CONSENSYS

Formal Specification and Verification of the Distributed Validator Technology Protocol

Roberto Saltini

Lead Researcher, ConsenSys R&D
consensys.net/research-and-development/

The 4 questions for today

Why do we need a Distributed Validator?

Why do we need to Formally Specify ad Verify the Distributed Validator Technology protocol?

How does our Formal Specification look like?

What have we achieved so far and what is left to do?

Acknowledgements

Grantors

ethereum foundation

Obol

ssv.network

CONSENSYS R&D

Distributed Systems Formal Verification

Roberto Saltini

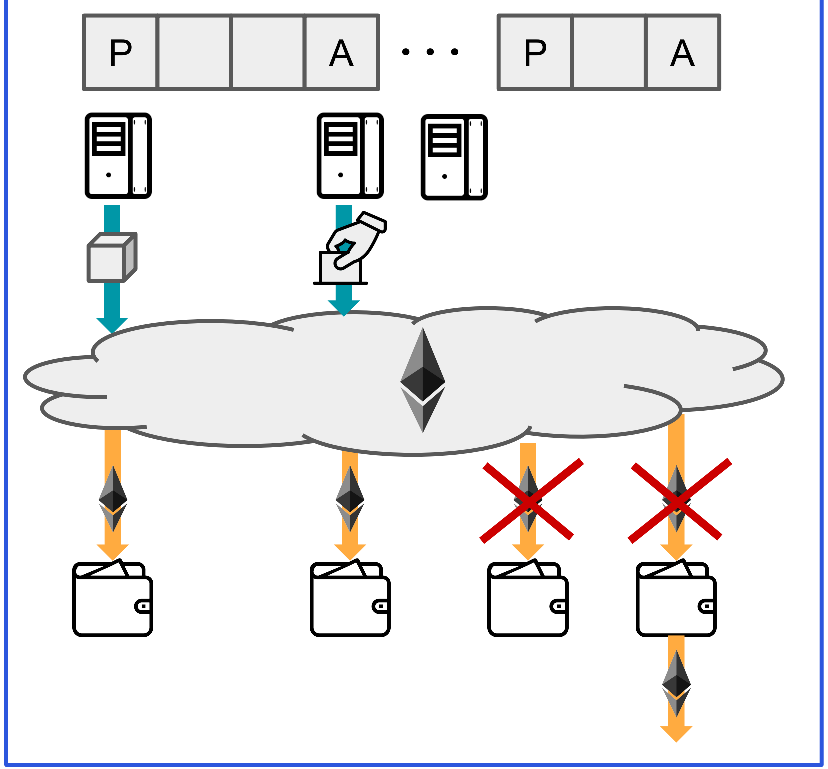
Thanh-Hai Tran

Consensys

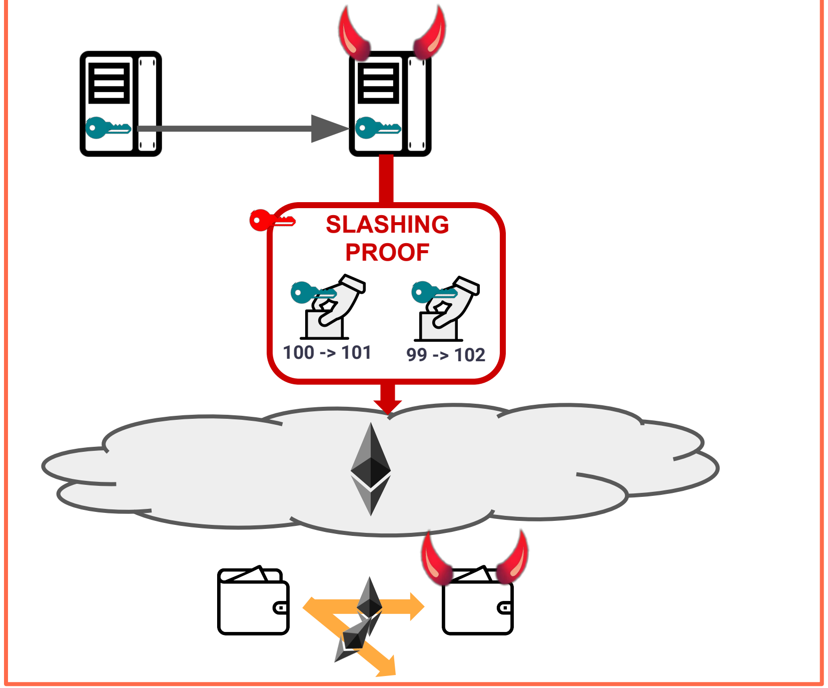
Why do we need a Distributed Validator?

What can go wrong with an Ethereum Validator?

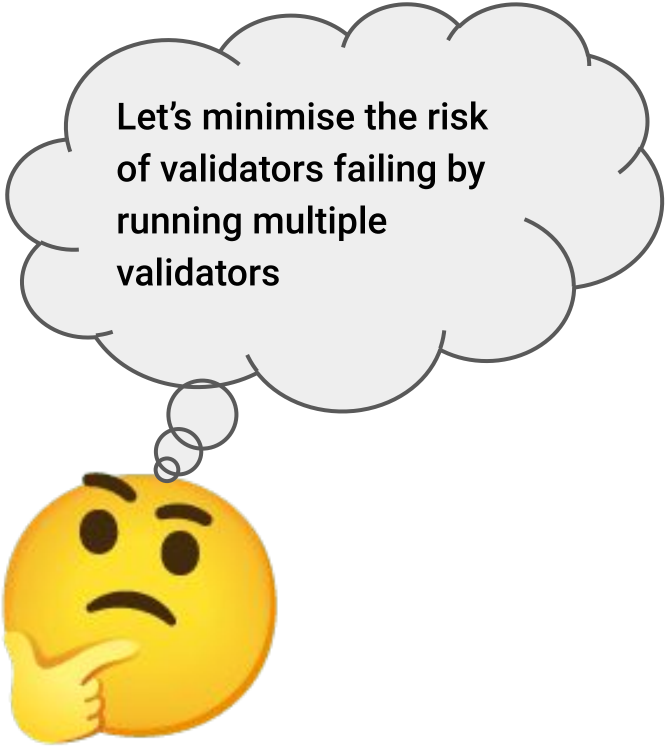
Node Failure → Miss Rewards
Incur Penalties



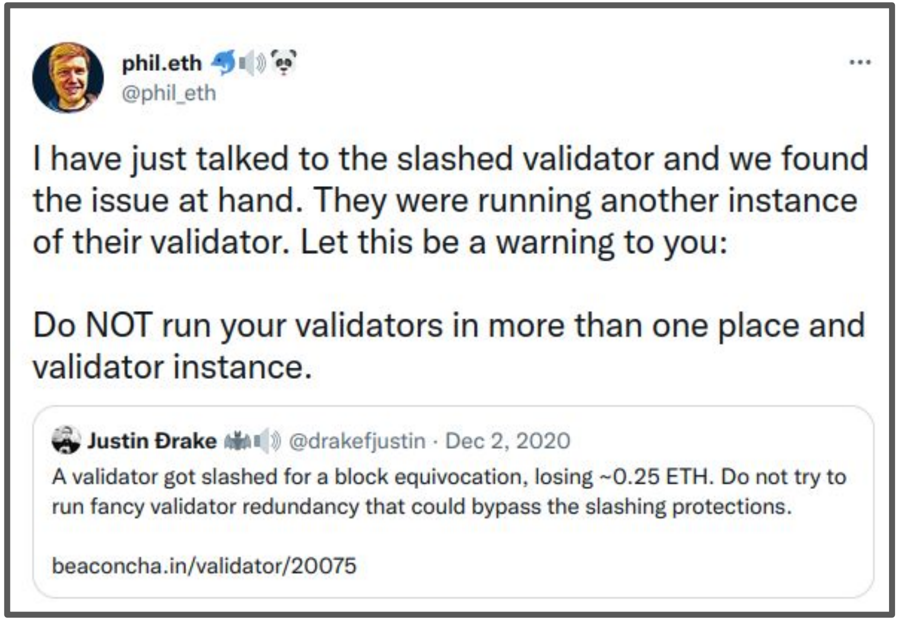
Validator Key Compromisation → Slashing



The solution is not as straightforward as one may think



Let’s minimise the risk of validators failing by running multiple validators



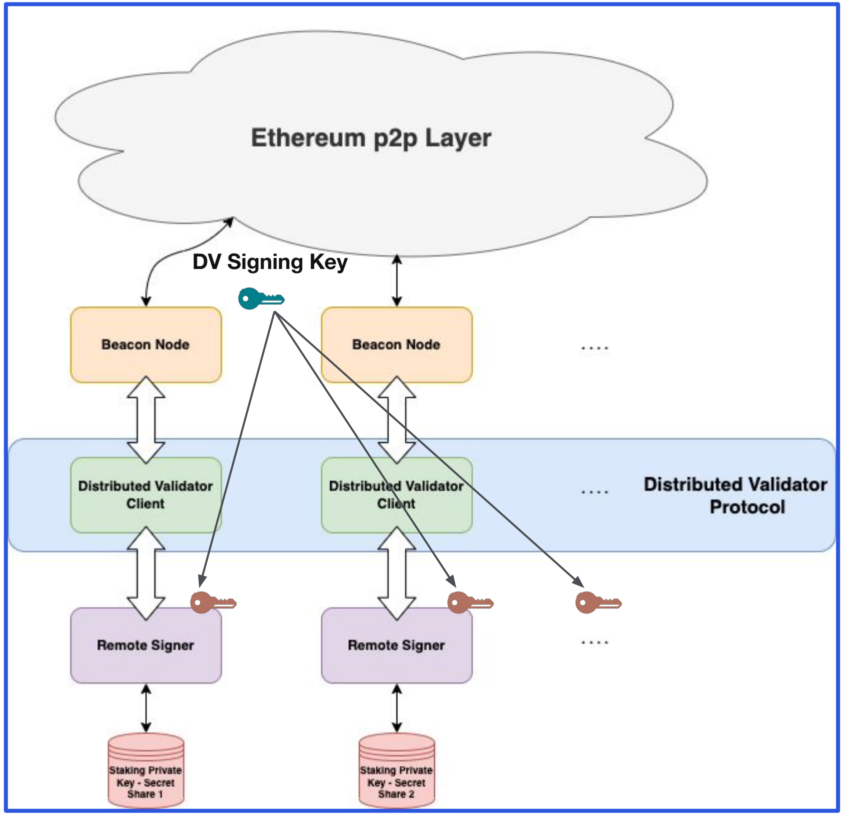
phil.eth
@phil_eth

I have just talked to the slashed validator and we found the issue at hand. They were running another instance of their validator. Let this be a warning to you:

Do NOT run your validators in more than one place and validator instance.

> **Justin Ðrake** @drakefjustin · Dec 2, 2020
> A validator got slashed for a block equivocation, losing ~0.25 ETH. Do not try to run fancy validator redundancy that could bypass the slashing protections.
>
> beaconcha.in/validator/20075

Nodes must be coordinated

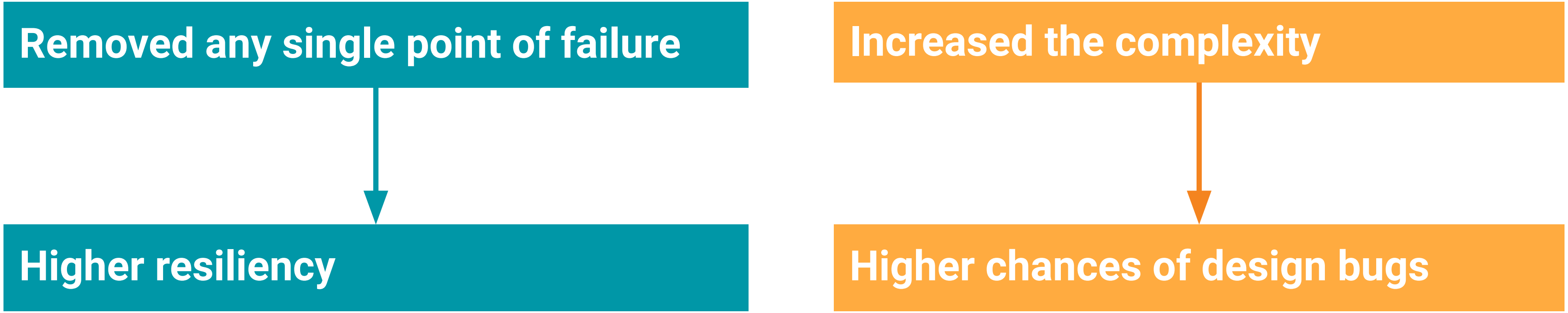

Ethereum p2p Layer
DV Signing Key
Beacon Node
Beacon Node
....
Distributed Validator Client
Distributed Validator Client
....
Distributed Validator Protocol
Remote Signer
Remote Signer
....
Staking Private Key - Secret Share 1
Staking Private Key - Secret Share 2

https://github.com/ethereum/distributed-validator-specs

CONSENSYS

Why do we need to Formally Specify ad Verify the Distributed Validator Technology protocol?

Now we have a distributed system to deal with



Removed any single point of failure

Higher resiliency

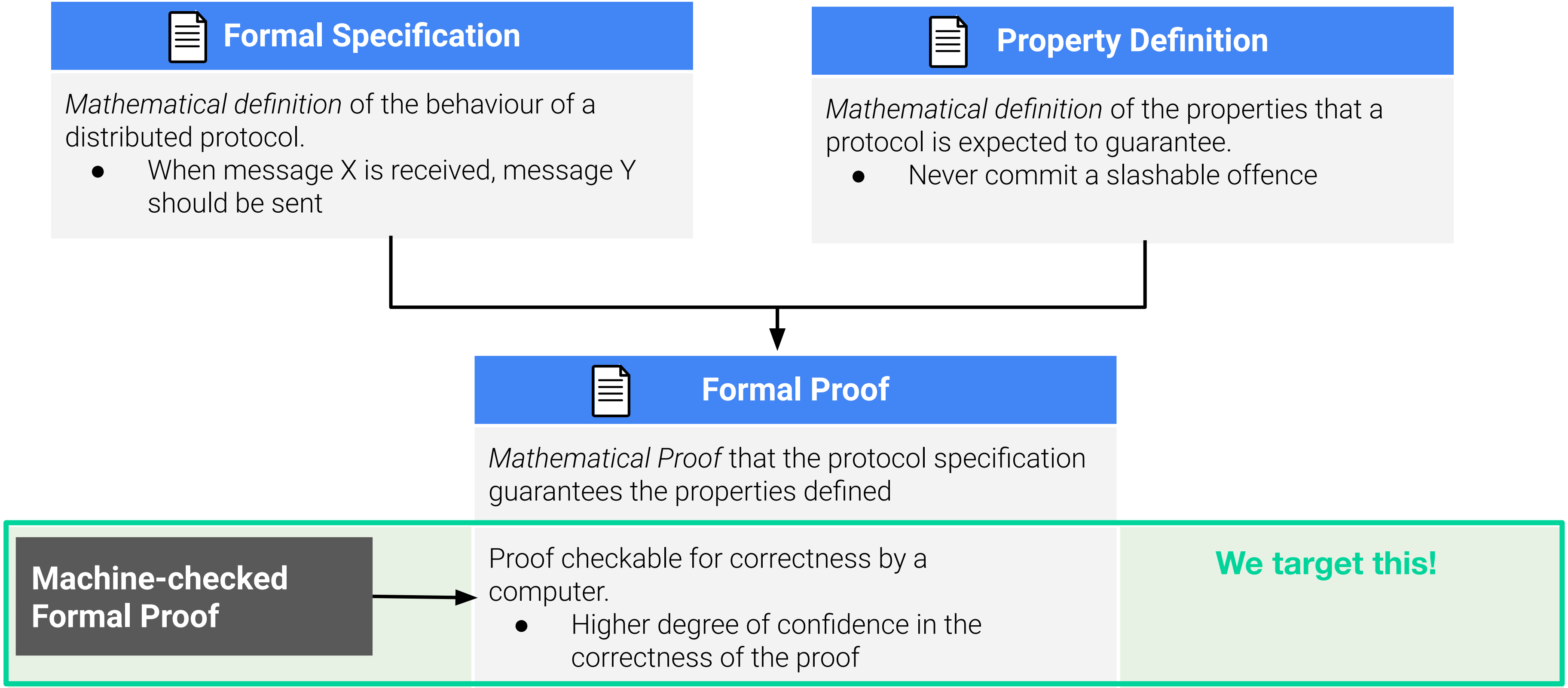
Increased the complexity

Higher chances of design bugs

How can Formal Verification help us?

	Formal Verification	Testing
	Exhaustive	**Non-exhaustive**
Network size considered	● Any size	● Only small sizes
Byzantine behaviours considered	● Any	● Only simplistic
If property X is not true	● It will be detected	● It may be detected
If property X is true	● Can prove it	● Cannot prove it

What does Formal Verification consist of?



Formal Specification

Mathematical definition of the behaviour of a distributed protocol.

- When message X is received, message Y should be sent

Property Definition

Mathematical definition of the properties that a protocol is expected to guarantee.

- Never commit a slashable offence

Formal Proof

Mathematical Proof that the protocol specification guarantees the properties defined

Machine-checked Formal Proof

Proof checkable for correctness by a computer.

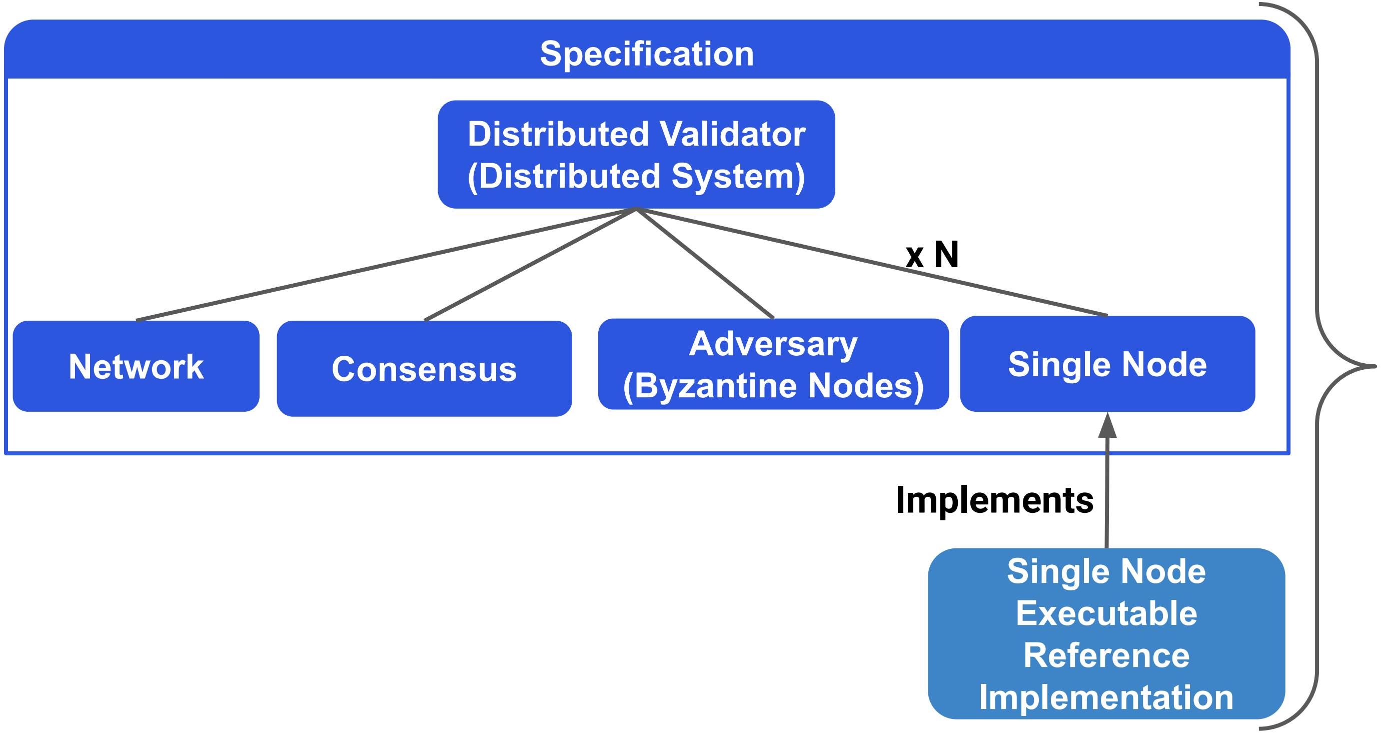
- Higher degree of confidence in the correctness of the proof

We target this!

CONSENSYS

How does our Formal Specification look like?

Our Formal Specification is Modular

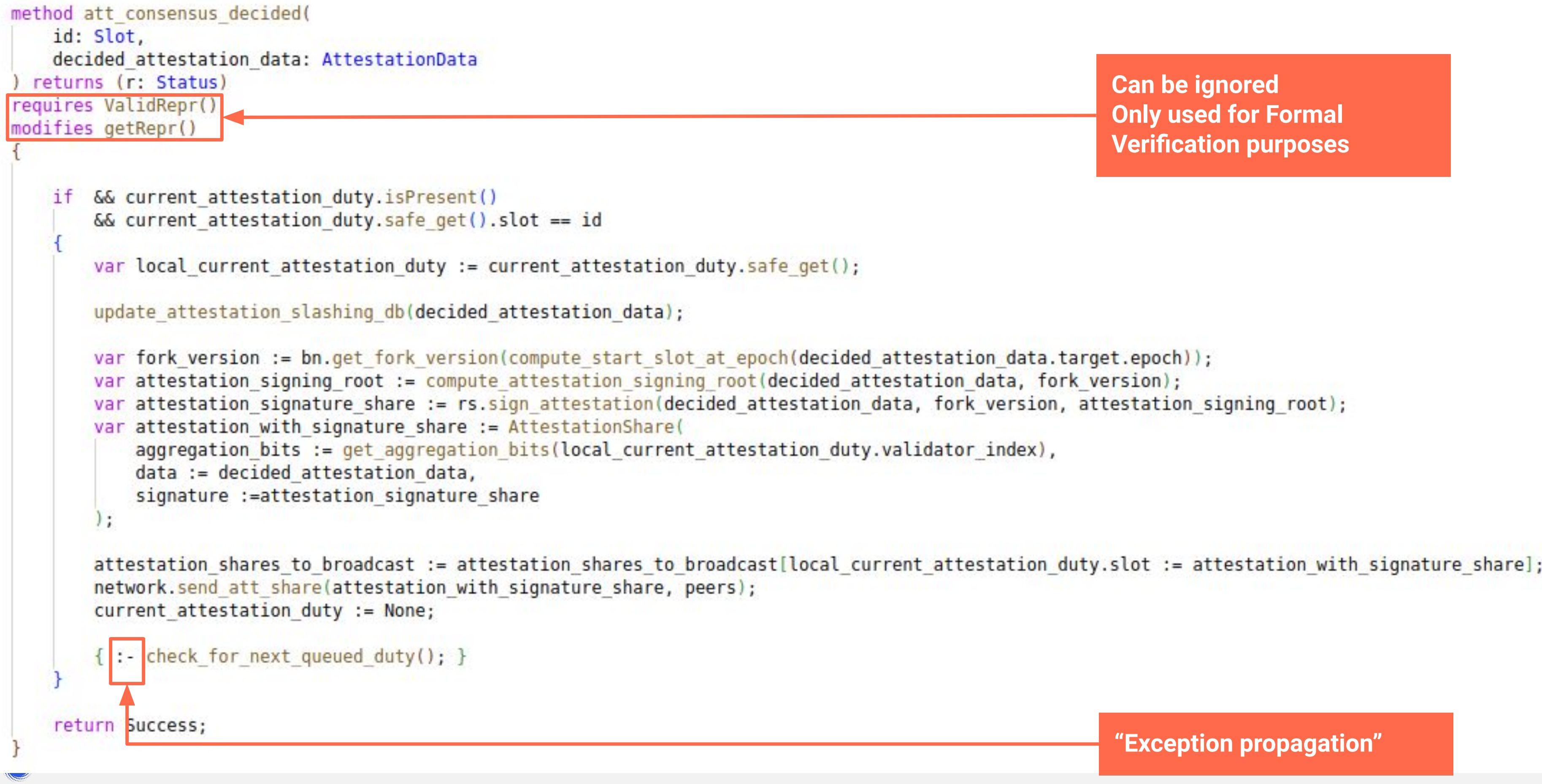

Specification
Distributed Validator (Distributed System)
x N
Network
Consensus
Adversary (Byzantine Nodes)
Single Node
Implements
Single Node Executable Reference Implementation

All written in Dafny

- Programming Language
- Python-like syntax
- Formal verification aware

https://github.com/dafny-lang/dafny

Let’s start from the Executable Reference Implementation



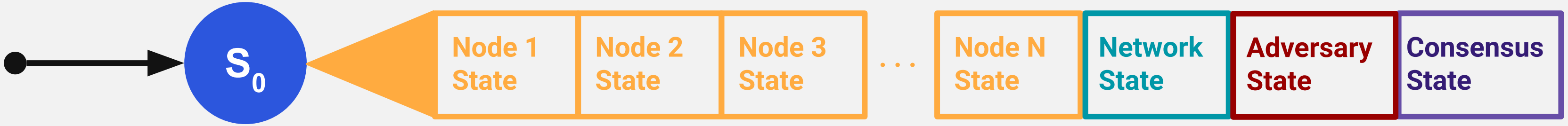
```
method att_consensus_decided(
    id: Slot,
    decided_attestation_data: AttestationData
) returns (r: Status)
requires ValidRepr()
modifies getRepr()
{

    if  && current_attestation_duty.isPresent()
        && current_attestation_duty.safe_get().slot == id
    {
        var local_current_attestation_duty := current_attestation_duty.safe_get();

        update_attestation_slashing_db(decided_attestation_data);

        var fork_version := bn.get_fork_version(compute_start_slot_at_epoch(decided_attestation_data.target.epoch));
        var attestation_signing_root := compute_attestation_signing_root(decided_attestation_data, fork_version);
        var attestation_signature_share := rs.sign_attestation(decided_attestation_data, fork_version, attestation_signing_root);
        var attestation_with_signature_share := AttestationShare(
            aggregation_bits := get_aggregation_bits(local_current_attestation_duty.validator_index),
            data := decided_attestation_data,
            signature :=attestation_signature_share
        );

        attestation_shares_to_broadcast := attestation_shares_to_broadcast[local_current_attestation_duty.slot := attestation_with_signature_share];
        network.send_att_share(attestation_with_signature_share, peers);
        current_attestation_duty := None;

        { :- check_for_next_queued_duty(); }
    }

    return Success;
}
```

Can be ignored
Only used for Formal
Verification purposes

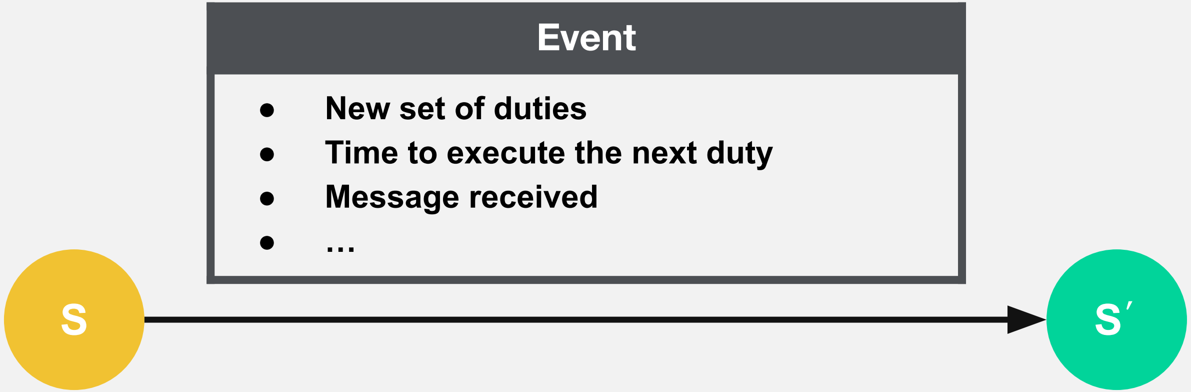
“Exception propagation”

The Distribute Validator Formal Specification defines how the system moves from one state to the next

Initial State



Transition

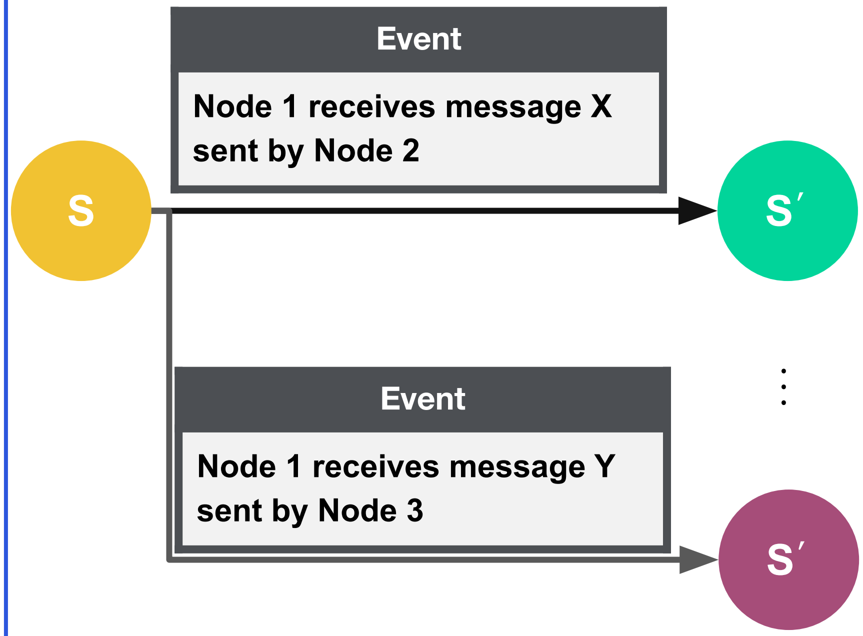


Event

- New set of duties
- Time to execute the next duty
- Message received
- …

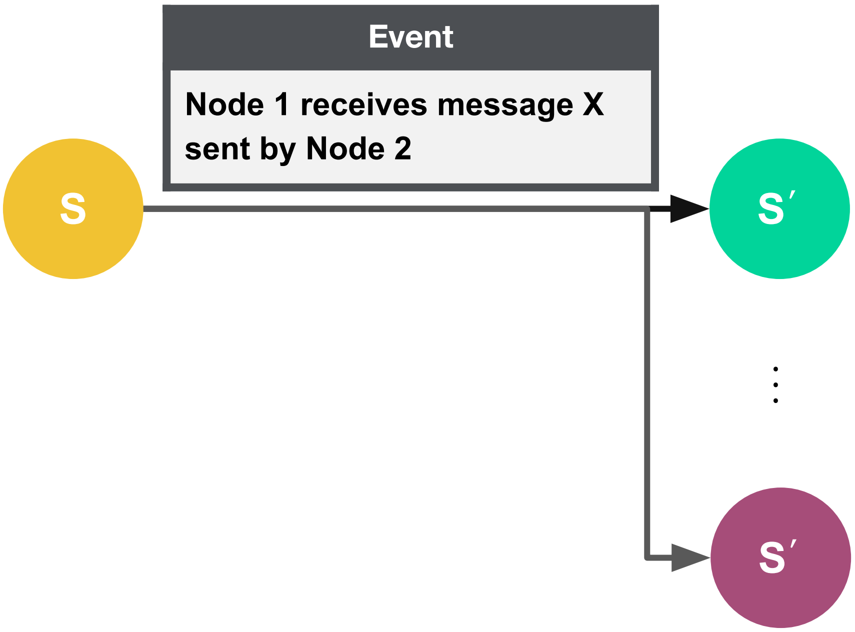
Our Formal Specification capture non deterministic behaviour

More than one event for each state


Event
Node 1 receives message X sent by Node 2
S
S′
Event
Node 1 receives message Y sent by Node 3
S′

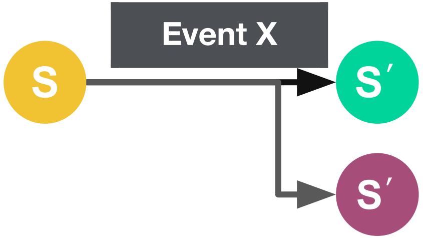
Captures network asynchrony

The same event can lead to more than one state

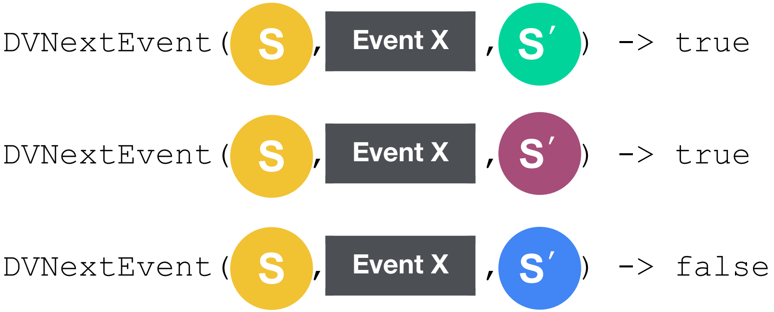


Allows to capture Byzantine behaviour

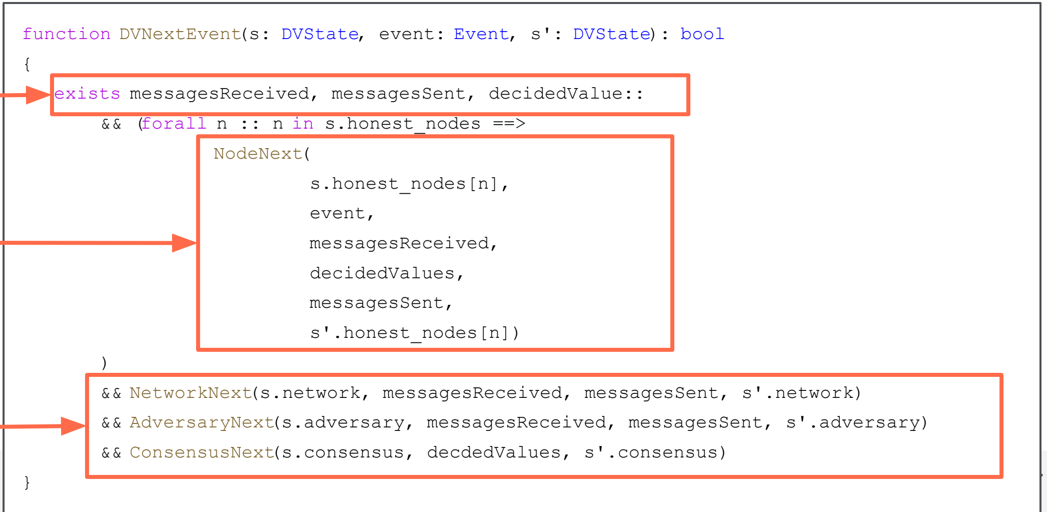
How is the Formal Specification written in Dafny?



```
DVNextEvent(src_state, event, dst_state): bool
```



```
DVNextEvent(S, Event X, S′) -> true
DVNextEvent(S, Event X, S′) -> true
DVNextEvent(S, Event X, S′) -> false
```



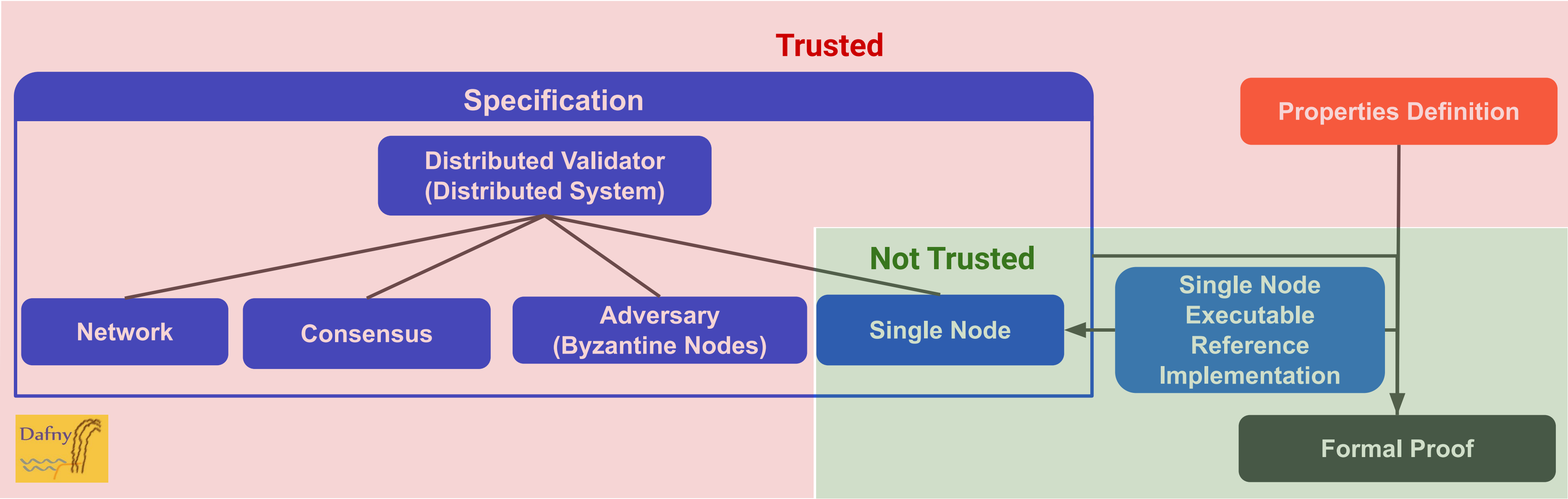
```
function DVNextEvent(s: DVState, event: Event, s': DVState): bool
{
 exists messagesReceived, messagesSent, decidedValue::
       && (forall n :: n in s.honest_nodes ==>
              NodeNext(
                       s.honest_nodes[n],
                       event,
                       messagesReceived,
                       decidedValues,
                       messagesSent,
                       s'.honest_nodes[n])
       )
       && NetworkNext(s.network, messagesReceived, messagesSent, s'.network)
       && AdversaryNext(s.adversary, messagesReceived, messagesSent, s'.adversary)
       && ConsensusNext(s.consensus, decdedValues, s'.consensus)
}
```

Source of non determinism

Single node transitions

Network, Adversary and Consensus transitions

What is trusted and what is not?


Trusted
Specification
Distributed Validator (Distributed System)
Network
Consensus
Adversary (Byzantine Nodes)
Not Trusted
Single Node
Single Node Executable Reference Implementation
Properties Definition
Formal Proof
Dafny

How do we ensure that what we trust is correct?

- Keep trusted specification part as simple as possible
- Peer review
- Use a Formal Verification tool with strong support

- Formally proving both safety (nothing bad will ever happen) and liveness properties (something good will eventually happen) highly reduces the chances of errors in the assumptions

CONSENSYS

What have we achieved so far and what is left to do?

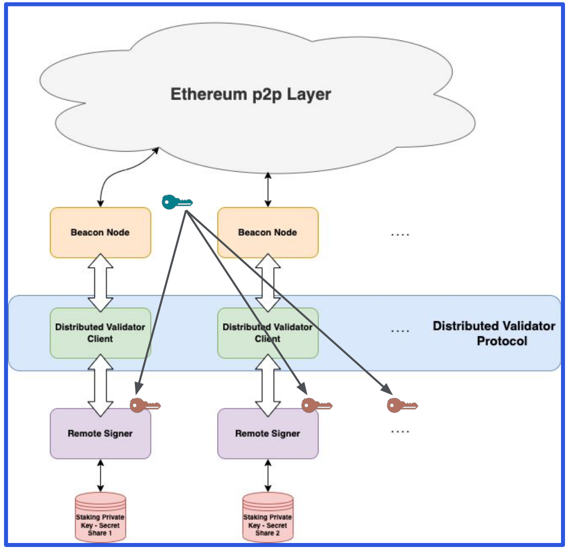
What have we been able to formally prove?

Assumptions

- Arbitrary message delay, including lost messages
- Beacon Nodes on completely different, but not conflicting, canonical chains
- The number of nodes that either are Byzantine or had their signing key compromised < 1/3
- Signature unforgeability and uniqueness
- Sound ⅔-threshold signature scheme

Proofs

- **No Slashable Attestations**
  Slashable attestations signed by the Distributed Validator signing key can never be created.


Ethereum p2p Layer
Beacon Node
Beacon Node
....
Distributed Validator Client
Distributed Validator Client
....
Distributed Validator Protocol
Remote Signer
Remote Signer
....
Staking Private Key - Secret Share 1
Staking Private Key - Secret Share 2

What have we been able to formally prove?

Assumptions

- Arbitrary message delay, including lost messages
- Beacon Nodes on completely different, but not conflicting, canonical chains
- The number of nodes that either are Byzantine or had their signing key compromised < 1/3
- Signature unforgeability and uniqueness
- Sound ⅔-threshold signature scheme

Proofs

- **No Slashable Attestations**
  Slashable attestations signed by the Distributed Validator signing key can never be created.
- **The reference implementation adheres to the specification**
  A Distributed Validator where each node runs the reference implementation ensures the “No Slahable Attestations” property defined above.

What is left to do?

Done ✔

- Formal Specification of Distributed Attestation signing
- Reference Implementation for Distributed Attestation signing
- Formal Proof of No Slashable Attestations
- github.com/ConsenSys/distributed-validator-formal-specs-and-verification

To Do ⚒

- Simplify the trusted part of the specification
- Formal Specification of Distributed Block Proposing and Block Header Signing
- Formal Proof of No Slashable Blocks
- Formal Proof that the Distributed Validator eventually issues valid attestations, blocks and block header signatures

github.com/ConsenSys/distributed-validator-formal-specs-and-verification

consensys.net/research-and-development

Thank You!

Roberto Saltini

Lead Researcher, ConsenSys R&D

roberto.saltini@consensys.net

www.linkedin.com/in/roberto-saltini/