Module Interfaces and Concurrency

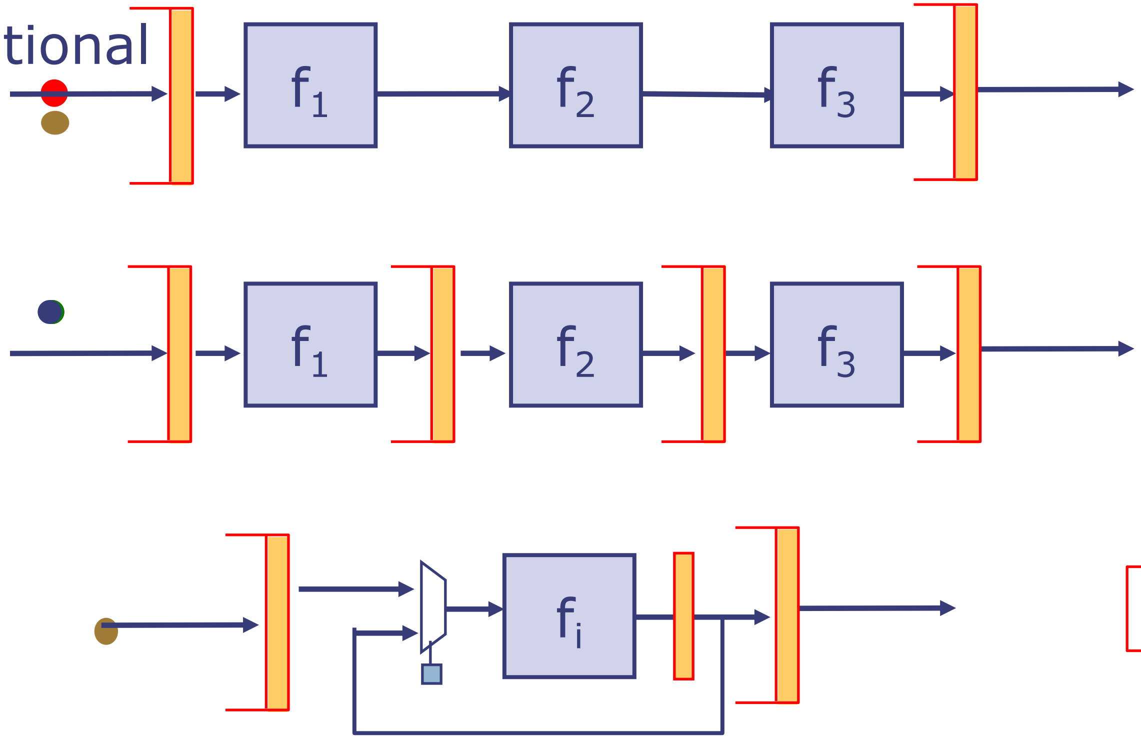
Arvind
Computer Science and Artificial Intelligence Laboratory
M.I.T.

Design Alternatives:
Latency and Throughput

Combinational (C)

Pipeline (P)

Folded: Reuse a block (F)



Suppose the latency of each

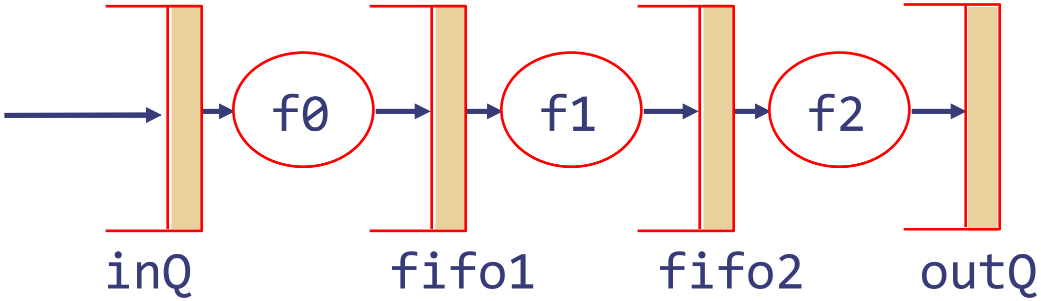
- f_i is 5ns
- FIFO is 1ns
- mux is 1ns
- register is 1ns

Area: F<C<P

- Latency: Total time to process 1 element
- Throughput: Rate at which new elements can be processed

	C	P	F
Latency (ns)	1+3*5+1=17	1+3*(5+1)=19	1+3*(1+5+1)+1=23
Throughput (1/ns)	1/17	1/(5+1) = 1/6	1/23
Clock period	16	6	7

A pipelined system



Without concurrent execution it is hardly a pipelined system

```
rule stage1;
  fifo1.enq(f0(inQ.first));
  inQ.deq;    endrule
rule stage2;
  fifo2.enq(f1(fifo1.first));
  fifo1.deq; endrule
rule stage3;
  outQ.enq(f2(fifo2.first));
  fifo2.deq; endrule
```

- When can stage1 rule fire?
  - inQ has an element
  - fifo1 has space
- Can these 3 rules execute concurrently?
  - Yes, but it must be possible to do enq and deq in a fifo simultaneously
- Can stage1 and stage3 execute concurrently?
  - Yes, even if enq and deq cannot be done simultaneously in a fifo

Multirule Systems

- Most systems we have seen so far had multiple rules but only one rule was ready to execute at any given time (pair-wise mutually exclusive rules)
- Consider a system where multiple rules can be ready to execute at a given time
  - When can two such rules be executed together?
  - What does the synthesized hardware look like for concurrent execution of rules?

One-rule-at-a-time semantics of Bluespec

Repeatedly:

- Select any rule that is ready to execute
- Compute the state updates
- Make the state updates

Any legal behavior of a Bluespec program can be explained by observing the state updates obtained by applying one rule at a time

However, for performance we execute multiple rules concurrently whenever possible

Concurrent execution of rules

- Two rules can execute concurrently, if concurrent execution would not cause a double-write error, *and*
- The final state can be obtained by executing rules one-at-a-time in some sequential order
- Can these rules execute concurrently without violating the one-rule-at-a-time-semantics?

Example 1

```
rule ra;
   x <= x+1;
endrule
rule rb;
   y <= y+2;
endrule
```

Example 2

```
rule ra;
   x <= y+1;
endrule
rule rb;
   y <= x+2;
endrule
```

Example 3

```
rule ra;
   x <= y+1;
endrule
rule rb;
   y <= y+2;
endrule
```

Concurrent Execution

Example 1

```
rule ra;
   x^{t+1} <= x^{t}+1;
endrule
rule rb;
   y^{t+1} <= y^{t}+2;
endrule
```

Example 2

```
rule ra;
   x^{t+1} <= y^{t}+1;
endrule
rule rb;
   y^{t+1} <= x^{t}+2;
endrule
```

Example 3

```
rule ra;
   x^{t+1} <= y^{t}+1;
endrule
rule rb;
   y^{t+1} <= y^{t}+2;
endrule
```

Final value of (x,y) given the initial values (0,0)

	Ex 1	Ex 2	Ex 3
Concurrent Execution	(1,2)	(1,2)	(1,2)
ra < rb			
rb < ra			

Executing ra before rb (ra < rb)

Example 1

```
rule ra;
   $x^{t+1}$ <= $x^{t}$+1;
endrule
rule rb;
   $y^{t+2}$ <= $y^{t+1}$+2;
endrule
```

Example 2

```
rule ra;
   $x^{t+1}$ <= $y^{t}$+1;
endrule
rule rb;
   $y^{t+2}$ <= $x^{t+1}$+2;
endrule
```

Example 3

```
rule ra;
   $x^{t+1}$ <= $y^{t}$+1;
endrule
rule rb;
   $y^{t+2}$ <= $y^{t+1}$+2;
endrule
```

Final value of (x,y) given the initial values (0,0)

	Ex 1	Ex 2	Ex 3
Concurrent Execution	(1,2)	(1,2)	(1,2)
ra < rb	(1,2)	(1,3)	(1,2)
rb < ra			

For any x, if there is no x^{t+1} defined, then $x^{t+1} = x^{t}$

Executing rb before ra (rb < ra)

Example 1

```
rule ra;
   x^{t+2} <= x^{t+1}+1;
endrule
rule rb;
   y^{t+1} <= y^{t}+2;
endrule
```

Example 2

```
rule ra;
   x^{t+2} <= y^{t+1}+1;
endrule
rule rb;
   y^{t+1} <= x^{t}+2;
endrule
```

Example 3

```
rule ra;
   x^{t+2} <= y^{t+1}+1;
endrule
rule rb;
   y^{t+1} <= y^{t}+2;
endrule
```

Final value of (x,y) given the initial values (0,0)

	Ex 1	Ex 2	Ex 3
Concurrent Execution	(1,2)	(1,2)	(1,2)
ra < rb	(1,2)	(1,3)	(1,2)
rb < ra	(1,2)	(3,2)	(3,2)

Can these rules execute concurrently?
(without violating the one-rule-at-a-time-semantics)

Example 1

```
rule ra;
   x <= x+1;
endrule
rule rb;
   y <= y+2;
endrule
```

Example 2

```
rule ra;
   x <= y+1;
endrule
rule rb;
   y <= x+2;
endrule
```

Example 3

```
rule ra;
   x <= y+1;
endrule
rule rb;
   y <= y+2;
endrule
```

Final value of (x,y) given the initial values (0,0)

	Ex 1	Ex 2	Ex 3
Concurrent Execution	(1,2)	(1,2)	(1,2)
ra < rb	= (1,2)	≠ (1,3)	= (1,2)
rb < ra	= (1,2)	≠ (3,2)	≠ (3,2)
	Conflict-Free (CF)	Conflict	ra<rb

Conflict Matrix (CM)

BSV compiler generates the pairwise conflict information

Example 1

```
rule ra;
   x <= x+1;
endrule
rule rb;
   y <= y+2;
endrule
```

	ra	rb
ra	C	CF
rb	CF	C

Example 2

```
rule ra;
   x <= y+1;
endrule
rule rb;
   y <= x+2;
endrule
```

	ra	rb
ra	C	C
rb	C	C

Example 3

```
rule ra;
   x <= y+1;
endrule
rule rb;
   y <= y+2;
endrule
```

	ra	rb
ra	C	<
rb	>	C

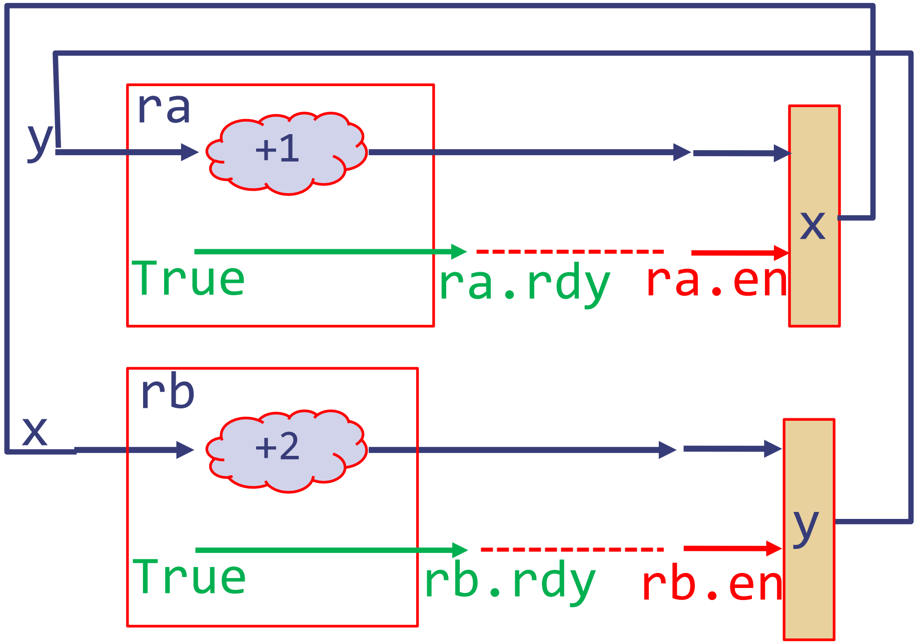
- C : rules can't be executed concurrently
- CF: rules can be executed concurrently; the net effect is the same as if ra executed before rb (ra<rb) or (rb<ra)
- ra < rb : rules can be executed concurrently; the net effect is as if ra executed before rb

Using *conflict* (CM) information in hardware synthesis

Concurrent rule execution

Example 2

```
rule ra;
   x <= y+1;
endrule
rule rb;
   y <= x+2;
endrule
```

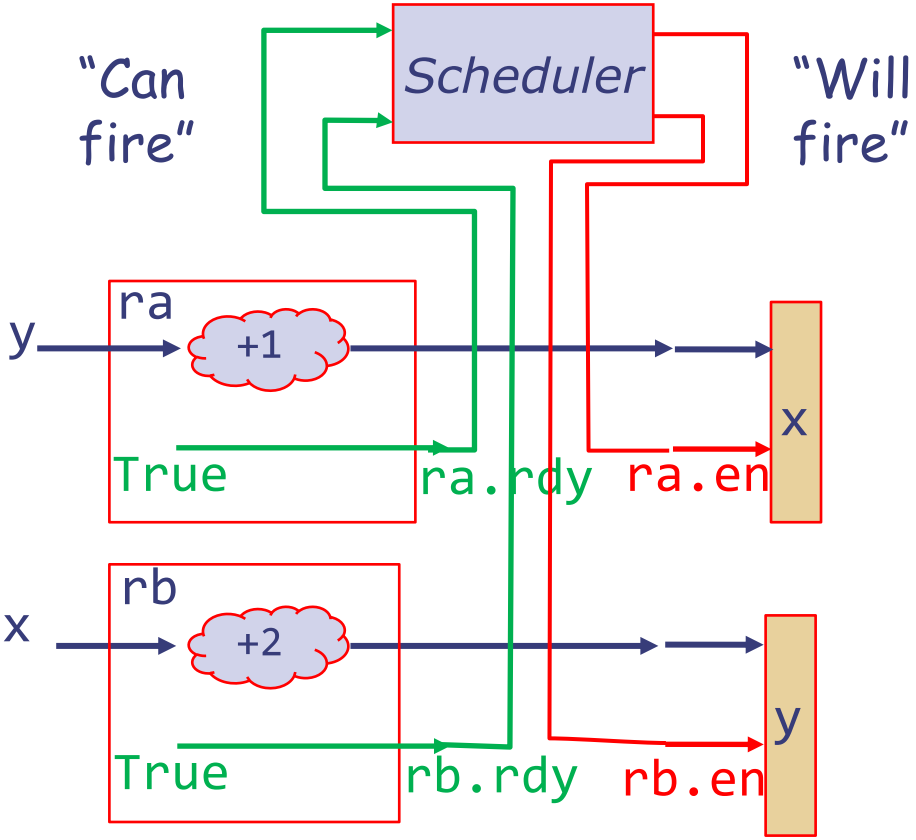


- Suppose we connect ra.rdy to ra.en, and rb.rdy to rb.en
- This circuit will execute rules ra and rb concurrently
- This circuit is correct only if rules ra and rb do not conflict
- But in this example rules ra and rb do conflict!

Need for a scheduler

Example 2

```
rule ra;
   x <= y+1;
endrule
rule rb;
   y <= x+2;
endrule
```



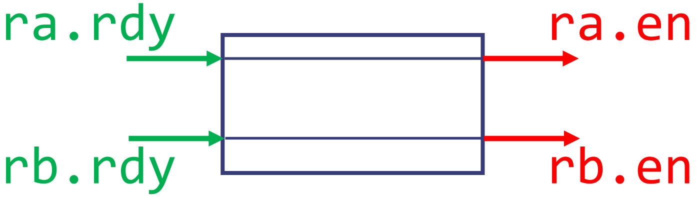
- Guards (rdy signals) of all rules are fed to a scheduler
- Using the CM, the scheduler lets only non-conflicting rules proceed

- *Scheduler is a pure combinational circuit with a small number of gates*
- *A correct but low performance scheduler may schedule only one rule at a time*

Example schedulers

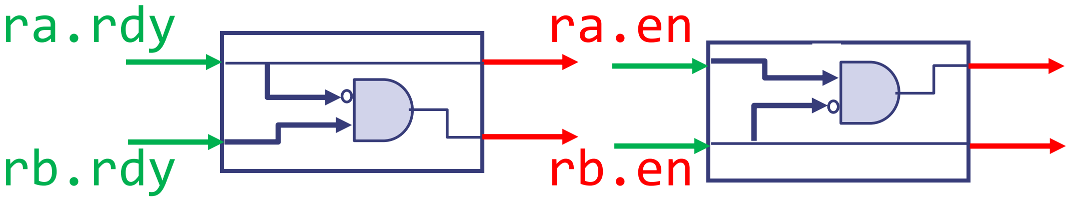
Example1

	ra	rb
ra	C	CF
rb	CF	C



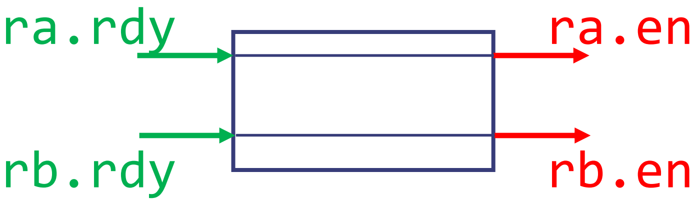
Example2

	ra	rb
ra	C	C
rb	C	C



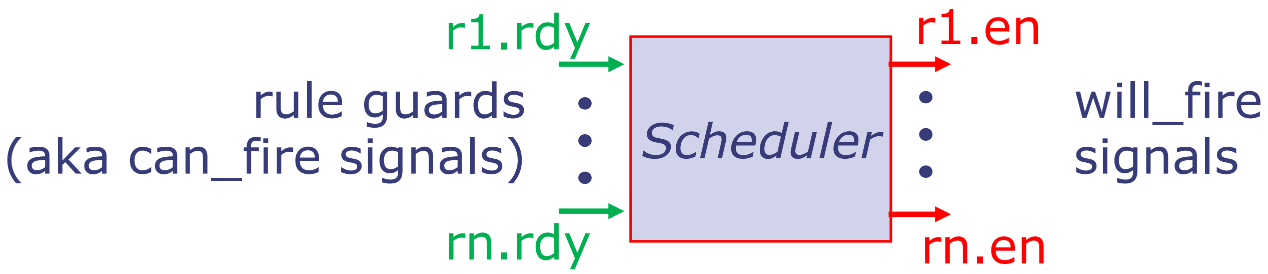
Example3

	ra	rb
ra	C	<
rb	>	C



The effect ra<rb would be achieved automatically

The rule scheduler



- Guards (r1.rdy … rn.rdy) of many rules may be true simultaneously, and some of them may conflict
- BSV compiler constructs a combinational scheduler circuit with the following property:

> for all pairs of rules *i* and *j*, if r^i.en and r^j.en are true then the corresponding r^i.rdy and r^j.rdy must be true and rules *i* and *j* *must not conflict* with each other

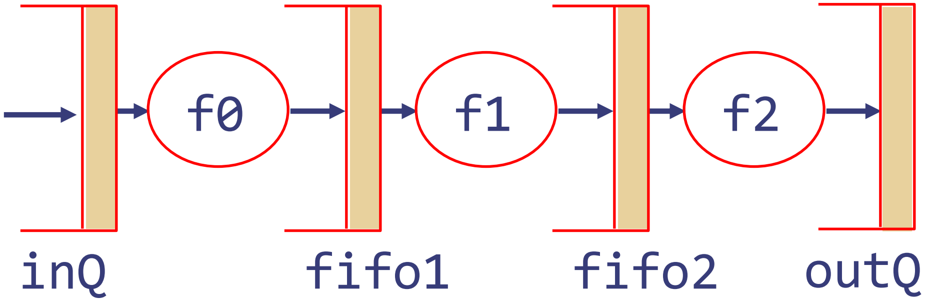
Takeaway

- One-rule-at-a-time semantics are important to understand the legal behaviors of a system
- Efficient hardware for multi-rule system requires that many rules execute in parallel without violating the one-rule-at-time semantics
- BSV compiler builds a scheduler circuit to execute as many rules as possible concurrently
- For high-performance designs we have to worry about the CM characteristics of our modules

More on this topic later in the course

We are not done yet!

```
rule stage1;
  fifo1.enq(f0(inQ.first));
  inQ.deq;   endrule
rule stage2;
  fifo2.enq(f1(fifo1.first));
  fifo1.deq; endrule
rule stage3;
  outQ.enq(f2(fifo2.first));
  fifo2.deq; endrule
```



- These rules can execute concurrently, only if fifos allow concurrent enq and deq
- In our one-element fifo design, enq and deq are mutually exclusive!
- We will design better FIFO's later

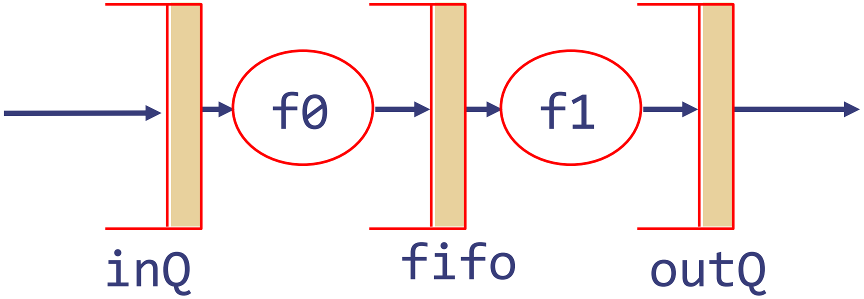
No pipelining

	enq	deq	first
enq	C	ME	ME
deq	ME	C	>
first	ME	<	CF

	enq	deq	first
enq	C	CF	CF
deq	CF	C	>
first	CF	<	CF

Take-home problem

- Draw a hardware circuit for this design, ignoring the internals of the fifo design



Hint

- Draw the guard for each rule
- Assume a enable signal for each rule
- Connect the rule ready and enable signals to a scheduler

```
rule stage1;
  fifo.enq(f0(inQ.first));
  inQ.deq;
endrule
rule stage2;
  outQ.enq(f1(fifo.first));
  fifo.deq;
endrule
```