Computing in/using memory -- II

Asif Ali Khan

Fall Semester 2024

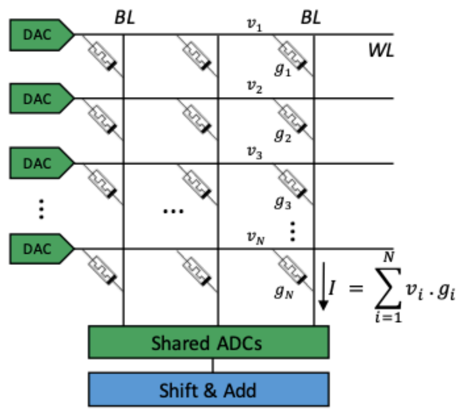
Department of Computer Systems Engineering

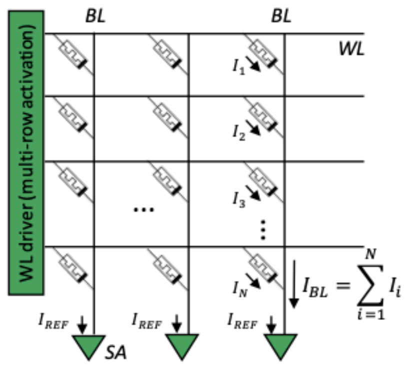
UET Peshawar, Pakistan

Nov 28, 2024

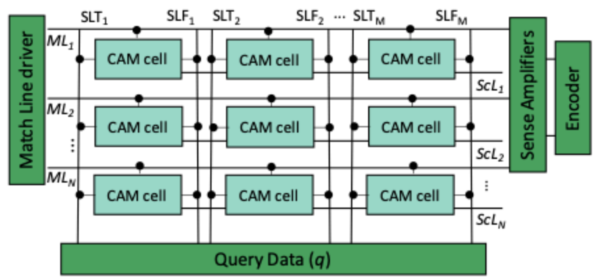
Recap: Compute-in-memory (CIM)

- ❑ The CIM paradigm aims to completely eliminate the data movement
- ❑ The fundamental idea is to exploit the physical properties of the memory devices to perform computations
- ❑ Not every computation can be performed with every technology

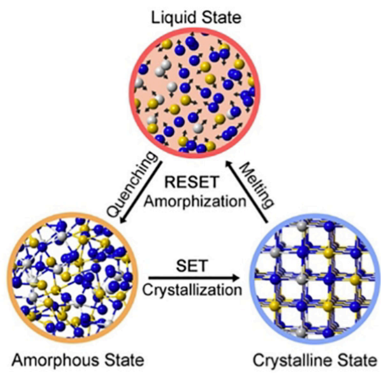




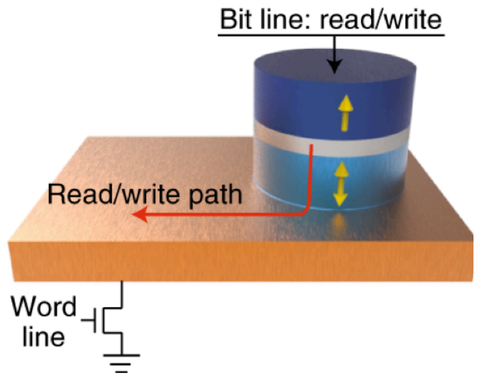
WL driver (multi-row activation)
BL
BL
WL
I_1
I_2
I_3
I_N
...
$I_{BL} = \sum_{i=1}^{N} I_i$
I_{REF}
SA



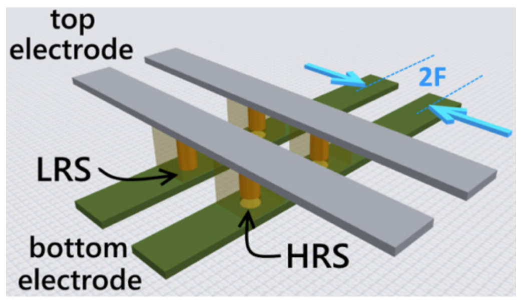
Recap: Emerging nonvolatile memories (NVMs)



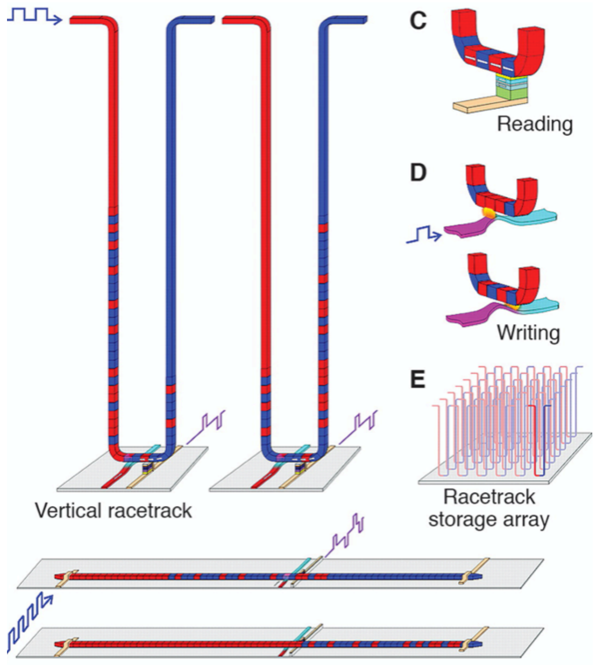
Zhang et al., 2020



G. Yu, 2020



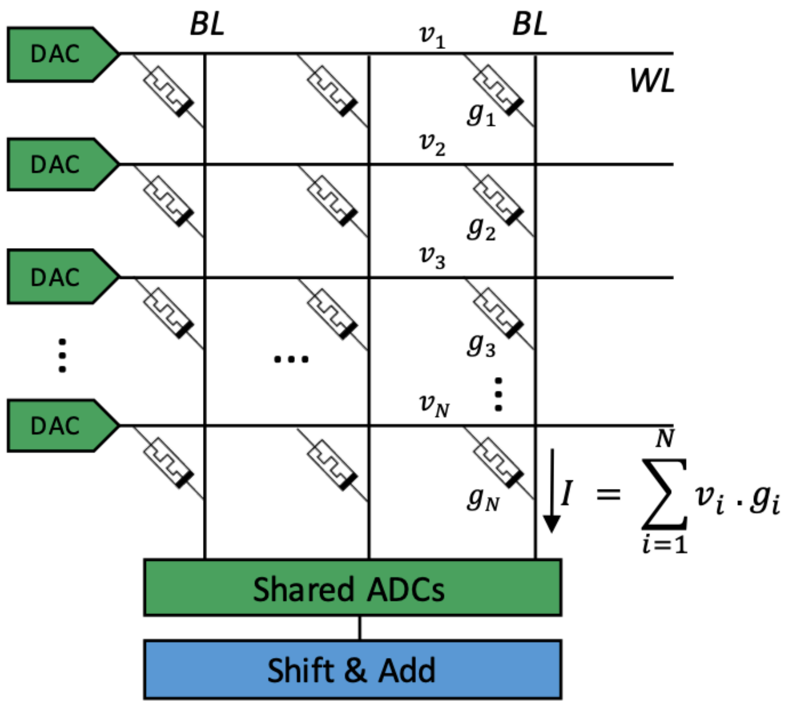
Lim et al, 2015



Parkin et al, 2008

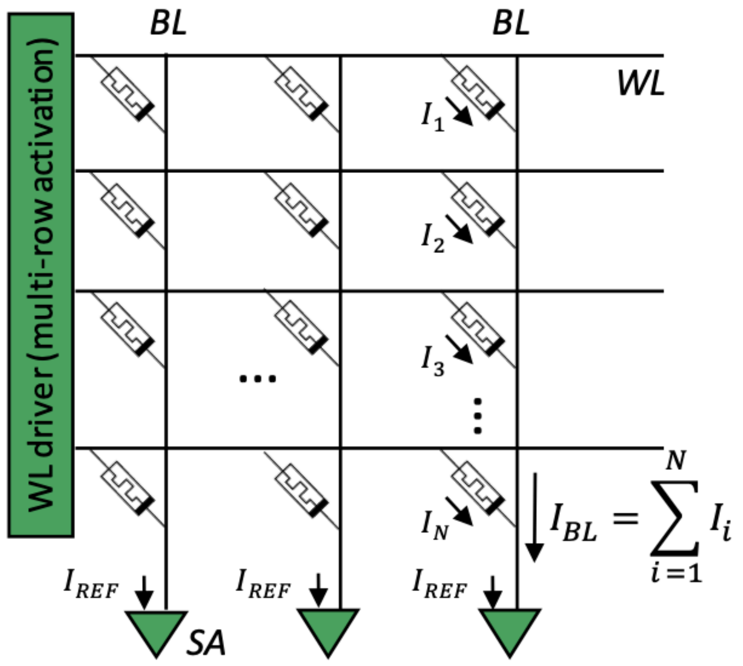
- ❑ Other options: FeFET, FeRAM etc
- ❑ Each technology has its strengths and challenges
- ❑ PCM and MRAM receive a lot of traction in industry

Recap: CIM crossbar



- ❑ Program one operand into memristors devices (conductance)
- ❑ Enable all wordlines simultaneously and apply another operand as input
- ❑ The accumulated current at the bitlines using Kirchoff's law produces the outcome of dot product
- ❑ Analog domain computation – results are approximate

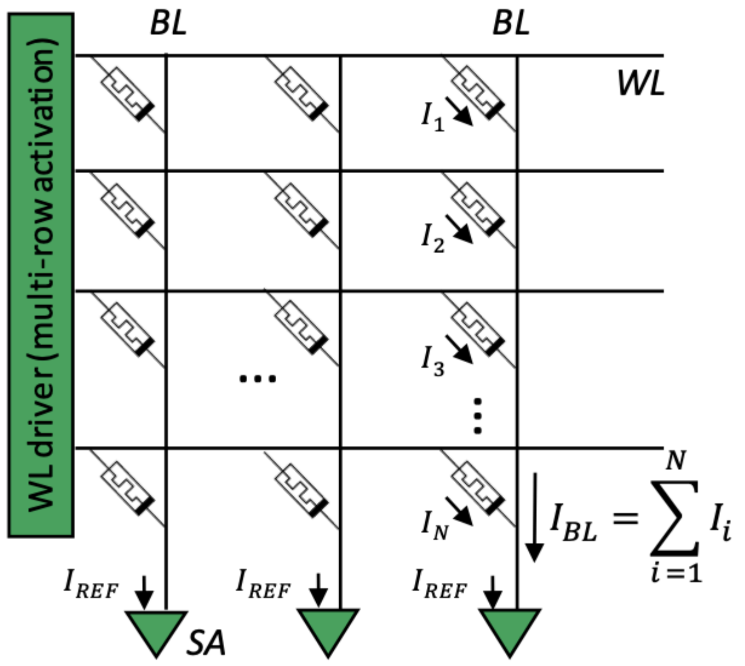
CIM logic



WL driver (multi-row activation)
BL
BL
WL
I_1
I_2
I_3
I_N
$I_{BL} = \sum_{i=1}^{N} I_i$
I_{REF}
I_{REF}
I_{REF}
SA

- Performs bulk-bitwise logic operations in-memory (operands and output are stored in the same array)

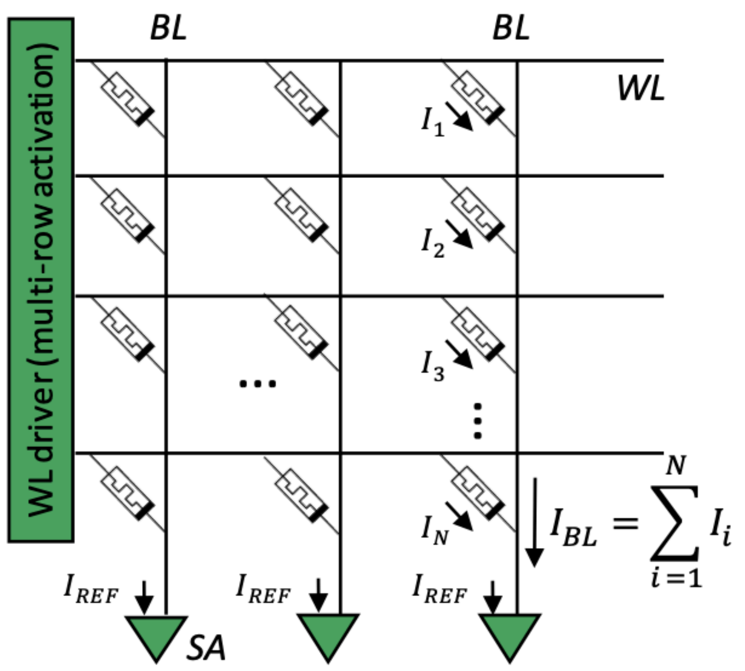
CIM logic



WL driver (multi-row activation)
BL
BL
WL
I_1
I_2
I_3
I_N
$I_{BL} = \sum_{i=1}^{N} I_i$
I_{REF}
I_{REF}
I_{REF}
SA

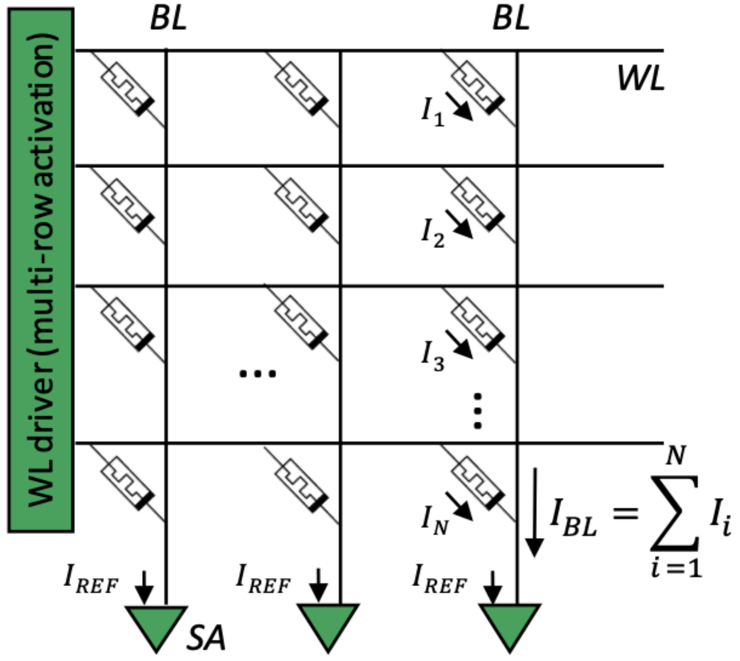
- Performs bulk-bitwise logic operations in-memory (operands and output are stored in the same array)
- Have applications in cryptography, databases and other domains

CIM logic


WL driver (multi-row activation)
BL
BL
WL
I_1
I_2
I_3
I_N
$I_{BL} = \sum_{i=1}^{N} I_i$
I_{REF}
I_{REF}
I_{REF}
SA

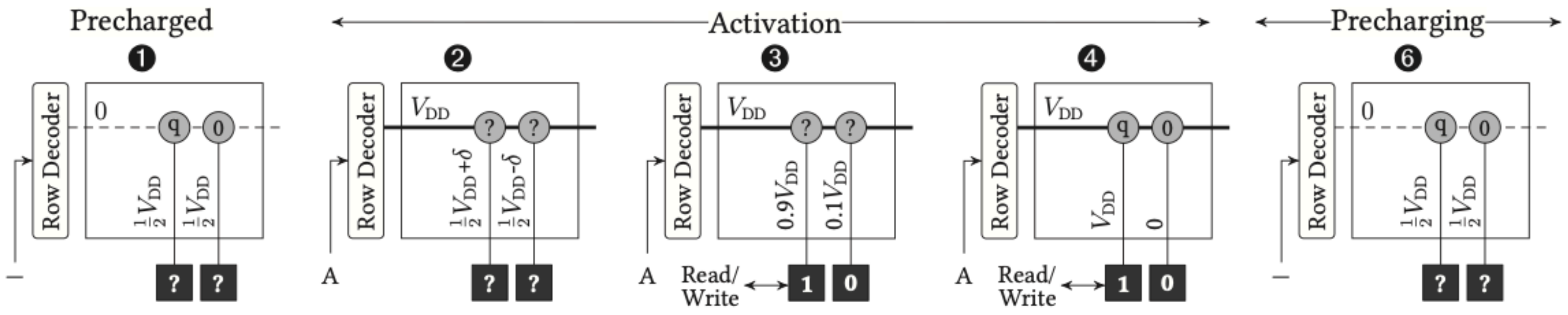
- Performs bulk-bitwise logic operations in-memory (operands and output are stored in the same array)
- Have applications in cryptography, databases and other domains
- The fundamental idea is still current/charge sharing

CIM logic


WL driver (multi-row activation)
BL
BL
WL
I_1
I_2
I_3
I_N
$I_{BL} = \sum_{i=1}^{N} I_i$
I_{REF}
I_{REF}
I_{REF}
SA

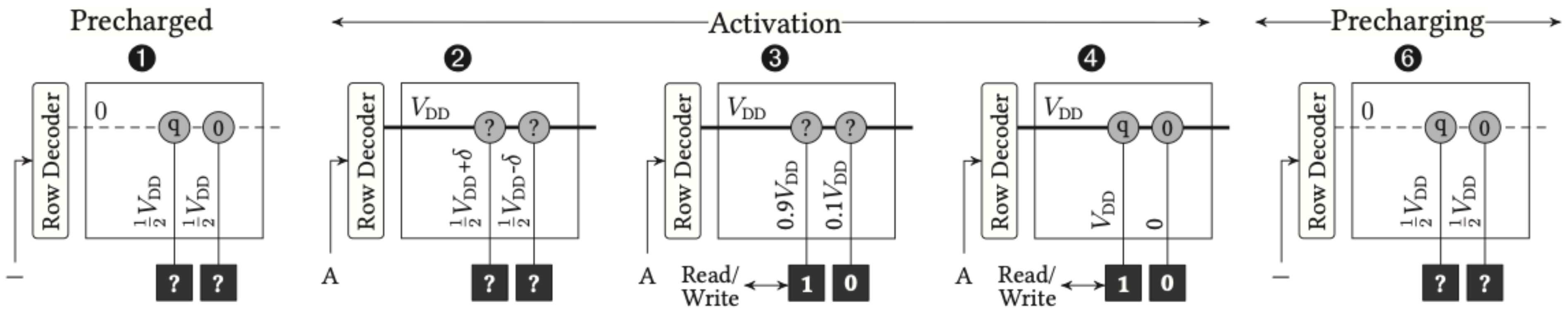
- Performs bulk-bitwise logic operations in-memory (operands and output are stored in the same array)
- Have applications in cryptography, databases and other domains
- The fundamental idea is still current/charge sharing
- The exact implementation depends on the specific technology

CIM-logic using DRAM



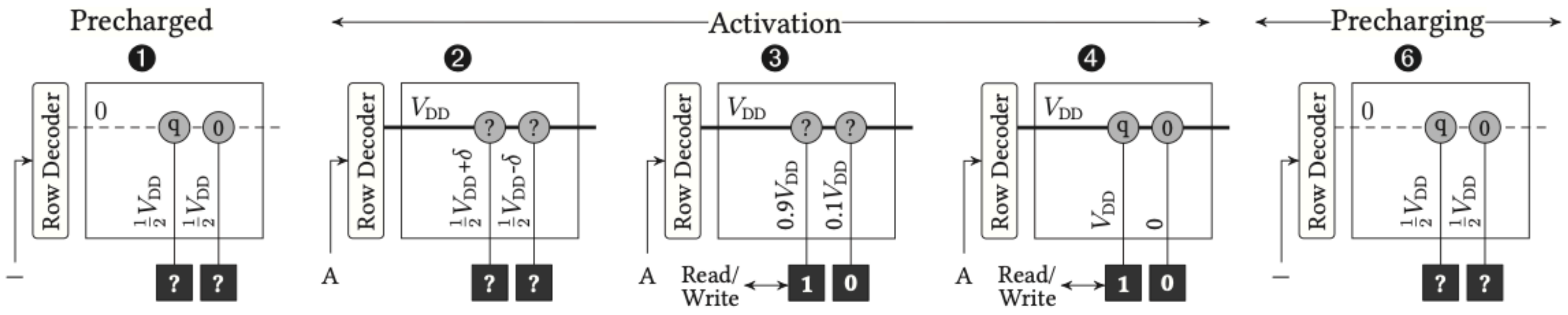
Sishadri et al., MICRO 2013

CIM-logic using DRAM



- ❑ Before discussing CIM-DRAM, lets first look at RowClone

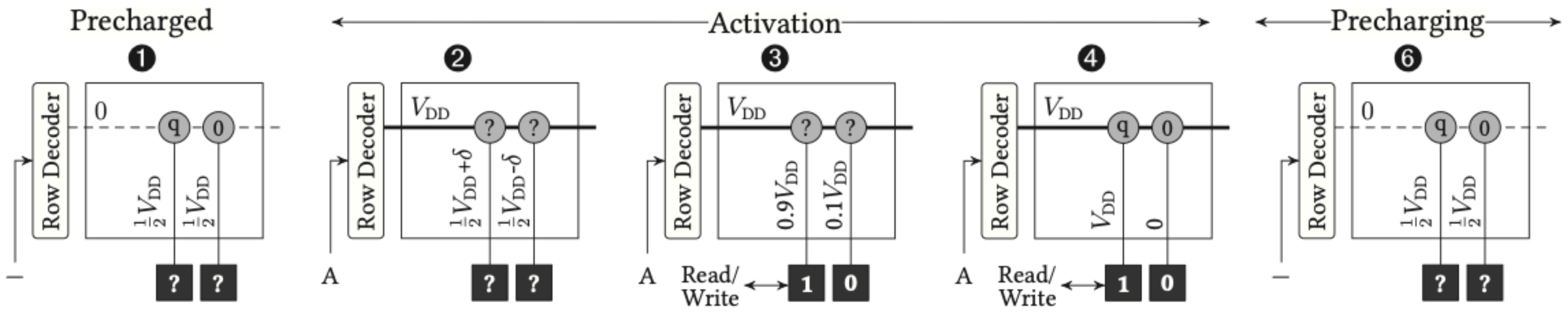
CIM-logic using DRAM



- Before discussing CIM-DRAM, lets first look at RowClone

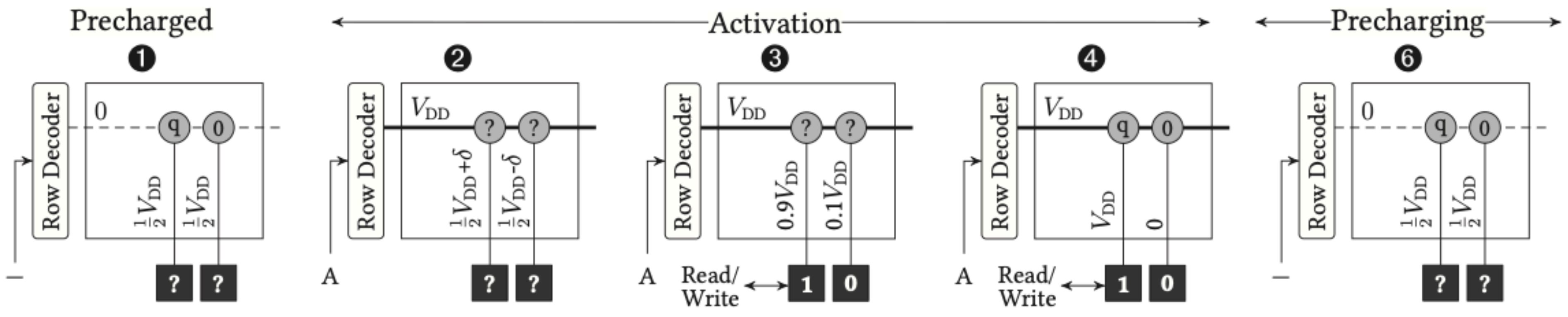
If a DRAM cell is connected to a bitline that is at a stable state (either V_{DD} or 0) instead of the equilibrium state ($\frac{1}{2}V_{DD}$), then the data in the cell is overwritten with the data (voltage level) on the bitline.

CIM-logic using DRAM



- To RowClone src to dst, ACTIVATE-ACTIVATE-PRECHARGE (AAP) is needed:

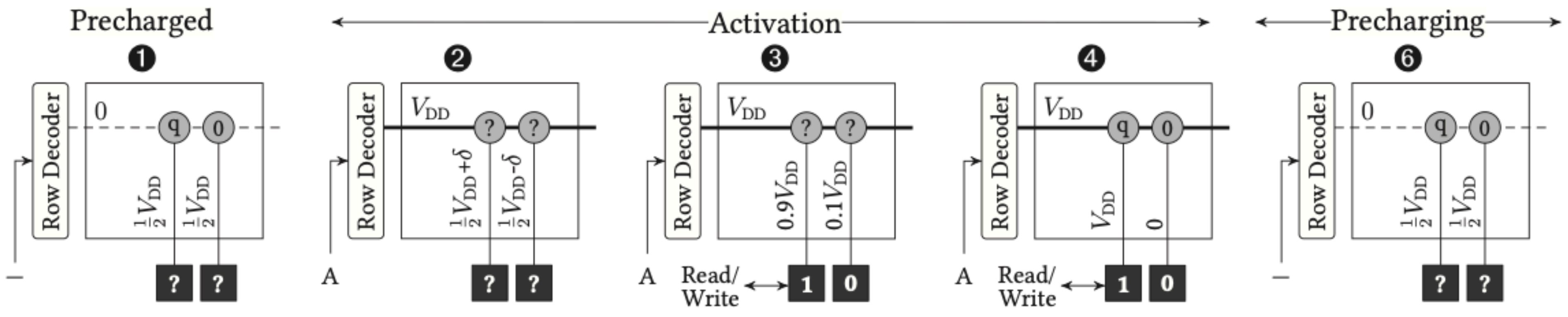
CIM-logic using DRAM



- To RowClone src to dst, ACTIVATE-ACTIVATE-PRECHARGE (AAP) is needed:
  - ACTIVATE src, the data appears on the bitlines (senseamps)

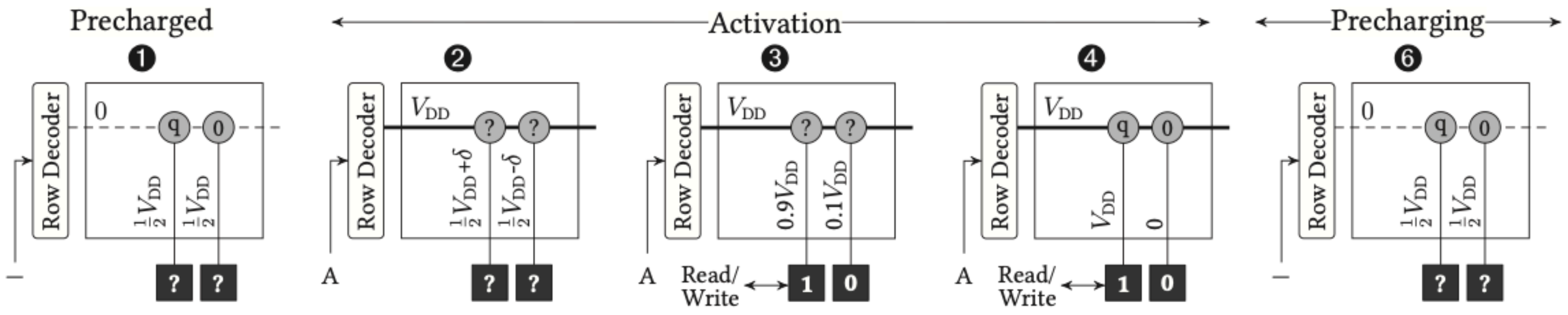
Sishadri et al., MICRO 2013

CIM-logic using DRAM



- To RowClone src to dst, ACTIVATE-ACTIVATE-PRECHARGE (AAP) is needed:
  - ACTIVATE src, the data appears on the bitlines (senseamps)
  - ACTIVATE dst, the stable bitline contents replaces contents of the dst row

CIM-logic using DRAM



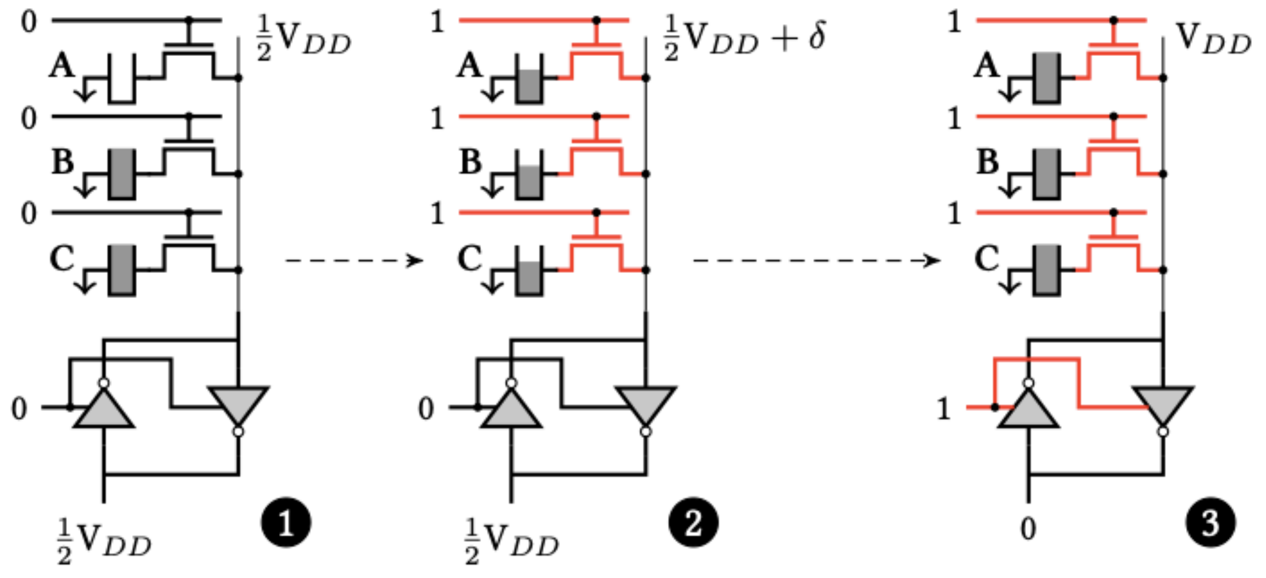
- To RowClone src to dst, ACTIVATE-ACTIVATE-PRECHARGE (AAP) is needed:
  - ACTIVATE src, the data appears on the bitlines (senseamps)
  - ACTIVATE dst, the stable bitline contents replaces contents of the dst row
  - A PRECHARGE command brings the bitline to the equilibrium state (VDD/2)

CIM-logic using DRAM

- Simultaneous activation of three rows in a DRAM array results in bit-wise majority
  - At least two cells have to be 1, for the output to be 1
  - The operation is called *triple row activation* (TRA)

CIM-logic using DRAM

- Simultaneous activation of three rows in a DRAM array results in bit-wise majority
  - At least two cells have to be one, for the output to be one
  - The operation is called *triple row activation* (TRA)



Seshadri et al., MICRO 2017

CIM-logic using DRAM

- Lets say `A, B, C` represent the state of the three cells
- The final state of the bitline is: `AB + AC + BC`

CIM-logic using DRAM

- Lets say `A, B, C` represent the state of the three cells
- The final state of the bitline is: `AB + AC + BC`
- This can be rewritten as: `C.(A+B)+C'.(AB)`

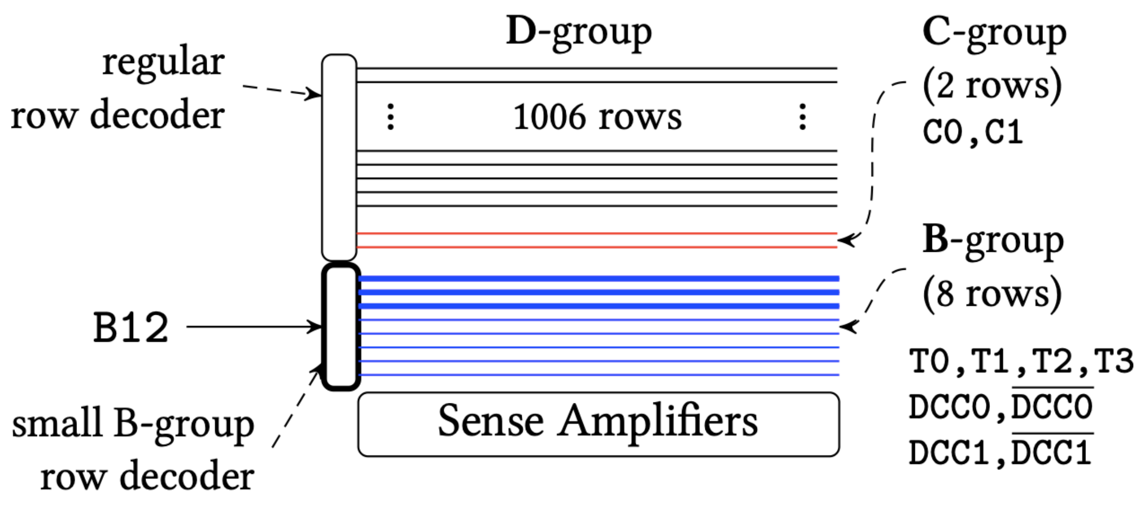
CIM-logic using DRAM

- Lets say `A, B, C` represent the state of the three cells
- The final state of the bitline is: `AB + AC + BC`
- This can be rewritten as: `C.(A+B)+C'.(AB)`
- By controlling C, we can implement both AND and OR operations
  - C=1, to implement OR and C=0, to implement AND operation

CIM-logic using DRAM

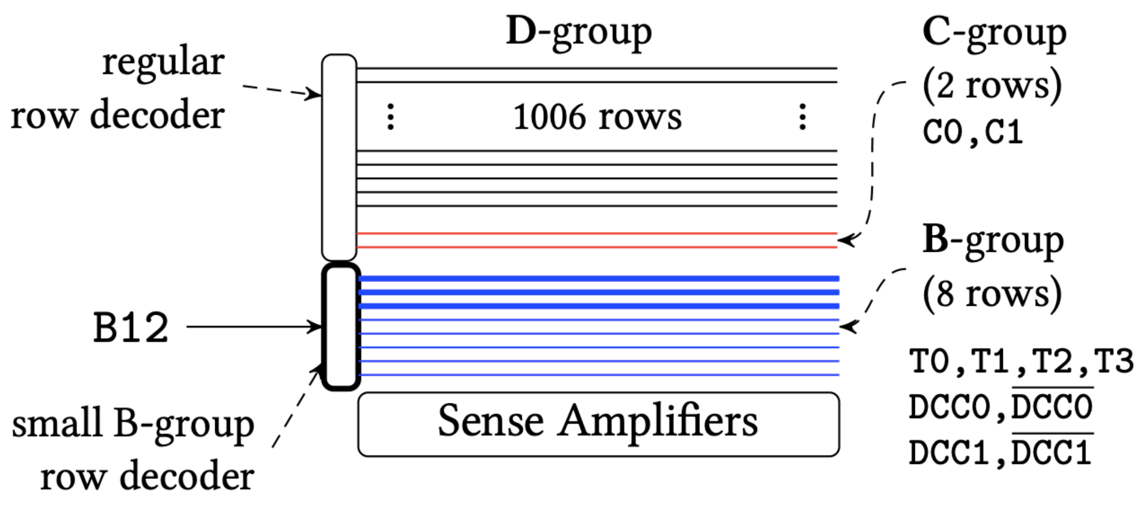
- Lets say `A, B, C` represent the state of the three cells
- The final state of the bitline is: `AB + AC + BC`
- This can be rewritten as: `C.(A+B)+C'.(AB)`
- By controlling C, we can implement both AND and OR operations
  - `C=1`, to implement OR and `C=0`, to implement AND operation
- Important: TRA destroys contents of the involved cells
  - Contents need to be copied to a different place first, if important/needed

Ambit -- CIM-logic using DRAM



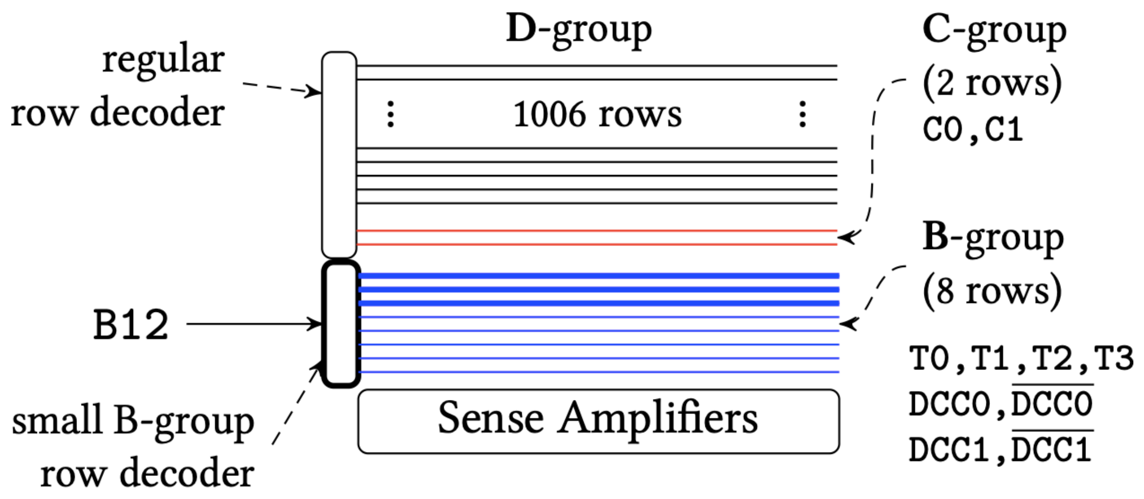
- Dedicated compute rows (B-group) to simplify the row-decoder design

Ambit -- CIM-logic using DRAM



- ❑ Dedicated compute rows (B-group) to simplify the row-decoder design
- ❑ The two control/constants rows (C-group) have constant 0 (C0) and 1 (C1)

Ambit -- CIM-logic using DRAM



- ❑ Dedicated compute rows (B-group) to simplify the row-decoder design
- ❑ The two control/constants rows (C-group) have constant 0 (C0) and 1 (C1)
- ❑ Majority of the rows are data rows (D-group), as in a typical DRAM array

Ambit -- CIM-logic using DRAM

- To perform a logic AND using Ambit:
  1. Copy A to T0
  2. Copy B to T1
  3. Copy C0 to T2
  4. Activate T0, T1, and T2 simultaneously
  5. Copy T0 to R (resultant row)

Ambit -- CIM-logic using DRAM

- To perform a logic AND using Ambit:
  1. Copy A to T0
  2. Copy B to T1
  3. Copy C0 to T2
  4. Activate T0, T1, and T2 simultaneously
  5. Copy T0 to R (resultant row)
- For logical OR, copy C1 in step 3 instead of C0

Ambit -- CIM-logic using DRAM

- To perform a logic AND using Ambit:
  1. Copy A to T0
  2. Copy B to T1
  3. Copy C0 to T2
  4. Activate T0, T1, and T2 simultaneously
  5. Copy T0 to R (resultant row)

- For logical OR, copy C1 in step 3 instead of C0

- Now that you know all the needed steps, how would you perform Di & Dj, using AAP commands ?

Seshadri et al., MICRO 2017

Ambit -- CIM-logic using DRAM

- To perform a logic AND using Ambit:
  1. Copy A to T0
  2. Copy B to T1
  3. Copy C0 to T2
  4. Activate T0, T1, and T2 simultaneously
  5. Copy T0 to R (resultant row)

```
AAP (Di,  B0) ;T0 = Di
AAP (Dj,  B1) ;T1 = Dj
AAP (C0,  B2) ;T2 = 0
AAP (B12, Dk) ;Dk = T0 & T1
```

- For logical OR, copy C1 in step 3 instead of C0

- Now that you know all the needed steps, how would you perform Di & Dj, using AAP commands ?

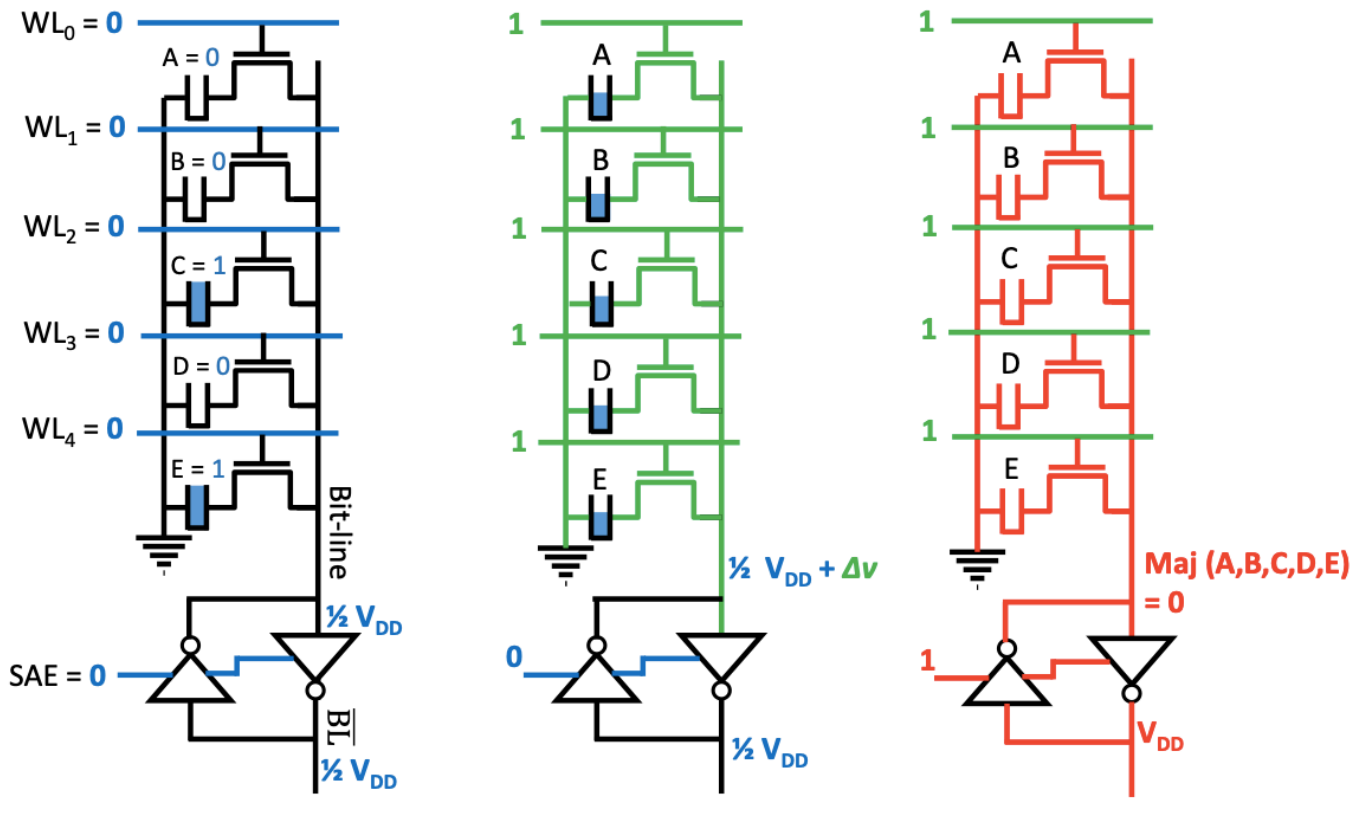
Ambit -- CIM-logic using DRAM

- To perform a logic AND using Ambit:
  1. Copy A to T0
  2. Copy B to T1
  3. Copy C0 to T2
  4. Activate T0, T1, and T2 simultaneously
  5. Copy T0 to R (resultant row)

```
AAP (Di,  B0) ;T0 = Di
AAP (Dj,  B1) ;T1 = Dj
AAP (C0,  B2) ;T2 = 0
AAP (B12, Dk) ;Dk = T0 & T1
```

- For logical OR, copy C1 in step 3 instead of C0
- Now that you know all the needed steps, how would you perform Di & Dj, using AAP commands ?
- The NOT operation is implemented using inverted bitline, and specialized DRAM cells

In-DRAM addition

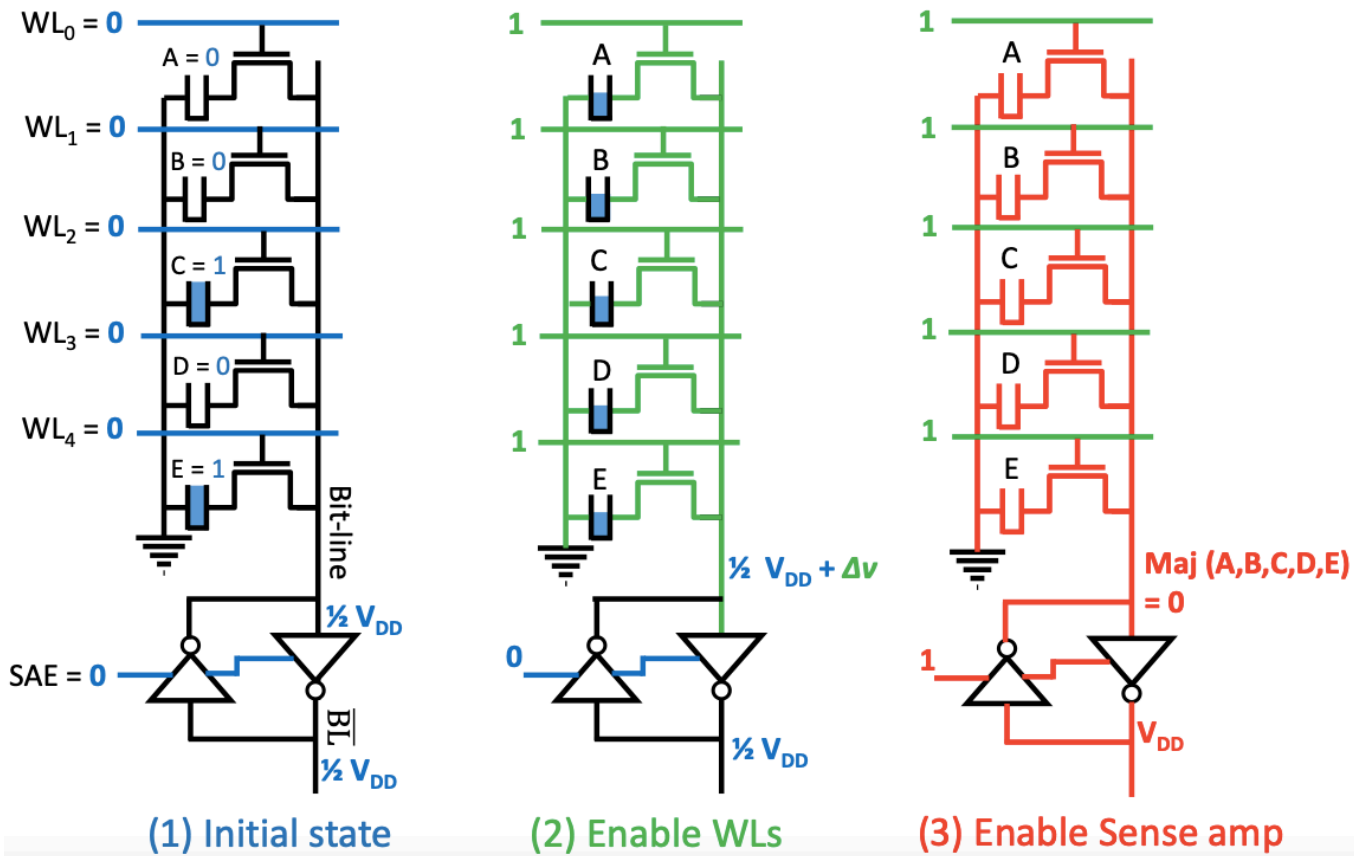


(1) Initial state (2) Enable WLs (3) Enable Sense amp

In-DRAM addition

$$Cout = Majority(A, B, Cin)$$

$$Sum = Majority(A, B, Cin, \overline{Cout}, \overline{Cout})$$

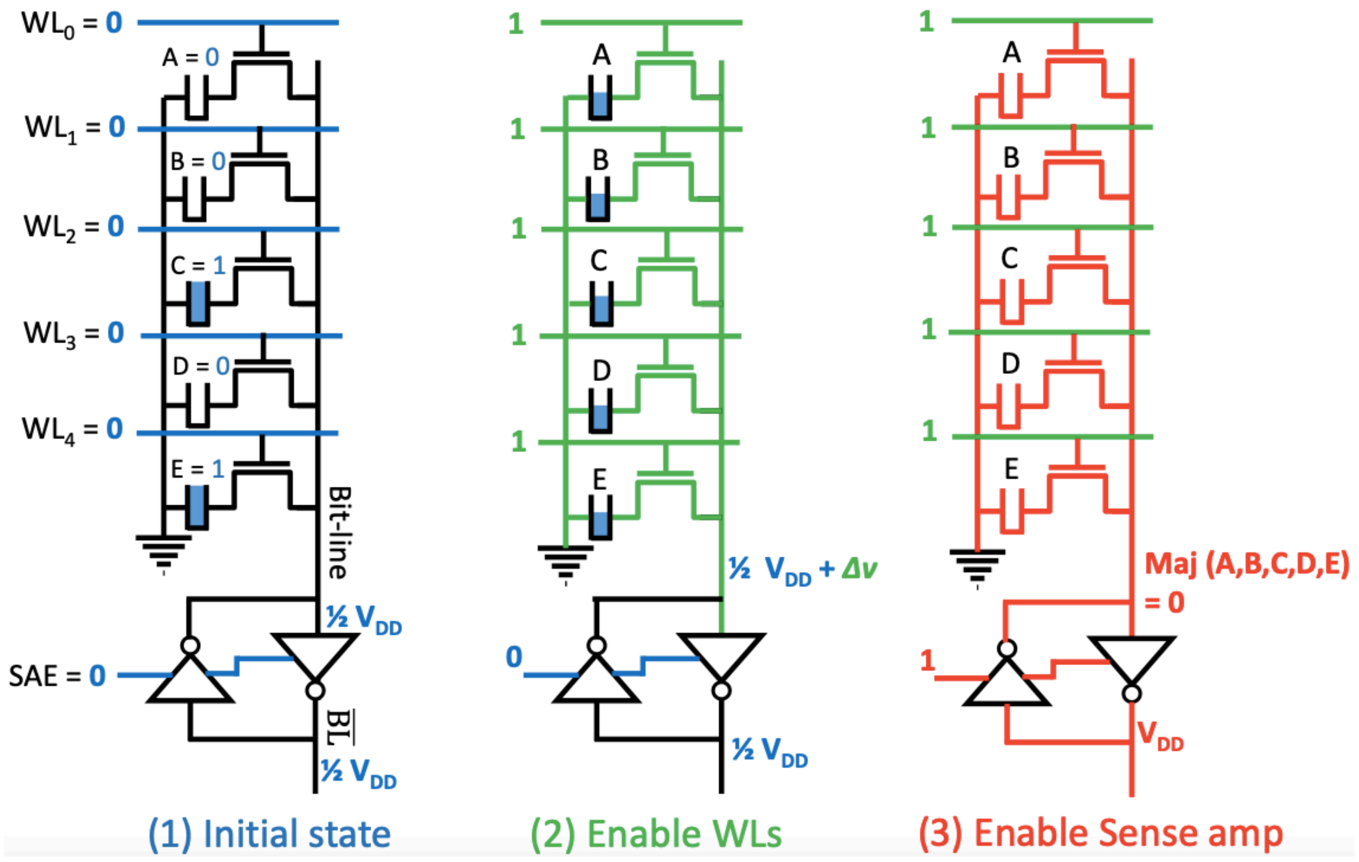


In-DRAM addition

$$Cout = Majority(A, B, Cin)$$

$$Sum = Majority(A, B, Cin, \overline{Cout}, \overline{Cout})$$

- Note that the operations are bit-serial

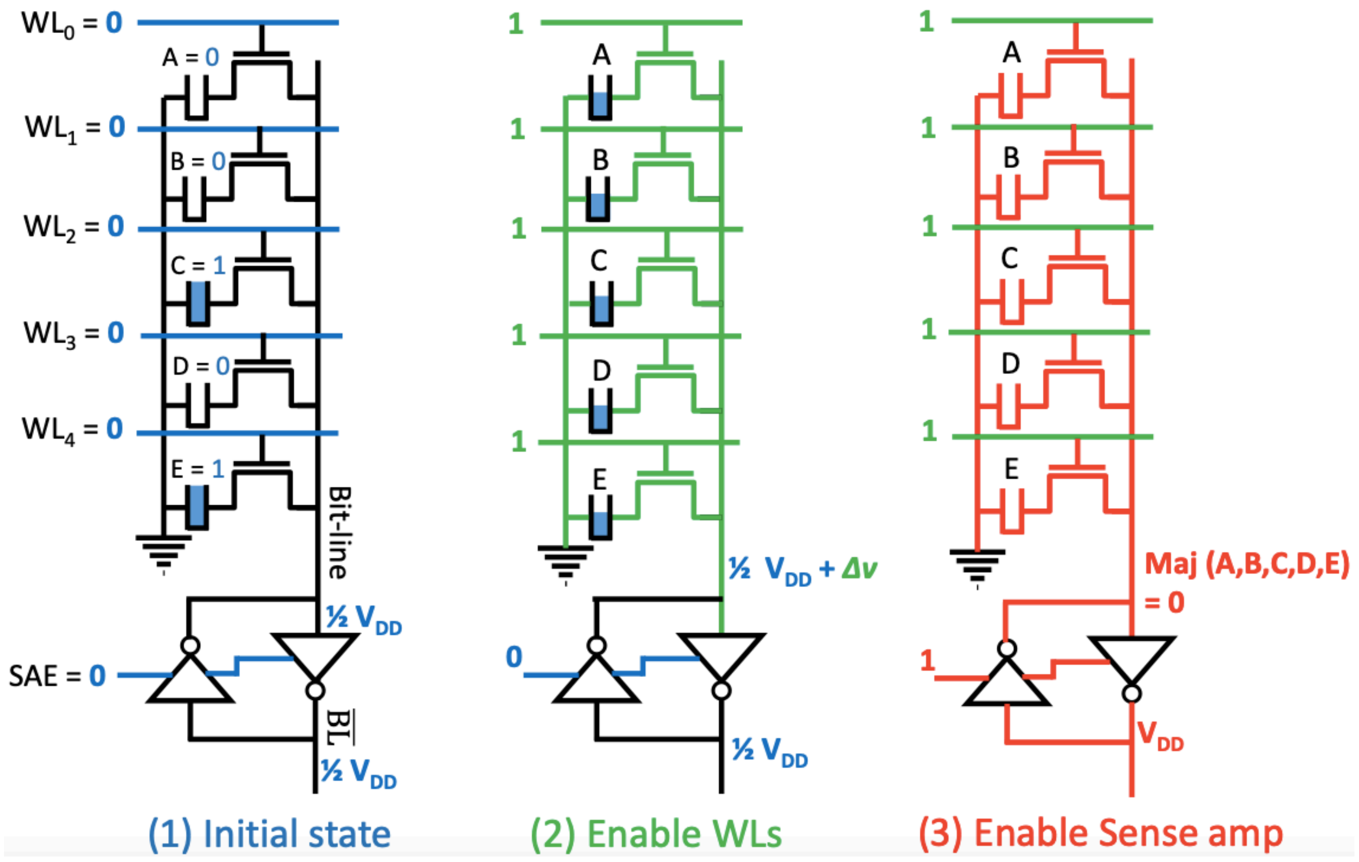


In-DRAM addition

$$Cout = Majority(A, B, Cin)$$

$$Sum = Majority(A, B, Cin, \overline{Cout}, \overline{Cout})$$

- Note that the operations are bit-serial
- … but word-parallel

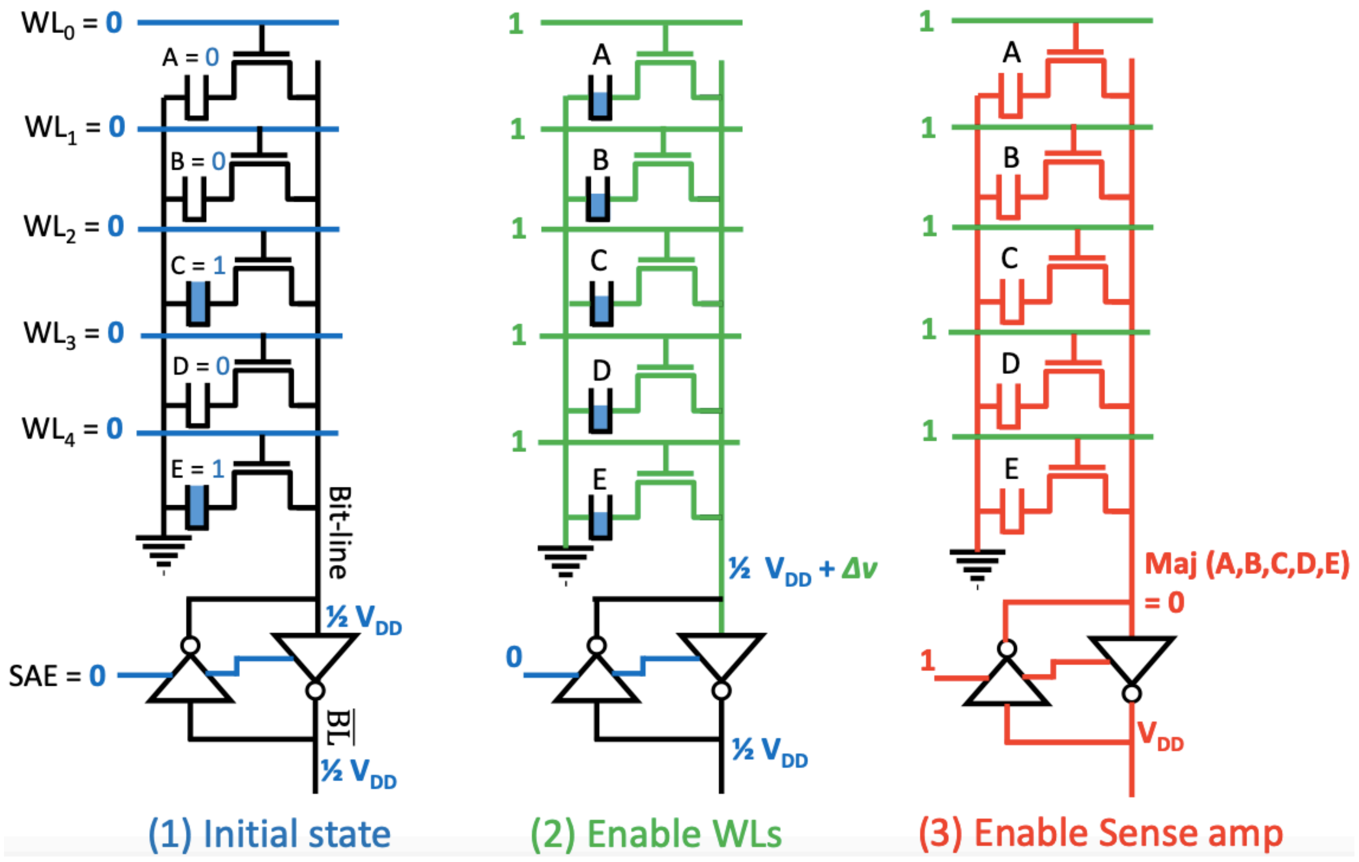


In-DRAM addition

$$Cout = Majority(A, B, Cin)$$

$$Sum = Majority(A, B, Cin, \overline{Cout}, \overline{Cout})$$

- ❑ Note that the operations are bit-serial
- ❑ … but word-parallel
- ❑ Particularly useful for bulk additions on low-precision numbers



Other operations using CIM-DRAM

- *Majority* can also be used to implement other operations (logic, arithmetic)

Other operations using CIM-DRAM

- *Majority* can also be used to implement other operations (logic, arithmetic)
- For masked counting using DRAM:
  - Assuming Johnson encoding
  - The shifting and invert-back can be implemented using basic in-DRAM ops.

Lima et al., Arxiv, 2024

Other operations using CIM-DRAM

- *Majority* can also be used to implement other operations (logic, arithmetic)
- For masked counting using DRAM:
  - Assuming Johnson encoding
  - The shifting and invert-back can be implemented using basic in-DRAM ops.

$$b_i = (b_i \wedge \overline{m}) \vee (b_{i-1} \wedge m), \quad \text{where } i \in \{n, n-1, \ldots, 2\}$$

$$b_1 = (b_1 \wedge \overline{m}) \vee (\overline{b_n} \wedge m)$$

Lima et al., Arxiv, 2024

CIM-logic using memristors

- The fundamental idea is similar to DRAM, i.e., multi-row activation

Xie et al., ISVLSI 2017

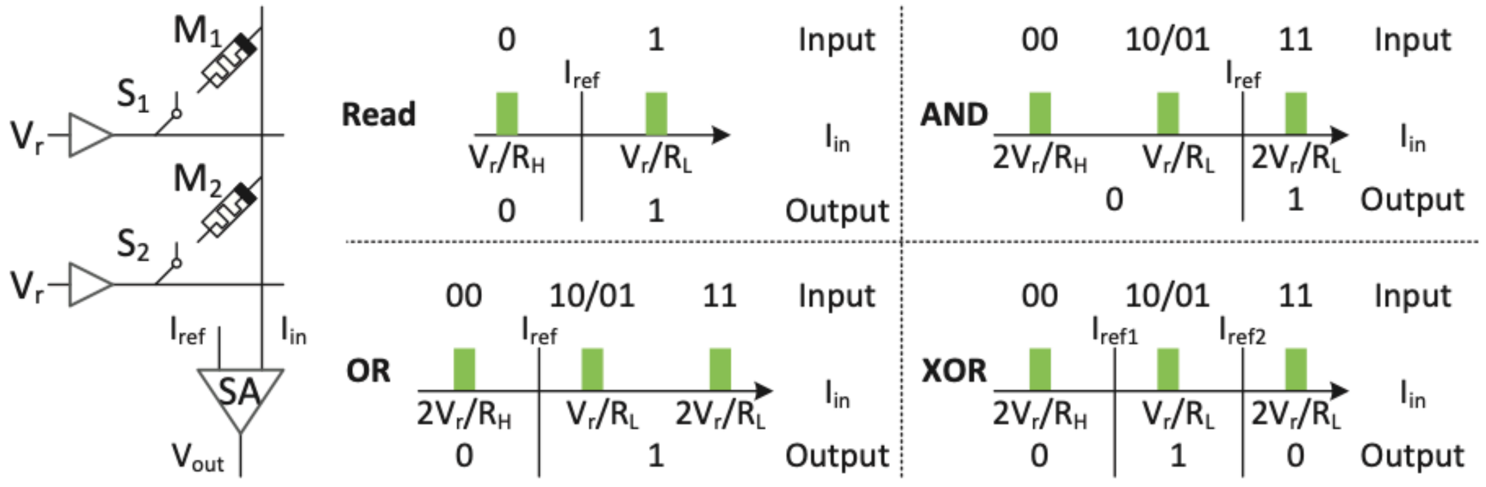
CIM-logic using memristors

- The fundamental idea is similar to DRAM, i.e., multi-row activation
- The output is compared to a reference voltage

Xie et al., ISVLSI 2017

CIM-logic using memristors

- The fundamental idea is similar to DRAM, i.e., multi-row activation
- The output is compared to a reference voltage



(a) Memory (b) References of Primitive Operations

Thank you!

asif.ali@uetpeshawar.edu.pk