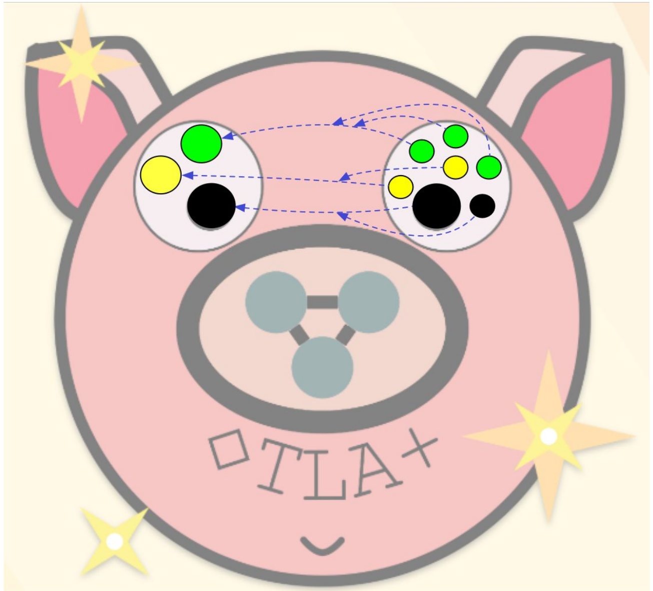
Automating Trace Validation with PGo



Finn Hackett and Ivan Beschastnikh
University of British Columbia

Building and Running Distributed Systems is Notoriously Error-prone

👉 TLA+ helps with this

🤯 Concurrency

🤯 Partial Failure

🤯 Networks

Implementation vs Abstraction in TLA+

The Helpful

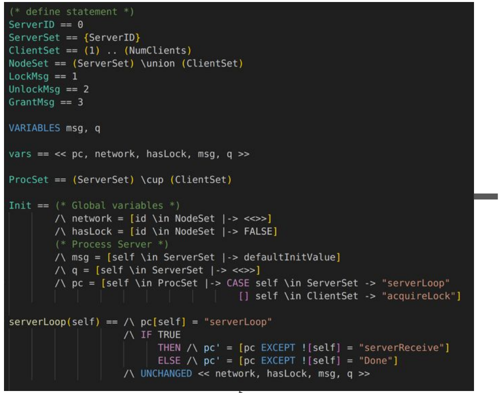
- ✅ Summarize complex behavior into a few state variables and actions
- ✅ Abstraction helps simplify state space for model checking

The Problematic

- ❌ Error-prone relationship with implementation
- ❌ Easy to assume subtly untrue things during modeling
- ❌ Verified models, compiled systematically into implementations, can still fail!
- 👉 Can address with trace validation

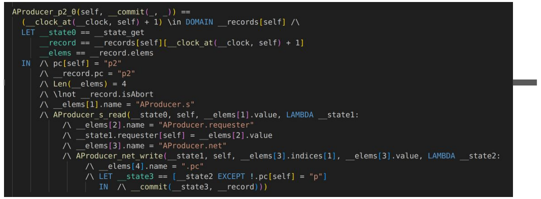
Trace Validation in a Nutshell

TLA+ Spec



```
(* define statement *)
ServerID == 0
ServerSet == {ServerID}
ClientSet == (1) .. (NumClients)
NodeSet == (ServerSet) \union (ClientSet)
LockMsg == 1
UnlockMsg == 2
GrantMsg == 3

VARIABLES msg, q

vars == << pc, network, hasLock, msg, q >>

ProcSet == (ServerSet) \cup (ClientSet)

Init == (* Global variables *)
        /\ network = [id \in NodeSet |-> <<>>]
        /\ hasLock = [id \in NodeSet |-> FALSE]
        (* Process Server *)
        /\ msg = [self \in ServerSet |-> defaultInitValue]
        /\ q = [self \in ServerSet |-> <<>>]
        /\ pc = [self \in ProcSet |-> CASE self \in ServerSet -> "serverLoop"
                                        [] self \in ClientSet -> "acquireLock"]

serverLoop(self) == /\ pc[self] = "serverLoop"
                    /\ IF TRUE
                          THEN /\ pc' = [pc EXCEPT ![self] = "serverReceive"]
                          ELSE /\ pc' = [pc EXCEPT ![self] = "Done"]
                    /\ UNCHANGED << network, hasLock, msg, q >>
```

Hand-written spec <-> log mapping in TLA+



```
AProducer_p2_0(self, __commit(_, _)) ==
  (__clock_at(__clock, self) + 1) \in DOMAIN __records[self] /\
  LET __state0 == __state_get
      __record == __records[self][__clock_at(__clock, self) + 1]
      __elems == __record.elems
  IN  /\ pc[self] = "p2"
      /\ __record.pc = "p2"
      /\ Len(__elems) = 4
      /\ \lnot __record.isAbort
      /\ __elems[1].name = "AProducer.s"
      /\ AProducer_s_read(__state0, self, __elems[1].value, LAMBDA __state1:
           /\ __elems[2].name = "AProducer.requester"
           /\ __state1.requester[self] = __elems[2].value
           /\ __elems[3].name = "AProducer.net"
           /\ AProducer_net_write(__state1, self, __elems[3].indices[1], __elems[3].value, LAMBDA __state2:
                /\ __elems[4].name = ".pc"
                /\ LET __state3 == [__state2 EXCEPT !.pc[self] = "p"]
                   IN  /\ __commit(__state3, __record)))
```

Impl log

"readMsg" ← *read*(.pc)
[type ↦ "B"] ← *read*(network, 1)
write(msg) ← [type ↦ "B"]
[type ↦ "B"] ← *read*(msg)
write(.pc) ← "processB"
commit()

Check w/ TLC: do they match?

Related Work: Specification Compilers, Trace Validation

TLA+ Spec Compilers

Erla+ [Erlang’24], PGo [ASPLOS’23]

Trace Validation (manual)

Confidential Consortium Framework [NSDI’25], etcd [Github’24],
Validating Traces of Distributed Programs [SEFM‘24],
eXtreme Modelling [VLDB’20]

Specification-guided Validation

Multi-grained Specifications / Conformance Checking [EuroSys’25],
SandTable [EuroSys’25], Mocket [EuroSys’23]

Beyond Manual Trace Validation

All existing trace validation implementations involve significant manual work.
Want trace validation to be more accessible.

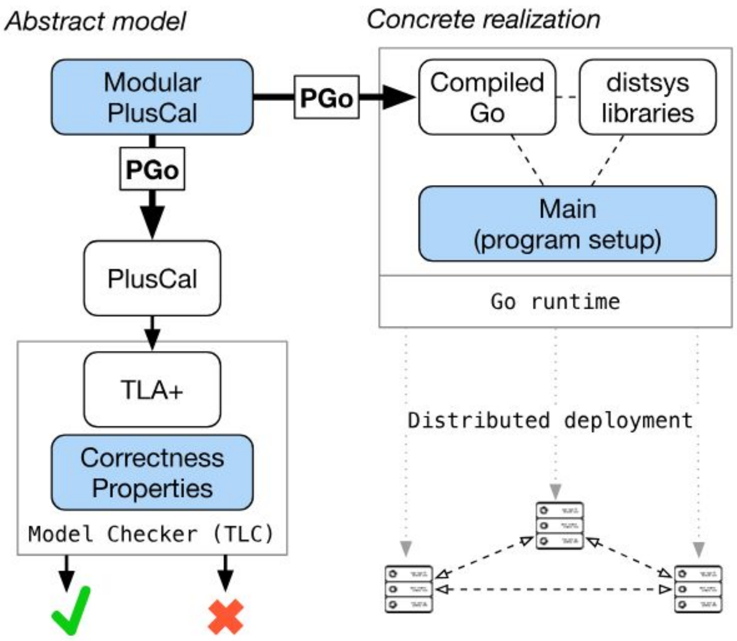
🤔 How much of the action semantics in related work can we automate?

🤔 Can we help auto-instrument the implementation too?

💡 We have the **PGo compiler**, can that help?

Automating Trace Validation with TraceLink

PGo and How it Helps

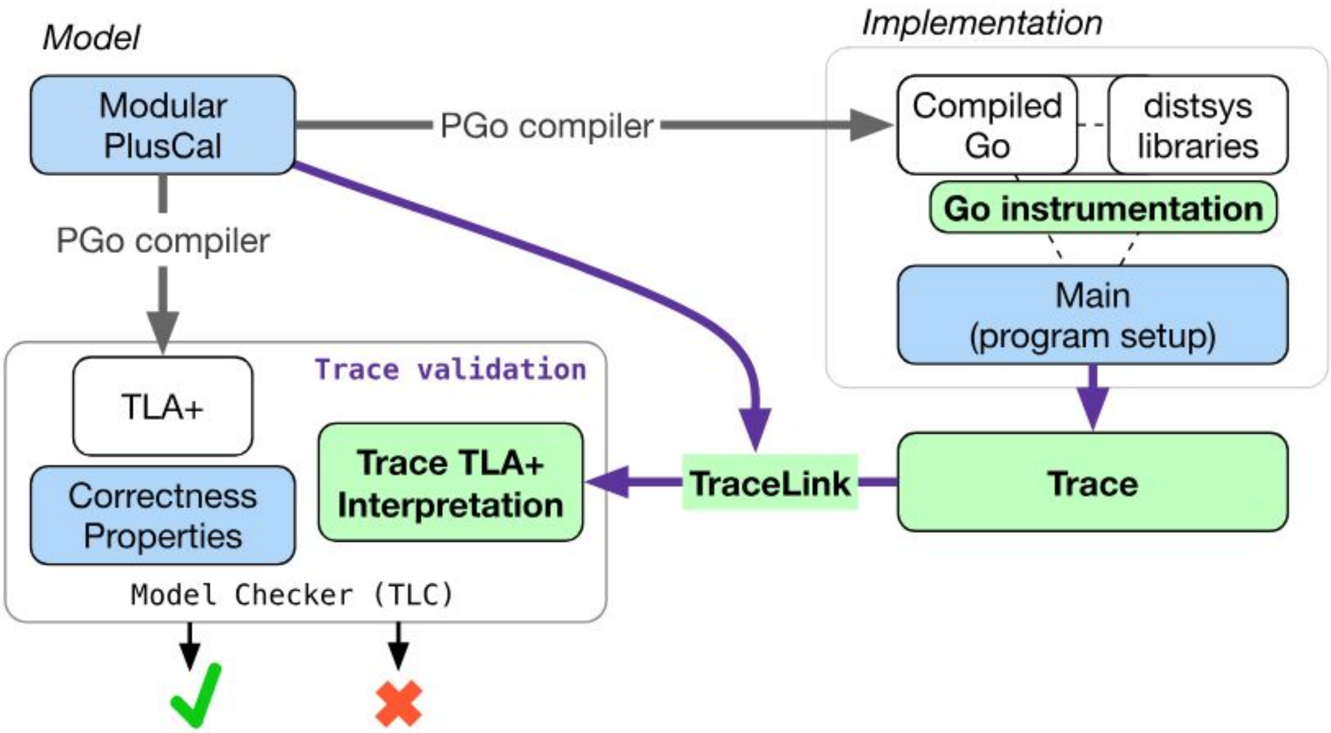


https://github.com/distCompiler/pgo

Compiler from Modular PlusCal (MPCal) to TLA+ and Go.

- 🎉 Full introspection of source model
- 🎉 Customizable runtime library for generated implementations

TraceLink: Push-button Validation of PGo Systems

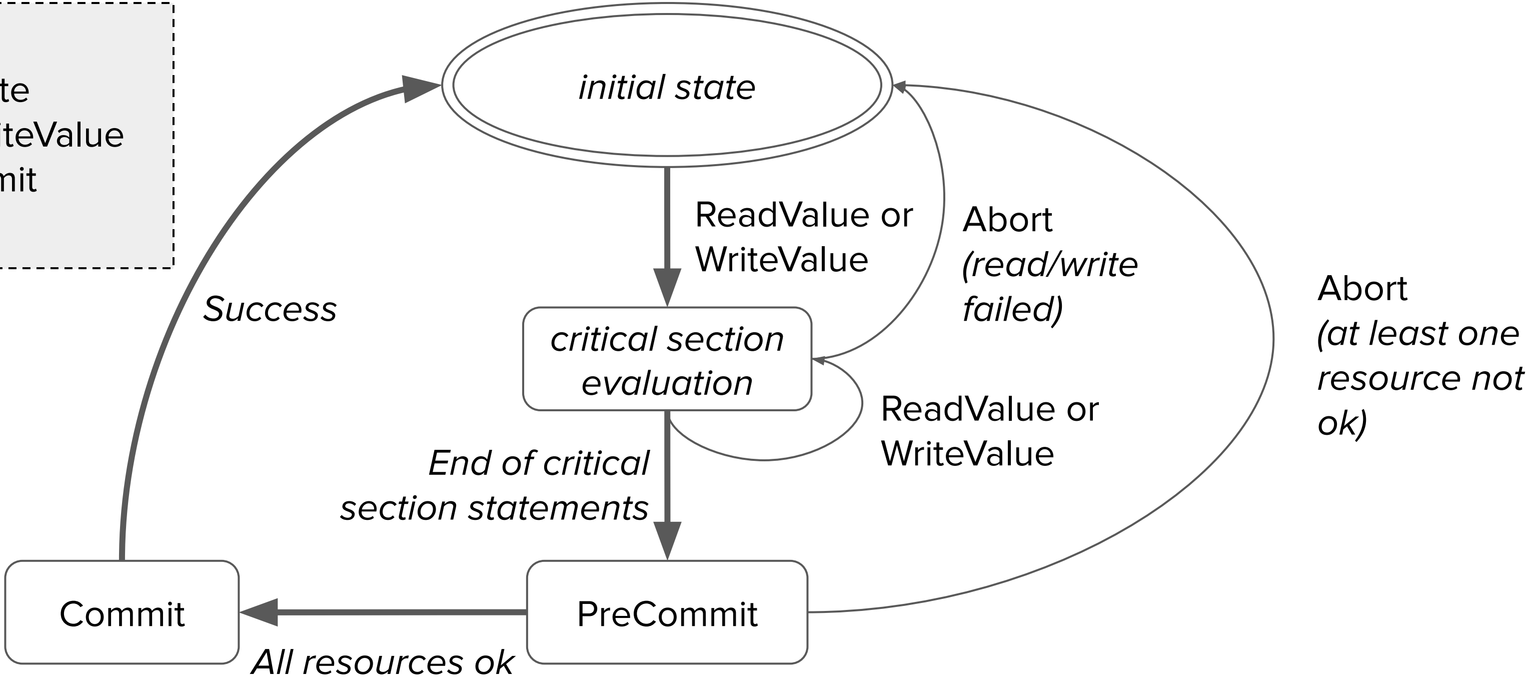


Demo Time

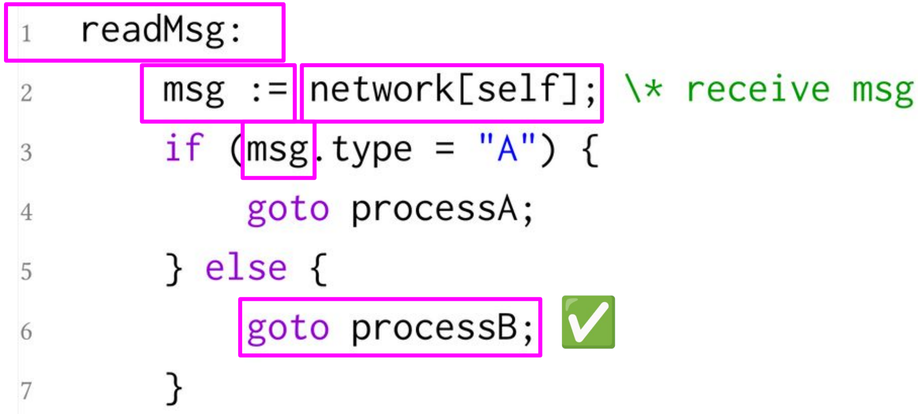
PGo Implementation Control Flow Primer

Happy path

1. Initial state
2. Read/WriteValue
3. PreCommit
4. Commit

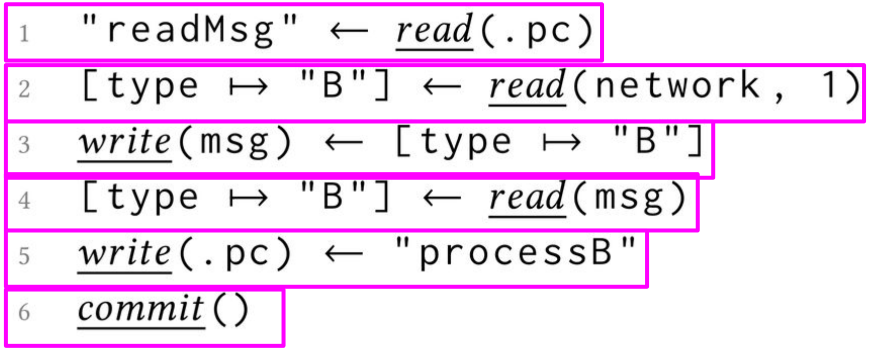

initial state
ReadValue or WriteValue
Abort (read/write failed)
Abort (at least one resource not ok)
Success
critical section evaluation
ReadValue or WriteValue
End of critical section statements
Commit
PreCommit
All resources ok

PGo Control Flow, Logged



```
readMsg:
    msg := network[self]; \* receive msg
    if (msg.type = "A") {
        goto processA;
    } else {
        goto processB; ✅
    }
```

Left: Modular PlusCal example, 1 critical section

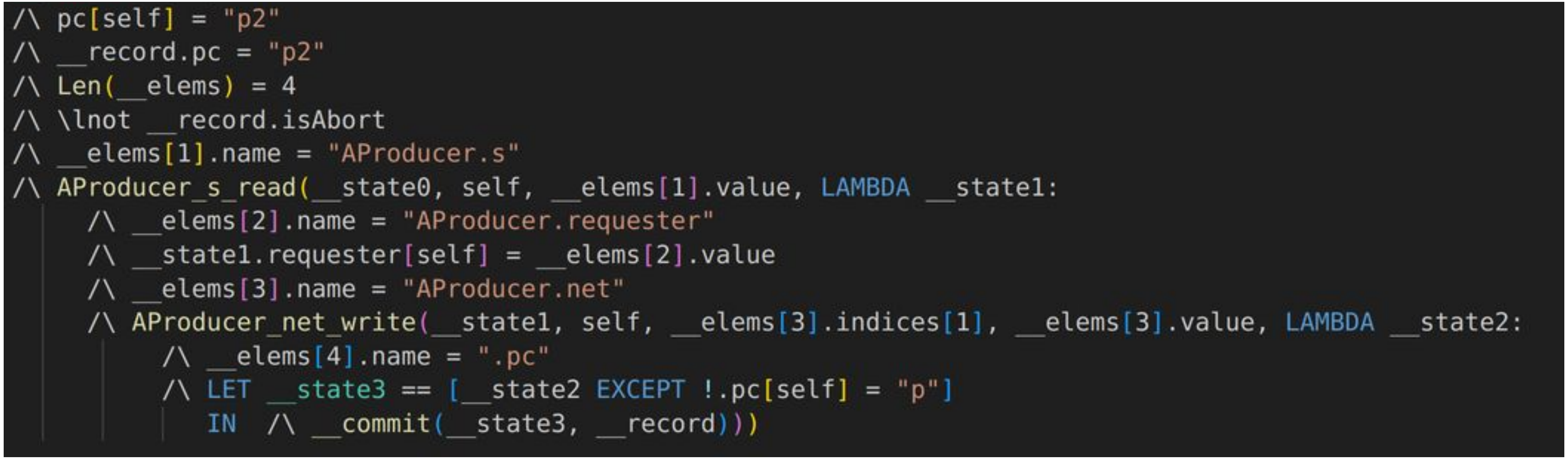


```
"readMsg" ← read(.pc)
[type ↦ "B"] ← read(network, 1)
write(msg) ← [type ↦ "B"]
[type ↦ "B"] ← read(msg)
write(.pc) ← "processB"
commit()
```

Right: Possible TraceLink implementation log

- Semantics are in terms of environment read/write
- Environment includes local vars, globals, network
- Entry ends in commit / abort: log first, then decide if it happened
- Aborted entries: check reads, ignore writes

A Brief Look at the Generated TLA+



```
/\ pc[self] = "p2"
/\ __record.pc = "p2"
/\ Len(__elems) = 4
/\ \lnot __record.isAbort
/\ __elems[1].name = "AProducer.s"
/\ AProducer_s_read(__state0, self, __elems[1].value, LAMBDA __state1:
     /\ __elems[2].name = "AProducer.requester"
     /\ __state1.requester[self] = __elems[2].value
     /\ __elems[3].name = "AProducer.net"
     /\ AProducer_net_write(__state1, self, __elems[3].indices[1], __elems[3].value, LAMBDA __state2:
          /\ __elems[4].name = ".pc"
          /\ LET __state3 == [__state2 EXCEPT !.pc[self] = "p"]
             IN  /\ __commit(__state3, __record)))
```

Three Steps Toward Practicality

1. **The log is going to be huge**

   Naive approach could generate >500,000 lines of TLA+, intractable.

   👉 How to generate compact TLA+?

2. **A distributed system has no total order on events**

   👉 TLA+ does, need to reconcile

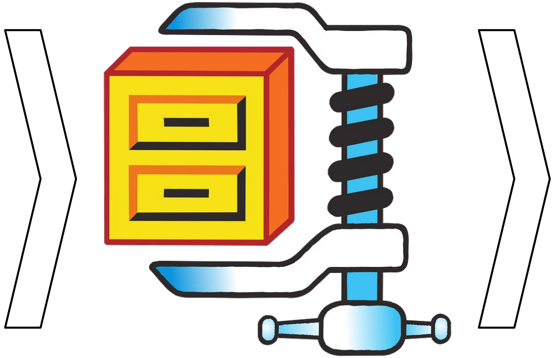
3. **Can’t validate what you can’t see**

   👉 Need to capture interesting traces

1. Why the Generated TLA+ is not >500,000 Lines Long

500,000 lines

```
log1 ==
  /\ x = 1
  /\ x’ = 2

log2 ==
  /\ x = 2
  /\ x’ = 3
\* ...
```

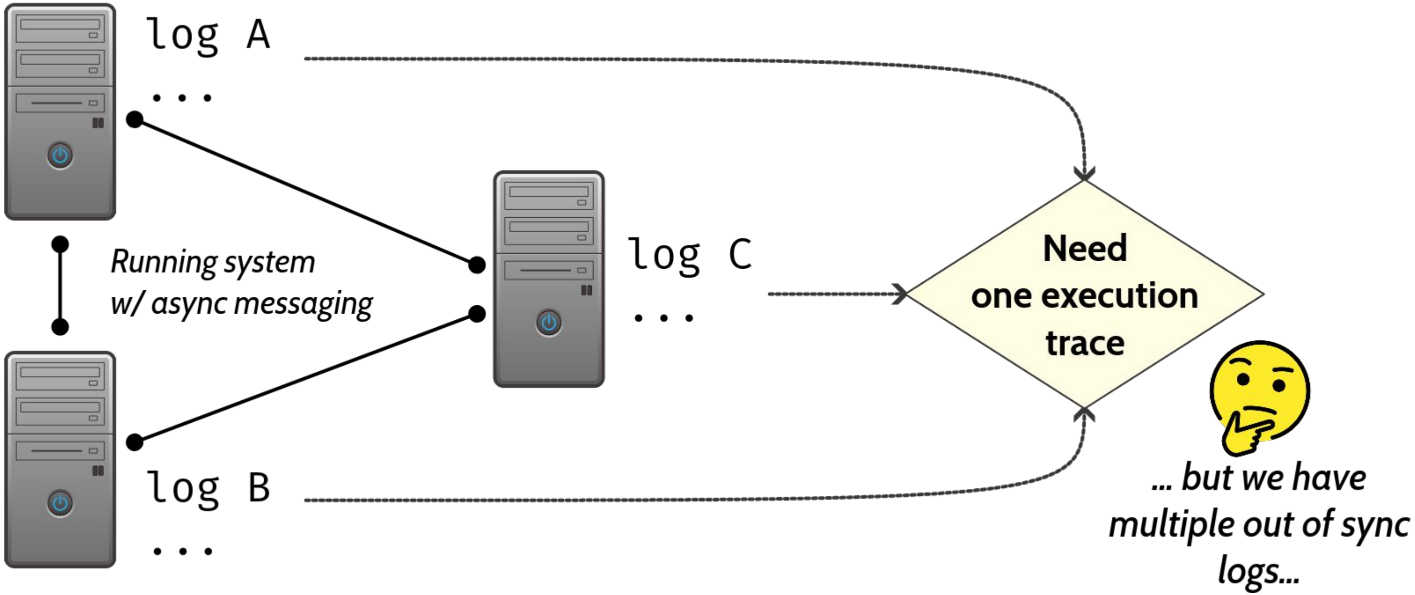


<1,000 lines

```
log(i) ==
  LET __elems ==
        __data[i]
  IN  /\ x = __elems[1]
      /\ x’ = __elems[2]
```

Key insight: same structure, different concrete values.
Put values in .bin file, keep TLA+ tractable.

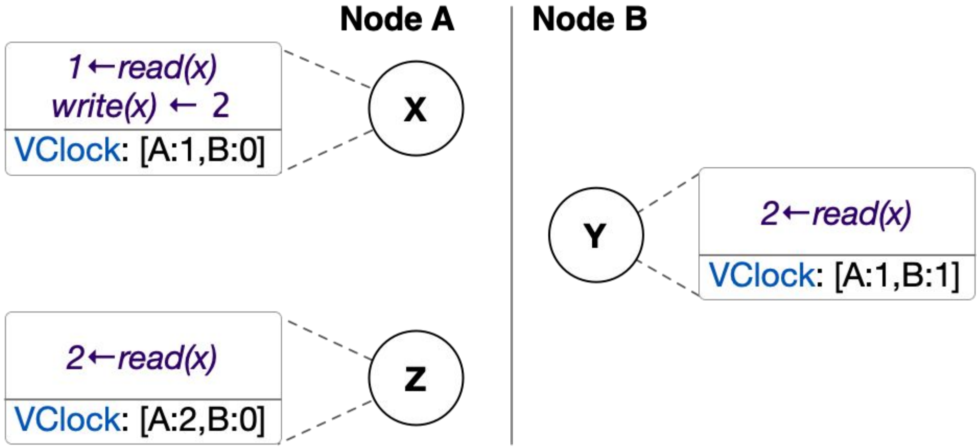
2. Asynchronous Logging vs TLA+ Total Order



- 👉 Track causality with vector clocks, get partial order
- 💡 Could look at timestamps (see future work)

2. Multi Critical Section Example

Init: `x = 1`



2 total orderings

X □ Y □ Z

X □ Z □ Y

Vector clocks map process id to logical clock (int), increase locally and merge during communication.

2. Strategies for Validating Possible Orderings

🤔 Pick one order (TLC depth-first mode, possible w/ extra flag)

🤔 Pick all orders (TLC breadth-first mode, default)

Both work, but significant tradeoff between performance and coverage.

New 🎉 helpful medium

Pick one order but check that every diverging order could work.

Uses depth-first mode with special generated action property.

3. Diverse Trace Generation

Trace validation can only see what the implementation did. Make sure the implementation does different things.

Theory: many classes of concurrency bug require a small number of changes to a concurrency schedule [ASPLOS ‘10]

Our practice: exponentially distributed sleeps between every MPCal operation.

Other options: Antithesis, Trace Aware Random Testing [OOPSLA’19],
Systematic Schedule Exploration [OSDI’14],
Systematic Testing of Multithreaded Programs [PLDI’07]

Selected Issues we Found

Systems we Tested

All test systems compiled with PGo (current limitation)

- **dqueue:** basic producer-consumer model. Good smoke test.
- **locksvc:** distributed lock service. Has concurrency + invariants.
- **raftkvs:** full-scale Raft-based key-value store, PGo’s main evaluation target.

Most bugs found at scale in raftkvs.

Log sizes up to **100k events**, across up to **26 processes**.

Some **counter-examples >10ks states deep**, needed special debug tech.

List of Bugs

- 🐛 2x network assumption 👉
- 🐛 1x PGo miscompilation 👉
- 🐛 2x instrumentation error 👉
- 🐛 2x timeout model
- 🐛 1x failure detector model
- 🐛 1x model abstraction

Modular PlusCal Environment Assumptions

TCP send-receive order *between different connections*

- Send 2 messages to same recipient over different connections
- We assume receive order ⇔ send order, which is incorrect

- True for same connection, accidentally assumed it for all messages to same recipient
- Subtle modeling error, can affect correctness

Credit to Horatiu Cirstea for initially showing this possibility.

PGo Miscompilation

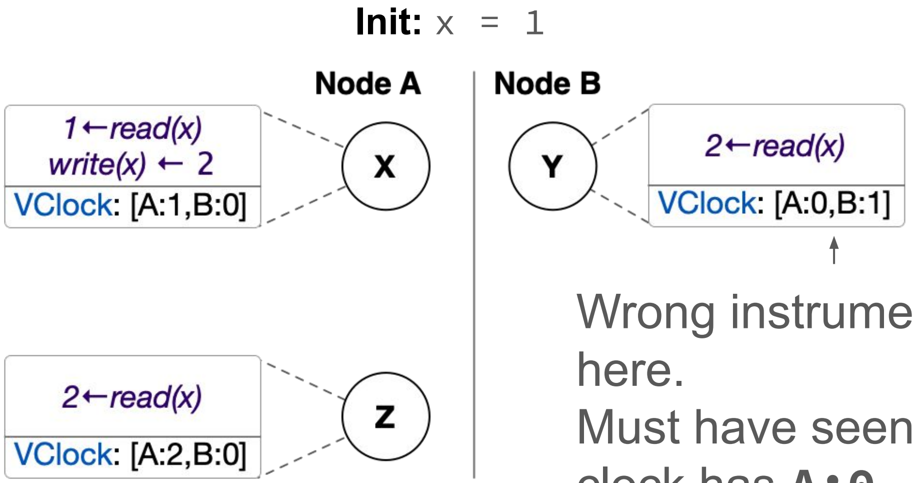
`[a |-> 1] @@ [a |-> 2] = ???`

- `@@` allows combination of functions / records with different domains, TLC-specific.
- PGo compiled `??? = [a |-> 2]` (keep right)
- TLC evaluated `??? = [a |-> 1]` (keep left, correct per manual)

Accidentally never cross-checked in properties.

Wrong spec + PGo miscompilation → correct implementation 🤯

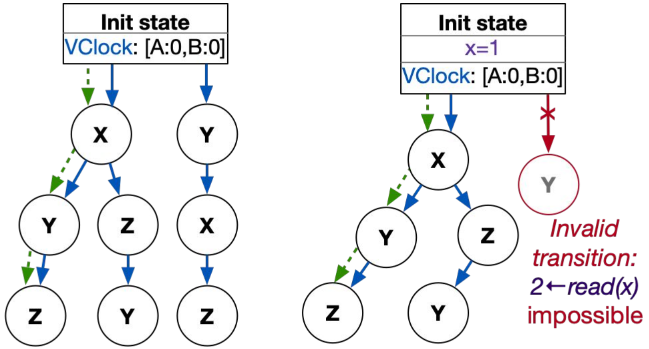
TraceLink Instrumentation Bugs (2 instances found)



Wrong instrumentation here.
Must have seen X, but clock has **`A:0`**
It should be **A:1**

No need to trust TraceLink instrumentation.

Wrong path identified



Going Forward

Considering Plain TLA+ Models

Can we port TraceLink to non-PGo systems and have it be useful?

TraceLink relies on:

- MPCal concepts like mapping macros
- Specific implementation log structure

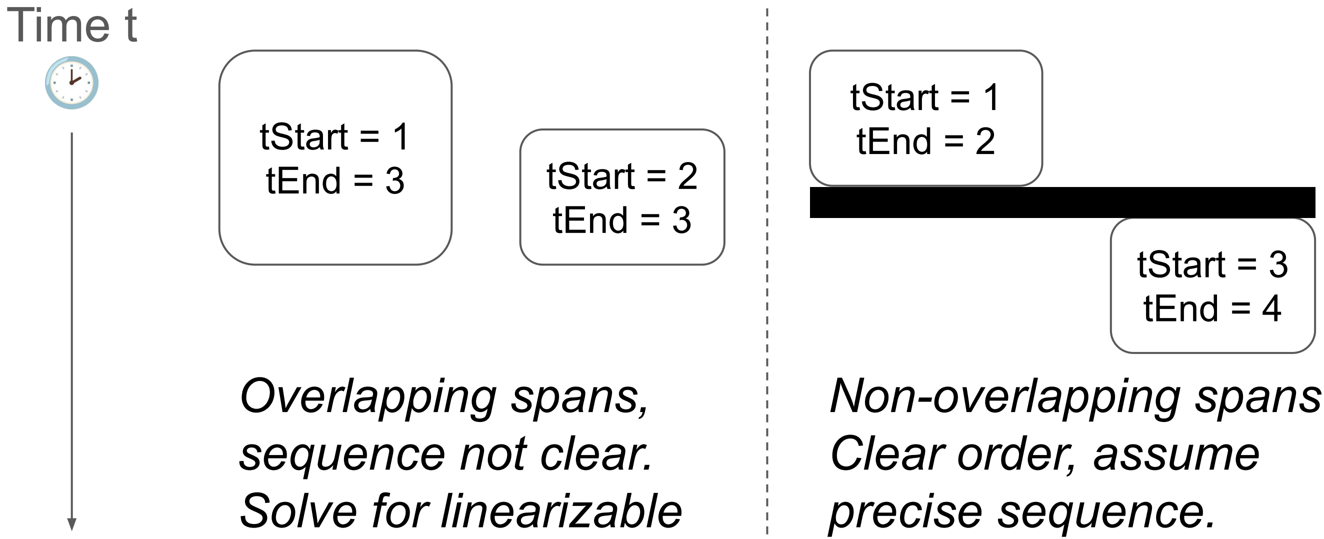
🤔 Imitate TraceLink’s log structure in hand-written implementation

🤔 Extend to industry logging, like spans?

🤔 Hand-adapting TLA+ to MPCal may be viable, or could be automated?

Causality and Real Time

When recording critical section start + end, we could recover partial order.



Overlapping spans, sequence not clear. Solve for linearizable order.

Non-overlapping spans. Clear order, assume precise sequence.

Note: can account for clock drift by adding error factor to time span.

🎉 *only overlapping spans need solving, order is otherwise clear.*

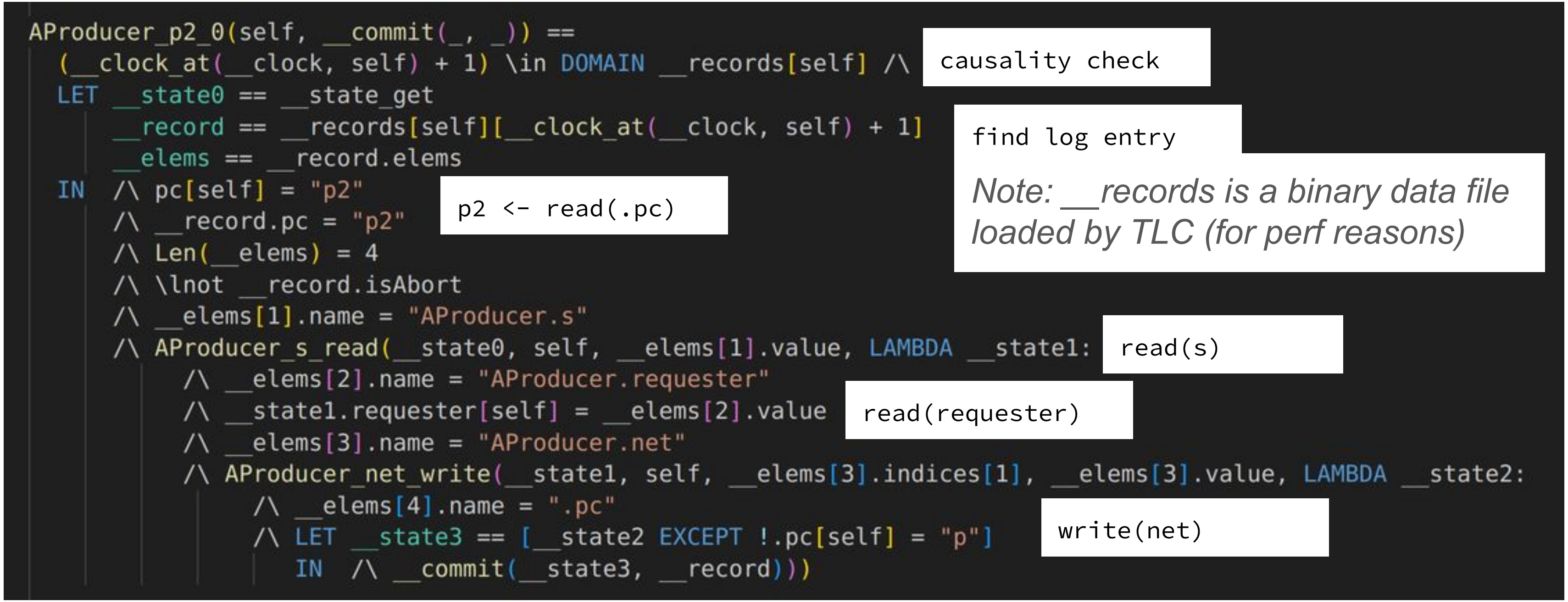
Contributions

Goal: make trace validation easier to apply.

- Implemented push-button validation for PGo systems
  - Automatically instrument PGo-generated systems with vector clocks
  - TraceLink uses MPCal specs and trace data to generate trace validation setup
- Found interesting bugs in PGo context, ideas to extend beyond PGo
- Will use this summer @ MongoDB®
- Try it yourself!

github.com/distCompiler/pgo

A Brief Look at the Generated TLA+



```
AProducer_p2_0(self, __commit(_, _)) ==
  (__clock_at(__clock, self) + 1) \in DOMAIN __records[self] /\
  LET __state0 == __state_get
      __record == __records[self][__clock_at(__clock, self) + 1]
      __elems == __record.elems
  IN  /\ pc[self] = "p2"
      /\ __record.pc = "p2"
      /\ Len(__elems) = 4
      /\ \lnot __record.isAbort
      /\ __elems[1].name = "AProducer.s"
      /\ AProducer_s_read(__state0, self, __elems[1].value, LAMBDA __state1:
           /\ __elems[2].name = "AProducer.requester"
           /\ __state1.requester[self] = __elems[2].value
           /\ __elems[3].name = "AProducer.net"
           /\ AProducer_net_write(__state1, self, __elems[3].indices[1], __elems[3].value, LAMBDA __state2:
                /\ __elems[4].name = ".pc"
                /\ LET __state3 == [__state2 EXCEPT !.pc[self] = "p"]
                   IN  /\ __commit(__state3, __record)))
```

causality check

find log entry

Note: __records is a binary data file loaded by TLC (for perf reasons)

p2 <- read(.pc)

read(s)

read(requester)

write(net)