

Flash-Cosmos

In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory

Jisung Park, Roknoddin Azizi, Geraldo F. Oliveira,
Mohammad Sadrosadati, Rakesh Nadig, David Novo, Juan Gómez Luna,
Myungsook Kim, and Onur Mutlu

Published at MICRO 2022

Rakesh Nadig
P&S SSD Course
26th October 2022



Executive Summary

- **Background:** Bulk bitwise operations are widely used in many important data-intensive applications, e.g., databases, graph processing, cryptography etc.
- **Problem:**
 - Performance and energy efficiency of bulk bitwise operations are bottlenecked by
 - 1) data movement between storage and the compute unit in traditional systems
 - 2) data sensing (serial reading of operands) in prior in-flash processing (IFP) techniques
 - Prior IFP techniques provide low reliability during computation
- **Goal:** Improve performance, energy efficiency and reliability of bulk bitwise operations in in-flash processing
- **Key Idea:** Flash-Cosmos (Flash-Computation with One-Shot Multi-Operand Sensing) is an in-flash processing technique that improves performance, energy efficiency and reliability of bulk bitwise operations using two key techniques:
 - Multi-Wordline Sensing (MWS): Enables multi-operand bulk bitwise operations with a single sensing (read) operation
 - Enhanced SLC-mode Programming (ESP): Increases the voltage margin between the erased and programmed states to provide higher reliability during in-flash computation
- **Key Results:** Flash-Cosmos is evaluated using 160 real 3D NAND flash chips and three real-world workloads
 - Flash-Cosmos improves the performance and energy efficiency by 3.5x and 3.3x over state-of-the-art IFP technique while providing high reliability during computation

Talk Outline

Motivation

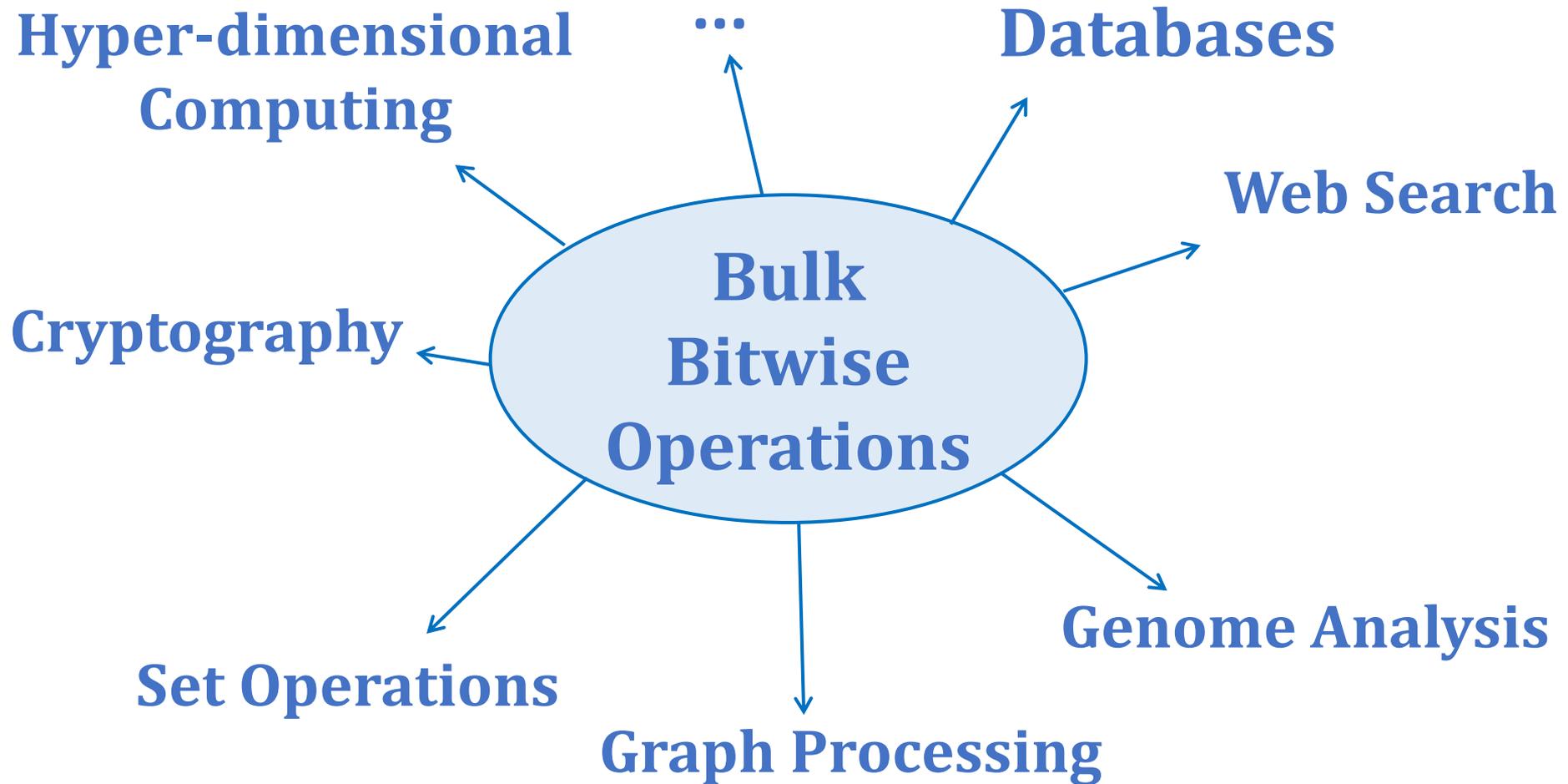
Background

Flash-Cosmos

Evaluation

Summary

Bulk Bitwise Operations



Bulk Bitwise Operations

Hyper-dimensional
Computing

...

Databases
(database queries
and indexing)

Data movement between compute units
and the memory hierarchy significantly affects
the performance of bulk bitwise operations

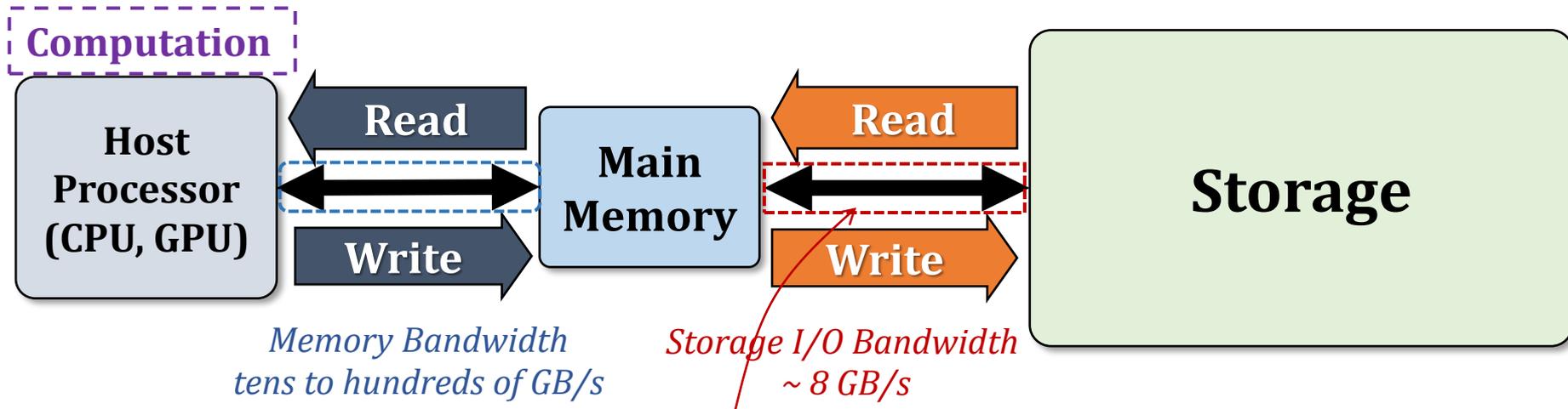
Set Operations

Graph Processing

Genome Analysis

Data-Movement Bottleneck

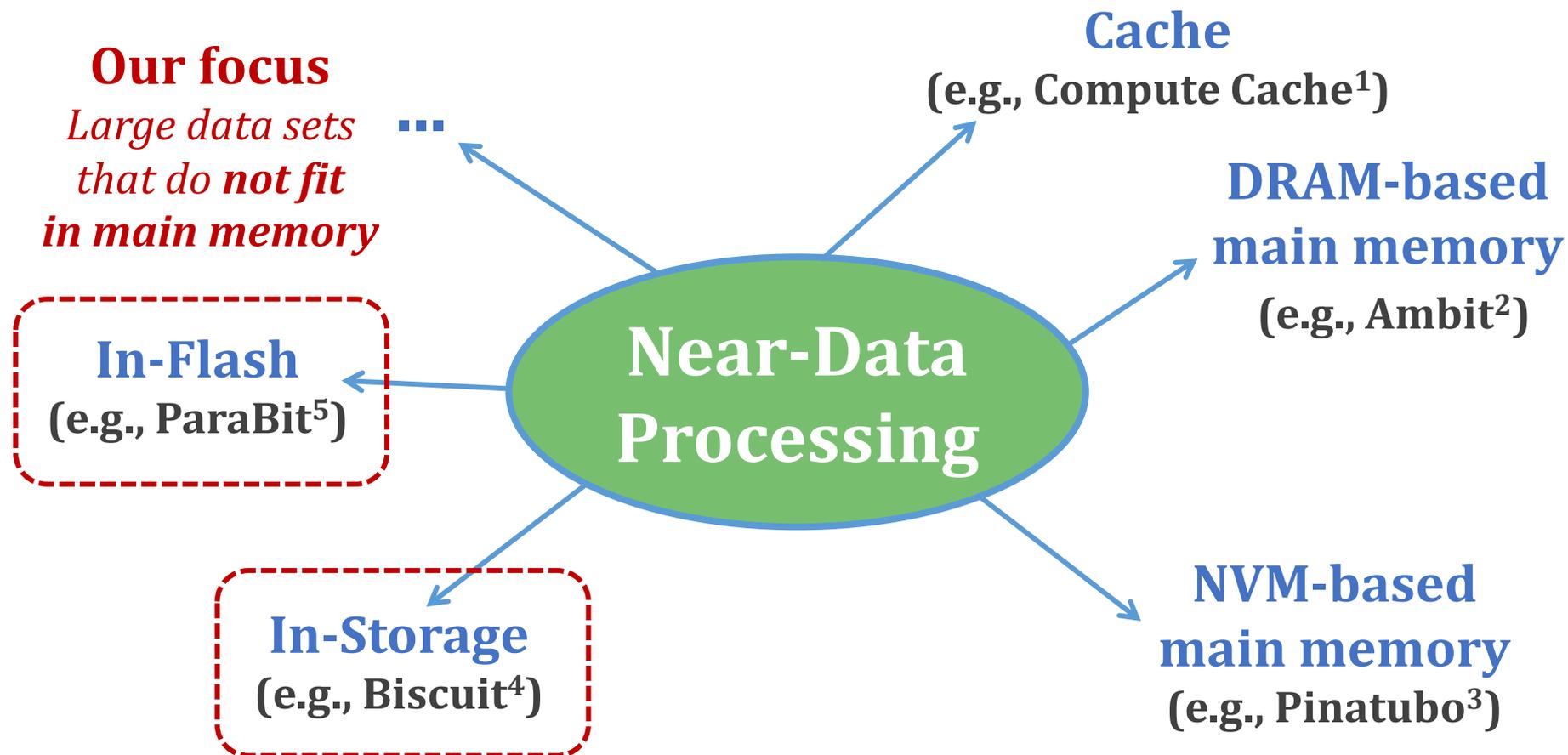
- Conventional systems perform outside-storage processing (OSP) after moving the data to host CPU through the memory hierarchy



Data Movement Bottleneck

The external I/O bandwidth of storage is the main bottleneck for data movement in OSP

NDP for Bulk Bitwise Operations



[1] Aga+, "Compute Caches," HPCA, 2017

[2] Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," MICRO, 2017

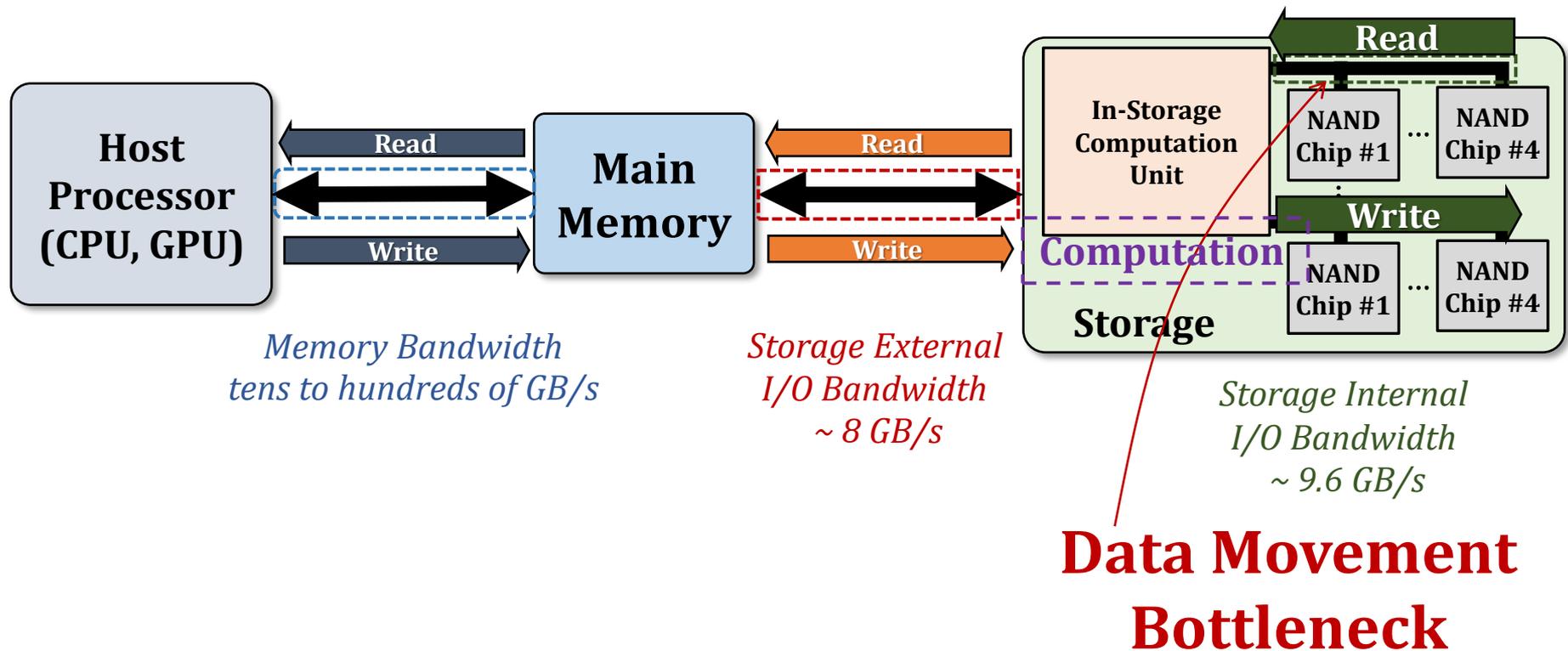
[3] Li+, "Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-Volatile Memories," DAC, 2016

[4] Gu+, "Biscuit: A Framework for Near-Data Processing of Big Data Workloads," ISCA, 2016

[5] Gao+, "ParaBit: Processing Parallel Bitwise Operations in NAND Flash Memory Based SSDs," MICRO, 2021

In-Storage Processing (ISP)

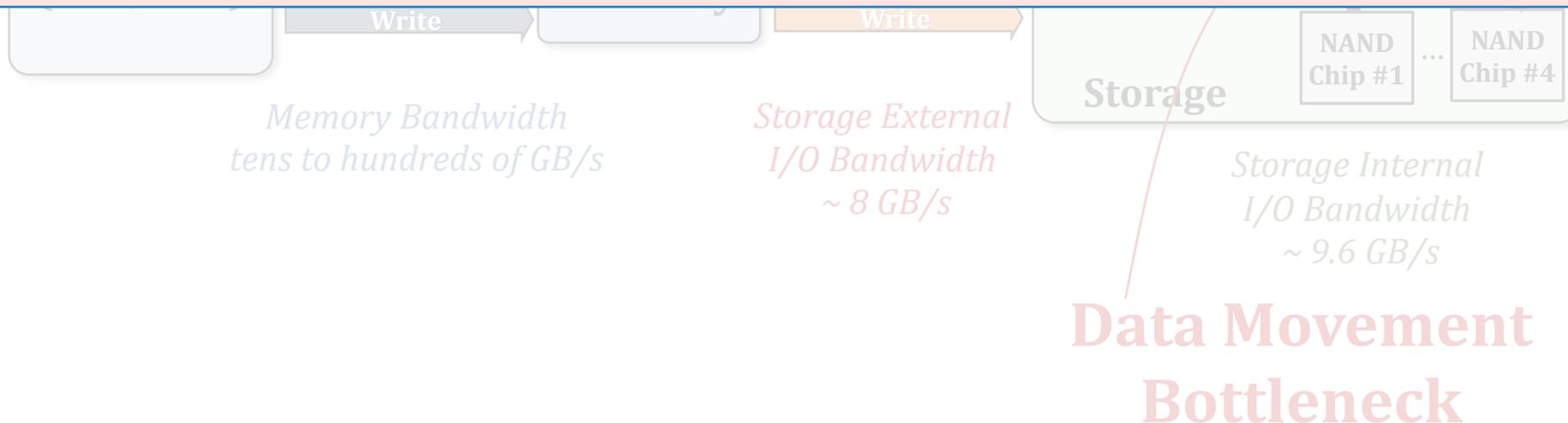
- ISP performs computation using an in-storage computation unit
- ISP reduces external data movement by transferring only the computation results to the host



In-Storage Processing (ISP)

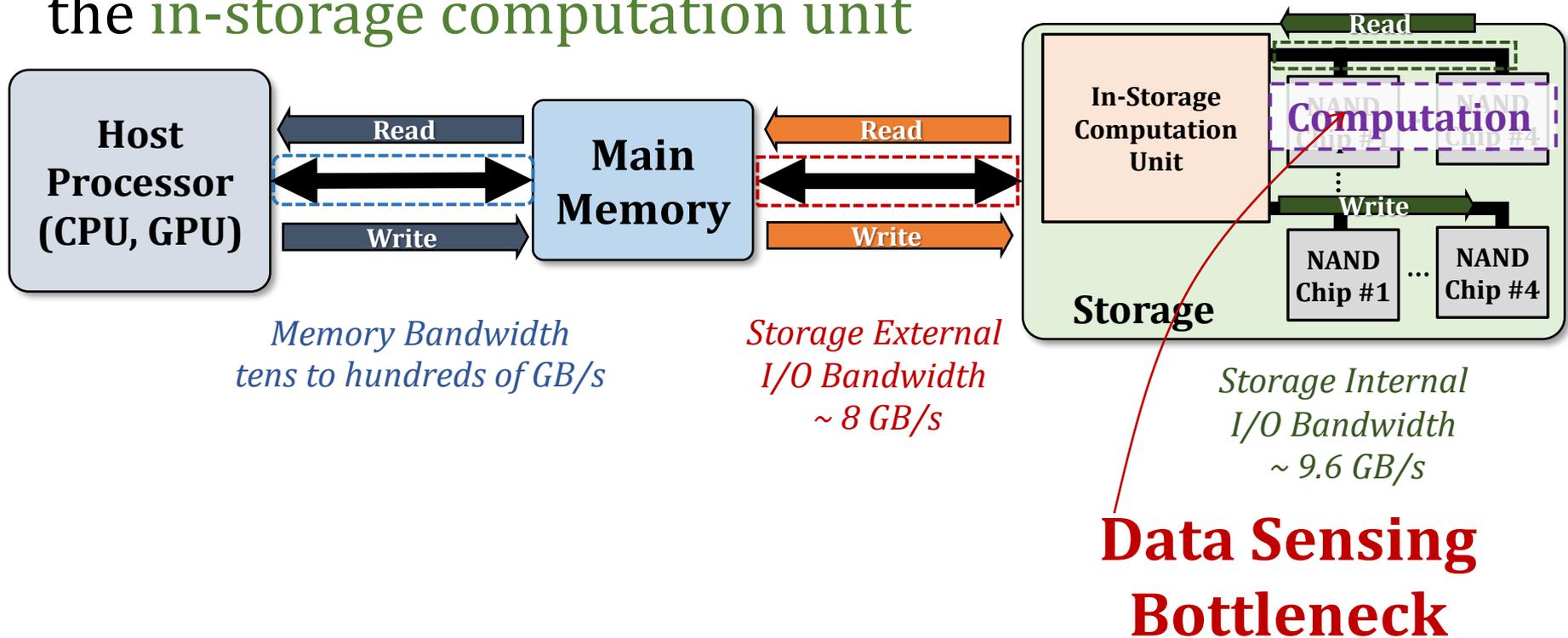
- ISP performs computation using the in-storage computation unit
- ISP reduces external data movement by transferring only the computation results to the host

Storage internal I/O bandwidth is the main bottleneck for data movement in ISP



In-Flash Processing (IFP)

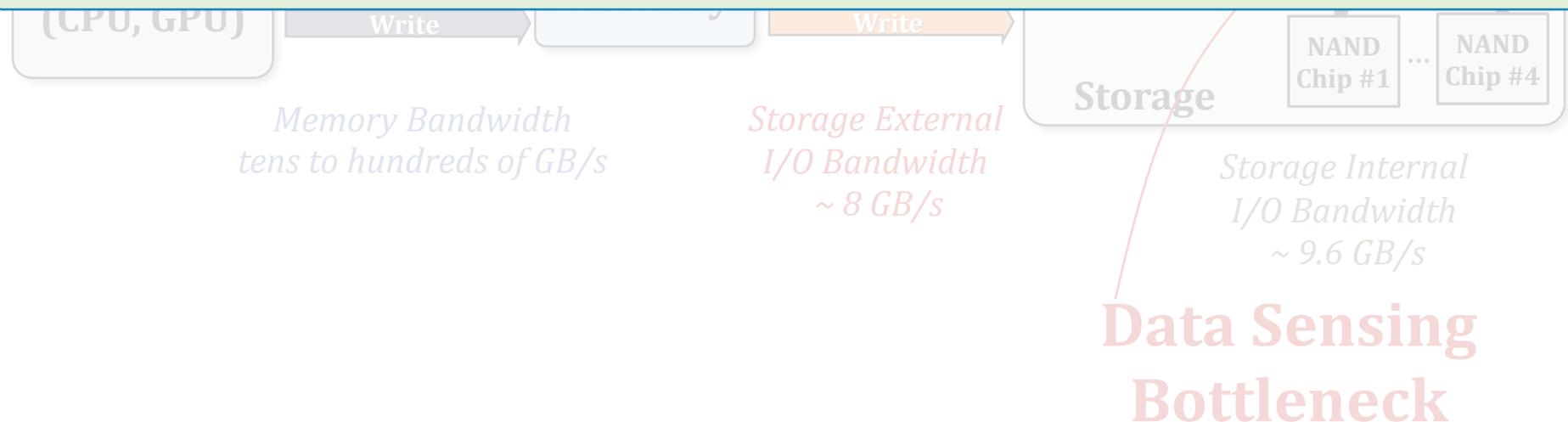
- IFP performs computation within the flash chips as the data operands are being read serially
- IFP reduces the internal data movement bottleneck in storage by transferring only the computation results to the in-storage computation unit



In-Flash Processing (IFP)

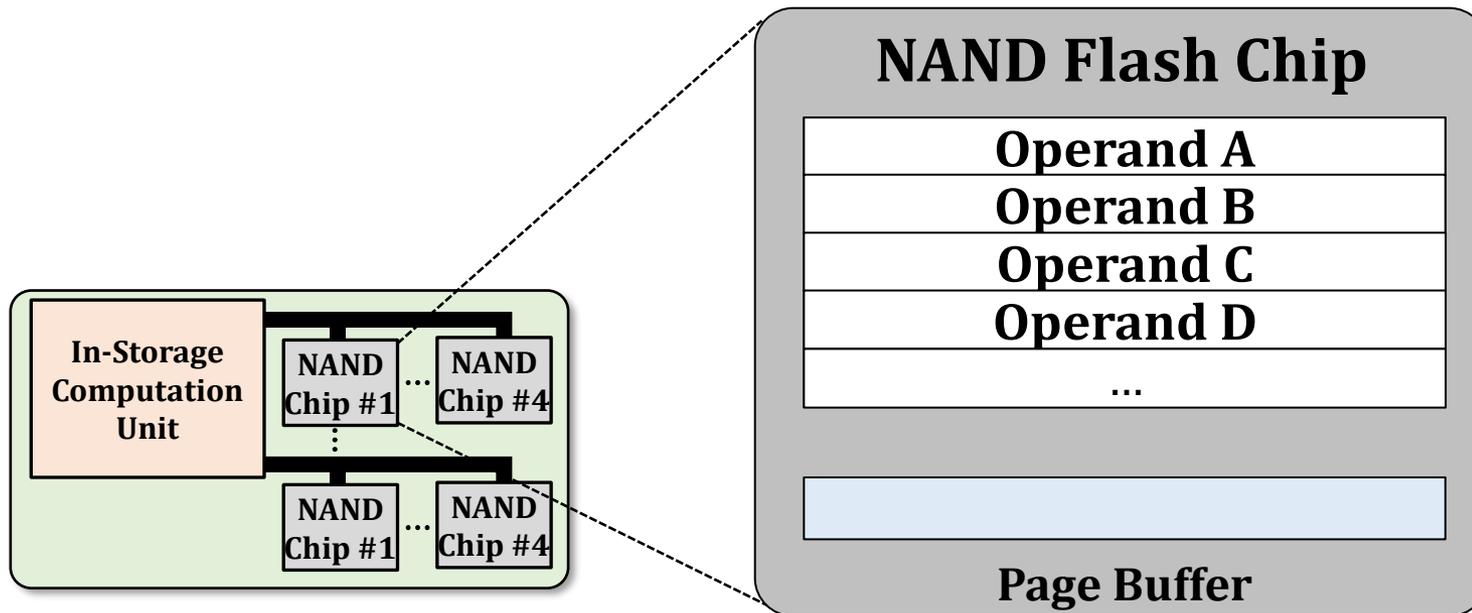
- IFP performs computation within the flash chips as the data operands are being read serially
- IFP reduces the internal data movement bottleneck in storage by transferring only the computation results to the in-storage computation unit

IFP fundamentally mitigates the data movement



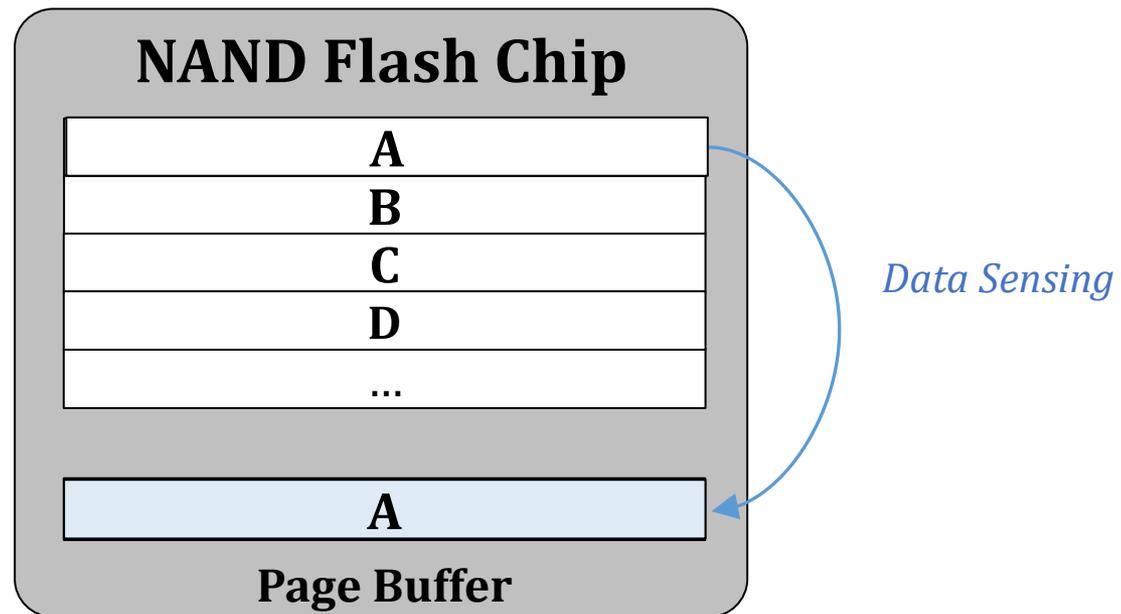
Data Sensing Bottleneck in IFP

- State-of-the-art IFP technique ^[1] performs bulk bitwise operations by controlling the latching circuit of the page buffer



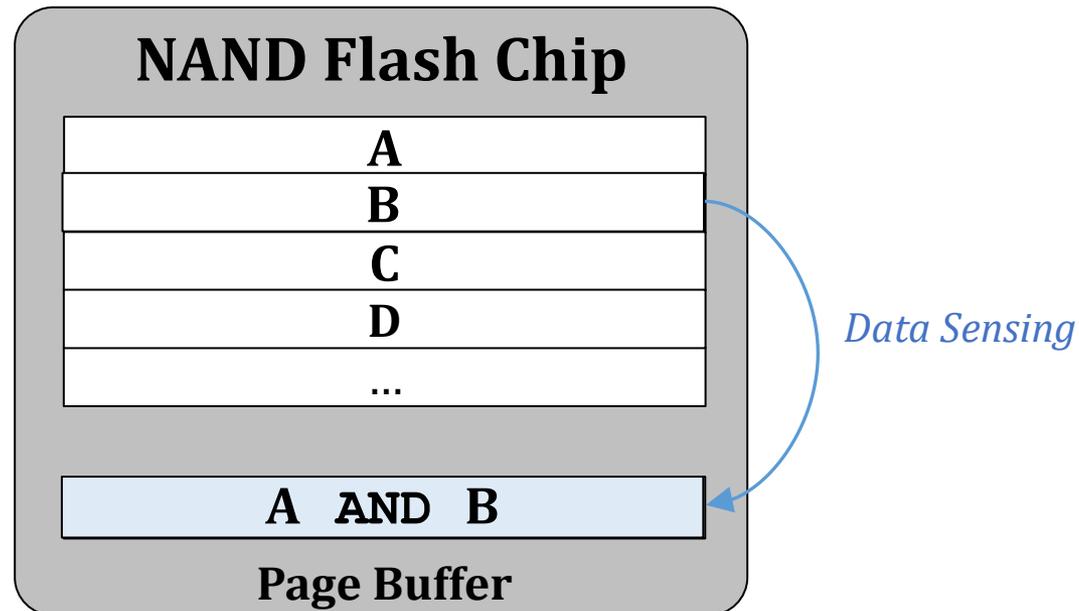
Data Sensing Bottleneck in IFP

- State-of-the-art IFP technique ^[1] performs bulk bitwise operations by controlling the latching circuit of the page buffer



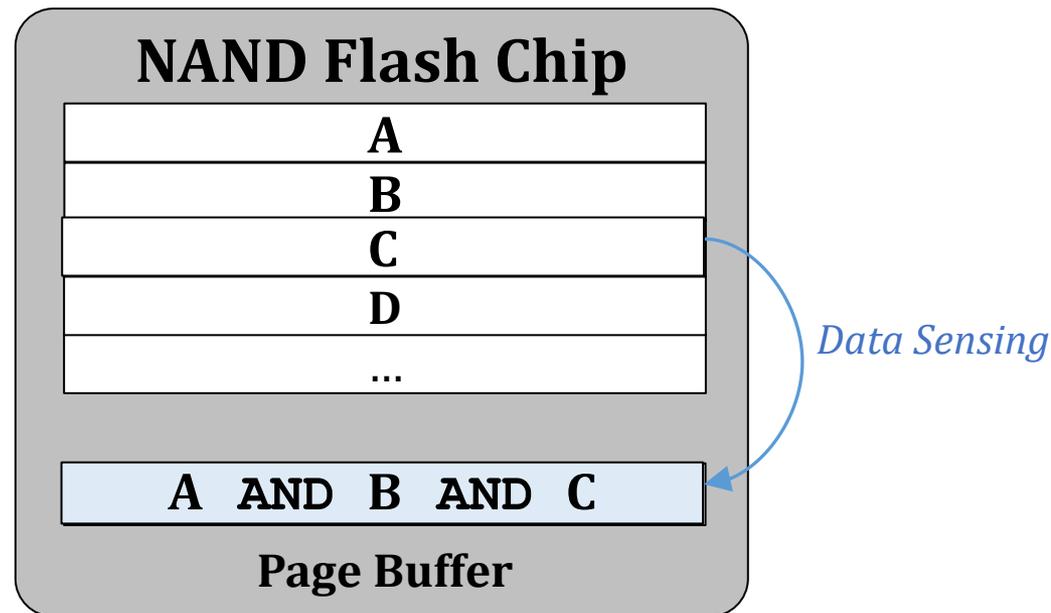
Data Sensing Bottleneck in IFP

- State-of-the-art IFP technique ^[1] performs bulk bitwise operations by controlling the latching circuit of the page buffer



Data Sensing Bottleneck in IFP

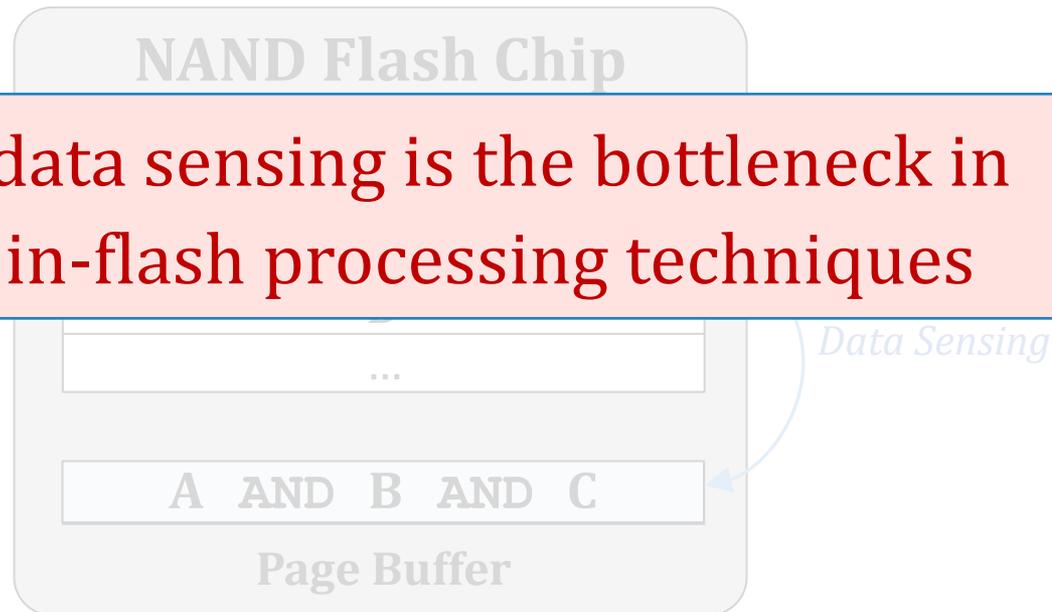
- State-of-the-art IFP technique ^[1] performs bulk bitwise operations by controlling the latching circuit of the page buffer



Data Sensing Bottleneck in IFP

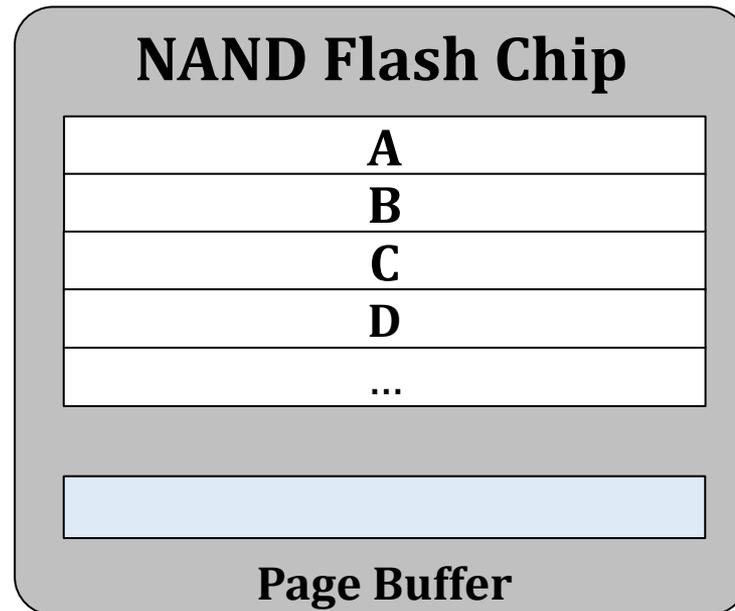
- State-of-the-art IFP technique ^[1] performs bulk bitwise operations by controlling the latching circuit of the page buffer

Serial data sensing is the bottleneck in prior in-flash processing techniques



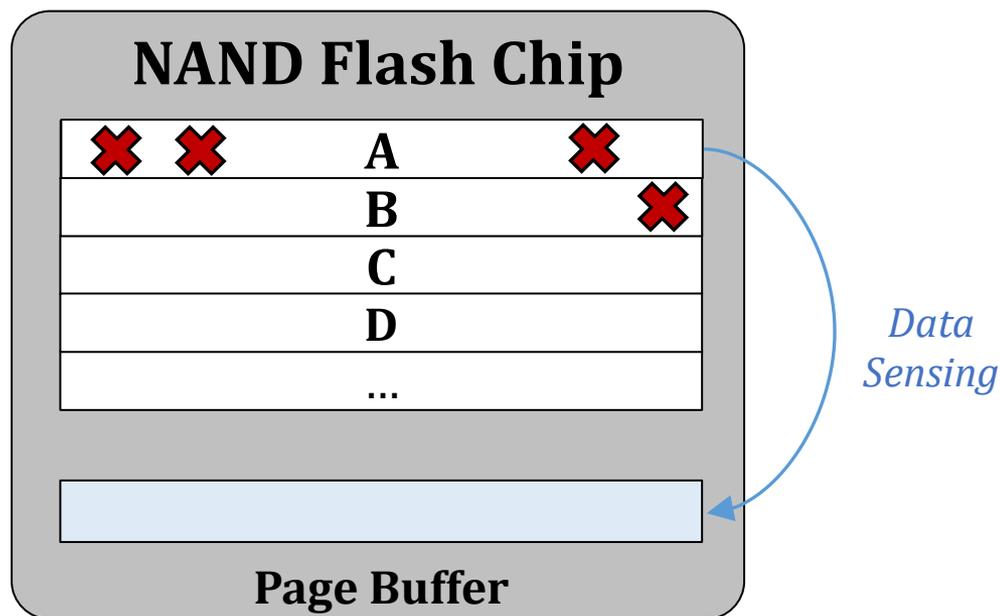
Reliability Issues in IFP

- Prior IFP approaches cannot leverage ECC and data-randomization techniques as computation is performed within the flash chips during data sensing



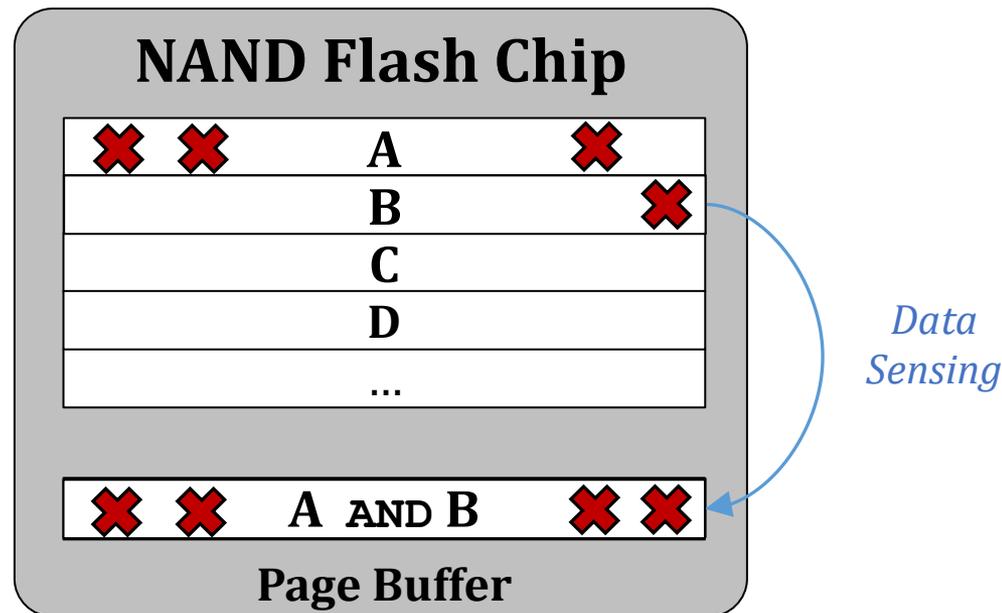
Reliability Issues in IFP

- Prior IFP approaches cannot leverage ECC and data-randomization techniques as computation is performed within the flash chips during data sensing



Reliability Issues in IFP

- Prior IFP approaches cannot leverage ECC and data-randomization techniques as computation is performed within the flash chips during data sensing



Reliability Issues in IFP

- Prior IFP approaches cannot leverage ECC and data-randomization techniques as computation is performed within the flash chips during data sensing

NAND Flash Chip

Prior IFP techniques requires the application to be highly error-tolerant

✘ ✘ A AND B ✘ ✘

Page Buffer

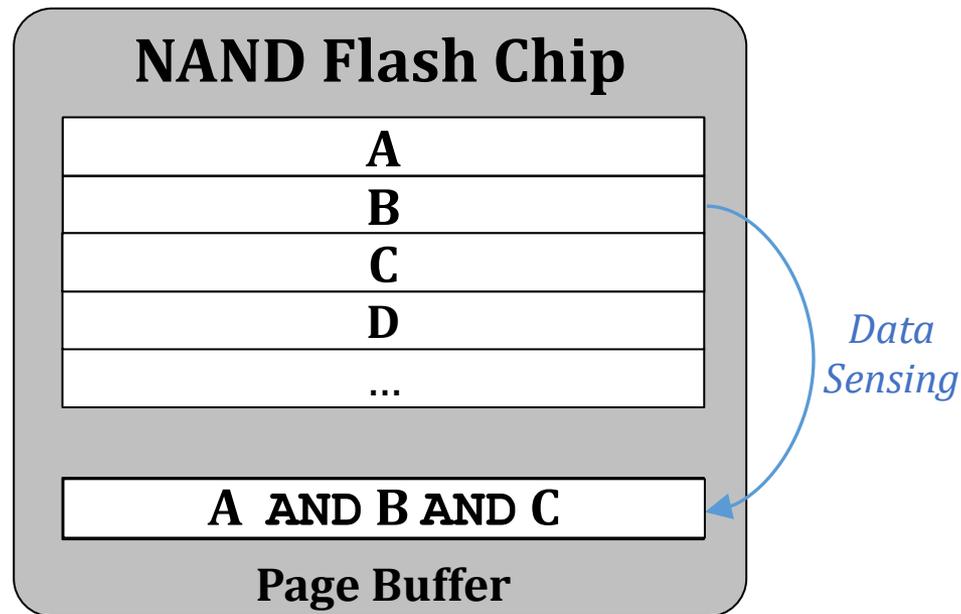
Our Goal

Address the bottleneck of state-of-the-art IFP techniques
(serial sensing of operands)

Make IFP reliable
(provide accurate computation results)

Our Proposal

- Flash-Cosmos enables
 - Computation on multiple operands using a single sensing operation
 - Provide high reliability during in-flash computation



Talk Outline

Motivation

Background

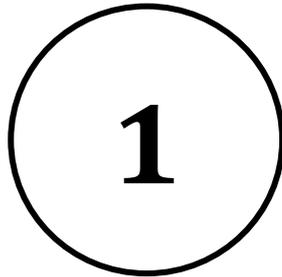
Flash-Cosmos

Evaluation

Summary

NAND Flash Basics: A Flash Cell

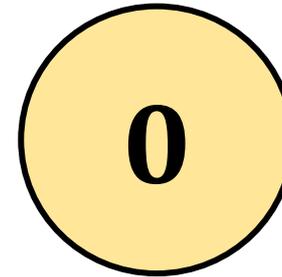
- A flash cell stores data by adjusting the **amount of charge** in the cell



**Erased Cell
(Low Charge Level)**



*Operates as a **resistor***



**Programmed Cell
(High Charge Level)**

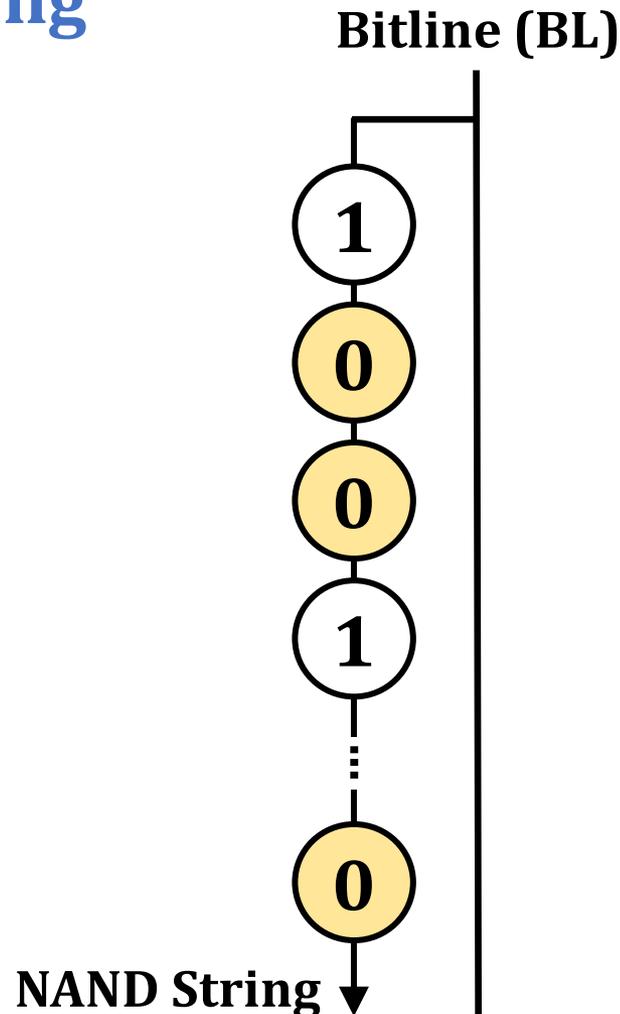


*Operates as an **open switch***

Activation

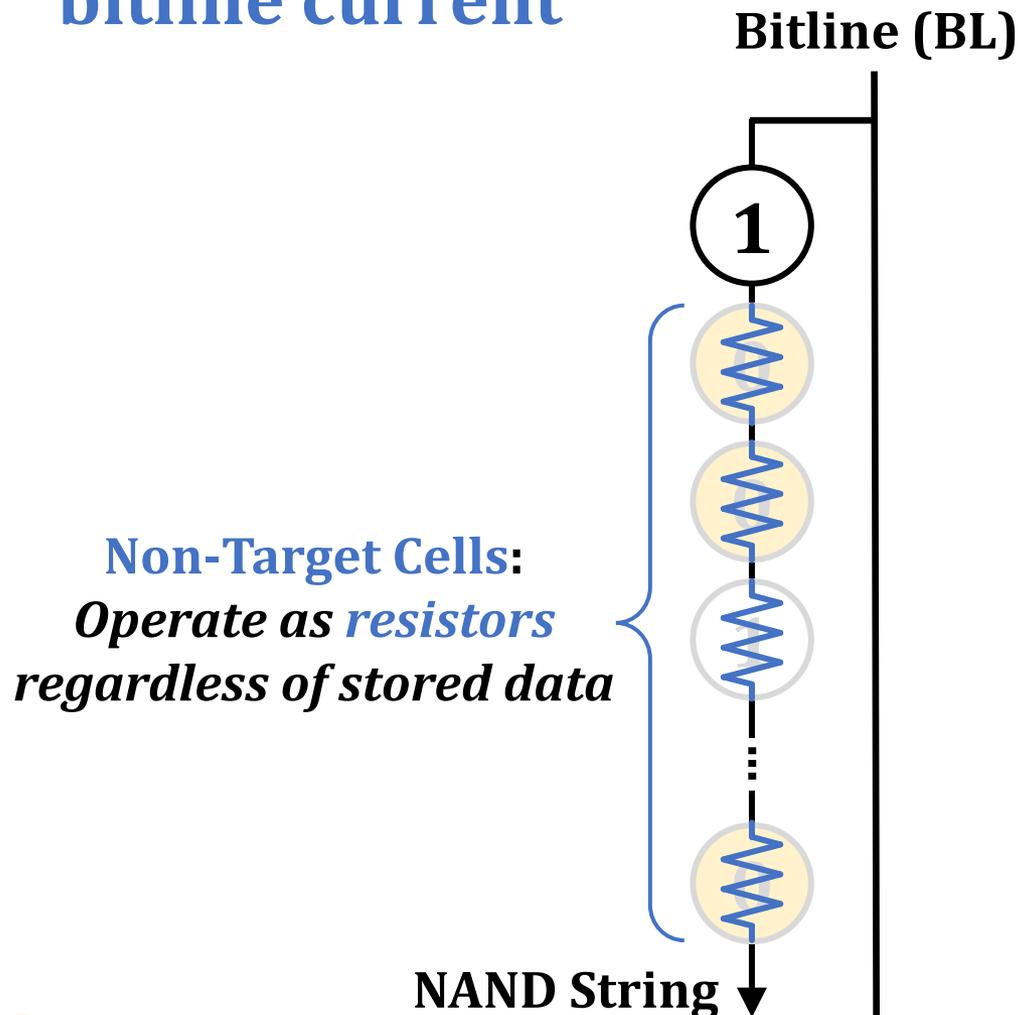
NAND Flash Basics: A NAND String

- A set of flash cells are **serially connected** to form a **NAND String**



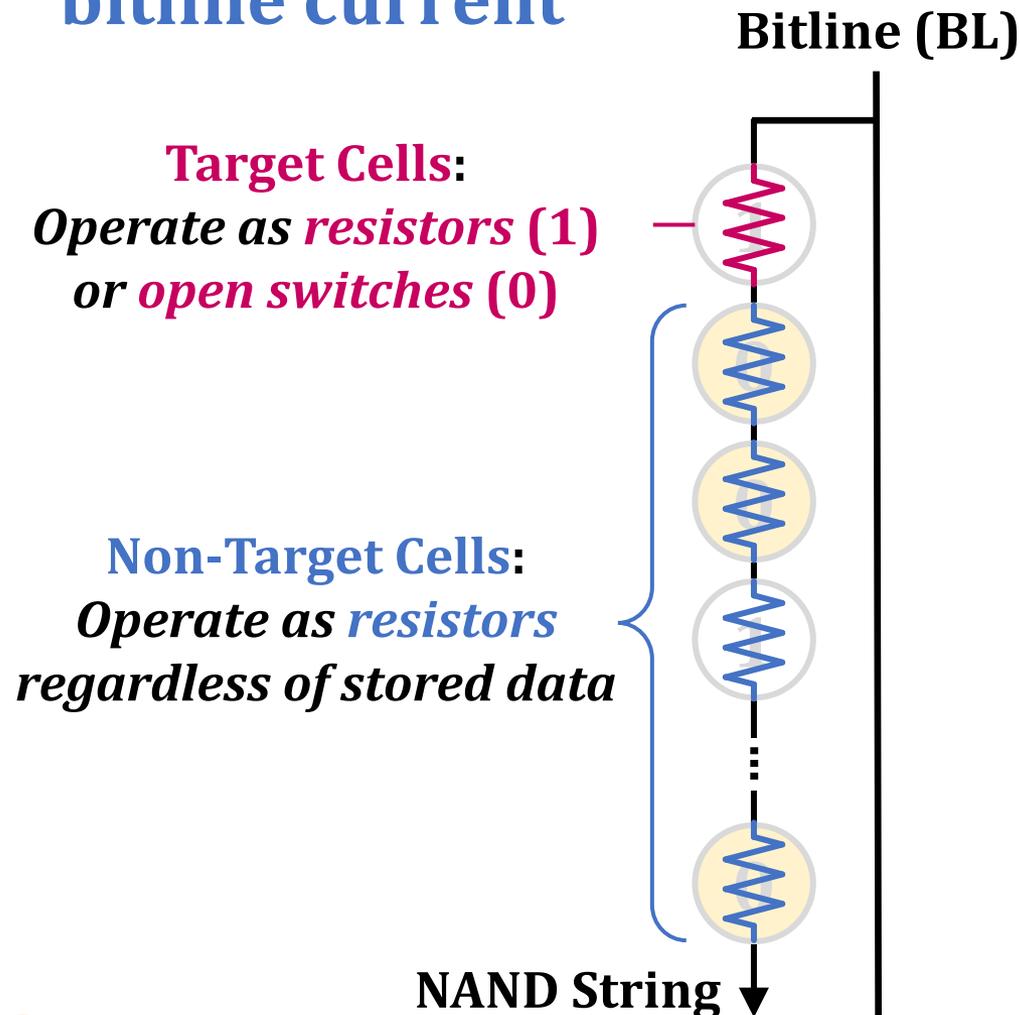
NAND Flash Basics: Read Mechanism

- NAND flash memory reads data by **checking the bitline current**



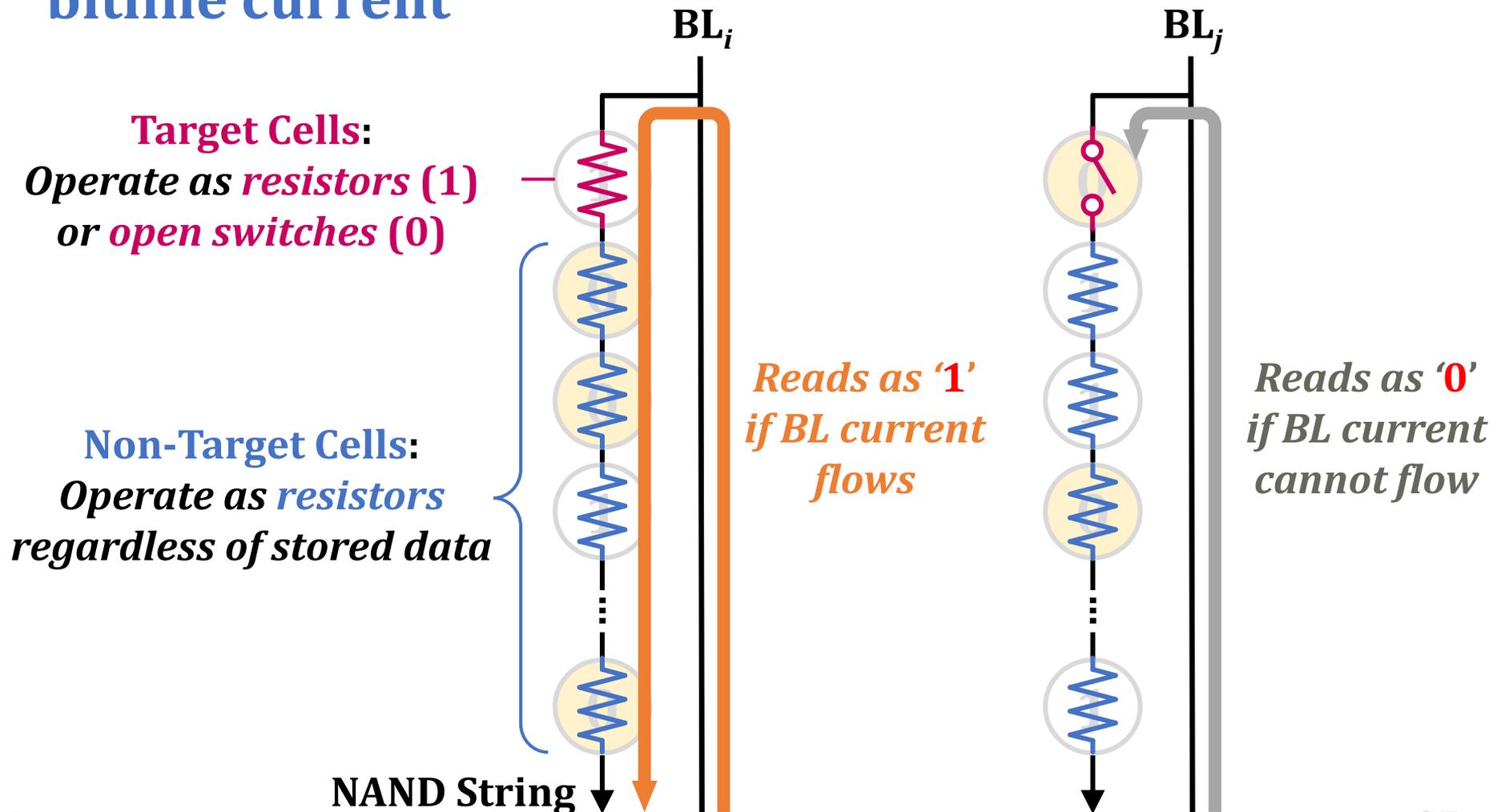
NAND Flash Basics: Read Mechanism

- NAND flash memory reads data by **checking the bitline current**



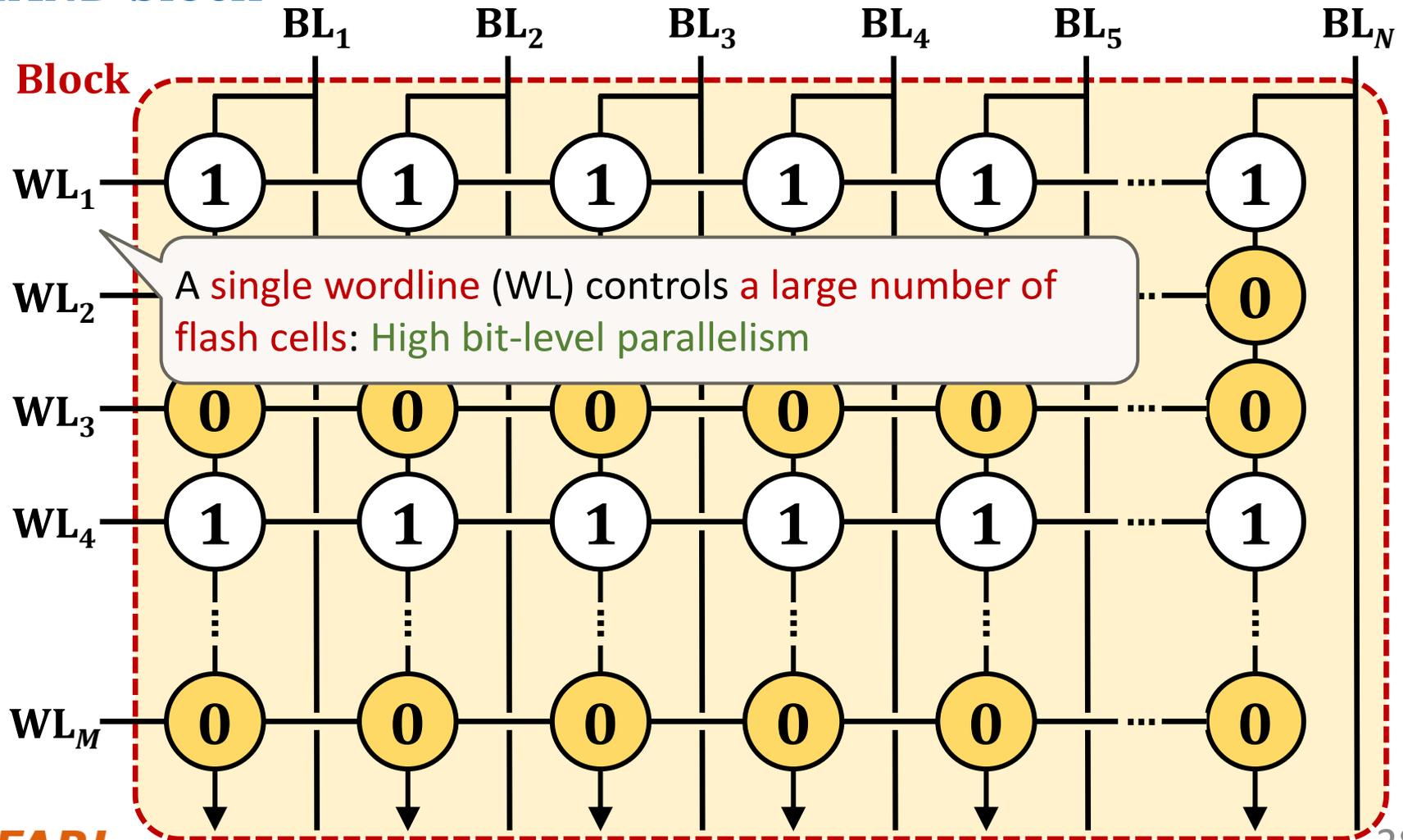
NAND Flash Basics: Read Mechanism

- NAND flash memory reads data by **checking the bitline current**



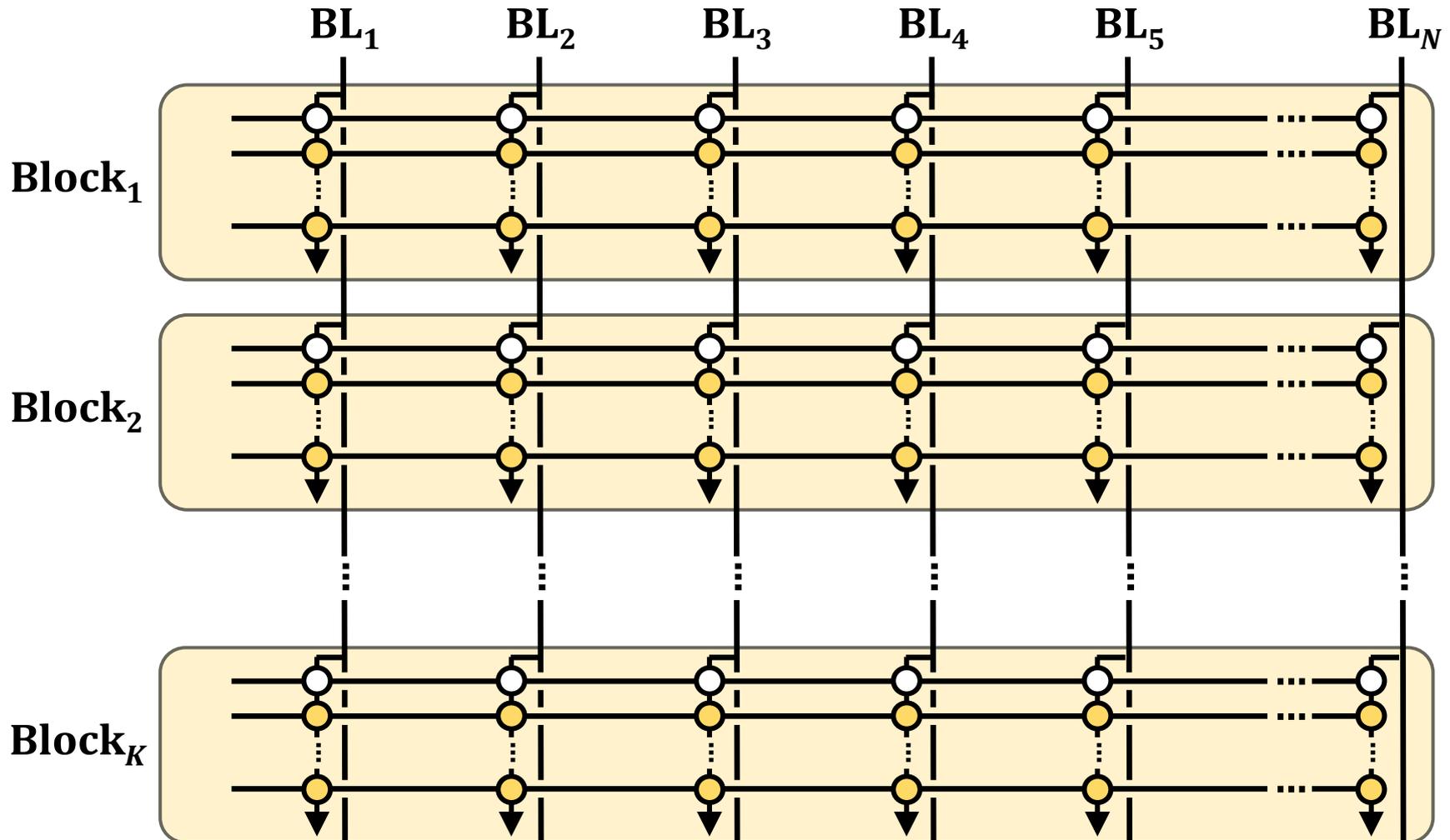
NAND Flash Basics: A NAND Flash Block

- NAND strings connected to different bitlines comprise a **NAND block**



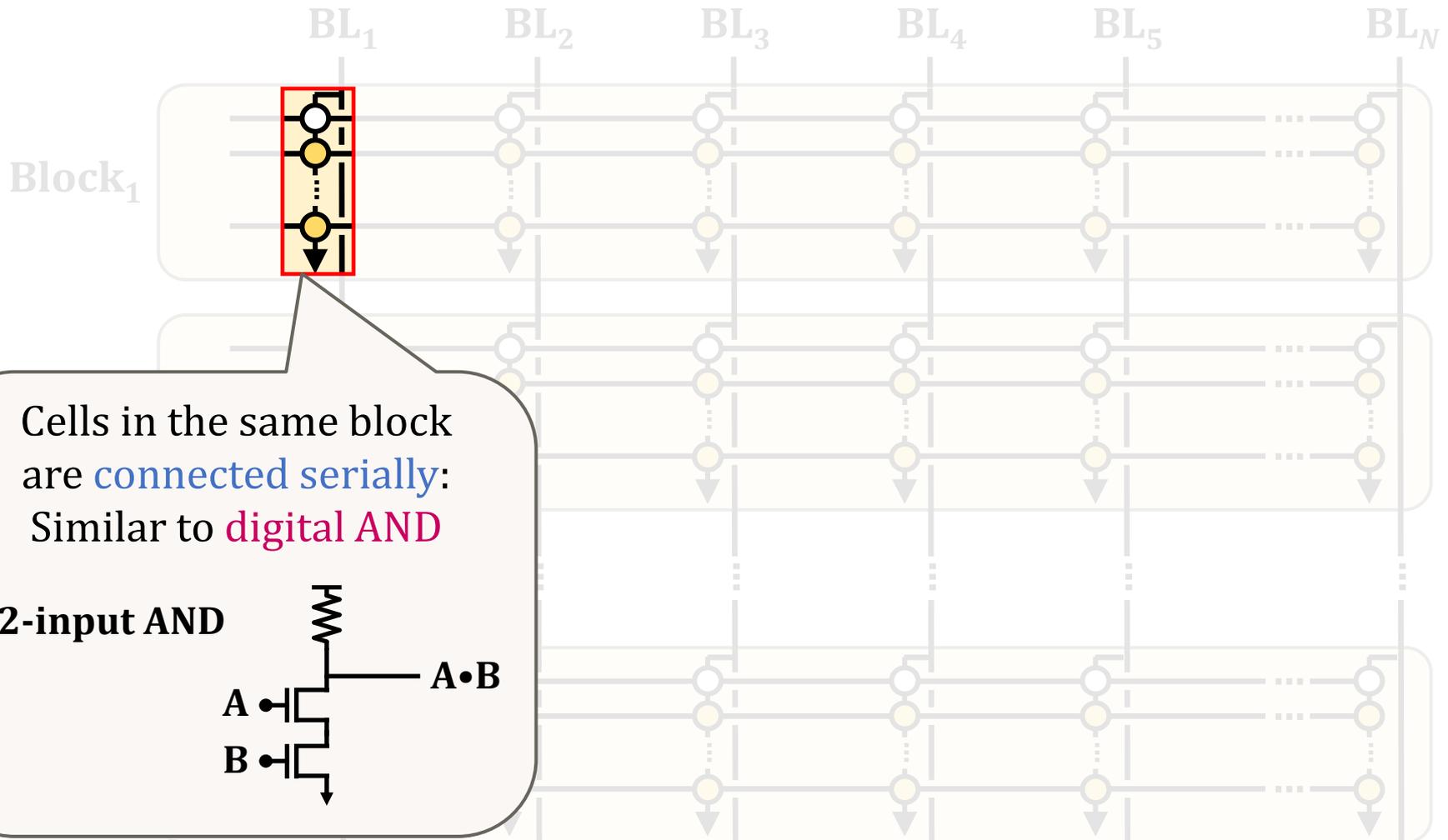
NAND Flash Basics: Block Organization

- A large number of blocks share the same bitlines



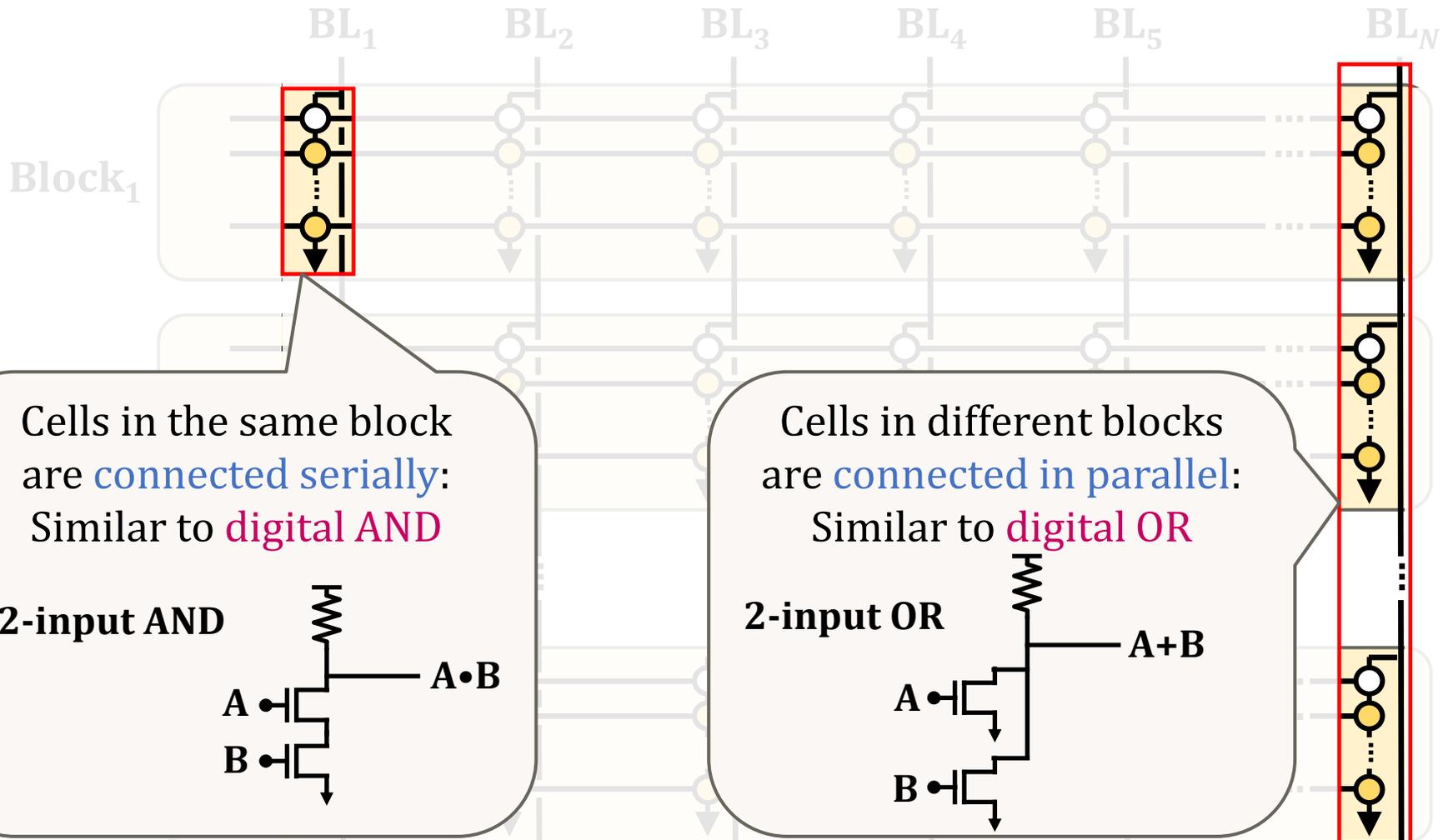
Similarity to Digital Logic Gates

- A large number of blocks share the same bitlines



Similarity to Digital Logic Gates

- A large number of blocks share the same bitlines.



Talk Outline

Motivation

Background

Flash-Cosmos

Evaluation

Summary

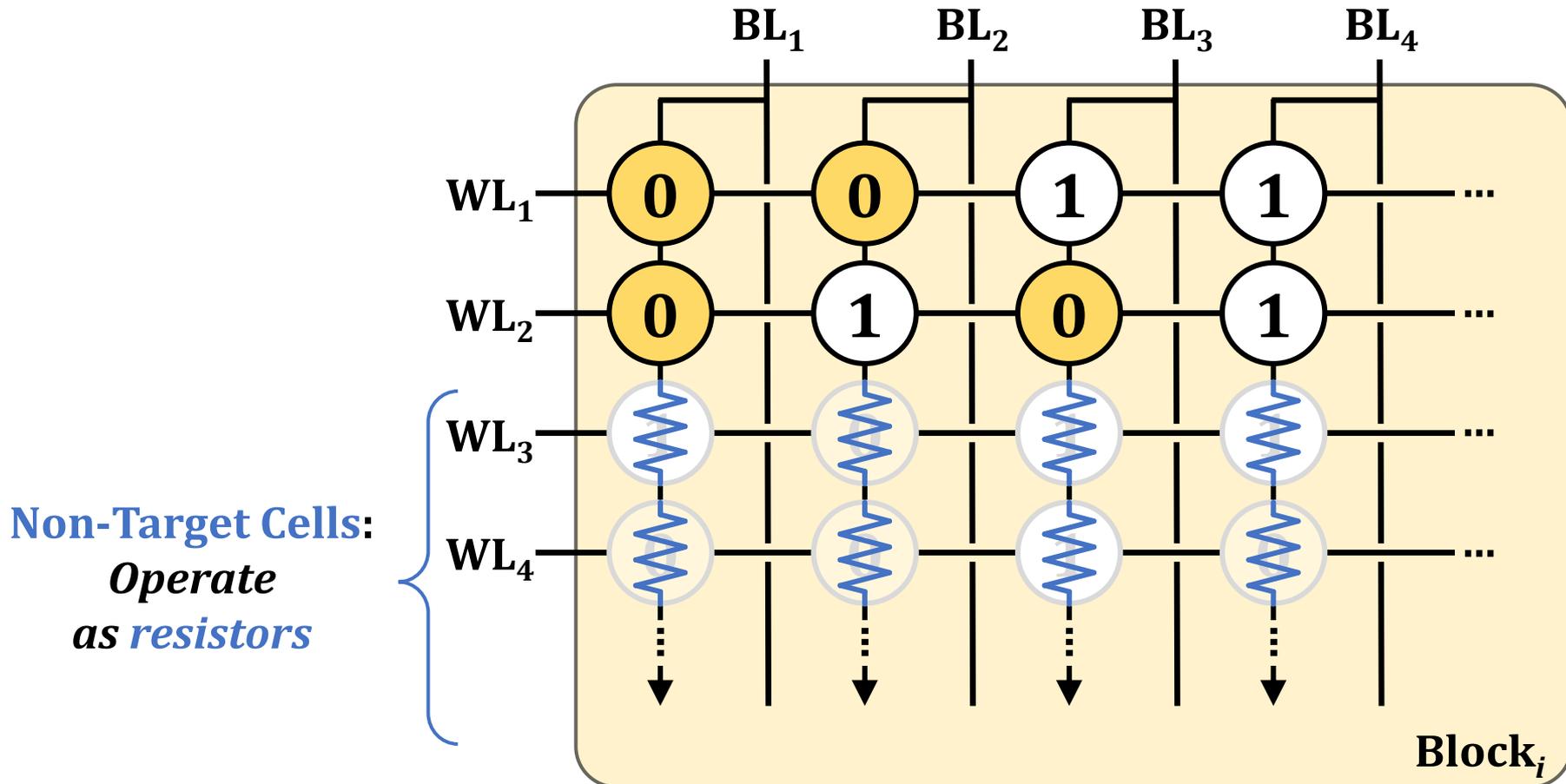
Flash-Cosmos: Overview



Enables in-flash bulk bitwise operations on multiple operands with a *single* sensing operation using Multi-Wordline Sensing (MWS)

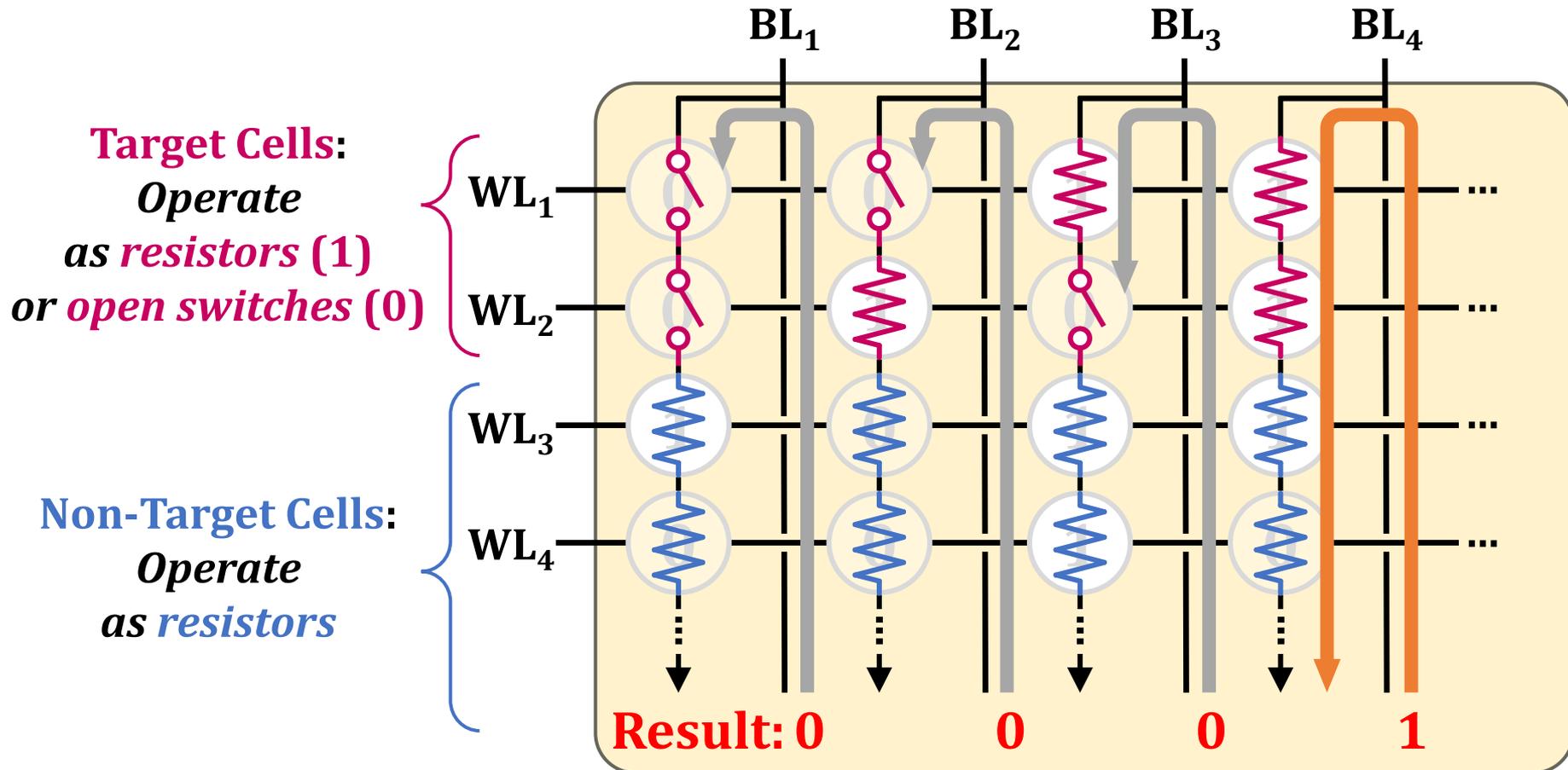
Multi-Wordline Sensing (MWS): Bitwise AND

- **Intra-Block MWS: Simultaneously activates multiple WLs in the same block**
 - **Bitwise AND** of the stored data in the WLs



Multi-Wordline Sensing (MWS): Bitwise AND

- **Intra-Block MWS: Simultaneously activates multiple WLs in the same block**
 - **Bitwise AND** of the stored data in the WLs



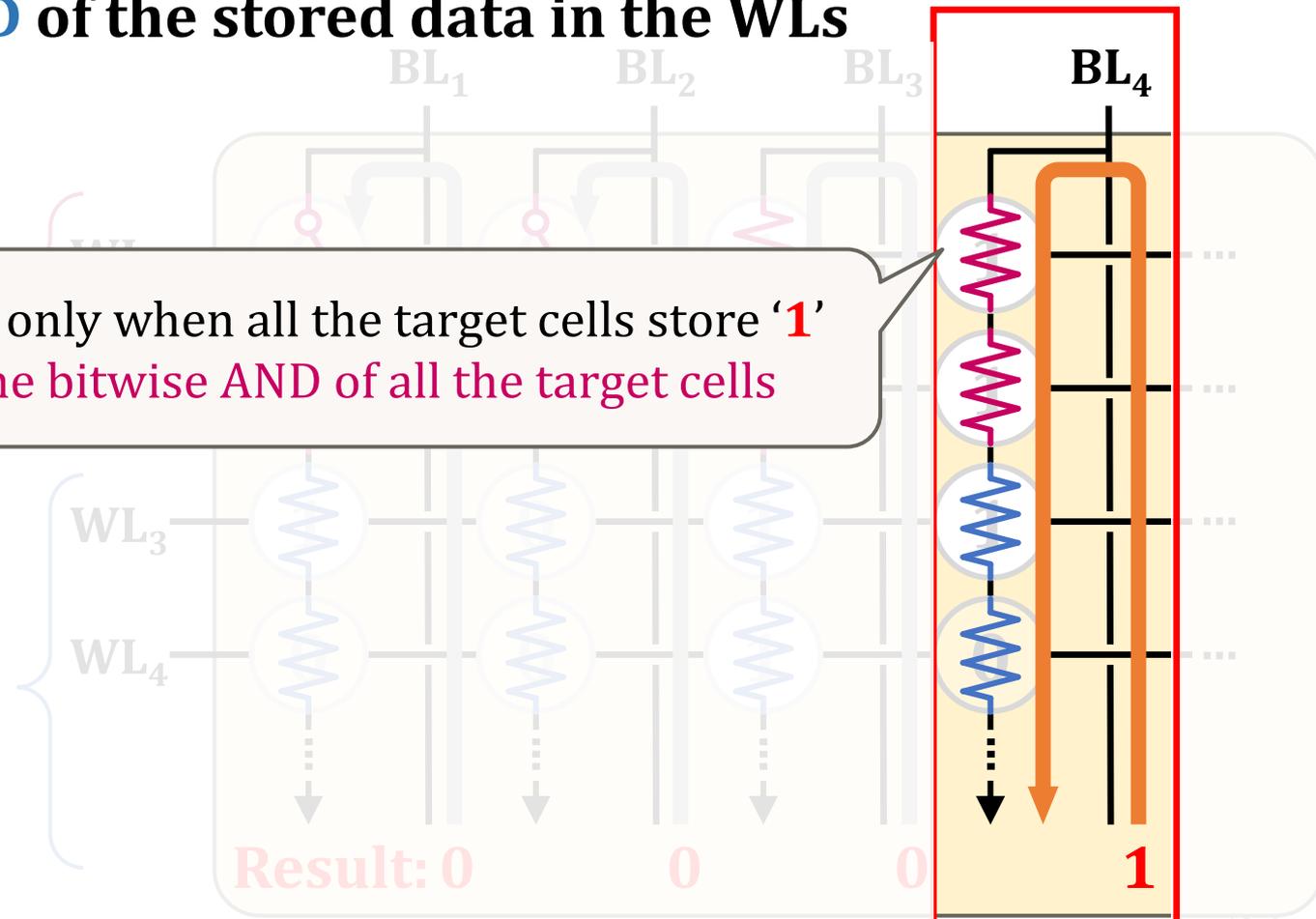
Multi-Wordline Sensing (MWS): Bitwise AND

- **Intra-Block MWS: Simultaneously activates multiple WLs in the same block**
 - **Bitwise AND** of the stored data in the WLs

Target Cell:

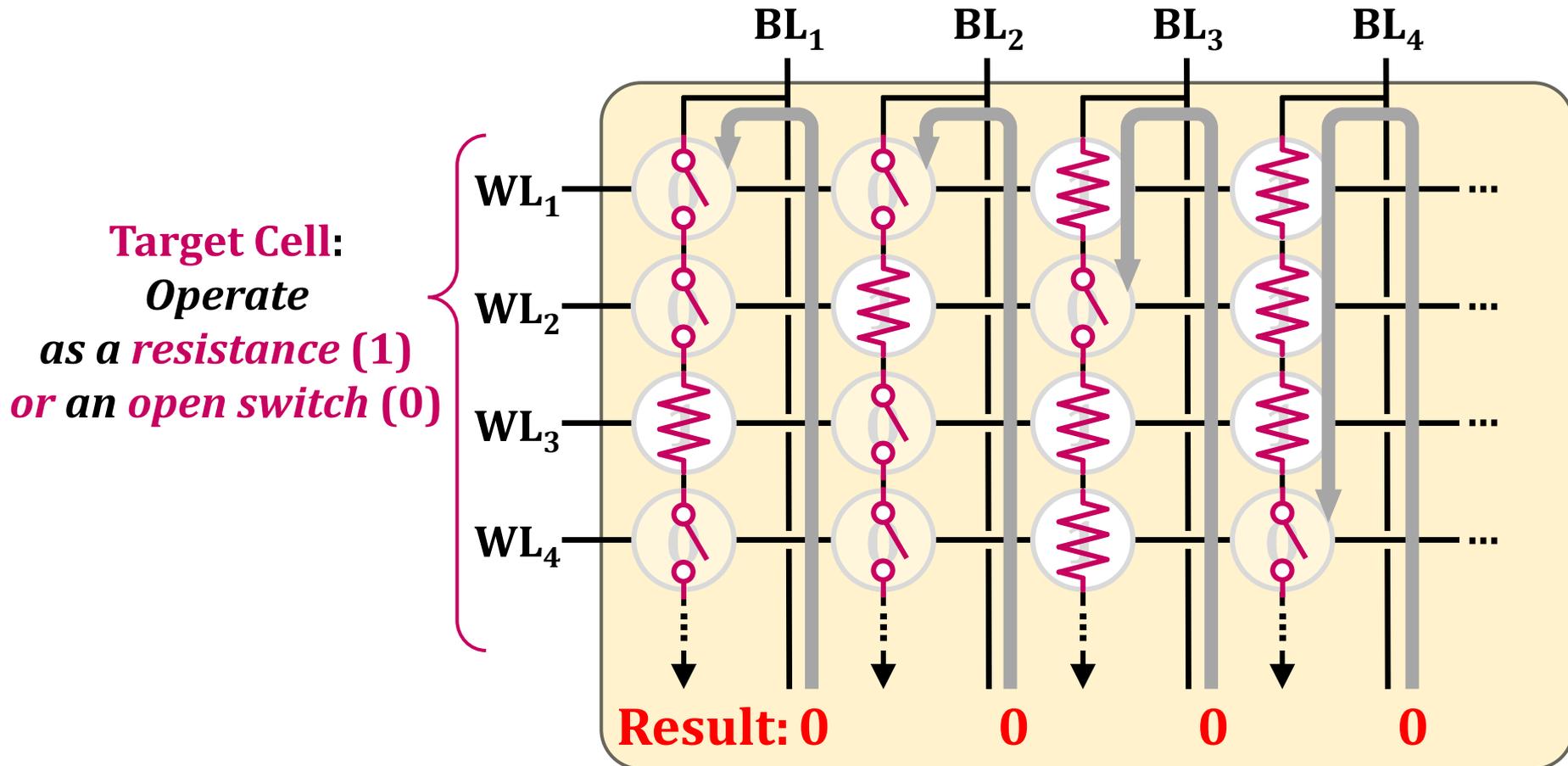
A bitline reads as '1' only when all the target cells store '1'
→ Equivalent to the bitwise AND of all the target cells

Non-Target Cell:
Operate as a resistance



Multi-Wordline Sensing (MWS): Bitwise AND

- **Intra-Block MWS: Simultaneously activates multiple WLs in the same block**
 - **Bitwise AND** of the stored data in the WLs

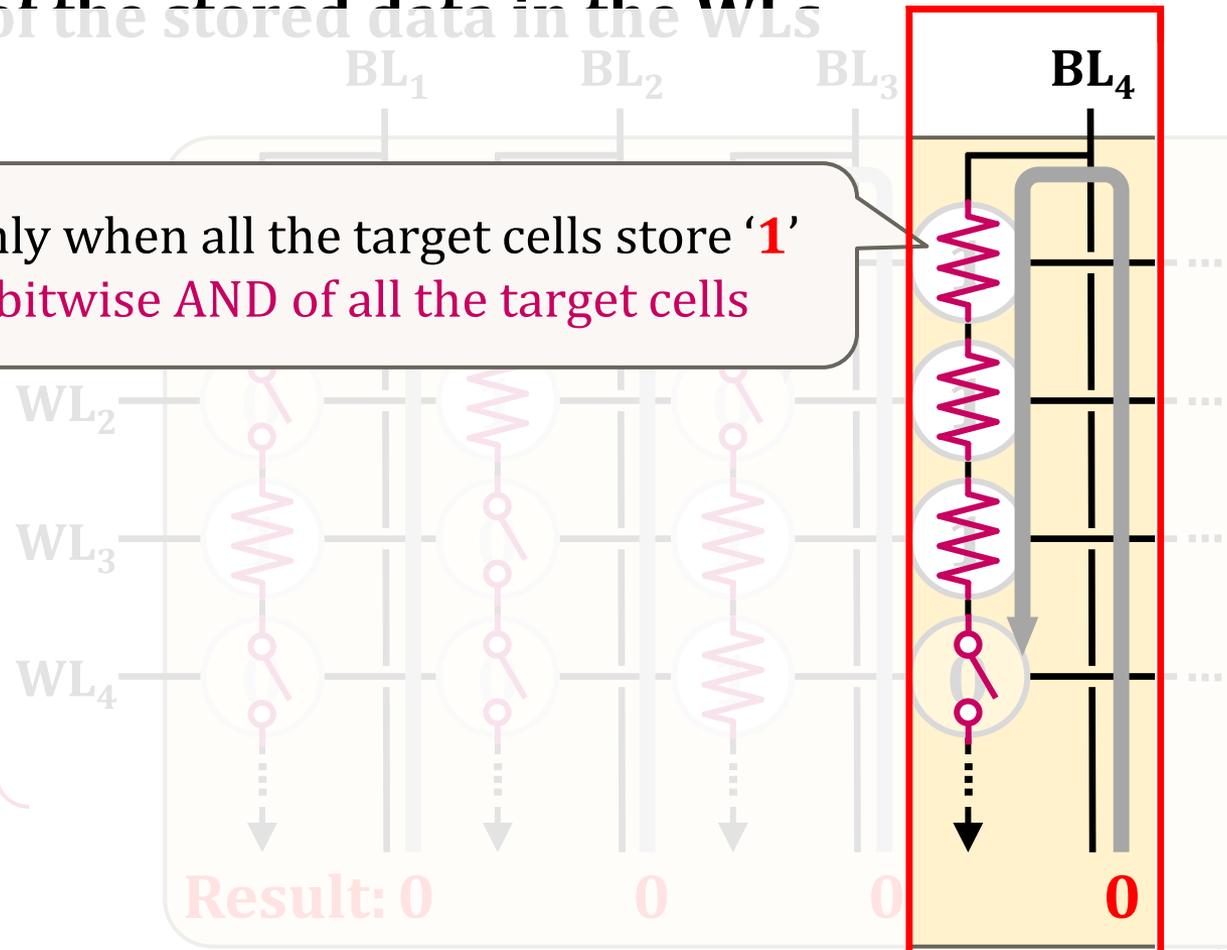


Multi-Wordline Sensing (MWS): Bitwise AND

- **Intra-Block MWS: Simultaneously activates multiple WLs in the same block**
 - Bitwise AND of the stored data in the WLs

A bitline reads as '1' only when all the target cells store '1'
→ Equivalent to the bitwise AND of all the target cells

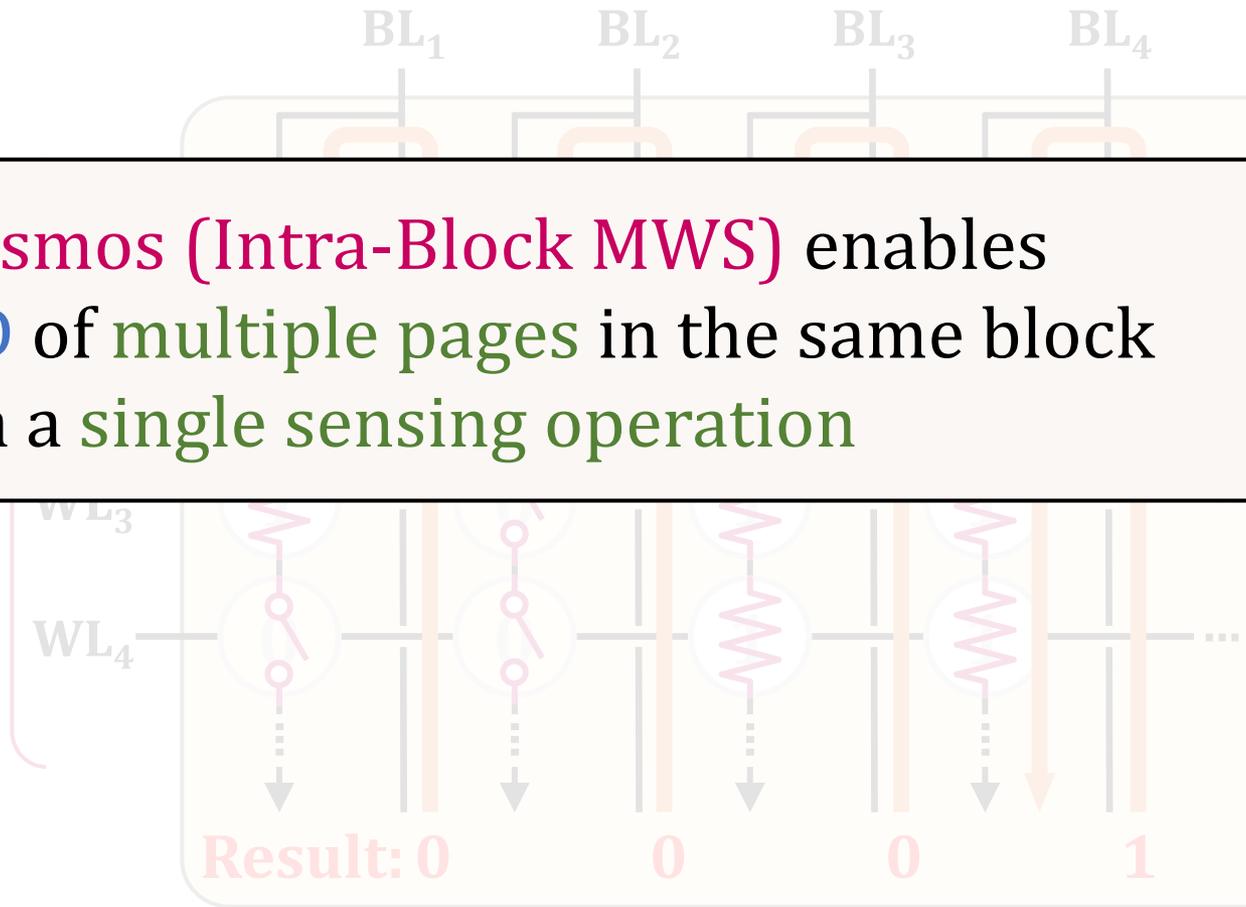
Operate
as a *resistance (1)*
or an *open switch (0)*



Multi-Wordline Sensing (MWS): Bitwise AND

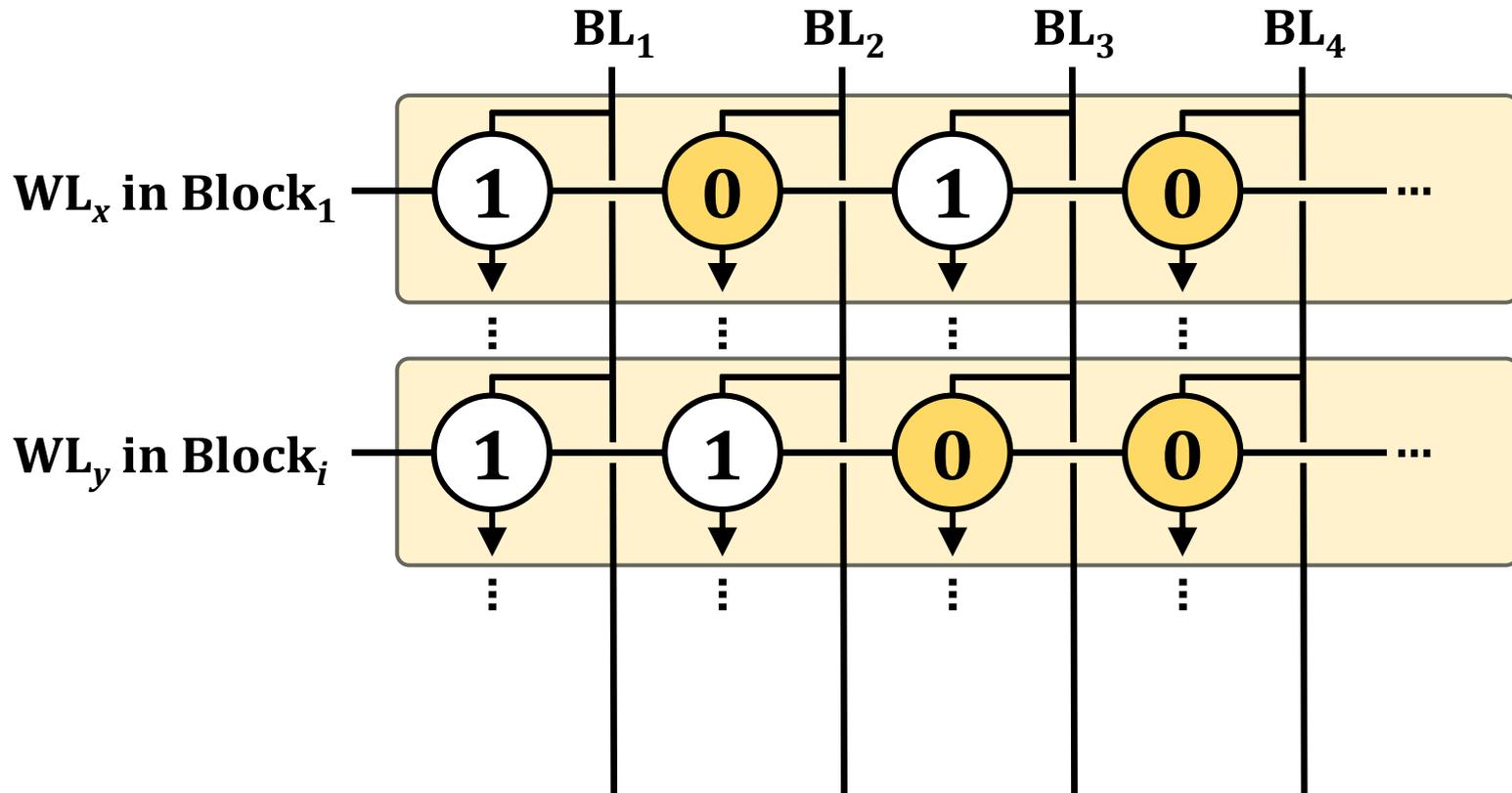
- Intra-Block MWS: Simultaneously activates multiple WLs in the same block
 - Bitwise AND of the stored data in the WLs

Flash-Cosmos (Intra-Block MWS) enables bitwise AND of multiple pages in the same block via a single sensing operation



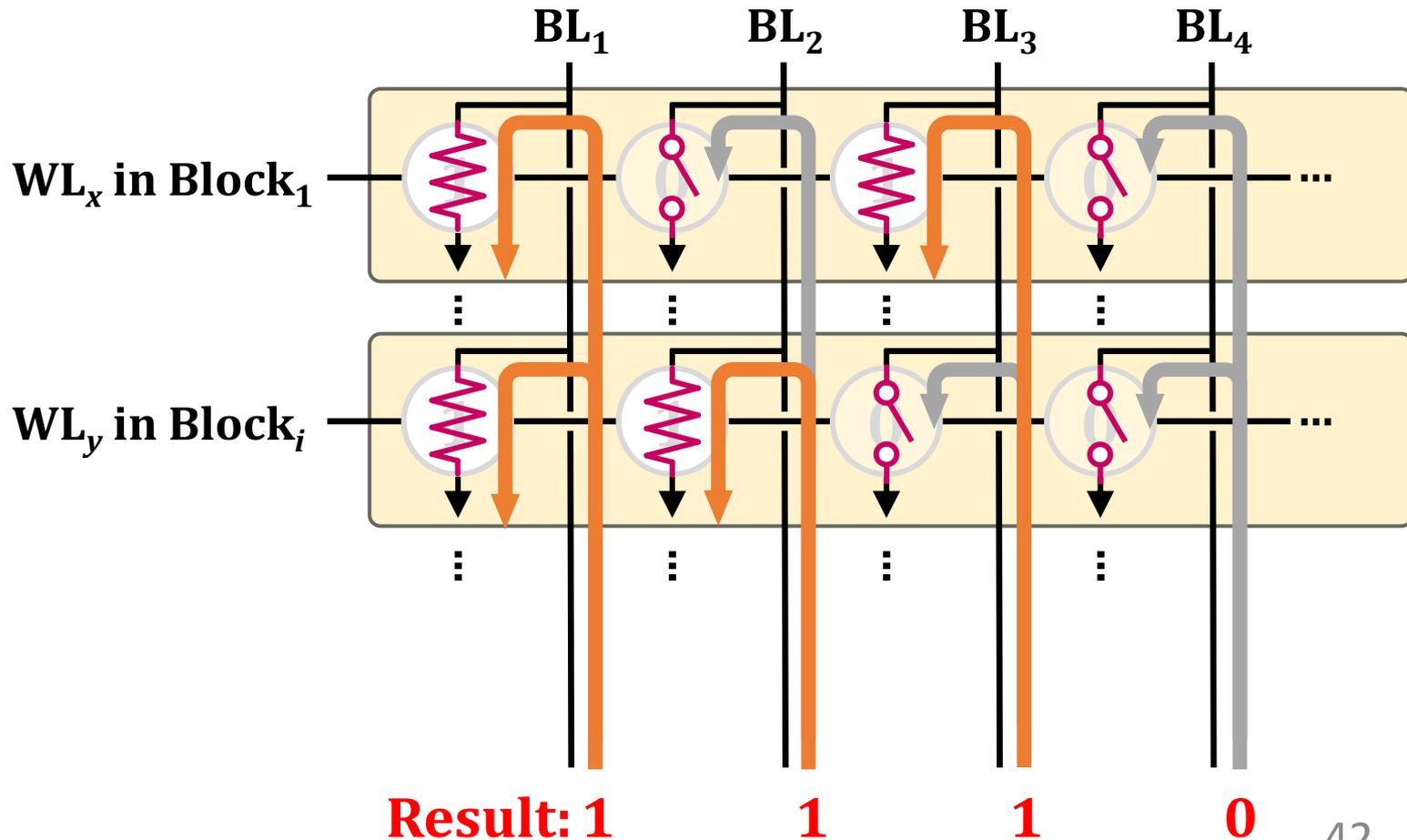
Multi-Wordline Sensing (MWS): Bitwise OR

- **Inter-Block MWS:** Simultaneously activates multiple WLs in different blocks
 - **Bitwise OR** of the stored data in the WLs



Multi-Wordline Sensing (MWS): Bitwise OR

- **Inter-Block MWS:** Simultaneously activates multiple WLs in different blocks
 - **Bitwise OR** of the stored data in the WLs

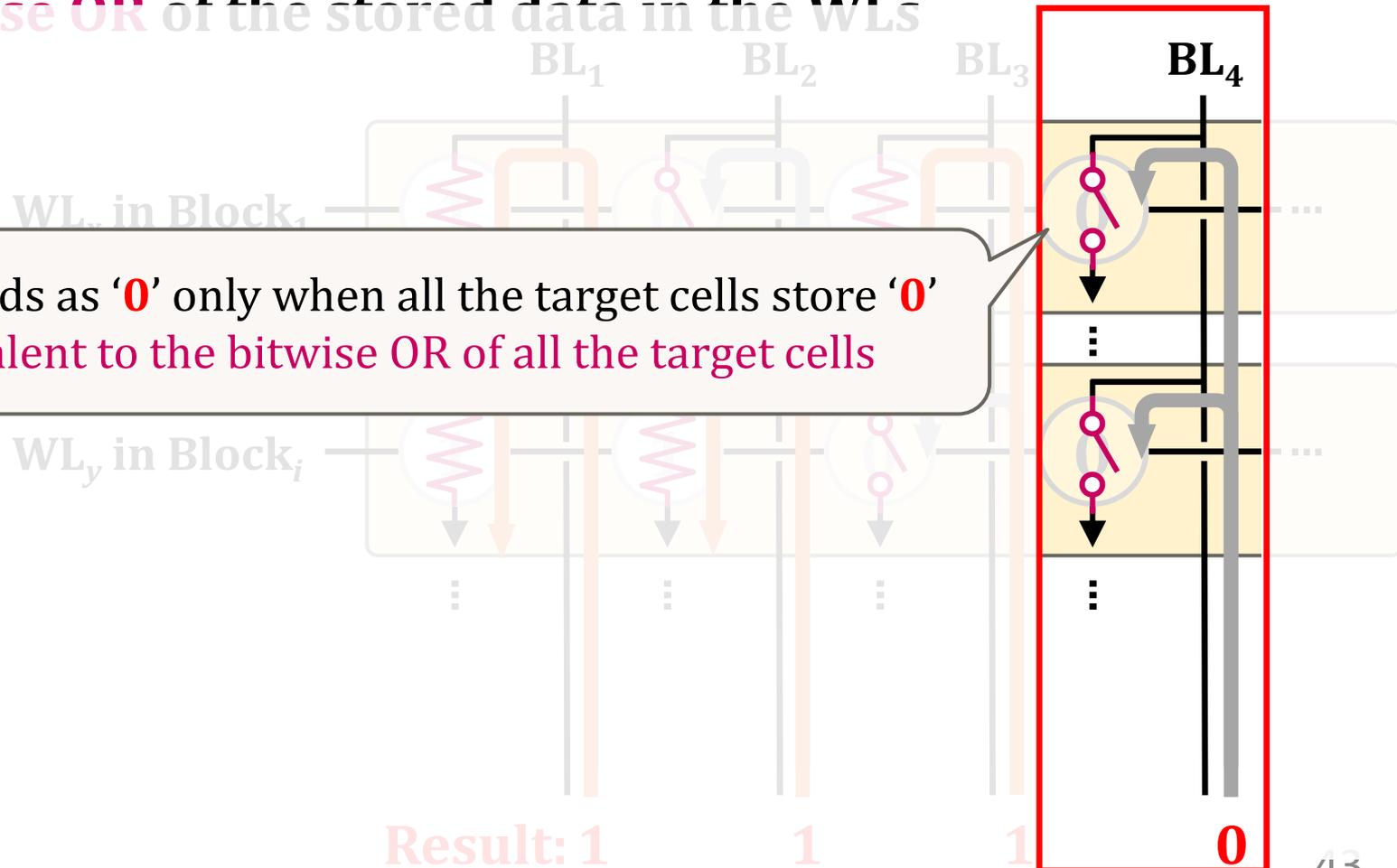


Multi-Wordline Sensing (MWS): Bitwise OR

- **Inter-Block MWS:** Simultaneously activates multiple WLs in different blocks

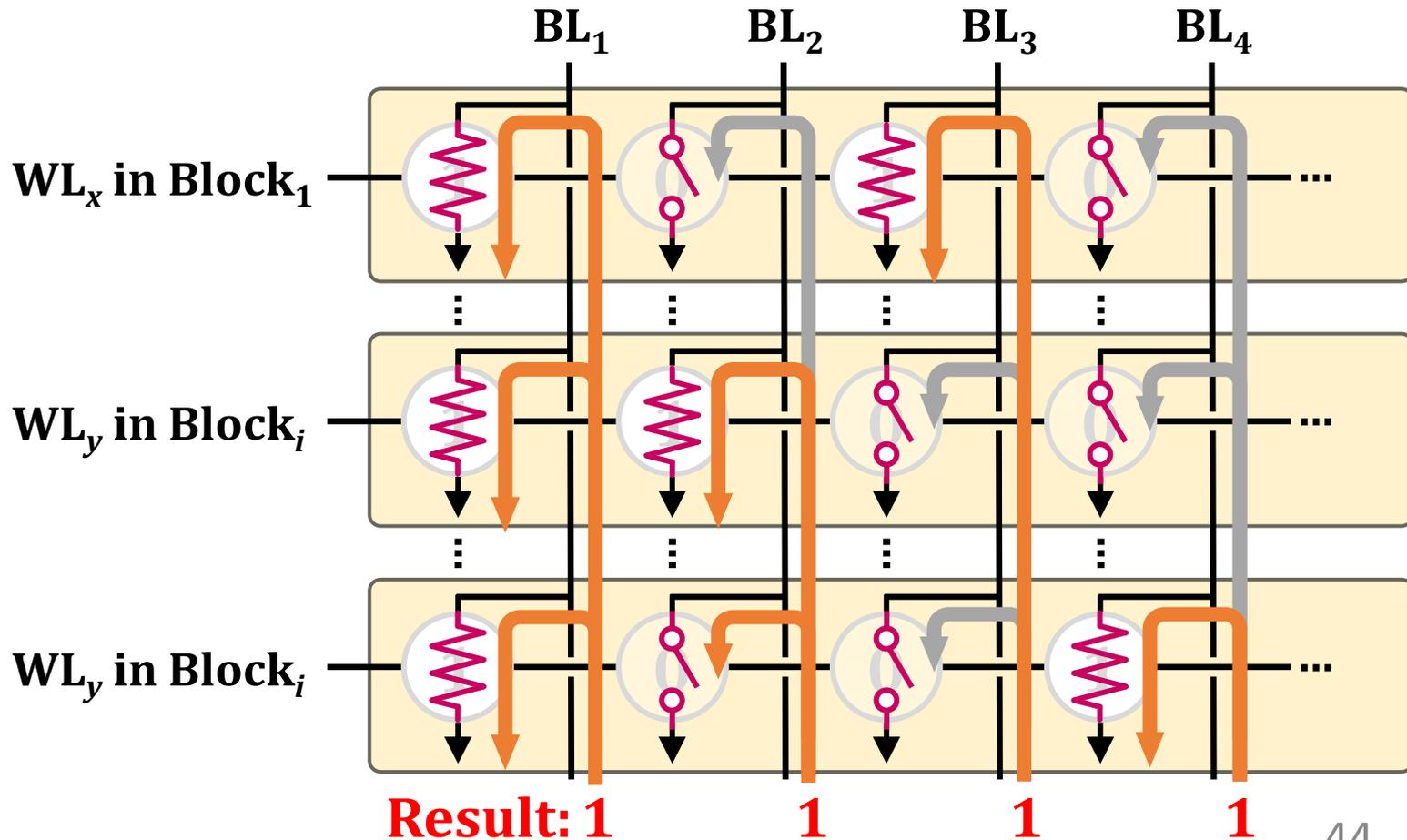
- Bitwise OR of the stored data in the WLs

A bitline reads as '0' only when all the target cells store '0'
→ Equivalent to the bitwise OR of all the target cells



Multi-Wordline Sensing (MWS): Bitwise OR

- **Inter-Block MWS:** Simultaneously activates multiple WLs in different blocks
 - **Bitwise OR** of the stored data in the WLs

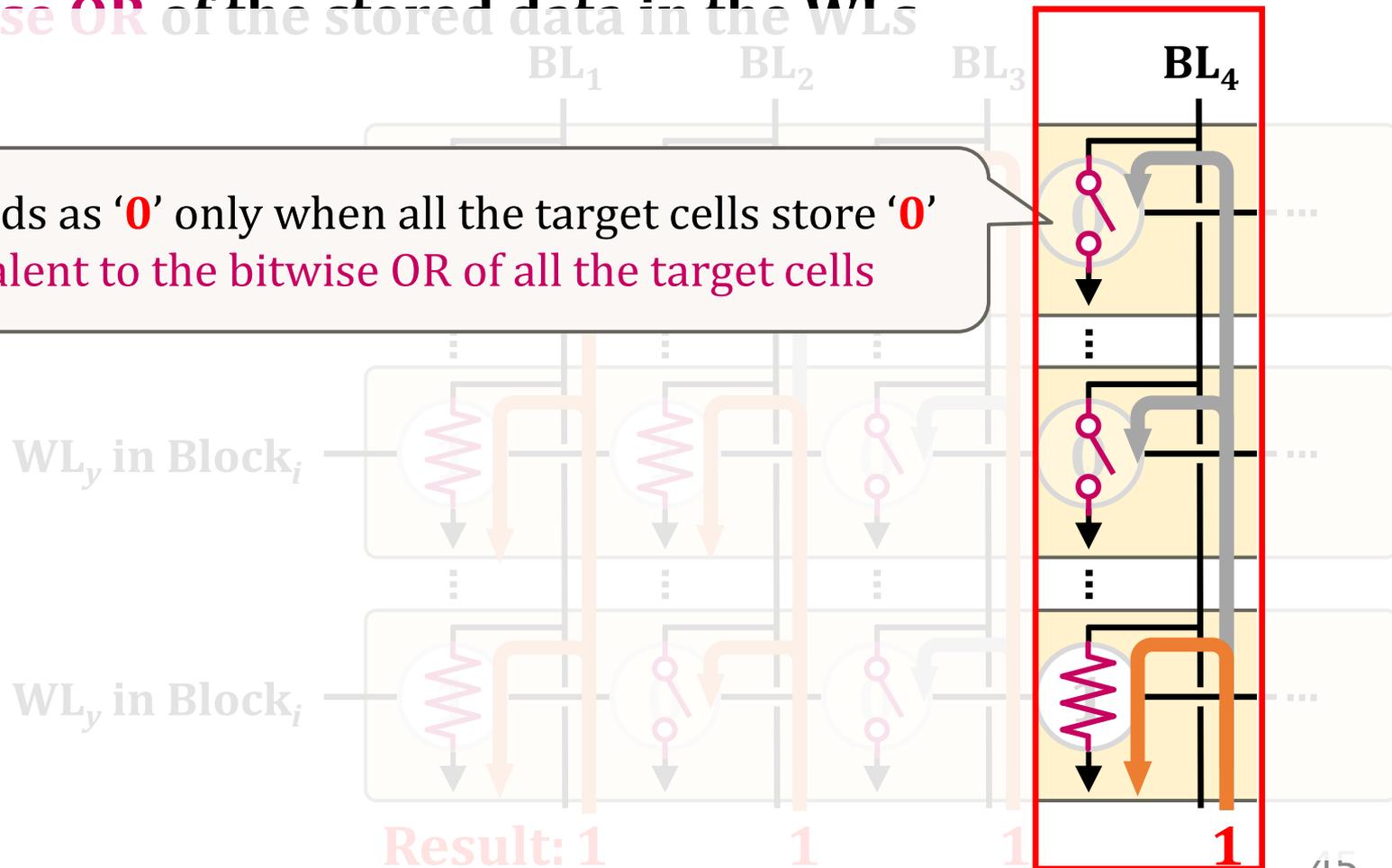


Multi-Wordline Sensing (MWS): Bitwise OR

- **Inter-Block MWS: Simultaneously activates multiple WLs in different blocks**

- Bitwise OR of the stored data in the WLs

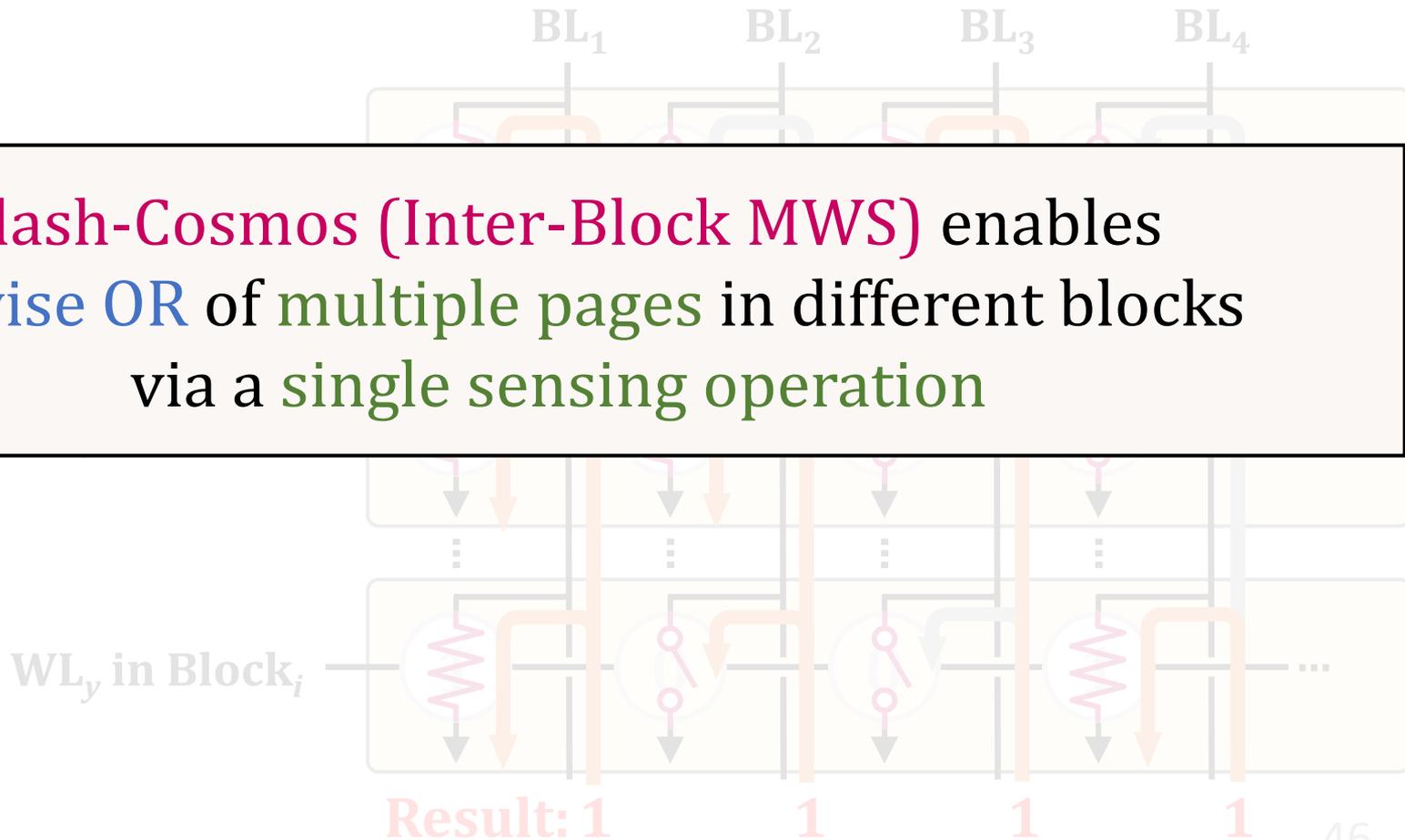
A bitline reads as '0' only when all the target cells store '0'
→ Equivalent to the bitwise OR of all the target cells



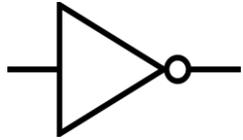
Multi-Wordline Sensing (MWS): Bitwise OR

- **Inter-Block MWS:** Simultaneously activates multiple WLs in different blocks
 - **Bitwise OR** of the stored data in the WLs

Flash-Cosmos (Inter-Block MWS) enables bitwise OR of multiple pages in different blocks via a single sensing operation

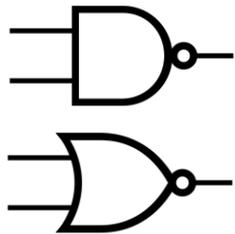


Supporting Other Bitwise Operations



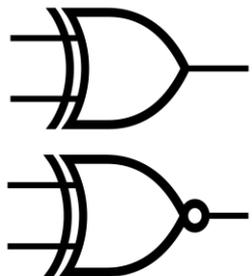
Bitwise NOT

Exploit **Inverse Read**^[1] which is supported in modern NAND flash memory for copy-back operations



Bitwise NAND/ NOR

Exploit **De Morgan's Laws**



Bitwise XOR/XNOR

Use **XOR between sensing and cache latches**^[2] which is also supported in NAND flash memory

[1] Lee+, "High-Performance 1-Gb-NAND Flash Memory with 0.12- μ m Technology," JSSC, 2002

[2] Kim+, "A 512-Gb 3-b/Cell 64-Stacked WL 3-D-NAND Flash Memory," JSSC, 2018

Flash-Cosmos: Overview



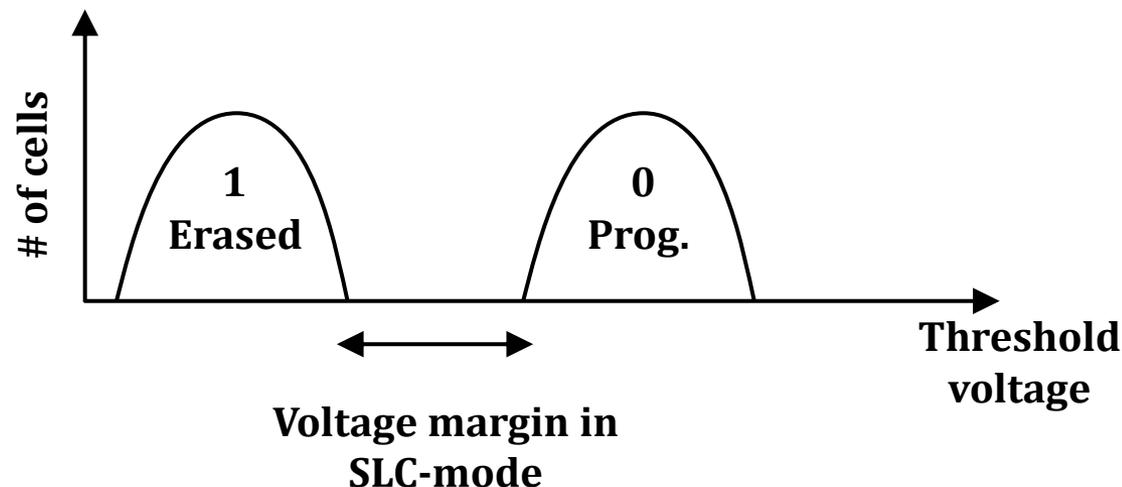
Enables in-flash bulk bitwise operations on multiple operands with a *single* sensing operation using Multi-Wordline Sensing (MWS)



Increases the reliability of in-flash bulk bitwise operations by using Enhanced SLC-mode Programming (ESP)

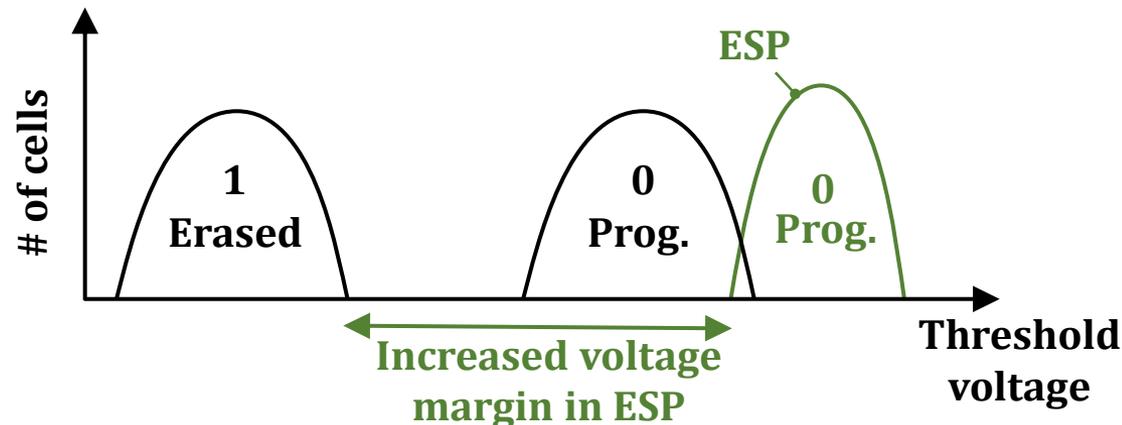
Enhanced SLC-Mode Programming (ESP)

- SLC-mode programming provides a large voltage margin between the erased and programmed states
- Based on our real device characterization, we observe that SLC-mode programming is still highly error-prone without the use of ECC and data-randomization



Enhanced SLC-Mode Programming (ESP)

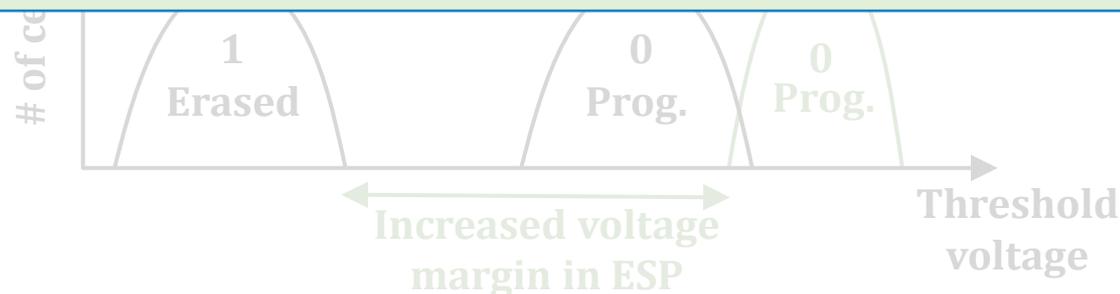
- ESP further **increases** the **voltage margin** between the erased and programmed states
- A **wider voltage margin** between the two states **improves** reliability by making the cells **less vulnerable** to errors



Enhanced SLC-Mode Programming (ESP)

- ESP increases the voltage margin between the erased and programmed states
- A wider voltage margin between the two states improves reliability during data sensing by making the cells less vulnerable to errors

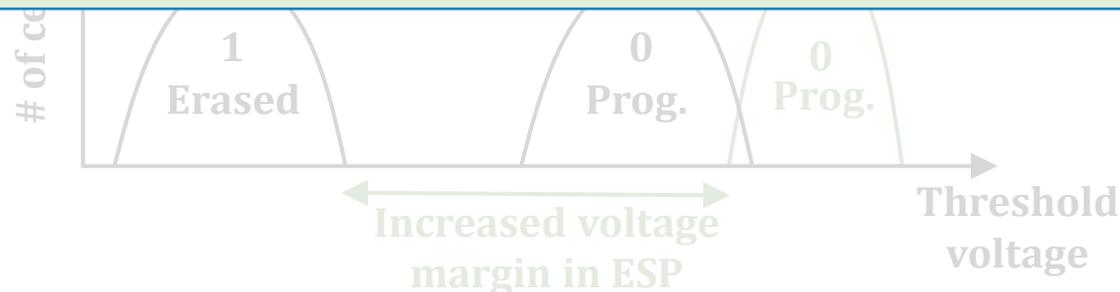
ESP improves the reliability of in-flash computation without the use of ECC or data-randomization techniques



Enhanced SLC-Mode Programming (ESP)

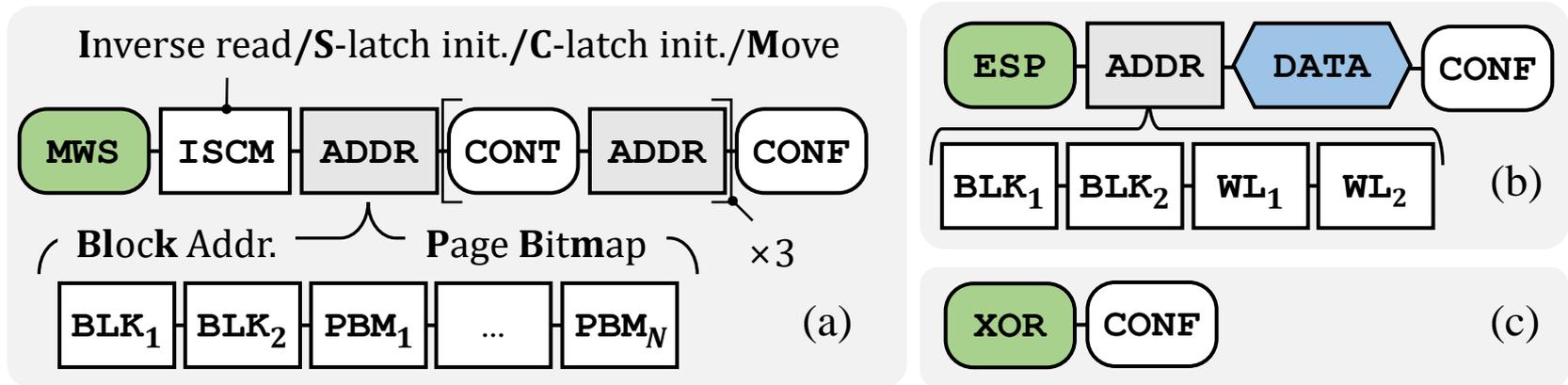
- ESP increases the voltage margin between the erased and programmed states
- A wider voltage margin between the two states improves reliability during data sensing by making the cells less vulnerable to errors

ESP can improve the reliability of prior in-flash processing techniques as well



New NAND Commands

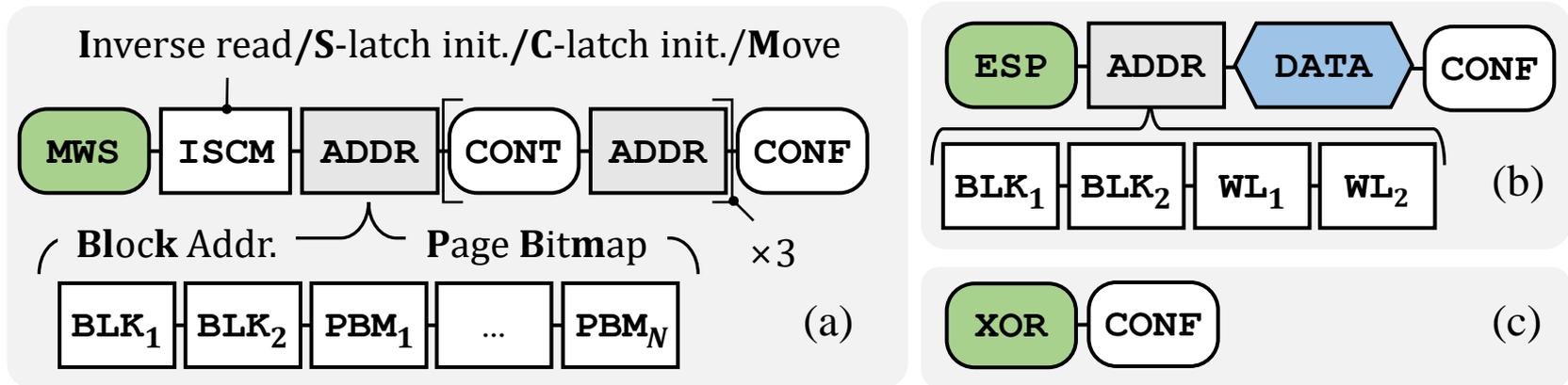
- **Three new commands to support Flash-Cosmos**



- **MWS command** to be used for,
 - Intra- and inter-block MWS
 - Inverse read
 - Accumulation of results of all reads
- **ISCM command** slot allows the flash controller to turn on/off four features,
 - Inverse-read mode (I)
 - Sensing-latch (S-latch) initialization (S)
 - Cache-latch (C-latch) initialization (C)
 - Move data from S-latch to C-latch (M)

New NAND Commands

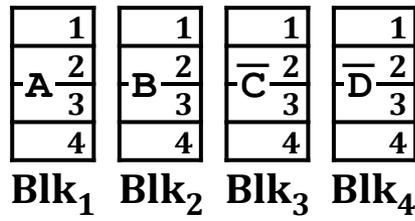
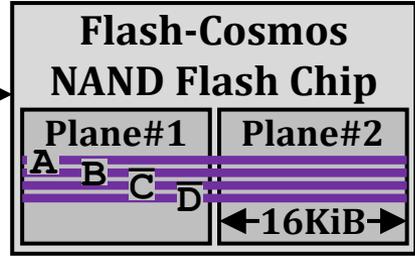
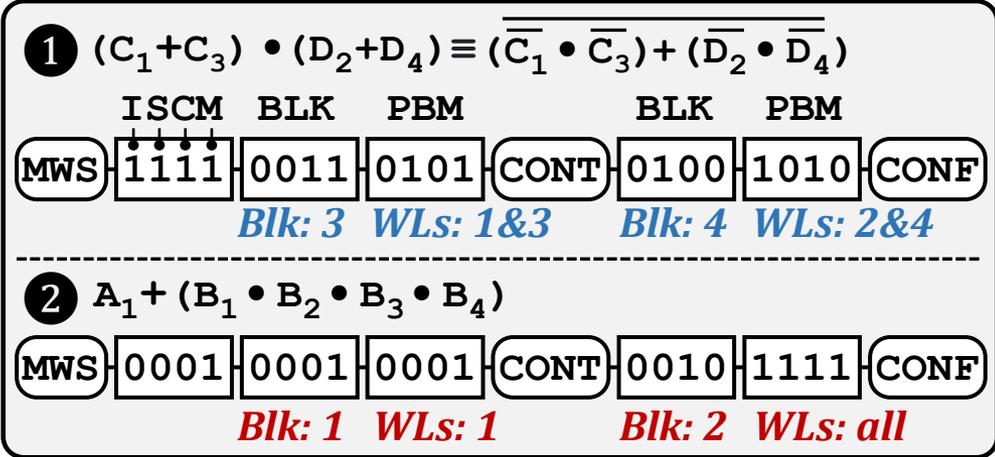
- **Three new commands to support Flash-Cosmos**



- **ESP** command works like a regular program command
- **XOR command** performs **bitwise XOR** between the **sensing and cache latches** and stores the result in C-latch

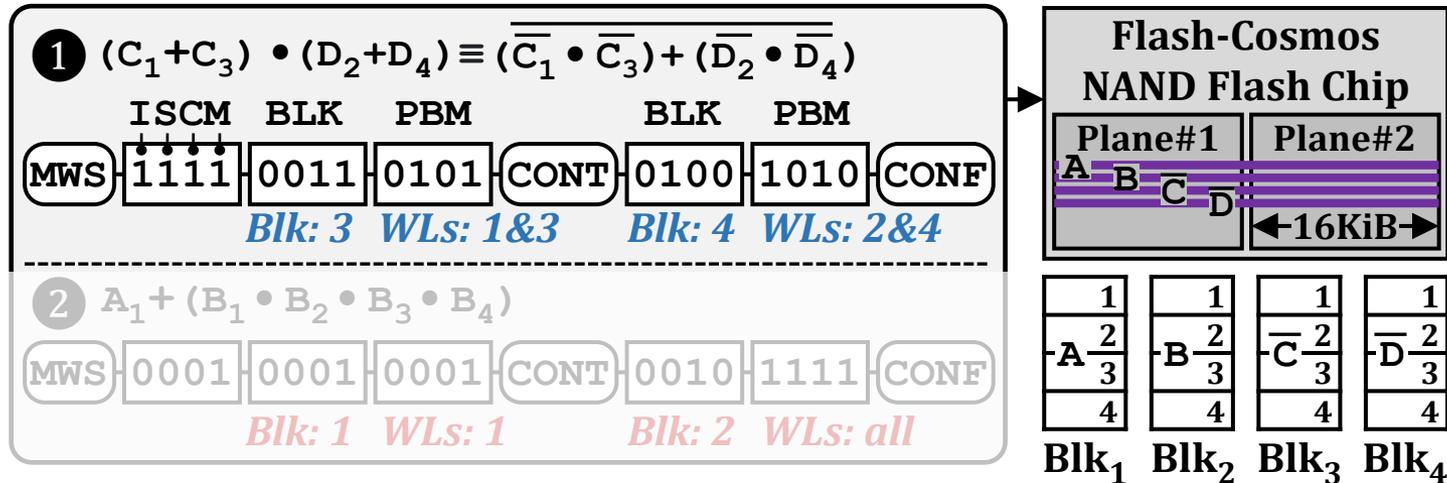
Operational example

$$\{A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4)\} \cdot (C_1 + C_3) \cdot (D_2 + D_4)$$



Operational example

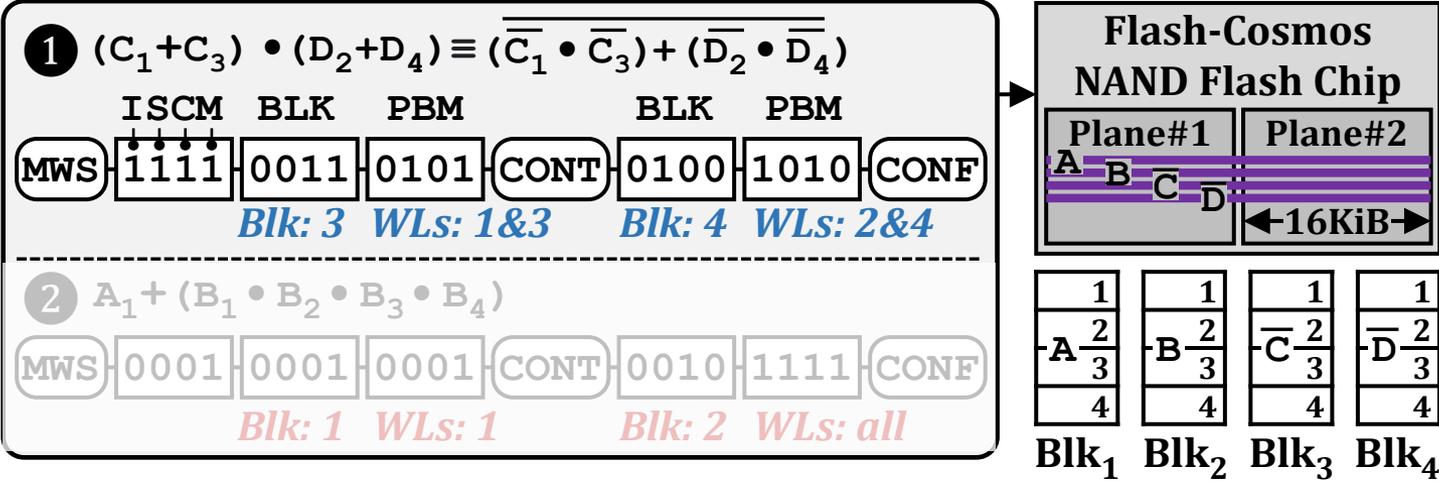
$$\{A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4)\} \cdot (C_1 + C_3) \cdot (D_2 + D_4)$$



Bit vectors C_i and D_i are programmed with their **inverse data** for the **bitwise OR operation**

Operational example

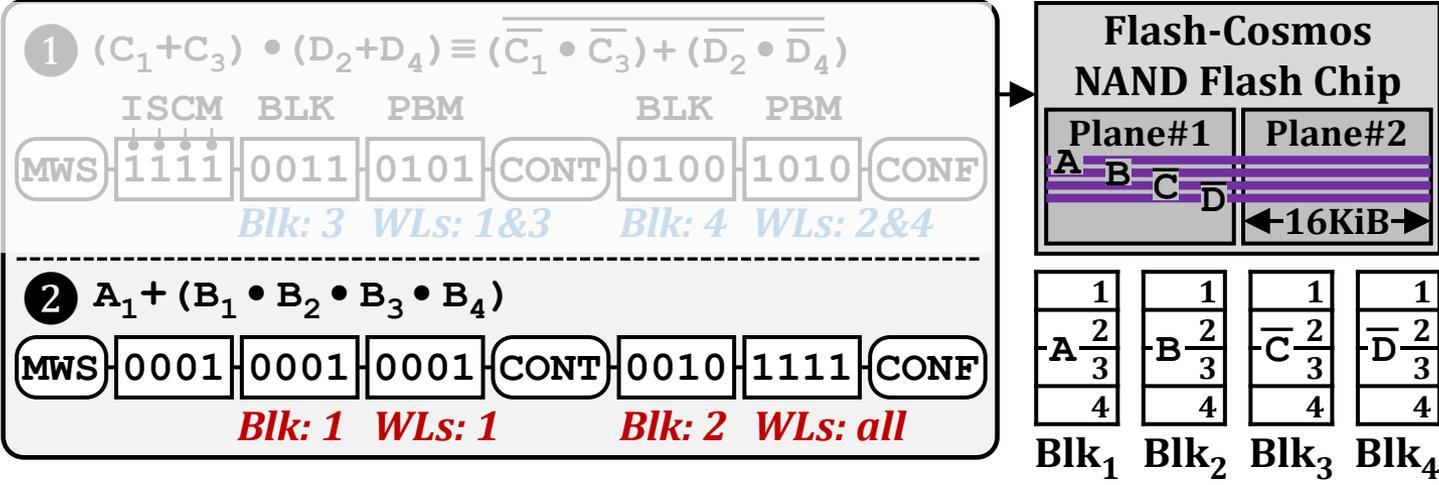
$$\{A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4)\} \cdot (C_1 + C_3) \cdot (D_2 + D_4)$$



Perform an **intra-block MWS command** for $(C_1 + C_3) \cdot (D_2 + D_4)$ by **enabling inverse-read mode and initialization of sensing and cache latches**

Operational example

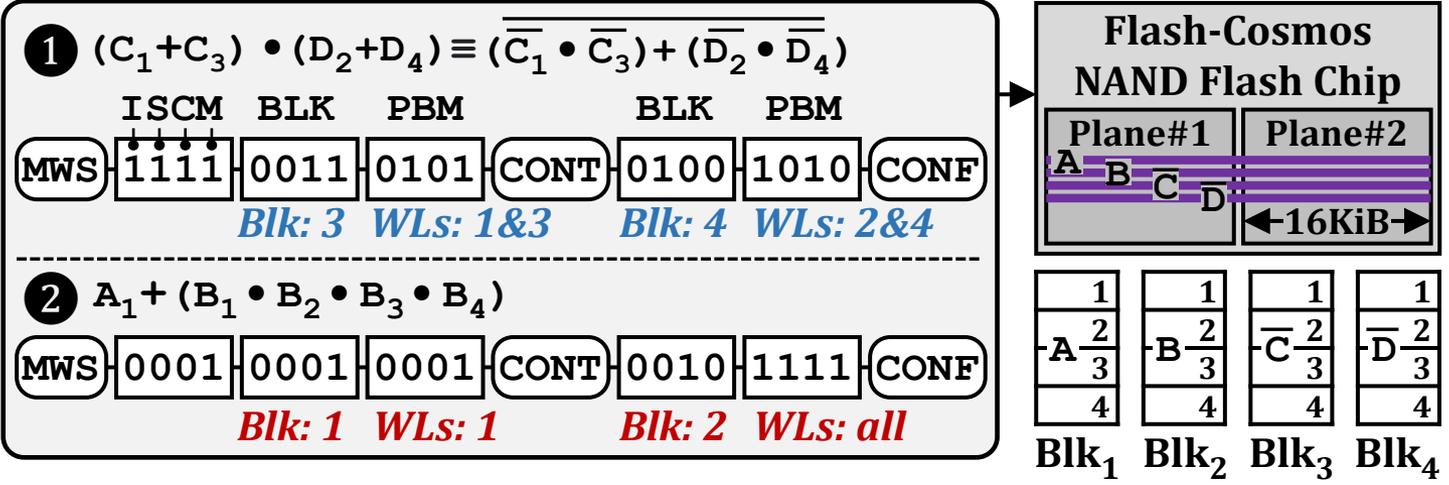
$$\{A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4)\} \cdot (C_1 + C_3) \cdot (D_2 + D_4)$$



Perform an **intra-block MWS** command for $\{A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4)\}$ while **disabling** the **inverse-read mode** and **initialization** of both latches

Operational example

$$\{A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4)\} \cdot (C_1 + C_3) \cdot (D_2 + D_4)$$



By disabling the initialization of latches, the result of the two intra-block MWS commands are accumulated in S-latch and C-latch

Talk Outline

Motivation

Background

Flash-Cosmos

Evaluation

Summary

Evaluation Methodology

- We evaluate Flash-Cosmos using

160 real state-of-the-art 3D NAND flash chips

Real Device Characterization

- We validate the **feasibility**, **performance**, and **reliability** of Flash-Cosmos
- 160 48-layer 3D TLC NAND flash chips
 - 3,686,400 tested wordlines
- Under worst-case operating conditions
 - 1-year retention time at 10K P/E cycles
 - Worst-case data patterns

Results: Real-Device Characterization

Both intra- and inter-block MWS operations require **no changes** to the cell array of **commodity NAND flash chips**

Both MWS operations can activate **multiple WLS** (**intra**: up to 48, **inter**: up to 4) at the same time with **small increase** in sensing latency (**< 10%**)

ESP significantly improves the **reliability** of computation results (**no observed bit error** in the tested flash cells)

Evaluation Methodology

- We evaluate Flash-Cosmos using

160 real state-of-the-art 3D NAND flash chips

Three real-world applications that perform
bulk bitwise operations

Evaluation with real-world workloads

- **Simulation**

- **MQSim [Tavakkol+, FAST'18]** to model the performance of Flash-Cosmos and the baselines

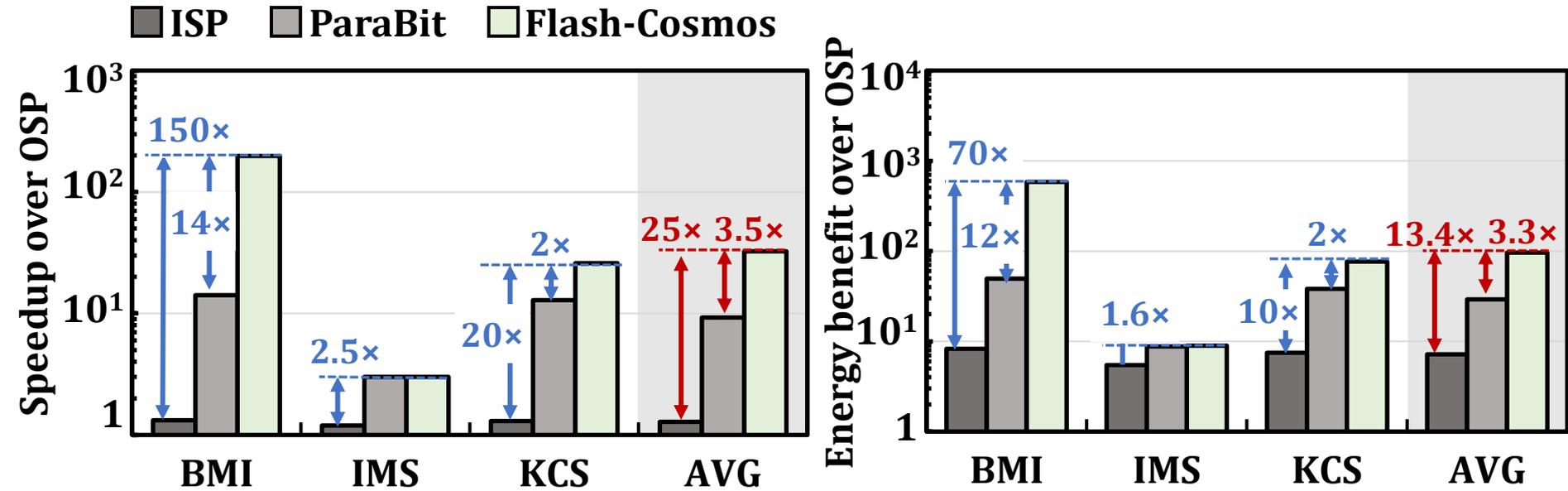
- **Workloads**

- Three real-world applications that heavily rely on bulk bitwise operations
- **Bitmap Indices (BMI)**: Bitwise AND of up to $\sim 1,000$ operands
- **Image Segmentation (IMS)**: Bitwise AND of 3 operands
- **k -clique star listing (KCS)**: Bitwise OR of up to 32 operands

- **Baselines**

- **Outside-Storage Processing (OSP)**: a multi-core CPU (Intel i7 11700K)
- **In-Storage Processing (ISP)**: an in-storage hardware accelerator
- **ParaBit [Gao+, MICRO'21]**: the state-of-the-art in-flash processing (IFP) mechanism

Results: Performance & Energy



Flash-Cosmos provides **significant performance & energy benefits** over all the baselines

The larger the number of operands,
the higher the performance & energy benefits

Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory

Jisung Park^{§∇} Roknoddin Azizi[§] Geraldo F. Oliveira[§] Mohammad Sadrosadati[§]
Rakesh Nadig[§] David Novo[†] Juan Gómez-Luna[§] Myungsuk Kim[‡] Onur Mutlu[§]

[§]*ETH Zürich* [∇]*POSTECH* [†]*LIRMM, Univ. Montpellier, CNRS* [‡]*Kyungpook National University*



<https://arxiv.org/abs/2209.05566.pdf>

Talk Outline

Motivation

Background

Flash-Cosmos

Evaluation of Flash-Cosmos and Key Results

Summary

Flash-Cosmos: Summary



First work to enable multi-operand bulk bitwise operations with a single sensing operation and high reliability



Improves performance by 3.5x/25x/32x on average over ParaBit/ISP/OSP across the workloads



Improves energy efficiency by 3.3x/13.4x/95x on average over ParaBit/ISP/OSP across the workloads



Low-cost & requires no changes to flash cell arrays

Flash-Cosmos

In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory

Jisung Park, Roknoddin Azizi, Geraldo F. Oliveira,
Mohammad Sadrosadati, Rakesh Nadig, David Novo, Juan Gómez Luna,
Myungsuk Kim, and Onur Mutlu

Published at MICRO 2022

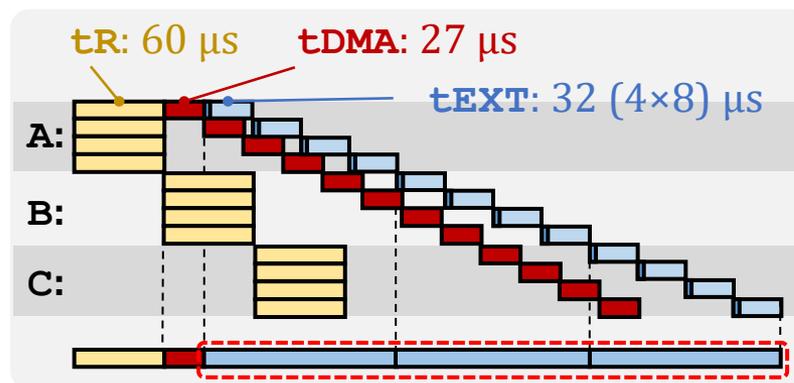
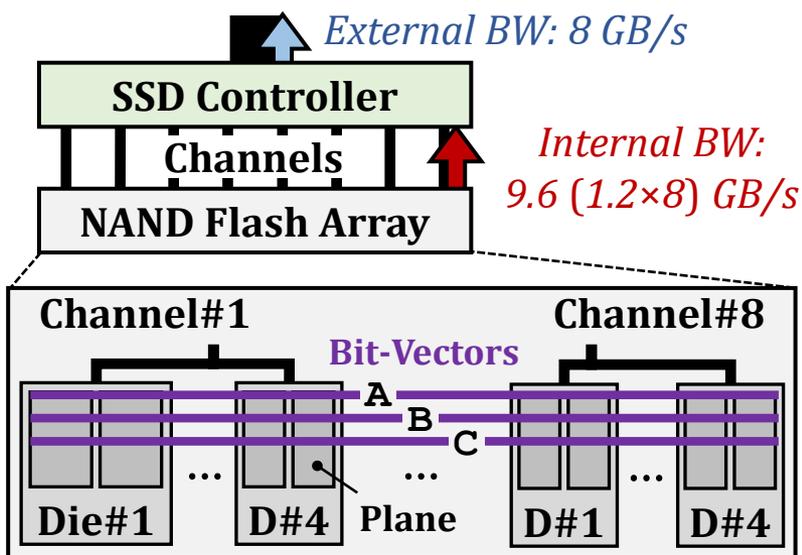
Rakesh Nadig
P&S SSD Course
26th October 2022



BACKUP SLIDES

Outside-Storage Processing (OSP)

- Moves every operand from the storage to the compute unit (CPU/GPU) for computation
- Performs the computation and writes the results back to the SSD

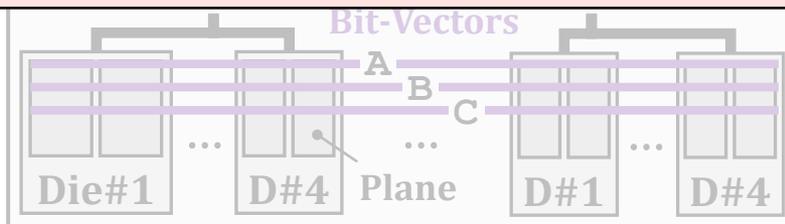


OSP: Outside-Storage Processing

Outside-Storage Processing (OSP)

- Moves every operand from the storage to the compute unit (CPU/GPU) for computation
- Performs the computation and writes the results back to the SSD

Outside-Storage Processing is bottlenecked by data movement between the compute unit and SSD (**SSD external bandwidth**)



Outside-Storage Processing (OSP)

- Moves every operand from the storage to the compute unit (CPU/GPU) for computation
- Performs the computation and writes the results back to the SSD

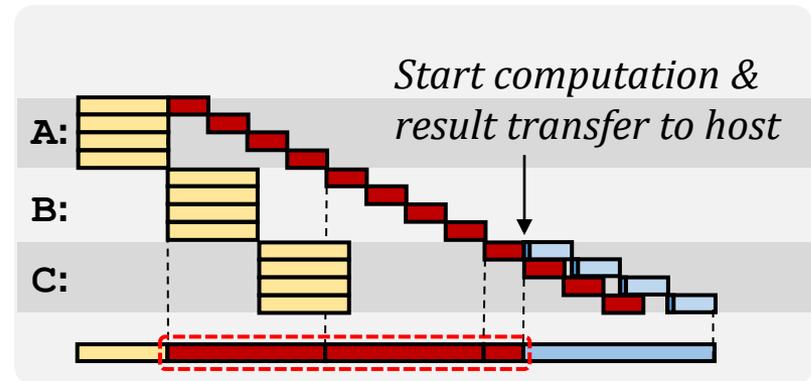
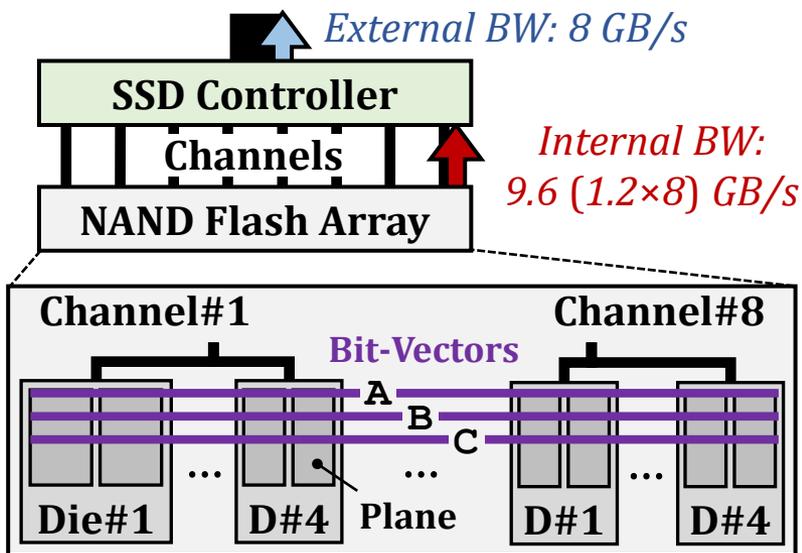
Near-Data Processing (NDP)

- ✓ moves computation closer to where the data resides
- ✓ is a promising approach to mitigate data movement



In-Storage Processing (ISP)

- Reads the operands from the NAND flash chips to the SSD Controller in a serial manner
- Performs the computation in the SSD controller
- Moves the computation result to the host



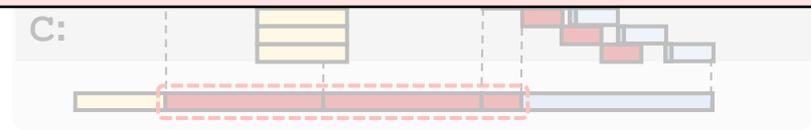
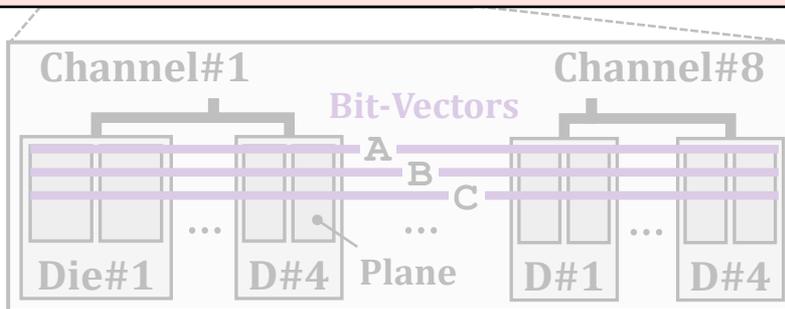
Bottleneck: Internal I/O

ISP: In-Storage Processing

In-Storage Processing (ISP)

- Reads the operands from the NAND flash chips to the SSD Controller in a serial manner
- Performs the computation in the SSD controller
- Moves the computation result to the host

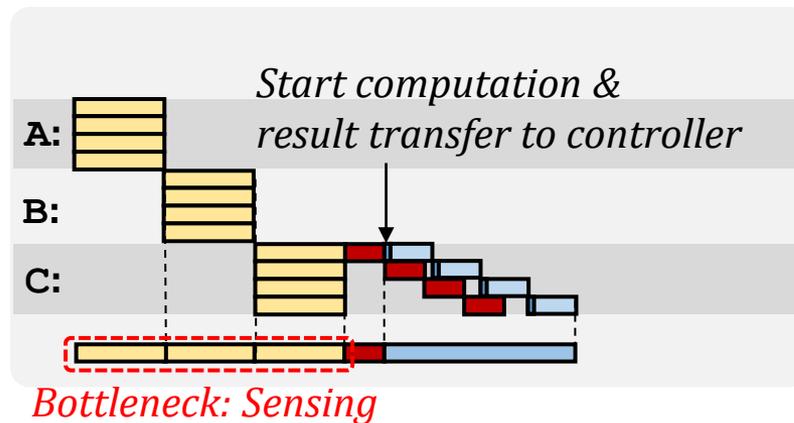
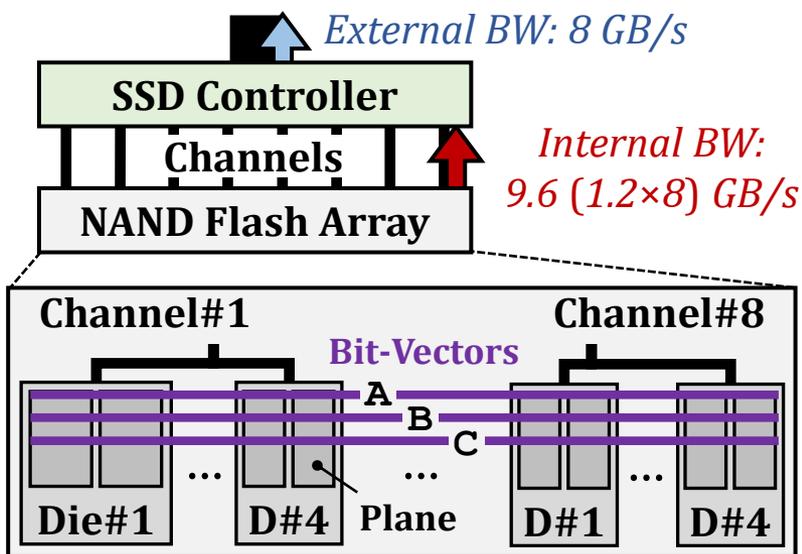
In-Storage Processing is bottlenecked by data movement between NAND flash memory and SSD controller (**SSD internal bandwidth**)



Bottleneck: Internal I/O
ISP: In-Storage Processing

In-Flash Processing (IFP)

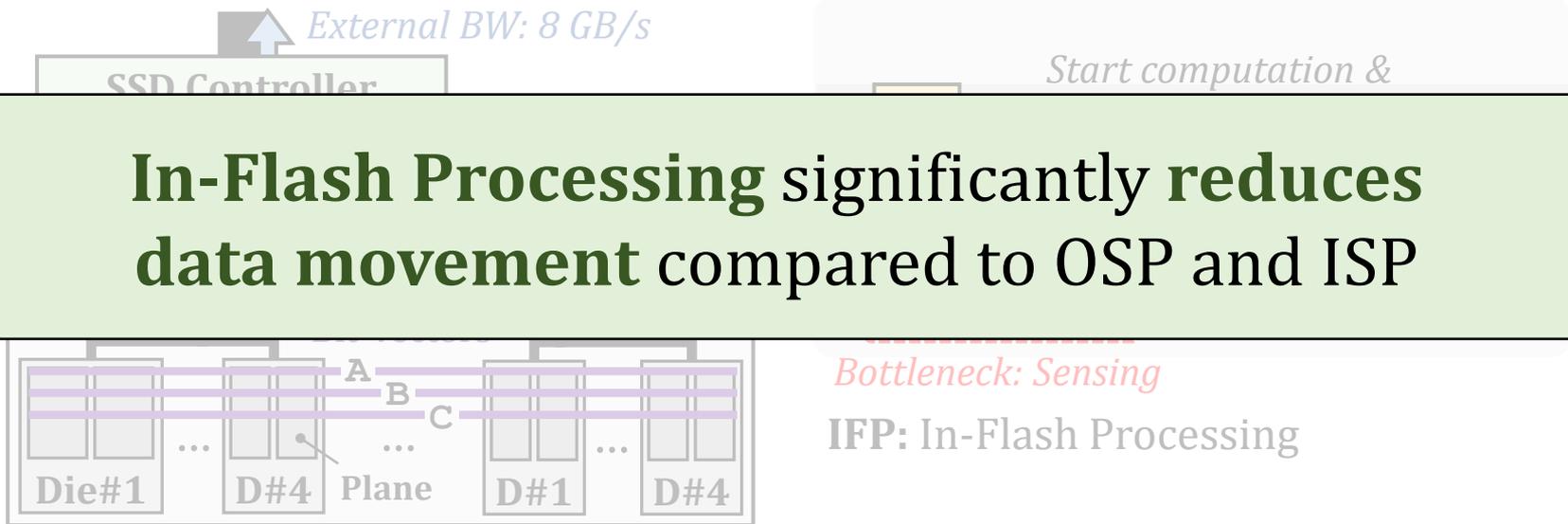
- Performs computation within the NAND flash chips
- Moves only the computation results from the NAND flash chips to the SSD controller and host



IFP: In-Flash Processing

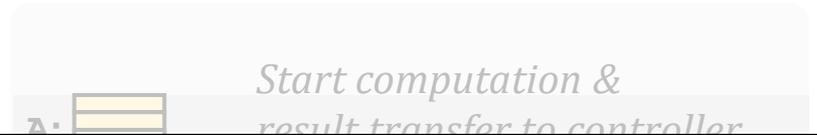
In-Flash Processing (IFP)

- Performs computation within the NAND flash chips
- Moves only the computation results from the NAND flash chips to the SSD controller and host

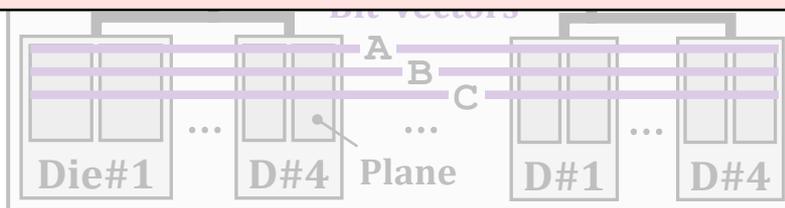


In-Flash Processing (IFP)

- Performs computation within the NAND flash chips
- Moves only the computation results from the NAND flash chips to the SSD controller and host



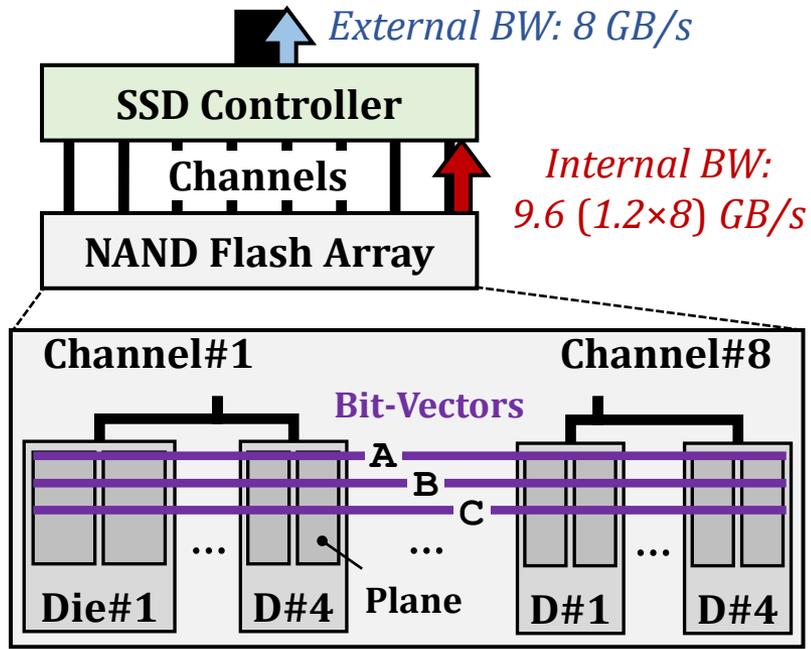
In-Flash Processing is bottlenecked by data sensing



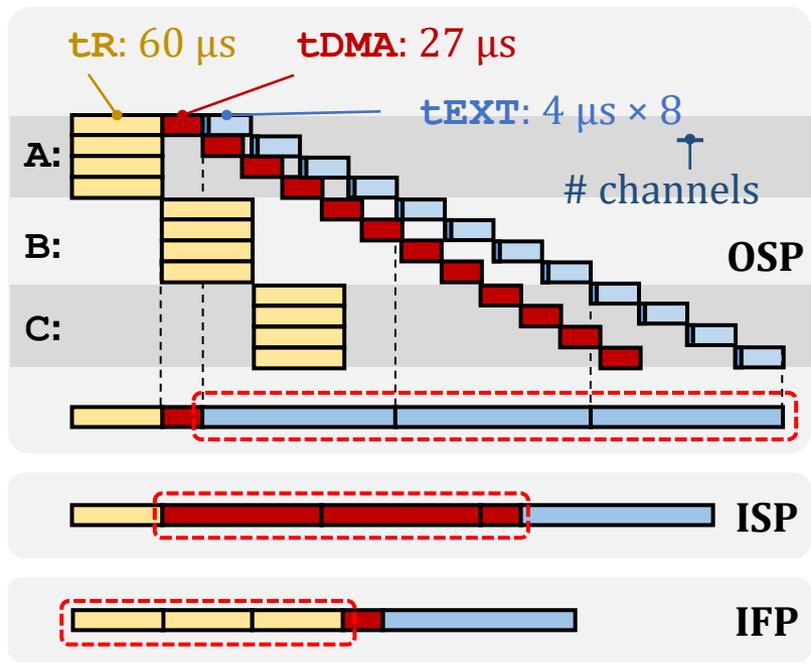
Bottleneck: Sensing

IFP: In-Flash Processing

Overview of three computation approaches



SSD organization



Channel#1 timeline

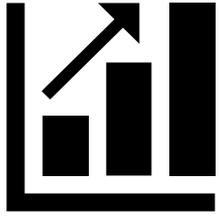
In-Flash processing significantly reduces data movement

Limitations of prior IFP techniques

Perform multiple **two-operand bitwise operations** in a **serial manner**, leading to **sensing bottleneck**

Require the **application** to **have high error tolerance** since these works **cannot leverage** any widely-used ECC or data-randomization techniques

Our Goals



**Improve the performance of
in-flash bulk bitwise operations**



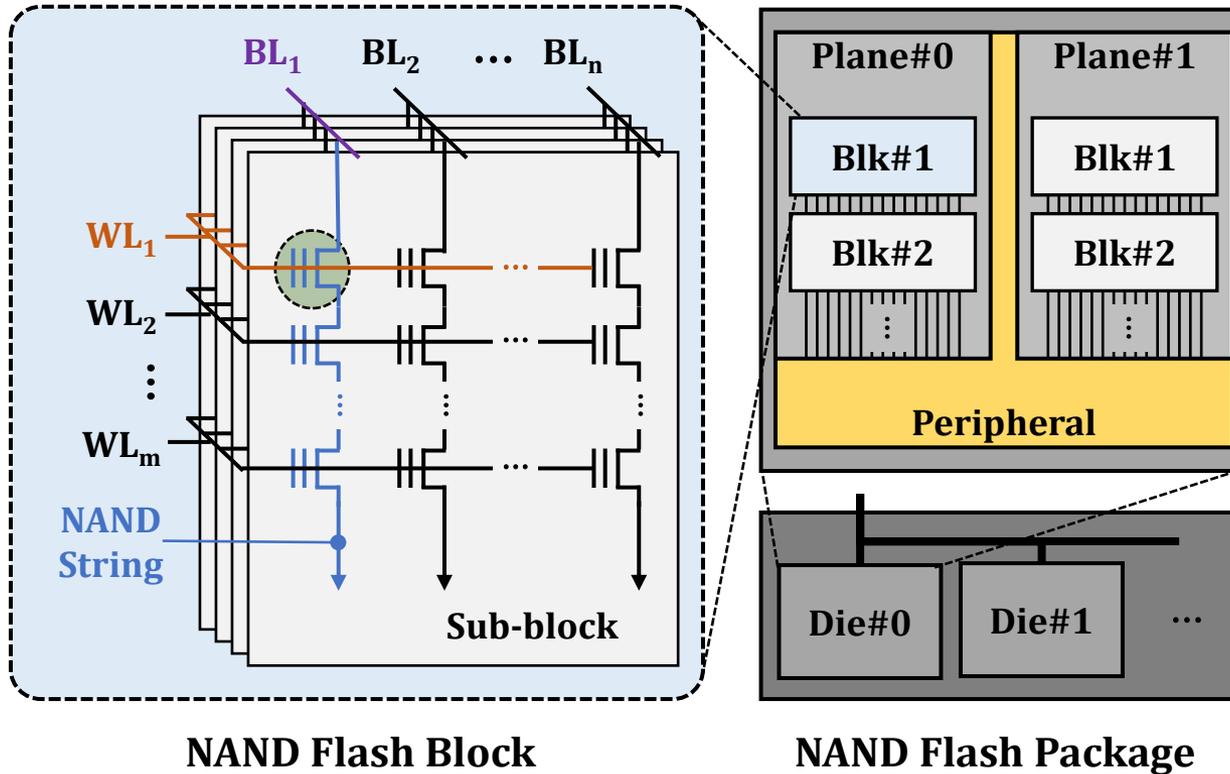
**Improve energy efficiency of
in-flash bulk bitwise operations**



**Provide high reliability for the computation
results of in-flash bulk bitwise operations**

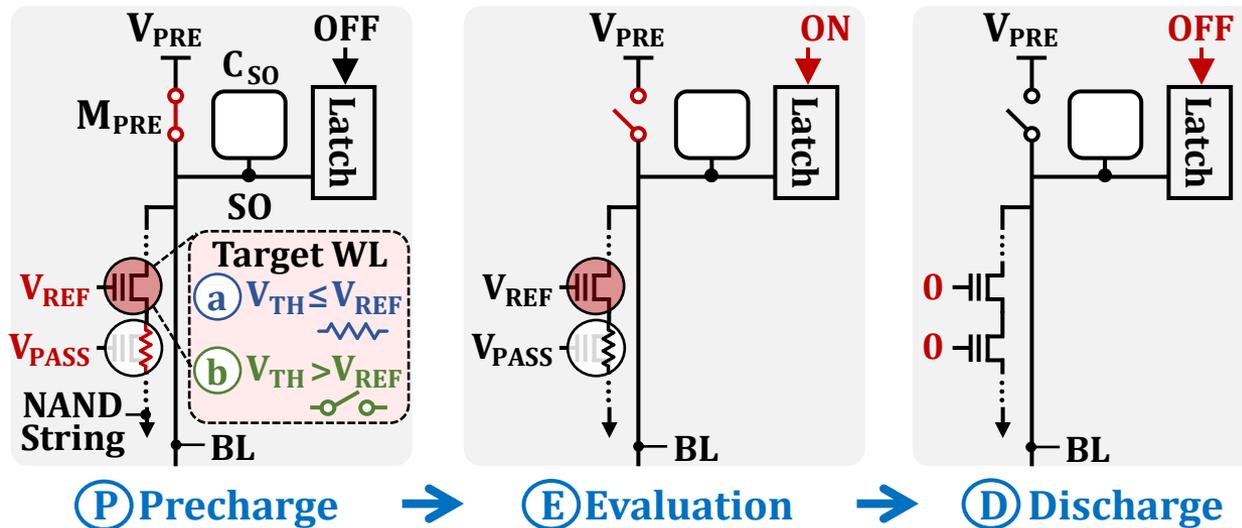
Background

- **NAND Flash Organization**



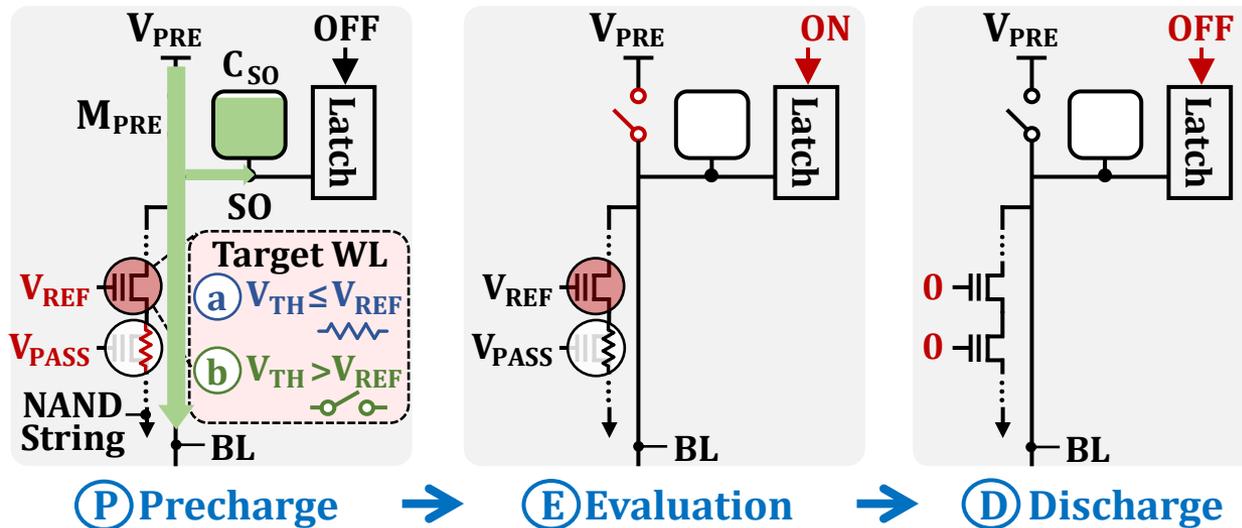
Background

- NAND flash read mechanism consists of three steps:
1) Precharge 2) Evaluation 3) Discharge



Background

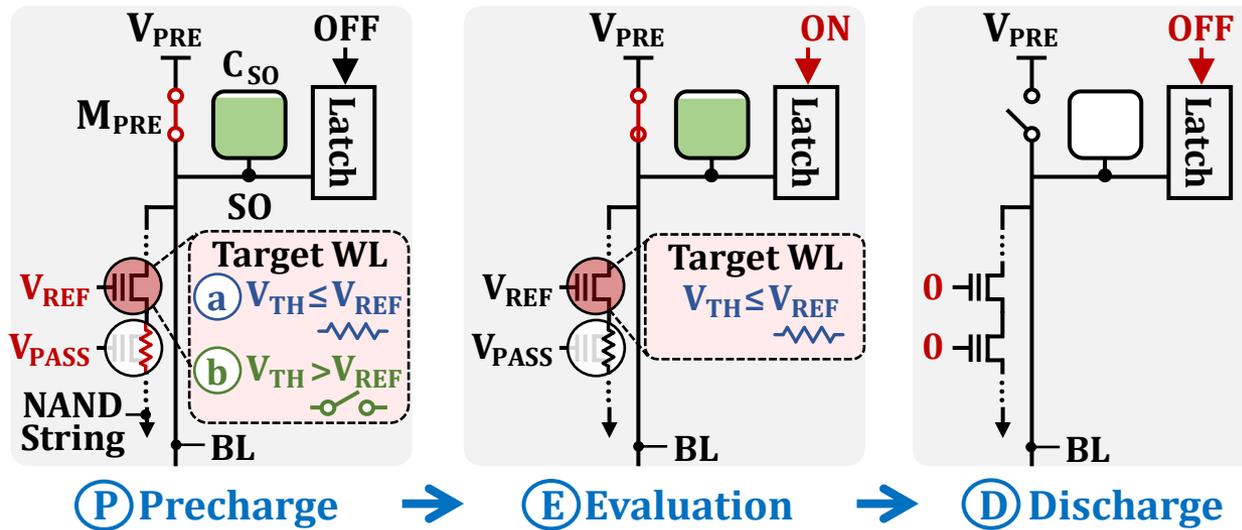
- Precharge



Enable precharge transistor M_{PRE} to charge all target BLs and their sense-out capacitors (C_{SO}) to V_{PRE}

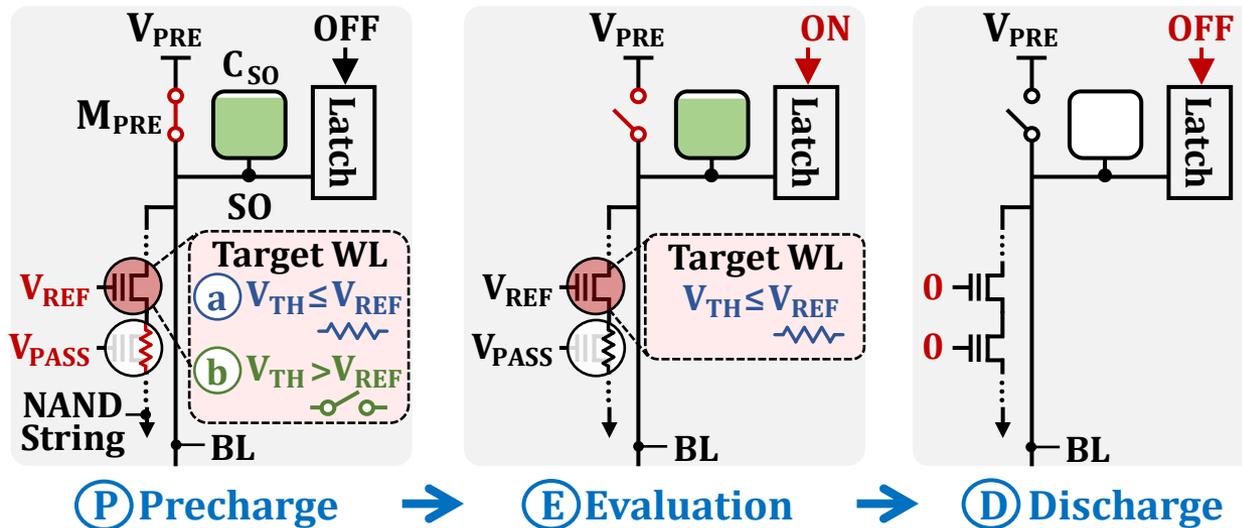
Background

- Evaluation



Background

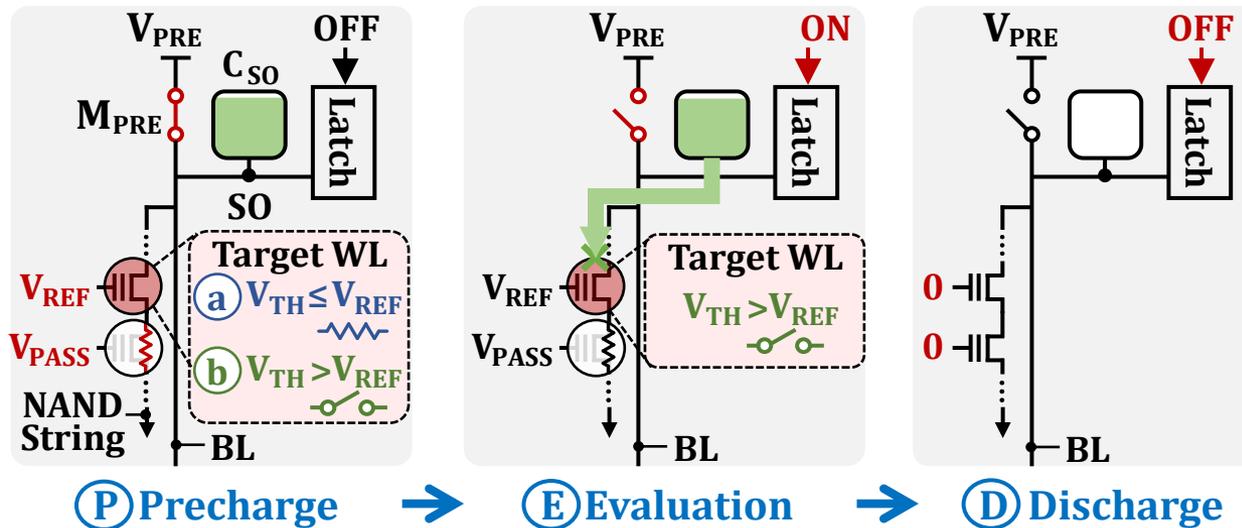
- Evaluation



Disconnect the BLs from V_{PRE} and enable the latching circuit

Background

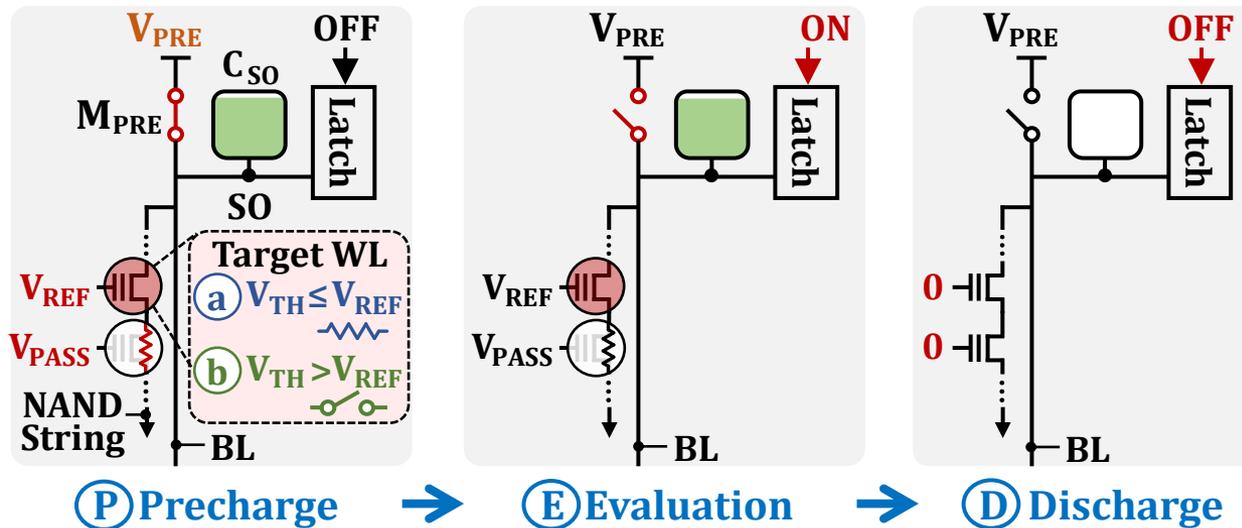
- Evaluation



If $V_{TH} > V_{REF}$, the target cell blocks the BL discharge current (Sensed as 0)

Background

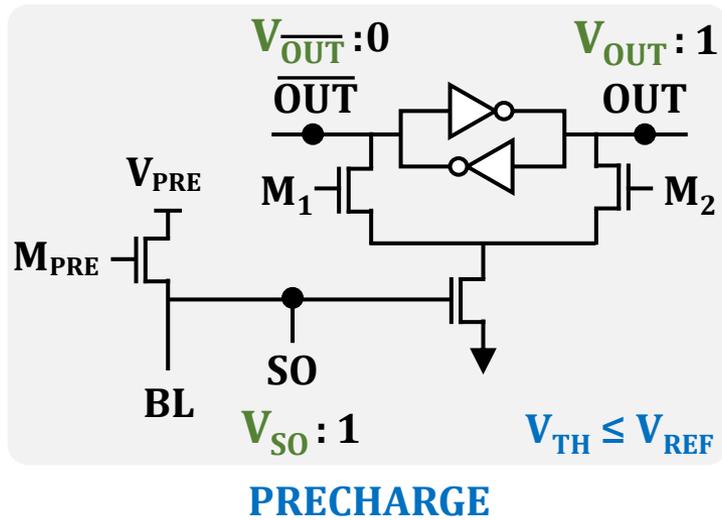
- Discharge



Bitlines are discharged to return the NAND string to its initial state for future operations

Background

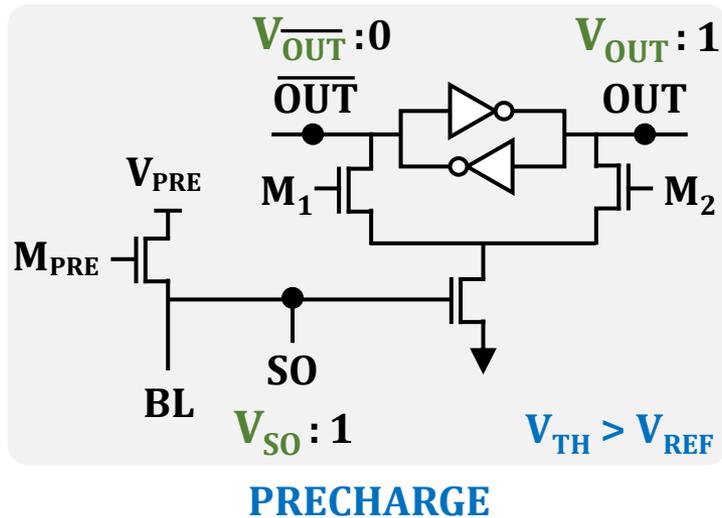
- Latching circuit during read operation



Charge in C_{SO} quickly flows through the NAND string making
 $V_{SO} = 0$ and $V_{OUT} = 1$

Background

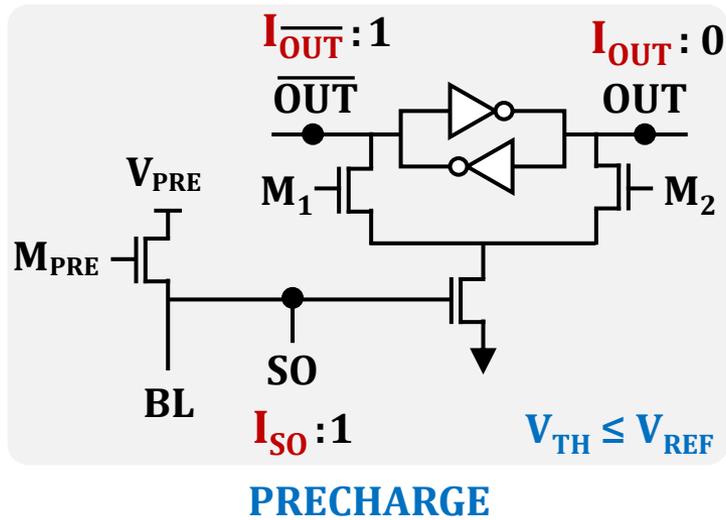
- Latching circuit during read operation



If $V_{TH} > V_{REF}$, C_{S0} cannot discharge due to the target cell acting as an open switch, leading to $V_{S0} = 1$ and $V_{OUT} = 0$

Background

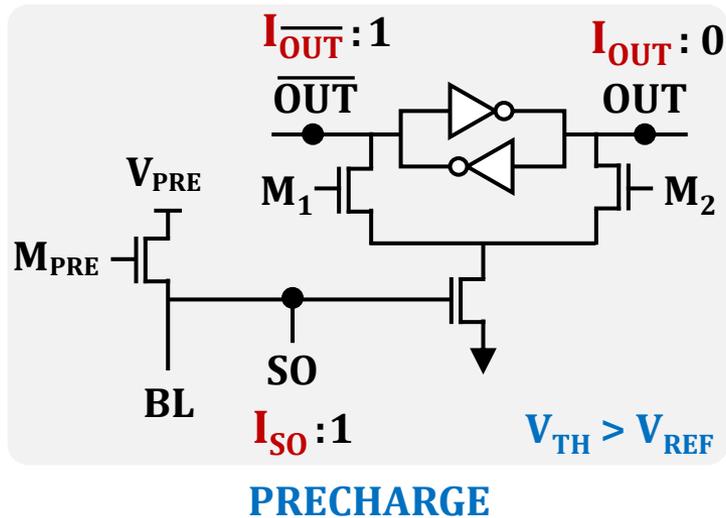
- Latching circuit during **inverse read** operation



Charge in C_{SO} quickly flows through the NAND string making
 $I_{SO} = 0$ and $I_{OUT} = 1$

Background

- Latching circuit during **inverse read** operation



If $V_{TH} > V_{REF}$, C_{S_0} cannot discharge due to the target cell acting as an open switch, leading to $I_{SO} = 1$ and $I_{OUT} = 0$

Evaluation Methodology

Real Host System	CPU: Intel Rocket Lake i7 11700K [91]; x86 [128]; 8 cores; out-of-order; 3.6 GHz;
	Main Memory: 64 GB; DDR4-3600; 4 channels;
Simulated SSD	48-WL-layer 3D TLC NAND flash-based SSD; 2 TB;
	Bandwidth: 8-GB/s external I/O bandwidth (4-lane PCIe Gen4); 1.2-GB/s Channel IO rate;
	NAND Config: 8 channels; 8 dies/channel; 2 planes/dies; 2,048 blocks/plane; 196 (4×48) WLs/block; 16 KiB/page;
	Latencies: tR (SLC mode): 22.5 μs; tMWS: 25 μs (Max. 4 blocks); tPROG (SLC/MLC/TLC mode): 200/500/700 μs; tESP: 400 μs;
Power: HW Accelerator (only in ISP): 93 pJ for 64B operation;	