Interledger.rs  

A blazing fast implementation of Interledger Protocol

In Rust ❤️

@gakonst / gakonst.com/interledger.pdf

before we start

Setup Rust

1. curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
2. rustup component add rls rust-analysis rust-src rust-docs
3. Cargo install clippy rustfmt racer
4. VScode plugin
5. https://hoverbear.org/blog/setting-up-a-rust-devenv/

Docker setup for testnet connections

- Get the image for the Node & settlement engines

  docker pull interledgerrs/testnet-bundle
- Get the image for the CLI

  docker pull interledgerrs/ilp-cli

Workshop goals

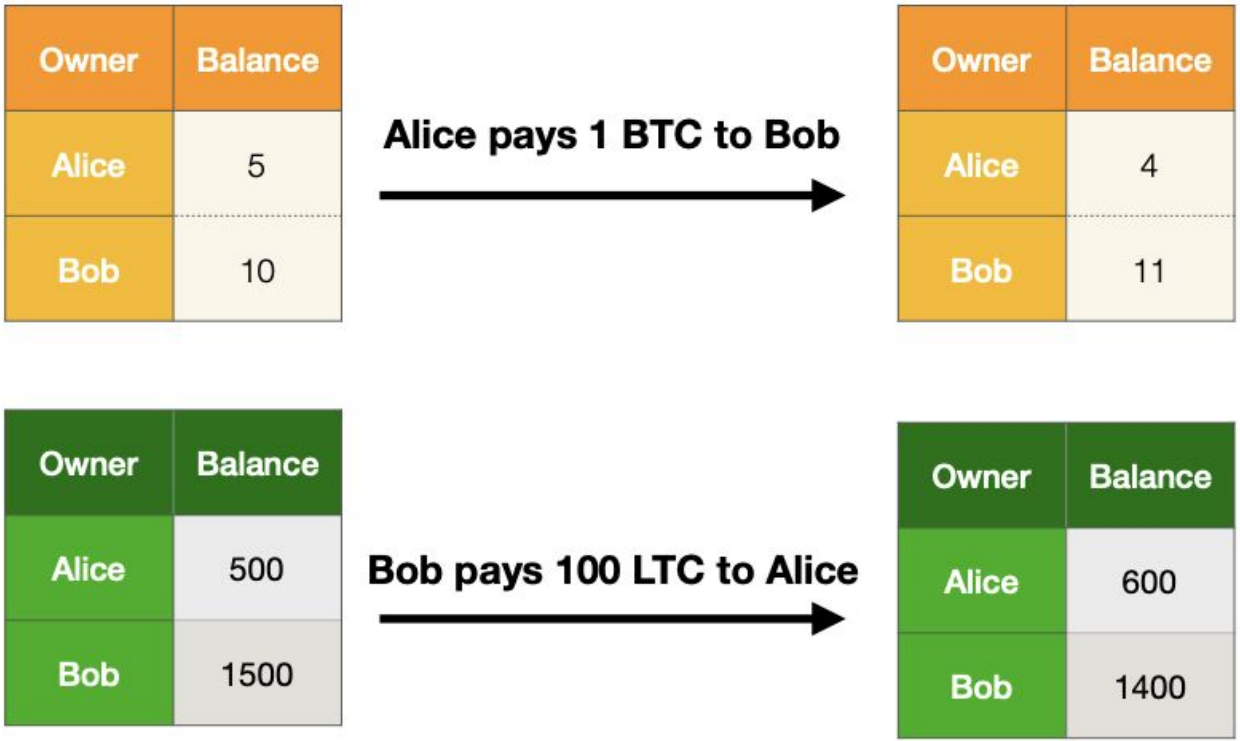
1. Interledger general overview
2. Understanding the settlement architecture
3. Under the hood of the Interledger Settlement, see how easy it is to integrate
4. Testnet is up and running, maybe you can make some test payments:)

Motivation:
Cross ledger (atomic) transactions

Atomic transactions on shared ledger

https://www.youtube.com/watch?v=qUAyW4pdooA

Cross Chain swap. Enforce atomicity?



HTLCs

```
OP_IF

    OP_SHA256 YOUR_HASH OP_EQUALVERIFY OP_DUP OP_HASH160
their_pubkey

OP_ELSE

    timeout OP_CSV OP_DROP OP_DUP OP_HASH160 your_pubkey

OP_ENDIF

OP_EQUALVERIFY OP_CHECKSIG
```

HTLCs

```
OP_IF

    OP_SHA256 YOUR_HASH OP_EQUALVERIFY OP_DUP OP_HASH160
their_pubkey

OP_ELSE

    timeout OP_CSV OP_DROP OP_DUP OP_HASH160 your_pubkey

OP_ENDIF

OP_EQUALVERIFY OP_CHECKSIG
```

Receiver claims by revealing preimage to hash

Sender reclaims after a timeout

Cross Chain Atomic Swaps (happy case)

Owner	Balance
Alice	5
Bob	10

Owner	Balance
Alice	4
Bob	10
HTLC	1

1. Alice locks 1 BTC into 48-hour HTLC, using hash of Alice's secret

Cross Chain Atomic Swaps (happy case)

Owner	Balance
Alice	5
Bob	10

→

Owner	Balance
Alice	4
Bob	10
HTLC	1

1. Alice locks 1 BTC into 48-hour HTLC, using hash of Alice's secret

2. Bob locks 100 LTC into 24-hour HTLC with same hash

Owner	Balance
Alice	500
Bob	1500

→

Owner	Balance
Alice	500
Bob	1400
HTLC	100

Cross Chain Atomic Swaps (happy case)

Owner	Balance
Alice	5
Bob	10

→

Owner	Balance
Alice	4
Bob	10
HTLC	1

1. Alice locks 1 BTC into 48-hour HTLC, using hash of Alice's secret

2. Bob locks 100 LTC into 24-hour HTLC with same hash

Owner	Balance
Alice	500
Bob	1500

→

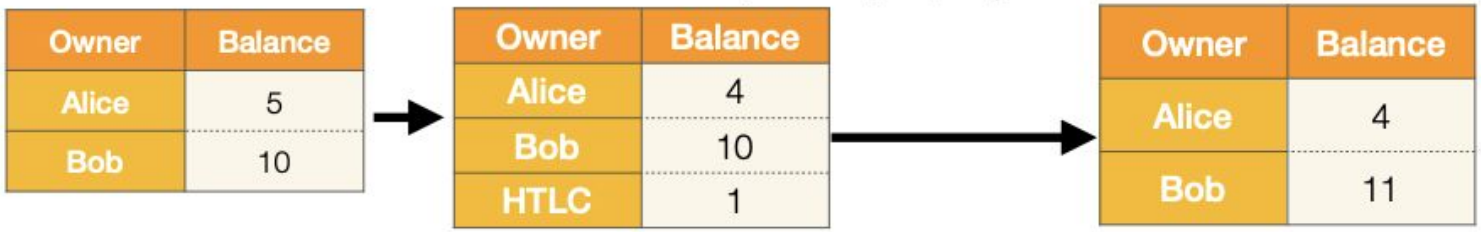
Owner	Balance
Alice	500
Bob	1400
HTLC	100

3. Alice reveals her secret to complete Litecoin HTLC

→

Owner	Balance
Alice	600
Bob	1400

Cross Chain Atomic Swaps (happy case)

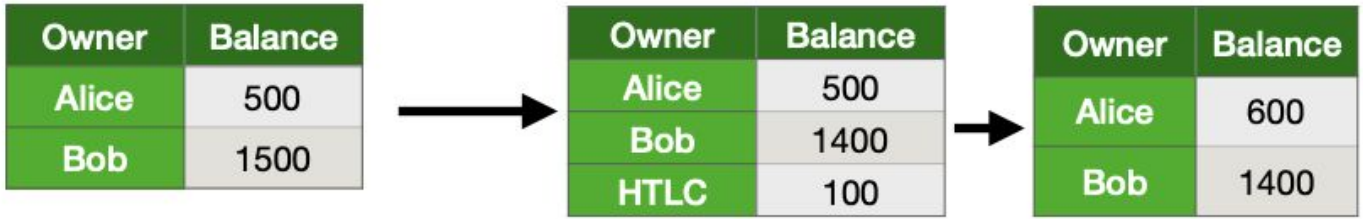


1. Alice locks 1 BTC into 48-hour HTLC, using hash of Alice's secret

4. Bob uses that secret to complete Bitcoin HTLC

2. Bob locks 100 LTC into 24-hour HTLC with same hash

3. Alice reveals her secret to complete Litecoin HTLC



HTLCs Considered Harmful

1. Free option problem: Alice can choose to not complete the trade if the exchange rate changes over time
2. Griefing attack: N * HTLCs chained (Lightning Network), means $N * x gets locked

Interledger: Packetized Payments

- Split your big atomic trade into small, economically insignificant trades
- Take turns executing tiny pieces of it, in sequence
- If your counterparty cheats you at any point, close the connection
    - Uses short-lived HTLCs for repudiation
- Works for multi-hop payments as well
- Does not work for non-fungible assets

How do you settle?

1. Agnostic! Any ledger
2. Isn’t that expensive for blockchains?
    a. Payment channels!
    b. “Cheap” L1s

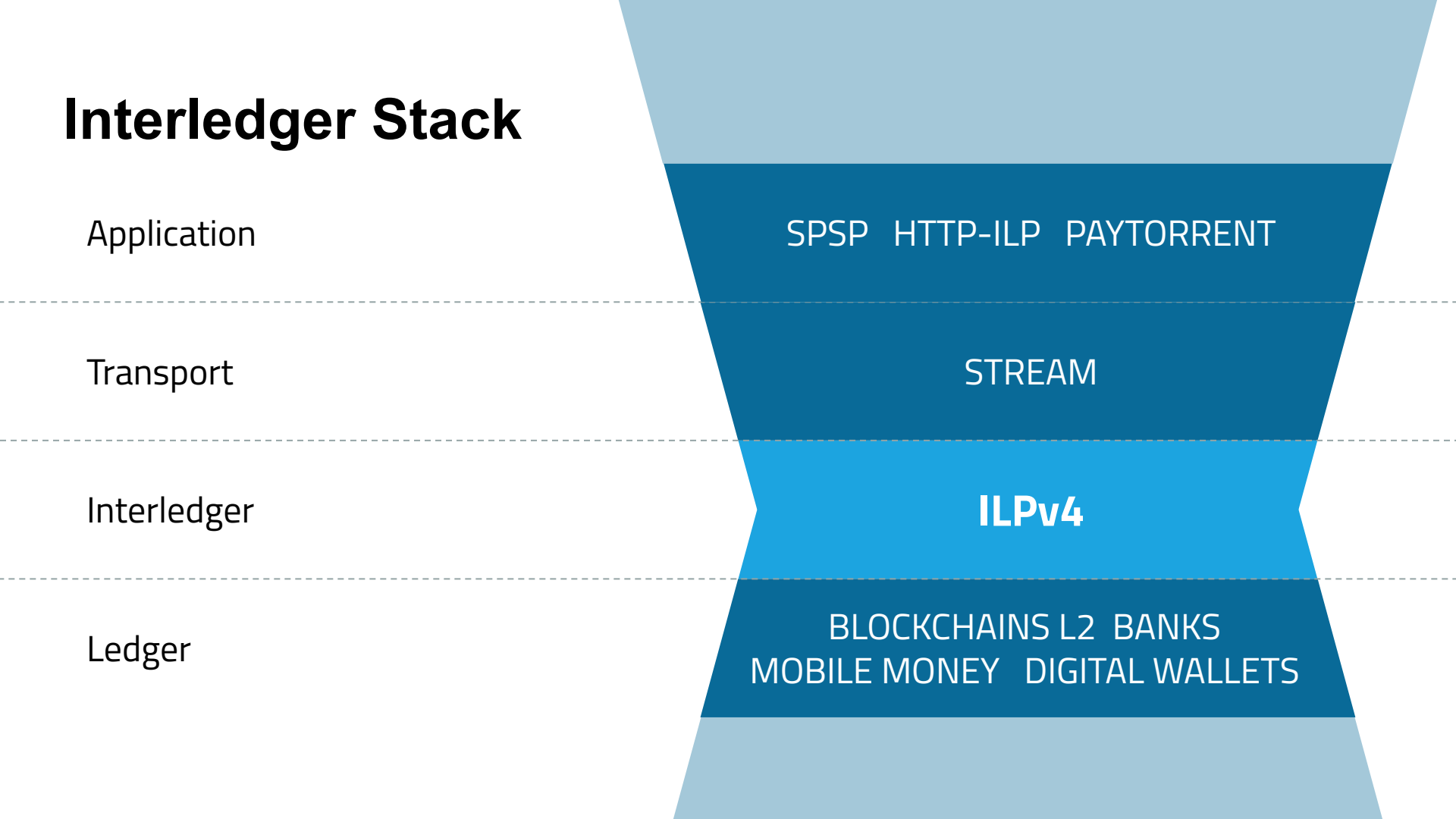
Trust?

1. Set trust limit to small amount
2. Limited to immediate counterparty (griefing in LN requires trusting all hops)

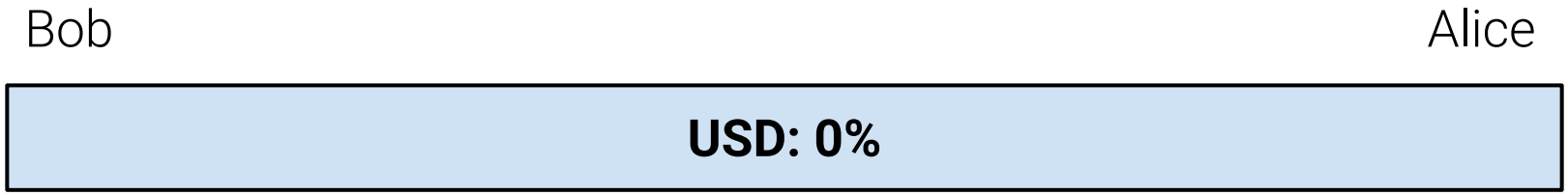
Design Goals

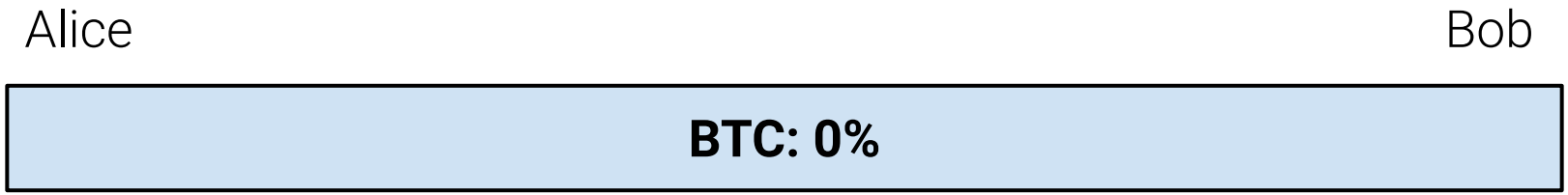
- **Neutrality**: no company, (crypto)currency, network
- **Interoperability**: no assumptions about the ledger
- **Security**:
  - Connectors must not be able to steal from senders
  - Senders should not be able to tie up the connector’s funds
  - If there is a security violation, restrict to immediate counterparty
- **Simplicity**: push complexity up the “interledger stack

Interledger Stack

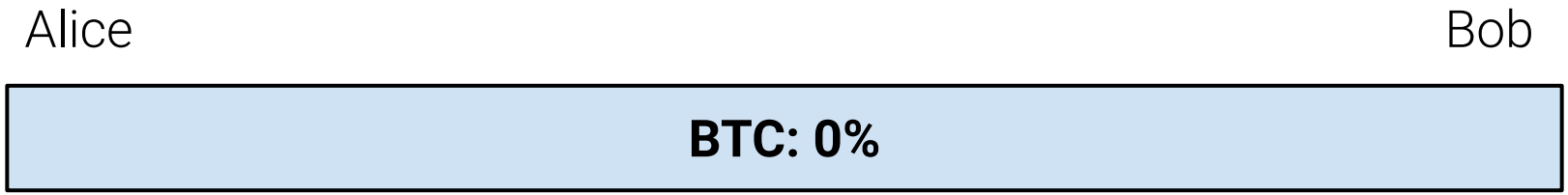


Streaming payments





Streaming payments



Streaming payments

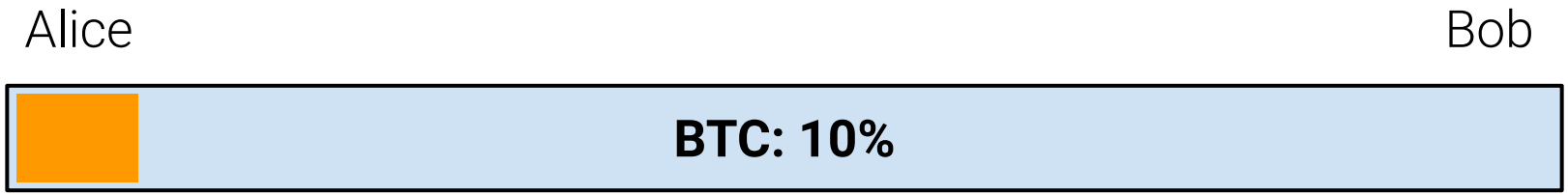
Bob

Alice

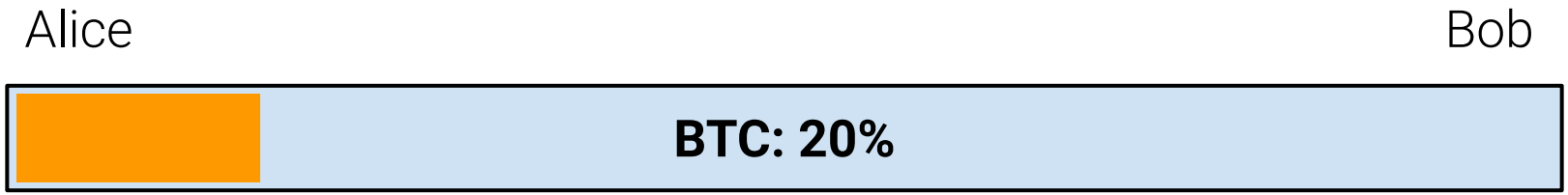
Alice

Bob

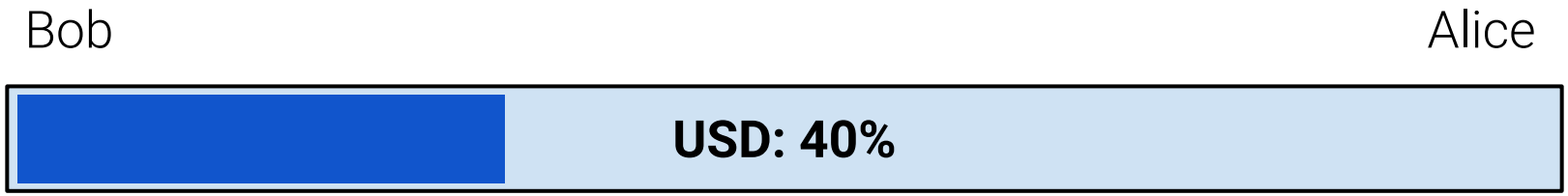
Streaming payments

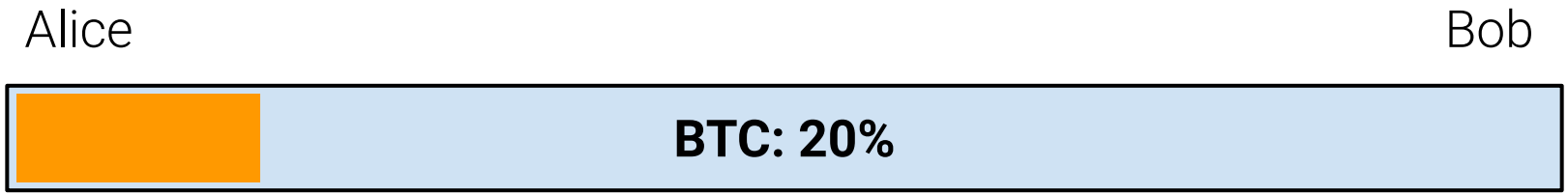


Streaming payments

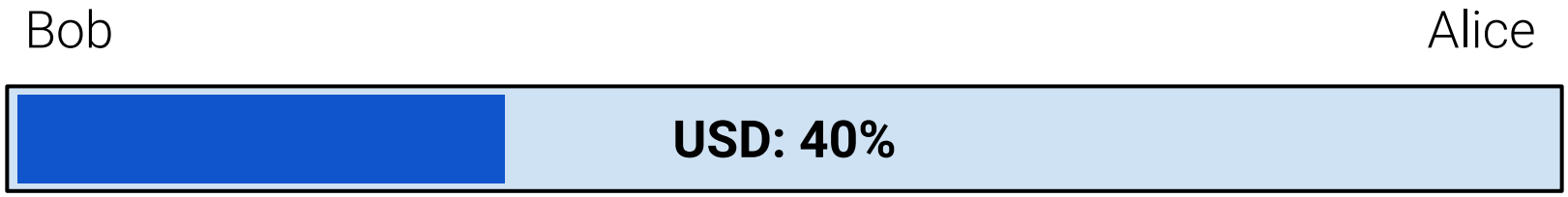


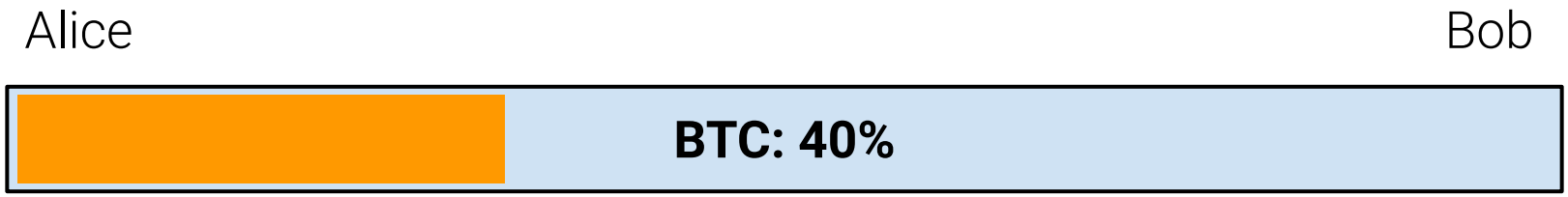
Streaming payments



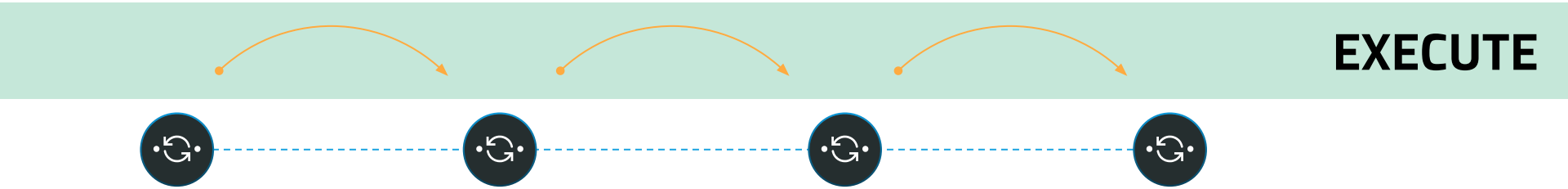


Streaming payments

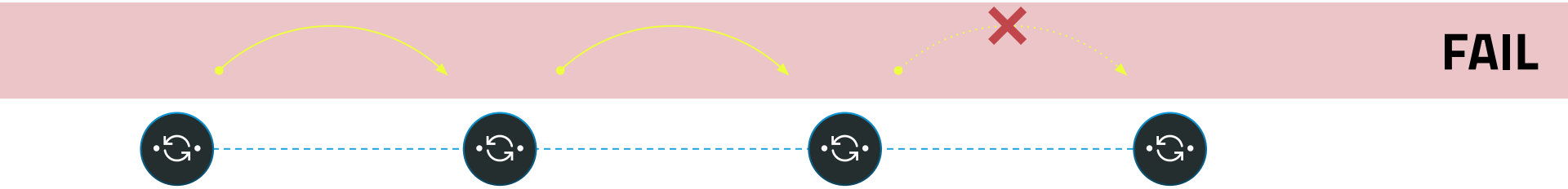




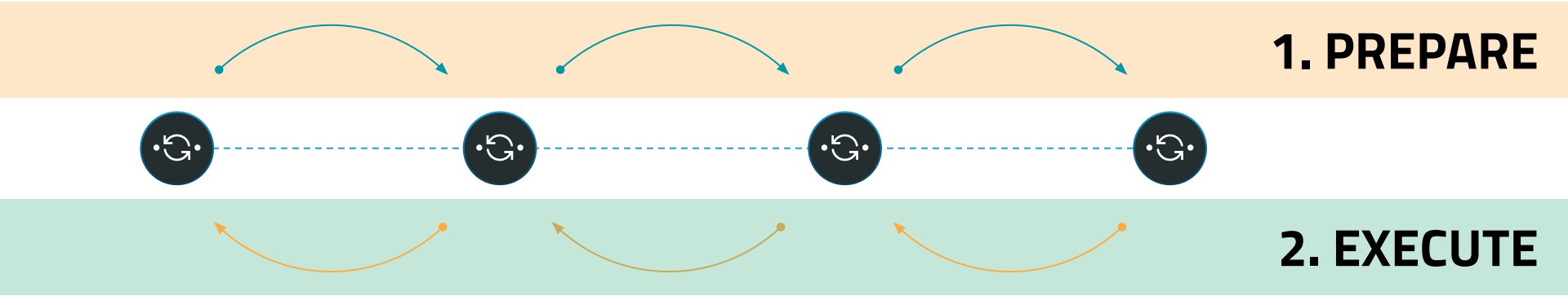
Optimistic Execution



Optimistic Execution (Correspondent Banking Today)



Two-Phase Execution Secures Multi-Hop Transfers



REFERENCES

J. Poon and T. Drya, *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*, 2015

S. Thomas and E. Schwartz, *A Protocol for Interledger Payments*, 2015

As the prepare propagates, funds get allocated

	Balance
a.alice	5
a	0
a.b	0
a.b.c	0
a.b.c.bob	0

“alice” has 5

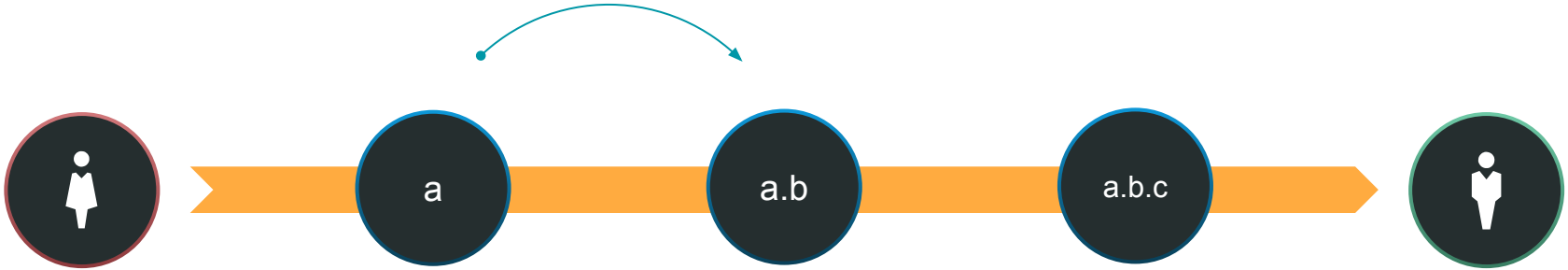
As the prepare propagates, funds get allocated

Wants to pay
a.b.c.bob

	Balance
a.alice	0
a	0
a.b	0
a.b.c	0
a.b.c.bob	0

“alice” has 0 (prepays)

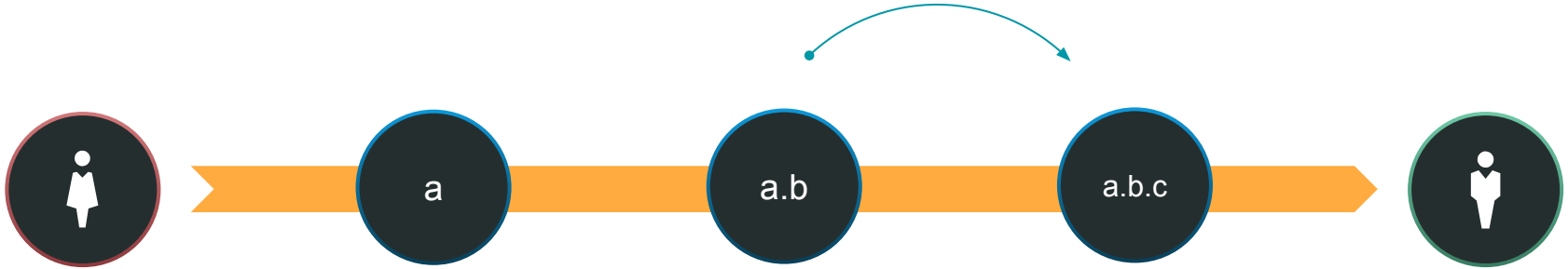
As the prepare propagates, funds get allocated



	Balance
a.alice	0
a	-5
a.b	0
a.b.c	0
a.b.c.bob	0

“a” owes “a.b” 5

As the prepare propagates, funds get allocated



	Balance
a.alice	0
a	-5
a.b	-5
a.b.c	0
a.b.c.bob	0

“a” owes “a.b” 5

“a.b” owes “a.b.c” 5

Packet routed: Increase receiver's balance

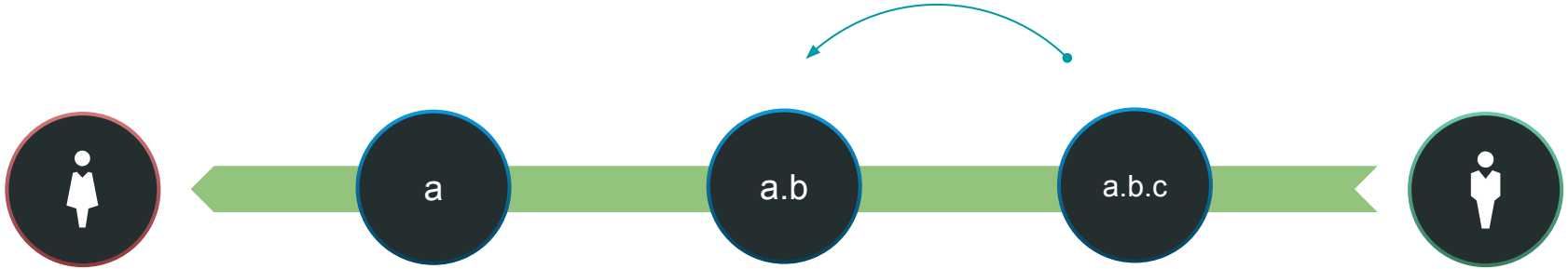
	Balance
a.alice	0
a	-5
a.b	-5
a.b.c	0
a.b.c.bob	5

"a" owes "a.b" 5

"a.b" owes "a.b.c" 5

"bob" receives 5

As the fulfill gets sent back, reset balances



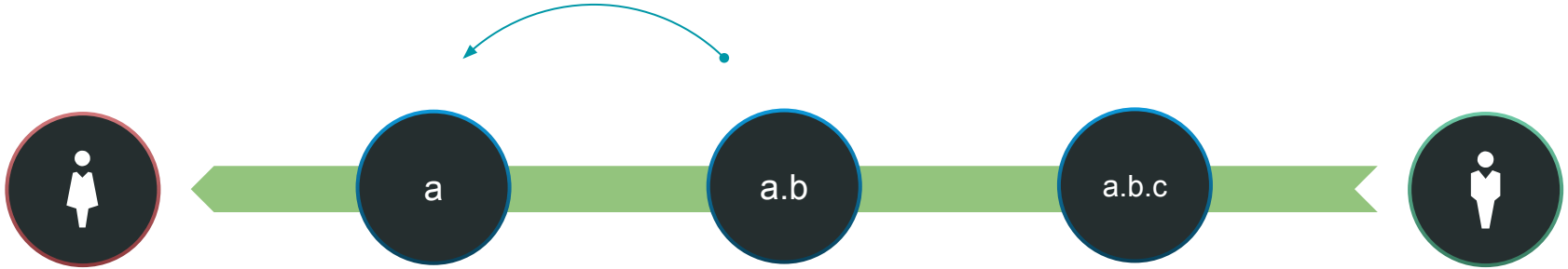
	Balance
a.alice	0
a	-5
a.b	0
a.b.c	0
a.b.c.bob	5

“a” owes “a.b” 5

“a.b” owes “a.b.c” 0

“bob” owns 5

As the fulfill gets sent back, reset balances



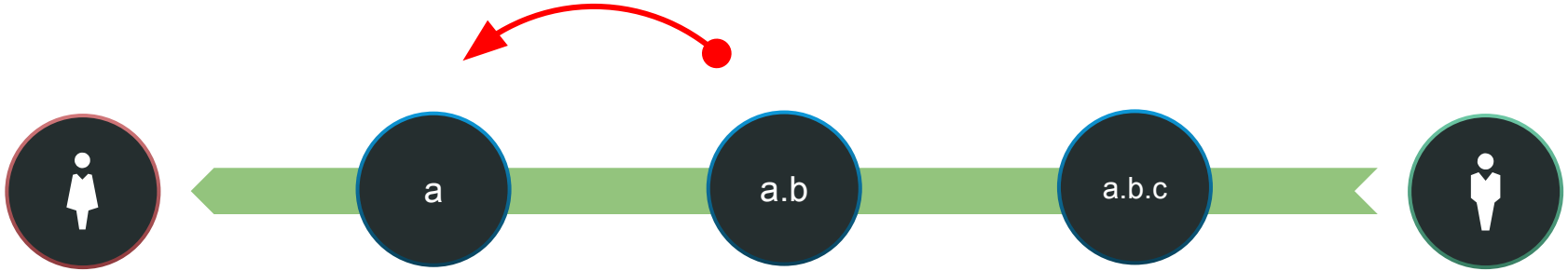
	Balance
a.alice	0
a	0
a.b	0
a.b.c	0
a.b.c.bob	5

“a” owes “a.b” 0

“a.b” owes “a.b.c” 0

“bob” owns 5

If any node rejects we credit them back



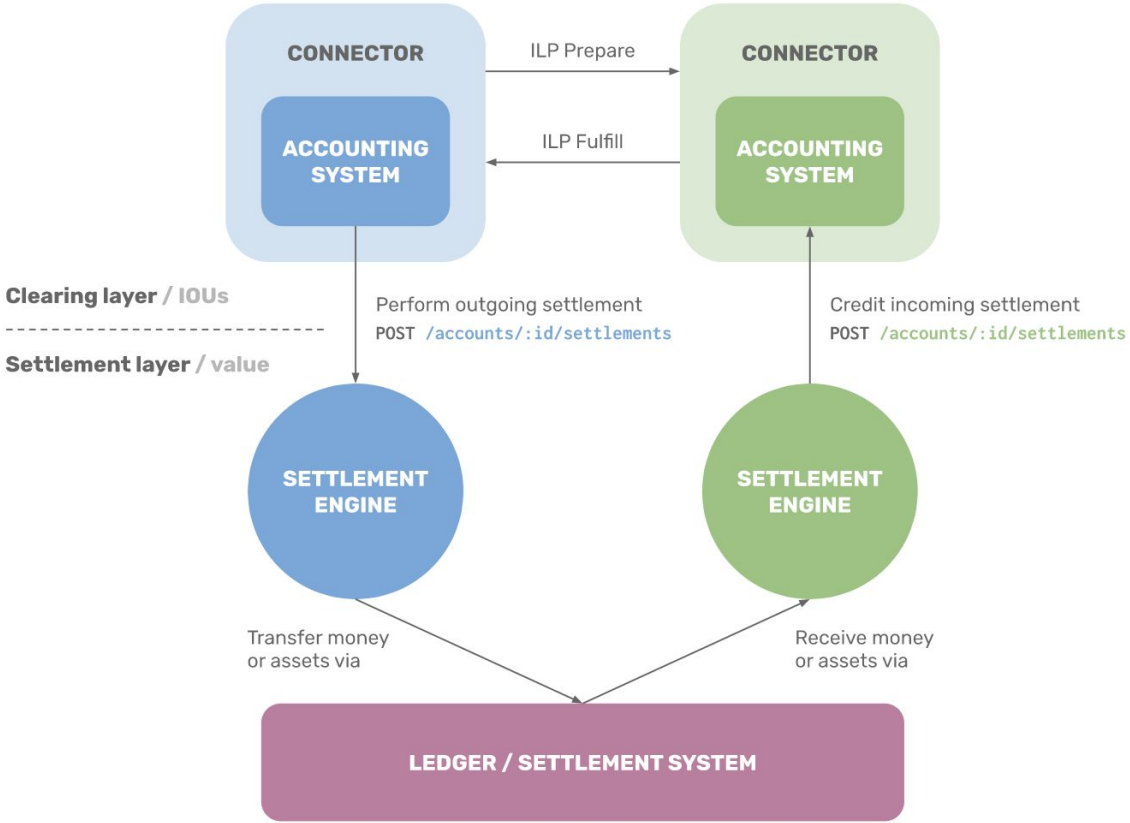
	Balance
a.alice	5
a	0
a.b	0
a.b.c	0
a.b.c.bob	0

“a.b” rejected, we should refund alice and assume that the payment failed

Existing implementations / community

1. Interledger-rs (Rust) - https://github.com/interledger-rs/interledger-rs
2. Rafiki (Typescript) - https://github.com/interledgerjs/rafiki•
3. Quilt (Java) - https://github.com/hyperledger/quilt
4. ILP-Connector (Typescript) - https://github.com/interledgerjs/ilp-connector
5. Orcus (Golang) - https://github.com/uroshercog/orcus

The settlement architecture



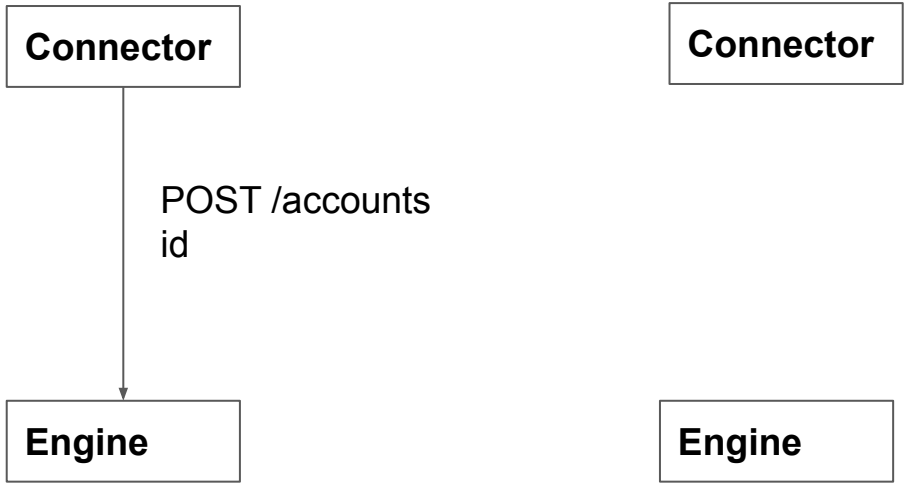
Available Settlement Engines

1. Ethereum (L1): https://github.com/interledger-rs/settlement-engines
2. XRP: https://github.com/interledgerjs/settlement-xrp
3. Paypal: https://github.com/interledgerjs/settlement-paypal
4. Lightning: https://github.com/interledgerjs/settlement-lightning/pull/39
5. …? Build your own!

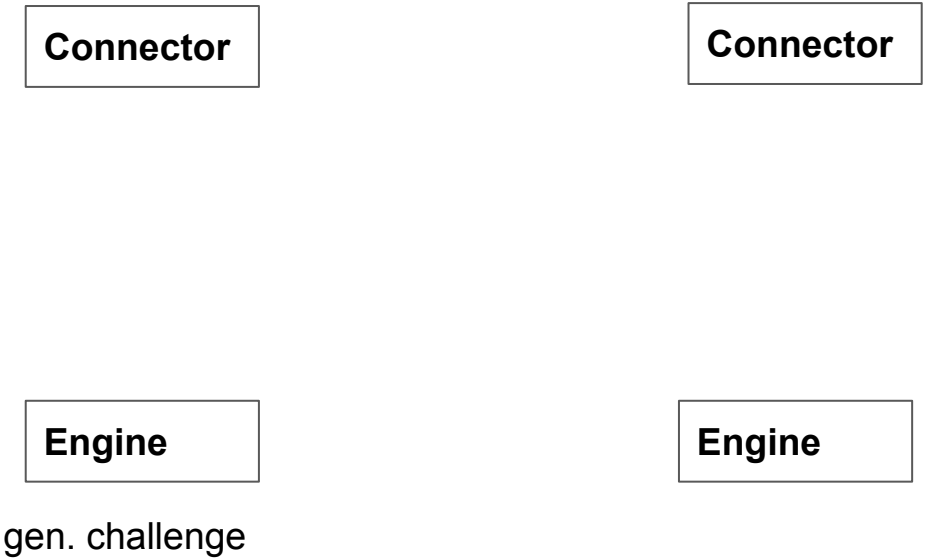
Account Creation example (like a handshake)

(ethereum engine implementation)

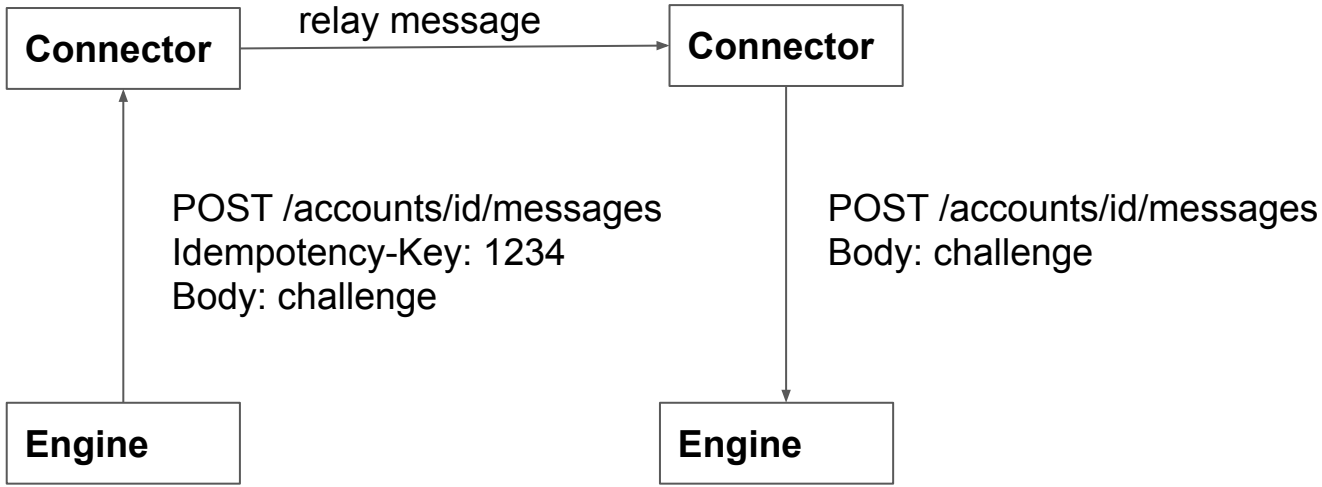
Connector asks engine to create an account



Engine generates a challenge for auth

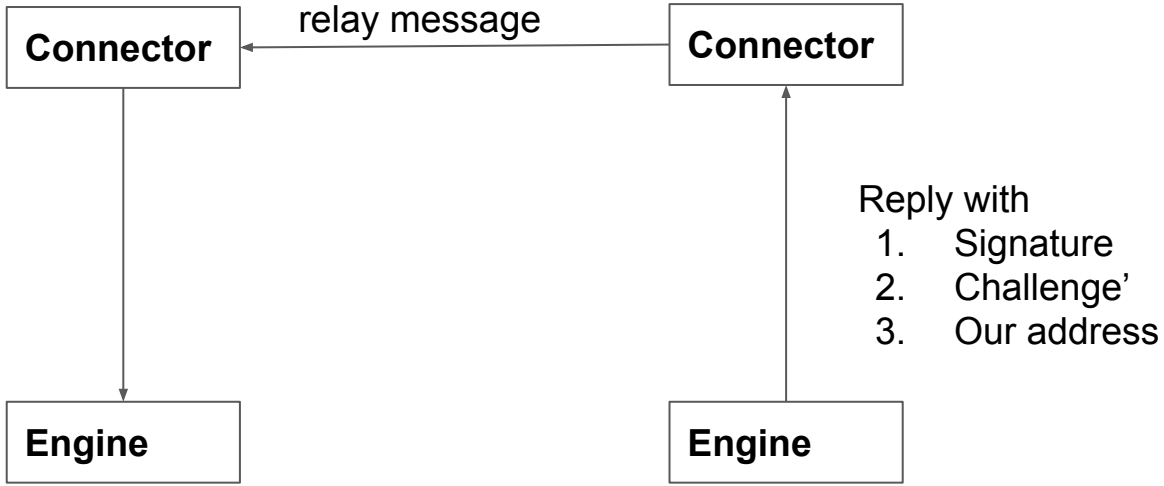


Challenge gets relayed to peer’s engine



Peer signs the challenge and generates c’

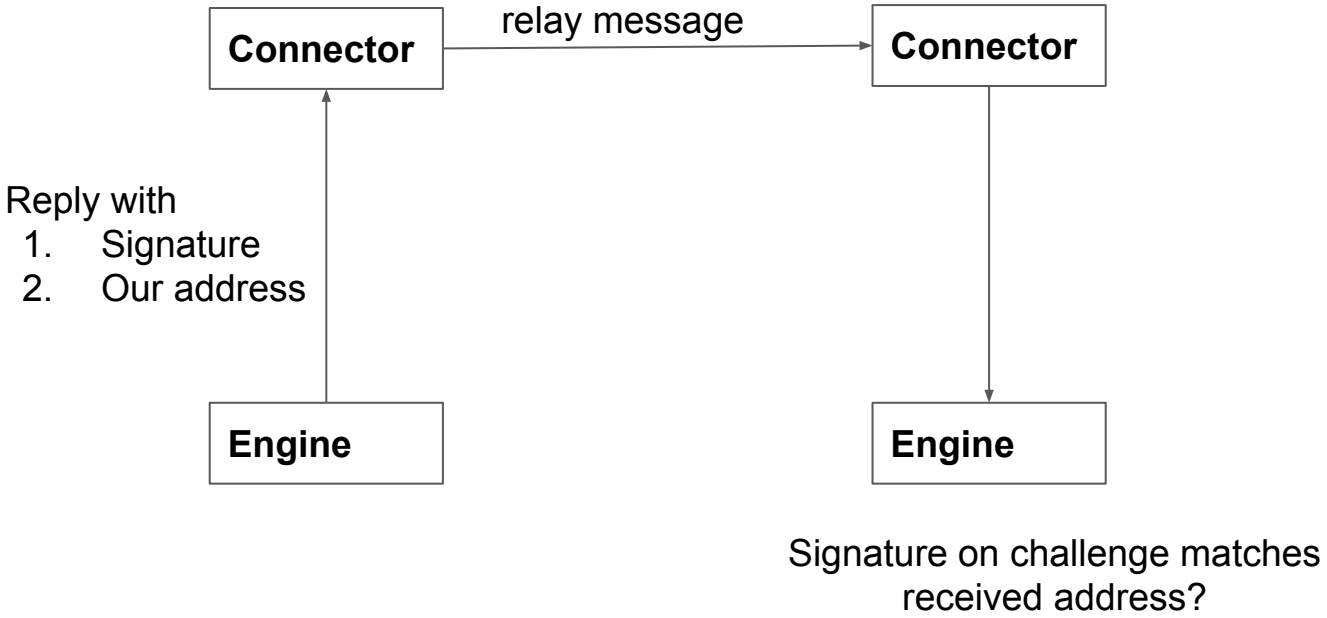
Verify signature on challenge



Signature on challenge matches received address?

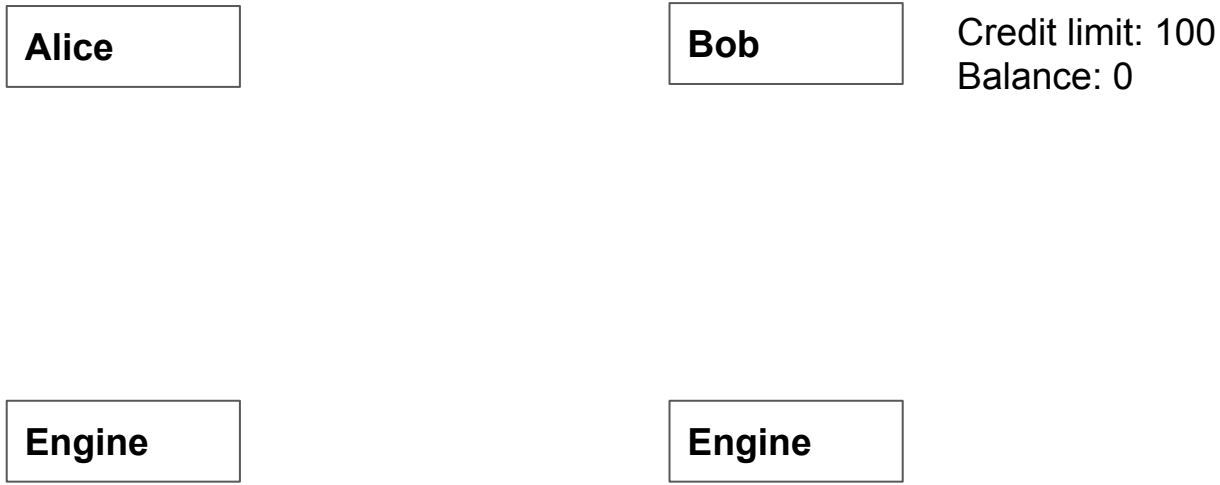
If OK, save account and reply with our own sig

One more roundtrip

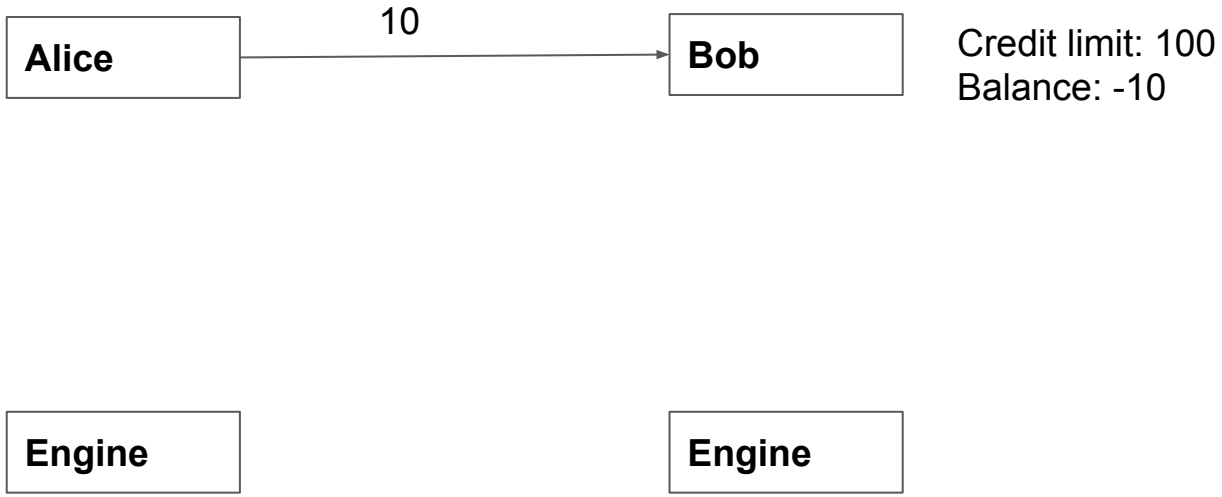


Peering complete

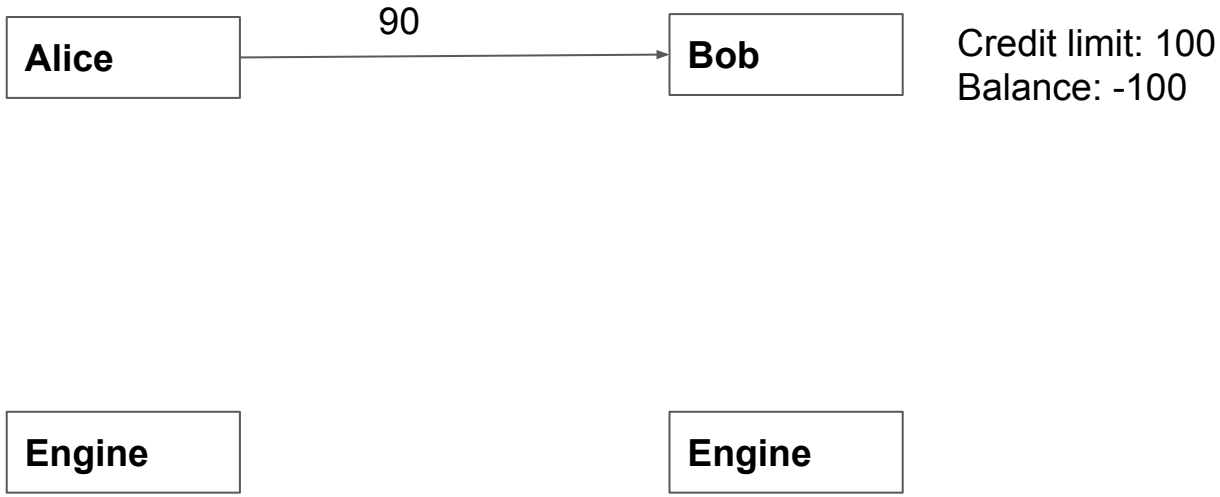
Triggering Settlement



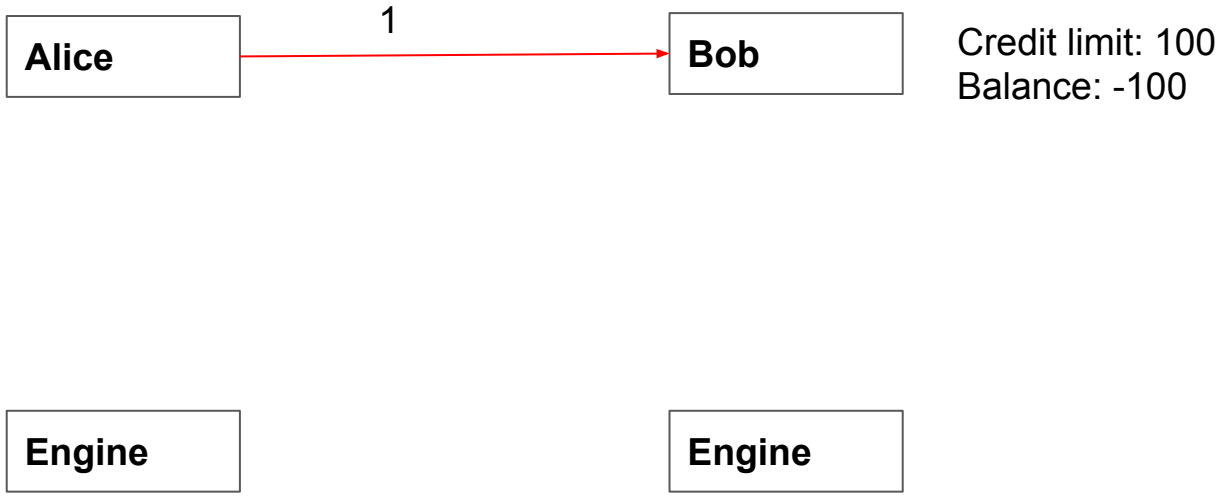
Triggering Settlement



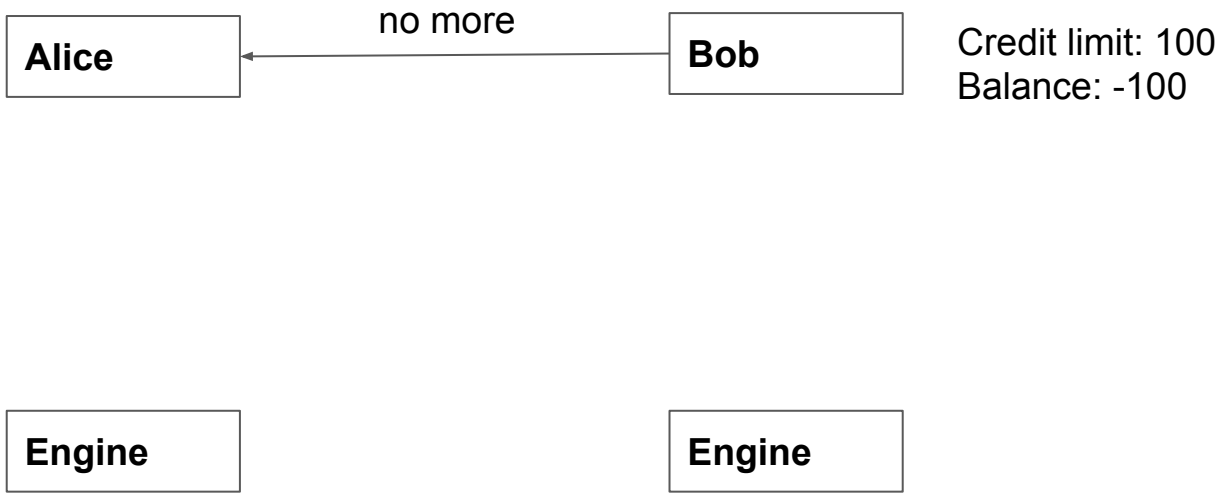
Triggering Settlement



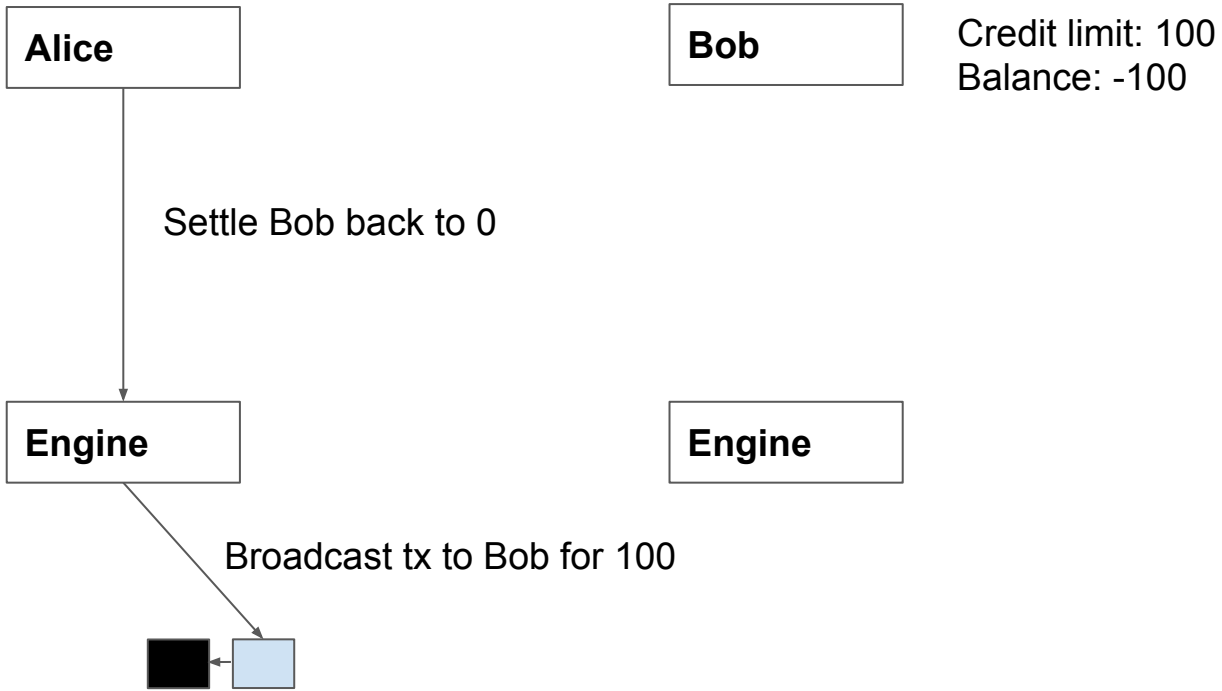
Triggering Settlement



Triggering Settlement



Notify engine to send an ETH transaction to Bob



Bob’s engine is waiting for enough confs

Alice

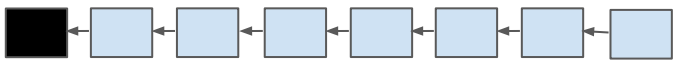
Bob

Credit limit: 100
Balance: -100

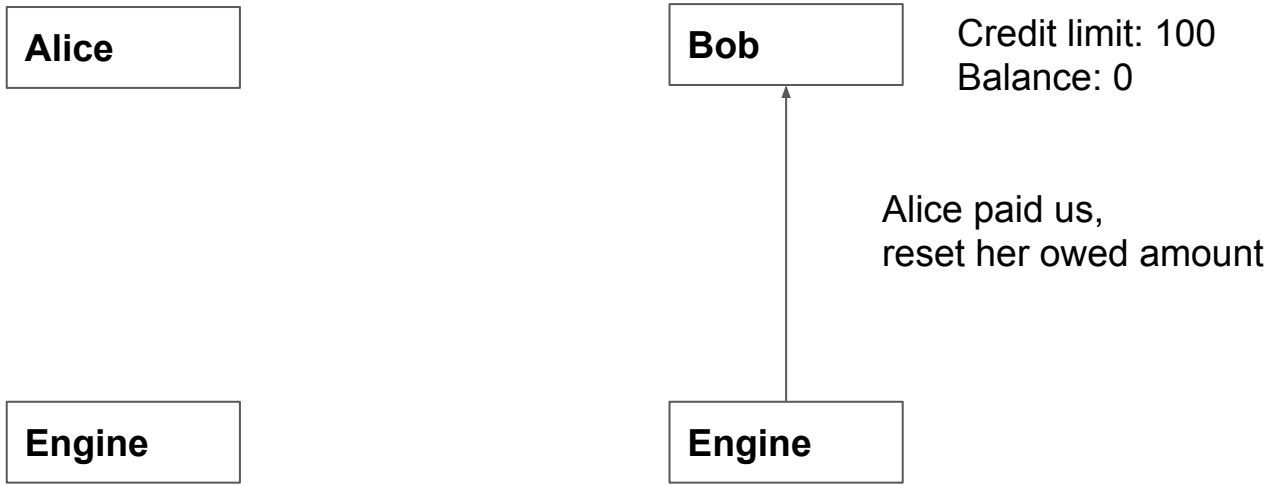
Engine

Engine

Watch incoming payments



Connector is notified, Alice can resume streaming



Standardized Settlement API (Connector)

Idempotency pattern by Stripe): `Header: Idempotency-Key`:
same input & key → same output, else 409 CONFLICT

Endpoints:

- `/accounts/:id/settlements`: Receive incoming settlement from engine

Body: amount & asset scale
e.g. {amount: 1, scale: 9} (same as {amount: 1e9, scale: 18})

- `/accounts/:id/messages`: Send message to peer's engine

Body: arbitrary data

https://interledger.org/rfcs/0038-settlement-engines/

Standardized Settlement API (Engine)

Endpoints:

- `/accounts`: Create account on engine
- `/accounts/:id/settlements`: Execute settlements received from connector

Body: amount & asset scale
e.g. {amount: 1, scale: 9} (same as {amount: 1e9, scale: 18})

- `/accounts/:id/messages`: Receive messages from peer's engine

Body: arbitrary data

https://interledger.org/rfcs/0038-settlement-engines/

demo

Demo on ETH / XRP Testnets

Connect to Rinkeby ETH: https://faucet.rinkeby.io/

```
docker run -it -e NAME=gakonst -e
ETH_SECRET_KEY=758B08B9DA8A68F12F3214D69DBE09B705FAC9DD0E01C9AD2391368C809C7
FB6 -e CURRENCY=ETH interledgerrs/testnet-bundle
```

Connect to XRP testnet:

```
docker run -it -e NAME=gakonst -e CURRENCY=XRP interledgerrs/testnet-bundle
```

https://github.com/interledger-rs/interledger-rs/blob/master/docs/testnet.md

< intermission >

Interledger.rs Architecture

- Every ILP packet is processed by a chain of stateless Services
- All state is kept in an underlying database or Store
- All details related to an account or peer are bundled in an Account object, which is loaded from the Store and passed through the Services
- Nothing is instantiated for each packet or for each account; services that behave differently depending on account-specific details or configuration use methods on the Account object to get those details and behave accordingly
- Multiple identical nodes / connectors can be run and pointed at the same underlying database to horizontally scale a deployment for increased throughput

https://github.com/interledger-rs/interledger-rs/tree/master/docs

Implementing a Settlement Engine

```rust
/// Trait consumed by the Settlement Engine HTTP API. Every settlement engine
/// MUST implement this trait, so th
pub trait SettlementEngine {
    fn send_money(
        &self,
        account_id: String,
        money: Quantity,
    ) -> Box<dyn Future<Item = ApiResponse, Error = ApiError> + Send>;

    fn receive_message(
        &self,
        account_id: String,
        message: Vec<u8>,
    ) -> Box<dyn Future<Item = ApiResponse, Error = ApiError> + Send>;

    fn create_account(
        &self,
        account_id: String,
    ) -> Box<dyn Future<Item = ApiResponse, Error = ApiError> + Send>;
}
```

```rust
#[derive(Debug, Clone, Serialize, Deserialize, Eq, PartialEq)]
pub struct Quantity {
    pub amount: String,
    pub scale: u8,
}
```

< dive into the code >

https://github.com/interledger-rs/interledger-rs/
https://github.com/interledger-rs/settlement-engines

Things you can do

1. Play with the testnet
2. Check the Open Issues!
3. Integrate with a Ledger of your choice
    a. Does not have to be in Rust! Check the TS engine for XRP for inspiration
4. Take the Lightning Network engine for a ride
5. Read the RFCs to get a deeper understanding of the protocol
6. Join the interledger slack discussions
7. Join the Interledger community call every 2 Wednesdays
8. Join the interledger discourse!