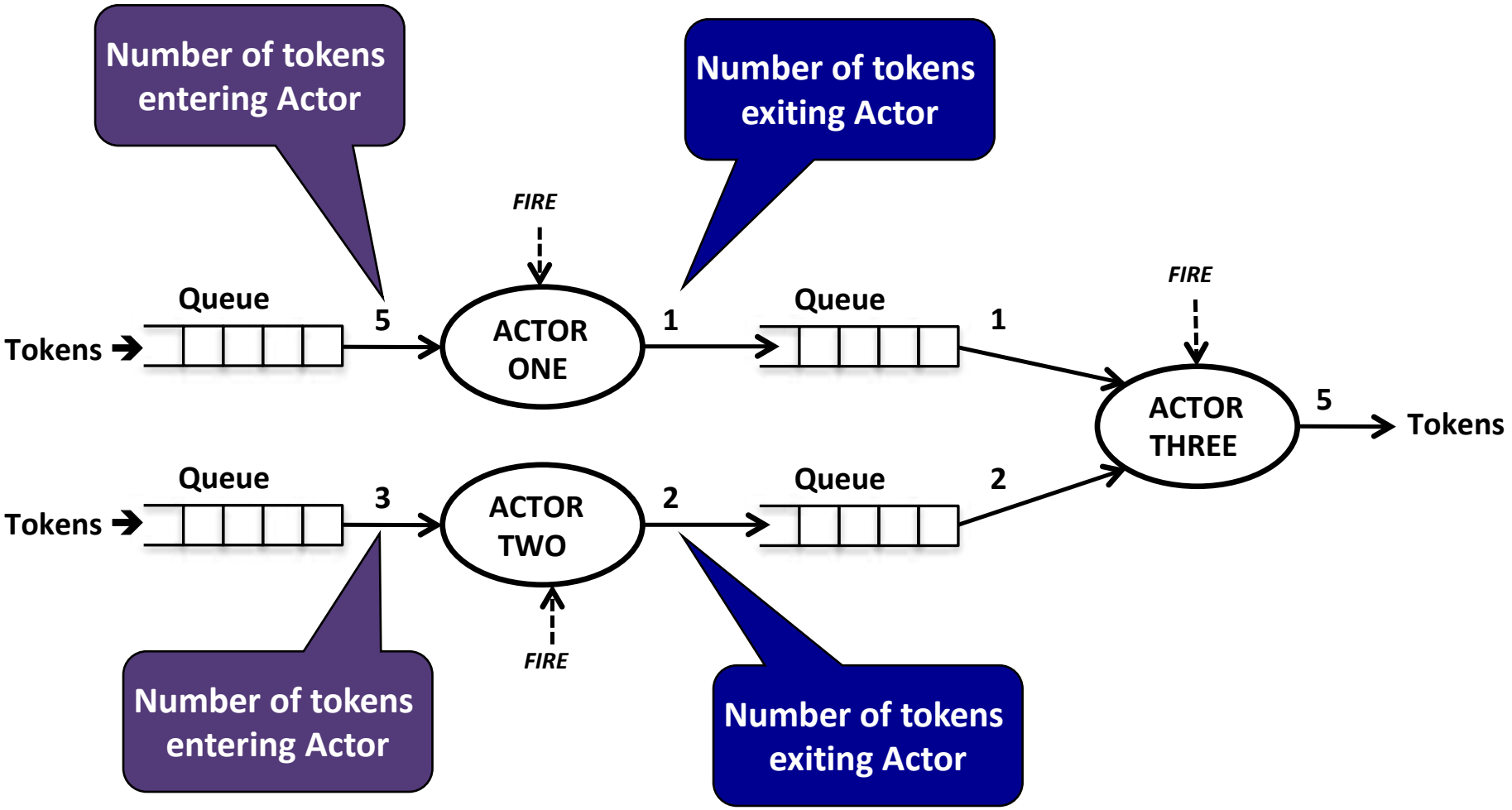
Lecture 18:
VLSI Building Blocks

Mark McDermott
Electrical and Computer Engineering
The University of Texas at Austin

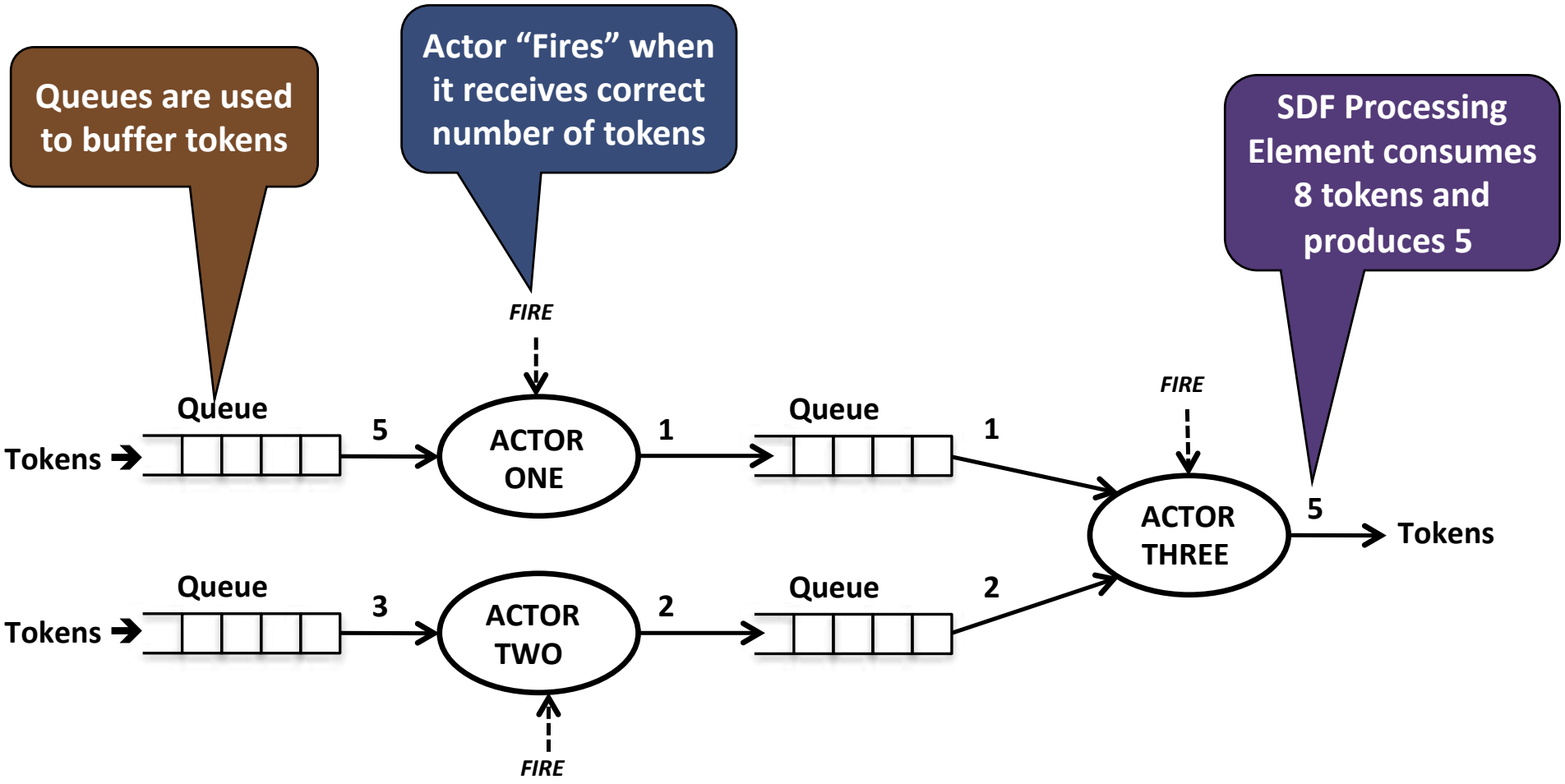
Agenda

- System overview
  - Dataflow programming paradigm
- Dataflow Processing Element (DPE)
  - Queues (FIFO), Stacks (LIFO), Queued-Stacks
  - Datapaths
  - Lookup Tables
  - Microcode Control
  - Finite State Machines (FSM)
- On-chip Networks
  - Serializers & Deserializers (SERDES)
  - Cross bar switches
  - Arbiters

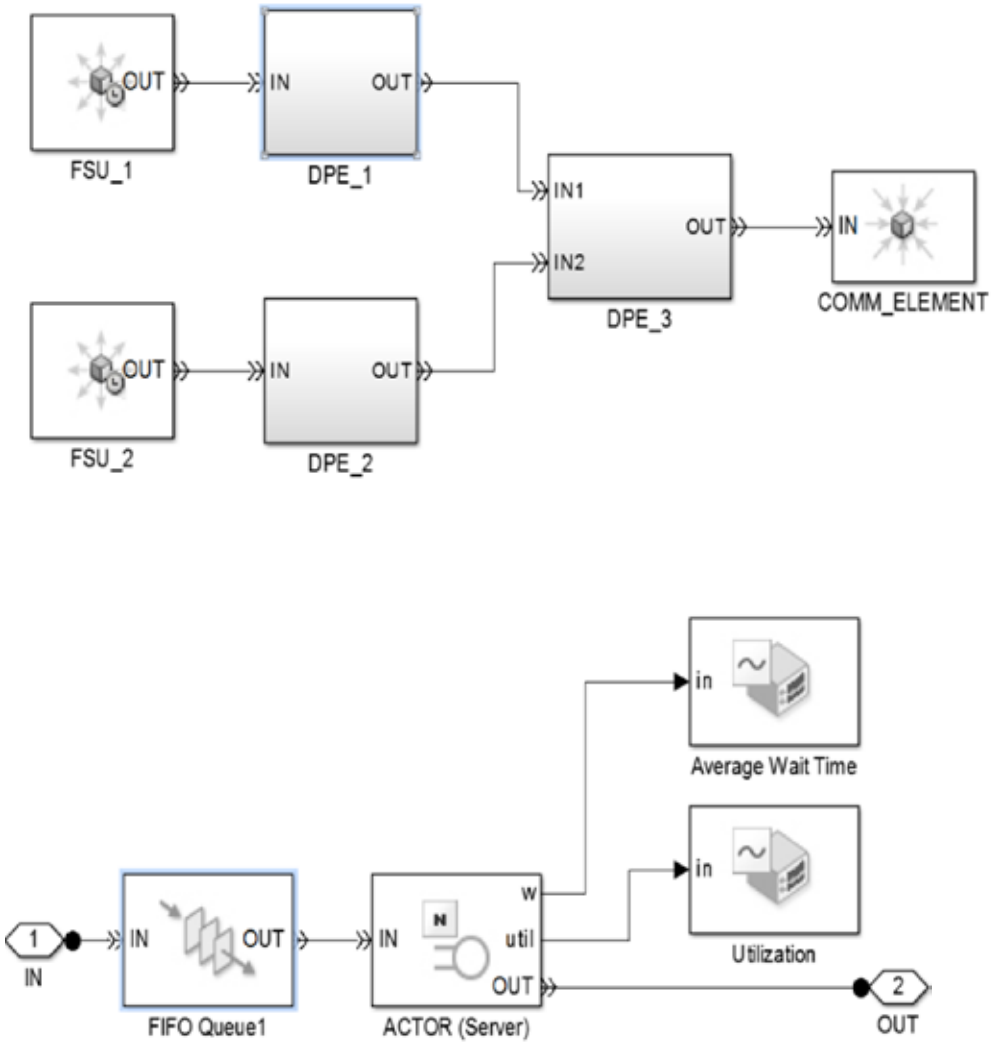
Synchronous Dataflow (SDF) Tutorial



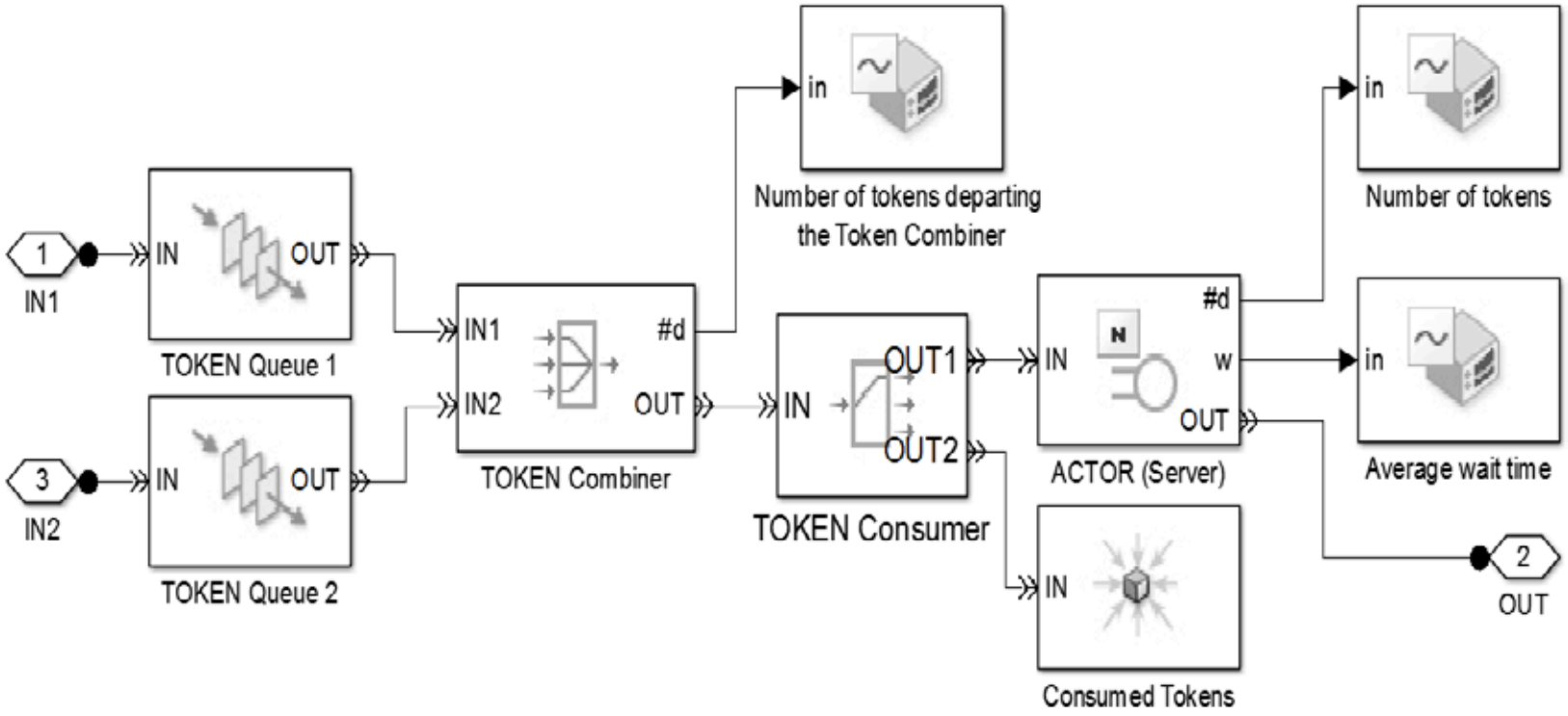
Synchronous Dataflow (SDF) Tutorial



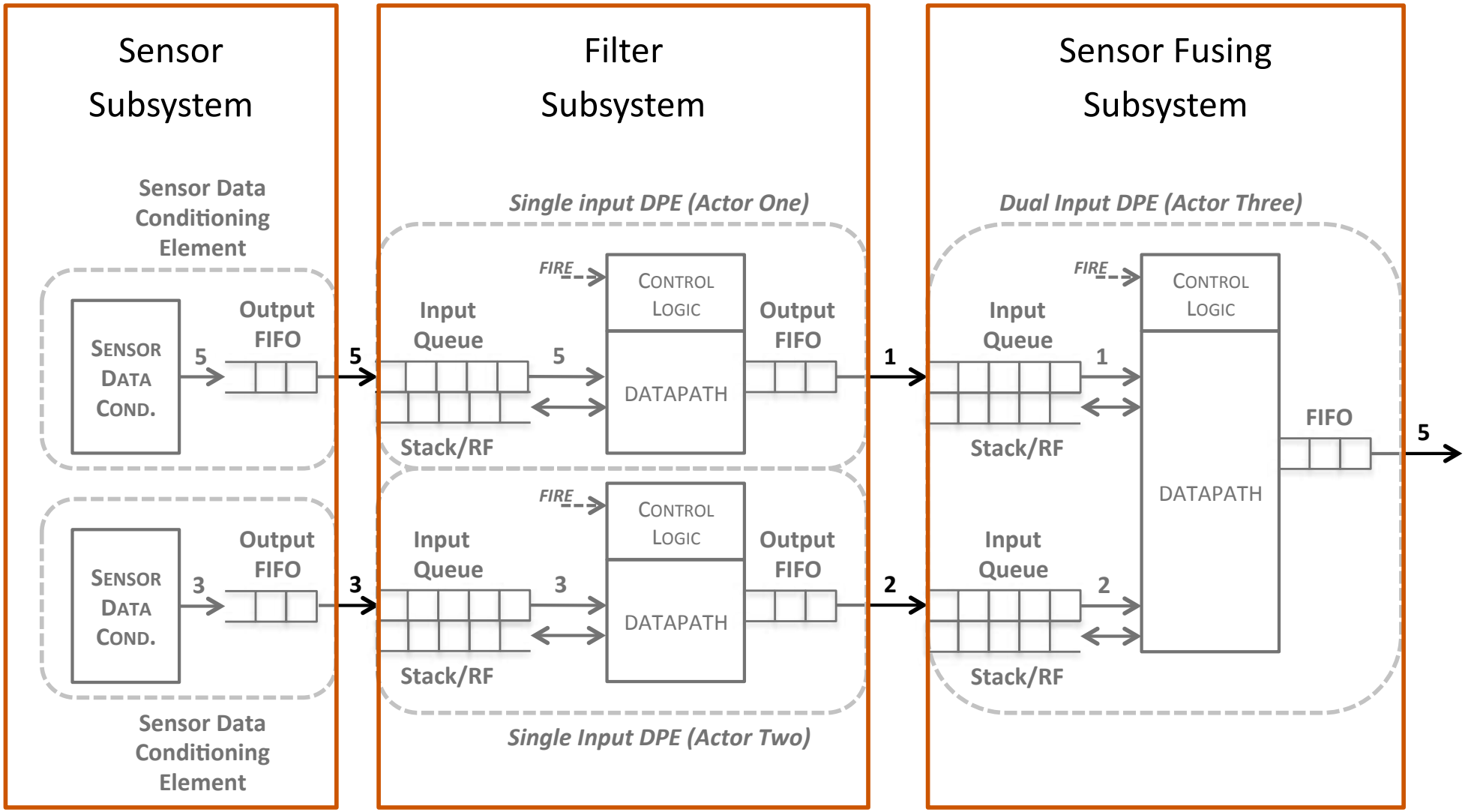
Programming a Dataflow Engine



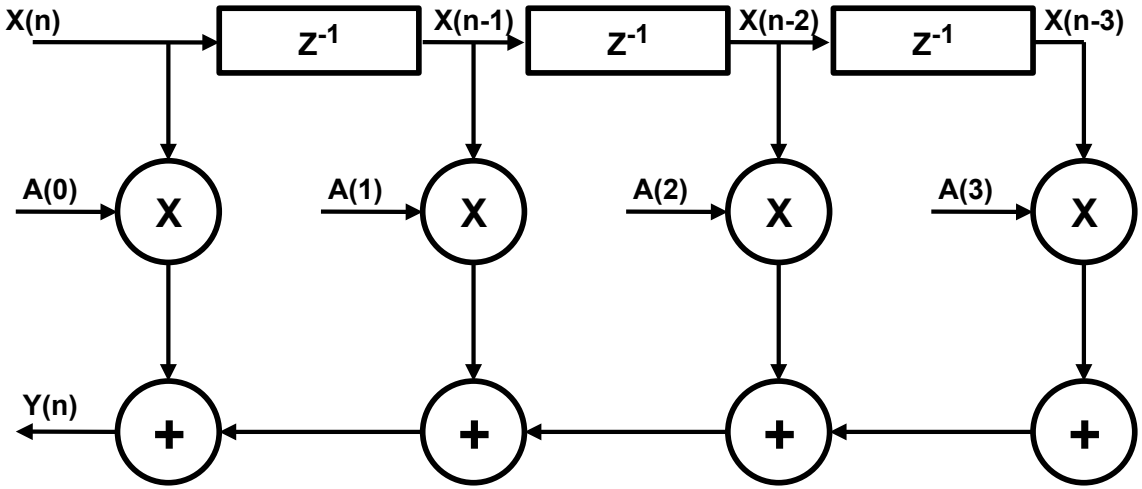
Programming a Dataflow Engine

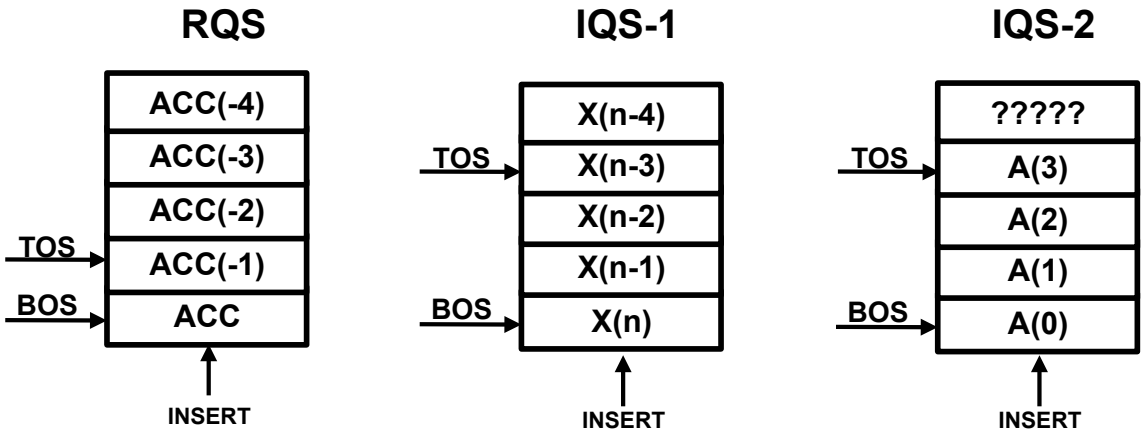


Dataflow Processing Element (DPE)

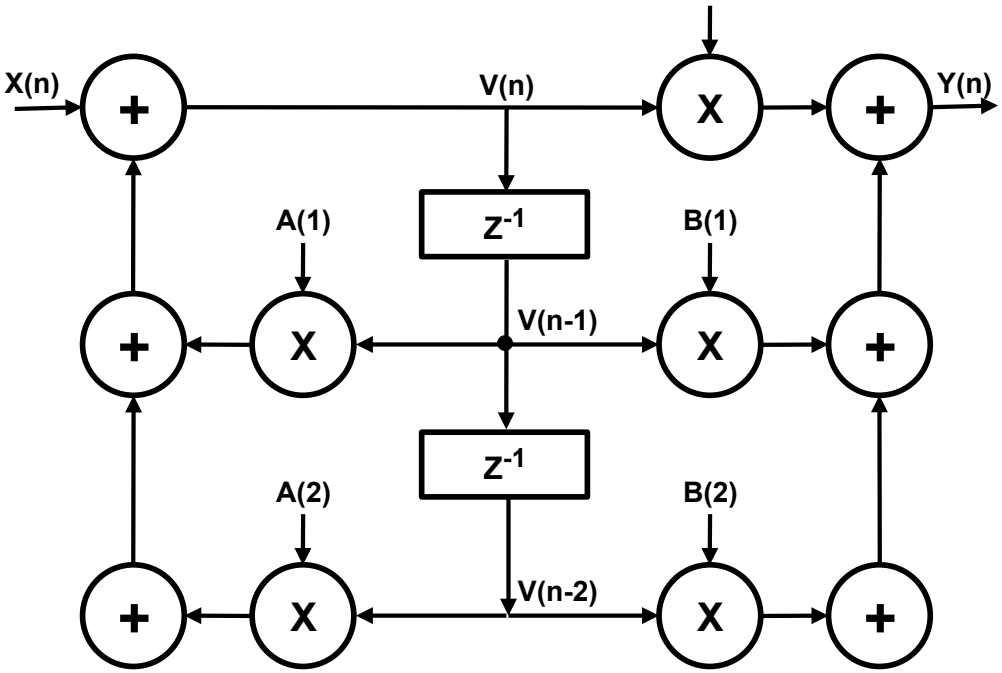


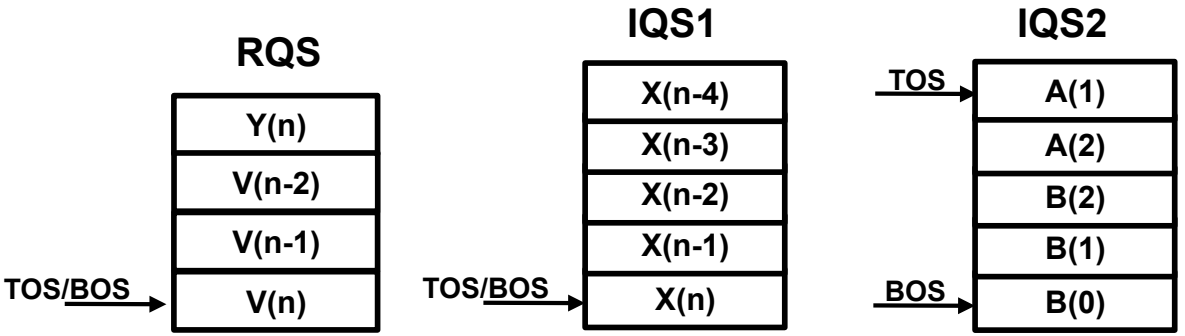
Filter Subsystem: FIR Filter



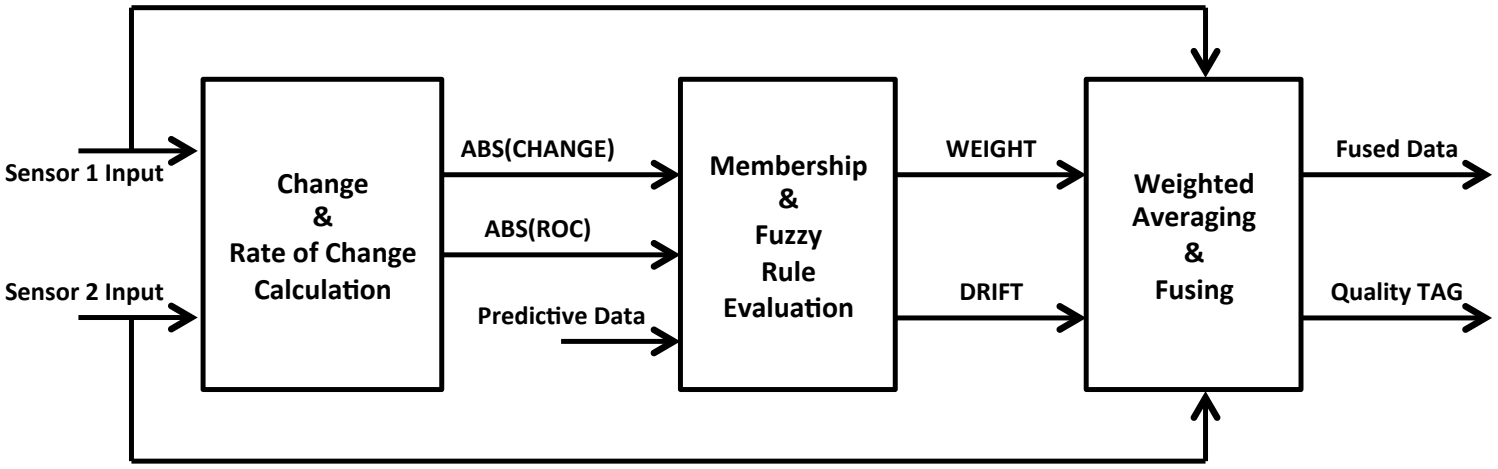


Filter Subsystem: IIR Filter

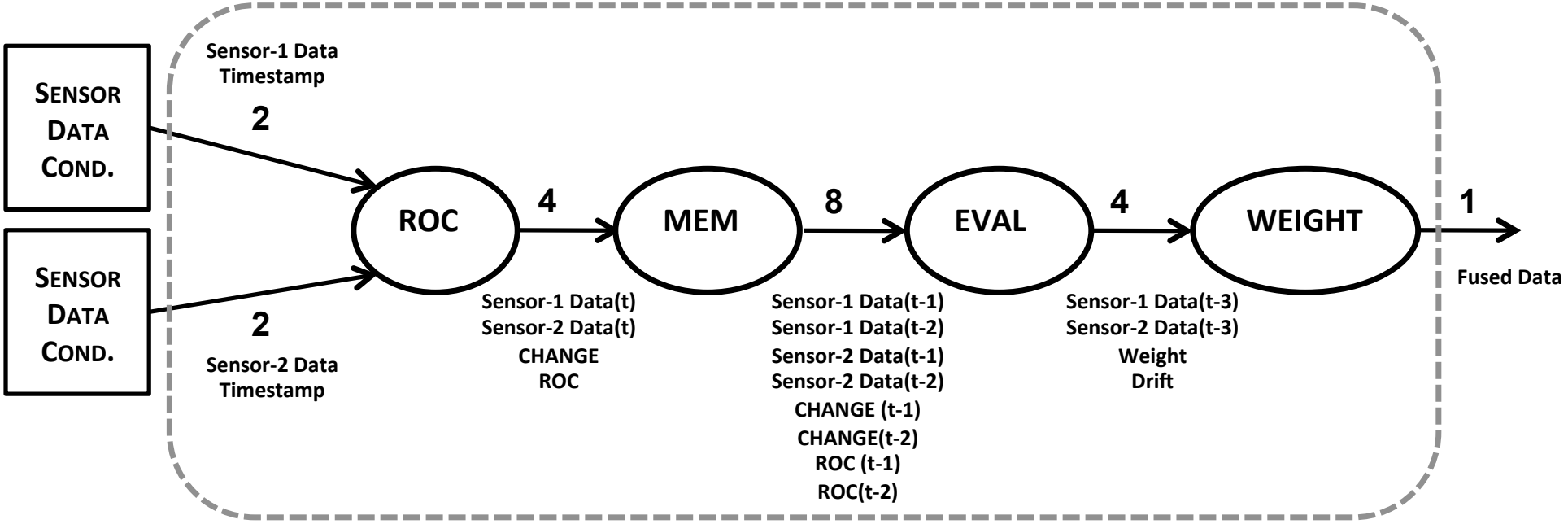




Fuzzy Logic Based Sensor Fusing DPE



Dual Input DPE (Four Actors: ROC, MEM, EVAL, WEIGHT)



Agenda

- System overview
- **Dataflow Processing Element (DPE)**
  - **Queues (FIFO), Stacks (LIFO), Queued-Stacks**
  - **Datapaths**
  - **Lookup Tables**
  - **Microcode Control**
  - **Finite State Machines (FSM)**
- On-chip Networks
  - Serializers & Deserializers (SERDES)
  - Cross bar switches
  - Arbiters

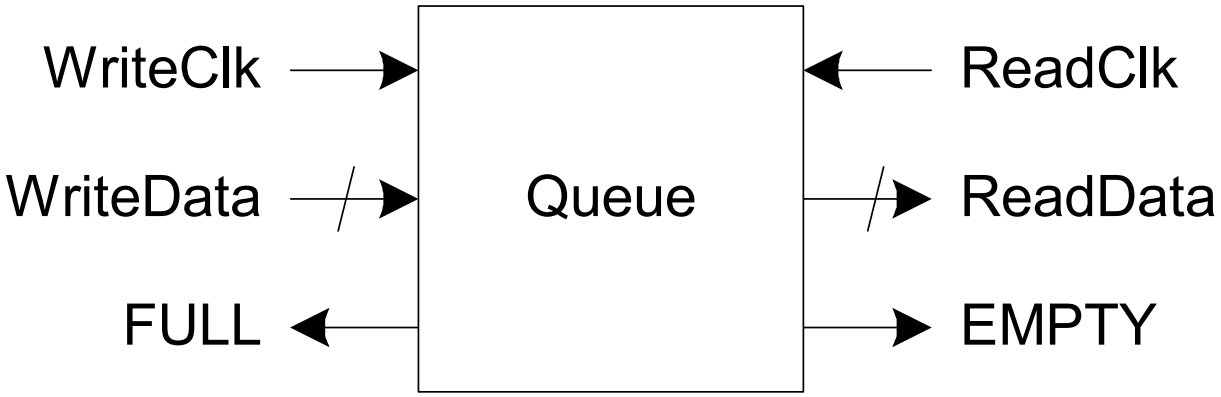
FIFO/Queue & LIFO/Stack

- **First In First Out (FIFO)**
  - Initialize read and write pointers to first element
    - Queue is EMPTY
  - On write, increment write pointer
  - If write almost catches read, Queue is FULL
  - On read, increment read pointer

- **Last In First Out (LIFO)**
  - Also called a stack
  - Use a single stack pointer for read and write

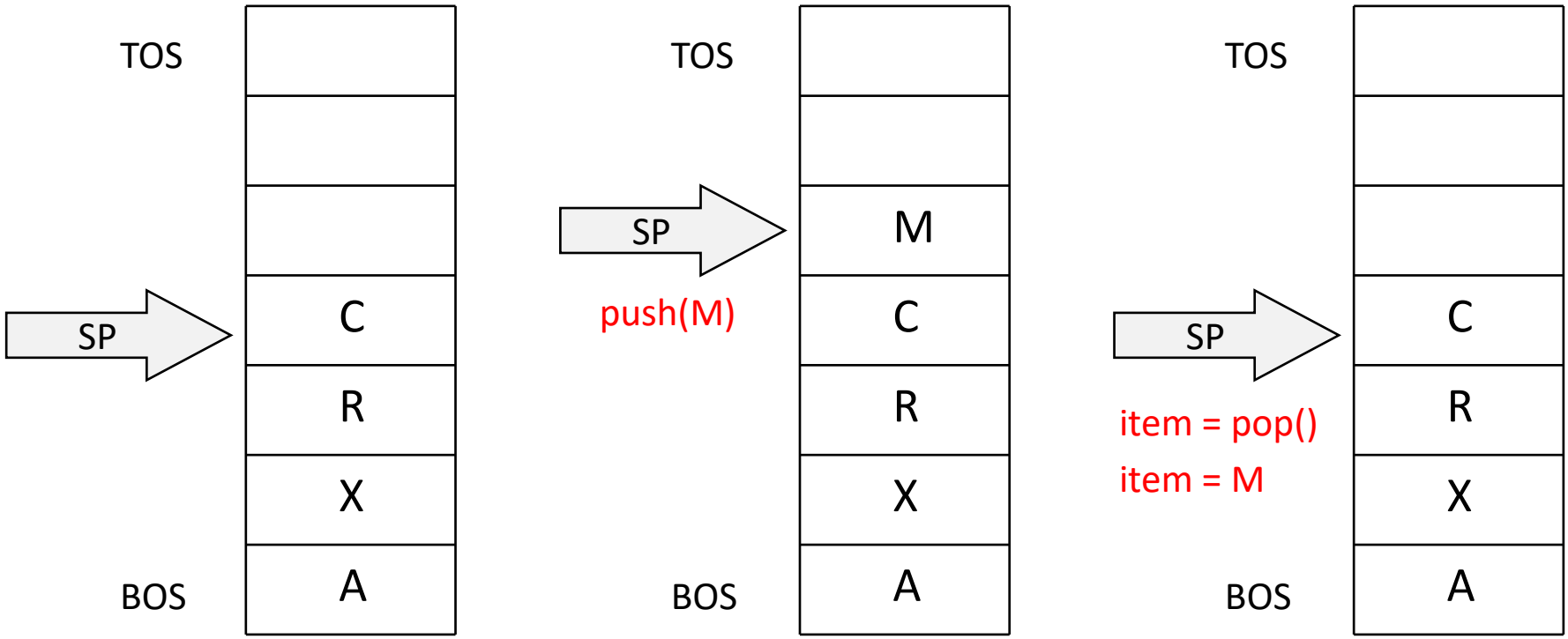
FIFO/Queues

- Queues allow data to be read and written at different rates.
- Read, Write each use their own clock, data
- Queue indicates whether it is full or empty
- Build with multi-ported SRAM and read/write counters (pointers)



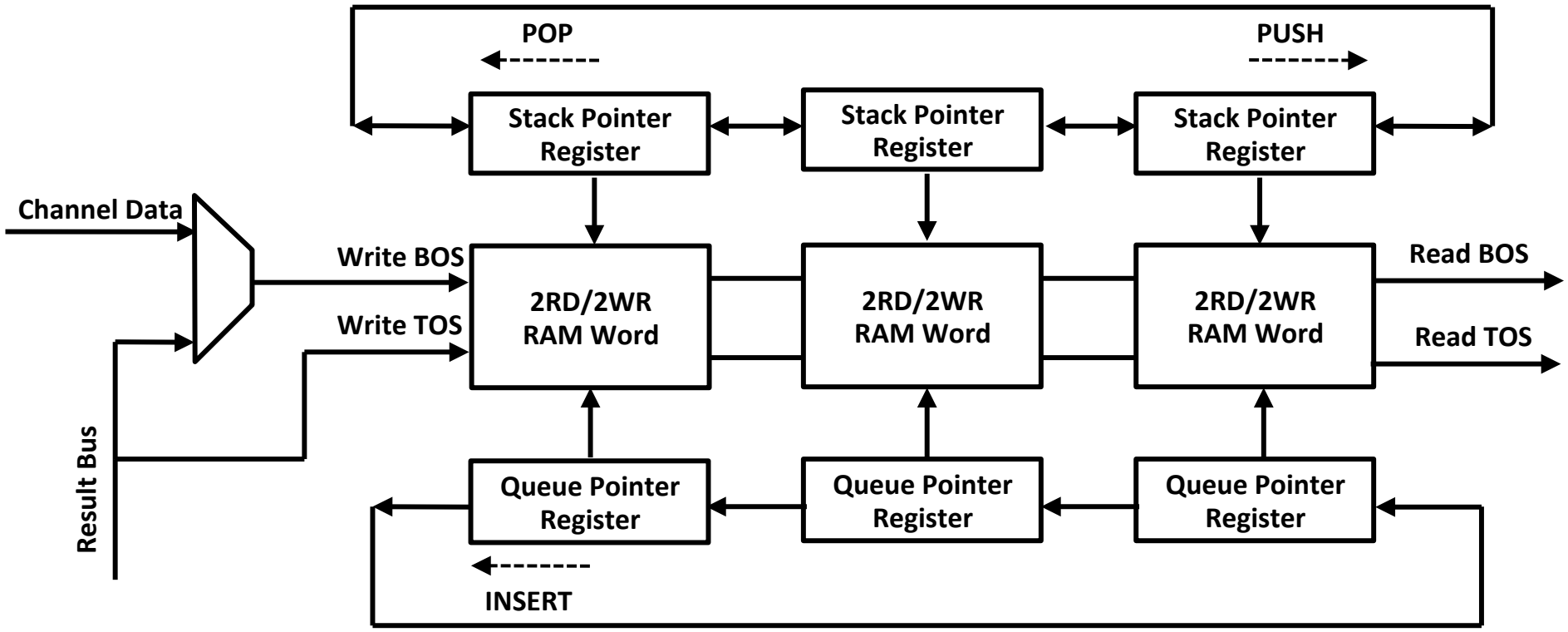
LIFO/Stack

- Stack uses a single pointer that points to last valid data entry.
- Pointer is pre-incremented before write during a push op
- Pointer is post-decremented after read during a pop op



Queued-Stack (QS)

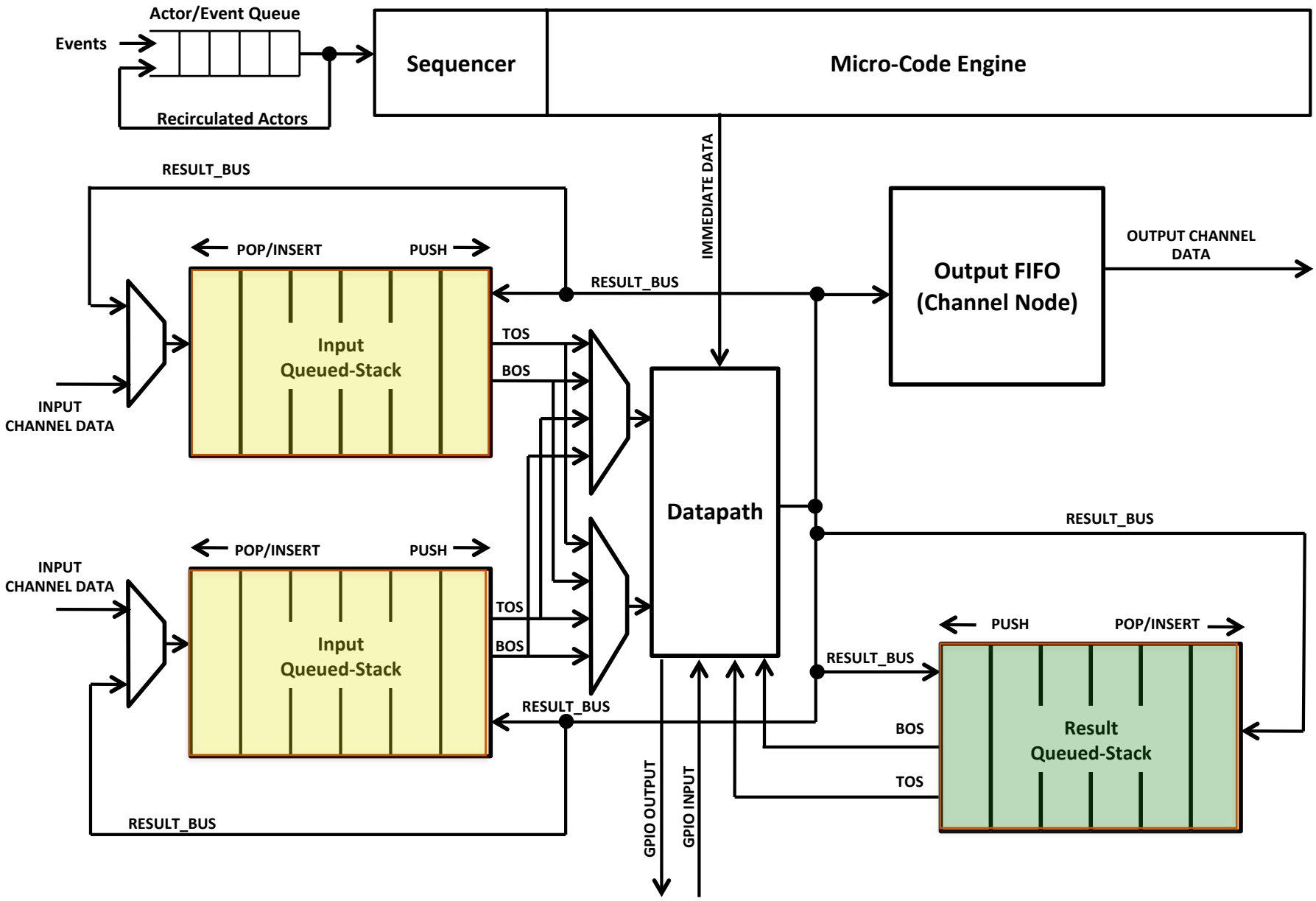
- Merges function of a Queue and a Stack into single element
- Uses two circular pointers and 2 port R/W memory
- Depth of the QS is determined by the algorithm executed by the DPE



QS Operations

Operation	Description
PUSH	Rotate TOS pointer right and write result-bus value to new TOS
POP	Rotate TOS pointer left (w/o write)
POP_WR	Rotate TOS pointer left and write result-bus value to new TOS
INS	Rotate BOS pointer left and write result-bus value to BOS
INS_NW	Rotate BOS pointer left (w/o write)
PUSH_NW	Rotate TOS pointer right (w/o write)
TOP	Write result bus value to TOS w/o rotating pointer
BOT	Write result bus value to BOS w/o rotating pointer
TOP_BOT	Write result bus value to TOS/BOS w/o rotating pointers
PUSH_INS	Rotate both pointers and write result-bus value to TOS/BOS
POP_BOT	Rotate TOS pointer left and write result-bus to BOS
POP_INS	Rotate TOS pointer left, rotate BOS left and write result-bus to new BOS
POP_WR_BOT	Rotate TOS pointer left and write result-bus to BOS and to new TOS
PUSH_NW_BOT	Rotate TOS pointer right and write result-bus to BOS
TOP_INS	Rotate BOS pointer left and write result-bus to TOS and to new BOS
NOP	No Operation

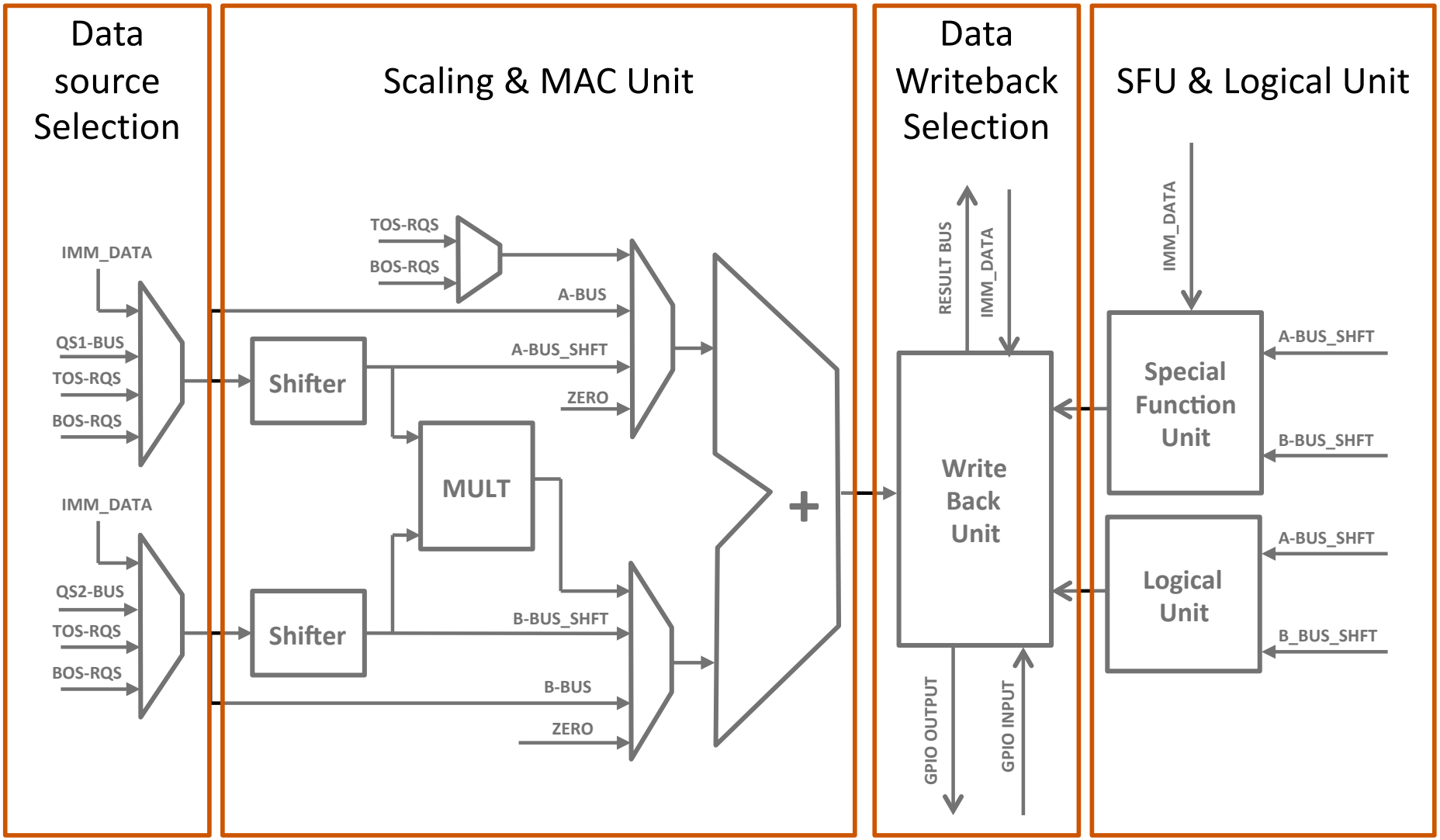
DPE using multiple Queued-Stacks



Agenda

- System overview
- Dataflow Processing Element (DPE)
  - Queues (FIFO), Stacks (LIFO), Queued-Stacks
  - **Datapaths**
  - Lookup Tables
  - Microcode Control
  - Finite State Machines (FSM)
- On-chip Networks
  - Serializers & Deserializers (SERDES)
  - Cross bar switches
  - Arbiters

DPE Datapath



Datapath Operations

SHIFT MODE A <44>	SHIFT A <38:34>	SHIFT TC A <32>	DATA TC <19>	MSB OP	OPERATION
1	0-31	0	-	-	LEFT ARITHMETIC SHIFT, LOGIC 0 PADDING
1	0-31	1	-	0	LEFT ARITHMETIC SHIFT, LOGIC 0 PADDING
1	0-31	1	0	0	RIGHT ARITHMETIC SHIFT, LOGIC 0 PADDING
1	0-31	1	1	1	RIGHT ARITHMETIC SHIFT, SIGN EXT. PADDING
0	0-31	0	-	0	LEFT BARREL SHIFT
0	0-31	1	-	1	RIGHT BARREL SHIFT

MULT MODE <31>	DATA TC <19>	OPERATION
0	-	NO OPERATION
1	0	UNSIGNED MULTIPLY
1	1	SIGNED MULTIPLY

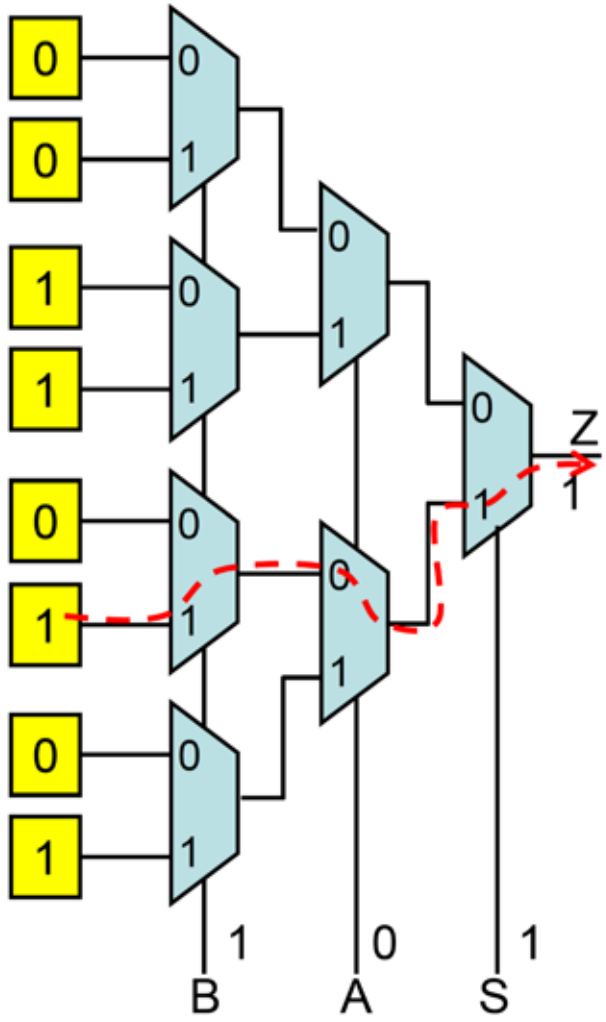
ADD/SUB <28>	SAT <30>	DATA TC <19>	OPERATION
0	0	0	UNSIGNED ADD
0	0	1	SIGNED ADD
0	1	0	SATURATED UNSIGNED ADD
0	1	1	SATURATED SIGNED ADD
1	0	0	UNSIGNED SUBTRACT
1	0	1	SIGNED SUBTRACT
1	1	0	SATURATED UNSIGNED SUBTRACT
1	1	1	SATURATED SIGNED SUBTRACT

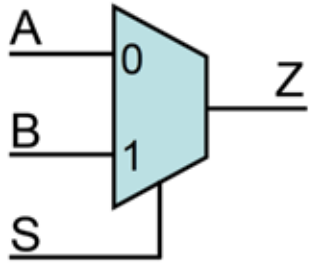
Agenda

- System overview
- Dataflow Processing Element (DPE)
  - Queues (FIFO), Stacks (LIFO), Queued-Stacks
  - Datapaths
  - **Lookup Tables**
  - Microcode Control
  - Finite State Machines (FSM)
- On-chip Networks
  - Serializers & Deserializers (SERDES)
  - Cross bar switches
  - Arbiters

Lookup Table (LUT)

- **Combinatorial logic is stored in Look-Up Tables (LUTs)**
  - Also called Function Generators (FGs)
  - Capacity is limited by the number of inputs, not by the complexity



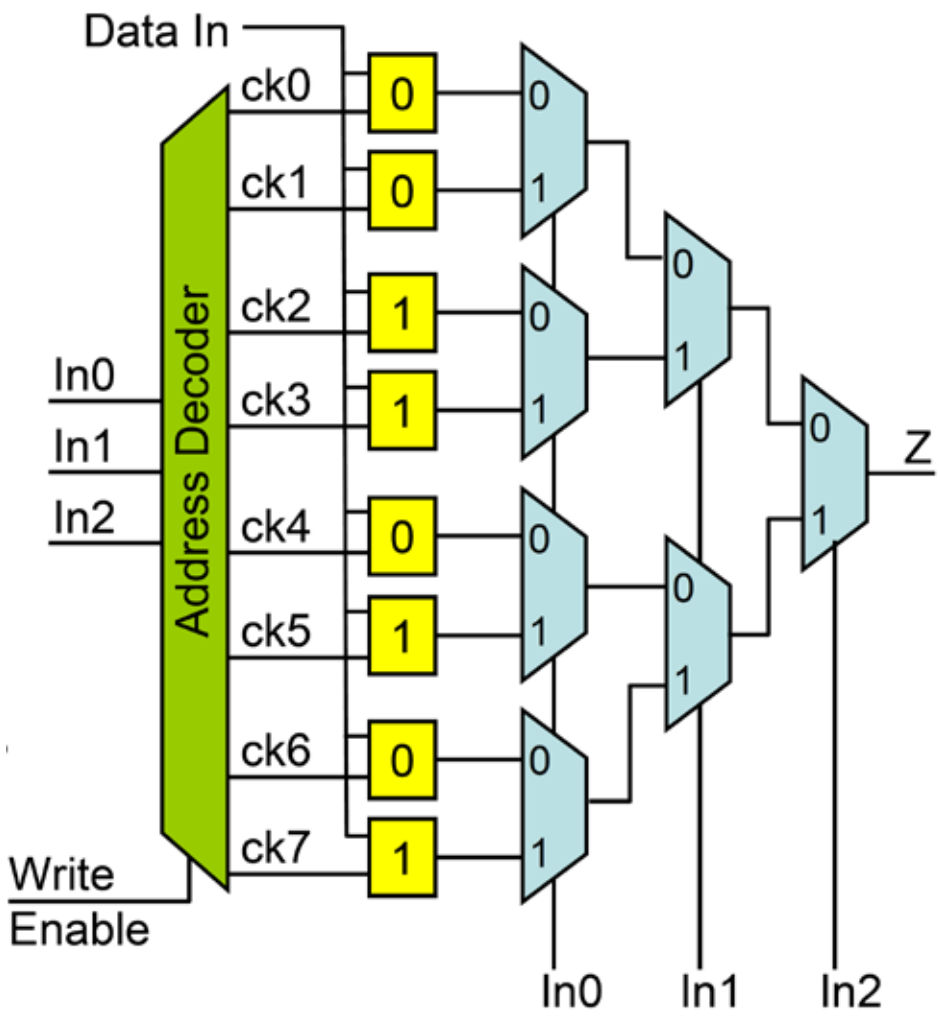


Truth table

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

SRAM Based Lookup Table

- Normal LUT mode performs read operations
- Address decoder with write enable generates clock signals to latches for write operations

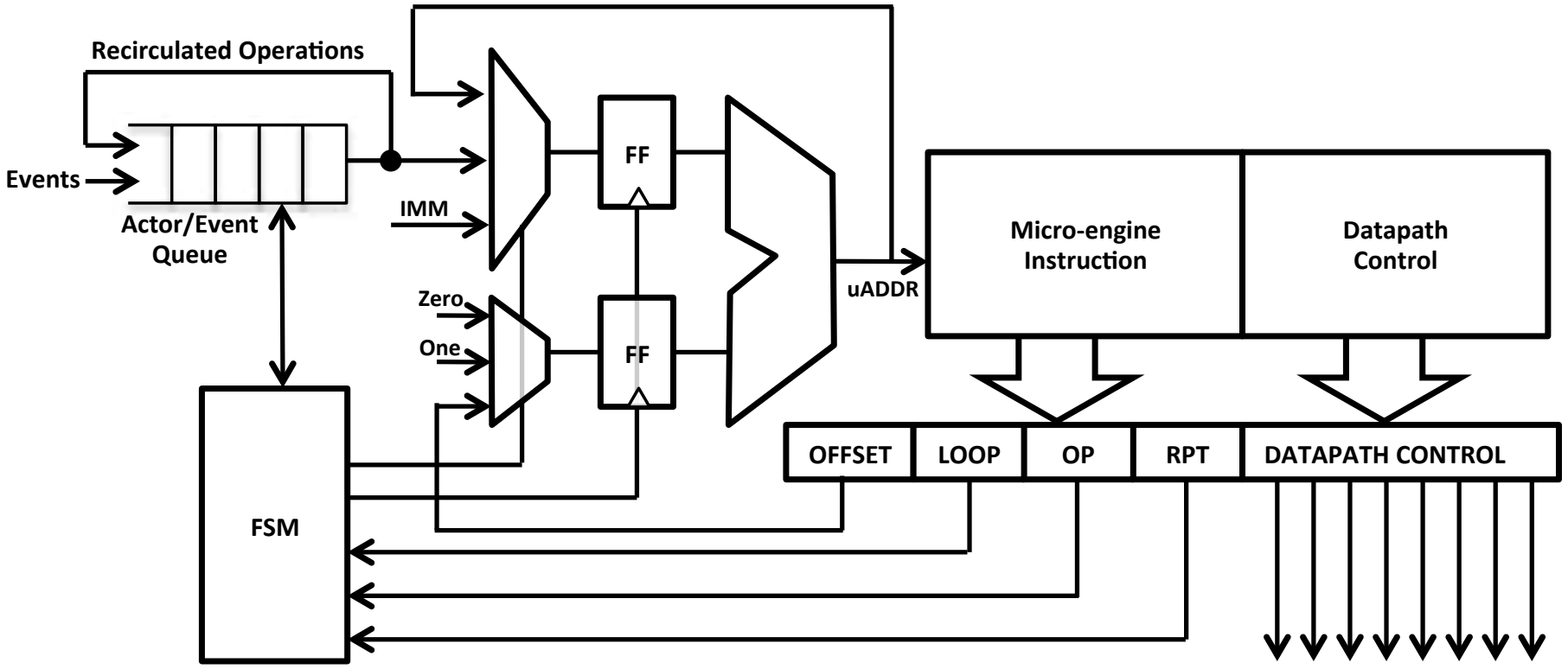


Agenda

- System overview
- Dataflow Processing Element (DPE)
  - Queues (FIFO), Stacks (LIFO), Queued-Stacks
  - Datapaths
  - Lookup Tables
  - **Microcode Control**
  - Finite State Machines (FSM)
- On-chip Networks
  - Serializers & Deserializers (SERDES)
  - Cross bar switches
  - Arbiters

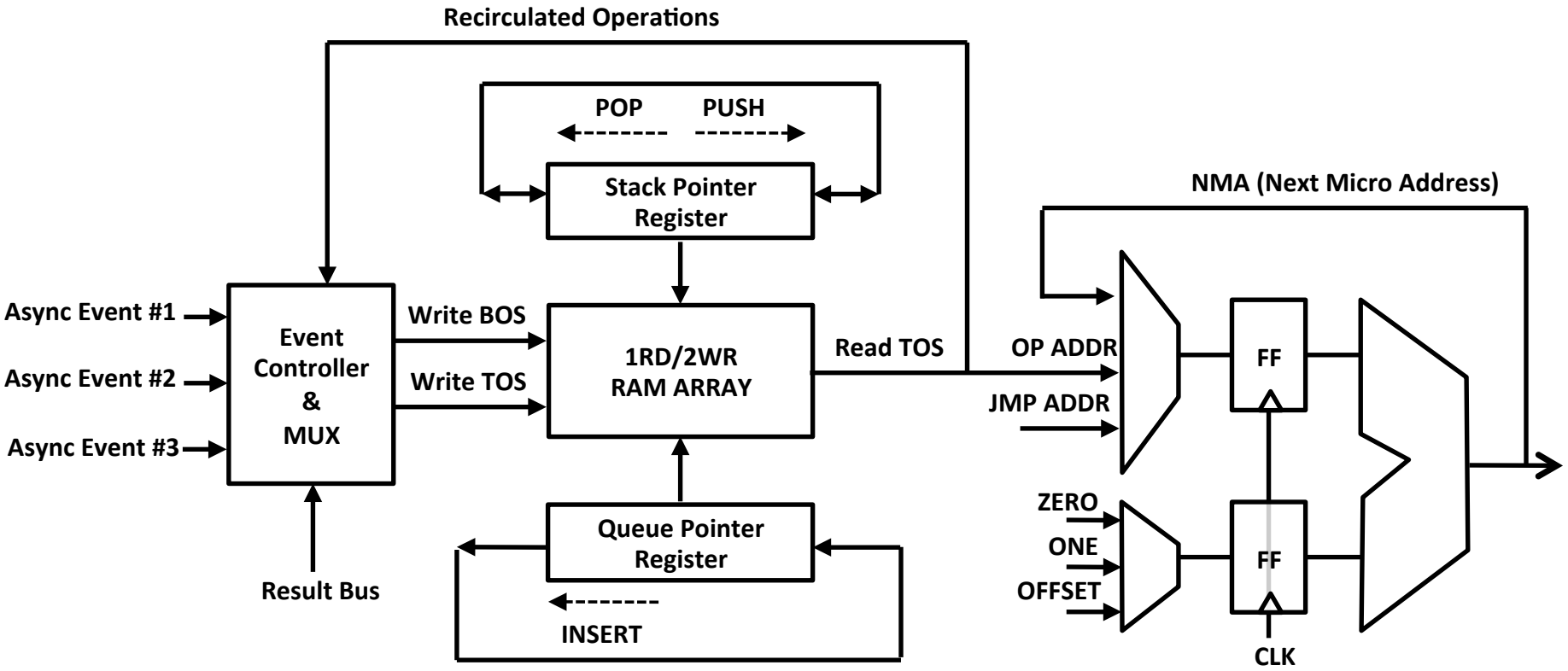
Microcode controller

- **Microcode word is 96 bits wide**
  - 32 bits is used for the micro-engine control
  - Remainder is used for datapath control
    - Combination of one-hot control bits and encoded bit fields



Next Micro Address Detail

- Sequences through a set of microcoded Actors to perform DPE functions.
- Asynchronous events inject special Actors into the sequence
  - Event Actors are squashed when completed



Microcode fields

Micro-engine control:

IMMED DATA <95:80>	BRANCH OFFSET <79:73>	LOOP COUNTER NUM <72:71>	REPEAT COUNT <70:68>	UCODE OP <67:64>

Datapath control:

SHFT MODE B <45>	SHFT MODE A <44>	SHFT B <43:39>	SHFT A <38:34>	SHFT TC B <33>	SHFT TC A <32>	MULT <31>	SAT <30>	DP EN <29>	ADD/SUB <28>

Queued-Stack control for next operand selection:

READ TOS QS2 <63>	READ TOS QS1 <62>	READ BOS QS2 <61>	READ BOS QS1 <60>	QS2 CTL <59:56>	QS1 CTL <55:52:>	B_BUS MUX <51:50>	A_BUS MUX <49:48>	CHAN MUX QS2 <47>	CHAN MUX QS1 <46>

Queued-Stack control for write-back operations:

QS2 BUS SEL <23:22>	QS1 BUS SEL <21:20>	WB MUX SEL <18:17>	RQS CTL <9:6>	RD RQS <4>	RQS BUS MUX <3>

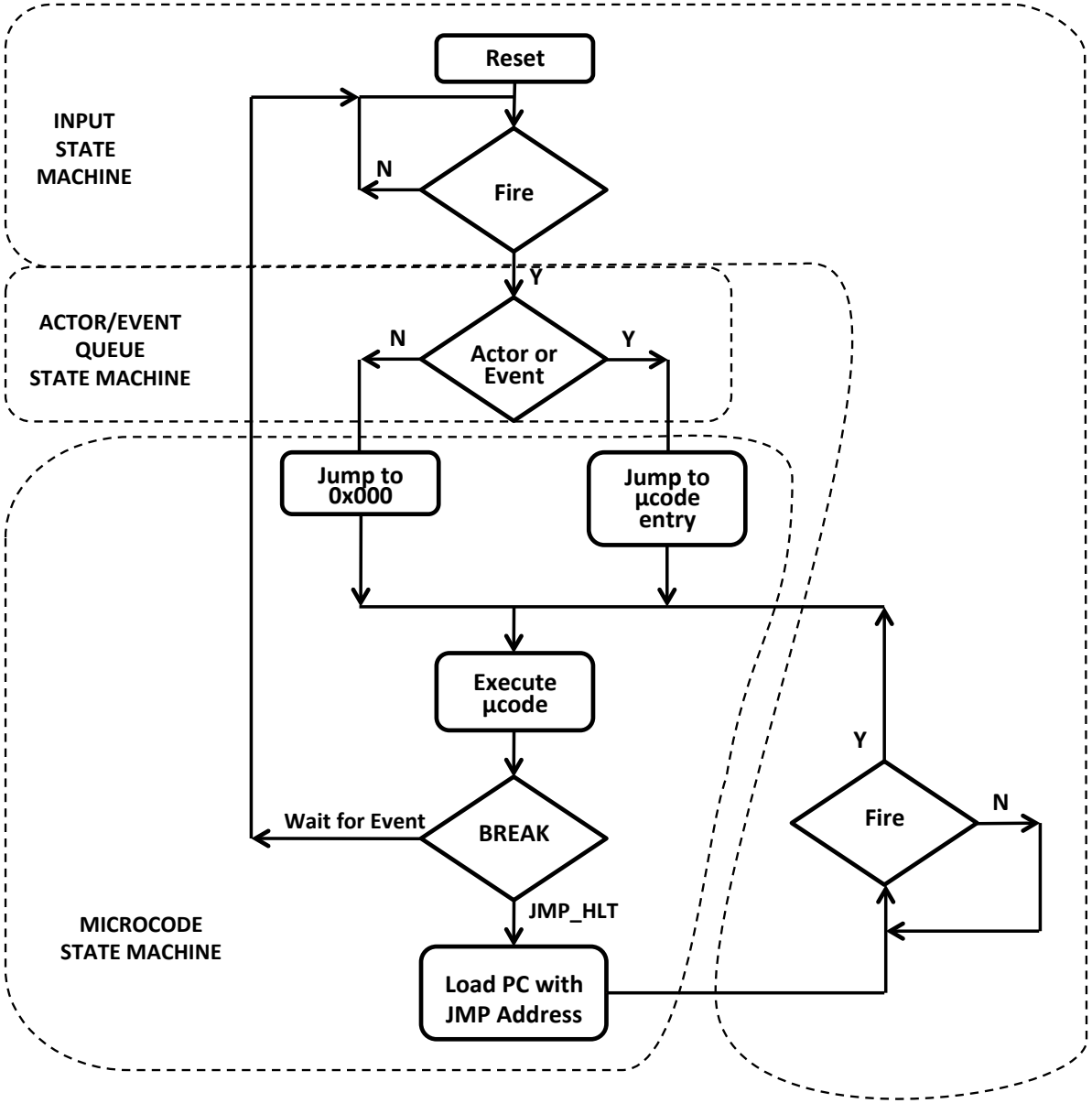
SFU & I/O control:

LOGIC OP <16:14>	SFU OP <13:11>	WR SFU LAT <10>	WR EQ LAT <5>	FIFO WE <2>	GPIO WE <0>

Agenda

- System overview
- Dataflow Processing Element (DPE)
  - Queues (FIFO), Stacks (LIFO), Queued-Stacks
  - Datapaths
  - Lookup Tables
  - Microcode Control
  - **Finite State Machines (FSM)**
- On-chip Networks
  - Serializers & Deserializers (SERDES)
  - Cross bar switches
  - Arbiters

Communicating FSM Control Flow



Master FSM Verilog Code

```
// ------------------------------------------------------------------------
//         MASTER STATE MACHINE
// ------------------------------------------------------------------------


always @(posedge clk or negedge reset_b) begin

            if (!reset_b)  begin            // In RESET
                state           <= 2'b0;    // Start in State 0
                sm_error        <= 1'b0;
                cpe_hold_out    <= 1'b1;    // Disable channel data to CPE
                chan_insert     <= 1'b0;
                fire_early      <= 1'b0;    // Negate signal
             end

        // ---------------------------------------------------------------
        // MAIN CONTROL LOOP
        // ---------------------------------------------------------------

            else if (reset_b) begin

                // -------------------------------------------------------------
                //    STATE 0
                // -------------------------------------------------------------

                if ((!cpe_ready_in) && (state_0)) begin
                    state           <=  2'b00;      // Stay in STATE 0
                    cpe_hold_out    <=  1'b0;       // Negate hold
                end

                else if ((cpe_ready_in) && (state_0)) begin
                    state           <=  2'b01;      // GOTO STATE 1
                end



                // -------------------------------------------------------------
                //    STATE 1
                // -------------------------------------------------------------

                else if (state_1) begin
                    state           <=  2'b10;   // GOTO  STATE 2
                    chan_insert     <=  1'b1;
                end

                // -------------------------------------------------------------
                //    STATE 2: insert channel data into FIFO
                // -------------------------------------------------------------


                else if ((!dwn_ctr_zd) && (state_2)) begin
                    state           <=2'b10;     // Stay in STATE 2
                end

                else if ((dwn_ctr_zd) && (state_2)) begin
                    state           <= 2'b11;    // GOTO  STATE 3
                    fire_early      <= 1'b1;    // Start ucode engine
                    cpe_hold_out    <= 1'b1;
                    chan_insert     <= 1'b0;
                end


                // -------------------------------------------------------------
                //    STATE 3: Wait for ucode engine to complete processing
                // -------------------------------------------------------------


                else if ((!ucode_idle) && (state_3)) begin
                    state           <=  2'b11;      // Stay in STATE 3
                    fire_early      <=  1'b0;       // Negate fire -
                                                    // ucode engine is running
                end

                else if ((ucode_idle) && (state_3)) begin
                    state           <=  2'b00;      // GOTO  STATE 0
                    cpe_hold_out    <=  1'b0;       // Negate hold
                    fire_early      <=  1'b0;
                end



        end         //  END of: else if (reset_b) begin

end     // END of:  always @(posedge clk or negedge reset_b) begin
```

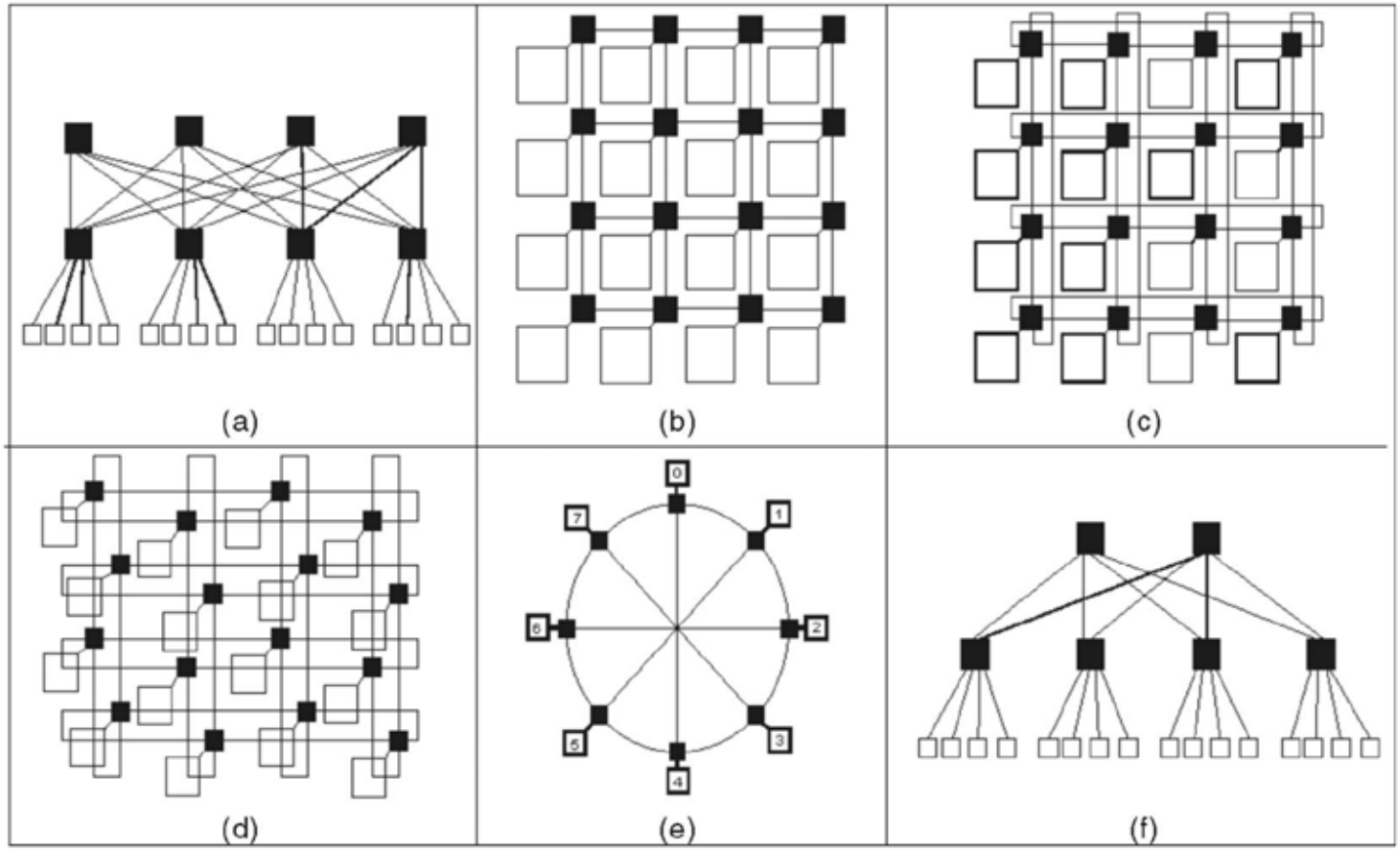
Agenda

- System overview
- Dataflow Processing Element (DPE)
  - Queues (FIFO), Stacks (LIFO), Queued-Stacks
  - Datapaths
  - Lookup Tables
  - Microcode Control
  - Finite State Machines (FSM)
- **On-chip Networks**
  - **Serializers & Deserializers (SERDES)**
  - **Cross bar switches**
  - **Arbiters**

NOC Topologies

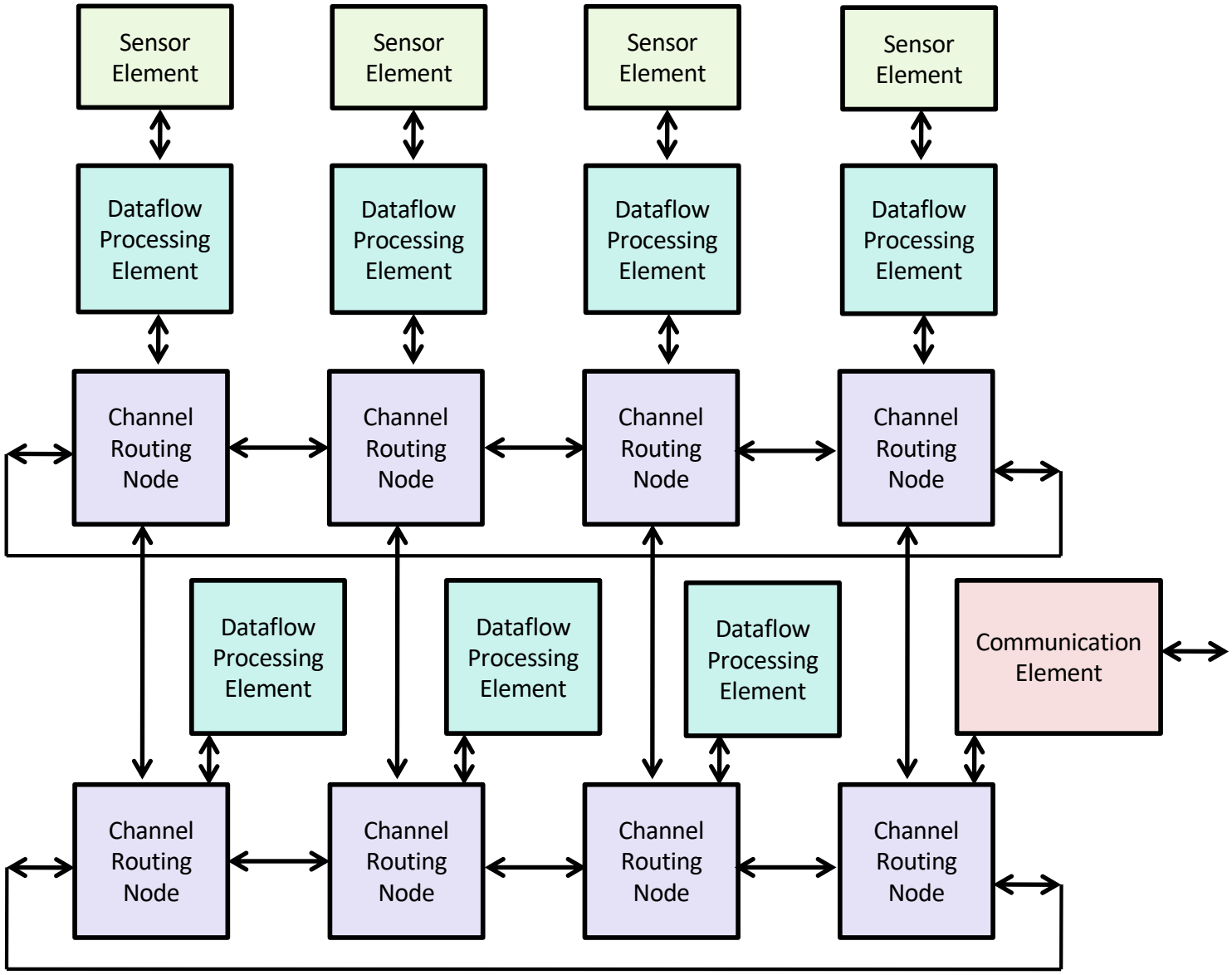
- **Heritage of networks with new constraints**
  - **Need to accommodate interconnects in a 2D layout**
  - **Cannot route long wires (clock frequency bound)**

a) SPIN,
b) CLICHE' & Mesh
c) Torus
d) Folded torus
e) Octagon
f) BFT

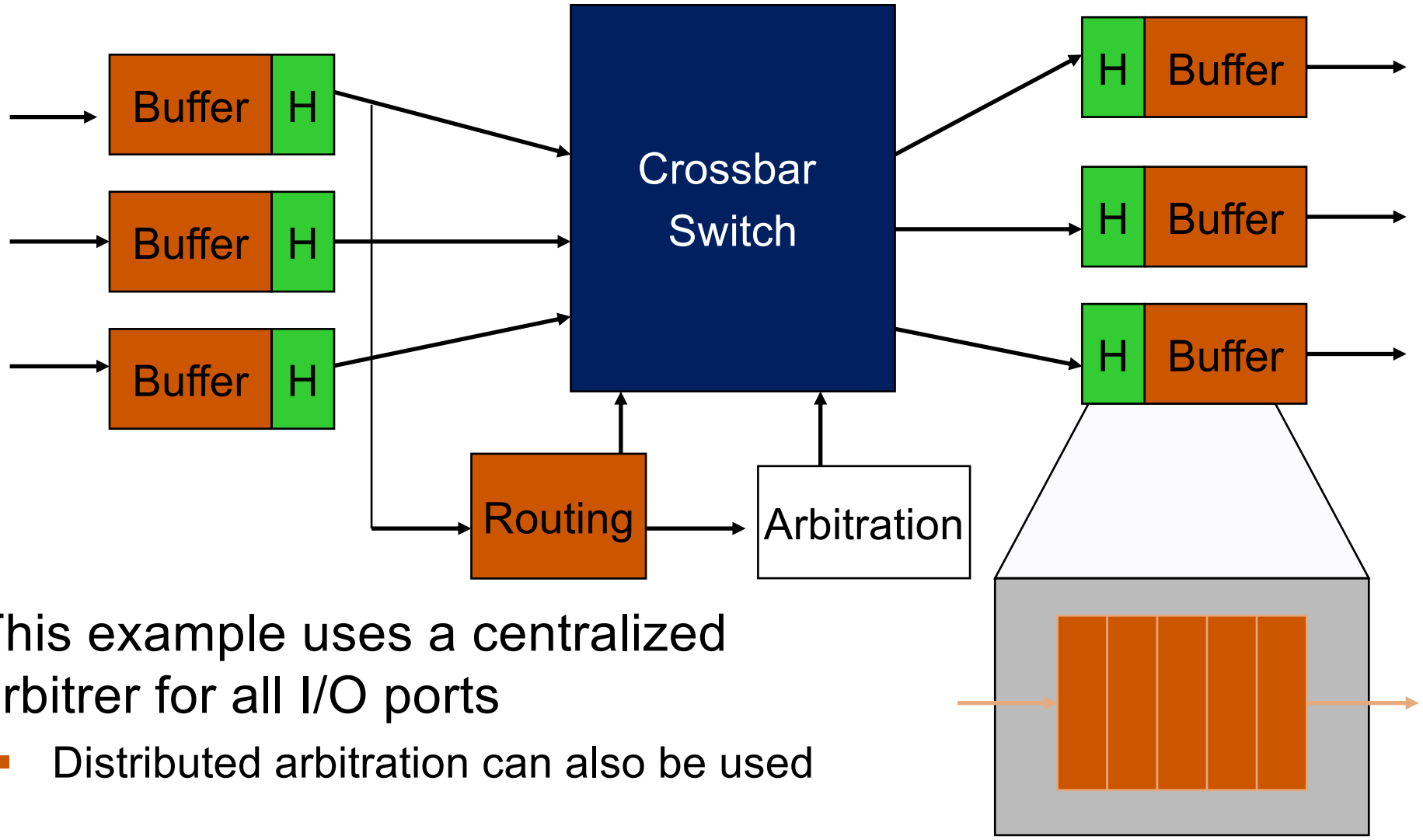


•Pande, et al 2005

Multiple DPEs connected via Channel Routers

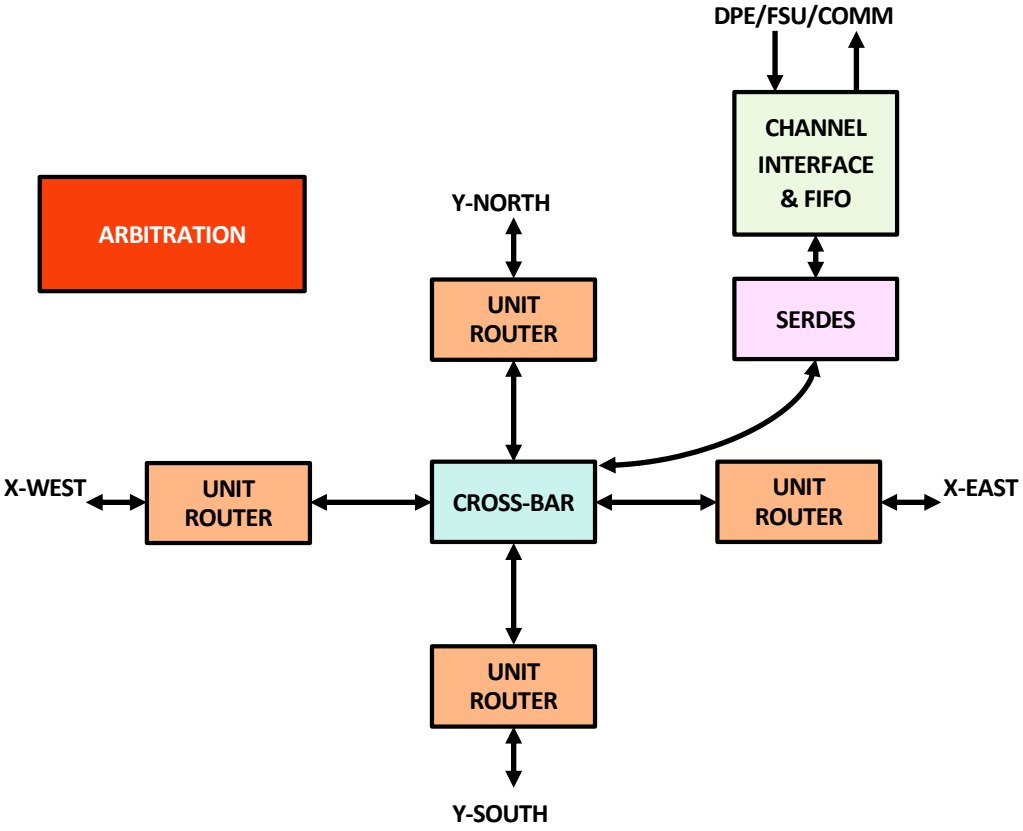


Typical NoC Router



- This example uses a centralized arbitrer for all I/O ports
  - Distributed arbitration can also be used

Channel Router Node

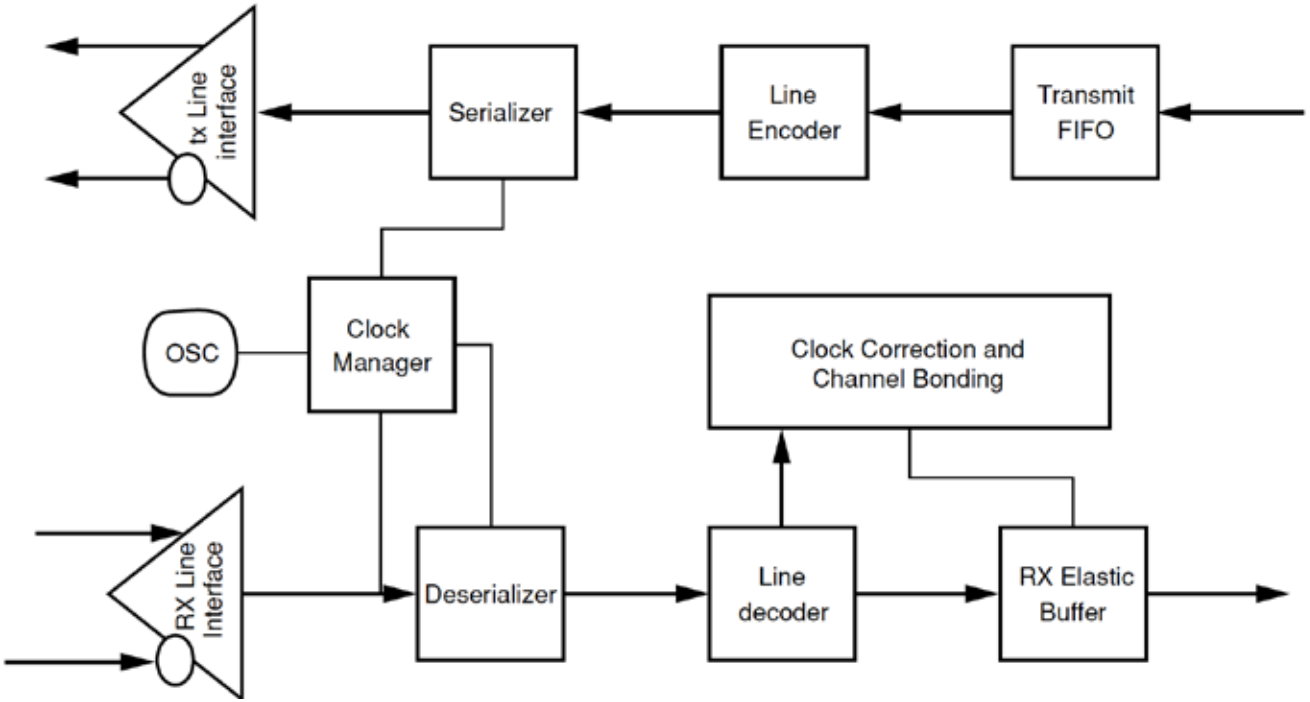


Agenda

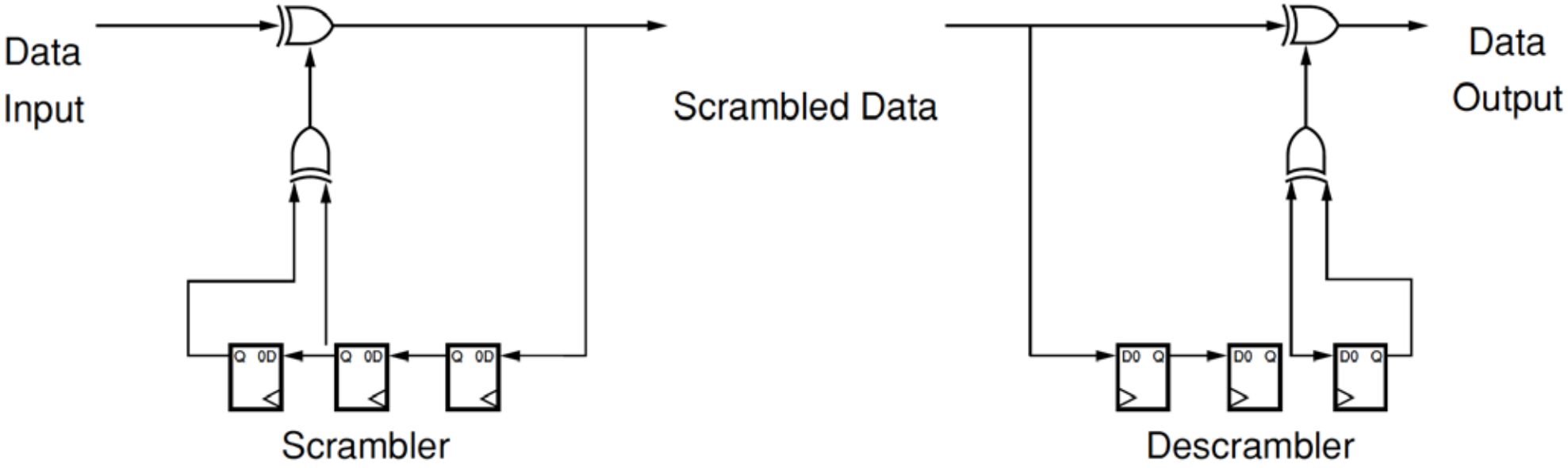
- System overview
- Dataflow Processing Element (DPE)
  - Queues (FIFO), Stacks (LIFO), Queued-Stacks
  - Datapaths
  - Lookup Tables
  - Microcode Control
  - Finite State Machines (FSM)
- On-chip Networks
  - **Serializers & Deserializers (SERDES)**
  - Cross bar switches
  - Arbiters

Serializers & Deserializers (SERDES)

- **SERializer**
  - Takes n bits of parallel data changing at rate y and transforms them into a serial stream at a rate of n times y.
- **DESerializer:**
  - Takes serial stream at a rate of n times y and changes it into parallel data of width n changing at rate y.



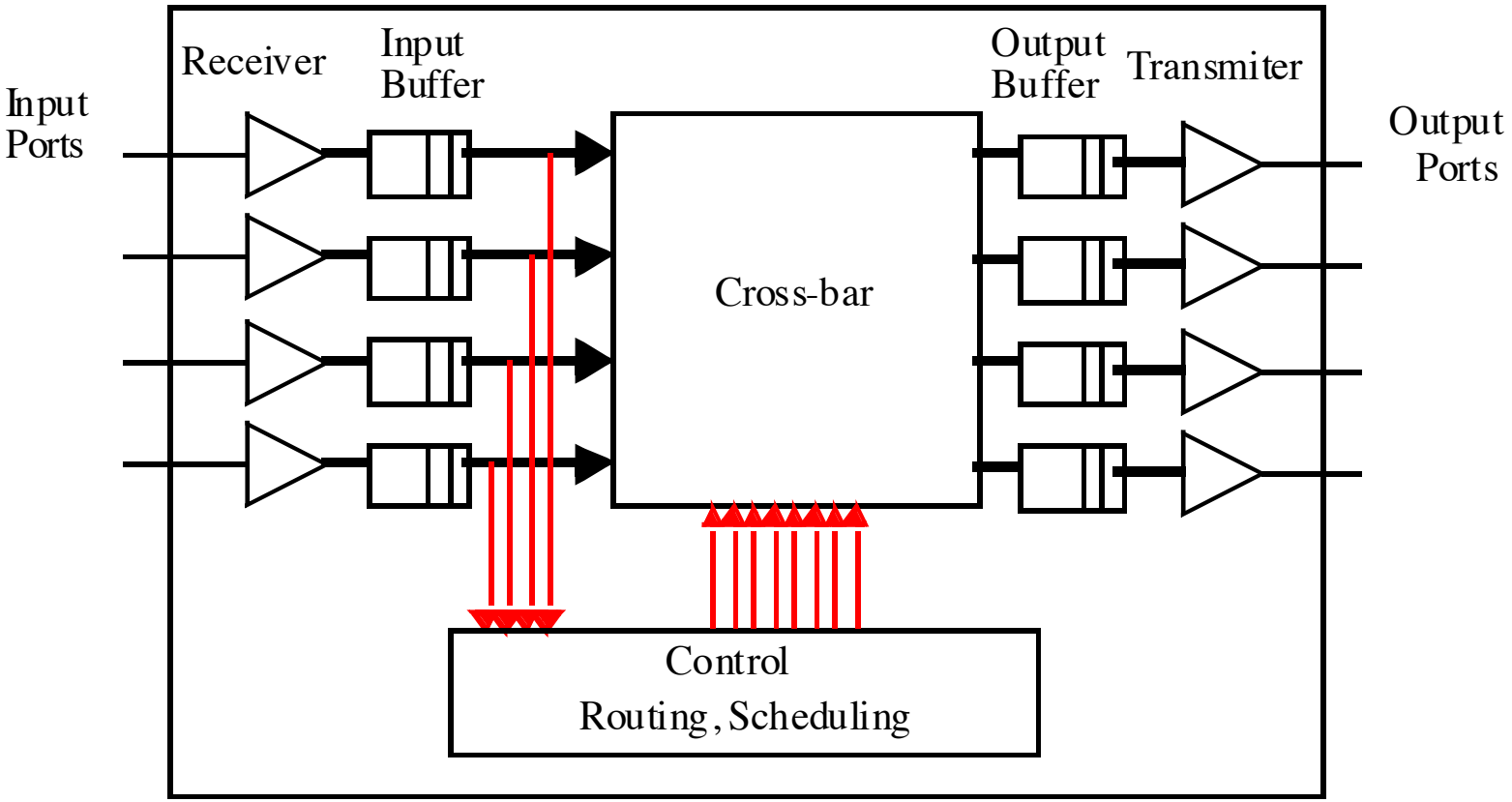
LFSR Based Scrambler/Descrambler



Agenda

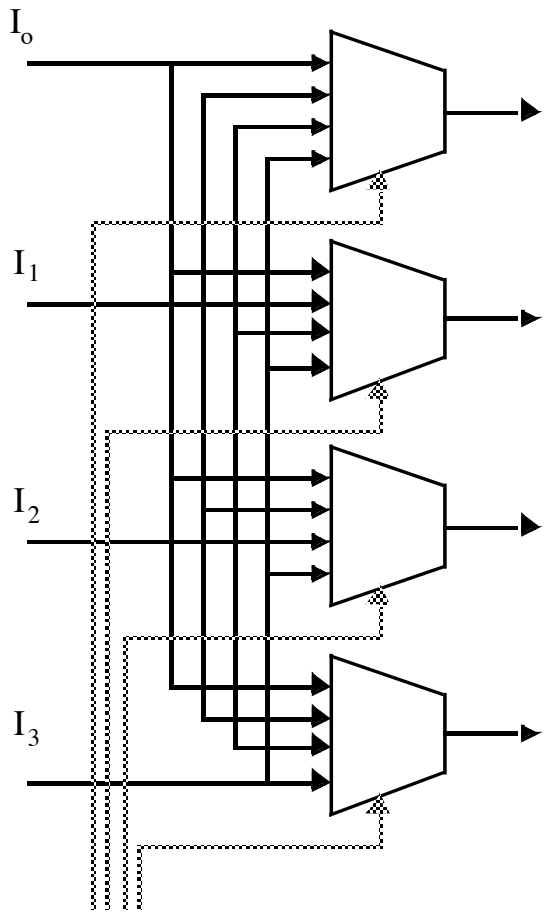
- System overview
- Dataflow Processing Element (DPE)
  - Queues (FIFO), Stacks (LIFO), Queued-Stacks
  - Datapaths
  - Lookup Tables
  - Microcode Control
  - Finite State Machines (FSM)
- On-chip Networks
  - Serializers & Deserializers (SERDES)
  - **Cross bar switches**
  - Arbiters

Crossbar Switching System

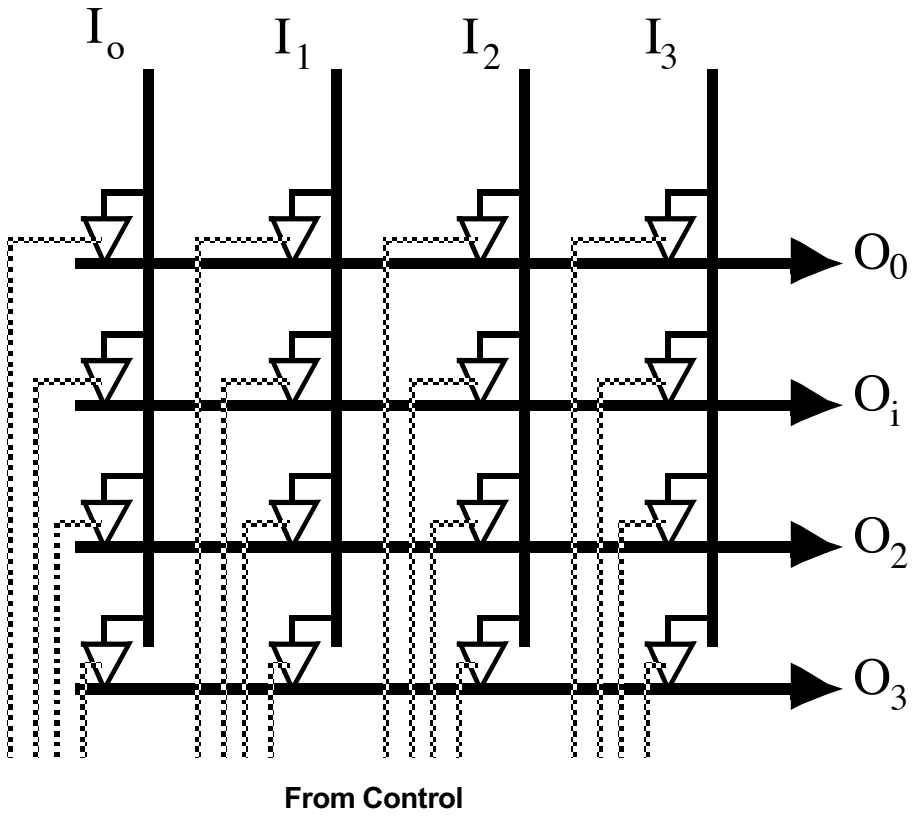


Complexity O(N2) for an NXN Crossbar**

Crossbar Switch Design



Multiplexors are controlled from the arbiter/controller/scheduler



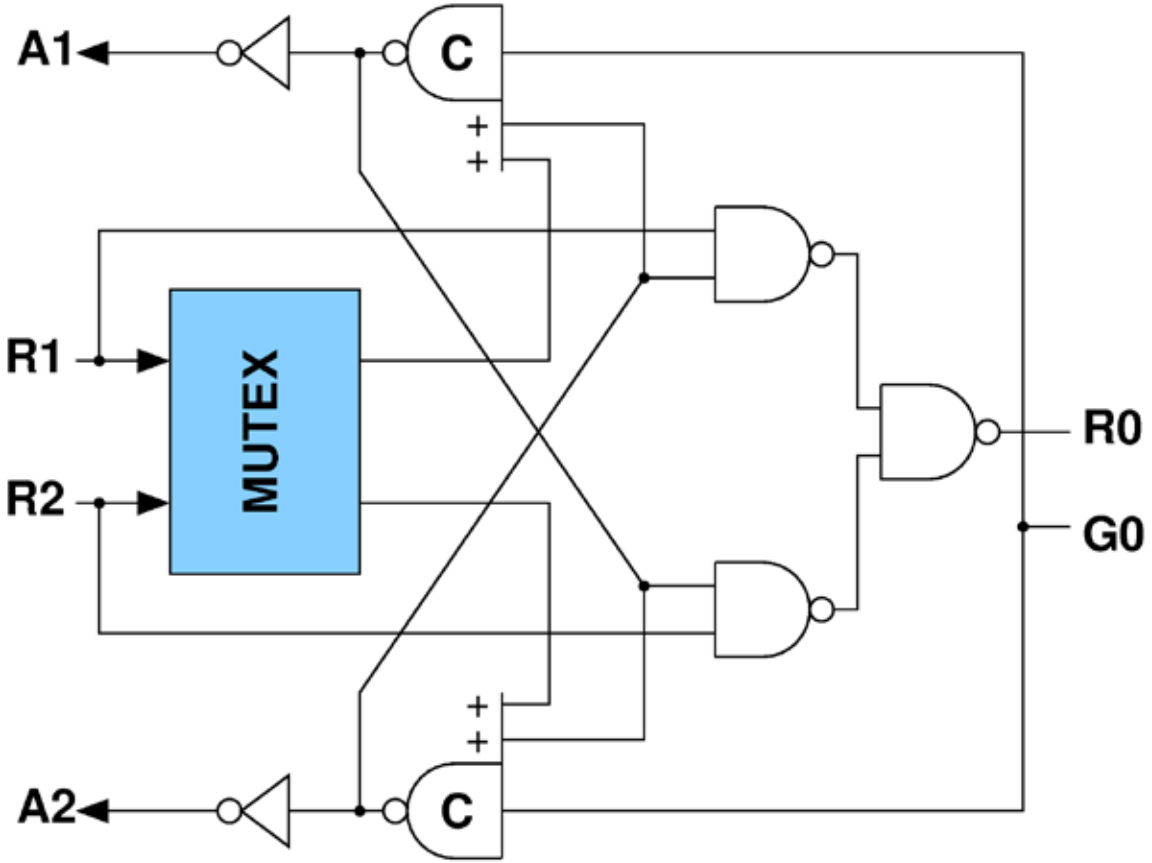
N**2 switches => Cost O(N**2)

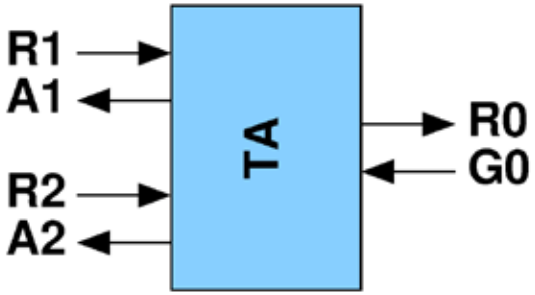
Time taken by the arbiter = O(N**2)

Agenda

- System overview
- Dataflow Processing Element (DPE)
  - Queues (FIFO), Stacks (LIFO), Queued-Stacks
  - Datapaths
  - Lookup Tables
  - Microcode Control
  - Finite State Machines (FSM)
- On-chip Networks
  - Serializers & Deserializers (SERDES)
  - Cross bar switches
  - **Arbiters**

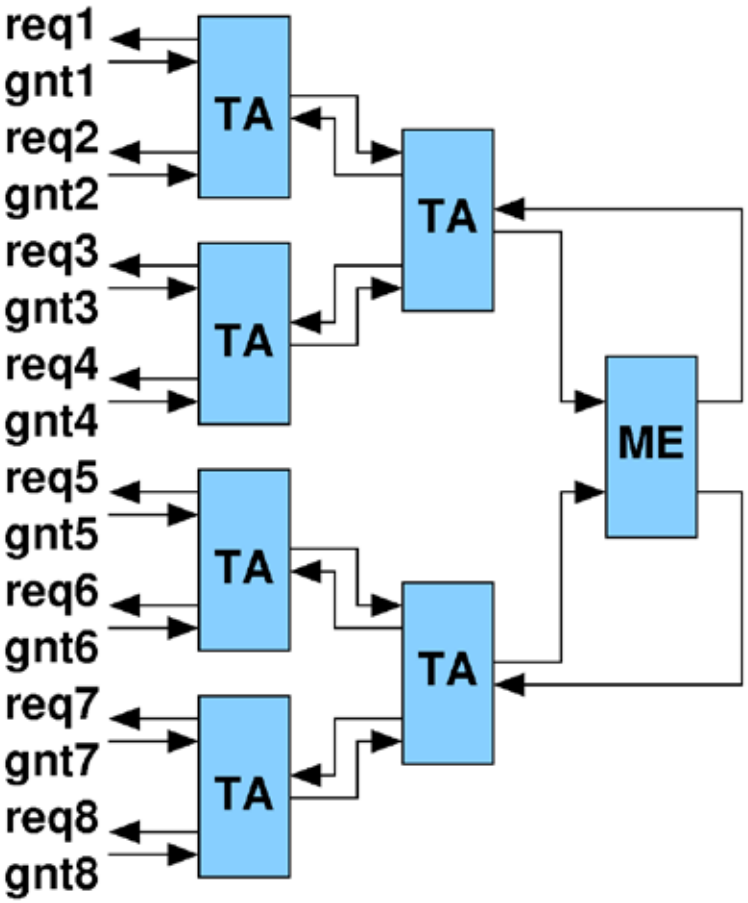
Tree Arbiter Element



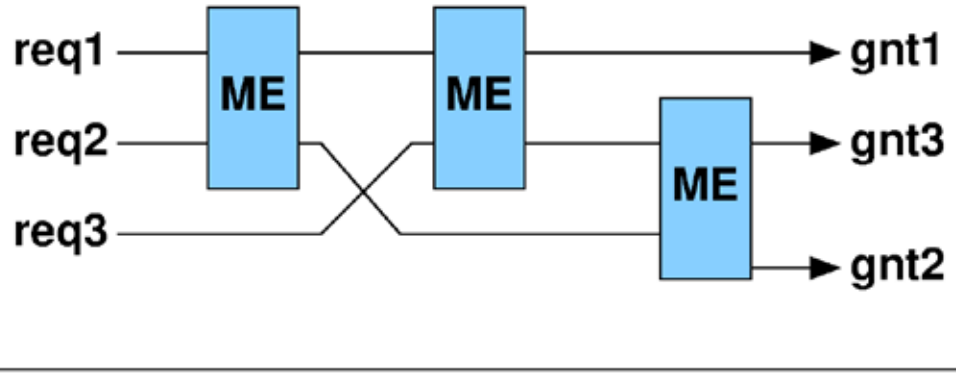


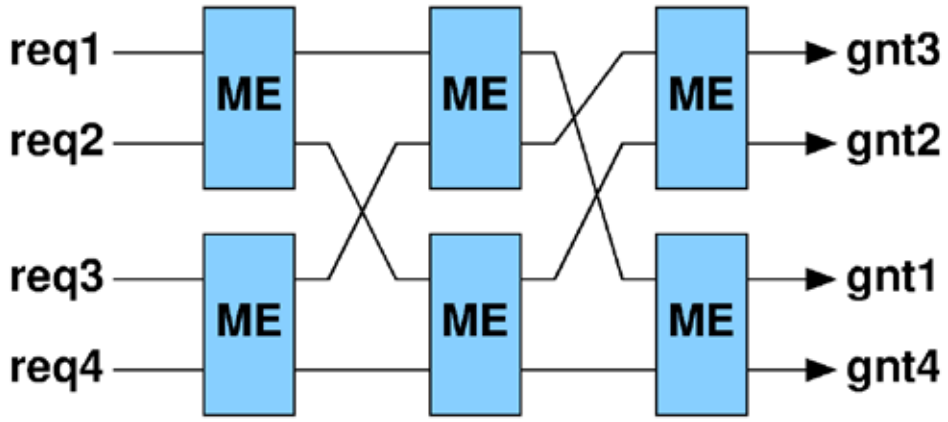
Tree–Arbiter Element

Multiway Arbiters



Tree Arbiter



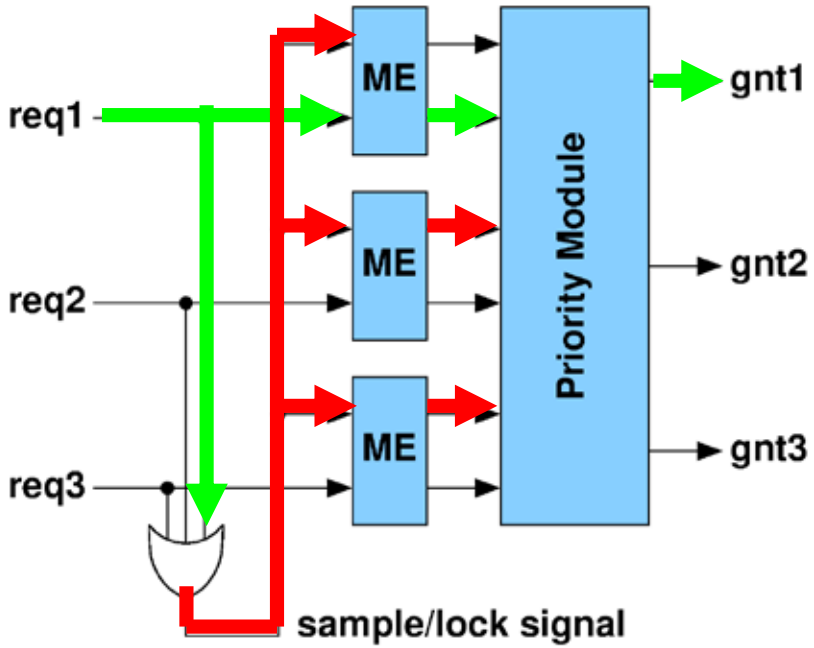


Cascaded MUTEXes

•Mullins, Cambridge

Static Priority Arbiters

- **"Priority Arbiters"**
  Bystrov/Kinniment/Yakovlev (ASYNC'00)
- **First stage samples/locks current request vector**
- **Static or dynamic priority**
- **Original design updated to tackle performance and QoS issues**
  Felicijan/Bainbridge/Furber (ICM'03)



Basis for sample/lock and prioritise arbiter

Mullins, Cambridge