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**

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 *

LNP: universal networking protocol for bitcoin world

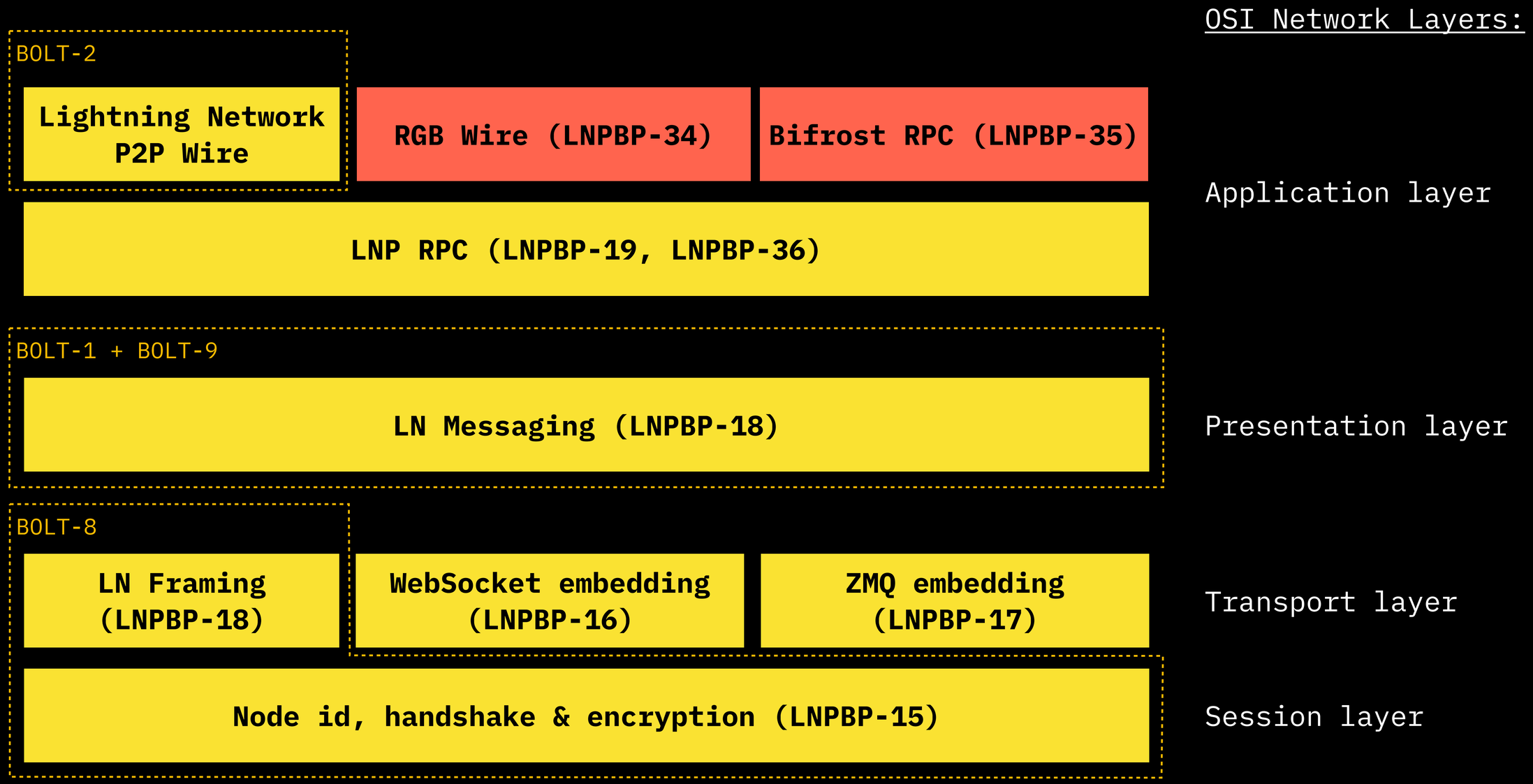
Decentralized & encrypted

- 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

Lightning Network Protocols (LNP) suite



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)

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

```yaml
--- # LNP API definition file; autogenerated from c-lightning CSV API file

types:
  - &node_id { snake: node_id, pascal: NodeId, camel: nodeId }
  - &wirestring { snake: wirestring, pascal: Wirestring, camel: wirestring }
  - &u16 { snake: u16, pascal: U16, camel: u16 }
  - &u8 { snake: u8, pascal: U8, camel: u8 }
  - &bip32_key_version { snake: bip32_key_version, pascal: Bip32KeyVersion, camel: bip32KeyVersion }
  - &chainparams { snake: chainparams, pascal: Chainparams, camel: chainparams }
  - &secret { snake: secret, pascal: Secret, camel: secret }
  - &privkey { snake: privkey, pascal: Privkey, camel: privkey }
  - &secrets { snake: secrets, pascal: Secrets, camel: secrets }
  - &sha256 { snake: sha256, pascal: Sha256, camel: sha256 }
  - &ext_key { snake: ext_key, pascal: ExtKey, camel: extKey }
```

LNP API YAML interface description

```yaml
messages:
  1000: &hsmstatus_client_bad_request
    - { name: id, type: *node_id }
    - { name: description, type: *wirestring }
    - { name: len, type: *u16 }
    - { name: msg, type: *u8, len_var: len }
  11: &hsm_init
    - { name: bip32_key_version, type: *bip32_key_version }
    - { name: chainparams, type: *chainparams }
    - { name: hsm_encryption_key, type: *secret, optional: true }
    - { name: dev_force_privkey, type: *privkey, optional: true }
    - { name: dev_force_bip32_seed, type: *secret, optional: true }
    - { name: dev_force_channel_secrets, type: *secrets, optional: true }
    - { name: dev_force_channel_secrets_shaseed, type: *sha256, optional: true }
  111: &hsm_init_reply
    - { name: node_id, type: *node_id }
    - { name: bip32, type: *ext_key }
  9: &hsm_client_hsmfd
    - { name: id, type: *node_id }
    - { name: dbid, type: *u64 }
    - { name: capabilities, type: *u64 }
  109: &hsm_client_hsmfd_reply
  10: &hsm_get_channel_basepoints
    - { name: peerid, type: *node_id }
    - { name: dbid, type: *u64 }
```

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 after LN P2P will be 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	optional	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
    /// Websocket-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
    /// Websocket-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:// [<node-id> @] <ip>|<onion> : <port>/ ? api=<p2p|rpc|sub>

Long road to the future

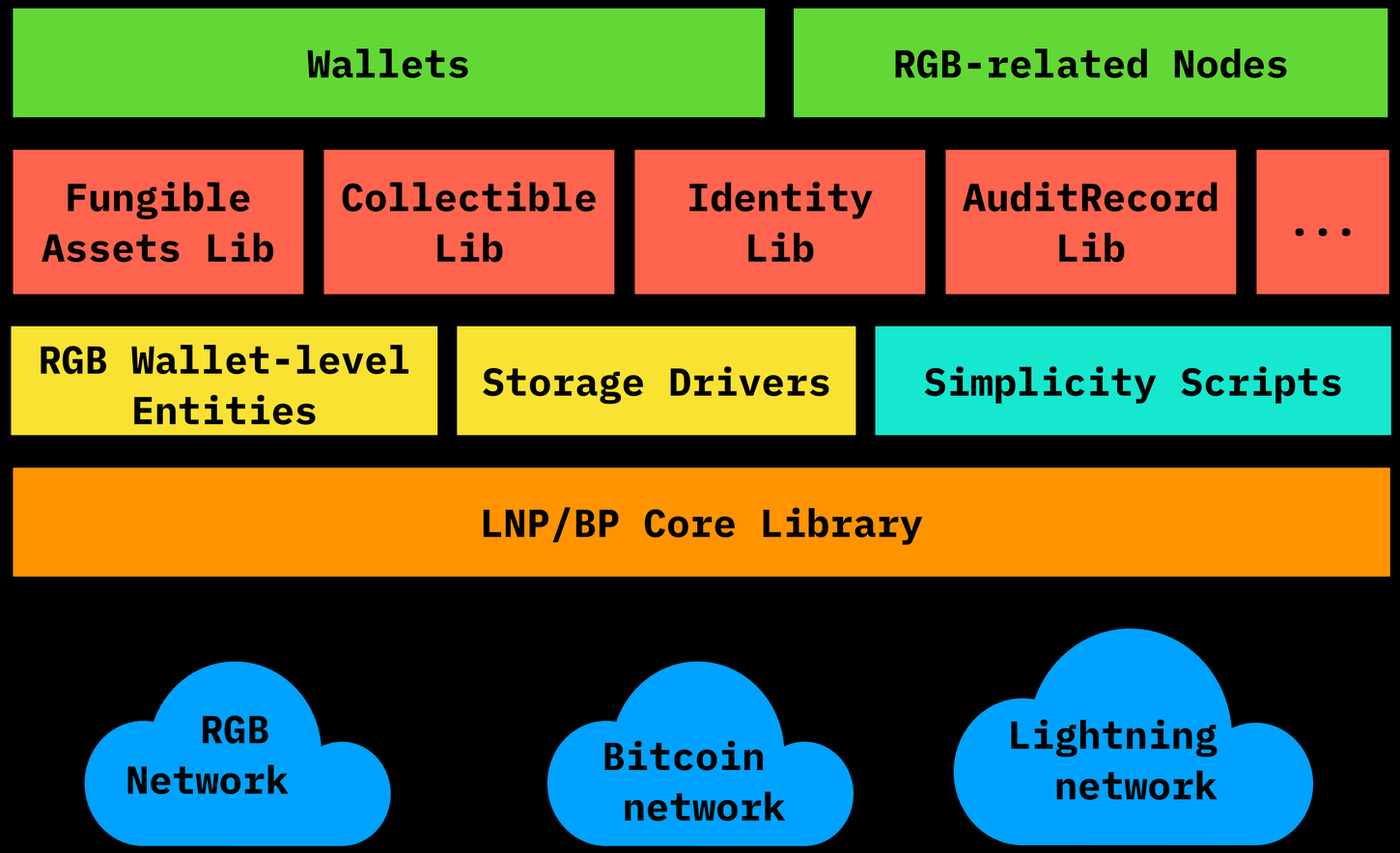
- LNP API stack can fix problems of
  modern TCP/IP combined with DNS & SSL:
  - decentralized network ids (public keys instead of certificates)
  - self-issues names (again, public keys)
  - end-to-end encryption, always
- Combined with TCP/IP/Tor, LNP API can help in building Internet2:
  confidential & 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

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**

RGB Software Architecture Layers



High-level abstractions specific for each Schema

Wallet and node integration stuff (+networking, storage, threads, signatures etc)

Consensus- and validation-critical things (serialization but not storage/networking; no async tasks or threads; thread-safe)

Wallet types possible with RGB:

Serverless wallet

- non-custodial
- fully validating(!)
- no server(s) required (outside of generic Bitcoin/LN)
- may work with public **Bifrost** servers to reduce locally-kept data size: anonymous indexes are moved to the cloud

Full wallet

- non-custodial
- fully validating(!)
- requires **Bifrost** with anonymous data for keeping RGB stash and it’s indexing

Light wallet

- custodial or non-custodial
- non-validating
- requires **RGB full validating node** (personal, otherwise exposes sensitive information)
- Not “SPV”: validation is still done, but at the level of server-side RGB full node

Nodes by LNP/BP Standards Association, part I

RGB node

- Not required for serverless or full wallets
- Does validation
- Keeps privacy-sensitive and ownership-critical information (RGB Stash)
- Can backed stash up in anonymous fashion with Bifrost (once Storm will be out)

Bifrost

- 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 index for wallets & full nodes (not required, but useful)
  - hold backup for RGB Stash (in encrypted form)
- Paid with Lightspeed & Storm (when its out)

- **Both use LNP API with ZMQ**

Nodes by LNP/BP Standards Association, part II

LNP node

- Can work with RGB node natively to bring RGB assets into Lightning channels
- Right now a hybrid rust & c-lightning node, having
  - c-lightning fork with HSMD API modification
  - rust-lightning-based HSMD replacement for c-lightning supporting DBCs
  - c-lightning RGB plugin with RGB node API
- In the future: full LN node with modular architecture for
  - Channel factories, bi-directional channels
  - Pay to point (HTLC replacement)
  - DLCs over LN
  - Schnorr, Taproot experimentation
  - Eltoo

BP node

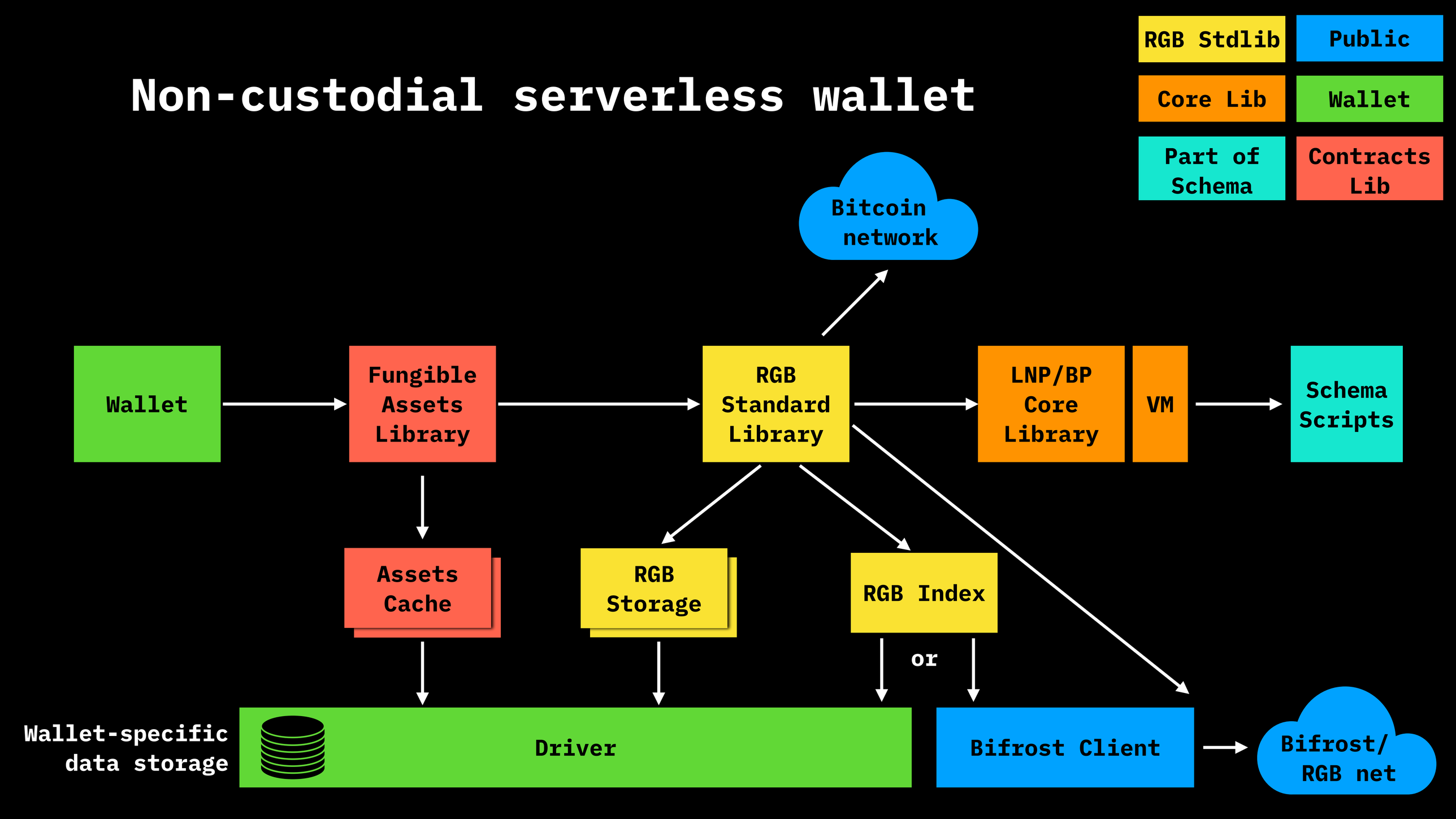
- Maintains bitcoin blockchain index required for RGB (allows to get spending tx ids for a given output)
- Also a replacement for Electrum server
- Uses universal bitcoin id’s (LNPBP-5) for making index ultra-compact
- In the future:
  - Full validating node with libconsensus
  - Mobile version of the full validated node with blockchain stored on the phone

- **Both use LNP API with either ZMQ/Websockets (for client-server API) and TCP/IP/Tor (for P2P)**

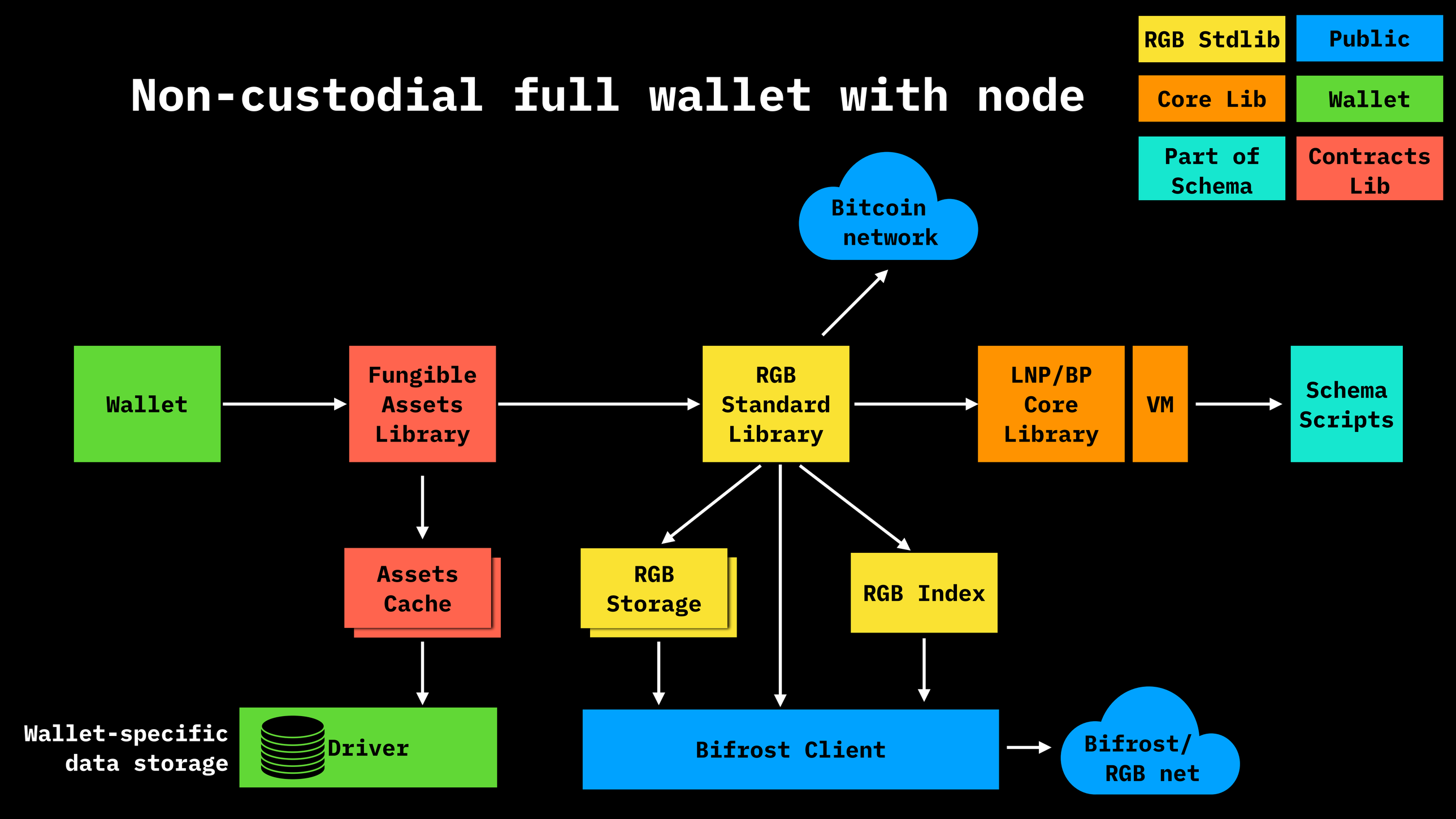
See “LNP/BP Nodes Initiative” presentation for more details on LNP and BP nodes

github.com/LNP-BP/devcalls/blob/master/LNP-BP Nodes Initiative.pdf

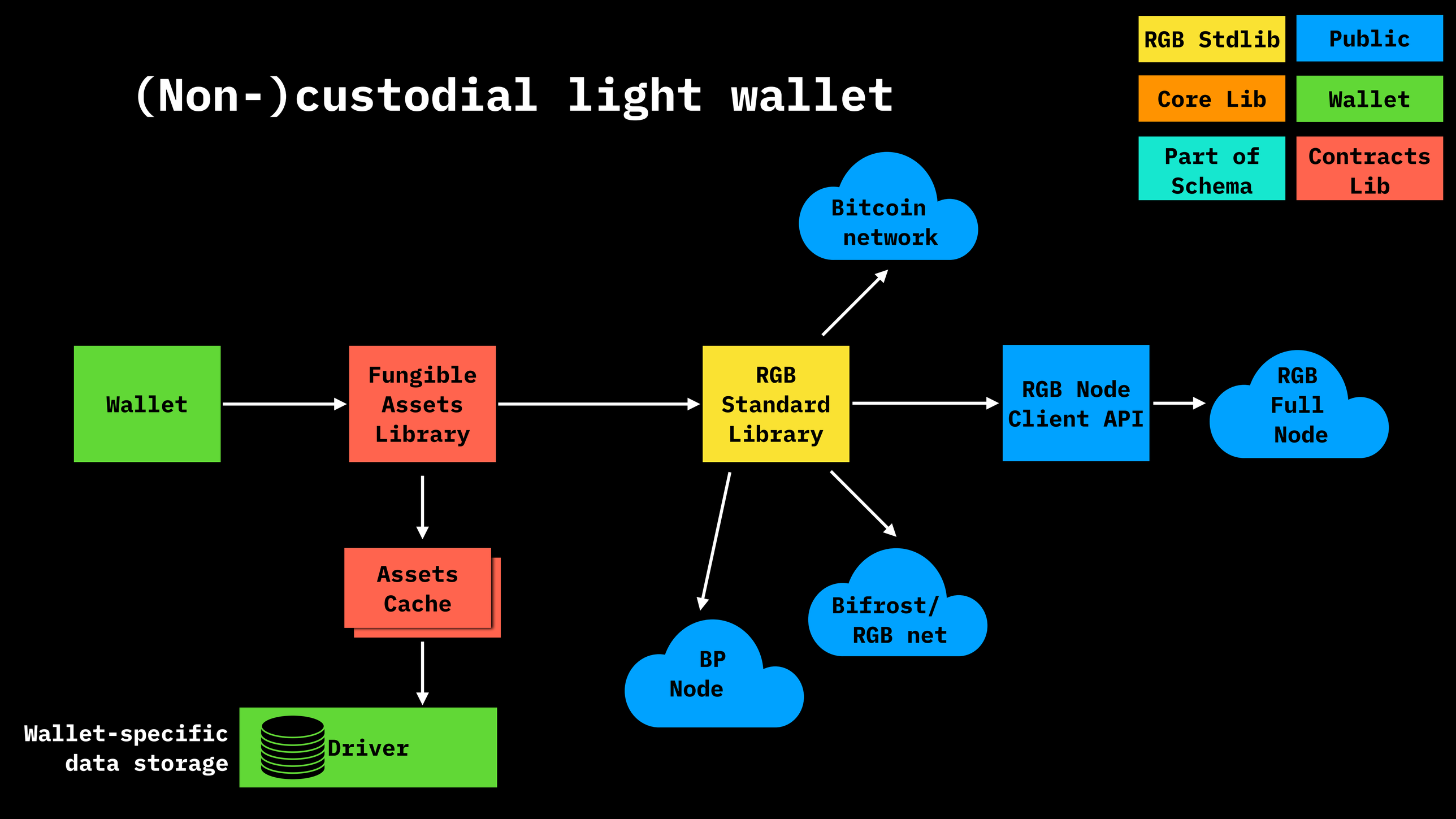
Non-custodial serverless wallet



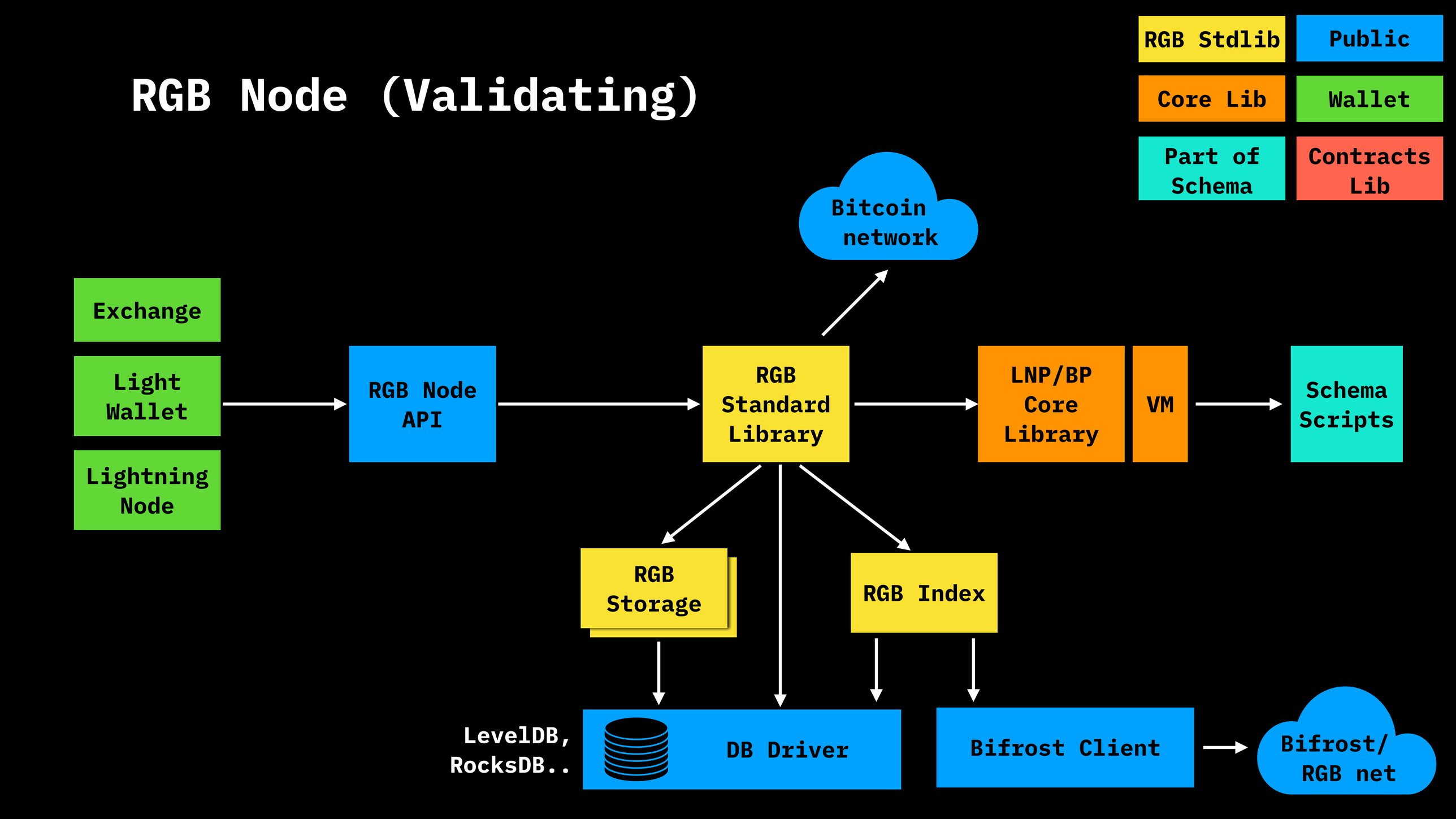
Non-custodial full wallet with node



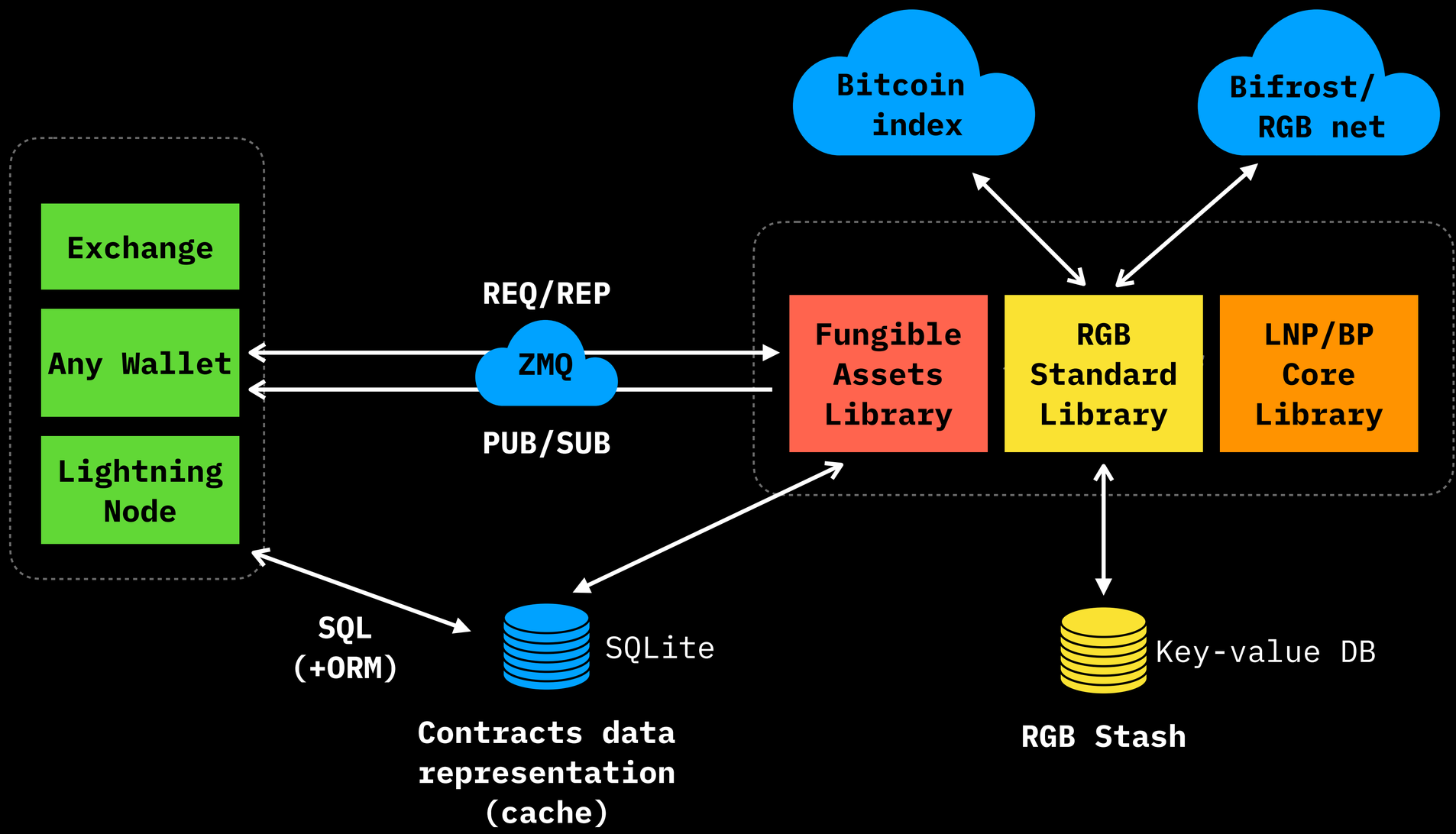
(Non-)custodial light wallet



RGB Node (Validating)



Universal RGB architecture



Shared

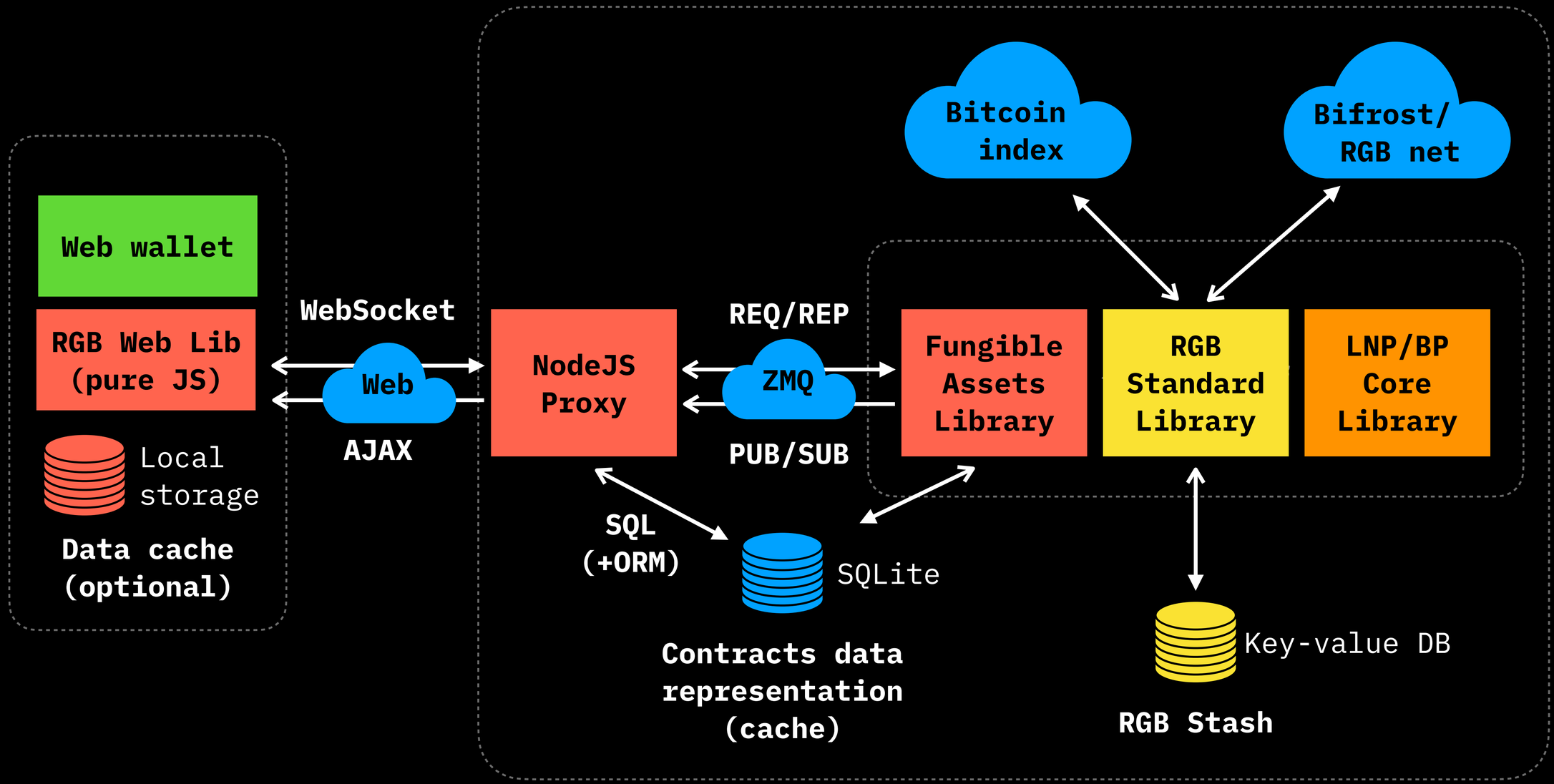
Clients

Contracts Lib

RGB Stdlib

Core Lib

Universal RGB architecture



Shared

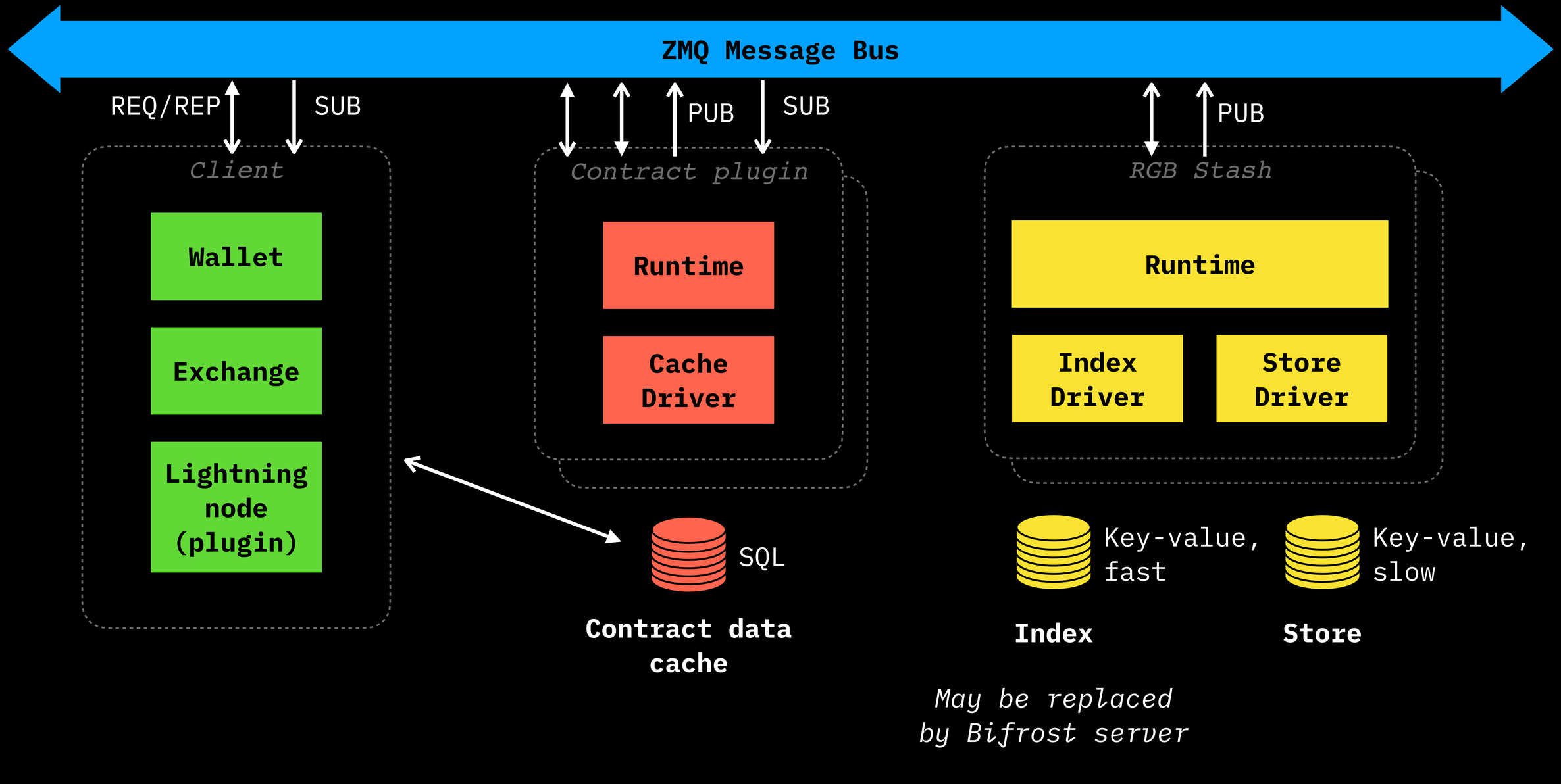
Clients

Contracts Lib

RGB Stdlib

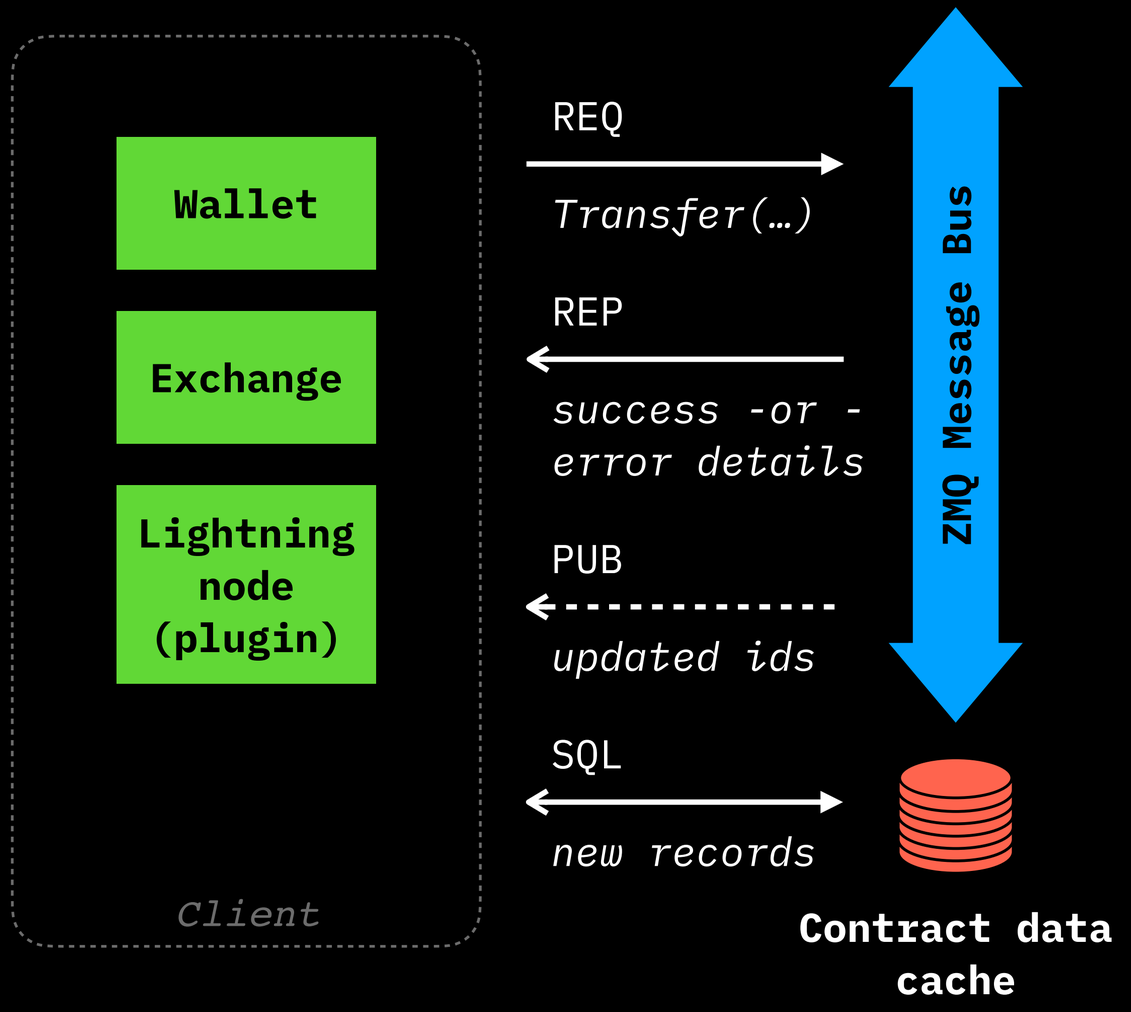
Core Lib

Universal RGB runtime architecture



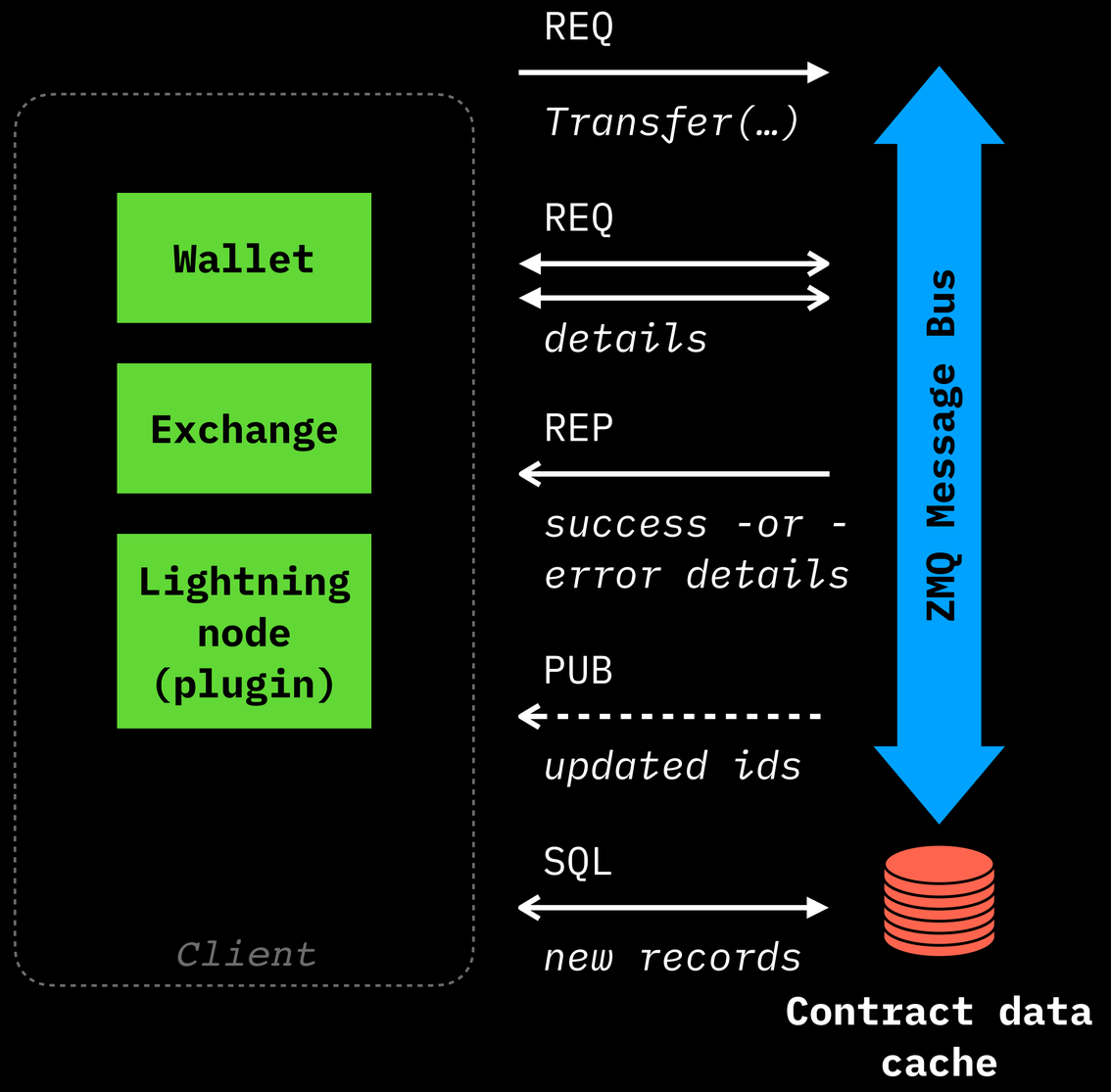
High-level RGB API integration

- 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



Variants for RGB 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)

Integration Stack

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

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

- Wallet using low-level class wrapper:

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