# Model-based testing with TLA+ and Apalache

TLA+ Community Event, October 2020

**Andrey Kuprianov**
**Igor Konnov**

informal SYSTEMS

# Who We Are

*Verifiable distributed*
**systems** *and* **organizations.**

**We envision an open-source ecosystem of cooperatively owned and governed distributed organizations running on reliable distributed systems.**

informal SYSTEMS

# Who We Are

## Blockchain Infrastructure

Tendermint-rs

ibc-rs

## Formal Verification Tools

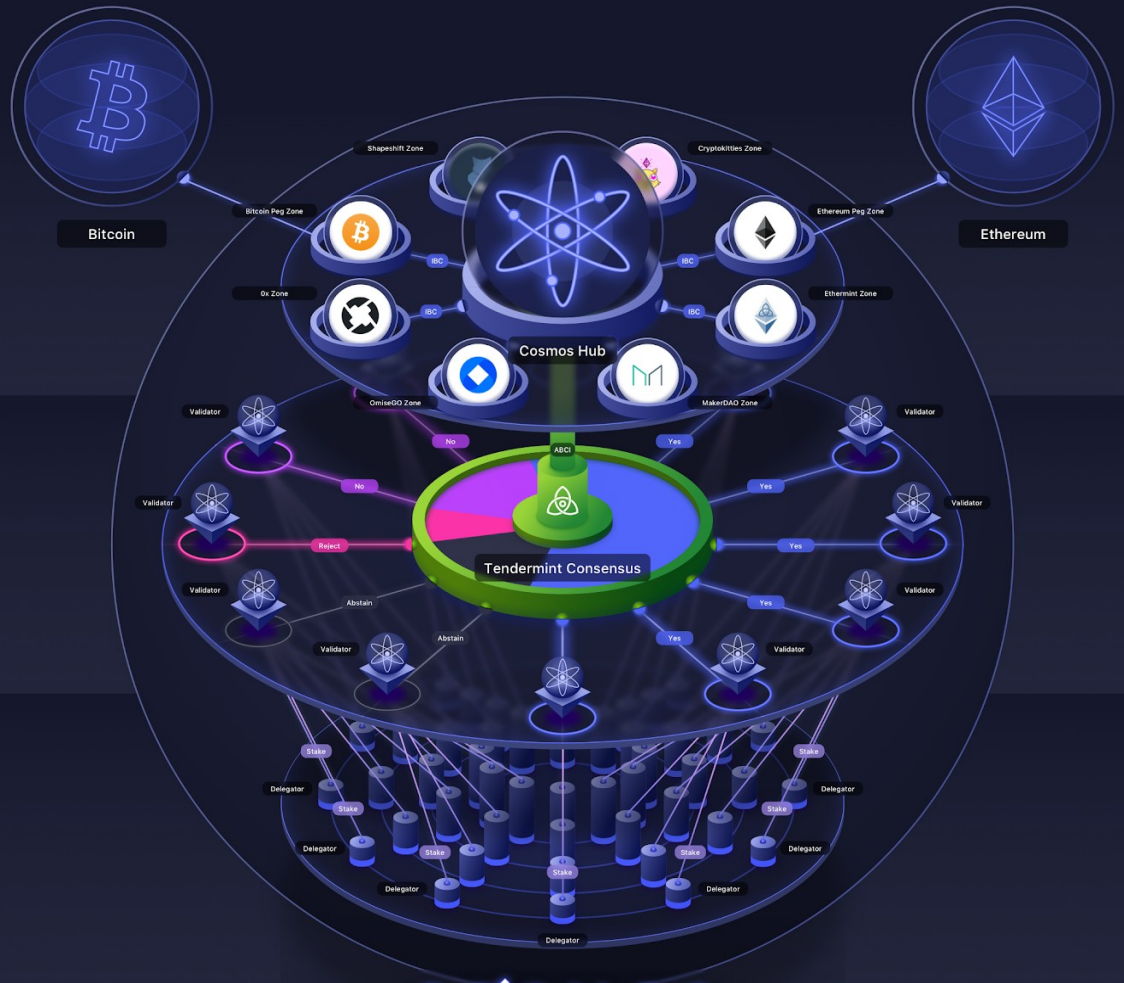
Apalache

## Business Operations Tools

Themis Contract

Our Infrastructure powers the COSMOS Network

informal SYSTEMS




Zones

Validators

Delegators

# COSMOS

INTERNET OF BLOCKCHAINS

https://cosmos.network

by @zcpeng

COSMOS

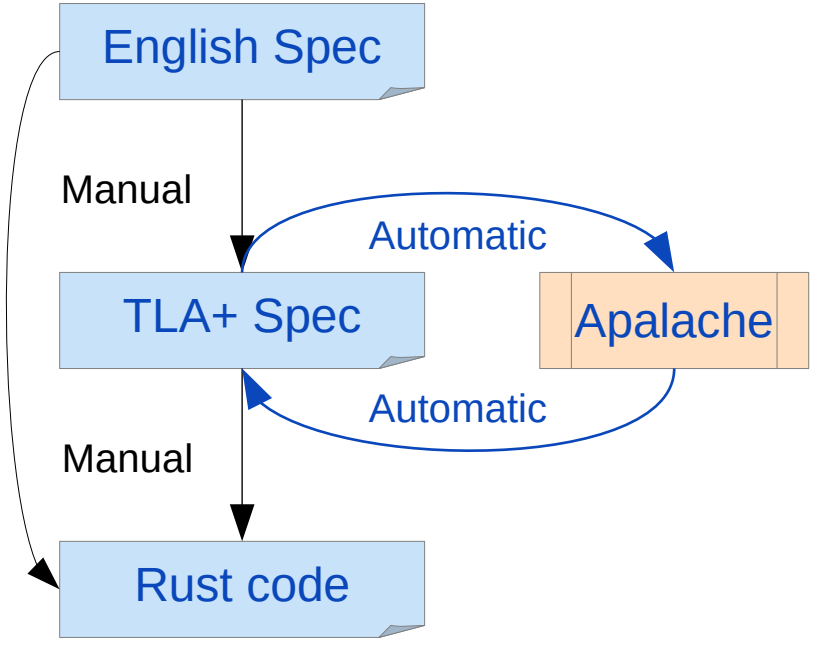
# STARGATE

The Internet of Blockchains is on the horizon.

# Verification Driven Development

github.com/informalsystems/vdd

- Developed English and TLA+ specifications of the core protocols
- Model-checked the TLA+ specs with Apalache
- Helped to fix subtle protocol issues, clarify the protocol understanding
- Realized the gap between the specs and the code

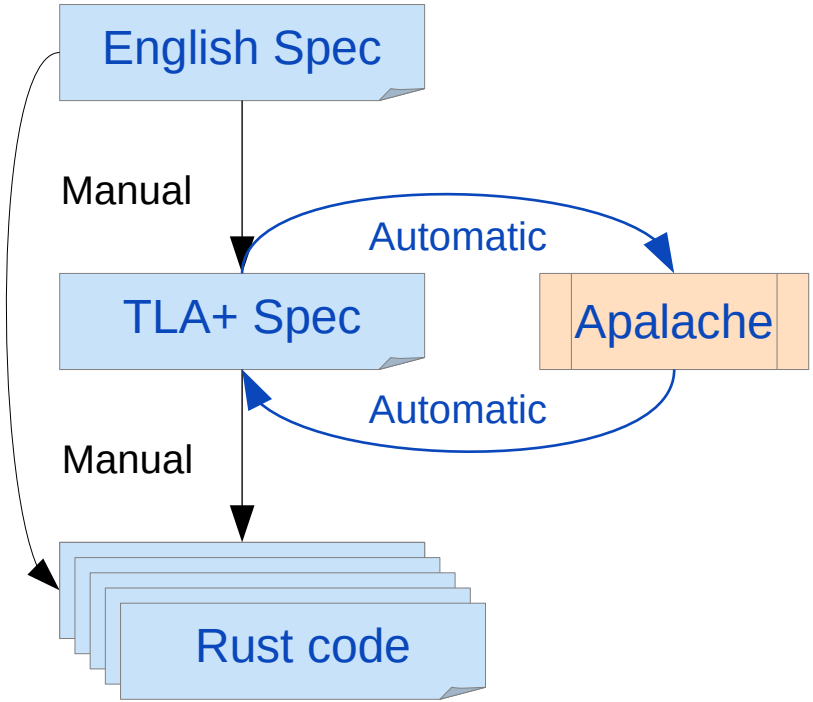



informal

# Verification Driven Development

github.com/informalsystems/vdd

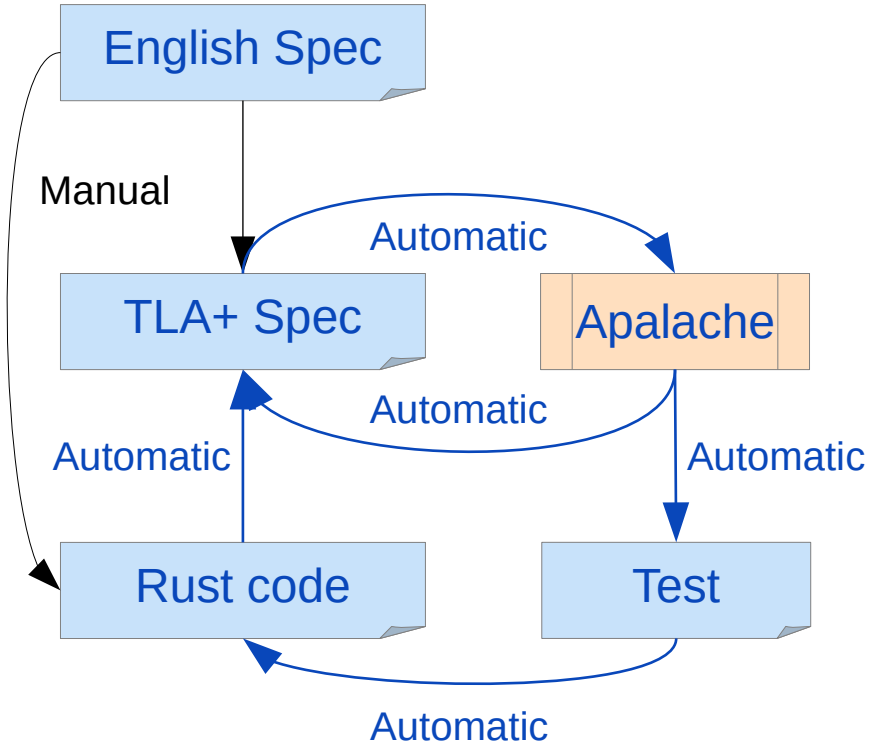
- Developed English and TLA+ specifications of the core protocols
- Model-checked the TLA+ specs with Apalache
- Helped to fix subtle protocol issues, clarify the protocol understanding
- Realized the gap between the specs and the code




informal

# Model-Based Testing

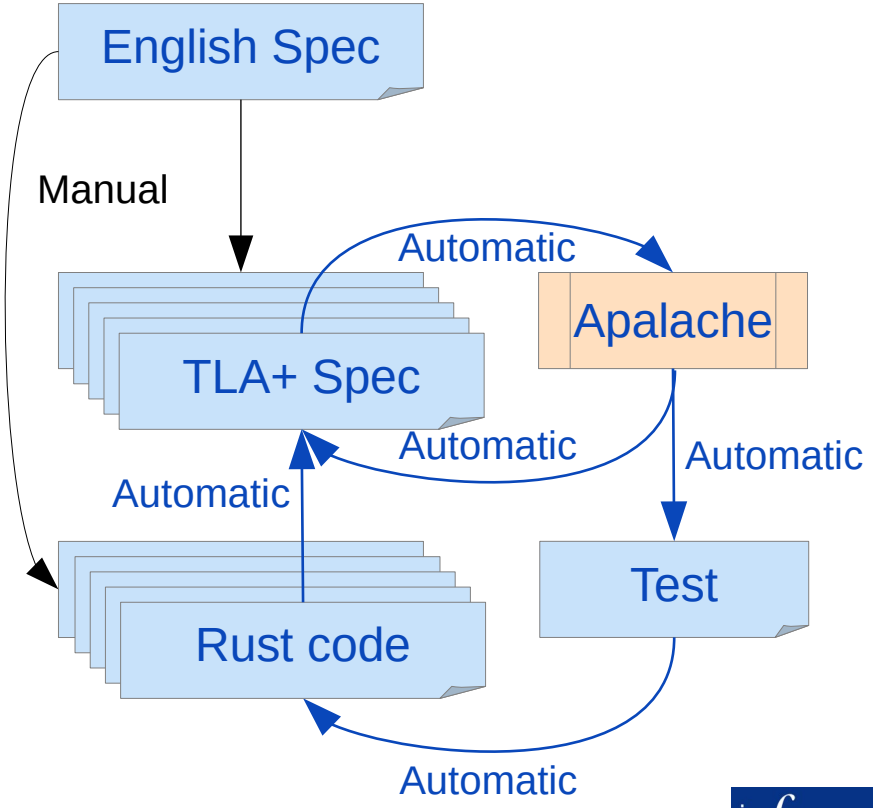
- Generate integration tests for production code from TLA+ specs and simple assertions
- Helped to eliminate the growing spec/code divergence
- Significantly improves
  - ease of writing / using the tests
  - tests maintainability
  - code coverage




informal

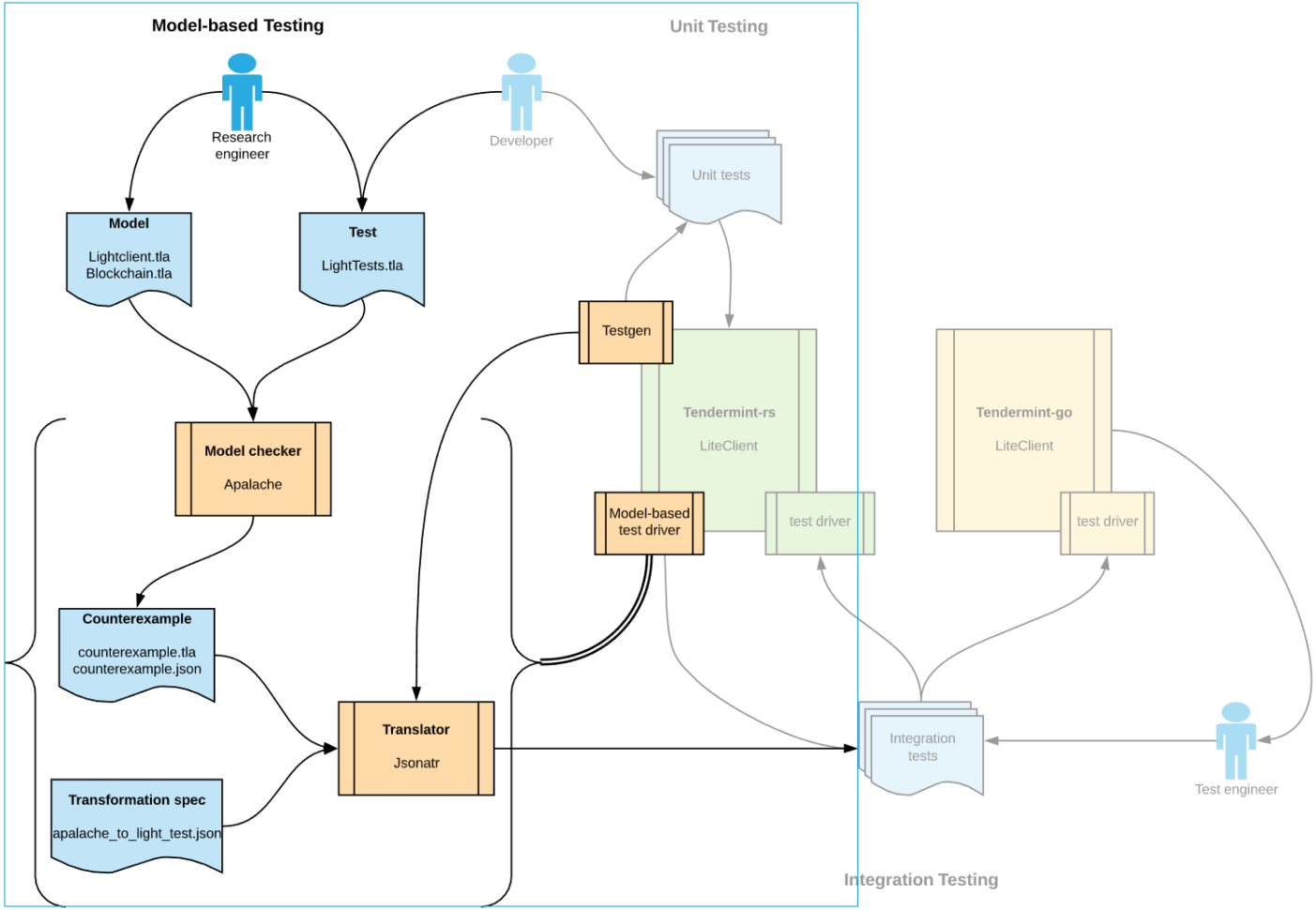
# Model-Based Testing

- Generate integration tests for production code from TLA+ specs and simple assertions
- Helped to eliminate the growing spec/code divergence
- Significantly improves
  - ease of writing / using the tests
  - tests maintainability
  - code coverage



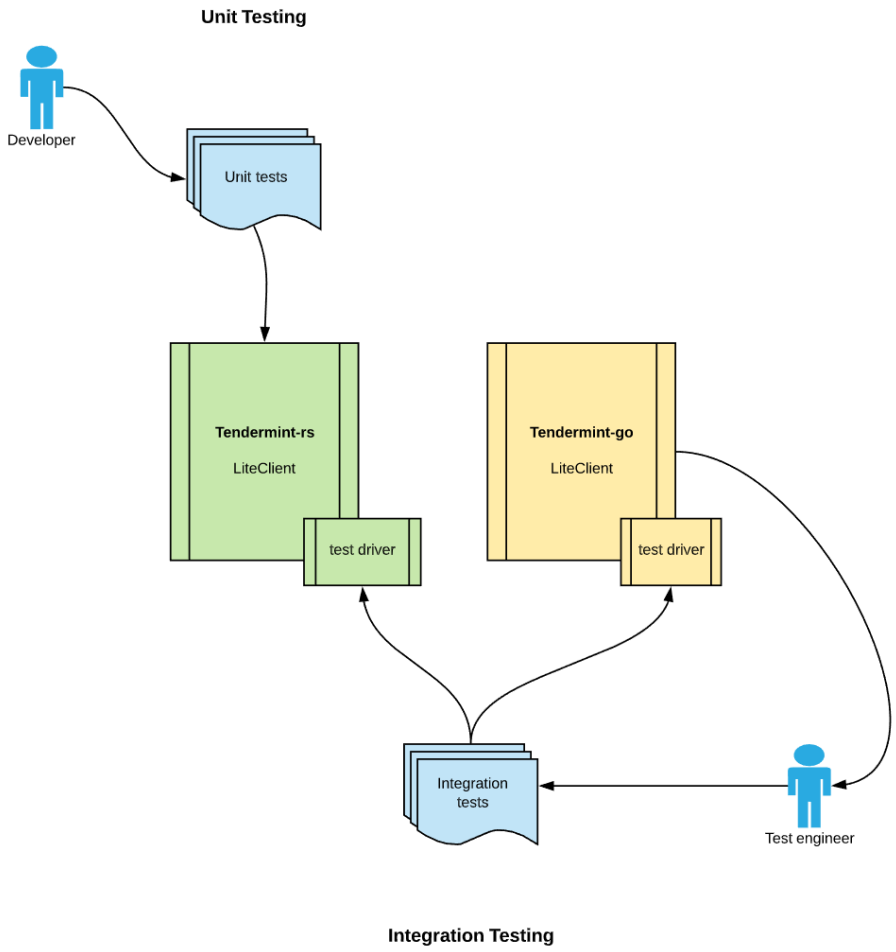

informal

# … before diving into MBT...




informal

# Standard testing




informal

# Standard testing: manual code generation

```go
func caseSingleSeqValidatorSetChangesMoreThanTwoThirds(valList ValList) {

    copiedValList := valList.Copy()
    description := "Case: two lite blocks, more than 2/3 validator set changes, no error"
    testCase := generateAndMakeNextValsUpdateTestCase(
        description,
        copiedValList.Validators[0:4],
        copiedValList.PrivVals[0:4],
        copiedValList.Validators[3:7],
        copiedValList.PrivVals[3:7],
        expectedOutputNoError,
    )
    file := SINGLE_STEP_SEQ_PATH + "validator_set/more_than_two_thirds_valset_changes.json"
    testCase.genJSON(file)
}
```

informal

# Standard testing: integration test


```
{
    "description": "Case: two lite blocks, more than 2/3 validator set changes, no error",
    "initial": {"trusting_period": "10800000000000"...},
    "input": [
        {
            "signed_header": {
                "header": {
                    "version": {"block": "0"...},
                    "chain_id": "test-chain-01",
                    "height": "2",
                    "time": "2019-11-02T15:04:10Z",
                    "last_block_id": {"hash": "15F15EF50BDE2018F4B129A827F90C18222C757770C8295EB8EE7BF50E761BC0"...},
                    "last_commit_hash": "D5439DD65D45EF1E51412691BCF2F6741D48AC1325572E08D48BD72F80669E70",
                    "data_hash": "",
                    "validators_hash": "26952B5D784A1564D167DF98D2D37376B5E77771928256D25E6FF9AE3AD11564",
                    "next_validators_hash": "48EDBAFFB564AAA7AE0A8E3DE2AFA0ED4F64444B48587937B4F454703596404C",
                    "consensus_hash": "048091BC7DDC283F77BFBF91D73C44DA58C3DF8A9CBC867405D8B7F3DAADA22F",
                    "app_hash": "6170705F68617368",
                    "last_results_hash": "",
                    "evidence_hash": "",
                    "proposer_address": "026CC7B6F3E62F789DBECEC59766888B5464737D"
                },
                "commit": {
                    "height": "2",
                    "round": "1",
                    "block_id": {"hash": "2DDDDF1297BED341DDDCA18670FD3CF583BC74D023DA2D8336AC2CDEEABA9BDA"...},
                    "signatures": [
                        {
                            "block_id_flag": 2,
                            "validator_address": "01F527D77D3FFCC4FCFF2DDC2952EEA5414F2A33",
                            "timestamp": "2019-11-02T15:04:15Z",
                            "signature": "/bNg77MUVBZ45y2iNUeLRqPDXphqo0xf2av9EazersxSxS6Gg9cBoAVkpKo9Ix5zOlrCuP7MNH9wYNYmbel2Cg=="
                        },
                        {"block_id_flag": 2...},
                        {"block_id_flag": 2...},
                        {"block_id_flag": 2...}
                    ]
                }
            },
```


informal

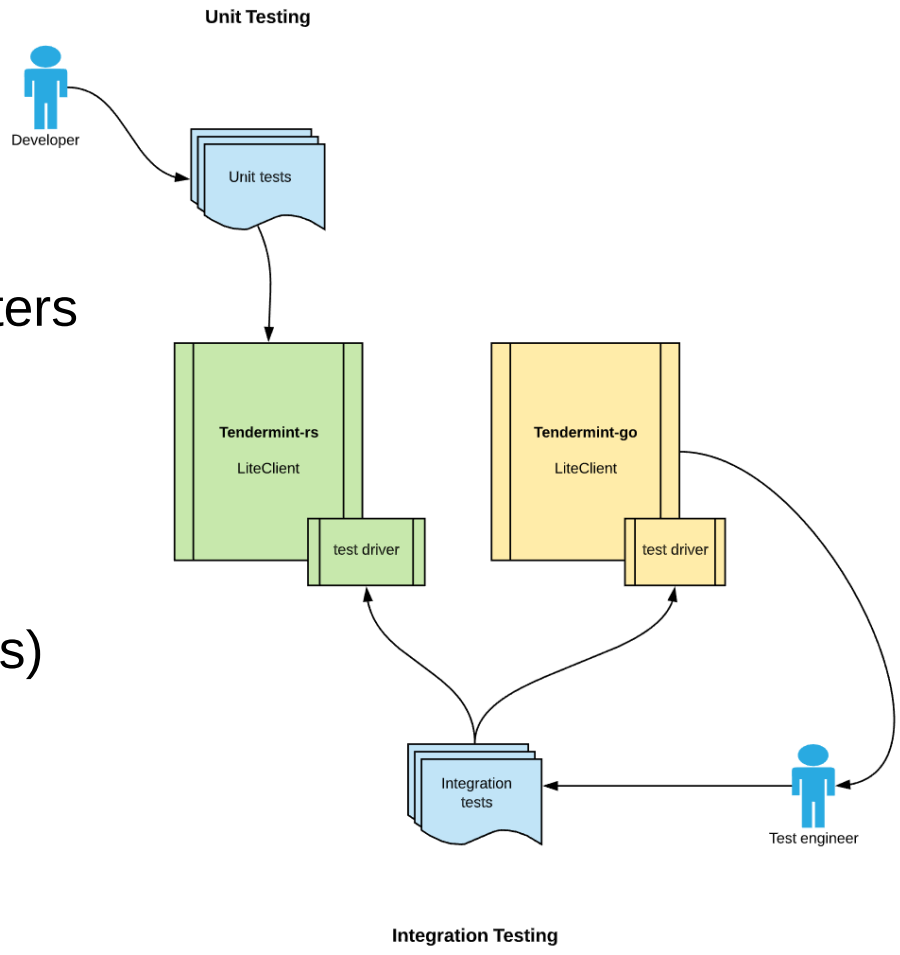
# Standard testing: test driver

```rust
fn single_step_test(tc: SingleStepTestCase) {
    let mut latest_trusted :Trusted = Trusted::new(
        signed_header: tc.initial.signed_header.clone(),
        next_validators: tc.initial.next_validator_set.clone(),
    );
    let clock_drift :Duration = Duration::from_secs( secs: 1);
    let trusting_period: Duration = tc.initial.trusting_period.into();
    for (i :usize , input :&BlockVerdict ) in tc.input.iter().enumerate() {
        let now :Time = input.now;
        match verify_single( trusted_state: latest_trusted.clone(), input: input.block.clone().into(),
            trust_threshold: TrustThreshold::default(), trusting_period, clock_drift, now) {
            Ok(new_state :LightBlock ) => {
                assert_eq!(input.verdict, LiteVerdict::Success);
                let expected_state: LightBlock = input.block.clone().into();
                assert_eq!(new_state, expected_state);
                latest_trusted = Trusted::new(new_state.signed_header, new_state.next_validators);
            }
            Err(e :Verdict ) => {
                match e {
                    Verdict::Invalid(_) => assert_eq!(input.verdict, LiteVerdict::Invalid),
                    Verdict::NotEnoughTrust(_) => {
                        assert_eq!(input.verdict, LiteVerdict::NotEnoughTrust)
                    }
                    Verdict::Success => {
                        panic!("verify_single() returned error with Verdict::Success")
                    }
```

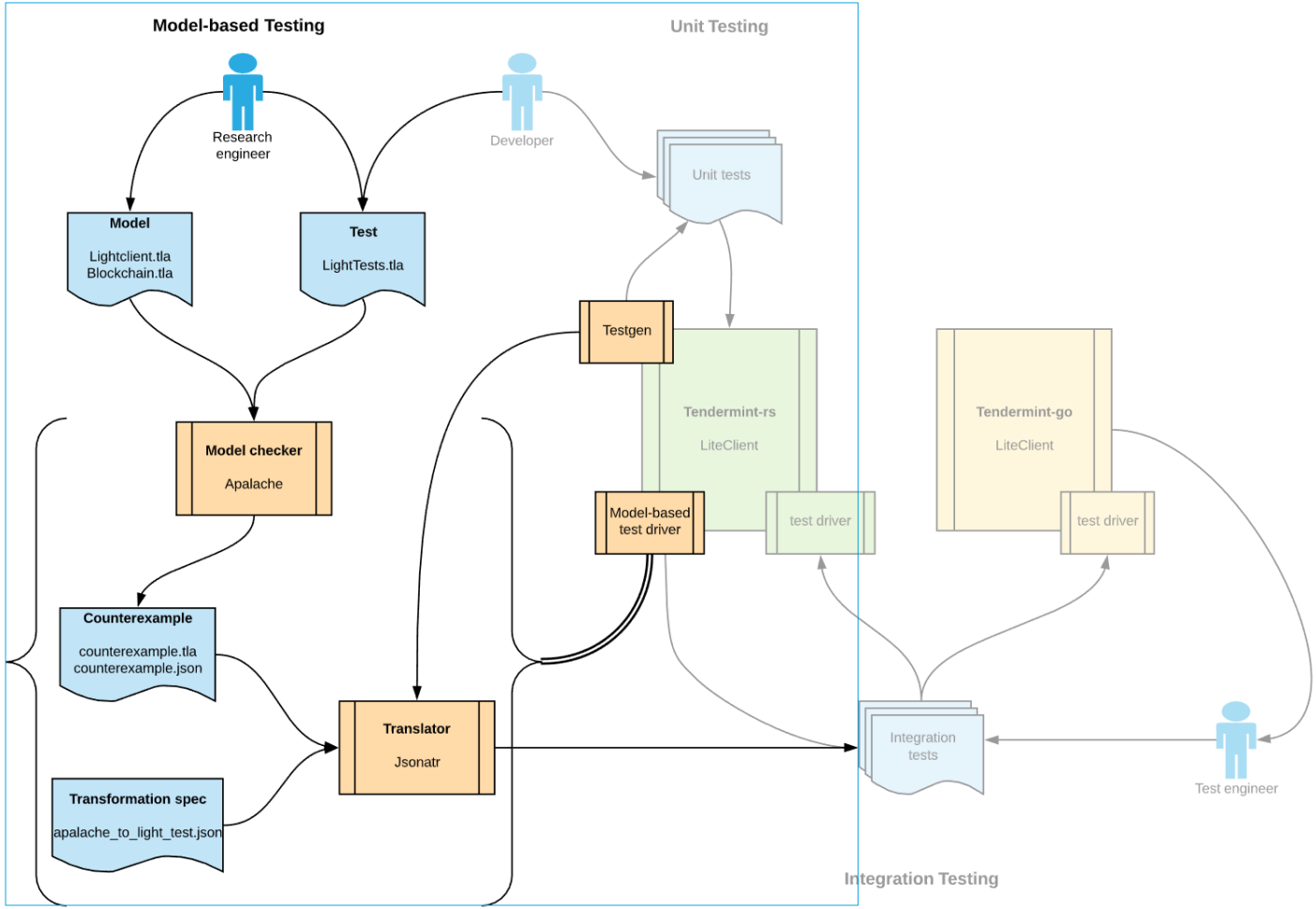
informal

# Standard testing

- Manual generation
  - lots of efforts
  - inflexible: fixes all parameters
- Hard to:
  - understand the intention
  - maintain
  - cover all cases (many 100s)



# Model-based testing



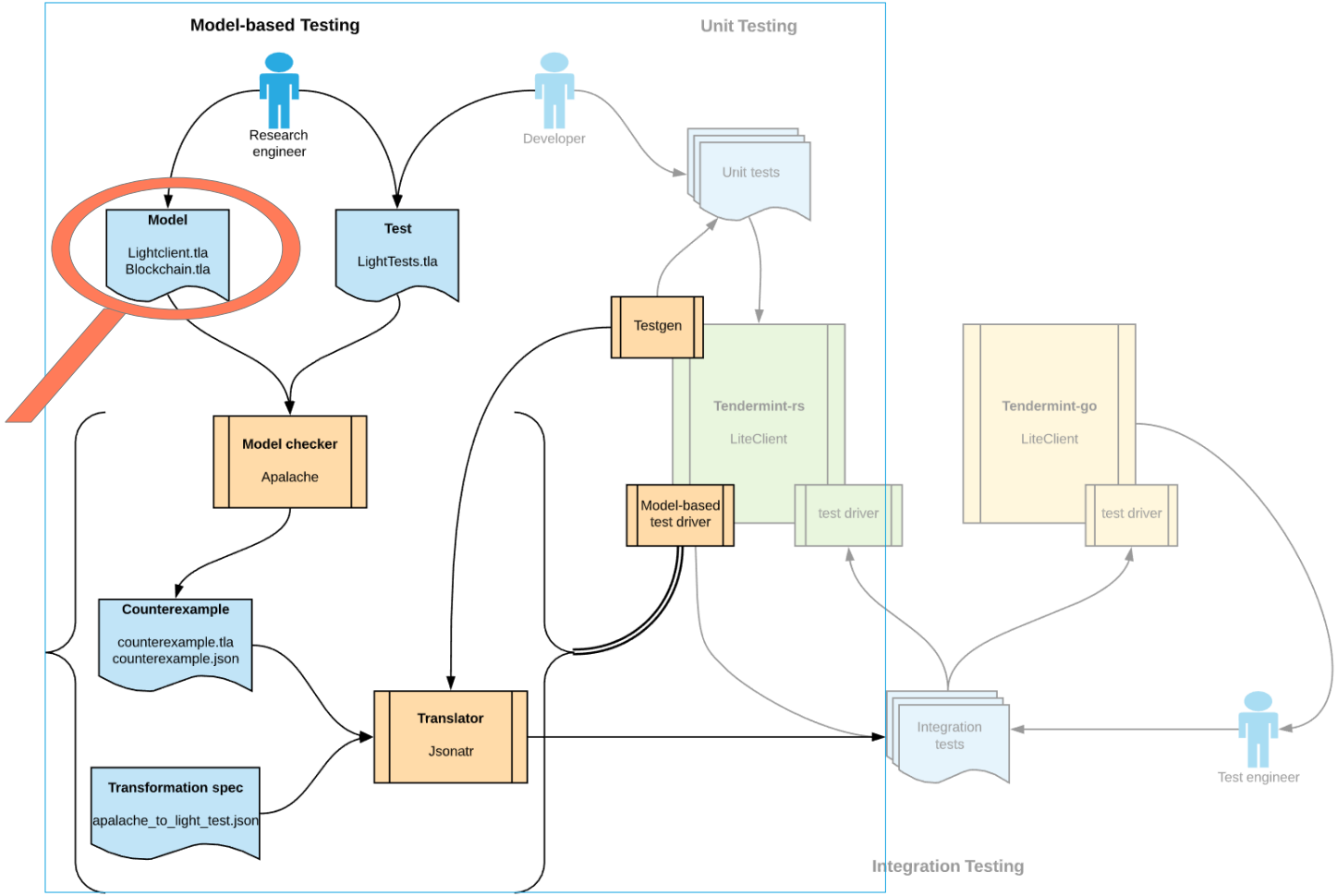

informal

# Demo: Continuous Integration

```
andrey@informal:~/work/tendermint-rs/light-client$ cargo test model_based -- --nocapture
    Finished test [unoptimized + debuginfo] target(s) in 0.07s
     Running /home/andrey/work/tendermint-rs/target/debug/deps/model_based-e07d57233ad3d7d3

running 1 test
  Running model-based single-step test: TestSuccess
  > running Apalache...
  > running static model-based single-step test: auto-generated from Apalache counterexample
    > step 0, expecting NotEnoughTrust
      > lite: NotEnoughTrust(NotEnoughTrust(VotingPowerTally { total: 150, tallied: 50, trust_threshold: TrustThresholdFraction { numerator: 1, denominator: 3 } }))
    > step 1, expecting Success
    > step 2, expecting Success
    > step 3, expecting Success
  Running model-based single-step test: TestFailure
  > running Apalache...
  > running static model-based single-step test: auto-generated from Apalache counterexample
    > step 0, expecting NotEnoughTrust
      > lite: NotEnoughTrust(NotEnoughTrust(VotingPowerTally { total: 100, tallied: 0, trust_threshold: TrustThresholdFraction { numerator: 1, denominator: 3 } }))
    > step 1, expecting Success
    > step 2, expecting Invalid
      > lite: Invalid(ImplementationSpecific("pre-commit length: 4 doesn\'t match validator length: 1"))
  Running model-based single-step test: TestValsetDifferentAllSteps
  > running Apalache...
  > running static model-based single-step test: auto-generated from Apalache counterexample
    > step 0, expecting NotEnoughTrust
      > lite: NotEnoughTrust(NotEnoughTrust(VotingPowerTally { total: 100, tallied: 0, trust_threshold: TrustThresholdFraction { numerator: 1, denominator: 3 } }))
    > step 1, expecting Success
    > step 2, expecting Success
  Running model-based single-step test: TestHeaderFromFuture
  > running Apalache...
  > running static model-based single-step test: auto-generated from Apalache counterexample
    > step 0, expecting Invalid
      > lite: Invalid(HeaderFromTheFuture { header_time: Time(1970-01-01T00:00:08Z), now: Time(1970-01-01T00:00:07Z) })
  Running model-based single-step test: TestUntrustedBeforeTrusted
  > running Apalache...
  > running static model-based single-step test: auto-generated from Apalache counterexample
    > step 0, expecting Invalid
      > lite: Invalid(NotWithinTrustPeriod { expires_at: Time(1970-01-01T00:23:21Z), now: Time(1970-01-01T00:23:24Z) })
```

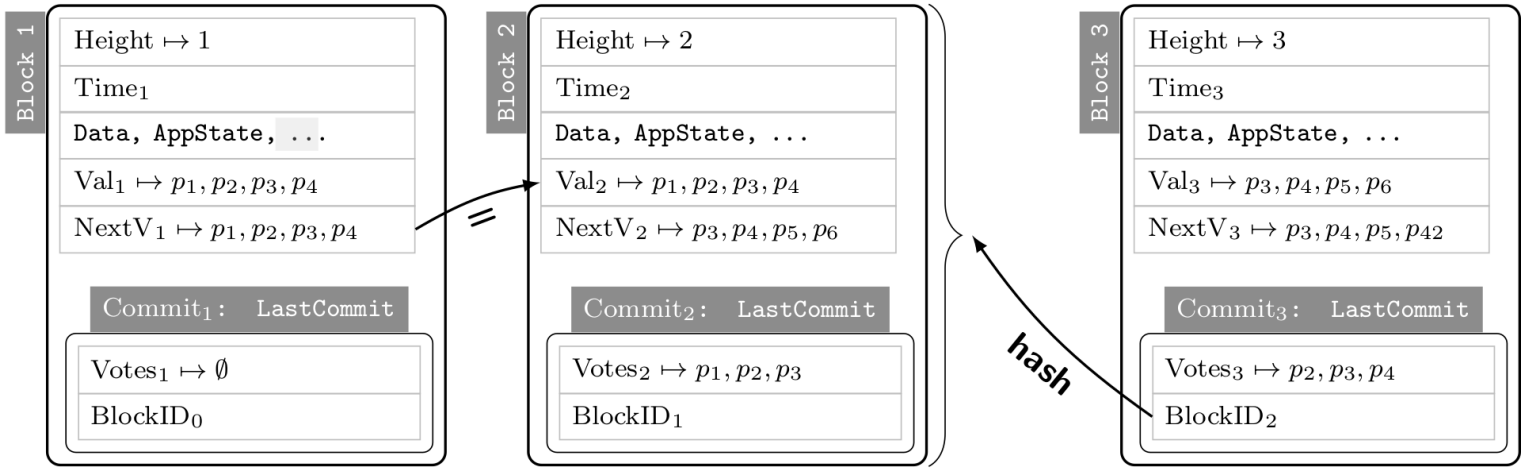
informal

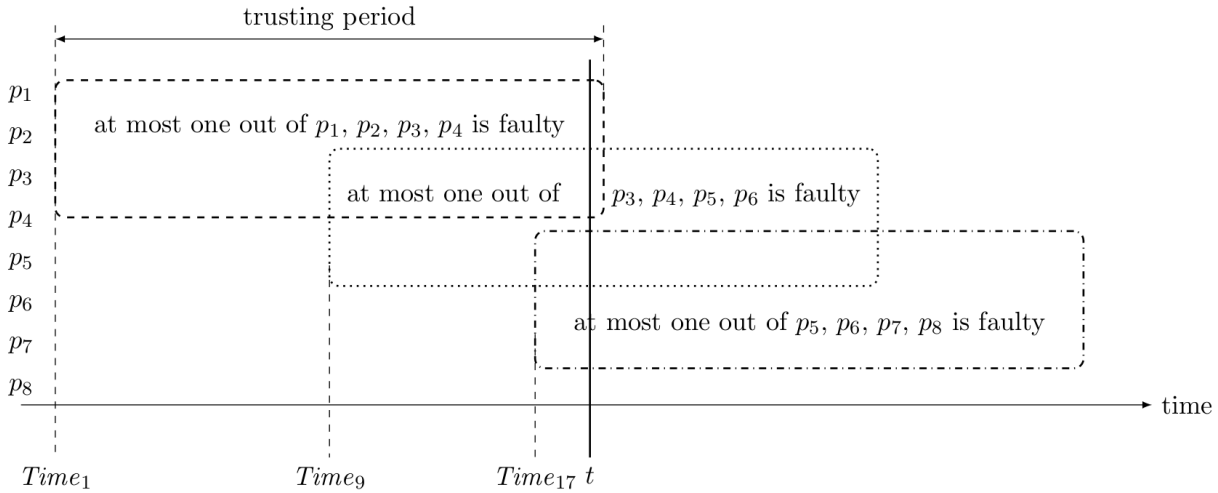
# Model: Tendermint LightClient




informal

# Tendermint Blockchain

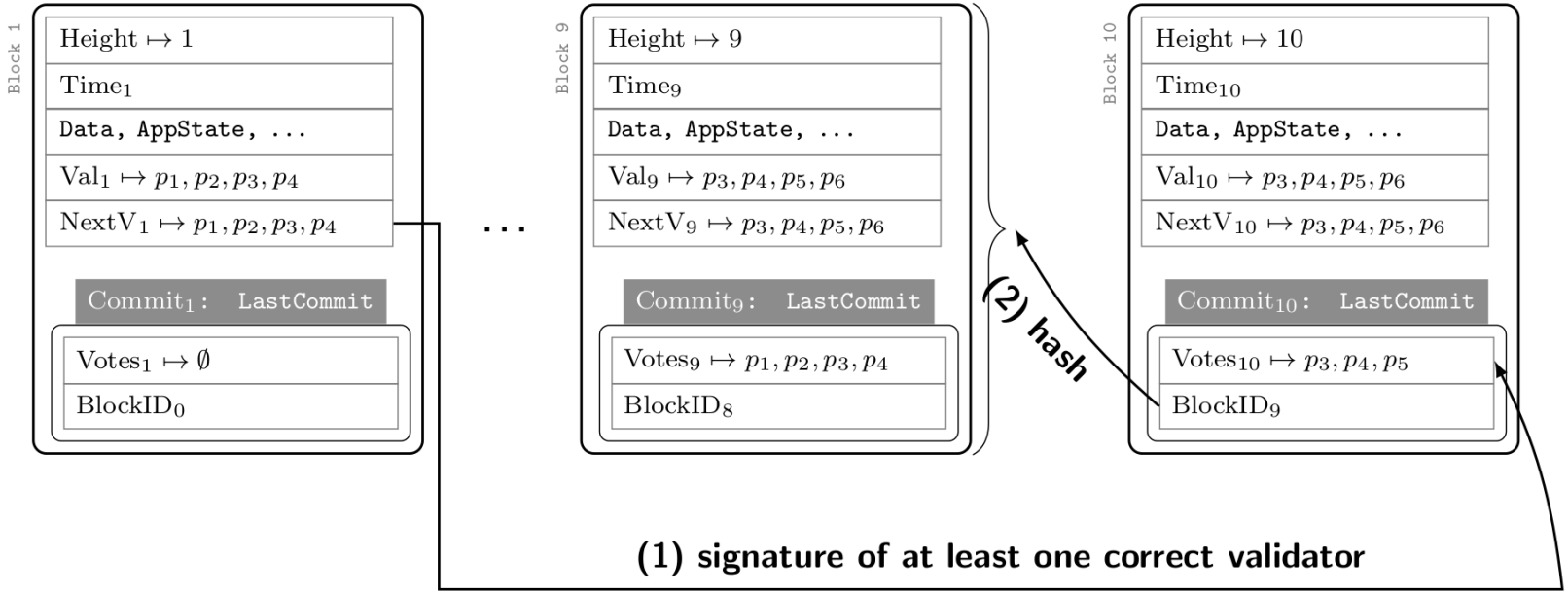


# Tendermint Blockchain

```
------------------------ MODULE Blockchain_002_draft ----------------------------
(*
  This is a high-level specification of Tendermint blockchain
  that is designed specifically for the light client.
  Validators have the voting power of one. If you like to model various
  voting powers, introduce multiple copies of the same validator
  (do not forget to give them unique names though).
 *)
EXTENDS Integers, FiniteSets

Min(a, b) == IF a < b THEN a ELSE b

CONSTANT
  AllNodes,
    (* a set of all nodes that can act as validators (correct and faulty) *)
  ULTIMATE_HEIGHT,
    (* a maximal height that can be ever reached (modelling artifact) *)
  TRUSTING_PERIOD
    (* the period within which the validators are trusted *)

Heights == 1..ULTIMATE_HEIGHT   (* possible heights *)

(* A commit is just a set of nodes who have committed the block *)
Commits == SUBSET AllNodes

(* The set of all block headers that can be on the blockchain.
   This is a simplified version of the Block data structure in the actual implementation. *)
BlockHeaders == [
  height: Heights,
    \* the block height
  time: Int,
    \* the block timestamp in some integer units
  lastCommit: Commits,
    \* the nodes who have voted on the previous block, the set itself instead of a hash
  (* in the implementation, only the hashes of V and NextV are stored in a block,
     as V and NextV are stored in the application state *)
  VS: SUBSET AllNodes,
    \* the validators of this bloc. We store the validators instead of the hash.
  NextVS: SUBSET AllNodes
    \* the validators of the next block. We store the next validators instead of the hash.
]

(* A signed header is just a header together with a set of commits *)
LightBlocks == [header: BlockHeaders, Commits: Commits]
```

# Tendermint Light Client: skipping verification




Details: A Tendermint Light Client **arxiv.org/abs/2010.07031**

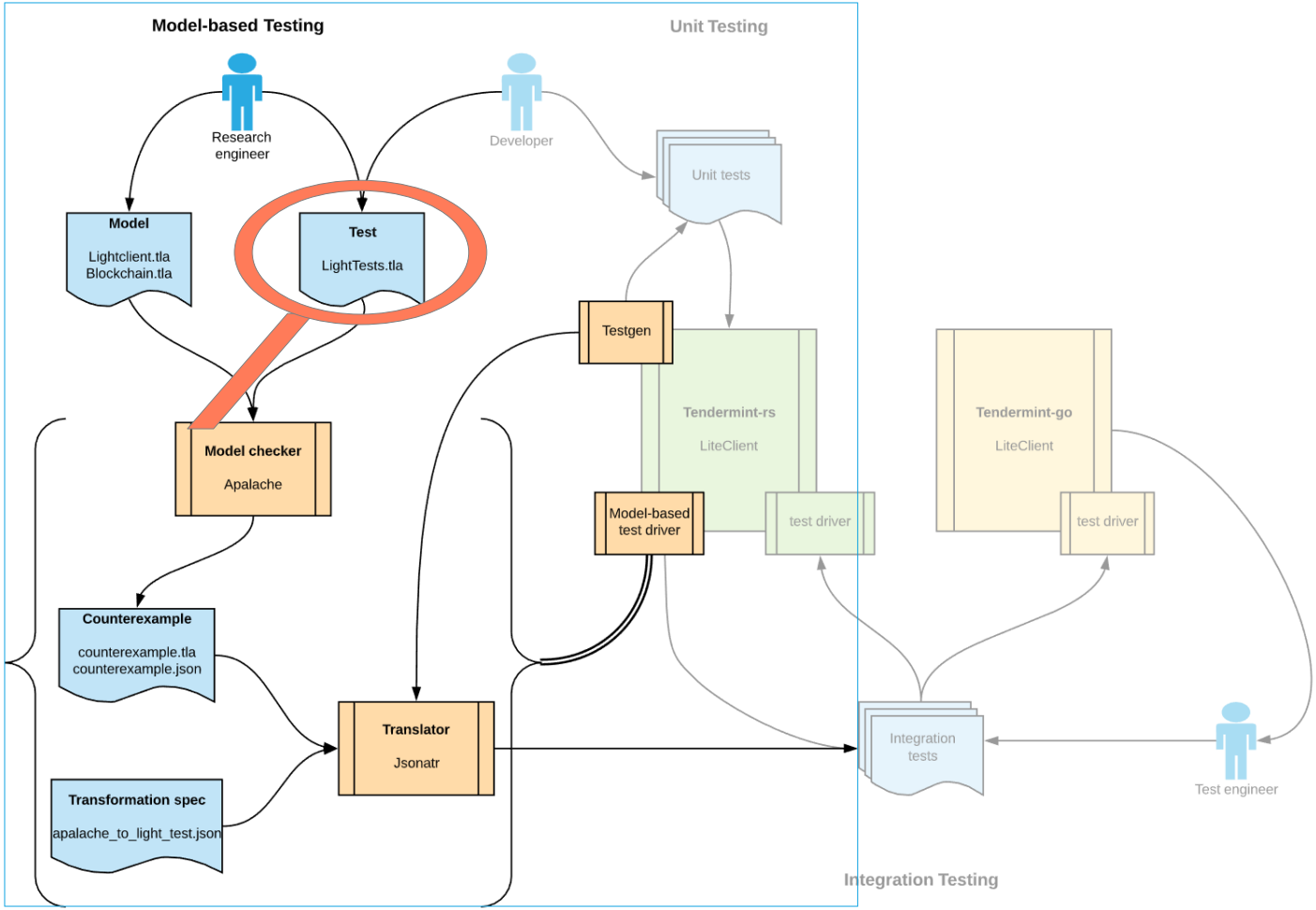
informal

# Tendermint Light Client

```
---------------------------- MODULE Lightclient_002_draft ----------------------------
(**
 * A state-machine specification of the lite client, following the English spec:
 *
 * https://github.com/tendermint/spec/blob/master/rust-spec/lightclient/verification/verification.md
 *)

EXTENDS Integers, FiniteSets

\* the parameters of Light Client
CONSTANTS
  TRUSTED_HEIGHT,
    (* an index of the block header that the light client trusts by social consensus *)
  TARGET_HEIGHT,
    (* an index of the block header that the light client tries to verify *)
  TRUSTING_PERIOD,
    (* the period within which the validators are trusted *)
  IS_PRIMARY_CORRECT
    (* is primary correct? *)

VARIABLES       (* see TypeOK below for the variable types *)
  state,        (* the current state of the light client *)
  nextHeight,   (* the next height to explore by the light client *)
  nprobes       (* the lite client iteration, or the number of block tests *)

(* the light store *)
VARIABLES
  fetchedLightBlocks, (* a function from heights to LightBlocks *)
  lightBlockStatus,   (* a function from heights to block statuses *)
  latestVerified      (* the latest verified block *)

(* the variables of the lite client *)
lcvars == <<state, nextHeight, fetchedLightBlocks, lightBlockStatus, latestVerified>>

(* the light client previous state components, used for monitoring *)
VARIABLES
  prevVerified,
  prevCurrent,
  prevNow,
  prevVerdict
```

informal

# TLA+ tests




informal

# TLA+ tests

```
\* Test an execution that finishes with failure
TestFailure ==
    /\ state = "finishedFailure"
    /\ Cardinality(DOMAIN fetchedLightBlocks) = TARGET_HEIGHT

\* Test an execution that finishes with success
TestSuccess ==
    /\ state = "finishedSuccess"
    /\ Cardinality(DOMAIN fetchedLightBlocks) = TARGET_HEIGHT

\* Test an execution where the validator sets differ at each step
TestValsetDifferentAllSteps ==
    /\ Cardinality(DOMAIN fetchedLightBlocks) = TARGET_HEIGHT
    /\ \A s1, s2 \in DOMAIN history :
       s1 /= s2  =>
       history[s1].current.header.VS /= history[s2].current.header.VS

\* Test an execution where a header is received from the future
TestHeaderFromFuture ==
    /\ \E s \in DOMAIN history :
       history[s].now < history[s].current.header.time

\* Test an execution where the untrusted header time is before the trusted header time
TestUntrustedBeforeTrusted ==
    /\ \E s \in DOMAIN history :
       history[s].current.header.time < history[s].verified.header.time
```

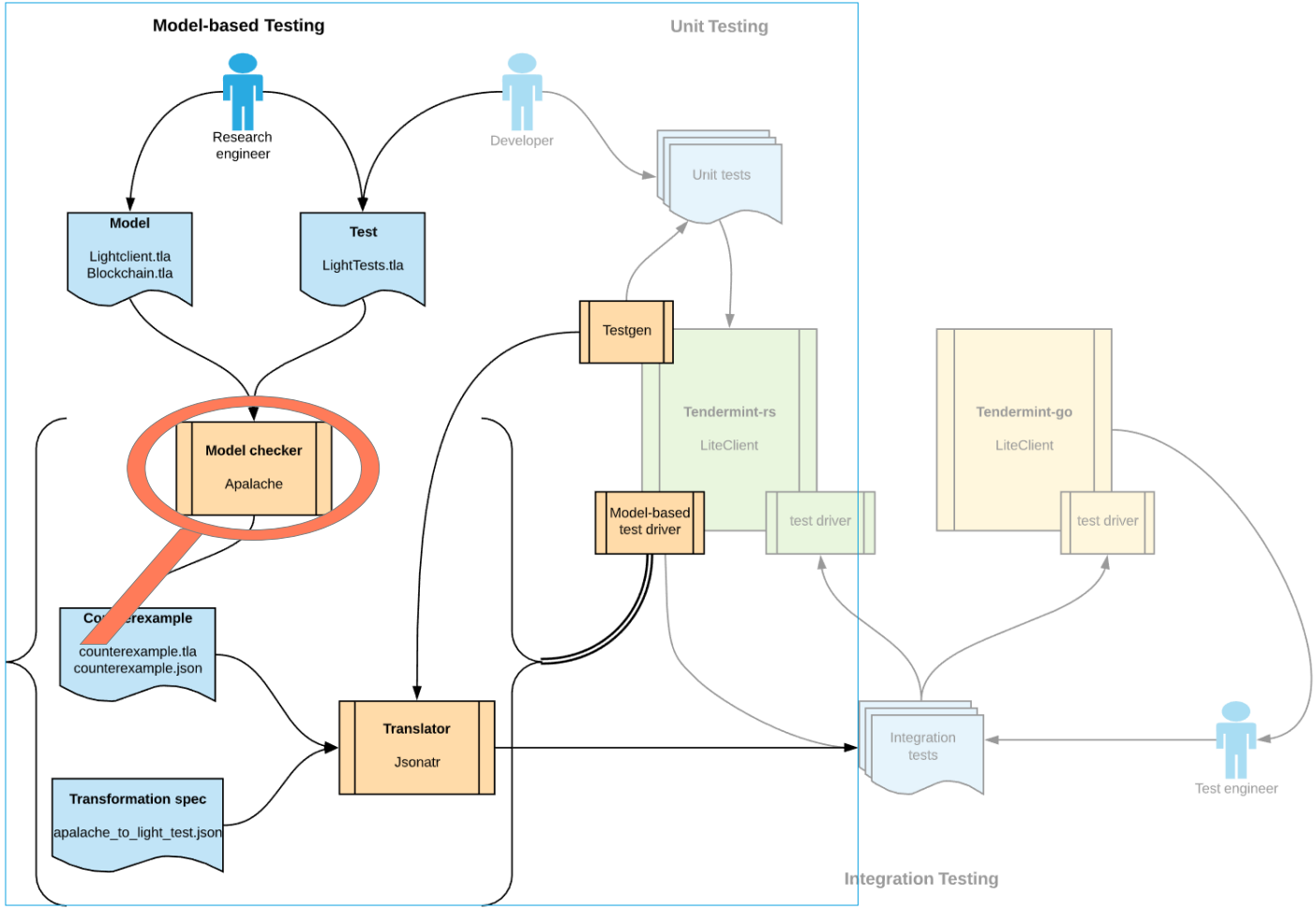
informal

# Modular extension of the TLA+ model

```
------------------------- MODULE LightTests --------------------------

EXTENDS Lightclient

(* The light client history, which is the function mapping states 0..nprobes
   to the record with fields:
   - verified: the latest verified block in the previous state
   - current: the block that is being checked in the previous state
   - now: the time point in the previous state
   - verdict: the light client verdict in the previous state
*)
VARIABLE
  history

\* This predicate extends the LightClient Init predicate with history tracking
InitTest ==
  /\ Init
  /\ history = [ n \in {0} |->
     [ verified |-> prevVerified, current |-> prevCurrent, now |-> prevNow, verdict |-> prevVerdict ]]

\* This predicate extends the LightClient Next predicate with history tracking
NextTest ==
  /\ Next
  /\ history' = [ n \in DOMAIN history \union {nprobes'} |->
       IF n = nprobes' THEN
         [ verified |-> prevVerified', current |-> prevCurrent', now |-> prevNow', verdict |-> prevVerdict' ]
       ELSE history[n]
     ]
```

informal

# Model-checker: Apalache



# Model checker: Apalache

apalache/README.md at unst

github.com/informalsystems/apalache/blob/unstable/README.md

## APALACHE

A symbolic model checker for TLA+

| master | unstable |
| --- | --- |
| build passing | build passing |

Apalache translates TLA+ in the logic supported by the SMT solvers, for instance, Microsoft Z3. Apalache can check inductive invariants (for fixed or bounded parameters) and check safety of bounded executions (bounded model checking). To see the list of supported TLA+ constructs, check the supported features. In general, Apalache runs under the same assumptions as TLC.

### Releases

Check the releases page.

We recommend you to run the latest docker image `apalache/mc:latest` and checkout the source code from master, which accumulate bugfixes over the latest release, see the manual. To try the latest cool features, check the unstable branch.

### Getting started

Read the user manual.

### Talks

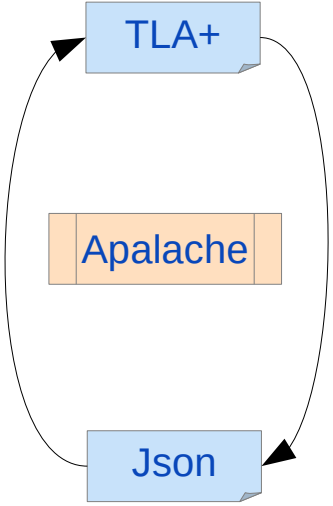
- Model-based testing with TLA+ and Apalache. TLA+ Community Event 2020 (October 2020).
- Type inference for TLA+ in Apalache. TLA+ Community Event 2020 (October 2020).
- Formal Spec and Model Checking of the Tendermint Blockchain Synchronization Protocol 2nd Workshop on Formal Methods for Blockchains (July 2020).

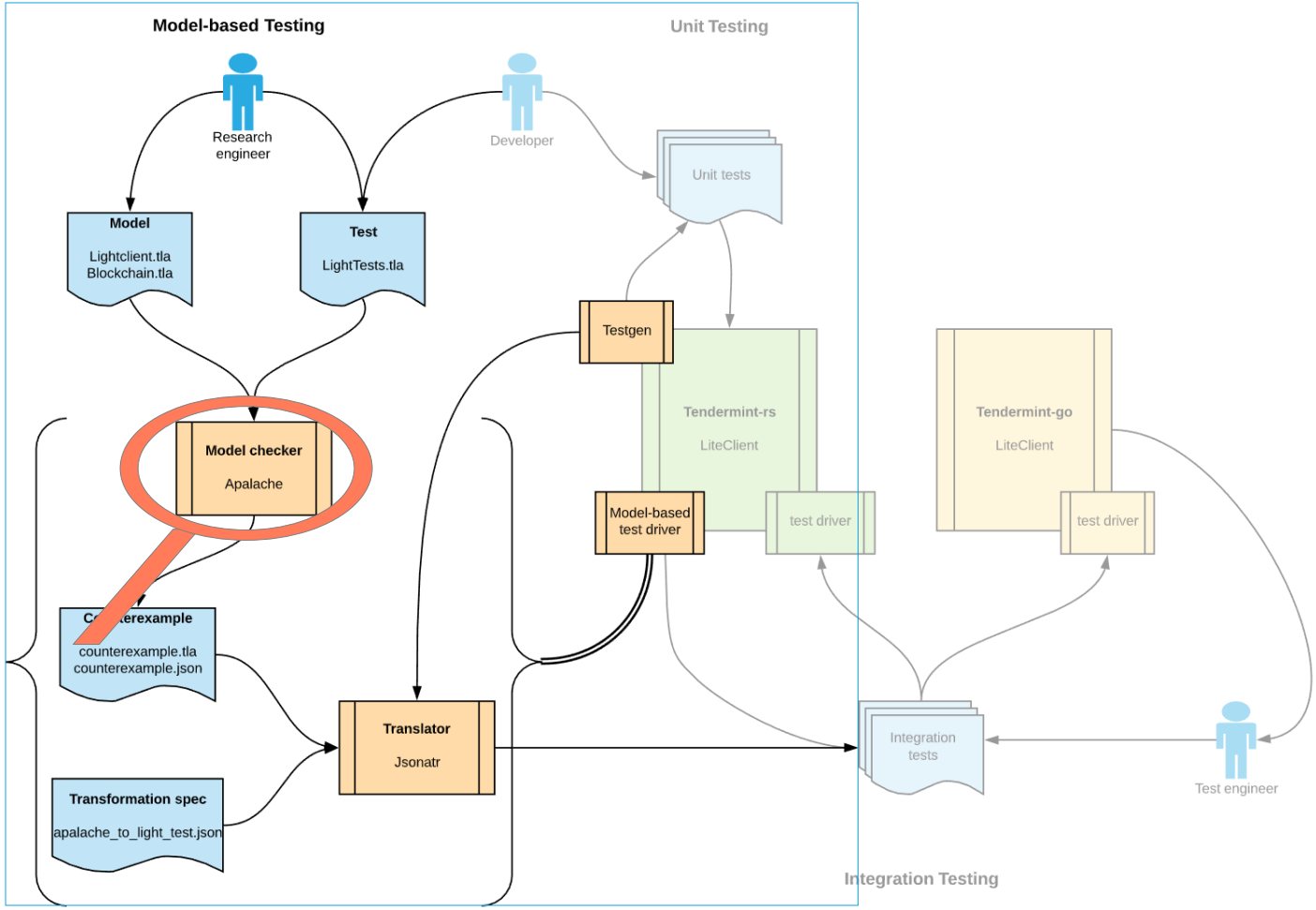
informal

# Apalache demo

```
# APALACHE version 0.7.3-SNAPSHOT build v0.7.0-62-gb643cc4
#
# WARNING: This tool is in the experimental stage.
#          Please report bugs at: [https://github.com/informalsystems/apalache/issues]

Checker options: filename=MC.tla, init=, next=, inv=                I@16:21:09.892
PASS #0: SanyParser                                                 I@16:21:10.302
Parsing file /home/andrey/work/tendermint-rs/docs/spec/lightclient/verification/MC5_5_faulty.toolbox/Model_1/MC.tla
Parsing file /home/andrey/work/tendermint-rs/docs/spec/lightclient/verification/MC5_5_faulty.toolbox/Model_1/MC5_5_faulty.tla
Parsing file /tmp/TLC.tla
Parsing file /home/andrey/work/tendermint-rs/docs/spec/lightclient/verification/MC5_5_faulty.toolbox/Model_1/Lightclient_A_1.tla
Parsing file /tmp/Integers.tla
Parsing file /tmp/FiniteSets.tla
Parsing file /tmp/Naturals.tla
Parsing file /home/andrey/work/tendermint-rs/docs/spec/lightclient/verification/MC5_5_faulty.toolbox/Model_1/Blockchain_A_1.tla
Parsing file /tmp/Sequences.tla
PASS #1: ConfigurationPass                                          I@16:21:11.156
  > Loading TLC configuration from MC.cfg                           I@16:21:11.157
PASS #2: UnrollPass                                                 I@16:21:11.341
  > Unroller                                                        I@16:21:11.344
PASS #3: InlinePass                                                 I@16:21:11.428
  > InlinerOfUserOper                                               I@16:21:11.429
  > LetInExpander                                                   I@16:21:11.460
PASS #4: PrimingPass                                                I@16:21:11.548
  > Introducing InitPrimed for Init'                                I@16:21:11.551
PASS #5: VCGen                                                      I@16:21:11.646
  > Producing verification conditions from the invariant PrecisionBuggyInv I@16:21:11.647
  > VCGen produced 1 verification condition(s)                      I@16:21:11.648
PASS #6: PreprocessingPass                                          I@16:21:11.736
  > Before preprocessing: unique renaming                           I@16:21:11.736
 > Applying standard transformations:                               I@16:21:11.744
  > Desugarer                                                       I@16:21:11.744
  > UniqueRenamer                                                   I@16:21:11.770
  > Normalizer                                                      I@16:21:11.869
  > Keramelizer                                                     I@16:21:11.880
  > After preprocessing: UniqueRenamer                              I@16:21:11.967
PASS #7: TransitionFinderPass                                       I@16:21:12.026
  > Found 1 initializing transitions                                I@16:21:12.354
  > Found 7 transitions                                             I@16:21:12.470
  > No constant initializer                                         I@16:21:12.470
  > Applying unique renaming                                        I@16:21:12.471
PASS #8: OptimizationPass                                           I@16:21:12.521
 > Applying optimizations:                                          I@16:21:12.525
  > ConstSimplifier                                                 I@16:21:12.526
  > ExprOptimizer                                                   I@16:21:12.543
  > ConstSimplifier                                                 I@16:21:12.578
PASS #9: AnalysisPass                                               I@16:21:12.622
 > Marking skolemizable existentials and sets to be expanded...     I@16:21:12.624
  > SkolemizationMarker                                             I@16:21:12.624
  > ExpansionMarker                                                 I@16:21:12.631
 > Running analyzers...                                             I@16:21:12.649
  > Introduced expression grades                                    I@16:21:12.682
  > Introduced 20 formula hints                                     I@16:21:12.682
PASS #10: BoundedChecker                                            I@16:21:12.683
No CONSTANT initializer given                                       I@16:21:12.746
Step 0, level 0: checking if 1 transition(s) are enabled and violate the invariant I@16:21:12.750
Step 0, level 1: collecting 1 enabled transition(s)                 I@16:21:13.921
Step 1, level 1: checking if 7 transition(s) are enabled and violate the invariant I@16:21:14.637
Invariant is violated at depth 1. Check the counterexample in any of counterexample.tla, MC.out, counterexample.json E@16:21:15.477
The outcome is: Error                                               I@16:21:15.491
Checker has found an error                                          I@16:21:15.493
It took me 0 days  0 hours  0 min  5 sec                            I@16:21:15.494
Total time: 5.654 sec                                               I@16:21:15.495
EXITCODE: OK
```

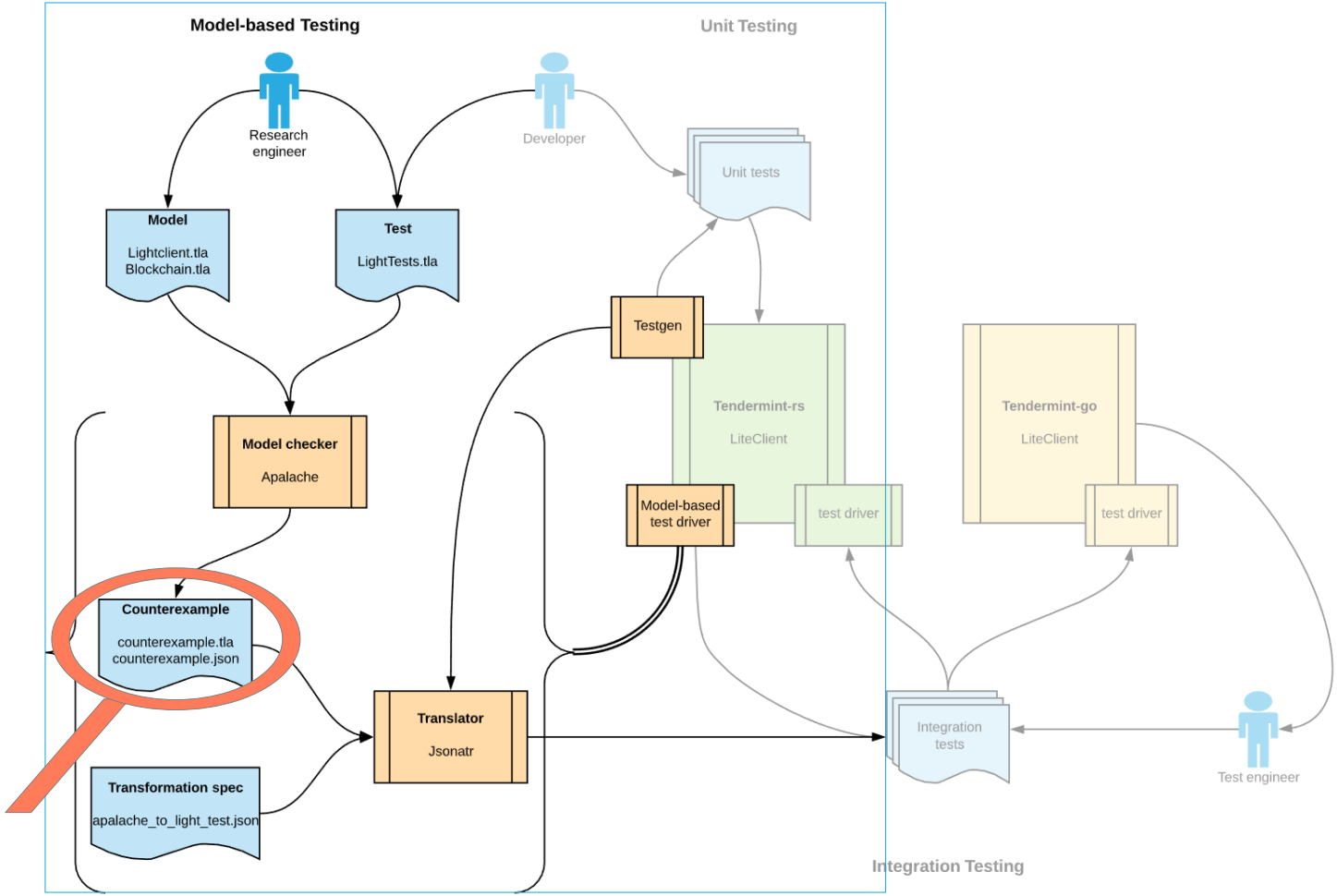
# Apalache extension: TLA+ ↔ Json




informal

# Counterexamples as tests



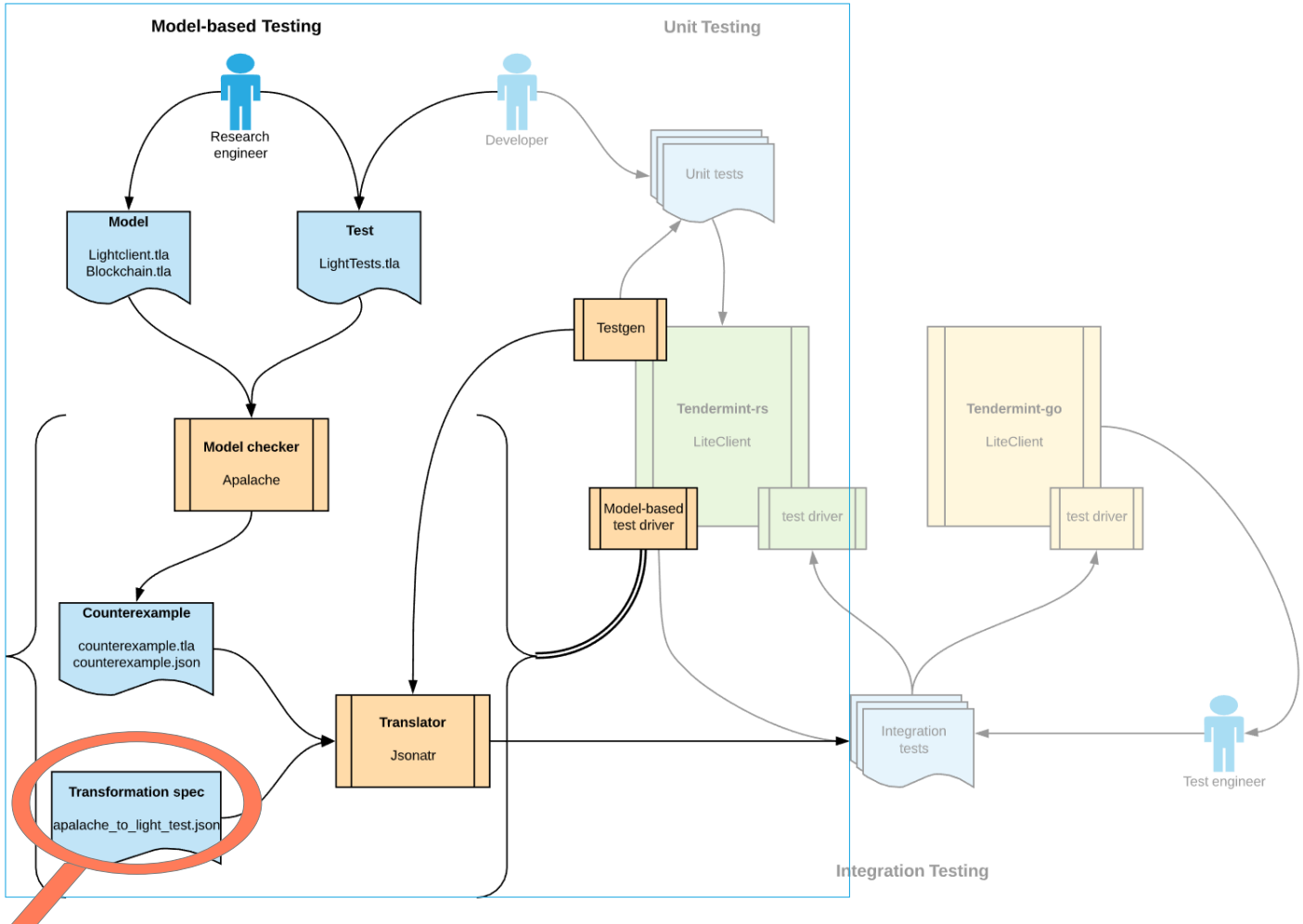
# Counterexamples as tests

```
history =
  0 :> ...
  @@ 1
    :> [current |->
        [Commits |-> {"n2"},
          header |->
            [NextVS |-> { "n1", "n3", "n4" },
              VS |-> {"n2"},
              height |-> 4,
              lastCommit |-> {"n4"},
              time |-> 5]],
      now |-> 1398,
      verdict |-> "NOT_ENOUGH_TRUST",
      verified |->
        [Commits |-> { "n1", "n2", "n3", "n4" },
          header |->
            [NextVS |-> { "n1", "n2", "n3" },
              VS |-> { "n1", "n2", "n3", "n4" },
              height |-> 1,
              lastCommit |-> {},
              time |-> 1]]]
  @@ 2
    :> [current |->
        [Commits |-> { "n1", "n2", "n3" },
          header |->
            [NextVS |-> {"n4"},
              VS |-> { "n1", "n2", "n3" },
              height |-> 2,
              lastCommit |-> { "n2", "n3", "n4" },
              time |-> 3]],
      now |-> 1400,
      verdict |-> "SUCCESS",
      verified |->
        [Commits |-> { "n1", "n2", "n3", "n4" },
          header |->
            [NextVS |-> { "n1", "n2", "n3" },
              VS |-> { "n1", "n2", "n3", "n4" },
              height |-> 1,
              lastCommit |-> {},
              time |-> 1]]]
  @@ 3
    :> ...
```


```
{ "key": { "str": "header" },
  "value": {
    "record": [
      { "key": { "str": "NextVS" },
        "value": {
          "enumSet": [
            { "str": "n4" }
          ]}},
      { "key": { "str": "VS" },
        "value": {
          "enumSet": [
            { "str": "n1" },
            { "str": "n2" },
            { "str": "n3" }
          ]}},
      { "key": { "str": "height" },
        "value": 2
      },
      { "key": { "str": "lastCommit" },
        "value": {
          "enumSet": [
            { "str": "n2" },
            { "str": "n3" },
            { "str": "n4" }
          ]}},
      { "key": { "str": "time" },
        "value": 3
      }
    ]
  }
}
```


informal

# Counterexamples as tests




informal

# Counterexample transformation

```
"input": [
  {
    "name": "block_to_initial_block",
    "description": "transforms a block from Apalache CE into a JSON-encoded Tendermint initial light block",
    "kind": "INLINE",
    "source": {
      "signed_header": "$ | block_to_signed_header",
      "next_validator_set": "$ | block_next_validators | ids_to_validator_set",
      "trusting_period": "1400000000000",
      "now": "$utc_timestamp"
    }
  },
  {
    "name": "state_to_lite_block_verdict",
    "description": "transforms a block from Apalache CE into a JSON-encoded Tendermint light block",
    "kind": "INLINE",
    "let": {
      "block": "$..[?(@.key.str == 'current')].value"
    },
    "source": {
      "block": {
        "signed_header": "$block | block_to_signed_header",
        "validator_set": "$block | block_validators | ids_to_validator_set",
        "next_validator_set": "$block | block_next_validators | ids_to_validator_set"
      },
      "now": "$..[?(@.key.str == 'now')].value | unwrap | tendermint_time",
      "verdict": "$..[?(@.key.str == 'verdict')].value.str | unwrap"
    }
  }
],
"output": {
  "description": "auto-generated from Apalache counterexample",
  "initial": "$history[0]..[?(@.key.str == 'current')].value | block_to_initial_block",
  "input": "$history[1:] | map(state_to_lite_block_verdict)"
}
```

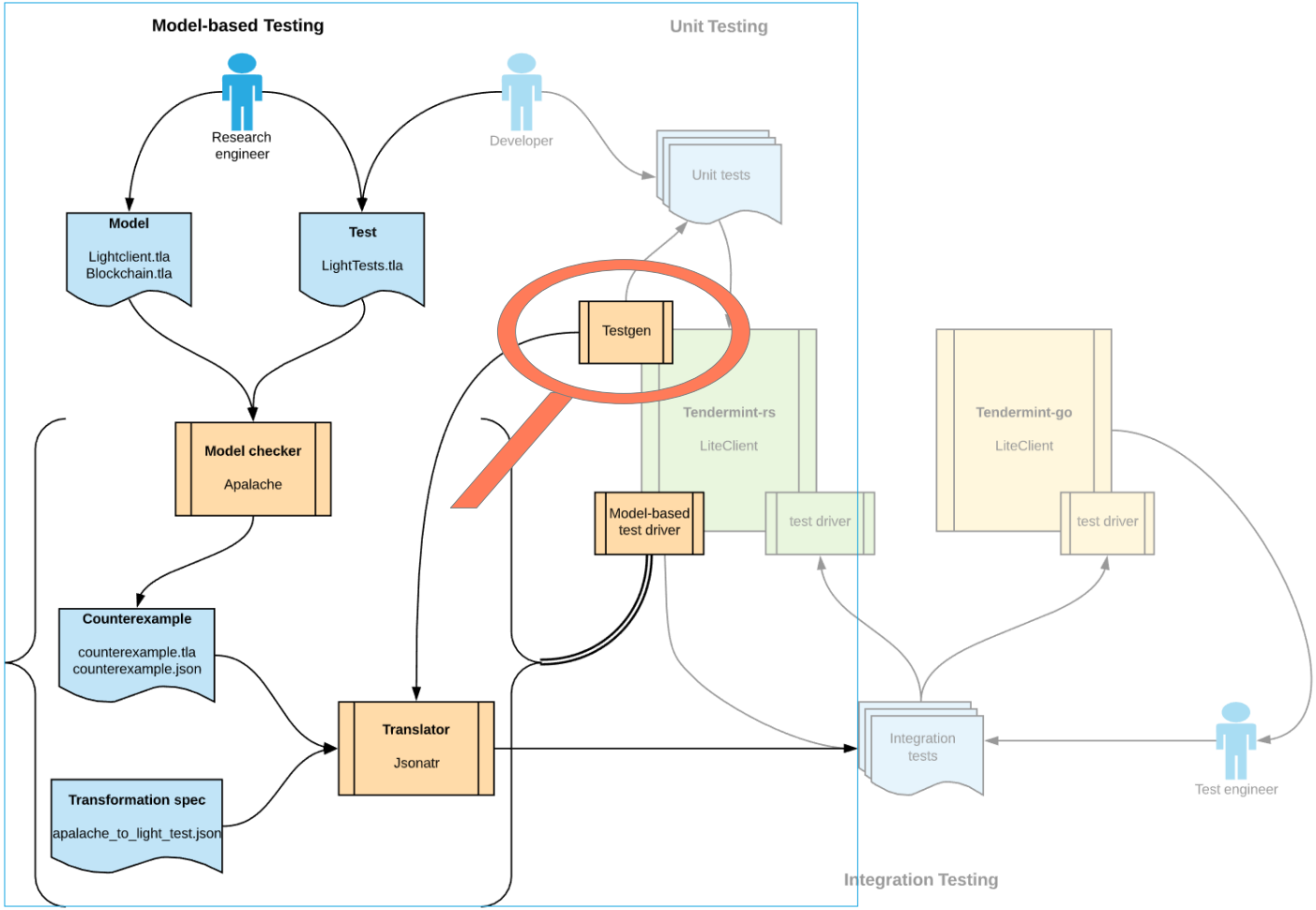
informal

# Counterexample transformation

```
{
  "description": "Transformers for generating Tendermint datastructures",
  "prerequisites": "add tendermint-testgen to your $PATH",
  "input": [
    {
      "name": "tendermint_validator",
      "kind": "COMMAND",
      "source": "tendermint-testgen --stdin validator"
    },
    {
      "name": "tendermint_header",
      "kind": "COMMAND",
      "source": "tendermint-testgen --stdin header"
    },
    {
      "name": "tendermint_commit",
      "kind": "COMMAND",
      "source": "tendermint-testgen --stdin commit"
    },
    {
      "name": "tendermint_vote",
      "kind": "COMMAND",
      "source": "tendermint-testgen --stdin vote"
    },
    {
      "name": "tendermint_time",
      "kind": "COMMAND",
      "source": "tendermint-testgen --stdin time"
    }
  ]
}
```

informal

# Testgen: from abstract to concrete



# Testgen: from abstract to concrete

Abstract: TLA+ → Testgen → Concrete: Rust

"n1" →

```
"validator": {
  "address": "6AE5C701F508EB5B63343858E068C5843F28105F",
  "pub_key": {
    "type": "tendermint/PubKeyEd25519",
    "value": "GQEC/HB4sDBAVhHtUzyv4yct9ZGnudaP209QQBSTfSQ="
  },
  "voting_power": "50",
  "proposer_priority": null
}
```

```
header |->
  [NextVS |-> {},
    VS |-> {"n4"},
    height |-> 4,
    lastCommit |-> "n1", "n2", "n3" },
    time |-> 1402]
```

→

```
"header": {
  "version": {
    "block": "0",
    "app": "0"
  },
  "chain_id": "test-chain",
  "height": "4",
  "time": "1970-01-01T00:23:22Z",
  "last_block_id": null,
  "last_commit_hash": A1227FA5ABACADB137F59906C4A604E806190C9991D6D965C50BED0D28CA8375,
  "data_hash": null,
  "validators_hash":  "C8F8530F1A2E69409F2E0B4F86BB568695BC9790BA77EAC1505600D5506E22DA",
  "next_validators_hash": "E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855",
  "consensus_hash": "C8F8530F1A2E69409F2E0B4F86BB568695BC9790BA77EAC1505600D5506E22DA",
  "app_hash": "",
  "last_results_hash": null,
  "evidence_hash": null,
  "proposer_address": "0616A636E7D0579A632EC37ED3C3F2B7E8522A0A"
}
```

informal

# MBT eliminates the divergence between the specs and the code

docs/spec/lightclient/verification/Blockchain_A_1.tla Outdated

```diff
@@ -75,7 +75,7 @@ LBT == [header |-> BT, Commits |-> {NT}]

  (* the header is still within the trusting period *)
  InTrustingPeriod(header) ==
-     now <= header.time + TRUSTING_PERIOD
+     now < header.time + TRUSTING_PERIOD
```

**shonfeder** 21 days ago Member

I see the `<=` actually appears to contradict the english spec which says

> - *LightStore* always contains a verified header whose age is less than the trusting period,

informal

# MBT eliminates the divergence between the specs and the code

docs/spec/lightclient/verification/Blockchain_A_1.tla Outdated

```diff
@@ -110,7 +110,7 @@ FaultAssumption(pFaultyNodes, pNow, pBlockchain) ==
  (* Can a block be produced by a correct peer, or an authenticated Byzantine peer *)
  IsLightBlockAllowedByDigitalSignatures(ht, block) ==
      \/ block.header = blockchain[ht] \* signed by correct and faulty (maybe)
-     \/ block.Commits \subseteq Faulty /\ block.header.height = ht \* signed only by faulty
+     \/ block.Commits \subseteq Faulty /\ block.header.height = ht /\ block.header.time > 0 \*
```

docs/spec/lightclient/verification/Lightclient_A_1.tla Outdated

```diff
-   /\ thdr.time <= uhdr.time
+   /\ thdr.time < uhdr.time
+      \* the untrusted block is not from the future
+   /\ uhdr.time <= now
```

informal

# MBT improves code coverage

## #551 Model-based tests for LightClient

Merged andrey-kuprianov

44.6% 100.0% (+1.5%)

Overview | Diff | Coverage Changes 13 | Files | Commits

/ light-client / src

| Files | | | | | Coverage |
|---|---|---|---|---|---|
| predicates.rs | ø | +70 | ø | -70 | +61.4% 85.9% |
| operations | ø | +34 | -1 | -33 | +19.6% 84.3% |
| tests.rs | ø | +4 | ø | -4 | +2.8% 69.5% |
| predicates/errors.rs | ø | +3 | ø | -3 | +7.3% 19.5% |
| components | ø | +2 | ø | -2 | +1.4% 23.9% |

/ light-client / src / operations

| Files | | | | | Coverage |
|---|---|---|---|---|---|
| hasher.rs | ø | +3 | ø | -3 | +33.3% 88.8% |
| commit_validator.rs | ø | +14 | ø | -14 | +46.6% 70.0% |
| voting_power.rs | ø | +17 | -1 | -16 | +12.6% 87.3% |

informal

# MBT finds real bugs

## Validators are wrongly sorted for v.0.34 #579

Closed **andrey-kuprianov** opened this issue 8 days ago · 0 comments

**andrey-kuprianov** commented 8 days ago • edited Member

There is an issue with the current sorting of validators which I've discovered because my MBT tests were failing:

In v.0.34 the validators should be first sorted by voting power (descending), then by address (ascending), but here they are sorted only by voting power (see #506). As a result, header hash validation fails.

The issue is actually also that it's not properly documented in the tendermint/spec repo and in the tendermint-rpc, the requirement is only implemented in tendermint-go.

I've already stumbled upon this before, but forgot to open the issues against the documentation repos (my bad!); will do that now. Will also fix the issue in this repo.

👍 1 🎉 1 ❤️ 1 🚀 1

informal

# Conclusion

- MBT significantly improves
  - ease of writing / using the tests
  - tests maintainability
  - code coverage
- MBT allows to keep specifications and code synchronized
- The benefits substantially outweigh infrastructure investments

# Work in progress

- Fuzzing
  - additional mutation at the level of datastructures
  - allows to cover scenarios inexpressible in the abstract model
  - already helped us to discover some bugs
- Extension to Tendermint-go
  - use auto-generated test files to test the Tendermint in Go LightClient
  - required to write a simple test driver in Go; close to be finished
- Extensions to IBC-rs and IBC-go
  - devised a concrete plan on replacing hand-written tests with model-based ones
  - the development team is very enthusiastic

# Future work

- Apalache extensions
  - continuation of  search to enumerate all counterexamples
  - alternative approaches for faster counterexample search
- Executable TLA+
  - code generation from TLA+; will allow to substantially speed up testing
- Distributed testing
  - cut the system at the interface points
  - replace some modules with executable specs

informal