LNP/BP Initiative:
Pushing the boundaries of what's possible with Lightning network further

Dr Maxim Orlovsky, CEO **Pandora Core**,
Leading engineer, **LNP/BP Standards Association**

github.com/LNP-BP
Twitter: @dr_orlovsky

There is a future for Lightning Network:

Not just a payment scalability layer for Bitcoin

– but a foundation layer for the future Internet of sovereign individuals

This future means that Lightning Network
has to be a layer
on which complex smart contracts

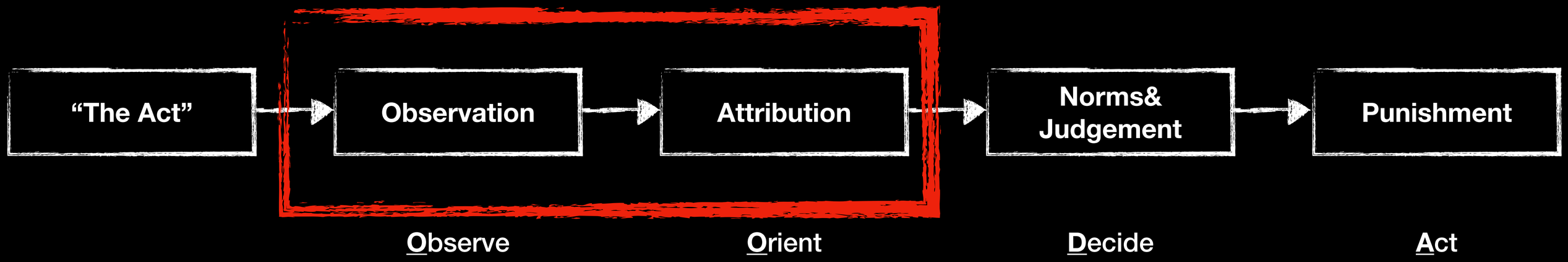
— a financially-enforceable trustless interactions —

can be built

We don’t need “smart contracts” like in “blockchain industry”

- Always a centralized or federated issuer controlling contract code forever
- No true ownership
- Totally KYC’d on two levels: miners & issuers

Smart contracts are for economy without the loop of attribution



Proper smart contract systems

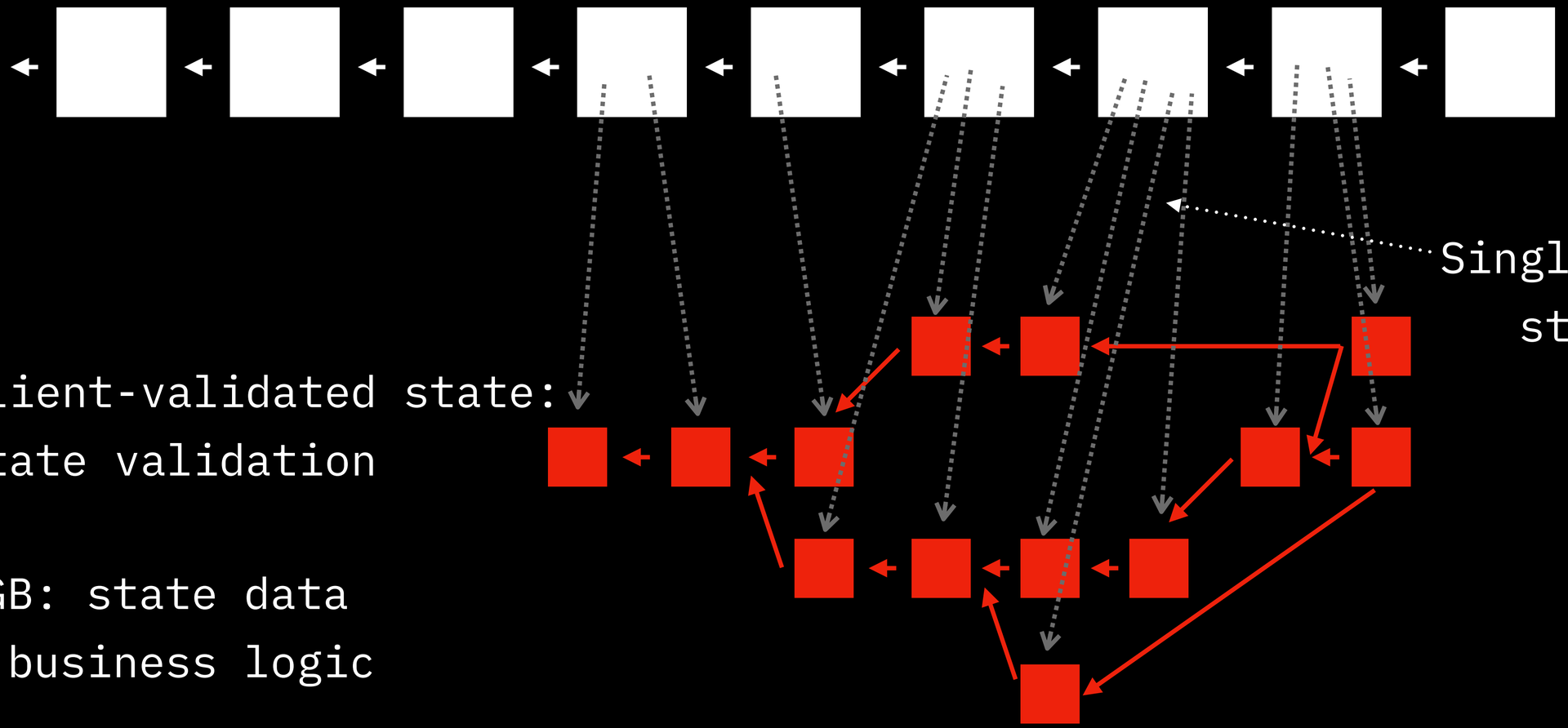
- Bitcoin Script for ownership (Bitcoin, data, assets)
- Scriptless-script based solutions for futures DLCs
- RGB for complex Turing-complete scripts and client-validated state

RGB: how to build rich smart contracts?

Step 1: Remove data from blockchain: client-side validation.
This gives

- Scalability
- Privacy
- No miner incentive extorsion or censorship
- Lightning Network support

Bitcoin blockchain:
state ownership



Single-use seals:
state bindings

Client-validated state:
state validation

RGB: state data
& business logic

RGB: how to build rich smart contracts?

Step 2: Add maximum privacy

- Confidential amounts with Pedersen commitments & bulletproofs
- Data merklzation: owner sees the state and its history only for the data he owns

RGB: how to build rich smart contracts?

Step 3: Separate owner from issuer

- There is only an owner; not only for assets, but for data
- Ownership is controlled by Bitcoin script smart contract — or script less script (DLCs etc)
- Issuer issues, and defines rules; once and for always
- Rules may be changed only by owners; not issuer nor miner

Building the future of Lightning Network

- Lightning Network will evolve
- …and Layer 3 apps will be built on it
- Let’s draw a reckless way forward:
  build a foundation for fast experimentation & extension

And LN should upgrade for …

- Bi-directional channels
- Payment points (PTLCs)
- Schnorr signatures
- Taproot
- eltoo
- … who knows?

All these upgrades are very complex with existing node architecture

New exiting apps on layer 2 & 3 are coming!

- Channel factories
- Discreet log contracts on LN
- Lightspeed: high-frequency micropayments
- Storm: storage and messaging; with multi-peer channels
- Prometheus: high-load computing; with multi-peer channels

LNP/BP Standards Association

- Founded in Oct 2019 by Pandora Core AG, Giacomo Zucco
- Supported by Bitfinex & Tether, Fulgur Ventures
- Oversees Layer 2 & 3-related standards and software dev for Bitcoin / Lightning Network ecosystem
- Contributions/ideas from many cypherpunks & Lightning Network developers
- github.com/LNP-BP

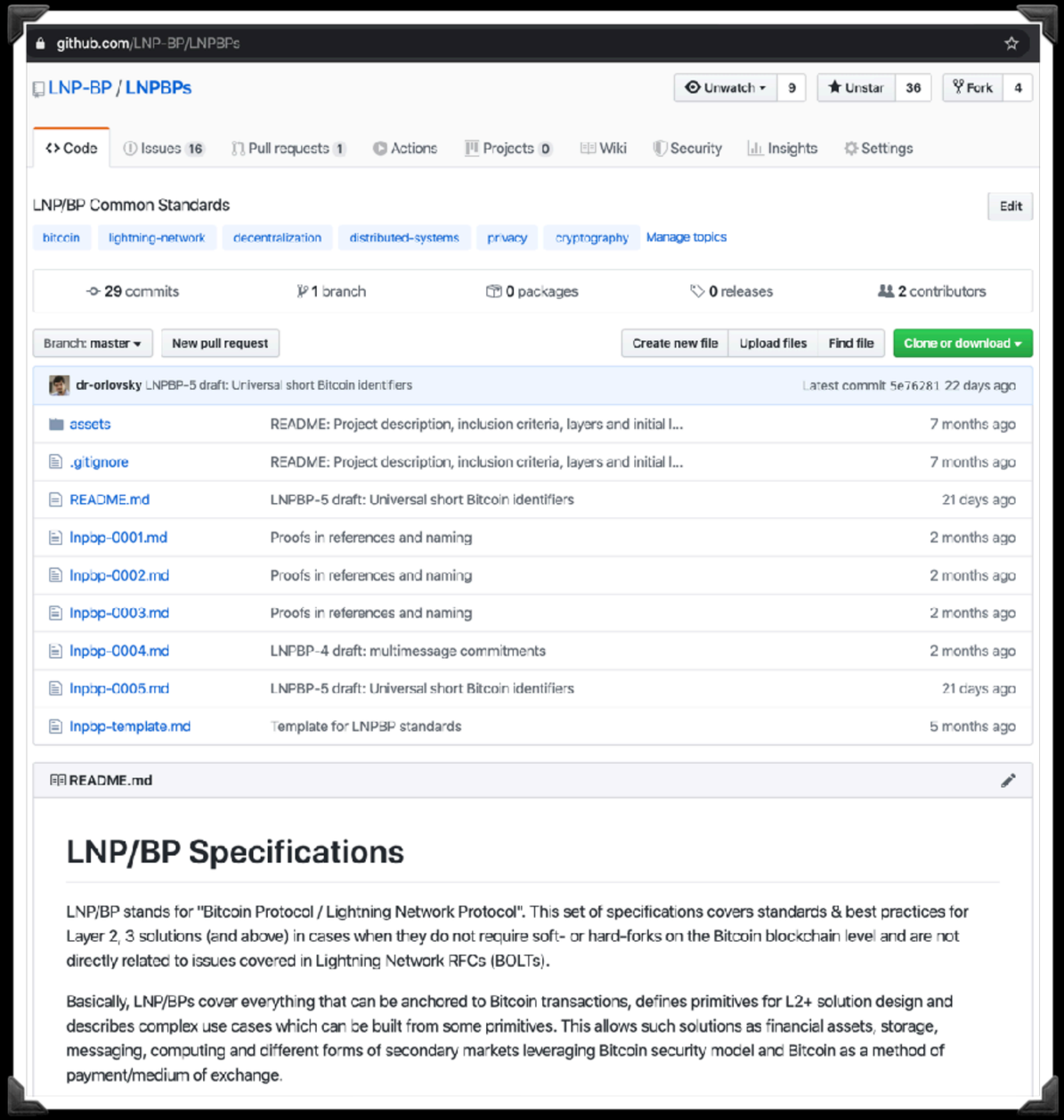
What to do to extend LN?

- Define new tech as a well-written standards: LNPBPs
- Reduce new protocol complexity to a few clear LN extensions: generalized Lighting Network
- Create extensible nodes & SDKs:
  LNP node, BP node, Wallet SDK
- Do reference implementations for the tech:
  rust-LNPBP library and it’s bindings to WASM/JS, Swift, Kotlin

Part I. Standards

github.com/LNP-BP/LNPBPs

- Standards & best practices for Layer 2, 3 solutions (and above)
- Must not require hard- or soft-forks for Bitcoin
- Must not distort Bitcoin miner's economic incentives
- Must not pollute Bitcoin blockchain with unnecessary non-transaction related data or have to maintain such pollution as low as possible
- Must not require a utility or security tokens to function
- Must not depend on non-bitcoin blockchains



List of current LNP/BP proposals

Number	Layer	Field	Title	Owner	Type	Status
1	Transaction (1)	Cryptographic primitives	Key tweaking: collision-resistant elliptic curve-based commitments	Maxim Orlovsky et al	Standard	Draft
2	Transaction (1)	Cryptographic primitives	Deterministic embedding of elliptic curve-based commitments into transaction outputs	Maxim Orlovsky et al	Standard	Draft
3	Transaction (1)	Cryptographic primitives	Deterministic definition of transaction output containing cryptographic commitment	Giacomo Zucco et al	Standard	Draft
4	Transaction (1)	Cryptographic primitives	Multi-message commitment scheme with zero-knowledge provable unique properties	Maxim Orlovsky	Standard	Draft
5	Transaction graph (2)	Client-side validation	Universal short Bitcoin identifiers for blocks, transactions and transaction inputs & outputs	Christian Decker et al	Standard	Draft
6	Transaction graph (2)	Client-side validation	Single-use seals with bitcoin transaction graph	Peter Todd et al	Informational	Draft
7	Off-chain data (3)	Cryptographic primitives	Confidential amounts for client-validated data	Maxim Orlovsky	Standard	Draft
8	Off-chain data (3)	Serialization	Types and requirements for off-chain data serialization	Maxim Orlovsky, Peter Todd	Informational	Draft
9	Off-chain data (3)	Serialization	Extra-transaction proof serialization for LNPBP-2	Maxim Orlovsky	Standard	Draft
10	Client-validated graphs (4)	Commitments	RGB: Client-validated rich state and smart contract system based on LNPBP1-8 standards	Maxim Orlovsky	Standard	Draft
12	Client-validated graphs (4)	Schemata	RGB Schema: presets for RGB smart contracting and state transitions	Maxim Orlovsky	Standard	Draft
13	Client-validated graphs (4)	Serialization	Network serialization standards for RGB schema	Maxim Orlovsky	Standard	Draft
14	Transaction graph (2)	State channels	Prometheus: trustless multiparty computing with escrow & arbitration	Maxim Orlovsky	Standard	Draft
15	Transaction graph (2)	State channels	Storm: trustless storage with escrow contracts	Maxim Orovsky	Standard	Draft
16	Transaction (1)	Transaction structure	ECDSA-based discrete-log contract transactions		Standard	Draft

List work-in-progress of LNP/BP proposals without an assigned standard number

Number	Layer	Field	Title	Owner	Type	Status
n/a	Transaction graph (2)	State channels	Lightning network eltoo transaction structure	Christian Decker	Standard	Draft
n/a	Transaction graph (2)	P2P	Pay-to-EC point multiparty transaction update coordination		Standard	Draft
n/a	Client-validated graphs (4)	P2P	RGB state announcements for Lightning Network protocol	n/a	Standard	Draft
n/a	Client-validated graphs (4)	P2P	RGB state updates over Lightning Network onion messaging	n/a	Standard	Draft
n/a	Application (5)	DEX/DMP	Spectrum: routed RGB state swaps over Lightning Network	n/a	Standard	Draft

Current set new protocols

- RGB & Spectrum: assets, identity, smart contracts & DEX on LN
- Lightspeed: high-frequency micropayments
- Storm: storage and messaging; with multi-peer channels
- Prometheus: high-load computing; with multi-peer channels

Simplifying protocols down to components

making Generalized Lightning Network:

the true “LNP” in LNP/BP, like TCP in TCP/IP

And LN should upgrade for …

- Bi-directional channels
- Payment points (PTLCs)
- Schnorr signatures
- Taproot
- eltoo
- Channel factories
- Discreet log contracts on LN
- Lightspeed: high-frequency micropayments
- Storm: storage and messaging; with multi-peer channels
- Prometheus: high-load computing; with multi-peer channels

What new protocols may require?

- Change the structure of the funding tx
- Tweak information in existing commitment tx outputs (like public key)
- Add custom outputs to the commitment transactions and dependent transaction trees
- Make HTLC a function based on previous point

github.com
/LNP-BP/LNPBPs/issues/24

Lightspeed micropayments for Lightning Network #24

Edit | New issue

Open | dr-orlovsky opened this issue 9 days ago · 0 comments

dr-orlovsky commented 9 days ago | Member

Abstract

This work proposes concept of fast and reliable micropayments protocol ("Lightspeed micropayments for Lightning Network") which allows millions of transactions per channel without any additional network traffic and any per-transaction signature generation. This requires generalization of the Lightning network (such that parties may be able to add additional outputs to the commitment transactions) and RGB.

1. Background

1.1. Setup

One of the main design goals for the Lightning Network was an idea of micropayments: fast repeating payments for small amounts (sometimes <1 satoshi) between two parties.

However, the current Lightning Network design prevents effective micropayments due to the number of factors, which we will examine on a model setup where Party 1 (Client) has to pay to Party 2 (Server) for each API call.

The setup of the configuration will require two Lightning Nodes; we will examine the simples case when there exists a direct channel between them:

Assignees
No one—assign yourself

Labels
WIP
proposal

Projects
Generalized Lightning Network
To do
Development Calls Agenda
In progress / In discussion

Milestone
No milestone

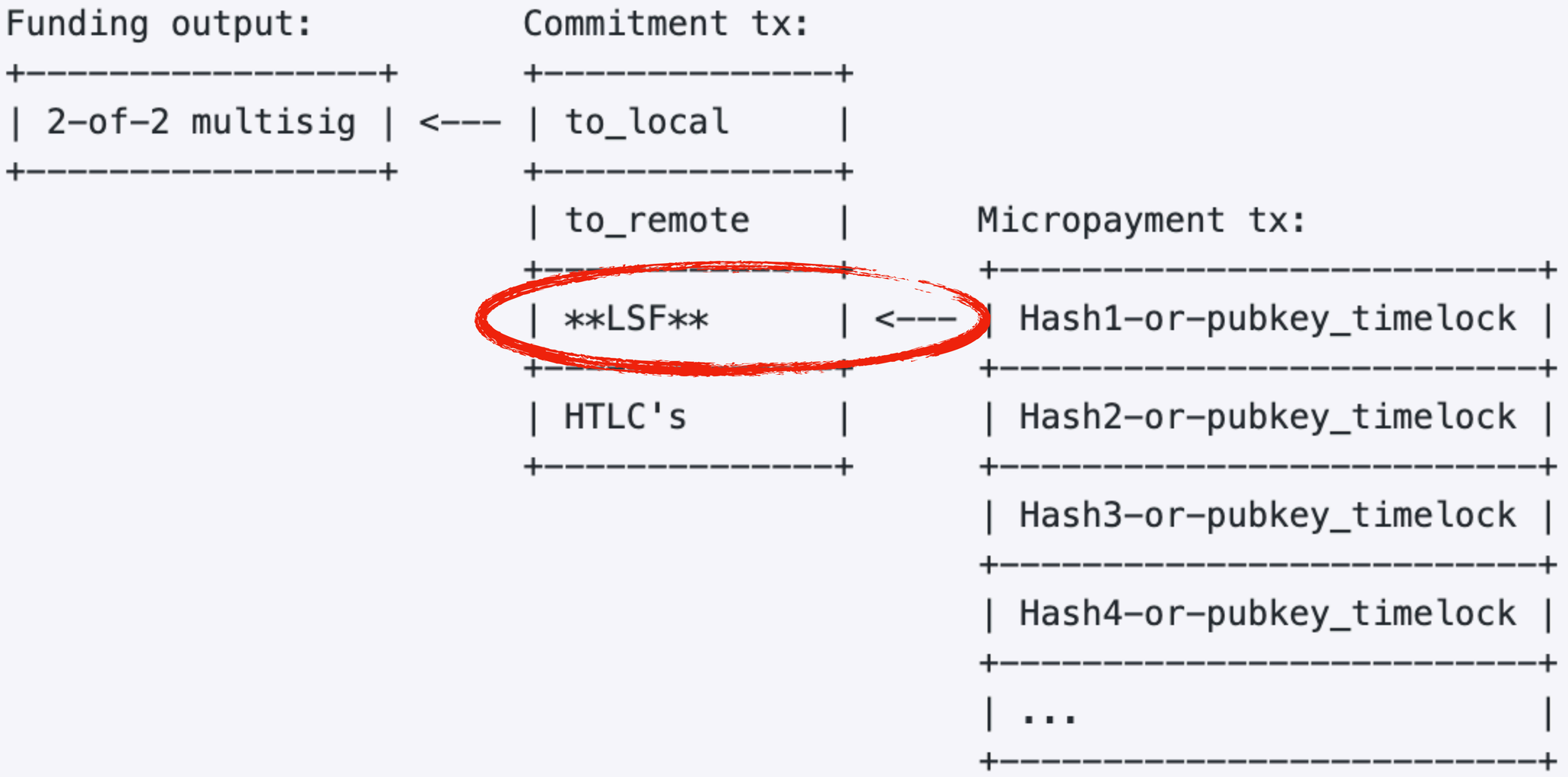
Linked pull requests
Successfully merging a pull request may close this issue.
None yet

Notifications | Customize
Unsubscribe
You're receiving notifications because you're watching this repository.

Problem with LN micropayments

Per each payment there must be:

- Generation of multiple signatures: at least 6 signatures
- 4 inter-process communications (2 on each side) between application and Lightning Node;
- 3 network requests/replies between Client and Server Lightning Nodes
- 2 network requests between Client and Server apps, including the actual API call


Funding output:
2-of-2 multisig
Commitment tx:
to_local
to_remote
LSF
HTLC's
Micropayment tx:
Hash1-or-pubkey_timelock
Hash2-or-pubkey_timelock
Hash3-or-pubkey_timelock
Hash4-or-pubkey_timelock
...

Still some problems

- Enormous LSF transaction size and it’s publication cost breaks micropayments economy
- Can’t efficiently work with subsatoshi payments
- Limited in the number of payments in a single micropayment transaction to the maximum number of outputs that can fit in the block (tens of thousands).

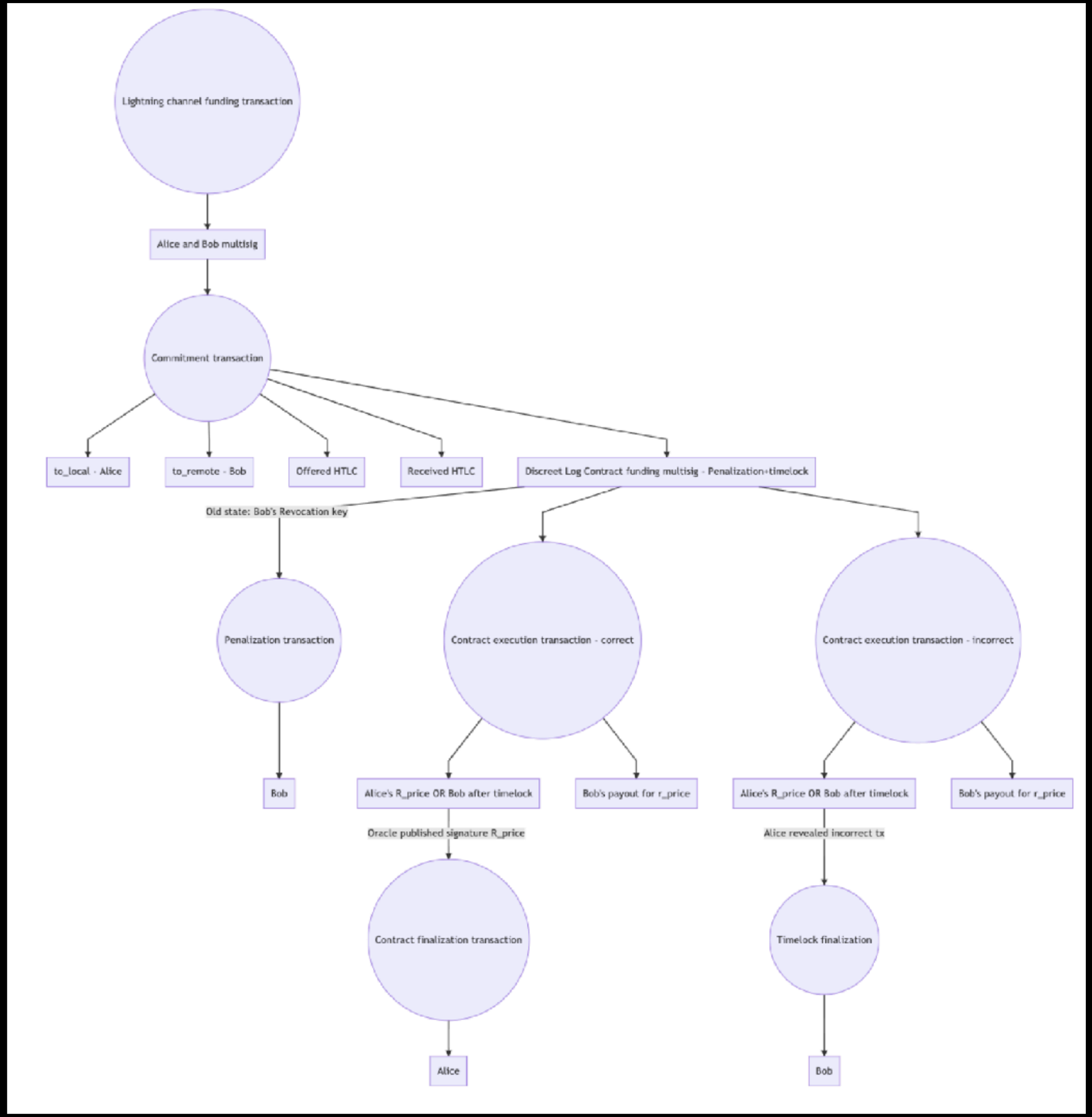

Funding output:
2-of-2 multisig
Commitment tx:
to_local
to_remote
LSF
HTLC's
Micropayment tx:
2-of-2+timelock multisig
Single-use seals
Client-validated RGB data:
Secret1-locked tokens
Secret2-locked tokens
Secret3-locked tokens
Secret4-locked tokens
...

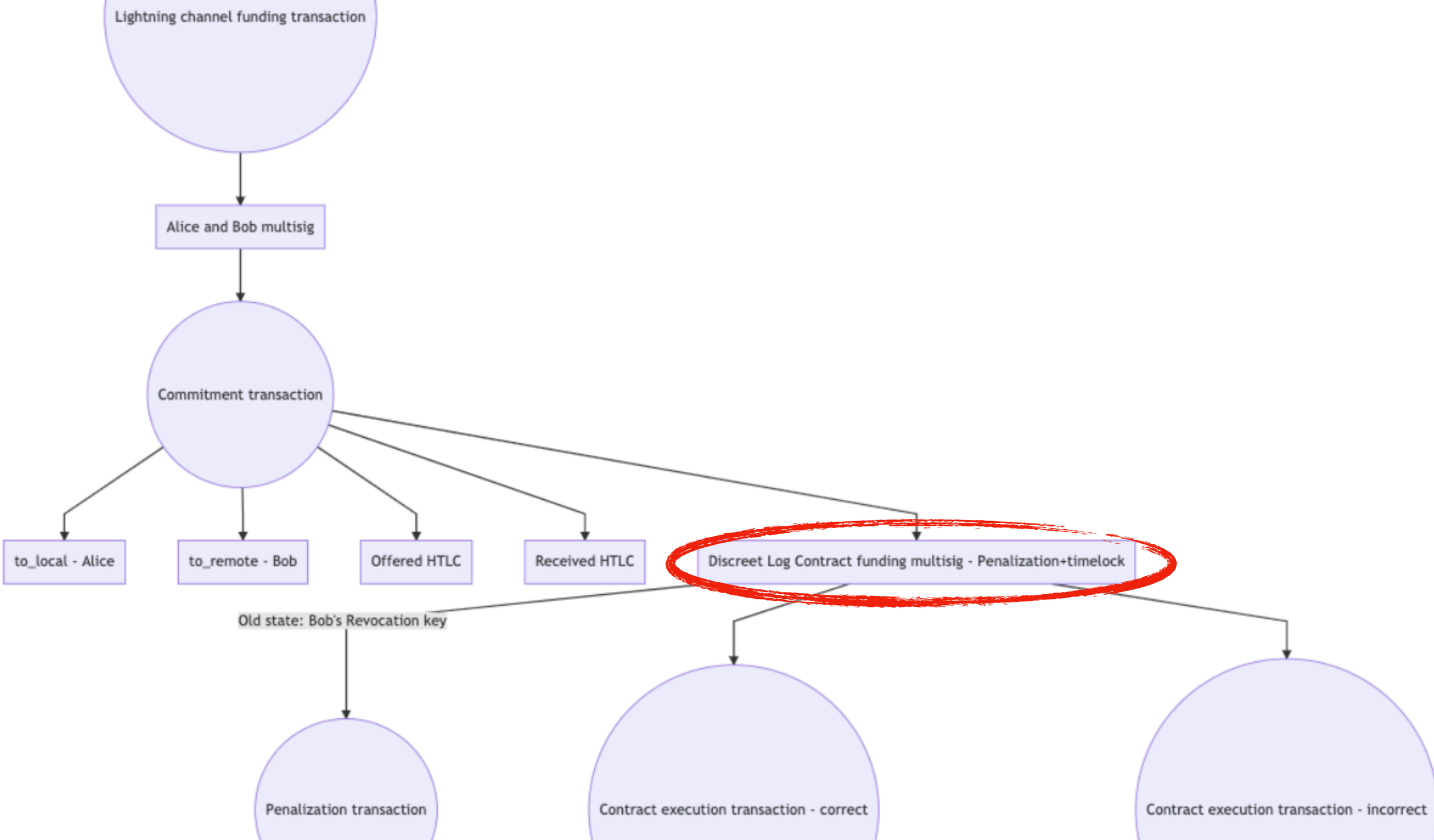
Lightspeed micropayments

- 0 overload on client or server per each payment
- scales to sub-satoshi payments
- can handle millions and millions of transactions
- no overload on commitment transaction: just a single output is added and a one new pre-signed transaction is required

DLCs over LN

by Juraj Bednár & René Pickhardt

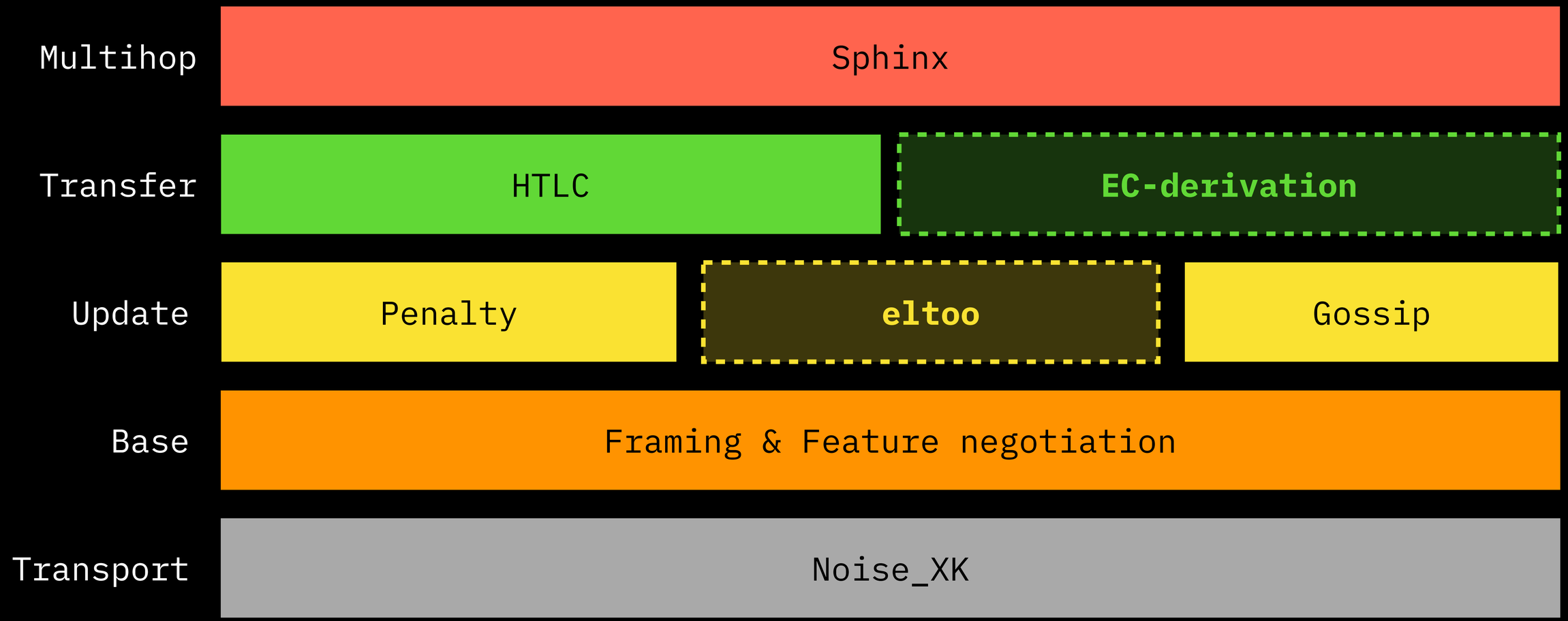



Lightning channel funding transaction
Alice and Bob multisig
Commitment transaction
to_local - Alice
to_remote - Bob
Offered HTLC
Received HTLC
Discreet Log Contract funding multisig - Penalization+timelock
Old state: Bob's Revocation key
Penalization transaction
Contract execution transaction - correct
Contract execution transaction - incorrect

So how to bring all of these to LN?

Lightning Network Architecture

after Christian Decker



LNP components for generalized LN

- **Transport Layer**, based on Noise_XK: BOLT-8
  P2P protocol for data transfer

- **Messaging Layer**: BOLT-1 extended with TLVs and custom messages
  RPC layer for decentralized Web

- **Channel Negotiation Layer**
  Uses messaging protocol to negotiate the structure for funding transaction and channel parameters

- **Channel Operation Layer**: the rest of the current LN extended with TLVs and custom messages to build custom commitments and commitment-based transactions

Customization via

- Exchange of PSBTs using messaging layer
- PSBTs can be used as a form of insert-update-delete channel API

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 & Eclair)

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

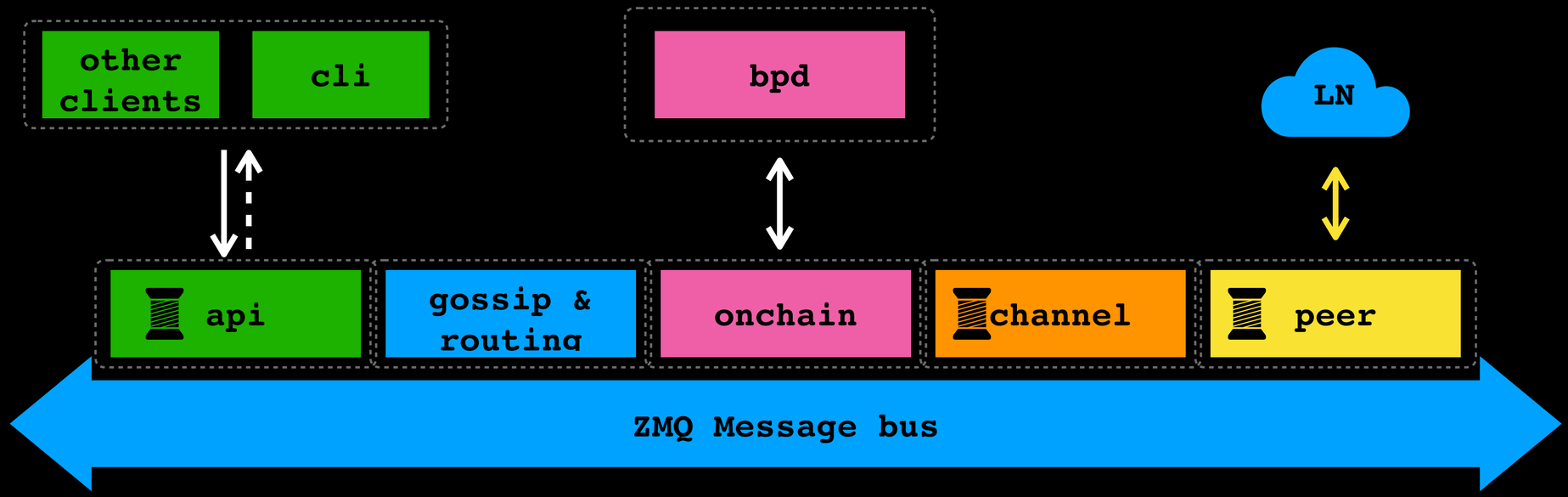
LNP node (Lightning node)

- Based on rust-lightning library by Matt Corallo @Square Crypto & @Chaincode Labs
- Multi-thread non-blocking microservice code
- 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)

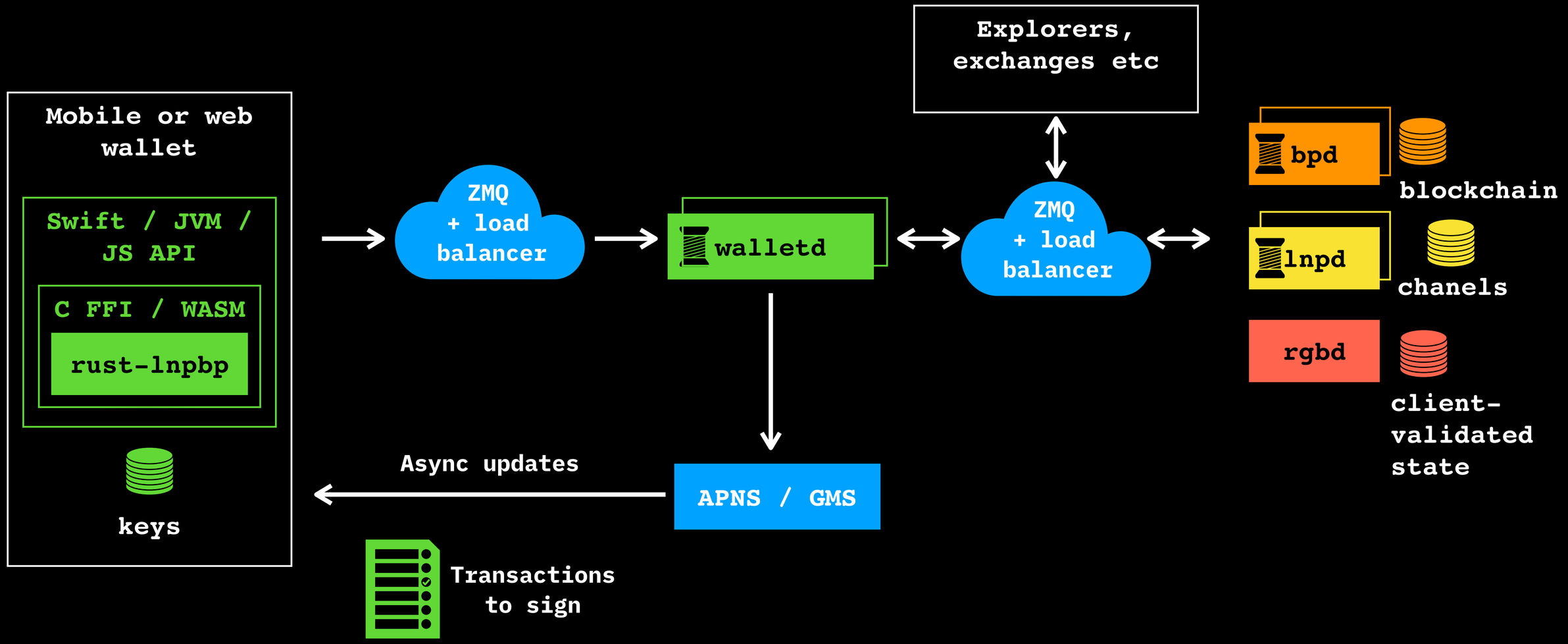
BP node

- Parse bitcoin blockchain into a compact database
- Based on **LNPBP-5** standard – joint work with C. Decker: https://github.com/LNP-BP/lnpbps/blob/master/lnpbp-0005.md
- Based on rust-bitcoin library by Andrew Poelstra @Blockstream
- Provide a query API for data that can't be provided by either Bitcoin Core and Electrum Server
  - ◉ getting transaction spending particular output
  - ◉ querying transactions by their script/**miniscript** code

New LN node architecture



Non-custodial clients/wallets



Contribute to github.com/LNP-BP!

- LNP/BP Standards development & discussions: github.com/LNP-BP/lnpbps
- Standard Rust library: github.com/LNP-BP/rust-lnpbp
- Bitcoin node: github.com/LNP-BP/bp-node
- Lightning node: github.com/LNP-BP/lnp-node
- Discussions @ Freenode IRC: #lnp-bp

Other talks and materials

- The Lightning Conference
- Transylvania
- Storm: Tel Aviv
- Prometheus: Riga