ComputeDRAM: In-Memory Compute using Off-the-Shelf DRAMs

IEEE/ACM International Symposium on Microarchitecture (MICRO-52) 12/2019

Fei Gao
Princeton University

Georgios Tziantzioulis
Princeton University

David Wentzlaff
Princeton University

Seminar in Computer Architecture
Today's Presenter: Patrick Eppensteiner

Executive Summary

Problem: Memory Wall

- Moving data from and to memory incurs long access latency
- Existing solutions are not feasible (for DRAM manufacturers)

Goal: Proof of concept that in-memory computation is possible with unmodified DRAM modules

ComputeDRAM

- In-memory computation using **minimal modifications**
- Off-the-shelf, unmodified, commercial DRAM
- **Row Copy, AND** and **OR**: logical completeness by saving inverse values as well
- Arbitrary computations made possible using a modified memory controller

Results

- In-memory computation possible with unmodified DRAM modules
- Financially feasible way for DRAM manufacturers to support in-memory compute

Problem & Goal

Memory wall problem

- Moving data utilizes a large portion of the overall system energy and program execution time
- Need for in-memory computation
- General problem memory processing tries to solve

Existing solutions exist, but not financially feasible for DRAM manufacturers

ComputeDRAM's goal

- Implementation of in-memory computation using **off-the-shelf, unmodified, commercial DRAM**

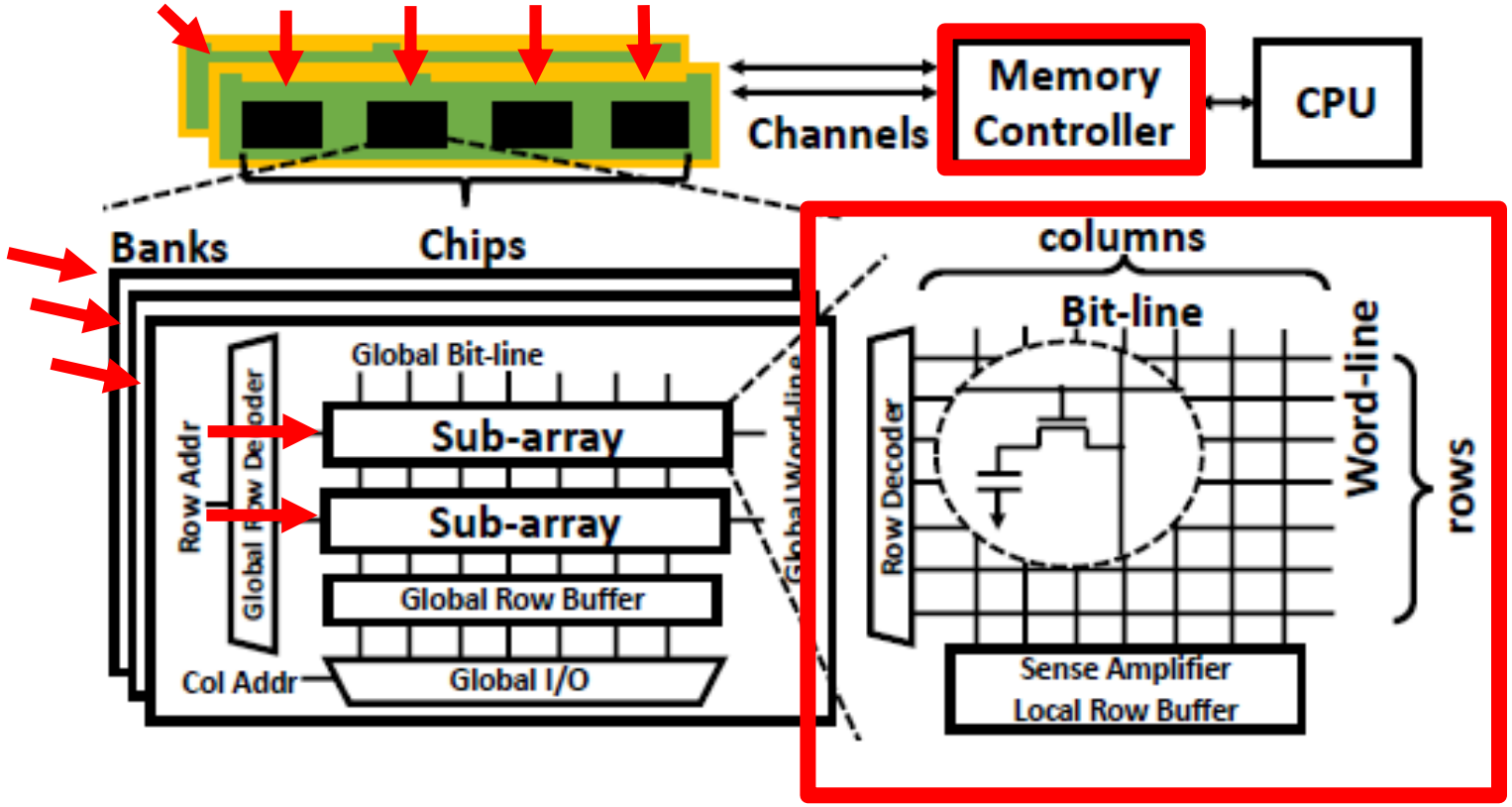
Main Goal

Prove that in-memory computation is **possible with unmodified DRAM modules** and the existence of a **financially feasible** way for DRAM manufacturers to **support in-memory compute.**

In other words: Show a **proof of concept** of in-memory computation utilizing **minimal modifications.**

- Maybe not in a practical way, but supports the idea of memory processing nonetheless

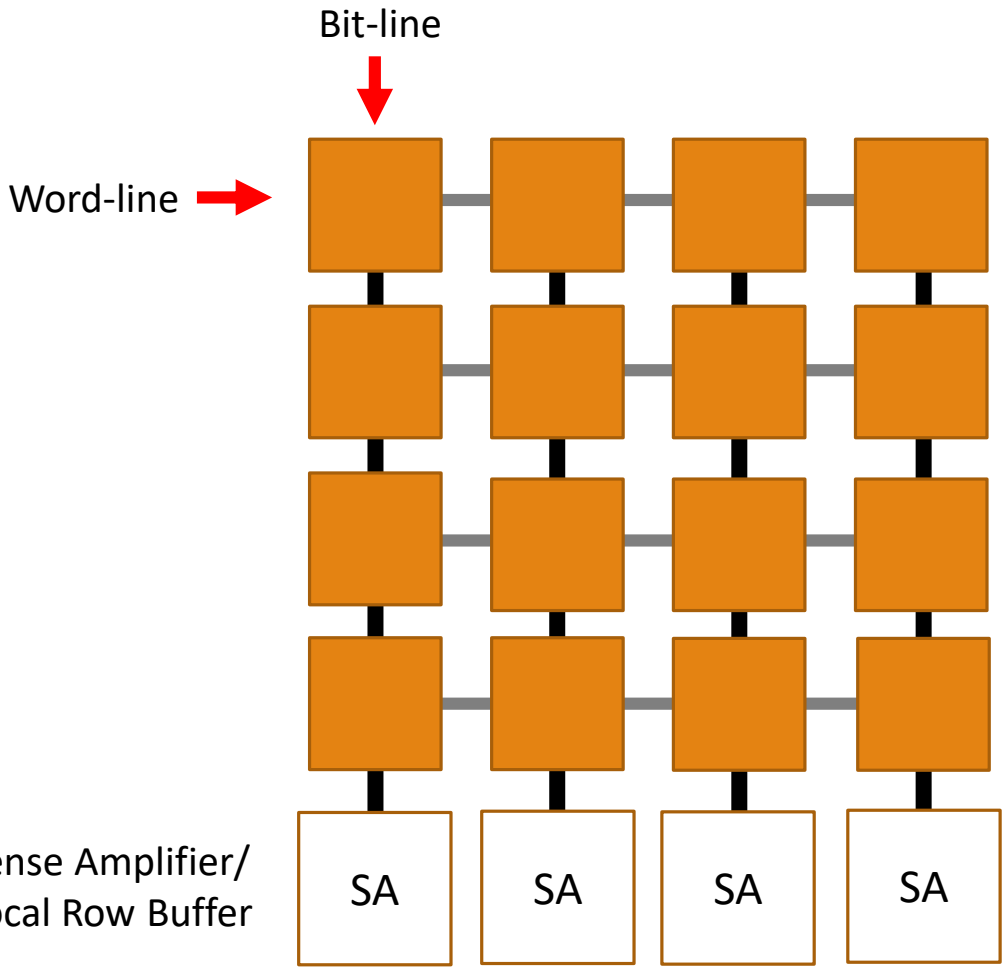
Background



Typical DRAM hierarchy:

- Up to two ranks
- Rank has 8 chips
- Chip has 8 banks
- Bank is split into sub-arrays

DRAM Sub-array



Columns connected by the **bit-line**

Rows connected by the **word-line**

Logical 1 at V_{dd}

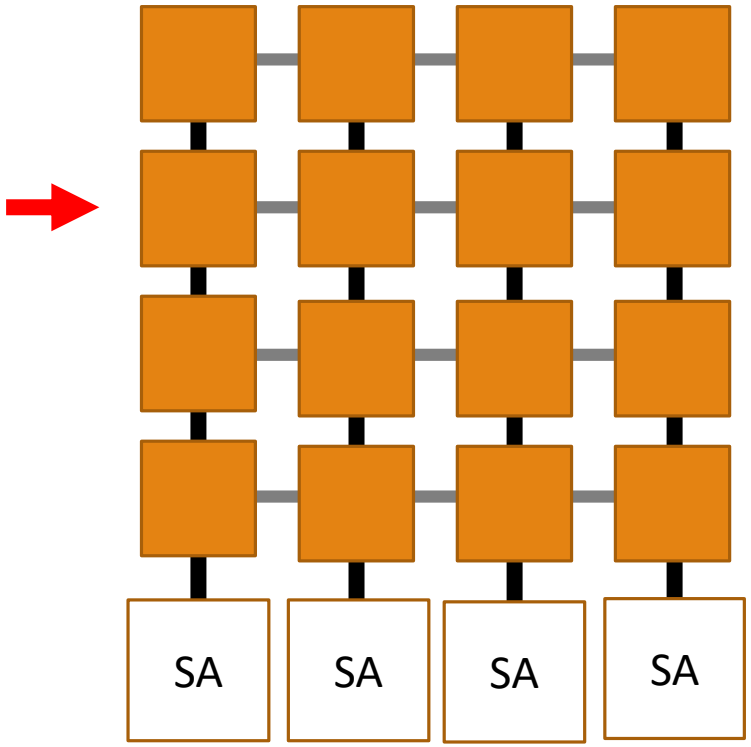
Logical 0 at $V = 0, (GND)$

Charge of bit-lines in idle state $V_{dd}/2$

Differential Sense Amplifier

- SA pulls Charge (of bit-line) up if $V > V_{dd}/2$
- SA pulls Charge (of bit-line) down if $V < V_{dd}/2$

Accessing DRAM (I)

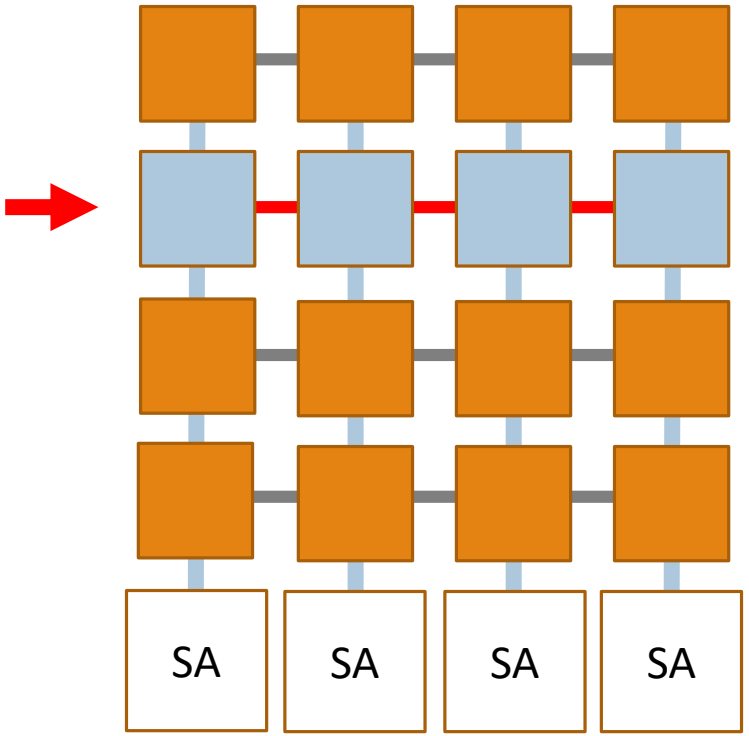


To access DRAM (read or write) the memory controller issues three main commands:

1) ACTIVATE
2) READ/WRITE
3) PRECHARGE

1) ACTIVATE:
   - Applies to row
   - Activates word-line connecting it to the bit-line
   - Sense amplifiers are enabled and amplify charge to V_{dd} or GND
   - Value of the cell is preserved

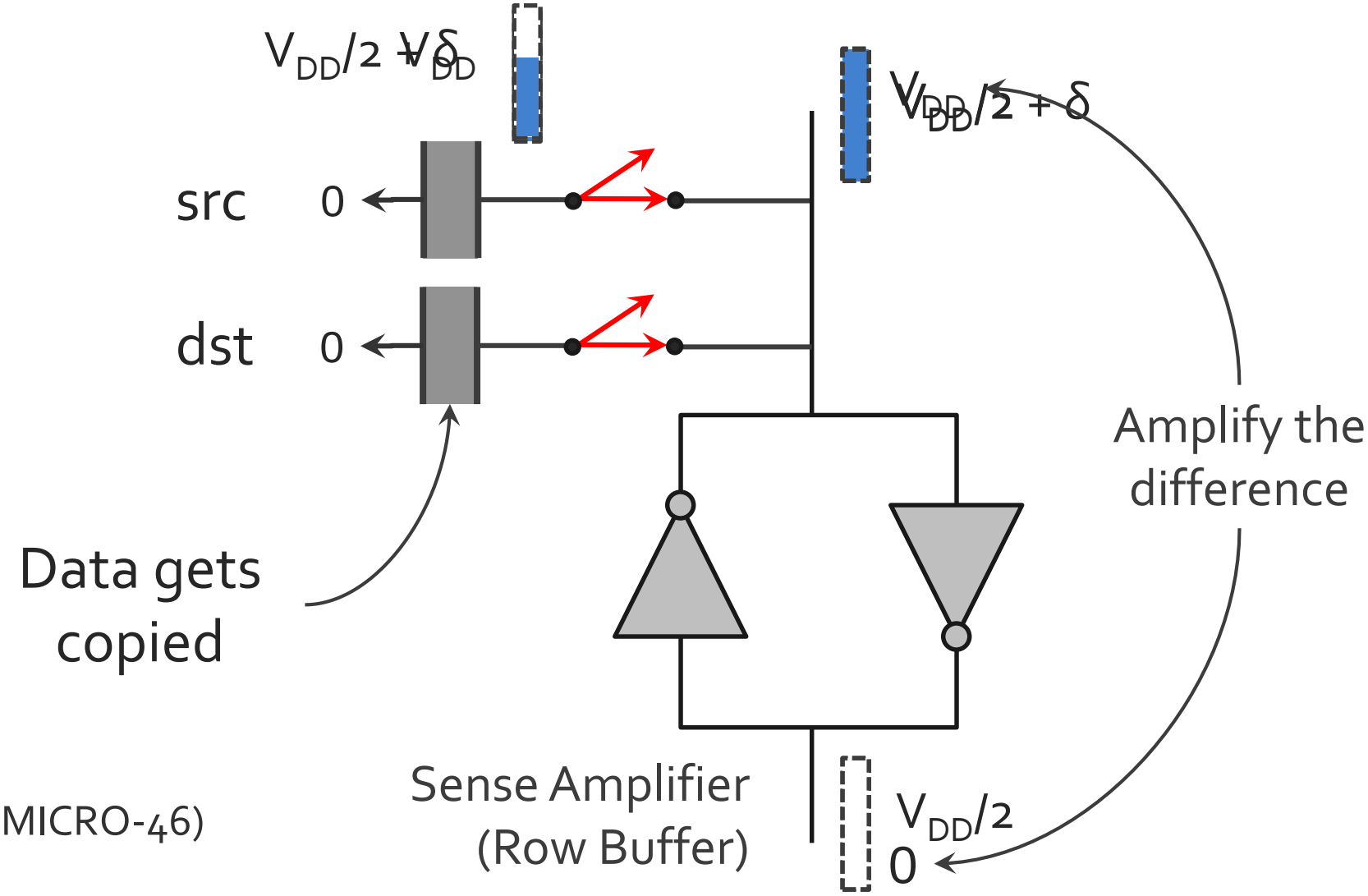
Accessing DRAM (II)



2) READ/WRITE:
   - Follows ACTIVATE
   - Reading/writing process is based on ACTIVATE and PRECHARGE commands

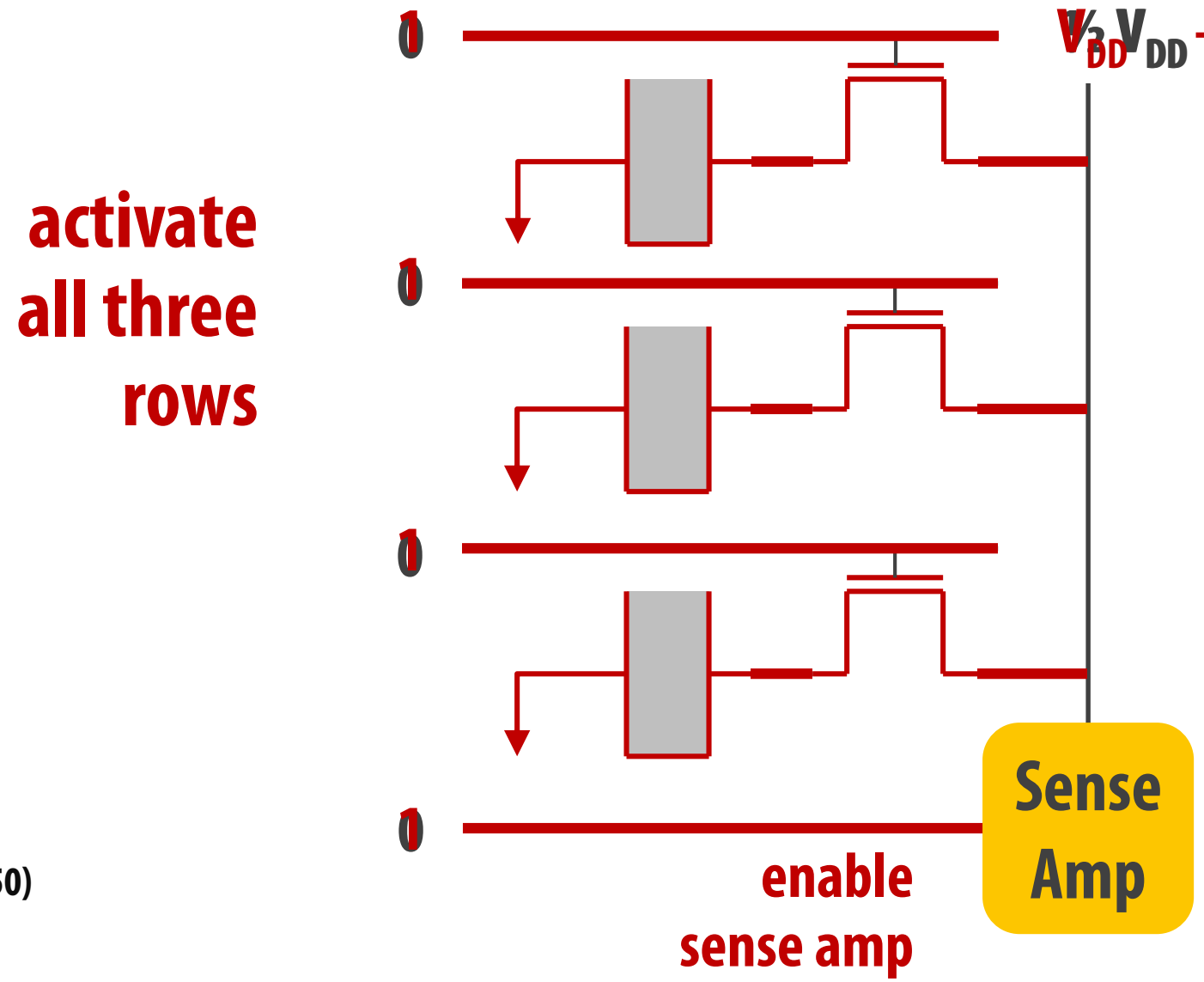
3) PRECHARGE:
   - Applies to bank
   - Closes currently (all) opened row(s)
   - Pull bit-lines to $V_{dd}/2$

RowClone: In-DRAM Bulk Copy and Initialization



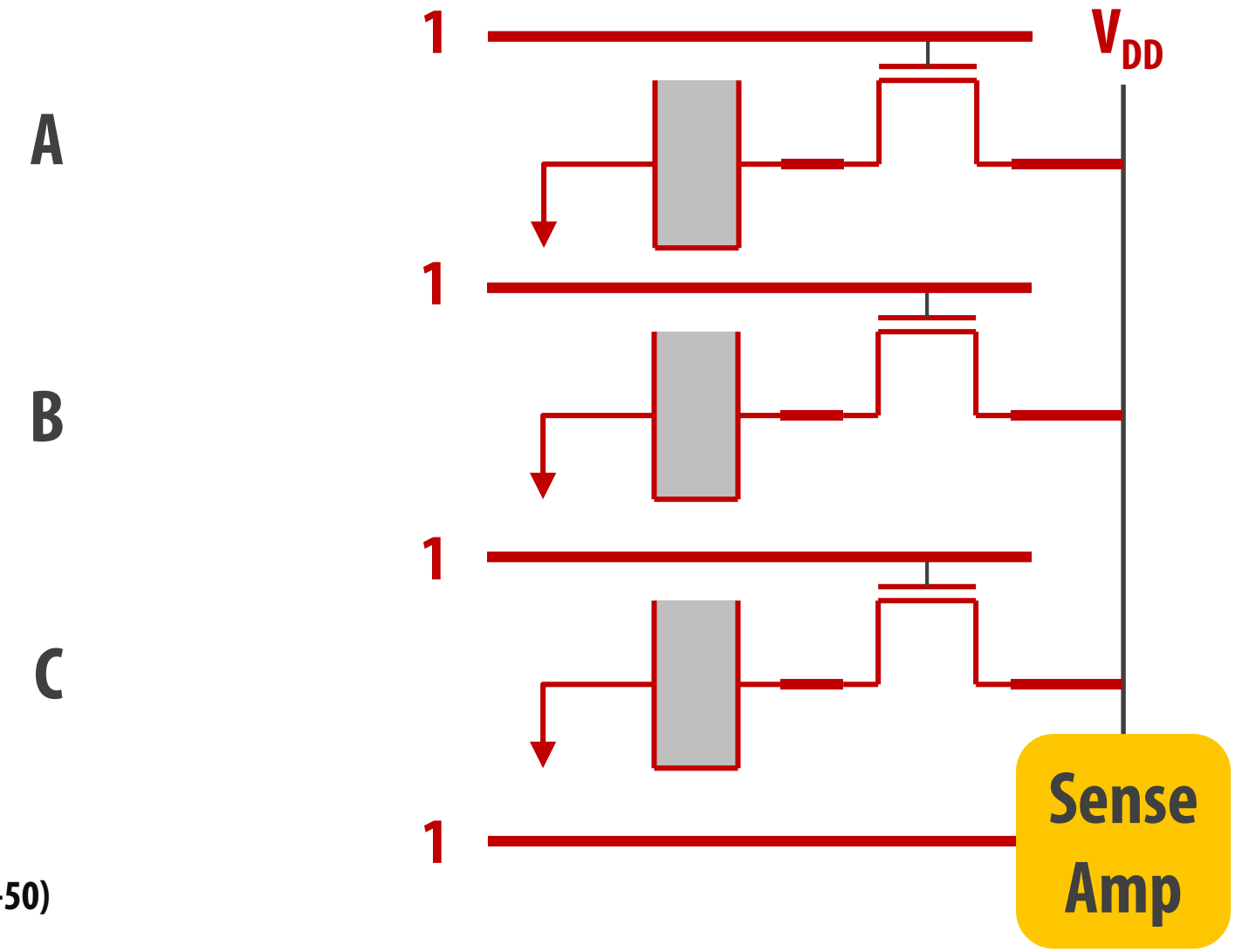
Credit: RowClone (MICRO-46) 12/2013

Ambit: In-DRAM Bulk Bitwise Operations (Majority Function)



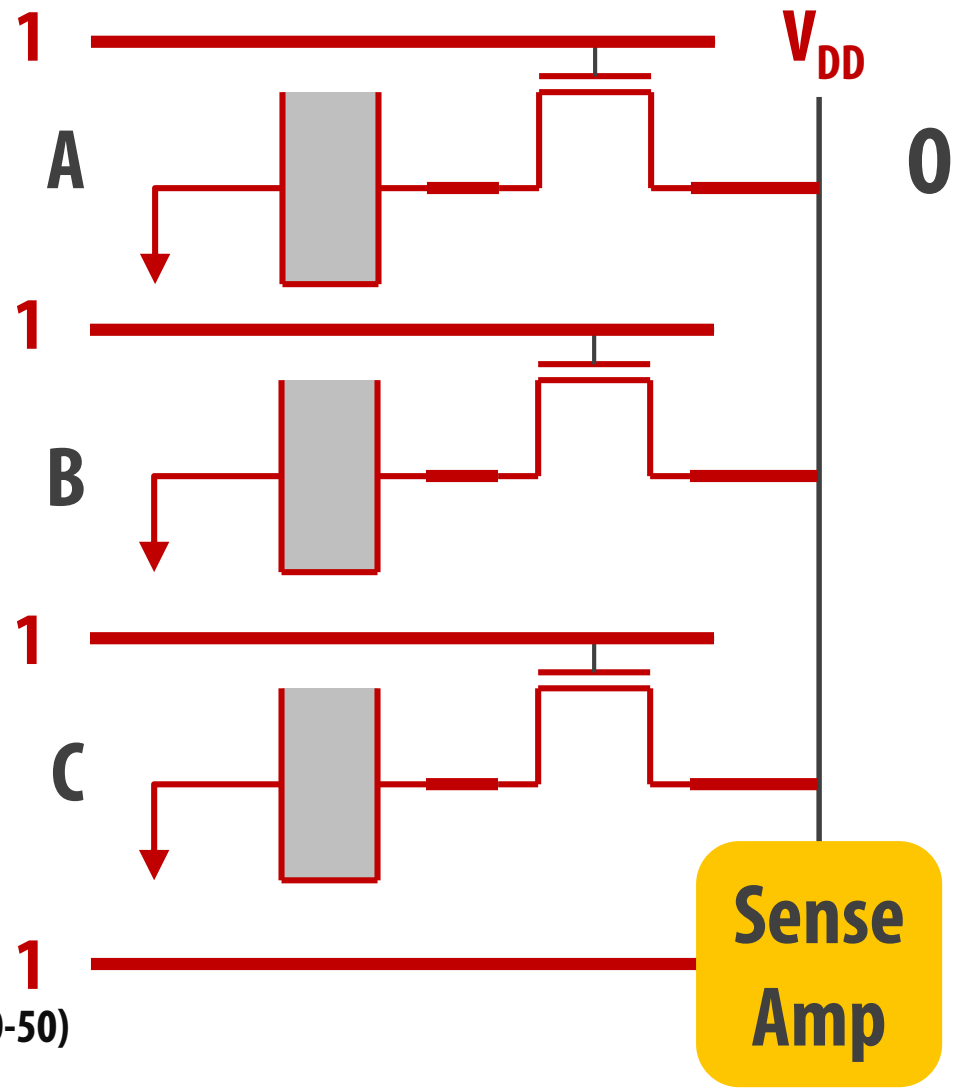
Credit: Ambit (MICRO-50)
10/2017

Bitwise AND/OR Using Triple-Row Activation



Credit: Ambit (MICRO-50)

Bitwise AND/OR Using Triple-Row Activation



$$\text{Output} = AB + BC + CA$$
$$= C(A \textbf{ OR } B) +$$
$$\sim C(A \textbf{ AND } B)$$

Control the value of C to perform bitwise OR or bitwise AND of A and B

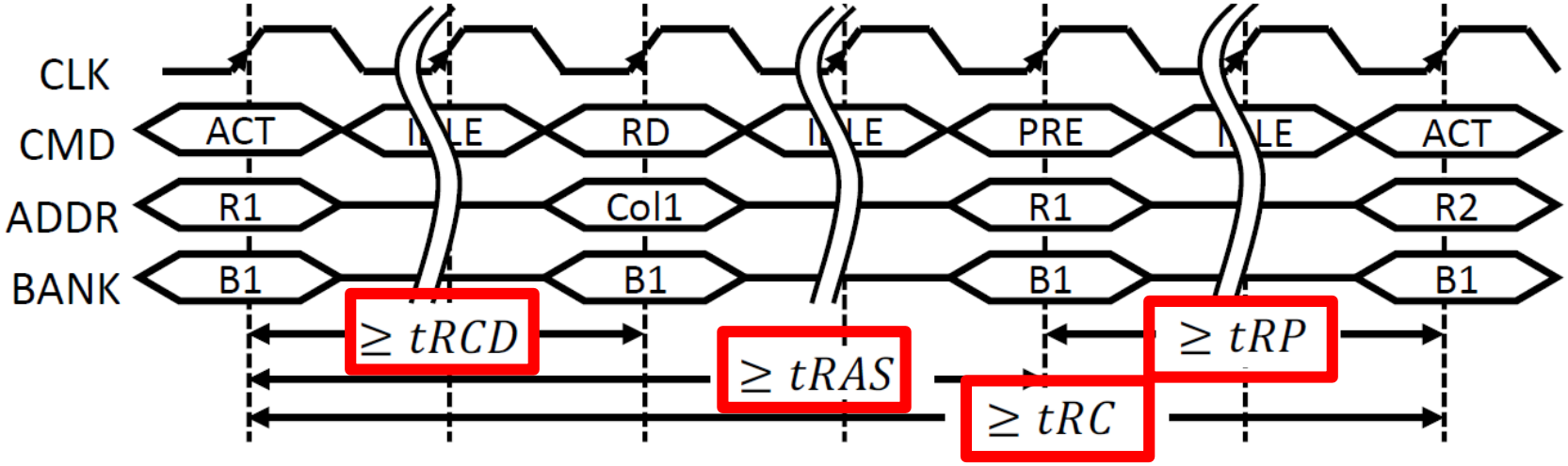
Credit: Ambit (MICRO-50)

DRAM Command Timings

Timings are important for correctness

Part of the specifications

- Row to Column Access Delay t_{RCD}
- Row Access Strobe t_{RAS}
- Row Precharge t_{RP}
- Row Cycle $t_{RC} = t_{RAS} + t_{RP}$



Novelty, Key Ideas and Approaches

Novelty

- Perform computations with **off-the-shelf, unmodified, commercial** DRAM

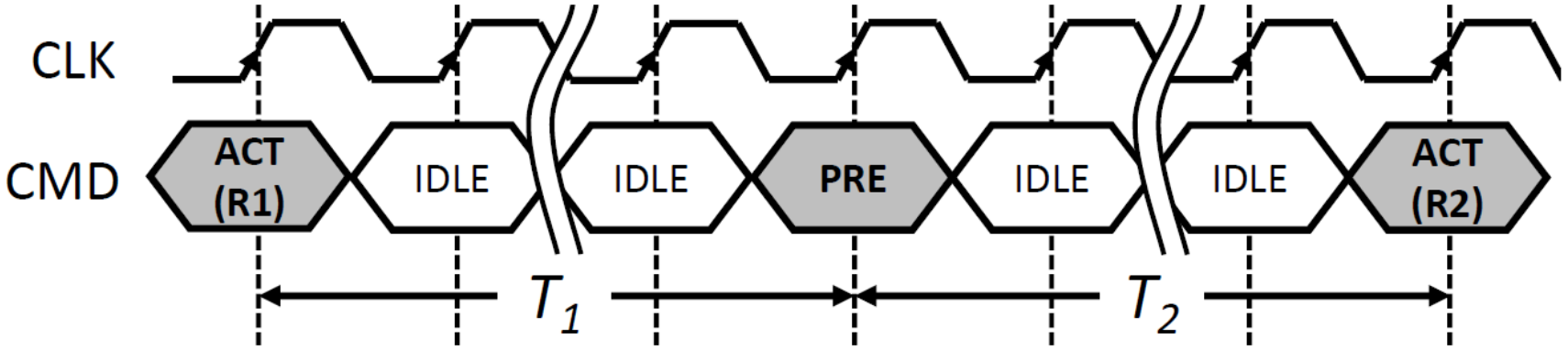
Key Idea

- Use command sequences that **violate the timing specifications** (t_{RAS} and t_{RP}).
- Hope to find **new functionality**

Only modify the memory controller (MC)

- Using SoftMC, an experimental FPGA-based memory controller design

Mechanisms



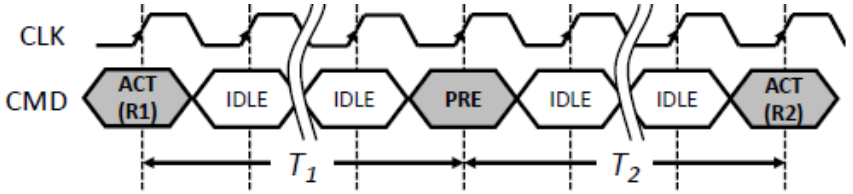
T_1 and T_2 are the amounts of IDLE cycles in the **time interval** between two commands

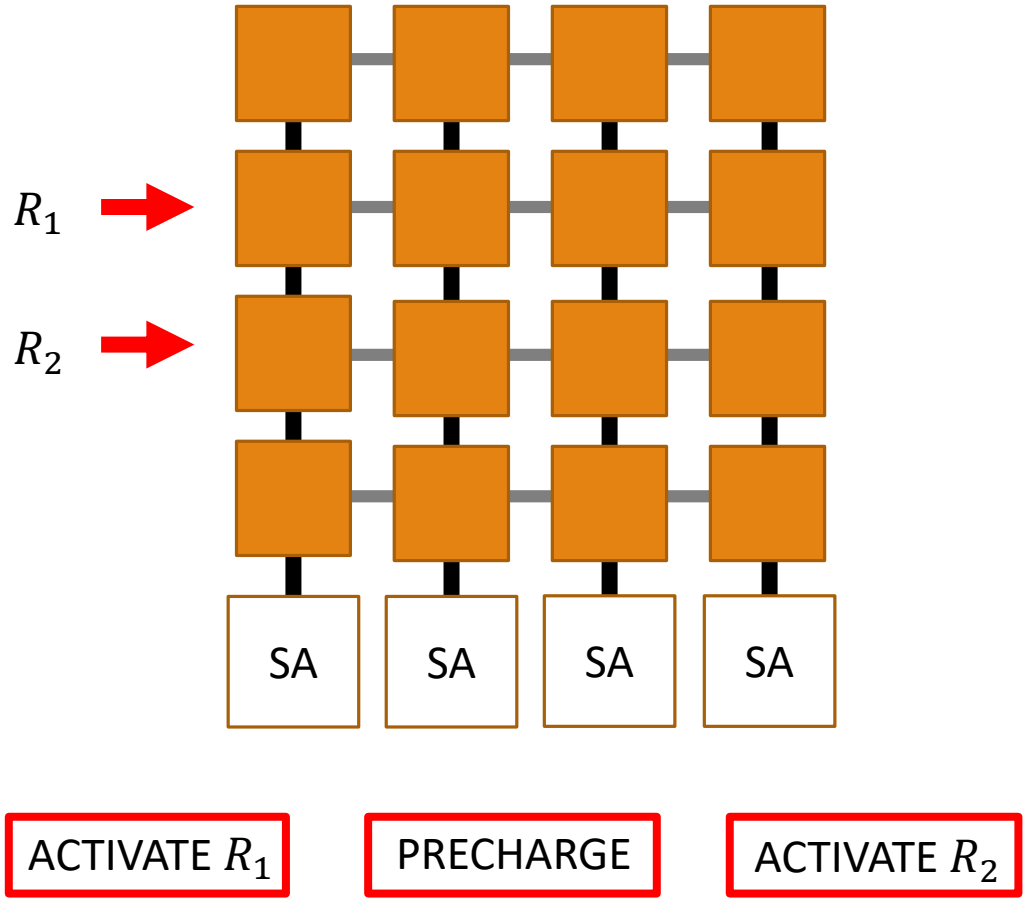
- e.g. $T_1 = 0$ means that there are **zero** IDLE cycles between ACTIVATE (R_1) and PRECHARGE

By reducing T_1 and T_2 below the specification limits, three basic in-memory operations can be implemented:

- Row Copy
- AND
- OR

Row Copy





Performs copy from R_1 to R_2

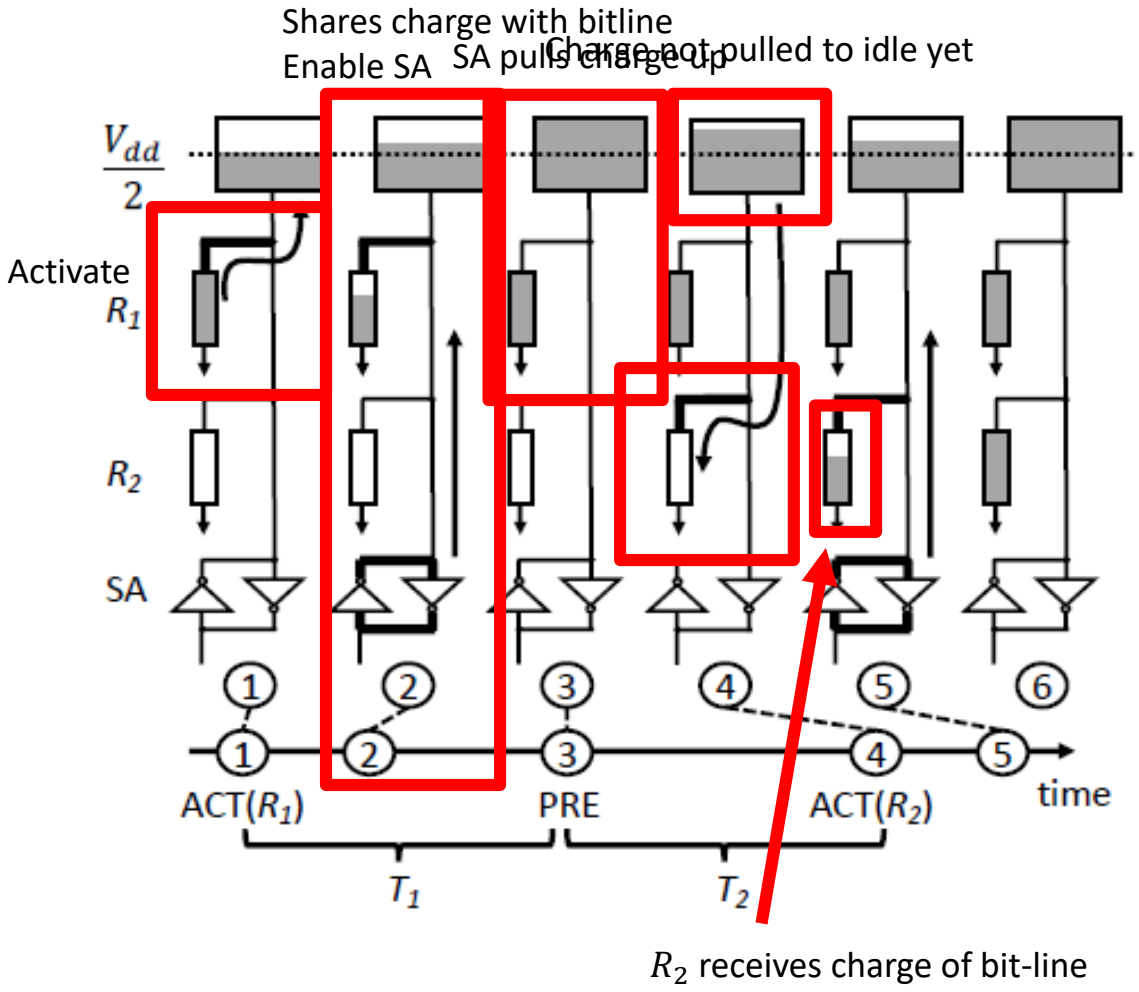
Interrupt PRECHARGE with ACTIVATE (R_2) before bit-line is pulled to $\frac{V_{dd}}{2}$

Note: It works because the capacitance of the bit-line is larger than of the cell

Observations:

- Source (R_1) and destination (R_2) row must **share physical bit-line**
- Two successive ACTIVATEs are needed
  - **PRECHARGE command is indispensable**

Row Copy Example



Performs copy from one row to another

- R_1 to R_2

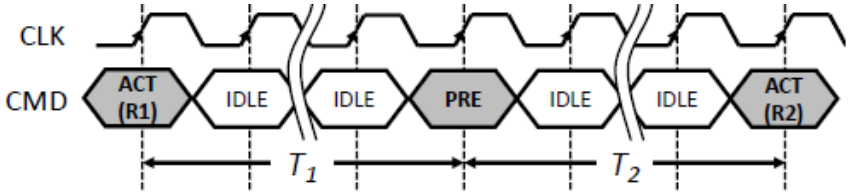
Interrupt PRECHARGE with ACTIVATE (R_2) before bit-line is pulled to $\frac{V_{dd}}{2}$

Note: Capacitance of the bit-line is larger than of the cell

Observations:

- Source (R_1) and destination (R_2) row must **share physical bit-line**
- Two successive ACTIVATEs are needed
  - **PRECHARGE command is indispensable**

AND/OR (I)

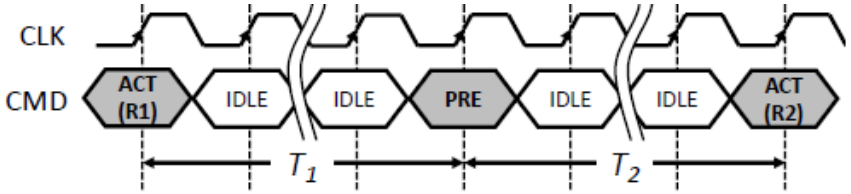


By reducing both T_1 and T_2 **three rows can be opened.**

Similar to the “majority” function in Ambit, however:

- Only **two rows** are explicitly activated in quick succession, not three
- Rows are not activated **at the same time**
- Rows are not equivalent in “majority wins”

AND/OR (II)



Third row is opened implicitly

- When the row address is updated from R_1 to R_2

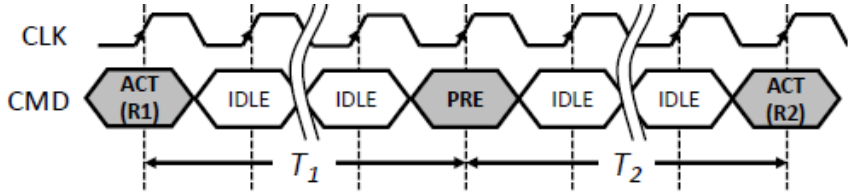
Observation: The update starts at the least significant bit

- When only 2 bits diverge, **only** the least significant bit is updated!
- E.g. $R_1 = 0\mathbf{0}_2$, $R_2 = 1\mathbf{1}_2$, opened row $R_3 = 0\mathbf{1}_2$
- Note: This does not work the same in every row decoder

Requirements:

- All three rows must share the physical bit-line
- Fix the addresses of R_1 and R_2 , such that R_3 opens predictably
- Need to initialize R_3 with all-ones for OR, initialize R_1 with all-zeros for AND

AND/OR Truth Table



The three fixed row addresses: $R_1 = 0\mathbf{1}_2$, $R_2 = 1\mathbf{0}_2$, opened row $R_3 = 0\mathbf{0}_2$

Order of activation: $R_1 \rightarrow R_3 \rightarrow R_2$

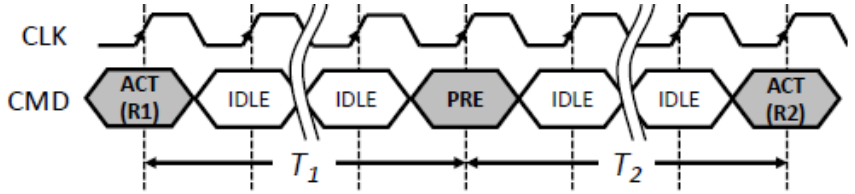
R_3 \ $R_1\ R_2$	**00**	**01**	**10**	**11**
0	0	0	X	1
1	0	1	1	1

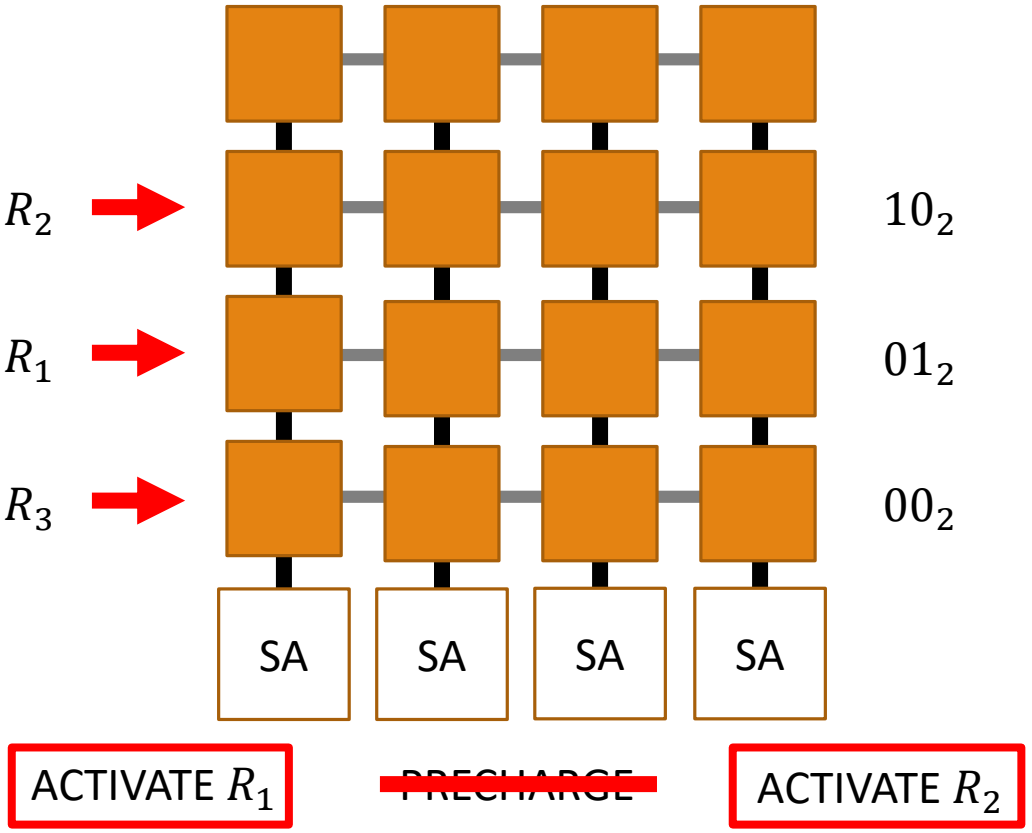
OR (using $R_3 = 1$)

Rows are not equivalent in “majority wins” !

AND (using $R_1 = 0$)

AND/OR (III)





$T_1 = T_2 = 0$ (no idle cycles)

ACTIVATE R_1 → PRECHARGE → ACTIVATE R_2

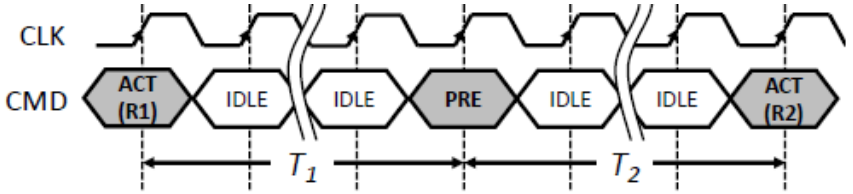
AND
- Constant: $R_1 = 0$
- Operands: R_2 and R_3

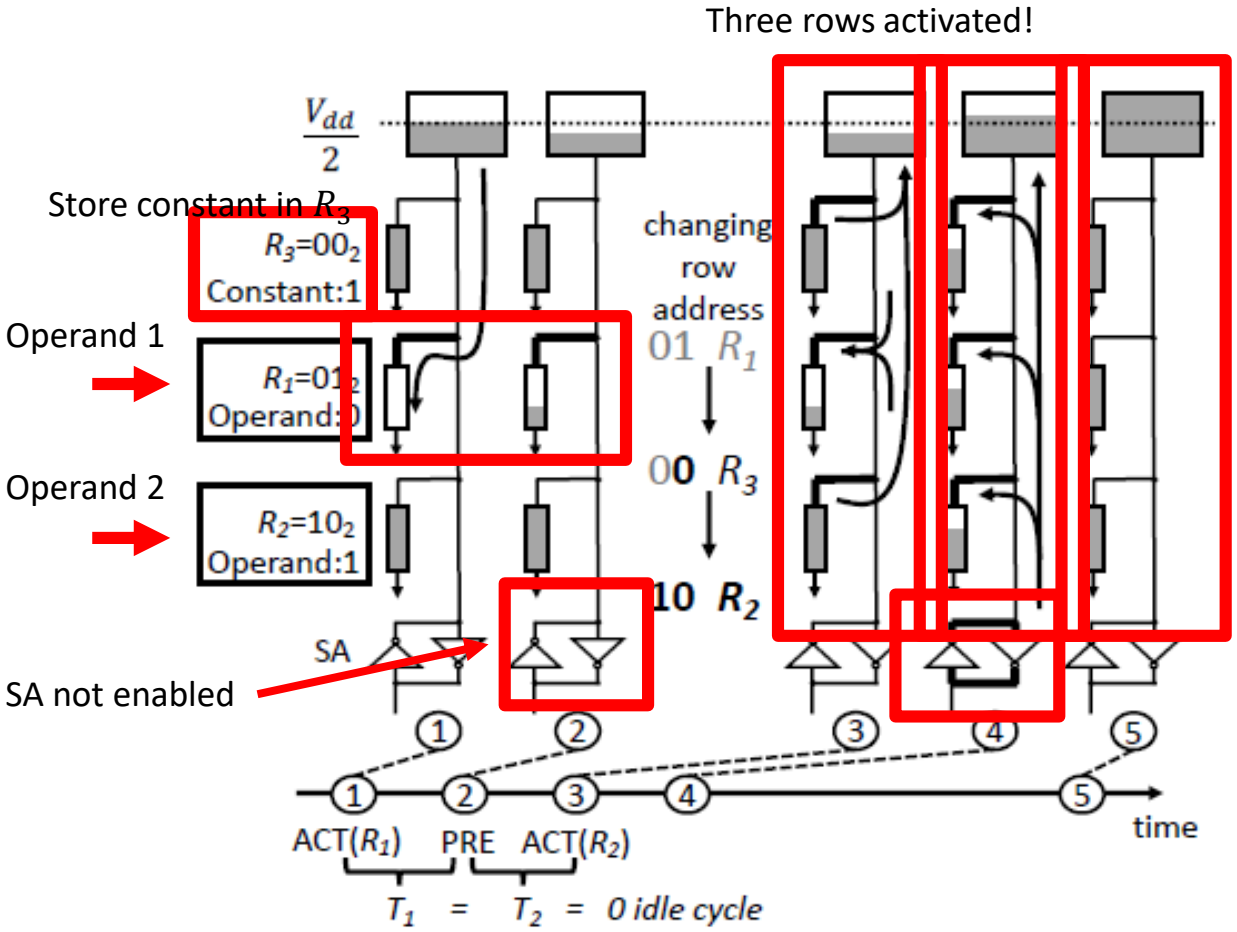
OR
- Constant: $R_3 = 1$
- Operands: R_1 and R_2

Interrupt ACTIVATE R_1 before SA is enabled

Interrupt PRECHARGE with ACTIVATE R_2

OR Example





$T_1 = T_2 = 0$ (no idle cycles)

ACTIVATE R_1 → PRECHARGE → ACTIVATE R_2

OR

- Constant: $R_3 = 1$
- Operands: R_1 and R_2

Implementations

Implementations

- AND, OR and RowCopy

Arbitrary Computations with Memory

- Achieve logical completeness using a pseudo-NOT implementation (always storing the inverse of a value as well)

Implementation choices

- Computation is performed in first three rows in each sub-array ending with 00, 01, 10.
- Operands are saved in rows after the first three and copied using row copy

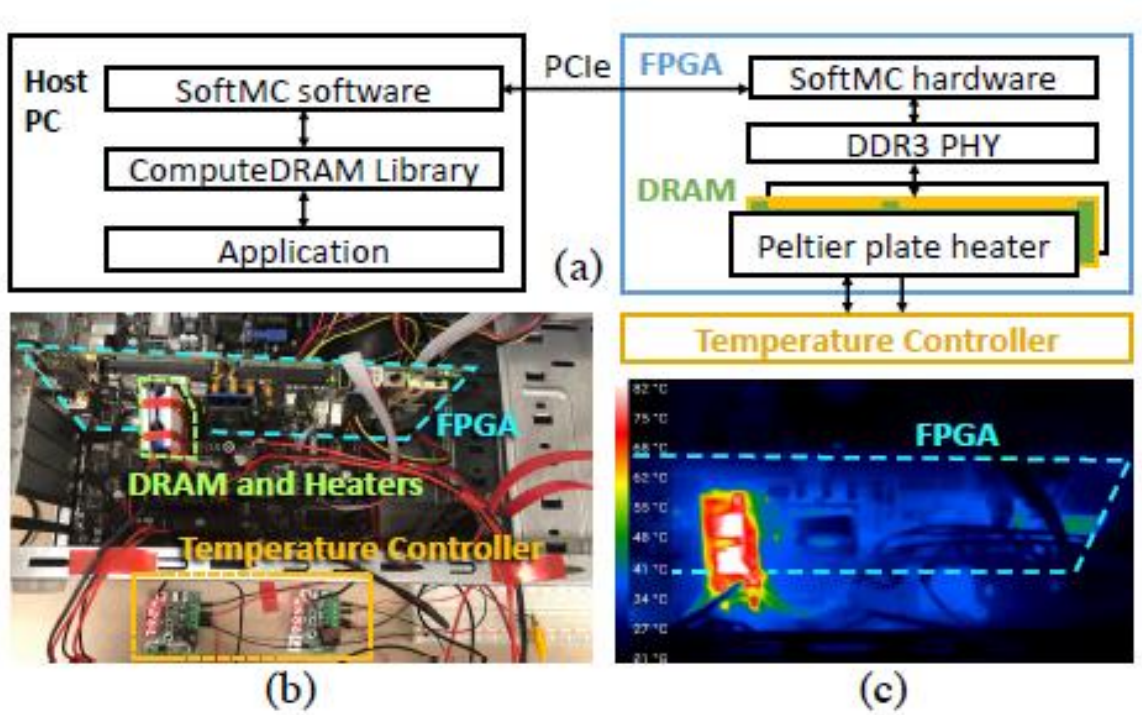
Methodology

Methodology

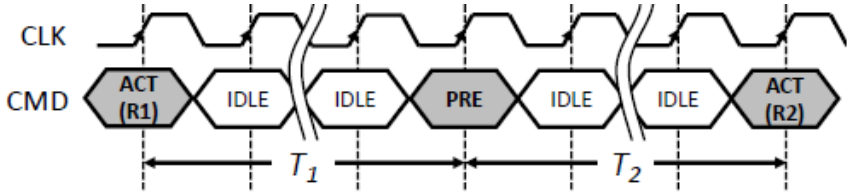
- SoftMC
- Choice of hardware constrains the timing intervals to multiples of 2.5ns
- only DDR3, no DDR4 tested
- temperature of the environment was tested as well using heaters

Addressing operation reliability issues

- Manufacturing Variations
- Row Remapping
- By using an error table can avoid “bad” rows



Evaluation



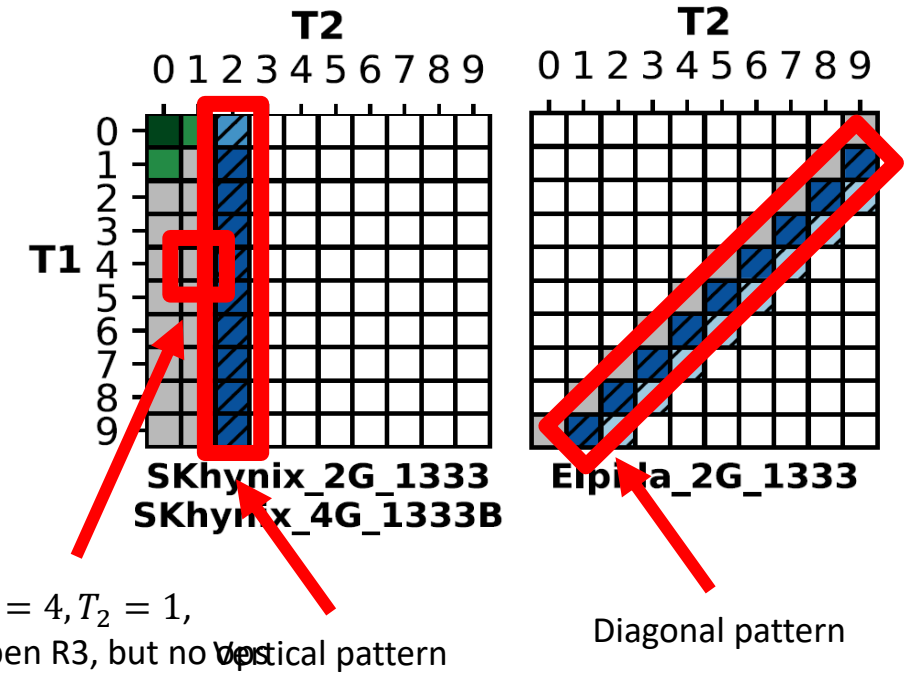
Nominal operation (1.5 V, 25-30 °C)

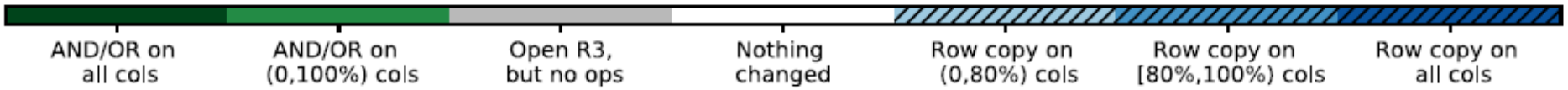
Not a reliability test
- Bank was chosen with the lowest success rate

Heatmap using T_1 and T_2 as coordinates
- Green color concerns ability to perform AND/OR
- Blue color concerns ability to perform Row copy

Row copy vertical pattern
- T_1 needs to be at least long enough for ACTIVATE to fully pull the bit-lines up or down
- T_2 needs to be short, but not too short
  - As not to open a third row

Row copy diagonal pattern
- Issue of second ACTIVATE is held back by the first ACTIVATE



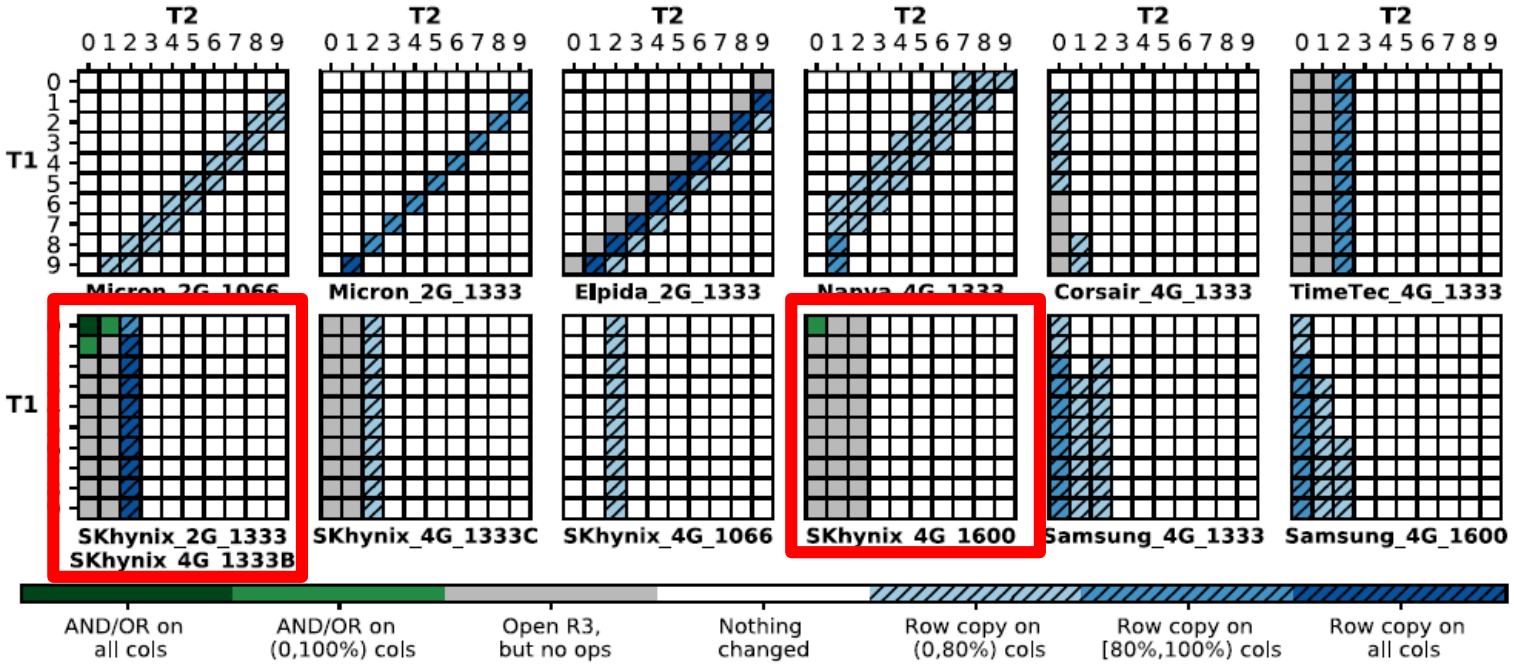


Heatmaps

All tested DRAM modules are able to perform at least **partial Row Copy**

SKhynix_2G_1333, SKhynix_4G_1333B and SKhynix_4G_1600 are able to perform at **least partial AND/OR**

SKhynix_2G_1333, SKhynix_4G_1333B is able to perform **all operations**



Conclusion

Problem: Memory Wall

- Moving data from and to memory incurs long access latency
- Existing solutions are not feasible (for DRAM manufacturers)

Goal: Proof of concept that in-memory computation is possible with unmodified DRAM modules

ComputeDRAM

- In-memory computation using **minimal modifications**
- Off-the-shelf, unmodified, commercial DRAM
- **Row Copy, AND** and **OR**: logical completeness by saving inverse values as well
- Arbitrary computations made possible using a modified memory controller

Results

- In-memory computation possible with unmodified DRAM modules
- Financially feasible way for DRAM manufacturers to support in-memory compute

Strengths

Proof-of-concept works

Paper goes even further

- Arbitrary computations (implementation is logically complete)
- Simple arithmetic example

Well written paper

Explanations are easy and intuitive to understand

Weaknesses

It is not clear what mechanisms follow from observation and what follows from factual knowledge about DRAM

- i.e. row address change does not work the same in every row decoder

It is not very clear which contribution is made by the paper and which contributions are from RowClone/Ambit

DRAM is unmodified but a lot of changes are needed in the system

Even though it might be clear, performance gains are only mentioned briefly on paper, no charts ☹

Further Readings

- **Ambit**: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology
  - https://people.inf.ethz.ch/omutlu/pub/ambit-bulk-bitwise-dram_micro17.pdf
- **RowClone**: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization
  - https://users.ece.cmu.edu/~omutlu/pub/rowclone_micro13.pdf
- **DRISA**: A DRAM-based Reconfigurable In-Situ Accelerator
  - https://www.ece.ucsb.edu/Faculty/selected_pubs/xie/2017-MICRO-DRISA%20A%20DRAM%20Based%20Reconfigurable%20In-Situ%20Accelerator.pdf
- **DrAcc**: a DRAM based Accelerator for Accurate CNN Inference
  - https://ieeexplore.ieee.org/document/8465866
- **Neural Cache:** Bit-Serial In-Cache Acceleration of Deep Neural Networks
  - http://blaauw.engin.umich.edu/wp-content/uploads/sites/342/2019/10/Neural-Cache_-Bit-Serial-In-Cache-Acceleration-of-Deep-Neural-Networks.pdf

Discussion

What makes this work (un)practical for actual use?

- What further modifications in the system are needed?
- What is the main overhead introduced in the implementation?

What effect does it have on DRAM manufacturers to support in-memory computation?

What type of workloads can benefit from this technology?

How can more complex computations be implemented?

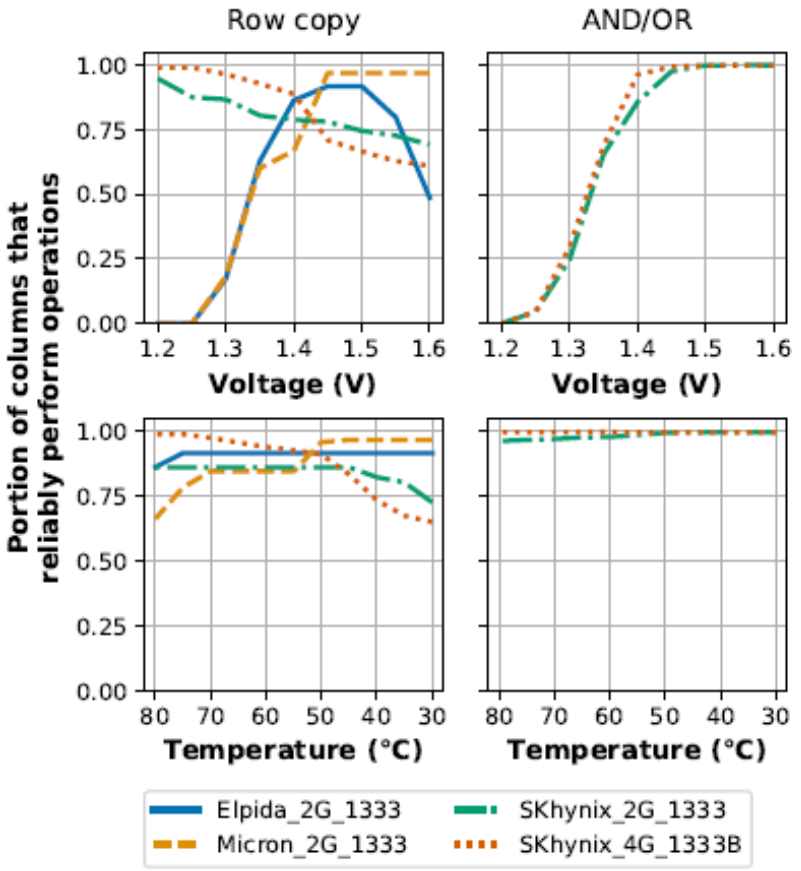
Appendix (I)

Evaluation at varying **voltage** and **temperature** of DRAM groups that had good results

- Using the **ratio of columns that reliably perform operations**
- Separate graphs for each voltage and temperature, row copy and AND/OR pair

AND/OR

- Degradation of operation reliability with lower voltage or higher temperature



Appendix (II)

Robustness of Operations

- CDF – Cumulative distribution function
- Performed using nominal supply voltage (1.5V) and room temperature (25-30C)

- i.e. how many columns have at least a success ratio of 0.5 or more for Micron_2G_1333
  - ~95%
  - The horizontal line means that the columns stay at 95% from almost a success ratio of 1.0, meaning that 5% of the columns are always unable to perform row copy