# Using formalism to design secure systems

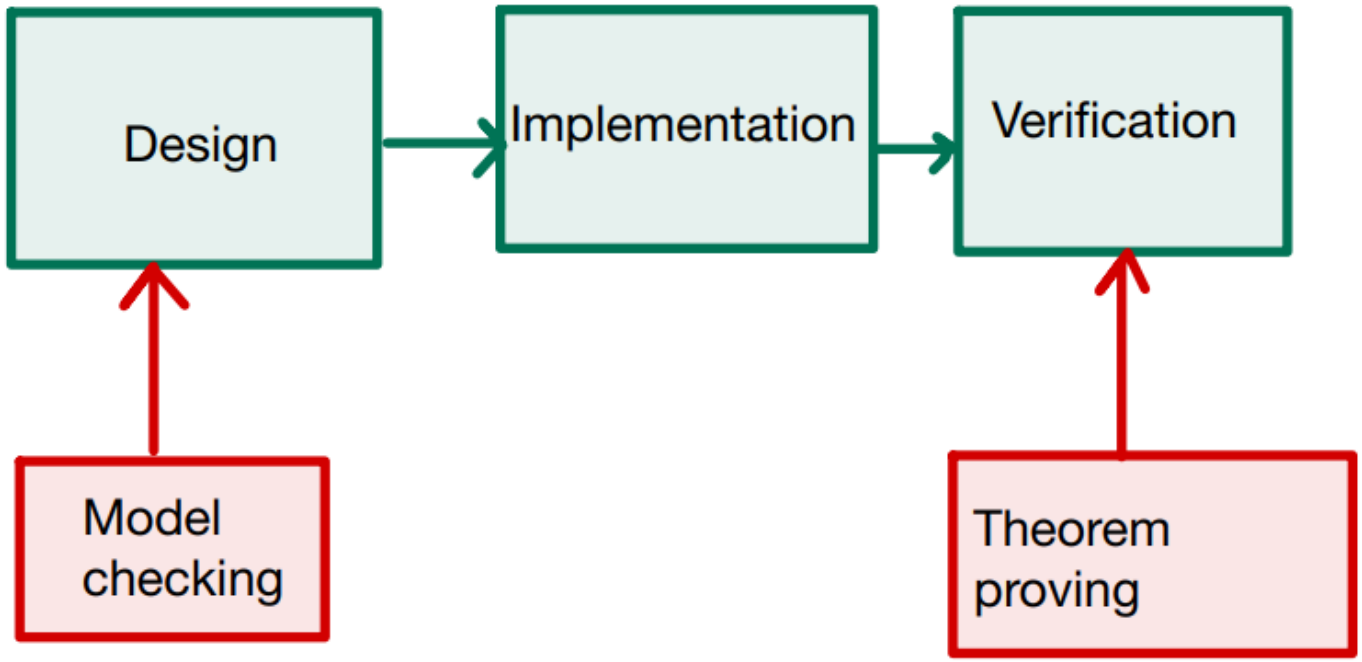
Stefan Nožinić (stefan@lugons.org)

September 7, 2023

# Agenda

- Motivating example
- What are formal specs?
- TLA+ and PlusCal
- One real world example
- Conclusion

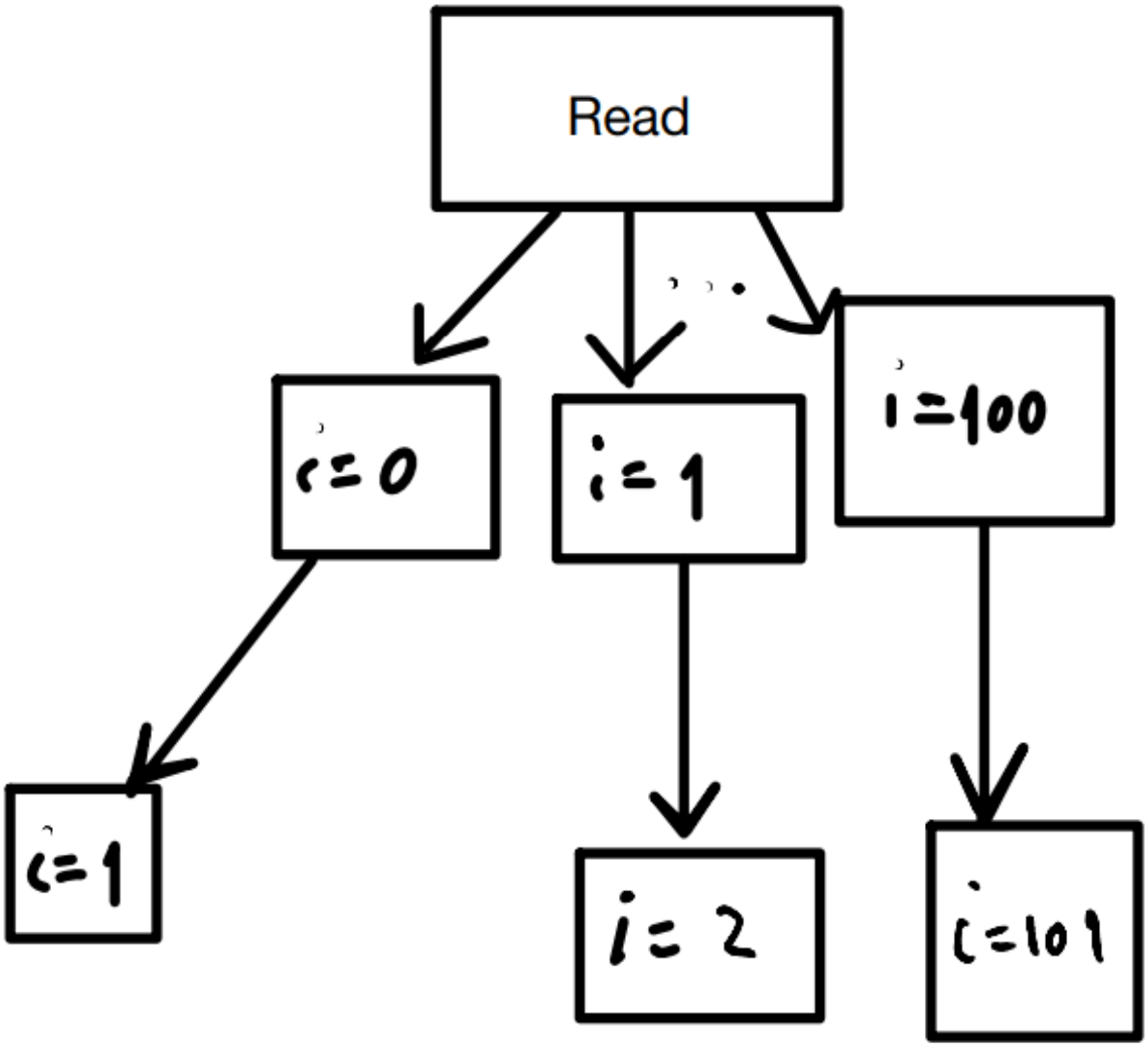
# Process


Design
Implementation
Verification
Model checking
Theorem proving

```
void main() {
    int i = getValue();
    i++;
    setValue(i);
}
```

**How to model this simple program formally as state machine?**

# State diagram



```
void thread2() {
    int i = getValue();
    i++;
    setValue(i);
}

void thread1() {
    int i = getValue();
    i++;
    setValue(i);
}
```

# Networked program

```
void processRequest() {
    Message *msg = receiveMessage();
    if (msg->type == CLIENT_REQUEST) {
        ...
    } else {
        ...
    }
    for (auto node : getNodes()) {
        node->sendMessage(NODE_REQUEST);
    }
}
```

```
void main() {
    int i = getValue(); // start
    i++; // middle
    setValue(i); // done
}
```

# TLA+

- Essentially we specify state machine and its properties
  - State variables
  - What are valid initial states
  - What are valid next states, given current state
  - Properties

MODULE *simple_increment_sm*

EXTENDS $TLC$, $Integers$

VARIABLES $i$, $pc$

$$init \triangleq i = 0 \land pc = \text{“start”}$$

$$next \triangleq \text{IF } pc = \text{“start”} \text{ THEN } (i' \in 0 \,..\, 1000) \land (pc' = \text{“middle”}) \text{ ELSE IF } pc = \text{“middle”} \text{ THEN } (i' = i + 1) \land (pc' = \text{“done”}) \text{ ELSE FALSE}$$

$$TerminationProperty \triangleq \Box(pc = \text{“done”} \Rightarrow i > 0)$$

# PlusCal

- A little more progreammer-friendly
- We specify processes and TLC will check all behaviours

# Real world example - health monitor

- We have several nodes (lets say nodes are 1, 2 and 3)
- Every node can reboot and recover later on
- Every node has one instance of service called "replicator"
- When node is down, its replicator instance gets transferred to another node which is up
- When we detect that replicato instance is stuck, we kill it and restart it
- We state that eventually if replicator is stuck, this will lead to either it being killed or recovered by itself

▼ **14: Orchestrator in heartbeat >>**

| | |
|---|---|
| ► alive (1) | {3} |
| killed (0) | {} |
| ► pc (4) M | (0 :> "RebootNode" @@ 1 :> "NodeDown" @@ 2 :> "NodeDown" … |
| ► replOwner (3) | <<2, 3, 3>> |
| ► replStuck (3) | <<TRUE, TRUE, FALSE>> |

▼ **15: RestartReplicator in heartbeat >>**

| | |
|---|---|
| ► alive (1) | {3} |
| killed (0) | {} |
| ► pc (4) M | (0 :> "RebootNode" @@ 1 :> "NodeDown" @@ 2 :> "NodeDown" … |
| ► replOwner (3) | <<2, 3, 3>> |
| ► replStuck (3) | <<TRUE, TRUE, FALSE>> |

▼ **16: RebootNode in heartbeat >>**

| | |
|---|---|
| ► alive (1) | {3} |
| killed (0) | {} |
| ► pc (4) M | (0 :> "Orchestrator" @@ 1 :> "NodeDown" @@ 2 :> "NodeDown" … |
| ► replOwner (3) | <<2, 3, 3>> |
| ► replStuck (3) | <<TRUE, TRUE, FALSE>> |

▼ **17: Orchestrator in heartbeat >>**

| | |
|---|---|
| ► alive (1) | {3} |
| killed (0) | {} |
| ► pc (4) M | (0 :> "RebootNode" @@ 1 :> "NodeDown" @@ 2 :> "NodeDown" … |
| ► replOwner (3) | <<2, 3, 3>> |
| ► replStuck (3) | <<TRUE, TRUE, FALSE>> |

▼ **18: P in heartbeat >>**

| | |
|---|---|
| ► alive (1) | {3} |
| killed (0) | {} |
| ► pc (4) M | (0 :> "RebootNode" @@ 1 :> "NodeDown" @@ 2 :> "NodeDown" … |
| ► replOwner (3) | <<2, 3, 3>> |
| ► replStuck (3) | <<TRUE, TRUE, FALSE>> |

► **15: Back to state >>**

## Status

Check again  Full output

Checking heartbeat.tla / heartbeat.cfg

Success : Fingerprint collision probability: 4.4E-11

Start: 13:23:48 (Jul 4), end: 13:23:55 (Jul 4)

---

## States

| Time | Diameter | Found | Distinct | Queue |
|---|---|---|---|---|
| 00:00:00 | 0 | 1 | 1 | 1 |
| 00:00:03 | 13 | 36 691 | 9 543 | 2 062 |
| 00:00:05 | 21 | 69 801 | 14 637 | 0 |
| 00:00:06 | 21 | 69 801 | 14 637 | 0 |

## Coverage

| Module | Action | Total | Distinct |
|---|---|---|---|
| heartbeat | Init | 1 | 1 |
| heartbeat | P | 11 895 | 5 256 |
| heartbeat | CheckIfStuck | 11 004 | 4 365 |
| heartbeat | RestartReplicator | 15 465 | 570 |
| heartbeat | NodeDown | 0 | 0 |
| heartbeat | Orchestrator | 9 894 | 2 964 |
| heartbeat | RebootNode | 7 245 | 208 |
| heartbeat | MakeReplicatorStuck | 14 637 | 1 273 |
| heartbeat | Terminating | 0 | 0 |

# Conclusion

- Formal specification can help us reason about systems and communicate better in teams
- There are tools to help us formally specify systems and to check its validity
- More granular we go, more validation we get

# Gossip session