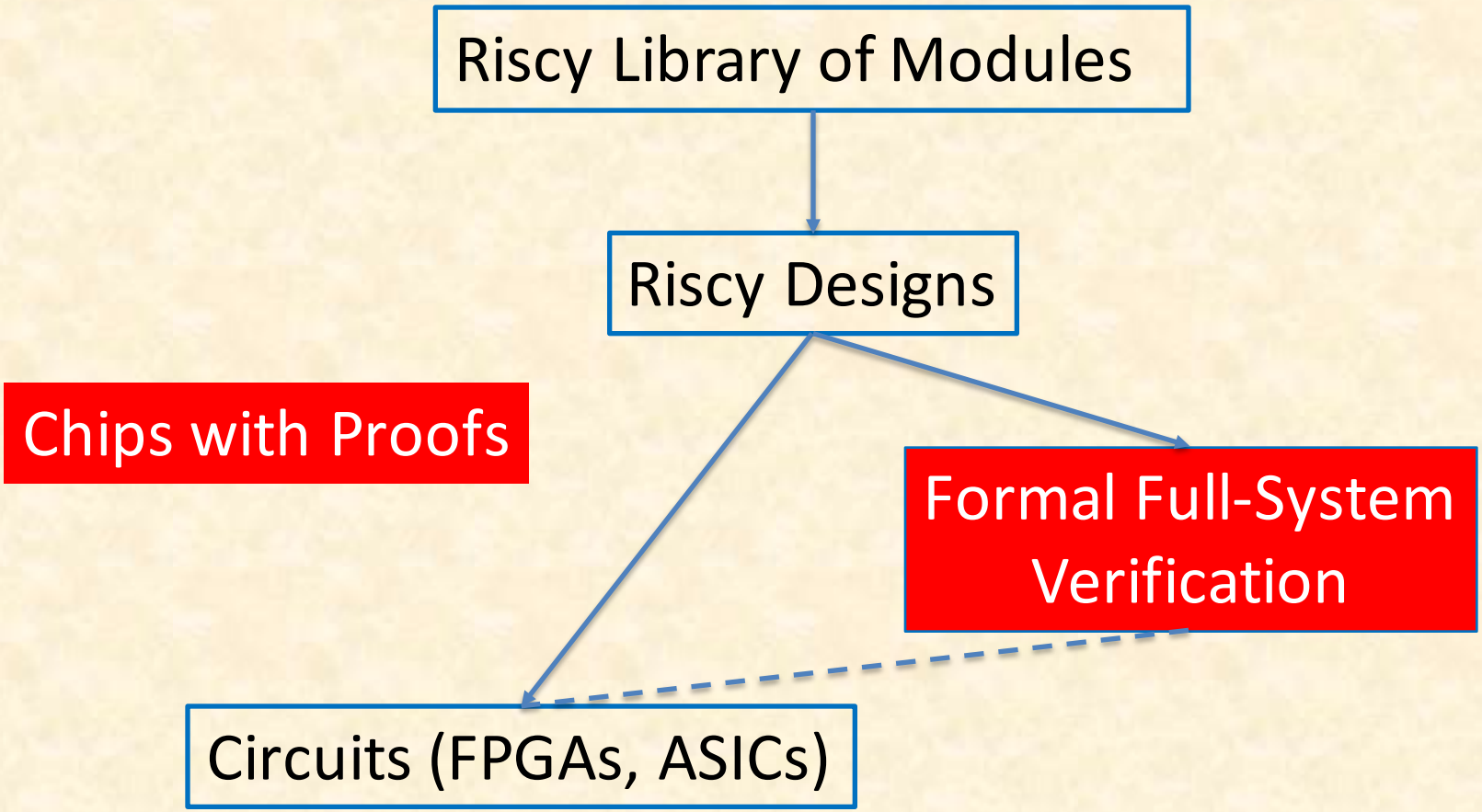
Kami: A Framework for (RISC-V) HW Verification

Murali Vijayaraghavan

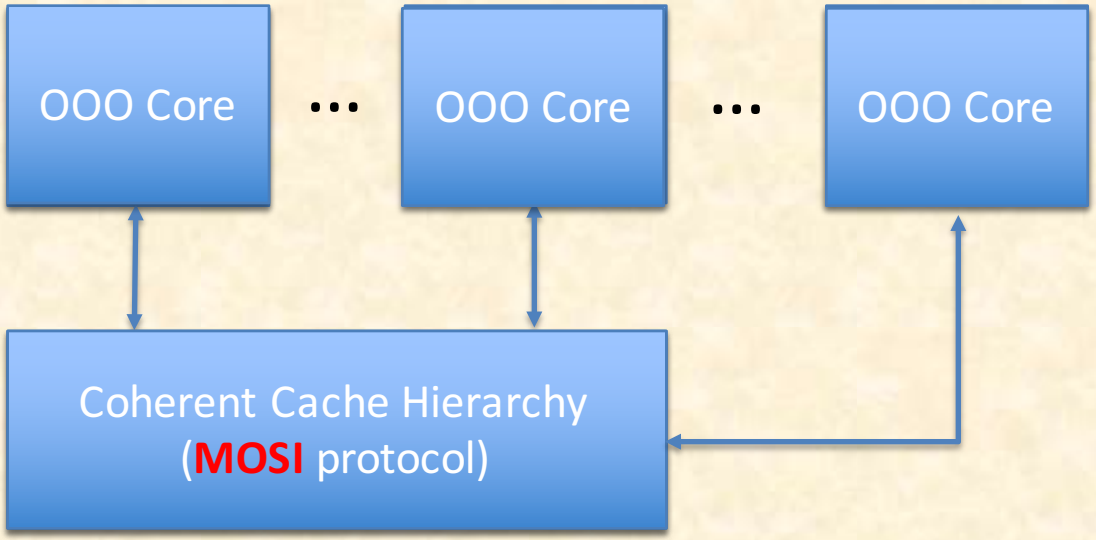
Joonwon Choi, Adam Chlipala, (Ben Sherman),

Andy Wright, Sizhuo Zhang, Thomas Bourgeat, Arvind

The Riscy Expedition by MIT



Modular Verification of a Full-System



$$\frac{(A+B)\checkmark \qquad (A' \text{ optimizes } A)\checkmark}{(A'+B)\ \checkmark}$$

Must be able to verify that optimization is correct independent of contexts

Must be able to verify in presence of parameters instead of just in concrete settings

Semantics for Modular Verification

A optimizes B $\Leftrightarrow$
IO sequences of A $\subseteq$ IO sequences of B

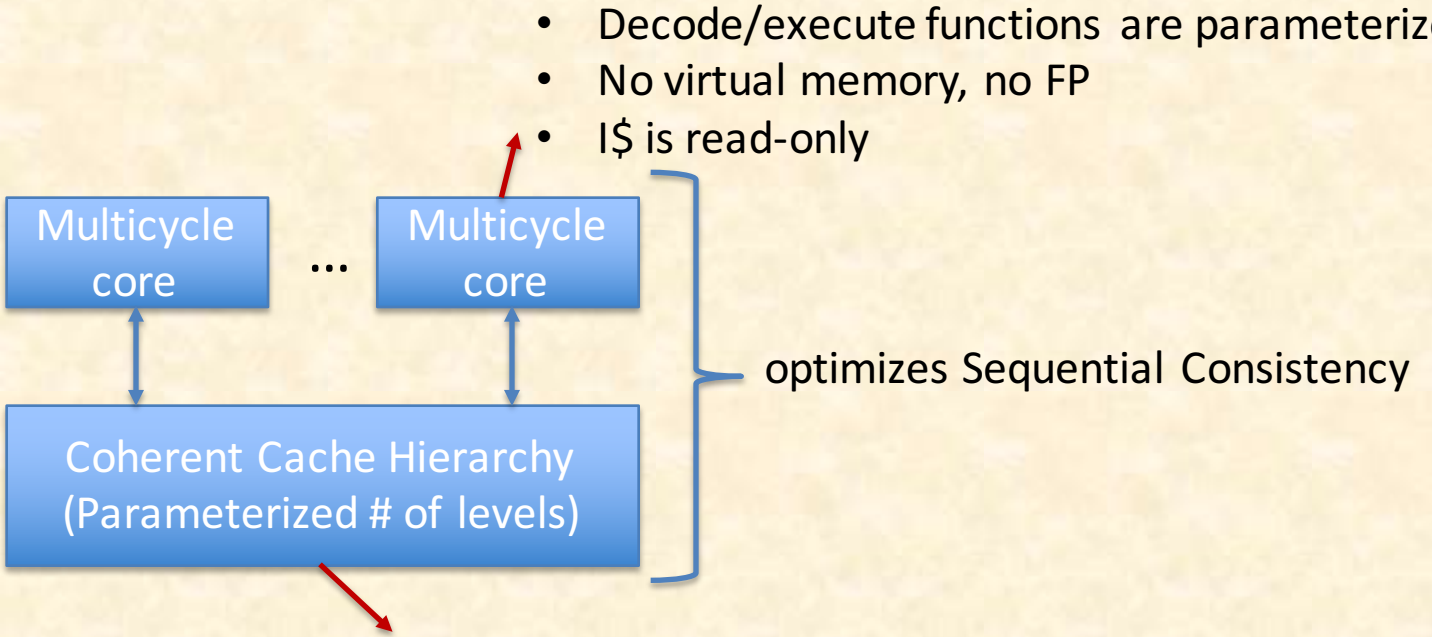
Kami Verification Framework

- DSL in the Coq Proof Assistant for verifying Bluespec-style H/W
  - Embodies the modular verification semantics
  - Descriptions in Kami can be transliterated from-and-to Bluespec
  - IO Ports are Bluespec methods, state transitions are Bluespec rules
- Supports arbitrary parametrization
  - For e.g, you can parameterize a cache hierarchy on arbitrarily shaped trees
  - Verfication theorems can be of the form

    "$\forall n$. Multicore with n processors implements SC"
- Enables semi-automatic verification
  - All invariants must be supplied manually
  - Proving invariants is mostly automatic

Work in Progress

- Finished building required theory and proof automation infrastructure
- Example we are working on:



Conclusions

- Kami: general-purpose HW formal verification framework used for Riscy expedition
  - Chips with Proofs: Plan is to verify a multiprocessor system with OOO cores connected to coherent cache hierarchies

We need a formal multicore/memory model specification first

Thank you!

http://plv.csail.mit.edu/kami

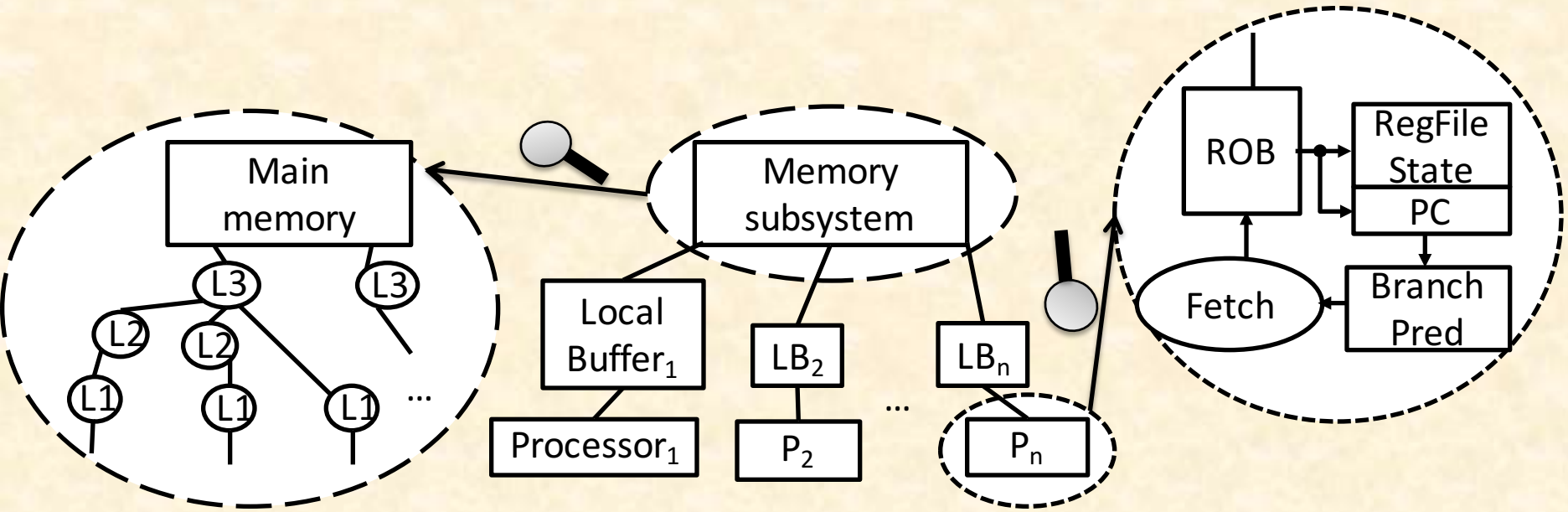
Backup

Example of a Cache rule in Kami

```
Rule "missByState" :=
   Read valid <- "procRqValid";
   Assert !#valid;
   Call rq <- rqFromProcFirst();
   LET idx <- getIdx #rq@."addr";
   Call tag <- readTag(#idx);
   Call cs <- readCs(#idx);
   Assert (#tag == getTag #rq@."addr" &&
           #cs == $ Sh && #rq@."op");
   Write "procRqValid" <- $$ true;
   Write "procRqReplace" <- $$ false;
   Write "procRqWait" <- $$ false;
   Write "procRq" <- #rq;
   Retv
```

Coq's "notation" mechanism allows using intuitive symbols without writing a parser

Verifying a RISC-V Multiprocessor System



- How do we verify that a fully optimized multiprocessor system containing OOO superscalar cores and a hierarchy of coherent caches implements the (multicore) RISC-V specification?

Challenges in Verification

- Formal Specification of multicore RISC-V has to be given first!
  - Includes memory model issues
- Verification should be done on the actual H/W as opposed to a (potentially simplified) model of the H/W
- Verification should be modular
  - Refining the processor from, say, an atomic I^2E processor to an OOO superscalar processor should not require re-verification of cache-coherence protocol
- Verification should support arbitrary parameterization
  - Verifying concrete instances, say, with 2-cores does not mean a 4-core or 8-core system is correct

1000-feet view of Modular Verification Methodology

- Modules are essentially (finite) state transition systems with inputs and outputs
  - In Bluespec, inputs and outputs are via method calls
- *A* refines *B* if any trace (sequences of I/Os) generated by *A* during a sequence of state transitions can be generated by *B*
- Modules compose if they generate identical traces for the communicating ports
  - The communicating port is hidden after composition

Caveats/TODOs with Kami Framework

- The Coq Proof Assistant requires supplying manual proofs that will be machine-checked
  - We are developing several tools to automate the task of proving non-complex invariants/theorems
  - But at the very least, the full set of invariants have to be supplied manually
- Specification must be rigorous
  - No room for "evolving" specifications
  - But components can be specified abstractly without giving implementations (for example, decoder/ALU can be specified as *uninterpreted functions* without giving a concrete instance)

Thank You