

Using Reverse Circuit Execution for Efficient Parallel Simulation of Logic Circuits

Dr. Kalyan Perumalla
Prof. Richard Fujimoto

Parallel and Distributed Simulation Group (PADS)
College of Computing, Georgia Tech
www.cc.gatech.edu/computing/pads

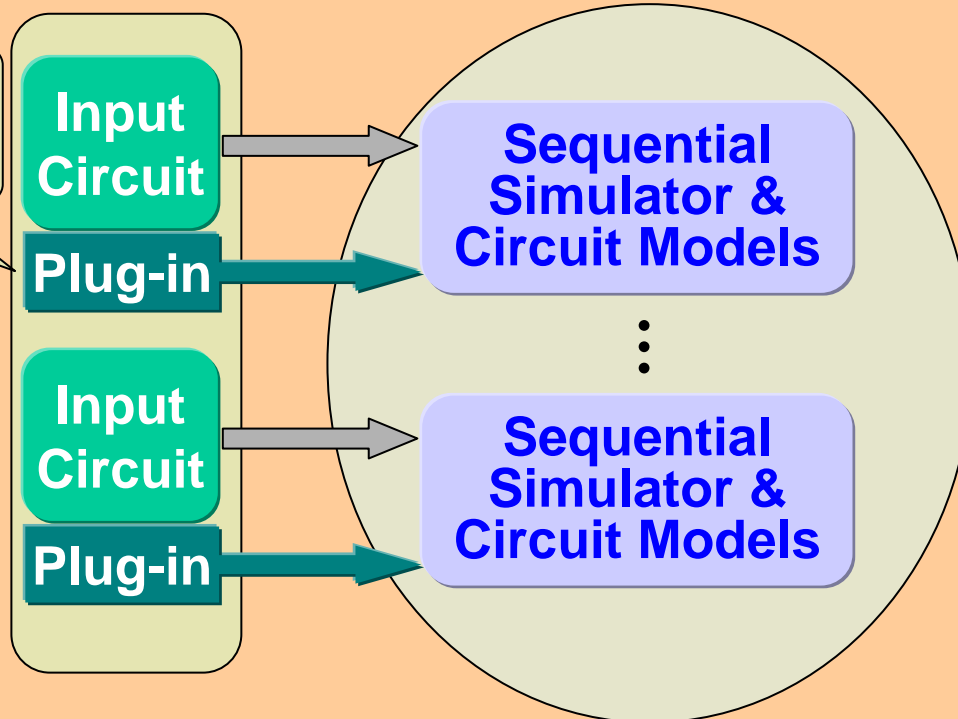
Work funded by the Georgia Yamacraw Mission
yamacraw.org

Problem Motivation & Solution

Enable parallel execution for commercial sequential circuit simulators

Motivation: overcome circuit size & runtime limitations of sequential execution

Solution:
Transform the input!



Challenge:
Can't modify these!
- proprietary software
- validated models

Presentation Outline

- Traditional parallel execution techniques
 - Sequential & Parallel DES Overview
 - Parallel synchronization approaches
- The reverse execution approach
 - How does it work?
 - Why choose it over others?
- How to apply reverse execution to circuit simulation
 - Example circuit
 - Generalized transformation architecture
 - Other details
- Conclusions & future work

Note:

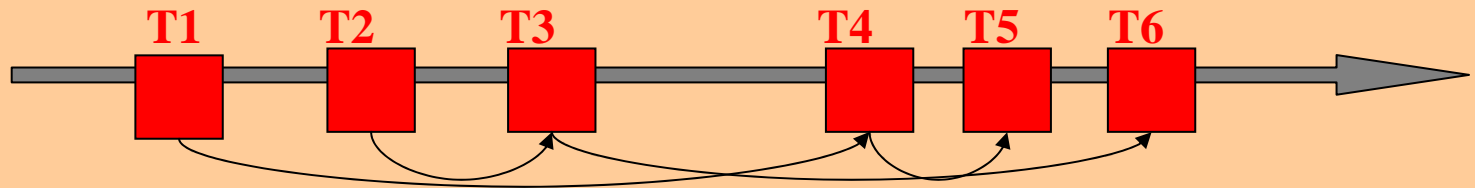
- Reverse computation is more generally applicable
- Here, applied to parallel logic circuit simulation

Sequential & Parallel Discrete Event Simulation

Sequential

1 processor

Example:

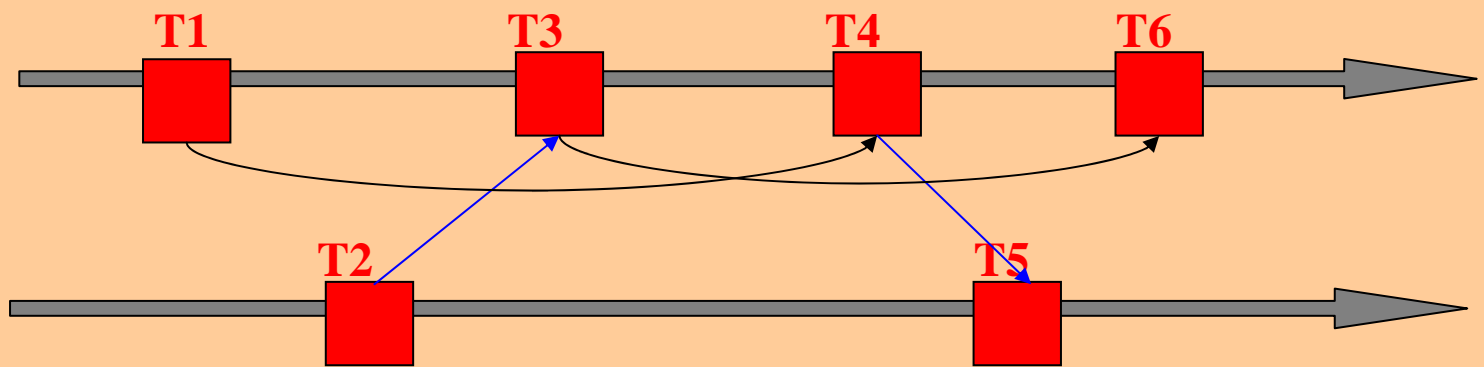


Parallel

n > 1 processors

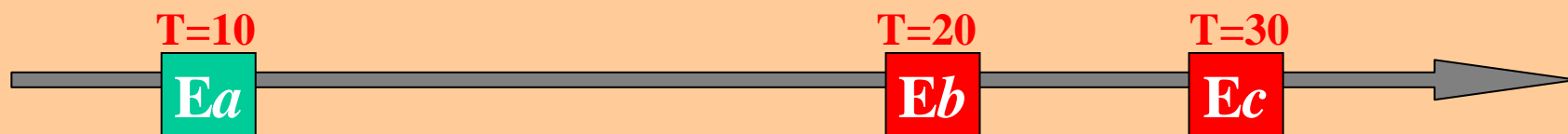
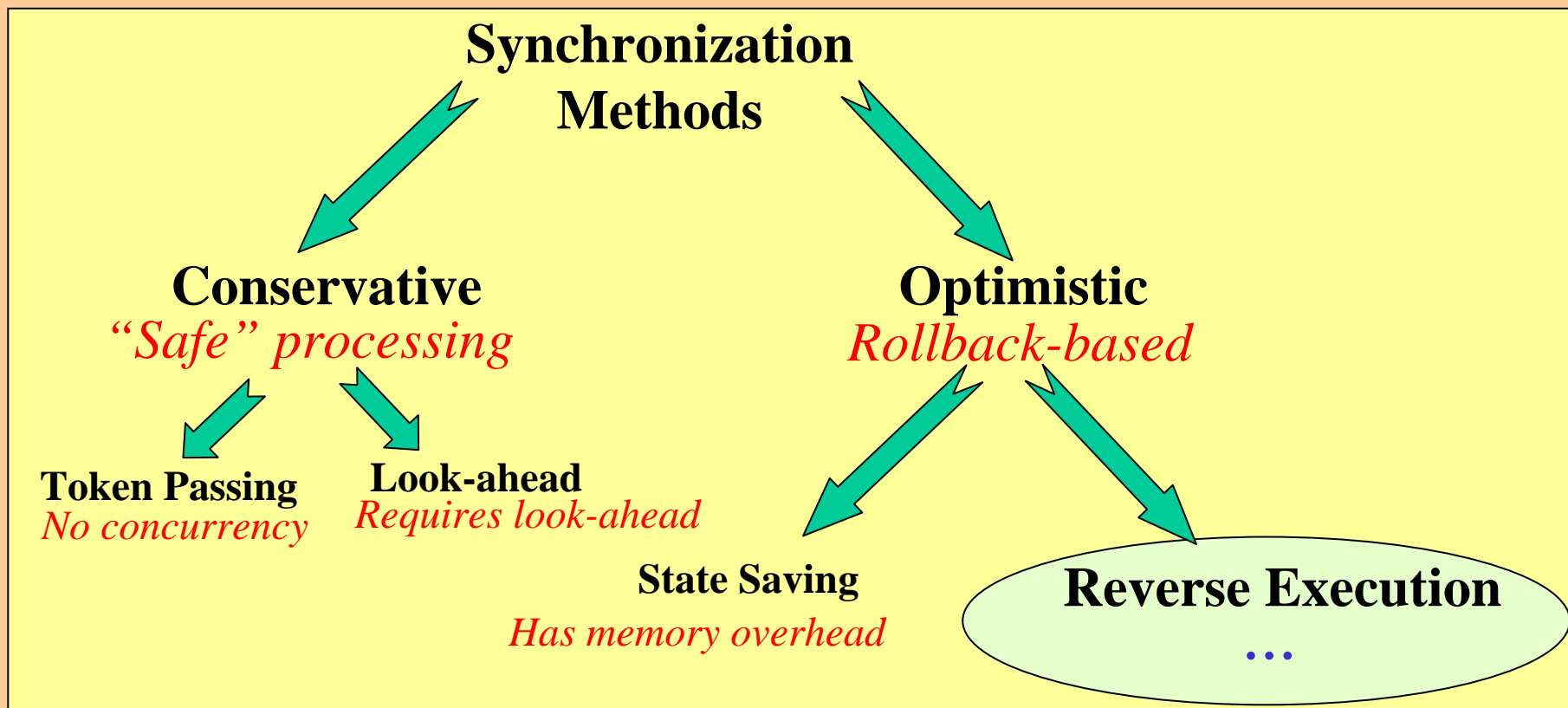
Example:

2 processors



Parallel Execution Techniques

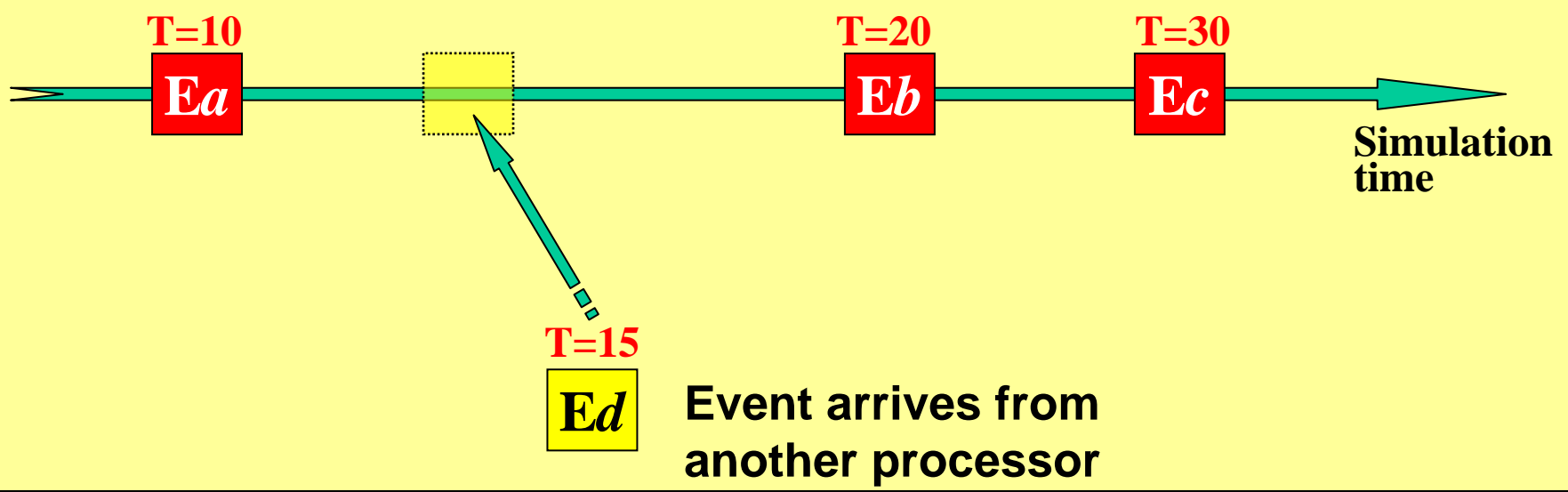
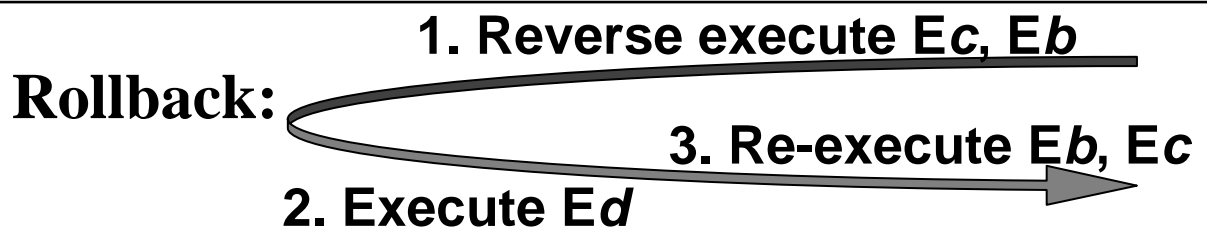
Goal: Ensure global timestamp-ordered processing.
=> Synchronization among simulators required.



Rollback Using Reverse Execution

Idea: Execute inverse operations to undo forward computation.

Example: Events E_a, E_b, E_c are processed. Later E_d arrives.



Why Choose Reverse Execution?

Advantages:

- Dynamic concurrency extraction
 - Due to optimistic processing
- Low memory overhead
 - No need to save state snapshots
- Can be automated
 - Pre-processor for production use

For circuit simulation:

- Doesn't require kernel/model source-code
 - Transformations on input circuit are sufficient

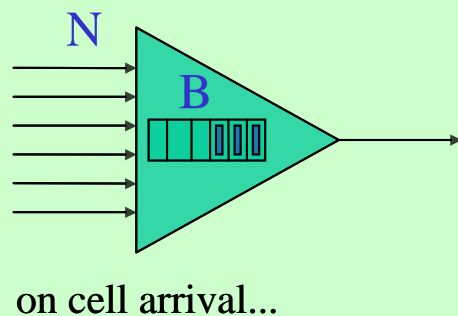
Side Note: Compression via Reverse Execution

- Can extract *core* effects of computation
- Size of state *trace* reduced significantly
- State trace is better compressible

Example:

Queue with
N inputs
and queue
size limit B

e.g. ATM
Multiplexer



Original

```
if( qlen < B )
  qlen++
  delays[qlen]++
else
  lost++
```

State Size
B+2 words

Forward

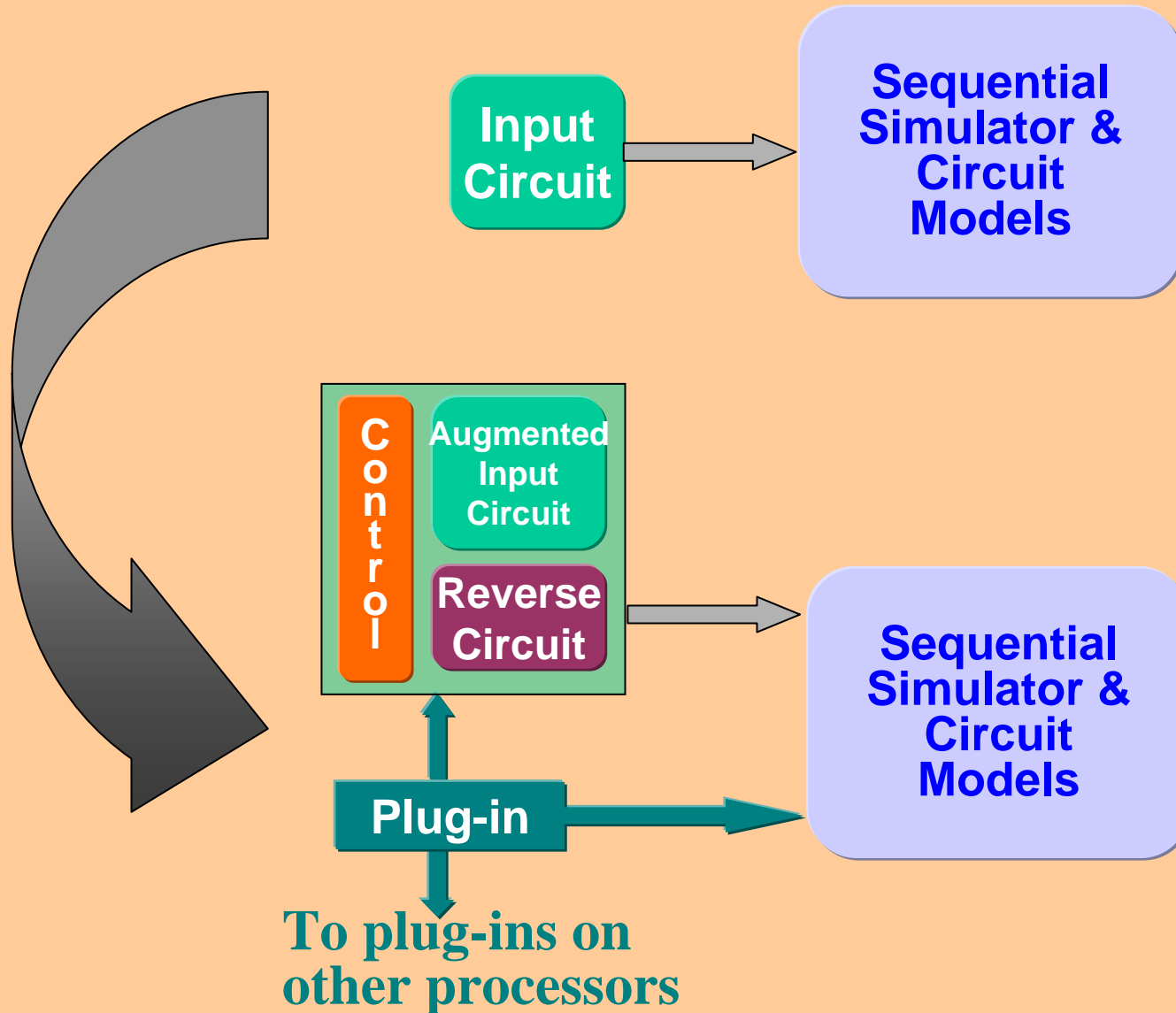
```
if( qlen < B )
  b = 1
  qlen++
  delays[qlen]++
else
  b = 0
  lost++
```

Reverse

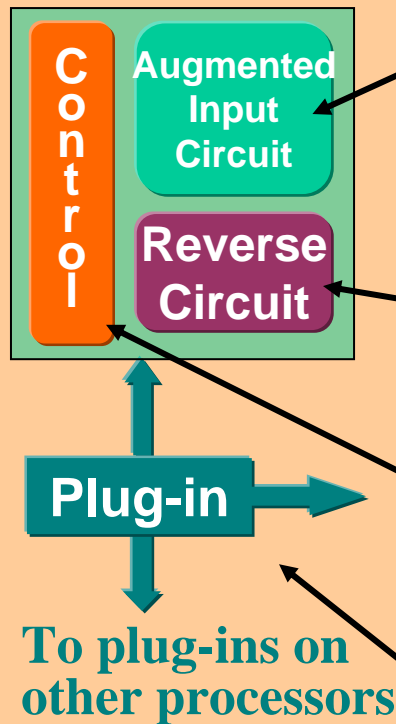
```
if( b == 1 )
  --delays[qlen]
  --qlen
else
  --lost
```

State Size
1 bit

Reverse *Circuit* Execution



Reverse *Circuit* Execution - Continued

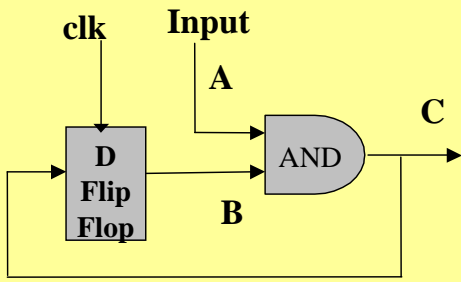


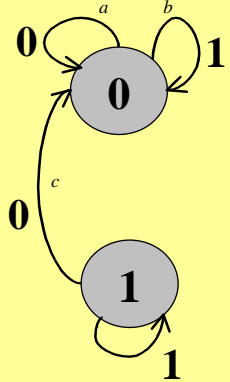
1. Augment the input *forward* circuit for reversibility
 - In general, input circuit not perfectly reversible (e.g. **AND**)
2. Generate *reverse* circuit from augmented forward circuit
3. Integrate forward & reverse circuits with *control* circuit
4. Drive the integrated circuit with *plug-in*.

Details in the
report

Augmenting Forward Circuit - Example

Original

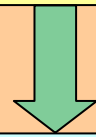




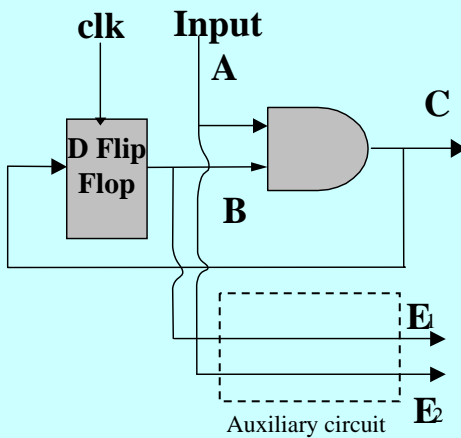
CS=current state, In=Input, NS=next state

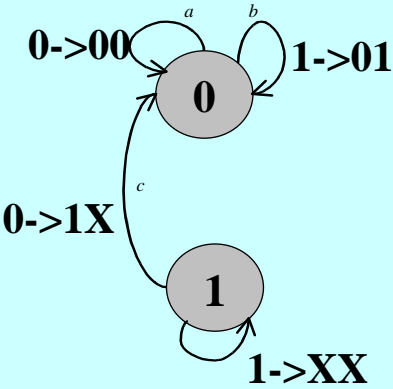
CS	In	NS
0	0	0
0	1	0
1	0	0
1	1	1

Transformed



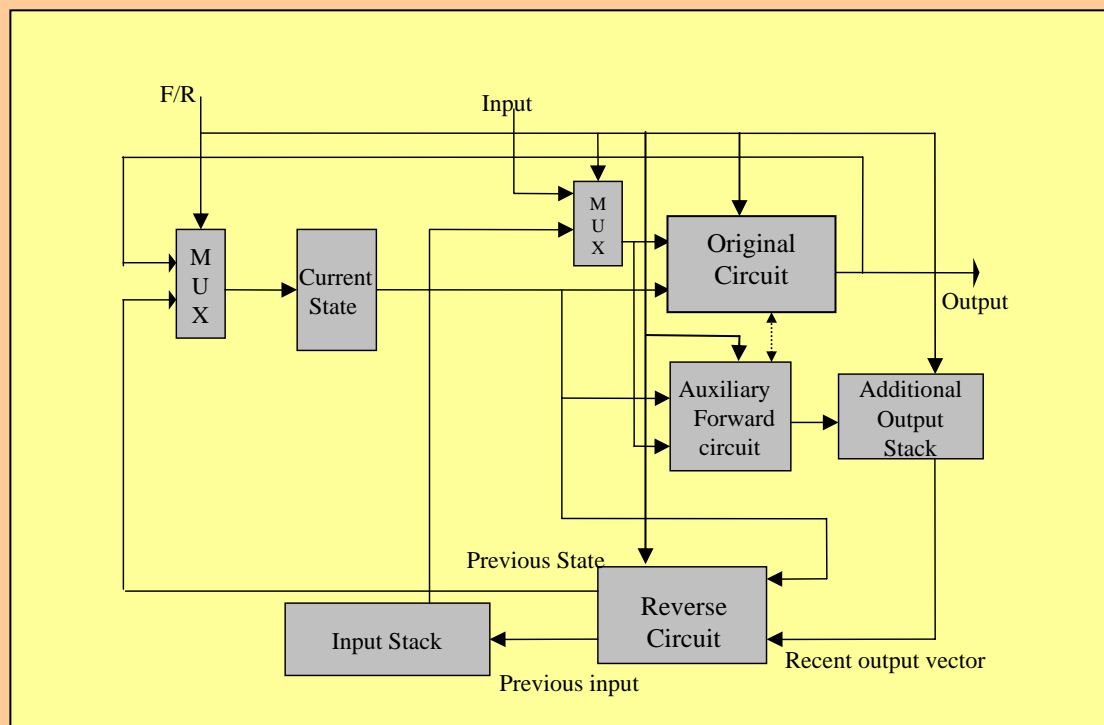
Log(Q) bits added
 Q=Maximum in-degree of any output vector





CS	In	NS	Extra	
			E1	E2
0	0	0	0	0
0	1	0	0	1
1	0	0	1	X
1	1	1	X	X

Forward + Reverse Circuit Architecture



- **F/R signal to switch between forward & reverse modes**
- **Auxiliary forward circuit generates additional output bits**
- **Output stack buffers output bits for reversal**
- **Input stack buffers input bits recovered during reversal**

Parallel Execution - Implementation

For each circuit partition (processor)

- Replace original circuit with forward/reverse circuit
- Use plug-ins to detect straggler events
- Initiate rollback upon receiving straggler events
- Schedule “infinitely fast” events for reverse execution during rollback
- Map simulator time to circuit virtual time and *vice versa* appropriately

Conclusions

- Novel parallelization approach - reverse circuit execution
 - Algorithms for reverse circuit generation
 - Architecture for forward-reverse circuit
- Better concurrency via optimistic execution
 - Using forward-reverse circuit
 - Can extract inherent parallelism
- Does not require simulator/model source-code
 - Automatic transformation of original input circuit
 - Mapping normal execution to optimistic execution
- Can extend to interoperate heterogeneous simulators

Future Work

- Automation & usability issues need to be addressed
- Simulator vs. Optimistic clocks need to be mapped
- Intermediate input/output bit stacks need to be dimensioned optimally for simulation speed
- Simulator statistics need to be reconciled with circuit rollback operation

Develop prototype using Cadence Verilog-XL

- Preliminary work completed with conservative synchronization using VPI