# Networking with LNP

universal networking protocol for bitcoin world
— and much beyond!

**LNP/BP Standards Association**
Prepared & supervised by **Dr Maxim Orlovsky**, **Pandora Core AG**

Created with support from **Bitfinex** and **Fulgur Ventures**

# The need for P2P & RPC solutions for Bitcoin ecosystem

## Protocols

- DLCs: networking part
- LN extensions: channel factories etc
- RGB: P2P, RPC, Bifrost…
- Watchtowers (BOLT-13)
- Storm, prometheus & much more will follow

## Products

- Better RPC for Bitcoin & LN nodes (JSON-RPC is really outdated)
- Microservice architectures (c-lightning a first example, but more will follow)

# What networking is made of?

- Encoding protocol
- Transport protocols (framing, encryption & session management)
- Procedure invocation standard (P2P, RPC ...)

# API Types

- **P2P**: peer-to-peer
  - peers, i.e. equal roles
  - sends message, no response
- **RPC**: remote procedure call or “client-server”
  - asymmetric roles (client & server)
  - client sends request to server and waits for reply
- **REST** and other RPC alternatives (**GraphAPI**): the same pattern as RPC; used in Web apps only, can be easily built with the same tools as RPC
- **SUB**: publication-subscription or PUB/SUB
  - asymmetric roles (publisher & subscriber)
  - publisher provides async event notifications to (potentially) multiple subscribers
  - subscriber does not send data to publisher

More information: https://github.com/LNP-BP/LNPBPs/issues/21

# Encoding standards

| | Transport | Languages | Code generation | Speed | Security | Interoperability | Community support |
|---|---|---|---|---|---|---|---|
| **JSON / XML** | Any | All | unvalidated key-values | low | low | perfect | perfect |
| **Strict / consensus** | Most | Rust | ? | high | high | bitcoin | extra low |
| **BOLT-1+9 (Message)** | Most | Rust, C, Go, Scala * | schema-validated key-values | intermediate | moderate | lightning | niche |
| **Avaro** | Only Avaro transport | Most | schema-validated key-values | high | moderate | hadoop | big data community |
| **Thrift binary** | Only Thrift transport | Most | generated native code | high | moderate | poor | moderate |
| **Thrift binary compact** | Only Thrift transport | Most | generated native code | high | moderate | poor | moderate |
| **Protobufs** | Most | Most | generated native code | high | moderate | high | good |

# Transport framing protocols

| | Connection | Security | Languages | Firewall performance |
|---|---|---|---|---|
| **ZMQ Framing** | POSIX Sockets, TCP, Inproc | None | Most | Bad |
| **Apache Thrift Framed** | HTTP, TCP, File, Inproc | None | Most | Moderate |
| **BOLT-8** | TCP, POSIX Sockets | Decentralized | Rust, C, Go, Scala * | Bad (good with Tor) |
| **gRPC Cronet** | TCP, POSIX Sockets | None | Most | Moderate |
| **HTTP** | TCP | TLS | Most | Good |
| **WebSocket** | TCP | TLS | Most | Moderate |
| **Raw TCP** | IP | TLS | Most | Moderate |

# Remote procedure calls & REST

| | Transport | Encoding | Category | Languages |
|---|---|---|---|---|
| **Apache Thrift** | HTTP, TCP, Inproc | JSON, Thrift | RPC | Most |
| **Apache Avaro** | HTTP, TCP, Inproc | Avaro | RPC | Most |
| **gRPC** | HTTP, TCP, Inproc | Protobuf | RPC | Most |
| **JSON-RPC** | HTTP, TCP | JSON | RPC | Most |
| **OpenAPI** | HTTP | JSON | REST | JS+ |
| **SOAP/WSDL** | HTTP | XML | RPC | Most |
| **WAMP** | WebSockets | JSON | RPC | JS+ |
| **XML-RPC** | HTTP | XML | RPC | Most |
| **ZMQs** | ZMQ Framing | Any | RPC | Most |
| **BOLT-1 (RPC)** | TCP, BOLT-8 | BOLT-1 | RPC | Rust, C, Go, Scala * |

# Requirements

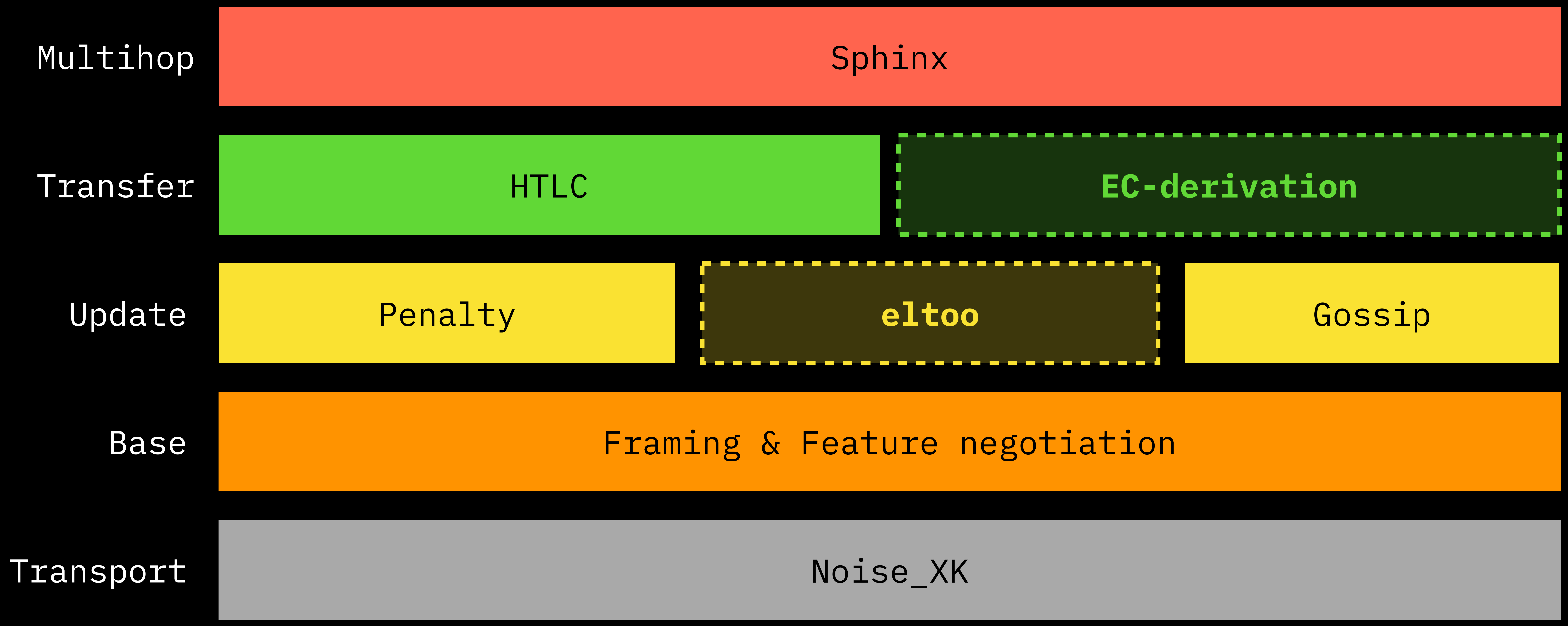
- SSL + DNS -> Tor-like id’s
- Routed
- End-to-end encrypted. Always.
- Native work with hashes, public keys etc
- Already have adoption
- Suited for both P2P & RPC
- Must work over Tor
- Works with ZMQ (optionally)
- Works over Websockets

- ~~OpenAPI~~
- ~~Thrift~~
- ~~Protobufs~~
- ~~Avaro~~
- ~~WAMP, crossbar.io~~

- **LN P2P!**

# Lightning Network Architecture

after Christian Decker



Multihop: Sphinx

Transfer: HTLC | **EC-derivation**

Update: Penalty | **eltoo** | Gossip

Base: Framing & Feature negotiation

Transport: Noise_XK

# Presenting LNP:

- We took LN P2P protocols (BOLT-8, BOLT-1, BOLT-9)
- …dissected into layers
- …added support for Websockets & ZMQ
- …added support for RPC & Pub/Sub APIs
- …added encoding enhancements

# LNP: universal networking protocol for bitcoin world

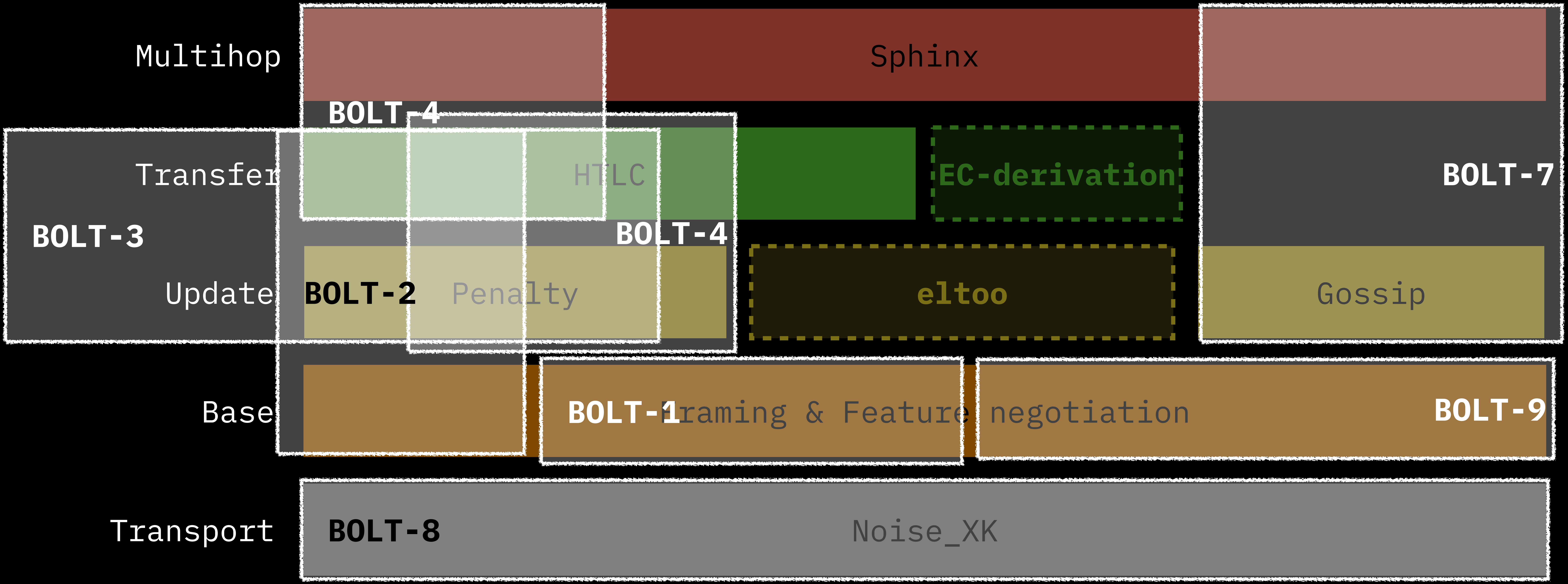
## Decentralized & encrypted

- **No SSL; no PKI**, meaning
  - no centrally-issued certificates & authorities
  - no dependency on DNSes, that can be censored
  - no dependency on CAs, that can be censored
- **Tor-like node ids** and **onion-routing**
- Complete **end-to-end encryption** for all data
- Uses
  - native bitcoin consensus encoding (where defined)
  - LN encoding (data types from BOLT-1, 2, 4, 7)
  - LNP/BP strict encoding (LNPBP-7) used by RGB
- Can pass firewalls (with Tor and UDP hole punching)

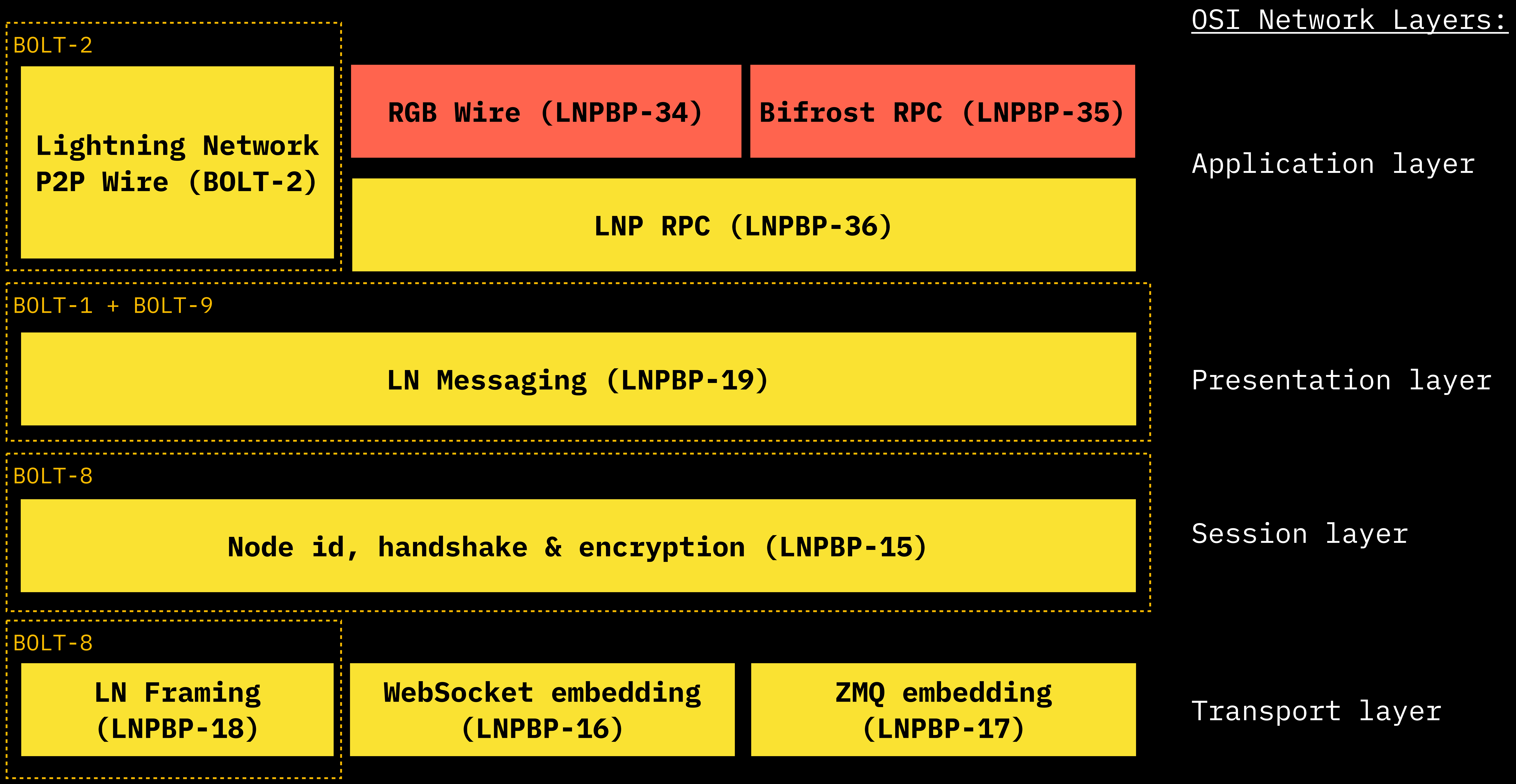
## Interoperable

- Lightning native citizen: already used by LN
- Natively works with Tor and raw TCP sockets
- Now extended to work over
  - UDP & UDP hole punching
  - WebSockets
  - ZMQ for in-process, inter-process and network comms.
  - May work with MTCP, QUICK
- Single RPC protocol standard for all LNP/BP apps
  - Microservice architectures (used by LNP, BP and RGB nodes internally)
  - Peer wire protocols (LN wire protocol, RGB wire protocol)
  - Client-server protocols (like cli tools): replacement for JSON-RPC

# Real BOLT Specifications



# Lightning Network Protocols (LNP) suite



OSI Network Layers:

BOLT-2

Lightning Network P2P Wire (BOLT-2)

RGB Wire (LNPBP-34)

Bifrost RPC (LNPBP-35)

LNP RPC (LNPBP-36)

Application layer

BOLT-1 + BOLT-9

LN Messaging (LNPBP-19)

Presentation layer

BOLT-8

Node id, handshake & encryption (LNPBP-15)

Session layer

BOLT-8

LN Framing (LNPBP-18)

WebSocket embedding (LNPBP-16)

ZMQ embedding (LNPBP-17)

Transport layer

# Lightning Network Protocols:
# LN wire protocol layers dissected

- **Session layer:** identification & encryption
  - Defined in **LNPBP-15**
  - Noise_XK based (first half of **BOLT-8**)
  - manage decentralized node ID:
    (Tor-like identity with Secp256k1 keys)
  - set up session-level encryption
  - do key rotation
- **Presentation layer:** identification & encryption
  - Message structure:
    - type (command)
    - payload parsing
    - TLVs
  - Defined in LNPBP-18
  - In fact, **BOLT-1** + **BOLT-9**
    + some additional recommendations

- **Transport layer:** framing protocols
  - LN native framing over TCP/IP or TCP/Tor (**BOLT-8** second half)
  - Support for ZMQ Sockets framing protocol in multiple variants:
    - P2P (PUSH/PULL ZMQ)
    - RPC (REQ/REP ZMQ)
    - Pub/Sub
  - ... over multiple connection layers:
    - Inproc & IPC (unencrypted)
    - TCP (encrypted & unencrypted)
    - UDP (potentially, important for Mesh & Satellite networks)
  - Support for WebSockets protocol
  - Support for SMTP protocol?
    (Christian Decker proposal)

# Rules for data serialisation

- Do not compress the data
- Use deterministically-defined value length
  (Bitcoin/LN VarInt are bad practices)
- Define both lower and upper bounds for each type validity:
  - ranges for the number of occurrences
  - ranges for possible value (or length in case of strings)
- No pointers/offsets/shifts, no linked lists

# LNP API (with C.Decker): LNPBP-36, 38, 39

## Interface description

- Another IDL standard?! — No!
- Already works in c-lightning
- You can describe interface in:
  - YAML
  - TOML
  - JSON
  - CSV-based custom c-lightning format
  - Special language for LNP API (LIDL)
  - Binary form
    (for network transfers & commitments)
- Can be provided in init message TLV extensions
- Will be defined in **LNPBP-19**

## Toolset

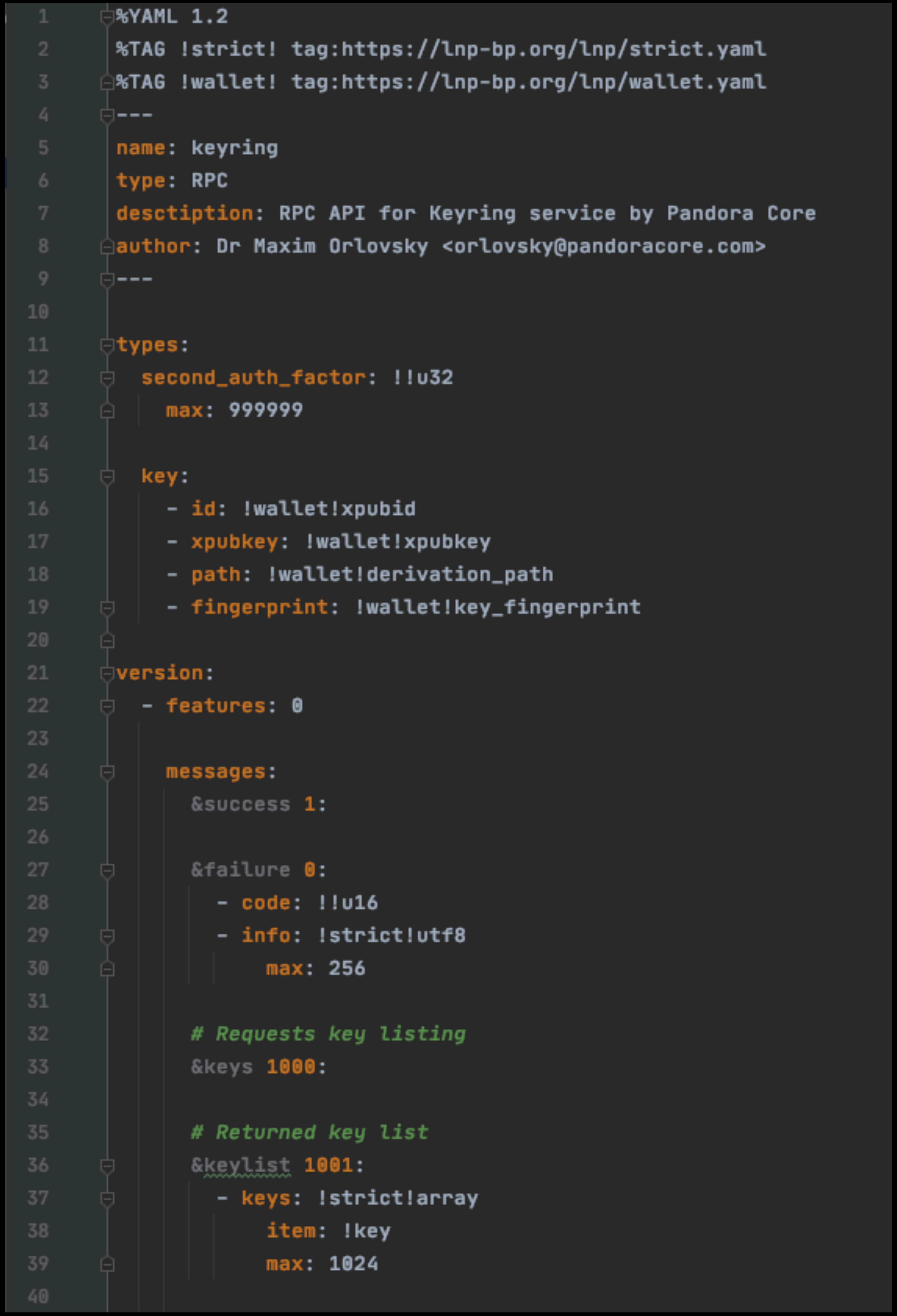
github.com/LNP-BP/lnp-api-tools

- Cross-conversion of the standards
- API validation
- Generate language-specific wrappers – but very small amount of code, audited by developers

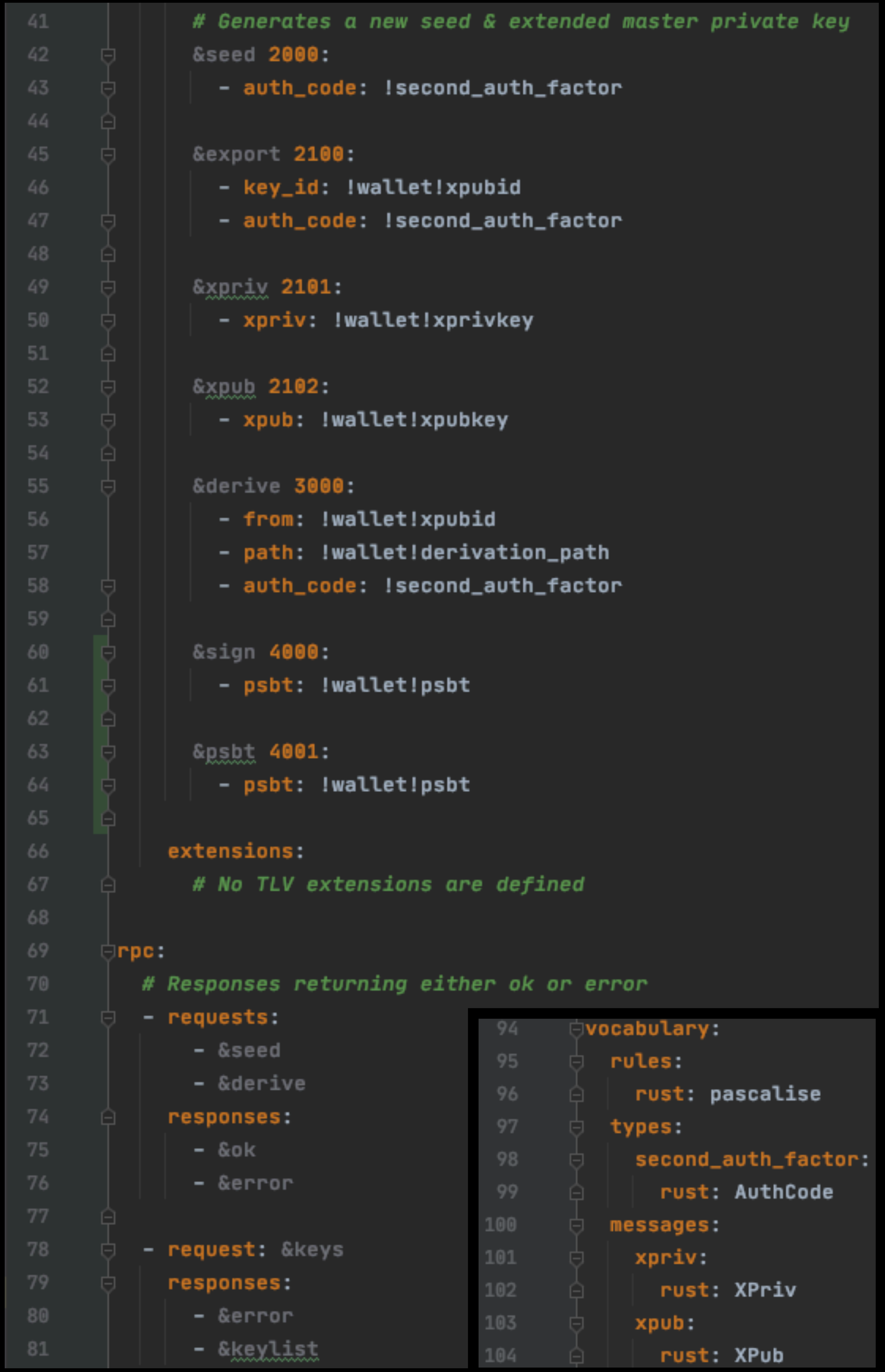
- Already used for LNP and c-lightning hybrids
- Used by all three nodes internally: BP, LNP, RGB
- Used for RPC APIs to all three nodes
- Will be used by Bifrost

# LNP API YAML interface description

- Follows strict encoding paradigm
- Language-specific customization
- Multi-file, allows extensions to existing protocols
- Can be used to write **formal deterministic** API specs (LNPBPs, BOLTS)



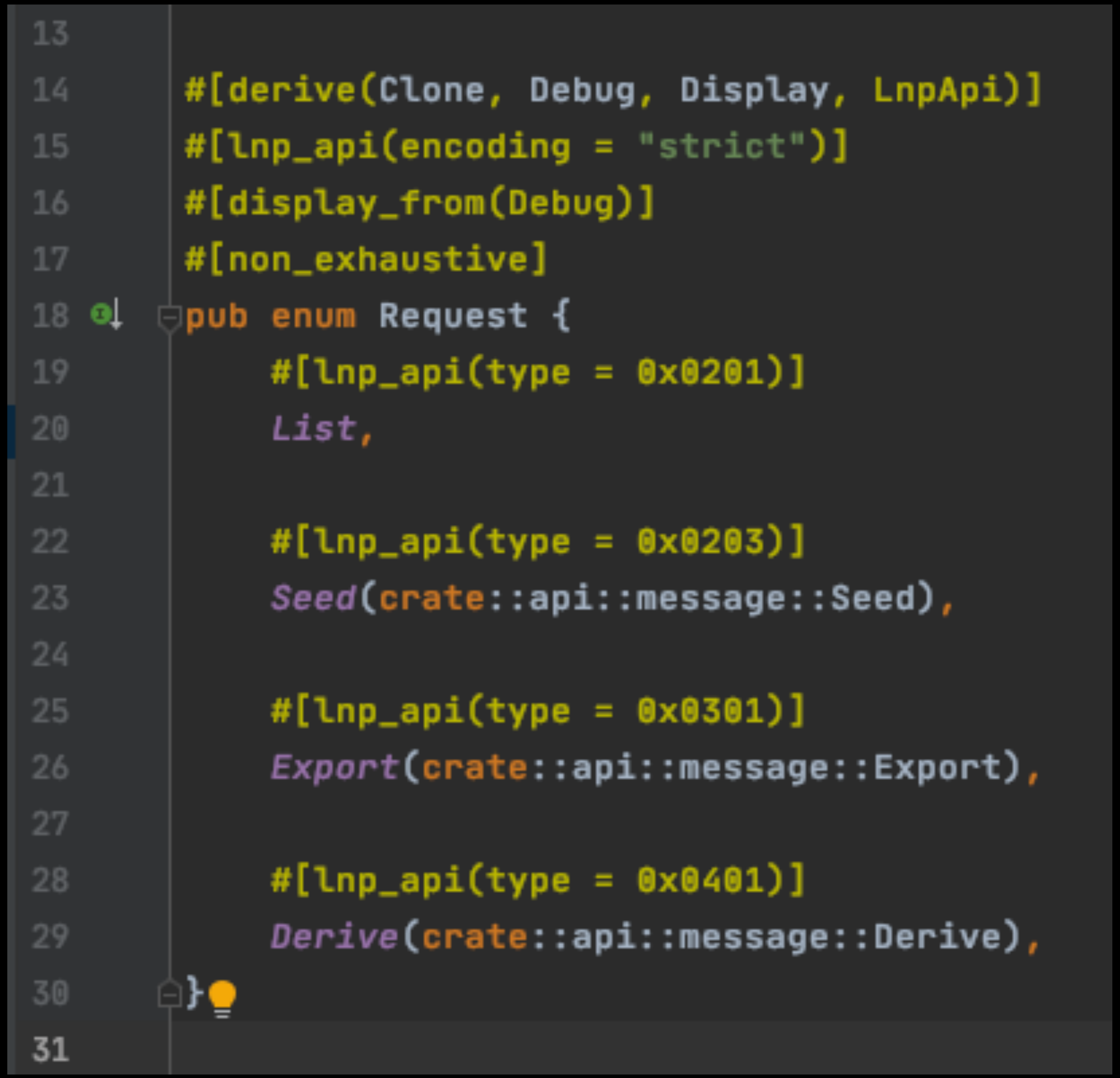
```yaml
%YAML 1.2
%TAG !strict! tag:https://lnp-bp.org/lnp/strict.yaml
%TAG !wallet! tag:https://lnp-bp.org/lnp/wallet.yaml
---
name: keyring
type: RPC
desctiption: RPC API for Keyring service by Pandora Core
author: Dr Maxim Orlovsky <orlovsky@pandoracore.com>
---

types:
  second_auth_factor: !!u32
    max: 999999

  key:
    - id: !wallet!xpubid
    - xpubkey: !wallet!xpubkey
    - path: !wallet!derivation_path
    - fingerprint: !wallet!key_fingerprint

version:
  - features: 0

    messages:
      &success 1:

      &failure 0:
        - code: !!u16
        - info: !strict!utf8
            max: 256

      # Requests key listing
      &keys 1000:

      # Returned key list
      &keylist 1001:
        - keys: !strict!array
            item: !key
            max: 1024

```



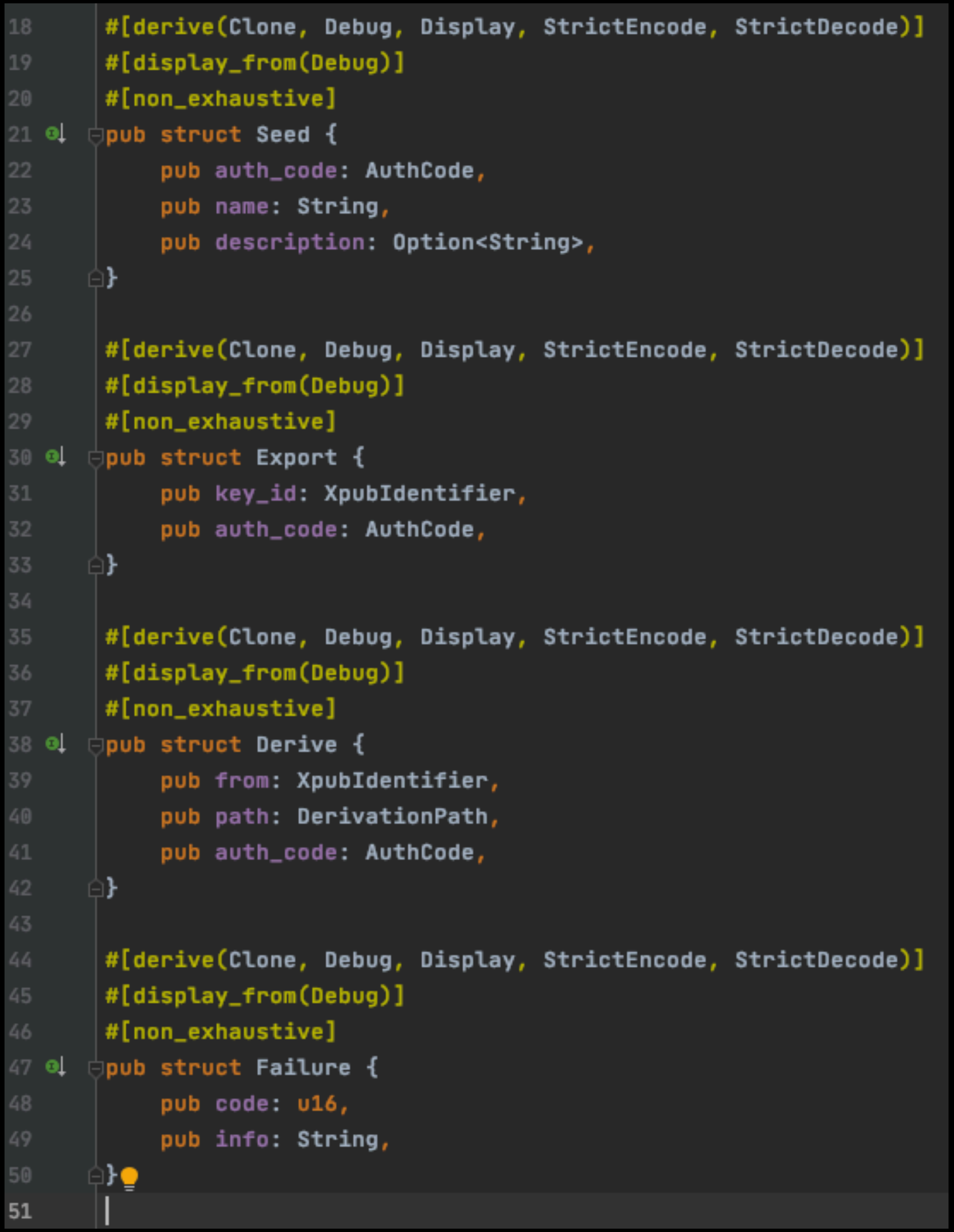
```yaml
      # Generates a new seed & extended master private key
      &seed 2000:
        - auth_code: !second_auth_factor

      &export 2100:
        - key_id: !wallet!xpubid
        - auth_code: !second_auth_factor

      &xpriv 2101:
        - xpriv: !wallet!xprivkey

      &xpub 2102:
        - xpub: !wallet!xpubkey

      &derive 3000:
        - from: !wallet!xpubid
        - path: !wallet!derivation_path
        - auth_code: !second_auth_factor

      &sign 4000:
        - psbt: !wallet!psbt

      &psbt 4001:
        - psbt: !wallet!psbt

    extensions:
      # No TLV extensions are defined

rpc:
  # Responses returning either ok or error
  - requests:
      - &seed
      - &derive
    responses:
      - &ok
      - &error

  - request: &keys
    responses:
      - &error
      - &keylist
```

```yaml
vocabulary:
  rules:
    rust: pascalise
  types:
    second_auth_factor:
      rust: AuthCode
  messages:
    xpriv:
      rust: XPriv
    xpub:
      rust: XPub
```

# Code autogeneration

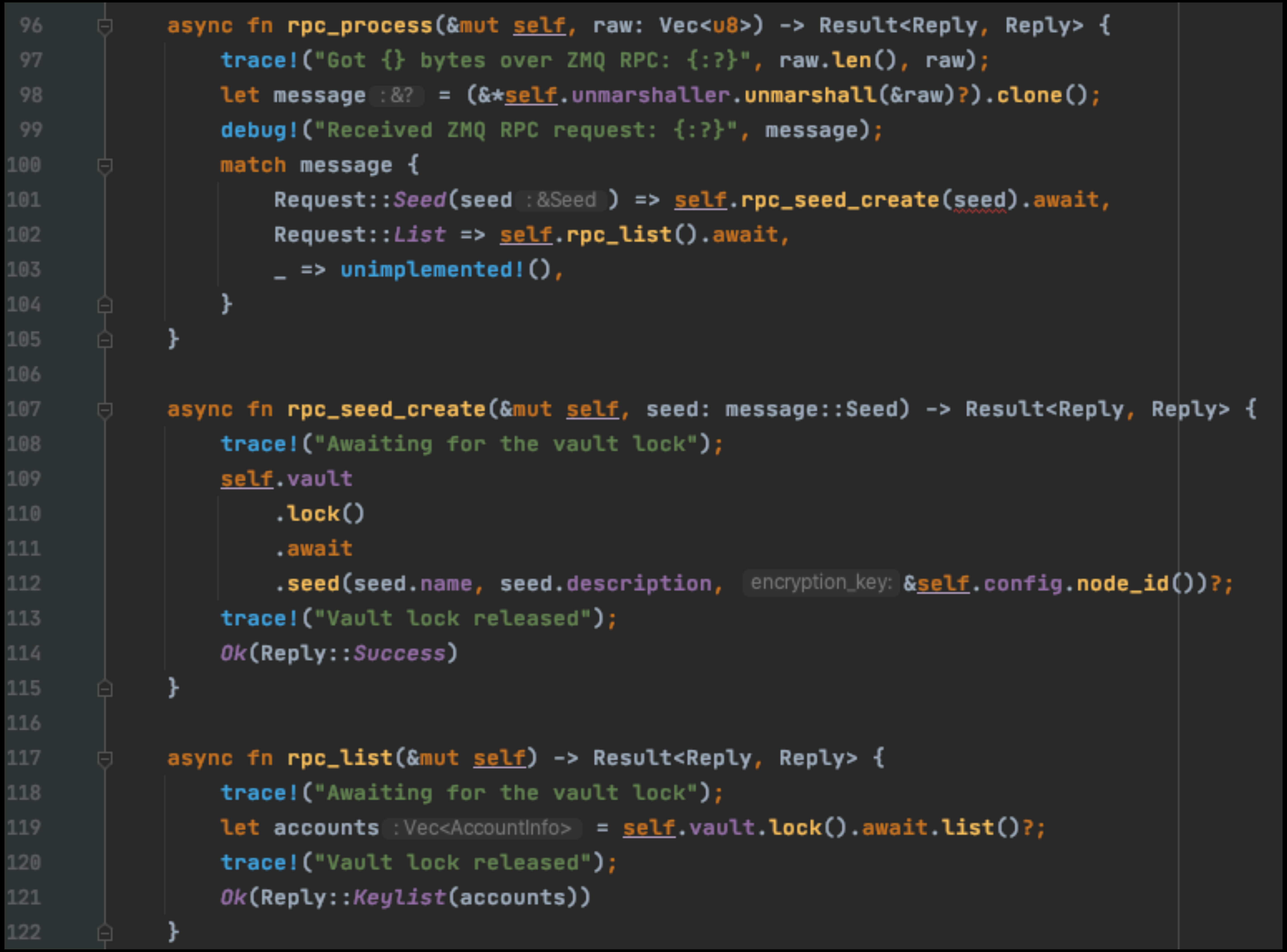


```
#[derive(Clone, Debug, Display, LnpApi)]
#[lnp_api(encoding = "strict")]
#[display_from(Debug)]
#[non_exhaustive]
pub enum Request {
    #[lnp_api(type = 0x0201)]
    List,

    #[lnp_api(type = 0x0203)]
    Seed(crate::api::message::Seed),

    #[lnp_api(type = 0x0301)]
    Export(crate::api::message::Export),

    #[lnp_api(type = 0x0401)]
    Derive(crate::api::message::Derive),
}
```



```
#[derive(Clone, Debug, Display, StrictEncode, StrictDecode)]
#[display_from(Debug)]
#[non_exhaustive]
pub struct Seed {
    pub auth_code: AuthCode,
    pub name: String,
    pub description: Option<String>,
}

#[derive(Clone, Debug, Display, StrictEncode, StrictDecode)]
#[display_from(Debug)]
#[non_exhaustive]
pub struct Export {
    pub key_id: XpubIdentifier,
    pub auth_code: AuthCode,
}

#[derive(Clone, Debug, Display, StrictEncode, StrictDecode)]
#[display_from(Debug)]
#[non_exhaustive]
pub struct Derive {
    pub from: XpubIdentifier,
    pub path: DerivationPath,
    pub auth_code: AuthCode,
}

#[derive(Clone, Debug, Display, StrictEncode, StrictDecode)]
#[display_from(Debug)]
#[non_exhaustive]
pub struct Failure {
    pub code: u16,
    pub info: String,
}
```

# Simplicity of implementation



```
async fn rpc_process(&mut self, raw: Vec<u8>) -> Result<Reply, Reply> {
    trace!("Got {} bytes over ZMQ RPC: {:?}", raw.len(), raw);
    let message :&? = (&*self.unmarshaller.unmarshall(&raw)?).clone();
    debug!("Received ZMQ RPC request: {:?}", message);
    match message {
        Request::Seed(seed :&Seed ) => self.rpc_seed_create(seed).await,
        Request::List => self.rpc_list().await,
        _ => unimplemented!(),
    }
}

async fn rpc_seed_create(&mut self, seed: message::Seed) -> Result<Reply, Reply> {
    trace!("Awaiting for the vault lock");
    self.vault
        .lock()
        .await
        .seed(seed.name, seed.description, encryption_key: &self.config.node_id())?;
    trace!("Vault lock released");
    Ok(Reply::Success)
}

async fn rpc_list(&mut self) -> Result<Reply, Reply> {
    trace!("Awaiting for the vault lock");
    let accounts :Vec<AccountInfo> = self.vault.lock().await.list()?;
    trace!("Vault lock released");
    Ok(Reply::Keylist(accounts))
}
```

# LNP API outside of RGB & LN

- Can be used to build **messenger** outside of LN network
  - optional (not required) bitcoin payments:
    - lightning invoices
    - LSAT
    - Lightspeed
  - always end-to-end encrypted, even if central server is present
  - can work over Tor and Mesh networks from day 0
- A proposal by A. Riard to move **Bitcoin Core RPC** on (de facto) this protocol https://twitter.com/Snyke/status/1262024134088970243?s=19

# Not a new standard!

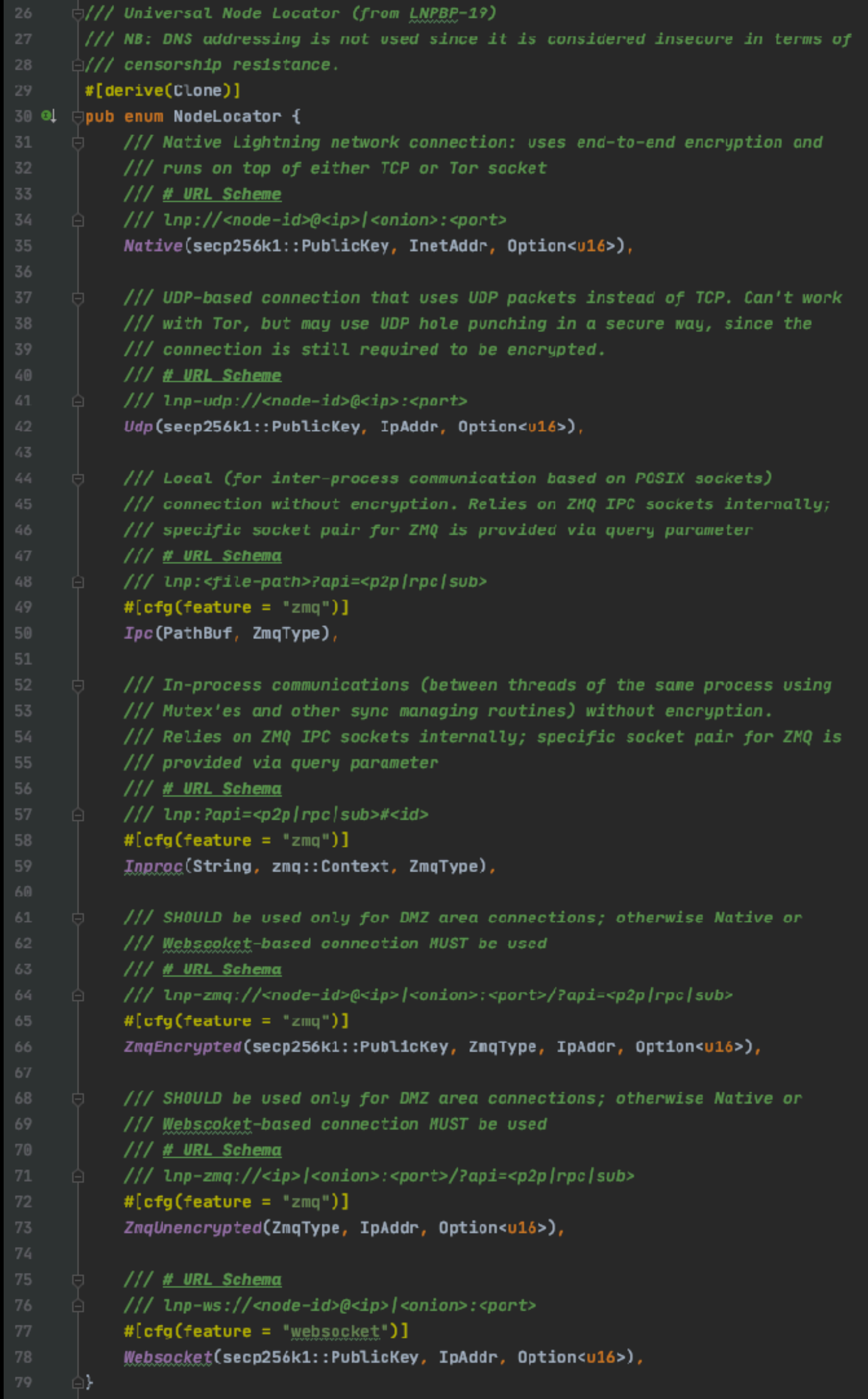
- any protocol designed in the same way as LN P2P will be automatically compliant
- **LNP API** is just a “soft-fork” extension of **LN P2P** protocols enabling them for different types of networks & transport layers (Mesh, Satellite, interprocess/IPC, Websockets etc)
- “Compatible without being (previously) aware”:
  - BOLT-13 (watchtowers)
  - Bitcoin Core RPC proposal (A. Riard)

# LNP API Summary

| Framing protocol | Standard | Encryption (BOLT-9 / LNPBP-15) | Possible API types | When to use |
|---|---|---|---|---|
| **TCP/IP & TCP/Tor** | BOLT-9,1 / LNPBP-18 | always | P2P, RPC | Default in network |
| **ZMQ** | LNPBP-17 | none | P2P, RPC, SUB | DMZ networking, ESB, IPC, inproc |
| **Websockets** | LNPBP-16 | always | P2P, RPC | Web apps |
| **UDP** | WIP | always | P2P | Low connectivity, Mesh, Satellite |
| **SMTP** | WIP | always | P2P | Mesh, Satellite, “Offline” (ulta-low connectivity) |

# LNP API transport layer selection

- Inter-process and in-process (inter-thread) APIs:
    - use unencrypted Inproc & IPC ZMQ
    - PULL/PUSH, REQ/REP and PUB/SUB sockets
- Client-server RPCs &
- P2P networks
    - use either TCP/IP, TCP/Tor or Websockets (for web-related systems); always encrypted
    - for DMZ, use ZMQ-based variant (may be unencrypted)
- Mesh and satellite networks
    - use UDP or SMTP; always encrypted



```rust
/// Universal Node Locator (from LNPBP-19)
/// NB: DNS addressing is not used since it is considered insecure in terms of
/// censorship resistance.
#[derive(Clone)]
pub enum NodeLocator {
    /// Native Lightning network connection: uses end-to-end encryption and
    /// runs on top of either TCP or Tor socket
    /// # URL Scheme
    /// lnp://<node-id>@<ip>|<onion>:<port>
    Native(secp256k1::PublicKey, InetAddr, Option<u16>),

    /// UDP-based connection that uses UDP packets instead of TCP. Can't work
    /// with Tor, but may use UDP hole punching in a secure way, since the
    /// connection is still required to be encrypted.
    /// # URL Scheme
    /// lnp-udp://<node-id>@<ip>:<port>
    Udp(secp256k1::PublicKey, IpAddr, Option<u16>),

    /// Local (for inter-process communication based on POSIX sockets)
    /// connection without encryption. Relies on ZMQ IPC sockets internally;
    /// specific socket pair for ZMQ is provided via query parameter
    /// # URL Schema
    /// lnp:<file-path>?api=<p2p|rpc|sub>
    #[cfg(feature = "zmq")]
    Ipc(PathBuf, ZmqType),

    /// In-process communications (between threads of the same process using
    /// Mutex'es and other sync managing routines) without encryption.
    /// Relies on ZMQ IPC sockets internally; specific socket pair for ZMQ is
    /// provided via query parameter
    /// # URL Schema
    /// lnp:?api=<p2p|rpc|sub>#<id>
    #[cfg(feature = "zmq")]
    Inproc(String, zmq::Context, ZmqType),

    /// SHOULD be used only for DMZ area connections; otherwise Native or
    /// Webscoket-based connection MUST be used
    /// # URL Schema
    /// lnp-zmq://<node-id>@<ip>|<onion>:<port>/?api=<p2p|rpc|sub>
    #[cfg(feature = "zmq")]
    ZmqEncrypted(secp256k1::PublicKey, ZmqType, IpAddr, Option<u16>),

    /// SHOULD be used only for DMZ area connections; otherwise Native or
    /// Webscoket-based connection MUST be used
    /// # URL Schema
    /// lnp-zmq://<ip>|<onion>:<port>/?api=<p2p|rpc|sub>
    #[cfg(feature = "zmq")]
    ZmqUnencrypted(ZmqType, IpAddr, Option<u16>),

    /// # URL Schema
    /// lnp-ws://<node-id>@<ip>|<onion>:<port>
    #[cfg(feature = "websocket")]
    Websocket(secp256k1::PublicKey, IpAddr, Option<u16>),
}
```

# LNP API URL schemes (LNPBP-39)

- Native (over **TCP/IP** and **TCP/Tor**)  
  lnp:// <node-id> @ <ip>|<onion> : <port>

- LNP over **Websockets**  
  lnp-ws:// <node-id> @ <ip>|<onion> : <port>

- LNP over **UDP** (UDP hole punching or low throughput/mesh networks)  
  lnp-udp:// <node-id> @ <ip> : <port>

- Inter-process and in-process communications (with **ZMQ**)  
  lnp-zmq: [<file-path>] ? api=<p2p|rpc|sub>

- LNP over **ZMQ** over TCP/IP or TCP/Tor  
  lnp-zmq:// <ip>|<onion> : <port>/ ? api=<p2p|rpc|sub>

# Big picture

- LNP networking is a first step towards generalized Lightning network
- LNP API stack can fix problems of modern TCP/IP combined with DNS & SSL:
  - decentralized network ids (public keys instead of certificates)
  - self-issues names (again, public keys)
  - end-to-end encryption, always
- Combined with TCP/IP/Tor, LNP API can help in building Internet2 (and not Web3:) confidential & censorship-resistant
- We design LNP API code to make future way into POSIX (Linux/UNIX) kernels
- May be, one day, Bitcoin/LN/RGB nodes will be part of OS kernel/distribution as well

# Let’s work together!

- Rust implementation:
  github.com/LNP-BP/rust-lnpbp/tree/master/src/lnp

- Sample usage:
  github.com/LNP-BP/rgb-node/blob/master/src/contracts/fungible/runtime.rs
  (LNP node and BP node will follow soon)

- API tools:
  github.com/LNP-BP/lnp-api-tools

- Standards:
  github.com/LNP-BP/LNPBPs

# RGB integration

Universal architecture and components for personal nodes, wallets, exchanges & payment providers

**LNP/BP Standards Association**
Prepared & supervised by **Dr Maxim Orlovsky**, **Pandora Core AG**

Created with support from **Bitfinex** and **Fulgur Ventures**

## Current LN nodes

- Hard to extend with custom messages (except c-lightning)
- One can’t modify the structure of commitment and other channel transactions
- “Hardcoded” to existing specs, no modularization

## And also LN should upgrade for …

- Schnorr signatures
- Taproot
- Payment points
- eltoo
- … who knows?

All these upgrades are very complex with existing node architecture

# LN software has to be ready for:

- Support for multi-peer channels
- Abstraction of commitment- and funding transaction structure
- Modularisation of penalty/escrow mechanics (HTLC->PTLC)
- Better separation of networking layers

# But are existing LN nodes ready to adopt that?

- No, at least without a deep refactoring of their architecture and lots of rewrites.

Why?

- Hardcoded uni-directed channel parameters
- No channel / connection concept separation
- Monolithic architecture (except c-Lightning)
- No plugin support (except c-Lightning)

# Architecture requirements

- Microservice-based: scalability up to multi-docker enterprise environments
- High-load processing: usage of ZeroMQ APIs instead of JSON RPC and unreliable IPC
- Subscription/push-based notification model for clients, non-custodial wallets etc
- Separation of Peers and Channels
- Extensible with new modular functionality

# LNP node (Lightning node)

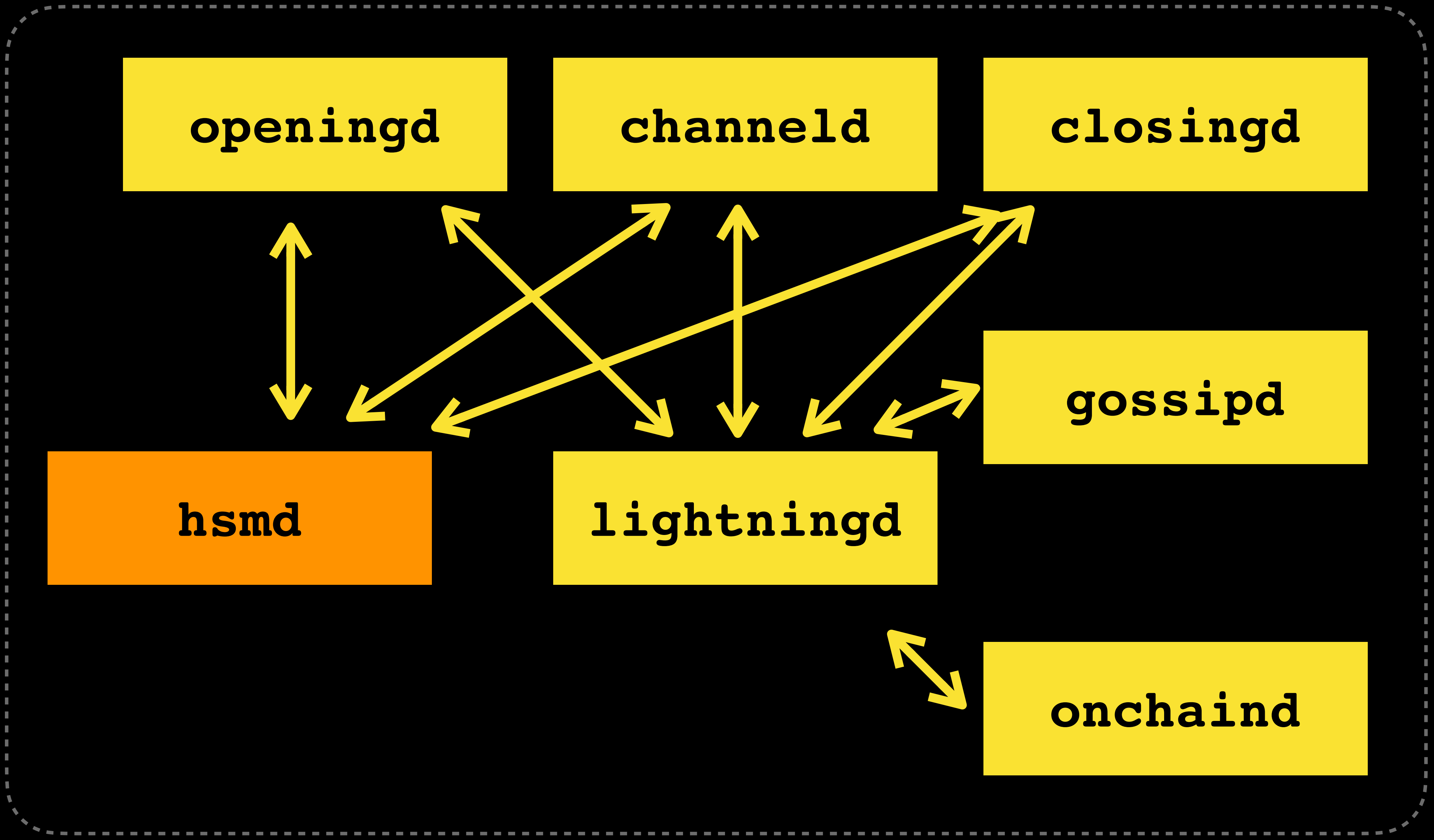
- Based on rust-lightning library by Matt Corallo @Square Crypto & @Chaincode Labs

- Utilising the same multi-thread non-blocking microservice code as Bitcoin transaction service

- Following best practices from c-Lightning architecture & extensibility

- Suited for generalised Lightning Network, ready for:
  - ◉ multi-peer channels / channel factories
  - ◉ multiple channels per peer
  - ◉ payment points
  - ◉ RGB, Spectrum
  - ◉ Protocols, that require modification of channel transaction structure (discreet log contracts, Storm, Prometheus)

# Our example to start with:
# c-Lightning multiprocess architecture



# LNP: new LN node architecture

docker container or volume
ZMQ REQ/REP
Double ZMQ REQ/REP
ZMQ PUB/SUB
connected to all channels and peers
Extensions/mods
txserv
keys daemon
PNS
push-notifications used by mobile non-custodials for signing txs
multipeer channel
onchain
cli
other clients
api
broker
routing
channel
gossip
channel
DLC
channel
RGB
peer
peer
peer
DLC
RGB
dlcd
rgbd
routing sqlite
per-channel state sqlite
gossip sqlite

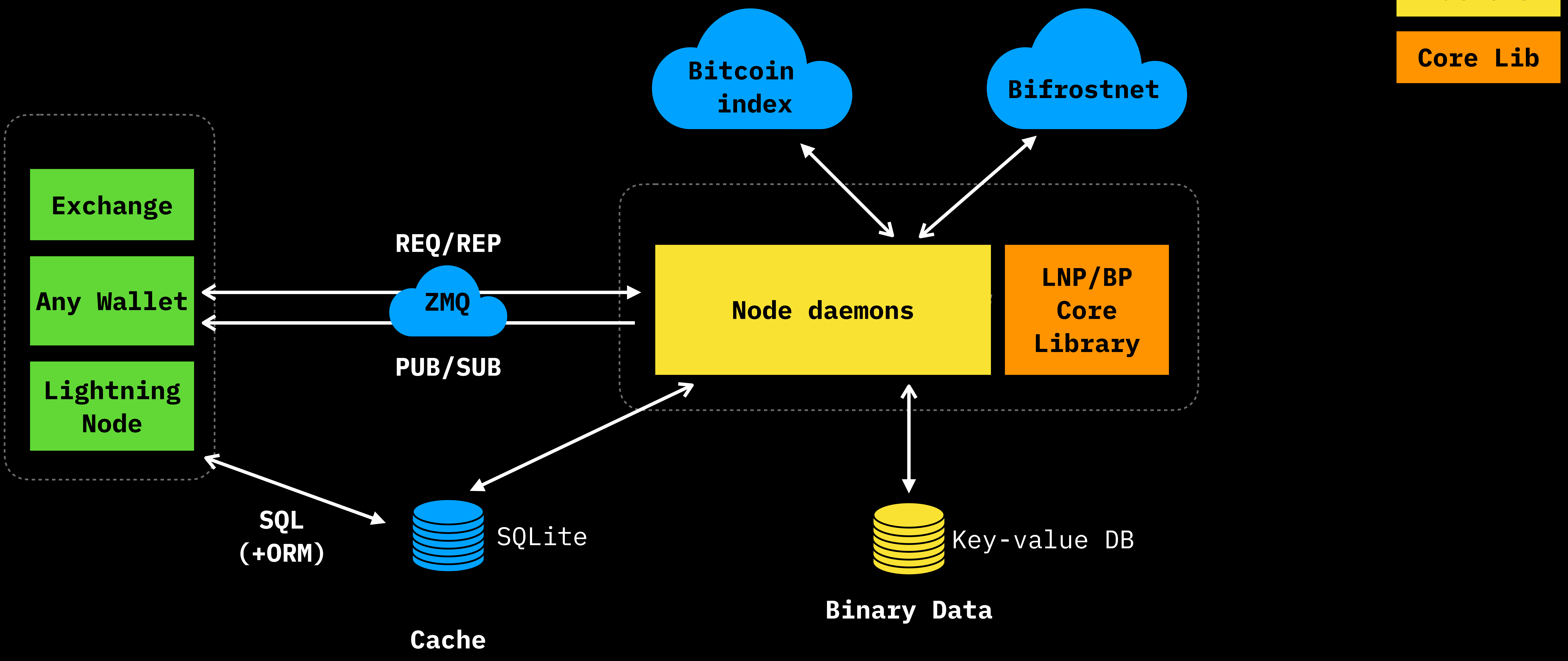
# Variants for node integration

- **Daemon-based**: multi-process elastic configuration (c-lightning-like)
  - Can be dockerized and scaled independently (by module)
  - Can run as geo-distributed cluster
  - Can be used on enterprise server or personal server
- **Service-based**: multi-threaded runtime
  - Runs in the same process as client app
  - Best for mobile
- **Proxy-based**: web model
  - Service- or daemon-based backed
  - NodeJS proxy storing RGB data on server
  - JS client library (cache-less)
- **Direct**: may be implemented in the future; not recommended
  - WASM and C FFI bindings with language-specific wrappers
  - No ZMQ, no multithreading

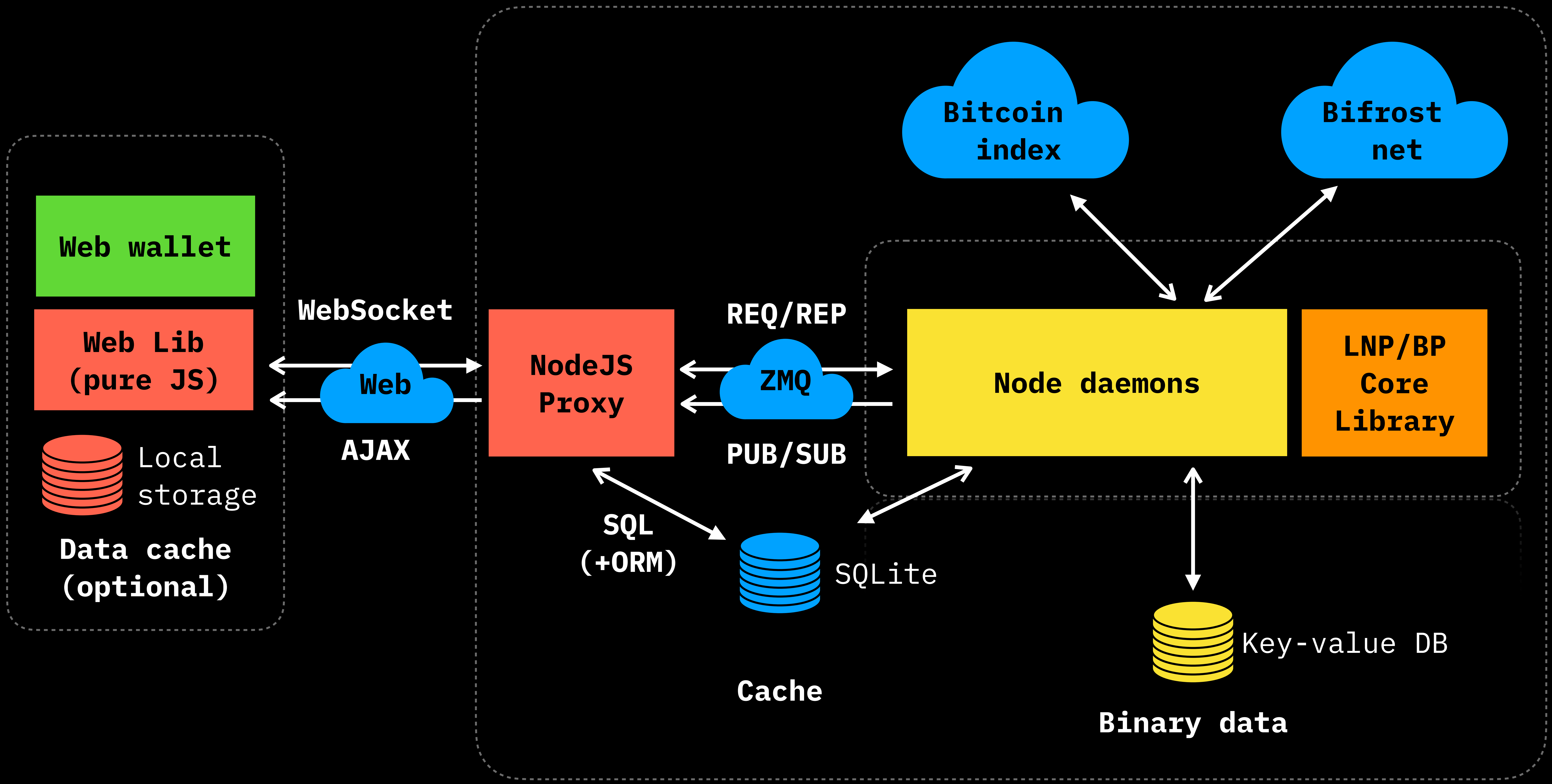
# RGB Integration SDK

- **Binaries**
  - platform-specific runtime library (for service-based integration)
  - executables (for daemon-based integration)
  - in future: WASM & binary library for direct integration (not recommended)
- **Docker images** (can be used in daemon-based integration only)
- Language-specific integration for loading RGB runtime as service
  - Swift
  - Kotlin
- Class abstractions in JS, Swift & Kotlin for
  - ZMQ Client API
  - ORM data objects
- Web proxy service (NodeJS)
- Future: language-specific direct integration class libraries (not recommended)

# Universal LNP/BP node architecture

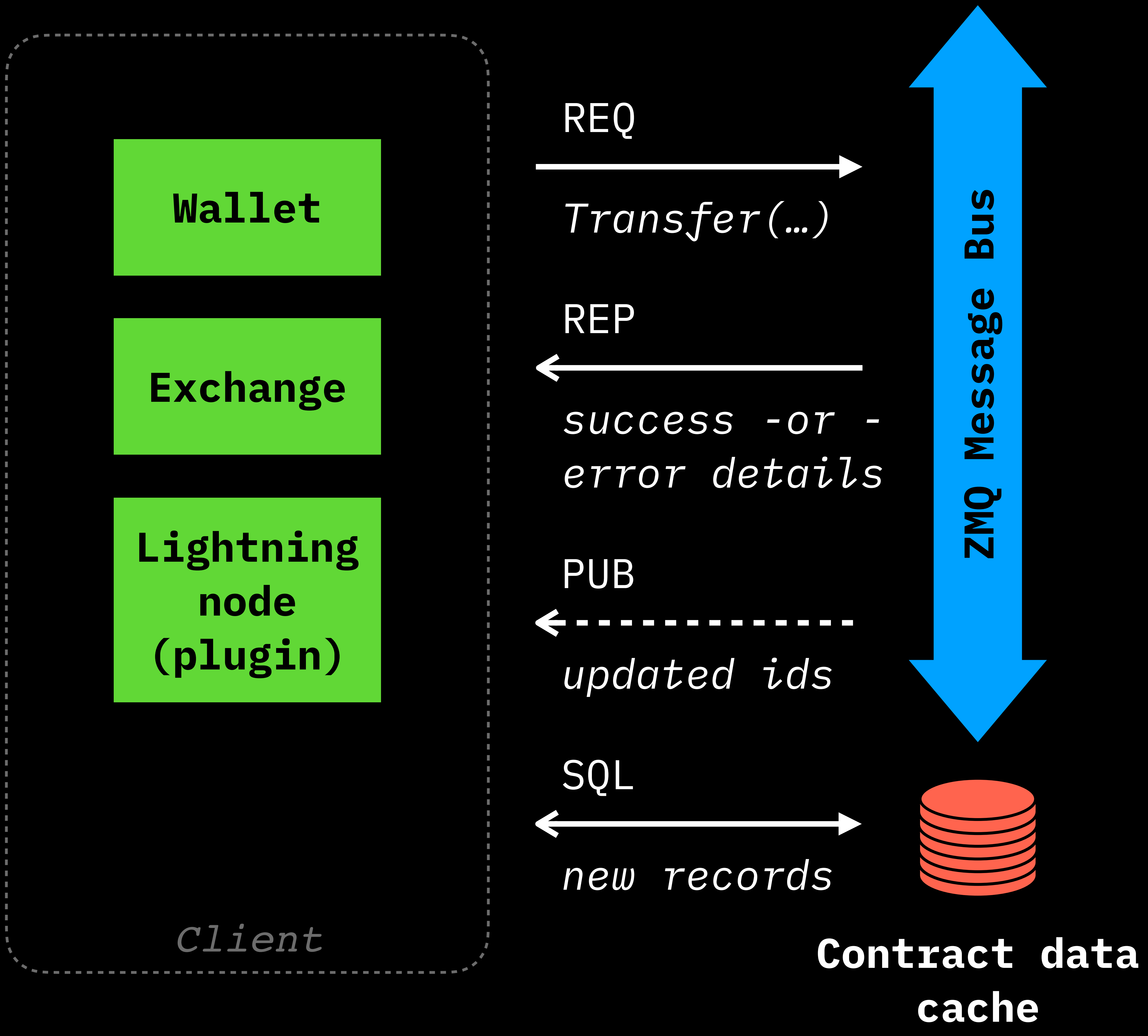


# Universal LNP/BP node architecture: Web



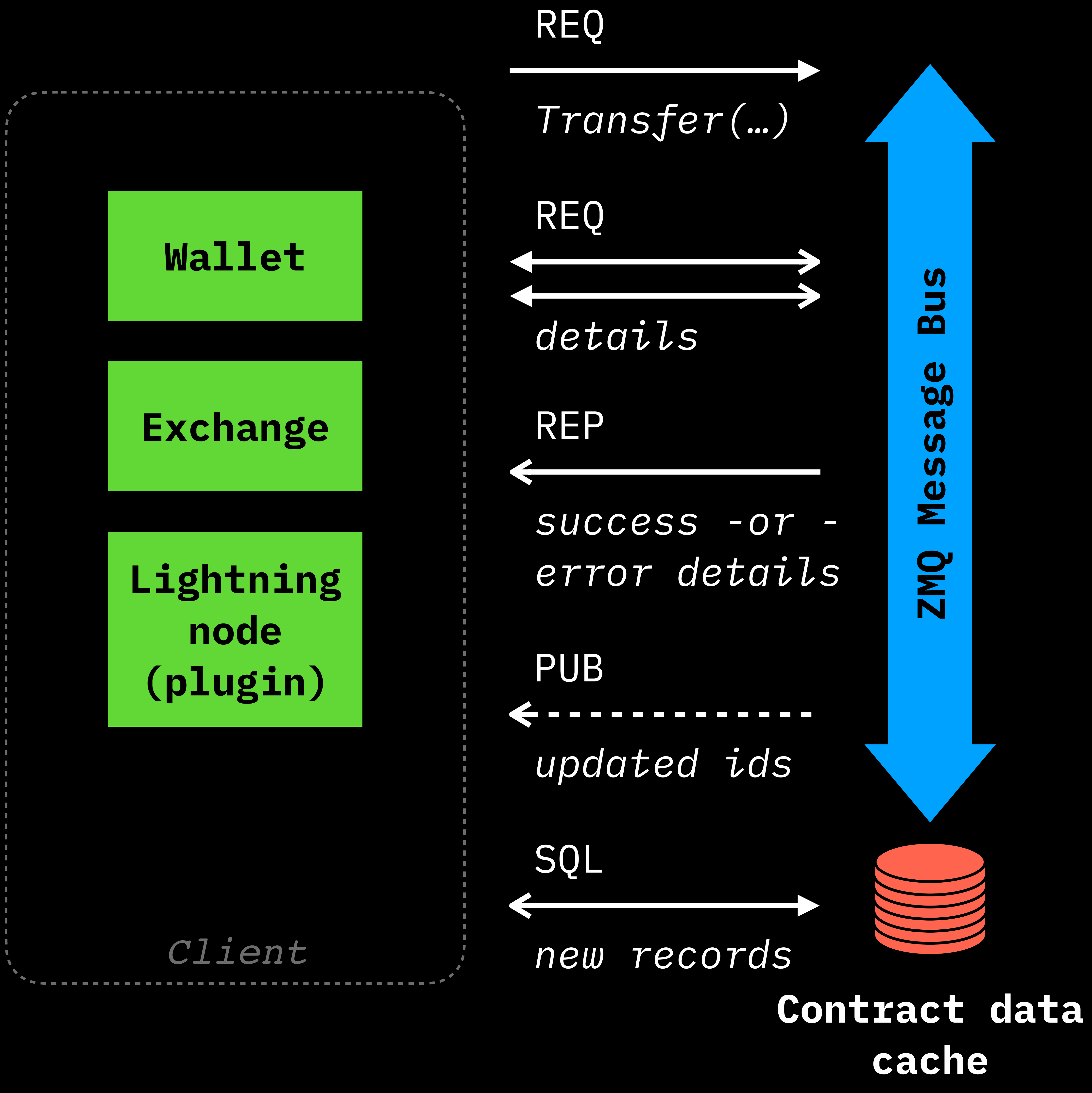
# High-level RGB API integration

- Send simple request via ZMQ API to corresponding contract service (like fungible assets)
- Get response of success/failure code
- Wait for PUB information which entities were updated
- Read updated entities from SQL using your ORM



# Low-level RGB API integration

- Open special per-request ZMQ REQ socket
- Send request via ZMQ API providing all required details via optional parameters; pass socket details
- Reply with necessary information on each incoming REQ (“callbacks”) until you’ll…
- …get response of success/failure code
- Wait for PUB information which entities were updated
- Read updated entities from SQL using your ORM



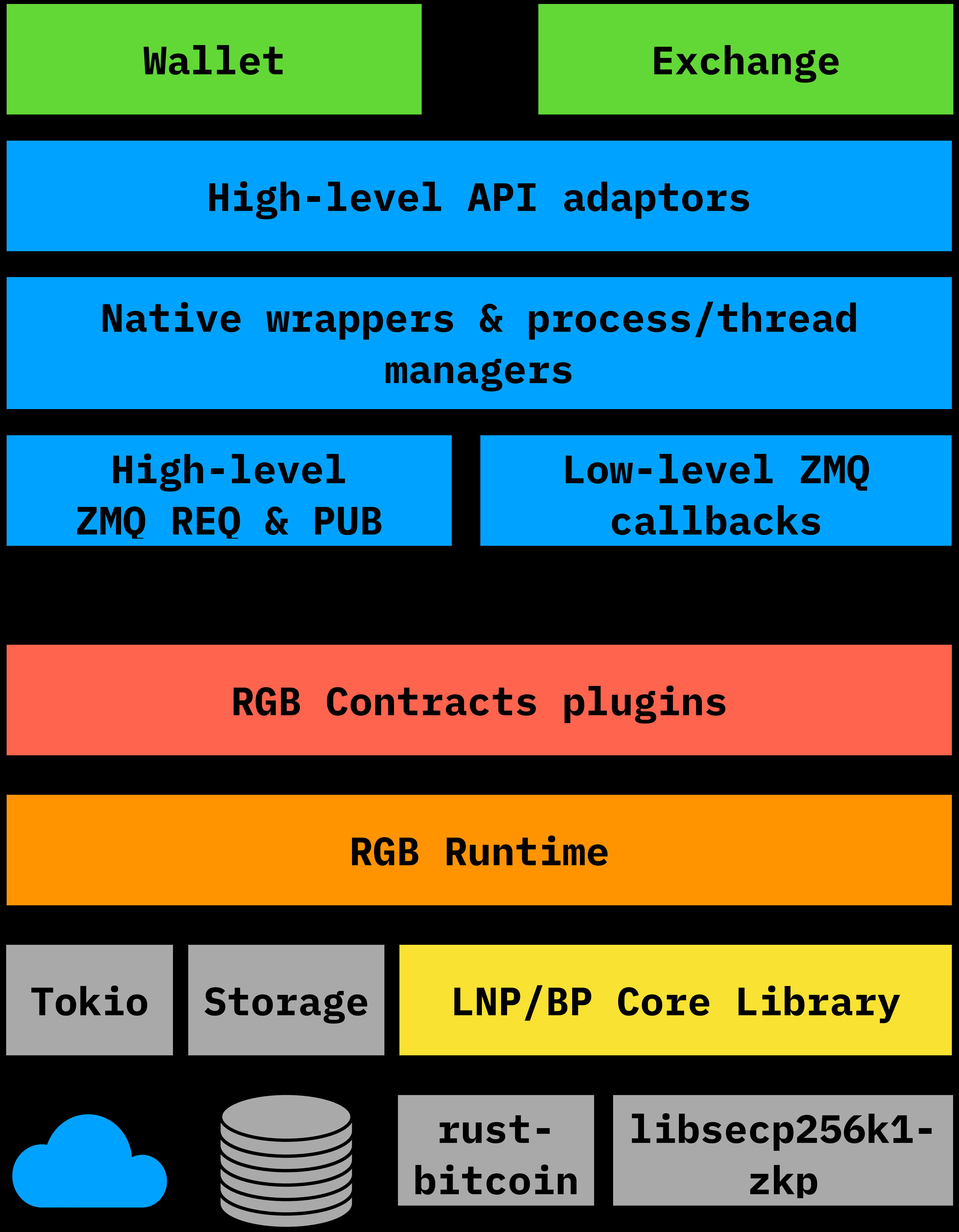
# Integration Stack

- Launch point:
  `node(config).launch()`

- Wallet using high-level class wrapper (sample for RGB node):
  `RGB(context).pay(invoice)`

- Wallet using low-level class wrapper:

```
rgb_pay_invoice(
  invoice,
  context,
  coordinator_callback,
  transaction_constructor_callback,
  coin_selection_callback)
```



# LNP/BP DEX

Integrating third-layer protocols to build DEX:
RGB, DLC, LNP

**LNP/BP Standards Association**
Prepared by **Dr Maxim Orlovsky**, **Pandora Core AG**

Created with support from **Bitfinex & Fulgur Ventures**

# Leverage existing & don't reinvent the wheel

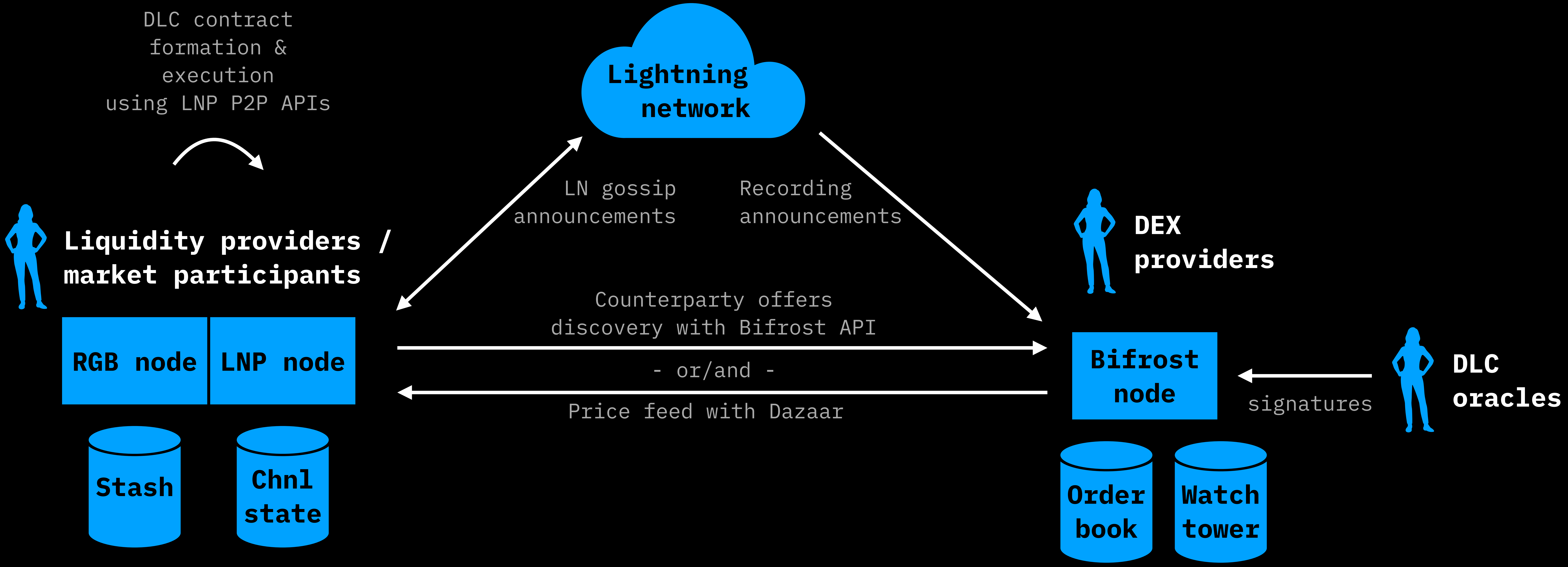
- Protocols:
  - **LNP**: networking protocol for end-to-end encrypted P2P and RPC APIs used by LN, RGB (potentially in future – DLC)
  - **RGB**: information exchange between LN nodes on their assets (based on LNP)
  - **DLC**: ongoing work on decentralized arbitrage network with oracles
  - **Dazaar**: real-time data exchange P2P protocol
- Software:
  - **LNP Node**: modular lightning node (under development) with support for RGB & LNP
  - **Bifrost**: data storage node (planned) used in RGB & Storm, which can be used for maintaining data on orderbooks

# Bifrost node & infrastructure

- General decentralized storage server:
  manages key-value indexes of encrypted, hashed or blinded blob data

- Used by RGB for:
  - accept payments confirmation
    (RGB Consignments) when receiver’s wallet is offline
  - hold backup for RGB Stash (in encrypted form)
- Other usage:
  - LN watchtowers
  - DEX orderbooks
  - Gossip message propagation (outside of native LN payments)
  - DLC oracle?
- Paid with Lightspeed & Storm (when its out)

# DEX with RGB, DLCs & generalized LN


DLC contract formation & execution using LNP P2P APIs
Lightning network
LN gossip announcements
Recording announcements
Liquidity providers / market participants
RGB node
LNP node
Stash
Chnl state
Counterparty offers discovery with Bifrost API
- or/and -
Price feed with Dazaar
DEX providers
Bifrost node
signatures
DLC oracles
Order book
Watch tower