Industrial Strength Refinement Checking

Jesse Bingham, John Erickson,
Gaurav Singh, and Flemming Andersen
Intel IAG
FMCAD 2009

Introduction

- Standard approach to FV of HW protocols
  - Develop high level model (HLM) in guarded-command-like language (eg Murphi, TLA, Spin etc)
  - Write invariants, e.g. cache coherence
  - Model check *as big as you can*
- So the HLM is golden, but what about the implementation (RTL)?
  - Ideal: *prove* that RTL implements HLM… hard!
  - Our solution: *test* that RTL implements HLM during dynamic simulation
  - *check* == *test* in this talk/paper

Key point #1

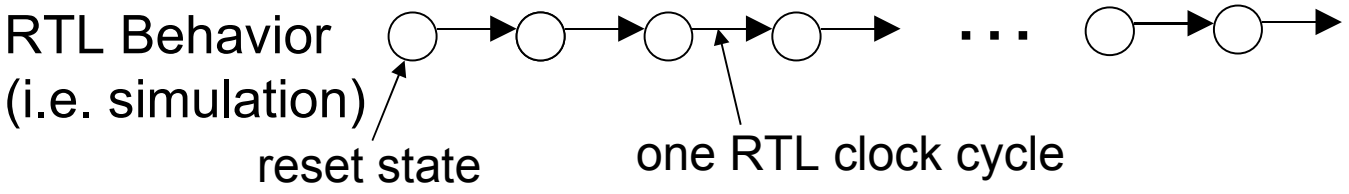
The ingredients needed for equivalence testing are also needed to prove implementation.

$\Rightarrow$ might as well start with testing

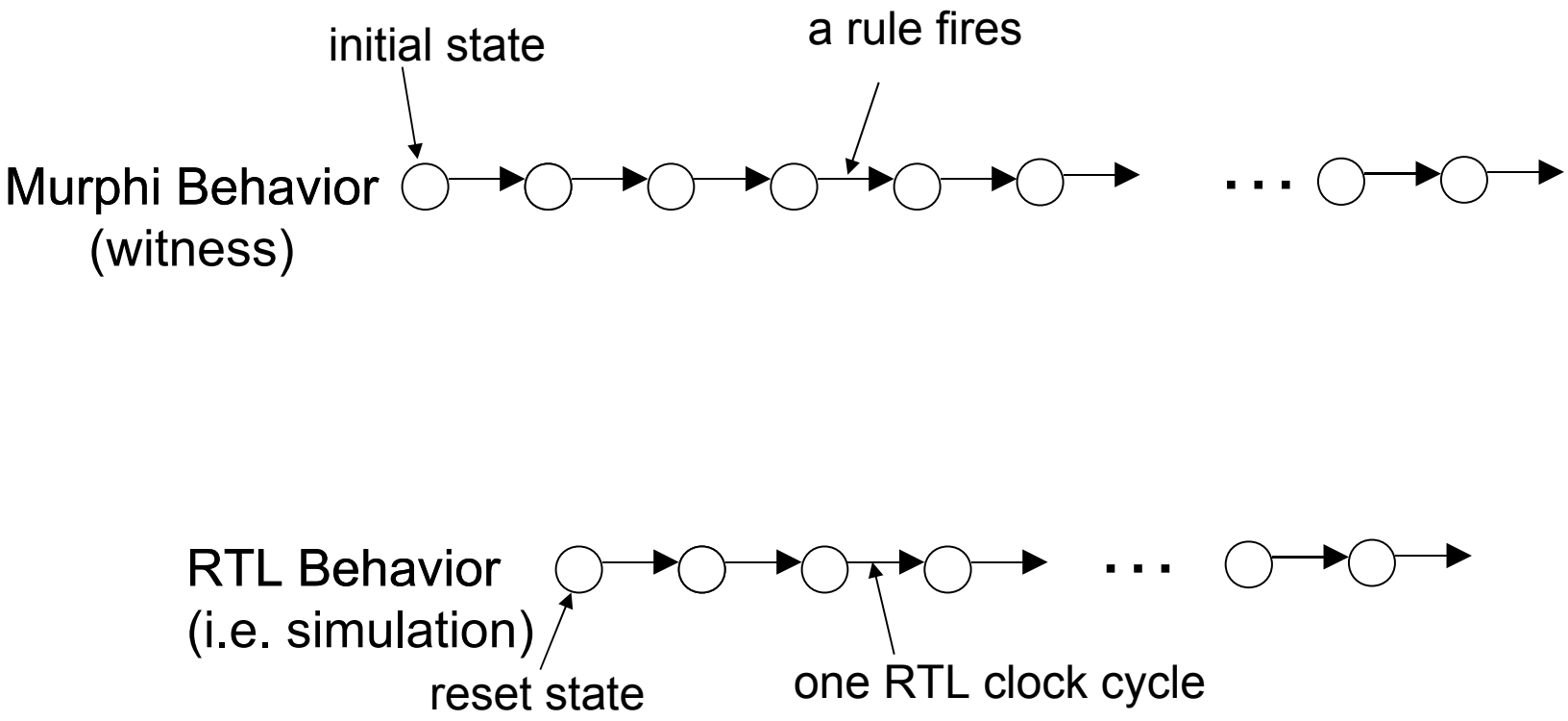
What should Implements *Mean*?

- What does it mean for RTL to implement HLM? They have different
  - execution semantics
  - state variables/representations
  - rule atomicity (HLM has more)
  - rule concurrency (RTL has more)
- Not always clear [Vardi FMCAD09]
- For our domain, we found a notion we call *behavioral refinement* appropriate…
  - Similar to notion of Bluespec and also super-scalar processor verification literature

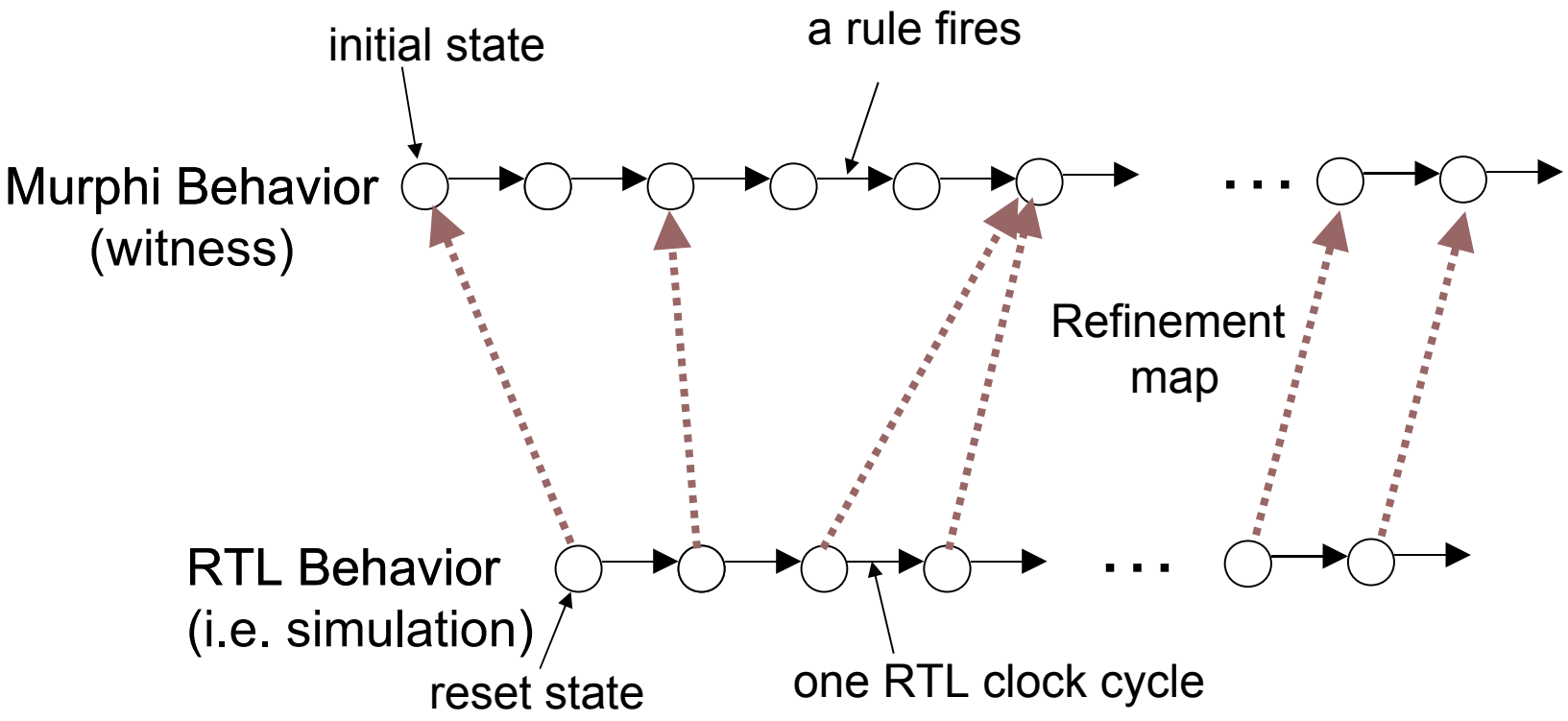
Behavioral Refinement



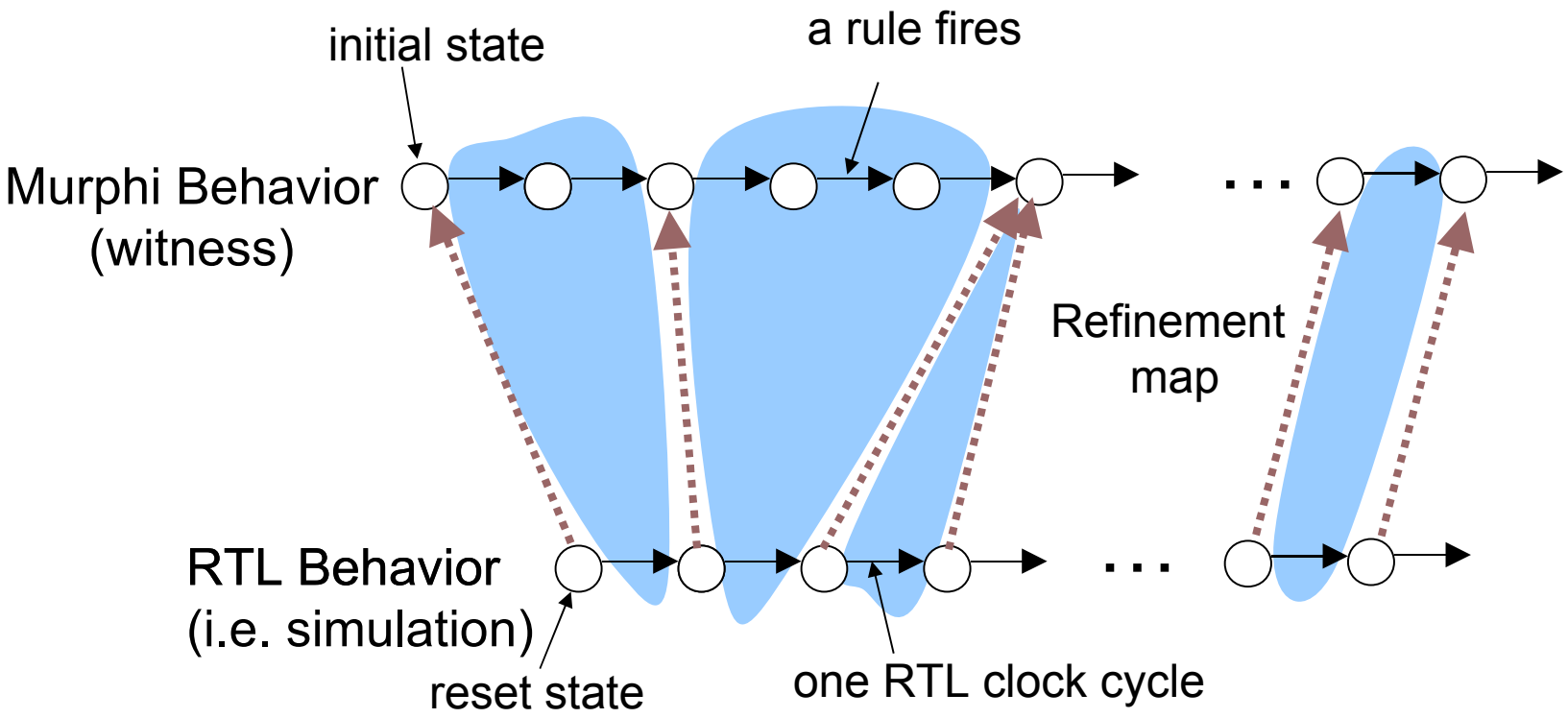
Behavioral Refinement



Behavioral Refinement



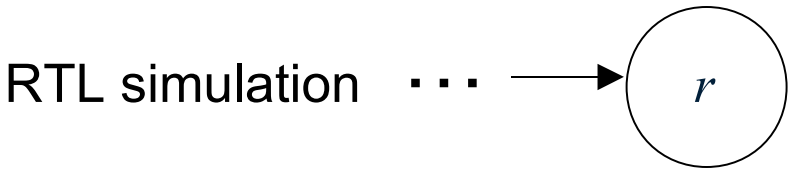
Behavioral Refinement



Each RTL clock cycle corresponds to zero or more rules firing

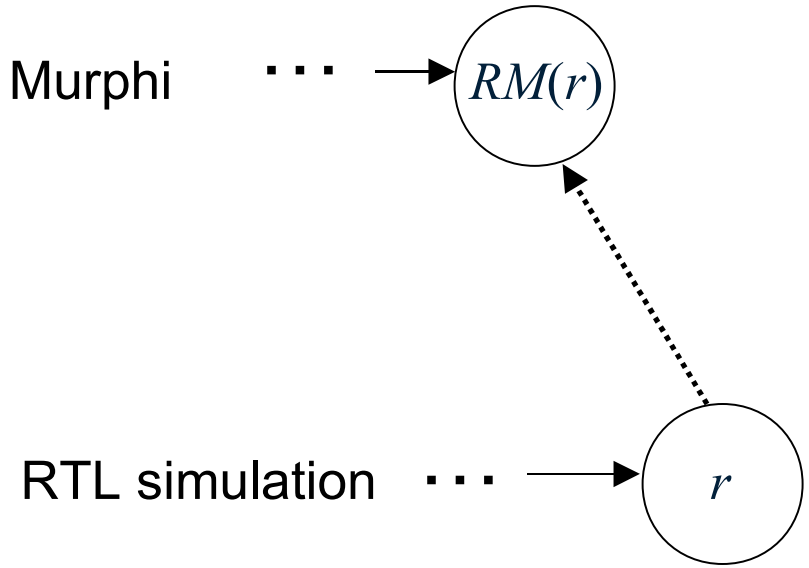
How Refinement Checker Works

Idea: at each RTL cycle, *select* what sequence of rules are about to fire



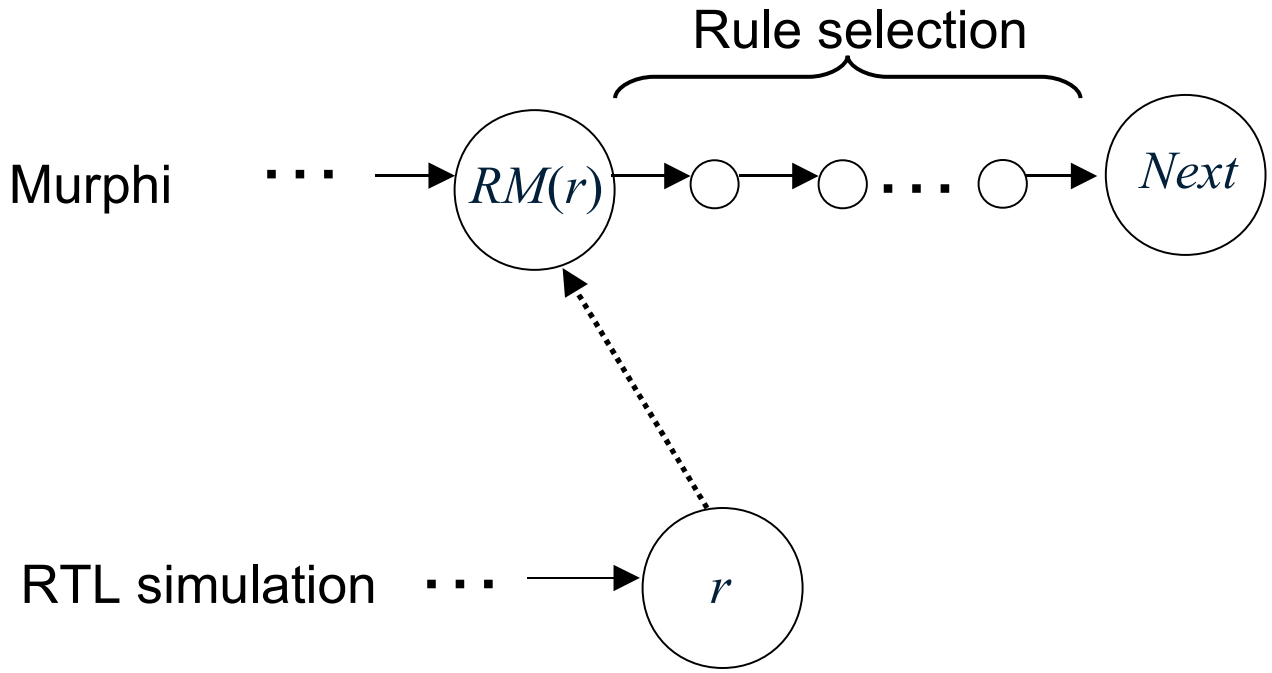
How Refinement Checker Works

Idea: at each RTL cycle, *select* what sequence of rules are about to fire



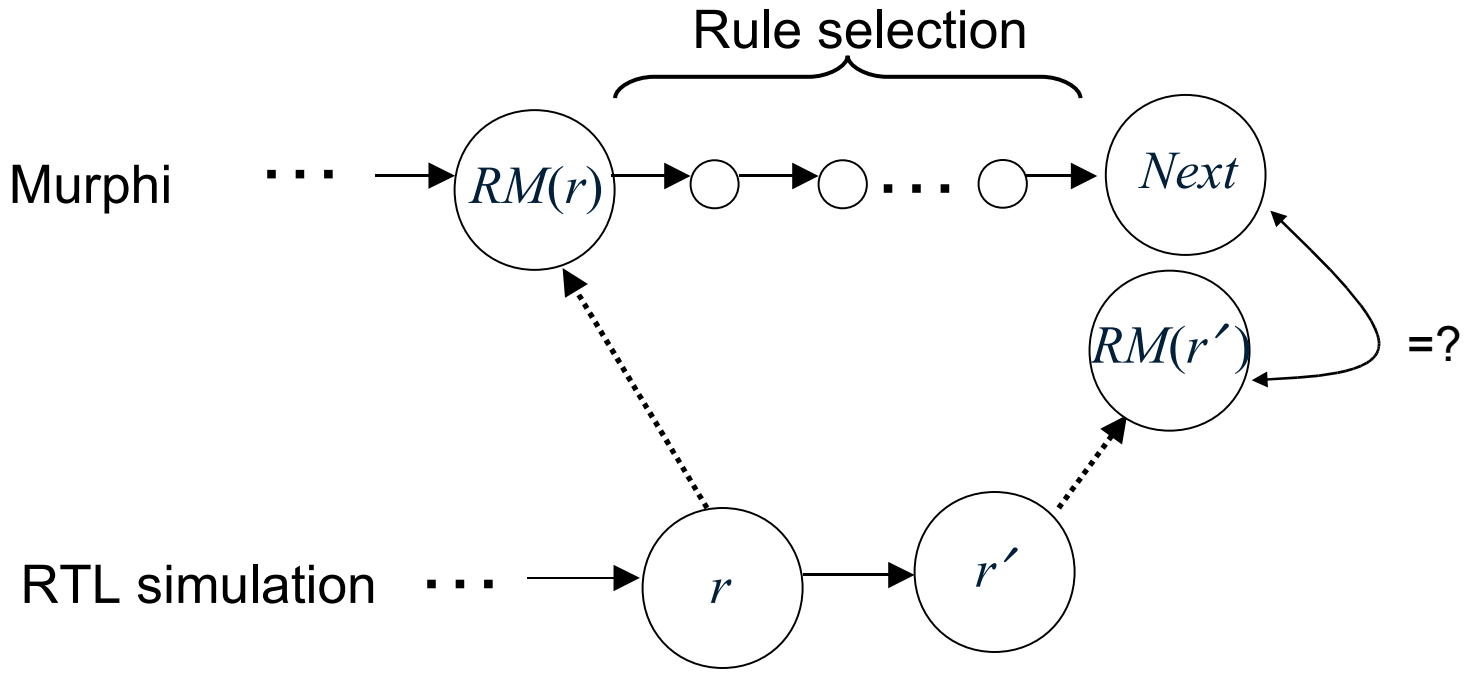
How Refinement Checker Works

Idea: at each RTL cycle, *select* what sequence of rules are about to fire

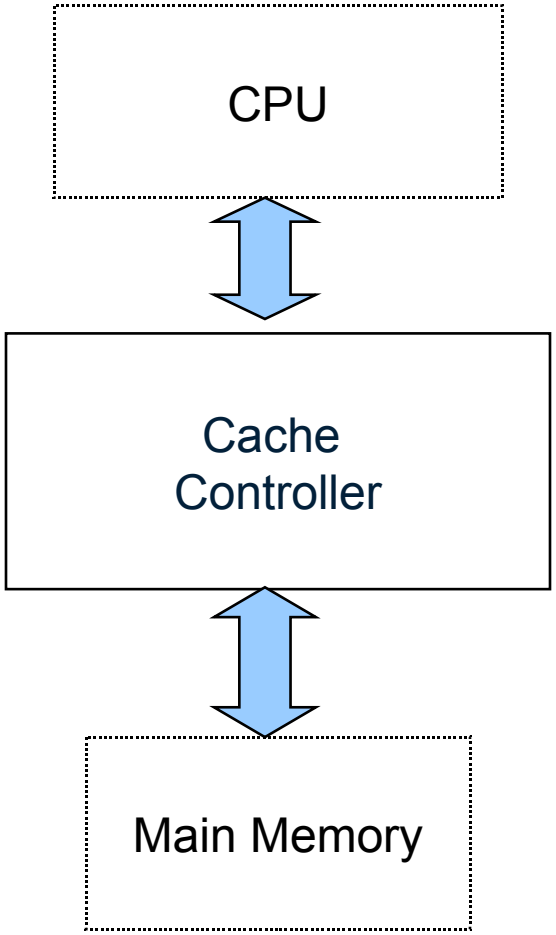


How Refinement Checker Works

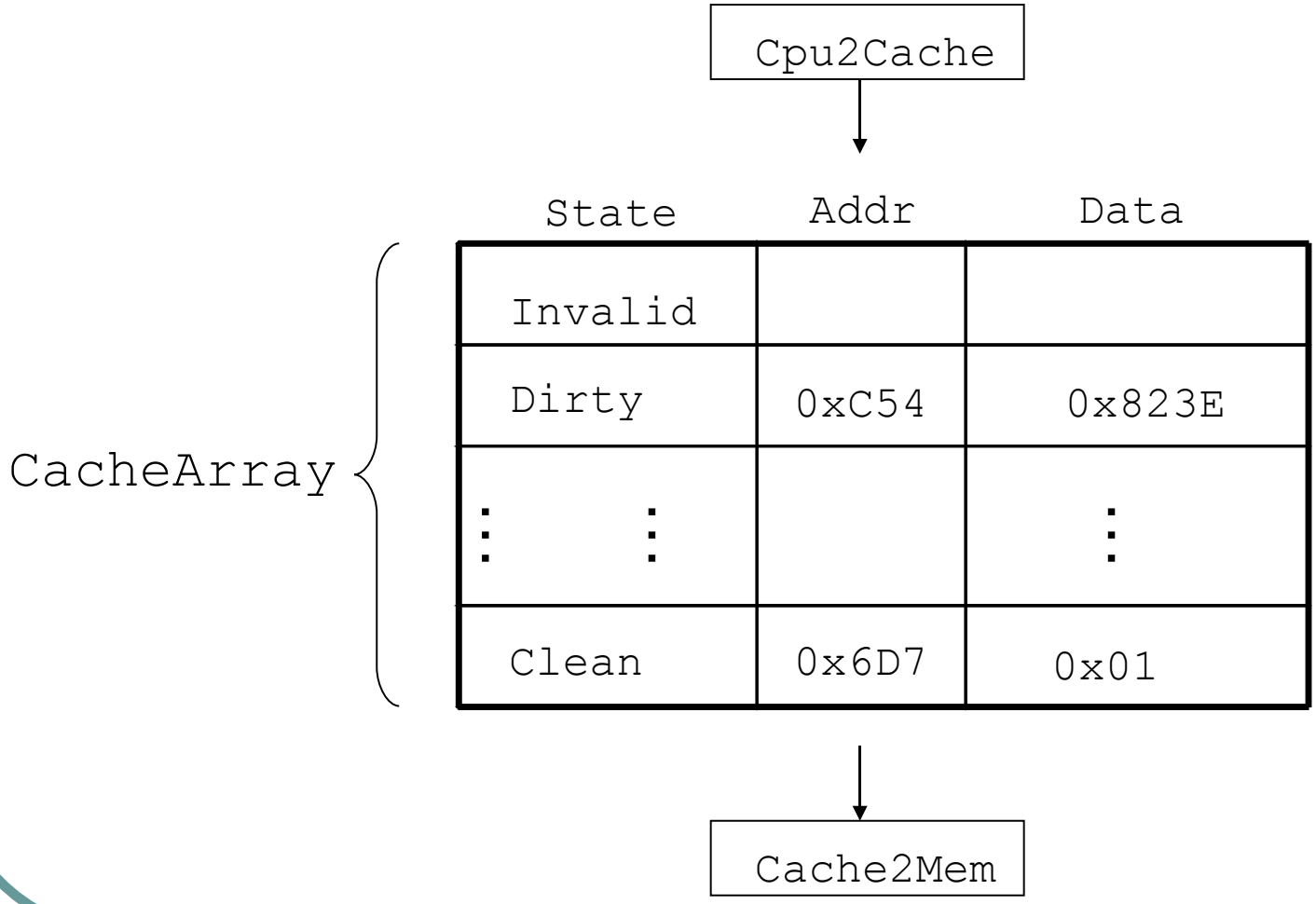
Idea: at each RTL cycle, *select* what sequence of rules are about to fire



Example: Toy Cache Controller



Toy Cache in Murphi



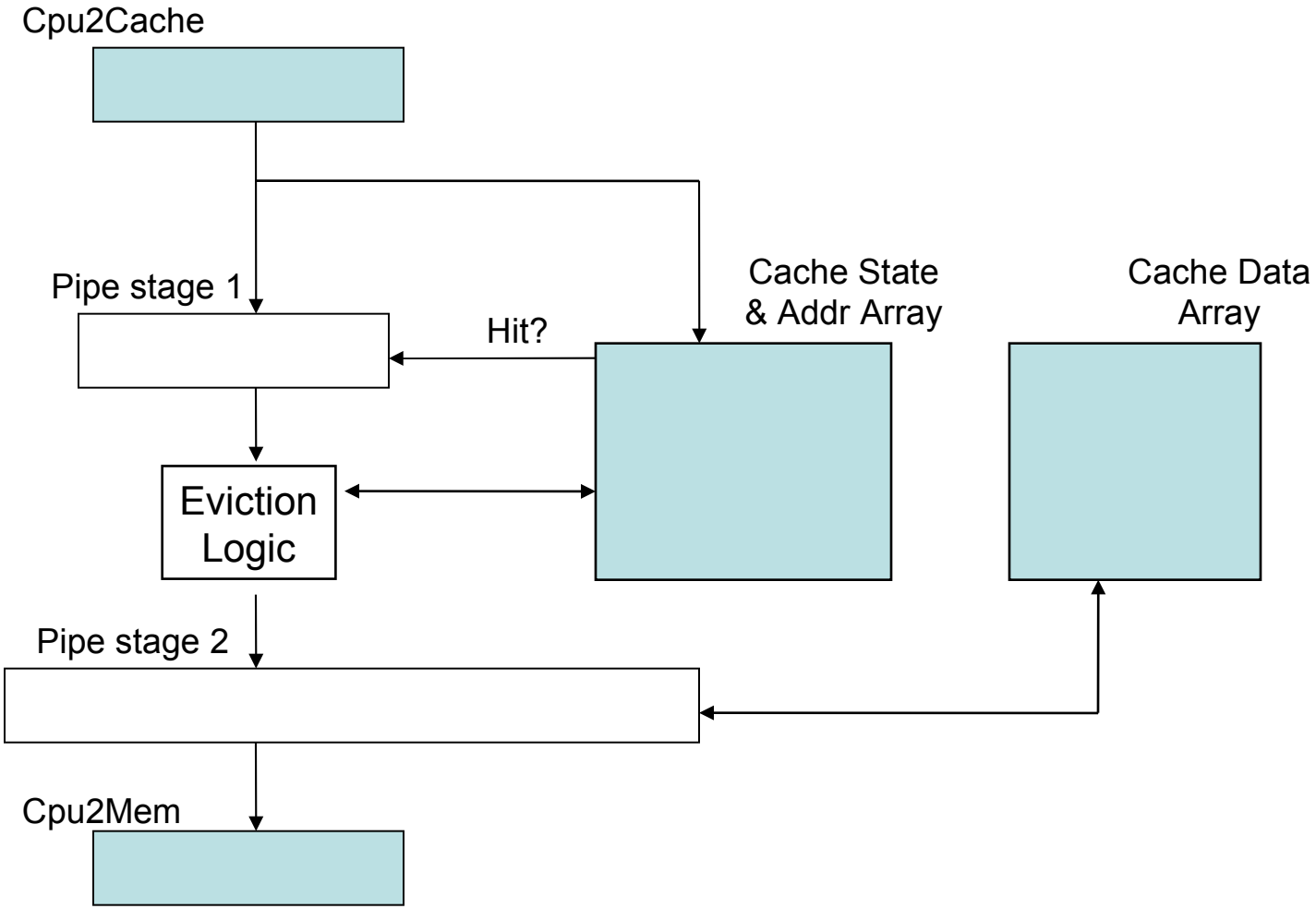
Eviction

```
Ruleset i : CacheIndex "Evict"
  CacheArray[i].State != Invalid
==>
  if (CacheArray[i].State == Dirty) begin
    Cache2Mem.opcode := WriteBack;
    Cache2Mem.Addr = CacheArray[i].Addr;
    Cache2Mem.Data = CacheArray[i].Data;
  end;
  CacheArray[i].State := Invalid;
end
```

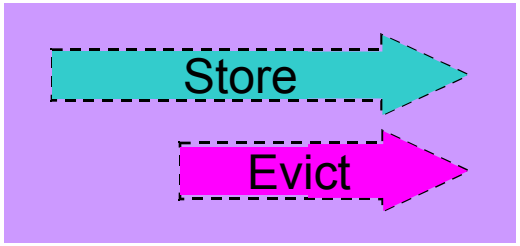
Receiving a Store Request

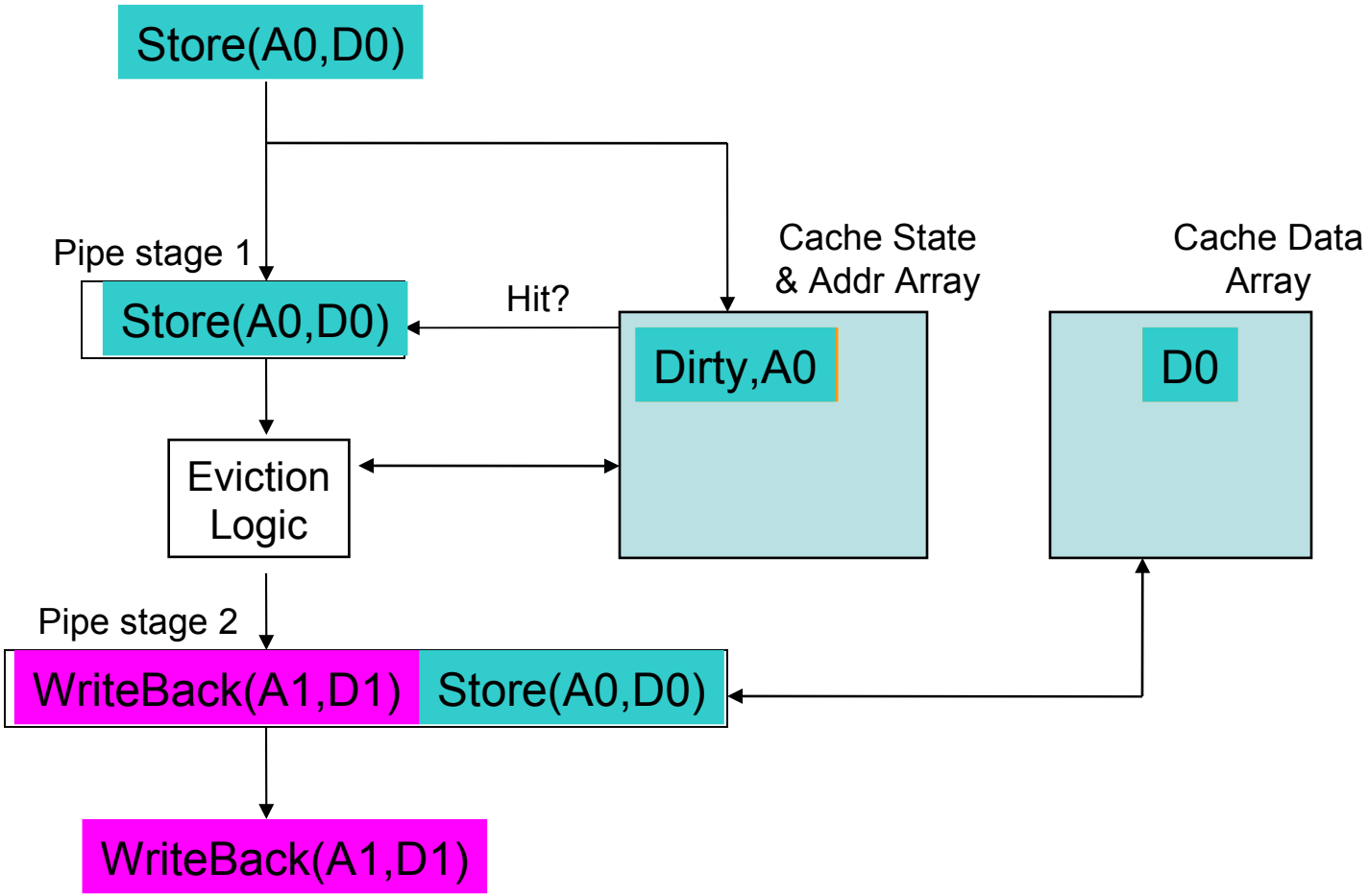
```
Ruleset i : CacheIndex "Recv_Store"
  Cpu2Cache.opcode = Store &
  ( ( CacheArray[i].State != Invalid &
      CacheArray[i].Addr = Cpu2Cache.Addr) |
    ( addr_misses_in_cache(Cpu2Cache.Addr) &
      CacheArray[i].State = Invalid ) ) )
==>
  CacheArray[i].Data := Cpu2Cache.Data;
  CacheArray[i].State := Dirty;
  Absorb(Cpu2Cache);
end
```

Cache Controller RTL



Example RTL Behavior





Key point #2

Pipelining causes rules that are atomic in Murphi to be non-atomic in the RTL…

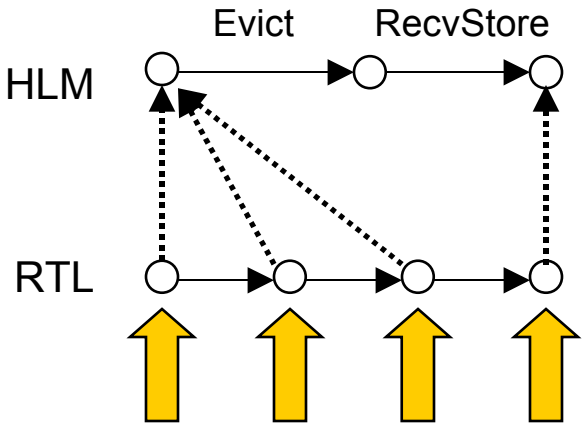
This non-atomicity is resolved by the refinement map & history variables

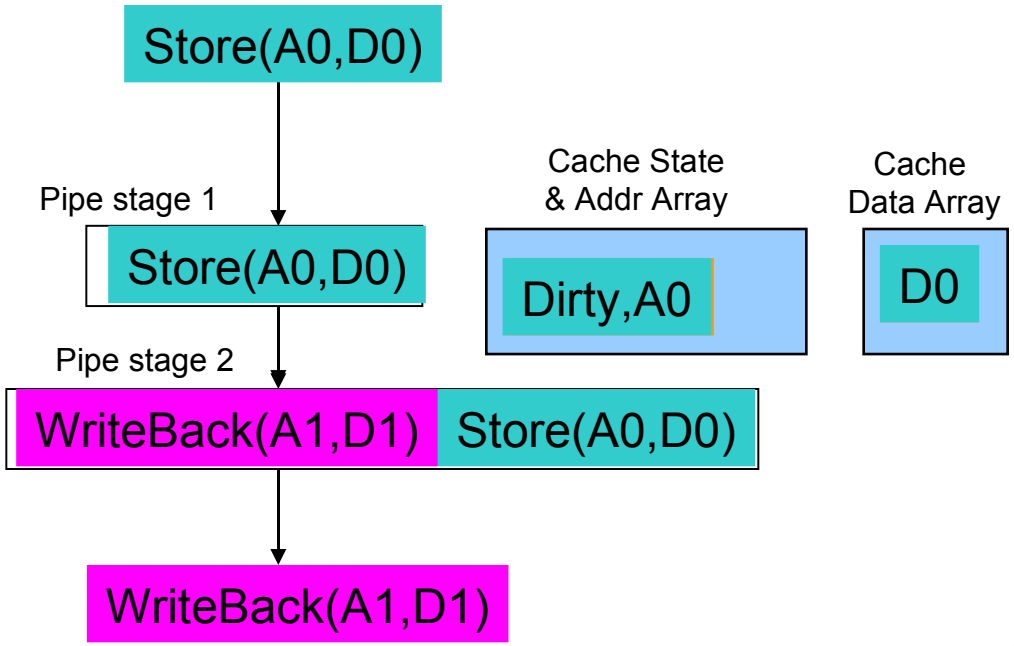
Key point #3

Murphi semantics fire one rule at a time, while RTL has true rule concurrency.

This is resolved by *rule selection*, which picks a sequence of Murphi rules to fire @ each RTL clock cycle

Example with Refinement Checker





BTW: Everything's System Verilog

HW designers
- RTL design under verification

HW validators
- Test stimulus

Us (FV team)
- Refinement Map
- Rule Selection

Paper gives disciplined approach to writing SV code for these buggers

- High Level Model
  - in consultation with Architects
  - compiled into SV by a tool *mu2sv*

⇒any off-the-shelf SV simulator works

mu2sv

- Translates a Murphi model into SV
- Typedefs, procedures, functions, procedures, invariants
- State variables get wrapped in a record type called **`MURPHI_STATE`**
- Murphi rule **`R`** becomes SV function

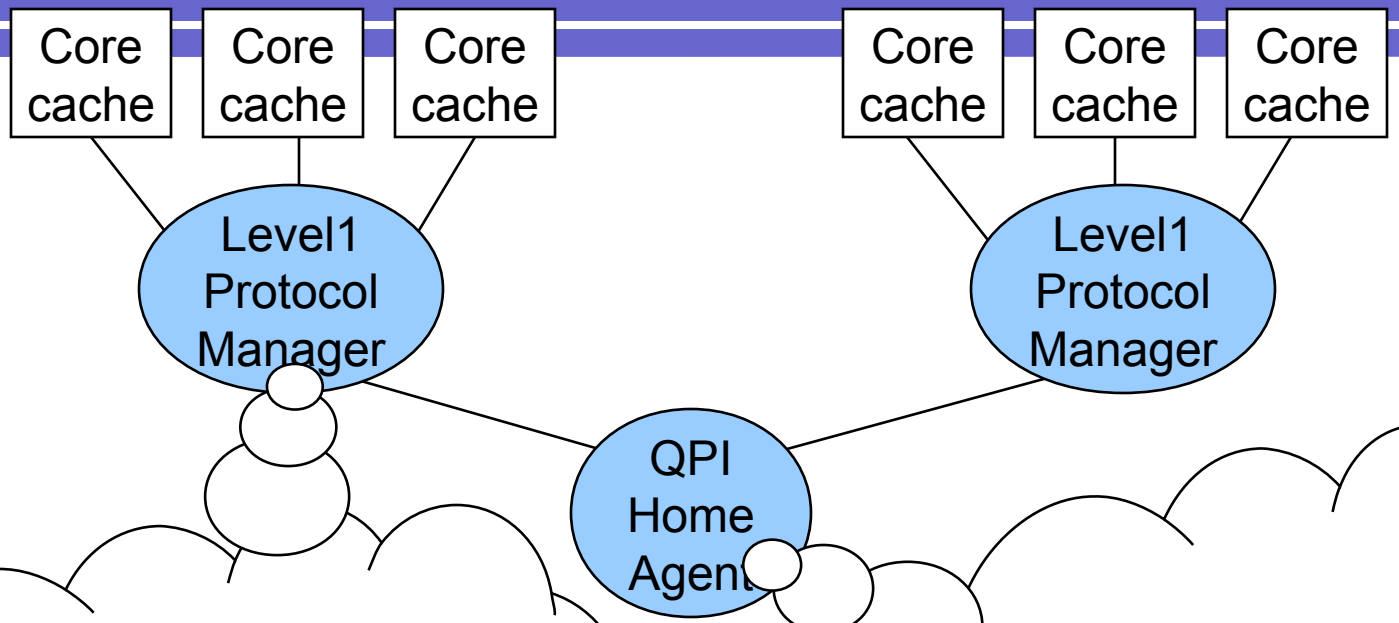
```
function MURPHI_STATE R_sv(MURPHI_STATE ms,...);
```

  - Errors if invoked when **`R`**'s guard is false in **`ms`**
- Rule coverage logging
  - Valuable feedback for test-writers

Inspiration

- S. Tasiran, Y. Yu, and B. Batson, *Linking simulation with formal verification at a higher level*. IEEE DToC, 2004.
  - Used TLA+ & linked TLC model checker to simulation engine
  - Done as research *after* the project was complete
  - Showed that subtle bug would have been caught

Application: Hierarchical Cache Protocol



- 3 person months to develop
- Caught 8 bugs during just 1 month of deployment!

- Was not deployed due to chip cancellation ;-(
- Could allow up to 8 murphi rules to fire per RTL clock