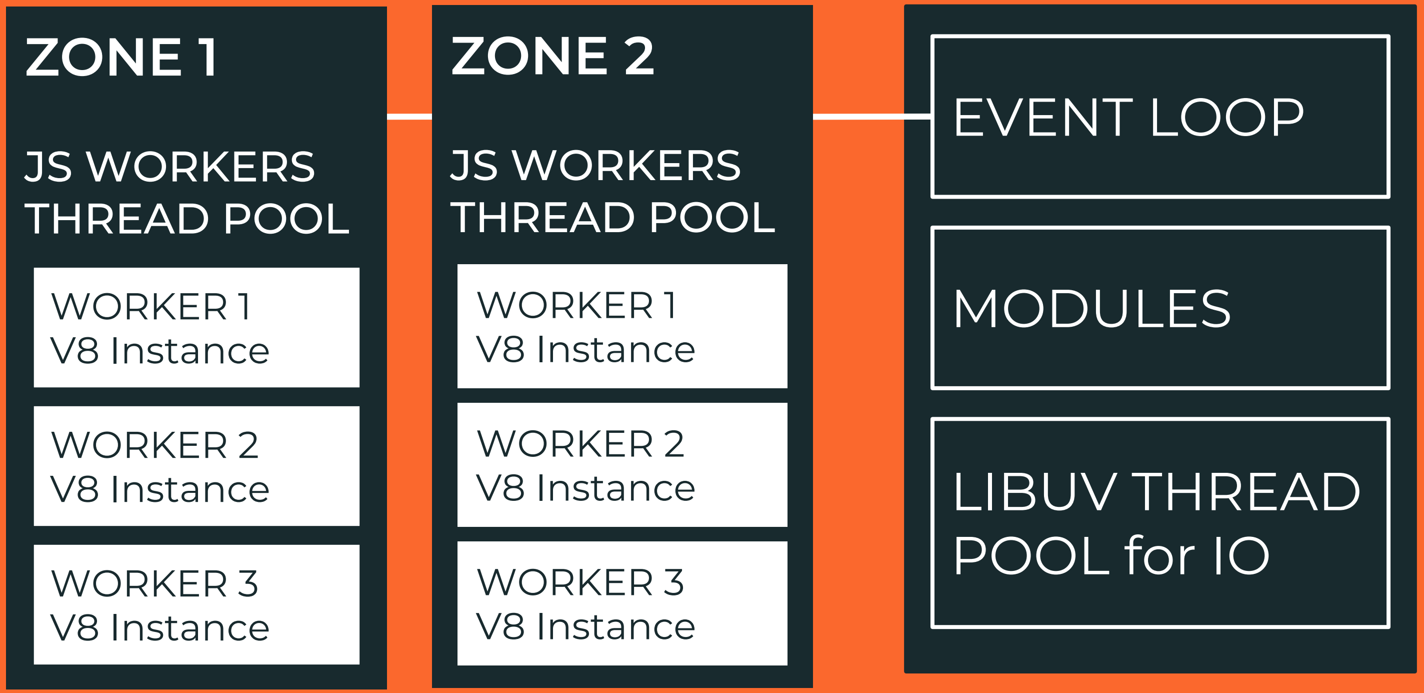
MESSAGE PASSING
2x vs IPC

MEMORY USAGE
6.7 MB
vs 8 MB

STARTUP TIME
50 ms
vs 70 ms

Napa.js
https://github.com/Microsoft/napajs

Node.js



ZONE 1

JS WORKERS THREAD POOL

WORKER 1
V8 Instance

WORKER 2
V8 Instance

WORKER 3
V8 Instance

ZONE 2

JS WORKERS THREAD POOL

WORKER 1
V8 Instance

WORKER 2
V8 Instance

WORKER 3
V8 Instance

EVENT LOOP

MODULES

LIBUV THREAD POOL for IO

ALIBABA ALIOS

SHARED
GLOBAL
MEMORY

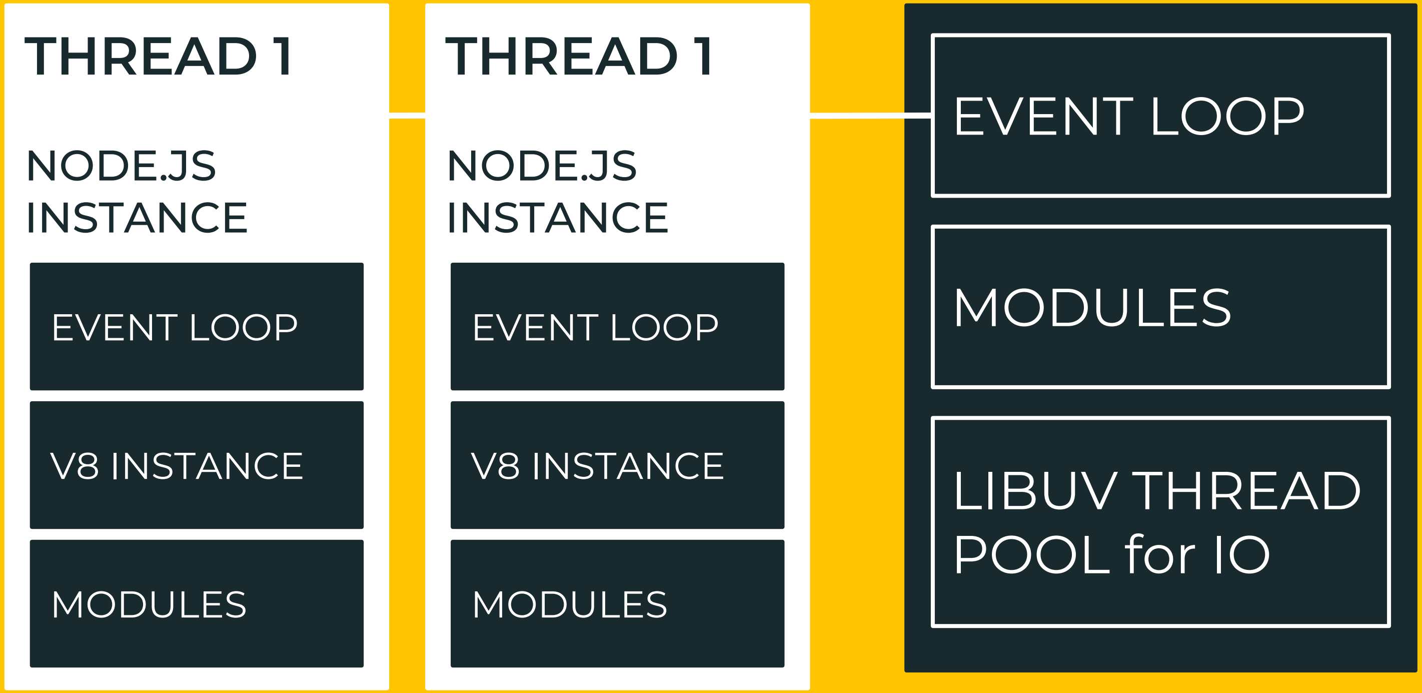
MEMORY
USAGE
2.5 MB
vs 8 MB

STARTUP
TIME
13 ms
vs 70 ms

ALiOS-node.js

https://github.com/alibaba/AliOS-nodejs

Node.js



THREAD 1

NODE.JS
INSTANCE

EVENT LOOP

V8 INSTANCE

MODULES

THREAD 1

NODE.JS
INSTANCE

EVENT LOOP

V8 INSTANCE

MODULES

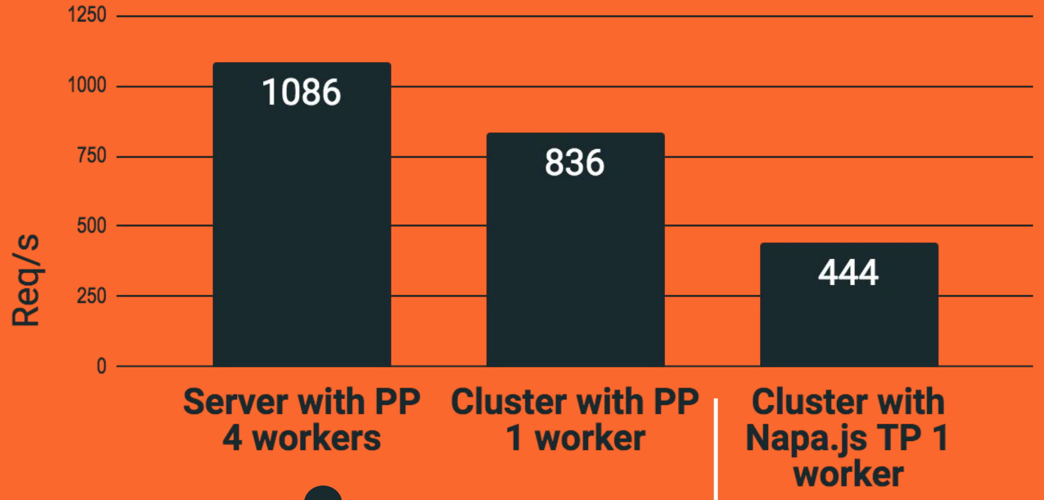
EVENT LOOP

MODULES

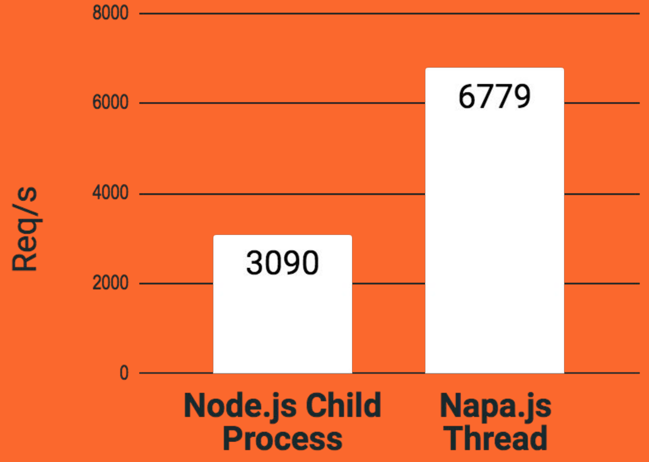
LIBUV THREAD
POOL for IO

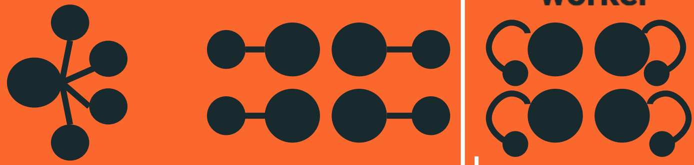
PROCESS/THREAD POOL ?

THROUGHPUT of CPU tasks execution



MESSAGE PASSING



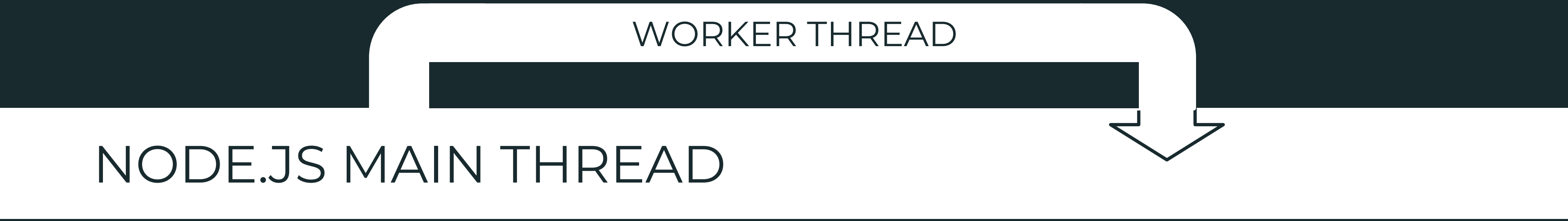


-20%

-10% with bigger message transfer

NEW WORKER API IN NODE.JS ?

TO PERFORM CPU TASKS

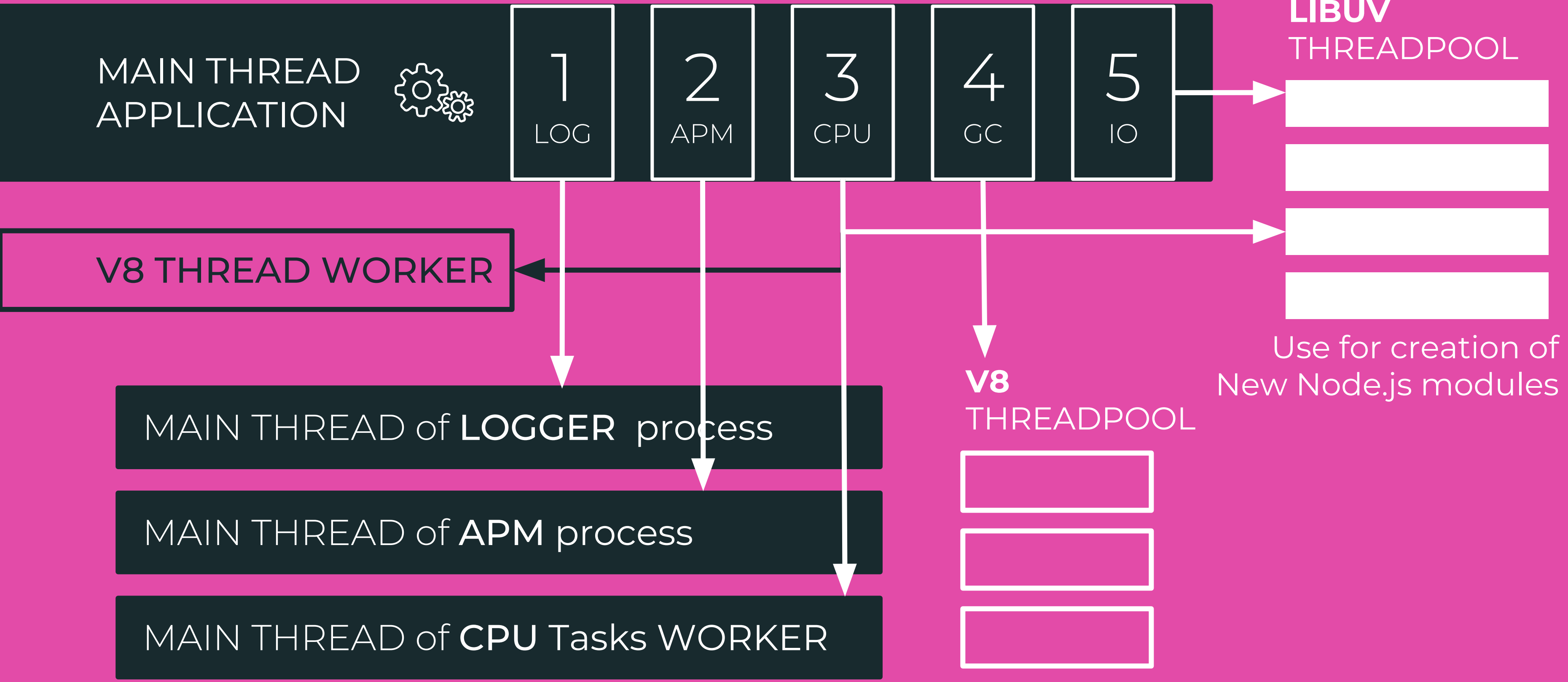


WORKER THREAD

NODE.JS MAIN THREAD

@matvi3nko

DECOMPOSED MAIN THREAD



MAIN THREAD APPLICATION

1 LOG

2 APM

3 CPU

4 GC

5 IO

LIBUV THREADPOOL

Use for creation of New Node.js modules

V8 THREAD WORKER

V8 THREADPOOL

MAIN THREAD of **LOGGER** process

MAIN THREAD of **APM** process

MAIN THREAD of **CPU** Tasks WORKER

@matvi3nko

FINALY 6X REQ/S THROUGHPUT

REFERENCES

Long-running Background Process in Node.js

https://vimeo.com/229536743

Background tasks in Node.js

https://www.youtube.com/watch?v=NNTsHzER31I&t=2207s

https://blog.evantahler.com/background-tasks-in-node-js-a-survey-with-redis-971d3575d9d2

Streaming Server-Side Rendering and Caching

https://zeit.co/blog/streaming-server-rendering-at-spectrum

https://github.com/zalando/tailor

Microservices on UI

https://www.youtube.com/watch?v=3l9IP9j5n1o

https://www.youtube.com/watch?v=E6_UyQPmiSg&t=2997s

REFERENCES

New Garbage Collection with threads

https://v8project.blogspot.ru/2017/11/

https://v8project.blogspot.com/2016/04/jank-busters-part-two-orinoco.html

Pino

https://github.com/pinojs/pino

New Worker API in Node.js discussion

https://github.com/nodejs/worker/issues/4

IPC Communication Performance
https://60devs.com/performance-of-inter-process-communications-in-nodejs.html

List of Parallel JS Projects

https://github.com/SyntheticSemantics/List-of-Parallel-JS-Projects

REFERENCES

https://github.com/nickkooper/Decomposition-of-the-Main-Thread-in-Node.js

THANKS!

Nikolay Matvienko

matvi3nko@gmail.com

Twitter.com/**matvi3nko**

github.com/nickkooper