bluespec

ESL
SYNTHESIS

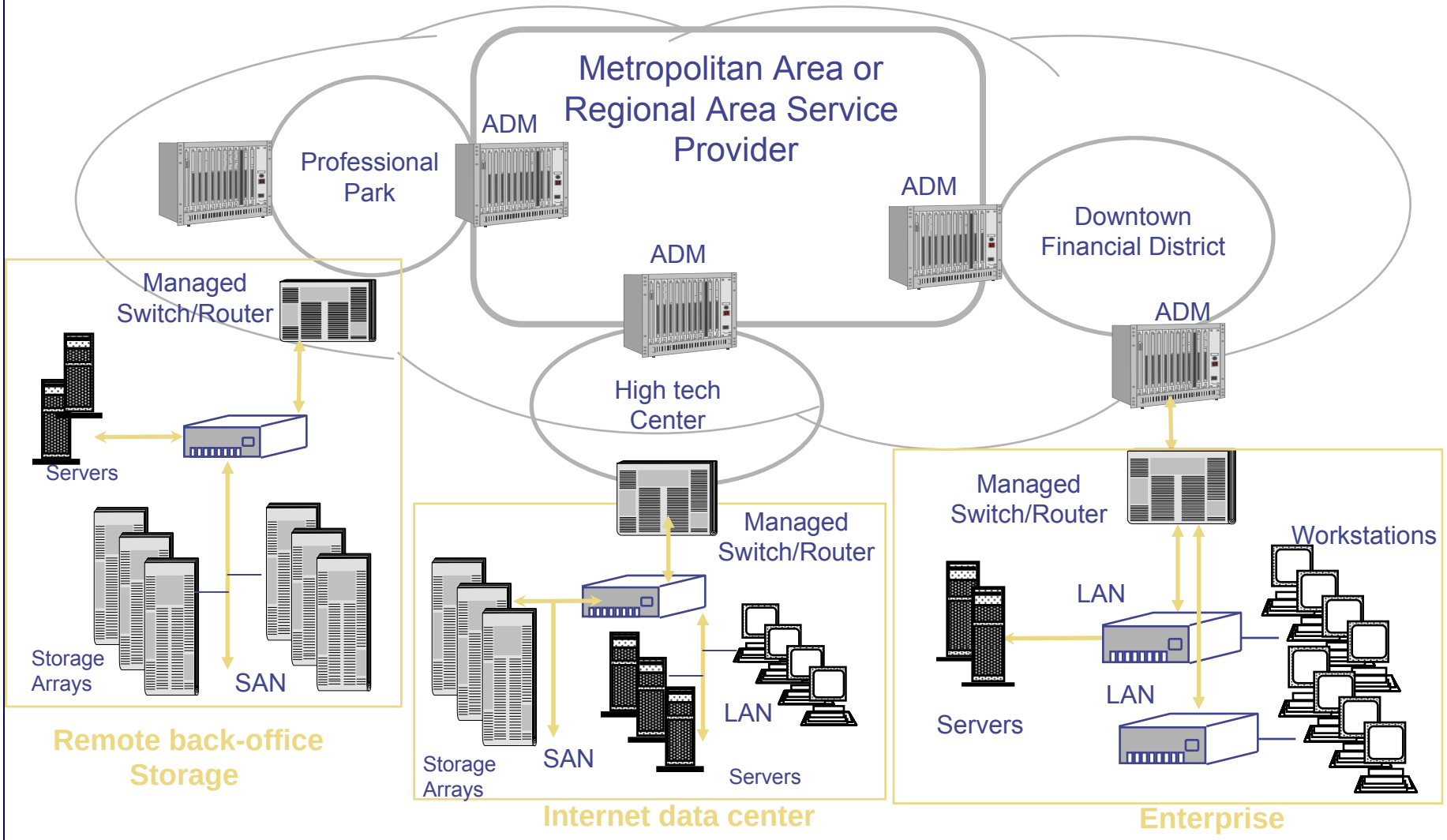
bluespec

or

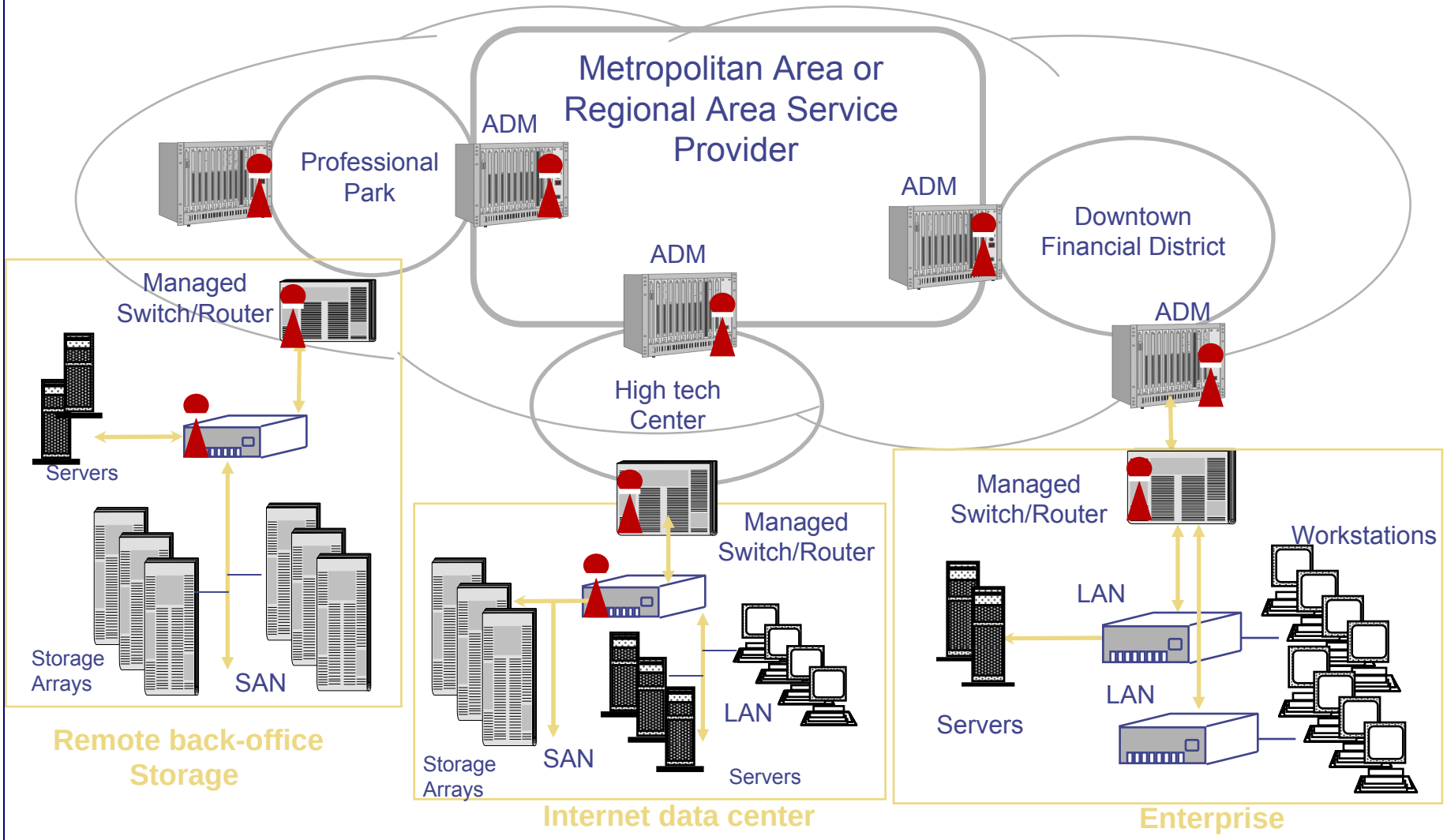
Arvind goes commercial

Joe Stoy, May 18 2007

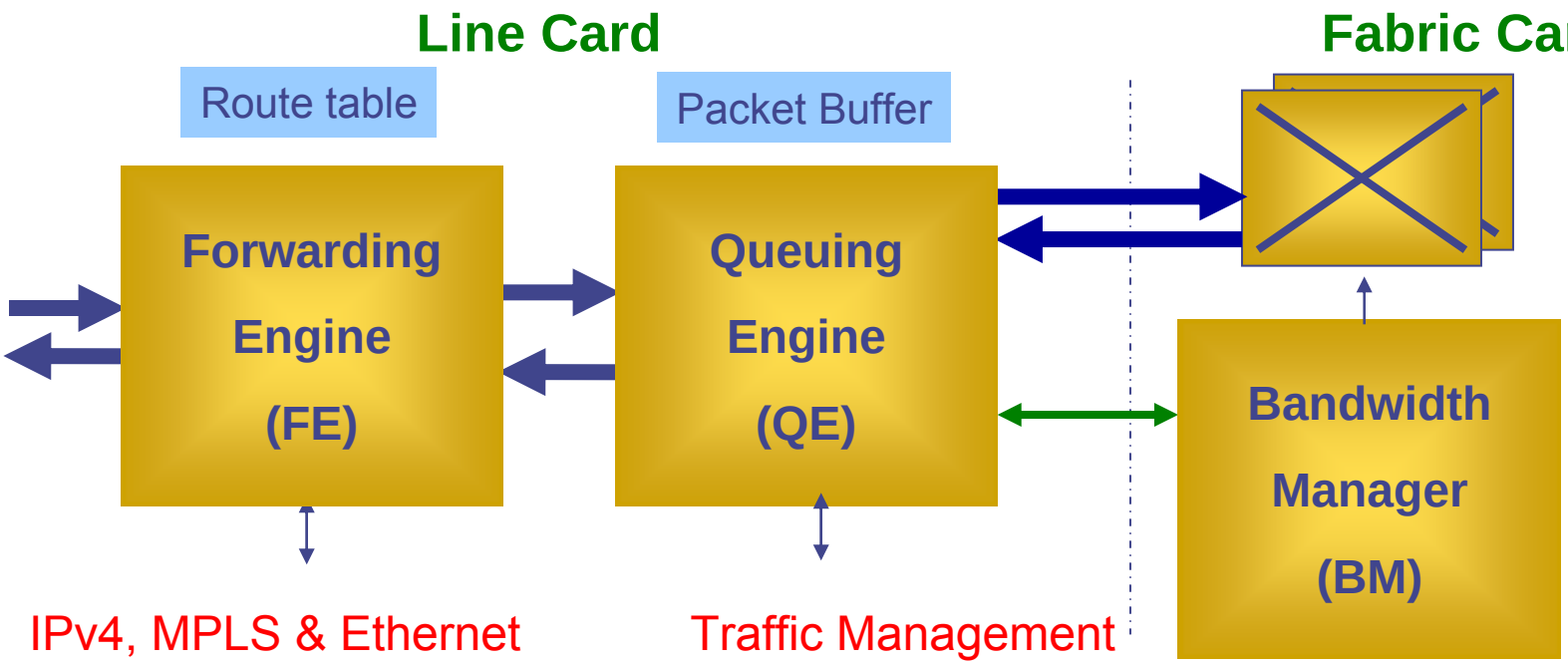
Network evolution: “IP Sprawl”



Network evolution: “IP Sprawl”



HIBEAM: Packet-Switch Architecture



IPv4, MPLS & Ethernet Packet Forwarding

- 500K IP entries
- 1M MPLS labels
- 1000 Layer 2/3/4 rules

Traffic Management

- DiffServ model
- Policing/Scheduling/Shaping

Packet Switching

- single store & forward
- 1 to 64 10G ports
- Global bandwidth control functions

Packet Processing

Traffic Mgmt & Switching

bluespec

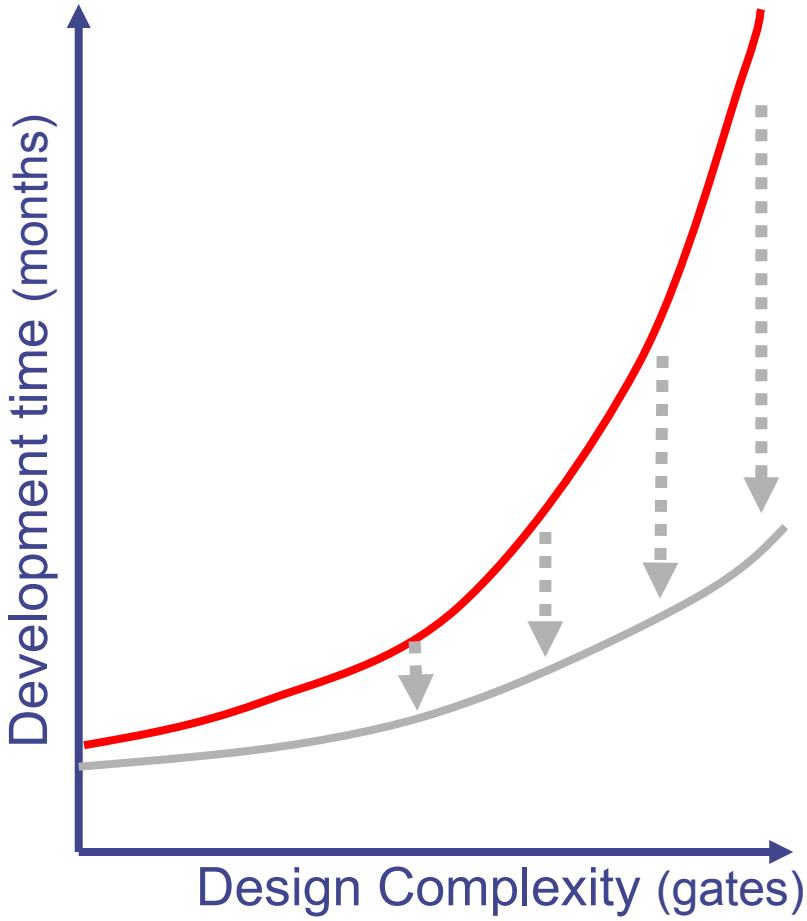
Delivering Efficient switch fabrics and building blocks for 10G IP differentiated services…

… lower cost

… higher port density

…less power

..by rewriting the rules for semiconductor development



Sandburst's revolutionary BlueSpec™ technology reduces ASIC development time by up to 50%!

bluespec

What is BlueSpec

- A language for hardware description
- Based on TRS (term rewriting systems)
- Syntax and type system based on Haskell
- Run-time execution model quite unlike Haskell's
- Compiled to structural Verilog
- . . . or C
- . . . or FPGAs.

Bluespec at Sandburst

- Lennart Augustsson designed the Bluespec language
  - Notation, type system, and static abstraction mechanisms borrow heavily from Haskell
- Designed and implemented the compiler
  - v1 circa 7/2000, v2 circa 9/2000, v3 11/2001
  - Mieszko Lis implemented the scheduler
  - Initially produced just Verilog
    - Process/execute with std. Verilog tools
  - Later (10/2000) also produced C
    - C code is "cycle accurate" to the Verilog
- Can mix Bluespec-generated code with other code (hand-written Verilog, legacy Verilog, imported IP, ...)

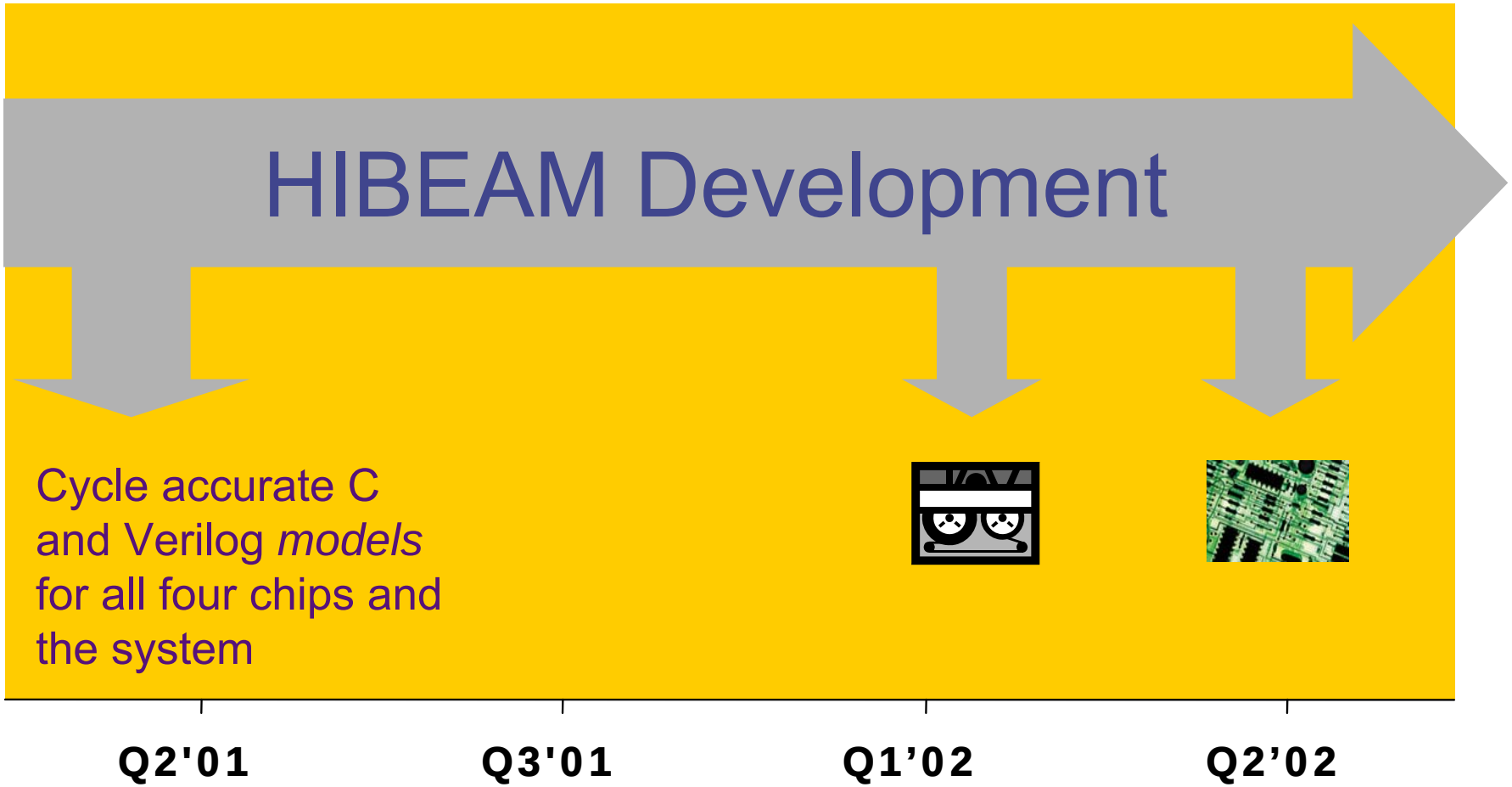
The Bluespec team

- Rishiyur Nikhil, Director (DEC/Compaq Research, MIT)
- Lennart Augustsson (CRT, Chalmers)
- Stephen Bailey (DEC, startups)
- Joe Stoy (Oxford)
- Mieszko Lis (MIT)
- Jacob Schwartz (MIT)
- Dan Rosenband (MIT)

plus Arvind

- Consultants
  - Professor James Hoe (CMU)
  - Professor Krste Asanovic (MIT)
  - Professor Srini Devadas (MIT)
  - Niklas Rojemo (Sweden)

BlueSpec™ Advantage:

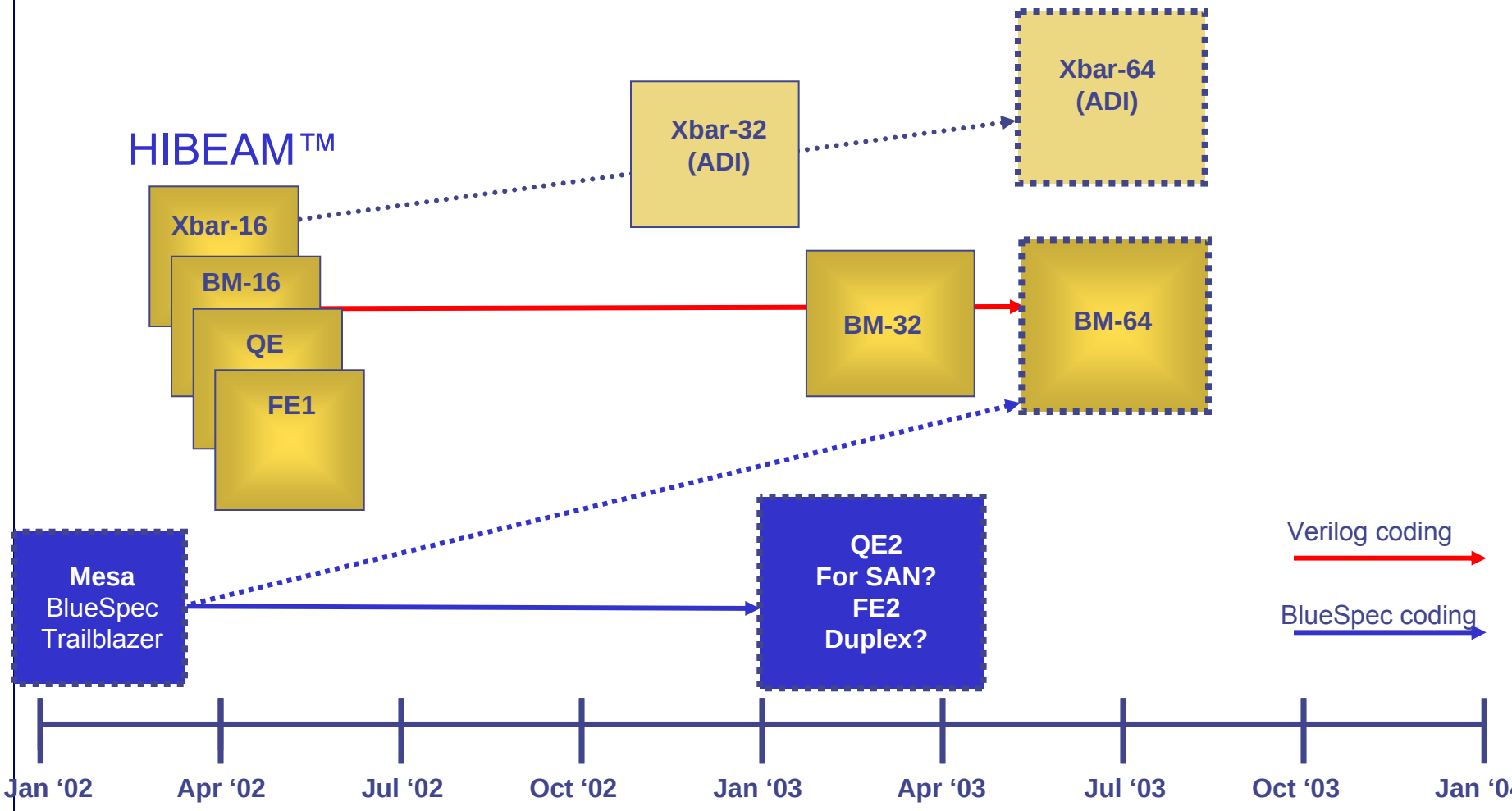


Model generation enables early customer engagements

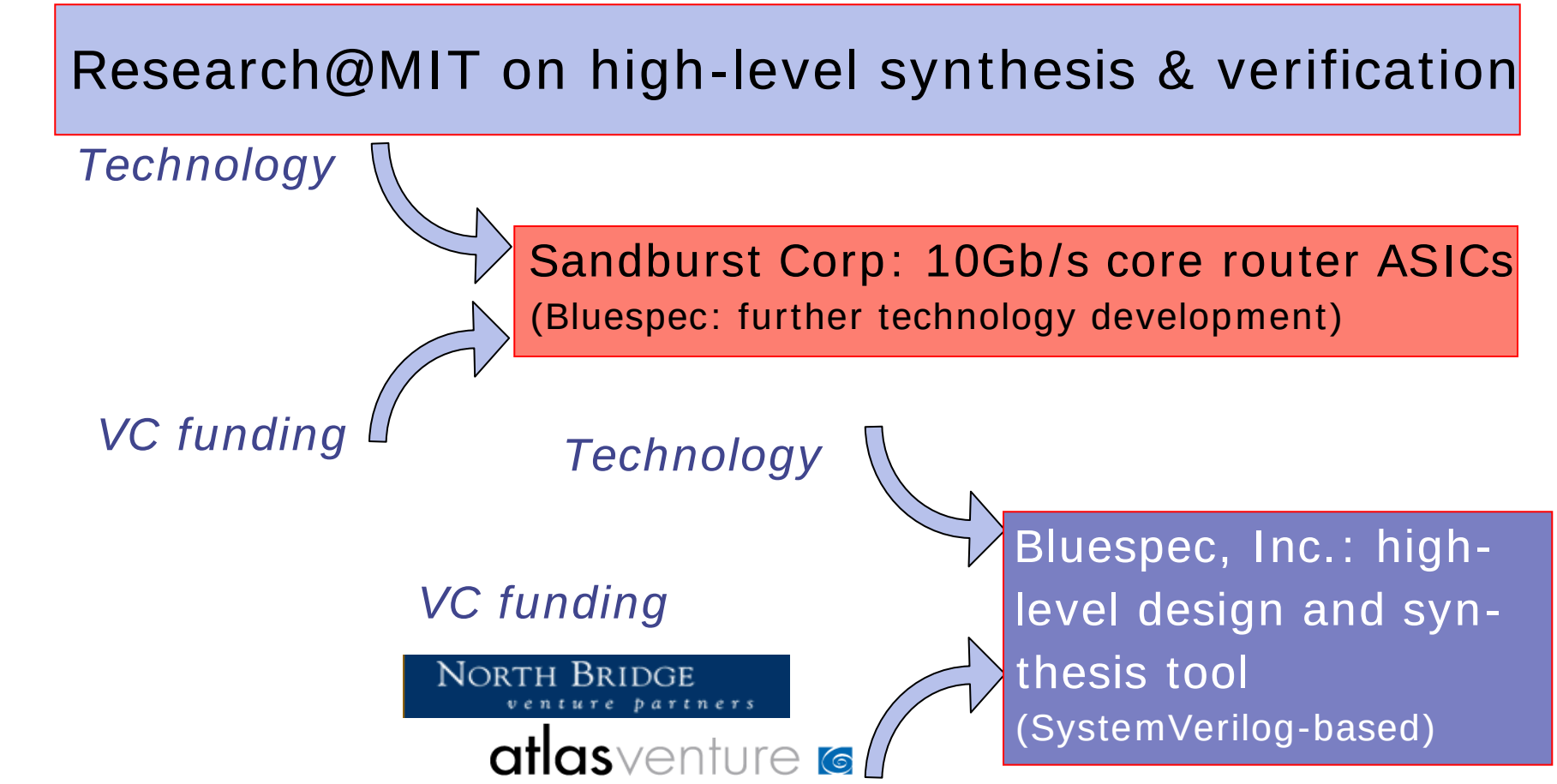
Bluespec use at Sandburst

- HIBEAM chip set system model for performance analysis
  - close to 15K lines of Bluespec
  - April 2001 through *the present*
- Mesa "pathfinding" project
  - Goal: flesh out the Bluespec "design flow"
  - Understand how Bluespec design fits into the larger picture of the full ASIC design process
  - 6/2001 through 11/2001

Development Roadmap



Bluespec, Inc. background



Bluespec has designs on saving billions for semiconductor industry

New company launches with $4 million in funding

By MARK PICKERING
INDIA New England Staff

WALTHAM, Mass. — Two local entrepreneurial luminaries have co-founded a new start-up, Bluespec, whose software is being used to speed up the microchip-design process.

Arvind

"I'm really quite delighted to have it going," said Arvind, a Bluespec co-founder and board member.

Arvind, a Massachusetts Institute of Technology professor, is best known for founding Andover-based Sandburst Corp., a semiconductor company backed by $36 million in venture capital.

"Arvind's research has come up with a key breakthrough" in how chips are designed, said Shiv Tasker, Bluespec co-founder and chief executive officer.

Tasker has more than 15 years of experience in the chip-design industry and, most recently, was CEO of Waltham-based PhaseForward, a health-related software company.

Tasker

Working together on Bluespec, the two have landed $4 million in venture capital from Atlas Venture and Waltham-based North Bridge Venture Partners.

There is a major opportunity for companies such as Bluespec "to save the semiconductor industry billions of dollars," said Rita Glover, principal analyst at EDA Today, which covers electronic-design automation.

Bluespec has 14 full-time employees, mostly in engineering, said Tasker. Now, efforts are moving beyond the research phase.

"The technology is very robust," said Tasker. "We're looking to ship products in the first quarter of 2004."

The process of starting Bluespec actually began last January, said Arvind, who goes by just the one name. At that point, Arvind had stepped down from his CEO role at Sandburst and was back teaching computer science at MIT.

Meanwhile, he had begun to see real potential in research he had been doing first at MIT and then at Sandburst, which owned all the related intellectual property, he said.

Ultimately, the company's board decided the software being developed now by Bluespec was not a good fit for Sandburst, a chip company.

Having separate businesses, Arvind said, also seemed a better strategy for appealing to venture capitalists.

"You don't want to confuse investors," he said.

Arvind stepped aside from voting on that matter, given that he could benefit from the decision being made, he said. Sandburst, where Arvind remains a board member, now has a minority stake in Bluespec.

Mutual friends then introduced Arvind and Tasker.

"The technology looked intriguing, so I wrote a business plan," said Tasker. "We had to assess what it would take to build something as a commercial product and to identify whether the market size was large enough to interest venture capitalists."

In fact, Tasker said, "the market is clamoring for solutions to design problems." It is increasingly difficult to design chips, because of the drive to put more and more complex electronics onto the same semiconductors.

The result, Tasker said, is that "the ability to manufacture chips has vastly outstripped the speed at which they are designed."

Working with the venture capitalists, Tasker and Arvind formed the company in June and gained funding in July.

Asked about his return to teaching after being Sandburst's CEO, Arvind said it suits him fine.

"My leave was for two years from MIT," Arvind said.

As for Bluespec, Arvind said, "I'm there one day a week, but there in spirit even more."

Asked if he thought of being a CEO there, Arvind insists genially: "I'm a college professor."

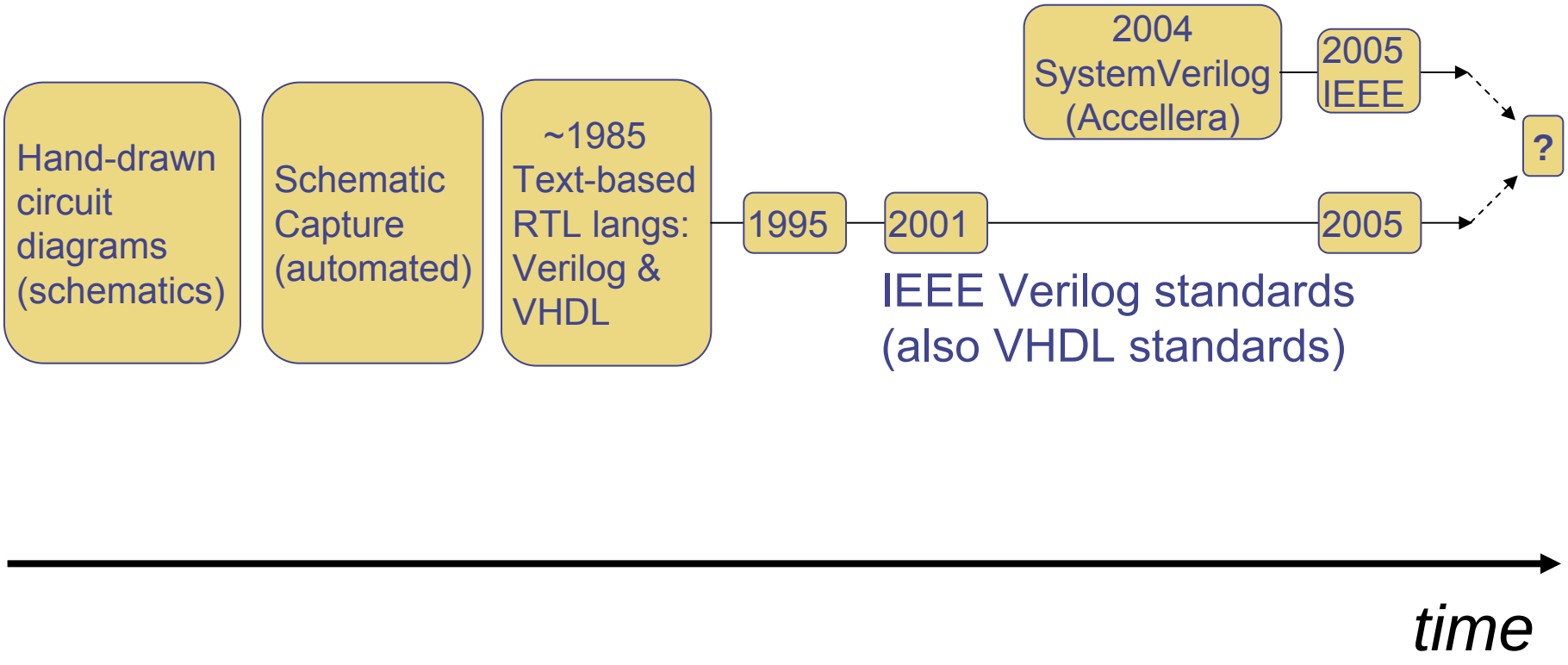
bluespec

The Bluespec team

- Rishiyur Nikhil, Director (DEC/Compaq Research, MIT)
- Lennart Augustsson (CRT, Chalmers)
- Stephen Bailey (DEC, startups)
- Joe Stoy (Oxford)
- Mieszko Lis (MIT)
- Jacob Schwartz (MIT)
- Dan Rosenband (MIT)

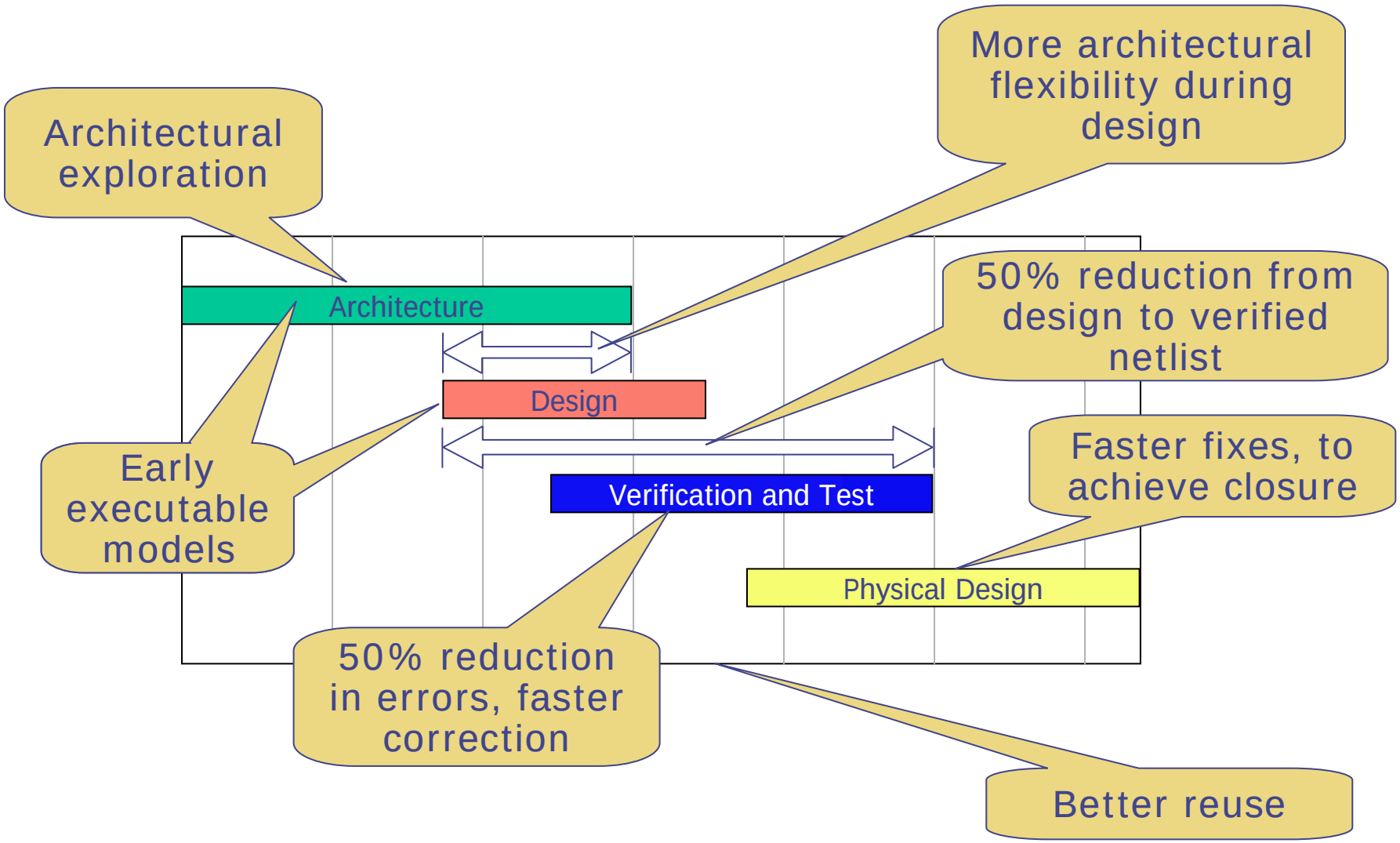
***plus* Arvind**

Evolution of HDLs (Hardware Description Languages)



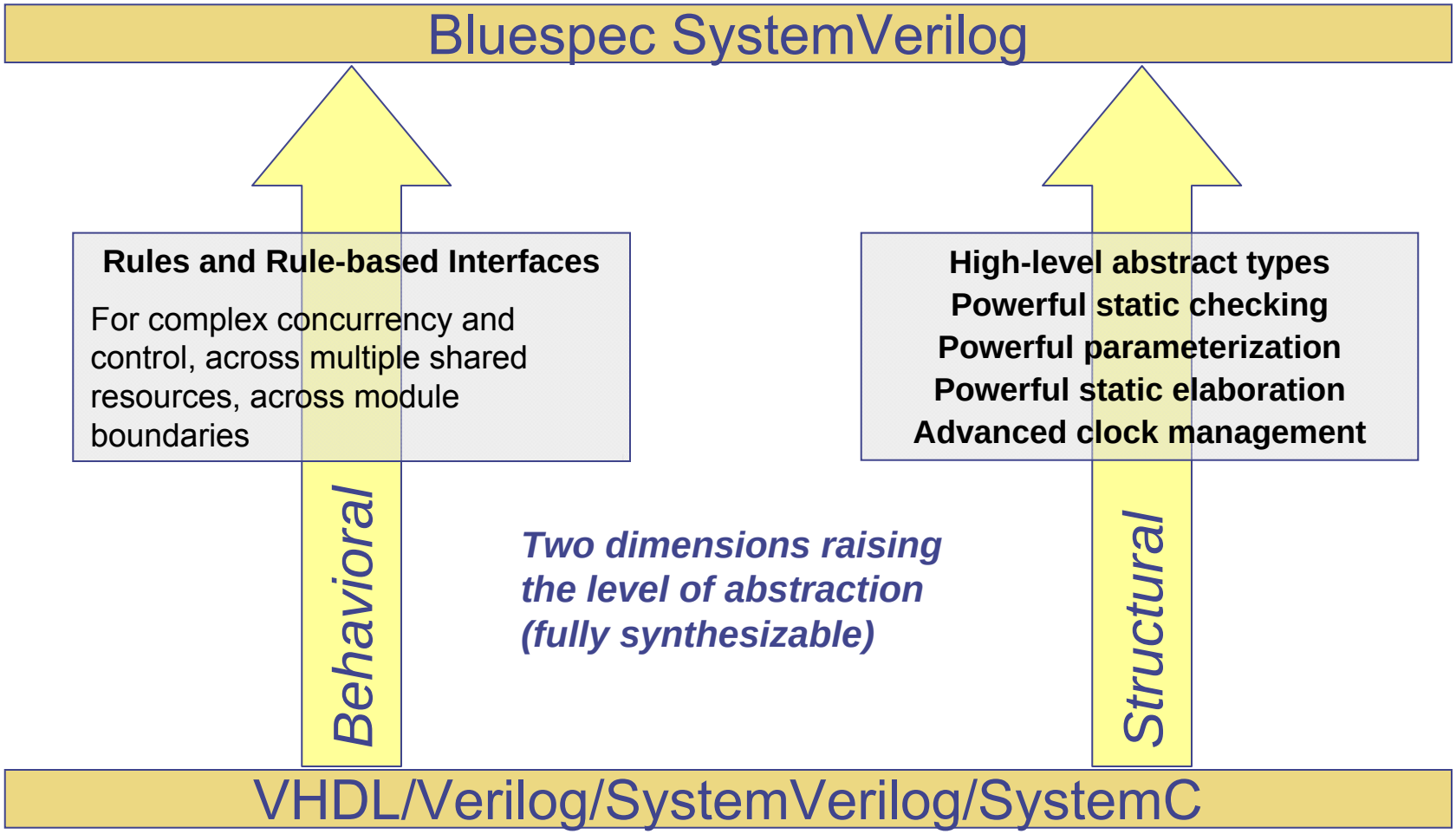
(RTL = Register-Transfer Level)

Bluespec: Better Design Accelerates Everything!

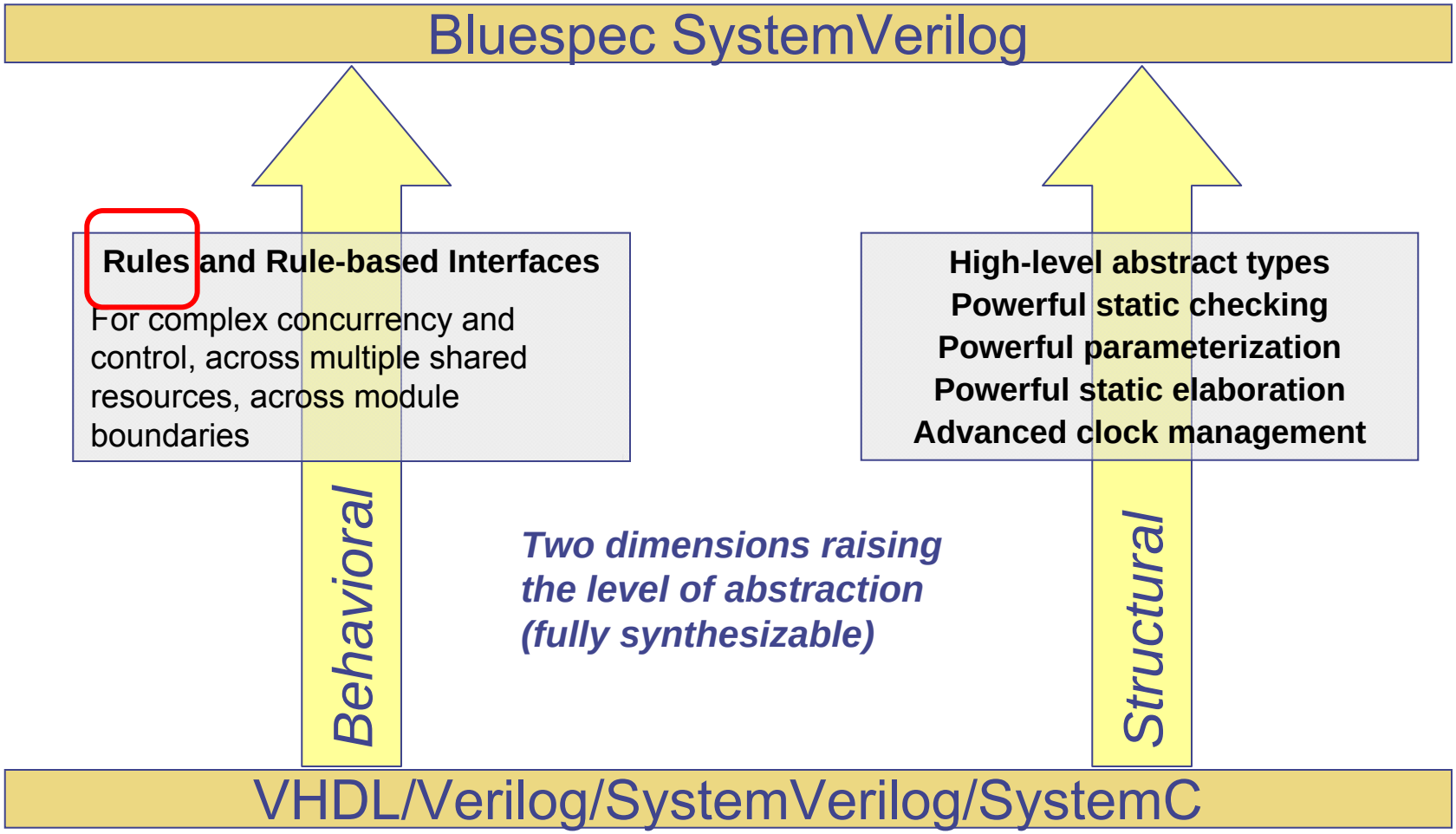


Fully synthesizable – without compromise!

Bluespec SystemVerilog™
A one slide overview



Bluespec SystemVerilog™ A one-slide overview



One-at-a-time intuitions

bluespec

- Untimed rule semantics are one-rule-at-a-time steps
- The Bluespec compiler schedules multiple rule steps into each clock, producing a timed semantics and resolving non-determinism
- But before we go there, let's build some HW intuitions based on one-rule-at-a-time
- We'll use the following example
  - (Can you guess what it computes? Hint: "Euclid")

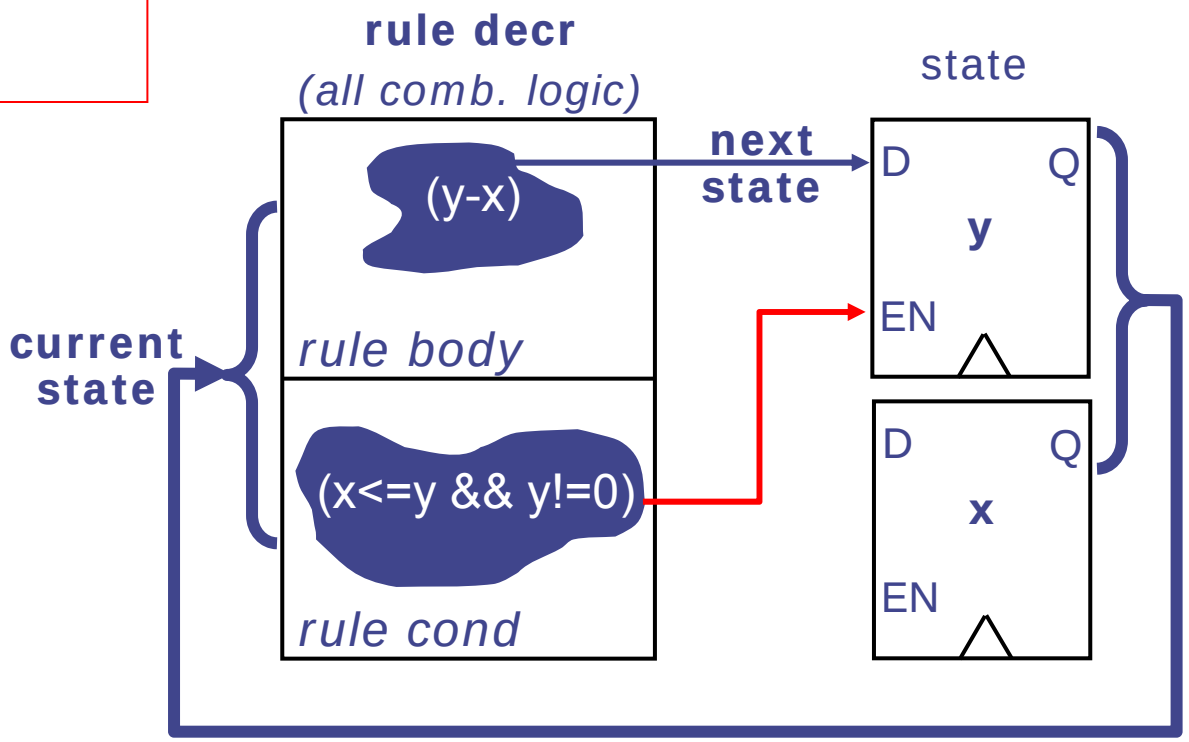
```
rule decr ( x <= y && y != 0 );
    y <= y - x;
endrule : decr

rule swap (x > y && y != 0);
    x <= y; y <= x;
endrule: swap
```

Answer: Euclid's algorithm for computing GCD (Greatest Common Divisor) of initial values in x and y registers; result is in x

Suppose we want to execute just one rule

- The HW would be quite simple

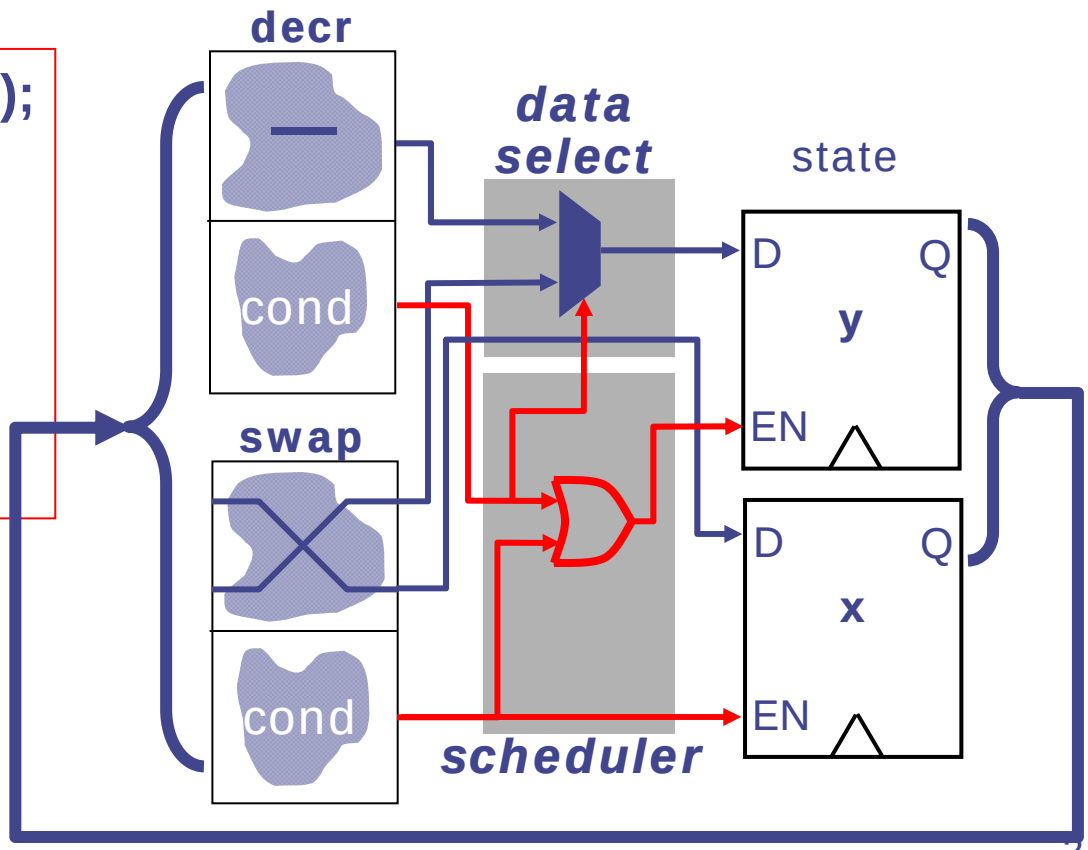
```
rule decr ( x <= y && y != 0 );
    y <= y - x;
endrule : decr
```



Two mutually exclusive rules

- (Later: what to do if they're not mutually exclusive)

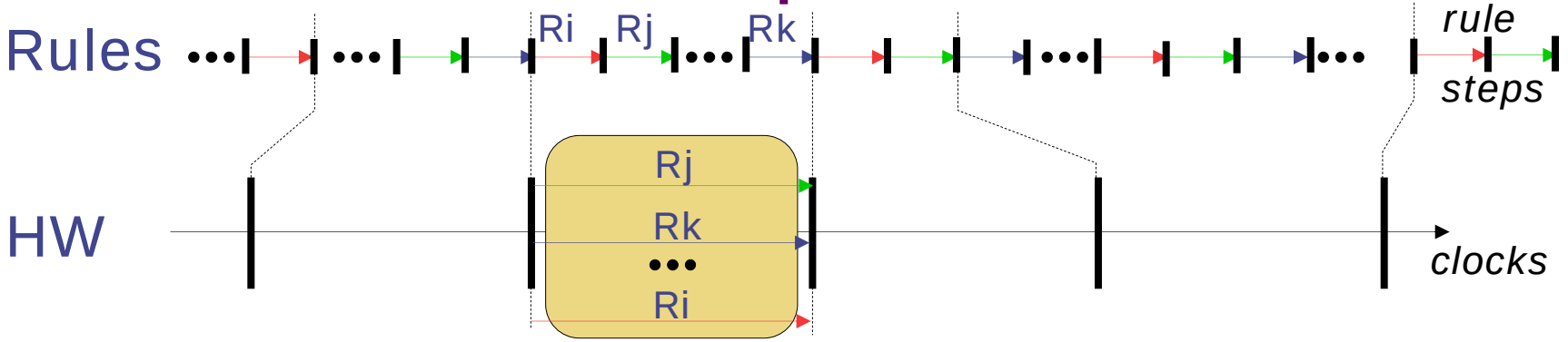
```
rule decr ( x <= y && y != 0 );
    y <= y - x;
endrule : decr

rule swap (x > y && y != 0);
    x <= y; y <= x;
endrule: swap
```



Practical rule composition

- In practice, we only do restricted rule compositions
- Every rule executes entirely within one cycle
- A rule fires at most once in a cycle (no Rule1<Rule1)
- Greedy: as many rules as possible per clock cycle
- Rule1 body does not feed into Rule2 cond
  - Therefore, rule conds can be evaluated based on state at beginning of cycle
- Rule1 body does not feed into Rule2 body
  - Therefore, Rule2 can use state from beginning of cycle

Practical Rule Composition



- No intra-cycle communication between rules, in the HW
- Correctness: HW scheduler only allows rules Ri, Rj, …, Rk to execute concurrently when *net state change* is equivalent to Ri<Rj<…<Rk
- Thus, never get into inconsistent state (no race conditions)
  - Every state in the HW exists in the one-rule-at-a-time semantics

Multiple Rules and Conflicts

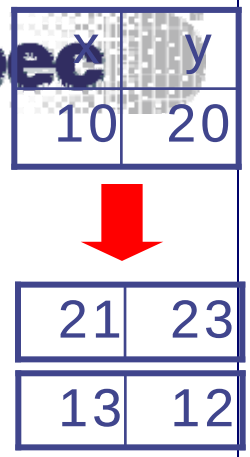
- If two rules are enabled in a particular cycle, what prevents them from executing concurrently?
- Answer: *conflicts*
  - Since state read/write ordering is different in rule-at-a-time semantics compared to concurrent execution, certain concurrent executions would result in inconsistent states
  - Certain *resource sharings* prevent concurrency

Conflict

- Rule untimed semantics:
  - r1 and then r2 (“r1<r2”), or
  - r2 and then r1 (“r2<r1”)

```
rule r1 (c1);
   x <= y + 1;
endrule

rule r2 (c2);
   y <= x + 2;
endrule
```

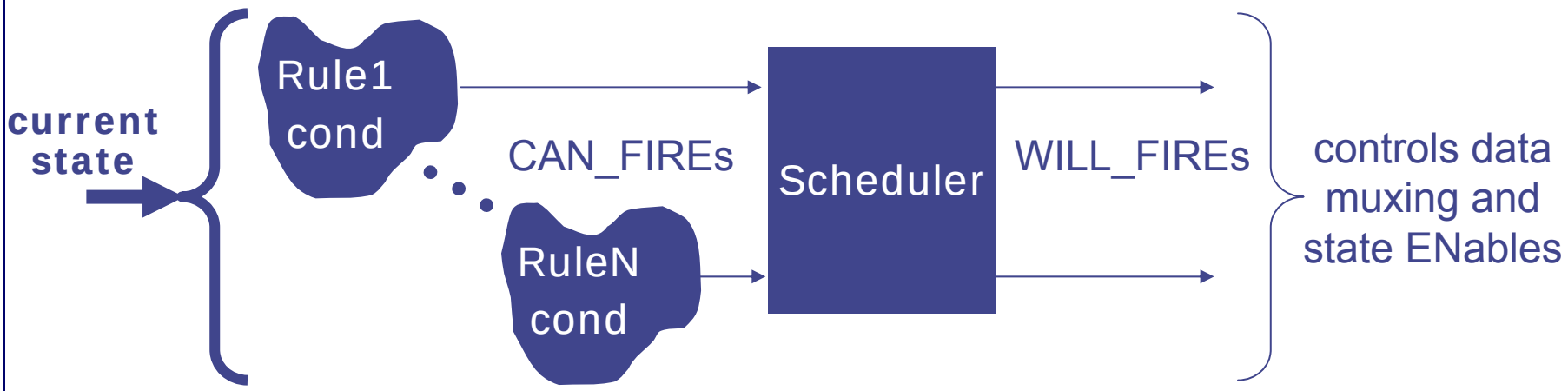
- HW: concurrent execution of r1 and r2
  - Each rule reads state (y,x) at start of cycle
  - Each rule updates state (y+1, x+2) at end of cycle



Net HW state change ≠ any order (r1<r2 or r2<r1)
➔ *not ok* to execute concurrently

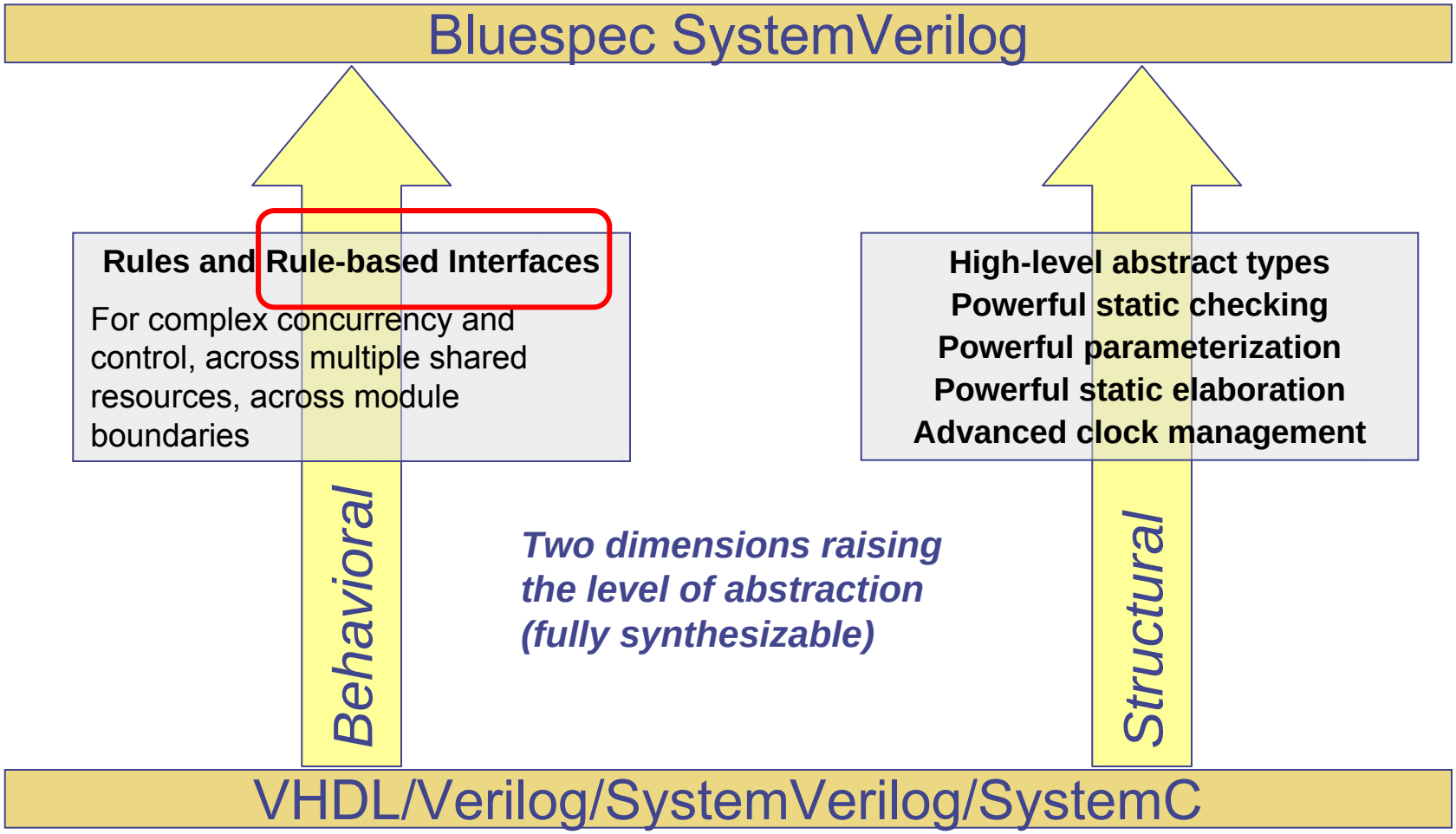
A rule is not a Verilog “always” block!
Bluespec HW scheduler will prevent these firing together

CAN_FIRE and WILL_FIRE

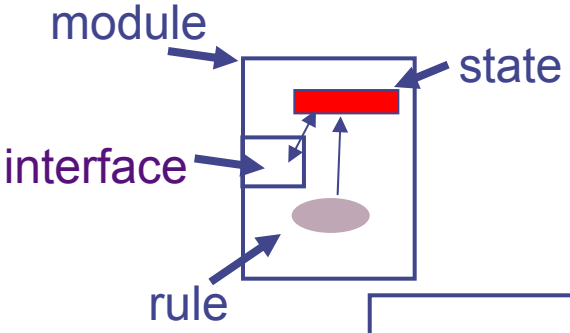


- Scheduler incorporates conflict analysis by compiler
  - CAN_FIRE is False ➔ WILL_FIRE is False
  - CAN_FIRE is True ➔ WILL_FIRE is
    - True if not precluded by conflicts with other rules
    - False otherwise (the rule is blocked for this cycle)

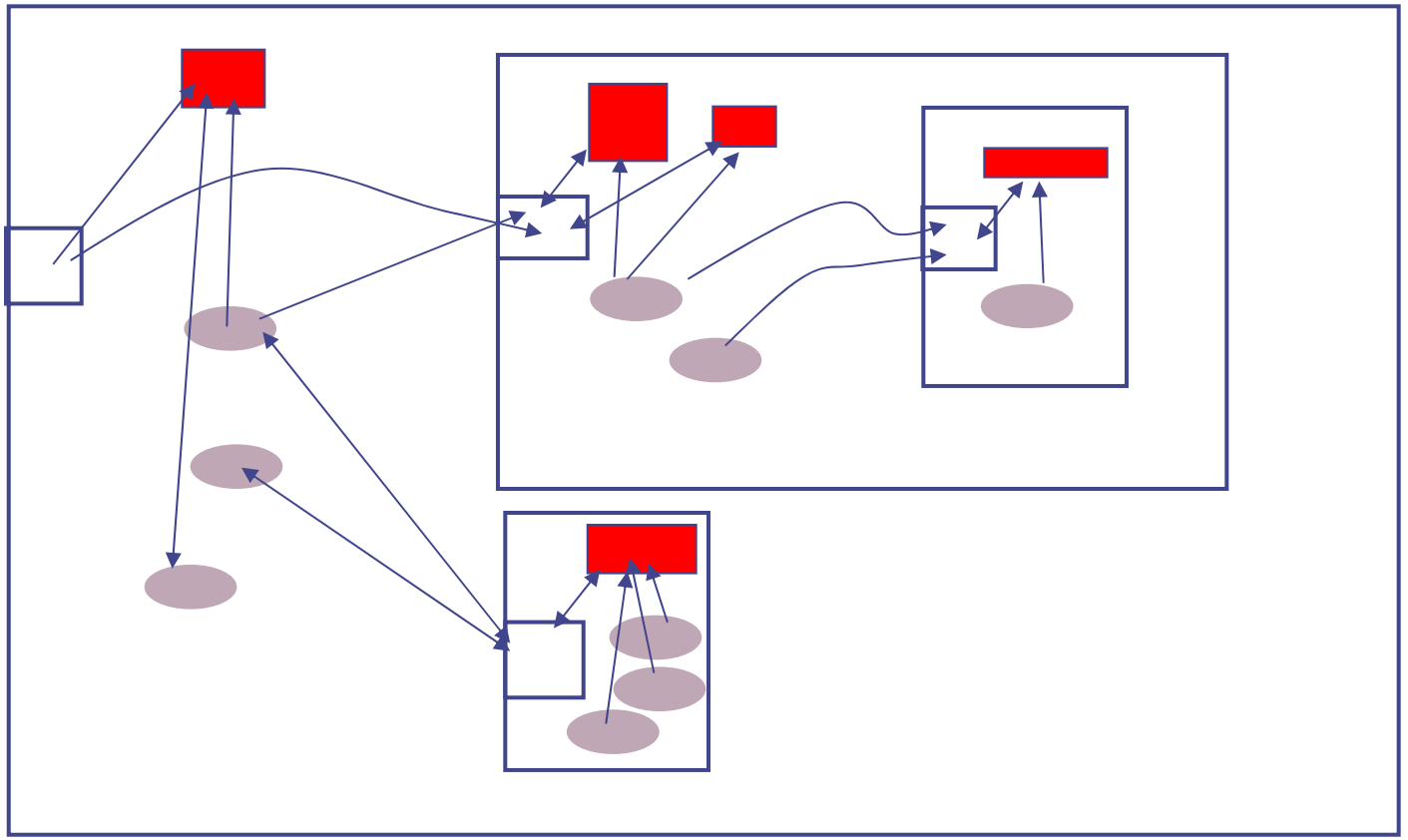
Bluespec SystemVerilog™ A one-slide overview



Modules, rules, interfaces, methods



The big picture: modules contain rules which use methods that are provided by sub-modules in their interfaces. Methods, too, can use other methods.



Multiplier Example

```
module mkTest ();

  Reg#(int) state <- mkReg(0);
  Mult_ifc  m       <- mkMult1();

  rule go (state == 0);
    m.start (9, 5);
    state <= 1;
  endrule

  rule finish (state == 1);
    $display ("Product = %d",m.result());
    state <= 2;
  endrule

endmodule: mkTest
```

```
interface Mult_ifc;
  method Action start (Tin x, Tin y);
  method Tout   result ();
endinterface: Mult_ifc
```

```
module mkMult1 (Mult_ifc);
  Reg#(Tout) product <- mkReg(0);
  Reg#(Tout) d       <- mkReg(0);
  Reg#(Tin)  r       <- mkReg(0);

  rule cycle (r != 0);
      if (r[0] == 1) product <= product + d;
      d <= d << 1;
       r <= r >> 1;
  endrule

  method Action start (x,y) if (r == 0);
       d <= x; r <= y; product <= 0;
  endmethod

  method result () if (r == 0);
       return product;
  endmethod

endmodule: mkMult1
```

Multiplier Example

```
mkTest :: Module Empty
mkTest =
  module
    state ::   Reg int
    state <-  mkReg 0

    m ::  Mult_ifc
    m <- mkMult1

    rules
      "go" : when state == 0  ==>
          m.start (9, 5)
          state := 1

      "finish" : when state == 1  ==>
          $display ("Product = %d",m.result())
          state := 2
```

```
interface Mult_ifc =
   start :: Tin ->Tin -> Action
   result :: Tout
```

```
mkMult1 ::  Module Mult_ifc
mkMult1 =
  module
      product ::  Reg Tout
      product <- mkReg 0
      d :: Reg Tout
      d    <- mkReg  0
       r :: Reg Tin
       r    <- mkReg  0
       rules
         "cycle": when r /= 0 ==>
             if r[0] == 1
                  product := product + d
             d := d << 1
              r := r >> 1
       interface
          start x y  = action { d:=x; r:=y; }
                           when r == 0
          result = product
                           when r == 0
```

Multiplier Example

```
ESL_MODULE (mkTest, ESL_EMPTY) {
  esl_reg<int> state;
  Mult_ifc * m;

  ESL_RULE (go, state == 0) {
     m->start (9, 5);
     state = 1;
  }

  ESL_RULE (finish, state == 1) {
     cout << “Product = ” << m->result()
          << endl;
     state = 2;
  }

  ESL_CTOR (mkTest) {
     ESL_NEW_REG(state, int, 0);
     m = new mkMult1();
     ESL_END_CTOR;
  }
}
```

```
ESL_INTERFACE ( Mult_ifc ) {
  ESL_ACTION_METHOD_INTERFACE
                                  (start, Tin x,Tin y);
  ESL_VALUE_METHOD_INTERFACE (result, Tout);
}
```

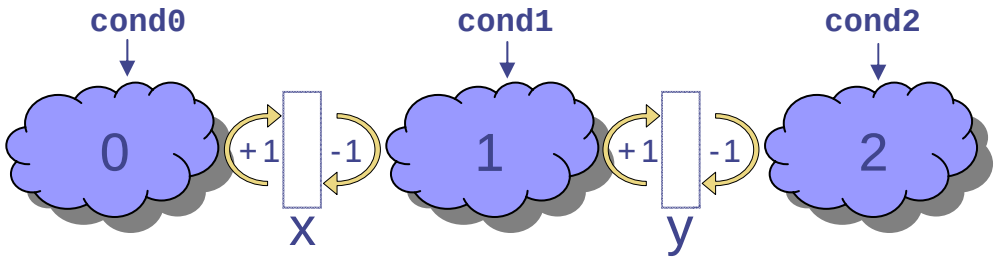
Concern: the “learning curve”

- Hardware designers:
  - “Atomic” means (if anything) “in the same clock cycle”
- Bluespec:
  - Many atomic actions in same cycle, as if they happened one at a time

Concern: the “learning curve”

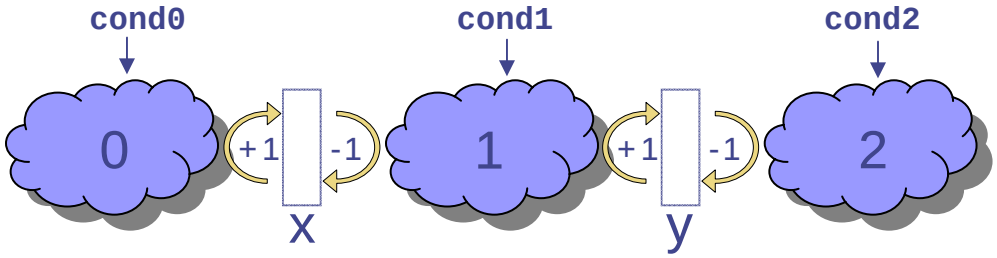
- Hardware designers:
  - Begin by designing the overall finite state machine to control the design
- Bluespec:
  - Design little rules locally: the compiler will generate the overall logic

Simple example with concurrency and shared resources



Process priority: 2 > 1 > 0

- Process 0: increments register ***x*** when ***cond0***
- Process 1: transfers a unit from register ***x*** to register ***y*** when ***cond1***
- Process 2: decrements register ***y*** when ***cond2***
- Each register can only be updated by one process on each clock. Priority: 2 > 1 > 0
- Just like real applications, e.g.:
  - Packet arrives, is processed, departs



Process priority: 2 > 1 > 0

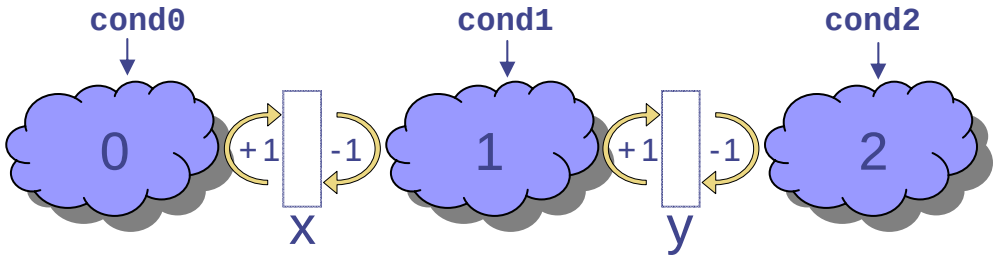
Which one is correct?

```
always @(posedge CLK) begin
  if (!cond2 && cond1)
    x <= x - 1;
  else if (cond0)
    x <= x + 1;

  if (cond2)
    y <= y - 1;
  else if (cond1)
    y <= y + 1;
end
```

```
always @(posedge CLK) begin
  if (!cond2 || cond1)
    x <= x - 1;
  else if (cond0)
    x <= x + 1;

  if (cond2)
    y <= y - 1;
  else if (cond1)
    y <= y + 1;
end
```

What's required to verify that they're correct?
What if the priorities changed: cond1 > cond2 > cond0?
What if the processes are in different modules?

With Bluespec, the design is direct



Process priority: 2 > 1 > 0

```
(* descending_urgency = "proc2, proc1, proc0" *)

rule proc0 (cond0);
    x <= x + 1;
endrule

rule proc1 (cond1);
    y <= y + 1;
    x <= x - 1;
endrule

rule proc2 (cond2);
    y <= y - 1;
endrule
```

Hand-written RTL:

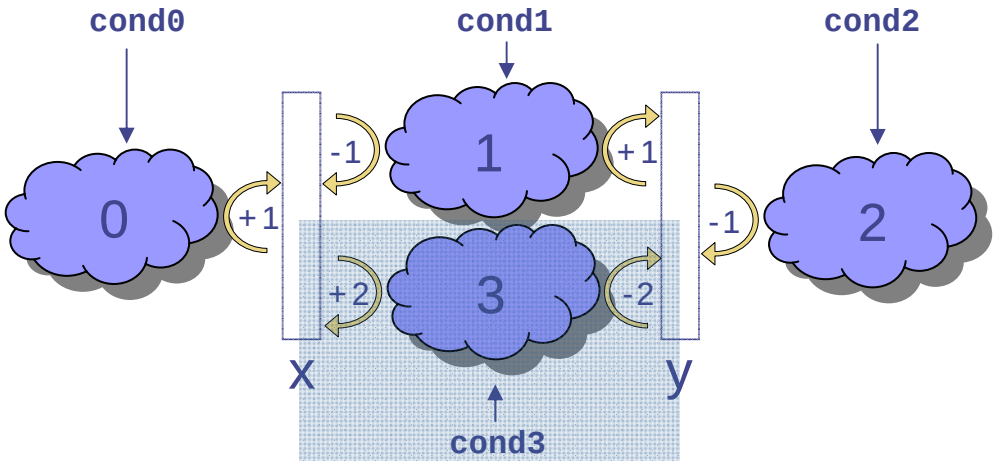
Complexity due to:

- ***State-centric (for synthesizability)***
- ***Scheduling clutter***

BSV:

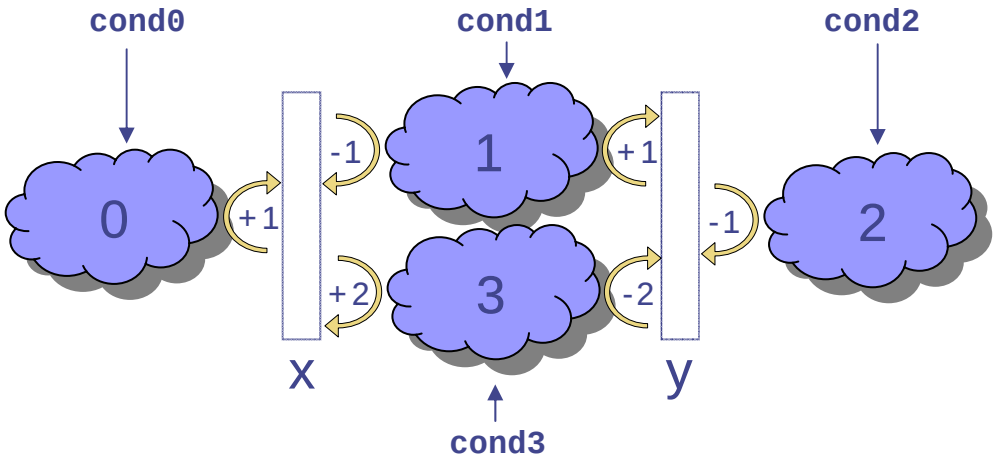
- ***Functional correctness follows directly from rule semantics***
- ***Executable spec (operation-centric)***
- ***Automatic handling of shared resource mux logic***
- ***Same hardware as the RTL***

Now, let's make a small change: add a new process and insert its priority



Process priority: 2 > 3 > 1 > 0

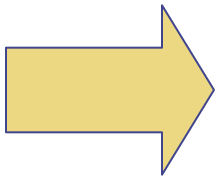
Changing the Bluespec design



Process priority: 2 > 3 > 1 > 0

Pre-Change

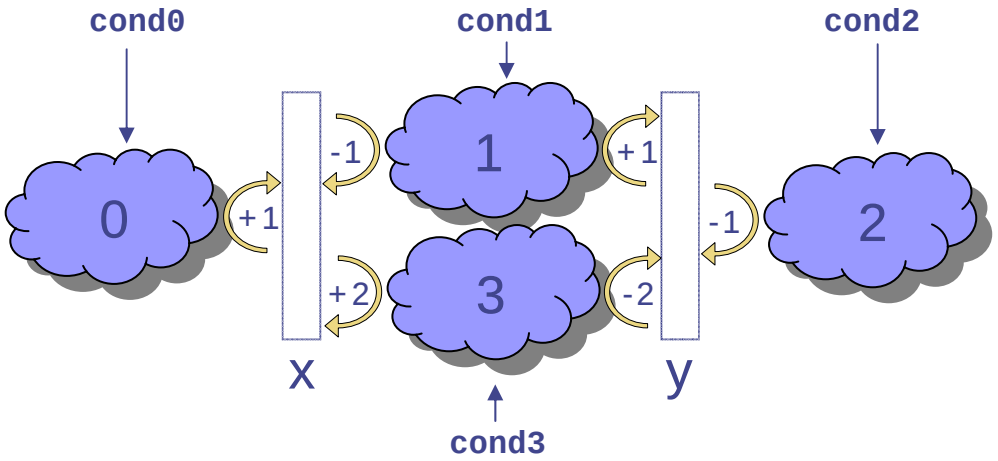
```
(* descending_urgency = “proc2, proc1, proc0” *)

rule proc0 (cond0);
    x <= x + 1;
endrule

rule proc1 (cond1);
    y <= y + 1;
    x <= x – 1;
endrule

rule proc2 (cond2);
    y <= y – 1;
endrule
```



```
(* descending_urgency = "proc2, proc3, proc1, proc0" *)

rule proc0 (cond0);
    x <= x + 1;
endrule

rule proc1 (cond1);
    y <= y + 1;
    x <= x - 1;
endrule

rule proc2 (cond2);
    y <= y - 1;
    x <= x + 1;
endrule

rule proc3 (cond3);
    y <= y - 2;
    x <= x + 2;
endrule
```

?

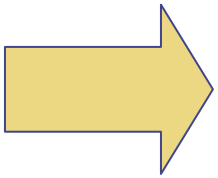
Changing the Verilog design



Process priority: 2 > 3 > 1 > 0

Pre-Change

```
always @(posedge CLK) begin
  if (!cond2 && cond1)
    x <= x – 1;
  else if (cond0)
    x <= x + 1;

  if (cond2)
    y <= y – 1;
  else if (cond1)
    y <= y + 1;
end
```



```
always @(posedge CLK) begin
  if ((cond2 && cond0) || (cond0 && !cond1 && !cond3))
    x <= x + 1;
  else if (cond3 && !cond2)
    x <= x + 2;
  else if (cond1 && !cond0)
    x <= x - 1

  if (cond2)
    y <= y - 1;
  else if (cond3)
    y <= y - 2;
  else if (cond1)
    y <= y + 1;
end
```

?

Concern: the “learning curve”

- Other ways to make customers immediately productive
  - Offer generic IP they can use “off the shelf” with very little training
  - They can customize
    - But then they’ll have to learn more, to do so

So how does Arvind fit into this?

bluespec

- Board member
- Provides longer-term view of the technical development

A FIFO problem

```
module mkFifo(Fifo#(t))
  provisos(Bits#(t,ts));
  Reg#(t) dta <- mkRegU;
  Reg#(Bool) full <- mkReg(False);

  method push(t x) if (!full);
    full <= True; dta <= x;
  endmethod

  method t pop() if (full);
    full <= False; return dta;
  endmethod
endmodule
```

A FIFO problem

- Immediate solution (Bluespec)
  - Invent “wires”
- Wires similar to registers, but:
  - Registers:
    - Read “early” in clock cycle, written at end
  - Wires:
    - Written first, read later
    - Value goes away at end of cycle

A FIFO problem

- Immediate solution (Bluespec)
  - New primitive: no great upheaval to language
  - Methods set wires; an internal rule makes the appropriate state change
  - Allows the “problem” to be overcome by enhancing the FIFO’s design

A FIFO problem

- Immediate solution (Bluespec)
- But:
  - Separating method from state change breaks atomicity – needs care to get internal behavior right
  - Leads users into bad practices, and spaghetti-like designs

A FIFO problem

- Longer-term solution (Arvind)
- Extend language: allow designers to specify scheduling requirements
  - *Performance Guarantees*
    - "Allow pop before push in same cycle"
    - Compiler generates necessary hardware
- Cleaner; more robust
- Need more intuition about what the compiler will do

A Final Concern

- Has Arvind's fame spread far and wide?

c:\users\stoy>finger arvind@mit.edu
[mit.edu]

1

Student data loaded as of May 17, Staff data loaded as of May 17.
Notify Personnel or use WebSIS as appropriate to change your information.

Our on-line help system describes
How to change data, how the directory works, where to get more info.
For a listing of help topics, enter finger help@mit.edu. Try finger help_about@mit.edu to read about how the directory works.
Directory bluepages may be found at http://mit.edu/communications/bp.

There were 3 matches to your request.
Complete information will be shown only when one individual matches your query. Resubmit your query with more information.
For example, use both firstname and lastname or use the alias field.

name: Arvind, Amarnath
department: System Design And Management
year: G
alias: A-arvind

name: Jairam, Arvind
department: Lincoln Laboratory
title: LL - Associate Staff
alias: A-jairam

name: Arvind
department: Dept of Electrical Engineering & Computer Science
title: Charles W & Jennifer C Johnson Professor in CS Eng
alias: arvind

c:\users\stoy>

c:\users\stoy>finger arvind@mit.edu
[mit.edu]

2

Student data loaded as of May 17, Staff data loaded as of May 17.
Notify Personnel or use WebSIS as appropriate to change your information.

Our on-line help system describes
How to change data, how the directory works, where to get more info.
For a listing of help topics, enter finger help@mit.edu. Try finger help_about@mit.edu to read about how the directory works.
Directory bluepages may be found at http://mit.edu/communications/bp.

There were 3 matches to your request.
Complete information will be shown only when one individual matches your query. Resubmit your query with more information.
For example, use both firstname and lastname or use the alias field.

name: Arvind, Amarnath
department: System Design And Management
year: G
alias: A-arvind

name: Jairam, Arvind
department: Lincoln Laboratory
title: LL - Associate Staff
alias: A-jairam

name: Arvind
department: Dept of Electrical Engineering & Computer Science
title: Charles W & Jennifer C Johnson Professor in CS Eng
alias: arvind

c:\users\stoy>

bluespec

Conclusion:

Arvind is recursively impossible.