Cool Kids <3 Erlang

Erlang and the Bitcoin blockchain

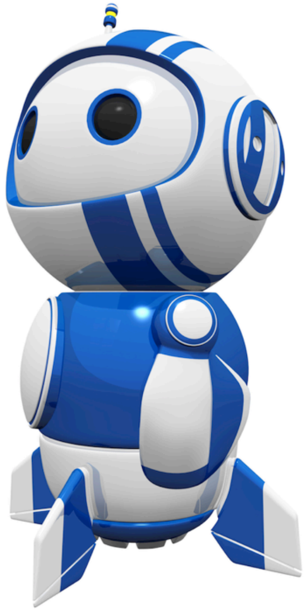
Erlang Factory Light Berlin 2014

42

Yes!

Talk

- Money
- Start Ups
- Bitcoin
- Erlang
- … all of that, shaken (not stirred).



“Do you understand money?”

Money is debt.

Money is debt.

Is it?

Money

With us for 5,000 years:

- Coins
- Notes
- Cards
- Crypto

Money

- Yap Stones
- Money in Games
- Bread Winning & Rent Taking

Money's a matter of functions four,

A Medium, a Measure, a Standard, a Store.

Moneylitility

- Scarcity
- Durability
- Portability
- Divisibility
- Verifiability
- Storability
- Fungibility
- Protectibilty
- Adoption

Bitcoin

- digital
- decentralized
- trust-less

Ledger and payment system.

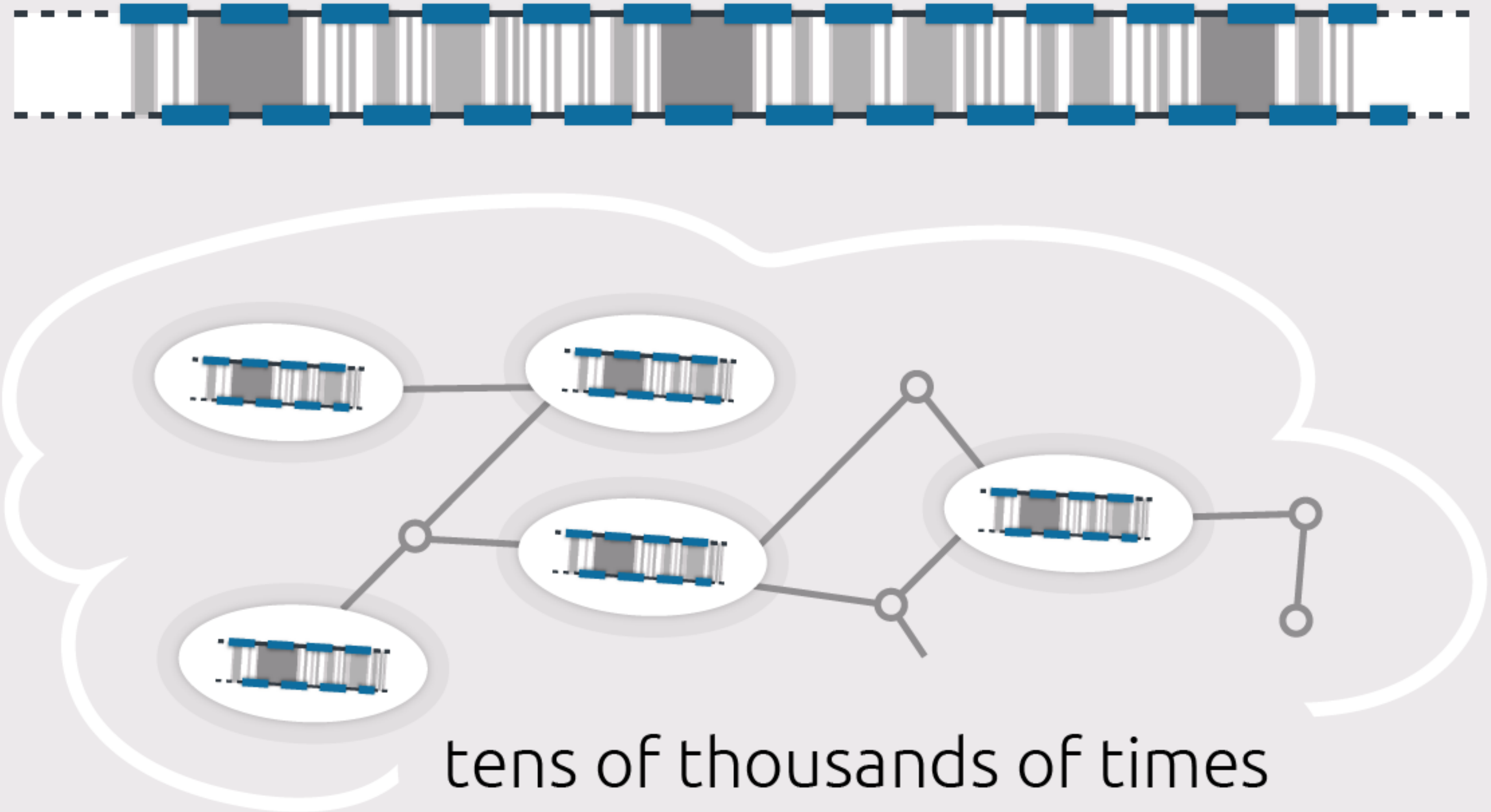
Bitcoin

- released in 2009
- $370 per bitcoin ca. market cap $5 billion
- accepted by ca. 80,000 merchants. 1-2 million users.
- more than $400 million VC money in 2014.

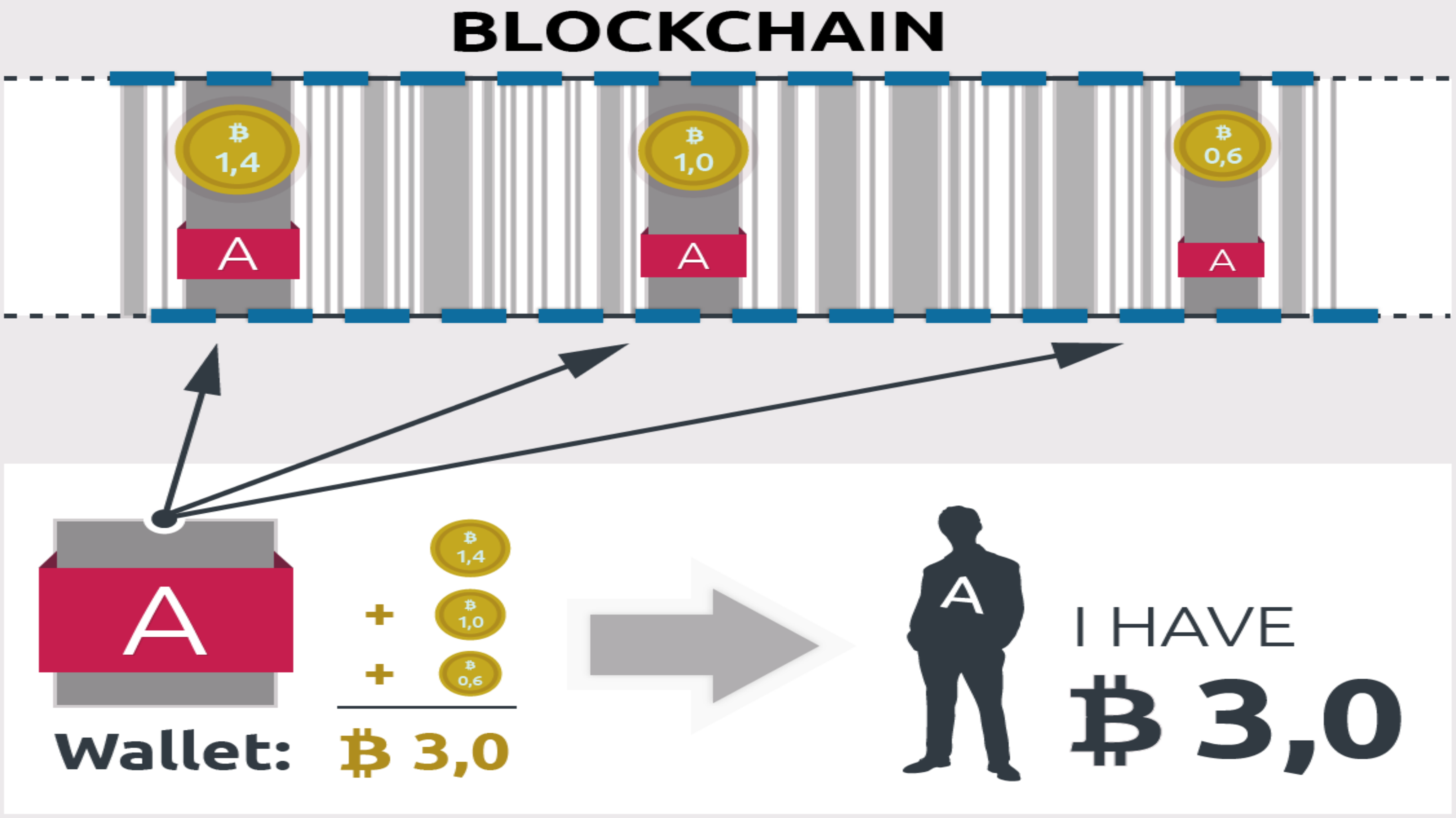
Bitcoin

How can that have value?

- Scarcity
- No central bank
- logarithmic inflation by protocol
  - 2014: 13.5 million bitcoins.
  - 2025: 20 million.
  - 2140: 21 million (max).



tens of thousands of times
mirrored in the cloud

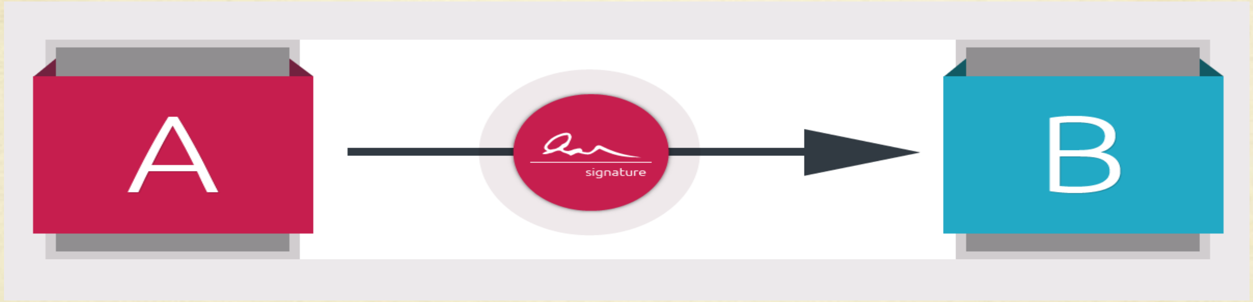

BLOCKCHAIN
฿ 1,4
A
฿ 1,0
A
฿ 0,6
A
A
฿ 1,4
+ ฿ 1,0
+ ฿ 0,6
Wallet: ฿ 3,0
A
I HAVE
฿ 3,0

Binary Cash

To be able to spend an unspent transaction, i.e. the money, the owner has a secret, digital key. These keys are conveniently managed by the wallet.

Each time a new transaction is initiated, the secret key to an unspent transaction is used to **sign** a new transaction.

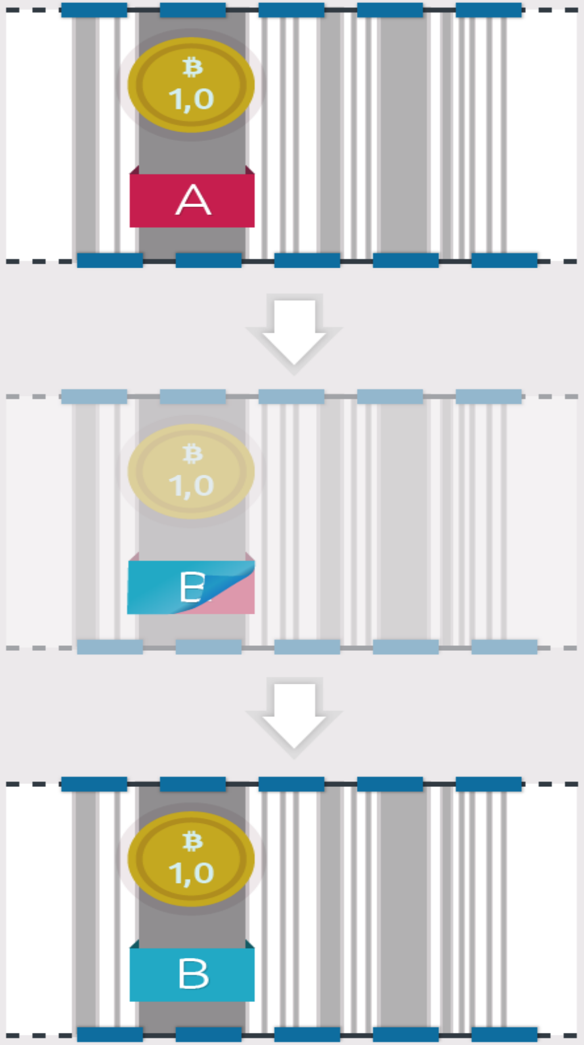
Since only the owner of a transaction has its secret key, only she can sign it, i.e. spend it.

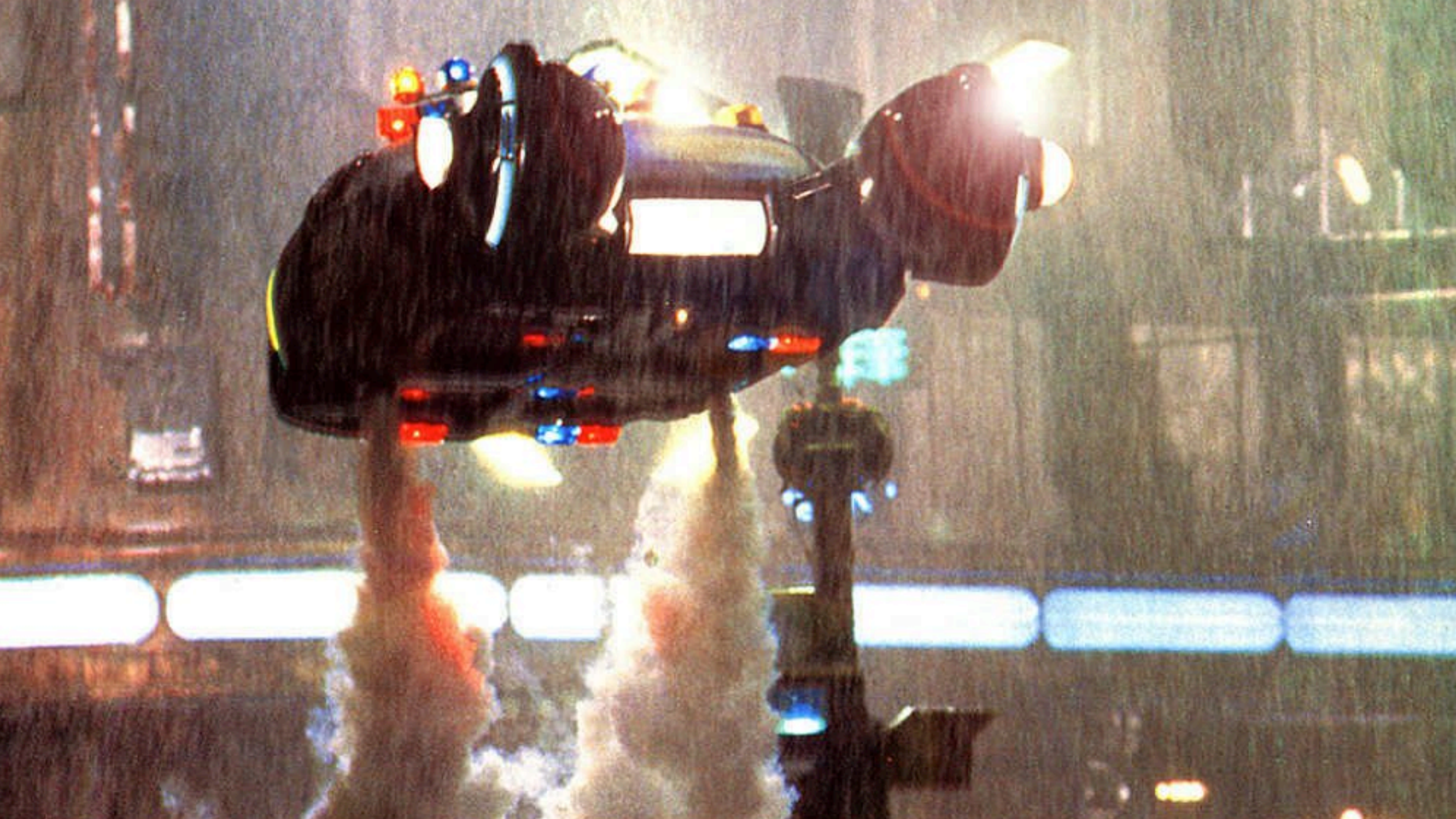


The ultimate beauty is that the secret keys effectively **are** the cash.

Paying with Bitcoins

- Paying to another person is facilitated by your wallet software creating and signing a new transaction to another person.
- The new transaction is *broadcast* to the network.
- There is no direct communication between the wallets.
- The wallet of the other person will see the new transaction and increase the wallet balance.
- An unspent transaction of yours is now sent, so your wallet will show a lower balance.
- There are *no* aggregate account balance figures in the blockchain, only transactions.



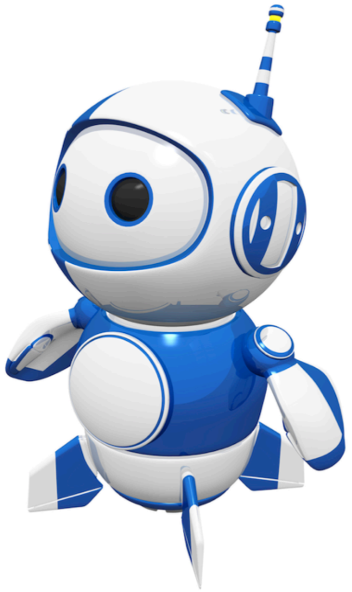


Start Up!

○ Insurances

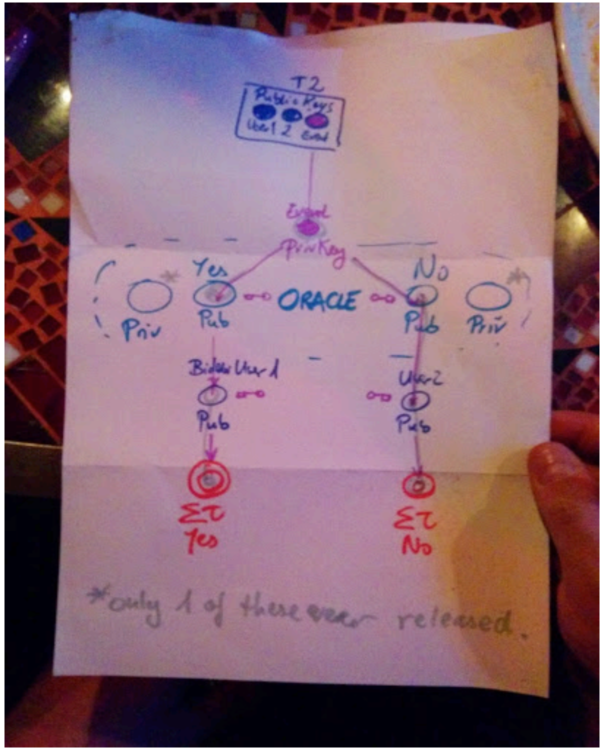
=> Sport Bets

=> World Cup!



The Project

- make a bet
- invite a friend
- friend enters a bet
- bet is decided
- money is cashed out



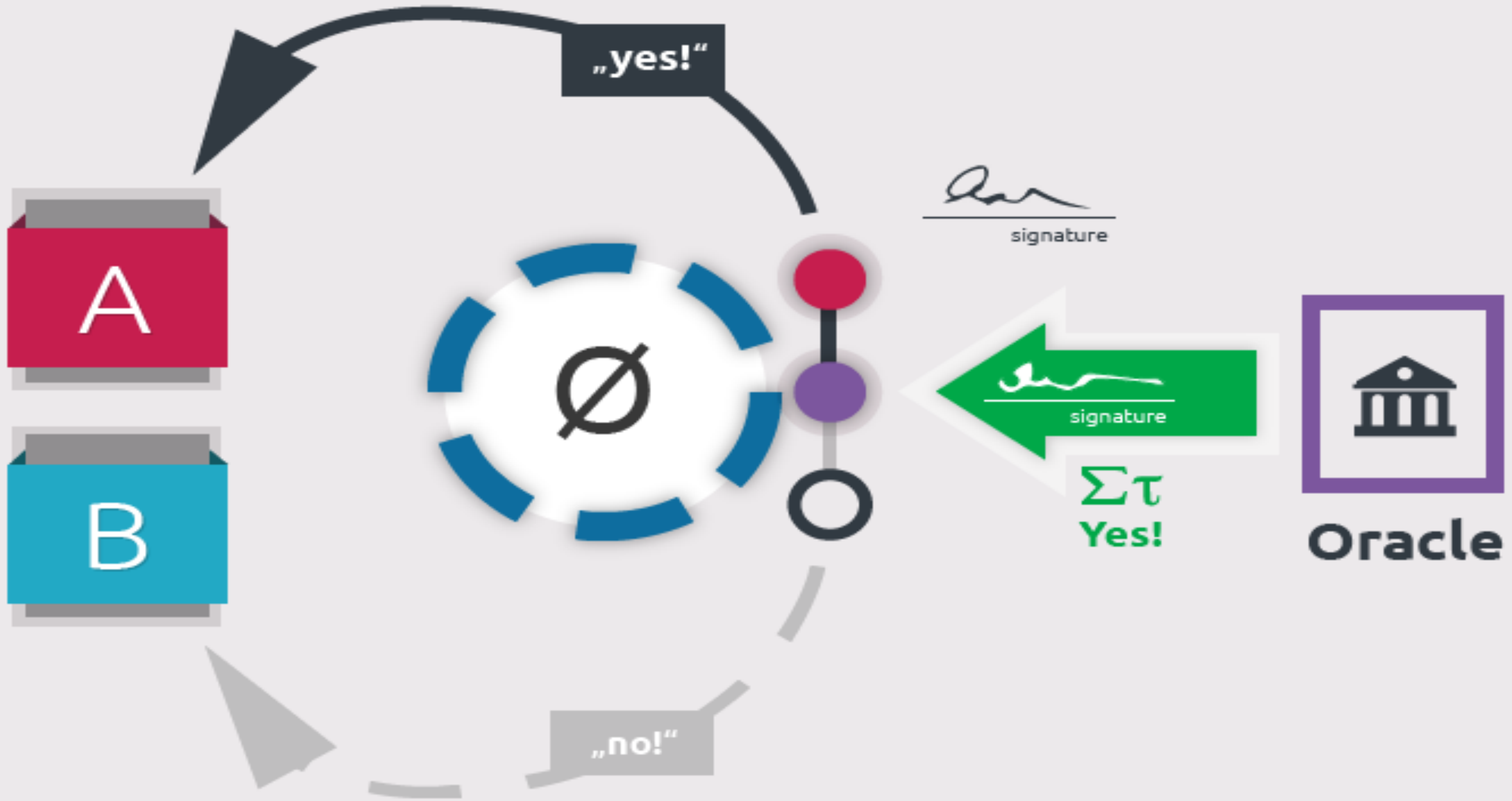
Multi-Signature Transactions

- Every transaction can have multiple parties paying into it, and multiple parties receiving the money.

- A transaction can also be constructed to require multiple keys to spend it. Such are called *multi-signature transactions* and are similar in function to a bank account that needs more than one signature to spend from. It's the same principle but with unprecedented, finer granularity, allowing for new business models.

- Multi-signature transactions can also be constructed to be spendable when only two out of three keys are present.

- For a multi-signature transaction, when the keys are known by separate people exclusively, then they own the Bitcoins collectively. no-one can spend the transation, i.e. the money, on its own.

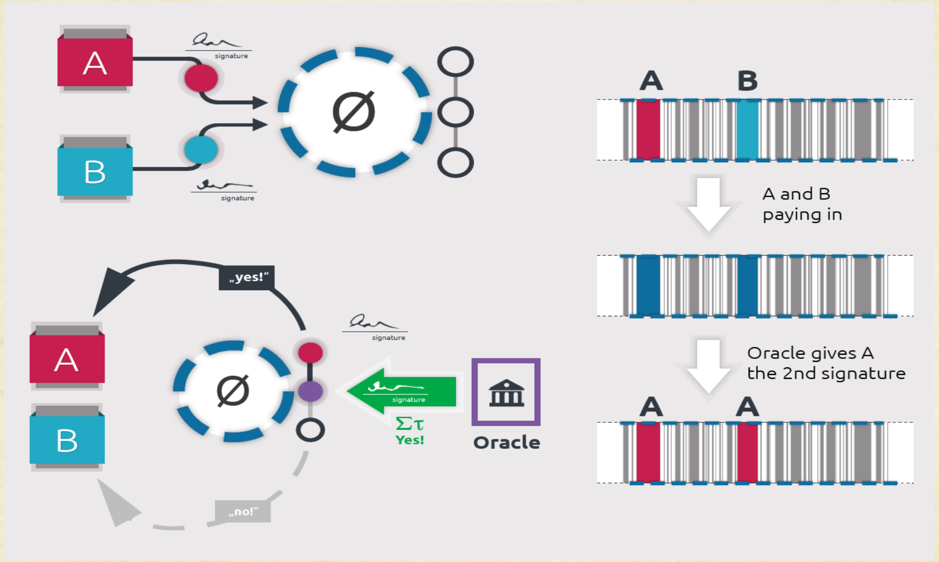

„yes!“
signature
A
B
Ø
signature
Στ
Yes!
Oracle
„no!“

Oracles

- Oracles provide facts expressed as cryptographic keys.
- Such oracle keys can be used to create escrow keys, which can unlock and spend micro-contracts as described, independently of the bet parties and the provider of the betting contract, Betwarp.
- Oracle services do not generally know about a contract being based on their service. They publish keys of real world events as a one-way service. They publish only one set of keys per event, not dealing with individual contracts.
- We are working with the leading oracle key provider, Mr Edgar, to establish an economy with independent event arbiters. Thus, in effect the information, and the escrow key expressing who wins a bet, are provided independently of us.

Micro Betting Contracts

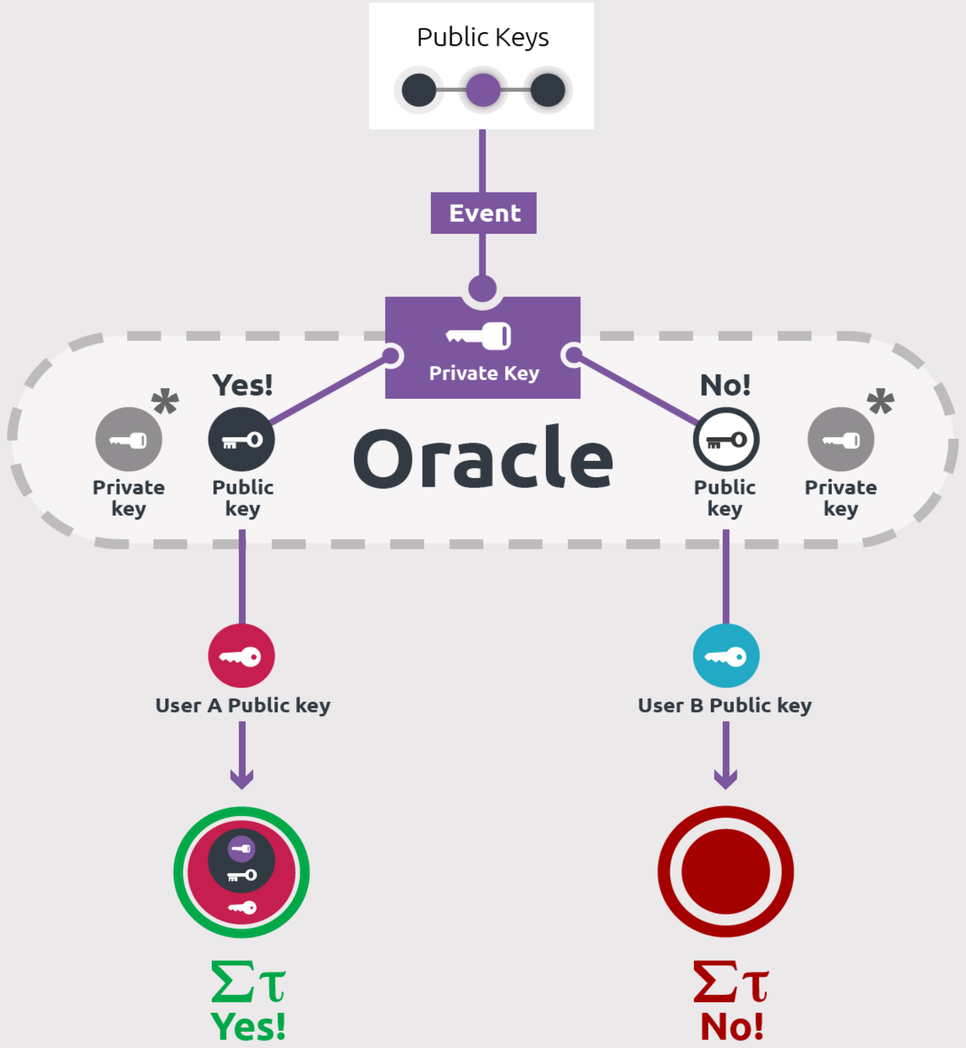
- Betwarp facilitates bets by creating a micro-contract between the betting parties where they lock in bitcoin which can be unlocked only with the aid of an oracle.



- This is a standard Bitcoin 2-out-of-3 transaction, where each party holds one secret key. When the oracle reveals the winning key, the winner cashes in.

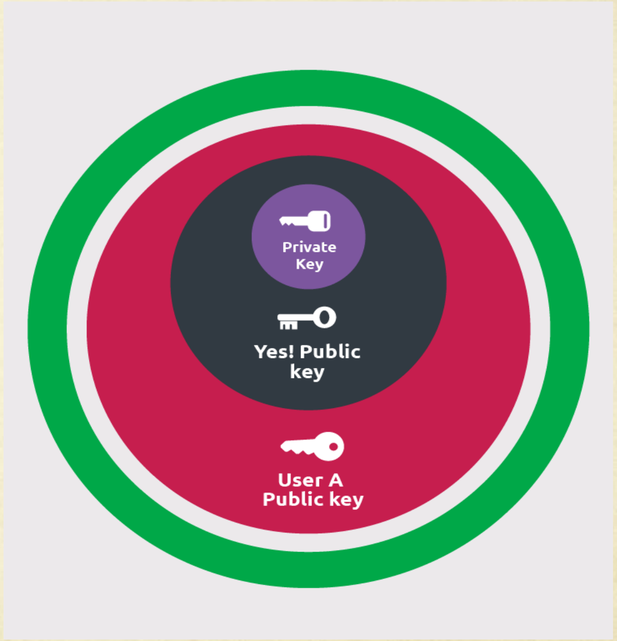
Double Enryption

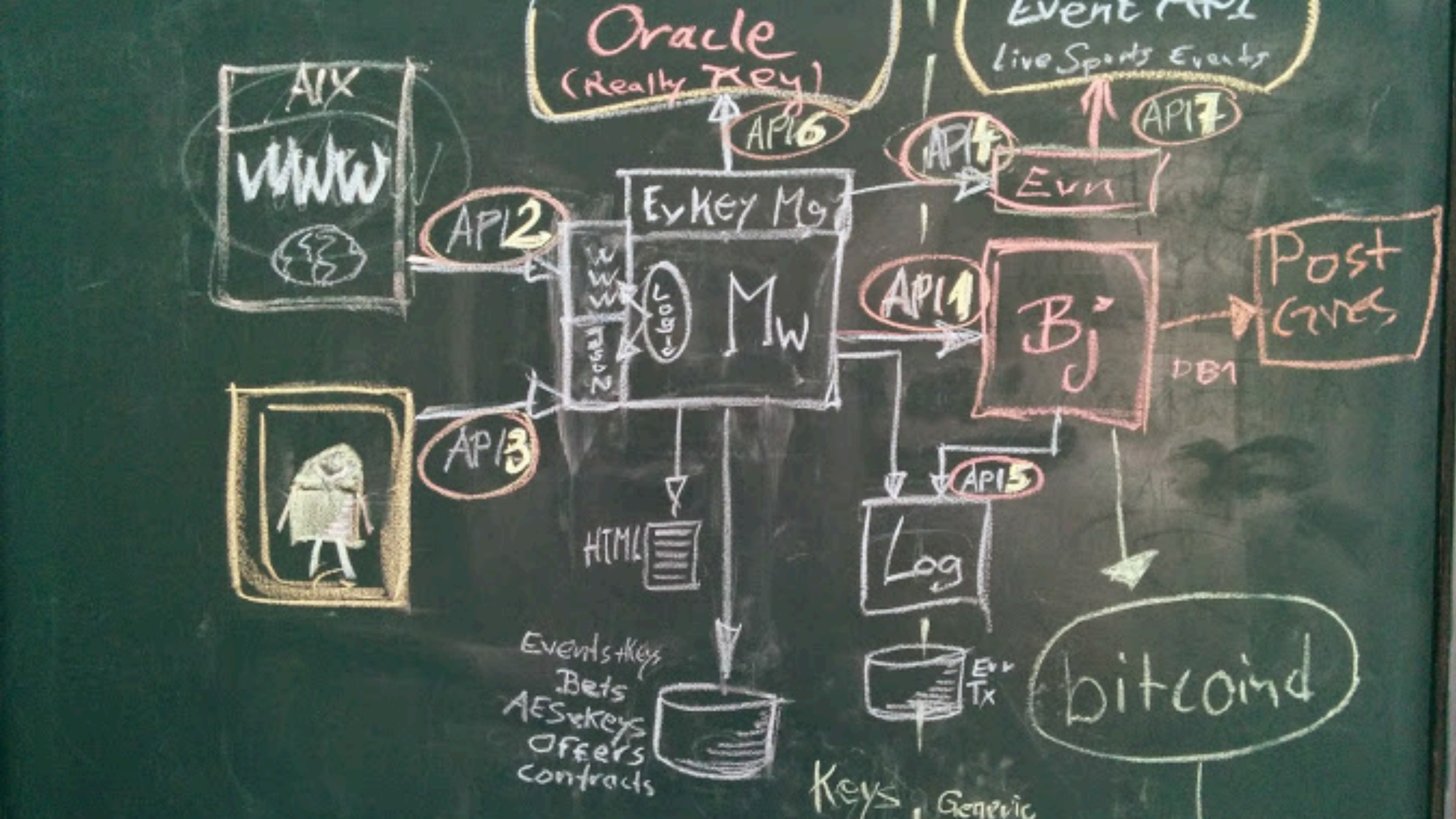
- Betwarp refines the oracle keys, prior to creating a bet.
- A winning key is created that is then encrypted with the oracle keys, Yes and No.
- The result is then encrypted again, using the parties' public keys.
- For the mobile apps, the secret keys are not received from us but rather, created in the app we provide, literally in the hands of the user, on his mobile device.



Double Enrypted Key Provision

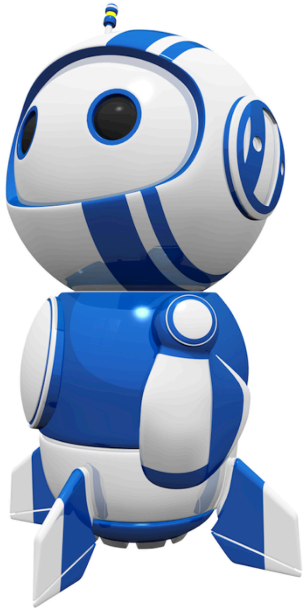
- The winner can unlock the secret winning key – which in turn unlocks the micro-contract – with the help of the secret key that the oracle publishes after the outcome of the event the bet referred to.
- Neither BetWarp nor the oracle ever have access to the secret keys of the betting parties.
- For delivery another encryption layer is added to make sure that no-one can steal the keys once the oracle key is published.




Oracle
(Reality Key)
Event API
Live Sports Events
API6
API4
API7
AIX
WWW
EvKey Mg
Evn
API2
WWW
Logic
Mw
Json
API1
Bj
Post
Gres
DB1
API3
API5
HTML
Log
Events+Keys
Bets
AESeKeys
Offers
contracts
Evr
Tx
Keys
Generic
bitcoind

The First Attempt

- use Java
- battle-tested, widely used, great library
- stuck with special cases



Erlang to the Rescue

- crashes don't pollute state
- retries often succeed
- all code in front of you
- bit-syntax for the win

© Can Stock Photo Inc. / 3DClipArtsDE

The Second Attempt

- use Erlang
- parse bits on protocol level
- 500 lines of code

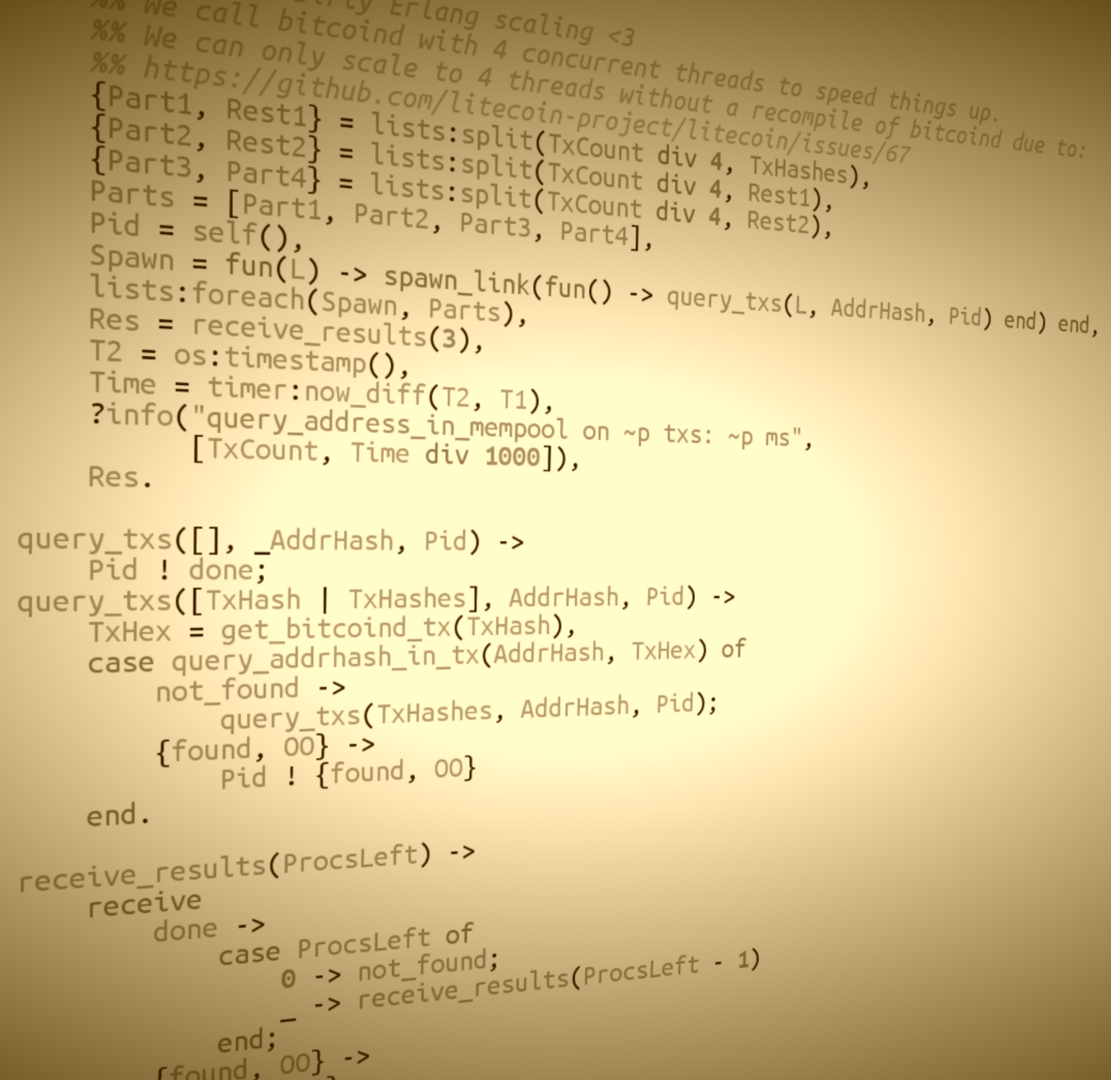
```
FinalT2 =
    <<TxVersion/binary,
      16#02,
      PrevOutPutTxHash:32/bytes,
      Index:32/little-integer,
      (varint(byte_size(GiverScriptSig)))/binary,
      GiverScriptSig/binary,
      Seq1/binary,
      PrevOutPutTxHash2:32/bytes,
      Index2:32/little-integer,
      (varint(byte_size(TakerScriptSig)))/binary,
      TakerScriptSig/binary,
      Seq2/binary,
      R8/binary
    >>,
                  <<?OP_DUP,
                    ?OP_HASH160,
                    20,
                    OutputAddrHash/binary,
                    ?OP_EQUALVERIFY,
                    ?OP_CHECKSIG>>,
UnsignedT3 =
    <<?TX_VERSION/binary,
      16#01, %% Tx input count
      (mw_lib:rev_bin(FinalT2Hash))/binary,
      16#00:32/little-integer, %% T2 has a single output
      (varint(byte_size(ScriptPubkey)))/binary,
      ScriptPubkey/binary,
      ?TX_IN_SEQ/binary,
      16#01, %% Tx output count
      OutputAmount:64/little-integer,
      (varint(byte_size(OutputScript)))/binary,
      OutputScript/binary,
      ?TX_LOCK_TIME/binary
```

```erlang
bitcoin_signature_der(<<16#30,
                        _TotalLen,
                        16#02,
                        RLen,
                        _R:RLen/bytes,
                        16#02,
                        SLen,
                        _S:SLen/bytes,
                        _HashType>>) -> true;
bitcoin_signature_der(_Bin) -> false.
```

```erlang
parse_block_header(<<FirstFour:4/bytes,
                    _HeaderLength:32/integer-little,
                    _Version:32/integer-little,
                    _HashPrevBlock:32/bytes,
                    _HashMerkleRoot:32/bytes,
                    Time:32/integer-little,
                    _TargetDifficulty:32/bits,
                    _Nonce:32/integer-little,
                    R/binary>> = Blocks) ->
```

```erlang
make_raw_inputs(Inputs) ->
    InputBin =
        fun({OutPointTxHash, OutPointIndex, InputScript, _Value}) ->
                <<
                  (mw_lib:rev_bin(OutPointTxHash))/binary,
                  OutPointIndex:32/little-integer,
                  (varint(byte_size(InputScript)))/binary,
                  InputScript/binary, %% scriptPubkey OR scriptSig
                  ?TX_IN_SEQ/binary
                >>
        end,
    <<(binary:list_to_bin(lists:map(InputBin, Inputs)))/binary>>.
```

```erlang
make_raw_inputs(Inputs) ->
    InputBin =
        fun({OutPointTxHash, OutPointIndex, InputScript, _Value}) ->
                <<
                 (mw_lib:rev_bin(OutPointTxHash))/binary,
                 OutPointIndex:32/little-integer,
                 (varint(byte_size(InputScript)))/binary,
                 InputScript/binary, %% scriptPubkey OR scriptSig
                 ?TX_IN_SEQ/binary
                >>
    end,
<<(binary:list_to_bin(lists:map(InputBin, Inputs)))/binary>>.
```



```
                  rly Erlang scaling <3
    %% We call bitcoind with 4 concurrent threads to speed things up.
    %% We can only scale to 4 threads without a recompile of bitcoind due to:
    %% https://github.com/litecoin-project/litecoin/issues/67
    {Part1, Rest1} = lists:split(TxCount div 4, TxHashes),
    {Part2, Rest2} = lists:split(TxCount div 4, Rest1),
    {Part3, Part4} = lists:split(TxCount div 4, Rest2),
    Parts = [Part1, Part2, Part3, Part4],
    Pid = self(),
    Spawn = fun(L) -> spawn_link(fun() -> query_txs(L, AddrHash, Pid) end) end,
    lists:foreach(Spawn, Parts),
    Res = receive_results(3),
    T2 = os:timestamp(),
    Time = timer:now_diff(T2, T1),
    ?info("query_address_in_mempool on ~p txs: ~p ms",
          [TxCount, Time div 1000]),
    Res.

query_txs([], _AddrHash, Pid) ->
    Pid ! done;
query_txs([TxHash | TxHashes], AddrHash, Pid) ->
    TxHex = get_bitcoind_tx(TxHash),
    case query_addrhash_in_tx(AddrHash, TxHex) of
        not_found ->
            query_txs(TxHashes, AddrHash, Pid);
        {found, OO} ->
            Pid ! {found, OO}
    end.

receive_results(ProcsLeft) ->
    receive
        done ->
            case ProcsLeft of
                0 -> not_found;
                _ -> receive_results(ProcsLeft - 1)
            end;
        {found, OO} ->
```

```erlang
hybrid_aes_rsa_enc(Plaintext, RSAPubkey) ->
    %% TODO: validate entropy source! We may want to block for /dev/random
    %% to be sure the AES key is cryptographically strong.
    AESKey = crypto:strong_rand_bytes(16),
    %% PKCS #7 padding; value of each padding byte is the integer representation
    %% of the number of padding bytes. We align to 16 bytes.
    PaddingLen = 16 - (byte_size(Plaintext) rem 16),
    Padding = binary:copy(<<PaddingLen>>, PaddingLen),
    PaddedPlaintext = <<Plaintext/binary, Padding/binary>>,
    Ciphertext = crypto:block_encrypt(aes_cbc128, AESKey,
                                      ?DEFAULT_AES_IV, PaddedPlaintext),

    %% Use OAEP as it's supported by Tom Wu's rsa2.js (RSADecryptOAEP)
    %% http://en.wikipedia.org/wiki/Optimal_Asymmetric_Encryption_Padding
    {_RecordName, Modulus, Exponent} = RSAPubkey,
    EncAESKey = crypto:public_encrypt(rsa, AESKey, [Exponent, Modulus],
                                      rsa_pkcs1_oaep_padding),

    %% Distinguishable prefix to identify the binary in case it's on the loose
    <<(mw_lib:hex_to_bin(?BINARY_PREFIX))/binary,
      EncAESKey/binary,
      Ciphertext/binary>>.
```

"Is it – possible that the world, at large, does not get the awesomeness *of Erlang?"*