Releasing RGB: smart contracts

for Bitcoin and Lightning Network

part 1

Dr Maxim Orlovsky

Chief engineer @ LNP/BP Standards Association – orlovsky@lnp-bp.org

FBDE A843 3DDC 1E69 FA90 C35E FFC0 2509 47E5 C6F7

Agenda

- Few words about us
- Overview of RGB as system
- Details about RGB needed for contract issuers, wallet devs and users
- RGB release roadmap
- RGB Node work overview & demo
- Issuing fungible assets on RGB

What would not be covered today

- Advanced RGB asset issuance
- How to integrate RGB into wallet and exchange
- Demo of using RGB with a wallet
- Demo of using RGB with LN
- RGB and NFTs
- Creating custom RGB contracts

These topics will be covered during the next call(s)

LNP/BP Association

LNP/BP Standards Association

LNP/BP Association mission is to ensure that the development of Bitcoin, Lightning and protocols on top of them allows to digitally manifest the values of **censorship resistance**, **individual privacy** and **unconfiscatability** at an economy-scale level

The largest Bitcoin tech non-profit in Switzerland

github.com/LNP-BP

Search or jump to... / Pull requests Issues Marketplace Explore

LNP/BP Association

Non-profit supervising layer 2 & 3 protocols on Bitcoin & Lightning Network

Bitcoin https://lnp-bp.org @lnp_bp info@@lnp-bp.org

Unfollow

Overview Repositories 43 Packages People 22 Teams 5 Projects 13 Settings

.github / profile / README.md

LNP/BP[1] Standards Association

We are Swiss non-profit **supervising layer 2 & 3 open standards and protocols for Bitcoin & Lightning Network**. We are creators of L2 and L3 protocols like RGB, Bifrost, Storm, Prometheus, Kaleidoscope and active builders of **#BiFi** (bitcoin finance) ecosystem on Lightning. We manage set of LNPBP standards and their opensource reference implementations under permissive MIT & Apache2 licenses. The Association was founded by @dr-orlovsky and @giacomozucco in 2019. You can read more about us on our website, lnp-bp.org and follow us on Twitter @lnpbp.

LNPBP Standards

The current list of standard can be found here. You can:

- submit a new standard proposal
- discuss preliminary ideas about new standards
- follow announcements about standard releases
- write about your implementation of one of the standards
- ask questions
- peer review & audit existing standards

People

View all

Invite someone

Top languages

Rust C Python Swift Dockerfile

Most used topics Manage

bitcoin client-side-validation lnp-bp lightning-network distributed-systems

LNP/BP Association in 2019-2022

- Led development of rust bitcoin **Taproot** implementation
- Re-implementated of Lightning network spec and created **a new rust LN node** done from scratch
- Developed all of the current **RGB protocol** design and did its consensus implementation
- Developed set of **LNP/BP nodes** to use RGB protocol with LN, Bifrost, Storm and wallets
- Completed design for fungible (assets) and NFT (including identity) RGB schemata

Contribute & donate

- Discussions, proposals, Q&A: GitHub.com/orgs/LNP-BP/discussions
- Standards: github.com/LNP-BP/LNPBPs
- RGB working group: github.com/RGB-WG
- Lightning working group: github.com/LNP-WG
- Decentralized storage & messaging group: github.com/Storm-WG

Support us donating on GitHub
– or by sending funds to QR code on the left

Important RGB update

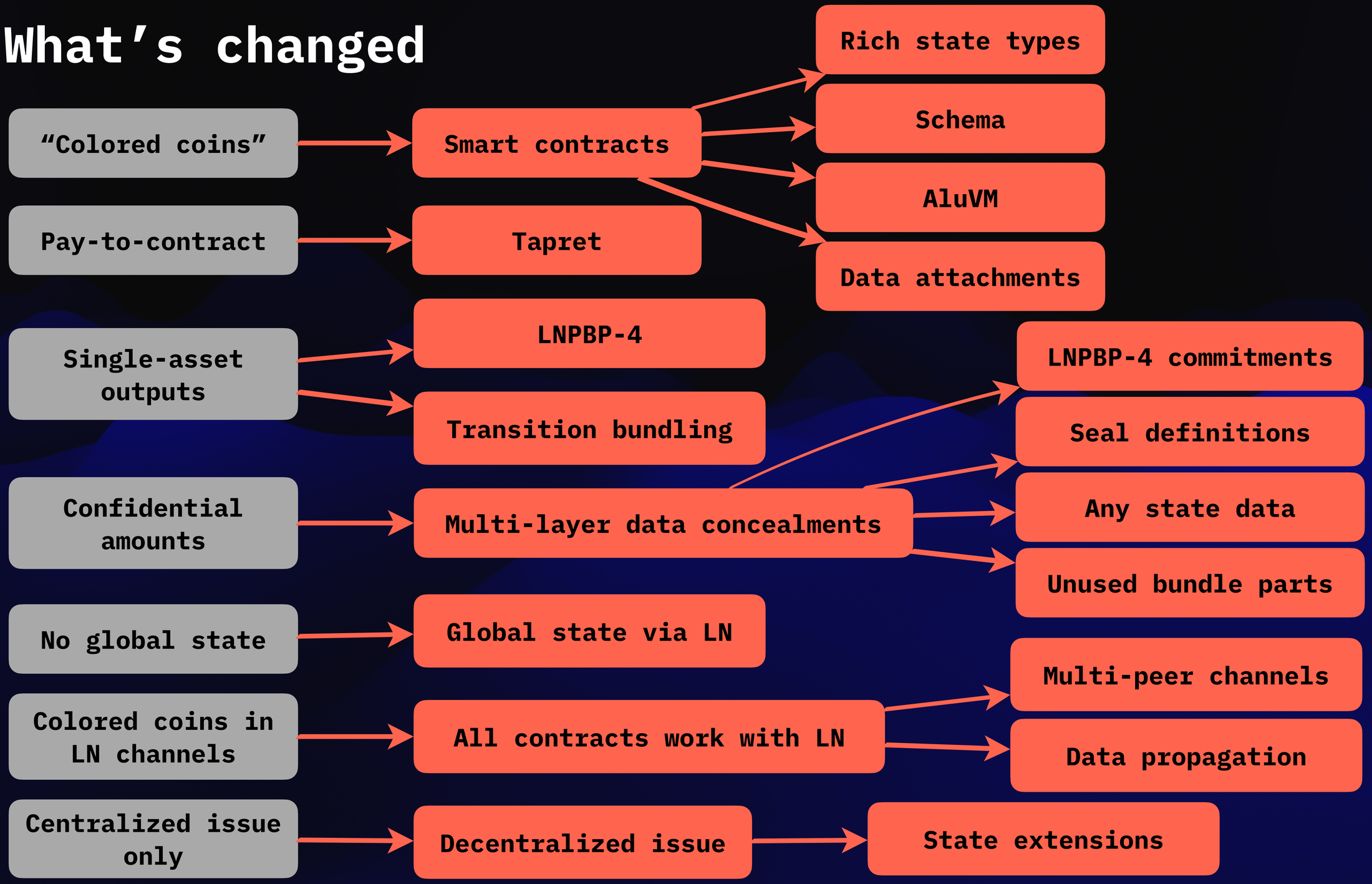
Forget everything you knew about “how RGB works” up to today

- In summer 2019 I started working on RGB early concept by Giacomo Zucco, Peter Todd and Alekos Filini with an aim to convert it into a fully-fledged smart contract system for bitcoin & LN

- This required refactoring and rethinking of each aspect of RGB throughout three years

- Today I am glad to present completely new RGB: confidential & scalable smart contracts for Bitcoin & LN

What’s changed



Compatibility

Taproot

Multisig wallets

Schnorr signatures

MuSig2

Hardware wallets

Miniscript

Atomic swaps

Adaptor signatures

Submarine swaps

Lightning Network

DLCs

CoinJoin

eltoo

Cross-input signature aggregation

PayJoin

SIGHASH_ANYPREVOUT

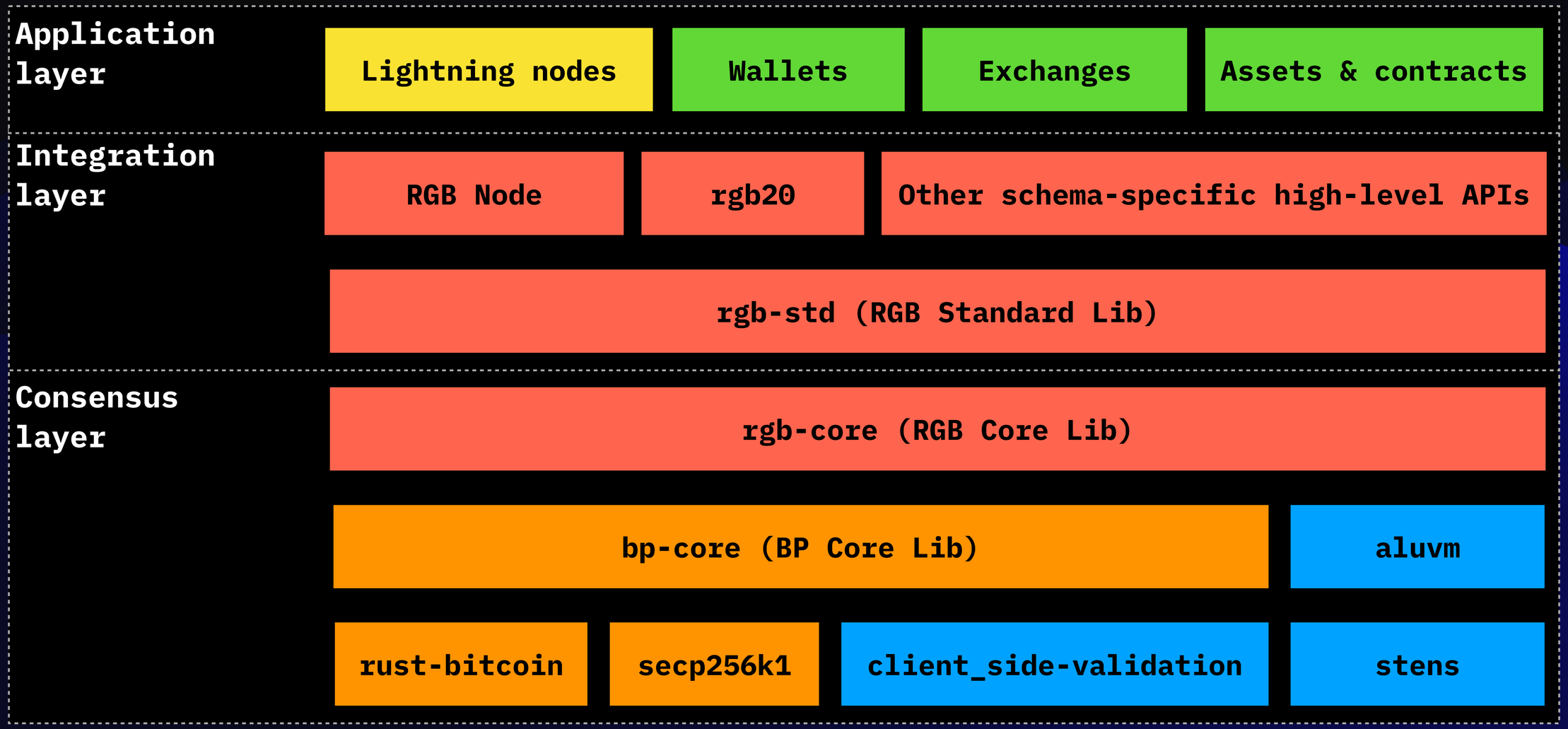
Cross-chain swaps

This all is available with RGB Core v0.7
released on the 13th of June

It’s not only about RGB!

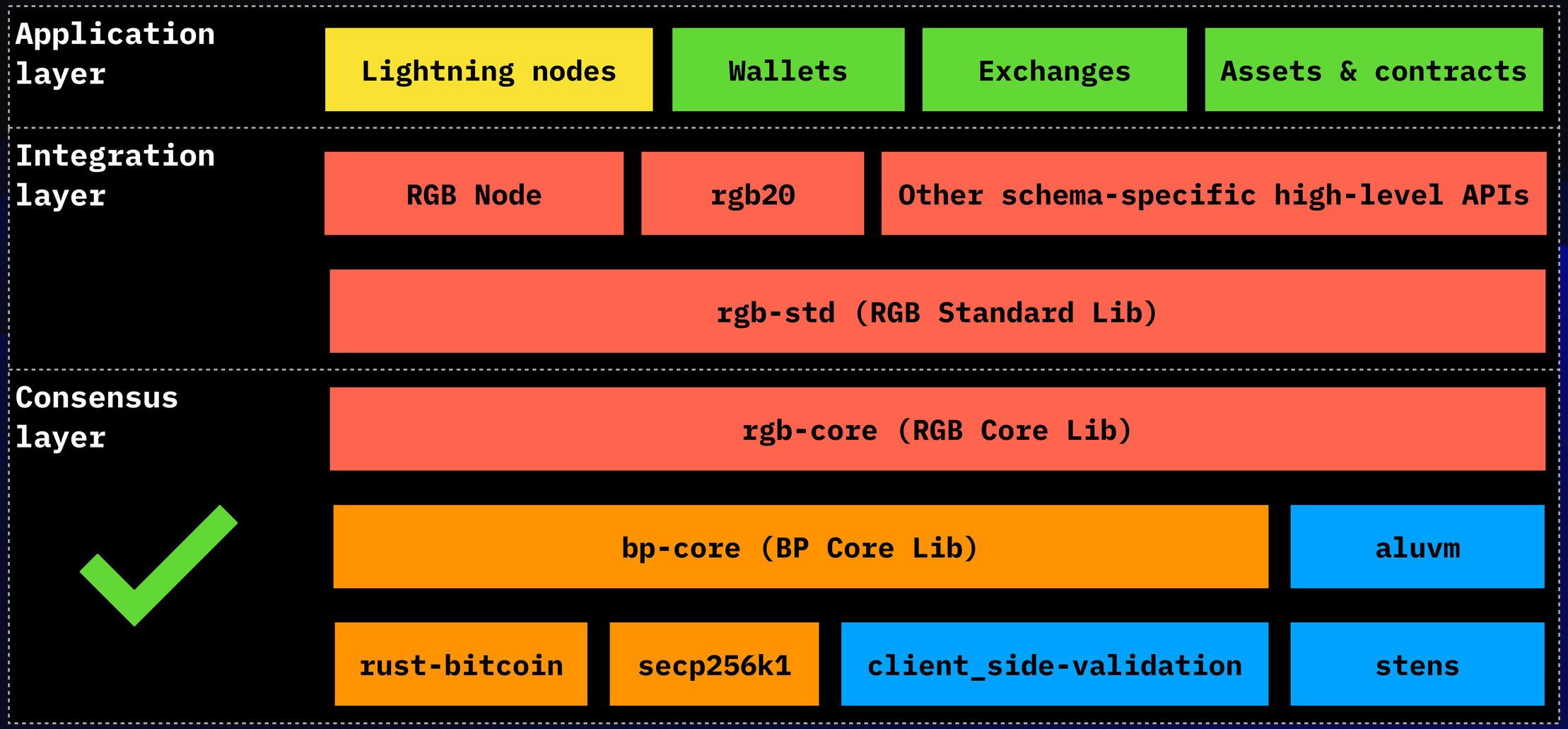
- **Bifrost**: Lightning network refactored
  - Taproot support
  - Arbitrary data messaging (see Storm below)
  - Multipeer channels & factories
  - Composable channels
  - DLC support
  - RGB support
  - Large data packets (from 64kb to up to 16MB)
- **Storm**: decentralized data propagation & storage

RGB Layers


Application layer
Lightning nodes
Wallets
Exchanges
Assets & contracts
Integration layer
RGB Node
rgb20
Other schema-specific high-level APIs
rgb-std (RGB Standard Lib)
Consensus layer
rgb-core (RGB Core Lib)
bp-core (BP Core Lib)
aluvm
rust-bitcoin
secp256k1
client_side-validation
stens

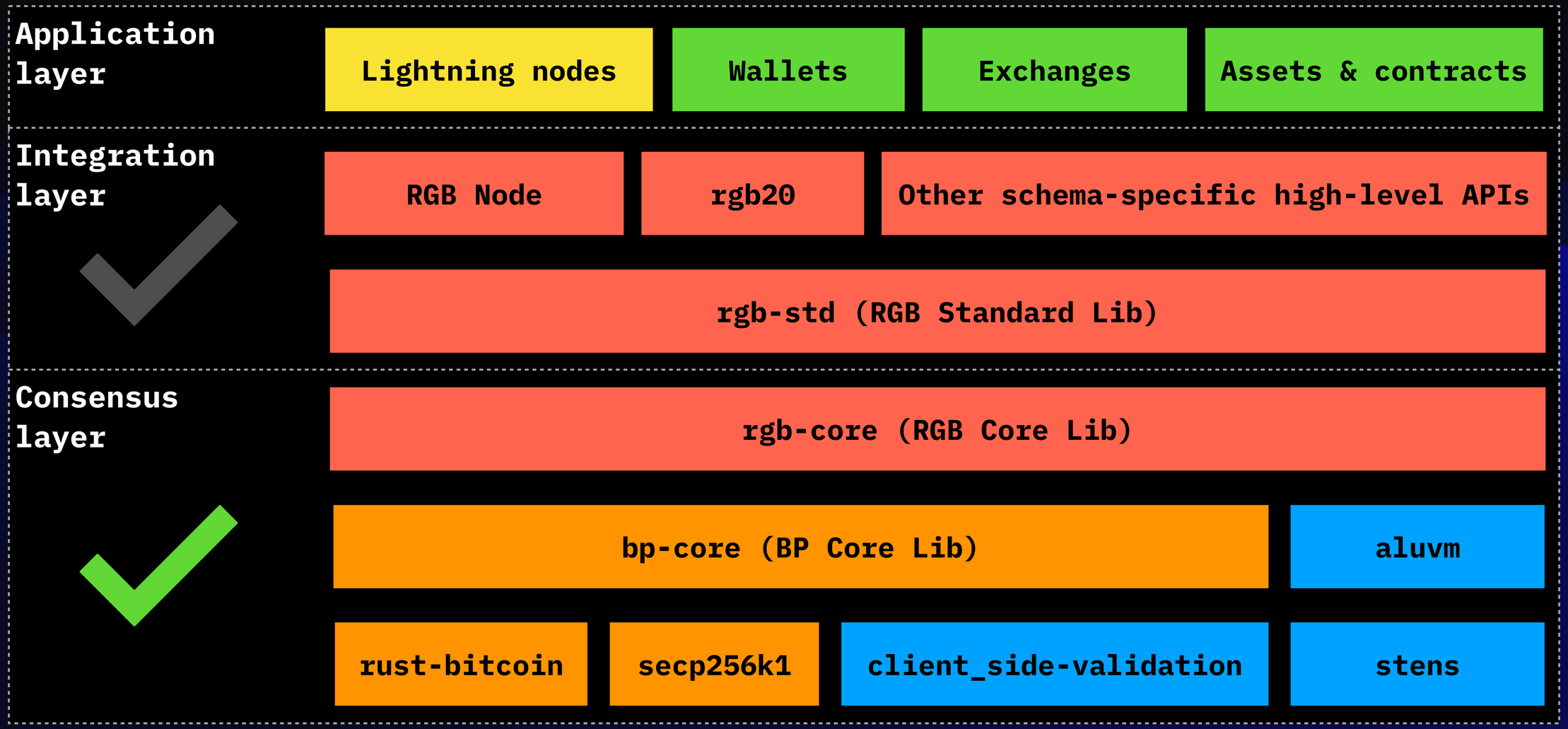
Since 13th June RGB consensus layer is freezed and accepts only bugfixes

RGB Layers


Application layer
Lightning nodes
Wallets
Exchanges
Assets & contracts
Integration layer
RGB Node
rgb20
Other schema-specific high-level APIs
rgb-std (RGB Standard Lib)
Consensus layer
rgb-core (RGB Core Lib)
bp-core (BP Core Lib)
aluvm
rust-bitcoin
secp256k1
client_side-validation
stens

Bifrost, Storm and **rust** wallet integration API comes with release v0.8 this week

RGB Layers


Application layer
Lightning nodes
Wallets
Exchanges
Assets & contracts
Integration layer
RGB Node
rgb20
Other schema-specific high-level APIs
rgb-std (RGB Standard Lib)
Consensus layer
rgb-core (RGB Core Lib)
bp-core (BP Core Lib)
aluvm
rust-bitcoin
secp256k1
client_side-validation
stens

RGB roadmap

	RGBv1 protocol release	RGB Node integrated with LN	RGB20: token contracts	RGB20 tokens in LN chanels	Real-world Assets on RGBs
2022:	13th Jun	1st Jul	13st Jul	1st Sep or earlier	?
	Consensus & validation rules	Command-line interface	Wallets & exchanges	Lightning payments	

When mainnet?

Buy RGB token!

The only way to get to the moon with RGB

LNP/BP Standards Association

Mainnet:

each asset is its own “chain”

LNP/BP Standards Association

RGB do not has any other network than a lightning network, where testnet and mainnet are together

LNP/BP Standards Association

It will be up to the issuers to decide when to issue assets on main net

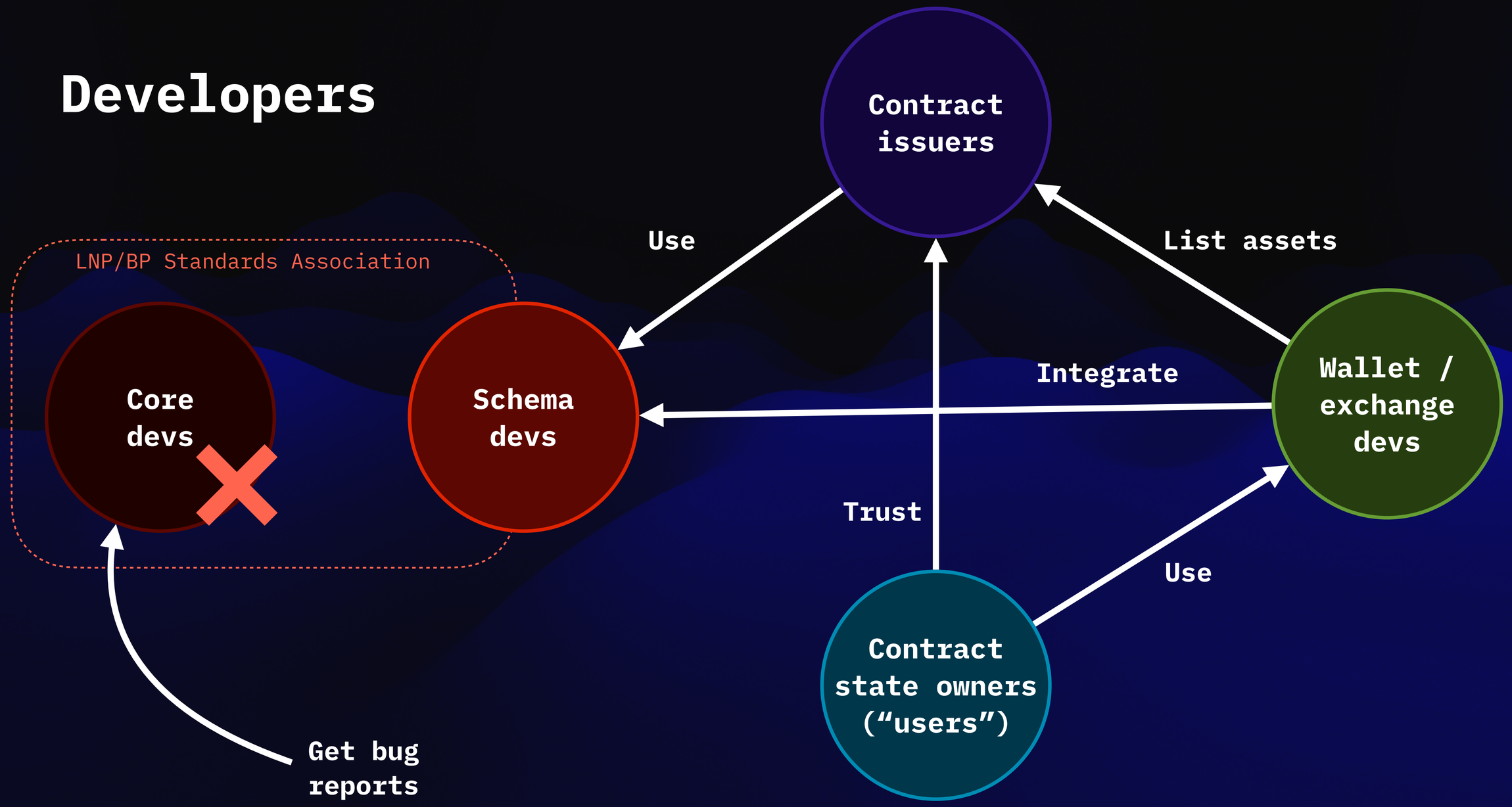
Remove hidden power of developers

Developed by non-profit Association LNP/BP Standards Association

RGB not be changeable after “take off”

Gives power back to contract/asset issuers, bringing fair play back

Developers


Contract issuers
Use
List assets
LNP/BP Standards Association
Core devs
Schema devs
Integrate
Wallet / exchange devs
Trust
Use
Contract state owners (“users”)
Get bug reports

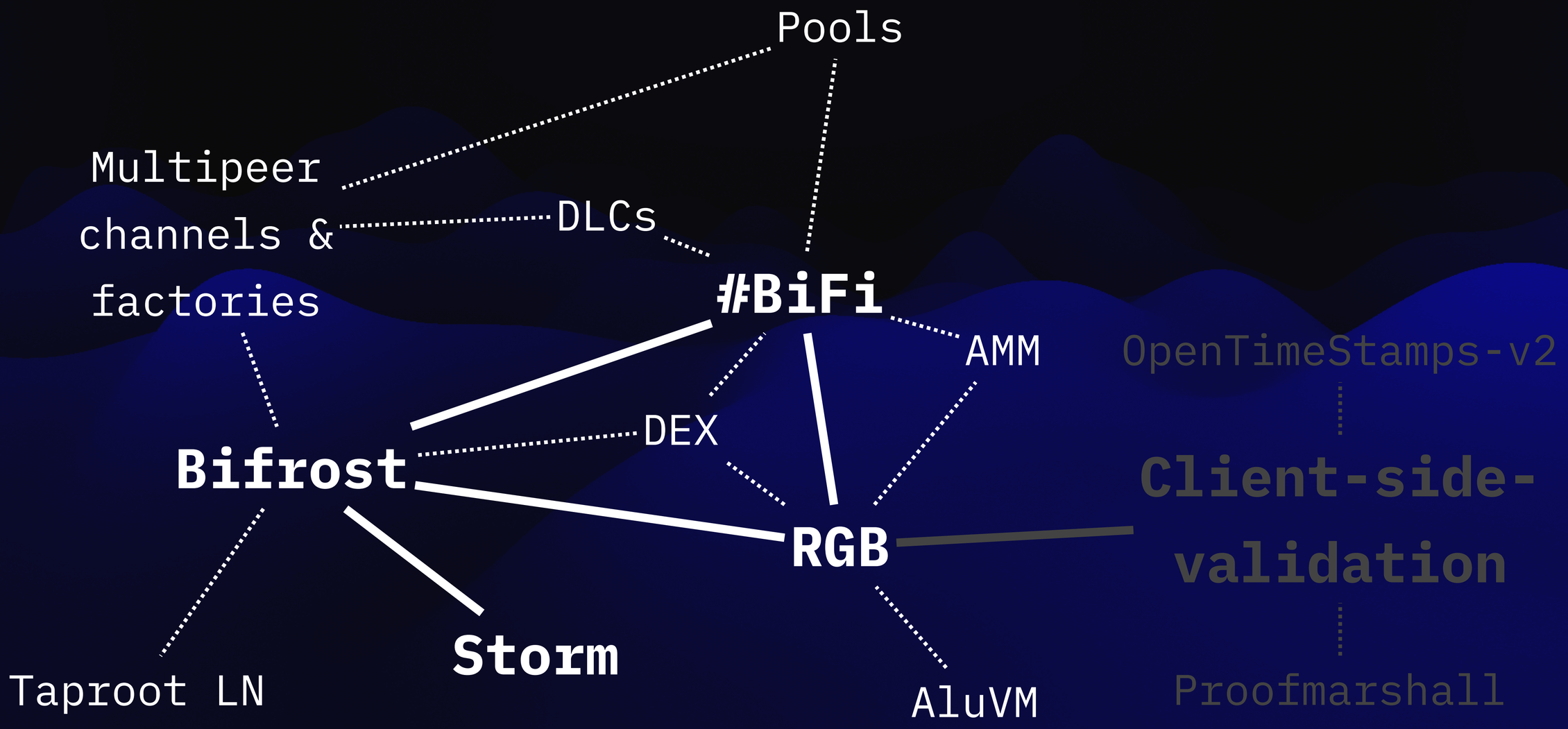
LNP/BP Standards Association in RGB context

- One-time activities:
  - **Designed RGB** as a smart contracting system
  - **Implemented RGB consensus layer** (RGB Core, client-side-validation libs, AluVM)
  - Responsible for the **ossification of RGB protocol** by making changes to the above impossible
  - Developed reference **RGB Node, LNP Node, Storm Node**

LNP/BP Standards Association in RGB context

- One-time activities:
  - ...
- Ongoing efforts:
  - Develop & standardize independent **RGB schemata** (RGB-20 etc)
  - Design & develop tools for **AluVM programming** (languages, toolchains etc)
  - Design & do reference implementations of **#BiFi** protocols based or enabling RGB smart contracts: Bifrost, DEX (Kaleidoscope), Storm (decentralized data) etc
  - Promote and educate RGB (if budget is sufficient)

“Constellation” of LNP/BP Association work

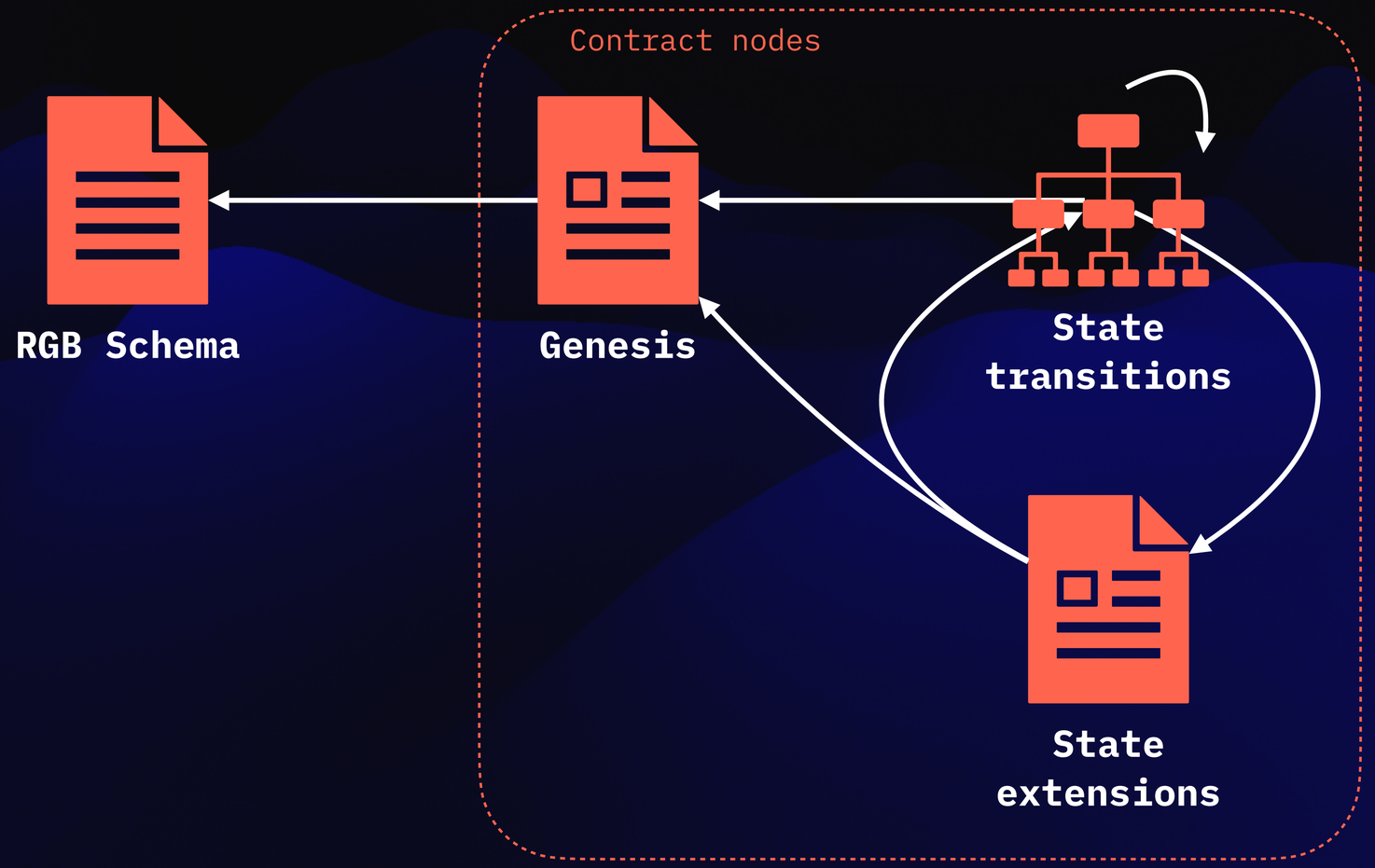


LNP/BP Standards Association will not

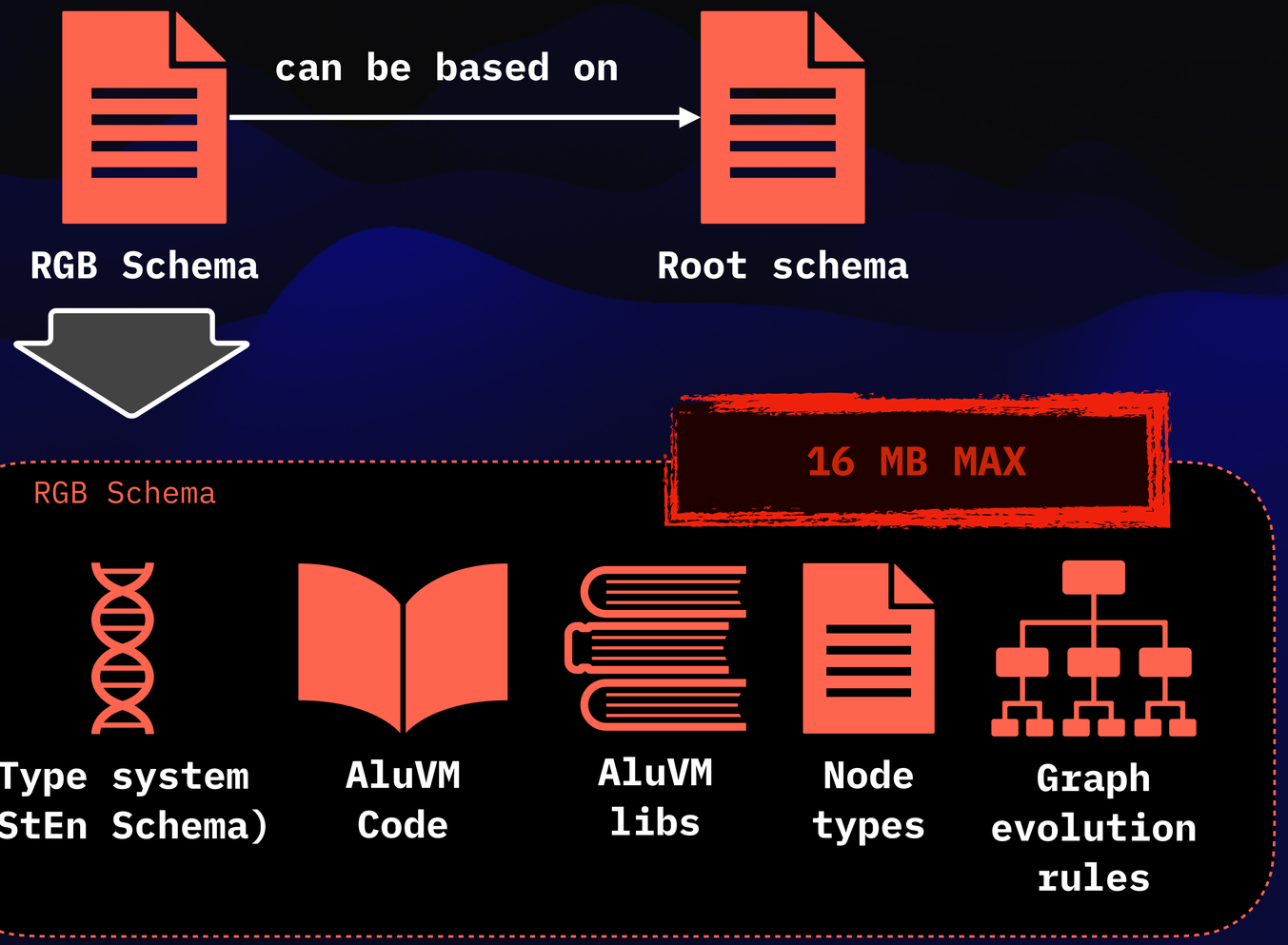
- **Introduce or discuss any changes to RGB consensus layer** outside of security-critical vulnerabilities fixes
- Be able to operate without budget, i.e.
  - We plan fundraising for the association activities this summer
  - We plan to charge for help to businesses, asset issuers etc and use this funds to support Association activities

How RGB smart contracts work

Anatomy of RGB contract



Anatomy of RGB schema

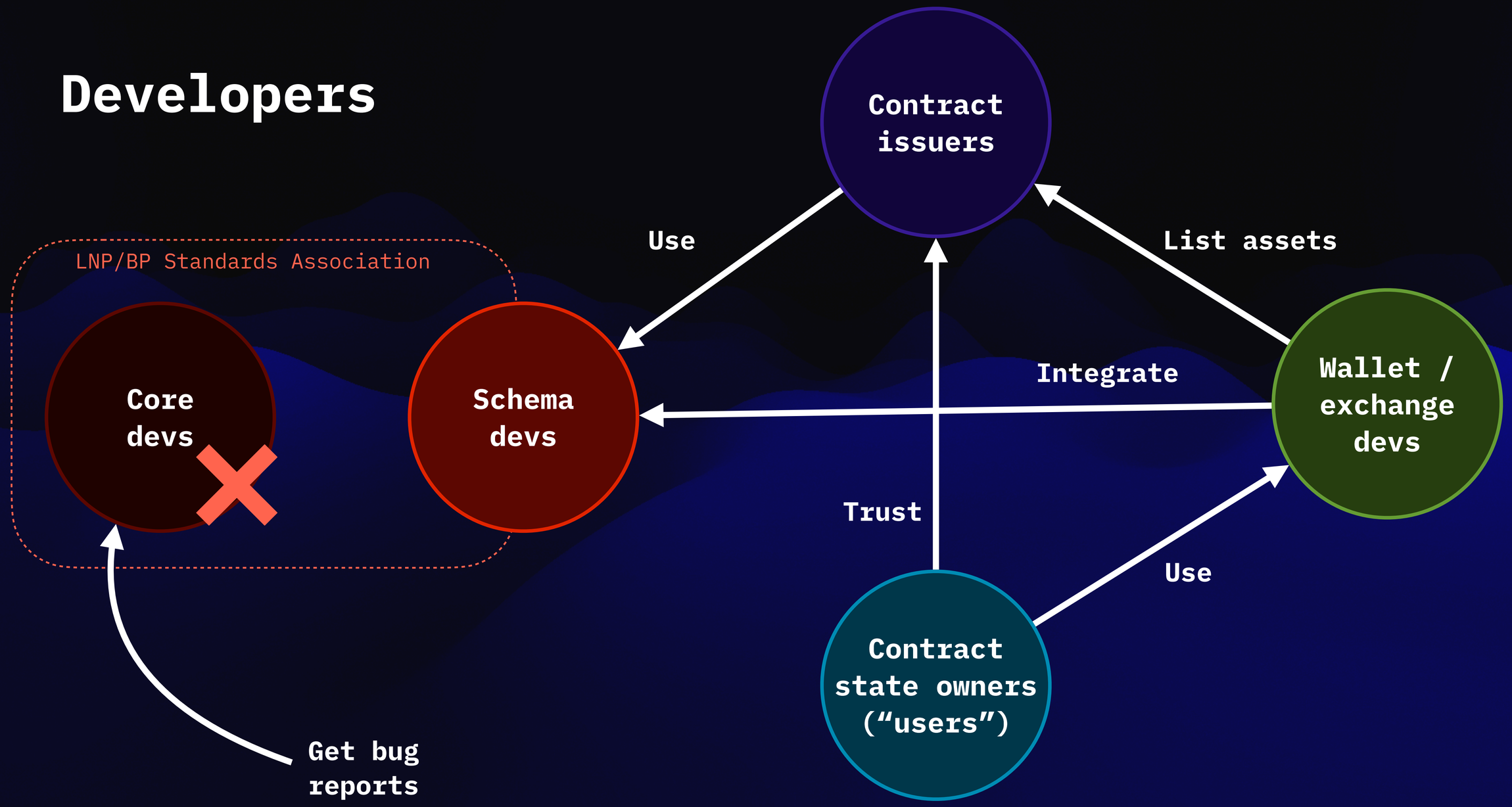


- Created by **schema developers**
  - Today LNP/BP Standards Association
  - Tomorrow anyone can do this
- Require programming AluVM
- Contains rules for
  - which state the contract has
  - contract state validation
  - how the contract DAG can evolve (“cellular automata” rules program)

Root schema

- Needed for wallets to be able to recognize contract data of certain class
- Just one level of hierarchy
- Child schemata can be only a strict subset of a root schema (for instance, specific asset type under generic asset parent schema)

Developers


Contract issuers
Use
List assets
LNP/BP Standards Association
Core devs
Schema devs
Integrate
Wallet / exchange devs
Trust
Use
Contract state owners (“users”)
Get bug reports

Type System (strict encoding schema)

Primitive types

- Integers
  - Signed
  - Unsigned
  - 8-to-1024 bit range
- Floats
  - Up to 256 bits
  - Machine-learning F16B type
  - Tappered float type
- ASCII character
- Unicode character

Collections

- Fixed-size arrays (good for hashes etc)
- Variable-size arrays (lists), including
  - Byte strings
  - Ascii strings
  - Unicode strings
- Variable-size unique arrays (sets)
- Variable-size key-value maps

Structured types

- Named types
- Made of fields using
  - Primitives
  - Collections
  - Other structured type

Type System (strict encoding schema)

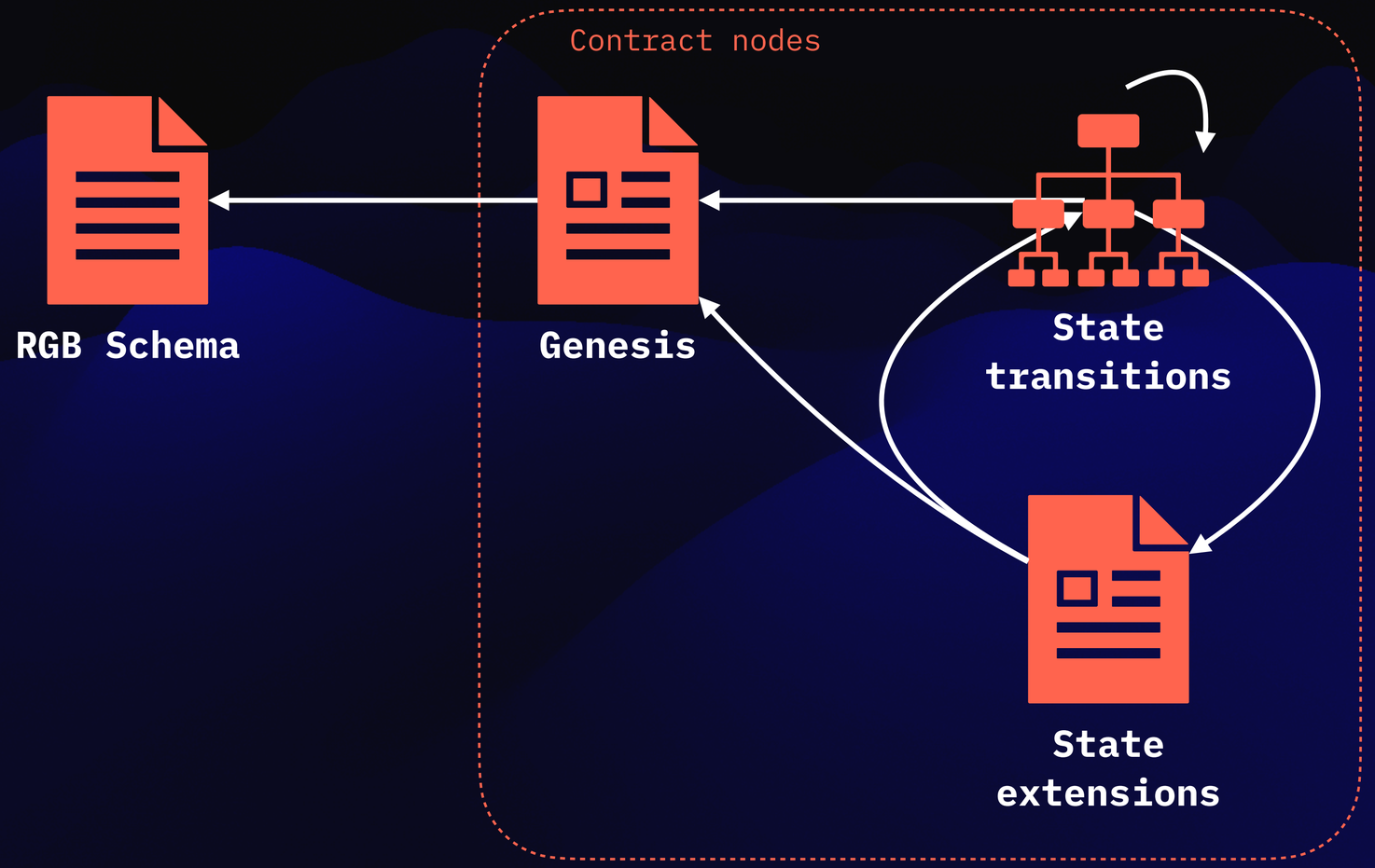
Structured type

Primitive type

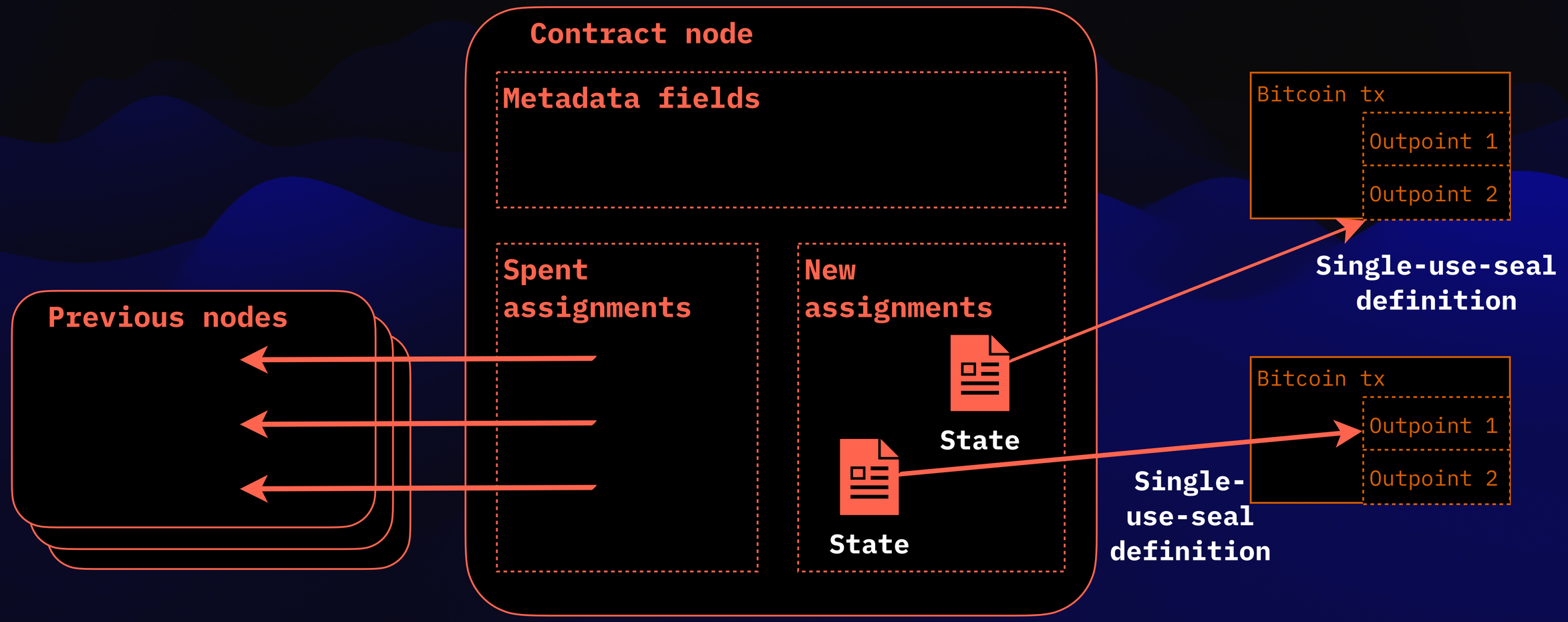
```
143     type_system! {
144         "OutPoint" :: {
145             StructField::new("Txid"),
146             StructField::primitive(PrimitiveType::U16),
147         },
148         "Txid" :: { StructField::array(PrimitiveType::U8, 32) }
149     }
```

Collection

Anatomy of RGB contract



Anatomy of RGB contract node



Owned state types

	Owned right	Owned value	Owned data	Owned attachment
Data:	None	64-bit signed or unsigned integer	Any rich data type	Any file
Type info:	None	Signed/ unsigned	Strict encoding schema	MIME type
Confidentiality:	Not required	Pedersen commitment + bulletproof	Hashing with blinding	Hashed file id
Size limits:	N/A	256 bytes, 64kb for proof	Up to 64kB	Up to ~500 GB

Single-use-seal definitions

Tagged hasing

txid | vout | blinding | method → txob1…

	txid	vout	blinding	method	txob1…
		32 bit	64 bit	1st Tapret 1st Opret	
Normal	±	✓	✓	✓	
Confidential					✓
Endseal		✓		✓	✓
Explicit	✓	✓		✓	

AluVM

Turing-complete virtual machine for RGB smart contracts

RGB client-side validation

Only restricts bitcoin script-based rules, not extends them.
i.e. you can spend bitcoin output, but get invalid RGB state (loose asset, for instance by trying to inflate its supply)

- **Bitcoin script** controls who owns (who can change the state)
- **RGB schema** defines structure of the contract:
  - Data types, types of ownership rights, state types
  - Possible construction of contract DAG
- **AluVM** performs dynamic checks on state changes during state transitions

What AluVM provides to RGB

- Access to the **RGB smart contract state** (RGB ISA extension)
  - Access to other RGB smart contracts (known to developers) - mechanism for RGB composability
- Access to **bitcoin blockchain data** (BP ISA extension)
- Access to **data from the Lightning network** (LNP ISA extension)
- Access to **oracles based on adaptor signatures** (SECP and DLC ISA extension)

With RGB BIP-119 for covenants is not needed!

Today the code looks this way

```
                read s[1], $input          ; AluRE ISA extension opcode to read
                                           ; user/dynamic data with the given $id
                read r256[1], $difficulty
                read a16[2], $cycle_limit
                mov  a16[1], 0             ; putting a value into register
loop:                                      ; label for cycle
                sha2 s[1], r256[2]         ; taking hash of the data
                extr r256[2], s[1]         ; saving hash result
                inc  a16[1]                ; counting steps
                ge   a16[1], a16[2]        ; making sure we do not exceed $cycle_limit
                jif  exceeded
                le   r256[2], r256[1]      ; checking against difficulty
                jif  done                  ; target difficulty reached!
                jmp  loop
done:           succ
exceeded:       fail                       ; failing since we exceeded $cycle_limit
```

Tomorrow: Contractum

```
schema RGB20Simple using Assets
    field Ticker :: UTF8^3..8
    field Name :: UTF8^8..32
    field ContractText :: UTF8*
    field DecFractions :: U8 <- precision U8
        -- here we must validate the value range,
        -- so we are taking U8 value with `<-`
        -- and making sure it fits into our requirements
        assert precision in 0..=18
    field IssuedAmount :: AssetAmount

    -- if this was NFT, we can add something like
    -- container Painting :: image/png

    homomorph AssignedAmount AssetAmount
    right Renomination
    -- hashed accumulating Engraving -- used in NFTs

    genesis :: Ticker, Name, ContractText?, IssuedAmount, allocation AssignedAmount*, Renomination?
        assert sum! *allocation == issuedAmount

    transition transfer :: spent AssignedAmount+ -> sent AssignedAmount+
        assert sum! #spent == sum #sent

    transition renominate :: Renomination -> Renomination?, Ticker?, Name?, ContractText?
        assert ticker? or name? or contractText?

    method totalIssued :: () -> AssetAmount
        return @self.issuedAmount
```

RGB smart contract computation model (PRISM)

- **Distributed** (state is distributed across multiple actors)
  - *Fully asynchronous*
  - Network-*fault tolerant*
  - achieves *consensus* by utilizing special form of “finality proofs” (PoW-UTXO-1stTapret/Opret single-use-seal consensus)
- **Partially-replicated** – **PR**ism
  (each participant/actor has a partial view of smart contract state;
  the whole state is a DAG)
- **Indeterministic** (affected by participants’ decisions and PoW randomness) – pr**I**sm
- **State machine** (equivalent* to universal Turing machine) – pri**SM**

Q Is there a “global state” for the contract?

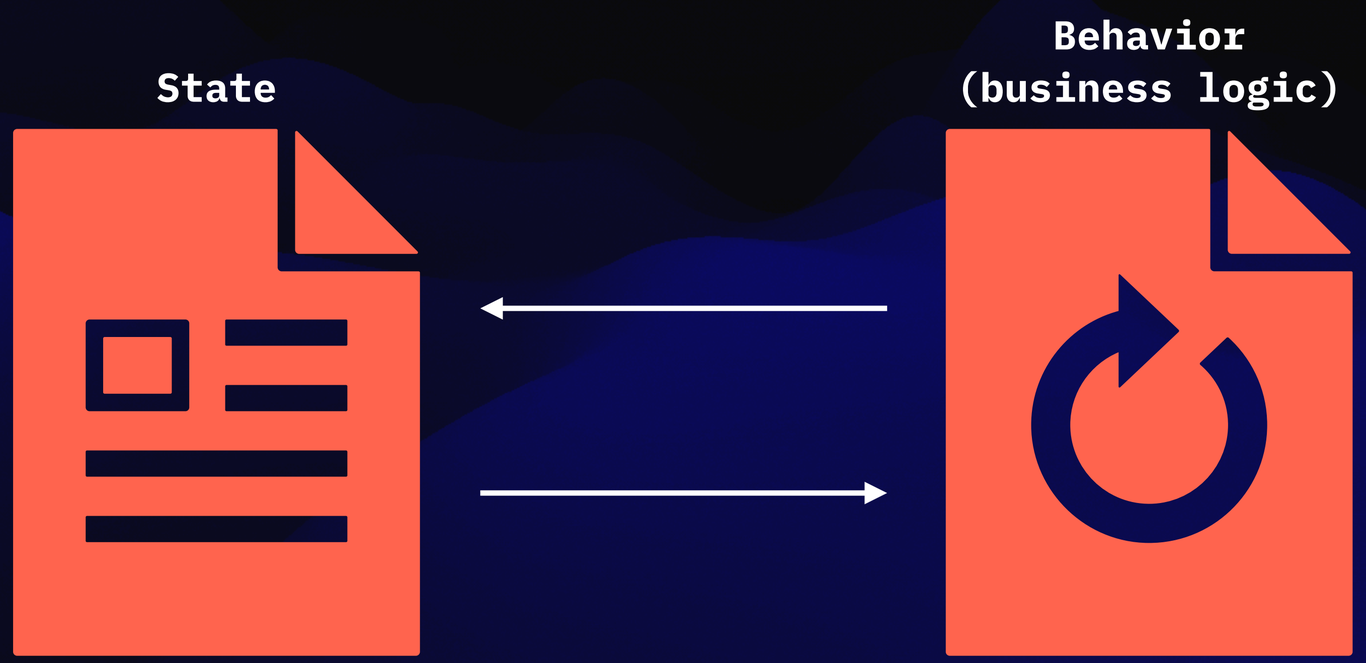
Yes, but in very special forms:

- Centralized, created by contract issuers
  - Distributed over Lightning network (Bifrost+Storm)
  - As a part of the asset information
  - It is a part of the consensus
- Decentralized
  - Created by contract users (if allowed by the schema)
  - Published to Storm decentralized storage
  - Not part of the consensus, but uses special game theoretic setup to arrange incentives for its persistence and availability with Storm

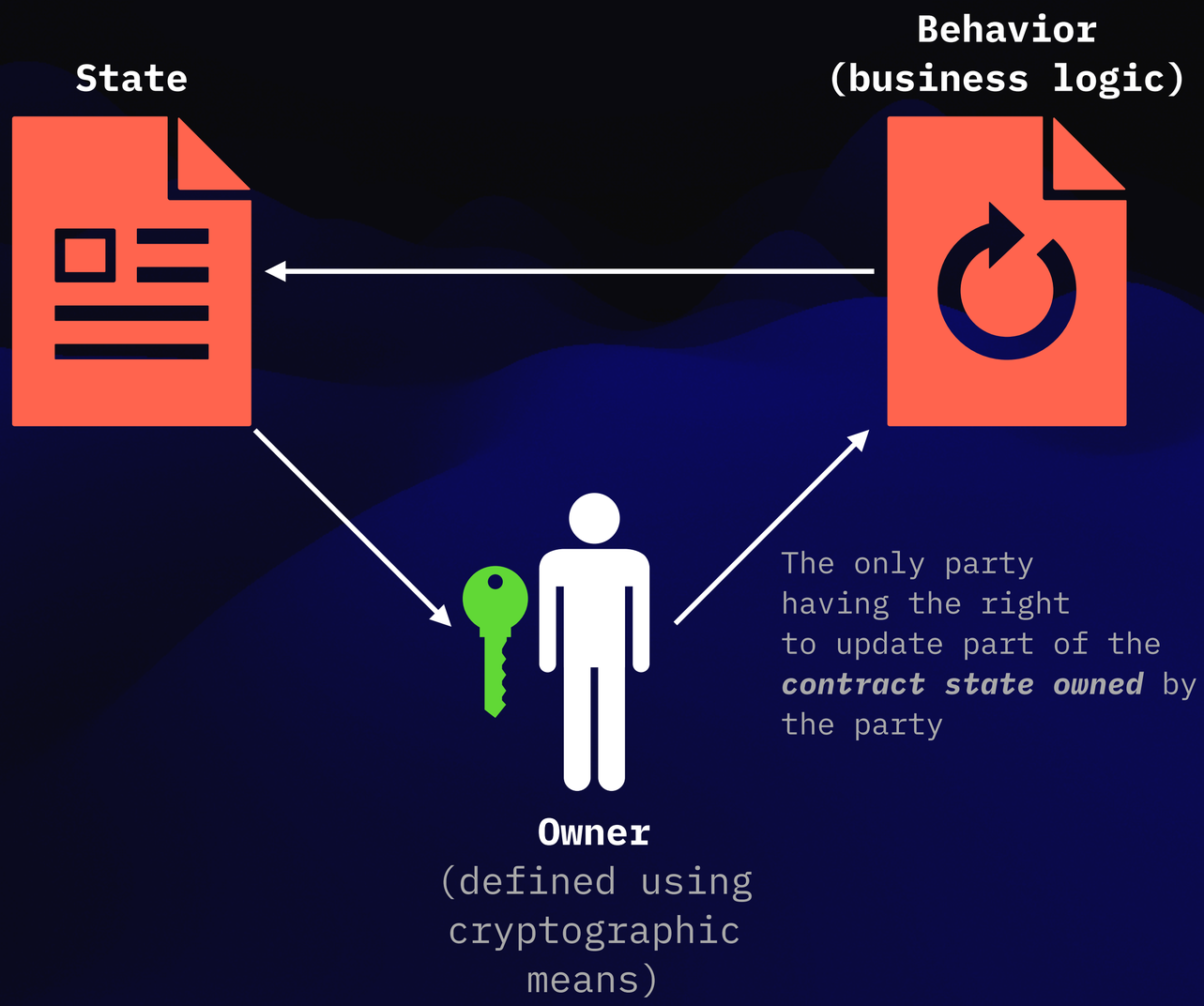
Q How I can “read contract” or “call contract method”?

- Yes, but this is not a part of RGB consensus layer

Smart contract



RGB smart contracts: owned state

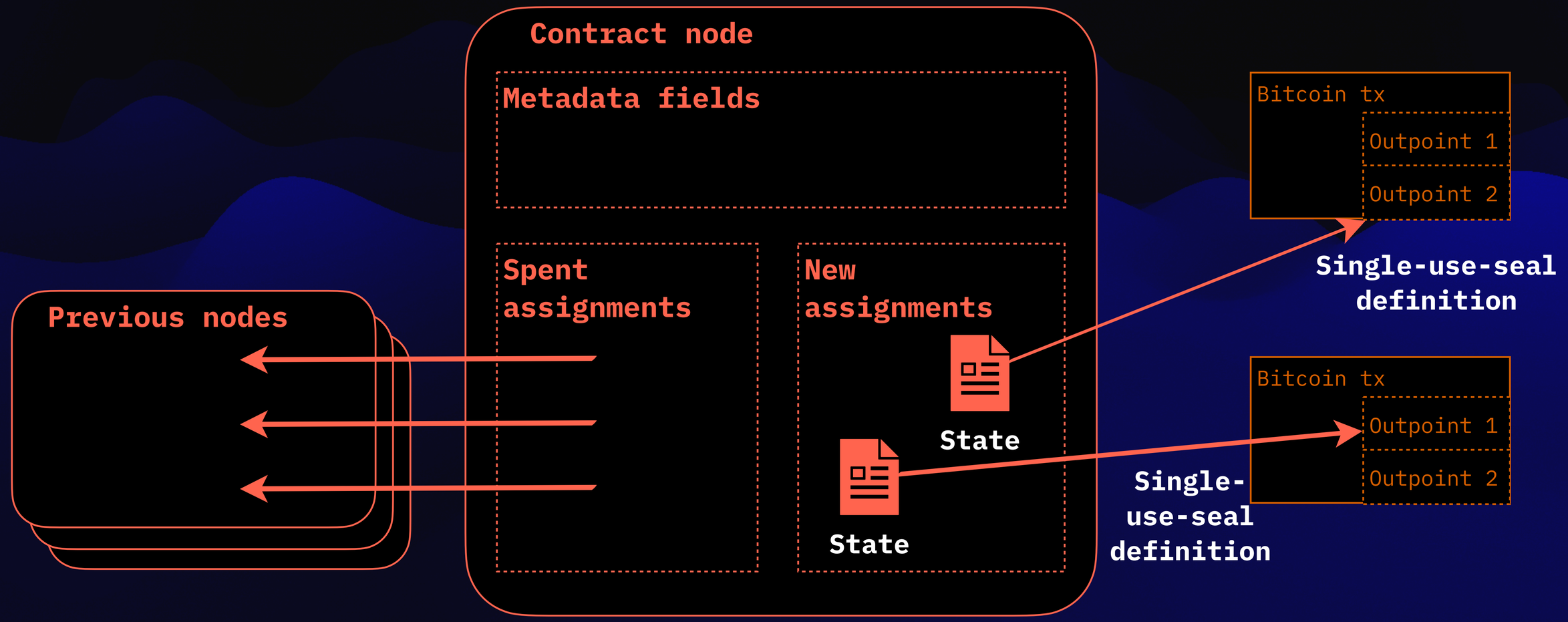


Q How I can “read contract” or “call contract method”?

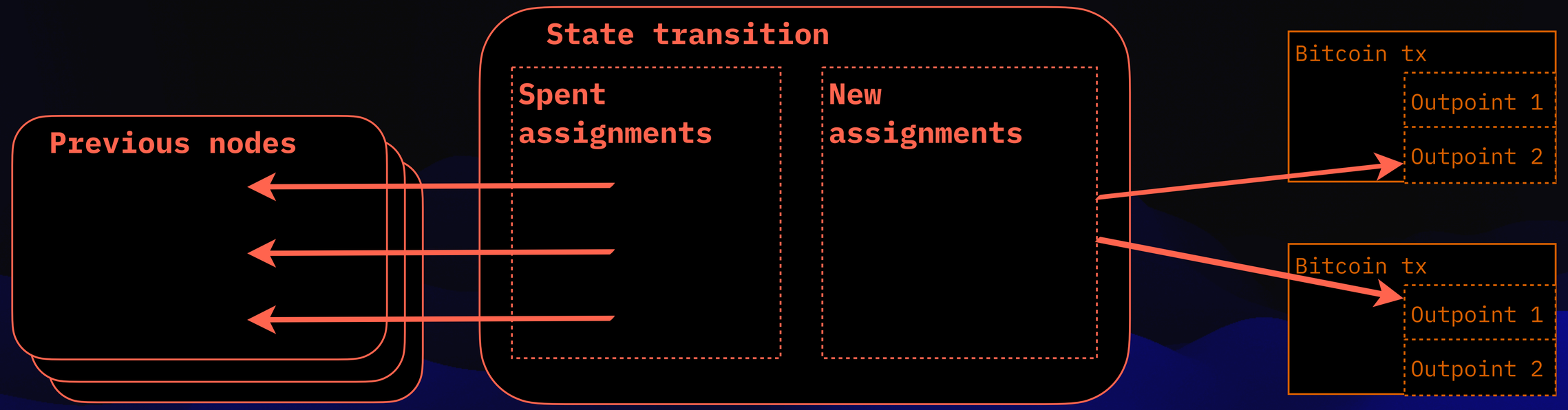
- Yes, but this is not a part of RGB consensus layer
- Contract/schema developers _may_ provide signed “contract API” code in form of AluVM script which can be used by RGB Node to
  - Return parsed part of the contract state
  - Prepare new specialised state transition
- This code does not affects RGB contract validation and is not part of the consensus
- Thus, unlike in Ethereum, it can be changed without contract change
- It is up to the contract users to decide which version of the code to use

Contract Containers

Anatomy of RGB contract node

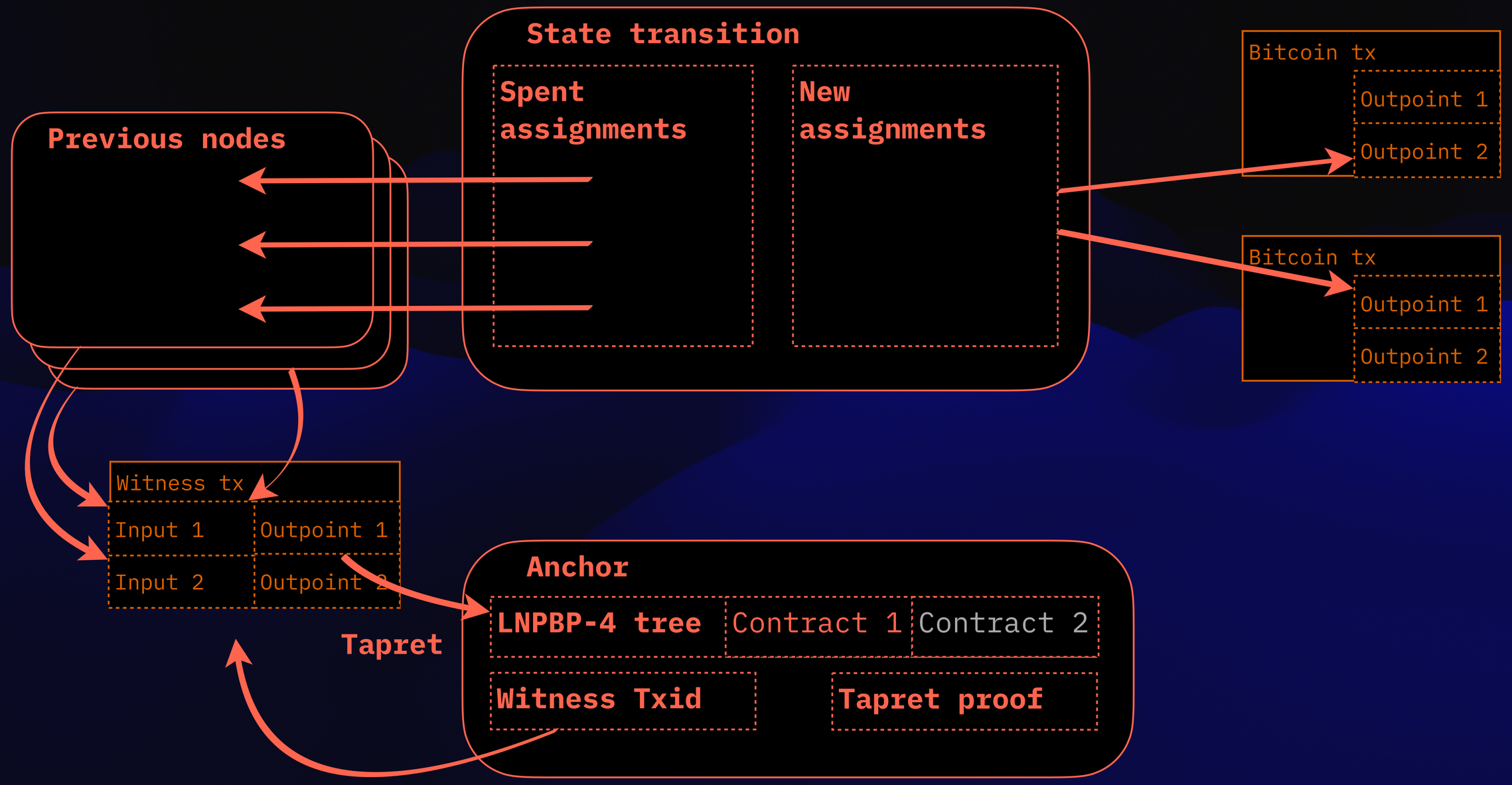


Anatomy of RGB anchored bundle

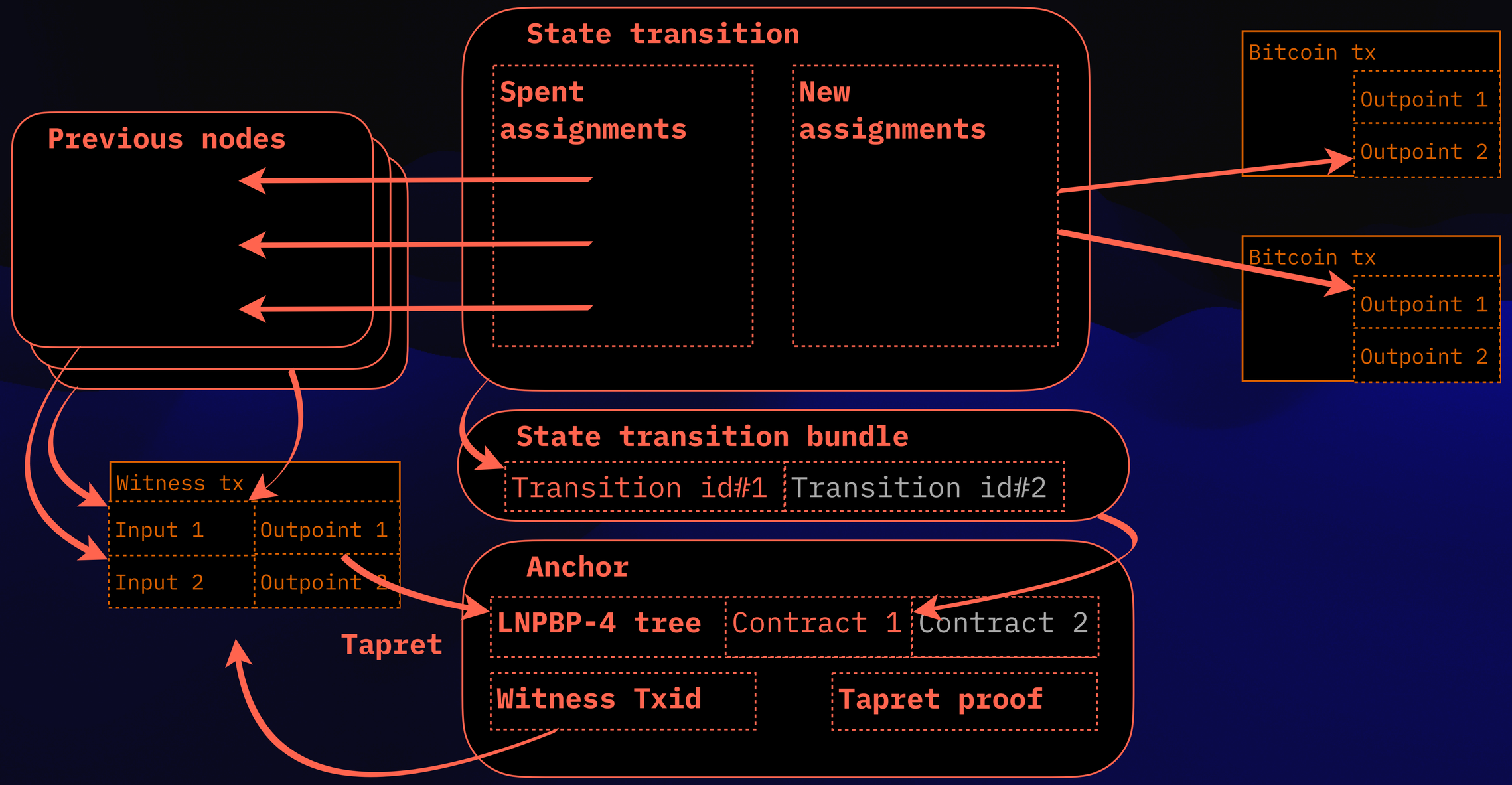


Anatomy of RGB anchored bundle

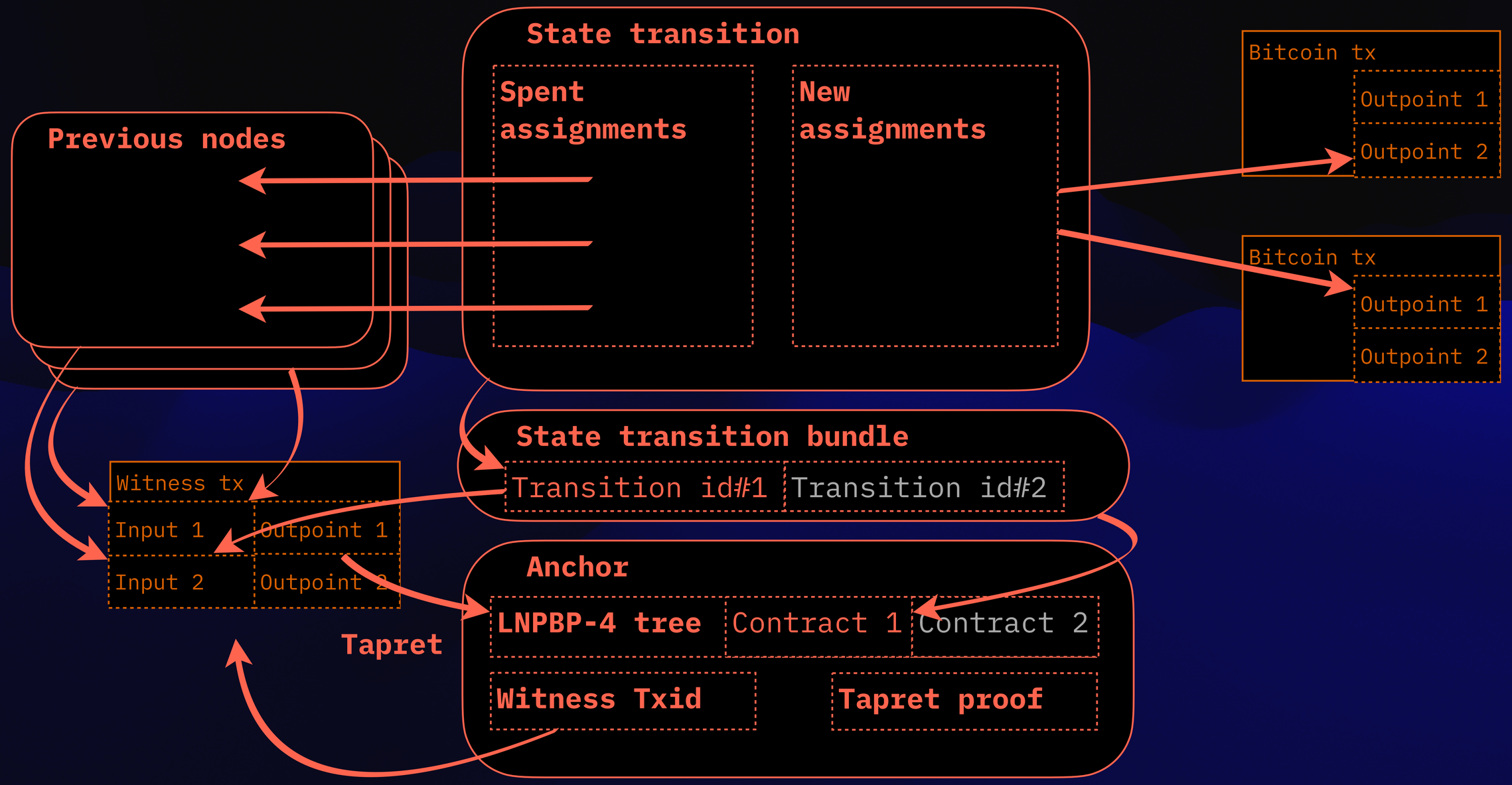
Anatomy of RGB anchored bundle



Anatomy of RGB anchored bundle



Anatomy of RGB anchored bundle



Terminology

- **Witness transaction:** transaction closing a (set of) seals, committing to a message via Tapret + LNPBP-4 commitment
- **Anchor:** client-side-validated (non-RGB-specific) data structure linking offchain data under multiple unrelated protocols to the chain
- **Transition bundle:** RGB-specific packing of a set of state transitions related to a single witness transaction
- **State transition:** change in RGB contract state. Closes seals, previously defined by other contract nodes; defines a set of new allocations
- **Seal definition:** pointer to bitcoin transaction output with additional data
- **Assignment:** state data assigned to a certain seal definition

RGB containers

Consignments

- Used in:
  - Distributing contract information (assets etc) – **Contracts**
  - Transferring ownership of contract assets / state – **State transfers**
- Creating consignment from stash data is called `**consign**` procedure
- Merging consignment into stash is called `**consume**`

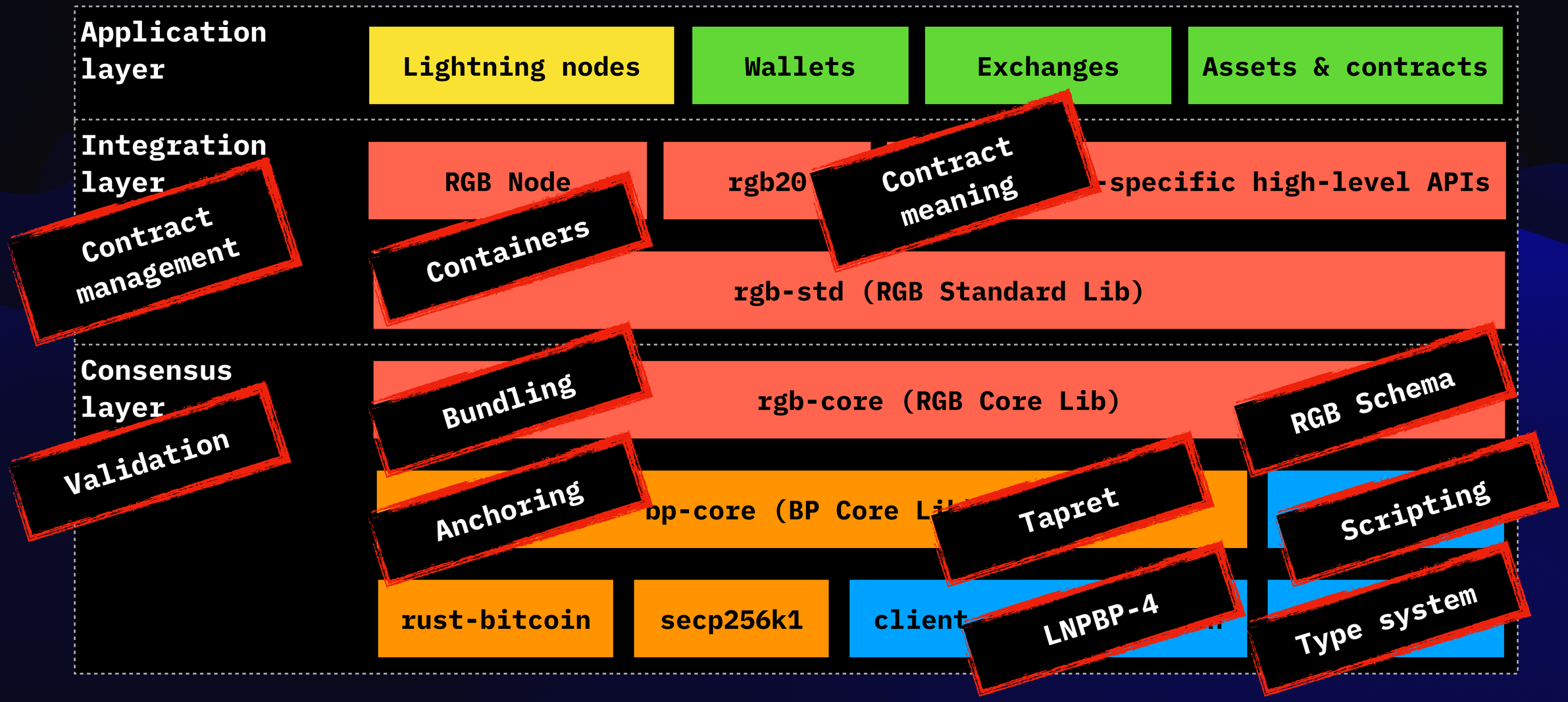
Disclosure

- Used
  - during transfers to maintain information about the change of contracts internally
  - for voluntary public disclosure of the information (proving certain facts)
- Creating disclosure from stash data is called `**disclose**` procedure
- Accepting disclosure into stash is called `**enclose**` procedure

Terminology recap

- **RGB container:** large piece of RGB client-side-validated data
- **Stash:** all client-side data related to a specific wallet descriptor (“one stash per each wallet, one stash per each LN node” rule)
- **Consignment:** single-contract transfer of information between users
  - **Contract:** information about RGB contract distributed by the issuers
  - **State transfer:** passage of RGB assets & state between contract users (from previous to a new owner)
- **Disclosure:** multi-contract information fragment revealing certain partial knowledge of RGB contracts

RGB Layers


Application layer
Lightning nodes
Wallets
Exchanges
Assets & contracts
Integration layer
RGB Node
rgb20
-specific high-level APIs
rgb-std (RGB Standard Lib)
Consensus layer
rgb-core (RGB Core Lib)
bp-core (BP Core Li
rust-bitcoin
secp256k1
client
Contract management
Containers
Contract meaning
Bundling
Validation
Anchoring
Tapret
LNPBP-4
RGB Schema
Scripting
Type system

The only task of RGB node is to manage and deal with RGB containers

RGB Node knows nothing about the meaning of contracts it operates with:

it can’t distinguish RGB20 fungible from NFT or other form of contract,
it knows nothing about assets etc

To get RGB working one needs **RGB node** integrated with **wallet**, **LN node**, **database** and (desirable) **Storm network**

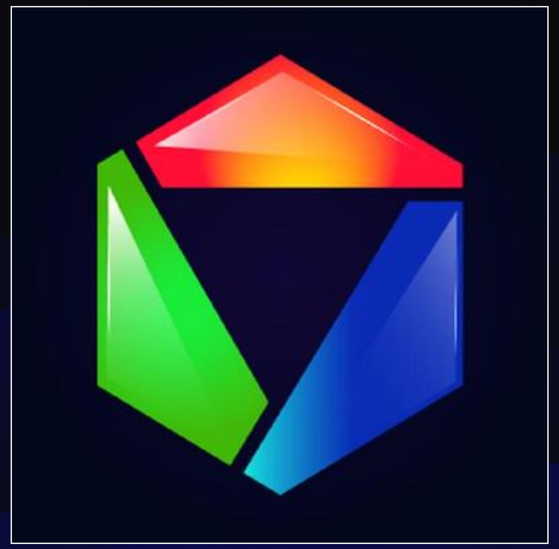
Storm network

- Operates on top of Lightning network (a layer on top of Bifrost subnetwork)
- **Data propagation, search and storage layer** blind to the specific data used in it
- Run by **Storm node** from LNP/BP Association, integrated with LNP Node
- Works with RGB via special RGB storm app - a part of RGB Node
- Used to propagate RGB Contracts and State transfers between issuers and users

- **Wallet defines scope of RGB stash** by providing wallet descriptor and filtering state related to it
- **Wallet adds meaning to RGB contracts** and data returned by RGB Node, using schema-specific libraries created by RGB schema devs (RGB20, NFT etc)
- **Wallet prepares PSBTs** and orchestrates overall state transfer process
- Storm node can be used with LNP Node to
  - Get information about existing assets
  - Set transfers to other users

  (this can be done manually, using custom centralized servers)

LNP/BP Nodes



RGB Node

Smart contract validation & state RPC

BP Node

Bitcoin indexing node (faster & more efficient than Electrum server)

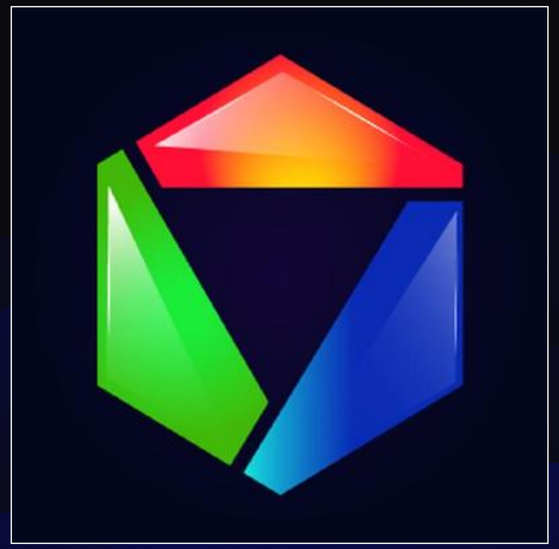
LNP Node

Lightning node supporting RGB, Taproot, DEX, BiFi

Storm Node

Decentralized storage, messaging & search

LNP/BP Nodes: v0.8 released this week



RGB Node

Smart contract validation & state RPC

BP Node

Bitcoin indexing node (faster & more efficient than Electrum server)

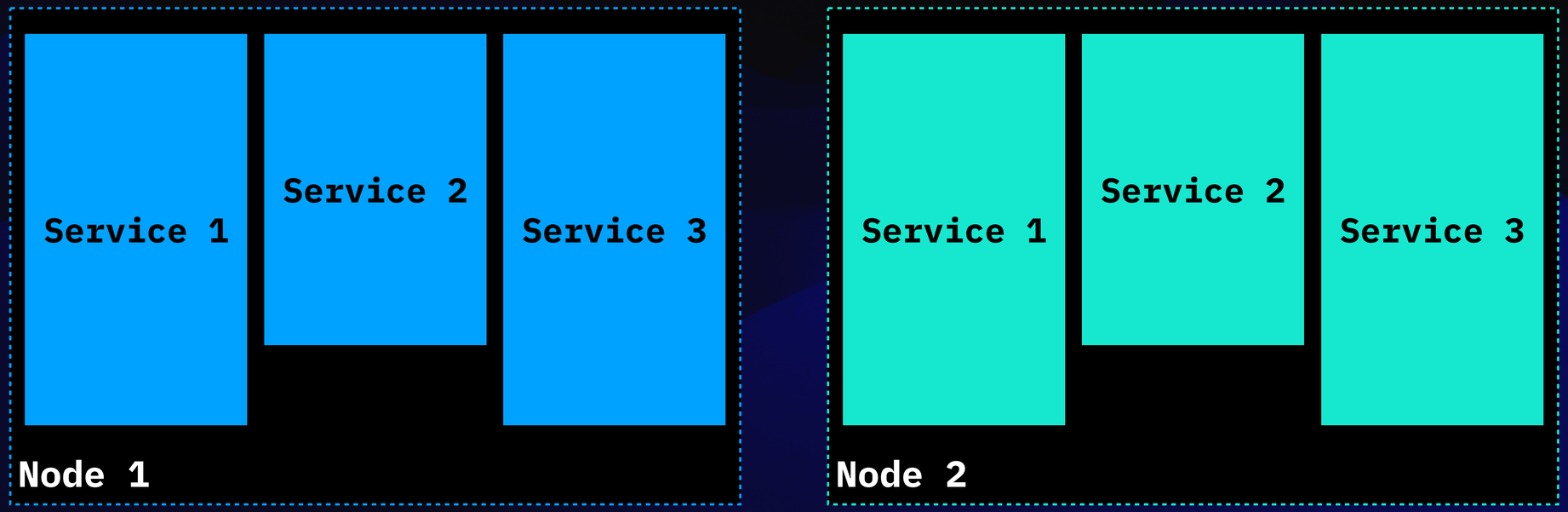
LNP Node

Lightning node supporting RGB transfers

Storm Node

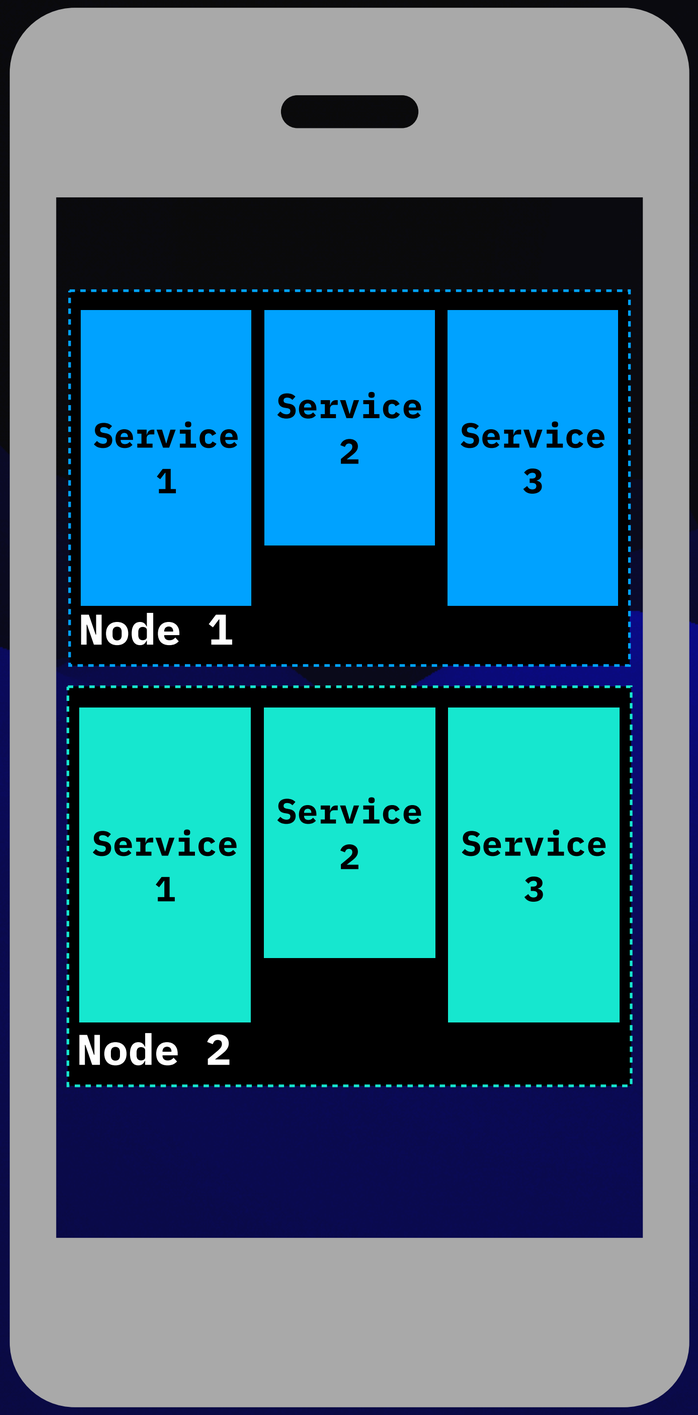
Decentralized data transfer

Node architecture



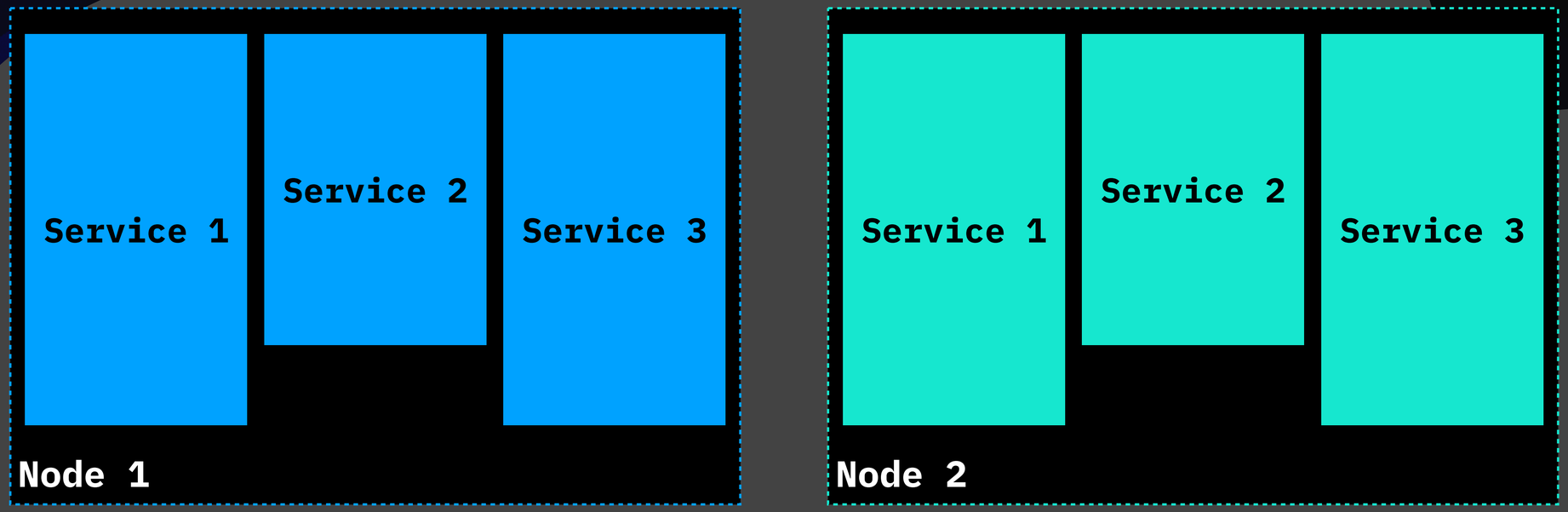
Each node is a set of microservices

Node architecture



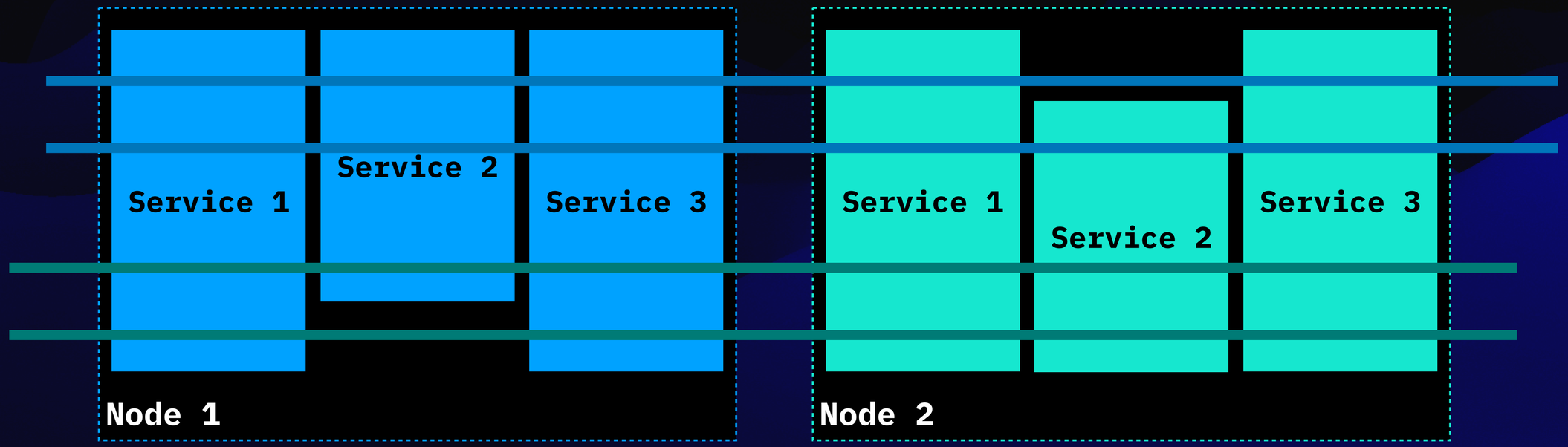
Which can be all-together embedded into a mobile app

Node architecture



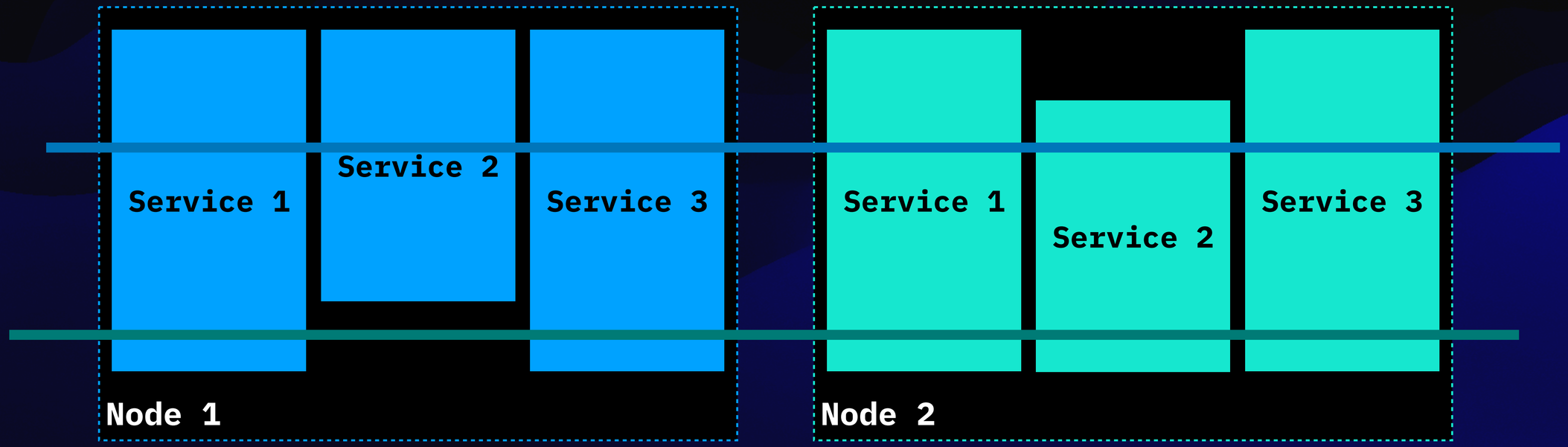
Or run across different servers in a cloud

Node architecture



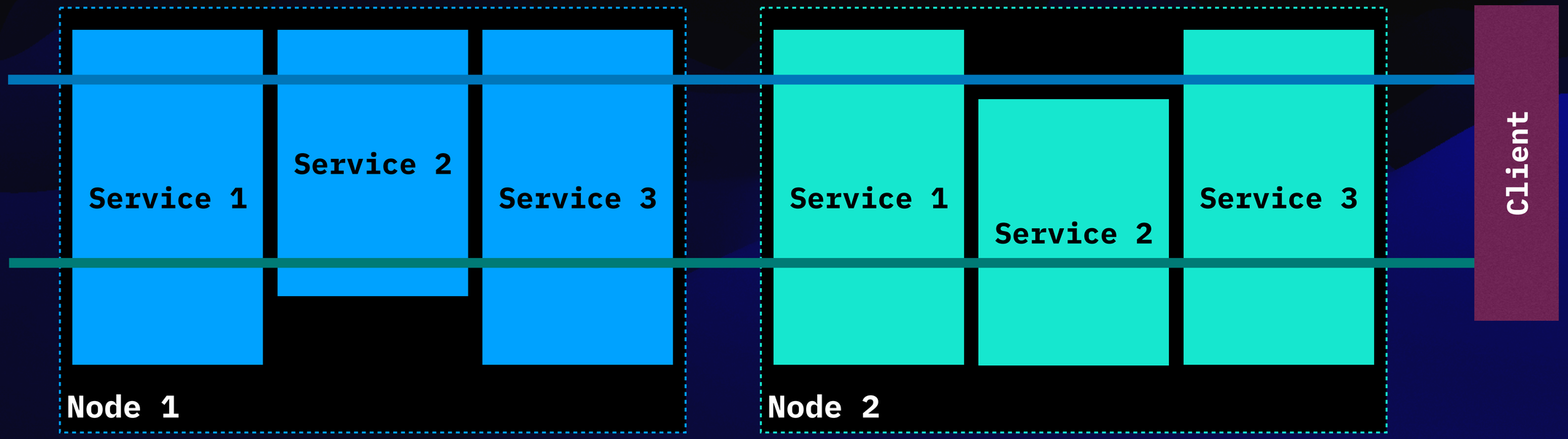
These services communicate via ZeroMQ-based service buses

Node architecture



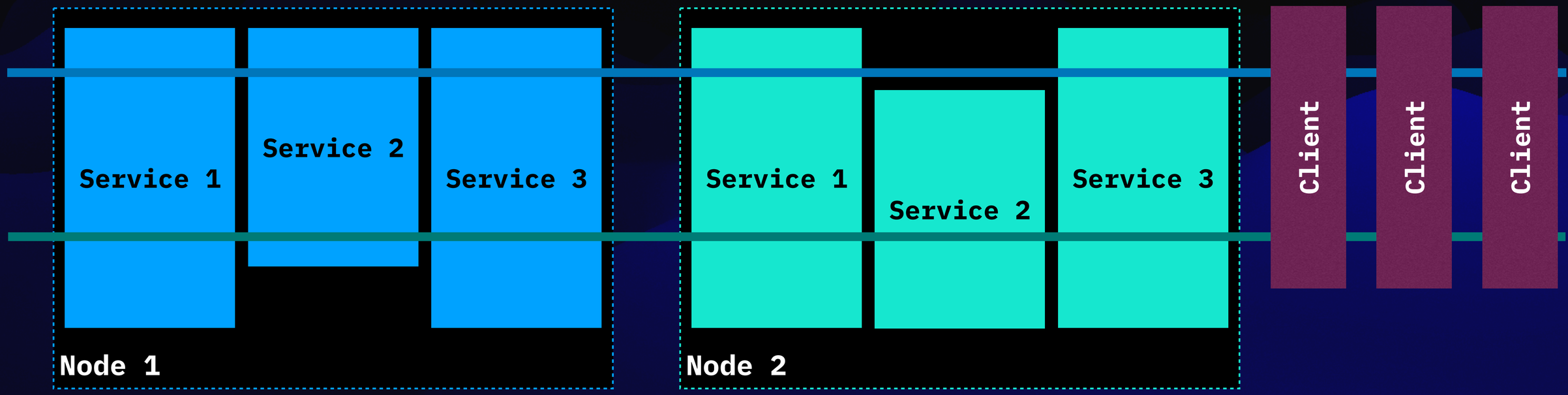
There are internal buses used for intra- and inter-node communications

Node architecture



And external buses providing RPC API to the clients (wallets, command-line tools and other integration points)

Node architecture

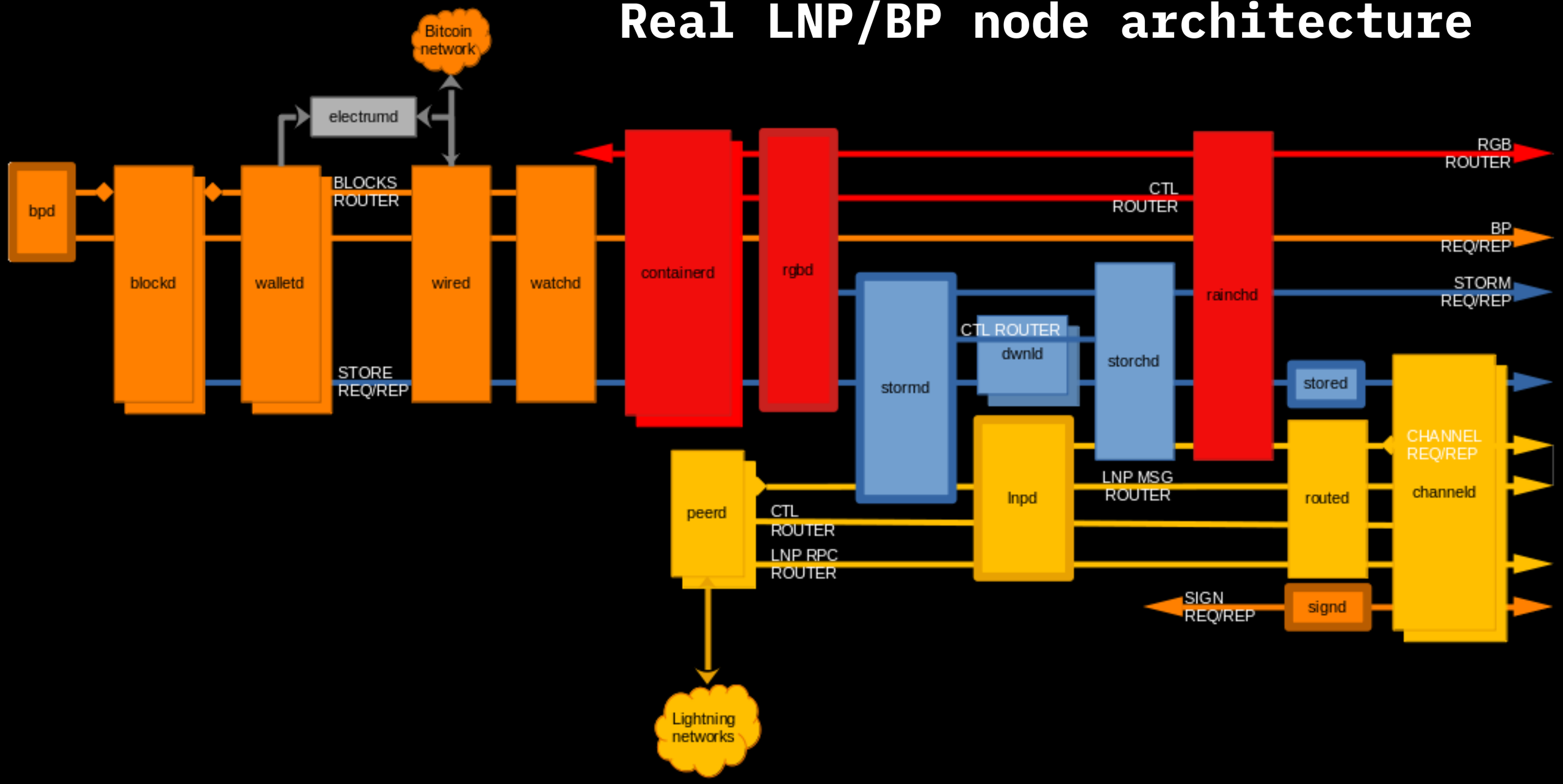


These buses can handle multiple clients connected at the same time

All of these is part of Internet2 microservice architecture developed by LNP/BP Standards Association over last two years

github.com/Internet2-WG

Real LNP/BP node architecture



Connecting nodes

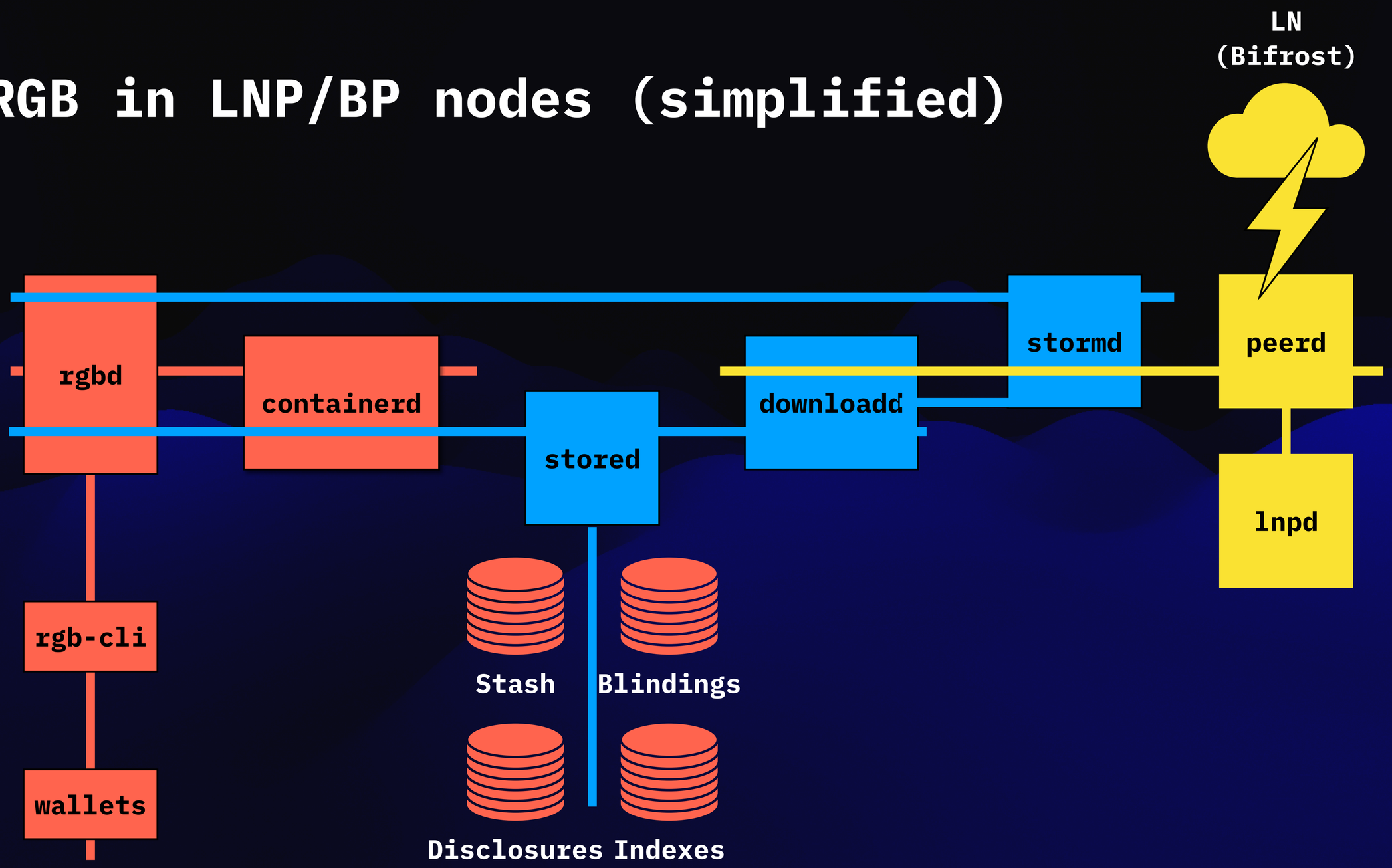
github.com/LNP-BP/nodes/wiki

Node	Bus name	Type	Default socket	TCP port	Cmd flag	Purpose
LNP Node	MSG	ESB	~/.lnp/msg		-M	Messages from and to LN handled through `peerd` to other services
LNP Node	CTL	ESB	~/.lnp/ctl		-X	Control messages from managing daemon (`lnpd`) to other LNP Node serivces
LNP Node	CHNL	ESB	~/.lnp/chnl		-C	Messages between `channeld` service and channel extension services
LNP Node	API	ESB		62962	-R	API for clients connecting the node
RGB Node	CTL	ESB	~/.rgb/ctl		-X	Control messages from managing daemon `rgbd` to `containerd`
RGB Node	API	ESB		63963	-R	API for clients connecting the node
BP Node	BLOCKS	ESB	~/.bp/blocks		-N	Notifications about new mined blocks to subscriber services
BP Node	API	ESB		61961	-R	API for clients connecting the node
BP Node	SIGN	ESB	~/.bp/sign		-K	Signer service
Storm Node	STORE	RPC	~/.storm/store	60960	-S	Storage RPC API for all services requiring persistence
Storm Node	STORM	ESB	~/.storm/storm		-E	Messages for Storm applications run as extension services
Storm Node	API	ESB	~/.storm/api	64964	-R	API for clients connecting the node

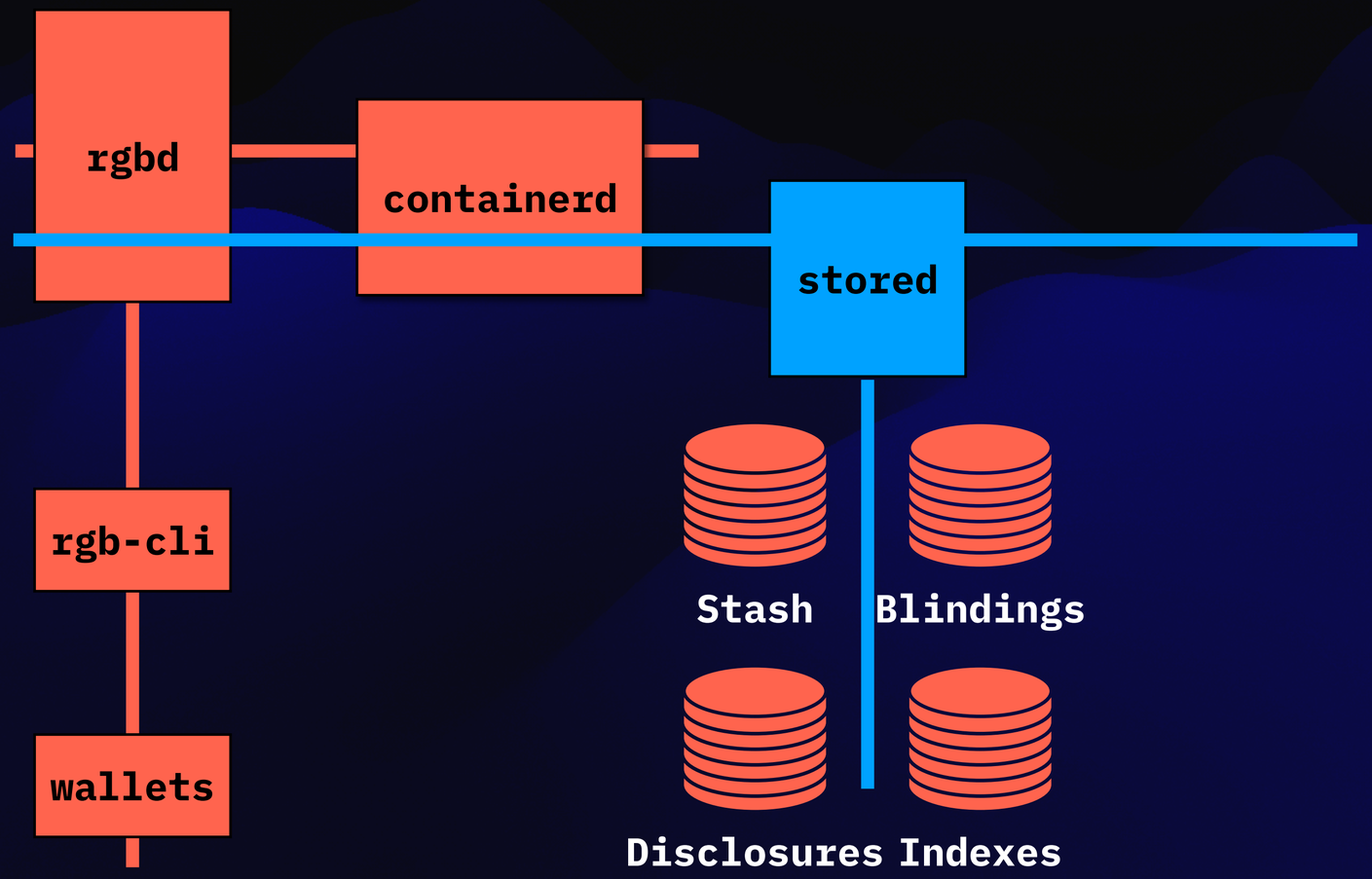
Key takeaways

- Node can be run on mobile, desktop and in the cloud
- Each node has a command-line tool to work with
- Node API is based on ZMQ and strict encoding (**ZMQ-RPC-Sten API**)
- It is **asynchronous**, i.e.
  - Do not blocks client
  - Can return multiple replies on a single response (progress updates etc)
  - Can update client when new blocks arrives /LN channel state changes
- Each node provides a small RPC rust API library to simple integration from rust projects
- Other languague integrations may require
  - Creation of a single bridge from ZMQ-RPC-Sten to HTTP-REST-JSON API
  - Development of languague-specific bindings to rust
  - Development of languague-specific serialization libs

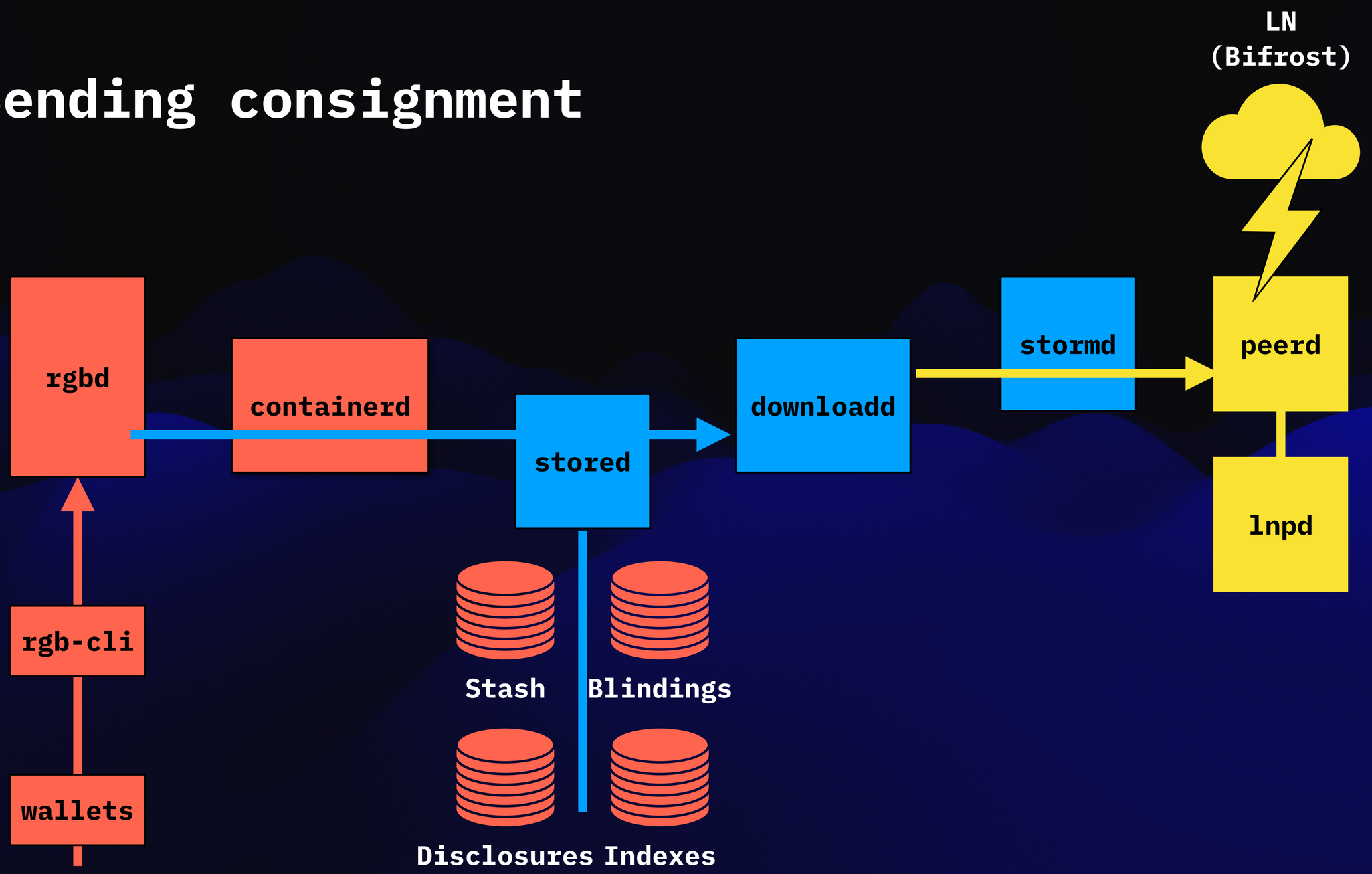
RGB in LNP/BP nodes (simplified)



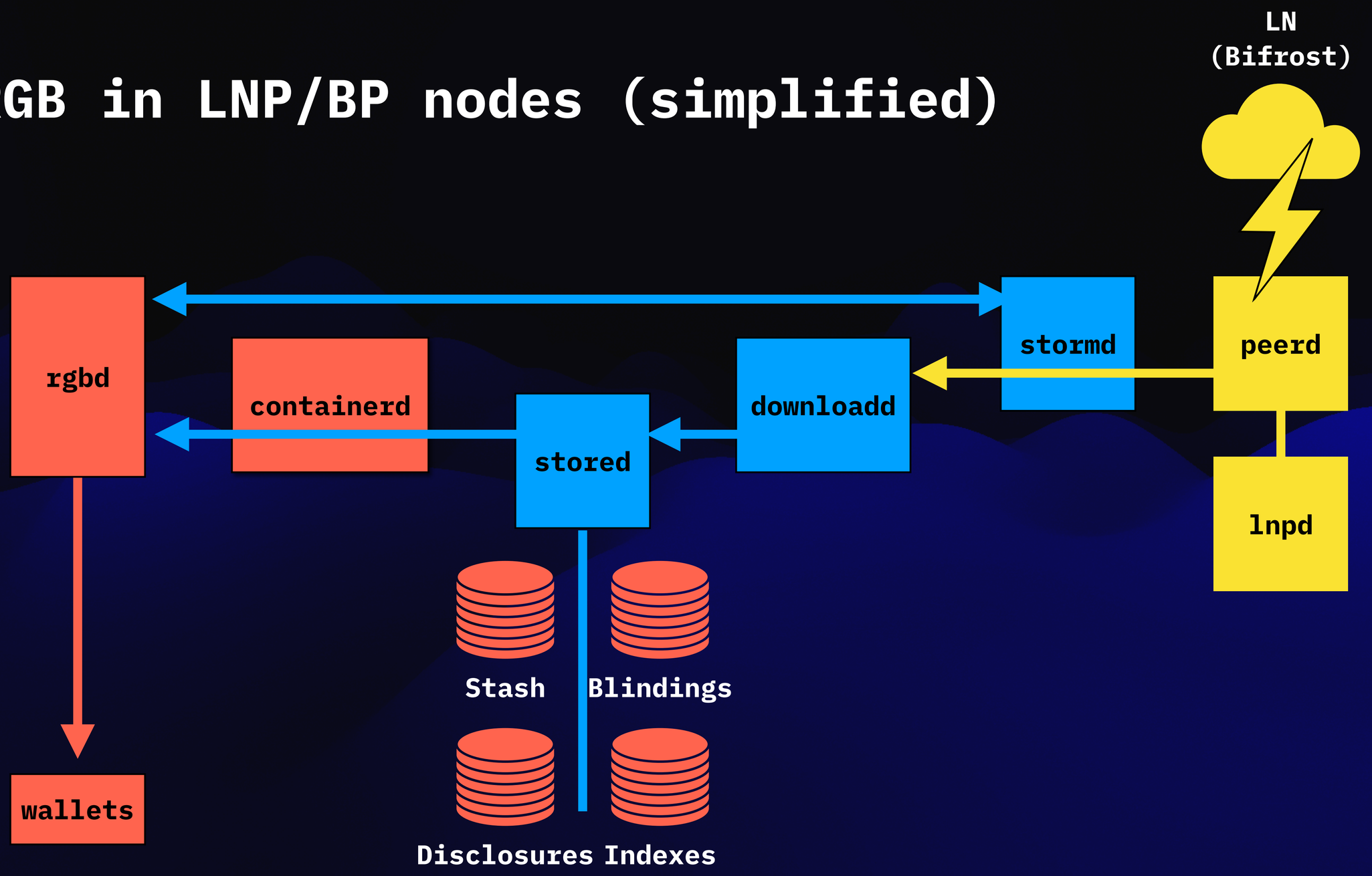
Getting info about the contract state



Sending consignment



RGB in LNP/BP nodes (simplified)



WHY JUST NOT
SIMPLY UPDATE RGB NODE?

Because of:

- Need to handle arbitrary smart contracts, not just tokens
- Unlock ability to all devs to create new forms of smart contracts w/o the need to update the node or ask LNP/BP Association
- Need to link the node to the Lightning network for in-channel asset operations
- Need for a decentralized way of RGB payments
- Providing clients with async updates on the contract state
- Make node reacting on blockchain reorgs
- Need to support multi-gigabyte consignments

RGB Node demo