TLA+

An introduction to writing “exhaustively testable pseudocode” [newcombe2015aws]

Seminar Advanced Programming Tools WS 22/23
By Daniel Stachnik, 31.01.2023

> “For the human makers of things, the incompleteness and inconsistencies of our ideas, become clear only during implementation.”
>
> *- Fred Brooks, The Mythical Man-Month, p. 15 [brooks1982month]*

How to verify behavior & properties?

	Design*	**Implementation**
Building a bridge	CAD	Manual Checks
Building software	?	Testing (automated, manual, …)

** designing = conceptualization and framing of a system [freeman2004design]*

How to verify behavior & properties?

	Design*	**Implementation**
Building a bridge	CAD	Manual Checks
Building software	**TLA+**	Testing (automated, manual, …)

** designing = conceptualization and framing of a system [freeman2004design]*

How to verify behavior & properties?

	Design*	**Implementation**
Building a bridge	CAD	Manual Checks
Building software	**TLA+ \in Formal Methods**	Testing (automated, manual, …)

** designing = conceptualization and framing of a system [freeman2004design]*

> “Going deeper, most software people are just trying to do FAB, and most of the tools are FAB tools — there is very little CAD and even less SIM in “software engineering”. To my old eyes, this doesn’t look or feel like real engineering process.”

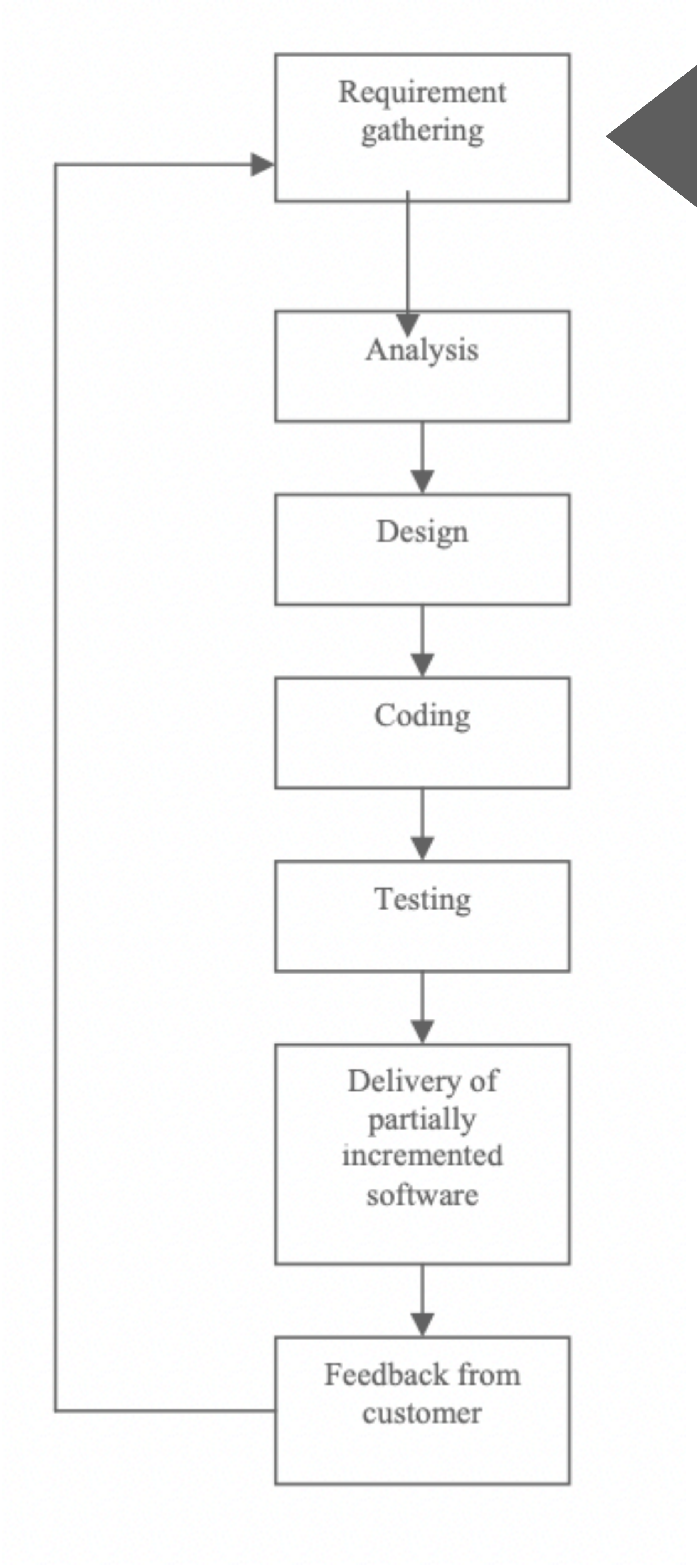
Alan Kay, https://www.quora.com/How-does-one-get-as-close-as-possible-to-mastering-software-engineering/answer/Alan-Kay-11?share=1

> “For the human makers of things, the incompleteness and inconsistencies of our ideas, become clear only during implementation.”
>
> *- Fred Brooks, The Mythical Man-Month, p. 15 [brooks1982month]*

➡ TLA+ and other lightweight formal methods try to lower “incompletenesses” and “inconsistencies” by formalizing the design part

Overview TLA+

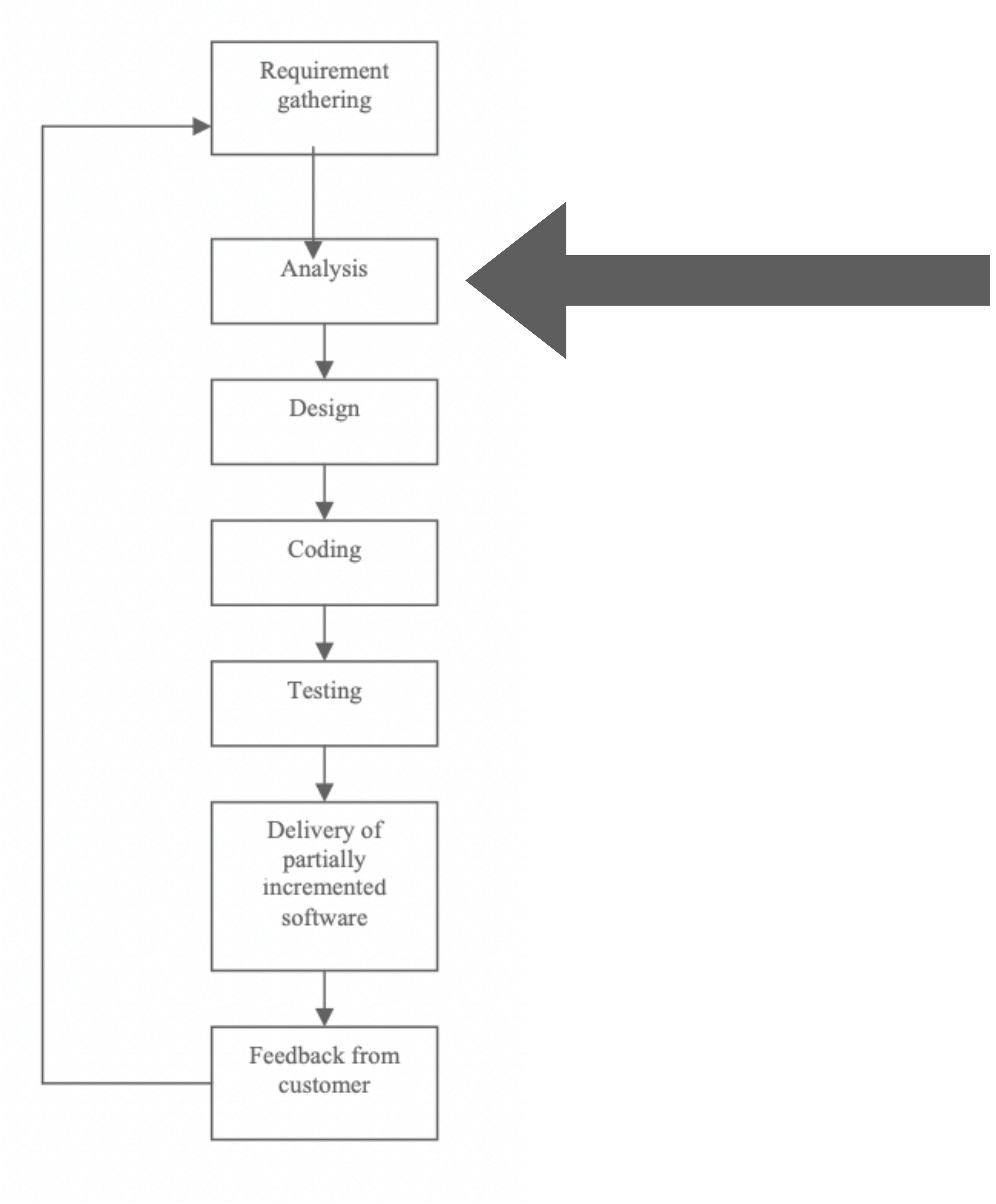
TLA+ Workflow



Agile Dev Cycle [sharma2012agile]

C: “I want my files to synchronize between all my devices”

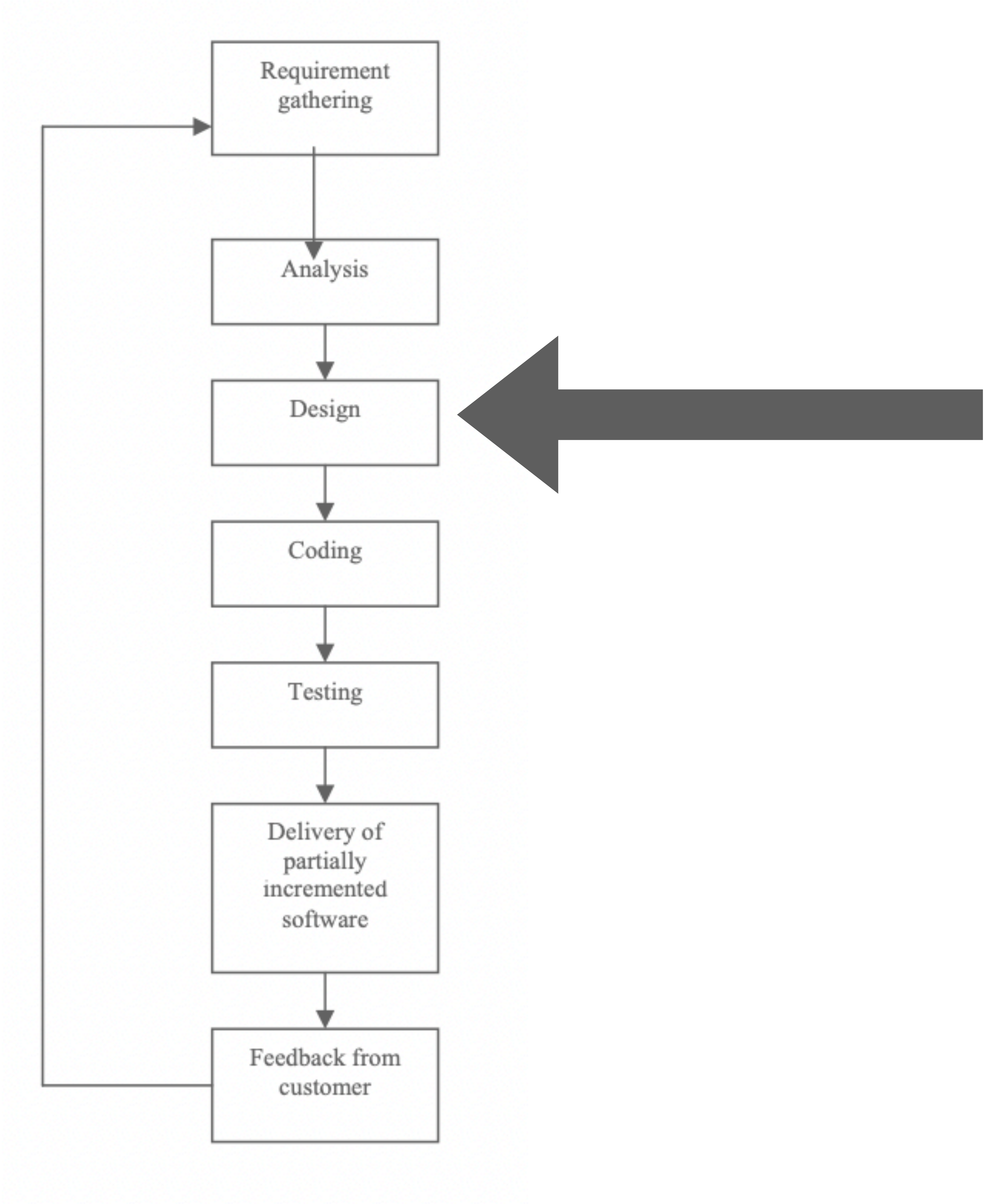
TLA+ Workflow



Agile Dev Cycle [sharma2012agile]

1. *SW supports multiple client devices connected over internet*
2. *Changes at one device will be propagated to other nodes*

TLA+ Workflow

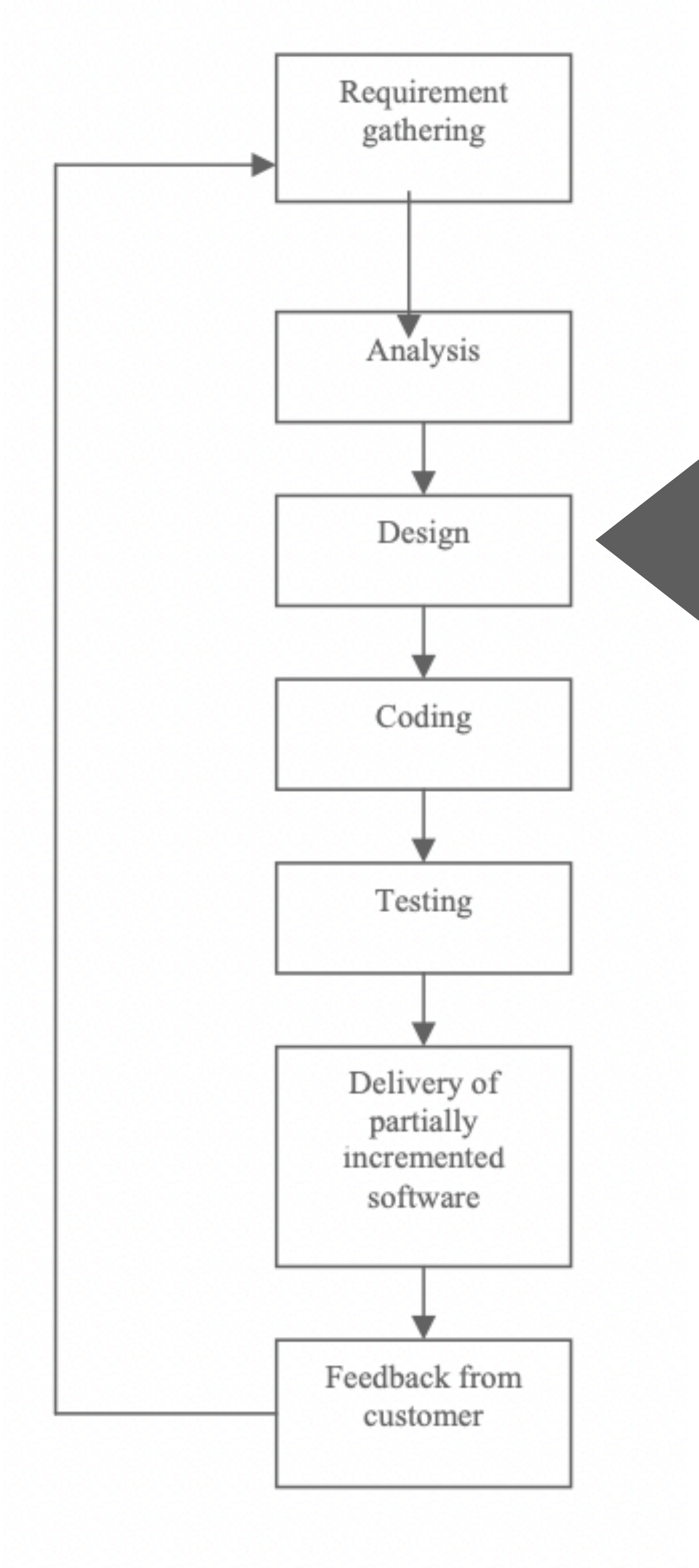


Agile Dev Cycle
[sharma2012agile]

Sketch idea

```
if madeChanges:
    Commit all changes to other active nodes
    If all confirm: commit on own device
    If any didn't confirm: retry after
    exponentially increasing time-out
```

TLA+ Workflow



Agile Dev Cycle [sharma2012agile]

Sketch idea

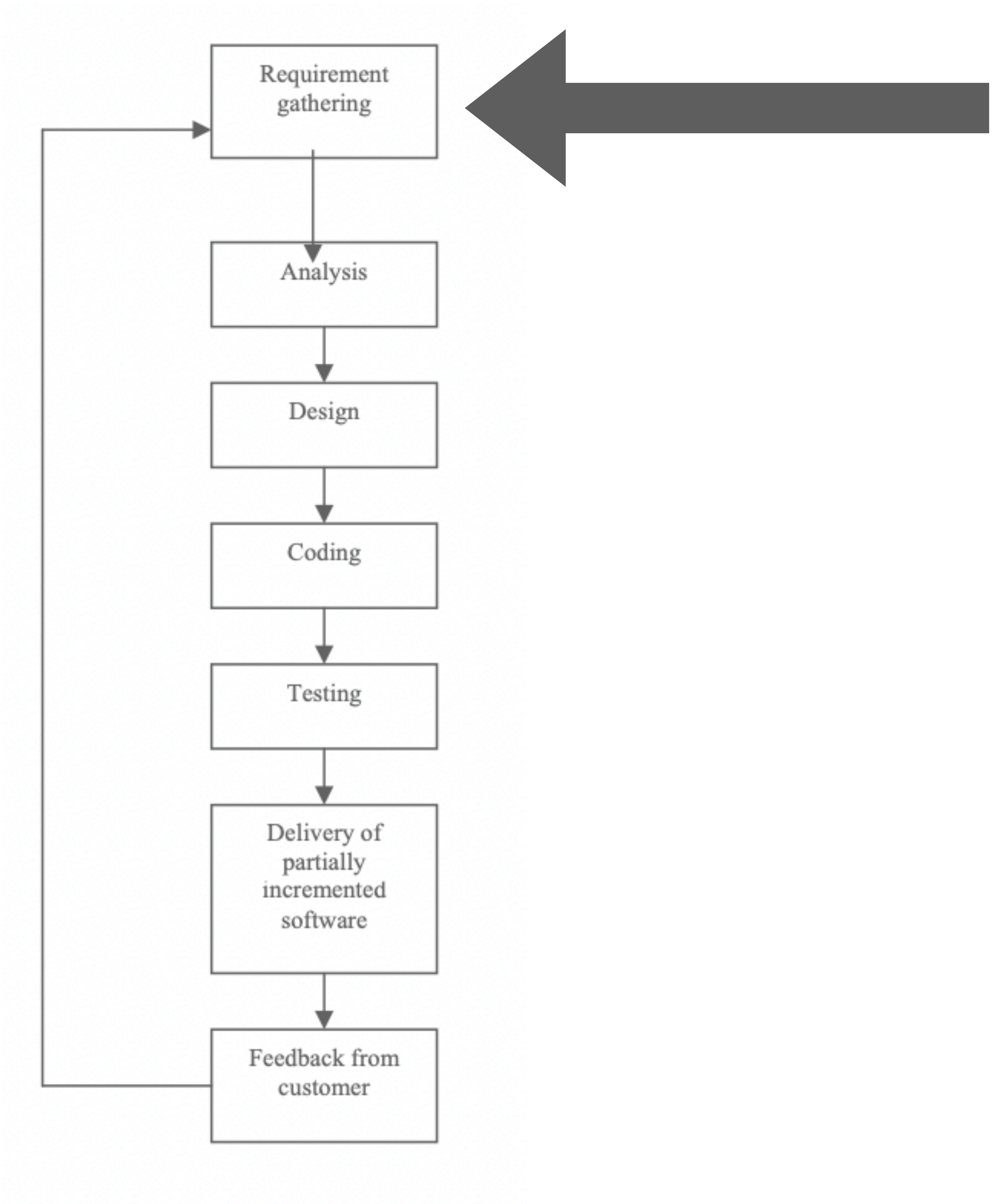
```
if madeChanges:
  Commit all changes to other active nodes
  If all confirm: commit on own device
  If any didn’t confirm: retry after
  exponentially increasing time-out
```

Write TLA+-spec *(~20 loc)*

Model-check spec

→ *Fails because of deadlock: two nodes can commit at same time and wait forever*

TLA+ Workflow



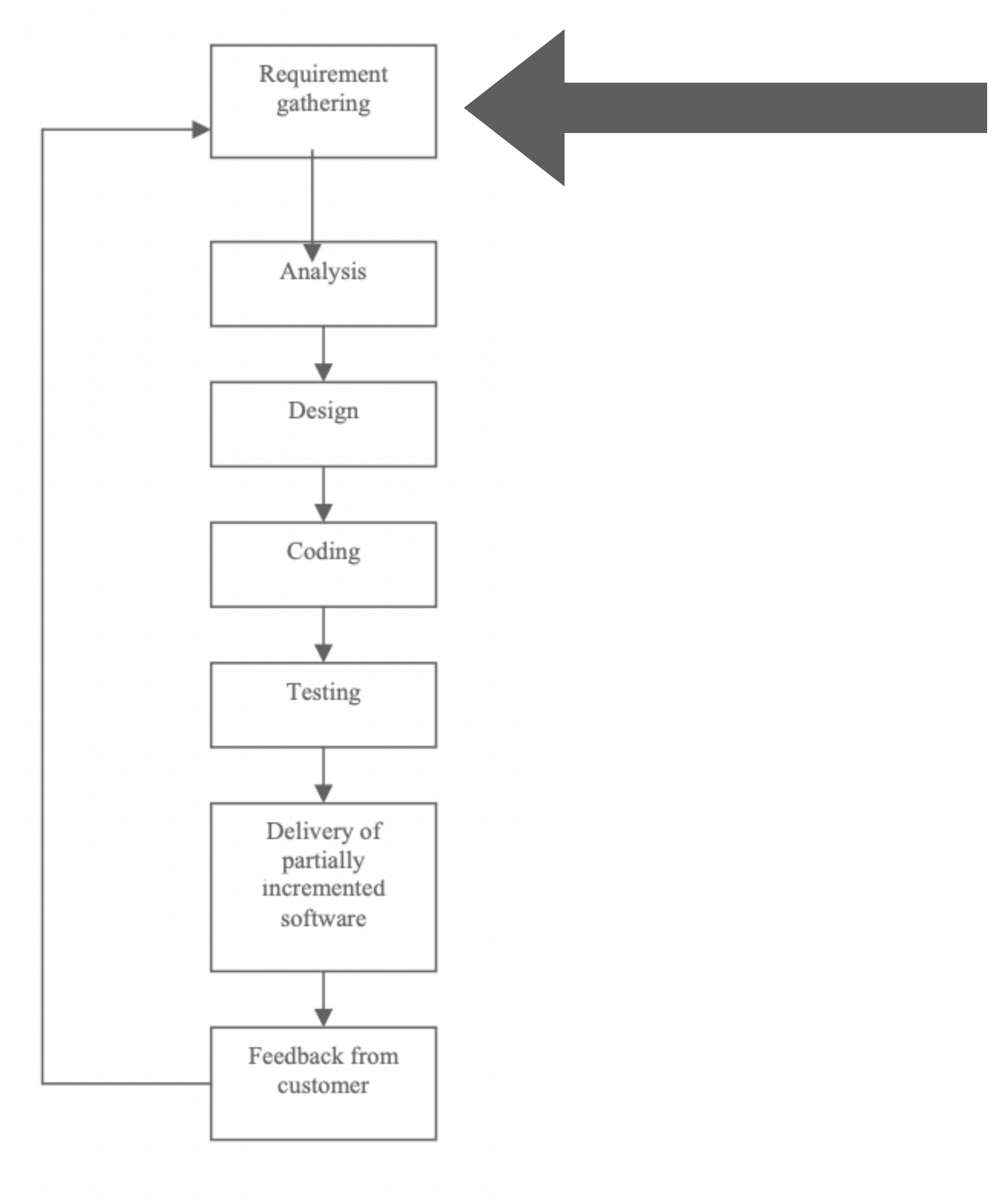
Agile Dev Cycle [sharma2012agile]

C: “I want my files to synchronize between all my devices”

You: “Support file changes on multiple devices at same time?”

C: “Yes”

TLA+ Workflow



Agile Dev Cycle [sharma2012agile]

C: “I want my files to synchronize between all my devices”

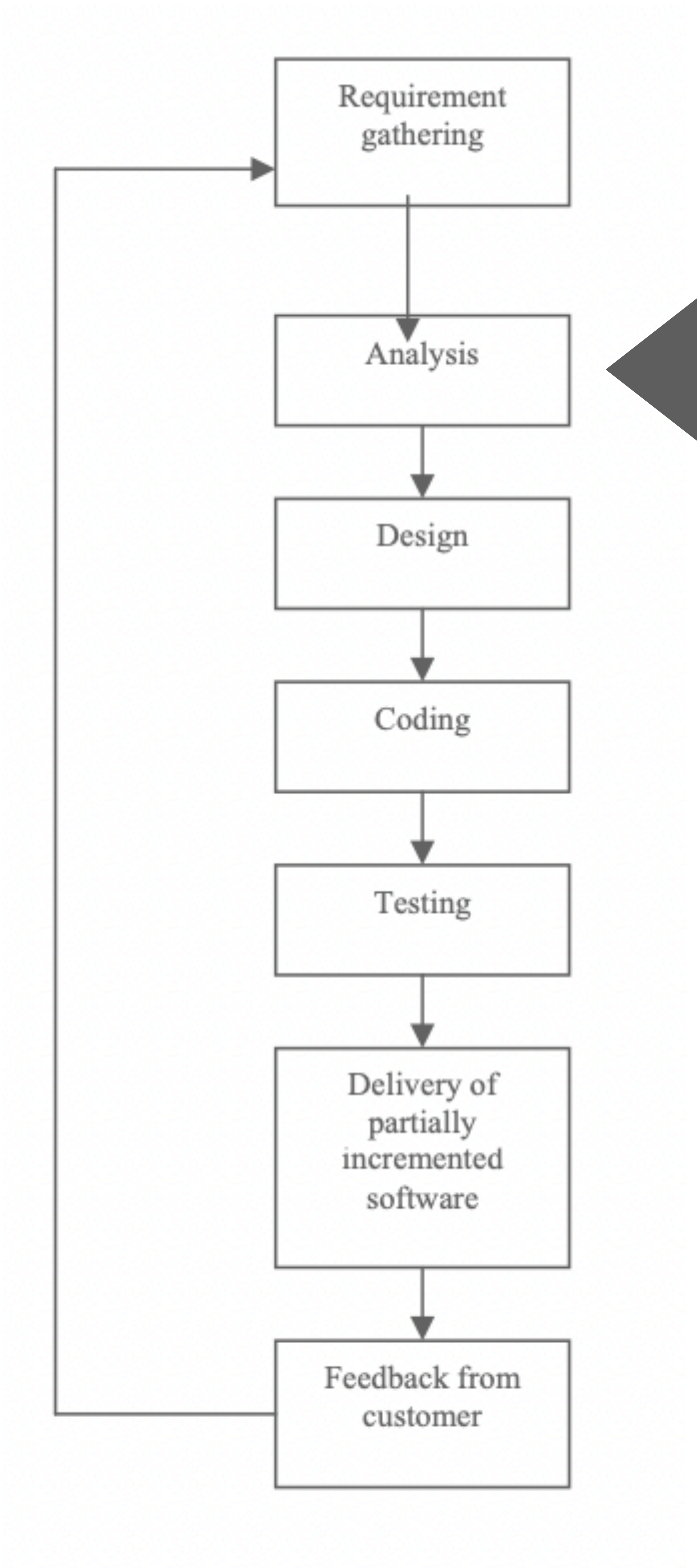
You: “Support file changes on multiple devices at same time?”

C: “Yes”

You: “That will cost extra”

C: “No worries, I know software design is hard”

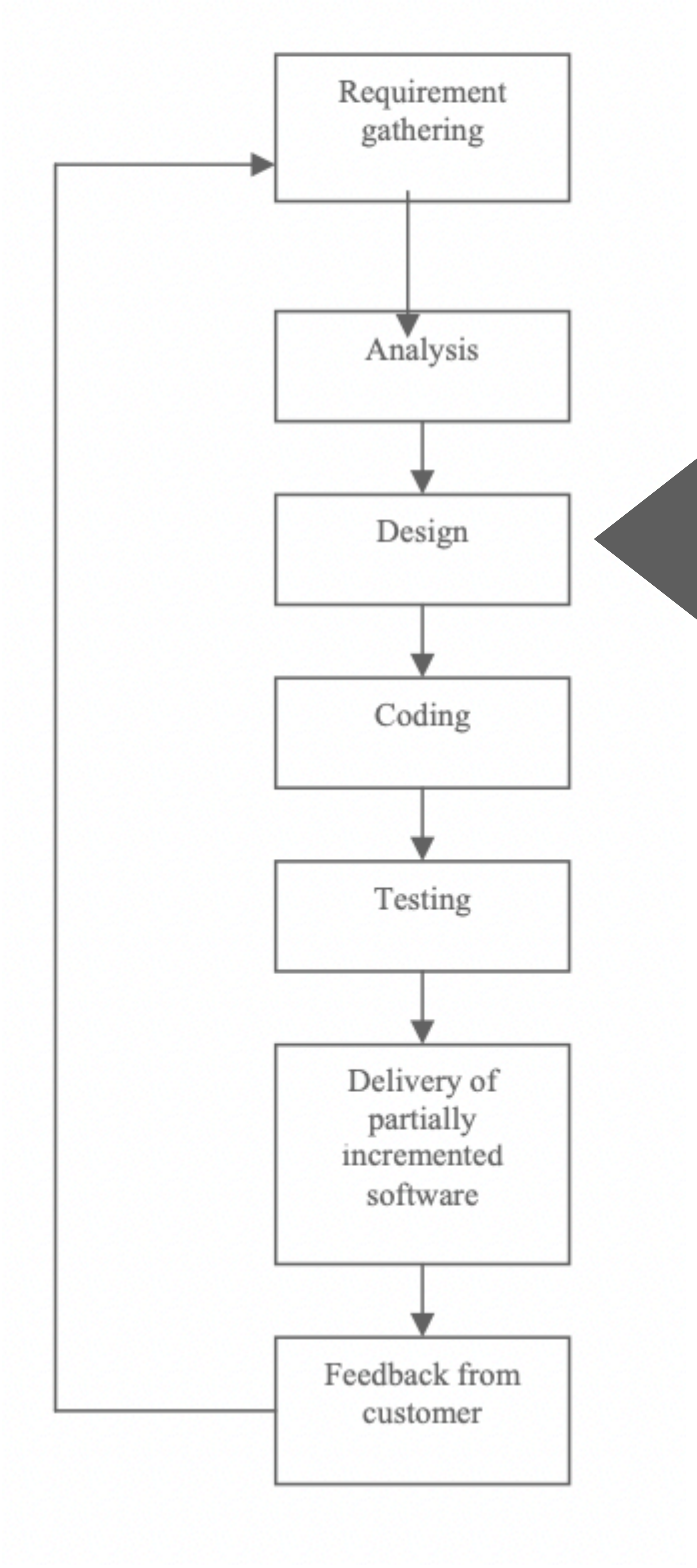
TLA+ Workflow



Agile Dev Cycle [sharma2012agile]

1. *SW supports multiple client devices connected over internet*
2. *Changes at one device will be propagated to other nodes*
3. *Multiple changes may be done at the same time*

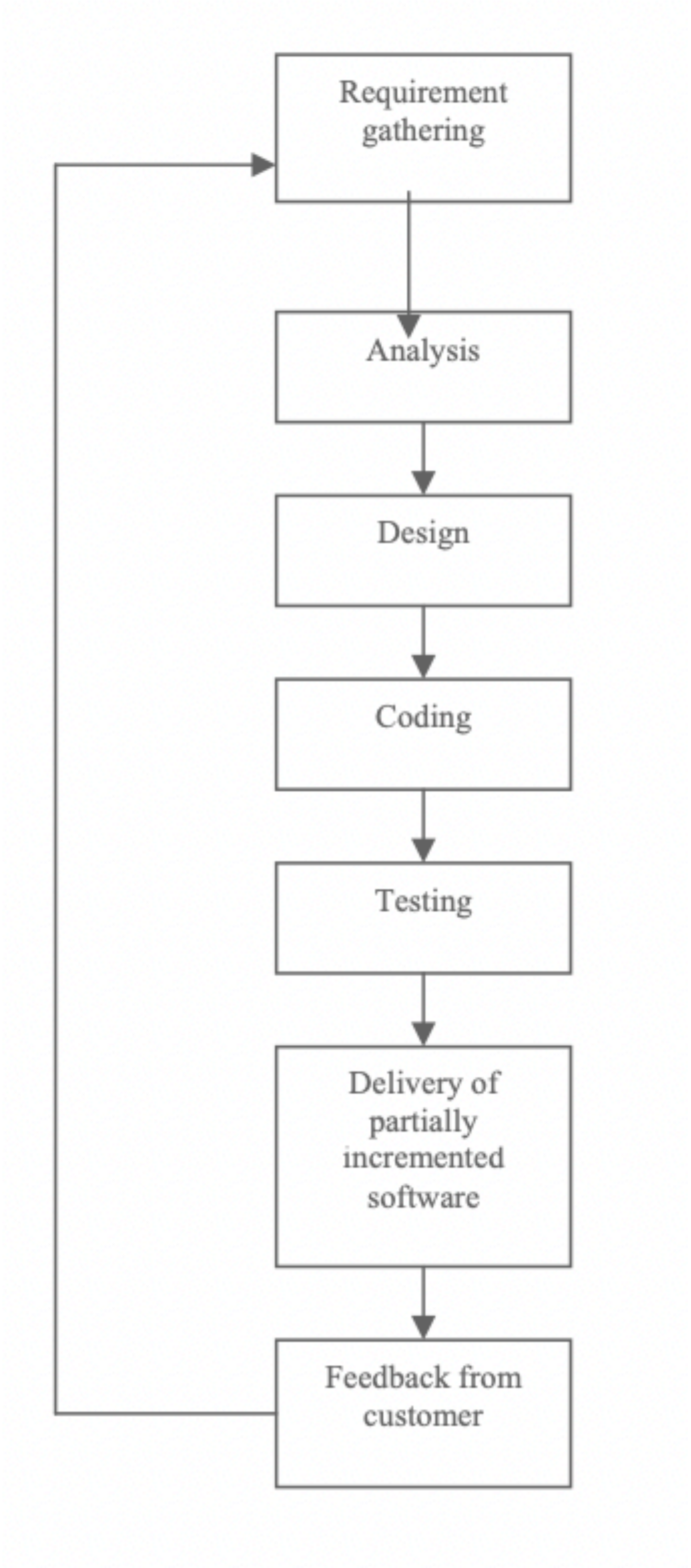
TLA+ Workflow



Agile Dev Cycle [sharma2012agile]

Sketch → Write Spec → Model-check → …

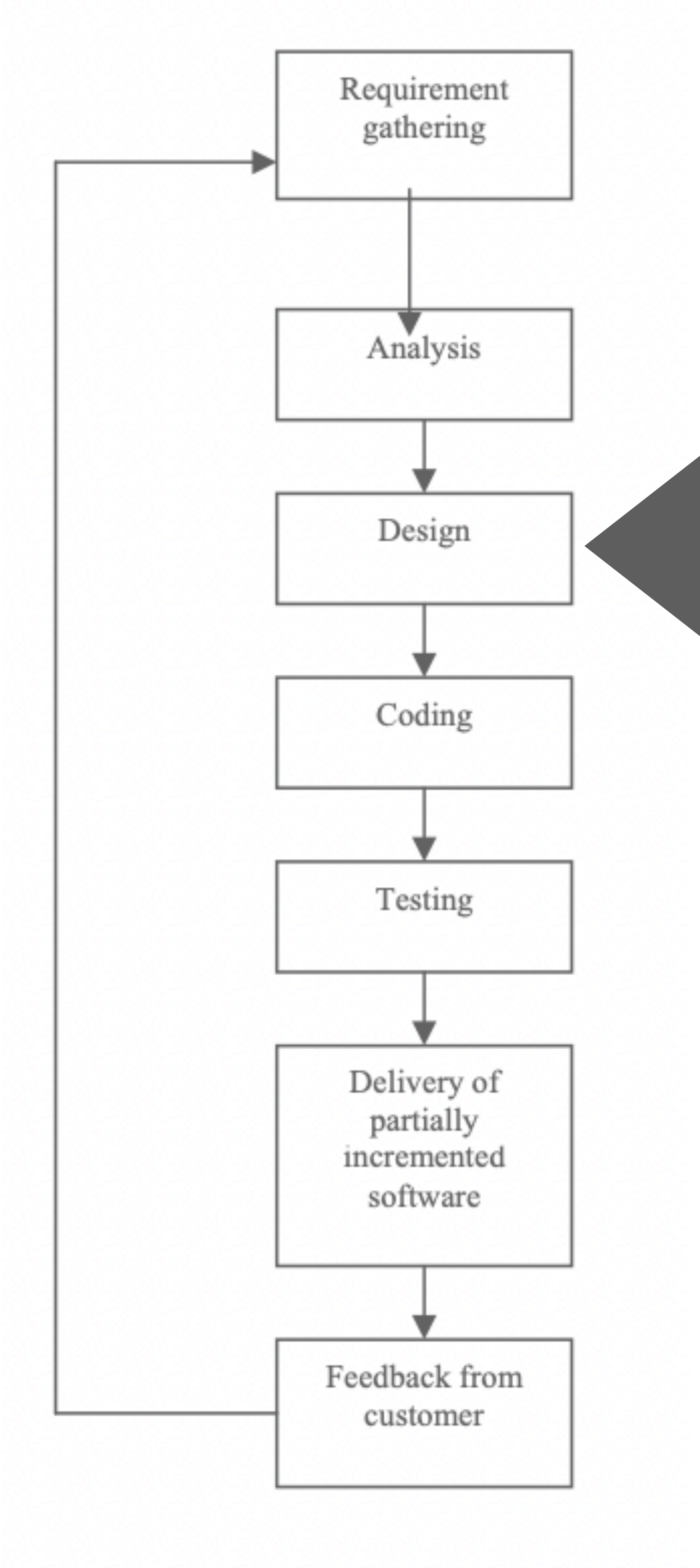
TLA+ Workflow



Agile Dev Cycle
[sharma2012agile]

How much time would you have needed, hadn’t you used TLA+?

TLA+ Workflow



Requirement gathering

Analysis

Design

Coding

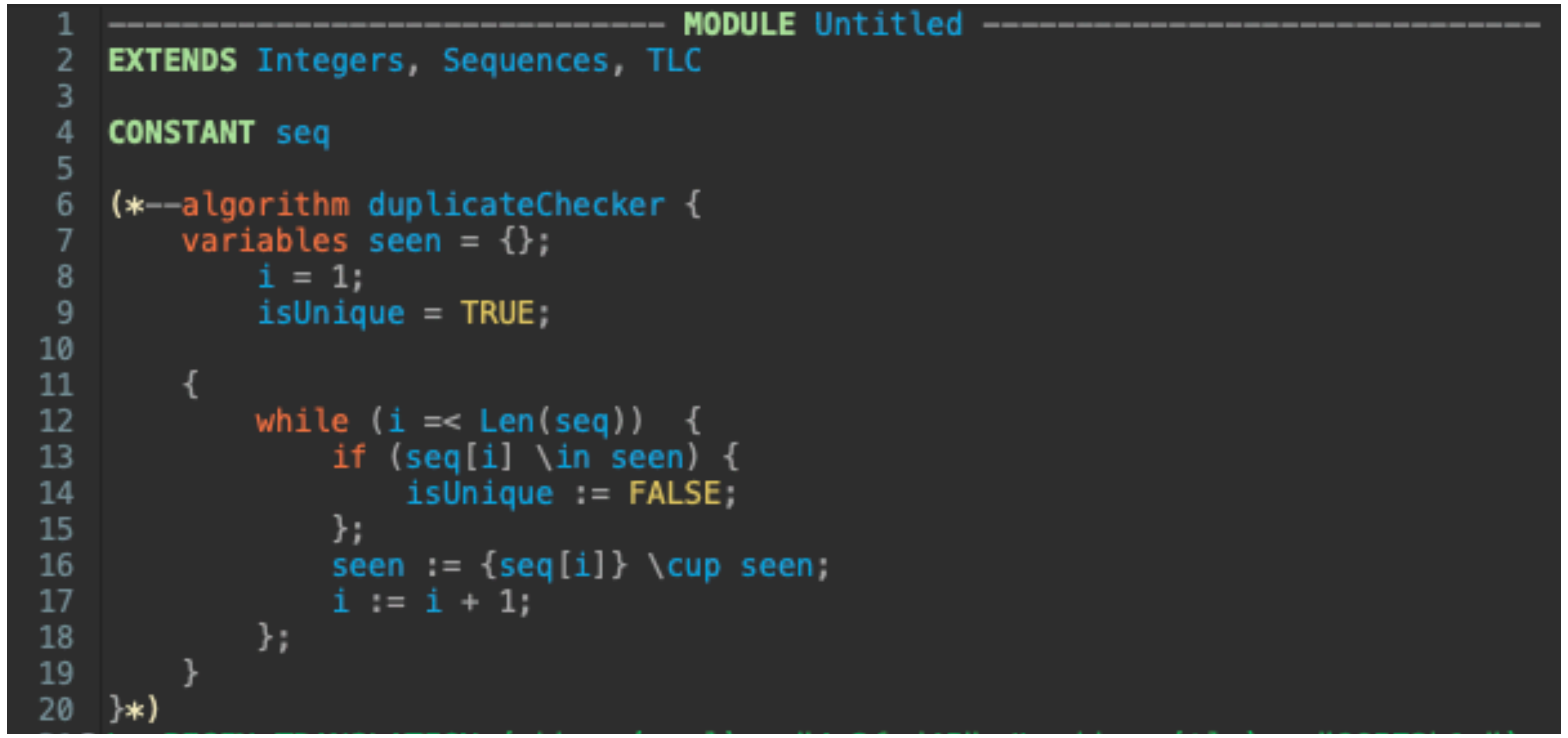
Testing

Delivery of partially incremented software

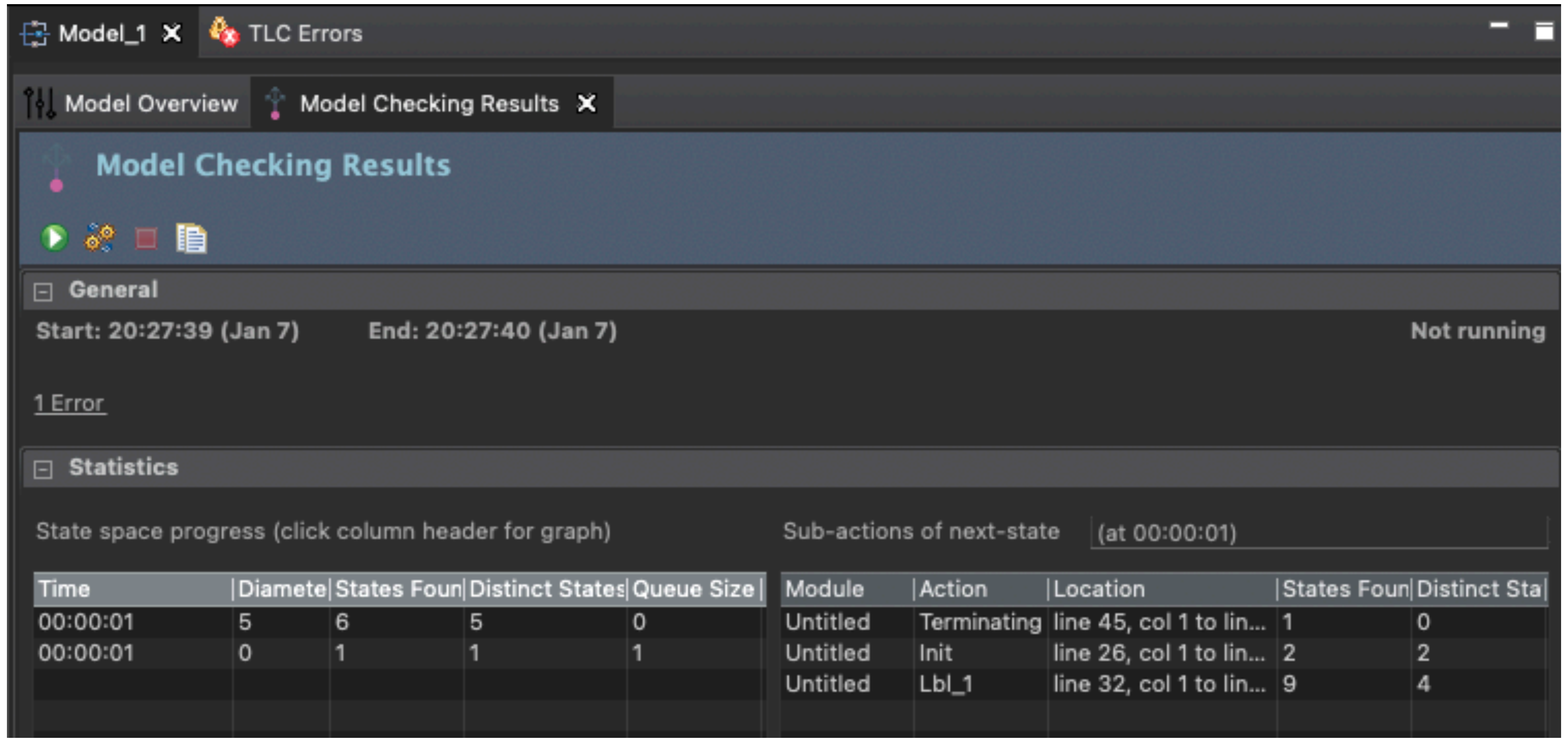
Feedback from customer

Agile Dev Cycle
[sharma2012agile]

1. Write specification



```
------------------------------- MODULE Untitled -------------------------------
EXTENDS Integers, Sequences, TLC

CONSTANT seq

(*--algorithm duplicateChecker {
    variables seen = {};
        i = 1;
        isUnique = TRUE;

    {
        while (i =< Len(seq))  {
            if (seq[i] \in seen) {
                isUnique := FALSE;
            };
            seen := {seq[i]} \cup seen;
            i := i + 1;
        };
    }
}*)
```



Model Checking Results

General

Start: 20:27:39 (Jan 7)   End: 20:27:40 (Jan 7)   Not running

1 Error

Statistics

State space progress (click column header for graph)

Time	Diamete	States Foun	Distinct States	Queue Size
00:00:01	5	6	5	0
00:00:01	0	1	1	1

Sub-actions of next-state (at 00:00:01)

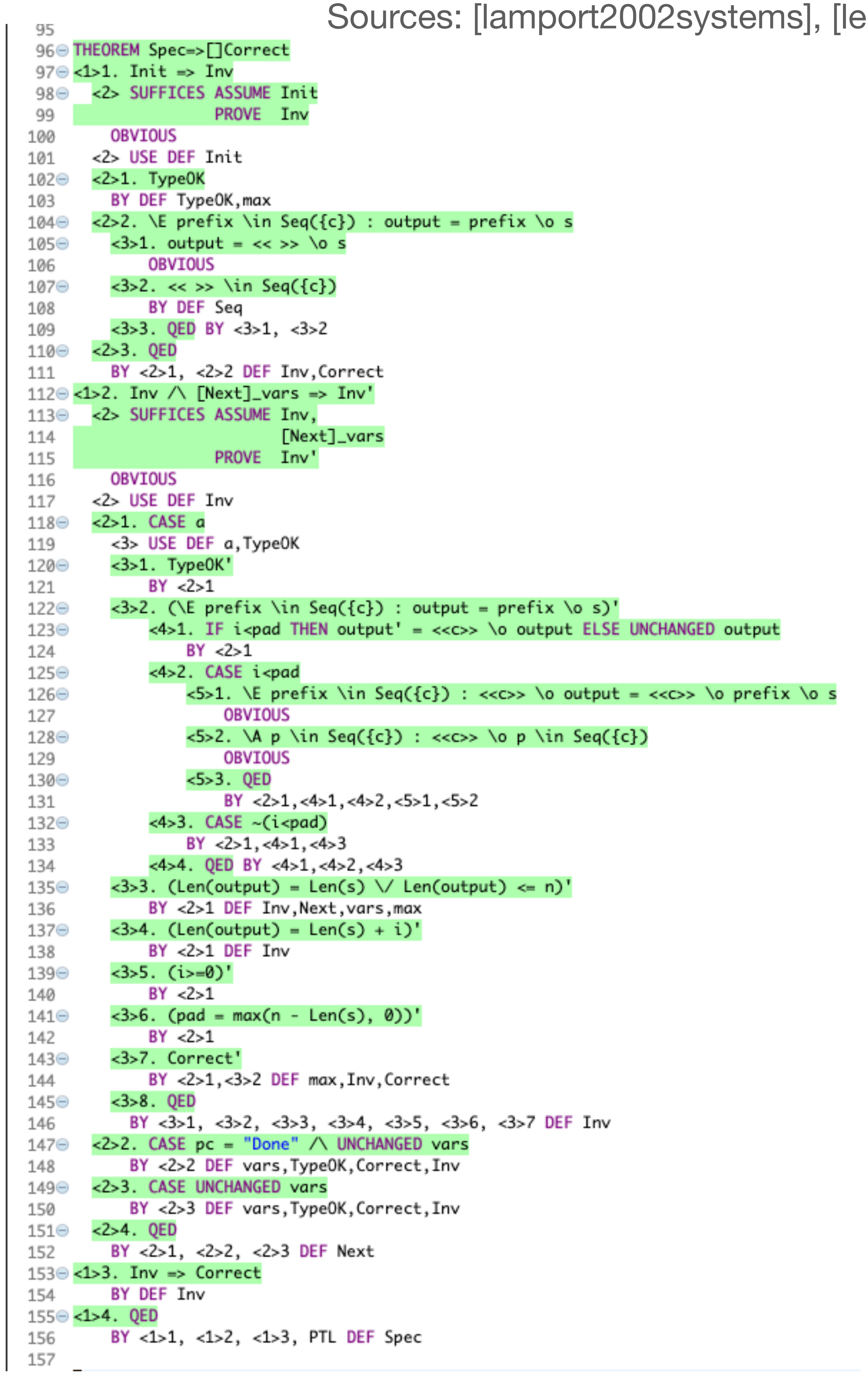
Module	Action	Location	States Foun	Distinct Sta
Untitled	Terminating	line 45, col 1 to lin...	1	0
Untitled	Init	line 26, col 1 to lin...	2	2
Untitled	Lbl_1	line 32, col 1 to lin...	9	4

2. Check model

The TLA+ Tools

- TLA+ = Temporal Logic of Actions Plus
- TLA+ is a specification language (not a programming language)
- Developed by Leslie Lamport to model/specify concurrent systems
- Relevant tools that work on TLA specs:
    - TLC Model Checker
    - TLAPS Proof System

Sources: [lamport2002systems], [learntla]



```
THEOREM Spec=>[]Correct
<1>1. Init => Inv
  <2> SUFFICES ASSUME Init
               PROVE  Inv
    OBVIOUS
  <2> USE DEF Init
  <2>1. TypeOK
    BY DEF TypeOK,max
  <2>2. \E prefix \in Seq({c}) : output = prefix \o s
    <3>1. output = << >> \o s
        OBVIOUS
    <3>2. << >> \in Seq({c})
        BY DEF Seq
    <3>3. QED BY <3>1, <3>2
  <2>3. QED
    BY <2>1, <2>2 DEF Inv,Correct
<1>2. Inv /\ [Next]_vars => Inv'
  <2> SUFFICES ASSUME Inv,
                      [Next]_vars
               PROVE  Inv'
    OBVIOUS
  <2> USE DEF Inv
  <2>1. CASE a
    <3> USE DEF a,TypeOK
    <3>1. TypeOK'
        BY <2>1
    <3>2. (\E prefix \in Seq({c}) : output = prefix \o s)'
        <4>1. IF i<pad THEN output' = <<c>> \o output ELSE UNCHANGED output
            BY <2>1
        <4>2. CASE i<pad
            <5>1. \E prefix \in Seq({c}) : <<c>> \o output = <<c>> \o prefix \o s
                OBVIOUS
            <5>2. \A p \in Seq({c}) : <<c>> \o p \in Seq({c})
                OBVIOUS
            <5>3. QED
                BY <2>1,<4>1,<4>2,<5>1,<5>2
        <4>3. CASE ~(i<pad)
            BY <2>1,<4>1,<4>3
        <4>4. QED BY <4>1,<4>2,<4>3
    <3>3. (Len(output) = Len(s) \/ Len(output) <= n)'
        BY <2>1 DEF Inv,Next,vars,max
    <3>4. (Len(output) = Len(s) + i)'
        BY <2>1 DEF Inv
    <3>5. (i>=0)'
        BY <2>1
    <3>6. (pad = max(n - Len(s), 0))'
        BY <2>1
    <3>7. Correct'
        BY <2>1,<3>2 DEF max,Inv,Correct
    <3>8. QED
      BY <3>1, <3>2, <3>3, <3>4, <3>5, <3>6, <3>7 DEF Inv
  <2>2. CASE pc = "Done" /\ UNCHANGED vars
      BY <2>2 DEF vars,TypeOK,Correct,Inv
  <2>3. CASE UNCHANGED vars
      BY <2>3 DEF vars,TypeOK,Correct,Inv
  <2>4. QED
    BY <2>1, <2>2, <2>3 DEF Next
<1>3. Inv => Correct
    BY DEF Inv
<1>4. QED
    BY <1>1, <1>2, <1>3, PTL DEF Spec
```

TLAPS spec, proving *“Leftpad”* [merz2020leftpad]

Story 1:
The Amazon Story

Source: [newcombe2015aws]

The Amazon Story

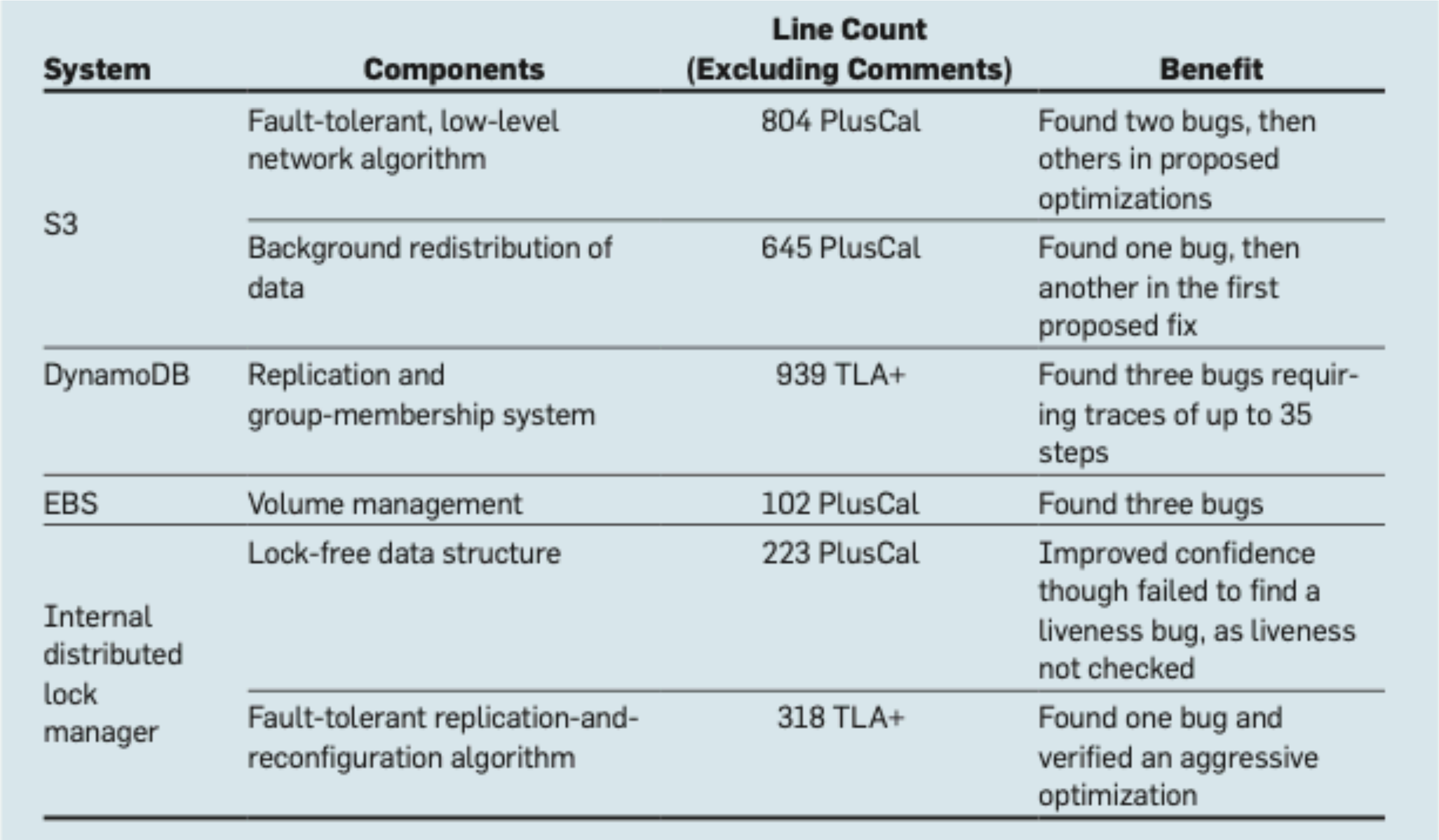
Background

- Amazon Web Services (AWS) provide a range of fault-tolerant distributed systems
- Critical systems are highly tested using conventional automated tests
    - → this is “inadequate as a method for finding subtle errors in design, as the number of reachable states of the code is astronomical.”
- Engineers check formal methods and choose TLA+

Source: [newcombe2015aws]

The Amazon Story

Usage



System	Components	Line Count (Excluding Comments)	Benefit
S3	Fault-tolerant, low-level network algorithm	804 PlusCal	Found two bugs, then others in proposed optimizations
	Background redistribution of data	645 PlusCal	Found one bug, then another in the first proposed fix
DynamoDB	Replication and group-membership system	939 TLA+	Found three bugs requiring traces of up to 35 steps
EBS	Volume management	102 PlusCal	Found three bugs
Internal distributed lock manager	Lock-free data structure	223 PlusCal	Improved confidence though failed to find a liveness bug, as liveness not checked
	Fault-tolerant replication-and-reconfiguration algorithm	318 TLA+	Found one bug and verified an aggressive optimization

Source: [newcombe2015aws]

The Amazon Story

Benefits of using TLA+

- Higher correctness
- Seen as “what-if”-tool for system designs → facilitates big + safe refactoring
- Precise Documentation
- Better design by “stating precisely ‘what needs to go right.’”

Image Source: https://www.mdr.de/wissen/esa-raketen-probleme-starts-gefaehrdet-100.html

Story 2:
The ESA Story

Source: [verhulst2011rtos]

The ESA Story

Background

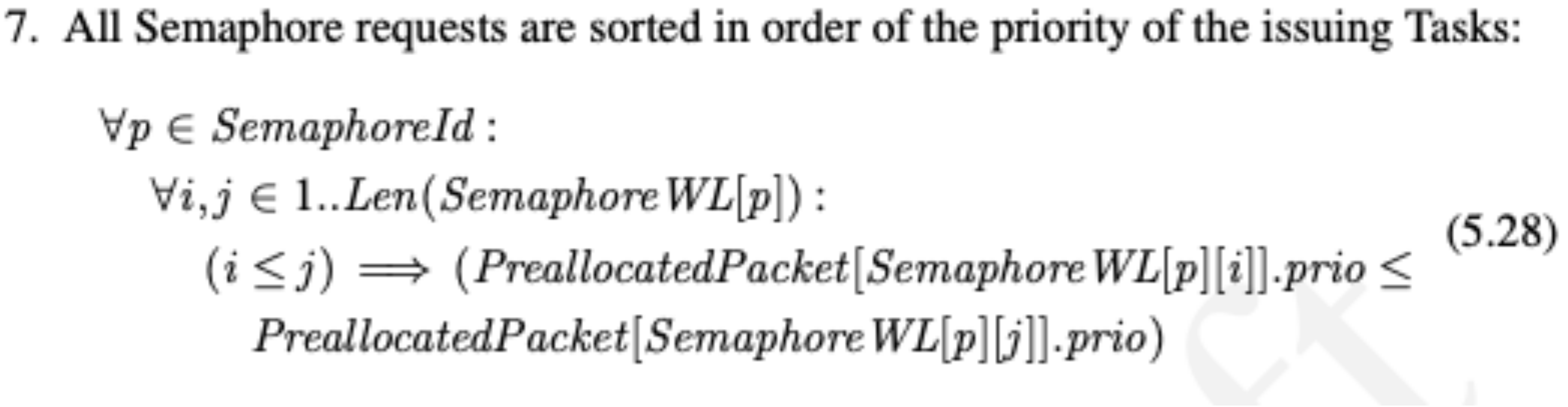
- ESA uses real-time operating systems (RTOS) for embedded systems
- RTOS considered “complex and difficult”
- Engineers wanted to develop it from ground-up to improve architecture and reduce complexities
- RTOS has many concurrent algorithms, needs to be “safe, correct and performing”
- Decided to use TLA+ at all abstraction levels (system-level to algorithm-level)

Source: [verhulst2011rtos]

The ESA Story

Example

Requirement: All Packets in the Semaphore Waiting List must be sorted, according to their priority, in decreasing order.



7. All Semaphore requests are sorted in order of the priority of the issuing Tasks:

$$\begin{aligned}
&\forall p \in SemaphoreId: \\
&\quad \forall i, j \in 1..Len(SemaphoreWL[p]): \\
&\qquad (i \leq j) \implies (PreallocatedPacket[SemaphoreWL[p][i]].prio \leq \\
&\qquad\quad PreallocatedPacket[SemaphoreWL[p][j]].prio)
\end{aligned} \tag{5.28}$$

[verhulst2011rtos], p. 101

Source: [verhulst2011rtos]

The ESA Story

Benefits of using TLA+

- TLA+ caused engineers to “formulate their thoughts clearly while exposing the hidden assumptions”
- “*[The TLA+]* abstraction helped a lot in coming to a much cleaner architecture (we witnessed first hand the brain washing done by years of C programming).” [lamport2019tla]
- → Clearer communication within team
- → Code size 5-10 times smaller than predecessor system
- → Required no “extra resources and time”

The ESA Story

Source: [verhulst2011rtos]

Benefits of using TLA+

- TLA+ caused engineers to “formulate their thoughts clearly while exposing the hidden assumptions”
- “*[The TLA+]* abstraction helped a lot in coming to a much cleaner architecture (we witnessed first hand the brain washing done by years of C programming).” [lamport2019tla]
- → Clearer communication within team
- → Code size 5-10 times smaller than predecessor system
- → Required no “extra resources and time”

Beware of implied causation: reimplemented OS + used different architecture

Source: [verhulst2011rtos]

OpemComRTOS Architecture

“An OpenComRTOS program is an implementation model of a more abstract architectural model that is composed of entities and interactions. In OpenComRTOS, we mainly have two types of entities: Tasks and Hubs.” [verhulst2011rtos], p. 105

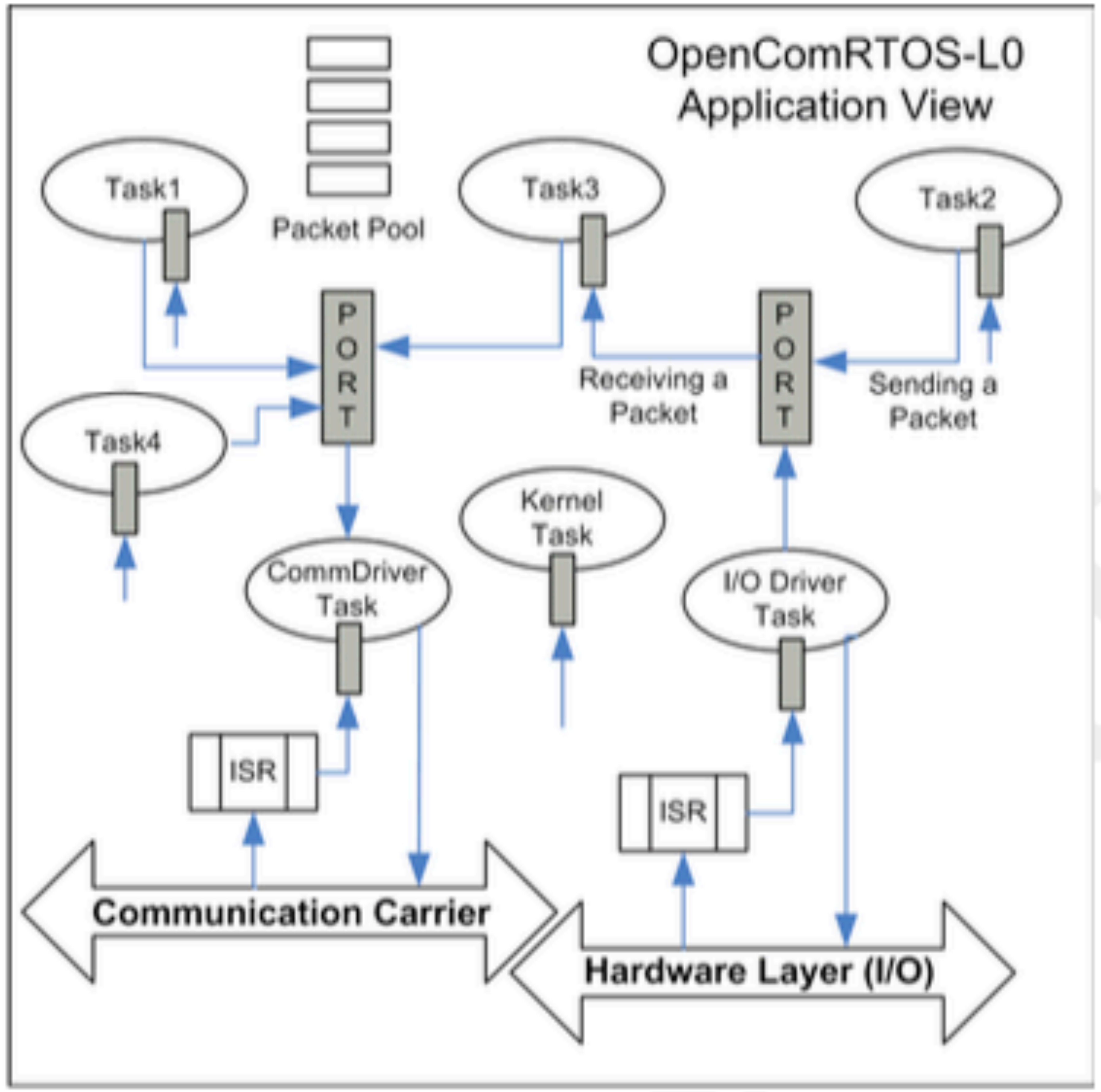


Fig. 6.1 OpenComRTOS application

Whorfian Syndrome

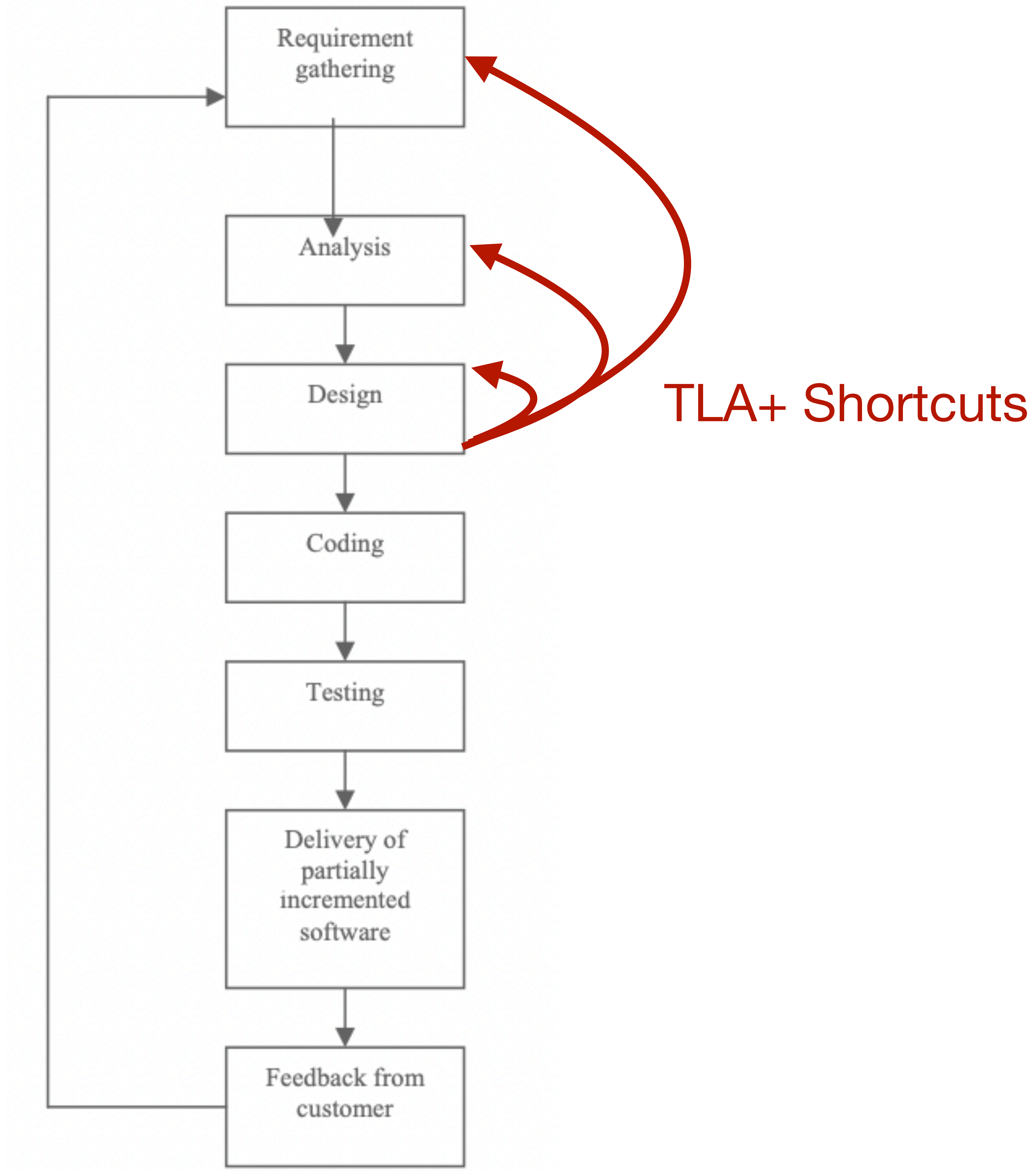
“Computer Science should be about concepts, not languages. […] I believe that the best way to get better programs is to teach programmers how to think better. […] But how does one teach concepts without getting distracted by the language in wich these concepts are expressed? My answer is to use the same language as every other branch of science and engineering – namely, mathematics”

Leslie Lamport, https://www.youtube.com/watch?v=15uy9Ga-14I

Learnings

TLA+ can provide…

- Higher Correctness
- Precise high-level overview revealing implicit assumptions in mental model
→ Cleaner Design
- Shorter (design) feedback loop



Agile Dev Cycle [sharma2012agile], adapted to TLA+

“TLA+ is the most valuable thing that I've learned in my professional career. […] It has changed how I think, by giving me a framework for constructing new kinds of mental-models, by revealing the **precise relationship between correctness properties** and system designs, and by allowing me to move from **‘plausible prose’ to precise statements** much earlier in the software development process.”

- Chris Newcombe, then Principal Engineer at Amazon, now at Oracle, [newcombe2012groups]

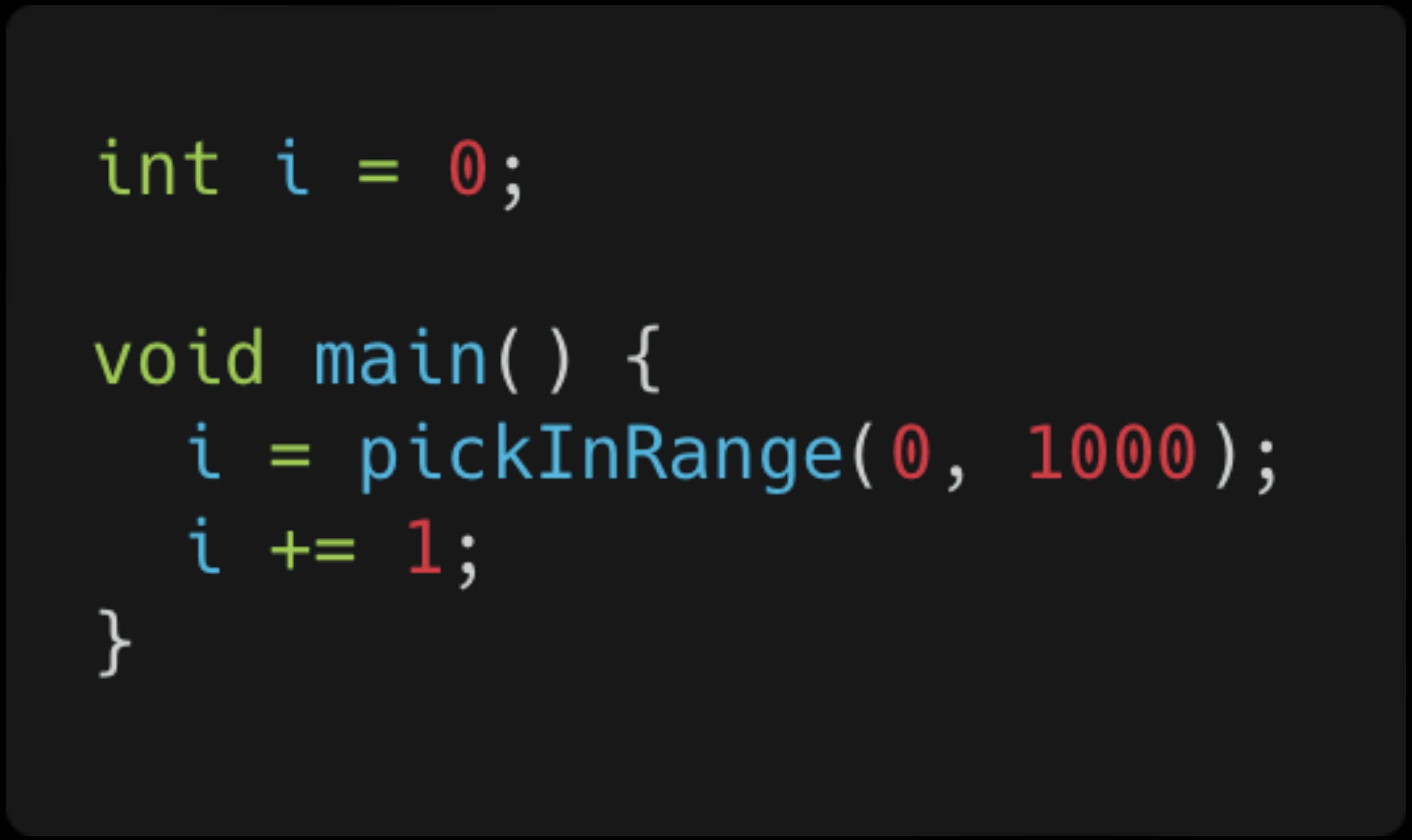
Introduction to TLA+

Sources:
[lamport2021tla]
[wlaschin2020tla]

<table>
<tr><th>Programming</th><th>Logic</th><th>TLA+</th></tr>
<tr><td>and, &&</td><td>$\land$</td><td>/\</td></tr>
<tr><td>or, ||</td><td>$\lor$</td><td>\/</td></tr>
<tr><td>not, !, ~</td><td>$\neg$</td><td>~</td></tr>
<tr><td>!=, <>, ~=</td><td>$\neq$</td><td>#, /=</td></tr>
<tr><td>all()</td><td>$\forall$</td><td>\A</td></tr>
<tr><td>any()</td><td>$\exists$</td><td>\E</td></tr>
<tr><td>Set union</td><td>$\cup$</td><td>\cup, \union</td></tr>
<tr><td>Set intersection</td><td>$\cap$</td><td>\cap, \intersect</td></tr>
<tr><td></td><td>$\implies$</td><td>“=>”</td></tr>
</table>

Math Refresher

Sources: [lamport2021tla]

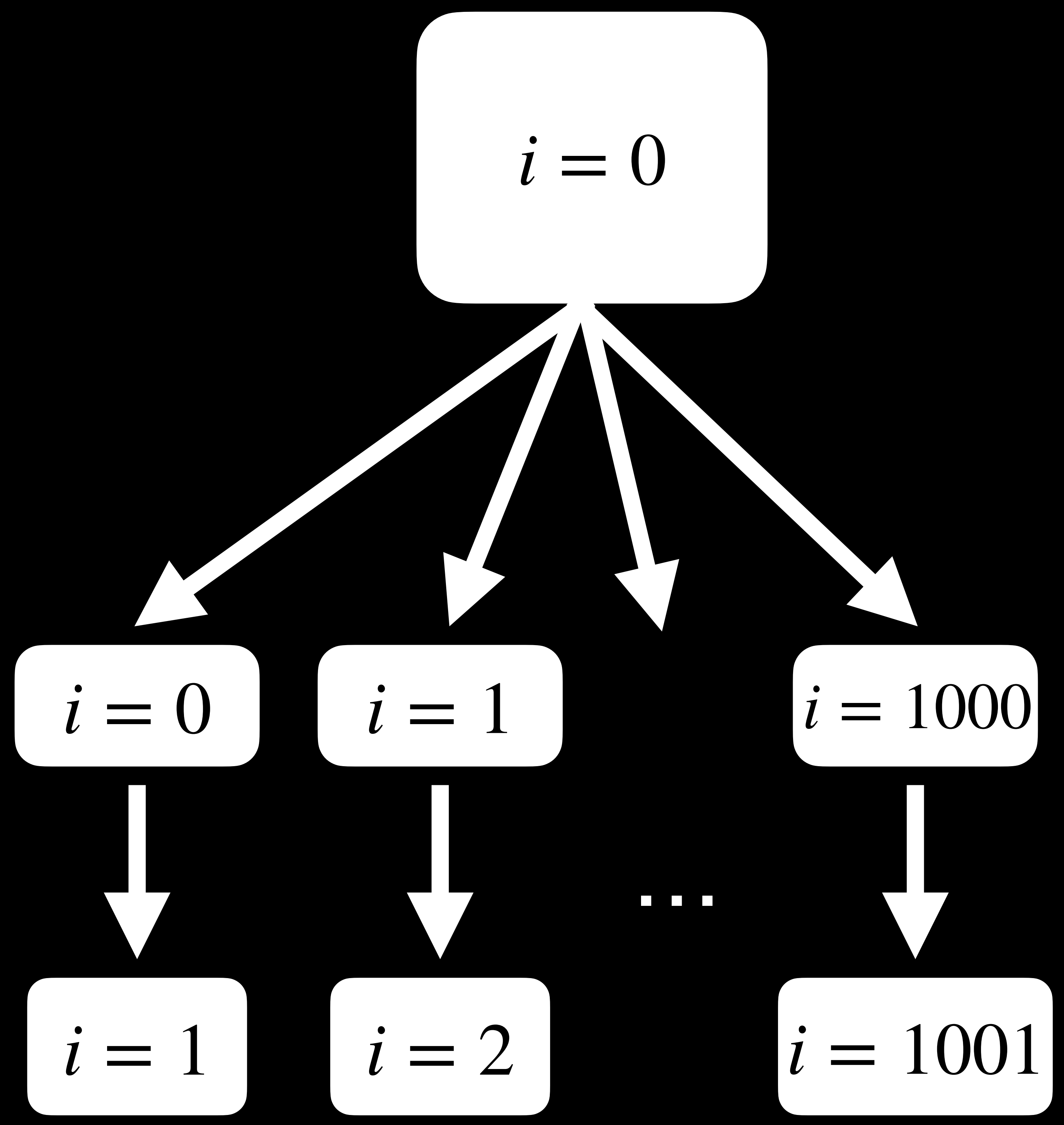


```
int i = 0;

void main() {
  i = pickInRange(0, 1000);
  i += 1;
}
```

Some Code

Sources:
[lamport2002systems]
[wlaschin2020tla]

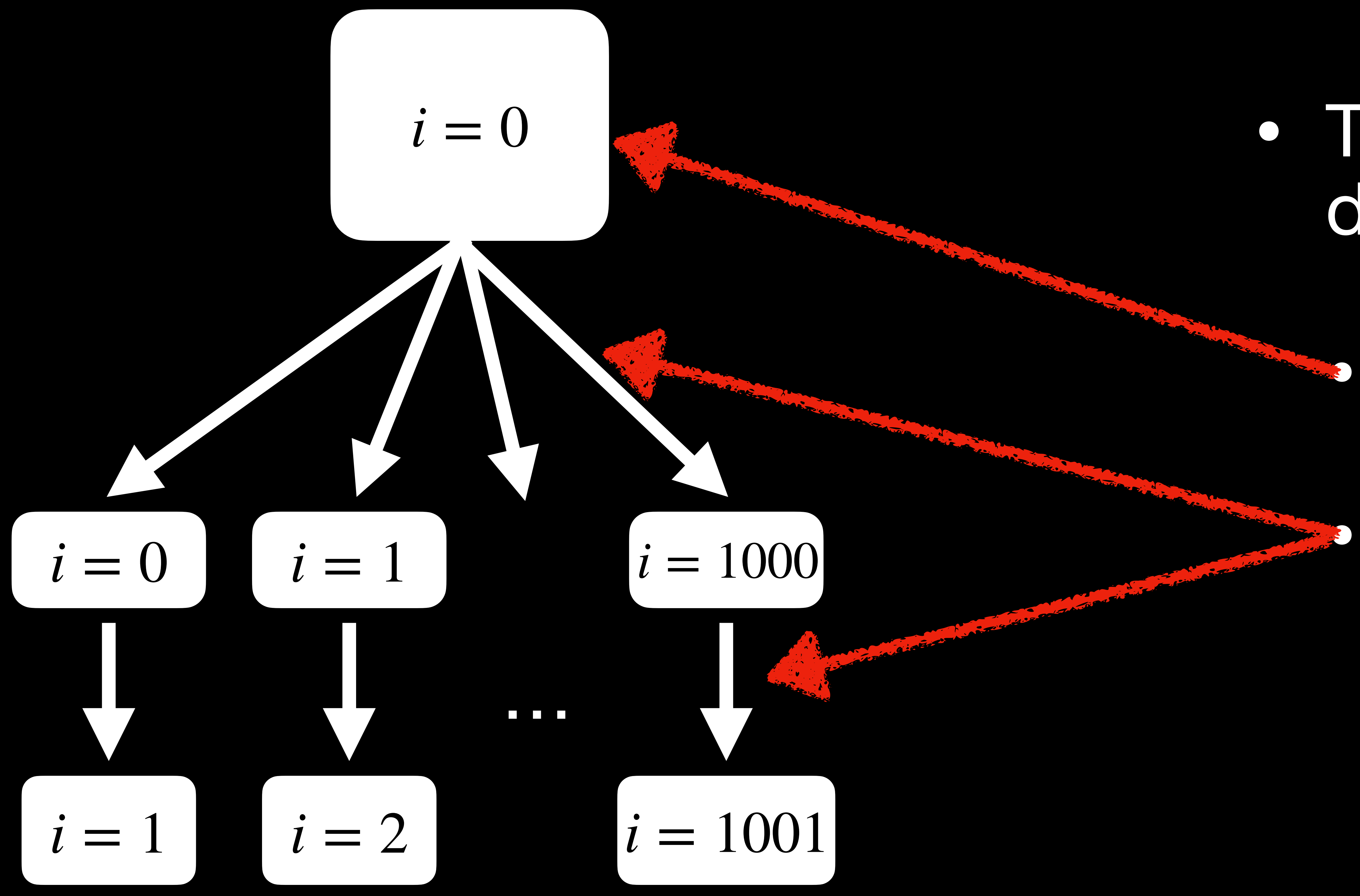
TLA+ models Programs as State Machines



- TLA+ follows mathematical convention of defining state machines using
  - *Init* as initial state
  - *Next* as state transition function for all states

Sources: [wlaschin2020tla] [lamport2002systems]

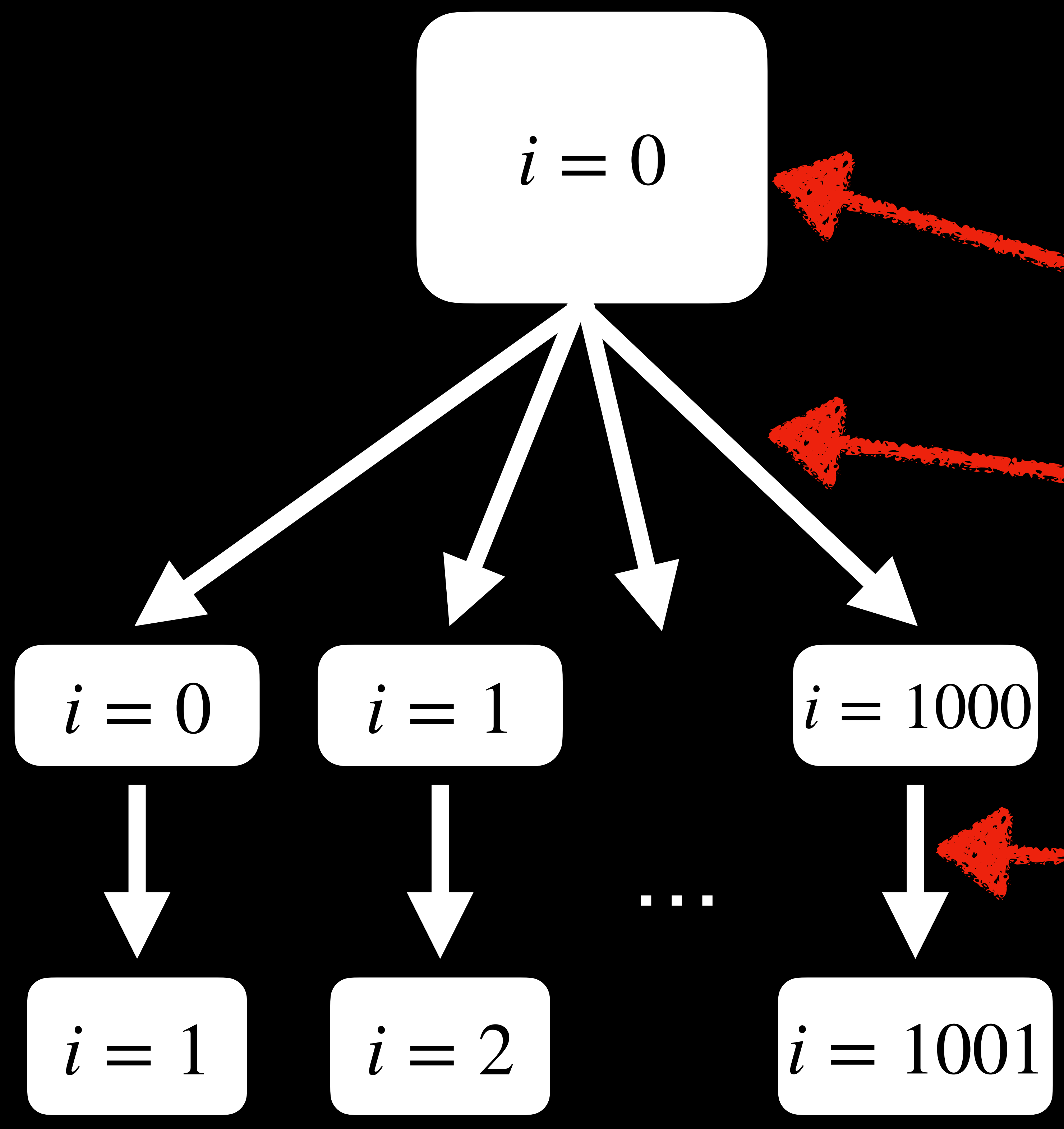
TLA+ models Programs as State Machines



- TLA+ follows mathematical convention of defining state machines using
    - $Init$ as initial state
    - $Next$ as state transition function for all states

Sources: [wlaschin2020tla] [lamport2002systems]

TLA+ models Programs as State Machines

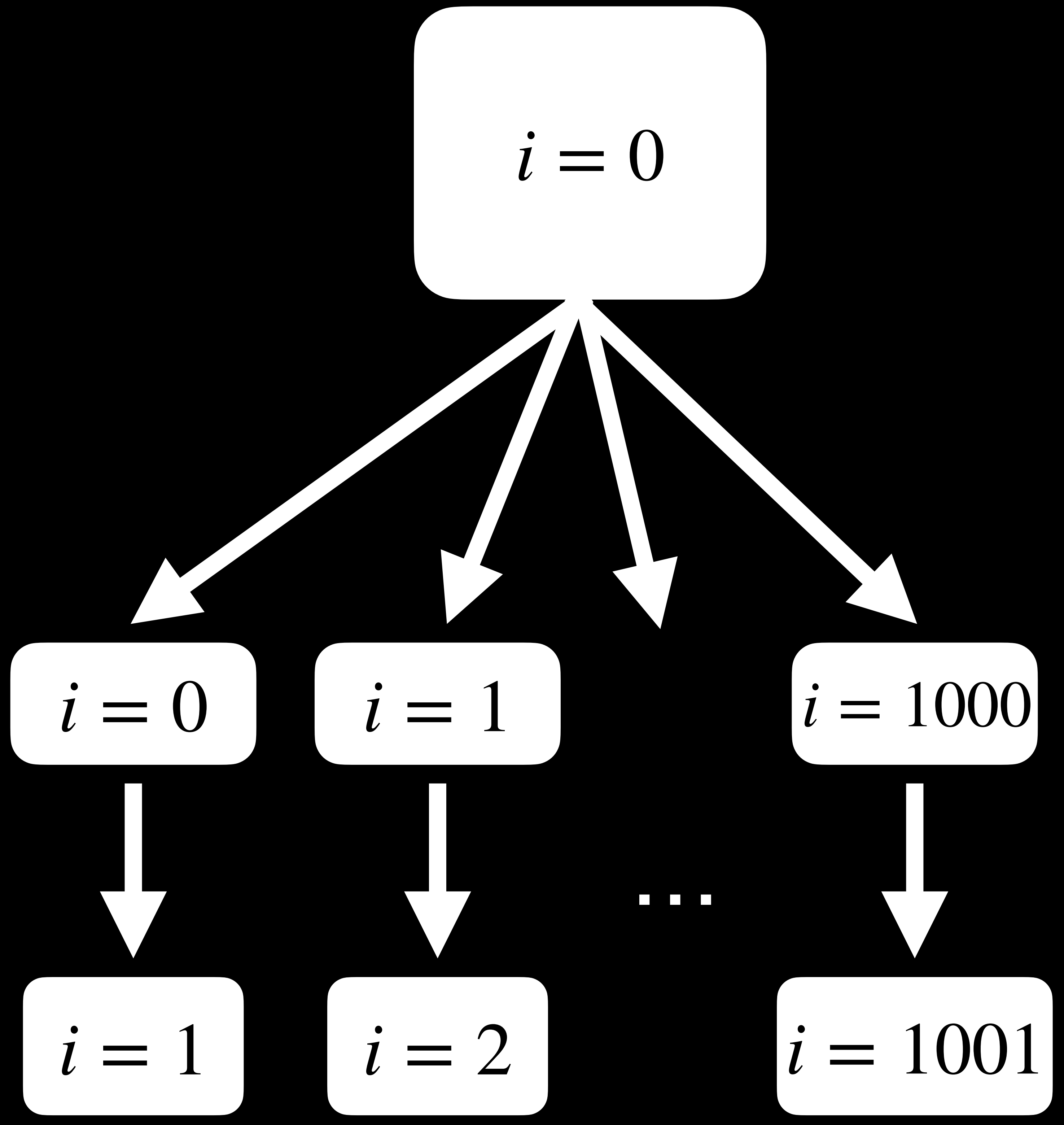


- TLA+ follows mathematical convention of defining state machines using
  - Init == (pc = “start”) ∧ (i = 0)
  - Next:

    ```
    Next == ∨ (pc = “start”
                ∧  i’ \in 0..1000
                ∧  pc’ = “middle”)
            ∨ (pc = “middle”
                ∧  i’ = i + 1
                ∧  pc’ = “done”)
    ```

Sources: [wlaschin2020tla] [lamport2002systems]

TLA+ models Programs as State Machines



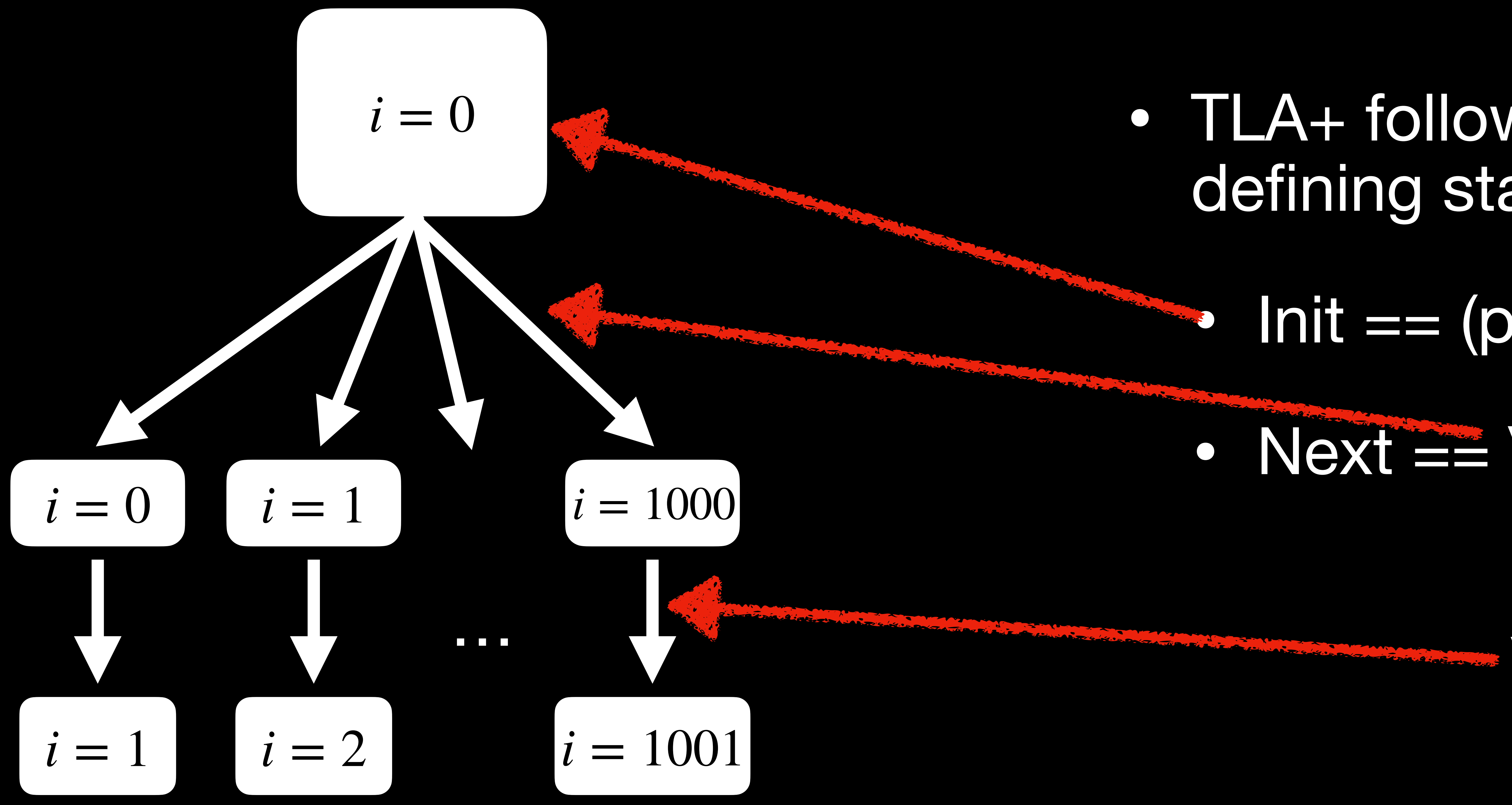
- TLA+ follows mathematical convention of defining state machines using
  - Init == (pc = “start”) ⋀ (i = 0)
  - Next == ⋁ (pc = “start”)

```
                ⋀ i’ \in 0..1000
                ⋀  pc’ = “middle”)
             ⋁ (pc = “middle”
                ⋀  i’ = i + 1
                ⋀  pc’ = “done”)
```

Implicitly define next state variables using ‘

Sources: [wlaschin2020tla] [lamport2002systems]

TLA+ models Programs as State Machines

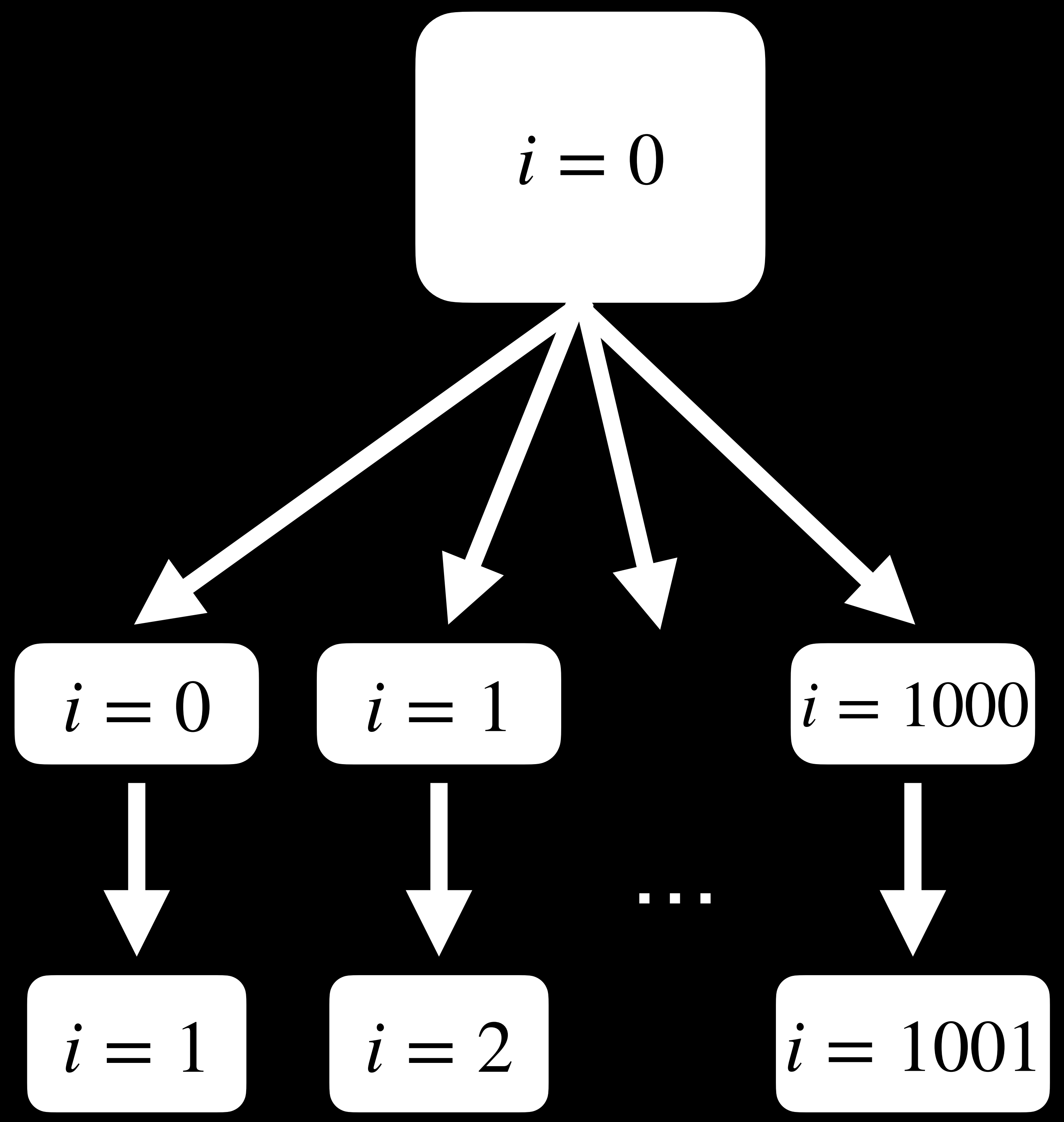


- TLA+ follows mathematical convention of defining state machines using
  - Init == (pc = “start”) ⋀ (i = 0)
  - Next == ⋁ ⋀ pc = “start”

```
Next == ⋁  ⋀  pc = “start”
           ⋀  i’ \in 0..1000
           ⋀  pc’ = “middle”
        ⋁  ⋀  pc = “middle”
           ⋀  i’ = i + 1
           ⋀  pc’ = “done”
```

Sources: [wlaschin2020tla] [lamport2002systems]

TLA+ models Programs as State Machines



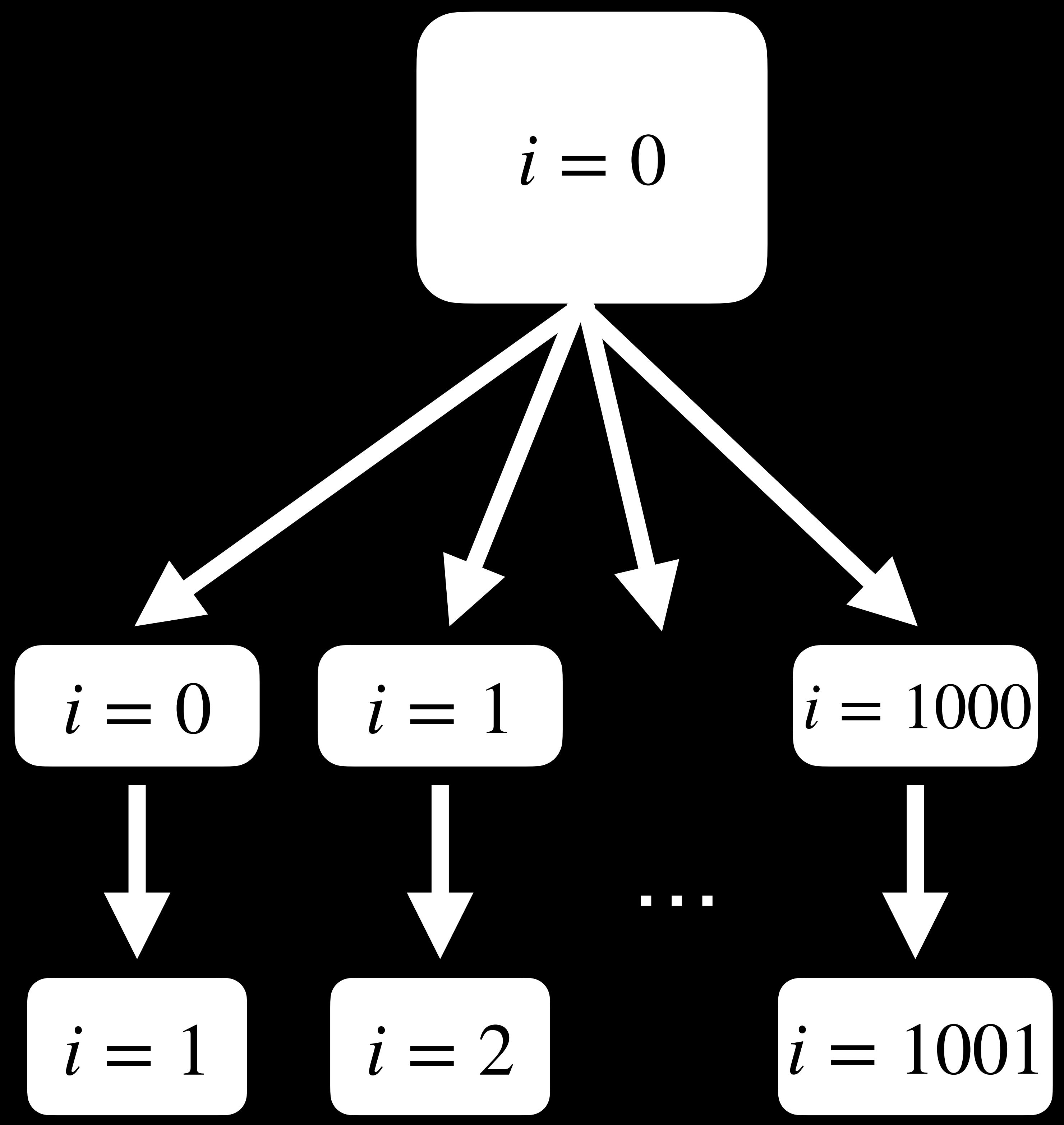
- TLA+ follows mathematical convention of defining state machines using
  - Init == (pc = “start”) ∧ (i = 0)
  - Next ==

```
Next == ∨  ∧  pc = “start”
           ∧  i’ \in 0..1000
           ∧  pc’ = “middle”
        ∨  ∧  pc = “middle”
           ∧  i’ = i + 1
           ∧  pc’ = “done”
```

What happens if multiple clauses match?

Sources: [wlaschin2020tla] [lamport2002systems]

TLA+ models Programs as State Machines



- TLA+ follows mathematical convention of defining state machines using
  - Init == (pc = “start”) ∧ (i = 0)
  - Next == ∨ ∧ pc = “start”

```
Next == ∨  ∧  pc = “start”
           ∧  i’ \in 0..1000
           ∧  pc’ = “middle”
        ∨  ∧  pc = “middle”
           ∧  i’ = i + 1
           ∧  pc’ = “done”
```

What happens if multiple clauses match?
All states are checked!

Sources: [lamport2002systems]

TLA+ checks Programs using Assertions and Invariants

- *Given: set P, integer n, ensure that set P contains only elements smaller than n*
- TLA+ uses invariants and assertions to check correctness
- E.g. in math could specify invariant as: $\forall p \in P : p < n$
  (in TLA+: \A p \in P: p < n)
- This ensures **safety** properties ("what mustn't fail")

Sources: [lamport2002systems]

TLA+ checks Programs using Assertions and Invariants

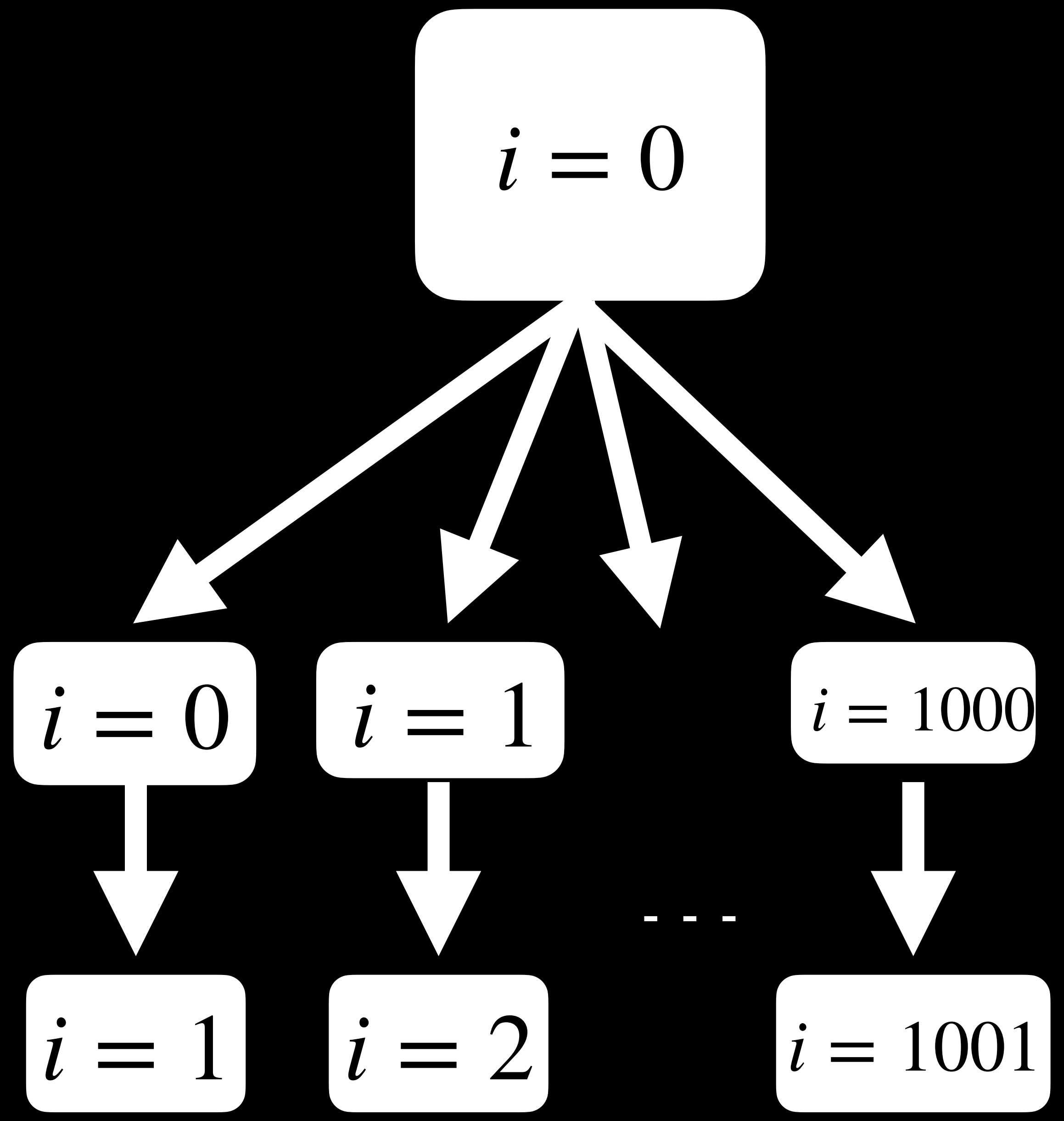
- TLA+ is build on temporal logic
- Temporal logic allows reasoning over time, i.e. reasoning about concurrent programs
- This ensures liveness properties (“Good things happen eventually”)
  - In practice, this covers only a minor part of TLA+ specs

In English	**In TLA+**
Eventually, there will be no elements smaller than n	`<>(\A p \in P : p < n)`
Forever, there will be no elements smaller than n	`[](\A p \in P : p < n)`
At some time, there will be no elements smaller than n ever again	`<>[](\A p \in P: p < n)`

Too unfamiliar?

Sources: [lamport2002systems] [lamport2016pluscal]

PlusCal is a DSL in TLA+ that is more similar to code



- Init == (pc = “start”) /\ (i = 0)
- Next == \/ /\ pc = “start”
  /\ i’ \in 0..1000
  /\ pc’ = “middle”
  \/ /\ pc = “middle”
  /\ i’ = i + 1
  /\ pc’ = “done”

```
(**--algorithm SomeName {
    variable i = 0;

    {
        with (n \in 0..1000) {
            i := n;
        };
        i := i + 1;
    }
}**)
```

Source: [lamport2011euclid]

Euclids Algorithm

```
--fair algorithm Euclids {
    (* declaration of global variables *)
    variables x = M, y = N;
    (* operator definitions *)
    define {
        p | q == \E d \in 1..q : q = d * p

        Divisors(q) == { d \in 1..q : d | q }

        Maximum(S) == CHOOSE n \in S : (\A i \in S: i =< n)

        GCD(p, q) == Maximum(Divisors(p) \cap Divisors(q))
    }
    (* algorithm body or process declarations *)
    {
        while (x # y) {
            if (x < y) {
                y := y - x;
            } else {
                x := x - y;
            }
        };
        print GCD(M,N);
        assert x = GCD(M,N) /\ y = GCD(M,N);
    }
}
```

```
p | q == \E d \in 1..q : q = d * p

Divisors(q) == { d \in 1..q : d | q }

Maximum(S) == CHOOSE n \in S : (\A i \in S: i =< n)

GCD(p, q) == Maximum(Divisors(p) \cap Divisors(q))


vars == << x, y, pc >>

Init == (* Global variables *)
        /\ x = M
        /\ y = N
        /\ pc = "Step"

Step == /\ pc = "Step"
         /\ IF x # y
               THEN /\ IF x < y
                          THEN /\ y' = y - x
                               /\ x' = x
                          ELSE /\ x' = x - y
                               /\ y' = y
                    /\ pc' = "Step"
               ELSE /\ PrintT(GCD(M,N))
                    /\ Assert(x = GCD(M,N) /\ y = GCD(M,N),
                              "Failure of assertion at line 31, column 9.")
                    /\ pc' = "Done"
                    /\ UNCHANGED << x, y >>

(* Allow infinite stuttering to prevent deadlock on termination. *)
Terminating == pc = "Done" /\ UNCHANGED vars

Next == Step \/ Terminating
```

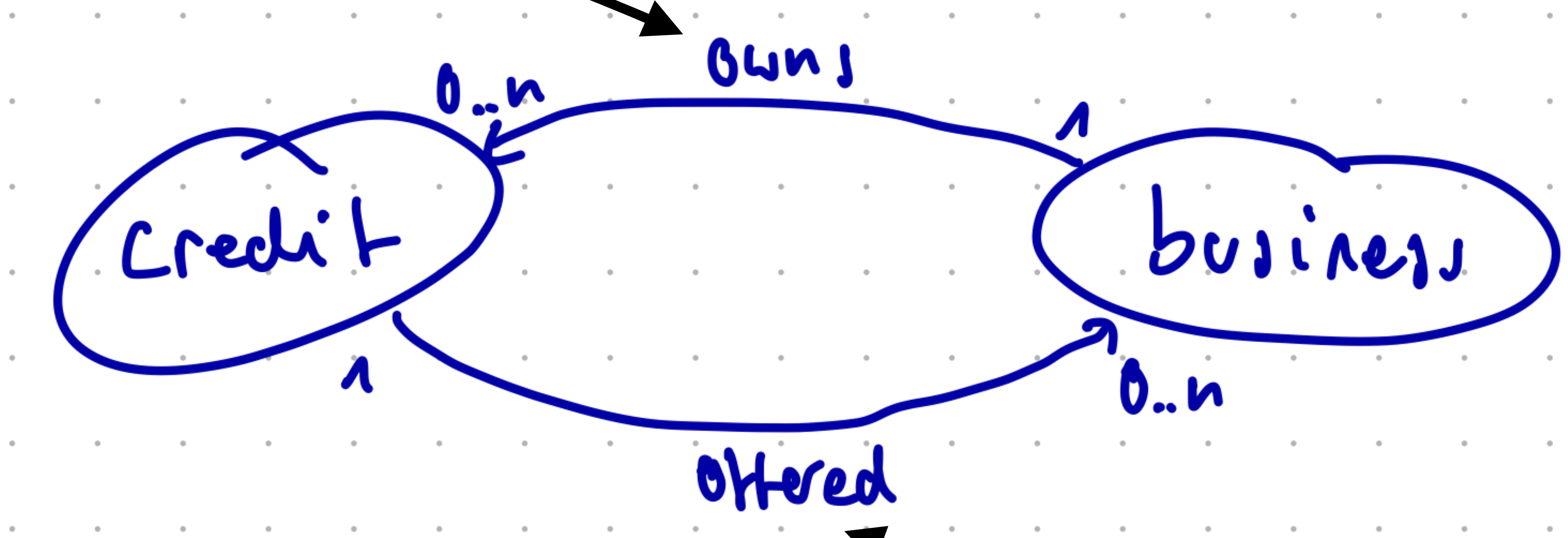
Demo

“Consider a bare minimum carbon credit trading platform:
Every credit has an owner.
An owner may offer the credit to a different user. The recipient user can accept the offer, in which case ownership of the credit transfers to them, or reject it, in which case nothing happens.
The accept/reject is asynchronous: you can offer the same credit to multiple people (to scaffold out trades), and the person may wait a day before accepting or rejecting an offer.”

(Credit: https://www.hillelwayne.com/post/business-case-formal-methods/)

Demo

Design



owners

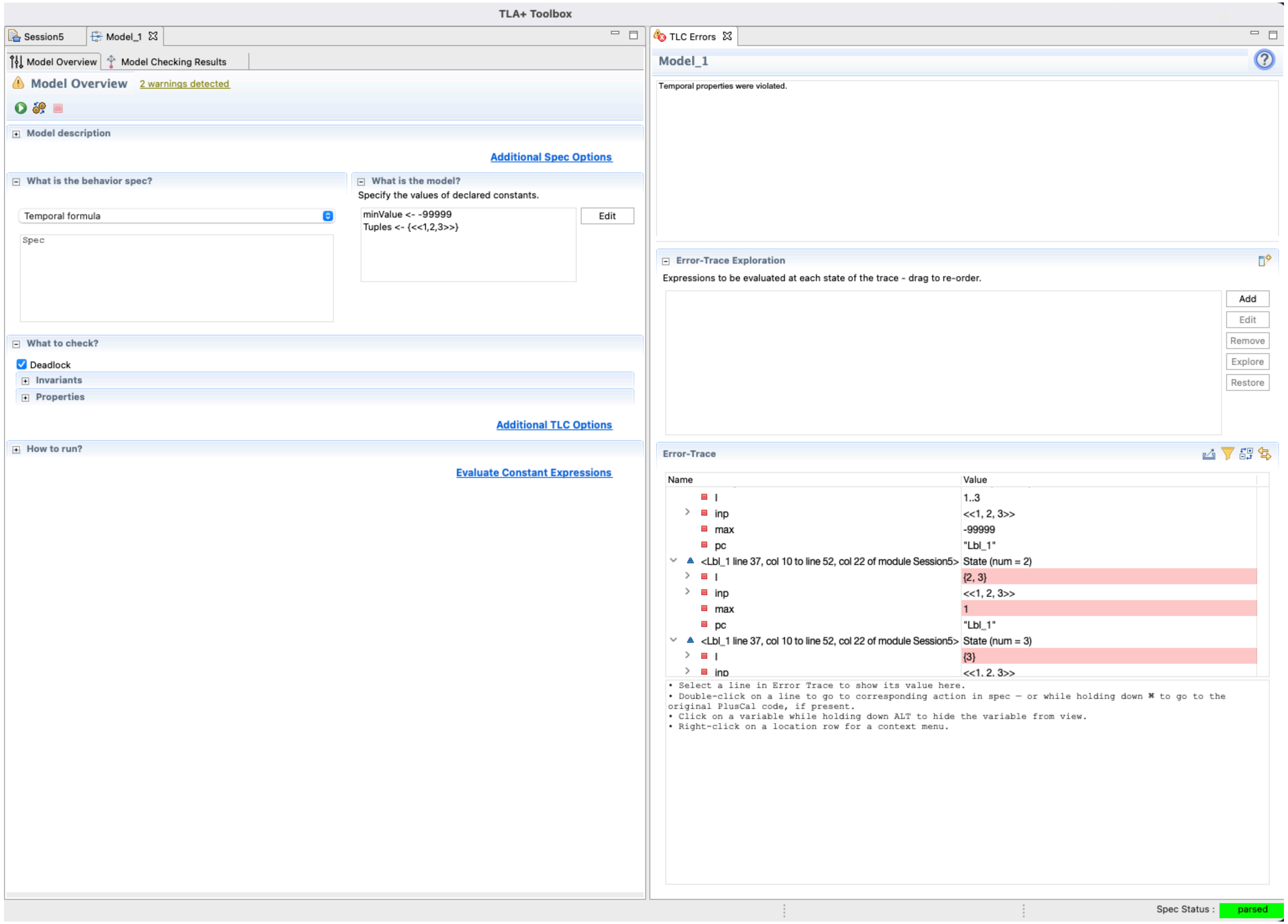
credit	Owner
“C1”	“HPI”
…	…

offers

from	to	credit
“HPI”	“MIT”	“C1”
…	…	…

TLC

TLA+’s model checker



Model Overview 2 warnings detected

Model description

Additional Spec Options

What is the behavior spec?

Temporal formula

Spec

What is the model?

Specify the values of declared constants.

minValue <- -99999
Tuples <- {<<1,2,3>>}

What to check?

☑ Deadlock

Invariants

Properties

Additional TLC Options

How to run?

Evaluate Constant Expressions

Model_1

Temporal properties were violated.

Error-Trace Exploration

Expressions to be evaluated at each state of the trace - drag to re-order.

Error-Trace

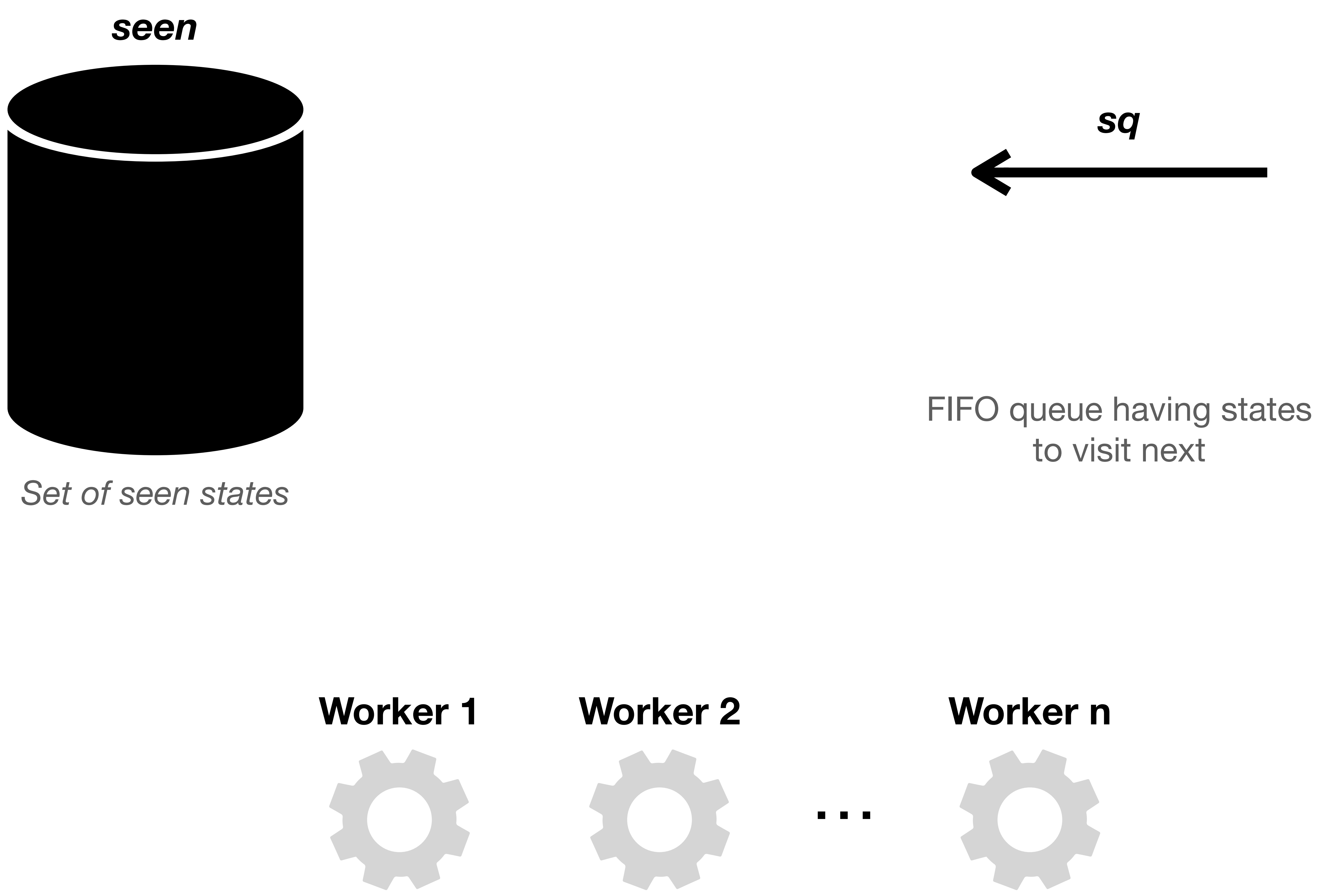
Name	Value
I	1..3
inp	<<1, 2, 3>>
max	-99999
pc	"Lbl_1"
<Lbl_1 line 37, col 10 to line 52, col 22 of module Session5>	State (num = 2)
I	{2, 3}
inp	<<1, 2, 3>>
max	1
pc	"Lbl_1"
<Lbl_1 line 37, col 10 to line 52, col 22 of module Session5>	State (num = 3)
I	{3}
inp	<<1, 2, 3>>

```
• Select a line in Error Trace to show its value here.
• Double-click on a line to go to corresponding action in spec — or while holding down ⌘ to go to the
original PlusCal code, if present.
• Click on a variable while holding down ALT to hide the variable from view.
• Right-click on a location row for a context menu.
```

Sources: [yu1999tlmc]

How TLC’s model checker works

Init



seen

Set of seen states

sq

FIFO queue having states
to visit next

Worker 1 **Worker 2** … **Worker n**

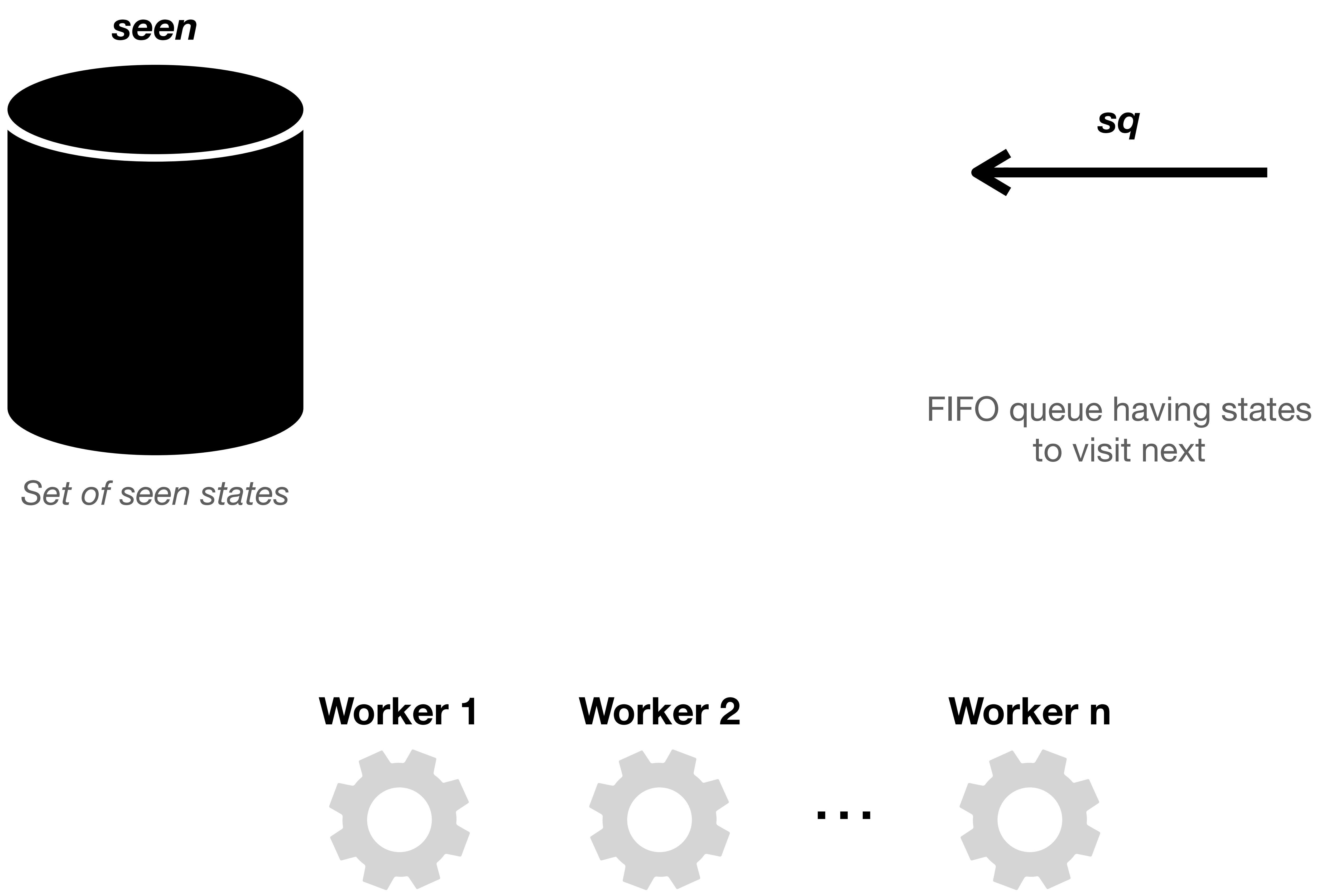
Sources: [yu1999tlmc]

How TLC's model checker works

Init

- Init == (pc = “start”) ∧ (i = 0)
- Next ==

```
Next == ∨ ∧  pc = “start”
          ∧  i’ \in 0..1000
          ∧  pc’ = “middle”
        ∨ ∧  pc = “middle”
          ∧  i’ = i + 1
          ∧  pc’ = “done”
```



seen

Set of seen states

sq

FIFO queue having states to visit next

Worker 1

Worker 2

…

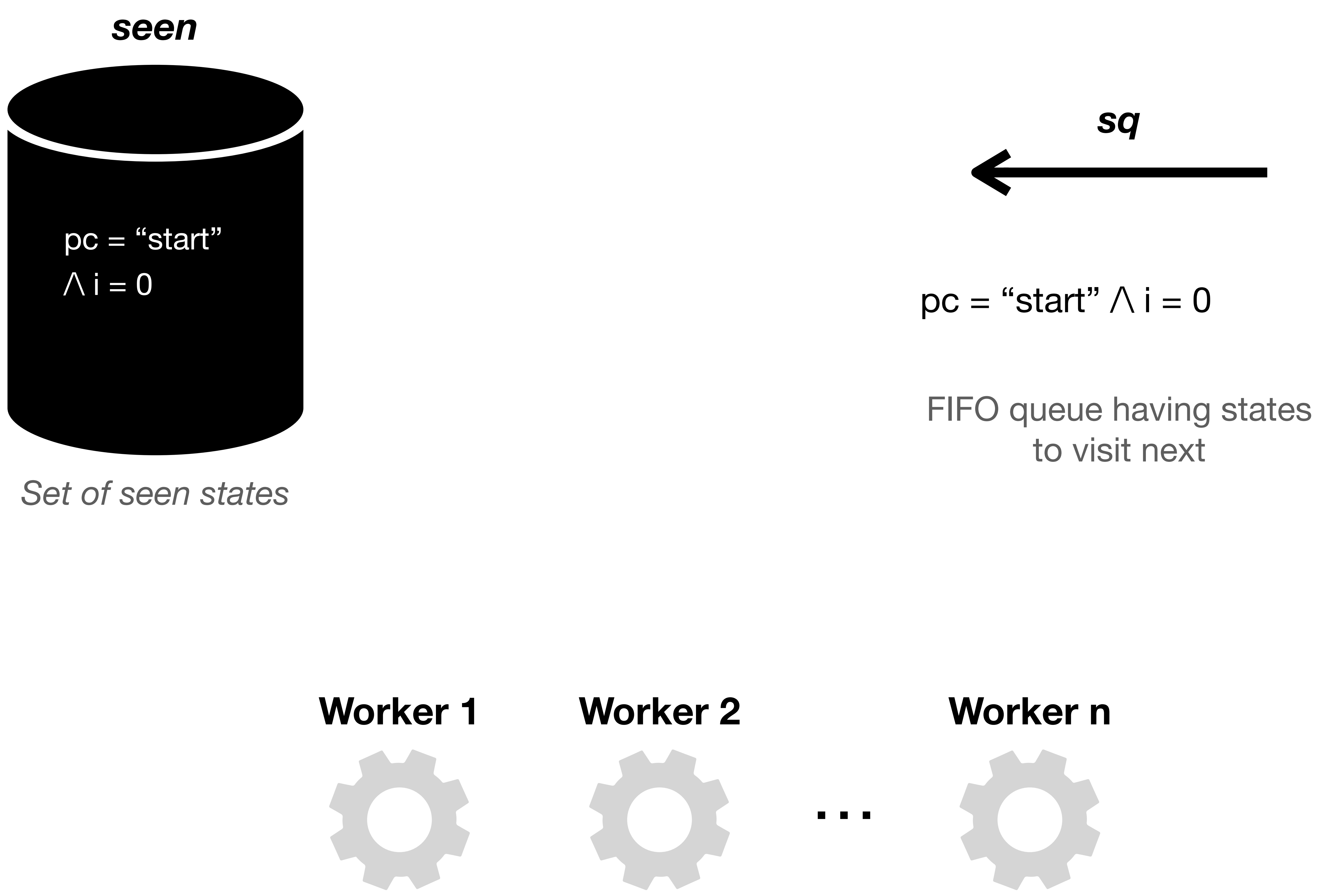
Worker n

Sources: [yu1999tlmc]

How TLC’s model checker works

Init

1. Add all possible init states to *seen* and *sq*



seen

pc = “start”
∧ i = 0

Set of seen states

sq

pc = “start” ∧ i = 0

FIFO queue having states to visit next

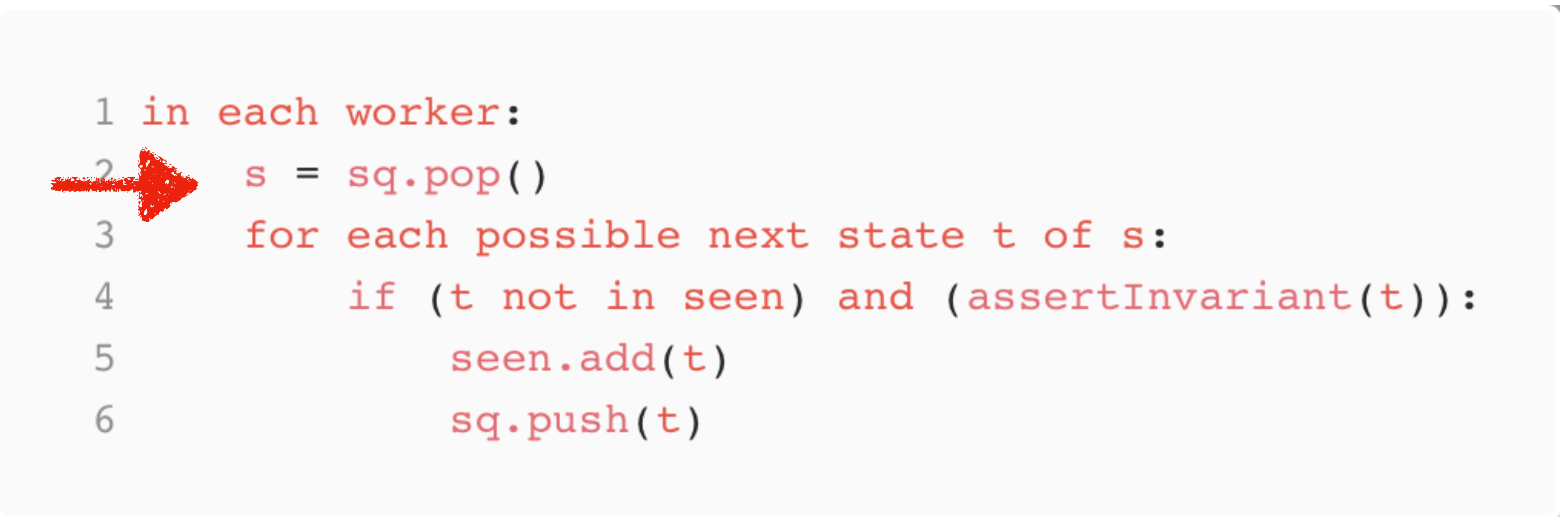
- Init == (pc = “start”) ∧ (i = 0)

- Next == ∨ ∧ pc = “start”

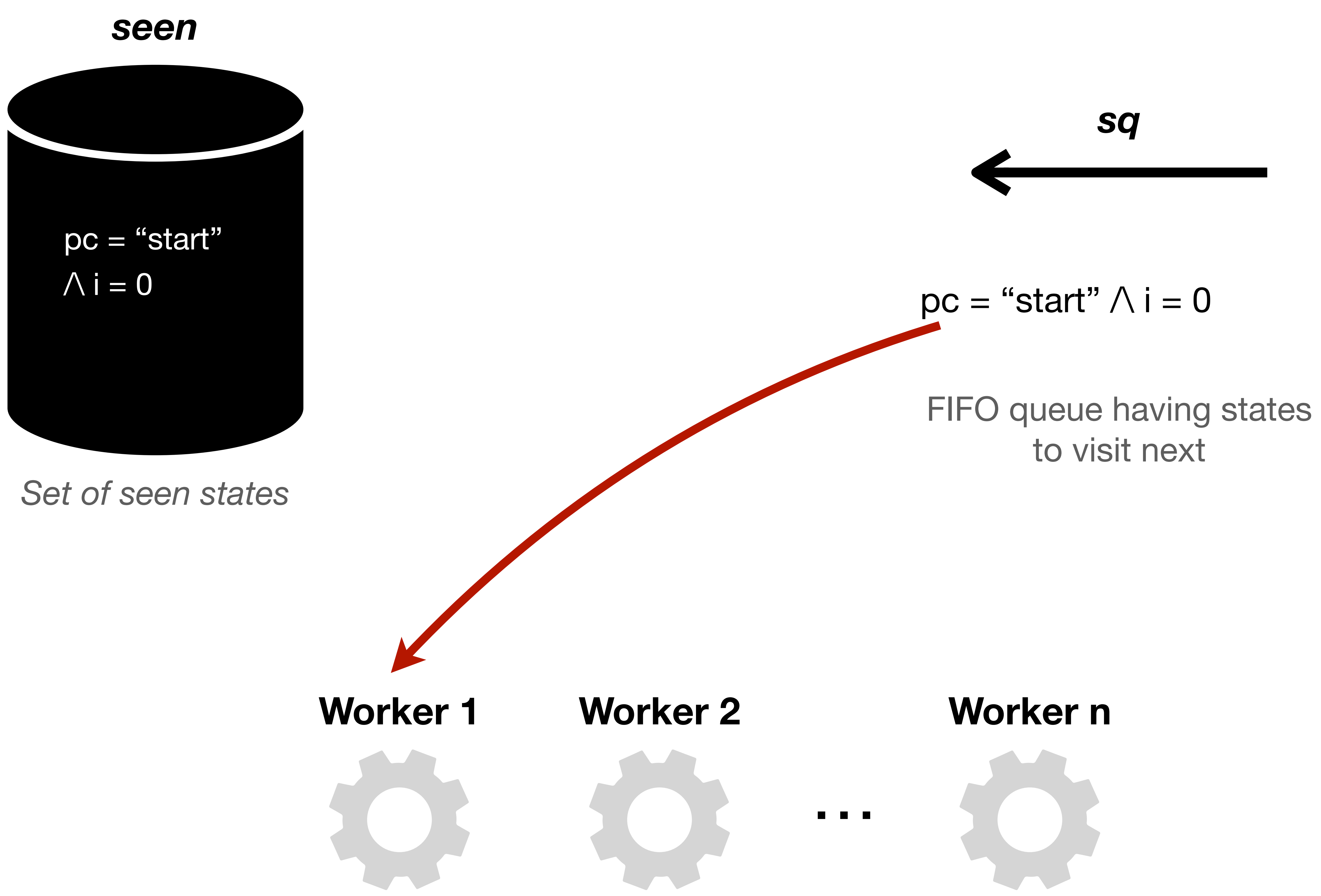
```
Next == ∨ ∧  pc = “start”
          ∧  i’ \in 0..1000
          ∧  pc’ = “middle”
        ∨ ∧  pc = “middle”
          ∧  i’ = i + 1
          ∧  pc’ = “done”
```

Sources: [yu1999tlmc]

How TLC’s model checker works



```
in each worker:
    s = sq.pop()
    for each possible next state t of s:
        if (t not in seen) and (assertInvariant(t)):
            seen.add(t)
            sq.push(t)
```



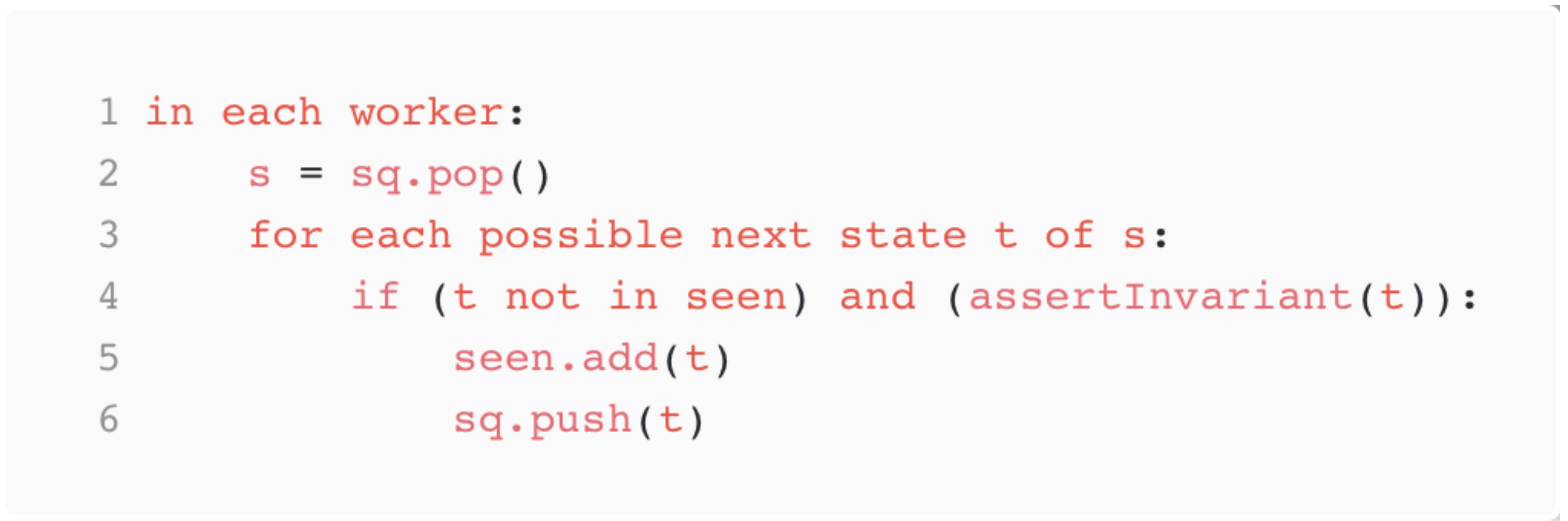
Set of seen states

FIFO queue having states to visit next

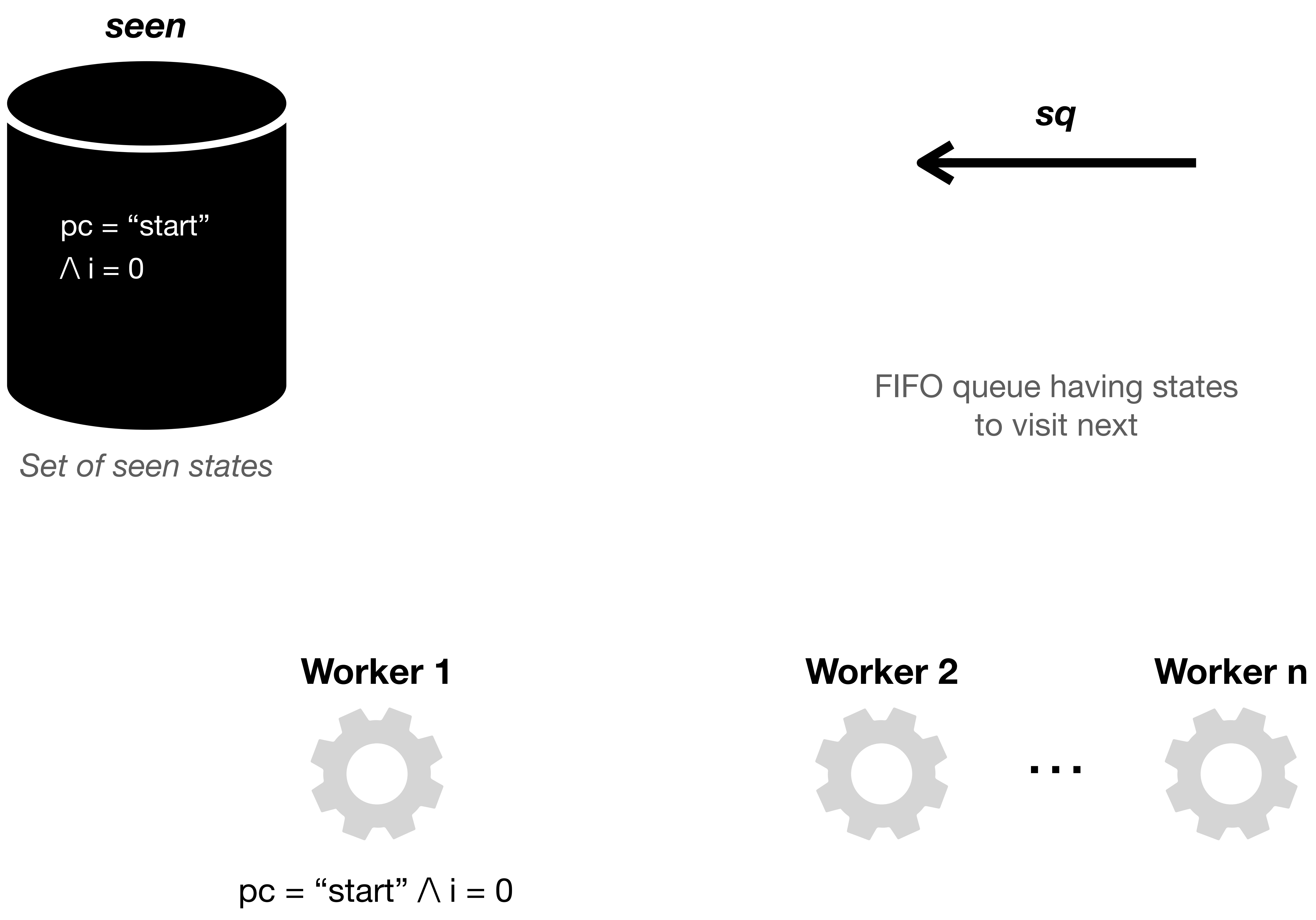
- Init == (pc = “start”) ⋀ (i = 0)
- Next == ⋁ ⋀ pc = “start”
             ⋀ i’ \in 0..1000
             ⋀ pc’ = “middle”
           ⋁ ⋀ pc = “middle”
             ⋀ i’ = i + 1
             ⋀ pc’ = “done”

Sources: [yu1999tlmc]

How TLC's model checker works



```
in each worker:
    s = sq.pop()
    for each possible next state t of s:
        if (t not in seen) and (assertInvariant(t)):
            seen.add(t)
            sq.push(t)
```



Set of seen states

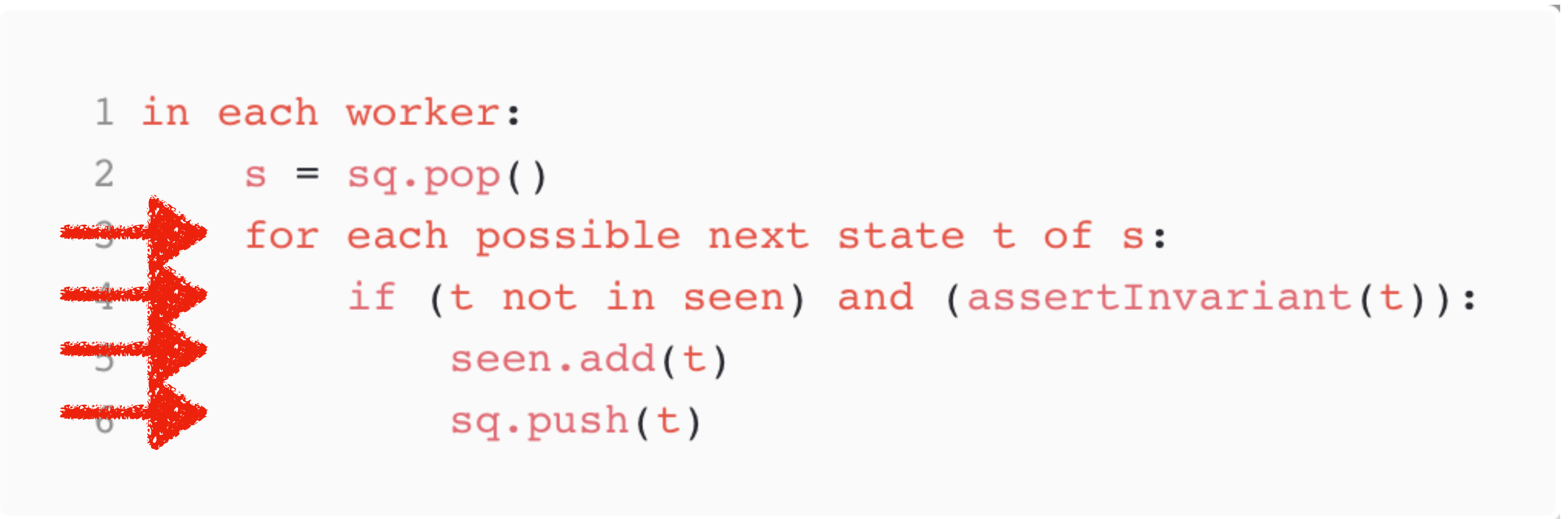
FIFO queue having states to visit next

- Init == (pc = "start") /\ (i = 0)
- Next == \/ /\ pc = "start"
  /\ i' \in 0..1000
  /\ pc' = "middle"
  \/ /\ pc = "middle"
  /\ i' = i + 1
  /\ pc' = "done"

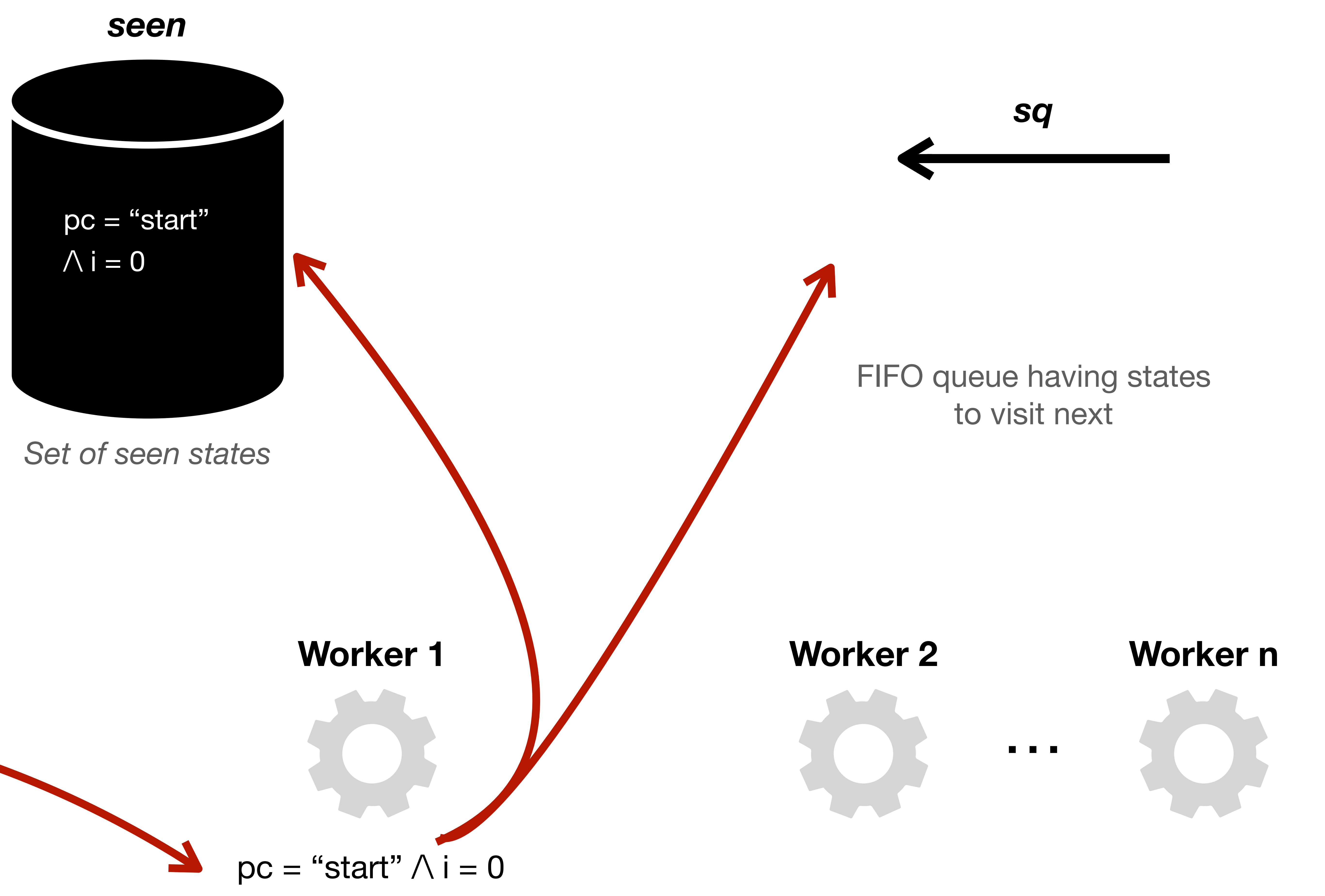
Sources: [yu1999tlmc]

How TLC's model checker works

Lines 3-6



```
1 in each worker:
2     s = sq.pop()
3     for each possible next state t of s:
4         if (t not in seen) and (assertInvariant(t)):
5             seen.add(t)
6             sq.push(t)
```

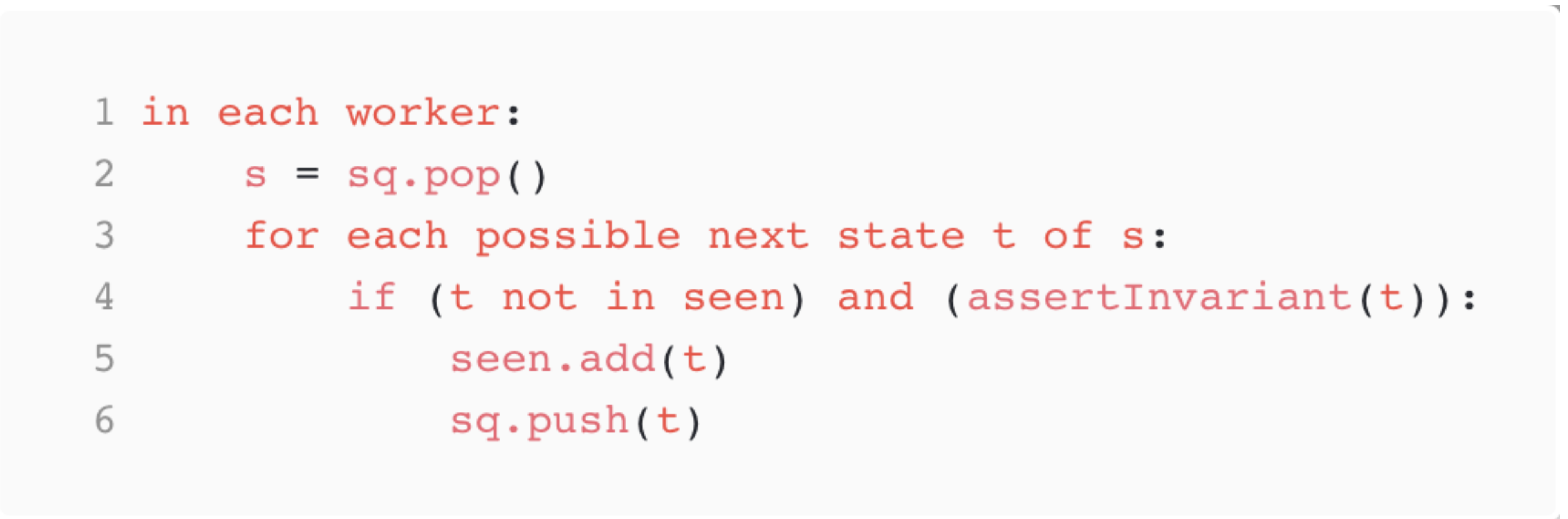


- Init == (pc = "start") ∧ (i = 0)
- Next == ∨ ∧ pc = "start"
                 ∧ i' \in 0..1000
                 ∧ pc' = "middle"
            ∨ ∧ pc = "middle"
                 ∧ i' = i + 1
                 ∧ pc' = "done"

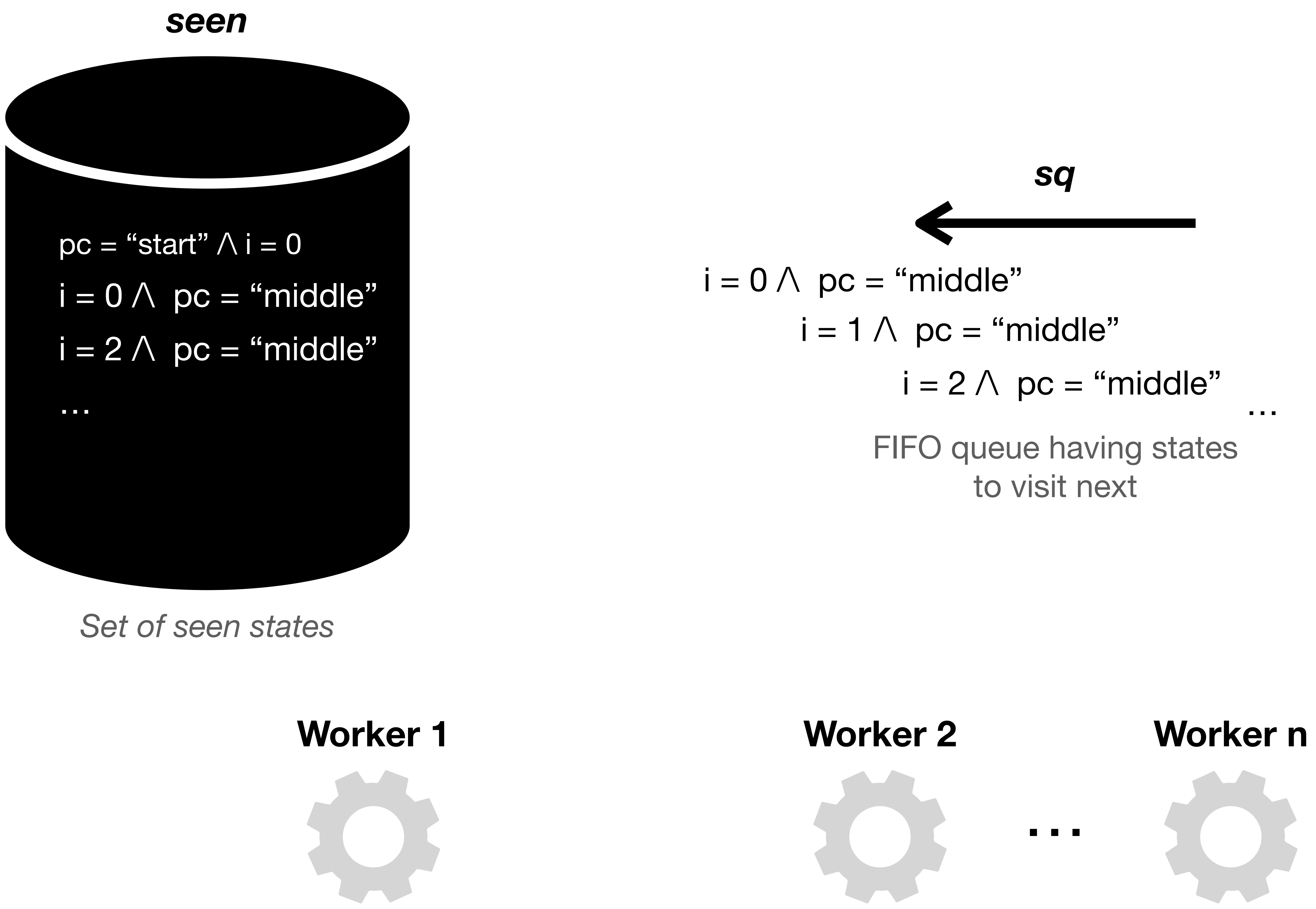
Sources: [yu1999tlmc]

How TLC’s model checker works

Init



```
in each worker:
    s = sq.pop()
    for each possible next state t of s:
        if (t not in seen) and (assertInvariant(t)):
            seen.add(t)
            sq.push(t)
```



seen

pc = “start” ⋀ i = 0
i = 0 ⋀ pc = “middle”
i = 2 ⋀ pc = “middle”
…

Set of seen states

sq

i = 0 ⋀ pc = “middle”
i = 1 ⋀ pc = “middle”
i = 2 ⋀ pc = “middle”
…

FIFO queue having states
to visit next

Worker 1

Worker 2

…

Worker n

- Init == (pc = “start”) ⋀ (i = 0)

- Next == ⋁ ⋀ pc = “start”
                 ⋀ i’ \in 0..1000
                 ⋀ pc’ = “middle”
             ⋁ ⋀ pc = “middle”
                 ⋀ i’ = i + 1
                 ⋀ pc’ = “done”

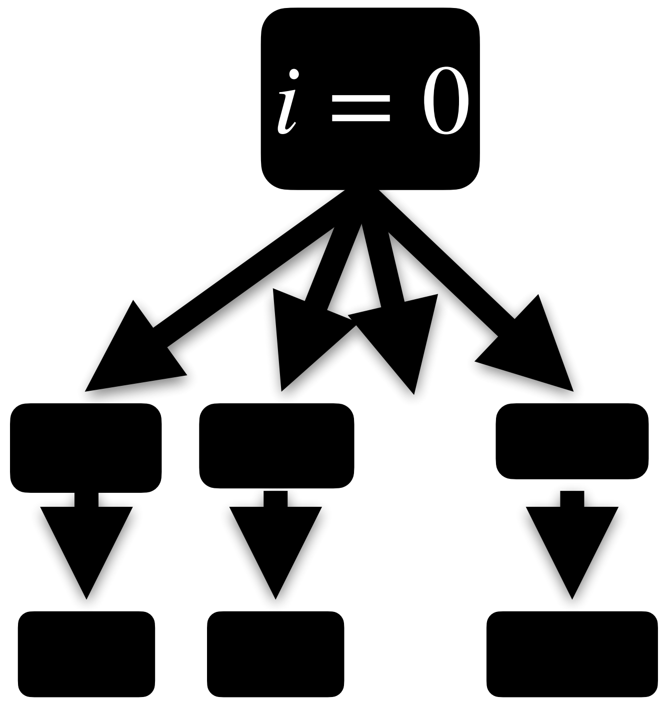
Limitations

Source: [learntla]

Theoretical Limitations

- No floats
- No support for probabilistic systems
- No chars
- Untyped
- No modeling of real-time properties
- Implementation Gap

State machine



Mathematical simplicity

`Set theory, predicate logic`
`[LamportTypes]`

Temporal Logic

`<>[](\A p \in P: p < n)`

Practical Limitations

- Learning curve
  - Amazon estimates 1 month [newcombe2015aws]
  - Intel estimates 3 months [batson2003intel]
- State explosion must be considered and gives upper bound for complexity of TLA+ specs
- Aged Tooling (no code completion, minimal linting, older eclipse-version of toolbox)
- Fragmentation between PlusCal (supports so-called C and P syntax) / TLA+ makes problem searching harder

```
(**--algorithm Init
    variable i

    begin
        with (n \in 0..1000) begin
            i := n
        end with
        i := i + 1
    end algorithm
*)
```

PlusCal P-syntax

```
(**--algorithm SomeName {
    variable i = 0;

    {
        with (n \in 0..1000) {
            i := n;
        };
        i := i + 1;
    }
}**)
```

PlusCal C-syntax

- Init == (pc = “start”) ⋀ (i = 0)
-
```
Next == ⋁ ⋀ pc = “start”
          ⋀ i’ \in 0..1000
          ⋀ pc’ = “middle”
        ⋁ ⋀ pc = “middle”
          ⋀ i’ = i + 1
          ⋀ pc’ = “done”
```

TLA+ syntax

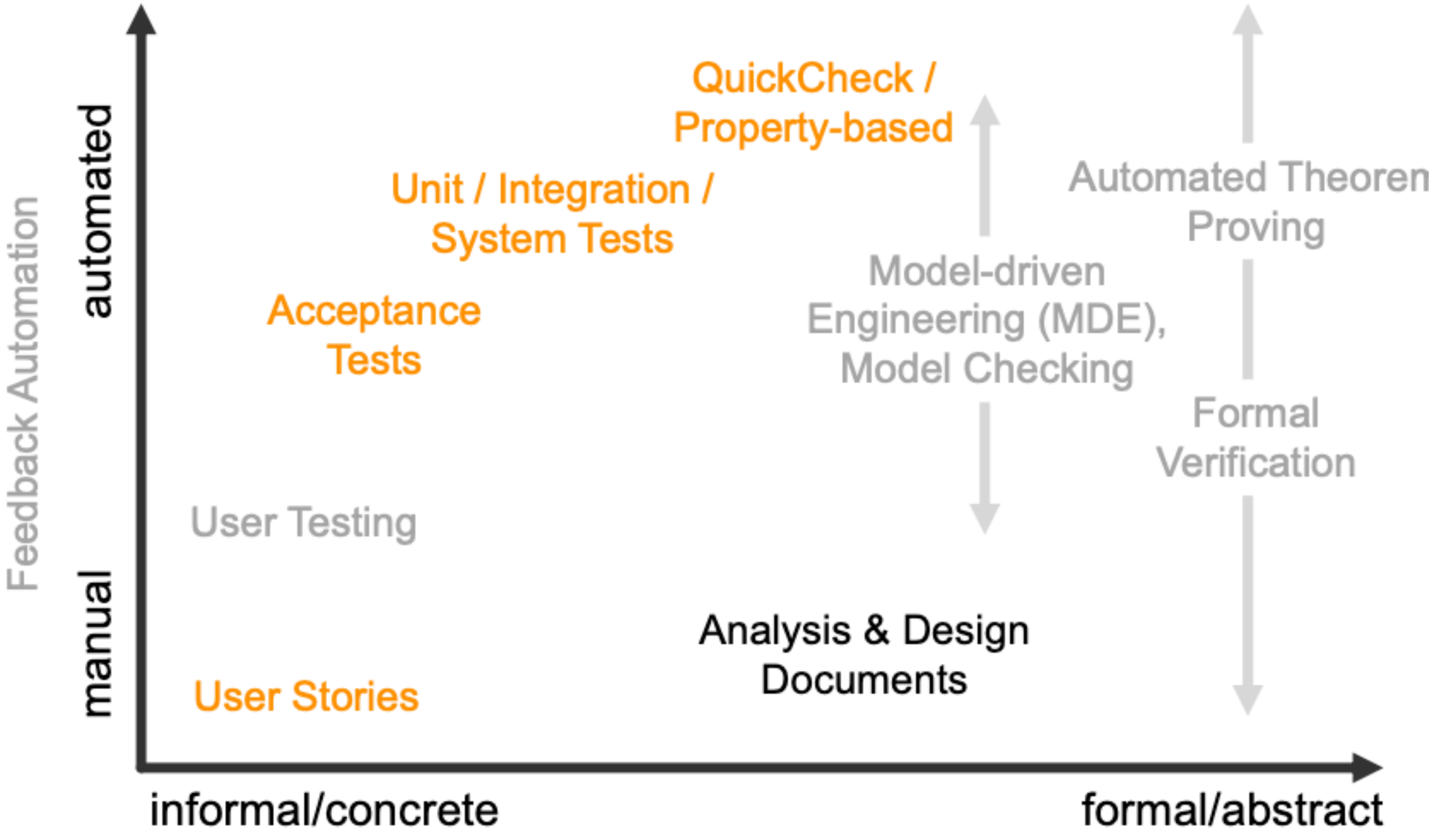
Alternatives

Source: [learntla]

TLA+ vs Testing

- Tests check implementation, TLA+ checks design
- Example from [learntla]:
  - *“our microservice architecture never submits the same payment twice, even if services go down”*
  - Very hard to test thoroughly
  - Easy to find concurrency-related bugs in TLA+

Levels of (Executable) Specification



Feedback Automation
automated
manual
informal/concrete
formal/abstract
QuickCheck / Property-based
Unit / Integration / System Tests
Acceptance Tests
User Testing
User Stories
Automated Theoren Proving
Model-driven Engineering (MDE), Model Checking
Formal Verification
Analysis & Design Documents

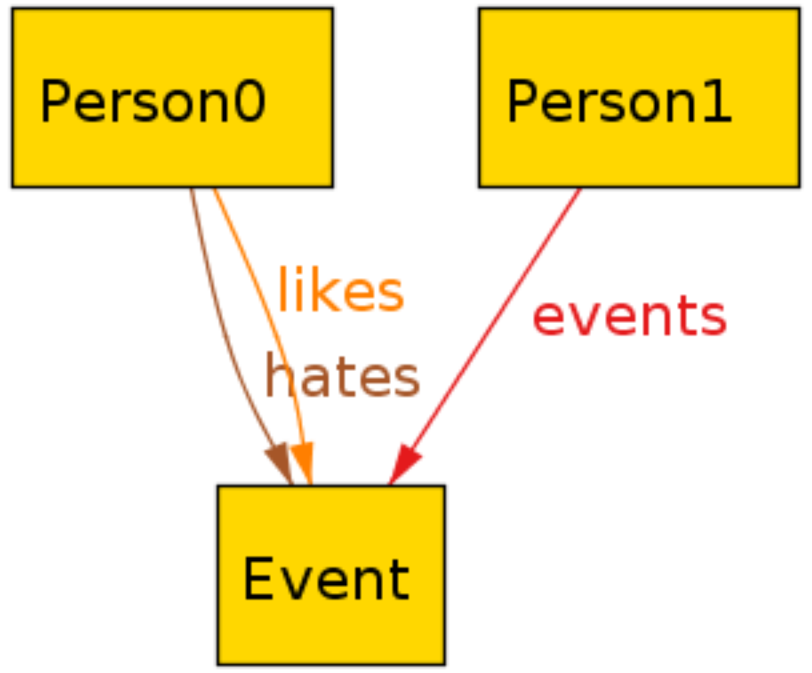
Source: SWA: Vorlesung Test-driven Development (TDD)

Source: [cunha2016alloy]

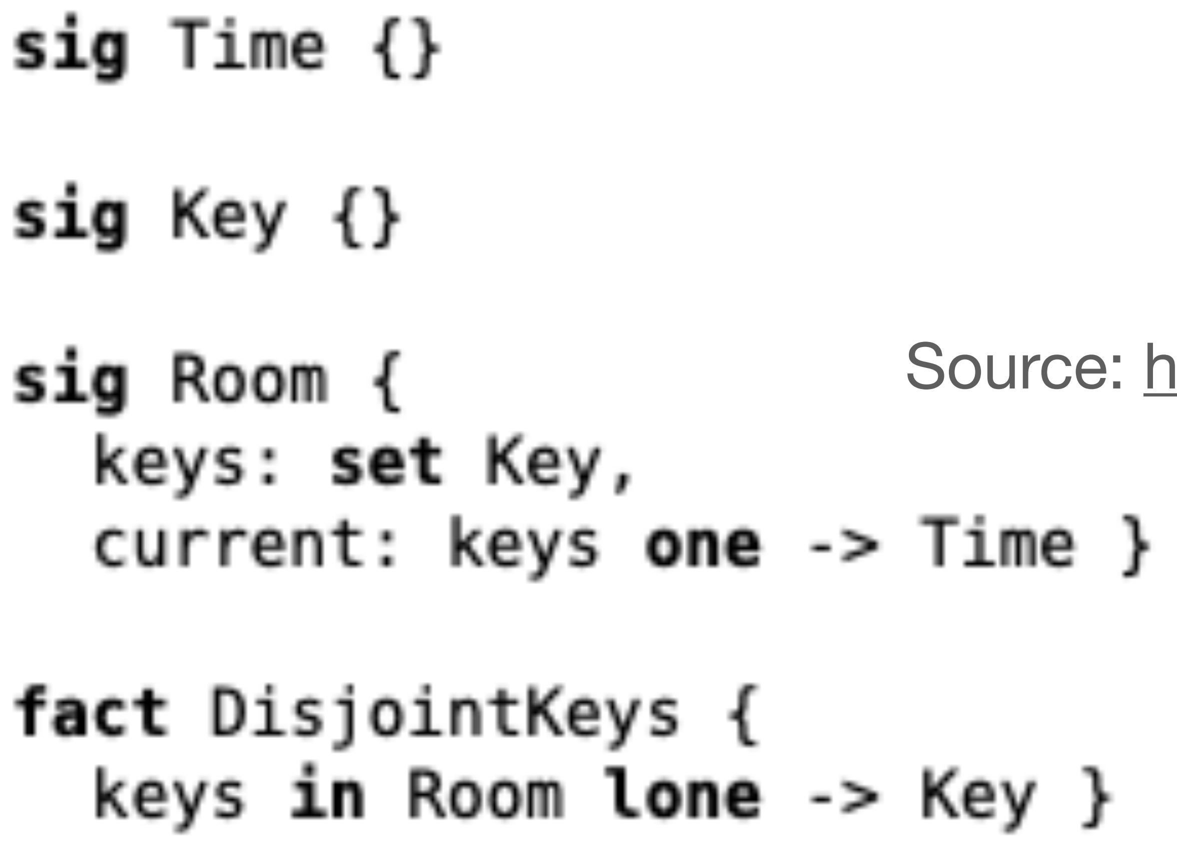
TLA+ vs other formal methods

TLA+ vs Alloy

- Alloy defines a spec, like TLA+
- Alloy was designed to describe the structure of a system
- TLA+ was build for modeling concurrent systems
- Pro Alloy: generally easier to use
- But: Concurrency is more difficult to model

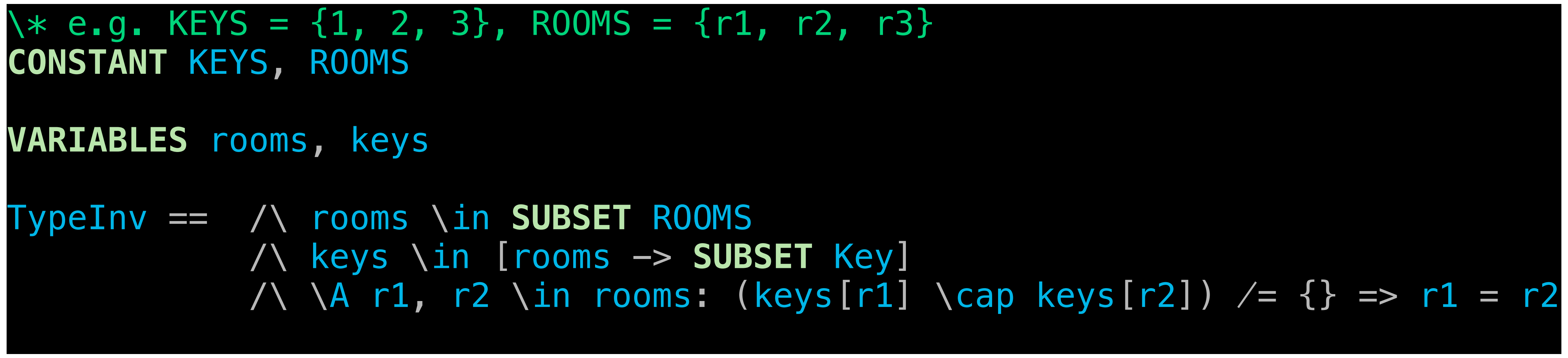


Source: https://www.hillelwayne.com/post/formally-modeling-migrations/



```
sig Time {}

sig Key {}

sig Room {
  keys: set Key,
  current: keys one -> Time }

fact DisjointKeys {
  keys in Room lone -> Key }
```

Source: [cunha2016alloy]



```
\* e.g. KEYS = {1, 2, 3}, ROOMS = {r1, r2, r3}
CONSTANT KEYS, ROOMS

VARIABLES rooms, keys

TypeInv ==  /\ rooms \in SUBSET ROOMS
            /\ keys \in [rooms -> SUBSET Key]
            /\ \A r1, r2 \in rooms: (keys[r1] \cap keys[r2]) /= {} => r1 = r2
```

Equivalent TLA+ as seen in [cunha2016alloy]

Source: [carre1990spark]

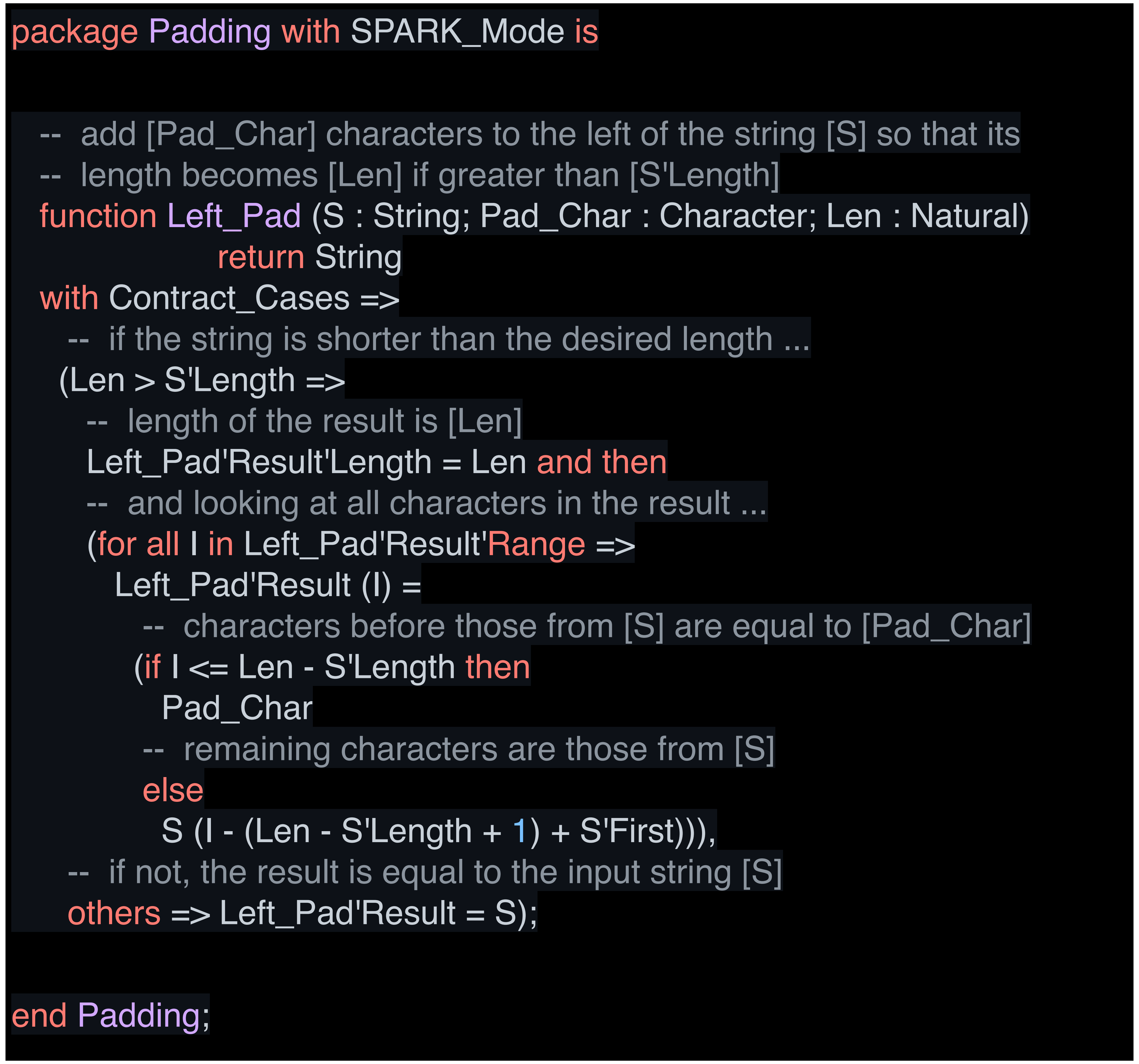
TLA+ vs other formal methods

TLA+ vs Spark

- Spark is a programming language offering capabilities for proving program correctness

- Pro Spark: No implementation gap because it works directly on code

- But: Formally verifying code increases development time drastically (according to [klein2018fm] ~3.3 times of traditional development)



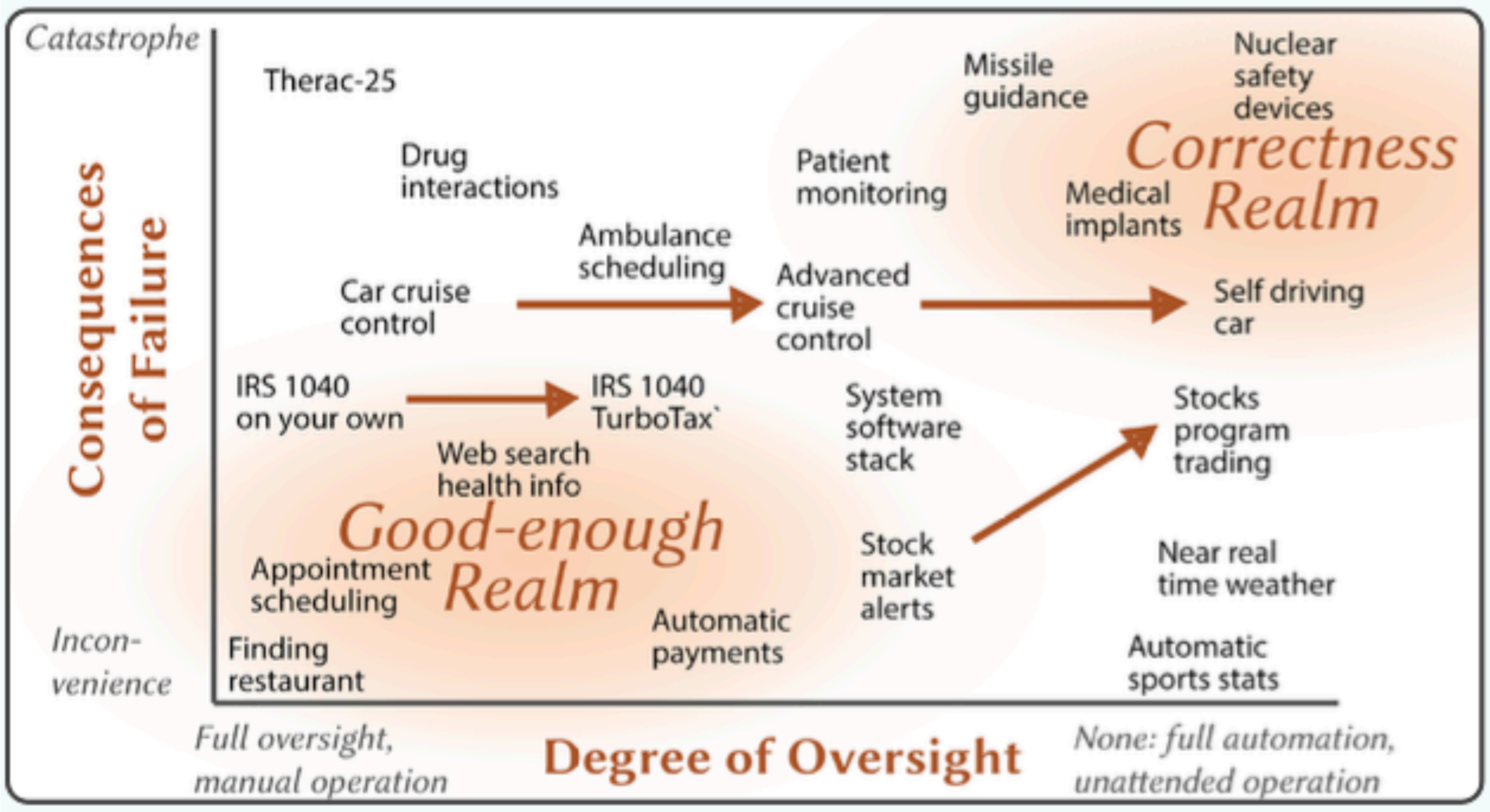
```
package Padding with SPARK_Mode is

  -- add [Pad_Char] characters to the left of the string [S] so that its
  -- length becomes [Len] if greater than [S'Length]
  function Left_Pad (S : String; Pad_Char : Character; Len : Natural)
                    return String
  with Contract_Cases =>
    -- if the string is shorter than the desired length ...
   (Len > S'Length =>
      -- length of the result is [Len]
      Left_Pad'Result'Length = Len and then
      -- and looking at all characters in the result ...
      (for all I in Left_Pad'Result'Range =>
        Left_Pad'Result (I) =
          -- characters before those from [S] are equal to [Pad_Char]
         (if I <= Len - S'Length then
           Pad_Char
          -- remaining characters are those from [S]
          else
           S (I - (Len - S'Length + 1) + S'First))),
    -- if not, the result is equal to the input string [S]
    others => Left_Pad'Result = S);


end Padding;
```

Source: [kanig2019spark]

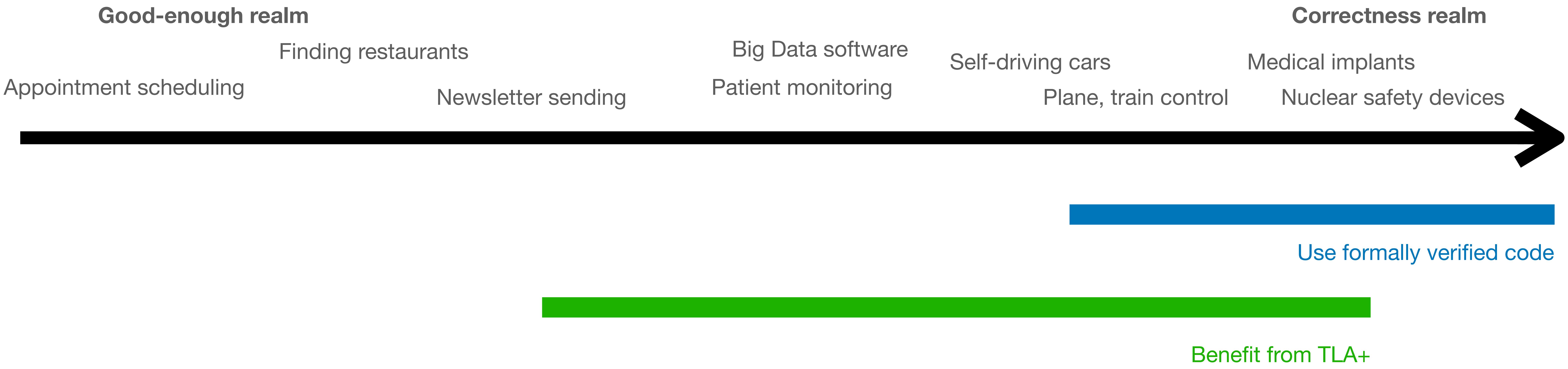
When should you use TLA+?

Source: [shaw2022myths]

“Good Enough is Often Good Enough” [shaw2022myths]



[shaw2022myths], p 22



Good-enough realm

Correctness realm

Appointment scheduling

Finding restaurants

Newsletter sending

Big Data software

Patient monitoring

Self-driving cars

Plane, train control

Medical implants

Nuclear safety devices

Use formally verified code

Benefit from TLA+

<u>Characterized by:</u>

- Software failure would lead to physical harm or sizable financial costs
- You need assurance that your high-level design works
- You are working on a concurrent system

TLA+ is a tradeoff between formally verifying code and development speed

Use TLA+ vs Don’t use TLA+

You need assurance that your high-level design works

You are working on a concurrent system

Software failure would lead to physical harm or sizable financial costs

All states of high-level design can be reasoned about easily

You don’t use concurrency

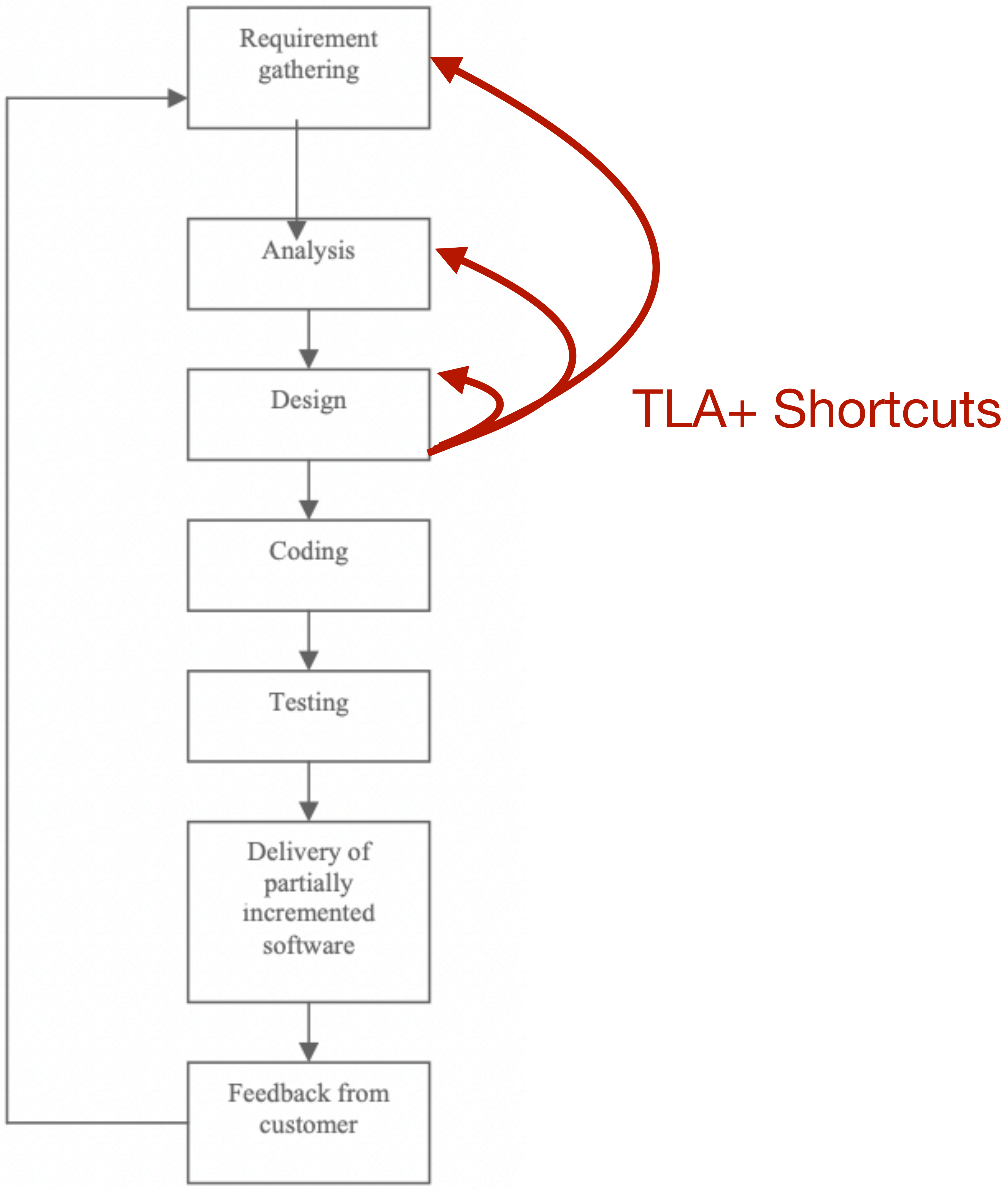
Production bugs do no harm

Your software relies on many probabilistic decisions

Implementation code needs to be formally verified anyhow

Summary

1. Formal specification languages like TLA+ provide a relatively easy way to check high-level concepts
2. It does that earlier in the development cycle than would otherwise be possible
3. As the model checker checks all states of specific cases exhaustively, you gain confidence in the correctness of your idea
4. My recommendation: Use lightweight formal methods like TLA+ whenever you are designing a system which is non-trivial



Agile Dev Cycle [sharma2012agile], adapted to TLA+

Sources

[newcombe2015aws]: Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M., & Deardeuff, M. (2015). *How Amazon web services uses formal methods.* Communications of the ACM, 58(4), 66-73.

[brooks1982month]: Brooks, Frederick P., Jr., 1931-. (1982). The Mythical man-month : essays on software engineering. Reading, Mass. :Addison-Wesley Pub. Co.

[freeman2004design]: Freeman, P., & Hart, D. (2004). A science of design for software-intensive systems. Communications of the ACM, 47(8), 19-21.

[sharma2012agile]: Sharma, S., Sarkar, D., & Gupta, D. (2012). Agile processes and methodologies: A conceptual study. International journal on computer science and Engineering, 4(5), 892.

[verhulst2011rtos]: Verhulst, E., Boute, R. T., Faria, J. M. S., Sputh, B. H., & Mezhuyev, V. (2011). *Formal Development of a Network-Centric RTOS: software engineering for reliable embedded systems.* Springer Science & Business Media.

[lamport2011euclid]: Lamport, L. (2011). Euclid writes an algorithm: A fairytale. International Journal of Software and Informatics 5, 1-2 (2011), 1.

[batson2003intel]: Batson, B., & Lamport, L. (2003). High-level specifications: Lessons from industry. In Formal Methods for Components and Objects: First International Symposium, FMCO 2002, Leiden, The Netherlands, November 5-8, 2002, Revised Lectures 1 (pp. 242-261). Springer Berlin Heidelberg.

[cunha2016alloy]: Cunha, A., & Macedo, N. Alloy meets TLA: An exploratory study.

[shaw2022myths]: Shaw, M. (2022). Myths and mythconceptions: What does it mean to be a programming language, anyhow?. Proceedings of the ACM on Programming Languages, 4(HOPL), 1-44.

[lamport2002systems]: Lamport, L. (2002). Specifying systems: the TLA+ language and tools for hardware and software engineers.

[lamport2016pluscal]: Lamport, L. (2016). A PlusCal User's Manual.

[klein2018fm]: Klein, G., Andronick, J., Fernandez, M., Kuz, I., Murray, T., & Heiser, G. (2018). Formally verified software in the real world. Communications of the ACM, 61(10), 68-77.

[carre1990spark]: Carré, B., & Garnsworthy, J. (1990, December). SPARK—an annotated Ada subset for safety-critical programming. In Proceedings of the conference on TRI-ADA'90 (pp. 392-402).

[woodcock2009fm]: Woodcock, J., Larsen, P. G., Bicarregui, J., & Fitzgerald, J. (2009). Formal methods: Practice and experience. ACM computing surveys (CSUR), 41(4), 1-36.

[yu1999tlmc]: Yu, Y., Manolios, P., & Lamport, L. (1999). Model checking TLA+ specifications. In Correct Hardware Design and Verification Methods: 10th IFIP WG10. 5 Advanced Research Working Conference, CHARME'99 BadHerrenalb, Germany, September 27–29, 1999 Proceedings 10 (pp. 54-66). Springer Berlin Heidelberg.

Chicago

[merz2020leftpad]: Merz, S., Kuppe, M. (2020) Let's Prove Leftpad - https://github.com/hwayne/lets-prove-leftpad/tree/master/tlaps (last accessed: 22.01.23).

Sources

[newcombe2012groups]: Newcombe, Chris, Google Groups: “Introductions”, https://groups.google.com/g/tlaplus/c/ZJCi-UF31fc/m/Mawvwi6U1CYJ (last accessed 30.01.2023).

[learntla]: Wayne, Hillel, Learn TLA+, https://www.learntla.com/ (last accessed 30.01.2023).

[lamport2019tla]: Lamport, Leslie, Industrial Use of TLA+, https://lamport.azurewebsites.net/tla/industrial-use.html (last accessed 30.01.2023).

[lamport2021tla]: Lamport, Leslie, The TLA+ Video Course, https://lamport.azurewebsites.net/video/videos.html, (last accessed 30.01.2023).

[wlaschin2020tla]: Wlaschin, Scott, (2020) “Building confidence in concurrent code with a model checker”, NDC Oslo 2020.

[kanig2019spark]: Kanig, Johannes, (2019), Let’s Prove Leftpad, https://github.com/hwayne/lets-prove-leftpad/tree/master/SPARK (last accessed 30.01.2023).

Appendix

IMHO: How could this develop?

“The best way to predict the future is to invent it” :-)

- Assumption:
  - Moore’s Law will make state explosions for many complex cases acceptable
  - Engineering is CAD <-> SIM <-> FAB & software engineering today is mostly FAB
- Formal methods give us insight into correctness and liveness properties
- Could they also inform implementation choices? Could they tell us what sorting algorithm would be most suitable? And maybe even give us an implementation?
- Could they detect what we are trying to do, and point us to better concepts? E.g. if “multiple processes change one variable” -> recommend appropriate async strategy (this would be proper CAD)

“Weeks of programming can save hours of planning”

- Unknown

“In other words, it's hard for a human being to look at the relatively short proofs used in standard mathematics and be able to determine whether they are good or not. So we can well imagine that if we had something wonderful that we don't currently have -- namely a great specification (meta)language -- that we might […] know if the specification we write really specifies what we are after.

[…]

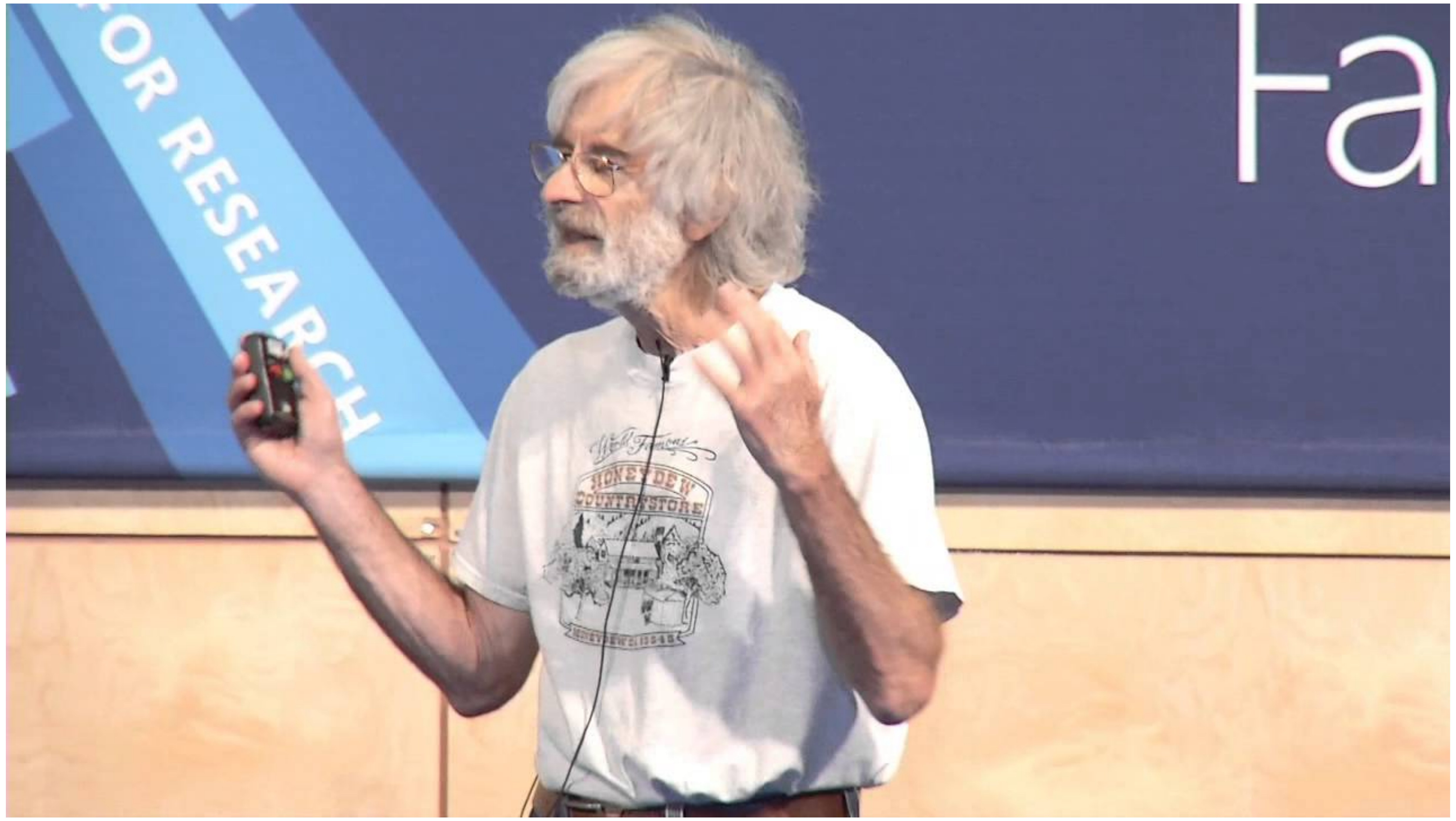
We can now think of the specification system as a kind of "active math" in that it is trying to be more like regular math than a standard programming language, but still has to be run and debugged to test it.”

Alan Kay, on formal methods in programming

https://www.quora.com/Is-using-classical-math-in-computing-a-big-mistake-Is-this-what-most-of-functional-programming-is-all-about

Lamport’s take on Specification by Example

52:40



Lamport’s take on “Coding nowadays is mostly connecting existing software”

55:18