December 5, 2025

國立臺灣大學
National Taiwan University



Memory and Storage Systems
電腦系統中的無名英雄

Yuan-Hao Chang (張原豪)

CSIE@NTU

Lecturer

Yuan-Hao Chang (張原豪)

- **IEEE Fellow (Class of 2023)**
- Professor, Dept. of CSIE, NTU (2025/8-Now)
- Assistant/Associate/Full Research Fellow, IIS, Academia Sinica (2011/8-2025/7)
- Research Interests: Memory/storage systems, computer systems.
- Publications: 75+ top conf., 75+ ACM/IEEE trans., 30+ US patents

Honor and Awards

1. Best paper awards of top conferences: CODES+ISSS 2024/2022/2019, ISLPED 2020
2. Best paper nominations of top conferences: ISLPED 2023/2022, DAC 2016/2004/2007, CODES 2007
3. OSDI'23 & OSDI'20 papers (17.6%): The most prestigious system conference (only two within 30 years in Taiwan)
4. **國科會傑出研究獎 (2021)、吳大猷紀念獎 (2016)**
5. **國科會中堅計畫 (2025)、中堅計畫 (2021)**

Academic Services:

1. ***Associate Editor-in-Chief of IEEE TETC***, **Associate Editor** of **IEEE TETC**, **ACM TOS**, **ACM TCPS**, and **ACM CSUR**
2. Program Committee of top conferences DAC, ICCAD, ISLPED, CODES, RTSS, RTAS, ICDCS, MASCOTS, etc.
3. Executive Committee Member of **ACM SIGDA** and **ACM SIGBED**.
4. Steering Committee, General Chair (2018) and Program Chair (2017) of IEEE NVMSA
5. Local Chair of IEEE RTSS 2023 and ACM/IEEE ISLPED 2017
6. Conference Chair of ACM/IEEE ESWEEK 2025

原豪的隨手雜記 -【豪豪老實說】
Johnson's Secret Recepits - No Sugar, All Spice!

Personal website: https://www.csie.ntu.edu.tw/~johnson/Notebook.php

Course Information

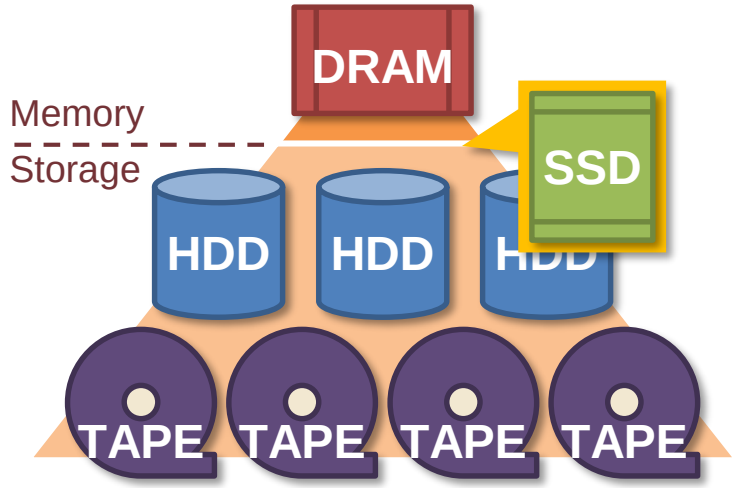
- **Lecturer**:
  - Yuan-Hao Chang (張原豪)
  - Office: R525
  - Phone: +886 2 33664888 ext. 525
  - Email: johnson[at]csie.ntu.edu.tw
- **Teaching Assistant:**
  - Liang-Chi Cheng (陳亮錡)
  - Lab: R438
  - gary0828gary[at]gmail.com

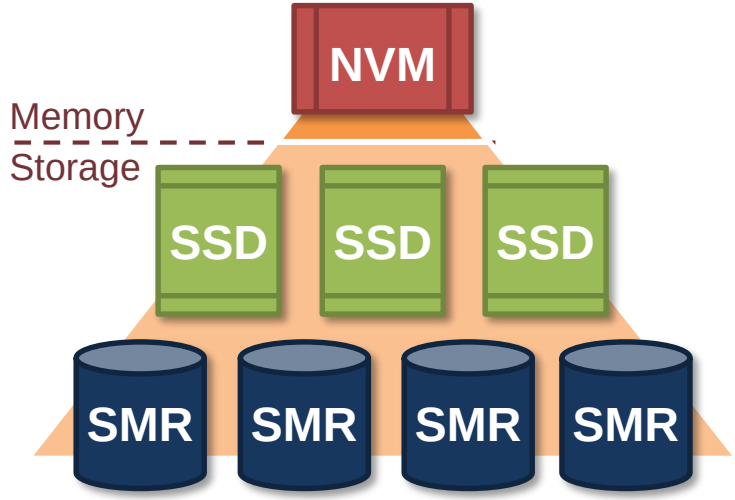
Covered Topics (but not limited to)

- NAND Flash Memory
- Non-volatile Memories
- Next-generation Hard Disk Drive
- Next-generation storage, e.g., Glass storage and DNA storage
- Data index structures: LSM Tree, Be-tree, Bloom filter, Cuckoo hashing, etc.
- File Systems
- Memory Systems
- In-memory, Near-memory, and In-storage Computing
- …

Next-generation Memory/Storage Systems

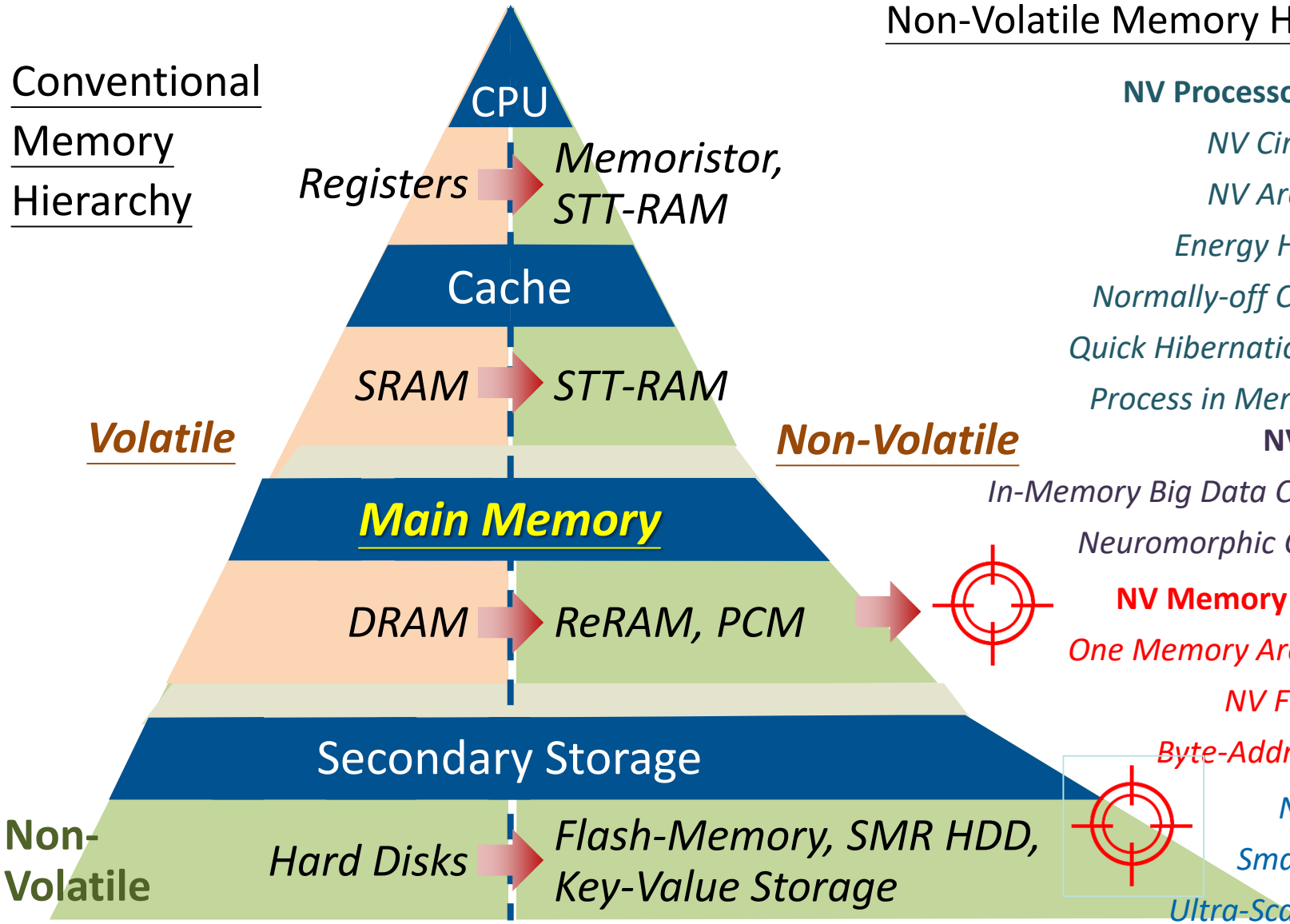
- ***Present***:
  - **DRAM:** Main memory
  - **Solid-state Drive (SSD):** Cache/buffer for high performance environments
  - **Hard Disk (HDD):** Main storage media.
  - **Magnetic tapes:** Deep data archiving
- ***Foreseeable Future***:
  - **Non-volatile memory (NVM):** Enlarge the scalability of in-memory computing
    - Optane DIMM, NVDIMM, Process-in-Memory
  - **SSDs:** Take over the main storage media
    - Reasons: Density ↑ and cost ↓
    - OCSSD, ZNS-SSD, Z-NAND
  - **HDDs** & **Tapes:** Replaced by new magnetic recording technologies
    - Objectives: Density ↑ and cost ↓
    - Promising Candidate: **SMR, IMR**





Memory Hierarchy

Conventional Memory Hierarchy



Non-Volatile Memory Hierarchy

NV Processor & Cache

NV Circuit/Chip,
NV Architecture,
Energy Harvesting,
Normally-off Computing,
Quick Hibernation/Restore
Process in Memory (PIM)

NV Memory

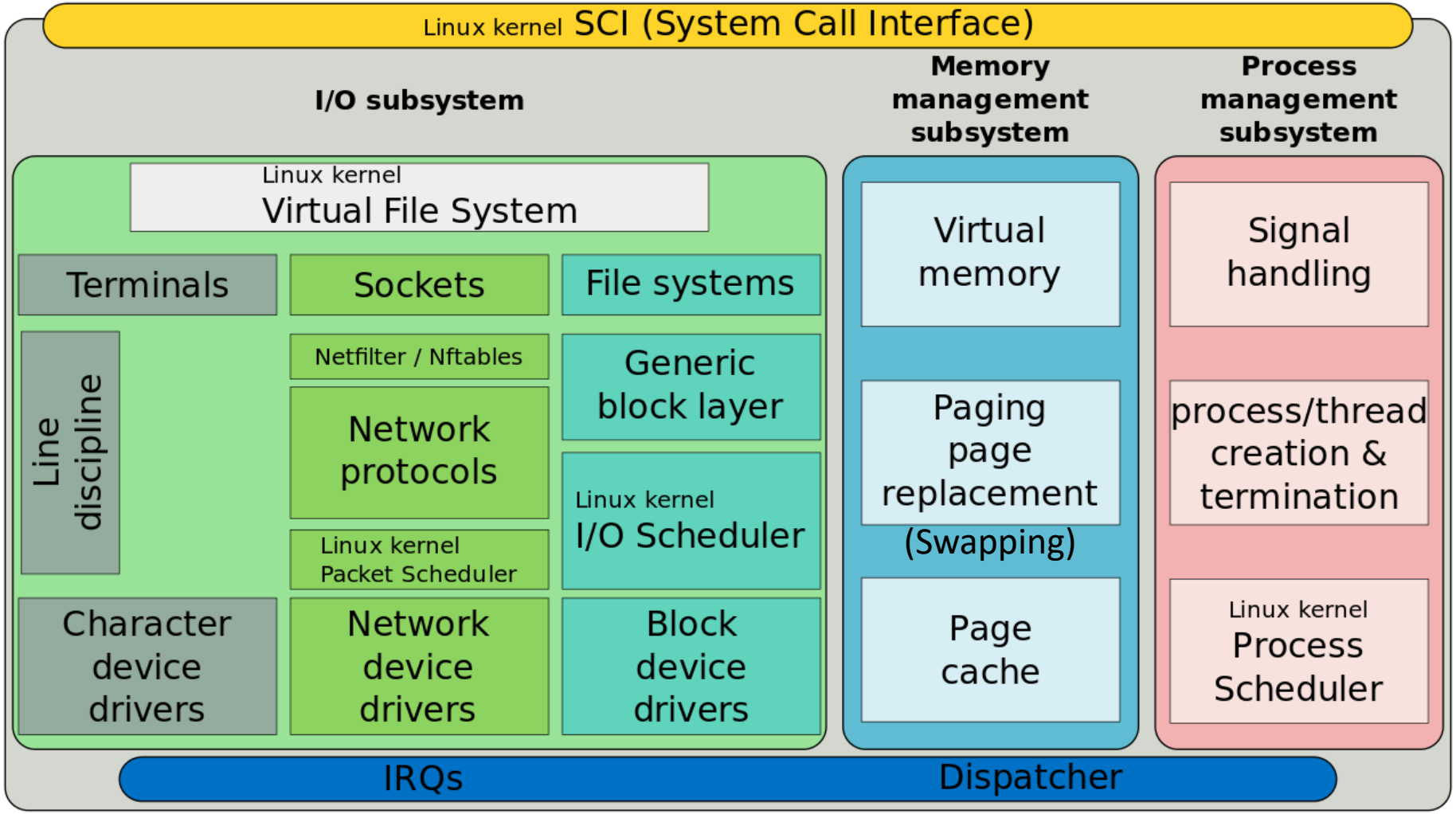
In-Memory Big Data Computing,
Neuromorphic Computing

NV Memory & Storage

One Memory Architecture,
NV File System,
Byte-Addressable FS

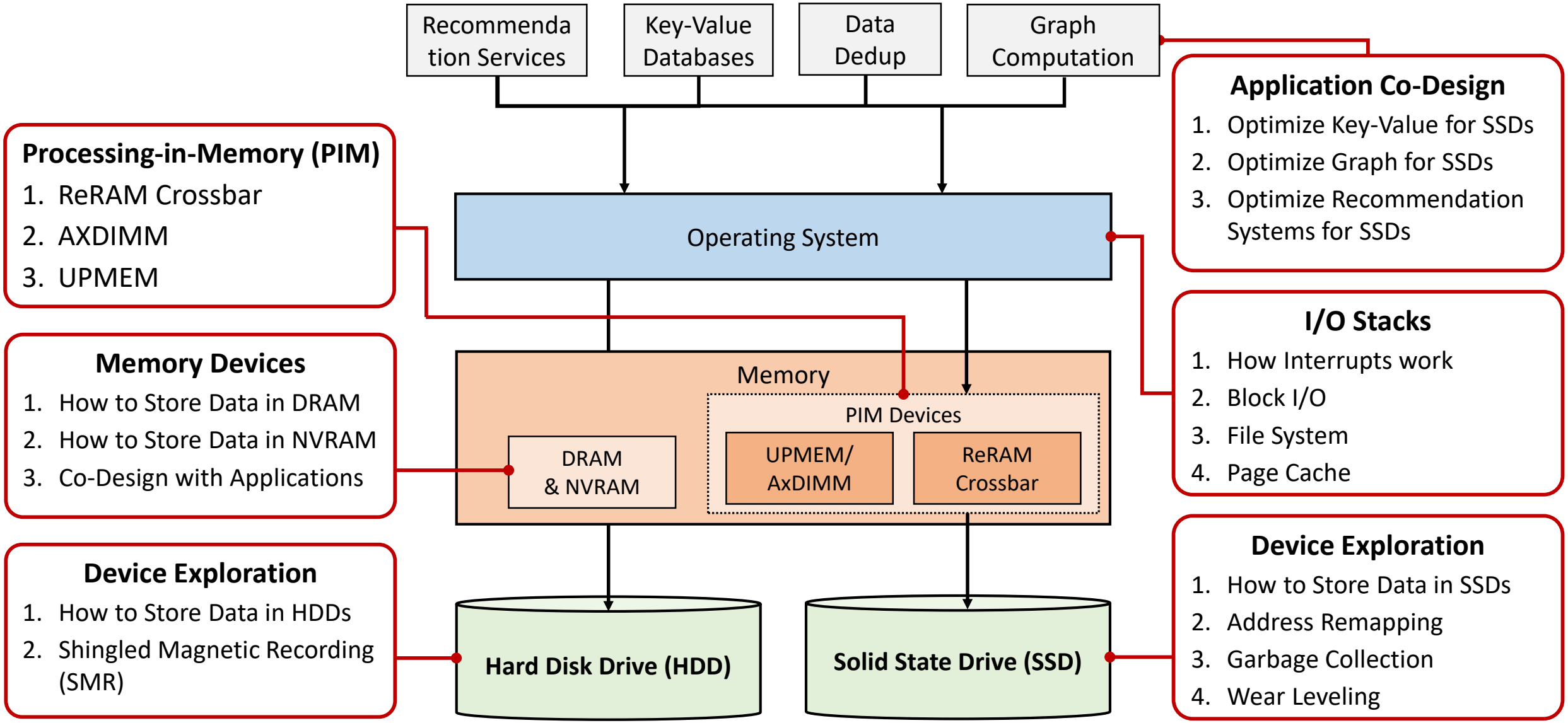
NV Storage
Smart Storage
Ultra-Scale Storage

Memory and Storage System – Linux Kernel Architecture



Swapping vs Page Cache (What's the purpose of Page Cache?)

What Could We Learn (Depend on How Far We can Go)



Processing-in-Memory (PIM)

1. ReRAM Crossbar
2. AXDIMM
3. UPMEM

Memory Devices

1. How to Store Data in DRAM
2. How to Store Data in NVRAM
3. Co-Design with Applications

Device Exploration

1. How to Store Data in HDDs
2. Shingled Magnetic Recording (SMR)

Application Co-Design

1. Optimize Key-Value for SSDs
2. Optimize Graph for SSDs
3. Optimize Recommendation Systems for SSDs

I/O Stacks

1. How Interrupts work
2. Block I/O
3. File System
4. Page Cache

Device Exploration

1. How to Store Data in SSDs
2. Address Remapping
3. Garbage Collection
4. Wear Leveling

國立臺灣大學
National Taiwan University



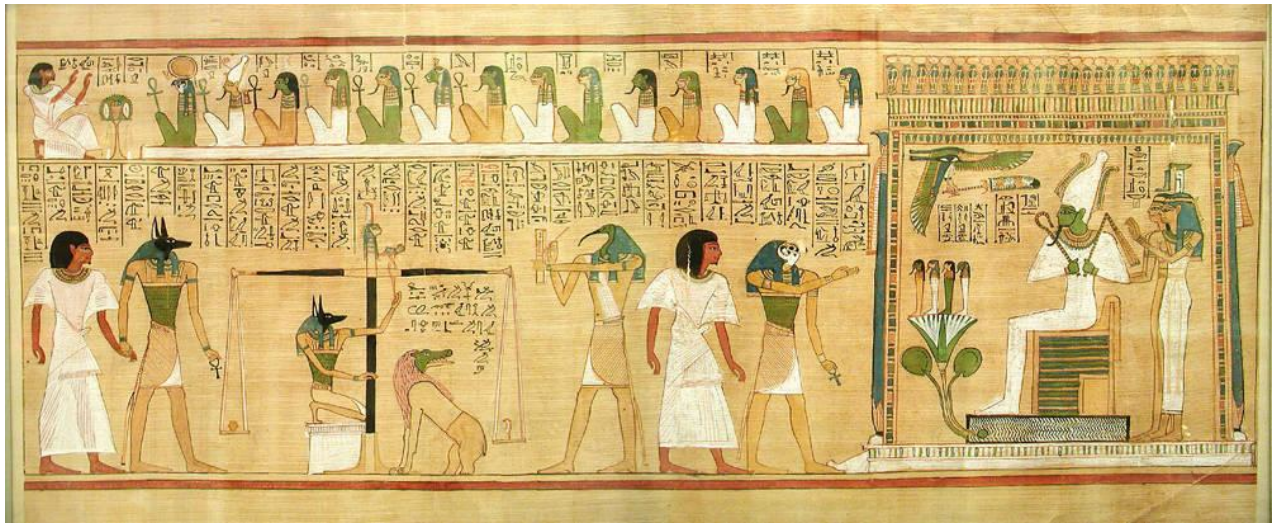
Overview
Memory and Storage Systems are Everywhere in Our Daily Life

Outline

- **Early Data Storage**
- Modern Data Storage
- Data Storage in Daily Life
- Next-Generation Data Storage
- Future Data Storage

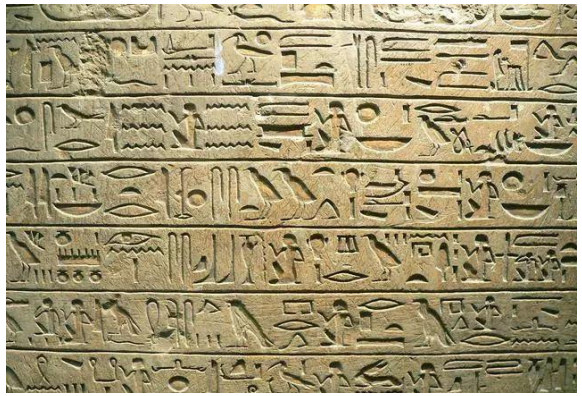
Early Data Storage

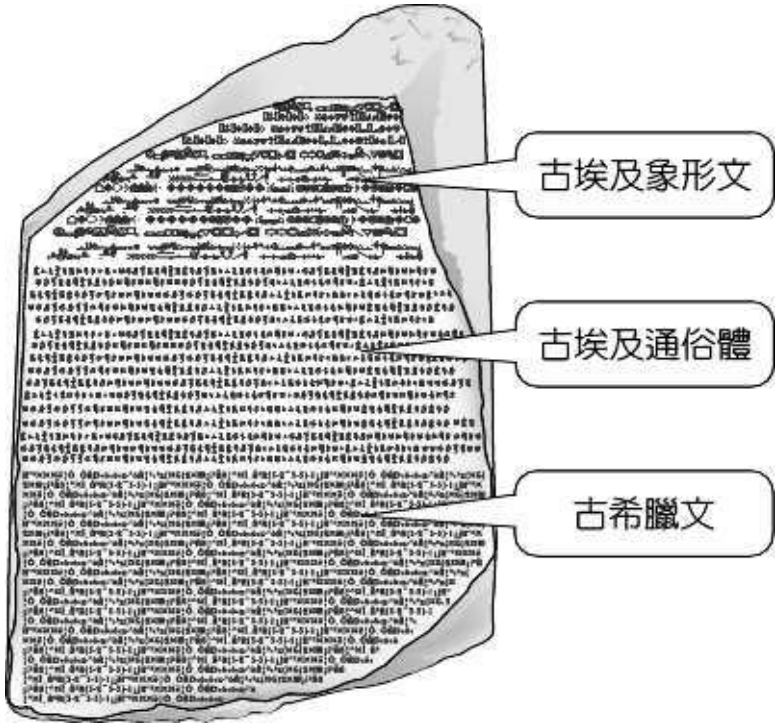
- Egyptian Papyrus (莎草紙): The world's earliest form of paper (3000 BC).
- The original manufacturing method has been lost.
- Though coarse in texture, it was extremely durable and, when well preserved, could last for centuries.

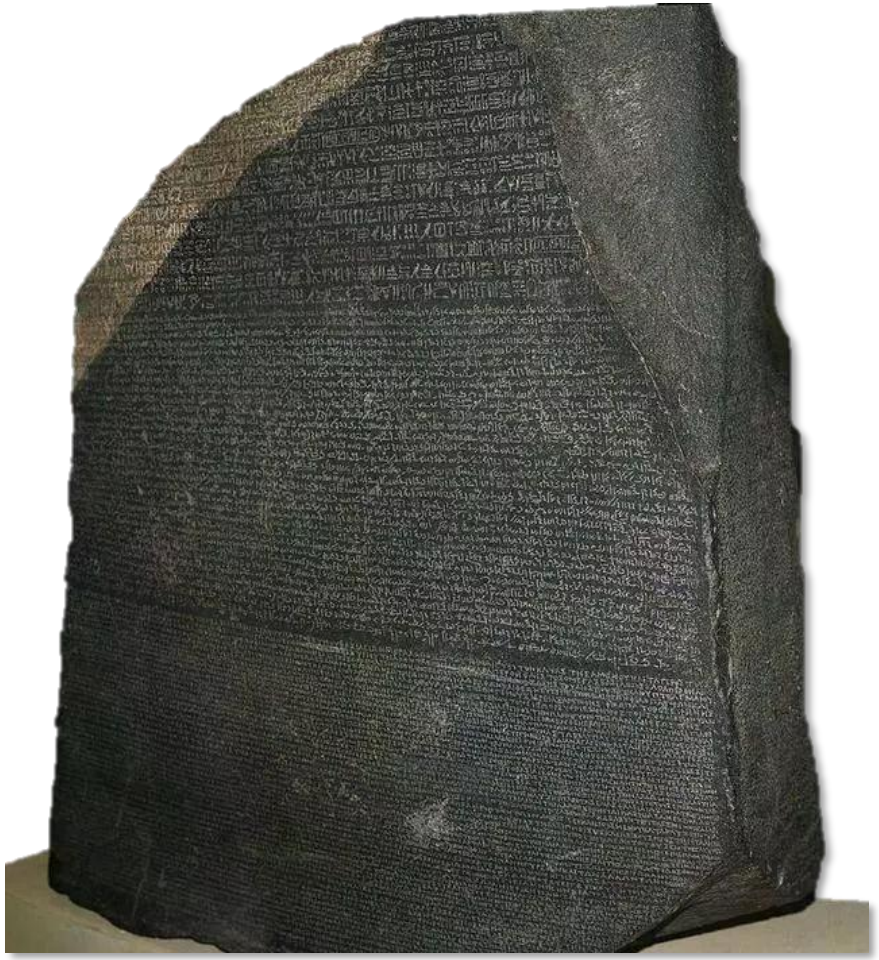


Early Data Storage

- 埃及羅賽塔石碑 Egyptian Rosetta Stone (196 BC):
  - Weighs 762 kilograms.
  - Data density: approximately 1,000 characters per metric ton (1000字/公噸).







Early Data Storage

- 中國竹簡（戰國至西晉，475 BC–316 AD）
- 資料密度：8000 字／公斤

- 「學富五車」：約 800 萬字。
- 「追憶逝水年華」：約 233 萬字。
- 「戰爭與和平」：約 93 萬字。

Early Data Storage

- 中國蔡倫造紙（63 AD–121 AD），為四大發明之一。
- 較莎草紙細緻、較石板與竹簡輕便，保存環境良好亦可耐用上百年。

Outline

- Early Data Storage
- **Modern Data Storage**
- Data Storage in Daily Life
- Next-Generation Data Storage
- Future Data Storage

Modern Data Storage: 1950-1980

1956

5 MB

IBM 305 RAMAC

1961

205 MB

Bryant Model-2 Series 4000

1971

360 KB–1.44 MB

軟性磁碟片 (floppy disks)

https://www.computerhistory.org/timeline/memory-storage/;https://zh.wikipedia.org/zh-tw/%E7%A3%81%E6%B3%A1%E5%AD%98%E5%82%A8%E5%99%A8;
http://s3data.computerhistory.org/brochures/bryant.model2.1965.102646212.pdf;
https://www.i-garden.org/blog/2009/12/02/%e5%84%b2%e5%ad%98%e5%aa%92%e9%ab%94%e6%bc%94%e8%ae%8a%e6%ad%b7%e5%8f%b2-storage-media-history-2-%e7%a1%ac%e7%a2%9f%e6%ad%b7%e5%8f%b2%e5%9c%96%e7%89%87-hard-disk-history/; https://www.ibm.com/ibm/history/ibm100/us/en/icons/floppy/transform/

Modern Data Storage: 1980-2000

1980

5 MB

Seagate ST506 硬式磁碟機

1980s

550MB

光碟片 (compact discs)

1990s

20 MB

固態硬碟 (solid state disk)

https://www.istockphoto.com/hk/%E5%9C%96%E7%89%87/compact-disc;https://www.ibm.com/ibm/history/ibm100/us/en/icons/floppy/transform/; https://en.wikipedia.org/wiki/Solid-state_drive

Modern Data Storage: 2020-2010

2000

4 GB

SanDisk
USB Flash Drives

2007

1 TB

Hitachi
First 1TB HDD

Shared Storage (Pay on Need)

Dropbox
Cloud Storage Service

2009

Customized by Customers

Cloud-based
network-attached
storage solutions

https://www.computerhistory.org/timeline/memory-storage/

Modern Data Storage: 2020-2020

Samsung V1–850
3DFirst 3D flash chip (32 layers)

Booming stage:
More emerging memory and storage media

https://www.cs.odu.edu/~tkennedy/cs300/development/Public/M01-HistoryOfComputerMemory/index.html;
https://www.nvmdurance.com/history-of-3d-nand-flash-memory/; https://www.nvmdurance.com/history-of-3d-nand-flash-memory/; https://news.xfastest.com/news/2477/2477/

Modern Data Storage: 2020-

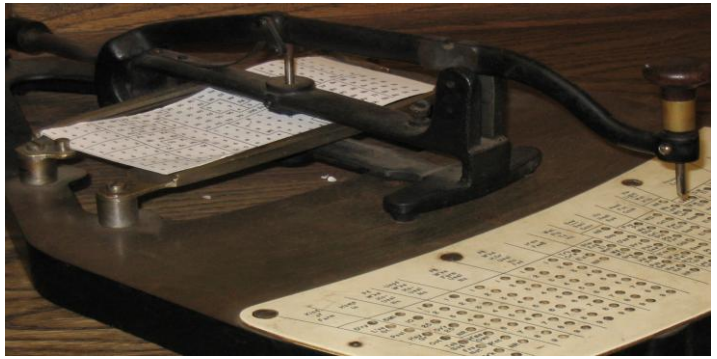
- In 2020, GitHub, the world's largest open-source platform, preserved humanity's most important code repositories in magnetic tape and stored them in an underground vault in the Arctic Circle—"The GitHub Arctic Code Vault (北極代碼庫)"—with the goal of safeguarding human knowledge permanently.
- *After several hundred years, no data stored in modern IC-based devices will survive.*

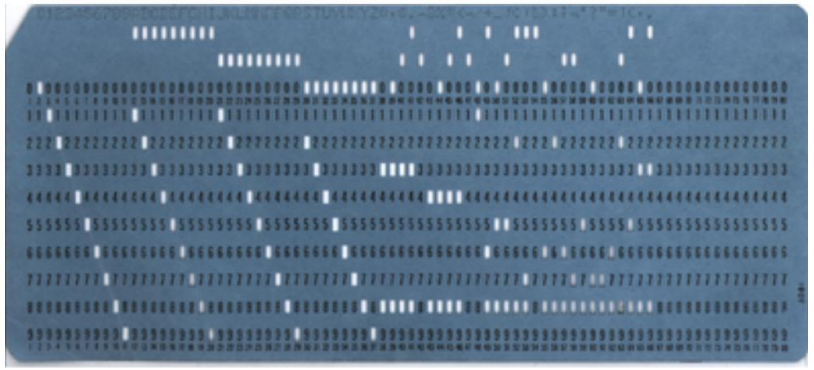
Outline

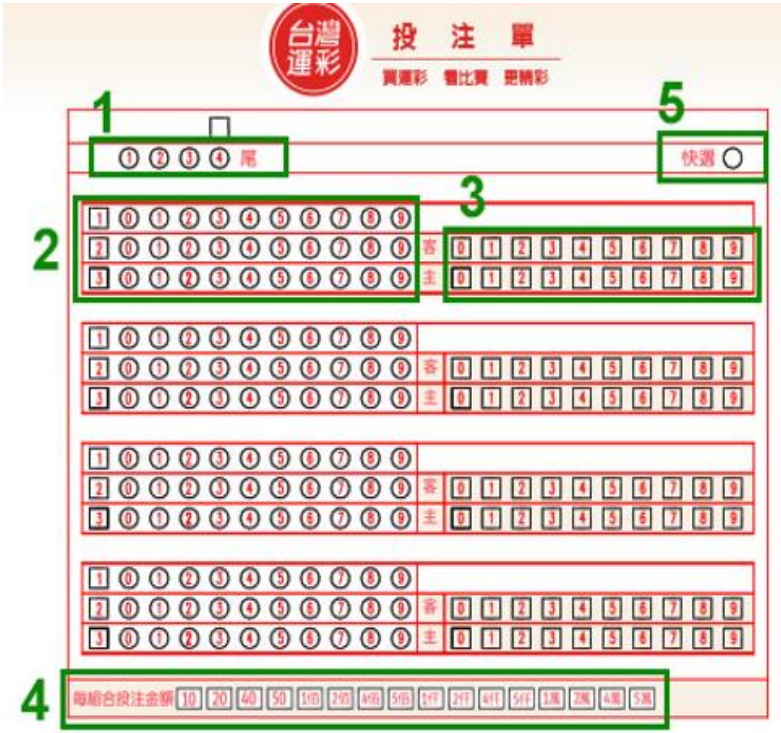
- Early Data Storage
- Modern Data Storage
- **Data Storage in Daily Life**
- Next-Generation Data Storage
- Future Data Storage

Data Storage in Daily Life – Punched Card

- Digital information was represented using holes and the absence of holes. It was first invented in the 1880s and applied in the U.S. Census.
- Early computers also used punched cards as the primary medium for program and data input.
- Although no longer in use, their design evolved into the optical mark recognition (OMR) cards commonly used today for lottery tickets and examinations.







Data Storage in Daily Life – Magnetic Tape

- Data is recorded using magnetic particles, allowing storage of various types of files.

Tapes for audio

Tapes for videos

Tapes in data centers (1980's)

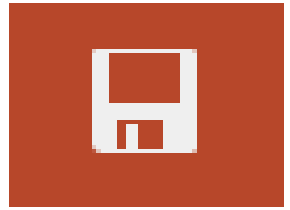
https://zh.wikipedia.org/zh-tw/%E7%A3%81%E5%B8%A6; https://commons.wikimedia.org/wiki/File:AlbanyStateUniversity_ComputerOperationsRoom_1981_3_skaliert.jpg

Data Storage in Daily Life – Magnetic Tape (Cont.)

- Even today, technology giants such as Google still use magnetic tapes to back up massive amounts of data.
  - Low cost
  - Long lifespan
  - Easy to preserve
  - High security

Data Storage in Daily Life – Floppy Disk

- Data was also recorded using magnetic particles, making it one of the important devices in early computers.
- However, due to its limited storage capacity (only 1.44 MB per disk) and slow read/write speed, it was gradually phased out.



Many software applications still use the image of a floppy disk as the icon for “Save File.”

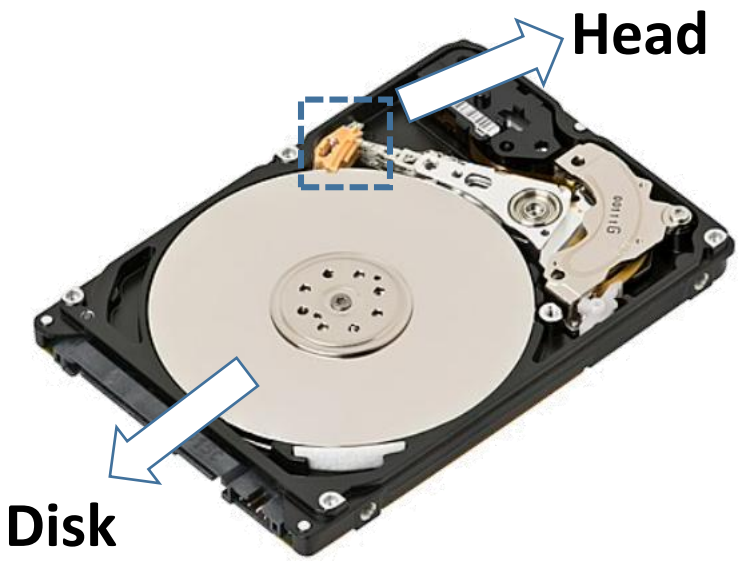
Data Storage in Daily Life – Optical Disk

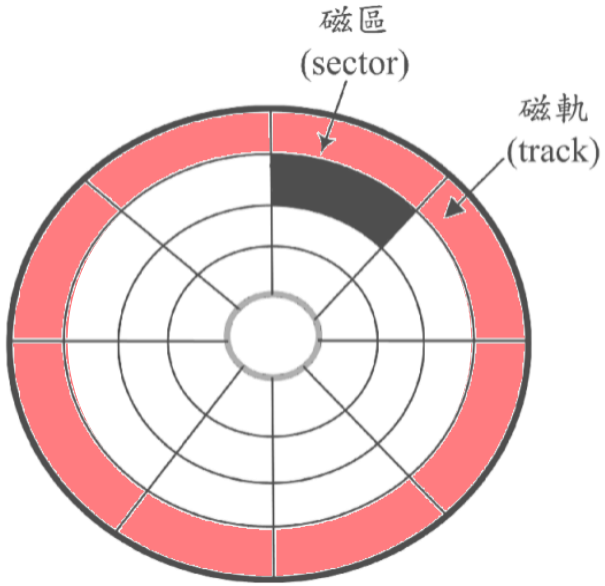
- Data is recorded and read using optical methods.
- The first generation of optical discs was the CD (compact disc, laser disc), which we are all familiar with.
- CDs were developed to store digital audio and are still the standard medium for commercial recordings today.
- VCDs were introduced as the standard for storing video data on CDs.
- Later, the second and third generations of optical disc technologies were developed: DVD (digital versatile disc) and Blu-ray Disc, which support higher resolutions and offer larger storage capacities.

https://zh.wikipedia.org/zh-tw/%E5%85%89%E7%A2%9F; https://zh.wikipedia.org/zh-tw/%E9%9B%B7%E5%B0%84%E5%94%B1%E7%89%87; https://zh.wikipedia.org/zh-tw/DVD; https://zh.wikipedia.org/zh-tw/%E8%97%8D%E5%85%89%E5%85%89%E7%A2%9F

Data Storage in Daily Life – Hard Disk Drive, HDD

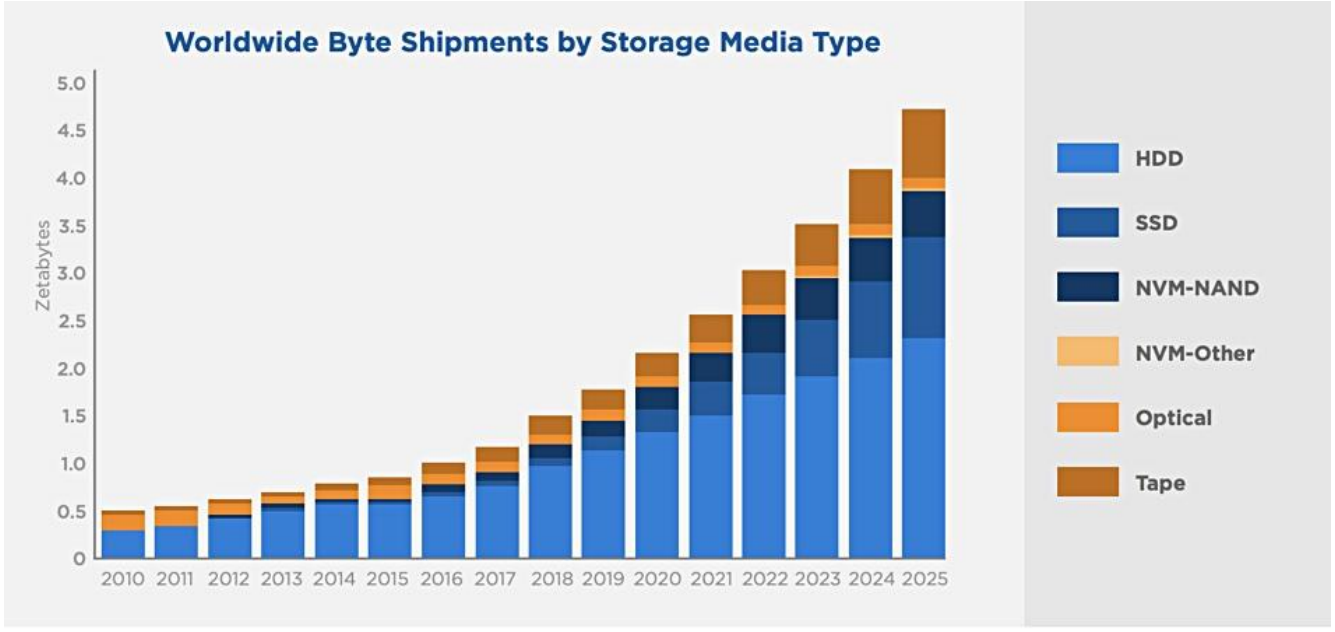
- Data are recorded using magnetic particles on the surface of the disk.
- With capacities reaching several terabytes, it is one of the mainstream storage devices today.



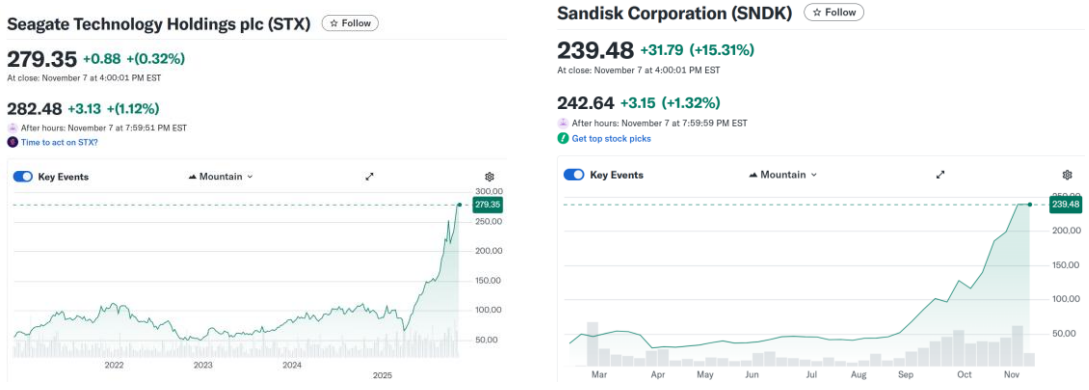


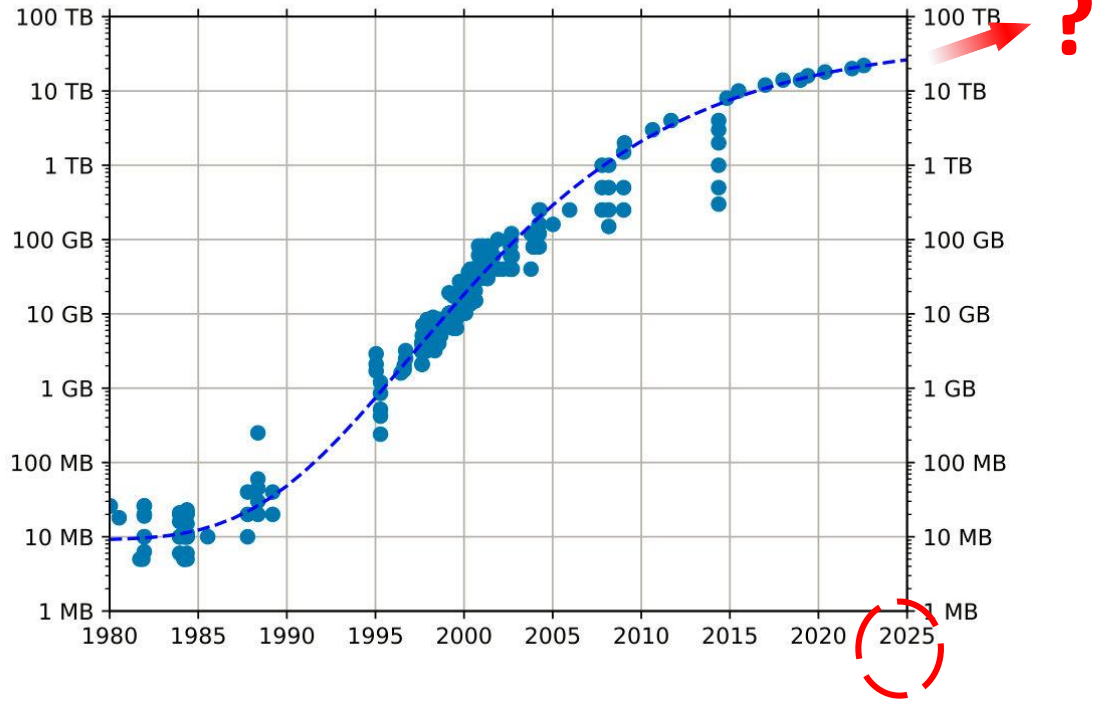
HDD Capacity Trend

- The capacity of HDDs increased at breakneck speed in past years.



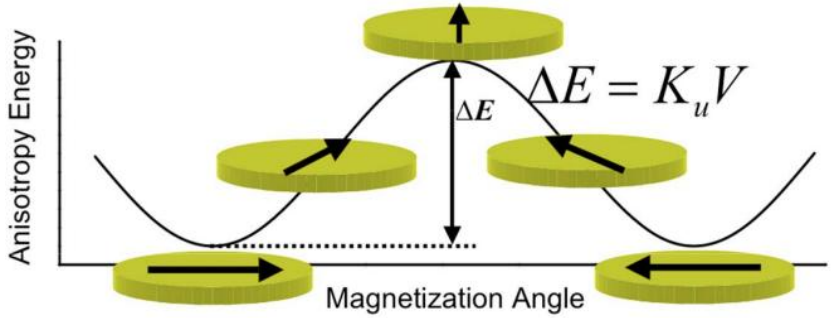
Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018



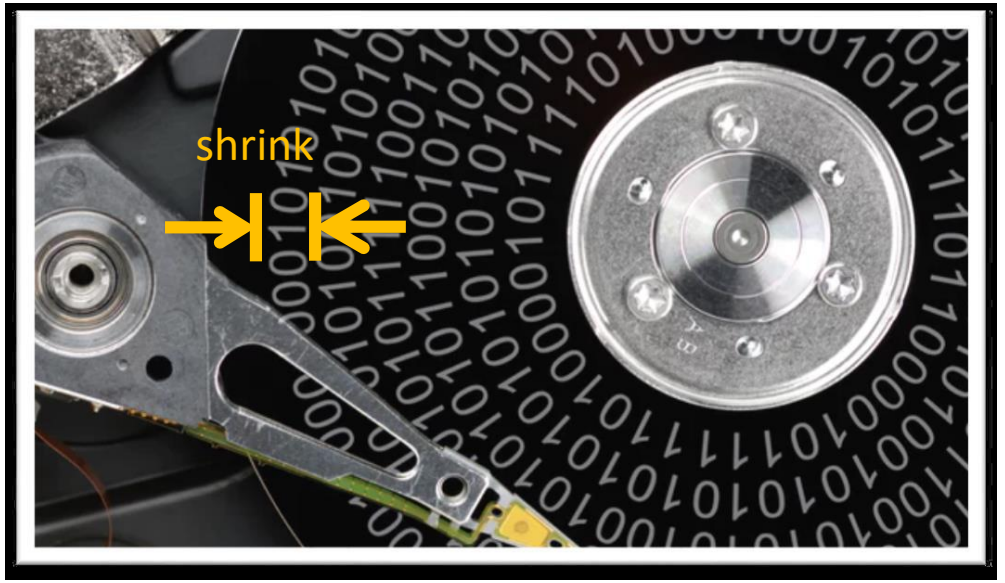


Shrinkage Bottleneck

- PMR has reached the **bottleneck** in providing higher areal density.
  - The maximal areal density is **1 TB per square inch**.
  - Because of the **superparamagnetic effect (SPE, 超順磁效應),** it is hard to shrink the volume of magnetic grains because it is easily be disturbed by thermal.
    - When the volume of a magnetic grain becomes sufficiently small, thermal fluctuations can cause its magnetization direction to spontaneously flip, leading to data loss or corruption. This phenomenon is known as the **superparamagnetic effect**.



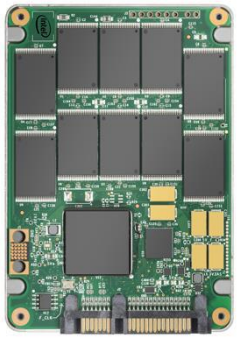
磁異向能(anisotropic energy)



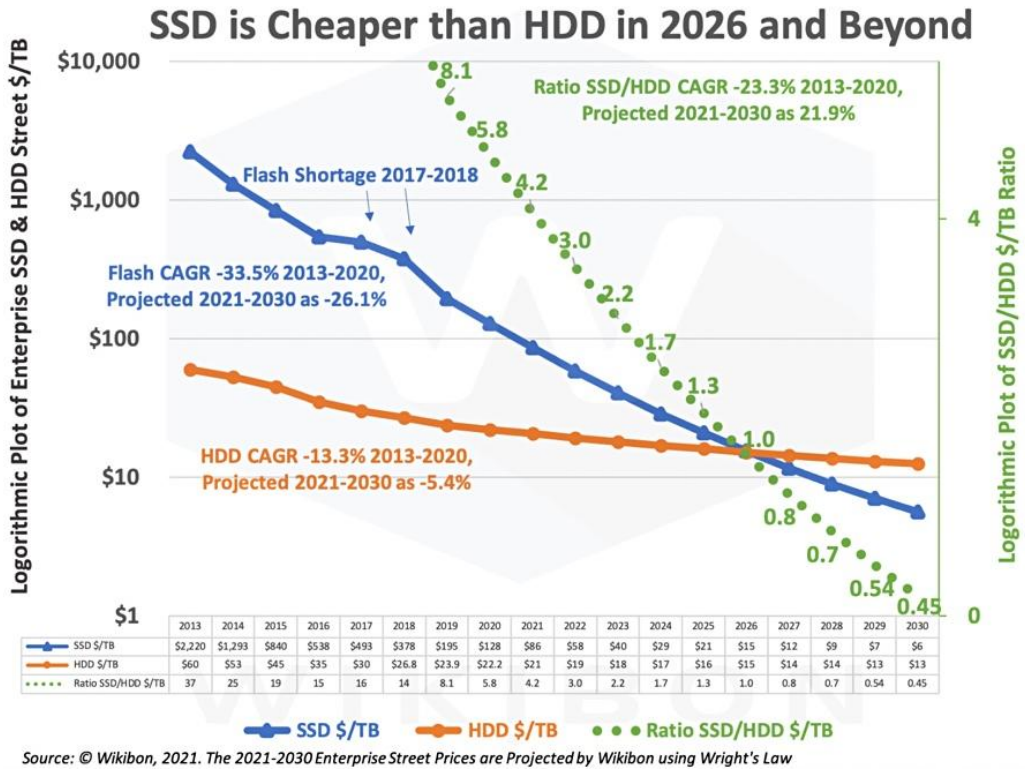
Breaking the Bottleneck or Dying...

- **Strong Rival**: Solid-State Drive (SSD)
  - **Flash memory** demonstrates several good advantages …
  - Flash memory is getting **cheaper** with **degraded reliability**.

SSD



- ✓ Faster
- ✓ Silent
- ✓ Shock-resistant
- ✓ Energy efficient
- ✓ Lighter
- ✓ Smaller



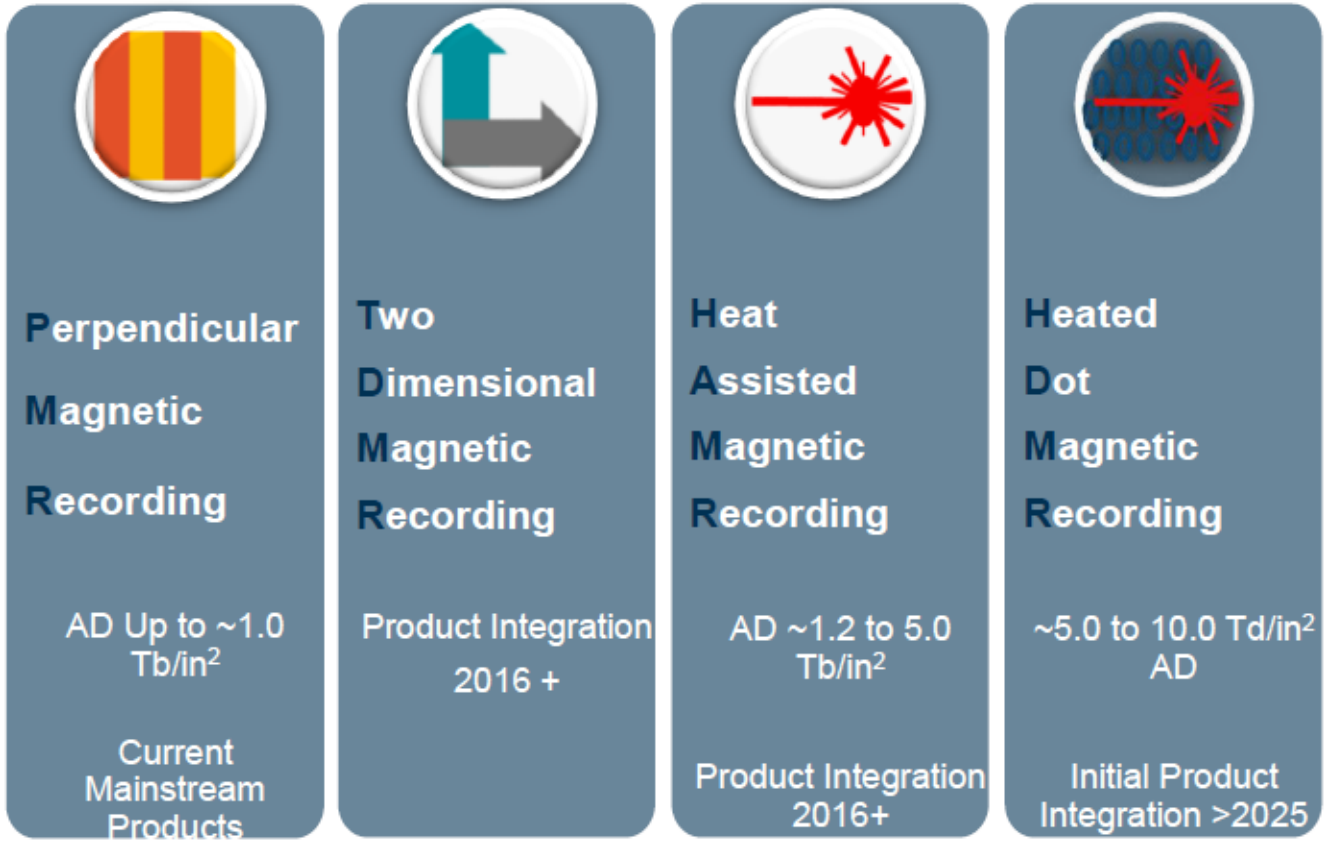
Source: © Wikibon, 2021. The 2021-2030 Enterprise Street Prices are Projected by Wikibon using Wright's Law

HDD

- ✓ Cheaper?
- ✓ Reliability

New Magnetic Recording Technologies

- HDDs are poised to keep evolving for the **increased areal density (AD)** with various new technologies:
  - **Less** things to do with firmware/software
-

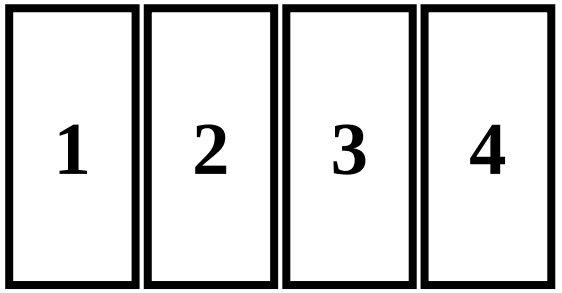


https://www.anandtech.com/show/10470/the-evolution-of-hdds-in-the-near-future-speaking-with-seagate-cto-mark

New Track Layouts

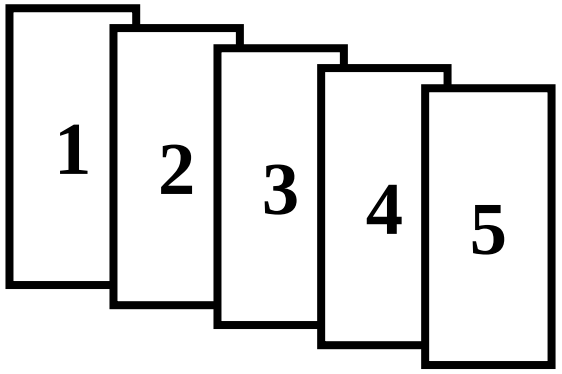
- HDDs are poised to keep evolving for the **increased areal density (AD)** with different track layouts:
  - **More** things to do with **firmware/software**

Conventional magnetic recording



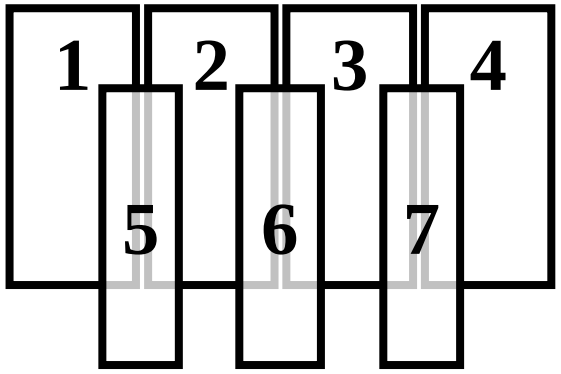
- ✓ Tracks **non-overlap**
  - ➢ **Tracks can be rewritten freely**

Shingled magnetic recording



- ✓ Tracks **overlap**
- ➢ **25% higher capacity** than CMR
- ➢ **Commercially available now**
- ➢ **Track rewrite issue**

Interlaced magnetic recording



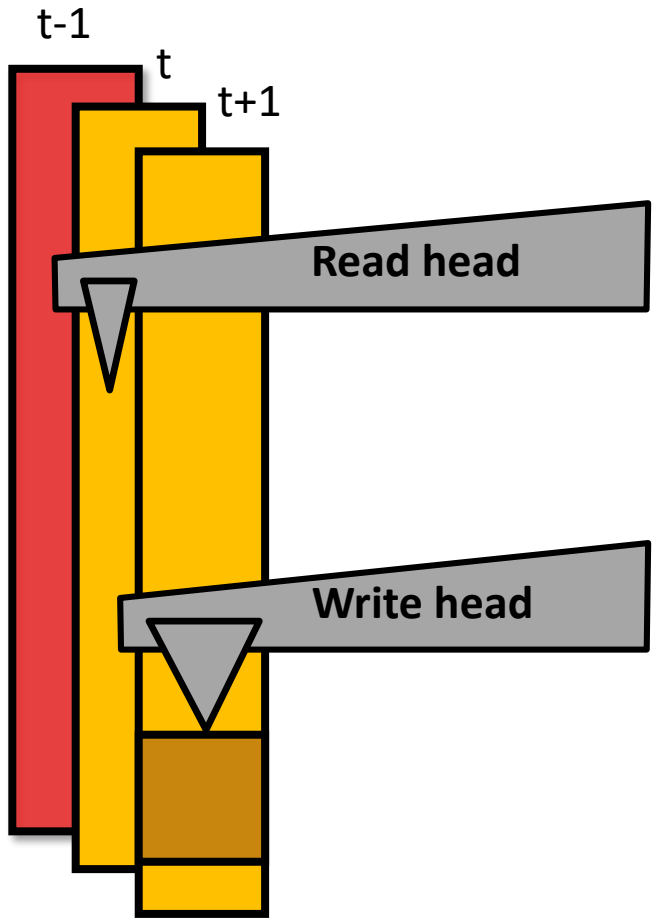
- ✓ Tracks **overlap**
- ➢ **40% higher capacity** than CMR
- ➢ **Not commercially available yet**
- ➢ **Track rewrite issue**

Shingle?

- https://www.youtube.com/watch?v=O8ZP-jIf8bs

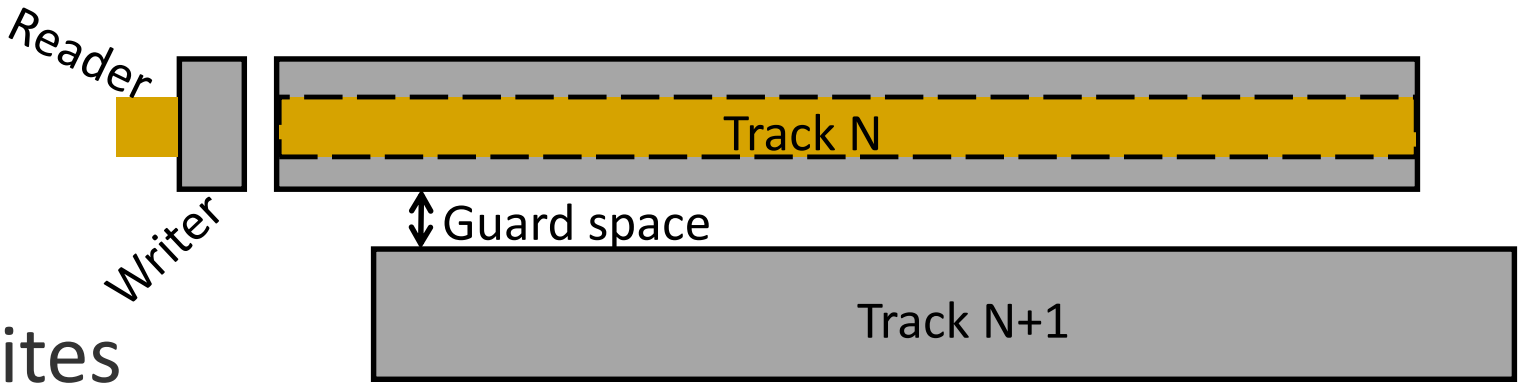
Shingled Magnetic Recording (SMR)

- **Key**: Read head is **more precise** than write head.
  - SMR is based on
    - **Writing** in a sequential way with tracks overlapped with the previous ones.
    - **Reading** the "exposed" data from shingled tracks.
  - **Advantages** of Shingled Tracks:
    - Areal density↑, Capacity↑, and Cost↓
    - No major changes of disk are needed.
  - **Design Challenge**:
    - Updating data to an existing track may destroy data on the subsequent tracks.
    - SMR management is needed.

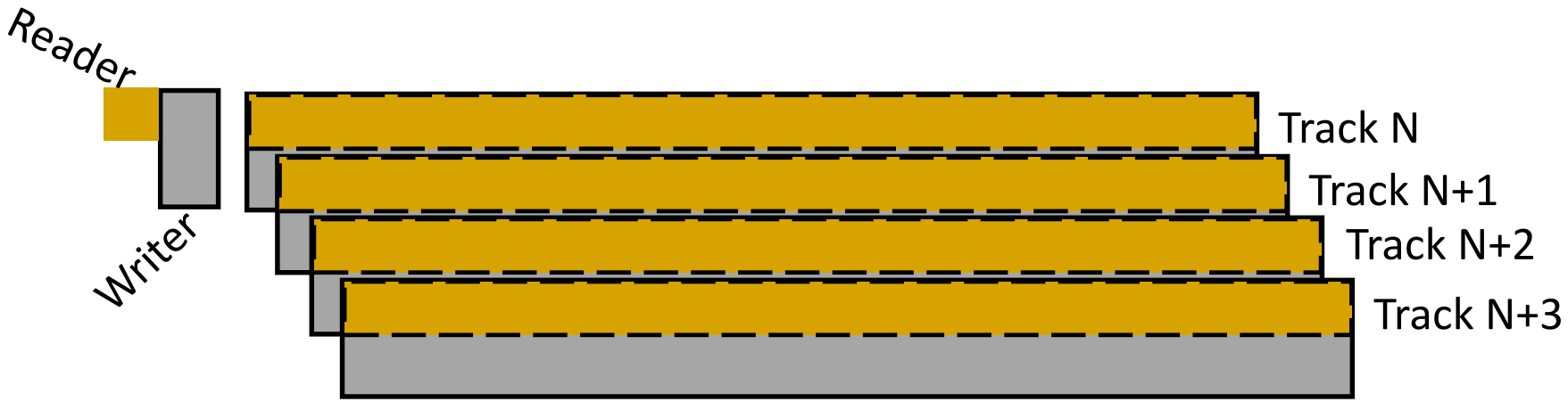


Conventional vs. Shingled Writes

- Conventional Writes



- SMR Writes

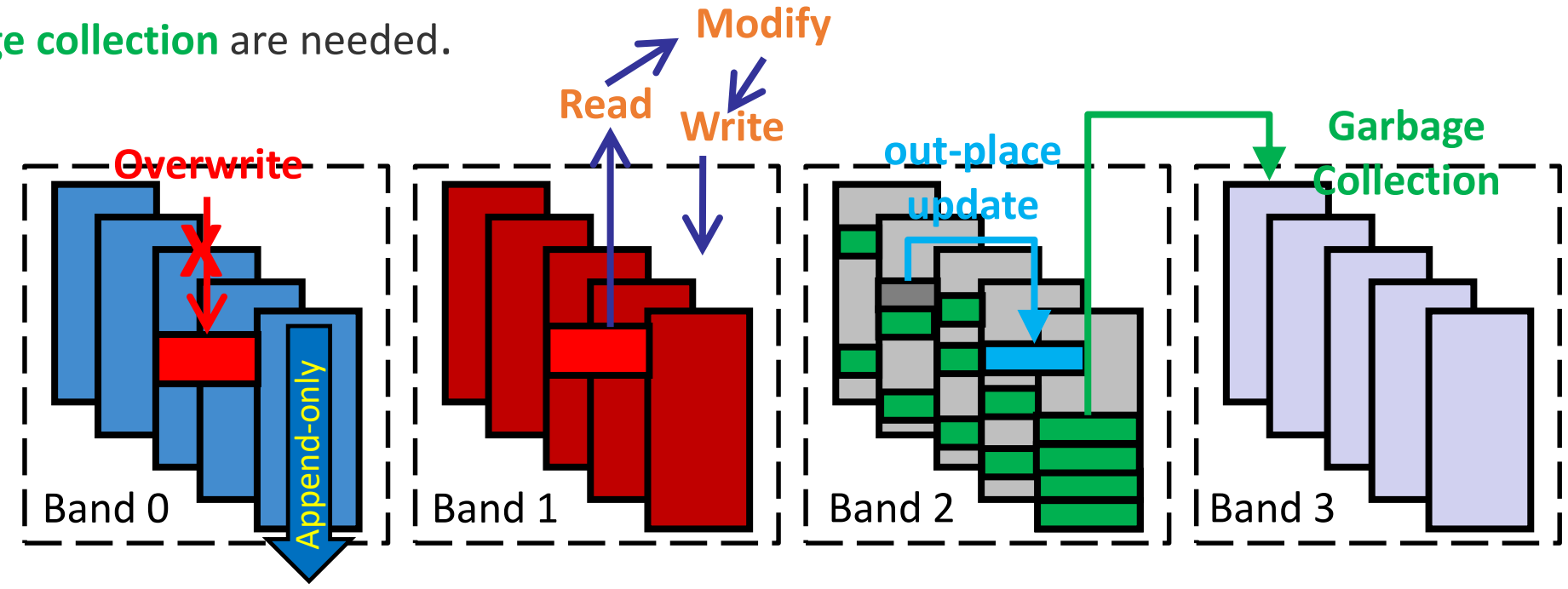


SMR Inherent Challenges

- **Constraint**: Writes to SMR must be strictly append-only on the current write position (i.e., write pointer).
  - **Band-based management** can mitigate this challenge.
  - **Overwrite** is still prohibited in a band.

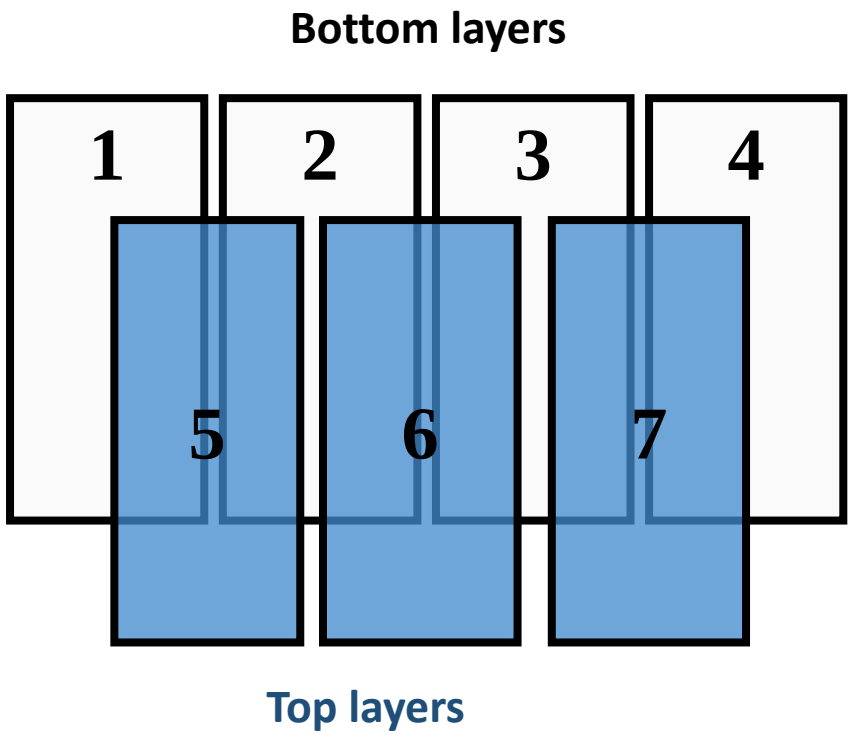
Approach 1) **Read-modify-write (RMW)** is expensive over a band.

Approach2) Although **out-place-update** may serve updates more efficiently, **address mapping** and **garbage collection** are needed.



Interlaced Magnetic Recording

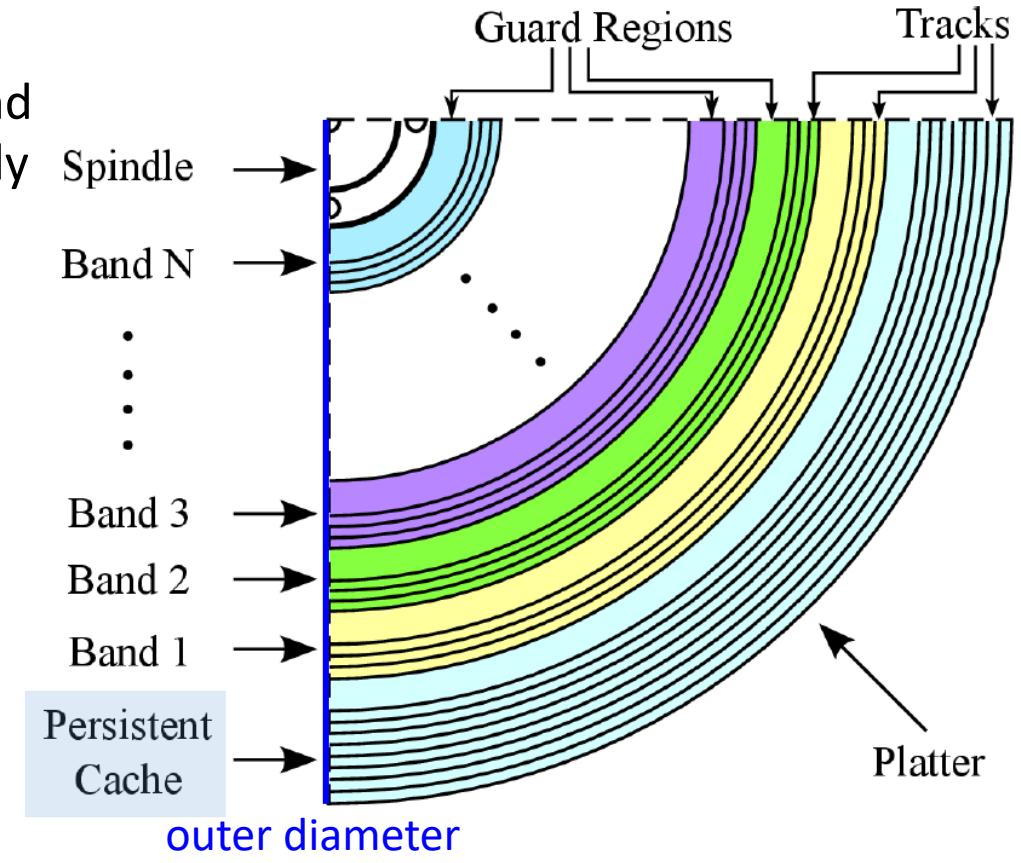
- Half of the tracks overlap
  - Bottom tracks are overlapped by top tracks
- Top tracks are narrower
  - Hold 80% - 90% as much data
    - No in-place updates are allowed for bottom tracks
      - Solution: RMW or using a translation layer
        - Track-based Translation Layers for Interlaced Magnetic Recording [ATC '19]



General Solution to Non-Seq. Writes

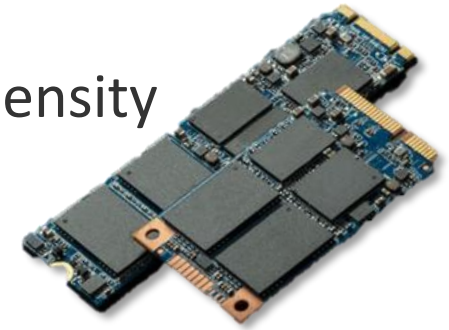
- **Persistent Cache**: A small region to **stage** non-sequential writes to all bands (via out-place updates).
  - **Non-sequential Write**: Data are not written to the current "write pointer" of a **band** (also called a **zone**).

  - It can postpone updates and reduce the number of costly RMWs to bands.
  - It can be made by any persistent storage technology such as:
    - **Disk itself**: SMR tracks at outer diameter (OD).
    - **Flash memory**; or
    - **Non-volatile memory**.



Data Storage in Daily Life – NAND Flash Memory

- Flash memory is currently the mainstream storage medium widely used in embedded devices, smartphones, tablets, personal computers, and even enterprise-level storage systems.
- **Features**
  - Nonvolatility: Like mechanical hard drives, it can retain data even when power is off.
  - High access performance: Significantly higher read and write performance than mechanical hard drives.
  - Low power consumption: Much lower operating energy consumption than mechanical hard drives.
  - Highly shock-resistant electronic storage medium: With no moving parts, it is more resistant to shocks compared to mechanical hard drives.
- **Development Trends**
  - From single-level cell (SLC) to multi-level cell (MLC): increases storage density and reduces cost.
  - From planar to 3D architecture: further increases storage density.



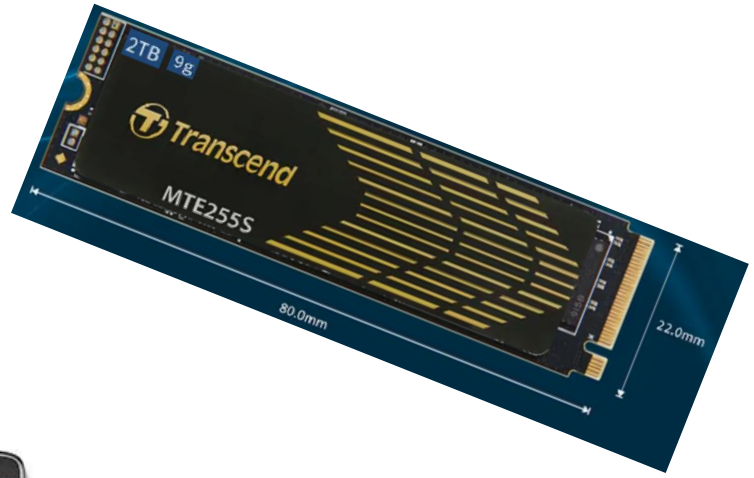
Data Storage in Daily Life – NAND Flash Memory

- Various flash memory devices: memory cards, USB flash drives, and solid-state drives (SSDs).
  - Memory Cards
  - USB flash Drives
  - SSDs

https://tw.buy.yahoo.com/gdsale/SanDisk-Extreme-Pro-SDXC--8110249.html?from=pla&gg=0&co_servername=Pmax_peripheral_220811&gclid=CjwKCAjwqJSaBhBUEiwAg5W9pywJ7aw-R5Tan8cqS__ZbxCZEksK-Qmt6FI4BA1s64SzolrRtKQGpBoCoKgQAvD_BwE&gclsrc=aw.ds;
https://www.momoshop.com.tw/goods/GoodsDetail.jsp?i_code=10384102&osm=Ad07&utm_source=googleshop&utm_medium=traffic_pla&utm_content=bn&gclid=CjwKCAjwqJSaBhBUEiwAg5W9p5i3Pi8YHtovA-9vf4TE55ebKeOVo646smOkg38TiiEo5w3CHpRWmhoC8V4QAvD_BwE;
https://tw.buy.yahoo.com/gdsale/SanDisk-Extreme-PRO-USB-3-2-%E5%9B%BA%E6%85%8B%E9%9A%A8%E8%BA%AB%E7%A2%9F1TB-%E5%85%AC%E5%8F%B8%E8%B2%A8-9462457.html?from=pla&gg=0&co_servername=Pmax_peripheral_220811&gclid=CjwKCAjwqJSaBhBUEiwAg5W9pyeDl2PWBGzczBqmRe7sbOUkhPquXYfgdJRHSoq5jsB-afPckOfNUhoCREwQAvD_BwE&gclsrc=aw.ds;
https://www.momoshop.com.tw/goods/GoodsDetail.jsp?i_code=10242621&osm=Ad07&utm_source=googleshop&utm_medium=traffic_pla&utm_content=bn&gclid=CjwKCAjwqJSaBhBUEiwAg5W9p0TIMRGkQZ5dN814VC6wm4jrITDltOXDzPOc4NleDvWWEbt7uC1-bxoCpiYQAvD_BwE;
https://www.momoshop.com.tw/goods/GoodsDetail.jsp?i_code=9573274&Area=search&mdiv=403&oid=1_5&cid=index&kw=SSD; https://www.momoshop.com.tw/goods/GoodsDetail.jsp?i_code=5619813&Area=search&mdiv=403&oid=1_15&cid=index&kw=SSD

Flash Memory

- Flash memory is a widely used memory/storage technology in today's products
  - E.g., USB drives, solid-state drives and mobile devices
- Flash memory is a type of non-volatile memory
  - It holds data states even when the power is lost.

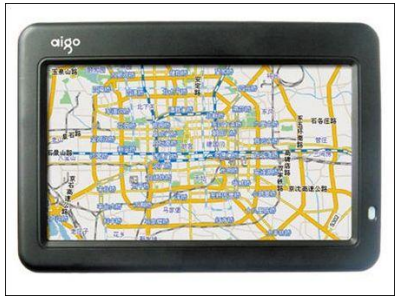


Why Flash Memory

- Diversified application domains
  - Portable storage devices
  - Consumer electronics
  - Industrial applications
  - Avionics / aerospace
- Nice features:
  - Shock-resistance
  - Non-volatility
  - Low energy consumption (Compared to HDD)
  - Small feature size
  - Low cost?

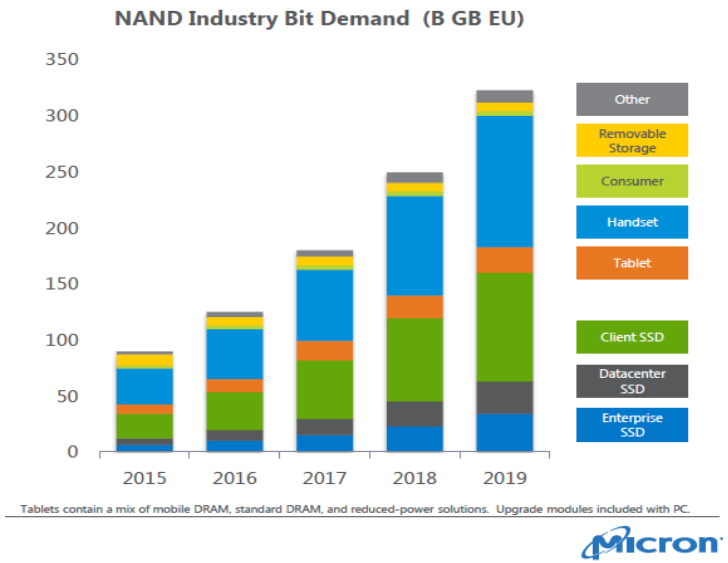




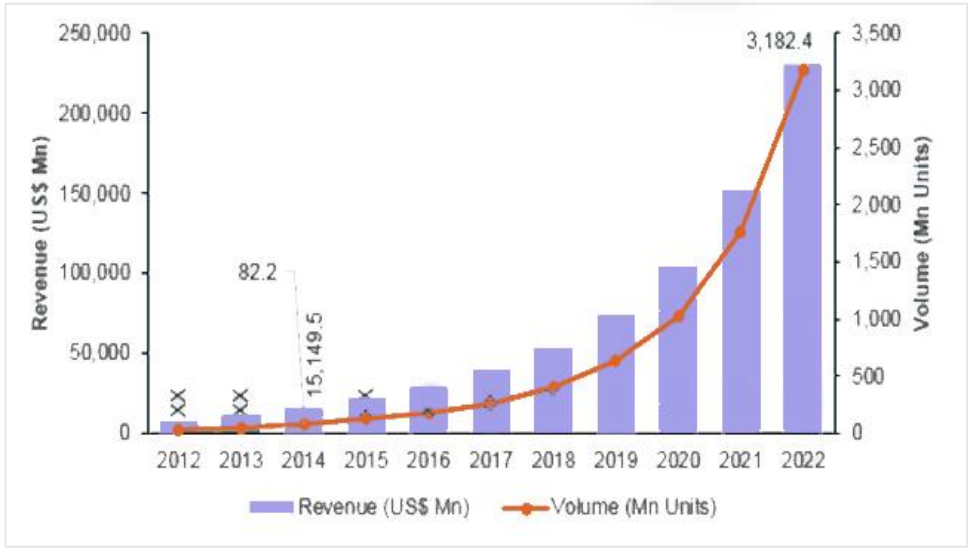



Trends in NAND Flash Memory

- Flash memory is widely adopted in various systems
- Price : **~40%** annual price reductions every year
- Revenue growth : **40%** every year



Sources : Micron, Intel And 3D NAND Post [link]

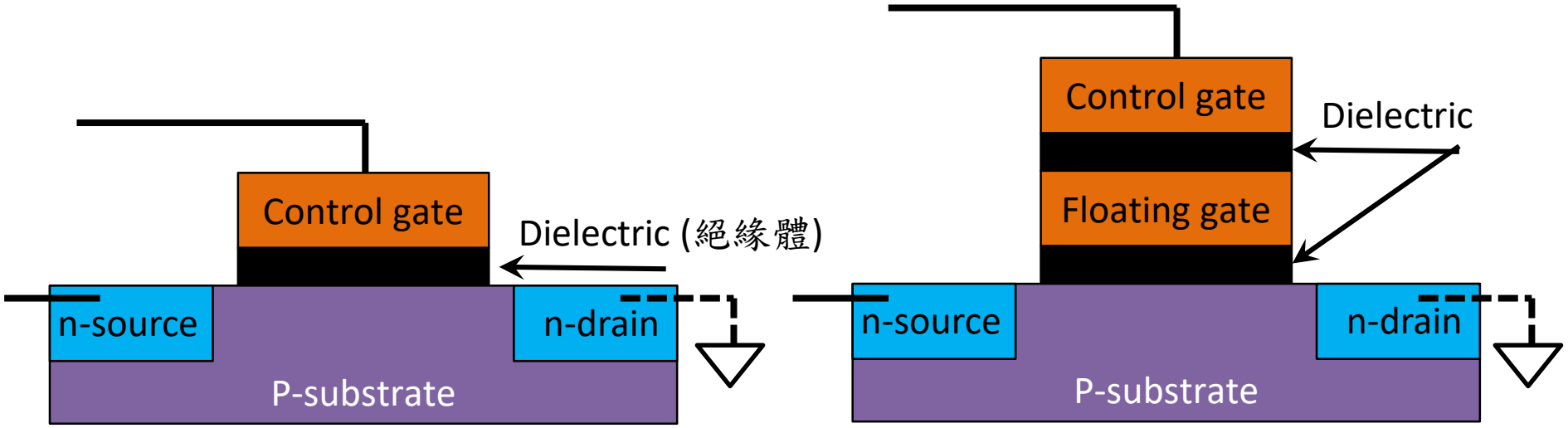


Sources : Global SSD Market to Post CAGR of 41% Until 2022, OriginStorage

The Greatest Invention in 1990s

- Invented by Dr. Fujio Masuoka (舛岡　富士雄) born in1943, while working for Toshiba around1980.
- First presented in IEEE International Electron Devices Meeting (IEDM), San Francisco, 1984.
- First commercialized by Intel as NOR flash in 1998 to replace ROM as BIOS and firmware.
- First NAND flash introduced as SmartMedia storage in 1995. (Our focus)

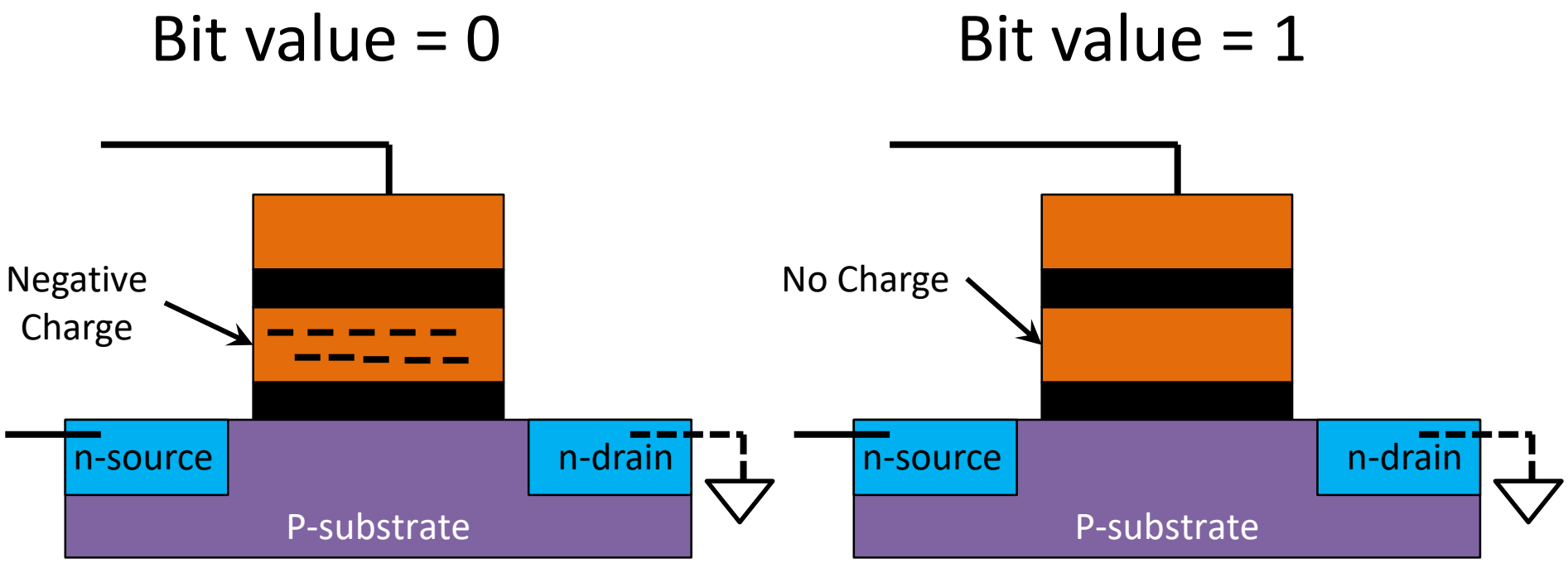
The Key Device: Floating-Gate Transistor



MOSFET **Floating-Gate Transistor**

MOSFET: metal–oxide–semiconductor field-effect transistor

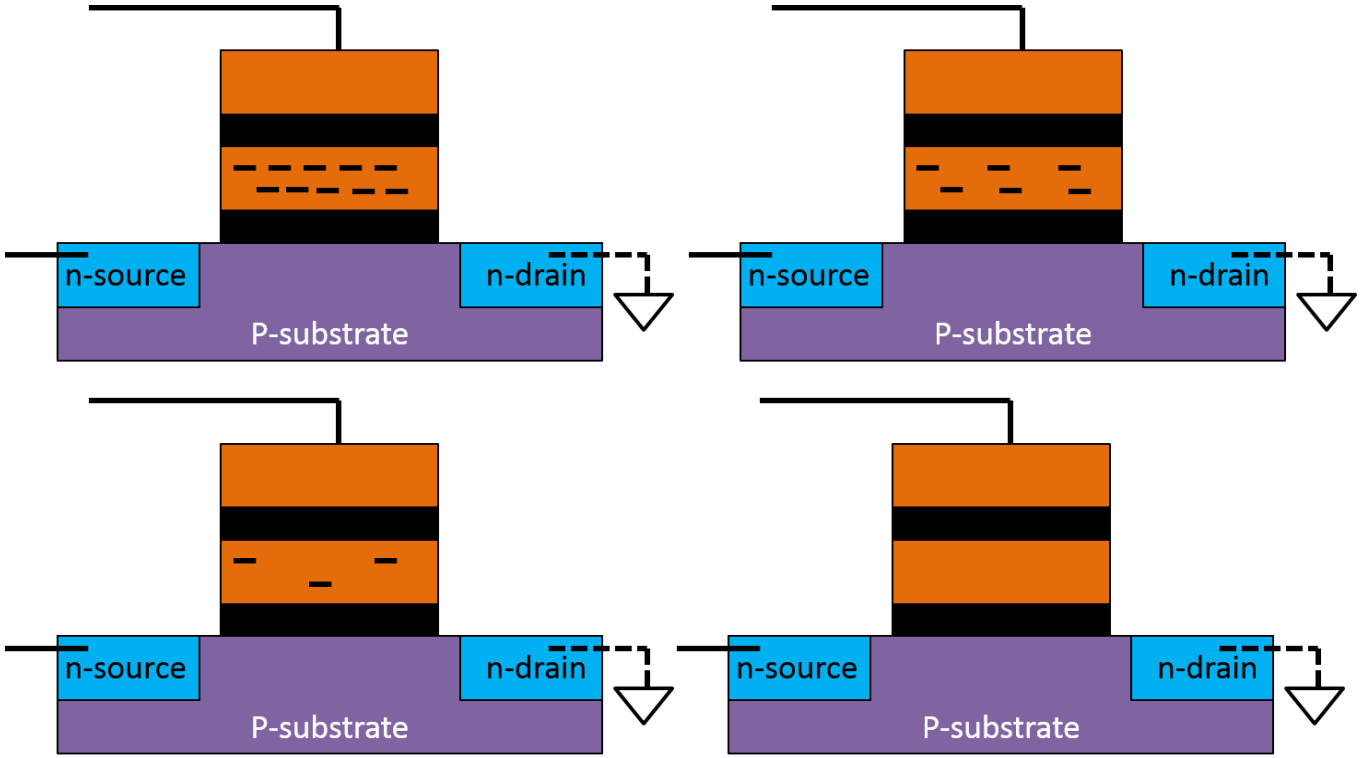
Single Level Cell



SLC (Single Level Cell)

- 2 charge states
- 1 bit per floating gate transistor

Multi Level Cell

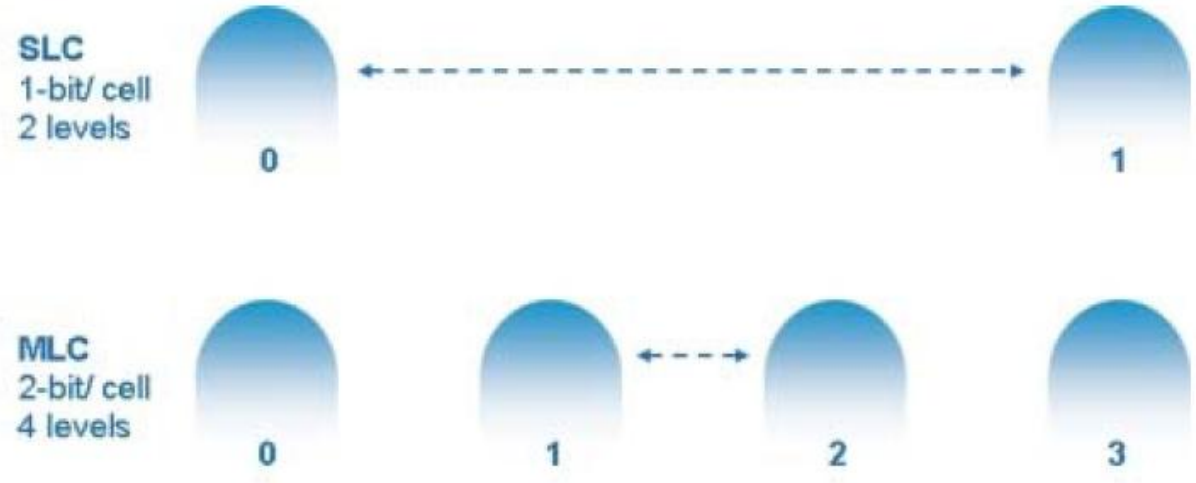


MLC (Multi Level Cell)

- > 2 charge states
- > 1 bit per floating gate transistor

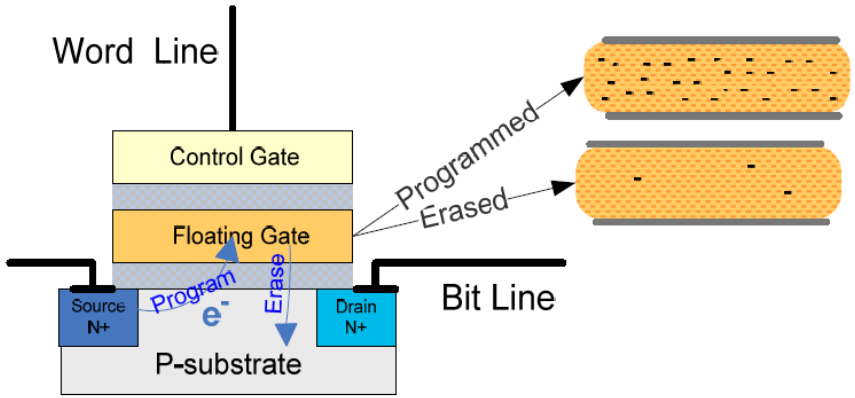
SLC vs. MLC

- Single-Level Cell (SLC) NAND
  - SLC provides faster write speed, lower error rate and longer write endurance
    - Usually used in the applications require high reliability, performance, and viability
- Multi-Level Cell (MLC) NAND
  - MLC allows each memory cell to store multiple bits of information
    - **Cost** and **reliability** are relatively low
    - Usually used in the cell phone, digital cameras, USB drives and memory cards

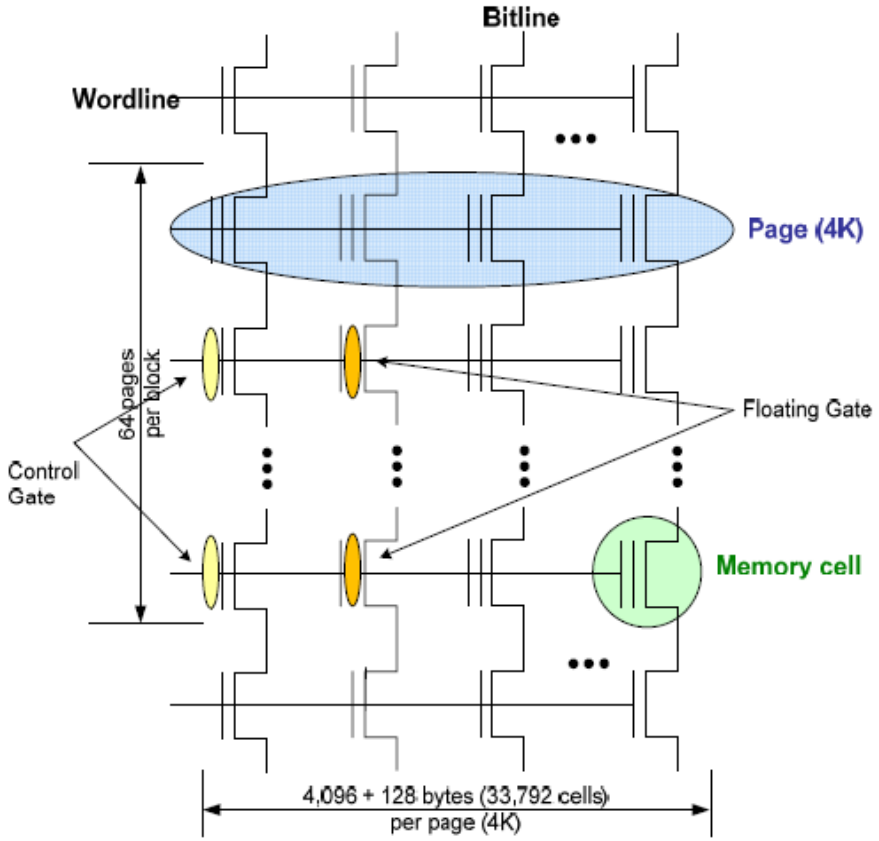


NAND Flash Architecture

- NAND flash cell architecture

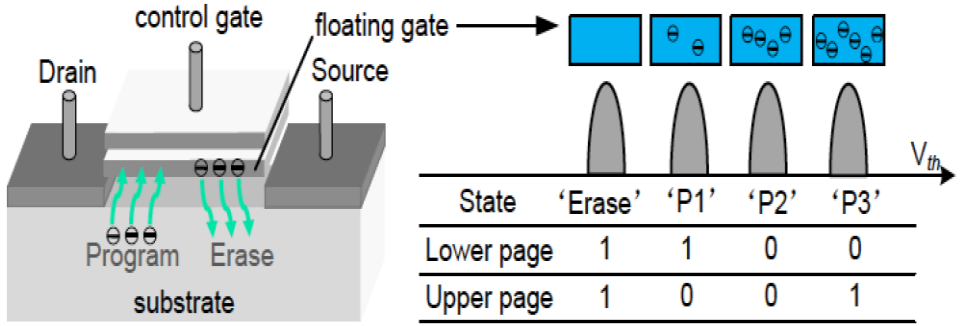


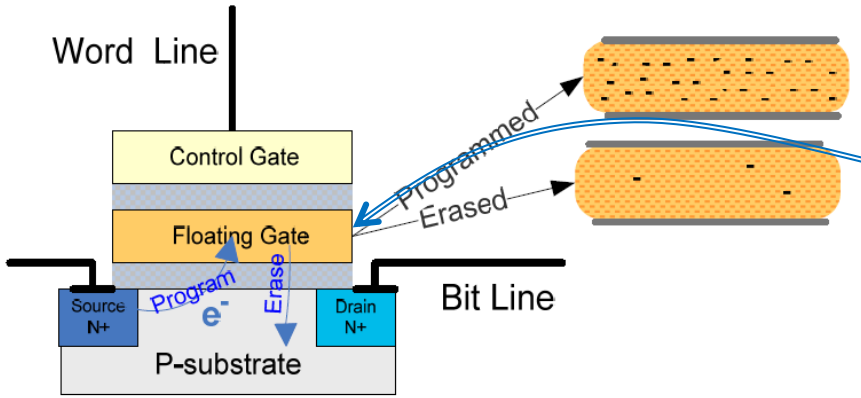
- NAND flash architecture

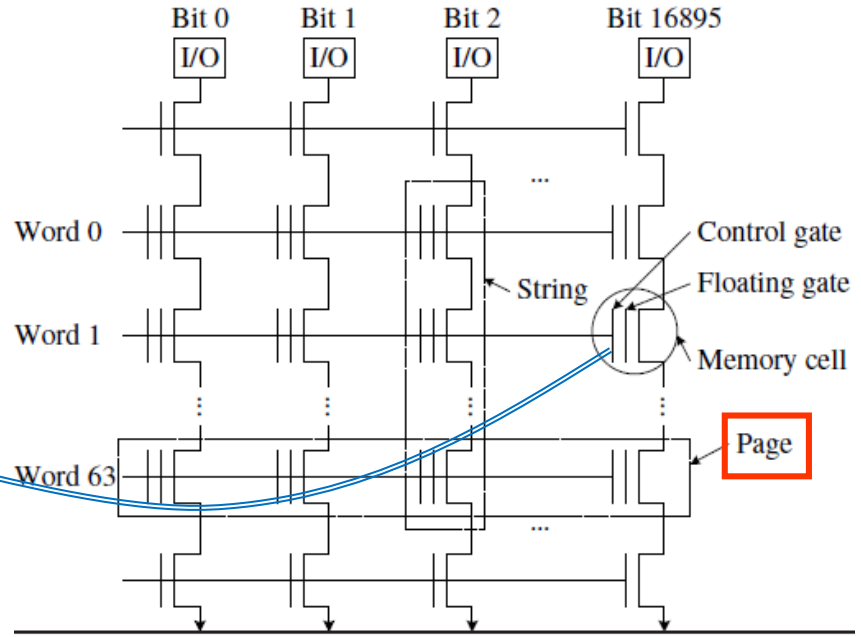


Flash Organization

- Program
  - Electron is moved into the floating gate (FG), and the threshold voltage is raised.
- Erase
  - Electron is removed from the floating gate (FG), and the threshold voltage is thus lowered.

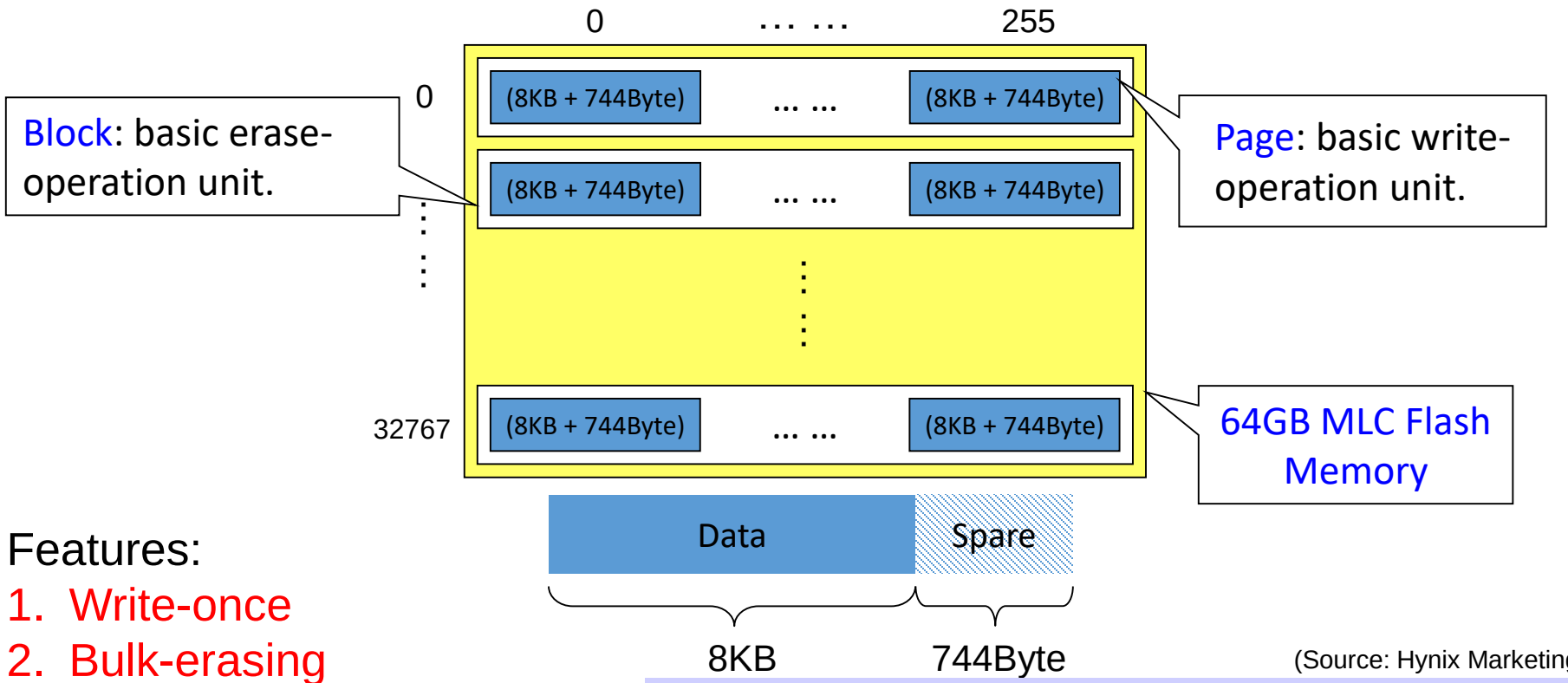






Traditional 2D Flash – A Block Structure

Characteristics of Flash Memory



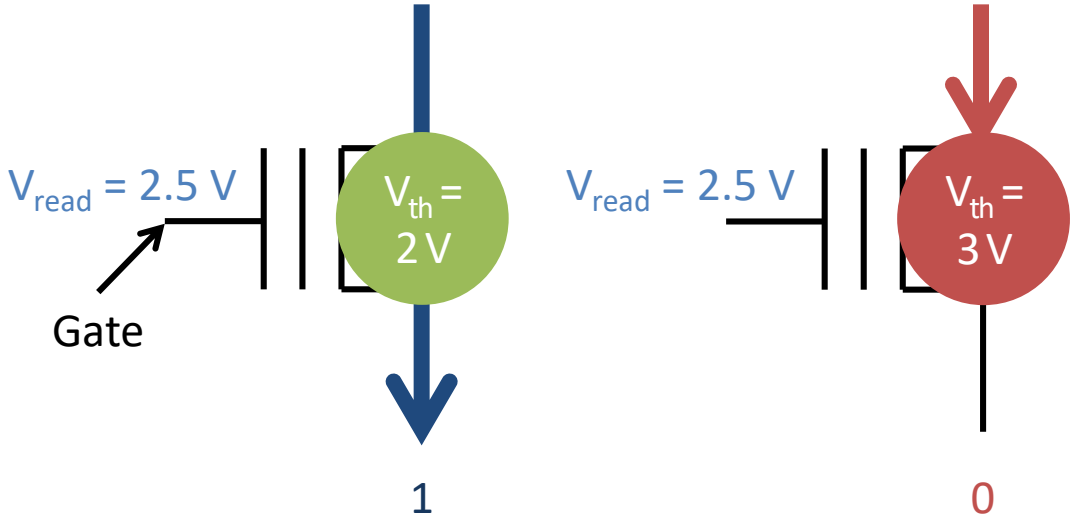
Features:

1. Write-once
2. Bulk-erasing
   → Out-of-place updating
   → Garbage collection
3. Limited erase cycles
   → Wear leveling

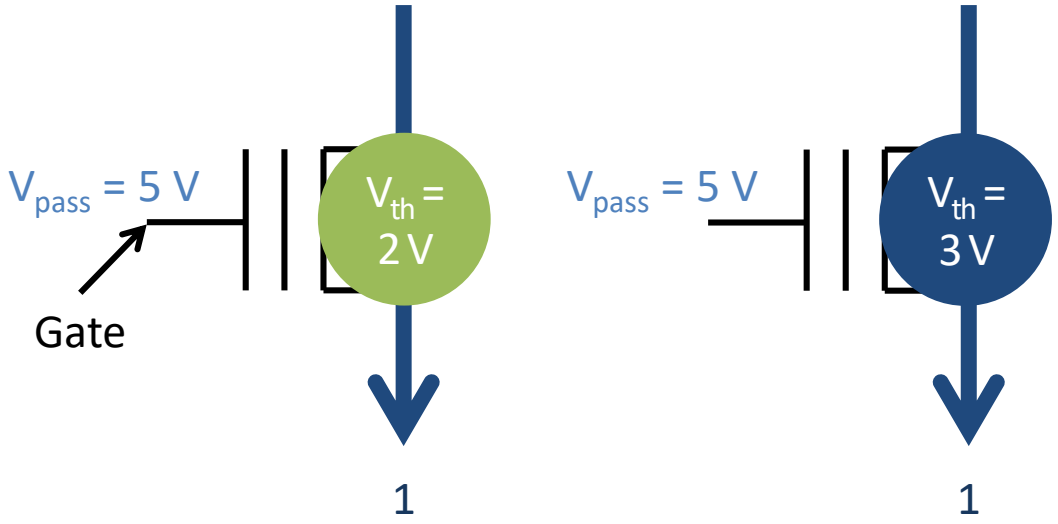
(Source: Hynix Marketing)

-	SLC			MLC		
NAND Process	70/60nm	50nm	40/30nm	70/60nm	50nm	40/30nm
ECC required	1-bit	1-bit	4-bit	4-bit	4~8 bit	12~24 bit or more
Erase Cycle	100K	100K	TBD	10K	5K~10K	3K~5K
Data Retention	10 yrs	10 yrs	10 yrs	10 yrs	10 yrs	5 yrs

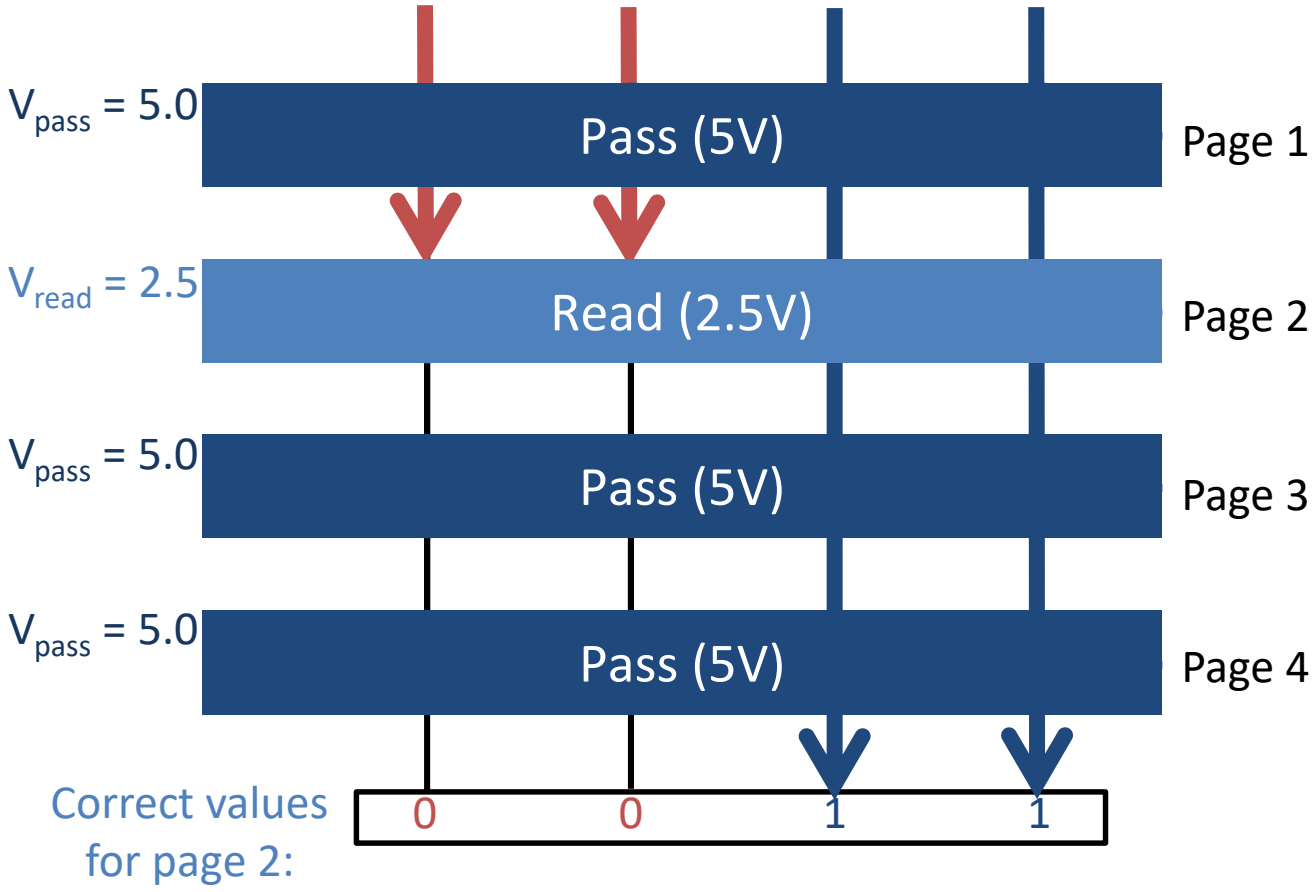
Flash Read



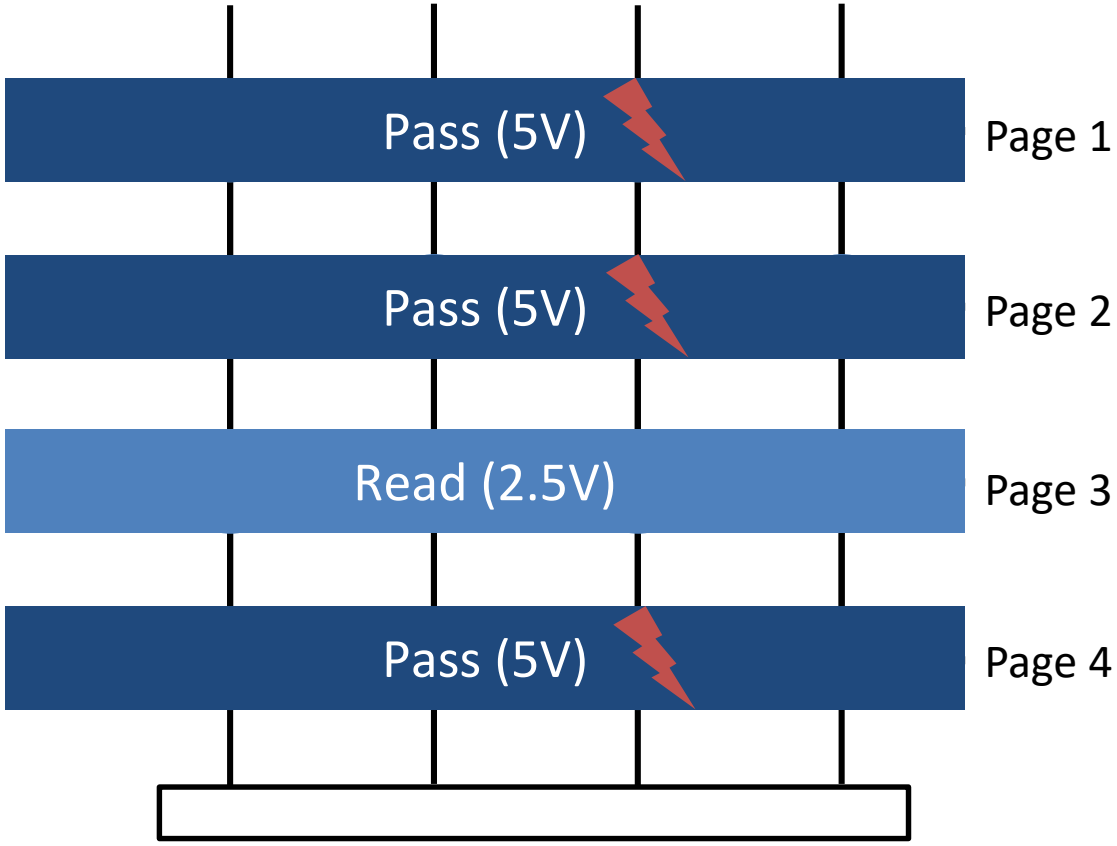
Flash Pass-Through



Read from Flash Cell Array

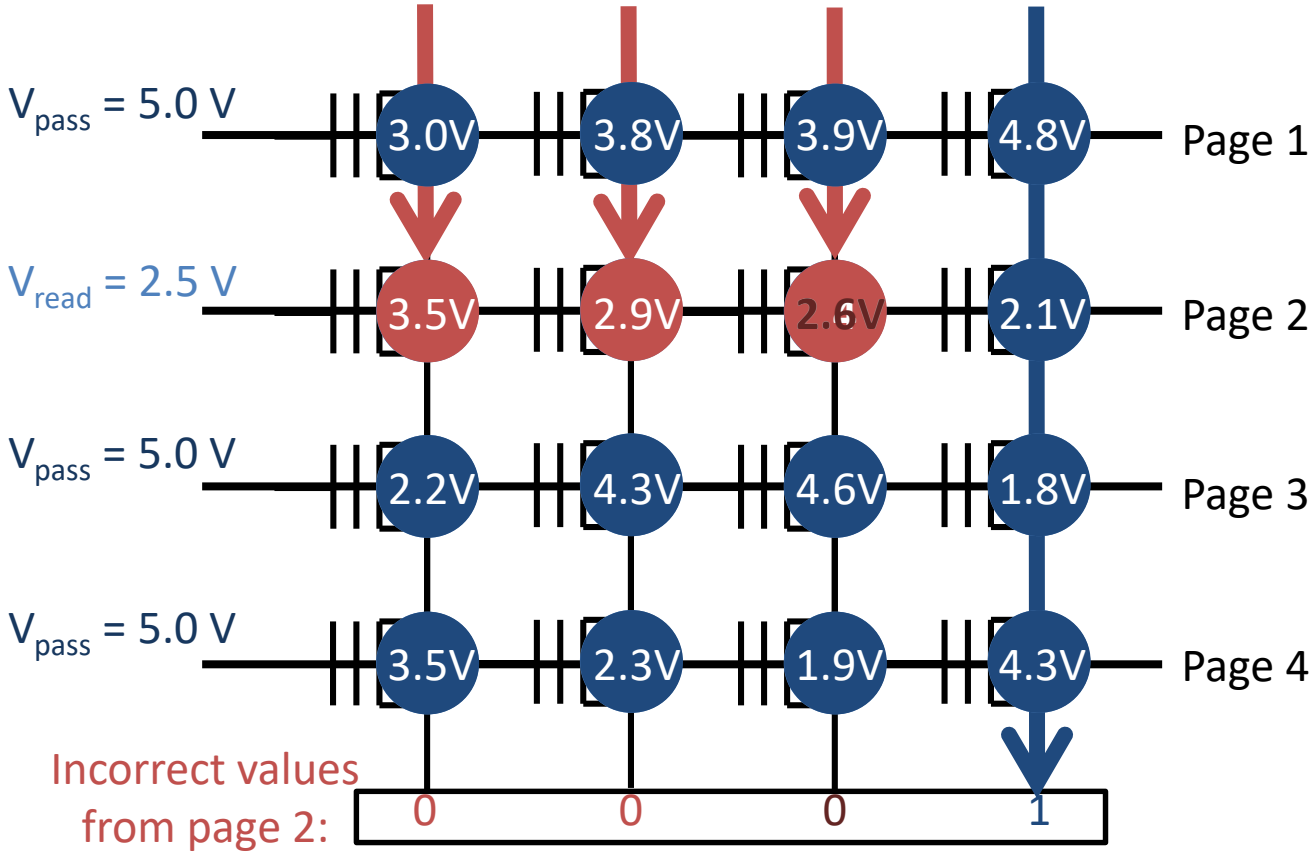


Read Disturb Problem : “Weak Programming” Effect



Repeatedly read page 3 (or any page other than page 2)

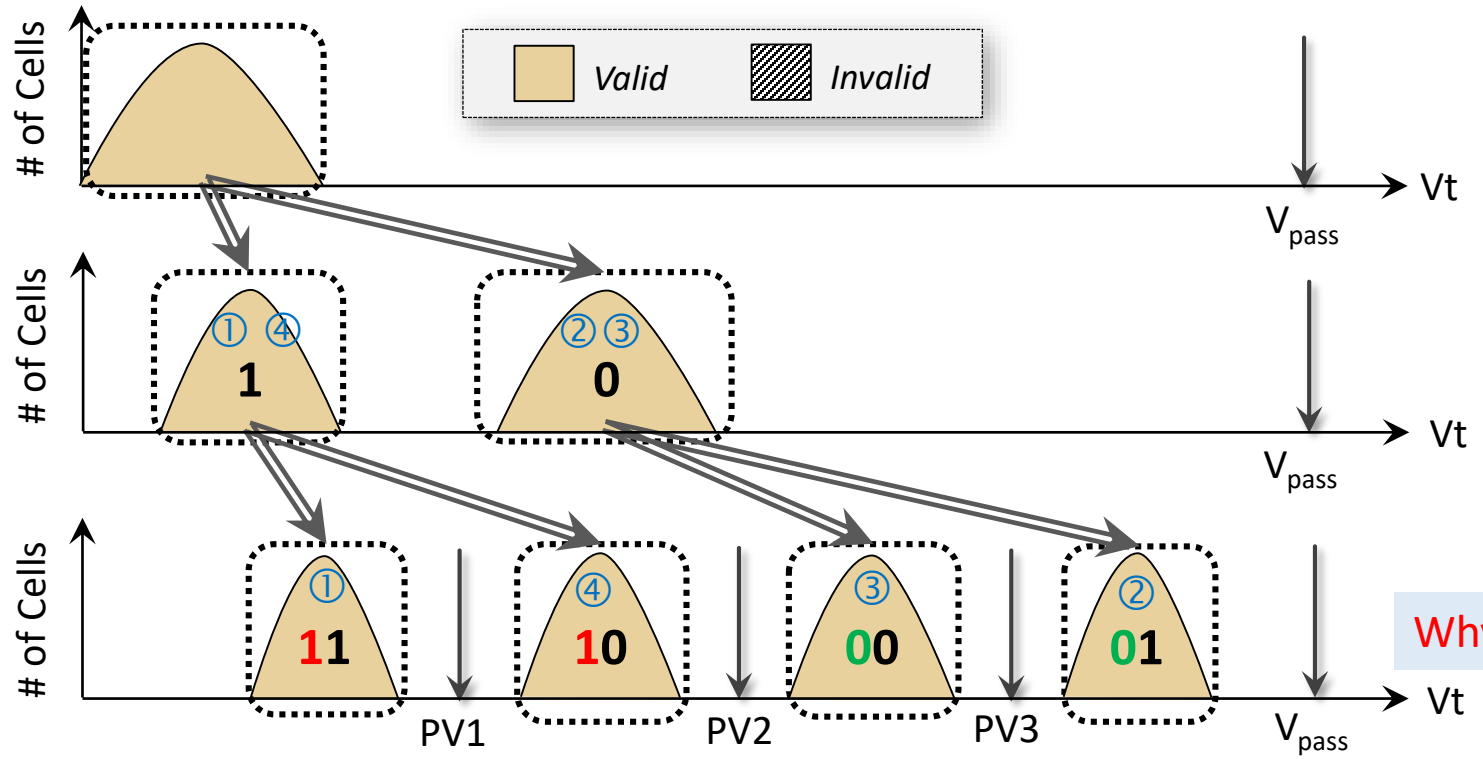
Read Disturb Problem: "Weak Programming" Effect



High pass-through voltage induces "weak-programming" effect

Program – MLC Page

1 0 0 1
1 1 0 0
① ② ③ ④

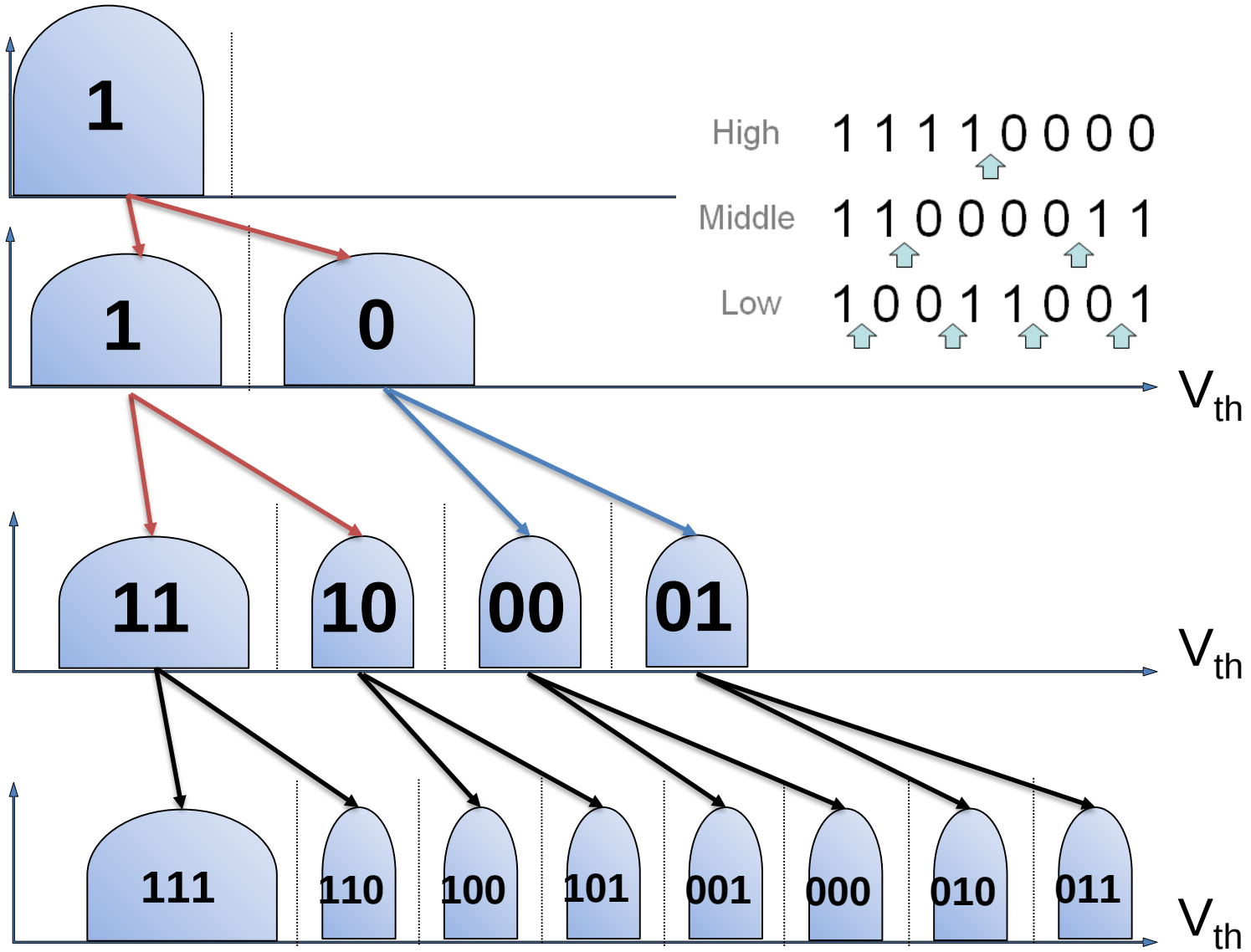


Why gray code?

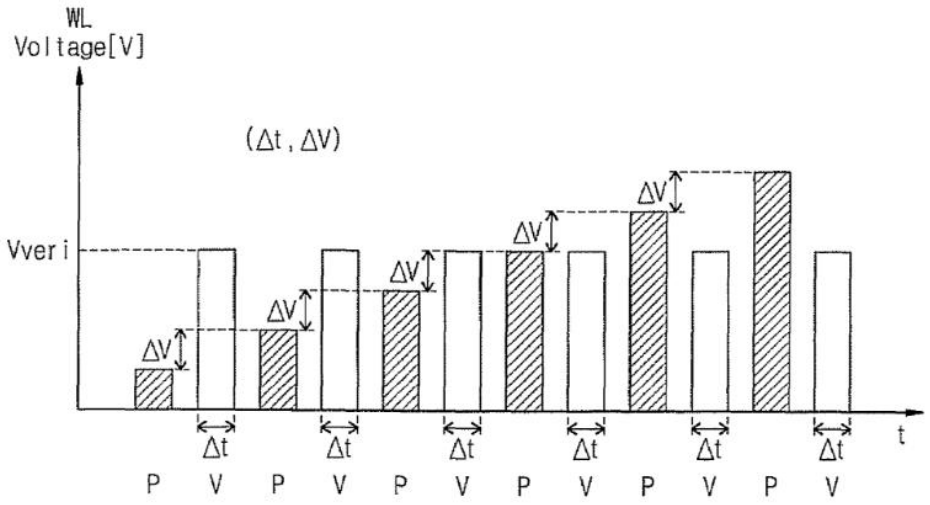
PV: Program Verify Voltage
V_{pass}: A Pass-gate Voltage
Vt: Threshold Voltage

1. ***Low efficiency in programming low pages***
2. ***High bit error rate in low pages***

Program – TLC Page (1-2-4 Sensing)



Program:
ISPP - Incremental Step Pulse Programming

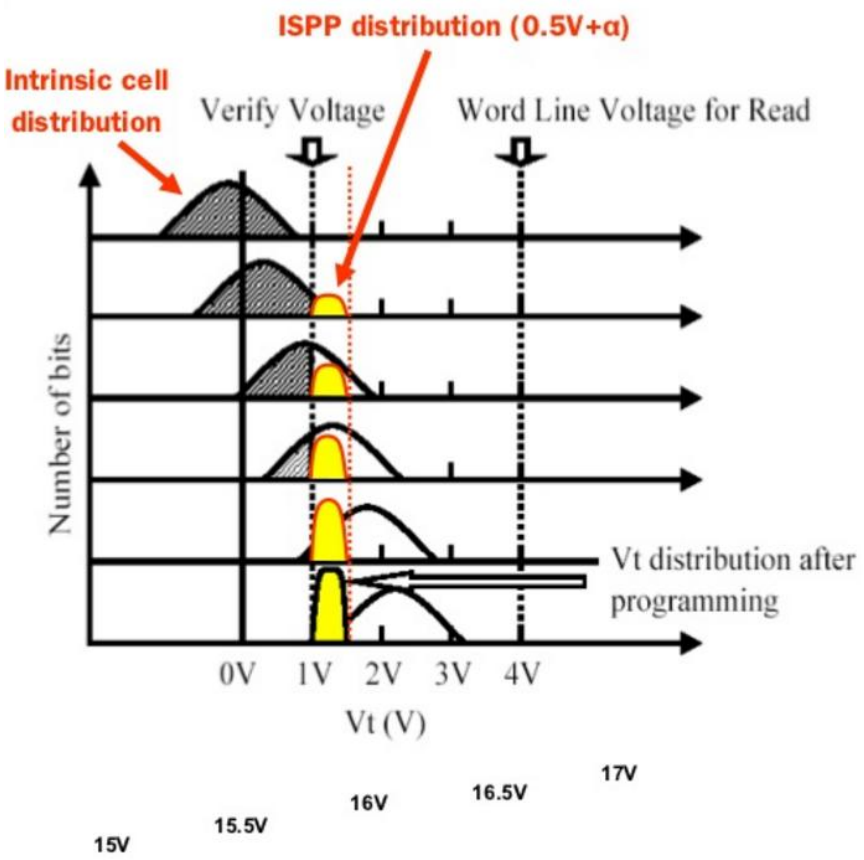


Slow cell
→ continuing program at higher bias

Fast cell
→ stop programming

B/L
0V→VCC

B/L
0V

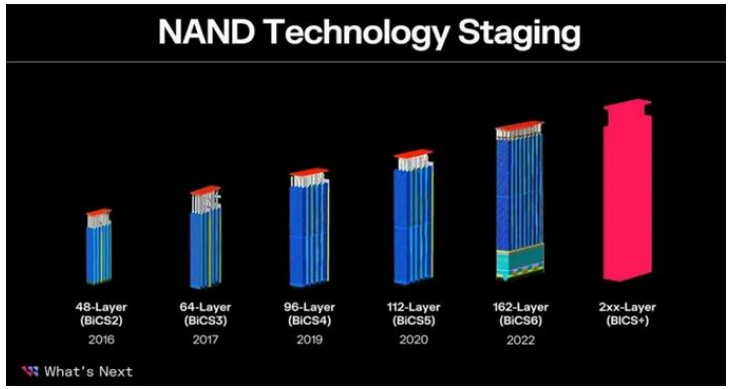


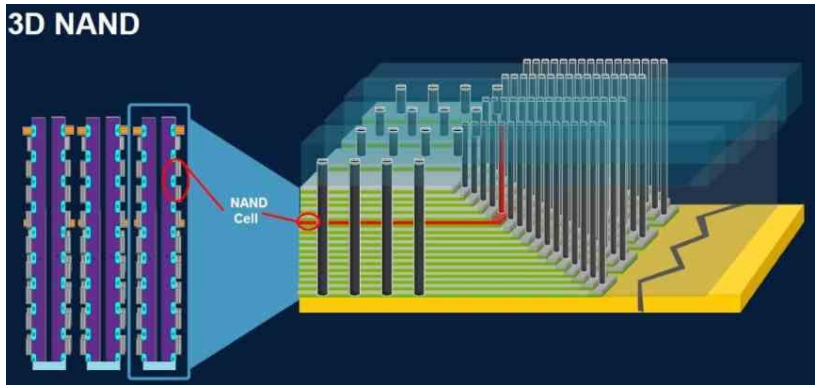
Flash Memory Technology

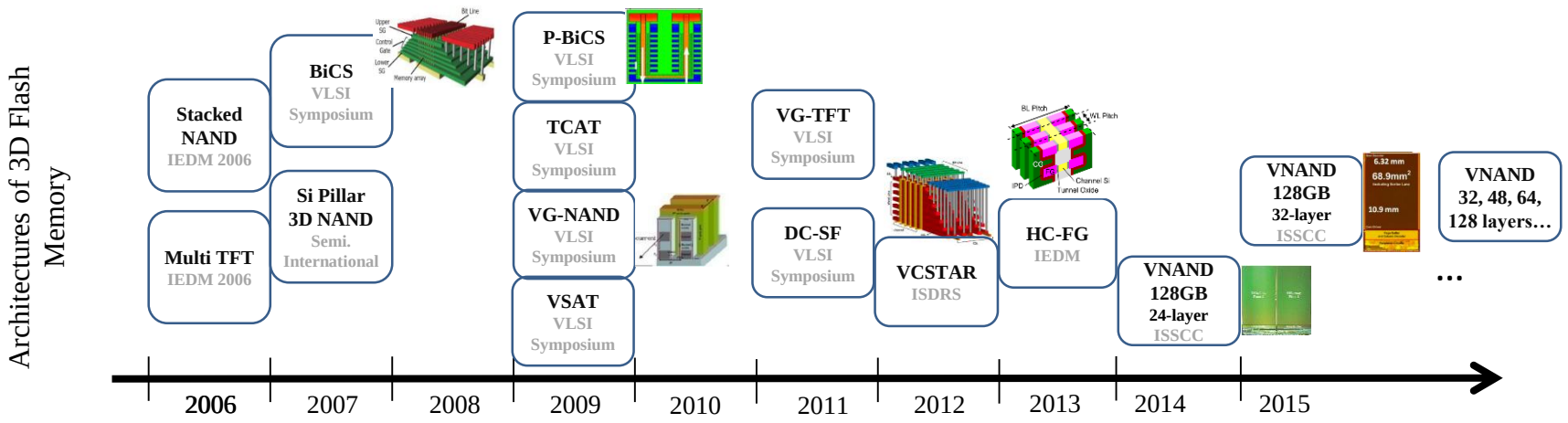
	SLC NAND Flash	MLC NAND Flash	MLC NOR Flash
Density	512Mbits – 4Gbits	1Gbit to 16Gbit	16Mbit to 1Gbit
Read Speed	24 MB/s	18.6 MB/s	103MB/s
Write Speed	8.0 MB/s	2.4 MB/s	0.47 MB/s
Erase Time	2.0 mSec	2.0mSec	900mSec
Interface	I/O – indirect access	I/O – indirect access	Random access
Application	Program/Data mass storage	Program/Data mass storage	Execute In Place (XIP)

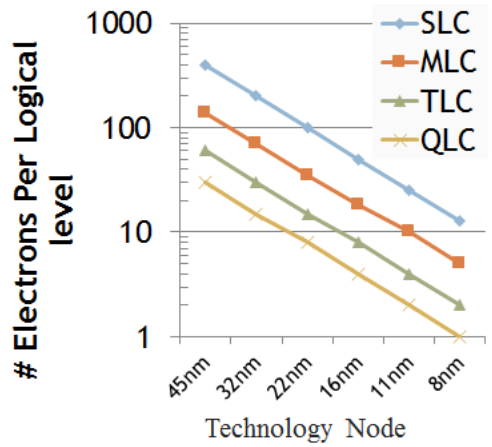
3D Flash Architecture Evolution

- 3D flash memory provides a good chance to further scale down the feature size and to reduce the bit cost.
  - ***Deliver very large storage space***
  - ***Worsen program disturbance***

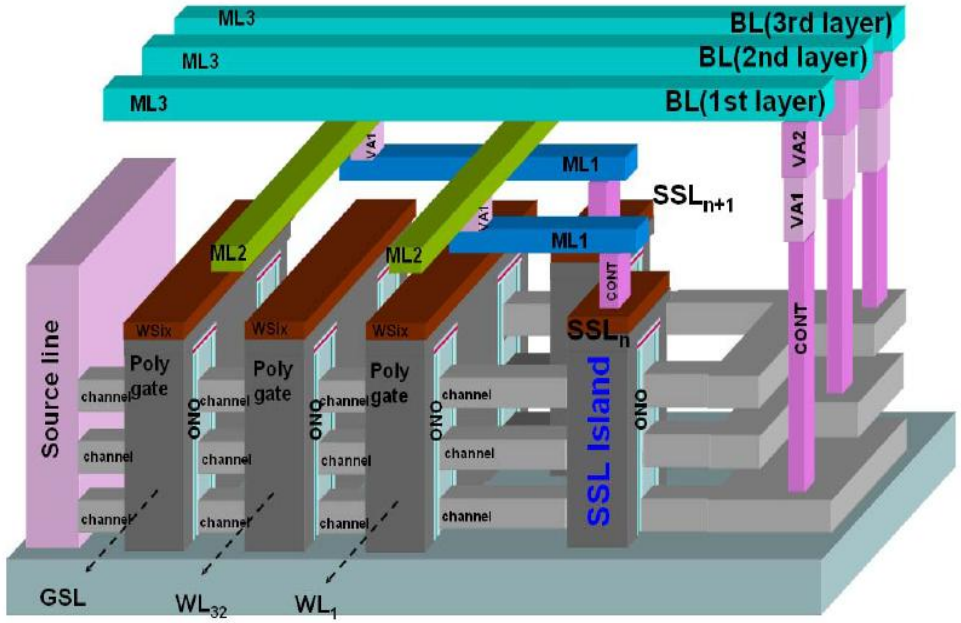


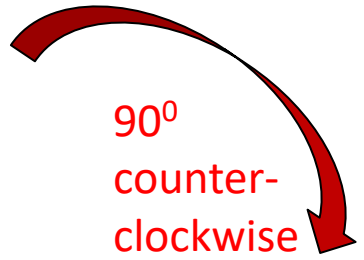


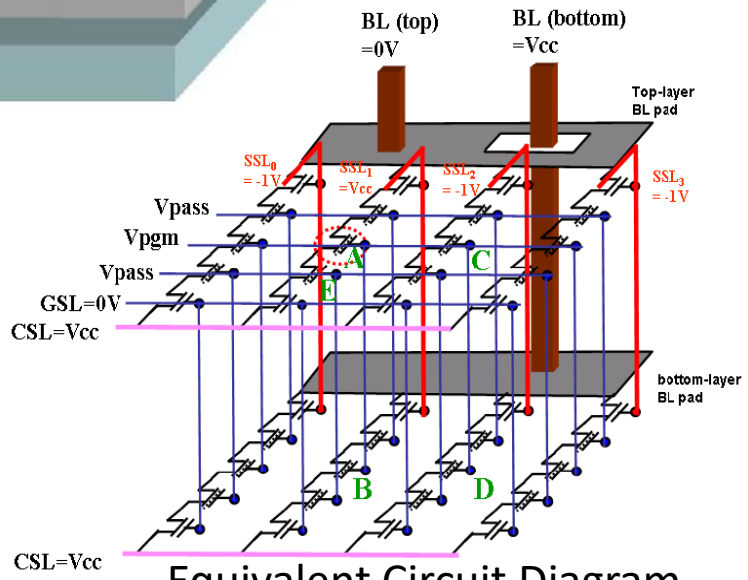




3D Flash Memory Architecture







Equivalent Circuit Diagram

Abbreviation	Full Name	Function
WL	Word Line	Controls the gate of each memory cell, used to select which cell to operate.
BL	Bit Line	Responsible for data read and write, connected to the drain of the NAND string.
CSL	Common Source Line	A shared line connecting the source of all cells in a string, provides a fixed potential (usually Vcc or GND).
SSL	Source Select Line	A select gate at the top of the NAND string, controls whether the bit line (BL) is connected to the string. (When SSL=Vcc, it is turned on)
GSL	Ground Select Line	A select gate at the bottom of the NAND string, controls whether the string's source is connected to the CSL. (In this figure, the string is on when GSL=0)
Vpgm	Program Voltage	A high voltage applied during programming (write) to change the stored charge.
Vpass	Pass Voltage	A voltage applied to non-selected cells in a string to keep them conducting, allowing data to pass through.

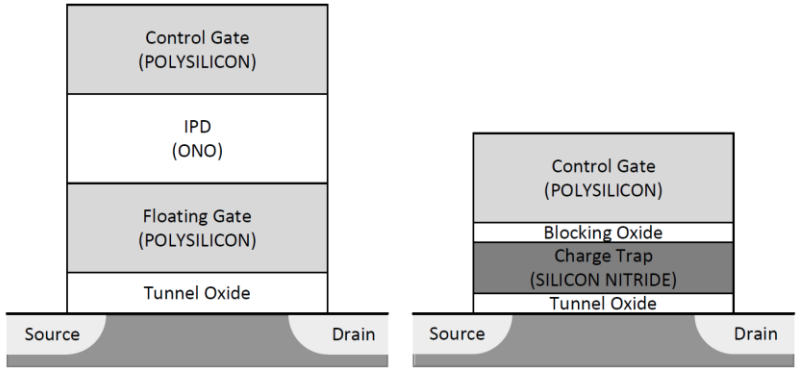
Vertical Gate 3D Flash Memory

- WL= word line
- Channel = bit line
- SSL = string select line
- CSL (Common Source Line)
- GSL (Ground Select Line)

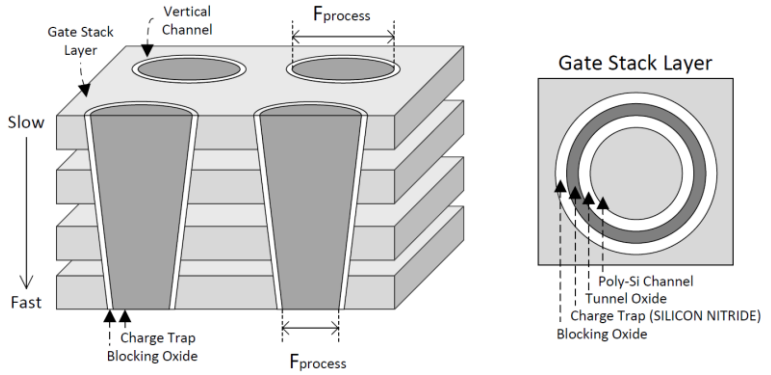
K.-P. Chang, H.-T. Lue, C.-P. Chen, C.-F. Chen, Y.-R. Chen, Y.-H. Hsiao, C.-C. Hsieh, Y.-H. Shih, T. Yang, K.-C. Chen, C.-H. Hung, and C.-Y. Lu. Memory architecture of 3d vertical gate (3dvg) nand flash using plural island-gate ssl decoding method and study of it's program inhibit characteristics. IMW '12.

3D Charge-Trap Flash Memory

- Tunnel oxide thickness of floating-gate cell must be more than 6-nm to prevent charge leakage and assure enough retention time.
  - Scaling down traditional floating-gate cell becomes much more challenging.
- Charge trap memories, such as SONOS and TANOS becomes popular in 3D flash memory



(a) Floating gate (b) Charge Trap



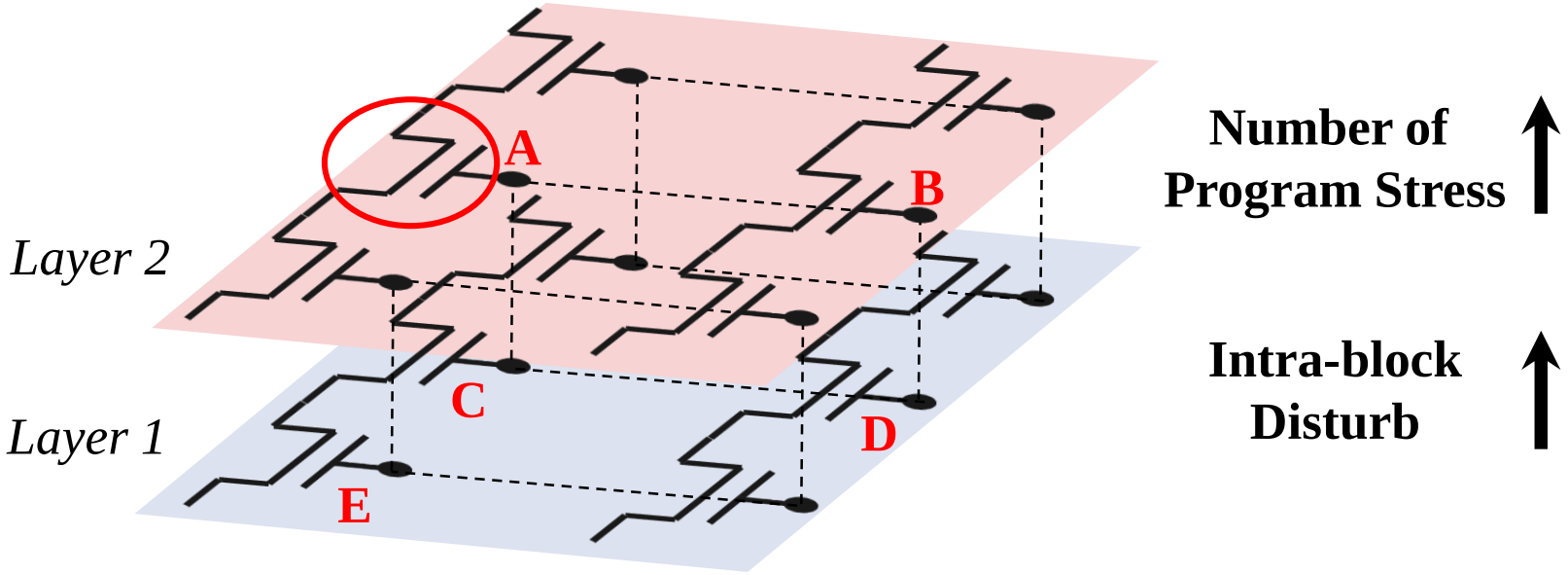
(a) 3D NAND vertical channel (b) Top-down view of vertical channel

Feature	Floating Gate (FG)	Charge Trap (CT)
Material	Poly-Si floating gate	Si_3N_4 charge-trap layer - Oxide–Nitride–Oxide (ONO) stack
Principle	Electrons are injected and stored in the floating gate (電子注入並儲存在浮置閘極中)	Electrons/holes are trapped in localized states within the nitride layer (電子/電洞被困在氮化矽的局部陷阱中)
Charge storage	Continuous conductive gate	Localized trap sites in nitride
Scalability	Hard to scale beyond 2D NAND	Well-suited for 3D NAND
Leakage risk	Higher (thin oxide leakage)	Lower
Cell interference	More (shared floating gate conductor)	Less (localized traps)
Industry usage	Older NOR / early NAND	Dominant in modern 3D NAND

Deteriorated Disturb on 3D Flash

- 3D flash memory introduces a new disturb, ***Z-direction Disturb***

  → Programming to Cell 'A' not only disturb Cell 'B', but also Cell 'C', 'D' and 'E'



Inherent Nature

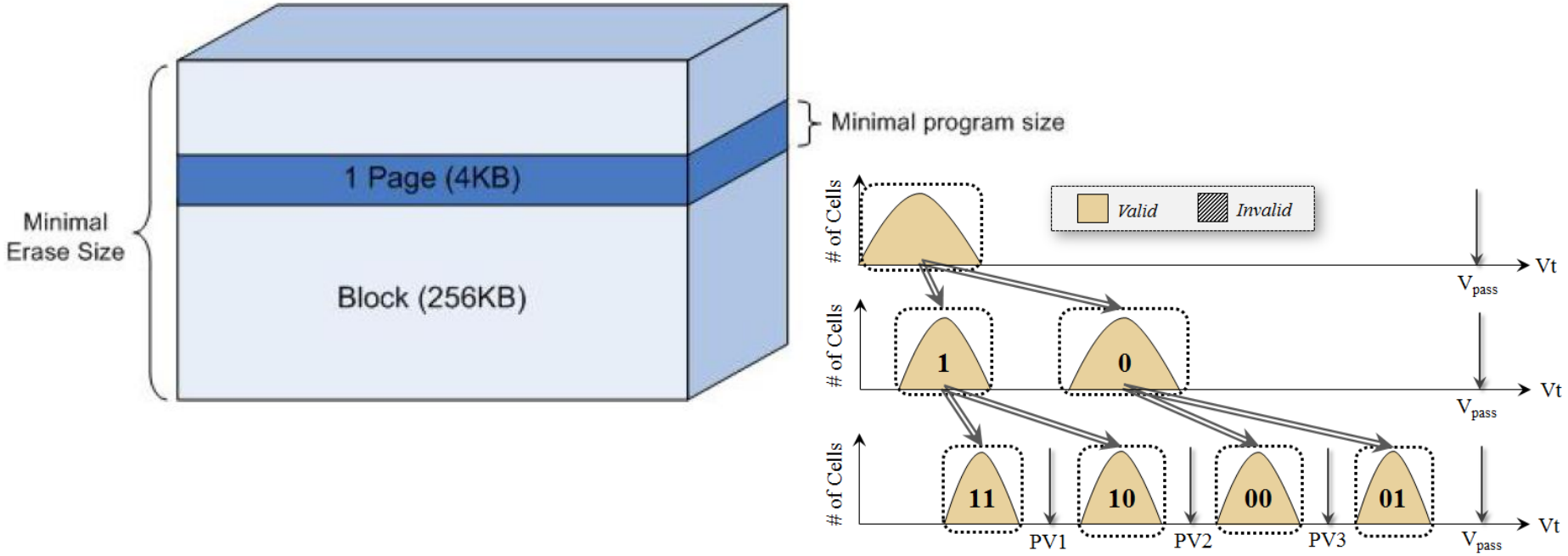
- Challenges of using NAND flash include
  - Need to erase before writing (write-once property)
  - Wear out mechanism that limits service life (Wear leveling)
  - Data errors caused by write and read disturb
  - Data retention errors
  - Management of initial and runtime bad blocks
    - If a flash chip has some bad blocks, it is considered a good chip or not?
- We need sophisticated flash management techniques to make flash become powerful storage

Research Issues

- Proper flash management makes flash memory a highly reliable data storage device
- Five significant factors affect reliability, performance and write endurance of SSD
  - SLC vs. MLC vs. TLC vs. 3D flash
  - Wear-leveling algorithms
  - Bad Block management techniques
  - Error detection and correction techniques
  - Write amplification

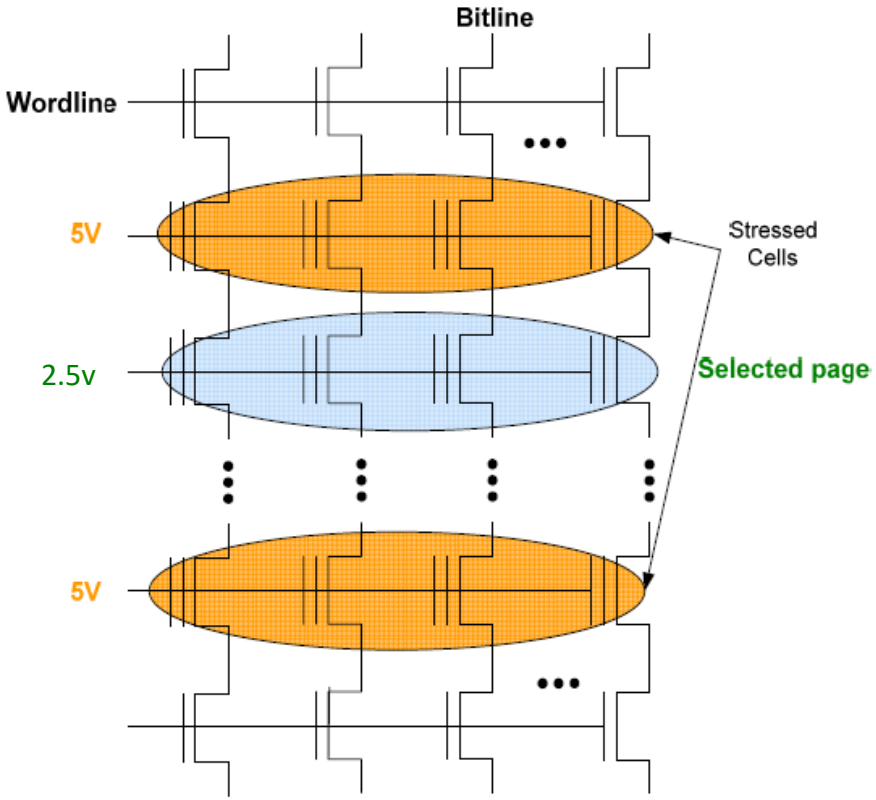
Erase Before Write

- The Erase operation sets all the bits in the block to a “1”.
- Once written to a “0”, the only way to reset a bit to a “1” is by erasing the entire block.

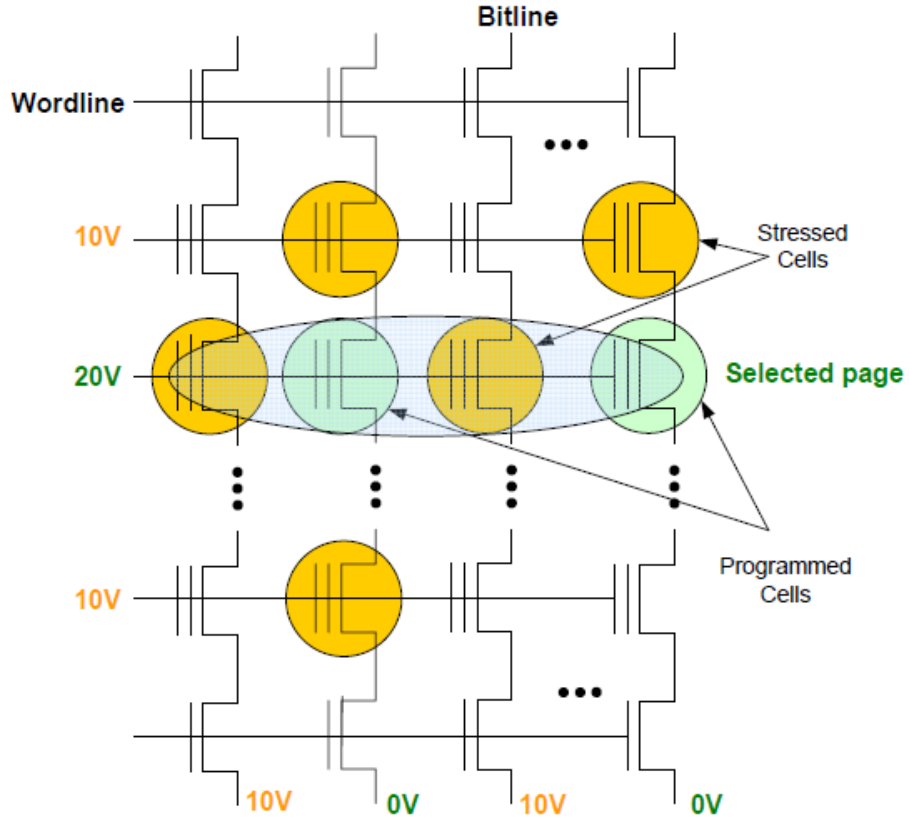


Read/Write Disturb

- Read disturb

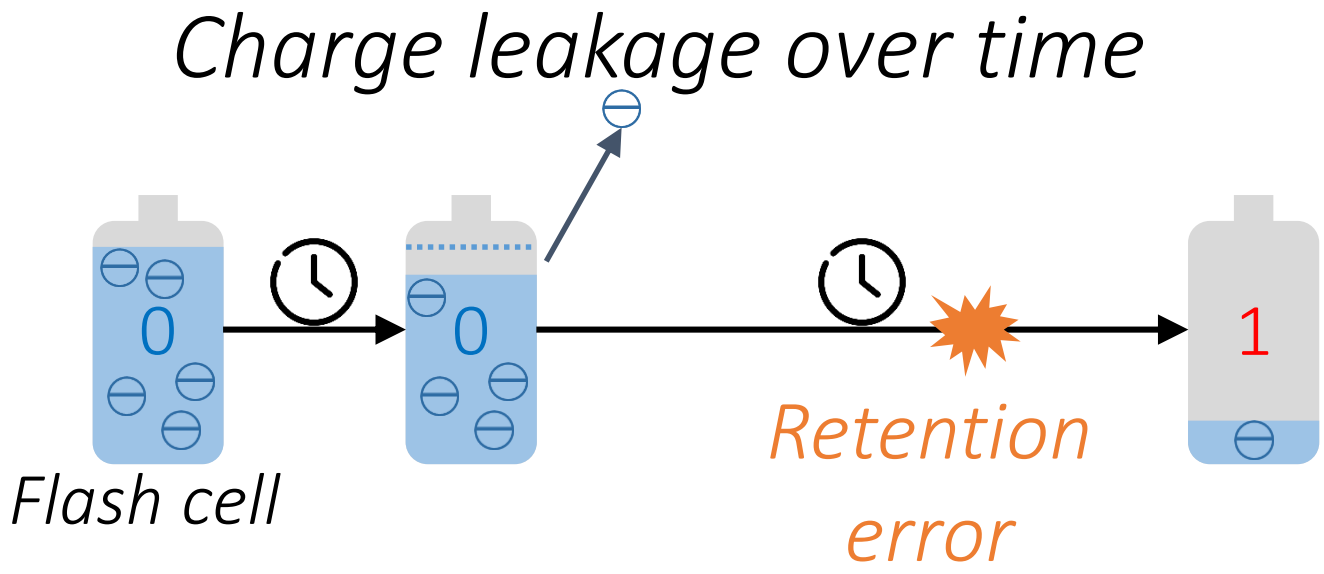


- Write disturb

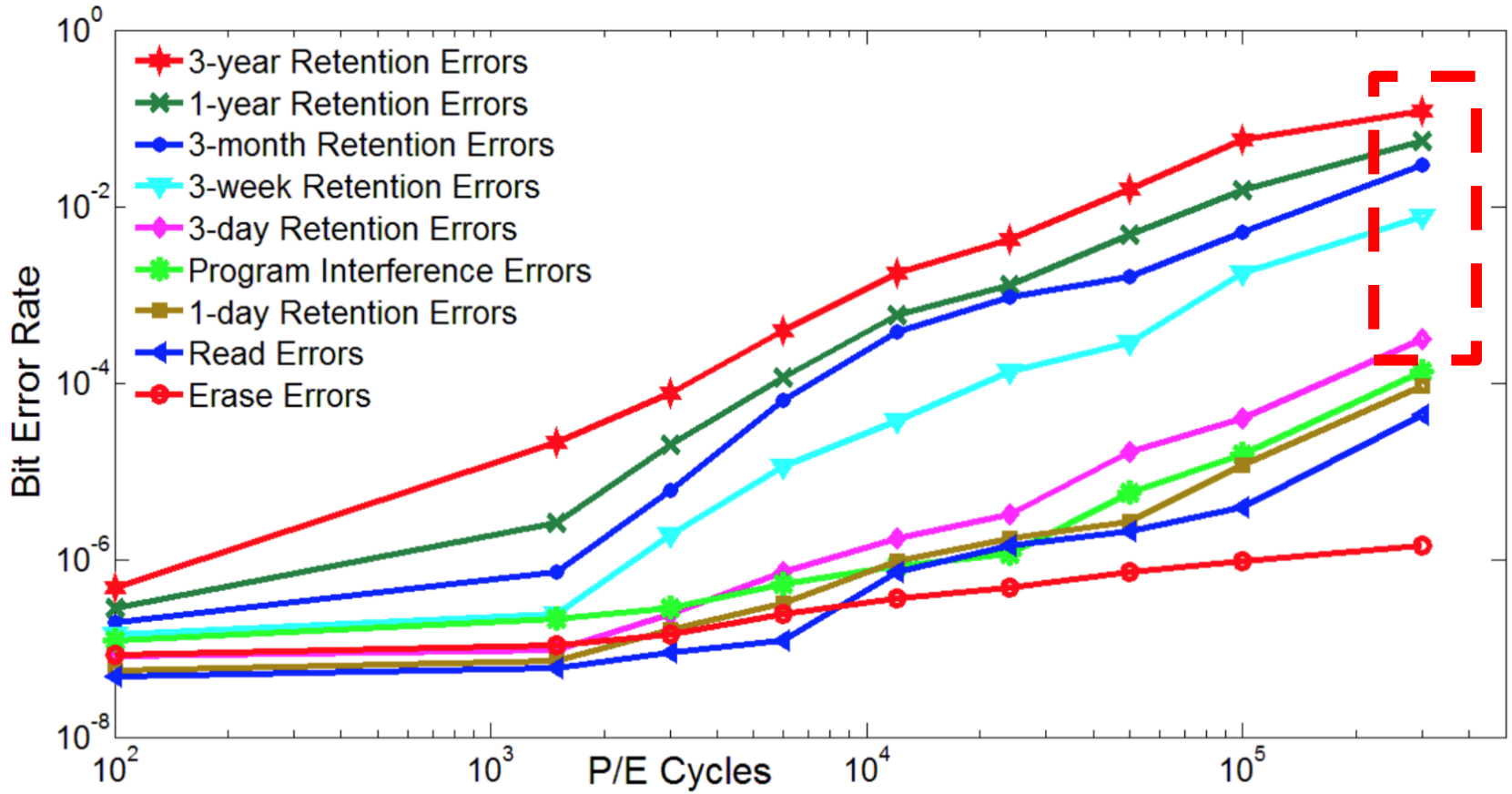


Data Retention Errors

- Data defines how long the written data remains valid within a memory device
- The data retention time is inversely related to the number of Program/Erase cycles (P/E cycles)



P/E Cycles vs. Bit Error Rates



BER of various type of errors as P/E cycles increase

[1] Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis.

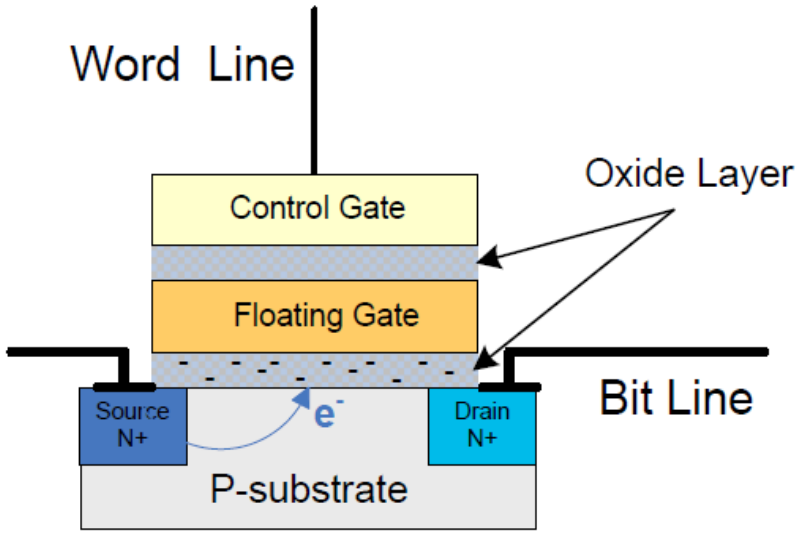
978-3-9810801-8-6/DATE12/©2012 EDAA

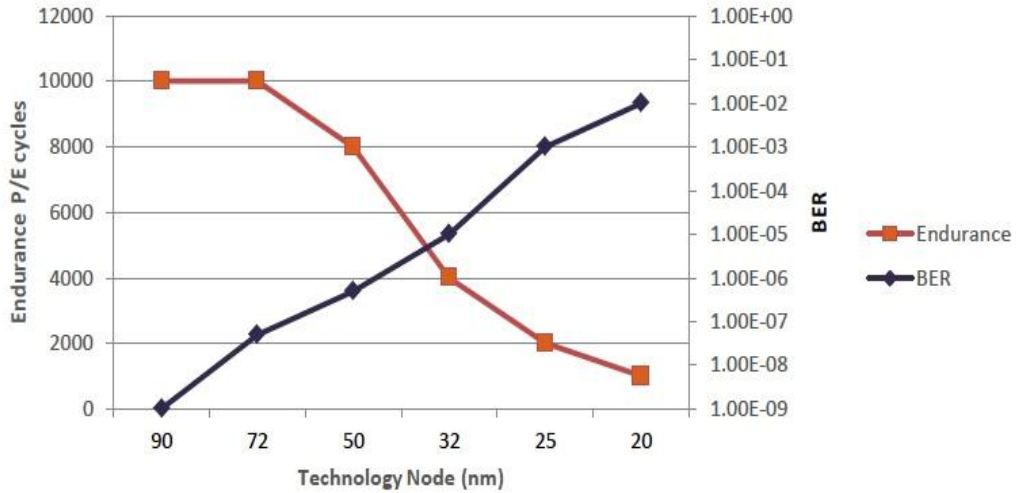
Bad Blocks

- Initial bad blocks
  - Due to the production yield constraints and the pressure to keep costs low
  - Up to 2% of the SLC flash can contain bad blocks; the number for MLC flash is about 5%
- Accumulated bad blocks
  - Due to multiple write/erase cycles, trapped electrons in the dielectric cause a permanent shift in the voltage levels of the cell

Limited Number of Writes

- Writing damages the oxide layer
- NAND flash has finite number of Write/Erase cycles
  - SLC: ~100K P/E cycles
  - MLC: ~1K ~ 10K P/E cycles
- Wear leveling is necessary





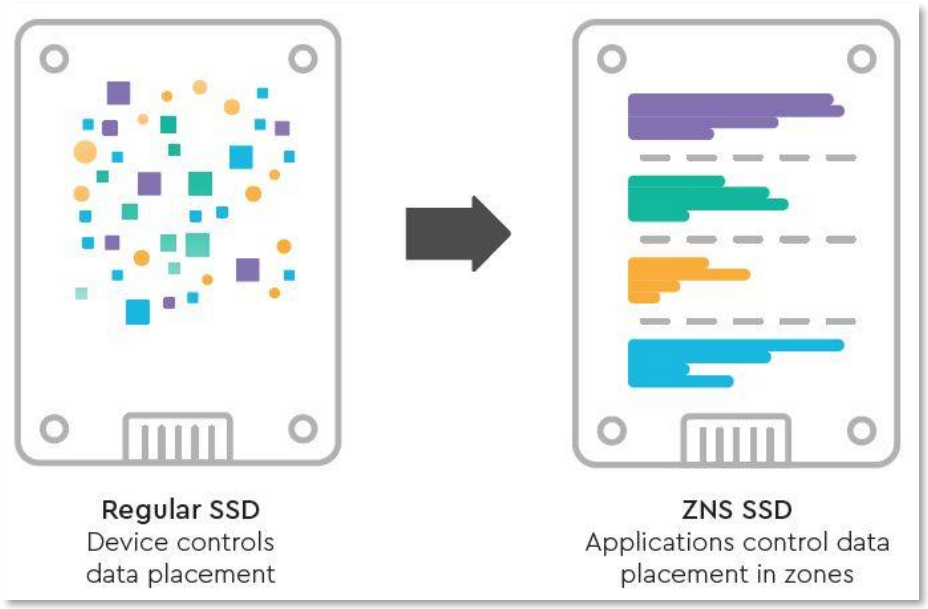
Source: EDN Network

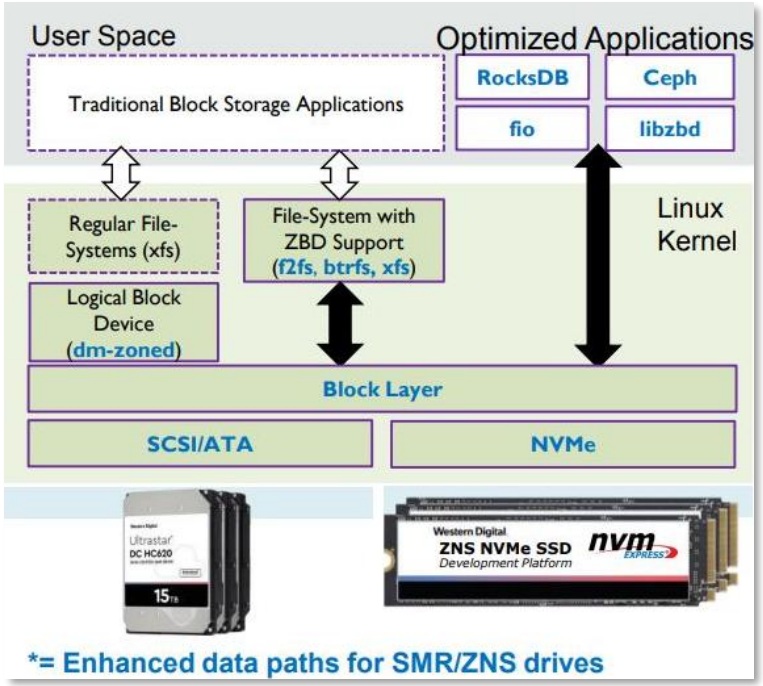
Data Storage in Daily Life
– NAND Flash Memory: Stop, Look, and Listen (停看聽)

- Flash memory has characteristics that are very different from traditional storage media such as mechanical hard drives!
  - Write operations are slower than read operations.
  - Limited write endurance.
  - Limited data retention time.
- To fully leverage the advantages of flash memory, supporting software, hardware, and firmware are required, including:
  - Flash Translation Layer (FTL)
  - Device drivers
  - File systems

Data Storage in Daily Life – OCSSD and ZNS SSD

- **Open-Channel SSD (OCSSD):** Allows flash memory management to be handled by the host CPU, overcoming the limitations of traditional SSDs where management is constrained by the storage device's limited computing resources.
- **Zoned Namespace SSD (ZNS SSD):** By further dividing the drive space into multiple zones and restricting data in each zone to sequential access (left figure), flash memory management can be significantly simplified and device lifespan extended. At the same time, related software and hardware must also be adapted to the new access interface (right figure).





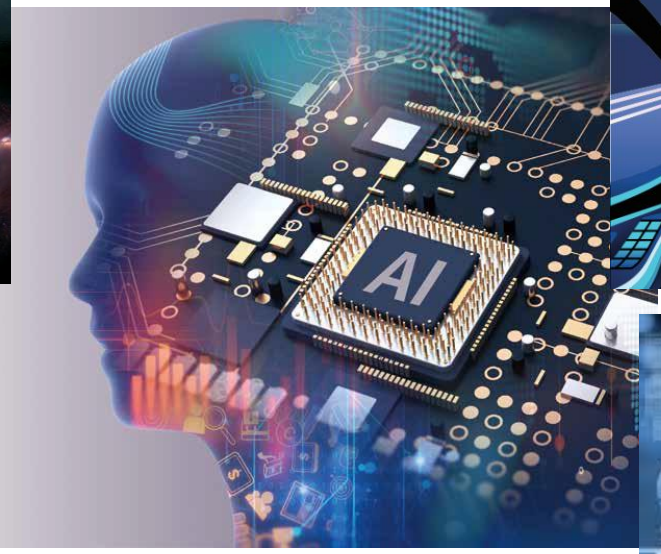
https://www.ithome.com.tw/review/141162

Outline

- Early Data Storage
- Modern Data Storage
- Data Storage in Daily Life
- **Next-Generation Data Storage**
- Future Data Storage

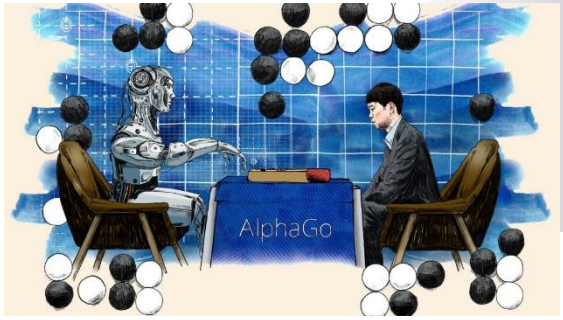
What's Next?

AI Render



Auto Driving

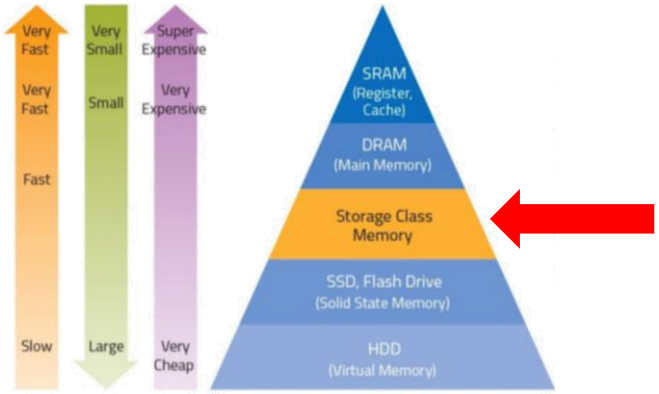
AI GO AlphaGo

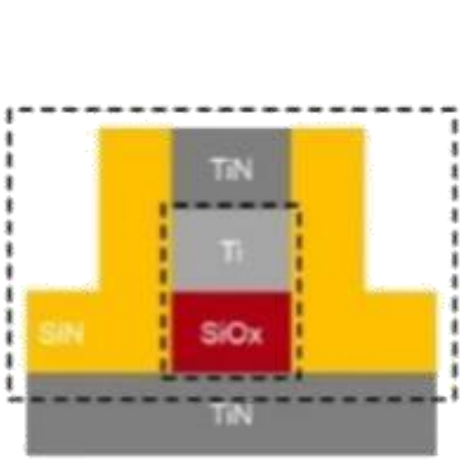


AI Manufcturing

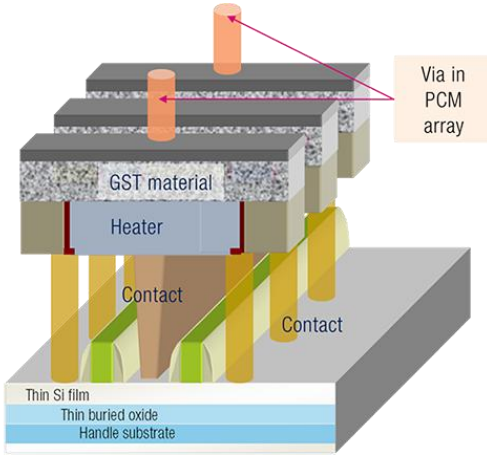
Next-Generation Storage

- New Non-Volatile Memory Technologies
  - **Non-volatility**: Data can still be retained in memory even after power is turned off.
  - **Higher density than DRAM**.
  - Faster speed and longer lifespan than flash memory.
  - Accelerates neural network computation.
  - Both memory and storage features.
  - **Processing-in-Memory (PIM)**.

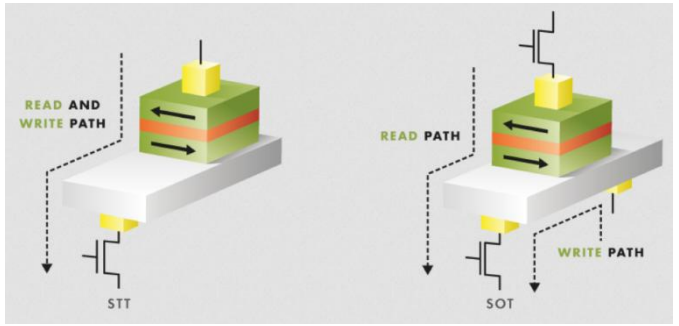




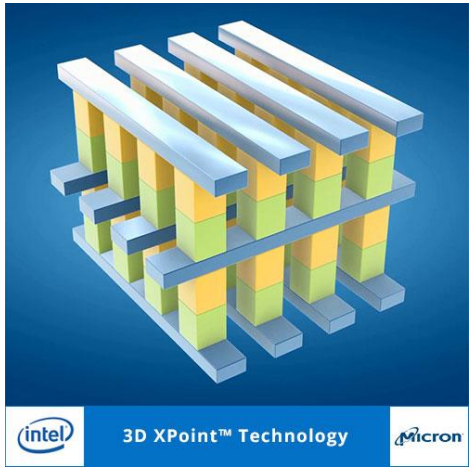
可變電阻式記憶體 **ReRAM**



相變化記憶體 **PCM**



磁阻式隨機存取記憶體 **MRAM**

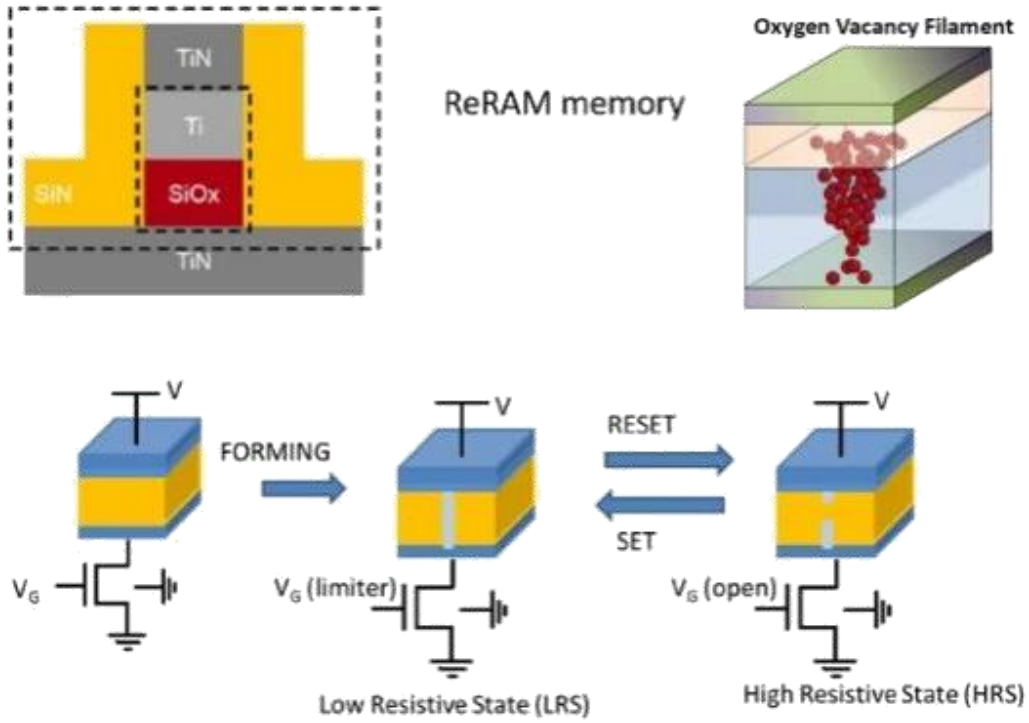


Intel 3D XPoint 記憶體

https://www.st.com/content/st_com/en/about/innovation---technology/PCM.html; https://3c.ltn.com.tw/news/19547; https://www.eettaiwan.com/20211116nt31-spin-orbit-torque-tackles-mram-constraints/

Resistive Random-access Memory, ReRAM

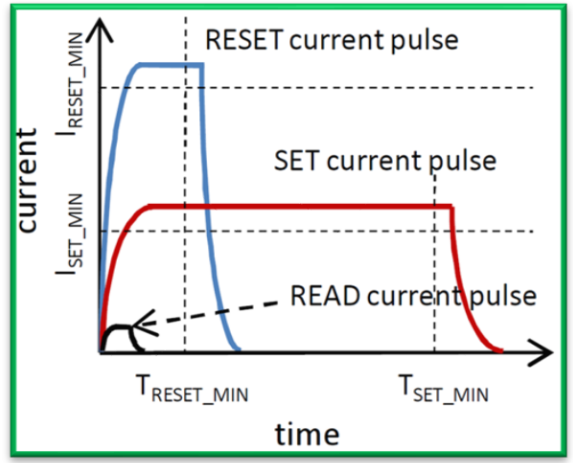
- **Composition**
  - Two layers of metal electrodes
  - A middle layer of transition metal oxide
- **Principle**
  - Applying different voltages changes the resistance of the transition metal oxide
  - By distinguishing between different resistance values, the data can be identified as 0 or 1
- **Characteristics**
  - Fast read/write time
  - Compact and small-area architecture ⇒ higher density and capacity
  - Lower operating voltage ⇒ energy saving
  - Longer lifespan compared to flash memory



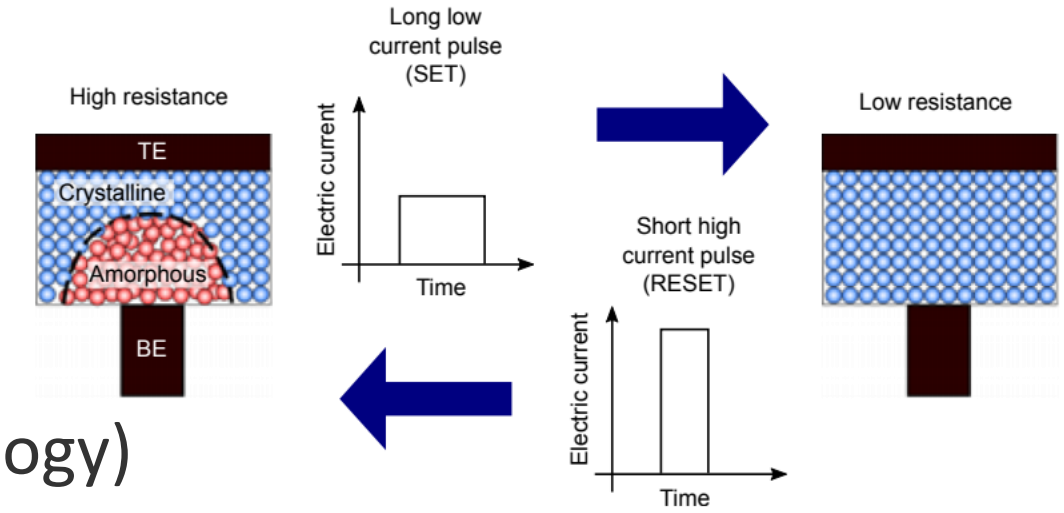
https://www.eettaiwan.com/20200320mt02-is-reram-ready-to-leave-rd-phase/ ; https://zh.m.wikipedia.org/zh-tw/%E5%8F%AF%E8%AE%8A%E9%9B%BB%E9%98%BB%E5%BC%8F%E8%A8%98%E6%86%B6%E9%AB%94

Phase-change Memory, PCM

- **Composition**
  - Made of chalcogenide glass (硫族化物的玻璃)
- **Principle**
  - The properties of chalcogenide glass can be altered by heating, changing its state between crystalline (晶體) and amorphous (非晶體)
- **Characteristics**
  - Fast read speed
  - Write speed about five times slower than read speed
  - Longer lifespan compared to flash
  - Intel Optane memory (3D XPoint technology)

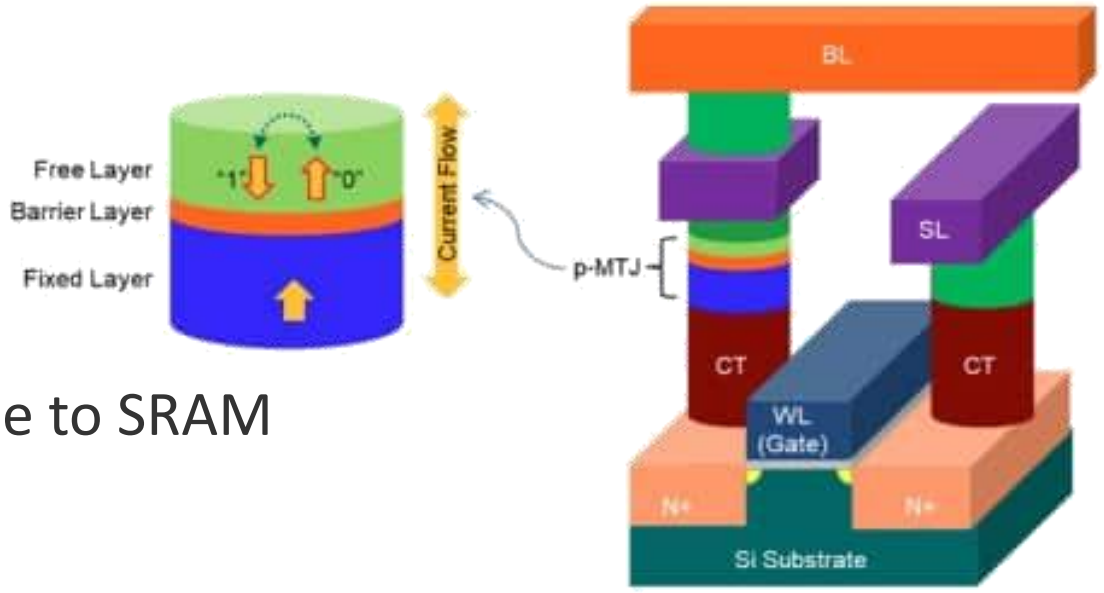


The Asymmetric Write Issue



Magnetoresistive Random Access Memory, MRAM

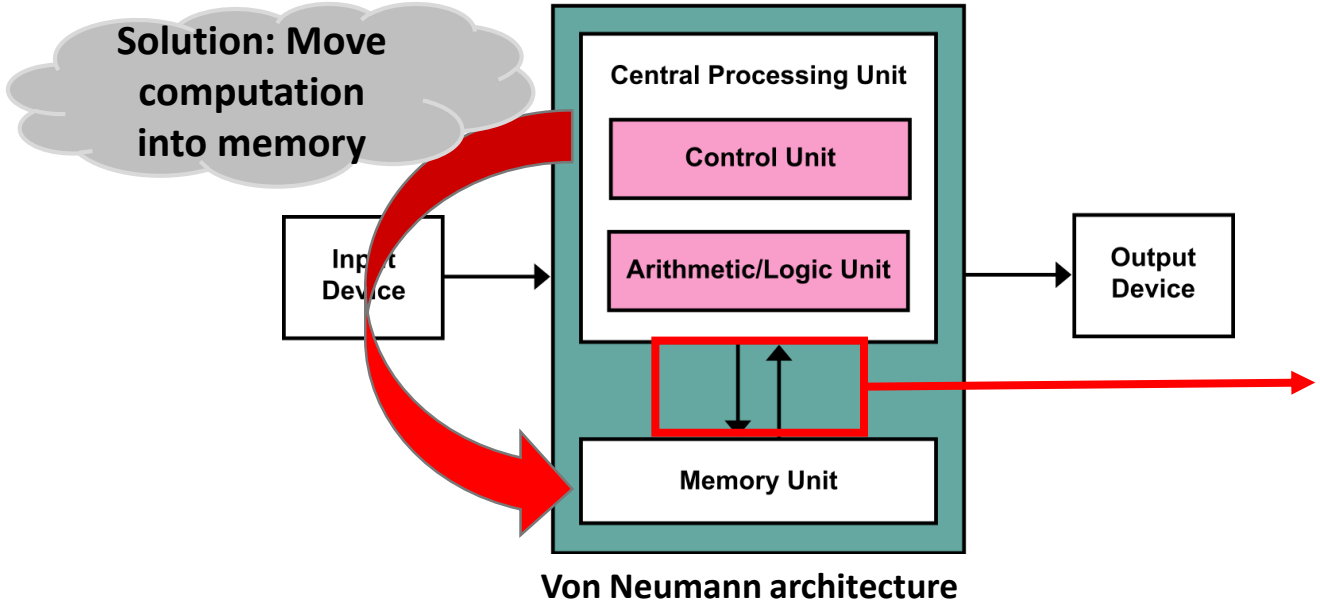
- **Composition**
  - Formed by two ferromagnetic layers
  - **Fixed layer**: a permanent magnet that maintains a specific polarity (blue part)
  - **Free layer**: can change its magnetization direction to store data (green part)
- **Principle**
  - The stored data is determined as 0 or 1 by detecting the magnetization direction of the free layer
- **Characteristics**
  - Potential to achieve read/write speeds close to SRAM
  - Very high density
  - Longer lifespan compared to flash memory

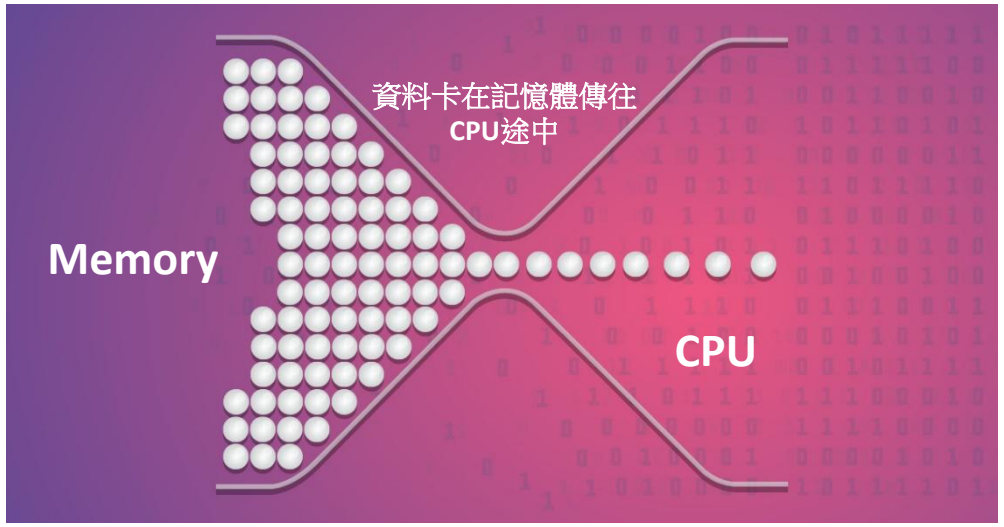


https://www.digikey.tw/zh/articles/how-to-use-mram-to-improve-edge-computing;
https://zh.wikipedia.org/wiki/%E7%A3%81%E9%98%BB%E5%BC%8F%E9%9A%A8%E6%A9%9F%E5%AD%98%E5%8F%96%E8%A8%98%E6%86%B6%E9%AB%94

Von Neumann Bottleneck

- In the current computer architecture, data must be transferred from the memory unit to the CPU for computation.
- However, AI and neural network models require much larger amounts of data than before, and the speed of memory transfer cannot keep up with the processing speed of the compute units, resulting in a memory bottleneck.

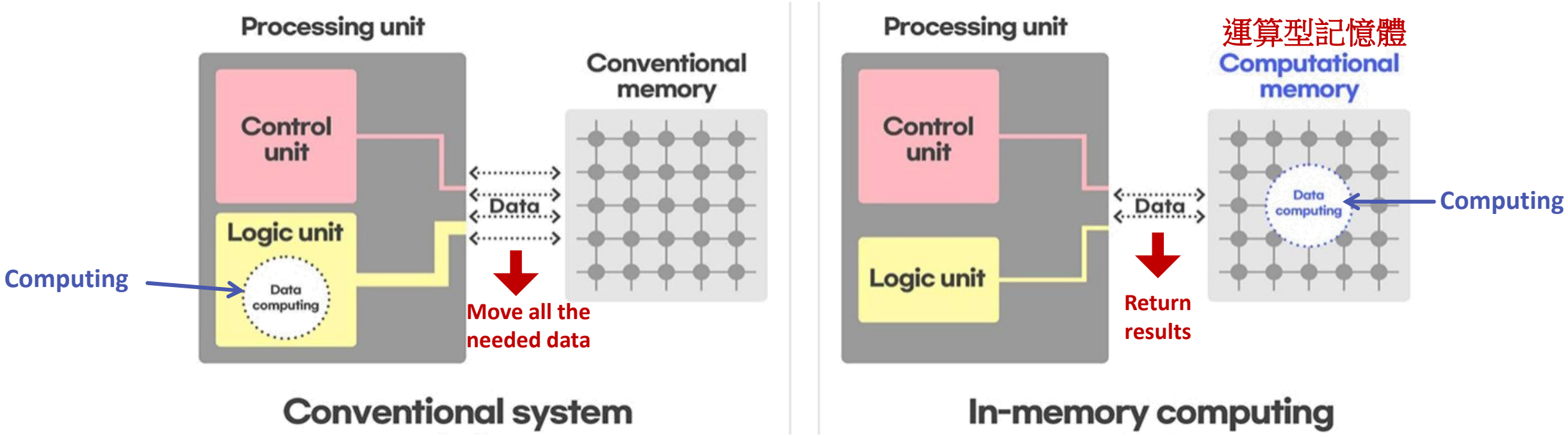




https://en.wikipedia.org/wiki/Von_Neumann_architecture; https://blog.indigovision.com/identifying-and-solving-data-bottlenecks-in-your-surveillance-network;

Processing-in-Memory, PIM

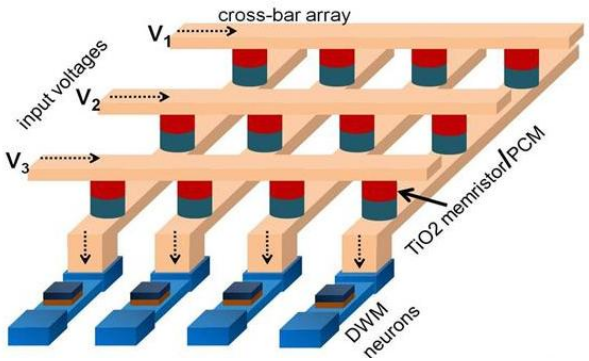
- With processing-in-memory, simple computations (such as multiplication, addition, comparison, and logical operations) can be performed without moving the data.
- The data remains within the processing memory, and only the results are returned, thereby alleviating the original data transfer bottleneck.



https://www.samsungsemiconstory.com/global/samsung-demonstrates-the-worlds-first-mram-based-in-memory-computing/

Memristor Crossbar Array

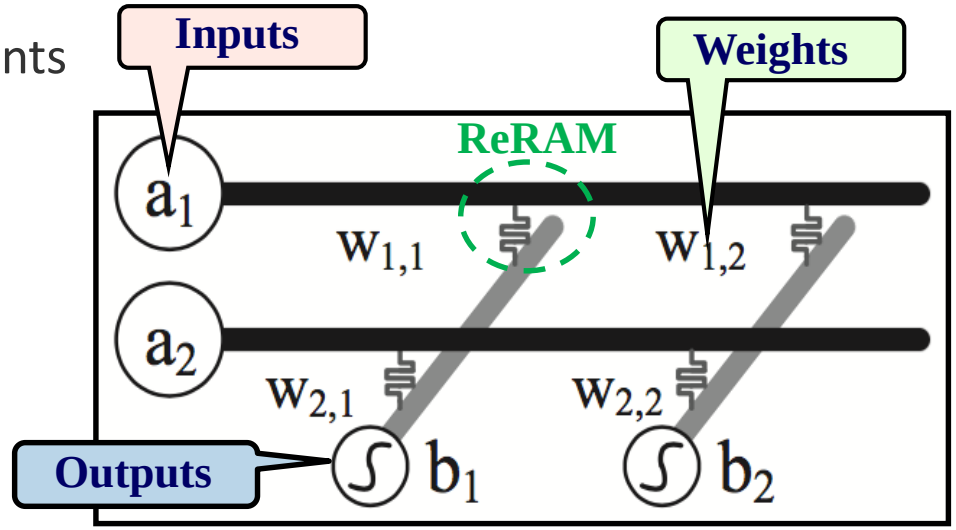
- **Memristor (Memory Resistor)**
  - *Memory* + *Resistor* ⇒ Memristor
  - Non-volatile ⇒ Data can be retained even after power is turned off
  - The resistance value of a memristor can be changed through memory write operations
  - ReRAM and PCM, as mentioned earlier, can be used as memristors
- Memristor Crossbar Array
  - Circuits are arranged in a checkerboard-like crossbar structure, with *intersections* between rows and columns
  - At each intersection, a memristor is connected
  - In the diagram (bottom-left), the red cube at the cross points represents a memristor



Matrix Multiplication:

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1 , a_2 \end{bmatrix} X \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix}$$

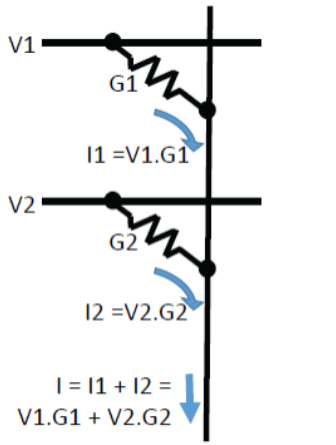
Outputs Inputs Weights



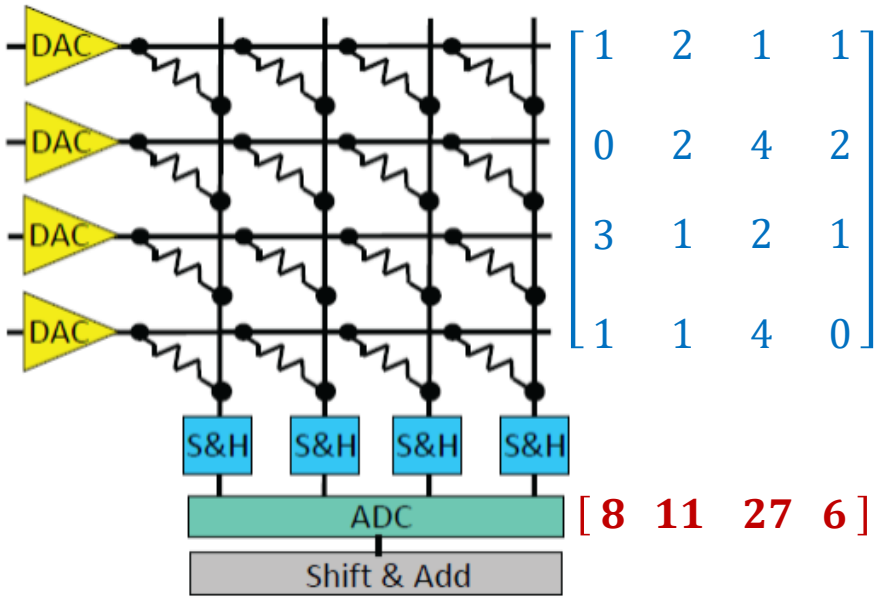
Crossbar Accelerators

Operating Principle of the Memristor Crossbar Array

- In-Memory Multiply-Accumulate (MAC) Operation
  - When an input voltage V is applied along a row and passes through a memristor with conductance G (where G = 1 / resistance R), a current I is generated, with $I = V \times G$.
  - The currents generated across multiple rows are then summed together along the column direction.
  - As shown in the lower-left diagram, the output current **I** in the column direction is the sum of the currents from the first row memristor ($\mathbf{I_1}$) and the second row memristor ($\mathbf{I_2}$).



(a) Multiply-Accumulate operation



$$\begin{bmatrix} 1 & 2 & 1 & 1 \\ 0 & 2 & 4 & 2 \\ 3 & 1 & 2 & 1 \\ 1 & 1 & 4 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 8 & 11 & 27 & 6 \end{bmatrix}$$

Each read on a column is a MAC of two vectors, where MAC is "Multiply and Accumulate"

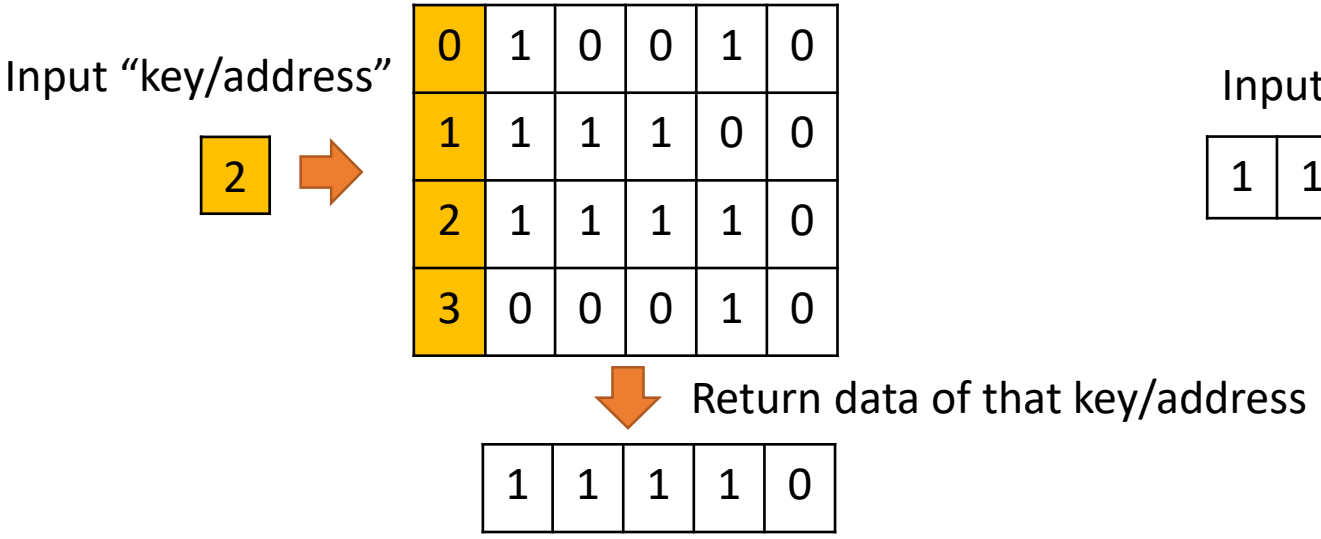
Shafiee, Ali, et al. "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. "*ACM SIGARCH Computer Architecture News* 44.3 (2016): 14-26.

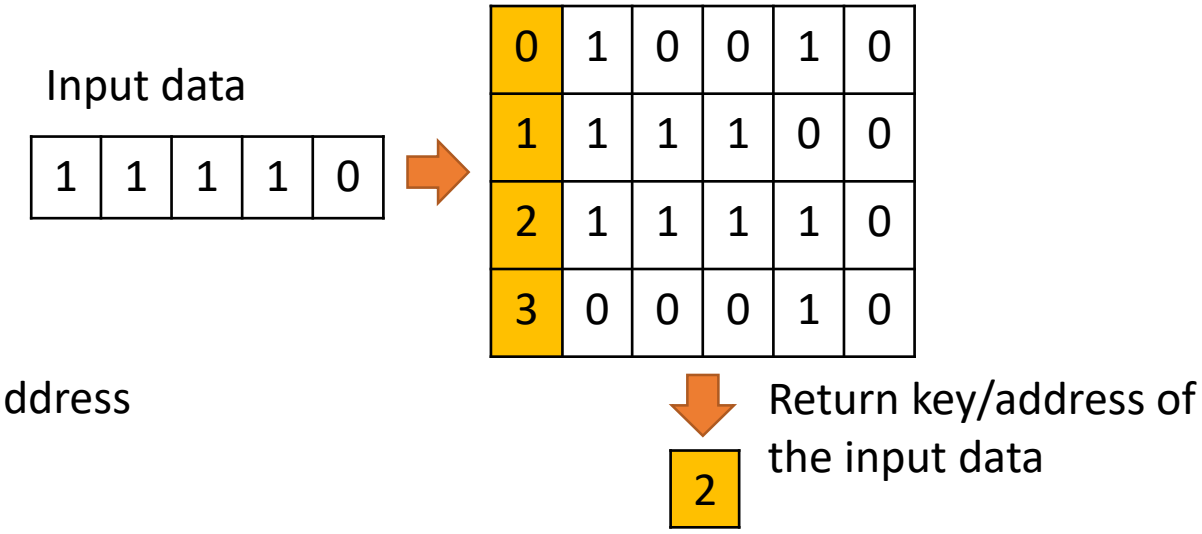
Content-Addressable Memory, CAS

- In-memory-searching

Random Access Memory (RAM)
- Read/write specific addresses
- Operates based on the given address

Content-Addressable Memory (CAM)
- Used to search whether specific data exists in memory
- Returns the address of the data
- Performs parallel comparison between the given data and the data stored in memory



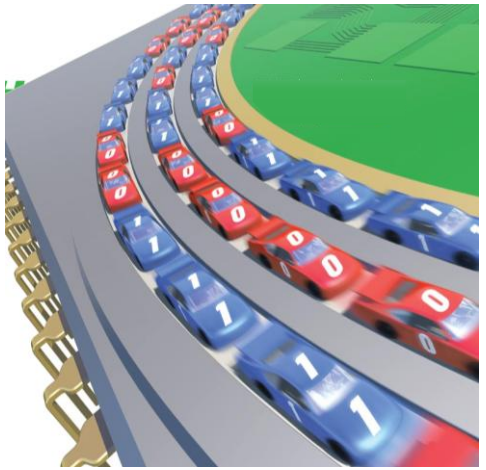


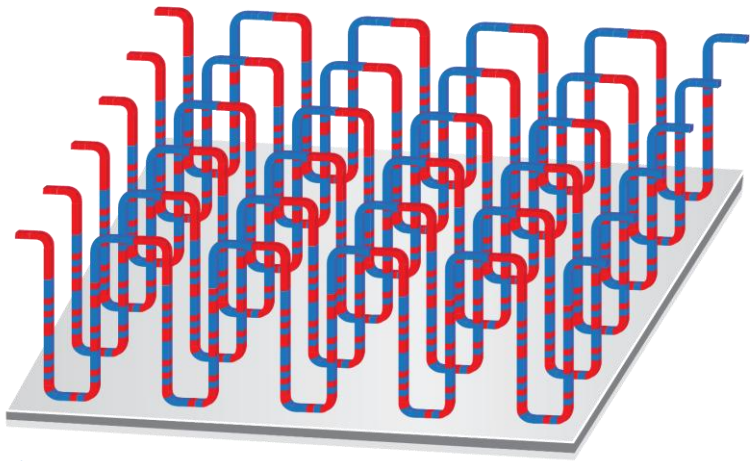
Outline

- Early Data Storage
- Modern Data Storage
- Data Storage in Daily Life
- Next-Generation Data Storage
- **Future Data Storage**

Future Data Storage – Racetrack Memory (RM)

- Racetrack memory is considered a rising star among non-volatile memories, expected to provide:
  - Read/write performance comparable to SRAM
  - Storage density comparable to flash memory
- In 2008, the team led by **Stuart Stephen Papworth Parkin** from the IBM Research Center in the UK pioneered a 3-bit racetrack memory.

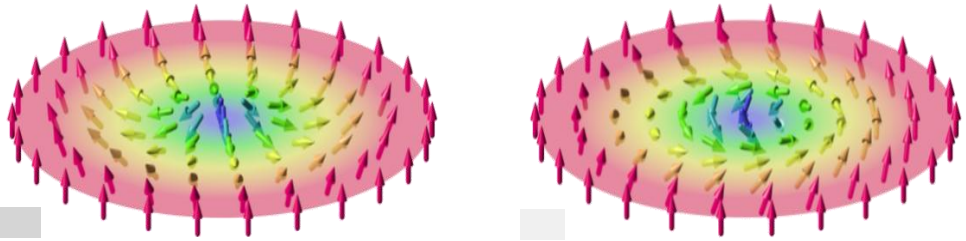




https://www.aip.org/stuart-parkin;
Parkin, Stuart S. P. "Data in the Fast Lanes of *RACETRACK MEMORY*." *Scientific American*, vol. 300, no. 6, 2009, pp. 76–81. *JSTOR*, http://www.jstor.org/stable/26001383. Accessed 26 Sep. 2022.

Skyrmion Racetrack Memory, SK-RM

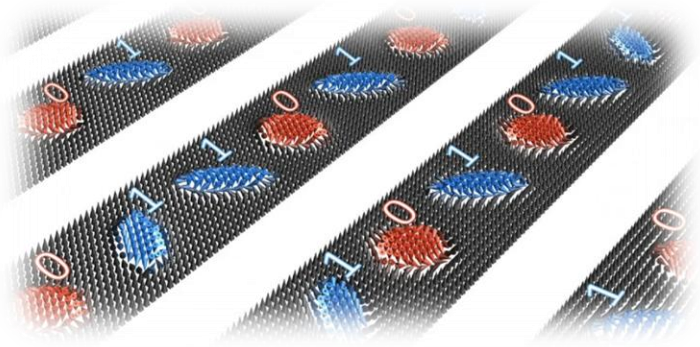
- In 1962, British physicist Tony Hilton Royle Skyrme discovered the magnetic skyrmion, a stable magnetic structure.
- By generating magnetic skyrmions on ultrathin metallic racetracks, the storage density of conventional racetrack memory can be greatly increased. This technology is called skyrmion racetrack memory (SK-RM).



(a) Hedgehog Skyrmion (刺猬斯格明子)

(b) Spiral Skyrmion (螺旋斯格明子)

Different Types of 3D Magnetic Skyrmions



Different Magnetic Skyrmions on the Racetrack

Advantages of Skyrmion Racetrack Memory

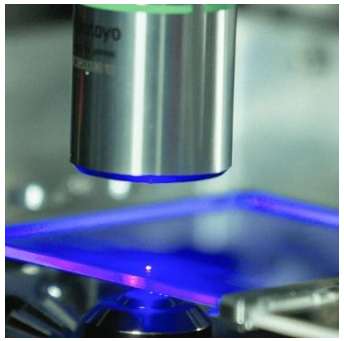
- **Ultra-high performance** ⇒ Comparable to SRAM with extremely fast access speed.
- **Ultra-high storage density** ⇒ Expected to surpass the extremely high density of flash memory and hard drives.
- **Ideal energy efficiency** ⇒ Both operation and access consume very little power.
- **Non-volatility** ⇒ Can retain data long-term without continuous power supply.
- **Longer memory lifespan** ⇒ Since data is stored and accessed based on physical magnetic principles, its lifespan may exceed that of flash memory and similar technologies.
- **Native support for Processing-in-Memory (PIM) technology** ⇒ Enables direct and rapid execution of massive logical or arithmetic operations within memory, without relying heavily on the processor.

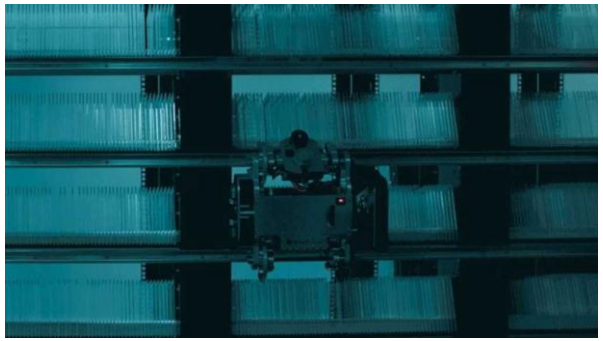
Advantages of Skyrmion Racetrack Memory

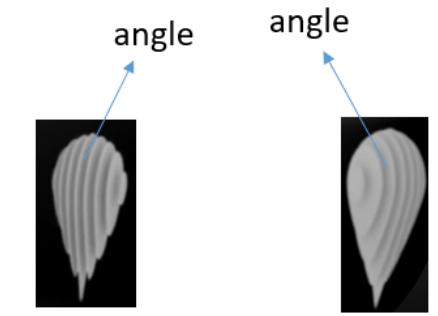
		Off-the-Shelf				Prototyping		
	Memristor	SRAM	DRAM	NAND flash	硬碟	PCM	STT-RAM	**SK-RM**
Density (F^2)	< 4	140	6–12	1–4	2/3	4–16	20–60	**High**
Energy (pJ)	0.1–3	0.0005	0.05	0.00002	$(1-10)^{10}$	2–25	2.5	**High**
Read latency (ns.)	< 10	0.1–0.3	10	100,000	$(5-8)^6$	$10-50$	10–35	**Short (~1 ns.)**
Write latency (ns.)	~10	0.1–0.3	10	100,000	$(5-8)^6$	50–500	10–90	**Short (~5 ns.)**
Retention time	Short	Short	Short	Long	Long	Long	Long	**Long**
Endurance (P/E cycles)	10^{12}	10^{16}	10^{16}	10^4	10^4	10^6	10^{15}	**High**

Glass Storage

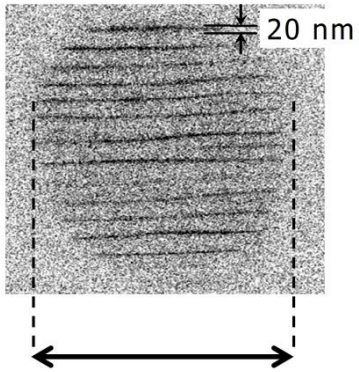
- Glass represents 0 and 1 based on the result of differer (相位延遲) and polarization angles (偏振角).
- Data can be stored for more than 10,000 years.



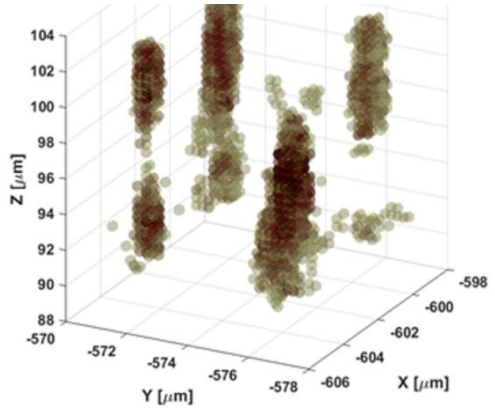




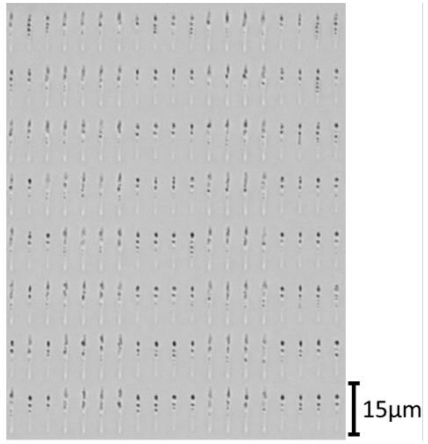
These voxels serve as the basic units of data, encoding bits through properties such as birefringence, retardance, and polarization angle.



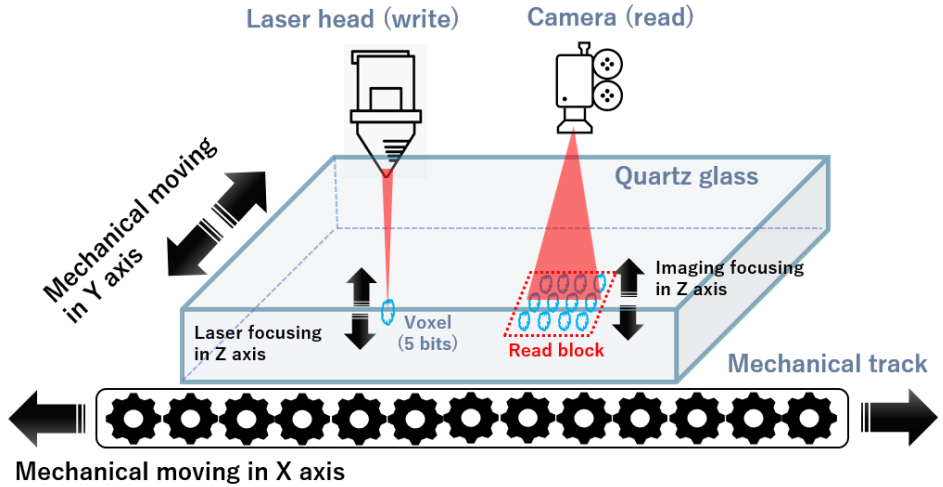
(a) Single voxel (top view)



(b) 3D reconstruction of four measured voxels

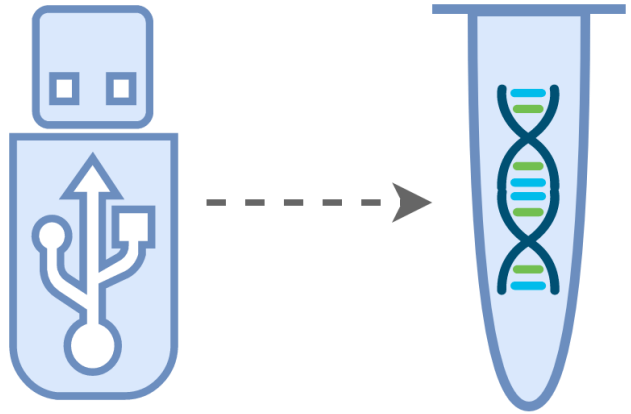


(c) Side view (8 layers)



Future Data Storage – DNA Storage

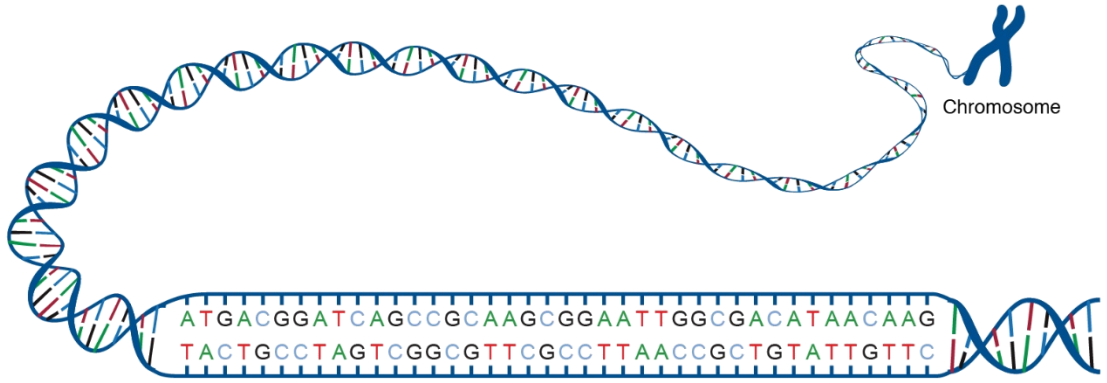
- In 2020, a study reported that the total amount of stored data in the world had reached 44 ZB.
- Each year, 16 ZB of new data is generated.
- Scientists estimate that by 2040, silicon resources will be exhausted, prompting the search for alternative storage solutions.



2016 → 4.4 ZB
2020 → 44 ZB
2025 → 160 ZB

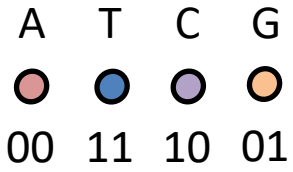
DNA Storage

- The most basic unit of computer data: 0 and 1.
- The most basic units of DNA: adenine (A), thymine (T), guanine (G), and cytosine (C).
- The human body is the most complex information repository.



DNA Nitrogenous bases (鹼基)	Data Encoding
A	00
C	01
G	10
T	11

DNA Storage



Input DNA

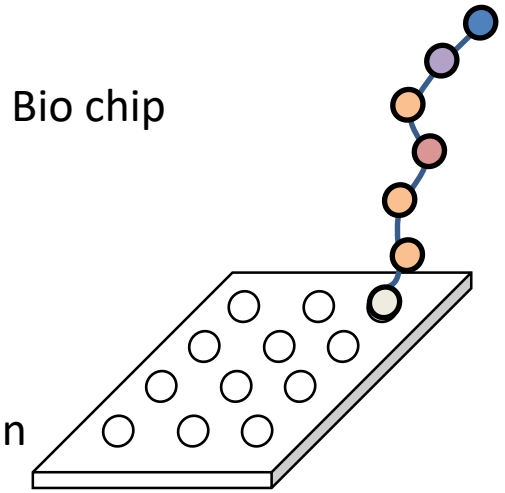
Digital Data

01 01 00 01 10 11……

Step 1

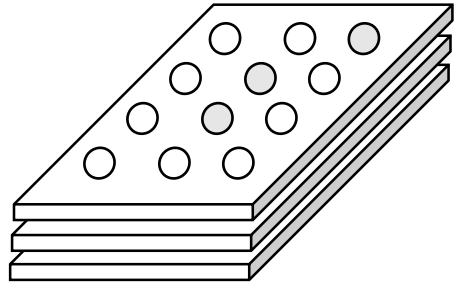
Data conversion

Bio chip

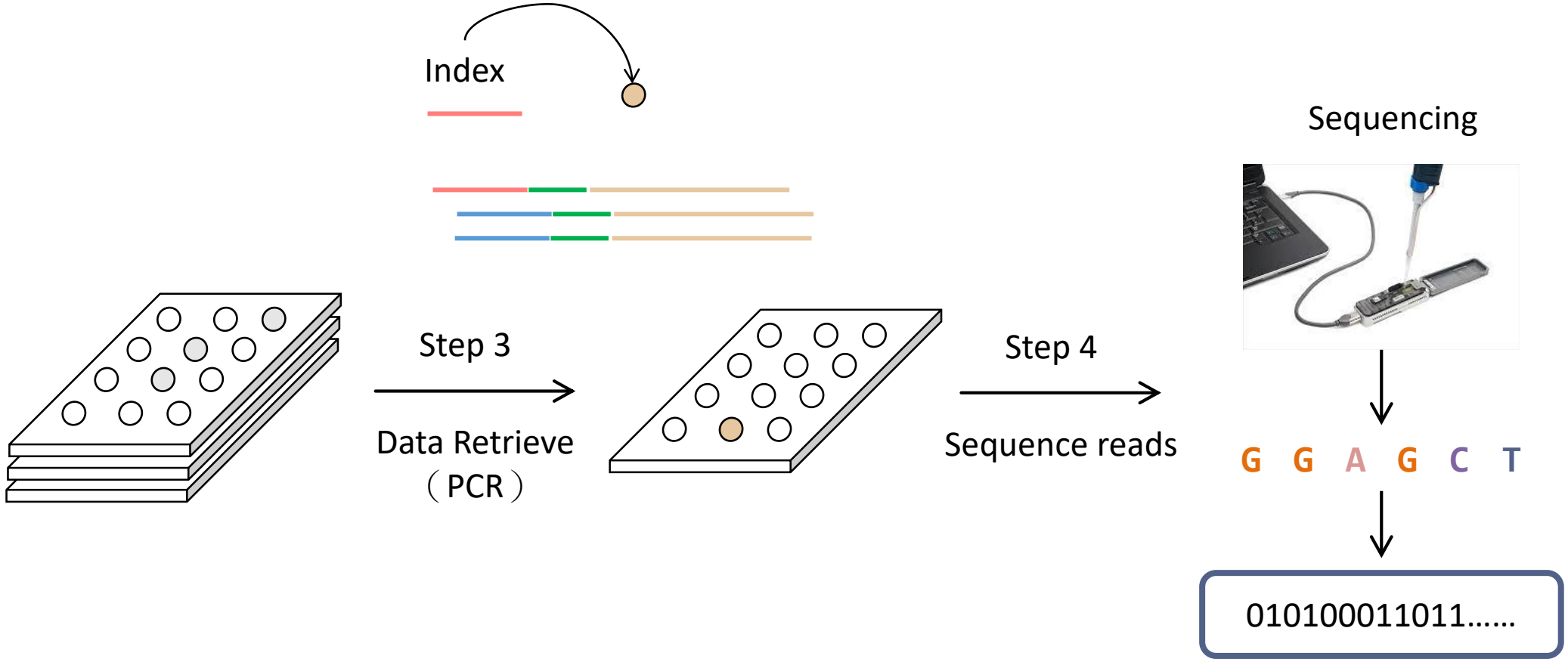


Step 2

Data writing



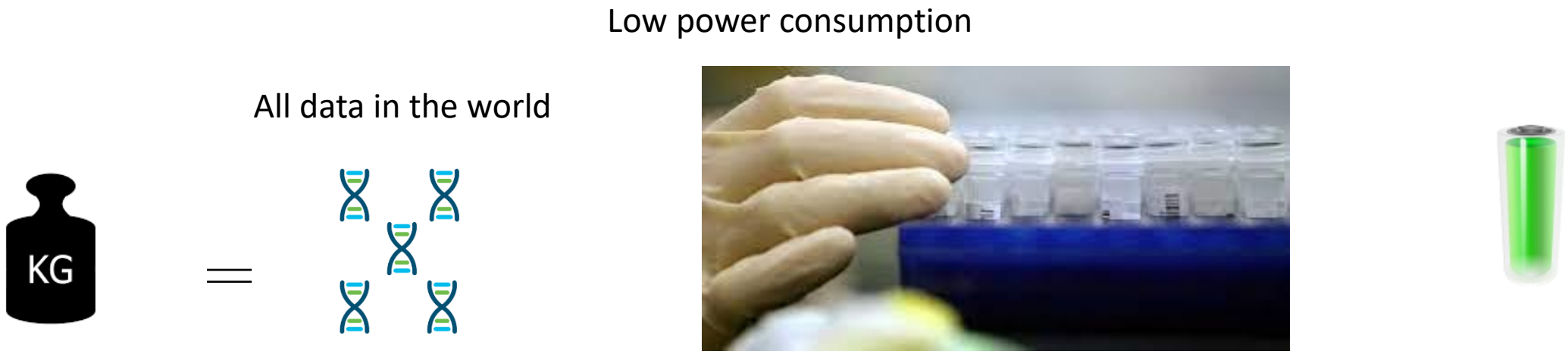
DNA Storage



https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTw5cDVQP4tjjYuvgphrysdk4F9IZSnYRw5HIc0AEh6l4sF1ObN

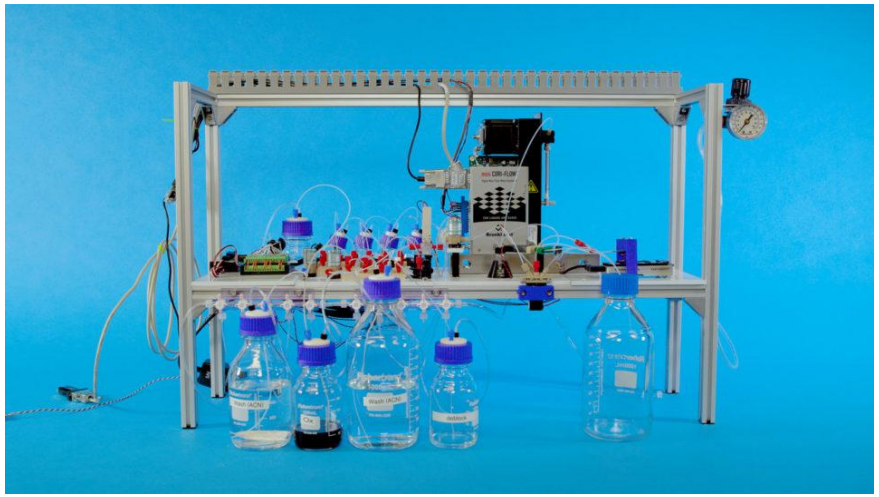
DNA Storage - Advantages

- Advantages
  - Large capacity and high density (1 gram of DNA = 215 million GB)
  - Long preservation period (under suitable conditions)
  - Low power consumption



DNA Storage - Challenges

- Challenges
  - How to achieve automation
  - Very long read and write times
  - Extremely high cost of data access



Conclusion

- Flash memory and HDDs have provided huge storage capacity.
- Non-volatile memories (NVMs) provide new possibility to serve as both memory and storage with even higher capacity.
- Upon facing von Neumann bottleneck, NVMs with processing-in-memory provide possibility to resolve this issue but face many design challenges.
- In the future, we need more storage capacity and more endurable storage media, and let us study how glass and DNS storage could lead us to the next storage generation.

國立臺灣大學
National Taiwan University



Unleashing the Potential of In-Memory Computing

Yuan-Hao Chang

Department of Computer Science and Information Engineering,

National Taiwan University, Taipei, Taiwan

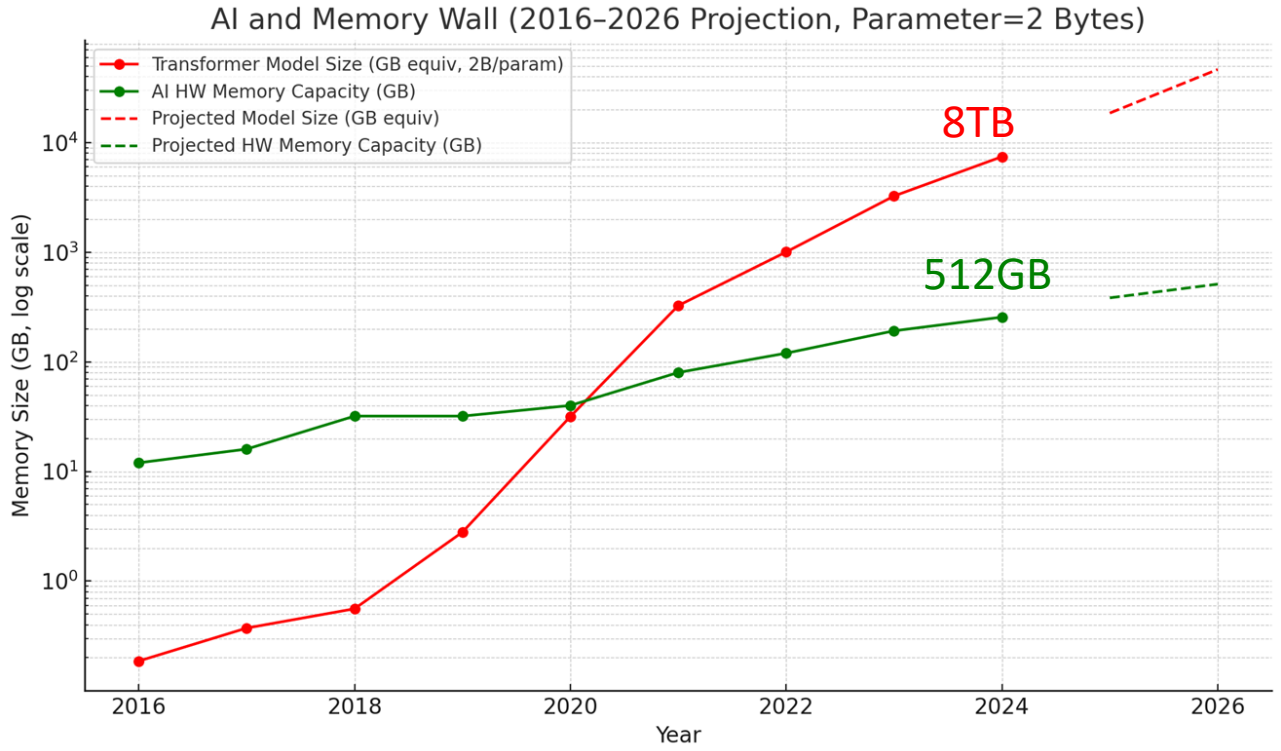
October 17, 2025

Outline

- **Introduction**
- ReRAM-based In-Memory Computing
- Flash-based In-Memory Computing
- Conclusion

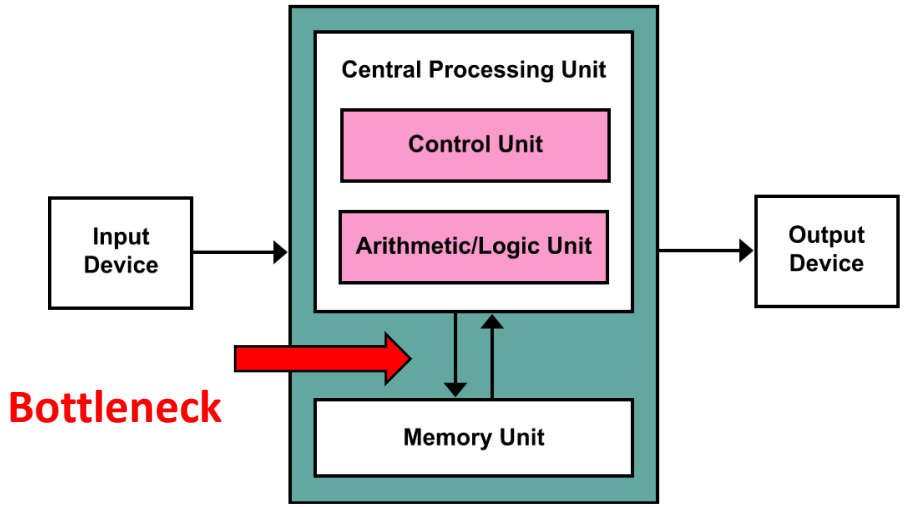
Tremendous Demands and Opportunities in the Era of Artificial Intelligence and Big Data

- To enhance the performance of AI model, more parameters are needed
  - Hardware needs higher memory bandwidth and capacity to support novel AI models
- Memory wall issue in AI
  - Growth of AI HW memory << Growth of model size (2024: Model size 8TB, DRAM 512GB)

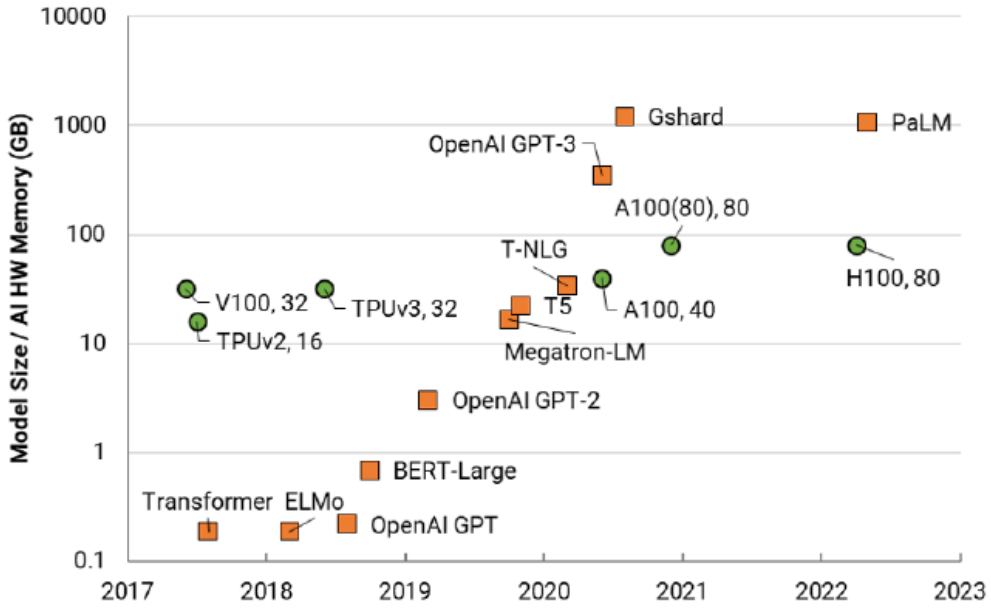


Fundamental Problems in Running AIs

- Bottleneck of von Neumann architectures
  - Memory wall: Lacks of bandwidth (for the growth of AI models)
  - **Tremendous data movement** (Processing Unit ↔ Memory unit)
- Growth of model size >> Growth of GPU memory
  - GPT-3 model(2020) >> H100 GPU(2022)
  - Need multiple GPUs => **High Cost**



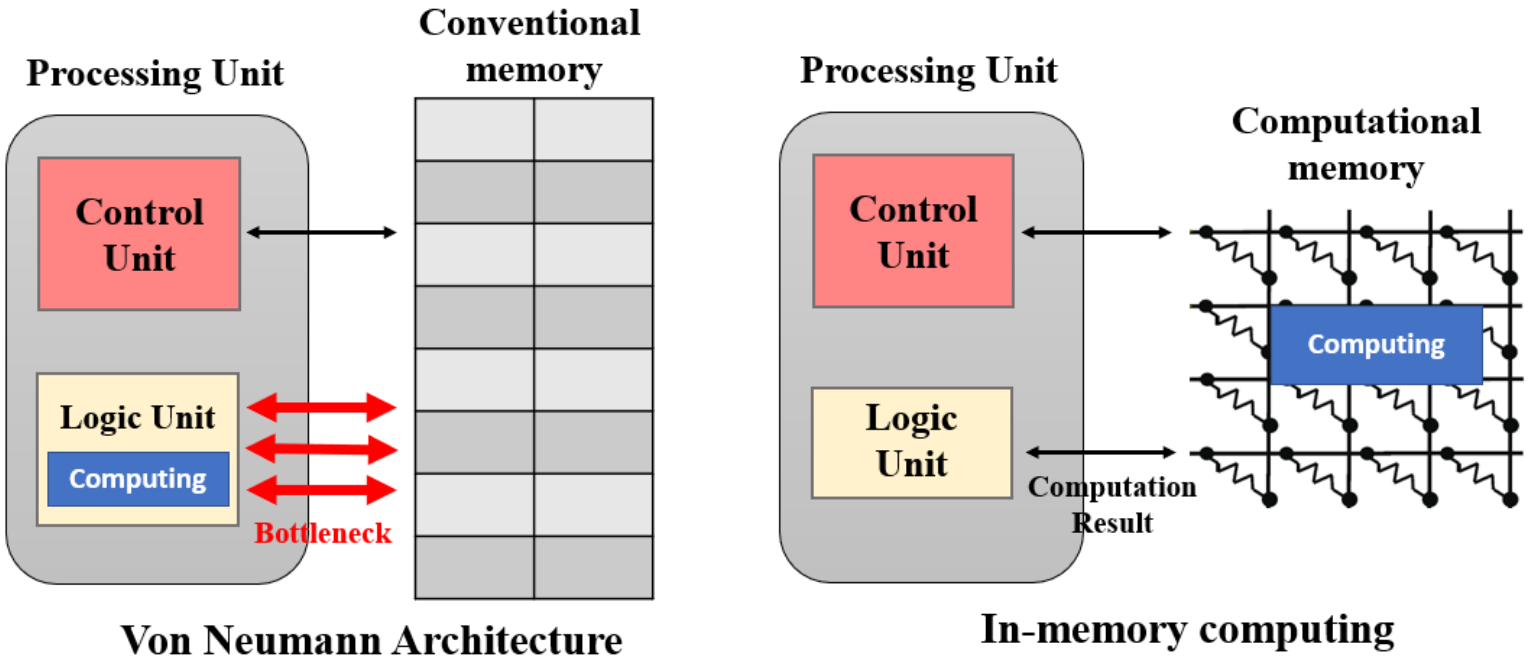
Ref: https://towardsdatascience.com/machine-learning-fundamentals-ii-neural-networks-f1e7b2cb3eef



Ref: Kim, J. et,al. OptimStore: In-Storage Optimization of Large Scale DNNs with On-Die Processing. In 2023 HPCA IEEE.

A Potential Solution

- Memory-Centric Computing – In-Memory Computing (or Processing-in-Memory)
  - Offload the computation into the memory unit (and even **the storage unit**)
  - Compute the computation during data access
  - Resolve the bottleneck of von Neumann architecture
  - Computational memory – **Crossbar array** (analog MAC) with non-volatile memory (NVM)



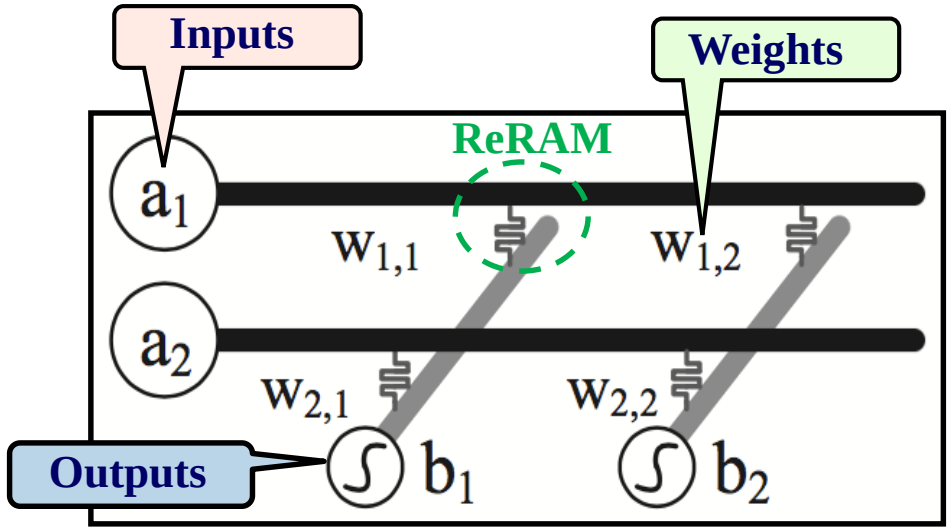
NVM-based Crossbar

- ReRAM-Based Crossbar for In-Memory Computing
  - **Crossbar:** Wordlines and bitlines are orthogonal in three-dimensional (3D) space, and ReRAM is used to connect them.
  - **ReRAM (or called Memristor):** It works by changing the resistance of the memory cell to represent different data states (e.g., 1 or 0).

Matrix Multiplication:

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1 , a_2 \end{bmatrix} X \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix}$$

Outputs Inputs Weights

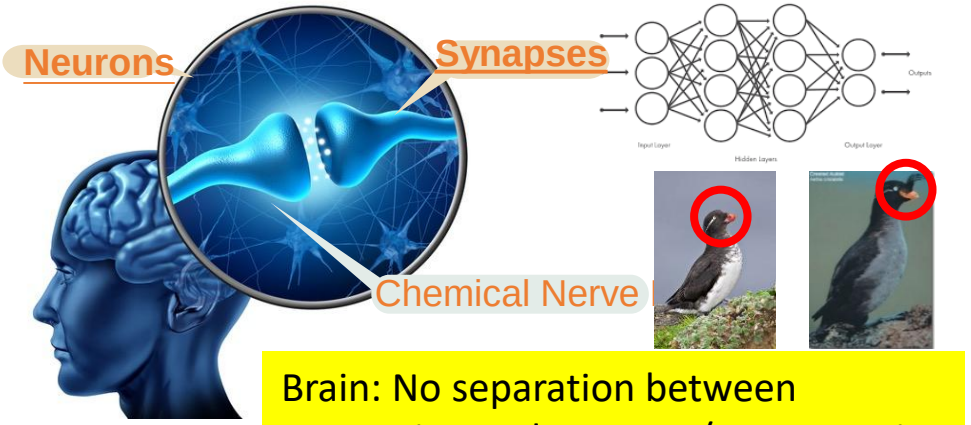


Crossbar Accelerators

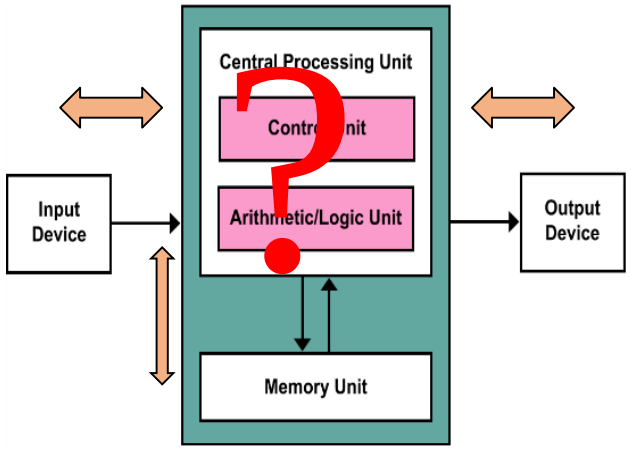
Source: Shafiee, Ali, et al. "**ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars.**" Proceedings of the 43rd International Symposium on Computer Architecture. IEEE Press, 2016.

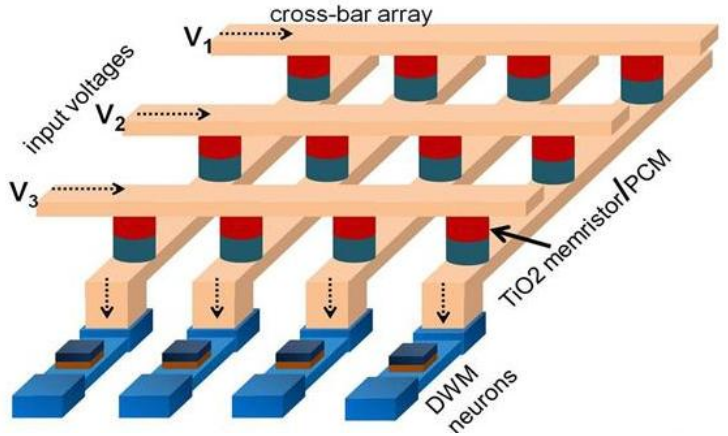
Rethinking of Computing, Memory, and Storage

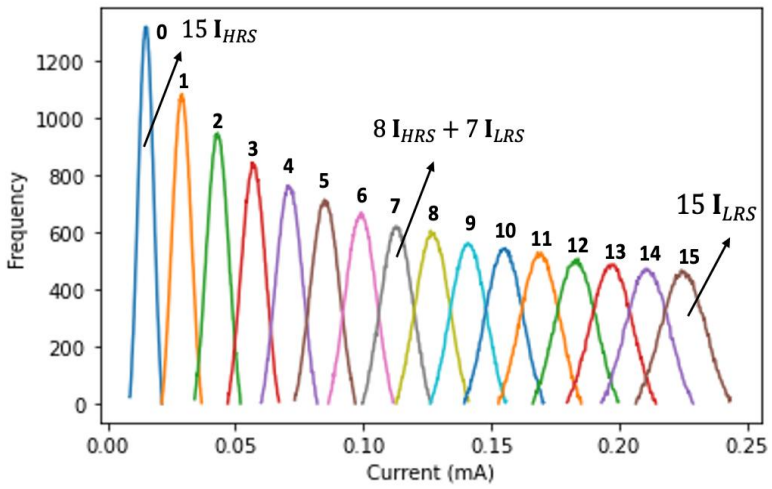
- Challenges in (Crossbar) In-Memory Computing
  - Reliability (Error rate)
  - Scalability (Space utilization)
  - Functionality (MAC, TCAM, Range)
  - Capacity (ReRAM vs Flash)



Brain: No separation between computing and memory/storage units







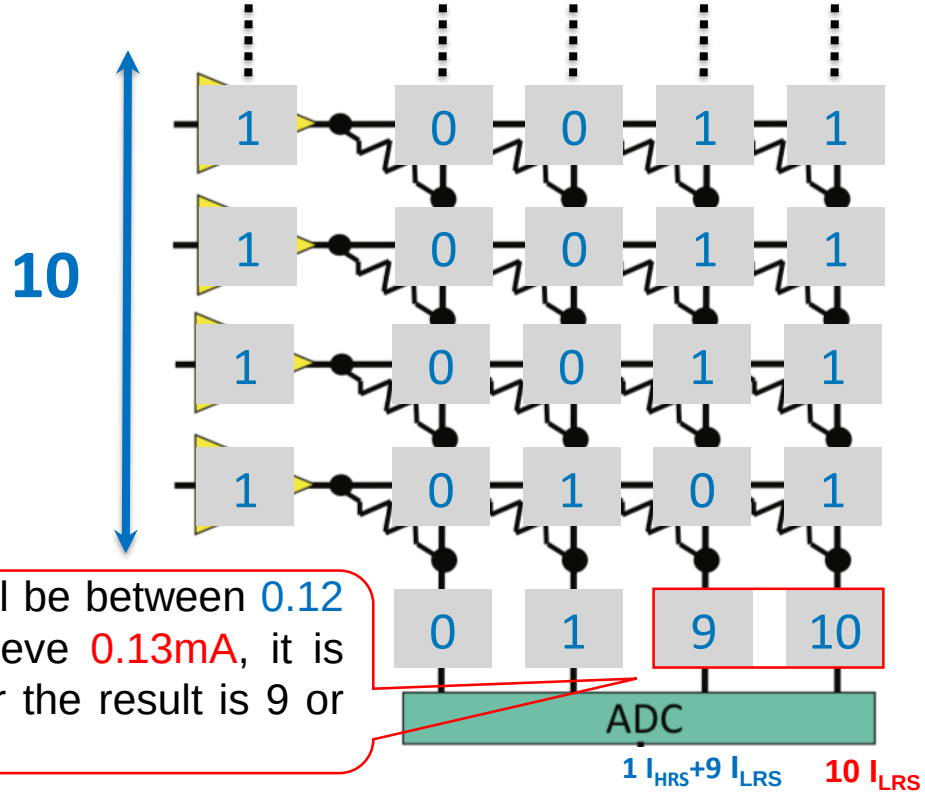
Outline

- Introduction
- **ReRAM-based In-Memory Computing**
- Flash-based In-Memory Computing
- Conclusion

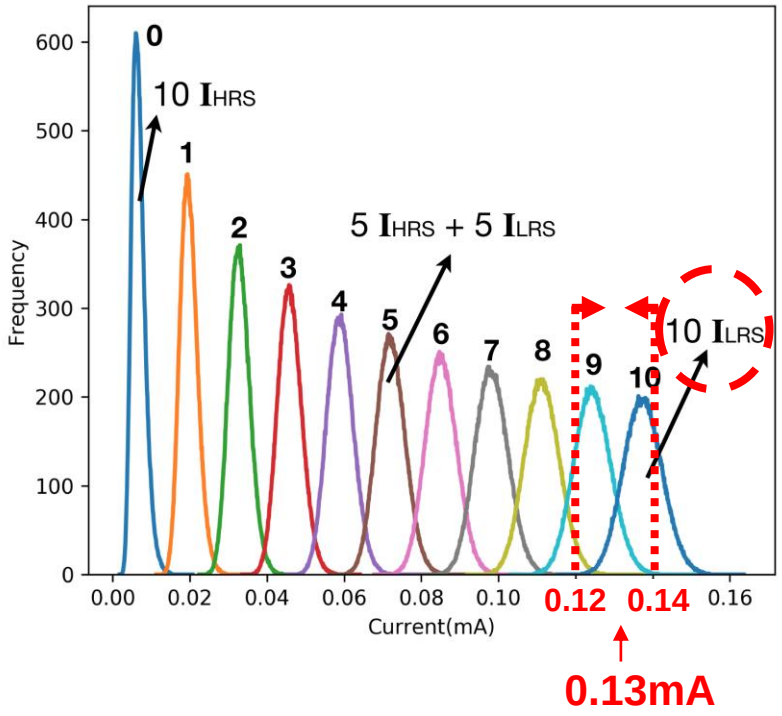
Minimizing Analog Variation Errors to Resolve the Scalability Issue of ReRAM-based Crossbar Accelerators

Challenges of ReRAM Crossbar: Overlapping Variation Error

- **Overlapping Analog Variation Error:** The accumulated currents in each bitline will be possibly converted to an incorrect digital value during the **analog to digital conversion**
  - The more Low Resistance States (LRS, 1) involved, the more serious the error is.
  - Scalability Issue: Overlapping Variation Error becomes more serious when more valid wordlines are involved.



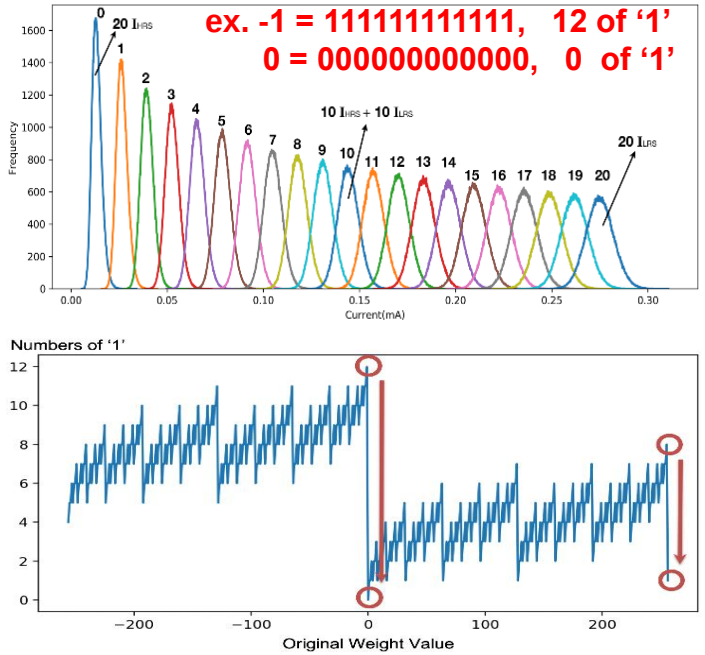
The output current will be between 0.12 – 0.14mA. If we retrieve 0.13mA, it is hard to judge whether the result is 9 or 10!



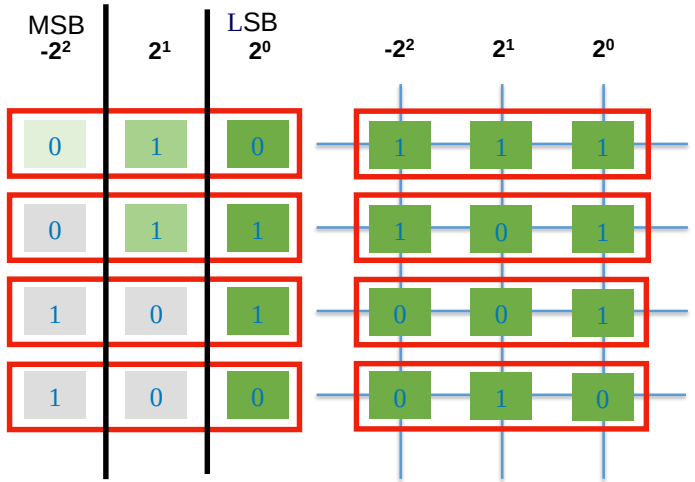
Minimizing Analog Variation Errors to Resolve the Scalability Issue of ReRAM-based Crossbar

- We are the pioneers to resolve the scalability problem by tackling the **analog variation error** for realizing *large-scaled* ReRAM-based crossbar accelerators
  - We tackle the representation of both inputs and model weight values.
  - We propose **an adaptive data manipulation design** that can improve the accuracy in running MNIST and CIFAR-10 by 1.3x and 2.6x respectively because of the significantly reduced analog variation errors.

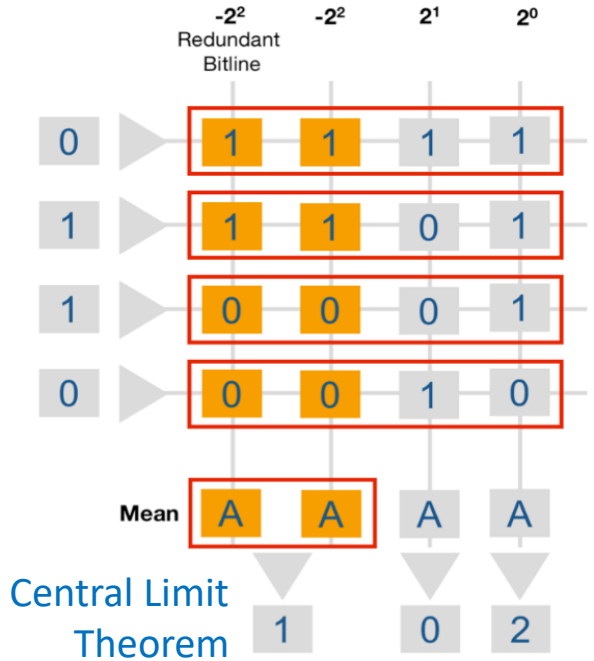
- **Weight-rounding design (WRD)**



- **Adaptive input subcycling design (AISD)**



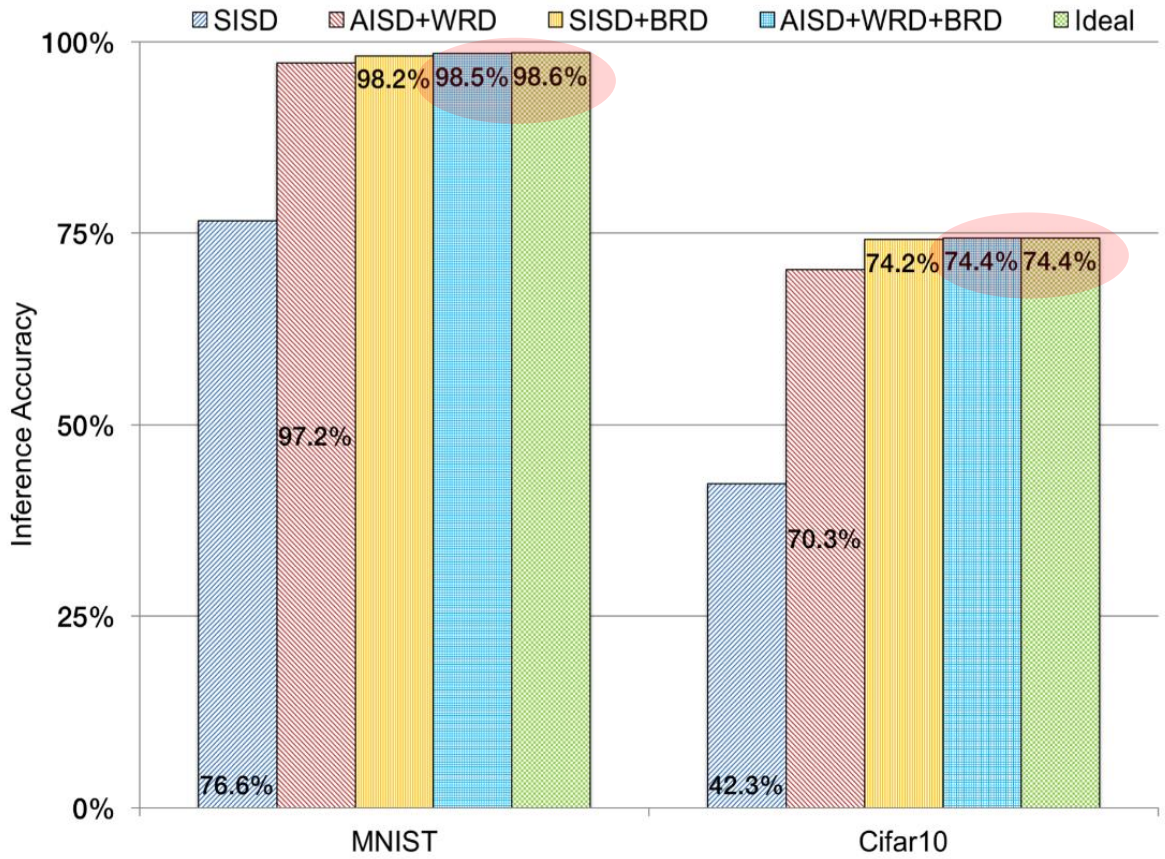
- **Bitline redundant design (BRD)**



Evaluation

- The proposed design can achieve almost the same accuracy as an ideal case, and these results imply that our design can successfully eliminate the overlapping variation error to resolve the scalability issue of ReRAM-based crossbar.

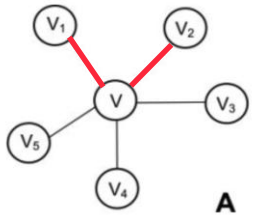
Weight Running Distance (WRD)
Adaptive Input Subcycling Design (AISD)
Bit Redundant Design (BRD)

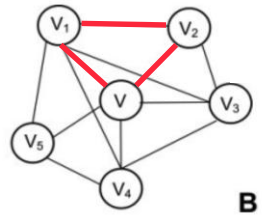


Space-efficient Graph Partitioning to Improve Performance and Space Utilization of ReRAM Crossbar [ICCAD'22]

Space Efficiency of Graph Partitioning for Crossbar

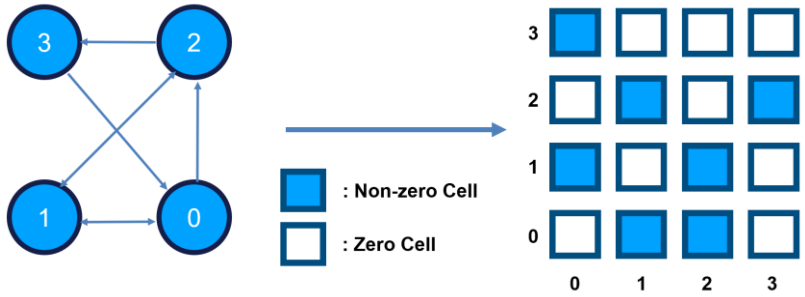
- **Graph Processing**
  - Many problems can be formulated as a **graph** and graph algorithms on Crossbar can be implemented by some form of matrix-vector multiplications
- **Adjacency Matrix for Original Graph Indices**
  - In general, graph data can be represented by an adjacency matrix
- **Observation**
  - Low space utilization
  - Low computational intensity
  - Irregular data access

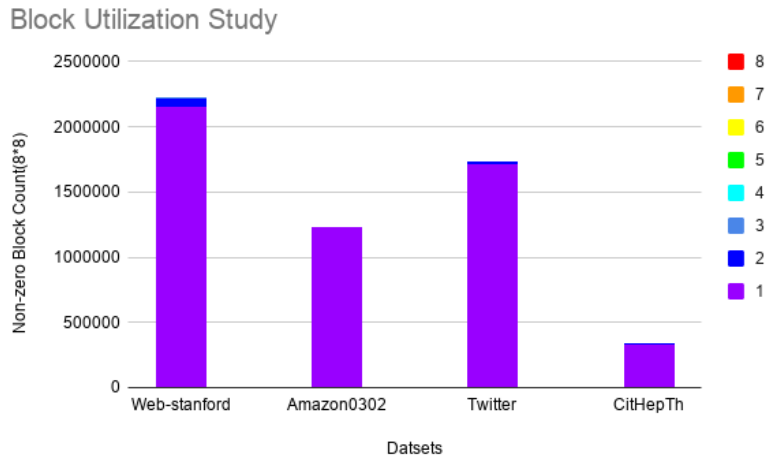




$$For\ an\ adjacency\ matrix: A$$

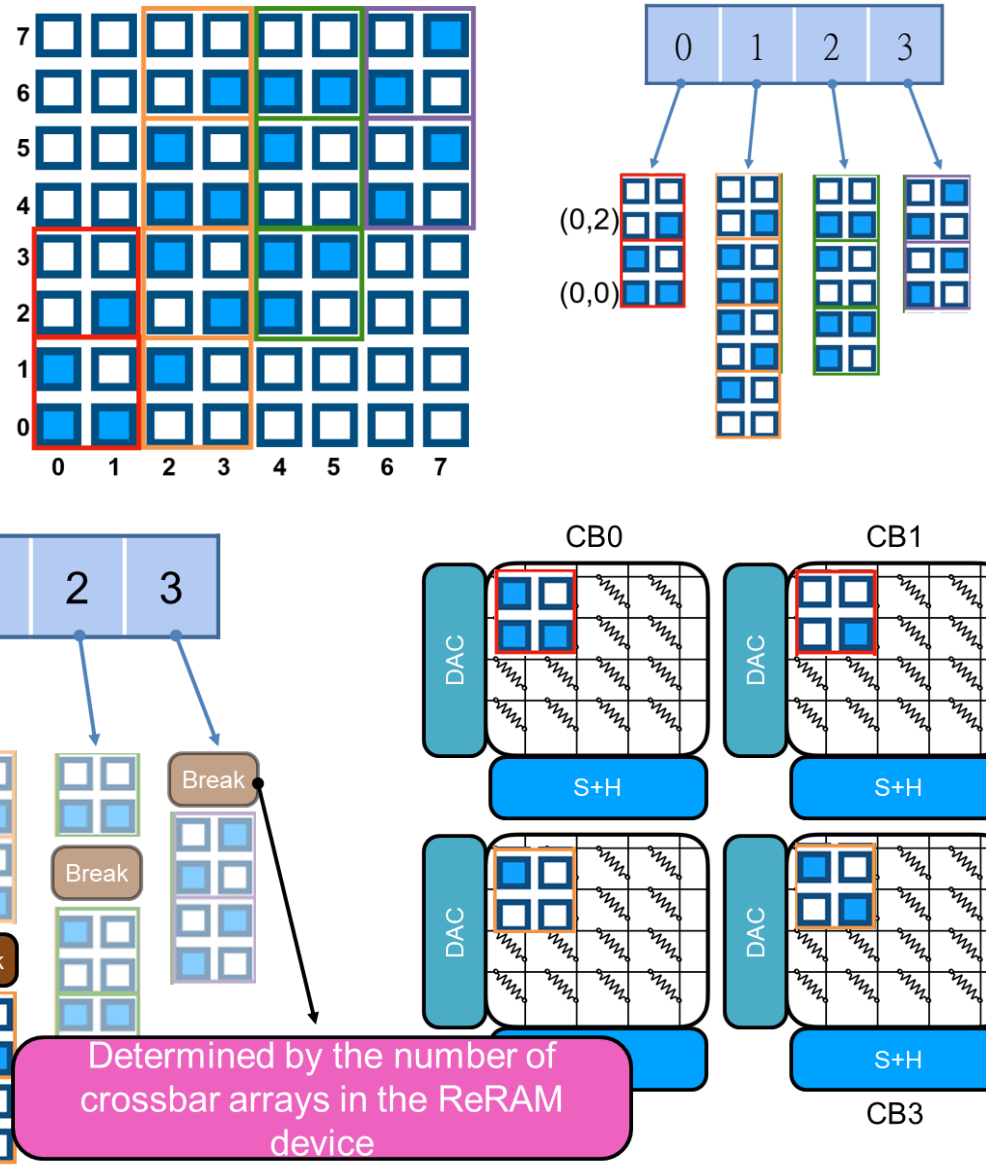
$$Triangle\ Counting = \frac{Trace(A^3)}{6}$$





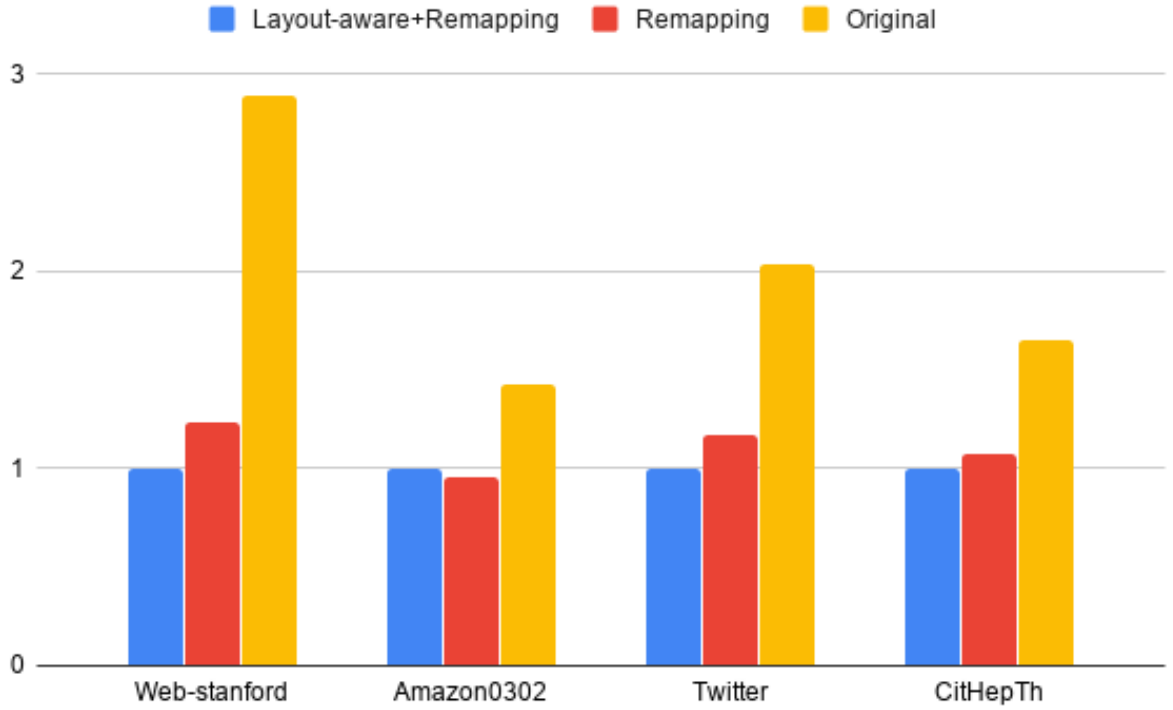
Our Method – Space-Efficient Graph Partitioning

- We are one of the pioneers to tackle both the **irregular access** and **space utilization** issues for **ReRAM Crossbar**
  - Reshape the adjacency matrix into crossbar efficiently
    - **Reorder the rows and columns to make all the non-zero values centralized to the diagonal of the matrix (done by BFS-like algorithm)**
      - Improve computational intensity by offline index remapping
  - Layout-aware graph data placement on ReRAM Crossbars
    - Data placement considering imbalanced edge distribution to overcome space utilization waste
    - Evenly distribute all the sub-matrixes with non-zero values to all the different crossbar arrays, so that all the crossbar arrays can compute in parallel

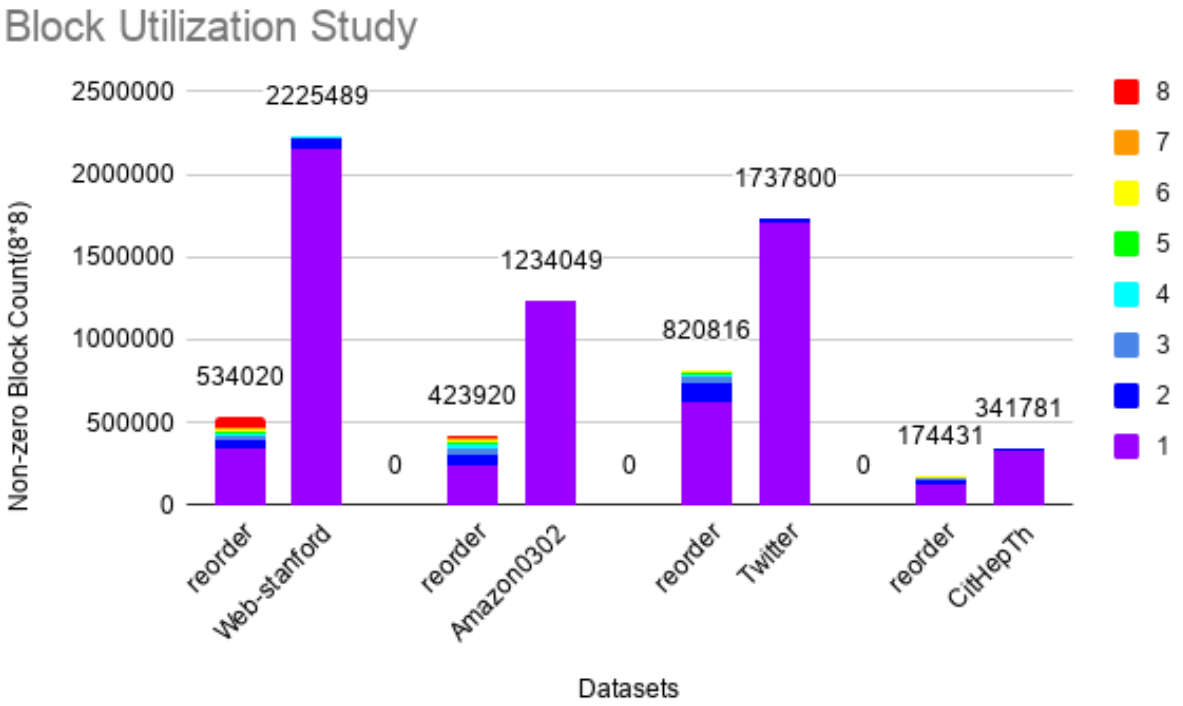


Evaluation

- Energy
  - 1.86x and 2.1x energy reduction with index remapping and remapping+layout-aware



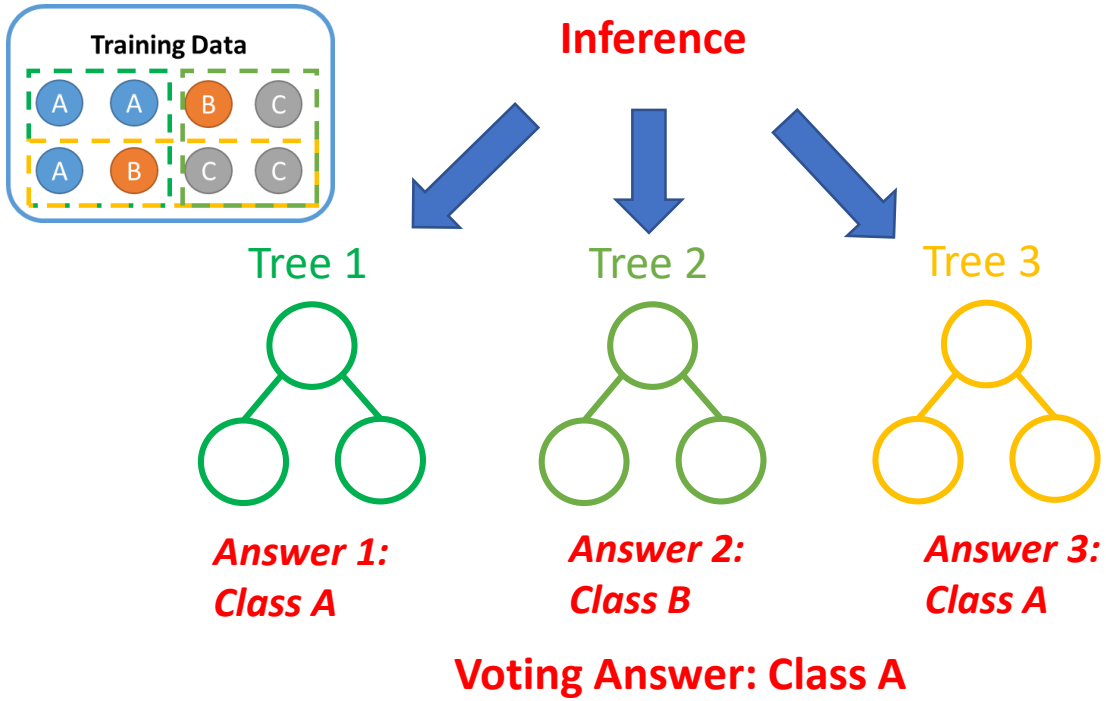
- Space utilization
  - 2.79x Space utilization improvement without hardware modification



A Digital 3D TCAM Accelerator for the Inference Phase of Random Forest [DAC'23]

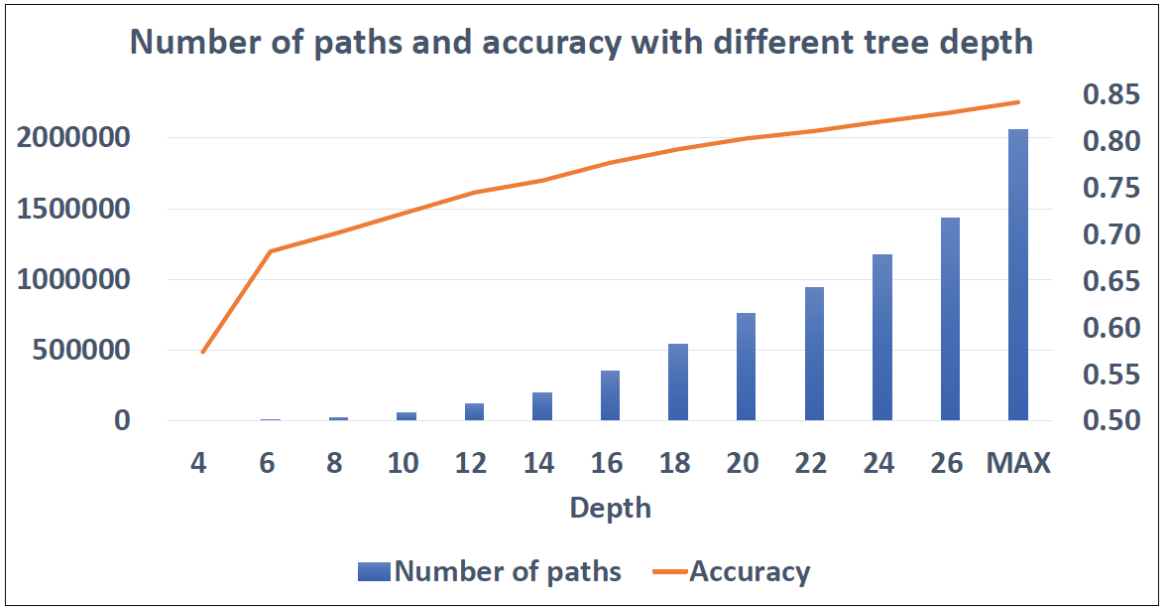
How to Search Random Forest with Crossbar Efficiently?

- Random Forest (RF):
  - An ensemble learning method to prevent overfitting
  - Training multiple decision trees with different subset
  - Inference: 1. Traverse all of decision trees
    2. Summarize (Voting or Average)



Challenges of accelerating RF:
1. Irregular tree structure
2. Non-uniform memory access patterns
3. Lots of data comparison operations

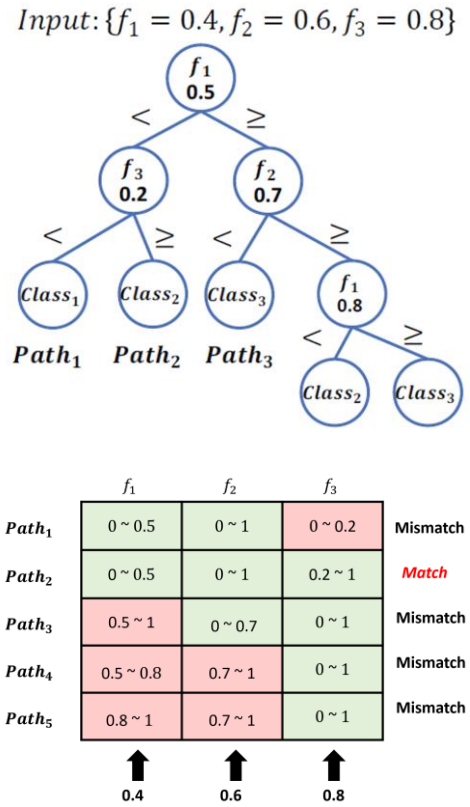
=> Existing Crossbar doesn't support ***range*** or ***non-equal*** comparison

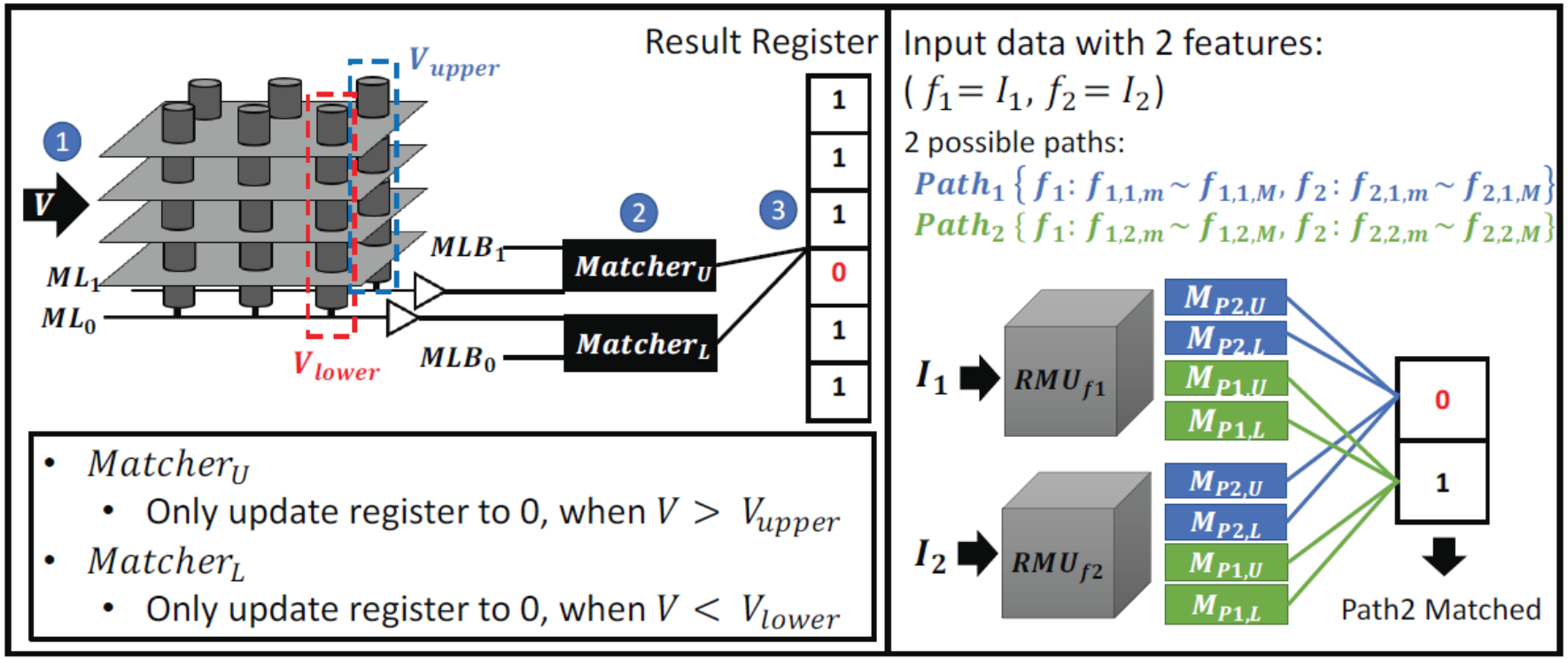


Our Method – Parallel Range Matching

This work **enables PIM supporting *range matching* in parallel**

- Support parallel range matching operation for inference of random forest
- Use **two types of matchers for upper and lower bound**, respectively
- **Connect results of the same path to a bit** of the result register

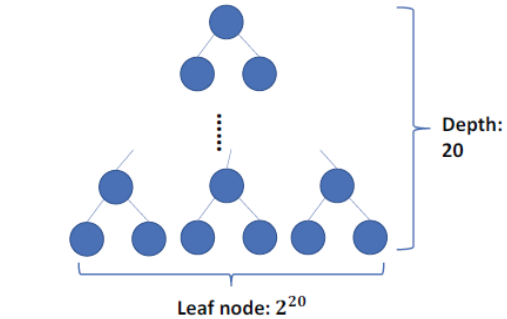




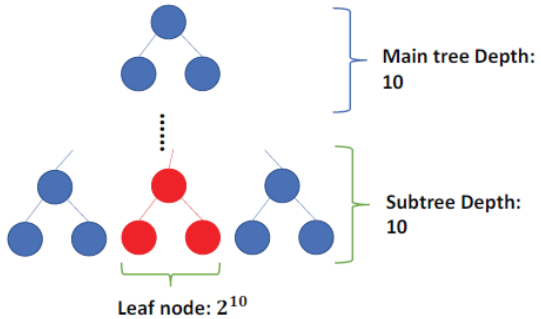
Our Method - Subtree Partitioning

- To reduce the computing complexity, we proposed to **partition the tree into multiple smaller subtrees**
  - Each subtree has the same number of paths as the number of rows of 3D TCAM
  - The result of upper subtree can be used to decide which subtree (Column of 3D TCAM) to be checked next

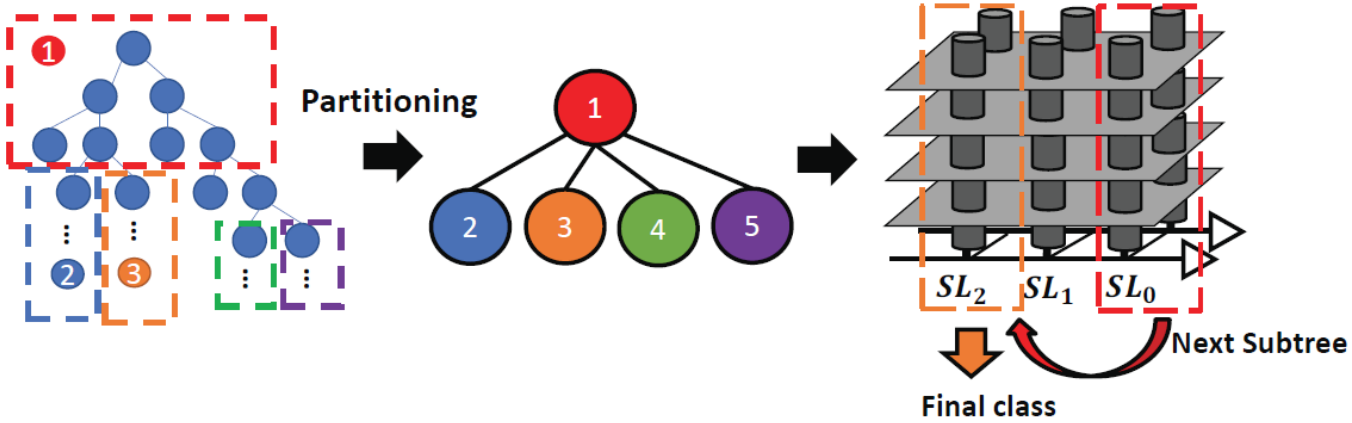
=> Eliminate lots of unmatched paths



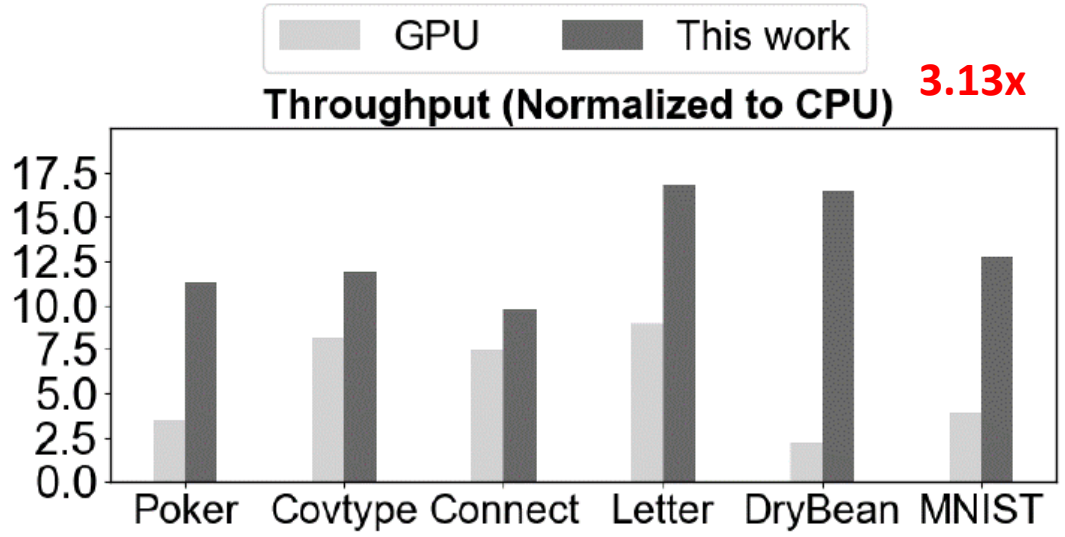
# of tree X Path(Leaf node) X # of feature
100 X 2^{20} X F

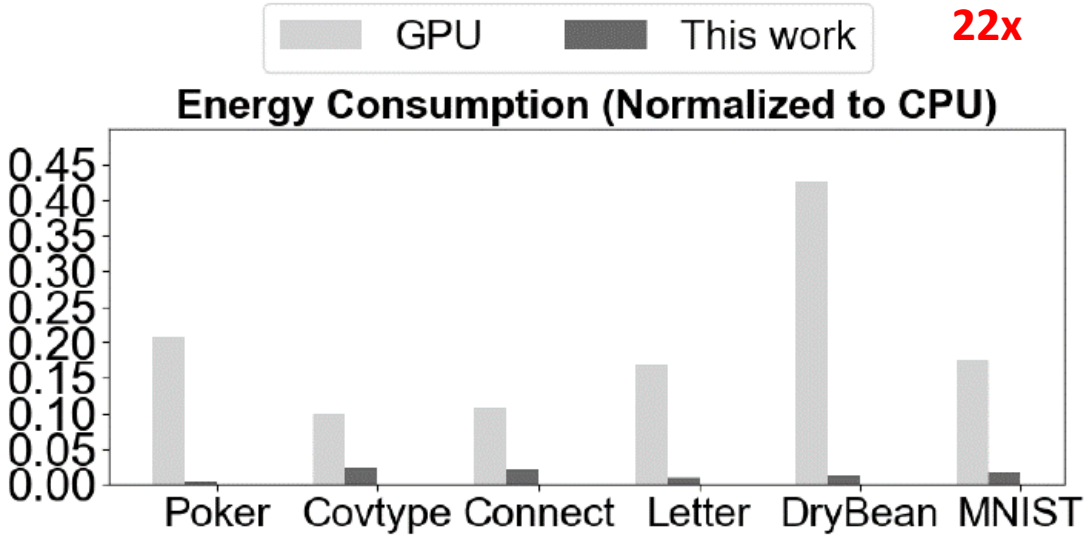


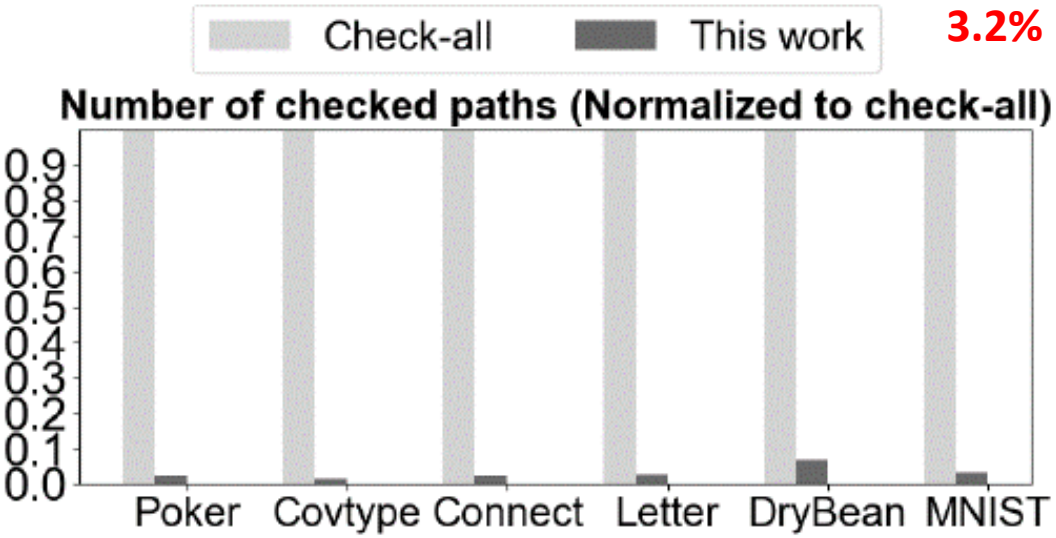
Main tree (Depth:10) + 1024 Subtrees (Depth:10)
# of tree X Leaf node X # of feature
Main tree: 100 X 2^{10} X F
Targeted tree: 100 X 2^{10} X F

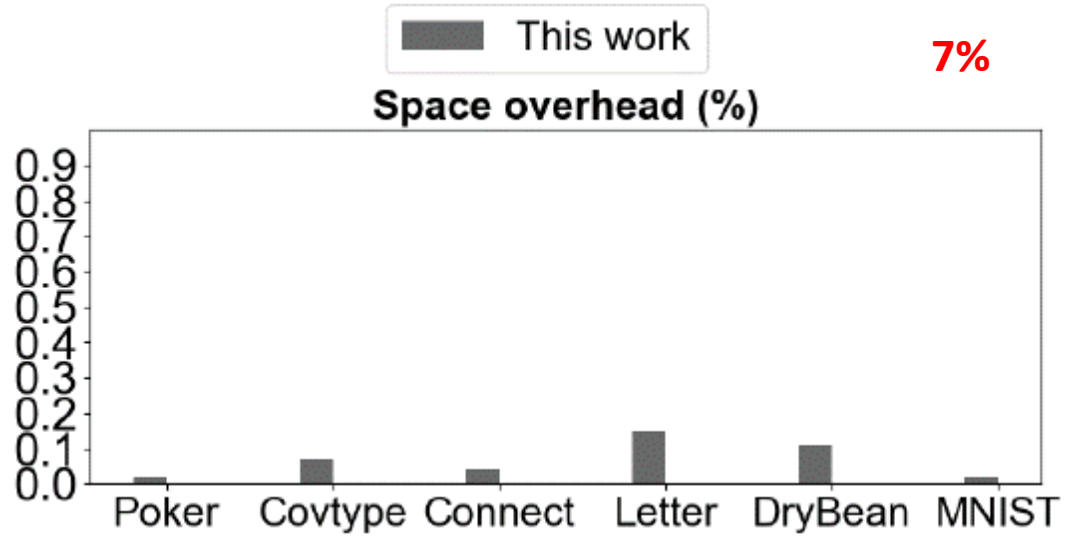


Evaluation


GPU
This work
Throughput (Normalized to CPU)
3.13x
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Poker
Covtype
Connect
Letter
DryBean
MNIST


GPU
This work
22x
Energy Consumption (Normalized to CPU)
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Poker
Covtype
Connect
Letter
DryBean
MNIST


Check-all
This work
3.2%
Number of checked paths (Normalized to check-all)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Poker
Covtype
Connect
Letter
DryBean
MNIST

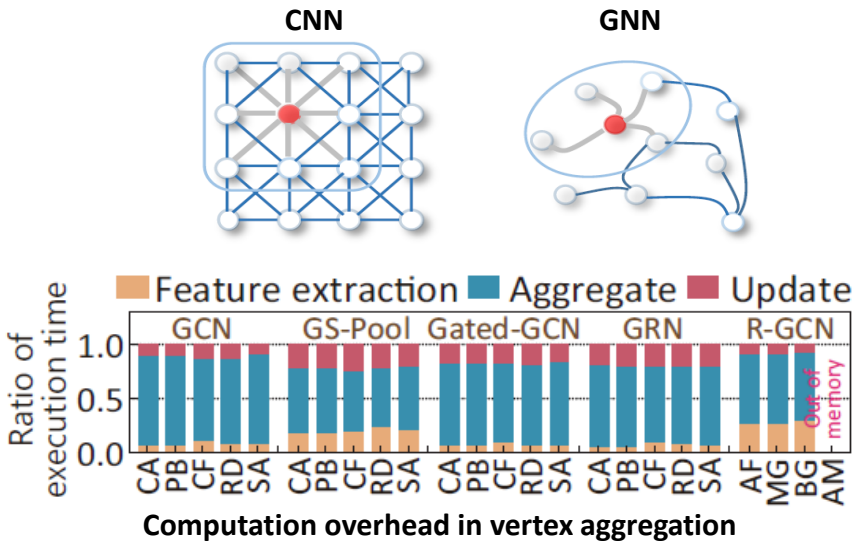

This work
7%
Space overhead (%)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Poker
Covtype
Connect
Letter
DryBean
MNIST

TCAM-GNN: A TCAM-based Data Processing Strategy for GNN over Sparse Graphs [IEEE TETC'24]

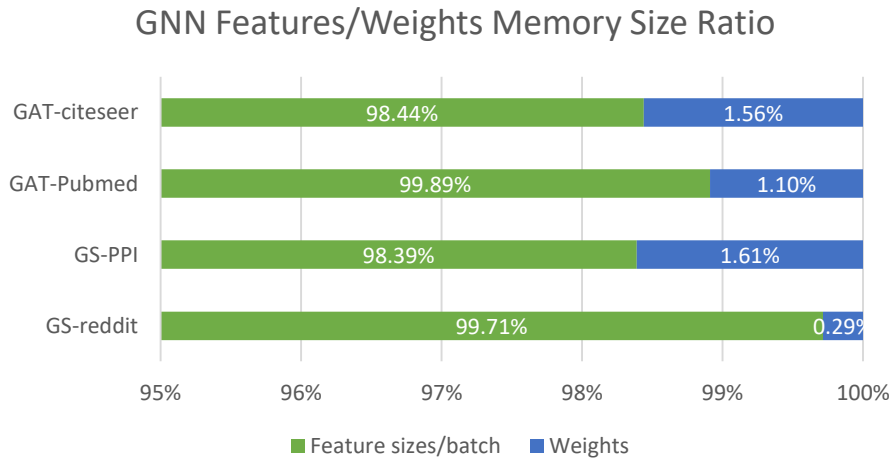
TCAM-GNN: A TCAM-based Data Processing Strategy for GNN over Sparse Graphs

[IEEE TETC'24]

- **Observation**:
  - Smart Utilizing ternary content addressable memory (TCAM) crossbars to enable intensive neighbor vertices sampling operation and efficiently support parallel data processing strategy in training phase of GNN.
- **Challenges:**
  - Large-scaled Graph: Huge memory overhead + Intensive data movement
  - Sparse Graph: Power law distribution + Poor locality + Irregular / imbalance memory access
  - Poor parallelization in processing over GPUs: Workload imbalance
- **Goal:** Enhance training/inferencing performance of graph neural network with ReRAM-based PIM accelerator.



Computation overhead in vertex aggregation



- Yu-Pang Wang, Wei-Chen Wang, Yuan-Hao Chang, Chieh-Lin Tsai, Tei-Wei Kuo, Chun-Feng Wu, Chien-Chung Ho, and Han-Wen Hu, "TCAM-GNN: A TCAM-based Data Processing Strategy for GNN over Sparse Graphs," IEEE Transactions on Emerging Topics in Computing (TETC), vol. 12, no. 3, pp. 891-904, Jul. 2024.

TCAM-GNN: A TCAM-based Data Processing Strategy for GNN over Sparse Graphs

[IEEE TETC'24]

- **For calculating $k+1_{th}$ embeddings of node v:**

1. Feature extraction

- Find neighbors of v => u_1, u_3, u_4
- Utilize the TCAM to perform **In-Memory-Searching**

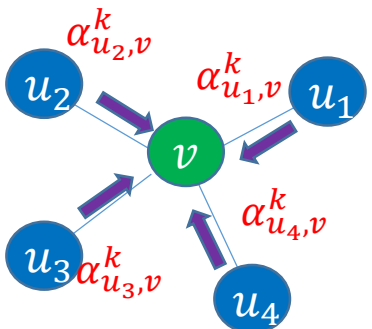
2. Aggregation

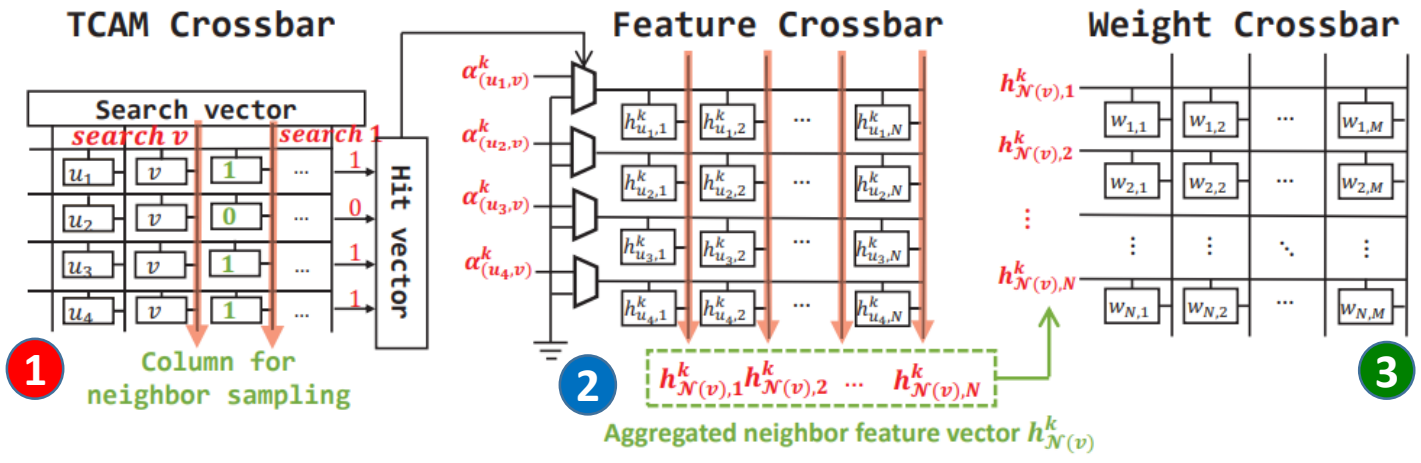
- Calculate the dot product of attention weight matrix and the feature embedding matrix of previous layer
- Adopt the ReRAM crossbar array to perform the MAC (Multiply-Accumulate)

3. Combine & Update

- Result: $(k+1)_{th}$ embeddings of node v
- Append the $(k+1)_{th}$ embeddings to the Feature Crossbar for next layer

$$h_v^{k+1} = W_{com}^k\left(\sum_{u\in\mathcal{N}(v)} \alpha_{u,v}^k\, h_u^k\right)$$

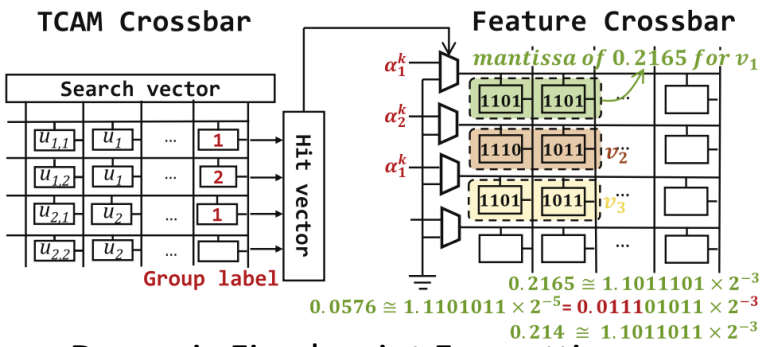




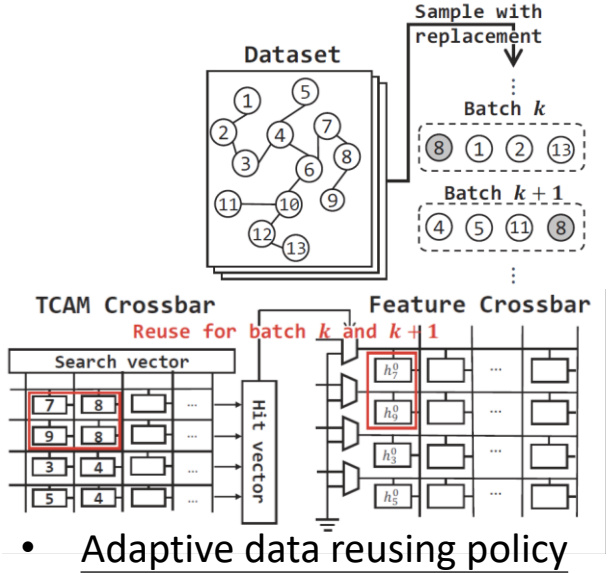
TCAM-GNN: A TCAM-based Data Processing Strategy for GNN over Sparse Graphs

[IEEE TETC'24]

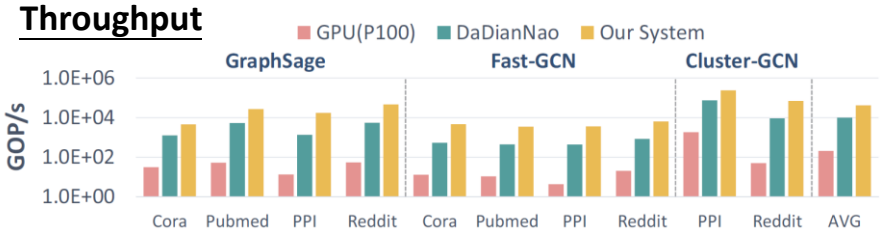
- **Contributions**:
  1. A high-throughput and energy-efficient ReRAM-based PIM accelerator with auxiliary TCAM crossbars is designed for training various graph neural networks over large-scale graphs
  2. We propose a novel dynamic fixed-point formatting approach to enable crossbars to handle GNN operations more efficiently
  3. An adaptive data reusing policy is designed to enhance the data locality of graph features
  4. The experimental results show that TCAM-GNN could improve performance by 4.25× and energy efficiency by 9.11× on average compared to the neural network accelerators

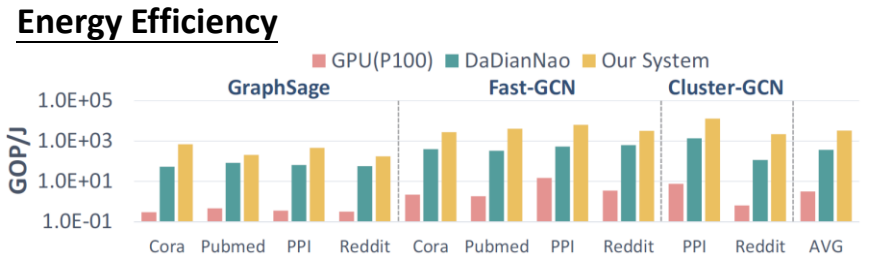


- Dynamic Fixed-point Formatting



- Adaptive data reusing policy

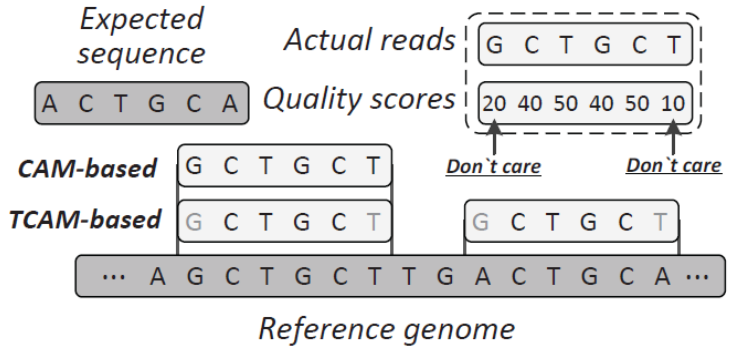


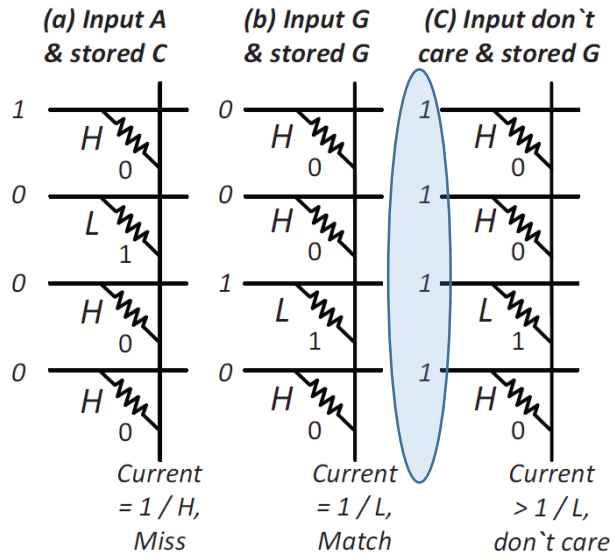


Enabling Highly-Efficient DNA Sequence Mapping via ReRAM-based TCAM

[ISLPED'23]

- **Observation**
  - In the post-pandemic era, third-generation DNA sequencing (TGS) has received increasing attention. However, much less effort has been devoted to DNA sequence mapping acceleration while considering both the memory wall issue and the challenges of TGS technologies.
- **Goal**
  - Propose a novel resistive random-access memory (ReRAM)-based ternary content-addressable memory (TCAM)
  - Exploit the intrinsic parallelity of ReRAM crossbar for mapping acceleration.
- **Main Idea**
  - Exploit the don't care feature of TCAM to mark nucleotides based on quality scores provided by TGS technologies.
  - Implement the functionality of TCAM within ReRAM crossbar circuitry without including new transistors and resistors





Reduce 99.72% energy and 99.76% latency, compared to the conventional CPU-based Minimap tool

- Yu-Shao Lai, Shuo-Han Chen, and Yuan-Hao Chang, "Enabling Highly-Efficient DNA Sequence Mapping via ReRAM-based TCAM," ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), Vienna, Austria, Aug. 7-8, 2023. **(Top Conference)**

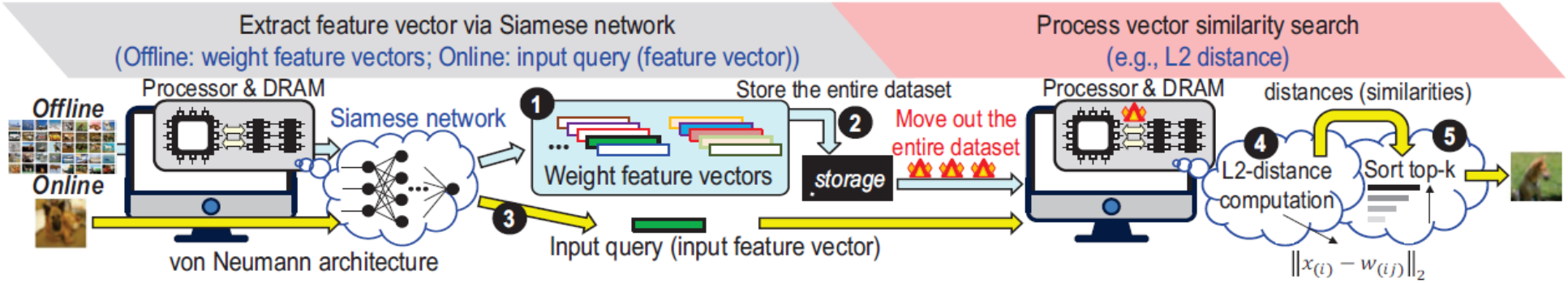
Outline

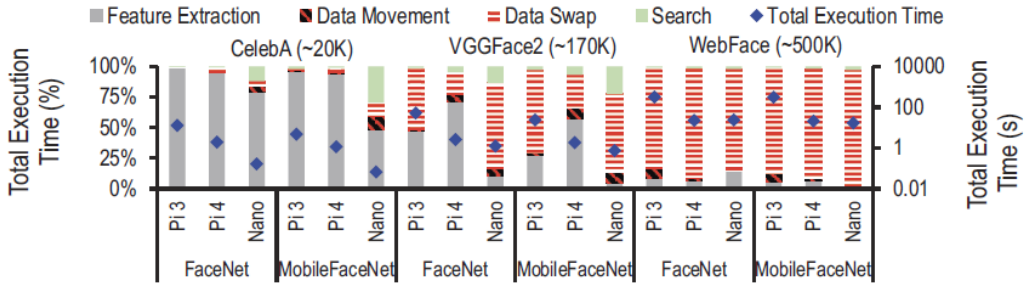
- Introduction
- ReRAM-based In-Memory Computing
- **Flash-based In-Memory Computing**
- Conclusion

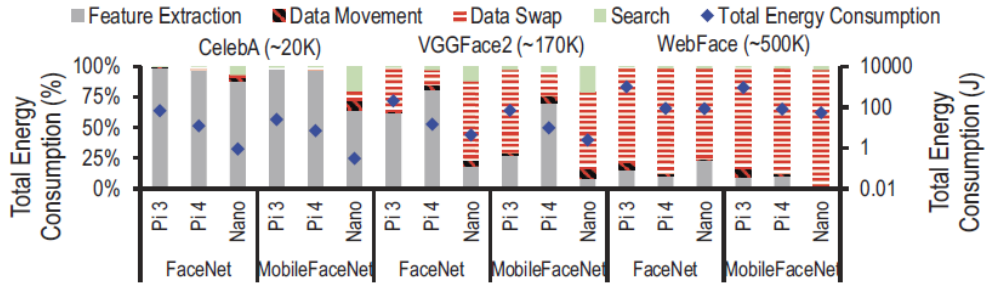
ICE: Intelligent Cognition Engine with NAND In-Memory Computing for Vector Similarity Search [MICRO'22]

ICE: Intelligent Cognition Engine with NAND In-Memory Computing for Vector Similarity Search [MICRO'22]

- **Motivation**
  - Existing vector similarity search (VSS) on edge devices is inefficient
    - Long search latency and large search energy due to **large unnecessary data movement**
  - Exploiting 3D NAND with nonvolatile IMC (nvIMC) for VSS will face two major challenges:
    - Digital-based solution: ECC is critical to the nvIMC design for VSS app., since it guarantees data reliability
    - Analog-based solution: Numerous ADCs and DACs increases the chip size



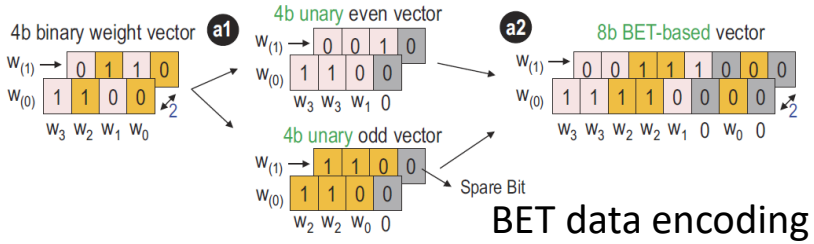




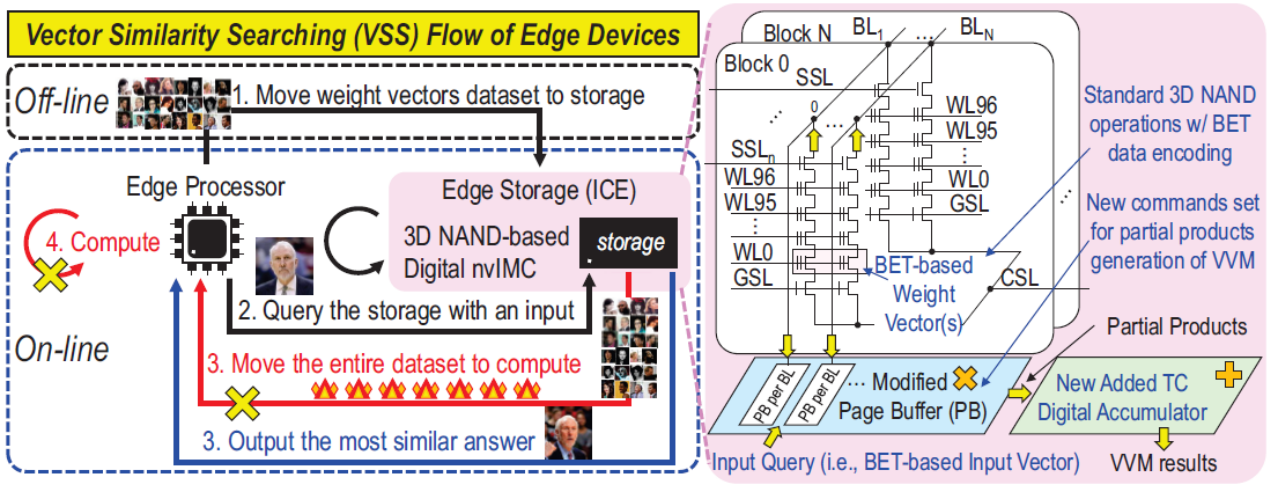
- Traditional face search on various edge devices: The searching time with large data will dominate the total execution time on edge devices.

ICE: Intelligent Cognition Engine with NAND In-Memory Computing for Vector Similarity Search [MICRO'22]

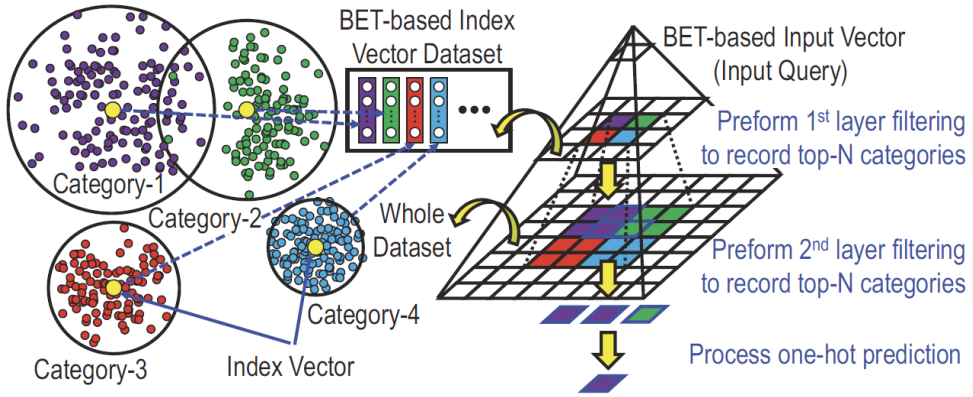
- **Main Idea**
  - We enable digital flash-based IMC accelerator (**ICE**) supporting VSS **on existing flash cards (e.g., eMMC)**
  - A bit-error-tolerance (BET) data encoding to mitigate the reliability issue
  - Enable digital IMC to mitigate the bit-error influence
  - Propose a hierarchical top-n search to filter out unneeded data
  - Remove ADC/DAC to resolve the energy issue



BET data encoding



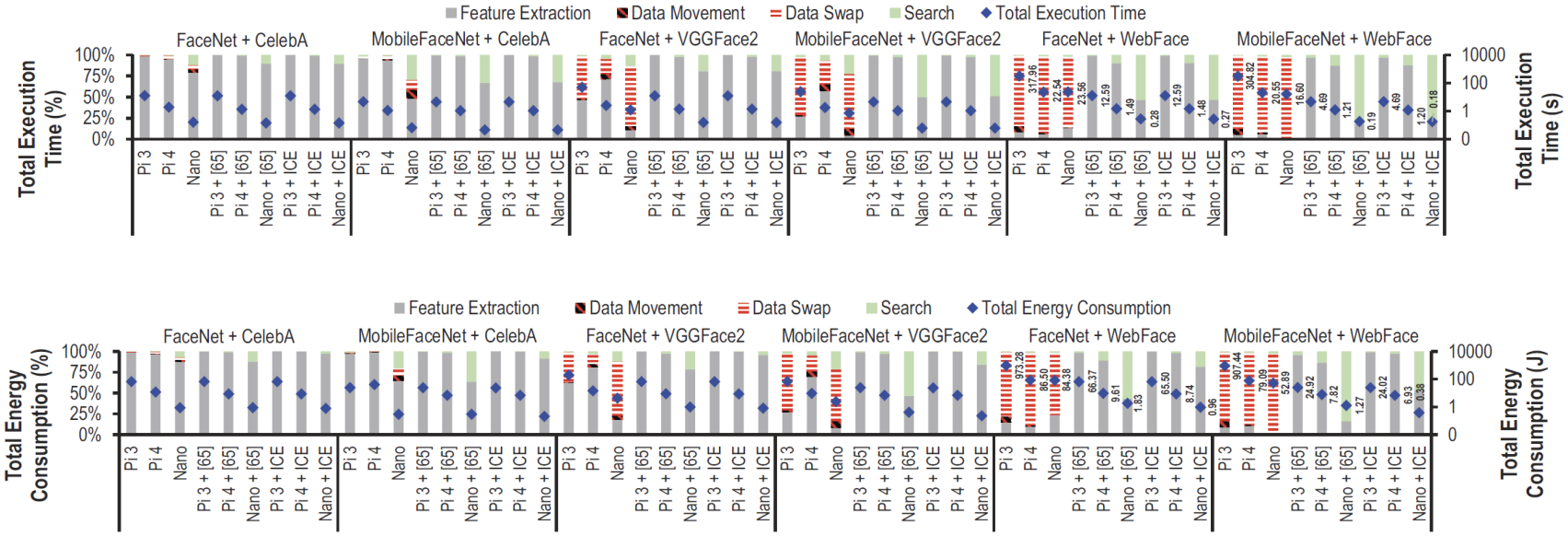
- Edge device and ICE cooperation for VSS acceleration



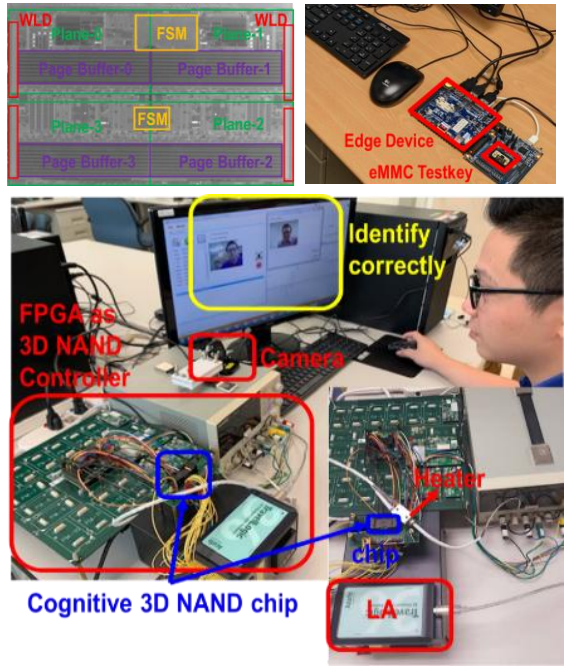
- Proposed hierarchical top-n search method

ICE: Intelligent Cognition Engine with NAND In-Memory Computing for Vector Similarity Search [MICRO'22]

- **Results**
  - Conduct a series of experimental measurements and simulations to evaluate the feasibility and capability of ICE on real 3D NAND chips
  - ICE only requires 1.6% area overhead in commercial 3D NAND chips
  - Compared to the state-of-the-art edge systems, ICE improves execution time by 17× to 95× and energy efficiency by 11× to 140×



- Execution time & Energy consumption analysis

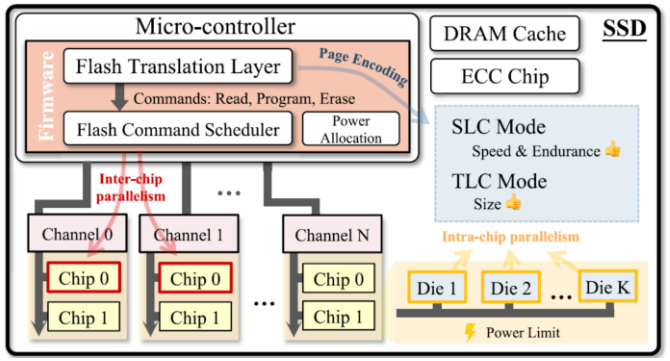


Search-in-Memory (SiM): Conducting Data-Bound Computations on Flash Memory Chip

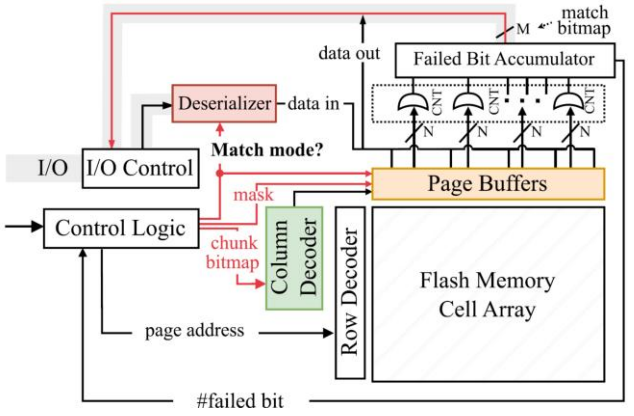
[IEEE TCAD'24]

[CODES'24 – Best Paper Award]

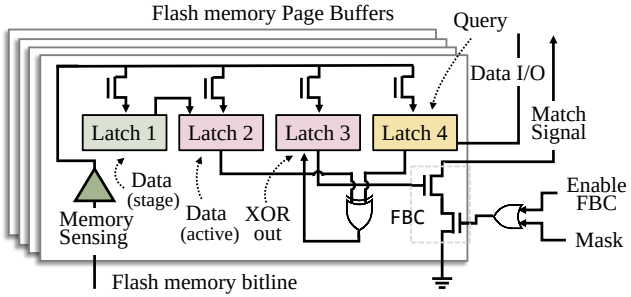
- **Motivation**
  - Data indexing is I/O bound
  - Existing computational memory solutions require intrusive design changes

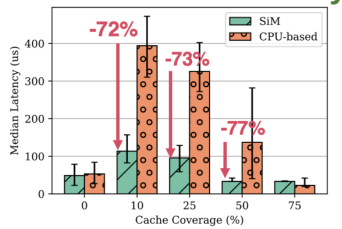


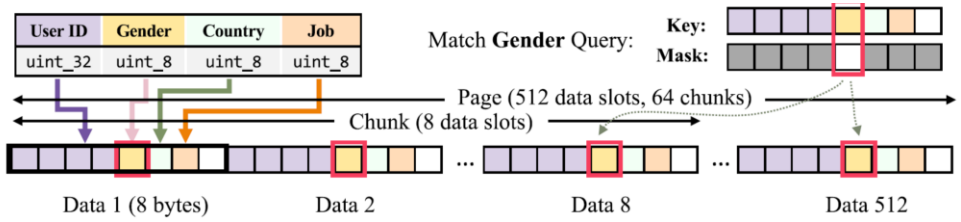
- **Main Idea**
  - Realize data matching through re-purposing existing circuits
  - Saving I/O by sending query into memory instead of reading page out of memory
  - Generic SIMD command interface useful for wide range of applications

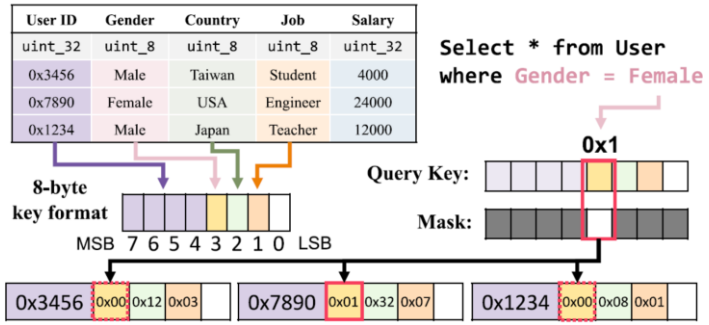


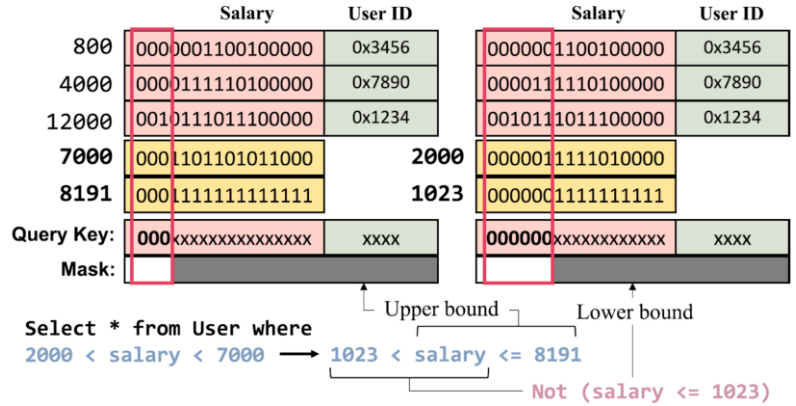
- **Results**
  - Extensive testing across a wide range of workloads reveals up to a 9× speedup in write-heavy workloads and up to 45% energy savings due to reduced read and write I/O.
  - Tail read latencies are reduced by up to 85%.







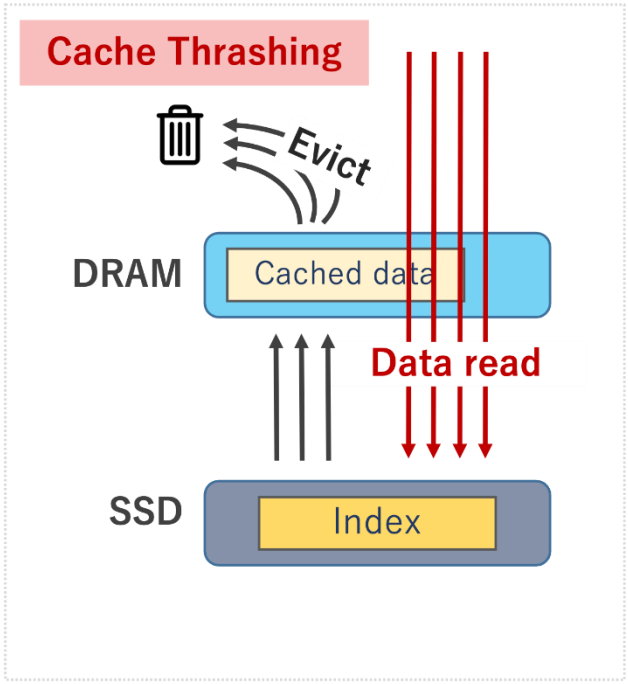




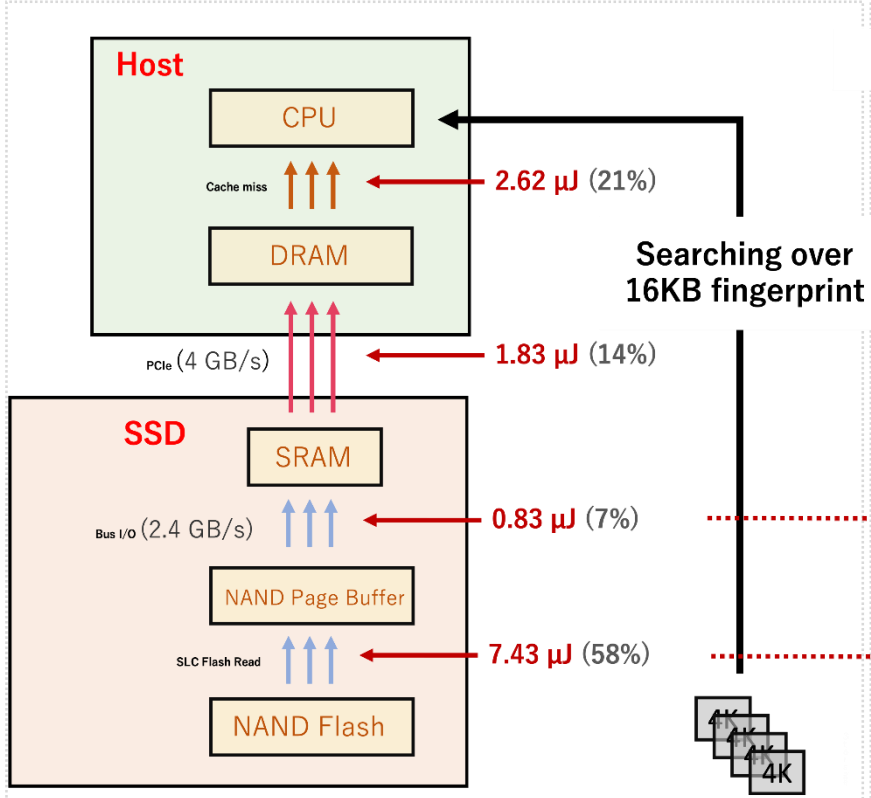
Search-in-Memory (SiM): Conducting Data-Bound Computations on Flash Memory Chip

[IEEE TCAD'24]
[CODES'24 – Best Paper Award]

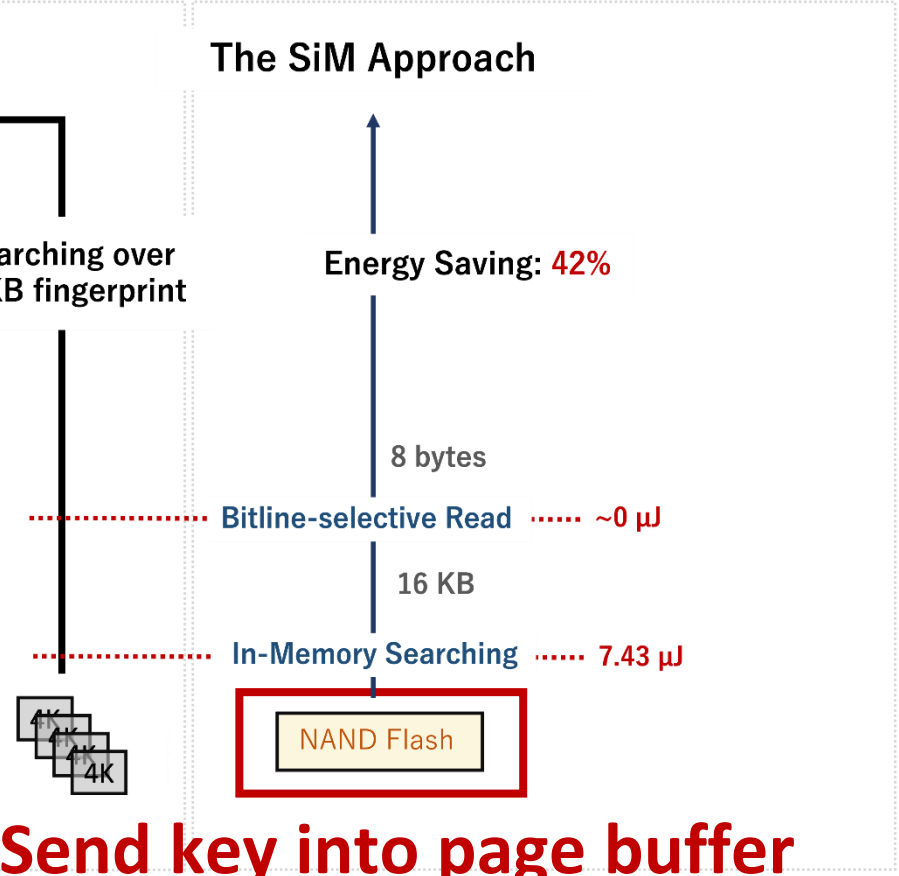
I/O Bottleneck in Database Index Querying



Heavy data transfer: High latency & energy consumption



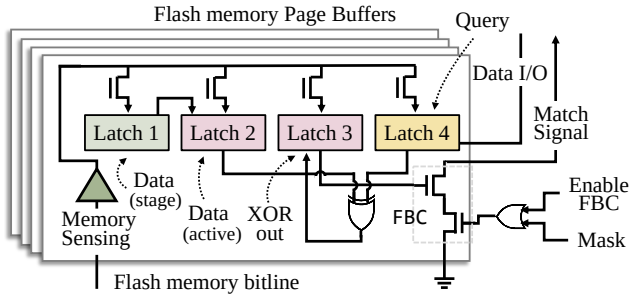
Our Approach

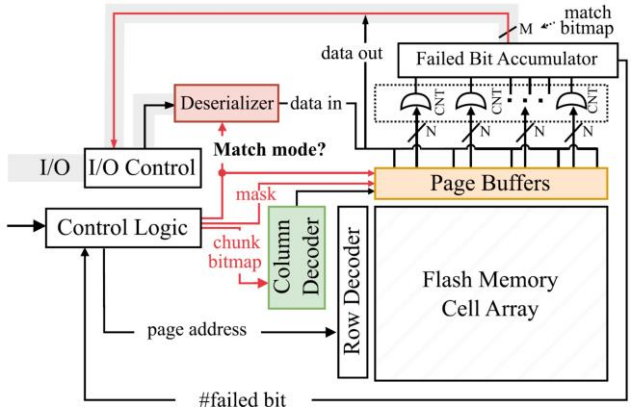


Send key into page buffer
In-situ data matching

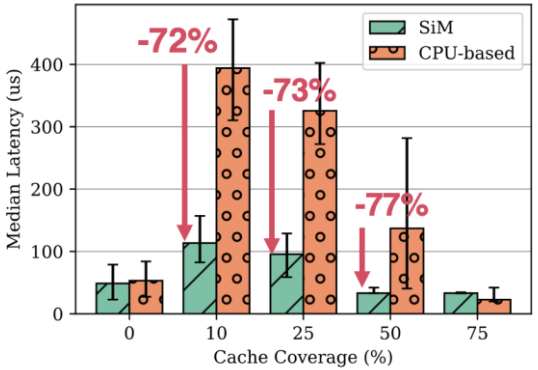
Search-in-Memory (SiM): Conducting Data-Bound Computations on Flash Memory Chip

- **Main Idea**
  - Realize data matching through re-purposing existing circuits
  - Saving I/O by sending query into memory instead of reading page out of memory
  - Generic SIMD command interface useful for wide range of applications
- **Results**
  - Extensive testing across a wide range of workloads reveals up to a 9X speedup in write-heavy workloads and up to 45% energy savings due to reduced read and write I/O.
  - Tail read latencies are reduced by up to 77%.

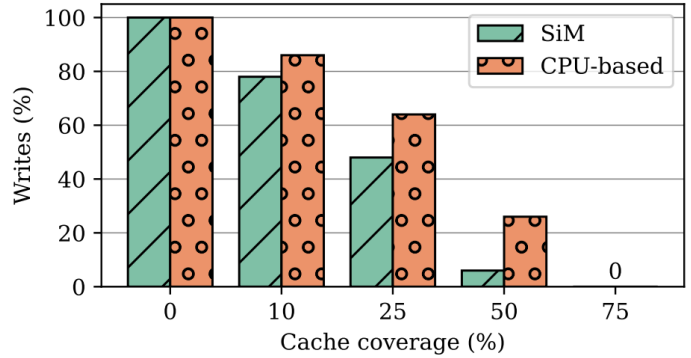




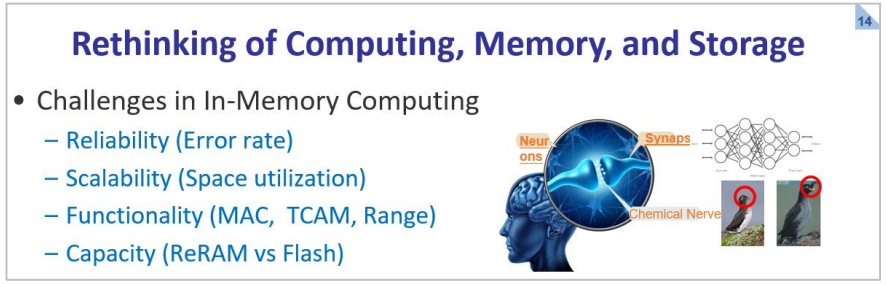
Read delay



Write Reduction



Conclusion



- To unleash the potential of ReRAM-based in-memory computing
  - Scalability/Reliability: Adaptive data manipulation to resolve the analog variation error
  - Space Utilization/Efficiency: Graph-aware data placement for the sparsity issue on graph processing
  - Functionality: Enable digital range comparison with Re-RAM crossbar
- To unleash the potential of Flash-based in-memory computing
  - Capacity:
    - Enable digital-based in-memory computing and huge parallelism with flash
    - Enable Search-in-Memory with flash to enhance the PIM capacity and parallelism degree.

- In the future, we will
  - Extend our study to more key AI functions and LLM-based applications, and
  - Develop enabling technologies for flash-based AI accelerators.

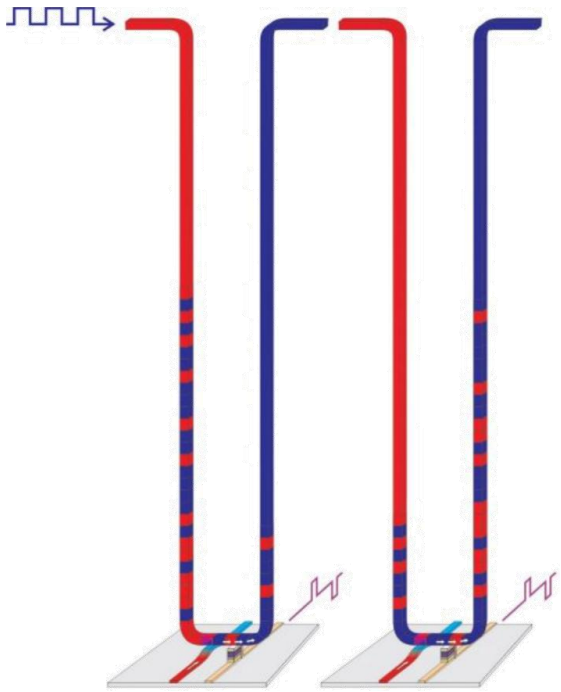
國立臺灣大學
National Taiwan University



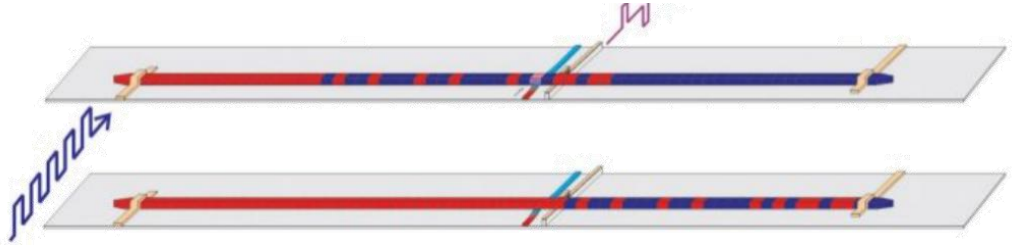
Racetrack Memory

Two Types of Racetrack Memories

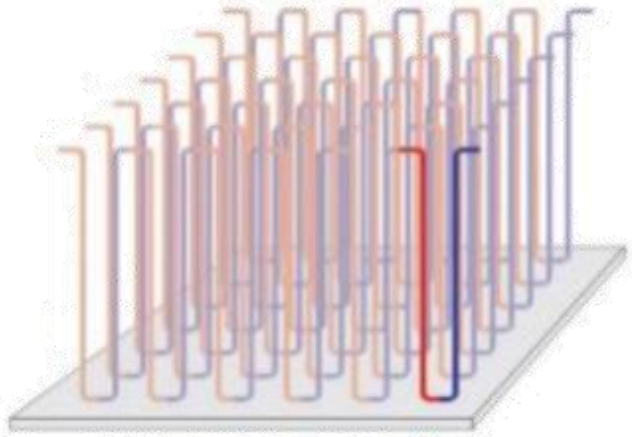
- **Domain-wall racetrack memory (DW-RM)**
- **Skyrmion racetrack memory (SK-RM)**



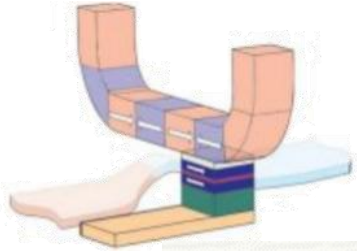
Vertical racetrack



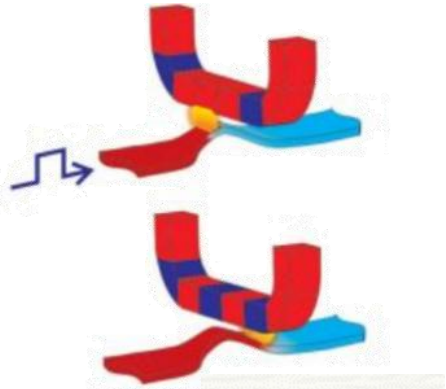
Horizontal racetrack



Racetrack storage array



Reading



Writing

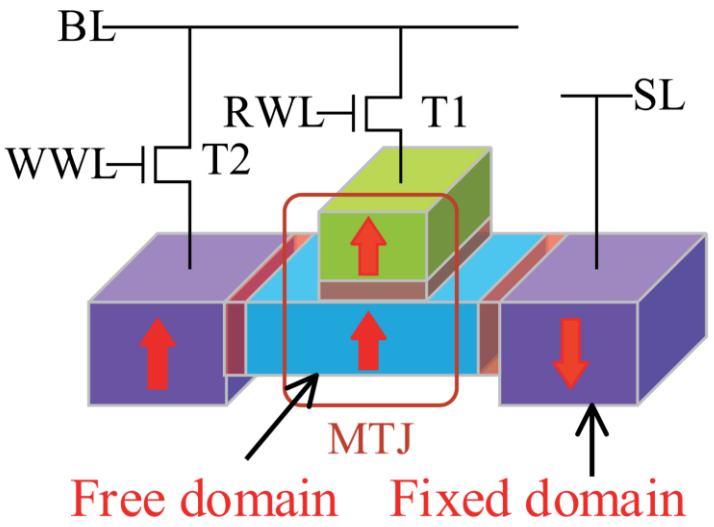
https://www.science.org/doi/pdf/10.1126/science.1145799

Domain Wall Memory (DWM)

- **DWM characteristics**
  - Nonvolatility
  - **Very high density (Macro-cell: ~4× than STT-MRAM)**
  - Ultra-low standby/leakage power
  - Efficient read/write energy consumption
  - **Comparable latencies to SRAM**
- **DWM cells**
  - **Micro-cell DWM**
  - **Macro-cell DWM**

https://dl.acm.org/doi/abs/10.1145/2744769.2744800

Micro-cell vs Macro-cell DWMs



Micro-cell DWM:

LOW LATENCY but

LOW DENSITY (cell size: $40F^2$)

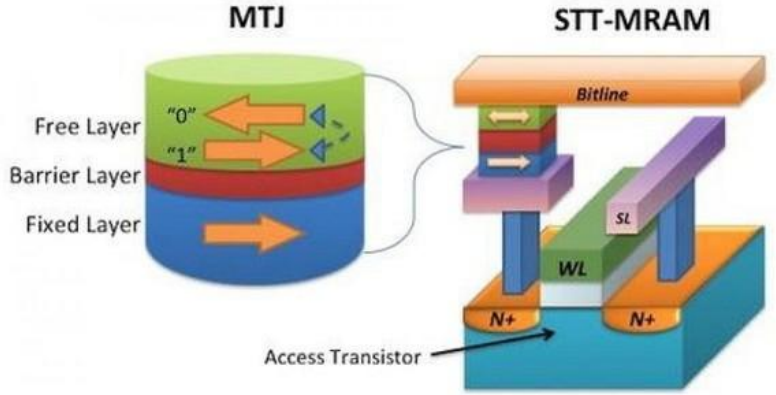
MTJ: Magnetic Tunnel Junction (MRAM)

SWL: Source Word Line

- The word line controls the access transistor that connects SL to the MTJ.

SL: Source Line

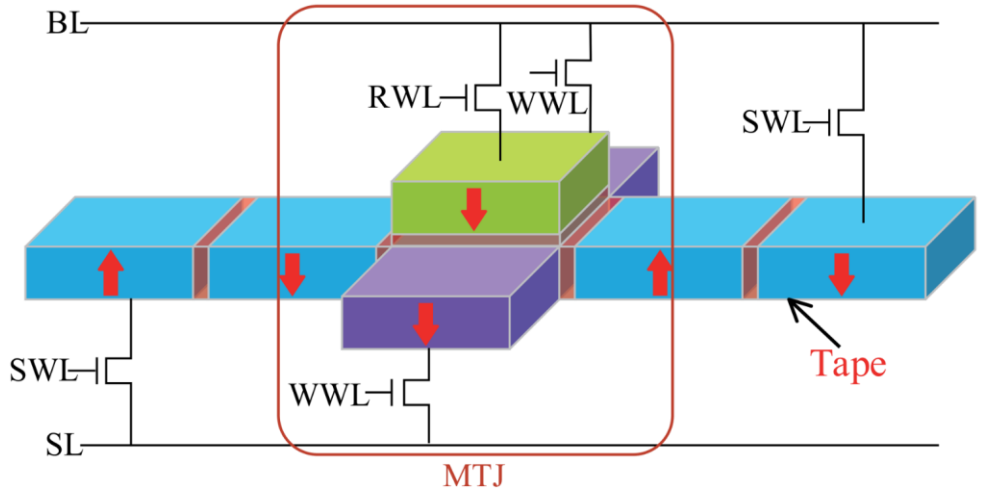
- Provides the current return path for MTJ read/write operations



Macro-cell DWM a.k.a. **domain-wall racetrack memory (DW-RM)**:

UNSTABLE LATENCY but

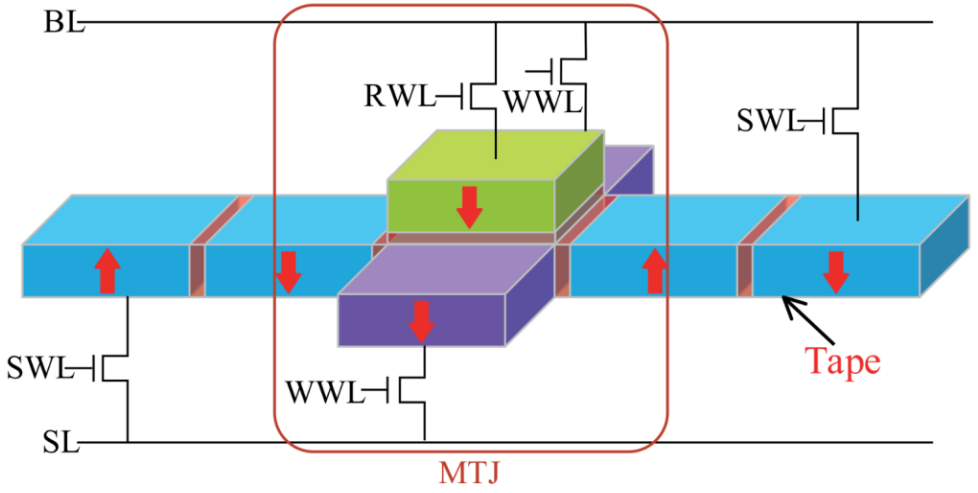
HIGH DENSITY (cell size: $4.5\ F^2$)

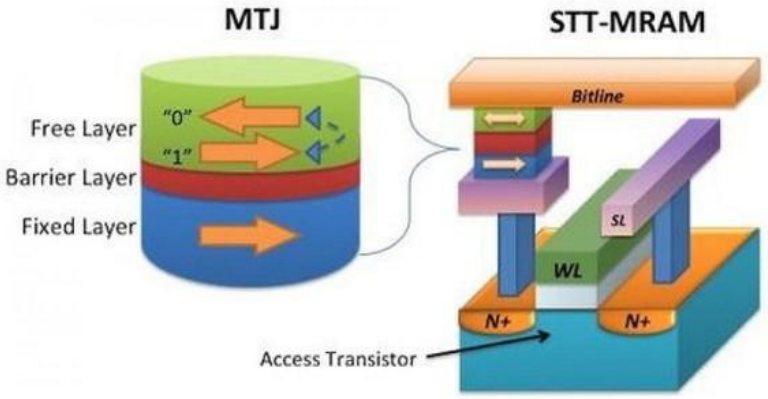


https://dl.acm.org/doi/abs/10.1145/2744769.2744800

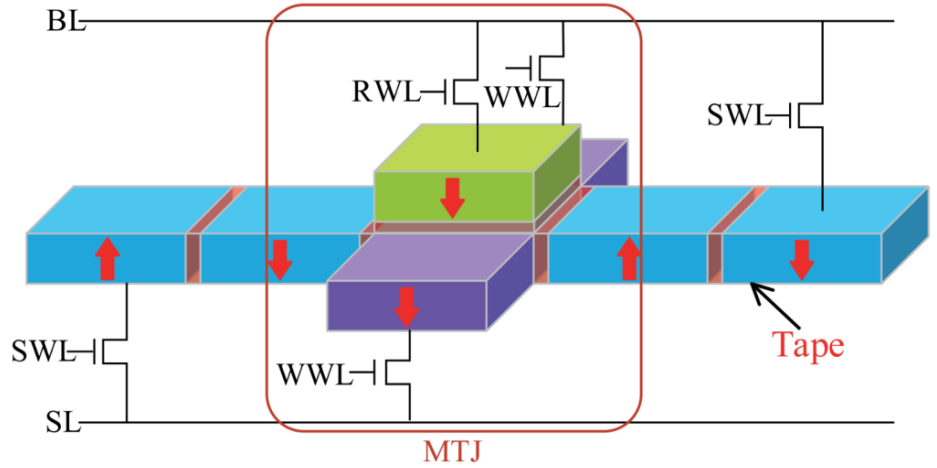
MTJ

- Each **MTJ cell** consists of:
  - **Bit Line (BL)** on the **top**,
  - **MTJ stack** in the middle,
  - **Access transistor** connecting to the **Source Line (SL)** at the **bottom**.
- When the **access transistor** (controlled by **SWL**) is turned on, BL and SL are electrically connected through the MTJ — forming the read/write path.





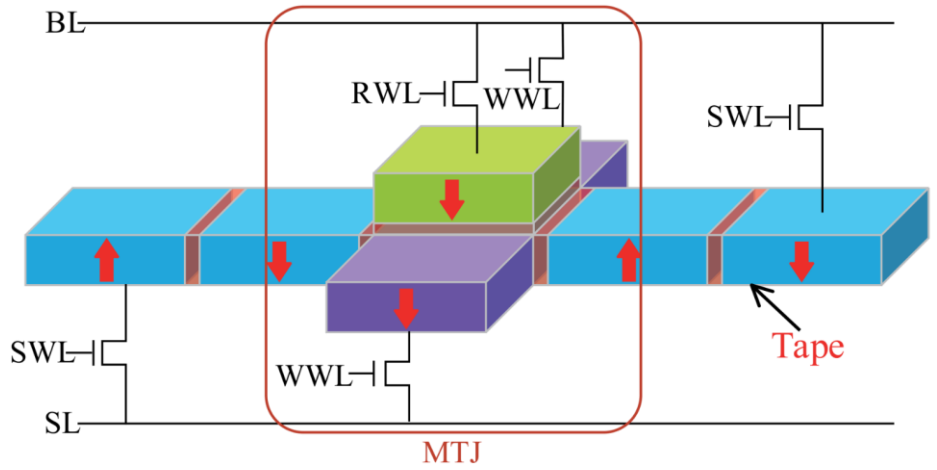
Read Operation for MTJ



1. **SWL** is **activated** → transistor **ON**.
2. A **small voltage** (read voltage) is applied between **BL** and **SL**.
   - Typically, **BL = V_read**, **SL = 0 V (ground)**.
3. A **tiny sensing current** flows through the MTJ.
   - The current is **smaller** than the write current, so it does **not flip** the free layer magnetization.
4. The **sense amplifier** detects the voltage drop or current level.
   - **Low resistance (parallel)** → higher current → logic 0
   - **High resistance (antiparallel)** → lower current → logic 1

- **Role of SL:**
  - Acts as the **ground reference** for the sense current.
  - Ensures the current path is complete and stable across the selected cell.

Write Operation for MTJ

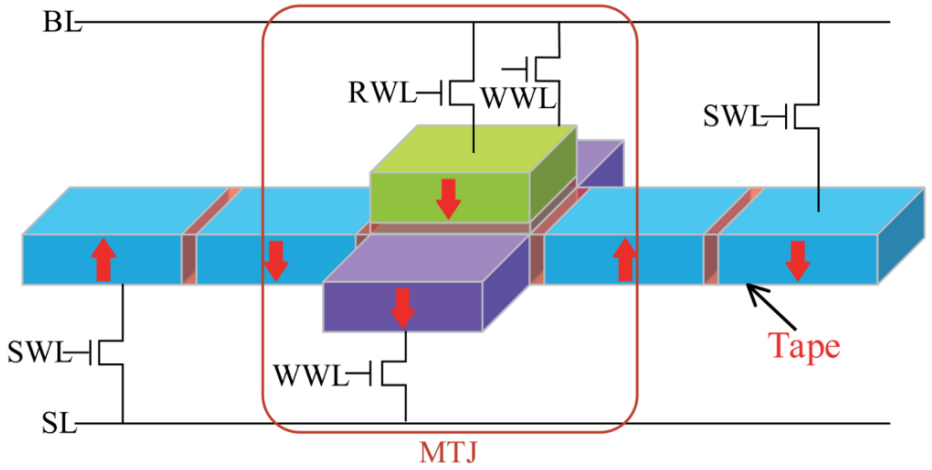


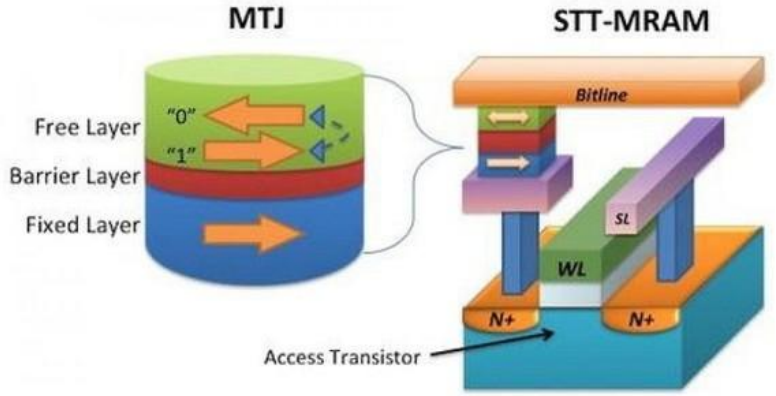
1. **SWL** is **activated** → transistor **ON**.
2. A **larger voltage** is applied between **BL** and **SL** to produce a **write current** I_write
   - Direction of the current determines the write value:
     - **BL → SL (downward current)** → flips magnetization one way (e.g., write 1)
     - **SL → BL (upward current)** → flips magnetization the opposite way (e.g., write 0)
3. **Spin-polarized electrons** transfer angular momentum to the free layer, causing its magnetization to reverse — this is the **Spin-Transfer Torque (STT)** effect.

- **Role of SL:**
  - Serves as the **current sink (or source)** depending on write direction.
  - Must be carefully biased to control current magnitude and avoid disturbing neighboring cells.

Read/Write Summary for MTJ

Operation	BL	SL	SWL	RWL	WWL	Current Path	SL Function	Description
Read	Small V_read	0 V	ON	ON	OFF	BL → MTJ → SL	Ground reference for sensing	Sense MTJ resistance (low or high)
Write "1"	High V_write	0 V	ON	OFF	ON	BL → MTJ → SL	Current sink (sets spin direction)	BL→MTJ→SL current sets magnetization
Write "0"	0 V	High V_write	ON	OFF	ON	SL → MTJ → BL	Current source (reverse spin direction)	Reverse current flips magnetization





Performance of Macro-cell DWM

- The performance of macro-cell DWM varies significantly with the data layout.
  - **Consecutive bit accesses → the best case**
  - **Random bit accesses → the worst case**

THE BEST CASE

Access latencies ≈ SRAM

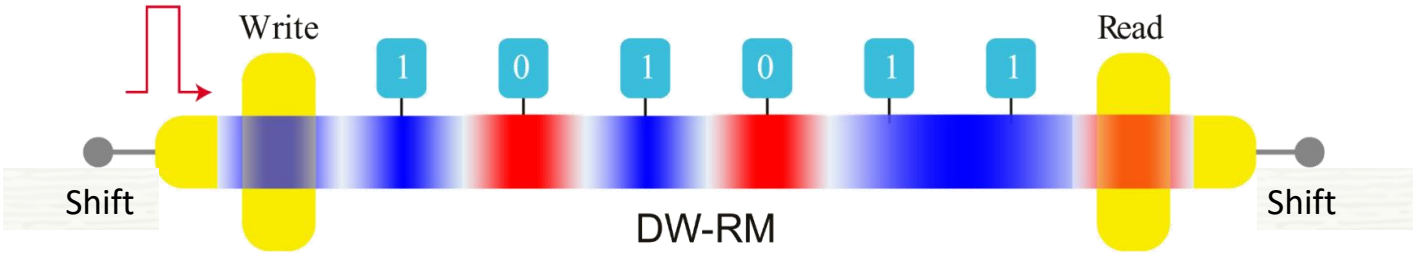
THE WORST CASE

Access latencies → Several times higher

https://dl.acm.org/doi/abs/10.1145/2744769.2744800

Domain-Wall Racetrack Memory (DW-RM)

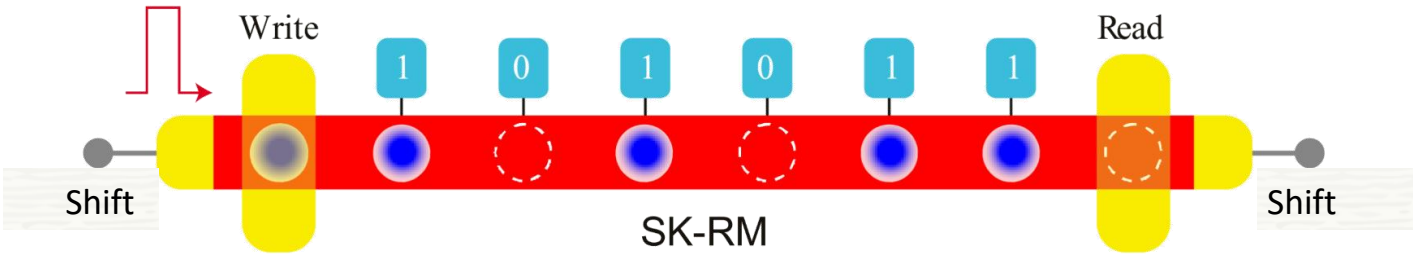
- **Operations**
  - Write
  - Read
  - **Shift**
- **Major challenges**
  - **DW size** Tradeoff **thermal stability**
  - **Critical current density for DW motion** Tradeoff **high velocity**



https://dl.acm.org/doi/abs/10.1145/3333336

Skyrmion Racetrack Memory (SK-RM)

- **Magnetic skyrmions**
  - Topological stability
  - Small size
  - High mobility
  - Extremely low depinning current

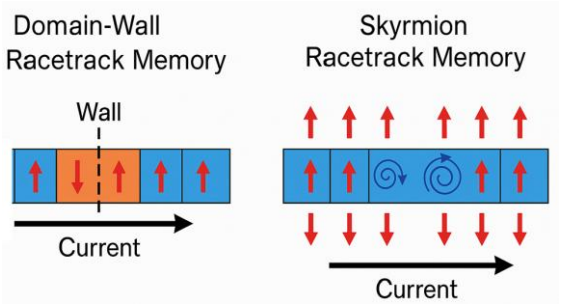


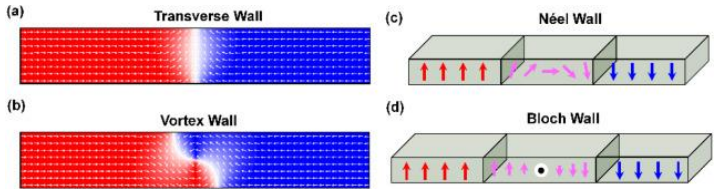
Skyrmion racetrack memory (SK-RM) is promising to replace DW-RM!

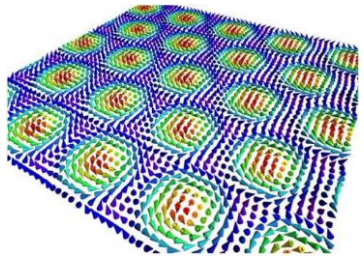
https://dl.acm.org/doi/abs/10.1145/3333336

Comparison between DW-RM and SK-RM

Feature	**Domain-Wall Racetrack Memory**	**Skyrmion Racetrack Memory**
Data element	Domain wall between ↑↓ domains	Skyrmion (localized spin vortex)
Size	Tens of nanometers	Few nanometers
Current density	High (10^{11}–10^{12} A/m²)	Very low (10^{6}–10^{8} A/m²)
Topology	*Trivial* — no topological protection	*Non-trivial* — characterized by a **skyrmion number** (±1)
Structure	Just a boundary between ↑ and ↓ domains	Continuous twist from ↑ in center to ↓ at edge (like a magnetic knot)
Topological stability	✗ None	☑ Yes — protected by topology
Energy efficiency	Moderate	Excellent
Operation speed	High	Very high
Maturity	More mature, experimentally demonstrated	Emerging, still under active research

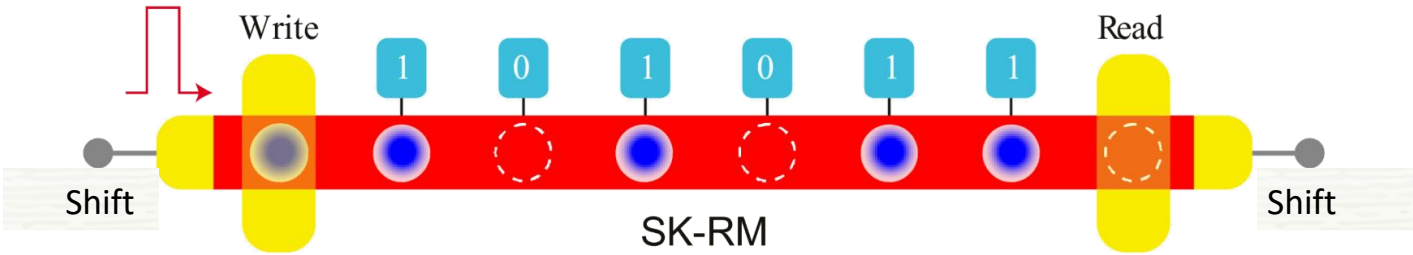






Operations of SK-RM

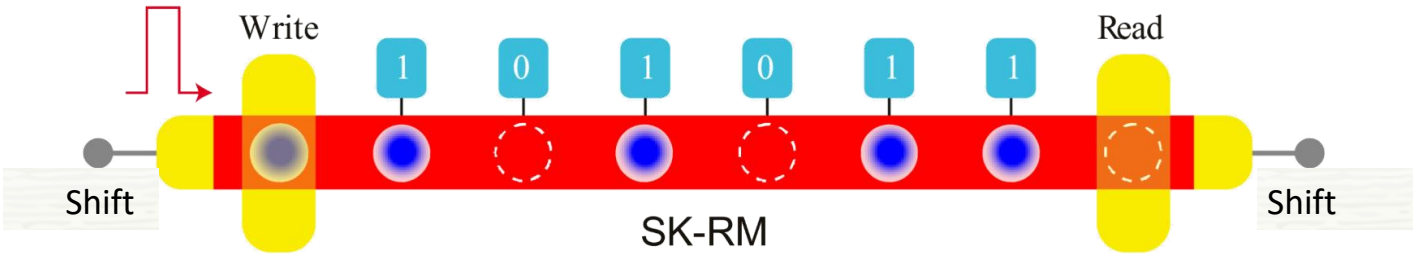
- **Basic operations**
  - Write
  - Read
  - Shift
- **Special operations**
  - Update
  - Insert
  - Delete

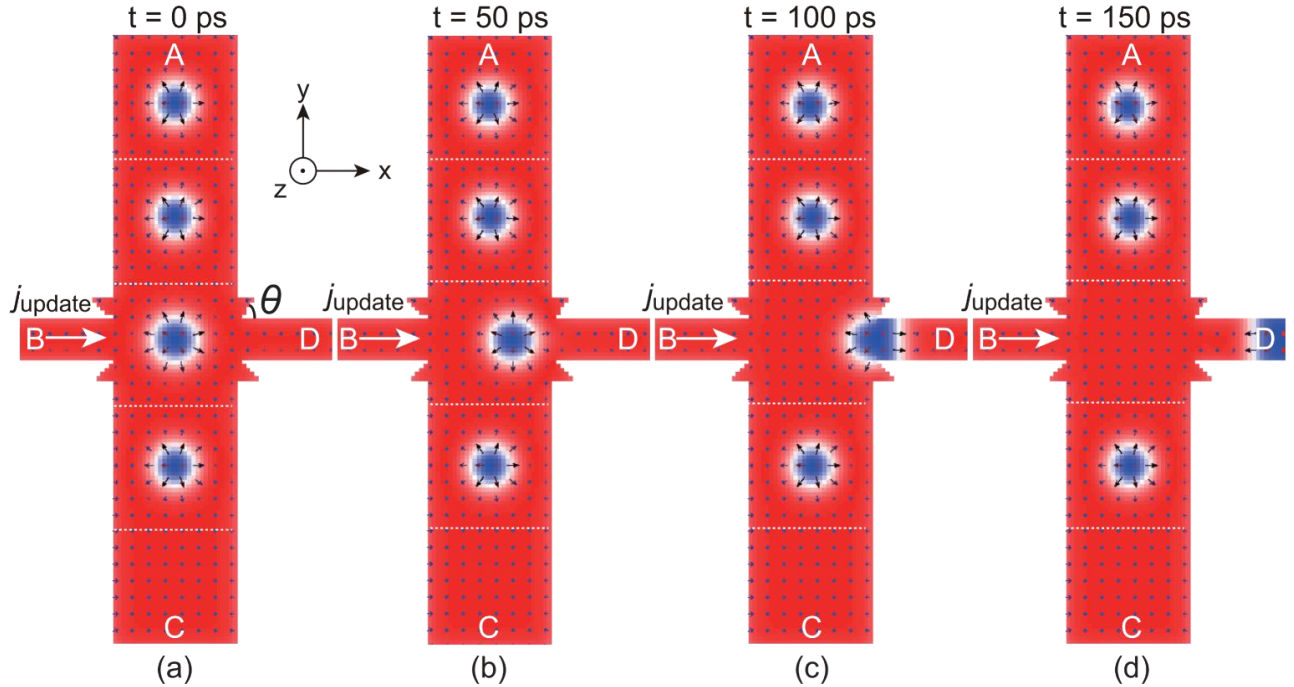


https://dl.acm.org/doi/abs/10.1145/3333336

Operations of SK-RM (Cont.)

- **Basic operations**
  - Write
  - Read
  - Shift
- **Special operations**
  - **Update**
  - Insert
  - Delete

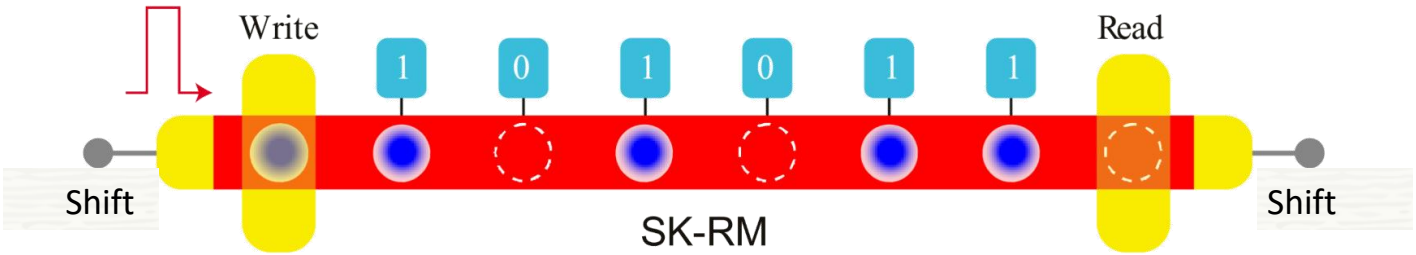


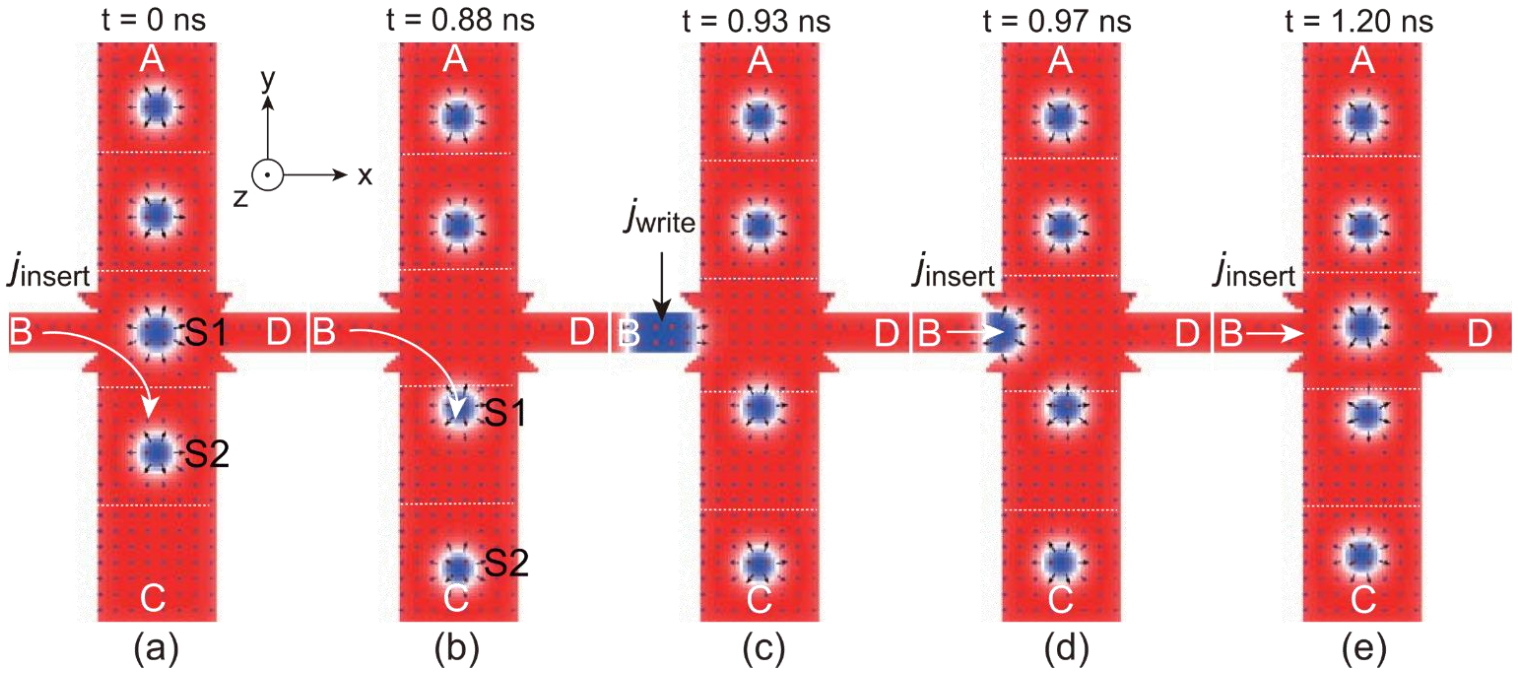


https://dl.acm.org/doi/abs/10.1145/3333336; https://ieeexplore.ieee.org/abstract/document/8107669

Operations of SK-RM (Cont.)

- **Basic operations**
  - Write
  - Read
  - Shift
- **Special operations**
  - Update
  - **Insert**
  - Delete

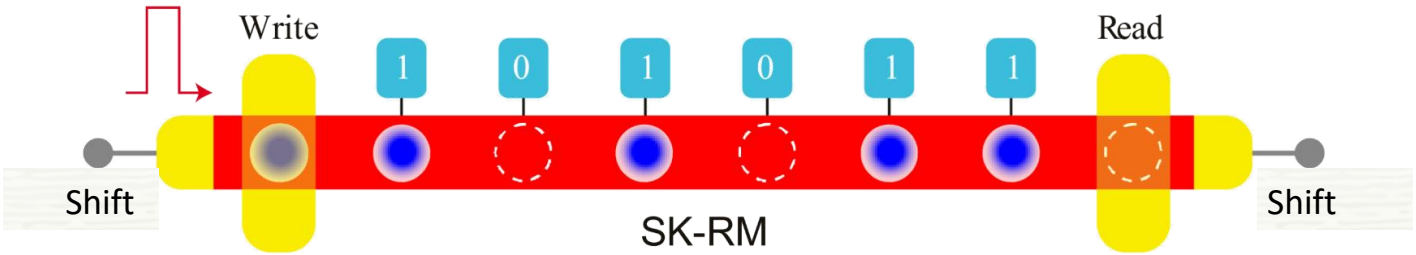


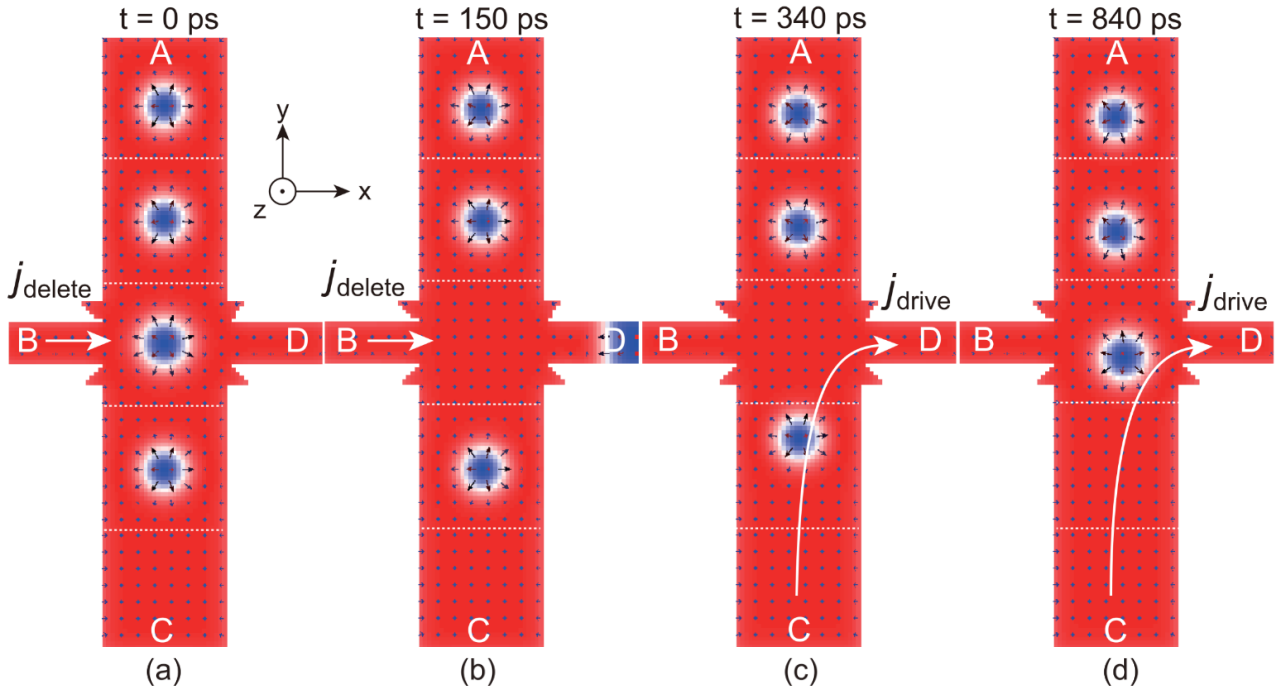


https://dl.acm.org/doi/abs/10.1145/3333336; https://ieeexplore.ieee.org/abstract/document/8107669

Operations of SK-RM (Cont.)

- **Basic operations**
  - Write
  - Read
  - Shift
- **Special operations**
  - Update
  - Insert
  - **Delete**

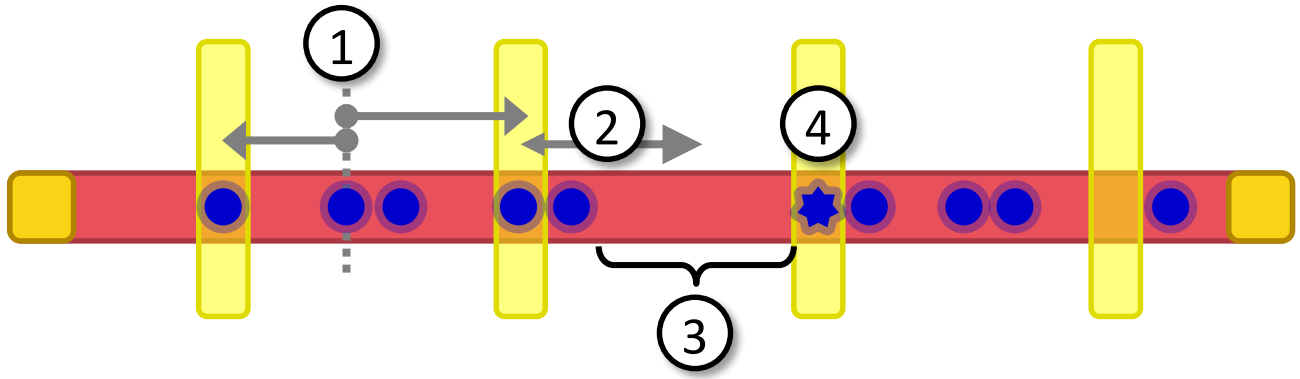




https://dl.acm.org/doi/abs/10.1145/3333336; https://ieeexplore.ieee.org/abstract/document/8107669

Challenges of SK-RM

1. **Time-consuming shift operations** are needed to align a bit with an access port before accesses.
2. **Position errors on shifts**: the drifting speed of skyrmions ≠ speed of nonskyrmions.
3. **Data representation errors**: cannot clearly distinguish how many consecutive bit-0s are there. More consecutive bit-0s are more prone to this type of error.
4. **Time & energy asymmetry to write bit-0s/bit-1s**: Skyrmion generation (creating a bit-1) takes more time & energy than performing a partial shift (creating a bit-0).

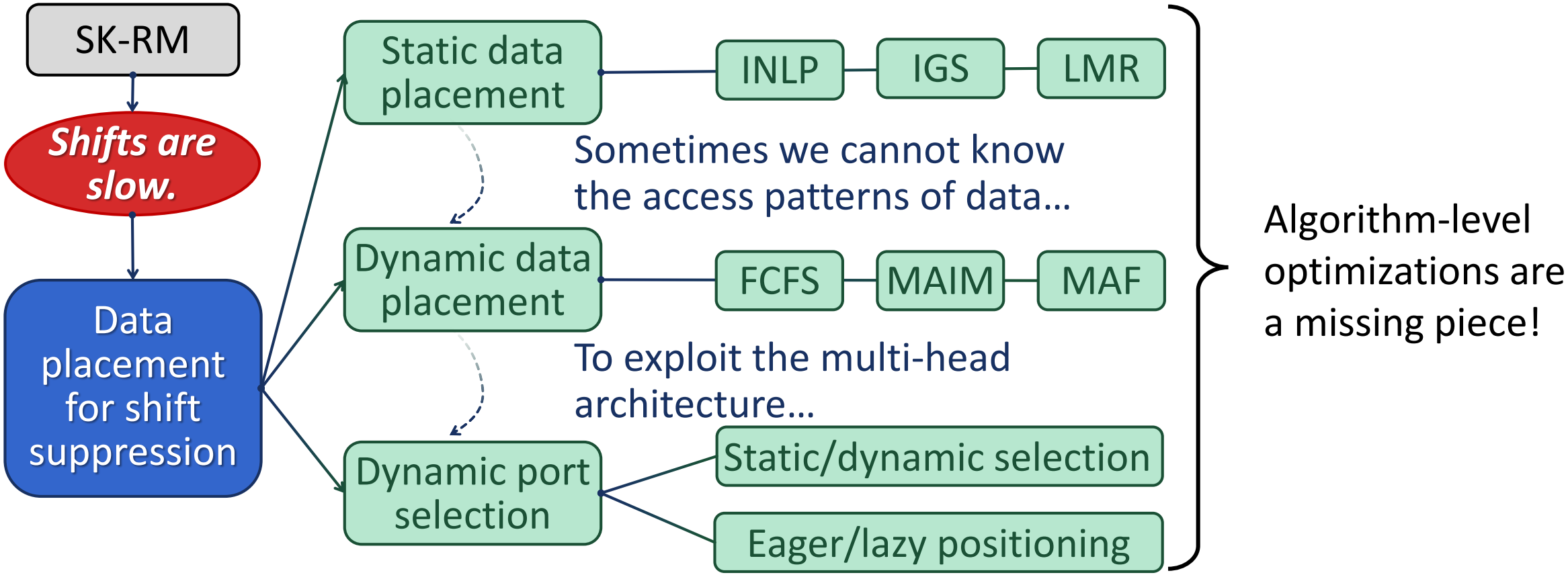


Shifting of Computation Models

- Computation models are impacted by the characteristics of memory media.
  - **Random-access machine (RAM) model**: for DRAM
    - Lacking of considerations of shift operations
  - **SK-RM based model: for SK-RM**
    - Shift operations amplify latency by up to s times

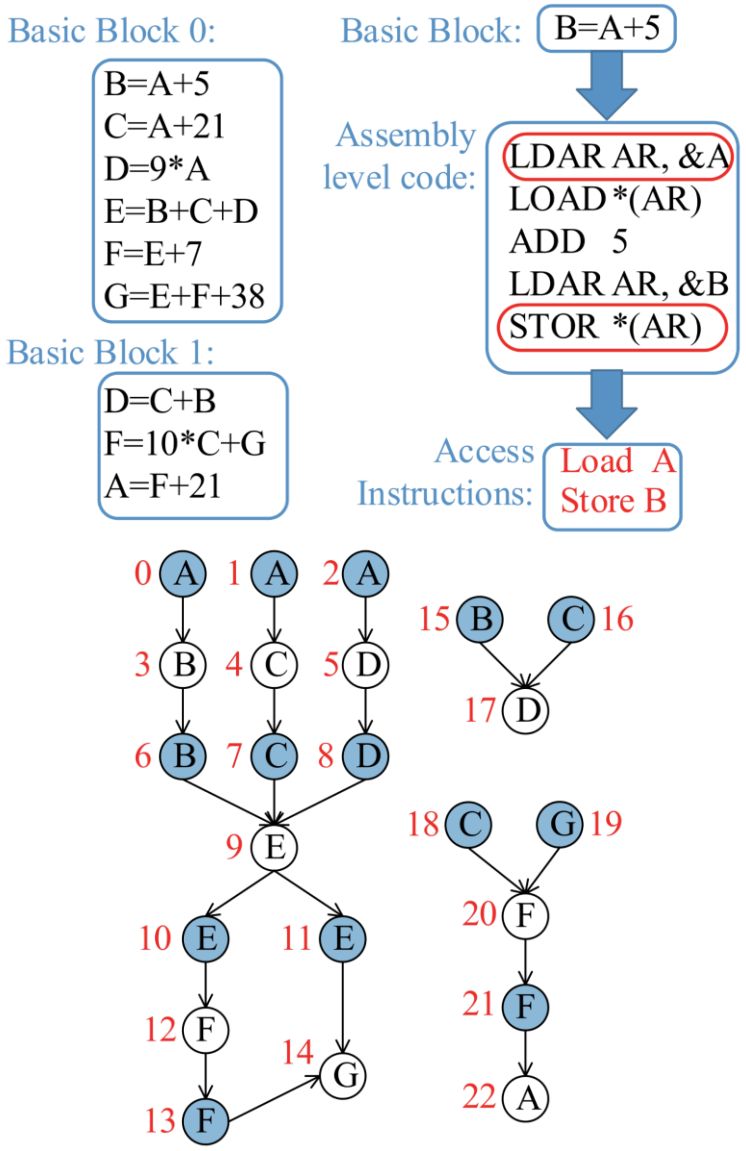
Existing algorithms should be redesigned to fit the novel SK-RM based model!

Big Map of Current Data Placement Policy



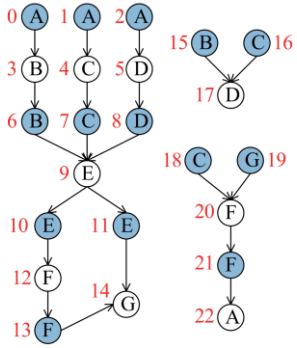
Optimal Static Data Placement

- Main idea: **code → data dependency → data layout to reduce shifts**.
- Solutions to the optimization problem
  - **Integer nonlinear programming (INLP)**: Minimize shifts; optimal but slow.
  - **Instructions group schedule (IGS)**: Minimize shifts but might violate the timing constraint (miss deadline).
    The main idea of IGS is that instructions which access same data are scheduled adjacently. In this way, the number of shift can be reduced.
  - **Longest move reduce (LMR)**: Rearrange data so that no data block requires excessive shifting.



https://ieeexplore.ieee.org/abstract/document/7442797

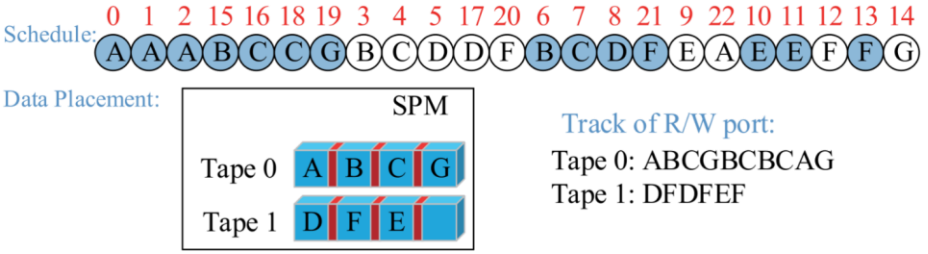
Optimal Static Data Placement (Cont.)



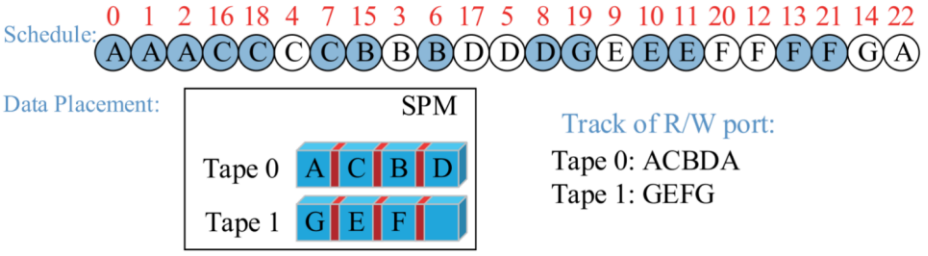
Assumption:
- **Instruction constraint:** Each instruction must be executed once.
- **Dependency constraint:** The dependencies (execution order) among instructions must be preserved.
- **Data allocation constraint:** Each piece of data must be placed on the tape.
- **Distance constraint:** When two instructions *a* and *b* are ordered in the schedule, it means that *b* must be executed right after *a*.

Objective function: Minimize the movement distance between two instructions.

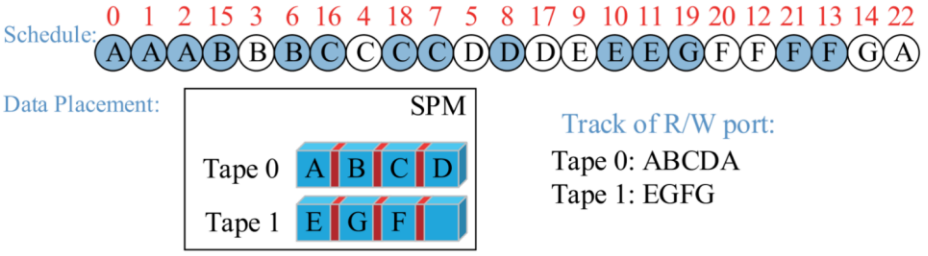
- Main idea: **code → data dependency → data layout to reduce shifts**.
- Solutions to the optimization problem
  - **Integer nonlinear programming (INLP)**: Minimize shifts; optimal but slow.
  - **Instructions group schedule (IGS)**: Minimize shifts but might violate the timing constraint (miss deadline). The main idea of IGS is that instructions which access same data are scheduled adjacently. In this way, the number of shift can be reduced.
  - **Longest move reduce (LMR)**: Rearrange data so that no data block requires excessive shifting.



Original: 18 shift operations



IGS: 10 shift operations

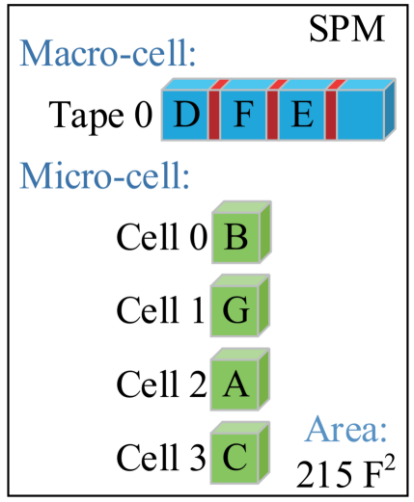


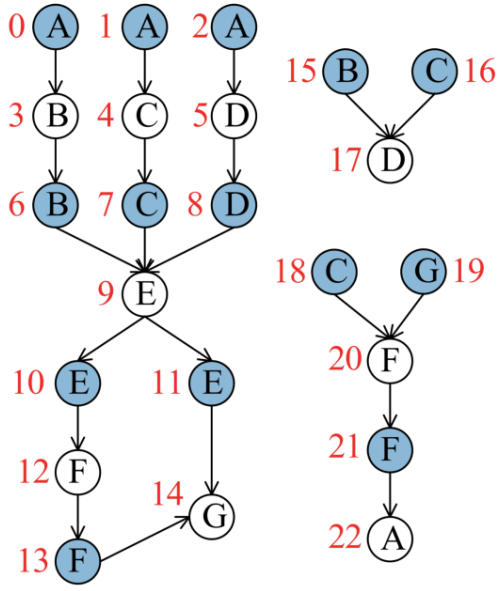
INLP: 9 shift operations

https://ieeexplore.ieee.org/abstract/document/7442797

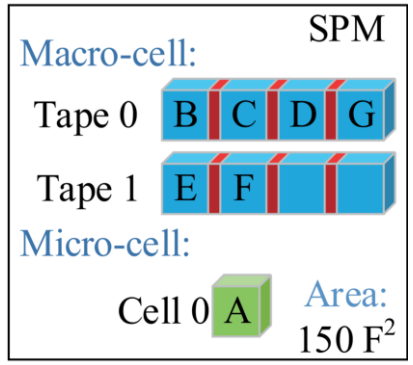
Optimal Static Data Placement (Cont.)

- Main idea: **code → data dependency → data layout to reduce shifts**.
- Solutions to the optimization problem
  - **Integer nonlinear programming (INLP)**: Minimize shifts; optimal but slow.
  - **Instructions group schedule (IGS)**: Minimize shifts but might violate the timing constraint (miss deadline).
    The main idea of IGS is that instructions which access same data are scheduled adjacently. In this way, the number of shift can be reduced.
  - **Longest move reduce (LMR)**: Rearrange data so that no data block requires excessive shifting.

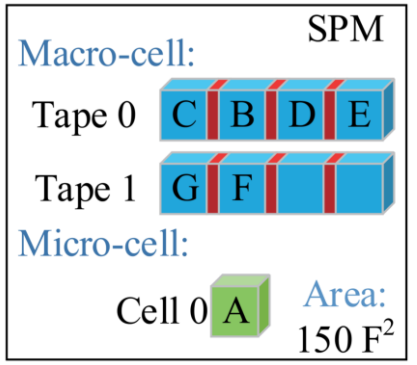




Original + LMR: 5 shift operations



INLP + LMR: 4 shift operations



INLP + LMR: 5 shift operations

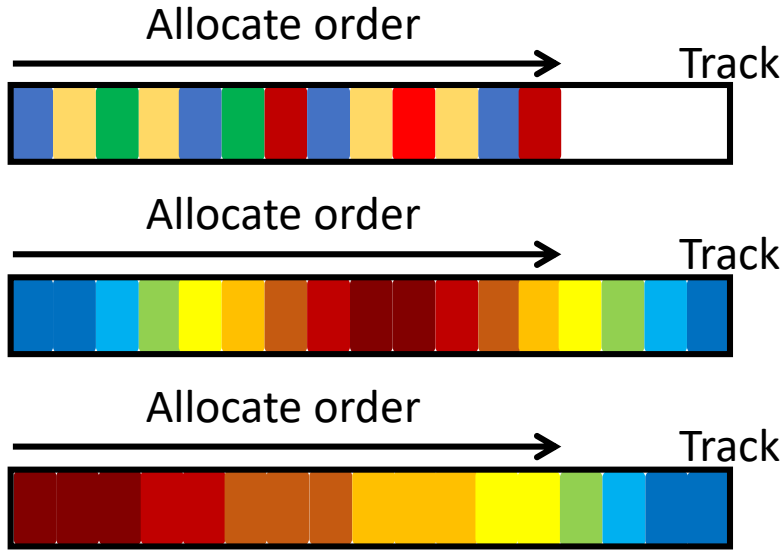
https://ieeexplore.ieee.org/abstract/document/7442797

Dynamic Data Placement

- Main idea: heuristics to place the written data on the positions in tracks.
- Approaches
  - **First come first store (FCFS)**: Allocate the data to the next free position on track.
  - **Most access in middle (MAIM)**: Frequently referenced data are placed in the middle part of each track.
  - **Most access first (MAF)**: Allocate the data from one end to the other according to the frequency.

- ***Red: hot data***
- ***Blue: cold data***

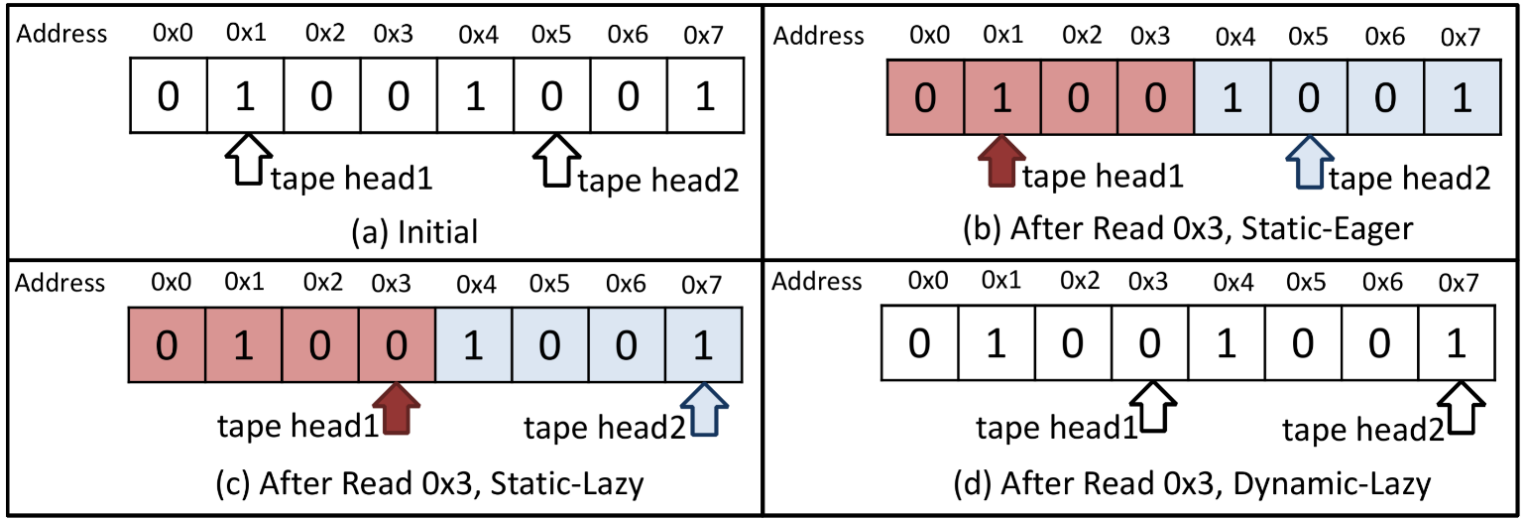
(The temperature of colors indicate the hotness of data)



https://ieeexplore.ieee.org/abstract/document/7304358

Dynamic Port Selection

- Main ideas:
  - ***TapeCache = "RM as LLC" + "HW/SW management co-design"***
  - HW/SW management co-design = read-only heads + head management policies
  - **Head management policies = Head selection + Head positioning**
    - Static/dynamic selection
    - Eager/lazy positioning



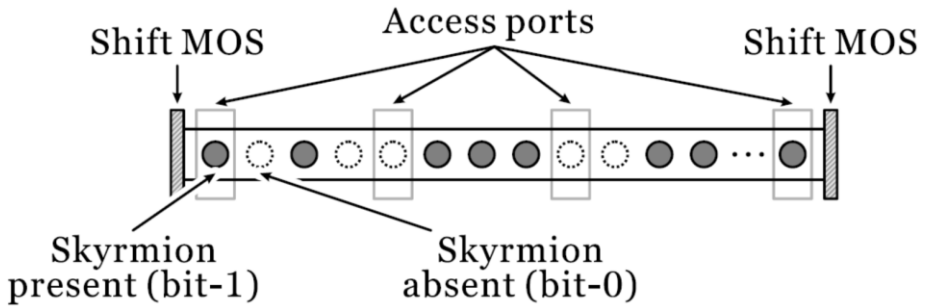
Policy	Meaning	Behavior
Static-Eager	Fixed head assignment + proactively reposition head after access	Always uses the same head; moves it right after each read to be ready for the next predicted access.
Static-Lazy	Fixed head assignment + move only when needed	Keeps head still after read; shifts only when future access demands it.
Dynamic-Lazy	Head chosen at runtime + move only when needed	Selects the head nearest to the target and doesn't reposition after read. Minimizes unnecessary movement.

https://dl.acm.org/doi/pdf/10.1145/2333660.2333707

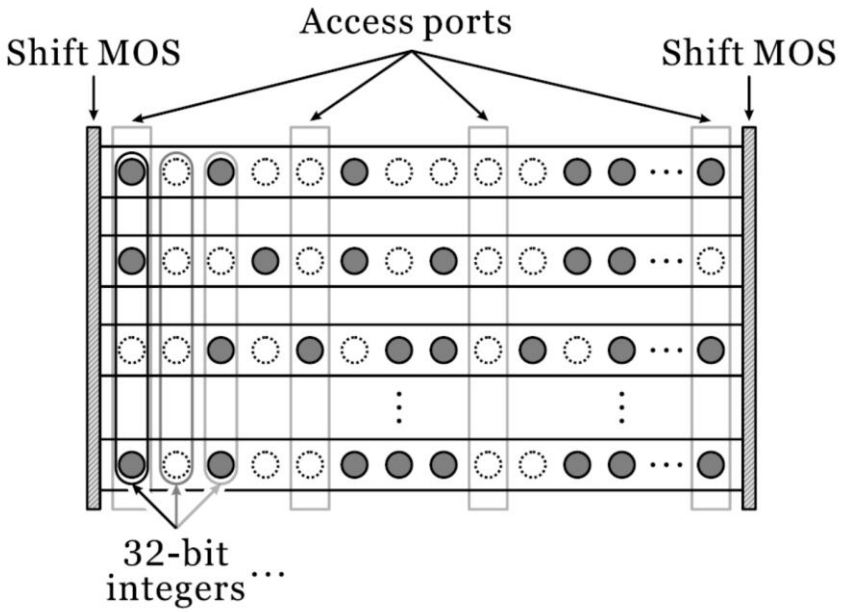
Algorithm Revisit

- Existing algorithms often access in-memory data in a **random-hopping manner**, which creates many unnecessary shifts.
- We want to re-design existing algorithms to reduce shifts and enhance performance.
- ***We take sorting algorithms as a case study, which provides hints for exploiting the random insert/delete operations to suppress shifts.***

Shift-limited Sort for Skyrmion Memory



Skyrmion memory with multiple access ports



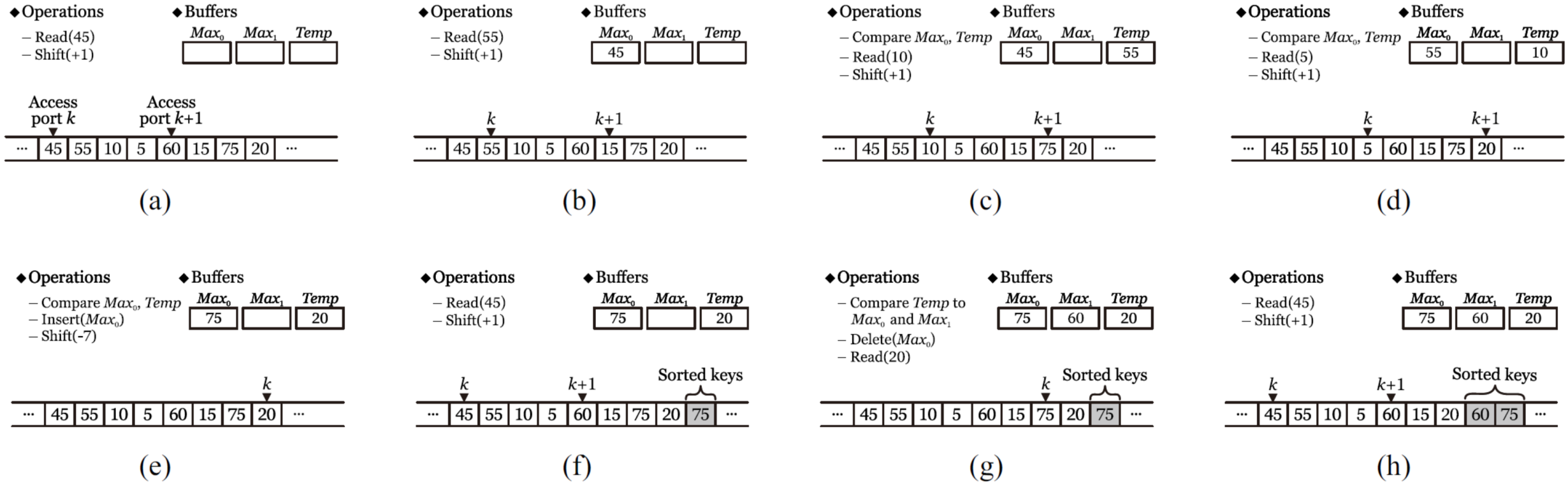
Bit-interleaved and block cluster architecture

Operations	Read	Shift	Delete	Insert
Latency	0.1 ns	0.5 ns	0.8 ns	1 ns

Latency of SK-RM operations

https://ieeexplore.ieee.org/abstract/document/9211559

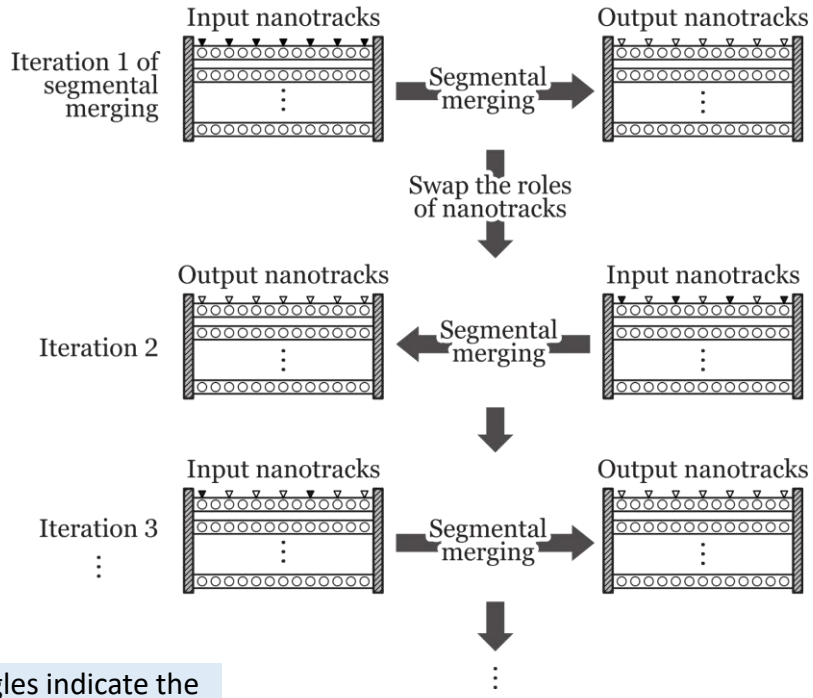
Motivation of Shifted-Limited Sort



Selection sort on SK-RM. (a) Initial state of the racetrack. (b) Resulted racetrack after the data are shifted left by 1 b. (c) Resulted racetrack after the data are shifted left again by 1 b. (d) Key 55 is compared to key 45 and replaces key 45 in the register Max_0. (e) Last key 20 is going to be compared to key 75. (f) Largest key 75 is written to the rightmost position of the nanotrack and is properly sorted. All data are then shifted to align the first key, 45, with the leftmost port. (g) In the second iteration of the selection sort, the maximum key of the previous iteration, 75, will be removed once encountered again. (h) Second-largest key 60 is written back to the rightmost position of the nanotrack, and is properly sorted.

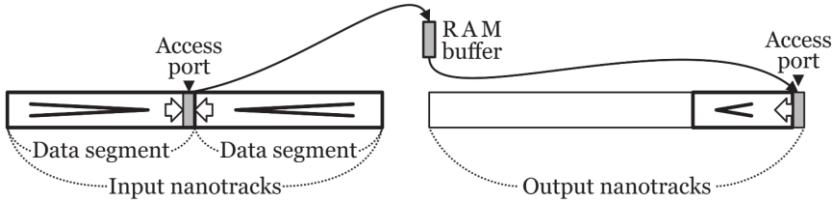
Method of Shifted-Limited Sort

- Two Phases of the Shift-limited Sort
  - **Pre-sorting phase**
    - The minimalist data segments are sorted in a recursive back-to-back manner.
  - **Merging phase**
    - Adjacent data segments are recursively merged until only one data segment lefts, which contains the sorted results.

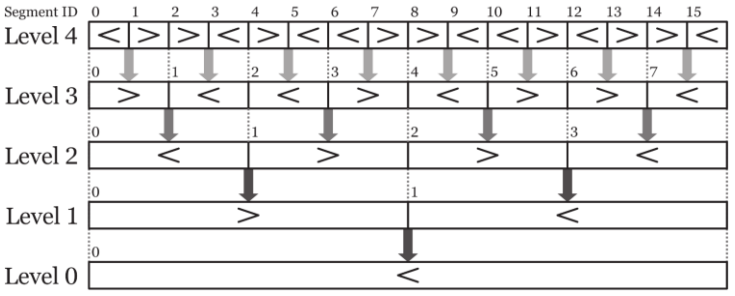


The solid triangles indicate the access ports to be used in this round of recursive merging.

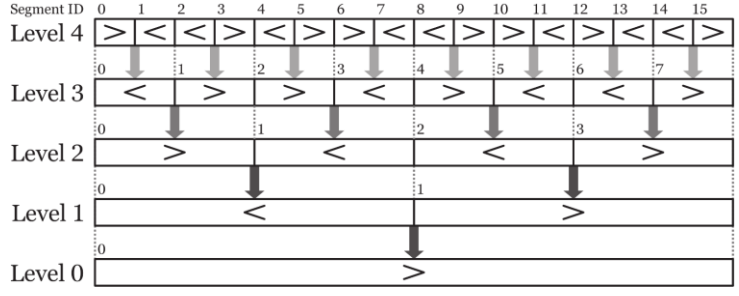
Recursive Segmental Merging



Segmental merging: efficiently merging two sorted arrays with minimal shifts



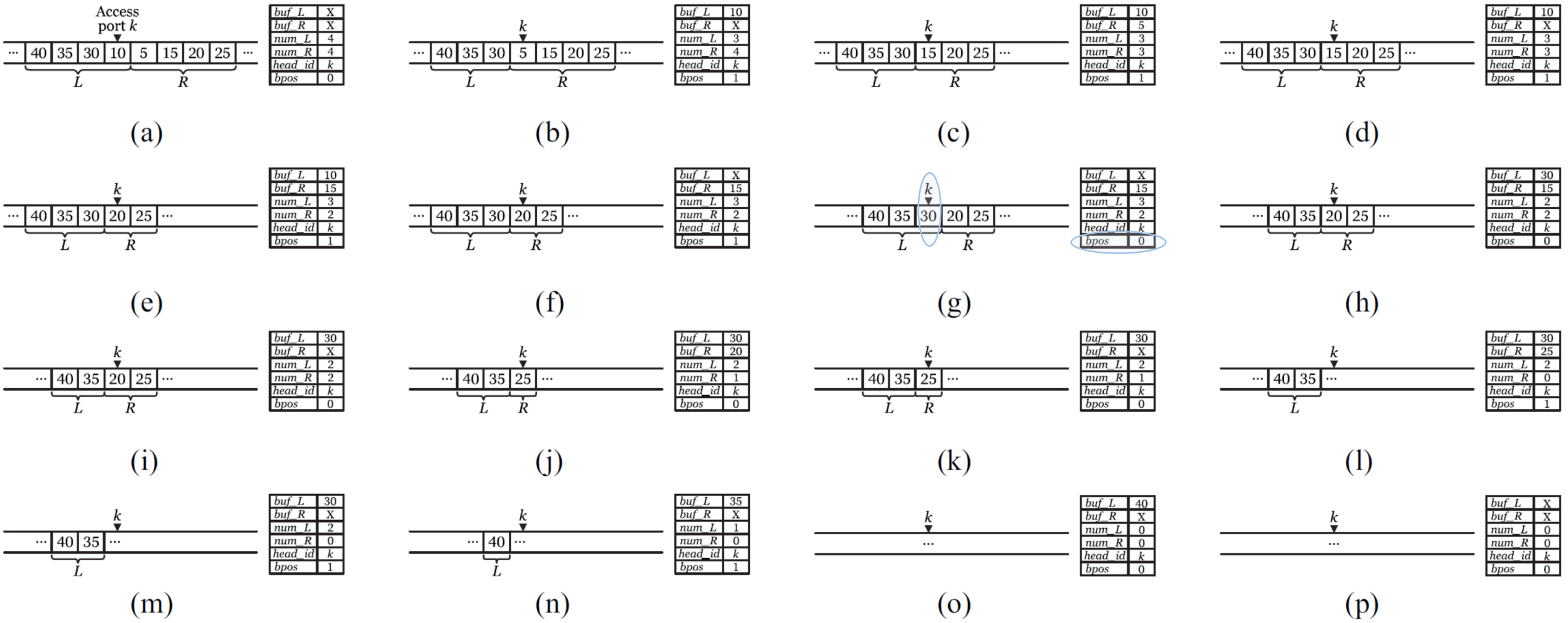
Target order: ascending



Target order: descending

https://ieeexplore.ieee.org/abstract/document/9211559

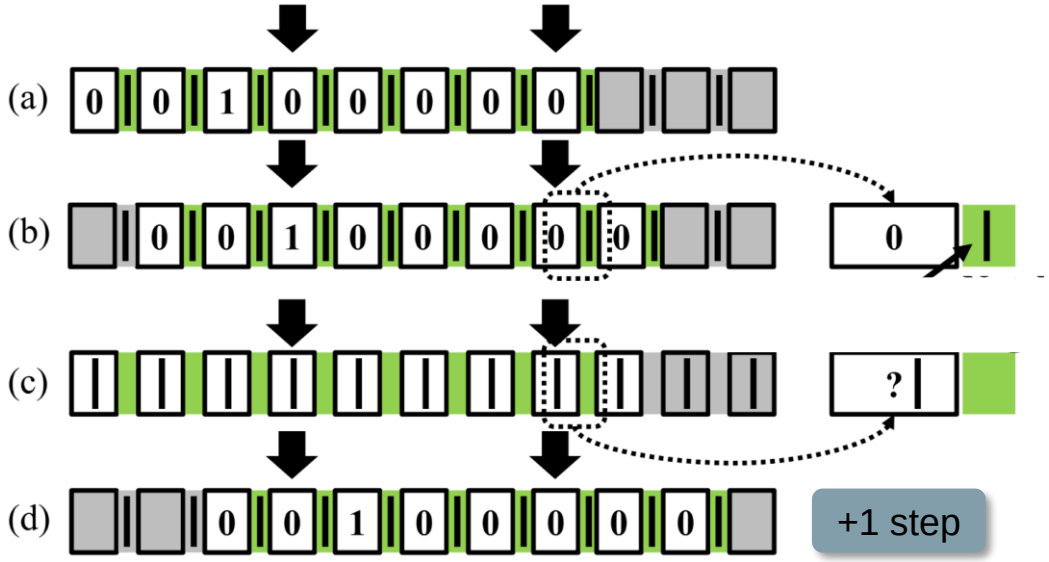
Running Example of Shift-Limited Sort



(a) Initial state of the racetrack. (b) D-SHL(10). (c) D-SHL(5). (d) Compare buf_L and buf_R; I-SHL(5). (e) D-SHL(15). (f) Compare buf_L and buf_R; I-SHL(10). (g) SHR(1). (h) D-SHL(30). (i) Compare buf_L and buf_R; I-SHL(15). (j) D-SHL(20). (k) Compare buf_L and buf_R; I-SHL(20). (l) D-SHL(25). (m) Compare buf_L and buf_R; I-SHL(25). (n) D-SHL(35). (o) D-SHL(40). (p) I-SHL(40).

Position Errors

- Main idea:
  - To eliminate the **stop-in-middle error, subthreshold shift (STS)** is proposed to perform a more reliable shift in two stages.
  - To detect and recover the **out-of-step error, position error correction code (p-ECC)** is proposed.



- Stop-in-middle error
- Out-of-step error
  - Under-shifted position error ($(-n)$-step error)
  - Over-shifted position error ($(+n)$-step error)

https://dl.acm.org/doi/abs/10.1145/2749469.2750388

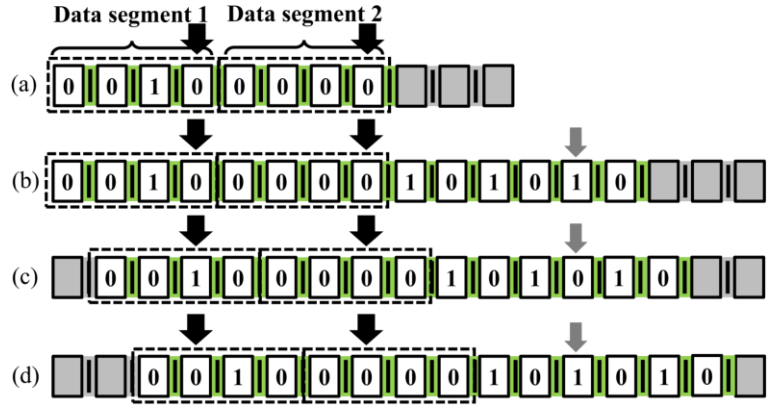
Position Errors (Cont.)

- **Sub-threshold Shift (STS)**
  - **Stage-1**: for an N-step distance shift operation, a pulse of high driving current density is applied.
  - **Stage-2**: after stage-1, an extra pulse of driving current is applied to make a sub-threshold shift.

Distance	$\pm k$ Step Error Rate		
	$k = 1$	$k = 2$	$k \geq 3$
1	4.55×10^{-5}	1.37×10^{-21}	too small
2	9.95×10^{-5}	1.19×10^{-20}	too small
3	2.07×10^{-4}	5.59×10^{-20}	too small
4	3.76×10^{-4}	1.80×10^{-19}	too small
5	5.94×10^{-4}	4.47×10^{-19}	too small
6	8.43×10^{-4}	9.96×10^{-18}	too small
7	1.10×10^{-3}	7.57×10^{-15}	too small

- Probability of out-of-step position error

- **Single-Step Error Detection (SED)**
  - When domains are successfully shifted by even-step distance, the p-ECC bit read out should be equal to that read out before shift operation.
  - When domains are successfully shifted by odd-step distance, the p-ECC bit read out is reversed.

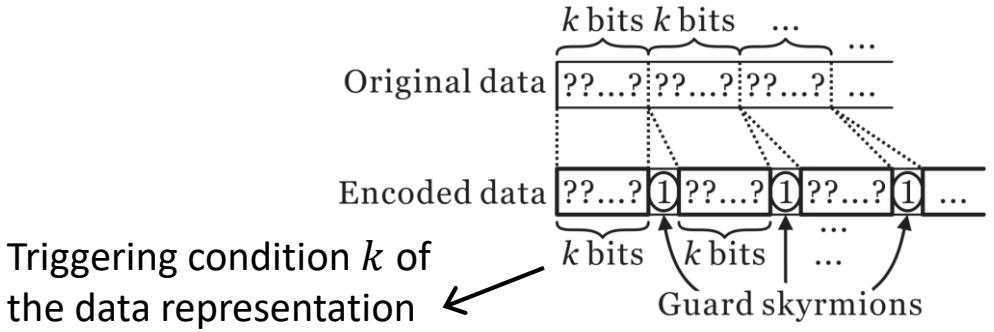


- SED p-ECC

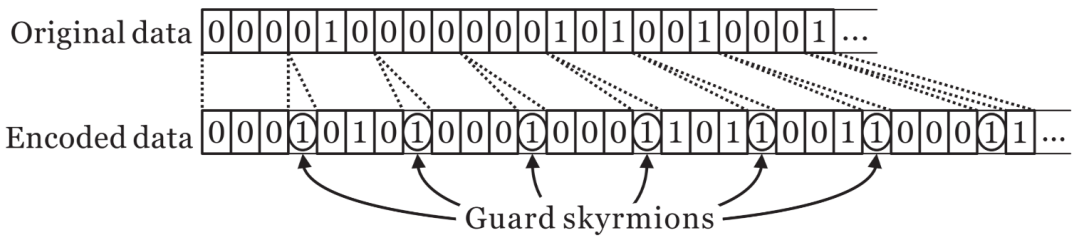
https://dl.acm.org/doi/abs/10.1145/2749469.2750388

Granularity-driven Management for Reliable Skyrmion Racetrack Memories

- Main Idea
  - Exploit two existing flyweight bit-stuffing approaches, called **FGS** and **GGS** in this work, to solve the data representation problem of SK-RM.
  - The design space of different **data layout** and **alignment strategies** is explored, with joint consideration over the parallel access capability of SK-RM.
    ⇒ We propose different management schemes, namely Modes S, M, and L, for data at different degrees of granularity.



Working principle.

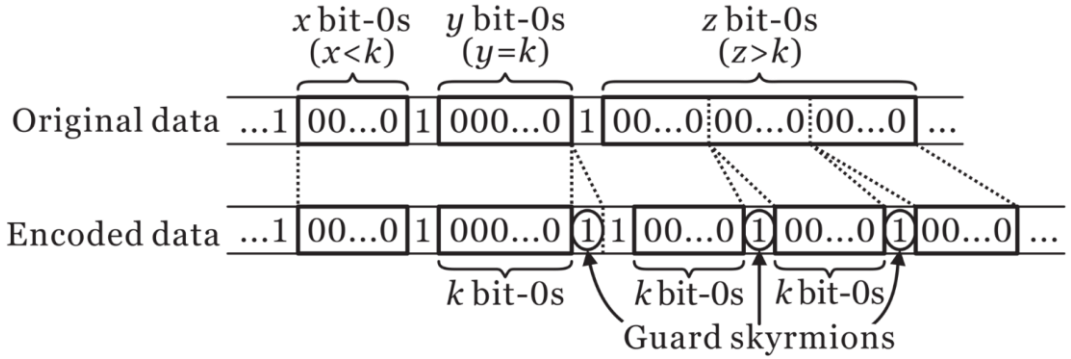


Working example with $k = 3$.

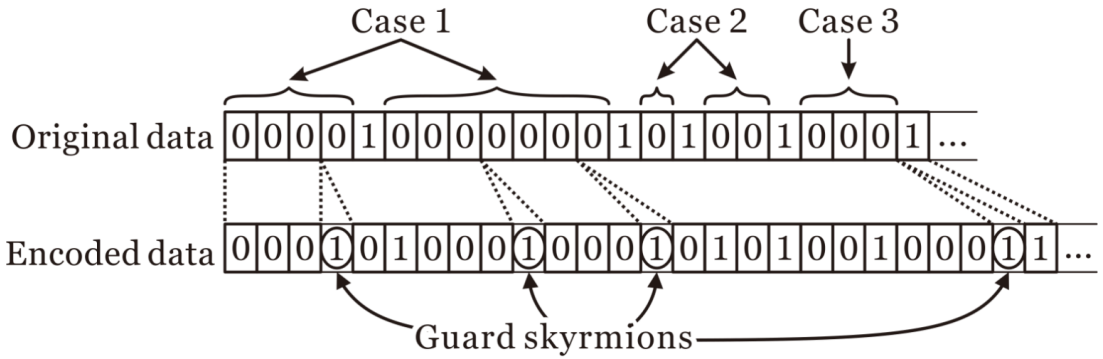
FGS: Fixed Guard Skyrmion

Granularity-driven Management for Reliable Skyrmion Racetrack Memories (Cont.)

- Main Idea
  - Exploit two existing flyweight bit-stuffing approaches, called **FGS** and **GGS** in this work, to solve the data representation problem of SK-RM.
  - The design space of different **data layout** and **alignment strategies** is explored, with joint consideration over the parallel access capability of SK-RM. ⇒ We propose different management schemes, namely Modes S, M, and L, for data at different degrees of granularity.



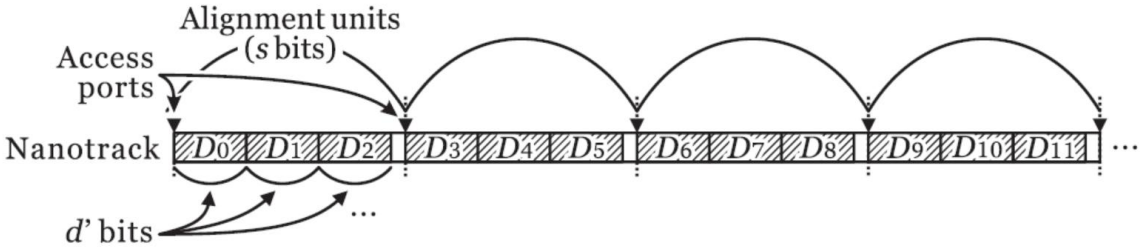
Working principle.



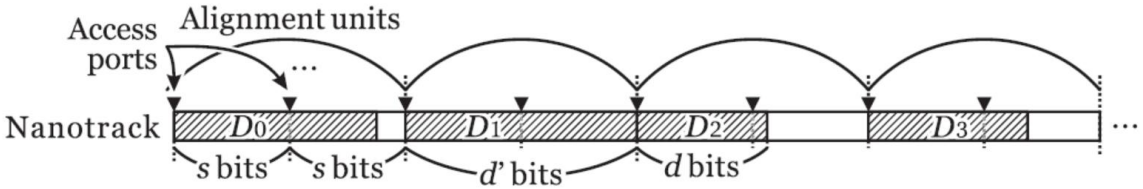
Working example with $k = 3$.

GGS: Greedy Guard Skyrmion

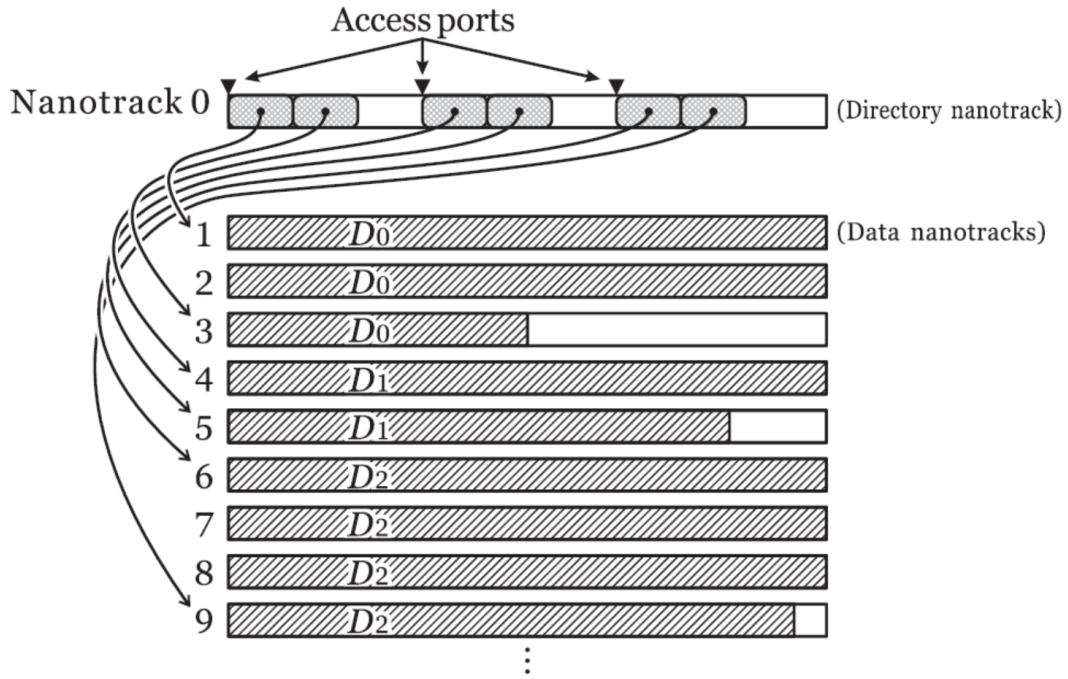
Granularity-driven Management for Reliable Skyrmion Racetrack Memories (Cont.)



(a) Mode S.



(b) Mode M.

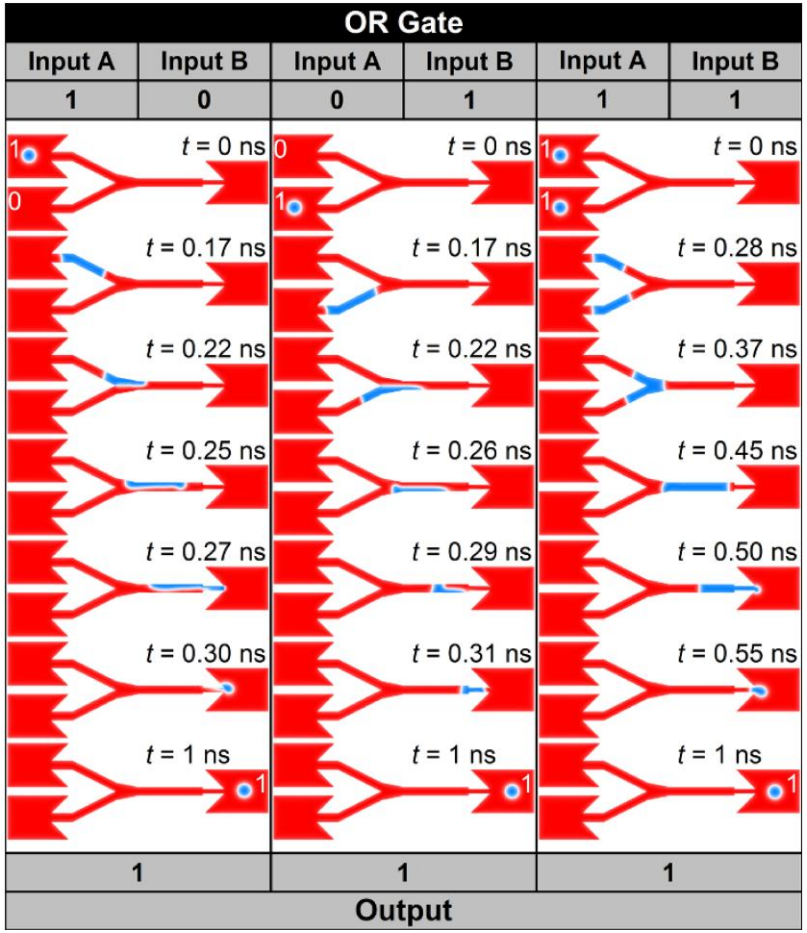


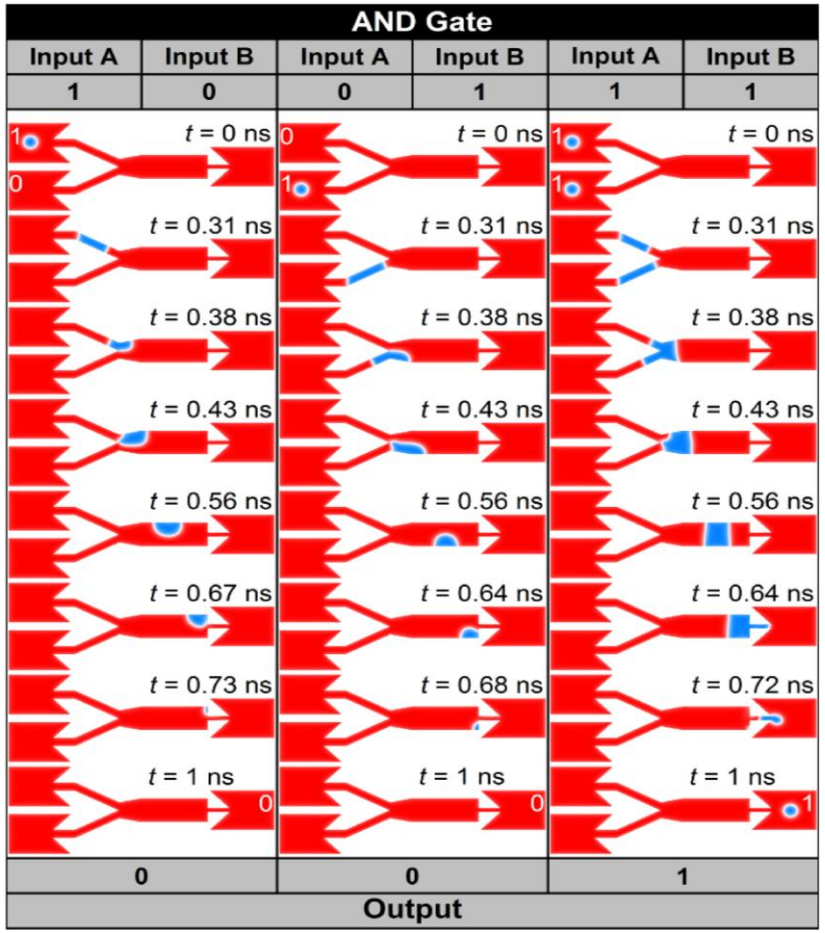
(c) Mode L.

Properties	Mode S	Mode M	Mode L
Encoded data item size	$<$ data segment.	$\geq$ data segment, $<$ nanotrack.	$\geq$ nanotrack.
Potential application scenarios	SPM & high-level caches	Low-level caches	Main memory & persistent storage
Variable-sized data item support?	✓	✓ ($<$ nanotrack size)	✓ ($\geq$ nanotrack size)
Data encoding	FGS	GGS	GGS
Data alignment	1^+ data items $\rightarrow$ 1 data segment	1 data item $\rightarrow$ 1^+ data segments	1 data item $\rightarrow$ 1^+ nanotracks
Directory needed?	×	×	✓

Skyrmion Based Logic Gates

- AND/OR gates





https://www.nature.com/articles/srep09400

Skyrmion Based Logic Gates (Cont.)

- Reversible AND/OR gate & INV/COPY gate

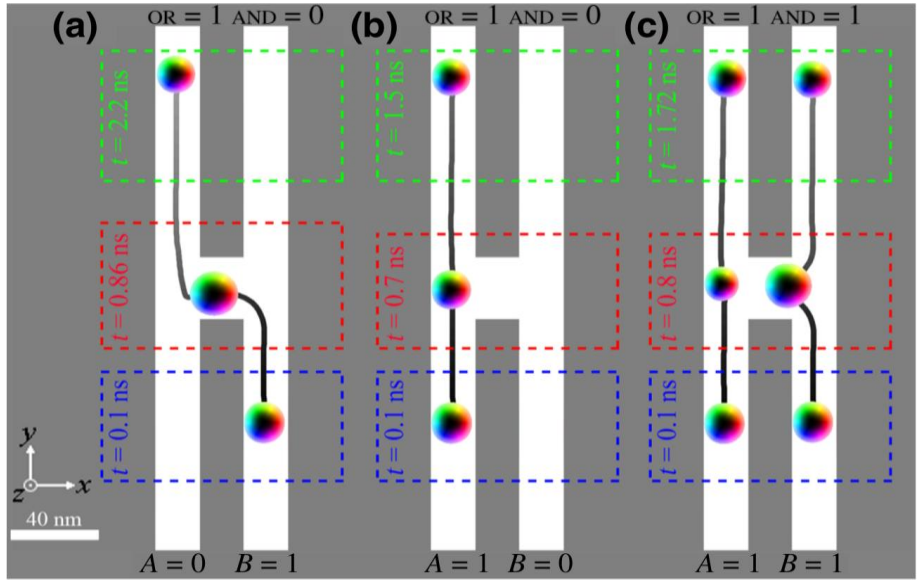


TABLE I. Truth table for the AND/OR gate.

Inputs			Outputs	
A	B	N	AND	OR
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	2	1	1

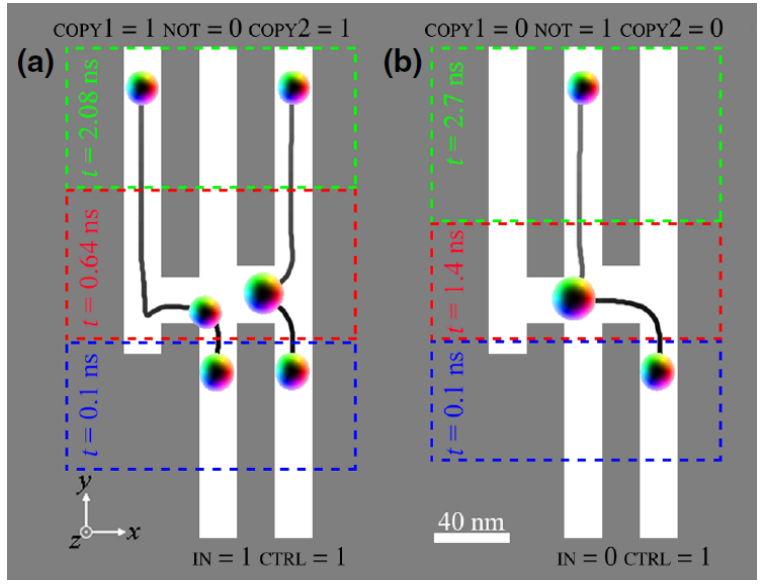


TABLE II. Truth table for the INV/COPY gate.

Inputs			Outputs		
CTRL	IN	N	COPY1	NOT	COPY2
1	0	1	0	1	0
1	1	2	1	0	1

https://journals.aps.org/prapplied/abstract/10.1103/PhysRevApplied.12.064053

Skyrmion Based Logic Gates (Cont.)

- Fredkin gates
  - Any logical or arithmetic operation can be constructed entirely of Fredkin gates.

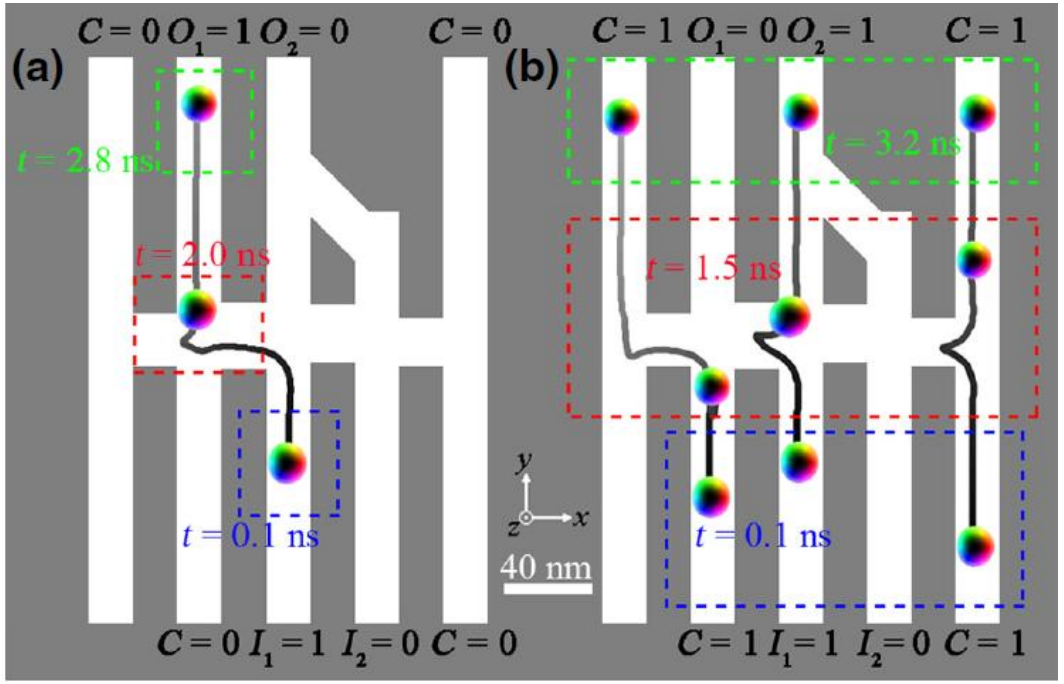


TABLE III. Truth table for the Fredkin gate.

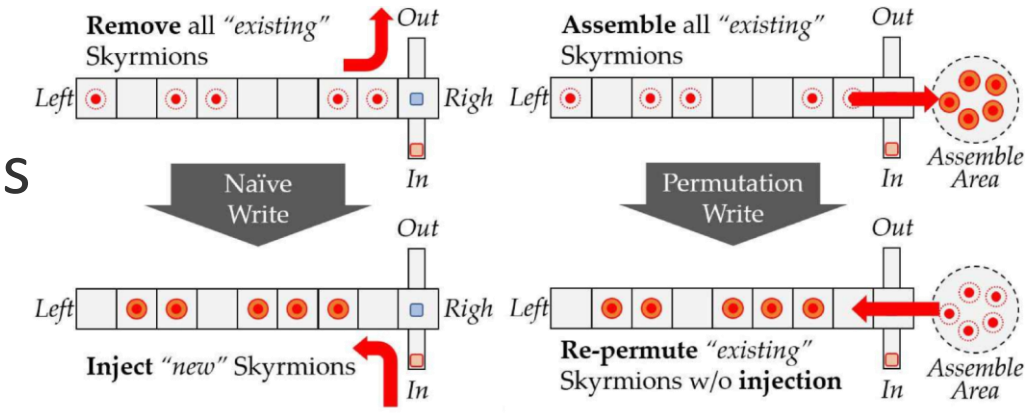
Inputs				Outputs		
C	I_1	I_2	N	C	O_1	O_2
0	0	0	0	0	0	0
0	0	1	1	0	0	1
0	1	0	1	0	1	0
0	1	1	2	0	1	1
1	0	0	2	1	0	0
1	0	1	3	1	1	0
1	1	0	3	1	0	1
1	1	1	4	1	1	1

https://journals.aps.org/prapplied/abstract/10.1103/PhysRevApplied.12.064053

Permutation Write to Reuse Skyrmion

Reusing Skyrmions

- Main idea: Reuse pre-produced skyrmions to reduce future skyrmion generation.
- Permutation-Write Strategy
  - **Assembling step**: Let all the existing Skyrmions will be “assembled consecutively” in the Assemble Area.
  - **Re-permuting step**: Re-permute the existing Skrymions (and to inject insufficient Skyrmions if needed).
  - **Removing step**: One-by-one remove all the excessive assembled Skyrmions from the track.

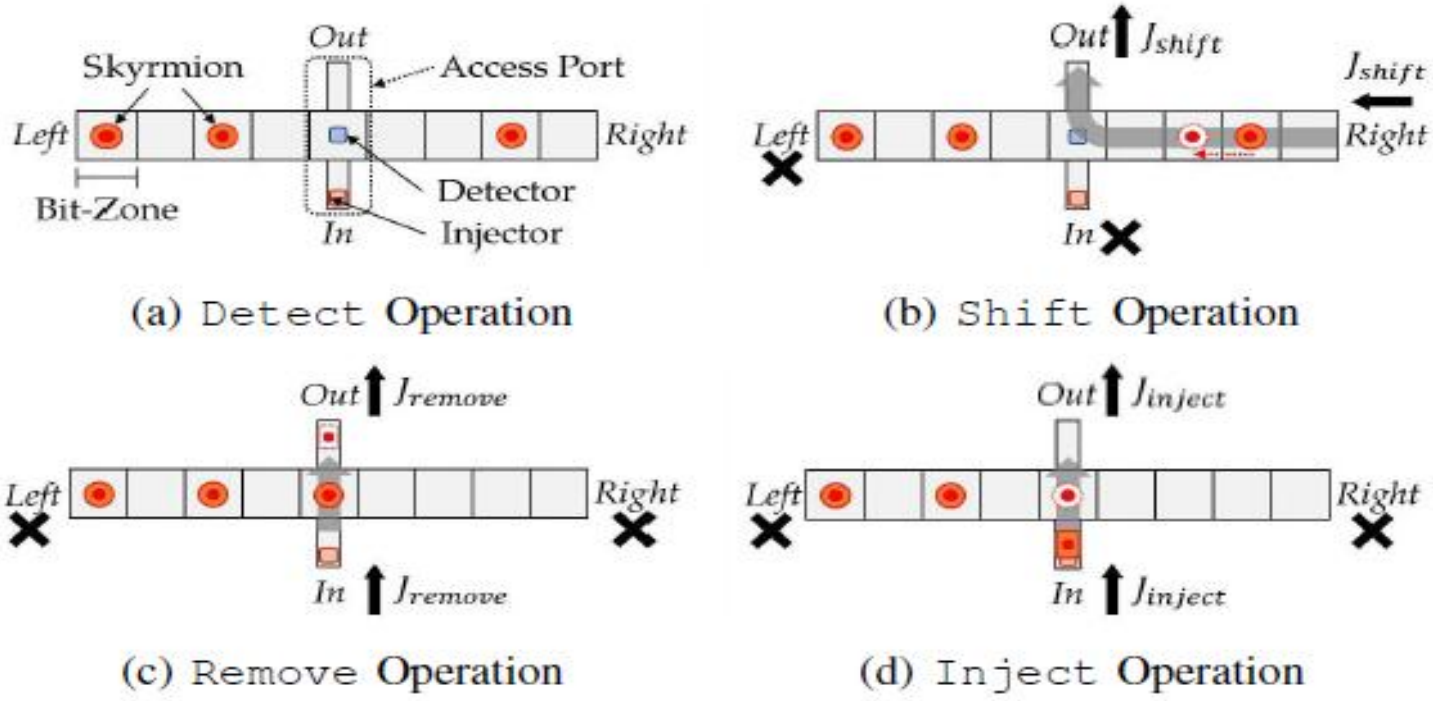


(a) Naïve Write Strategy (b) Permutation-Write Strategy

https://ieeexplore.ieee.org/abstract/document/9218642

Basic Structure and Operations for Sky-RM

- Basic Structure
  - Injector
  - Detector
- 4 Basic Operations
  - Detect (read)
  - Remove (write 0)
  - Shift
  - Inject (write 1)

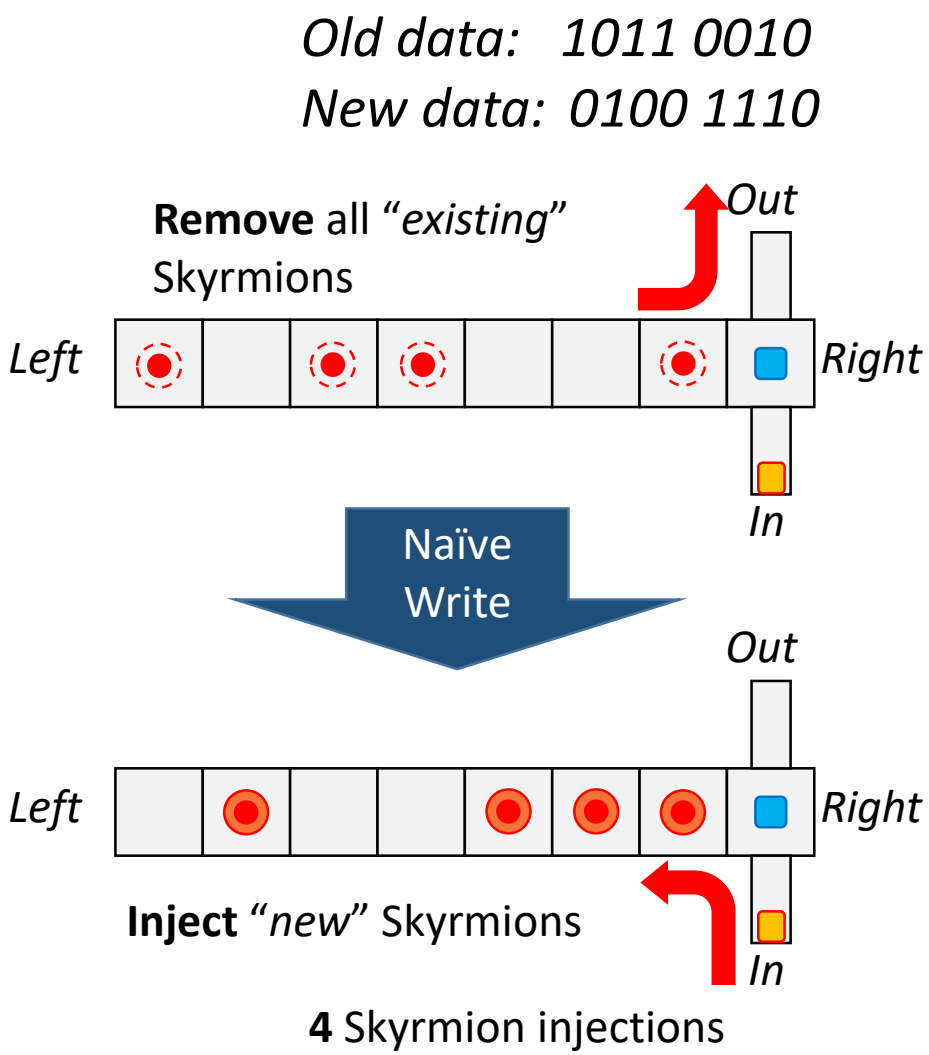


Cross-shape racetrack w/ 4 terminals

SK-RM Challenge

- **High overhead** when writing new data (Injecting new skyrmions to the racetrack)

- Right figure shows a Naïve write strategy.
  - Remove all existing Skyrmions by "**shift right**" and "**remove**"
  - "**Inject**" and "**shift left**" new Skyrmions



Existing Works for NVM Write Reduction

- **Data-Comparison Write (DCW)**
  - Compare the contents between old and new data words
  - Only update the bits when corresponding bit values are opposite in both words.

  3 Skyrmion injections in the previous example

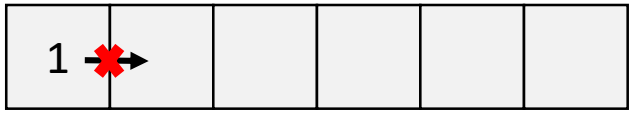
- **Flip-N-Write**
  - Inherits the design of DCW
  - Introduce a flip-bit to bound the total number of bit updates
  - Set the flip-bit, every bits will be interpreted as its opposite value

  2 Skyrmion injections in the previous example

Motivation

- **DCW** and **Flip-N-Write** are developed based on cell-based NVM
  - Bit-by-bit comparison
  - Overlook the **topologically-protected** and **particle-like** feature of SK-RM.

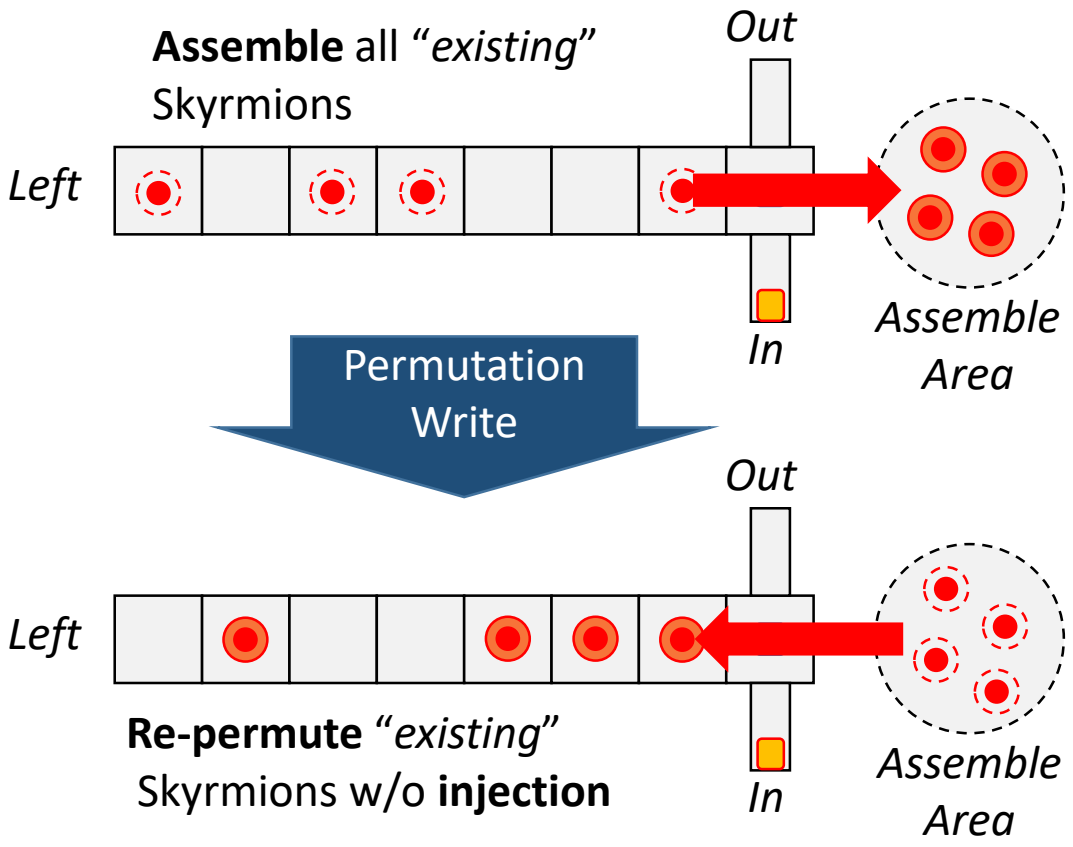
Cell-based NVM:



- **Permutation-Write** (PW) is proposed to exploit the unique features of SK-RM
  - Not write reduction, but **injection reduction** (i.e., minimizing 0→1)
  - **Re-use** the existing Skyrmions to form new data
  - Instead of bit-by-bit comparison, PW focuses **on total number of "1(s)" (Skyrmions)** on the racetrack.

Design of Permutation-Write (PW) Strategy

- Assemble Skyrmions in old data
- Re-permute the Assembled Skyrmions to form the new data



Old data: 1011 0010
New data: 0100 1110

NO Skyrmion injection!!!

Architecture: Word-based Structure for SK-RM

- Treat SK-RM as a promising replacement of memory
- Modern CPU usually accesses the memory contents in the unit of a word (e.g., 32 bits or 64 bits)

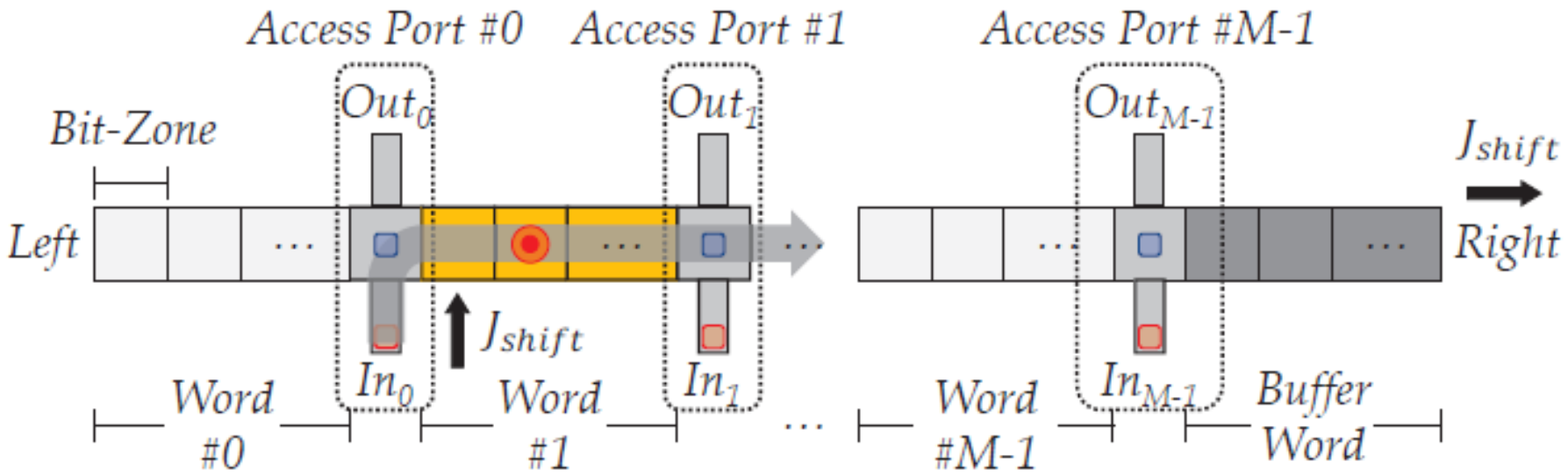
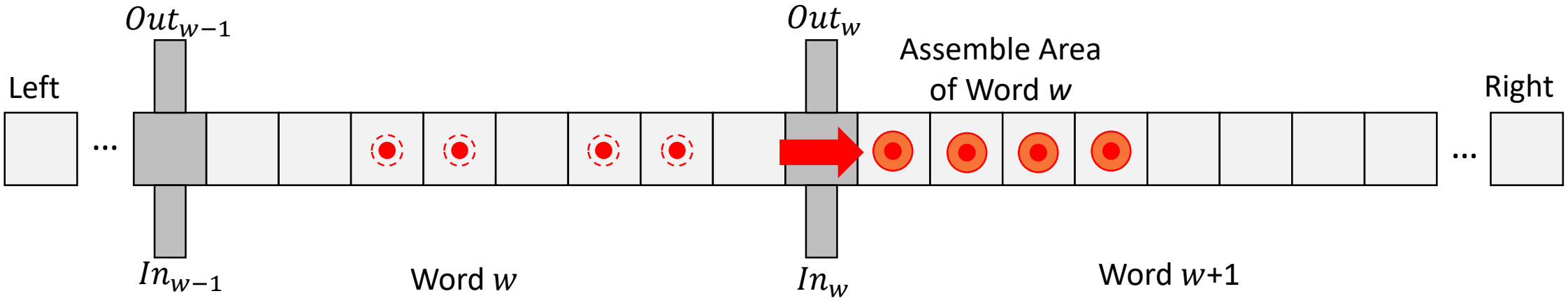


Fig. 3. Word-based Structure of Sky-RM.

PW Algorithm

- Step1: Assembling old Skyrmions
- Example: Write new data to Word w
  - Old Word w data: 0011 0110
  - New Word w data: 1000 1001

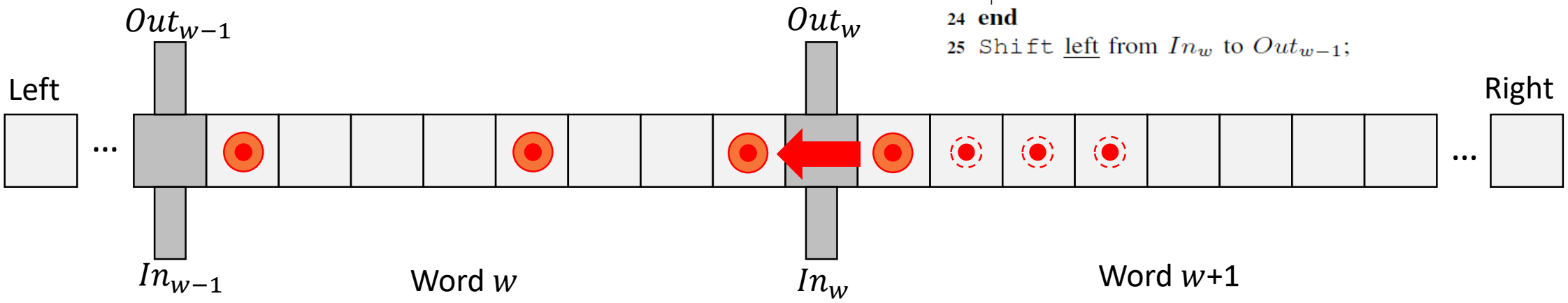
```
// 1) Assemble the Existing Skyrmions
1  sky_count ← 0;
2  Shift right from In_{w-1} to Out_w;
3  for i = N : 1 do
4      b ← Detect the binary value of i^th bit in word w;
5      if b == 0 then
6          Shift out the bit 0 from In_{w-1} to Out_w;
7      else
8          Shift right an existing Skyrmion from In_{w-1} to
           Right;
9          sky_count ← sky_count + 1;
10     end
11 end
```



PW Algorithm (Conti)

- Step2: Re-permuting & Injecting
- Example: Write new data to Word w
  - Old Word w data: 0011 0110
  - New Word w data: 1000 1001

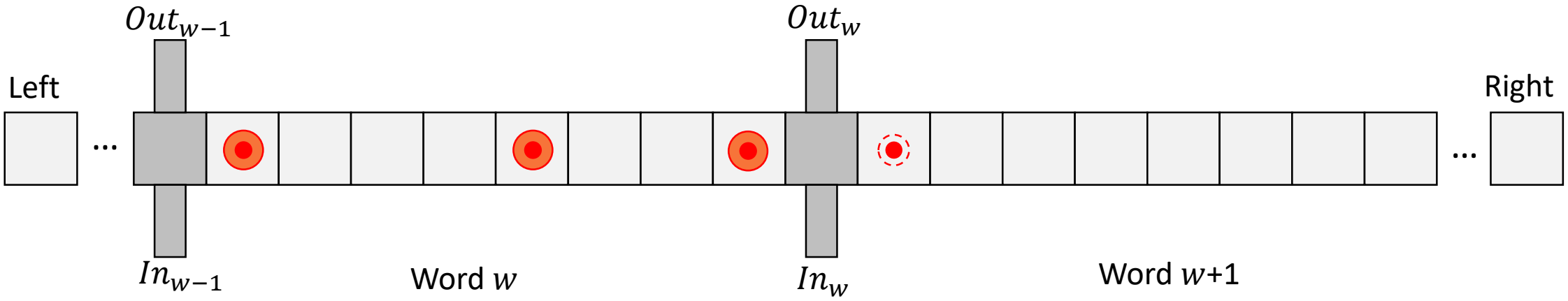
```
// 2) Re-permute the Existing Skyrmions &
   Inject New Skyrmions (if needed)
12 for i = 1 : N do
13     if D[i] == 0 then
14         Shift left from In_w to Out_{w-1} for creating a bit "0"
           into the word w;
15     else
16         if sky_count>0 then
17             Shift left an existing Skyrmion from Right to
               Out_{w-1};
18             sky_count ← sky_count - 1;
19         else
20             Shift left from In_w to Out_{w-1};
21             Inject a new Skyrmion from In_w;
22         end
23     end
24 end
25 Shift left from In_w to Out_{w-1};
```



PW Algorithm (Conti)

- Step3: Remove excess Skyrmions (if any)
- Example: Write new data to Word w
  - Old Word w data: 0011 0110
  - New Word w data: 1000 1001

```
// 3) Remove Excess Skyrmions (if any)
while sky_count>0 do
    Shift left from Right to Out_w;
    Remove an excess Skyrmion from Out_w;
    sky_count ← sky_count - 1;
end
```



Analysis

- Given old data word D' & new data word D, an **Analysis Table** can be constructed to summarize the costs of different write strategies.
  - N is the word size
  - Q (resp. Q') is the number of one in D (resp. D')

TABLE I
OPERATIONAL COSTS UPON WRITING A DATA WORD BY DIFFERENT WRITE STRATEGIES.

	Naïve	DCW	Flip-N-Write	Permutation-Write
Shift	$2N$	$2N$	$2(N+1)$	$2(N+1)+max\{Q'-Q,0\}$
Detect	0	N	$N+1$	N
Remove	N	$\sum_{i=0}^{N-1}(D[i]\oplus D'[i])\wedge D'[i]$	$(f\oplus f'\wedge f')+\sum_{i=0}^{N-1}(\overline{D}[i]\oplus D'[i]\wedge D'[i])$	$max\{Q'-Q,0\}$
Inject	Q	$\sum_{i=0}^{N-1}(D[i]\oplus D'[i])\wedge D[i]$	$(f\oplus f'\wedge f)+\sum_{i=0}^{N-1}(\overline{D}[i]\oplus D'[i]\wedge \overline{D}[i])$	$max\{Q-Q',0\}$

Experimental Setup

- Evaluation Dataset: **MiBench**
- Evaluated Strategies:
  - **Naïve**, **DCW**, **Flip-N-Write**, and **Permutation-Write**

- Since the SK-RM is still maturing, instead of showing the real value, we measure the **operational costs** of evaluated strategies.
- We develop an in-house trace-driven simulator to model the **word-based structure of Sky-RM** and measure the **operational costs**.

Experimental Results

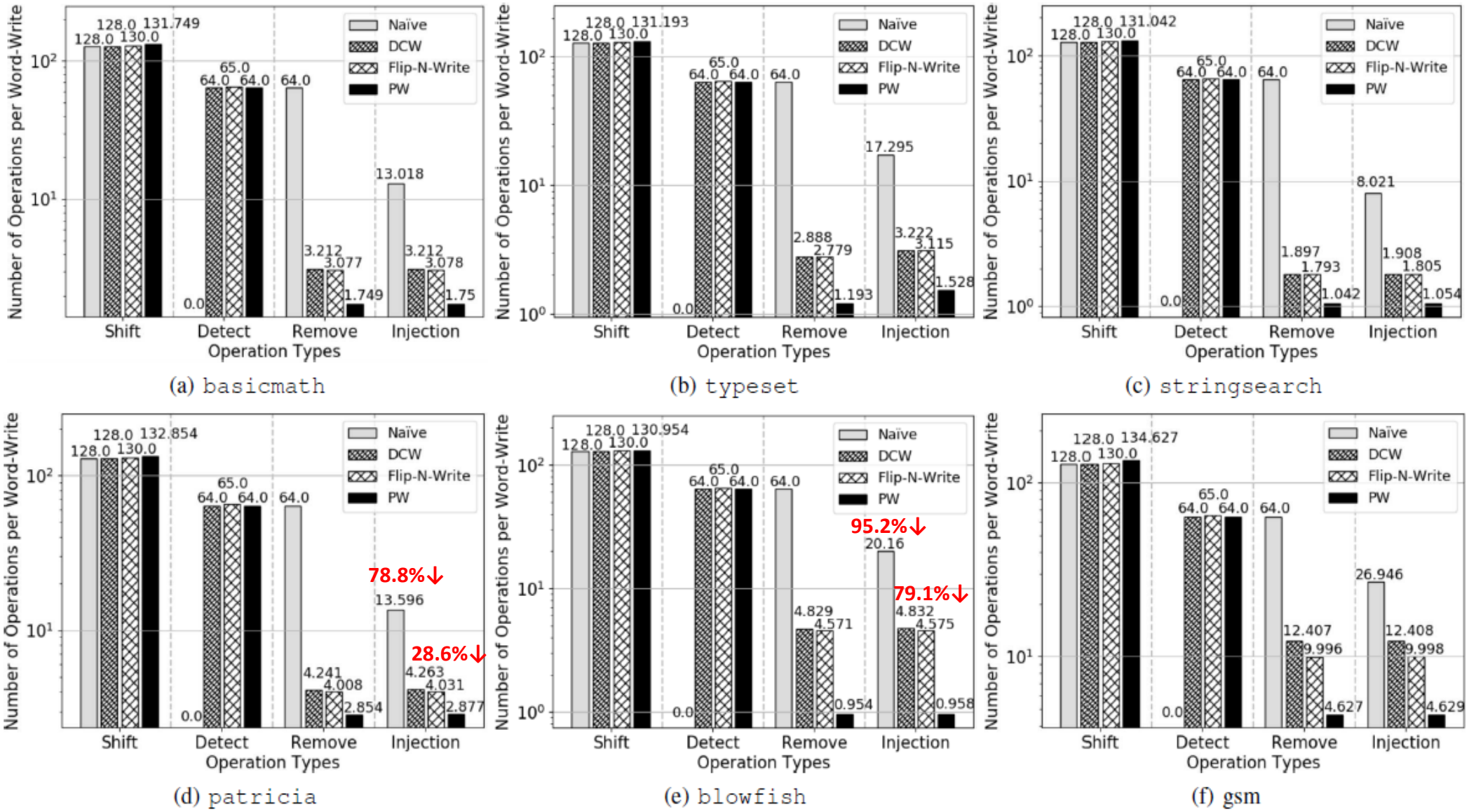


Fig. 5. The average number of operations per 64-bit-word write introduced by the evaluated write strategies under various MiBench programs.

Summary of Experimental Results

- On average, PW strategy can reduce the number of injection upon writing data words by:

Naïve	DCW	Flip-N-Write
86%	54%	49%

- By applying the real value, PW can improve by at most:

	Naïve	DCW & Flip-N-Write
Latency	35%	8.5%
Energy	85%	21.6%

References

- https://www.science.org/doi/pdf/10.1126/science.1145799
- https://dl.acm.org/doi/abs/10.1145/2744769.2744800
- https://dl.acm.org/doi/abs/10.1145/3333336
- https://ieeexplore.ieee.org/abstract/document/8107669
- https://ieeexplore.ieee.org/abstract/document/7442797
- https://ieeexplore.ieee.org/abstract/document/7304358
- https://dl.acm.org/doi/pdf/10.1145/2333660.2333707
- https://ieeexplore.ieee.org/abstract/document/9218642
- https://ieeexplore.ieee.org/abstract/document/9211559
- https://www.nature.com/articles/srep09400
- https://journals.aps.org/prapplied/abstract/10.1103/PhysRevApplied.12.064053

國立臺灣大學
National Taiwan University



Future Storages – Glass Storage and DNA Storages

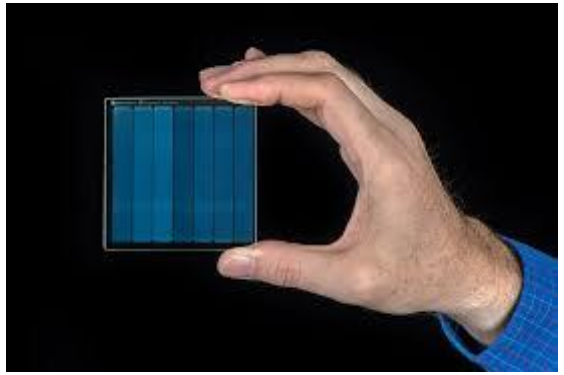
國立臺灣大學
National Taiwan University

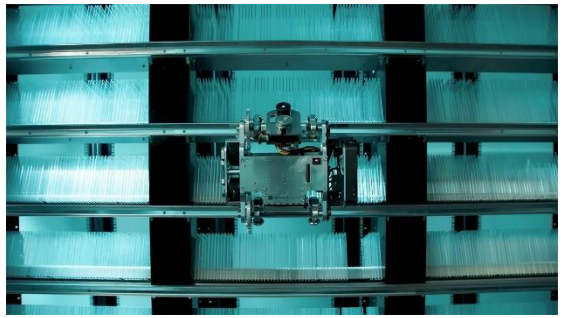


Introduction to Glass Storage System

Introduction

- **HDD (Hard Disk Drive)**
- **Tape (Magnetic Storage)**
- **Optical Storage (Blu-ray, etc.)**
  - Capacity limited to < 1 TB per disc.
  - Multi-layer structure but aging issues remain.
  - Lifespan ~100 years under ideal conditions.
- **DNA Storage**
  - Extremely high theoretical density.
  - Current challenges: high synthesis cost, slow write/read speed, low throughput.

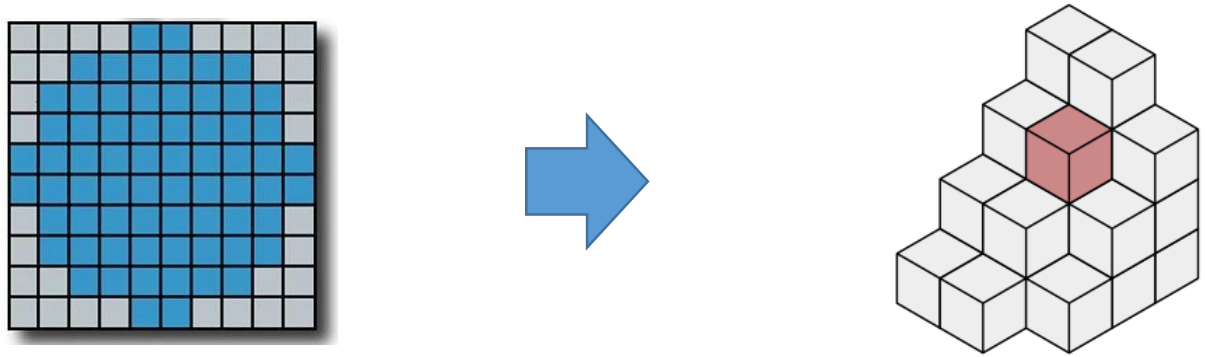




Property	HDD	Tape	DNA	**Glass**
Lifespan	5–10 yr	30 yr	1000 yr	**1000+ yr**
Density	Medium	High	Very high	**High (3D)**
Access time	ms	min	hr	sec–min
Cost	Low	Very low	High	**Low (mass-producible)**

https://newatlas.com/computers/microsoft-project-silica-glass-data-storage-10000-years/

Glass Storage – Voxel (體素)



Pixel　　Voxel

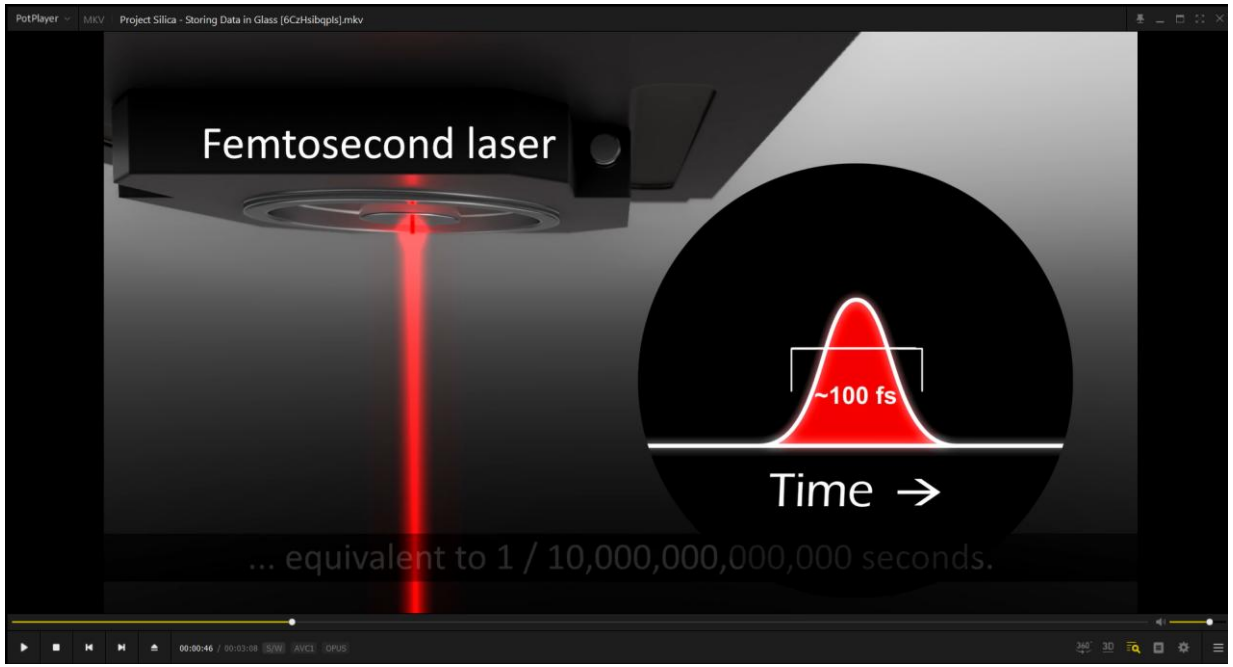
Glass: A New Media for a New Era?:

"When the beam from a femtosecond (10^{-15}) laser (飛秒雷射) is focused inside a block of fused silica (熔融石英), a permanent **small 3D nanostructure** (which we will call a "**voxel**") forms in the **silica**."

System Architecture

- Write head: Femtosecond (10^{-15}s) laser system
- Read head: Polarization-sensitive imaging
- Control unit: Alignment, focusing, and layer control
- Storage library: 3D glass slabs for large-scale data

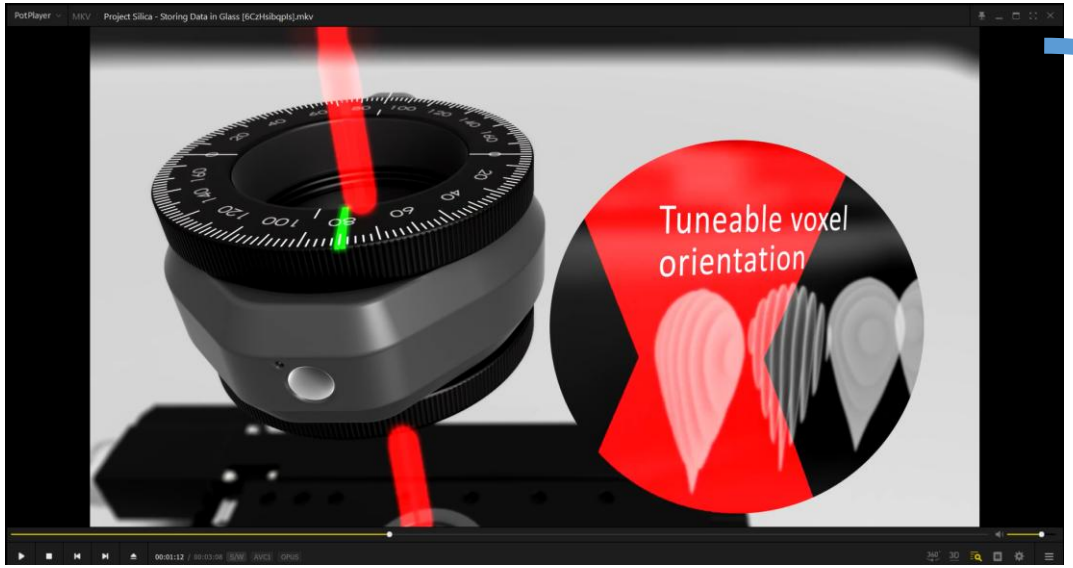
Storing Bits in Glass (Write) (1)

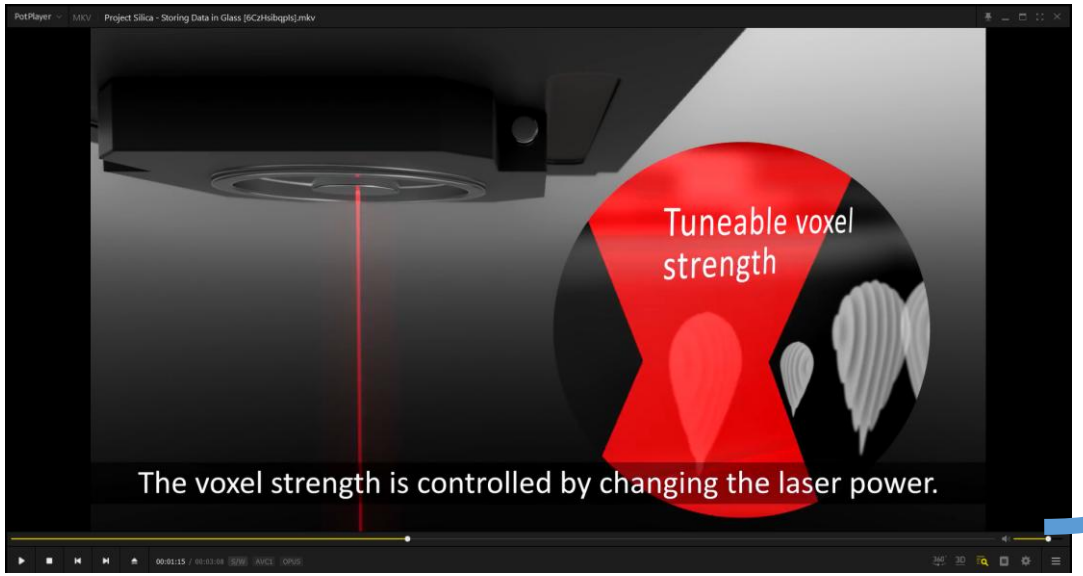


Reference: Project Silica - Storing Data in Glass - Microsoft Research

Storing Bits in Glass (Write) (2)

voxel



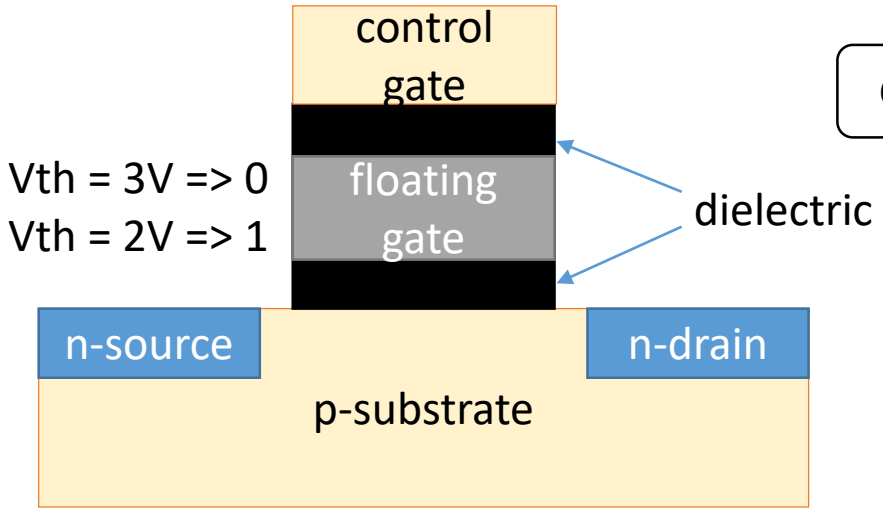


2d

Reference: Project Silica - Storing Data in Glass - Microsoft Research

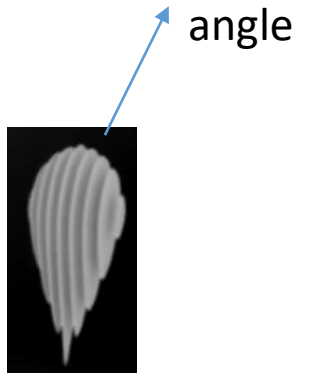
Storing Bits in Glass (Write) (3)

Feature	NAND Flash	Glass Storage
Physical medium	Silicon	Fused silica (glass)
Mechanism	Electrons in floating gate	Laser-written 3D voxels
Bit definition	Charge level (0/1)	**Polarization/voxel orientation**
Durability	10^4–10^5 write cycles	>1000 years, no degradation
Suitable for	Active memory / SSD	Long-term cold data archive

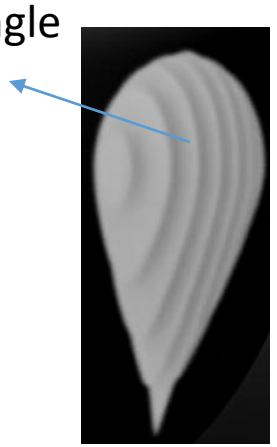


NAND Flash Structure

Glass Structure



Angel = 30° & Size = 0.4 um
=> 00001



Angel = 330° & Size = 0.8 um
=> 01110

Decoding Bits in Glass (Read)

Glass: A New Media for a New Era?:

"**Each voxel** has a property called ***form birefringence (形式雙折射)***, **nanostructure** has different physical properties than the surrounding **silica** material."

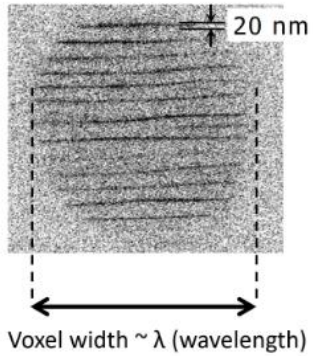
Light with a different polarization
+
Voxel exhibits a different refractive index

→

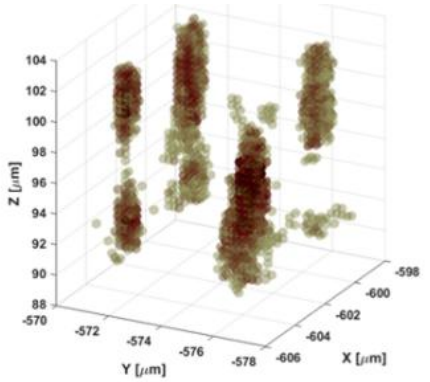
Retardance (延遲性), a shift of several nanometers
&
Change in the polarization angle of incoming light

→

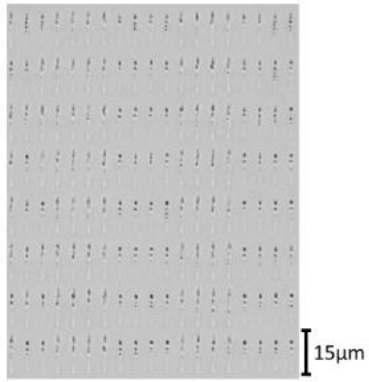
Bits



(a) Single voxel (top view)



(b) 3D reconstruction of four measured voxels

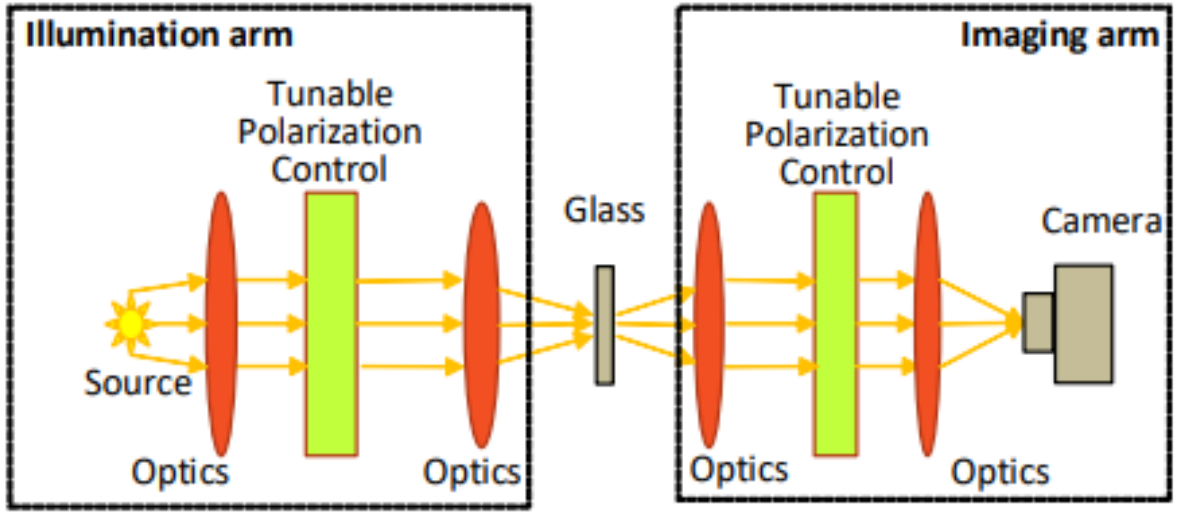


(c) Side view (8 layers)

Figure 1: Voxels in fused silica

Reading Data from Glass (Optical Setup)

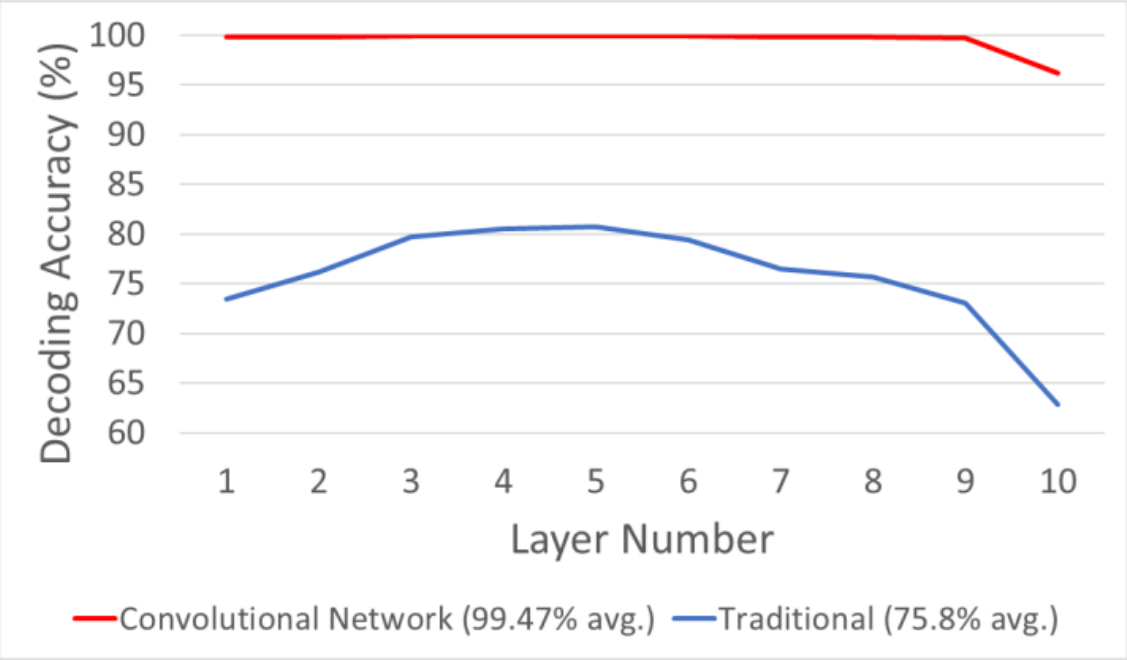
- Data retrieved using polarized light microscopy.
- Light source + tunable polarization optics → illuminate voxel.
- **Camera + analyzer capture birefringence (雙折射) response** (retardance, angle shift).
- Each voxel's optical response → bit values.



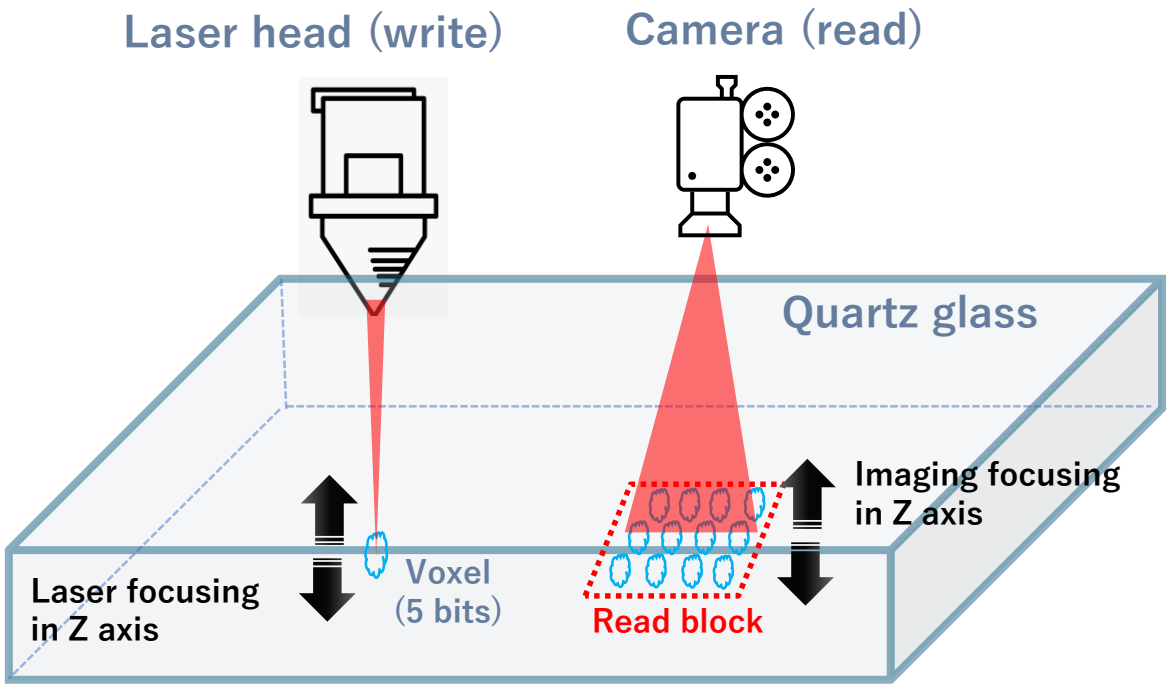
Read head hardware

Machine Learning-Based Reading

- Traditional thresholding → Sensitive to noise (physics-based approach)
  - Laser variation
  - Environmental noise
  - Imperfect optics
- CNN-based decoding learns voxel patterns directly
  - Subtle variations in voxel **shape, size, and orientation** that correspond to specific bit values
  - Achieves >99% decoding accuracy

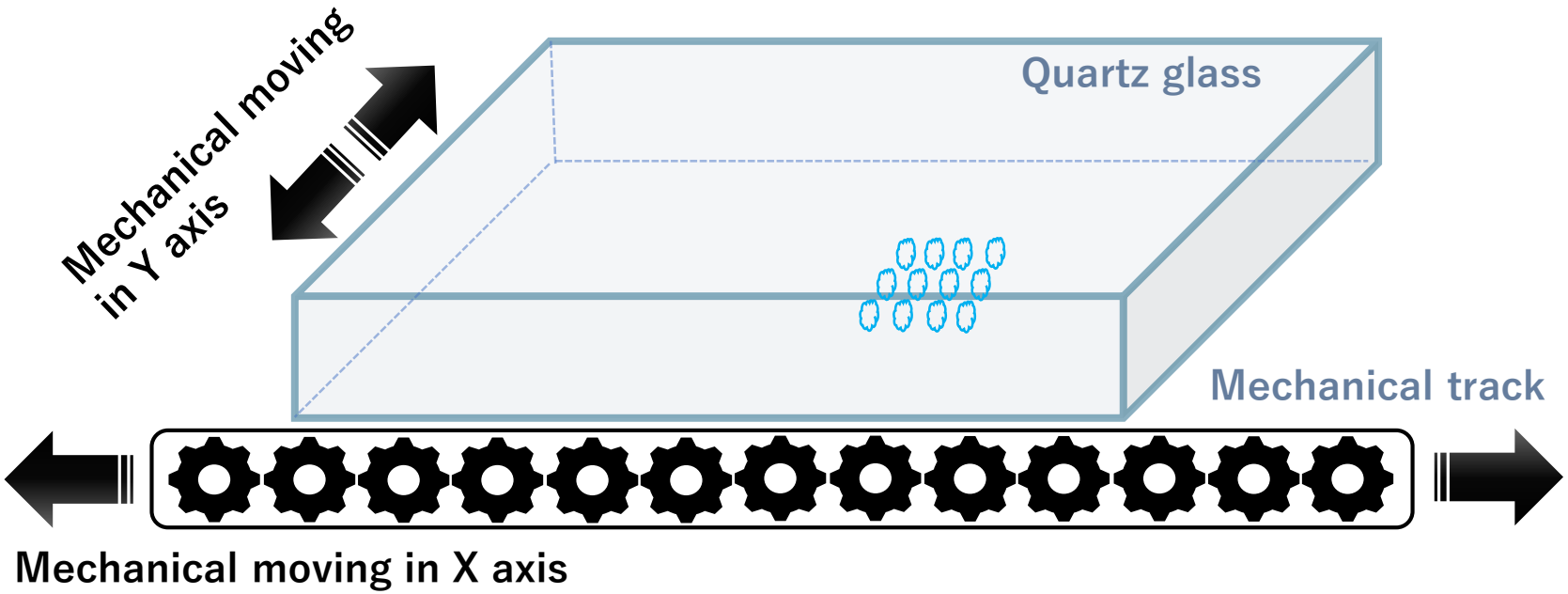


Storing Bits in Glass (1)



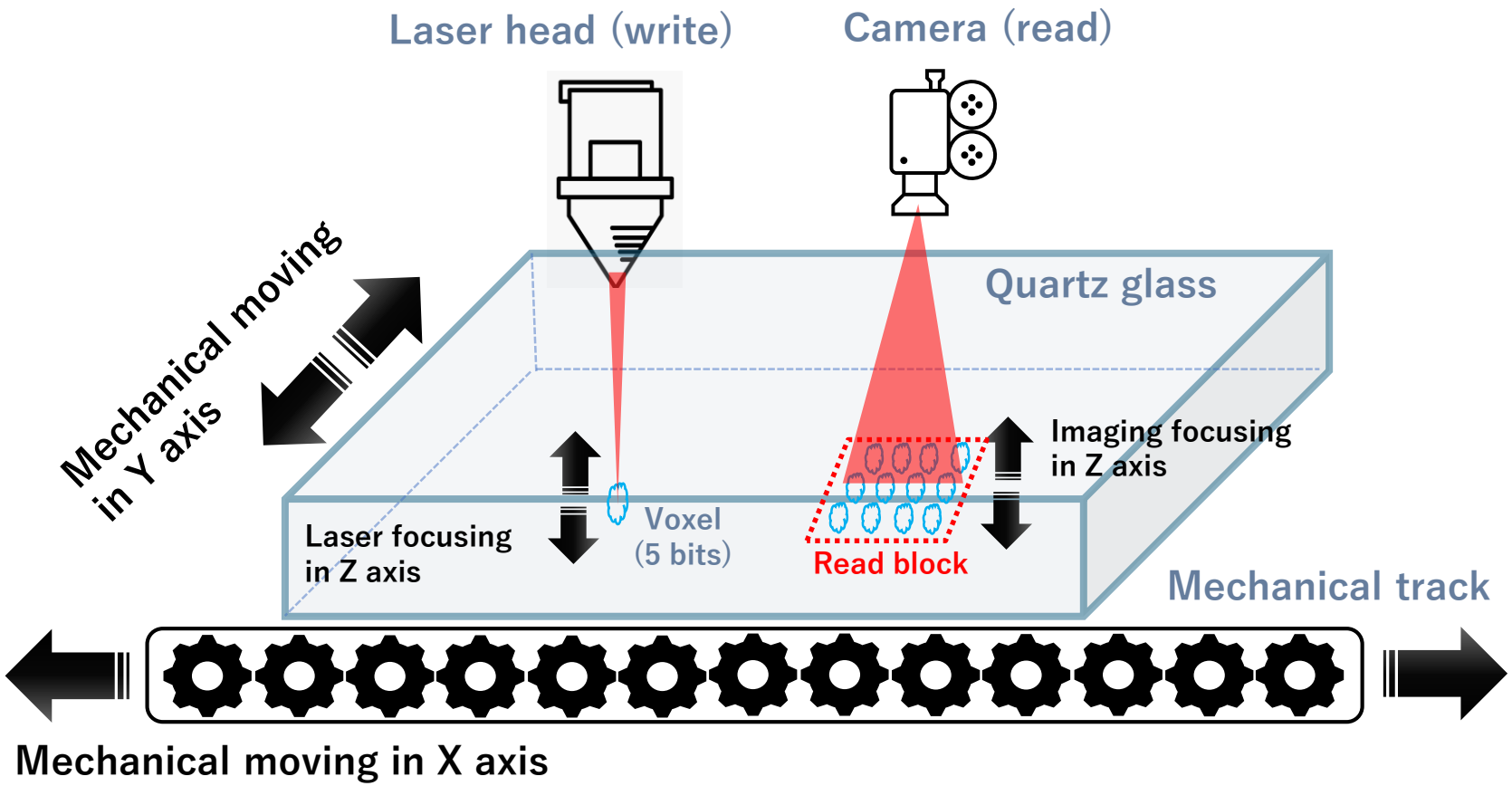
- Architecture of glass storage: separation of write and read devices.

Storing Bits in Glass (2)



- Architecture of glass storage: separation of write and read devices.

Storing Bits in Glass (3)

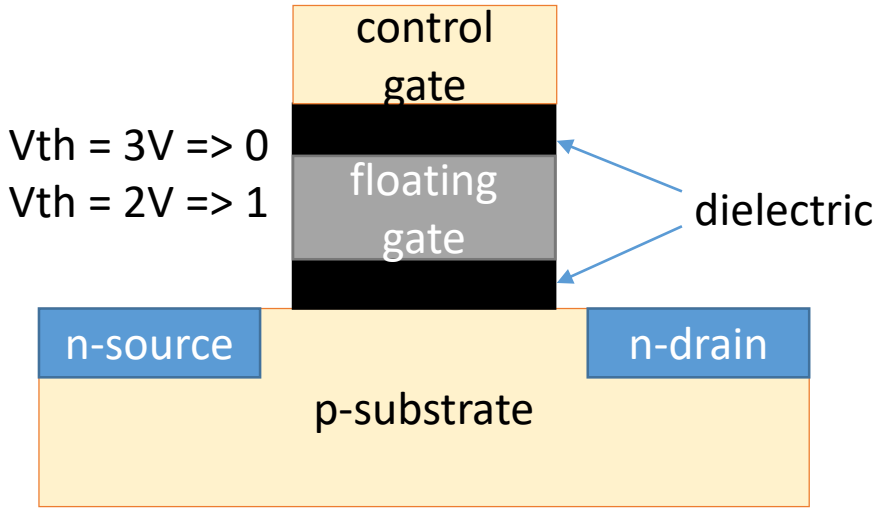


- Architecture of glass storage: separation of write and read devices.

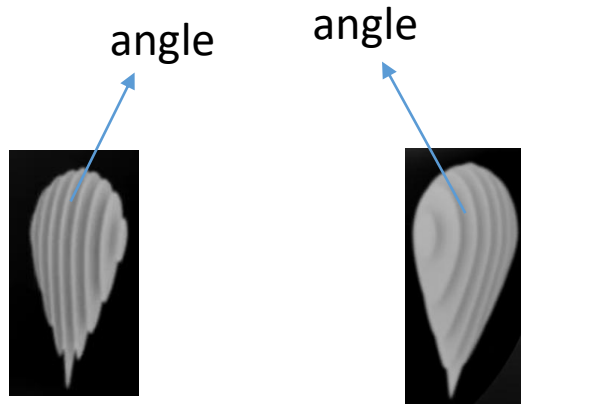
Storing Bits – Flash vs. Glass

- NAND Flash structure represents the values 0 and 1 based on the **number of electrons on the floating gate.**
- Glass represents 0 and 1 based on **the result of different retardance and polarization angles.**

NAND Flash Structure



Glass Structure



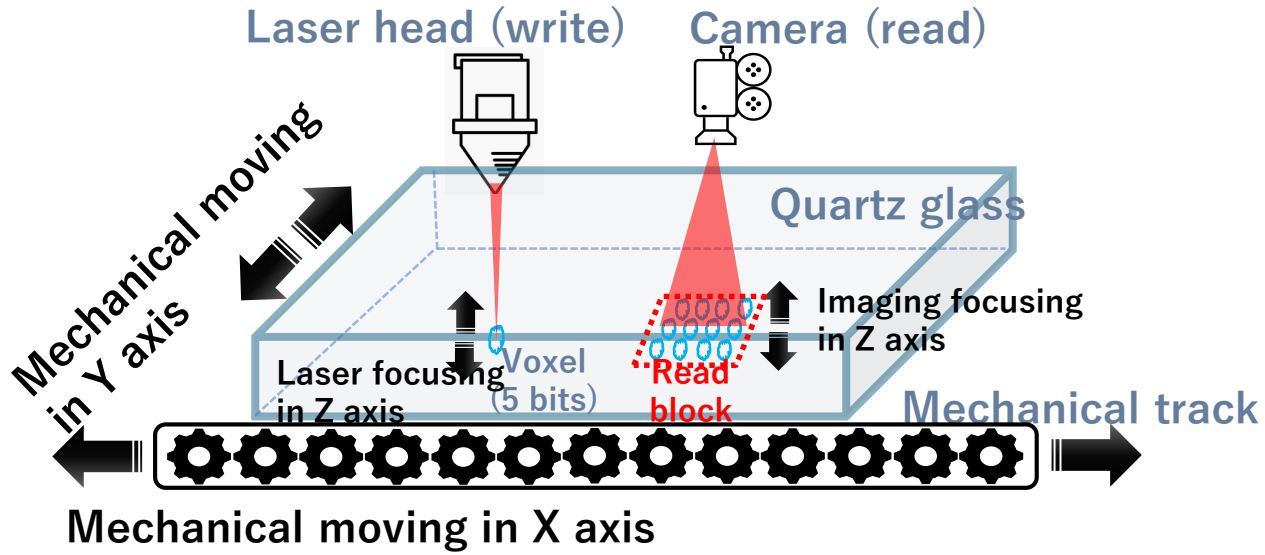
These voxels serve as the basic units of data, encoding bits through properties such as birefringence (利用兩束光不同偏振方向造成相位差), retardance (利用雙折射材料中兩束光的不同度造成延遲量), and polarization angle (光波電場振盪方向與某個參考方向的夾角).

Glass Storage: Write-Once-Read-Many Scenario

- Previous studies have demonstrated **rewrite** capabilities in quartz glass [3], [5].
- However, the current primary application remains in archival storage for infrequently accessed data, such as that used in cloud-scale archival systems [4].
- Due to the high time cost of rewriting, glass storage today is best suited for **write-once-read-many (WORM)** scenarios [2].

Motivation (1)

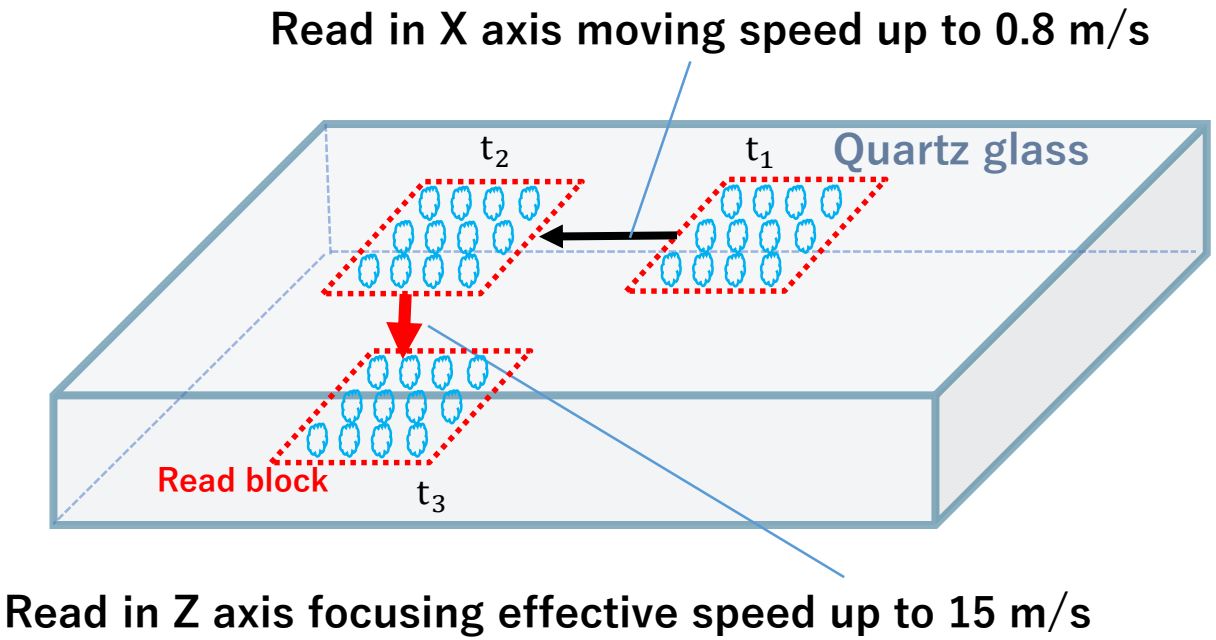
- These legacy algorithms are primarily designed for rotational media like HDDs, where seek latency is a function of **angular distance** rather than the physical inertia of **linear mechanical stages**.
- They overlook the directional switching penalties and fail to optimize for **continuous motion factor (X->Y)**.



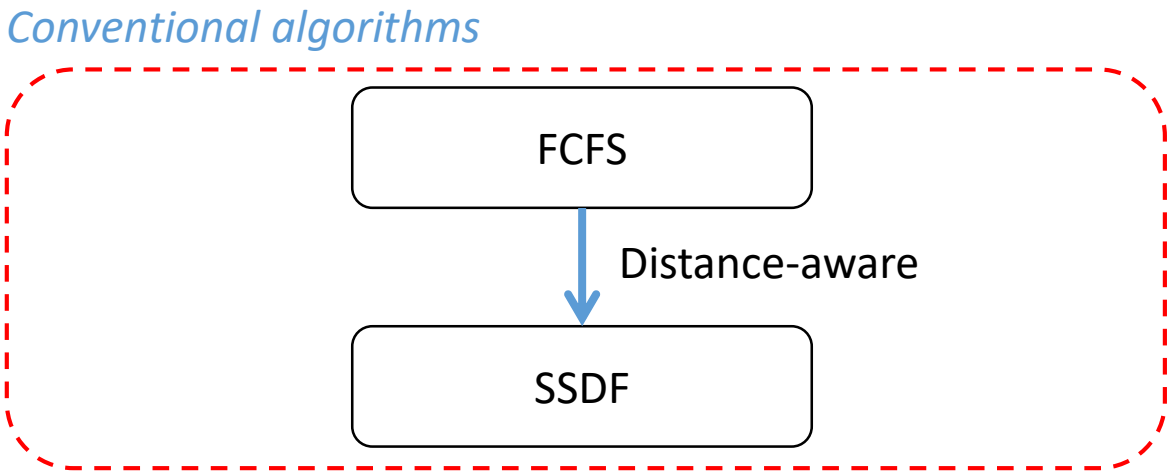
- Architecture of glass storage: separation of write and read devices.

Motivation (2)

- Conventional algorithms can't leverage **asymmetry in axis efficiency (Z-First).**



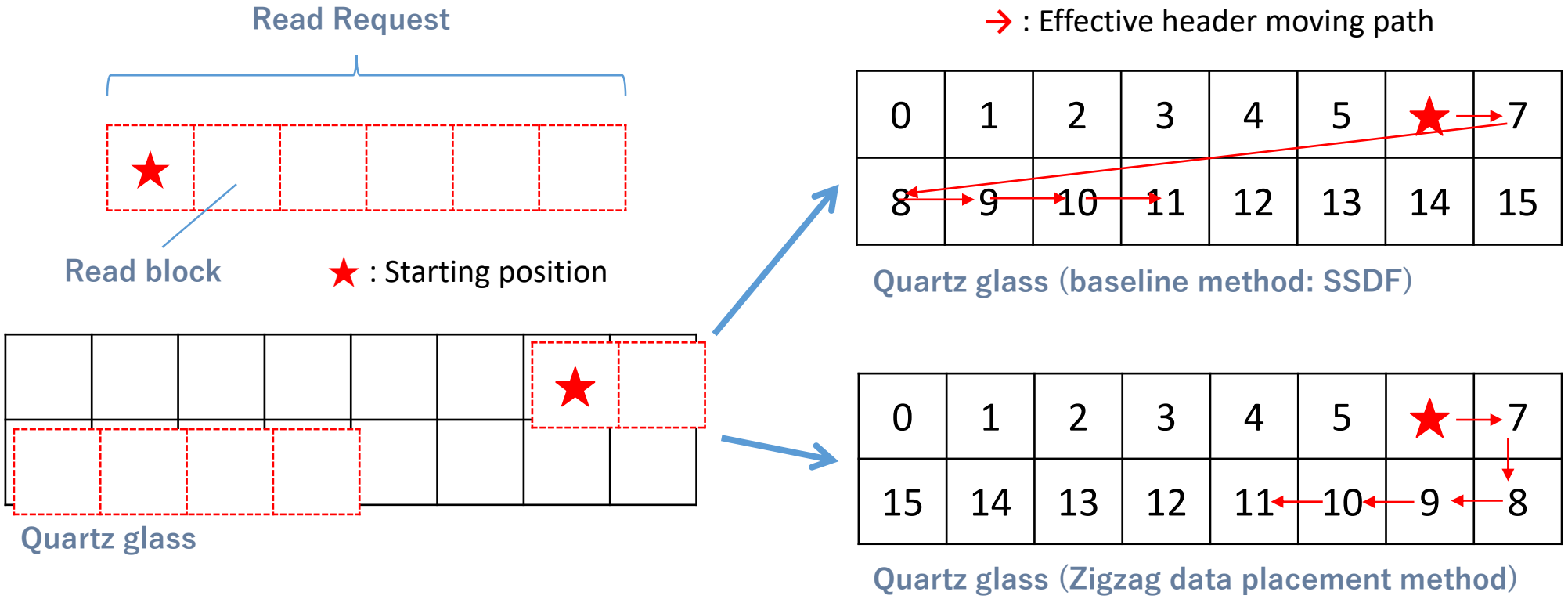
Conventional Algorithms



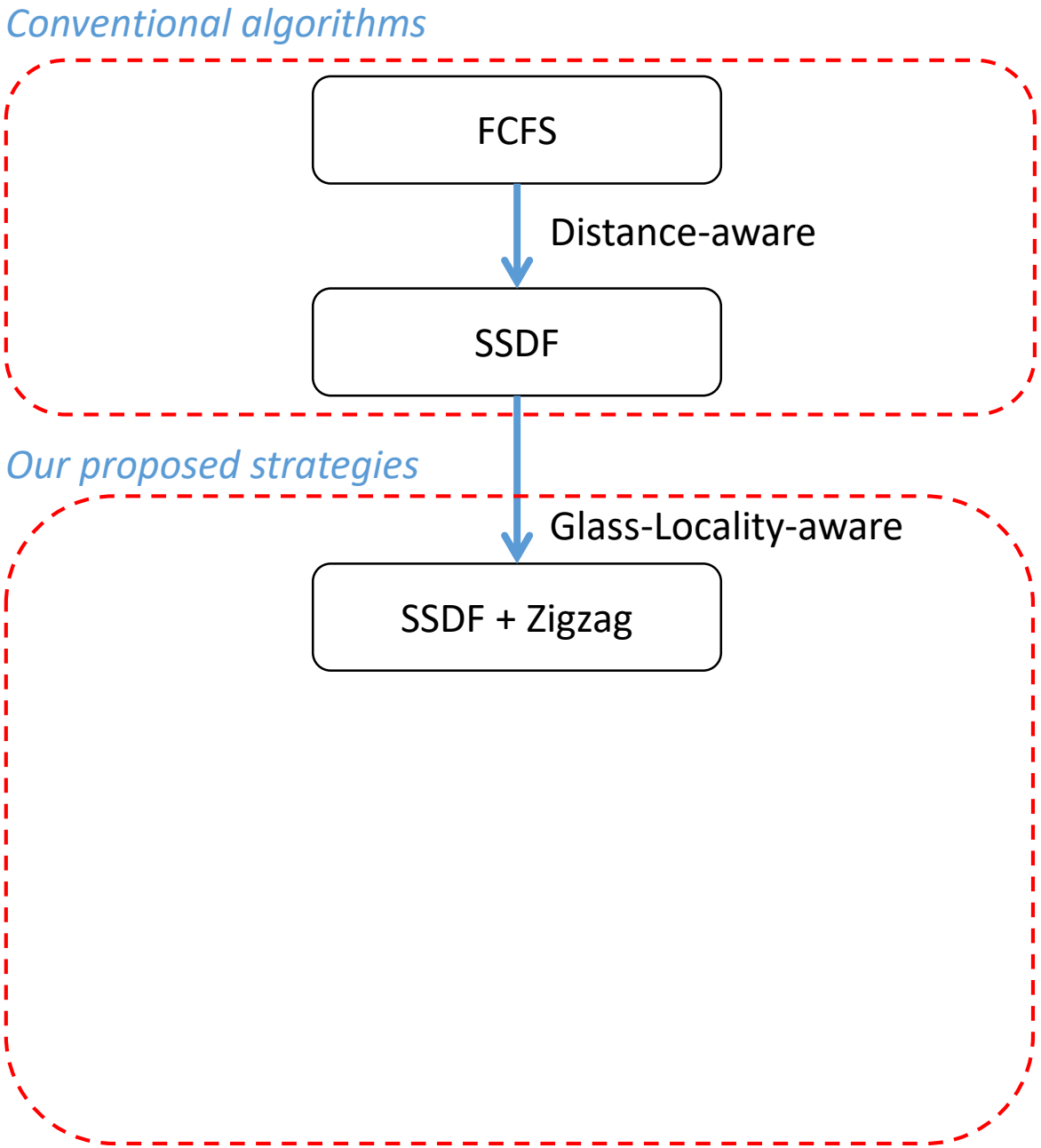
SSDF: Shortest Seek Distance First

Proposed Strategy – Zigzag

- Zigzag data placement improves stage traversal efficiency by minimizing directional changes compared to shortest seek distance first (SSDF).



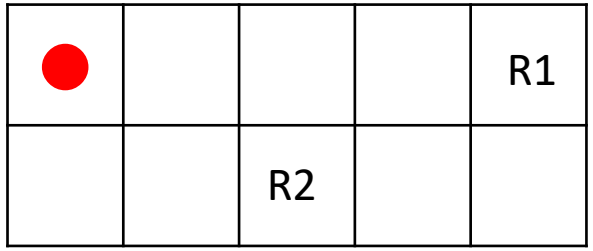
Proposed Strategies



Proposed Strategy – SMTF (Seek Moving Time First)

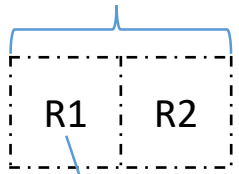
● : Head initial position

→ : Effective head moving path



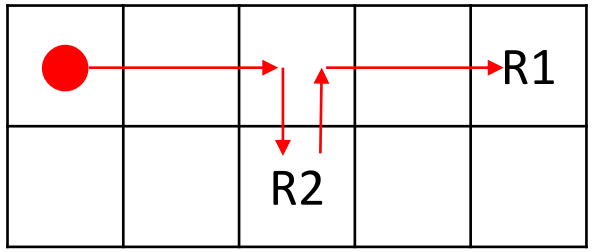
Example of read request

Read Request Queue

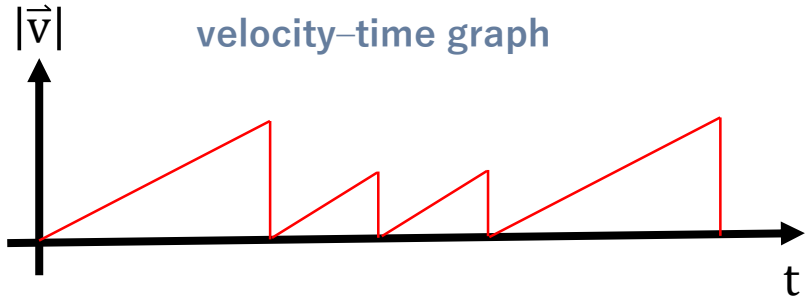


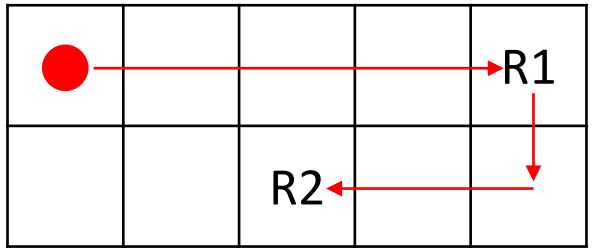
Read request

Comparison between SSDF and SMTF scheduling strategies. SMTF outperforms SSDF by accounting for acceleration and deceleration overheads in mechanical stage movement, resulting in reduced total seek time.

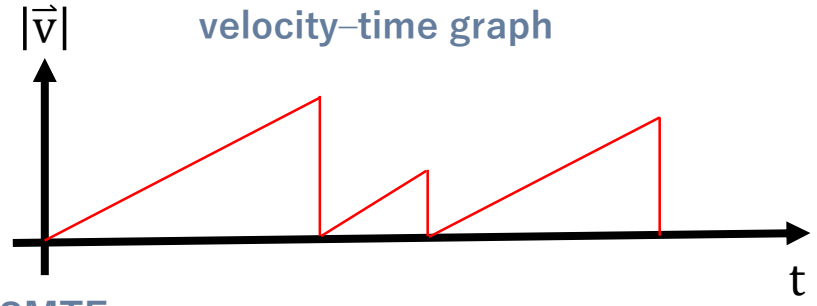


Baseline scheduling: SSDF

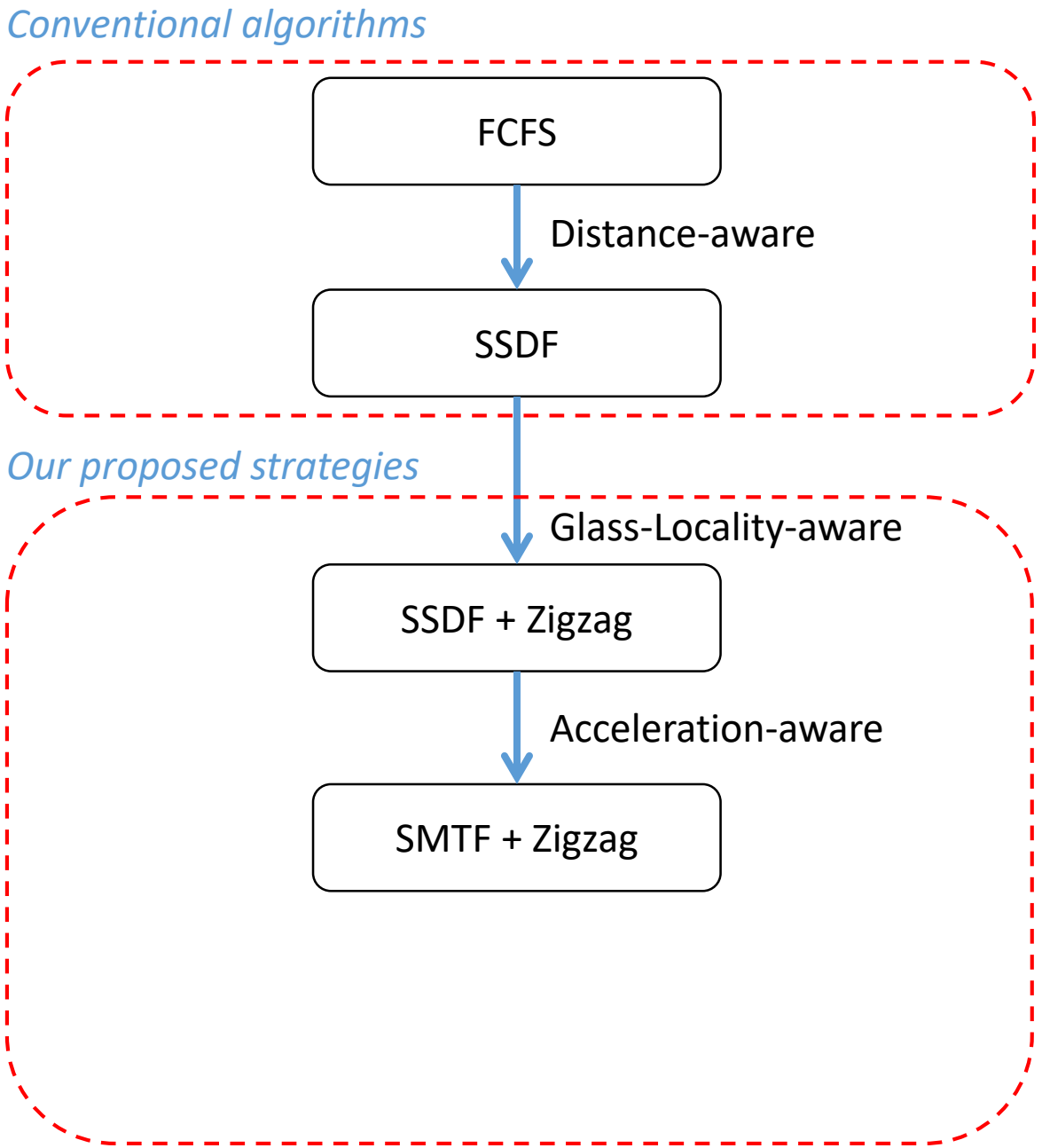




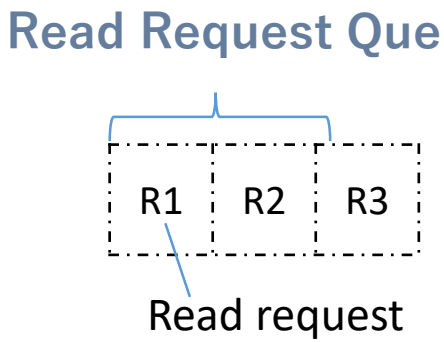
Acceleration-considering scheduling: SMTF

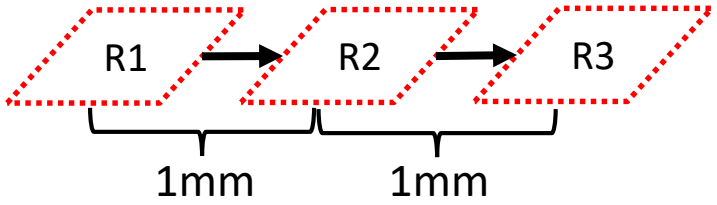


SMTF (Seek Moving Time First)



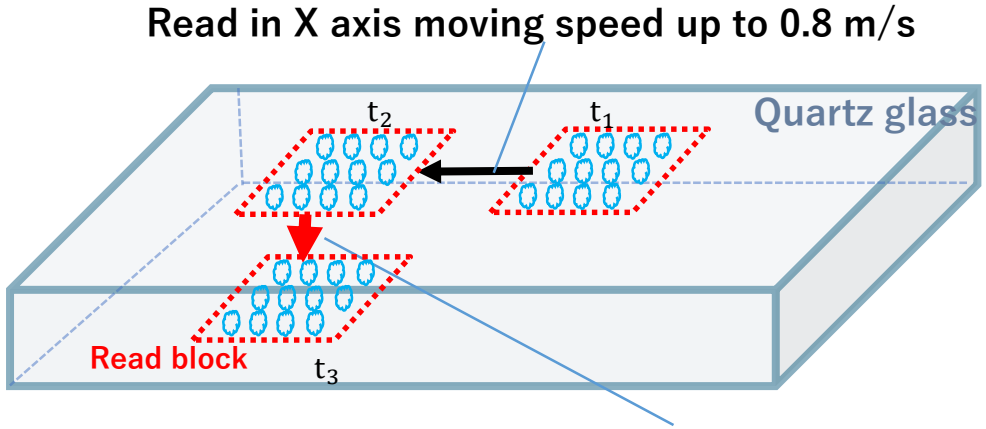
Proposed Strategy – Z-First Data Placement

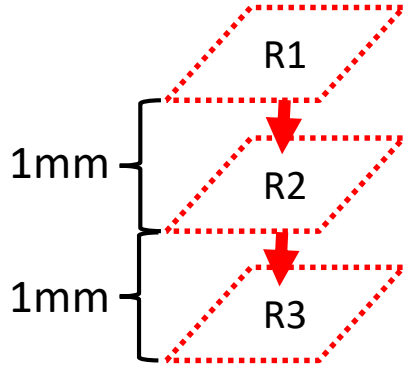




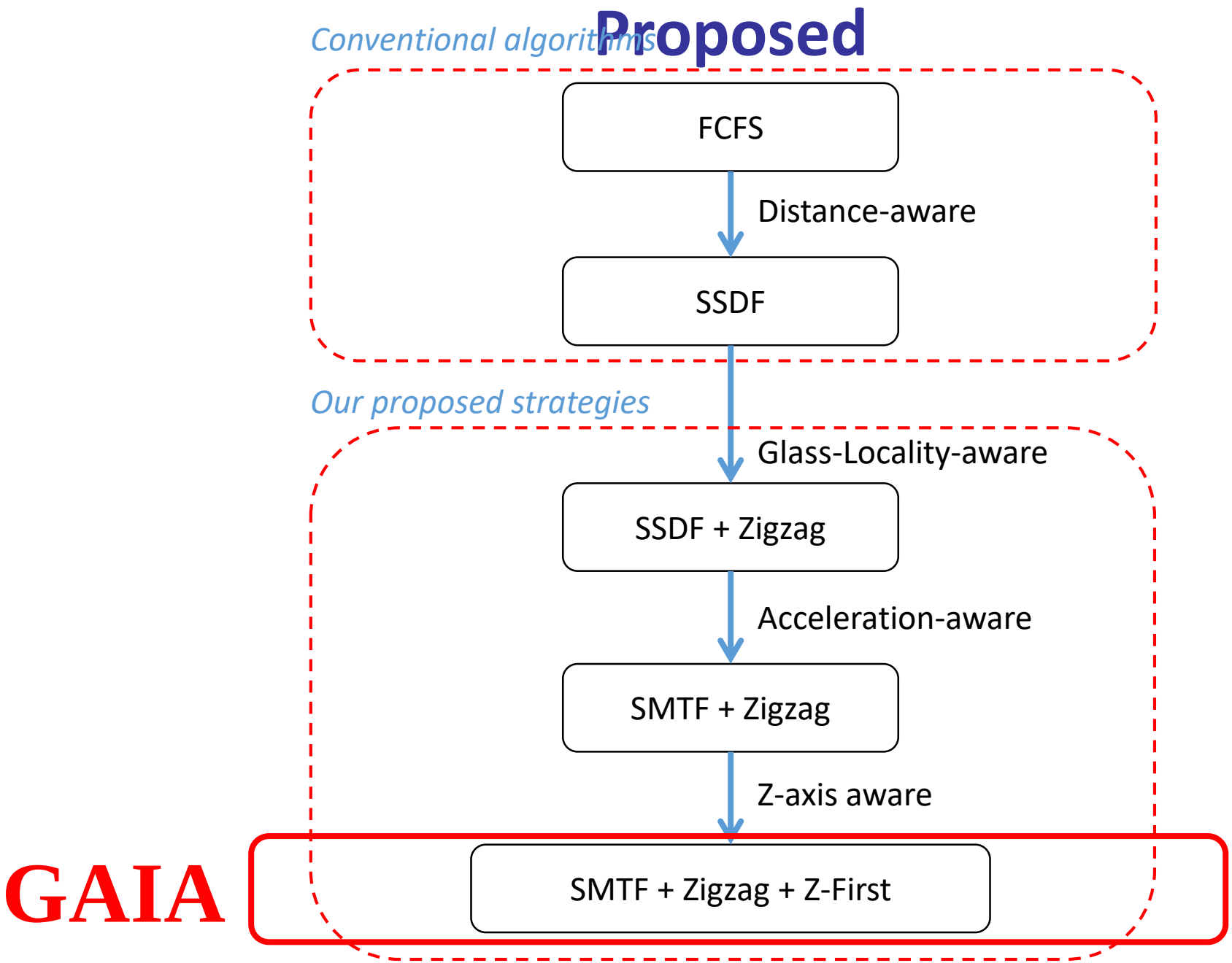
- **3 requests completed needs more than 2.5 ms in X axis.**

$\because 2mm/(0.8m/s) = 2.5\,ms$





- **3 requests completed only needs 200 us in Z axis.**


Proposed
Conventional algorithms
FCFS
Distance-aware
SSDF
Our proposed strategies
Glass-Locality-aware
SSDF + Zigzag
Acceleration-aware
SMTF + Zigzag
Z-axis aware
SMTF + Zigzag + Z-First
GAIA

Performance Metrics and Evaluation Setup

- In our evaluation, total **latency** is used as the primary performance metric.
- To simulate realistic workloads, we adopt an I/O trace dataset from Fujitsu.
- This metric captures both mechanical movement delays and optical focal adjustment time.

Stage Movement Simulation

- We simulate the stage movement by treating it as the motion of a read/write head, since all movement is relative and the speed is the same.
- HDD seek time simulation formula: ***Extract and infer quickly: Obtaining sector geometry of modern hard disk drives [TOS'2010]*** **[6].**
- The simulation of the read/write head follows two situations:
  - When the displacement d is less than a threshold m, the head undergoes uniformly accelerated motion.
  - When the displacement d is greater than or equal to the threshold m, the head slides at a constant velocity.

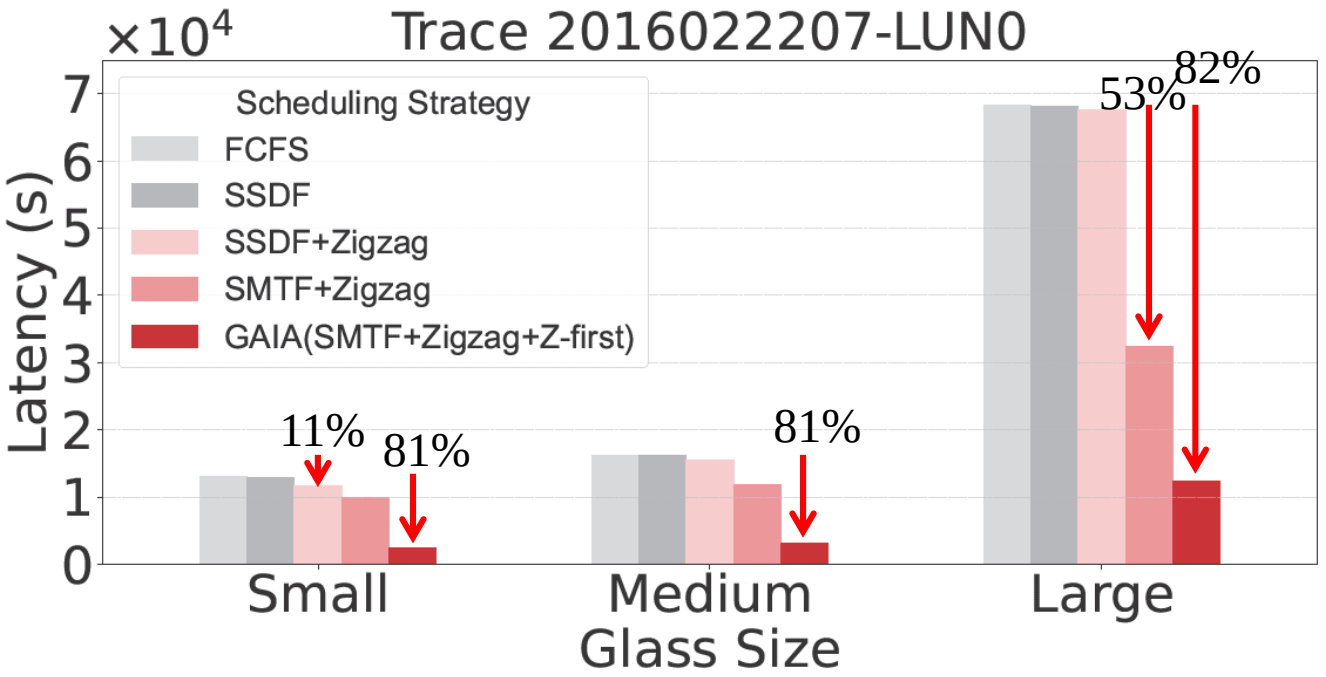
$$f_{seek}(d) = \begin{cases} p + q\sqrt{d}\ if\ (d < m) \\ r + sd\ if\ (d \geq m) \end{cases}$$

$$d = \frac{1}{2} \times (at^2):\ t = O(\sqrt{d})$$

Evaluation Results (1)

- SSDF + Zigzag strategy achieves an improvement of 11% over FCFS when the glass size is small.
- SMTF + Zigzag strategy reduces total latency by 53% more than the SSDF + Zigzag strategy for large glass sizes.
- GAIA yields latency reductions across all glass sizes, achieving improvements 81%-82%.

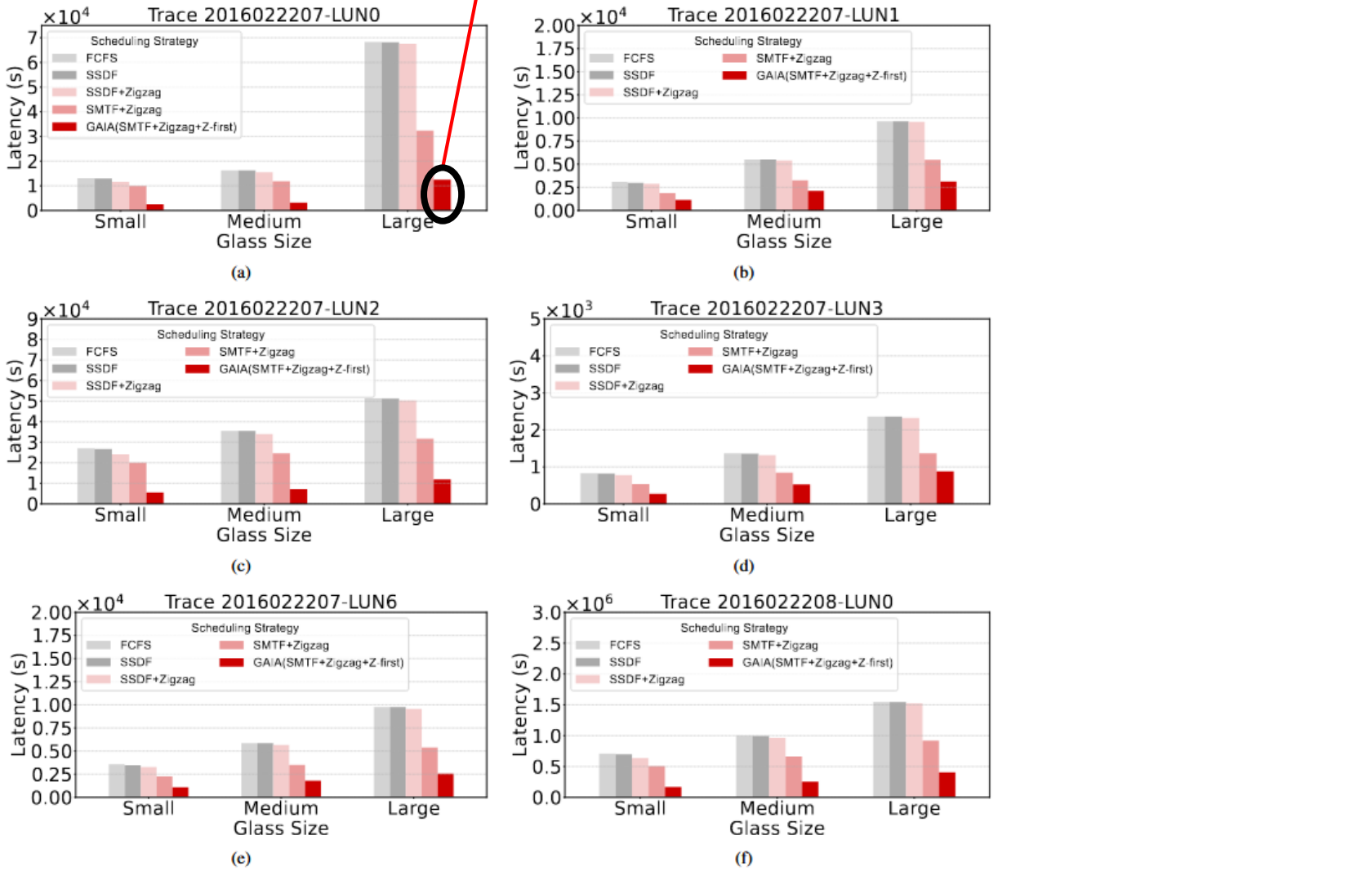
- Evaluation on total requests latency time for different glass sizes.



Evaluation Results (2)

- Evaluation on total requests latency time for different glass sizes and traces.

18% of FCFS (= reducing 82% latency)



Reference

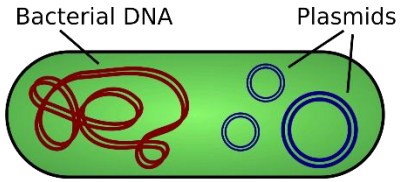
- **Glass: A New Media for a New Era?**
- https://www.usenix.org/conference/hotstorage18/presentation/anderson

國立臺灣大學
National Taiwan University



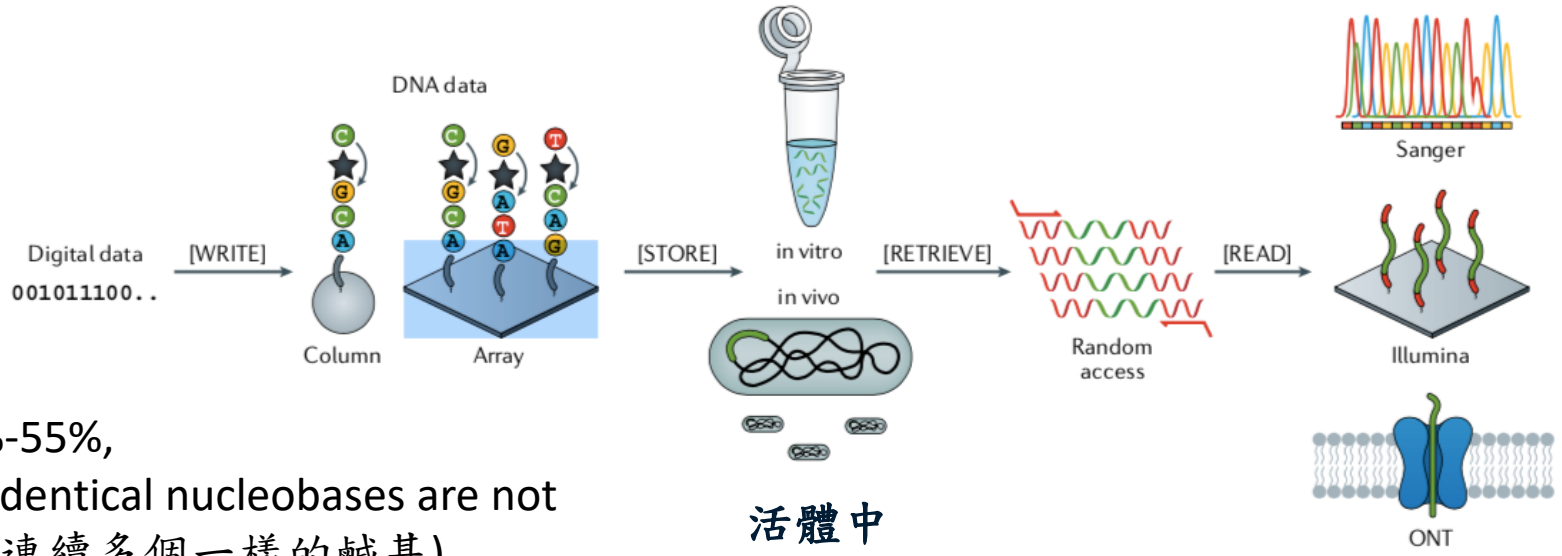
Introduction to DNA Storage System

Introduction



- **Steps**
  - **WRITE: Map binary bits (0/1) to DNA bases (A/T/C/G).**
    - **Synthesize short DNA oligos (DNA寡核苷酸) according to the encoded sequences.**
  - **STORE:**
    - **Preserve DNA in vitro (tubes, dried spots, silica, etc.) or in vivo (inside living cells as plasmids/genomic inserts).**
  - **RETRIEVE (Random access)**
    - **Use barcodes/primers to selectively amplify or pull the specific strands that contain the target file/data chunk.**
  - **READ (Sequencing)**
    - **Sequence the retrieved DNA with platforms such as Sanger, Illumina, or Oxford Nanopore (ONT).**
  - **DECODE**
    - **Convert sequencing reads back to bits.**

試管或培養皿中



活體中

Sanger sequencing

Next Generation Seq. (NGS)

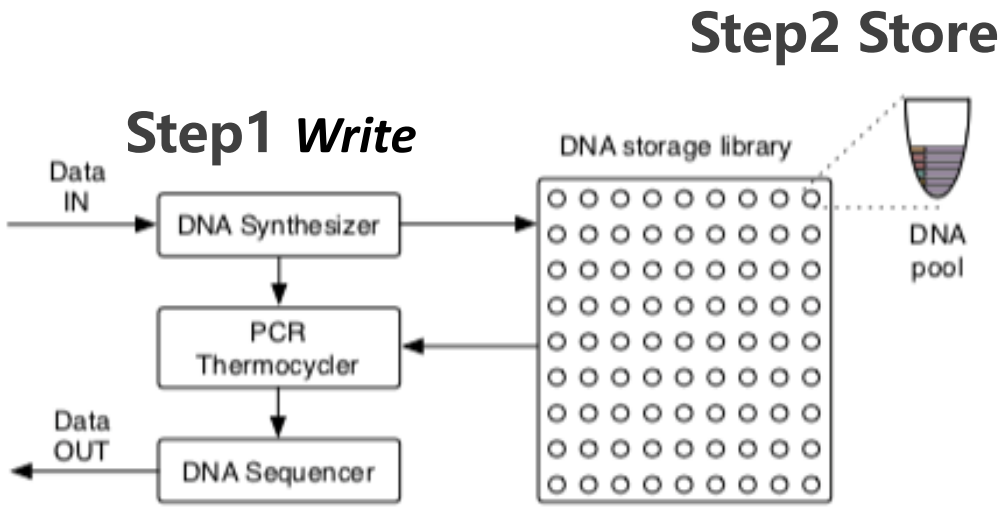
Constraint:
(1) GC ratio: 45%-55%,
(2) Consecutive identical nucleobases are not allowed (不可以連續多個一樣的鹼基)

DNA Data Storage: Write & Store Process

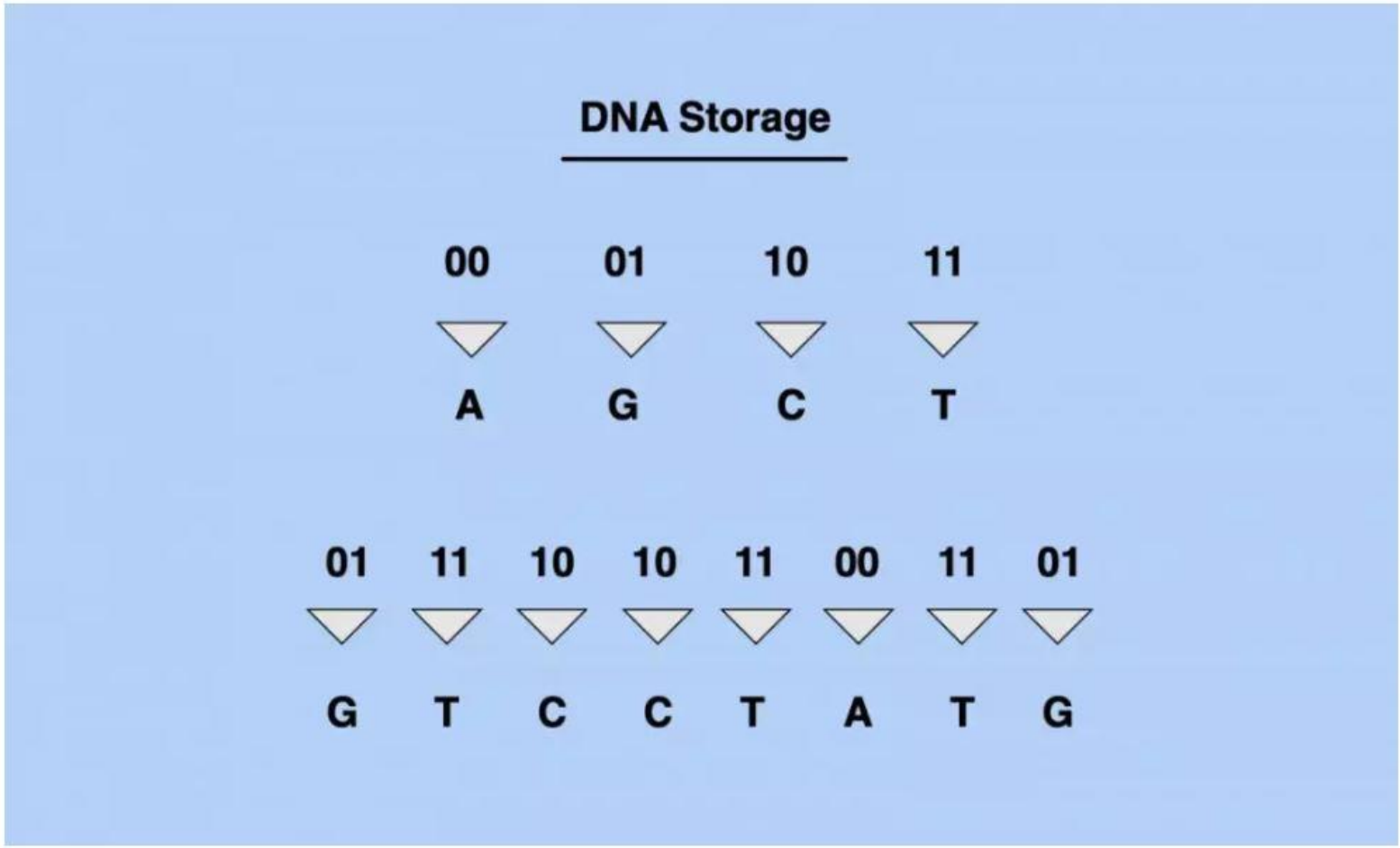
- **WRITE (Encoding & Synthesis)**
  - Convert binary data (0/1) → DNA bases (A/T/C/G)
  - Use **DNA synthesizer** to produce short DNA oligos
  - Each strand encodes a small fragment of digital data
  - Correspond to the *Write* stage in overall pipeline
- **STORE (DNA Storage Library)**
  - Collect synthesized DNA into **storage wells / DNA pool**
  - Represented as a **DNA storage library** for data archiving
  - Enables long-term preservation and compact density
- **Supporting Components**
  - **PCR Thermocycler**: used later to amplify or access specific strands
  - **DNA Sequencer**: used in the next step to read data back

Oligos or oligonucleotides are short, single stranded or double stranded fragments of DNA or RNA.

PCR stands for **Polymerase Chain Reaction**. It is a laboratory technique used to make millions (or even billions) of copies of a specific segment of DNA. You can think of it as a **molecular photocopier**.

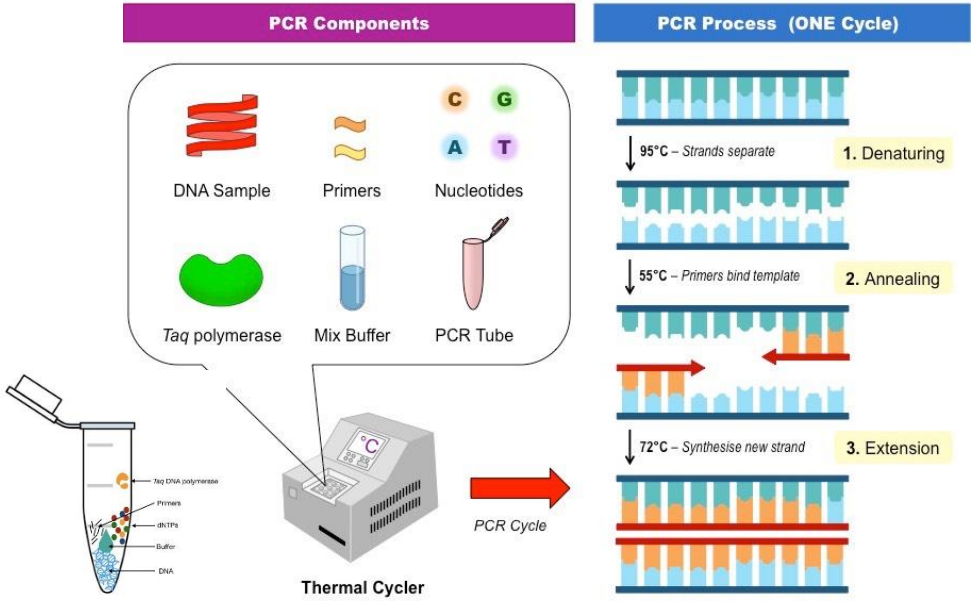


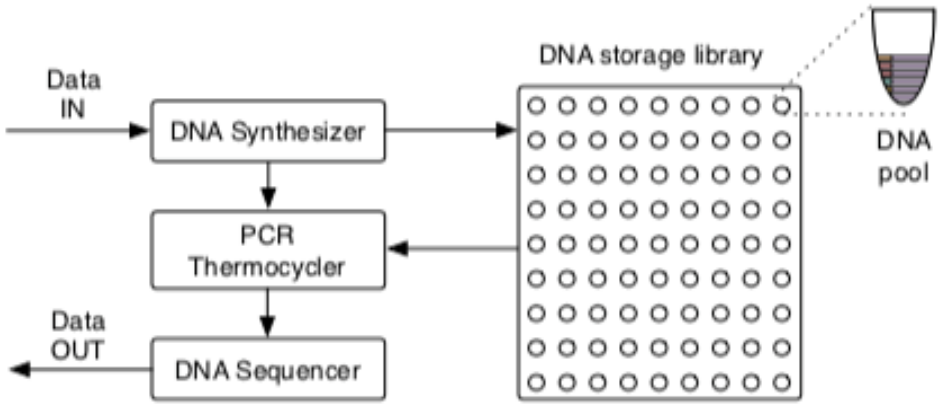
DNA Data Storage: Write & Store Process



DNA Data Storage: Retrieve & Read Process

- **RETRIEVE (Random Access)**
  - Use **primers** to locate specific DNA strands in the storage pool
  - Each data fragment has a **unique primer pair**
  - PCR Thermocycler amplifies only target sequences
  - Enable selective access to desired files (e.g., “find my PPT data”)
- **READ (Sequencing & Decoding)**
  - Amplified DNA is sent to **automated sequencing machines**
  - Sequencers generate **reads** that represent A/T/C/G signals
  - Reads are **decoded back to binary (0/1)** with error correction
  - Digital data is reconstructed and output to computer

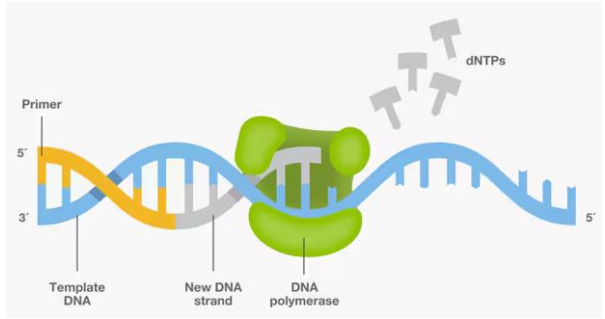


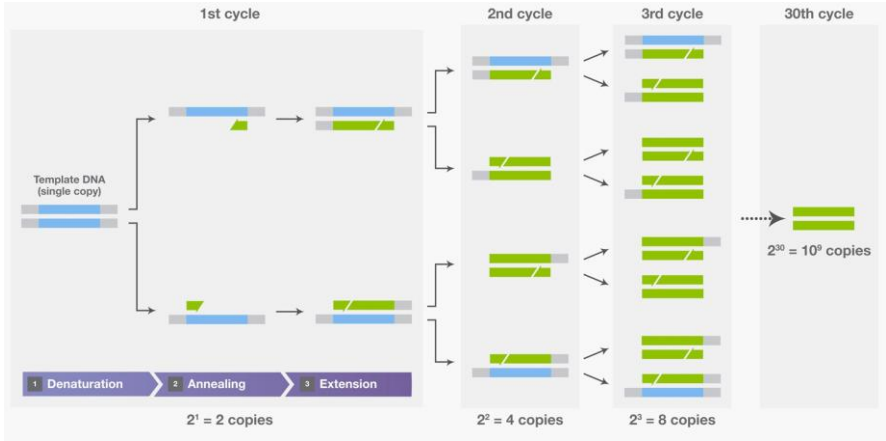


Step4 Read

PCR Machine (thermocycler)

Step3 *Retrieve*



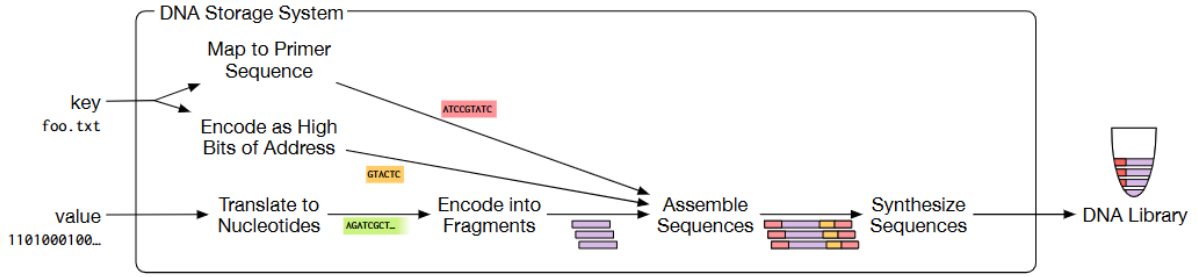


Overview of a DNA Storage as a Key-value Store

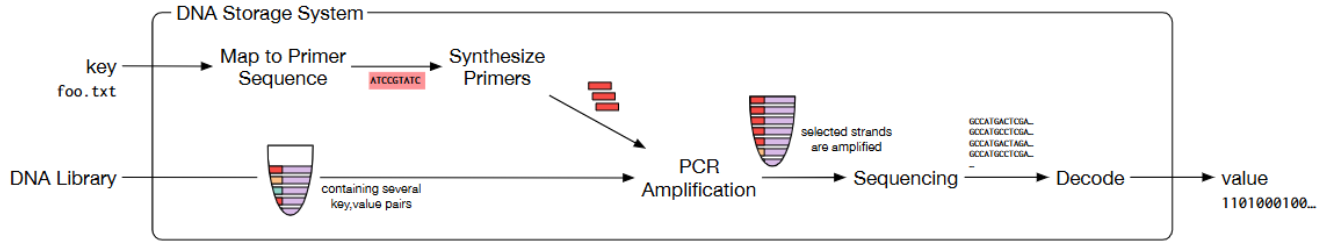
- **Encoding (Data Preparation)**
  - Input = (key, value)
    - Value = data content (e.g., text, image, PPT)
    - Key = address + protection
  - Encode data from **binary → DNA bases (A/T/C/G)**
  - **Assign unique primer pairs** for each data block
  - Translate to DNA fragments
  - Store in DNA library / DNA pool

- **Retrieval (Random Access via PCR)**
  - To read specific data, use **primers** as keys.
  - PCR amplification with the matching primers selectively amplifies target DNA.
  - Enables **random access** from the DNA pool.

- **Sequencing & Decoding (Read)**
  - Amplified DNA is read by **sequencers** (e.g., Illumina, ONT).
  - Output: A/T/C/G reads → decoded back to binary (0/1).
  - Apply **error correction** → reconstruct original digital data.



(a) The write process performs `put(key, value)`, generating a DNA library.

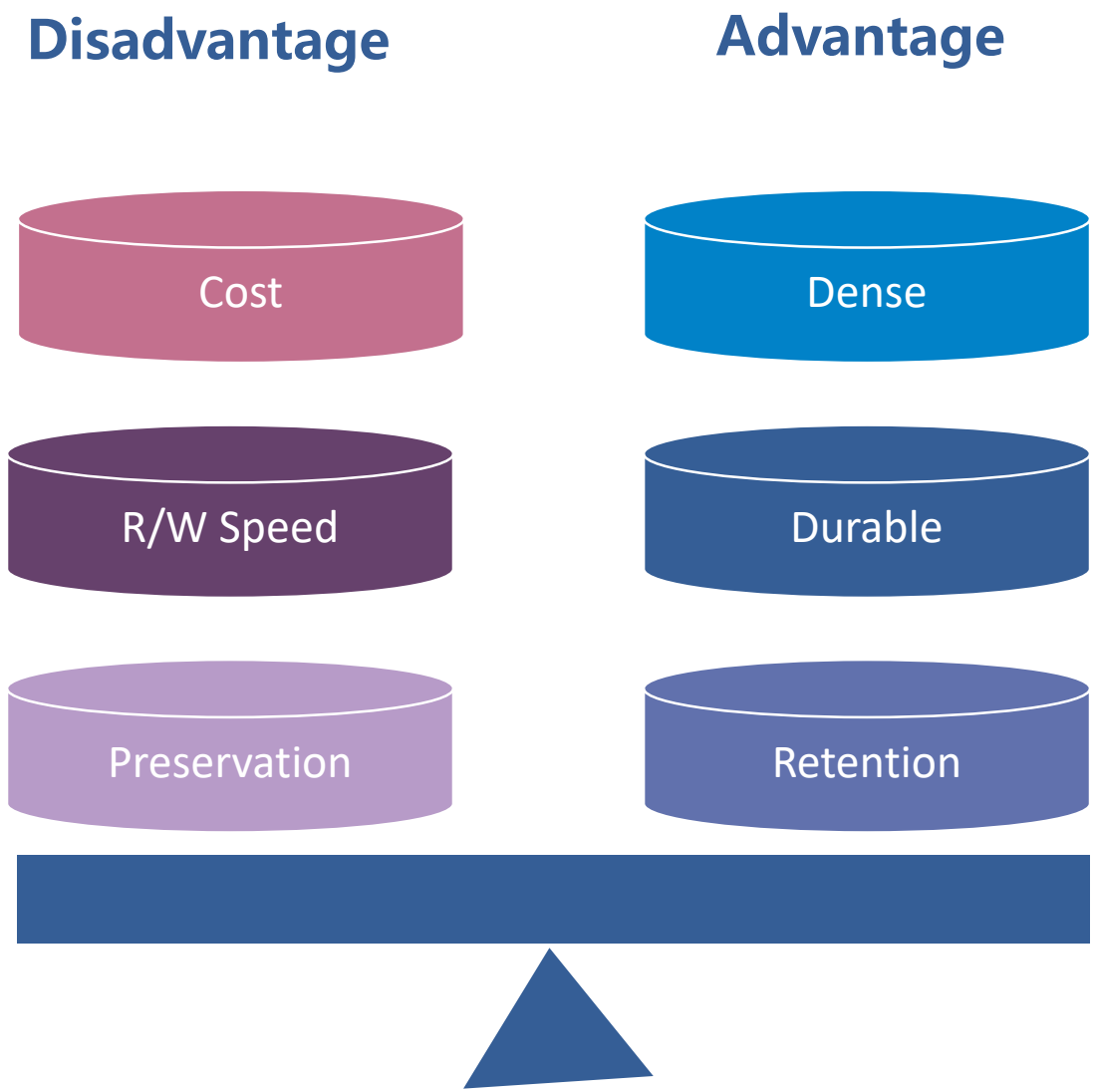


(b) The read process performs `get(key)` on a DNA library, returning the value.

Summary

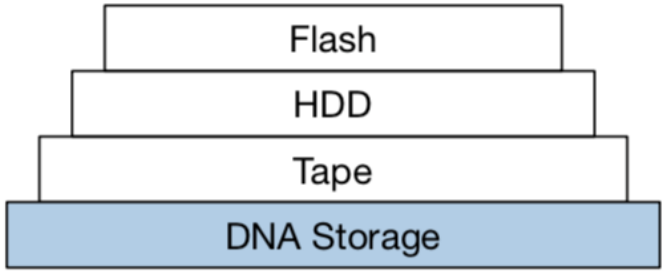
Stage	Action	Key Concept
Encoding	Map 0/1 → A/T/C/G	Add primer, redundancy
Synthesis	DNA writing	Create physical molecules
Storage	Keep DNA	In vitro / in vivo
Retrieval	Select target	Primer = key
Decoding	Read & reconstruct	Sequencing + ECC

Pros and Cons

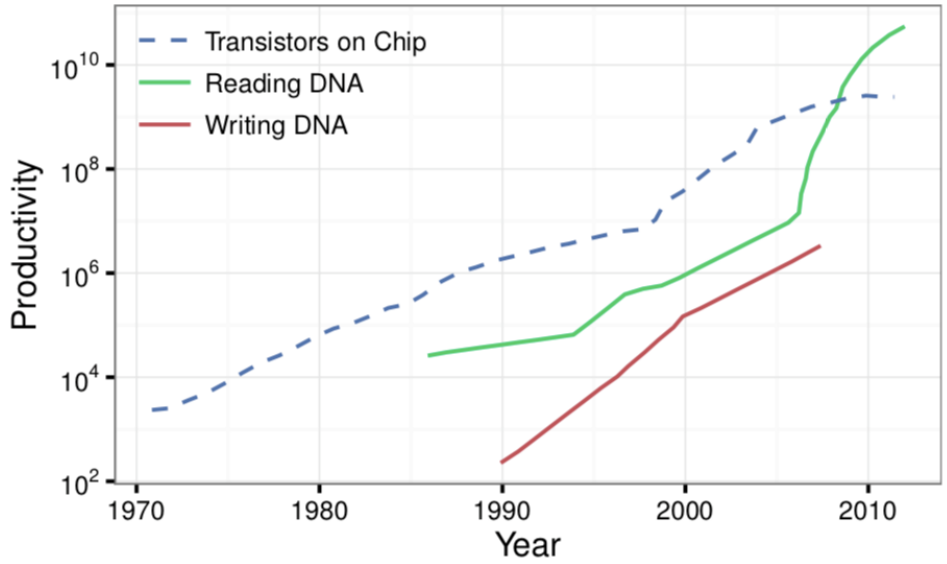


Storage systems

- DNA storage as the bottom level of the storage hierarchy



	Access Time	*Durability*
Flash	ms	~5 yrs
HDD	10s ms	~5 yrs
Tape	minutes	~15-30 yrs
DNA Storage	10s hrs	centuries



Reference

- Bornholt, James, et al. "A DNA-based archival storage system." *Proceedings of the twenty-first international conference on architectural support for programming languages and operating systems*. 2016.

- Lin, Kevin N., et al. "Dynamic and scalable DNA-based information storage." *Nature communications* 11.1 (2020): 2981.

December 5, 2025

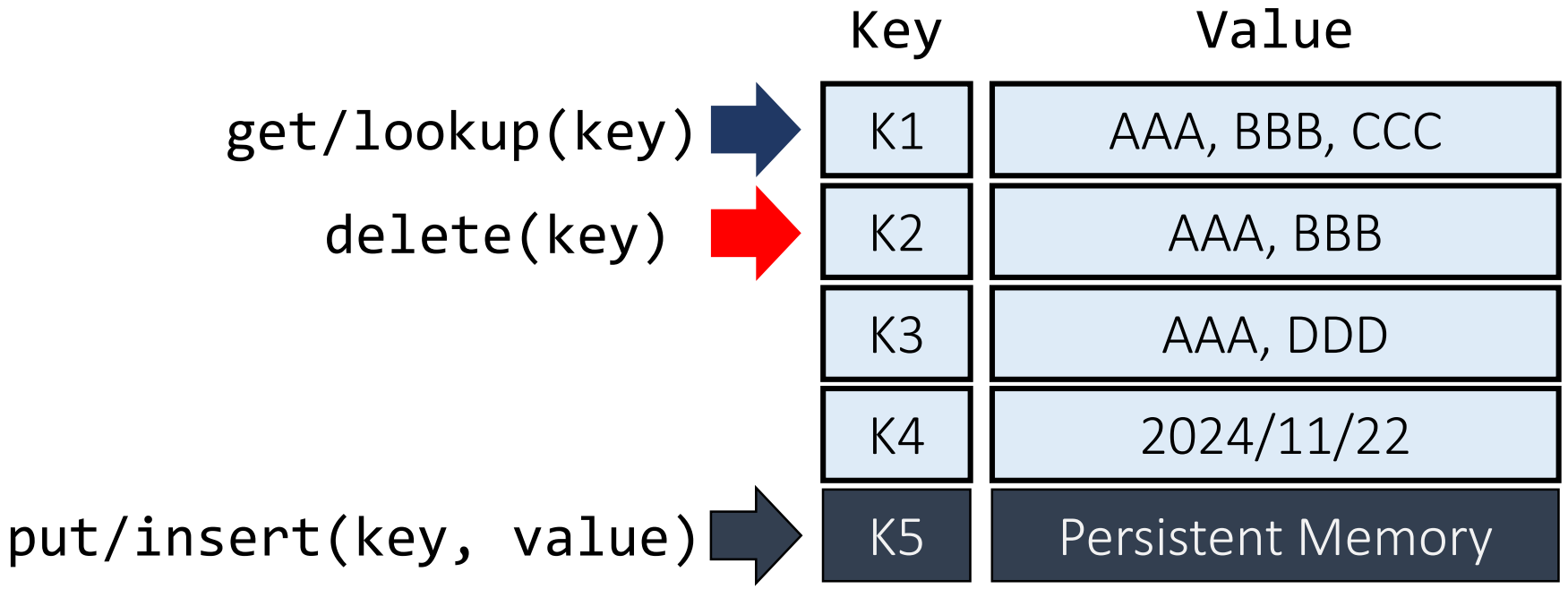
Data index structures:
LSM-Tree, Hash, and Learned Indexes

Outline

- **Log-Structured Merge-Tree (LSM-tree)**
- B^{ε}-tree

Persistent Key-Value Store

- **Persistent key-value (KV) stores** play a critical role in a variety of modern data-intensive applications:
  - Such as e-commerce, cloud data, and social networking.
- In a KV store, data are stored as key-value pairs.
  - A unique `key` is associated with a `value` of "any form".



Log-Structured Merge-Tree (LSM-Tree)

- For write-intensive workloads, KV stores based on **LSM-tree** have become the state of the art.
  - Various distributed or local stores built on LSM-trees are widely deployed in largescale environments, such as:
    - BigTable and LevelDB at **Google**;
    - Cassandra, Hbase, and RocksDB at **Facebook**; and
    - PNUTS at **Yahoo!**

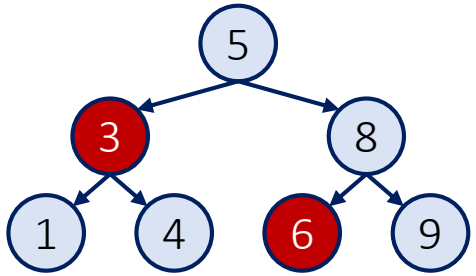
- The main advantage of LSM-trees is that they maintain **sequential access patterns for writes**.
  - The success of LSM-tree is tied closely to its usage upon classic **hard-disk drives (HDDs):** In which, random I/Os are over 100× slower than sequential ones.

Overall Architecture of LSM-Tree

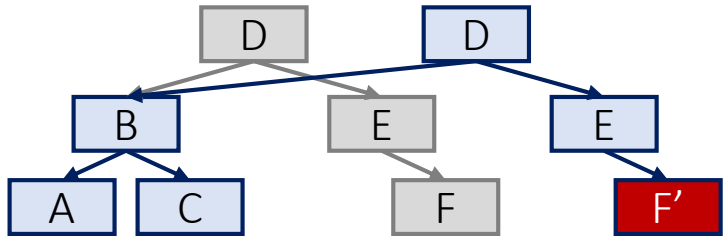
- An **LSM-tree** consists of a number of components of exponentially increasing sizes, **C0** to **Ck**:

C0 is a memory-resident, update-in-place sorted tree.

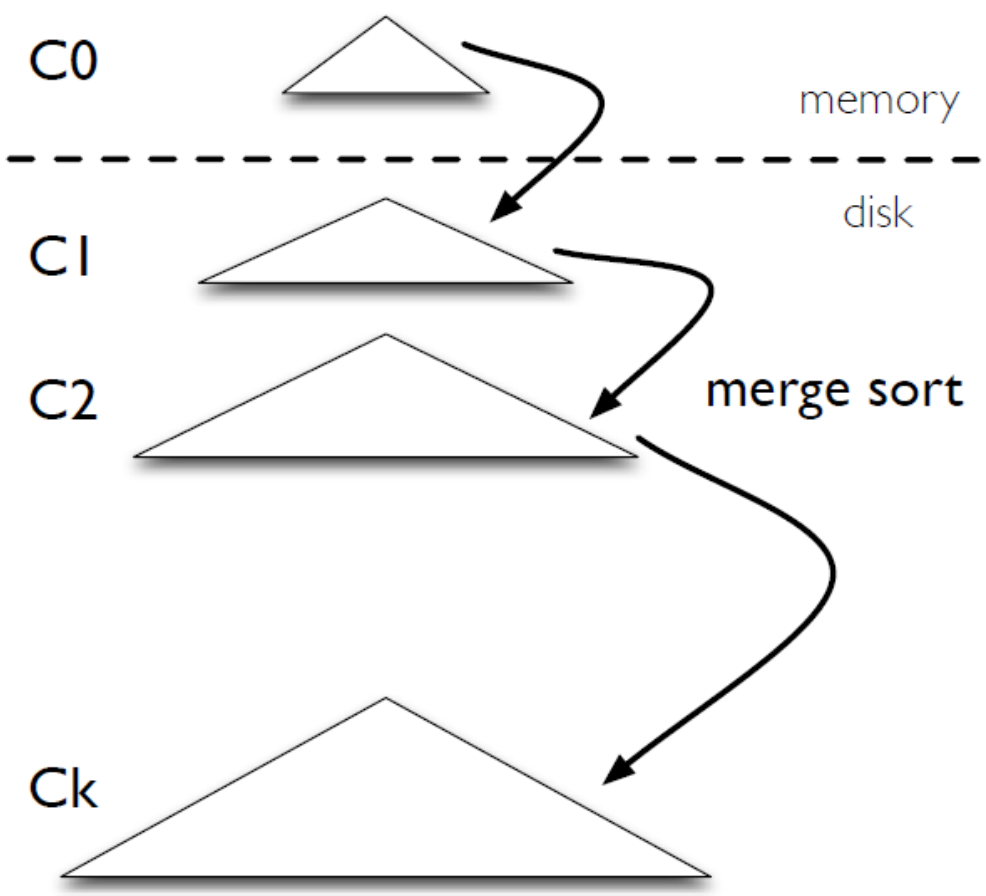
Memory



C1~Ck are disk-resident, append-only B-trees.

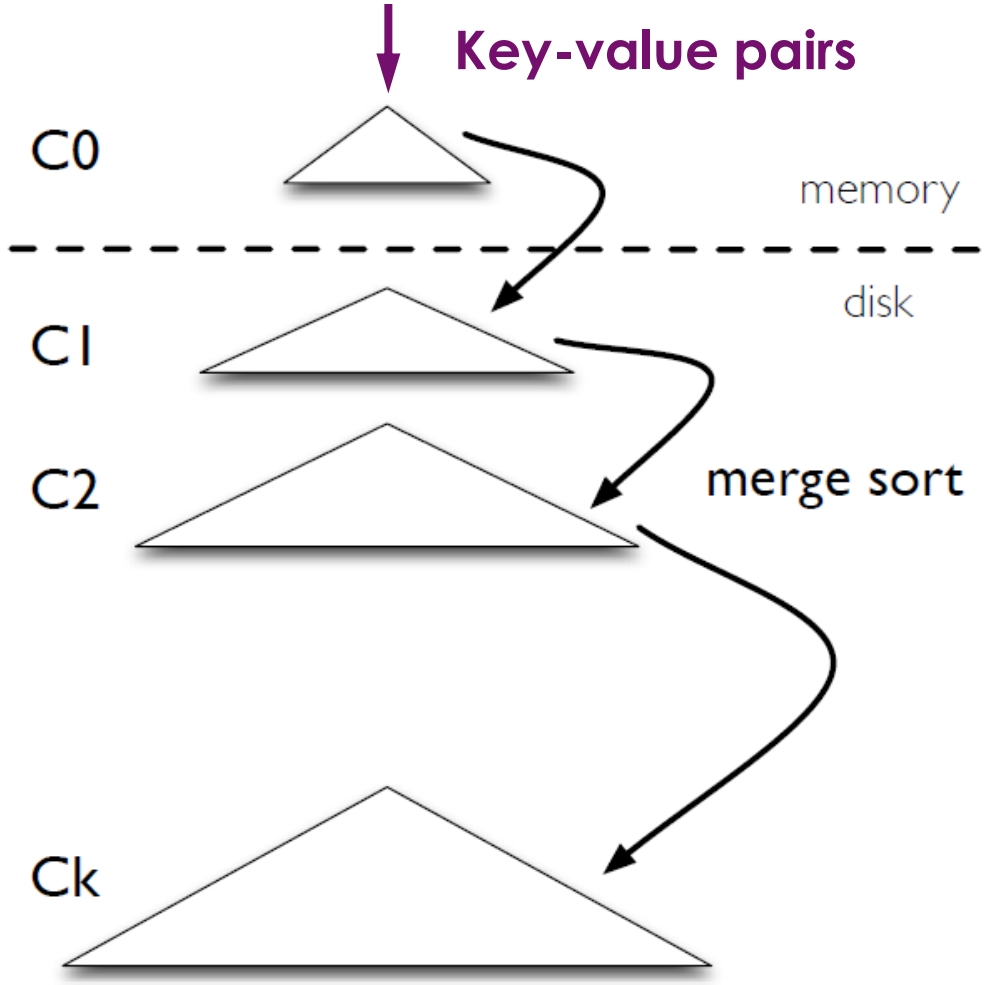


Disk



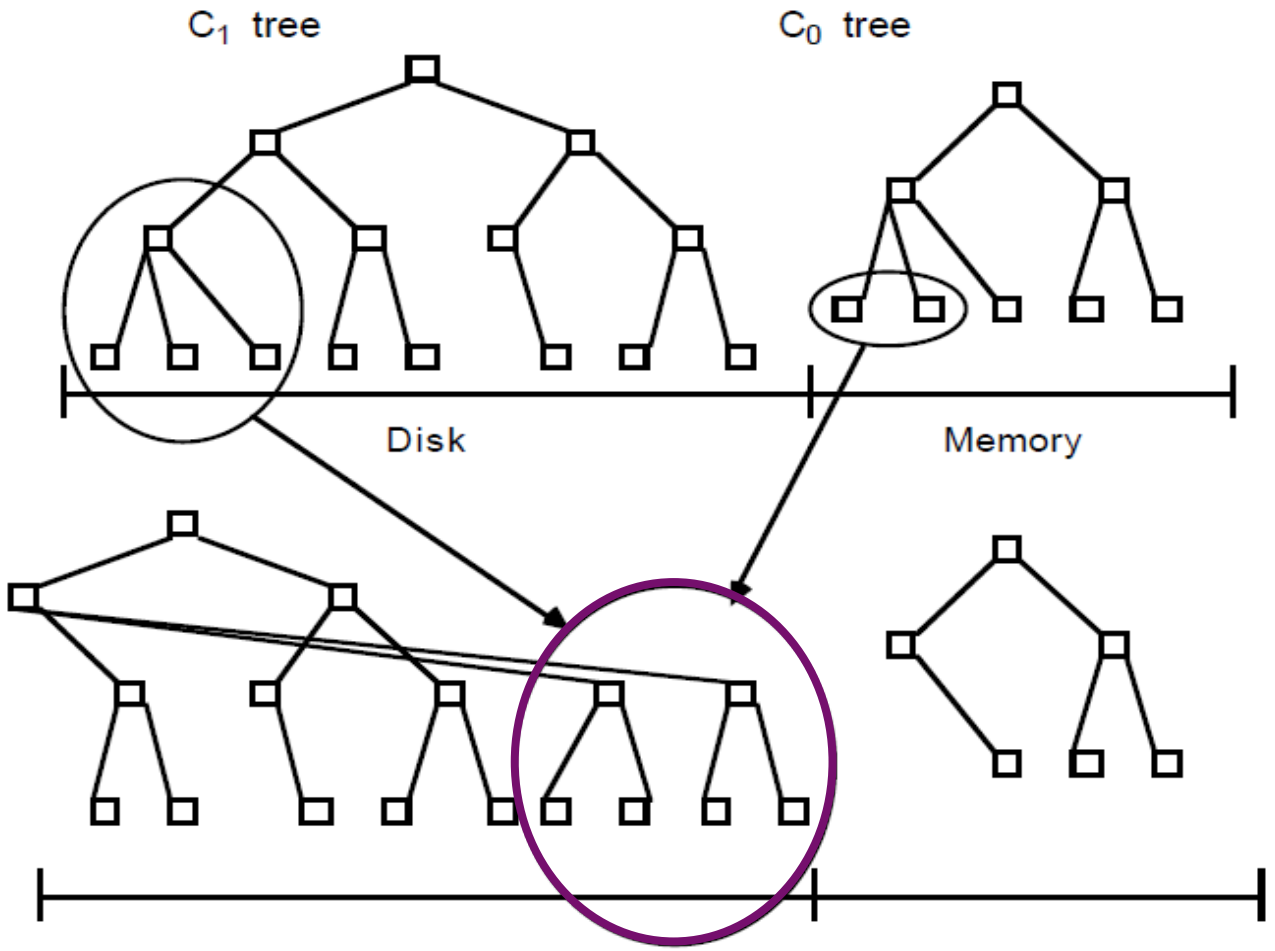
LSM-Tree: Insertion & Compaction (1/2)

- Key-value pairs are always **inserted** into the LSM-tree via the in-memory **C0**.
  - Once **C0** reaches its **size limit**, **C0** will be merged with the on-disk **C1** by the compaction process.
    - The newly merged tree **C1'** will be appended into disk, replacing the old **C1**.
  - Compaction also takes place for all on-disk components, when any **Ci** reaches its **size limit.**



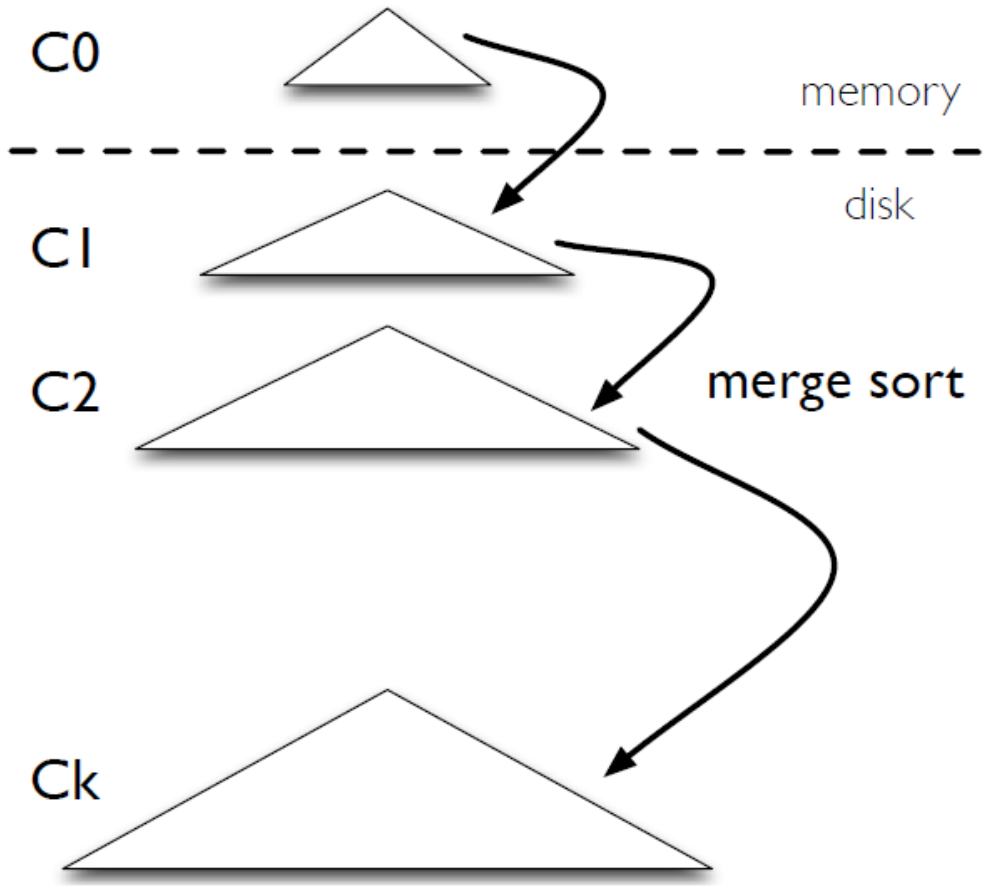
LSM-Tree: Insertion & Compaction (2/2)

- During the compaction, the newly merged blocks are written to **new** disk positions.



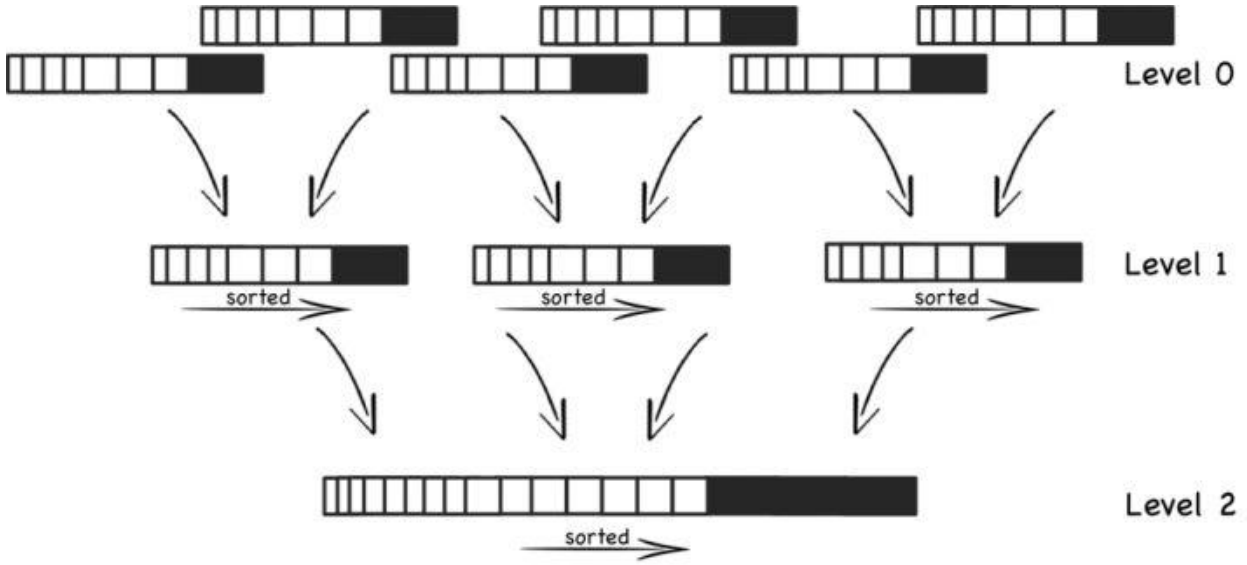
LSM-Tree: Lookup

- To serve a **lookup** operation, LSM-trees may need to search over multiple components.
  - Components are scanned in a cascading fashion, from **C0** to the smallest component **Ci** containing the requested data.
    - Why? **C0** contains the freshest data, followed by **C1**, and so on.
  - Hence LSM-trees are more useful when inserts are more dominant than lookups.



Outline

- **Log-Structured Merge-Tree (LSM-tree)**
  - Persistent Key-Value Store
  - **LevelDB (by Google)**
  - WiscKey: Separating Keys from Values
  - Single-Level KV Store with PM
- B^{ε}-tree

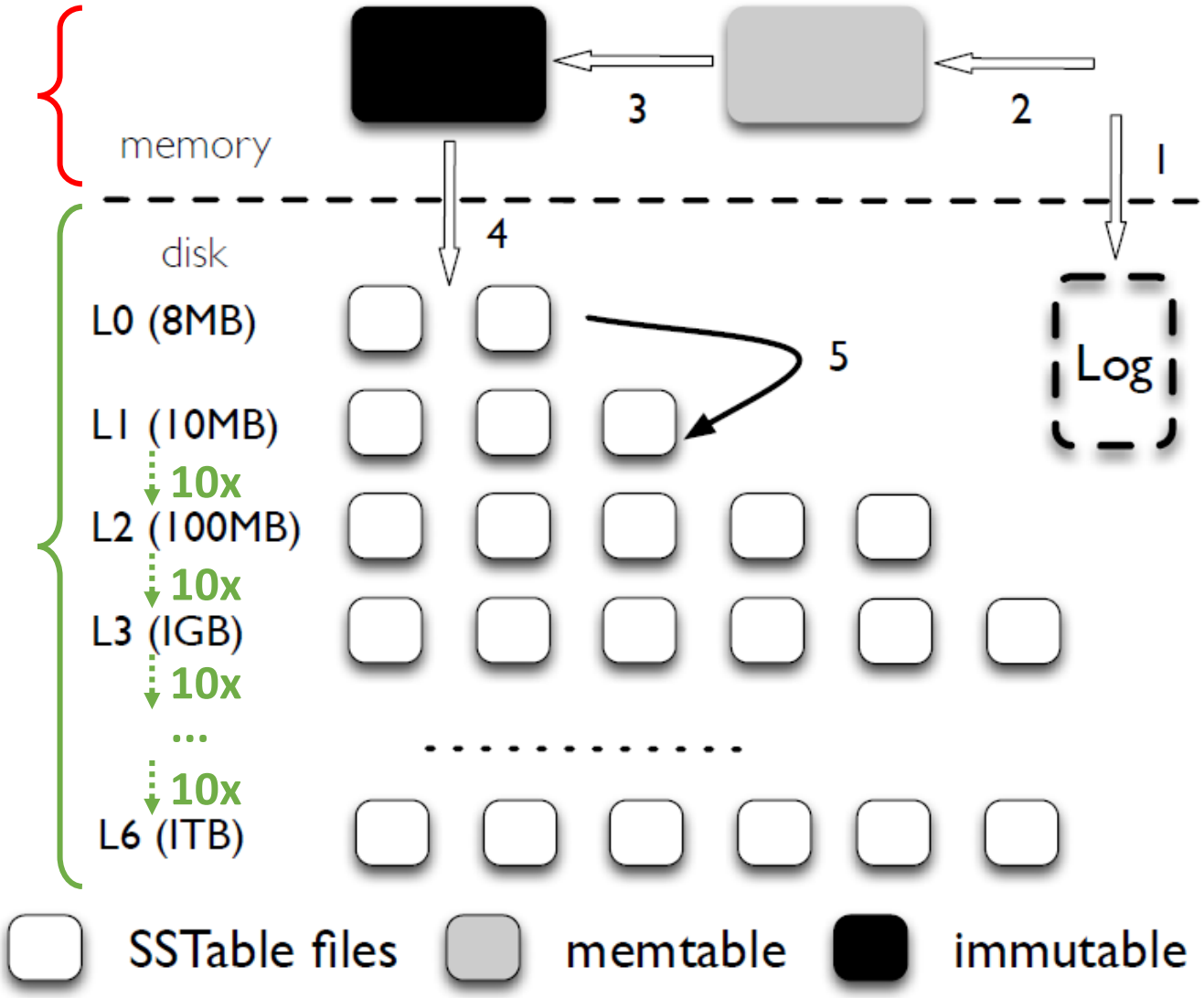


LevelDB (by Google)

- **LevelDB** is a key-value store based on LSM-trees.

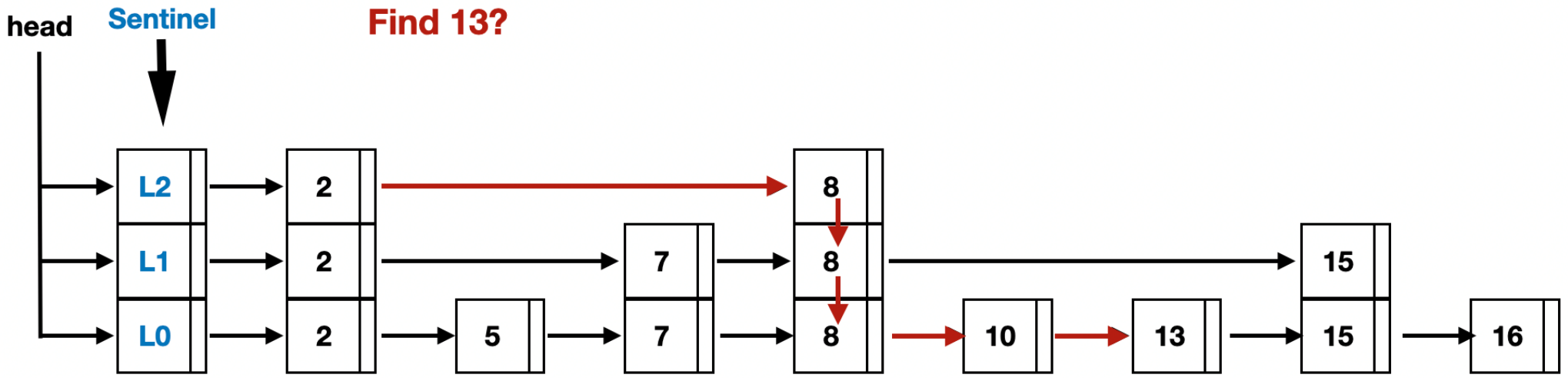
2 in-memory **sorted skiplists** (i.e., `memtable` and `immutable memtable`)

7 "levels" (**L0** to **L6**) of on-disk **sorted string tables** (SSTables)



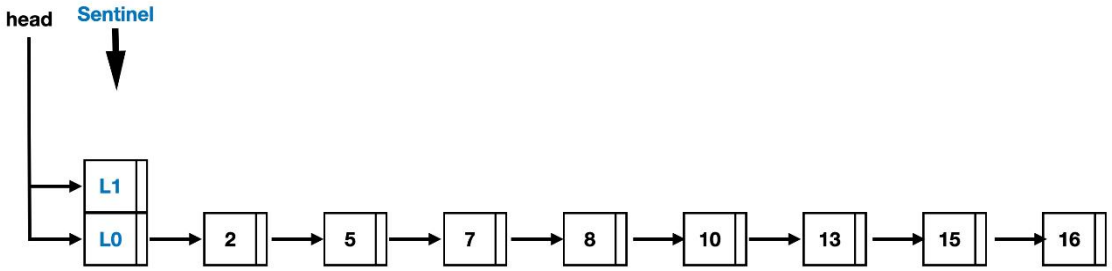
Skiplist

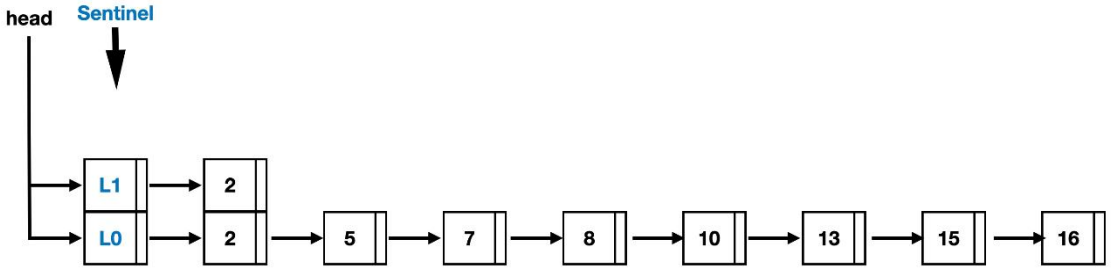
- A **skip list** is a **probabilistic, multi-level linked list** that provides fast search, insert, and delete operations—similar to a balanced tree (like a B-tree or Red-Black tree)—but **simpler to implement**.
  - Rotation operation is not needed.

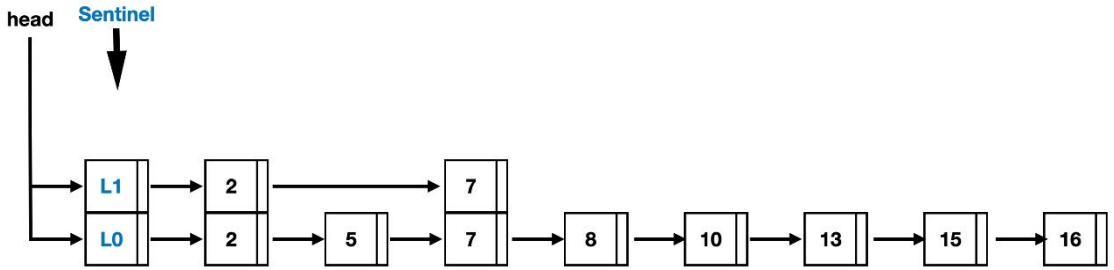


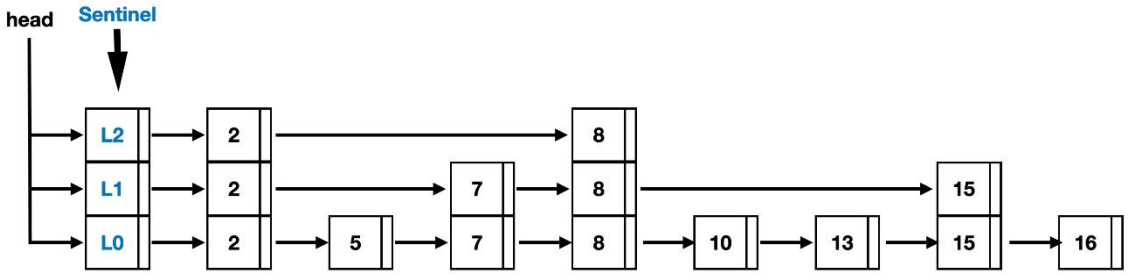
Skiplist

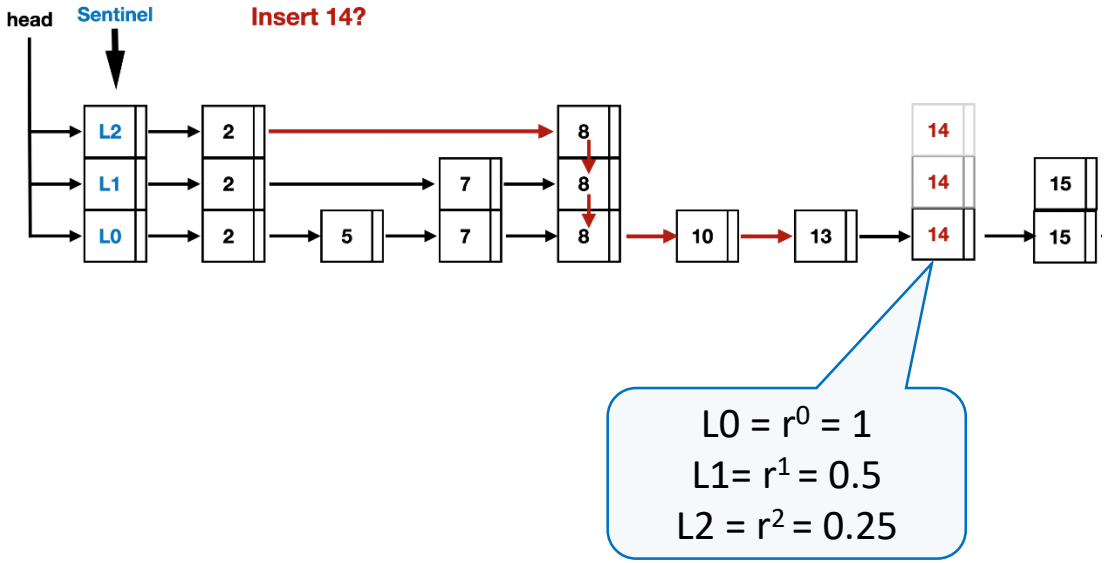
- An example: n=8, r=0.5





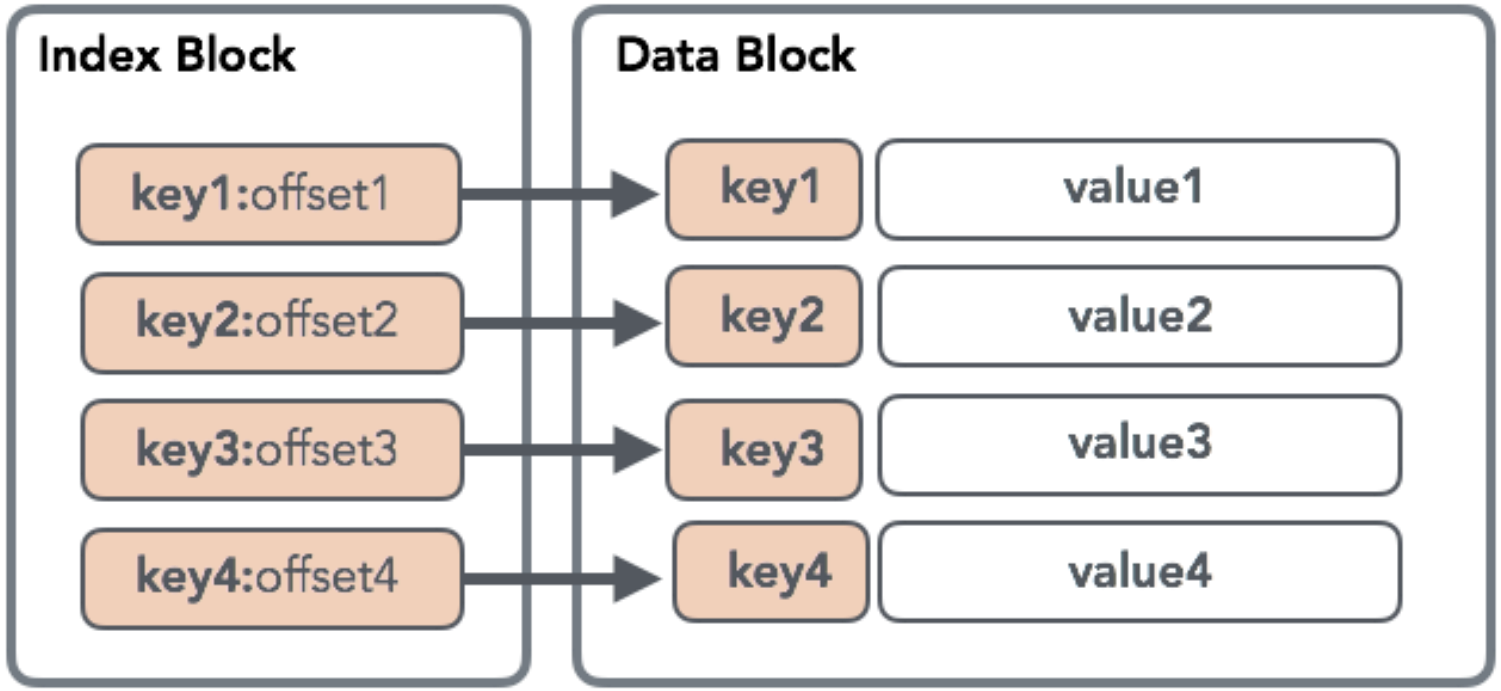






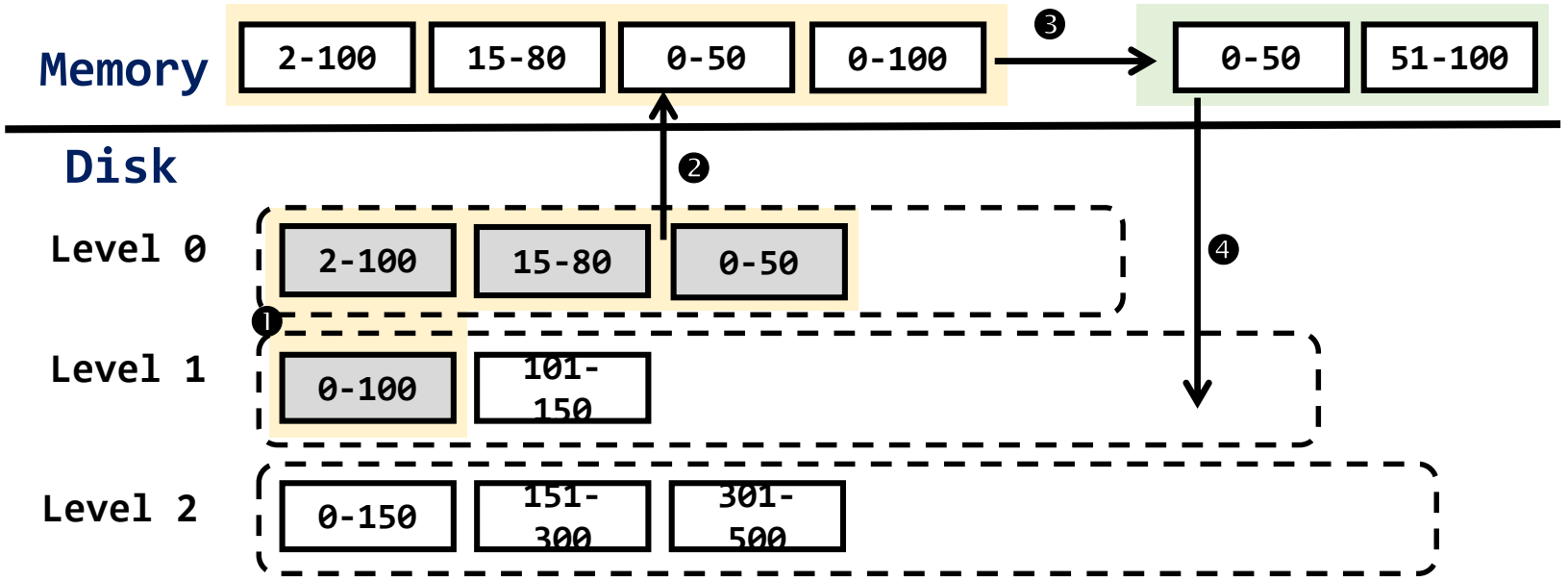
Sorted String Table

- A **sorted string table** (**SSTable**) is simply a file which contains a set of arbitrary, sorted key-value pairs.

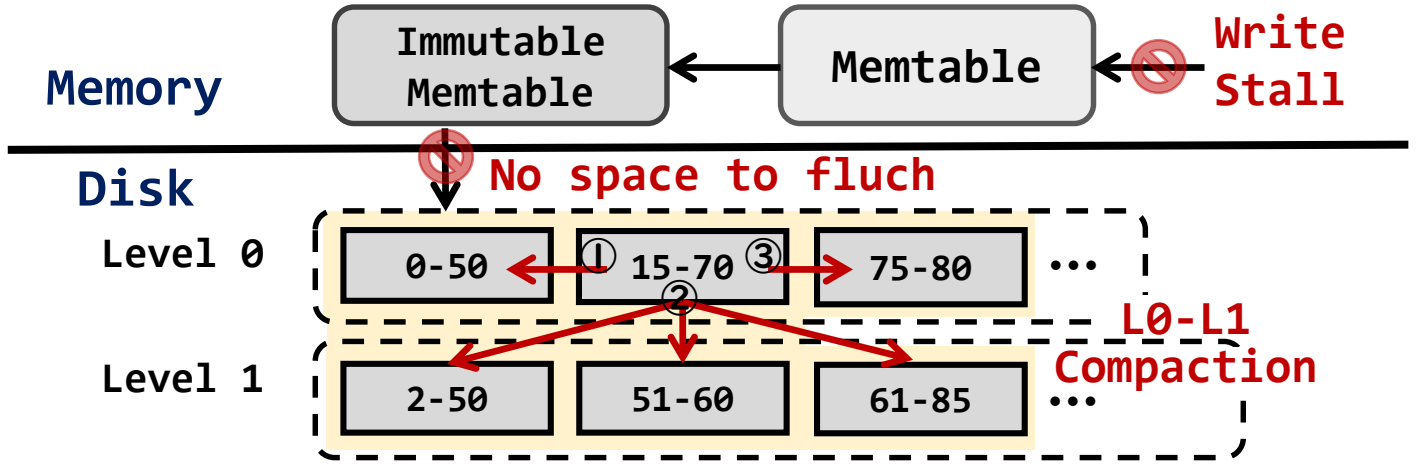


  - **Strength**: High throughput for sequential I/O workloads
  - **Weakness**: Large I/O rewrite for random insert/deletion

LevelDB: Insertion & Compaction

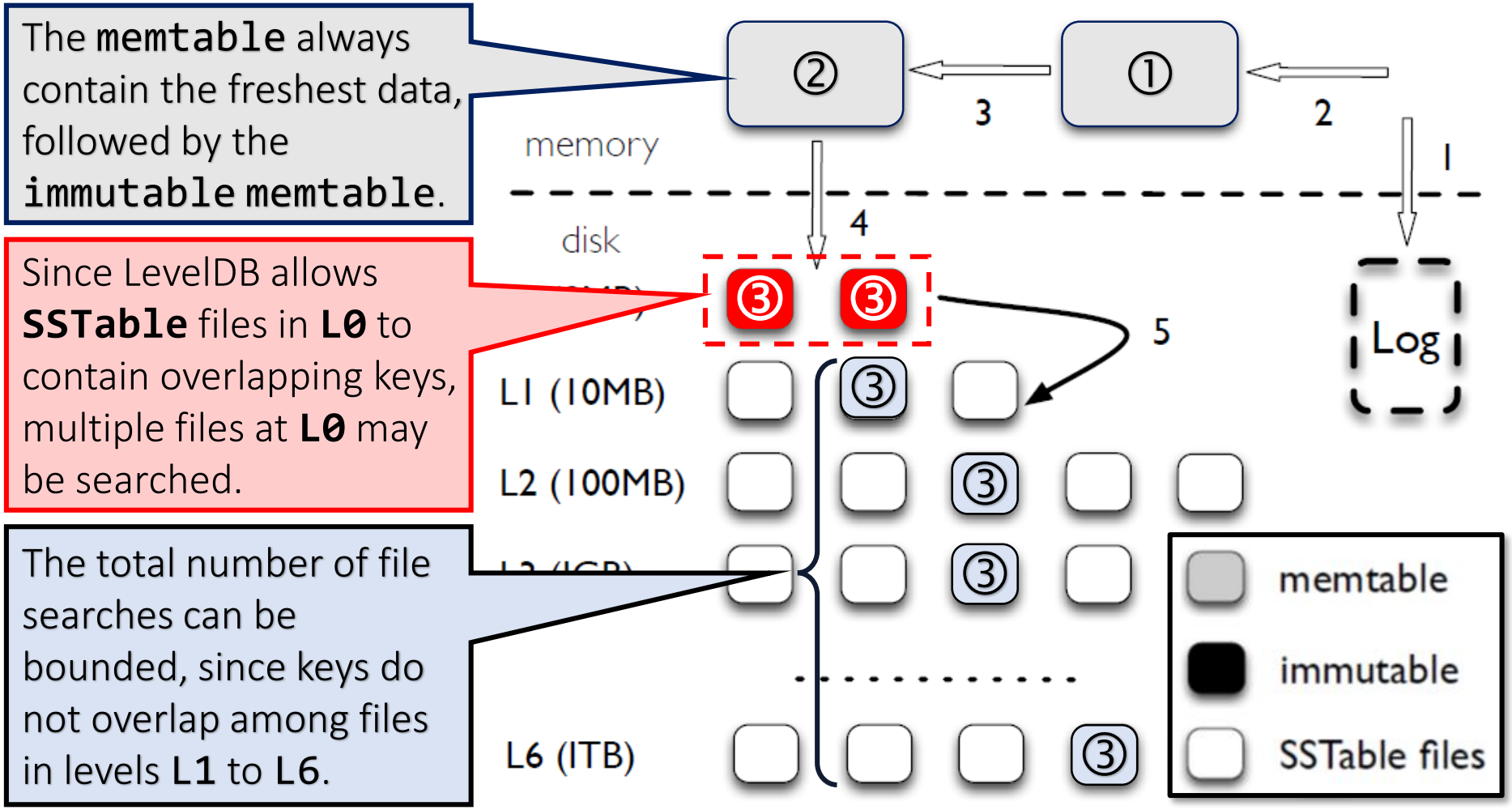


L0-L1 compaction can lead to serious latency and write stall issue



LevelDB: Lookup

- LevelDB searches for a requested KV pair as follows:
  - ① `memtable`, ② `immutable memtable`, ③ files of `L0` to L6 in order

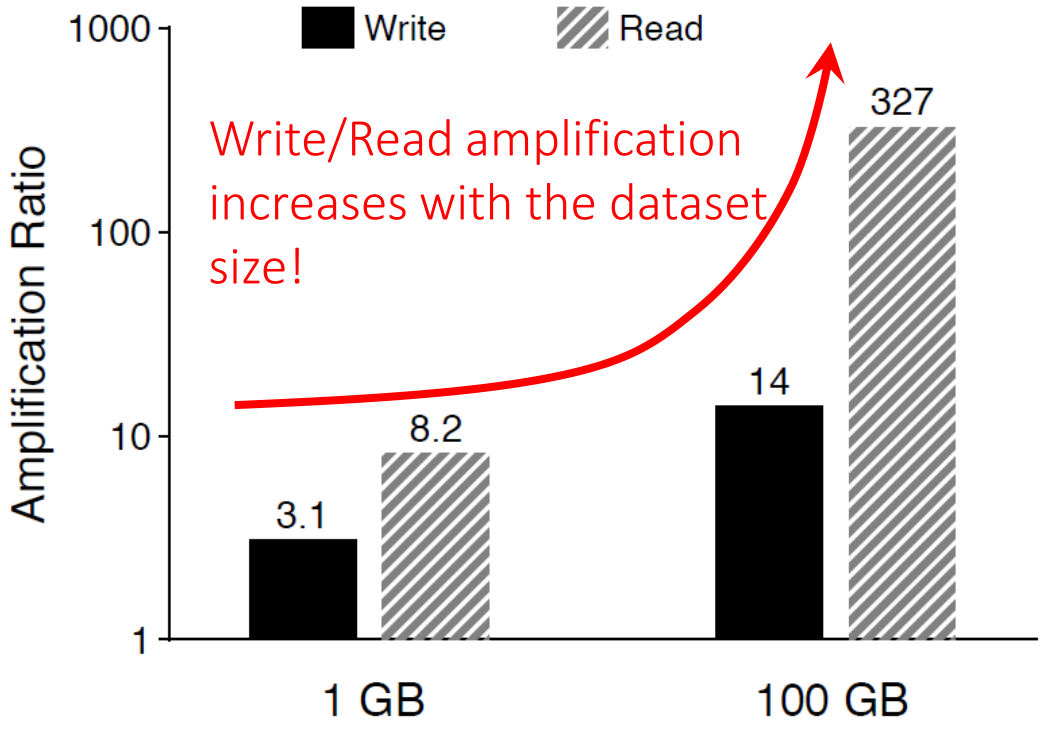


Outline

- **Log-Structured Merge-Tree (LSM-tree)**
  - Persistent Key-Value Store
  - LevelDB (by Google)
  - **WiscKey: Separating Keys from Values**
  - Single-Level KV Store with PM
- B^{ε}-tree

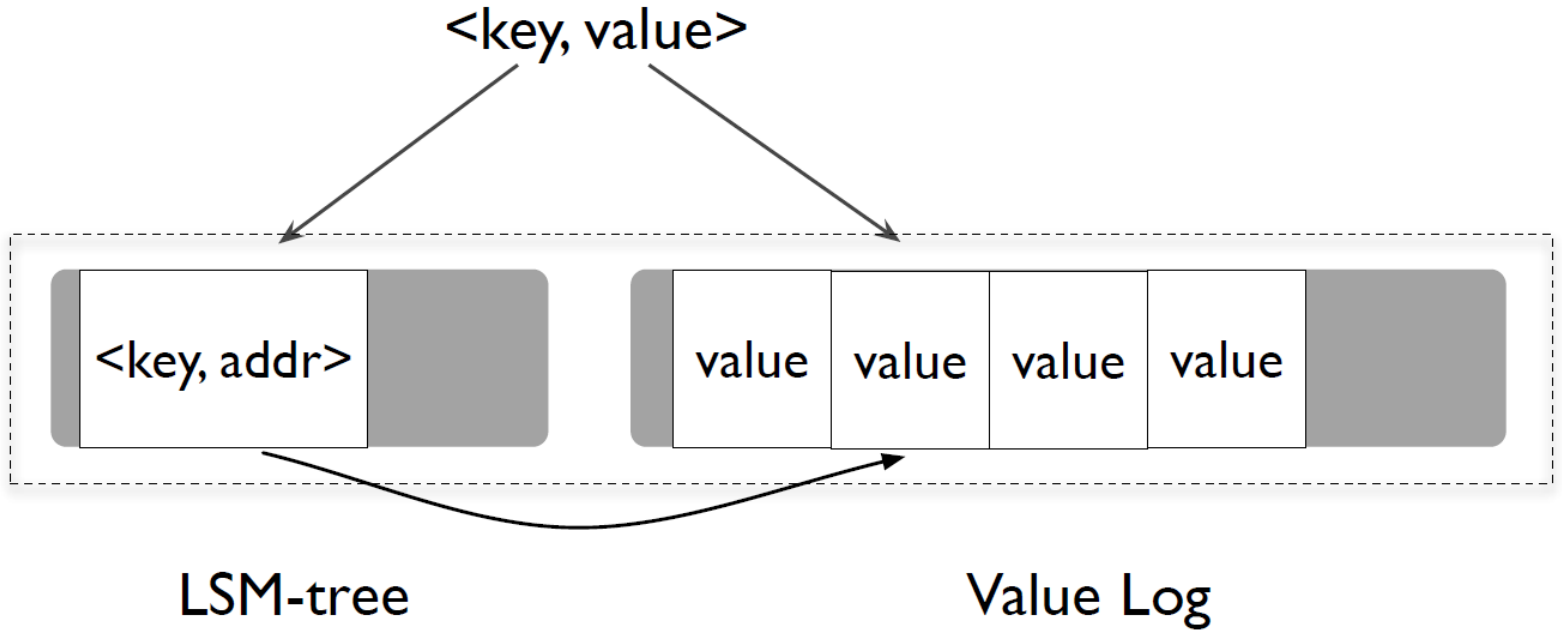
WiscKey: Separating Keys from Values

- **Write and Read Amplification!**
- **Write** and **read amplification** are major problems in LSM-tree based key-value stores such as LevelDB.
  - **Write (Read) Amplification**: the ratio between the amount of data written to (read from) the storage and the amount of data requested by the user.
- The source of **write amplification** in LevelDB:
  - LevelDB writes more data than necessary to achieve mostly-sequential disk access.
- The sources of **read amplification** in LevelDB:
  - To lookup a key-value pair, LevelDB needs to check multiple SSTable files in multiple levels.
  - To find a key-value pair within an SSTable file, LevelDB needs to read multiple metadata blocks within the file.



Key-Value Separation

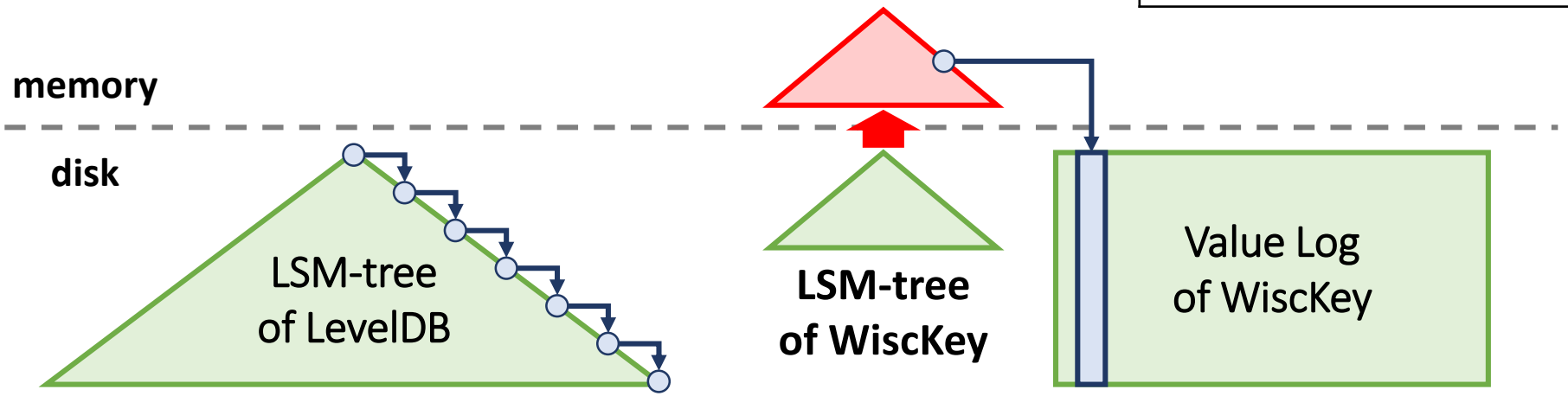
- The major performance cost of LSM-trees is the compaction, which constantly sorts SSTable files.
- **Key-Value Separation**: Compaction only needs to sort keys, while values can be managed separately.
  - Only the "location" (**addr**) of value is stored in the LSM-tree, while real values are stored in a separate value log file.



Benefits of Key-Value Separation

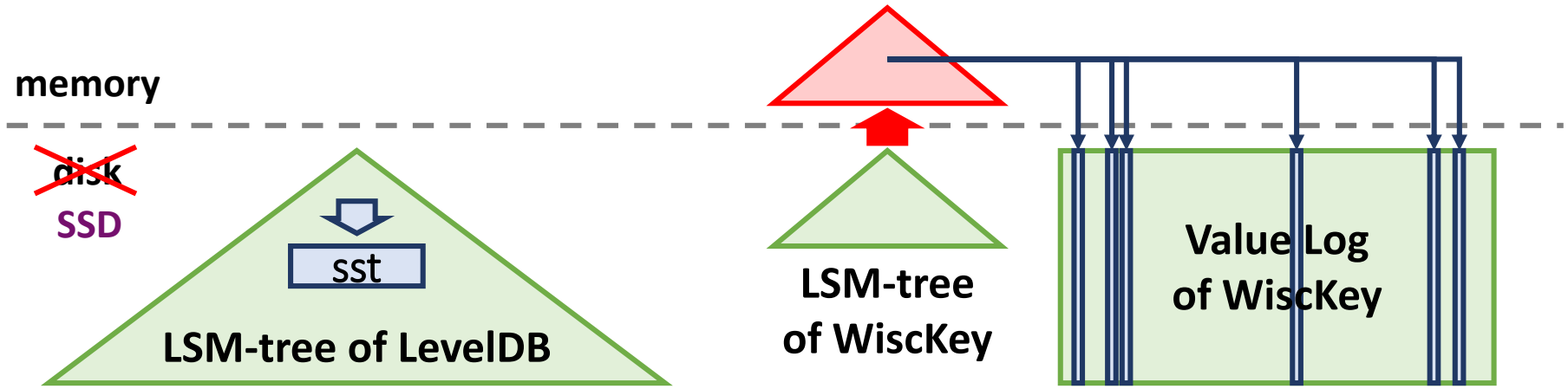
- The LSM-tree of WiscKey becomes much smaller than that of LevelDB.
  - Compacting only keys could significantly reduce the **write amplification**, especially for workloads that have a moderately large value size.
  - A significant portion of the LSM-tree can be possibly cached in memory (to reduce the **read amplification**).
    - A lookup may search fewer levels of table files in the LSM-tree.
    - Most lookups only require a single random read (for the value).

Component	Recommended size
Key	**< 100 bytes** (ideal), avoid >1 KB
Value	**< 4 KB** (ideal), avoid >1 MB
SSTable block	4KB – 4MB (typically)



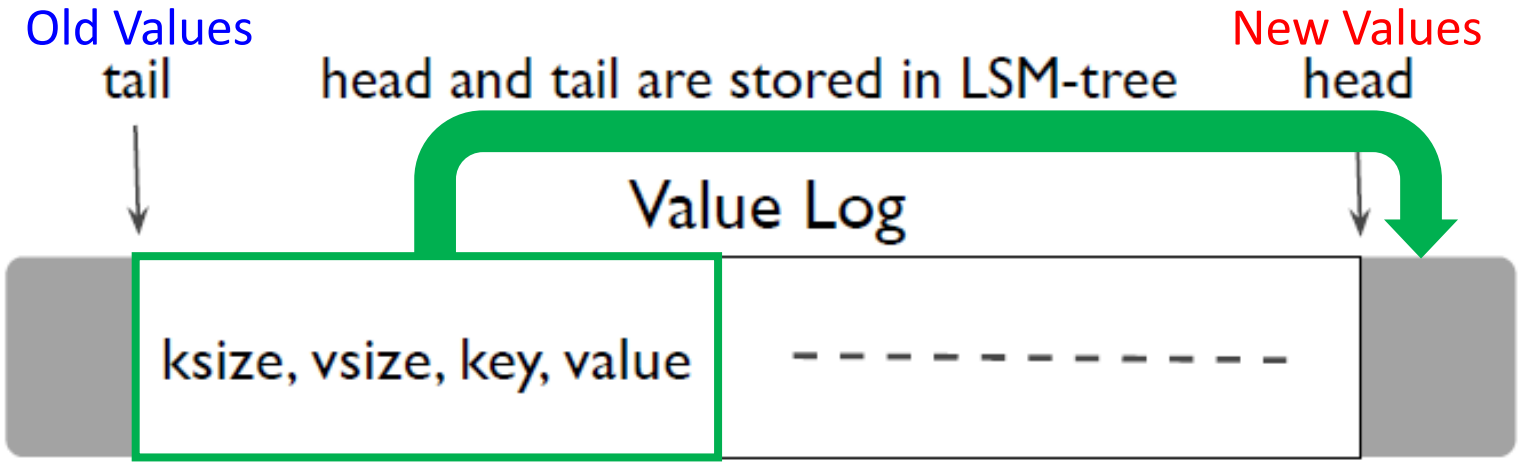
Challenges of Key-Value Separation

- Key-value separation may leads to many **challenges**:
- **Challenge #1**: Since keys and values are separately stored in WiscKey, range queries require multiple random reads, which are not efficient to the disk.
- The design of WiscKey is highly SSD optimized.
  - **Parallel random reads** with a fairly large request size can fully utilize the internal parallelism of SSD, getting performance similar to sequential reads.



Challenges of Key-Value Separation

- **Challenge #2**: Since WiscKey does not compact values, it needs a special garbage collector to reclaim space occupied by deleted/overwritten values in **vLog**.
- WiscKey targets a lightweight and online GC: It only keeps valid values in a contiguous range of **vLog**.
  - Valid values are appended back to the head of **vLog**.
  - Both keys and values should be kept in **vLog** to determine whether a value is valid or not (**by querying the LSM-tree**).

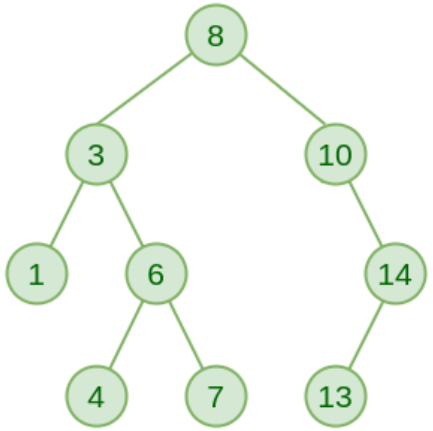


Outline

- Log-Structured Merge-Tree (LSM-tree)
- **B^{ε}-tree**

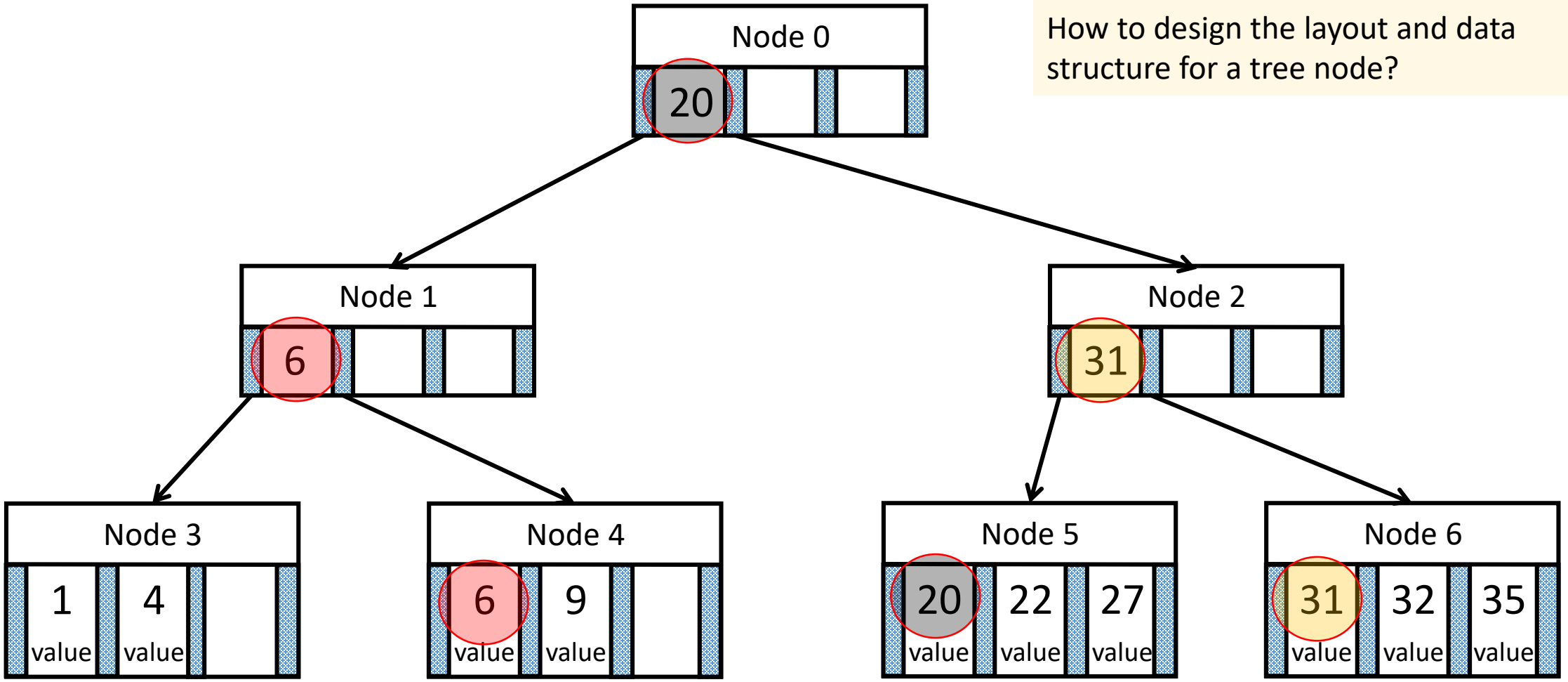
B-tree

- An index structure accelerates the search operations on databases or file systems.
- B-tree is a self-balanced tree structure which is designed to support efficient search, insert, and delete operations.
- Each node in a B-tree can store multiple key-value pairs.



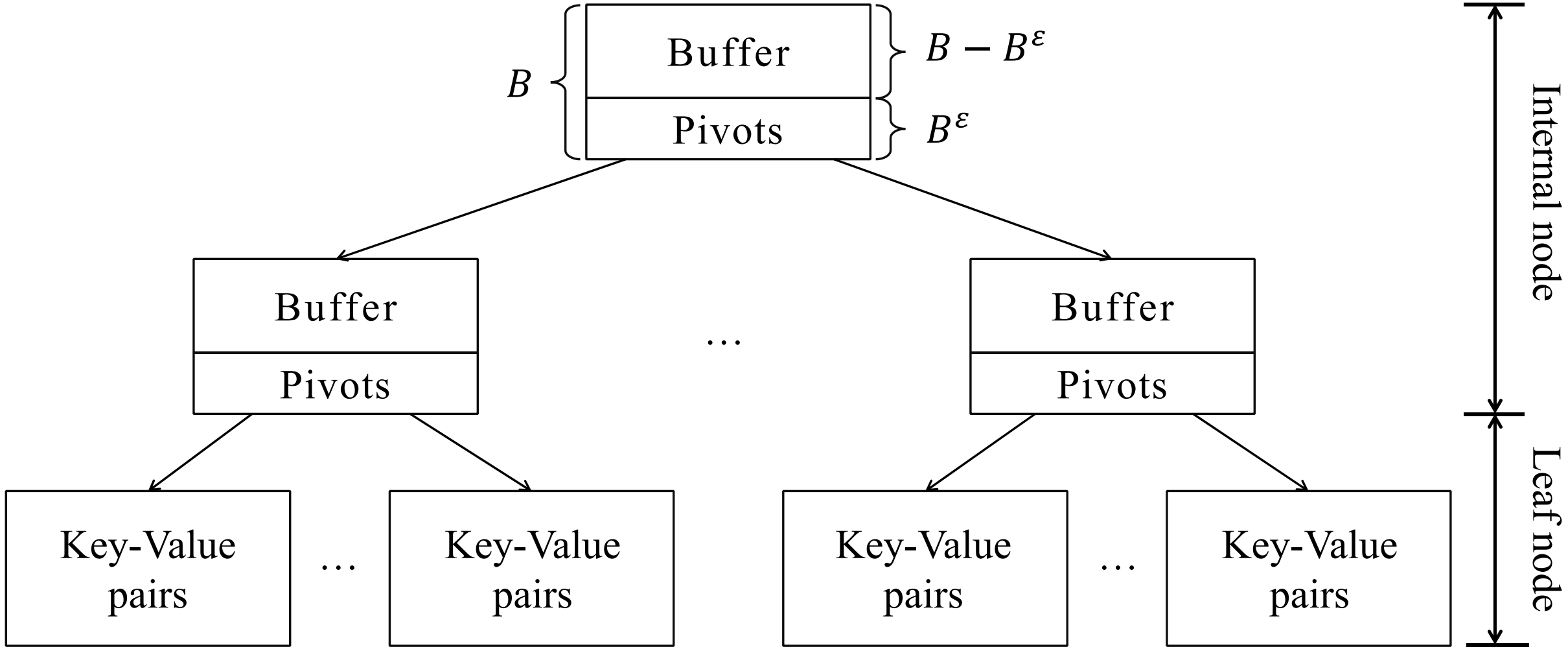
Binary Search Tree

An Example of B$^+$-tree



How to design the layout and data structure for a tree node?

B^{ε}-tree (B-epsilon-tree)



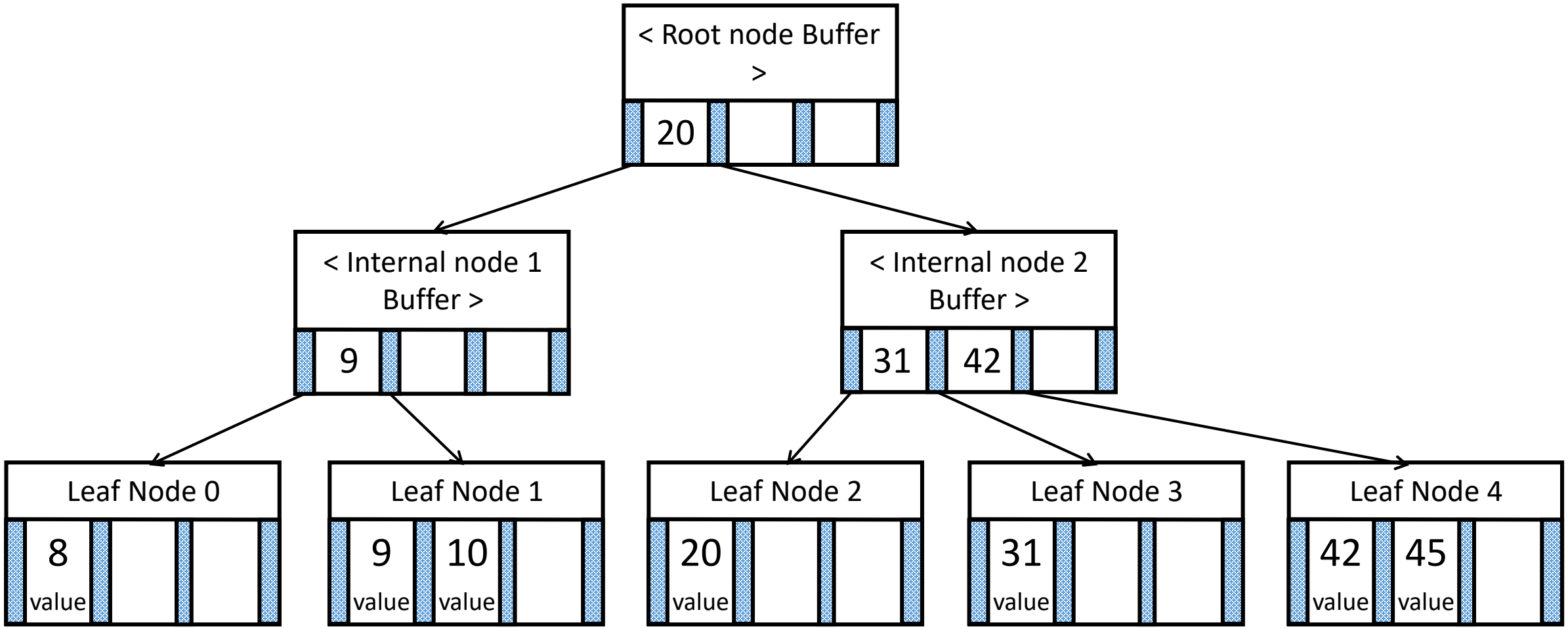
https://www3.cs.stonybrook.edu/~bender/newpub/2015-BenderFaJa-login-wods.pdf

B^{ε}-tree

- Insert, update and delete operations are encoded as messages and initially added to the **buffer** of the root node.
- When a node's buffer is full, the messages will propagate downward until they reach the leaf nodes, called a **flush**.
- The buffer scheme postpones the time triggering tree-balance.

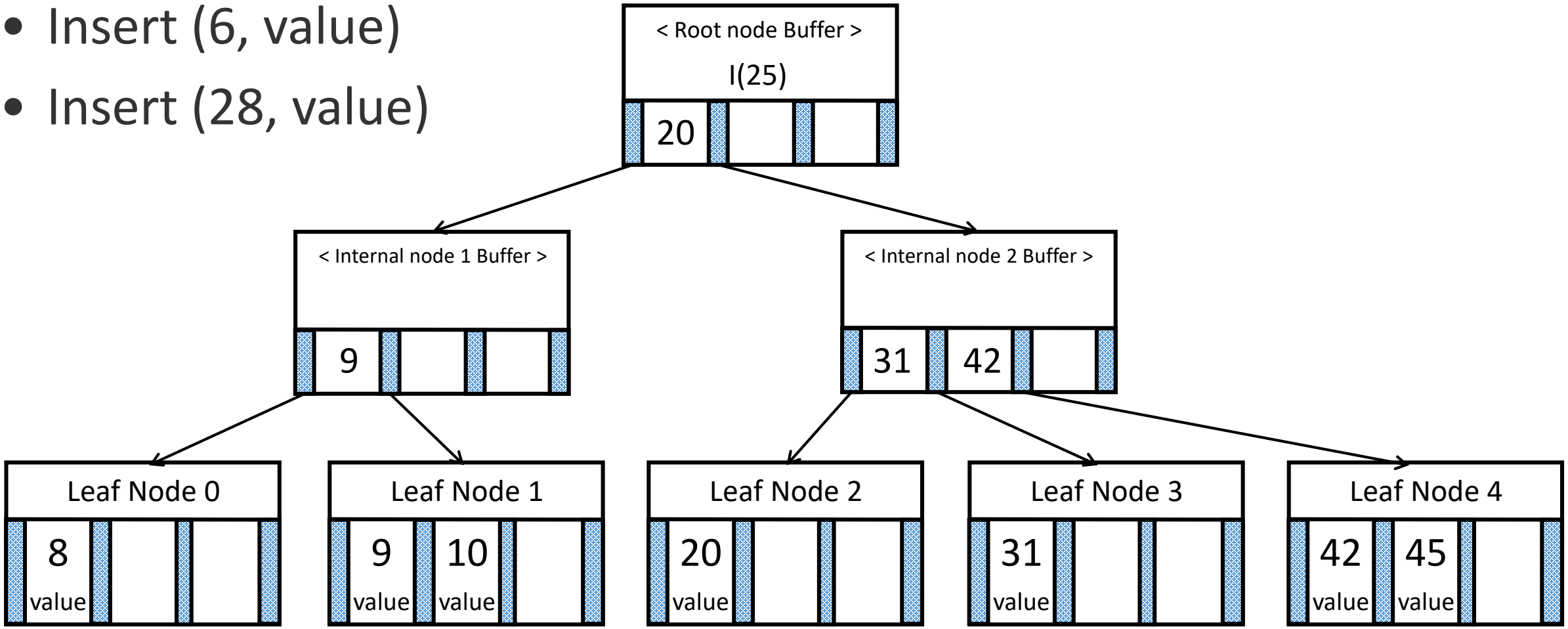
An Example of B$^{\varepsilon}$-tree

- Insert (25, value)



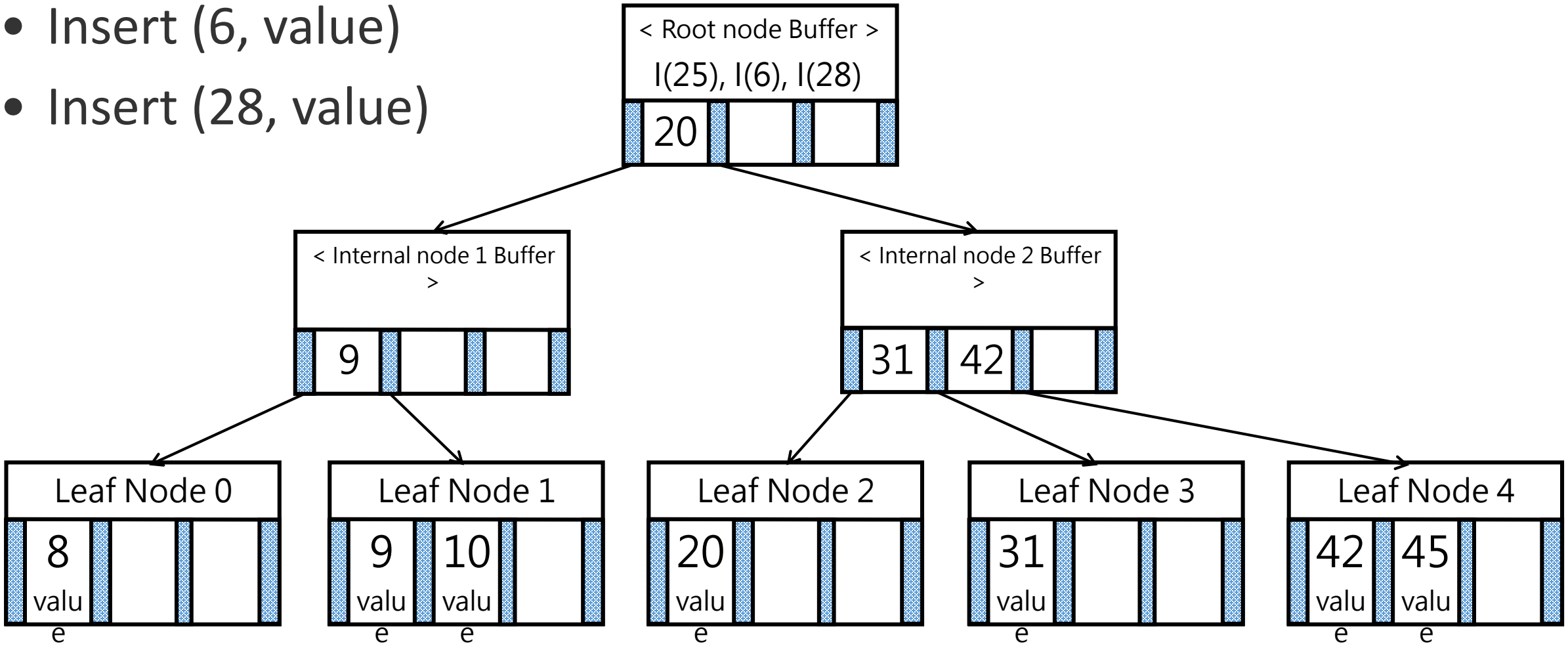
An Example of B$^{\varepsilon}$-tree

- Insert (25, value)
- Insert (6, value)
- Insert (28, value)



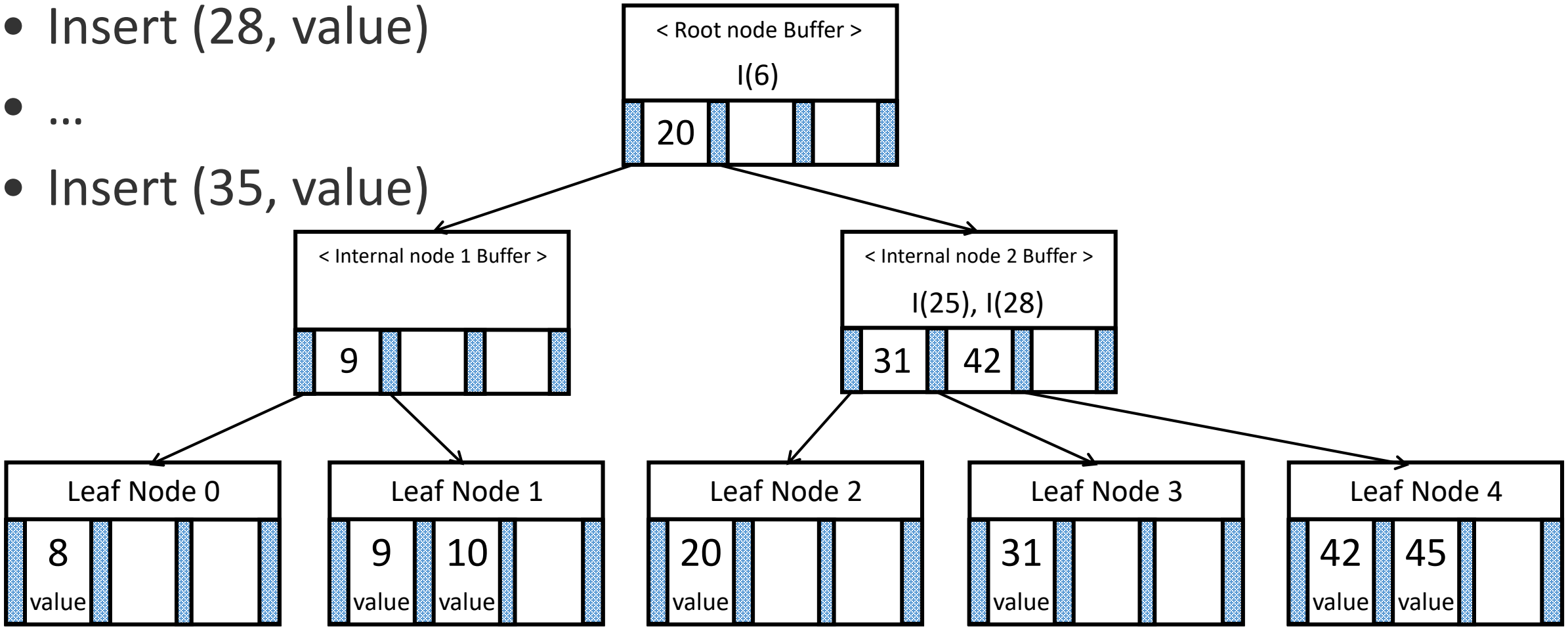
An Example of B$^{\varepsilon}$-tree

- Insert (25, value)
- Insert (6, value)
- Insert (28, value)



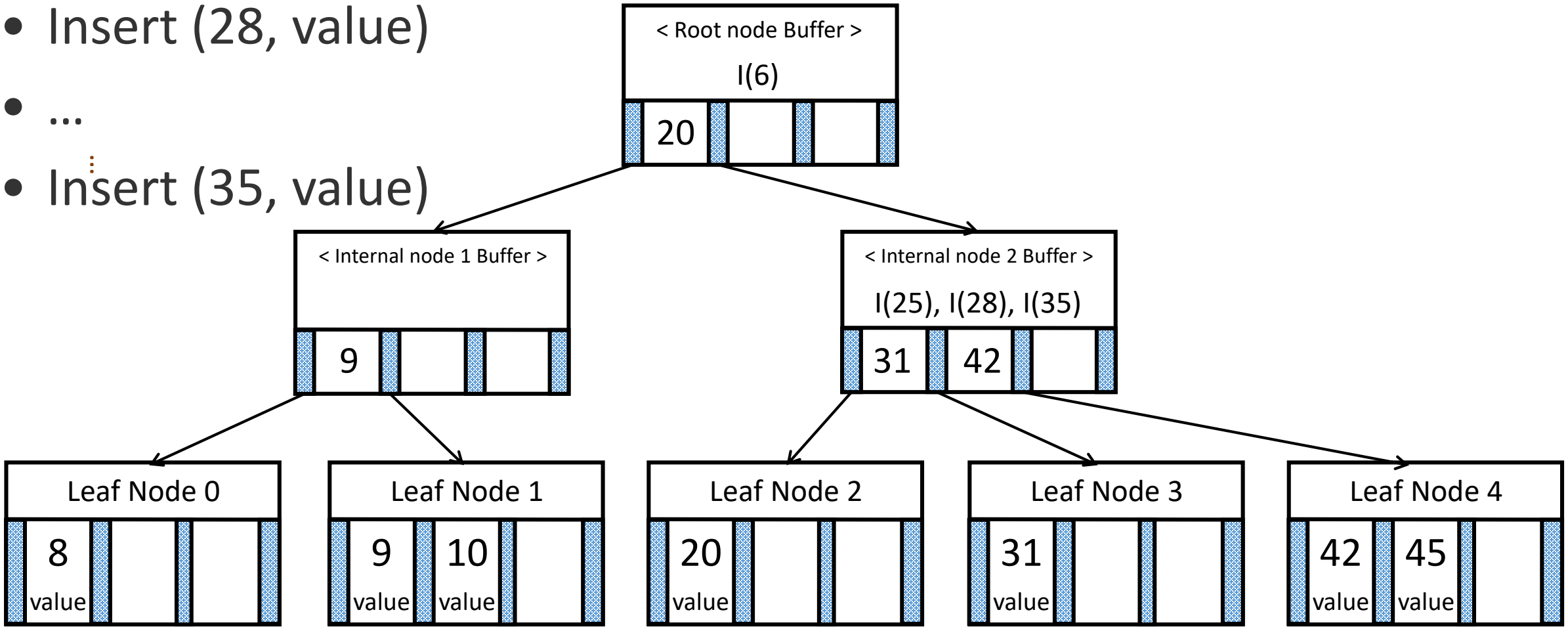
An Example of B$^{\varepsilon}$-tree: Flush

- …
- Insert (28, value)
- …
- Insert (35, value)



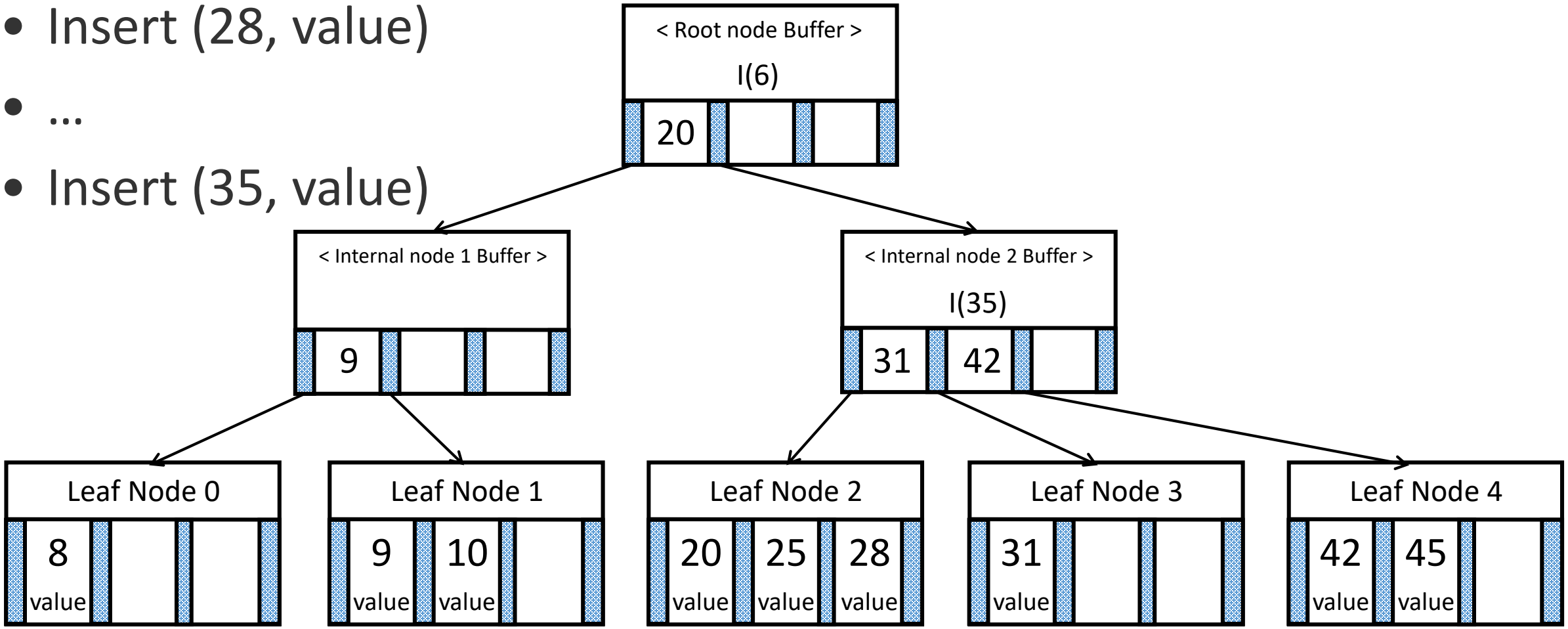
An Example of B$^{\varepsilon}$-tree: Flush

- …
- Insert (28, value)
- …
- Insert (35, value)



An Example of B$^{\varepsilon}$-tree: Flush

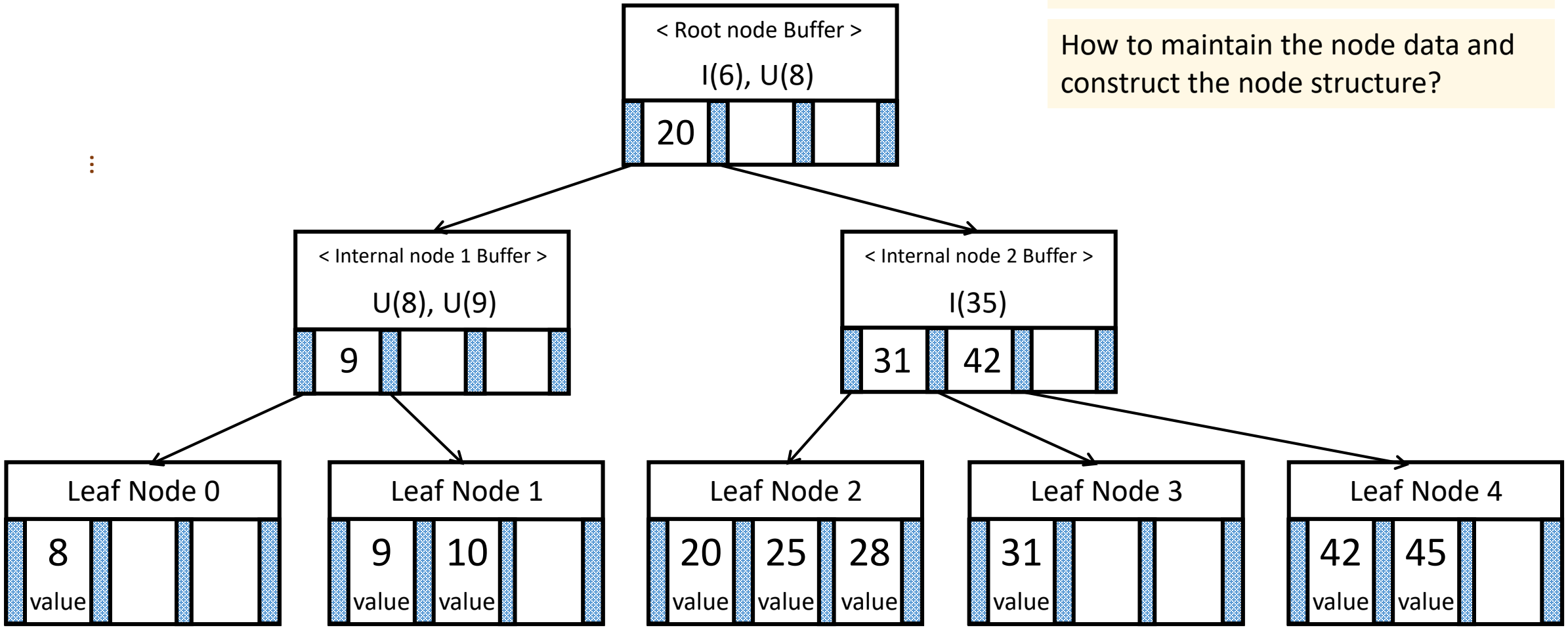
- …
- Insert (28, value)
- …
- Insert (35, value)



An Example of B$^{\varepsilon}$-tree: Flush

How to determine the to-be-flushed keys with considering the underlining storage characteristics?

How to maintain the node data and construct the node structure?



Summary

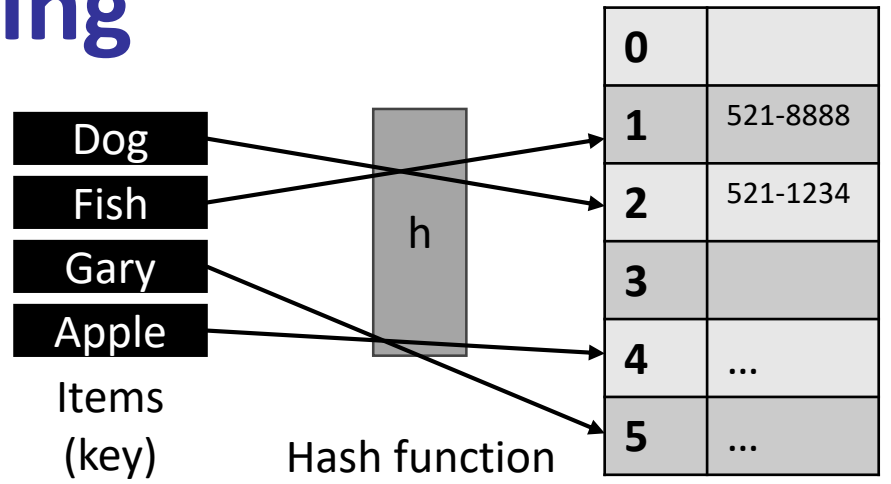
Category	LSM-tree (Log-Structured Merge Tree)	Bε-tree (B-epsilon Tree)
Characteristics	• **Multi-level structure** (Compaction). • **Sequential writes** on disk.	• **B+-tree variant** • **Reduces write frequency**
Advantages	- Excellent sequential write performance. - High throughput for insert-heavy workloads. - Mature and widely deployed.	- Lower write amplification. - Balanced read/write performance. - Better suited for frequent updates and random writes.
Disadvantages	- **High write amplification** due to multi-level compaction. - **Read amplification** (must check multiple levels). - Expensive updates and deletions.	- More **complex structure** and implementation. - **Buffer management** adds runtime overhead. - Not as widely supported in modern systems.
Typical Applications	RocksDB, LevelDB, WiscKey	Workloads needing balanced read/write under update-heavy environments.

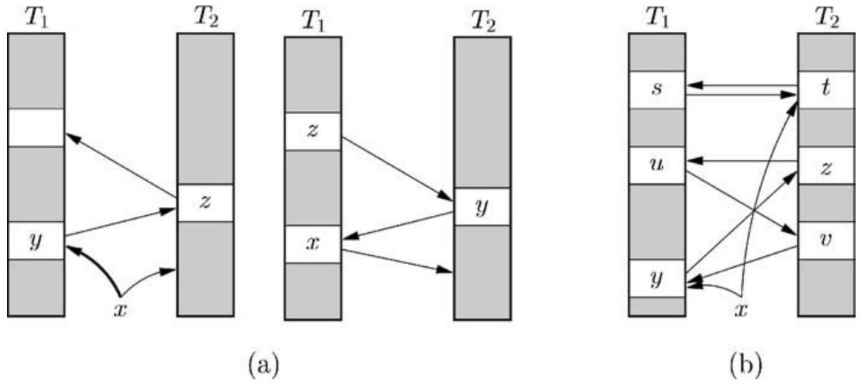
Outline

- **Hash**
- Learned index

Background: Hashing

- **What is Hashing?**
  - Hashing is a technique that maps a key to a fixed-size index using a hash function.
  - It enables fast data retrieval with expected **O(1)** lookup time.
- **Static Hashing**
  - Uses a **fixed-size hash table**.
  - Hash function example:
    - **H(key) = key % table_size**
  - Keys are stored in buckets indexed by the hash value.
- **Drawbacks**
  - **Hash Collisions:** Multiple keys map to the same bucket.
  - **Full-Table Rehashing:**
    - When the table becomes full or collisions increase, the entire table must be enlarged and rebuilt
      → **Very expensive operation** (shown in page 3 of the reference slide)
- **Common Collision-Handling Techniques**
  - **Linear Probing** – Find next available slot sequentially.
  - **Chaining** – Use linked lists to store multiple keys per bucket.
  - **Double Hashing / Cuckoo Hashing**





Cuckoo hashing

When maintained in PM (NVM), multiple non-trivial challenges exist

Cuckoo Hashing

- Cuckoo hashing is a hashing-based dictionary data structure designed to provide **worst-case constant-time lookups** while maintaining **linear space usage**
- **Key Idea**
  - Cuckoo hashing uses **two hash tables** (or two positions within a single table) and **two hash functions**
  - Every key xxx is stored **in exactly one of two possible locations**

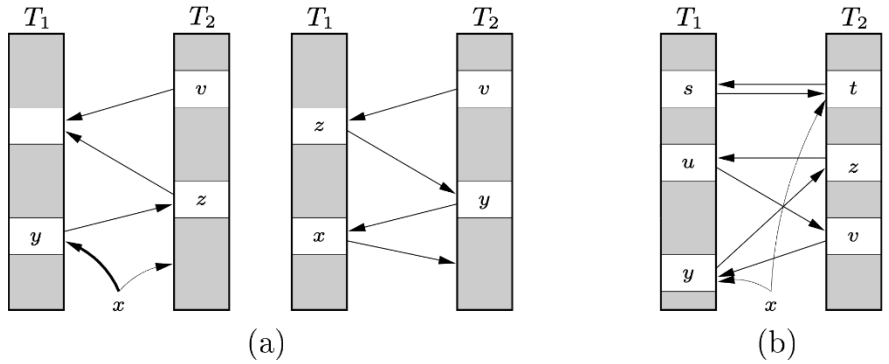
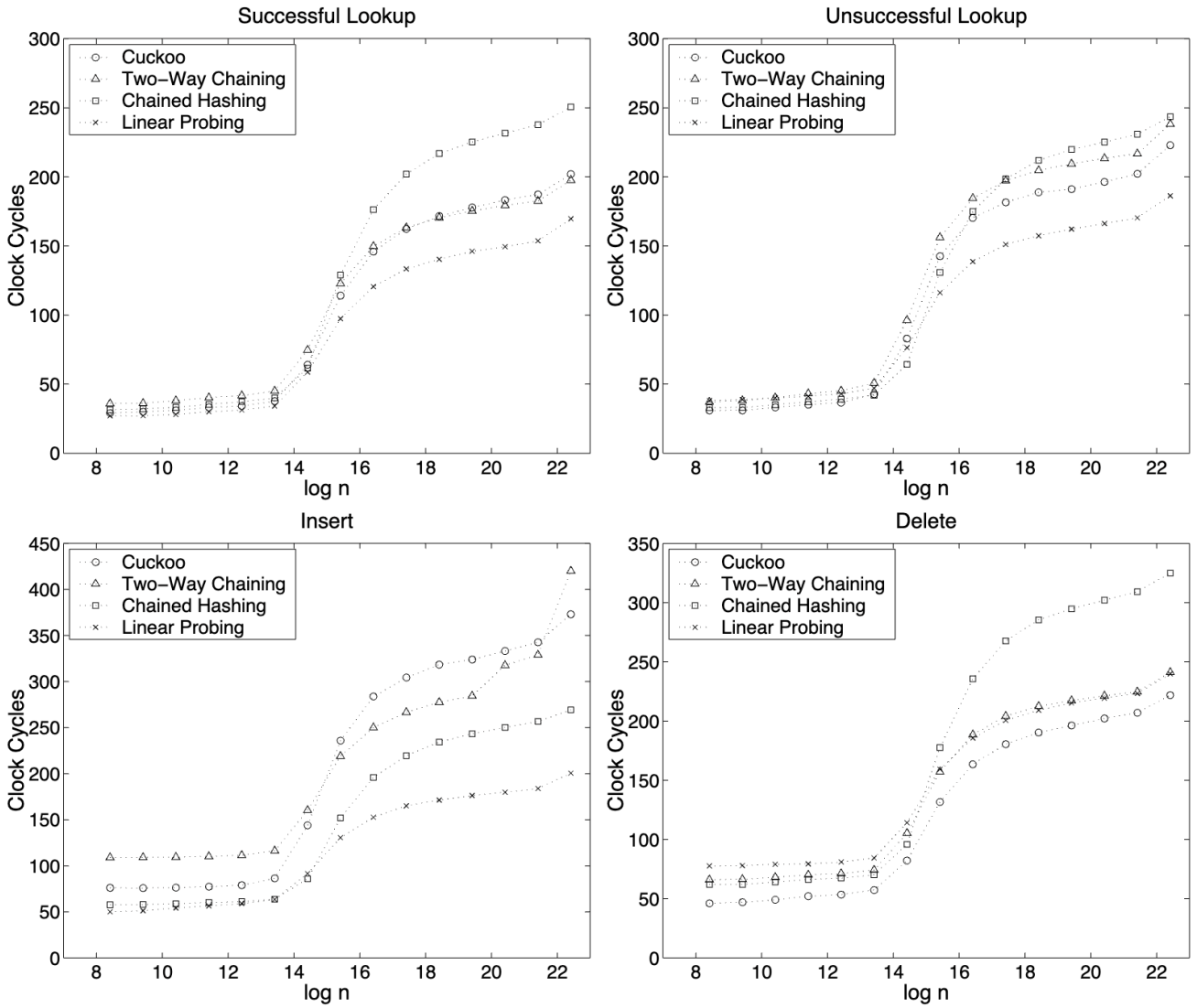


Figure 1: (a) Key x is successfully inserted by moving keys y and z to the other table. (b) Key x cannot be accomodated and a rehash is necessary.

Insertions follow the behavior of a cuckoo bird laying its egg in another bird's nest—hence the name.

Cuckoo Hashing

- Performance Characteristics
  - Worst-case O(1) lookup time (only two cell accesses; can be parallelized)
  - Expected amortized O(1) insertion and deletion
  - Space usage of about three words per key in practice
  - High practical performance, comparable to well-optimized classical hash tables



State-of-the-arts (Hash for Persistent Memory)

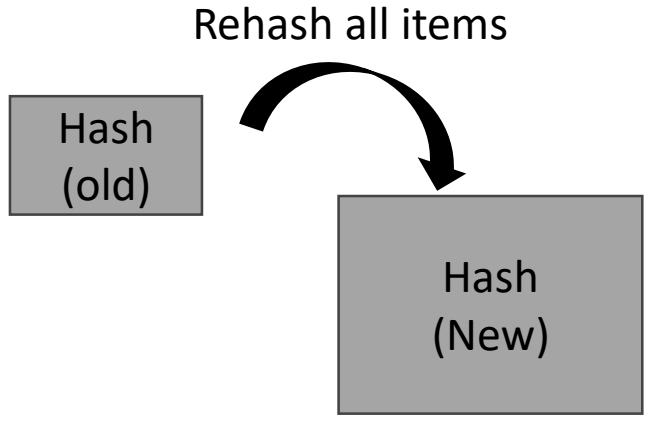
- (Level) Write-Optimized and High-Performance Hashing Index Scheme for Persistent Memory [OSDI'18]
- (CCEH) Write-Optimized Dynamic Hashing for Persistent Memory [Fast'19]
- SEPH: Scalable, Efficient, and Predictable Hashing on Persistent Memory [OSDI'23]

State-of-the-arts (Hash for Persistent Memory)

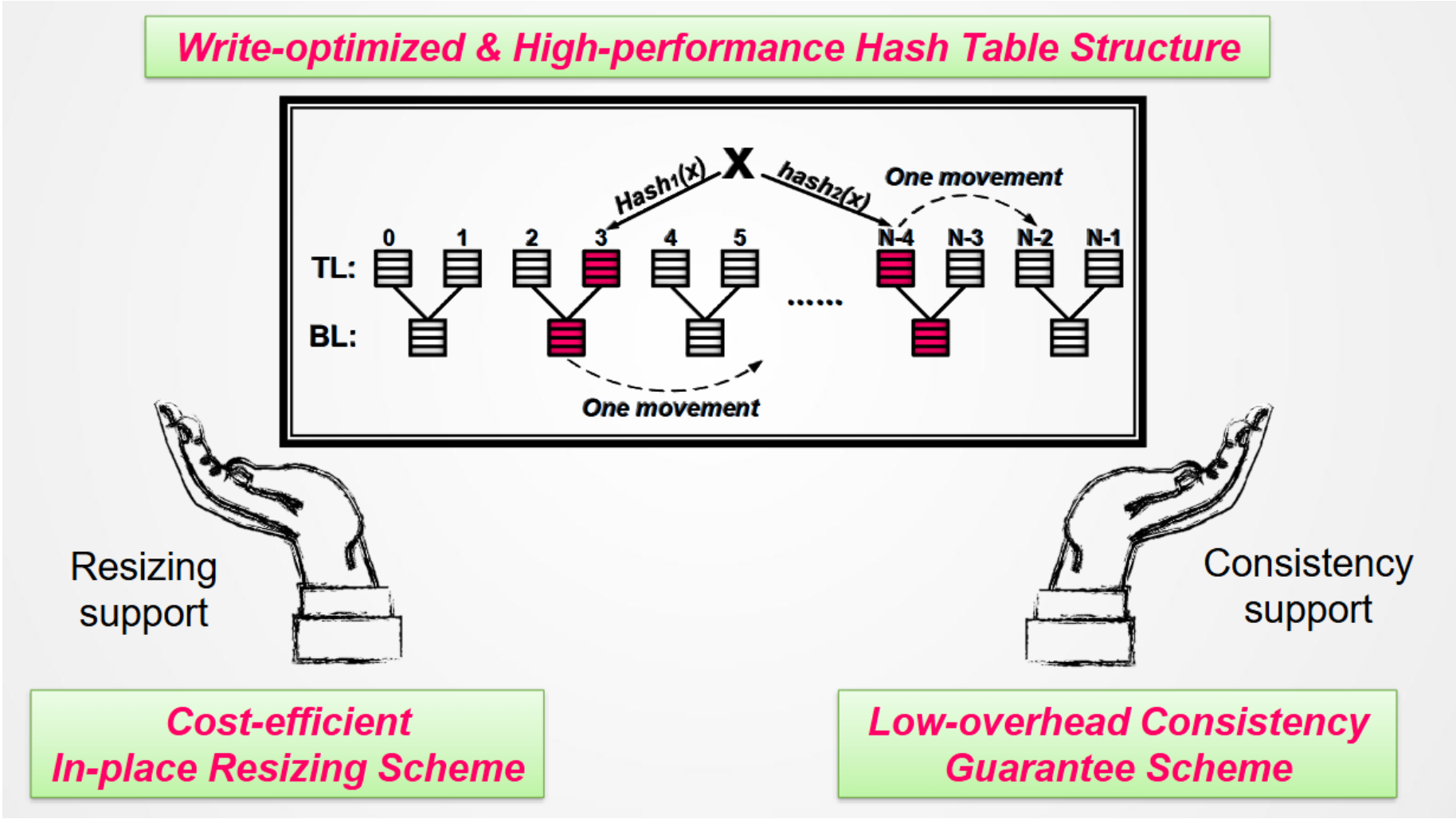
- **(Level) Write-Optimized and High-Performance Hashing Index Scheme for Persistent Memory [OSDI'18]**
- (CCEH) Write-Optimized Dynamic Hashing for Persistent Memory [Fast'19]
- Dash: Scalable Hashing on Persistent Memory [PVLDB'20]
- SEPH: Scalable, Efficient, and Predictable Hashing on Persistent Memory [OSDI'23]

Challenges

- Challenges of Hashing Indexes for PM
  - **High overhead for consistency guarantee**
    - Ordering memory writes
      - Cache line flush and memory fence instructions
    - Avoiding partial updates for non-atomic writes
      - Logging or copy-on-write (CoW) mechanisms
  - **Performance degradation for reducing writes**
    - Hashing schemes for DRAM usually cause many extra writes for dealing with **hash collisions**
  - **Cost inefficiency for resizing hash table**
    - Double the table size and iteratively rehash all items
    - Take O(N) time to complete
    - **N insertions** with cache line flushes & memory fences



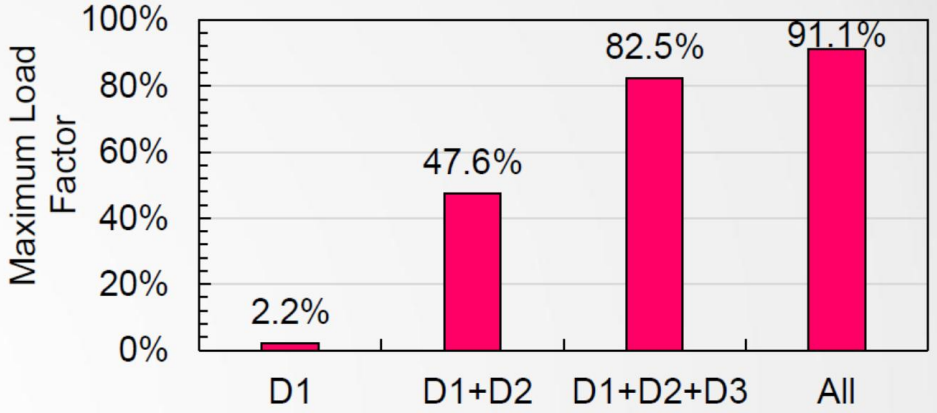
Level Hashing

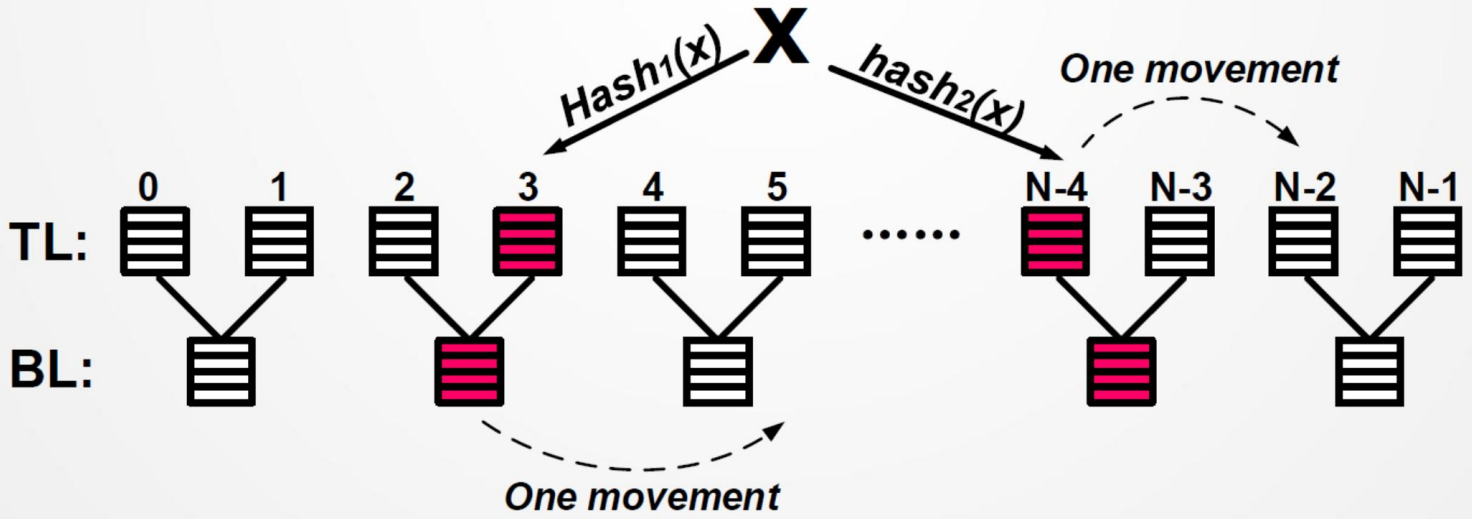


https://www.usenix.org/sites/default/files/conference/protected-files/osdi18_slides_zuo.pdf#page=9.00

Write-optimized Hash Table Structure

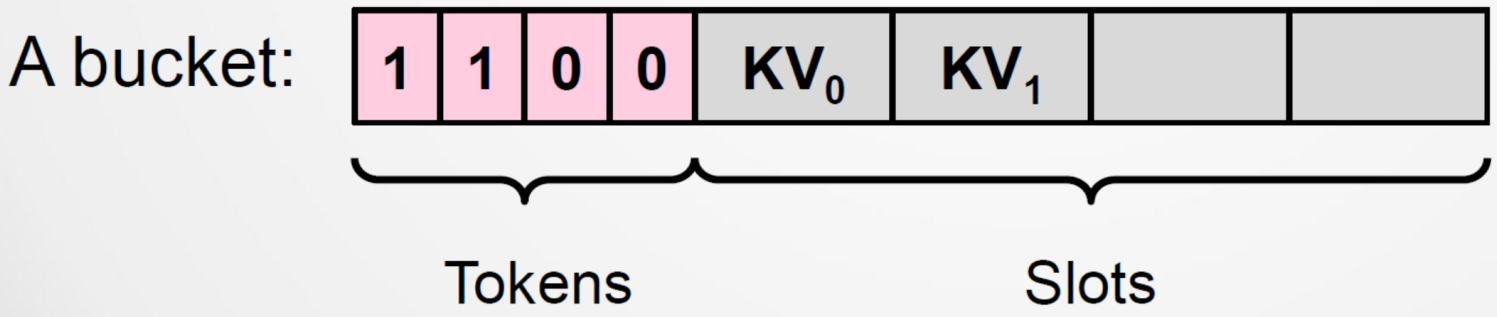
① Multiple slots per bucket
② Two hash locations for each key
③ Sharing-based two-level structure
④ **At most one movement for each successful insertion**





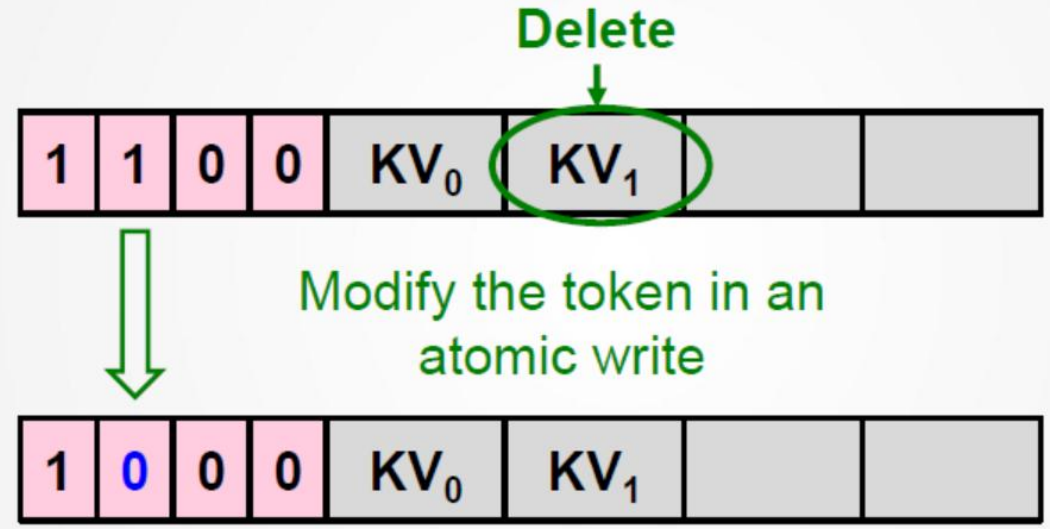
Low-overhead Consistency Guarantee

- A token associated with each slot in the open-addressing hash tables
  - Indicate whether the slot is empty
  - A token is 1 bit, e.g., “1” for non-empty, “0” for empty
- Modifying the token area only needs an atomic write
  - Leveraging the token to perform log-free operations



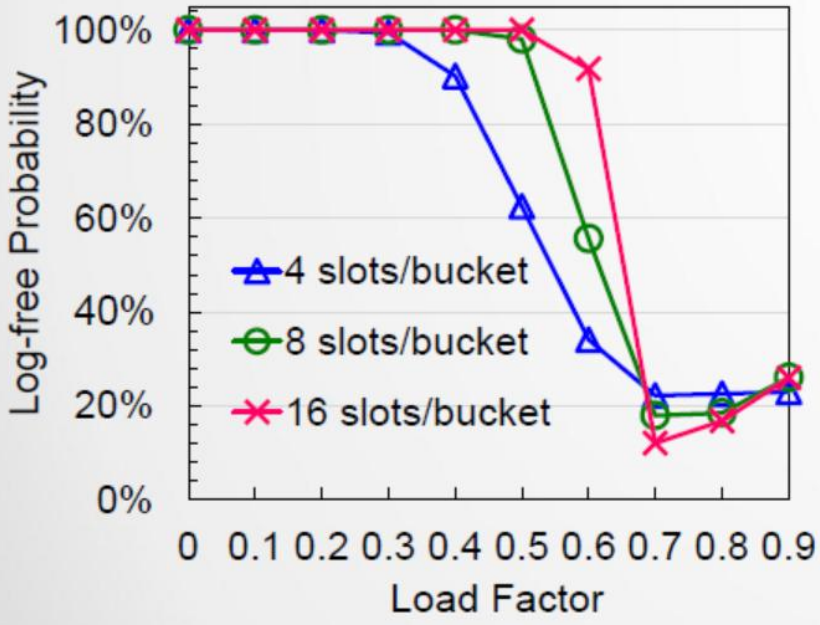
Log-free Delection

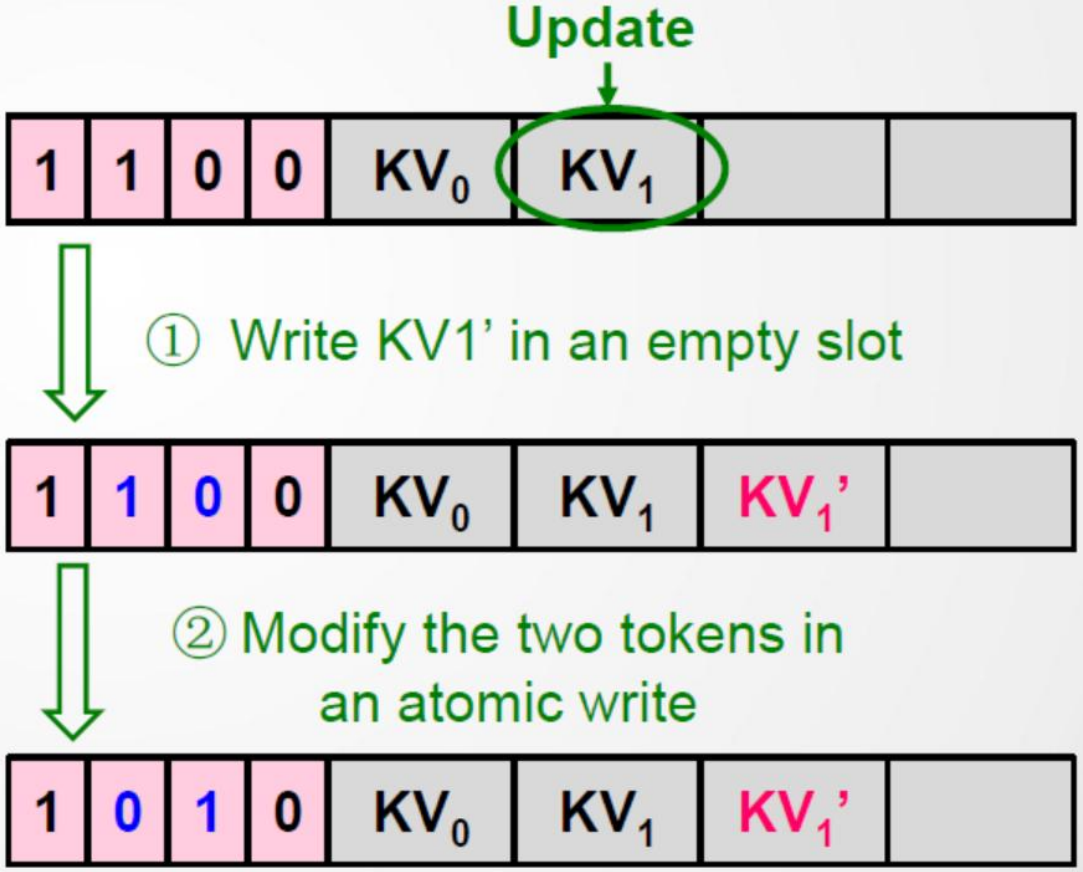
- Delete an existing item



Log-free Update

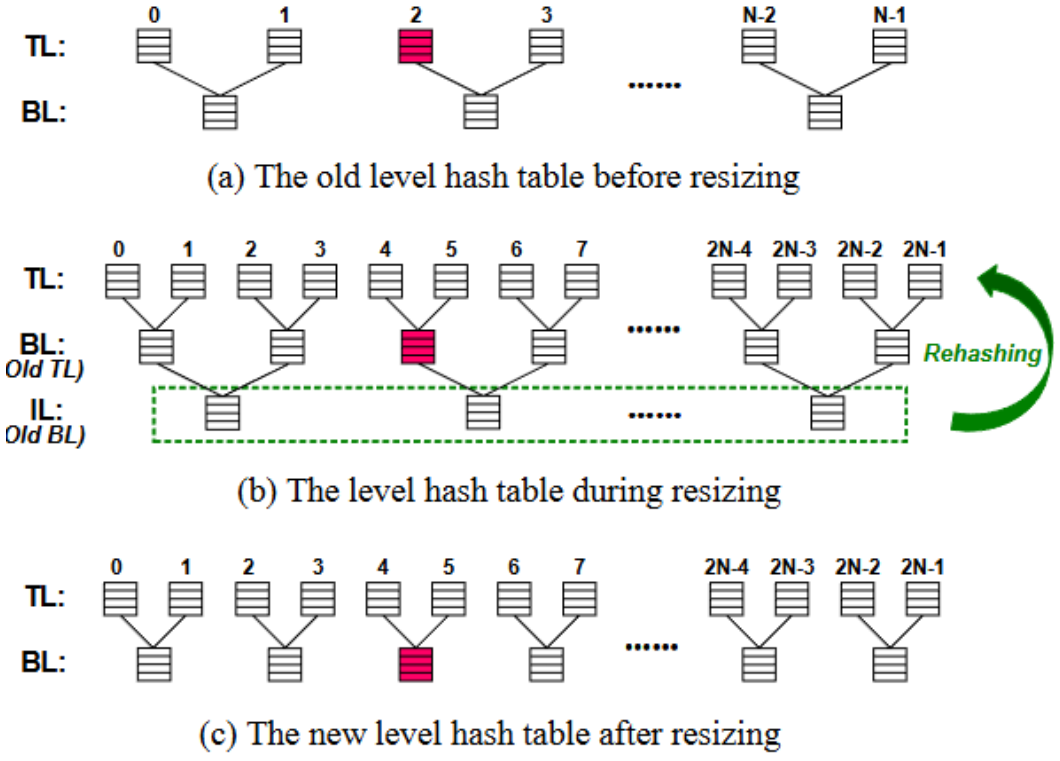
- Our scheme: check whether there is an empty slot in the bucket storing the old item
  - Yes: log-free update
  - No: using logging

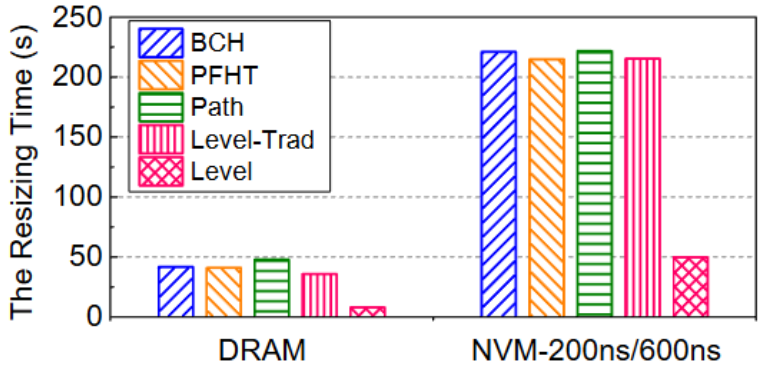




(Level) Write-Optimized and High-Performance Hashing Index Scheme for Persistent Memory [OSDI'18]

- Uses a **Top Level (TL)** and **Bottom Level (BL)** with multiple slots per bucket.
- At-Most-One-Movement Insertion
  - Any successful insertion triggers **at most one item relocation**.
  - Greatly reduces PM writes.
  - Only ~1.2% of insertions cause a single move.
- In-Place Incremental Resizing
  - Resizing doubles the table **by adding a new top level**, not rebuilding from scratch.
  - Only 1/3 of buckets (old bottom level) require rehashing.

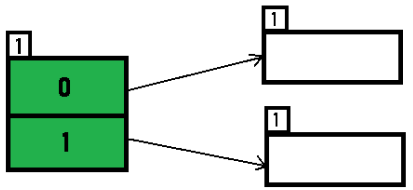


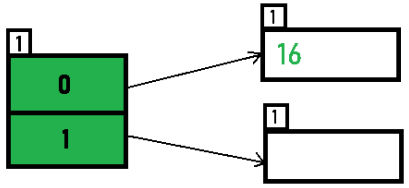


State-of-the-arts (Hash for Persistent Memory)

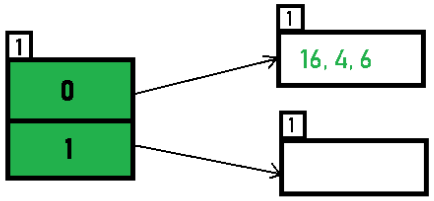
- (Level) Write-Optimized and High-Performance Hashing Index Scheme for Persistent Memory [OSDI'18]
- **(CCEH) Write-Optimized Dynamic Hashing for Persistent Memory [Fast'19]**
- **Dash: Scalable Hashing on Persistent Memory [PVLDB'20]**
- SEPH: Scalable, Efficient, and Predictable Hashing on Persistent Memory [OSDI'23]

A Running Example of Extendible Hashing





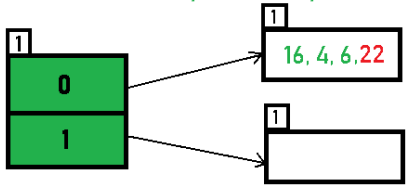
Hash(16)= 10000



Hash(4)=100
Hash(6)=110

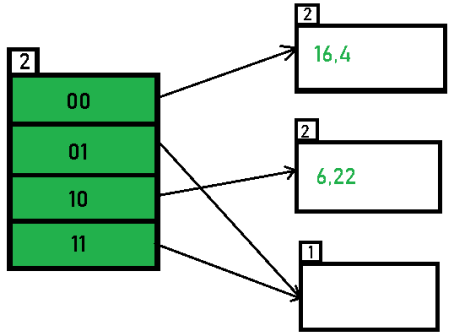
OverFlow Condition

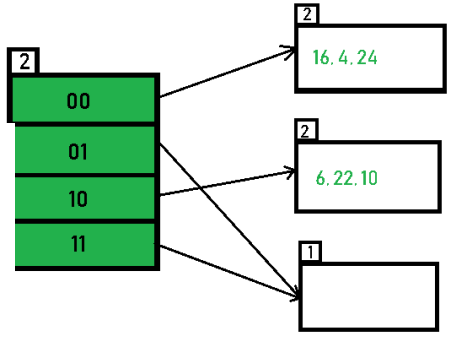
Here, Local Depth=Global Depth



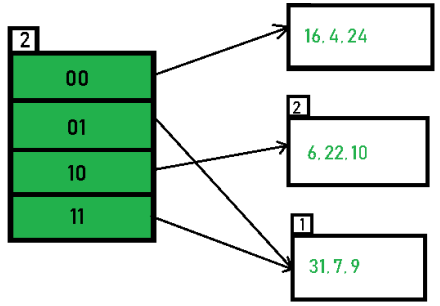
Hash(22)=10110

After Bucket Split and Directory Expansion



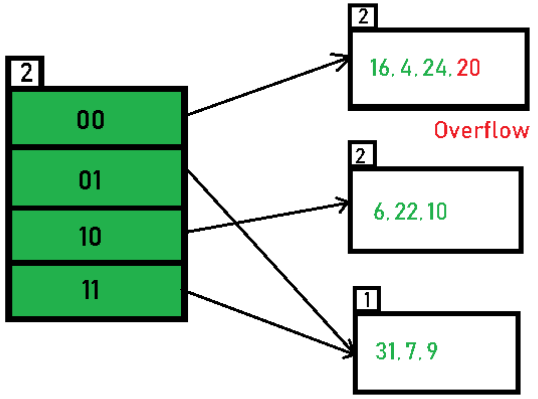


Hash(24)= 11000
Hash(10)=1010

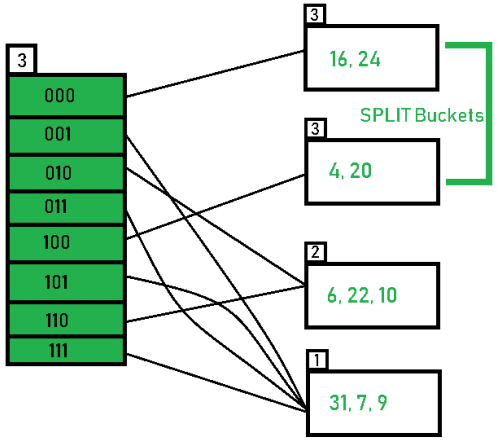


Hash(31)= 11111
Hash(7)= 111
Hash(9)= 1001

OverFlow, Local Depth= Global Depth

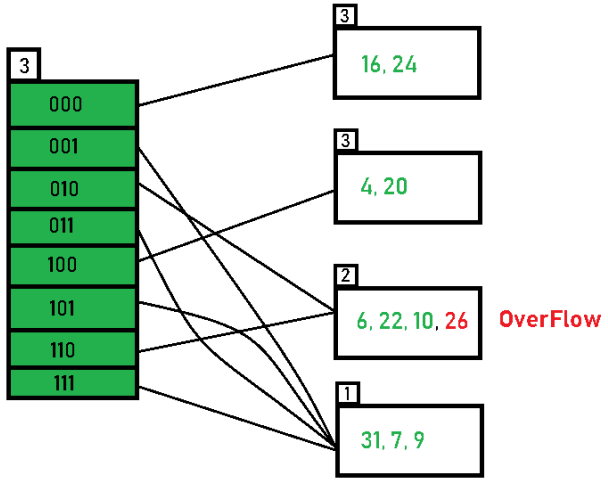


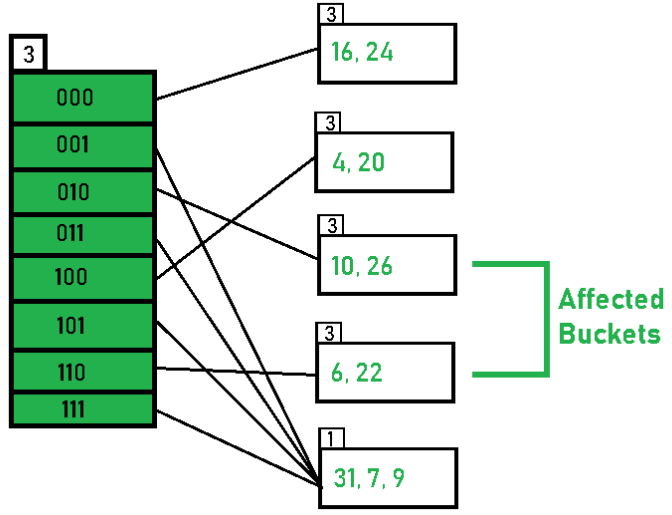
Hash(20)=10100



Hash(26)=11010

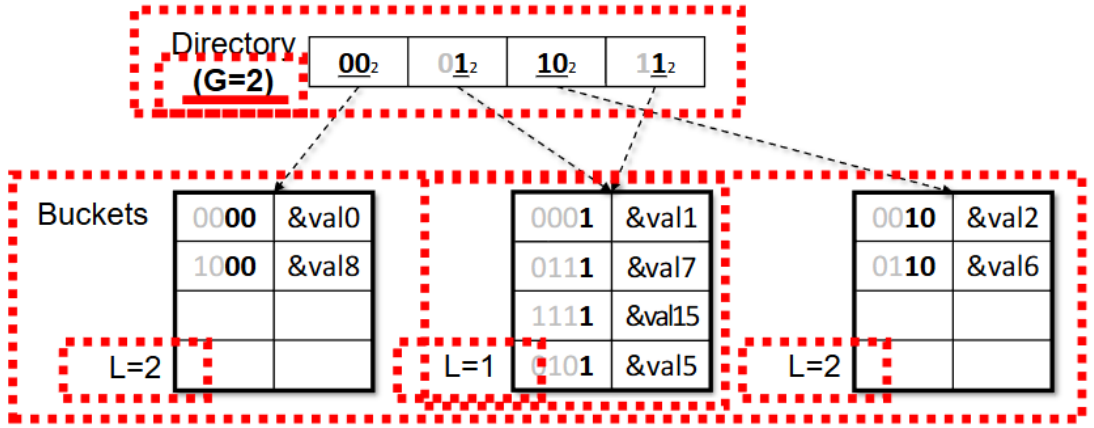
OverFlow, Local Depth <Global Depth

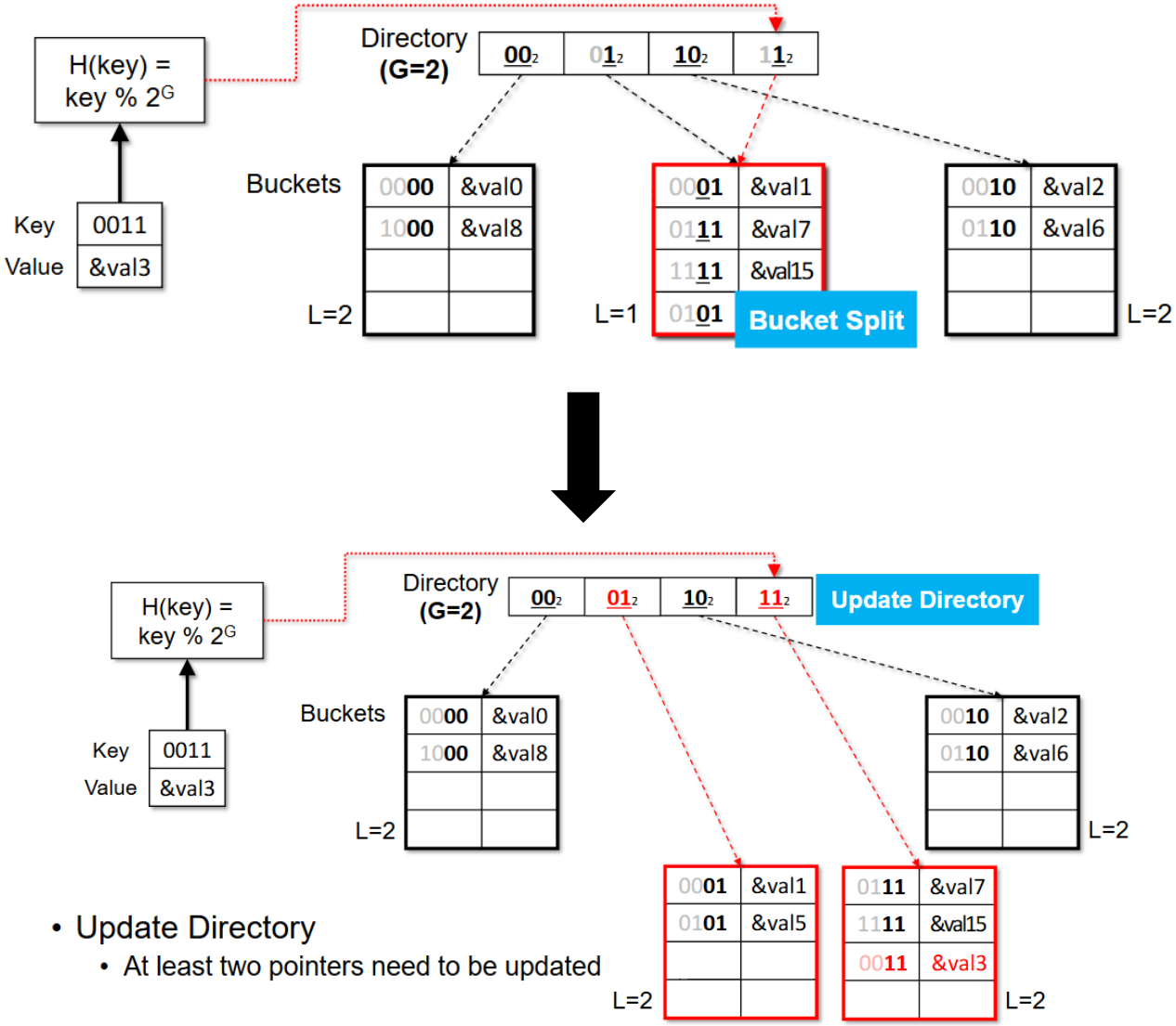




Extendible Hashing

- Extendible Hashing (1979)
  - In extendible hashing, re-hashing is an incremental operation
  - For **time-sensitive applications** that need to be less affected by **full-table rehashing**
- **Any limitation?**



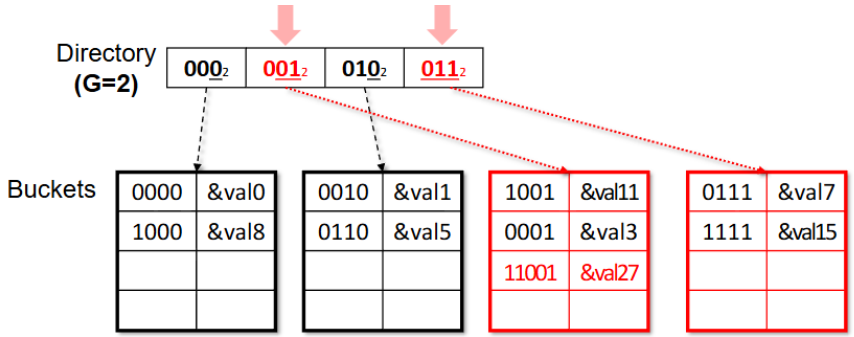


Characteristics of Extendible Hashing

- Advantages:
  - Data retrieval is less expensive (in terms of computing).
  - No problem of data-loss since the storage capacity increases dynamically.
  - With dynamic changes in hashing function, associated old values are rehashed w.r.t the new hash function.
- Limitations:
  - The directory size may increase significantly if several records are hashed on the same directory while keeping the record distribution non-uniform.
  - Size of every bucket is fixed.
  - Memory is wasted in pointers when the global depth and local depth difference becomes drastic.
  - This method is complicated to code.

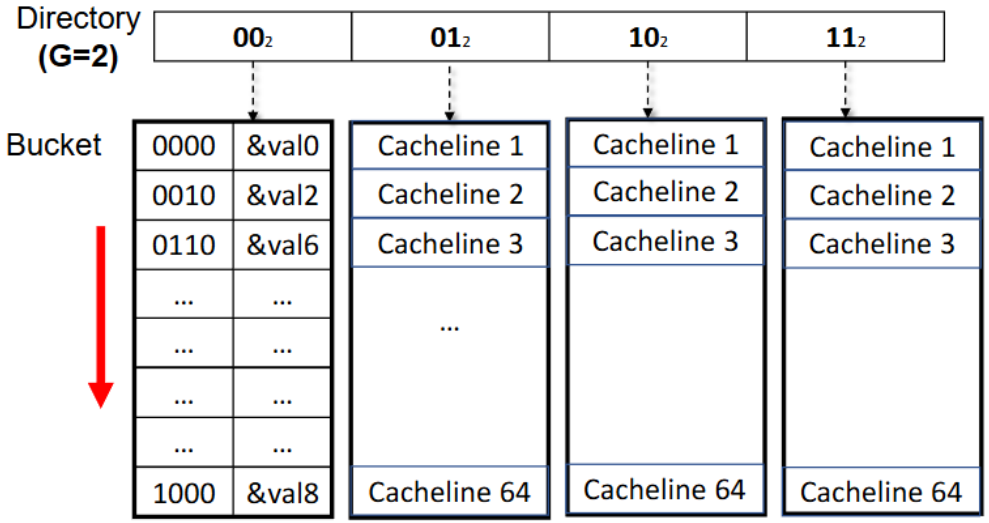
CCEH

- CCEH: Cacheline-Conscious Extendible Hashing
- 3-Level Structure
  - Introduce an intermediate level, Segment
  - Lookup via only two cacheline accesses
- Failure-atomic Directory Updates
  - Introduce the split buddy tree to manage split history
- Failure-atomic Segment Split
  - Lazy deletion scheme to minimize dirty writes



Problems

Split operation updates multiple pointers → Not Failure-Atomic

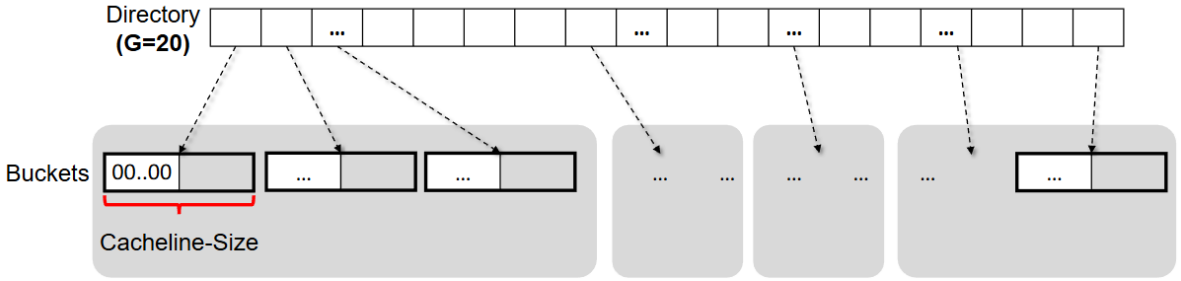


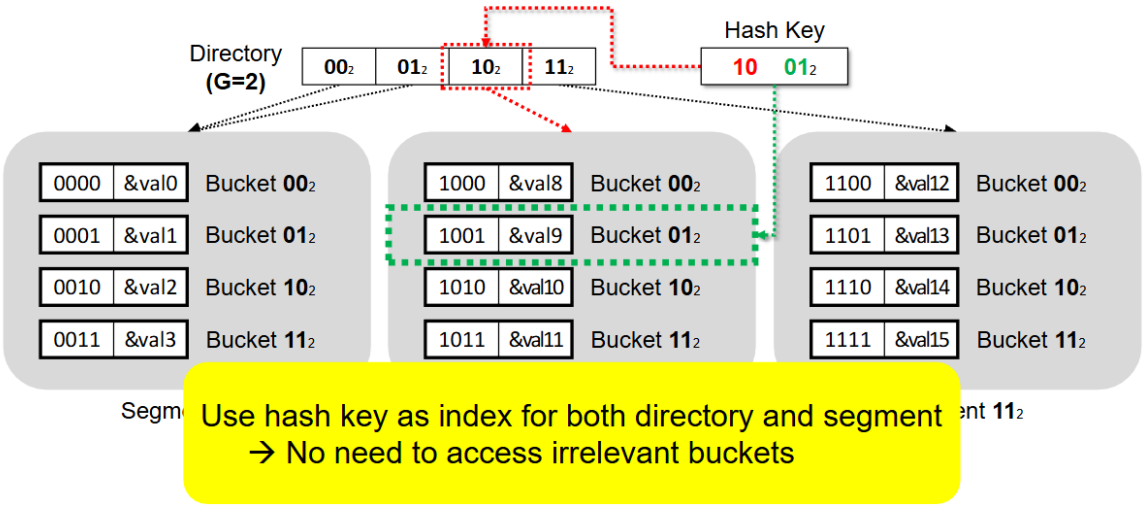
Problems

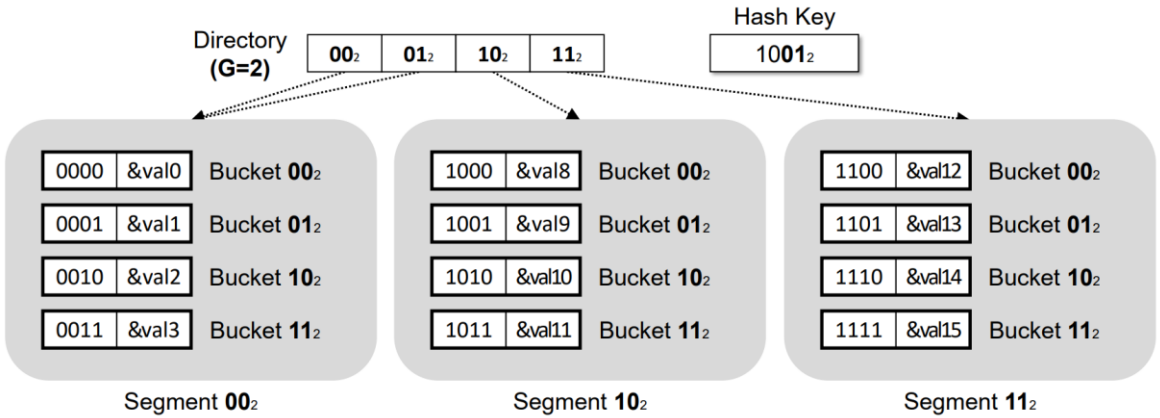
- Page-sized bucket → 64 cacheline accesses per bucket

CCEH (Cont.)

- A group of multiple cacheline-sized buckets = Segment
  - Directory → Segment → Cacheline-sized Bucket





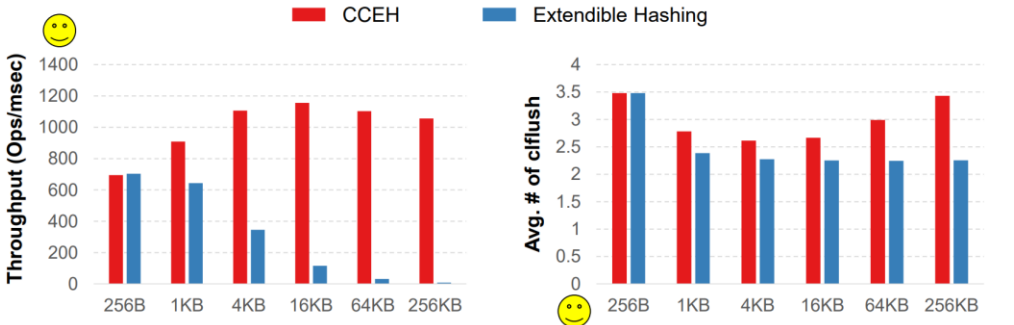


Any drawback?

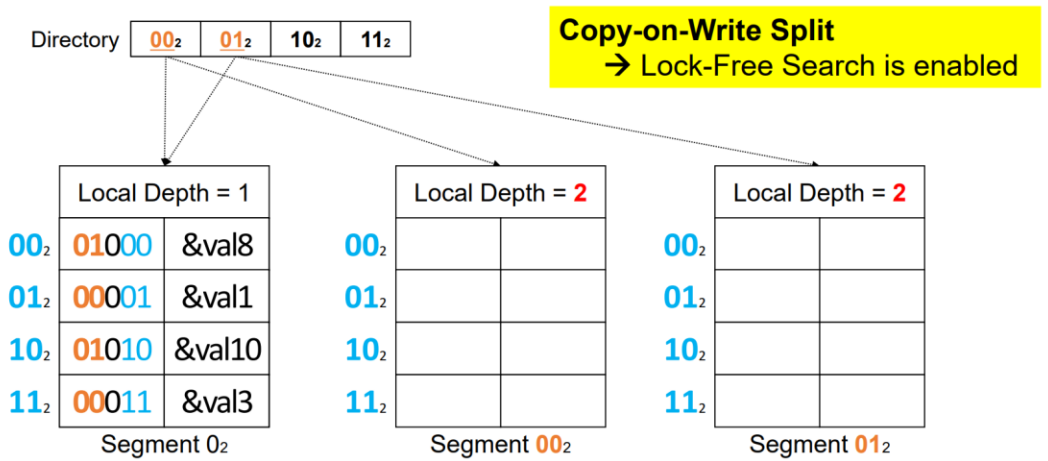
Using intermediate level "**Segment**",
CCEH reduces directory size while keeping bucket size small

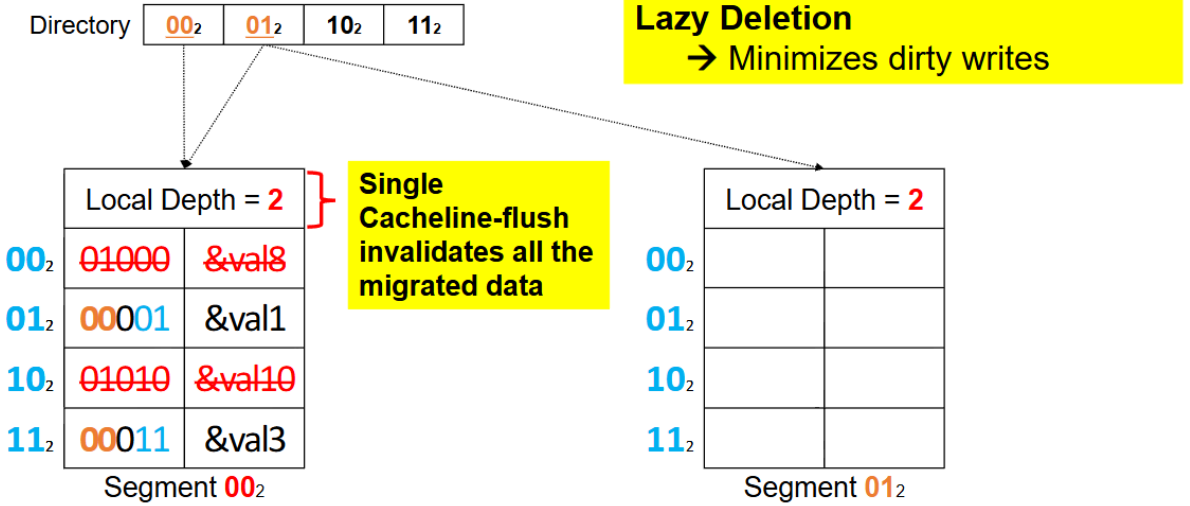
CCEH (Cont.)

- Copy-on-Write Split
  - Enable lock-free search, avoids in-place modification.
- Lazy Deletion
  - A single cacheline flush invalidates migrated entries.
  - Dramatically reduces dirty writes during split.



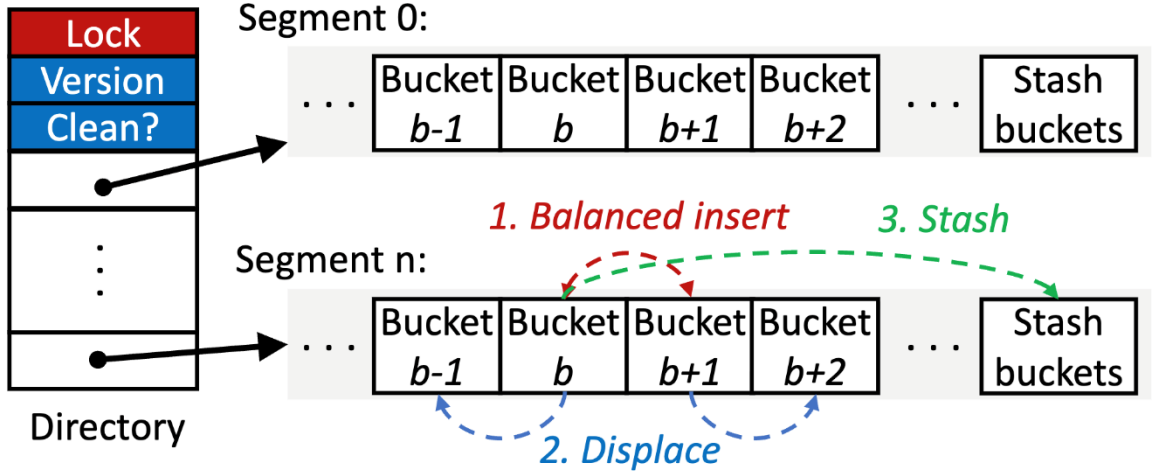
CCEH compared to legacy Extendible Hashing
Cons: Low utilization and more cacheline flushes due to hash collisions
Pros: Constant number of cacheline accesses with varying directory size

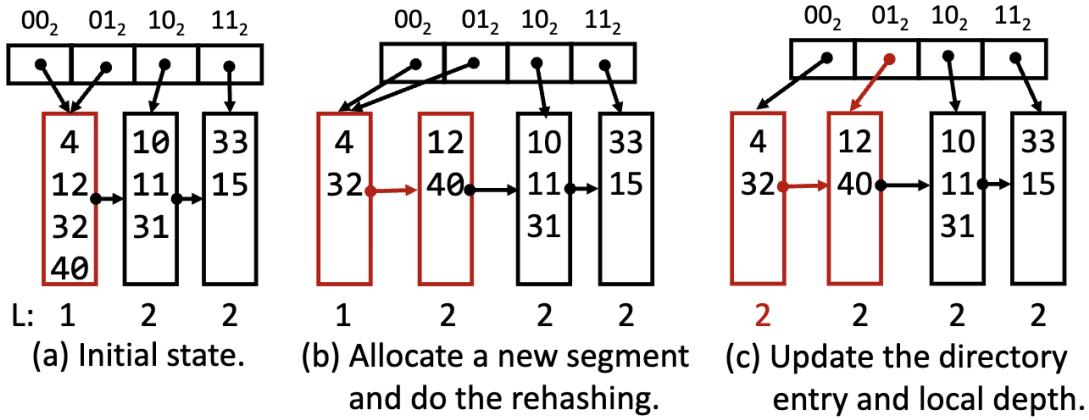




Dash: Scalable Hashing on Persistent Memory

- Core Design Principles of DASH
  - Avoid Unnecessary PM Reads and Writes
    - **Fingerprinting** to avoid reading full keys and unnecessary bucket scans
    - **Compact metadata** to detect negative probes early
  - Lightweight Concurrency
    - **Traditional locking causes extra PM writes**.
- DASH Overview: Architecture and Key Components
  - Builds on Extendible Hashing, using:
    - A directory of pointers
    - A set of segments, each containing multiple buckets
    - Normal buckets + stash buckets for overflow handling

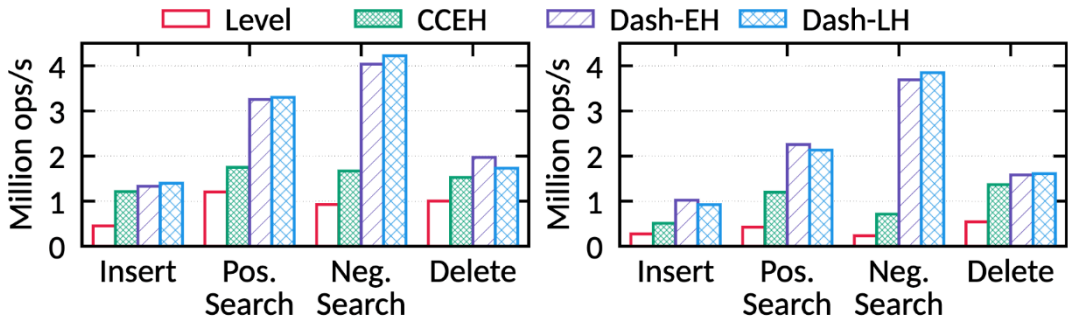


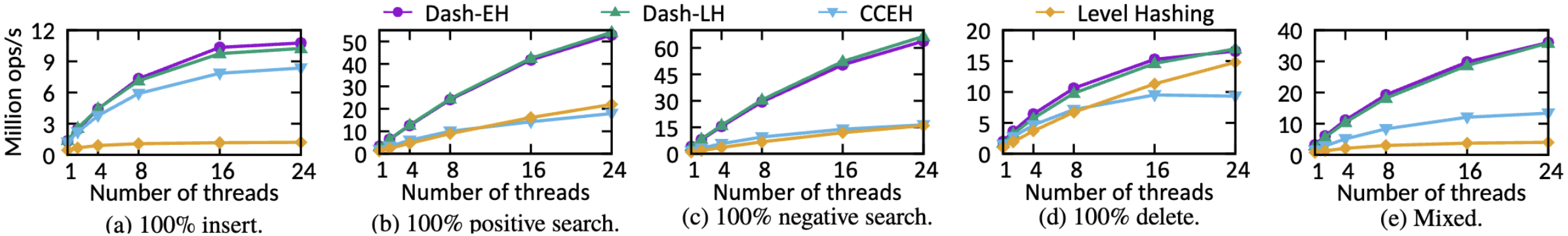


Dash: Scalable Hashing on Persistent Memory

- Positive/Negative Search
  - Dash is **faster** than CCEH and Level Hashing
    - Fingerprinting avoids unnecessary key comparisons → dramatically fewer PM reads
    - Fingerprint + overflow metadata allows early rejection without scanning full buckets
- Insert Performance: Comparable to CCEH, Much Better than Level

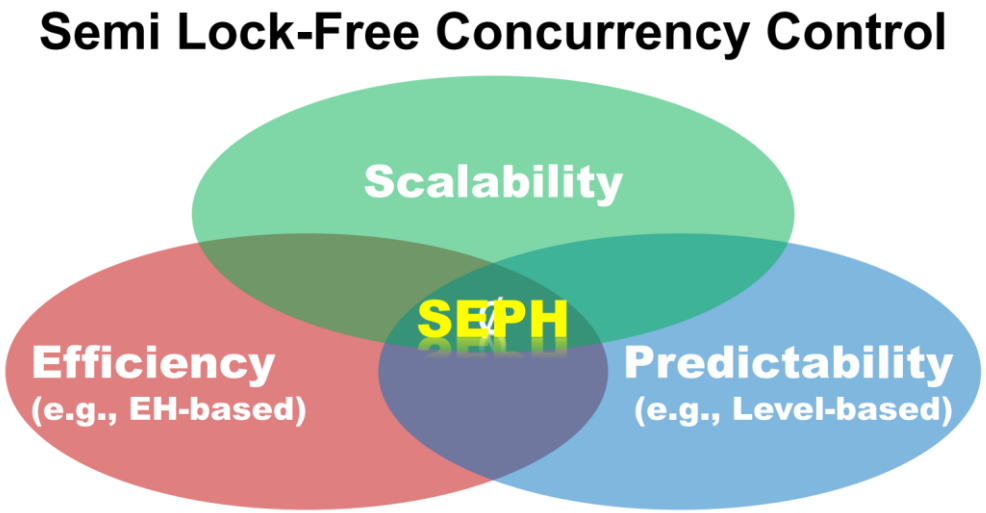
On real Intel Optane DCPMM:
Dash scales with up to ~3.9× better performance than prior stateof-the-art, while maintaining desirable properties, including high load factor and sub-second level instant recovery.





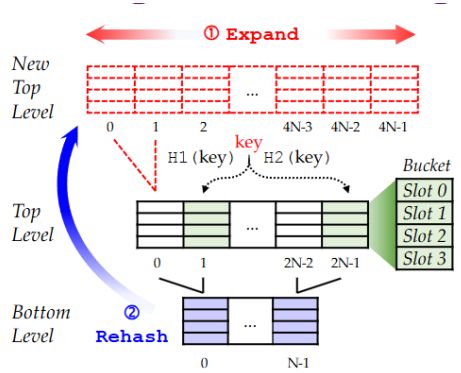
State-of-the-arts (Hash for Persistent Memory)

- (Level) Write-Optimized and High-Performance Hashing Index Scheme for Persistent Memory [OSDI'18]
- (CCEH) Write-Optimized Dynamic Hashing for Persistent Memory [Fast'19]
- Dash: Scalable Hashing on Persistent Memory [PVLDB'20]
- **SEPH: Scalable, Efficient, and Predictable Hashing on Persistent Memory [OSDI'23]** https://www.usenix.org/conference/osdi23/presentation/wang-chao

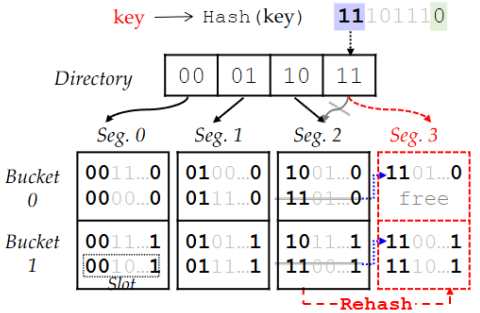


Observation

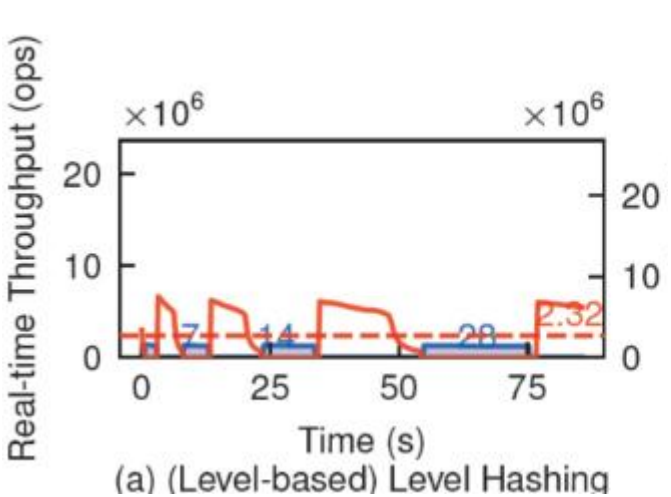
- Indexes/algorithms need to be “re-tailored” for PM
- **Existing PM Hashing Schemes**
- **(we just talk about them!)**
  - Level-based PM Hashing, e.g., Level Hashing [OSDI’18]
  - Extendible Hashing (EH)-based, e.g., CCEH [FAST’19]
- **However!!!**
  - Existing PM hashing schemes face the dilemma between the performance **efficiency** and **predictability**
  - **Performance scalability** is limited due to excessive writes in handling concurrency control.

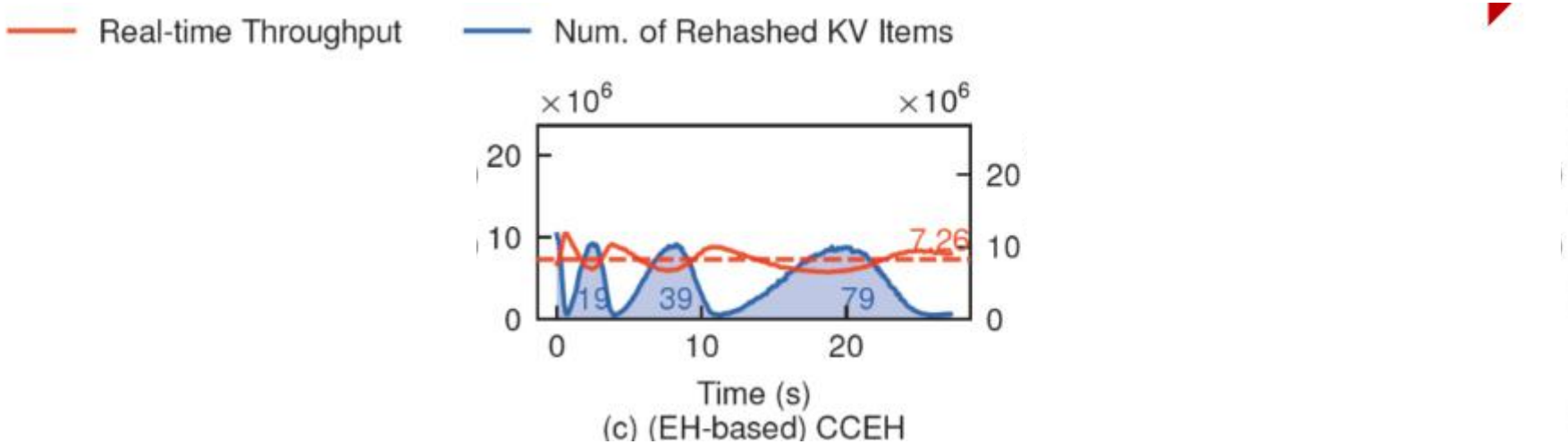


• Level-based PM Hashing



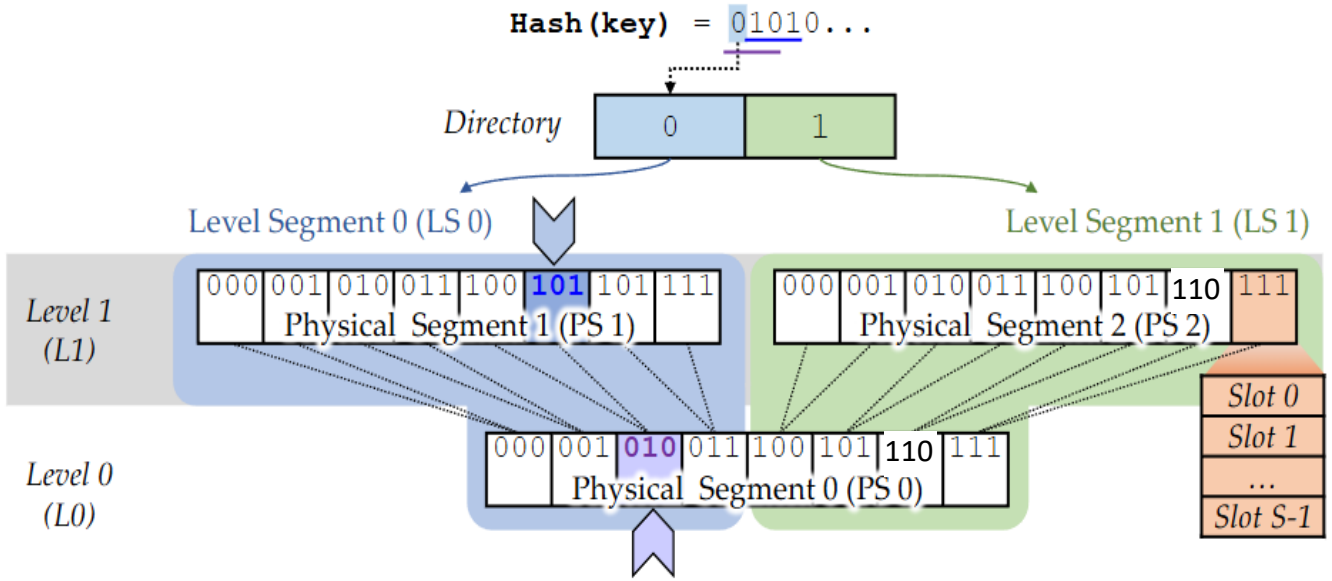
• EH-based PM Hashing





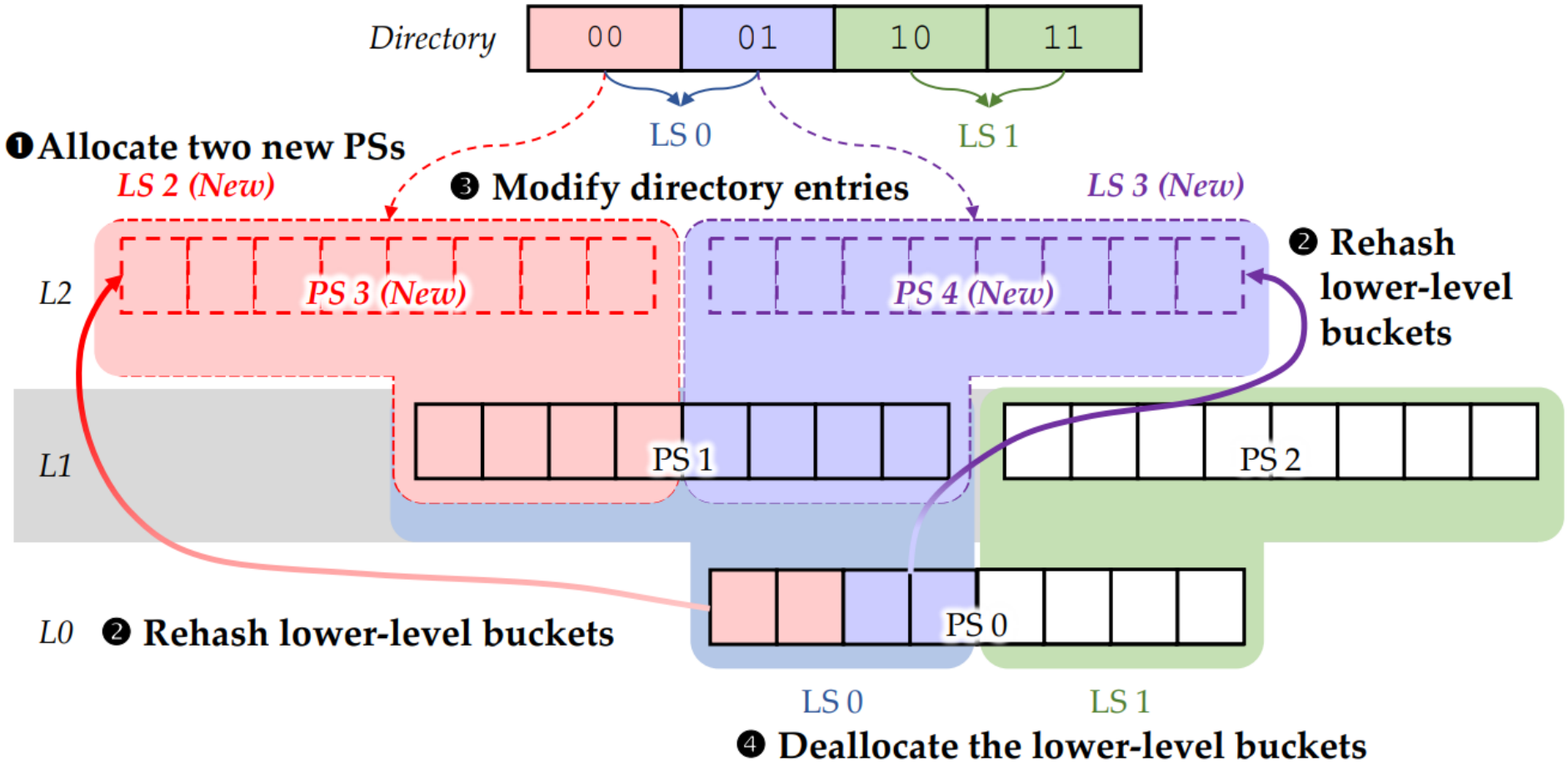
SEPH: Scalable, Efficient, and Predictable Hashing on Persistent Memory

- **Level Segment (LS) Architecture**
  - LS = **Level-based logical structure** + **Physical Segment (PS)** data layout.
    - Cacheline-conscious lookup (high efficiency)
    - Low-overhead resizing (high predictability)
- **One-Third Split (likes level-based)**



One-third Split

– Splits one LS into two, but only rehashes "1/3" of the KV items



Summary

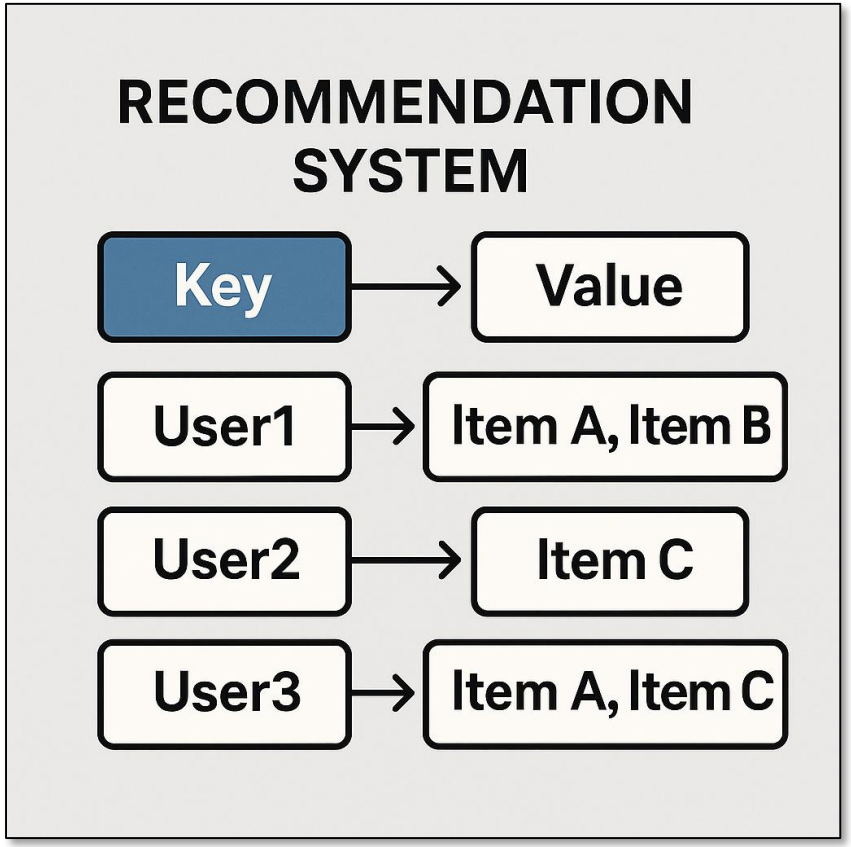
Design Type	Core Structure	Strengths	Weaknesses
EH-based (e.g., CCEH / Dash)	Segment + Directory	• Cacheline-friendly lookup • High efficiency	• Directory manipulation overhead • Large-scale rehashing during splits
Level-based (e.g., Level / Clevel)	Two-level shared buckets	• Predictable & cost-efficient resizing • Minimal data movement	• Throughput instability due to load-factor fluctuation • Sensitivity to bucket-level imbalance
Others **SEPH** (OSDI'23)	**Level Segment (LS): Hybrid of Level + Segment**	• Predictability + segment efficiency • Stable throughput • Low-overhead resizing	?

Outline

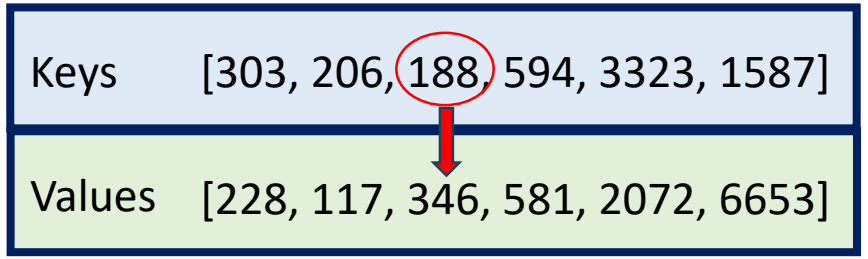
- Hash
- **Learned index**

Introduction

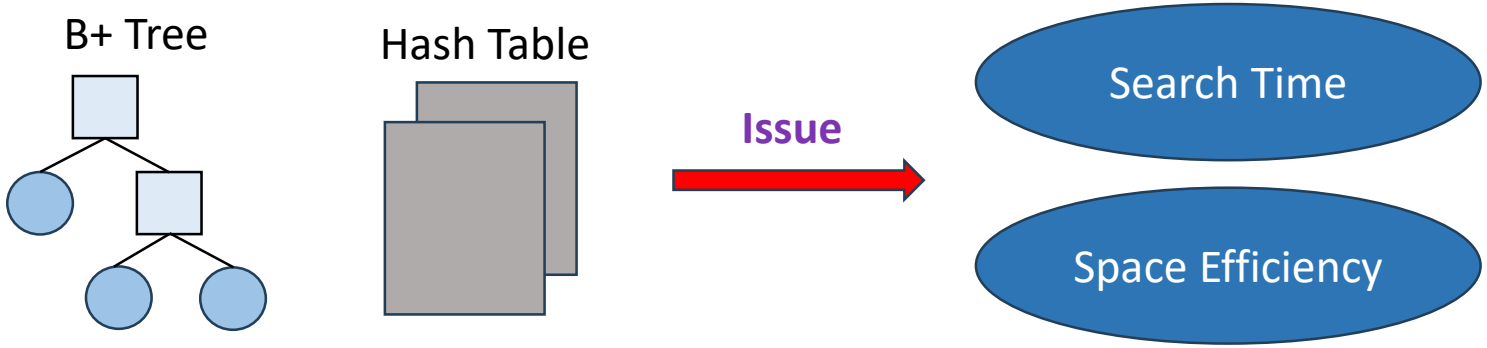
- **Why Learned Index Matters**
  - **Key-value storage: The core of data access in database system**
    - e.g., Recommendation System, User Profile Management, Product Information



Given a key, our goal is to retrieve its corresponding value.

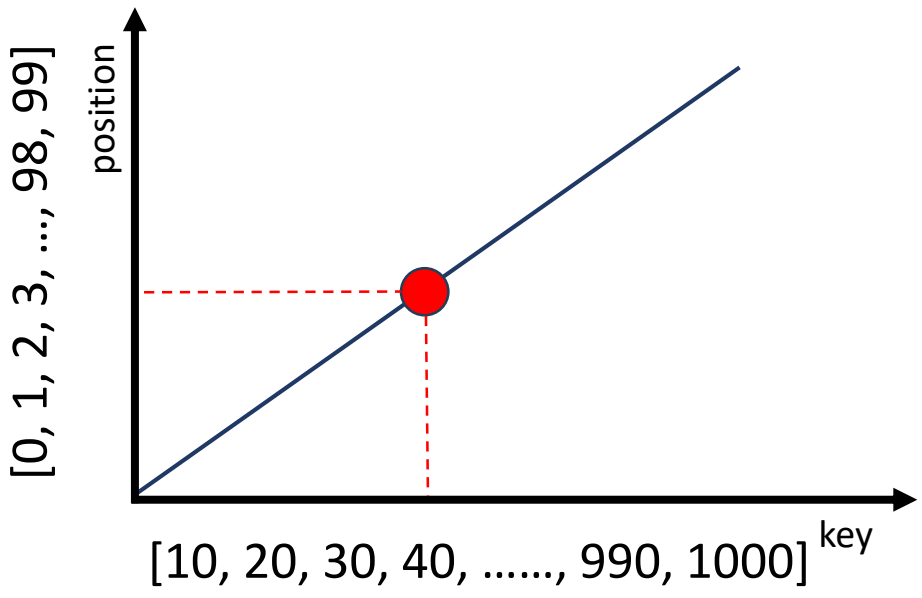


How do we know the position of the key?



Introduction

- **How Learned Index models retrieve the value**
  - **Ideal scenario**
    - After sorting, keys often exhibit a near-linear relationship with their positions
    - Learned models capture the keys' distribution and approximate the positions accordingly
    - An example: *position = a * key + b*



In this case, a = 0.1 and b = -1.

If we try to get the value of key 40

Through a simple calculation:

position = 0.1 * 40 + (-1) = 3

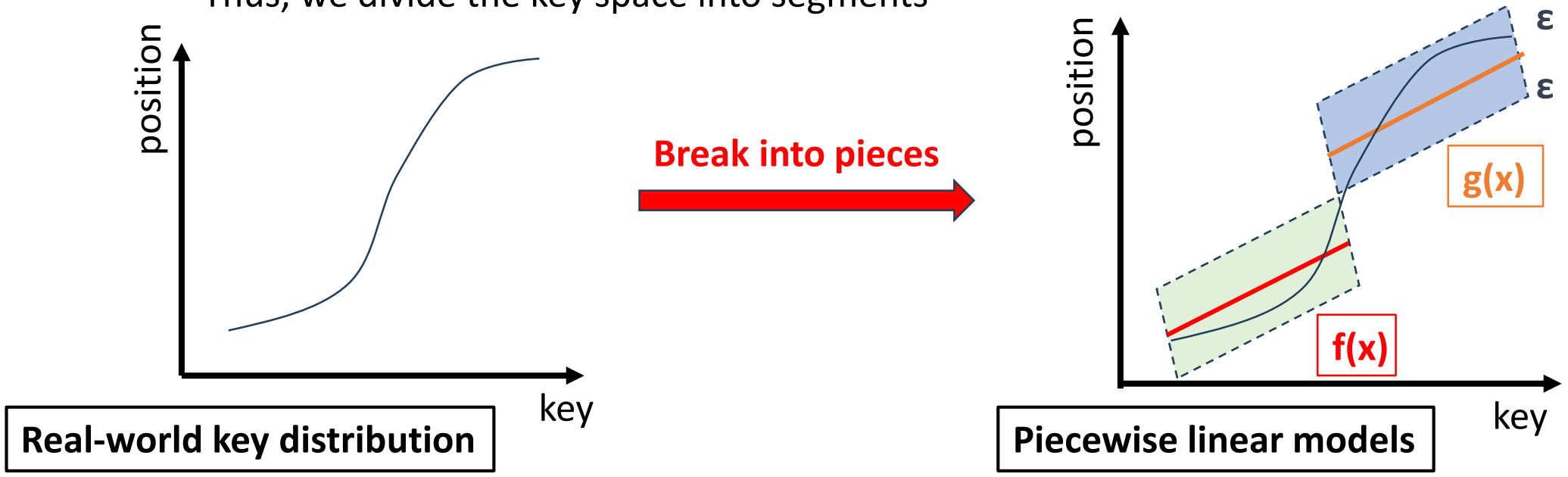
Then access Values[3] to retrieve the value

How to Build a Learned Index

How to Build a Learned Index

Piecewise linear model

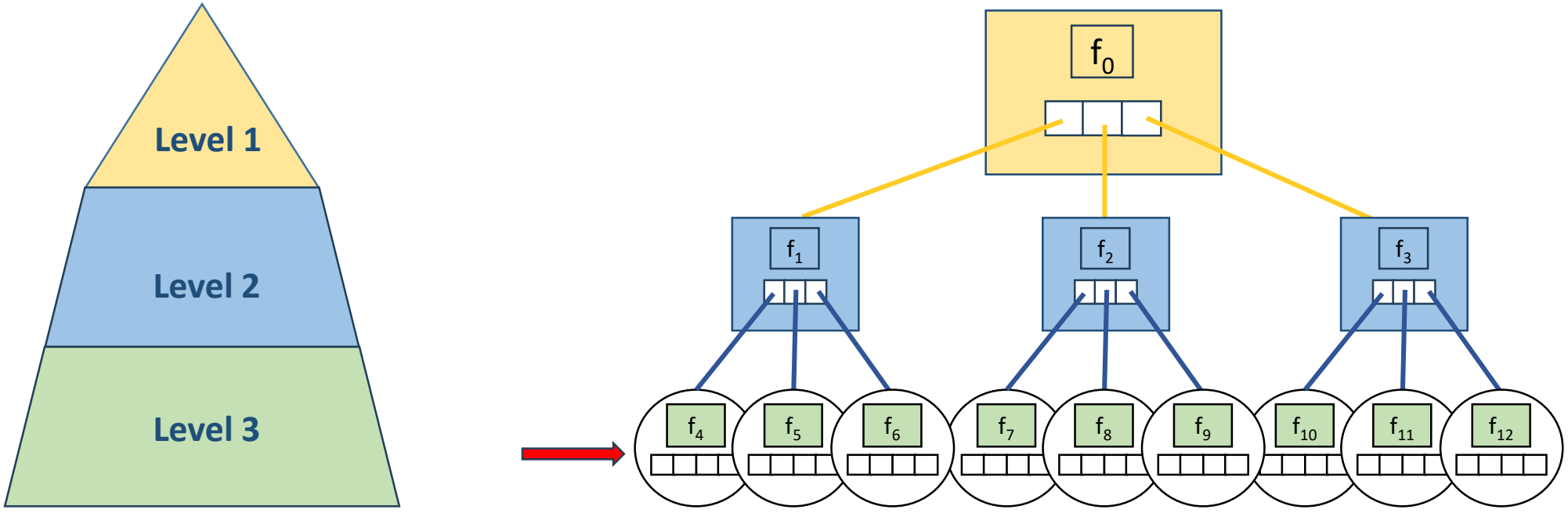
- A simple linear model (e.g., y = ax + b) works well **if the data follows a uniform trend**
- A single line might not accurately capture complex distributions
- Thus, we divide the key space into segments



Recursively partition the key space until each model achieves sufficient accuracy.

How to Build a Learned Index

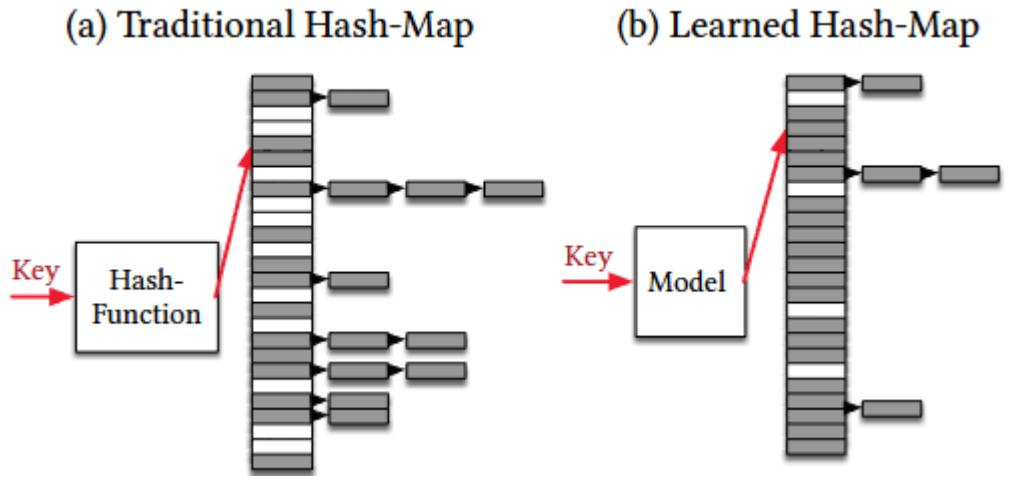
- **How to Build a Learned Index**
  - **Recursive Model Index (RMI)**
    - The top-level model predicts which sub-model to use
    - The leaf-level model then predicts the approximate position



After recursive segmentation, the approximate position of a key is predicted in a leaf model.

Traditional Hash-map vs Learned Hash-map

- Hash-map: bucket = hash(key)
- **Learned Hash-map** does:
  - predicted_pos = model(key) ← regression model
  - bucket = predicted_pos / bucket_range
- **How the Model Works**
  - A small regression model approximates the **CDF of the keys**.
  - Keys that are larger map to larger predicted positions.
  - Produces a mapping that is **more aligned with the actual distribution**, rather than random.
- **Advantages Over Traditional Hashing**
  - **Much fewer collisions** due to CDF-aware bucket placement
  - **Higher cache efficiency** (fewer probes)
  - **Simple model** (linear regression) with negligible runtime cost
  - Compatible with existing hash-table structures



	% Conflicts Hash Map	% Conflicts Model	Reduction
Map Data	35.3%	07.9%	77.5%
Web Data	35.3%	24.7%	30.0%
Log Normal	35.4%	25.9%	26.7%

What are the issues with learned indexes?

Kraska, Tim, et al. "The case for learned index structures." *Proceedings of the 2018 international conference on management of data*. 2018.


thank you
danke
рахмат
謝謝
ngiyabonga
teşekkür ederim
Баярлалаа
спасибо
faafetai lava
mersi
kia ora
barka
welalin
tack
spas
vinaka
спасибі
blagodaram
dank je
misaotra
matondo
paldies
grazzi
gracias
mahalo
tapadh leat
хвала
asante
manana
obrigada
tenki
murakoze
kiitos
dankie
nanni
nandri
dhanyavad
mauruuru
köszönöm
hvala
gràçie
bayarlalaa
bedankt
enkosi
akun
dankon
aciū
dziękuję
djiere dieuf
tau
mochchakkeram
дякую
mamnun
chokrane
chnorakaloutioun
gratias ago
gràcies
sulpáy
taiku
go raibh maith agat
sobodi
děkuji
sukriya
kop khun krap
arigatô
takk
dakujem
trugarez
obrigado
mèsi
didi madloba
sagolun
najis tuke
ありがとう
tanemirt
grazie
rahmet
shukriya
mercé
мерси
kam sah hamnida
rahmat
terima kasih
diolch
dhanyavadagalu
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ
merci