

Stop testing, start proving

Valentin Bergeron

Typelevel Traverses Berlin

Follow along



Slides

<https://vbergeron.github.io/rocq-solid/decks/stop-testing-long.pdf>



Site

<https://vbergeron.github.io/rocq-solid/>

Everything was green

The context

- A Scala backend for a small e-commerce startup
- A roadmap that doubles every quarter
- Claude writes the code. Claude writes the tests. Life is good.



Hey Claude, can you implement partial refunds please ?
Make no mistakes !!

Hey Claude, can you implement partial refunds please ?
Make no mistakes !!

Result: **180k€ loss in one weekend**

Before

```
case class Order(id: OrderId, paid: Money, refund: Option[Refund])

def canRefund(o: Order, amount: Money): Boolean =
  o.refund.isEmpty || o.refund.head.amount + amount ≤ o.paid
```

After

```
case class Order(id: OrderId, paid: Money, refunds: List[Refund])

def canRefund(o: Order, amount: Money): Boolean =
  o.refunds.isEmpty || o.refunds.head.amount + amount ≤ o.paid
```

Vacuously True Driven Development

The test

```
test("a refund never exceeds what was paid") {  
  val order    = Fixtures.paidOrder(amount = 100)  
  val refunds = db.refunds.findByOrder(order.id)  
  refunds.foreach { r =>  
    assert(r.amount ≤ order.paid)  
  }  
}
```

Vacuously True Driven Development

Meanwhile, in `Fixtures.scala`

```
object Fixtures:  
  def paidOrder(amount: Money): Order =  
    db.orders.insert(Order(OrderId.random(), paid = amount, refunds = Nil))
```

Everything was green

Every test was green

Every pull request was reviewed and approved

The code did exactly what it said

Nobody wrote down **what must be true.**



It is all about proofs and trust

It is all about proofs and trust.

Descartes as a shadow FP maximalist ?



Empiricists

Knowledge comes from **experience**

Locke, Berkeley, Hume

VS



Rationalists

Knowledge comes from **reason**

Descartes, Spinoza, Leibniz

Descartes as a shadow FP maximalist ?



Testers

Trust the CI bro

Locke, Berkeley, Hume

VS



Typers

If it compiles it works

Descartes, Spinoza, Leibniz

Collatz: evidence without proof

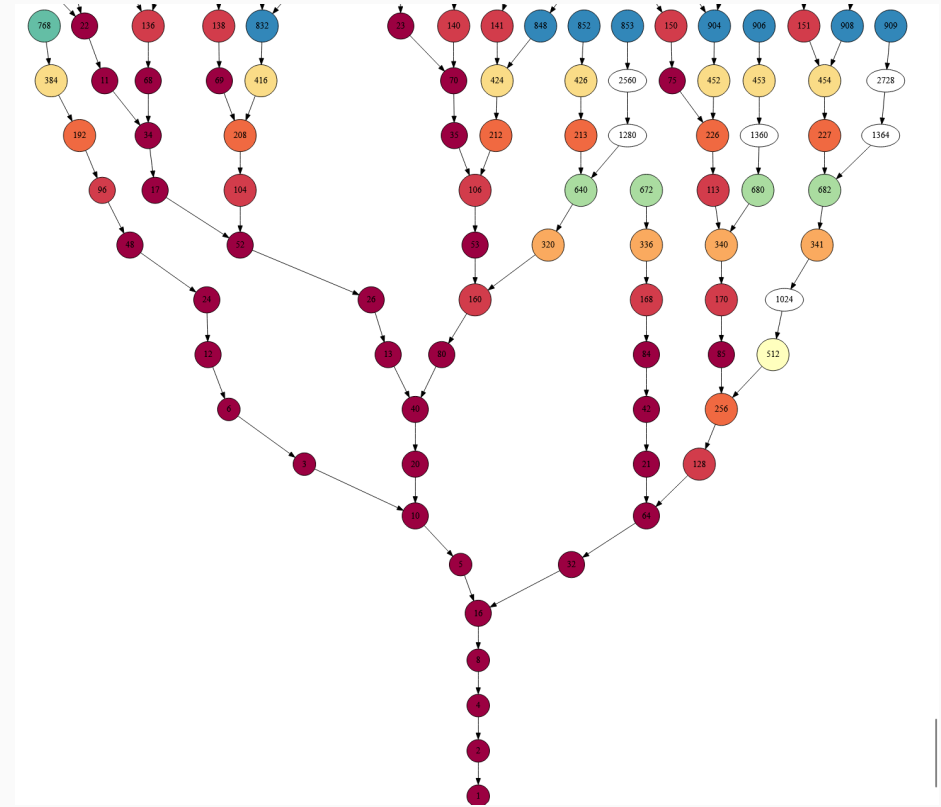
The rule: take any n . If even, $n \rightarrow \frac{n}{2}$; if odd, $n \rightarrow 3n + 1$. Repeat.

Conjecture (Collatz, 1937): you always end up at 1.

$6 \rightarrow 3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$

- Checked for every n up to $2^{71} \approx 2.36 \times 10^{21}$ (Barina, 2025)
- Still no proof

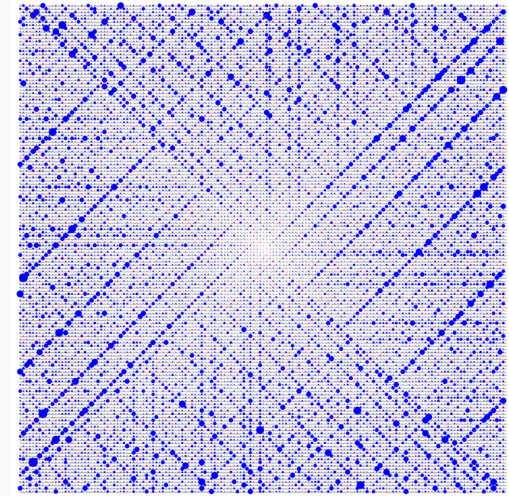
Billions of billions of cases $\neq$ certainty: the next number could be the exception.



Reason alone: truths no experiment can reach

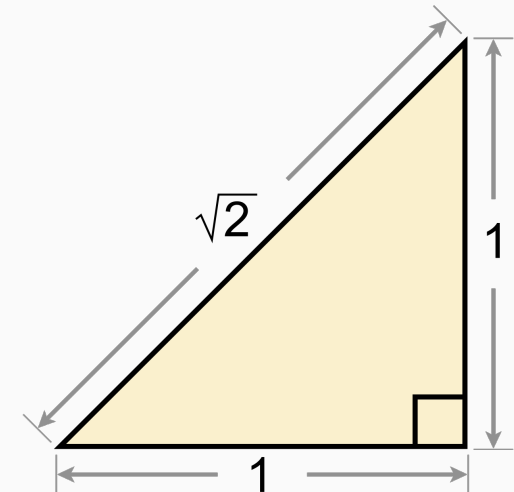
There are infinitely many primes (Euclid, c. 300 BCE)

Suppose the list is finite: $p_1, \dots, p_n$. Then $N = p_1 p_2 \cdots p_n + 1$ leaves remainder 1 when divided by each p_i , so its prime factors are missing from the list. Contradiction.



$\sqrt{2}$ is irrational (Pythagoreans, 5th c. BCE)

Suppose $\sqrt{2} = \frac{p}{q}$ in lowest terms. Then $p^2 = 2q^2$, so p is even: $p = 2k$. Then $q^2 = 2k^2$, so q is even too. Contradiction.



Theorem provers

Theorem (Curry 1934–1958, Howard 1969).

Proofs in intuitionistic logic are exactly the programs of the typed λ -calculus: a proposition is provable if and only if the corresponding type has a program.

Checking a proof = checking a program's type.

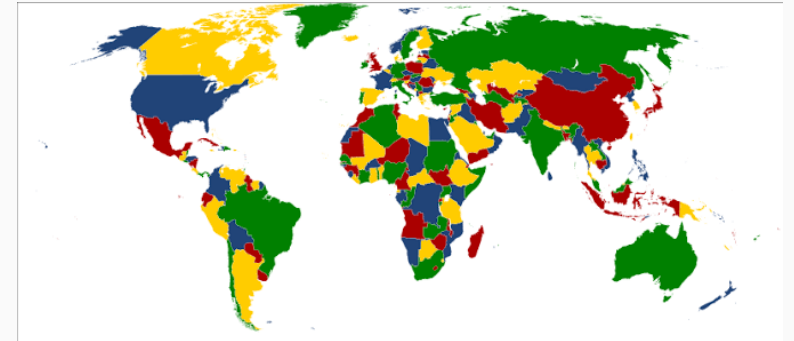
A theorem prover **is** a type checker
for a language whose type system is powerful enough
to express **arbitrary statements about its terms**.

The Four Colour Theorem: a proof no human can read

Any map can be coloured with 4 colours so that neighbouring regions differ.

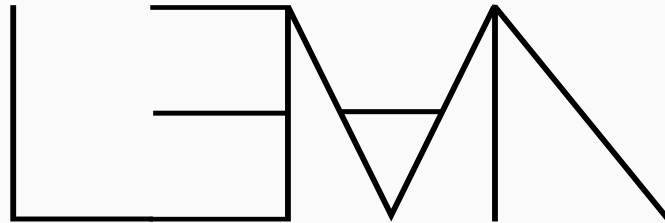
- 1879** Kempe publishes a “proof”, accepted for 11 years
- 1890** Heawood finds the flaw: back to square one
- 1976** Appel & Haken: a computer checks 1,900 cases in over 1,000 hours
- 1996** Robertson, Sanders, Seymour & Thomas: a simpler proof, still computer-assisted
- 2005** Gonthier verifies the full proof in the Coq proof assistant

If no human can check it, is it still a proof? Do we *know* the theorem, or do we *trust* the machine?



- 1984** Coquand and Huet start it at Inria
- 1989** First release, named **Coq**
- 2005** **The four colour theorem**, checked
- 2006** CompCert, a C compiler proved correct
- 2013** ACM Software System Award
- 2025** Renamed **Rocq**, after Rocquencourt





Isabelle

An OS kernel, proved

seL4 flew a military helicopter that red teams could not break into

F*

You used it today

HACL*, the proved crypto inside Firefox, Linux and Python

Dafny

At the heart of AWS

The engine that evaluates IAM policies, rewritten and proved

TLA+

Bugs caught before the code

Amazon found subtle design bugs in S3 and DynamoDB

ACL2

The Pentium, never again

AMD proved its floating point division in 1996

Agda

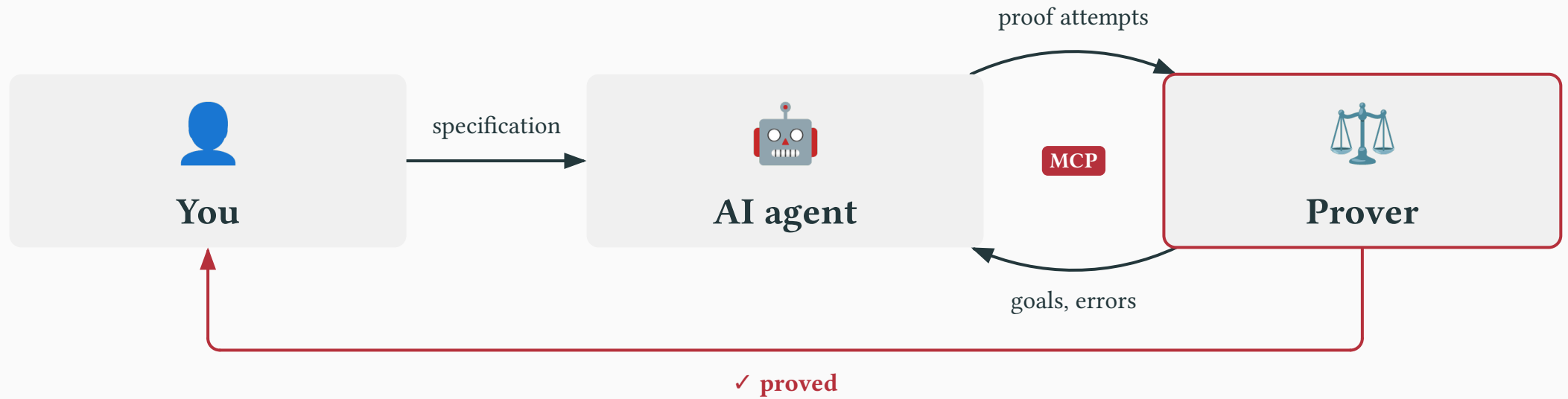
Proofs are programs

The playground where type theorists try their ideas first

Vericoding

Writing a proof is hard.

Writing **what you want to prove** is much easier.



Example: the code

```
Inductive event :=  
| Paid (id amount : nat)  
| Refunded (id amount : nat).  
  
Record order := mk_order {  
  seen : list nat;    (* ids already handled *)  
  paid : nat;  
  refunded : nat  
}.
```

```
(* Was this event already handled? *)  
Definition already_seen  
  (o : order) (e : event) : bool := ...  
  
(* Apply one event to an order. *)  
Definition handle  
  (o : order) (e : event) : order := ...  
  
(* The order, rebuilt from its events. *)  
Definition replay  
  (events : list event) : order := ...
```

Example: the specification

```
Theorem never_refund_more_than_paid : forall (events : list event),  
  refunded (replay events) ≤ paid (replay events).
```

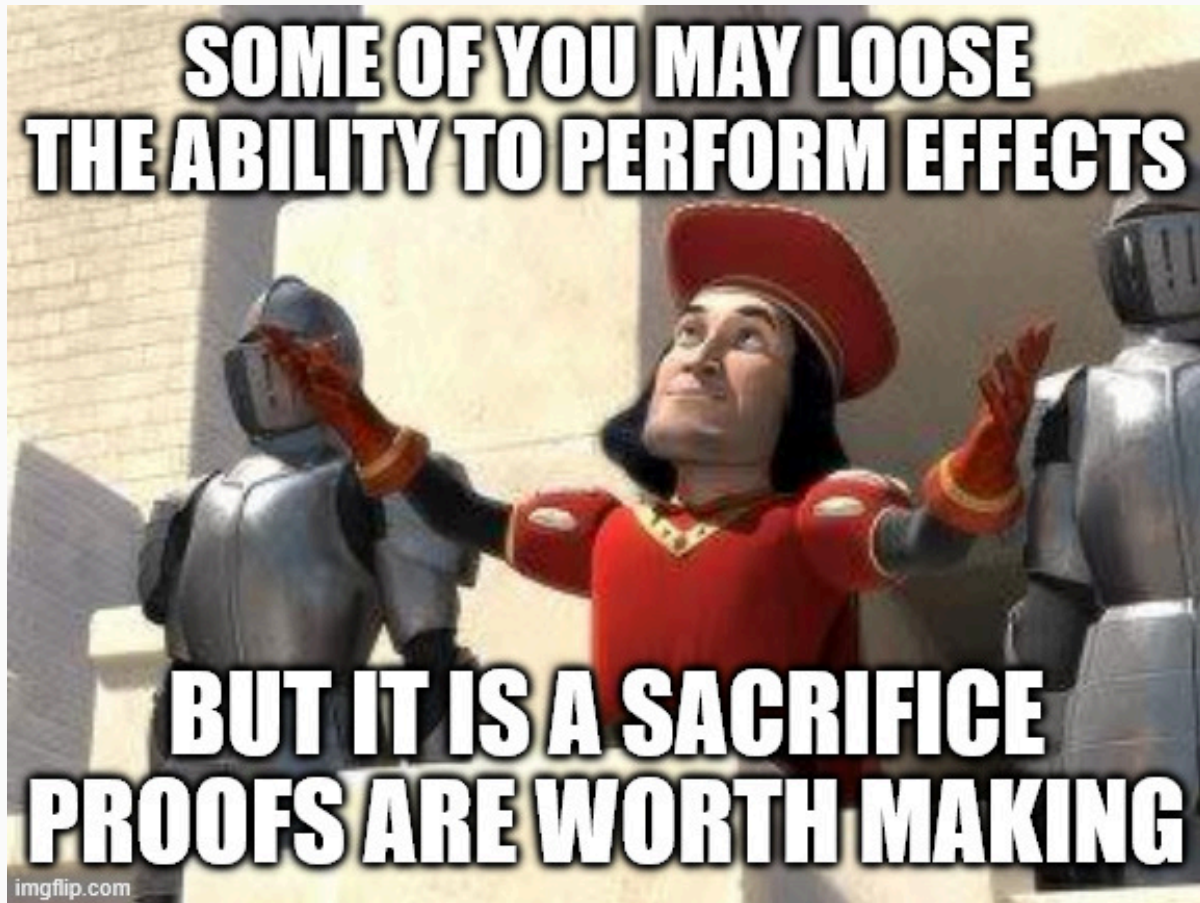
- Written by **you**: an order never refunds more than it was paid
- True for **every** sequence of events: not just the cases you thought of testing
- Two lines to review, readable **without the proof**

Example: the proof

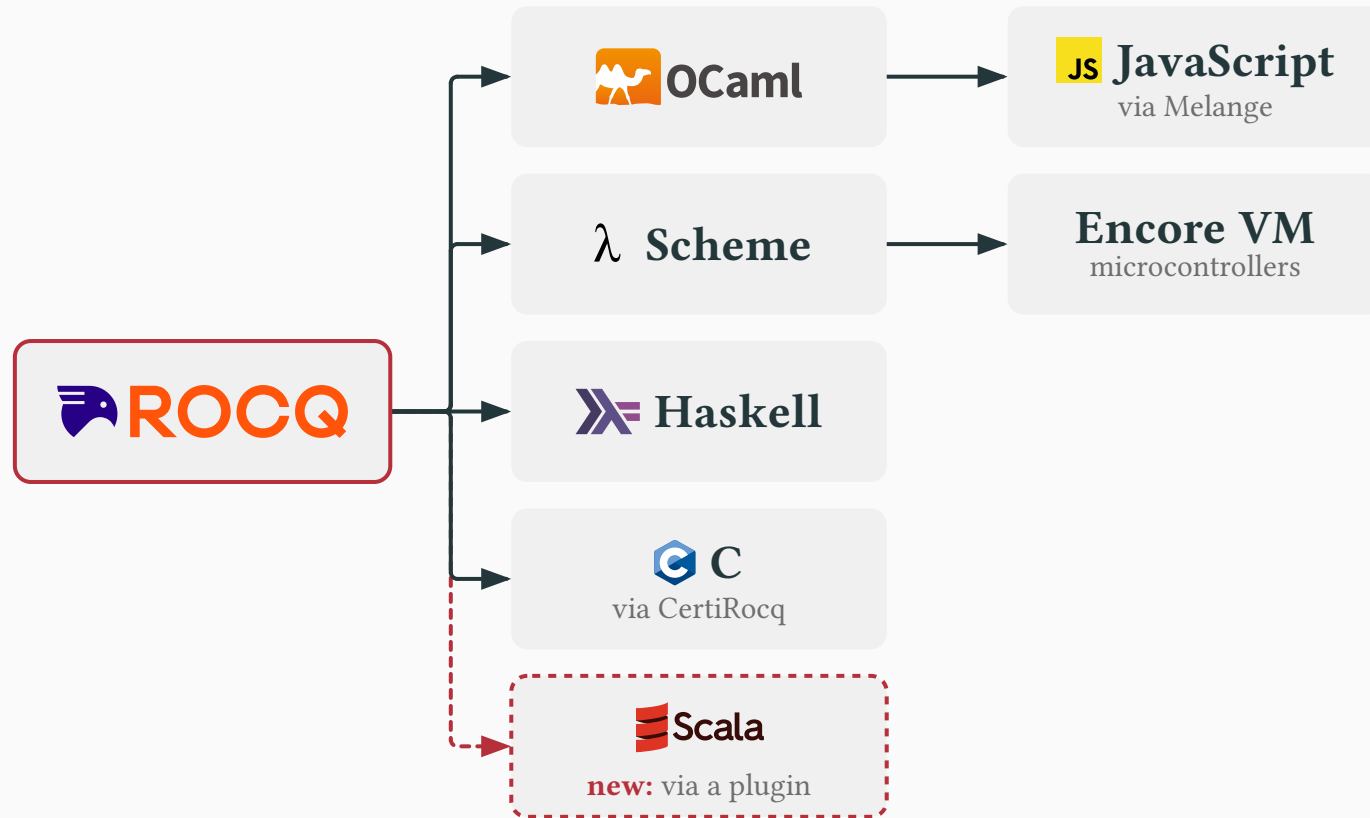
```
Lemma handle_keeps_refunds : forall o e,  
  refunded o ≤ paid o → refunded (handle o e) ≤ paid (handle o e).  
Proof.  
  intros o e H. unfold handle.  
  destruct (already_seen o e); [exact H|].  
  destruct e as [i n | i n]; cbn; [lia|].  
  destruct (refunded o + n ≤? paid o) eqn:E; cbn; [apply Nat.leb_le in E|]; lia.  
Qed.  
  
Theorem never_refund_more_than_paid : forall (events : list event),  
  refunded (replay events) ≤ paid (replay events).  
Proof.  
  intro events. unfold replay.  
  assert (H : refunded empty ≤ paid empty) by (cbn; lia).  
  revert H. generalize empty as o.  
  induction events as [| e es IH]; intros o H; cbn; [exact H|].  
  apply IH, handle_keeps_refunds, H.  
Qed.
```

- Written by **the agent**: first a lemma about **one** event, then induction over **all** of them
- Qed.: Rocq accepted every step. A wrong proof **does not compile**
- You do not have to read it, only the statement

Shipping proved code

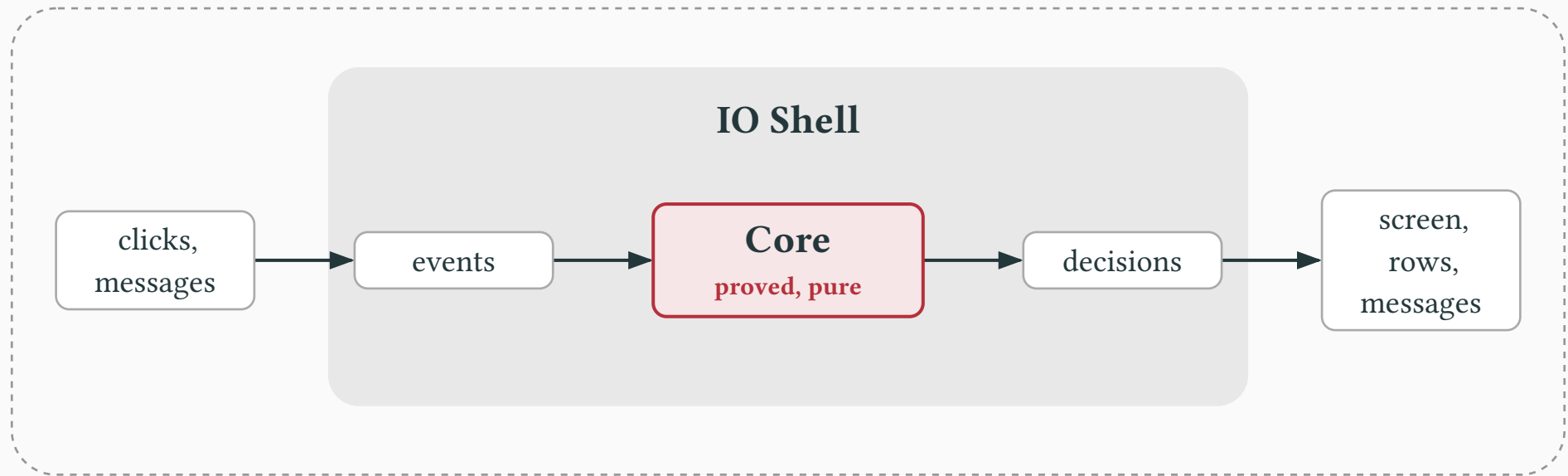


Extraction: from proof to production



What it changes in your architecture

A proved decisional **core**, a thin interpreting **shell**



Where it fits best

Reducers

$(\text{state}, \text{action}) \Rightarrow \text{state}$

Forms, wizards, carts
Undo / redo
Redux, Elm, useReducer

Event processors

$(\text{state}, \text{event}) \Rightarrow (\text{state}, \text{decisions})$

Kafka consumers
Event sourcing, ledgers
Replay **is** a fold

Actor behaviours

$(\text{state}, \text{message}) \Rightarrow (\text{state}, \text{replies})$

Pekko / Akka Behavior
Protocols, sessions
Firmware state machines

What it changes in your codebase

- **Tests** move to the shell around it
- **Review** moves to the specifications. Proofs are machine checked
- **CI** checks the proofs on every build
- **Rules** are written down. Invariants cannot rot

Be the one who writes down **what must be true.**

Thank you for your attention

Questions?



Slides

<https://vbergeron.github.io/rocq-solid/decks/stop-testing-long.pdf>



Site

<https://vbergeron.github.io/rocq-solid/>