Memristive Memory Processing Unit (mMPU)

Real Processing in Memory using Memristors

Nishil Talati, Rotem Ben Hur, Nimrod Wald, Ameer Haj Ali,
Ben Perach, Natan Peled, **Ronny Ronen** and Shahar Kvatinsky

Technion – Israel Institute of Technology

Yale80 in 2019, July 2, 2019

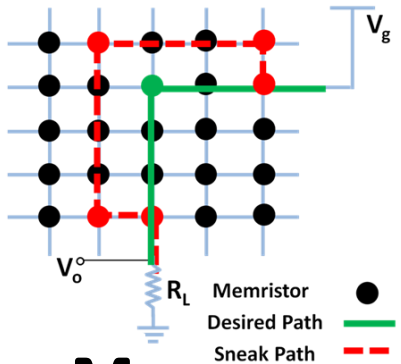
ASIC^2
ARCHITECTURES
SYSTEMS
INTELLIGENT COMPUTING
INTEGRATED CIRCUITS

erc European Research Council

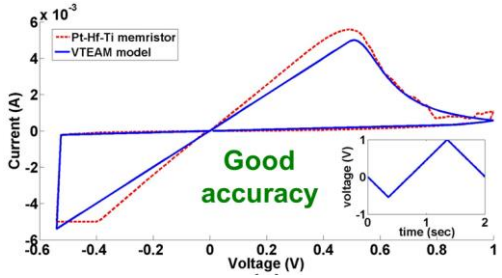
TECHNION Israel Institute of Technology

The ASIC² Group

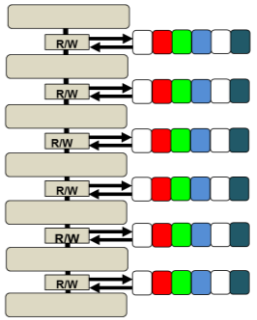
- Emerging technologies: Design, Simulation, Modeling, Applications
- Explore computation beyond von Neumann architectures



Memory design



Simulation tools



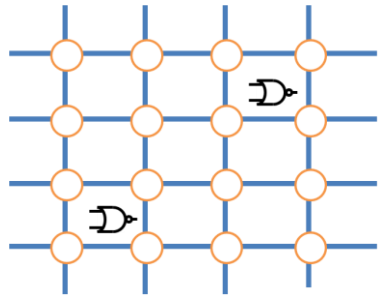
Efficient processors

Neuromorphic computing

Hardware security

Mixed signal & RF circuits

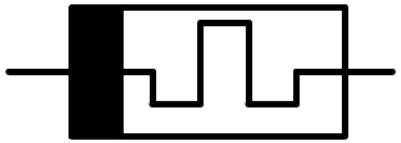
Cytomorphic electronics



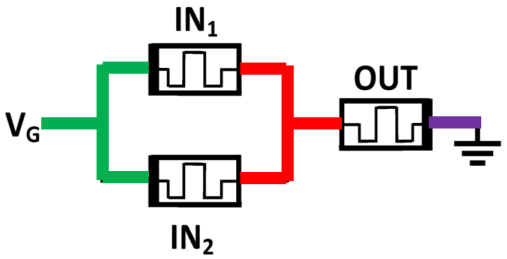
Memristive Memory Processing Unit (mMPU)

ASIC²

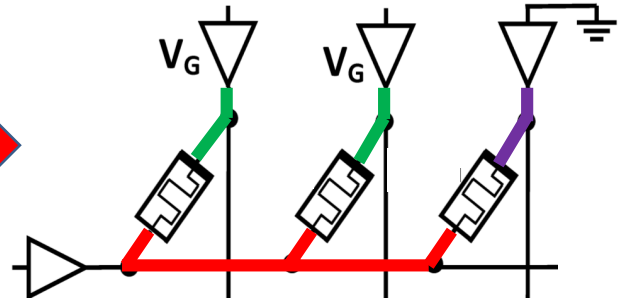
Talk Summary in a Single Slide



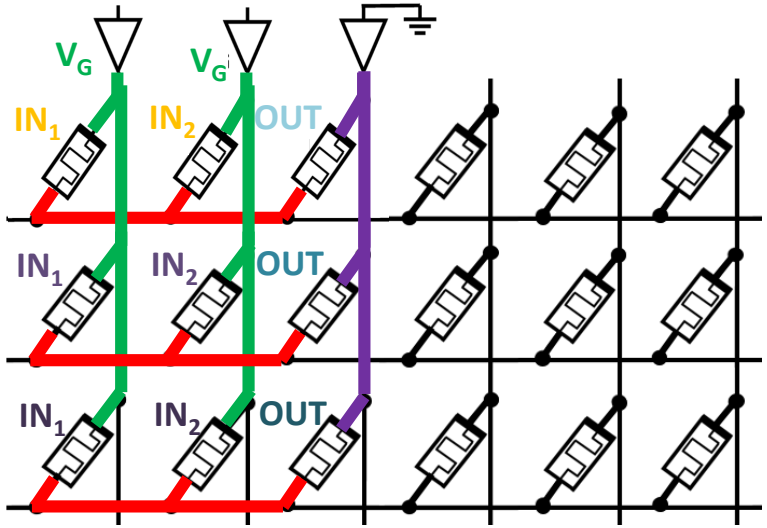
A memristor memory cell



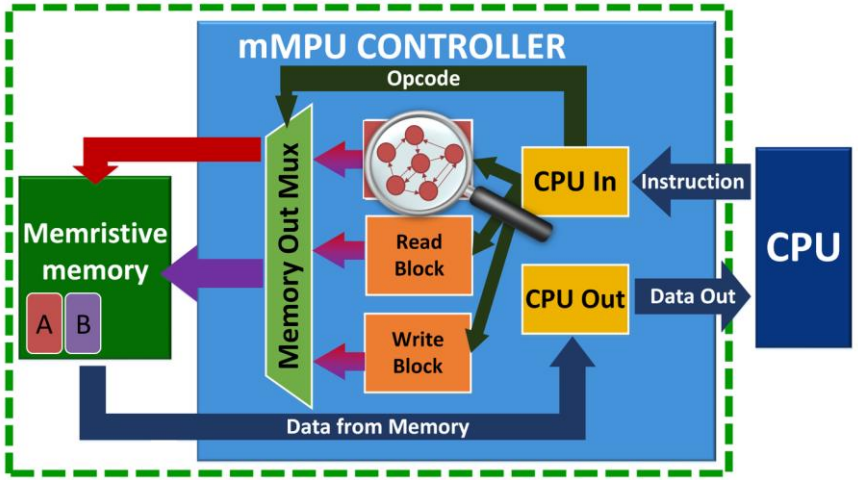
NOR logic Gate (MAGIC)



Crossbar Compatible



SIMD computing in memory

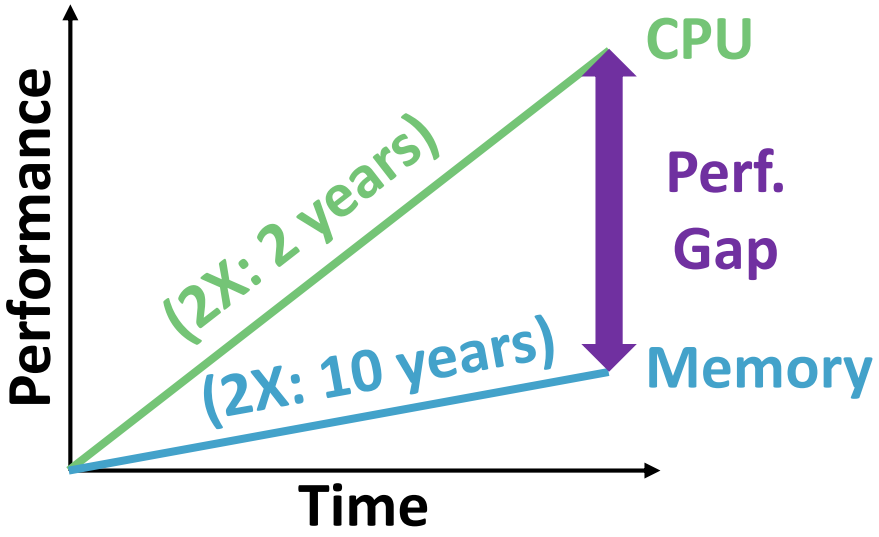


Control & Data

True Processing in Memory

The von Neumann Bottleneck

Latency

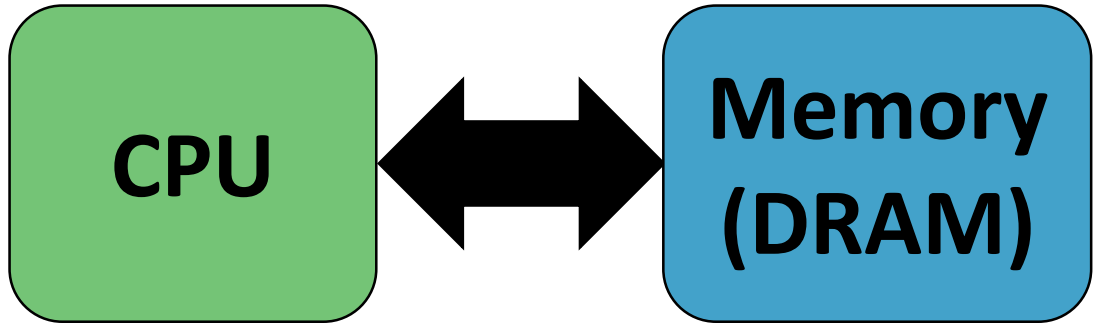


Energy

Operation (16-bit operand)	Energy/Op (45nm)	Cost (vs. Add)
Add operation	0.18 pJ	1X
Load from on-chip SRAM	11 pJ	~60X
Send to off-chip DRAM	**640 pJ**	**~3600X**

Pedram *et al.*, IDT, 2017

The von Neumann Machine



Processing In Memory (PIM) – Higher Performance, Lower Energy!

The Problem – and the Solution Strategy
Onur Mutlu – July 1, 2019

The Problem

Processing of data
is performed
far away from the data

62.7% of the total system energy is spent on data movement

We Need A Paradigm Shift To …

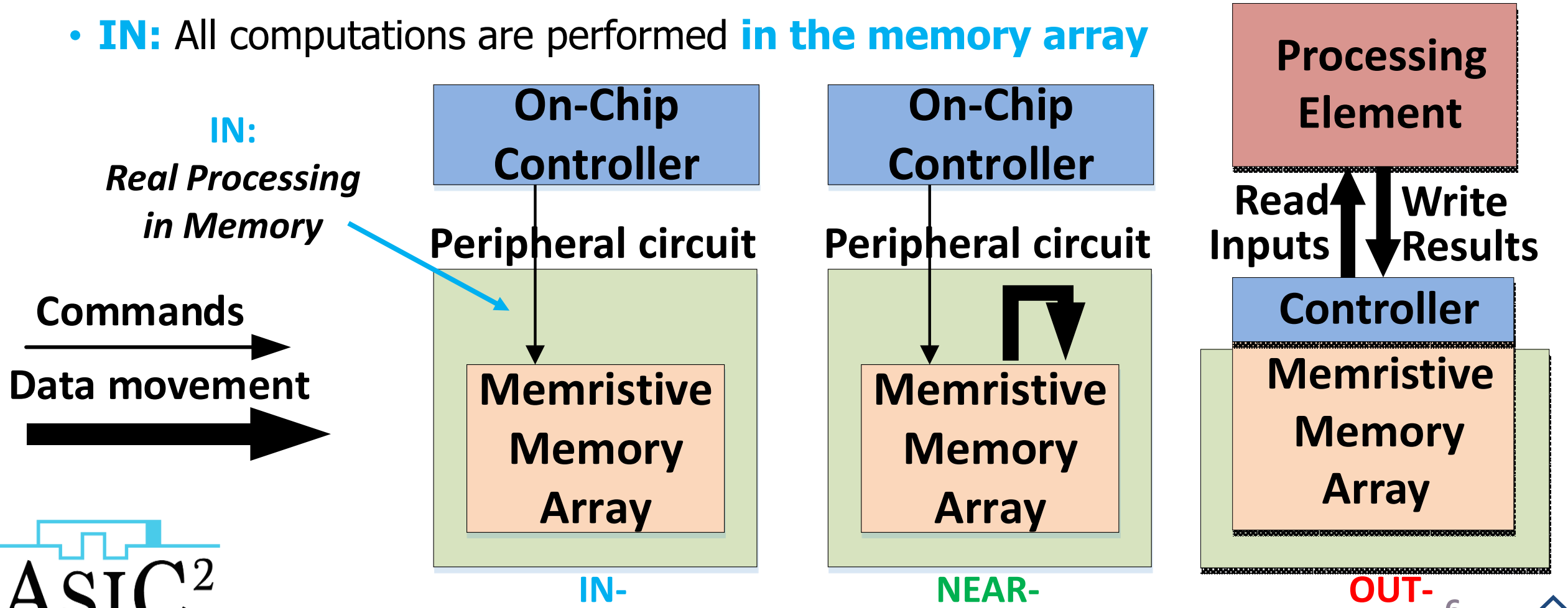
- Enable computation with minimal data movement
- Compute where it makes sense (where data resides)
- Make computing architectures more data-centric

ASIC²

Processing In Memory (PIM) – Higher Performance, Lower Energy!

In-/Near-/Out- Memory Computing

- **OUT:** All computations are done out of the memory array → **data movement, dedicated processing units**
- **NEAR:** Some computations done out of the memory array → **data movement**
- **IN:** All computations are performed **in the memory array**

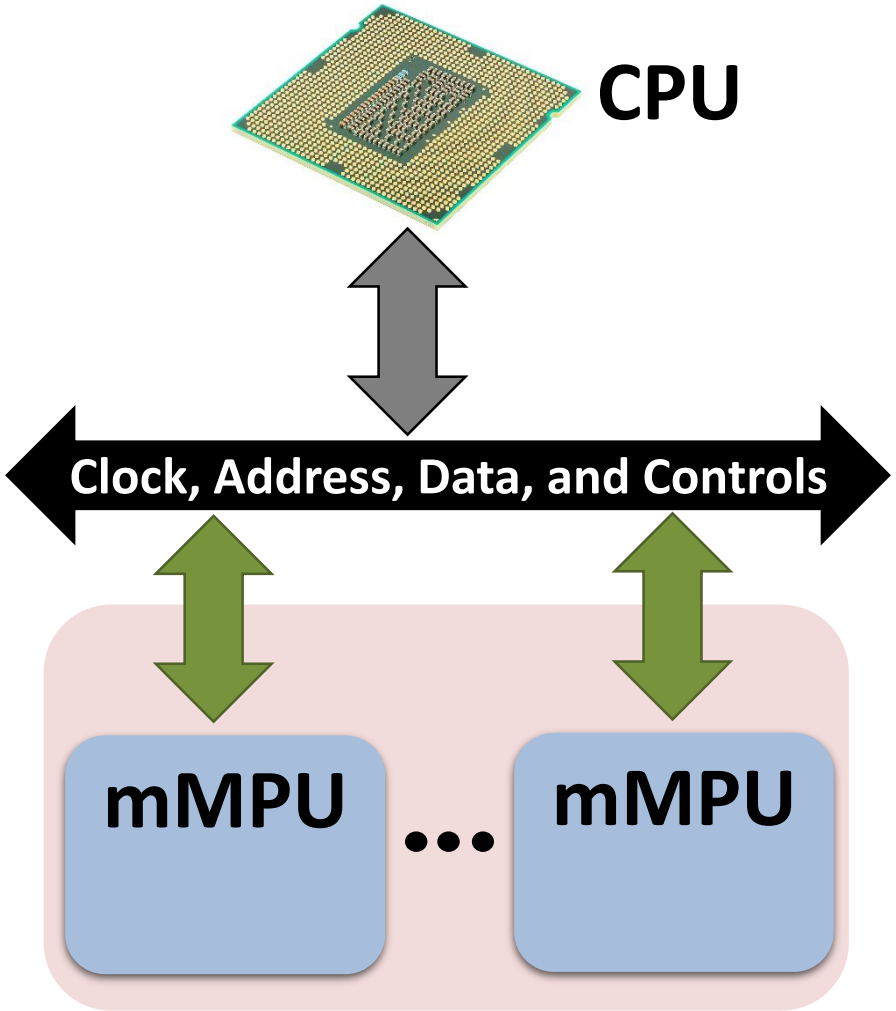


J. Reuben et al., "Memristive Logic: A Framework for Evaluation and Comparison", PATMOS 2017

mMPU: Potential Solution to the von Neumann Bottleneck

Moving from conventional DRAM to memristive memory

mMPU: performing computation *USING* the memristive memory cells

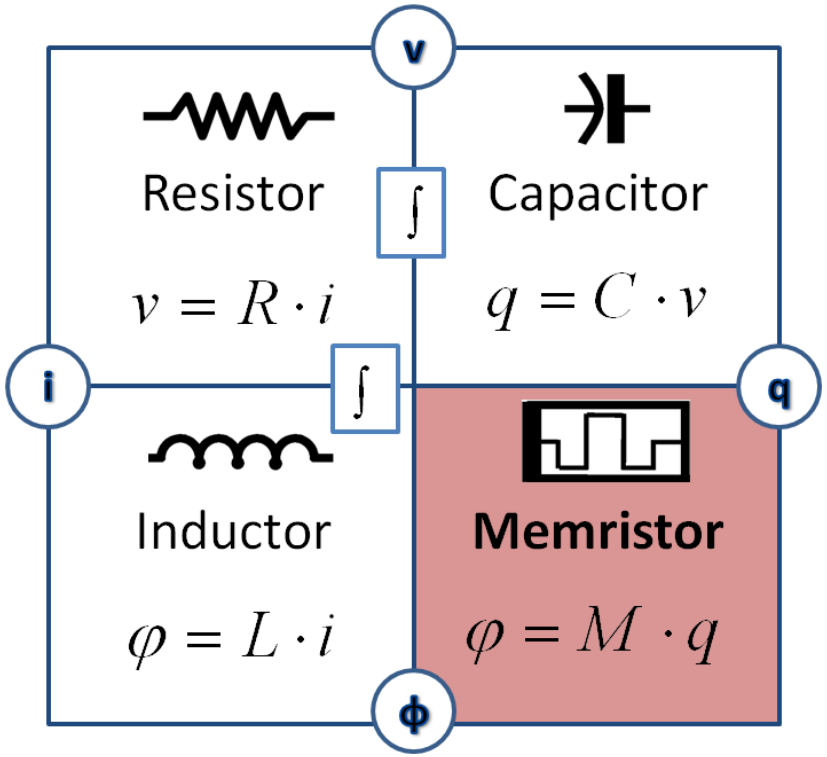


ASIC²

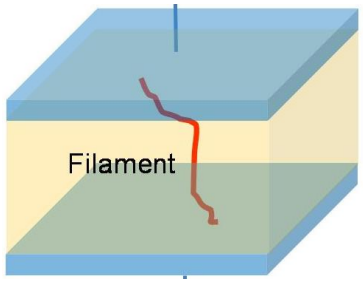
Outline

- Background
- **Memristor basics**
- Logic using memory cells
- Processing in Memory - the mMPU
- System design using the mMPU
- Conclusions

Memristor: The Fourth Basic Element

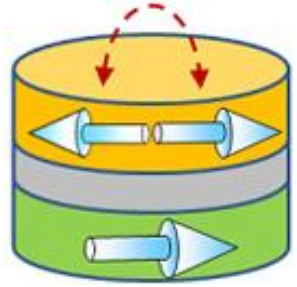


Resistive RAM (RRAM)



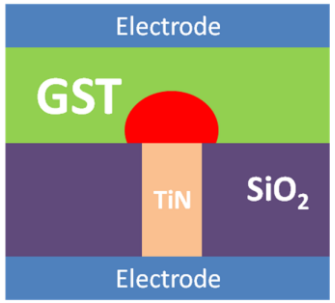
WD SONY HP SAMSUNG Panasonic tsmc SK hynix TOSHIBA Crossbar

STT MRAM



EVERSPIN TECHNOLOGIES HITACHI CROCUS Technology Blossoming future TOSHIBA tsmc SAMSUNG QUALCOMM

Phase Change Memory (PCM)

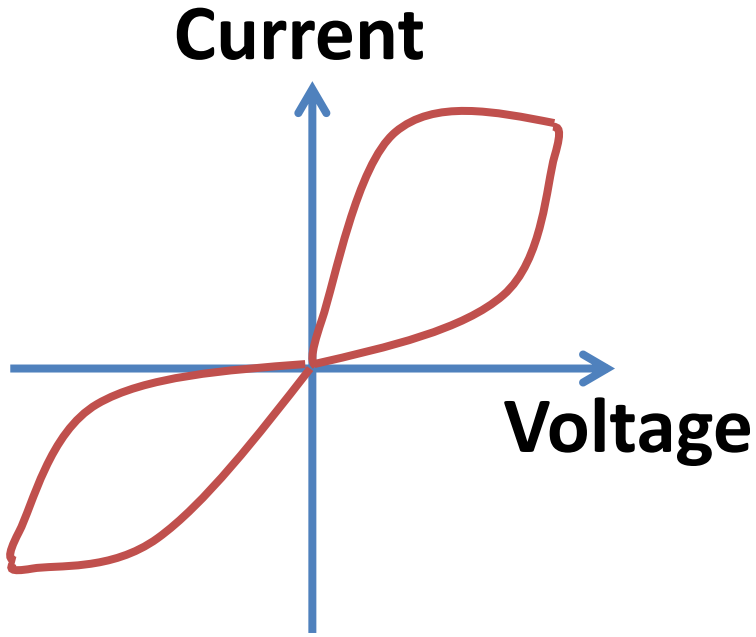


SAMSUNG intel IBM Micron SK hynix

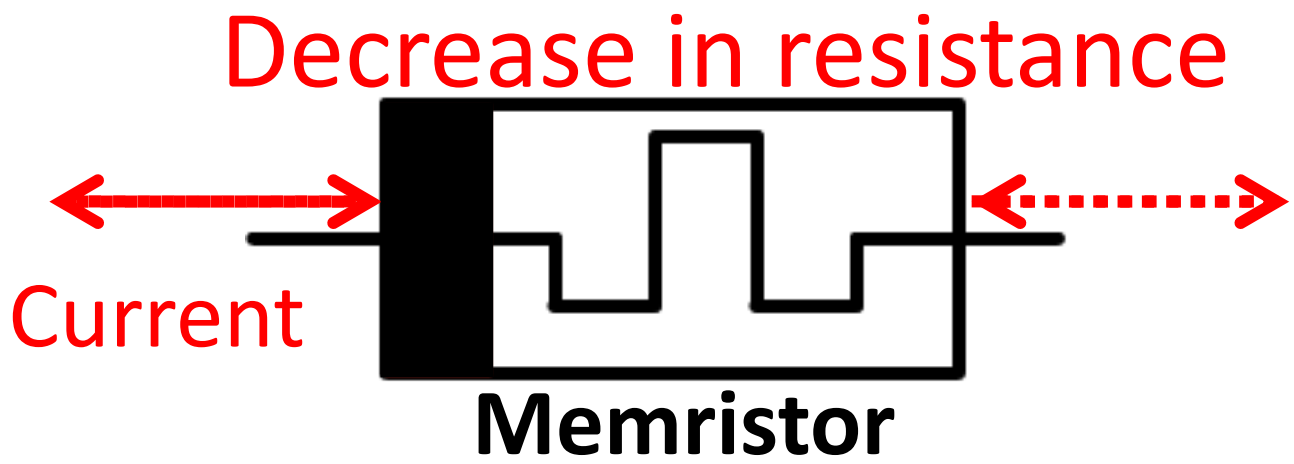
ASIC²

L.O. Chua, "Memristor – The Missing Circuit Element," IEEE Trans. on Circuit Theory, 1971

Memristor – Memory Resistor
Resistor with Varying Resistance

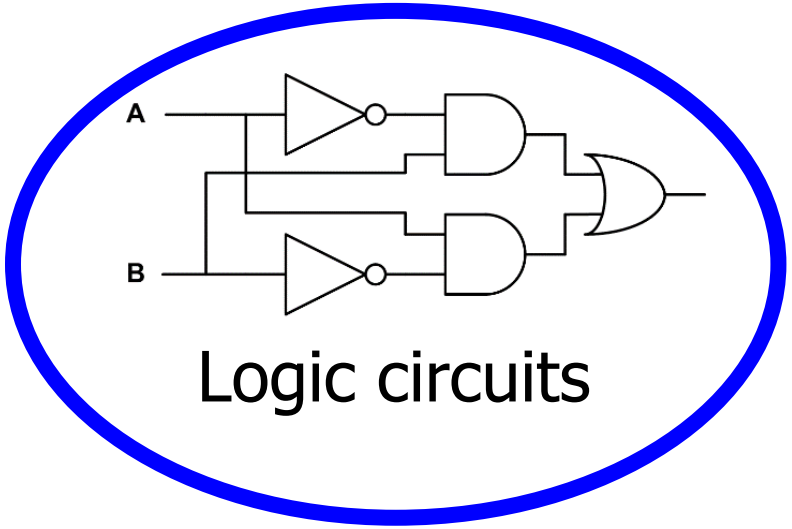


Low resistive state (R_{ON}, LRS)
High resistive state (R_{OFF}, HRS)

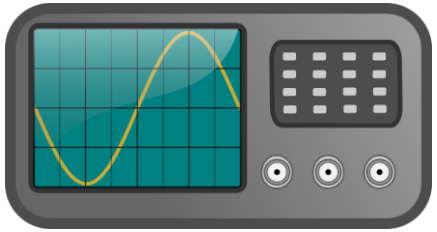


Applications for Memristors

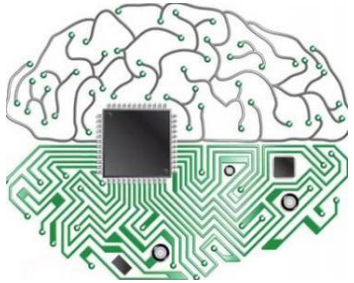
Memory



Logic circuits



Analog circuits

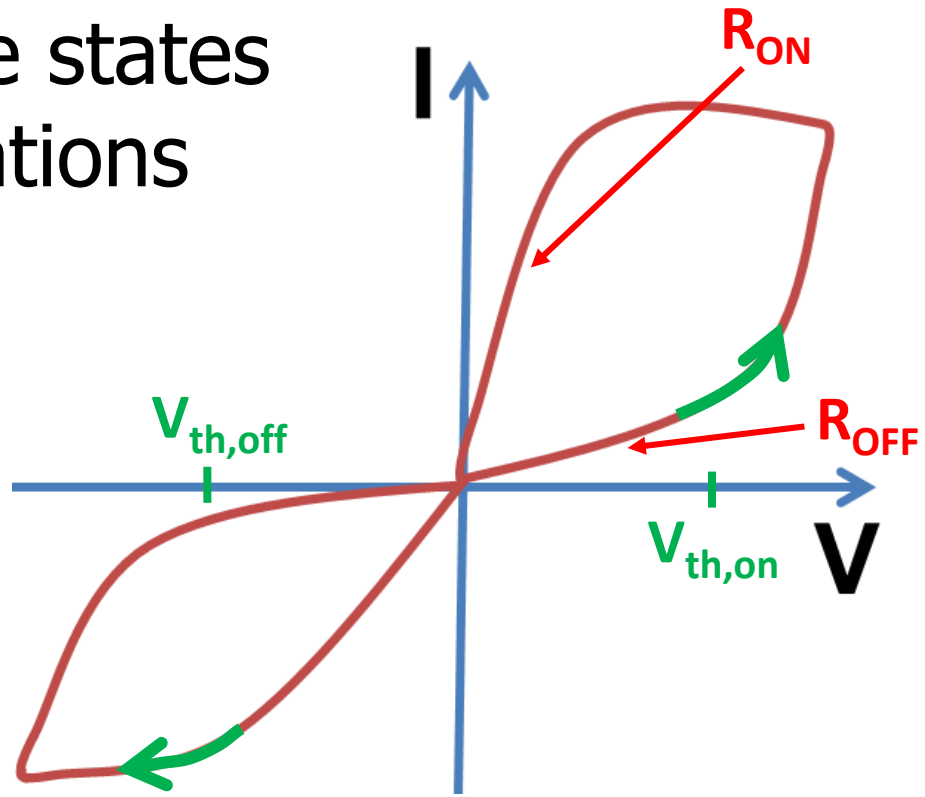


Neuromorphic computing

Security

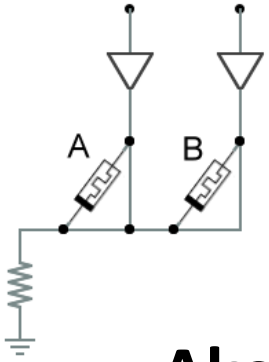
Important Memristors Attributes for Logic

- Hysteresis – state is preserved
- Distinct high/low resistance states (HRS/LRS) – binary applications
- Threshold current/voltage for switching

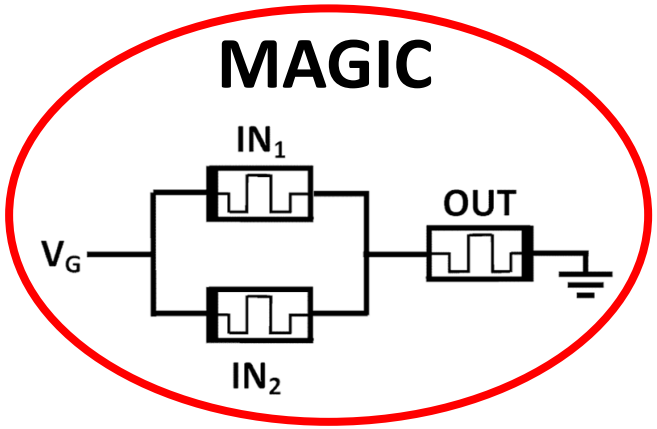


Logic Families Using Memristors

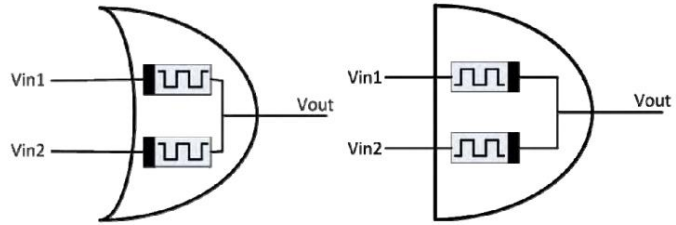
IMPLY



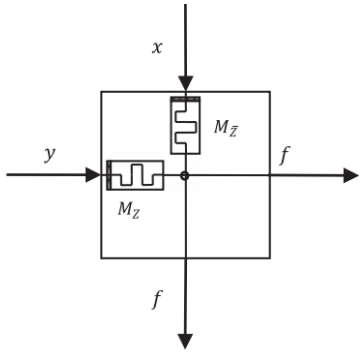
MAGIC



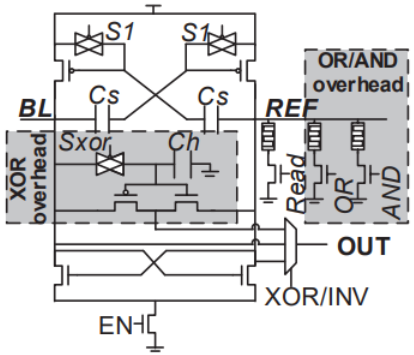
MRL



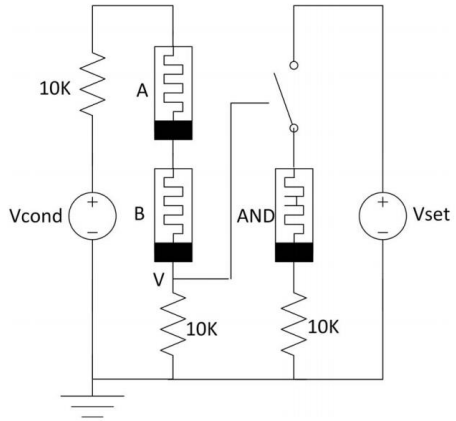
Akers



PINATUBO



MAD



J. Reuben et al., "Memristive Logic: A Framework for Evaluation and Comparison", PATMOS 2017

Outline

- Background
- Memristor basics
- **Logic using memory cells**
- Processing in Memory - the mMPU
- System design using the mMPU
- Conclusions

MAGIC – Memristor Aided loGIC

Initialize OUT to R_{ON}

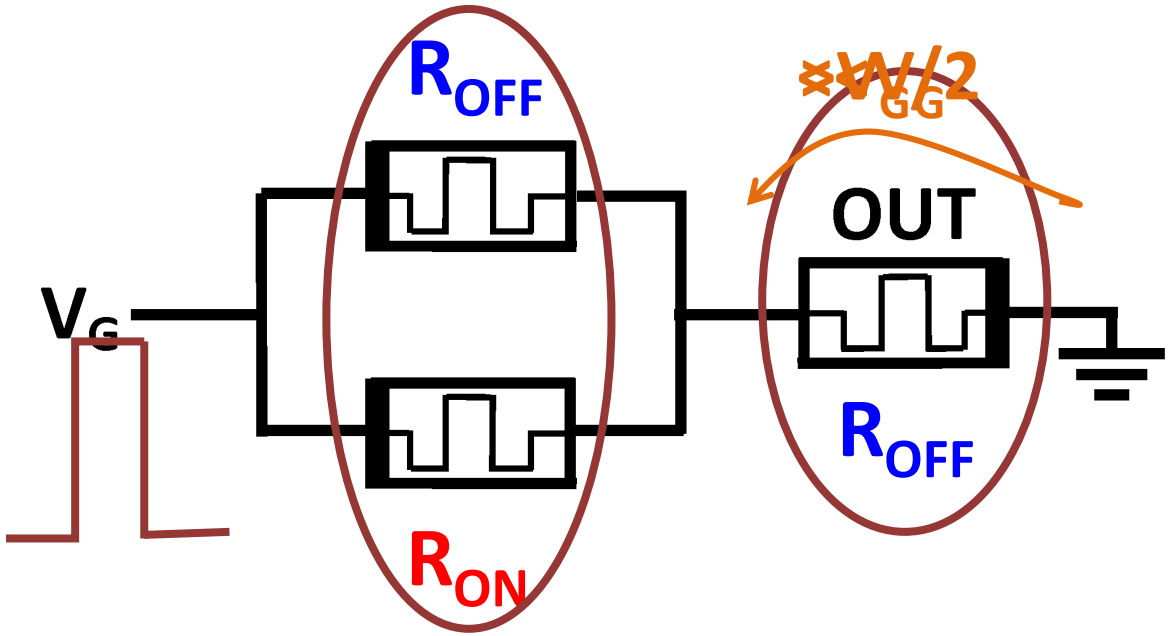
R_{ON} = Logic '1'

R_{OFF} = Logic '0'

$R_{OFF} >> R_{ON}$

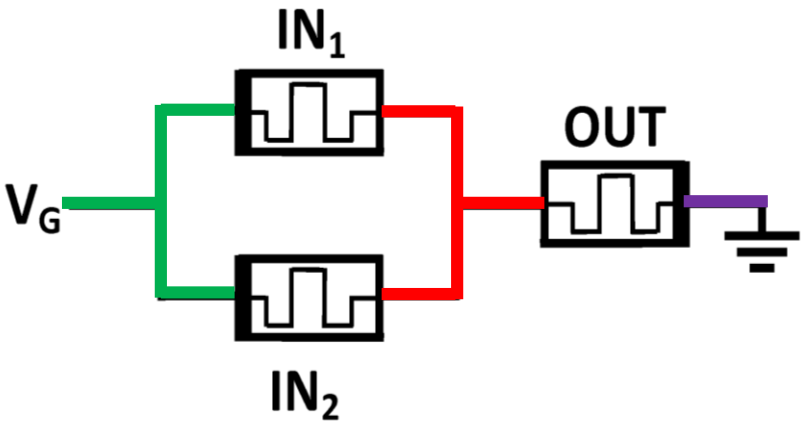
IN_1	IN_2	OUT
0	0	1
0	1	0
1	0	0
1	1	0

NOR Operation



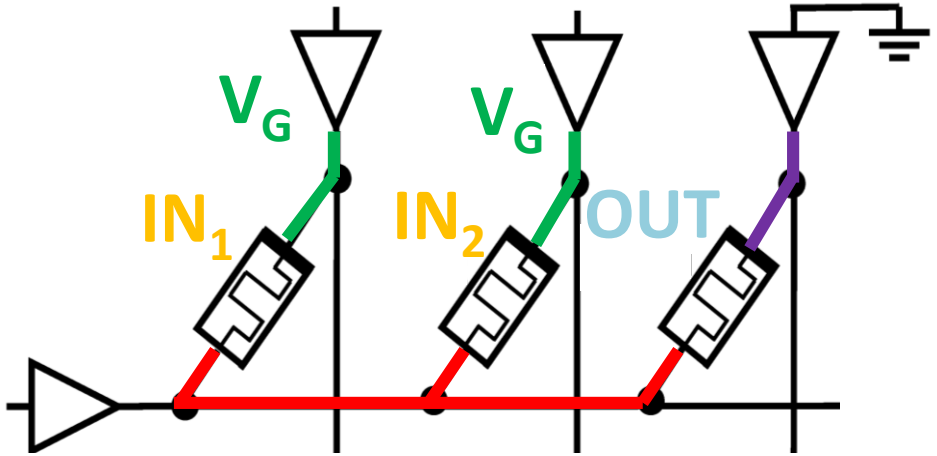
S. Kvatinsky *et al.*, "MAGIC – Memristor Aided LoGIC," TCAS II, Nov. 2014

MAGIC NOR in Memristive Crossbar



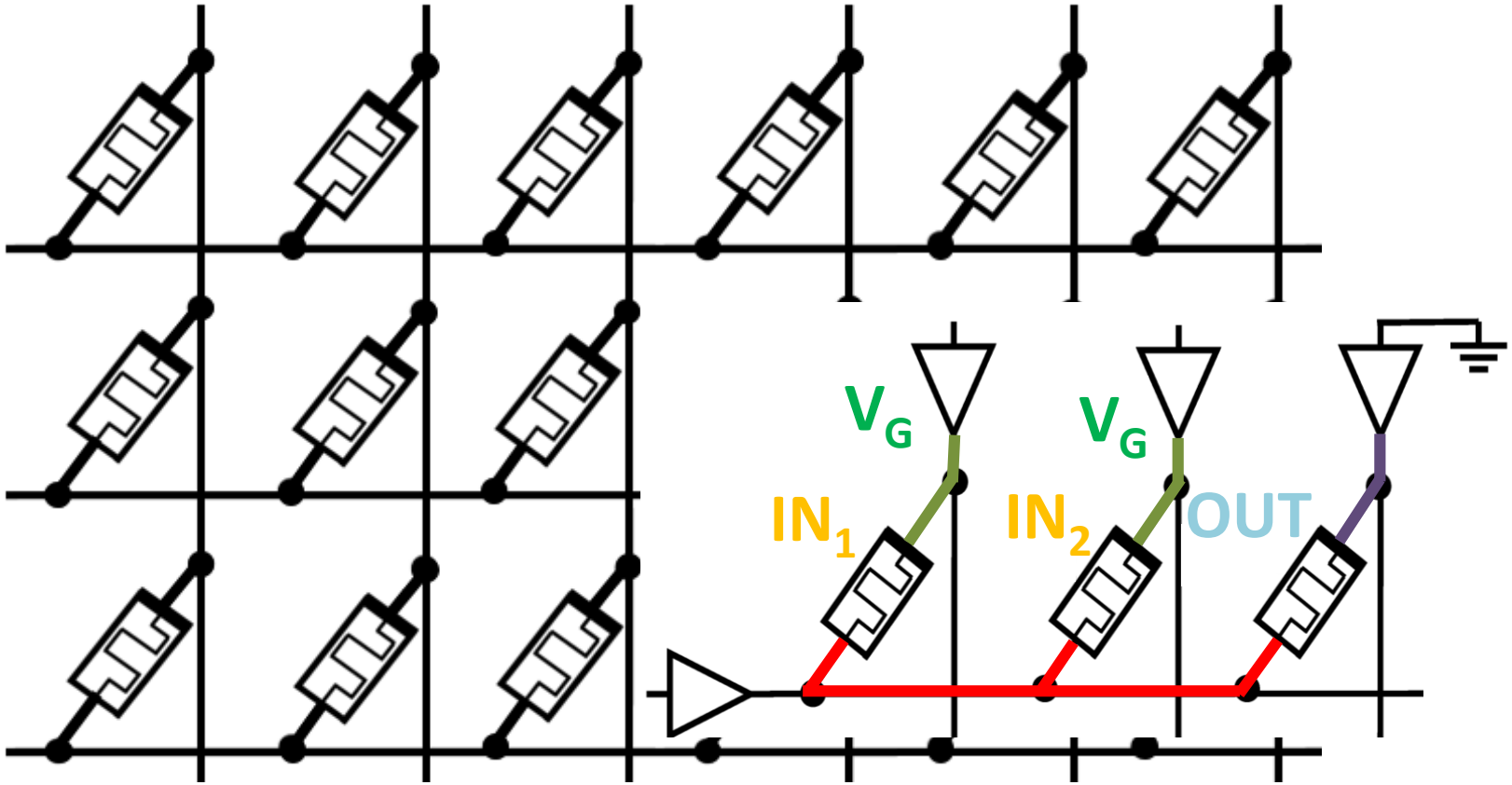
Functionally complete

Crossbar Compatible

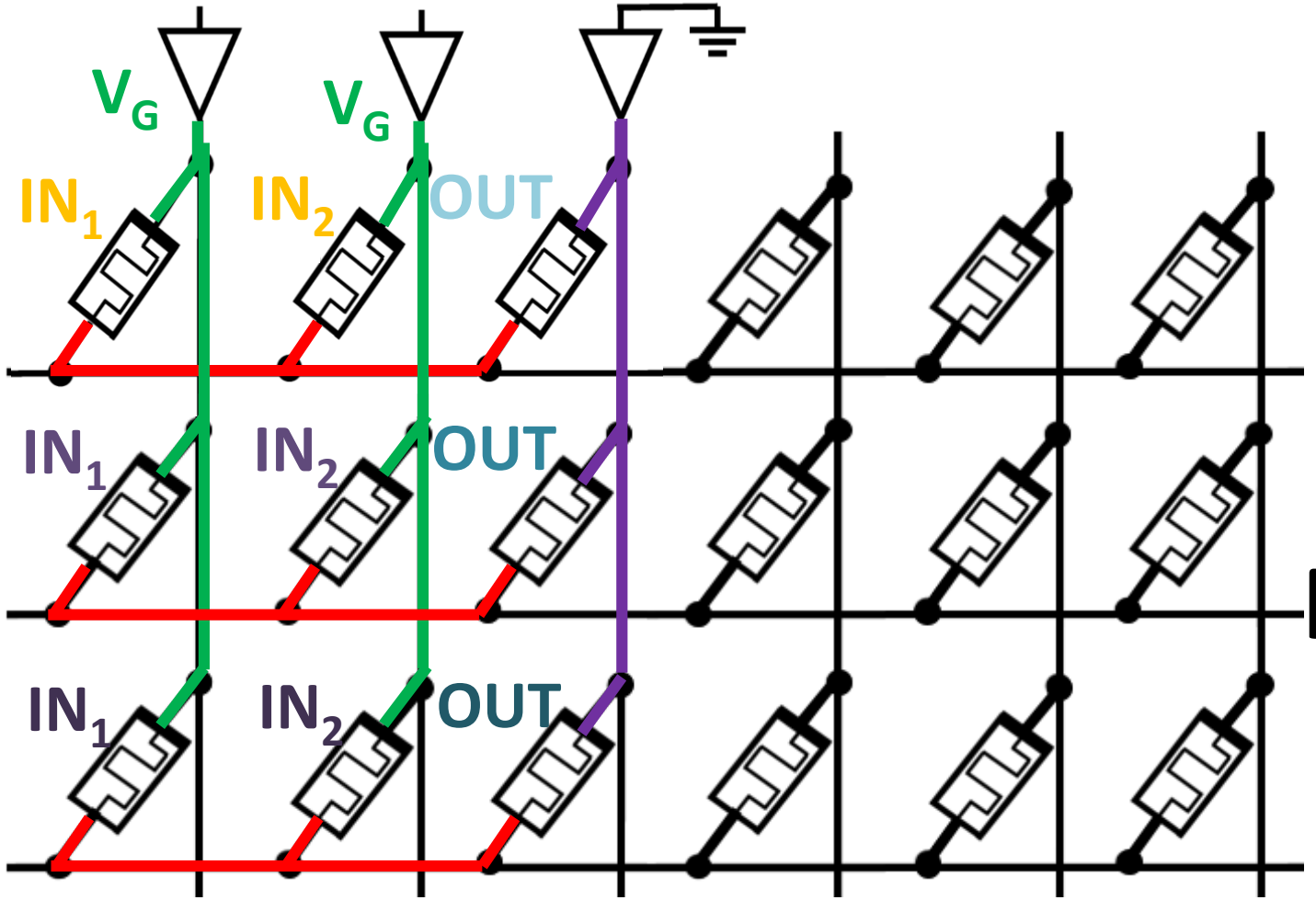


ASIC²

MAGIC NOR in a Memristive Memory



Parallel Execution of MAGIC Gates



Efficient SIMD Realization

N. Talati *et al.*, "Logic Design within Memristive Memories using Memristor-Aided loGIC (MAGIC)," *TNANO*, 2016

Outline

- Background
- Memristor basics
- Logic using memory cells
- **Processing in Memory - the mMPU**
- System design using the mMPU
- Conclusions

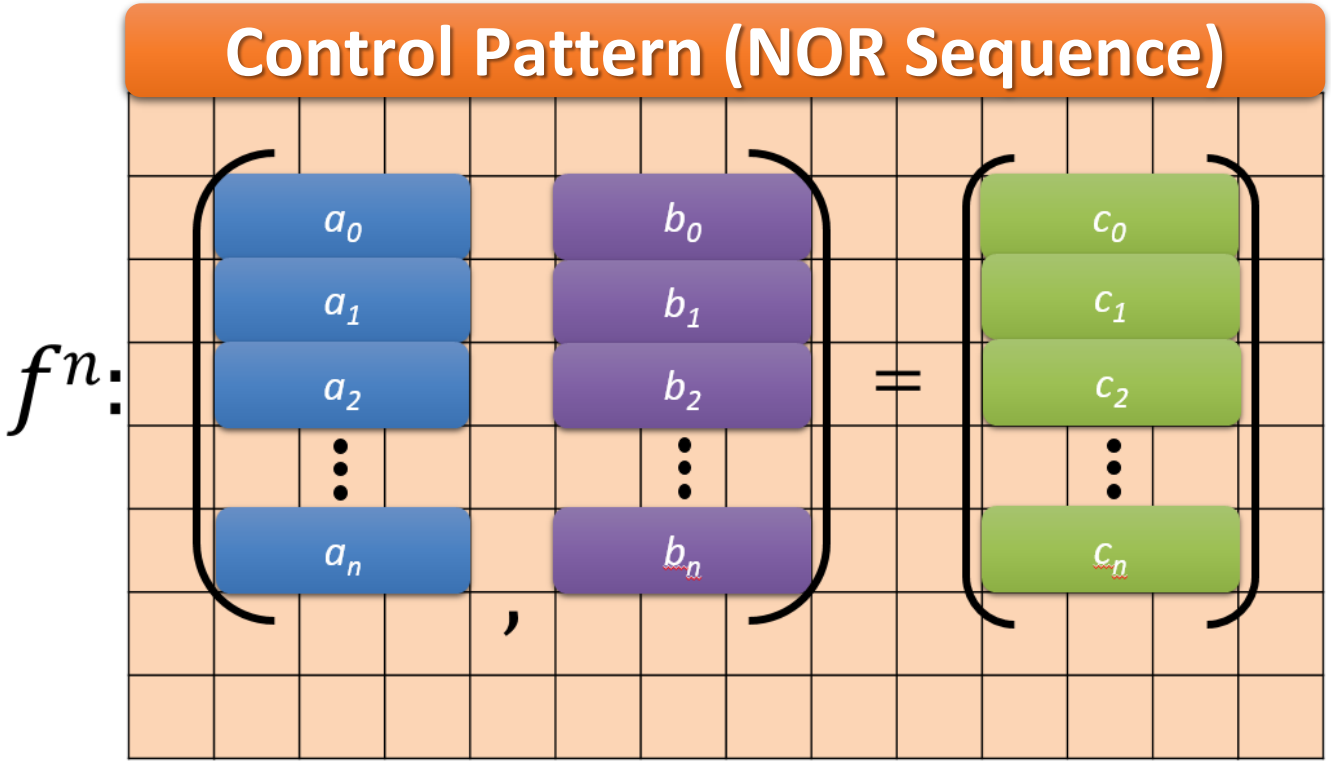
The Idea: Throughput Improvement

- Locate element computation to a single row

  => Execution of a single instance in each row in parallel

- Implement desired function using NOR/NOT sequence

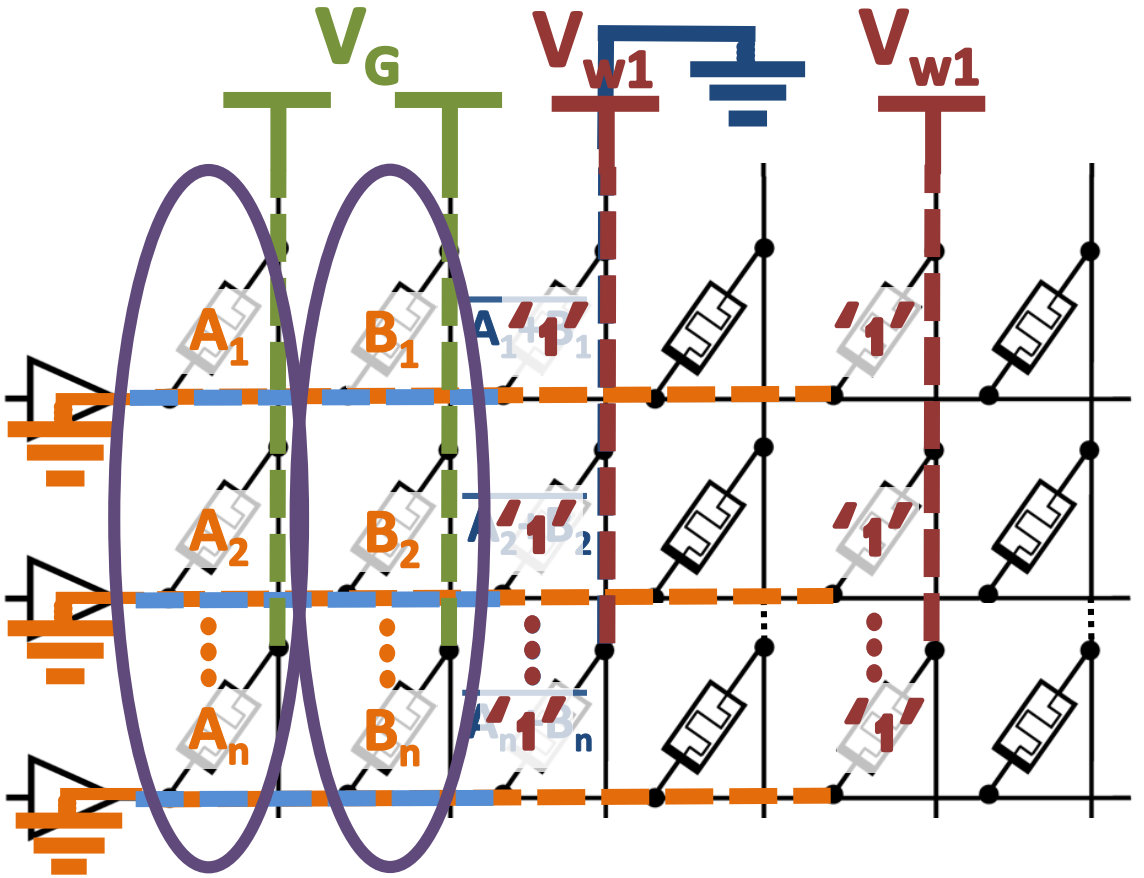
$$Throughput = \frac{\#instances}{Latency}$$



ASIC²

Example: In-Memory Parallel Execution

Initialize NOR(A_i,B_i) OUT to '1'

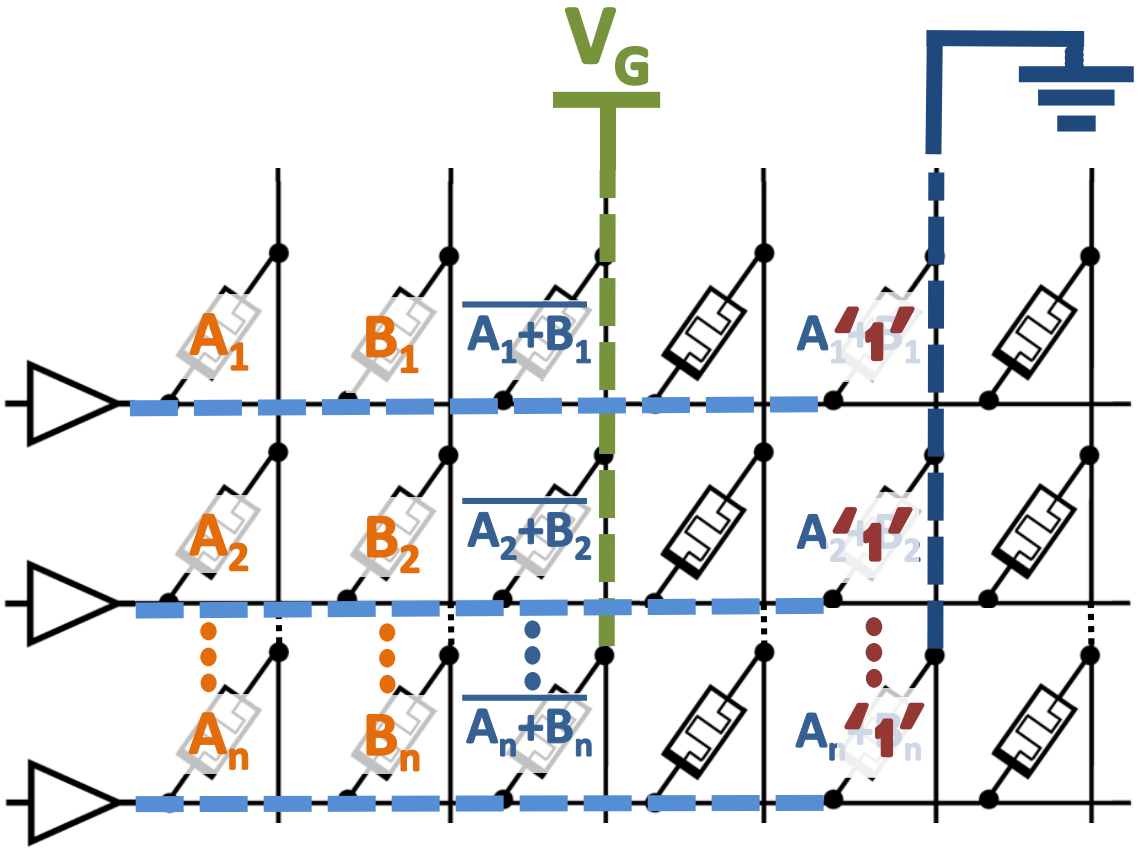


$$OR(A_i, B_i)$$
$$\forall i = 1, \ldots, N$$

$$A \text{ or } B = not(A \text{ nor } B)$$

Example: In-Memory Parallel Execution

NOT($\overline{A+B}$)



$$OR(A_i, B_i)$$
$$\forall i = 1, \ldots, N$$

$$A \text{ or } B = not(A \text{ nor } B)$$

Total*:

Latency: 2 steps

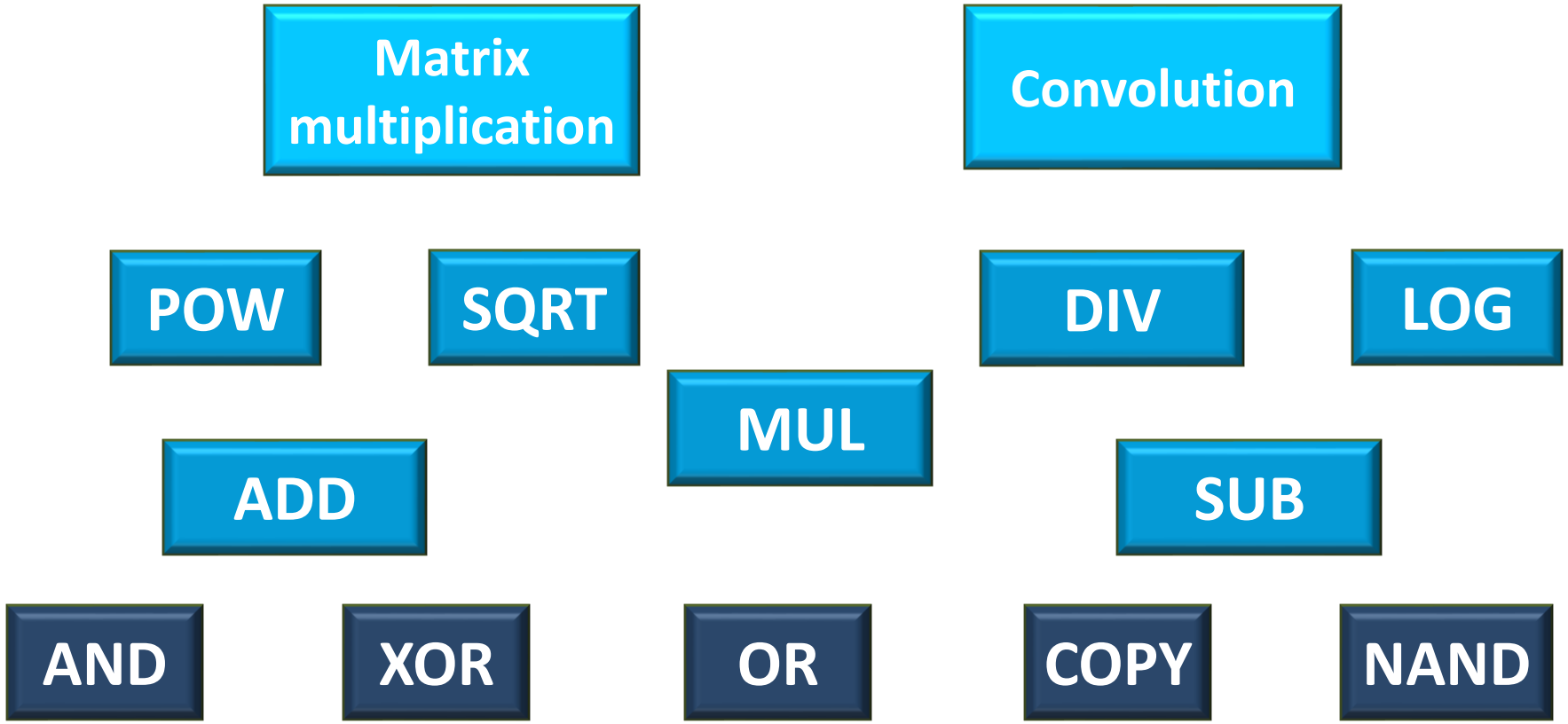
Area: (2+2) cells

Energy: 2 operations

* Per element, ignoring initialization

ASIC²

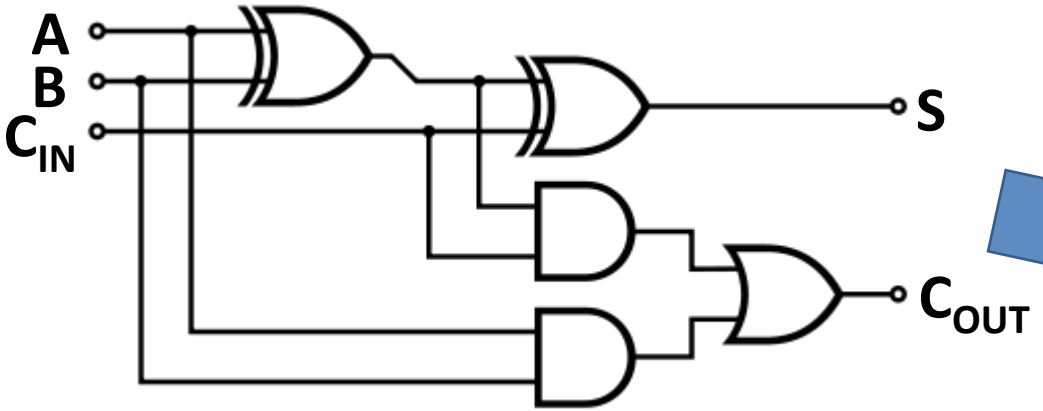
Hierarchy of Logical Functions

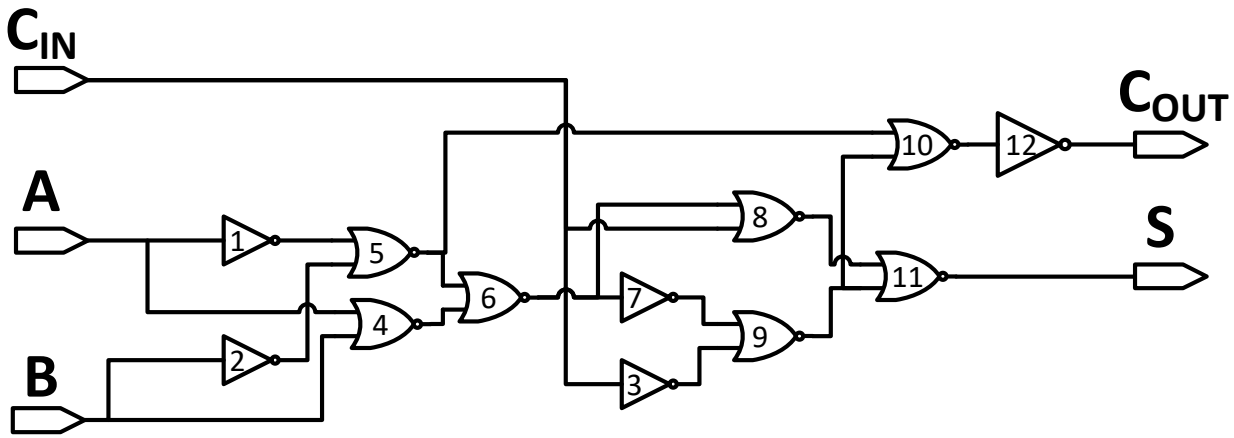


MAGIC NOR/NOT

Complete logic family

Example: Full Adder (1)





$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = A \cdot B + B \cdot C_{in} + A \cdot C_{in}$$

- Generate NOR/NOT sequence
- Existing CAD tools do it very well!
- **Can reuse interim results to save space!**

Total[1]:

Latency: 12 steps[2]

Area: <=(3+12) cells

Energy: 12 operations

[1] Per element, ignoring initialization

[2] Can be done w/ 9 NORs

ASIC²

R. Ben Hur, *et al.*, "Synthesis and Mapping of Boolean Functions for Memristor Aided Logic (MAGIC)", ICCAD 2017

Example: Full Adder (2)

Naïve column allocation

full_adder_12gates.v. in/out+gates = 3/2+10;

```
 1: T1 = NOT I1          % alloc: R3 = NOT I1
 2: T2 = NOT I2          % alloc: R4 = NOT I2
 3: T5 = NOR2 T1,T2      % alloc: R7 = NOR2 R3,R4
 4: T4 = NOR2 I1,I2      % alloc: R6 = NOR2 I1,I2
 5: T6 = NOR2 T5,T4      % alloc: R8 = NOR2 R7,R6
 6: T8 = NOR2 T6,I3      % alloc: R10 = NOR2 R8,I3
 7: T7 = NOT T6          % alloc: R9 = NOT R8
 8: T3 = NOT I3          % alloc: R5 = NOT I3
 9: T9 = NOR2 T7,T3      % alloc: R11 = NOR2 R9,R5
10: O11 = NOR2 T8,T9     % alloc: R1 = NOR2 R10,R11
11: T10 = NOR2 T5,T9     % alloc: R12 = NOR2 R7,R11
12: O12 = NOT T10        % alloc: R2 = NOT R12
```

Smart column allocation

full_adder_12gates.v. in/out+gates = 3/2+10;

```
 1: T1 = NOT I1          % alloc: R3 = NOT I1
 2: T2 = NOT I2          % alloc: R1 = NOT I2
 3: T5 = NOR2 T1,T2      % alloc: R2 = NOR2 R3,R1
 4: T4 = NOR2 I1,I2      % alloc: R3 = NOR2 I1,I2
 5: T6 = NOR2 T5,T4      % alloc: R1 = NOR2 R2,R3
 6: T8 = NOR2 T6,I3      % alloc: R3 = NOR2 R1,I3
 7: T7 = NOT T6          % alloc: R4 = NOT R1
 8: T3 = NOT I3          % alloc: R1 = NOT I3
 9: T9 = NOR2 T7,T3      % alloc: R5 = NOR2 R4,R1
10: O11 = NOR2 T8,T9     % alloc: R1 = NOR2 R3,R5
11: T10 = NOR2 T5,T9     % alloc: R3 = NOR2 R2,R5
12: O12 = NOT T10        % alloc: R2 = NOT R3
```

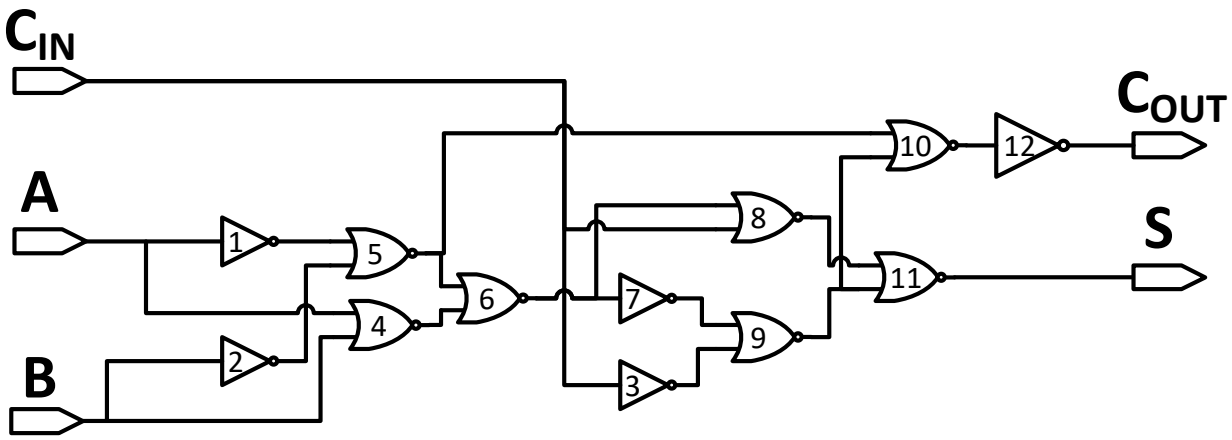
No Cell Reuse

12 cells

↓

5 cells

Best Cell Reuse



Total*:

Latency: 12 steps

Area (max): (3+12) cells

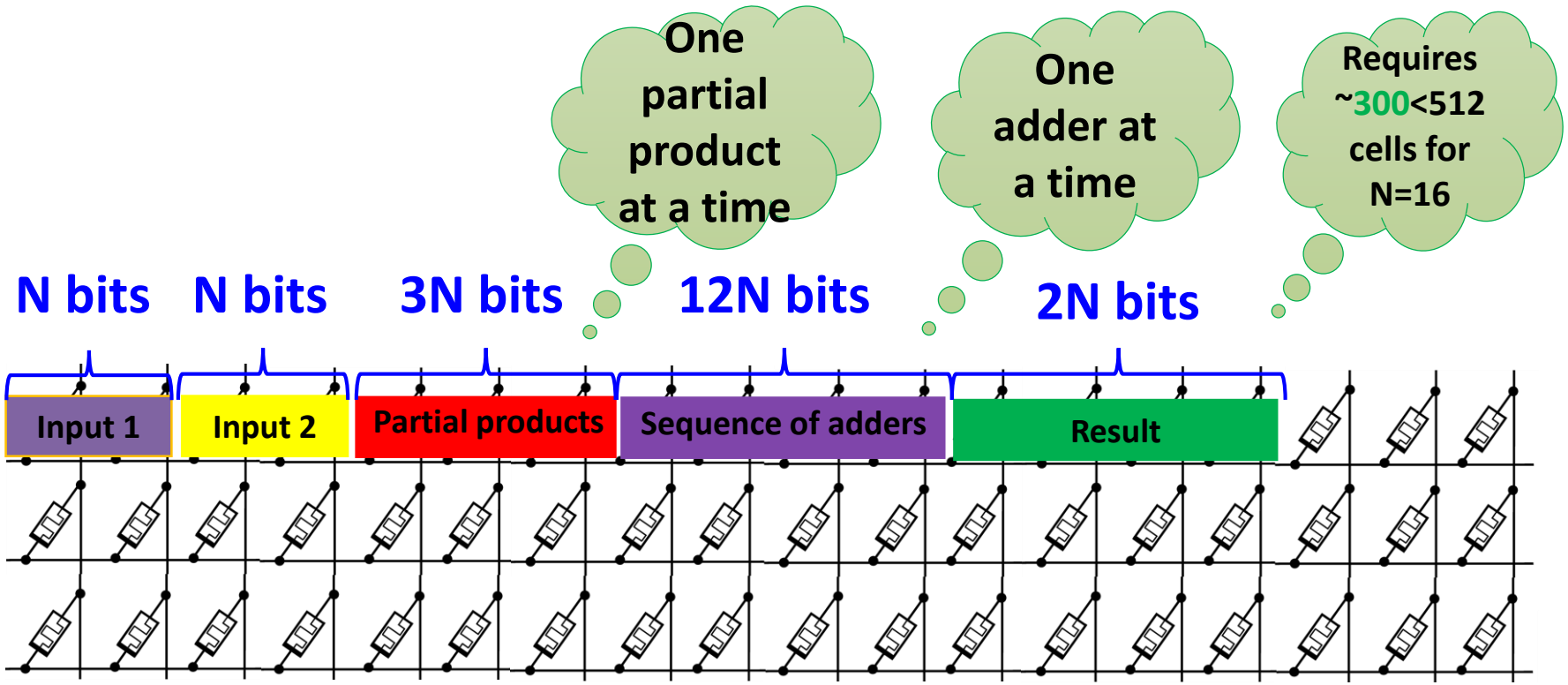
Area (min): (3+5) cells

Energy: 12 operations

* Per element, ignoring initialization

Smart Reuse → Column usage reduction!

Example: N-bit Multiplication




```
  1101
x 1011
------
  1011
 0000
1011
1011
--------
10001111
```

Addition ⊕

- **Showcase PIM implementation of complex operation**
- N=16 ➔ **~3700 NOR operations** $(O(N^2))$
- Naïve column allocation: **~3700 Cells** $(O(N^2))$
- Optimized manual/automictic allocation: **~300 Cells!** $(O(N))$

Total*:
Latency: ~3700 steps
Area (max): ~3700 cells
Area (min): ~300 cells
Energy: ~3700 operations

* Per element, ignoring initialization

ASIC²

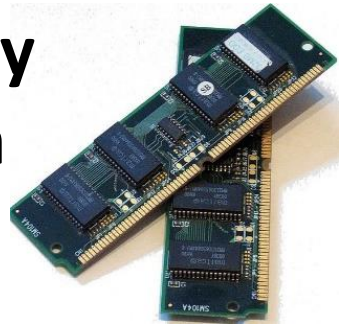
Ameer Haj-Ali *et al.*, "Efficient Algorithms for In-memory Fixed Point Multiplication Using MAGIC," 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018

Outline

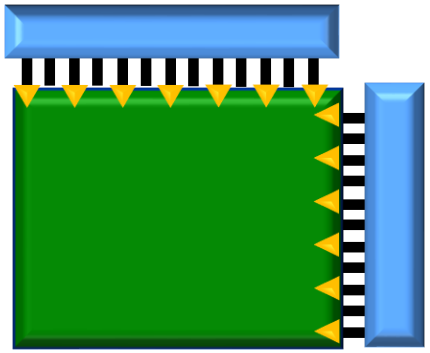
- Background
- Memristor basics
- Logic using memory cells
- Processing in Memory - the mMPU
- **System design using the mMPU**
- Conclusions

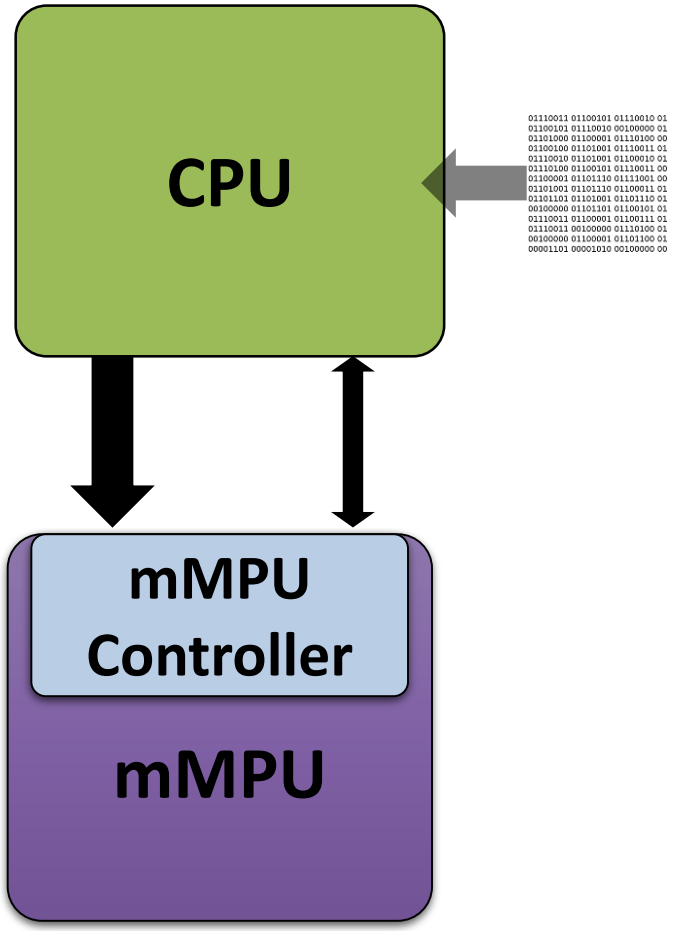
System Design using the mMPU

Memory Design

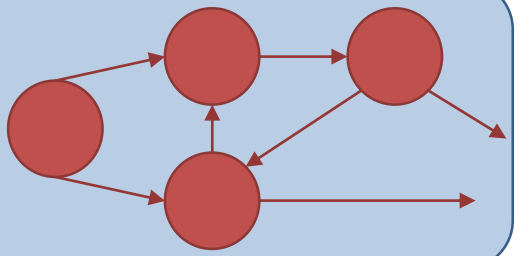


Periphery Design



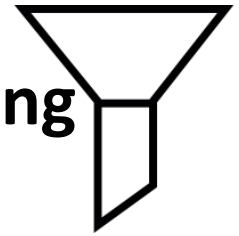


mMPU Controller Design and Optimization

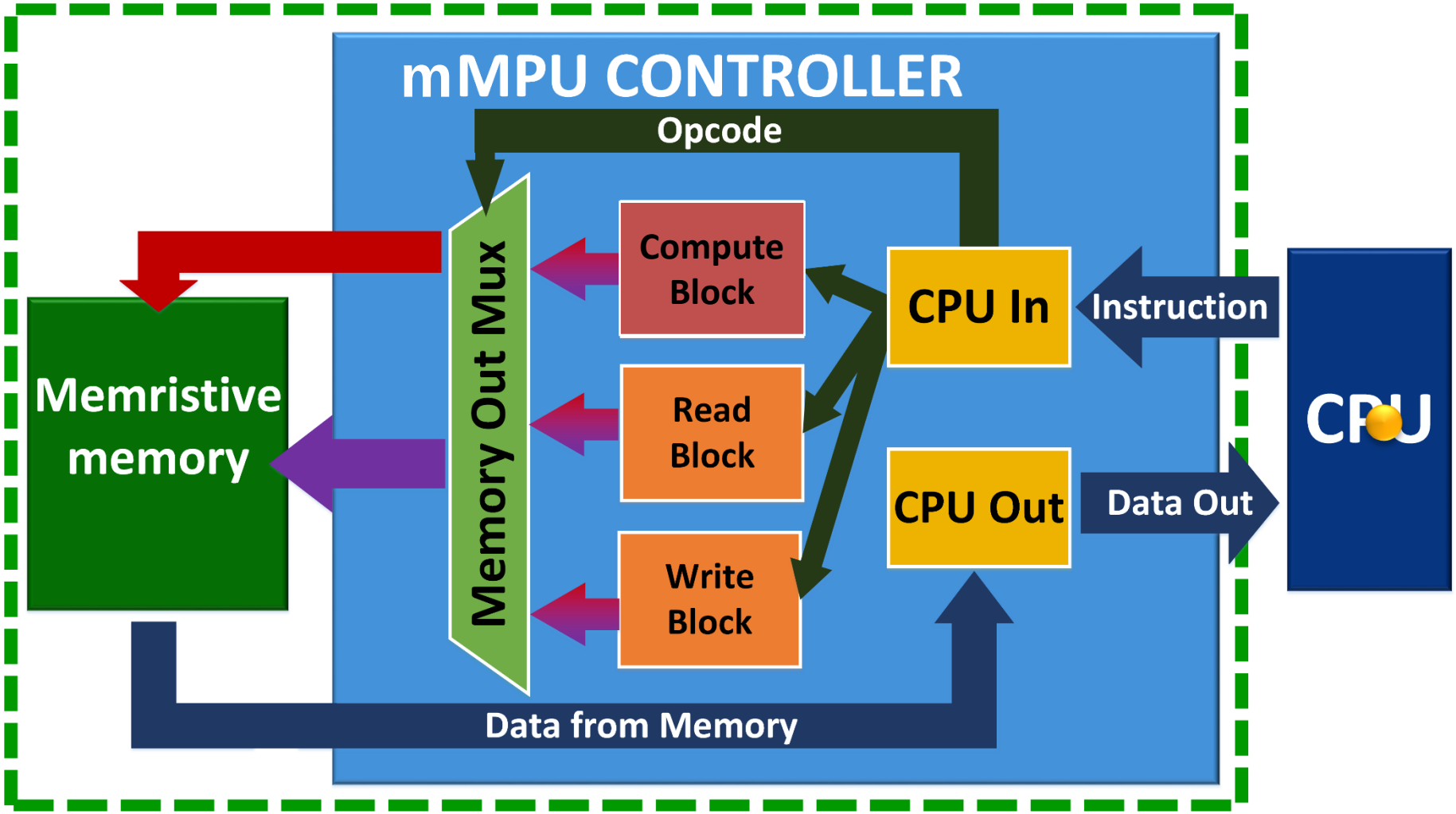


Applications

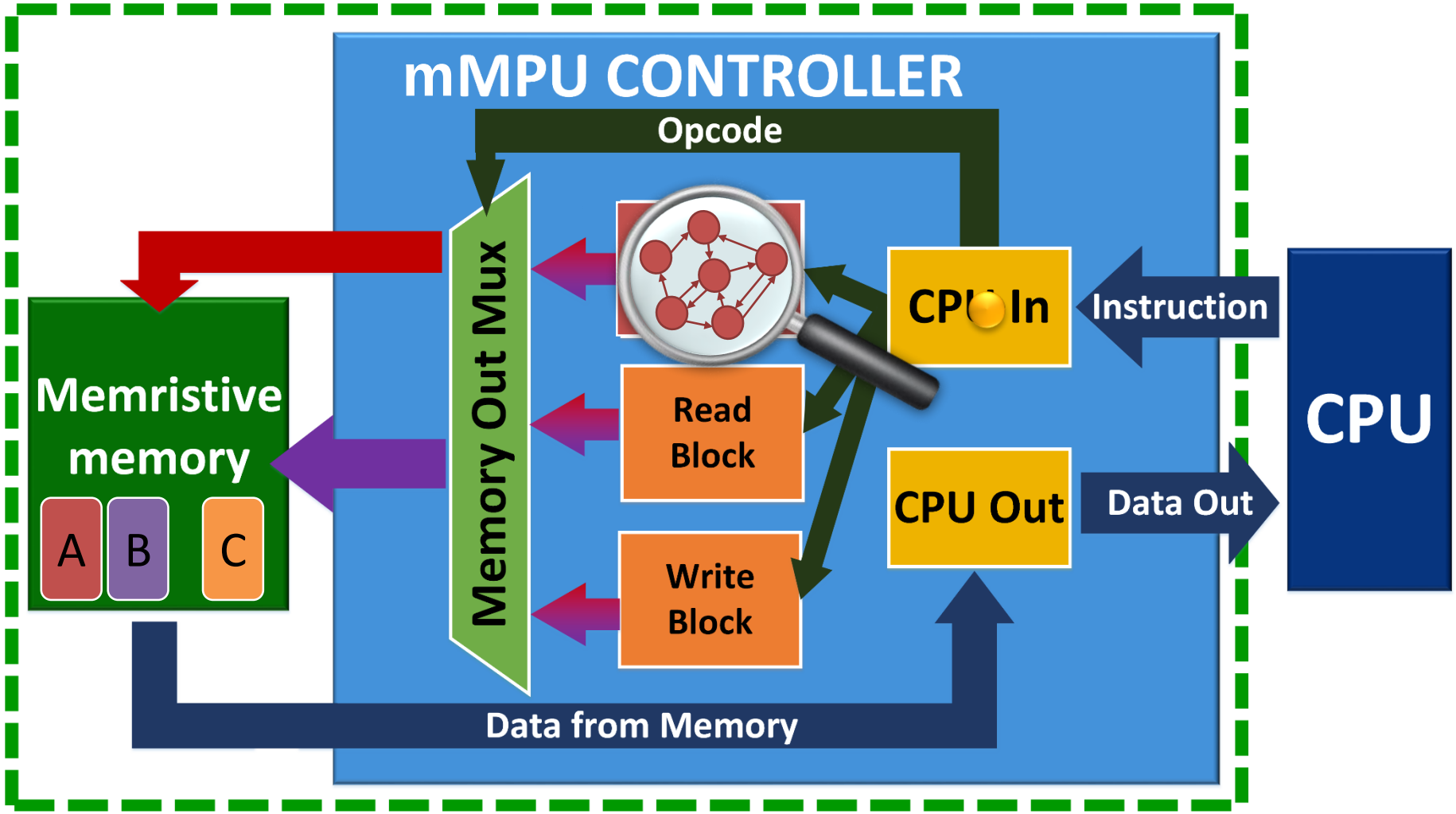
Programming Model



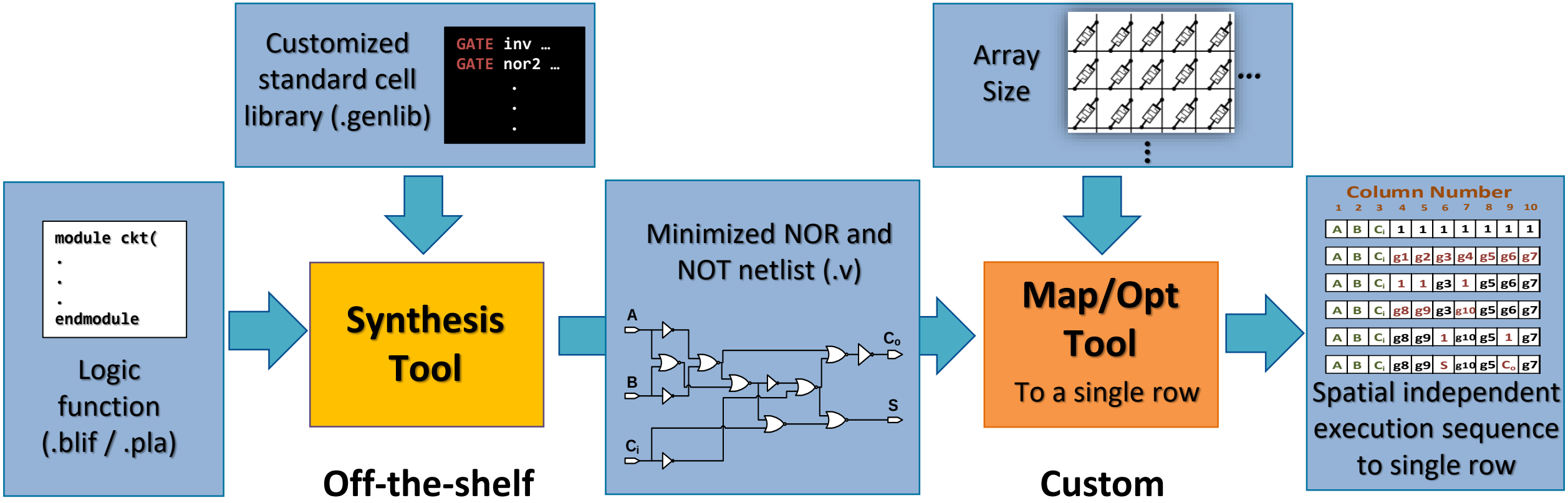
Challenge 1
mMPU Controller μ-architecture



Challenge 1: Arithmetic/Logical Operations in the mMPU

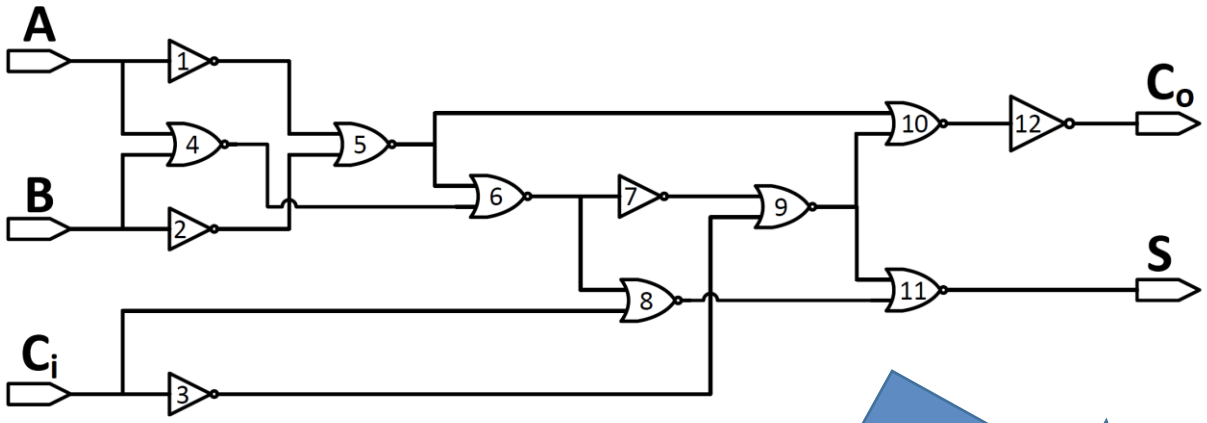


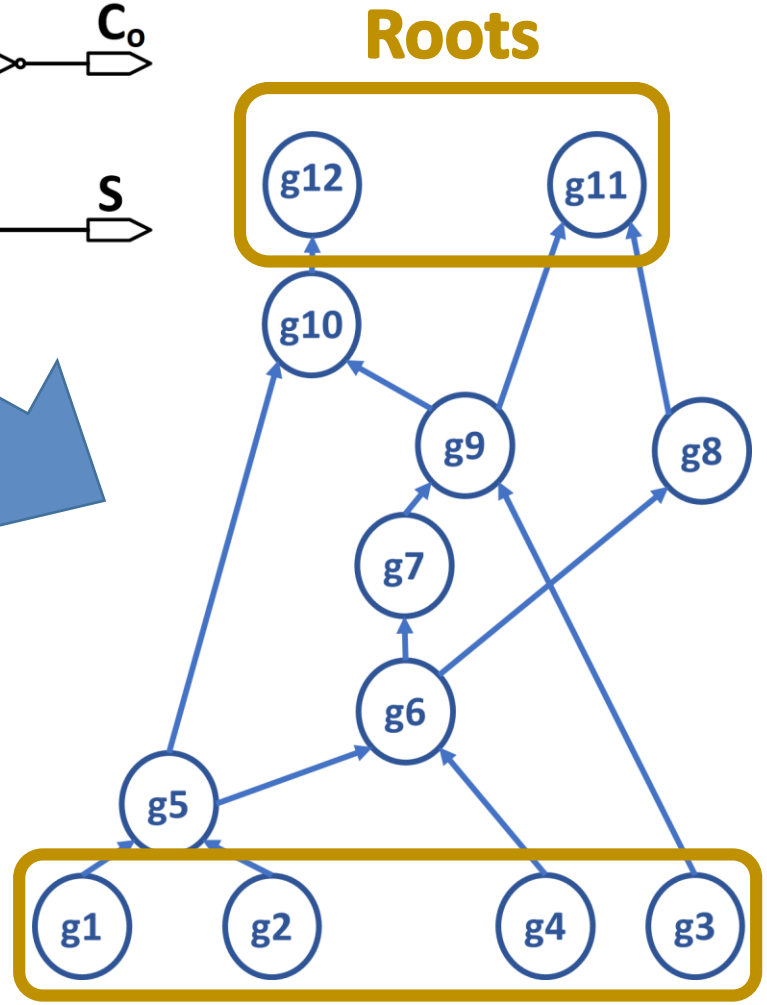
Challenge 2:
Code Generation and Storage allocation



- (adapted from) R. Ben Hur, *et al.*, "Synthesis and Mapping of Boolean Functions for Memristor Aided Logic (MAGIC)", ICCAD 2017
- R. Ben Hur, R. Ronen, *et al.*, "SIMPLER MAGIC: Synthesis and Mapping of In-Memory Logic Executed in a Single Row to Improve Throughput", submitted

Challenge 2a: Mapping into a Limited-size Array





- Netlist ➔ DAG*
- Traverse (DFS) – similar to register allocation
- The execution order influences:
  - How many cells are enough for the execution
  - How many initialization cycles are necessary

ASIC²

**DAG: Direct Acyclic Graph*

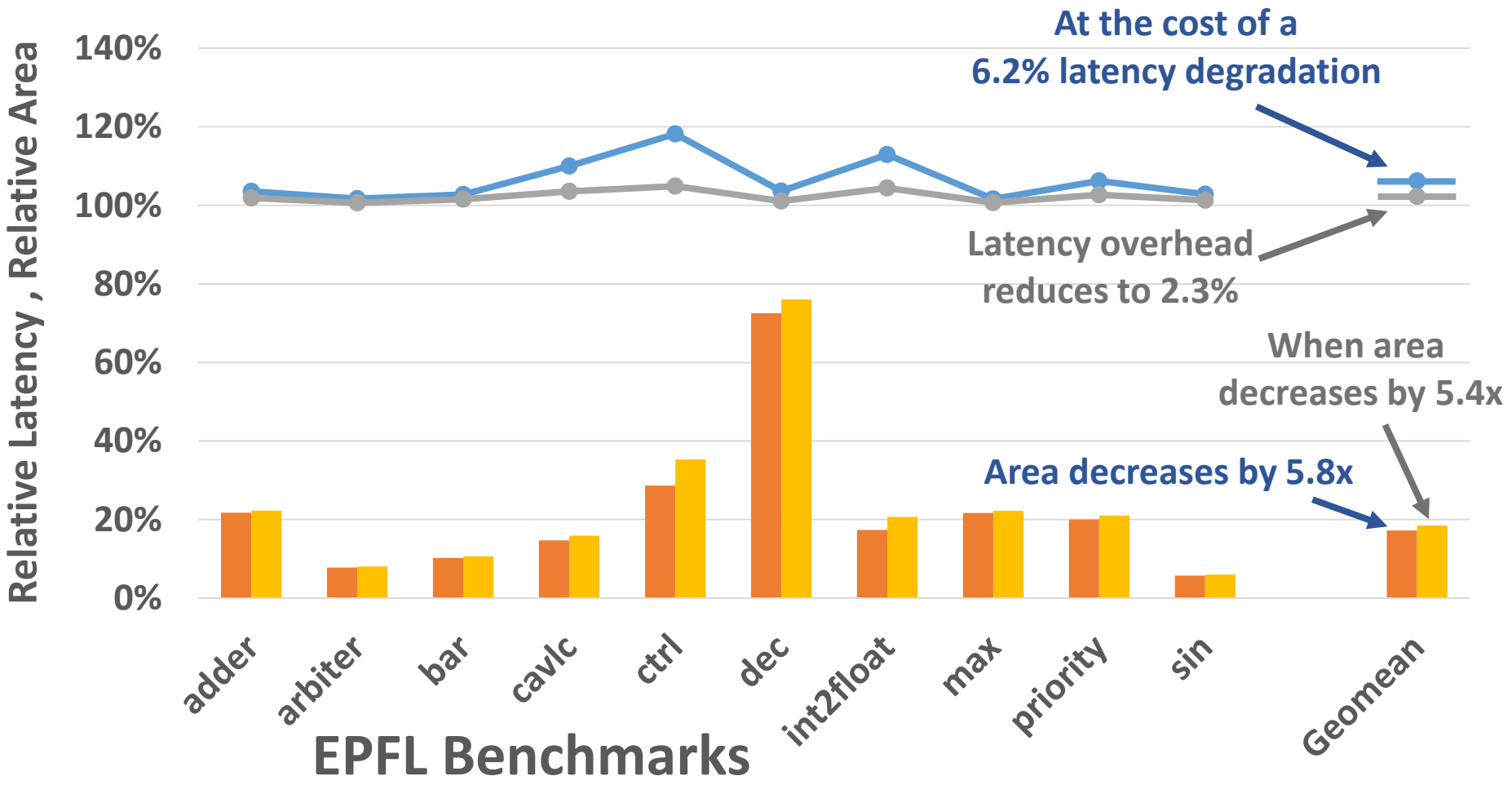
Challenge 2a: Mapping into a Limited-size Array

Cycle	Netlist	Physical Allocation
1	w1 = NOT A	R2 = NOT I1
2	w2 = NOT B	R1 = NOT I2
3	w5 = NOR2 w1,w2	R4 = NOR2 R2,R1
4	w4 = NOR2 A,B	R3 = NOR2 I1,I2
5	w6 = NOR2 w5,w4	R5 = NOR2 R4,R3
6		**ReUsing 3 registers = 2 1 3
7	w8 = NOR2 w6,Cin	R2 = NOR2 R5,I3
8	w7 = NOT w6	R1 = NOT R5
9	w3 = NOT Cin	R3 = NOT I3
10		**ReUsing 1 registers = 5
11	w9 = NOR2 w7,w3	R5 = NOR2 R1,R3
12		**ReUsing 2 registers = 1 3
13	S = NOR2 w8,w9	R1 = NOR2 R2,R5
14	w10 = NOR2 w5,w9	R3 = NOR2 R4,R5
15		**ReUsing 3 registers = 2 4 5
16	Cout = NOT w10	R2 = NOT R3

Minimum: 8 columns (cells)

→ 16 cycles for execution of *N* instances

Challenge 2a: Latency/Area Relative to Unrestricted Area (No Reuse)



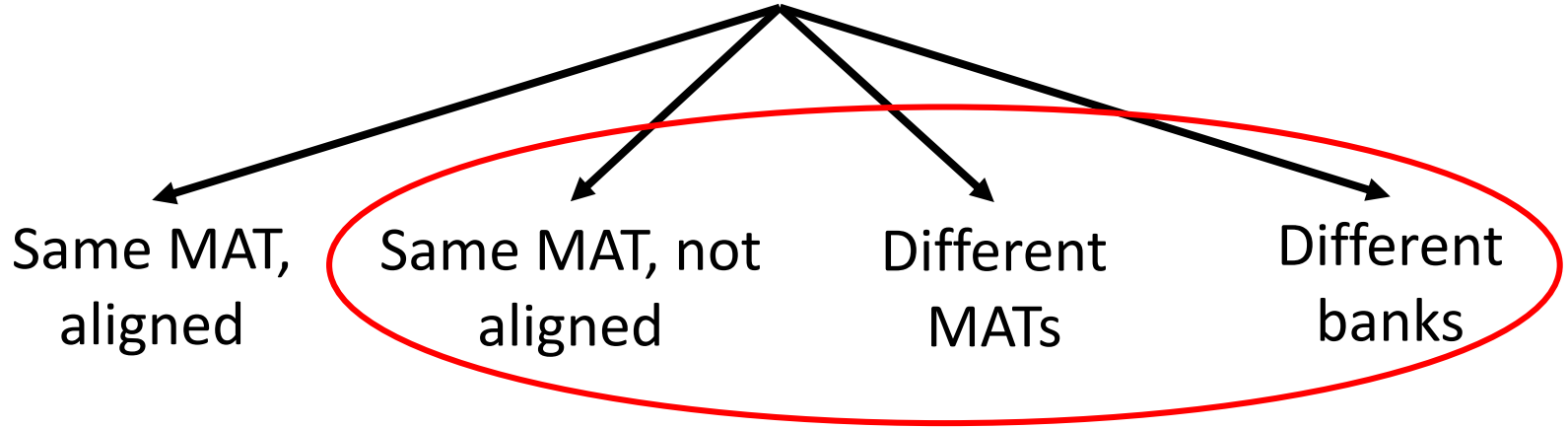
ASIC²

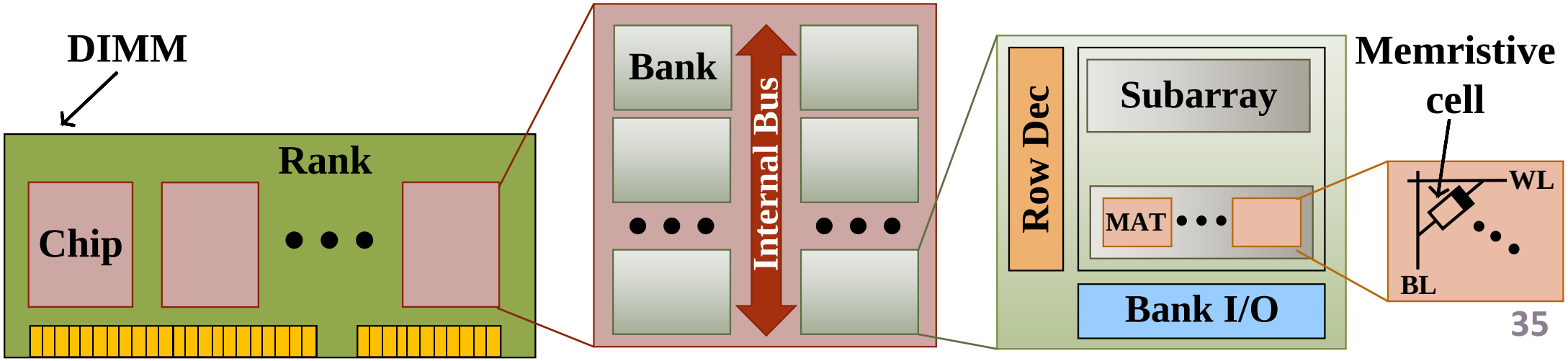
Challenge 3: Data Movement

Processing-in-memory Instruction

PIM_ADD
#Op1, #Op2, #Out

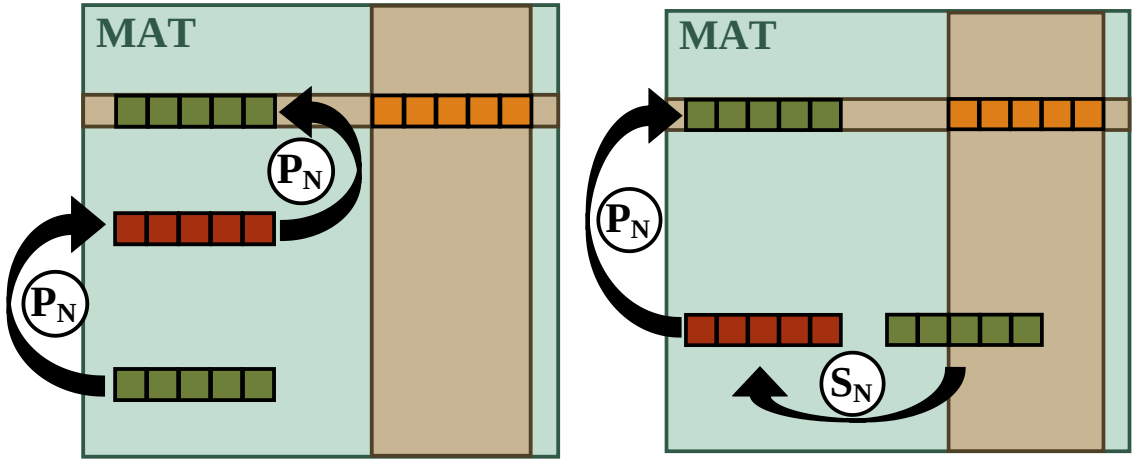
Relative locations of op1 and op2



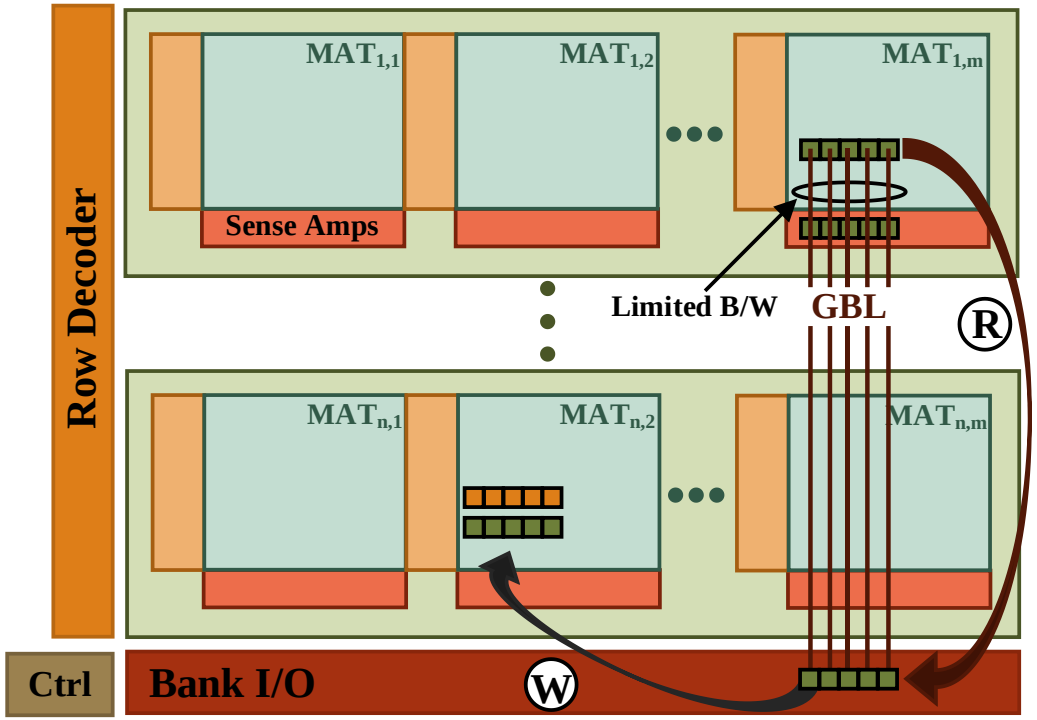


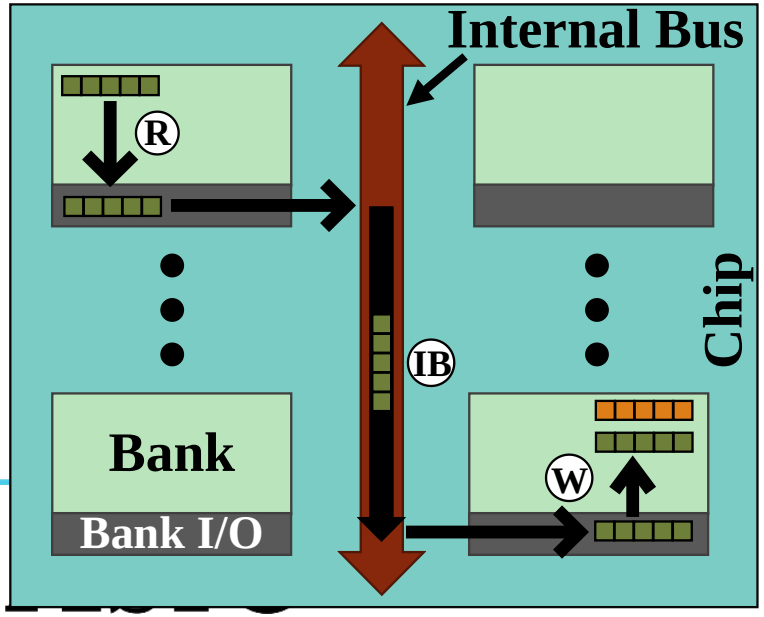
Challenge 3: Modes of Data Transfer

Intra-MAT data transfer



Intra-bank data transfer





Inter-bank data transfer

N. Talati *et al.*, "Practical Challenges in Delivering the Promises of Real Processing-in-Memory Machines," DATE 2018

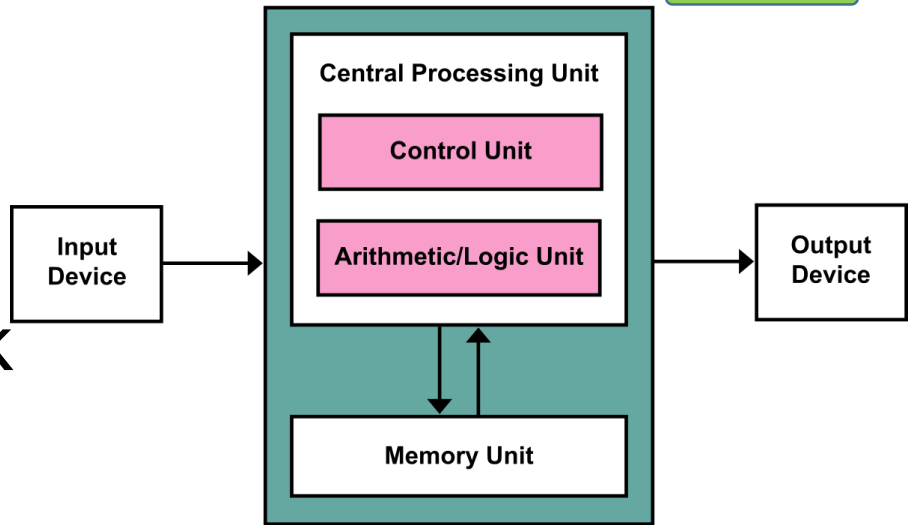
More mMPU Research Challenges

- Defining the mMPU operation
  - Instructions; Micro-Operation Format
  - Control Storage
  - Optimized compilation of mMPU "program"
  - Passing info from the mMPU to the runtime systems
- Data mapping & alignment
  - *"Where is address A+1?"*
  - Which variables stay in the same MAT?
  - Cost of data alignment
  - Memory controller to support PIM
- Understanding the electrical limitations
- Rigorous evaluation of mMPU benefits
- Endurance aspects

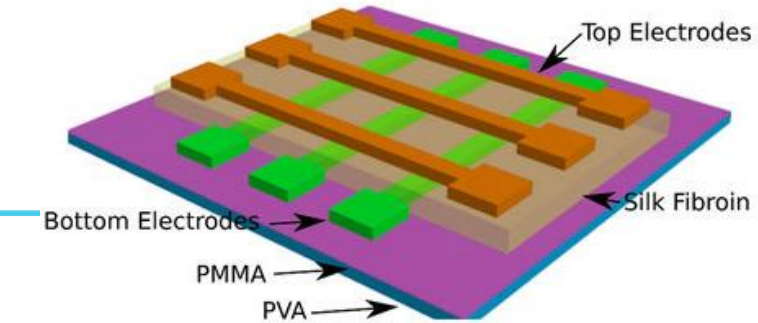
....

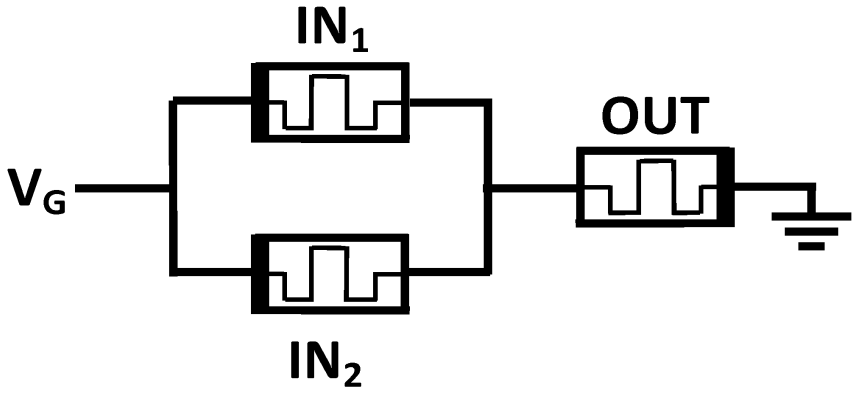
ASIC²

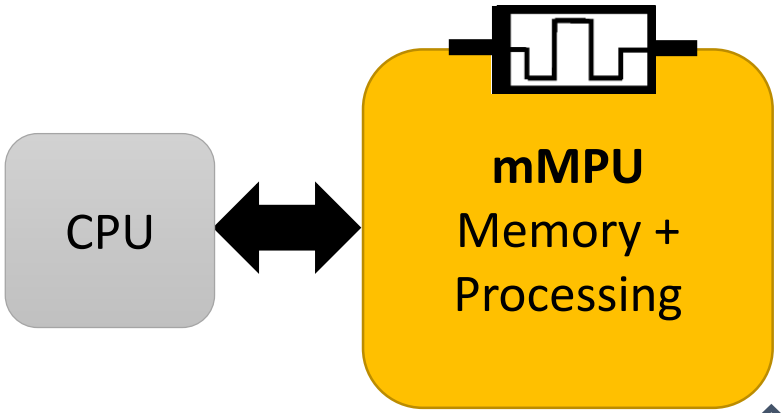
Conclusions



- Data transfer → the von Neumann bottleneck
- MAGIC – the scalable building block for processing in memory
- mMPU – a real processing-in-memory machine
  - Requires a new computing paradigm
- Our group aims to develop a working end-to-end mMPU system








Thanks!
ASIC² Summer 2018