Beyond Bandwidth Doubling: Embrace Bit-Flips & Unlock Processing-in-NAND

Maximilian Berens, Yun-Chih Chen, Jian-Jia Chen, Jens Teubner
TU Dortmund University
prename.surname@cs.tu-dortmund.de

***Abstract*—NVMe SSDs offer unprecedented capacity and bandwidth and new PCIe standards promise even more. However, the underlying technology, NAND memory, already struggles with significant heat and power consumption challenges. Just like microprocessors before, NAND also experiences Dark Silicon, preventing performance from improving at the same pace as capacity. Much of the power (and thus heat) within a NAND chip results from transferring data at a high rate, another symptom of a *compute-centric* style of processing. Therefore, we argue for *data-centric* Processing-in-NAND (PiN). However, PiN comes with significant challenges, such as limited capabilities and the need to cope with bit-flip errors. Even beyond Processing-in-Memory (PiM), databases may soon have to accept that memory is not error-free, an assumption that comes at a significant cost in power, capacity and performance. Our discussion suggests that no PiN design will serve as a singular, universally applicable solution to the bandwidth problem. Instead, successful integration into database architecture requires carefully identifying PiN-compatible functionality and abstractions, and cooperation with other innovations, such as Computational Storage and CXL. Lastly, we analyze the fundamental error tolerance of Bloom filters and binary sketches as PiM-compatible data structures, which we believe may be of independent interest.**

I. Introduction

NVMe SSDs are continually expanding their bandwidth and capacity to meet growing data demands, with PCIe 5.0 cards offering up to 12 GB/s of bandwidth and over 60 TB of capacity. This makes them not only attractive for large-scale analytic workloads, but also as "graceful extensions" of HTAP in-memory databases [1]. However, even achieving the full bandwidth of the hardware available today comes with significant challenges. For one, a full array of SSDs in a large server consumes significant power and can easily surpass the TDP of a large processing unit. For example, Micron's 6550 ION SSD[1] can draw 5-6.25W per PCIe lane and modern servers offer 96-128 lanes per socket. To cope with the immense heat, devices are known to significantly throttle their speed, which makes dedicated cooling a necessity. Additionally, the underlying technology, NAND flash memory, not only struggles with power efficiency, but also reliability [2], [3].

Yet, upcoming PCIe 6.0 and 7.0 standards promise to double bandwidth yet again. Soon, NVMe SSDs, and therefore databases as well, may face the limits of Dennard scaling, where no more performance gains can be achieved without hitting physical limitations. Just like microprocessors before, NAND has Dark Silicon, too, preventing performance from improving at the same pace as capacity [2]. Much of the power (and thus heat) within a NAND chip results from transferring data at a high rate [4], which is yet another symptom of a compute-centric style of processing.

[1]https://www.micron.com/products/storage/ssd/data-center-ssd/6550-ion

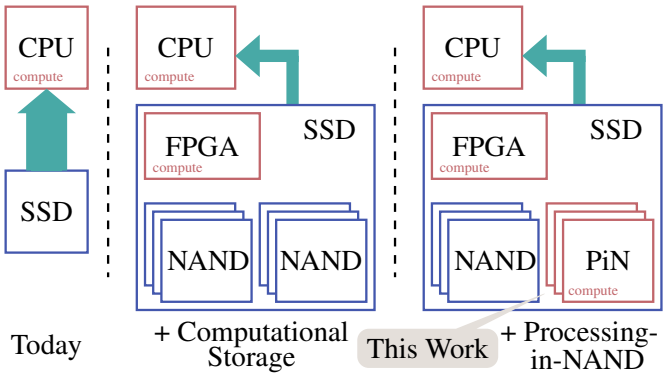

Fig. 1. Data-centric computing allows to shift the bottleneck. *Left*: von Neumann-style architecture. *Middle*: Computational Storage architecture. *Right*: Processing-in-NAND architecture.

Reducing data transmission, rather than relying solely on bandwidth scaling, has emerged as a fundamental solution to the bottleneck that is data transfer. Many researchers have proposed data-centric approaches that push computations to SSDs (e.g., Computational Storage [5]) or, even further, directly into flash memory chips through Processing-in-NAND (PiN) [4], [6]–[9]. PiN, positioned furthest from the CPU at the bottom of the memory hierarchy, has the greatest potential for reducing data movement and thus represents a key step towards pushing SSD performance scaling further, as illustrated in Fig. 1. However, PiN is not a universal solution and comes with inherent trade-offs.

No Magic Bullet: Power and transistors are finite resources in every microchip. Adding more logic circuitry to NAND memory, such as for computation or additional read-out functionality, inevitably reduces capacity and increases cost. Since fully utilizing complex logic without exceeding the chip's limited power budget is often impractical. Typically, only simple logic can be embedded within memory, while more complex logic is implemented outside the flash memory chip.

Off-the-shelf (OTS-) PiN takes a different approach by repurposing existing circuits within the memory chip to perform basic calculations [4], [6]. This reduces data transfers at an affordable $/TB cost while preserving the chip's primary storage functionality. However, OTS-PiN typically only supports

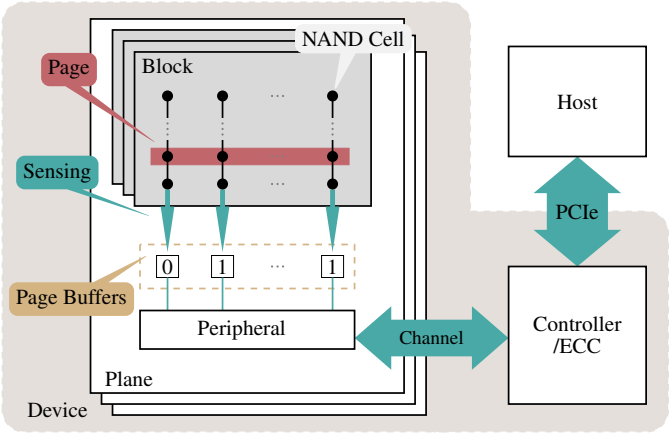

Fig. 2. Simplified structure of a NAND flash memory chip, showing a plane with its independent page buffers. Each page buffer handles one bit in a page.

primitive operations and may sacrifice some storage capacity to improve reliability. Additionally, it generally lacks hardware error correction, requiring applications to incorporate built-in resilience against bit-flip errors.

In this paper, we explore how PiN could help databases achieve performance growth beyond the limits of bandwidth scaling. We begin by examining why newer generations of NAND chips, despite their increased capacity, are approaching the limits of performance scaling (§II). Next, we look at how OTS-PiN offers a promising path forward and address its practical constraints, such as the absence of hardware ECC (§III-A). We also emphasize the importance of error-tolerant data structures in databases (§III-B). Building on this, we explore strategies for databases to effectively leverage OTS-PiN (§IV). We then demonstrate that software-based error tolerance can adapt flexibly to dynamic error rates, offering greater efficiency than hardware-based approaches (§V). Finally, we analyze two widely used probabilistic data structures, Bloom filters and binary sketches, thereby demonstrating their suitability for PiN (§VI).

II. The End of NAND Performance Scaling

Although recent generations of SSDs offer unprecedented *bandwidth*, the access *latency* of NAND-flash memory has actually been deteriorating for some time now [10]. This is not apparent when looking at product descriptions, because manufacturers are playing a whole deck of "tricks" to compensate for the compromises that come with the quest for ever-increasing capacity. However, as an SSD ages with usage, NAND's performance deterioration becomes visible [2].

A. Understanding Performance

To understand the issue and why NAND performance scaling is in jeopardy, we will first describe the factors that impact bandwidth: NAND parallelism and access latency. We then discuss why the current tricks offer no sustainable path forward and ultimately require us to rethink the current approach.

Flash Parallelism: To achieve sufficient bandwidth despite high latency, SSDs employ various forms of parallelism. A single SSD has multiple NAND-flash *chips* (usually 4-8), each

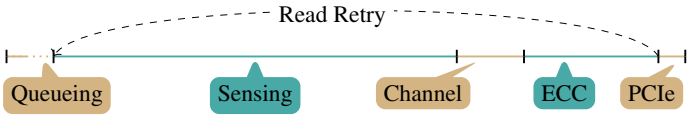

Fig. 3. A breakdown of the steps that make up page read latency: Command send and queuing, sensing, channel transfer, error-correction and external transfer. Read-retry may increase the latency further.

partitioned into so called *planes* (usually 4-6)[2]. Each plane contains a set of page-sized SRAM registers (*page buffers* in Fig. 2) and is capable of sequentially serving requests targeting its (several hundred) flash *blocks*, each block containing (thousands of) pages. For a write, an entire block must be erased first before individual pages can be programmed, a so-called "program-erase" (P/E) cycle. Reads are performed at page granularity of usually 4 or 16 KiB.

Read Latency: When a read request is issued, a command is first sent from the controller to the plane's command queue. To execute the read, the plane first "*senses*" the data into the page buffers by applying a certain voltage. When a cell is programmed, electrons are deposited to represent a charge. During sensing, the cell may or may not have enough charge to conduct the applied voltage, representing a single bit, called "Single-Level Cell" (SLC) NAND. After the page is fully sensed, it is transferred over a bus, which is shared by multiple planes, to the device DRAM ("channel transfer" in Fig. 2) and then checked for bit-flips. If the page passes error correction, it will be send to the host via PCIe, otherwise the read process may be restarted with different parameters to hopefully observe fewer bit-flips. A breakdown of the entire latency can be found in Fig. 3.

Capacity over Latency: Memory technology scaling was the driver of both performance and capacity for a long time. But decreasing the feature size of memory cells comes with significant hurdles, namely high manufacturing cost and degrading performance and reliability [11]: To avoid this issue, manufacturers have instead focused on 3D stacking [11][3] and the number of layers is increasing [12].

Another way to scale up capacity is to store more than one bit per cell. Because both initial programming charge and sensing voltage can be varied between multiple levels, probing for more than one state becomes possible: Triple-Level Cells (TLC), for example, utilize the voltage stored in a cell to represent for three bits, resulting in dividing the voltage range into 2^3 (8) intervals, each associated with a different state. Although this, too, increases capacity, access latency becomes longer. Specifically, TLC requires up to seven read operations to decode all data bits from a single cell, resulting in 2.3X latency compared to SLC for the same amount of data (1 read for 1 bit versus 7 reads for 3 bits).

Latency Mitigation: To mitigate the long latency, some NVMe SSDs can initially write data in "SLC mode" by storing only a one bit per cell. Once this initial capacity threshold (about 20% to 30% of the advertised capacity) is exceeded, the

[2]We omit some levels of the hierarchy (package & die) for simplicity.

[3]As of 2021, the feature size actually *increased* from 14-15 nm to 38 nm [11].

SSD switches to "TLC mode". This can degrade random read throughput by more than 30% [13]. Subsequently, latencies deteriorate with increased capacity usage.

B. Errors in NAND Flash

Another important aspect of latency are bit-flips, or more accurately, error correction. There are various situations for errors to occur in NAND flash: Bits may "flip" during both write or read processes, but the most common errors today are *retention errors* caused by electrons leaking from a cell over time[4] [14], [15].

NAND is known to degrade on write, causing error rates to differ temporally and spatially: Repeated program-and-erase (P/E) cycles on the same block "age" the underlying material, causing it to leak electrons faster and be generally more susceptible to disturbances [14]. Further, due to lithographic imperfections, the material quality across 3D-NAND layers can vary, a phenomenon known as *process variation* [3]. Error rates between pages can also differ, even though they are stored in the same set of cells (a "word-line") [14]. For additional information on NAND flash errors see for example [3], [14].

Error guarantees for today's memory have to be very strict in order to ensure viability for a wide range of applications. For enterprise SSDs, the JEDEC standard defines 10^{-15} as the highest, acceptable *uncorrectable bit error rate* (UBER), i.e., after ECC [16]. The *raw bit error rate* (RBER) can be much larger, i.e., in the order of $0.001 - 0.1$, after a few days of retention time [3], [15]. As a result, powerful error-correction codes (ECC) are necessary to uphold the illusion of error-less data stored in (actually very error-prone) memory.

Bit-Flips vs. Latency: Low-Density Parity-Check (LDPC), the de-facto standard for ECC in SSDs, is applied iteratively: If a step is not able to correct enough errors, the read-process is *restarted* (see Fig. 3), applying different voltage levels during sensing and a more heavy-weight error correction step. This can have detrimental impact on the latency. For example, an initial 85µs access latency can grow to more than 1ms due to ECC [3]. As memory cells become smaller, fewer charges are used to store data bits, making them more susceptible to errors due to charge leakage. Thus, these so called "Fail-Slow" symptoms in SSDs are expected to become more severe with further technology scaling [2].

Error Mitigation: SSDs employ various strategies to avoid errors. Ultimately, error rate, which directly impacts latency, is dependent on many factors. For example, material quality, endured P/E cycles, retention time (i.e., the time since a page was written) and *temperature*: Heat causes cells to leak electrons faster and disturbs signal transmission in electrical circuits. To illustrate, a page can require up to 6 read-retries when retained for 2 days at 85°C, which is equivalent to 168 days of retention time at 40°C [15]. This makes both passive (cooling) and active heat-management (performance throttling) a necessity in modern SSDs.

Another measure, relevant to later discussions on processing-in-NAND, is *data scrambling* [14]: To avoid data-pattern-specific "program disturbance" errors, a page is deterministically turned into a (pseudo random) stream of equally many zeros and ones before being actually written. Because data is de-scrambled only after it was already retrieved from the NAND array, the ability to perform PiN operations may be hindered. Some methods can alleviate this problem by enhancing the programming step at the cost of latency [6]. Further, the data structures discussed in V are designed to achieve uniformity and independence between bits, allowing to skip this measure.

When errors do rise beyond a certain rate, data is *scrubbed* by writing the block to a different location [2]. Eventually, blocks deemed too unreliable by the controller are decomissioned, reducing (internal) capacity. To use up P/E cycles uniformly is the concern of complex wear-leveling strategies employed by the SSD controller. Notably, devices usually opt for performance degradation over offering less storage capacity to the host [2]. This requires allotting extra "hidden" blocks solely for the purpose of replacing the ones that got decomissioned.

C. Scaling Bandwidth & Dark NAND

The influence of temperature on the processor design has been extensively discussed in the literature. The thermal issue has led to the failure of Dennard scaling [17]. This has been recognized as the *dark silicon* problem [17], in which silicons are not able to be utilized due to high temperature. Temperature also plays a critical role in maintaining data integrity, but it also heavily influences the hardware design of NAND flash chips. Just like microprocessors before, NAND flash meets the end of Dennard scaling. With growing transistor density, the power draw also increases, converting into heat. Generally, ensuring stable functionality and longevity becomes increasingly more difficult once we approach the limits of the physical laws. As mentioned above, NAND technology scaling was halted, focusing on capacity over performance.

Scaling Parallelism: Given stagnant access latencies, bandwidth scaling within a single NAND flash chip may be gained by scaling up the plane count. Although planes in modern SSDs are only capable of very rudimentary logic functionality, they are independent logic units and can independently serve requests and therefore determine the level of parallelism within a chip. But manufacturers today still primarily focus on improving the number of pages per plane, rather than plane count, as a way to reduce the cost per gigabyte. This decision to invest chip area in memory cells rather than in peripheral logic is influenced by economical rationales.

Even though the latency-component of channel transfers is comparatively small (see Fig. 3), achieving it requires a significant investment of *instantaneous* power for every single transfer. Previous designs [18], that focus on parallelism over capacity, draw significant amounts of power, require strong cooling to compensate and are very expensive.

Dark NAND: Eventually, increasing the power draw beyond a certain degree leads to "*Dark NAND*", where not all parts of the NAND chip can be used at any given point in time in order to avoid surpassing the thermal budget. Explicit cooling and

[4]Depending on how multi-bit cells encode the different voltage levels, retention errors can result in both types of bit-flips ($0 \rightarrow 1$ and $1 \rightarrow 0$).

thermal throttling are required to maintain stable operations. Like its multicore scaling in the CPU counterpart [17], multi-plane scaling is reaching its limit.

Note that our discussion so far focused on the relative costs of the steps of *reading* data, instead of program & erase, i.e., write, operations. Although writes do have higher latency and cost more power, they are usually executed as background operations in times of less device load. Further, as we will argue below, the *high, relative power requirements* of bus transfers during reads are what motivates processing-in-memory as a potential avenue for future performance scaling.

Scaling Flash-Chips: Increasing the number of NAND-flash chips per SSD as a means to improve bandwidth comes with considerable disadvantages. Adding more chips is cost-inefficient, i.e., doubling bandwidth also (at least) doubles cost. Further, contention of shared, device internal resources, most notably the power supply, DRAM and controller, increases. Any new chip also represents an additional point of failure: When a chip degrades over time ("ages") beyond a critical level, the device not only offers less capacity but also less overall bandwidth. A single chip failing usually warrants decommissioning the whole device, which becomes more costly and increases material waste.

III. Near-Data Processing is Not Enough

One of the largest factors of power consumption (although not latency) during a NAND-flash read are channel transfers, i.e., getting the data out of the plane's page buffers. Thus, methods that reduce data transfer volumes *before they leave the page buffers* directly address the high power demand of channel transfers and thus, the thermal limitations of NAND flash design [4].

In this section, we first suggest PiN as a way to achieve better power efficiency and argue for "Off-the-Shelf" PiN. We describe two prominent representatives and their limits as a basis for further discussions. In the second subsection, we bring up the issue of randomly occurring bit-flips and the cost of ECC, which has to be faced by both PiN and PiM.

A. Processing-in-NAND

PiN, a subset of Processing-in-Memory (PiM), promises better *power efficiency* by empowering individual planes with compute capabilities or, even more extreme, using the memory array itself for computations. However, PiN's functionality is inherently *limited* because complex logic circuits compete with memory for chip area and generate heat, which can compromise memory reliability. This constrains technical and economical feasibility. As a result, PiN designs typically face two options: either *specialize* for a specific application or support only a limited set of *primitive* operations.

Specialized PiN: In specialized circuits, functionality is tailored to a narrow range of applications. Examples are conditional page reads based on string-matching [7] or vector multiplication [9]. This specialization makes these designs less flexible and requires substantial extensions to existing hardware designs. They are therefore less appealing to manufacturers who generally prioritize more versatile, general-purpose solutions with larger markets.

Off-The-Shelf PiN: Instead of creating a purpose-build design from scratch, re-purposing functions in existing circuit designs offers a different approach. The majority of space in a flash chip is dedicated to memory cells. But the plane's peripheral circuit, i.e., the area close to the actual memory, contains a rich amount of logic responsible for accessing or verifying the memory. Off-The-Shelf (OTS) PiN makes use of these circuits to conduct computations, thus limiting modifications to the chip and lowering the adoption barrier.

In the following, we discuss Search-in-Memory (SiM) [4] and FlashCosmos [6], two prominent examples of OTS PiN that form a basis for further discussions.

a) ***Search-in-Memory****:* Search-in-Memory (SiM) re-purposes existing circuits in conventional NAND flash chip to support two new functions: the *search* command for fine-grained matching and the *gather* command. *search* treats a data page as an array of 8-byte words and matches an 8-byte input key against each word stored in the target page, using a (masked) bit-wise `AND`. It evaluates to True, iff all (relevant) bits match. A *search* on a 4 KiB page therefore results in a 512-bit bitmap. The *gather* command pulls a subset of 64-byte chunks, indicated by a bitmap operand, from a page. The two commands always start with a *page open* command, which senses the target page into the page buffers.

Unlike typical page reads, data from the page is not sent to the SSD controller. Instead, a *search* command sends a query word (the "key") to the page buffer for comparison and only transmits the result back. Multiple SiM commands can be run on the page while it is cached in the registers. Once all relevant commands have been completed, a *page close* command will release the data from the register. When the SSD controller executes SiM commands, it can reduce the clock rate of the I/O bus to reduce the power consumption for the channel transfer. This does not affect the latency of the operation because the amount of transferred data is considerably reduced. SiM's efficiency relies heavily on making use of word-aligned arrays.

b) ***FlashCosmos****:* FlashCosmos [6] leverages the sensing mechanism of NAND flash memory to perform bit-wise operations: `NOT` on a single page, `AND` across pages within the same block, and `OR` across pages in different blocks. By combining these basic operations, FlashCosmos can implement more complex logic functions, such as `NOR`, while reducing data transfer overhead by sending only a single "result page" instead of multiple "operand pages."

FlashCosmos requires careful placement of operand pages to align with the logic operations being performed. For example, if a bit-wise `OR` operation involves pages within the same block, page migration is necessary to move one operand to a different block. Moreover, unlike SiM, which sends queries directly to the page buffer for matching, FlashCosmos requires operands to be pre-programmed into memory. This makes FlashCosmos less suitable for workloads with long computation sequences that produce multiple intermediate results, as writing intermediates to NAND memory is slow. Despite these

limitations, SiM and FlashCosmos adhere to existing memory sensing mechanisms and can complement each other.

Design Limitations: Beyond bit-wise logic, many basic operations supported by general purpose processors are based on passing information "between bits". Examples are `popcnt` or integer addition, that requires a carry bit. Functions like (word-wise) addition (or inequality) require complex circuits and therefore represent substantial changes over current (OTS-) NAND designs. In this sense, a plane's page buffers should be thought of as a set of independent 1-bit registers ("latches") instead of one "page-sized register", similar to that of a CPU.

To still be able to implement *search*, which emits only one bit per word, SiM makes use of Failed Bit Counting (FBC) [19], which allows SSDs to verify the integrity of data programmed into flash memory. A program operation requires multiple rounds, with each round generating a pass/fail signal from each page buffer to indicate whether the cells have reached the target accuracy level. These signals are then aggregated to determine if the programmed data meets reliability requirements. This same circuitry can be adapted to perform a (word-wise) `popcnt` operation, though the result must be quantized to avoid high circuit overhead from precise counting.

B. Processing in Unreliable Memory

Generally, bit-flips pose a significant challenge for processing-in-*memory*, because logic and memory are tightly integrated, using techniques like 3D wafer bonding [20] and Peripheral-under-Circuit (PuC). Bypassing existing ECC mechanisms this way requires new approaches to error management, not just for NAND-based PiM.

Hardware Error Correction: Many data-center-grade DRAM memory arrays are tightly integrated with ECC on the same chip to present a seemingly error-free interface. However, as memory cell becomes smaller, data reliability continues to degrade, making on-die ECC increasingly costly and, at times, ineffective [21]. For example, DRAM refresh overhead has increased 9.4 times when scaling from 128Mbit to 16Gbit chips [21]. Similarly, DDR5 ECC-DIMMs with on-die ECC for single-bit error correction now incur a 32.8% increase in area and parity overhead [22]. Despite these efforts to create an error-free memory interface, errors are still inevitable as memory reliability can deteriorate beyond ECC's correction capabilities. In some cases, ECC can even introduce additional errors [21]. High-Bandwidth Memory (HBM), which builds on DRAM technology, inherits these reliability challenges while introducing additional error sources due to its multi-layered architecture [23]. As memory scaling continues to degrade reliability, applications that lack built-in resilience against silent bit errors and rely solely on hardware-based error correction face significant scalability challenges. Further, they are more vulnerable to side-channel attacks like Rowhammer [21], which can induce bit-flips even with ECC in place.

Hardware ECC conflicts with PiN: Hardware ECC requires significant power and chip area, directly competing with resources needed for PiM functionality. As a result, UPMEM [24], a PiM implementation for DRAM, omits hardware ECC entirely and relies on the assumption that the underlying DRAM is error-free.

For NAND flash, the challenge is even greater due to its inherently higher error susceptibility compared to DRAM. Powerful ECC circuits like LDPC are essential to ensure reliability but come with significant costs, including storage overhead for parity bits, increased read latency, and substantial demands on chip area and power. Additionally, these ECC circuits require significant testing and design costs. Consequently, integrating ECC into PiN may be impractical [7], [25].

Over-Reliance on Hardware ECC: Many software systems recognize that storage is not flawless but still assume that bit-flips are rare [26]. For example, file systems and databases like ZFS and RocksDB detect errors and simply abort operations, expecting that a re-read will resolve the issue. However, as memory reliability worsens with advanced technology, errors are likely to become more frequent, making operation aborts a common occurrence and causing severe service disruptions.

Error-Aware Software: Ultimately, solely relying on hardware to maintain an error-free memory interface is both costly and increasingly impractical. One way to address these challenges is to expose information about error characteristics to software, allowing it to make informed decisions and lessen the burden (and reliance) on hardware-based ECC [21]. To take it one step further, memory hardware may be bought with different error guarantees [21]. We will continue the discussion on error-aware software in §V.

Error-Tolerant Databases: The relevance of errors in database applications has been a topic for some time now [27]. For example, Koldiz et al. [28] suggest to use arithmetic coding to enable robust computations and on-the-fly error detection for in-memory databases. In [29], they suggest methods to improve the reliability of B^+-trees. While previous works mostly focus on DRAM and are not necessarily compatible with the PiN architectures considered in this paper, we believe that many of their approaches still offer important pointers to enable databases on increasingly less-reliable hardware.

IV. Processing-in-NAND for Database Processing

In this section, we discuss challenges of integrating PiN into databases. We first claim that simply equipping NAND with compute is not sufficient and requires a more holistic solution. Then, we suggest secondary indexing as a prime candidate for PiN, outlining several approaches. Thereafter, we discuss that efficient integration also requires a suitable interface. The last subsection mentions various other remaining challenges.

A. No Singular Solution

PiN alone cannot fully address NAND's scaling bottleneck due to its inherently limited functionality. Computational Storage (CS), on the other hand, offers more flexibility by implementing application-specific logic in compute units *near* (i.e., outside) the memory chips [5], [30] (Fig. 1). While PiN reduces internal data movement, CS complements it by handling tasks PiN cannot, such as complex arithmetic or aggregating PiN results. This creates a symbiotic relationship that balances flexibility and data transfer efficiency.

This architecture enables a streaming data-flow approach, where data from multiple sources is incrementally refined. By the time it reaches the CPU, the data has been highly filtered and condensed. This approach is similar to the concept of a data-flow architecture [31] discussed in the context of cloud environments.

Heterogeneous PiN: Generally, specialization provides higher efficiency. In this sense, instead of just one type, a database system may want to make use of various different PiN-capable memories, each catering to a different function. For example, one type of memory may be used for bitmap indices [6], another for Bloom filters (see §VI-A) and yet another for binary sketches (see §VI-B).

B. PiN for Secondary Indexing

Databases in an analytical context can require large indices and systems often have to support high dimensional queries, therefore benefiting heavily from columnar storage layouts. Still, for queries with filter predicates, workloads can still suffer from significant read amplification. Indices can help by skipping irrelevant chunks, but they struggle with the large access granularity of NAND-flash memory. File formats, such as Apache Parquet and ORC, a (column) chunk of data may even become significantly larger than a physical page. Being usually light-weight, the indices, e.g, zone maps or Bloom filters, employed in this context often sacrifice discriminative power for flexibility, as well as cheap storage and evaluation.

One role we envision PiN to take over is bulk-index evaluation [4], [6]. In such a scenario, PiN-capable memories may contain index information and may be paired with regular, "reliable" storage that contains the corresponding, actual data, i.e., tables. When the host (declaratively) specifies relevant data, the device may answer this query and provide local data that qualifies. To avoid scanning the tables, it may instead perform an index search implemented with PiN operations. The device, which knows the mapping between index structures and tables, can then facilitate fetching the results from the reliable (also device-local) storage, without involving the host. The fetched data can be further refined and transformed using a device-internal accelerator, before it is eventually send to the host.

Benefits: Indices would no longer need to leave their underlying, persistent memory. This avoids initial deserialization phases and frees up DRAM, SRAM caches and processors in both host and device. Further, in case of ad-hoc queries, i.e., where filter attributes change frequently, keeping various different variants of indices may become feasible due to the cheaper cost of NAND over DRAM. In essence, PiN allows to transform NAND capacity improvements into better application performance and more efficient usage of other resources.

Simplicity preferred: Generally, the layout of data structures must adhere to the "SIMD" processing style of PiN, which relies on assessing the data on a page in a very specific way. For instance, with FlashCosmos, only individual bits across 2 (or more) different pages can be combined logically, resulting in a page-sized result. For Search-in-Memory, an 8 byte query argument is compared individually with all 512 many 8 byte words on a 4 KiB page. While only rudimentary, SiM performs a "bit-wise aggregation" to check if all bits in a word did match or not, producing a 512 bit result for the entire page. Notably, many types of indices, such as B^+-trees and zone maps, are based on (integer or float) inequality instructions, which are not easily available in OTS-PiN (see §III-A).

With this processing style, data should be carefully aligned, and page-sized "arrays" of a single data type are preferred. Workloads with complex arithmetic should not be offloaded to PiN and are better handled by upper-layer compute units (see §IV-A). PiN is most effective for tasks with simple primitives. For example, it can be used to probe external hash table buckets containing fixed-length hash fingerprints [4] or to perform an exhaustive exact search across a large array of embeddings, which are commonly used to represent images or objects [32].

Bitmap Indices: With the continued trend of NAND towards improved capacity, workloads may want to consider spending more space on indices in order to accelerate their queries. Especially more heavy-weight approaches, such as (binned) *bitmap indices*, can occupy considerable storage space and are often too large to be kept in DRAM, especially for high-dimensional datasets. As a result, their high access cost can make them unattractive in current von Neumann-style systems. A PiN-based index evaluation entirely avoids moving the bulk of index information to the host, reducing the interconnect bottleneck and host's page cache usage [4]. Although bitmap indices have a larger storage footprint than, for instance, the light-weight approaches used in Parquet, they have much greater discriminative power. Further, the ability to combine arbitrary attributes efficiently with simple logic instructions ensures flexibility for ad-hoc queries, where the set of relevant attributes may change from query to query. FlashCosmos enables page-wise logic operations and therefore directly implements the evaluation of bitmap indices [6].

Match-based Indices: PiN can be utilized to implement *partial* index searches through masking. One example is querying *Bloom filter* with a logical `AND` operation (see §VI-A). Another example is querying a relational database table encoded as compact keys. These keys are created by concatenating truncated or compressed representations of multiple attribute values from a single tuple. Such key encoding is often employed to run relational databases on top of NoSQL storage engines, as exmplified by MyRocks[5].

Storing these encoded keys in OTS-PiN-capable storage allows SiM's SEARCH operation to perform exhaustive "fuzzy" searches, with the ability to skip irrelevant attributes using the instruction's mask argument. Furthermore, quantization, where a single value represents a subset or range of the actual attribute domain, allows the use of logical `AND`-based instructions for rudimentary range-query support in key-matching.

Approximate Nearest-Neighbor Search: Another potentially interesting data structure are *binary sketches* [33], [34]. They are used to represent high-dimensional, non-relational objects,

[5]https://github.com/facebook/mysql-5.6/wiki/MyRocks-record-format

such as images or documents, with the goal to (approximately) preserve the relative distances between different objects of a database. Binary sketches therefore support workloads where the task is to find objects that are most similar to a query object. Usually, approximate Nearest-Neighbor strategies pursue an "inverted" approach by partitioning a large database of sketches, which allows to skip parts of the search space. But at their core, they still rely on an exhaustive search step, which is potentially executable with PiN. We discuss binary sketches more in §VI-B.

C. Transparent Interfaces & Device Communication

When integrating PiN, a key consideration is how its functionality can be made available to applications. A major cost factor in modern device communication is the (block-based) NVMe interface. Unlike DRAM, where requests are transparently handled by the memory controller, I/O operations require software to explicitly manage them, either through a kernel module or a kernel-bypass library [35]. However, since both approaches involve the CPU, they can introduce significant overhead. A more efficient alternative would be for the device to manage follow-up requests *internally*. For example, in the index applications discussed in §IV-B, shortening the communication path could yield significant performance gains compared to host-based communication.

CXL & Database Kernels: Approaches like Compute Express Link (CXL) [6] can offer a more efficient interface by moving data transfer obligations to a dedicated controller. CXL enables the exchange of memory between devices, the host, and other platforms in a unified, (optionally) coherent way. Beyond simply exchanging memory, it is possible to back up memory ranges exposed via CXL not with actual physical memory. Instead, accessing such a range would transparently trigger a computation and dynamically generate the necessary data. This concept, dubbed Database Kernels [30], allows applications to offer, for example, the same, physical data in both a columnar or row-based representation. For more dynamic operations, the host can, before access, expose operands of the computation by using a different memory region, which is coherently available to the device.

Such an interface may also be used to expose data in PiN-capable memory to the host and/or an accelerator. For example, one memory range may present a page containing a set of search keys in their raw form for maintenance, i.e., writing the data into storage. Another (potentially smaller) range contains the result of a PiN operation executed on the very same page.

D. Open Challenges

The integration of PiN and CS presents numerous challenges. Here, we outline potential topics for further research.

Data Formats and Exchange Protocols: (PiN-backed) CS represents a primary data source, which raises the question on how results and meta-data will be exchanged. Or more specifically, how are (intermediate) result sizes communicated and which representation should results have? Do we apply compression before transfer? When should tuple materialization happen, do the established wisdoms still apply? Can we make use of existing works on accelerators?

Capability, how much is too much? Given its heterogeneity, a system using PiN would require careful balance to properly utilize all available resources. While filters are an obvious candidate to push down, it may be tempting to augment storage for increasingly more complex tasks, such as aggregations. This is not just a matter of identifying proper operators. E.g., a declarative interface would also require the ability to optimize internally. For instance, at which granularity should data be offered and how are requests scheduled internally? Should a device keep (or even track) meta-data (e.g., attribute distribution or usage patterns) to make informed decisions?

Answers to these questions (and others) will eventually come down to power efficiency and affordability, i.e., $/TB. Databases will have to find ways to disaggregate database workloads to appropriately utilize increasingly more heterogeneous hardware architectures.

V. Error-Adaptive Data Structures

In §III-B, we discuss the limitations of relying on hardware ECC and emphasize the importance of making applications resilient to bit-flips. In this section, we argue that resilience alone is not enough; data structures must also be error-adaptive. Designing data structures to be both error-resilient and adaptive not only enables efficient PiN processing, but also provides benefits when they are queried on the CPU.

Varying Impact: Depending on the application, a single bit-flip may have varying impact. For example, a flip in the less significant bits of a floating-point representation usually has much less severe consequences than a flip in the exponent. Similarly, flips in pointers are more detrimental than in a table integer and sometimes a crash is preferable over silent data corruption. Further, the impact of a single bit-flip is particularly severe in compact or highly compressed data structures [28], where every bit carries a high amount of information. In contrast, approximate or probabilistic data structures are more suitable for coping with bit-flips, because they can be designed and configured to account for "additional sources of inaccuracy".

Adaptive vs. Worst-Case: As discussed in §II-B, error rates in different memory regions can vary significantly. Given the error differences across layers of 3D-memory (see §II, [3]), the common approach is to apply the strongest ECC across the entire memory, targeting the least reliable regions. However, this "worst-case approach" is costly and unscalable, as reliability discrepancies between layers are likely to increase with continued scaling. On the other hand, since current and historical error rates are often already available to devices for wear-leveling and data scrubbing [36] and can also be predicted [3], adaptive data structures can unlock many potential optimization opportunities.

Benefits of Error-Adaptivity: With knowledge of error rates across memory regions, software can make informed decisions about data placement and data structure configuration [21]. For

[6] https://computeexpresslink.org/

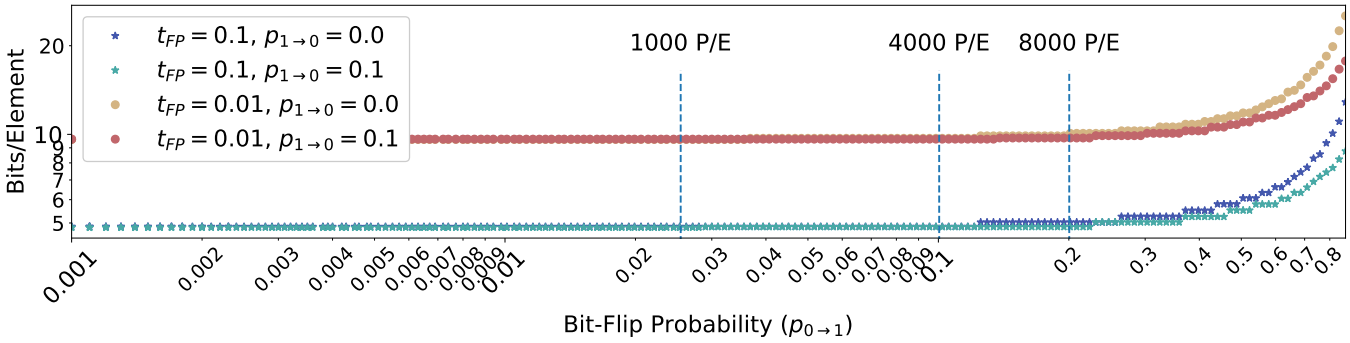

Fig. 4. FPR-restricted (t_{FP}) bits-per-element for a Bloom filter, given bit-flip probabilities $p_{0\to1}$ and $p_{1\to0}$. For reference, the vertical bars indicate the RBER of an older flash chip from Micron for different P/E cycles, after retaining data for 64 days (reproduced from [3]).

example, Tian et al. [3] propose partitioning NAND blocks by error levels, enabling SSD controllers to place "hot" pages in regions requiring less error correction, thereby improving latency for frequently accessed data. Similarly, applications can exploit this knowledge to tailor the protection levels of memory regions to specific reliability requirements, reducing the overhead of overprotection and improving efficiency [37]. Aligning an application's error tolerance levels with the device's varying error rates unlocks further optimization potential:

- *Tailored Maintenance:* Error-tolerance levels specified by applications can flow back to the controller to optimize maintenance and data scrubbing, potentially increasing the longevity of NAND memory. For example, scrubbing a page may be delayed if its current error level is still acceptable for a specific application, or the controller may decide to skip scrubbing altogether, allowing the application to recreate the data later. Similarly, applications may continue using blocks that would otherwise have been decommissioned based on general worst-case criteria.
- *Domain-Specific Error-Correction:* Applications may employ their own domain-specific error-correction mechanisms, which can be more efficient than general-purpose approaches [29]. This approach gives applications more control over performance and storage overhead trade-offs, allowing these decisions to be made before *writing* data, rather than being fixed at design time by the manufacturer.
- *Embracing Bit-Flips:* Many probabilistic data structures, as we will discuss in §VI-A and §VI-B, are inherently error-resilient and can be queried directly without requiring error correction (and its associated overhead). In fact, the NVMe 1.4 standard introduces "Read Recovery Levels," an optimization that aligns with this approach. This feature allows software to control the recovery effort during reads to avoid long decoding latencies for aged data pages and lets software RAID take over the recovery. This feature can also be used to bypass recovery for data that is known to be resilient.

VI. PiN-native Data Structures

Bloom filters and binary sketches are probabilistic data structures that can be queried without requiring complex instructions and, as we will demonstrate, are inherently tolerant to bit-flips. Unlike many database indices, where bit-flips can render the structure completely unusable, these data structures degrade gracefully: query efficiency decreases proportionally to the bit error rate rather than failing entirely. Furthermore, both data structures contain a roughly uniform distribution of zero and one bits, which coincides with the goal of data scrambling, a measure employed by NAND-flash SSDs to avoid data-dependent disturbance errors (see §II-B).

In this section, we first analyze the impact of bit-flips on the efficiency and correctness of these data structures. We then demonstrate how to optimize their parameters to adapt to the changing error rates of the underlying memory, enabling the realization of the optimization opportunities discussed in §V.

We use the following notation: Let $p_{\uparrow\downarrow}$ denote the overall error probability, $p_{0\to1}$ the probability of a flip from 0 to 1, and $p_{1\to0}$ the probability of a flip from 1 to 0. We assume that any bit within a page may flip independently of its physical position and the state of other bits.

A. Bloom Filters

A Bloom filter is used to probabilistically answer set-membership queries. It consists of a fixed-length bit-vector and a set of independent, uniform hash-functions that each map to positions in this vector. Elements are inserted by setting bits, determined by applying the hash functions, to 1. To test for membership, we first have to hash the query element and then test if the corresponding subset of bits were set before. A result may be "false-positive", indicating membership despite the element not actually being inserted before. See [38] for an overview on Bloom filters. In terms of PiN operations, a Bloom filter can be evaluated with a bit-wise-AND operation, available by OTS-PiN approaches like SiM [4].

Due to the relatively small number of relevant bits, the probability of encountering any bit-flip is relatively low. If the query object is not part of the represented set, flips may even be beneficial by turning a false-positive (where all bits are set to 1 by chance) into a true-negative. On the flip-side, if the query object actually is part of the set, flips may introduce false-negative decisions, i.e., where one or more of all considered bits flip from 1 to a 0, which causes the filter to "miss" items.

Impact on Storage Cost: Usually, Bloom filters are designed to not surpass a specified false-positive rate (FPR). Given a fixed number of bits for the filter, this leads to a maximum number of elements that can be inserted ("capacity"), before this rate would be surpassed. For target FPR t_{FP}, size of the filter m and a number of hash-functions k, the maximum capacity of a Bloom filter is $n = -\frac{(m-1)}{k} \ln\left(1 - {t_{\text{FP}}}^{1/k}\right) - 0.5$ [39][7]. To account for bit-flip probabilities $p_{1\to0}$ and $p_{0\to1}$, the number of hash-functions has to be set to (at least):

$$k_{\text{opt}} = \left\lceil \frac{\ln(t_{\text{FP}})}{\ln(0.5) + \ln(1 - p_{1\to0} + p_{0\to1})} \right\rceil \tag{1}$$

We refer to the appendix §A for a proof.

Plugging k_{opt} into the capacity formula above yields the "FPR-limited" capacity[8]. Observe that $0 \to 1$ flips increase the FPR and $1 \to 0$ flips decrease it. As a result, both types cancel each other out: If $p_{1\to0} = p_{0\to1}$, Eq. (1) coincides with the usual formula without errors and bit-flips have no impact at all on the FPR of the filter. Or in other words, for every decision that becomes false-positive due to bit-flips, a false-positive also becomes true-negative. Without bit-flips, the optimal capacity is usually associated with a uniform distribution of ones and zeros in the filter, so a bias in either direction impacts capacity (as determined by k_{opt}) positively or negatively.

Quantifying Bit-Flip Impact: The upper limit to k_{opt} and the capacity formula gives us the tools to assess the robustness of Bloom filters. We illustrate the impact of bit-flips on the bits per element under false-positive restrictions in Fig. 4. For growing $0 \to 1$ errors, the storage cost per element increases eventually, or equivalently, the capacity decreases. But using a larger k is not necessary until $p_{0\to1}$ is well above 0.01, a value significantly higher than JEDEC's 10^{-15}. For example, for $t_{\text{FP}} = 0.1$ and $p_{1\to0} = 0$, the very first jump (from $k = 3$ to $k = 4$) occurs at $p_{0\to1} \approx 0.125$, raising storage cost from 4.86 bits to 5.03 bits per element. Before, $0 \to 1$ errors do not have any impact on the choice of k and therefore the (FPR-limited) capacity, offering the same performance as a Bloom filter in regular "error-less" memory. Further, $1 \to 0$ flips delay the increase in bits per element. As indicated by the vertical bars, these error ranges can be reached even by older memory chips.

False-Negatives Guarantees: While $1 \to 0$ flips are beneficial for the capacity of the filter, a newly-introduced, non-zero false-negative rate impacts the quality of the actual result, because actual members of the set can now be "overlooked". The false-negative rate is given by $\text{FNR} = 1 - (1 - p_{1\to0})^k$, which is the complement of the probability that none of the 1 bits flip. For applications that can tolerate a certain false-negative rate of t_{FN}, we can rearrange for k and identify an *upper limit* to the number of hash-functions:

$$t_{\text{FN}} = 1 - (1 - p_{1\to0})^k$$
$$\Rightarrow k_{\text{max}} = \left\lfloor \frac{\ln(1 - t_{\text{FN}})}{\ln(1 - p_{1\to0})} \right\rfloor$$

If $k_{\text{max}} < k_{\text{opt}}$, the filter can not be build optimally. This is because the optimal capacity can only be achieved with an even chance of encountering a 0 or 1. To compensate, we may add fewer elements in order to require fewer hash-functions for the same FPR. We therefore speak of an "FNR-limited" capacity.

Impact of False-Negatives: Although the introduction of false-negatives is not acceptable for some applications, others can tolerate it. In fact, across a wide range of applications in distributed systems surveyed in [40], various Bloom filter variants allow for false negatives. While introducing false negatives may seem undesirable, it is often a necessary trade-off in applications that require item deletion [41] or an improved false-positive ratio [42].

Exploiting Error-Asymmetry: When $p_{1\to0} = p_{0\to1}$, the capacity is not FPR-limited, but may be FNR-limited. But with a "one-sided" error we are able to exploit that $1 \to 0$ flips *increase both capacity and FNR*. This allows, for example, to strengthen correctness over capacity by inverting the bit-interpretation of the bitmap of a Bloom-filter, if $p_{1\to0} \geq p_{0\to1}$.

SLC NAND allows to ensure a high asymmetry, i.e., to make the error one-sided. This is possible because the largest source of bit-flips today is electron leakage [14]. If there are only two states to differentiate, cells can only drop once, from the higher to the lower. In contrast, any substantial increase of the cell charge is significantly less likely [14]. While being more expensive, SLC is also overall less error prone, allows faster read and write and offers more durability than, for example, TLC NAND.

PiN Implementation: For applications, we envision a page storing a whole set of Bloom filters, or parts thereof. Assuming the example configuration for Chen et al.'s SiM [4], a 4KiB page may consist of 512 many 64 bit words. A page may then contain, for instance, the first 64 bits of a (potentially longer) Bloom filter. For evaluation, a SEARCH instruction is issued with a query element representing the (partial) result of the hash-function evaluation. Matching the query with each filter prefix results in a positive response, if all relevant bits are set in both query and filter. Compared to a regular page access, which requires transmitting the whole 4KiB page, only the 512 bit result must be transmitted over the channel bus. Results from multiple pages are combined with a logical AND, potentially using an accelerator within the storage device. Each bit in the final result then indicates if the input set corresponding to the bit's position did contain the query, or not. Such a setup may find applications in large-scale analytics file formats (e.g., Apache Parquet) or in Key-Value stores, such as RocksDB, where separate indices are used for many independent sets of elements.

Error-Adaptive Parametrization: As outlined above, the optimal capacity depends on the bit-flip rate and essentially dictates the number of hash functions and subsequently the

[7]This slightly modified version sacrifices "1 element and 0.5 bits" for a more accurate estimate of the FPR (or, after rearranging, capacity) [39].

[8]Note that, due to hash collisions, the actual number of relevant bits may be smaller than k, especially for very small k and m.

filter capacity. To illustrate, even the pages within the same 3D NAND block and written at approximately the same time can have vastly different error levels that differ by up to one order of magnitude [43]. Crucially, error rates are usually not uniformly distributed and pages with extreme rates potentially being outliers [43]. A page with an error above a certain threshold will eventually drop in capacity (see Fig. 4). An application may therefore under-utilize Bloom filters stored across this block, if it calculates the maximum capacity based on worst-case error rates. While this is less of an issue for highly asymmetric error environments, overestimating $1 \to 0$ errors can lead to severe restrictions.

To illustrate, assuming $p_{0\to1} > p_{1\to0}$ and $t_{\text{FP}} = t_{\text{FN}} = 0.01$, classifying all pages in a block to have, say, a worst-case ratio of $p_{1\to0} = 0.0034$, restricts k to at most $\left\lfloor \frac{\ln(1-0.01)}{\ln(1-0.0034)} \right\rfloor = 2$. With $k = 2$, the bits per element ratio in the worst case would be $\frac{m-1}{n+0.5} = 18.98$. If the actual probability is $p_{1\to0} = 0.002$ or lower instead, k may be as large as 5, costing only 9.85 bits per element, a difference of factor 2. We therefore caution against using worst-case assumptions to characterize NAND flash errors.

Overall, we observe that *Bloom filters are inherently tolerant to one-sided bit-flip errors*. Large errors can be compensated for by sacrificing some some bits per element at extreme error rates, i.e., adding fewer elements than usually possible. Lastly, we point out that the dynamic nature of errors and deviation of theoretical assumptions in real-world hardware require further investigation.

B. Binary Sketches

Binary sketches are compact, fixed length bit vectors used for similarity search and allow for efficient comparison of two instances of a more complex, usually high-dimensional object, such as images or documents. These sketches can be evaluated by performing a bit-wise `xor` operation between a query bit vector and the bit vector of an instance from the database, followed by a `popcnt` to get their Hamming distance. This is a basic building block for nearest neighbor search, which relies on first identifying all elements that have a Hamming distance below a certain threshold.

There are various methods for forming binary sketches, for instance HIOB [33]. See [34] for a survey. Besides relying on the Hamming distance, these methods usually share the objective of achieving independence between bits, as well as "balanced" bits, i.e., a uniform distribution of 0 and 1 bits. The inherently approximate character of the search, as well as a potential of the two types of bit-flips to negate each other's impact on the distance calculation, makes binary sketch evaluation interesting for PiN.

Impact of Bit-Flips: We will now consider how bit-flip probabilities affect the effectiveness and result quality of finding the set $\{s \mid d(s,q) \leq r\}$, or, in other words, the set of all neighbors s that have Hamming-distance d of at most r to query-sketch q. When any of the bits in a sketch flips, the resulting distance to a query is either decreased or increased by 1, depending on if the flipped bit now matches the corresponding bit or not. Thus, for the distance to suddenly become smaller than a given threshold r, decreases in distance must be more numerous than increases. This implies that sketches with high, actual distance are less likely to become neighbors due to bit-flips. Still, with increasing (overall) bit-flip probability $p_{\uparrow\downarrow}$ (the type of flip is irrelevant), more and more neighbors will be "swapped-out" for (mostly not-too-distant) non-neighbors (false-positives), while some actual neighbors may not be found (false-negatives).

To illustrate this effect, Fig. 5 shows how $p_{\uparrow\downarrow}$ affects the distribution of *actual* distances in the result set, if we filter according to the *observed* distances. We've shaded the sufficiently-small-distances, i.e., true-positive neighbors, as well as the 95%-percentile of the distances, to indicate the actual distance most candidates will have in the result set. With $p_{\uparrow\downarrow} = 0$, all distances are within the required threshold, because observed and actual distance coincide. At a low rate of bit-flips, candidates with actual distance larger than the threshold may be selected, albeit with low chance and most probably being only slightly more far away than r. Ultimately, the distribution approaches the initial binomial distribution with increasing error, indicating that the process eventually degrades to a random sampling of candidates. We derive the shown distribution in §B.

Error-Adaptive Parametrization: A single distance alteration has less relative impact in a larger sketch. However, considering more bits also increases the probability of encountering an error. To compensate for larger error, false-positives can be filtered out in a later step, when the actual objects are accessed and refined further. This allows to choose a less strict threshold at the cost of additional post processing and larger result size.

PiN Implementation: Following a similar processing schema as for Bloom filters, a page may contain a sequence of fixed-length sketches. But in contrast to Bloom filters, binary sketches are more challenging to realize via PiN. One issue is the need of a *popcnt* aggregation, which is harder to implement than an equality test (see III-A). Further, transmitting the result of one *popcnt* per word is also less power efficient, because each result requires multiple bits. Instead, it would be preferable to also incorporate the thresholding operation into the circuit, which results in only one bit. Another issue is the word size, which ideally coincides with the sketch size. Smaller words would require adding up multiple *popcnt*s before thresholding can occur. Splitting the operation as well is possible but introduces another source of errors. Moreover, these modifications specialize the circuit further towards a single application, potentially making it less universally applicable.

Overall, binary sketches can also *tolerate significant error rates before overlooking too many neighbors*. A more detailed evaluation of their suitability for implementation in PiN is left for future research.

VII. Conclusion

Multi-core parallelism cannot offer a free lunch due to inherently sequential portions in an algorithm (Amdahl's law). Nevertheless, it has provided a way forward when processor

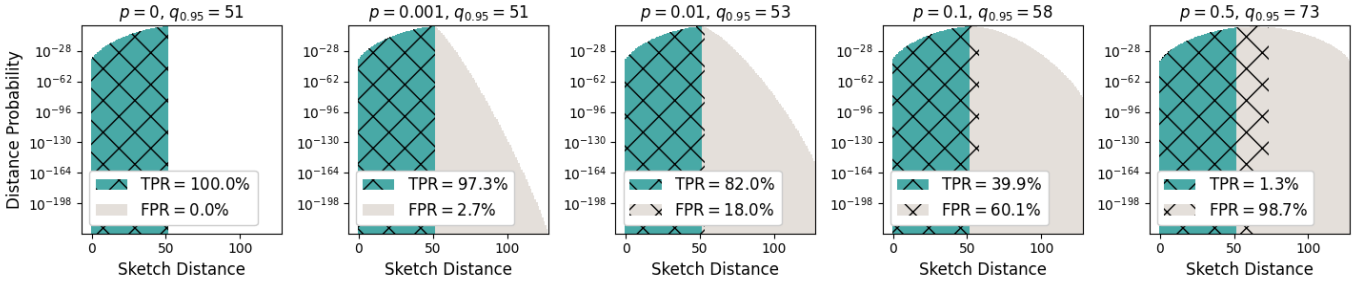

Fig. 5. Distribution $Pr(H_0 \mid H \leq r)$ for sketch size $S = 128$, selectivity 0.001 (i.e., threshold $r = 51$) and different bit-flip probabilities $p_{\uparrow\downarrow}$.

frequency scaling stopped. Similarly, there is a need to find an alternative to NAND flash bandwidth doubling, which, we argue, should be Processing-in-NAND. However, its successful integration requires identifying database functions that can both be efficiently realized in hardware and are worthwhile to "push down". Further, the illusion of error-free memory comes at substantial costs, such as pessimistic hardware ECC, and may soon no longer be affordable. Databases will eventually need to promote error tolerance and the ever-increasing hardware heterogeneity as first-class concerns in their design. Beyond methods to detect and correct errors, embracing bit-flips with resilient data structures may offer a way forward. We find that bulk index evaluation using Bloom filters and binary sketches presents a promising starting point for enabling PiN and achieving better efficiency with error-conscious and error-adaptive application design.

Appendix A
Error-Dependent Bloom Filter Capacity

Let $p_{1\to1} := 1 - p_{1\to0}$ be the probability that a 1 is retained and $p_{[0]} := 1 - p_{[1]}$ the probability of an initial 0. Then, the probability of *observing* a 1 after bits may have flipped is ${p_{[1]}}^{\text{final}} = p_{[0]}p_{0\to1} + p_{[1]}p_{1\to1}$. The chance of a false-positive is then equal to ${{p_{[1]}}^{\text{final}}}^k$. For fixed bit-flip probabilities and a target FPR t_{FP}, this limits the number of hash-functions k that we are allowed to employ from below. Note that the lowest, inherent FPR (and thus maximum capacity) of a Bloom filter is assumed if $p_{[1]} = p_{[0]} = 0.5$, which we will substitute in the following. Rearranging for k yields:

$$
\begin{aligned}
{{p_{[1]}}^{\text{final}}}^k &\leq t_{\text{FP}} \\
k &\geq \frac{\ln(t_{\text{FP}})}{\ln((1-p_{1\to0})p_{[1]} + p_{0\to1}(1-p_{[1]}))} \\
k &\geq \frac{\ln(t_{\text{FP}})}{\ln(0.5 \cdot ((1-p_{1\to0}) + p_{0\to1}))} \\
\Rightarrow k_{\text{opt}} &:= \left\lceil \frac{\ln(t_{\text{FP}})}{\ln(0.5) + \ln(1 - p_{1\to0} + p_{0\to1})} \right\rceil
\end{aligned}
$$

Appendix B
Hamming Distance Distribution

Let S be the length of the binary sketch and $H_0, H \in [0, 1, \ldots, S]$ denote the random variables for the *actual* (before bit-flips) and the *observed* Hamming distance (after bit-flips), respectively. Under the assumption of independent, uniformly distributed bits of a random sketch from the database, $H_0 \sim \text{binom}(0.5, S)$. Let $N, M \sim \text{binom}(p_{\uparrow\downarrow}, S)$ be independent random variables for the bit-flips that decrease or increase the distance, respectively. We can then express the observed distance H as the sum of these variables, i.e., $H = h_0 - N + M$. The probability of observing a specific distance of h, given an actual distance of h_0, is then:

$$
\begin{aligned}
Pr(H = h \mid H_0 = h_0) = \sum_{n=0}^{h_0} & Pr(N = n) \cdot Pr(M = m) \\
& \cdot I(0 \leq m \leq S - h_0)
\end{aligned}
$$

where $m = n + h - h_0$ and the indicator function I drops invalid sum terms for given n. The probability distribution of H_0 after filtering is then:

$$
\begin{aligned}
& Pr(H_0 = h_0 \mid H \leq r) \\
&= \frac{\sum_{h=0}^{r} Pr(H_0 = h_0, H = h)}{Pr(H \leq r)} \\
&= \frac{\sum_{h=0}^{r} Pr(H = h \mid H_0 = h_0) \cdot Pr(H_0 = h_0)}{\sum_{h=0}^{r} Pr(H = h)}
\end{aligned}
$$

with $Pr(H = h) = \sum_{\hat{h}_0=0}^{S} Pr(H = h \mid H_0 = \hat{h}_0) \cdot Pr(H_0 = \hat{h}_0)$.

References

[1] T. Neumann and M. J. Freitag, "Umbra: A disk-based system with in-memory performance," in *10th Conference on Innovative Data Systems Research, CIDR, Amsterdam, Netherlands, January 12-15, 2020.*

[2] Z. Jiao, X. Zhang, H. Shin, J. Choi, and B. S. Kim, "The design and implementation of a capacity-variant storage system," in *22nd USENIX Conference on File and Storage Technologies, FAST, Santa Clara, CA, USA, February 27-29*, 2024, pp. 159–176.

[3] H. Tian, J. Tang, J. Li, Z. Sha, F. Yang, Z. Cai, and J. Liao, "Modeling retention errors of 3D nand flash for optimizing data placement," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 29, no. 4, Jun. 2024.

[4] Y.-C. Chen, Y.-H. Chang, and T.-W. Kuo, "Search-in-Memory: Reliable, versatile, and efficient data matching in SSD's NAND flash memory chip for data indexing acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 3864–3875, 2024.

[5] A. Lerner and P. Bonnet, "Not your grandpa's SSD: The era of co-designed storage devices," in *International Conference on Management of Data (ICDE)*, 2021, p. 2852–2858.

[6] J. Park, R. Azizi, G. F. Oliveira, M. Sadrosadati, R. Nadig, D. Novo, J. Gómez-Luna, M. Kim, and O. Mutlu, "Flash-Cosmos: In-flash bulk bitwise operations using inherent computation capability of NAND flash memory," in *55th International Symposium on Microarchitecture (MICRO)*, 2022, pp. 937–955.

[7] M. Chun, J. Lee, S. Lee, M. Kim, and J. Kim, "PiF: in-flash acceleration for data-intensive applications," *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*, 2022.

[8] H.-W. Hu, W.-C. Wang, C. K. Chen, Y.-C. Lee *et al.*, "A 512Gb in-memory-computing 3D-NAND flash supporting similar-vector-matching operations on edge-ai devices," *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 65, pp. 138–140, 2022.

[9] H.-W. Hu, W.-C. Wang, Y.-H. Chang, Y.-C. Lee, B.-R. Lin, H.-M. Wang, Y.-P. Lin, Y.-M. Huang, C.-Y. Lee, T.-H. Su, C.-C. Hsieh, C.-M. Hu, Y.-T. Lai, C.-K. Chen, H.-S. Chen, H.-P. Li, T.-W. Kuo, M.-F. Chang, K.-C. Wang, C.-H. Hung, and C.-Y. Lu, "ICE: An intelligent cognition engine with 3D nand-based in-memory computing for vector similarity search acceleration," in *55th International Symposium on Microarchitecture (MICRO)*, 2022, pp. 763–783.

[10] C. Liu, J. B. Kotra, M. Jung, M. T. Kandemir, and C. R. Das, "SOML read: Rethinking the read operation granularity of 3D NAND SSDs," in *24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) , April 13-17, 2019*, pp. 955–969.

[11] S. S. Kim, S. K. Yong, W. Kim, S. Kang, H. W. Park, K. J. Yoon, D. S. Sheen, S. Lee, and C. S. Hwang, "Review of semiconductor flash memory devices for material and process issues," *Advanced Materials*, vol. 35, no. 43, p. 2200659, 2023.

[12] "Nand flash density race," https://www.trendforce.com/news/2024/06/27/news-outpacing-samsung-or-the-end-of-race-kioxia-eyes-1000-layer-nand-by-2027/, 2024, accessed: 2024-07-22.

[13] T. O. Page, "The impact of SLC cache in performance of an NVMe SSD: Benchmarks and results," August 2024, accessed: 2024-11-25. [Online]. Available: https://theoverclockingpage.com/2024/08/23/the-impact-of-slc-cache-in-performance-of-an-nvme-ssd-benchmarks-and-results

[14] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based Solid-State Drives," *Proc. IEEE*, vol. 105, no. 9, pp. 1666–1704, 2017.

[15] M. Ye, Q. Li, Y. Lv, J. Zhang, T. Ren, D. Wen, T. Kuo, and C. J. Xue, "Achieving near-zero read retry for 3D NAND flash memory," in *29th International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS), La Jolla, CA, USA, 27 April - 1 May 2024*, pp. 55–70.

[16] JEDEC Solid State Technology Association, "Failure Mechanisms and Models for Semiconductor Devices," Tech. Rep. JEP122H, Sept 2016.

[17] H. Esmaeilzadeh, E. R. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," *IEEE Micro*, vol. 32, no. 3, pp. 122–134, 2012.

[18] T. Shiozawa, H. Kajihara, T. Endo, and K. Hiwada, "Emerging usage and evaluation of low latency flash," in *International Memory Workshop (IMW)*, 2020, pp. 1–4.

[19] N. Choi and J. Kim, "Modeling and simulation of nand flash memory sensing systems with cell-to-cell vth variations," in *International Conference On Computer Aided Design (ICCAD)*, 2020.

[20] Z. Yue, H. Wang, J. Fang, J. Deng, G. Lu, F. Tu, R. Guo, Y. Li, Y. Qin, Y. Wang, C. Li, H. Han, S. Wei, Y. Hu, and S. Yin, "Exploiting similarity opportunities of emerging vision ai models on hybrid bonding architecture," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 396–409.

[21] M. Patel, T. Shahroodi, A. Manglik, A. G. Yaglikçi, A. Olgun, H. Luo, and O. Mutlu, "Rethinking the producer-consumer relationship in modern dram-based systems," *CoRR*, vol. abs/2401.16279, 2024.

[22] D. Kim, J. Lee, W. Jung, M. B. Sullivan, and J. Kim, "Unity ECC: unified memory protection against bit and chip errors," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Denver, CO, USA, November 12-17*, 2023, pp. 48:1–48:16.

[23] R. Wu, S. Zhou, J. Lu, Z. Shen, Z. Xu, J. Shu, K. Yang, F. Lin, and Y. Zhang, "Removing obstacles before breaking through the memory wall: A close look at HBM errors in the field," in *USENIX Annual Technical Conference (ATC), Santa Clara, CA, USA, July 10-12*, 2024, pp. 851–867.

[24] J. Gómez-Luna, I. E. Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture," 2022.

[25] H. Cilasun, S. Resch, Z. I. Chowdhury, M. Zabihi, Y. Lv, B. Zink, J. Wang, S. S. Sapatnekar, and U. R. Karpuzcu, "On error correction for nonvolatile Processing-In-Memory," in *51st Annual International Symposium on Computer Architecture (ISCA), Buenos Aires, Argentina, June 29 - July 3*, 2024, pp. 678–692.

[26] S. Jaffer, S. Maneas, A. A. Hwang, and B. Schroeder, "The reliability of modern file systems in the face of SSD errors," *ACM Trans. Storage*, vol. 16, no. 1, pp. 2:1–2:28, 2020.

[27] M. Böhm, W. Lehner, and C. Fetzer, "Resiliency-aware data management," *Proc. VLDB Endow.*, vol. 4, no. 12, pp. 1462–1465, 2011.

[28] T. Kolditz, D. Habich, W. Lehner, M. Werner, and S. T. de Bruijn, "AHEAD: Adaptable data hardening for on-the-fly hardware error detection during database query processing," in *International Conference on Management of Data*, New York, NY, USA, 2018, p. 1619–1634.

[29] T. Kolditz, T. Kissinger, B. Schlegel, D. Habich, and W. Lehner, "Online bit flip detection for in-memory b-trees on unreliable hardware," in *10th International Workshop on Data Management on New Hardware (DaMoN)*, New York, NY, USA, 2014.

[30] S. Lee, A. Lerner, P. Bonnet, and P. Cudré-Mauroux, "Database kernels: Seamless integration of database systems and fast storage via CXL," in *14th Conference on Innovative Data Systems Research (CIDR), Chaminade, HI, USA, January 14-17*, 2024.

[31] A. Lerner and G. Alonso, "Data flow architectures for data processing on modern hardware," in *40th International Conference on Data Engineering (ICDE)*, 2024, pp. 5511–5522.

[32] F. Yang, A. Kale, Y. Bubnov, L. Stein, Q. Wang, M. H. Kiapour, and R. Piramuthu, "Visual search at eBay," in *23rd SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pp. 2101–2110.

[33] E. Thordsen and E. Schubert, "An alternating optimization scheme for binary sketches for cosine similarity search," in *Similarity Search and Applications - 16th International Conference, SISAP 2023, A Coruña, Spain, October 9-11*, ser. Lecture Notes in Computer Science, vol. 14289. Springer, 2023, pp. 41–55.

[34] D. Rachkovskij, "Index structures for fast similarity search for binary vectors," *Cybernetics and Systems Analysis*, vol. 53, pp. 799–820, 2017.

[35] G. Haas and V. Leis, "What modern nvme storage can do, and how to exploit it: High-performance I/O for high-performance storage engines," *Proc. VLDB Endow.*, vol. 16, no. 9, pp. 2090–2102, 2023.

[36] Q. Li, M. Ye, Y. Cui, L. Shi, X. Li, T.-W. Kuo, and C. J. Xue, "Shaving retries with sentinels for fast read over high-density 3D flash," in *53rd Annual International Symposium on Microarchitecture (MICRO)*, 2020, pp. 483–495.

[37] Y.-C. Chen, C.-F. Wu, Y.-H. Chang, and T.-W. Kuo, "Zonelife: How to utilize data lifetime semantics to make SSDs smarter," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 8, pp. 2488–2499, 2023.

[38] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, "Optimizing bloom filter: Challenges, solutions, and comparisons," *IEEE Commun. Surv. Tutorials*, vol. 21, no. 2, pp. 1912–1949, 2019.

[39] A. Goel and P. Gupta, "Small subset queries and bloom filters using ternary associative memories, with applications," in *SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, New York, New York, USA, 14-18 June*, 2010, pp. 143–154.

[40] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 1, pp. 131–155, 2012.

[41] D. Guo, Y. Liu, X. Li, and P. Yang, "False negative problem of counting bloom filter," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 5, pp. 651–664, 2010.

[42] B. Donnet, B. Baynat, and T. Friedman, "Retouched bloom filters: allowing networked applications to trade off selected false positives against false negatives," in *Conference on Emerging Network Experiment and Technology, CoNEXT, Lisboa, Portugal, Dec. 4-7*, 2006, p. 13.

[43] Q. Li, L. Shi, Y. Cui, and C. J. Xue, "Exploiting asymmetric errors for LDPC decoding optimization on 3D NAND flash memory," *IEEE Trans. Computers*, vol. 69, no. 4, pp. 475–488, 2020.