What I Learn About NodeJS During Write Forwarding Benchmarking

Amazon Aurora

by naor tedgi (Abu Emma)

The Goal:

i want to determine if we can move all our services to us and using write forwarding to write data from all our applications

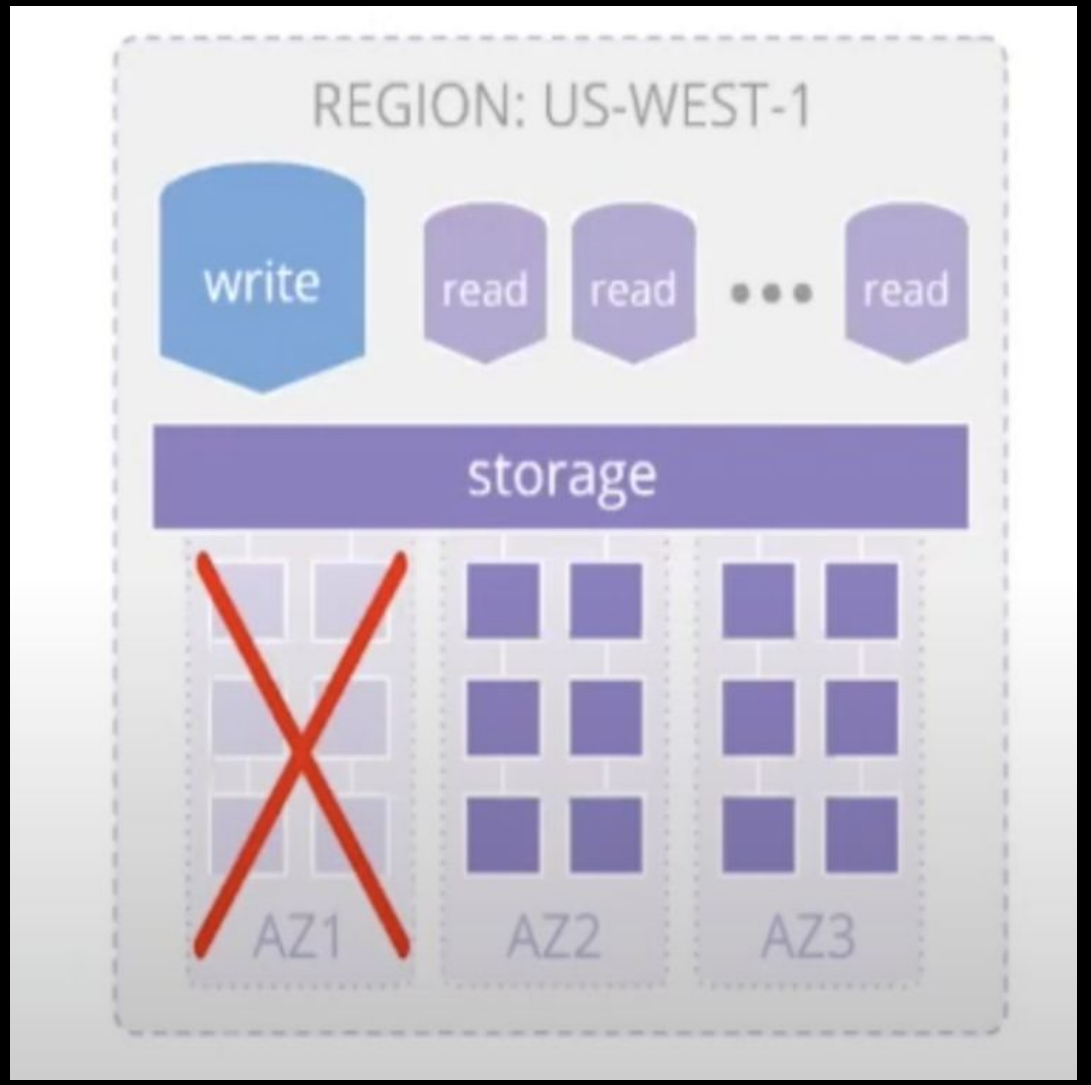
- as you know we really wants to move all our infrastructure to US
- Today our Mysql Writer Located in EU and shared with supply and delivery
- today the services deployed in US **(offline-js)** using Mysql readers located in US to read and Write-Forwarding in order to write

what is write Forwarding?

- write layer, read layer, storage layer
- qurum reading

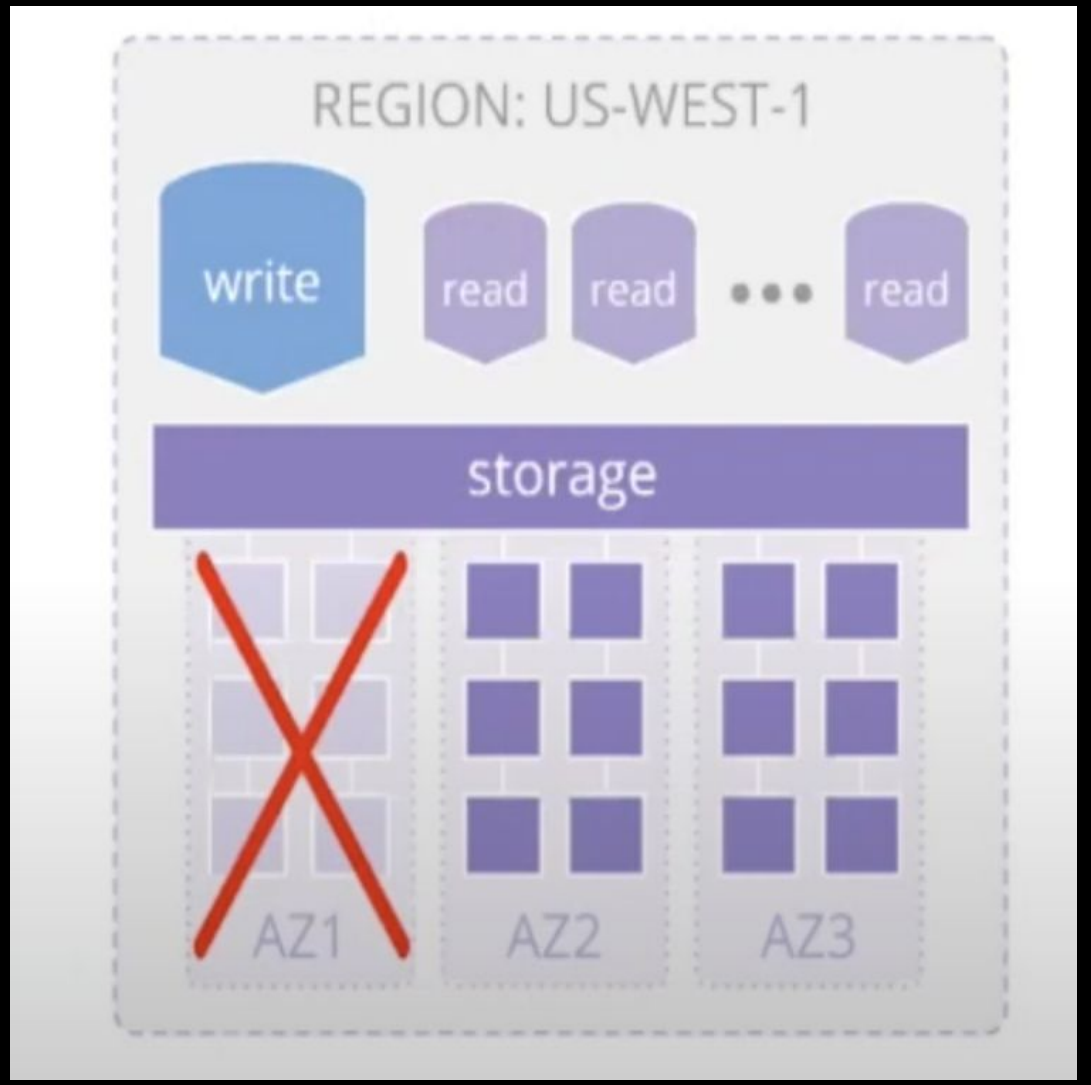
what is write Forwarding?

- write layer, read layer, storage layer
- qurum reading



what is write Forwarding?

- write layer, read layer, storage layer
- qurum reading



what is write Forwarding?

your applications can simply send both read and write requests to a reader in a secondary Region, and Global Database will take care of forwarding the write requests to the writer in the primary Region.

```
`SET aurora_replica_read_consistency = GLOBAL`
```

Benchmarking - TRY 0

1. Make the simplest code you can to simultate your case
   - 100 insert , 100 Delete , 100 Update , 100 select
   - update a map with counter that calculate how many queries run
     and how much time it takes for example **{delete:{total:50:totalTime:1342s}}**
     using Event Emitter

2. Run code multiple times
3. Always calculate the average of X Runs and not single
4. Repeat X times

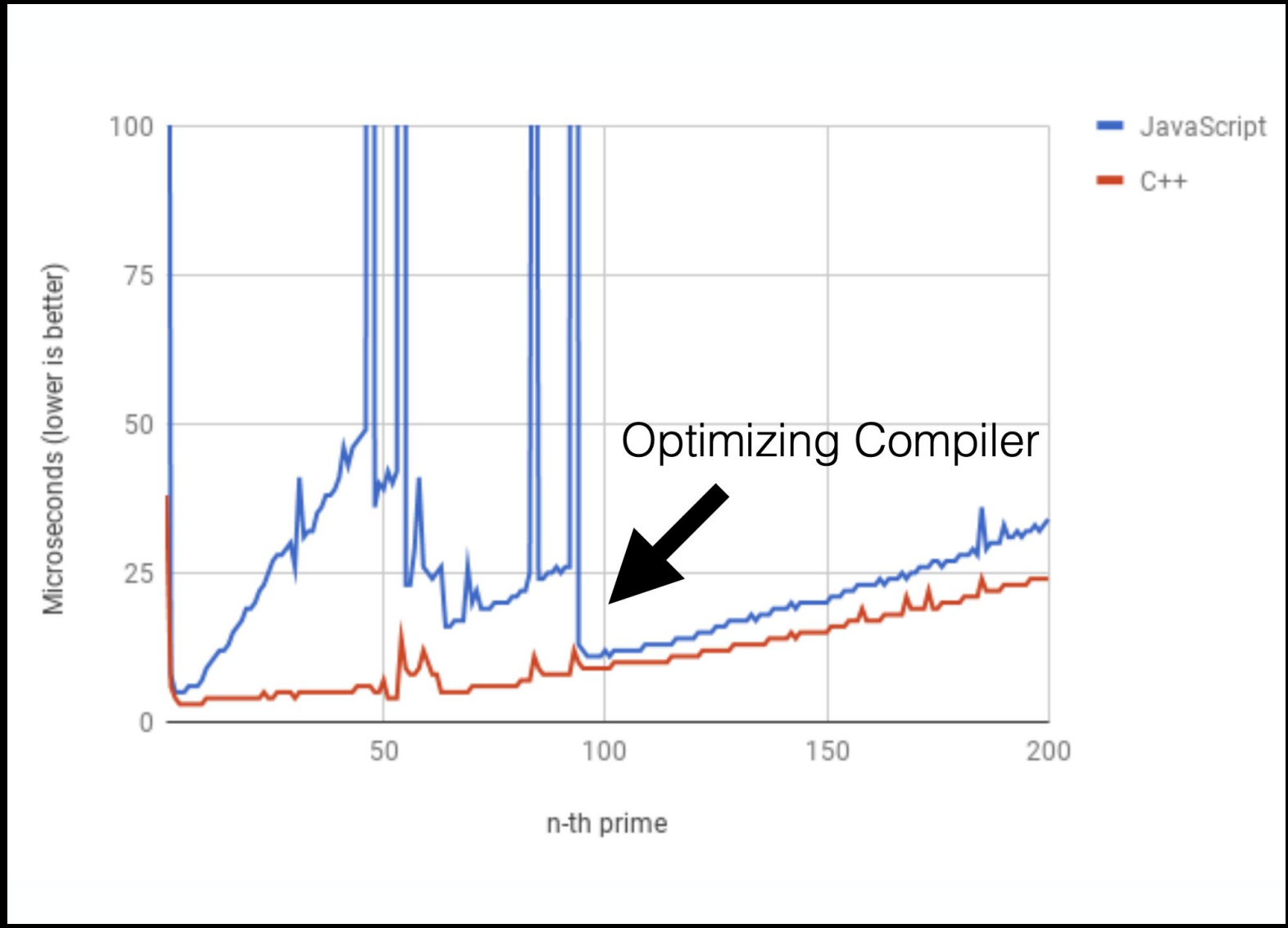
run from my local PC

Run code multiple times **JIT COMPILER** (**Run: node --print-opt-code example.js**)

```
const ITRATIONS = 200000

function myFunc(obj) {
 return obj.x ;
}

const obj1 = { x: 1 }

for (let i = 0; i < ITERATIONS; ++i) {
 myFunc({ ...obj1 });
}
```

```
--- Optimized code ---
optimization_id = 2
source_position = 73
kind = TURBOFAN
name = myFunc
stack_slots = 6
compiler = turbofan
address = 0x10face601

Instructions (size = 204)
0x10face680     0  488b59a0            REX.W movq rbx,[rcx-0x60]
0x10face684     4  f6430f01            testb [rbx+0xf],0x1
0x10face688     8  7405                jz 0x10face68f  <+0xf>
0x10face68a     a  e9718cc3fa          jmp 0x10a707300  (CompileLazyDeoptimizedCode)    ;; r
untime entry
0x10face68f     f  55                  push rbp
0x10face690    10  4889e5              REX.W movq rbp,rsp
0x10face693    13  56                  push rsi
0x10face694    14  57                  push rdi
0x10face695    15  50                  push rax
0x10face696    16  4883ec08            REX.W subq rsp,0x8
0x10face69a    1a  488975e0            REX.W movq [rbp-0x20],rsi
0x10face69e    1e  493b65a0            REX.W cmpq rsp,[r13-0x60] (external value (StackGuard
::address_of_jslimit()))
```

Run code multiple times



The reason : Shape File Cache And Hidden function, Not going to dive in only if you want me to

Run code multiple times

C++ AddOns , WebAssembly packages takes a lot of time to load

TRY 0 : conclusions - Failure (WF is not stable)

- after 10-15 seconds A lot of queries failed with error timeout and the process crush
- for The queries that work the time was almost X2 in US

The Problem Was with Promise.all()
question what is the order here of printing?

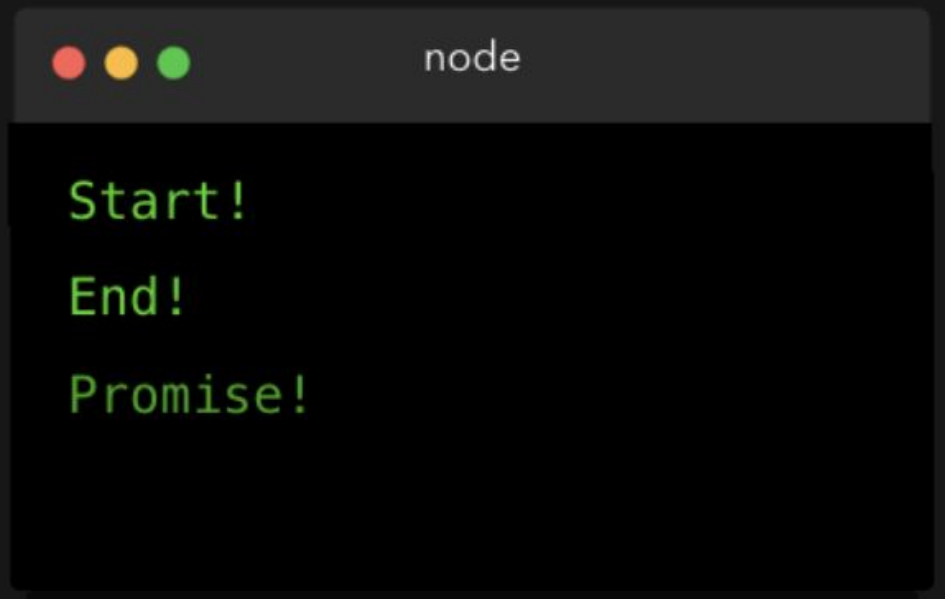
```
console.log('Start!')

Promise.resolve('Promise!')
  .then(res => console.log(res))

console.log('End!')
```

TRY 0 : conclusions - Failure (WF is not stable)

- after 10-15 seconds A lot of queries failed with error timeout and the process crush
- for The queries that work the time was almost X2 from running in from EU

- The Problem Was with Promise.all()

```
console.log('Start!')

Promise.resolve('Promise!')
  .then(res => console.log(res))

console.log('End!')
```

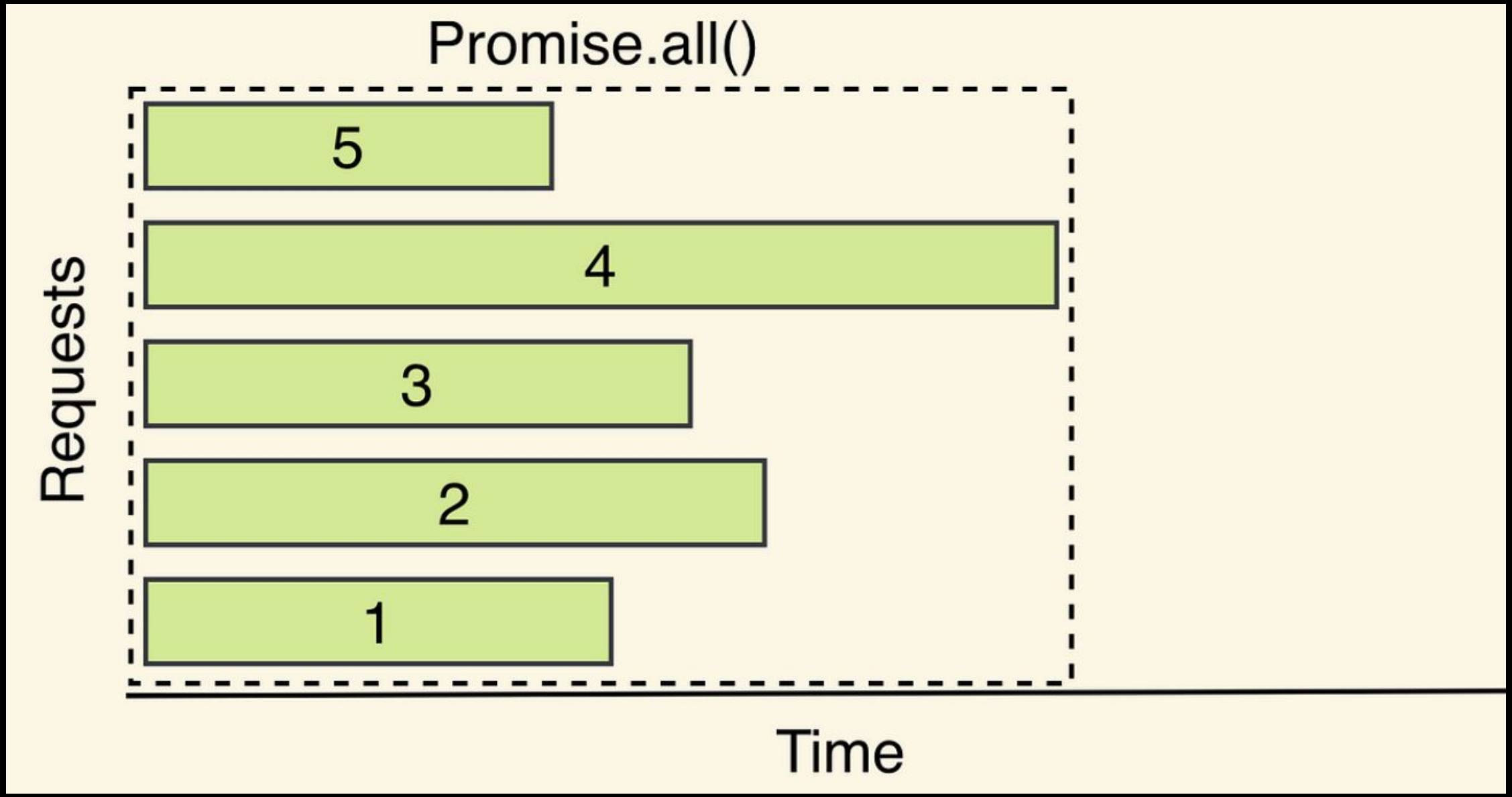


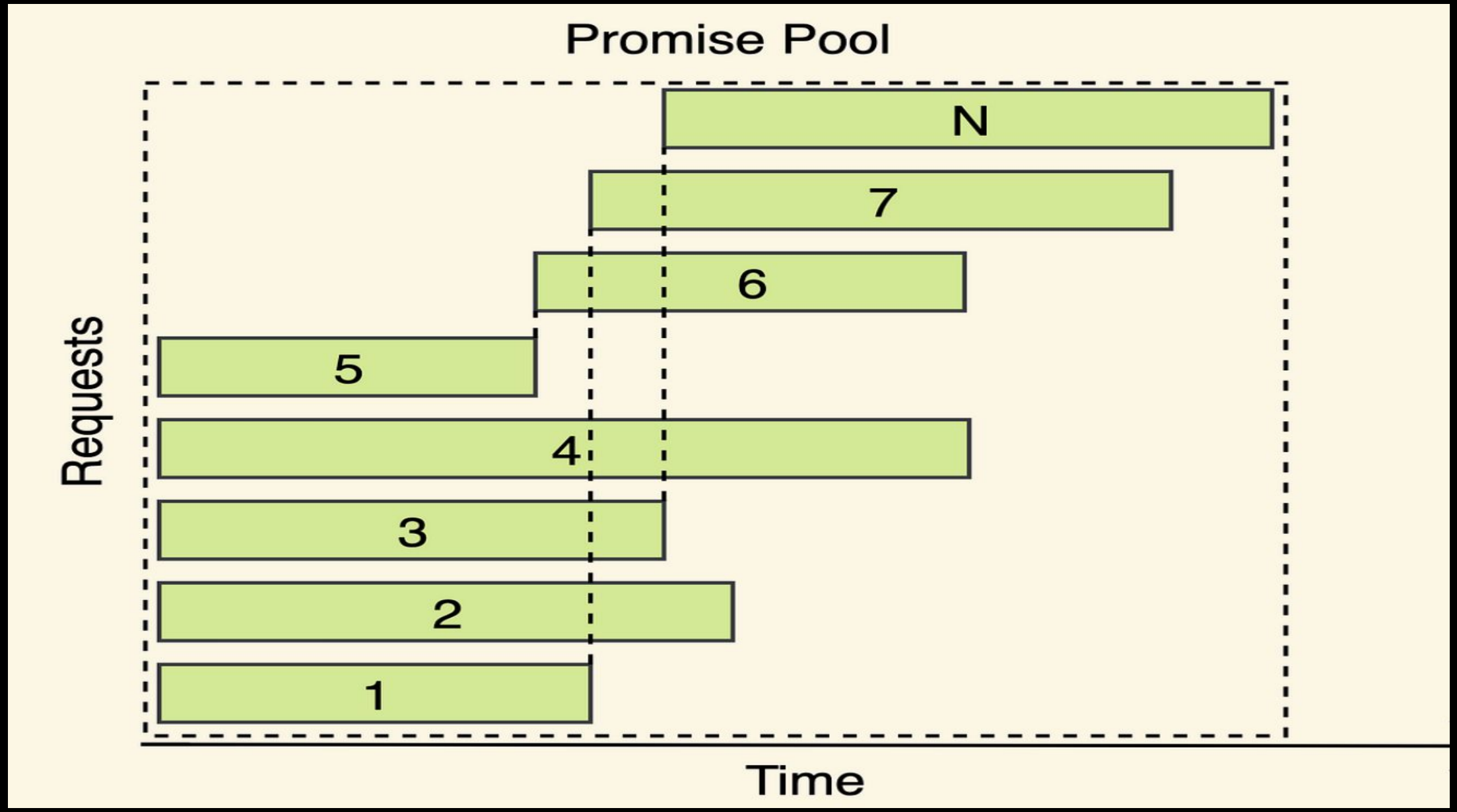
(Macro)task	`setTimeout` | `setInterval` | `setImmediate`
Microtask	`process.nextTick` | `Promise callback` | `queueMicrotask`

1. All functions in that are currently in the call stack get executed. When they returned a value, they get popped off the stack.
2. When the call stack is empty, *all* queued up microtasks are popped onto the callstack one by one, and get executed

when i ran Promise.all in it takes a lot of time to resolve i wasted a lot of time on Garbage collection instead on running the query

i add throttling to the procedure instead of creating 1000 queries at the same time i restrict it only to 10 running together


Promise.all()
Requests
5
4
3
2
1
Time


Promise Pool
N
7
6
5
4
3
2
1
Requests
Time

TRY 1 : conclusions - Stable - but running slow

- the experiment isn't correct because i ran it from my local PC to EU And US
- i saw the difference from eu to us have 2 second for each query

when i write the benchmark i set `aurora_replica_read_consistency`
for every query at the US WF - result is when running the experiment on US
i run 2X queries 1 for the set 1 for execution

i split the mysql connection pool to read and write connection
and set the Wf variable on the query creation

TRY 1 : conclusions - Stable - but running slow

```
const newConnectionHandler = (connection: mysql.PoolConnection, mode: string)=> {
 if (process?.argv[2] === "us" && mode === "write") {
   console.log("set aurora_replica_read_consistency to openXPool connection created");
   connection.query("SET aurora_replica_read_consistency = 'session'");
 }
 console.log("openXPool connection created");
 connection.on("error", (err: any) => {
   console.error("openXPool connection error:", err);
 });
};

writeOpenXPool.on("connection", (connection) => {
 newConnectionHandler(connection, "write");
});
```

TRY 2 :

- i ran the experiment from a machine in US
- i add connection pool for Write and Read
- i remove the first iteration from average because i need to create more connection then in EU and it was 4x time slower than other iterations

TRY 2 : Looks Great Except Insert campaigns

- i ran the experiment from a machine in US
- i add connection pool for Write and Read
- i remove the first iteration from average because i need to create more connection then in EU in it was 4x time slower than other iteration

region	WF	insertCampaigns	selectCampaigns	selectCampaign	updateBids	updateAssets
EU		35.57	22.75	51.07	167.48	2043
US	E	562.8	18.45	29.07	250.95	2072.24
US	S	564.12	16.93	30.88	254.238	2076.1
US	G	566.67	20.3	30.68	237.78	2073.76

TRY 2 : Looks Great Except Insert campaigns

- i thought it’s ok and that is the best we can get
- then after talking with ofir from elad’s team he talled me i have a problem with the insert campaign statement because i am running 50 insert queries in parallel
  i am wasting a lot of time waiting for locks!! and in aurora it even worse

https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-mysql-write-forwarding.html

In sessions that use write forwarding, you can only use the `REPEATABLE READ` isolation level. Although you can also use the `READ COMMITTED` isolation level with Aurora Replicas, that isolation level doesn't work with write forwarding. For information about the `REPEATABLE READ` and `READ COMMITTED` isolation levels, see Aurora MySQL isolation levels.

TRY 2 : Looks Great

- i ran the experiment again this time i shuffle the queries and keep them at the same order for both benchmarking
- that way i avoid the lock time

```
Math.random = () => {
    const x = Math.sin(seed++) * 10000;
    return x - Math.floor(x);
  };
```

TRY 2 : Looks Great

region	WF	insertCampaigns	selectCampaigns	selectCampaign	updateBids	updateAssets
EU		35.57	22.75	51.07	167.48	2043
US	E	82.8	18.45	29.07	250.95	2072.24
US	S	83.12	16.93	30.88	254.238	2076.1
US	G	109.7	20.3	30.68	237.78	2073.76

Summary

1. Make the simplest code you can to simultate your case
2. Run code multiple times
3. Always calculate the average of X Runs and not single run
4. Repeat the entire experiment X times
5. Check runtime factors not impacting your experiment
6. Use the same logic you are going to use in production (connection pool)