Purely Functional gRPC & HTTP/2 with OCaml

$ whoami

Hi 🖐, my name is Adam.

Software engineer at Dialo.ai

adamchol.com (very info-rich webpage, btw)

@adamchol_

@adamchol

me@adamchol.com

ENGINEERED FOR THE CHAOS OF REAL CALLS

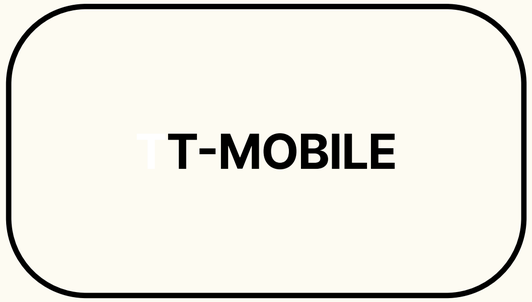
Voice AI Agents as a Service

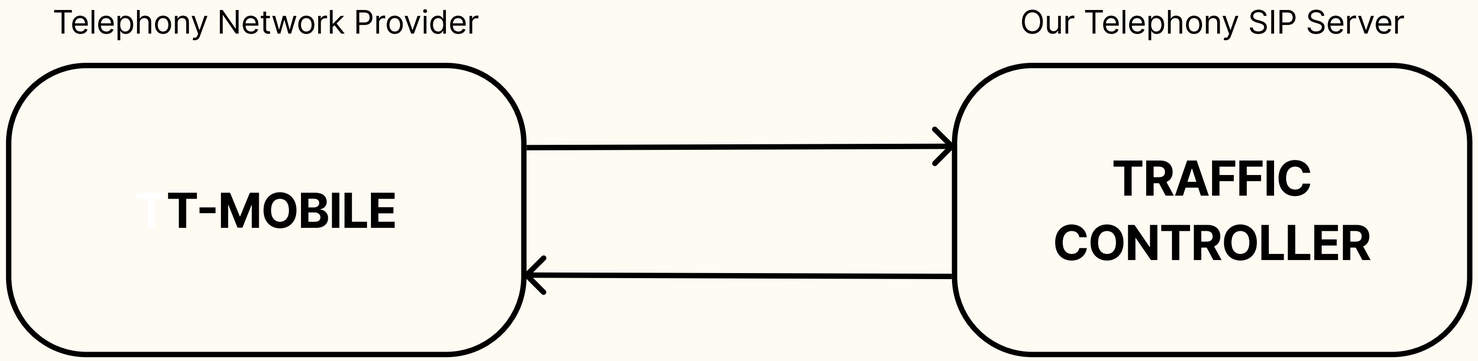
We build and run voice AI agents that handle millions of real calls [noisy lines, heavy accents, unpredictable customers] and plug into complex enterprise systems.

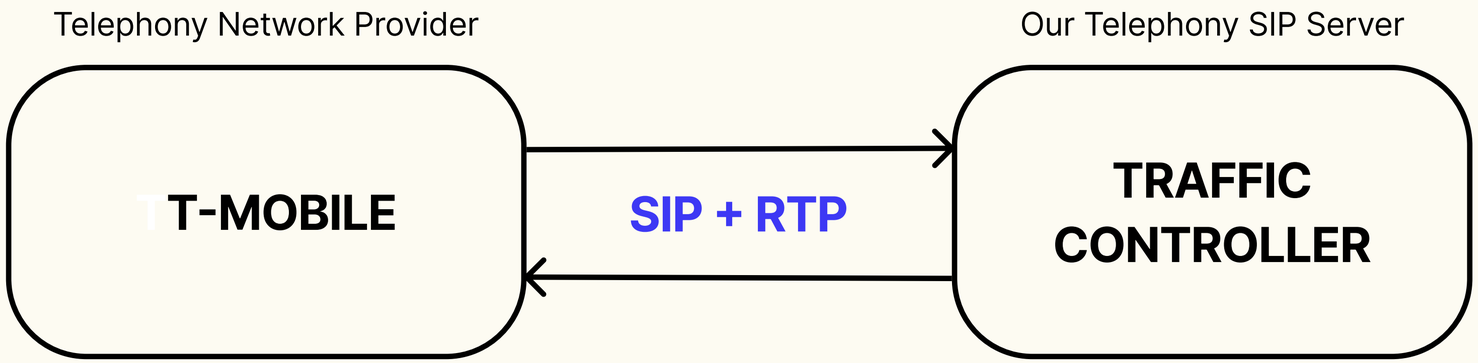
INFRASTRUCTURE UNDER YOUR CONTROL

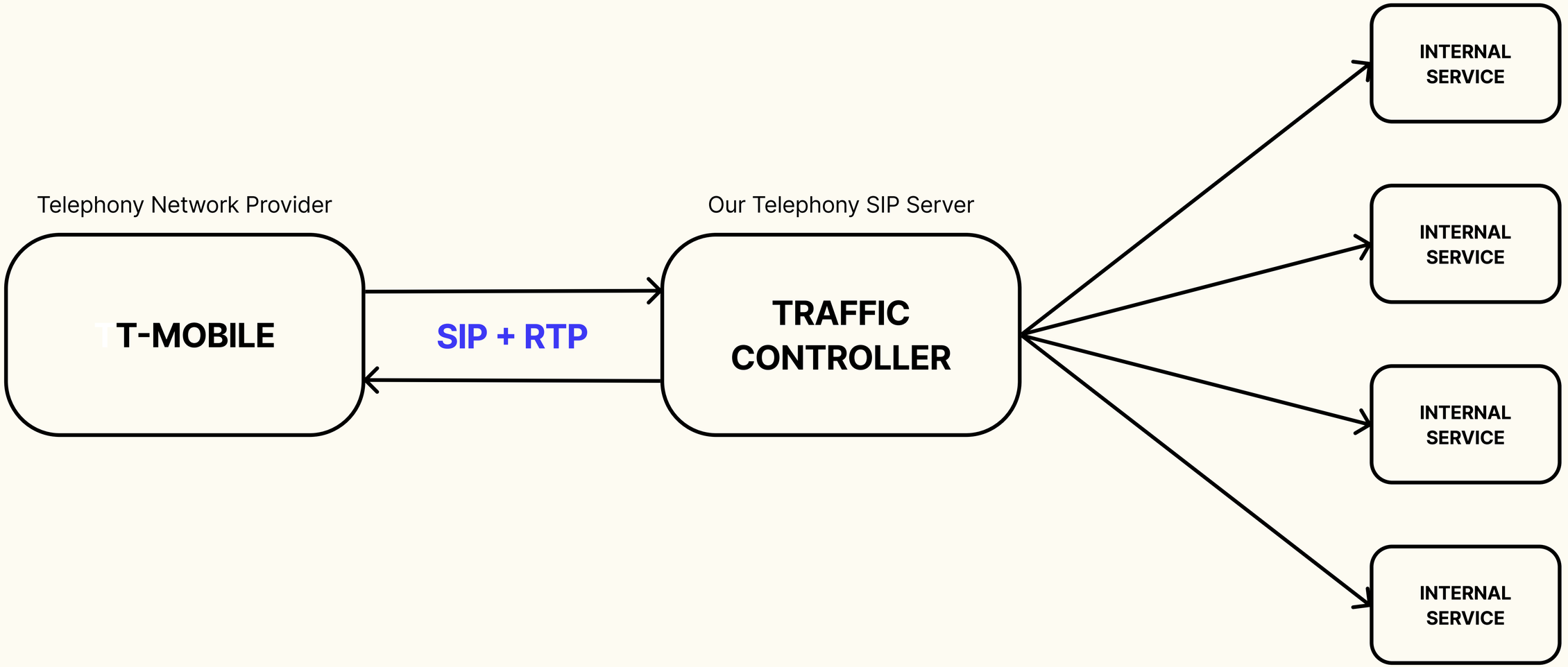
Agents fine-tuned for your workflows, tailored to customers

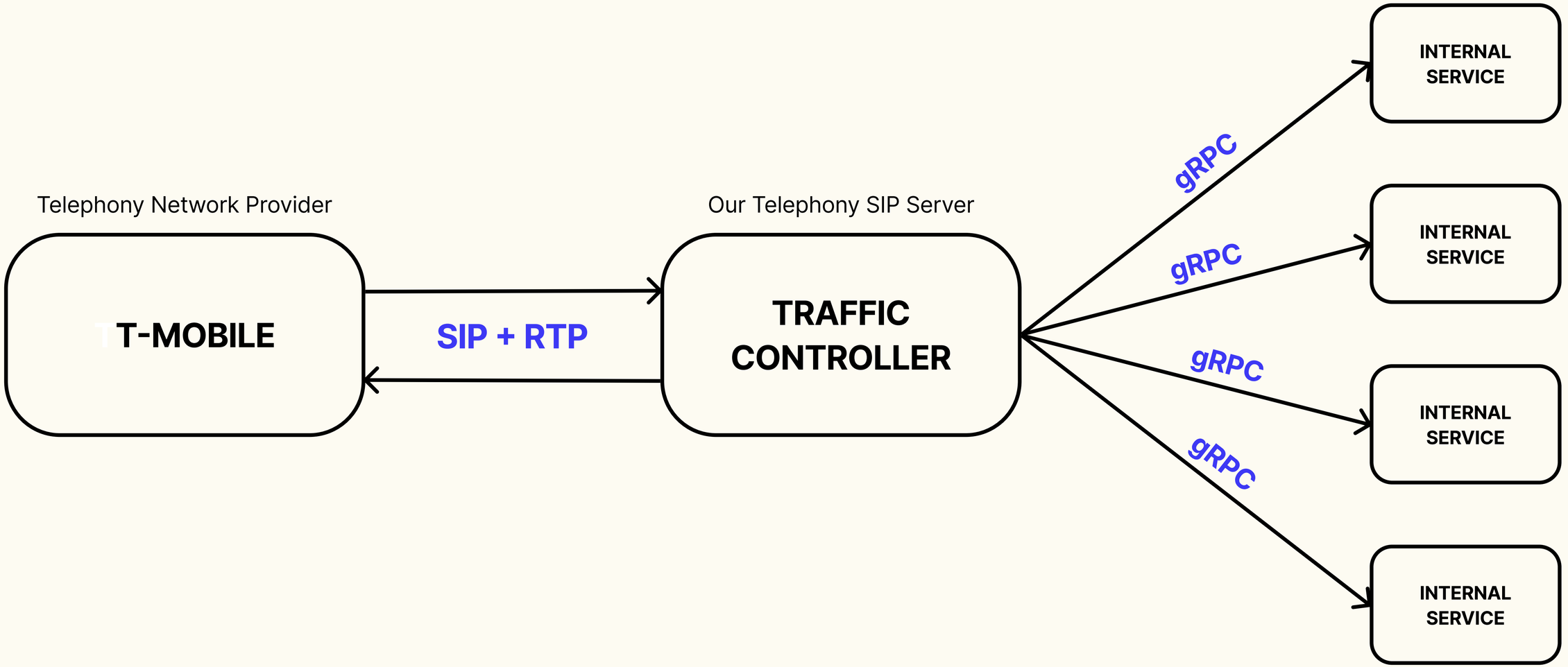
Telephony Network Provider

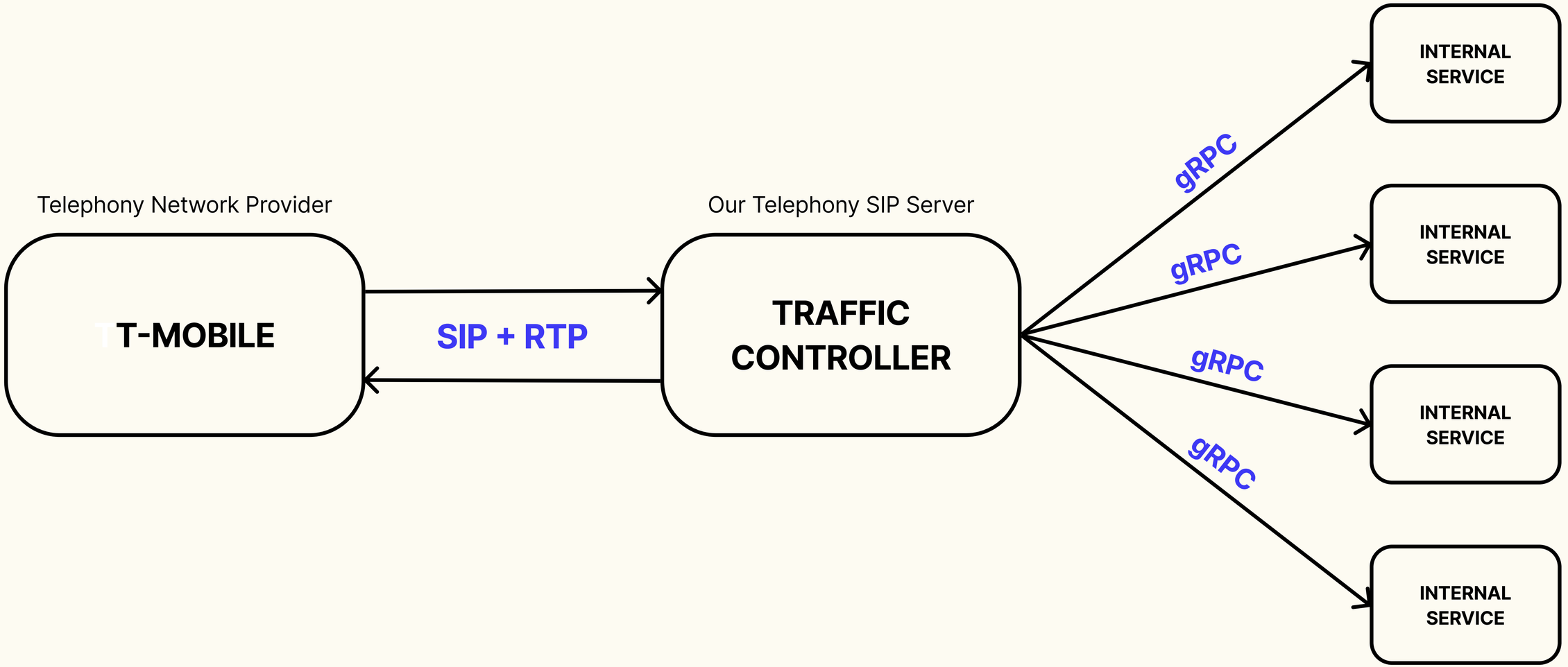



Telephony Network Provider
T-MOBILE
Our Telephony SIP Server
TRAFFIC CONTROLLER

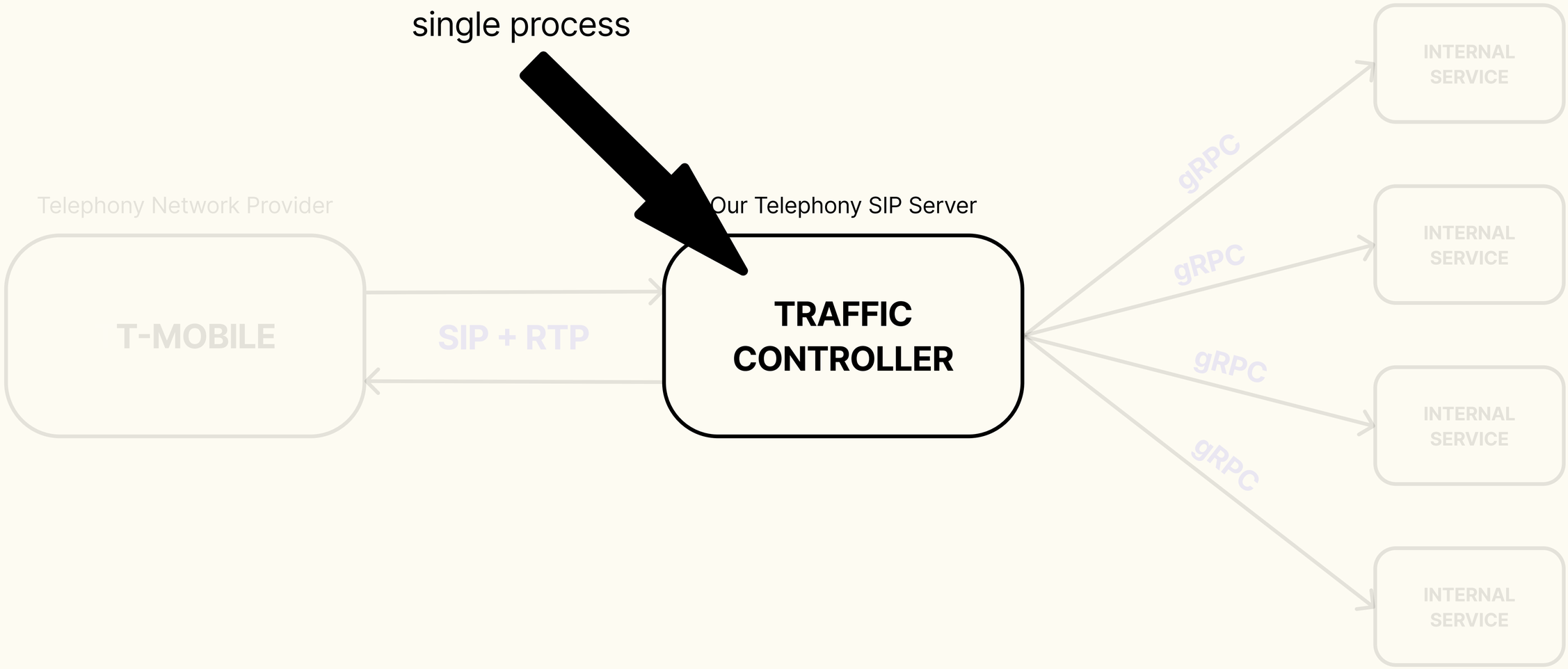

Telephony Network Provider
Our Telephony SIP Server
T-MOBILE
SIP + RTP
TRAFFIC CONTROLLER


Telephony Network Provider
T-MOBILE
SIP + RTP
Our Telephony SIP Server
TRAFFIC CONTROLLER
INTERNAL SERVICE
INTERNAL SERVICE
INTERNAL SERVICE
INTERNAL SERVICE


Telephony Network Provider
T-MOBILE
SIP + RTP
Our Telephony SIP Server
TRAFFIC CONTROLLER
gRPC
gRPC
gRPC
gRPC
INTERNAL SERVICE
INTERNAL SERVICE
INTERNAL SERVICE
INTERNAL SERVICE

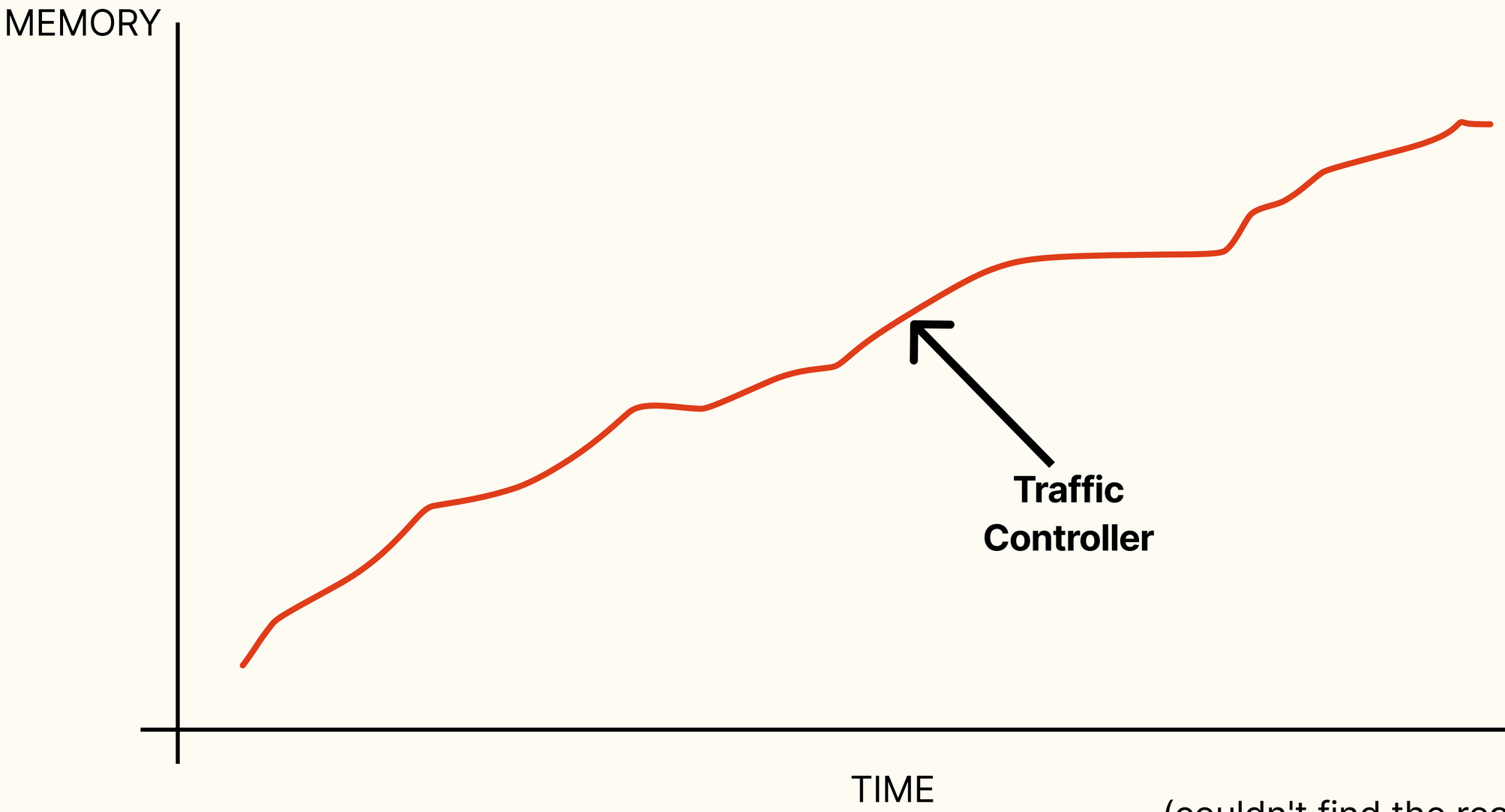


Even 100,000s calls a day = 100s of concurrent calls



Even 100,000s calls a day = 100s of concurrent calls

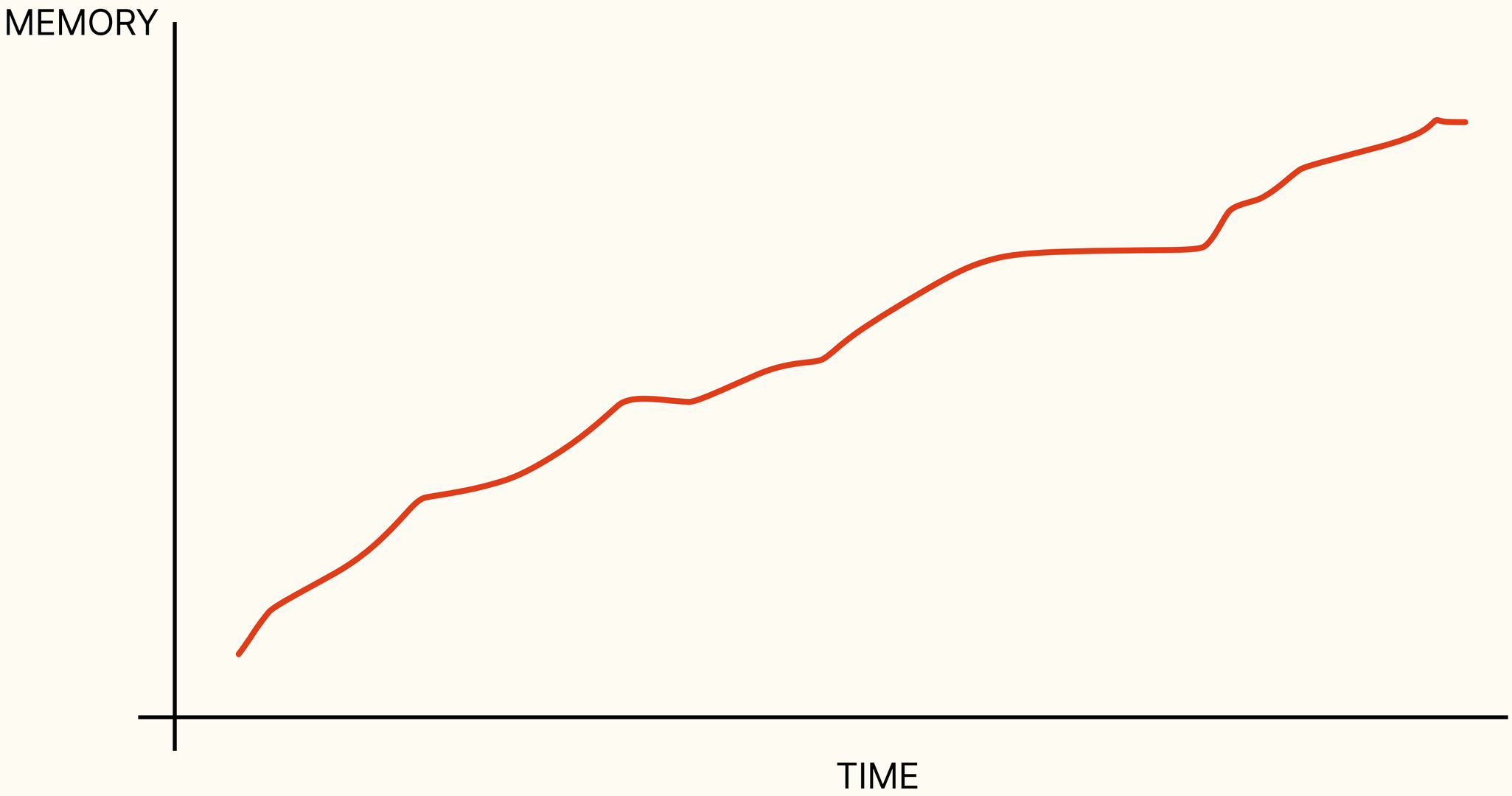
The Problem - memory leak



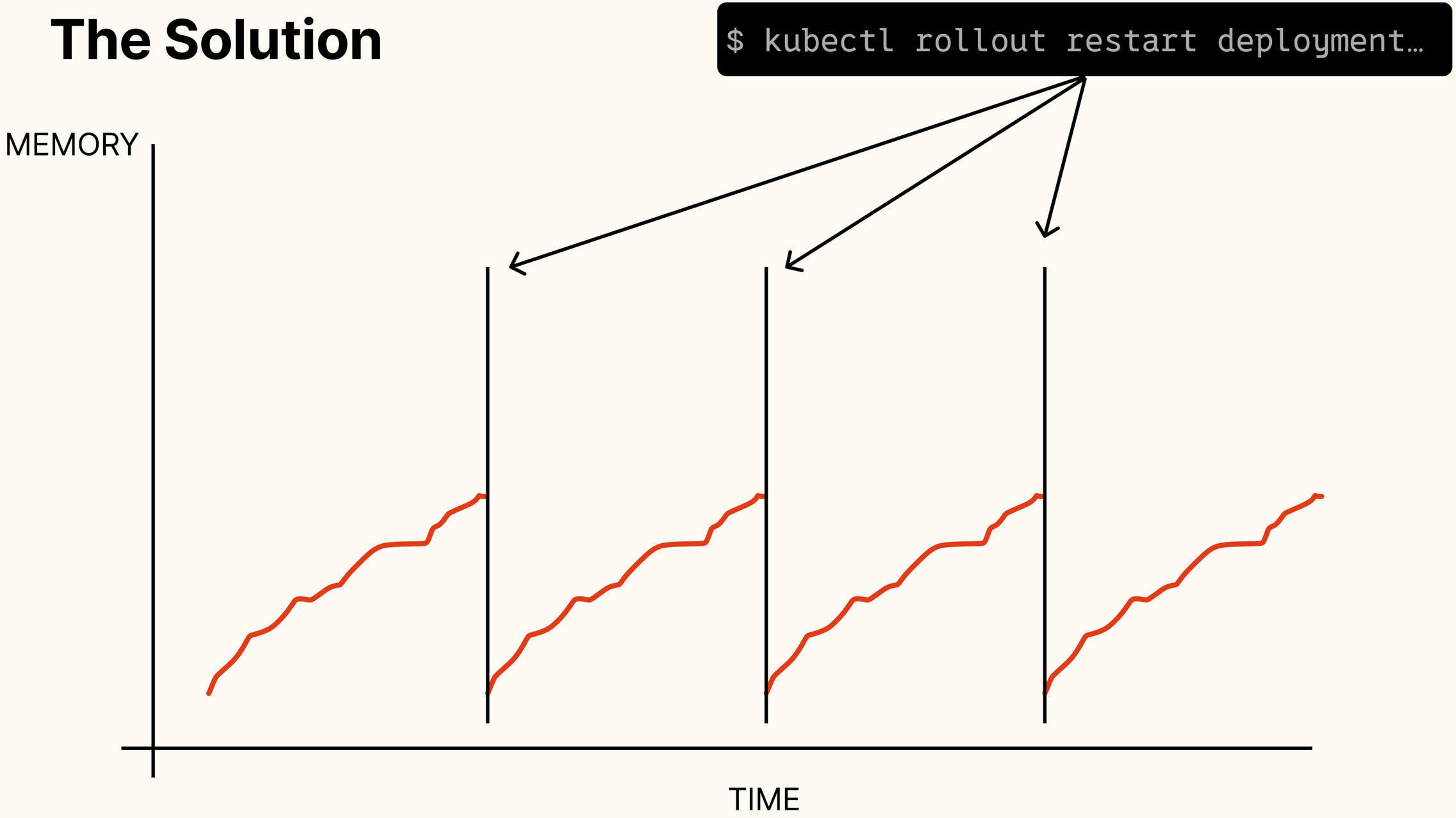
(couldn't find the real metrics graph lol)

The Solution

```
$ kubectl rollout restart deployment…
```



The Solution



Rohan Vohra
@Ron85059

Engineers after fixing the bug by just restarting the system

Last edited 10:16 AM · Sep 4, 2025 · **44** Views

supermassive backlog @FakeRyanGosling · Jul 6, 2021

Who wants to share a dirty secret of their industry?

Here is mine: daily reboots are cheaper than fixing memory leaks

114 344 3K

Dennis Clark
@boomerangfish

2000: Reboot servers nightly before memory leaks cause a crash

2010: Nightly rebooting is bad! We must find the memory leaks instead!

2020: Auto-replace containers and VMs whenever they exceed a memory threshold

1:27 PM · Jul 7, 2021

Pfingerz
@passionfingerz

make memory leaks easier to fix? automate cleanup? nah son we going to restart the service on every call

10:51 AM · Sep 4, 2025 · **60** Views

Dmitrii Kovanikov
@ChShersh

Subscribe

I've been programming for 16 years.

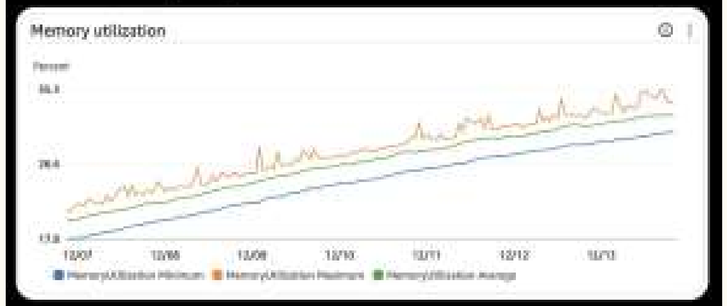
It still annoys me that restarting a service weekly is an industry-wide accepted solution to fixing memory leaks.

3:08 PM · Aug 9, 2025 · **907.9K** Views

Vic
@VicVijayakumar

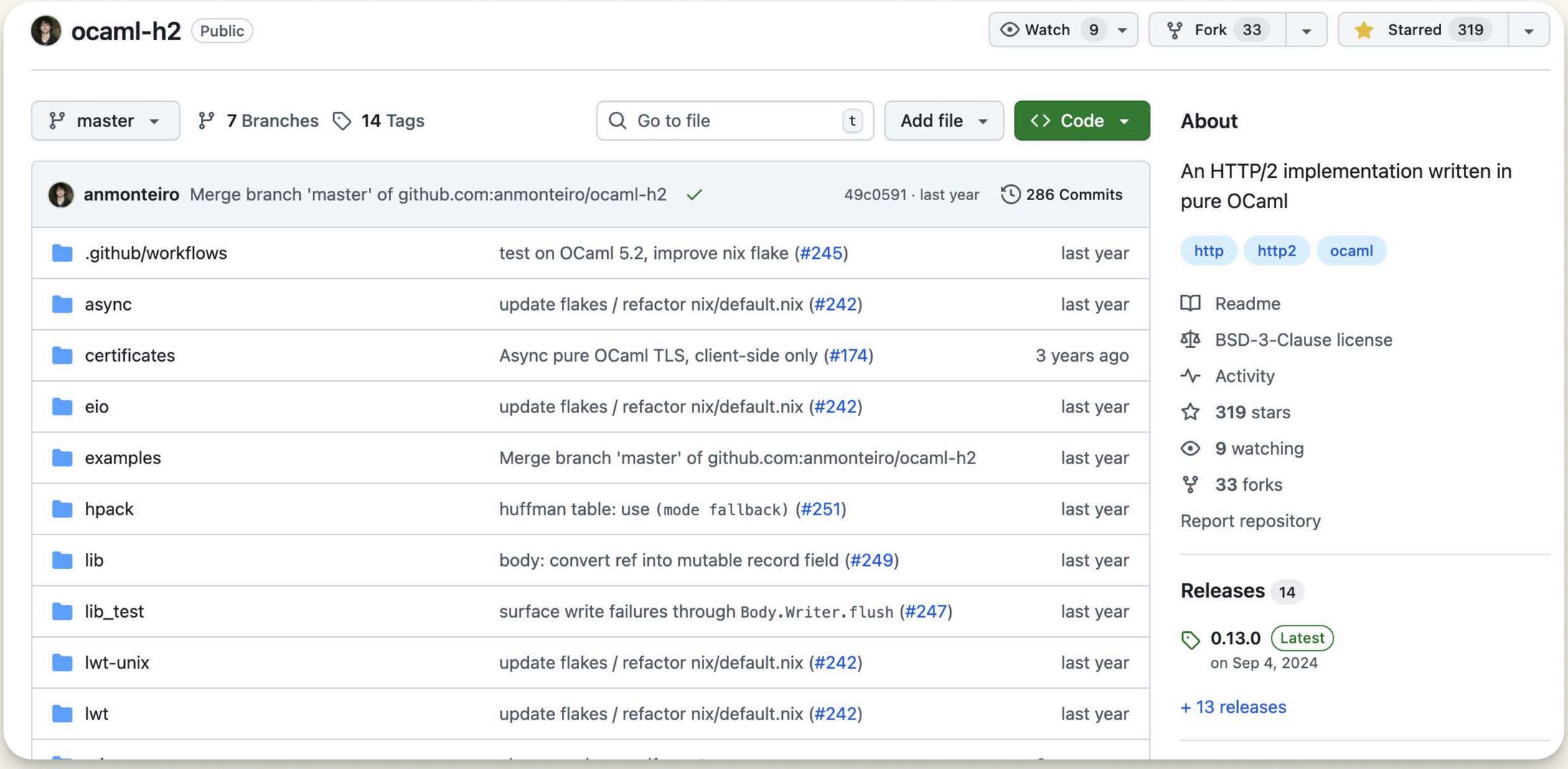
Subscribe

they won't tell you this in SRE school but the easiest way to fix a memory leak is to restart your pods on a schedule



4:04 AM · Dec 14, 2024 · **340.7K** Views

HTTP/2 Implementation



ocaml-h2 Public

Watch 9 | Fork 33 | Starred 319

master | 7 Branches | 14 Tags | Go to file t | Add file | Code

anmonteiro Merge branch 'master' of github.com:anmonteiro/ocaml-h2 ✓ 49c0591 · last year | 286 Commits

.github/workflows	test on OCaml 5.2, improve nix flake (#245)	last year
async	update flakes / refactor nix/default.nix (#242)	last year
certificates	Async pure OCaml TLS, client-side only (#174)	3 years ago
eio	update flakes / refactor nix/default.nix (#242)	last year
examples	Merge branch 'master' of github.com:anmonteiro/ocaml-h2	last year
hpack	huffman table: use `(mode fallback)` (#251)	last year
lib	body: convert ref into mutable record field (#249)	last year
lib_test	surface write failures through `Body.Writer.flush` (#247)	last year
lwt-unix	update flakes / refactor nix/default.nix (#242)	last year
lwt	update flakes / refactor nix/default.nix (#242)	last year

About

An HTTP/2 implementation written in pure OCaml

http | http2 | ocaml

Readme
BSD-3-Clause license
Activity
319 stars
9 watching
33 forks
Report repository

Releases 14

0.13.0 Latest
on Sep 4, 2024

+ 13 releases

Why not just patch the previous one?

Why not just patch the previous one?

- h2 is based on an older RFC

Why not just patch the previous one?

- h2 is based on an older RFC
- bug in the logic that should be removed anyway

Why not just patch the previous one?

- h2 is based on an older RFC
- bug in the logic that should be removed anyway
- overly complex code (mostly skill issue on my side)

Why not just patch the previous one?

- h2 is based on an older RFC
- bug in the logic that should be removed anyway
- overly complex code (mostly skill issue on my side)
- opportunity for fully effects-based implementation

Why not just patch the previous one?

- h2 is based on an older RFC
- bug in the logic that should be removed anyway
- overly complex code (mostly skill issue on my side)
- opportunity for fully effects-based implementation
- API for a neat integration into new gRPC

Implementing HTTP/2

Goals for Implementation:

Goals for Implementation:

- latest spec - RFC 9113

Goals for Implementation:

- latest spec - RFC 9113
- effects-based with Eio

Goals for Implementation:

- latest spec - RFC 9113
- effects-based with Eio
- Cstruct’s over bytes

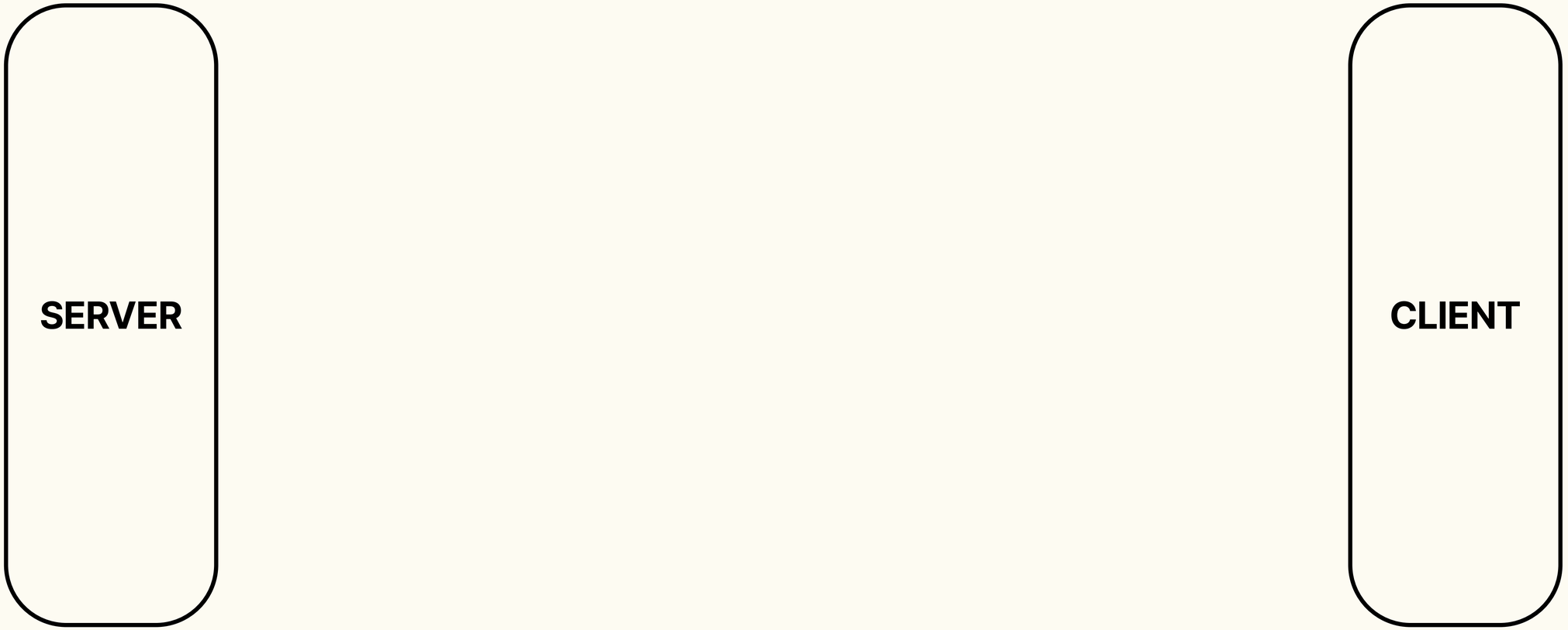
Goals for Implementation:

- latest spec - RFC 9113
- effects-based with Eio
- Cstruct’s over bytes
- modular parsers and serializers

Goals for Implementation:

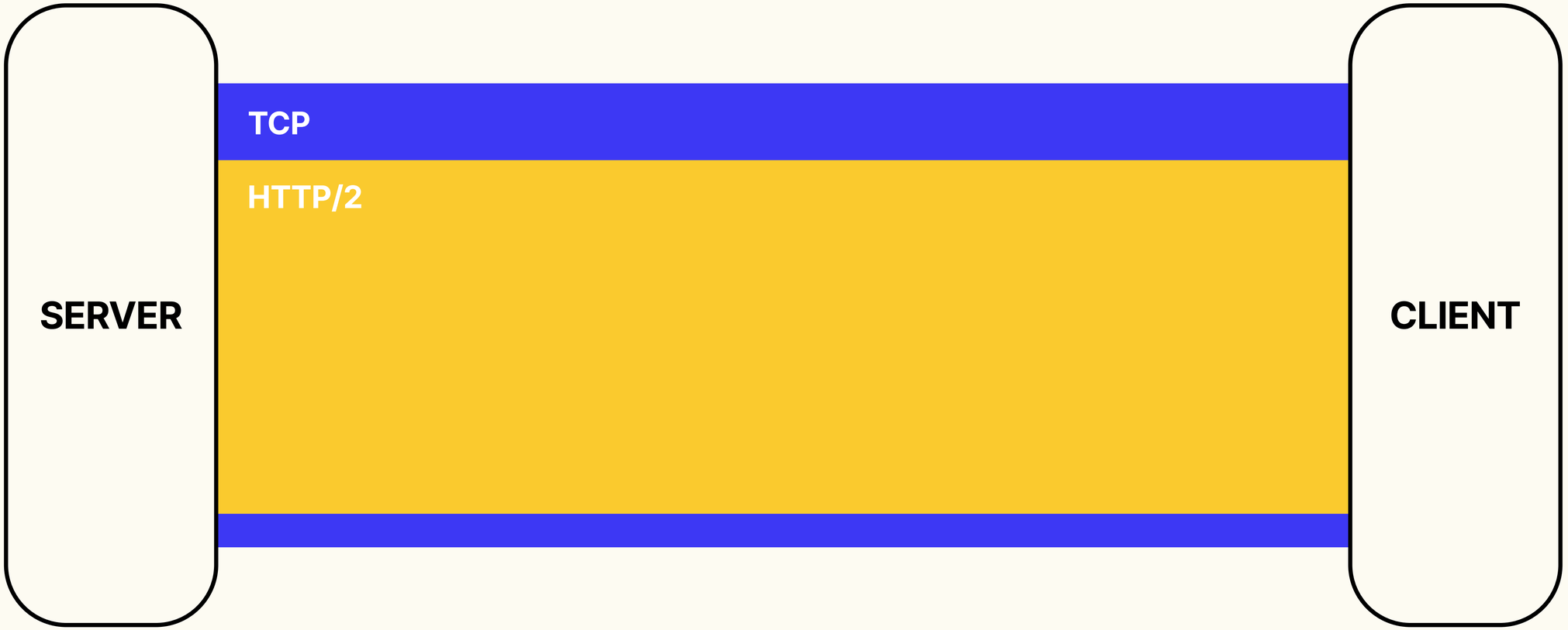
- latest spec - RFC 9113
- effects-based with Eio
- Cstruct’s over bytes
- modular parsers and serializers
- immutable state
- functional API

HTTP/2 Crash Course!



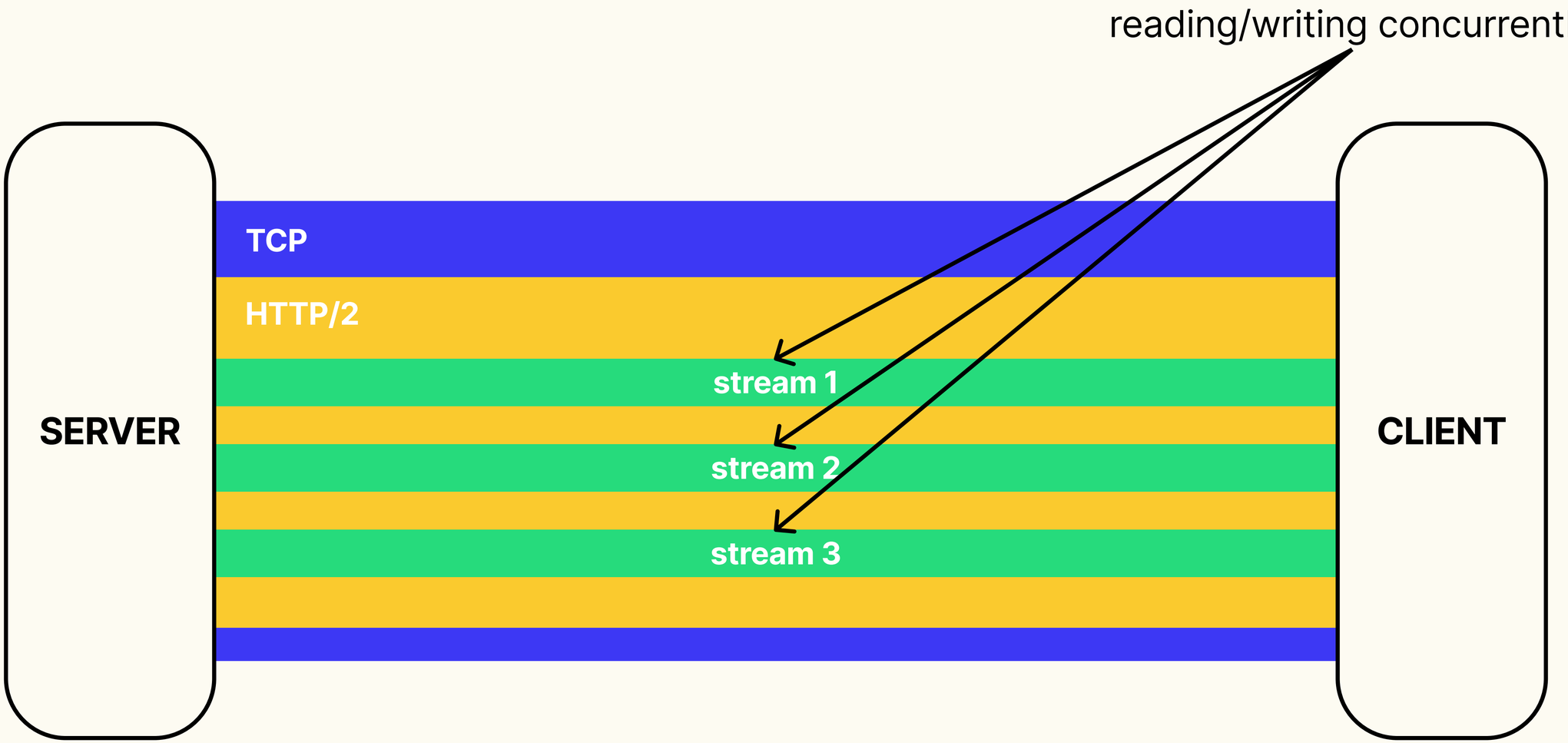
HTTP/2 Crash Course!

HTTP/2 Crash Course!

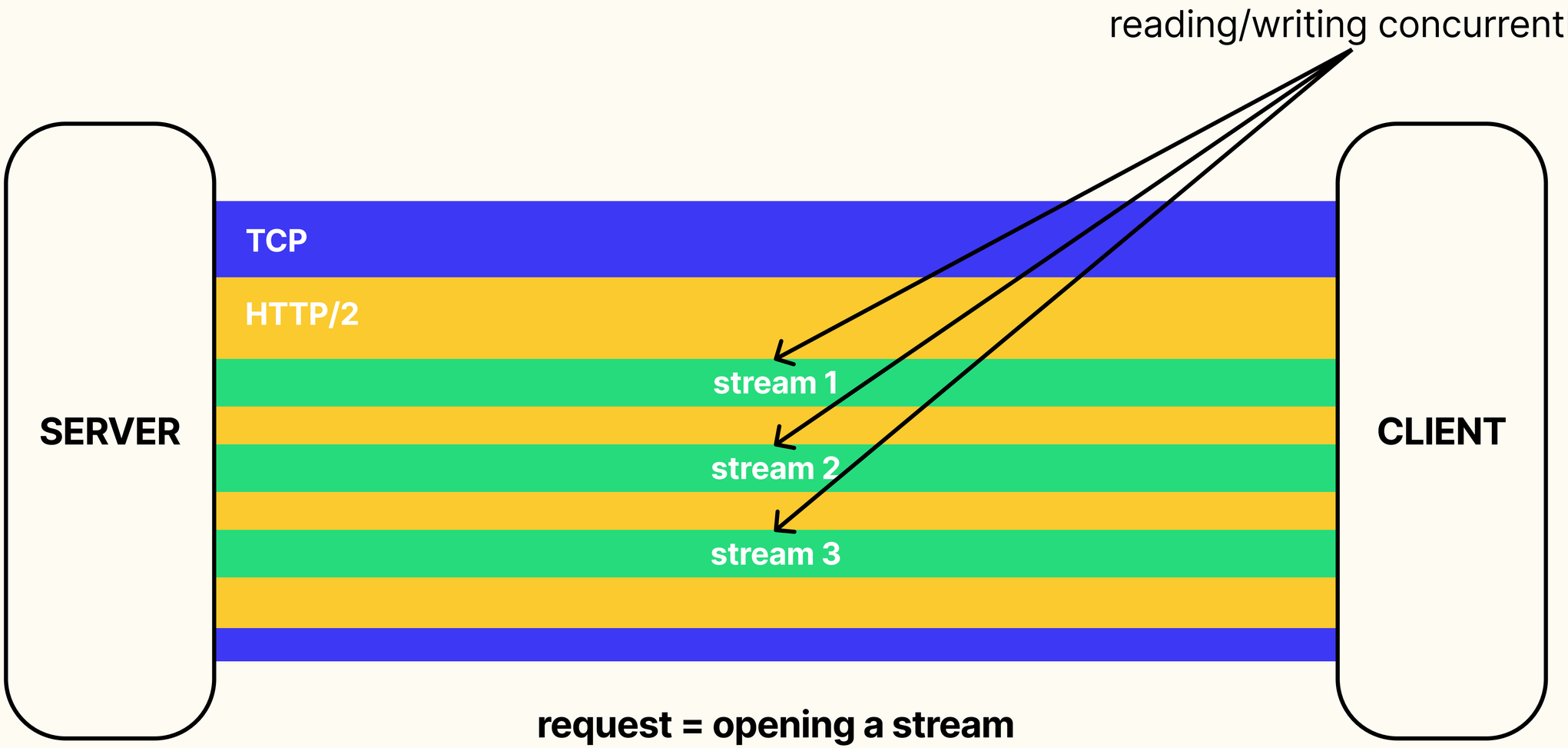


HTTP/2 Crash Course!

HTTP/2 Crash Course!



HTTP/2 Crash Course!



request = opening a stream

request body = data streaming

Stream: Internet Engineering Task Force (IETF)
RFC: 9113
Obsoletes: 7540, 8740
Category: Standards Track
Published: June 2022
ISSN: 2070-1721
Authors: M. Thomson, Ed. *Mozilla*
C. Benfield, Ed. *Apple Inc.*

Print

RFC 9113
HTTP/2

Abstract

This specification describes an optimized expression of the semantics of the Hypertext Transfer Protocol (HTTP), referred to as HTTP version 2 (HTTP/2). HTTP/2 enables a more efficient use of network resources and a reduced latency by introducing field compression and allowing multiple concurrent exchanges on the same connection.

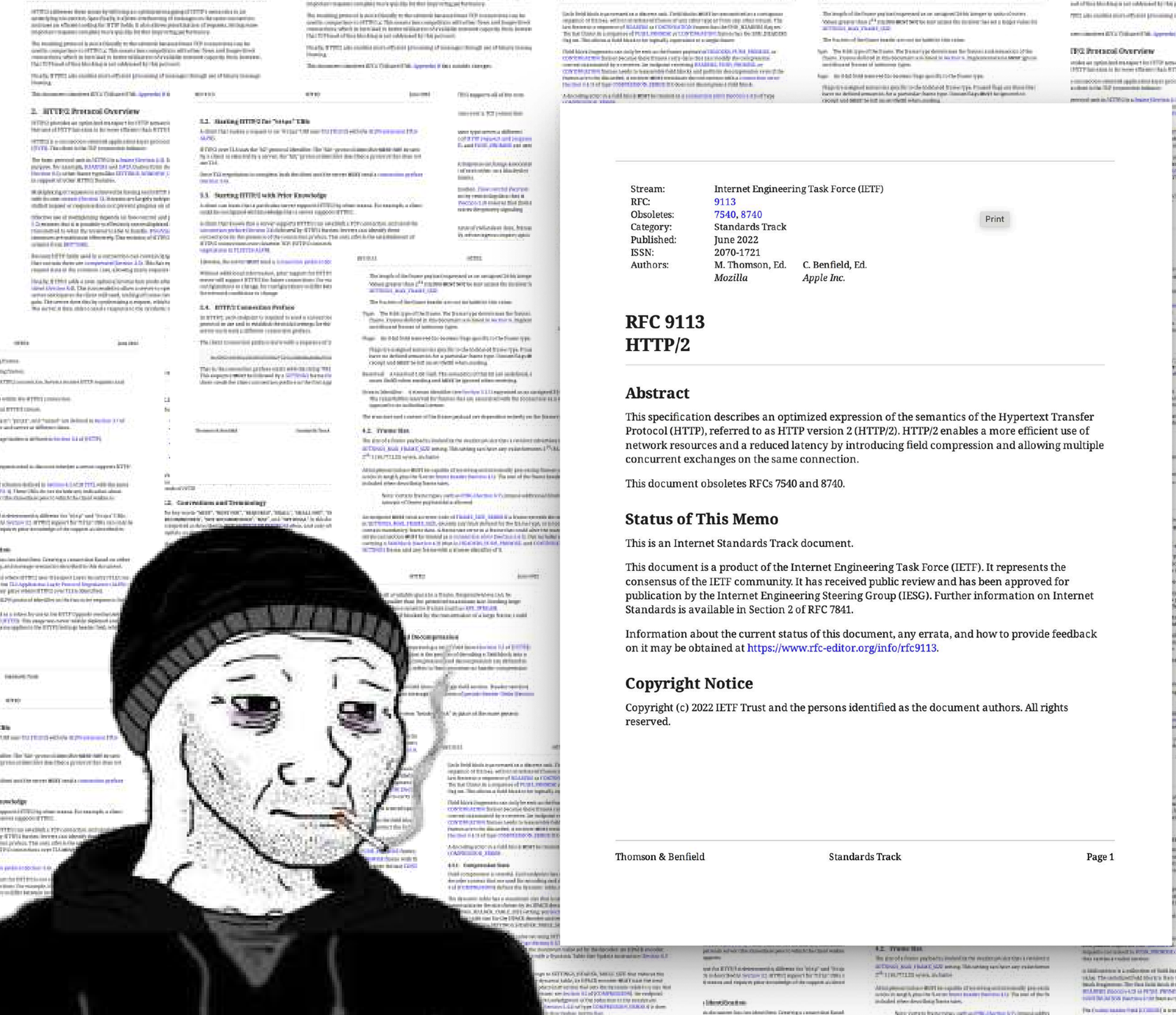
This document obsoletes RFCs 7540 and 8740.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9113.

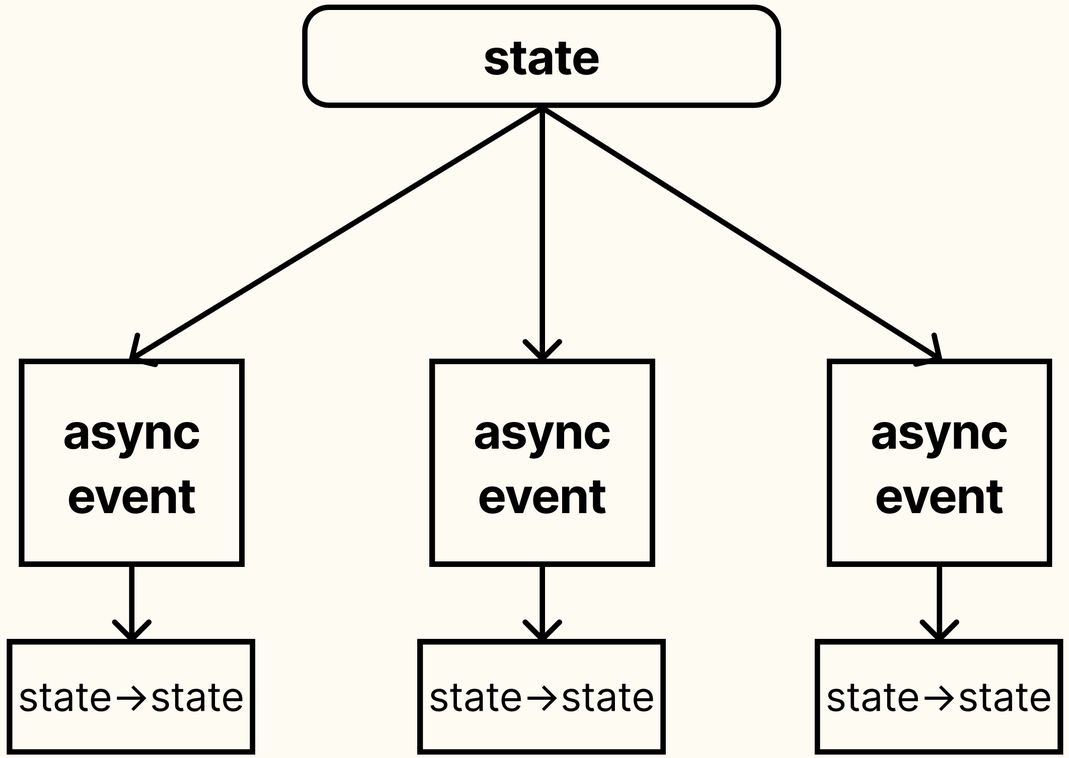


Goals for Implementation:

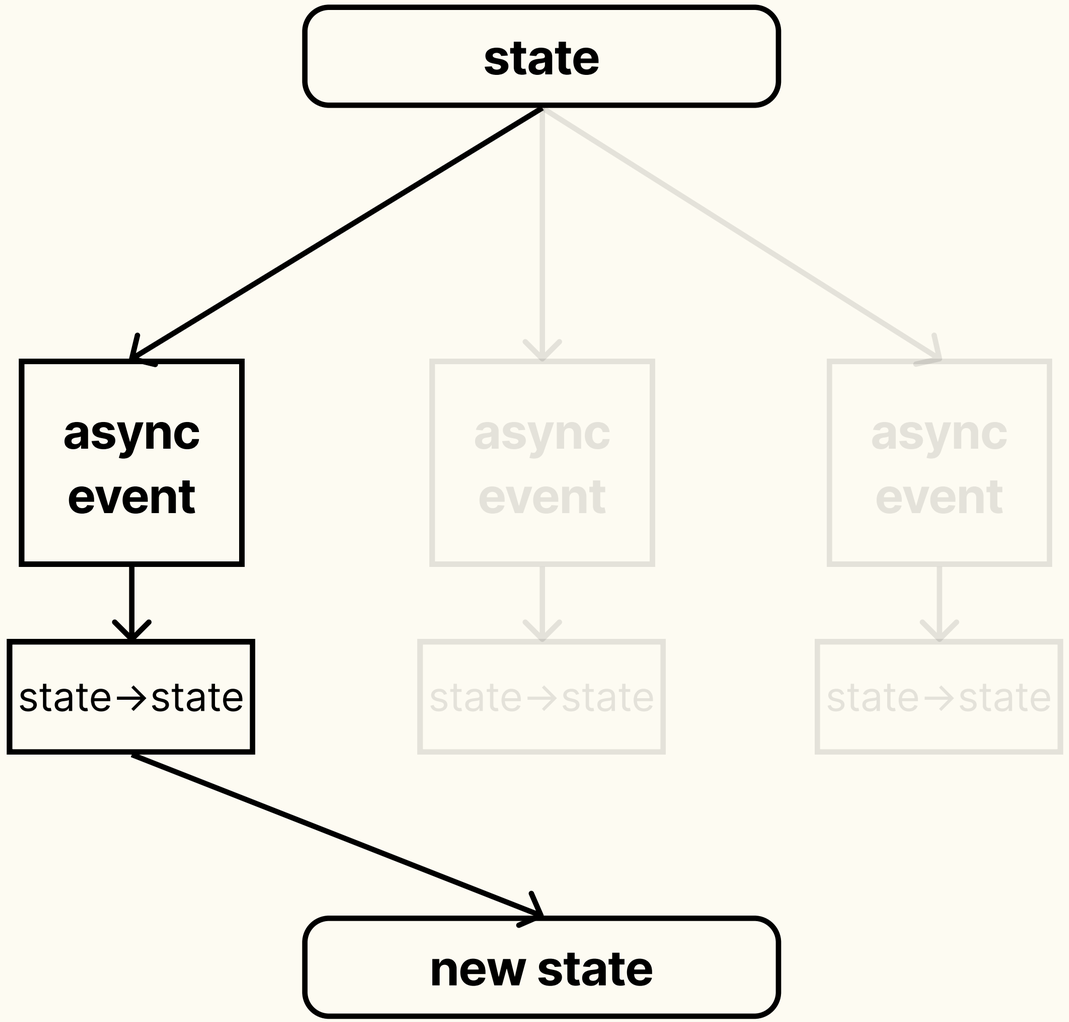
- latest spec - RFC 9113
- effects-based with Eio
- Cstruct’s over bytes
- modular parsers and serializers
- immutable state
- functional API

Immutable State Event Loop

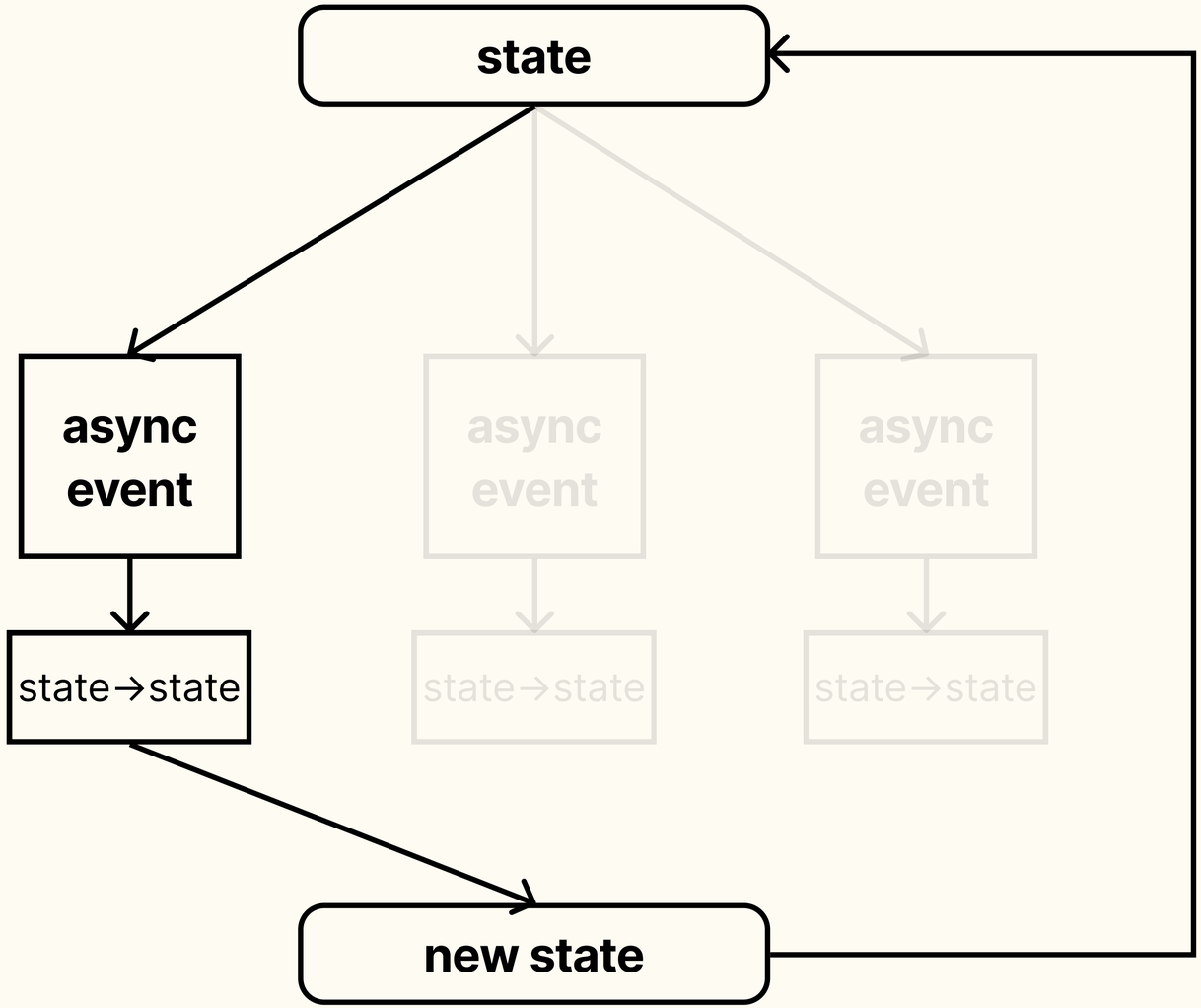
Immutable State Event Loop



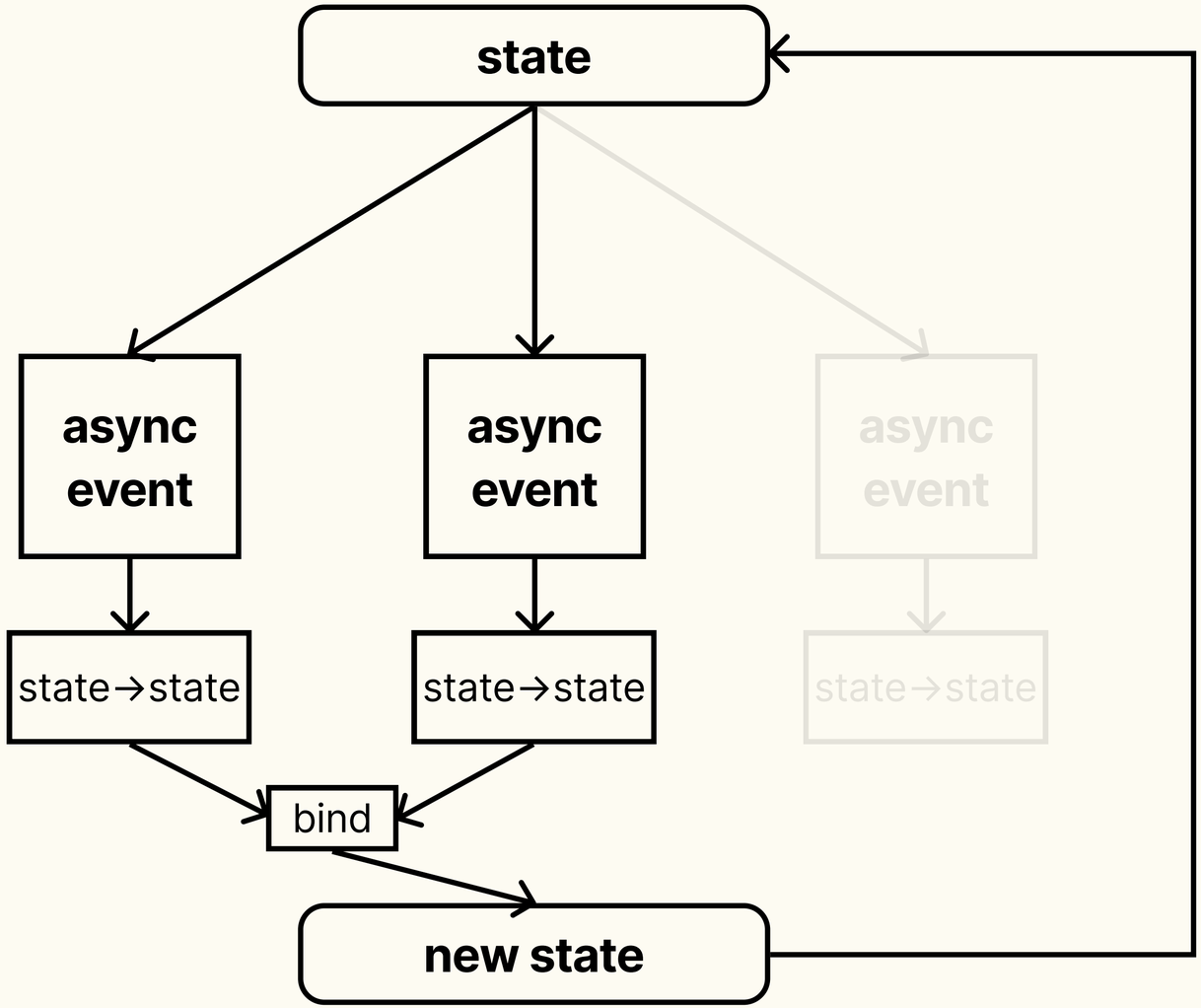
Immutable State Event Loop



Immutable State Event Loop



Immutable State Event Loop



Simple Event Loop

```
let async_event : unit -> state -> state =
```

Simple Event Loop

```
let async_event : unit -> state -> state =
  fun () ->

    (* async op *)
    let bytes_read = Eio.Flow.single_read socket buffer in
```

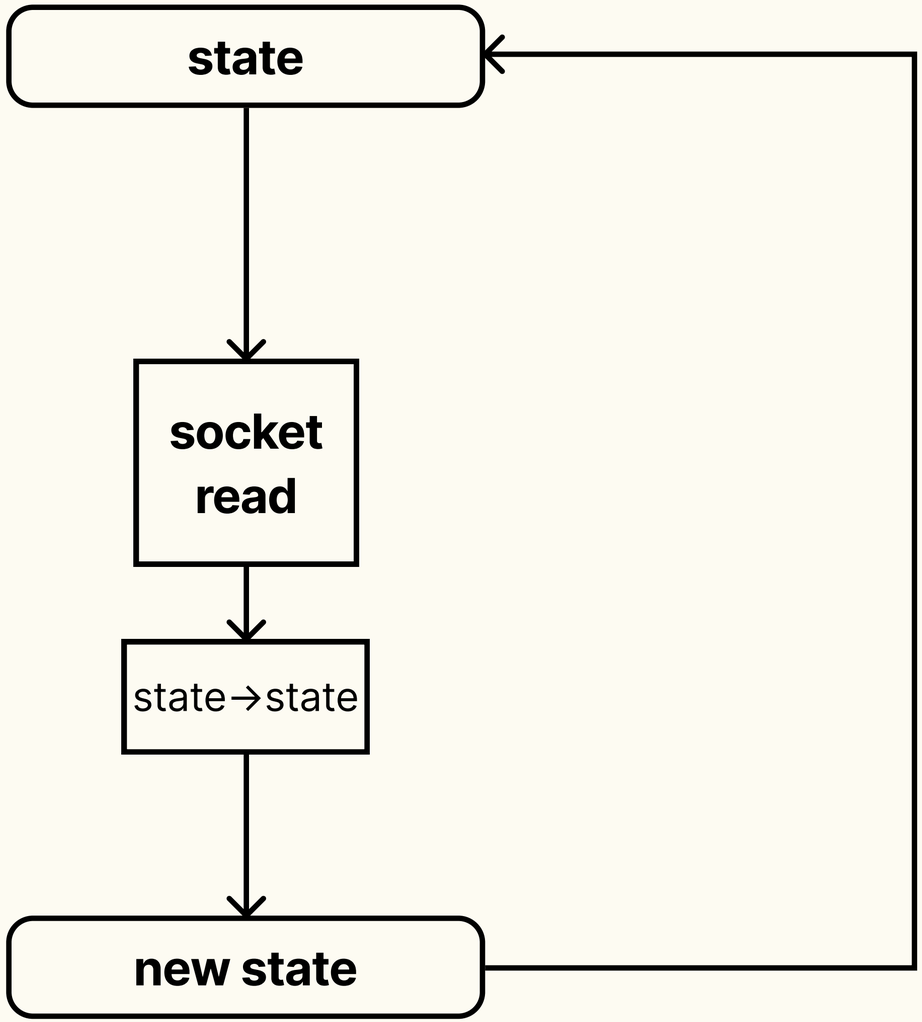
Simple Event Loop

```
let async_event : unit -> state -> state =
  fun () ->

    (* async op *)
    let bytes_read = Eio.Flow.single_read socket buffer in

    fun state ->

      (* change the state *)
      { state with field = val }
```

Simple Event Loop

```
let combine transition1 transition2 =
  (* bind the state transitions *)
  fun state -> transition2 (transition1 state)

let async_event : unit -> state -> state =
  fun () ->

    (* async op *)
    let bytes_read = Eio.Flow.single_read socket buffer in

    fun state ->

      (* change the state *)
      { state with field = val }

let rec runloop : state -> state =
  fun state ->

    (* race async events *)
    let next_state = Eio.Fiber.any ~combine [ async_event; (* other events *) ] in

    runloop (next_state state)
```

HTTP/2 Event Loop



HTTP/2 Event Loop - write/read in “h2”

```
Body.Writer.write_bigstring writer ~off ~len payload;

Body.Writer.close writer;

(* oh no! *)
Body.Writer.write_bigstring writer ~off ~len payload;
```

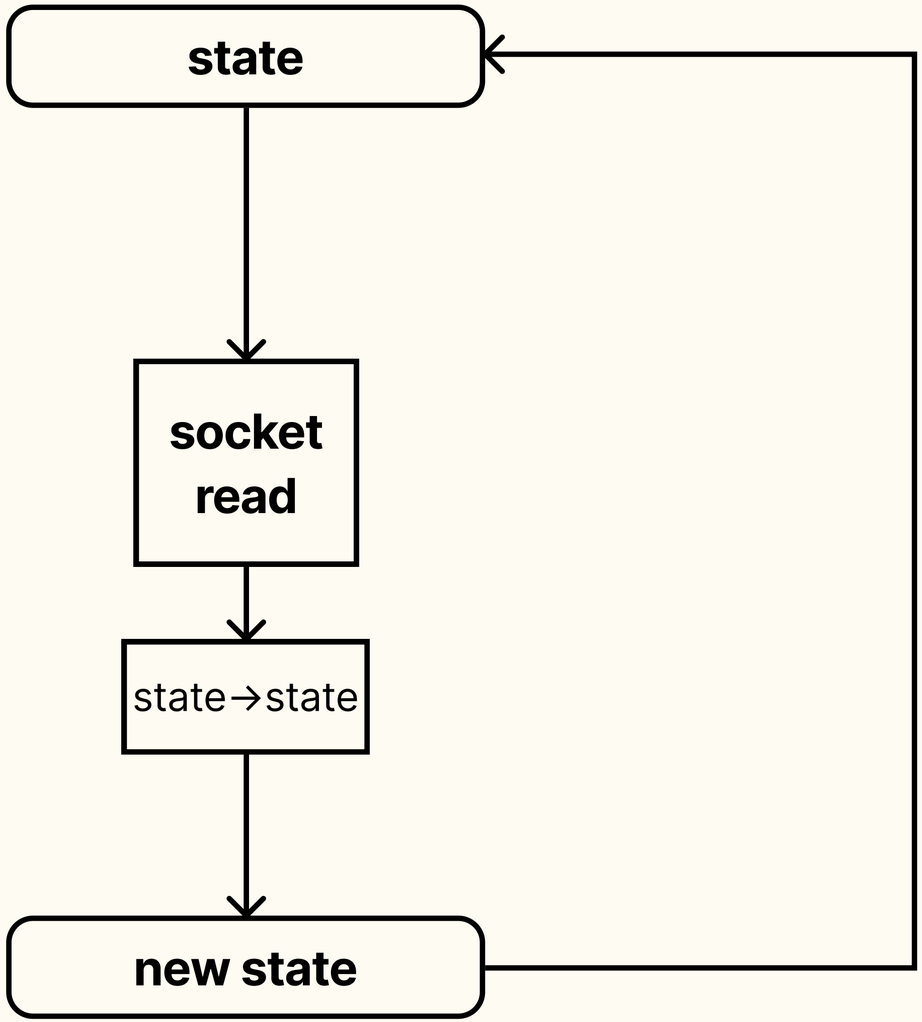
```
let rec aux () =
  Body.Reader.schedule_read reader
    ~on_eof:(fun () -> (* handle EOF *))
    ~on_read:(fun data ~off ~len ->
        (* handle data *)

        (* read more *)
        aux ())
in
aux ()
```

HTTP/2 Event Loop - write/read in the new implementation

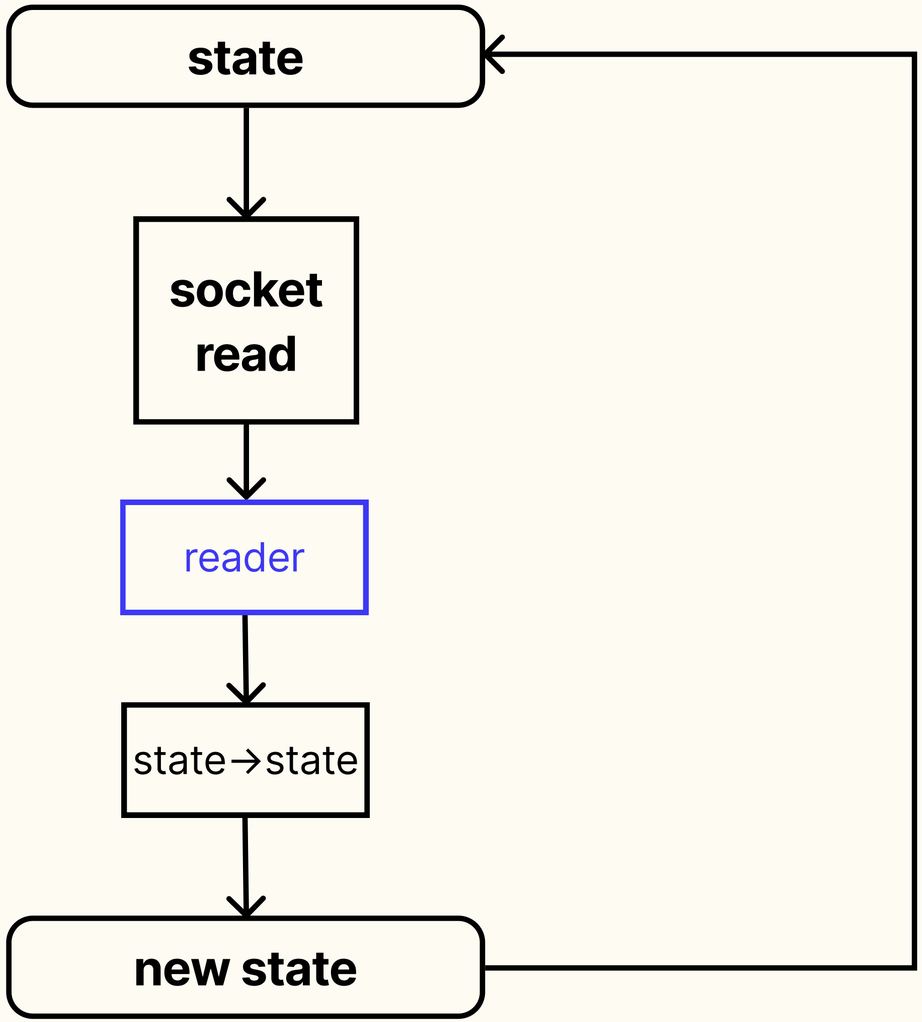
```
let writer : unit -> Cstruct.t =
  fun () ->
    (* get data from somewhere and return *)
    (* e.g. read file *)
    data
```

```
let reader : Cstruct.t option -> unit =
  function
  | Some data ->
    (* handle data *)
    ()
  | None ->
    (* EOF *)
    ()
```

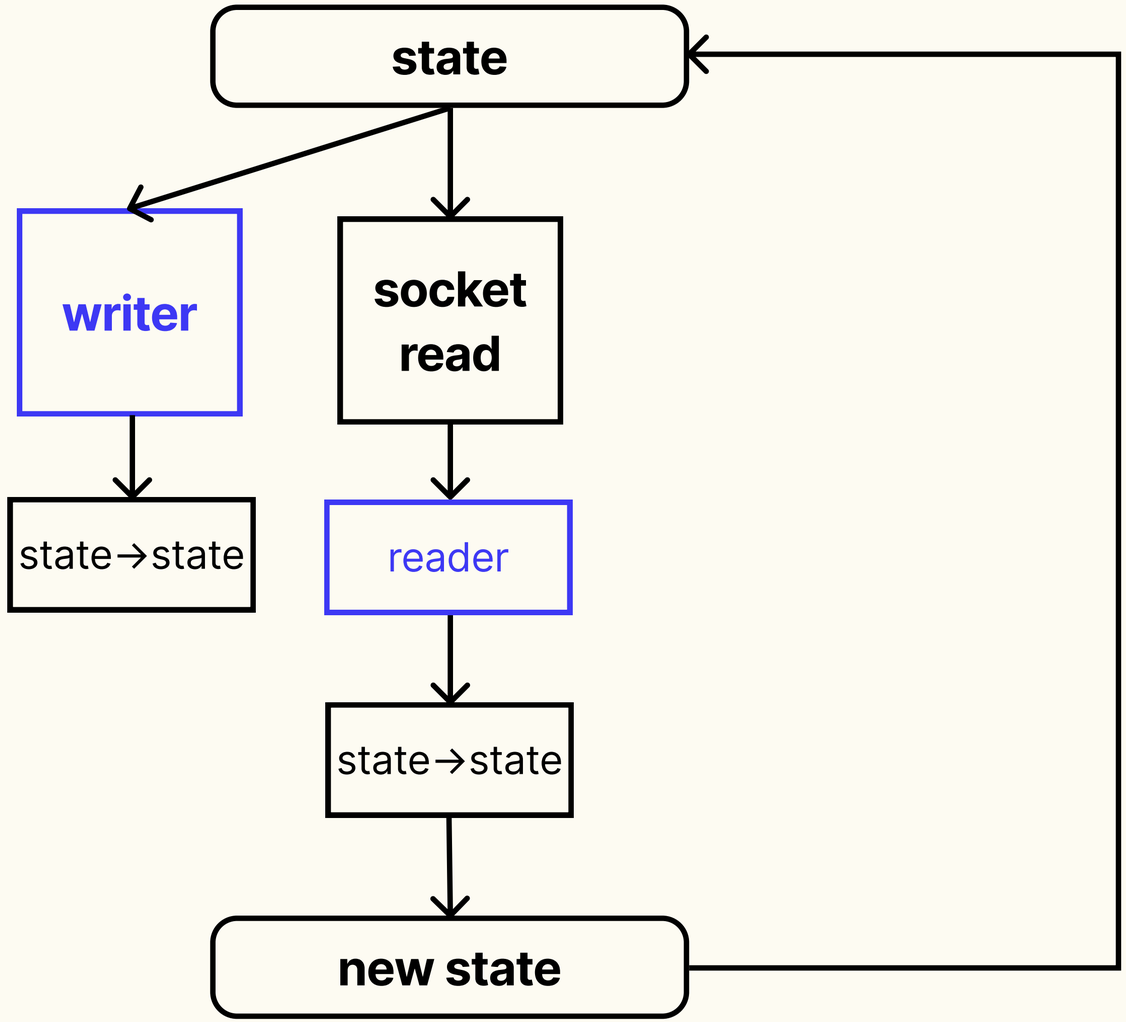
HTTP/2 Event Loop



HTTP/2 Event Loop



HTTP/2 Event Loop



HTTP/2 Event Loop - write/read in the new implementation

```
let writer : unit -> Cstruct.t =
  fun () ->
    (* get data from somewhere and return *)
    (* e.g. read file *)
    data
```

```
let reader : Cstruct.t option -> unit =
  function
  | Some data ->
    (* handle data *)
    ()
  | None ->
    (* EOF *)
    ()
```

HTTP/2 Event Loop - write/read in the new implementation

```
let writer : 'a -> Cstruct.t * 'a =
```

HTTP/2 Event Loop - write/read in the new implementation

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

let writer : 'a -> Cstruct.t * 'a =
```

HTTP/2 Event Loop - write/read in the new implementation

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

let writer : 'a -> Cstruct.t * 'a =
  (* match on context, make decision *)
  function
    | Init metadata1 ->
      (* process init *)

      (data, Ongoing metadata2)
    | Ongoing metadata2 ->
      (* process ongoing stage *)
      ...

      (data, next_stage)
```

HTTP/2 Event Loop - write/read in the new implementation

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

let writer : 'a -> Cstruct.t * 'a =
  (* match on context, make decision *)
  function
    | Init metadata1 ->
      (* process init *)

      (data, Ongoing metadata2)
    | Ongoing metadata2 ->
      (* process ongoing stage *)
      ...

      (data, next_stage)
```

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

let reader : Cstruct.t option -> 'a -> 'a =
```

HTTP/2 Event Loop - write/read in the new implementation

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

let writer : 'a -> Cstruct.t * 'a =
  (* match on context, make decision *)
  function
    | Init metadata1 ->
      (* process init *)

      (data, Ongoing metadata2)
    | Ongoing metadata2 ->
      (* process ongoing stage *)
      ......

      (data, next_stage)
```

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

let reader : Cstruct.t option -> 'a -> 'a =
  fun data context ->
    (*
      return new stage
      based on current one and data
    *)
    match data, context with
    | Some data, Init metadata1 as ctx ->
      (* process data in Init stage *)

      ctx
    | Some data, Ongoing metadata2 ->
      (* process data in Ongoing stage *)
      ......

      next_stage
    | None, Ongoing metadata3 ->
      (* process EOF in Ongoing *)

      next_stage
```

HTTP/2 Event Loop - write/read in the new implementation

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

let writer : 'a -> Cstruct.t * 'a =
  (* match on context, make decision *)
  function
    | Init metadata1 ->
      (* process init *)

      (data, Ongoing metadata2)
    | Ongoing metadata2 ->
      (* process ongoing stage *)
      ......

      (data, next_stage)
  ..
```

HTTP/2 Event Loop - write/read in the new implementation

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

let writer : 'a -> Cstruct.t * 'a =
  (* match on context, make decision *)
  function
    | Init metadata1 ->
      (* process init *)

      (data, Ongoing metadata2)
    | Ongoing metadata2 ->
      (* process ongoing stage *)
      ...

      (data, next_stage)
```

```
let handler : Cstruct.t option -> 'a -> 'a action list =
```

HTTP/2 Event Loop - write/read in the new implementation

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

let writer : 'a -> Cstruct.t * 'a =
  (* match on context, make decision *)
  function
    | Init metadata1 ->
      (* process init *)

      (data, Ongoing metadata2)
    | Ongoing metadata2 ->
      (* process ongoing stage *)
      ......

      (data, next_stage)
```

```
type 'a action =
  [ `NewContext of 'a
  | `Data of Cstruct.t list
  | `Close ]

let handler : Cstruct.t option -> 'a -> 'a action list =
```

HTTP/2 Event Loop - write/read in the new implementation

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

let writer : 'a -> Cstruct.t * 'a =
  (* match on context, make decision *)
  function
    | Init metadata1 ->
      (* process init *)

      (data, Ongoing metadata2)
    | Ongoing metadata2 ->
      (* process ongoing stage *)
      ......

      (data, next_stage)
```

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

type 'a action =
  [ `NewContext of 'a
  | `Data of Cstruct.t list
  | `Close ]

let handler : Cstruct.t option -> 'a -> 'a action list =
  fun data context ->
    (* return new stage based on current one and data *)
    match data, context with
    | Some data, Init metadata1 as ctx ->
      (* process data in Init stage *)

      [ `NewContext ctx; `Data data2 ]
    | Some data, Ongoing metadata2 ->
      (* process data in Ongoing stage *)

      [ `NewContext next_stage ]
    | None, Ongoing metadata3 ->
      (* process EOF in Ongoing *)

      [ `NewContext next_stage ]

  | _ -> [ `Close ]
```

HTTP/2 Event Loop - write/read in the new implementation

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

let generator : 'a -> Cstruct.t * 'a =
  (* match on context, make decision *)
  function
    | Init metadata1 ->
      (* process init *)

      (data, Ongoing metadata2)
    | Ongoing metadata2 ->
      (* process ongoing stage *)

      (data, next_stage)
```

(optional)

```
type call_stage =
  | Init of { field1 : typ }
  | Ongoing of { field2 : typ }

type 'a action =
  [ `NewContext of 'a
  | `Data of Cstruct.t list
  | `Close ]

let handler : Cstruct.t option -> 'a -> 'a action list =
  fun data context ->
    (* return new stage based on current one and data *)
    match data, context with
    | Some data, Init metadata1 as ctx ->
      (* process data in Init stage *)

      [ `NewContext ctx; `Data data2 ]
    | Some data, Ongoing metadata2 ->
      (* process data in Ongoing stage *)

      [ `NewContext next_stage ]
    | None, Ongoing metadata3 ->
      (* process EOF in Ongoing *)

      [ `NewContext next_stage ]

  | _ -> [ `Close ]
```

Creating new connection

h2

```
let conn = H2_eio.Client.create_connection ~sw ~error_handler socket (* forks a fiber *)

H2_eio.Client.request conn request ~response_handler
```

Creating new connection

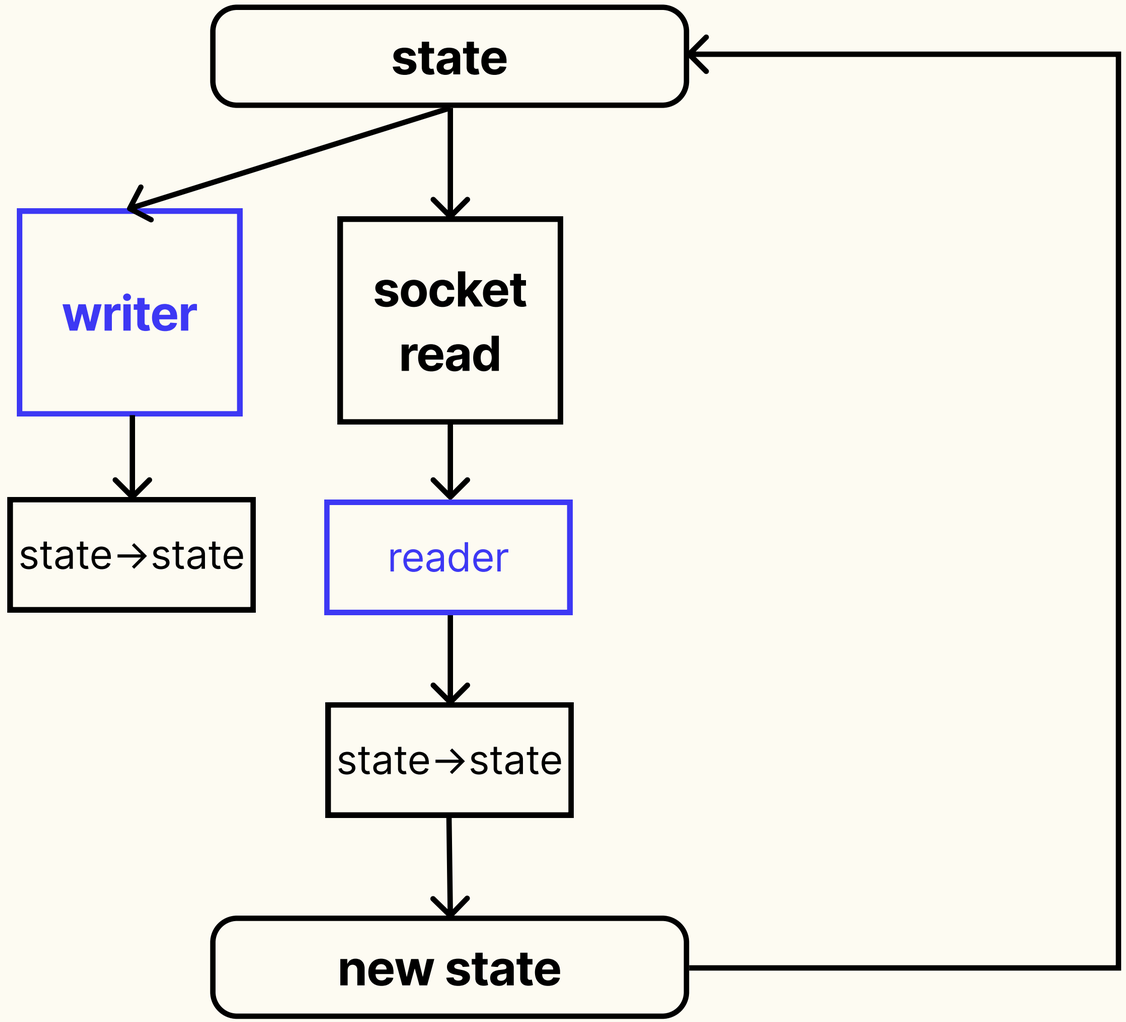
h2

```
let conn = H2_eio.Client.create_connection ~sw ~error_handler socket (* forks a fiber *)

H2_eio.Client.request conn request ~response_handler
```

new impl

```
let initial_iteration = Haha.Client.connect socket in

let rec runloop : Haha.Client.iteration -> unit =
  function
  | InProgress continue ->
    let next_iter = continue [ Request request; Shutdown ] in
    runloop next_iter
  | Error _ -> (* handle connection error *) ()
  | End -> (* handle shutdown *) ()
```

HTTP/2 Event Loop

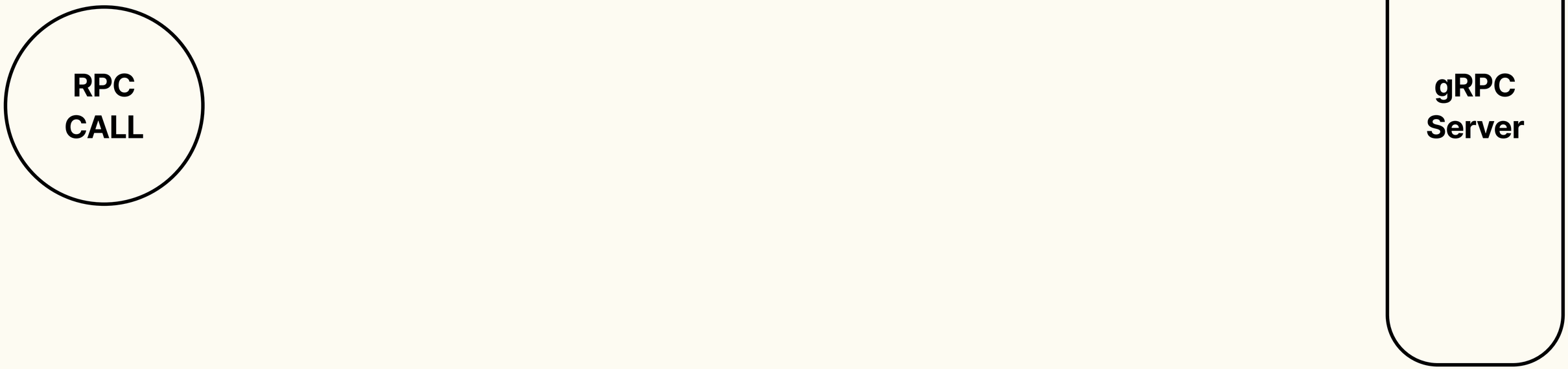


gRPC Channel

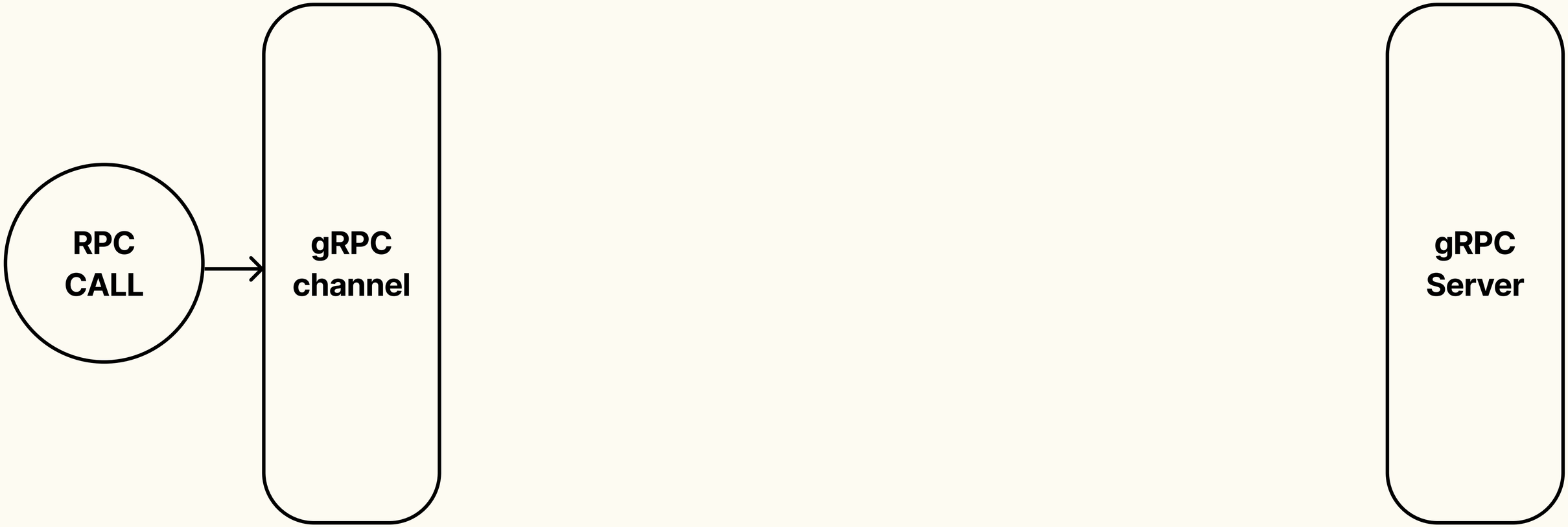
Client-side abstraction in gRPC for managing multiple HTTP/2 connections and streams in those connections.

HTTP/2 Connection

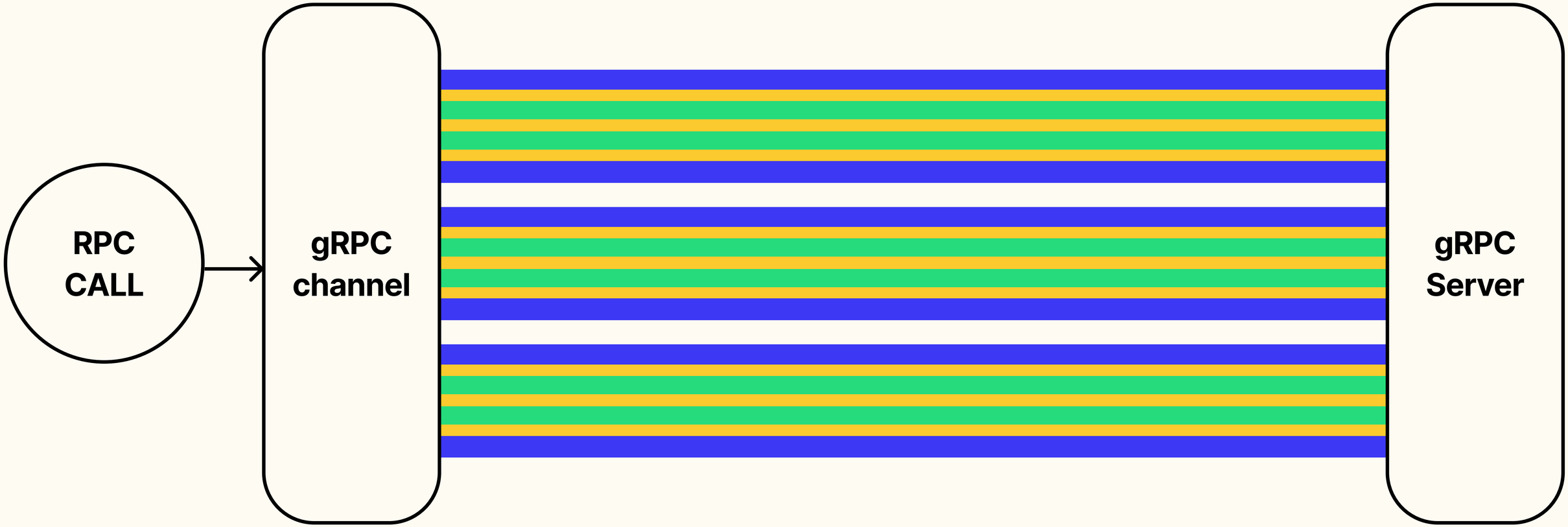
gRPC Channel



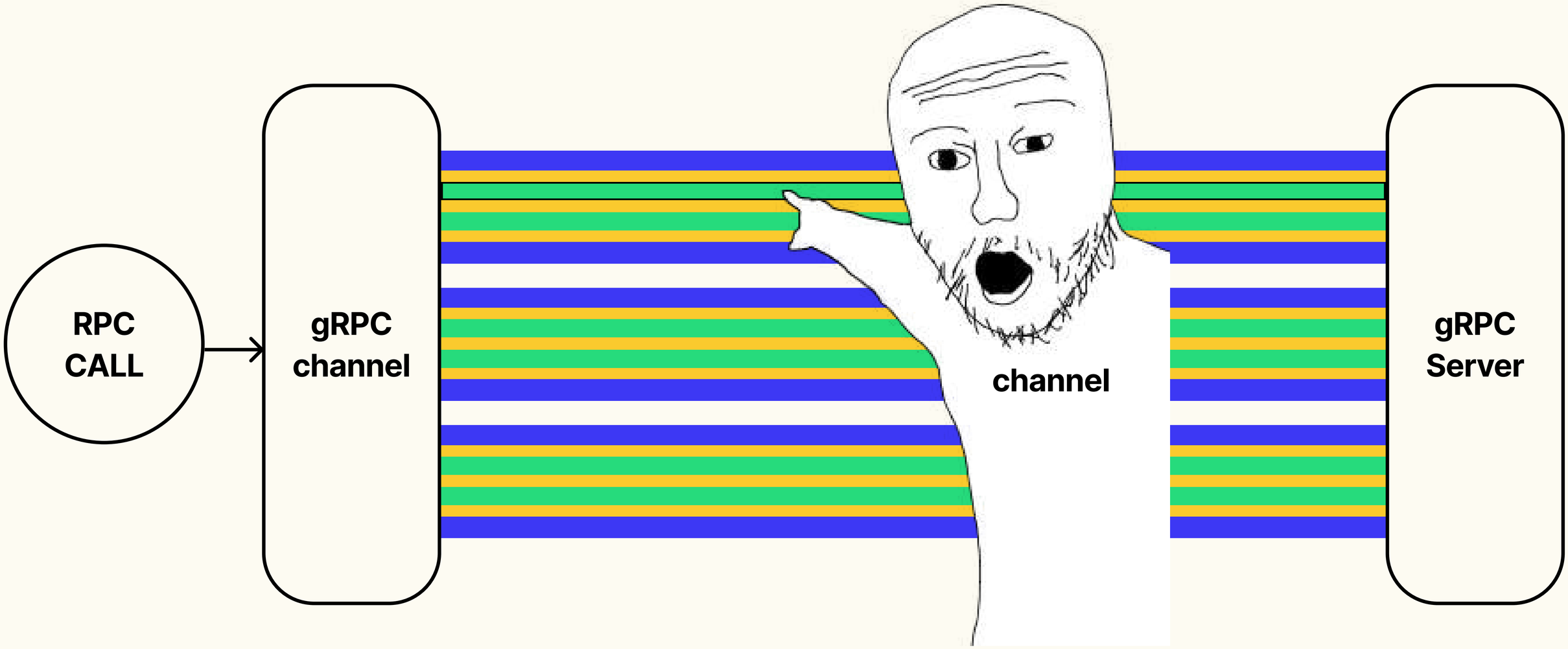
gRPC Channel



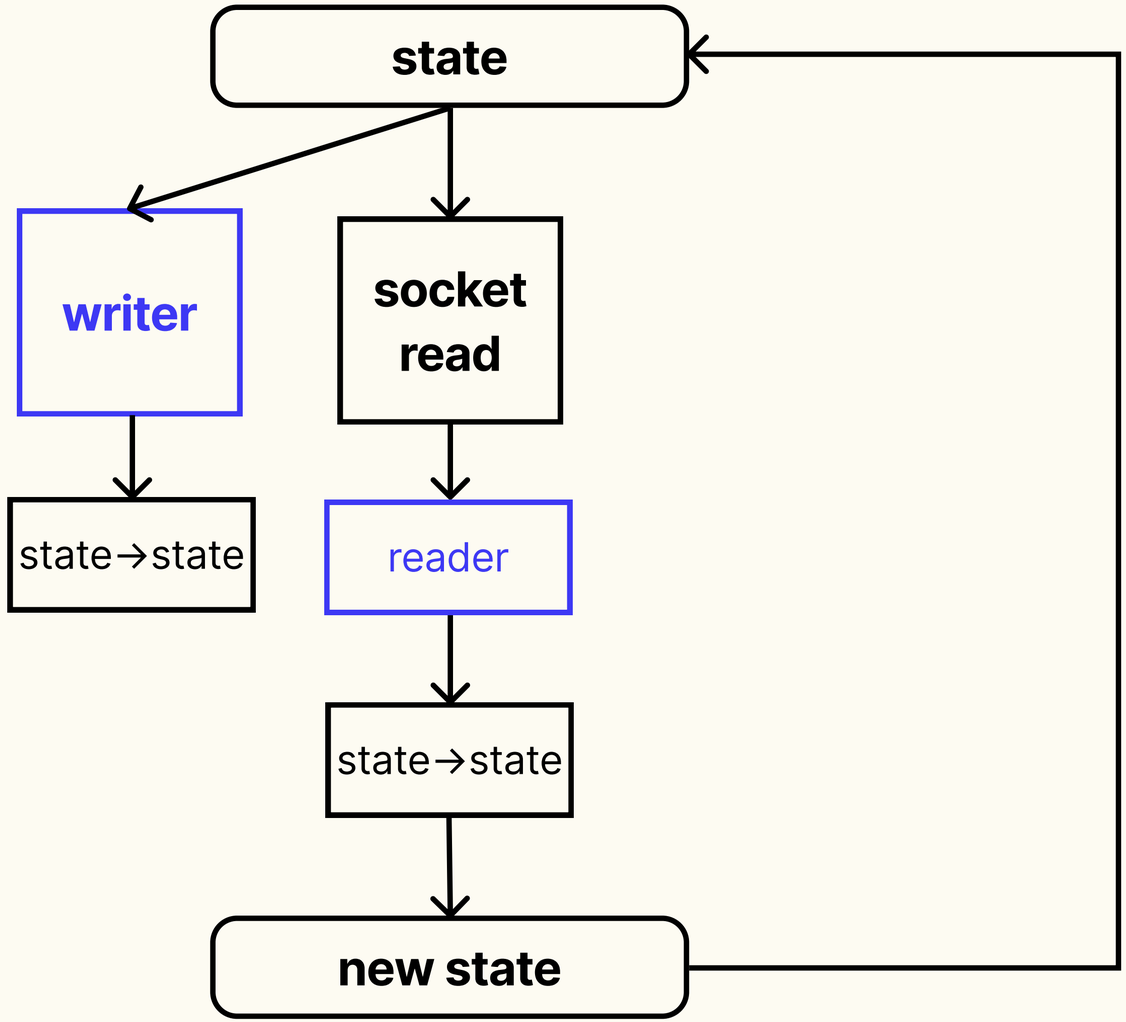
gRPC Channel



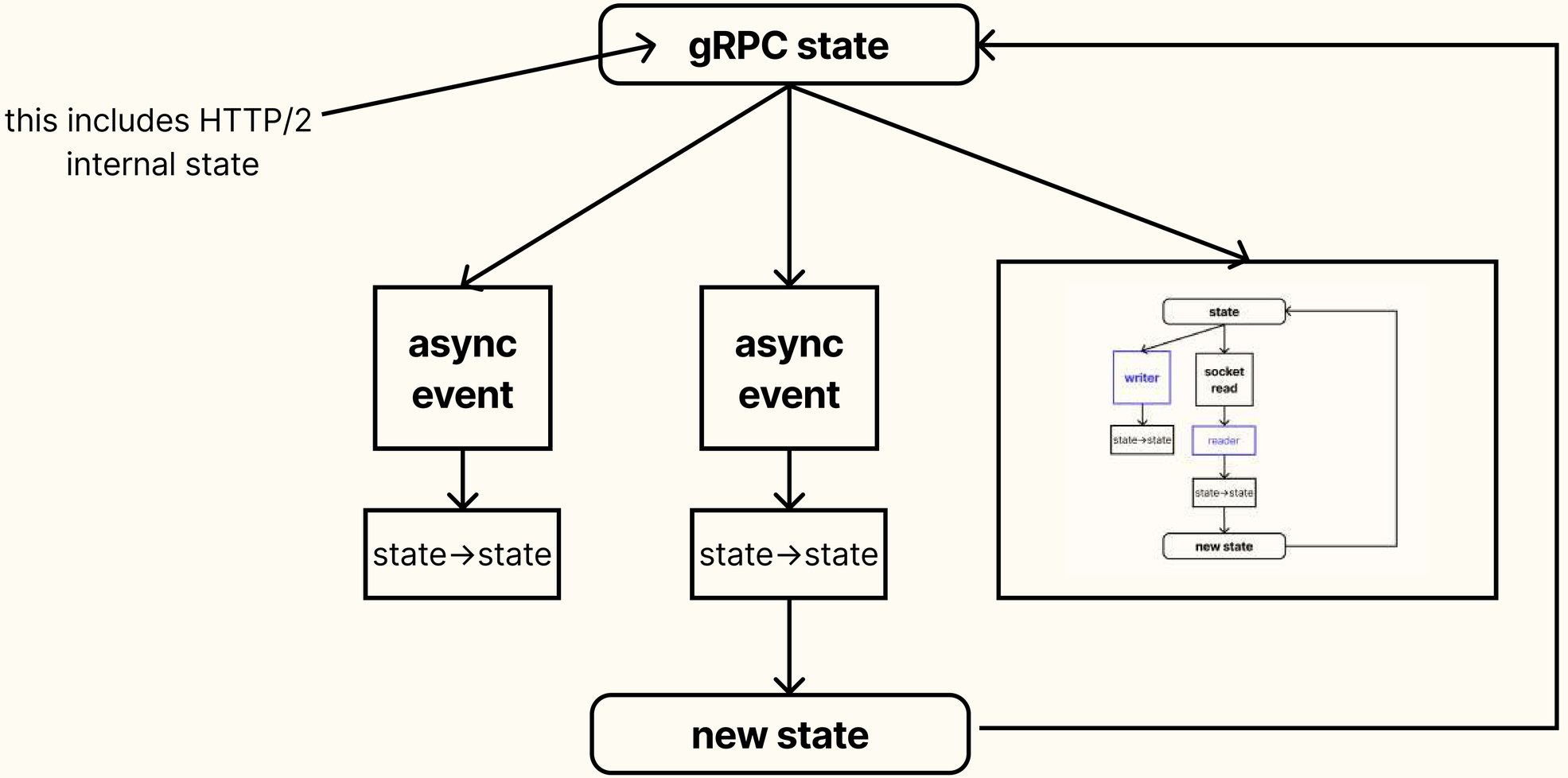
gRPC Channel



HTTP/2 Event Loop



Bigger Event Loop - gRPC, web framework



ocaml-grpc's channel - event loop

```
Fiber.fork ~sw (fun () ->
    let rec runloop state =
      let new_connection = start_connection ~connect_socket in

      let new_stream_event : event =
        make_new_stream_event ~new_connection ~max_streams ~request_stream
      in
      let shutdown_event : event = make_shutdown_event ~shutdown_promise in
      let connections_events : event list =
        List.map make_connections_event state.connection_pool
      in

      let transition : transition =
        match (state.shutdown, connections_events) with
        | true, [] -> fun _ -> None
        | true, _ ->
            Fiber.any ~combine (new_stream_event :: connections_events)
        | _ ->
            Fiber.any ~combine
              (shutdown_event :: new_stream_event :: connections_events)
      in

      Option.iter runloop (transition state)
    in

    runloop { connection_pool = []; shutdown = false; next_id = 0 });
```

ocaml-grpc’s channel - event loop

```
Fiber.fork ~sw (fun () ->
    let rec runloop state =
      let new_connection = start_connection ~connect_socket in

      let new_stream_event : event =
        make_new_stream_event ~new_connection ~max_streams ~request_stream
      in
      let shutdown_event : event = make_shutdown_event ~shutdown_promise in
      let connections_events : event list =
        List.map make_connections_event state.connection_pool
      in

      let transition : transition =
        match (state.shutdown, connections_events) with
        | true, [] -> fun _ -> None
        | true, _ ->
            Fiber.any ~combine (new_stream_event :: connections_events)
        | _ ->
            Fiber.any ~combine
              (shutdown_event :: new_stream_event :: connections_events)
      in

      Option.iter runloop (transition state)
    in

    runloop { connection_pool = []; shutdown = false; next_id = 0 });
```

ocaml-grpc's channel - event loop

```
Fiber.fork ~sw (fun () ->
    let rec runloop state =
      let new_connection = start_connection ~connect_socket in

      let new_stream_event : event =
        make_new_stream_event ~new_connection ~max_streams ~request_stream
      in
      let shutdown_event : event = make_shutdown_event ~shutdown_promise in
      let connections_events : event list =
        List.map make_connections_event state.connection_pool
      in

      let transition : transition =
        match (state.shutdown, connections_events) with
        | true, [] -> fun _ -> None
        | true, _ ->
            Fiber.any ~combine (new_stream_event :: connections_events)
        | _ ->
            Fiber.any ~combine
              (shutdown_event :: new_stream_event :: connections_events)
      in

      Option.iter runloop (transition state)
    in

    runloop { connection_pool = []; shutdown = false; next_id = 0 });
```

ocaml-grpc's channel - event loop

```
Fiber.fork ~sw (fun () ->
    let rec runloop state =
      let new_connection = start_connection ~connect_socket in

      let new_stream_event : event =
        make_new_stream_event ~new_connection ~max_streams ~request_stream
      in
      let shutdown_event : event = make_shutdown_event ~shutdown_promise in
      let connections_events : event list =
        List.map make_connections_event state.connection_pool
      in

      let transition : transition =
        match (state.shutdown, connections_events) with
        | true, [] -> fun _ -> None
        | true, _ ->
            Fiber.any ~combine (new_stream_event :: connections_events)
        | _ ->
            Fiber.any ~combine
              (shutdown_event :: new_stream_event :: connections_events)
      in

      Option.iter runloop (transition state)
    in

    runloop { connection_pool = []; shutdown = false; next_id = 0 });
```

ocaml-grpc's channel - event loop

```
Fiber.fork ~sw (fun () ->
    let rec runloop state =
      let new_connection = start_connection ~connect_socket in

      let new_stream_event : event =
        make_new_stream_event ~new_connection ~max_streams ~request_stream
      in
      let shutdown_event : event = make_shutdown_event ~shutdown_promise in
      let connections_events : event list =
        List.map make_connections_event state.connection_pool
      in

      let transition : transition =
        match (state.shutdown, connections_events) with
        | true, [] -> fun _ -> None
        | true, _ ->
            Fiber.any ~combine (new_stream_event :: connections_events)
        | _ ->
            Fiber.any ~combine
              (shutdown_event :: new_stream_event :: connections_events)
      in

      Option.iter runloop (transition state)
    in

    runloop { connection_pool = []; shutdown = false; next_id = 0 });
```

ocaml-grpc's channel - HTTP/2 event

```
let make_connections_event : connection -> event =
 fun { id = id'; pending_inputs = inputs; next_iter; _ } () ->
  let iteration = next_iter inputs in

  fun state ->
    let new_pool =
      match iteration.state with
      | End -> List.filter (fun { id; _ } -> id <> id') state.connection_pool
      | Error _ ->
          List.filter (fun { id; _ } -> id <> id') state.connection_pool
      | InProgress next_iter ->
          List.map
            (fun conn ->
              if conn.id = id' then
                let new_conn =
                  {
                    conn with
                    next_iter;
                    pending_inputs =
                      List.(drop (length inputs) conn.pending_inputs);
                    open_streams = iteration.active_streams;
                  }
                in

                if
                  new_conn.open_streams < 1
                  && List.length state.connection_pool > 1
                then shutdown_connection new_conn
                else new_conn
```

ocaml-grpc's channel - HTTP/2 event

```
let make_connections_event : connection -> event =
 fun { id = id'; pending_inputs = inputs; next_iter; _ } () ->
  let iteration = next_iter inputs in

  fun state ->
    let new_pool =
      match iteration.state with
      | End -> List.filter (fun { id; _ } -> id <> id') state.connection_pool
      | Error _ ->
          List.filter (fun { id; _ } -> id <> id') state.connection_pool
      | InProgress next_iter ->
          List.map
            (fun conn ->
              if conn.id = id' then
                let new_conn =
                  {
                    conn with
                    next_iter;
                    pending_inputs =
                      List.(drop (length inputs) conn.pending_inputs);
                    open_streams = iteration.active_streams;
                  }
                in

                if
                  new_conn.open_streams < 1
                  && List.length state.connection_pool > 1
                then shutdown_connection new_conn
                else new_conn
```

ocaml-grpc's channel - HTTP/2 event

```
let make_connections_event : connection -> event =
 fun { id = id'; pending_inputs = inputs; next_iter; _ } () ->
  let iteration = next_iter inputs in

  fun state ->
    let new_pool =
      match iteration.state with
      | End -> List.filter (fun { id; _ } -> id <> id') state.connection_pool
      | Error _ ->
          List.filter (fun { id; _ } -> id <> id') state.connection_pool
      | InProgress next_iter ->
          List.map
            (fun conn ->
              if conn.id = id' then
                let new_conn =
                  {
                    conn with
                    next_iter;
                    pending_inputs =
                      List.(drop (length inputs) conn.pending_inputs);
                    open_streams = iteration.active_streams;
                  }
                in

                if
                  new_conn.open_streams < 1
                  && List.length state.connection_pool > 1
                then shutdown_connection new_conn
                else new_conn
```

ocaml-grpc's channel - HTTP/2 event

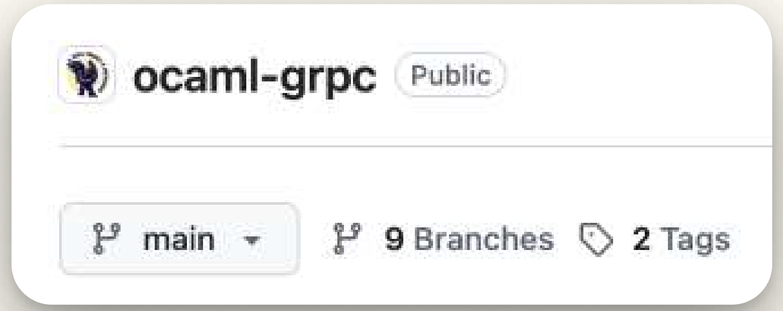
```
let make_connections_event : connection -> event =
 fun { id = id'; pending_inputs = inputs; next_iter; _ } () ->
  let iteration = next_iter inputs in

  fun state ->
    let new_pool =
      match iteration.state with
      | End -> List.filter (fun { id; _ } -> id <> id') state.connection_pool
      | Error _ ->
          List.filter (fun { id; _ } -> id <> id') state.connection_pool
      | InProgress next_iter ->
          List.map
            (fun conn ->
              if conn.id = id' then
                let new_conn =
                  {
                    conn with
                    next_iter;
                    pending_inputs =
                      List.(drop (length inputs) conn.pending_inputs);
                    open_streams = iteration.active_streams;
                  }
                in

                if
                  new_conn.open_streams < 1
                  && List.length state.connection_pool > 1
                then shutdown_connection new_conn
                else new_conn
```

ocaml-grpc API

```
let channel = Grpc.Channel.create ~sw ~net:env#net "localhost:8080" in

let handler : rpc_input -> 'a -> 'a Grpc.action list =
  fun msg context ->
    (* return new stage based on current one and data *)
    match msg, context with
    | MetaRequest meta, Init metadata1 ->
      (* e.g. init a new call *)

      [ `Msg (Initialized meta2) ]
    | Audio cstruct, Ongoing metadata2 ->
      (* e.g. accumulate transcription *)

      [ `NewContext next_stage ]
    | TransferRequest meta, Ongoing metadata3 ->
      (* e.g. transfer the call *)

      [ `NewContext next_stage ]

  | _ -> [ `Close ]
in

let writer : 'a -> rpc_output * 'a =
  function
  | Ongoing meta as ctx ->
    (* e.g. read RTP *)
    (Audio rtp_audio, ctx)
  | _ -> ()
in

match MyCoolService.my_cool_rpc ~channel ~writer handler with
| Ok final_context -> (* e.g. save info to DB *) ()
| Error (err, final_context) -> (* e.g. handle err and save *) ()
```

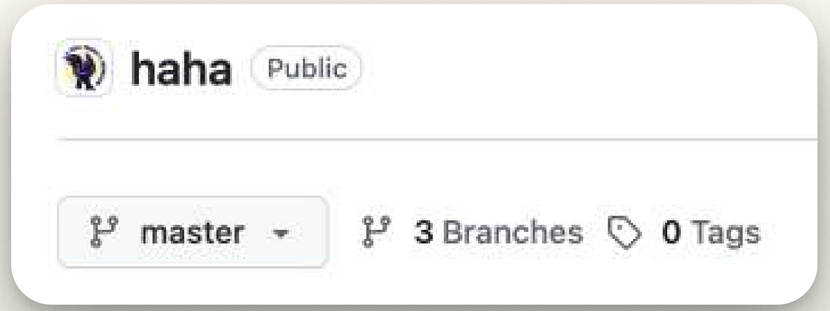
gRPC

dialohq/ocaml-grpc



HTTP/2

dialohq/haha



We’re hiring at Dialo!

Thank you!

dialo.ai (we’re hiring!)

adamchol.com (very info-rich webpage, btw)

@adamchol_

@adamchol

me@adamchol.com