P&S Genomics

Lecture 10: GenStore

Nika Mansouri Ghiasi

ETH Zürich
Spring 2023
11 May 2023

GenStore:
A High-Performance In-Storage Processing System for Genome Sequence Analysis

P&S Accelerating Genomics, 8 December 2022

Nika Mansouri Ghiasi, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadhoun Alserr, Rachata Ausavarungnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu

SAFARI

ETH zürich

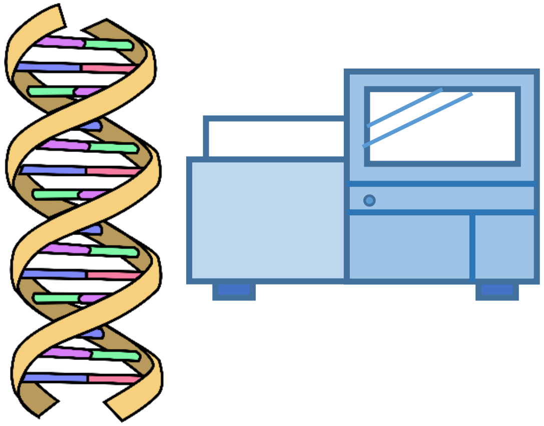
bionano GENOMICS

UNIVERSITY OF TORONTO

Genome Sequence Analysis

- Genome sequence analysis is critical for many applications
  - Personalized medicine
  - Outbreak tracing
  - Evolutionary studies

- Genome sequencing machines extract smaller fragments of the original DNA sequence, known as reads

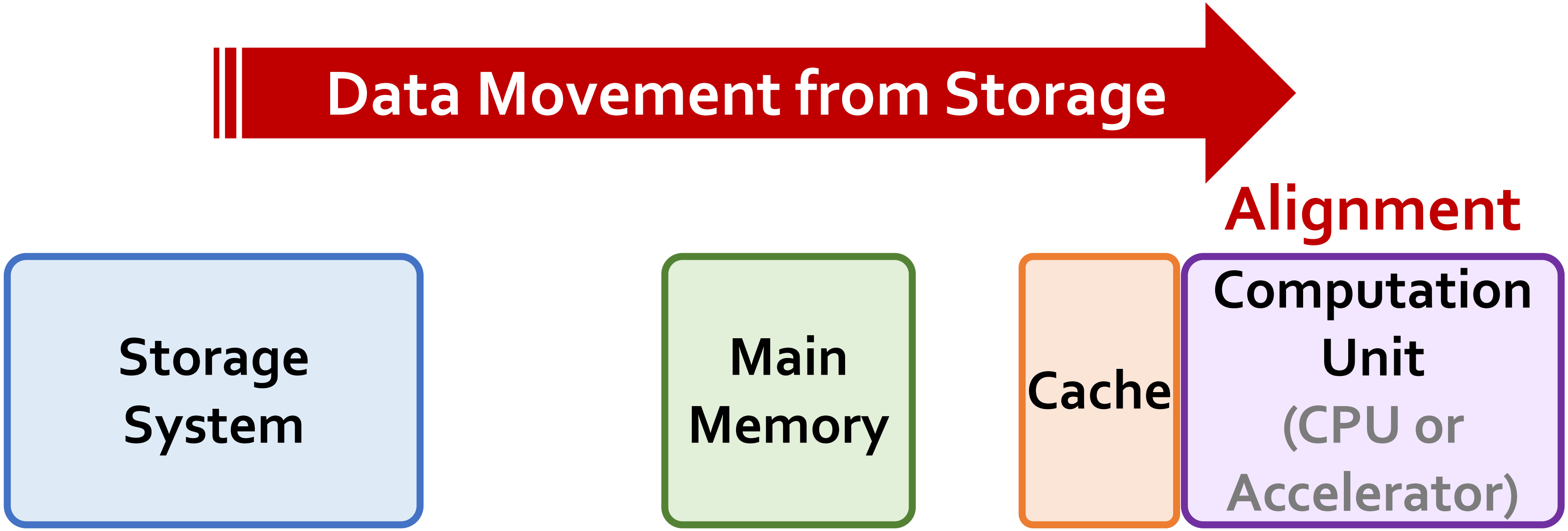


Genome Sequence Analysis

- **Read mapping:** first key step in genome sequence analysis
  - Aligns reads to potential matching locations in the reference genome
  - For each matching location, the alignment step finds the degree of similarity (alignment score)

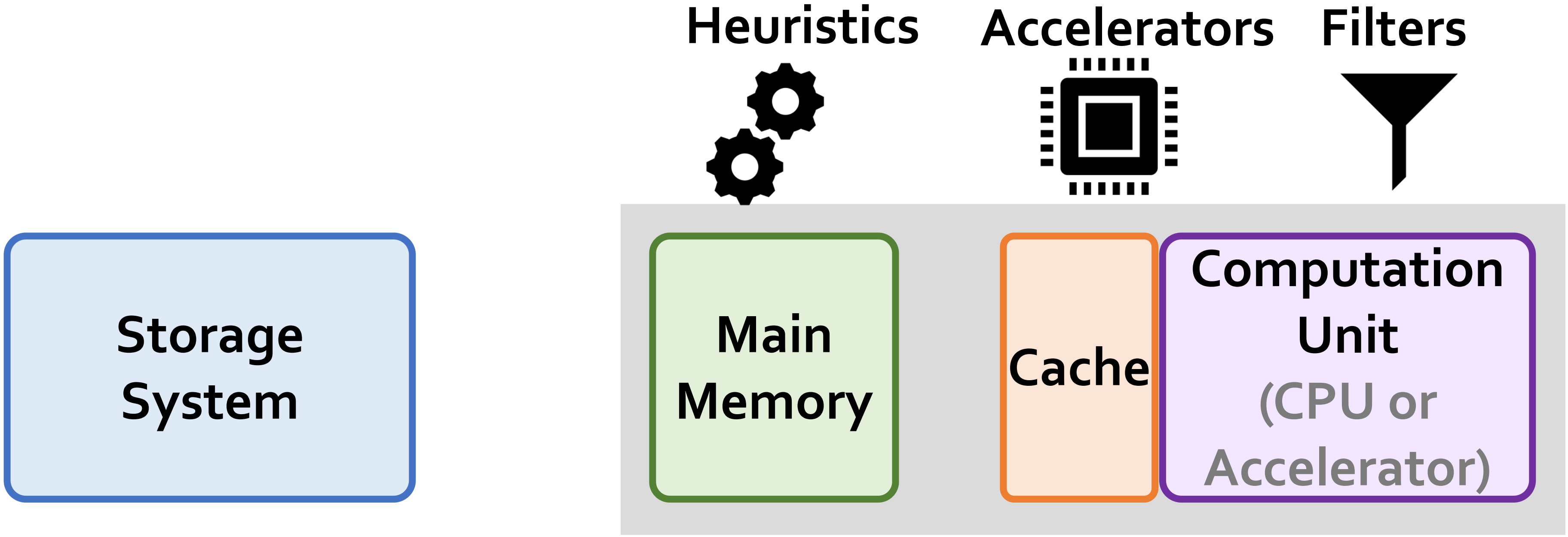
- Calculating the alignment score requires computationally-expensive approximate string matching (ASM) to account for differences between reads and the reference genome due to:
  - Sequencing errors
  - Genetic variation

Genome Sequence Analysis



- ✗ Computation overhead
- ✗ Data movement overhead

Accelerating Genome Sequence Analysis

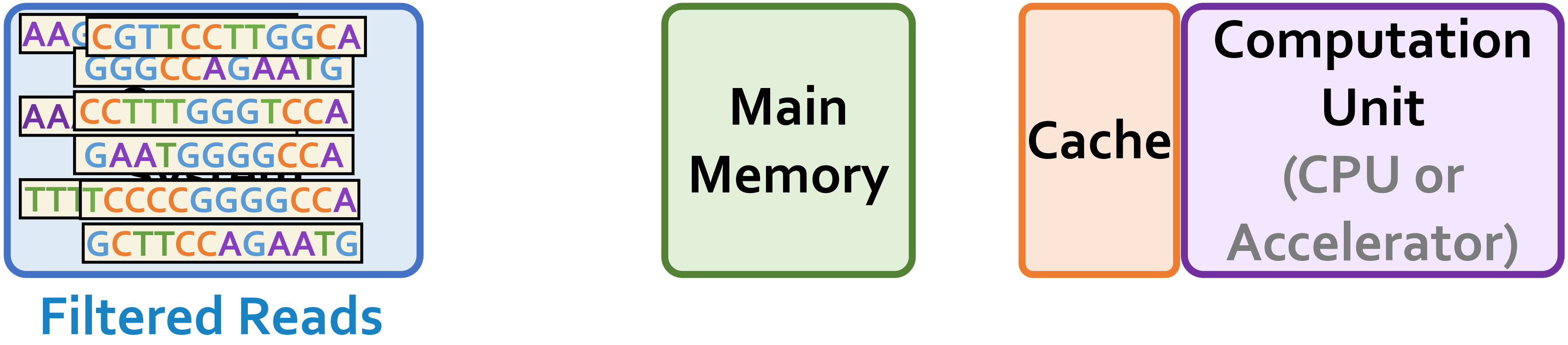


✓ Computation overhead

✗ Data movement overhead

Key Idea

Filter* reads that do *not* require alignment *inside the storage system



Exactly-matching reads

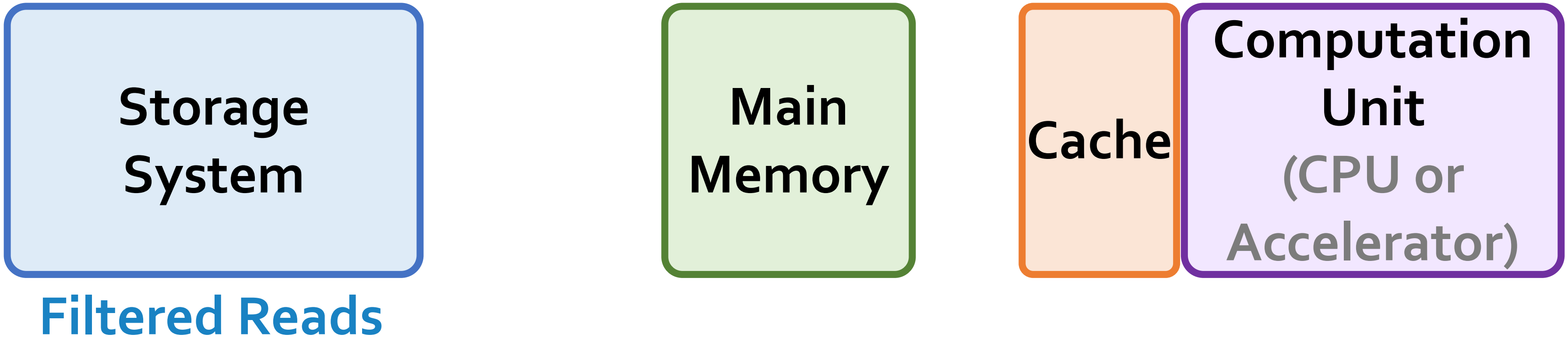
Do not need expensive approximate string matching during alignment

Non-matching reads

Do not have potential matching locations and can skip alignment

Challenges

Filter reads that do *not* require alignment *inside the storage system*

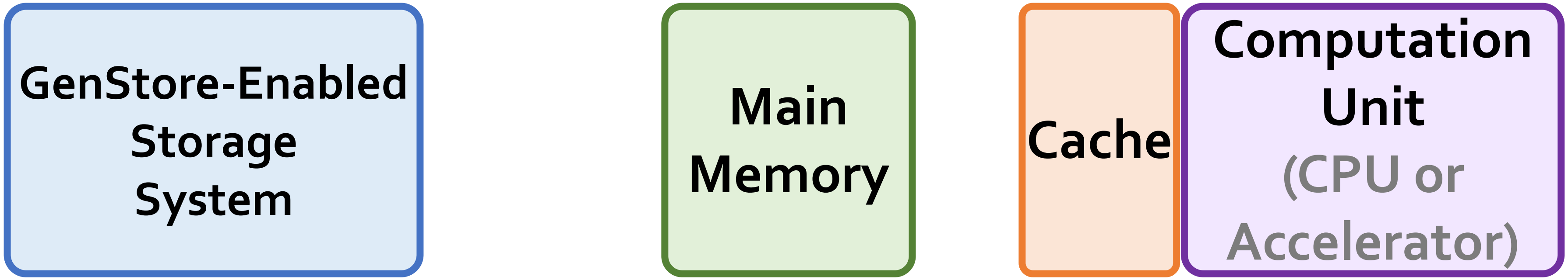


Read mapping workloads can exhibit different behavior

There are limited hardware resources in the storage system

GenStore

Filter reads that do *not* require alignment
inside the storage system



✓ Computation overhead

✓ Data movement overhead

GenStore provides significant speedup (1.4x - 33.6x) and energy reduction (3.9x – 29.2x) at low cost

Outline

Background

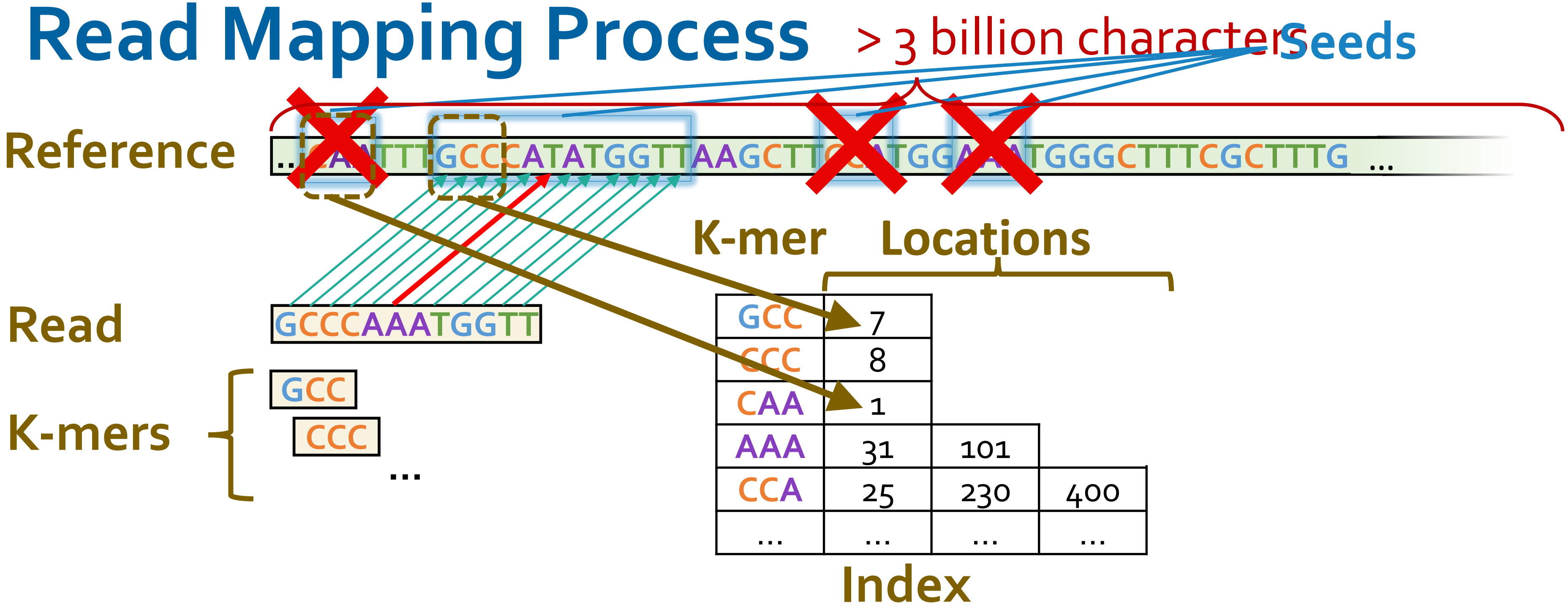
Motivation and Goal

GenStore

Evaluation

Conclusions

Read Mapping Process



K-mer	Locations		
GCC	7		
CCC	8		
CAA	1		
AAA	31	101	
CCA	25	230	400
...	...	...	...

Seeding	Determine potential matching locations (seeds) in the reference genome
Seed Filtering (e.g., Chaining)	Prune some seeds in the reference genome
Alignment	Determine the exact differences between the read and the reference genome

Outline

Background

Motivation and Goal

GenStore

Evaluation

Conclusions

Motivation

- Case study on a real-world genomic read dataset
  - Various read mapping systems
  - Various state-of-the-art SSD configurations

The ideal in-storage filter significantly improves performance by

1) **reducing the computation overhead**
2) **reducing the data movement overhead**

Motivation

- Case study on a real-world genomic read dataset
  - Various read mapping systems
  - Various state-of-the-art SSD configurations

Filtering outside SSD provides lower performance benefit since it

1) does not reduce the data movement overhead

2) must compete with read mapping for system resources

A HW accelerator reduces the computation bottleneck,

which makes I/O a larger bottleneck in the system

Our Goal

Design an in-storage filter for genome sequence analysis in a cost-effective manner

Design Objectives:

Performance

Provide high in-storage filtering performance to overlap the filtering with the read mapping of unfiltered data

Applicability

Support reads with 1) different properties and 2) different degrees of genetic variation in the compared genomes

Low-cost

Do not require significant hardware overhead

Outline

Background

Motivation and Goal

GenStore

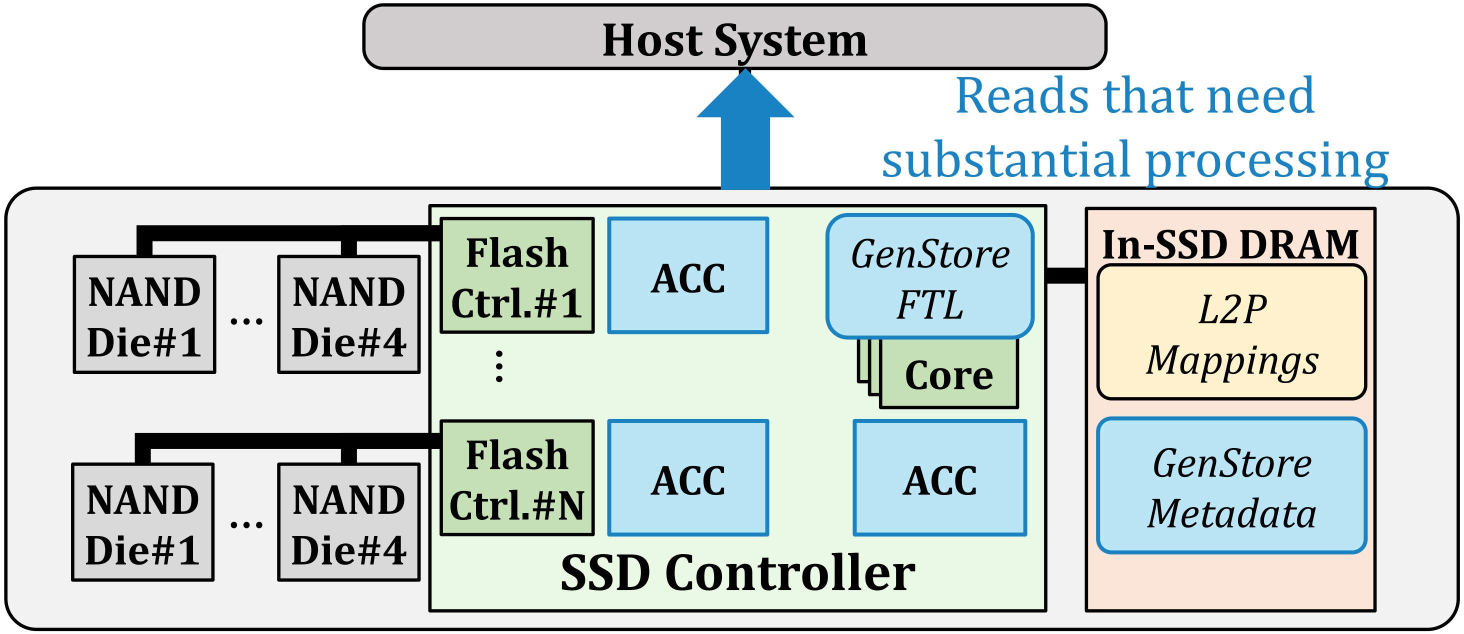
Evaluation

Conclusions

GenStore

- **Key idea:** Filter reads that do not require alignment inside the storage system

- **Challenges**
  - Different behavior across read mapping workloads
  - Limited hardware resources in the SSD



Filtering Opportunities

- Sequencing machines produce one of two kinds of reads
  - **Short reads:** highly accurate and short
  - **Long reads:** less accurate and long

Reads that do not require the expensive alignment step:

Exactly-matching reads

Do not need expensive approximate string matching during alignment

- Low sequencing error rates (short reads) combined with
- Low genetic variation

Non-matching reads

Do not have potential matching locations, so they skip alignment

- High sequencing error rates (long reads) or
- High genetic variation (short or long reads)

GenStore

GenStore-**EM** for **E**xactly-**M**atching Reads

GenStore-**NM** for **N**on-**M**atching Reads

GenStore

GenStore-EM for Exactly-Matching Reads

GenStore-NM for Non-Matching Reads

GenStore-EM

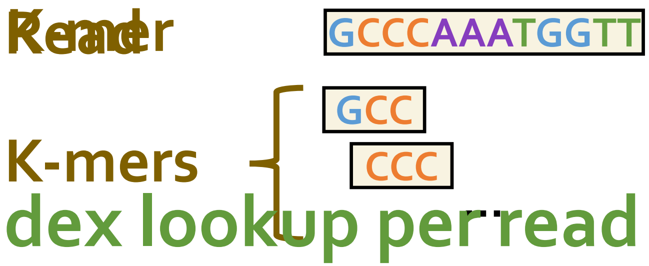
- Efficient in-storage filter for reads with at least one exact match in the reference genome
- Uses simple operations, without requiring alignment
- **Challenge:** large number of random accesses per read to the reference genome and its index

Expensive random accesses to flash chips

Limited DRAM capacity inside the SSD

GenStore-EM: Data Structures

- **Read-sized k-mers:** to reduce the number of accesses per each read



✓ **Only one index lookup per read**

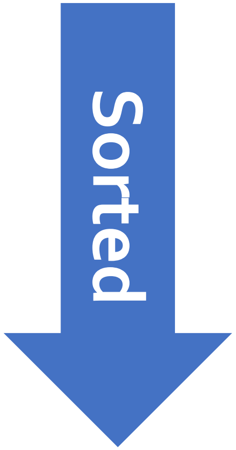
- **Sorted read-sized k-mers:** to avoid random accesses to the index

✓ **Sequential scan of the read set and the index**

GenStore-EM: Data Structures

Sorted Read Table

	Read
	AAAAAAAAAAA
	AAAAAAAAAAG
	AAAAAAAAACT
	…

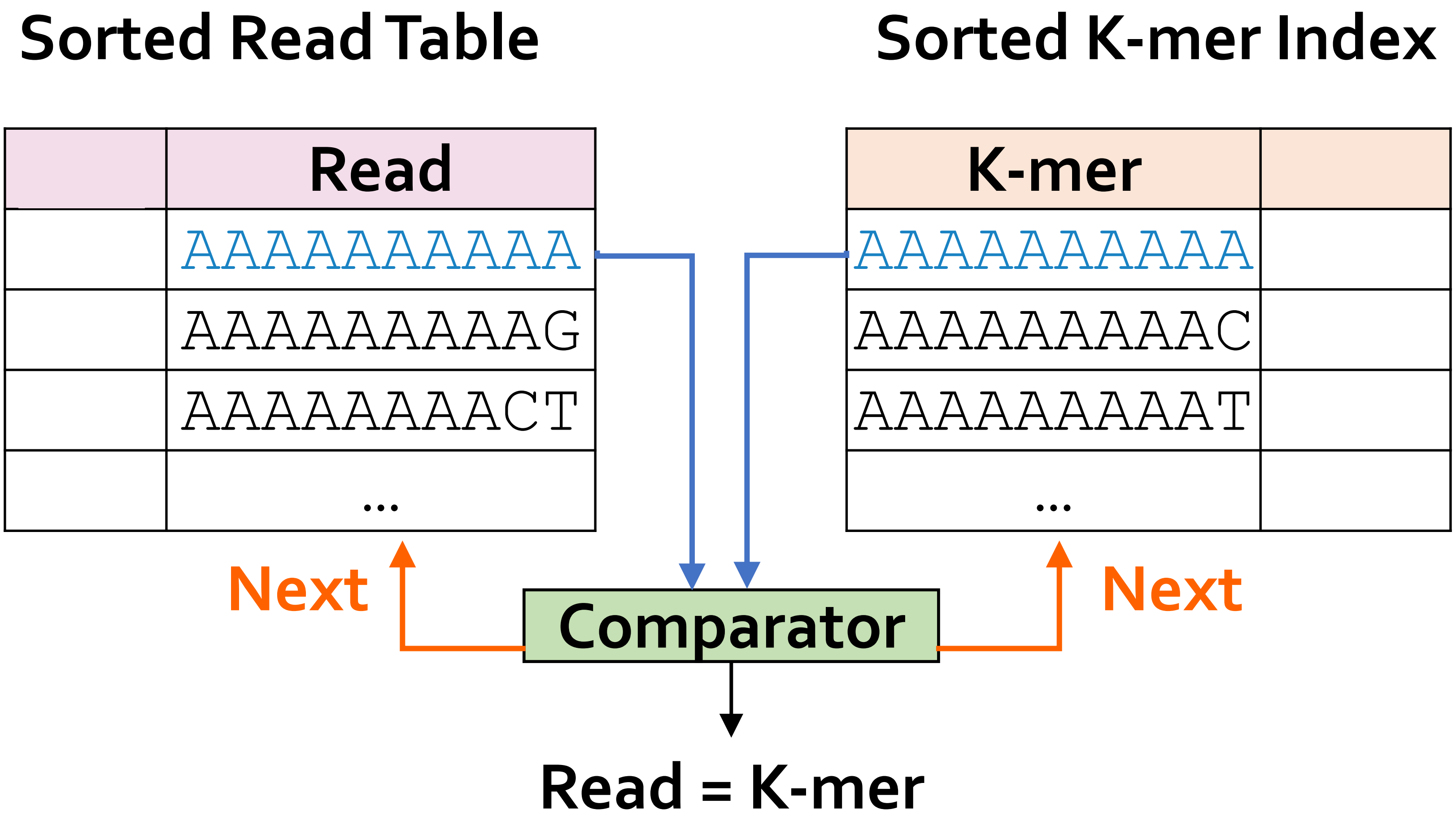


Sorted K-mer Index

K-mer	
AAAAAAAAAAA	
AAAAAAAAAAC	
AAAAAAAAAAT	
…	

Read-sized K-mers

GenStore-EM: Finding a Match



Sorted Read Table

	Read
	AAAAAAAAAA
	AAAAAAAAAG
	AAAAAAAACT
	…

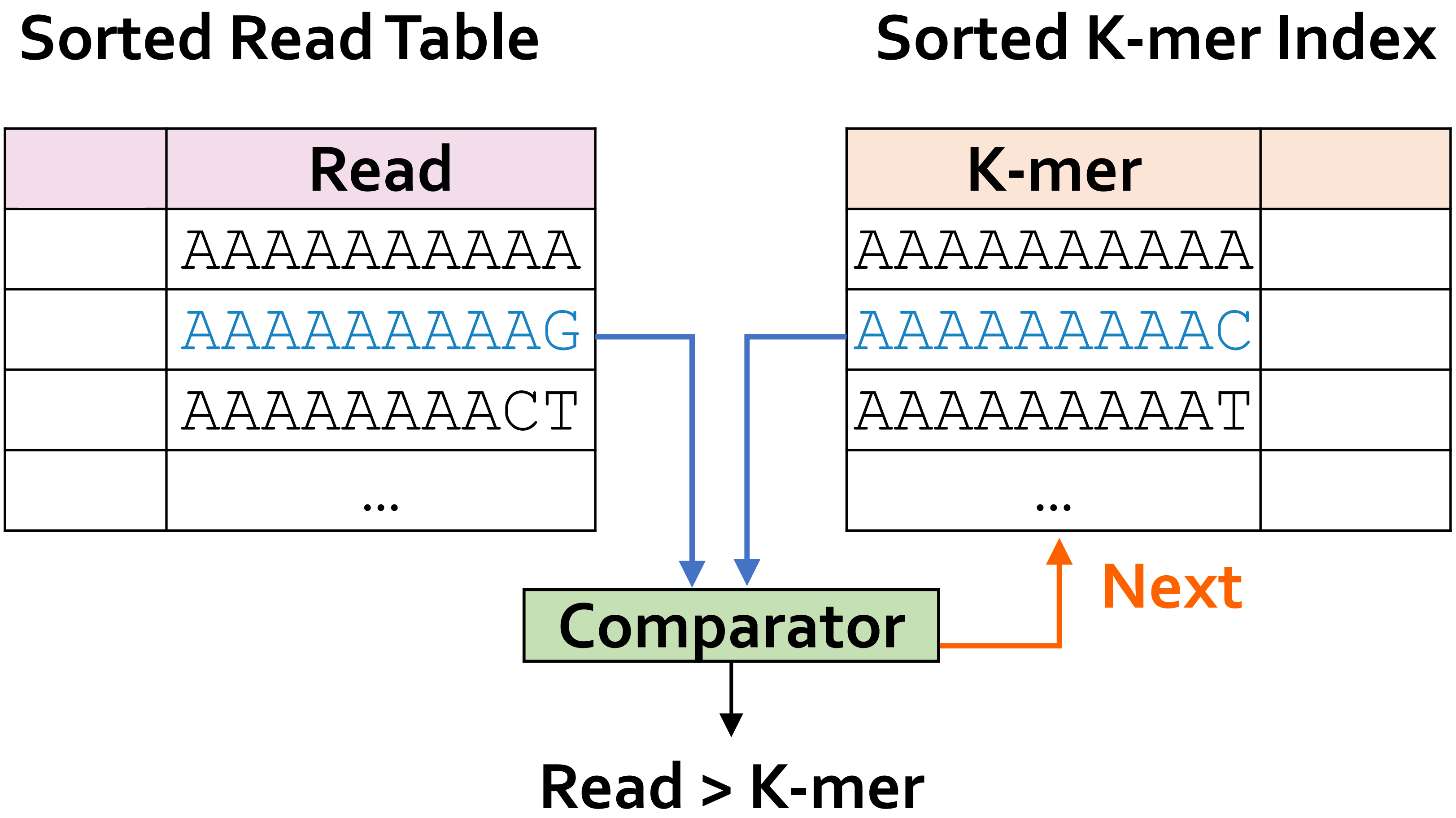
Sorted K-mer Index

K-mer	
AAAAAAAAAAA	
AAAAAAAAAAC	
AAAAAAAAAAT	
…	

Read = K-mer

Exact match → Filter the read

GenStore-EM: Not Finding a Match



Sorted Read Table

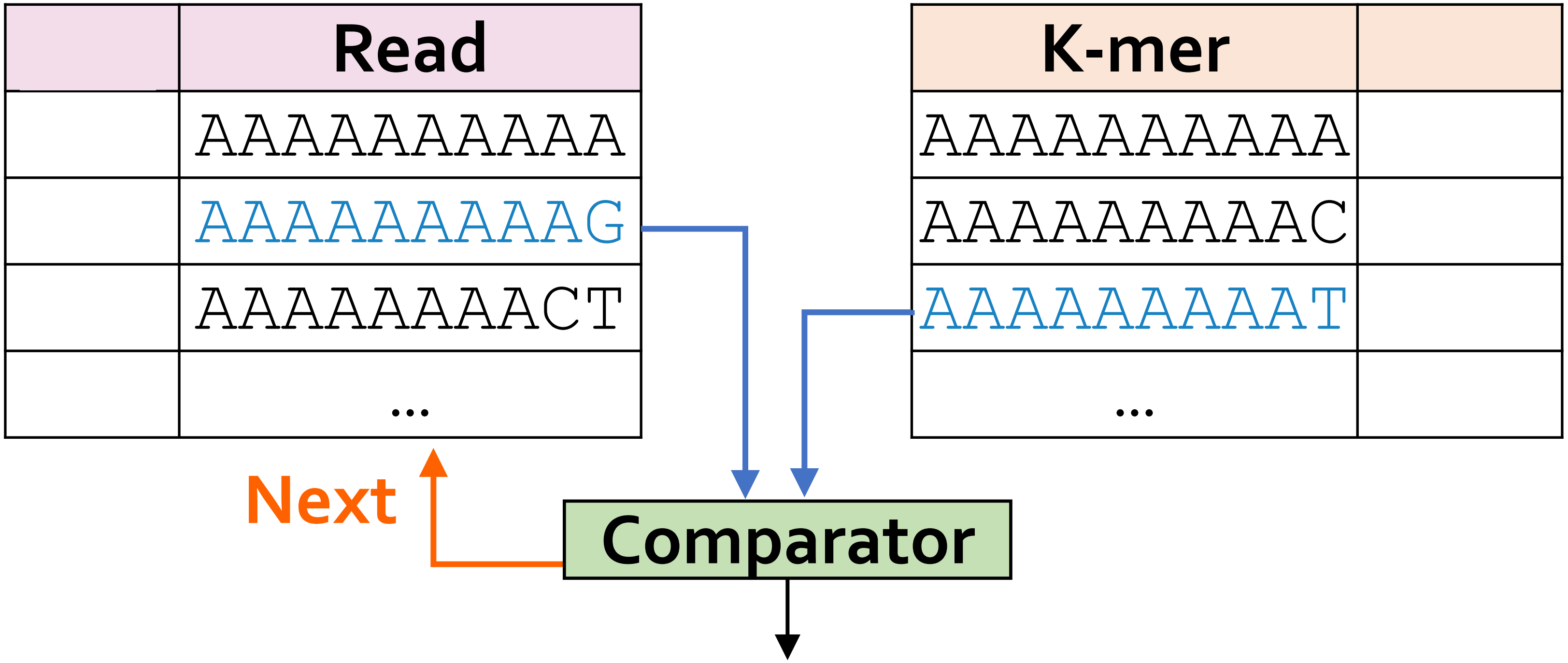
	Read
	AAAAAAAAAA
	AAAAAAAAAG
	AAAAAAAACT
	…

Sorted K-mer Index

K-mer	
AAAAAAAAAA	
AAAAAAAAAC	
AAAAAAAAAT	
…	

Read > K-mer

GenStore-EM: Not Finding a Match



Sorted Read Table

	Read
	AAAAAAAAAA
	AAAAAAAAAG
	AAAAAAAACT
	…

Sorted K-mer Index

K-mer	
AAAAAAAAAA	
AAAAAAAAAC	
AAAAAAAAAT	
…	

Read < K-mer

Not an exact match → Send to read mapper

GenStore-EM: Not Finding a Match

Sorted Read Table

	Read
	AAAAAAAAAA

Sorted K-mer Index

K-mer	
AAAAAAAAAA	

✓ Avoids random accesses

✓ Simple low-cost logic

Comparator

Read < K-mer

Not an exact match → Send to read mapper

GenStore-EM: Optimization

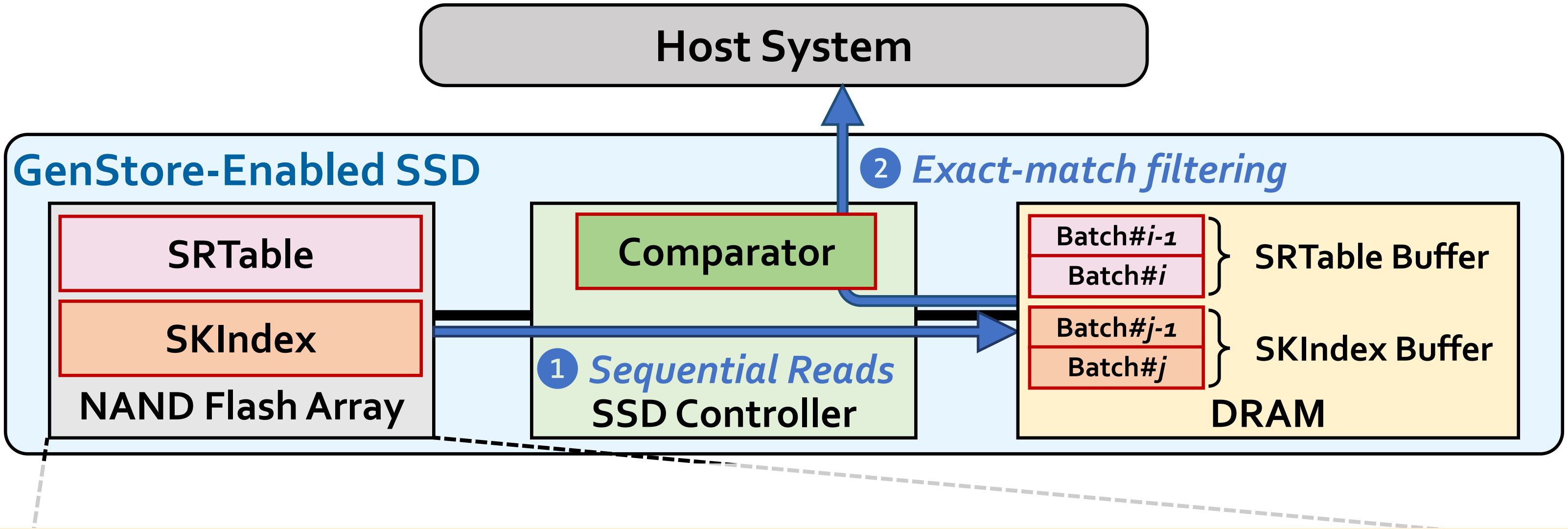
- Read-sized k-mer index takes up a large amount of space (126 GB for human index) due to the larger number of unique k-mers

Sorted K-mer Index

Strong Hash Value	Loc.
1	1, 8, ...
4	51
7	23, 37
16	...

Using strong hash values instead of read-sized k-mers reduces the size of the index by 3.9x

GenStore-EM: Design



Host System

GenStore-Enabled SSD

② *Exact-match filtering*

SRTable

SKIndex

NAND Flash Array

Comparator

① *Sequential Reads*

SSD Controller

Batch#*i*-1
Batch#*i*
SRTable Buffer

Batch#*j*-1
Batch#*j*
SKIndex Buffer

DRAM

Steps 1 and 2 are pipelined.
During filtering, GenStore-EM sends the unfiltered reads to the host system.

Data is evenly distributed between channels, dies, and planes to leverage the full internal bandwidth of the SSD

GenStore

GenStore-EM for Exactly-Matching Reads

GenStore-NM for Non-Matching Reads

GenStore-NM

- Efficient chaining-based in-storage filter to prune most of the non-matching reads

Seeding	Determine potential matching locations (seeds) in the reference genome
Seed Filtering (e.g., Chaining)	Prune some seeds in the reference genome
Alignment	Determine the exact differences between the read and the reference genome

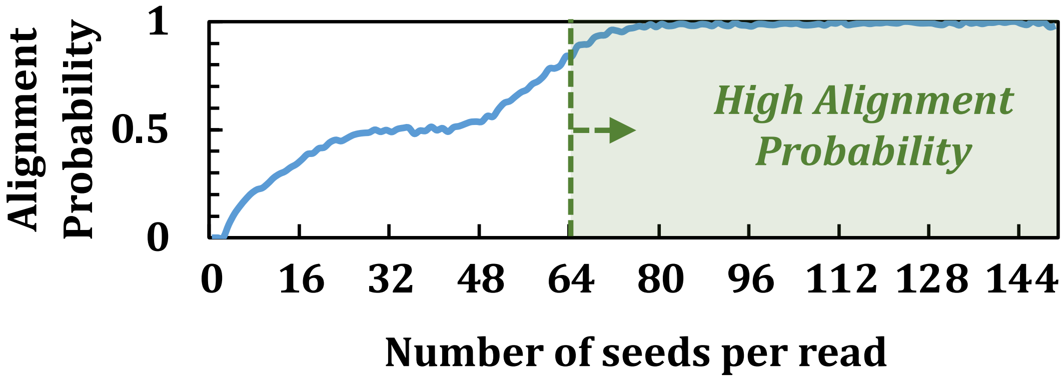
- **Challenge:** how to perform chaining inside the SSD

Costly dynamic programming on many seeds in each read

Particularly challenging for long reads with many seeds

GenStore-NM: Mechanism

- GenStore-NM uses a light-weight chaining filter
  - Selectively performs chaining only on reads with a small number of seeds
  - Directly sends reads that require more complex chaining to the host system

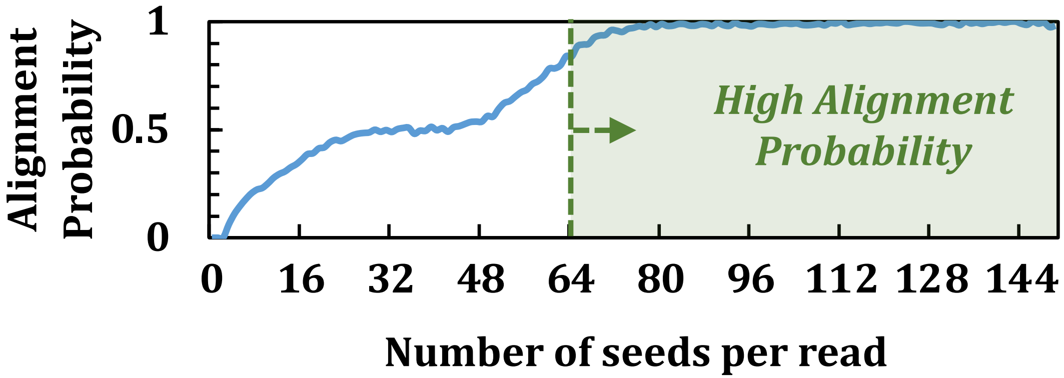


Reads with a sufficiently large number of seeds are very likely to align to the reference genome

✓ **Filters many non-aligning reads without costly hardware resources in the SSD**

GenStore-NM: Mechanism

- GenStore-NM uses a light-weight chaining filter
  - Selectively performs chaining only on reads with a small number of seeds
  - Directly sends reads that require more complex chaining to the host system



Reads with a sufficiently large number of seeds are very likely to align to the reference genome

Details on GenStore-NM's design are in the paper

Outline

- Background
- Motivation and Goal
- GenStore
- Evaluation
- Conclusions

Evaluation Methodology

Read Mappers

- **Base:** state-of-the-art software or hardware read mappers
  - Minimap2 [Bioinformatics'18]: software mapper for short and long reads
  - GenCache [MICRO'19]: hardware mapper for short reads
  - Darwin [ASPLOS'18]: hardware mapper for long reads
- **GS:** Base integrated with GenStore

SSD Configurations

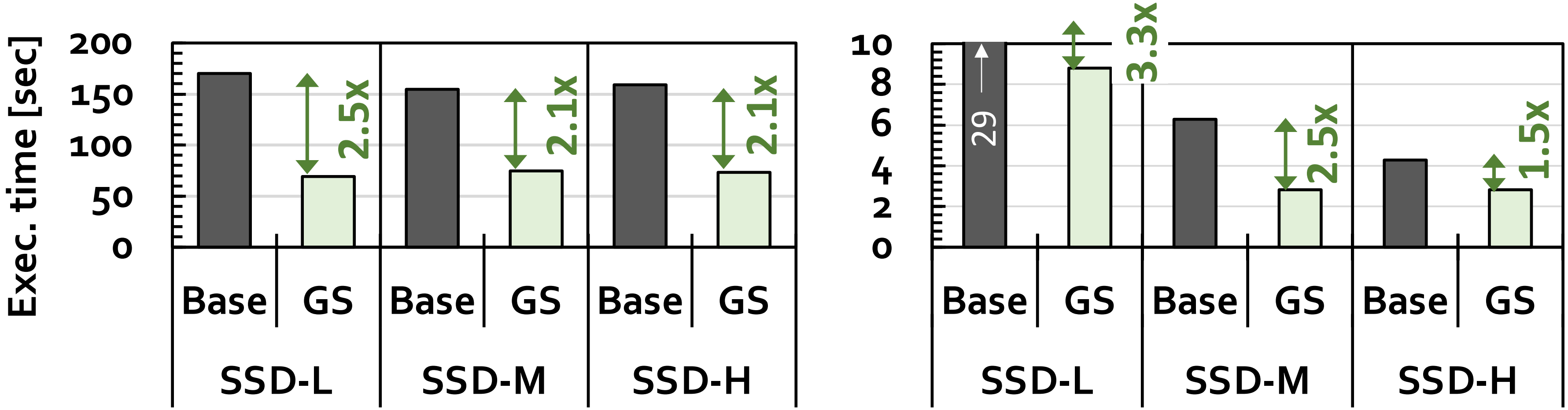
- **SSD-L:** with SATA3 interface (0.5 GB/s sequential read bandwidth)
- **SSD-M:** with PCIe Gen3 interface (3.5 GB/s sequential read bandwidth)
- **SSD-H:** with PCIe Gen4 interface (7 GB/s sequential read bandwidth)

Performance – GenStore-EM

For a read set with 80% exactly-matching reads

With the Software Mapper

With the Hardware Mapper



2.1× - 2.5× speedup compared to the software Base

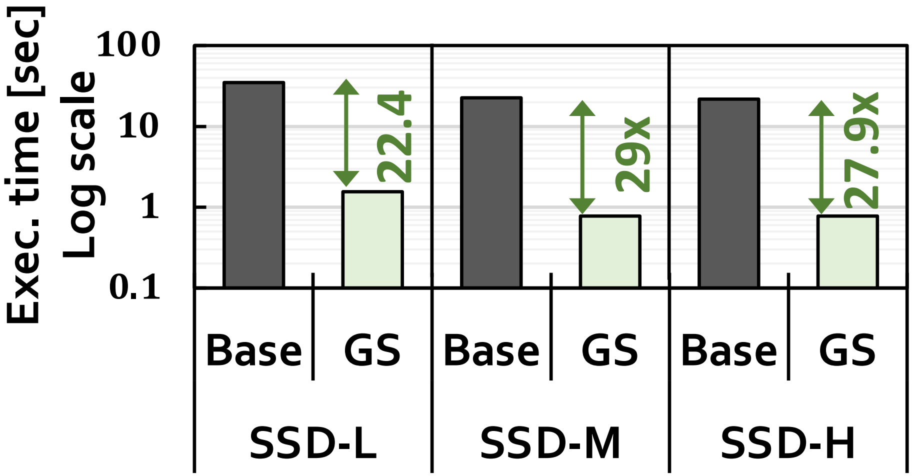
1.5× – 3.3× speedup compared to the hardware Base

On average 3.92× energy reduction

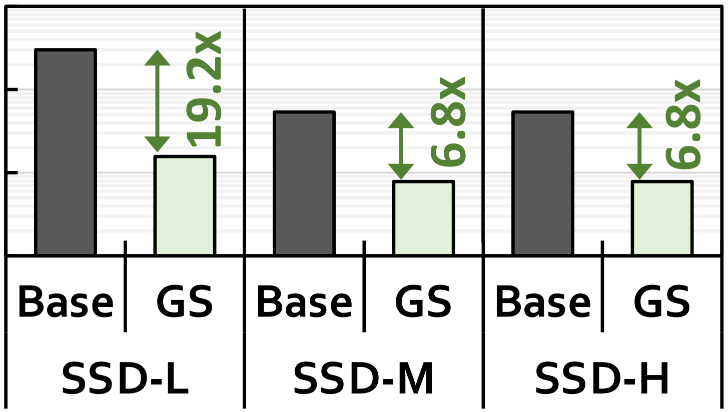
Performance – GenStore-NM

For a read set with 99.7% non-matching reads

With the Software Mapper



With the Hardware Mapper



22.4× – 27.9× speedup compared to the software Base

6.8× – 19.2× speedup compared to the hardware Base

On average 27.2× energy reduction

Area and Power

- Based on **Synthesis** of **GenStore** accelerators using the Synopsys Design Compiler @ 65nm technology node

Logic unit	# of instances	Area [mm²]	Power [mW]
Comparator	1 per SSD	0.0007	0.14
K -mer Window	2 per channel	0.0018	0.27
Hash Accelerator	2 per SSD	0.008	1.8
Location Buffer	1 per channel	0.00725	0.37375
Chaining Buffer	1 per channel	0.008	0.95
Chaining PE	1 per channel	0.004	0.98
Control	1 per SSD	0.0002	0.11
Total for an 8-channel SSD	-	0.2	26.6

Only 0.006% of a 14nm Intel Processor, less than 9.5% of the three ARM processors in a SATA SSD controller

Other Results in the Paper

- Effect of read set features on performance
  - Data size (up to 440 GB)
  - Filter ratio

- Performance benefit of an implementation of GenStore outside the SSD
  - In some cases, it provides performance benefits due more efficient streaming accesses
  - Provides significantly lower benefit compared to GenStore

- More detailed characterization of non-matching reads across different read mapping use cases and species

Outline

Background

Motivation and Goal

GenStore

Evaluation

Conclusions

Conclusion

- There has been significant effort into improving read mapping performance through efficient heuristics, hardware acceleration, accurate filters
- **Problem**: while these approaches address the computation overhead, none of them alleviate the **data movement overhead** from storage
- **Goal**: improve the performance of genome sequence analysis by effectively reducing unnecessary data movement from the storage system
- **Idea:** filter reads that **do not require the expensive alignment** computation **in the storage system** to fundamentally reduce the data movement overhead
- **Challenges**:
  - Read mapping workloads can exhibit **different behavior**
  - There are **limited available hardware resources** in the storage system
- **GenStore:** the *first* in-storage processing system designed for genome sequence analysis to reduce both the computation and data movement overhead
- **Key Results:** GenStore provides significant **speedup (1.4x - 33.6x)** and **energy reduction (3.9x – 29.2x)** at **low cost**

GenStore:
A High-Performance In-Storage Processing System for Genome Sequence Analysis

Nika Mansouri Ghiasi (*mnika@ethz.ch*)
Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun,
Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadhoun Alserr,
Rachata Ausavarungnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu

SAFARI

P&S Genomics

Lecture 10: GenStore

Nika Mansouri Ghiasi

ETH Zürich

Spring 2023
11 May 2023

Backup Slides

End-to-End Workflow of Genome Sequence Analysis

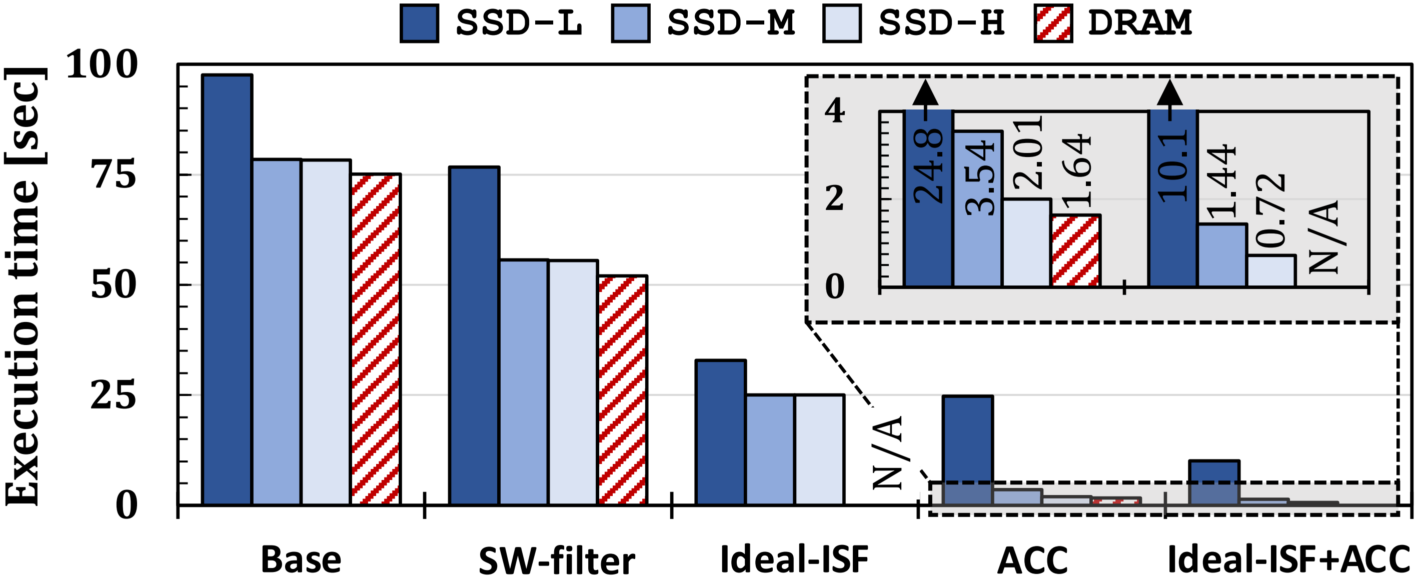
- There are three key initial steps in a standard genome sequencing and analysis workflow
  - Collection, preparation, and sequencing of a DNA sample in the laboratory
  - Basecalling
  - Read mapping

- Genomic read sets can be obtained by
  - Sequencing a DNA sample and storing the generated read set into the SSD of a sequencing machine
  - Downloading read sets from publicly available repositories and storing them into an SSD

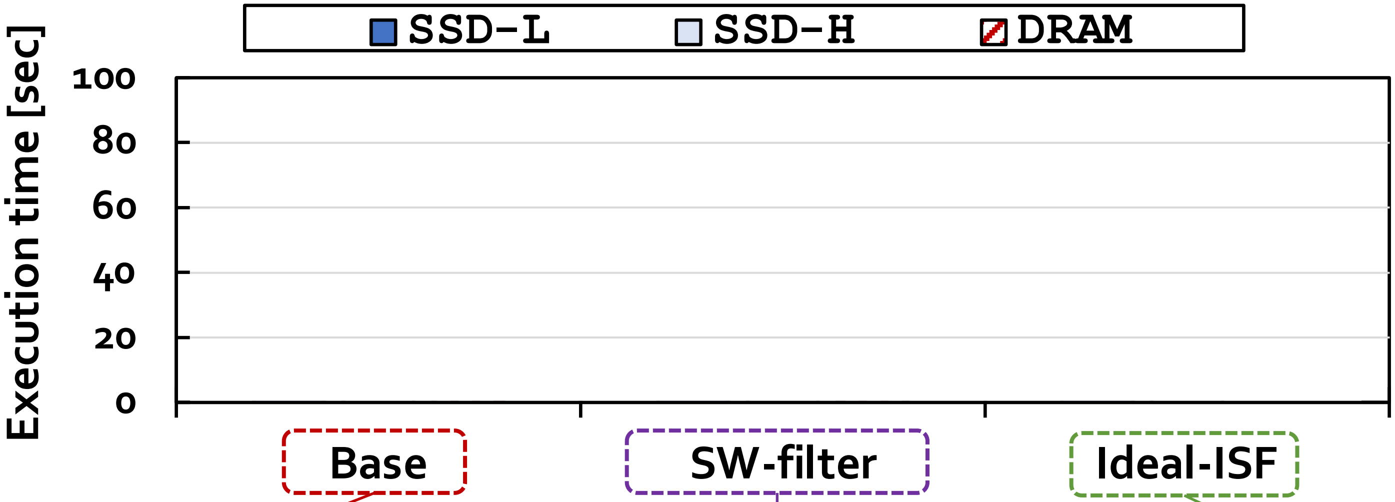
- We focus on optimizing the performance of read mapping because sequencing and basecalling are performed only once per read set, whereas read mapping can be performed many times
  - Analyzing the differences between a reads from an individual and many reference genomes of other individuals
  - Repeating the read mapping step many times to improve the outcome of read mapping

- Improving read mapping performance is critical in almost all genomic analyses that use sequencing
  - 45% of the execution time when discovering sequence variants in cancer genomics studies
  - 60% of the execution time when profiling the species composition of a multi-species (i.e., metagenomic) read

Motivation


SSD-L
SSD-M
SSD-H
DRAM
Execution time [sec]
100
75
50
25
0
Base
SW-filter
Ideal-ISF
N/A
ACC
Ideal-ISF+ACC
4
2
0
24.8
3.54
2.01
1.64
10.1
1.44
0.72
N/A

Motivation



State-of-the-art software read mapper, Minimap2

Base integrated with an ideal in-storage filter

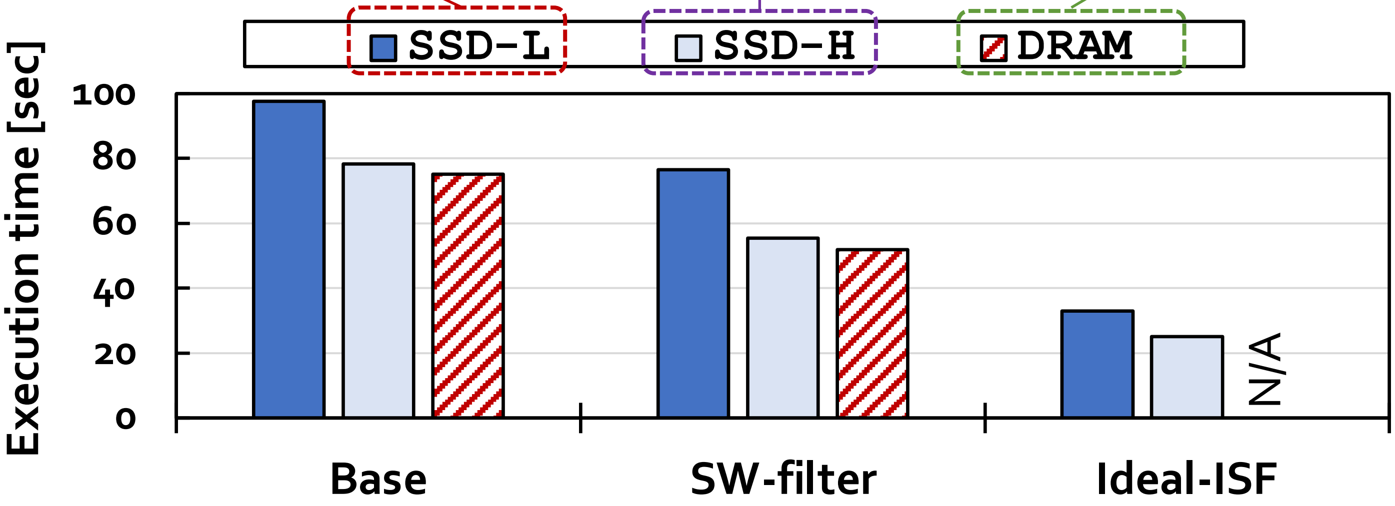
Base integrated with a software filter that prunes **80%** of exactly-matching reads

Motivation

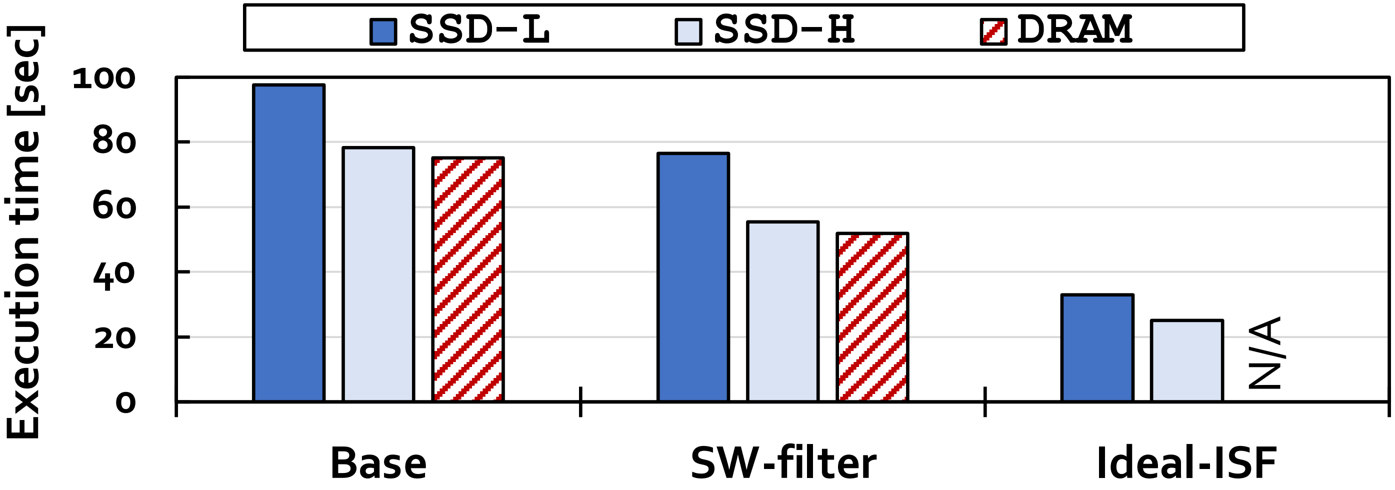
High-end SSD with PCIe Gen4 interface (7 GB/s)

Low-end SSD with SATA3 interface (0.5 GB/s)

Data preloaded in DRAM, with no I/O overhead



Benefits of Ideal In-Storage Filter

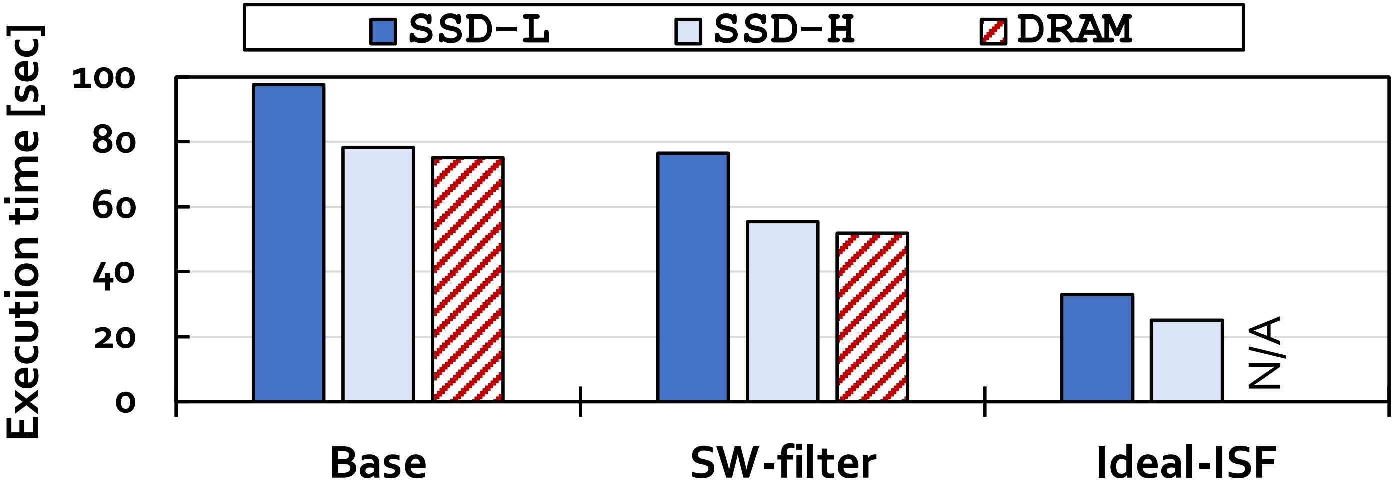


The ideal in-storage filter significantly improves performance by

1) Reducing computation overhead
2) Reducing data movement overhead

SAFARI

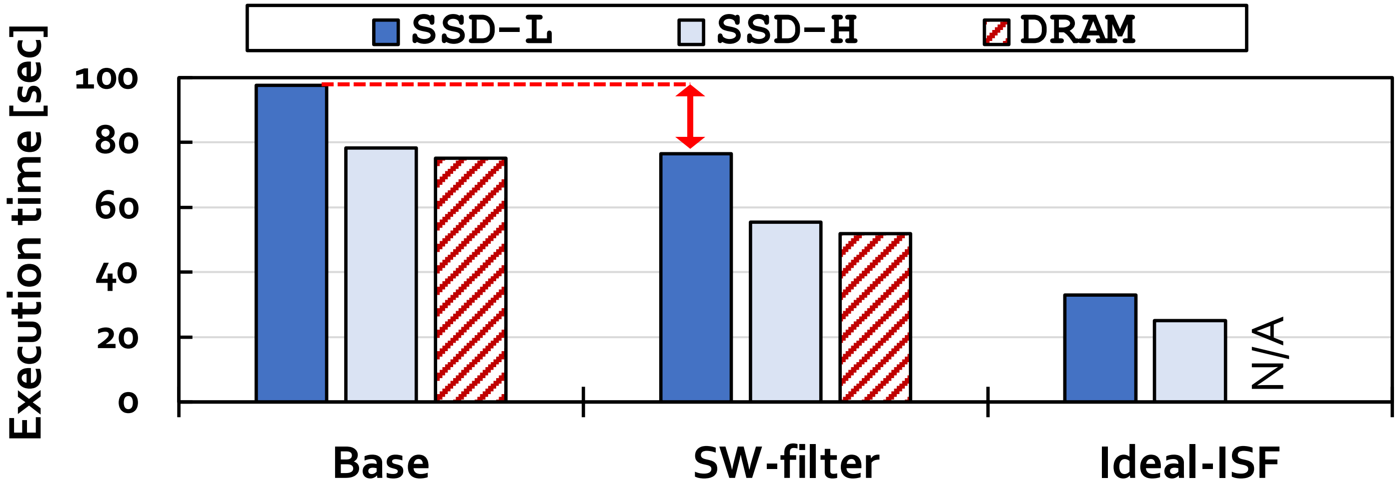
Overheads of Software Mappers



I/O has a significant impact on application performance

which can be alleviated at the cost of expensive storage devices and interfaces

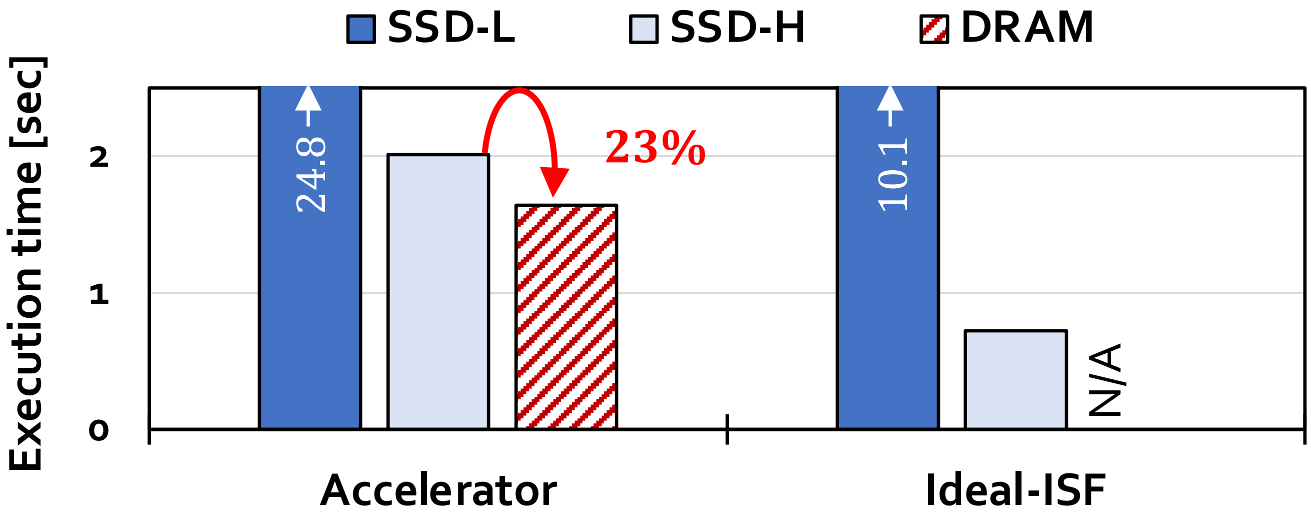
Overheads of Software Mappers



SW-filter provides limited benefits compared to Base

The filtering process outside the SSD must compete with the read mapping process for the resources in the system

Overheads of Hardware Mappers



Even the high-end SSD does not fully alleviate the storage bottleneck

The ideal in-storage filter significantly improves performance

Ideal-OSF

- Execution time of an ideal in-storage filter:

$$T_{\text{Ideal-ISF}} = T_{\text{I/O-Ref}} + \max\left\{T_{\text{I/O-Unfiltered}}, T_{\text{RM-Unfiltered}}\right\}$$

- Execution time of an ideal outside-storage filter:
  - 60% slower than Ideal-ISF in our analysis

$$T_{\text{Ideal-OSF}} = T_{\text{I/O-Ref}} + \max\left\{T_{\text{I/O-All-Reads}}, T_{\text{RM-Unfiltered}}\right\}$$

Comparison to PIM

- Even though read mapping applications could also benefit from other near-data, in-storage processing can fundamentally address the data movement problem by filtering large, low-reuse data where the data initially resides.

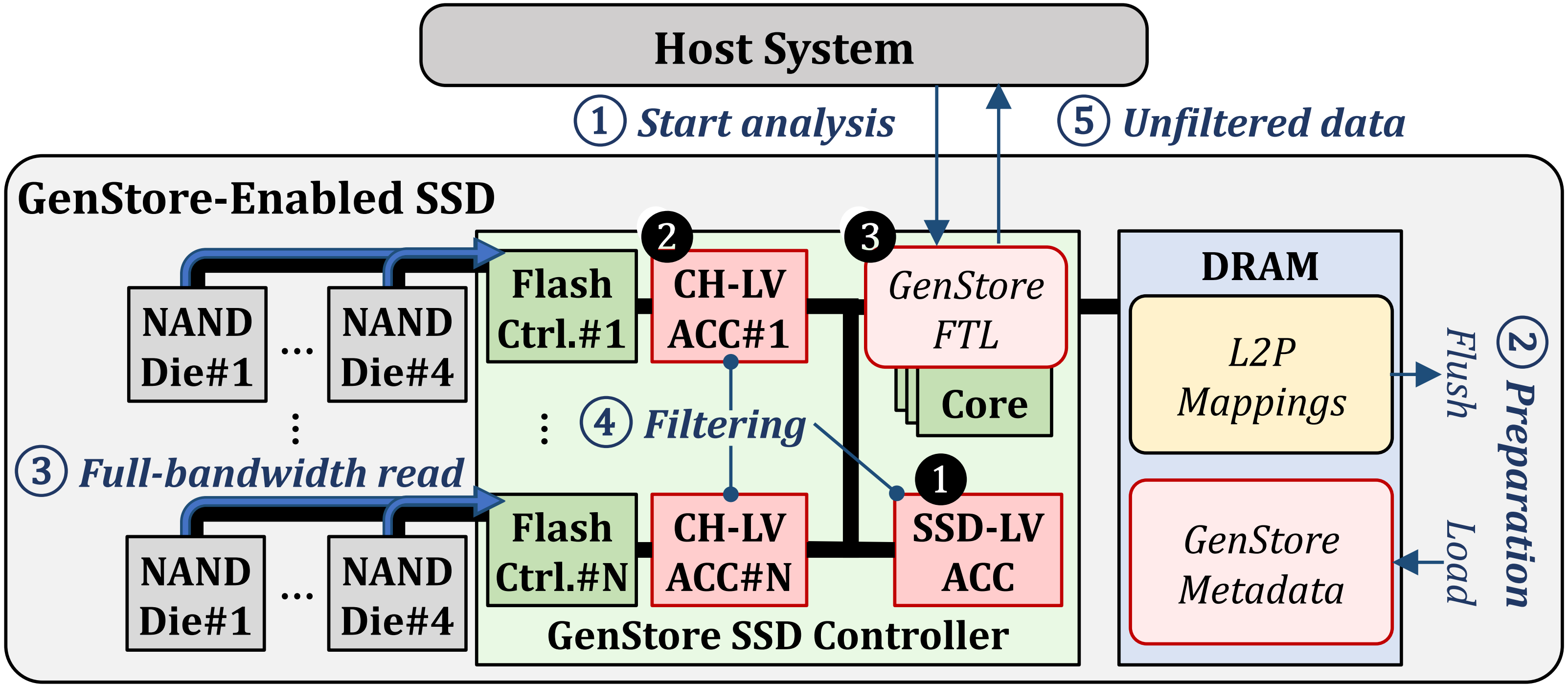
- Even if an ideal accelerator achieved a zero execution time, there would still exist the need to bring the data from storage to the accelerator.
  - 2.15x slower than the execution time that Ideal-ISF+ACC provides in our motivational analysis

In-storage filter can be integrated with any read mapping accelerator, including PIM accelerators, to alleviate their data movement overhead.

Long Read Use Cases

Use case	Input read set (Short/Long)	Size [GB]	Reference	Align [%]
Sequencing errors	ERR3988483 (L) [157]	54	hg38 [144]	**47.4**
	HG002_ONT_20200204 (L) [158]	371		**69.3**
Rapidly evolving samples	SRR5413248 (L) [157]	1.69	NZ_NJEX02 [159]	**60.0**
	SRR12423642 (S) [157]	0.466	NC_045512.2 [160]	**23.1**
No reference	SRR6767727 (L) [157]	12.4	NZ_NJEX02 [159]	**0.35**
	SRR9953689 (L) [157]	15.9		**37.0**
Contamination	SRR9953689 (L) [157]	15.9	hg38 [144]	**1.0**

FTL


Host System
① Start analysis
⑤ Unfiltered data
GenStore-Enabled SSD
NAND Die#1
…
NAND Die#4
⋮
③ Full-bandwidth read
NAND Die#1
…
NAND Die#4
Flash Ctrl.#1
❷
CH-LV ACC#1
❸
GenStore FTL
Core
⋮
④ Filtering
Flash Ctrl.#N
CH-LV ACC#N
❶
SSD-LV ACC
GenStore SSD Controller
DRAM
L2P Mappings
GenStore Metadata
Flush
Load
② Preparation

FTL: Metadata

- GenStore metadata includes the mapping information of the data structures necessary for read mapping acceleration

- In accelerator mode, GenStore also keeps in internal DRAM other metadata structures of the regular FTL
  - Examples include the page status table and block read counts which need to be updated during the filtering process

- We carefully design GenStore to only sequentially access the underlying NAND flash chips while operating as an accelerator
  - Requires only a small amount of metadata to access the stored data

FTL: Data Placement

- GenStore needs to properly place its data structures to enable the full utilization of the internal SSD bandwidth
- When each data structure is initially written to the SSD, GenStore sequentially and evenly distributes it across NAND flash chips
- GenStore can specify the physical location of a 30-GB data structure by maintaining only the list of 1,250 (30 GB/24 MB) physical block addresses
- It significantly reduces the size of the necessary mapping information from 300 MB (with conventional 4-KiB page mapping) to only 5 KB (1,250 4 bytes)

FTL: SSD Management Tasks

- In accelerator mode, GenStore only reads data structures to perform filtering, and does not write any new data
  - GenStore does not require any write-related SSD-management tasks such as garbage collection and wear-leveling

- The other tasks necessary for ensuring data reliability can be done before or after the filtering process
  - GenStore significantly limits the amount of data whose retention age would exceed the manufacturer-specified threshold since GenStore's filtering process takes a short time.
  - GenStore-FTL can easily avoid read disturbance errors for data with high read counts since GenStore sequentially reads NAND flash blocks only once during filtering

Data Sizes

- Conventional k-mer index in Minimap2 + reference genome: 7 GB (k = 15)

- Read-sized k-mer index before optimization: 126 GB (k= 150)

- Read-sized k-mer index after optimization: 32 GB (k = 150)

SSD Specs

- **SSD-L:** SATA3 interface (0.5 GB/s sequential read)
  - 1.2 GB/s per channel bandwidth
  - 8 channels

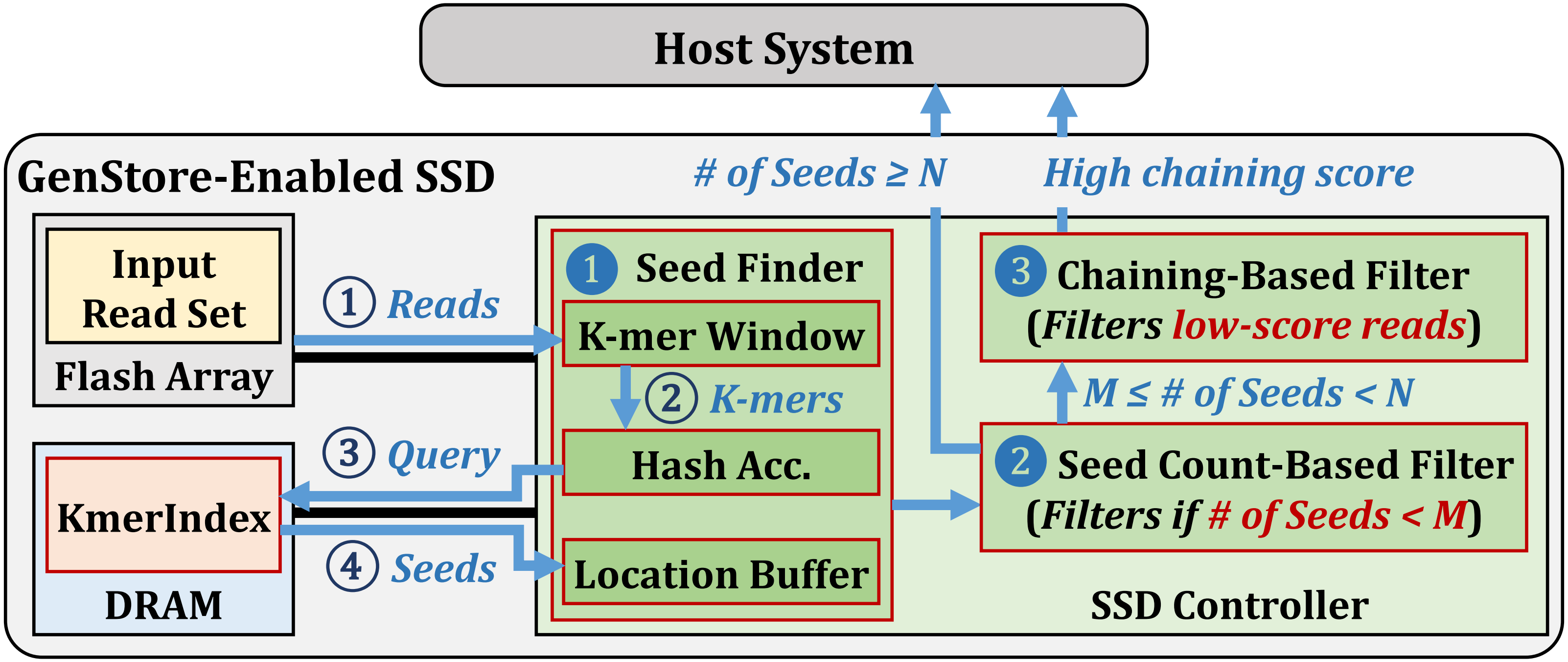
- **SSD-L:** PCIe Gen3 M.2 interface (3.5 GB/s sequential read)
  - 1.2 GB/s per channel bandwidth
  - 16 channels

- **SSD-L:** PCIe Gen4 interface (7 GB/s sequential read)
  - 1.2 GB/s per channel bandwidth
  - 16 channels

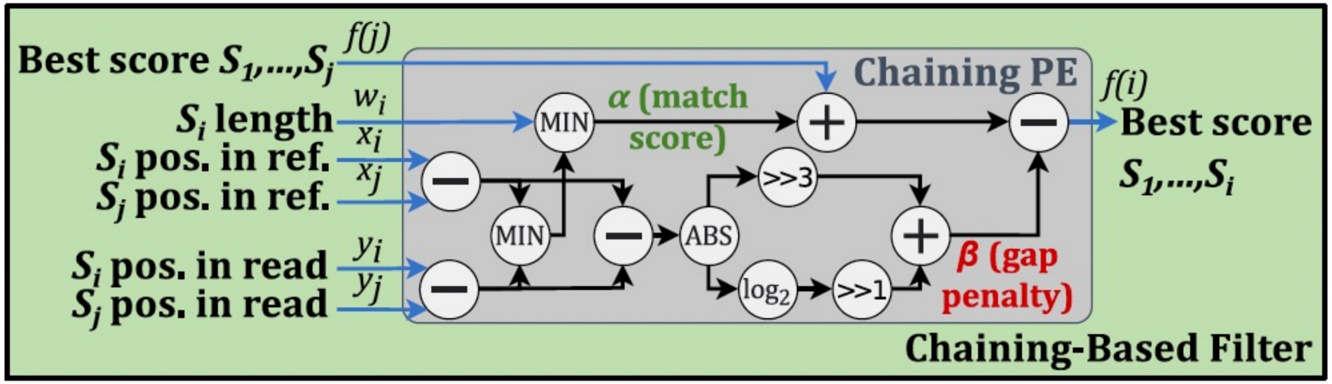
Evaluation Methodology

- **Performance modeling**
  - Ramulator for DRAM timing
  - MQSim for SSD timing
  - We model the end-to-end throughput of GenStore based on the throughput of each GenStore pipeline stage
    - Accessing NAND flash chips
    - Accessing internal DRAM
    - Accelerator computation
    - Transferring unfiltered data to the host
- **Real system results**
  - AMD EPYC 7742 CPU
  - 1TB DDR4 DRAM
  - AMD µProf

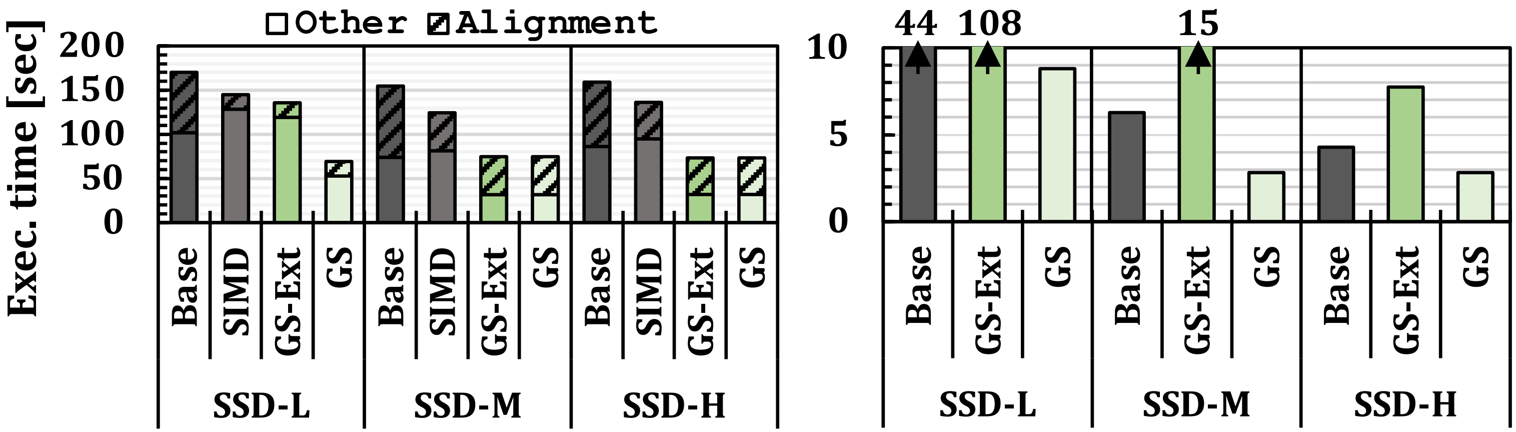
GenStore-NM


Host System
GenStore-Enabled SSD
# of Seeds ≥ N
High chaining score
Input Read Set
Flash Array
① Reads
③ Query
④ Seeds
KmerIndex
DRAM
1 Seed Finder
K-mer Window
② K-mers
Hash Acc.
Location Buffer
3 Chaining-Based Filter
(Filters low-score reads)
M ≤ # of Seeds < N
2 Seed Count-Based Filter
(Filters if # of Seeds < M)
SSD Controller

Chaining Processing Element



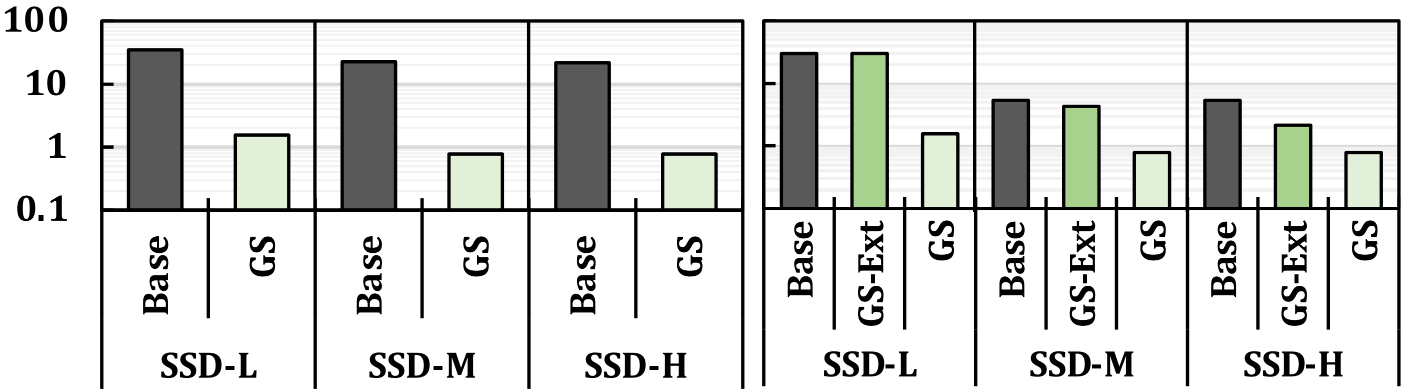
GenStore-EM



GS-Ext provides significant performance improvements over both Base and SIMD in SSD-M and SSD-H.

GS-Ext provides limited benefits over SIMD in SSD-L due to low external I/O bandwidth.

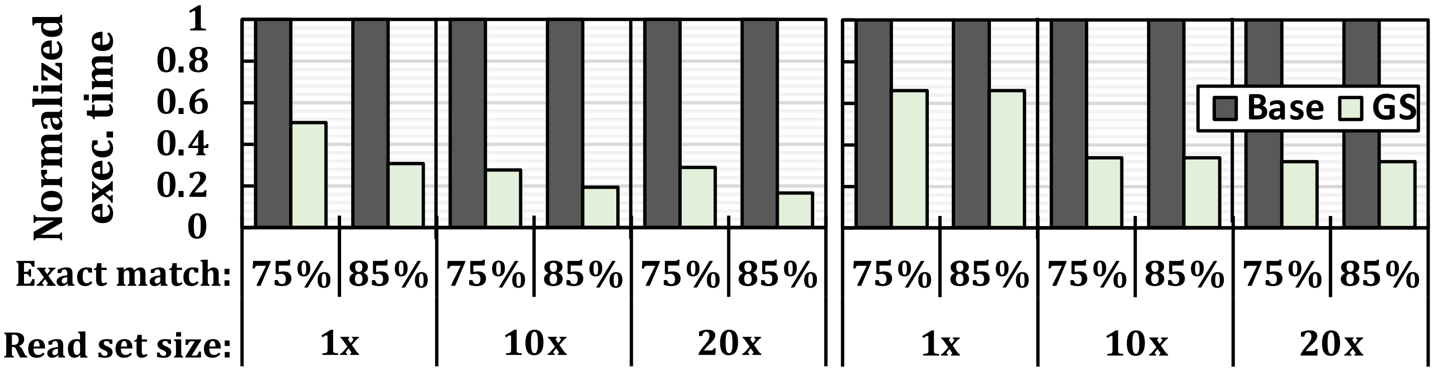
GenStore-NM



GS-Ext performs significantly slower than Base (2.28x - 1.91x) on all systems.

Effect of Inputs on GenStore-EM

$$DM\_Saving = \frac{Size_{Ref} + Size_{ReadSet}}{Size_{Ref} + Size_{ReadSet} \times (1 - Ratio_{Filter})}$$



Effect of Inputs on GenStore-NM

$$DM\_Saving = \frac{Size_{Ref} + Size_{ReadSet}}{Size_{Ref} + Size_{ReadSet} \times (1 - Ratio_{Filter})}$$