Nim and C libraries

Agenda

- Nim introduction
- Why C?
- Use C libraries in Nim
- Expose Nim as a C library
- nwaku (Nim Waku) everywhere

Nim introduction

- Statically typed language
- Python-like syntax
- It can compile to C, C++ and JavaScript
- Flexible memory management (GC, manual management)

Why C?

- Universal
  - Every OS and CPU understands C
  - C is the de facto interop language
- Libraries for +40 years
- Direct hardware control, e.g. Linux kernel

Use C libraries in Nim

- Easy

-
  ```
  proc pqinitOpenSSL*(do_ssl: int32,
  do_crypto: int32) {.cdecl, dynlib:
  dllName,
     importc: "PQinitOpenSSL".}
  ```
- Then, library can be dynamic or statically linked
- F.e, in Waku, used to integrate libpq; mysql; zerokit (Rust.)

Export Nim as a C library

- Not straightforward
- A secondary thread is needed if Nim project uses GC
- Additional wrapper is needed on every target language

Nwaku everywhere

- nwaku is a Waku client implemented in Nim
- nwaku can run in:
  - Golang
  - Rust
  - Python
  - C/C++
  - Android
  - …

Nwaku everywhere

- Callbacks approach is used
- Asynchronous library
- libwaku.nim ⇒ libwaku.so/.dll or libwaku.a/.lib
- libwaku.h is also exported

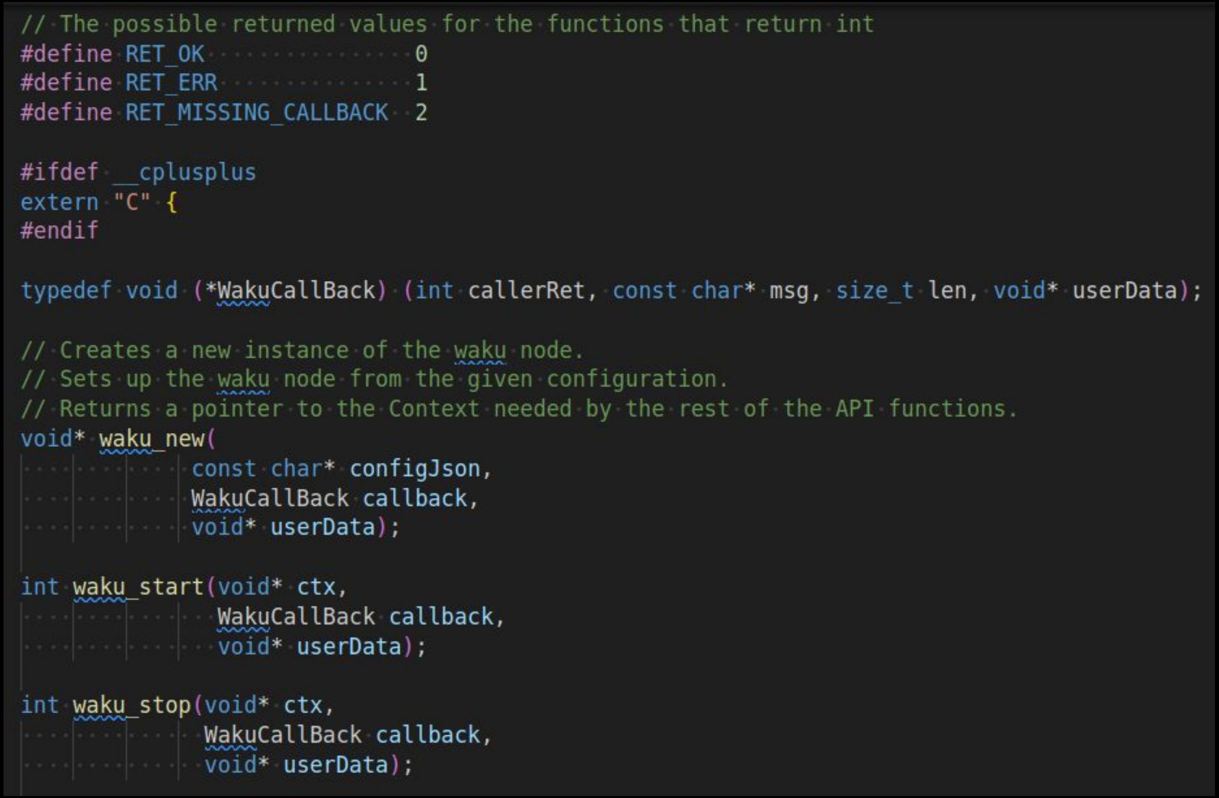
Nwaku everywhere

- In libwaku.nim we don’t use GC’ed types (closure, ref object, string, or seq[T])
- Memory is managed manually in libwaku: main thread creates objects and the waku thread consumes it and frees memory

Nim & C

Nwaku everywhere

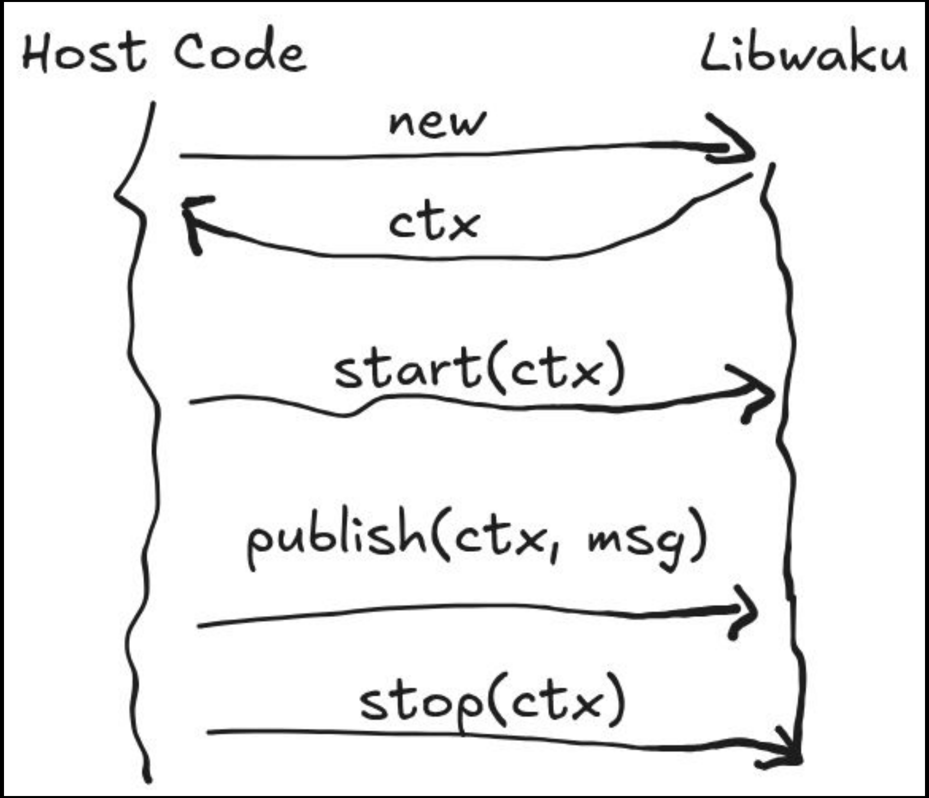
libwaku.h



```c
// The possible returned values for the functions that return int
#define RET_OK                0
#define RET_ERR               1
#define RET_MISSING_CALLBACK  2

#ifdef __cplusplus
extern "C" {
#endif

typedef void (*WakuCallBack) (int callerRet, const char* msg, size_t len, void* userData);

// Creates a new instance of the waku node.
// Sets up the waku node from the given configuration.
// Returns a pointer to the Context needed by the rest of the API functions.
void* waku_new(
             const char* configJson,
             WakuCallBack callback,
             void* userData);

int waku_start(void* ctx,
               WakuCallBack callback,
               void* userData);

int waku_stop(void* ctx,
              WakuCallBack callback,
              void* userData);
```

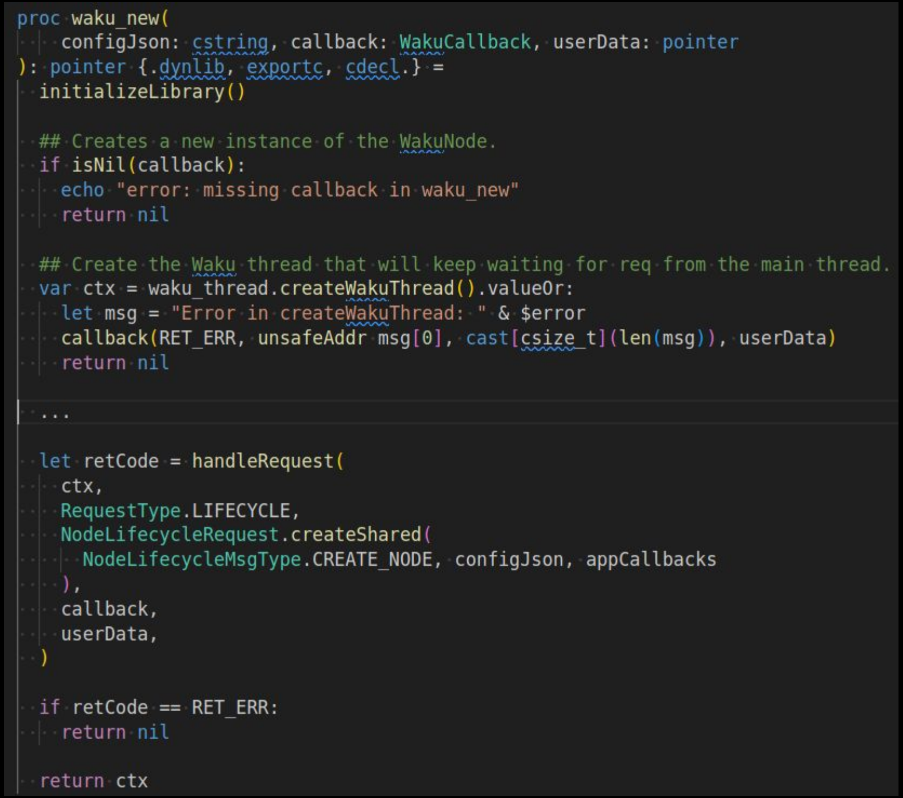
Nwaku everywhere

The waku thread attends API requests coming from the host code



Nwaku everywhere

waku_new creates the waku thread and returns the ctx needed by the host code.

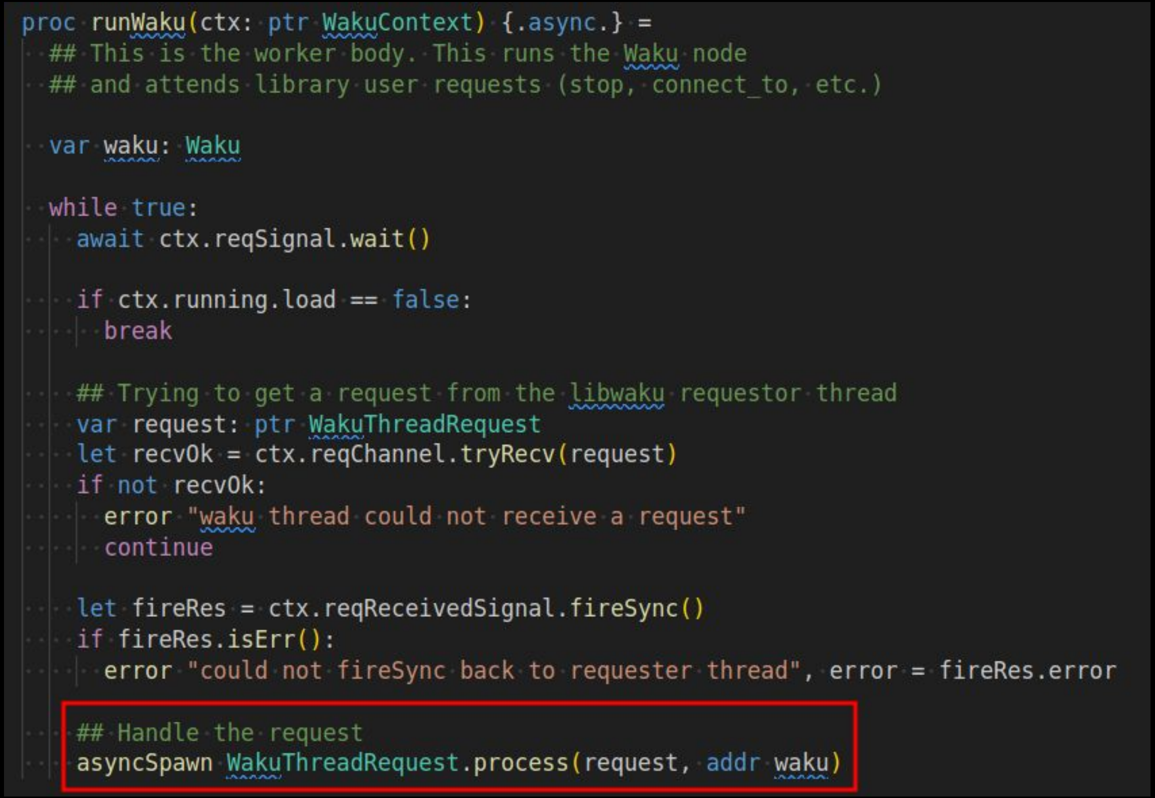


```nim
proc waku_new(
    configJson: cstring, callback: WakuCallback, userData: pointer
): pointer {.dynlib, exportc, cdecl.} =
  initializeLibrary()

  ## Creates a new instance of the WakuNode.
  if isNil(callback):
    echo "error: missing callback in waku_new"
    return nil

  ## Create the Waku thread that will keep waiting for req from the main thread.
  var ctx = waku_thread.createWakuThread().valueOr:
    let msg = "Error in createWakuThread: " & $error
    callback(RET_ERR, unsafeAddr msg[0], cast[csize_t](len(msg)), userData)
    return nil

  ...

  let retCode = handleRequest(
    ctx,
    RequestType.LIFECYCLE,
    NodeLifecycleRequest.createShared(
      NodeLifecycleMsgType.CREATE_NODE, configJson, appCallbacks
    ),
    callback,
    userData,
  )

  if retCode == RET_ERR:
    return nil

  return ctx
```

Nim & C

Nwaku everywhere

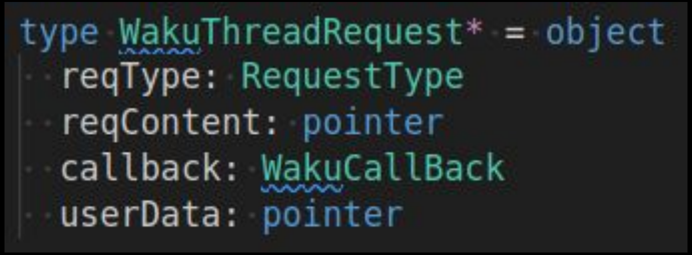
The API requests are handled asynchronously



```nim
proc runWaku(ctx: ptr WakuContext) {.async.} =
  ## This is the worker body. This runs the Waku node
  ## and attends library user requests (stop, connect_to, etc.)

  var waku: Waku

  while true:
    await ctx.reqSignal.wait()

    if ctx.running.load == false:
      break

    ## Trying to get a request from the libwaku requestor thread
    var request: ptr WakuThreadRequest
    let recvOk = ctx.reqChannel.tryRecv(request)
    if not recvOk:
      error "waku thread could not receive a request"
      continue

    let fireRes = ctx.reqReceivedSignal.fireSync()
    if fireRes.isErr():
      error "could not fireSync back to requester thread", error = fireRes.error

    ## Handle the request
    asyncSpawn WakuThreadRequest.process(request, addr waku)
```

Nwaku everywhere

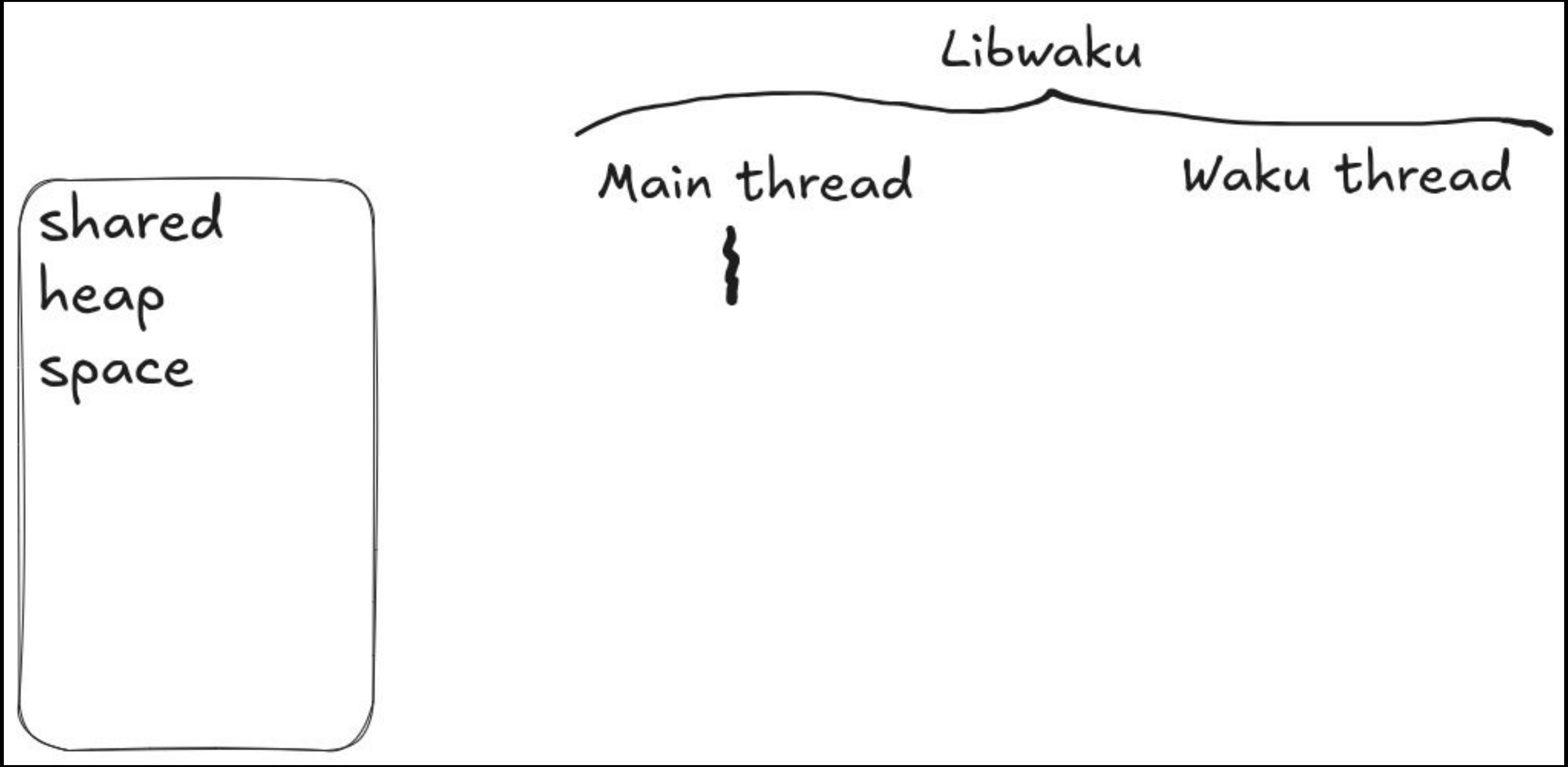
1. The main thread creates a request object in the shared heap
2. Main thread sends the request's ptr to the waku thread
3. The waku thread attends the request and deallocates the inter-thread shared memory



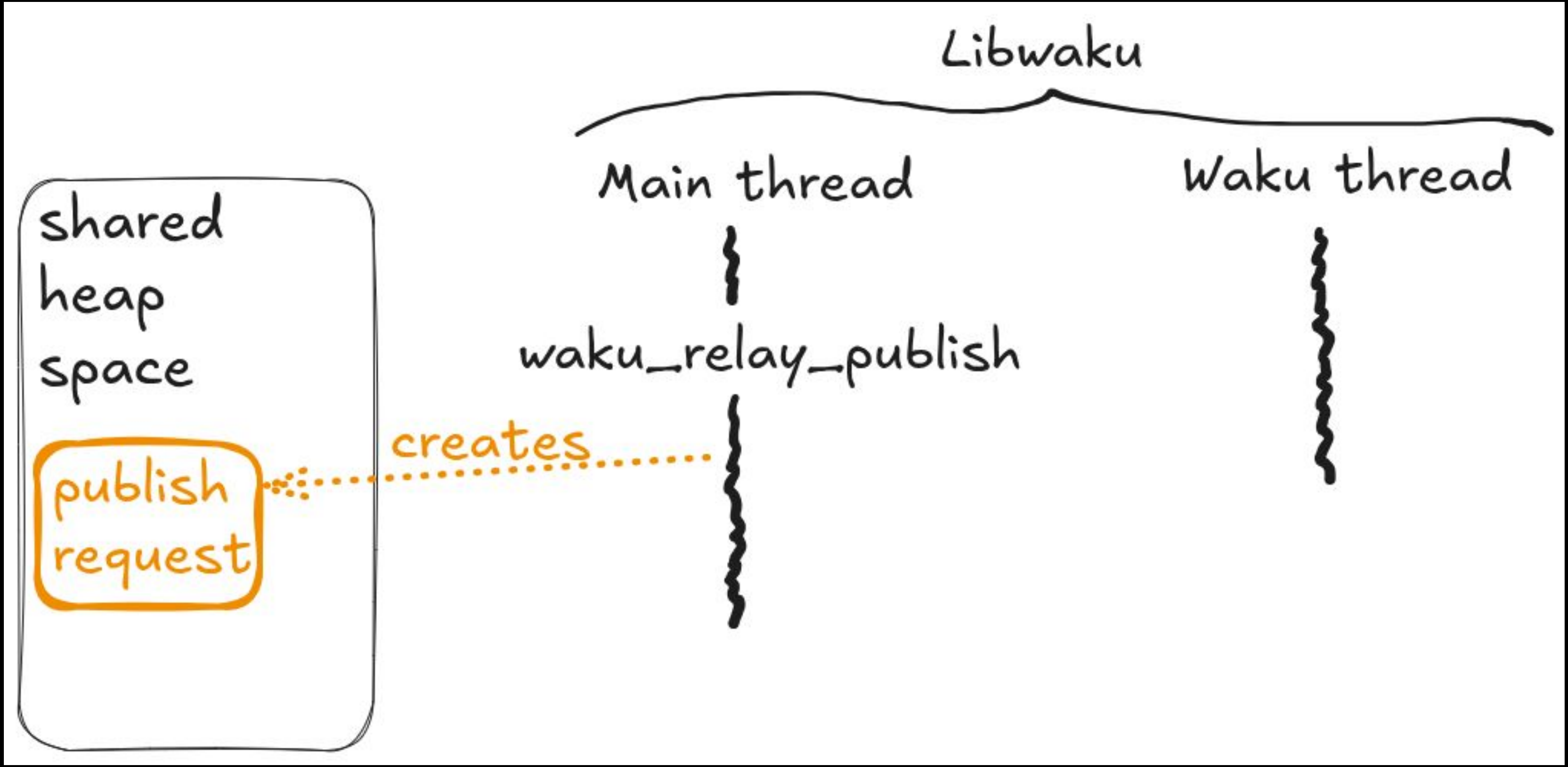
```
type WakuThreadRequest* = object
  reqType: RequestType
  reqContent: pointer
  callback: WakuCallBack
  userData: pointer
```

Nim & C

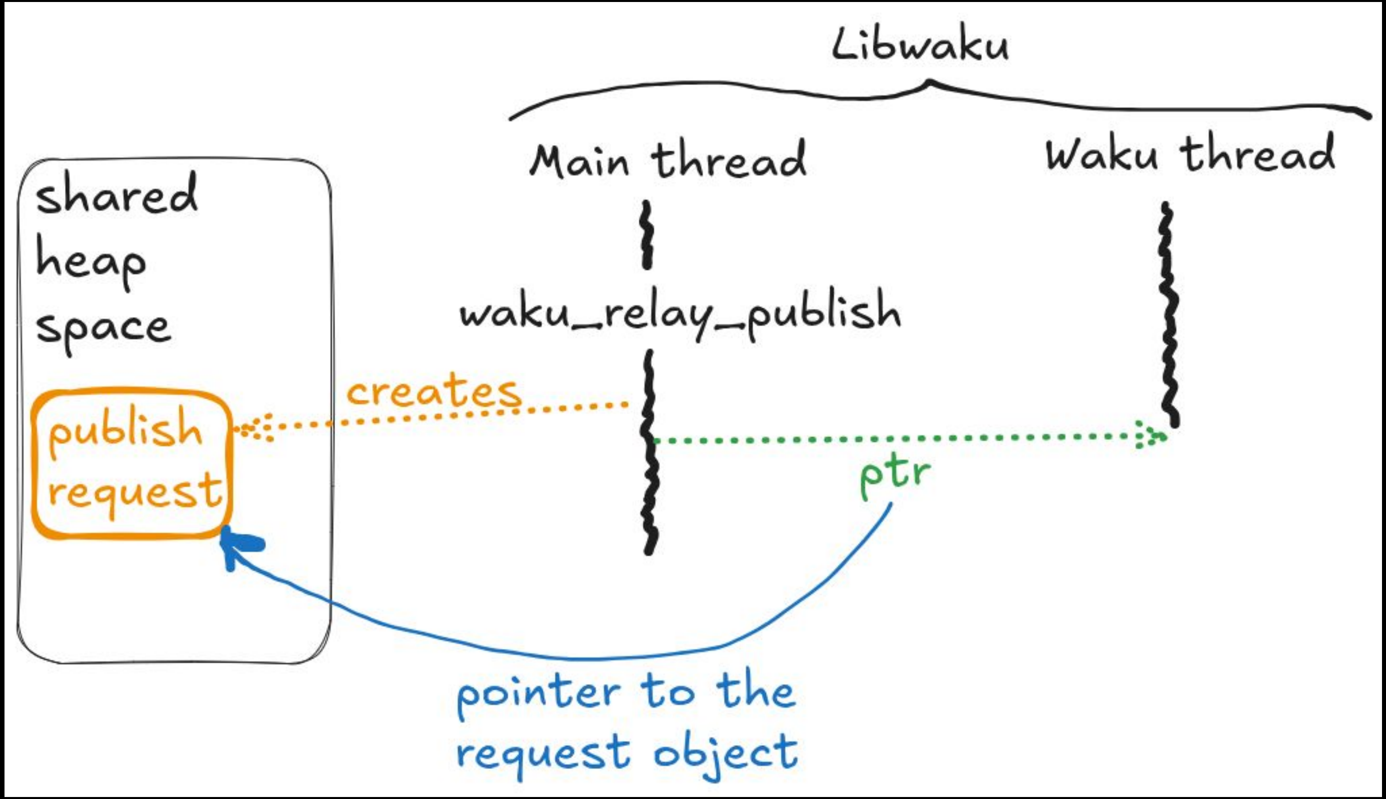
Nwaku everywhere



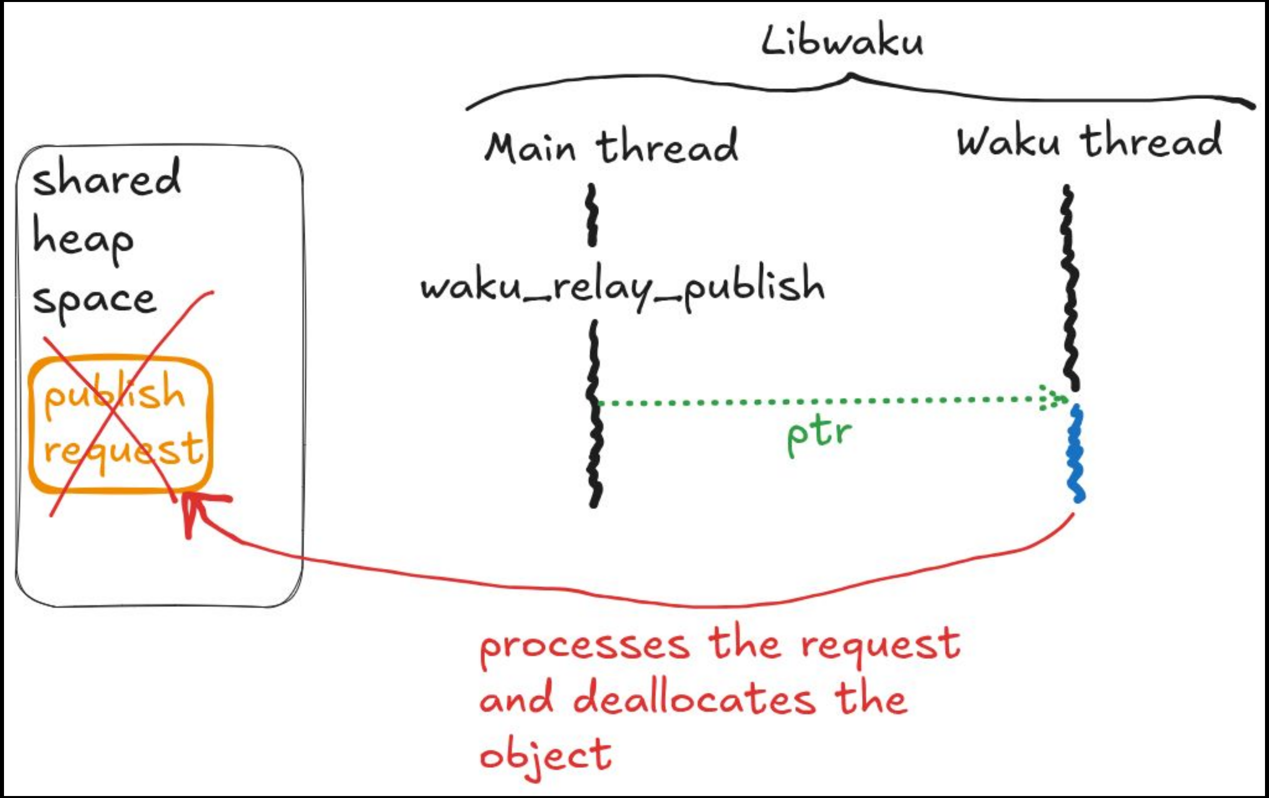
Nwaku everywhere



Nwaku everywhere



Nwaku everywhere



Nwaku everywhere

Golang https://github.com/waku-org/waku-go-bindings

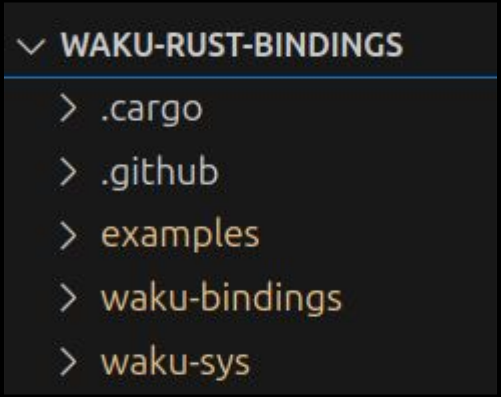
- cgo is used to expose libwaku to Golang
- …/waku/nwaku.go is the most important file
- go get -u github.com/waku-org/waku-go-bindings

Nim & C

Nwaku everywhere

Rust

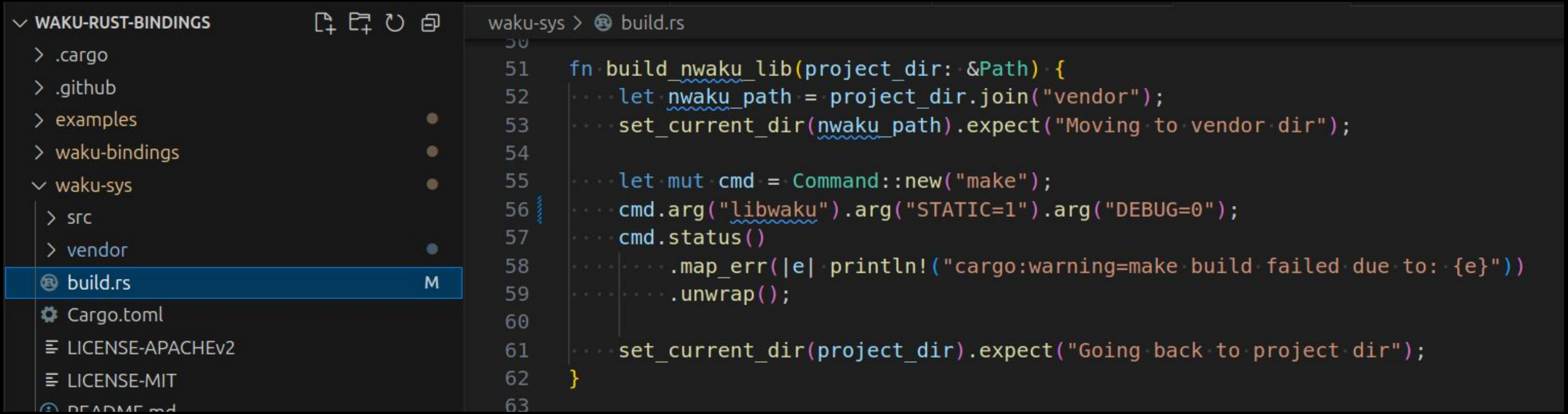
https://github.com/waku-org/waku-rust-bindings



Nim & C

Nwaku everywhere

Rust https://github.com/waku-org/waku-rust-bindings


WAKU-RUST-BINDINGS
- .cargo
- .github
- examples
- waku-bindings
- waku-sys
  - src
  - vendor
  - build.rs M
  - Cargo.toml
  - LICENSE-APACHEv2
  - LICENSE-MIT

waku-sys > build.rs

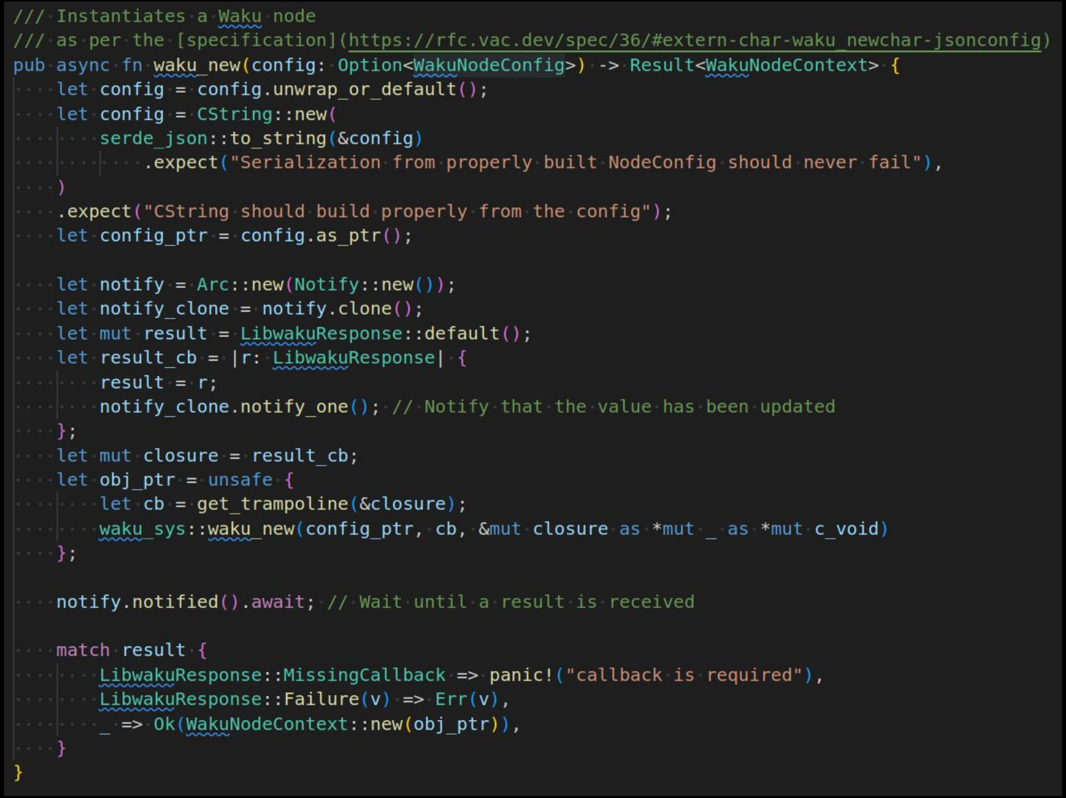
```rust
fn build_nwaku_lib(project_dir: &Path) {
    let nwaku_path = project_dir.join("vendor");
    set_current_dir(nwaku_path).expect("Moving to vendor dir");

    let mut cmd = Command::new("make");
    cmd.arg("libwaku").arg("STATIC=1").arg("DEBUG=0");
    cmd.status()
        .map_err(|e| println!("cargo:warning=make build failed due to: {e}"))
        .unwrap();

    set_current_dir(project_dir).expect("Going back to project dir");
}
```

Nim & C

Nwaku everywhere

Rust

https://github.com/waku-org/waku-rust-bindings

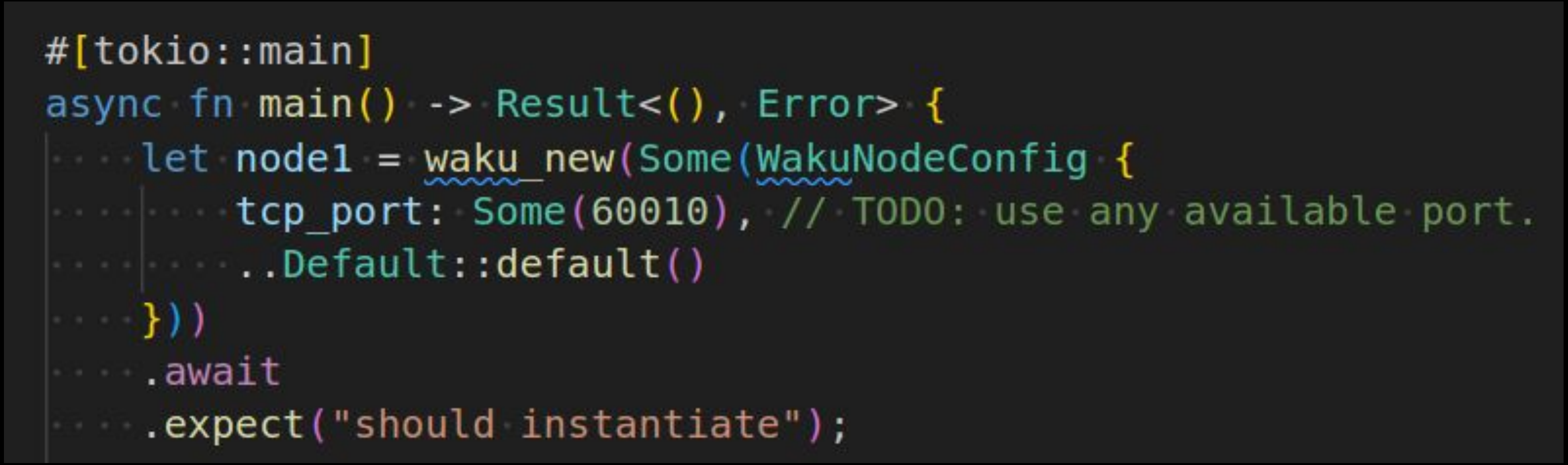


```rust
/// Instantiates a Waku node
/// as per the [specification](https://rfc.vac.dev/spec/36/#extern-char-waku_newchar-jsonconfig)
pub async fn waku_new(config: Option<WakuNodeConfig>) -> Result<WakuNodeContext> {
    let config = config.unwrap_or_default();
    let config = CString::new(
        serde_json::to_string(&config)
            .expect("Serialization from properly built NodeConfig should never fail"),
    )
    .expect("CString should build properly from the config");
    let config_ptr = config.as_ptr();

    let notify = Arc::new(Notify::new());
    let notify_clone = notify.clone();
    let mut result = LibwakuResponse::default();
    let result_cb = |r: LibwakuResponse| {
        result = r;
        notify_clone.notify_one(); // Notify that the value has been updated
    };
    let mut closure = result_cb;
    let obj_ptr = unsafe {
        let cb = get_trampoline(&closure);
        waku_sys::waku_new(config_ptr, cb, &mut closure as *mut _ as *mut c_void)
    };

    notify.notified().await; // Wait until a result is received

    match result {
        LibwakuResponse::MissingCallback => panic!("callback is required"),
        LibwakuResponse::Failure(v) => Err(v),
        _ => Ok(WakuNodeContext::new(obj_ptr)),
    }
}
```

Nim & C

Nwaku everywhere

Rust https://github.com/waku-org/waku-rust-bindings

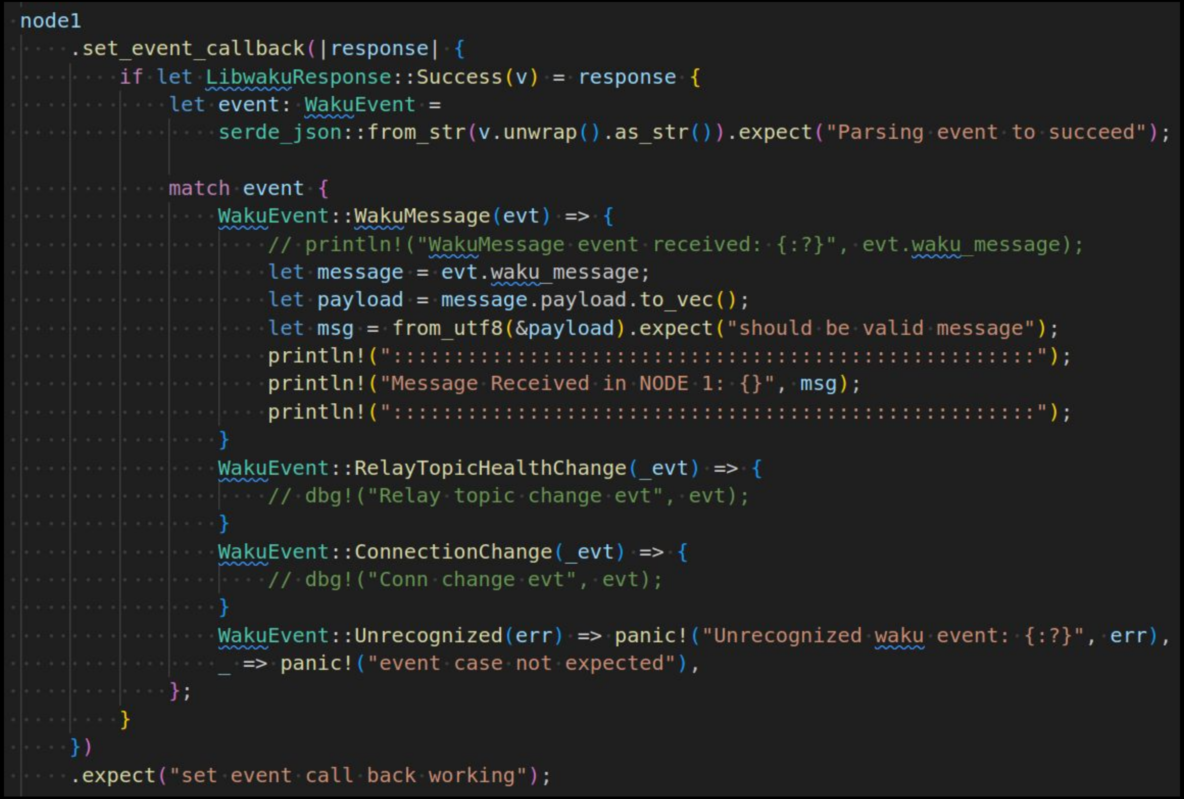


```
#[tokio::main]
async fn main() -> Result<(), Error> {
    let node1 = waku_new(Some(WakuNodeConfig {
        tcp_port: Some(60010), // TODO: use any available port.
        ..Default::default()
    }))
    .await
    .expect("should instantiate");
```

Nwaku everywhere

Rust

https://github.com/waku-org/waku-rust-bindings



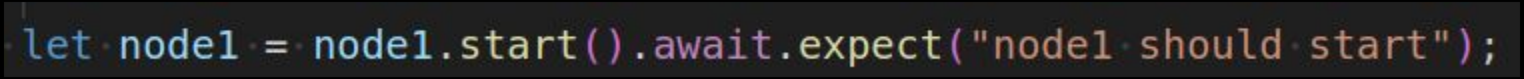
```
node1
    .set_event_callback(|response| {
        if let LibwakuResponse::Success(v) = response {
            let event: WakuEvent =
                serde_json::from_str(v.unwrap().as_str()).expect("Parsing event to succeed");

            match event {
                WakuEvent::WakuMessage(evt) => {
                    // println!("WakuMessage event received: {:?}", evt.waku_message);
                    let message = evt.waku_message;
                    let payload = message.payload.to_vec();
                    let msg = from_utf8(&payload).expect("should be valid message");
                    println!("::::::::::::::::::::::::::::::::::::::::::::::::::::");
                    println!("Message Received in NODE 1: {}", msg);
                    println!("::::::::::::::::::::::::::::::::::::::::::::::::::::");
                }
                WakuEvent::RelayTopicHealthChange(_evt) => {
                    // dbg!("Relay topic change evt", evt);
                }
                WakuEvent::ConnectionChange(_evt) => {
                    // dbg!("Conn change evt", evt);
                }
                WakuEvent::Unrecognized(err) => panic!("Unrecognized waku event: {:?}", err),
                _ => panic!("event case not expected"),
            };
        }
    })
    .expect("set event call back working");
```

Nwaku everywhere

Nim & C

Rust

https://github.com/waku-org/waku-rust-bindings



```
let node1 = node1.start().await.expect("node1 should start");
```

Waku

Q & A