Memristor Overwrite Logic (MOL) for Energy-Efficient In-Memory DNN

Khaled Alhaj Ali , Mostafa Rizk, Amer Baghdadi, Jean-Philippe Diguet, Jalal Jomaah

2020 IEEE International Symposium on Circuits & Systems
Virtual, October 10-21, 2020

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Lebanese University

ISCAS 2020

OUTLINE

1. Context and Motivation
2. Memristor-based logic design
3. MOL – Memristor Overwrite Logic for In-Memory Computing
4. Proposed computational memory
5. MOL-based In-memory DNN
6. Conclusion and future work

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

OUTLINE

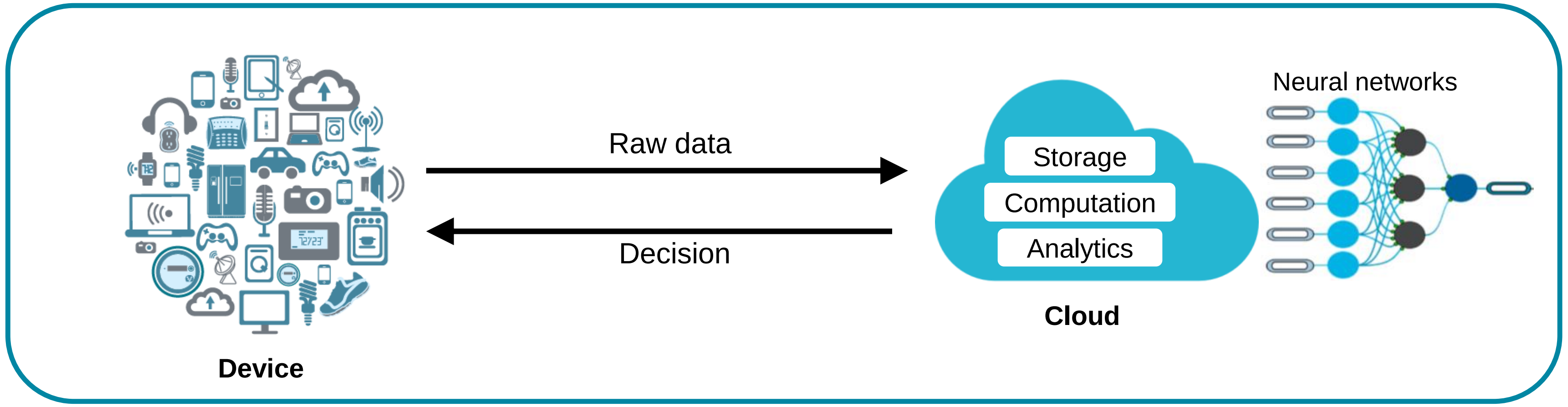
ISCAS 2020

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Context & motivation

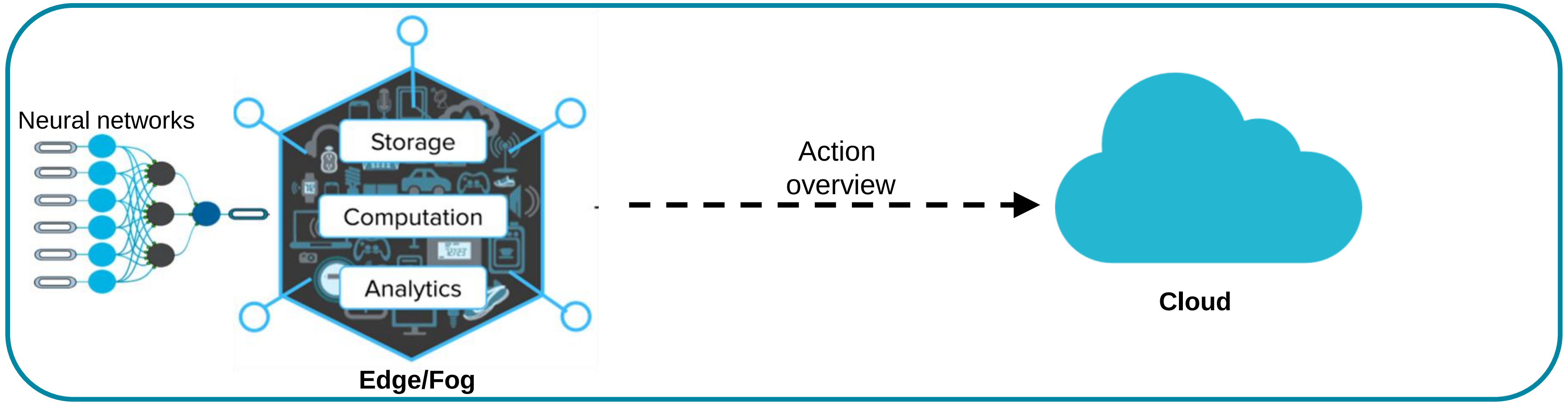
Cloud and Edge computing

Cloud computing



☹ Does not ensure the required real time response

Edge computing



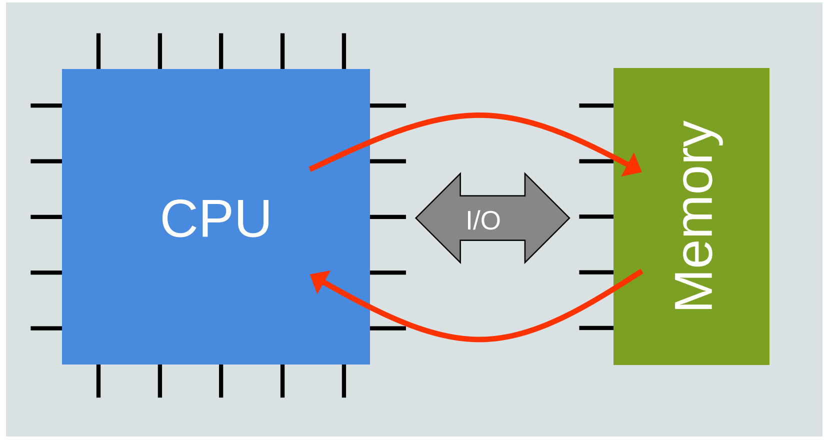
☹ Slow processing speed and high energy consumption

From Von Neumann to in-memory computing

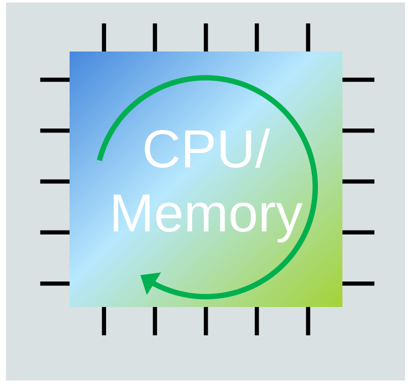
- Intensive data transfer between memory and CPU/GPU
- Large size of the processed data

Memristor-based logic design enables true in-memory computing

Von Neumann model



In-memory computing



ISCAS 2020

OUTLINE

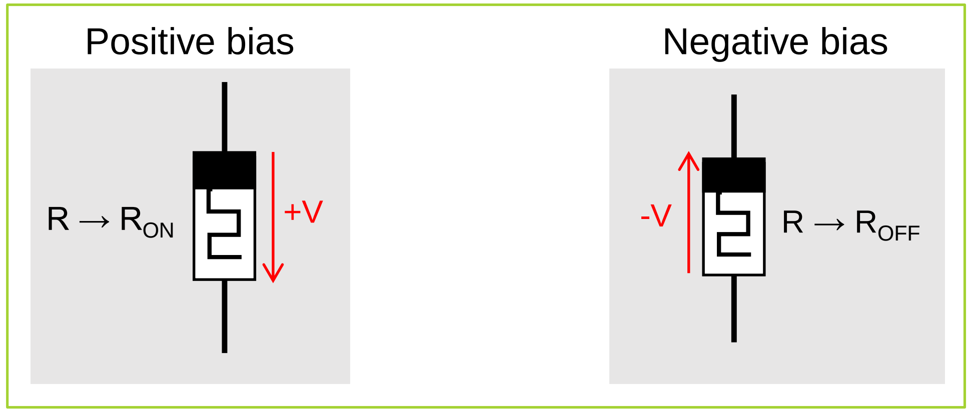
IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Basics of memristor technology

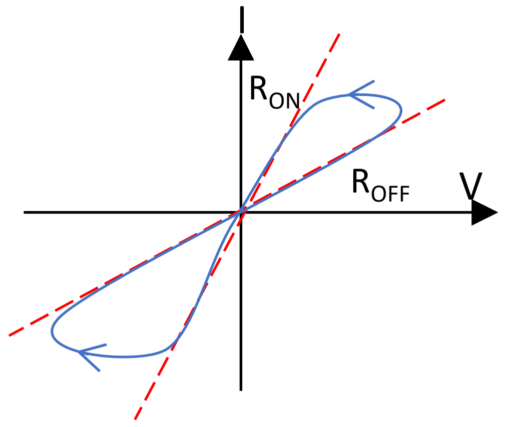
- Recent development of new non-volatile memory technologies (Memristor)
  - Theoretically predicted in 1971 by Chua
  - Received attention in 2008 (HP labs)
  - Triggered many efforts to explore their use in different applications

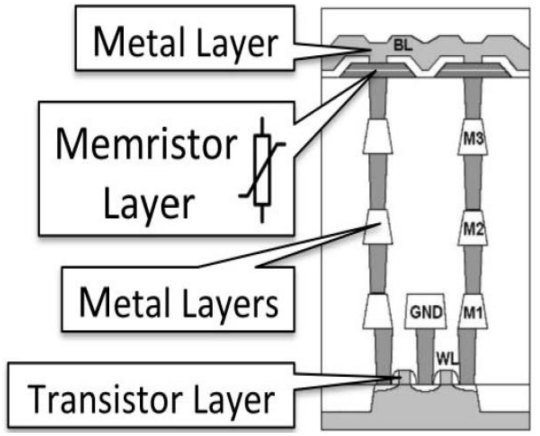
- Basic operation

Memorize last resistance state



Hysteresis loop

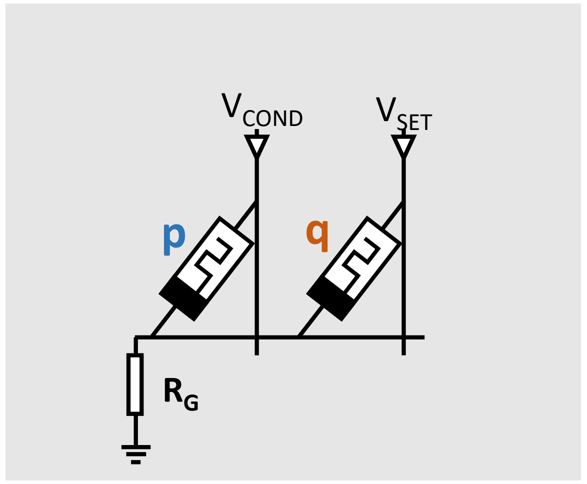




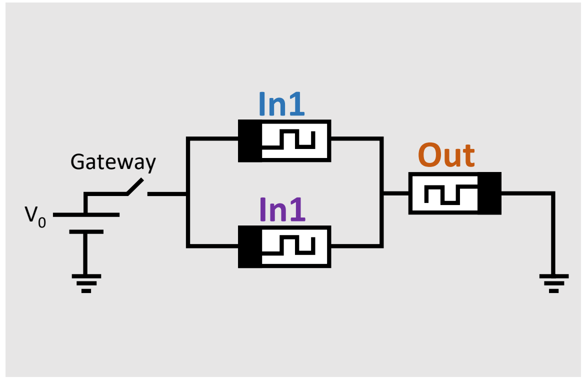
Limitations of existing logic design styles

- IMPLY and MAGIC-NOR:
  - ☹ Partial switching in IMPLY
  - ☹ State drift in IMPLY and MAGIC
  - ☹ Requirement of memristive devices with high R_{OFF} /R_{ON} ratio
  - ☹ High number of computational cycles

IMPLY



MAGIC-NOR



OUTLINE

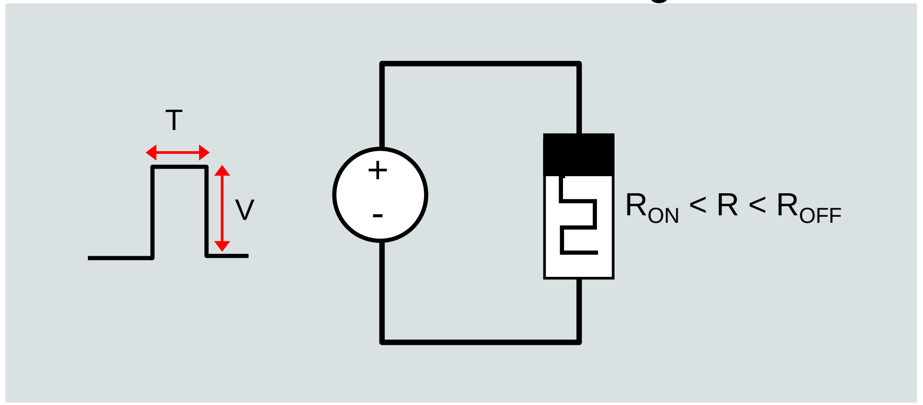
ISCAS 2020

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Digital representation of memristor

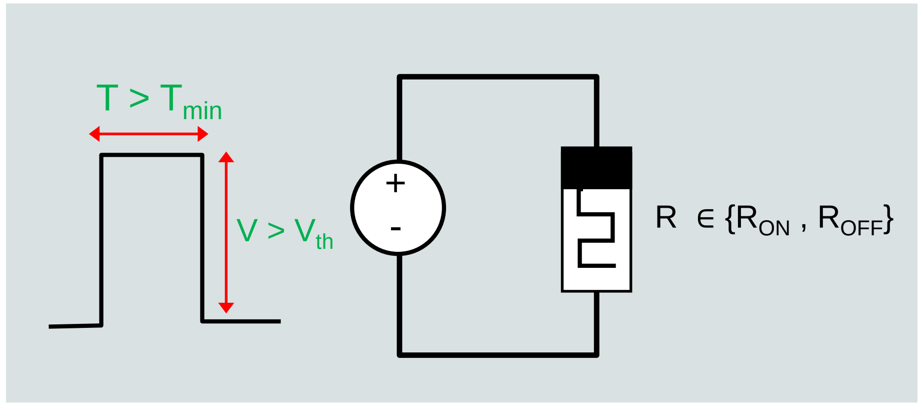
– Non sufficient magnitude or duration of the bias

Partial resistance switching



– V > Vth and T > Tmin

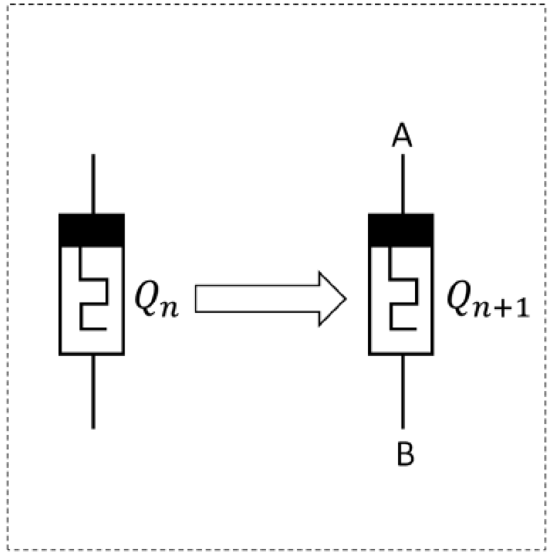
Binary resistance switching



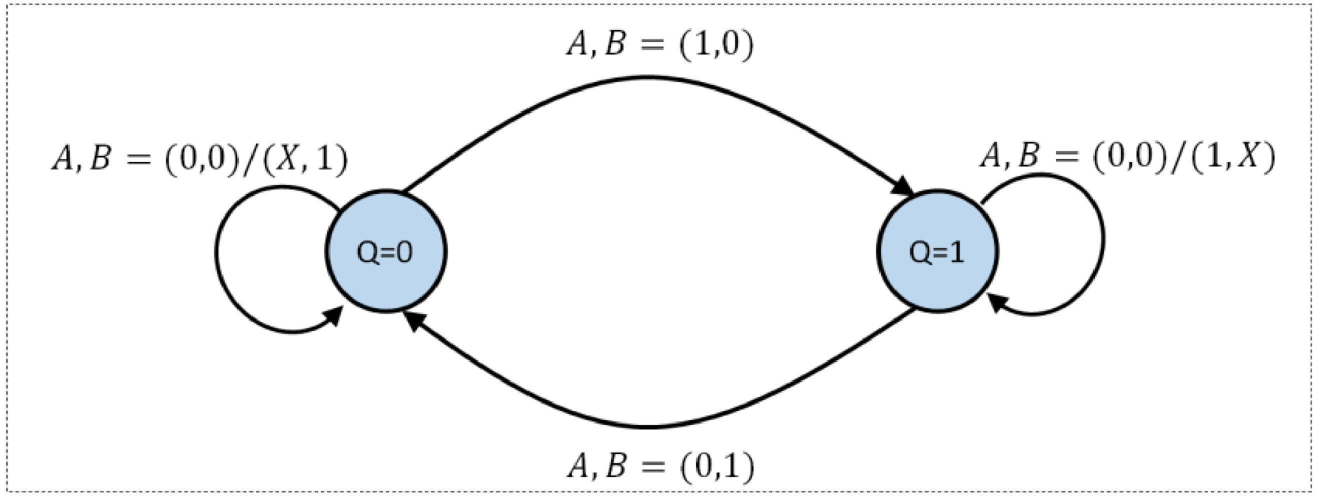
☺ We are able to define the internal state of the memristor in digital domain

Digital representation of memristor

Current & next state



Finite state machine (FSM)



State equation

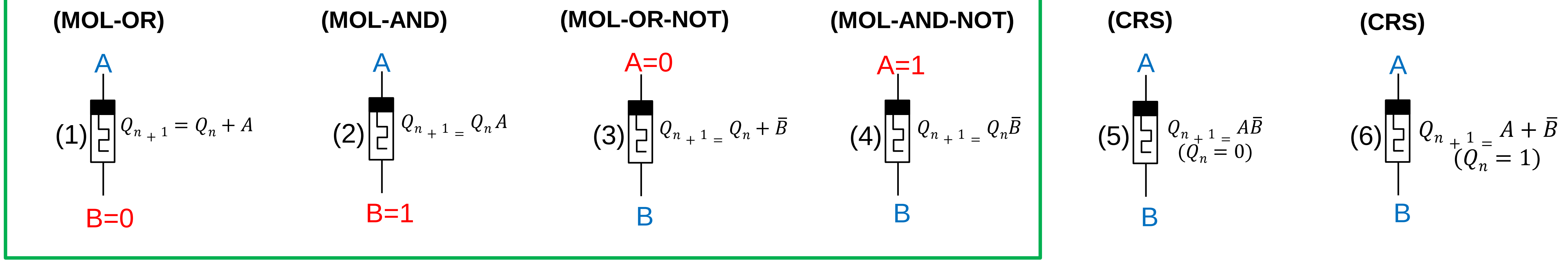
$$\boxed{\Rightarrow Q_{n+1} = Q_n A + Q_n \bar{B} + A\bar{B}}$$

MOL logic design

- Derived logic cases

$$Q_{n+1} = Q_nA + Q_n\bar{B} + A\bar{B} \quad \Rightarrow \quad Q_{n+1} = \begin{cases} Q_n + A\,, & B=0, \quad case:1 \\ Q_nA\,, & B=1, \quad case:2 \\ Q_n + \overline{B}\,, & A=0, \quad case:3 \\ Q_n\overline{B}\,, & A=1, \quad case:4 \\ A\overline{B}\,, & Q_n=0, \quad case:5 \\ A+\overline{B}\,, & Q_n=1, \quad case:6 \end{cases}$$

MOL operations



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Lab-STICC

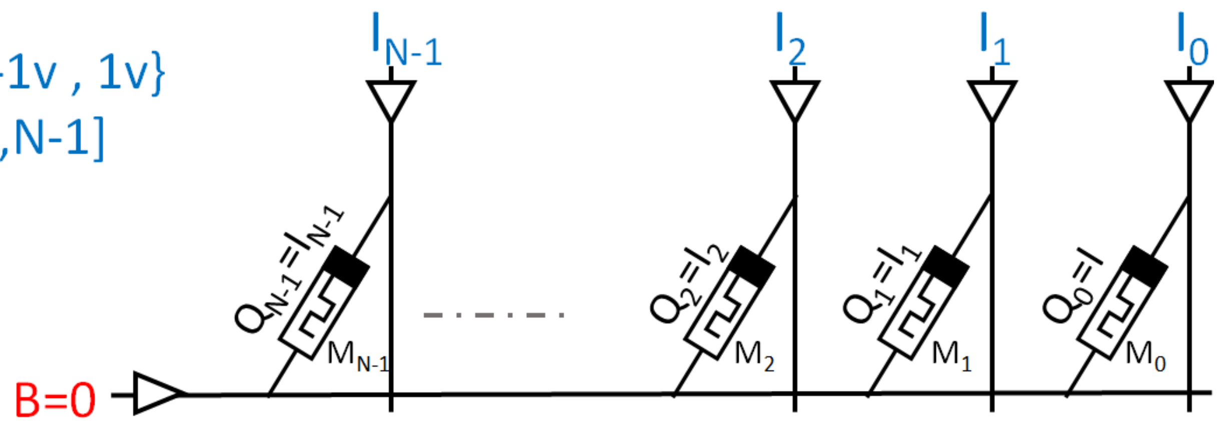
MOL logic design

- MOL on a vector of bits

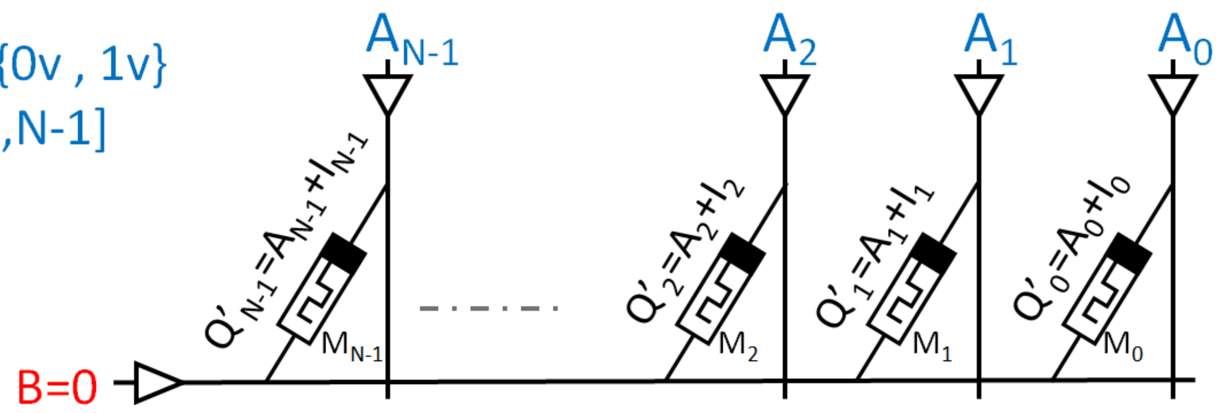
MOL-OR

- Write
- Overwrite (while selecting by 0)

$I_k \in \{-1v , 1v\}$
$k \in [0,N-1]$



$A_k \in \{0v , 1v\}$
$k \in [0,N-1]$

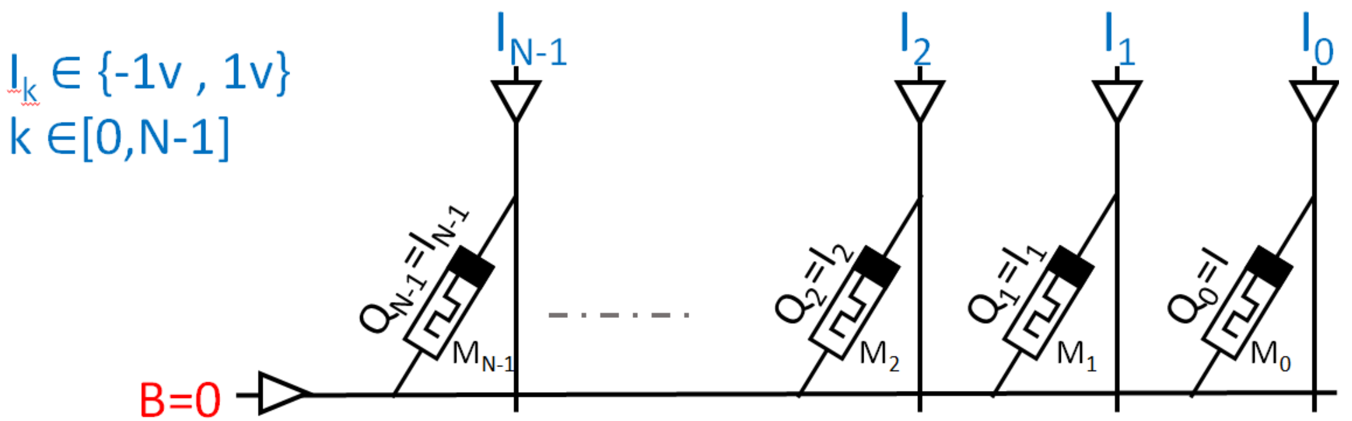


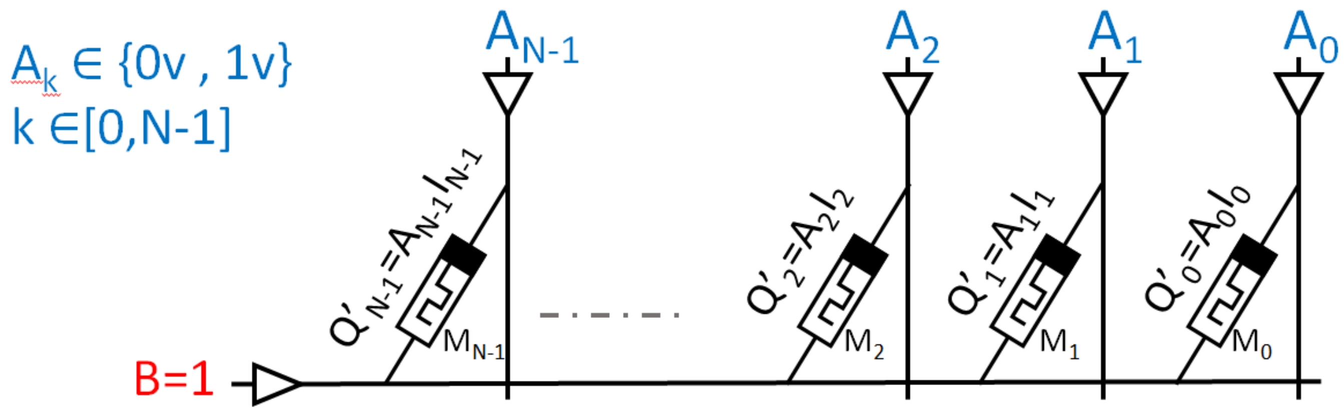
MOL logic design

- MOL on a vector of bits

MOL-AND

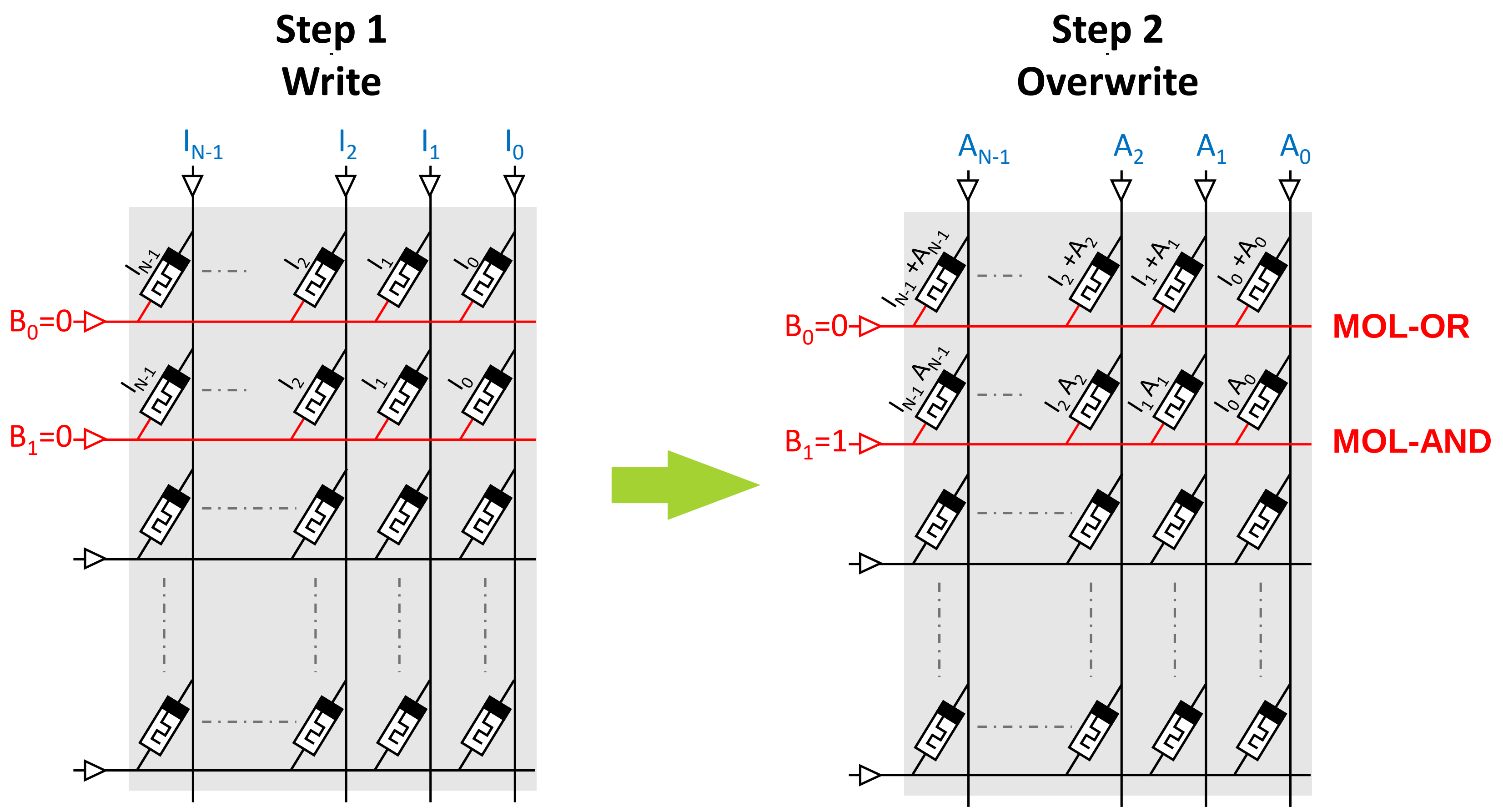
- Write
- Overwrite (while selecting by 1)





MOL logic design

- MOL in crossbar array
  - Write new data bits
  - Overwrite the already stored data bits



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Lab-STICC

ISCAS 2020

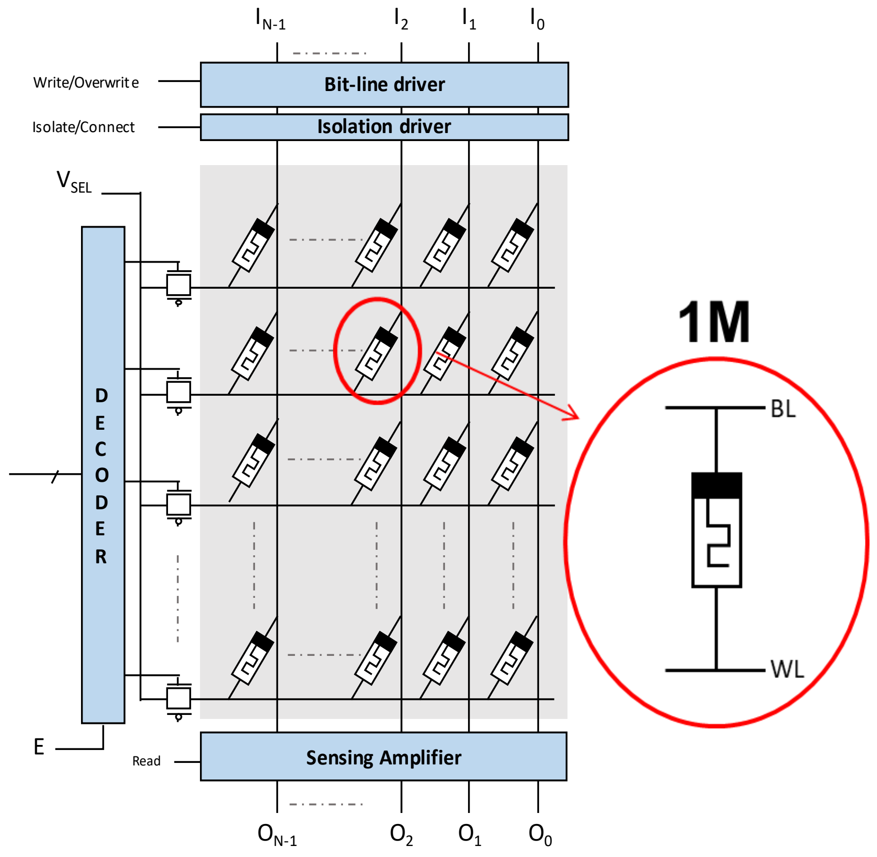
OUTLINE

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

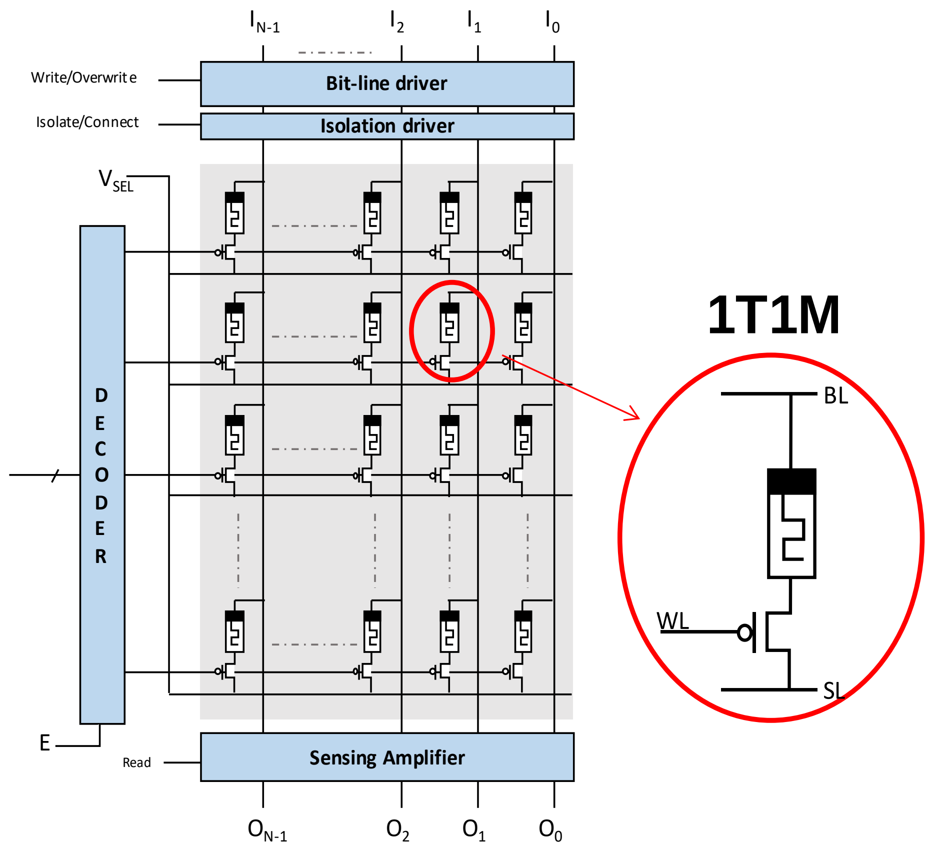
Proposed computational memory

MOL-based computational memory

- **1M** MOL architecture



- **1T1M** MOL architecture

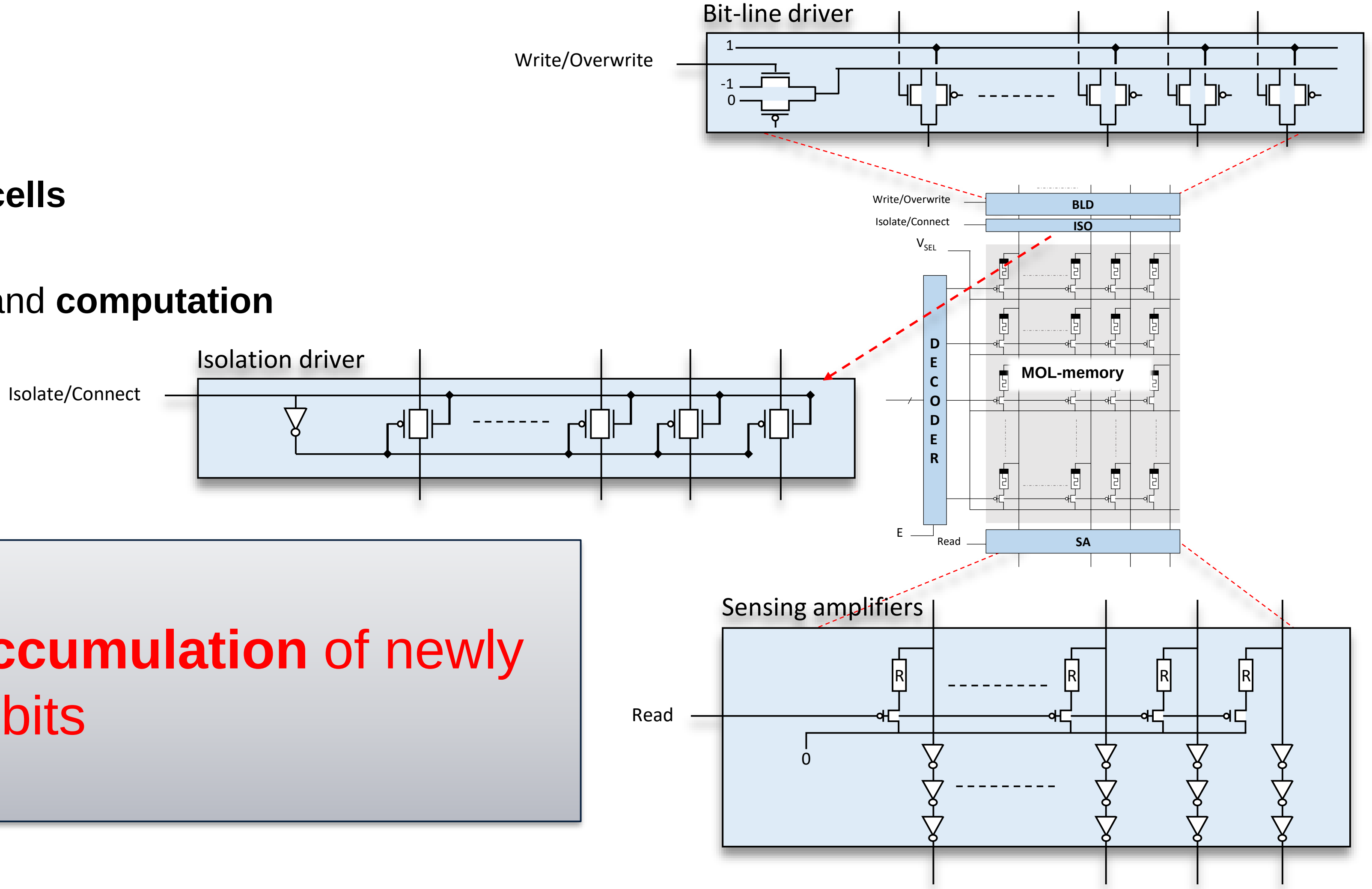


- Four modes:
  1. Write mode
  2. Overwrite mode
  3. Read mode
  4. Idle mode

Proposed computational memory

MOL-based computational memory

- Drivers
  - Efficiently shared to all **memristive cells**
  - Efficiently shared between **storage** and **computation**

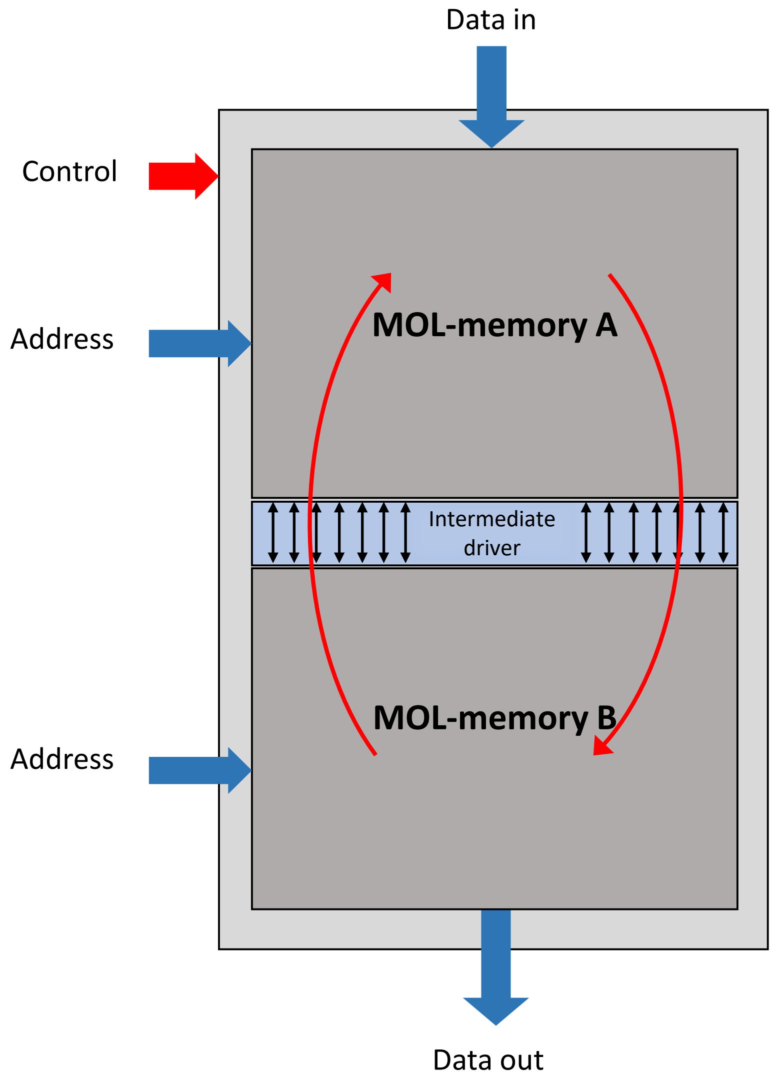


☹ **Supports only logic accumulation** of newly arriving bits

Proposed computational memory

MOL-based computational memory

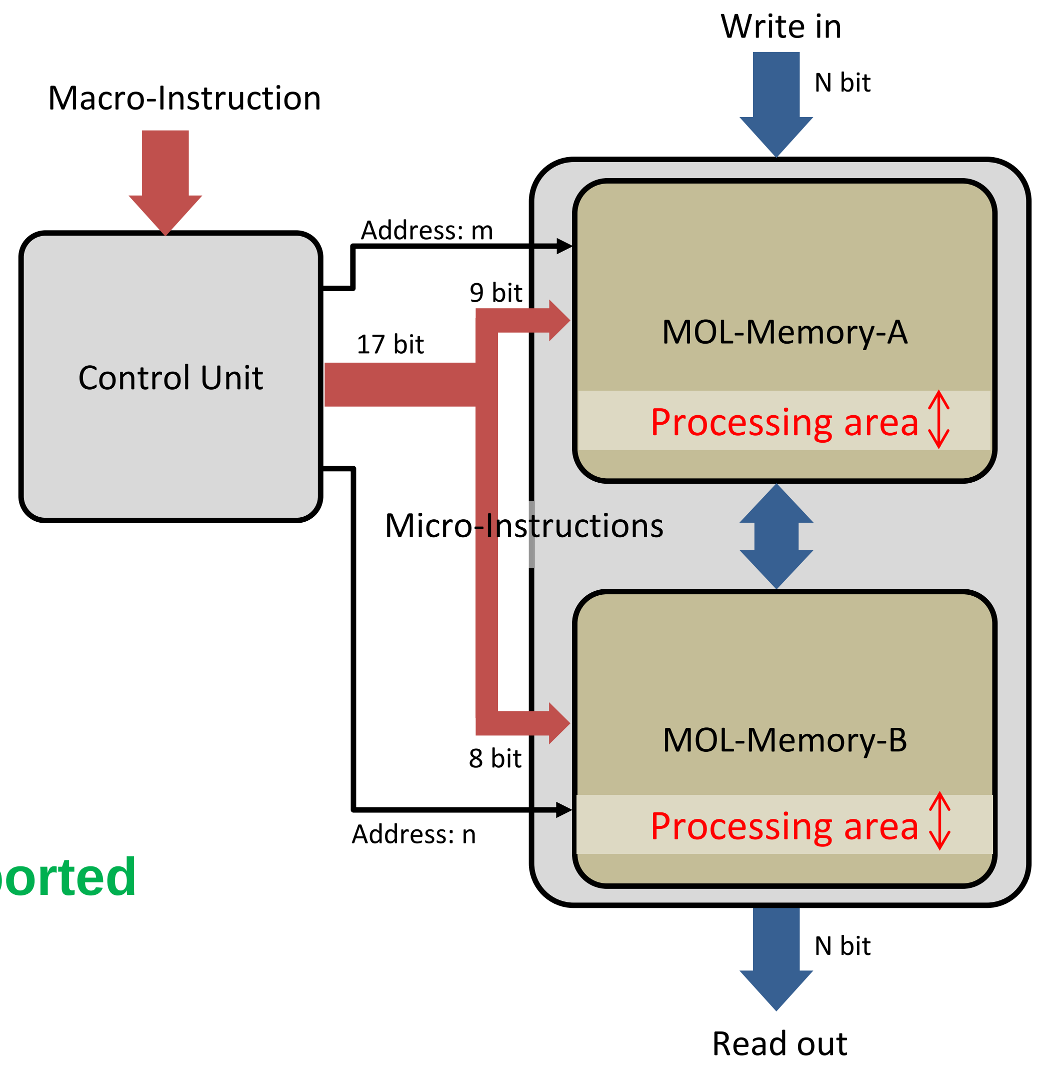
- Two coupled MOL-memories
  - Two interconnected MOL-memory blocks
  - Perform MOL **between any two wordlines**



Proposed computational memory

MOL-based computational memory

- Performing arithmetic tasks
  - At each time step, either **storage** or **computation**
  - An arithmetic task (**Macro-Instruction**) is broken into several MOL operations (**Micro-instructions**)
  - A **processing area** is reserved for computation
  - **Dynamically changed** to maintain uniform endurance of cells



More than 30 micro-instructions are supported

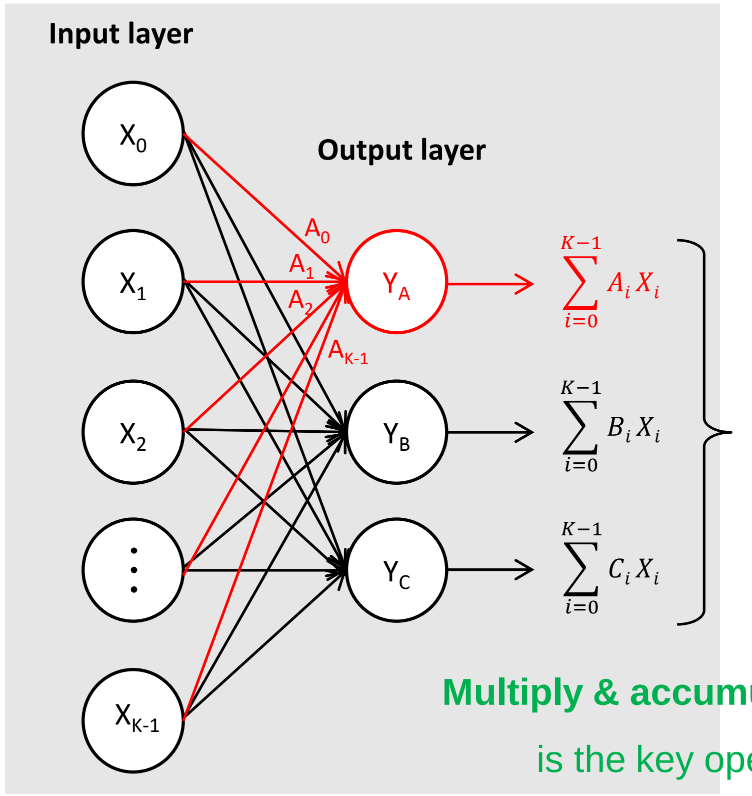
OUTLINE

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

MOL – based in-memory DNN

Convolution process

Convolution process in neural networks



Multiply & accumulate (MAC)

is the key operation

Input Data **Neuron Weights** **Output Equations**

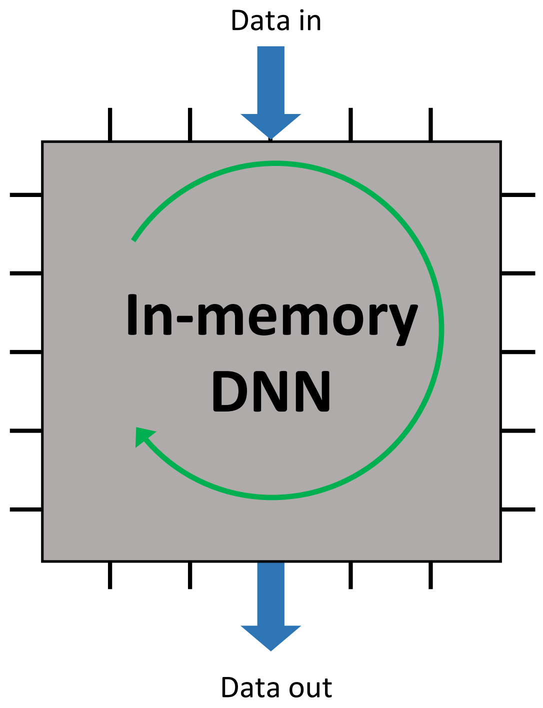
$$[X_0 \quad X_1 \ldots \quad X_{K-1}] * \begin{bmatrix} A_0 & B_0 & C_0 \\ A_1 & B_1 & C_1 \\ \cdots & \cdots & \cdots \\ A_{K-1} & B_{K-1} & C_{K-1} \end{bmatrix} = \begin{bmatrix} Y_A = X_0A_0 + X_1A_1 + \cdots X_{K-1}A_{K-1} \\ Y_B = X_0B_0 + X_1B_1 + \cdots X_{K-1}B_{K-1} \\ Y_C = X_0C_0 + X_1C_1 + \cdots X_{K-1}C_{K-1} \end{bmatrix}$$

- Number of MAC is proportional to the size of the network
- **For large DNN**, implies intensive data movement between memory and processing cores
- Inefficient in terms of **time** and **energy**

MOL – based in-memory DNN

Convolution in-memory

- Multiply and Accumulate (MAC) process



Multiplication

1010	Multiplicand
× 1011	Multiplier
1010	Partial product 1
1010	Partial product 2
0000	Partial product 3
1010	Partial product 4
1101110	**Product**

Significant number of computational steps (bottleneck)

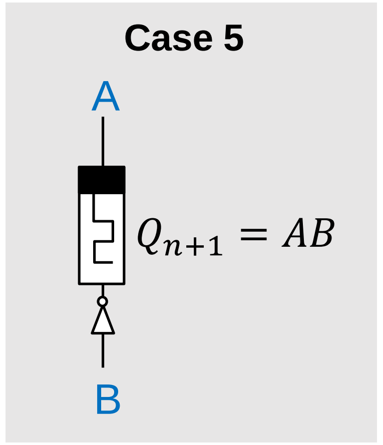
Need for an optimized **MAC** process inside memory

DNN in-memory

- MOL-based In-memory DNN computing

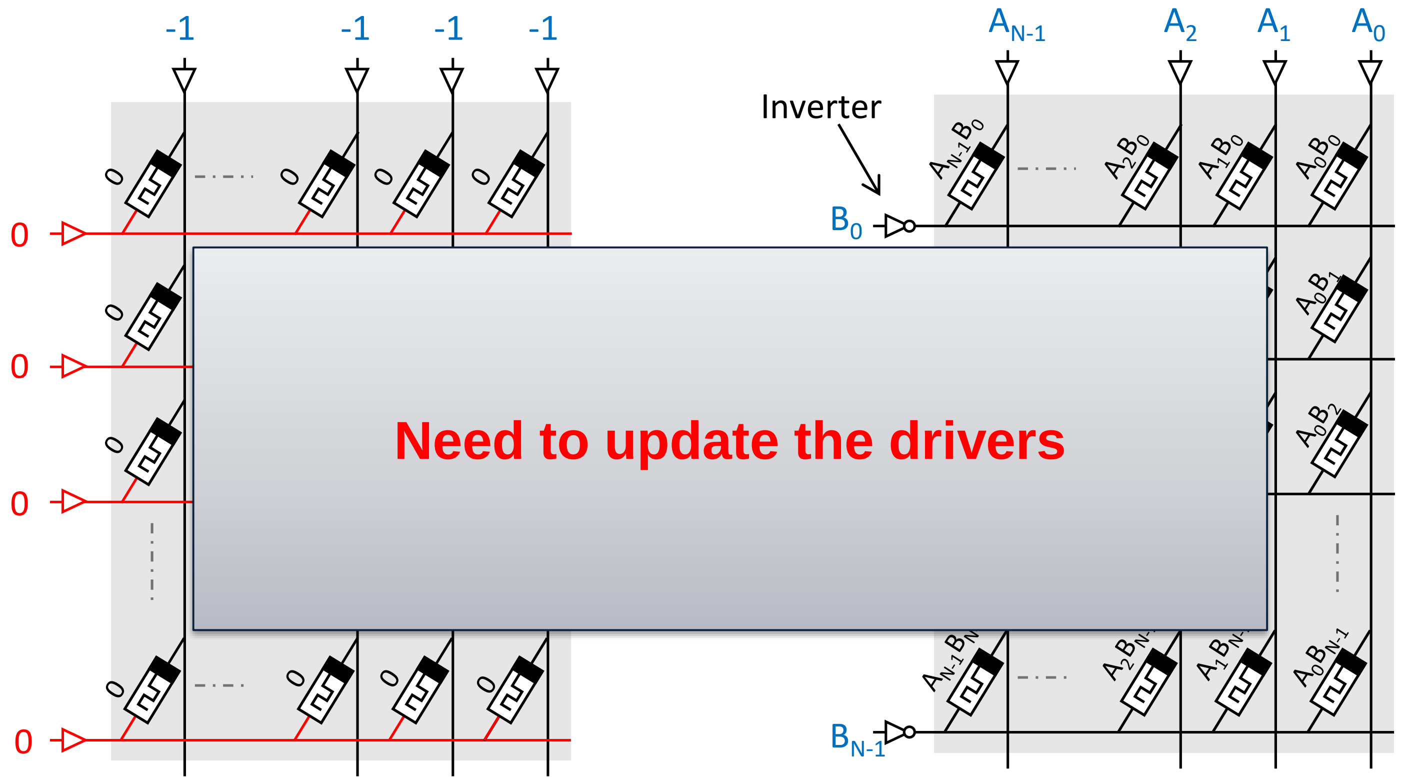
Realization of Partial product

- Initialize all cells to the logic zero
- Vector A and an inverted vector B are fed to the columns and rows



Step 1
Initialize by zeros

Step 2
Partial product array

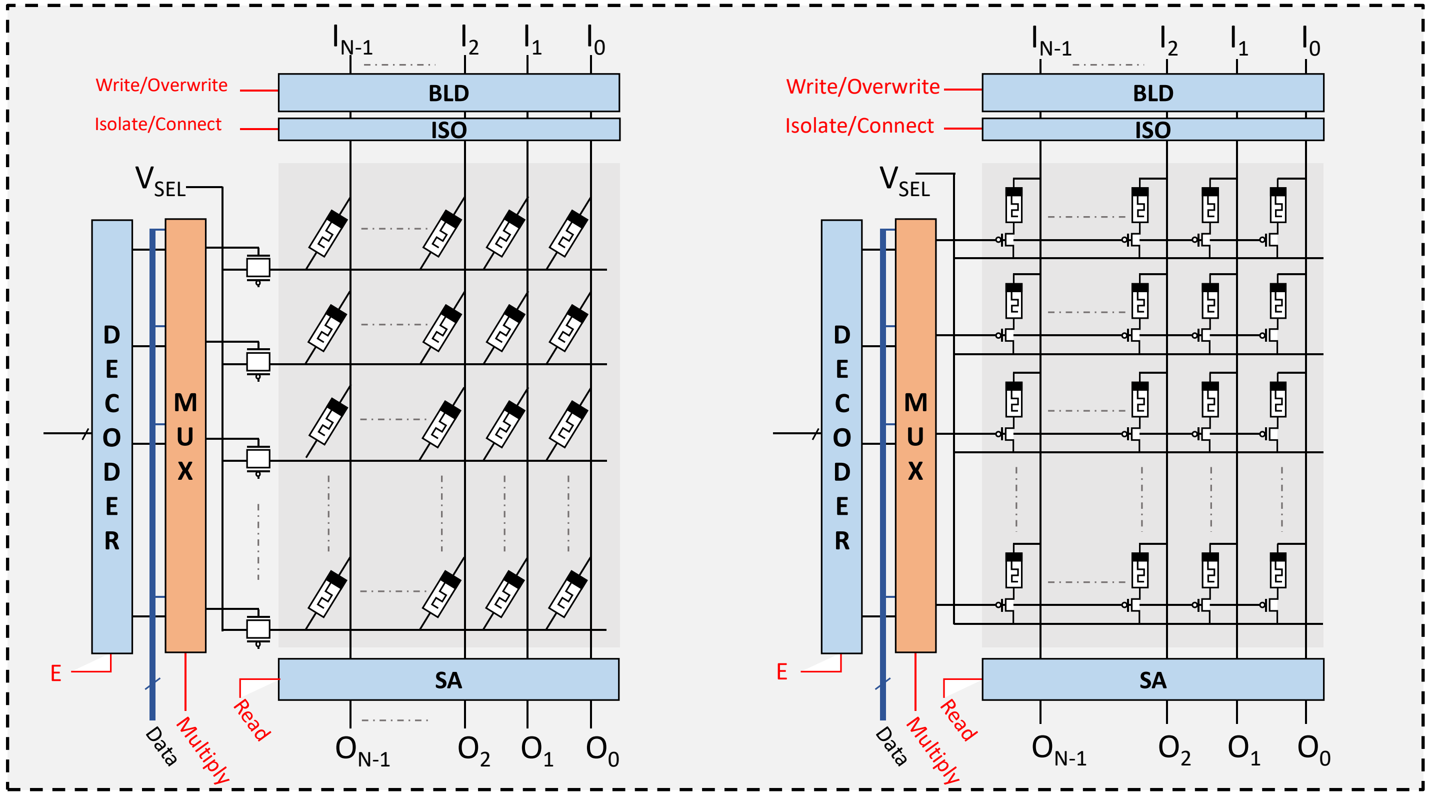


Need to update the drivers

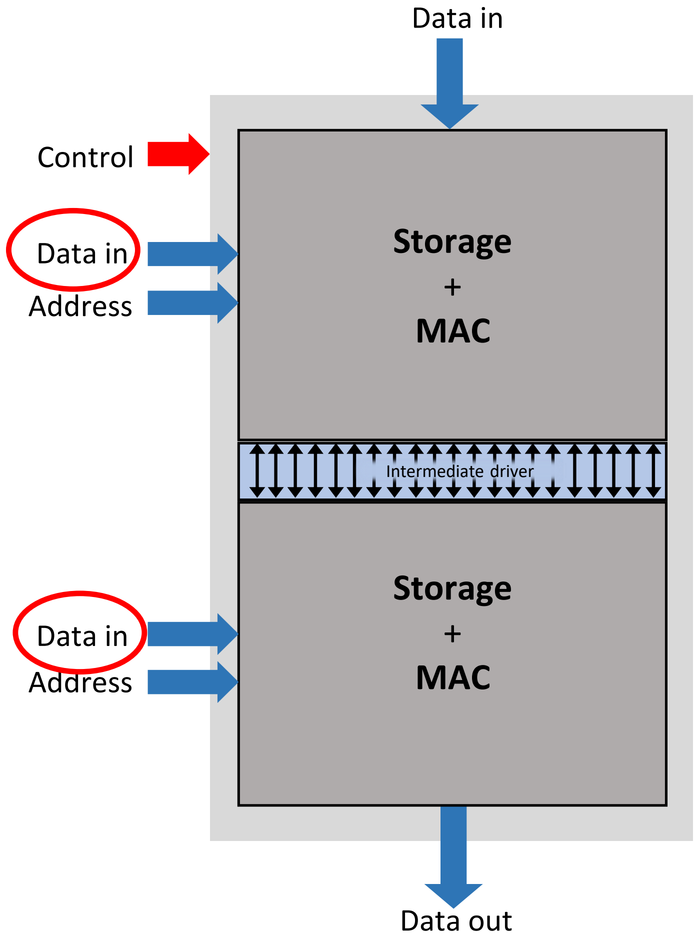
DNN in-memory

- MOL-based In-memory DNN computing

MOL-memory with **Updated drivers**



Computational memory (CMEM)



MOL – based in-memory DNN

DNN in-memory

- MOL-based In-memory DNN computing

Addition of Partial product

– $M \times N$ multiplication requires $(M-1)(6N+1)$ cycles

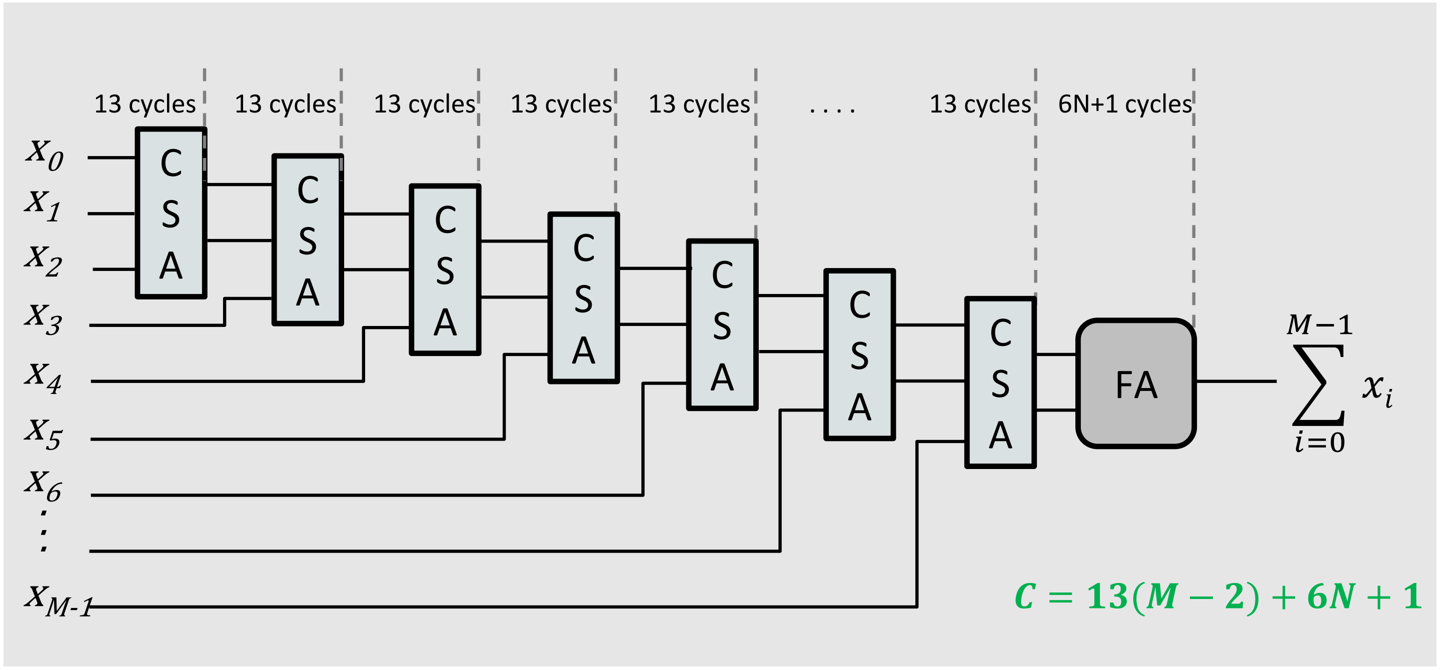
Use the method of carry save adder (CSA):

– Provides a 3:2 operands reduction

– Fixed latency

– Number of cycles reduced to $C = 13(M-2) + 6N + 1$

CSA in a tree-like structure



MOL – based in-memory DNN

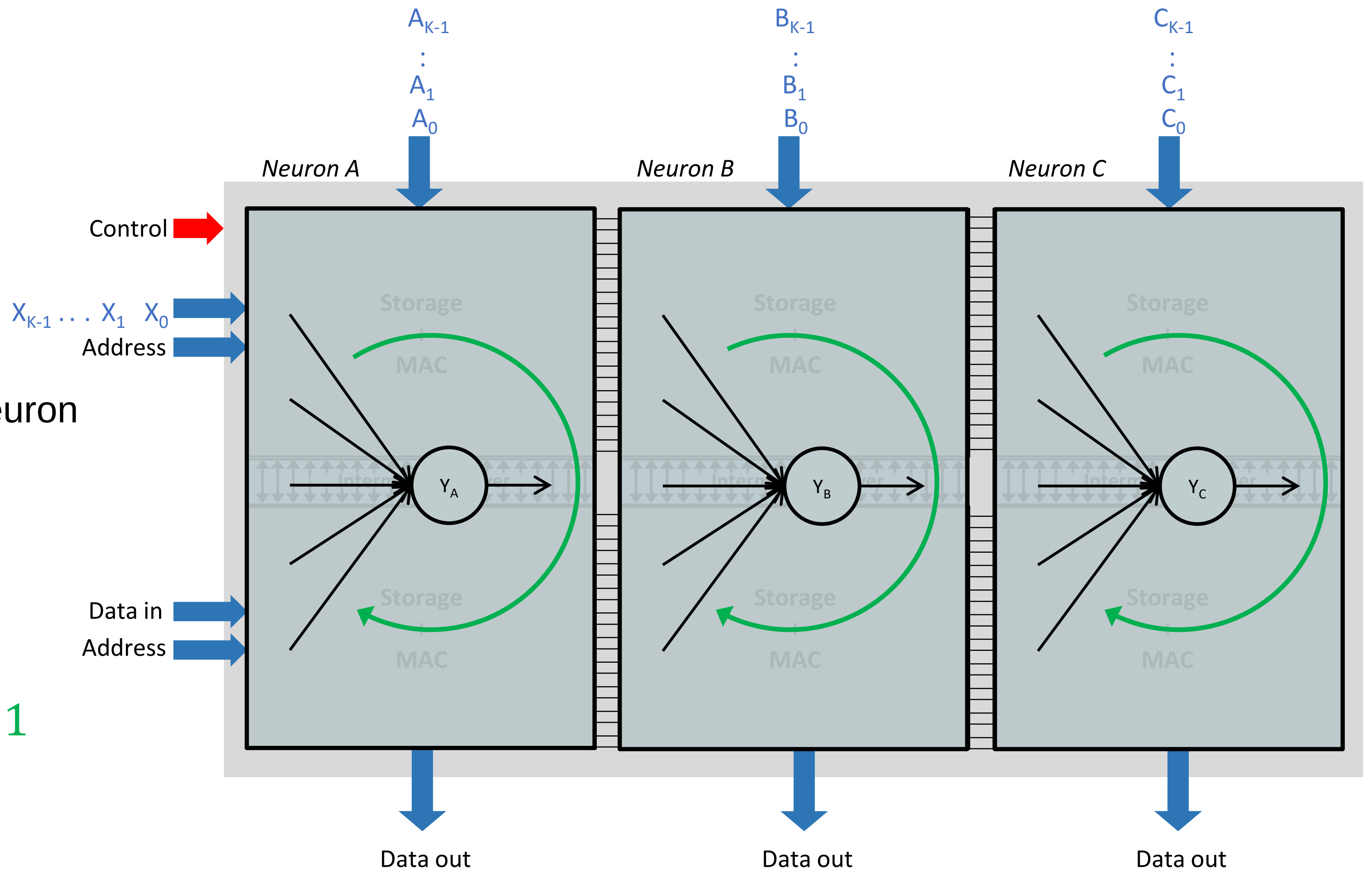
DNN in-memory

- MOL-based In-memory DNN computing

CMEM-based DNN architecture

- Interconnected CMEM blocks
- Each CMEM compute the output of a single neuron
- Parallel execution is possible

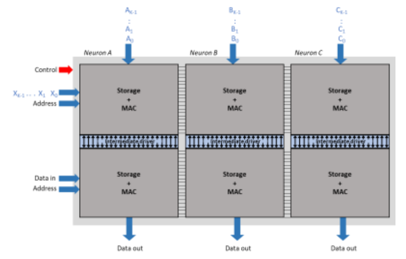
$$Total\ latency = (C + 15)K + 6(N + M) + 1$$



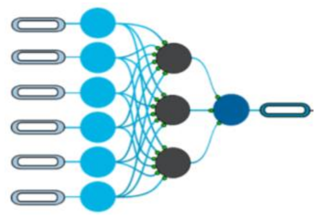
OUTLINE

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

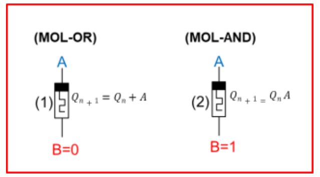
Conclusion



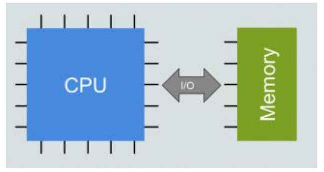
☑ Novel architecture design for **in-memory DNN** applications



☑ Programmable and allows to execute any sequence of arithmetic tasks including **MAC**



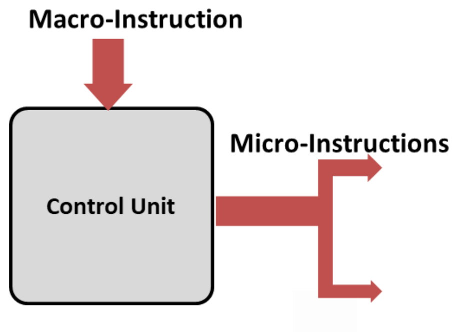
☑ Computation is performed based on our proposed **MOL design style**



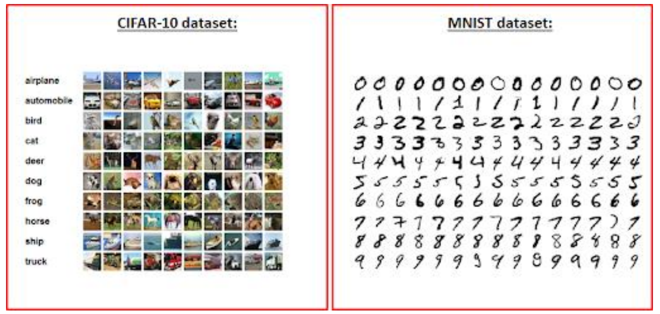
☑ Addresses the inefficiency of moving data between memory and processing cores which is **time** and **energy** consuming.

Future works

- **Control unit** and the associated micro-operations synthesis tool



- Perform DNN on **real dataset** such as CIFAR-10 or MNIST



- Performance evaluation and **comparison** with state-of the art CPU and GPU implementations.

Thank you for your attention