RGBCON 0

LNP/BP Standards Association

Non-conference for client-validated state (RGB)

Organised by **LNP/BP Standards Association**, using donations from
Bitfinex & Tether, Fulgur Ventures, Poseidon Group

Days I-II Re-cap

Solved questions

- Multiple improvements to public key tweaking
- Analysis on different aspects of Taproot interoperability and compatibility
- Roadmap for the scripting system

Unsolved questions

- Zero-knowledge proofs for multi-asset transfers
- Proposal to remove range proofs (needs validation)
- Better public key fingerprinting in LockScripts:
  - Usage of miniscript determinism
  - Usage of stack content and Script emulator

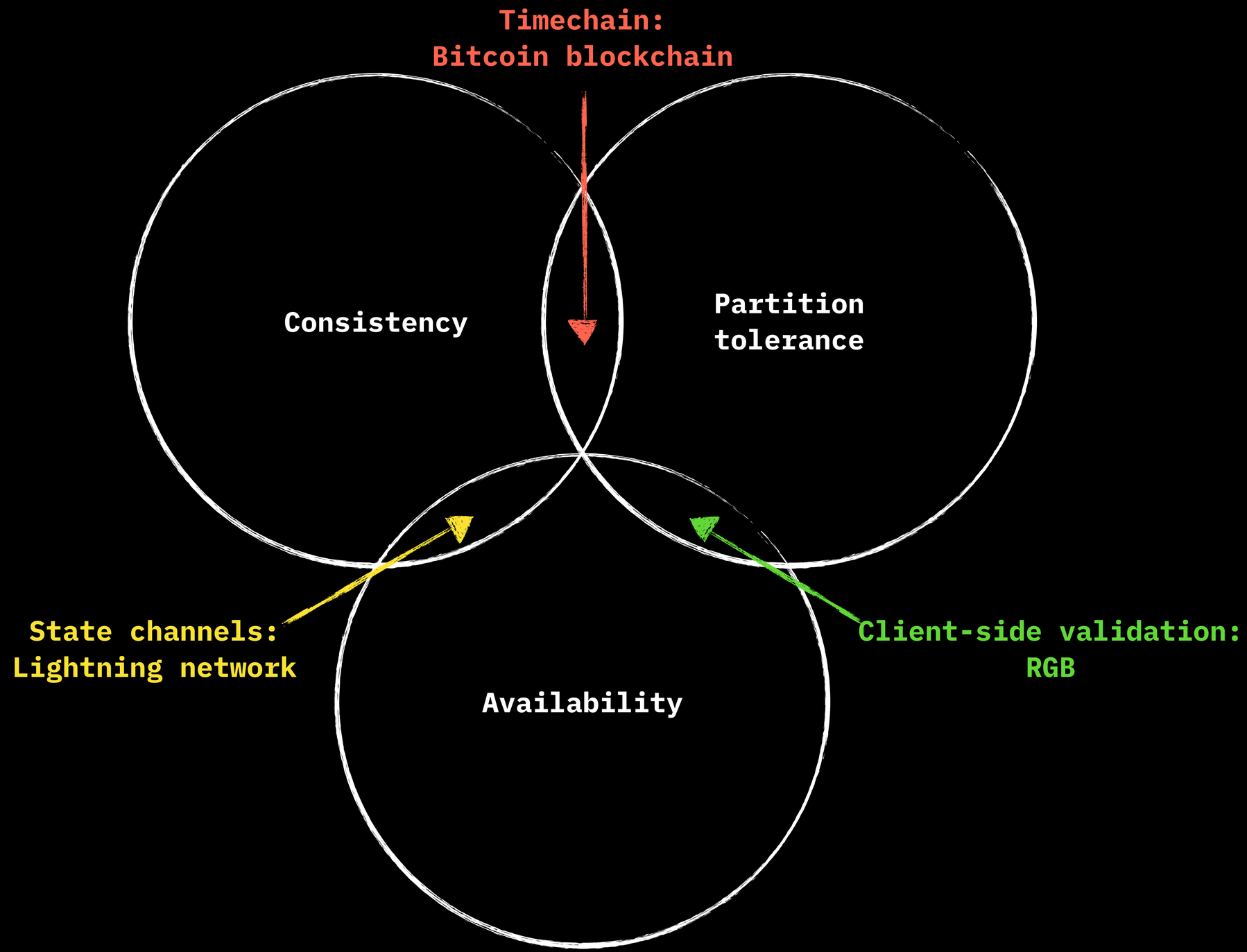
LNP/BP Standards Association

Day III: Lightning Network. Architecture

Dr Maxim Orlovsky, CEO **Pandora Core**, Secretary of LNP/BP Standards Association

Part I: Lightning Network

What we need to know and keep in mind while designing RGB and Spectrum on top of Lightning Network


Timechain:
Bitcoin blockchain
Consistency
Partition
tolerance
Availability
State channels:
Lightning network
Client-side validation:
RGB

Lightning Network

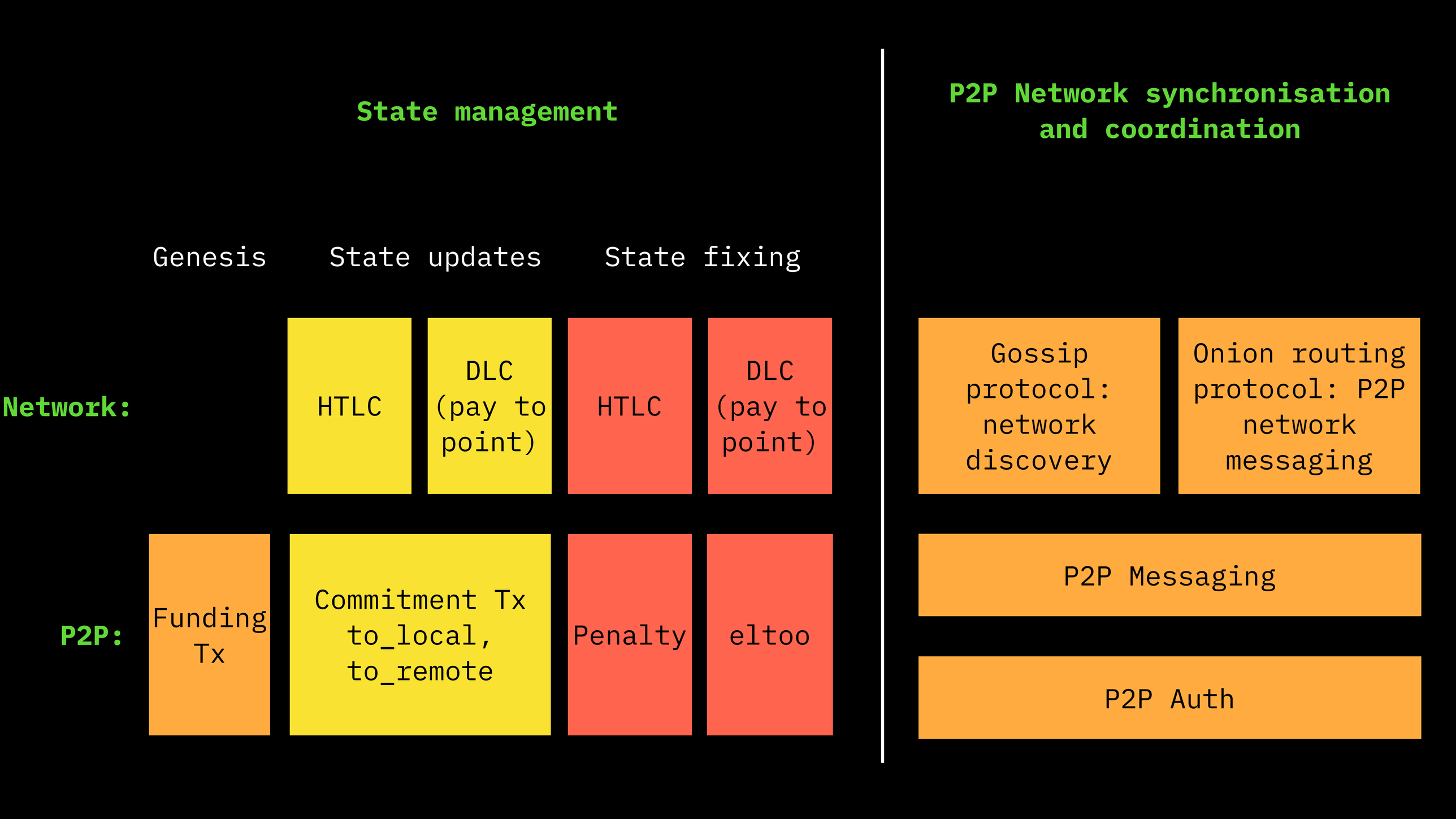
A case of distributed database with

- replicated state
- over a fixed set of participants/replicas
- may not contain the history of state transitions
- no consensus protocol

Scalable for internal state transitions (transaction)

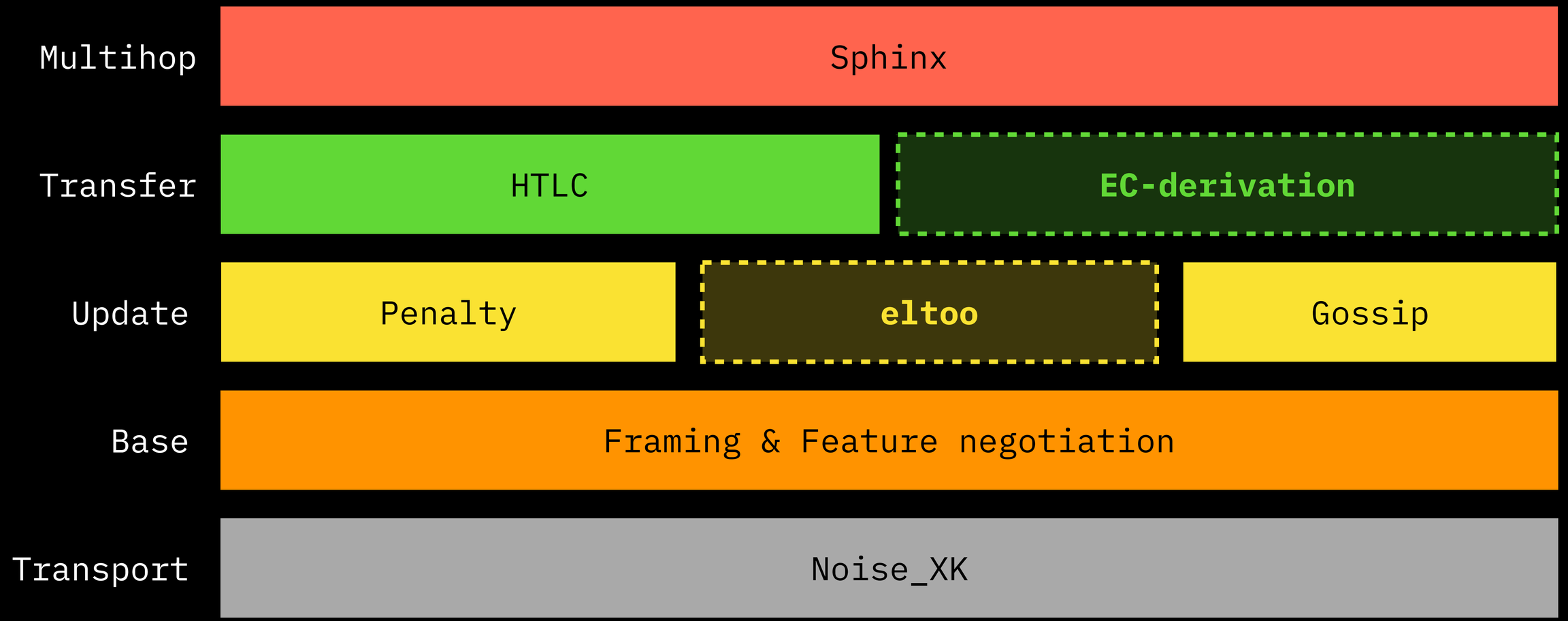
Not scalable for the number of nodes and external coordinated state transitions

Intermediate privacy


State management
P2P Network synchronisation and coordination
Genesis
State updates
State fixing
Network:
HTLC
DLC (pay to point)
HTLC
DLC (pay to point)
Gossip protocol: network discovery
Onion routing protocol: P2P network messaging
P2P:
Funding Tx
Commitment Tx to_local, to_remote
Penalty
eltoo
P2P Messaging
P2P Auth

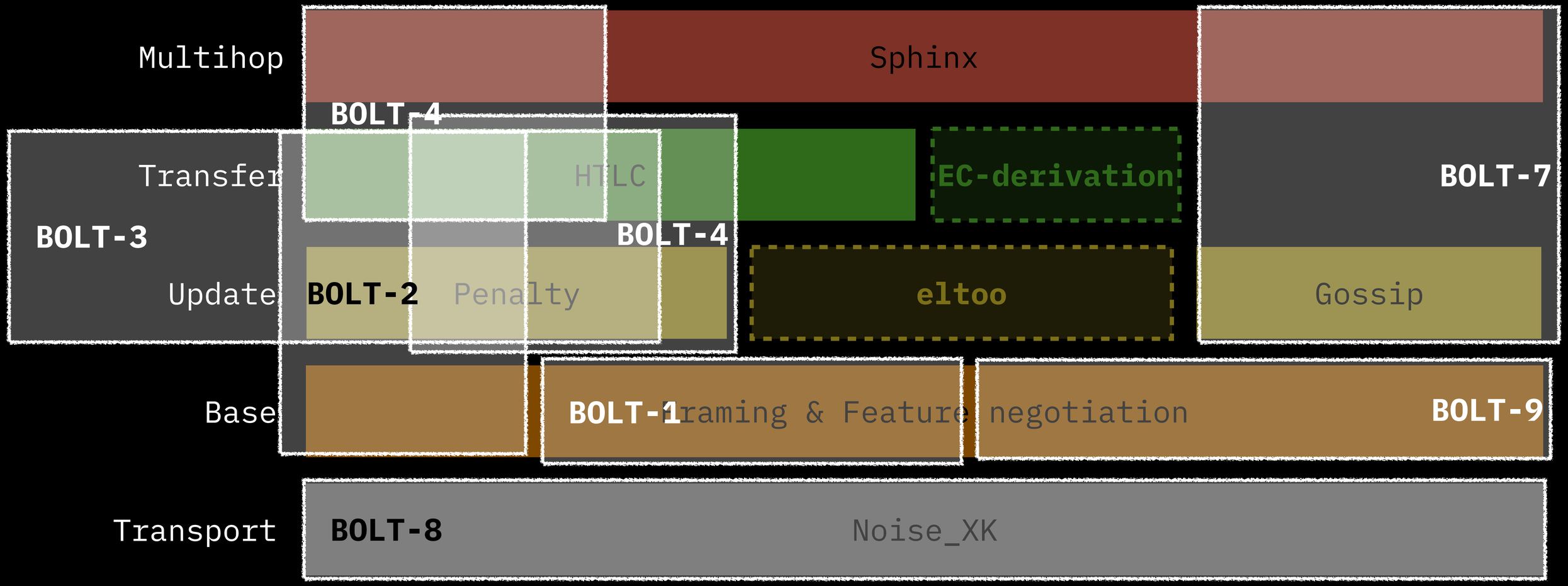
Lightning Network Architecture

after Christian Decker



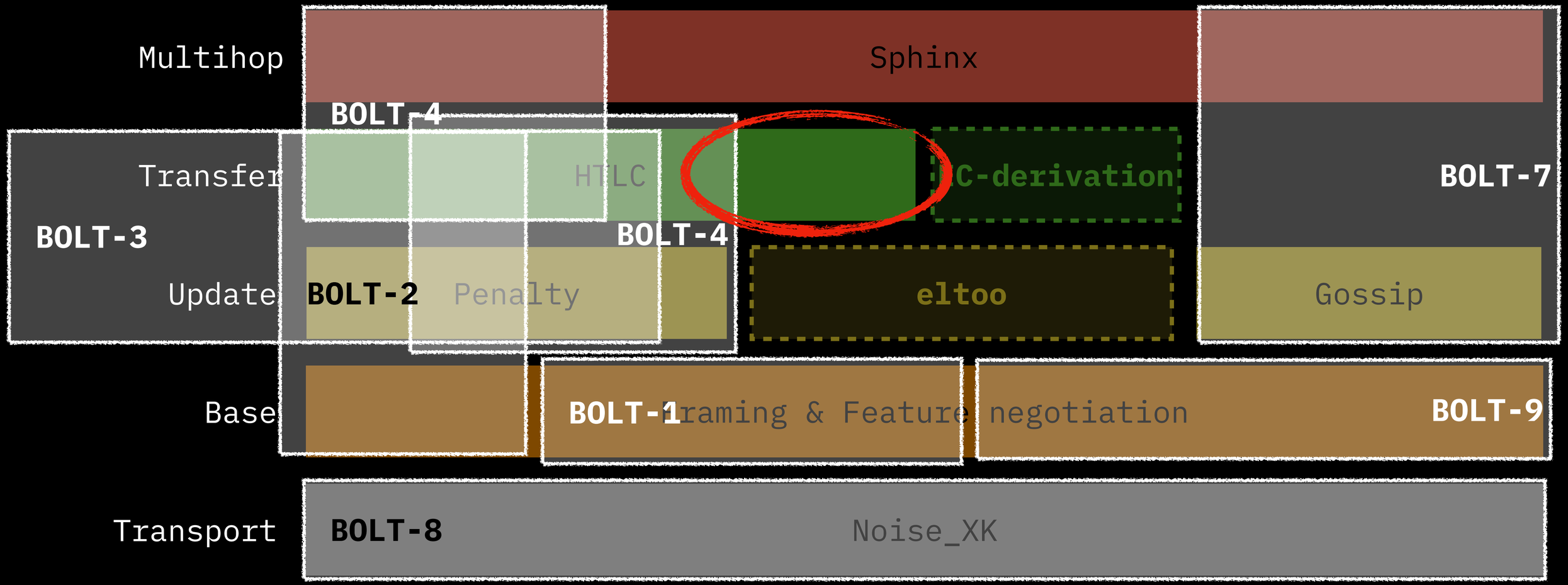
PROBLEM I
SPECIFICATIONS

Real BOLT Specifications



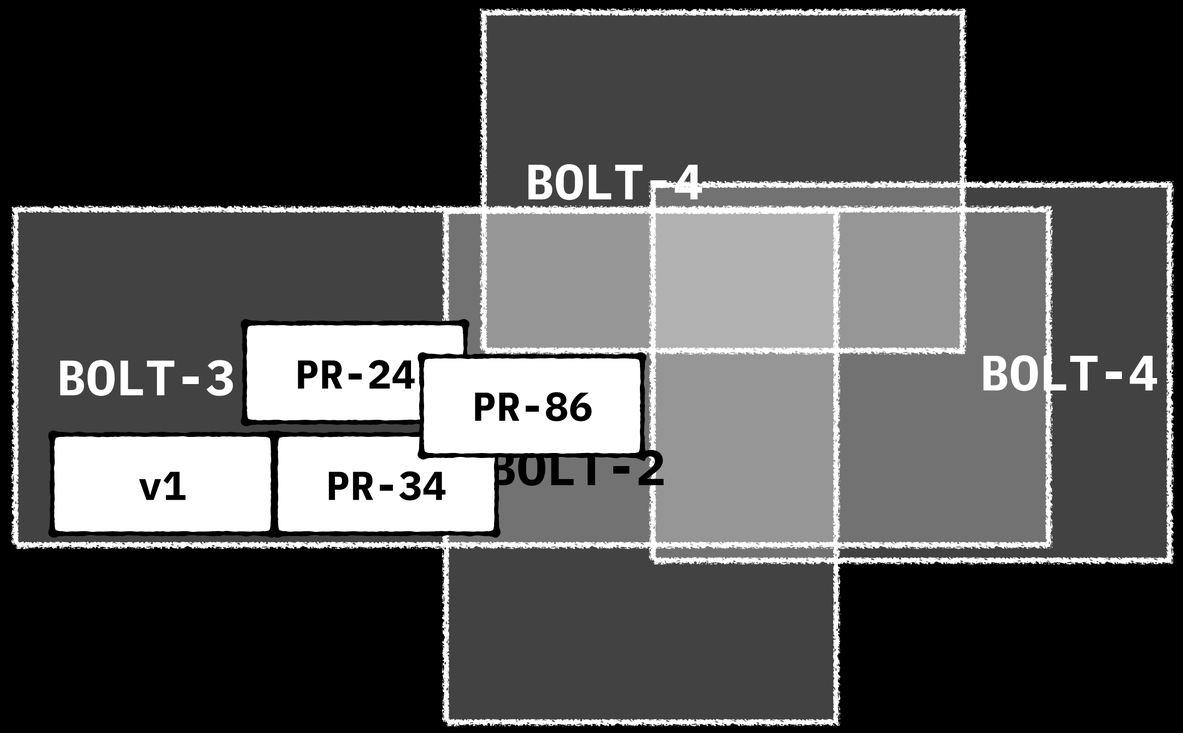
With the current specification approach it is **impossible to reconstruct** even such a simple thing as **a sequence diagram** that can simultaneously cover chain of transaction updates and messaging between the nodes **for multi-hop payments**

BOLTs coverage for real LN is lacunar



Specification Timechain

of healthy man



of ~~smoker~~ BOLT

Why proper abstraction _is_ important

- Security
- Interoperability
- Software upgradability
- New features

PROBLEM II

LN IS NOT DESIGNED WITH L3+ IN MIND

2 types of extending LN functionality

- Extending **payment LN functionality**, like:
  - routing algorithms
  - channel rebalancing
  - submarine swaps
  - ...
- **Layers on top of LN**, utilising:
  - onion-routed messaging protocol
  - gossip protocol
  - commitment transactions for commitments :)

The missing parts:

- No proper extensibility for LN nodes in the specs (it’s up to node implementation to decide)
  - ➡ No way to create cross-node extensions
- No proper layering of the LN tech itself
- No clear way to introduce features:
  - ➡ even with feature flags, it’s unclear how to introduce new TLV types without conflicts

PROBLEM III
NOT EVEN A BETA YET

What is Lightning Network today

- **Two-party protocols (P2P):**
  - Authentication and messaging
  - Funding protocol: single-party
  - State updates: V0 Witness commitment transaction
  - State restoration: penalty transaction
- **Multiparty protocols (Network):**
  - Gossip: network discovery
  - Routing: Sphinx
  - Value transfers: HTLC
  - Onion-routed messaging: P2P communication over network

What will Lightning Network be tomorrow

- **Two-party protocols (P2P):**
  - Authentication and messaging
  - Funding protocol: bi-party -> factories & nodelets
  - State updates: V0 Witness -> V1 Taproot commitment transaction
  - State restoration: penalty transaction -> eltoo
- **Multiparty protocols (Network):**
  - Gossip: network discovery -> trampoline
  - Routing: Sphinx -> new routing protocols
  - Value transfers: HTLC -> adaptor signatures/DLC (“pay to EC point”)
  - Onion-routed messaging: P2P communication over network

What will change the LN (chronological order):

- **Internal improvements**
  - P2WSH to_remote output
  - New routing protocols
  - ???
- **Schnorr signatures**
  - DLC payment routing: pay to elliptic curve point (with adaptor signatures)
  - Nodelets: multiparty P2P channels
- **Taproot & tapscript (comes bundled with Schnorr)**
  - Changed Tx structure: no scripts in outputs, only P2TR outputs
- **SIGHASH_NOINPUT**
  - Different state restoration protocol: eltoo instead of penalty

Many of those require Bitcoin softforks

It will take years for LN protocols to become stable

Before that, using LN in L3 means constant re-implementation and re-integration

LN problems with coming or future solution

- Channel close locks funds: P2WPKH -> P2SH for *to_remote* output
- Multihop payment fee cheating: HTLC -> DLC (requires Schnorr)
- State restoration: penalty -> eltoo (requires SIGHASH_NOINPUT)
- Liquidity:
  - Submarine swaps
  - Bi-funded channels
  - New protocols on channel rebalancing
  - ***Spectrum***
- On-chain channel opening
  - Channel factories
  - Nodelets (requires Schnorr)

LN problems with no definite solution:

- Routing: not scalable
  - New routing protocol proposals
  - Trampoline nodes proposal
- Reverse American call option: very important to RGB
- Privacy — ***RGB*** or ***Confidential Transactions***

Our aim is to fix different parts of LN and become able to provide robust operations for LN-based payments with bitcoin and USTD, which should boost the adoption of censorship-resistant money

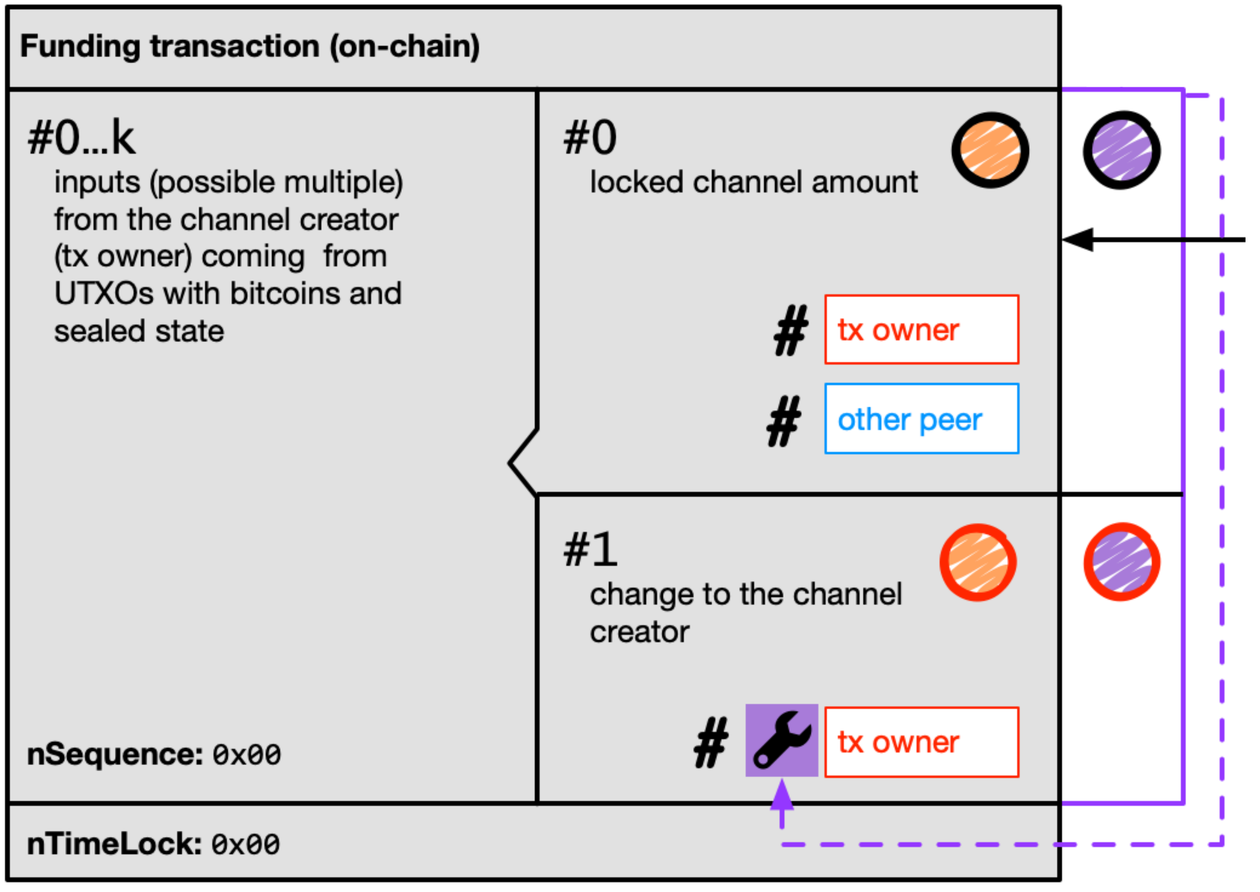
Proposed solution:

- Production-ready subnet for LN with stable functionality and feature flag signalling
- Strict standards-following basic LN node with all new changes and enhancements moved into separate external code modules (plugins, alternative channel daemons implementations)

Part II: CVS/RGB on Lightning Network

RGB & Spectrum

- Multiple **privacy** enhancements:
  - Client-side validation: zero L1/L2 visibility
  - Zero-knowledge for confidential amounts
  - Much more complicated LN network transfer analysis
- Enhances **LNP/BP adoption**, especially for stable coins
- Potential **off-chain BTC** transfer L2/L3 solution



Legend:

On-chain transaction

Partially-signed transaction

Possible future transaction

Offchain state

Bitcoins inside tx output

State bound to the tx output

Bitcoins or sealed state owned by the tx owner/ local party ("Alice")

Bitcoins or sealed state owned by the tx other peer / remote party ("Bob")

Local party:

Signature

Unsigned

Public key

Remote party:

Signature

Unsigned

Public key

\# Hash value of everything that follows after this sign

LNPBP-1 tweaked public key containing the state commitment

State commitment

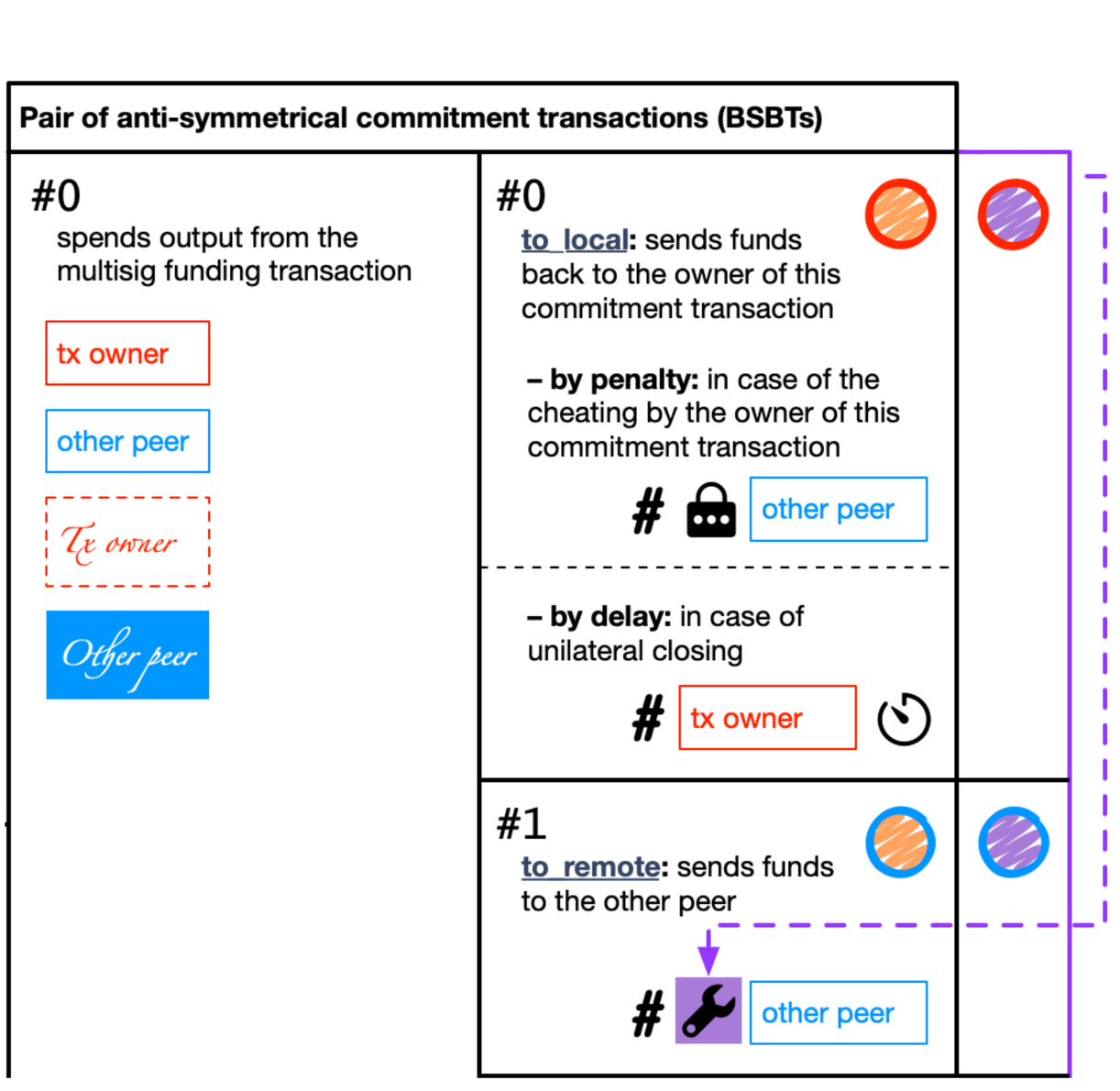
Derived public key requiring both parties

Hashed secret lock

Unlocking secret preimage

OP_CSV

OP_CTLV



Legend:

On-chain transaction

Partially-signed transaction

Possible future transaction

Offchain state

Bitcoins inside tx output

State bound to the tx output

Bitcoins or sealed state owned by the tx owner/ local party ("Alice")

Bitcoins or sealed state owned by the tx other peer / remote party ("Bob")

Local party:	Remote party:
Signature	Signature
Unsigned	Unsigned
Public key	Public key

\# Hash value of everything that follows after this sign

LNPBP-1 tweaked public key containing the state commitment

State commitment

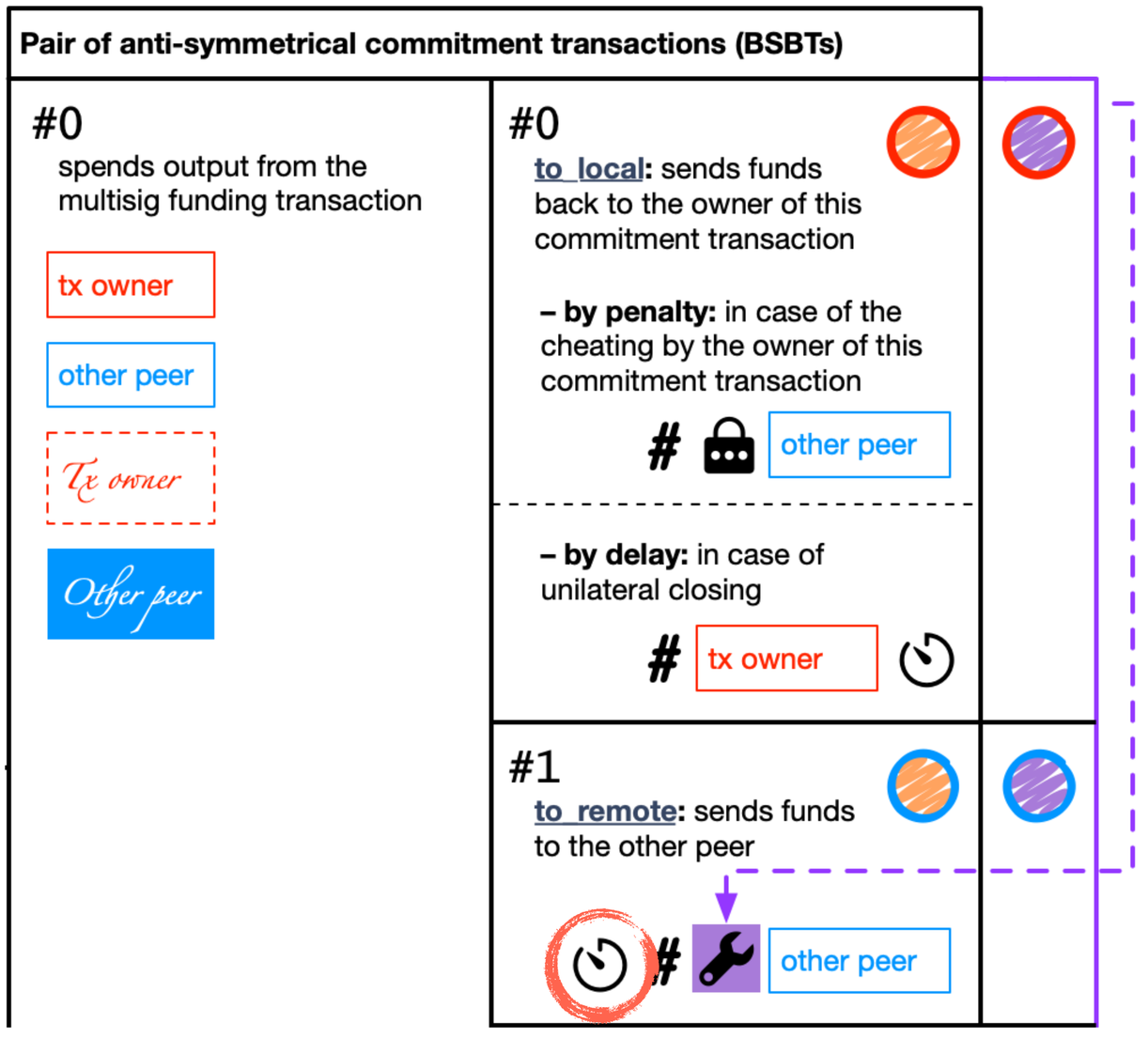
Derived public key requiring both parties

Hashed secret lock

Unlocking secret preimage

OP_CSV

OP_CTLV



Legend:

On-chain transaction

Partially-signed transaction

Possible future transaction

Offchain state

Bitcoins inside tx output

State bound to the tx output

Bitcoins or sealed state owned by the tx owner/ local party ("Alice")

Bitcoins or sealed state owned by the tx other peer / remote party ("Bob")

Local party:	Remote party:
Signature	Signature
Unsigned	Unsigned
Public key	Public key

\# Hash value of everything that follows after this sign

LNPBP-1 tweaked public key containing the state commitment

State commitment

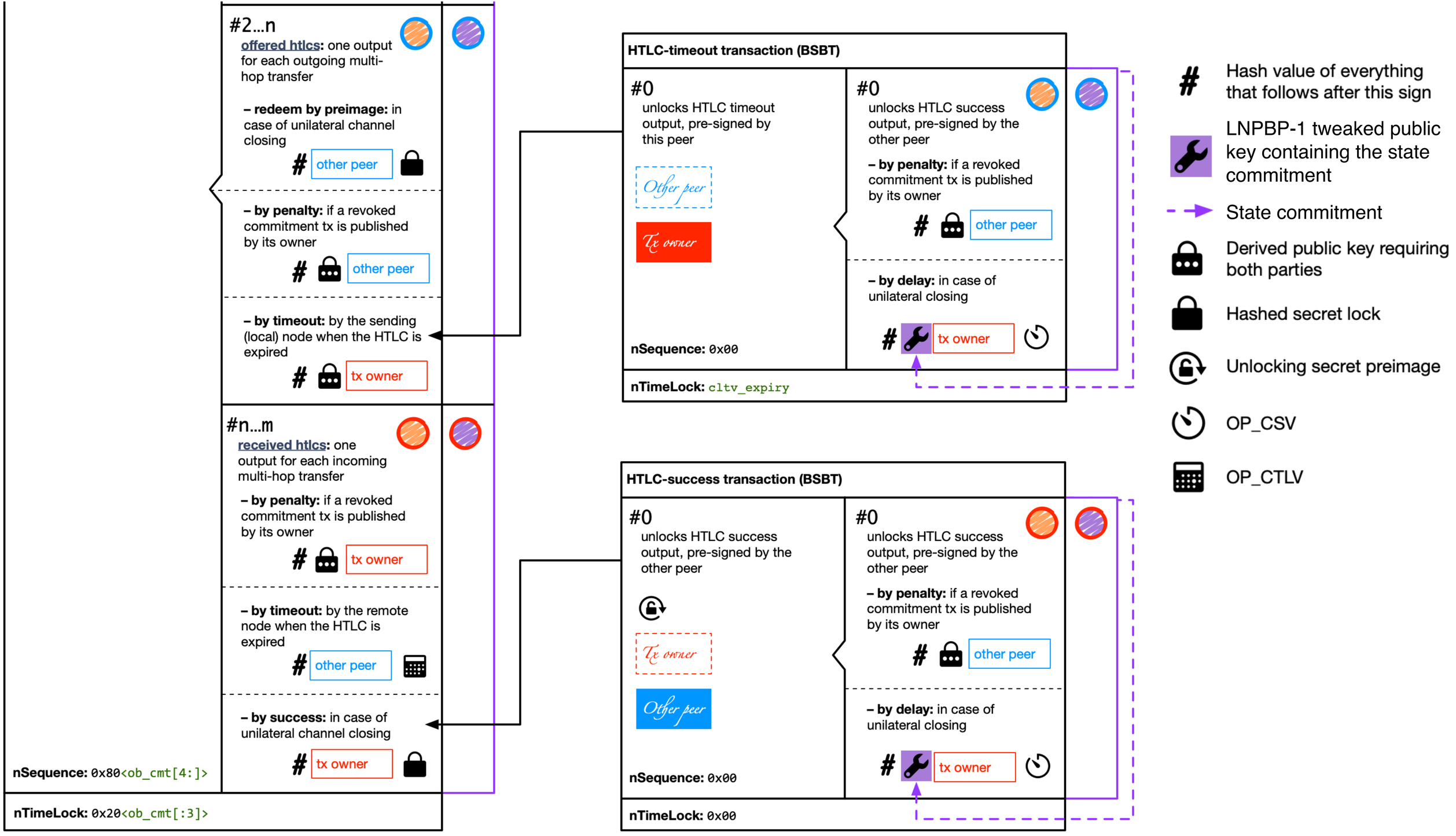
Derived public key requiring both parties

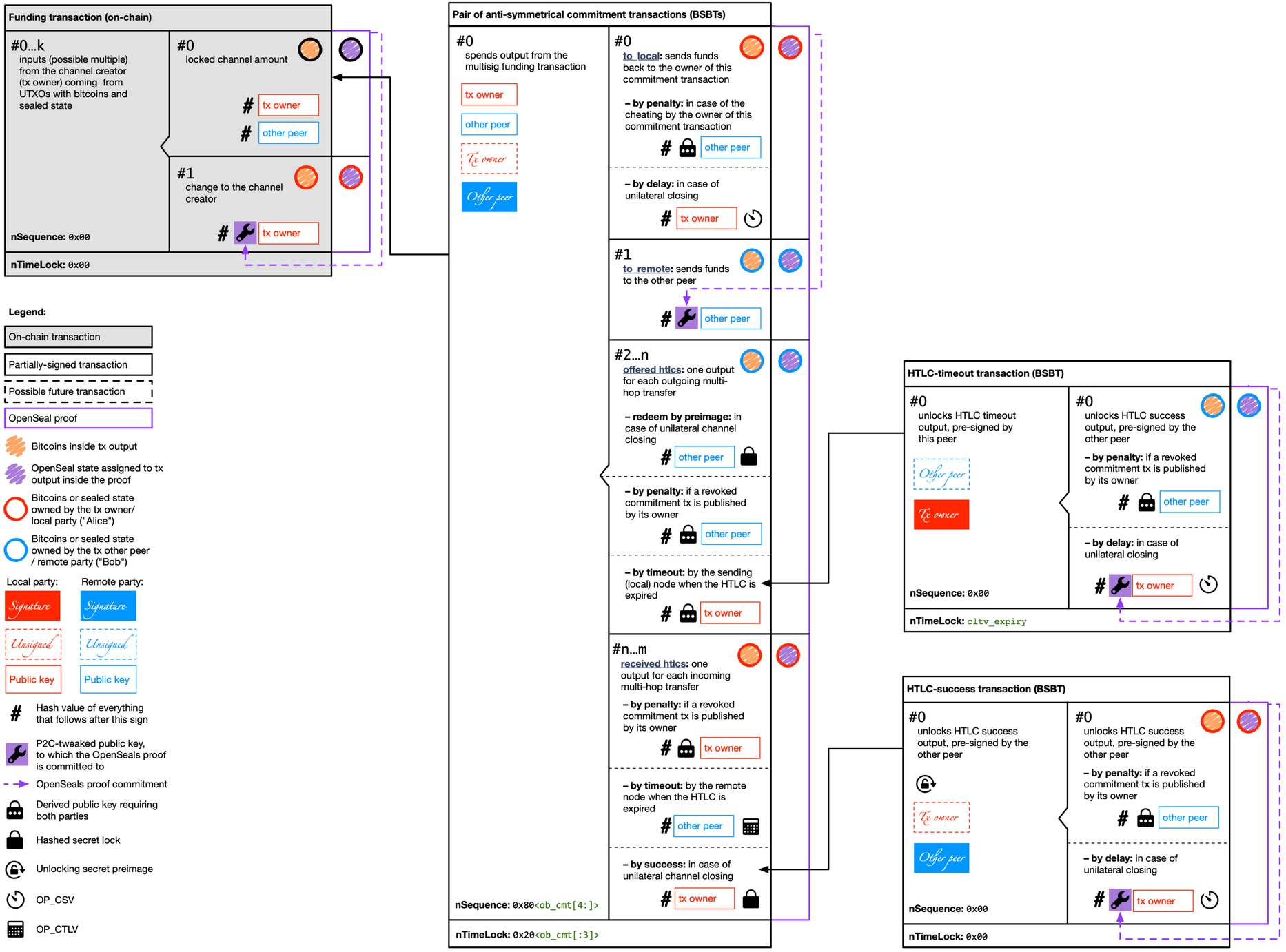
Hashed secret lock

Unlocking secret preimage

OP_CSV

OP_CTLV


#2...n
offered htlcs: one output for each outgoing multi-hop transfer
– redeem by preimage: in case of unilateral channel closing
other peer
– by penalty: if a revoked commitment tx is published by its owner
other peer
– by timeout: by the sending (local) node when the HTLC is expired
tx owner
#n...m
received htlcs: one output for each incoming multi-hop transfer
– by penalty: if a revoked commitment tx is published by its owner
tx owner
– by timeout: by the remote node when the HTLC is expired
other peer
– by success: in case of unilateral channel closing
tx owner
nSequence: 0x80<ob_cmt[4:]>
nTimeLock: 0x20<ob_cmt[:3]>
HTLC-timeout transaction (BSBT)
#0
unlocks HTLC timeout output, pre-signed by this peer
Other peer
Tx owner
nSequence: 0x00
#0
unlocks HTLC success output, pre-signed by the other peer
– by penalty: if a revoked commitment tx is published by its owner
other peer
– by delay: in case of unilateral closing
tx owner
nTimeLock: cltv_expiry
HTLC-success transaction (BSBT)
#0
unlocks HTLC success output, pre-signed by the other peer
Tx owner
Other peer
nSequence: 0x00
#0
unlocks HTLC success output, pre-signed by the other peer
– by penalty: if a revoked commitment tx is published by its owner
other peer
– by delay: in case of unilateral closing
tx owner
nTimeLock: 0x00
Hash value of everything that follows after this sign
LNPBP-1 tweaked public key containing the state commitment
State commitment
Derived public key requiring both parties
Hashed secret lock
Unlocking secret preimage
OP_CSV
OP_CTLV


Funding transaction (on-chain)
#0...k
inputs (possible multiple) from the channel creator (tx owner) coming from UTXOs with bitcoins and sealed state
#0
locked channel amount
tx owner
other peer
#1
change to the channel creator
tx owner
nSequence: 0x00
nTimeLock: 0x00
Pair of anti-symmetrical commitment transactions (BSBTs)
#0
spends output from the multisig funding transaction
tx owner
other peer
Tx owner
Other peer
#0
to_local: sends funds back to the owner of this commitment transaction
– by penalty: in case of the cheating by the owner of this commitment transaction
other peer
– by delay: in case of unilateral closing
tx owner
#1
to_remote: sends funds to the other peer
other peer
#2...n
offered htlcs: one output for each outgoing multi-hop transfer
– redeem by preimage: in case of unilateral channel closing
other peer
– by penalty: if a revoked commitment tx is published by its owner
other peer
– by timeout: by the sending (local) node when the HTLC is expired
tx owner
#n...m
received htlcs: one output for each incoming multi-hop transfer
– by penalty: if a revoked commitment tx is published by its owner
tx owner
– by timeout: by the remote node when the HTLC is expired
other peer
– by success: in case of unilateral channel closing
tx owner
nSequence: 0x80<ob_cmt[4:]>
nTimeLock: 0x20<ob_cmt[:3]>
HTLC-timeout transaction (BSBT)
#0
unlocks HTLC timeout output, pre-signed by this peer
Other peer
Tx owner
nSequence: 0x00
#0
unlocks HTLC success output, pre-signed by the other peer
– by penalty: if a revoked commitment tx is published by its owner
other peer
– by delay: in case of unilateral closing
tx owner
nTimeLock: cltv_expiry
HTLC-success transaction (BSBT)
#0
unlocks HTLC success output, pre-signed by the other peer
Tx owner
Other peer
nSequence: 0x00
#0
unlocks HTLC success output, pre-signed by the other peer
– by penalty: if a revoked commitment tx is published by its owner
other peer
– by delay: in case of unilateral closing
tx owner
nTimeLock: 0x00
Legend:
On-chain transaction
Partially-signed transaction
Possible future transaction
OpenSeal proof
Bitcoins inside tx output
OpenSeal state assigned to tx output inside the proof
Bitcoins or sealed state owned by the tx owner/ local party ("Alice")
Bitcoins or sealed state owned by the tx other peer / remote party ("Bob")
Local party:
Remote party:
Signature
Signature
Unsigned
Unsigned
Public key
Public key
Hash value of everything that follows after this sign
P2C-tweaked public key, to which the OpenSeals proof is committed to
OpenSeals proof commitment
Derived public key requiring both parties
Hashed secret lock
Unlocking secret preimage
OP_CSV
OP_CTLV

Can we do CVS over LN?

- Embed commitments into P2WSH - and later P2TR: done
- Embed new data into messaging protocol:
  solved with TLV extensions
- Negotiate state included in the channel on opening:
  local feature bits + message extensions
- Find a way to publish information on the balances with gossip protocol: global feature bits + message extensions
  (not yet fully supported in gossips)
- Routing: does not require changes
- Restoration: maintain all state proofs on channel updates

Problem #1

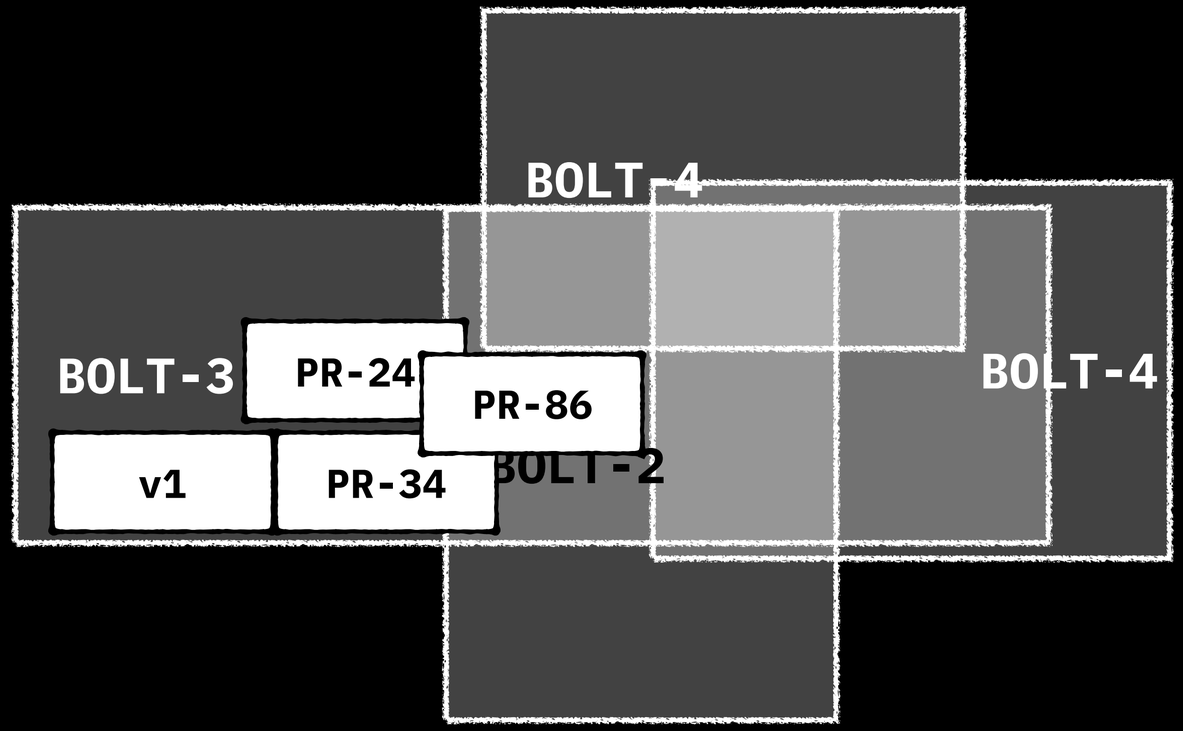
- Really hard to integrate the required functionality into the existing LN node software: no proper extension mechanics
  - Fork: cost of maintenance
  - Do own node: cost of implementation & maintenance - but puts us on the LN political map
  - Create a new extension standard
  - Partner with existing node development company

Solution #1

- Partner with existing node development company: Blockstream/c-lightning
- Work on a new extension standard later

Problem #2

specs as should be



BOLT specs

Solution #2

- LNP/BP standardisation effort will also cover important aspects of better Lightning network standards

Problems #N: Readiness for future updates

- **P2WSH to_remote**: ready (LNPBP-2)
- **Changes in gossip protocols**: high risk for both RGB & Spectrum
- **Changes in routing protocols**: unknown risk, high in some cases (Spectrum with Trampoline)
- **Schnorr, taproot**: ready (LNPBP-2)
- **DLS**, or pay to curve point: no changes required
- **eltoo**/SIGHASH_NOINPUT: no changes required, we just need to get rid of the historical CVS/RGB data, no channel update

Building components

- Can be done with **L1** alone, but needs LN to scale:
  - Cryptographic commitments
  - Client-side validated state
- Can’t be done without Lightning Network:
  - Onion-routed p2p messaging
  - Network-wide gossip messaging

Messaging over Lightning Network for RGB

- Announce support for:
  - RGB – local and global features, init + gossips
  - List of supported assets (=genesis states) – init + gossips
- We need TLV extensions to be allowed in gossips
- Utilise TLV extensions for data transfer in peer and onion-routed messages
- Transfer large client-validated data blobs over the separate communication channel

RGB-enabled channel operation

1. Upon establishing connection, include a feature bits indicating the RGB support into the init message.

2. If the remote node supports CVS it will connect over dedicated link to receive state history information for validation.

3. The nodes SHOULD persist the information on which states are supported by their connected peers.

4. If the node starts supporting new state history roots, it MUST close and re-open its connections.

Channel establishment and updates

- **init**: odd RGB feature bit
- **open_channel**: TLV extended data section with
  - the list of CVS genesis states that the node plans to include into the channel
  - the owned state for each of them
  - the RGB connection details for transferring the proofs
  - the remote party must decline the channel if it contains any CVP genesis that is not supported by it, or if it was not convinced by proofs
- **accept_channel**: the same data structure as was in the *open_channel*
- No additional information is included into the **funding_created**, **funding_signed** and **funding_locked** messages
- **update_add_htlc**: TLV extension with the information required to deterministically reconstruct new version of the state transition and commit to it in the transaction output

Gossip protocol

- **node_announcement**, **channel_announcement** messages will contain:
  - Feature bits
  - TLV extension listing supported genesis states
- **channel_update** messages will contain:
  - TLV extension with updated listing for the supported genesis states
  - The fees will be taken in millisatoshis

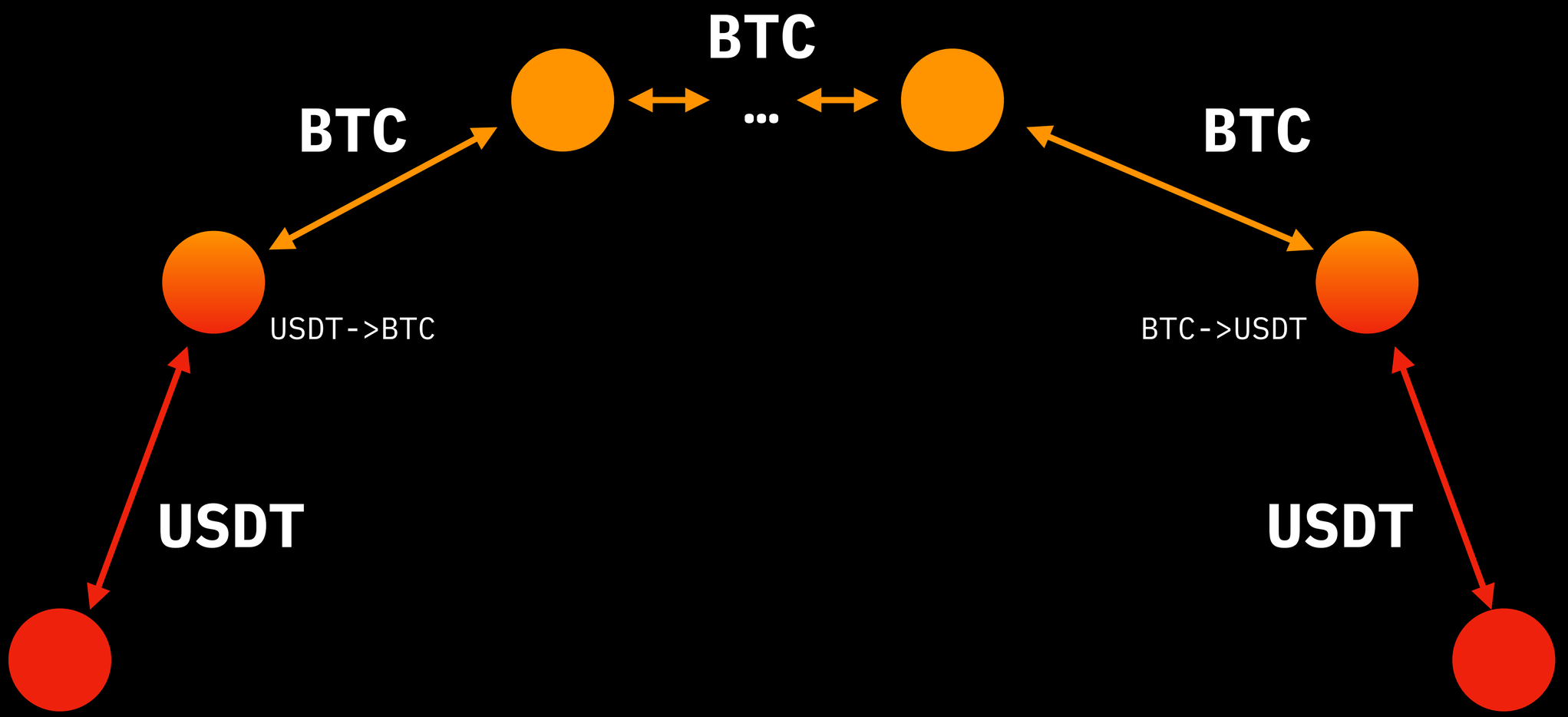
Onion messaging

- Extra information for the receiver is included in the form of TLV extension
- Intermediate nodes will not need to support RGB, since they will not see the end message

Part III: Spectrum

The DEX done the right way

Spectrum: DEX



Spectrum: Decentralised Exchange

- Uses the gossip protocol for ask/bid price information
- Liquidity providers help multi-hop payments
- New monetisation model for Lightning Network nodes
- Decentralized secondary markets
- ***Disrupts Lightning Network transaction analysis***

Spectrum

- Announce support for Spectrum with odd feature flag
- Announce pairs of supported genesis states and bid/ask prices in gossip messages
- Sender should construct the route taking into account this information
- Requirements for exchanges should be embedded into onion message packet and read by Spectrum-enabled nodes

Simple?

- As of today, requires basically no changes to LN, will work out of the box, however:
  - Reverse American call option problem
- Tomorrow may be broken by:
  - Trampoline
  - New gossip protocol limitations

RGB and Spectrum do not require changes
into LN protocols;
we just need to built software that
covers all additional functionality and
integrates with LN nodes

Part IV: Software and architecture

Components designed by LNP/BP Standards Association

- **LNPBP Core Library**: reference implementation of LNPBPs standards. “Code is the spec”, while the standards are just a more readable version of it with some explanations.
  - Strong test coverage, detailed docs
  - Special WASM build
  - LNP-BP Core Library C bindings allowing to call it from: C/C++ | Python | Go | NodeJS | JVM-based languages (Java, Scala, Kotlin…)

https://github.com/lnp-bp/rust-lnpbp

Components designed by LNP/BP Standards Association

- **LNP-BP Daemons**: suite of daemons with micro service architecture and ZMQ inter-process communications implementing the complete set of LNP/BP and RGB protocols
  https://github.com/lnp-bp/lnpbpd

- **Wallet Wrappers**: WASM/JavaScript, Swift and Kotlin wrappers around LNPBP Core Library C bindings for simple usage in mobile and web wallet software

Components designed by LNP/BP Standards Association

- **Asset Schemata**: Schema for main RGB state types: Assets, collectibles, identity, ...
- **Simplicity Core Lib**: a set of smart contract script primitives planned to be developed (potentially with Blockstream)

Components used and contributed by LNP/BP Standards Association

- **rust-libsecp256**, based on libsecp256
- **rust-libsecp256-zpk**: bulletproofs and Pedersen commitments
- **bitcoin_hashes**: hash function primitives
- **rust-bitcoin**: bitcoin primitives
- **rust-lightning**: lightning network primitives
- **c-lightning**: c-lightning node

We welcome development of

- RGB-enabled **Explorers**
  - ***Rainbow Esplora*** *– a Blockstream Esplora fork made by Datagnition with RGB support*
  - *?*
- **Wallets**
  - ***Eidoo*** *– a wallet by* Poseidon Group
  - ***Globular*** *– a wallet by* inbitcoin
  - ***Bluewallet*** *– a well-known wallet for iOS*
  - ***The Dark One*** *– a new wallet from* Datagnition*, “Keybase for cypherpunks”: LN, IRC messaging, RGB-based identity management, RGB assets, Taproot, Storm, Prometheus*
  - ***Wasabi wallet****? – world-famous privacy wallet with CoinJoin*

We welcome development of

- **Command-line tools**
  - ***lbx***: *command-line Swiss knife implementing all LNPBP operations.*
    https://github.com/lnp-bp/lbx
    Inspired by:
    bx *from* libbitcoin *by* Eric Voskuil *and*
    hal *from* rust-bitcoin *by* Steven Roose

- RGB- and Spectrum-enabled **Lightning nodes**
  - *c-lightning fork in cooperation with Blockstream*
  - *rust-based lightning node in cooperation with Chaincode*

Components status

	Language	Current contributors	Repositories
LNPBP Core Library	Rust	Pandora Core	lnpbp/rust-lnpbp
CLI	Rust	Pandora Core	lnpbp/lbx
LNPBP Daemons	Rust + wrappers	Pandora Core, Bitcoin Core, Blockstream, Chaincode	lnpbp/lnpbpd
Wallet wrappers	WASM, Swift, Kotlin, JavaScript	Hyperdivision, Pandora Core, Datagnition	*vendor-specific*
Asset schemata	CVS Schema	Pandora Core, inbitcoin, Chainside	n/a
Simplicity core lib	Simplicity	(Blockstream?)	?
Wallet (The Dark One)	Swift	Datagnition	*planned*
Explorer (Rainbow Esplora)	JavaScript	Datagnition	datagnition/esplora

Rust language

- Deterministic execution of the compiled program
- No garbage collection
- Generics system most close to C++
- C interoperability (bindings for C, Python, Swift, Java/Kotlin, Go)
- Most advanced WASM toolchain

Project modules / libraries

RGB: assets

client-side validated state graphs

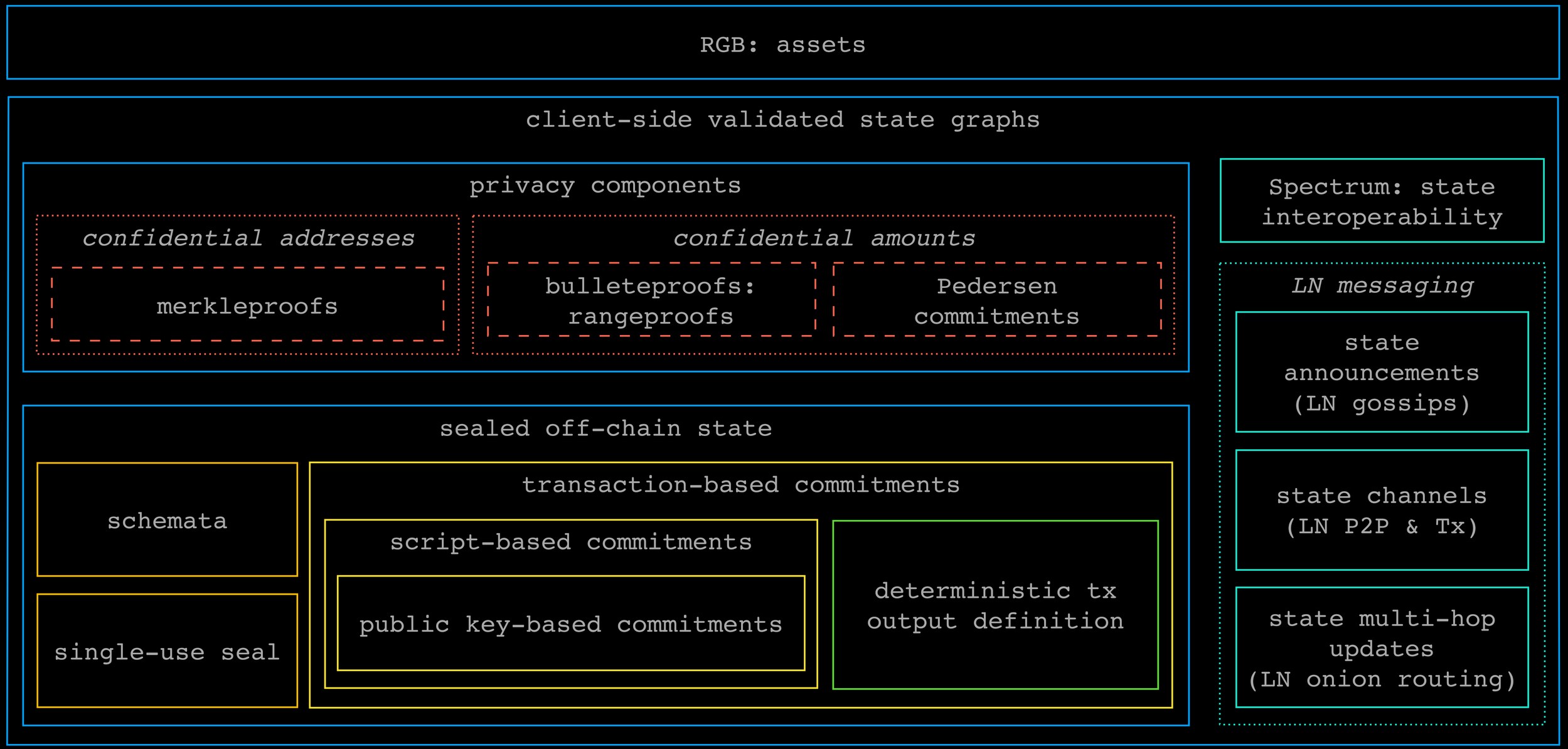
privacy components

sealed state and schemata

Spectrum: state interoperability

LN in-channel state information

Project modules / libraries



Rules for commitment serialisation

- Do not compress the data
- Use deterministically-defined value length
  (Bitcoin’s VarInt is a bad practice)
- No pointers/offsets/shifts, no linked lists
- Merkle trees must also commit to the depth of each branch
- Define bounds for each type validity
- Must be composed of the nested digests (prevention of length-extension attacks)
- Must be prefixed with protocol-specific tag before commitment
- First 8 bytes must deterministically define the length of the committed data

Runtime, serialisation and commitments

```
mod serialize {
    mod commitment {
        pub struct Transition<const N: usize, const M: usize> {
            pub no: u32,
            pub merkle_root: MerkleRoot,
            pub seal: [SealHash; N],
            pub state: [StateHash; N],
            pub meta: [MetaHash; M],
            pub script: ScriptHash,
        }
    }
}
```

```
mod runtime {
    pub struct Transition {
        pub commitment_source: serialize::commitment::Transition,
        pub known_seal_state: Vec<self::SealState>,
        pub known_seal_meta: Vec<self::MetaRecord>,
        pub script: Option<self::SimplicityScript>,
    }
}
```

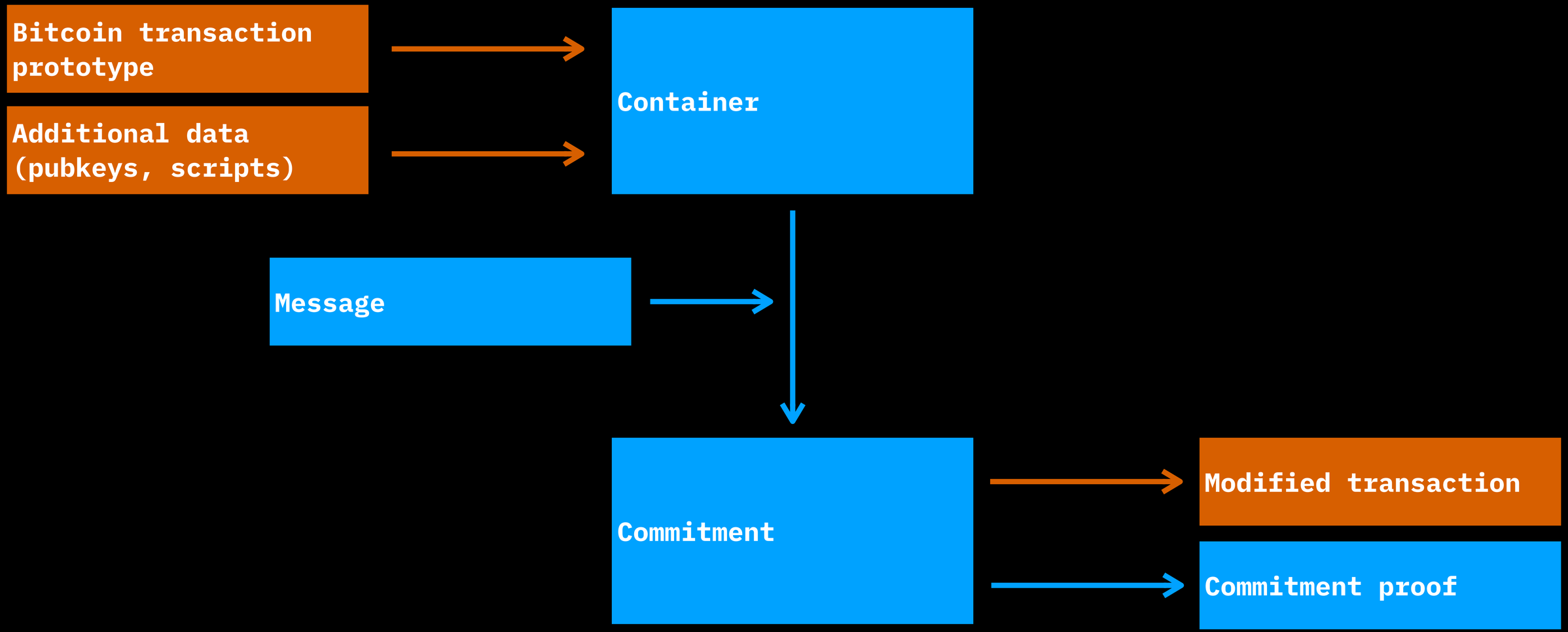
```rust
mod runtime {
    pub struct Transition {
        pub commitment_source: serialize::commitment::Transition,
        pub known_seal_state: Vec<self::SealState>,
        pub known_seal_meta: Vec<self::MetaRecord>,
        pub script: Option<self::SimplicityScript>,
    }

    pub trait StorageSerialize {
        fn storage_serialize(&self, stream: &io::Stream) -> Result<(), io::Error>;
        fn storage_deserialize(stream: &io::Stream) -> Self;
    }

    pub trait NetworkSerialize {
        fn network_serialize(&self, stream: &io::Stream) -> Result<(), io::Error>;
        fn network_deserialize(stream: &io::Stream) -> Self;
    }

    impl StorageSerialize for Transition { /* ... */ }
    impl NetworkSerialize for Transition { /* ... */ }
}
```

Transaction commitment system: *commit*



Containers *vs* Commitments sample

```rust
#[derive(Clone, Eq, PartialEq)]
pub struct TxContainer {
    pub entropy: u32,
    pub tx: Transaction,
    pub fee_output: u32,
    pub txout_container: TxoutContainer,
}

#[derive(Clone, Eq, PartialEq)]
pub struct TxCommitment {
    pub entropy: u32,
    pub tx: Transaction,
    pub tweaked: TxoutCommitment,
    pub original: TxoutContainer,
}
```

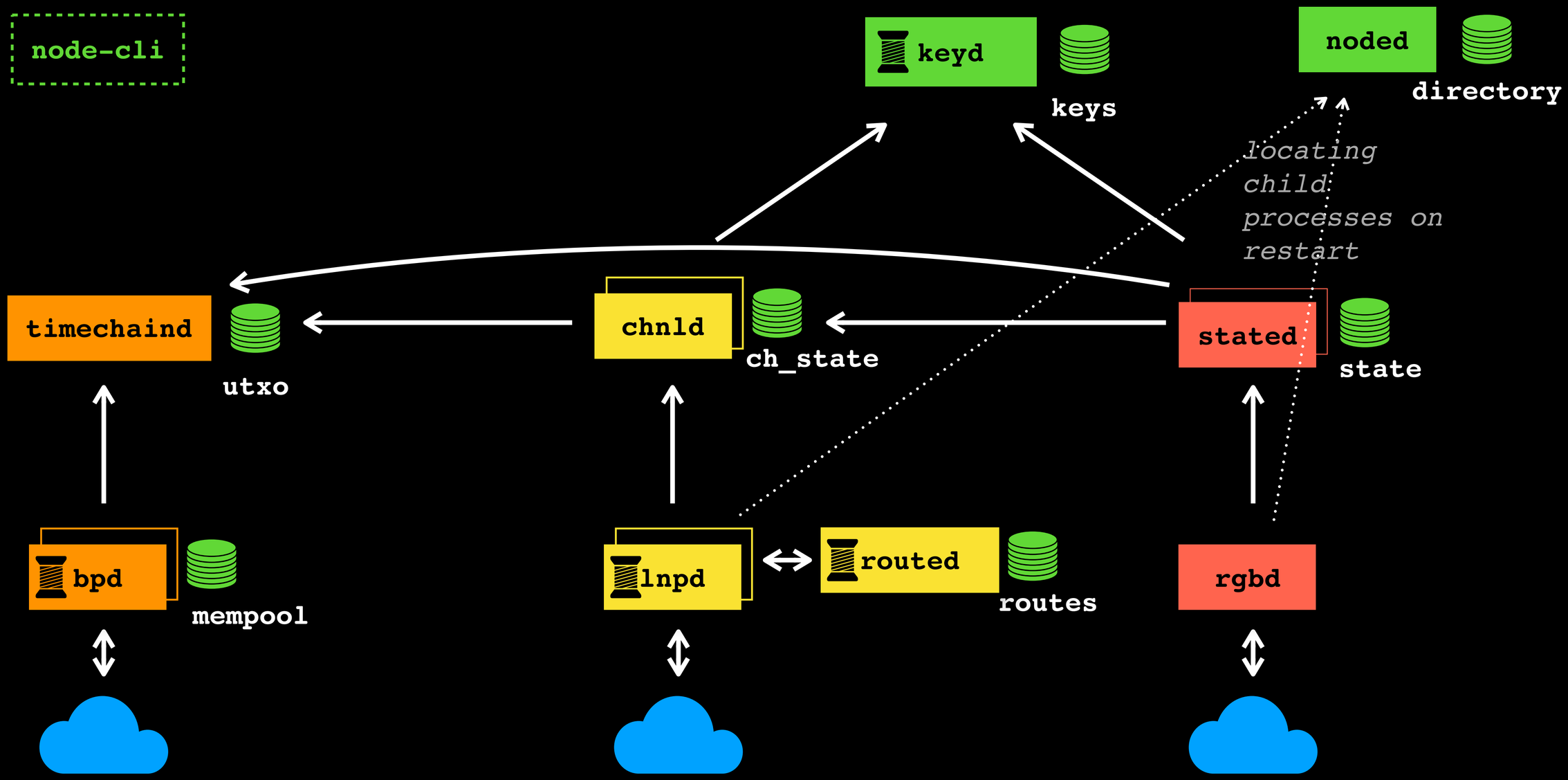
RGB architecture options

- Microservice-based (*vs* monolithic in Bitcoin/lnd/eclair and multiprocess in c-lightning)
- IPC communications via ZeroMQ; no JSON-RPCs and other outdated staff
- Authentication with Macaroons
- Re-using existing daemons, services, components

Why not to extend this to the whole LNP/BP tech stack?

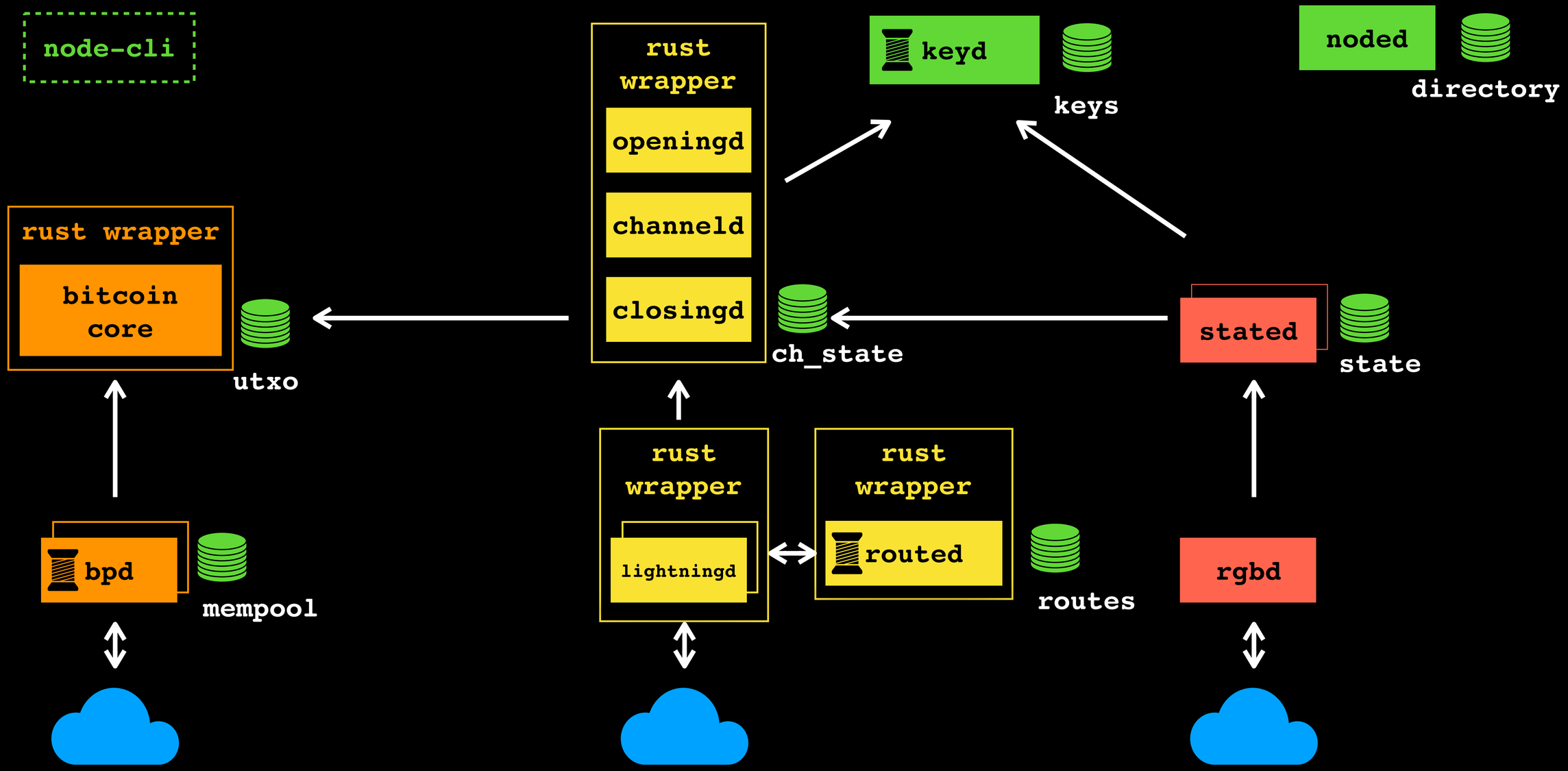
The actual effort may not be that large comparing to the effort to fight/support outdated standards and poorly-designed historical architectures

Let’s imagine the world where everything
is done the right way...

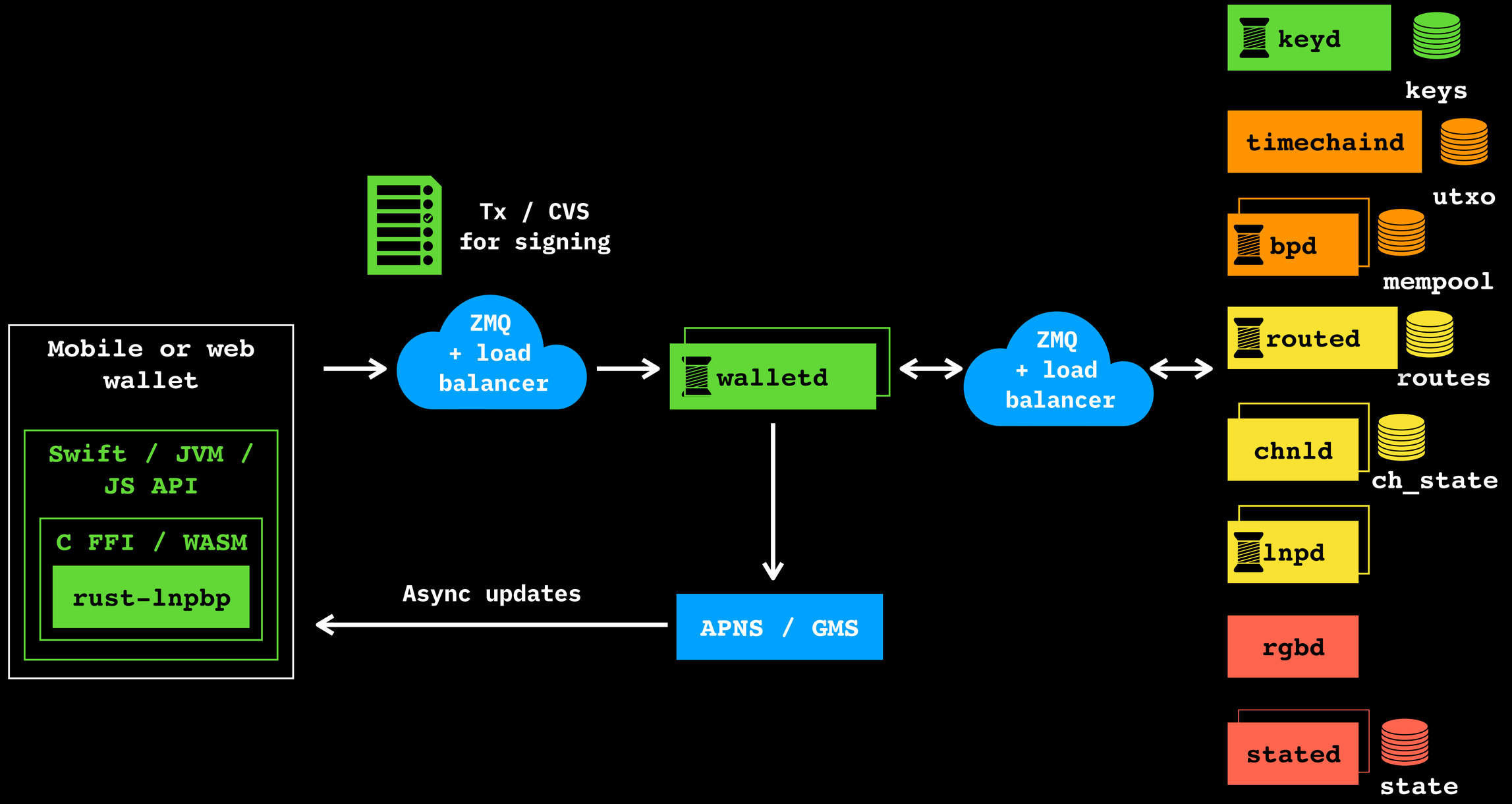

node-cli
keyd
keys
noded
directory
locating child processes on restart
timechaind
utxo
chnld
ch_state
stated
state
bpd
mempool
lnpd
routed
routes
rgbd

But can we get there without dramatic changes today?

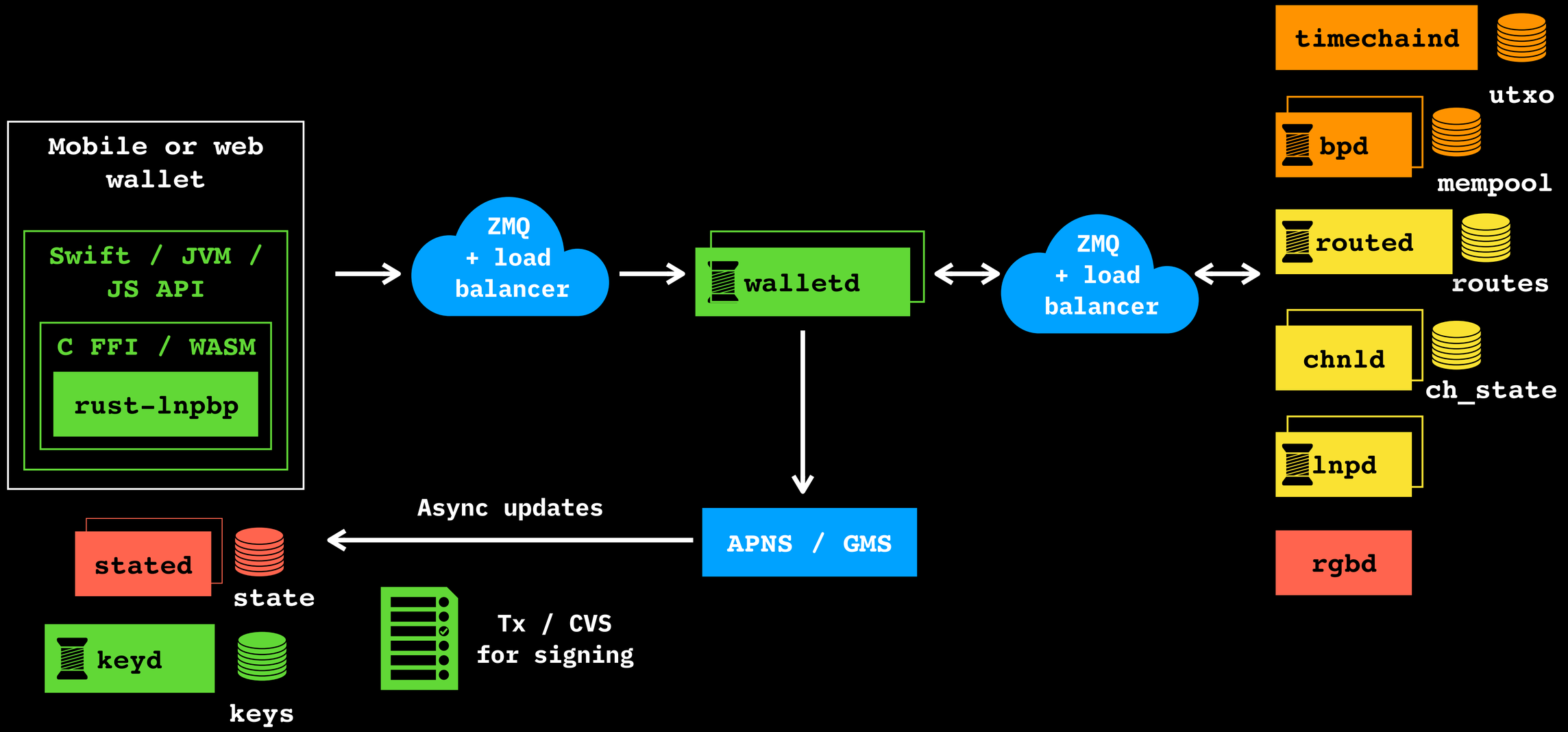
— Yes!


node-cli
rust wrapper
openingd
channeld
closingd
keyd
keys
noded
directory
rust wrapper
bitcoin core
utxo
ch_state
stated
state
bpd
mempool
rust wrapper
lightningd
rust wrapper
routed
routes
rgbd

Custodial clients/wallets



Non-custodial clients/wallets



Part V: Conclusions

Project development streams

- **Core RGB** based on client-side validation paradigm

  security-critical part
  core reviewers: Peter Todd, Andrew Poelstra, maybe Peter Wuille

- **Lightning Network RGB & Spectrum** based on gossip & onion routing

  scalability-critical part
  core reviewers: Christian Decker, maybe Alex Bosworth

- **Confidential Assets** interoperability

  privacy- and interoperability-critical part
  core reviewers: Andrew Poelstra, Adam Back

We have to launch all streams together
after thorough review and verification

Project development risks

- **Core RGB:**

  Peter Todd input is critical in security assessment;
  multiple re-designs of protocol structure

- **Lightning Network:**

  immature LN functionality will require constant re-integration
  need to implement & support own LN (sub)node

- **Confidential Assets:**

  complexity of zero-knowledge components
  risks of hidden inflation

Detected & addressed challenges so far

- **Fitting RGB into LN in standards-compliant way**
  - ➙ final parts of the problem were solved with the latest BOLT changes discussed and accepted during the Lightning Conference

- **LN instability:** the protocol will undergo drastic unpredictable changes at least every quarter
  - ➙ Modularise and create proper abstractions within RGB and Spectrum, increasing its flexibility and decreasing risks: if something becomes incompatible it can be replaced as a module
  - ➙ Work with Christian Decker & community on a more strict & fixed LN standards and rust node (also see the next point)
  - ➙ Change a number of previously-defined RGB parts (e.g. commitments)

Detected & addressed challenges so far

- **RGB can’t be implemented as just an extension to Lightning Nodes due to architecture limitations**
  - → Together with Christian Decker we found a way to avoid the need for having a completely maintained c-lightning fork and step-by-step build a rust-node attached to a mainstream c-lightning implementation
  - → Chaincode and Wasabi wallets may co-operate on this development
  - → It will take months to do the necessary changes even outside of RGB/Spectrum functionality :(

Detected & addressed challenges so far

- **Confidential assets interoperability**
  - → Adam Back is willing to explore the potential of a joint asset standard for Bitcoin, Liquid and LN with RGB+CA joint spec
  - → Client-side validation paradigm can be very useful for Liquid, since it delivers both better privacy and scalability
  - → With Andrew Poelstra we need to explore the space of possible options
  - → I have prepared a draft, but much more work is still required

Project development status

- **Core RGB:**

  50% of specs;
  50% of code;
  3 previous spec iterations and 4 implementations have been discarded.

- **Lightning Network:**

  50% of final specs;
  all issues and problems are analysed and we know how to address them;
  design for a c-lightning fork and further maintenance.

- **Confidential Assets:**

  generic design for embedding zero-knowledge & building CA compatibility;
  Blockstream awareness and openness to work on generic industry standard for assets.

What is currently required

- **Core RGB:**

  Peter Todd: new review on single-use seals specs & code.

- **Lightning Network:**

  me & Christian Decker: refactor c-lightning architecture & specs;
  get full-time developer;
  me & Chaincode: work on rust implementation of rgb-specific daemons for c-lightning.

- **Confidential Assets:**

  work with Andrew Poelstra on finalising CA design;
  find a solution for protecting confidentiality of asset types in multi-asset transfers.

TL;DR

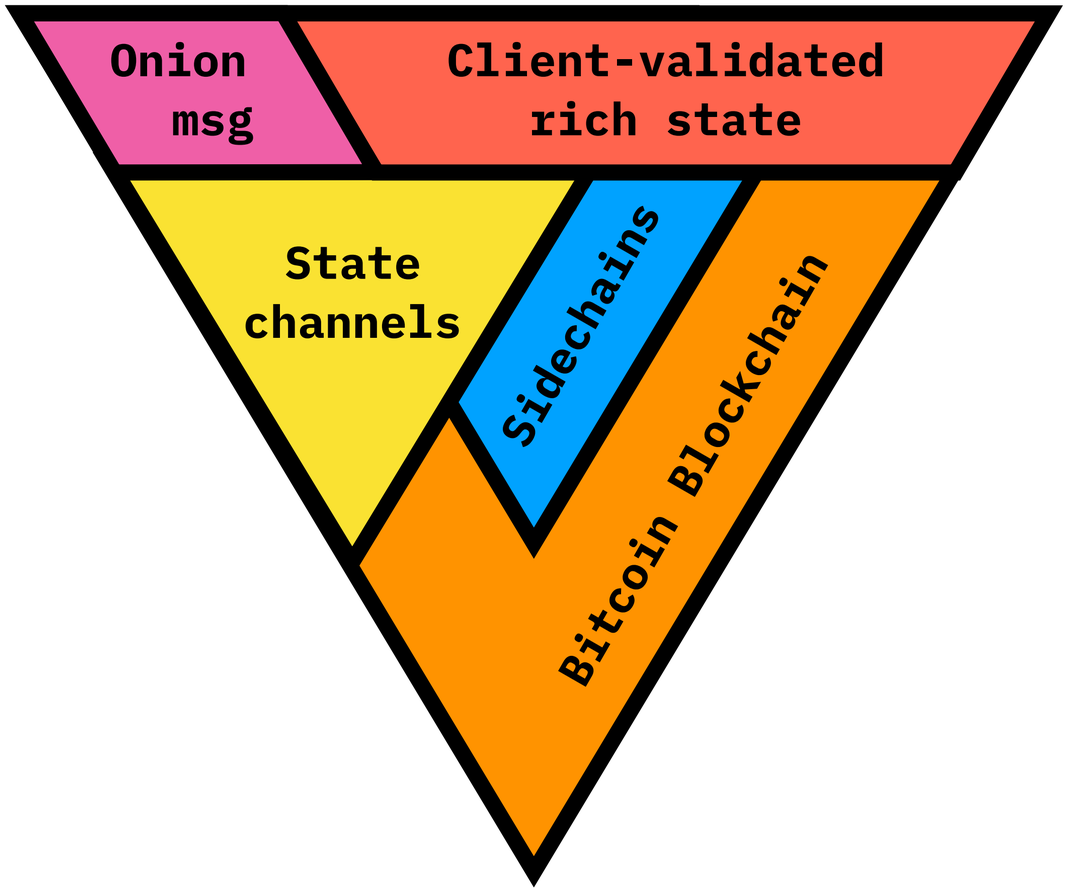
- **Positive things:**
  - ★ Industry-wide involvement
  - ★ High potential for common standard with next version of Confidential Assets
  - ★ Better privacy features than were expected originally
  - ★ Many security risks and potential vulnerabilities have been mitigated
  - ★ Better modularisation and layerisation that simplify future flexibility
  - ★ Found a way to implement LN-specific parts; conflicts with standards have been mitigated
- **Negative things:**
  - ★ Still a lot of stuff to do and develop
  - ★ Many external risks with LN internal instability
  - ★ Much more work on LN than was expected (need to re-implement part of LN node functionality)
  - ★ We will be very restricted in what we can upgrade after the first release, so we need to deliver 100% working and tested product from day 1 in production

Part VI: Bigger picture

RGB “owned state” allows better smart contracts

- Bitcoin-based basic ownership rules
- Programmable state: assets, games, identity, reputation etc
- No on-chain space usage
- Off-chain storage sharking
- Strong privacy with multiple layers of zero-knowledge
- Works over lightning, scalable
- Embedded DEX with Spectrum
- Off-chain Turing completeness and formal verification with Simplicity to apply additional rules for state transitions

“Ethereum 2.0 made right”


Onion
msg
Client-validated
rich state
State
channels
Sidechains
Bitcoin Blockchain

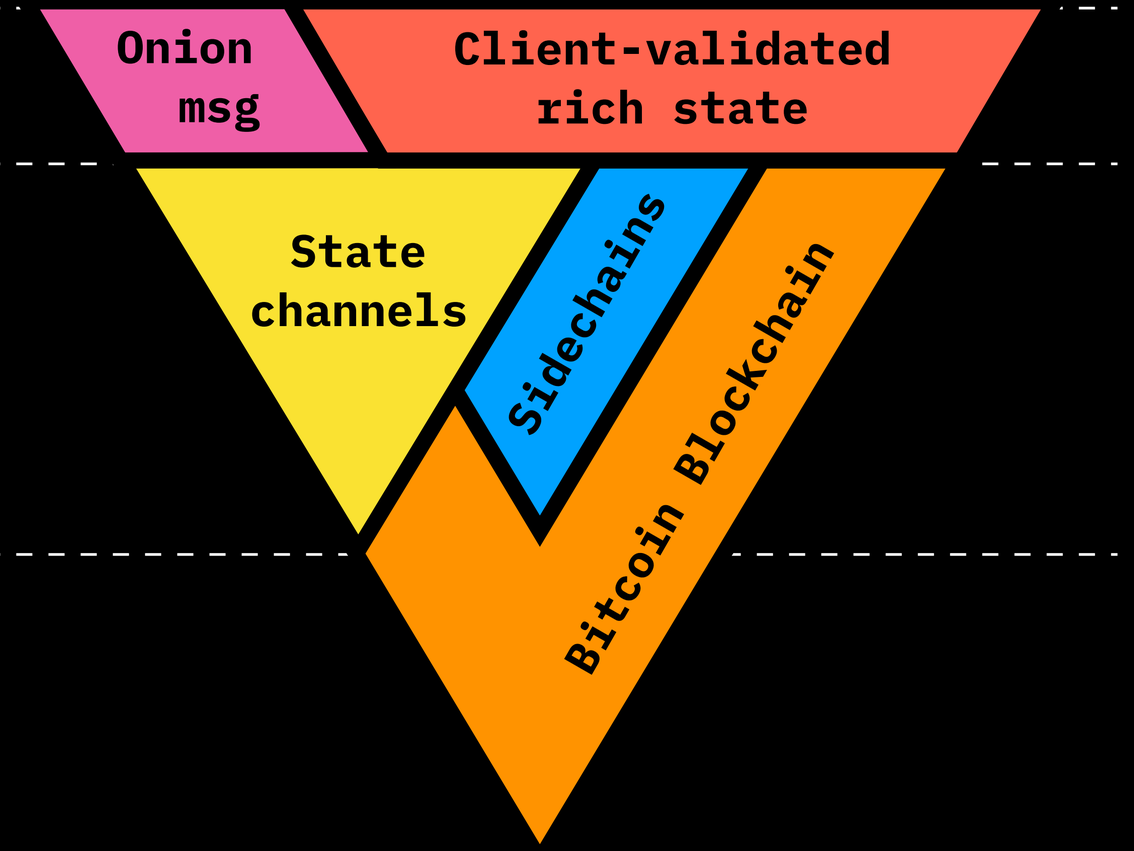
LNP BP Standards Association

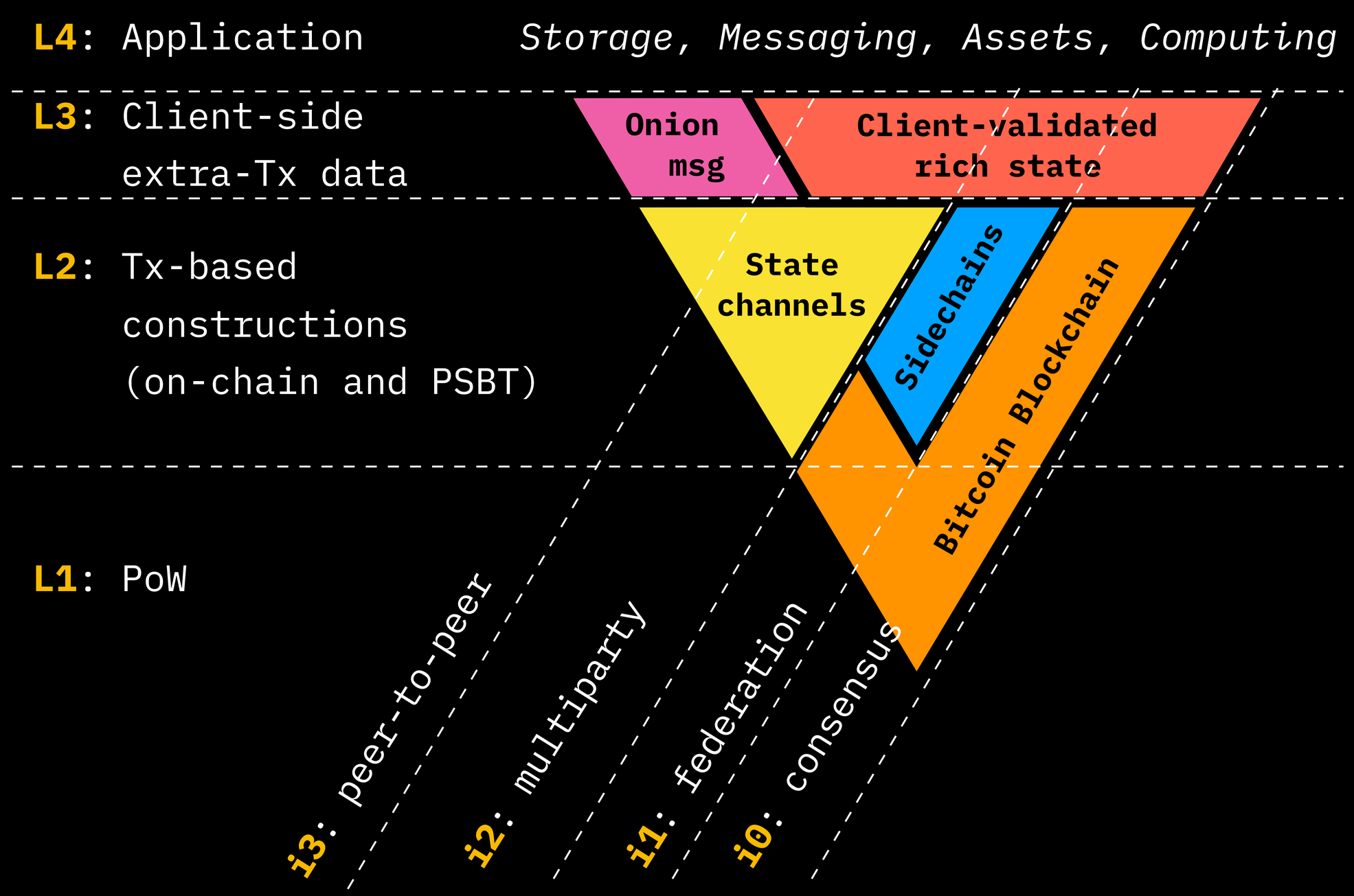
L4: Application *Storage, Messaging, Assets, Computing*

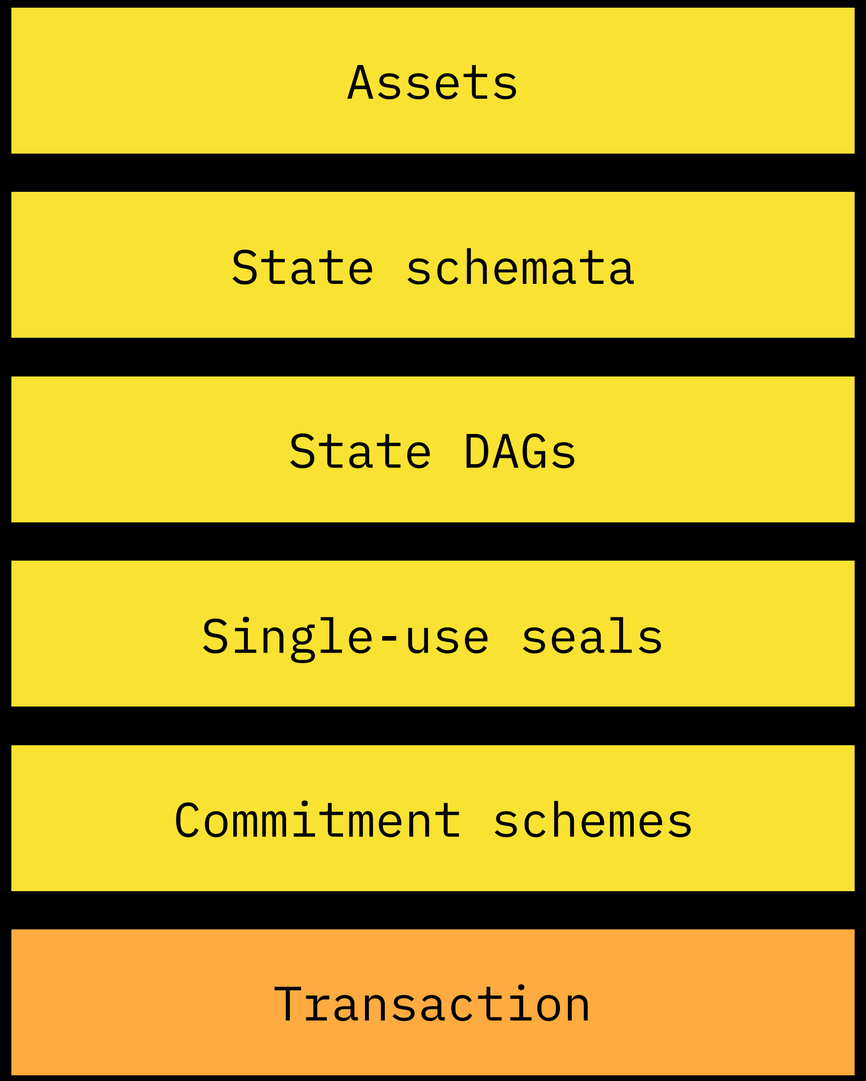
L3: Client-side extra-Tx data

L2: Tx-based constructions (on-chain and PSBT)

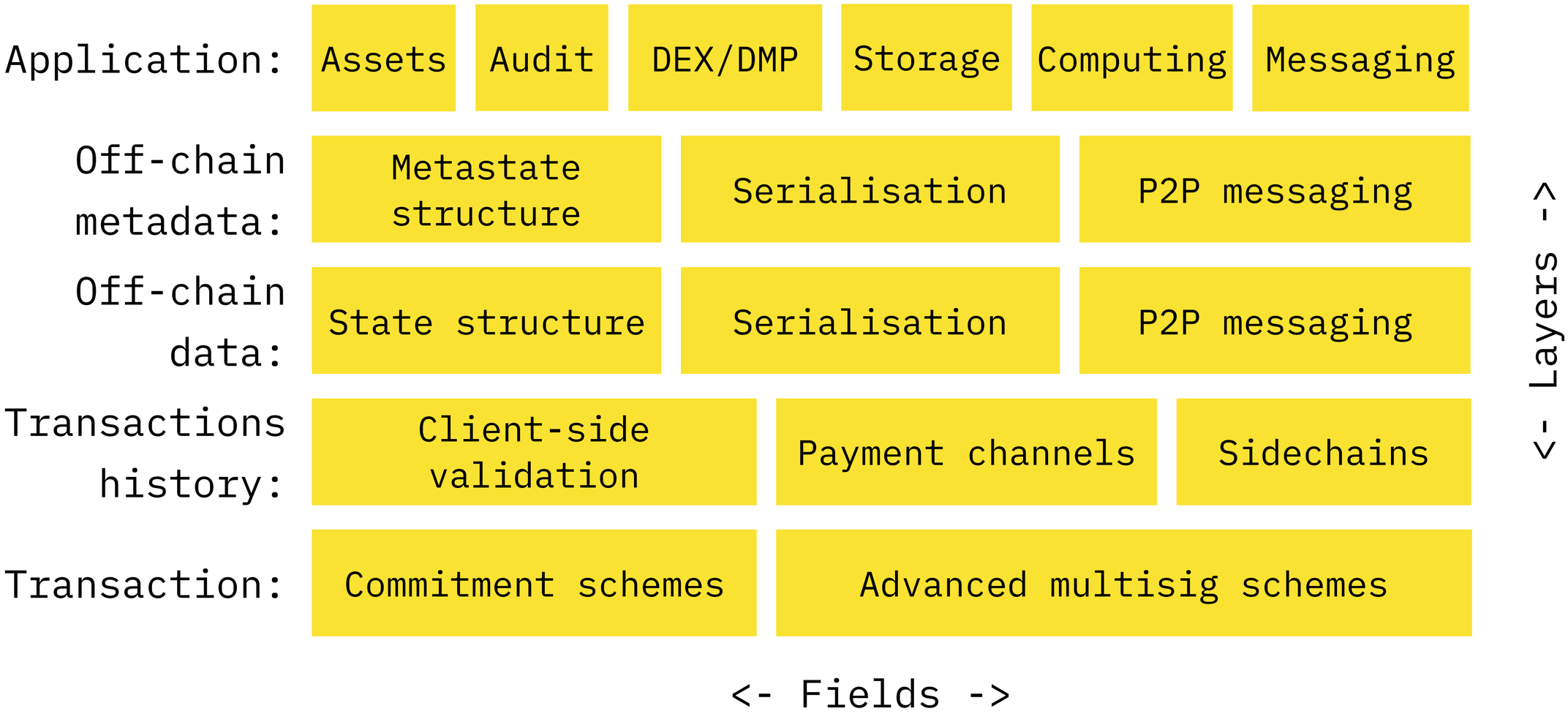
L1: PoW




L4: Application
Storage, Messaging, Assets, Computing
L3: Client-side extra-Tx data
L2: Tx-based constructions (on-chain and PSBT)
L1: PoW
Onion msg
Client-validated rich state
State channels
Sidechains
Bitcoin Blockchain
i3: peer-to-peer
i2: multiparty
i1: federation
i0: consensus


Assets
State schemata
State DAGs
Single-use seals
Commitment schemes
Transaction

LNP/BP Tech Stack Structure



LNP/BP Standards (outside of BIP scope)

GitHub, Inc. (US) | https://github.com/lnp-bp/lnpbps

LNP-BP / lnpbps

Unwatch 2 | Unstar 3 | Fork 0

Code | Issues 2 | Pull requests 0 | Projects 0 | Wiki | Security | Insights | Settings

LNP/BP Specifications

Edit

bitcoin | lightning-network | decentralization | distributed-systems | privacy | cryptography | Manage topics

3 commits | 1 branch | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

dr-orlovsky LNPBPs-0001 (early draft): Cryptographic commitments with public key ... Latest commit 55b573c 17 days ago

assets	README: Project description, inclusion criteria, layers and initial l...	17 days ago
.gitignore	README: Project description, inclusion criteria, layers and initial l...	17 days ago
README.md	LNPBPs-0001 (early draft): Cryptographic commitments with public key ...	17 days ago
lnpbps-0001.md	LNPBPs-0001 (early draft): Cryptographic commitments with public key ...	17 days ago

README.md

LNP/BP Specifications

LNP/BP stands for "Bitcoin Protocol / Lightning Network Protocol". This set of specifications covers standards & best practices for Layer 2, 3 solutions (and above) in cases when they do not require soft- or hard-forks on the Bitcoin blockchain level and are not directly related to issues covered in Lightning Network RFCs (BOLTs).

https://github.com/lnp-bp

Number	Layer	Field	Title	Owner	Type	Status
1	Transaction (1)	Cryptographic primitives	Cryptographic commitments with public key tweaking	n/a	Standard	Draft
2	Transaction DAG (2)	Client-side validation	Simple single-use seal for LNP/BP	n/a	Standard	Draft
3	Offchain data (3)	Consensus rules	State history directed acyclic graphs on Bitcoin	n/a	Standard	Draft
4	Offchain data (3)	Serialization	Serialization for state history DAGs, LNPBPs-4	n/a	Standard	Draft
5	Offchain metadata (4)	Serialization	Schemata for rich state	n/a	Standard	Draft
6	Application (5)	Assets	RGB, part 1: Fungible centrally-issued assets with client-side validation	n/a	Standard	Draft
7	Offchain data & metadata (3-4)	P2P messaging	State announcements for Lightning Network gossip protocol	n/a	Standard	Draft
8	Offchain data & metadata (3-4)	P2P messaging	State updates over Lightning Network onion messaging	n/a	Standard	Draft
9	Application (5)	DEX/DMP	Spectrum: decentralized market / exchange over Lightning Network	n/a	Standard	Draft
10	Offchain metadata (4)	Assets	RGB, part 2: Zero-knowledge proofs for asset transfers	n/a	Standard	Draft

LNP/BP Association supervises:

- Client-validated owned state, including assets — **RGB**
- Decentralized exchange (DEX) of assets & tokenized goods — **Spectrum**
- Improvements of the Lightning Network formal standards
- Storage & messaging with economic guarantees — **Storm**
- Computing, including high-load computing (e.g. machine learning tasks) — **Prometheus**

LNP/BP Association participants:

Technical

- BHB Network & Blockchain Labs
- Blockstream (individual contributors)
- Pandora Core
- inbitcoin
- Chainside
- Hyperdivision
- Wasabi (prospective)
- Datagnition

Financial

- Bitfinex & Tether
- Poseidon Group
- Fulgur Ventures
- Vaultoro (prospective)
- Six (prospective)

Thanks to:

Olga Ukolova

— the organiser of the event

Financial support from Bitfinex and
other sponsors

Those who helped with doing the event:

— Sabina Sachtachtinskagia

— BEN Italy

— Federico Tenga