Developing Blockchain Software

David Schwartz, Chief Cryptographer

CPPCON: September 22, 2016

About Me

David Schwartz

Chief Cryptographer at Ripple

One of the original architects of the Ripple Consensus Ledger

Known as JoelKatz in many online communities ripple

Global Leader in Distributed Financial Technology

135

Team members

⅔ engineering talent

Our Experience

Financial Services
- J.P. Morgan
- Citi
- Merrill Lynch
- BlackRock
- Visa
- Fiserv
- Paypal
- Prosper

Technology
- Google
- Apple
- Yahoo
- Bloomberg
- NASA

Regulation
- Federal Reserve
- SEC
- DTCC
- NSA

San Francisco | NYC | London | Sydney | Luxembourg

Sample of our Customers and Partners

Banking Partners

Santander
CIBC
UniCredit
MIZUHO
SHRB 上海华瑞银行
Standard Chartered
BMO
nab
UBS
Westpac
SCB ไทยพาณิชย์
ATB Financial
Reise Bank
بنك أبوظبي الوطني NBAD
CBW BANK
fidor BANK

Consulting Partnerships

accenture
CGI
Deloitte.

Technology Partnerships

CGI
D+H
EXPERTUS
Volanté
intellect EU

Blockchains

What is a blockchain and what is one good for?

Blockchains record state and history

State is modified by transactions

Everyone eventually agrees on the transactions

Can be used to transfer tokens

What is a blockchain and what is one good for?

Assets are owned by identities

Identities are public keys

Authority is proven by digital signatures

Transactions are signed

Integrity is protected by secure hashes

So it’s just a database?

What is a blockchain and what is one good for?

Double Spending

If Alice has $10, she can send it to Bob

Or she can send it to Charlie

But, if she can do both, we have a problem

Sending to Charlie must stop her from sending to Bob

What is a blockchain and what is one good for?

What's the Problem?

The usual solution is a central authority

Banks, for example

They prevent double spending by reconciling against a ledger

Can also be done with secure hardware

Ultimately, you need a central authority

What is a blockchain and what is one good for?

Before blockchains:

Hashcash: Currency generated by proof of work

B-Money: Trust the servers

Ripple classic: Lots of authorities

Bitcoin

What is a blockchain and what is one good for?

Bitcoin

The first blockchain

Literally a chain of blocks

Each block contains the hash of the previous block

Transactions transfer a native token

What is a blockchain and what is one good for?

UTXO Model

UTXO = Unspent transaction output

Network state is a set of valid UTXOs

Payments gather UTXOs into a pile

Payments create new UTXOs

We assume the network agrees on the set of UTXOs

What is a blockchain and what is one good for?

Bitcoins are currency

Scarce

Fungible

Divisible

Durable

Transferable

What is a blockchain and what is one good for?

Bitcoin Mining

Mining generates bitcoins

Miners are incentivized to lengthen the longest chain

The longest chain “wins”

We have eventual consistency

Double spend problem solved

What is a blockchain and what is one good for?

Bitcoin

Currency plus payment system

Payment system provides ultimate grounding

System regulates introduction of new currency

Supply is ultimately fixed

What is a blockchain and what is one good for?

Bitcoin

Rules are notionally set in stone

They can be changed by social consensus

The past can be rewritten

Mining uses a lot of power to secure transactions

UTXO model

Ripple

A platform for issuing, holding, transferring, and trading arbitrary assets.

Ripple

A platform for issuing, holding, transferring, and trading arbitrary assets.

Some history

Began in 2011

Distributed agreement protocol instead of proof of work

Replace blocks with ledgers

Allow arbitrary assets

Ripple

A platform for issuing, holding, transferring, and trading arbitrary assets.

Ledger

Ledger replaces UTXO

Ledgers form a secure hash chain

Ledger contains all current state information

Transaction sets advance the ledger

Prior ledgers can be forgotten

Ripple

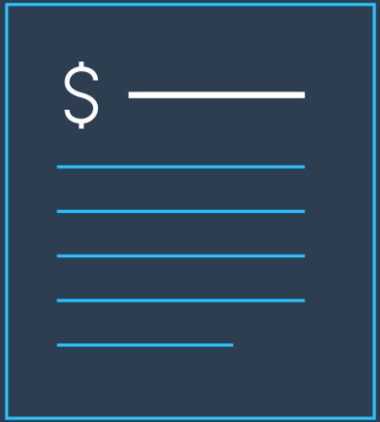
A platform for issuing, holding, transferring, and trading arbitrary assets.

Ledger

Contains transactions

Contains metadata

Supports more complex transactions



Ripple

A platform for issuing, holding, transferring, and trading arbitrary assets.

Consensus

Distributed agreement protocol similar to PBFT

Does not require 100% agreement on the participants

Does require substantial agreement on the participants

Ripple

A platform for issuing, holding, transferring, and trading arbitrary assets.

Key Points of Consensus

Ripple’s method of solving the double spend problem

Validators agree on a group of transactions to be applied in a given ledger

Validators sign each ledger they build

Analogous to a room full of people trying to agree

All honest servers place a high value on agreement, second only to correctness

Consensus

Establishes transaction ordering

Consensus

Establishes transaction ordering

Why is transaction ordering important?

Transaction validity is deterministic

Transaction execution is deterministic

Transactions either conflict or they don’t

If they do, the second one must fail

Consensus

Establishes transaction ordering

What do validators do?

Agree on the last closed ledger

Propose sets of transactions to include in the next ledger

Avalanche to consensus

Apply agreed transactions according to deterministic rules

Publish a signed validation of the new last closed ledger

Consensus

Establishes transaction ordering

Why is consensus robust?

If a transaction has no reason not to be included, all honest validators will vote to include it

If a transaction has some reason not to be included, it is okay if it is not included

Valid transactions that do not get into the consensus set will be voted into the next set by all honest validators

Algorithm is biased to exclude transactions to reduce overlap required

Ripple

A platform for issuing, holding, transferring, and trading arbitrary assets.

Advantages of consensus

No rotating dictators

Choose who to trust

Fast

Past cannot be rewritten

Advantages of ledgers

Reliable agreement on network state

Control over the growth of state

Faster spin up of new nodes

Ripple

A platform for issuing, holding, transferring, and trading arbitrary assets.

Key Features

Open source, ISC license

Public ledger, public transactions, public history

Equal access, peer-to-peer, no central authority

Fast transactions with reliable confirmation

Sophisticated cross-currency and cross-issuer payments

How RCL Works

How RCL Works

Arbitrary assets

Assets are identified by issuer and currency

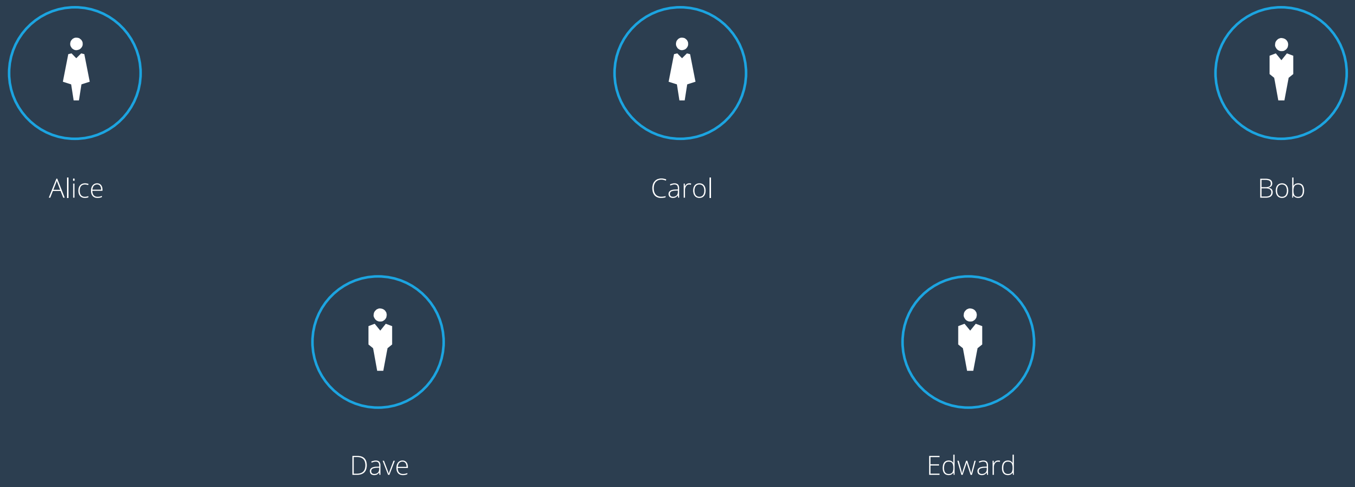
You must choose to hold an asset

Assets have counterparties

Assets can reflect legal obligations

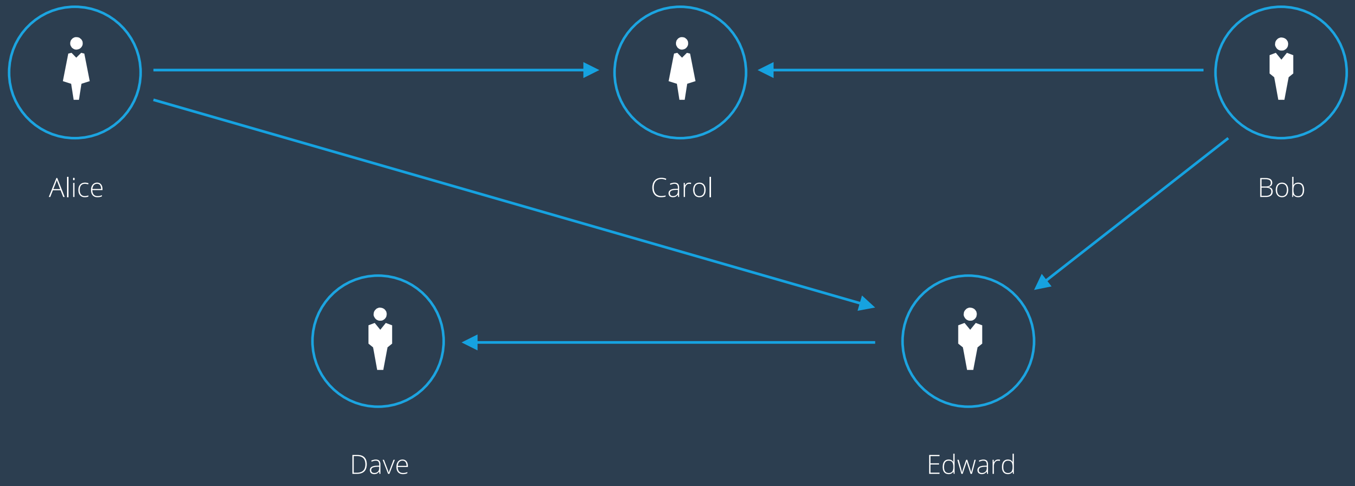
Accounts

Identities in the network



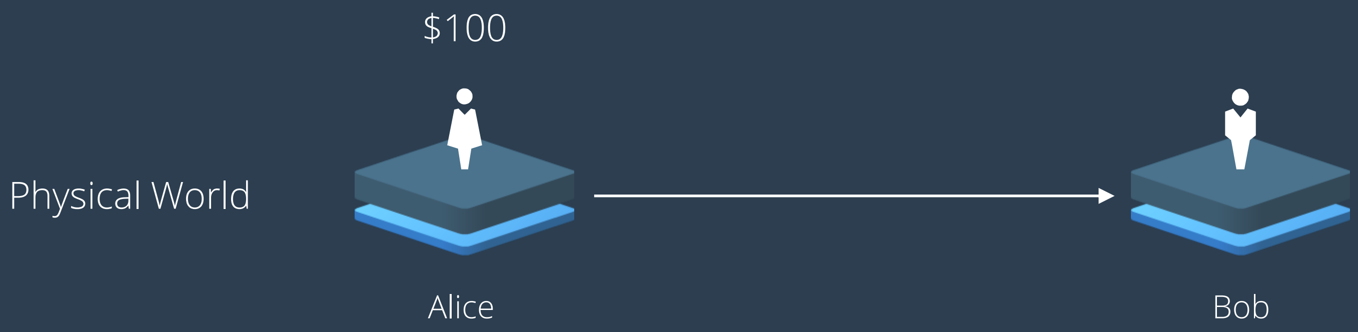
Trust Lines

A directed graph



Balances

Having money



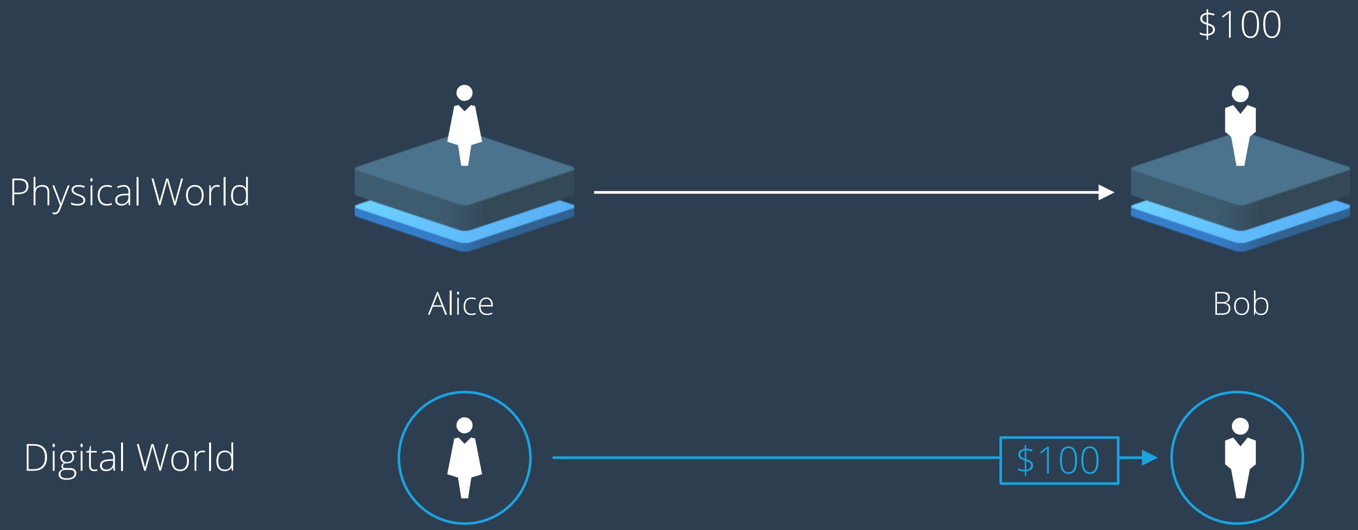
Balances

Having money

Physical World

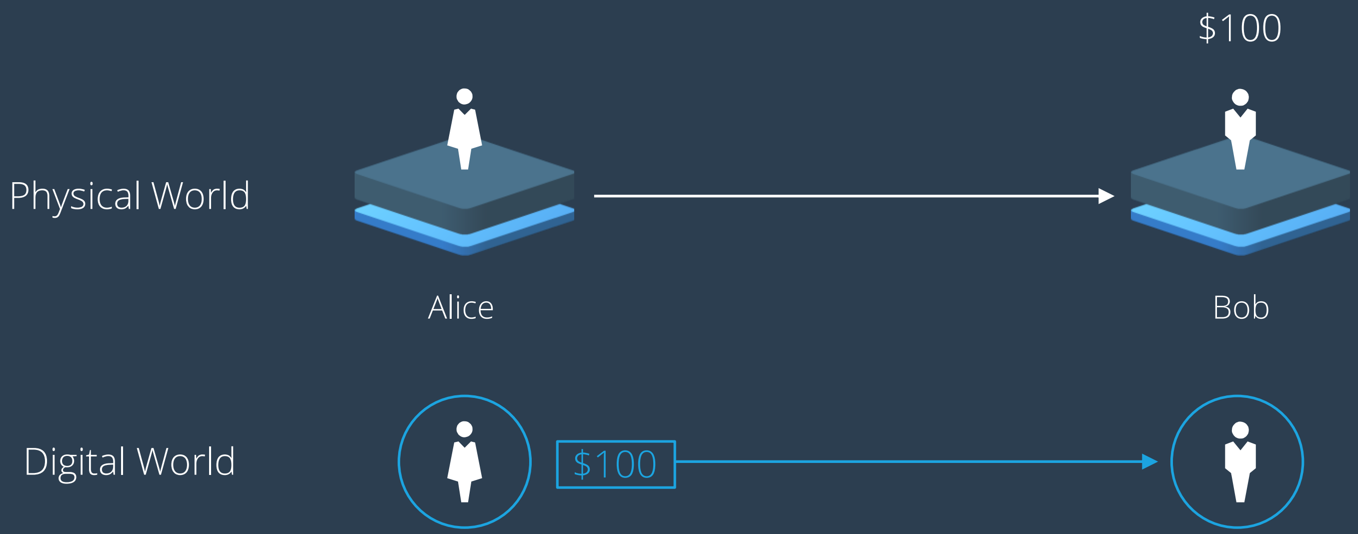
Issuance

Digitizing money



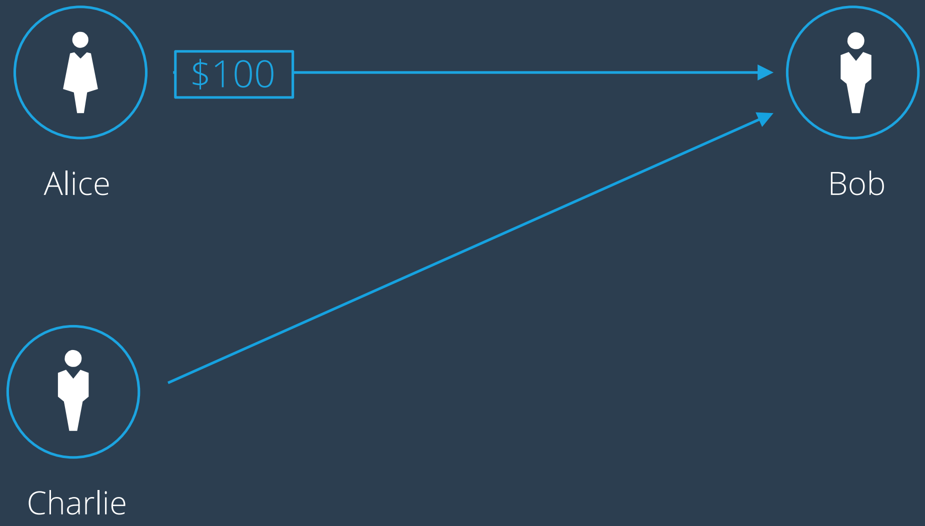
Issuance

Digitizing money



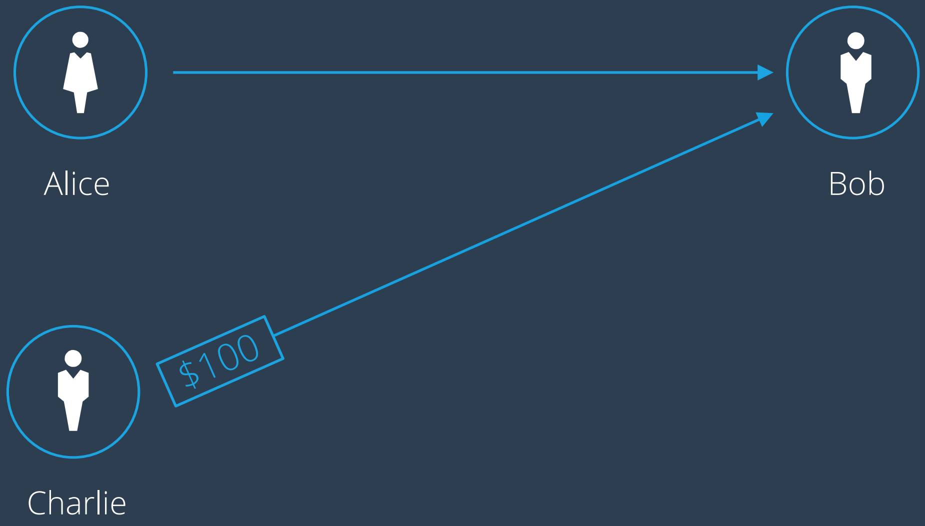
Transfer

Payments work



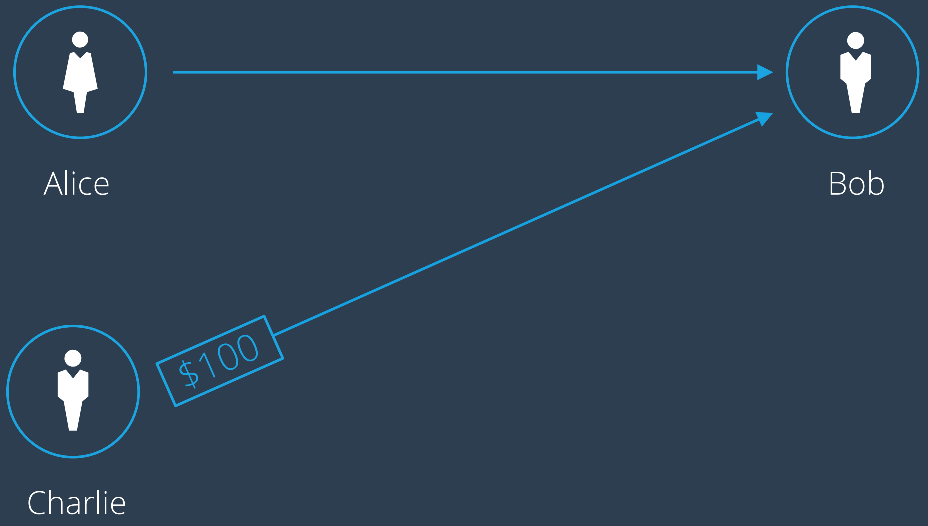
Transfer

Payments work



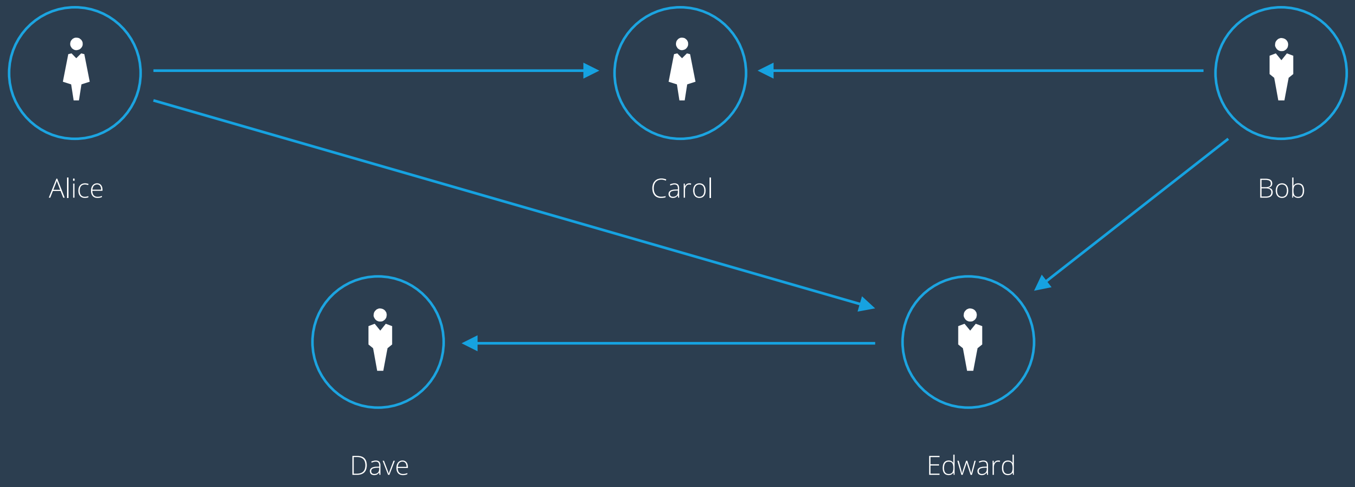
Transfer

Payments work



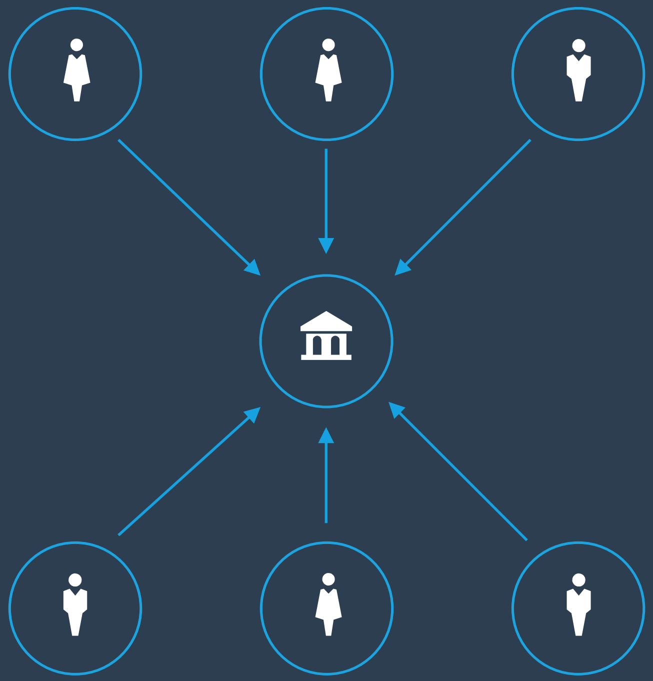
Usability?

Not so much



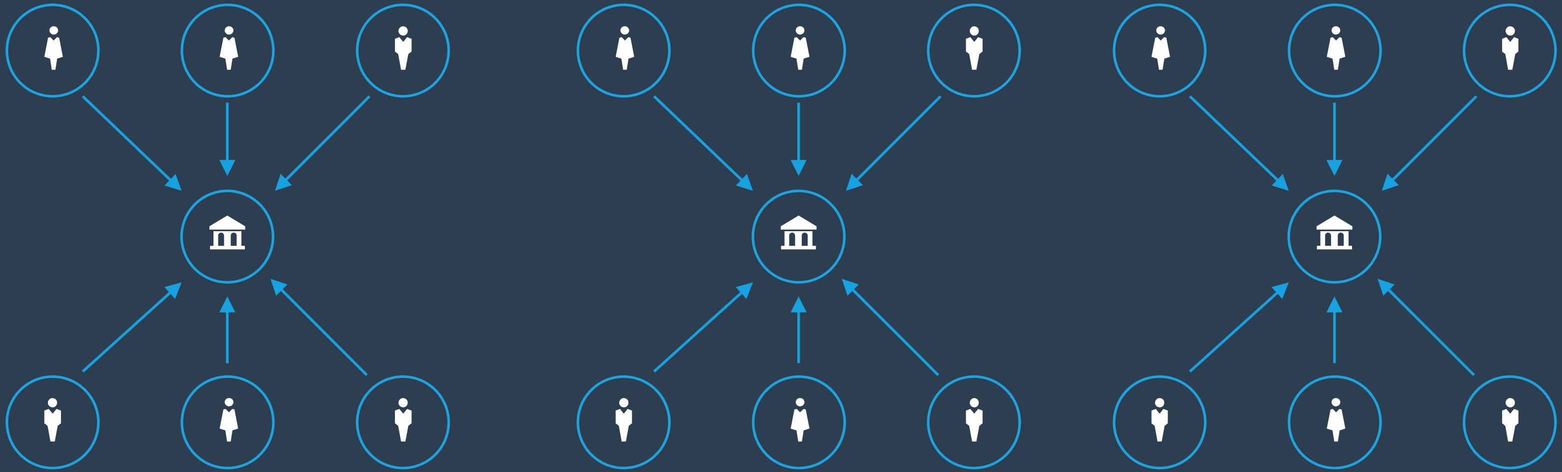
Gateways

Hubs of trust



Gateways

Islands of trust



Gateways

Islands of trust

How RCL Works

Arbitrary assets

Money does not really move

Payments swap ownership of assets

Sender loses custody of the asset they sent

Recipient gains custody of the asset they wanted

Payments “ripple through” intermediaries

How RCL Works

Social credit

Instead of borrowing money, exchange IOUs of equal value

Balances are tracked automatically

Settlement is done as needed

Default requires abandoning the currency, account, or system

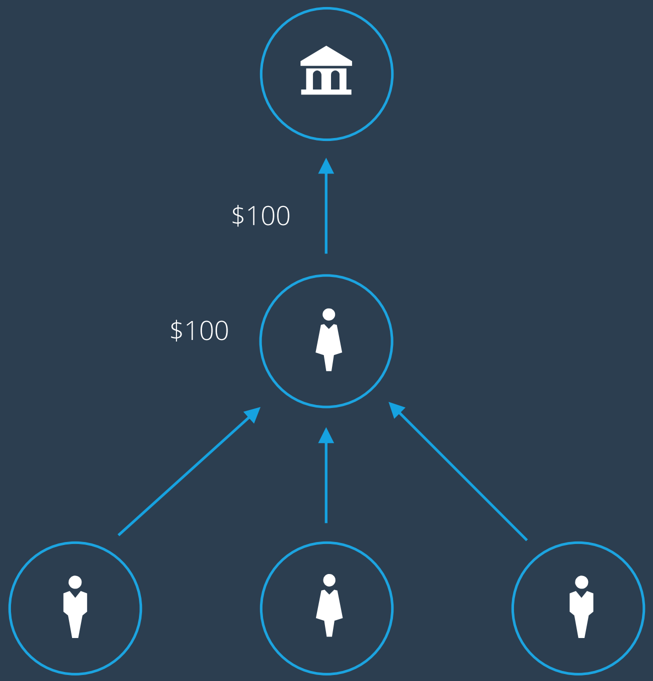
Defaults do not propagate

Allowance

Social credit

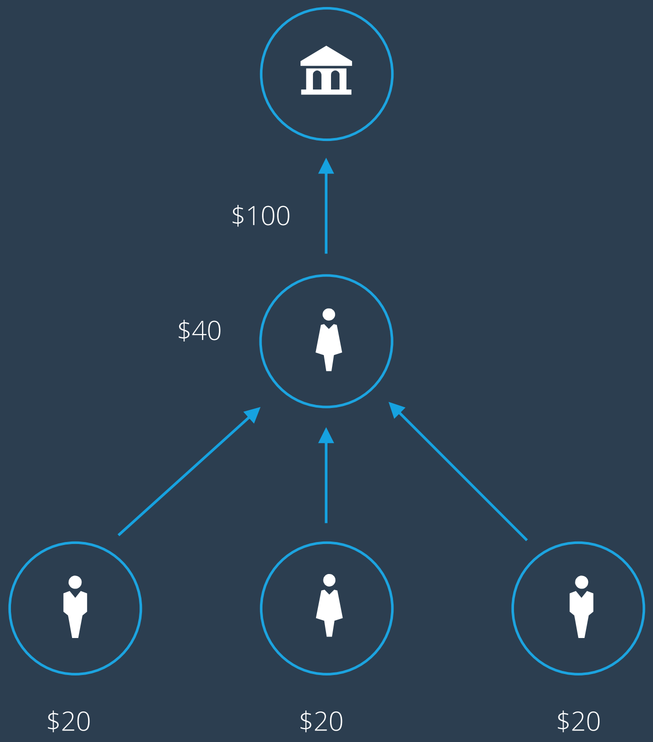
Allowance

Social credit



Allowance

Social credit



How RCL Works

Social credit

Works on RCL today

Considered a pretty crazy idea

Private Blockchains

Why would anyone want a private blockchain?

Private blockchains

Participants are controlled

Transactions can be private

No need for a native token

Why would anyone want a private blockchain?

Private blockchains

Attacks can be mitigated

Can react to legal process

Can be managed

Why would anyone want a private blockchain?

Private blockchains

Good for organizations of frenemies

Redundancy is built in

Can be self-governing

One Ledger to Rule Them All

One Ledger to Rule Them All

The great thing about ledgers

Banks have ledgers

People want different things from ledgers

We want innovation in ledgers

One ledger cannot satisfy everyone

One Ledger to Rule Them All

The great thing about ledgers

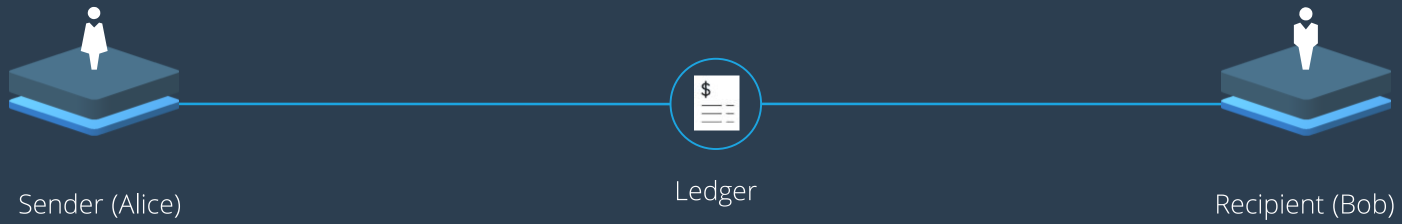
Ledgers should not be islands

We need a way to make payments across ledgers

It has to be a neutral standard

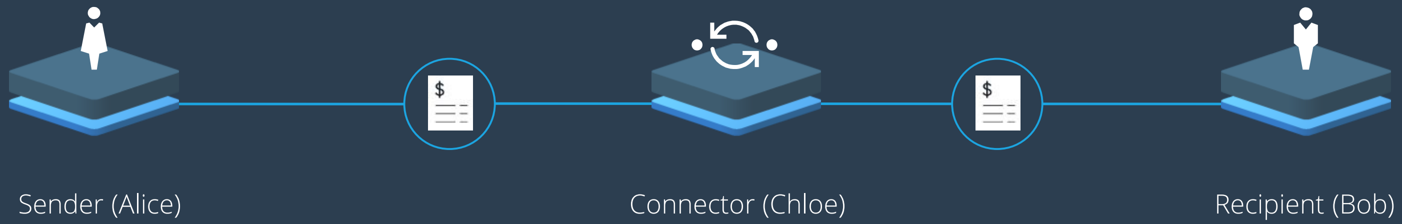
Interledger

Ledgers track accounts and balances

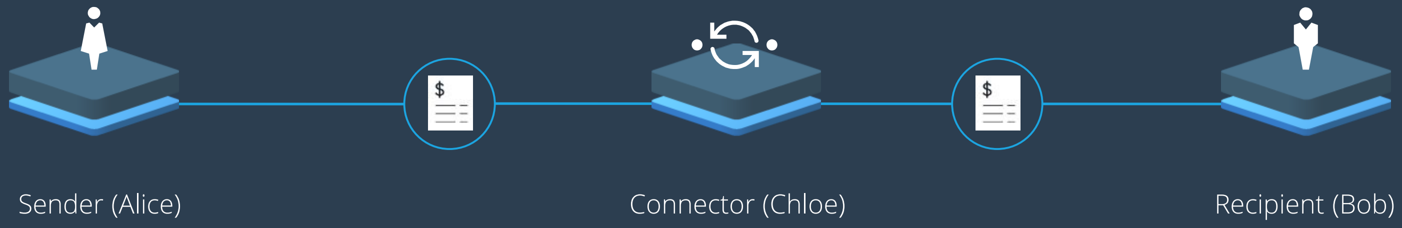


But not everyone is on the same ledger

Connectors relay money



Connectors relay money



Alice's Bank Ledger

Alice	100
Chloe	0

100

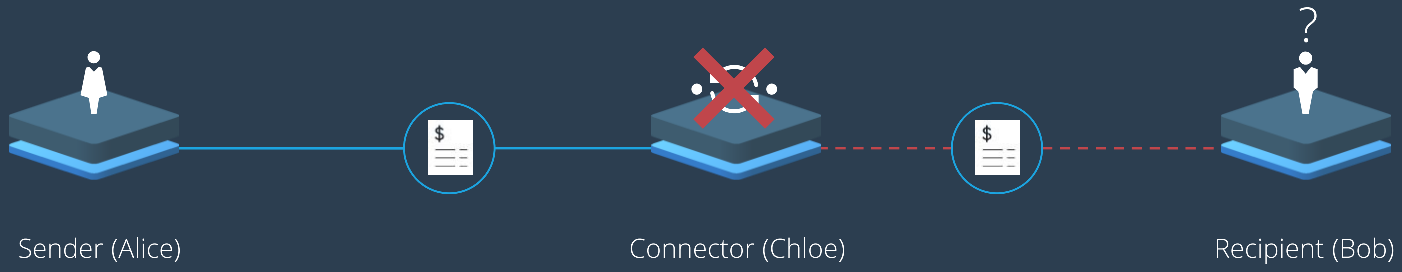
Bob's Bank Ledger

Chloe	100
Bob	0

100

What if the connector drops it?

Money would be lost



Alice's Bank Ledger

Alice	0
Chloe	100

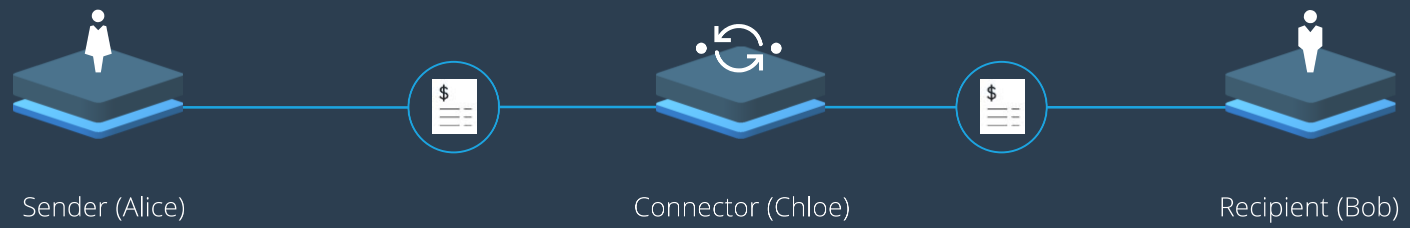
100

Bob's Bank Ledger

Chloe	100
Bob	0

Escrow provides security

Ledger-provided escrow reduces risk



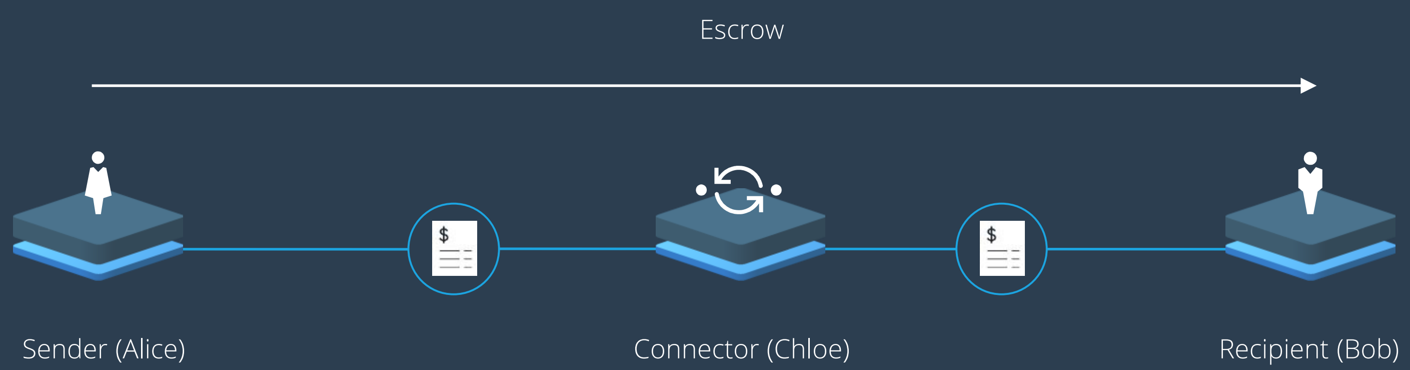
Alice's Bank Ledger

Alice	100
Escrow	0
Chloe	0

Bob's Bank Ledger

Chloe	100
Escrow	0
Bob	0

Funds are escrowed from left to right



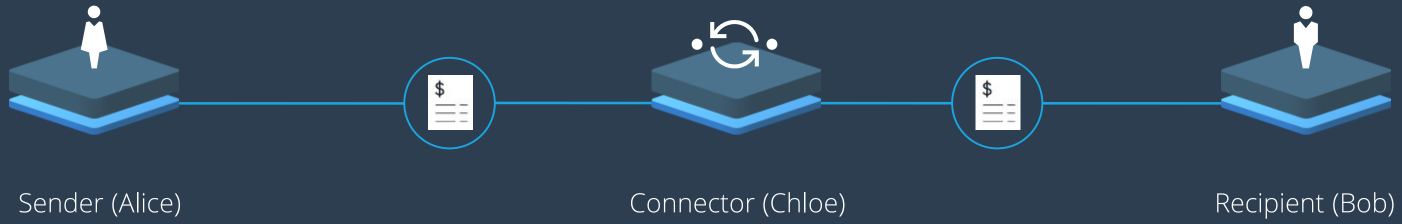
Alice's Bank Ledger

Alice	100
Escrow	0
Chloe	0

Bob's Bank Ledger

Chloe	100
Escrow	0
Bob	0

Sender puts funds into escrow

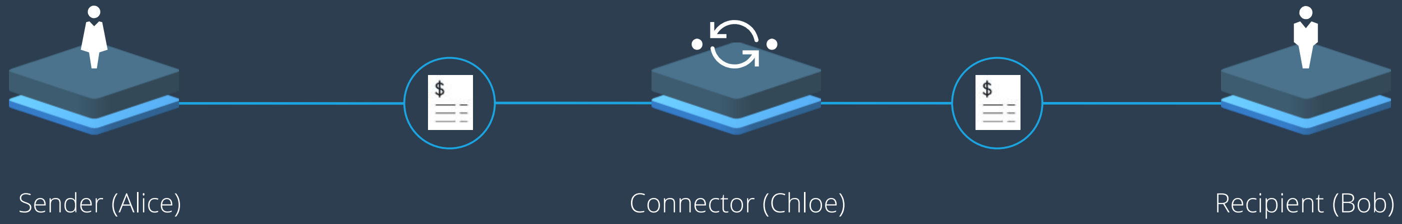


Alice’s Bank Ledger

Alice	100
Escrow	0
Chloe	0

100

Connector put funds into escrow



Alice's Bank Ledger

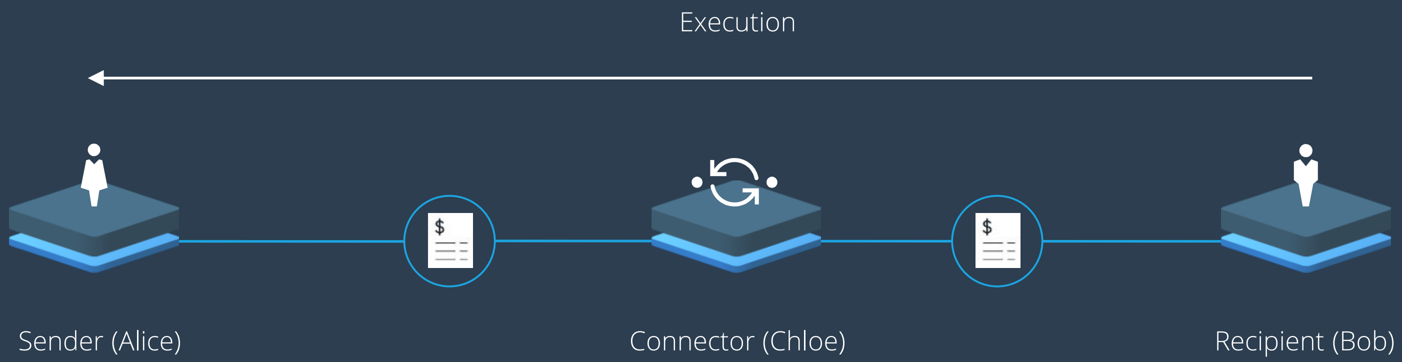
Alice	0
Escrow	100
Chloe	0

Bob's Bank Ledger

Chloe	100
Escrow	0
Bob	0

100

Transfers are executed right to left



Alice's Bank Ledger

Alice	0
Escrow	100
Chloe	0

Bob's Bank Ledger

Chloe	0
Escrow	100
Bob	0

Recipient signs receipt

Sender (Alice)

Connector (Chloe)

Recipient (Bob)

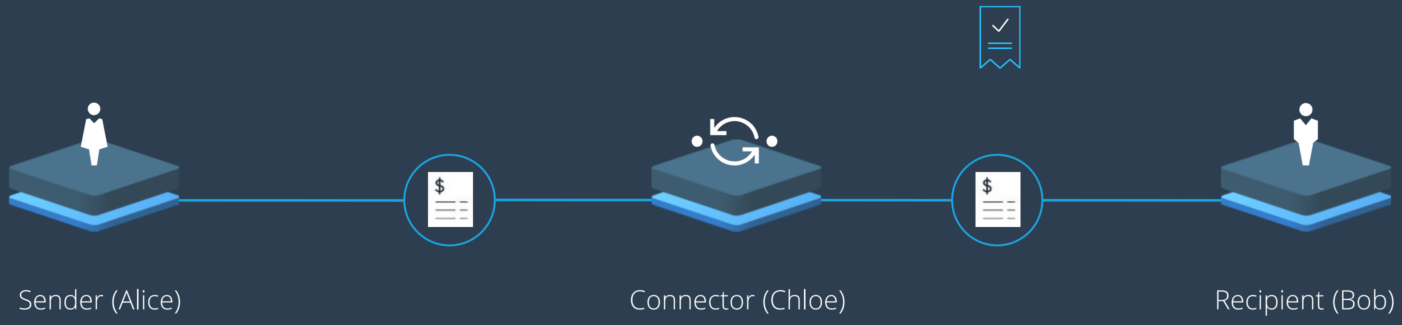
Alice's Bank Ledger

Alice	0
Escrow	100
Chloe	0

Bob's Bank Ledger

Chloe	0
Escrow	100
Bob	0

Receipt releases funds from escrow



Alice's Bank Ledger

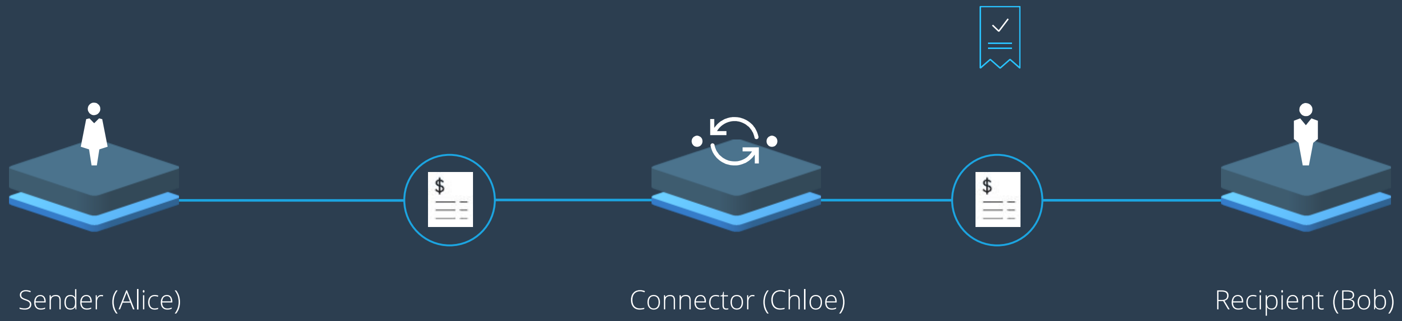
Alice	0
Escrow	100
Chloe	0

Bob's Bank Ledger

Chloe	0
Escrow	100
Bob	0

100

Receipt releases funds from escrow



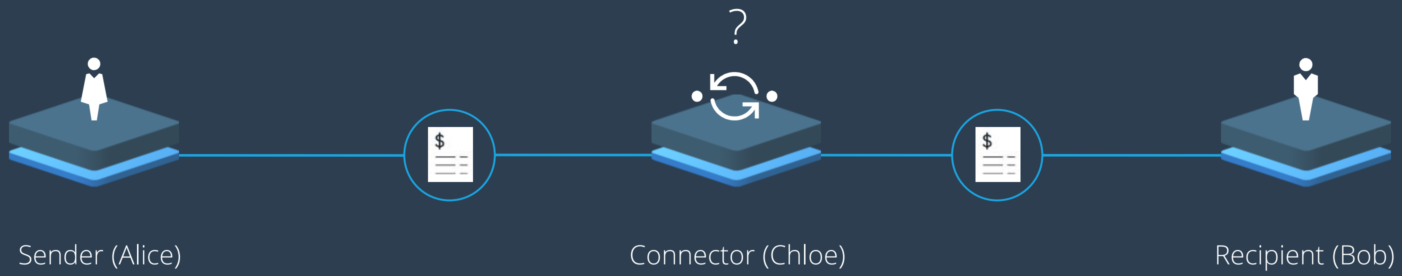
Alice's Bank Ledger

Alice	0
Escrow	100
Chloe	0

Bob's Bank Ledger

Chloe	0
Escrow	0
Bob	100

How does the connector get reimbursed?



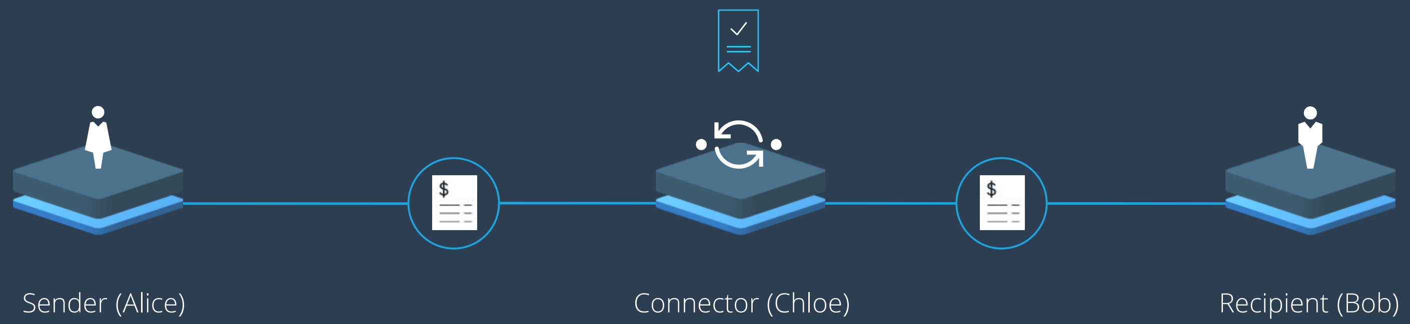
Alice's Bank Ledger

Alice	0
Escrow	100
Chloe	0

Bob's Bank Ledger

Chloe	0
Escrow	0
Bob	100

Connector gets receipt from ledger



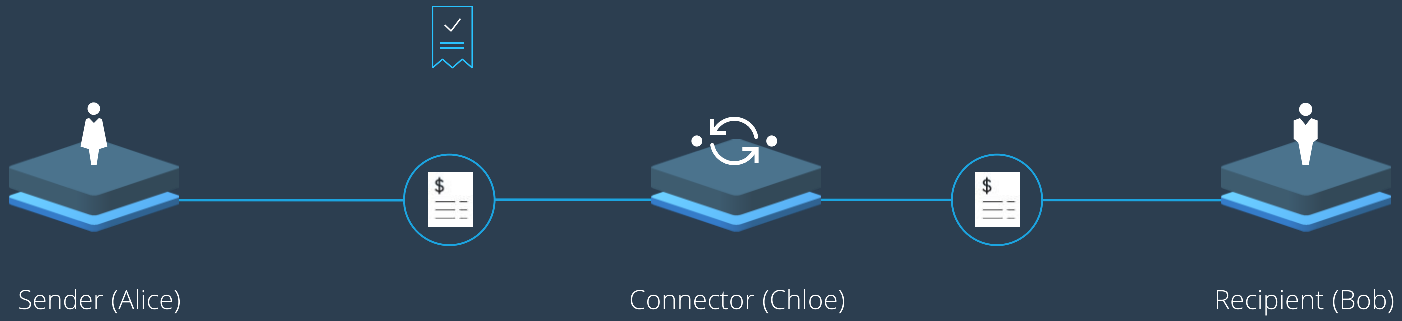
Alice's Bank Ledger

Alice	0
Escrow	100
Chloe	0

Bob's Bank Ledger

Chloe	0
Escrow	0
Bob	100

Connector passes on the receipt



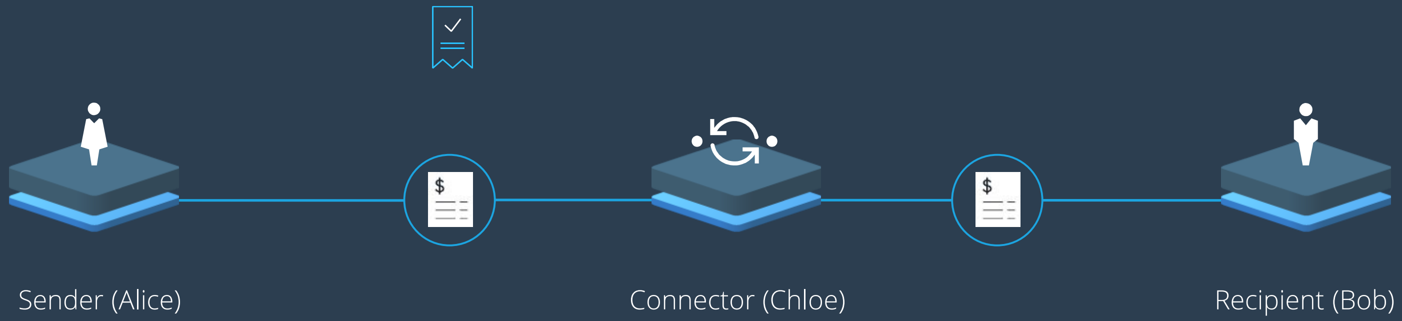
Alice's Bank Ledger

Alice	0
Escrow	100
Chloe	0

Bob's Bank Ledger

Chloe	0
Escrow	0
Bob	100

Receipt releases funds from escrow



Alice's Bank Ledger

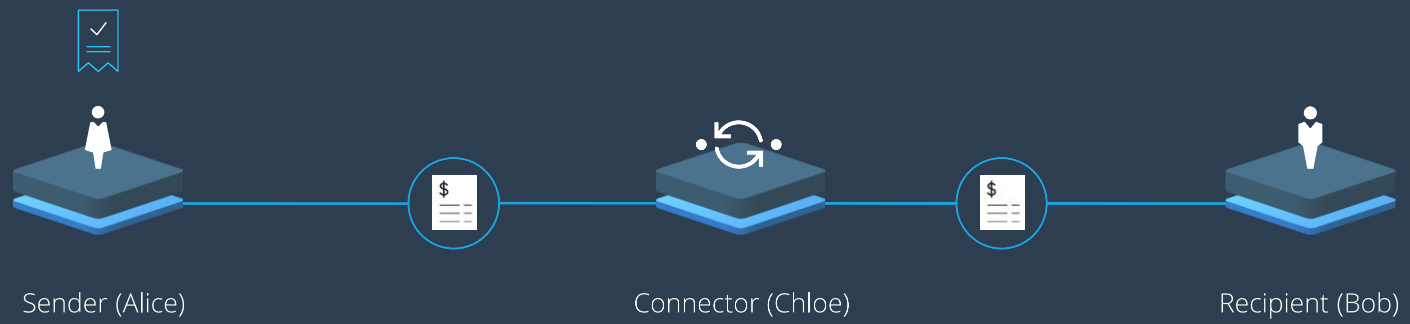
Alice	0
Escrow	100
Chloe	0

100

Bob's Bank Ledger

Chloe	0
Escrow	0
Bob	100

Payment is complete



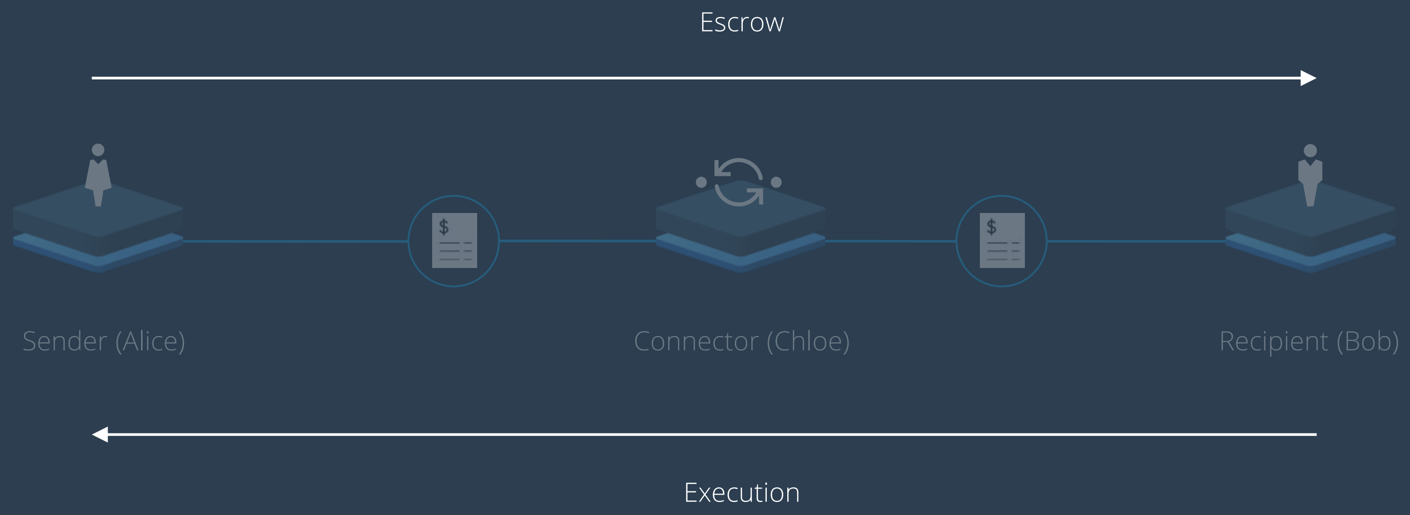
Alice's Bank Ledger

Alice	0
Escrow	0
Chloe	100

Bob's Bank Ledger

Chloe	0
Escrow	0
Bob	100

Transfers are escrowed L2R, executed R2L



Interledger

The ledger just needs to support two operations

Lock: Hold funds

Transfer: Release funds

Most ledgers can easily do this

Interledger

Cryptoconditions specify the release rules

Precise specification ensures agreement

One ledger’s receipt is another ledger’s release condition

Interledger

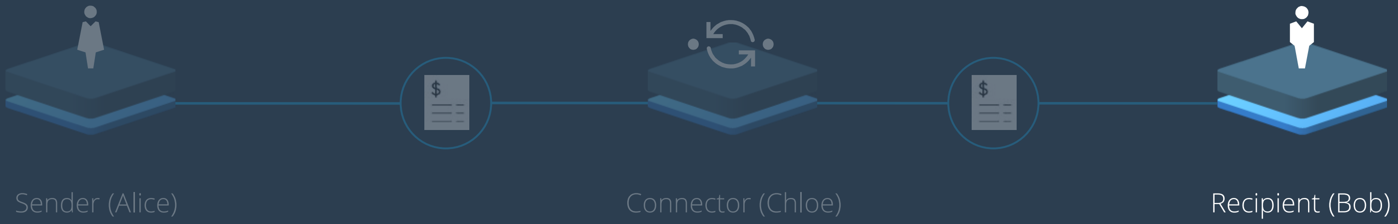
Leverages the trust that already exists

Anyone who has funds on a ledger trusts that ledger

Anyone willing to receive funds on a ledger trusts that ledger

Nobody has to trust the connectors

Bob



Alice’s Bank Ledger

Alice	0
Escrow	0
Chloe	100

Bob’s Bank Ledger

Chloe	0
Escrow	0
Bob	100

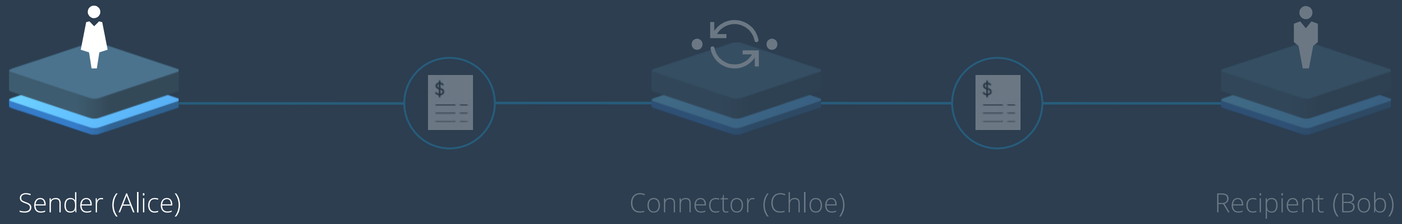
Bob

Must trust his ledger, since it will hold his money

Does not want Alice to have proof of payment unless he is assured funds

Does not trust Alice or Chloe

Alice



Alice's Bank Ledger

Alice	0
Escrow	0
Chloe	100

Bob's Bank Ledger

Chloe	0
Escrow	0
Bob	100

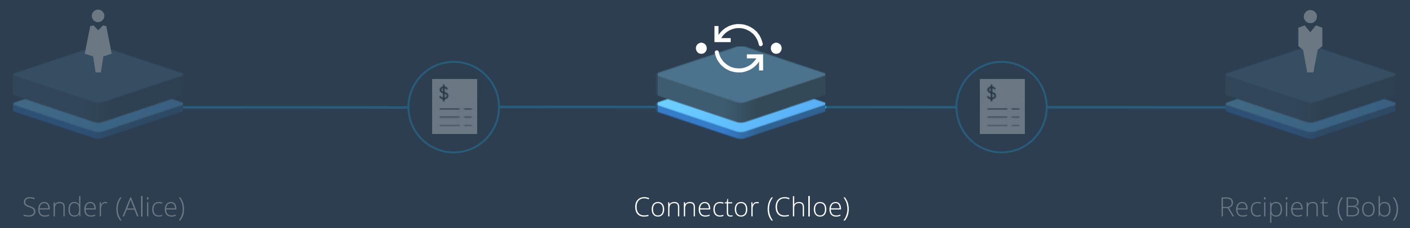
Alice

Must trust her ledger, since it has her money

Does not want to lose funds without a receipt Bob must honor

Does not trust Chloe

Chloe



Alice's Bank Ledger

Alice	0
Escrow	0
Chloe	100

Bob's Bank Ledger

Chloe	0
Escrow	0
Bob	100

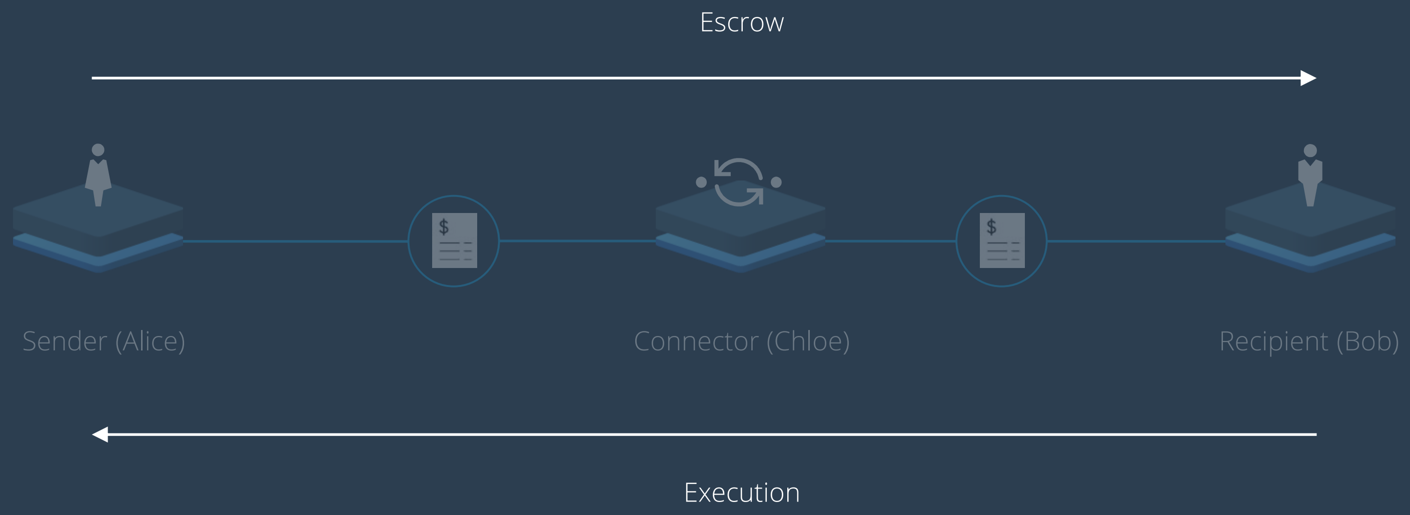
Chloe

Must trust both ledgers

Does not trust Alice or Bob

Does not want to pay Bob unless he gets paid by Alice

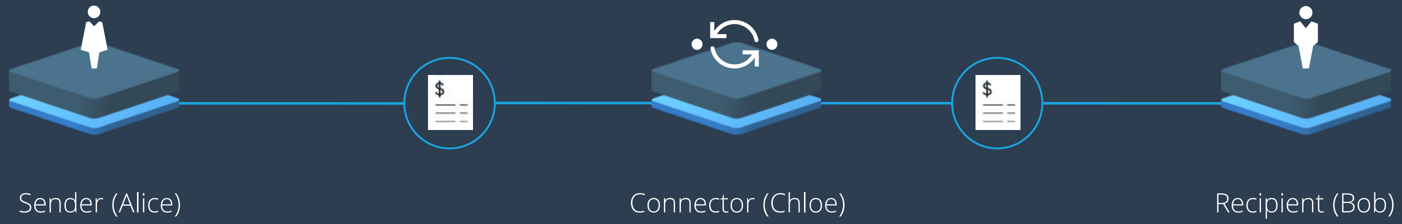
Mission Accomplished!



Is it really that simple?

Sometimes

Sender puts funds into escrow



Alice's Bank Ledger

Alice	100
Escrow	0
Chloe	0

100

Bob's Bank Ledger

Chloe	100
Escrow	0
Bob	0

Release condition is payment to recipient

Sender (Alice) | Connector (Chloe) | Recipient (Bob)

Alice's Bank Ledger

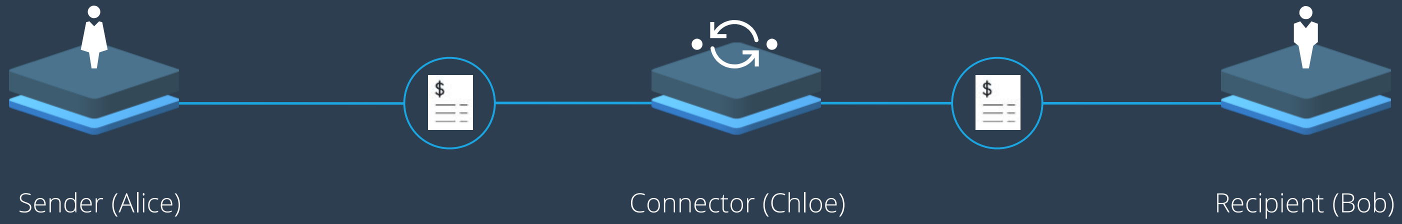
Alice	100
Escrow	0
Chloe	0

100

Bob's Bank Ledger

Chloe	100
Escrow	0
Bob	0

But what is the failure condition?



Alice's Bank Ledger

Alice	100
Escrow	0
Chloe	0

100

Bob's Bank Ledger

Chloe	100
Escrow	0
Bob	0

Failure conditions

Connector cannot meet payment terms

Connector loses connectivity

Ledger loses connectivity

Some component stops operating

Failure conditions

Sender wants fast release

Otherwise, sender must trust connector or take risk

Connector does not want to incur risk

Risk stems from inability to get receipt to the other ledger

Low-value payments

You can use a release time

Connector can price in the risk of failure

Sufficient for small payments

High-value payments

Must ensure agreement on transaction success or failure

Long lock times are a problem

Need proof that something did not happen

Simple schemes cannot provide this “proof of absence”

Byzantine Generals

Byzantine Generals Problem

Each side should commit if, and only if, the other side will

At some point, at least one side must commit irrevocably

But that will never happen unless one side commits irrevocably first

But we cannot commit irrevocably until we know the other side has

PBFT

Byzantine agreement protocol

Can tolerate some faulty nodes

Non-faulty nodes agree

Combines nicely with crypto

Byzantine Generals Problem

High-value payments in ILP is a BG problem

Consensus is a BG problem

The double-spend problem is a BG problem

Actually, lots of problems are BG problems

Byzantine Generals in ILP

Very easy to solve

We have algorithms like PBFT

Arrangement can be private, ephemeral

What about blockchains?

Easy for private blockchains

Harder problem for public blockchains

Proof of work is a solution

Distributed agreement is another

Now that we're all experts

Development Challenges

Attack surface

Public blockchains must be fortresses

Code is public

Vulnerabilities are painful

This makes development much slower, maybe 10X

Public APIs

Blockchain development challenges

Resource Management

We have to keep up with the network

We have to respond to remote queries

We have to respond to local queries

We have to cache

Data Representation

Binary Formats

Transactions need to be signed

All kinds of objects need to be hashed

This requires unique binary representations

Data Representation

Binary Formats

Non-binary representations are convenient too

Humans like them

Javascript likes them

Blockchain development challenges

Performance

Some tasks are embarrassingly parallel

Some tasks don’t parallelize at all

It is all important

Blockchains do not scale horizontally … yet!

Blockchain development challenges

Isolation

Transaction operations must be deterministic

Some designs fail catastrophically otherwise

It is easy to get non-deterministic behavior by accident

This is a hard problem for smart contracts

Meeting challenges with C++

C++ features

Move semantics

Expensive types can have value semantics

Copies are only made when necessary

Often requires no code changes

When it does, they're usually minimal

C++ features

Lambdas

Enables visitor patterns

Allows you to preserve layering

Allows work to be deferred and dispatched

Makes coroutines simple

C++ features

Compile-time polymorphism

Polymorphic code gets fully-optimized

It can even inline

Responsibilities can be separated

C++ features

Type composition

Write code once

Get excellent API

boost::optional

std::shared_ptr / std::weak_ptr

Code isolation

Namespaces

Separation of implementation from API

API for use, API for derivation

C++ features

Mature tools

We have at least three solid compilers

Great tools for performance analysis

Tools for finding concurrency violations

Libraries for just about everything

C++ features

```
In file included from /usr/include/boost/intrusive/rbtree.hpp:23:0,
                 from /usr/include/boost/intrusive/set.hpp:20,
                 from src/beast/include/beast/http/basic_headers.hpp:14,
                 from src/beast/include/beast/http/message.hpp:11,
                 from src/beast/include/beast/http/message_v1.hpp:11,
                 from src/ripple/server/Handoff.h:24,
                 from src/ripple/overlay/Overlay.h:26,
                 from src/ripple/app/ledger/impl/InboundLedger.cpp:31,
                 from src/ripple/unity/app_ledger.cpp:34:
/usr/include/boost/intrusive/bstree.hpp:653:91: error: expected primary-expression before '>' token
    static const bool stateful_value_traits = detail::is_stateful_value_traits<value_traits>::value;
                                                                                           ^
/usr/include/boost/intrusive/bstree.hpp:653:92: error: '::value' has not been declared
    static const bool stateful_value_traits = detail::is_stateful_value_traits<value_traits>::value;
                                                                                            ^~
/usr/include/boost/intrusive/bstree.hpp:653:92: note: suggested alternative:
In file included from src/ripple/app/ledger/AcceptedLedgerTx.cpp:25:0,
                 from src/ripple/unity/app_ledger.cpp:23:
src/ripple/protocol/JsonFields.h:448:7: note:   'ripple::jss::value'
 JSS ( value );                       // out: STAmount
       ^
src/ripple/protocol/JsonFields.h:30:47: note: in definition of macro 'JSS'
 #define JSS(x) constexpr ::Json::StaticString x ( #x )
                                               ^
In file included from /usr/include/boost/intrusive/rbtree.hpp:23:0,
                 from /usr/include/boost/intrusive/set.hpp:20,
                 from src/beast/include/beast/http/basic_headers.hpp:14,
                 from src/beast/include/beast/http/message.hpp:11,
                 from src/beast/include/beast/http/message_v1.hpp:11,
                 from src/ripple/server/Handoff.h:24,
                 from src/ripple/overlay/Overlay.h:26,
                 from src/ripple/app/ledger/impl/InboundLedger.cpp:31,
                 from src/ripple/unity/app_ledger.cpp:34:
/usr/include/boost/intrusive/bstree.hpp:660:47: error: 'is_safe_autounlink' was not declared in this scope
    static const bool safemode_or_autounlink = is_safe_autounlink<value_traits::link_mode>::value;
                                               ^~~~~~~~~~~~~~~~~~
/usr/include/boost/intrusive/bstree.hpp:660:47: note: suggested alternative:
```

Maybe not so much

C++ features

Hand-optimized primitives

Very little code is worth hand-optimizing

But for the code that is, the payoff is enormous

Digital signatures are worth it

Calls are cheap, sometimes even inline

Leverage work across projects

C++ features

Slicing Problem

Had to include one bad thing

Programmers like value semantics

Polymorphism and value semantics mix badly

Slicing

Not great solutions

Raw pointers

Unique pointers

Shared pointers

Clone idiom

Slicing

We don’t need one great solution

Compile-time polymorphism, templates

Maybe **std::variant** in C++17?

Winning

#Winning

Caching

Use of strong and weak pointers

Cache holds strong and weak pointers

Access promotes a weak pointer to a strong pointer

Time demotes a strong pointer to a weak pointer

Use pins an item in the cache, good things happen for free

Caching

Algorithmic complexity attacks

You have to use hashing

Attackers can, to some extent, choose the hashes

You cannot keep the scheme secret

Solution: salted hashes

NuDB

Key / Value Store

Fixed length keys

Variable length data

Retrieve by key only (or traverse)

NuDB

Key / Value Store

Transactions

Bits of hash trees

Ledger state entries

NuDB

What’s out there

Memory demand scales with data size

Relies on caching for performance

Performance drops as data size increases

NuDB

Tradeoffs

Assumes caching is useless

Performance levels off as data size increases

Then no penalty for massive databases

Memory use scales with write rate

NuDB

Tradeoffs

What is the best you can do?

For fetches of data not present, 1 I/O

For fetches of data present, 2 I/Os

Performance limit is SSD IOPs

NuDB comes really close to that

NuDB

Design features

Data is append only

Two or three files are used

Writes are journaled

NuDB

Design features

Index consists of hash buckets

Bucket count is dynamically increased

Writes do not block reads

Reads do not block each other

NuDB

C++ features

Header only

Templated visitor

Compile-time asserts

NuDB

Templated visitor

```
template <class Codec, class Function>
bool
visit(
        path_type const& path,
        std::size_t read_size,
        Function&& f)
{
```

NuDB

Static assert

```
using hash_t = uint48_t;

static_assert(field<hash_t>::size<=sizeof(std::size_t),"");
```

Using C++

Beast

Header only

Provides Boost-like API

Supports HTTP and websockets

Asynchronous and synchronous

Using C++

Beast

```cpp
#include <beast/core/to_string.hpp>
#include <beast/websocket.hpp>
#include <boost/asio.hpp>
#include <iostream>
#include <string>

int main()
{
    // Normal boost::asio setup
    std::string const host = "echo.websocket.org";
    boost::asio::io_service ios;
    boost::asio::ip::tcp::resolver r{ios};
    boost::asio::ip::tcp::socket sock{ios};
    boost::asio::connect(sock,
        r.resolve(boost::asio::ip::tcp::resolver::query{host, "80"}));

    // WebSocket connect and send message using beast
    beast::websocket::stream<boost::asio::ip::tcp::socket&> ws{sock};
    ws.handshake(host, "/");
    ws.write(boost::asio::buffer("Hello, world!"));

    // Receive WebSocket message, print and close using beast
    beast::streambuf sb;
    beast::websocket::opcode op;
    ws.read(op, sb);
    ws.close(beast::websocket::close_code::normal);
    std::cout << to_string(sb.data()) << "\n";
}
```

```cpp
#include <beast/http.hpp>
#include <boost/asio.hpp>
#include <iostream>
#include <string>

int main()
{
    // Normal boost::asio setup
    std::string const host = "boost.org";
    boost::asio::io_service ios;
    boost::asio::ip::tcp::resolver r{ios};
    boost::asio::ip::tcp::socket sock{ios};
    boost::asio::connect(sock,
        r.resolve(boost::asio::ip::tcp::resolver::query{host, "http"}));

    // Send HTTP request using beast
    beast::http::request_v1<beast::http::empty_body> req;
    req.method = "GET";
    req.url = "/";
    req.version = 11;
    req.headers.replace("Host", host + ":" + std::to_string(sock.remote_endpoint().port()));
    req.headers.replace("User-Agent", "Beast");
    beast::http::prepare(req);
    beast::http::write(sock, req);

    // Receive and print HTTP response using beast
    beast::streambuf sb;
    beast::http::response_v1<beast::http::streambuf_body> resp;
    beast::http::read(sock, sb, resp);
    std::cout << resp;
}
```

Polymorphic currency types

Ripple has both a native currency and arbitrary assets

Some objects can hold a currency of either type

Some objects can only hold one kind of currency

Virtual functions not a good fit, partly due to slicing

Solution: templates

Concepts are light

Concepts cannot slice

Common code stays simple and easy to understand

Using C++

Solution: templates

```
template <class TIn, class TOut>
class TOfferStreamBase
{
...
protected:
        TOffer <TIn, TOut> offer_;
        boost::optional <TOut> ownerFunds_;
```

Fin