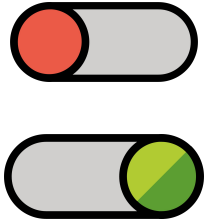
Finite State Machines Made Easy



Marco Ippolito

Marco Ippolito

Senior Security Engineer @HeroDevs

Node.js core maintainer


MY FIRST DAY
GOING TO NEW JOB
imgflip.com


MakeAGIF.com

JACKS OR BETTER

Scala reale (royal flush)	250	500	750	1000	1250
Scala colore (straight flush)	50	100	150	200	250
Poker (four of a kind)	25	50	75	100	125
Full (full house)	9	18	27	36	45
Colore (flush)	6	12	18	24	30
Scala (straight)	4	8	12	16	20
Tris (three of a kind)	3	6	9	12	15
Doppia coppia (two pairs)	2	4	6	8	10
Jacks or Better	1	2	3	4	5

Eventuali malfunzionamenti annullano tutte le giocate e i pagamenti

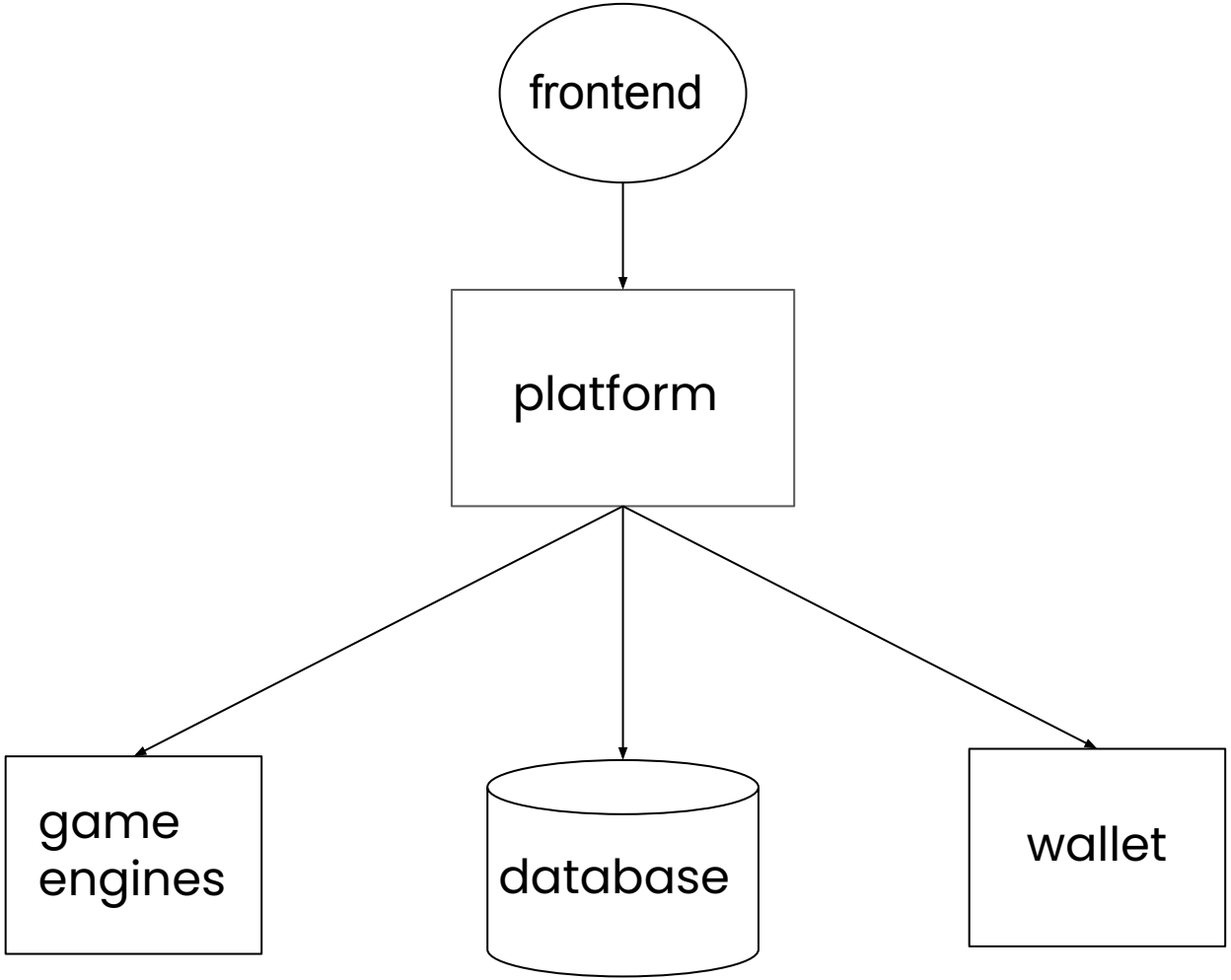
SELEZIONA LE CARTE CHE VUOI TENERE CLICCANDOCI SOPRA

BET
€ 0,20

BET BASE
€ 0,20

0%

CREDITI
€ 999,80


frontend
platform
game engines
database
wallet

The platform

- Compliant with Italian regulation
- Driven by business and not by IT
- Supports different game providers
- Distributed

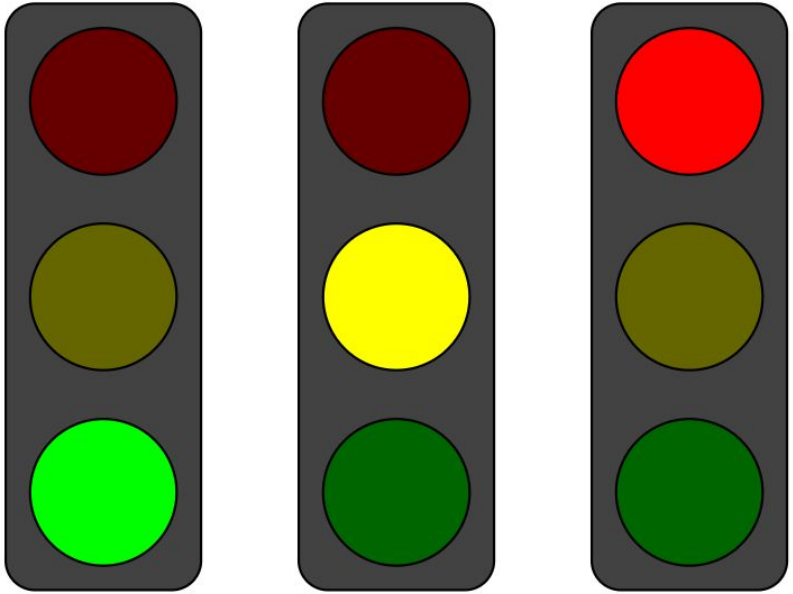
The issues

- Increasingly complex features made code unmaintainable
- Racing conditions from times
- Unclear boundaries between game engine and platform
- EXPEN$$$IVE BUGS

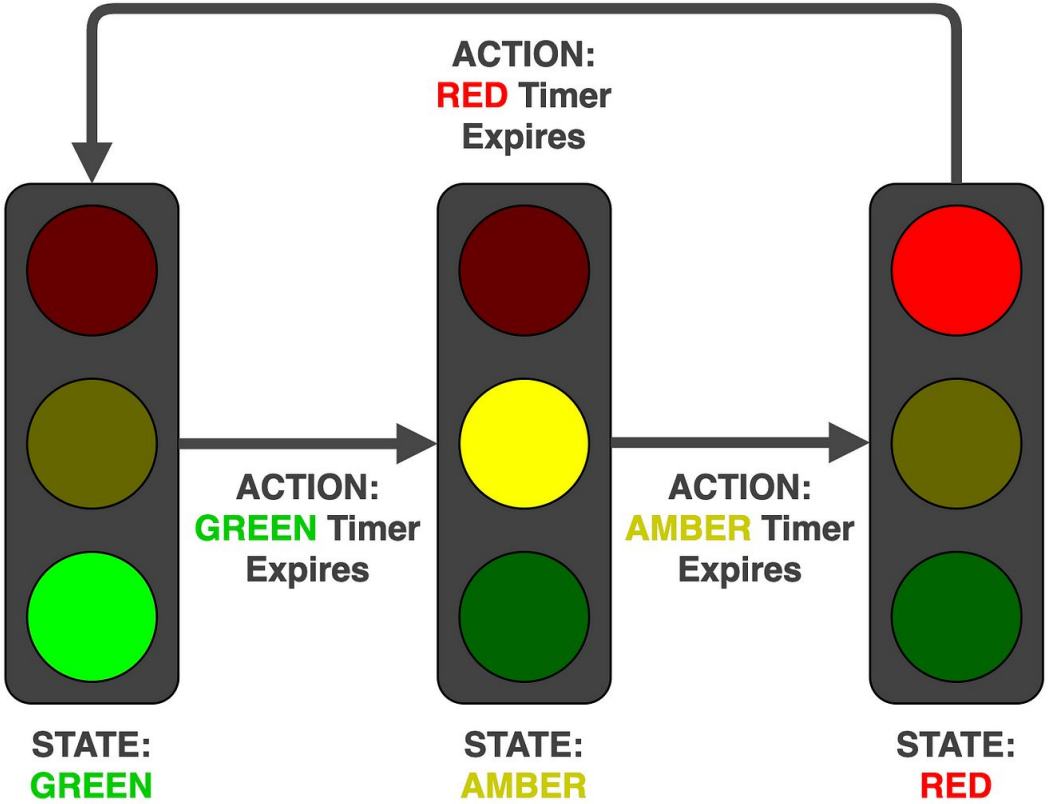

Gentlemen, gentlemen,
gentlemen!
MakeAGIF.com

State machine

A computational model that consists of **states**, **transitions**, and **events**.



State machine

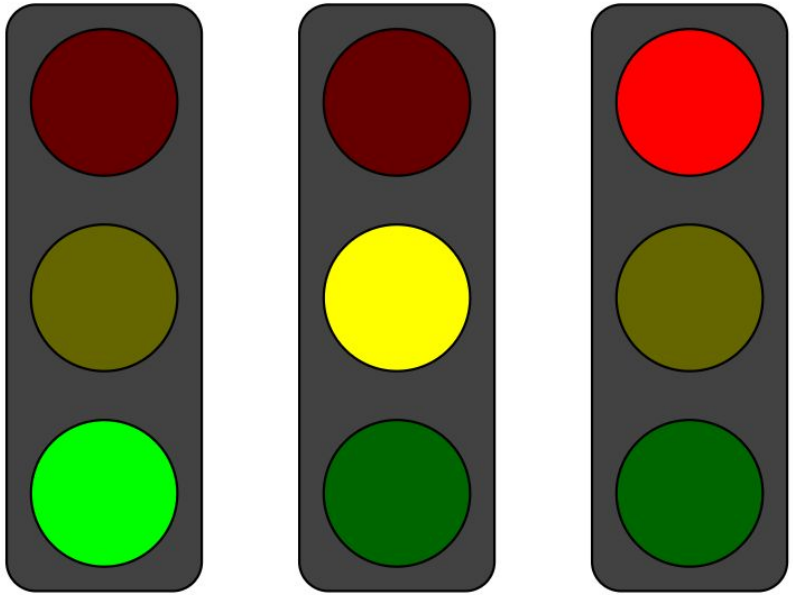


Different types

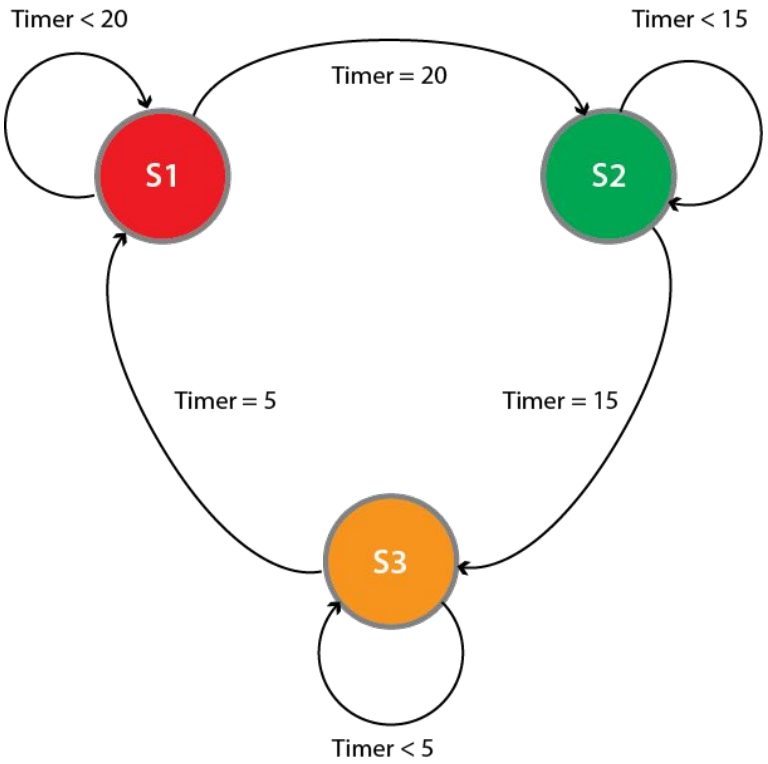
- Moore
- Mealy
- Deterministic
- Non deterministic

Moore Machine

The output depends **only** on the current state.



Moore Machine



Moore

Pros 📈

- Predictable and easy to debug
- Simple to implement

Cons 📉

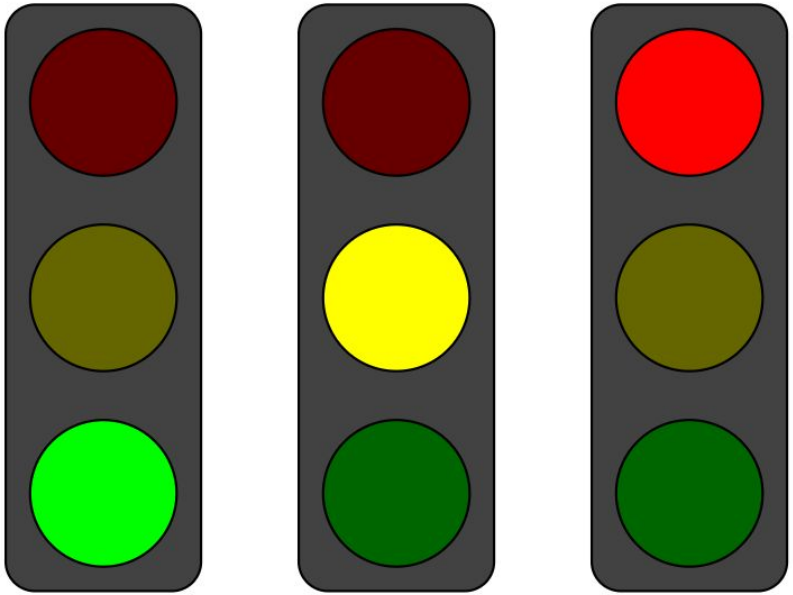
- States can proliferate quickly
- The flow diagram can become very large
- A lot of transitions

Different types

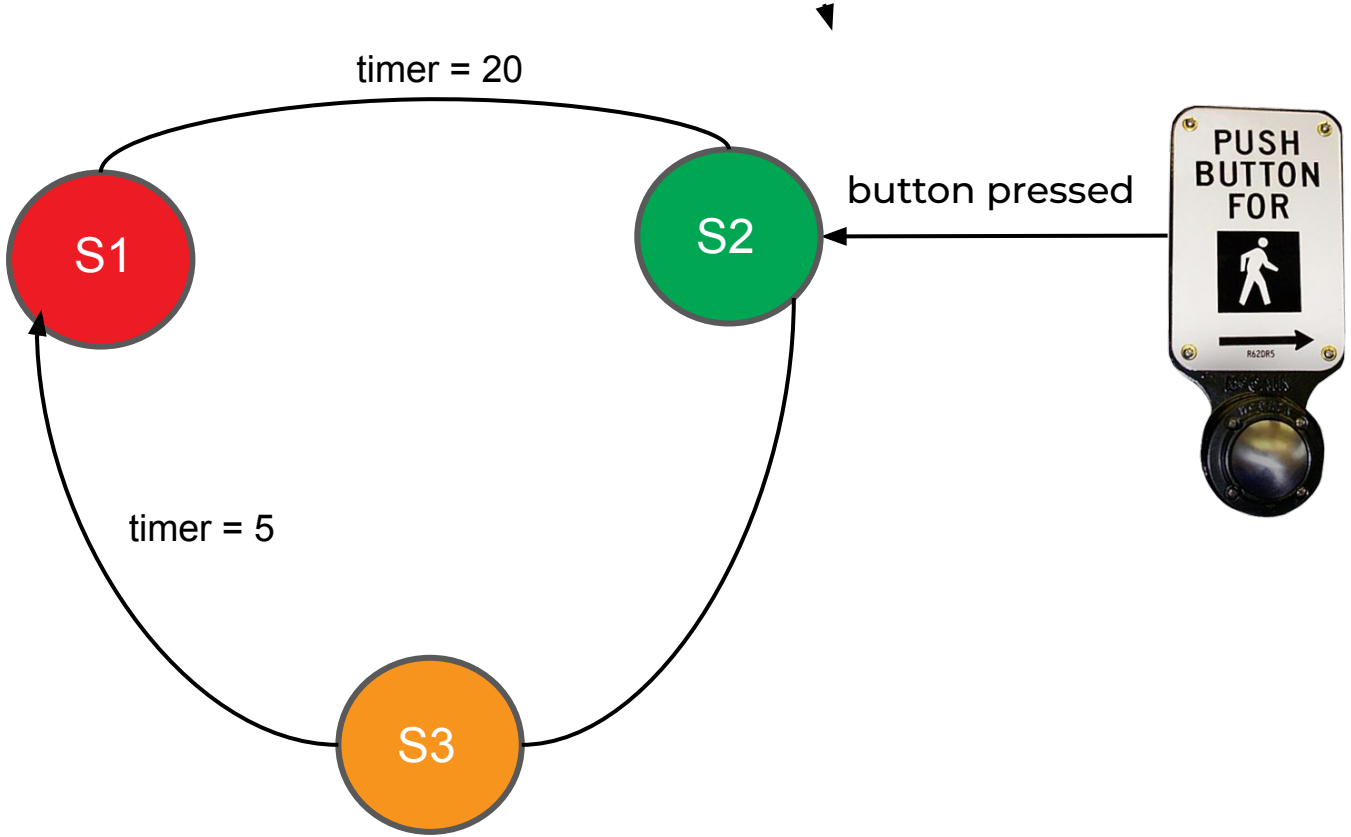
- Moore
- Mealy
- Deterministic
- Non deterministic

Mealy Machine

The output depends on **both** the **current state** and the **input**.



Mealy Machine



Mealy

Pros 📈

- Compact
- Flexible, a single state can produce more outputs

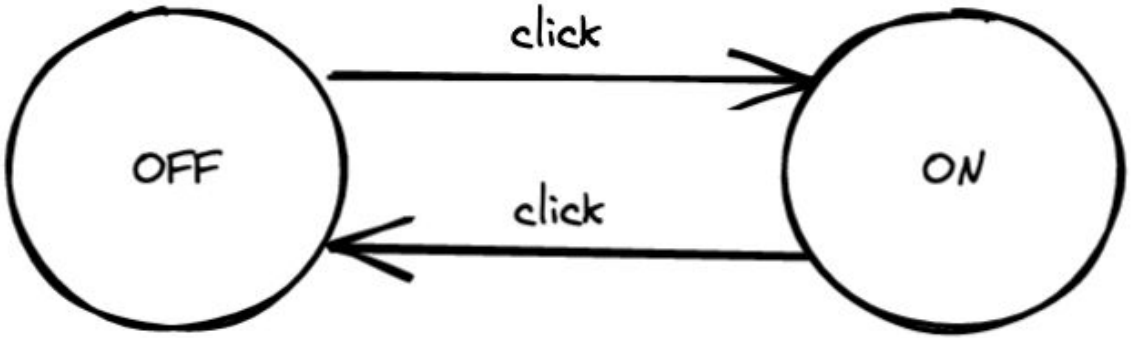
Cons 📉

- Complex
- Hard to debug since the output relies on the current state + input

Different types

- Moore
- Mealy
- Deterministic
- Non deterministic

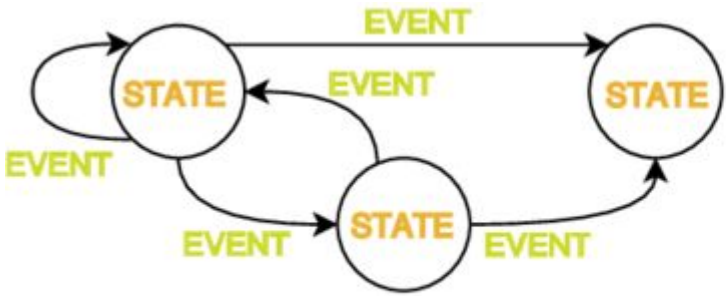
Deterministic



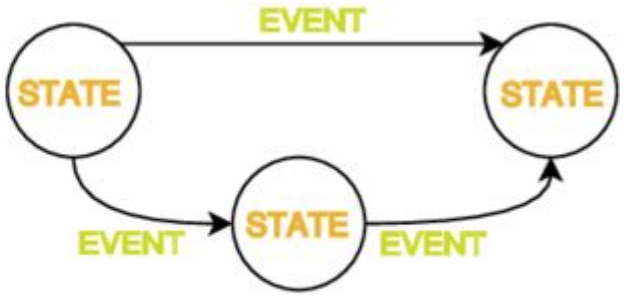
Different types

- Moore
- Mealy
- Deterministic
- Non deterministic

Non Deterministic



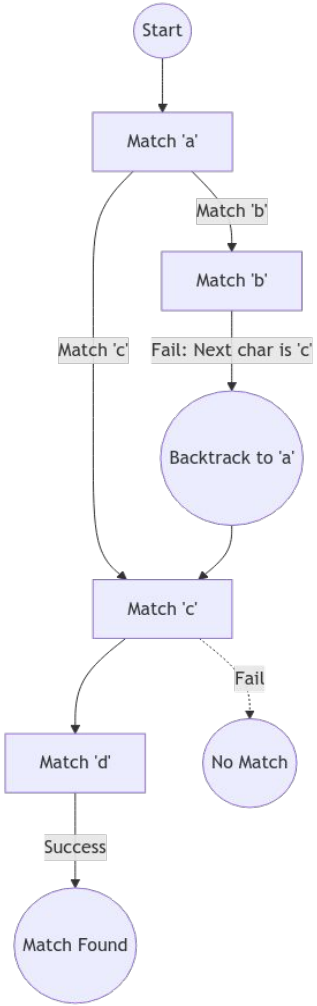
non-deterministic



deterministic

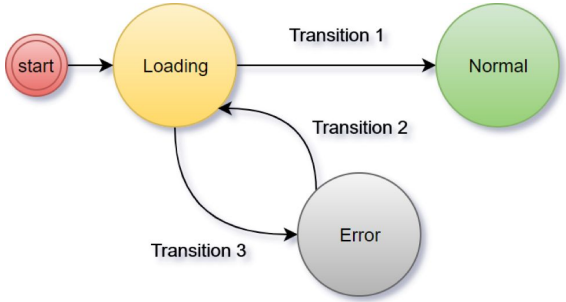
Regex

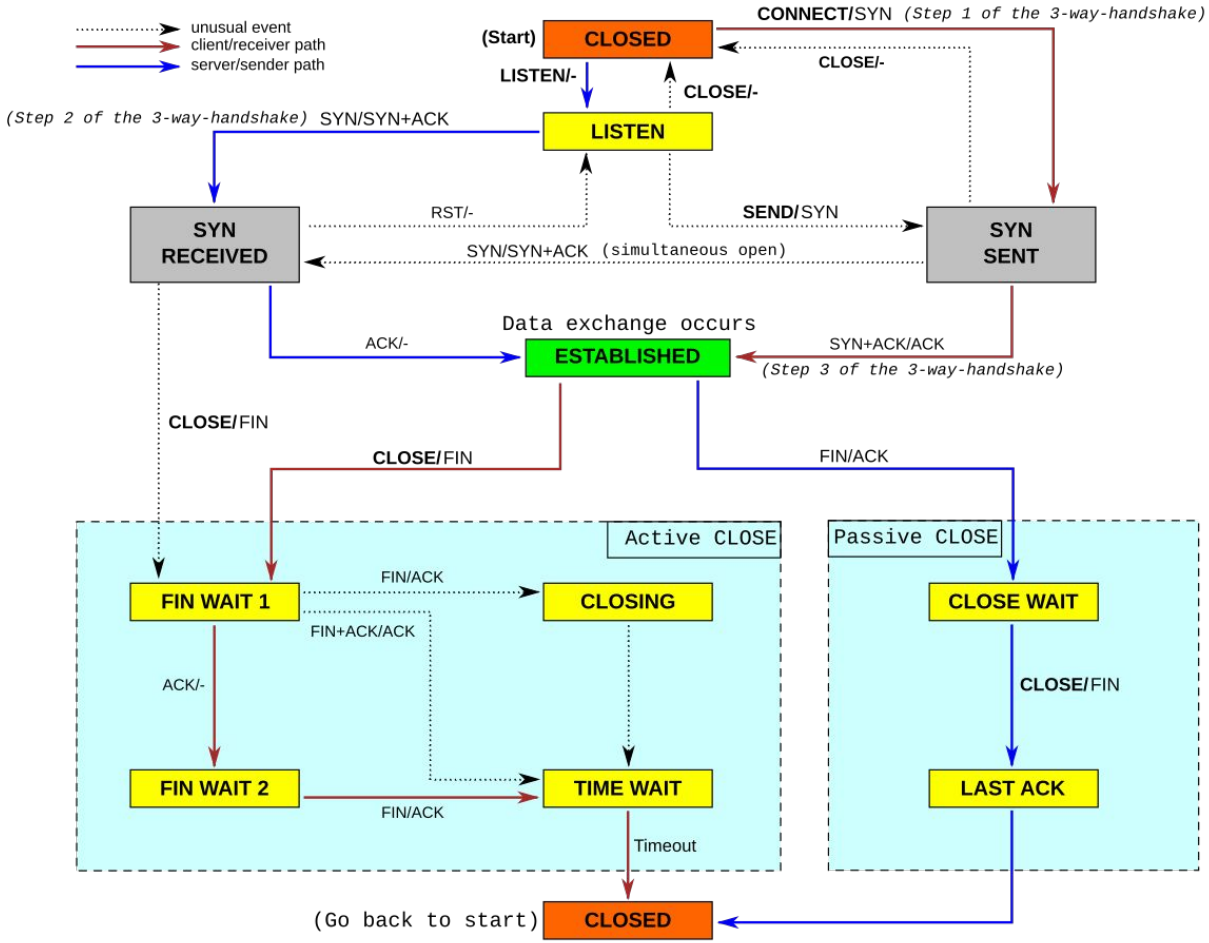
Pattern matching is driven by the language elements in the regular expression and not by the characters to be matched in the input string. Therefore, the regular expression engine tries to fully match optional or alternative sub-expressions.



Lore

Finite State Machines are widely used in low-level software like drivers and networking where behavior is well defined.




unusual event
client/receiver path
server/sender path
(Start)
CLOSED
CONNECT/SYN (Step 1 of the 3-way-handshake)
CLOSE/-
LISTEN/-
CLOSE/-
(Step 2 of the 3-way-handshake) SYN/SYN+ACK
LISTEN
SYN RECEIVED
RST/-
SEND/SYN
SYN/SYN+ACK (simultaneous open)
SYN SENT
Data exchange occurs
ACK/-
ESTABLISHED
SYN+ACK/ACK
(Step 3 of the 3-way-handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
Active CLOSE
Passive CLOSE
FIN WAIT 1
FIN/ACK
CLOSING
FIN+ACK/ACK
CLOSE WAIT
ACK/-
CLOSE/FIN
FIN WAIT 2
TIME WAIT
FIN/ACK
LAST ACK
Timeout
(Go back to start)
CLOSED

State machines

Pros 📈

- Very easy to test
- Can be understood by business people
- Easy logging and monitoring
- Simplifies complexity

State machines

Cons📉

- Synchronous by nature
- Stateful
- Mealy vs Moore?
- Lack of libraries in Node.js

State machines

Cons📉

- Synchronous by nature
- Stateful
- Mealy vs Moore?
- Lack of libraries in Node.js

Stateful

- Very very hard to make it work in distributed systems
- Always need to snapshot the current state of the machine
- Requires in memory database
- Restoring state is expensive

State machines

Cons📉

- Synchronous by nature
- Stateful
- Mealy vs Moore?
- Lack of libraries in Node.js


¿Porque no los dos?

Mealy vs Moore

- Mix, you can use both at the same time
- Re-use state a to accept multiple inputs
- When a state has too many input, break into different states
- Balance

State machines

Cons📉

- Synchronous by nature
- Stateful
- Mealy vs Moore?
- Lack of libraries in Node.js

Lack of libraries in Node.js

- Most libraries are frontend oriented
- xstate is great but complex
- Most libraries are biased towards a fsm type (deterministic vs non deterministic)
- Lightweight

Fiume

npm v0.3.5 | build passing | code style biome | Bundle Size 1.2 KiB

Fiume is a zero-dependency, simple, and flexible state machine library written in TypeScript. It supports *Deterministic* and partially *Non-Deterministic* state machines. It is compatible with all JavaScript runtimes and is designed to manage the flow of a system through various states. This library provides a lightweight and intuitive way to define states, transitions, and hooks for state entry, exit, and transition events.

Unlike other libraries, **Fiume** does not require hardcoding state transitions. Instead, you can write the transition logic inside the `transitionTo` function.

Docs

You can find documentation and examples at **fiume.dev**.

Installation

```
npm install fiume
```

Install

```
> npm i fiume
```

Repository

github.com/marco-ippolito/fiume

Homepage

fiume.marcoippolito.dev

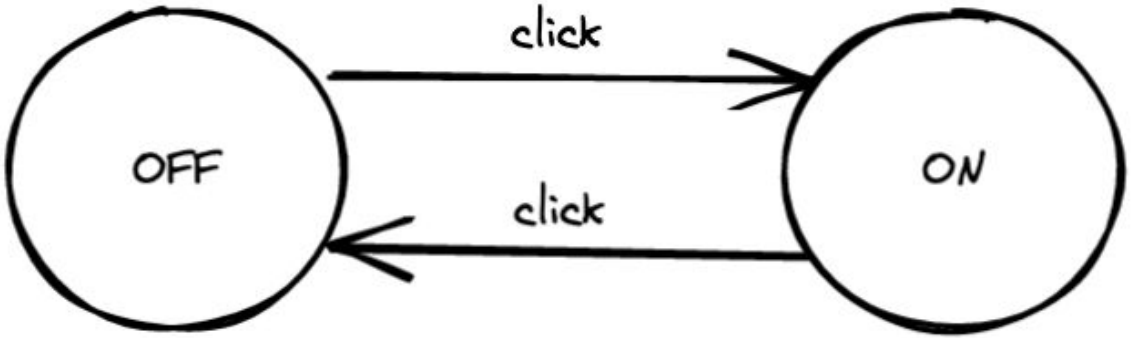
2024-07-23 to 2024-07-29

201

Version	License
0.3.5	Apache-2.0

Unpacked Size	Total Files
36 kB	11

Issues	Pull Requests
0	0


click
OFF
ON
click

```
import { StateMachine, State } from "fiume";

// Define a simple ON-OFF machine
const states: Array<State> = [
  {
    id: "OFF",
    initial: true,
    transitionGuard: ({ event }) => event === 'button clicked',
    transitionTo: () => "ON",
  },
  {
    id: "ON",
    transitionGuard: ({ event }) => event === 'button clicked',
    transitionTo: () => "OFF",
  },
];

// Create a state machine instance
const machine = StateMachine.from(states);
```

```
// Start the state machine
await machine.start();
console.log(machine.currentStateId); // OFF

// Trigger a transition by sending an event
await machine.send('button clicked');
console.log(machine.currentStateId); // ON

// Trigger another transition
await machine.send('button clicked');
console.log(machine.currentStateId); // OFF

// Trigger another transition
await machine.send('wrong event'); // wrong event wont trigger the transition
console.log(machine.currentStateId); // OFF
```


16:31
BOOK OF RA
4
2
9
6
1
7
8
3
5
Menu
Hold for AutoSpin
Bet 9
987
Credit
5
Last Win

```
{
    id: "START",
    initial: true,
    transitionGuard: ({ event }) => event?.type === EVENTS.START,
    transitionTo: async ({ context }) => {
        context.initialState = await getSlotInitialState();
        context.balance = await getBalance();
        return "BET";
    },
}
```

```
await machine.send({ type: EVENTS.START });
```

```
{
    id: "BET",
    transitionGuard: ({ event }) => event?.type === EVENTS.BET,
    transitionTo: async ({ event, context }) => {
        const e = event as SlotMachineBetEvent;
        if (e.value > context.balance) {
            throw new Error("INVALID BET AMOUNT");
        }
        context.bet = e.value;
        context.balance = await removeFromBalance(e.value);
        return "STEP";
    },
}
```


BOOK OF RA deluxe
18:19
4
2
9
6
1
7
8
3
5
10
10
4
2
8
6
1
7
9
3
5
Credit
500.00
Please place your bet
Last Win
0.00
Autoplay
Paytable
Lines
10
Bet/Line
0.04
Bet
0.40
Gamble
Start

```
await machine.send({ type: EVENTS.BET, value: 10 })
```

```
{
    id: "STEP",
    transitionGuard: ({ event }) => event?.type === EVENTS.STEP,
    transitionTo: async ({ context }) => {
        const result = await getResult();
        if (result) {
            console.log("WIN!");
            await addWin(context.bet);
        } else {
            console.log("LOST!");
        }
        console.log("balance:", await getBalance());
        return "START";
    },
}
```

```
await machine.send({ type: EVENTS.STEP });
```


1500
500
1000
2000
MINI
MINI
500
MINI
MAJOR
2000
500
BONUS REELS IN PLAY
WIN
CREDIT
40796
BET
500
10¢
PRESS START TO BEGIN

Thanks for listening!