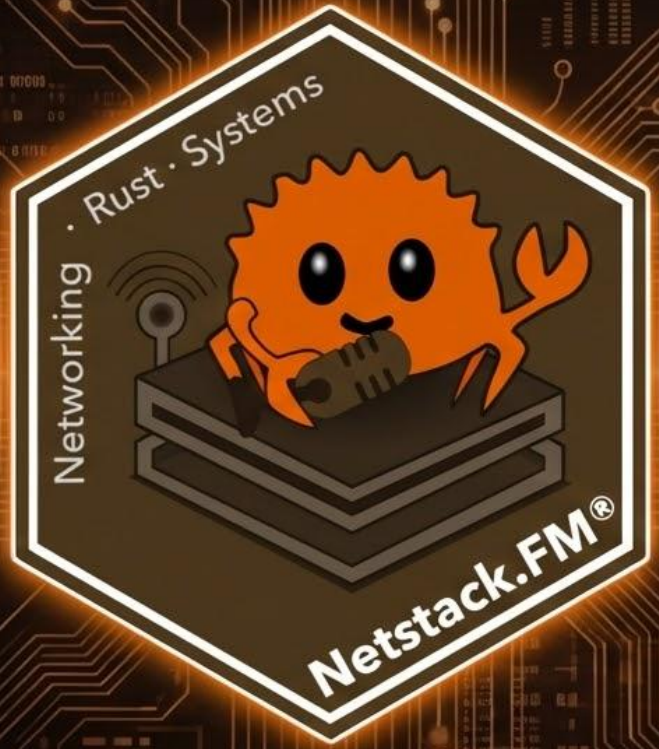

ramaproxy.org
rama®
rama
FOSDEM 2026
Sun, 1st of February
Speaker
Glen De Cauwsemaecker
Co-Founder Plabayo

> plabayo.tech/

Plabayo is a software studio based in Gent, Belgium, focused on building resilient, interoperable, and secure digital infrastructure.



A podcast about networking, Rust,
and everything in between.

netstack.fm

ラマ | rama

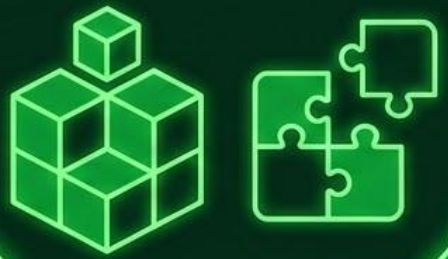
a modular service framework to move
and transform packets

github.com/plabayo/rama

Why Rama?

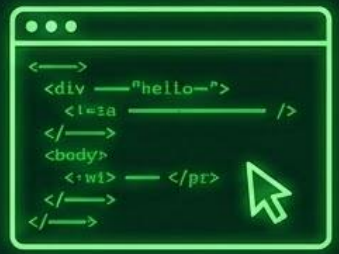
https://ramaproxy.org/book/why_rama.html

Use an off the shelf solution



VS

Write it from scratch



explicit flow reflected in code

```rust
let http_request_svc = (
    TraceLayer::new_for_http(),
    ConsumeErrLayer::default(),
    ProxyAuthLayer::new(basic!("tom", "clancy")),
    UpgradeLayer::new(
        exec.clone(),
        MethodMatcher::CONNECT,
        service_fn(http_connect_accept),
        ConsumeErrLayer::default().into_layer(Forwarder::ctx(exec)),
    ),
    RemoveResponseHeaderLayer::hop_by_hop(),
    RemoveRequestHeaderLayer::hop_by_hop(),
)
    .into_layer(service_fn(http_plain_proxy));

let http_server = HttpServer::auto(exec.clone()).service(http_request_svc);


let tls_acceptor = TlsAcceptorService::new(tls_service_data, http_server.clone(), true);
let auto_tls_acceptor = TlsPeekRouter::new(tls_acceptor).with_fallback(http_server);

let socks5_acceptor =
    Socks5Acceptor::default().with_authorizer(basic!("john", "secret").into_authorizer());
let auto_socks5_acceptor =
    Socks5PeekRouter::new(socks5_acceptor).with_fallback(auto_tls_acceptor);


let tcp_service = TcpListener::bind("127.0.0.1:62029", exec.clone())
    .await
    .expect("bind http+https+socks5+socks5h proxy to 127.0.0.1:62029");

graceful.spawn_task(tcp_service.serve(auto_socks5_acceptor));
```

http proxy

http

[tls]

[socks5]

tcp

everything is a service

```
async fn serve(input) -> Result<Output, Error>
```

```rust
/// A [`Service`] that produces rama services,
/// to serve given an input, be it transport layer Inputs or application layer http requests,
/// or something else entirely.
pub trait Service<Input>: Sized + Send + Sync + 'static {
    /// The type of the output returned by the service.
    type Output: Send + 'static;

    /// The type of error returned by the service.
    type Error: Send + 'static;

    /// Serve an output or an error for the given input
    fn serve(
        &self,
        input: Input,
    ) -> impl Future<Output = Result<Self::Output, Self::Error>> + Send + '_;

    /// Box this service to allow for dynamic dispatch.
    fn boxed(self) -> BoxService<Input, Self::Output, Self::Error> {
        BoxService::new(self)
    }
}
```

finagle (~2011)

```
abstract class
  Service[-Req, +Rep]
  extends (Req) =>
  Future[Rep]
  with Closable

Companion object Service
```

tower (~2016) & tower-async (~2018)

```
pub trait Service<Request> {
  ...
  fn poll_ready(...) ...
  fn call(...) -> Self::Future;
}
```

```
pub trait Service<Request> {
  ...
  fn call(&self, req: Request)
    -> impl Future<Output =
      Result<Self::Response,
      Self::Error>>;
}
```

rama 2024-2026

```
pub trait Service<Input>: Sized
  + Send + Sync + 'static {

  type Output: Send + 'static;
  type Error: Send + 'static;

  fn serve(&self, input: Input)
    -> impl Future<Output =
      Result<Self::Output,
      Self::Error>> + Send;
}
```

everything is a service

```rust
let http_request_svc = (
    TraceLayer::new_for_http(),
    ConsumeErrLayer::default(),
    ProxyAuthLayer::new(basic!("tom", "clancy")),
    UpgradeLayer::new(
        exec.clone(),
        MethodMatcher::CONNECT,
        service_fn(http_connect_accept),
        ConsumeErrLayer::default().into_layer(Forwarder::ctx(exec)),
    ),
    RemoveResponseHeaderLayer::hop_by_hop(),
    RemoveRequestHeaderLayer::hop_by_hop(),
)
    .into_layer(service_fn(http_plain_proxy));

let http_server = HttpServer::auto(exec.clone()).service(http_request_svc);


let tls_acceptor = TlsAcceptorService::new(tls_service_data, http_server.clone(), true);
let auto_tls_acceptor = TlsPeekRouter::new(tls_acceptor).with_fallback(http_server);

let socks5_acceptor =
    Socks5Acceptor::default().with_authorizer(basic!("john", "secret").into_authorizer());
let auto_socks5_acceptor =
    Socks5PeekRouter::new(socks5_acceptor).with_fallback(auto_tls_acceptor);


let tcp_service = TcpListener::bind("127.0.0.1:62029", exec.clone())
    .await
    .expect("bind http+https+socks5+socks5h proxy to 127.0.0.1:62029");

graceful.spawn_task(tcp_service.serve(auto_socks5_acceptor));
```

http proxy

http

[tls]

[socks5]

tcp

Services all the way down…

```
let auto_socks5_acceptor: Socks5PeekRouter<Socks5Acceptor<Connector<TcpConnector,
StreamForwardService>, (), (), StaticAuthorizer<Basic>>,
TlsPeekRouter<TlsAcceptorService<HttpService<Builder, Trace<ConsumeErr<ProxyAuthService<Basic,
Basic, UpgradeService<RemoveResponseHeader<RemoveRequestHeader<ServiceFn<fn
http_plain_proxy(Request) -> impl Future<Output = Result<Response, Infallible>>, (Request,),
impl Future<Output = Result<Response, Infallible>>, Response, Infallible>>>, Response>>,
Trace>, SharedClassifier<ServerErrorsAsFailures>>>>, HttpService<Builder,
Trace<ConsumeErr<ProxyAuthService<Basic, Basic,
UpgradeService<RemoveResponseHeader<RemoveRequestHeader<ServiceFn<fn http_plain_proxy(Request)
-> impl Future<Output = Result<Response, Infallible>>, (Request,), impl Future<Output =
Result<Response, Infallible>>, Response, Infallible>>>, Response>>, Trace>,
SharedClassifier<ServerErrorsAsFailures>>>>>
```

`async fn serve(TcpStream) -> Result<(), Error>`

stack as trait bounds

```rust
async fn mitm_websocket<S>
(client: &S, req: Request)
→ Response
where
    S: Service<Request,
    Output = Response,
    Error: Into<BoxError>>,
{...
}
```

```rust
pub struct AcmeClient {
    https_client:

  BoxService<Request,
  Response, OpaqueError>,

    ...
}
```

simplicity

& be explicit

rama-tcp/src/server/listener.rs +1 -4 Viewed

```
@@ -9,7 +9,6 @@ use rama_net::socket::Interface;
 9   use rama_net::stream::Socket;                    9   use rama_net::stream::Socket;
10   use rama_net::stream::SocketInfo;                10   use rama_net::stream::SocketInfo;
11   use std::pin::pin;                               11   use std::pin::pin;
12 - use std::sync::Arc;
13   use std::{io, net::SocketAddr};                  12   use std::{io, net::SocketAddr};
14   use tokio::net::TcpListener as TokioTcpListener; 13   use tokio::net::TcpListener as TokioTcpListener;
15                                                    14
@@ -313,10 +312,8 @@ impl TcpListener {
313      /// gracefully if the [`TcpListener`] is  312      /// gracefully if the [`TcpListener`] is
         configured with a graceful [`Executor`].         configured with a graceful [`Executor`].
314      pub async fn serve<S>(self, service: S)  313      pub async fn serve<S>(self, service: S)
315      where                                    314      where
316 -        S: Service<TcpStream>,               315 +        S: Service<TcpStream> + Clone,
317      {                                        316      {
318 -        let service = Arc::new(service);
319 -
320          let guard = self.exec.guard().cloned();  317          let guard = self.exec.guard().cloned();
321          let cancelled_fut = async {              318          let cancelled_fut = async {
322              if let Some(guard) = guard {         319              if let Some(guard) = guard {
```

rama-core/src/inspect/request.rs -36 Viewed

```
@@ -1,36 +0,0 @@
- use crate::Service;
-
- /// A special kind of [`Service`] which has access only to the Request,
- /// but not to the Response.
- ///
- /// Useful in case you want to explicitly
- /// restrict this acccess or because the Response would
- /// anyway not yet be produced at the point this inspector would be layered.
- pub trait RequestInspector<RequestIn>: Send + Sync + 'static {
-     /// The type of error returned by the service.
-     type Error: Send + 'static;
-     type RequestOut: Send + 'static;
-
-     /// Inspect the request, modify it if needed or desired, and return it.
-     fn inspect_request(
-         &self,
-         req: RequestIn,
-     ) -> impl Future<Output = Result<Self::RequestOut, Self::Error>> + Send + '_;
- }
- impl<S, RequestIn, RequestOut> RequestInspector<RequestIn> for S
- where
-     S: Service<RequestIn, Response = RequestOut>,
-     RequestIn: Send + 'static,
-     RequestOut: Send + 'static,
- {
-     type Error = S::Error;
-     type RequestOut = RequestOut;
-
-     fn inspect_request(
-         &self,
-         req: RequestIn,
-     ) -> impl Future<Output = Result<Self::RequestOut, Self::Error>> + Send + '_ {
-         self.serve(req)
-     }
- }
```

```
7    /// A [`Service`] that produces rama services,
8  - /// to serve requests with, be it transport layer requests or application
     layer requests.
9  - pub trait Service<S, Request>: Sized + Send + Sync + 'static {
10 -     /// The type of response returned by the service.
11 -     type Response: Send + 'static;
12
13       /// The type of error returned by the service.
14       type Error: Send + 'static;
15
16 -     /// Serve a response or error for the given request,
17 -     /// using the given context.
18       fn serve(
19           &self,
20 -         ctx: Context<S>,
21 -         req: Request,
22 -     ) -> impl Future<Output = Result<Self::Response, Self::Error>> + Send
     + '_;
23
```

```
9    /// A [`Service`] that produces rama services,
10 + /// to serve given an input, be it transport layer Inputs or application
     layer http requests,
11 + /// or something else entirely.
12 + pub trait Service<Input>: Sized + Send + Sync + 'static {
13 +     /// The type of the output returned by the service.
14 +     type Output: Send + 'static;
15
16       /// The type of error returned by the service.
17       type Error: Send + 'static;
18
19 +     /// Serve an output or an error for the given input
20       fn serve(
21           &self,
22 +         input: Input,
23 +     ) -> impl Future<Output = Result<Self::Output, Self::Error>> + Send +
     '_;
24
```

```rust
#[derive(Clone, Debug)]
pub struct MyState {
    pub db: Arc<Database>,
}

pub(super) fn web_svc(
    state: MyState,
) -> impl Service<Request, Output = Response, Error = Infallible> {
    Router::new_with_state(state)
        .with_get("/", index)
        .with_post("/user/{id}", create_user)
        .with_get("/user{id}", get_user)
}

async fn create_user(
    State(MyState { db }): State<MyState>,
    Path(Params { id }): Path<Params>,
    Json(meta): Json<UserMeta>,
) -> impl IntoResponse {
    db.insert(id, meta).await;
    StatusCode::OK
}
```

Extensions

Any state that is optional, and especially optional state injected by middleware, can be inserted using extensions.

Extensions

```rust
pub trait ExtensionsRef {
    // Required method
    fn extensions(&self) -> &Extensions;
}
```

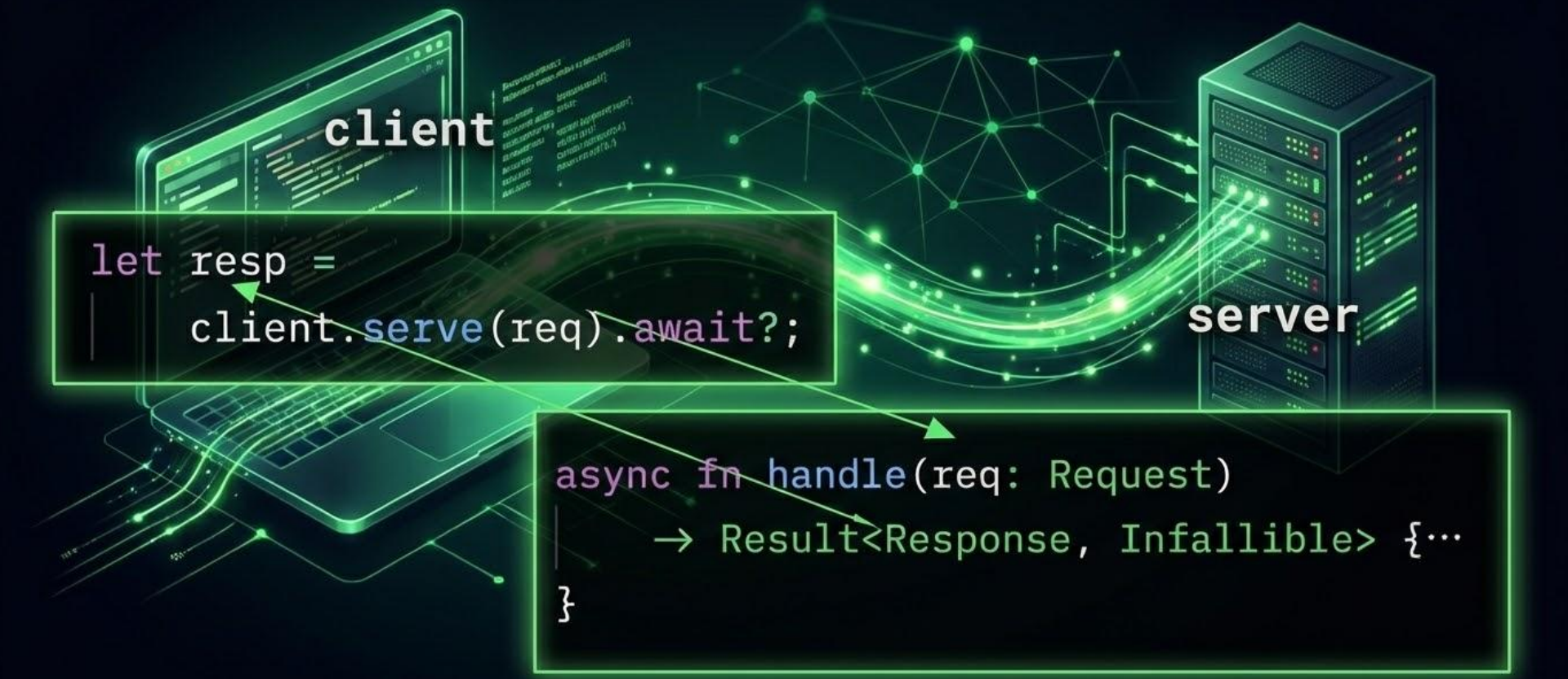
```rust
pub trait ExtensionsMut: ExtensionsRef {
    // Required method
    fn extensions_mut(&mut self) -> &mut Extensions;
}
```

```rust
impl Extensions

pub fn new() -> Extensions

pub fn insert<T>(&mut self, val: T) -> &T
where
    T: Extension + Clone,

pub fn extend(&mut self, extensions: Extensions)

pub fn contains<T>(&self) -> bool
where
    T: Extension + Clone,

pub fn get<T>(&self) -> Option<&T>
where
    T: Extension + Clone,

pub fn get_or_insert<T, F>(&mut self, create_fn: F) -> &T
where
    T: Extension + Clone,
    F: FnOnce() -> T,

pub fn first<T>(&self) -> Option<&T>
where
    T: Extension + Clone,

pub fn iter<T>(&self) -> impl Iterator<Item = &T>
where
    T: Extension + Clone,
```

```
async fn serve(input)
-> Result<Output, Error>
```


rama
proxy
Server + Client

async fn serve(Request) -> Result<Response, Error>



“Extension” traits

```
let client: Trace<Decompression<AddAuthorization<Retry
<ManagedPolicy<ExponentialBackoff<fn default<HasherRng>() →
HasherRng>>, EasyHttpWebClient<RetryBody,
EstablishedClientConnection<HttpClientService<RetryBody>, Request
<RetryBody>>, ()>>>>, SharedClassifier<ServerErrorsAsFailures>>
```

```
let info: Info = client
    .get(format!("http://{ADDRESS}/info"))
    .header("x-magic", "42")
    .typed_header(Accept::json())
    .send()
    .await?
    .try_into_json()
    .await?;
```

```
impl<S, Body> HttpClientExt for S
where
    S: Service<Request, Output = Response<Body>, Error: Into<BoxError>>,
{
    type ExecuteResponse = Response<Body>;
    type ExecuteError = S::Error;

    fn get(&self, url: impl IntoUrl) → RequestBuilder<'_, Self, Self::ExecuteResponse> {
        self.request(Method::GET, url)
    }

    fn request(
        &self,
        method: Method,
        url: impl IntoUrl,
    ) → RequestBuilder<'_, Self, Self::ExecuteResponse> {
```

Trait HttpClientExt

Is it a server?
Is it a client?

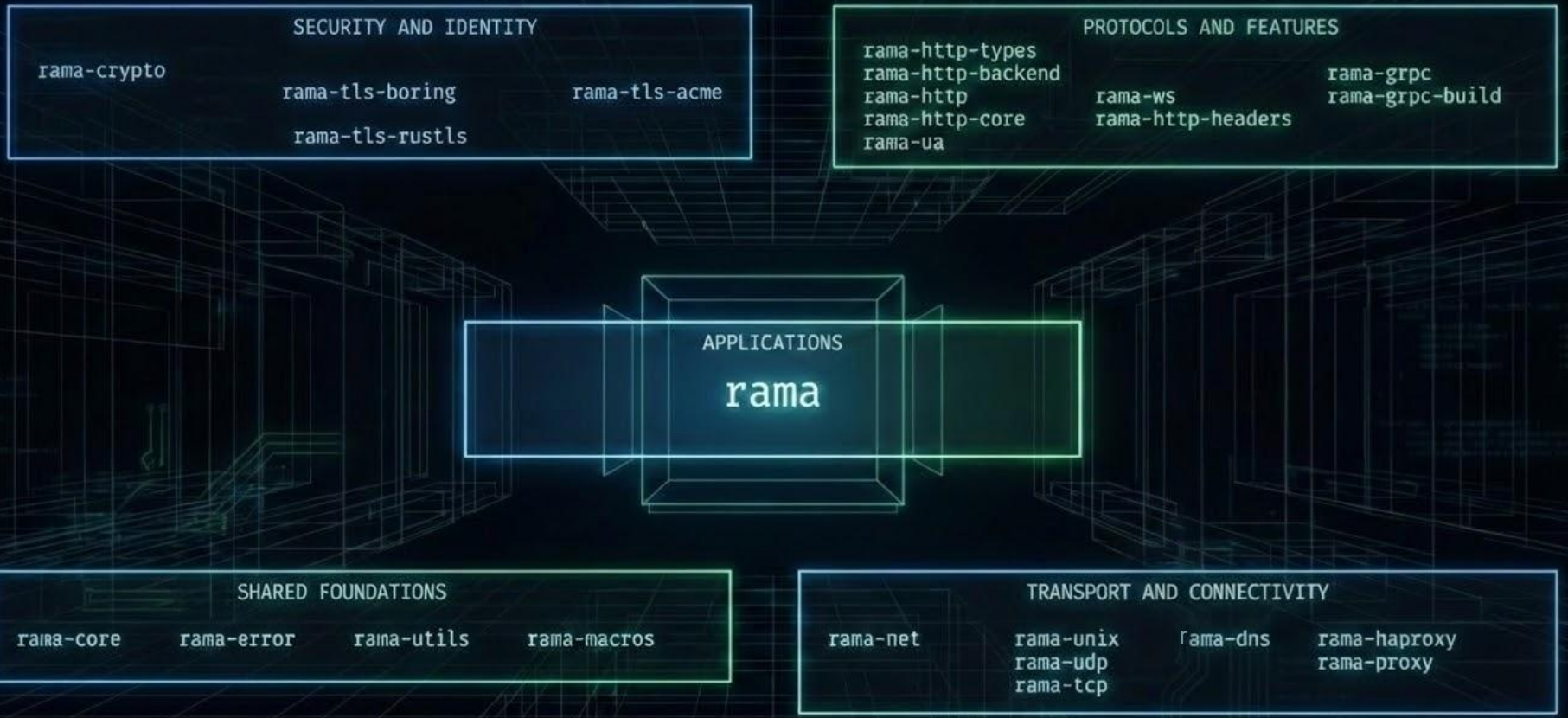
It's a service!

```rust
let app = (
    MapResponseBodyLayer::new(Body::new),
    TraceLayer::new_for_http(),
)
    .into_layer(Arc::new(
        Router::new()
            .with_get("/", Html(INDEX_CONTENT))
            .with_get("/api/events", api_events_endpoint),
    ));

let resp: Response
size = 144 (0x90), align = 0x8, needs Drop
let resp = app.get("/").send().await?;
```

```
async fn serve(Request) -> Result<Response, Error>
```

Rama

overview and what’s next


SECURITY AND IDENTITY
rama-crypto
rama-tls-boring
rama-tls-acme
rama-tls-rustls
PROTOCOLS AND FEATURES
rama-http-types
rama-http-backend
rama-http
rama-http-core
rama-ua
rama-ws
rama-http-headers
rama-grpc
rama-grpc-build
APPLICATIONS
rama
SHARED FOUNDATIONS
rama-core
rama-error
rama-utils
rama-macros
TRANSPORT AND CONNECTIVITY
rama-net
rama-unix
rama-udp
rama-tcp
rama-dns
rama-haproxy
rama-proxy

Rama

ramaproxy.org/book/

Intro to rama: Zen of services, middleware (layers), tips, dynamic dispatch, telemetry, user agents, pattern, ...

Intro to proxies: reverse, TLS termination, HTTP(S), SOCKS5(H), SNI, MITM, Distortion, HaProxy and more...

And more: FAQ, deploy, rama CLI, clients, servers, transport protocols....

Later this year… Rama Tutorials!

Errors

rama::error

> The greatest mistake is to imagine that we never err.
>
> — *Thomas Carlyle.*

Trait Definition

```rust
pub trait ErrorContext: SealedErrorContext {
    type Context;

    // Required methods
    fn context<M>(self, context: M) -> Self::Context
       where M: Display + Send + Sync + 'static;

    fn with_context<C, F>(self, context: F) -> Self::Context
       where C: Display + Send + Sync + 'static,
             F: FnOnce() -> C;
}
```

Usage Example

```rust
let result = "hello".parse::<i32>().context("parse integer");
```

Rama

Public Services

Sponsored by fly.io

http(s)://http-test.ramaproxy.org

http test resources (e.g. compression, streaming, SSE, …)

http(s)://echo.ramaproxy.org

Echo back (in json) input from client tls, tcp and request (http)

http(s)://fp.ramaproxy.org

Fingerprint service, also used to gather **UA profiles** for rama-ua

https://ipv[46].ramaproxy.org

Return your fingerprint in **html/txt/json** format

Rama examples

Fully tested, production grade examples covering the entire networking stack

- HTTP services, routing, files, forms, streaming, HAR, health checks
- Server Sent Events, ndjson streams, real time data flows
- WebSocket servers, compression, TLS, HTTP/2, Autobahn tests
- gRPC services with build tooling
- HTTP, HTTPS and SOCKS5 proxies, including MITM and combos
- TLS, ACME, Rustls, BoringSSL, SNI routing, mutual TLS
- TCP, UDP and Unix sockets, zero downtime restarts
- HaProxy support and Tower integration

https://github.com/plabayo/rama/tree/main/examples

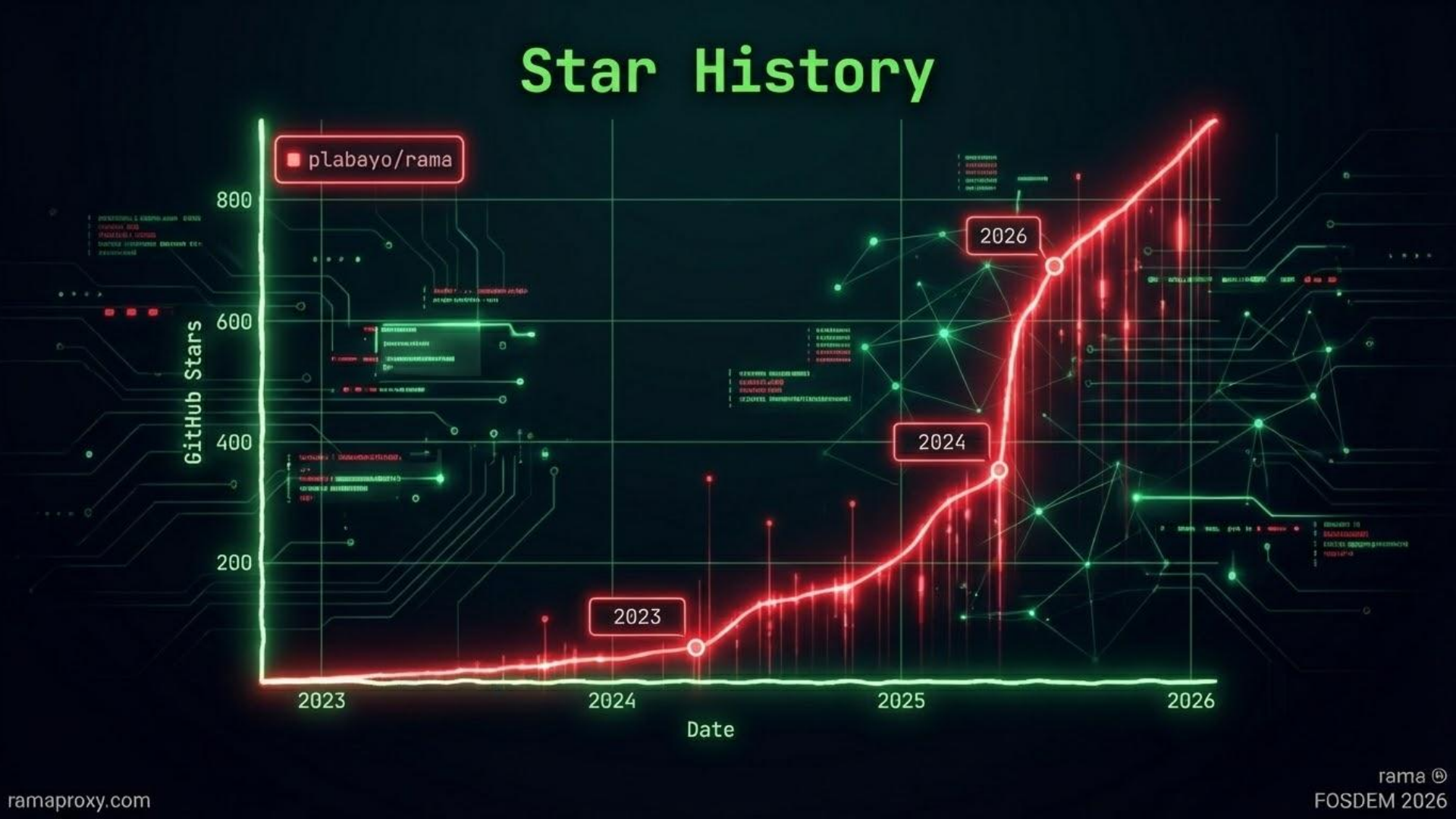
Rama Releases

Soon™ … rama v0.3, what's left to do?

- `extension` Rework
- `rama-net::uri` (and usage of in other parts of rama)
- Removal of OpaqueError (in public)
- Some more minor improvements…

Starting from 0.3 (~ February), we will do regular minor/major releases

Star History



Open source • Public development • Actively maintained • Production partnerships available

Build network services with a Rust framework you control

✔ A coherent framework for building network services

✔ One foundation for proxies, gateways, servers and clients

✔ Open source maintained, with production support available

View partnership plans

Explore the open source framework

Rama Community

Sharing is Caring

Join a supportive network of developers. Share knowledge, contribute code, and build resilient infrastructure together.

ラマ | rama

a modular service framework to move
and transform packets

github.com/plabayo/rama