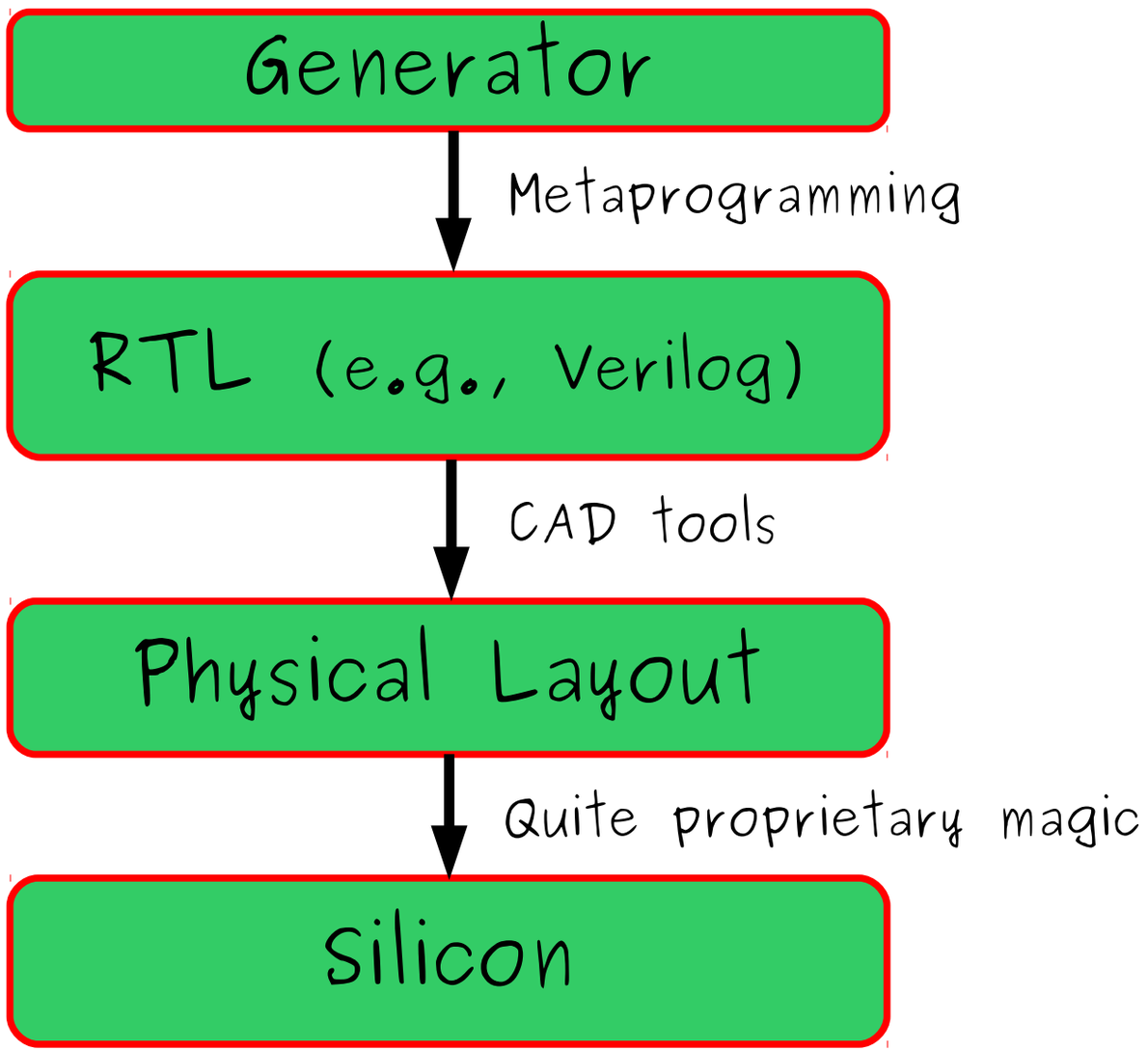
Strong Formal Verification for RISC-V

From Instruction-Set Manual to RTL

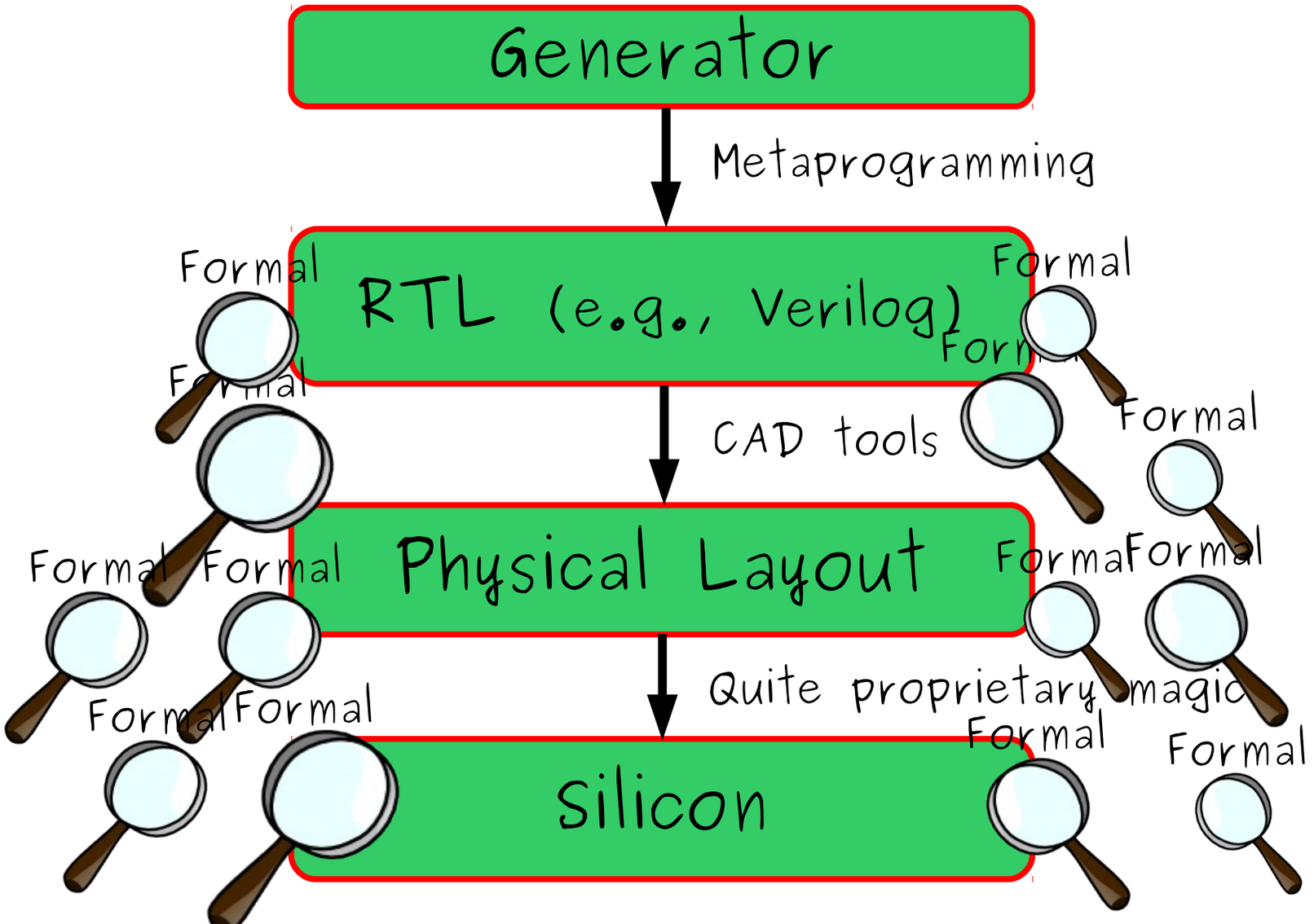
Adam Chlipala
MIT CSAIL
RISC-V Workshop
November 2017

Joint work with: Arvind, Thomas Bourgeat, Joonwon Choi, Ian Clester, Samuel Duchovni, Jamey Hicks, Muralidaran Vijayaraghavan, Andrew Wright

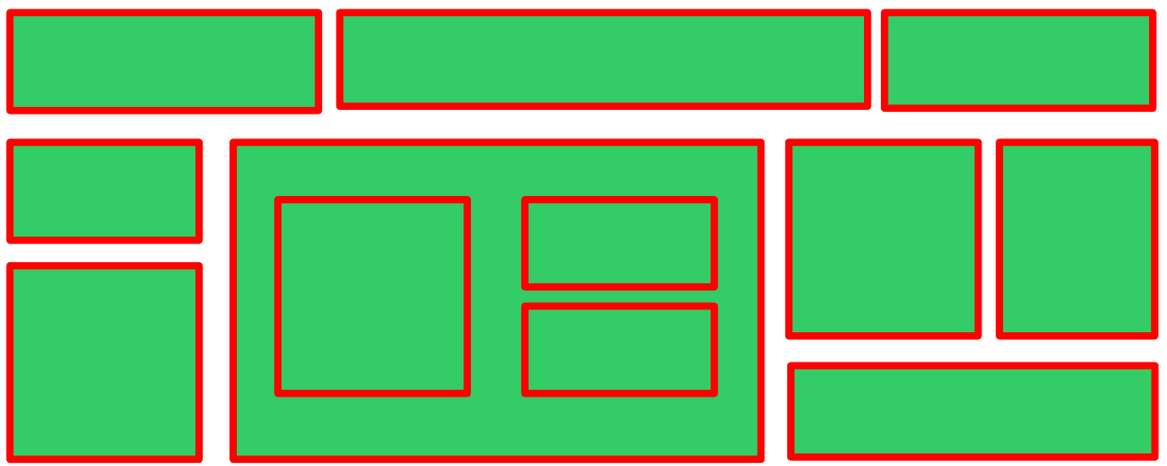
A Cartoon View of Digital Hardware Design



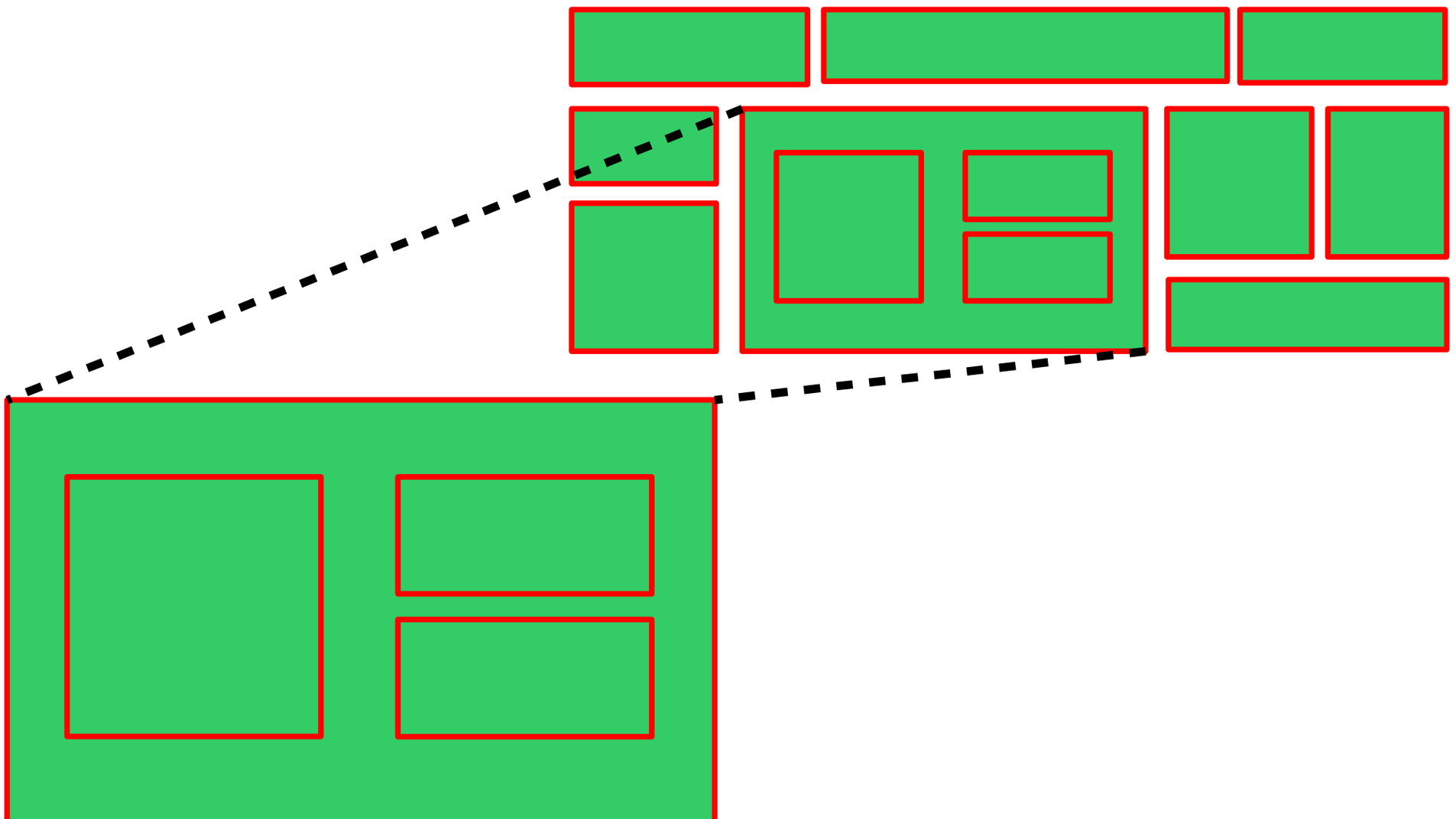
A Cartoon View of Digital Hardware Design



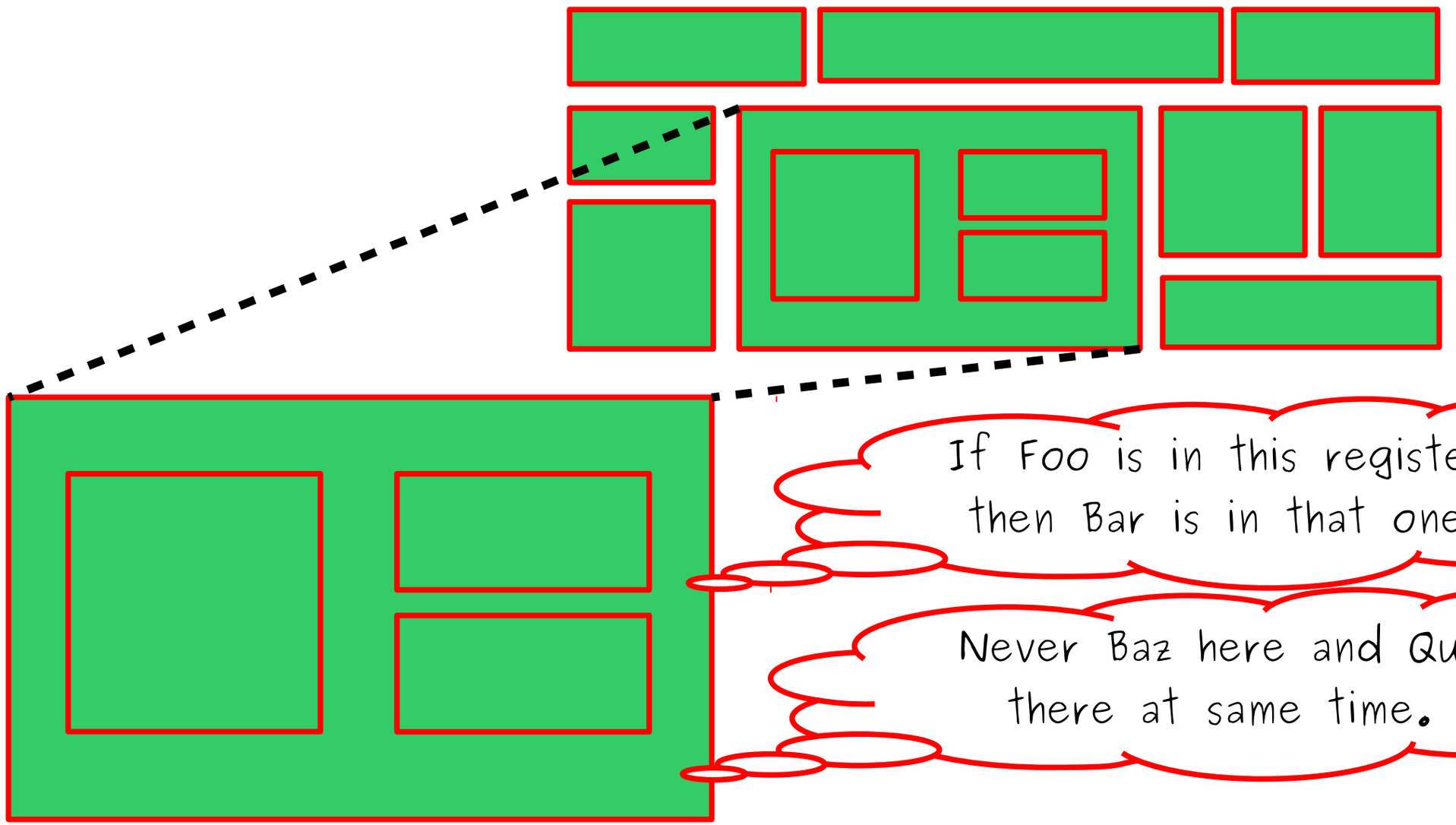
Simplification #1: Prove a Shallow Property



Simplification #1: Prove a Shallow Property



Simplification #1: Prove a Shallow Property



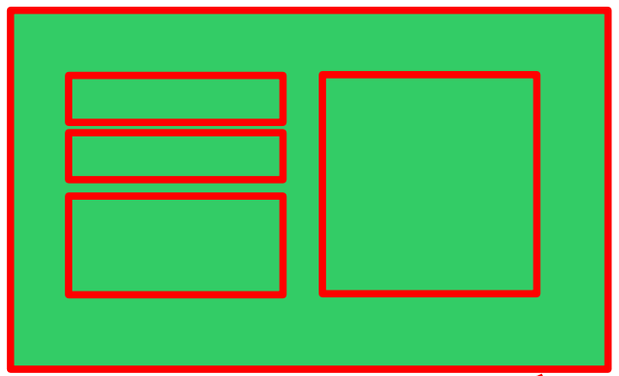
Common practice: prove some

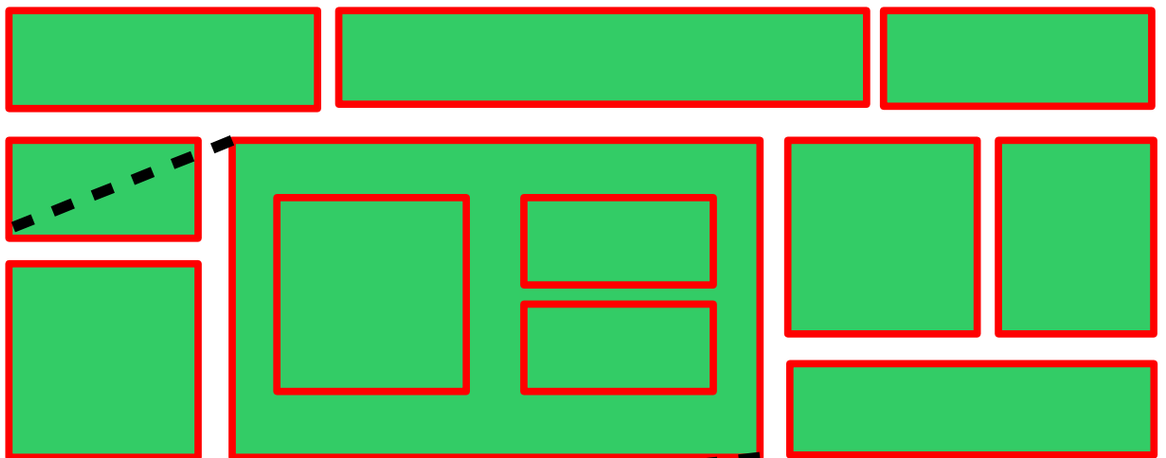
Invariants

If Foo is in this register, then Bar is in that one.

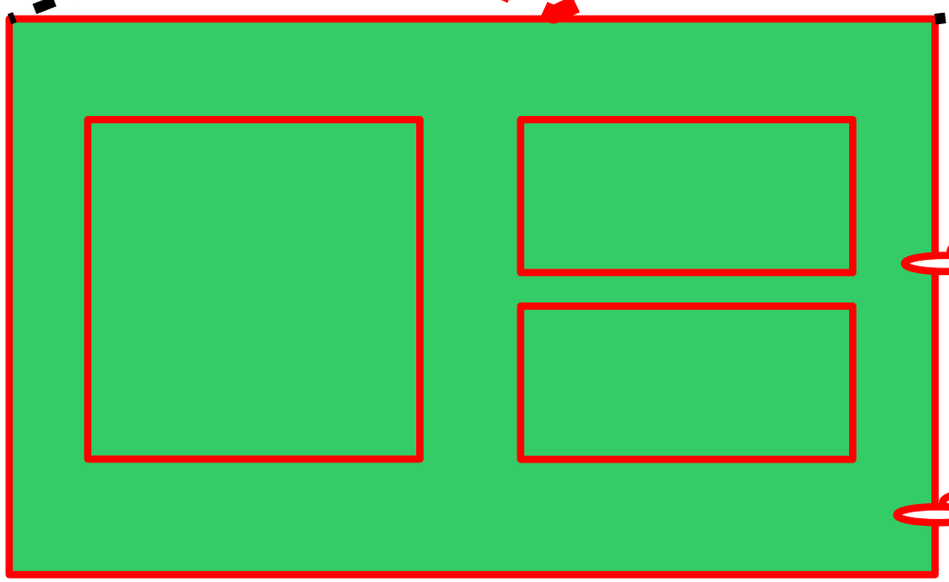
Never Baz here and Qux there at same time.

Simplification #1: Prove a Shallow Property





or Boolean equivalence check



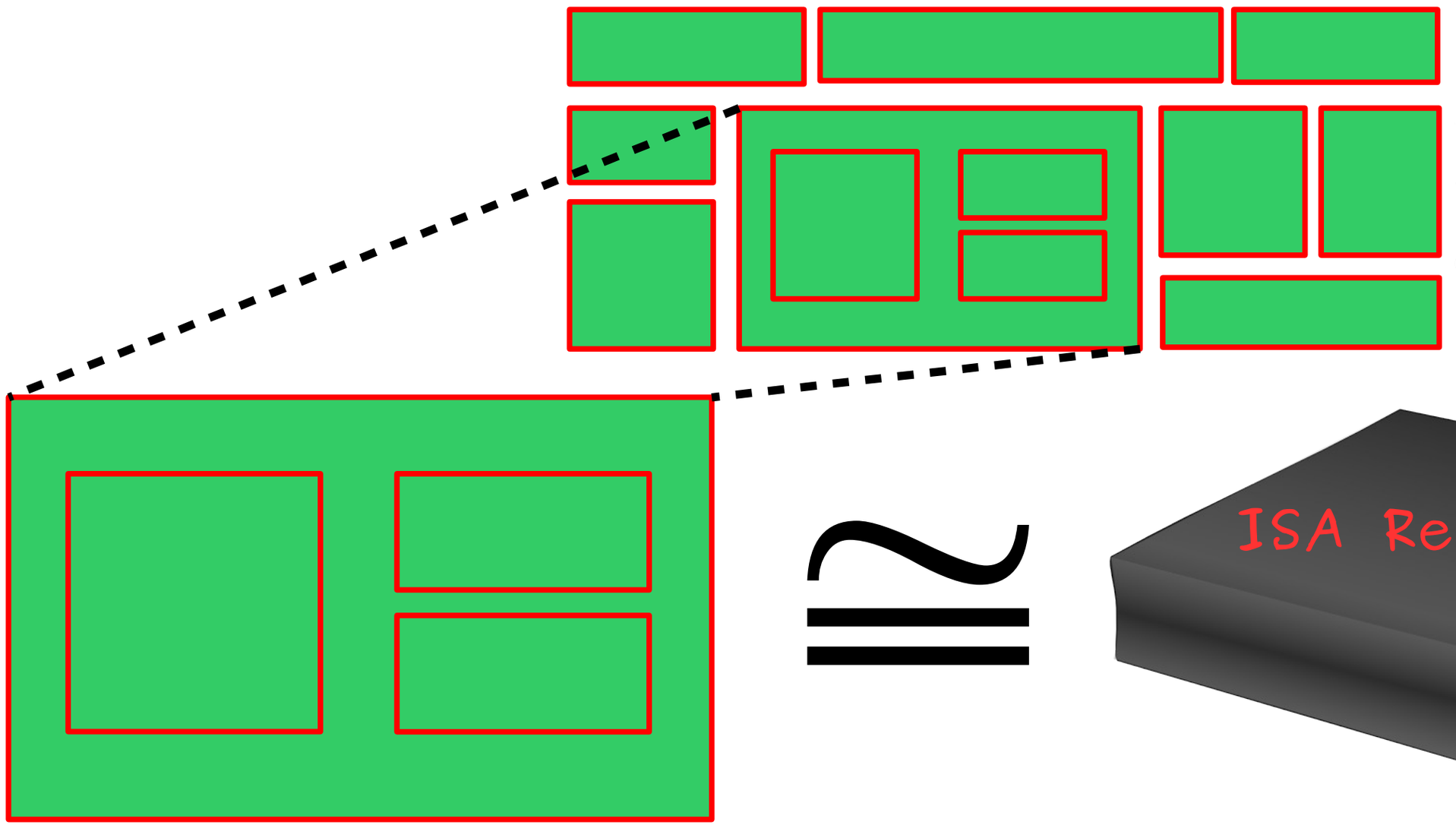
Common practice: prove some

Invariants

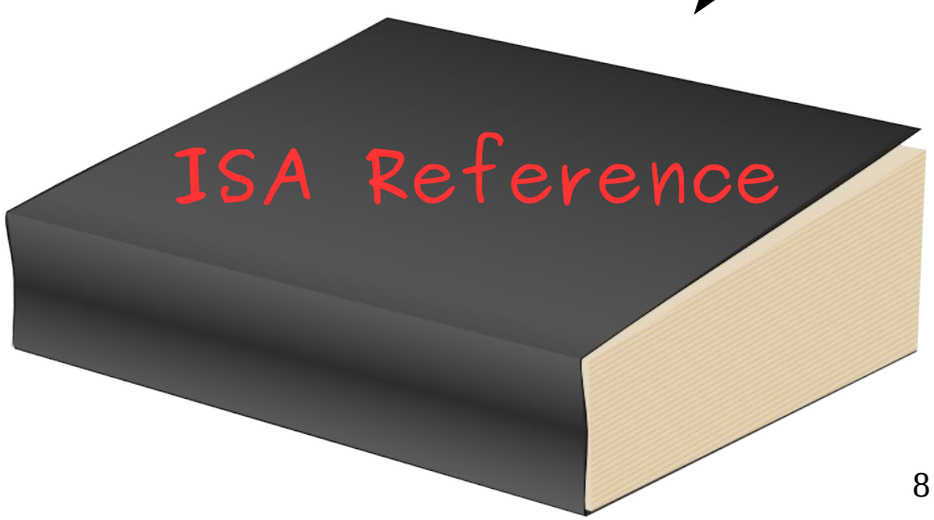
If Foo is in this register, then Bar is in that one.

Never Baz here and Qux there at same time.

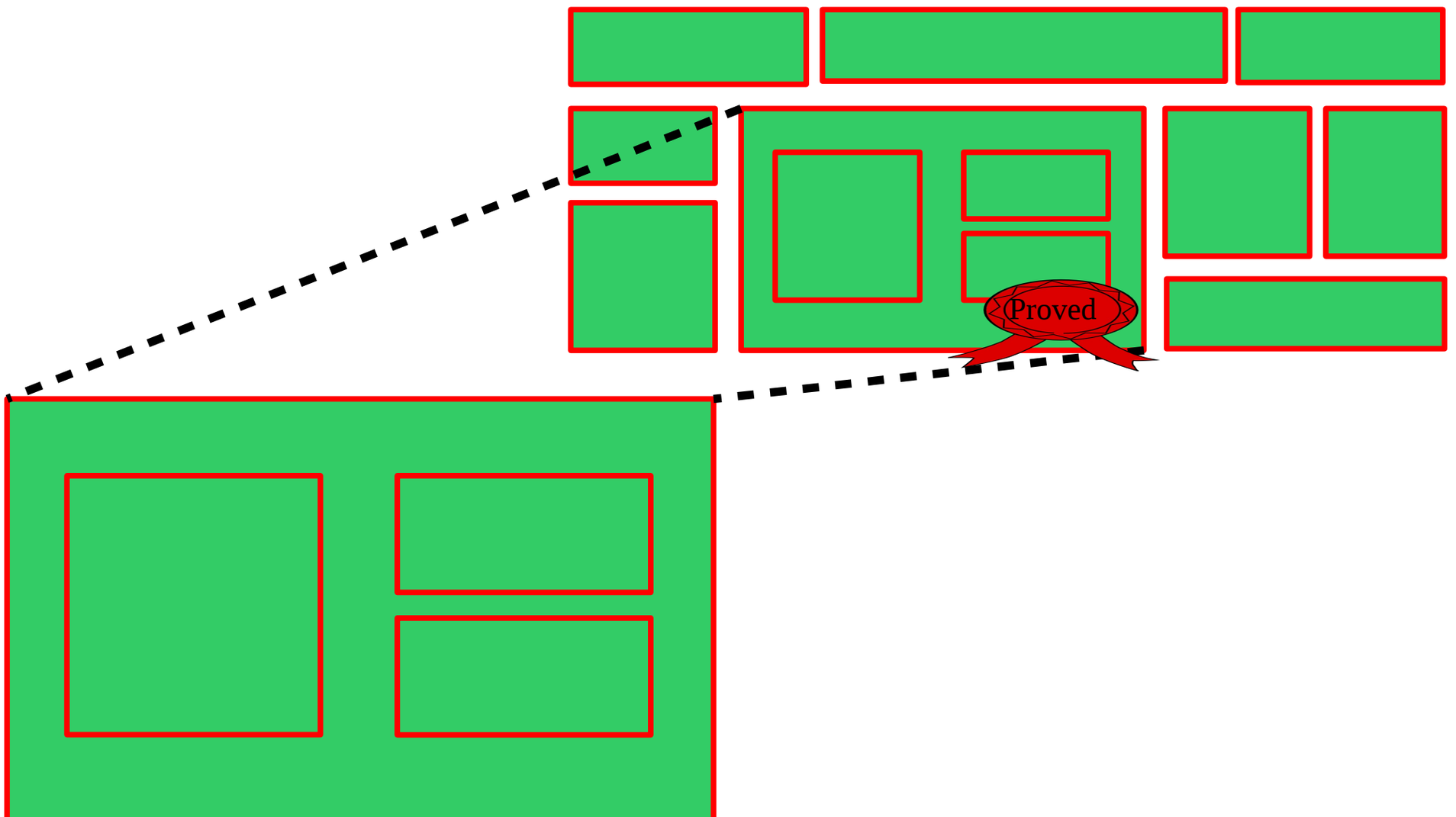
Simplification #1: Prove a Shallow Property



The Kami way:
Behavioral refinement
of functional spec

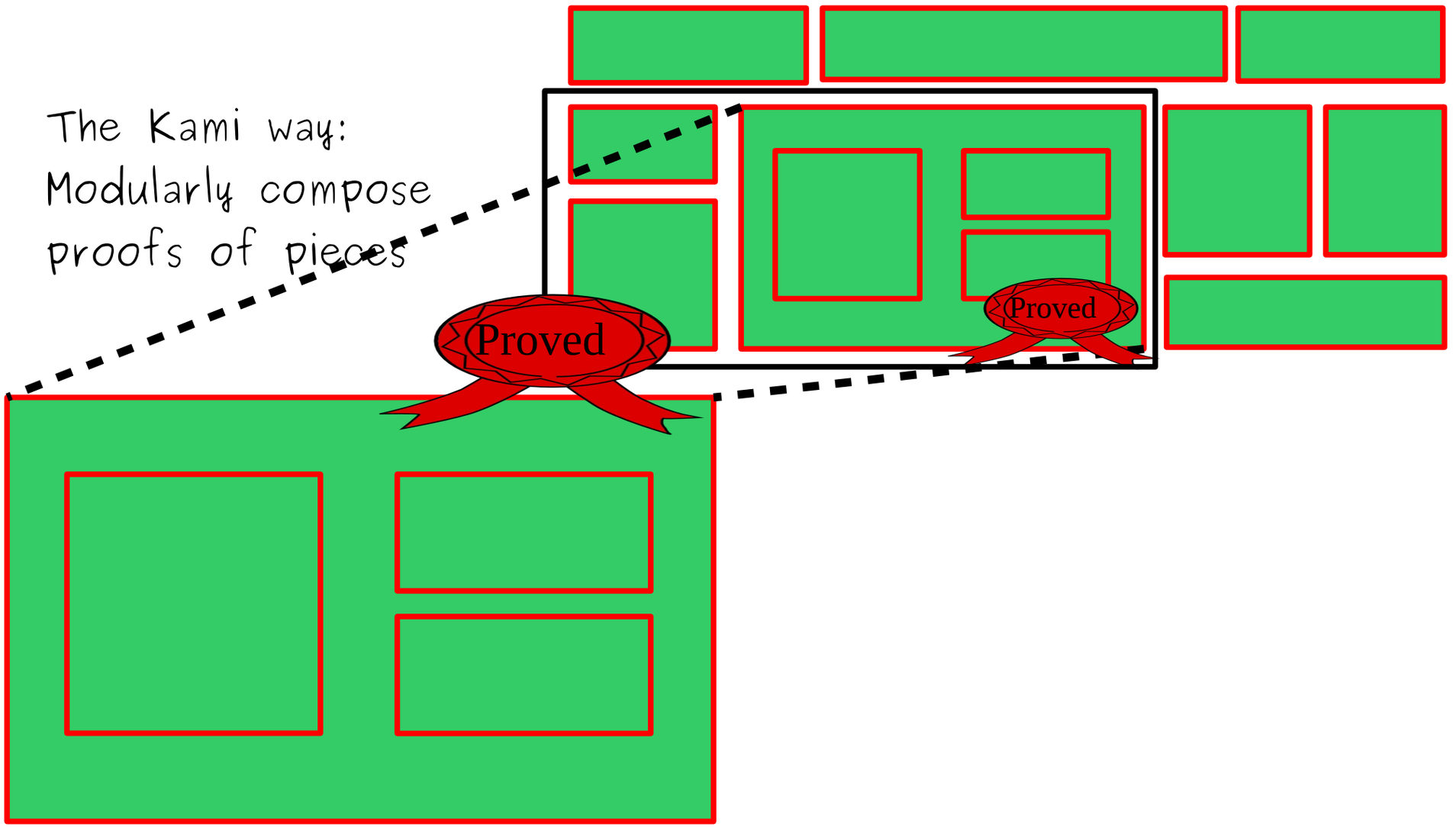


Simplification #2: Analyze Isolated Components



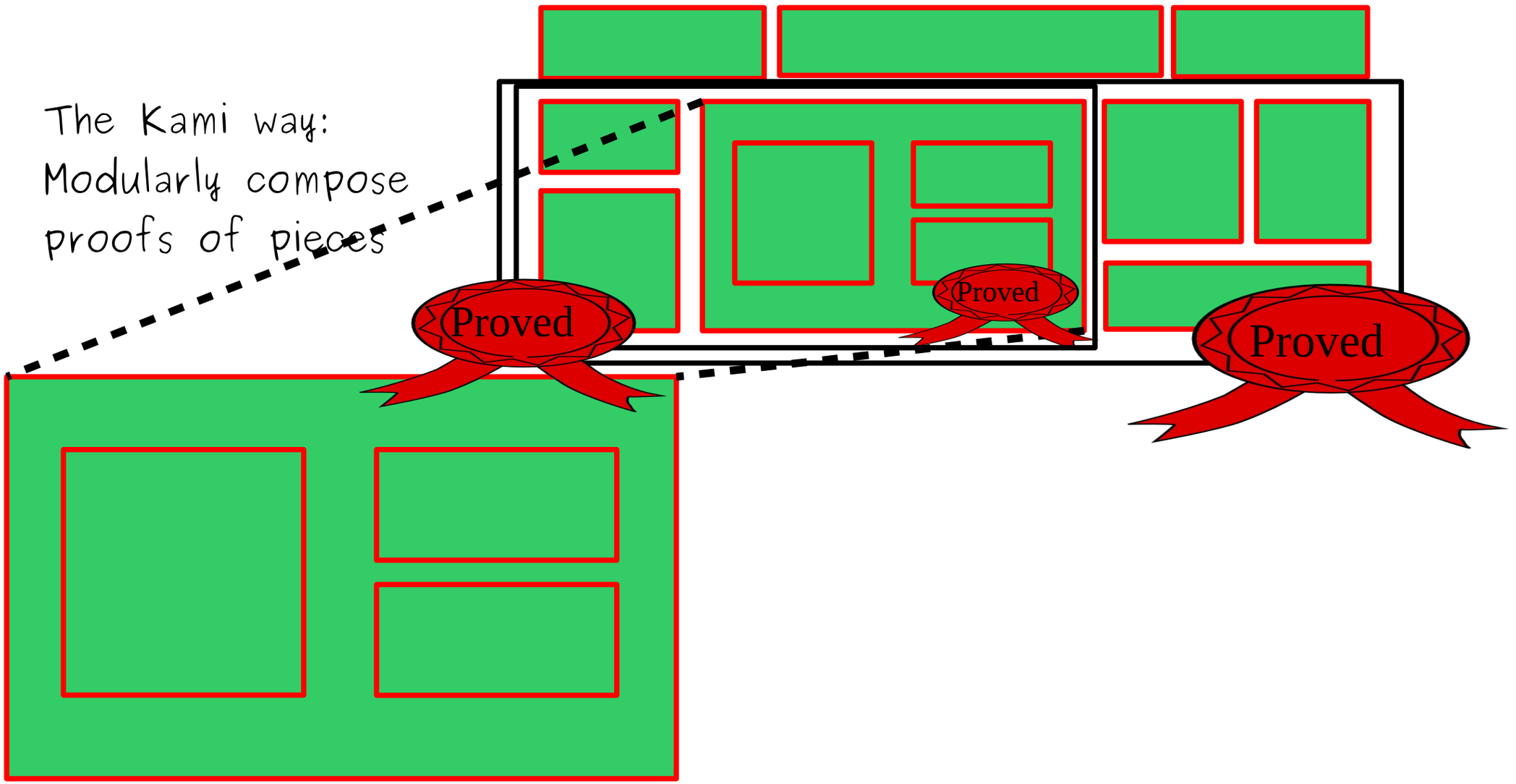
Simplification #2: Analyze Isolated Components

The Kami way:
Modularly compose
proofs of pieces

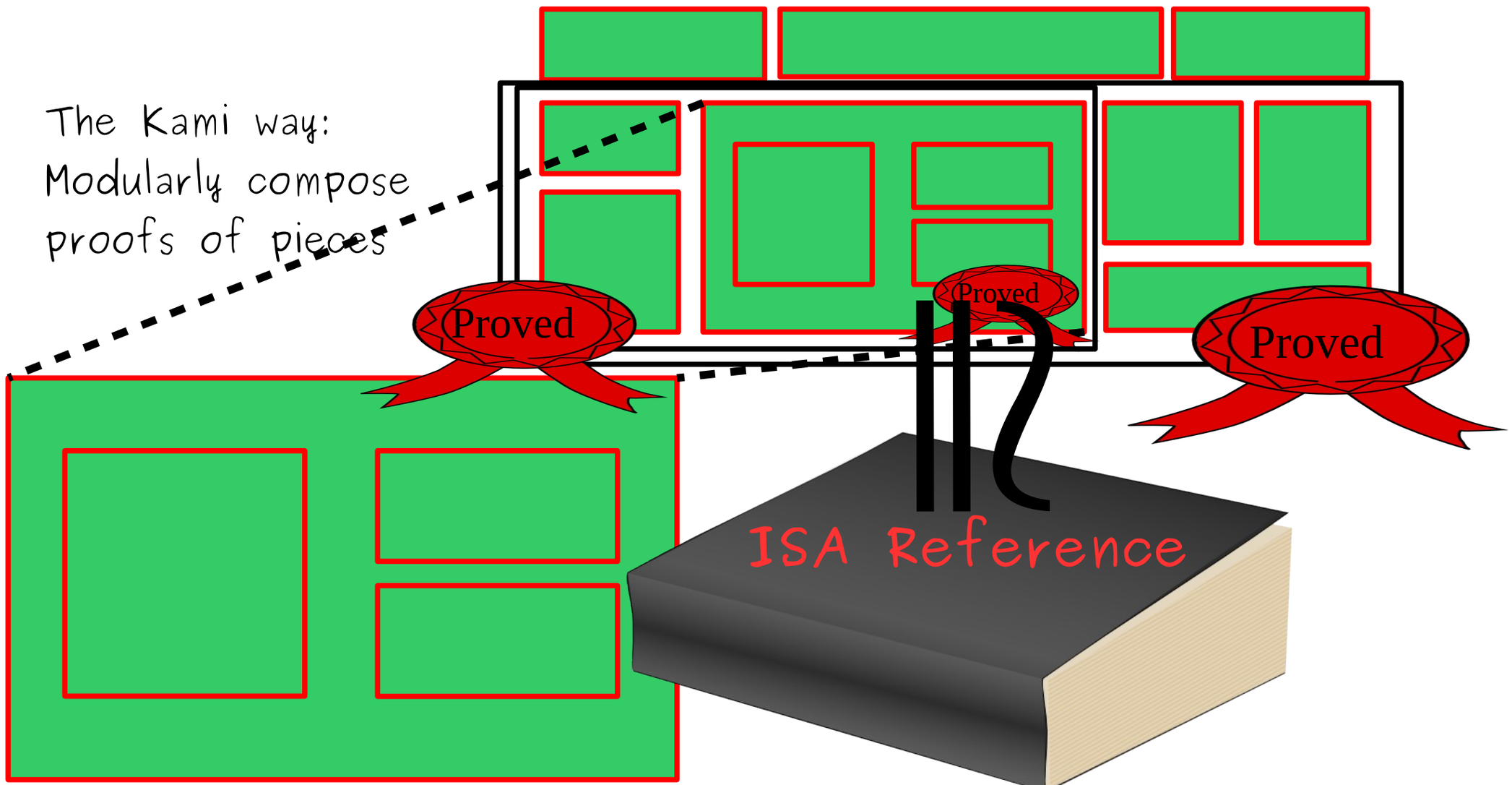


Simplification #2: Analyze Isolated Components

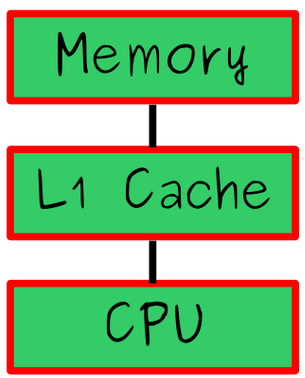
The Kami way:
Modularly compose
proofs of pieces



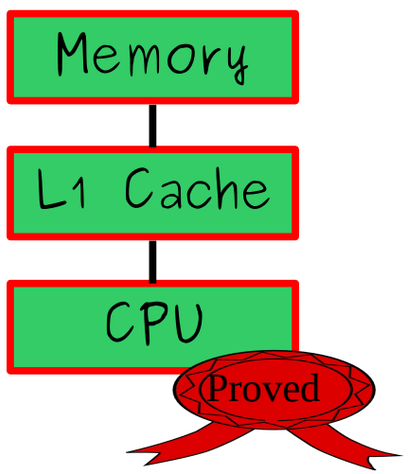
Simplification #2: Analyze Isolated Components

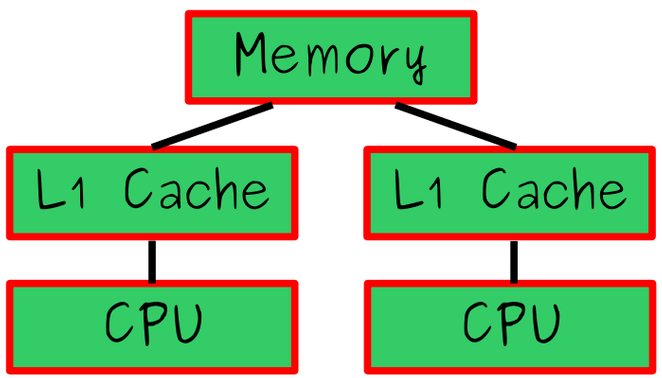


Simplification #3: Start Over For Each Design

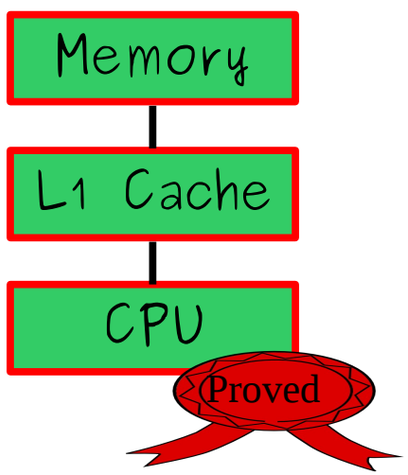


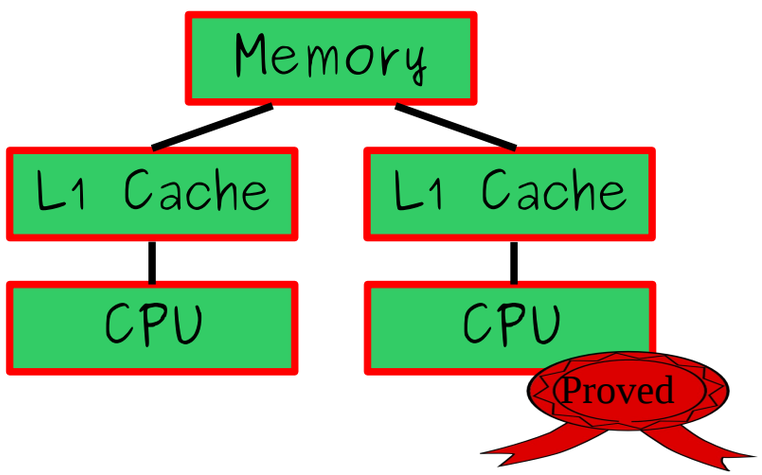
Simplification #3: Start Over For Each Design

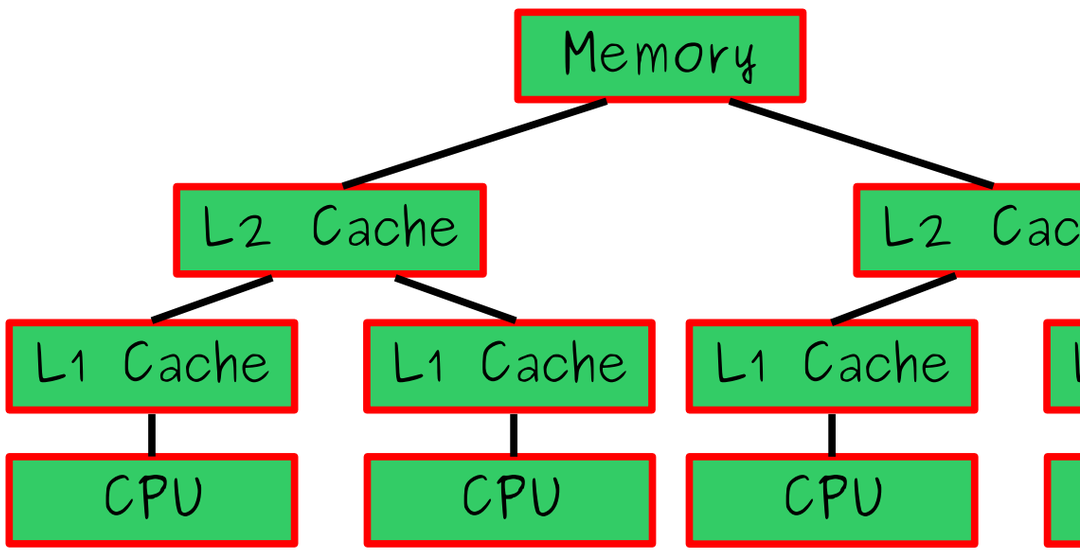




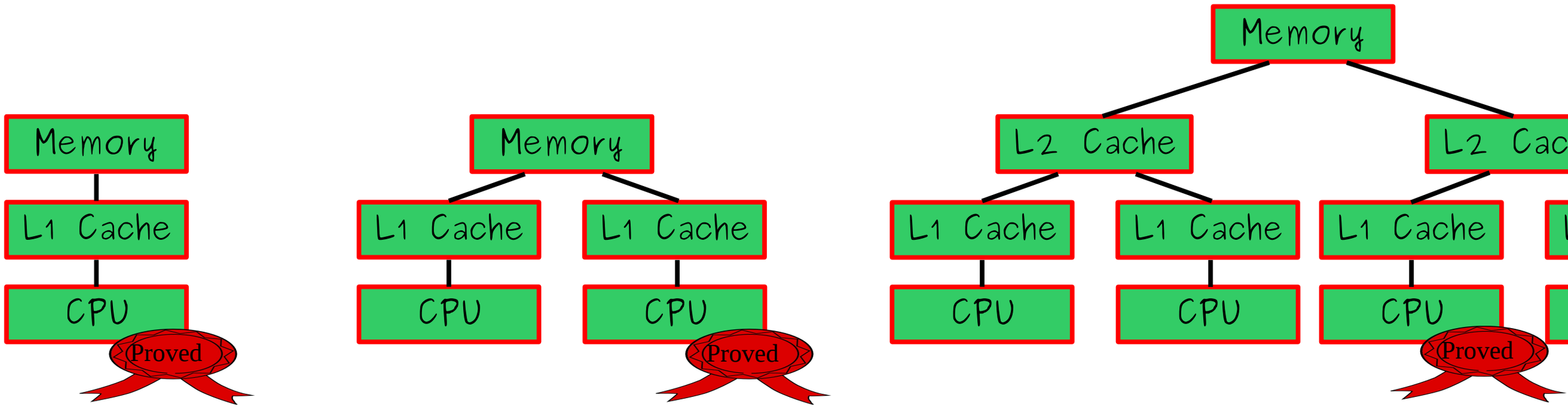
Simplification #3: Start Over For Each Design



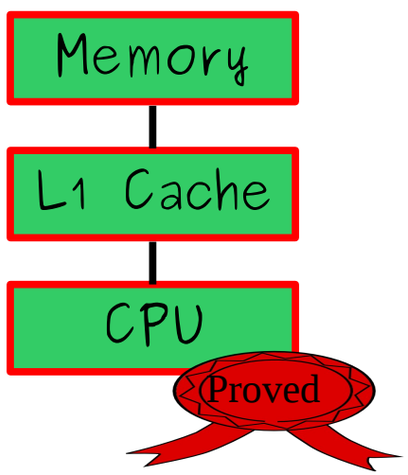


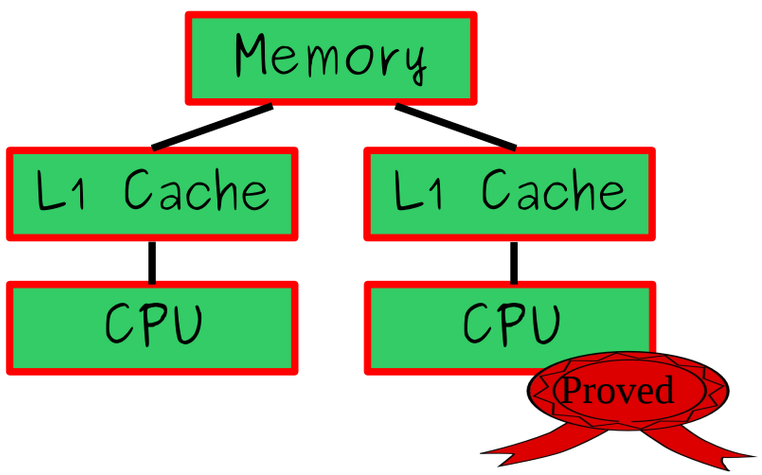


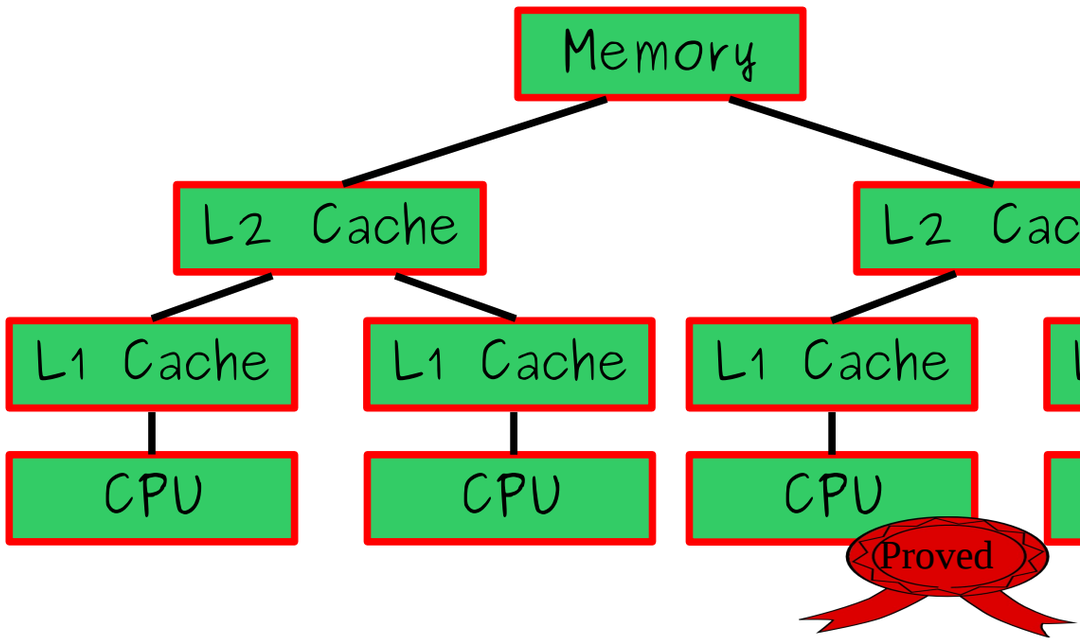
Simplification #3: Start Over For Each Design



Simplification #3: Start Over For Each Design

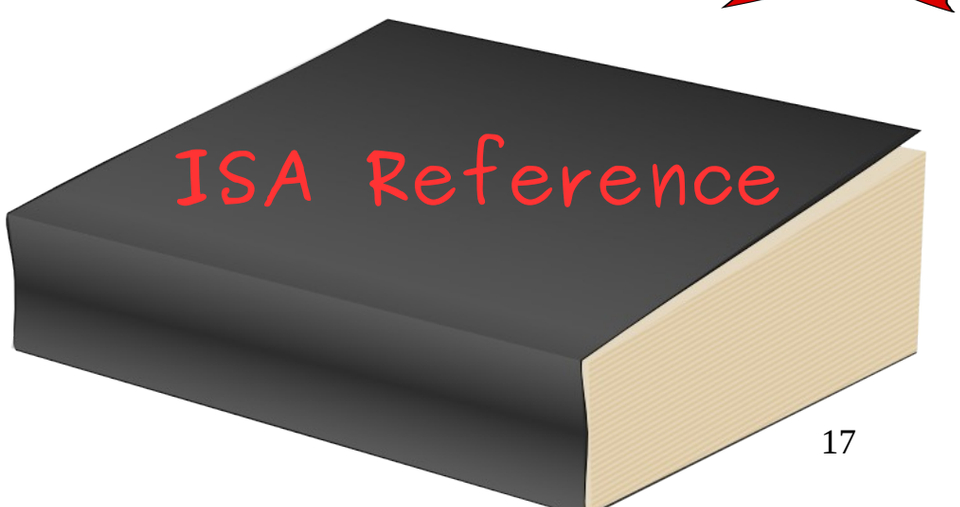






The Kami way:
Prove once for all parameters

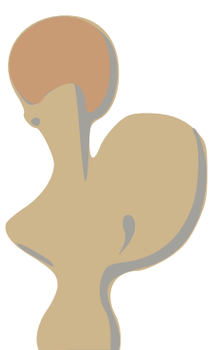
∀ trees. ≅



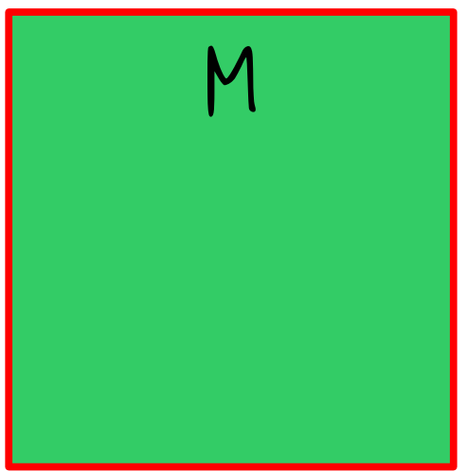
A **framework** to support *implementing, specifying, formally verifying,* and *compiling* hardware designs

based on the **Bluespec** high-level hardware design language

and the **Coq** proof assistant

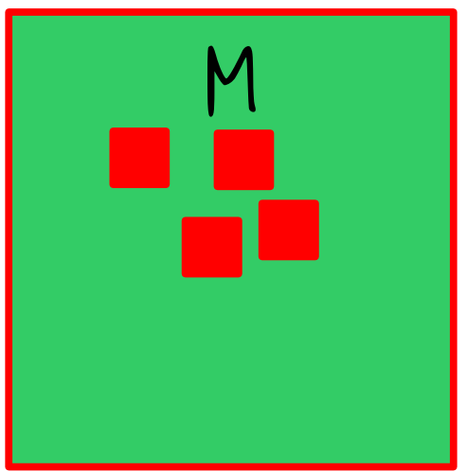


The Big Ideas (from Bluespec)



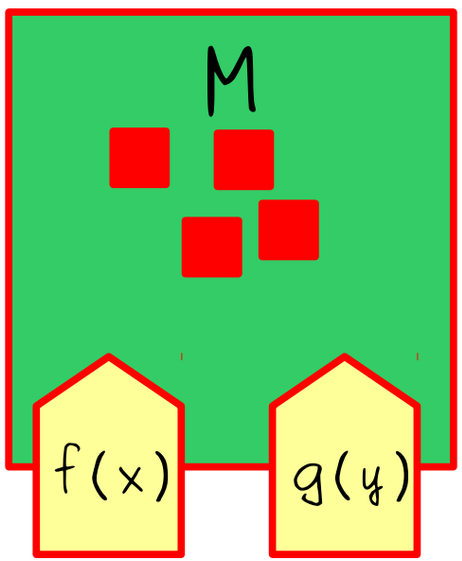
Program modules are objects

The Big Ideas (from Bluespec)



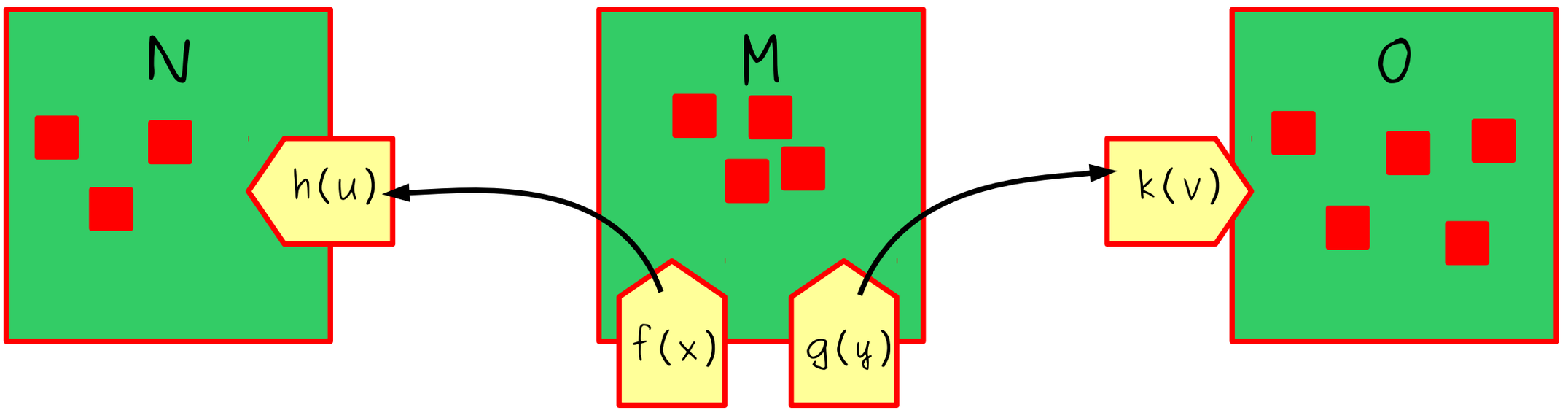
Program modules are objects
with mutable private state,

The Big Ideas (from Bluespec)



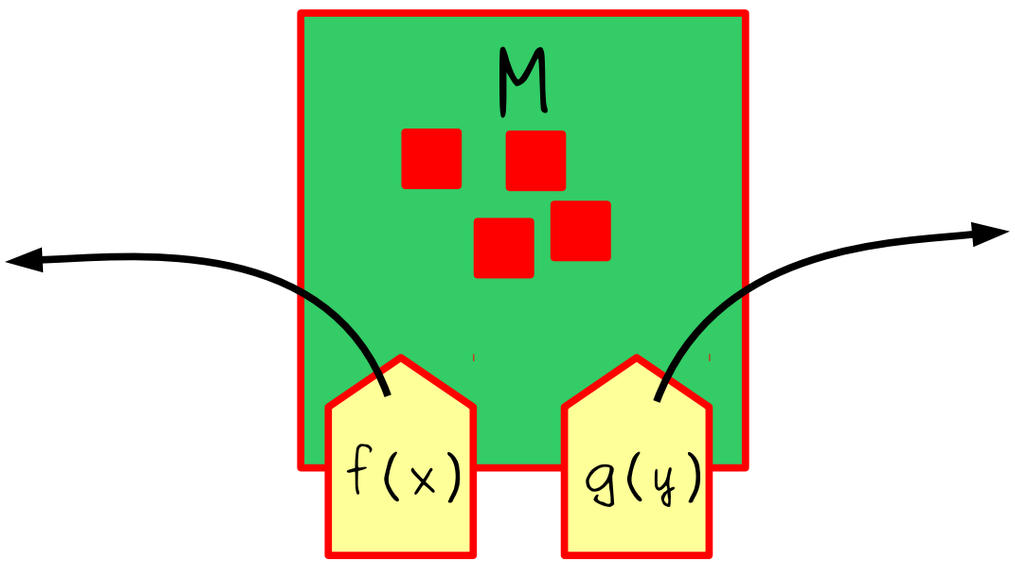
Program modules are objects
with mutable private state,
accessed via methods.

The Big Ideas (from Bluespec)



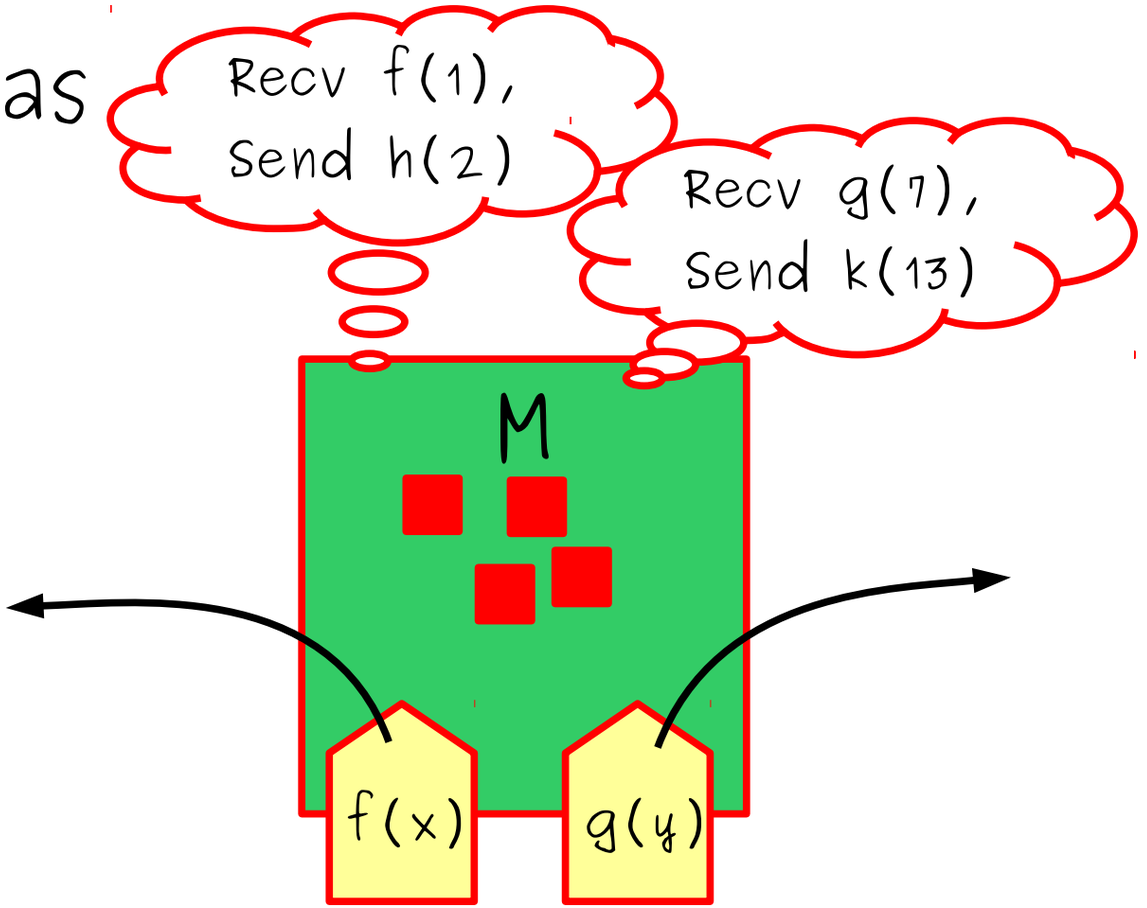
Program modules are objects
with mutable private state,
accessed via methods.

The Big Ideas



Every method call appears to execute **atomically.**

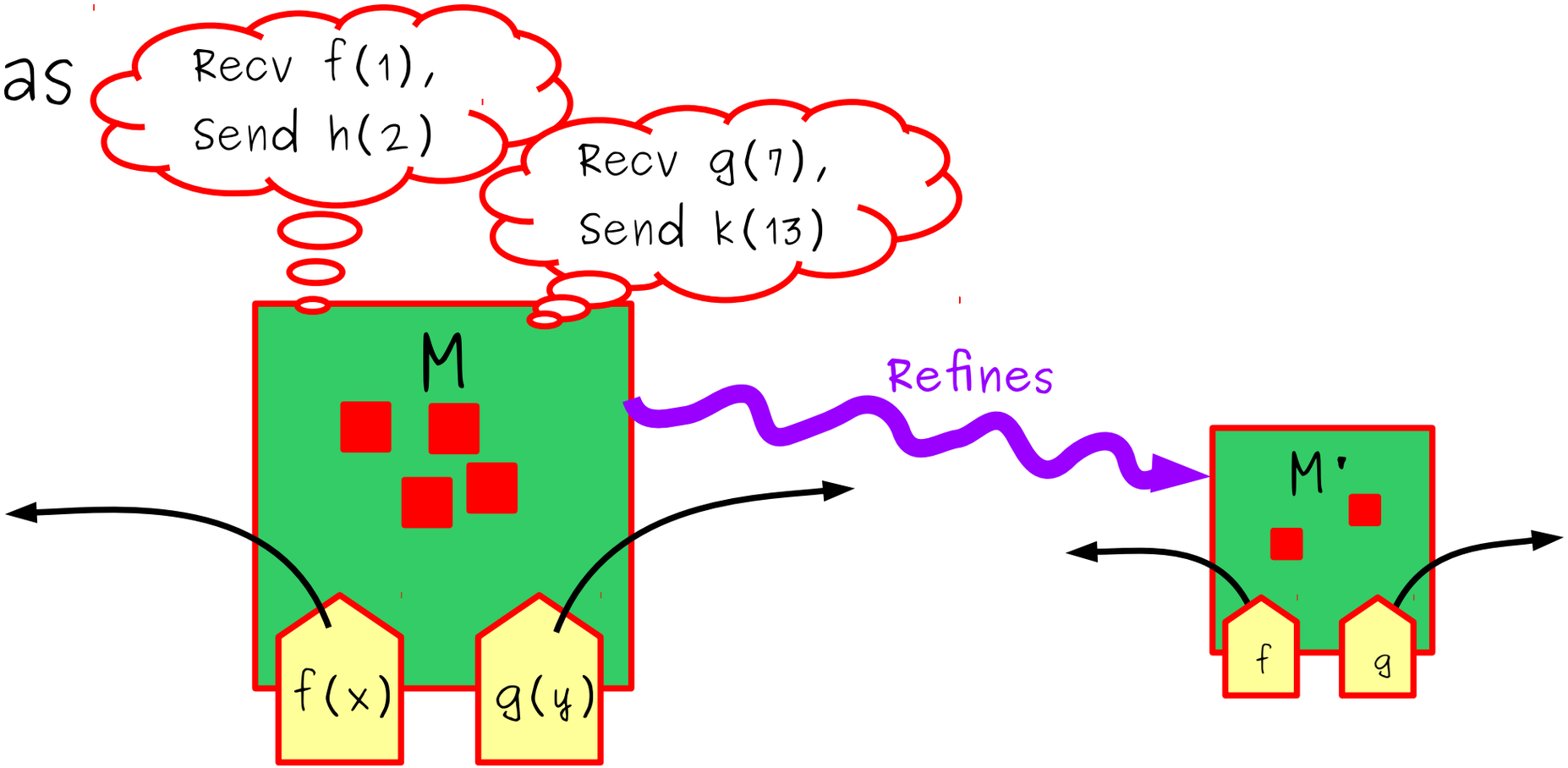
The Big Ideas



Every method call appears to execute **atomically.**

Any step is summarized by a *trace* of calls.

The Big Ideas

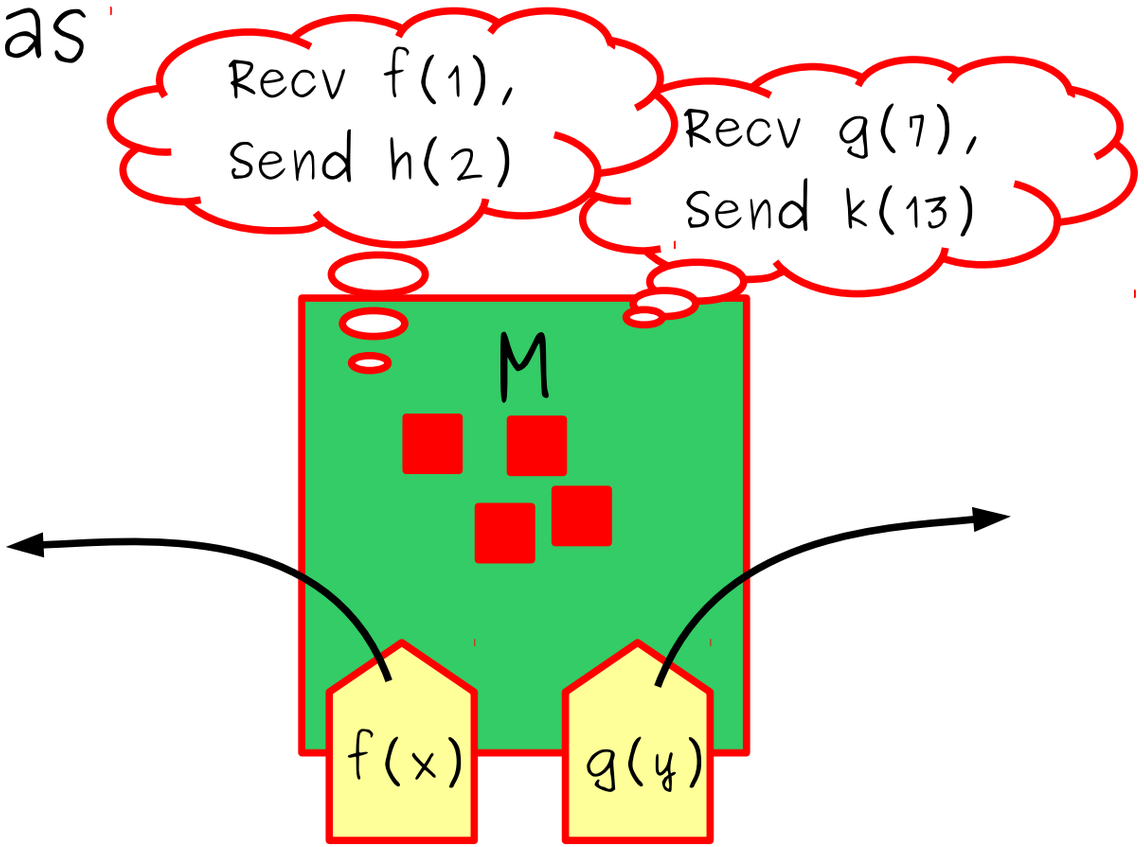


Every method call appears to execute **atomically.**

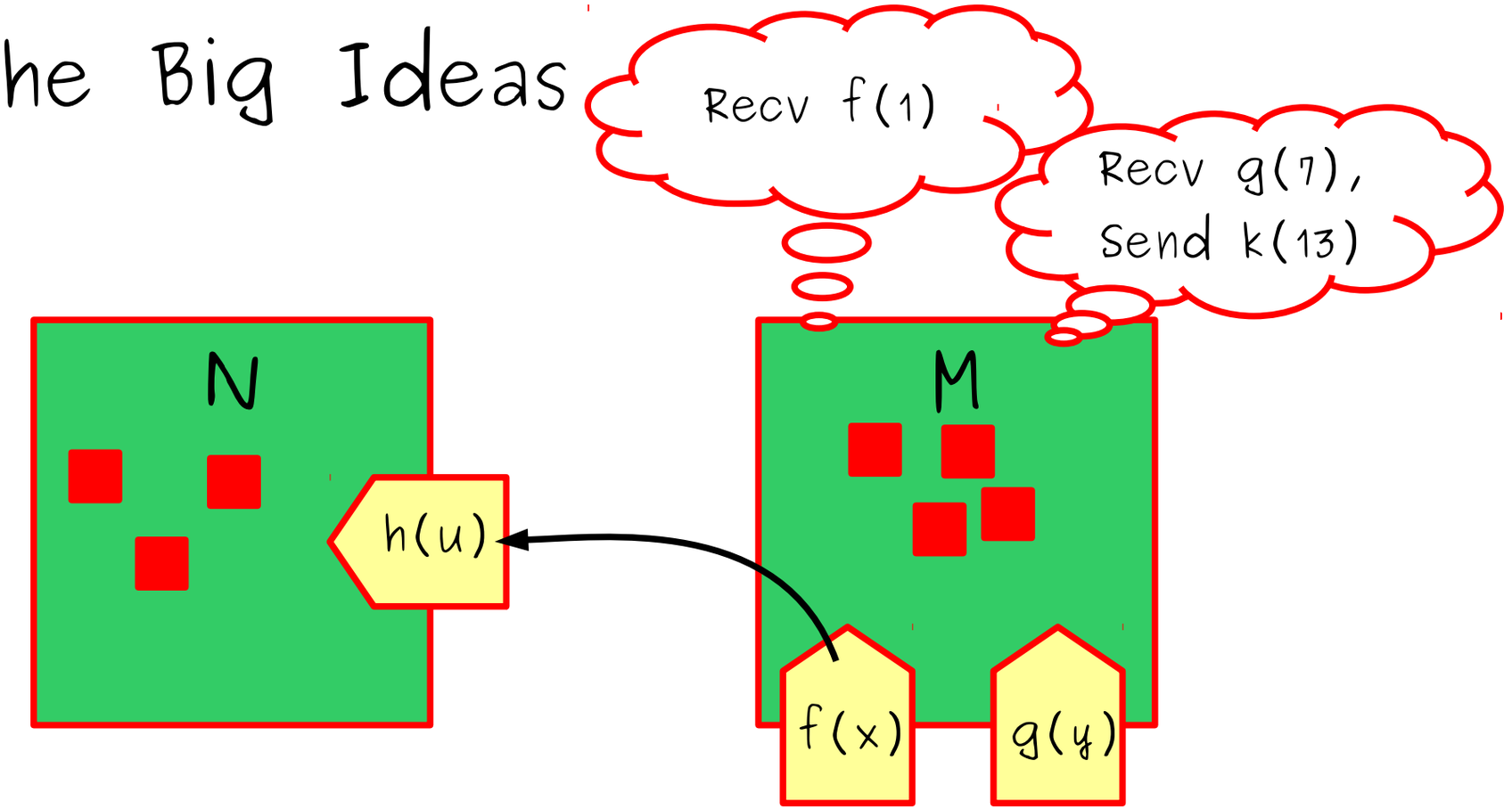
Any step is summarized by a *trace* of calls.

Object *refinement* is inclusion of possible traces.

The Big Ideas

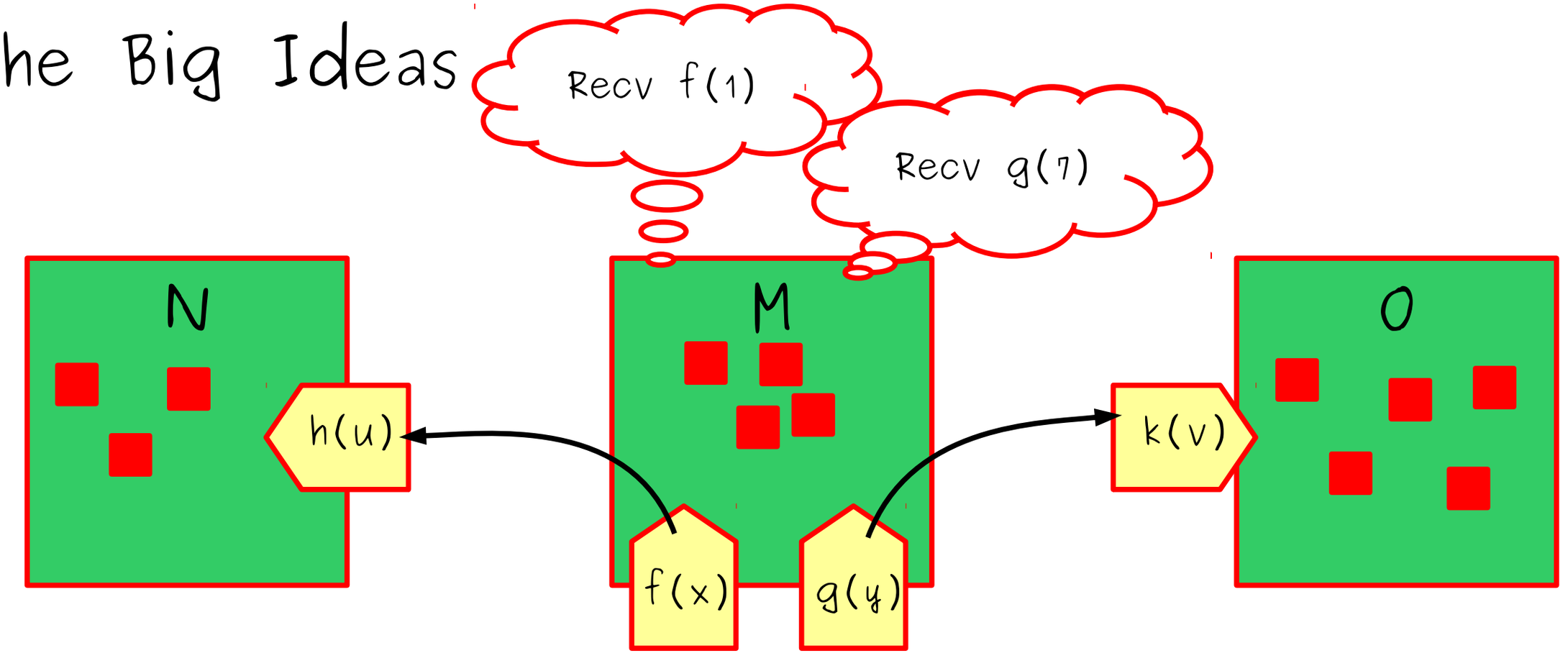


The Big Ideas



Composing objects hides internal method calls.

The Big Ideas



Composing objects hides internal method calls.

Some Example Kami Code (simple FIFO)

```
Definition deq {ty} : ActionT ty dType :=
  Read isEmpty <- ^empty;
  Assert !#isEmpty;
  Read eltT <- ^elt;
  Read enqPT <- ^enqP;
  Read deqPT <- ^deqP;
  Write ^full <- $$false;
  LET next_deqP <- (#deqPT + $1) :: Bit sz;
  Write ^empty <- (#enqPT == #next_deqP);
  Write ^deqP <- #next_deqP;
  Ret #eltT@[#deqPT].
```

An Example Kami Proof (pipelined processor)

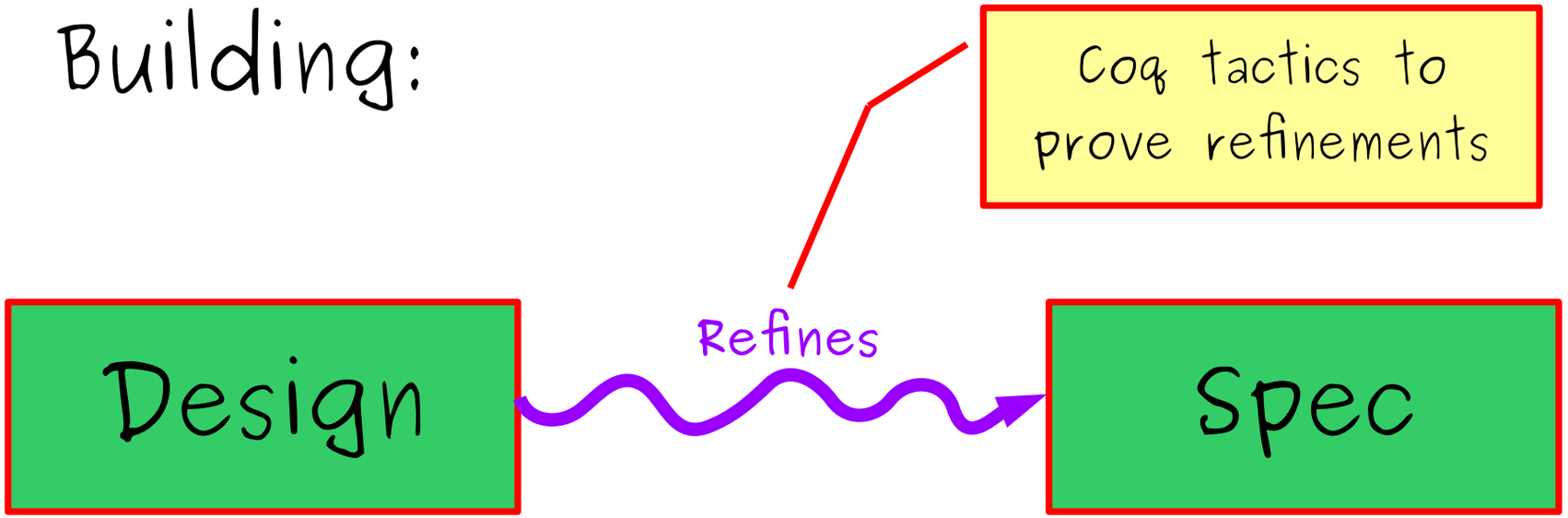
```
Lemma p4st_refines_p3st: p4st <<== p3st.
Proof.
  kmodular.
  - kdisj_edms_cms_ex 0.
  - kdisj_ecms_dms_ex 0.
  - apply fetchDecode_refines_fetchNDecode; auto.
  - krefl.
Qed.
```

Uses standard **Coq** ASCII syntax for mathematical proofs.

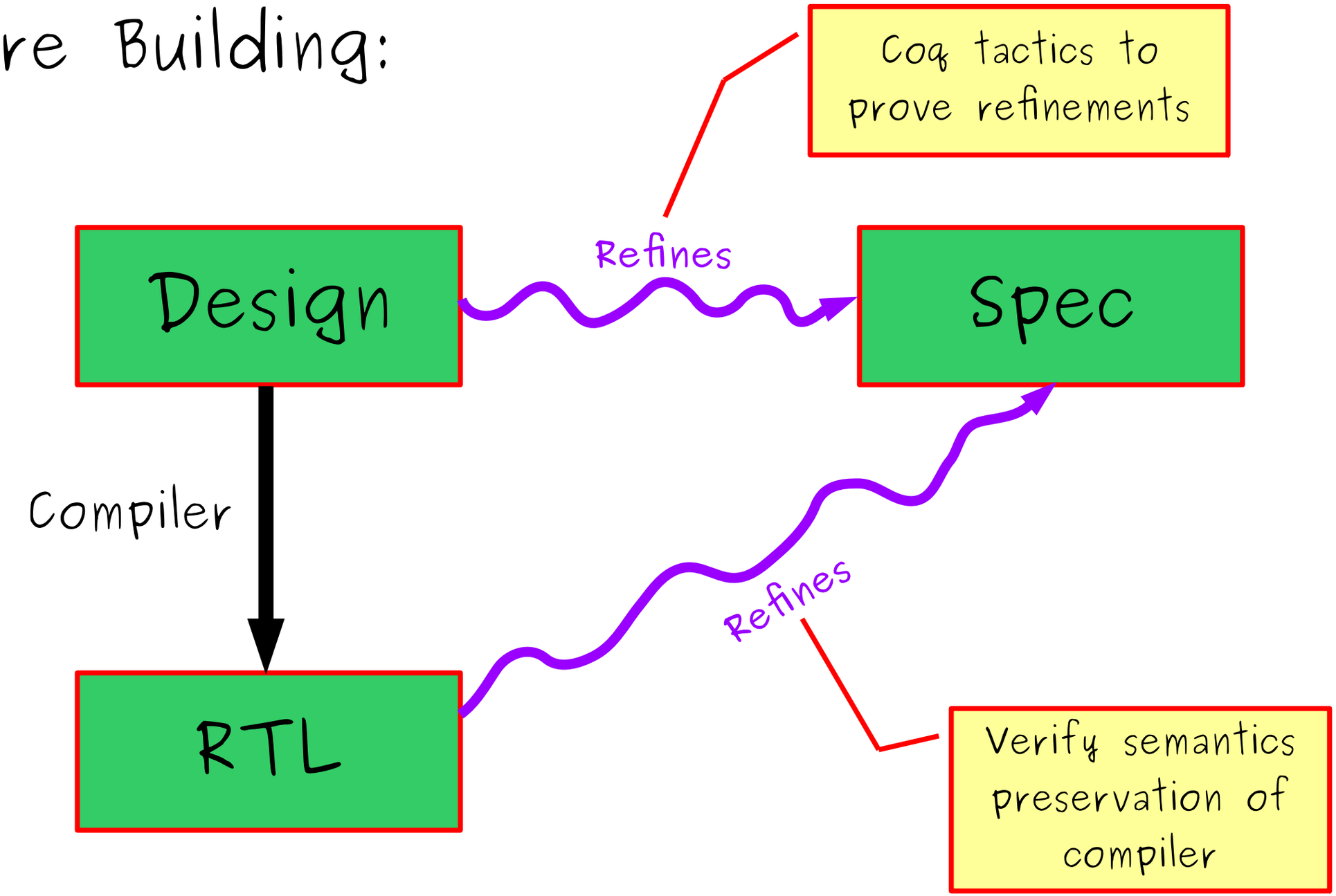
These proofs are checked **automatically**, just like type checking.

We inherit streamlined **IDE support** for Coq.

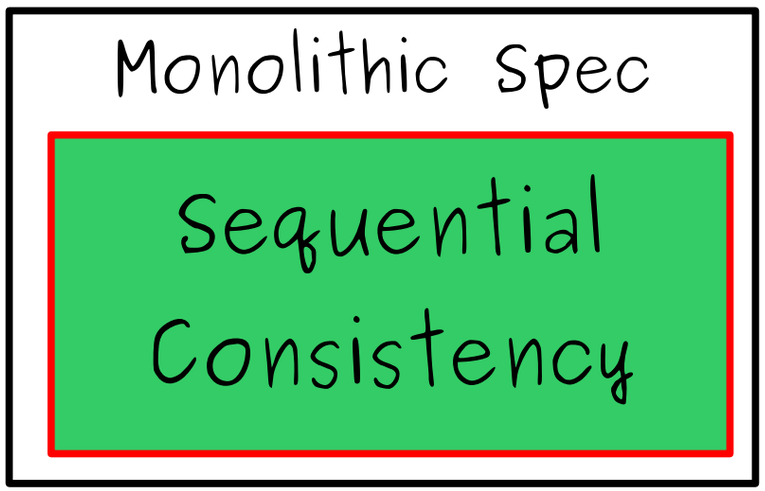
We Are Building:



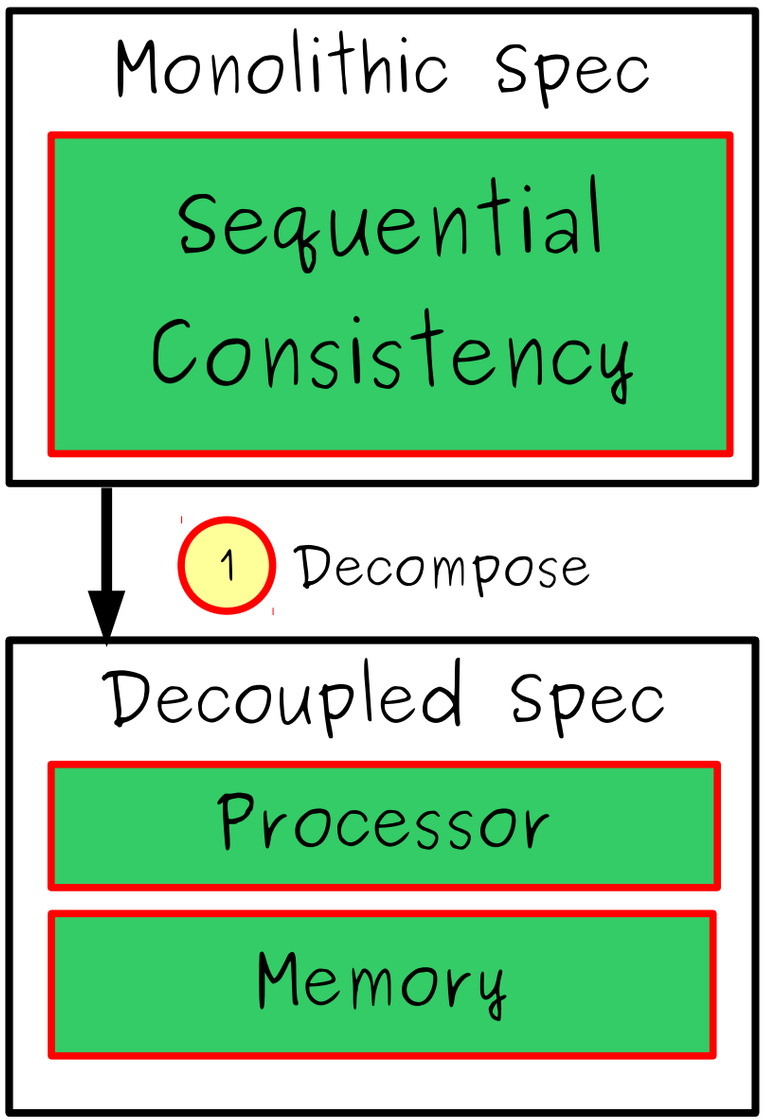
We Are Building:



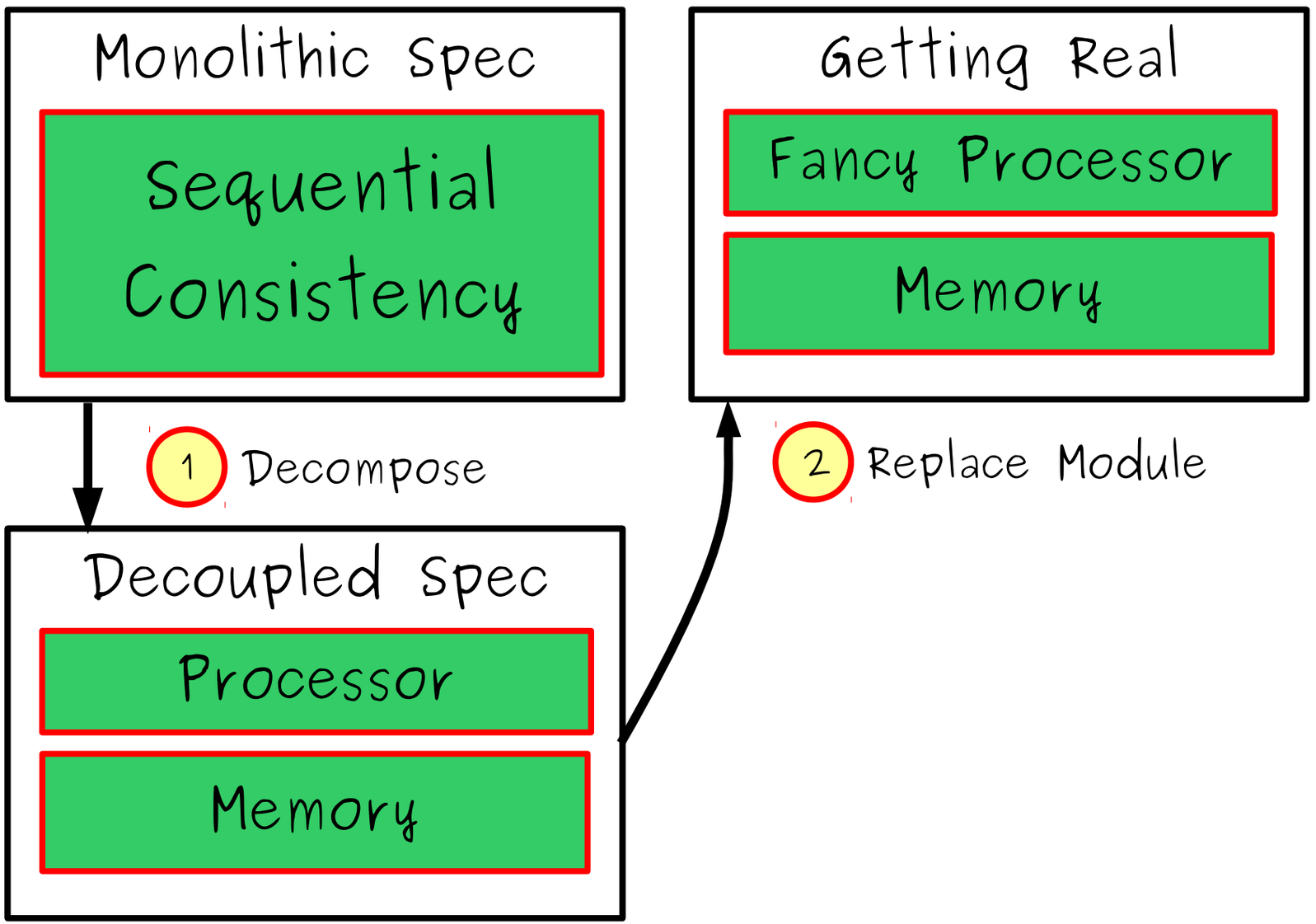
Some Useful Refinement Tactics



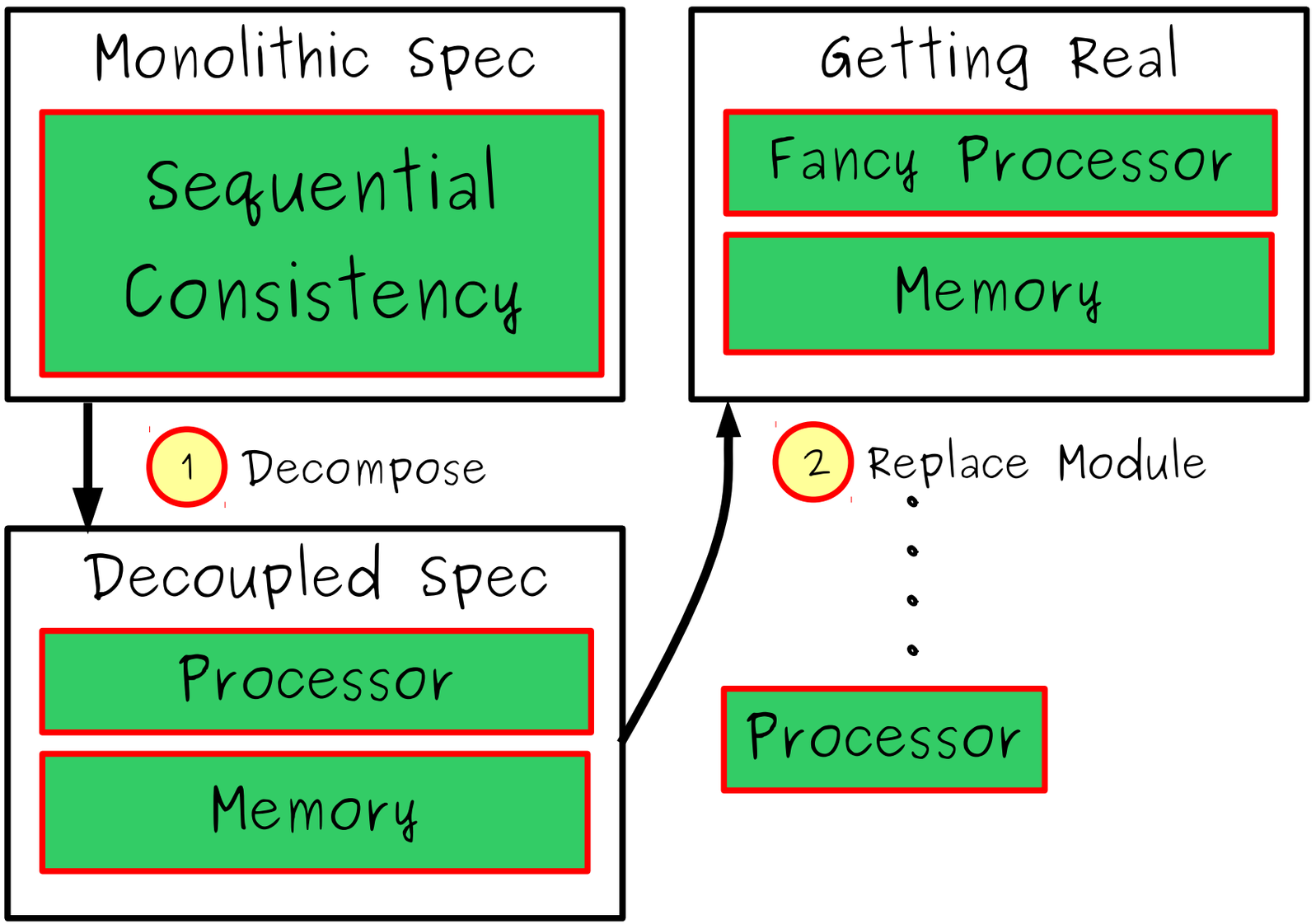
Some Useful Refinement Tactics



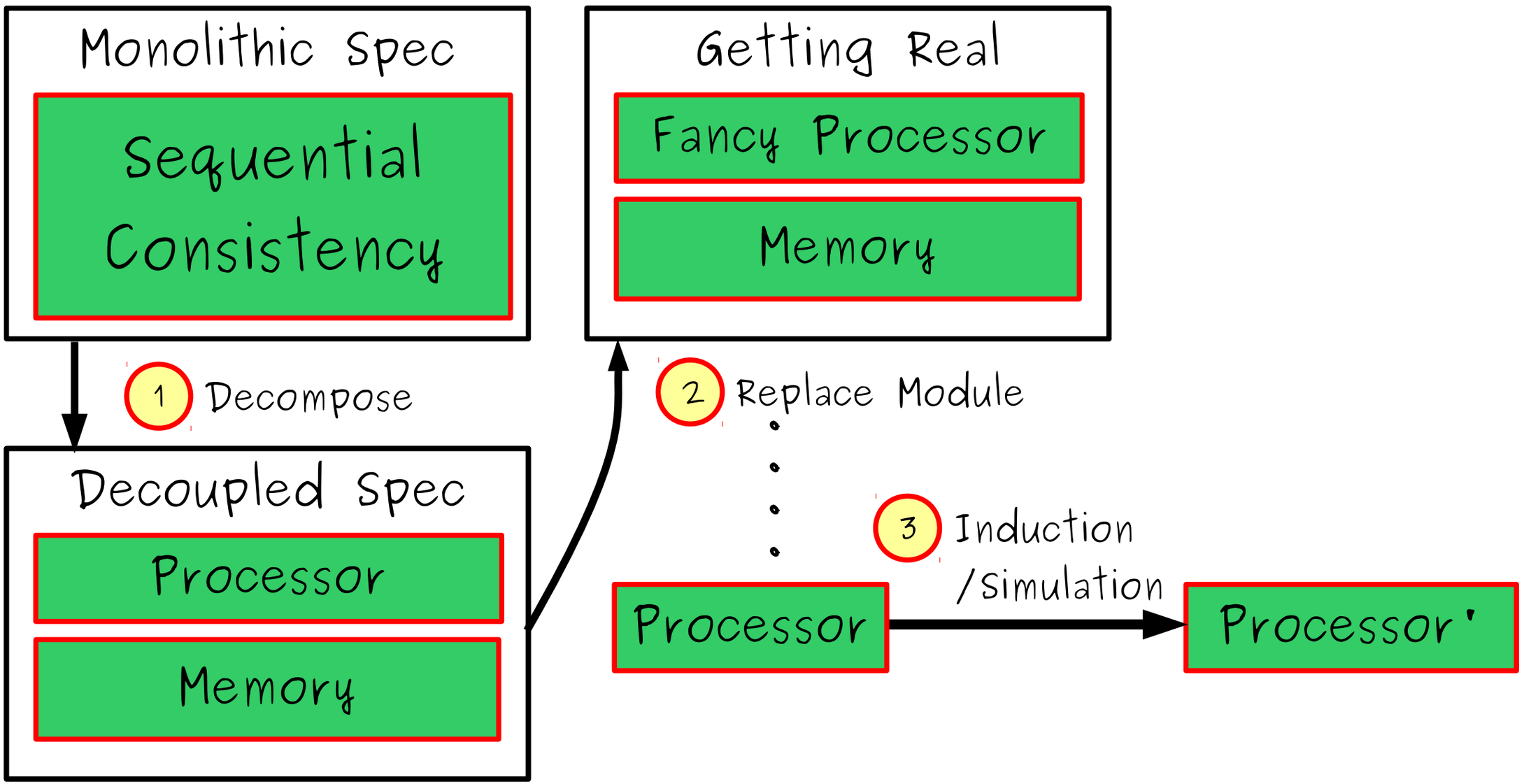
Some Useful Refinement Tactics



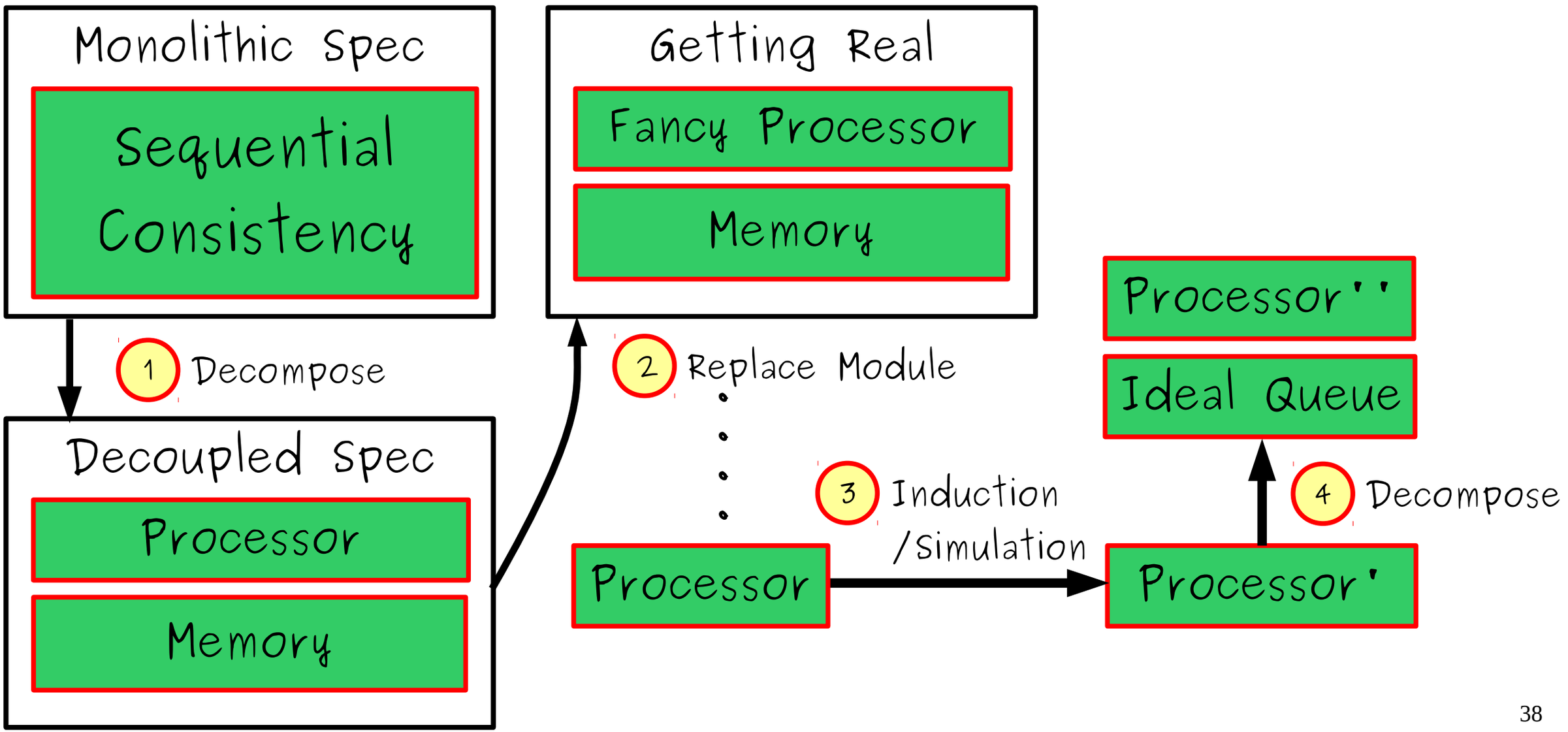
Some Useful Refinement Tactics



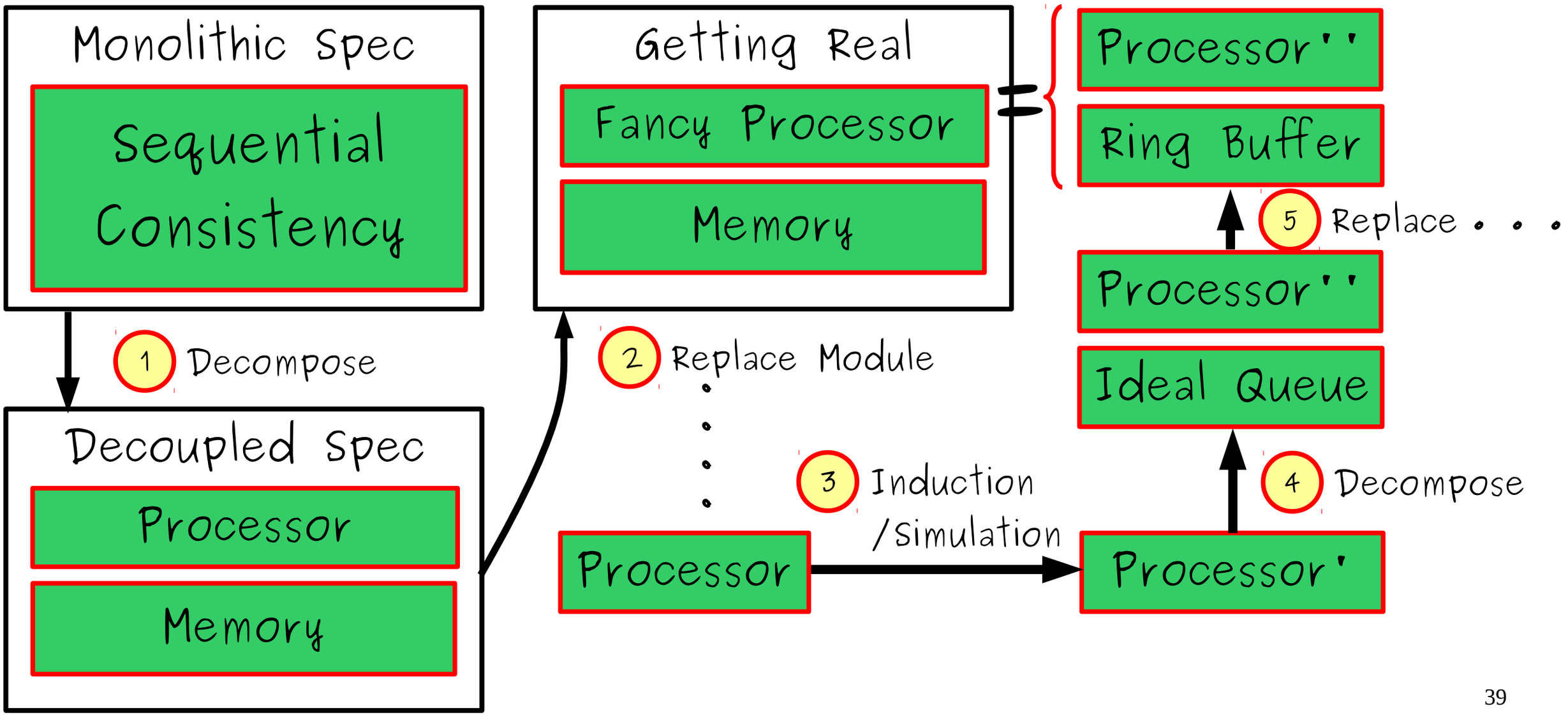
Some Useful Refinement Tactics



Some Useful Refinement Tactics



Some Useful Refinement Tactics



Key Ingredient

Formal Semantics for RISC-V ISA(s)

Nikhil just explained the semantics style.
We are building a translator for the semantics into the language of Coq/Kami.

An Open Library of Formally Verified Components

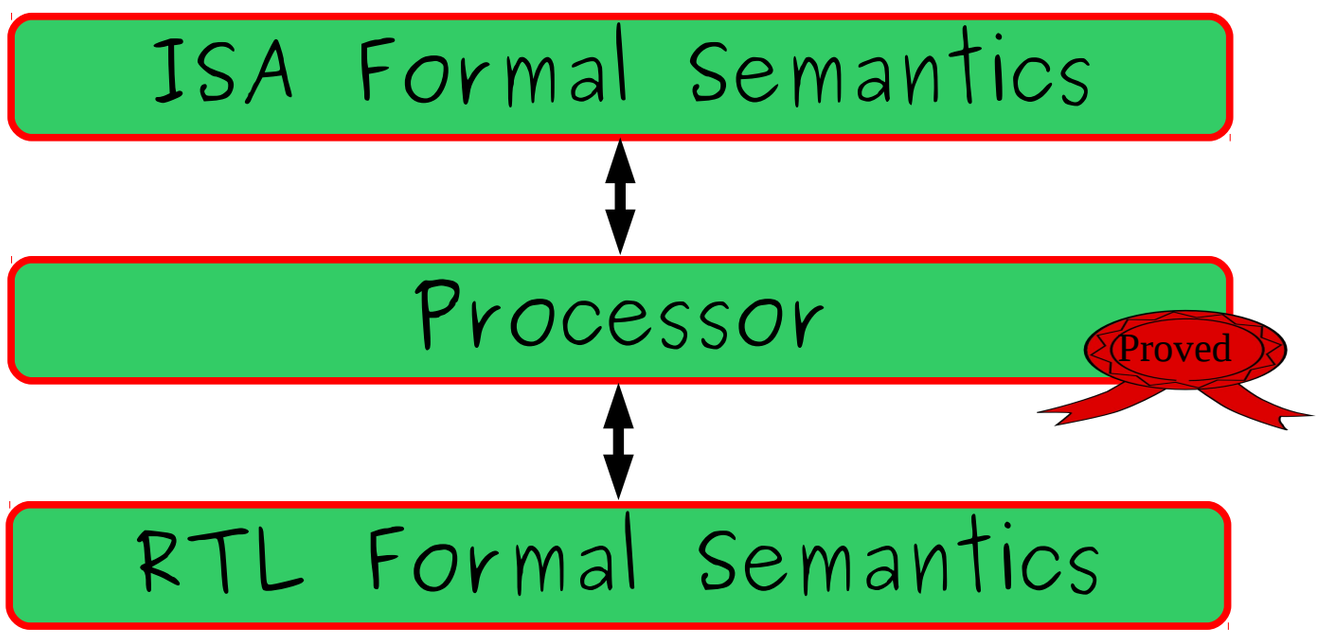
- Microcontroller-class RV32I (multicore; U)
- Desktop-class RV64IMA (multicore; U,S,M)
- Cache-coherent memory system

Reuse our proofs when composing our components with your own formally verified **accelerators**!

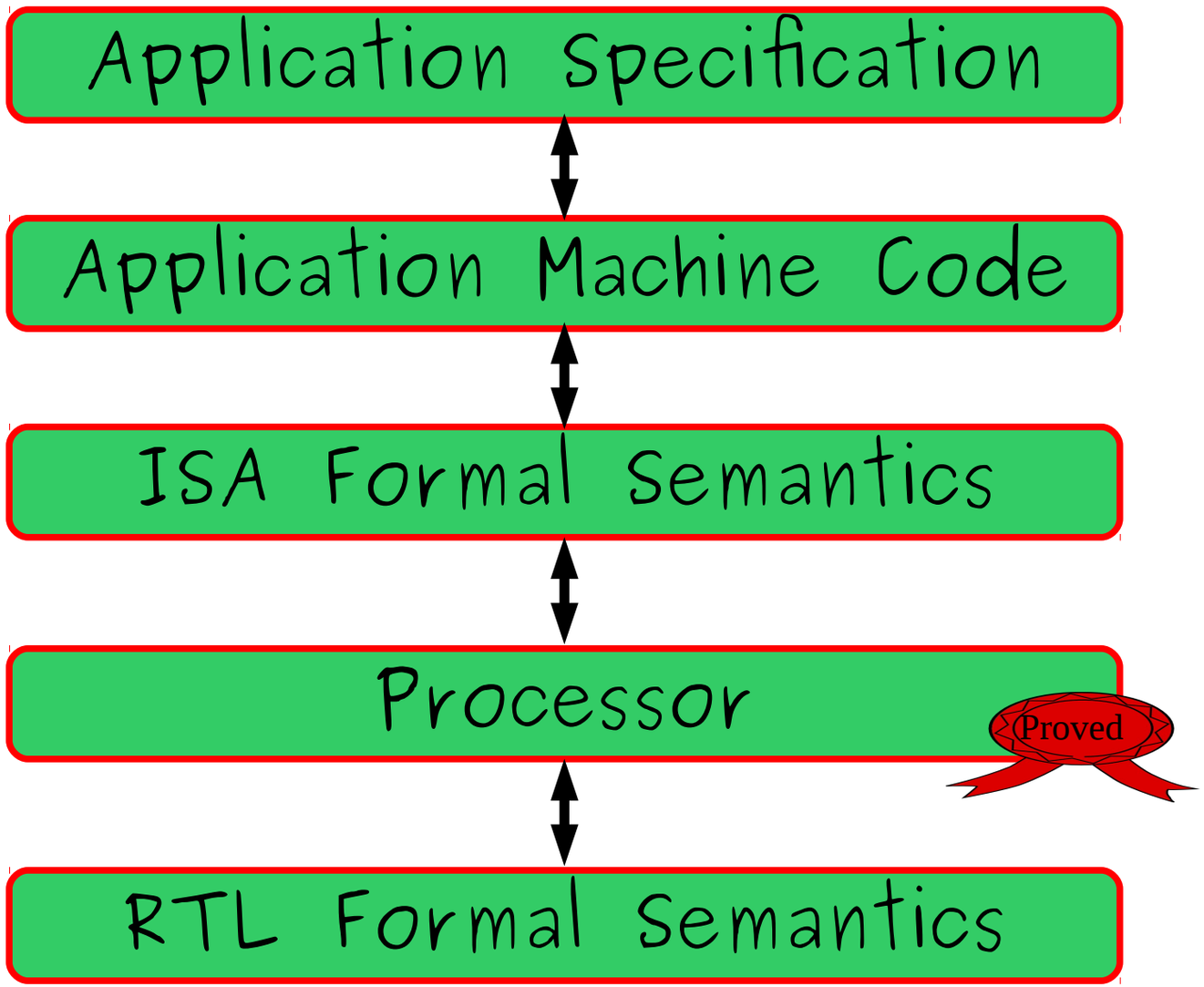
The Promise of this Approach

ISA Formal Semantics

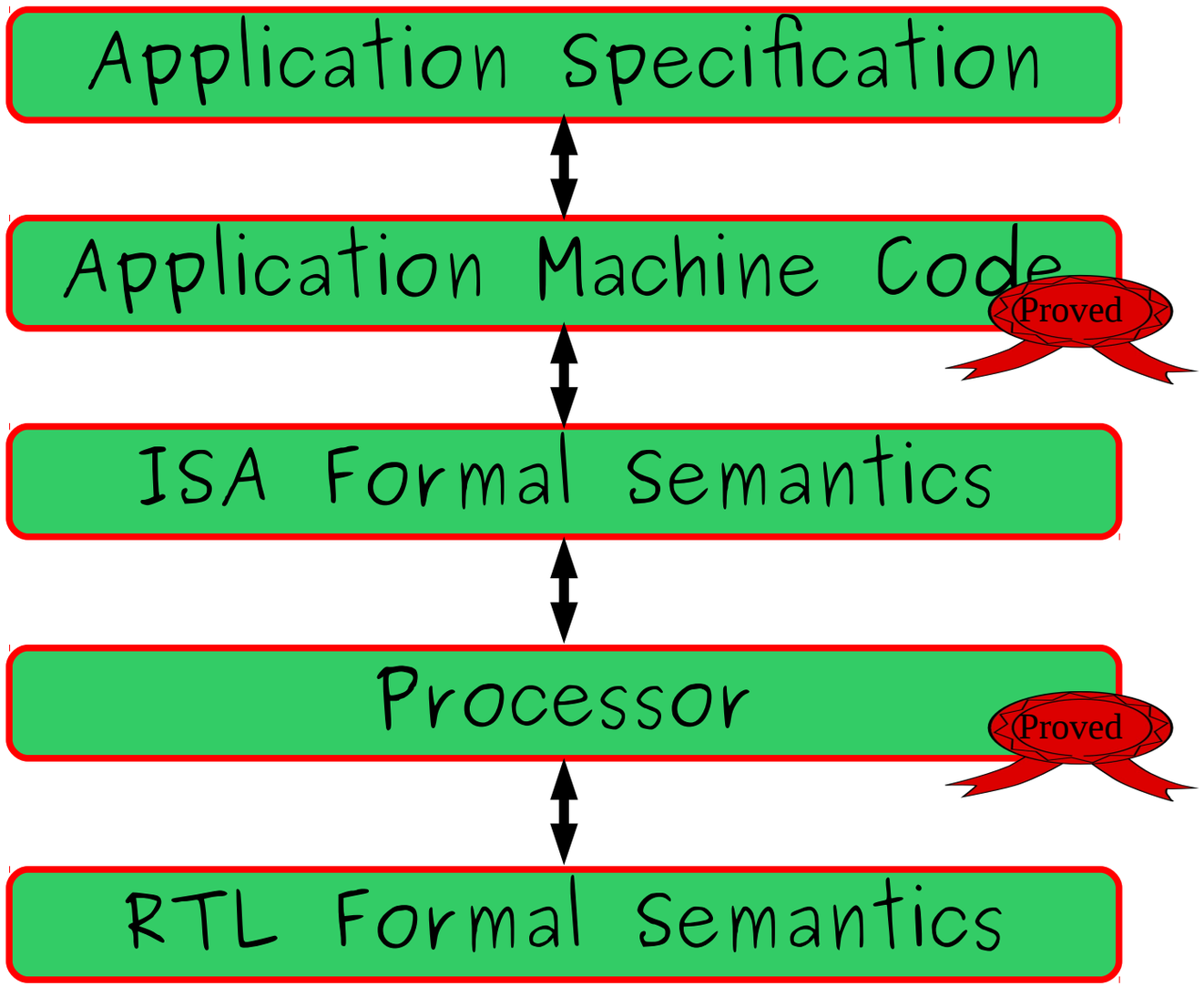
The Promise of this Approach



The Promise of this Approach



The Promise of this Approach



The Trusted Computing Base

Where can defects go uncaught?

The Trusted Computing Base

Where can defects go uncaught?

- ☑ Coq proof checker (small & general-purpose)
- ☑ RTL formal semantics
- ☑ Application specification

The Trusted Computing Base

Where can defects go uncaught?

- [x] Coq proof checker (small & general-purpose)
- [x] RTL formal semantics
- [x] Application specification
- [ ] ISA formal semantics
- [ ] Hardware design (Bluespec, RTL, …)
- [ ] Software implementation (C, …)

Shameless plug:

Part of a larger project:
The Science of Deep Specification
A National Science Foundation
Expedition in Computing

https://deepspec.org/

Join our mailing list for updates on our 2018 summer school: hands-on training with these tools!