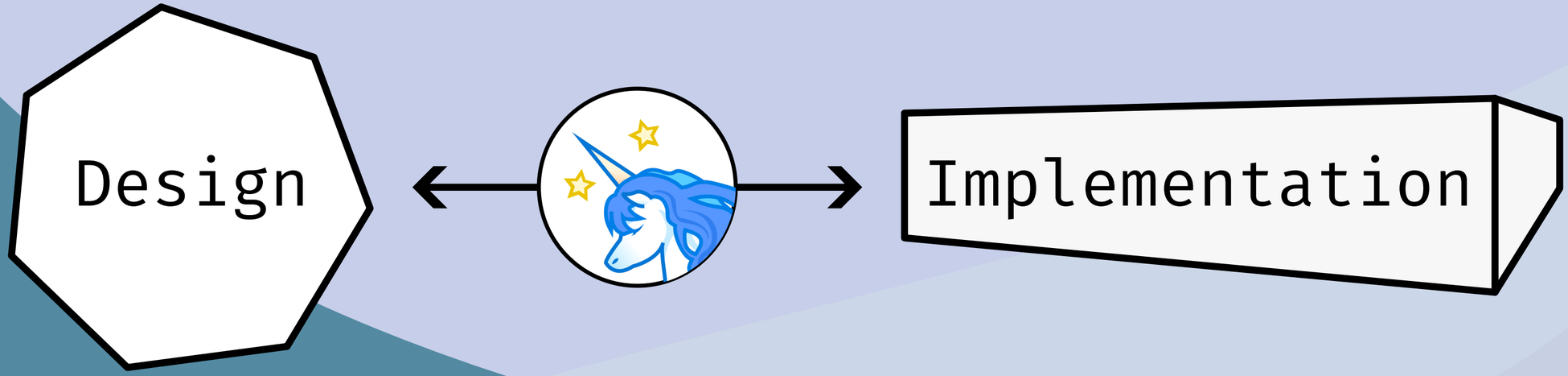
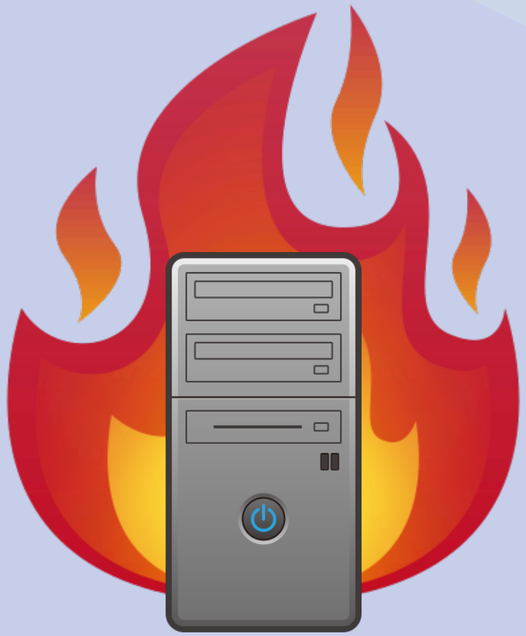
# Promises and Challenges in Bridging TLA+ Designs with Implementations

*A. Finn Hackett*



# Building and Running Distributed Systems is Notoriously Error-prone



# Building and Running Distributed Systems is Notoriously Error-prone

# Building and Running Distributed Systems is Notoriously Error-prone

# Building and Running Distributed Systems is Notoriously Error-prone

🚀 Analyzable designs

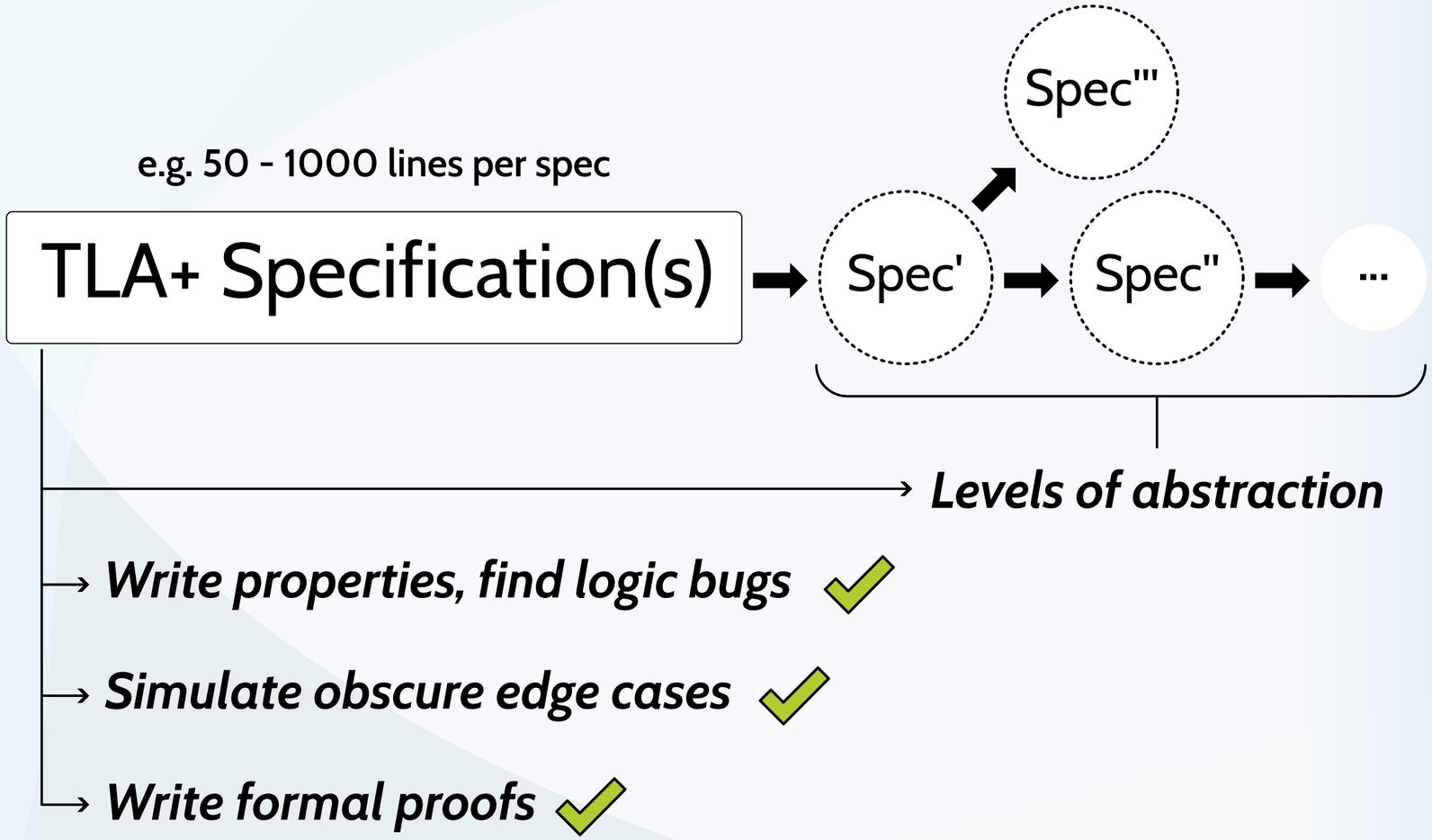

(or Alloy, P, Promela, Dafny, Verus, F*, Coq)

# Formal Methods

🚀 Find edge cases

🚀 Proofs

# Usage of TLA+


e.g. 50 - 1000 lines per spec
TLA+ Specification(s)
Spec'''
Spec'
Spec''
...
Levels of abstraction
Write properties, find logic bugs
Simulate obscure edge cases
Write formal proofs

# Usage of TLA+

e.g. 50 - 1000 lines per spec

TLA+ Specification(s) → Spec' → Spec" → ... → Spec'" → Implementation(s)

*Levels of abstraction*

- *Write properties, find logic bugs* ✓
- *Simulate obscure edge cases* ✓
- *Write formal proofs* ✓

**Recurring question:**
How can we be (more) sure impl and spec match?

# If We Were Sure Our Models and Implementations Matched...

# If We Were Sure Our Models and Implementations Matched…

Only bug possible is wrong correctness properties

# If We Were Sure Our Models and Implementations Matched...

Only bug possible is wrong correctness properties

Unreasonably precise monitoring for free using verification tools

# If We Were Sure Our Models and Implementations Matched...

Only bug possible is wrong correctness properties

Unreasonably precise monitoring for free using verification tools

If we're really really sure, do we even need different spec + impl code?

# How Have We Attempted Implementation Linking?

[@Confidential Consortium Framework @etcd]

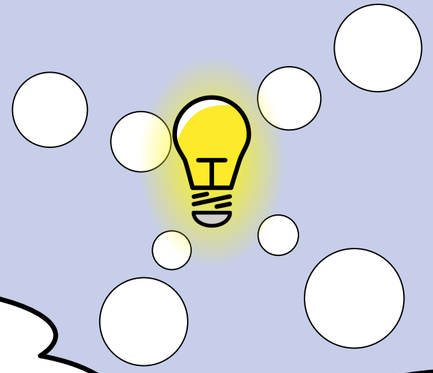
## Trace Validation

*e.g. collect structured logs + compare with TLA+*

## Compile the TLA+

*e.g. the PGo project, PlusPy, Elixir*

[Related to:
- Verdi
- IronFleet
- F*]



## Test Case Generation

*e.g. use execution traces as test scenarios*

## Runtime Monitoring

*e.g. put/compile the TLA+ assertions in your code*

[See: Choreographic PlusCal, Elixir ver.]

# Tradeoffs in Trace Validation

- ☑ Directly observes the implementation, could catch wide range of errors

  e.g. misconfiguration, wrong assumption in TLA+

- 🚀 Strictly beyond spec verification

# Tradeoffs in Trace Validation

☑ Directly observes the implementation, could catch wide range of errors

e.g. misconfiguration, wrong assumption in TLA+

🚀 Strictly beyond spec verification

🤔 Manual effort needed to instrument + handle logs

... how much effort can we automate?

# Tradeoffs in Trace Validation

☑ Directly observes the implementation, could catch wide range of errors

e.g. misconfiguration, wrong assumption in TLA+

🚀 Strictly beyond spec verification

🤔 Manual effort needed to instrument + handle logs

... how much effort can we automate?

❌ Incomplete: if you don't see the implementation do it, you don't check it

🤷 Better than nothing to use it in your integration tests

# Generating Test Cases

🤔 **Trace Validation**

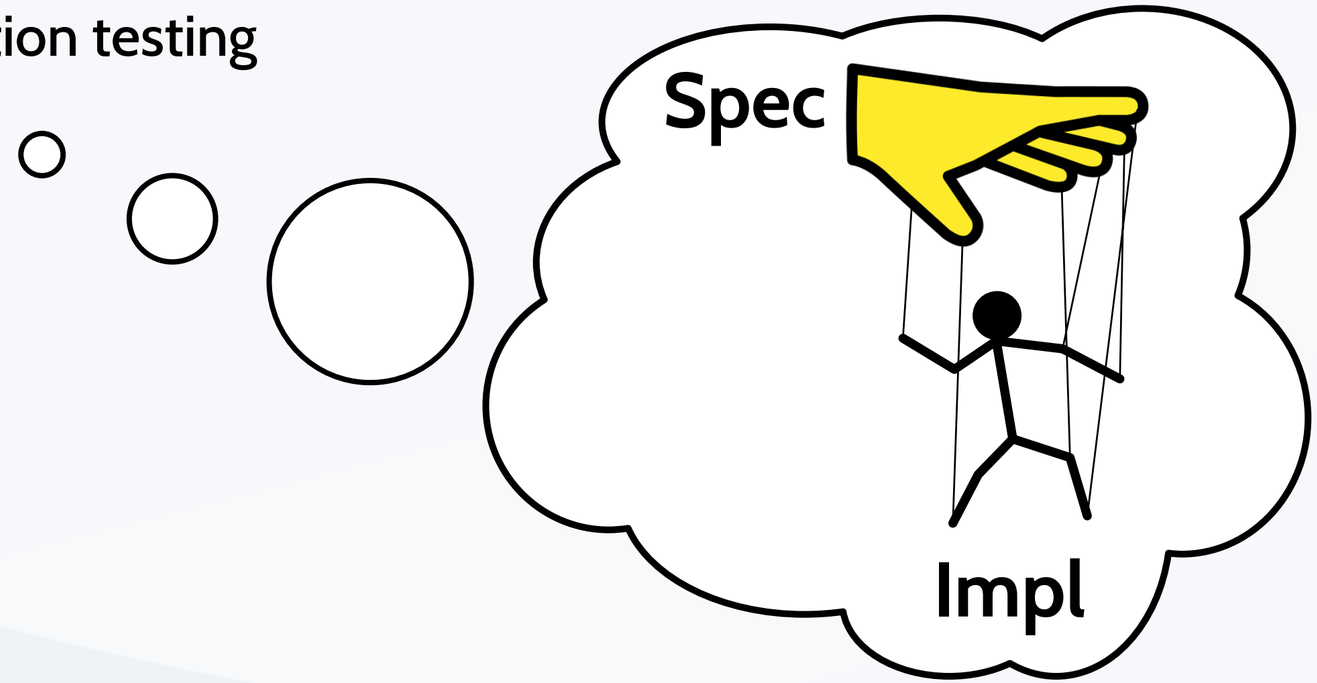
❌ Incomplete: if you don't see the implementation do it, you don't check it

# Generating Test Cases

🤔 **Trace Validation**

❌ Incomplete: if you don't see the implementation do it, you don't check it

💡 Let the spec drive implementation testing



# Tradeoffs in Test Case Generation

- ☑ Ensures implementation state space is actually explored
  -  Different from implementation model checking, but similar effect

# Tradeoffs in Test Case Generation

- ☑ Ensures implementation state space is actually explored
  - 💡 Different from implementation model checking, but similar effect
- 🤔 Extracting implementation behavior and state is still non-trivial
  - 🚀 … can be partly automated, but fundamental refinement job remains

# Tradeoffs in Test Case Generation

☑ Ensures implementation state space is actually explored

- 💡 Different from implementation model checking, but similar effect

🤔 Extracting implementation behavior and state is still non-trivial

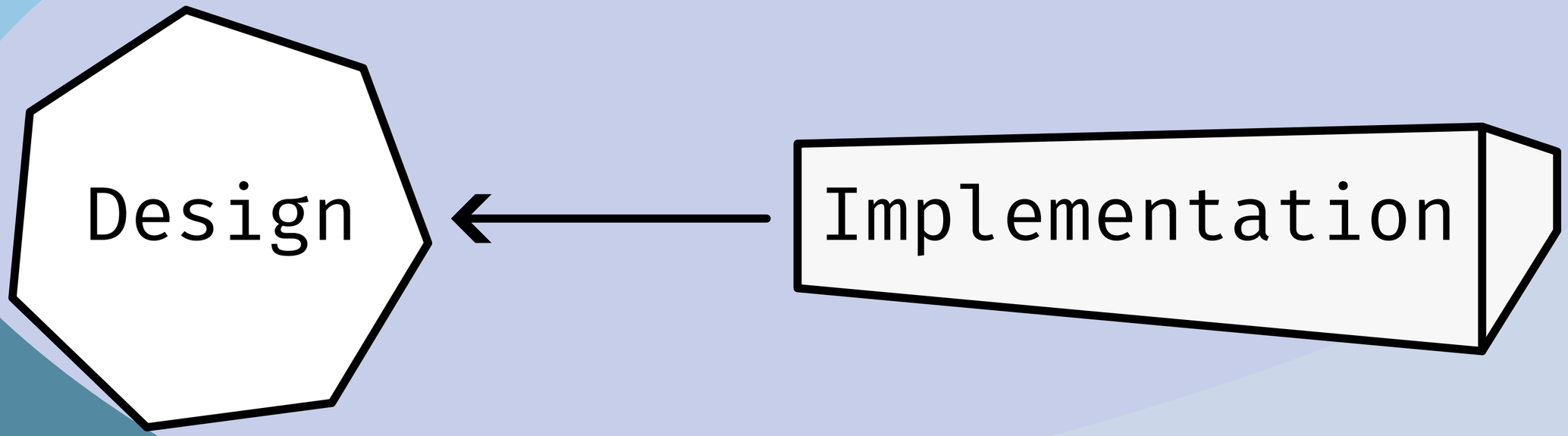
- 🚀 … can be partly automated, but fundamental refinement job remains

🤔 For existing implementation, need to retrofit deterministic exploration

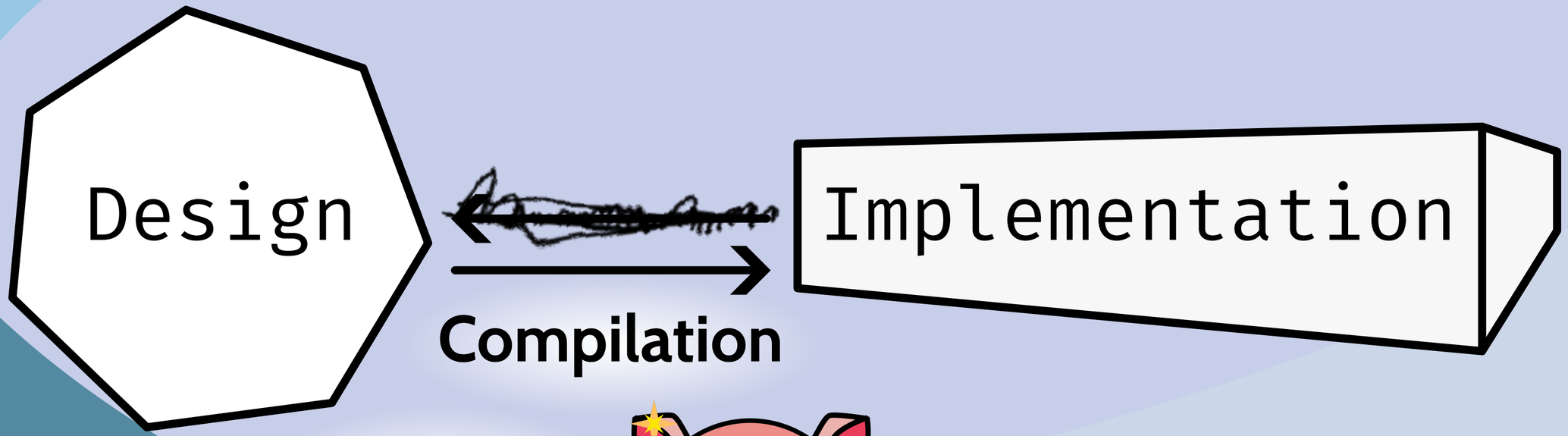
e.g. get a custom scheduler, or otherwise control all system actions

[See: Kani, Coyote, Chaos Engineering, Jepsen]

# Other Direction: Compile the Design



# Other Direction: Compile the Design




e.g. PGo
[ASPLOS'23]

# Tradeoffs in Specification Compilation

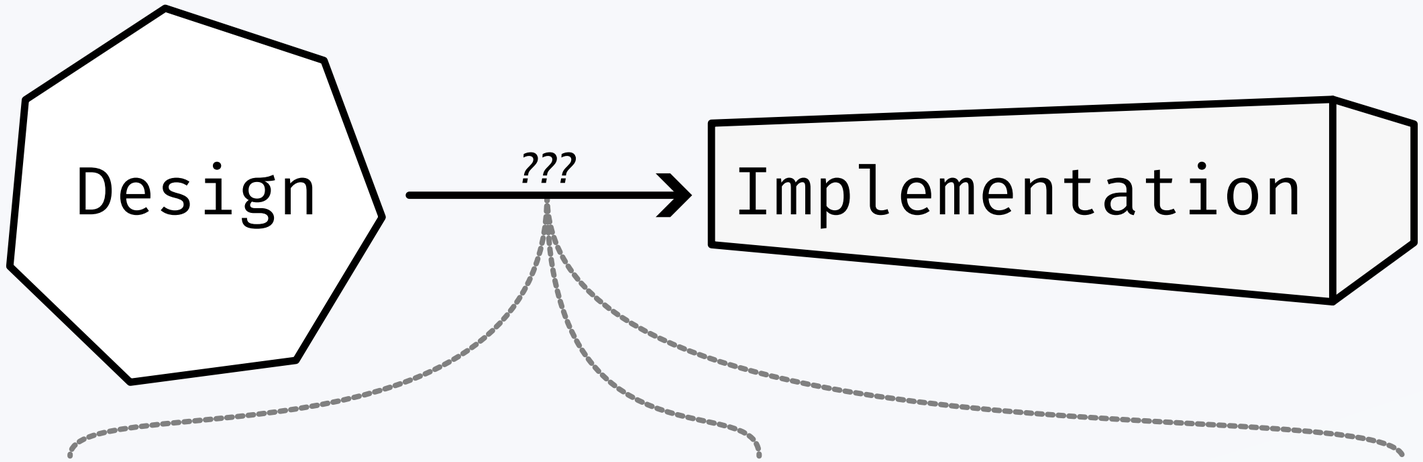
- ☑ Directly generates link between spec and implementation

... so that's it, problem solved right?

# Tradeoffs in Specification Compilation

☑ Directly generates link between spec and implementation

... so that's it, problem solved right?




Translating data structures right

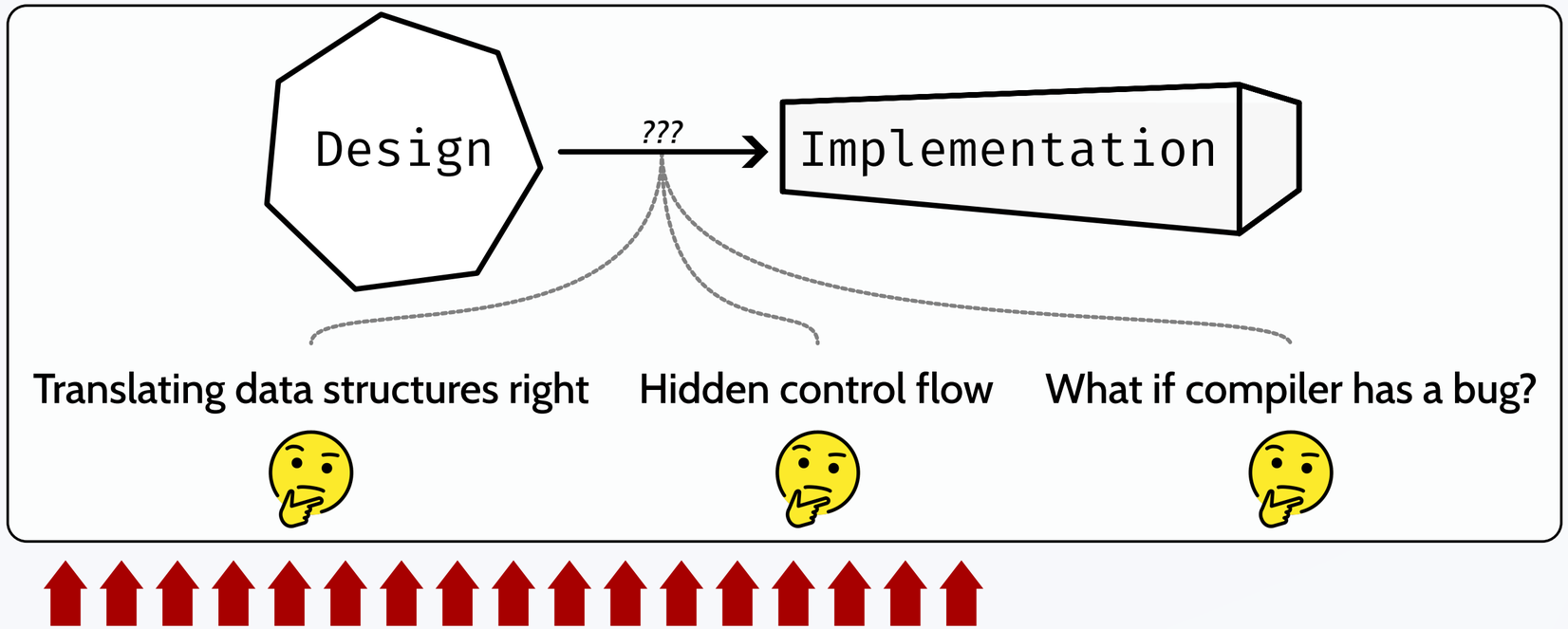
Hidden control flow

What if compiler has a bug?

# Ongoing Work...

# Ongoing Work: DCal, a More Customizable PGo

Move impl-oriented changes away from spec.



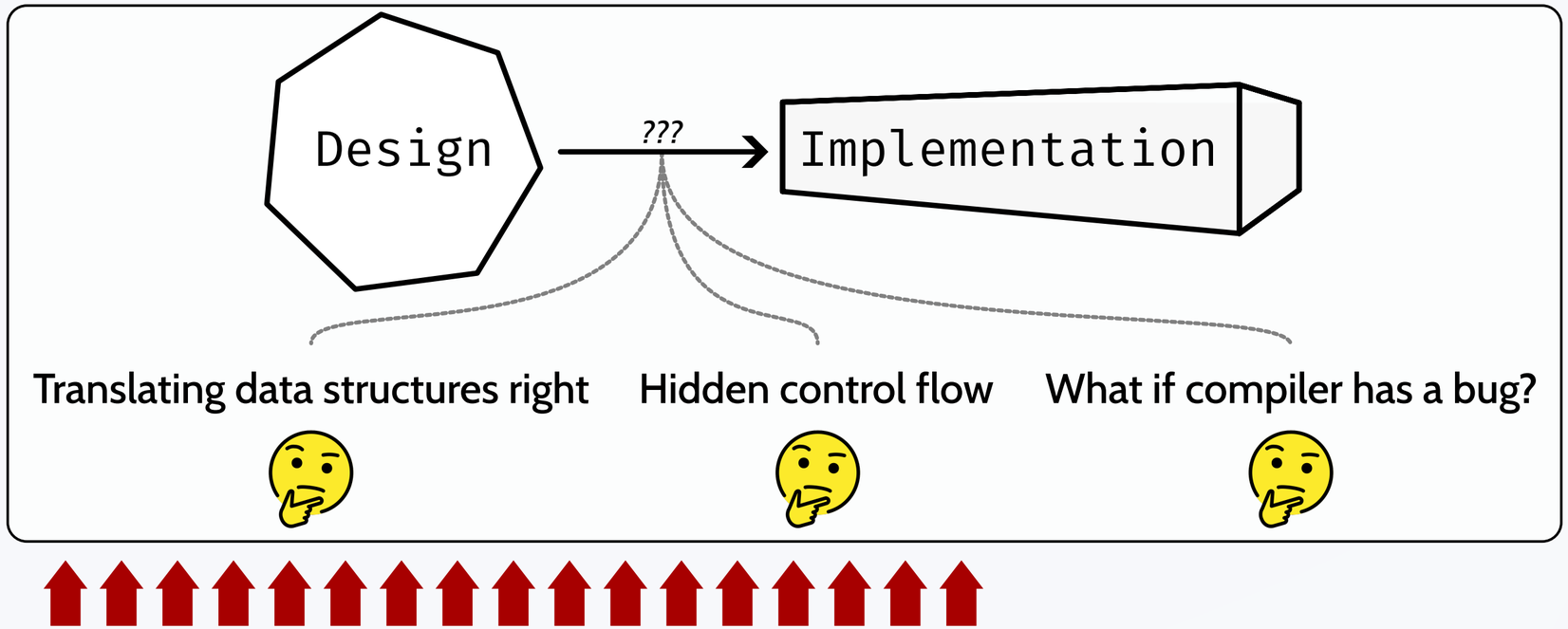
# Ongoing Work: DCal, a More Customizable PGo

🌈 Move impl-oriented changes away from spec.

👉 **PGo uses fixed data structures.**
General-purpose, but can be inappropriate.

*e.g. log structures: often specialized in practice, but PGo forces general purpose sequence type.*

💡 *Constraint system to specialize abstract TLA+ data specs.*



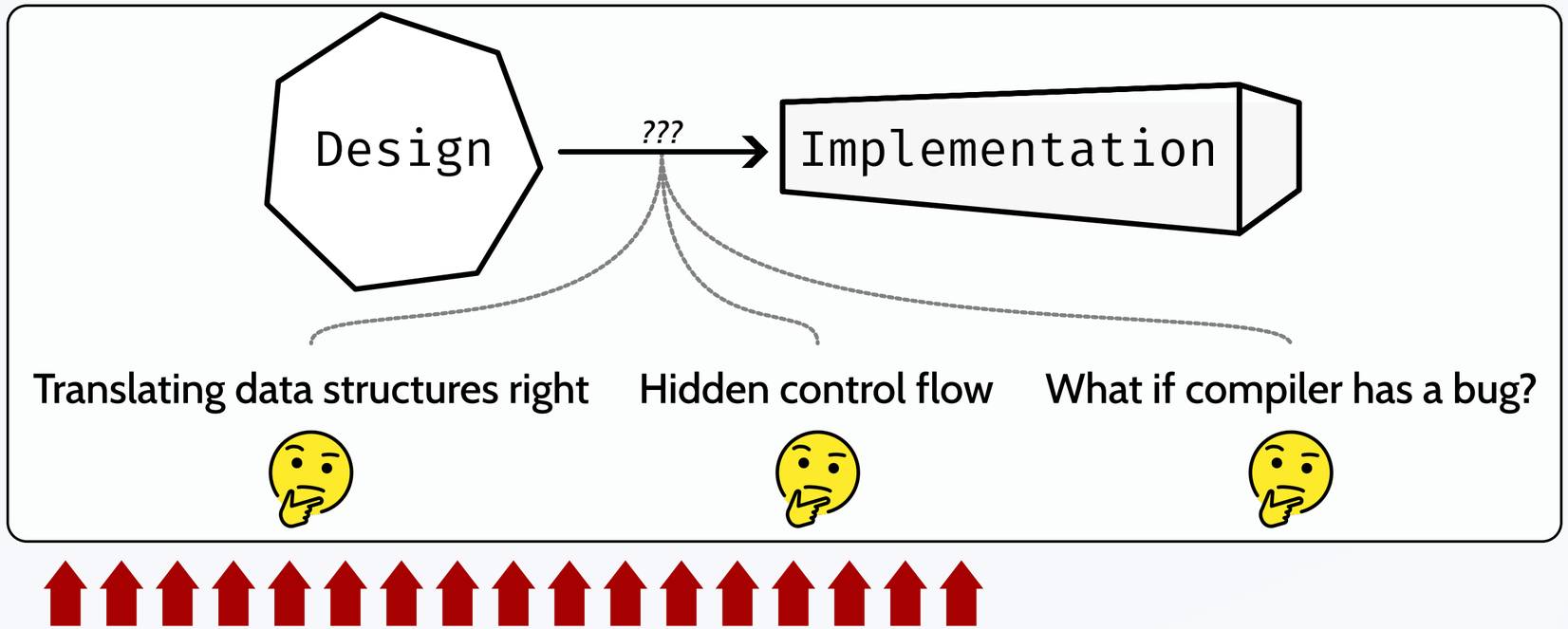
# Ongoing Work: DCal, a More Customizable PGo

🌈 **Move impl-oriented changes away from spec.**

👉 **PGo uses fixed data structures.**
General-purpose, but can be inappropriate.

*e.g. log structures: often specialized in practice, but PGo forces general purpose sequence type.*

💡 *Constraint system to specialize abstract TLA+ data specs.*




👉 **PGo's control flow impl is black-box and fixed.**
Difficult to specialize compiler's output.

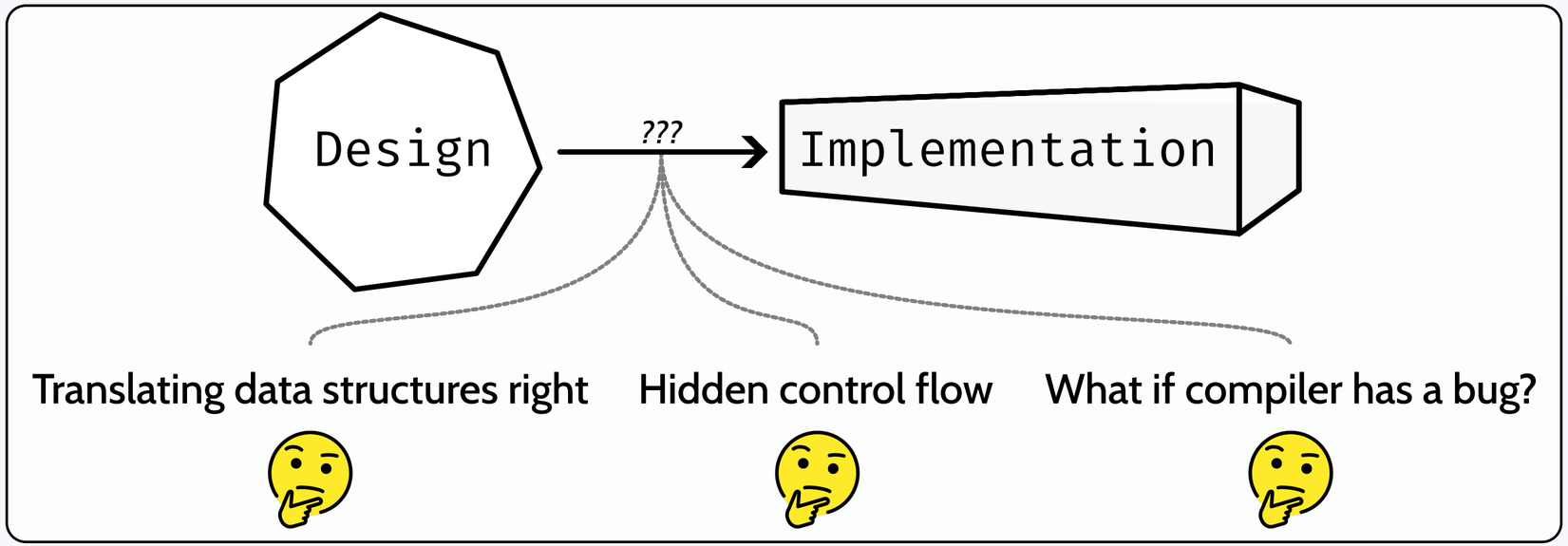
*e.g. can't compile disjunction to I/O select primitive.*

💡 Write specific strategies as meta-programs / compiler plugins.

# Ongoing Work: TraceLink, Compiler-assisted Trace Validation

🤔 **Trace Validation** → Manual effort needed to instrument + handle logs

… how much effort can we automate?



# Ongoing Work: TraceLink, Compiler-assisted Trace Validation

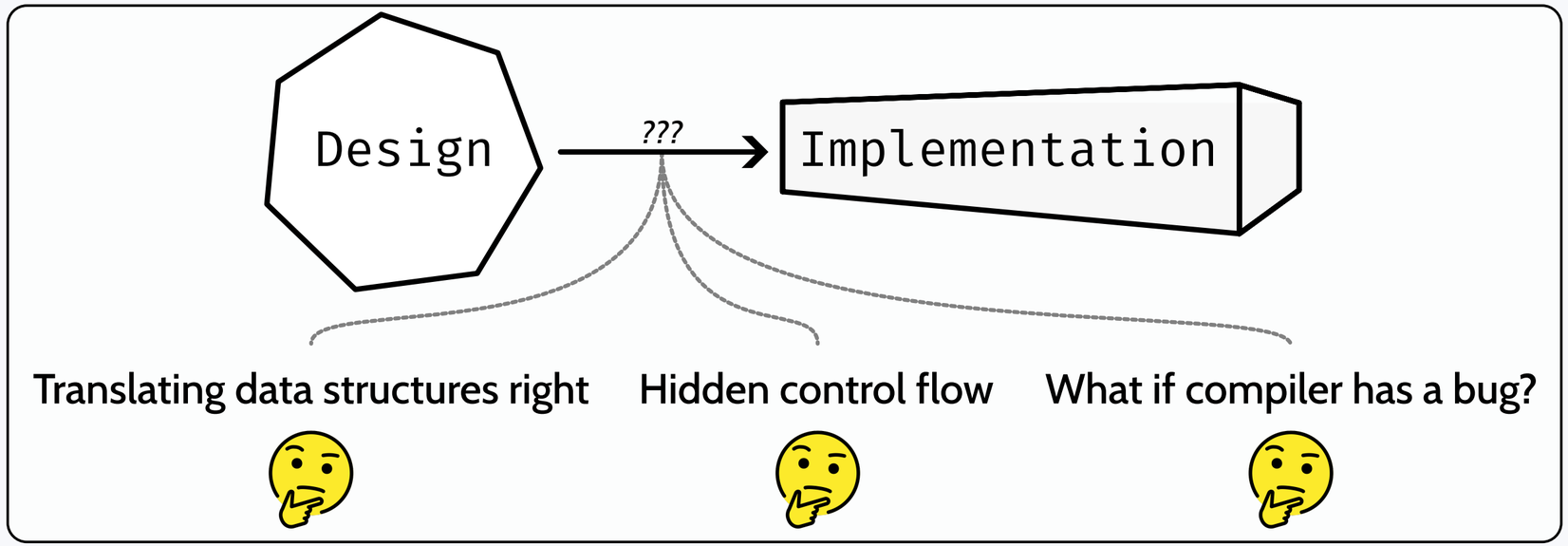
Trace Validation → Manual effort needed to instrument + handle logs

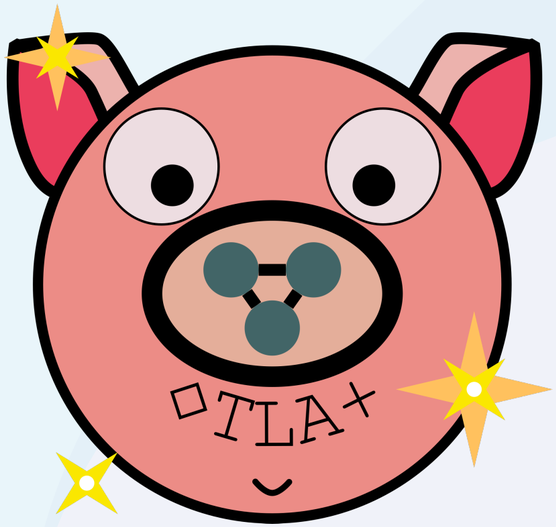
… how much effort can we automate?

**How to find problems in the compiled system?**

*Use the compiler to automate trace validation workflow.*

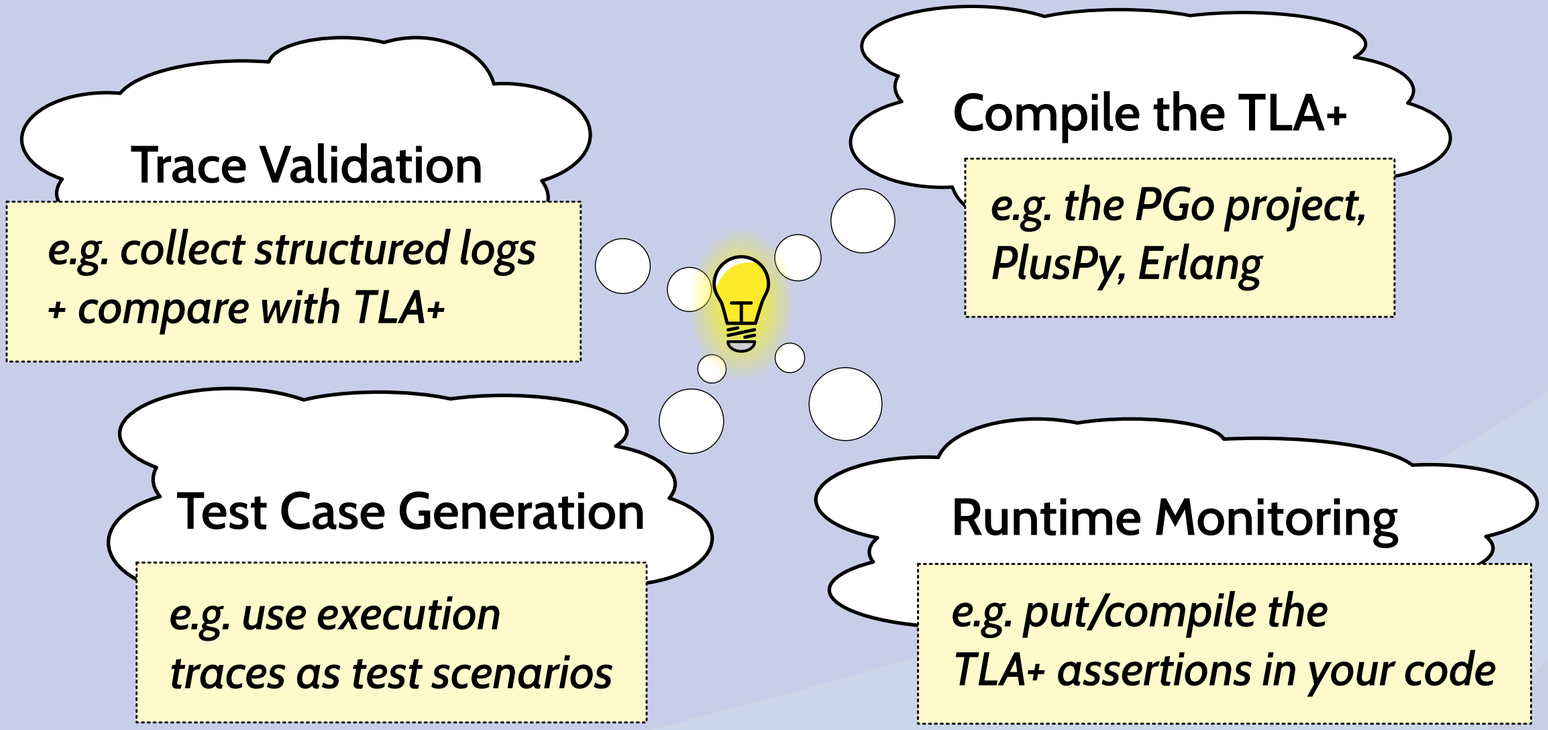
*Use model to analyze trace validation soundness.*






distcompiler.github.io

# Promises and Challenges in Bridging TLA+ Designs with Implementations




## Any Questions?

# Trace Validation: Refinement w/ Implementation Traces

Existing TLA+ Specification

Refinement

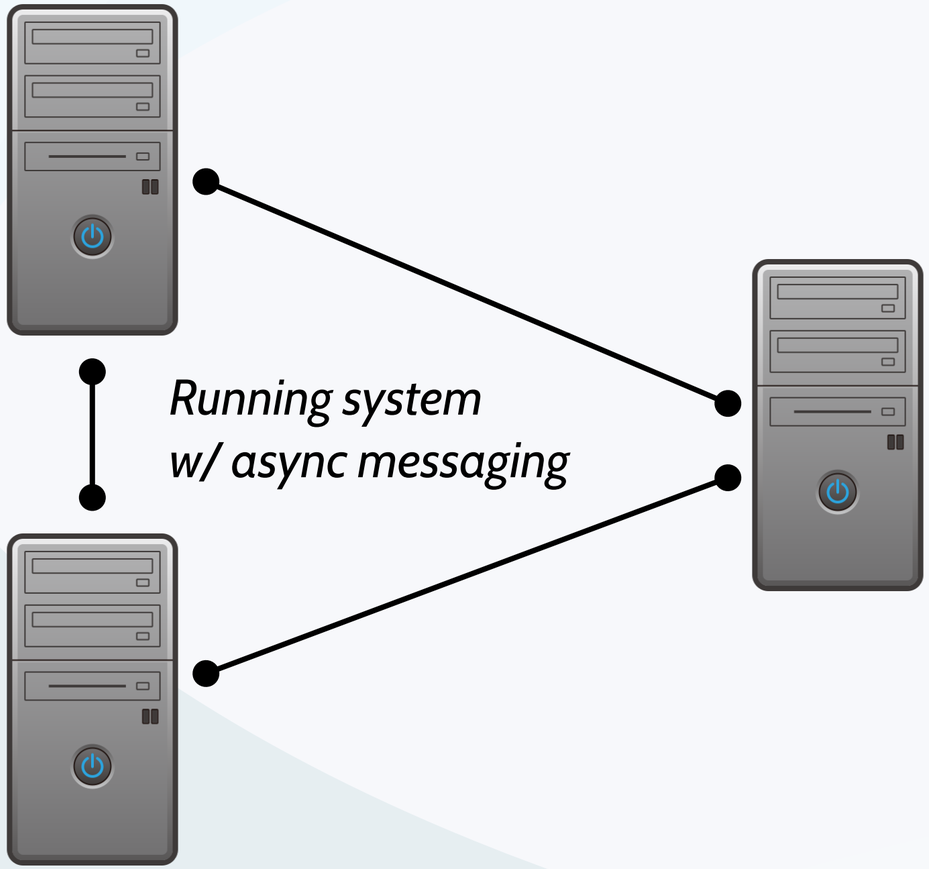
logging impl. behavior

...

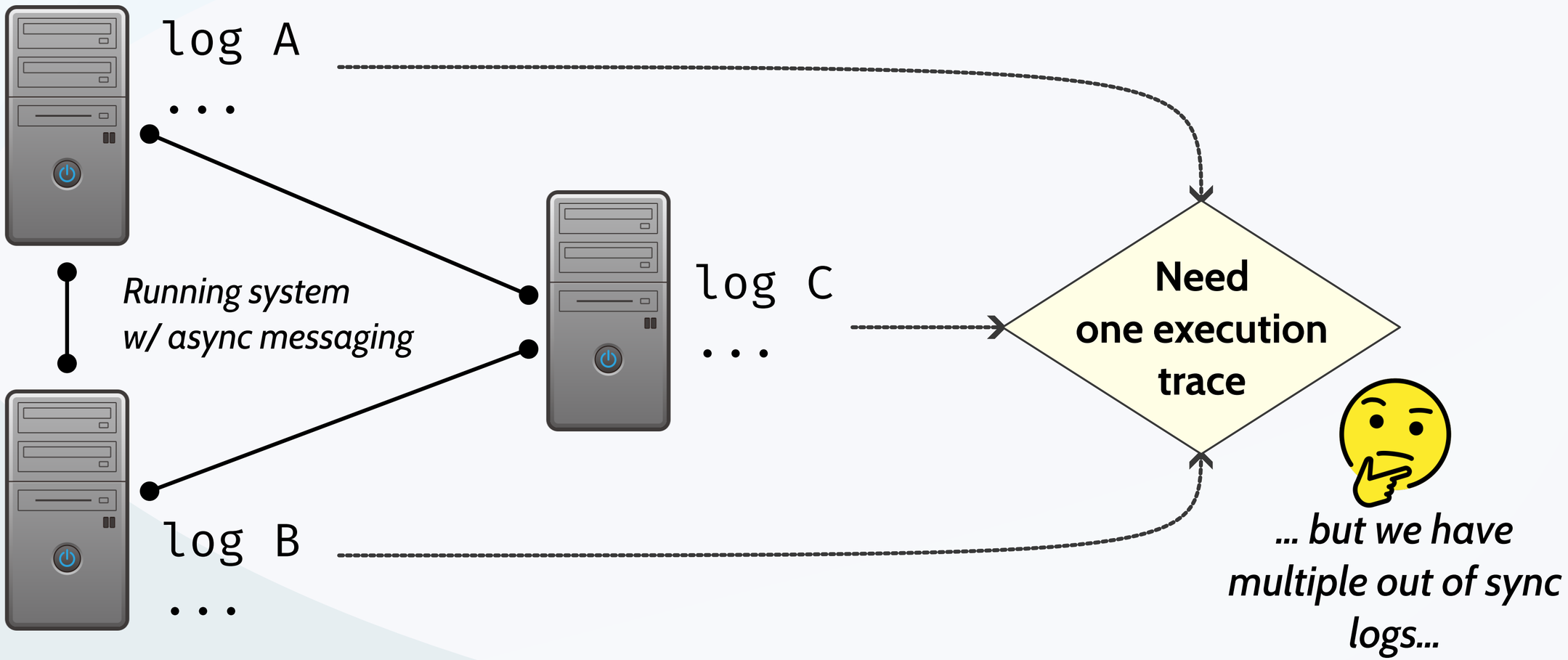
```
dict["x"] := "y";
```

```
> try put(key="x", value="y")
> tcp error
> retry
> timeout
> backoff
...
> retry
> ok
```

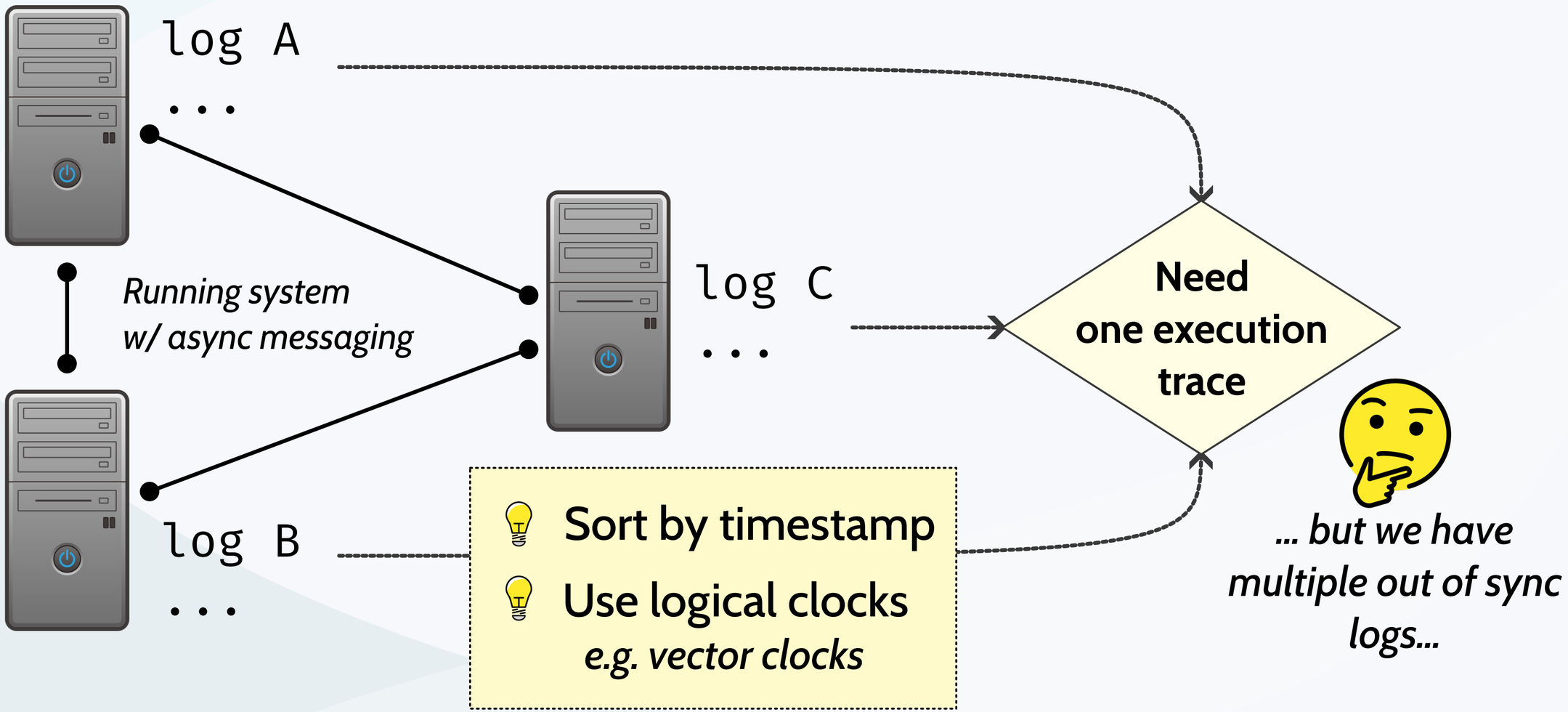
# Trace Validation: the Order Problem



# Trace Validation: the Order Problem

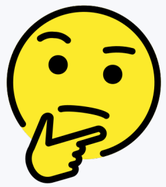


# Trace Validation: the Order Problem



# Trace Validation: Trouble with Levels of Detail

```
KVWrite("k1", "v1");
```

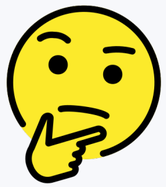


*Placeholder values*
*that don't match*
*the real system*
*... or log is incomplete*

```
> try put(key="x", value="y")
> tcp error
> retry
> timeout
> backoff
...
> retry
> ok
```

# Trace Validation: Trouble with Levels of Detail

```
KVWrite("k1", "v1");
```



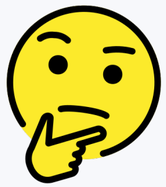
*Placeholder values*
*that don't match*
*the real system*
*... or log is incomplete*

```
> try put(key="x", value="y")
> tcp error
> retry
> timeout
> backoff
...
> retry
> ok
```

Log info that matches? Inconvenient, often impossible.

# Trace Validation: Trouble with Levels of Detail

```
KVWrite("k1", "v1");
```



*Placeholder values*
*that don't match*
*the real system*
*... or log is incomplete*

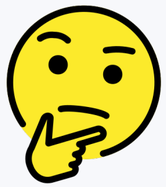
```
> try put(key="x", value="y")
> tcp error
> retry
> timeout
> backoff
...
> retry
> ok
```

Log info that matches? Inconvenient, often impossible.

Manually fix gaps in TLA+? Shown to work well, but not automatic.

# Trace Validation: Trouble with Levels of Detail

```
KVWrite("k1", "v1");
```



*Placeholder values*
*that don't match*
*the real system*
*... or log is incomplete*

```
> try put(key="x", value="y")
> tcp error
> retry
> timeout
> backoff
...
> retry
> ok
```

Log info that matches? Inconvenient, often impossible.

Manually fix gaps in TLA+? Shown to work well, but not automatic.

Use symbolic reasoning to lazy-fill spec holes? Potential future work.

# Trace Validation: In Practice

**eXtreme Modelling in Practice @ MongoDB [VLDB '20]**

*Tried matching logs with a spec, ran into trouble relating the 2 in a strict sense.*

**INSIGHT:** strict, direct comparison works poorly for complex systems.

**Bridging the Verifiability Gap @ Open Networking Foundation [TLA+Conf '20]**

*Used TLA+ properties (not the whole spec) as assertions over captured traces.*

**INSIGHT:** for some cases, you don't need the whole spec or refinement.

**Validating System Executions* with the TLA+ Tools @ Microsoft [TLA+Conf '24]**

*Developed state-based logging discipline and method for indirect spec-trace relationship.*

**INSIGHT:** you can patch "holes" in the trace with more TLA+ if you're careful.

# Generating Test Cases: In Practice

**Kayfabe, Model-based testing with TLA+ and Apalache [TLA+Conf '20]**

*For systems co-written with specs, control and trace evaluation w/ Apalache.*

**INSIGHT:** can build systems w/ a control interface for testing; manual but effective

**Using Lightweight Formal Methods to Validate a KV Storage Node in Amazon S3 [SOSP '21]**

*Wrote Rust programs that acted like TLA+ specs, compared running spec- and real-programs..*

**INSIGHT:** concrete programs can act like specs, though without direct TLA+ link

**Model Checking Guided Testing for Distributed Systems [EuroSys '23]**

*Read TLC state graph, generate synthetic test sequences for auto-instrumented real systems.*

**INSIGHT:** given additional manual TLA+ work, can test-drive concrete system with TLC

# Specification Compilation: Translating Data Structures

***Abstract definition of a log structure (from e.g. Raft spec)***

```
Record == [term: Nat, cmd: String, client: Nat]
Log == Seq(Record)
```

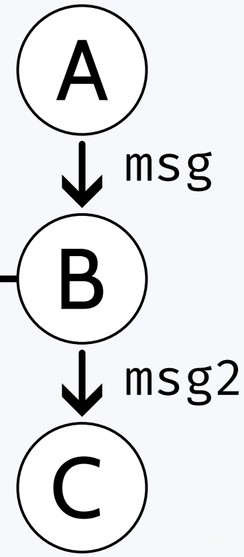
What data structure should the implementation use?
"Good enough" general structure?

... needs fast append, access to tail...
**!! must persist to disk**

# Specification Compilation: Hidden Control Flow

**Consider: critical section receives msg from node A, then sends msg2 to node C.**

```
MyCriticalSection:
  msg := read from A;
  msg2 := Process(msg);
  send msg2 to C;
```

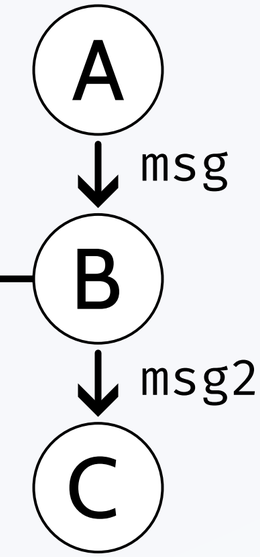



*Thanks to Markus for finding a real example of this in a hand-translated impl.*

# Specification Compilation: Hidden Control Flow

**Consider: critical section receives msg from node A, then sends msg2 to node C.**

```
MyCriticalSection:
  msg := read from A;
  msg2 := Process(msg);
  send msg2 to C;
```



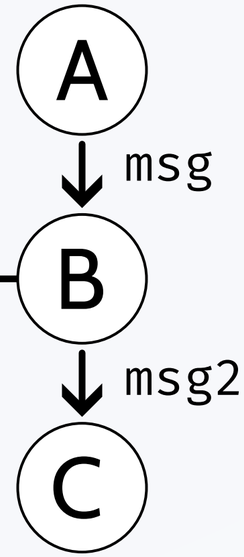

Even if we don't model it, this can fail in impl.

*Thanks to Markus for finding a real example of this in a hand-translated impl.*

# Specification Compilation: Hidden Control Flow

**Consider: critical section receives msg from node A, then sends msg2 to node C.**

```
MyCriticalSection:
  msg := read from A;
  msg2 := Process(msg);
  send msg2 to C;
```




Even if we don't model it, this can fail in impl.

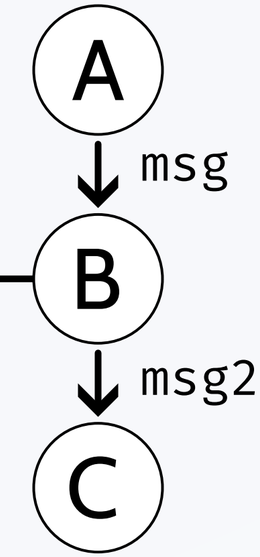
If we run these 3 lines as-is, failing send to C means we "forget" the first msg.
**Unsound!**

*Thanks to Markus for finding a real example of this in a hand-translated impl.*

# Specification Compilation: Hidden Control Flow

**Consider: critical section receives msg from node A, then sends msg2 to node C.**

```
MyCriticalSection:
  msg := read from A;
  msg2 := Process(msg);
  send msg2 to C;
```




Even if we don't model it, this can fail in impl.

If we run these 3 lines as-is, failing send to C means we "forget" the first msg.
**Unsound!**

A correct implementation must "remember" msg until it can send msg2!

*Thanks to Markus for finding a real example of this in a hand-translated impl.*

# Specification Compilation: What if it Goes Wrong?

# Specification Compilation: What if it Goes Wrong?

Entirely correct system could be misconfigured

# Specification Compilation: What if it Goes Wrong?

Entirely correct system could be misconfigured

Model could make unrealistic assumptions (assume lossless net, get lossy)

# Specification Compilation: What if it Goes Wrong?

Entirely correct system could be misconfigured

Model could make unrealistic assumptions (assume lossless net, get lossy)

Compiler could output wrong code

# Specification Compilation: What if it Goes Wrong?

Entirely correct system could be misconfigured

Model could make unrealistic assumptions (assume lossless net, get lossy)

Compiler could output wrong code

For pt. 3, could formally verify compiler, e.g. CompCert [ERST '16]

# Specification Compilation: What if it Goes Wrong?

Entirely correct system could be misconfigured

Model could make unrealistic assumptions (assume lossless net, get lossy)

Compiler could output wrong code

For pt. 3, could formally verify compiler, e.g. CompCert [ERST '16]

Can do trace validation on compiled system. Might be easier to automate?

# Specification Compilation: In Practice

**tlaplus/PlusPy:** *evaluates TLA+ actions and expressions. Ignores hidden control flow.*

**Elixir Translator [SAST, TLA+Conf '22]:** *translates TLA+ actions into Elixir code.*

*Translation is literal, primarily for monitoring.*

**PGo [ASPLOS '23, TLA+Conf '22 '19]:** *compiles Modular PlusCal into Go w/ custom IO options.* 

*Uses special protocol to auto-implement hidden control flow; evaluated on full-scale systems.*

**Choreographic PlusCal [TASE '23]:** *compiles TLA+ actions into Go monitors.*

🤔 **... compilation seems popular for monitoring implementations ...**

# Specification Compilation: In Practice

**tlaplus/PlusPy:** *evaluates TLA+ actions and expressions. Ignores hidden control flow.*

**Elixir Translator [SAST, TLA+Conf '22]:** *translates TLA+ actions into Elixir code.*

*Translation is literal, primarily for monitoring.*

**PGo [ASPLOS '23, TLA+Conf '22 '19]:** *compiles Modular PlusCal into Go w/ custom IO options.* 

*Uses special protocol to auto-implement hidden control flow; evaluated on full-scale systems.*

👉 *Currently, only full Spec2Code attempt.*

**Choreographic PlusCal [TASE '23]:** *compiles TLA+ actions into Go monitors.*

🤔 **... compilation seems popular for monitoring implementations ...**