Bluespec: Why chip design can't be left EE's

Arvind
CSAIL (formerly LCS and AI Lab)
Massachusetts Institute of Technology

University of California, Irvine
March 22, 2004

A looming crisis in chip design

- Microprocessors
  - 100M gates ⇒ 1B gates
- ASIC's
  - 5M to 10M gates ⇒ 50M to 100M gates
  - 18 months to design but o*nly an eight-month selling opportunity in the market*

Issues:
design time,
cost,
team size, …

- 5M gate ASIC costs $10M to design/fab
  - ~20 manyears ≅ $3M
  - Tool cost ≅ $2.5M + $1.5M(Hardware)
  - NRE ≅ $1.5M + $1M for each spin

Pressing problems of chip design

- Problems of the small
  - Leaky transistors, porous oxide, multiple Vt and their control

    The "electrical engineering" in IC design
- Problems of the large
  - Millions of transistors, thousands of complex blocks working together correctly
  - Design methods must scale: Hierarchical organizations, correctness by construction, abstractions and static analysis, formal verification ...

    The "computer science" in IC design

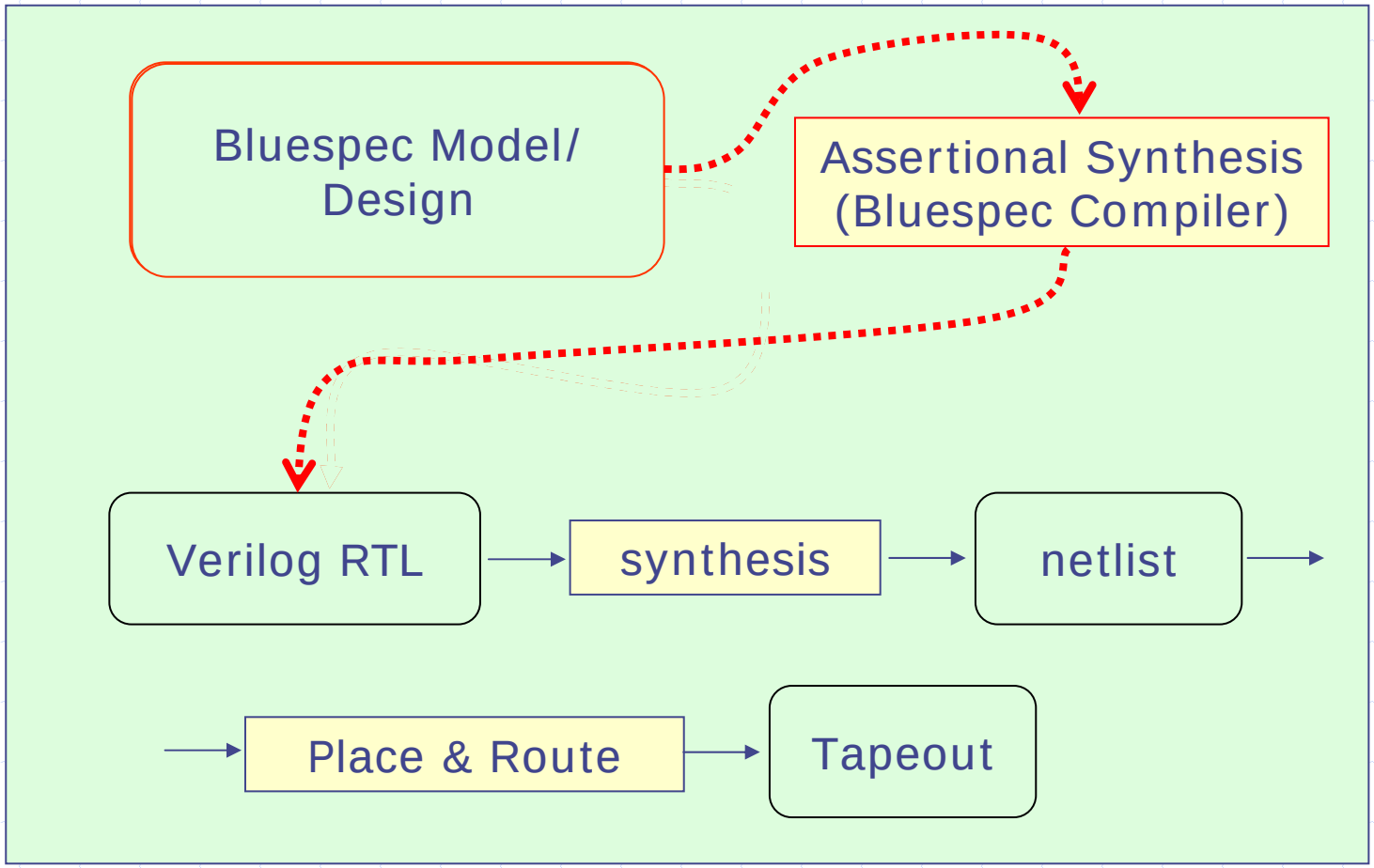
Example: Power Management

- EE's have identified the problem and shown the solution at the circuit level
  - Clock gating
  - Power gating

but an application of these ideas requires analysis of a design at a much higher level (e.g., microarchitecture, RTL) than circuits

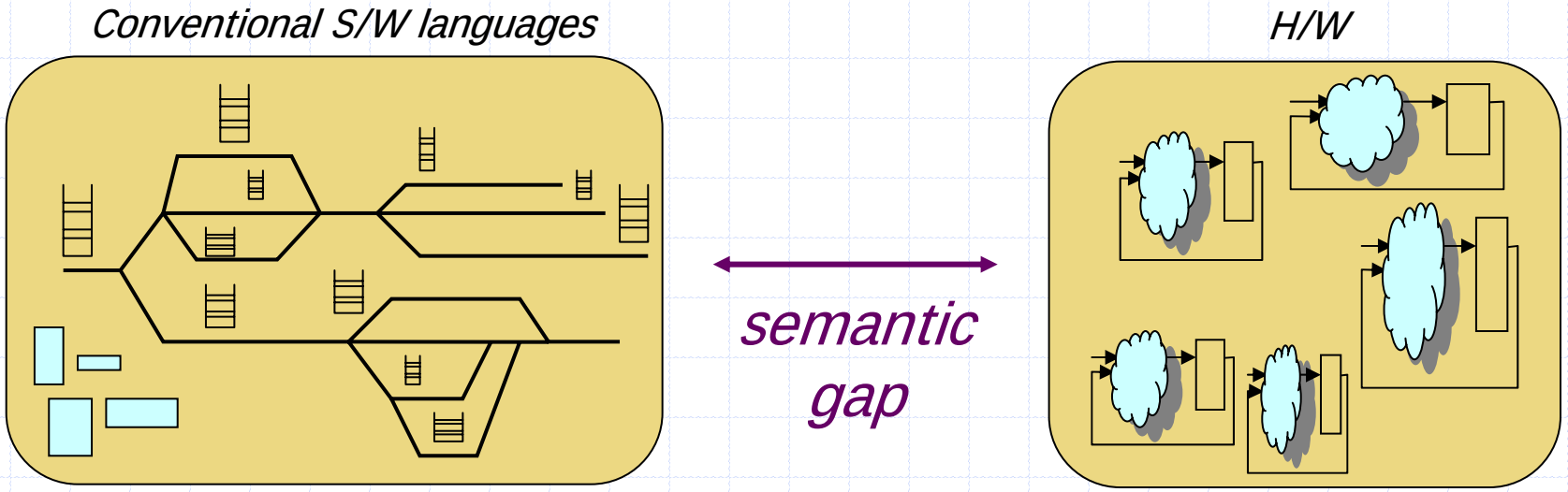
- CS folks have better tools and methodologies for solving these problems *provided they don't tune out at the first mention of clock skews and leakage currents.*

Bluespec Design Flow

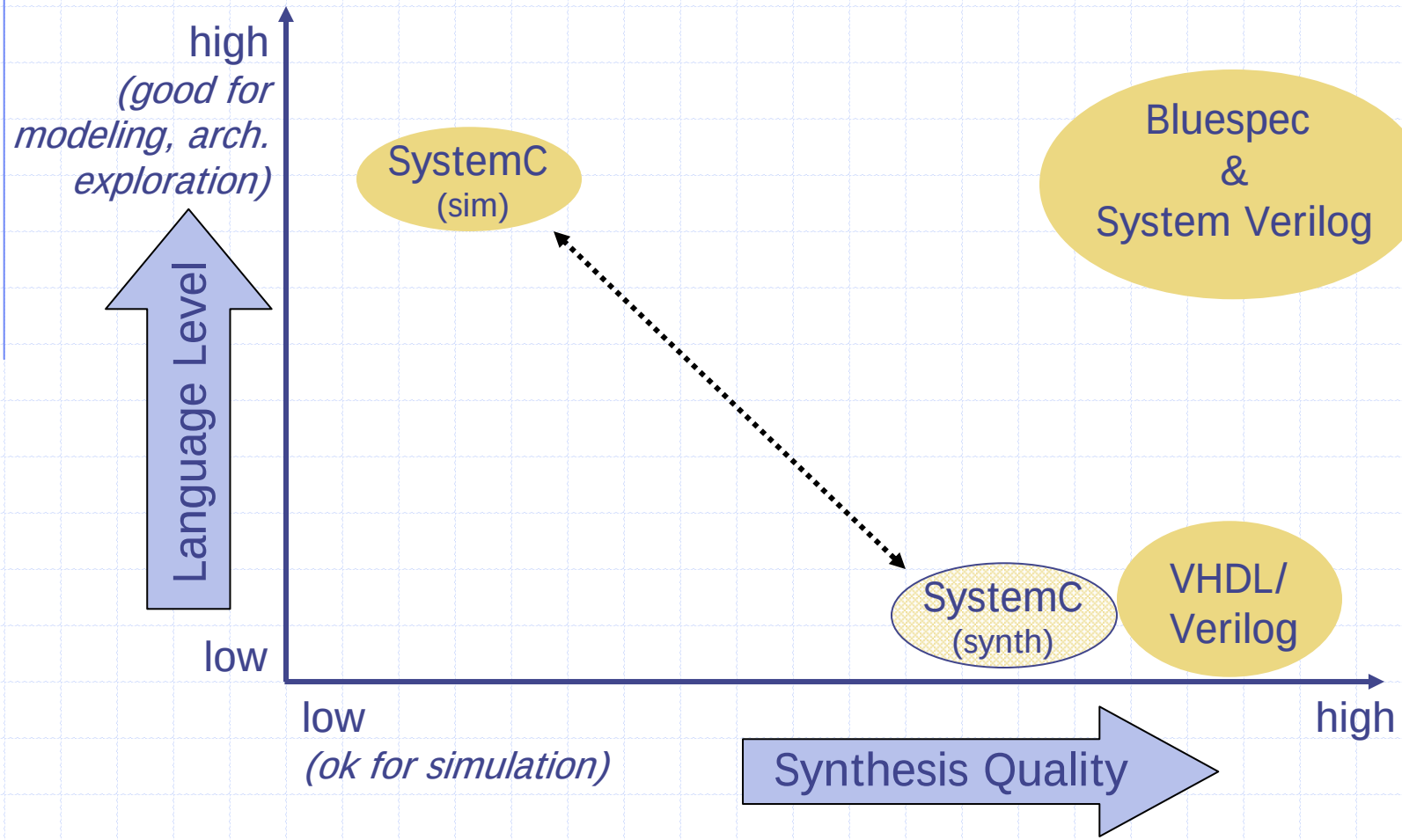


What Ails High-Level Synthesis?

- People have viewed “high-level synthesis” or “behavioral synthesis” as moving hardware languages closer to C, C++
  - this has created a semantic gap for hardware designer’s, and
  - a nightmare for hardware synthesis tools



Bluespec & SystemVerilog

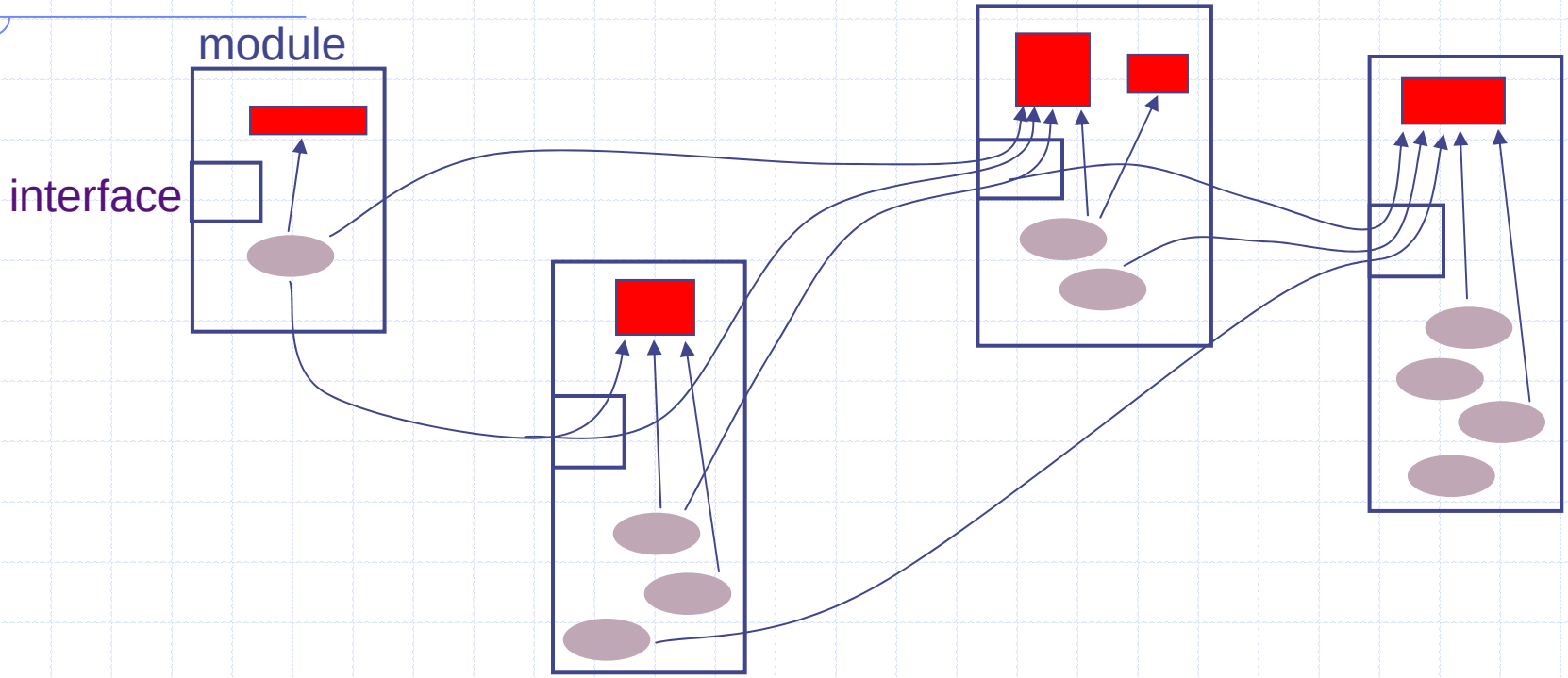


What is Bluespec

An register transfer language with

- Atomicity assertions
  - Any behavior can be understood in terms of a series of atomic actions on state elements (e.g., FFs, Registers, Reg Files, FIFO's, RAMs, ...)
- Powerful "generics" and "generate"
  - Static elaboration of source code to generate *both datapaths and control*

Bluespec: State and Rules organized into *modules*



All the *state* (e.g., Registers, FIFOs, RAMs, ...) is explicit.

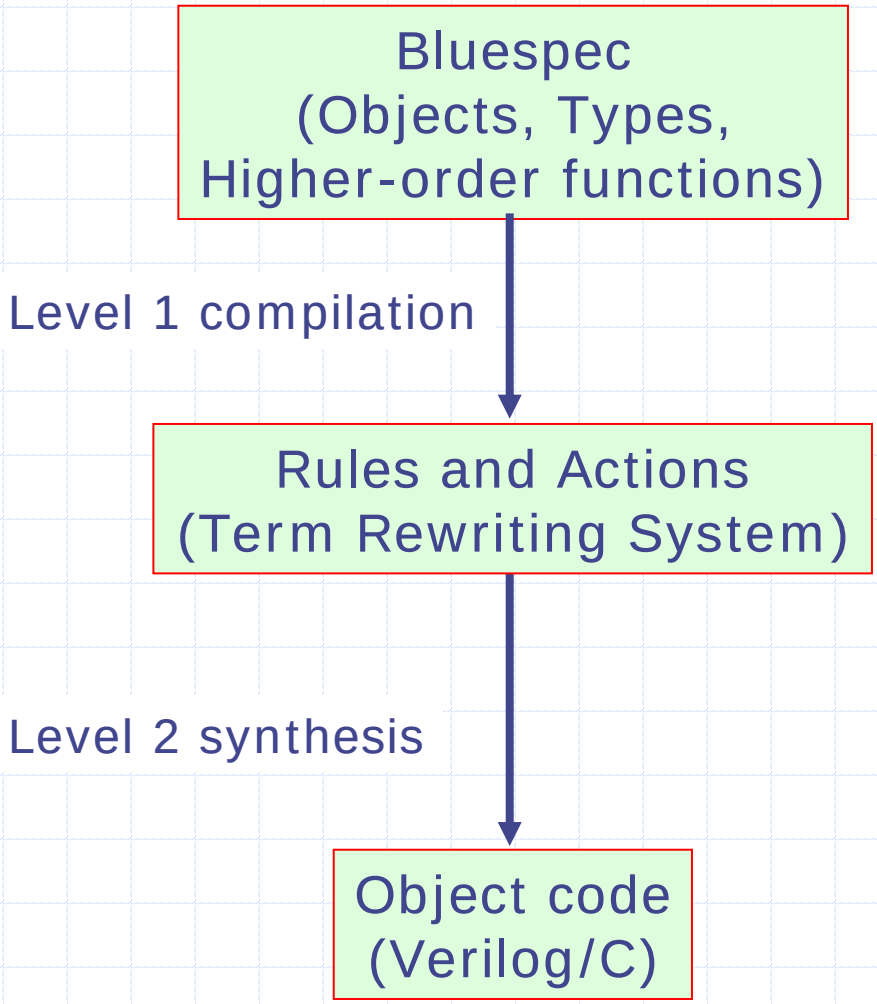
Behavior is expressed in terms of atomic actions on the state:

Rule: condition ➔ action

Modules are like *objects* (private state, interface methods, rules).

Rules can manipulate state in other modules only *via* their interfaces.

Bluespec: Two-Level Compilation



Lennart Augustsson
@Sandburst 2000-2002

- Type checking
- Massive partial evaluation and static elaboration

- Rule conflict analysis
- Rule scheduling

James Hoe & Arvind
@MIT 1997-2000

GCD in Bluespec

```
module mkGCD (GCD);
  Reg#(int) x;   Reg#(int) y;
  x = mkReg(_);  y = mkReg(0);

  rule ((x > y) && (y != 0))  begin x <= y; y <= x; end
  rule ((x <= y) && (y != 0)) y <= y - x;

  method start(ix,iy) when (y==0) begin x <= ix; y <= iy; end
  method result       when (y==0) return x;
endmodule;
```

State

Internal behavior

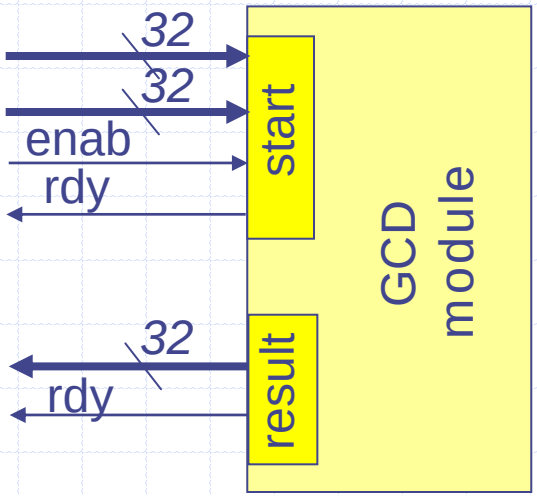
External interface

`rule' is like `always'

NB: Not the syntax of the current compiler

GCD Hardware Module

implicit conditions




```
interface GCD;
  method Action  start (int x, int y);
  method int result;
endinterface
```

Nikhil's visualization

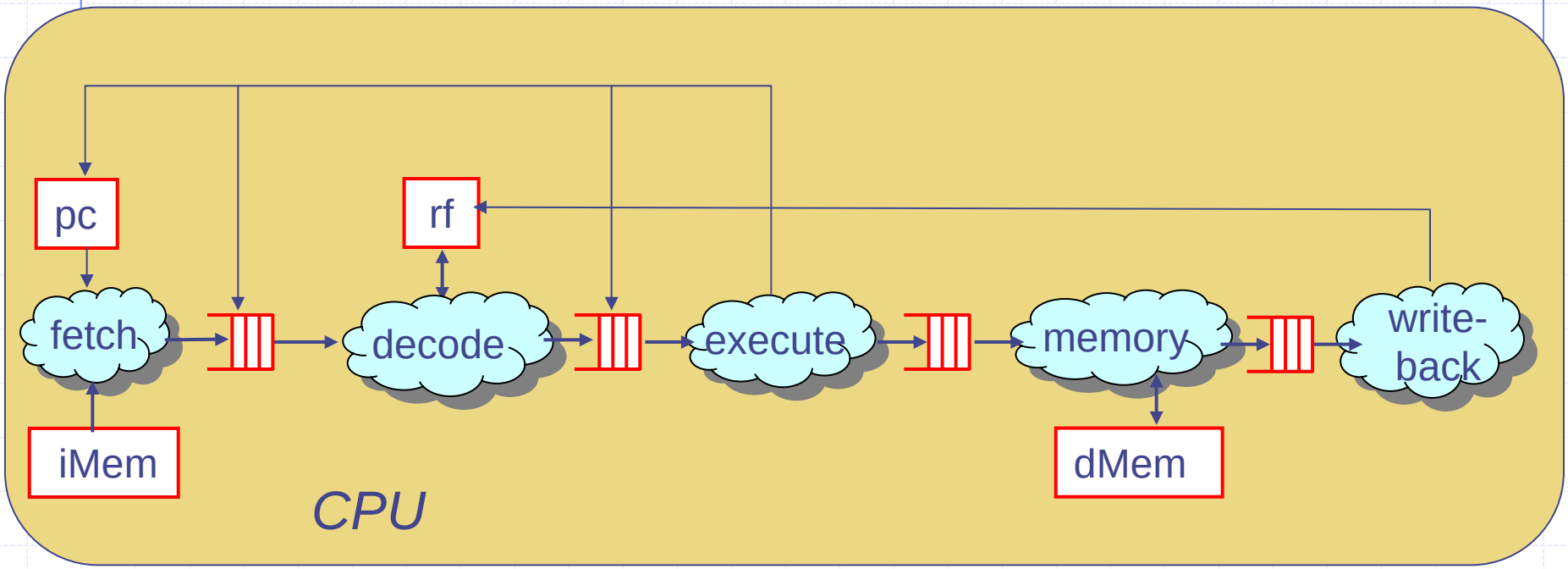
Generated Verilog RTL: GCD

```
module mkGCD(CLK, RST_N,start__1, start__2, E_start_, ...)
  input CLK; ...
  output start__rdy; ...
  wire [31 : 0] x$get; ...
  assign result_ = x$get;
  assign _d5 = y$get == 32'd0;
  ...
  assign _d3 = x$get ^ 32'h80000000) <= (y$get ^ 32'h80000000);
  assign C___2 = _d3 && !_d5;
  ...
  assign x$set = E_start_ || P___1;
  assign x$set_1 = P___1 ? y$get : start__1;
  assign P___2 = _d3 && !_d5;
  ...
  assign y$set_1 =
      {32{P___2}} & y$get - x$get | {32{_dt1}} & x$get |
      {32{_dt2}} & start__2;
  RegUN #(32) i_x(.CLK(CLK), .RST_N(RST_N), .val(x$set_1), ...)
  RegN  #(32) i_y(.CLK(CLK), .RST_N(RST_N), .init(32'd0),  ...)
endmodule
```

Examples

- Processor and FIFOs
  - MIPS on FPGA
- Multicast Arbiter
  - how to pipeline on the fly

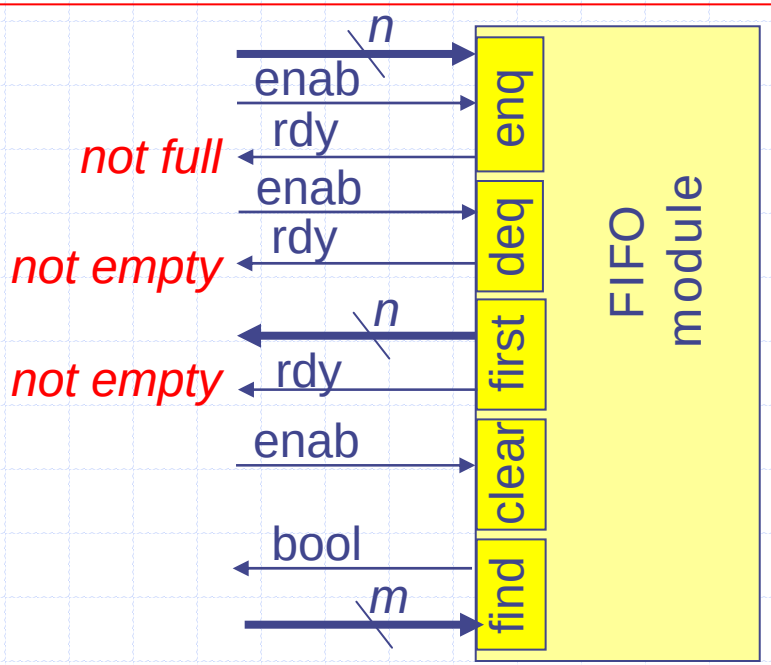
Processor Pipelines and FIFOs



Synthesis: Dan Rosenband @ MIT

FIFO (glue between stages)

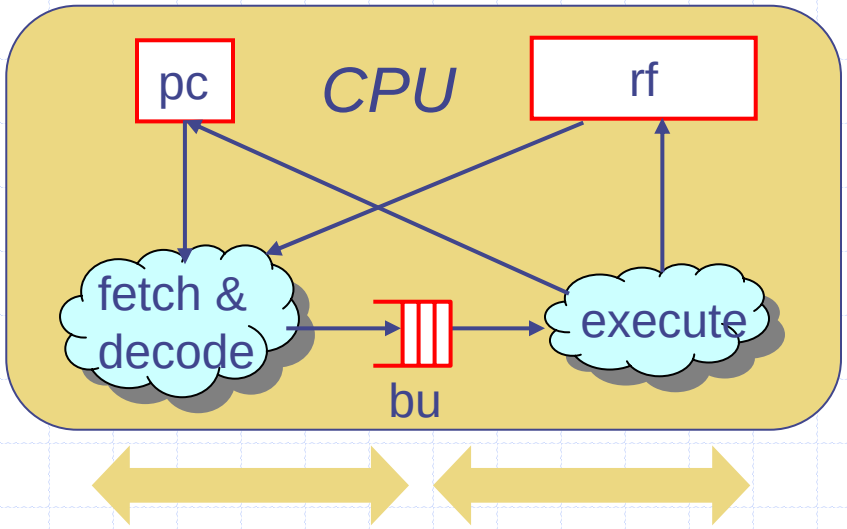
```
interface  FIFO #(parameter type t, tr);
  method Action enq(t);     // enqueue an item
  method Action deq();      // remove oldest entry
  method t first();         // inspect oldest item
  method Action clear();    // make FIFO empty
  method Bool find(tr);     // search FIFO
endinterface
```



n = # of bits needed to represent the values of type “t“

m = # of bits needed to represent the values of type “tr"

Rules for Add



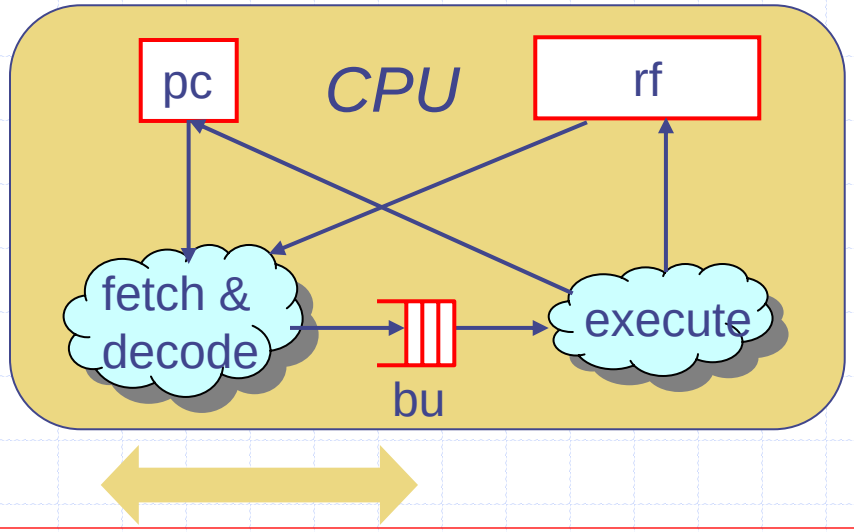
implicit checks:
bu notfull

```
rule (match(iMem[pc],Add{rd,ra,rb})) begin
  bu.enq (EAdd{rd, rf[ra], rf[rb]});
  pc <= pc + 1;
end
```

bu notempty

```
rule (match(bu.first(),EAdd{rd,va,vb})) begin
  rf[rd] <= va + vb;
  bu.deq();
end
```

Fetch & Decode Rule: *Reexamined*

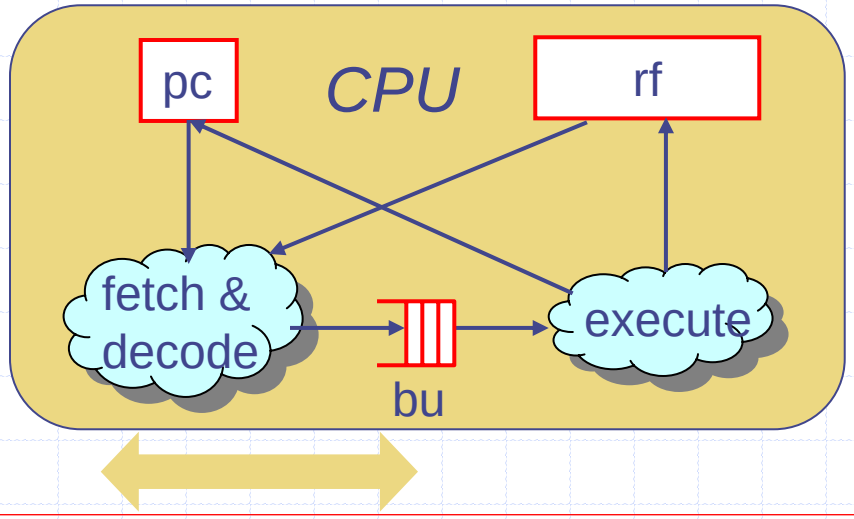



```
rule (match(iMem[pc],Add{rd,ra,rb})) begin

  bu.enq (EAdd{rd, rf[ra], rf[rb]});

  pc <= pc + 1;
end
```

Wrong! Because instructions in bu may be modifying ra or rb

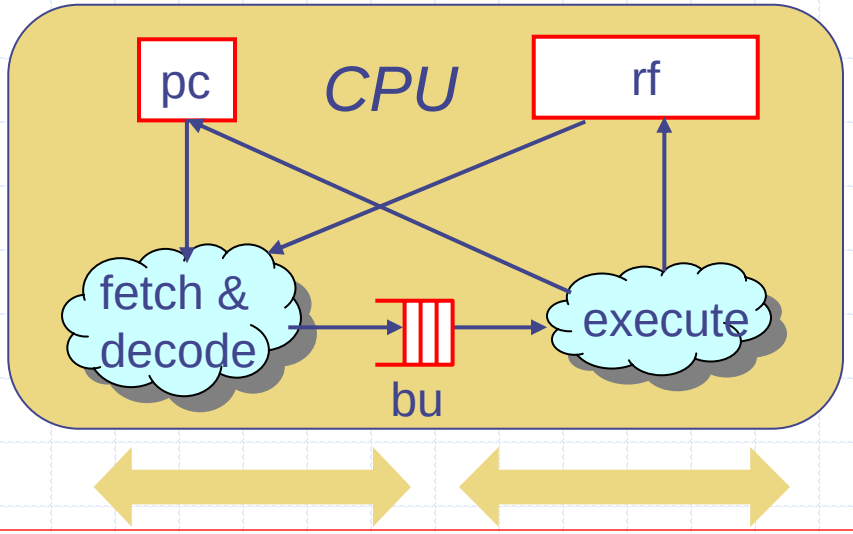
stall !

Fetch & Decode Rule: *corrected*




```
rule (match(iMem[pc],Add{rd,rb,ra}) &&
      !bu.find(ra) && !bu.find(rb)) begin
  bu.enq (EAdd{rd,rf[ra],rf[rb]});
  pc <= pc + 1;
end
```

Rules for Branch

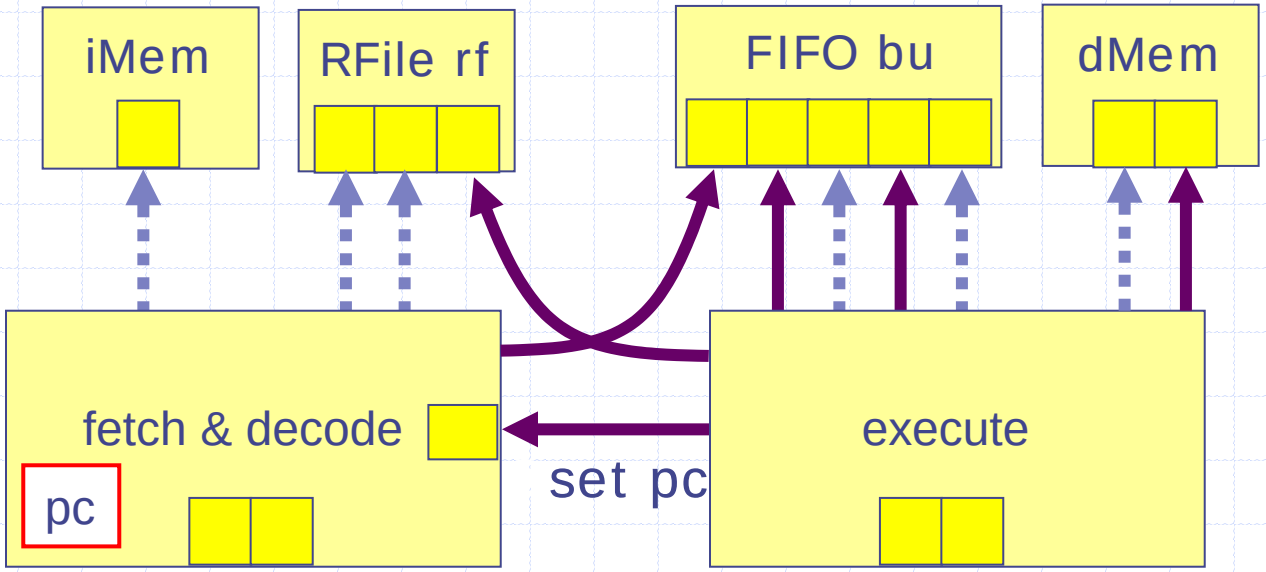


rule-atomicity ensures that pc update, and discard of pre-fetched instrs in bu, are done consistently

```
rule (match(iMem[pc], Bz{rc,addr})) &&
      !bu.find(rc) && !bu.find(addr))
  begin bu.enq (EBz{rf[rc],rf[addr]});
        pc <= pc + 1;
  end
```

```
rule (match(bu.first(),EBz{vc,va}) && (vc == 0))
  begin pc <= va; bu.clear(); end

rule (match(it, EBz) && (vc != 0))
  bu.deq;
```

Modular organization



Method calls embody both data and control (i.e., protocol)

Read method call

Action method call

Current focus@MIT: Modular compilation of large designs (e.g., Reorder buffer)

IA64 Modeling in Bluespec

CMU-Intel collaboration

- Develop an Itanium μarch model that is
  - concise and malleable
  - *executable and synthesizable*
- FPGA Prototyping
  - XC2V6000 FPGA interfaced to P6 memory bus
  - Executes binaries natively against a real PC environment (i.e., memory & I/O devices)
- An evaluation vehicle for:
  - Functionality and performance: a fast μarchitecture emulator to run real software
  - Implementation: a synthesizable description to assess feasibility, design complexity and implementation cost

Roland Wunderlich & James Hoe @ CMU

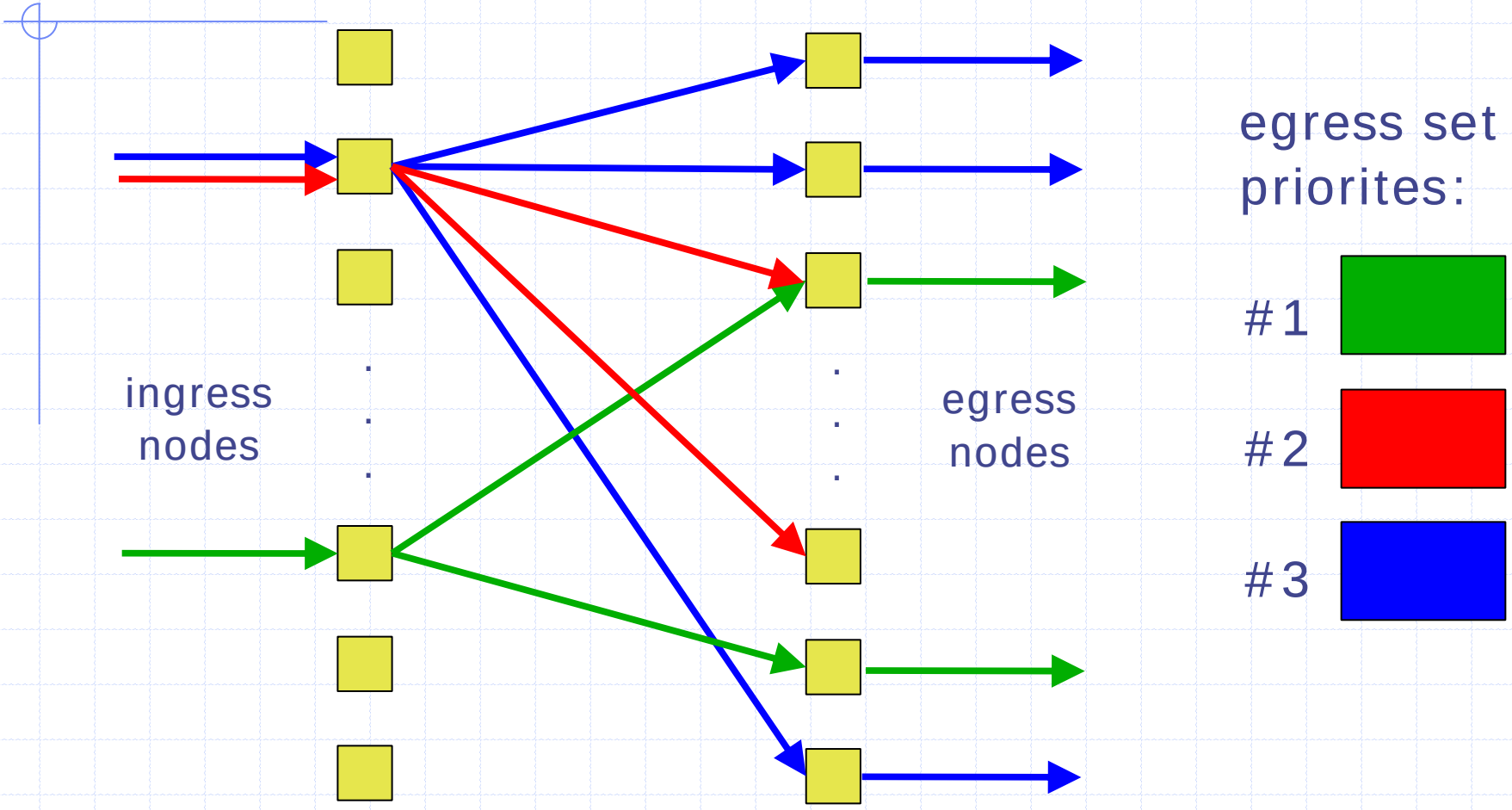
Steve Hynal(SCL) & Shih-Lien Liu(MRL)

The “Arbiter” Project @ Sandburst

- Apples-to-apples comparison
  - Re-code product chip (“arbiter”) in Bluespec
  - 2 Bluespec engineers, 2 ASIC designers; 9/1/02 to 12/31/02 (all of them on a learning curve)
  - Completed synthesis, test insertion, physical layout, timing analysis on each sub block (hard macro)

Comparison with Verilog for Multicast Arbiter	
Speed	Both met goal: 200MHz
Area	Both ~ 1.55M gates Multicast arbiter: 19% smaller, 228K gates
Lines of code	4.7K (versus 65.5K)
Verification	66% fewer bugs
Architectural Exploration	3 major alternatives explored without affecting verification

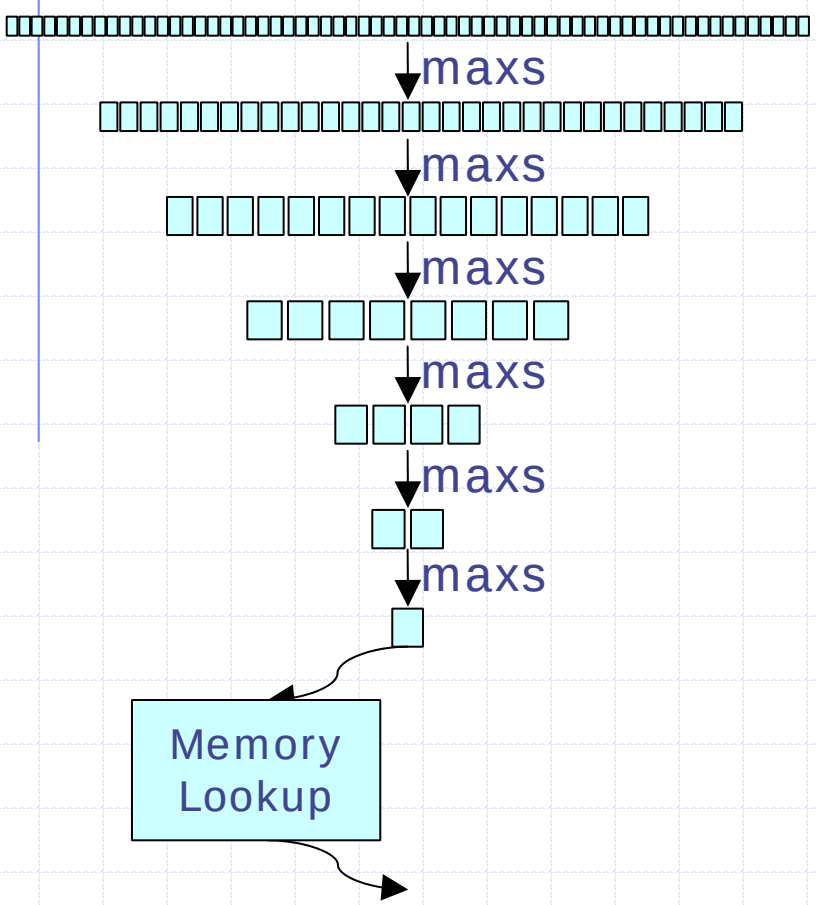
10 Gbs/line Multicast Arbiter



1. find highest-priority valid egress set for each ingress
2. remove all but highest-priority of conflicting grants

Design Problem

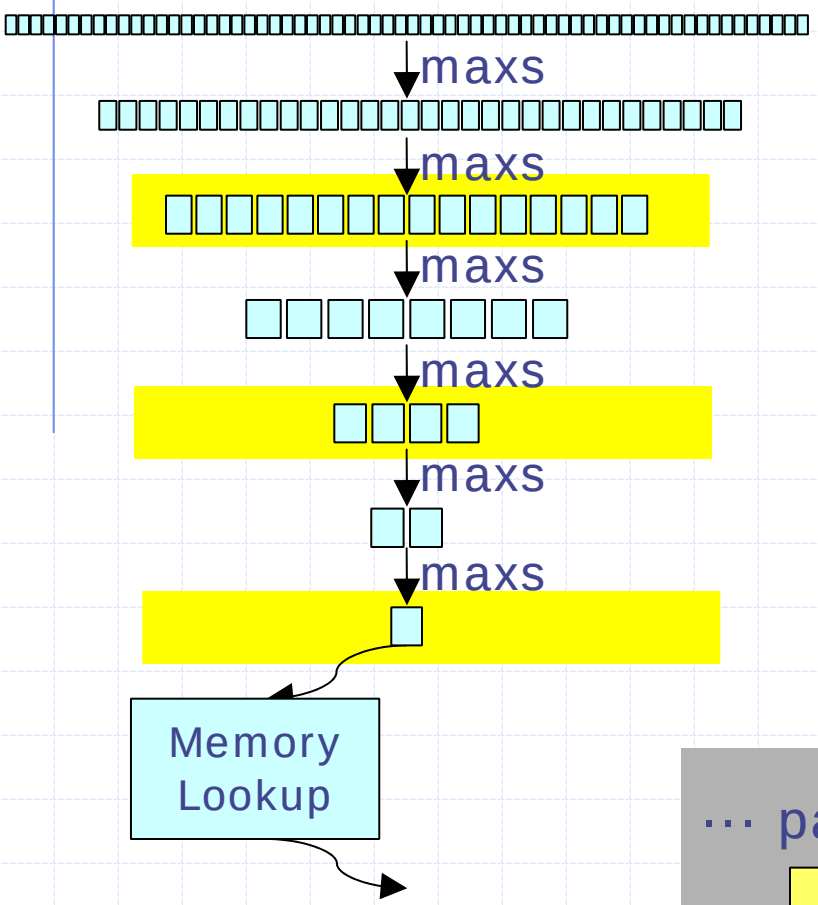
64 Items: Index of Item with Max Value




```
Push<VectorN<64,Pri>>>
indexOfMax =
      pass ( maxs ) >>
      pass ( maxs ) >>
      pass ( maxs ) >>
      pass ( maxs ) >>
      pass ( maxs ) >>
      pass ( maxs ) >>
      pass ( get_index ) >>
      tramToPush ( emapMem );
      outputMemValue;
```

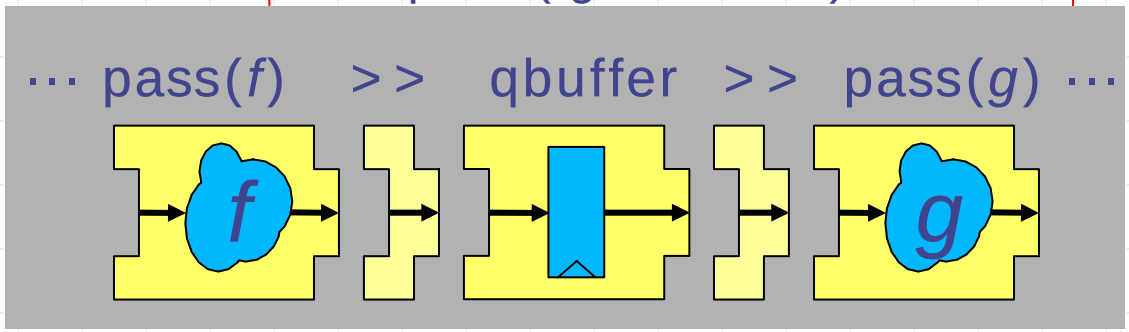
maxs: takes a list & compares priority of adjacent pairs, return list of half the size

Pipeline to meet throughput constraints



Enabled by higher-order programming

```
Push<VectorN<64,Pri>>>
indexOfMax =
      pass ( maxs ) >>
      pass ( maxs ) >>
      qbuffer >>
      pass ( maxs ) >>
      pass ( maxs ) >>
      qbuffer >>
      pass ( maxs ) >>
      pass ( maxs ) >>
      qbuffer >>
      pass ( get_index ) >>
```



Some advanced features

- Connectables
  - A class with “get” and “put” operations
  - Useful for modeling a variety of interfaces
    - direct, registered, FIFO-based, credit-based …
- Multiple clock domains
  - A class of types that are permitted to cross clock domains (e.g., some Connectables)
- Collecting and implicitly connecting some objects spread over various modules
  - e.g., PCI addressable registers

Related (non-MIT) work

- Use of atomic actions
  - to describe concurrent and distributed systems
    - Dijkstra, Hoare, Milner
    - Chandy & Misra (Unity), Lamport, $\text{Lynch}_{\text{MIT}}$ (I/O Automata)
  - in hardware verification and modeling
    - Dill (Murphi), ...
  - in hardware synthesis
    - Straunstrup (Synchronous Transactions)
    - Black & Sere (Action Systems)
- Static elaboration, embedded languages
  - Functional languages (Lisp, Scheme, Haskell, ML)
  - Modern HDLs (Verilog 2001's "generate")

Whom have I forgotten?

MIT work, 1997-2000

- TRAC compiler (TRS notation to Verilog)
  - pipelined MIPS core, superscalar, RC6 encryption, ...
  - ~35K gates, ~50MHz
  - Area & speed comparable to hand-coded Verilog

    [VLSI 99, ICCAD 00]
- Modeling and stepwise refinement
  - speculative & superscalar microarchitectures [IEEE Micro99]
  - memory models [ISCA99]
  - cache coherence protocols [ICS99]
- Verification of an implementation TRS against a reference TRS [IEEE Micro99, FME01]
- Source-to-source transformation on TRS's
  - Superscalar versions of TRS's can be derived mechanically from pipelined TRS's. [Lis MS MIT 00]

Summary

- High-level synthesis is ready for exploitation
- Key enablers (the magic)
  - Hardware model: Atomic actions on state elements
  - A two level language where all the programming sophistication is used to “generate” the hardware model via static elaboration
- Benefits
  - Dramatically reduce design time to *verified netlist*
  - More architectural exploration and more robust designs (in the same schedule)
  - Rapid evaluation of micro-architectures using FPGAs
  - Enabling IP creation, maintenance and deployment

Contributors

- MIT
  - Arvind
  - Dan Rosenband
  - Alfred Man Ng
  - Byungsub Kim
  - Nirav Dave
  - ...
- Bluespec Inc
  - R.S. Nikhil*
  - Mieszko Lis*
  - Jacob Schwartz*
  - Joe Stoy*

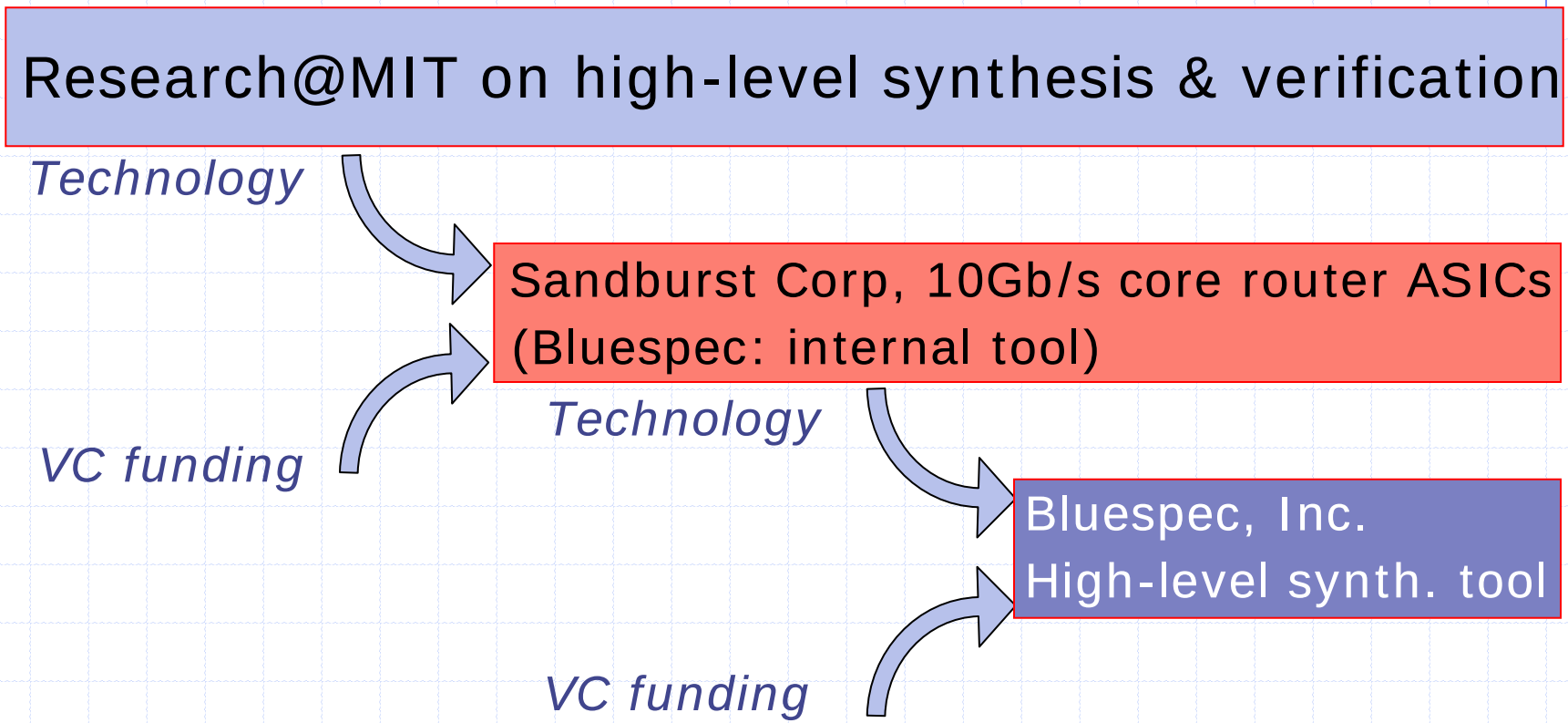
James Hoe, CMU

Lennart Augustsson, Sandburst

Xiaowei Shen, IBM

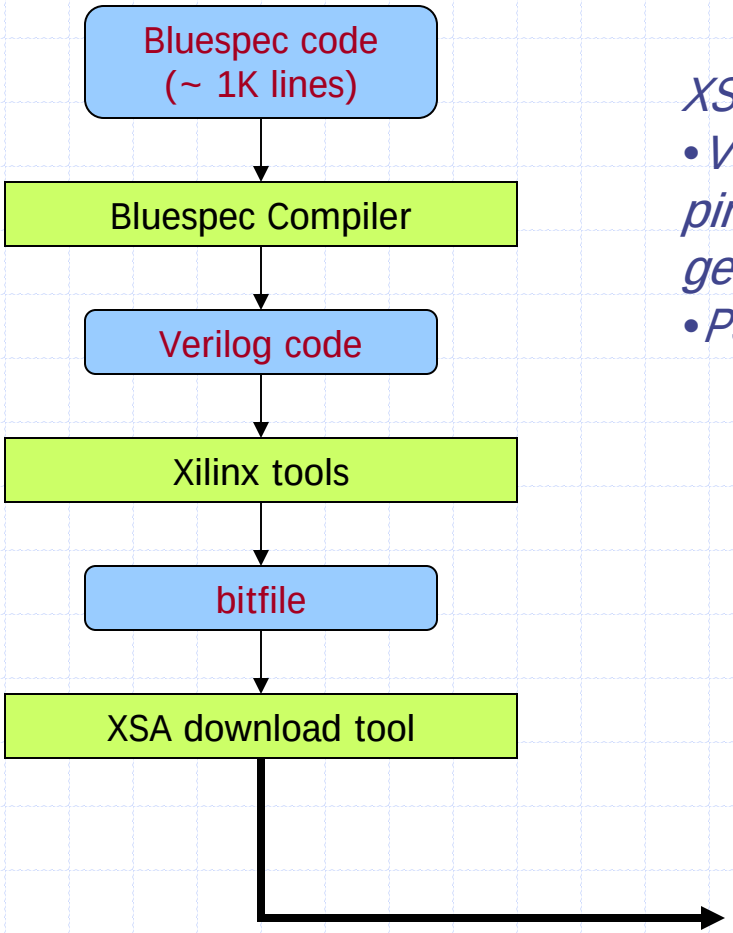
* Previously at Sandburst

Bluespec, Inc.



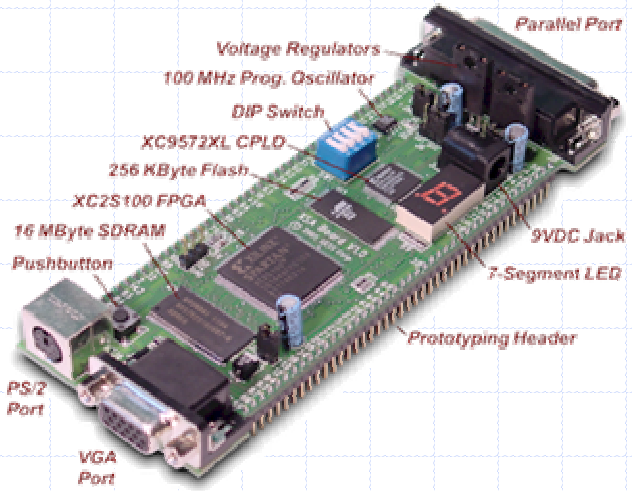
Extras

Pong in Bluespec



XSA board (w. Xilinx XC2S100 FPGA)

- *VGA socket pins connect directly to FPGA pins (no video hardware; video signal generated by Bluespec code)*
- *PS2 keyboard pins ... ditto ...*



Synthesis from Bluespec

TRS Execution Semantics

Given a set of rules and an initial term s

While (some rules are applicable to s)

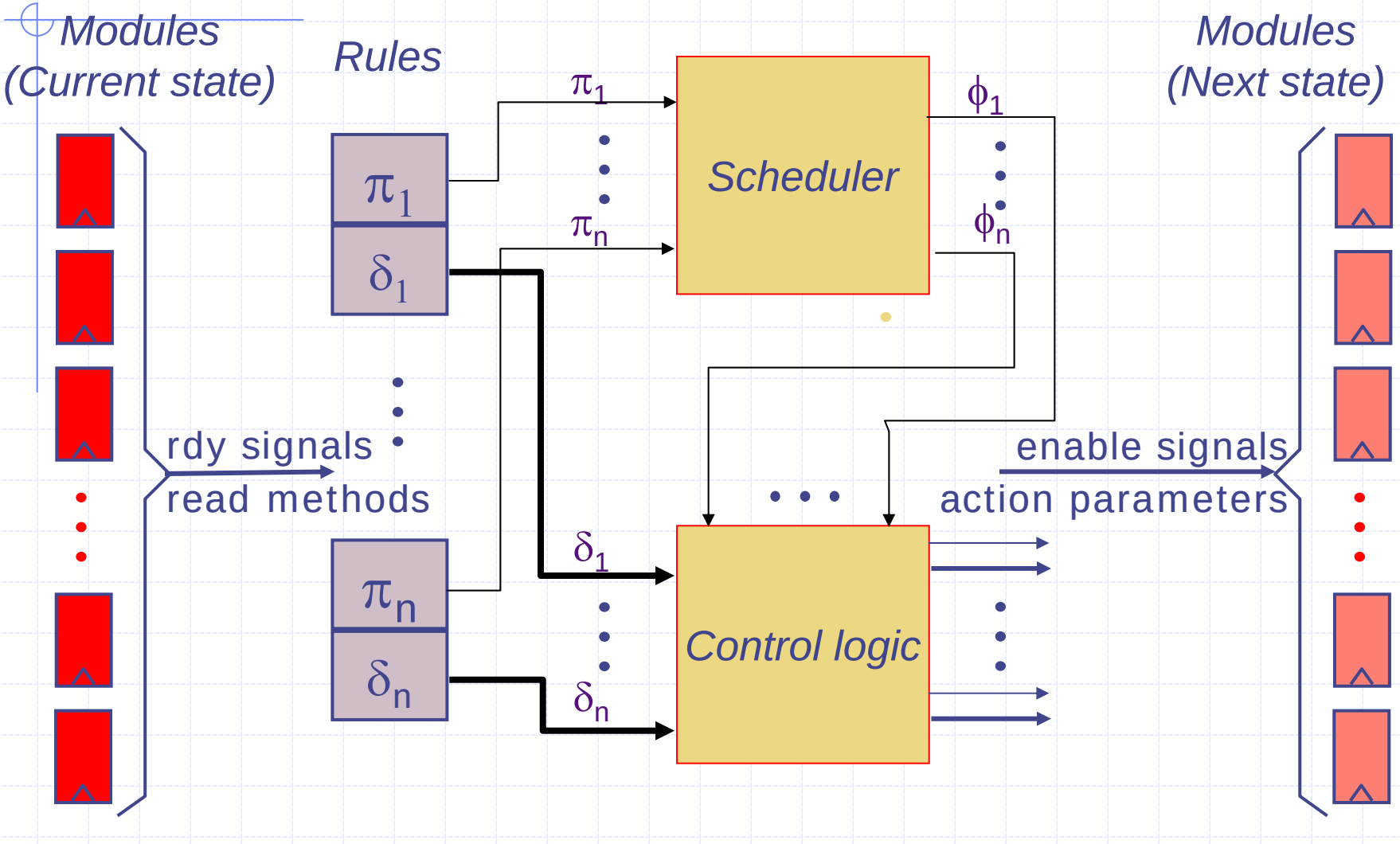
- choose an applicable rule

  (non-deterministic)

- apply the rule atomically to s

Synthesis problem: Generate a hardware scheduler that allows execution of as many enabled rules as possible at each clock without violating the semantics and generate all the associated control logic.

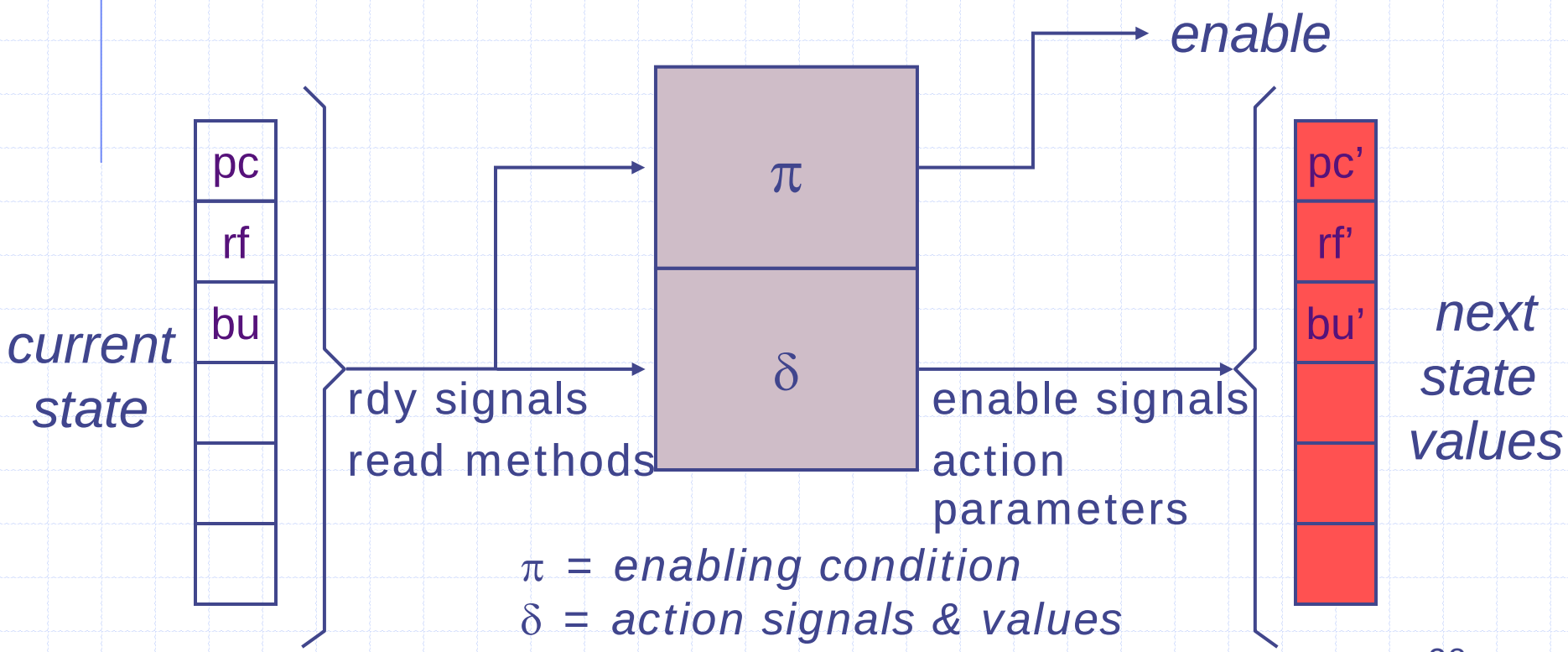
Scheduling and control logic



Compiling a Rule

"Bz Taken":

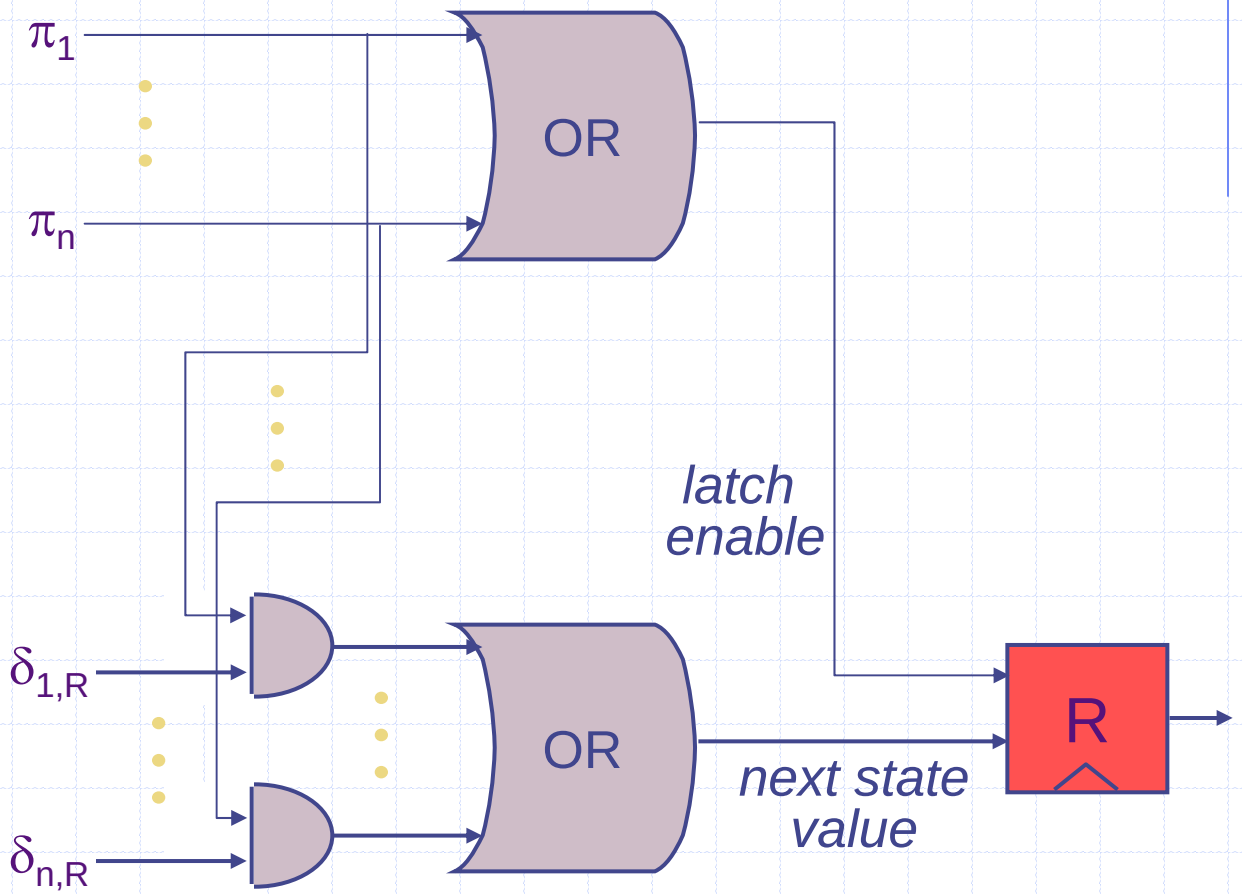
```
when (Bz rc ra) <- bu.first, rf.sub rc == 0
  ==> action pc := rf.sub ra
                   bu.clear
```



Combining State Updates: *strawman*

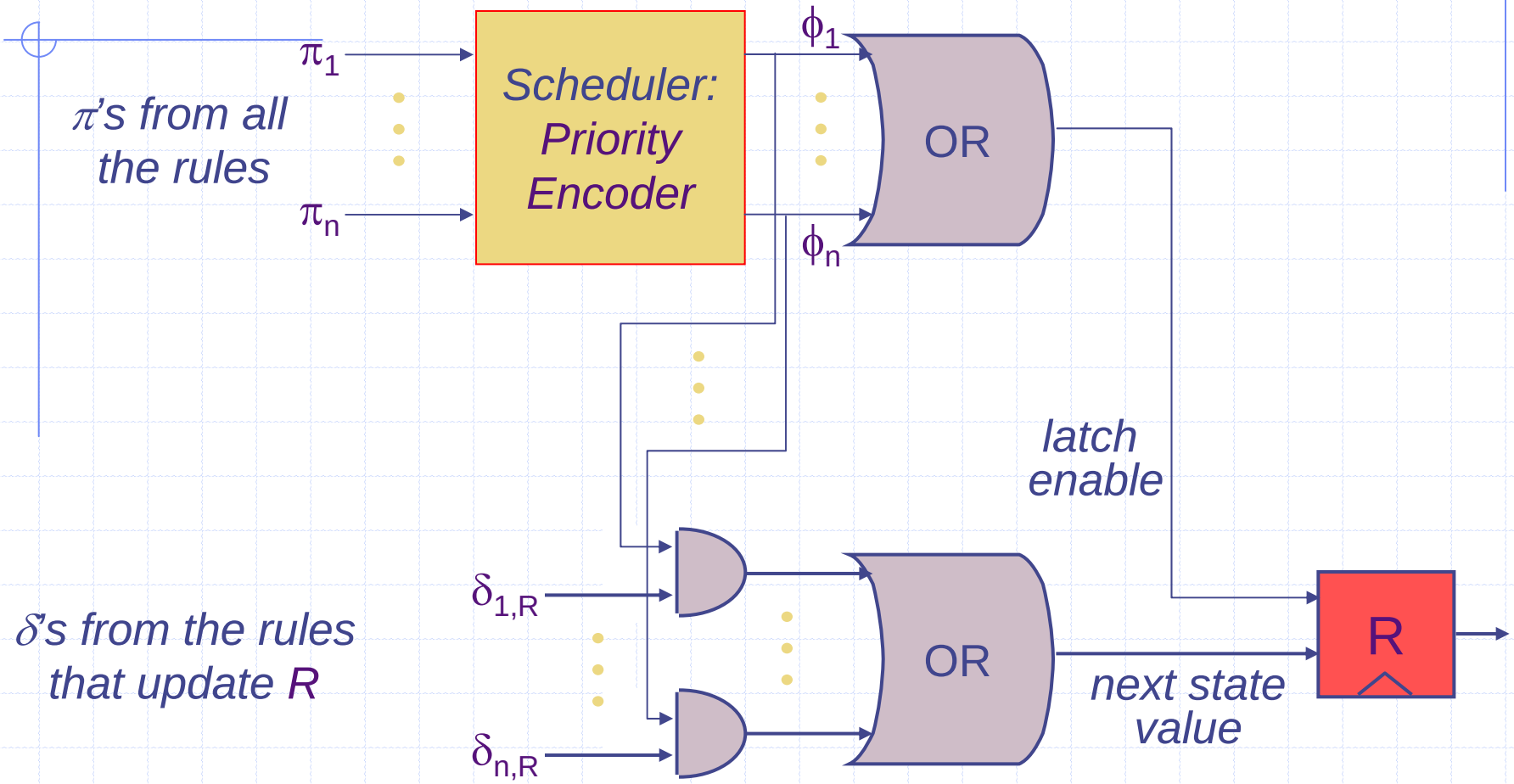
π's from the rules that update R

δ's from the rules that update R



What if more than one rule is enabled?

Combining State Updates

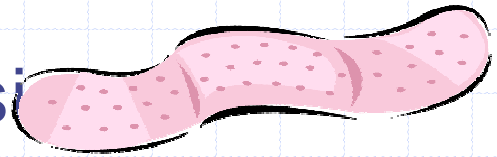


Scheduler ensures that at most one ϕ_i is true

Miscellaneous slides

Verification Solutions

- Assertions (Specman/Vera etc.) improve the productivity of the verification engineer but don't address the root cause
  - Design complexities increasi
  - Verilog/VHDL has seen no fundamental enhancements in synthesis capability in almost 15 years



Bluespec can change this.

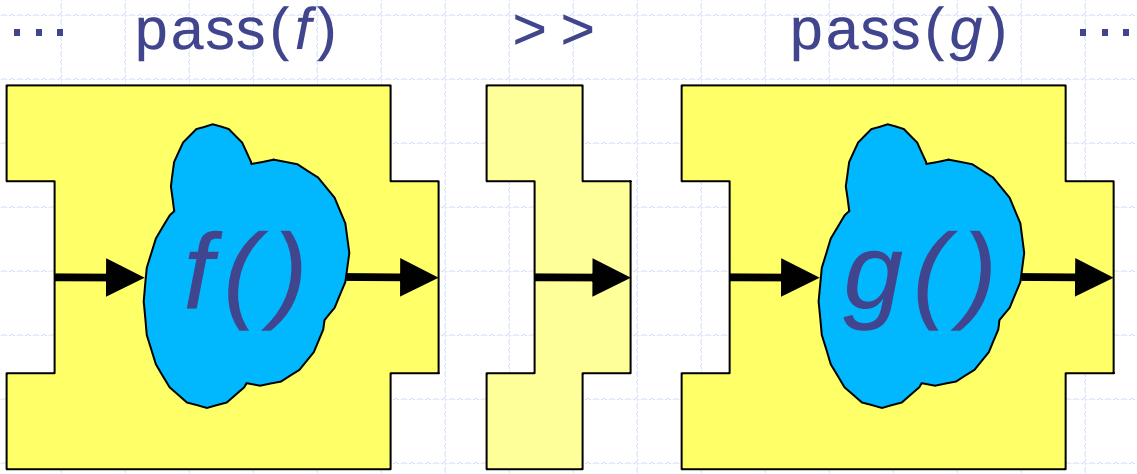
Technology validation @ Sandburst

- Modeling of Hibeam chip set
  - 12K lines of Bluespec code, accurate chip and block boundaries, accurate inter-chip & inter-block messaging
  - Used for QoS algorithm analysis and validation
  - This is *the* model of Sandburst's Hibeam chip set (there is no other C++ or SystemC model)
  - *Learning: capable of supporting large programs and models*
- Synthesis – Mesa Project 2001
  - IP Packet lookup. Subset of a Sandburst Hibeam product chip; same tools, libraries
  - ~ 8K lines Bluespec,
  - ~ 400K gates, 3mm sq die, 185 MHz
  - Included Verilog cores: on-chip memory, BIST, off-chip memory, jtag, scan, high-speed serial I/O, PCI bus, *etc.*
  - *Learning: fits comfortably in ecosystem of other tools*
- Synthesis - Re-code product chip (“arbiter”) in Bluespec 2002

Key concept: “Push” interface, and transformers

```
interface Push<a>
    Action push (a x);
endinterface
```

pass(*f*)	pass input through *f*
>>	connect two *Push* blocks



Key concept: “Push” interface, and transformers

```
interface Push<a>
    Action push (a x);
endinterface
```

pass(*f*)	pass input through *f*
>>	connect two *Push* blocks
qbuffer	buffer input in a FIFO (>> with a register)