



School of Computer Science & Engineering
Trustworthy Systems Group



Verifying Device Drivers with Pancake

Junming Zhao

PhD Student, UNSW Sydney

junming.zhao@unsw.edu.au

Supervisors:

Dr. Rob Sison

Dr. Thomas Sewell

Scientia Professor Gernot Heiser

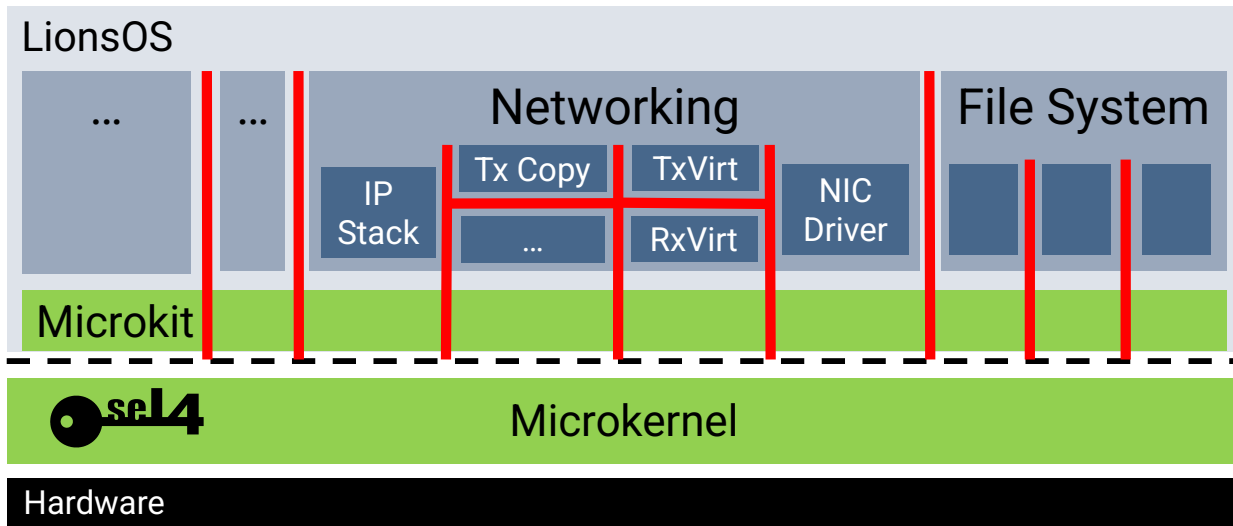
1. **Problem: Verify LionsOS subsystems**

2. Recap: Pancake

3. Pancake-to-Viper

4. Ethernet Driver Verification

5. Concurrent Composition



- Binary-level
- Repeatable
- Concurrency

Approach: Pancake with toolchains

1. Problem: Verify LionsOS subsystems

2. **Recap: Pancake**

3. Pancake-to-Viper

4. Ethernet Driver Verification

5. Concurrent Composition



PANCAKE

A Language for
Verified Systems Programming

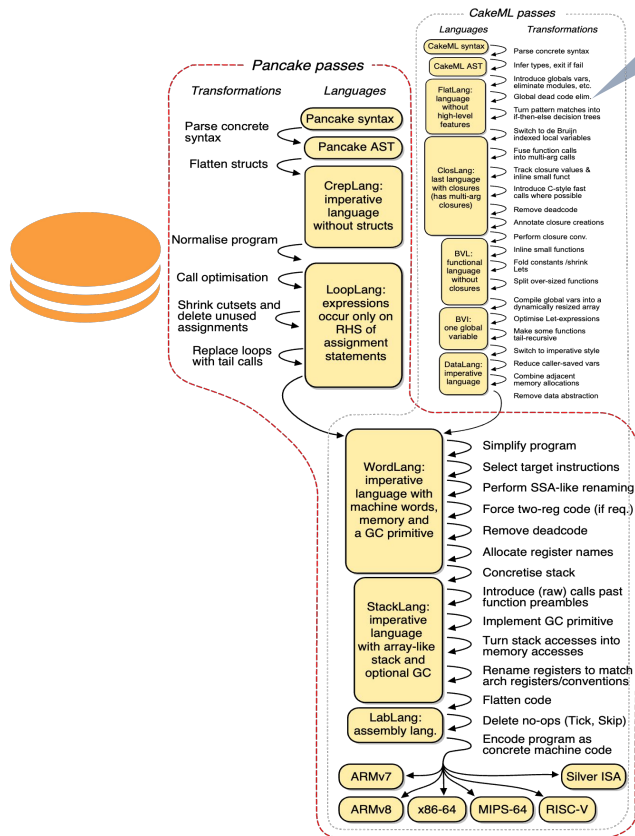


CakeML



Main Features

- System Programming
- Verified Compiler



1. Problem: Verify LionsOS subsystems
2. Recap: Pancake
- 3. Pancake-to-Viper**
4. Ethernet Driver Verification
5. Concurrent Composition



Viper



• ETH, Zurich

• Intermediate language for verification

• Automated verifier

• Various frontend tools

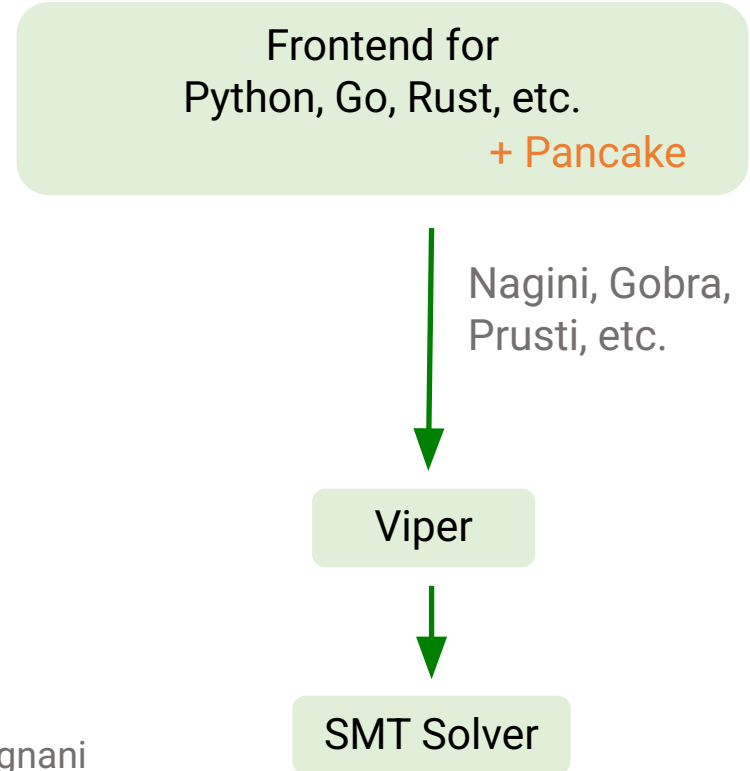
Viper (Verification Infrastructure for Permission-based Reasoning) is a language and suite of tools, providing an architecture on which new verification tools and prototypes can be developed simply and quickly. Viper is being developed at ETH Zurich in close collaboration with the team of Alex Summers at UBC.



Viper



- ETH, Zurich
- Intermediate language for verification
- Automated verifier
- Various frontend tools



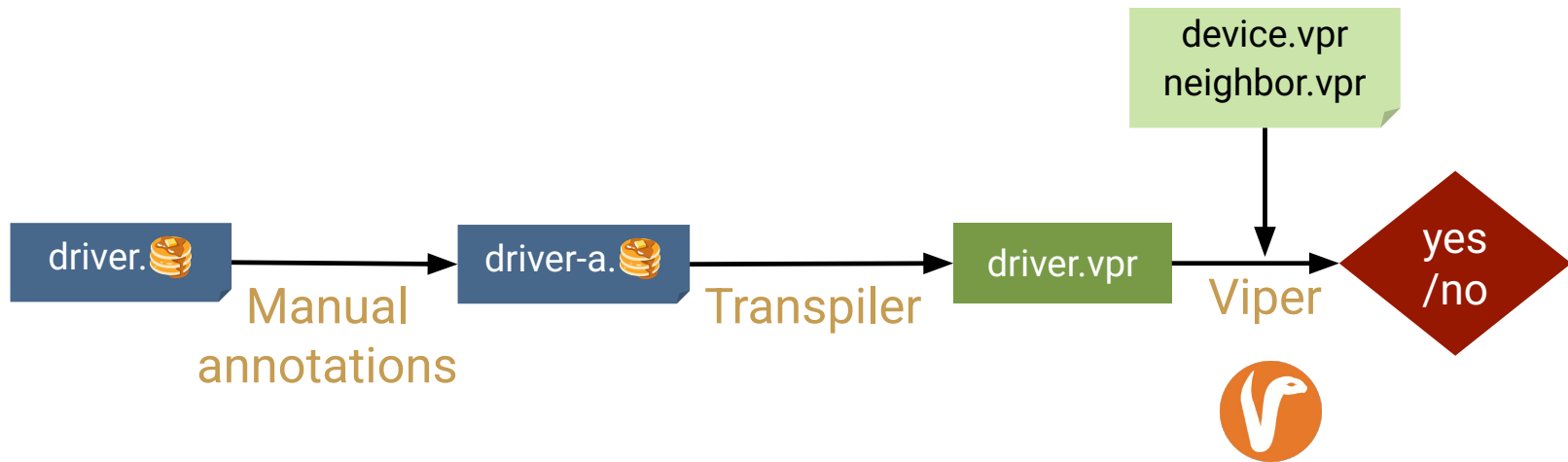
ETH student: Alessandro Legnani

trustworthy.systems/pancake-playground





Pancake-to-Viper

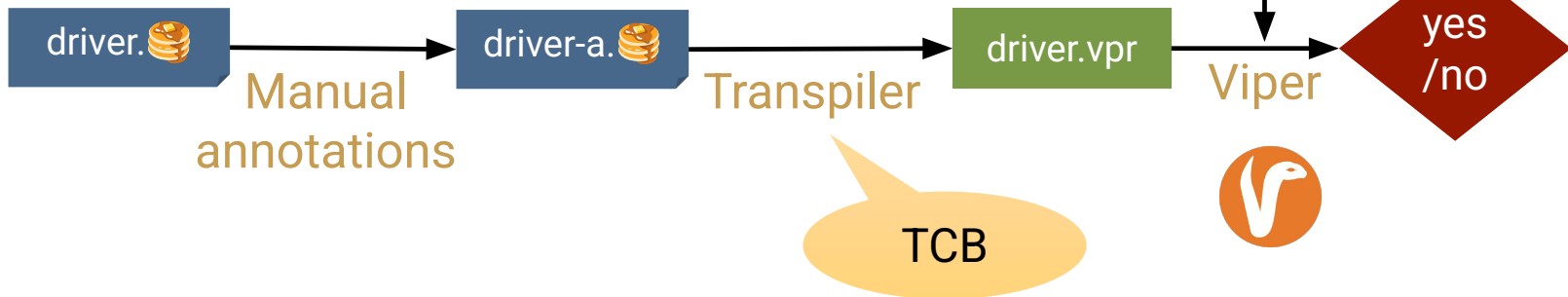




RESEARCH-ARTICLE | [OPEN ACCESS](#) |

Towards Trustworthy Automated Program Verifiers: Formally Validating Translations into an Intermediate Verification Language

Authors: [Gaurav Parthasarathy](#), [Thibault Dardinier](#), [Benjamin Bonneau](#), [Peter Müller](#), [Alexander J. Summers](#) | [Authors Info & Claims](#)

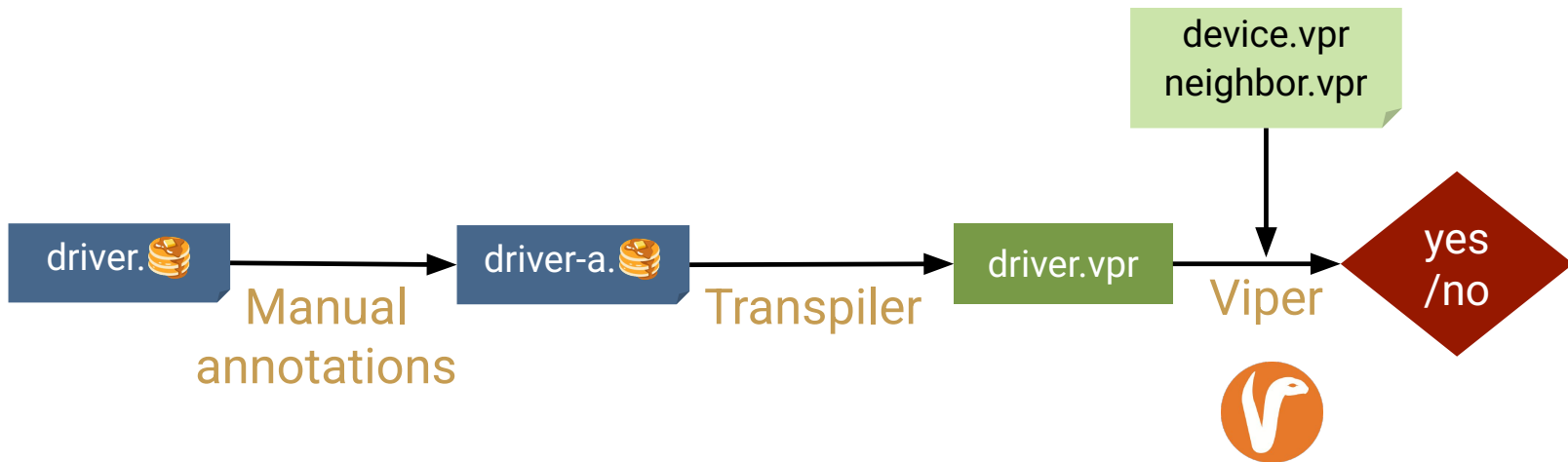




Pancake-to-Viper



Automated theorem proving: Spec \rightarrow binary

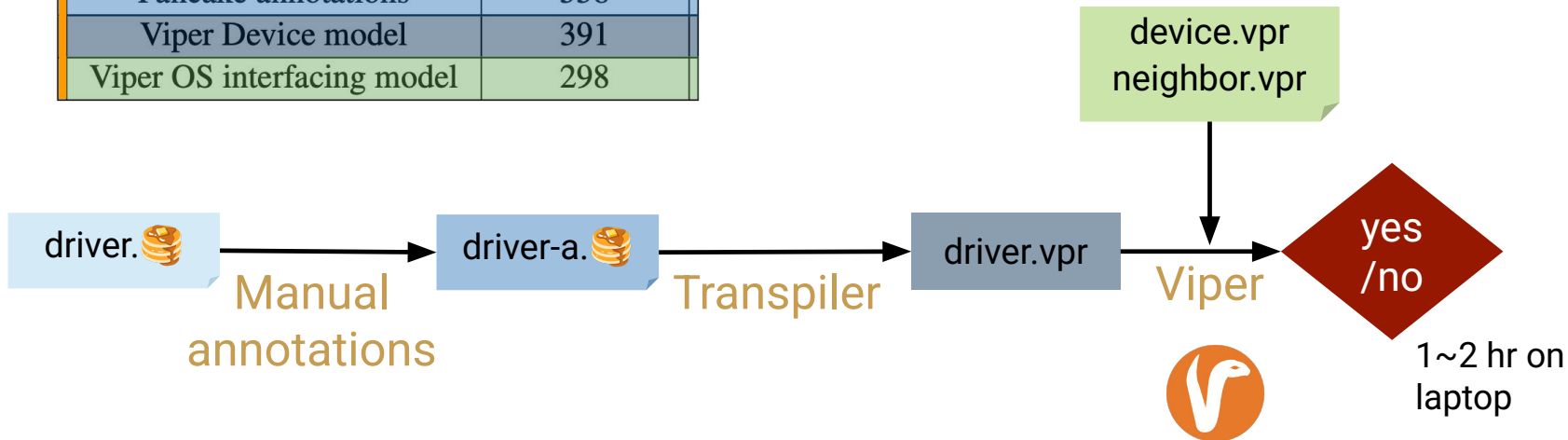


1. Problem: Verify LionsOS subsystems
2. Recap: Pancake
3. Pancake-to-Viper
- 4. Ethernet Driver Verification**
5. Concurrent Composition

Ethernet Driver Verification



Component	Line Count
C Driver	351
Pancake Driver	411
Pancake annotations	558
Viper Device model	391
Viper OS interfacing model	298



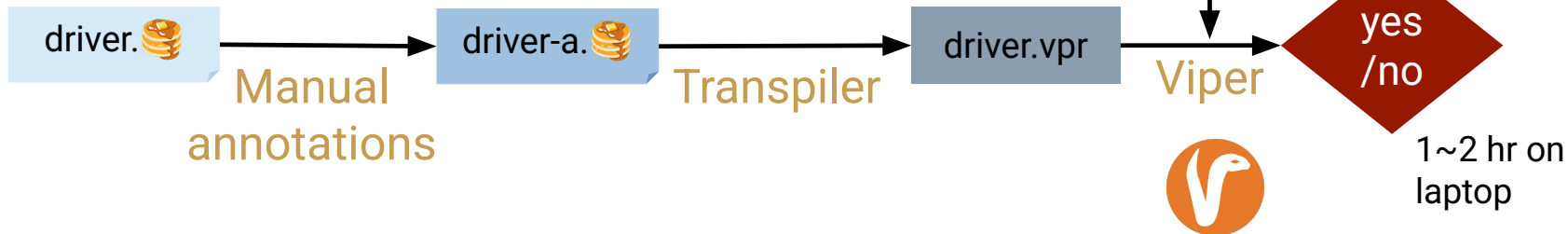
Ethernet Driver Verification



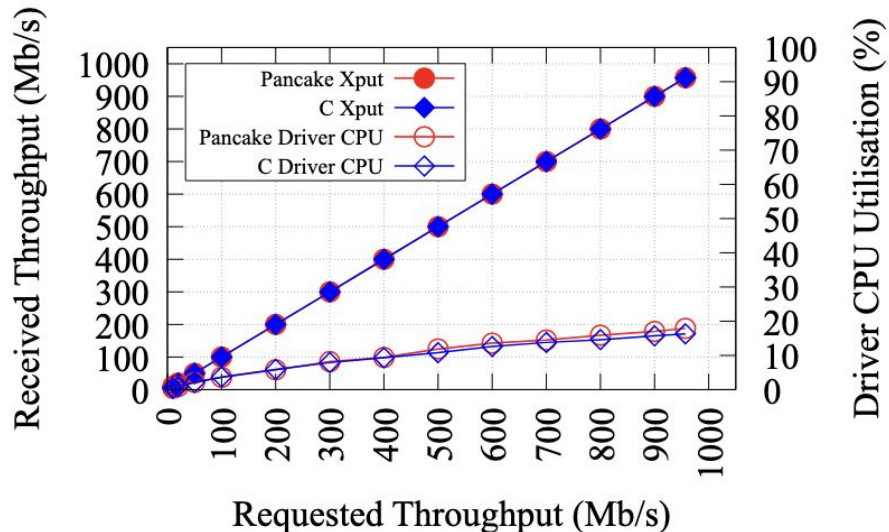
Component	Line Count
C Driver	351
Pancake Driver	411
Pancake annotations	558
Viper Device model	391
Viper OS interfacing model	298

Properties

- Device Safety
- Data integrity
- Memory isolation




Ethernet Driver Verification



~20% Overhead vs. Unverified C driver

Ethernet Driver Verification



 [arXiv](#) > [cs](#) > [arXiv:2501.08249](#)

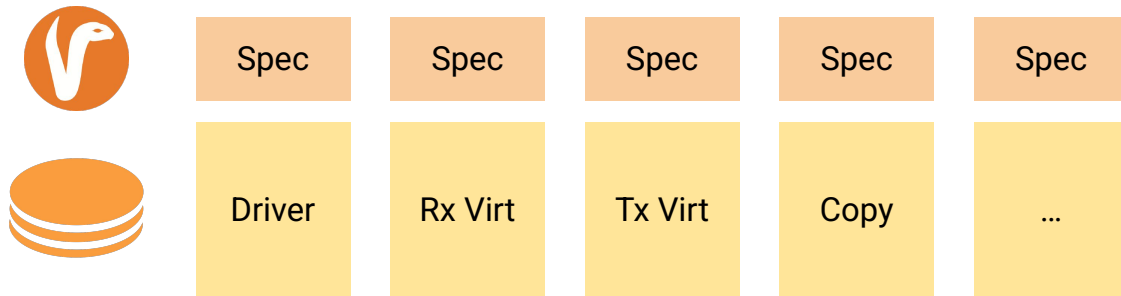
Search...
Help | Adv

Computer Science > Programming Languages

[Submitted on 14 Jan 2025 (v1), last revised 30 May 2025 (this version, v2)]

Verifying Device Drivers with Pancake

Junming Zhao, Miki Tanaka, Johannes Åman Pohjola, Alessandro Legnani, Tiana Tsang Ung, H. Truong, Tsun Wang Sau, Thomas Sewell, Rob Sison, Hira Syeda, Magnus Myreen, Michael Norrish, Gernot Heiser

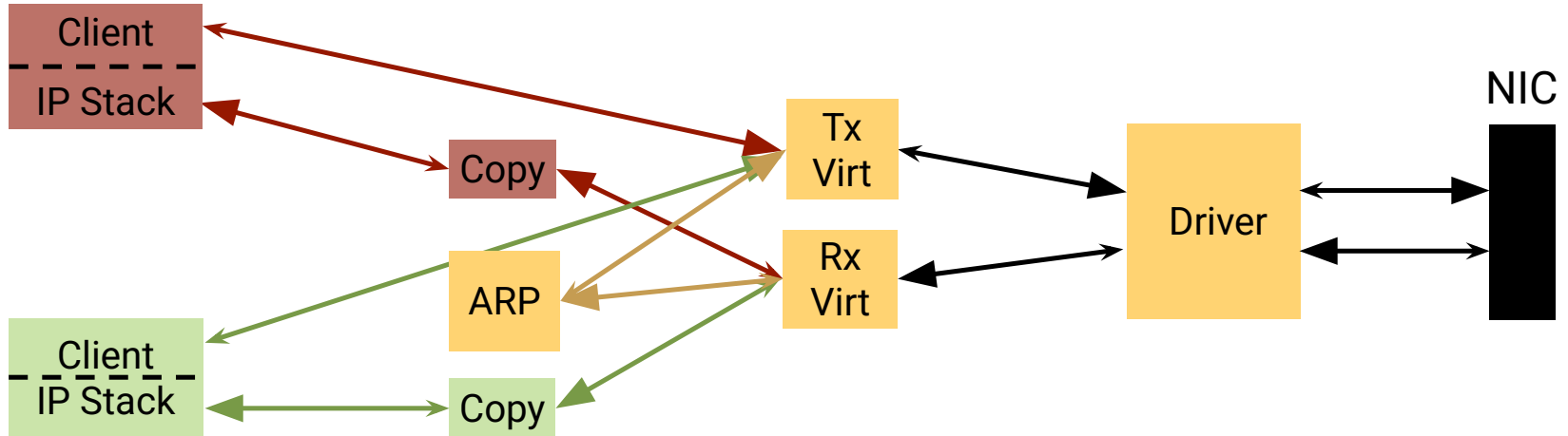


1. Problem: Verify LionsOS subsystems
2. Recap: Pancake
3. Pancake-to-Viper
4. Ethernet Driver Verification
- 5. Concurrent Composition**

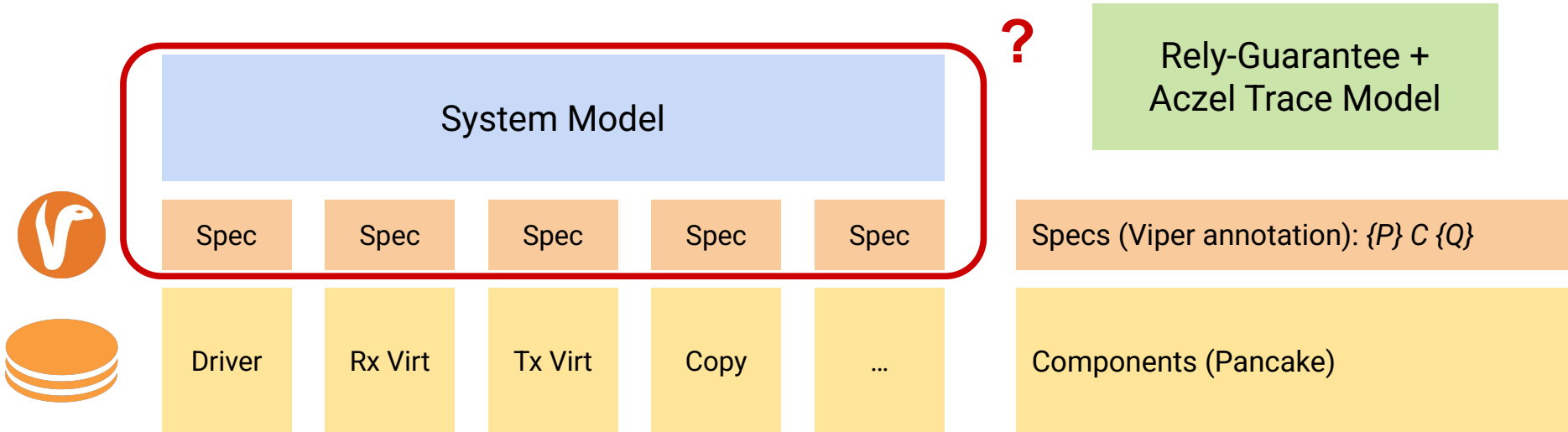
Concurrent Composition



Viper Model: Sequential $\{P\} C \{Q\}$



Concurrent Composition



Concurrent Composition



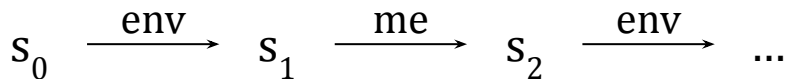
Rely-Guarantee — $\{P, R\} \text{ } C \text{ } \{Q, G\}$

- **Pre-condition** (P): Initial state
- **Post-condition** (Q): Final state
- **Rely** (R): **Environmental** steps
- **Guarantee** (G): **My** steps

$\text{rely_env}(s_0, s_1): \text{bool}$

$\text{guar_me}(s_1, s_2): \text{bool}$

Aczel trace model



Concurrent Composition



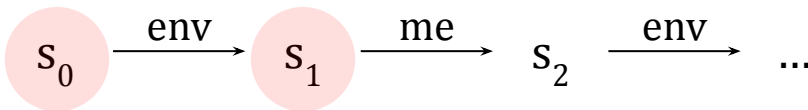
Rely-Guarantee — $\{P, R\} \text{ } C \text{ } \{Q, G\}$

- **Pre-condition** (P): Initial state
- **Post-condition** (Q): Final state
- **Rely** (R): **Environmental** steps
- **Guarantee** (G): **My** steps

$\text{rely_env}(s_0, s_1): \text{bool}$

$\text{guar_me}(s_1, s_2): \text{bool}$

Aczel trace model



Concurrent Composition



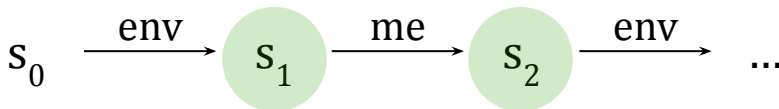
Rely-Guarantee — $\{P, R\} \text{ } C \text{ } \{Q, G\}$

- **Pre-condition** (P): Initial state
- **Post-condition** (Q): Final state
- **Rely** (R): **Environmental** steps
- **Guarantee** (G): **My** steps

$\text{rely_env}(s_0, s_1): \text{bool}$

$\text{guar_me}(s_1, s_2): \text{bool}$

Aczel trace model



Concurrent Composition



Aczel trace model $s_0 \xrightarrow{\text{env}} s_1 \xrightarrow{\text{me}} s_2 \xrightarrow{\text{env}} s_3$

```
method interference(shared_mem: Seq[Ref])  
  ensures rely_env(old(shared_mem), shared_mem)
```

$\text{rely_env}(s_0, s_1): \text{bool}$

```
method shared_store(shared_mem: Seq[Ref], addr: Int, value: Int)  
  ensures guar_me(old(shared_mem), shared_mem)
```

$\text{guar_me}(s_1, s_2): \text{bool}$

Concurrent Composition



Aczel trace model $s_0 \xrightarrow{\text{env}} s_1 \xrightarrow{\text{me}} s_2 \xrightarrow{\text{env}} s_3$

```
method interference(shared_mem: Seq[Ref])
  ensures rely_env(old(shared_mem), shared_mem)
```

```
method shared_store(shared_mem: Seq[Ref], addr: Int, value: Int)
  ensures guar_me(old(shared_mem), shared_mem)
```

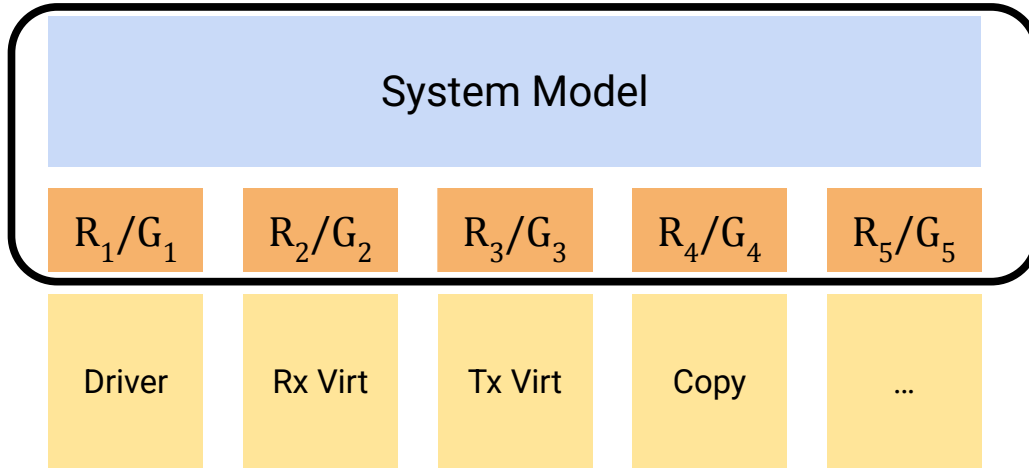
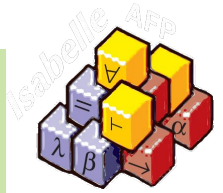
```
method some_function(shared_mem: Seq[Ref])
  requires precondition(shared_mem)
  ensures postcondition(shared_mem)
{
  interference(shared_mem); // s0 --(env)--> s1
  shared_store(shared_mem, addr, value); // s1 --(me)--> s2
  interference(shared_mem); // s2 --(env)--> s3
}
```

$\text{rely_env}(s_0, s_1): \text{bool}$

$\text{guar_me}(s_1, s_2): \text{bool}$

$\{ [s_0, s_1, s_2, s_3] \mid$
 $\text{pre}(s_0) \wedge$
 $\text{rely_env}(s_0, s_1) \wedge$
 $\text{guar_me}(s_1, s_2) \wedge$
 $\text{rely_env}(s_2, s_3) \wedge$
 $\text{post}(s_3) \}$

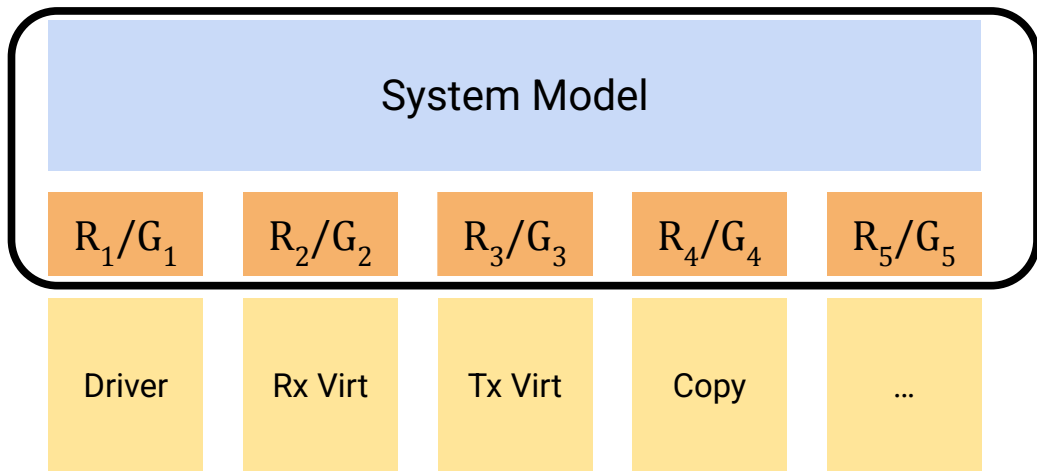
Concurrent Composition



Rely-Guarantee +
Aczel Trace Model

$$\forall i : \bigwedge_j \in \{1, \dots, n\} \setminus \{i\} G_j \models R_i$$

Concurrent Composition

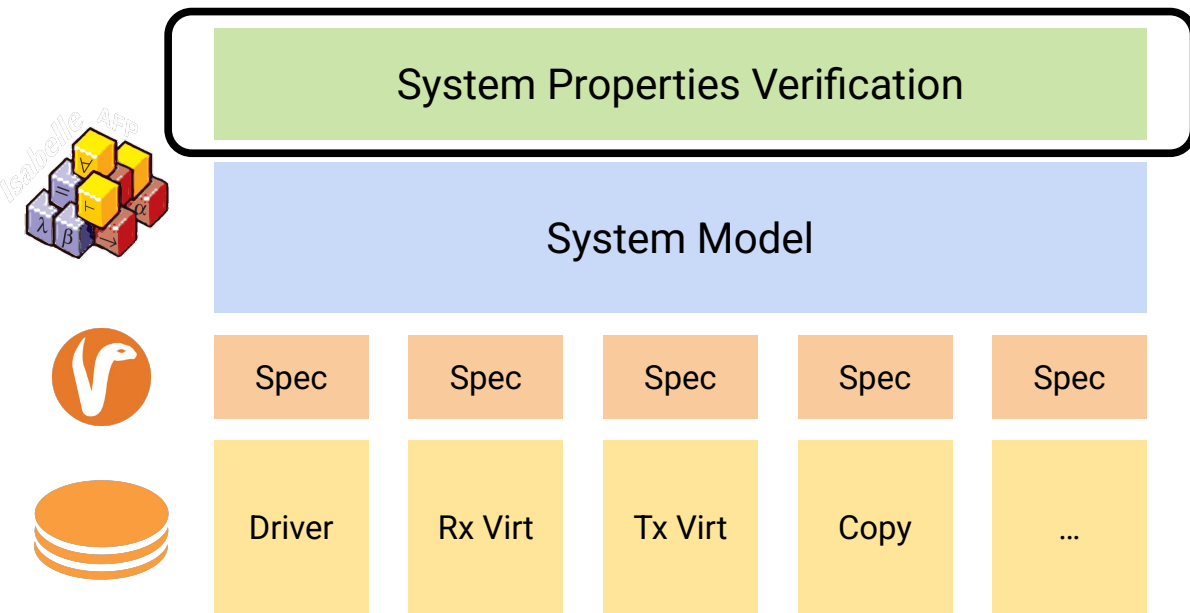


l4v trace monad lib

```
type_synonym ('s, 'a) tmonad =  
  "'s  $\Rightarrow$   
  ((tmid  $\times$  's) list  $\times$  ('s, 'a) tmres)  
  set"
```

```
definition validI ::  
  "('s  $\Rightarrow$  's  $\Rightarrow$  bool)  $\Rightarrow$   
  's rg_pred  $\Rightarrow$   
  ('s, 'a) tmonad  $\Rightarrow$   
  's rg_pred  $\Rightarrow$   
  ('a  $\Rightarrow$  's  $\Rightarrow$  's  $\Rightarrow$  bool)  $\Rightarrow$   
  bool"  
  ("({_}, /{_}) / _ / ({_}, /{_})")  
where  
  "{P}, {R} f {G}, {Q}  $\equiv$  ..."
```

Concurrent Composition



- **Correctness Properties**
 - `recv/send()`
- **Safety Properties**
 - Memory safety
 - Protocol adherence
 - Device safety
- **Liveness Properties**
 - Packet delivery
 - ...

Recap:

Verifying Device Driver w/ Pancake

- (Relatively) Low verification effort
- Compositional verification for concurrency
- Applicable to real-world, performant systems

Repeatable, systematic approach for verifying seL4-based services



Thank you