



ZILLIQA: A TALE OF SHARDING

Amrit Kumar



@ZILLIQA



ZILLIQA.COM

THROUGHPUT OF BLOCKCHAINS



BITCOIN
7 TX/S



THROUGHPUT OF PAYMENT SYSTEMS



VISA
8000 TX/S



CONGESTION & HIGH GAS



~US\$50 Gas Fee
With CryptoKitties



@ZILLIQA



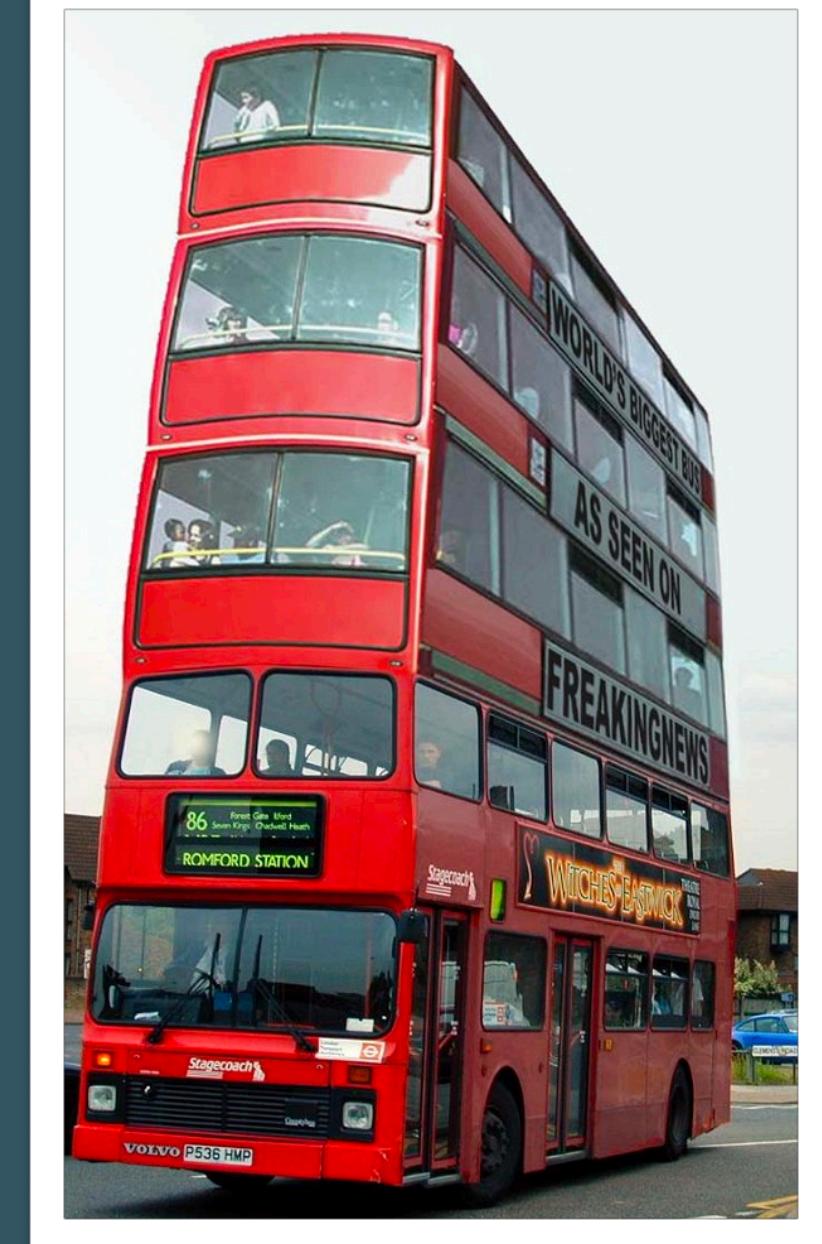
ZILLIQA.COM



TOWARDS GREATER SCALABILITY

LARGER BLOCKS

Not a 100x
Scaling Factor



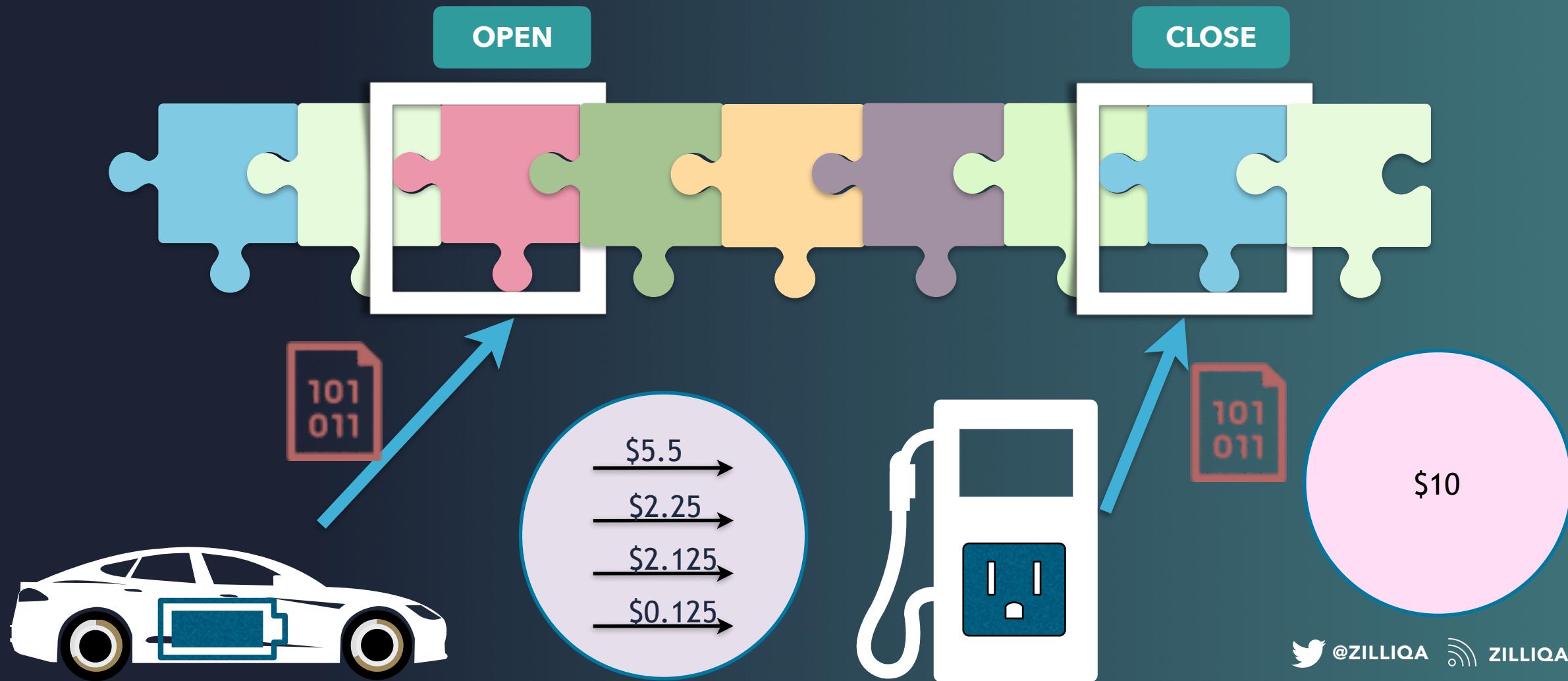
DELEGATED CONSENSUS



DECENTRALIZATION
vs
PERFORMANCE

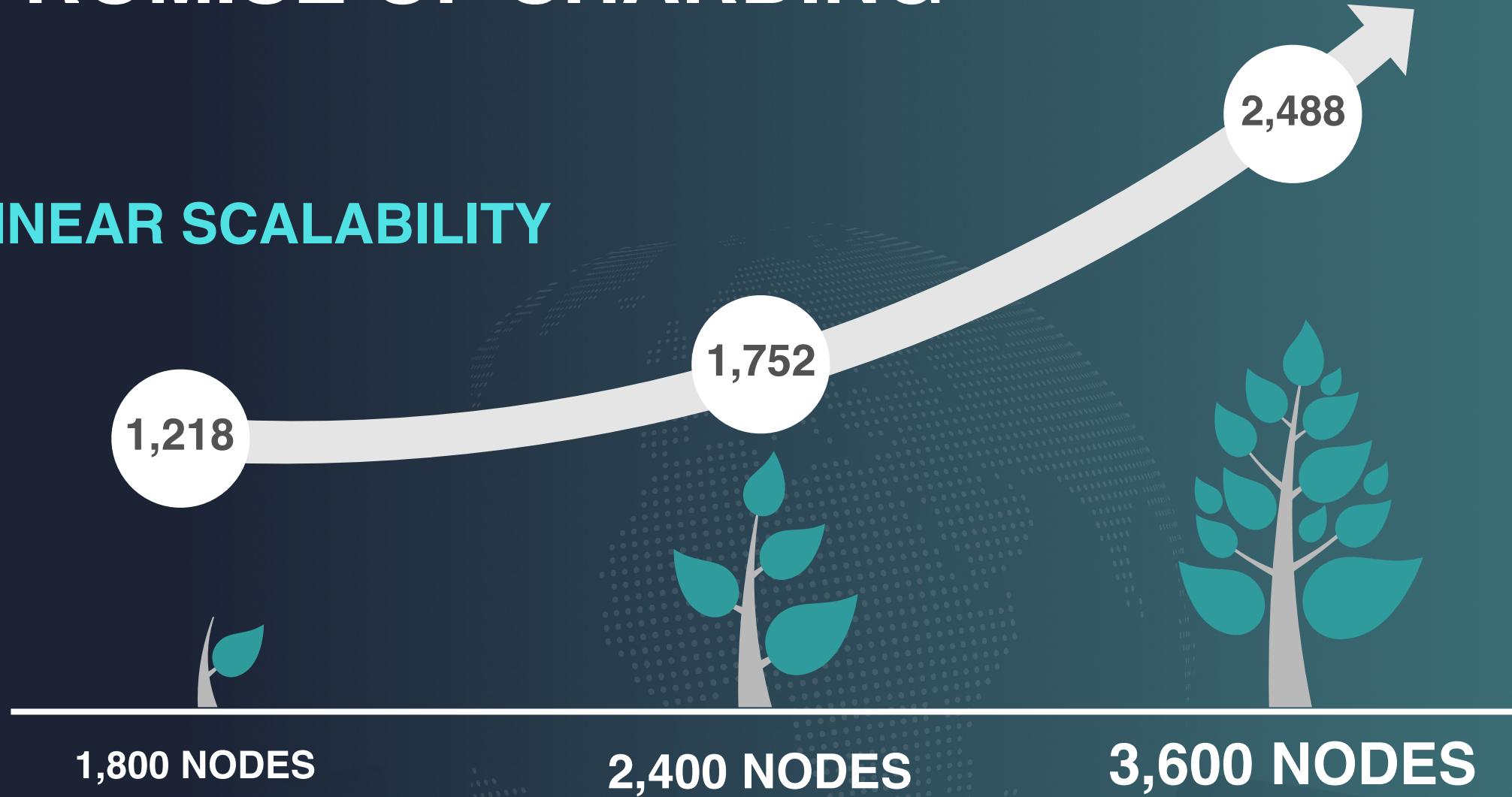
OFFCHAIN SOLUTIONS

Complementary to
onchain scalability

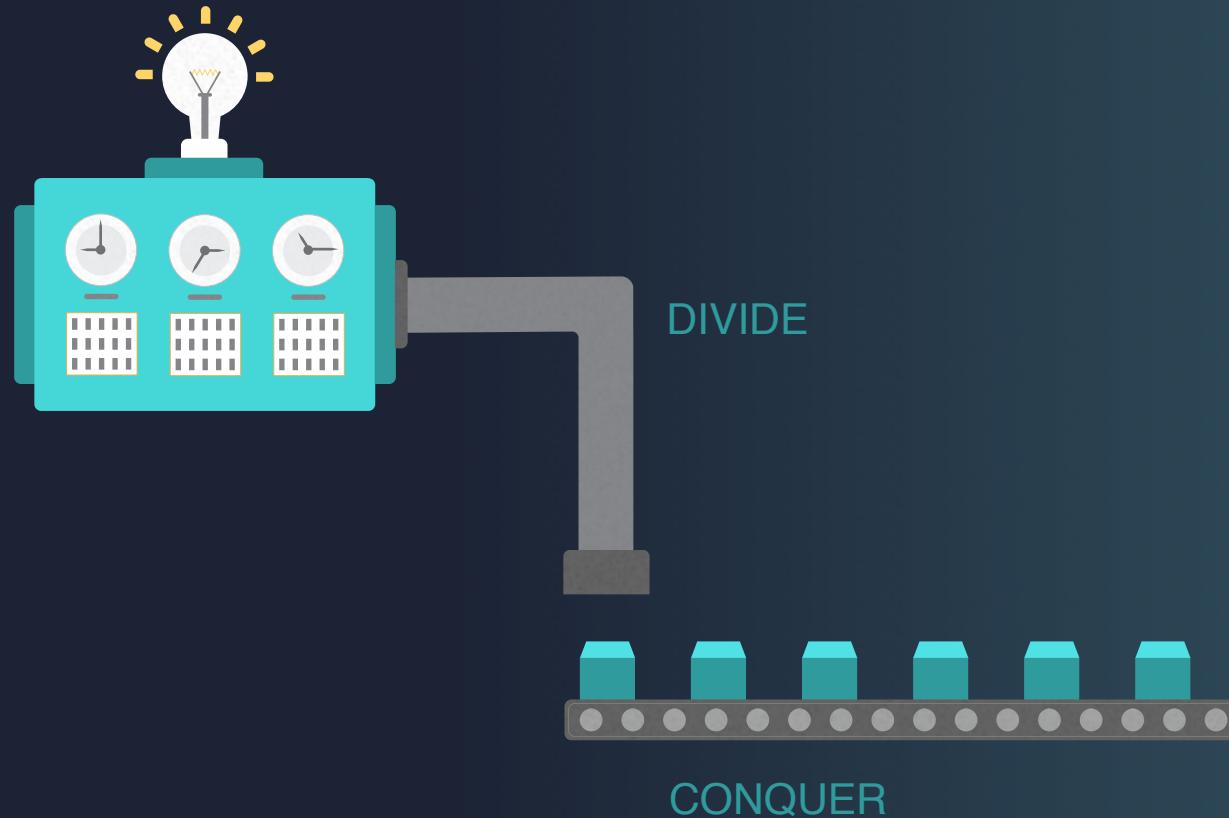


PROMISE OF SHARDING

LINEAR SCALABILITY



CHALLENGES OF SHARDING



Uncompromised *security*

Strong *decentralization*

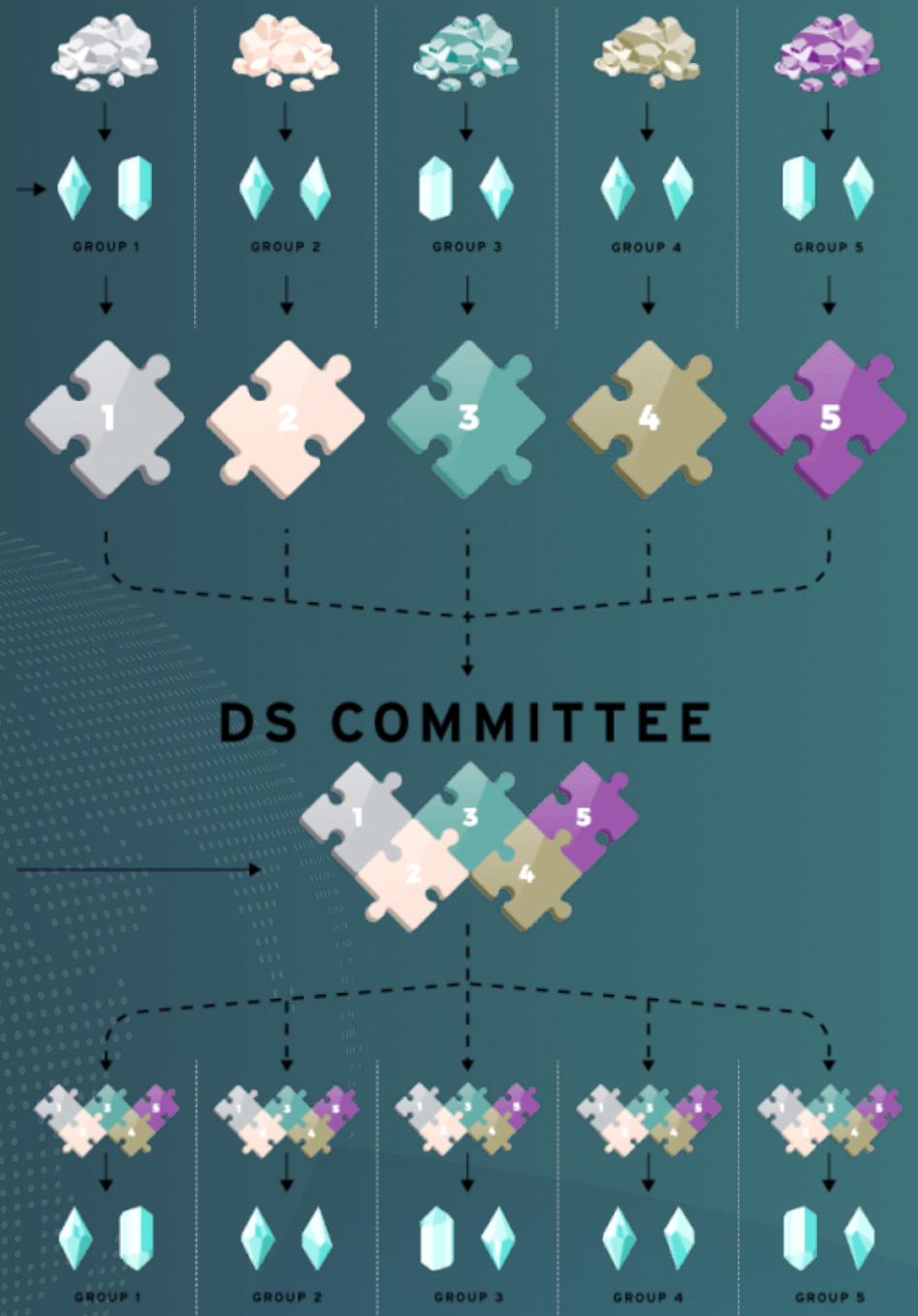
Efficient *contract* execution

NETWORK SHARDING

Sybil defense
Preventing flooding of malicious nodes

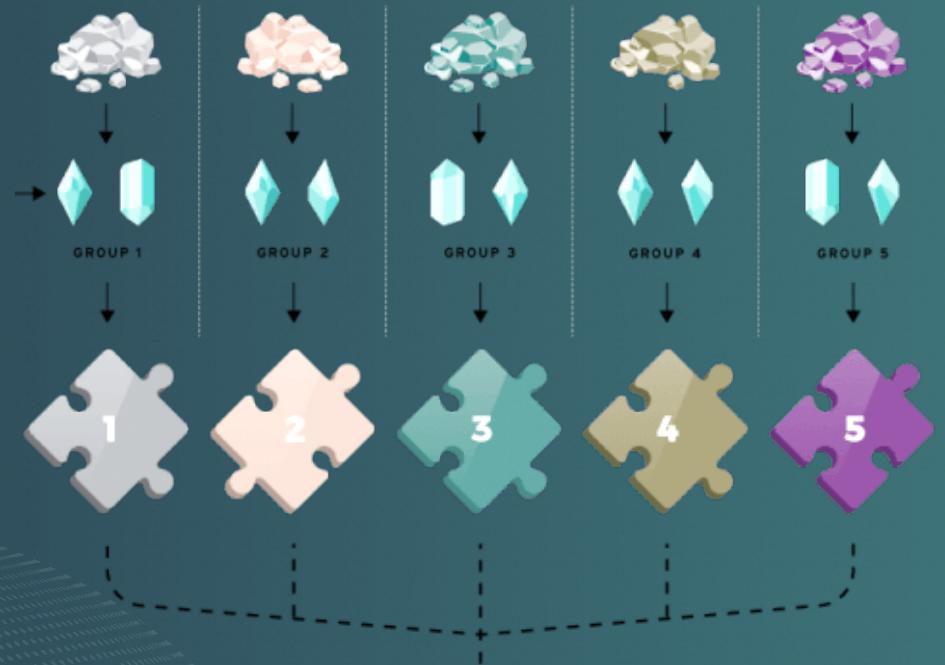


600-node consensus
Secure & decentralized decision making

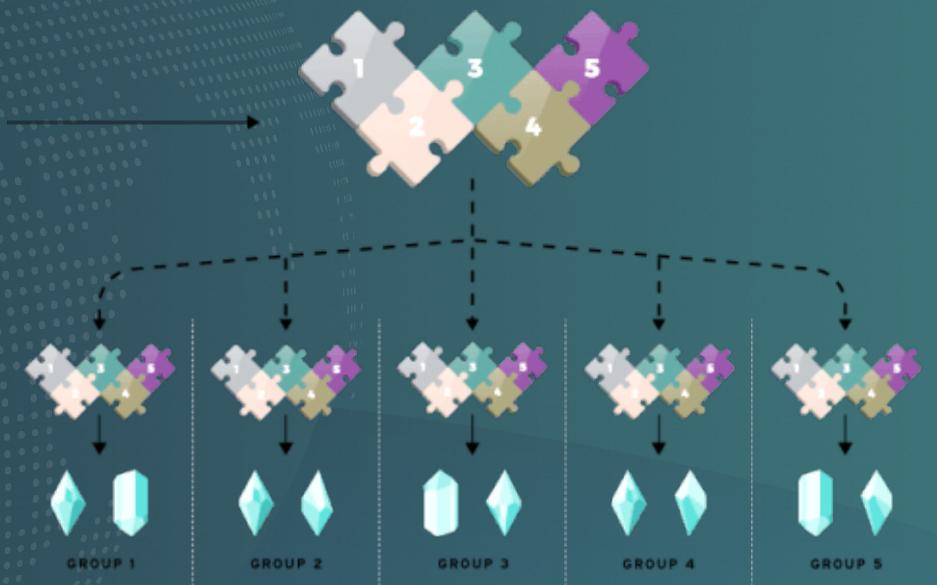


KEY DECISION IN SHARDING

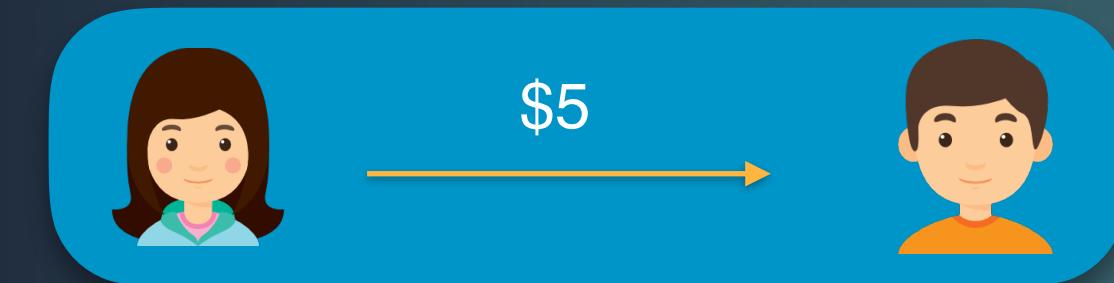
What goes to *which* shard?



DS COMMITTEE



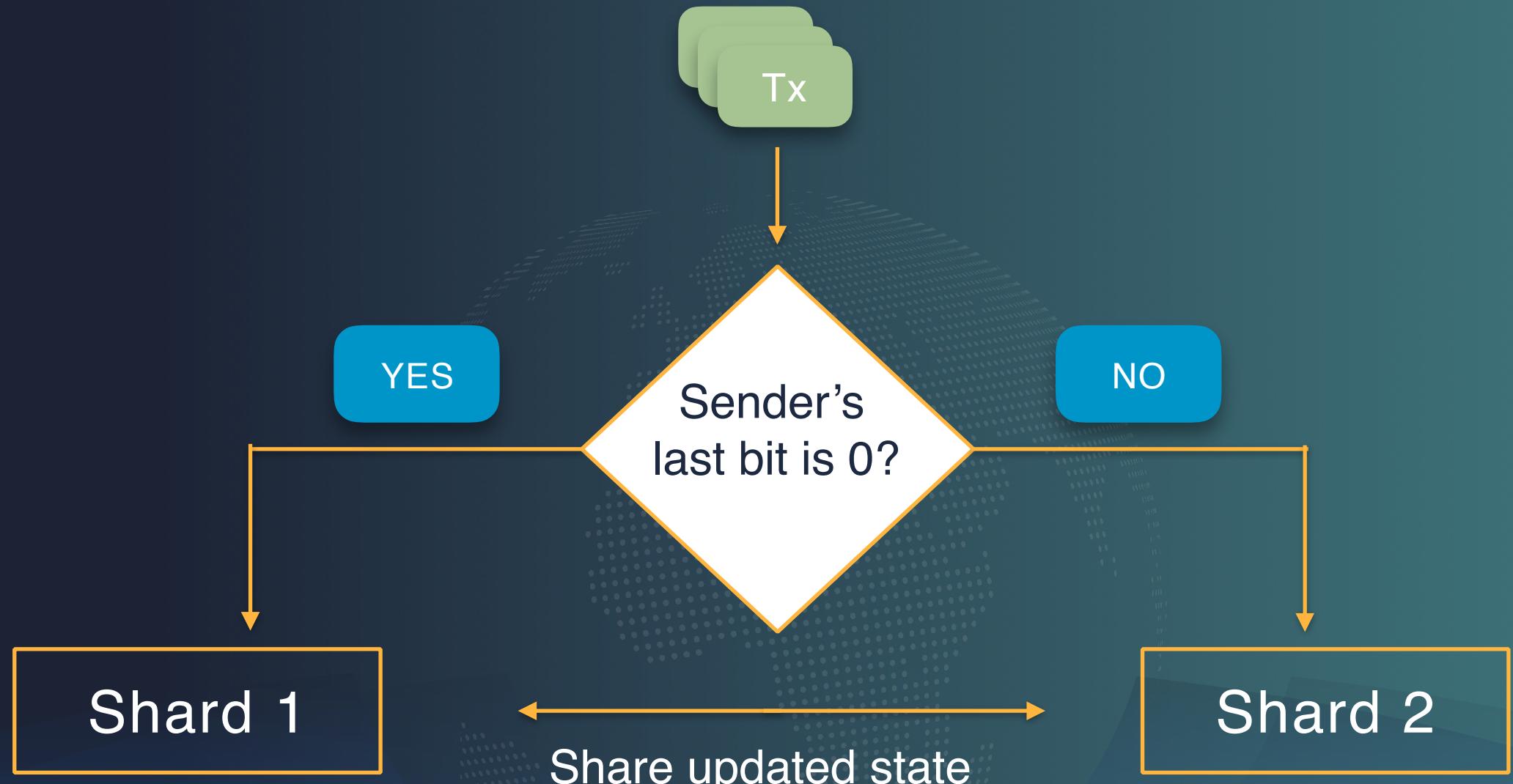
WHAT GOES TO WHICH SHARD?



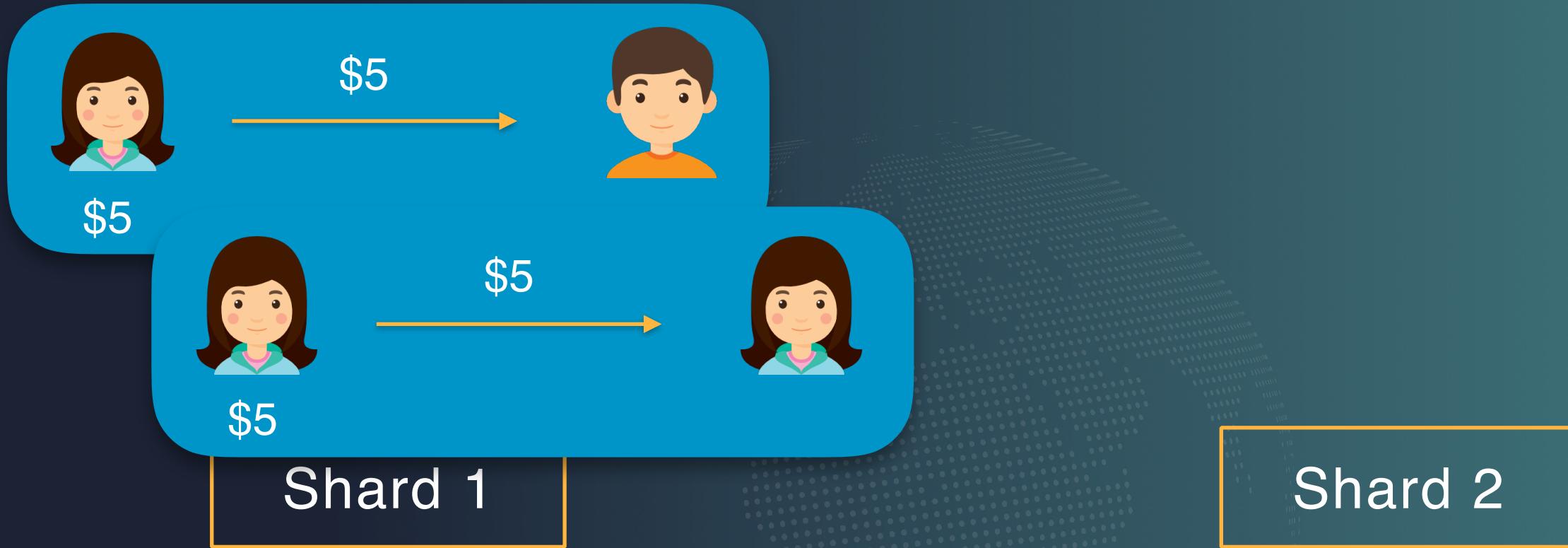
Shard 1

Shard 2

PAYMENTS WITH SHARDING



DOUBLE SPENDS?



CHALLENGES FROM CONTRACTS



(CONTRACT) STATE MATTERS

REGULAR PAYMENTS

- Only **sender's and recipient's states** are affected.
- Only the **sender can change the states**.

CONTRACT EXECUTION

VS

- Potentially **any account's state** can be affected.
- Potentially **any account can change the state**.



@ZILLIQA



ZILLIQA.COM

ONE SOLUTION

State Locking using 2PC



**Account Locking
limits parallel
processing**

concurrent changes



Improved Locking and Other Solutions

Lock states
rather than
accounts

Limit users from
calling certain
contracts

Break states
into UTXO-
like objects

Becomes application
specific

May limit usability

**Lose benefits of
accounts**



Our Approach

CONFLICT RESOLUTION *via* DETERMINISTIC TX ASSIGNMENT



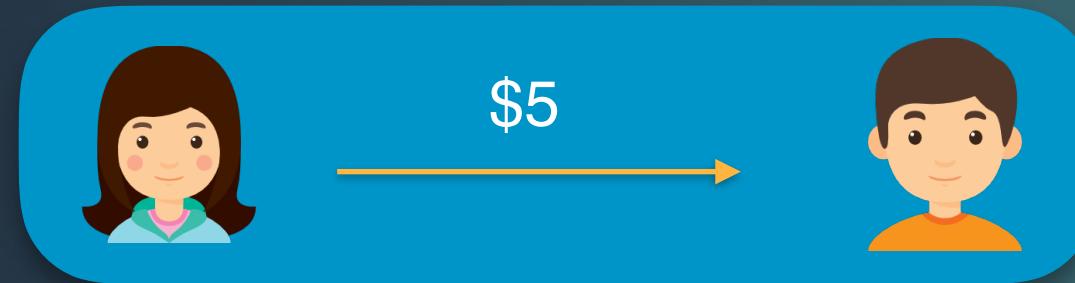
@ZILLIQA



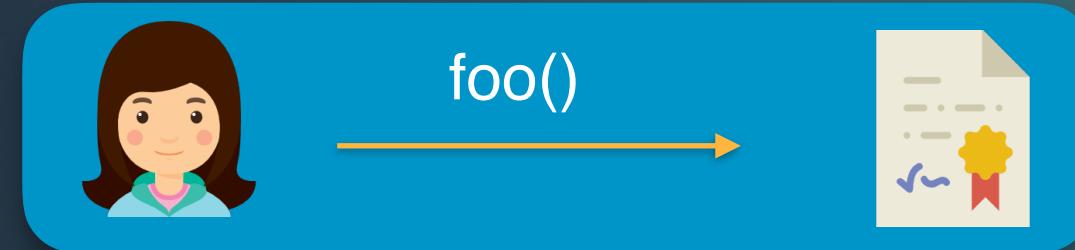
ZILLIQA.COM

TRANSACTION TYPES

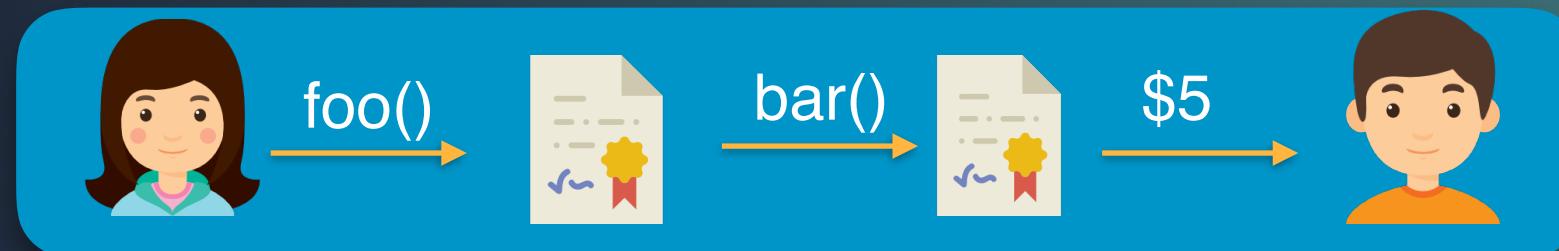
Type I



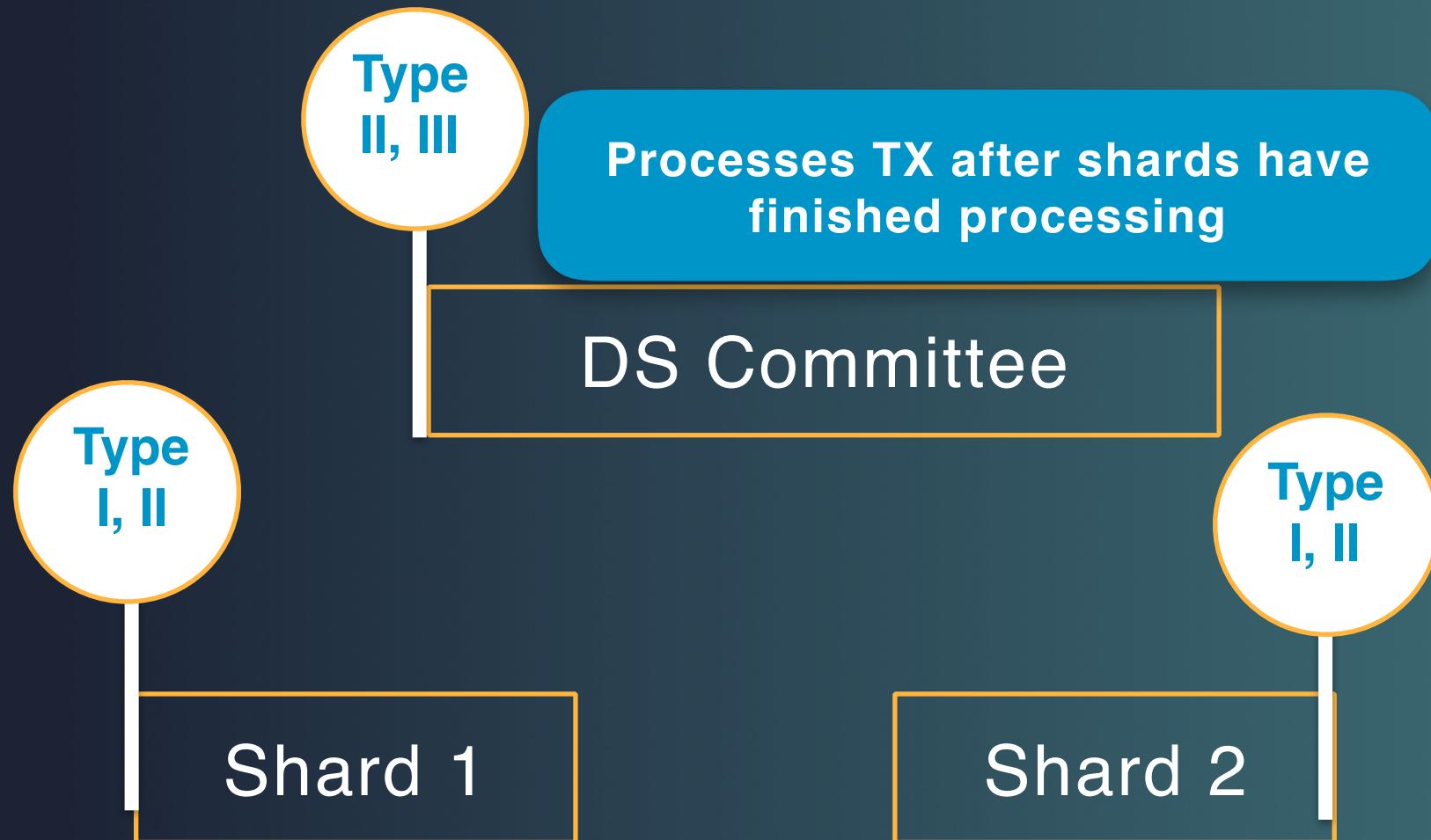
Type II



Type III

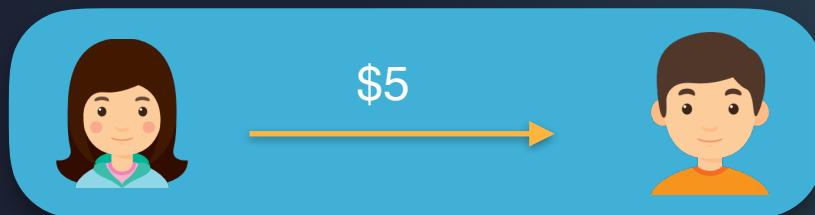


TRANSACTION PROCESSING



SHARDING TX & CONTRACTS

Type I

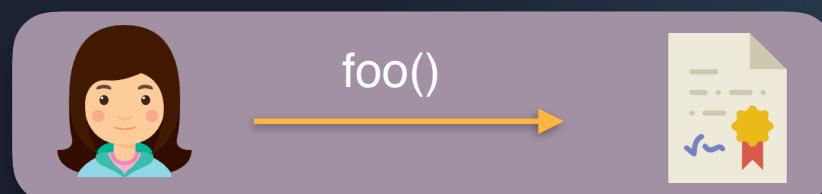


FROM 0X...0

FROM 0X...1

Shard 1

Type II



FROM & TO 0X...0

FROM & TO 0X...1

Shard 2

Type III



OTHERS

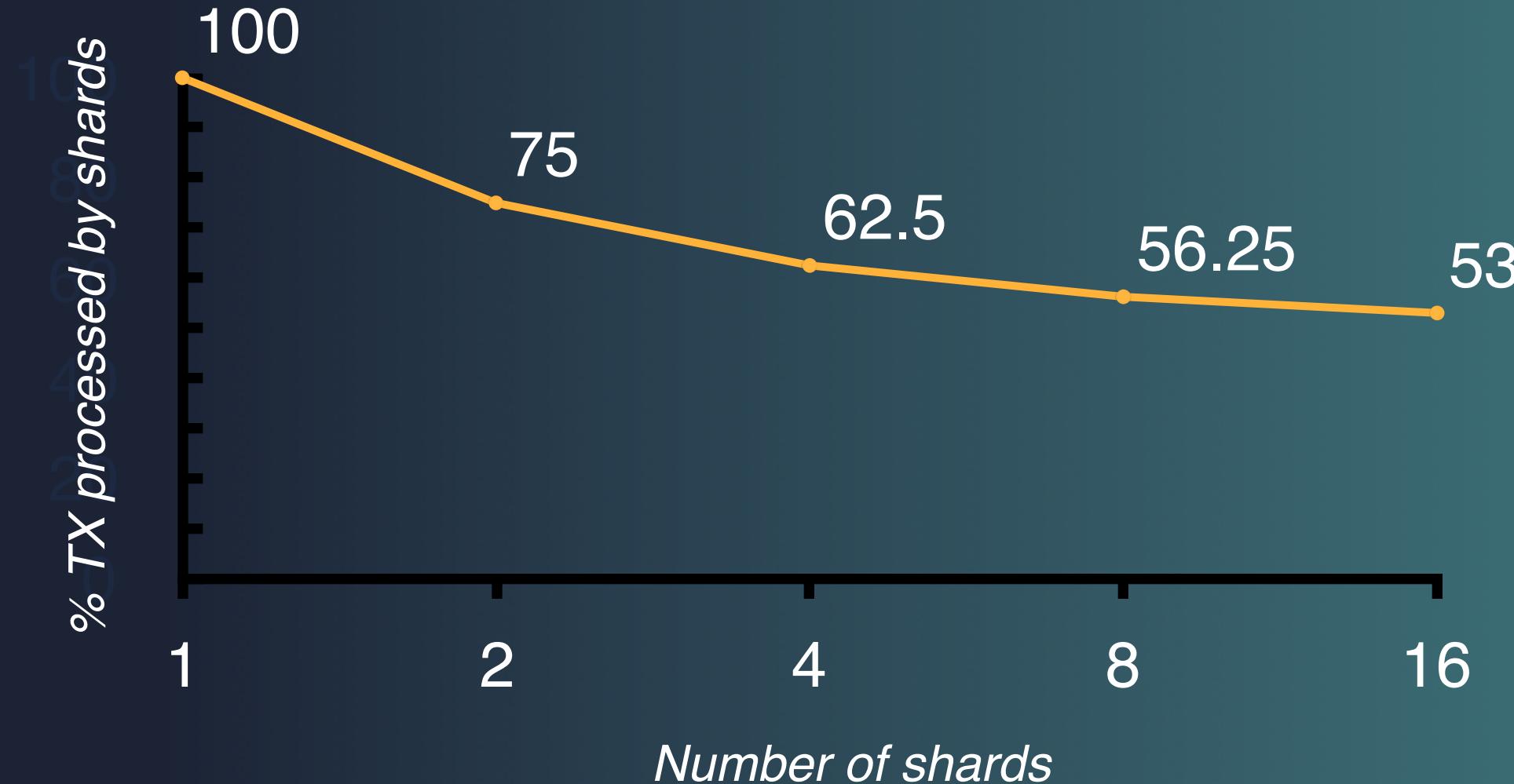
DS Committee

Example

Recipient

	U1(0)	U2(0)	U3 (1)	U4(1)	C1(0)	C2(0)	C3(1)	C4(1)
U1(0)	S1	S1	S1	S1	S1	S1	DS	DS
U2(0)	S1	S1	S1	S1	S1	S1	DS	DS
U3(1)	S2	S2	S2	S2	DS	DS	S2	S2
U4(1)	S2	S2	S2	S2	DS	DS	S2	S2

TX PROCESSED BY SHARDS



FUTURE WORK

Optimizations for Type II & III

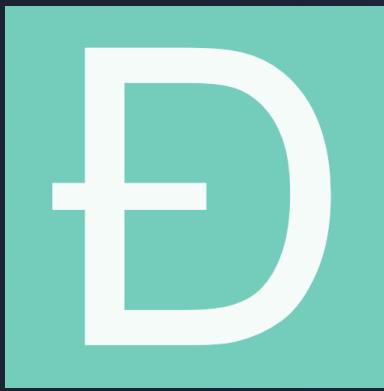
Language support for greater parallelism

Client support for better efficiency



SCILLA

INCIDENTS WITH SMART CONTRACTS



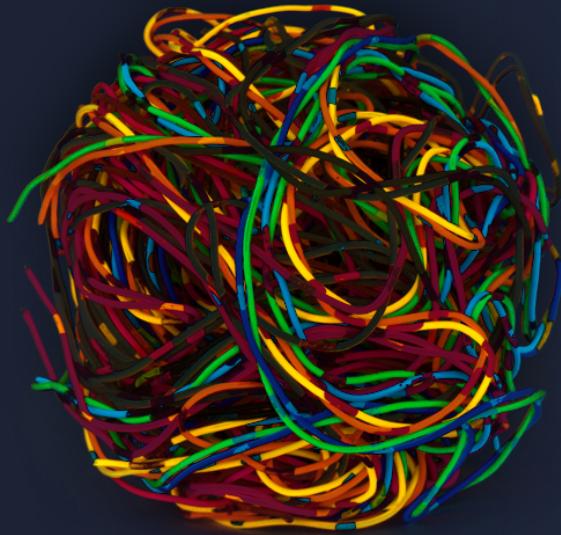
\$60 MIL
STOLEN



\$300 MIL
FROZEN



UNDERLYING CAUSES



COMPLEXITY



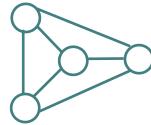
EXPECTED VS
UNEXPECTED
BEHAVIOR



NO FORMAL
VERIFICATION

SCILLA

SMART CONTRACTS ON ZILLIQA



NON-TURING
COMPLETE



DECIDABLE
CONTRACTS



AMENABLE TO
FORMAL VERIFICATION

CLEAN SEPARATION:
COMMUNICATION VS
COMPUTATION

KICKSTARTER IN SCILLA

IMMUTABLE PARAMS

```
contract Crowdfunding
  (owner      : ByStr20,
   deadline   : BNum,
   goal       : UInt128)
```

MUTABLE STATE

```
field backers : Map Address ⇒ Int = Emp
field success : Bool = False
```

STATE TRANSITIONS

```
transition Donate ()
```

```
transition GetFunds ()
```

```
transition ClaimBack ()
```

DAO INCIDENT

SOLIDITY

```
function reclaim
{
    uint amount = contributors[msg.sender]
    if(msg.sender.call.value(amount) == false)
        throw
    // reset the amount for sender
    contributors[msg.sender] = 0;
}
```

SEND

CALLBACK



INSTRUCTION NEVER EXECUTED

PREVENTING DAO INCIDENT

SECURITY RECOMMENDATION

```
// THIS CONTRACT HAS A BUG, DO NOT USE
function reclaim
{
    uint amount = contributors[msg.sender];
    if(msg.sender.call.value(amount) == false)
        throw
    // reset the amount for sender
    contributors[msg.sender] = 0;
}
```

```
// SAFE TO USE
function reclaim
{
    uint amount = contributors[msg.sender];
    contributors[msg.sender] = 0;
    msg.sender.transfer(amount);
}
```

CHECK-EFFECTS-INTERACTIONS

PREVENTING DAO INCIDENT AT THE LANGUAGE LEVEL

SOLIDITY

```
// SAFE TO USE

function reclaim()
{
    uint amount
    contributors[msg.sender] = 0;
    msg.sender.transfer(amount);
}
```

EXTERNAL CALLS ALWAYS HAPPEN AT THE END
REENTRANCY FREE

SCILLA

```
transition Reclaim
    // Check if the sender is eligible to
    // reclaim
    if (...)

    count → 0,
    → "failure">)

    // remove sender from the list
    let v = get(contributors, sender) in
    contributors := remove(contributors,
    sender);
    send (<to → sender, amount → v,
    tag → "main", msg → "refunded">)
```

HACKS ON MULTISIG CONTRACT

```
contract UnsafeContract2{
    address owner;

    function initowner(address _owner) { owner = _owner; }

    function transferTo(uint _amount, address _recipient) {
        if (msg.sender == owner)
            _recipient.transfer(_amount);
    }
}
```

Anyone can
set `_owner`

Transfer once
`_owner` is set

INTEGER OVERFLOW ATTACKS

```
function batchTransfer(address[]receivers, uint256 _value, bool) {
    uint cnt = _receivers.length;
    uint256 amount = uint256(cnt) * _value;
    require(_value > 0 && balances[msg.sender] >= amount);
    balances[msg.sender] = balances[msg.sender].sub(amount);
    for (uint i = 0; i < cnt; i++) {
        balances[_receivers[i]] = balances[_receivers[i]];
        Transfer(msg.sender, _receivers[i], _value);
    }
    return true;
}
```

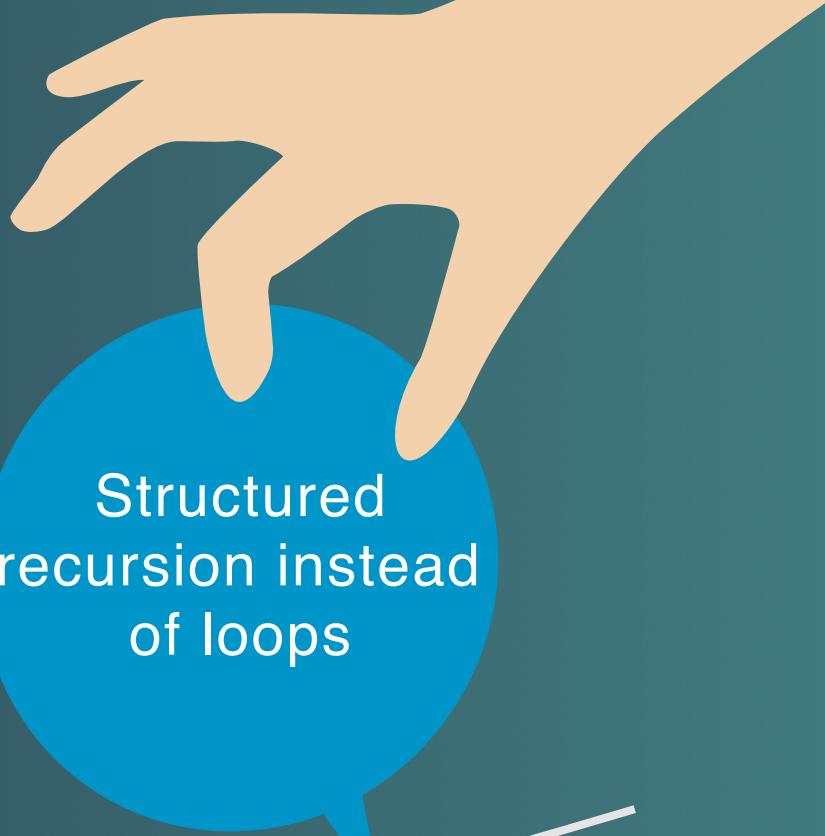
What if:

_value = 2^{255}
cnt = 2

amount = 0

Becomes possible to transfer
a large number of tokens

SECURITY FEATURES



Clean separation
between mutable
and immutable
params

Handles
arithmetic
underflows and
overflows

Structured
recursion instead
of loops

All programs
terminate and easier
resource analysis

No need for external
safe math like libs

Prevents multi-sig
wallet like incidents

THANK YOU!



ZILLIQA
—
/'ZILIKθ/



ENQUIRY@ZILLIQA.COM

@ZILLIQA

ZILLIQA.COM