A Configurable High-Throughput Linear Sorter System

- Jorge Ortiz

  Information and Telecommunication Technology Center

  2335 Irving Hill Road

  Lawrence, KS

  jorgeo@ku.edu

- David Andrews

  Computer Science and Computer Engineering

  The University of Arkansas

  504 J.B. Hunt Building, Fayetteville, AR

  dandrews@uark.edu

Introduction

Introduction

- Sorting an important system function Popular sorting algorithms not efficient or fast in hardware implementations
- Linear sorters ideal for hardware, but sort at a rate of 1 value per cycle
- Sorting networks better at throughput, but with high area and latency cost
- Need a better solution for high throughput, low latency sorting

Contributions

- Expanding the linear sorter implementation and making it versatile, reconfigurable and better suited for streaming input and output
- Parallelizing the linear sorter for increased throughput
- Implementing the high-throughput linear sorter, and outmatching the performance of current linear sorter approaches

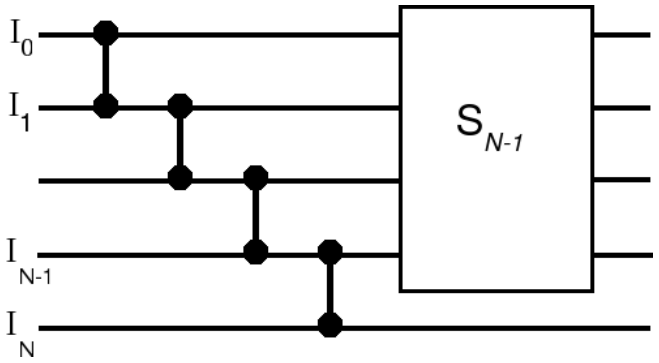
Background

Background

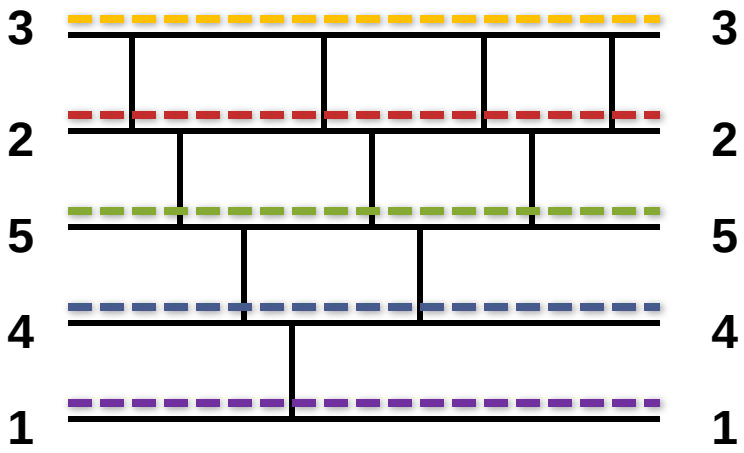
- Software quicksort, mergesort and heapsort use divide-and-conquer techniques to achieve efficiency
- Hardware sorting plagued with overhead from data movements, synchronization, bookkeeping and memory accesses
- Need better use of concurrent data comparisons and swaps, rather than the extended execution of multiple assembly instructions like its software counterpart

Sorting Networks

- Swap comparators sort pairs of values
- Sink lowest value, then operate on remaining S_{n-1} items
- Receive parallel data at inputs
- High #PE and latency, resort with each new insertion

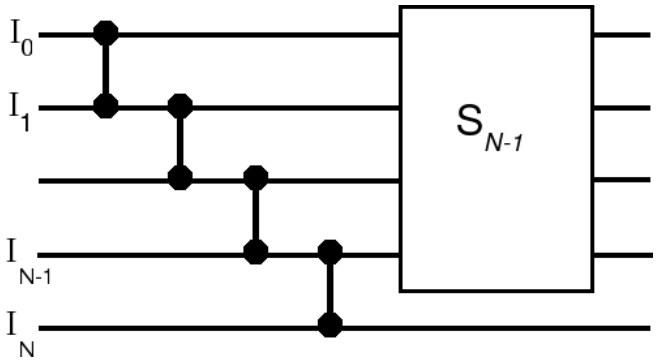


Bubble Sort

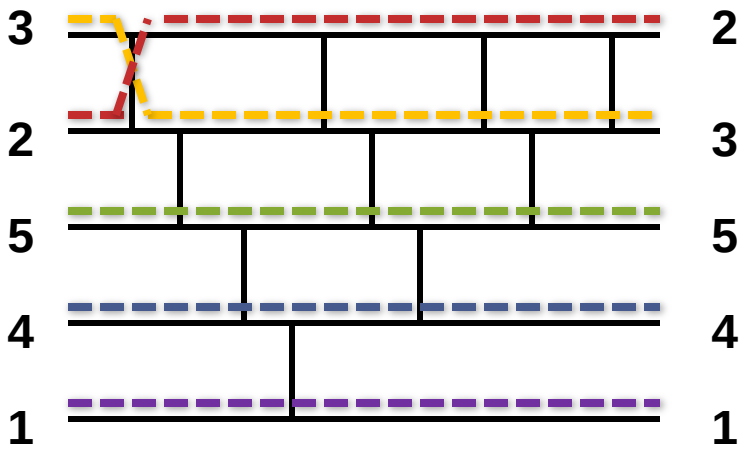


Sorting Networks

- Swap comparators sort pairs of values
- Sink lowest value, then operate on remaining S_{n-1} items
- Receive parallel data at inputs
- High #PE and latency, resort with each new insertion

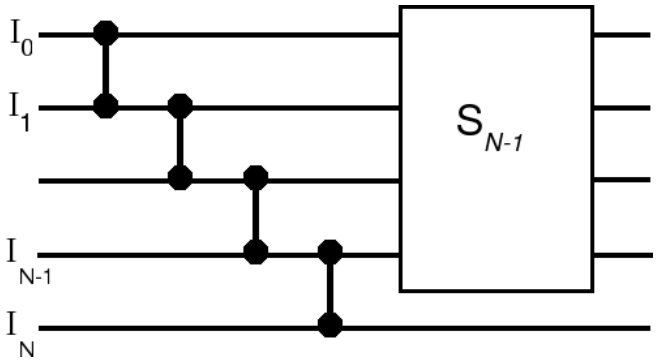


Bubble Sort

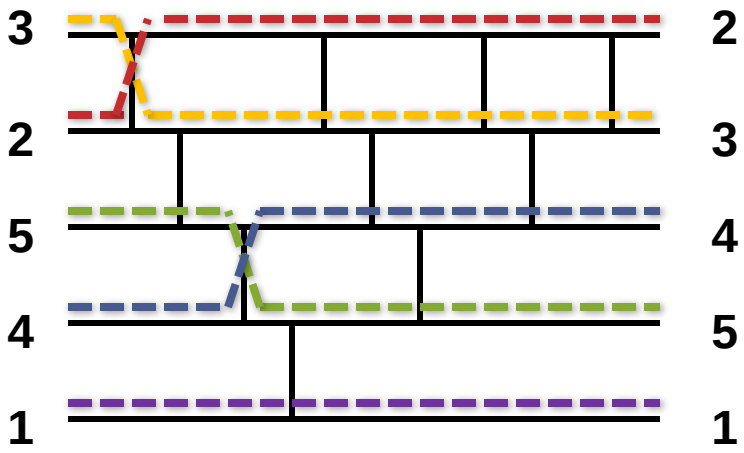


Sorting Networks

- Swap comparators sort pairs of values
- Sink lowest value, then operate on remaining S_{n-1} items
- Receive parallel data at inputs
- High #PE and latency, resort with each new insertion

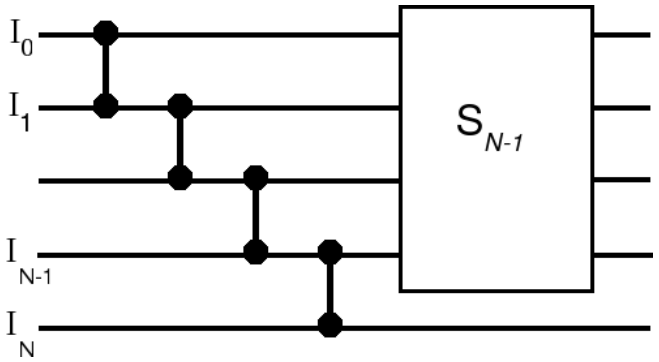


Bubble Sort

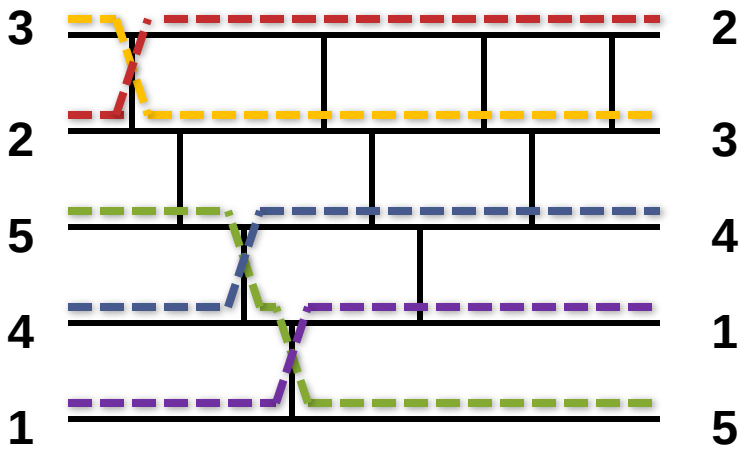


Sorting Networks

- Swap comparators sort pairs of values
- Sink lowest value, then operate on remaining S_{n-1} items
- Receive parallel data at inputs
- High #PE and latency, resort with each new insertion

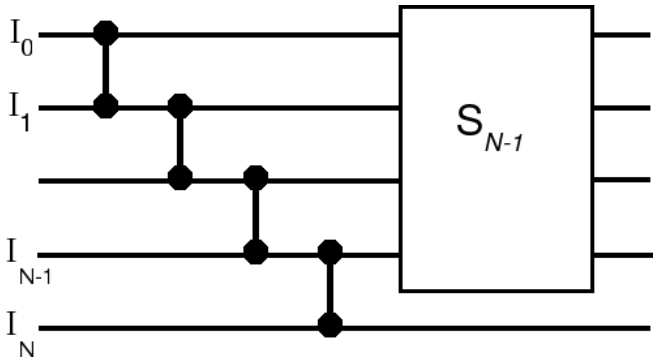


Bubble Sort

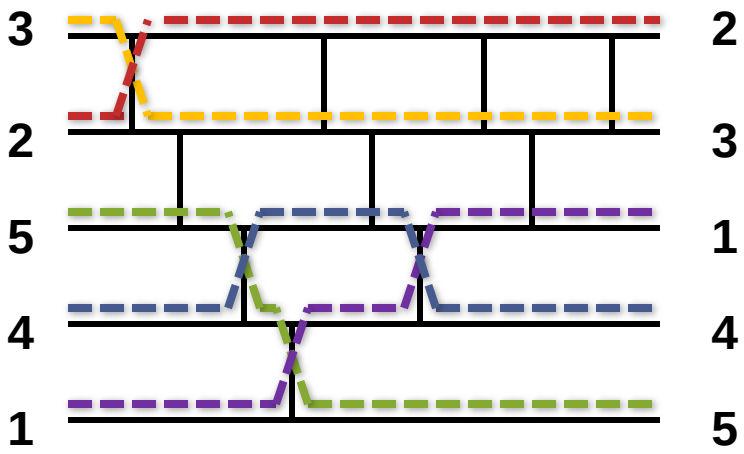


Sorting Networks

- Swap comparators sort pairs of values
- Sink lowest value, then operate on remaining S_{n-1} items
- Receive parallel data at inputs
- High #PE and latency, resort with each new insertion

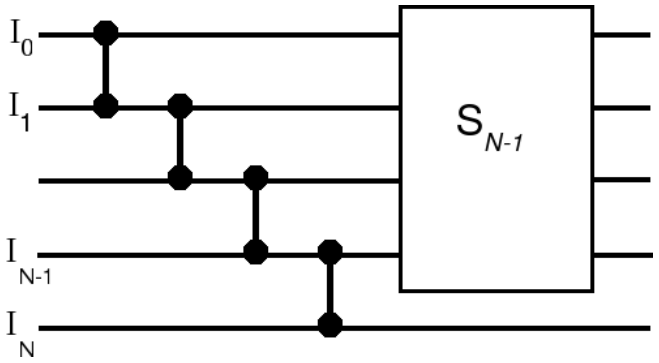


Bubble Sort

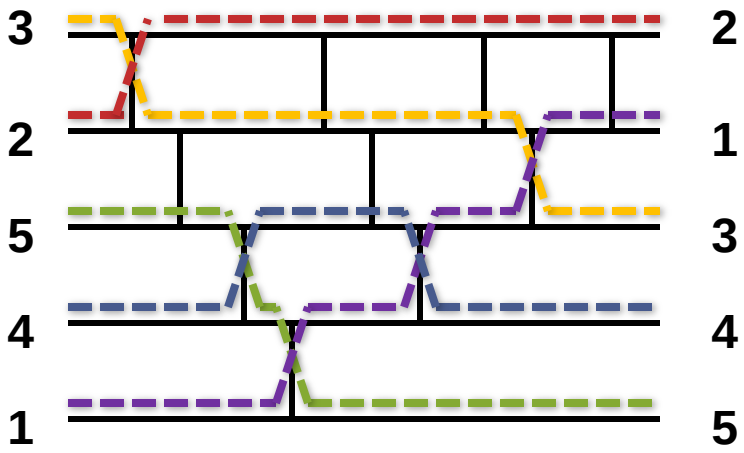


Sorting Networks

- Swap comparators sort pairs of values
- Sink lowest value, then operate on remaining S_{n-1} items
- Receive parallel data at inputs
- High #PE and latency, resort with each new insertion

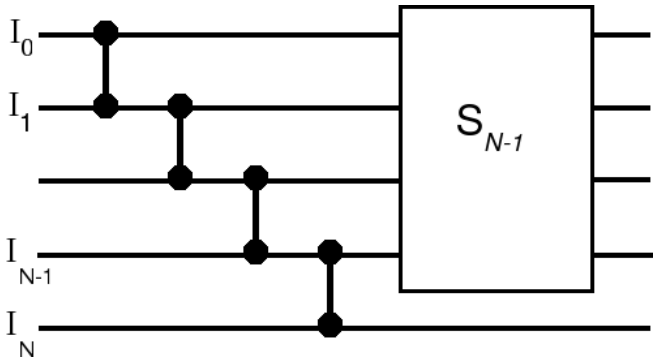


Bubble Sort

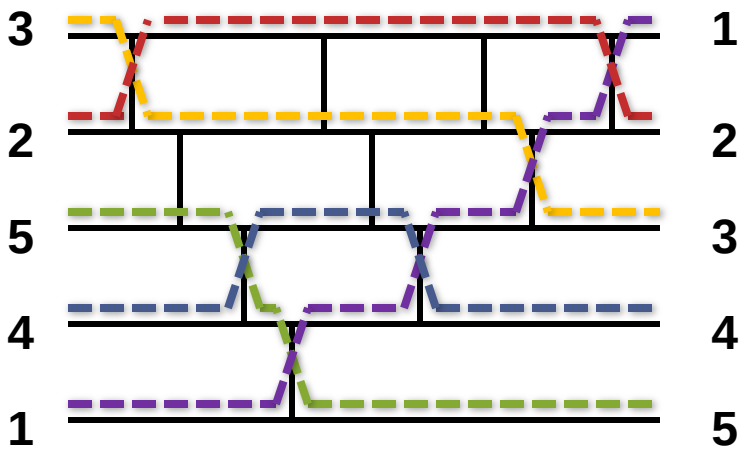


Sorting Networks

- Swap comparators sort pairs of values
- Sink lowest value, then operate on remaining S_{n-1} items
- Receive parallel data at inputs
- High #PE and latency, resort with each new insertion



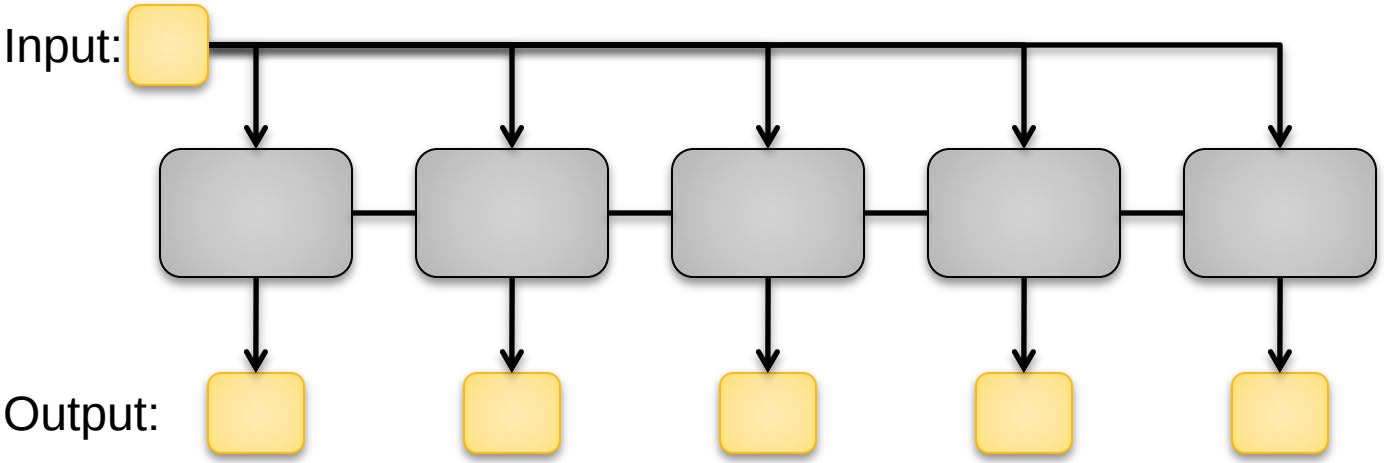
Bubble Sort



Linear Sorters

- Sorted insertions
- Forwards incoming value to all nodes
- Each node shifts autonomously depending on neighbors' values

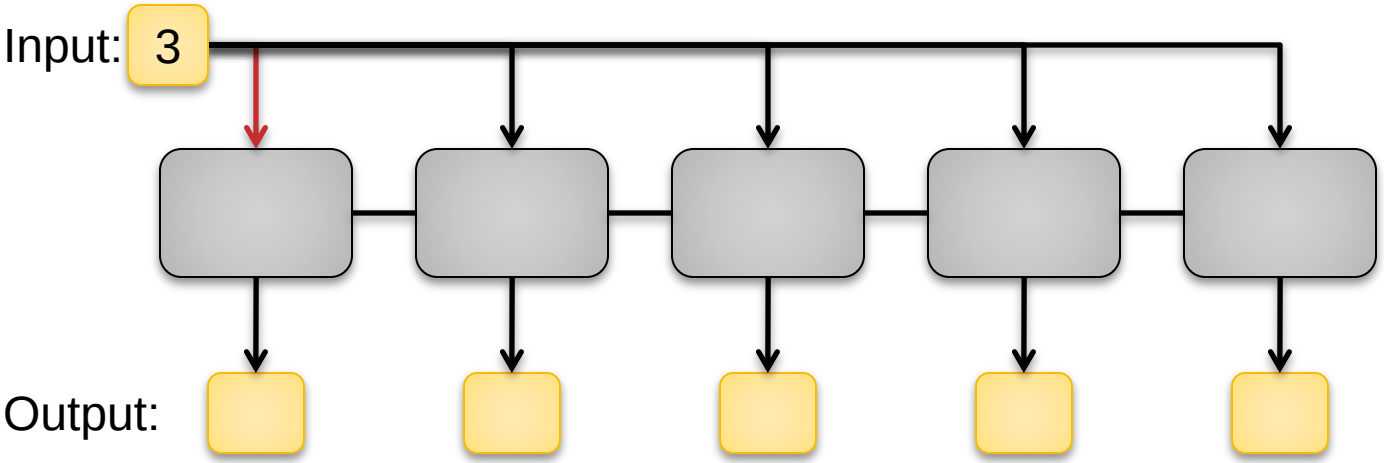
- Single clock latency, small logic & regular structure
- Streaming input & output
- Serial input, need higher throughput



Linear Sorters

- Sorted insertions
- Forwards incoming value to all nodes
- Each node shifts autonomously depending on neighbors' values

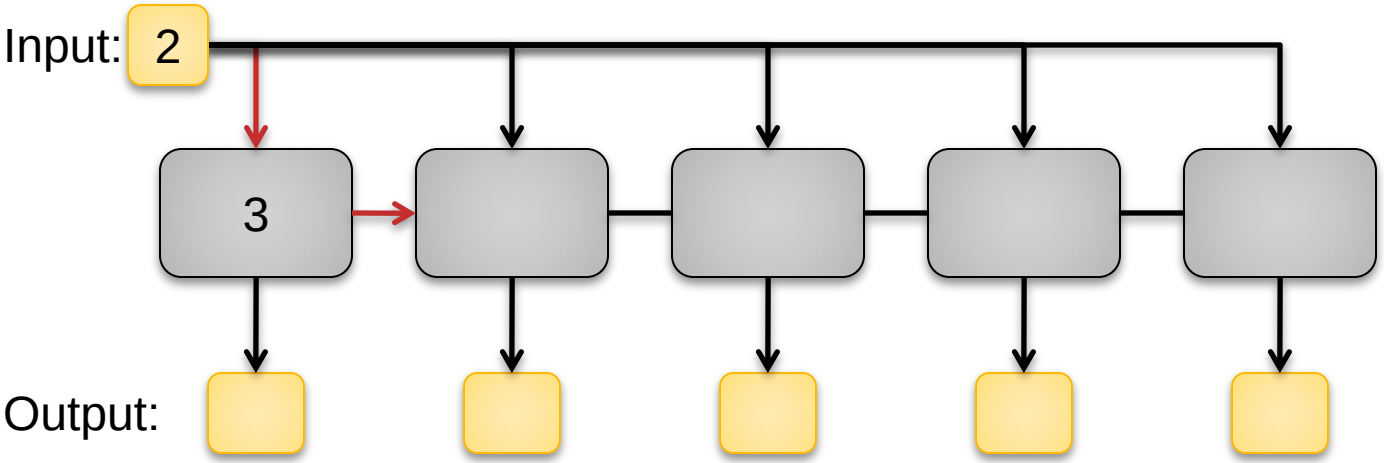
- Single clock latency, small logic & regular structure
- Streaming input & output
- Serial input, need higher throughput



Linear Sorters

- Sorted insertions
- Forwards incoming value to all nodes
- Each node shifts autonomously depending on neighbors' values

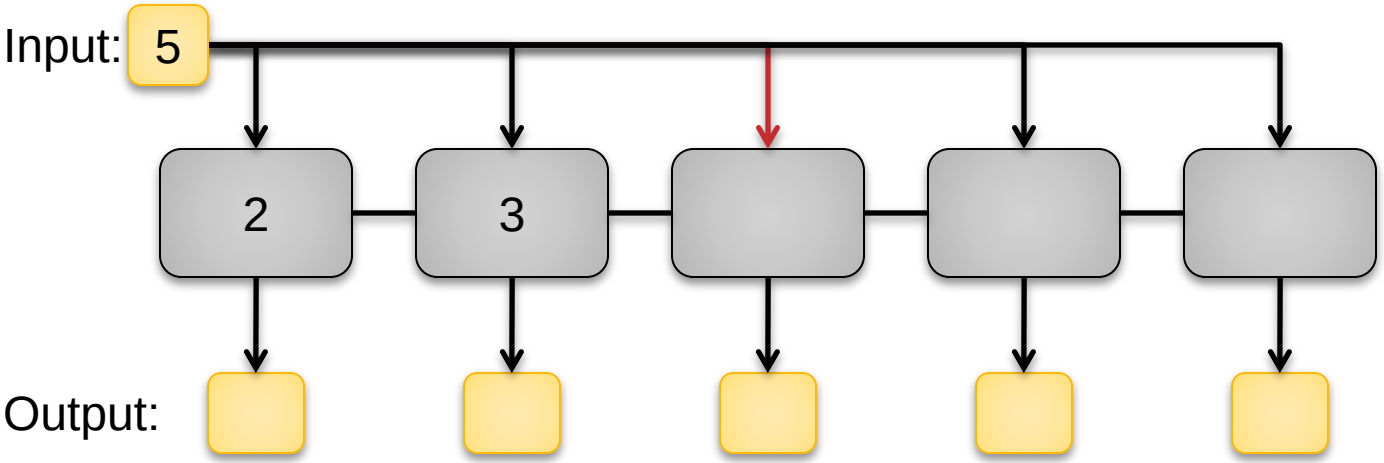
- Single clock latency, small logic & regular structure
- Streaming input & output
- Serial input, need higher throughput



Linear Sorters

- Sorted insertions
- Forwards incoming value to all nodes
- Each node shifts autonomously depending on neighbors' values

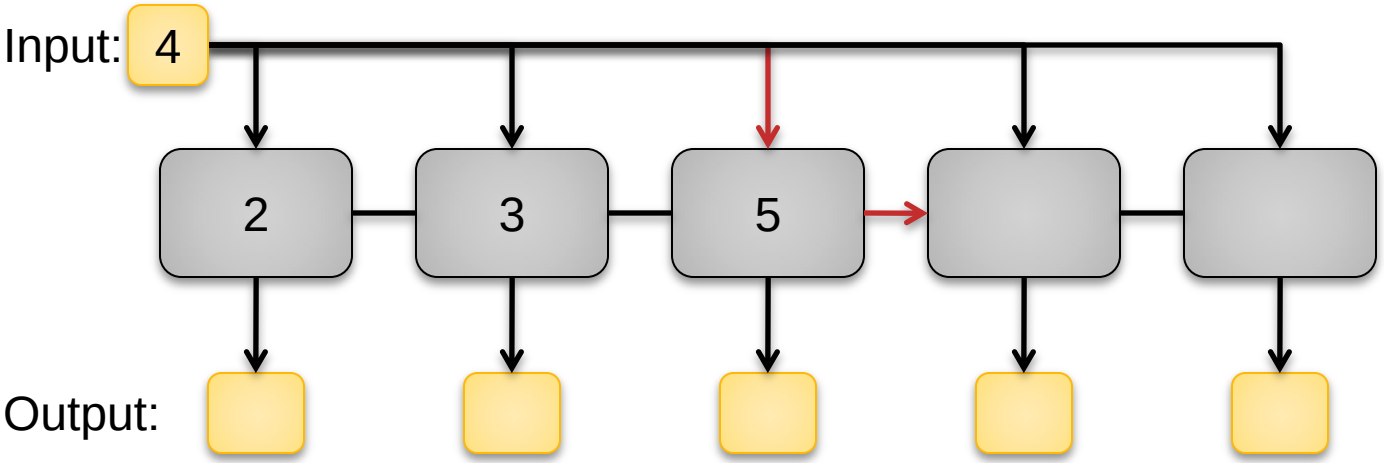
- Single clock latency, small logic & regular structure
- Streaming input & output
- Serial input, need higher throughput



Linear Sorters

- Sorted insertions
- Forwards incoming value to all nodes
- Each node shifts autonomously depending on neighbors' values

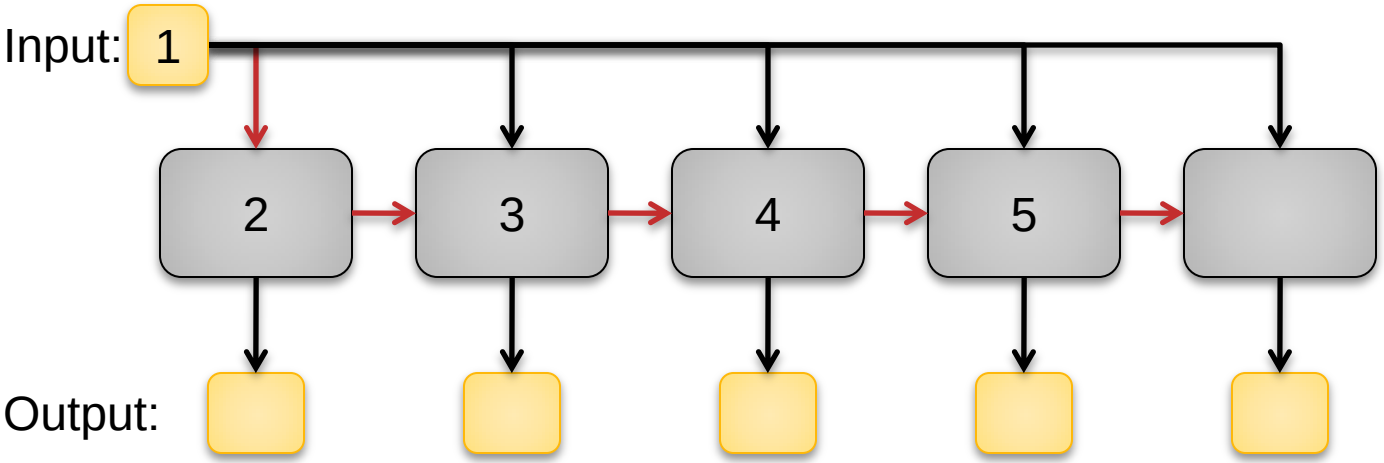
- Single clock latency, small logic & regular structure
- Streaming input & output
- Serial input, need higher throughput



Linear Sorters

- Sorted insertions
- Forwards incoming value to all nodes
- Each node shifts autonomously depending on neighbors' values

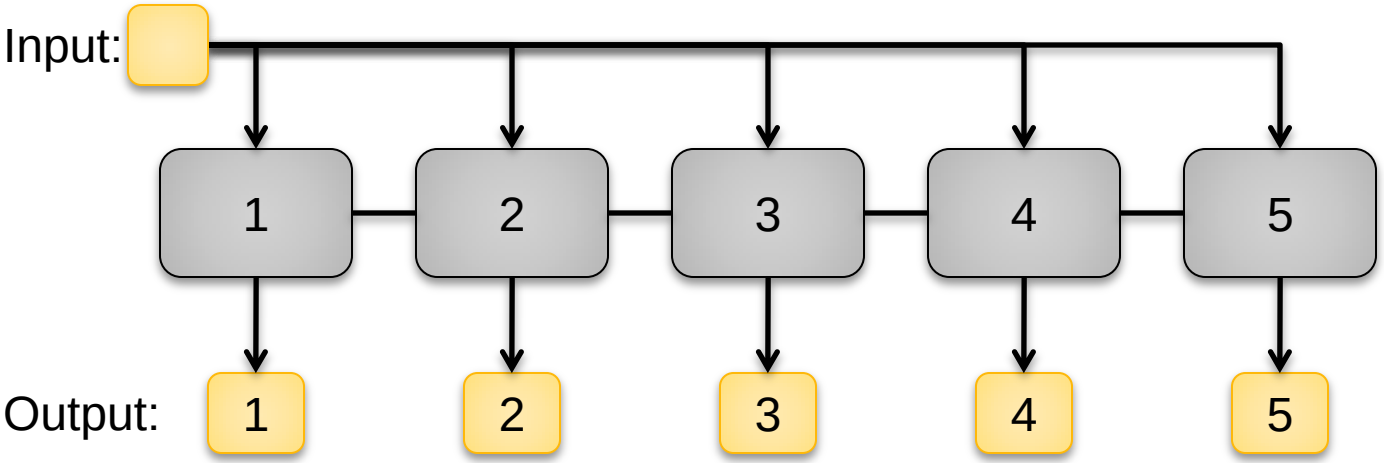
- Single clock latency, small logic & regular structure
- Streaming input & output
- Serial input, need higher throughput



Linear Sorters

- Sorted insertions
- Forwards incoming value to all nodes
- Each node shifts autonomously depending on neighbors' values

- Single clock latency, small logic & regular structure
- Streaming input & output
- Serial input, need higher throughput



Configurable Linear Sorter

Configurable Linear Sorter

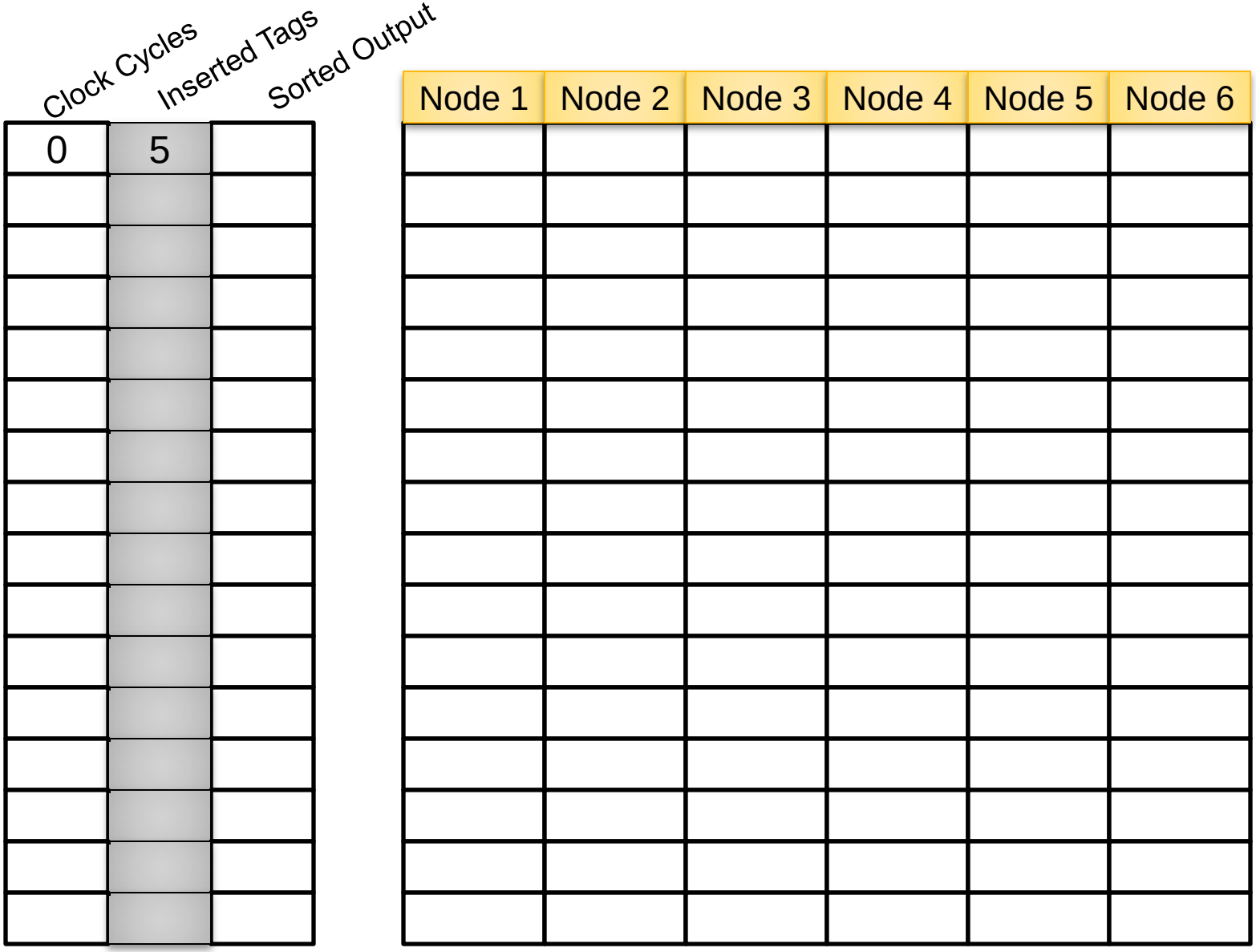
- Increase versatility for linear sorters
- Configurable:
  - Linear sorter depth
  - Sorting direction
  - Sort on tags (for example, timestamps) rather than data
  - User-defined data and tag size

Configurable Linear Sorter

Increase functionality for linear sorters

1. Detect full conditions
2. Buffer input while full
3. Retrieve output serially for streaming
4. Delete top value, freeing nodes
5. Augment with left shift functionality
6. Test tags before deleting them

Extended Linear Sorter System



Extended Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					

Extended Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				

Extended Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	
3	2	

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				
5	6	7			

Extended Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	
3	2	
4	1	

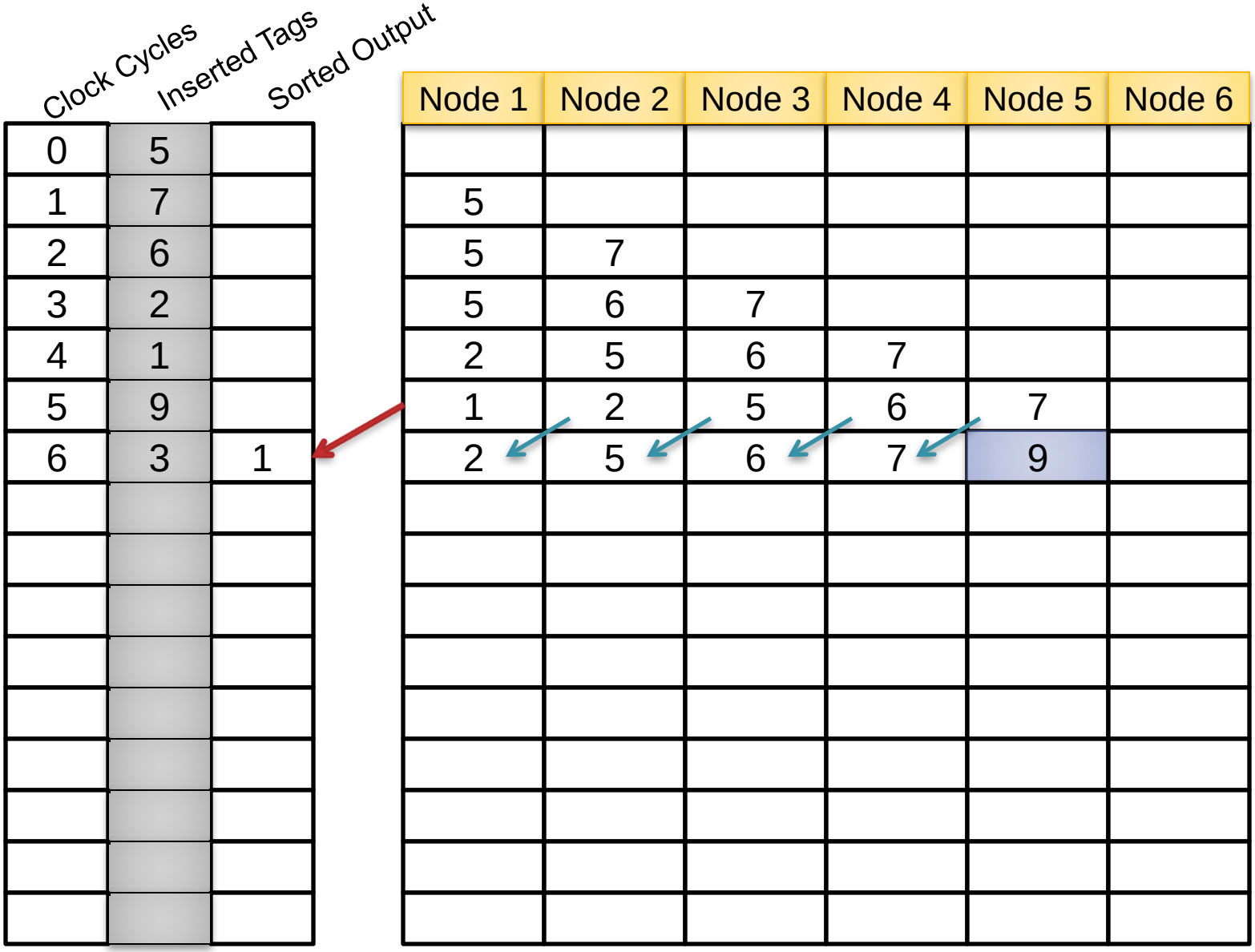
Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				
5	6	7			
2	5	6	7		

Extended Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	
3	2	
4	1	
5	9	

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				
5	6	7			
2	5	6	7		
1	2	5	6	7	

Extended Linear Sorter System



Extended Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	
3	2	
4	1	
5	9	
6	3	1
7	8	2

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				
5	6	7			
2	5	6	7		
1	2	5	6	7	
2	5	6	7	9	
3	5	6	7	9	

Extended Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	
3	2	
4	1	
5	9	
6	3	1
7	8	2
8	4	3

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				
5	6	7			
2	5	6	7		
1	2	5	6	7	
2	5	6	7	9	
3	5	6	7	9	
5	6	7	8	9	

Interleaved Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	
3	2	
4	1	
5	9	
6	3	1
7	8	2
8	4	3
9		

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				
5	6	7			
2	5	6	7		
1	2	5	6	7	
2	5	6	7	9	
3	5	6	7	9	
5	6	7	8	9	
4	5	6	7	8	9

Interleaved Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	
3	2	
4	1	
5	9	
6	3	1
7	8	2
8	4	3
9		
10		4

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				
5	6	7			
2	5	6	7		
1	2	5	6	7	
2	5	6	7	9	
3	5	6	7	9	
5	6	7	8	9	
4	5	6	7	8	9
5	6	7	8	9	

Interleaved Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	
3	2	
4	1	
5	9	
6	3	1
7	8	2
8	4	3
9		
10		4
11		5

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				
5	6	7			
2	5	6	7		
1	2	5	6	7	
2	5	6	7	9	
3	5	6	7	9	
5	6	7	8	9	
4	5	6	7	8	9
5	6	7	8	9	
6	7	8	9		

Extended Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	
3	2	
4	1	
5	9	
6	3	1
7	8	2
8	4	3
9		
10		4
11		5
12		6

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				
5	6	7			
2	5	6	7		
1	2	5	6	7	
2	5	6	7	9	
3	5	6	7	9	
5	6	7	8	9	
4	5	6	7	8	9
5	6	7	8	9	
6	7	8	9		
7	8	9			

Extended Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	
3	2	
4	1	
5	9	
6	3	1
7	8	2
8	4	3
9		
10		4
11		5
12		6
13		7

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				
5	6	7			
2	5	6	7		
1	2	5	6	7	
2	5	6	7	9	
3	5	6	7	9	
5	6	7	8	9	
4	5	6	7	8	9
5	6	7	8	9	
6	7	8	9		
7	8	9			
8	9				

Extended Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	
3	2	
4	1	
5	9	
6	3	1
7	8	2
8	4	3
9		
10		4
11		5
12		6
13		7
14		8

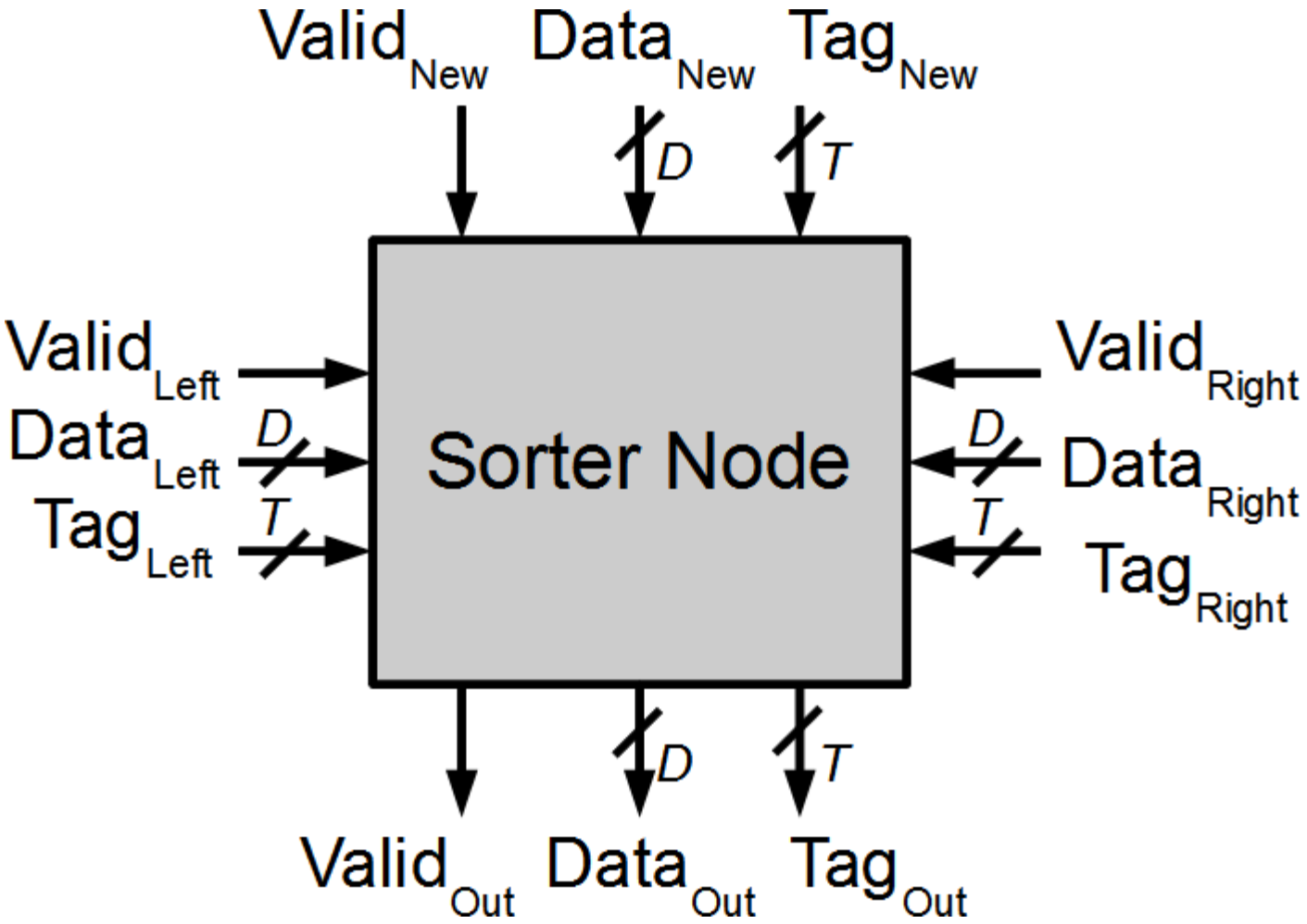
Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				
5	6	7			
2	5	6	7		
1	2	5	6	7	
2	5	6	7	9	
3	5	6	7	9	
5	6	7	8	9	
4	5	6	7	8	9
5	6	7	8	9	
6	7	8	9		
7	8	9			
8	9				
9					

Extended Linear Sorter System

Clock Cycles	Inserted Tags	Sorted Output
0	5	
1	7	
2	6	
3	2	
4	1	
5	9	
6	3	1
7	8	2
8	4	3
9		
10		4
11		5
12		6
13		7
14		8
15		9

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
5					
5	7				
5	6	7			
2	5	6	7		
1	2	5	6	7	
2	5	6	7	9	
3	5	6	7	9	
5	6	7	8	9	
4	5	6	7	8	9
5	6	7	8	9	
6	7	8	9		
7	8	9			
8	9				
9					

Sorter Node Architecture



Interleaved Linear Sorter

Interleaved Linear Sorter

- Increase throughput by using multiple linear sorters with parallel inputs
- Interleave parallel inputs into linear sorters through modulo arithmetic
- Distribute data evenly among linear sorters to avoid full conditions
- Service each linear sorter in round-robin fashion to resort their outputs

Interleaved Linear Sorter

- ILS width = 4
- Four parallel inputs
  - **{13, 04, 03, 10}**
- After interleaving mod 4
  - **{04, 13, 10, 03}**
  - [00, 01, 02, 03]

- But, what happens for inputs
  - **{07, 05, 01, 06}** ?

Interleaved Linear Sorter

Clock Cycles	Inserted Tags			
0	13	4	3	10

LS 0	LS 1	LS 2	LS 3

Interleaved Linear Sorter

Clock Cycles	Inserted Tags			
0	13	4	3	10
1	7	5	1	6

LS 0	LS 1	LS 2	LS 3
4	13	10	3

Inserted

Interleaved Linear Sorter

Clock Cycles	Inserted Tags			
0	13	4	3	10
1	7	5	1	6
2			1	

LS 0	LS 1	LS 2	LS 3
4	5	6	3
	13	10	7

Preempted

Inserted

Interleaved Linear Sorter

Clock Cycles	Inserted Tags			
0	13	4	3	10
1	7	5	1	6
2			1	
3	2	9	12	15

LS 0	LS 1	LS 2	LS 3
4	1	6	3
	5	10	7
	13		

Preempted

Inserted

Interleaved Linear Sorter

Clock Cycles	Inserted Tags			
0	13	4	3	10
1	7	5	1	6
2			1	
3	2	9	12	15
4	11	0	14	8

LS 0	LS 1	LS 2	LS 3
4	1	2	3
12	5	6	7
	9	10	15
	13		

Preempted

Inserted

Interleaved Linear Sorter

Clock Cycles	Inserted Tags			
0	13	4	3	10
1	7	5	1	6
2			1	
3	2	9	12	15
4	11	0	14	8
5				8

LS 0	LS 1	LS 2	LS 3
0	1	2	3
4	5	6	7
12	9	10	11
	13	14	15

Preempted

Inserted

Interleaved Linear Sorter

Clock Cycles	Inserted Tags			
0	13	4	3	10
1	7	5	1	6
2			1	
3	2	9	12	15
4	11	0	14	8
5				8
6				

LS 0	LS 1	LS 2	LS 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Preempted

Inserted

Interleaved Linear Sorter

Clock Cycles	Inserted Tags			
0	13	4	3	10
1	7	5	1	6
2			1	
3	2	9	12	15
4	11	0	14	8
5				8
6				

LS 0	LS 1	LS 2	LS 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Preempted

Inserted

Input Distribution and Latency

- Assume uniformly distributed data
- With $W = 2$, 50% chance of interleaving delays, adding an extra clock cycle
- With $W > 2$, more probabilities of stalling, adding 1+ clock cycles

Average latency for Interleaved Linear Sorters of length *W*

ILS Width *W*	Clock Cycles
1	1.000
2	1.500
4	2.125
8	2.597
16	3.078

Larger ILS widths allow parallel sorting and increase throughput. However, they have complex routing and additional delays

Output Logic

- Accumulate values before output becomes relevant
- Increase linear sorter depth to accumulate more data

- Re-sort top values from each linear sorter to ensure continuity (particularly for contiguous values)
- Service in round-robin fashion: Test the top tag of each linear sorter before deleting

Streaming output

LS 0	LS 1	LS 2	LS 3
0	1	2	3
4	5	6	7
8	9	14	11
	13		15

OK

OK

Output {8,9}; Wait for 10

Hardware Implementation

FPGA Area

- Xilinx Virtex-5 FPGA
- 8-bit tags, 8-bit data
- 17 slices per sorter node
- Linear Sorter depth of 16

Interleaved Linear Sorter FPGA Area

Linear Sorters, *W*	Total Slices	Slices/Node	Area Overhead	Interleaving Area (slices)
1	278	17.4	2.3%	7
2	641	20.0	17.6%	113
4	1294	20.2	18.8%	243
8	2612	20.4	20.0%	522
16	5250	20.5	20.6%	1081

FPGA Throughput

- f_{clk} x ILS width *W*
- Includes logic and routing delay for interleaving data for *W* linear sorters
- Averaged for sorter depth from 1 up to 256 nodes

Interleaved Linear Sorter FPGA Frequency & Throughput

Linear Sorters, *W*	Frequency (MHz)	Throughput (millions/sec)
1	299	299
2	275	550
4	275	1101
8	132	1058
16	40	645

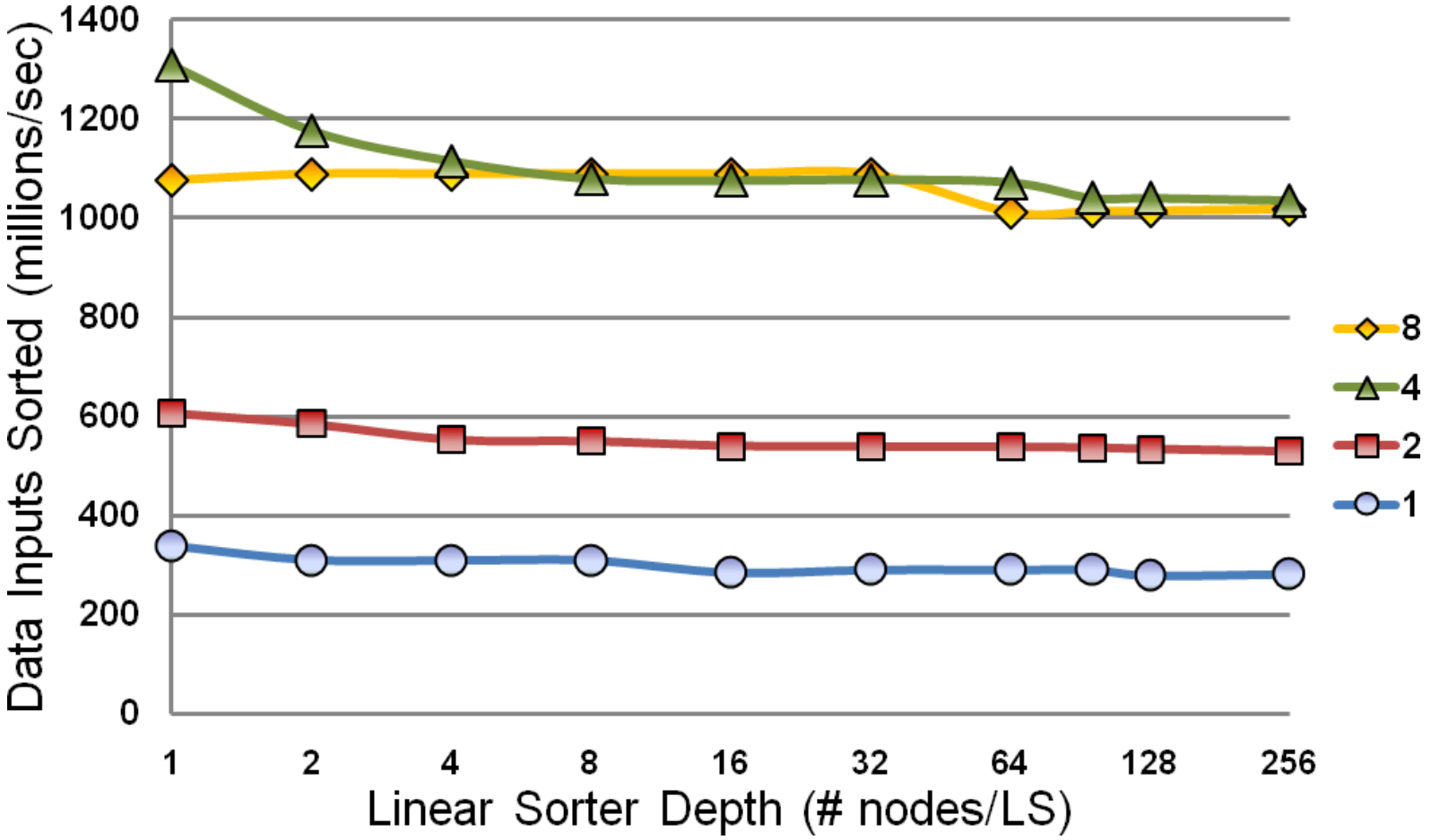
FPGA Throughput

- W=16, large logic & routing delays at inputs
- First three cases need single 6-input LUT for routing. Two and four LUTs needed for W=8 and W=16, respectively.

Interleaved Linear Sorter FPGA Frequency & Throughput

Linear Sorters, *W*	Frequency (MHz)	Throughput (millions/sec)
1	299	299
2	275	550
4	275	1101
8	132	1058
16	40	645

Maximum ILS Throughput



Average speedups of 1.0, 1.8, 3.7 and 3.5 against single linear sorter

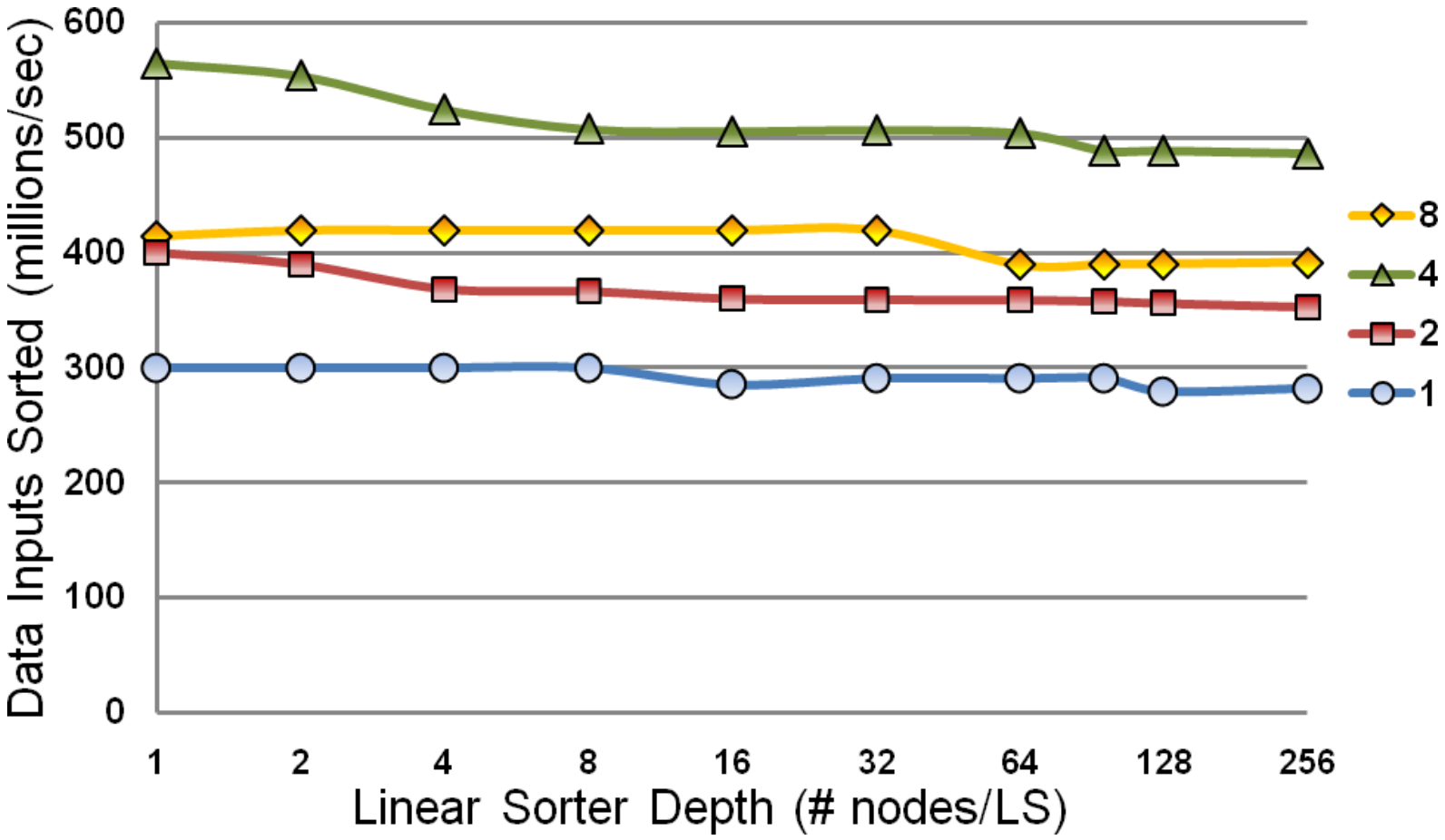
Throughput considerations

- Interleaving contention results in an average latency which increases with ILS width *W*

Average latency for Interleaved Linear Sorters of length *W*

ILS Width *W*	Clock Cycles
1	1.000
2	1.500
4	2.125
8	2.597
16	3.078

Normalized ILS Throughput



Average speedups of 1.0, 1.3, 1.8 and 1.4 against single linear sorter

Virtex II-Pro implementation

- 100 MHz bus frequency
- Data resided on BlockRAMs
- Tested Interleaved Linear Sorter *W*=4

- Compared against MicroBlaze running quicksort in C
- Timing includes bus arbitration, read
- and writes over the OPB
- Final result is saved in BlockRAM

Three test scenarios

1. MicroBlaze BRAM write & read-back
   - MB writes BRAM unsorted data
   - MB sends start signal to ILS
   - MB reads back sorted values over OPB
2. MicroBlaze BRAM write
   - Same as scenario 1
   - No need for read back into MB
3. No MicroBlaze
   - Hardware-only streaming output approach
   - No need for OPB requests
   - Output consumed by other hardware components immediately

ILS speedup over MicroBlaze

System	Clock cycles	Speedup
MicroBlaze quicksort	49,982	1
ILS 1 – MB write & read	2272	22
ILS 2 – MB write	732	68
ILS 3 – Hardware-only	30	1666

- *32-bit data*
- *16-bit tags*
- *64 values*
- *ILS width of 4*

ILS speedup against Sorting Network

System	Execution time (ns)	Speedup
Batcher odd-even	95	1.0
ILS – Hardware only	123	0.8

- *16-bit data-tags*
- *32 values*
- *ILS width of 4*

- Sorting network requires static data set
- Re-sorts the full set of data upon a single new insertion

Conclusions

Conclusions

- Linear sorters appropriate to overcome sorting network disadvantages, but limited throughput
- Interleaved Linear Sorters allow high throughput, configuration of width, depth, tag and data size to match system requirements
- ILS speedup of 1.8 over regular linear sorter (3.7 for best case)
- ILS speedup of 68 against embedded software counterpart (1666 for best case)

Questions