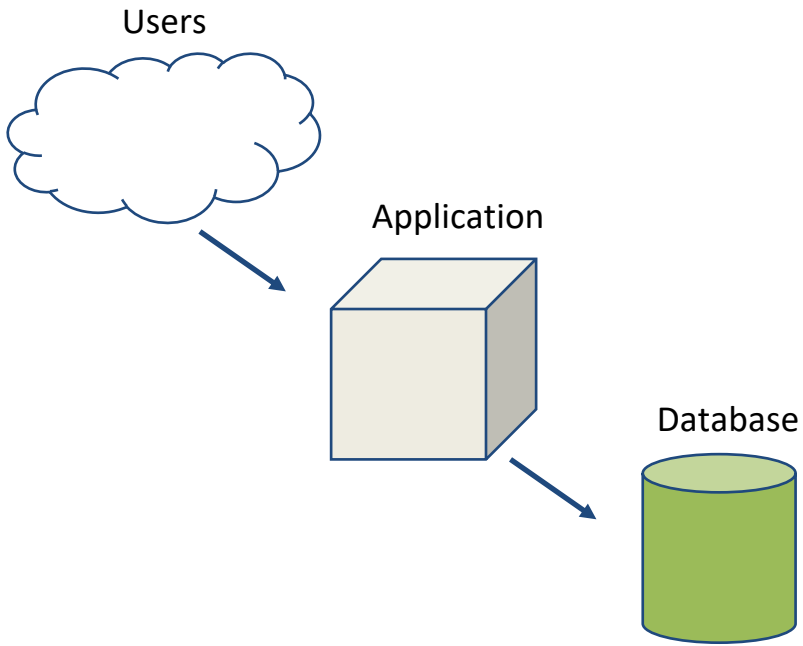
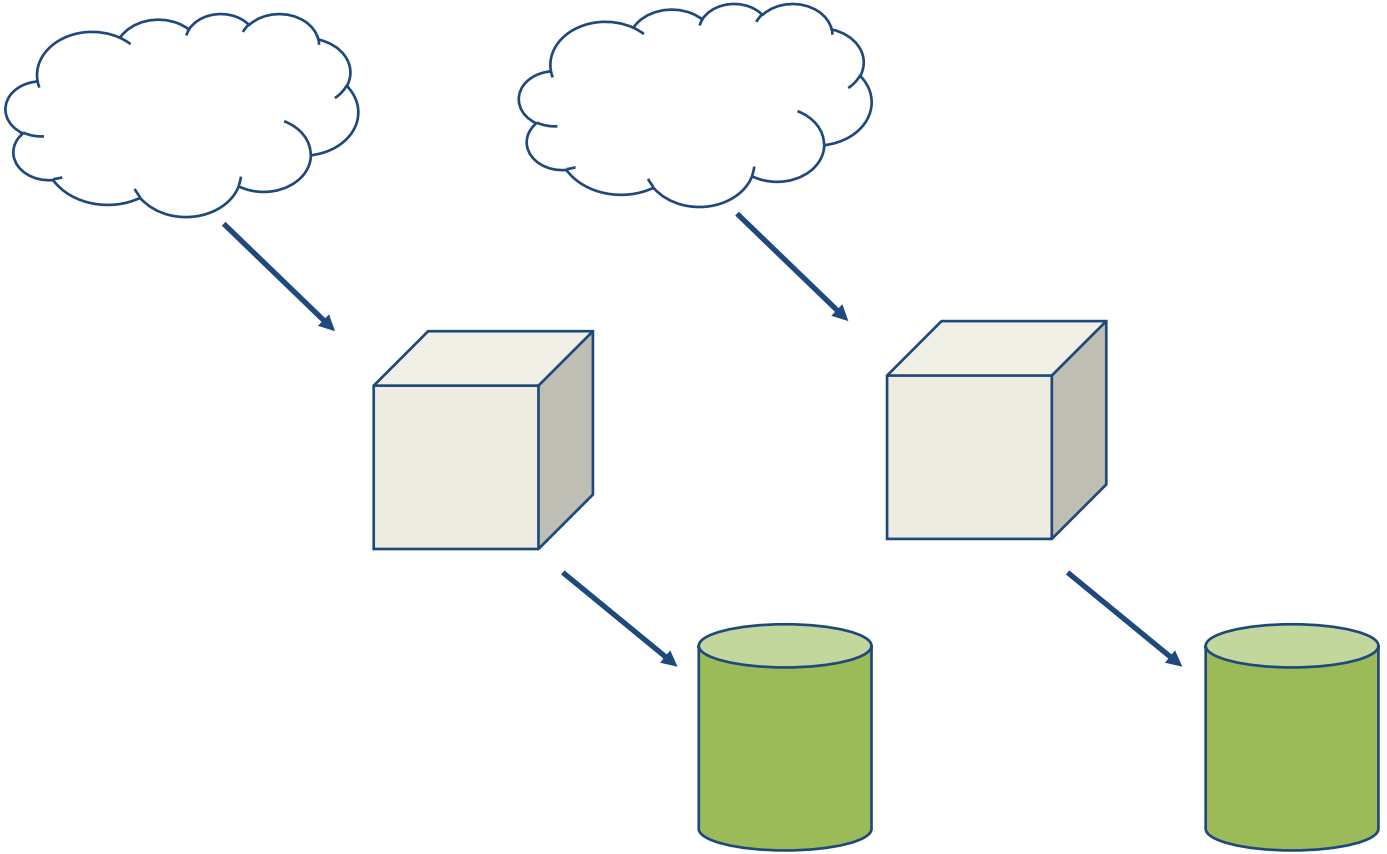
How to sleep well after a major code refactoring

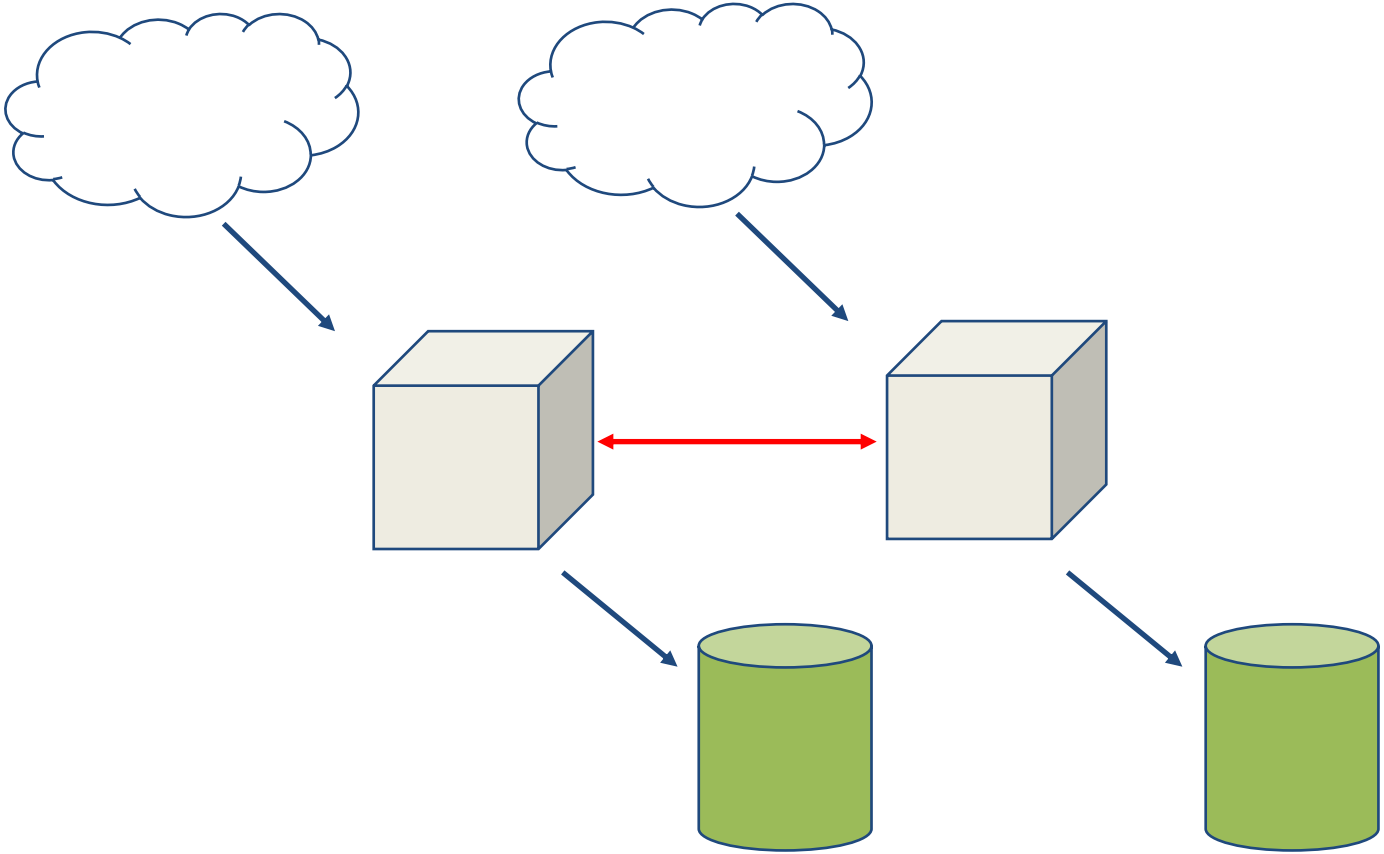
Thomas Arts

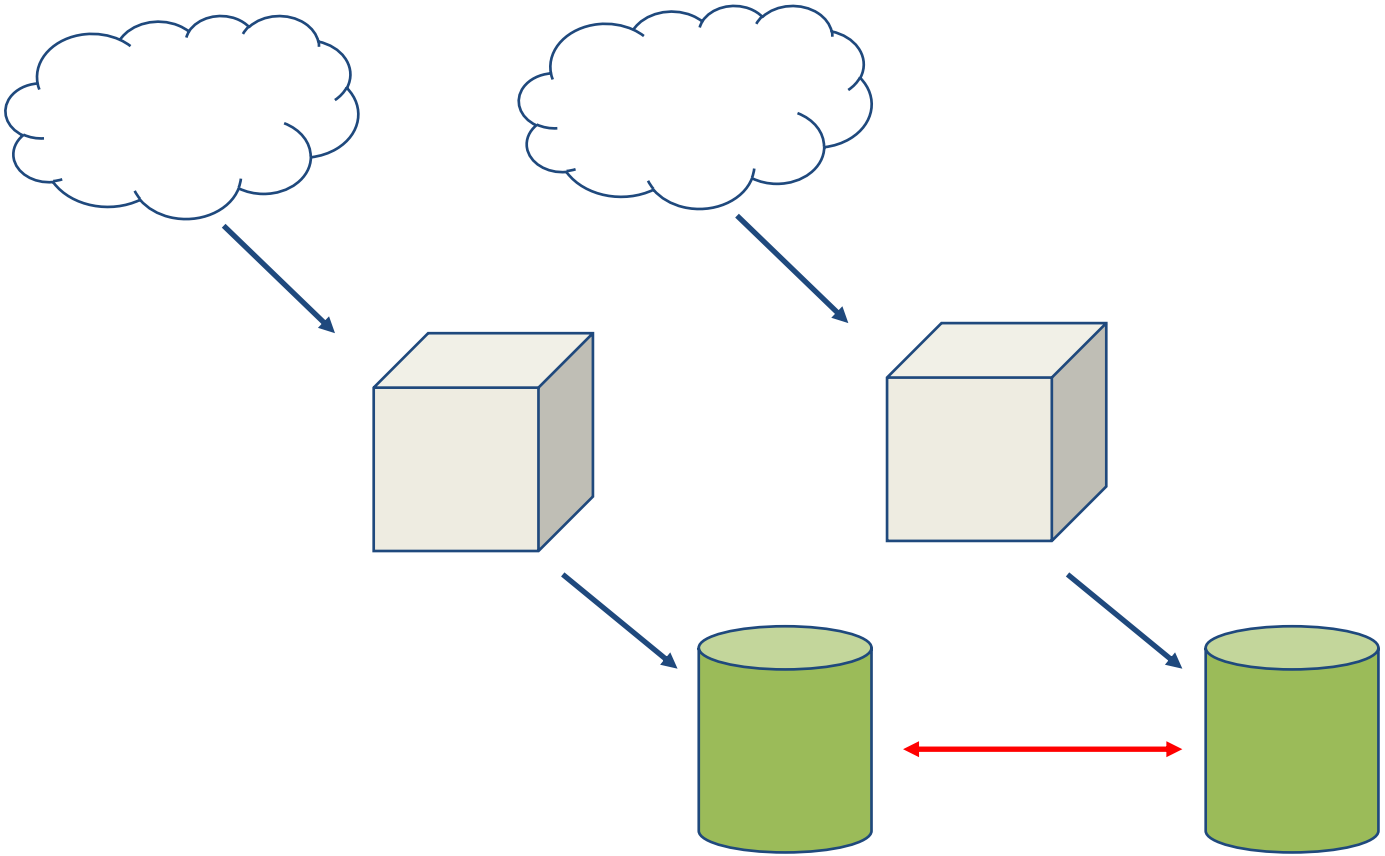
QuviQ

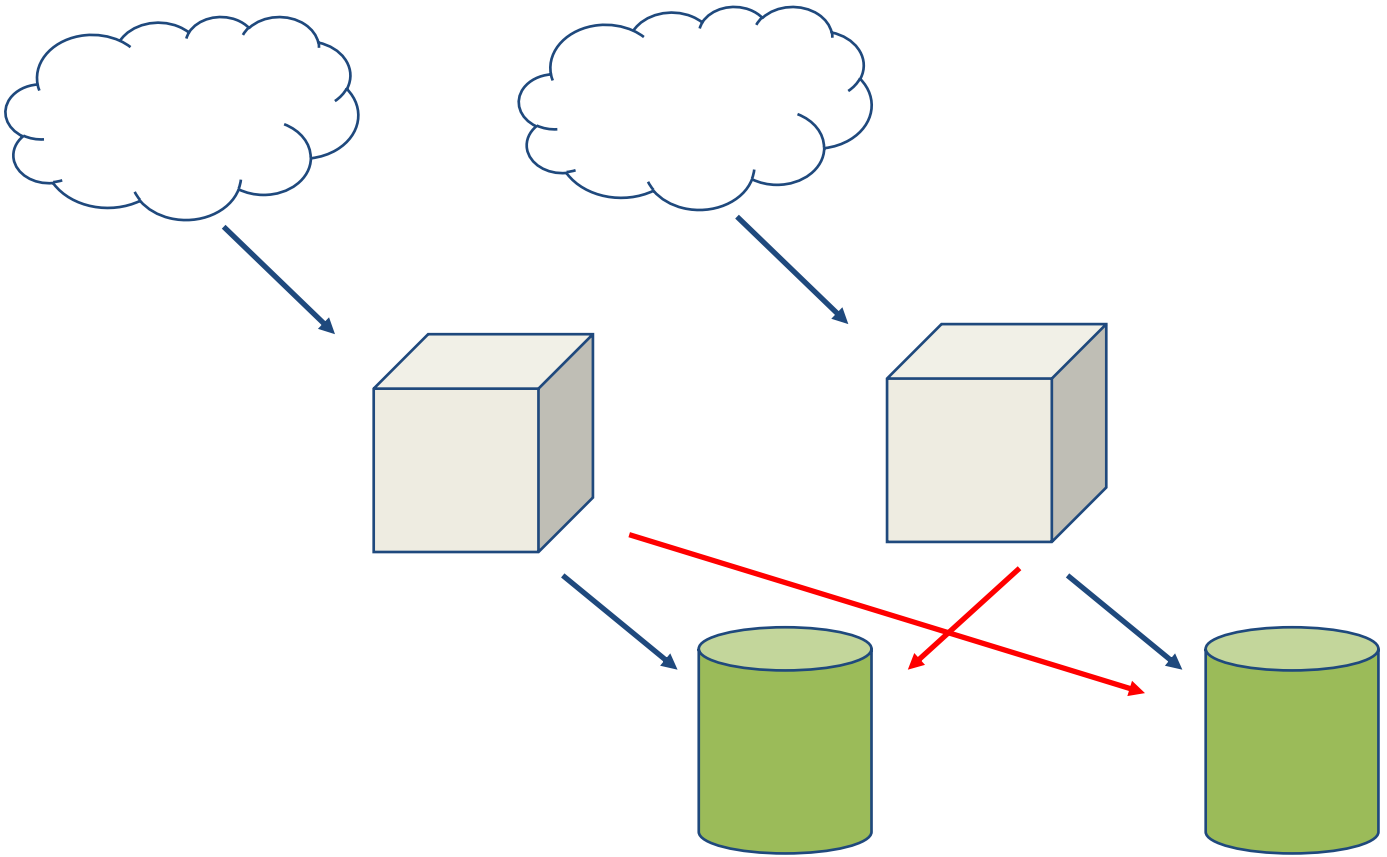
Tobias Lindahl

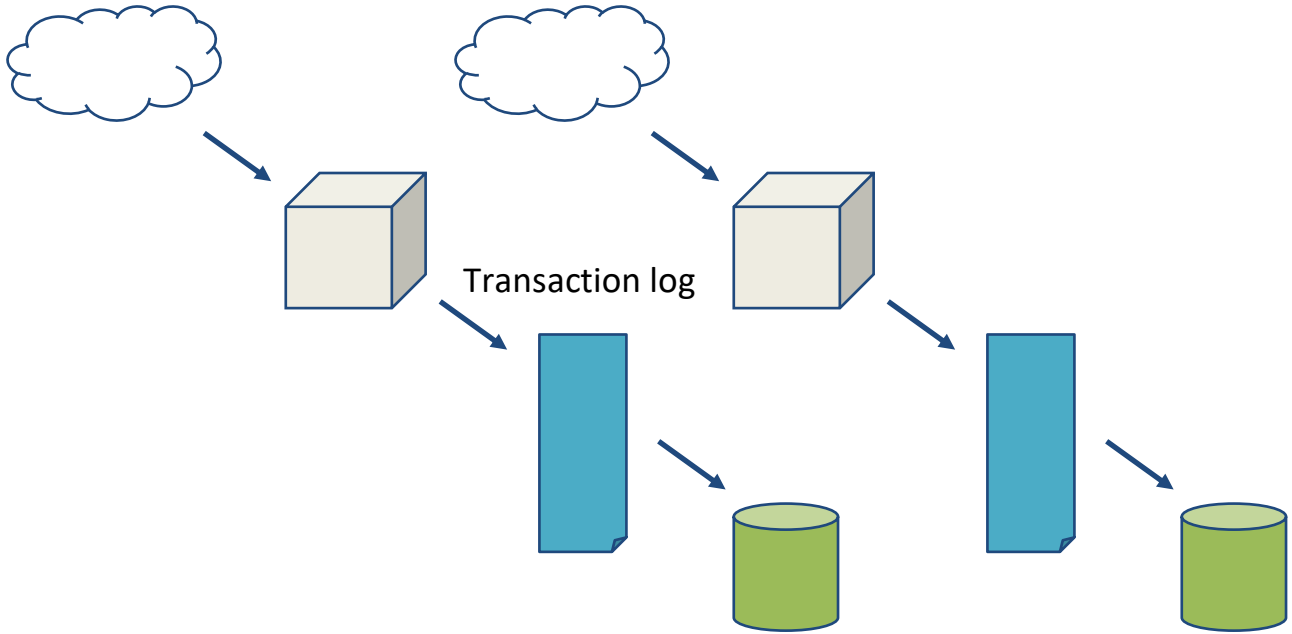

Users
Application
Database

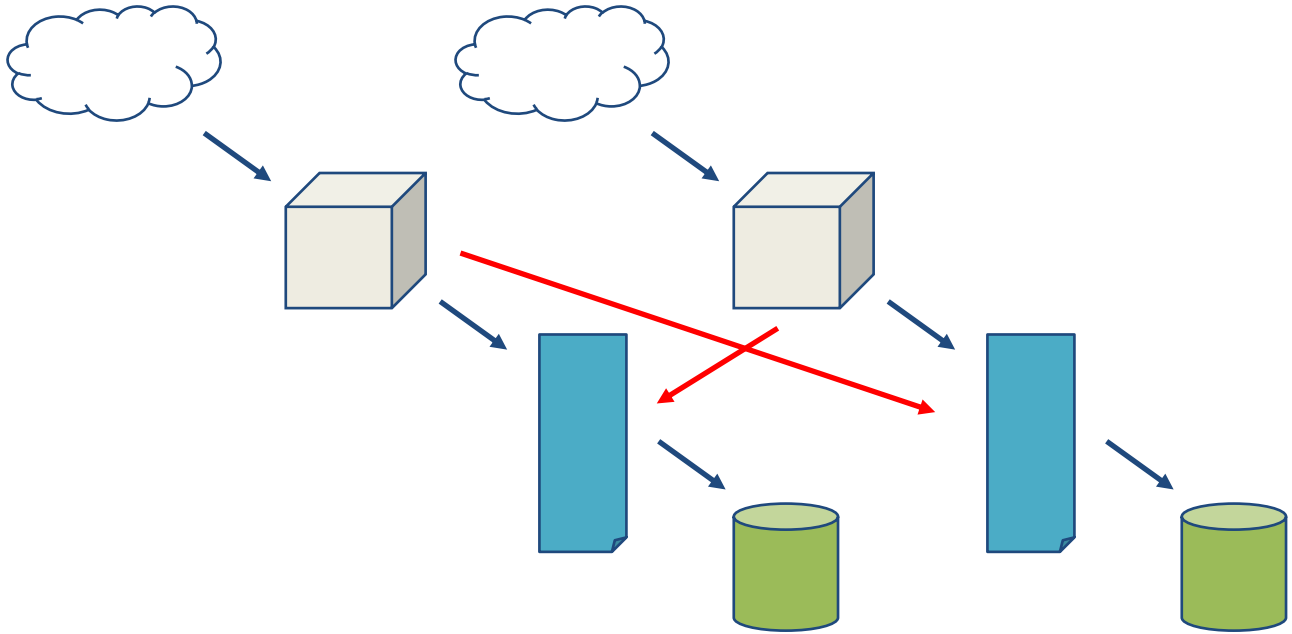


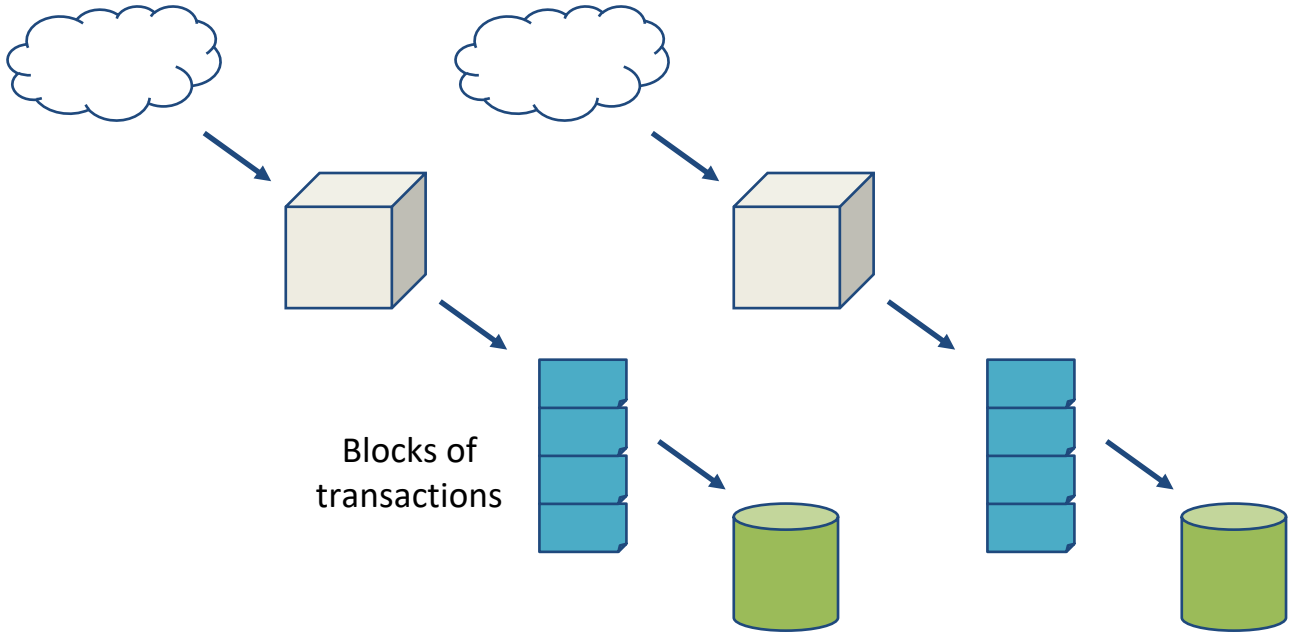


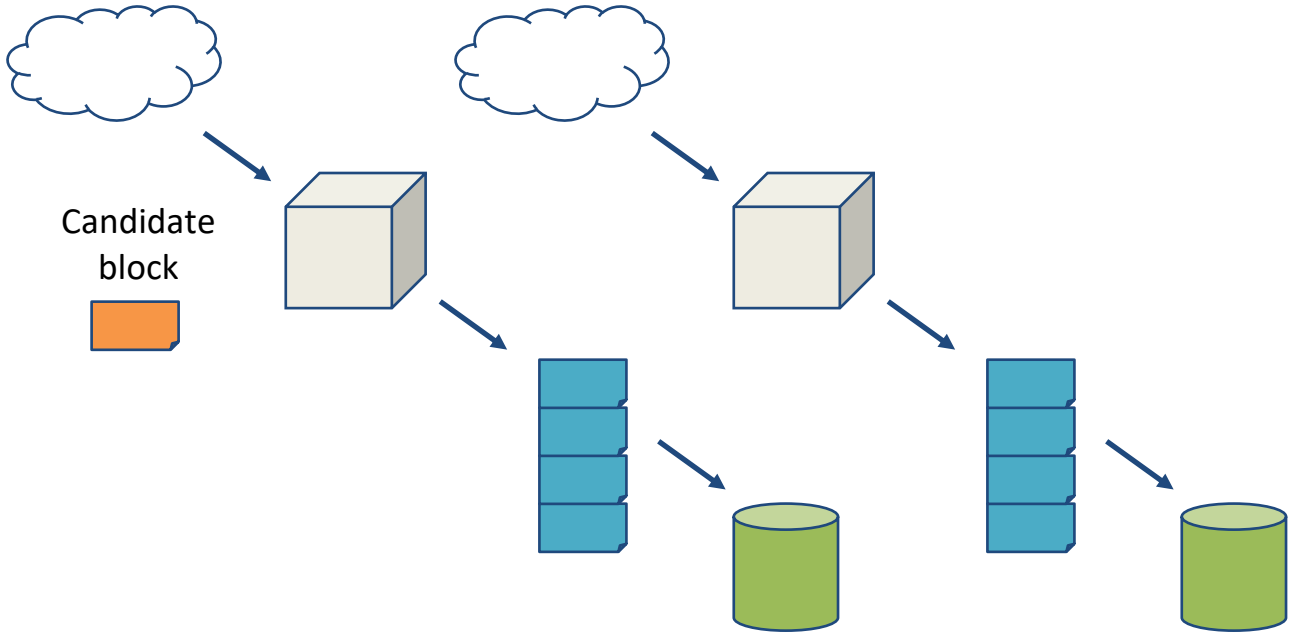


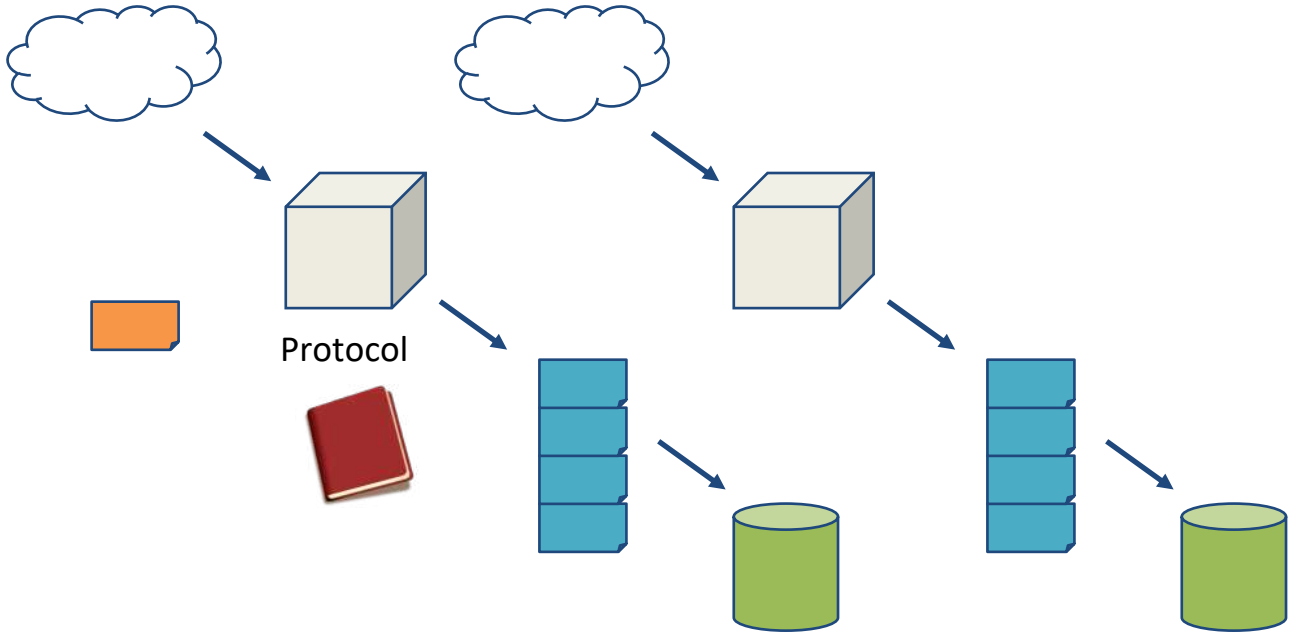


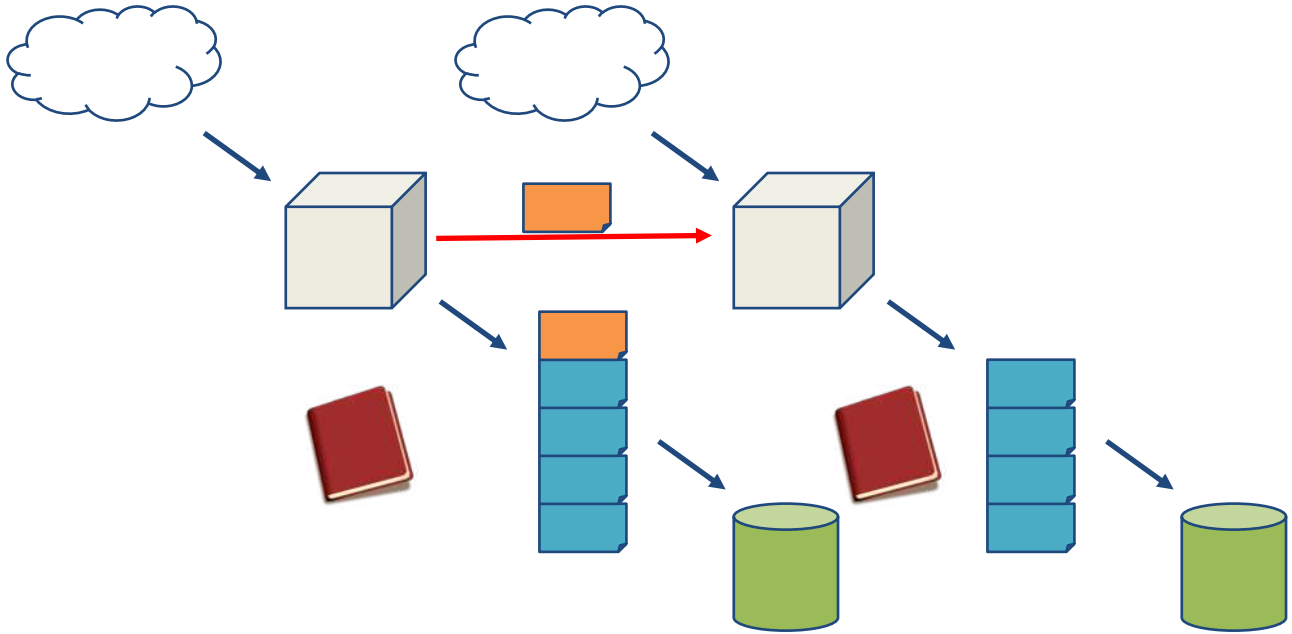

Transaction log

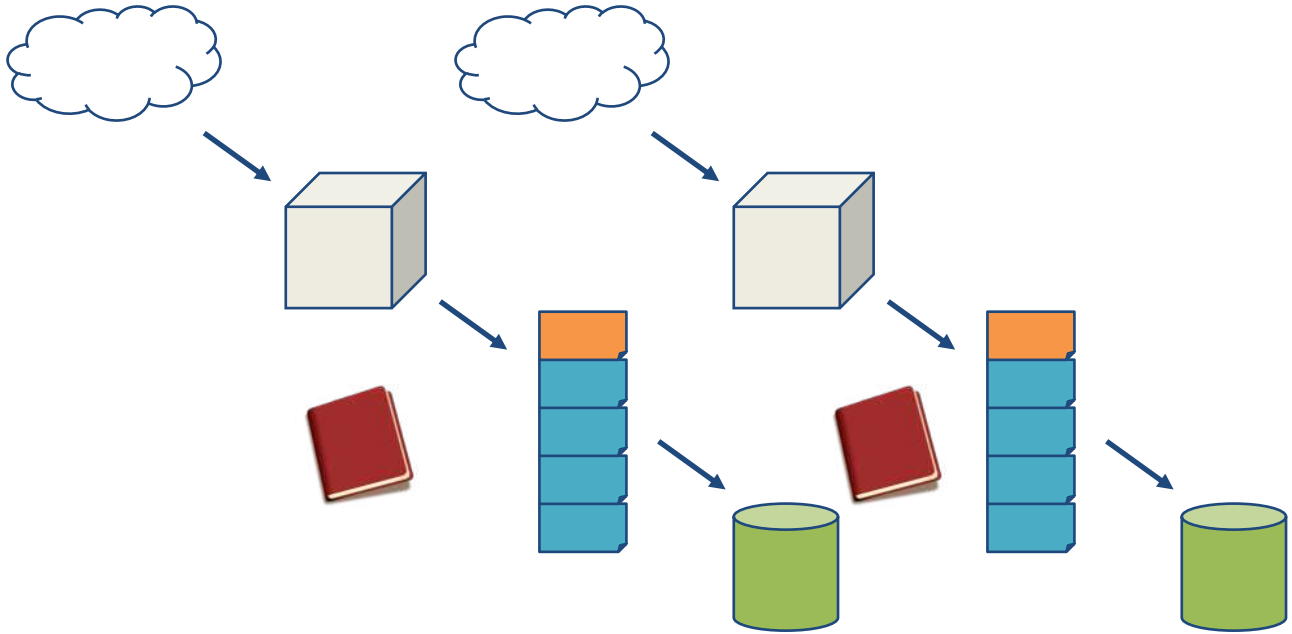


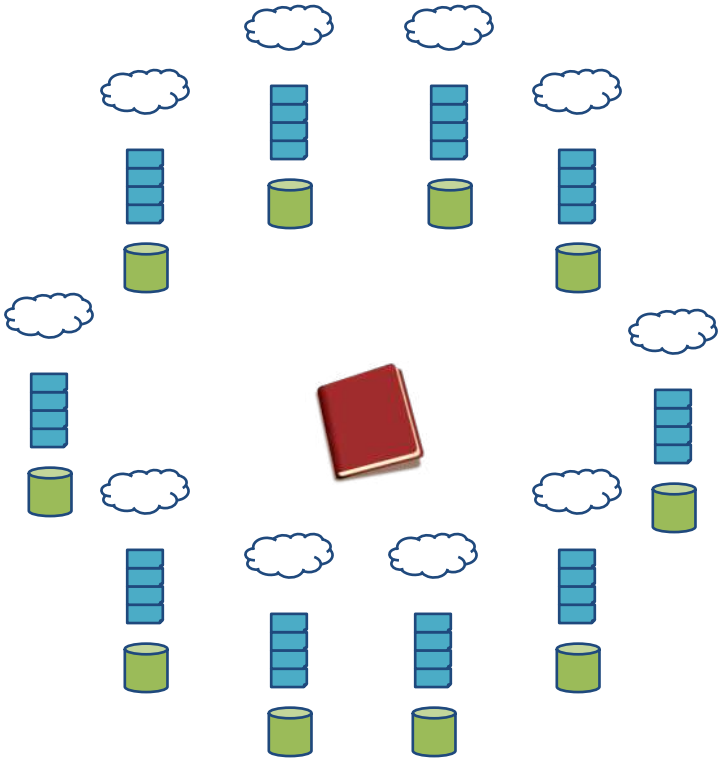

Blocks of
transactions

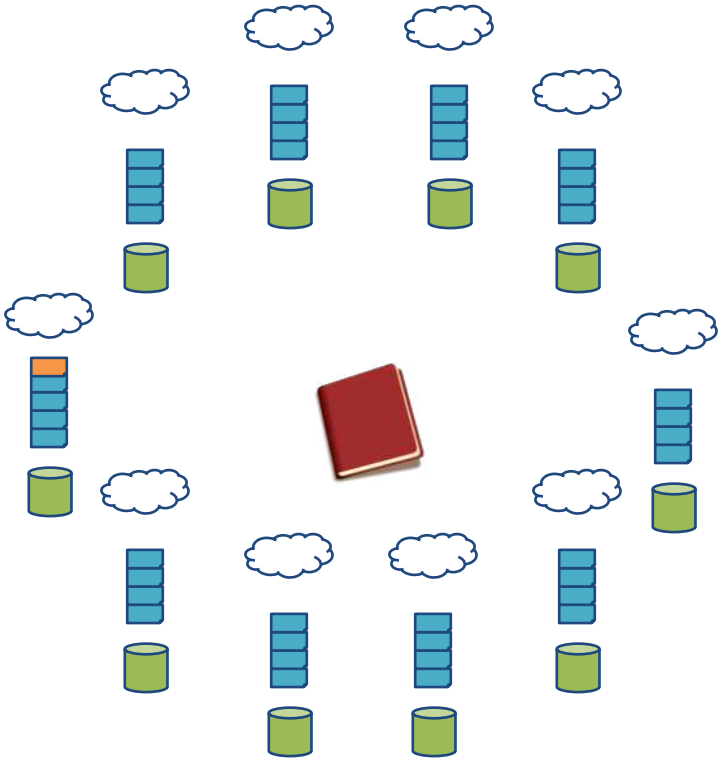

Candidate
block

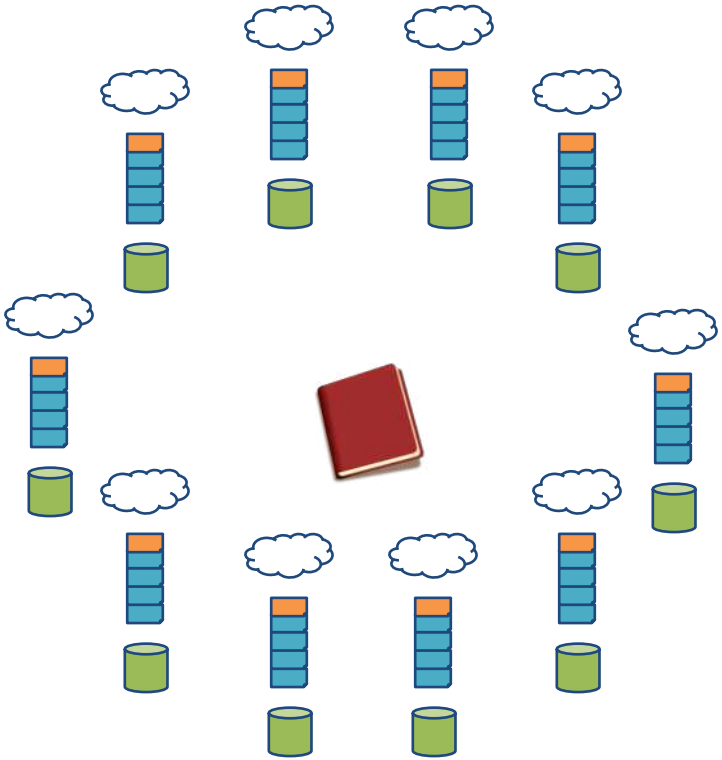

Protocol









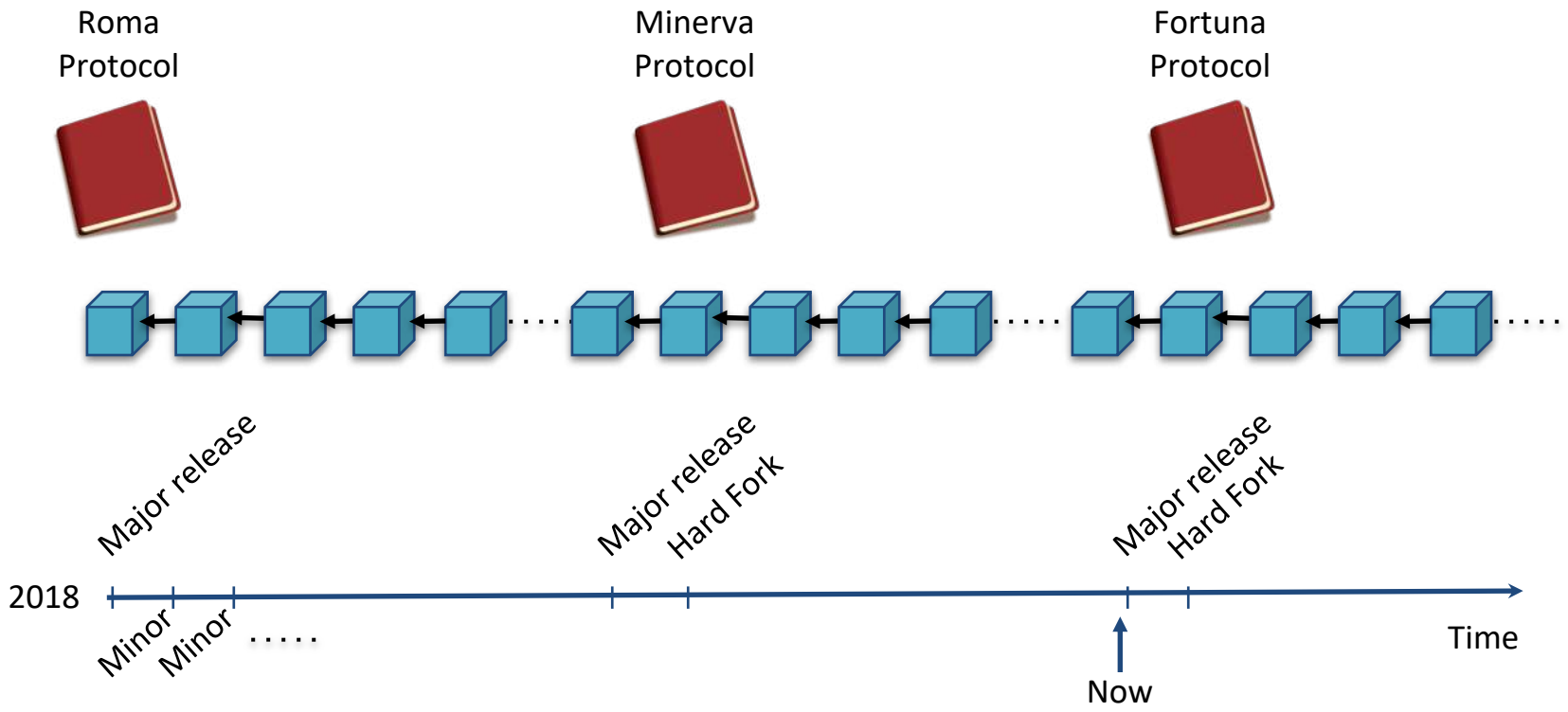


æternity blockchain

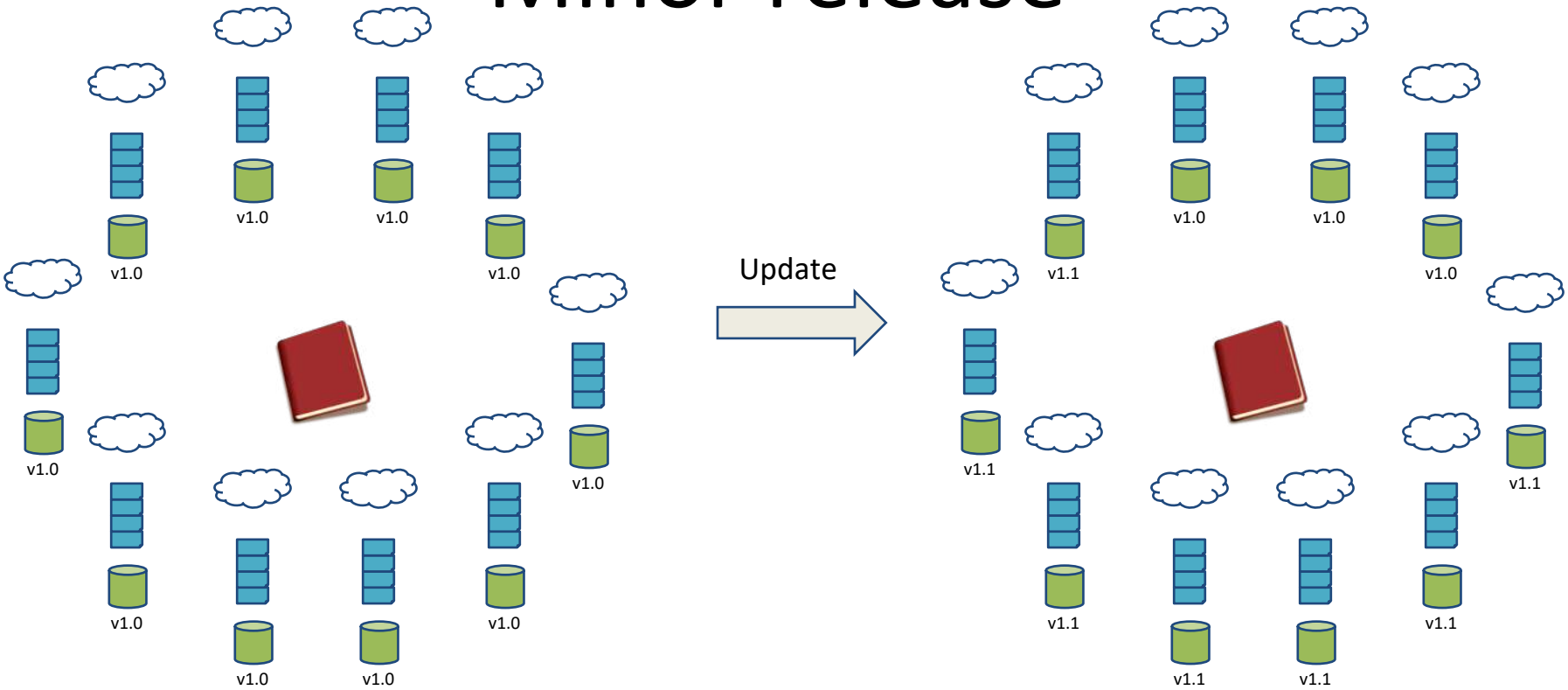
- First class objects
  - Accounts
  - Oracles
  - Name service
  - Tokens
  - Contracts
  - State channels

- Transactions
  - Spend tokens
  - Create/call contracts
  - Register a name
  - Query an oracle
  - ...

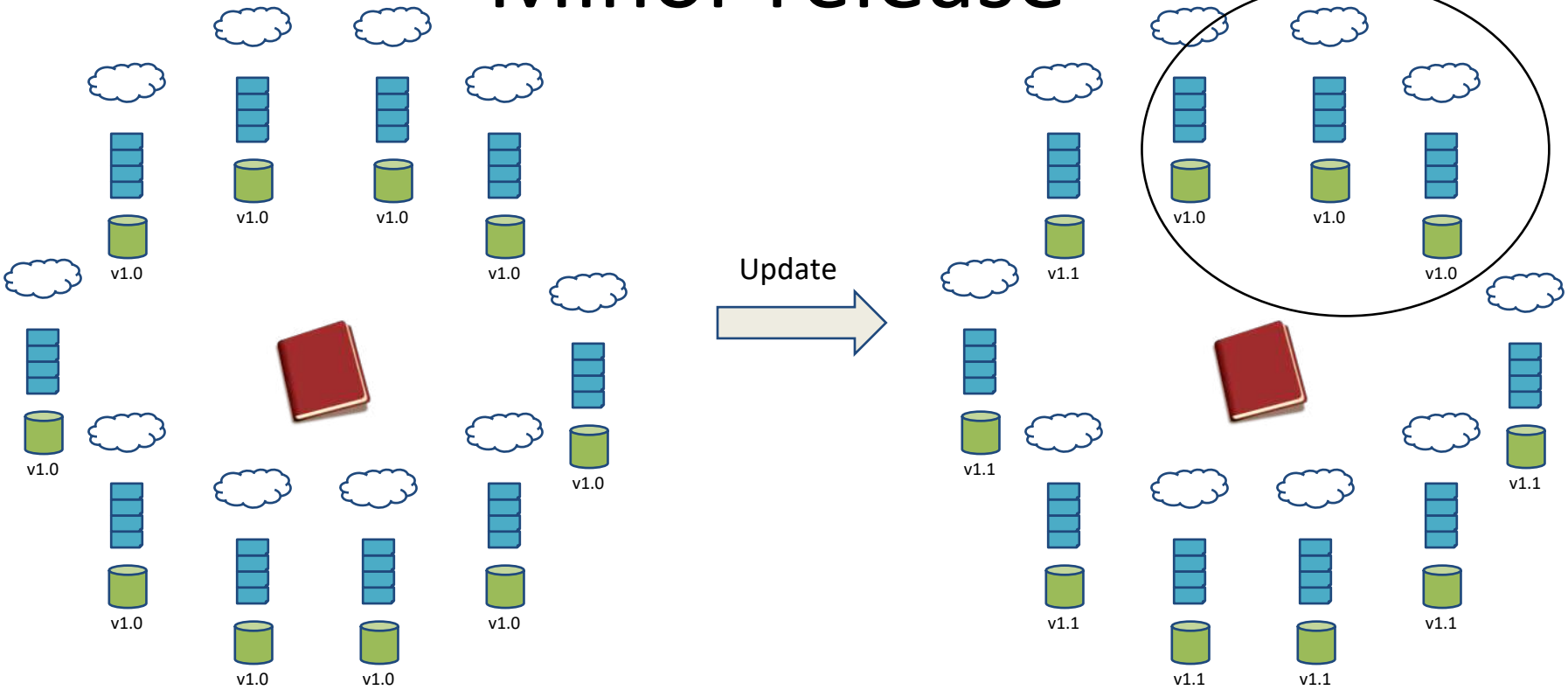
æternity blockchain



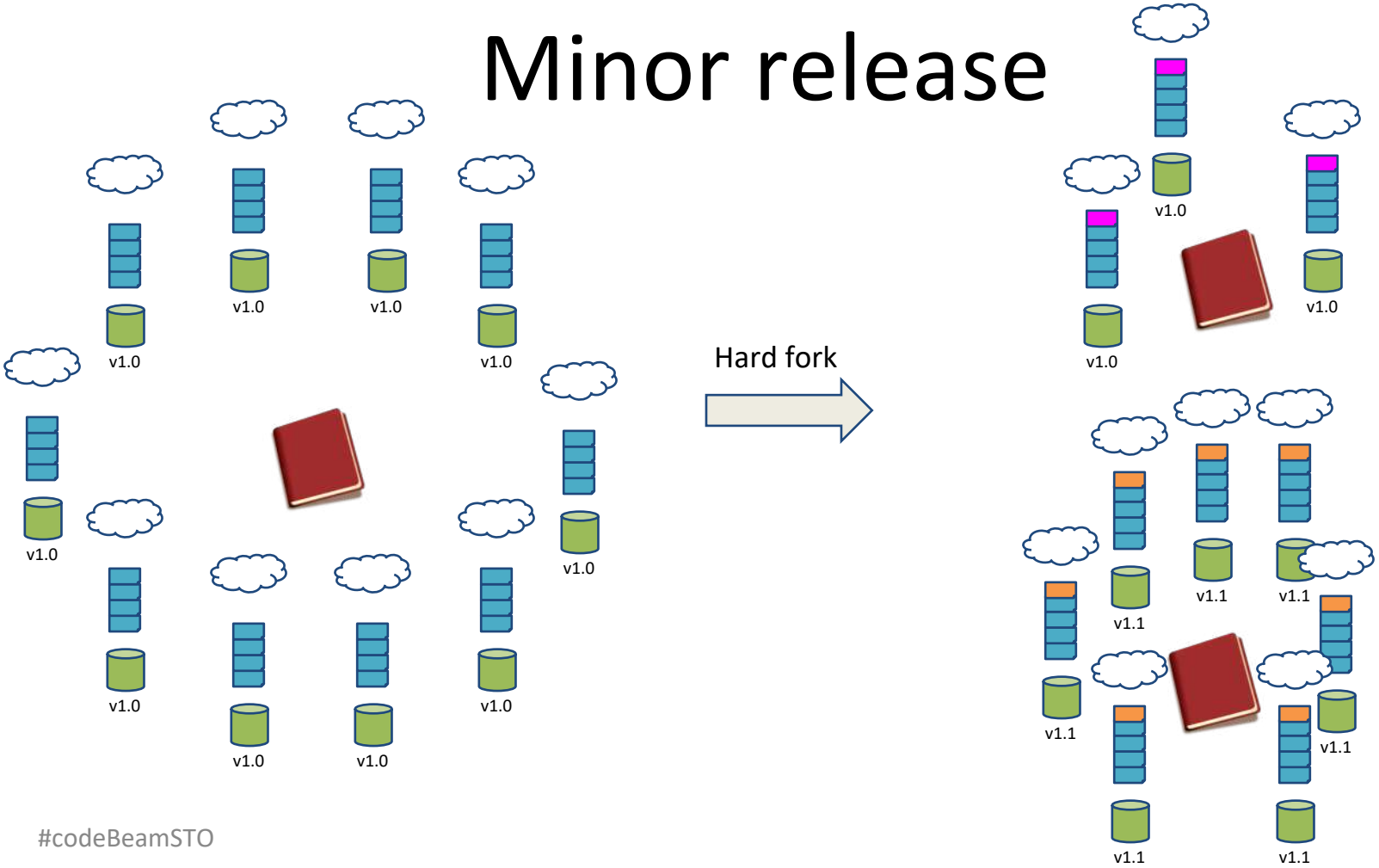
Minor release



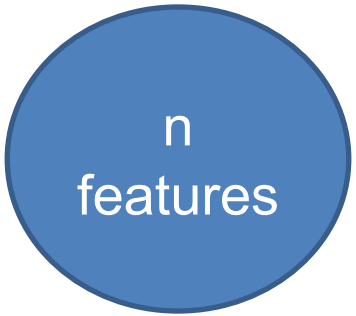
Minor release



Minor release



Why is testing hard?

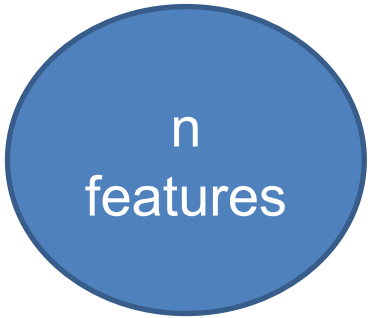


O(n) test cases

3—4 tests per feature

features	tests	additional tests when adding 1 feature
20	80	4

Why is testing hard?

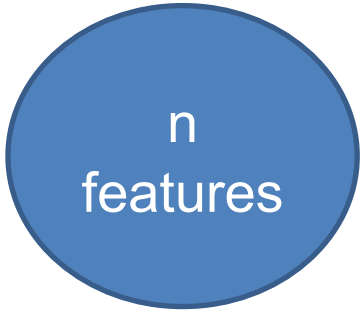


pairs of features

$O(n^2)$ test cases

features	tests	additional tests when adding 1 feature
20	80 + 190	4 + 20

Why is testing hard?



$O(n^3)$ test cases

triples of features

features	tests	additional tests when adding 1 feature
20	80 + 190 + 1140	4 + 20 + 190

Don’t write tests!

Generate them

from properties

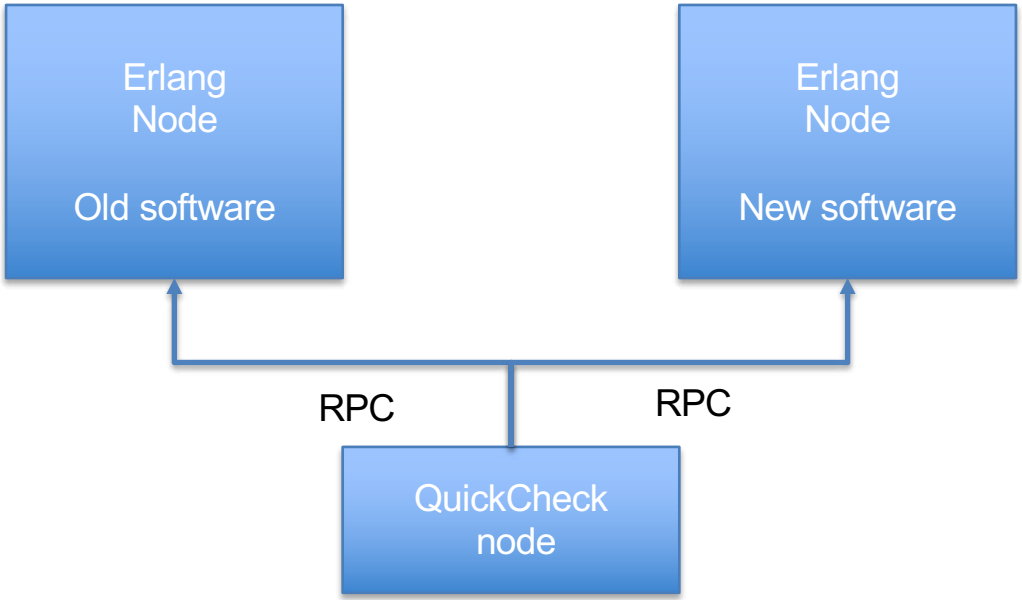
QuickCheck

1999—invented by Koen Claessen and John Hughes, for Haskell

2006—Quviq founded by Thomas Arts and John Hughes, marketing Erlang version
Many extensions

Finding deep bugs for Ericsson, Volvo Cars, Klarna, Basho, NHS, etc…

Generate transactions



Old and new should give same result

Simple example

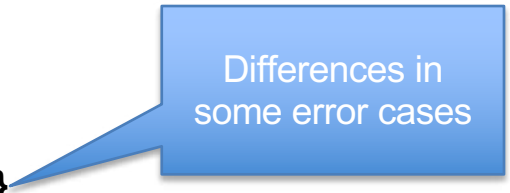
- Two operations:
  - `chain:spend(Tx)` → `ok | {error, Reason}`
  - `chain:balances()` → `#{pubkey() => balance}`

Generate random Txs… most are not even valid

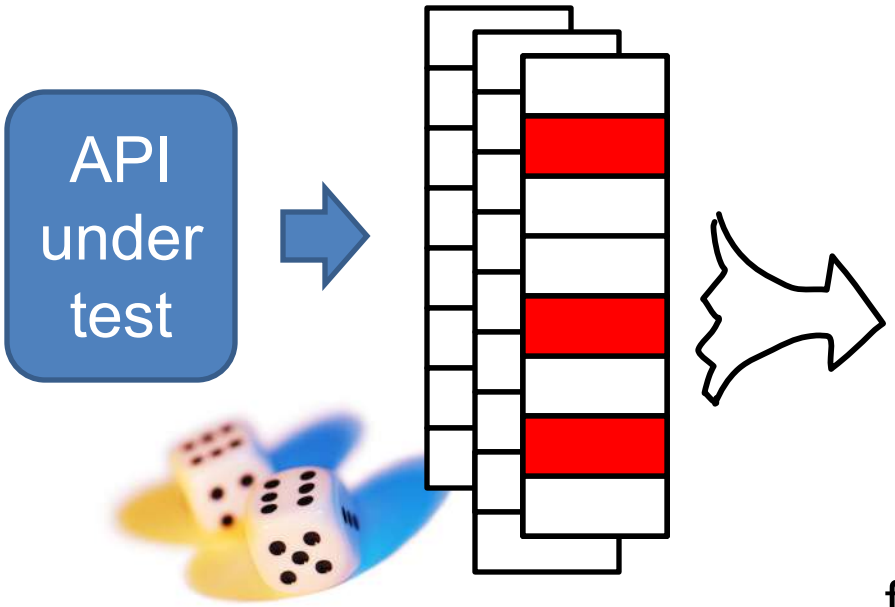
demo1

Tests fail

```
Shrinking xxxxx...x.x.x.(6 times)

transactions_eqc:spend(#{amount => 1, fee => 0, nonce => 1,
      receiver
        => <<143, 38, 179, 196, 165, 202, 141, 213, 47, 66, 61, 71, 146,
             39, 161, 41, 5, 1, 13, 223, 57, 9, 149, 94, 223, 197, 7, 46, 30,
             232, 255, 39>>,
      sender
        => <<22, 102, 237, 224, 220, 101, 96, 64, 230, 32, 76, 247, 126,
             241, 187, 151, 183, 188, 42, 207, 30, 209, 61, 224, 155, 229,
             187, 48, 229, 61, 21, 228>>}) ->
  {{error, invalid_tx}, {error, fee_too_low}}

Reason:
  Post-condition failed:
  {error, invalid_tx} /= {error, fee_too_low}
result: failed
false
```



QuickCheck



A minimal
failing example

Property


```
prop_transactions() ->
  ?FORALL(Cmds, commands(?MODULE),
      begin
        start_nodes(),

        {H, S, Res} = run_commands(Cmds),
        BalanceDiffs =
            differences(rpc(oldnode, chain, balances, []),
                        rpc(newnode, chain, balances, [])),

        stop_nodes(),

        conjunction([{result, Res == ok},
                     {balances, equals(BalanceDiffs, [])}
                    ])
      end).
```

Generate commands

```
%% --- Operation: spend ---
spend_args(_S) ->
     [#{sender   => pubkey(32),
        receiver => pubkey(32),
        amount   => int(),
        fee.     => int(),
        nonce    => int()}].

spend_call(_S, [Tx]) ->
   {rpc(oldnode, chain, spend, [Tx]),
    rpc(newnode, chain, spend, [Tx])}.

spend_post(_S, Args, {Old, New}) ->
     eq(Old, New).
```

Errors are compared

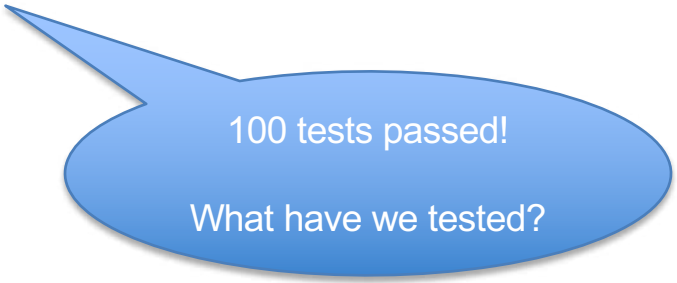
Generate commands

```
%% --- Operation: spend ---
spend_args(_S) ->
     [#{sender   => pubkey(32),
        receiver => pubkey(32),
        amount   => int(),
        fee.     => int(),
        nonce    => int()}].

spend_call(_S, [Tx]) ->
   {rpc(oldnode, chain, spend, [Tx]),
    rpc(newnode, chain, spend, [Tx])}.

spend_post(_S, _Args, {{error, _}, {error, _}}) ->
     true;
spend_post(_S, _Args, {Old, New}) ->
     eq(Old, New).
```

Don't care which error

Relax error comparison

```
1> eqc:quickcheck(transactions_eqc:prop_transactions()).
.....................................................................................................
OK, passed 100 tests
```



Generate commands

```
spend_args(_S) ->
     [#{sender   => pubkey(32),
        receiver => pubkey(32),
        amount   => int(),
        fee.     => int(),
        nonce    => int()}].

spend_call(_S, [Tx]) ->
   {rpc(oldnode, chain, spend, [Tx]),
    rpc(newnode, chain, spend, [Tx])}.

spend_post(_S, _Args, {{error, _}, {error, _}}) ->
     true;
spend_post(_S, _Args, {Old, New}) ->
     eq(Old, New).
```

What have we tested?

Generate random Txs… most are not even valid
Record what we test:

```
spend_features(_S, _Args, {Old, New}) -> [Old].
```

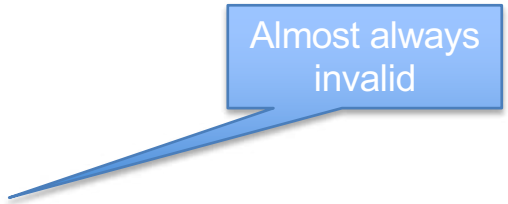
.....(x10).....
.....(x100).....
.....
OK, passed 10000 tests

88.629% {error, invalid_tx}
 6.740% {error, sender_account_unknown}
 4.631% {error, fee_too_low}

length: Count: 10000 Min: 0 Max: 158 Avg: 15.66370 StdDev: 16.78488 Total: 156637
true

Generate commands

```
spend_args(_S) ->
     [#{sender   => pubkey(32),
        receiver => pubkey(32),
        amount   => int(),
        fee.     => int(),
        nonce    => int()}].

spend_call(S, [Tx]) ->
   {rpc(oldnode, chain, spend, [Tx]),
    rpc(newnode, chain, spend, [Tx])}.
```



Generate more valid Txs

```
spend_args(_S) ->
  [#{sender   => pubkey(32),
     receiver => pubkey(32),
     amount   => frequency([{99, choose(10, 1000)}, {1, int()}]),
     fee      => frequency([{50, choose(10, 100)}, {1, int()}]),
     nonce    => int()
    }].

spend_call(S, [Tx]) ->
   {rpc(oldnode, chain, spend, [Tx]),
    rpc(newnode, chain, spend, [Tx])}.
```

Better distribution

- Before

88.629% {error, invalid_tx}
6.740% {error, sender_account_unknown}
4.631% {error, fee_too_low}

- After

96.0% {error, sender_account_unknown}
3.5% {error, invalid_tx}
0.5% {error, fee_too_low}

Generate valid sender accounts

```
initial_state() ->
   #{pubkeys =>
        [<<227,81,22,143,182,219,118,194,214,164,169,153,6,190,90,72,72,
           11,74,195,160,239,10,12,98,144,86,254,51,110,216,22>>]}.


spend_args(S) ->
  [#{sender   => frequency([{49, account_gen(existing, S)},
                            { 1, account_gen(new, S)}]),
     receiver => frequency([{1, account_gen(existing, S)},
                            {1, account_gen(new, S)}]),
     amount   => frequency([{99, choose(10, 1000)}, {1, int()}]),
     fee      => frequency([{50, choose(10, 100)}, {1, int()}]),
     nonce    => int()
    }].
```

Generate valid sender accounts

```
initial_state() ->
   #{pubkeys =>
        [<<227,81,22,143,182,219,118,194,214,164,169,153,6,190,90,72,72,
           11,74,195,160,239,10,12,98,144,86,254,51,110,216,22>>]}.


account_gen(existing, #{pubkeys := Keys}) ->
     elements(Keys);
account_gen(new, #{pubkeys := Keys}) ->
     ?SUCHTHAT(Key, pubkey(32), not lists:member(Key, Keys)).
```

Generate valid sender accounts

```
spend_args(S) ->
  [#{sender   => frequency([{49, account_gen(existing, S)},
                            { 1, account_gen(new, S)}]),
     receiver => frequency([{1, account_gen(existing, S)},
                            {1, account_gen(new, S)}]),
     amount   => frequency([{99, choose(10, 1000)}, {1, int()}]),
     fee      => frequency([{50, choose(10, 100)}, {1, int()}]),
     nonce    => int()
    }].


spend_next(#{pubkeys := Keys} = S, _, [Tx]) ->
     S#{pubkeys => Keys ++ [maps:get(receiver, Tx)]}.
```

Running tests

```
Shrinking x...x.(4 times)


transactions_eqc:spend(#{amount => 10, fee => 10, nonce => 1,
      receiver
        => <<227, 81, 22, 143, 182, 219, 118, 194, 214, 164, 169, 153, 6,
             190, 90, 72, 72, 11, 74, 195, 160, 239, 10, 12, 98, 144, 86, 254,
             51, 110, 216, 22>>,
      sender
        => <<227, 81, 22, 143, 182, 219, 118, 194, 214, 164, 169, 153, 6,
             190, 90, 72, 72, 11, 74, 195, 160, 239, 10, 12, 98, 144, 86, 254,
             51, 110, 216, 22>>}) ->
  {ok, ok}
balances: failed
   #{<<227, 81, 22, 143, 182, 219, 118, 194, 214, 164, 169, 153, 6,
       190, 90, 72, 72, 11, 74, 195, 160, 239, 10, 12, 98, 144, 86, 254,
       51, 110, 216, 22>>
       => {wrong_balance, 199999980, 199999990}}
/=
   #{}
false
```

Distribution still poor

```
..................................................................
.....................................
OK, passed 100 tests


55.5% {error,invalid_tx}
24.3% {error,sender_account_unknown}
16.9% {error,nonce_too_high}
 1.9% ok
 0.8% {error,nonce_too_low}
 0.5% {error,fee_too_low}
 0.2% {error,not_enough_funds}


length:    Count: 100   Min: 0   Max: 70   Avg: 12.270   StdDev: 14.099   Total: 1227
accounts:  Count: 100   Min: 1   Max: 71   Avg: 13.270   StdDev: 14.099   Total: 1327
```

We need valid nonces

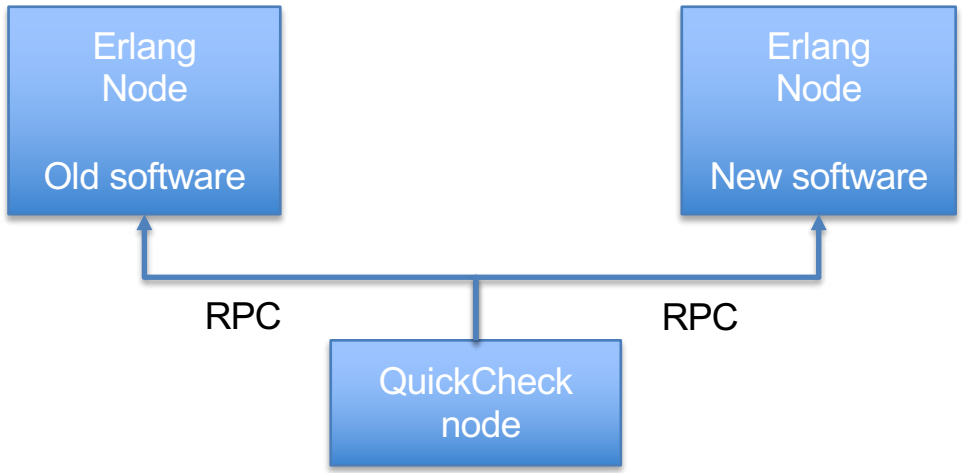
We need more of those

Each spend adds account

Sleep well

We must be 100% compatible with old implementation of same protocol

QuickCheck can provide this trust (more tests, weird tests)



Questions

- https://aeternity.com/
- https://github.com/aeternity/aeternity/
- https://github.com/Quviq/epoch-eqc