Storage-Centric System Designs for Enabling Fast, Efficient, and Low-Cost Genomic and Metagenomic Analyses

Nika Mansourighiasi

Doctoral Examination
31.03.2026

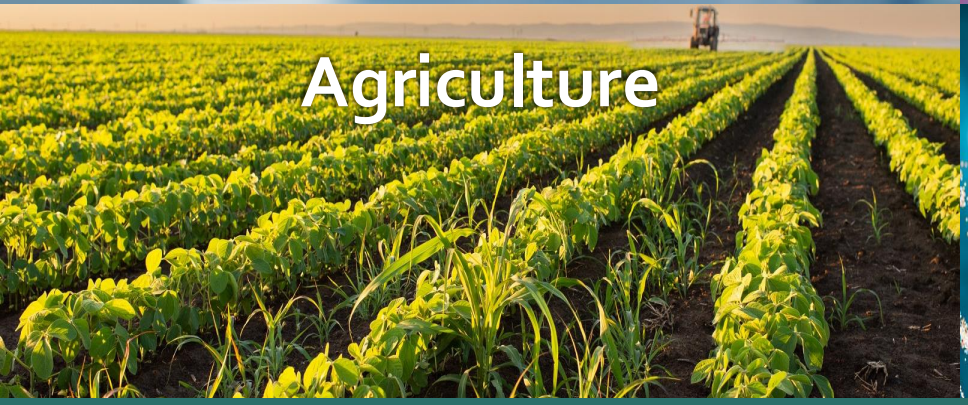
Advisor:
Onur Mutlu (ETH Zurich)

Co-Examiners:
Can Alkan (Bilkent University)
Reetuparna Das (University of Michigan)
Wen-mei Hwu (NVIDIA & University of Illinois Urbana-Champaign)
Jangwoo Kim (Seoul National University & MangoBoost)
Yatish Turakhia (University of California San Diego)

SAFARI

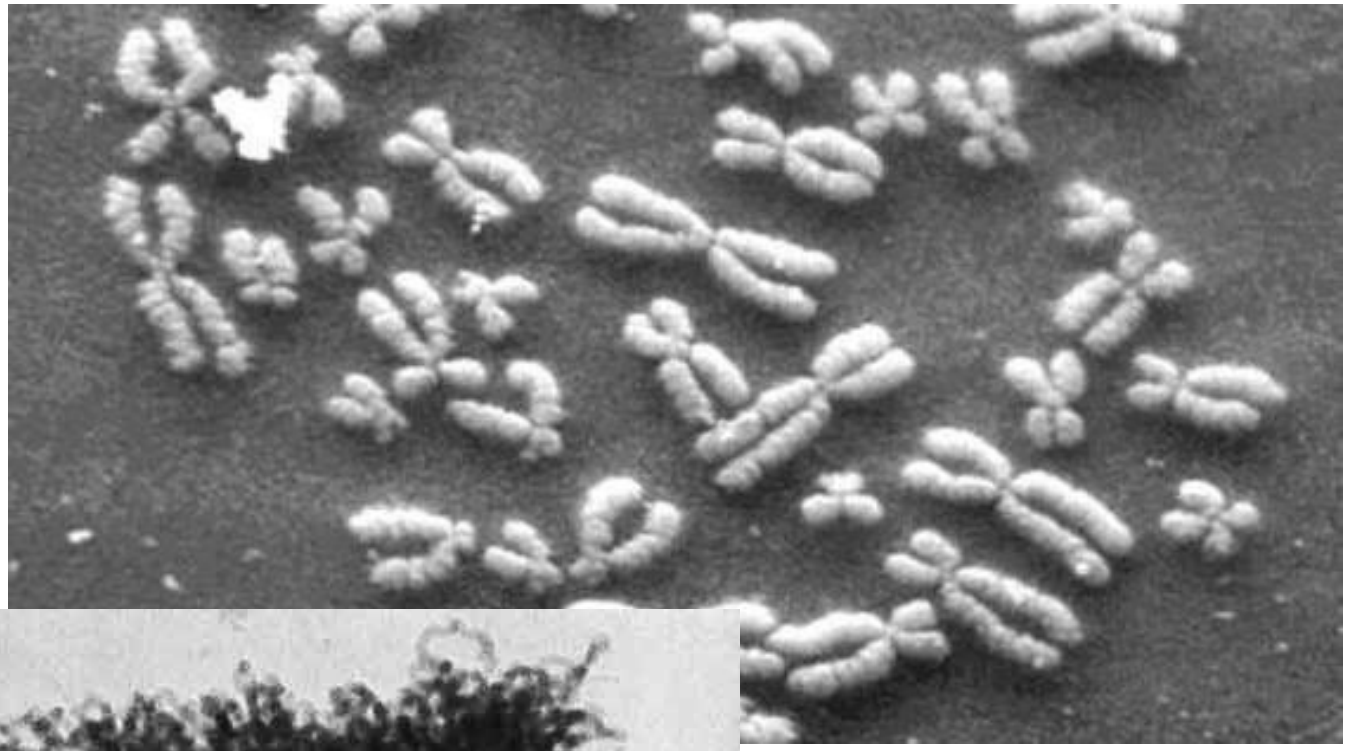
ETH zürich

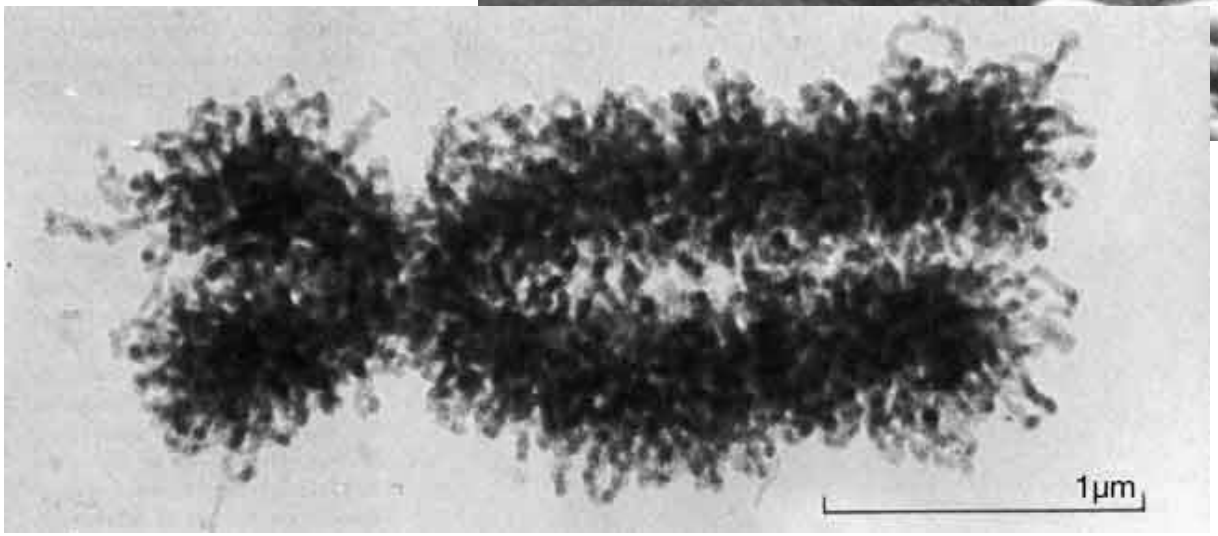
Applications of (Meta)Genome Analysis



& Many More…

DNA Under Electron Microscope





human chromosome #12
from HeLa cell

High-Throughput Sequencers

Illumina MiSeq

Pacific Biosciences Sequel II

Oxford Nanopore PromethION

Oxford Nanopore MinION

Illumina NovaSeq 6000

Pacific Biosciences RS II

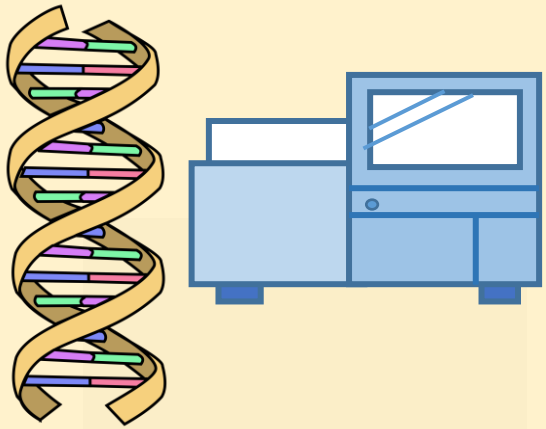
Oxford Nanopore GridION

… and more! All produce data with different properties.

High-Throughput Sequencers

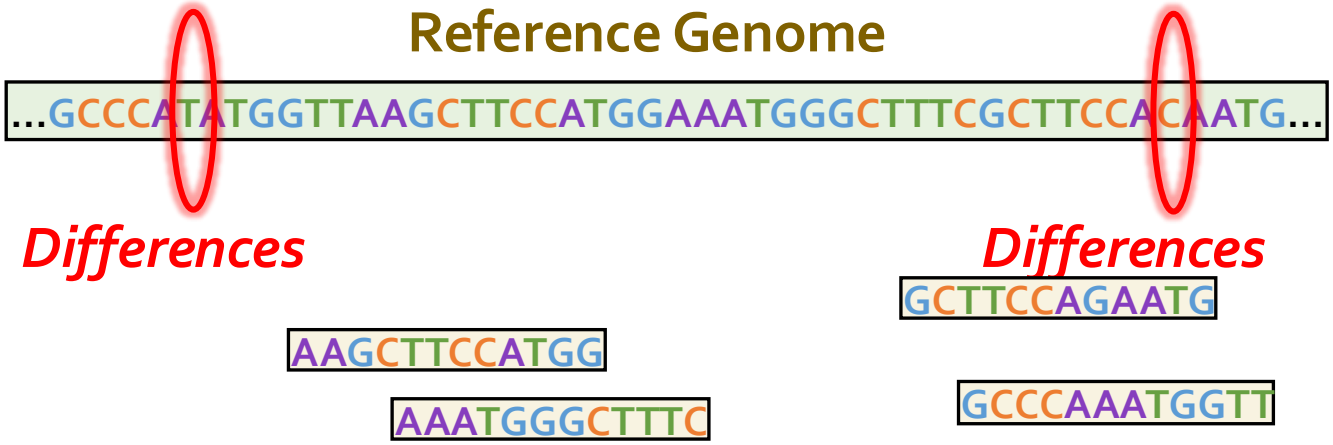
Current sequencing technologies cannot sequence very long DNA molecules end-to-end

Instead, they extract smaller fragments of the original DNA sequence, known as reads



Genome Sequence Analysis

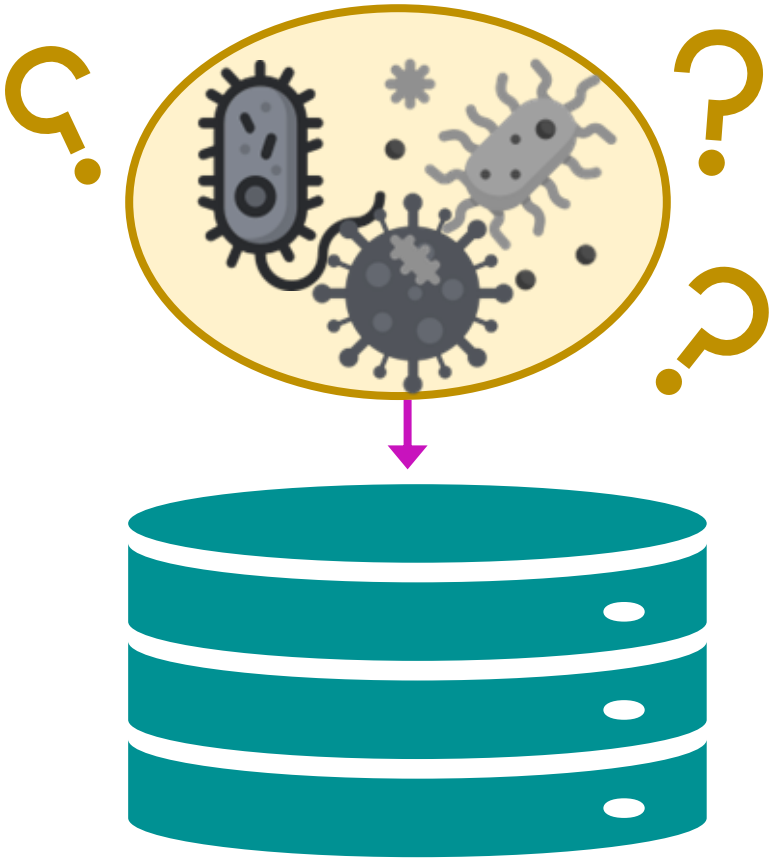
- **Read mapping:** first key step in genome sequence analysis
  - Aligns reads to potential matching locations in the reference genome
  - For each matching location, the alignment step finds the degree of similarity (alignment score)



- Calculating the alignment score requires computationally-expensive approximate string matching (ASM) to account for differences between reads and the reference genome

What is Metagenomics?

- ***Metagenomics:*** Study of genome sequences of **diverse organisms** within a **shared environment** (e.g., blood, ocean, soil)



A large database containing information on **many species**

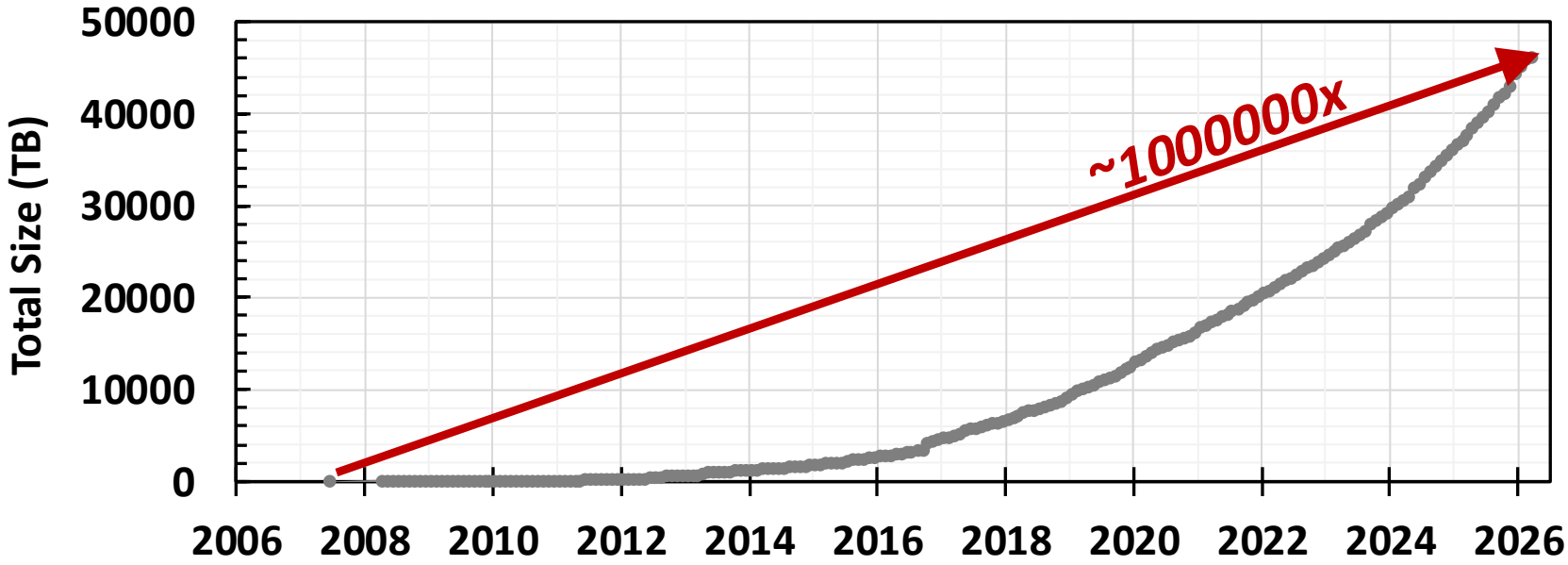
(e.g., > 100 TBs in emerging databases)

Sequence Data is Growing at a Fast Pace

Important Role in Many Fields

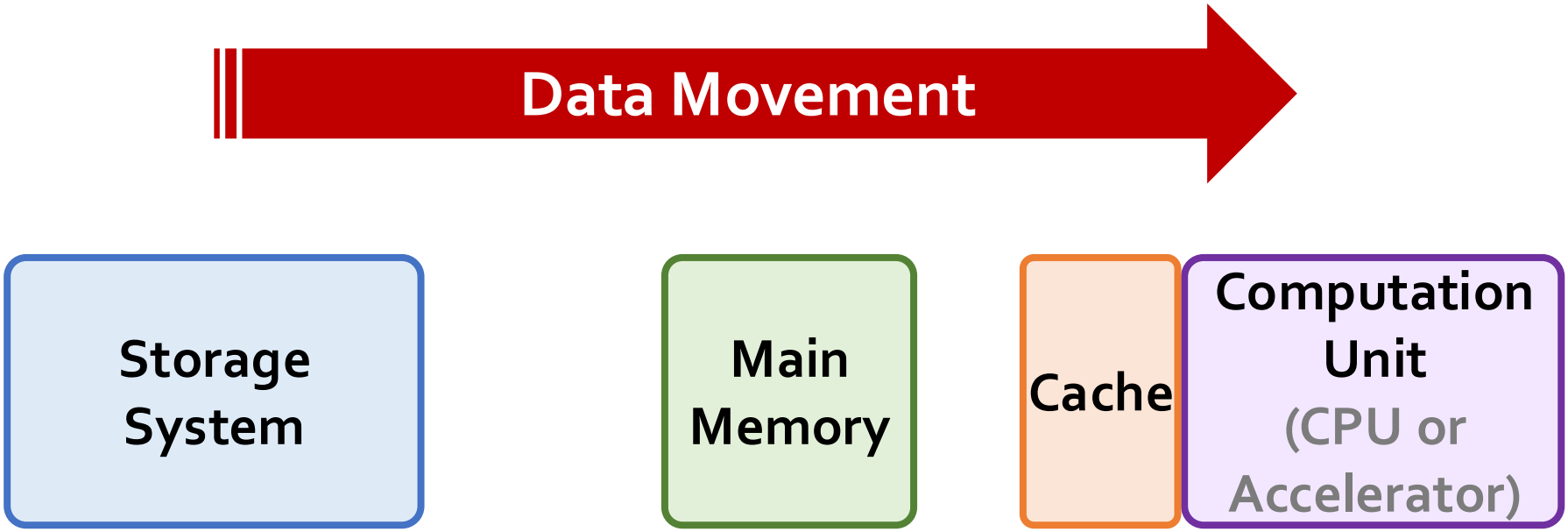
Large Reduction in Sequencing Cost

Fast growth in sequence data size



Publicly-Hosted Genomic Sequence Reads Data Size in Sequence Read Archive (SRA) Over Years

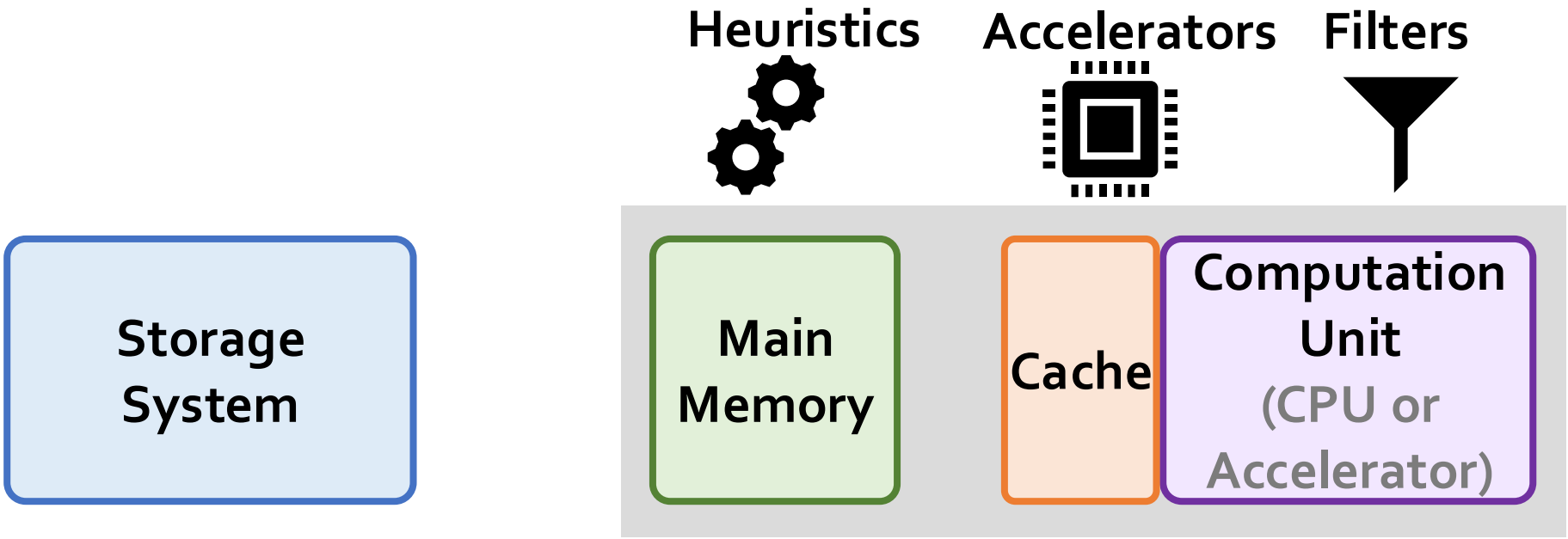
Overheads of (Meta)Genomic Analyses



✕ Computation overhead

✕ Data movement overhead

Accelerating (Meta)Genome Sequence Analysis



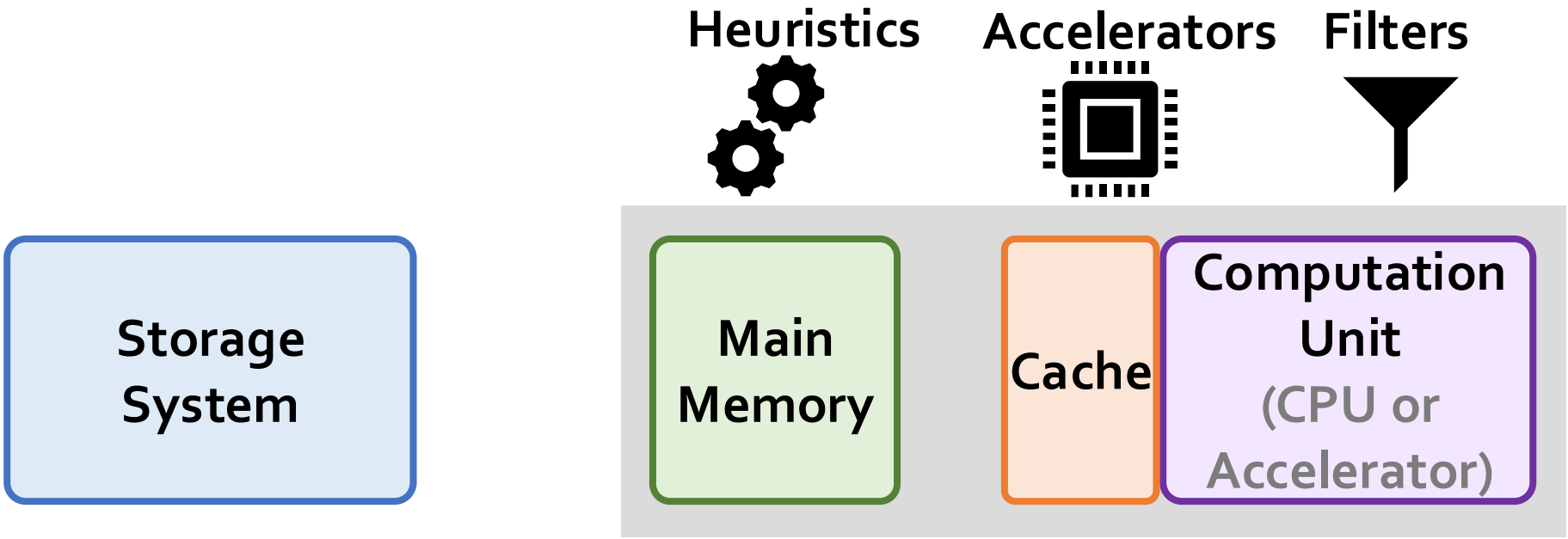
✓ Computation overhead

Problems

Accessing stored sequence data and feeding it to the analysis units

Overhead of **moving large amounts of low-reuse data**
from the **storage system**
& **unnecessary computational burden** on the **rest of the system**

Accelerating (Meta)Genome Sequence Analysis



✓ Computation overhead

✗ Data movement overhead

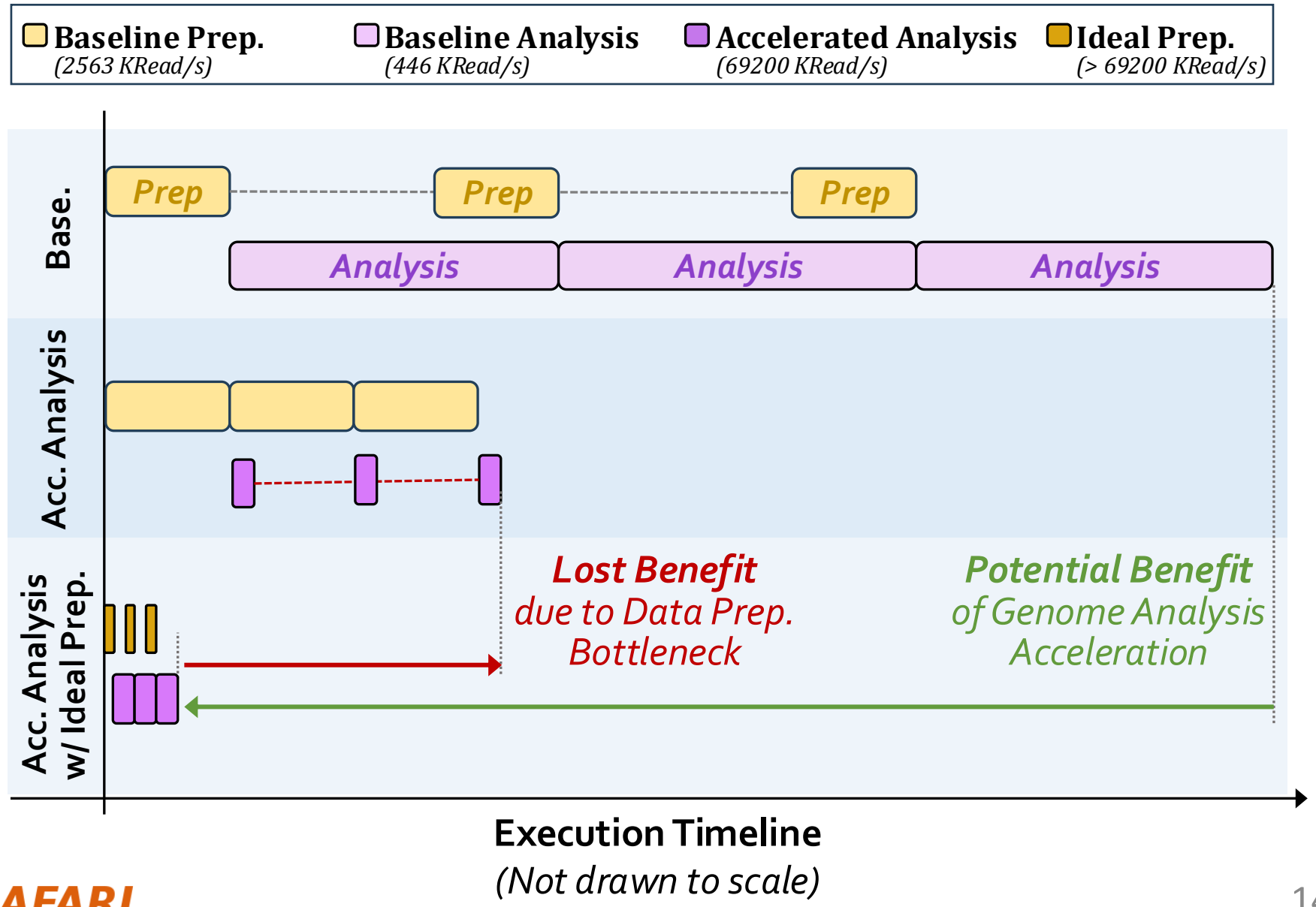
Problems

Accessing stored sequence data and feeding it to the analysis units

Overhead of **moving large amounts of low-reuse data**
from the **storage system**
& **unnecessary computational burden** on the **rest of the system**

Data preparation bottleneck,
where compressed genomic sequence data needs to
be first **decompressed** and **formatted** before it can be analyzed

Effect of the Data Preparation Bottleneck



Thesis Statement

*By designing customized **storage-centric systems** that efficiently*

① ***analyze** genomic and metagenomic data **inside the storage system***

② *enable **highly-compressed storage** and **high-performance access** of large-scale sequence data*

we can alleviate the overheads of

data movement *computation* *data preparation*

thereby, significantly improving

performance *energy efficiency* *cost effectiveness*

Our Approach

*Designing customized **storage-centric systems** that efficiently*

① ***analyze** genomic and metagenomic data **inside the storage system***

② *enable **highly-compressed storage** and **high-performance access** of large-scale sequence data*

we can alleviate the overheads of

data movement *computation* *data preparation*

thereby, significantly improving

performance *energy efficiency* *cost effectiveness*

Our Approach

*Designing customized **storage-centric systems** that efficiently*

① ***analyze** genomic and metagenomic data **inside the storage system***

1 GenStore [ASPLOS'22]
In-storage filtering
for genomic analysis

2 MegIS [ISCA'24]
Cooperative in-storage processing
for metagenomic analysis

3 GRAINS [ISCA'26]
Storage-aware algorithm-architecture co-design
for graph-based (meta)genomic analyses

② *enable **highly-compressed storage** and **high-performance access** of large-scale sequence data*

4 SAGe [HPCA'26]
Algorithm-architecture co-design
for storing and accessing sequence data

Our Approach

*Designing customized **storage-centric systems** that efficiently*

① ***analyze** genomic and metagenomic data **inside the storage system***

1 GenStore [ASPLOS'22]
In-storage filtering
for genomic analysis

2 MegIS [ISCA'24]
Cooperative in-storage processing
for metagenomic analysis

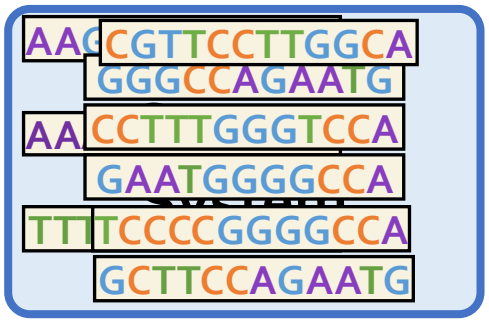
3 GRAINS [ISCA'26]
Storage-aware algorithm-architecture co-design
for graph-based (meta)genomic analyses

② *enable **highly-compressed storage** and **high-performance access** of large-scale sequence data*

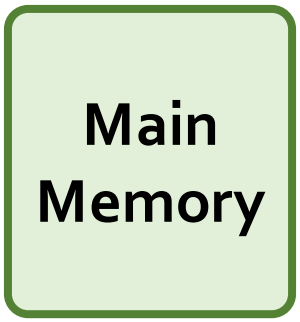
4 SAGe [HPCA'26]
Algorithm-architecture co-design
for storing and accessing sequence data

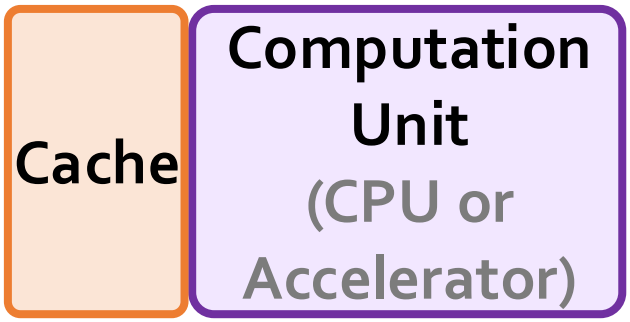
Key Idea

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



Filtered Reads





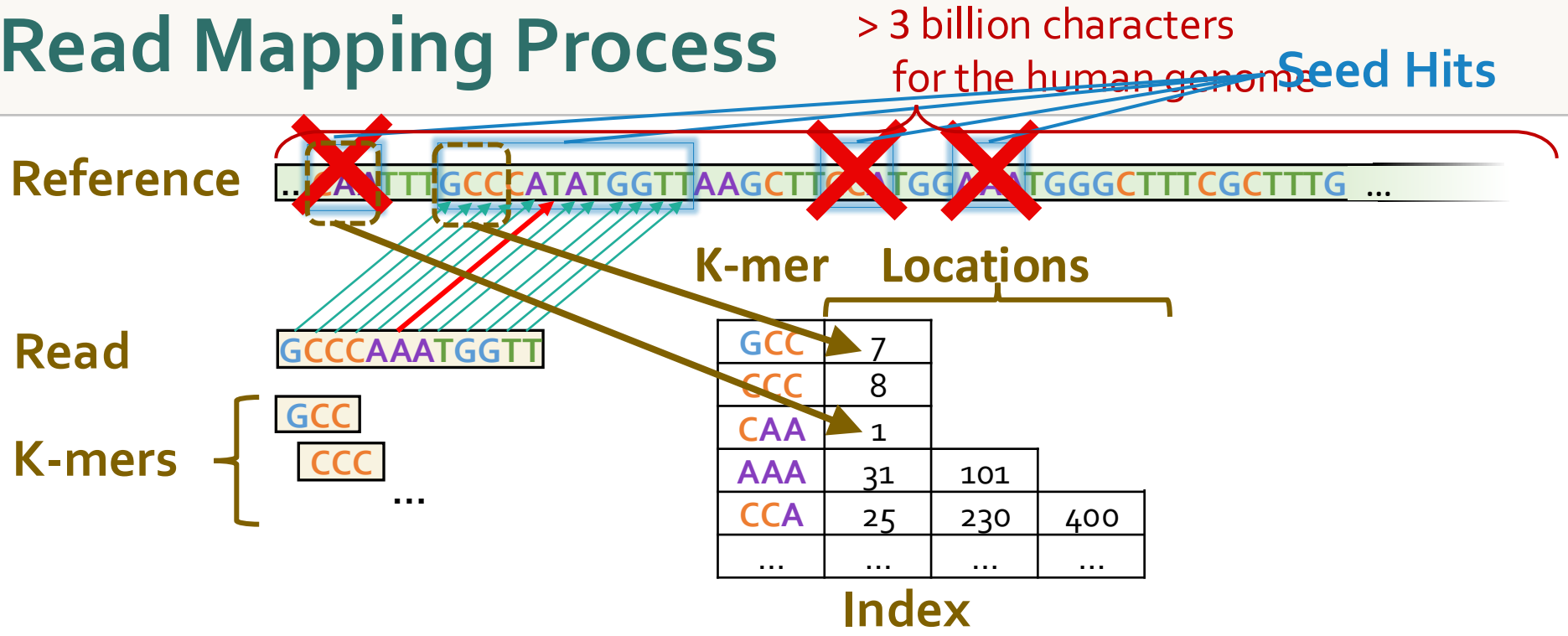
Exactly-matching reads

Do not need expensive approximate string matching during alignment

Non-matching reads

Do not have potential matching locations and can skip alignment

Read Mapping Process



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

GenStore

GenStore for Exactly-Matching Reads

GenStore for Non-Matching Reads

GenStore

GenStore for Exactly-Matching Reads

GenStore for Non-Matching Reads

GenStore for Exactly-Matching Reads

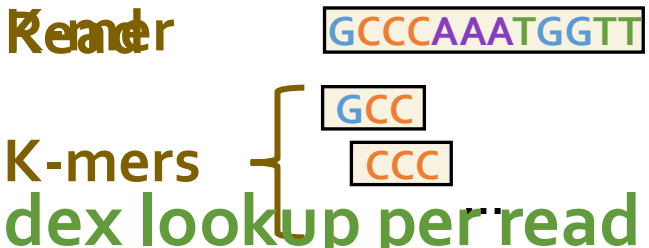
- Efficient in-storage filter for reads with at least one exact match in the reference genome
- Uses simple operations, without requiring alignment
- **Challenge:** Large number of random accesses per read to the reference genome and its index

Expensive random accesses to flash chips

Limited DRAM capacity inside the SSD

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

Index 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
…	…

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

Evaluation Methodology

Read Mappers

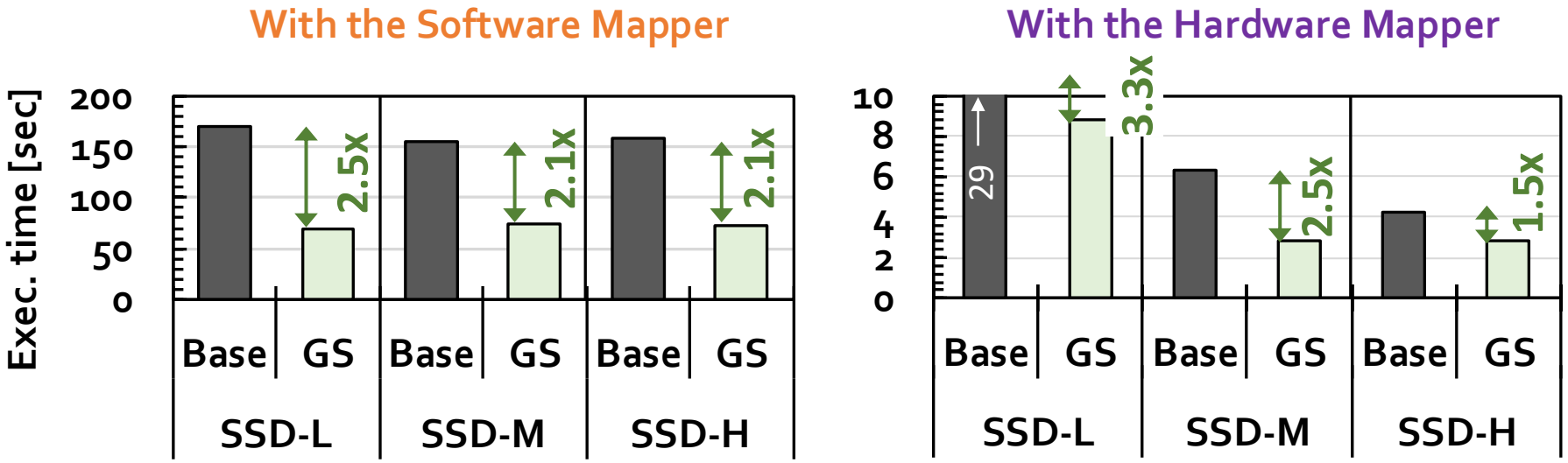
- **Base:** state-of-the-art software or hardware read mappers
  - Minimap2 [Bioinformatics'18]: software mapper for short and long reads
  - GenCache [MICRO'19]: hardware mapper for short reads
  - Darwin [ASPLOS'18]: hardware mapper for long reads
- **GS:** Base integrated with GenStore

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)

GenStore for Exactly-Matching Reads

For a read set with 80% exactly-matching reads



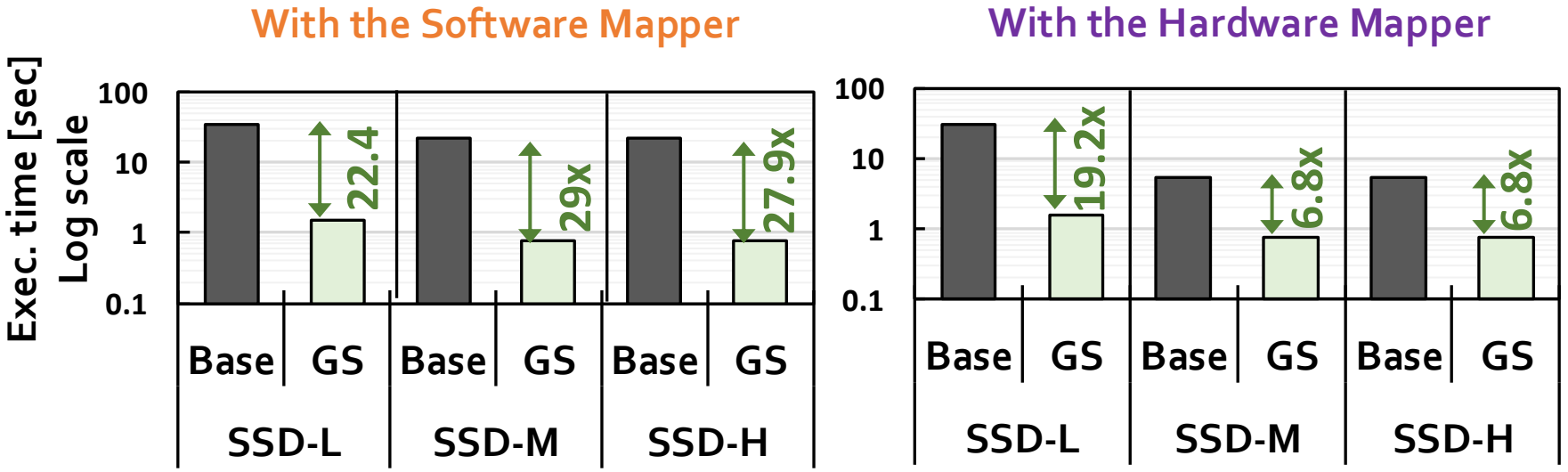
2.1× - 2.5× speedup compared to the software Base

1.5× – 3.3× speedup compared to the hardware Base

On average 3.92× energy reduction

GenStore for Non-Matching Reads

For a read set with 99.7% non-matching reads



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

Our Approach

*Designing customized **storage-centric systems** that efficiently*

① ***analyze** genomic and metagenomic data **inside the storage system***

1 GenStore [ASPLOS'22]
In-storage filtering
for genomic analysis

2 MegIS [ISCA'24]
Cooperative in-storage processing
for metagenomic analysis

3 GRAINS [ISCA'26]
Storage-aware algorithm-architecture co-design
for graph-based (meta)genomic analyses

② *enable **highly-compressed storage** and **high-performance access** of large-scale sequence data*

4 SAGe [HPCA'26]
Algorithm-architecture co-design
for storing and accessing sequence data

Our Approach

*Designing customized **storage-centric systems** that efficiently*

① ***analyze** genomic and metagenomic data **inside the storage system***

1 GenStore [ASPLOS'22]
In-storage filtering
for genomic analysis

2 MegIS [ISCA'24]
Cooperative in-storage processing
for metagenomic analysis

3 GRAINS [ISCA'26]
Storage-aware algorithm-architecture co-design
for graph-based (meta)genomic analyses

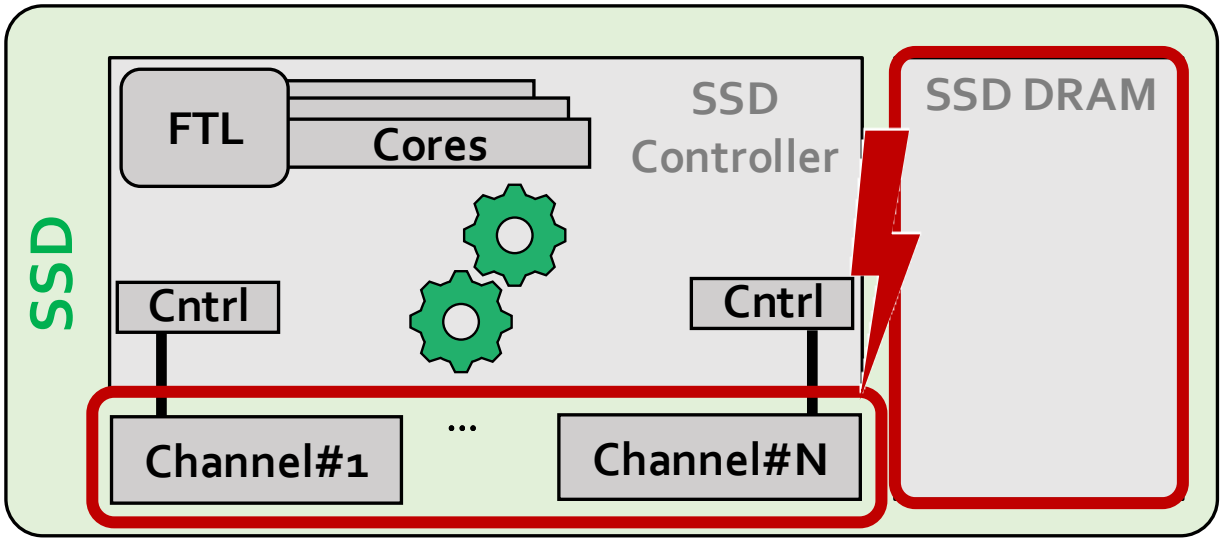
② *enable **highly-compressed storage** and **high-performance access** of large-scale sequence data*

4 SAGe [HPCA'26]
Algorithm-architecture co-design
for storing and accessing sequence data

Challenges of Storage-Centric Computing

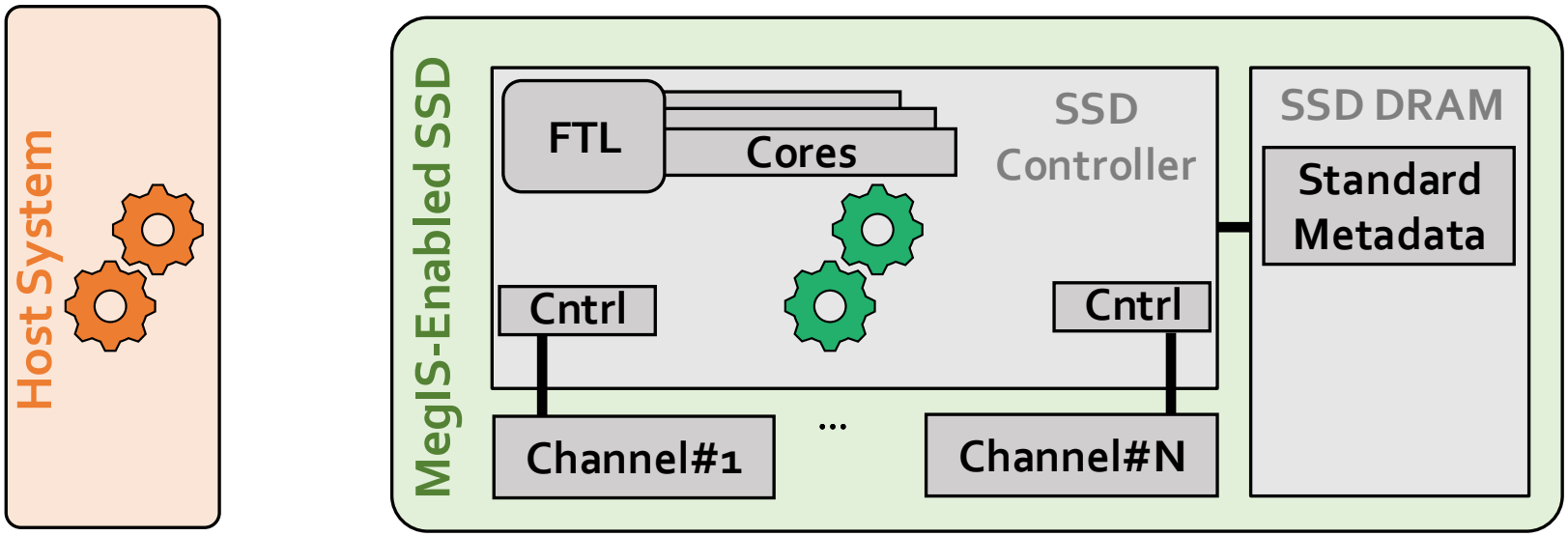
Metagenomic analysis tools cannot run in-storage due to SSD limits

- Long **latency of NAND flash** chips
- Limited **DRAM capacity** inside the SSD
- Limited **DRAM bandwidth** inside the SSD

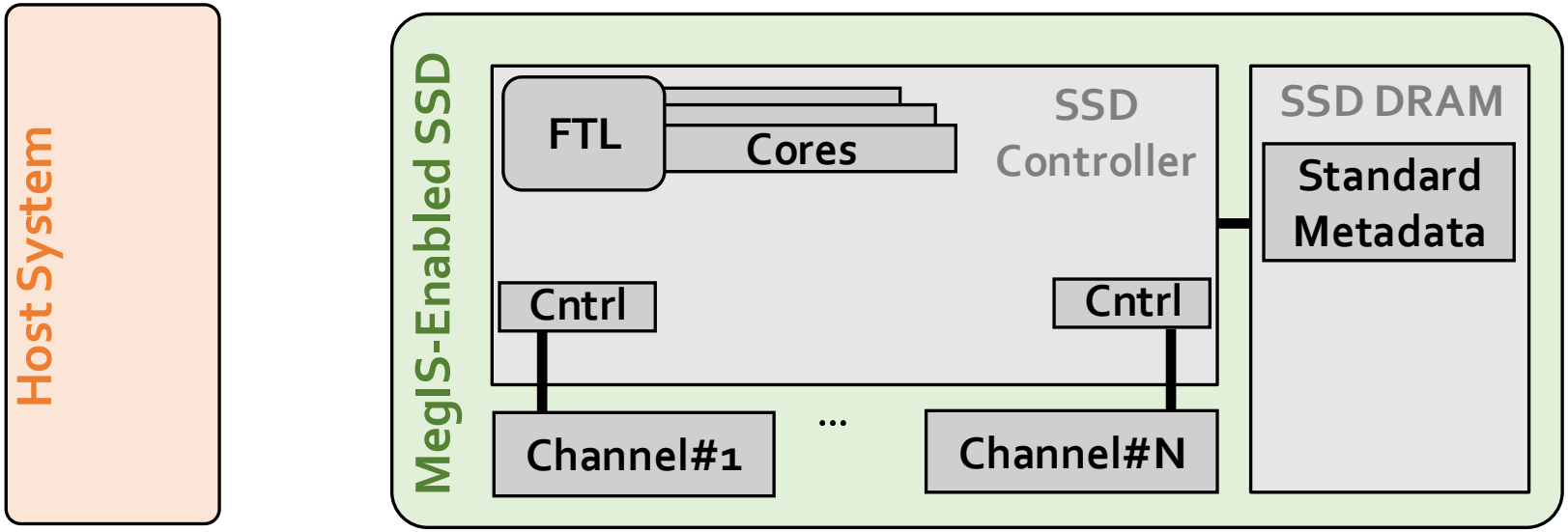


MegIS: Metagenomics In-Storage

- First in-storage system for *end-to-end* metagenomic analysis
- **Idea:** Cooperative in-storage processing for metagenomic analysis
  - Algorithm-Architecture co-design between the



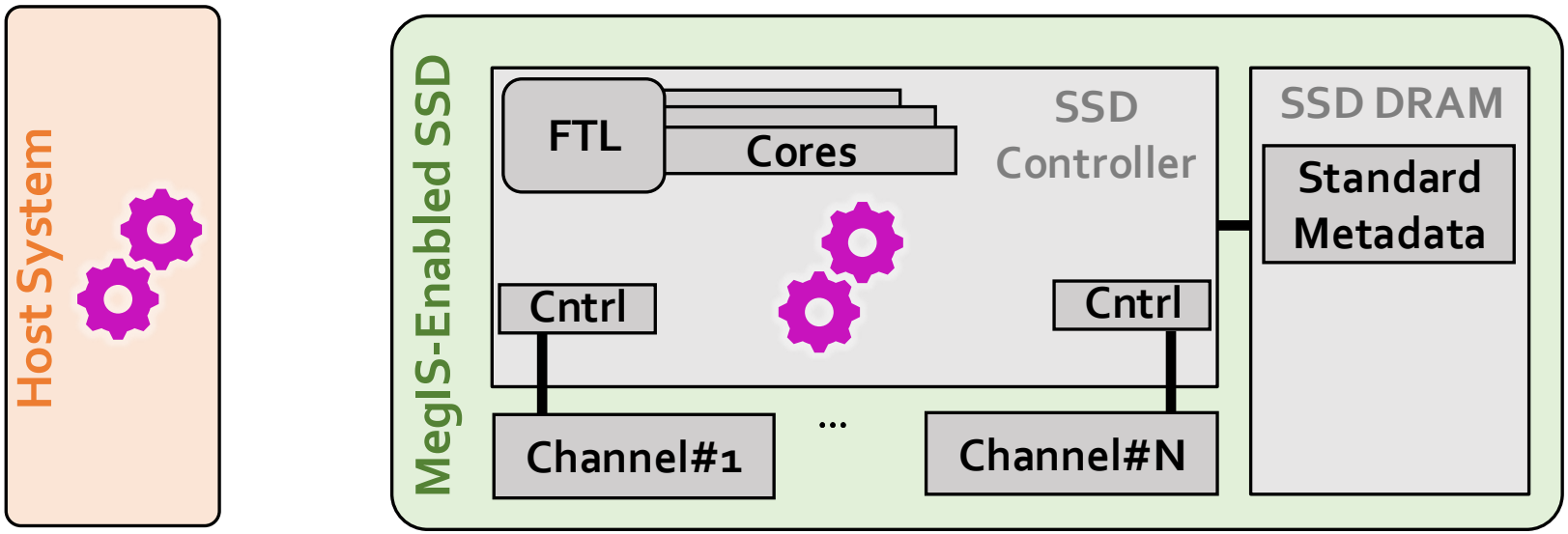
MegIS Algorithm-Architecture Co-Design



MegIS Algorithm-Architecture Co-Design

Task partitioning and mapping

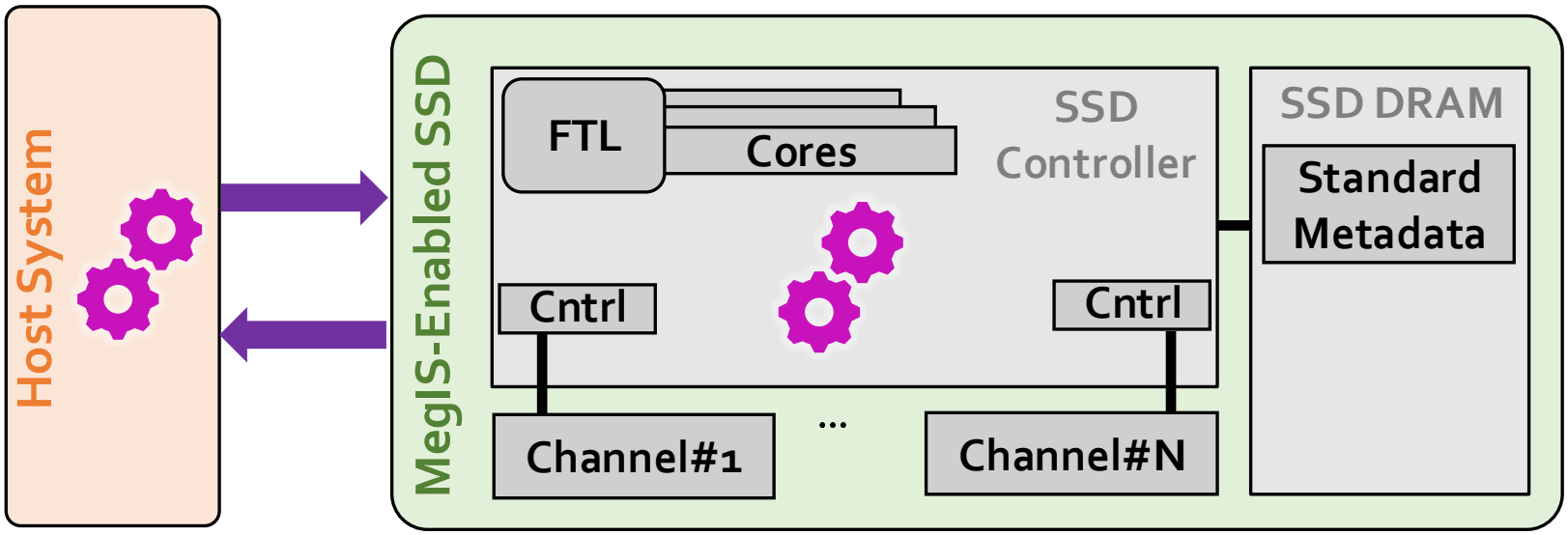
- *Each step executes in its most suitable system*



MegIS Algorithm-Architecture Co-Design

Task partitioning and mapping
- *Each step executes in its most suitable system*

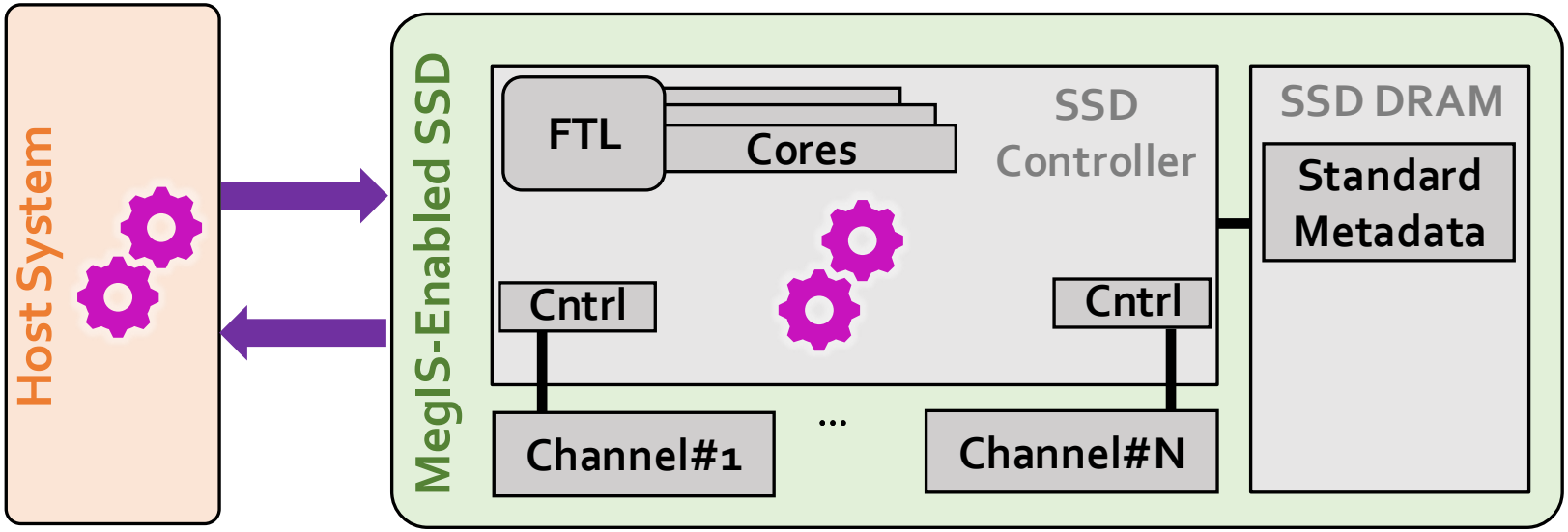
Data/computation flow coordination
- *Reduce communication overhead*
- *Reduce #writes to flash chips*



MegIS Algorithm-Architecture Co-Design

Task partitioning and mapping
- *Each step executes in its most suitable system*

Data/computation flow coordination
- *Reduce communication overhead*
- *Reduce #writes to flash chips*

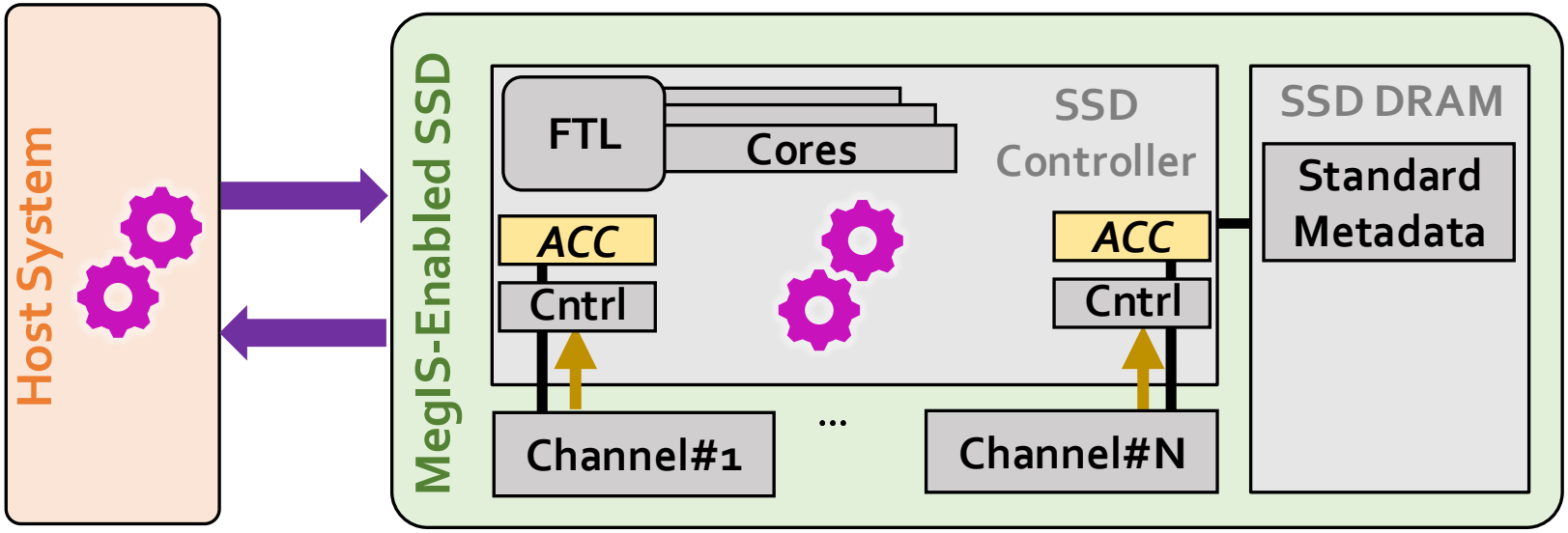


Storage-aware algorithms
- *Enable efficient access patterns to the SSD*

MegIS Algorithm-Architecture Co-Design

Task partitioning and mapping
- *Each step executes in its most suitable system*

Data/computation flow coordination
- *Reduce communication overhead*
- *Reduce #writes to flash chips*



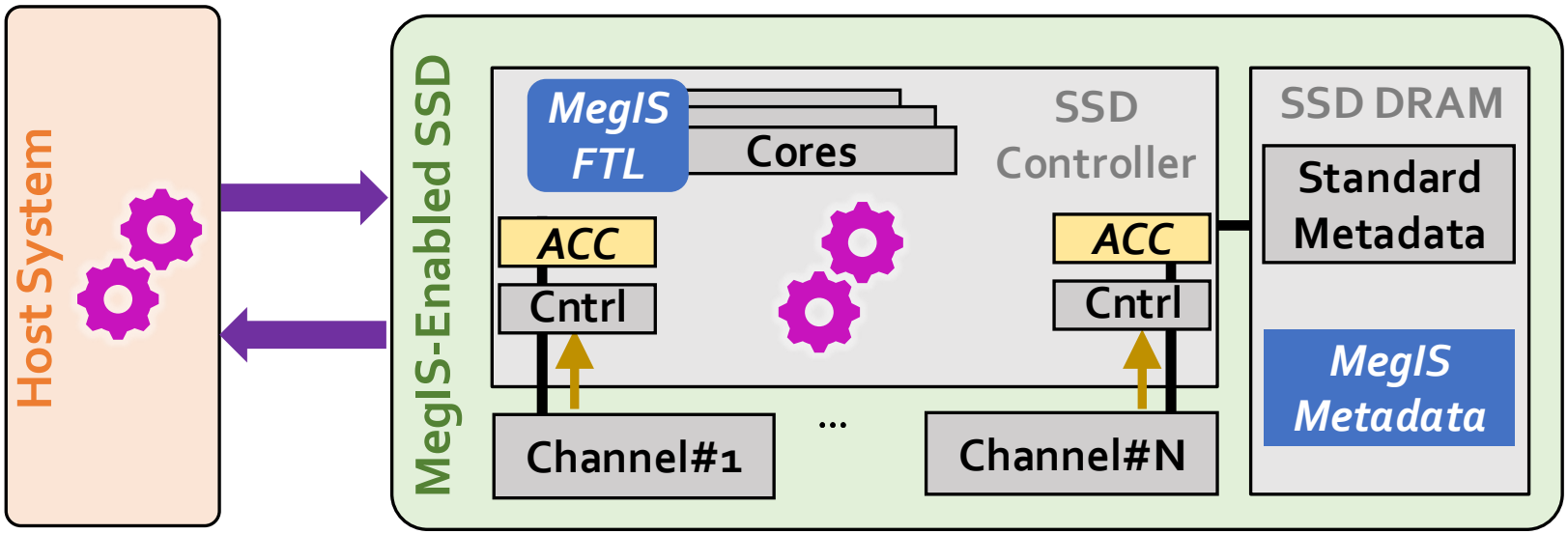
Storage-aware algorithms
- *Enable efficient access patterns to the SSD*

Lightweight in-storage accelerators
- *Minimize SRAM/DRAM buffer spaces needed inside the SSD*

MegIS Algorithm-Architecture Co-Design

Task partitioning and mapping
- *Each step executes in its most suitable system*

Data/computation flow coordination
- *Reduce communication overhead*
- *Reduce #writes to flash chips*



Storage-aware algorithms
- *Enable efficient access patterns to the SSD*

Lightweight in-storage accelerators
- *Minimize SRAM/DRAM buffer spaces needed inside the SSD*

Data mapping scheme and Flash Translation Layer (FTL)
- *Specialize to the characteristics of metagenomic analysis*
- *Leverage the SSD's full internal bandwidth*

Evaluation Methodology Overview (I)

Performance, Energy, and Power Analysis

Hardware Components

- Synthesized Verilog model for the in-storage accelerators
- MQSim *[Tavakkol+, FAST'18]* for SSD's internal operations
- Ramulator *[Kim+, CAL'15]* for SSD's internal DRAM

Software Components

Measure on a real system:

- AMD® EPYC® host CPU with 128 physical cores
- 1 TB host DRAM

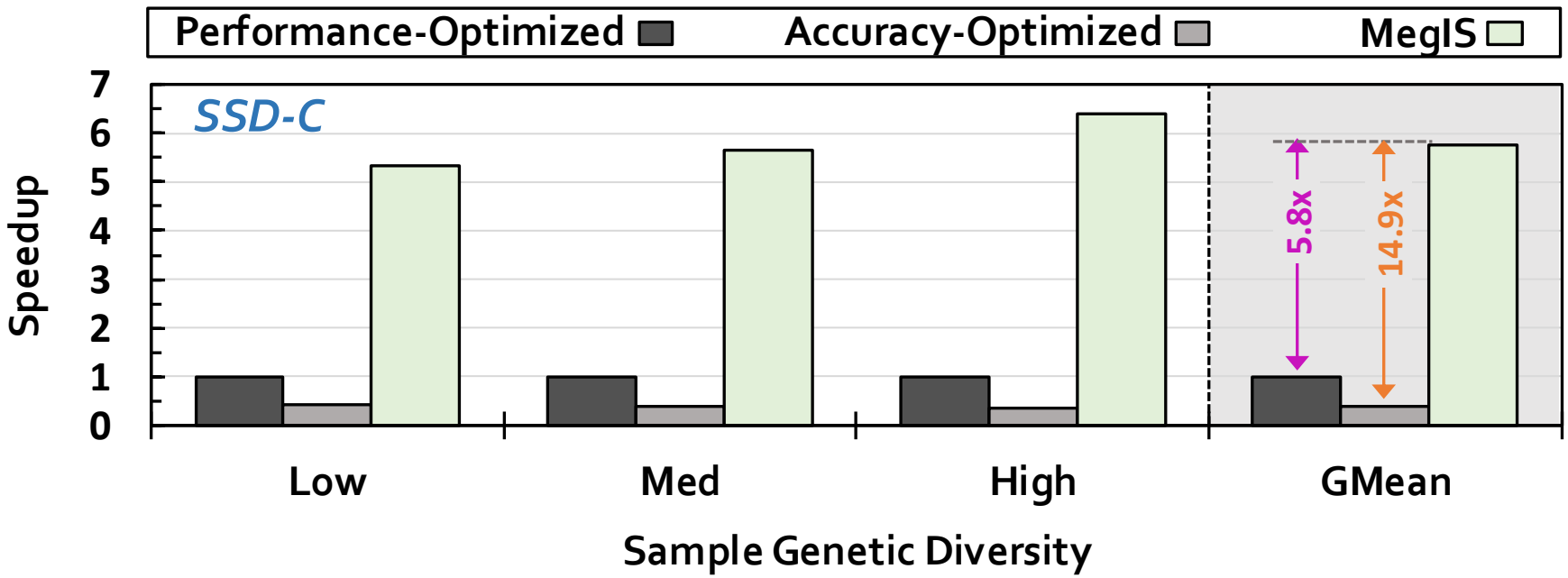
Baseline Comparison Points

- **Performance-optimized software**, Kraken2 *[Genome Biology'19]*
- **Accuracy-optimized software**, Metalign *[Genome Biology'20]*
- **PIM hardware-accelerated tool** (using processing-in-memory), Sieve *[ISCA'21]*

SSD Configurations

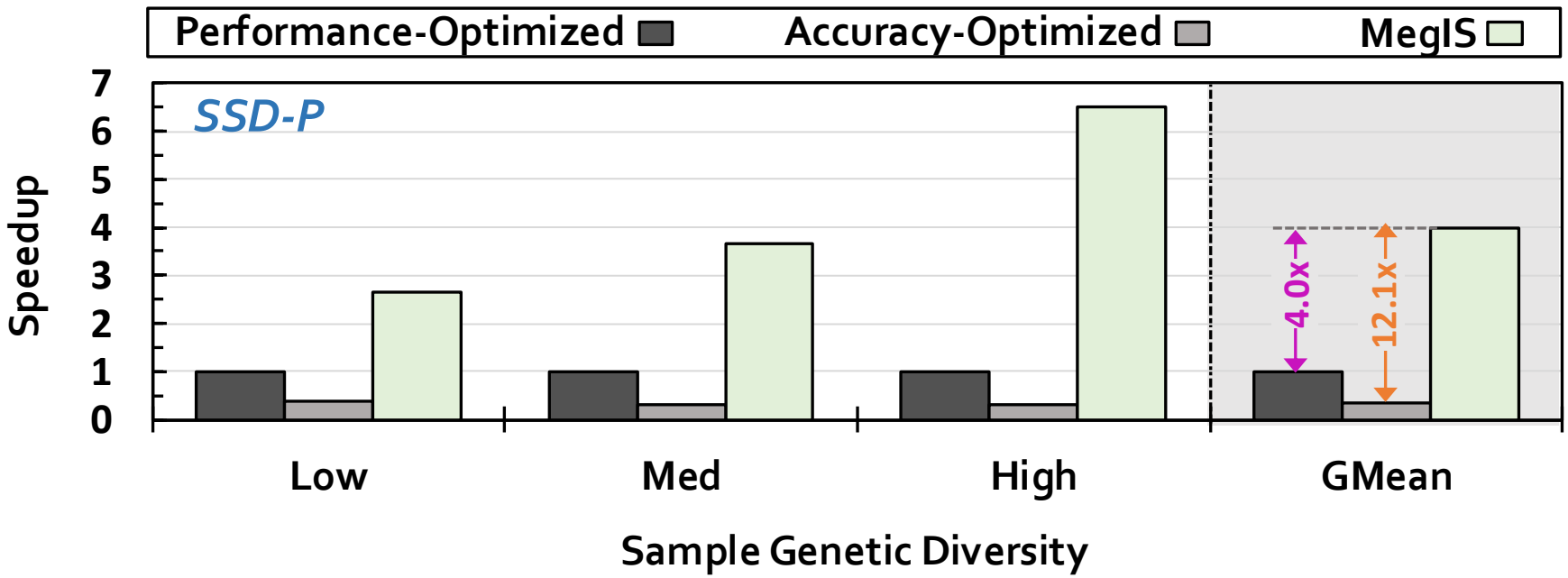
- **SSD-C:** With SATA3 interface (0.5 GB/s sequential read bandwidth)
- **SSD-P:** With PCIe Gen4 interface (7 GB/s sequential read bandwidth)

Speedup over Software (with Cost-Optimized SSD)



MegIS provides significant speedup over both Performance-Optimized and Accuracy-Optimized baselines

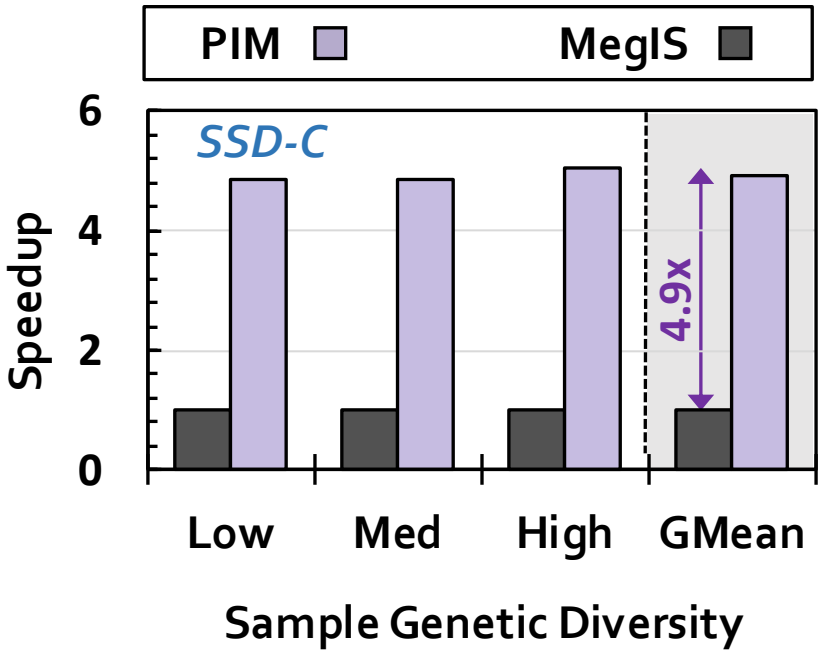
Speedup over Software (with Performance-Optimized SSD)

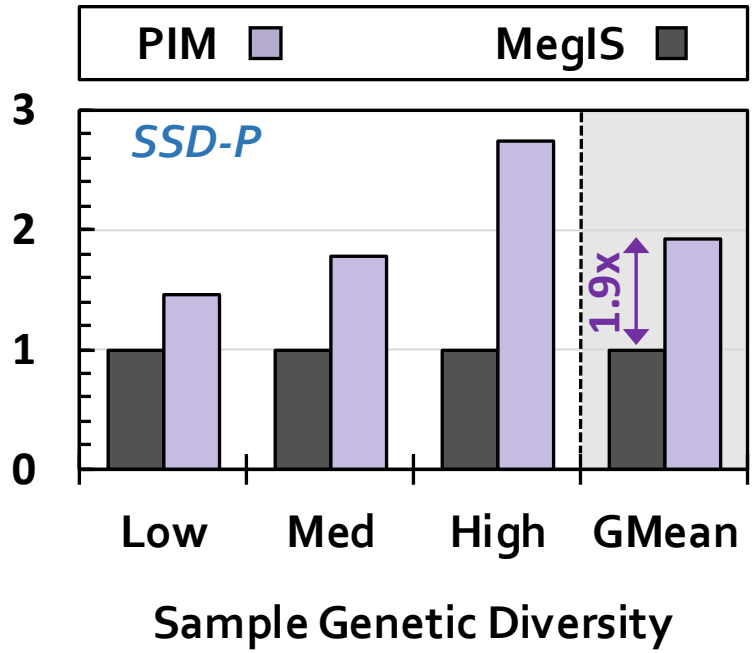


MegIS provides significant speedup over both Performance-Optimized and Accuracy-Optimized baselines

MegIS improves performance on both cost-optimized and performance-optimized SSDs

Speedup over the PIM Hardware Baseline

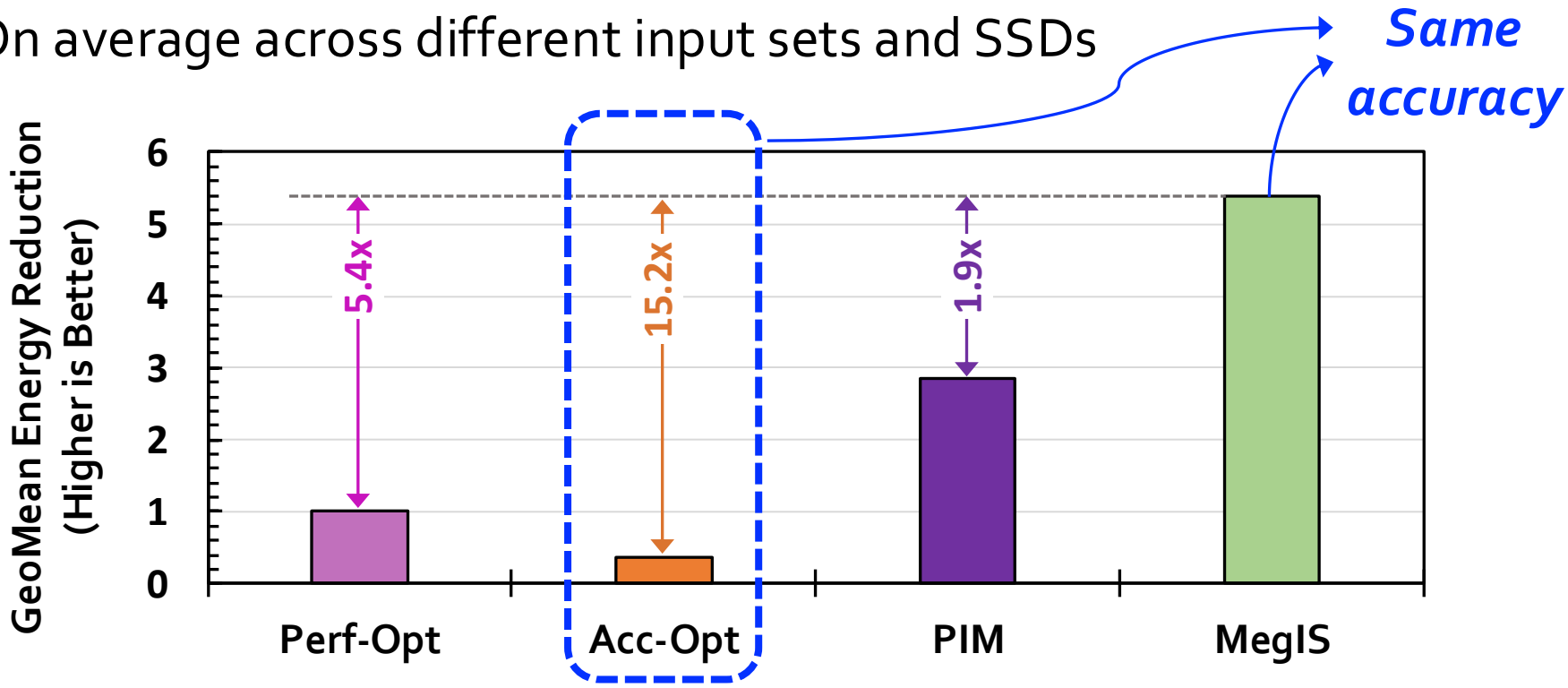




MegIS provides significant speedup over the PIM baseline

Reduction in Energy Consumption

- On average across different input sets and SSDs



MegIS provides significant energy reduction over the Performance-Optimized, Accuracy-Optimized, and PIM baselines

Accuracy, Area, and Power

Accuracy

- **Same accuracy** as the **accuracy-optimized** baseline
- **Significantly higher accuracy** than the **performance-optimized** and **PIM** baselines
  - 4.6 – 5.2× higher F1 score
  - 3 – 24% lower L1 norm error

Area and Power

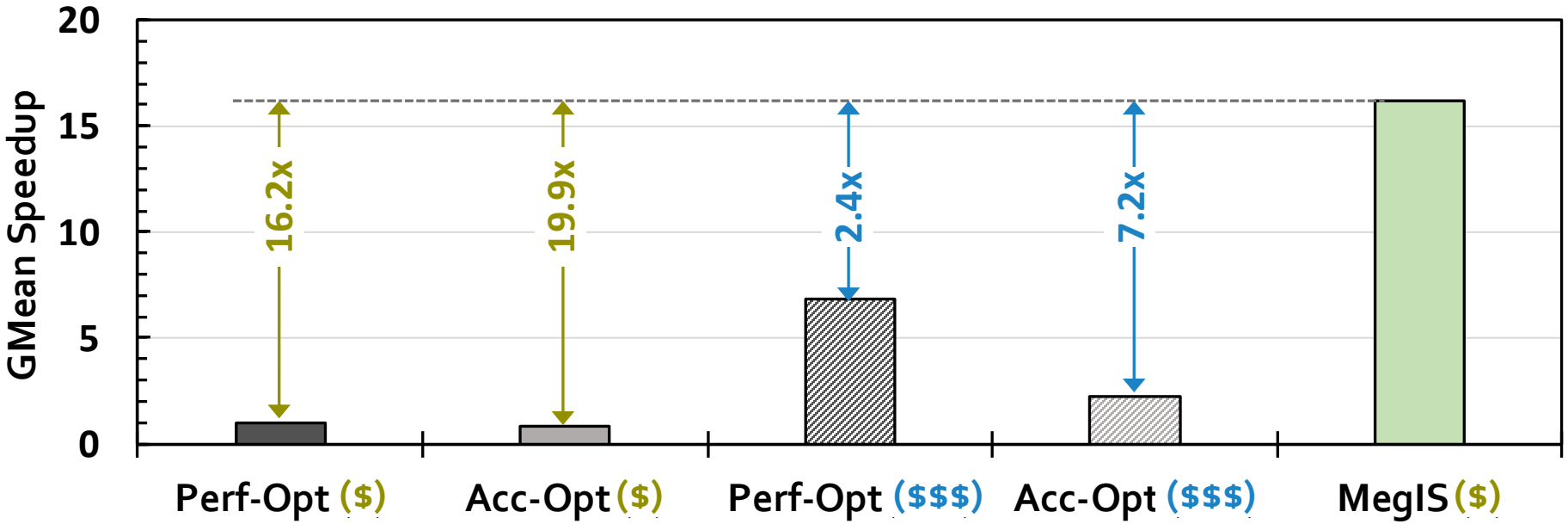
Total for an 8-channel SSD:

- **Area:** 0.04 mm^2
- **Power:** 7.658 mW

*(Only **1.7%** of the area and **4.6%** of the power consumption of three ARM Cortex R4 cores in an SSD controller)*

System Cost-Efficiency

- **Cost-optimized system ($):** With SSD-C and 64-GB DRAM
- **Performance-optimized system ($$$):** With SSD-P and 1-TB DRAM



MegIS outperforms the baselines

***even* when running on a much less costly system**

Our Approach

*Designing customized **storage-centric systems** that efficiently*

① ***analyze** genomic and metagenomic data **inside the storage system***

1 GenStore [ASPLOS'22]
In-storage filtering
for genomic analysis

2 MegIS [ISCA'24]
Cooperative in-storage processing
for metagenomic analysis

3 GRAINS [ISCA'26]
Storage-aware algorithm-architecture co-design
for graph-based (meta)genomic analyses

② *enable **highly-compressed storage** and **high-performance access** of large-scale sequence data*

4 SAGe [HPCA'26]
Algorithm-architecture co-design
for storing and accessing sequence data

Our Approach

*Designing customized **storage-centric systems** that efficiently*

① ***analyze** genomic and metagenomic data **inside the storage system***

1 **GenStore** [ASPLOS'22]
In-storage filtering
for genomic analysis

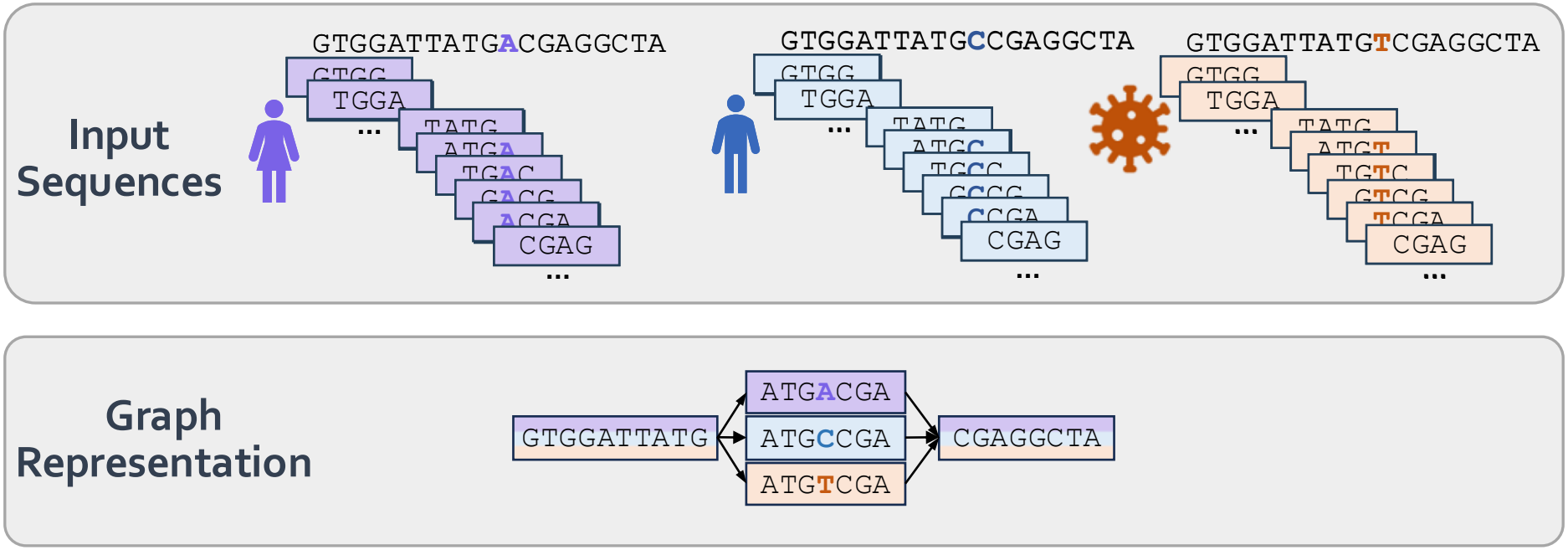
2 **MegIS** [ISCA'24]
Cooperative in-storage processing
for metagenomic analysis

3 **GRAINS** [ISCA'26]
Storage-aware algorithm-architecture co-design
for graph-based (meta)genomic analyses

② *enable **highly-compressed storage** and **high-performance access** of large-scale sequence data*

4 **SAGe** [HPCA'26]
Algorithm-architecture co-design
for storing and accessing sequence data

Large-Scale Genome Graphs



Powerful and efficient representations of genomic data

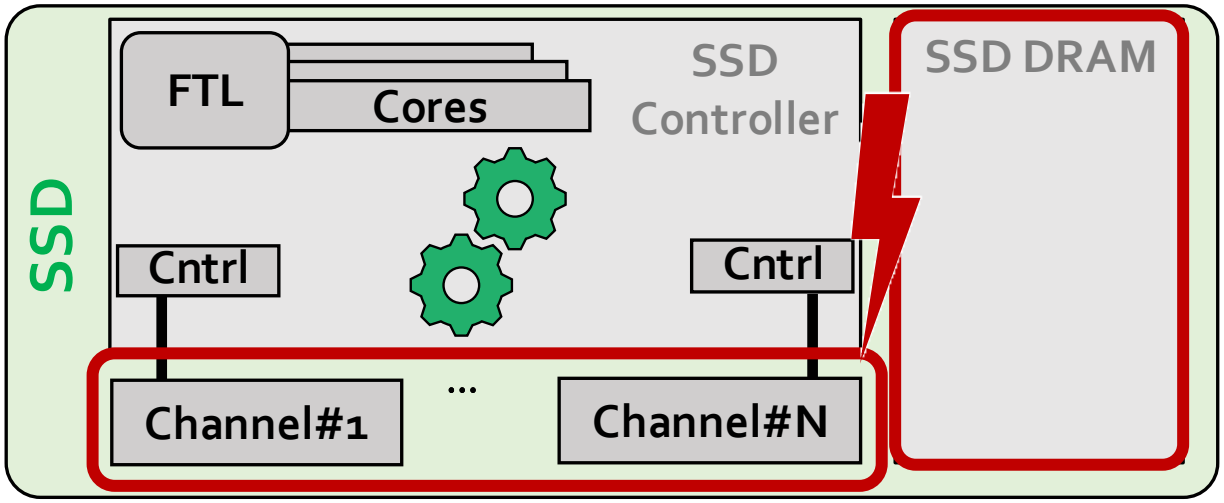
✓ **Great expressive power**

✓ **Compact representations**

Critical for enabling massive, population-scale analyses

Challenges of Storage-Centric Computing

Graph-based sequence analysis tools cannot run in storage due to SSD limits



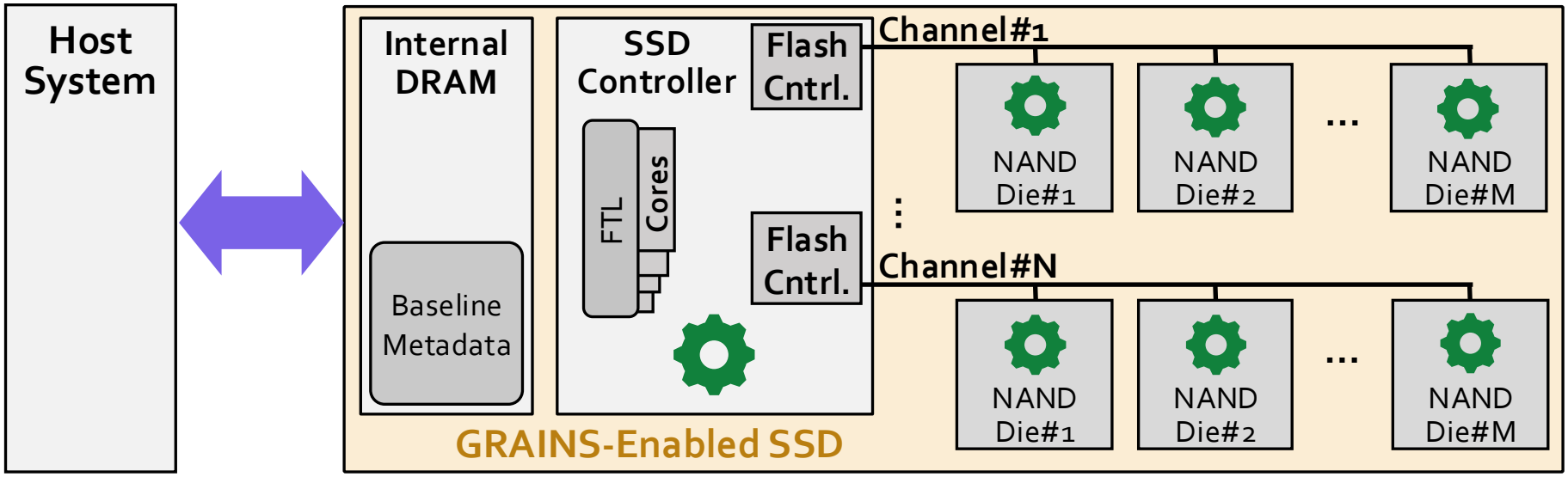
New challenges due to:

Dependent and random accesses to the sequence graph

Different properties from conventional, non-genome graphs

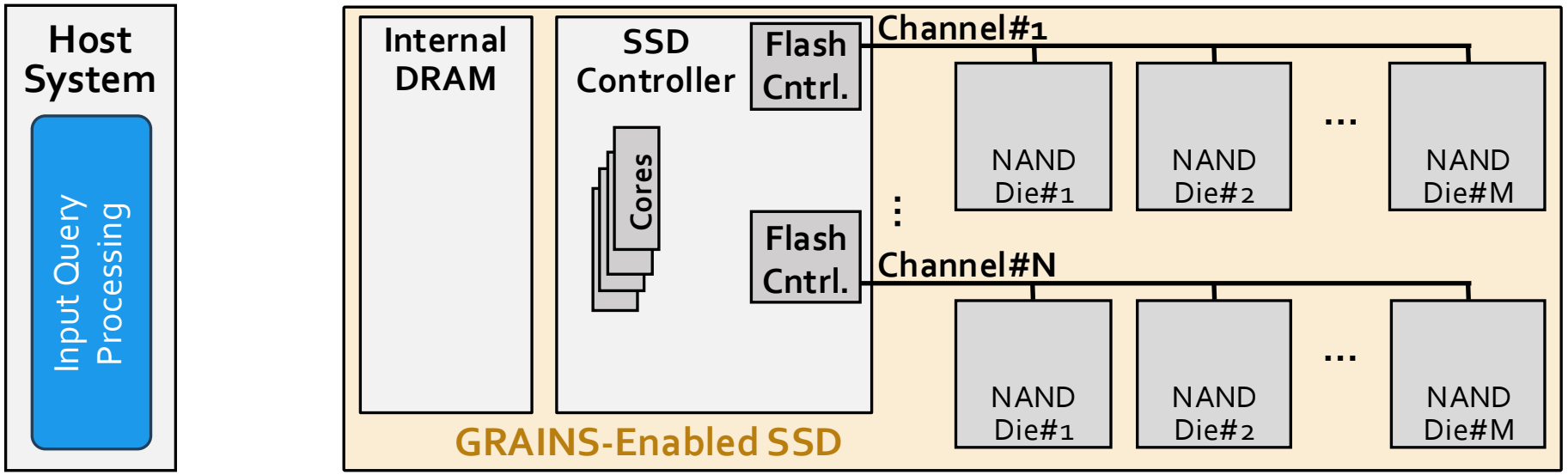
GRAINS

- The *first* system for analysis with large-scale genome graphs in storage
- Enabled via **storage-aware algorithm-architecture co-design**, based on the **properties of large-scale genome graphs** to
  - Make the execution pipeline **more storage-friendly**
  - Further improve **performance**, **energy-efficiency**, and **cost-effectiveness** via **storage-centric computing**



GRAINS's Algorithm-Architecture Co-Design

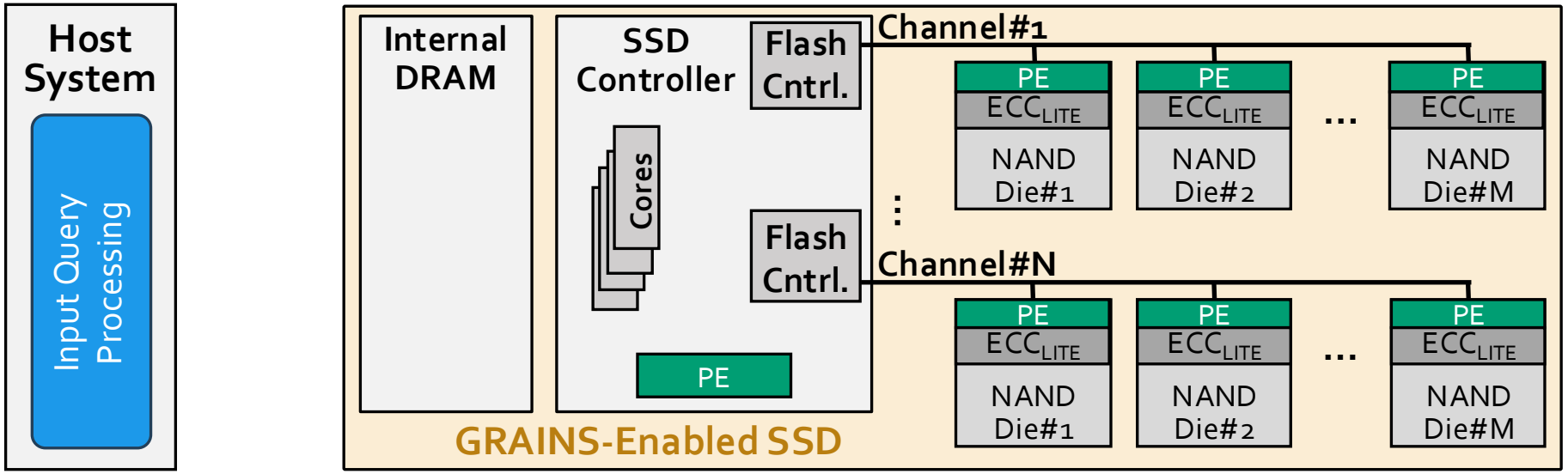
New batching technique and execution flow
based on the unique features of genome graphs that reduces #random accesses



GRAINS's Algorithm-Architecture Co-Design

New batching technique and execution flow
based on the unique features of genome graphs that reduces #random accesses

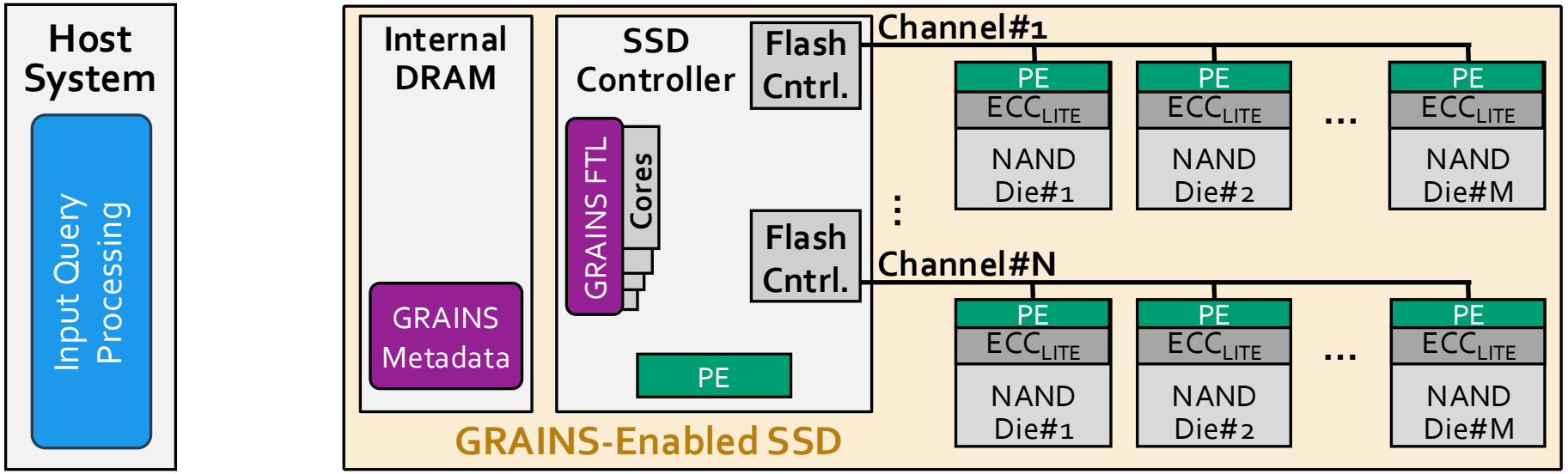
In-storage and In-flash processing
to avoid transferring low-reused pages and bandwidth waste



GRAINS's Algorithm-Architecture Co-Design

New batching technique and execution flow
based on the unique features of genome graphs that reduces #random accesses

In-storage and In-flash processing
to avoid transferring low-reused pages and bandwidth waste

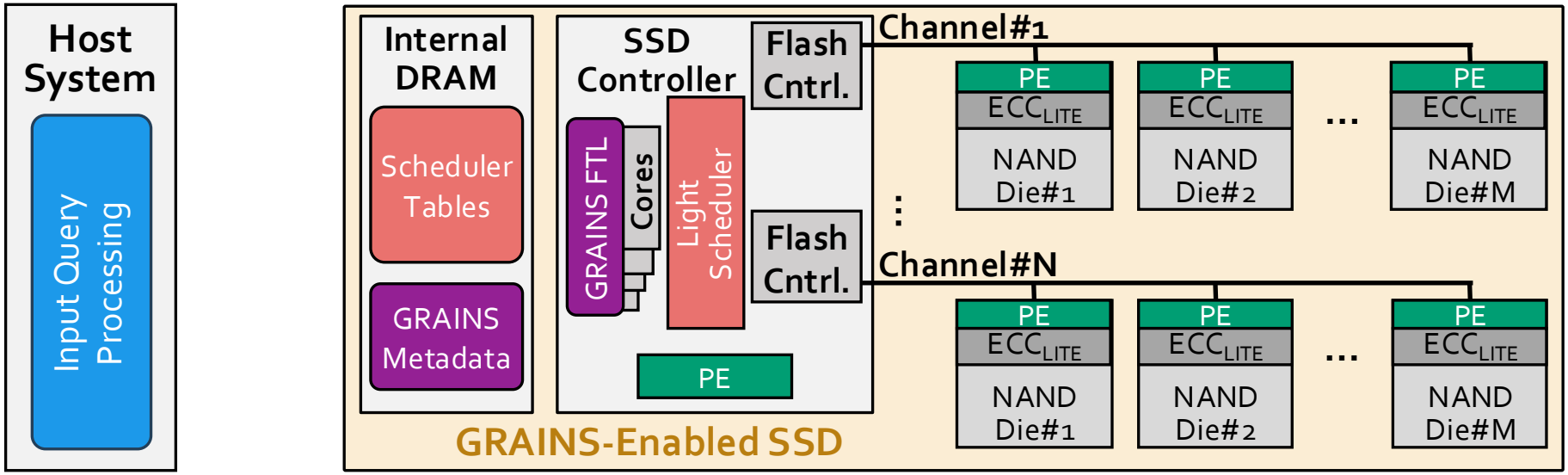


FTL and data placement
based on genome graph properties, to leverage SSD's full internal bandwidth

GRAINS's Algorithm-Architecture Co-Design

New batching technique and execution flow
based on the unique features of genome graphs that reduces #random accesses

In-storage and In-flash processing
to avoid transferring low-reused pages and bandwidth waste



FTL and data placement
based on genome graph properties, to leverage SSD's full internal bandwidth

Lightweight scheduling technique
enabled by repurposing the existing SSD structures to leverage the full parallelism of all flash dies

Evaluation Methodology Overview (I)

Performance, Energy, and Power Analysis

Hardware Components

- Synthesized Verilog model for the in-storage accelerators
- MQSim *[Tavakkol+, FAST'18]* for SSD's internal operations
- Ramulator *[Kim+, CAL'15]* for SSD's internal DRAM

Software Components

Measure on a real system:

- AMD® EPYC® host CPU with 128 physical cores
- 1.5 TB host DRAM

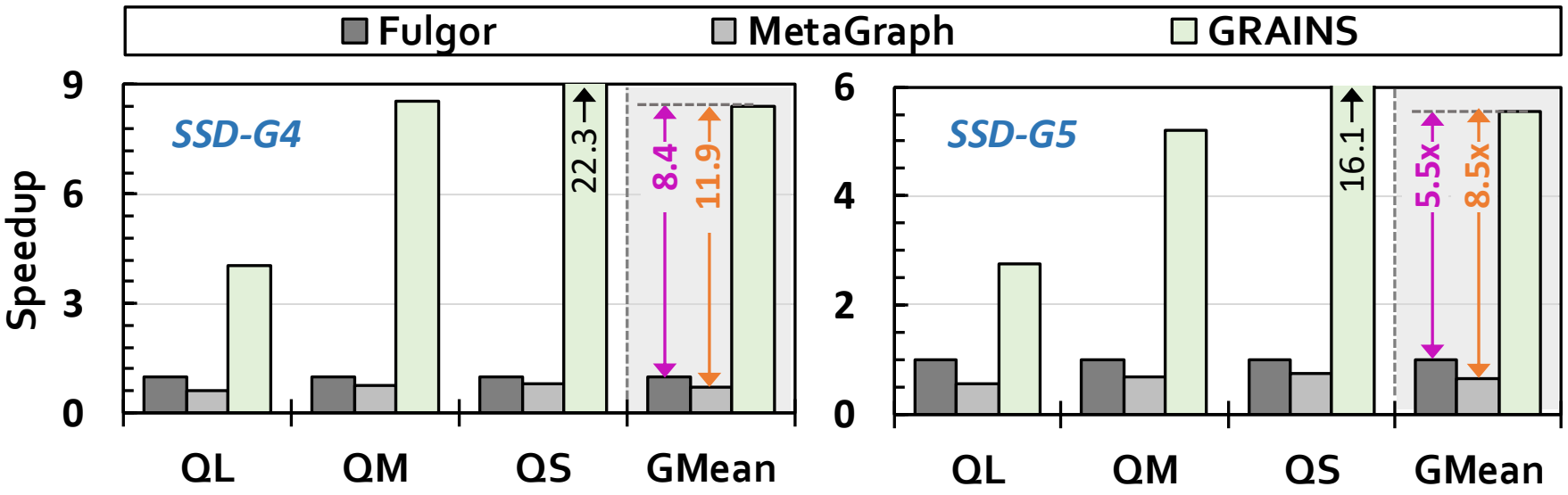
Baseline Comparison Points

- **Fulgor** *[WABI'23]*: A state-of-the-art node-centric tool
- **MetaGraph** *[Nature'25]*: A state-of-the-art edge-centric tool
- **AccIdealMem**: An idealized accelerator for graph queries with no main memory overheads, but still suffers from the I/O overhead of moving a large amount of low-reuse data

SSD Configurations

- **SSD-G4:** With PCIe Gen4 interface (7 GB/s sequential read bandwidth)
- **SSD-G5:** With PCIe Gen5 interface (14.8 GB/s sequential read bandwidth)

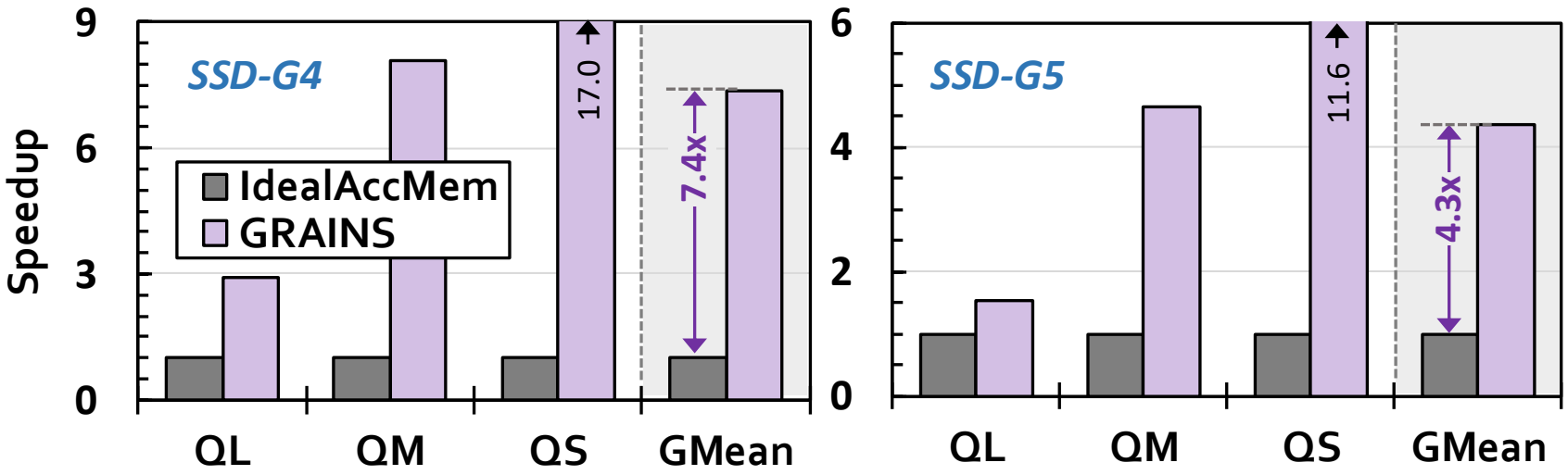
Benefits over Software



GRAINS provides significant speedup over Fulgor and Metagraph baselines, with both SSD-G4 and SSD-G5

GRAINS improves average energy-efficiency by 11.3x and 21.5x over Fulgor and Metagraph, respectively

Benefits over Hardware



GRAINS provides significant speedups over AccIdealMem baseline

GRAINS improves average energy-efficiency by 3.1—20.7x over AccIdealMem

Area and Power

- Based on **Synthesis** of **GRAINS's hardware units** using the Synopsys Design Compiler @ 22nm technology node

- ECC_{LITE} area based on the original paper *[AiF, ISCA'25]*

Logic unit	Area [mm²]	Power [mW]
On SSD Controller	0.0025	0.21
On Die (ECC_LITE)	0.036	18.04
On Die (Others)	0.000093	0.01

On-controller logic units are 0.7% of the four cores on an SSD controller

On-die logic units take only 0.2% of the flash die's area and fit well within its power budget

Our Approach

*Designing customized **storage-centric systems** that efficiently*

① ***analyze** genomic and metagenomic data **inside the storage system***

1 GenStore [ASPLOS'22]
In-storage filtering
for genomic analysis

2 MegIS [ISCA'24]
Cooperative in-storage processing
for metagenomic analysis

3 GRAINS [ISCA'26]
Storage-aware algorithm-architecture co-design
for graph-based (meta)genomic analyses

② *enable **highly-compressed storage** and **high-performance access** of large-scale sequence data*

4 SAGe [HPCA'26]
Algorithm-architecture co-design
for storing and accessing sequence data

Our Approach

*Designing customized **storage-centric systems** that efficiently*

① ***analyze** genomic and metagenomic data **inside the storage system***

1 **GenStore** [ASPLOS'22]
In-storage filtering
for genomic analysis

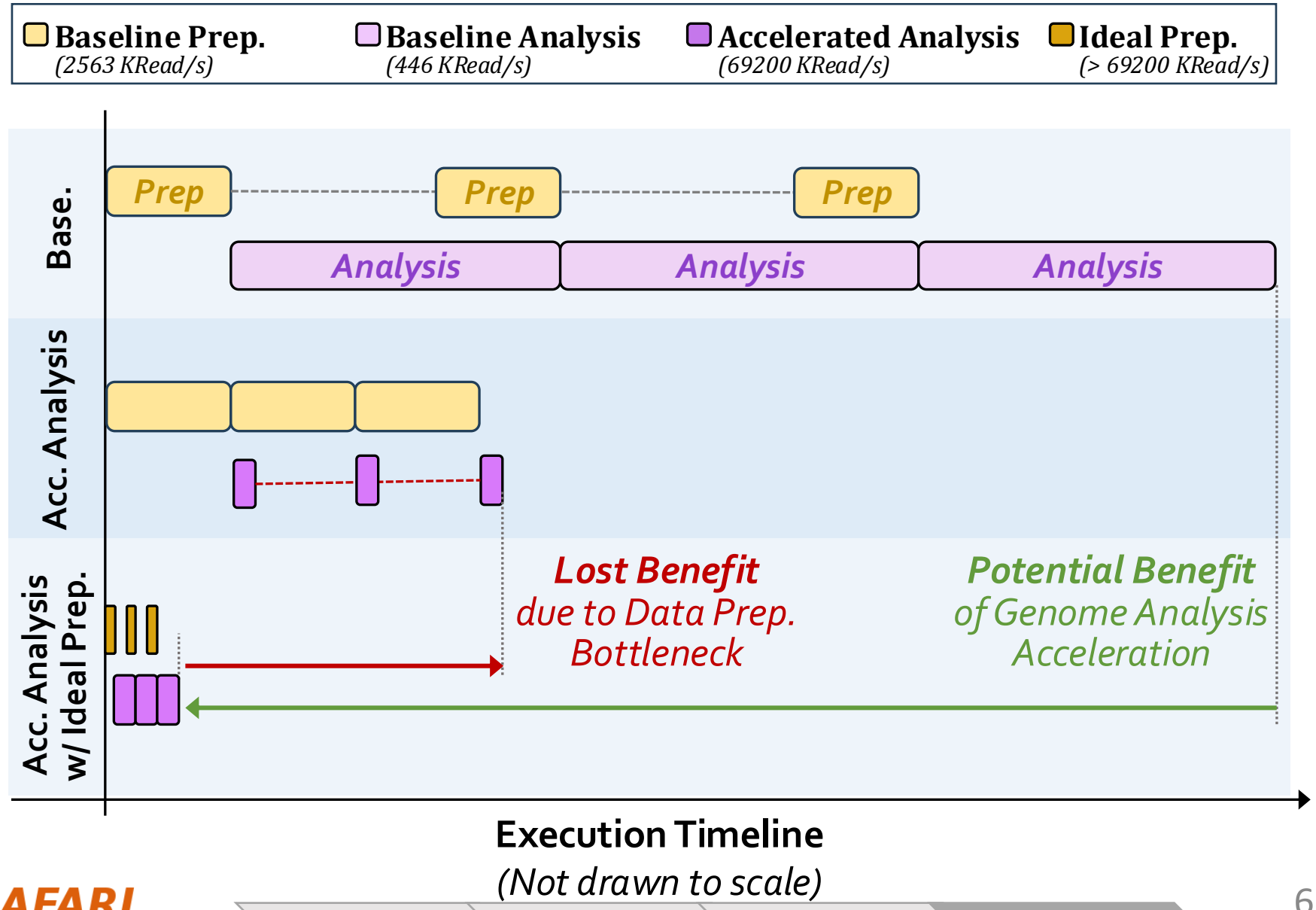
2 **MegIS** [ISCA'24]
Cooperative in-storage processing
for metagenomic analysis

3 **GRAINS** [ISCA'26]
Storage-aware algorithm-architecture co-design
for graph-based (meta)genomic analyses

② *enable **highly-compressed storage** and **high-performance access** of large-scale sequence data*

4 **SAGe** [HPCA'26]
Algorithm-architecture co-design
for storing and accessing sequence data

Recall: Effect of Data Preparation Bottleneck



Mitigating Data Preparation Bottleneck

We need to effectively mitigate the data preparation bottleneck while meeting four key requirements:

High performance

High energy efficiency

High compression ratio

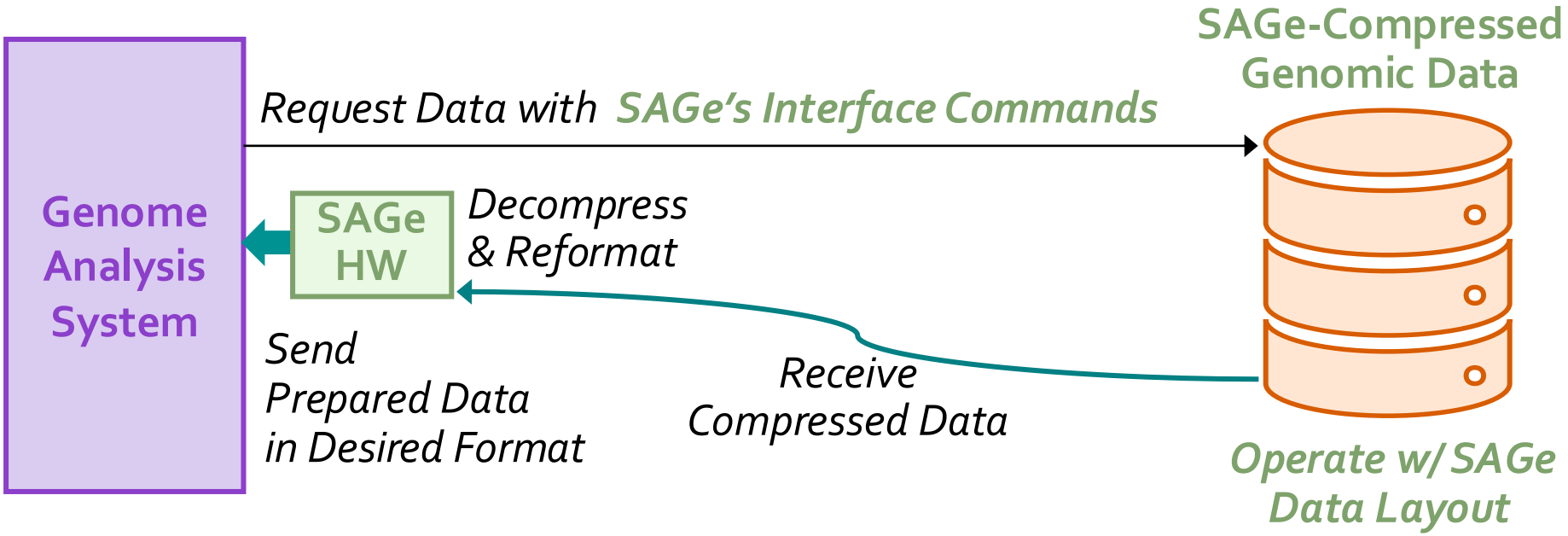
Low overhead

Meeting all these requirements is challenging

Even more challenging in genome analysis systems used in hardware resource-constrained environments

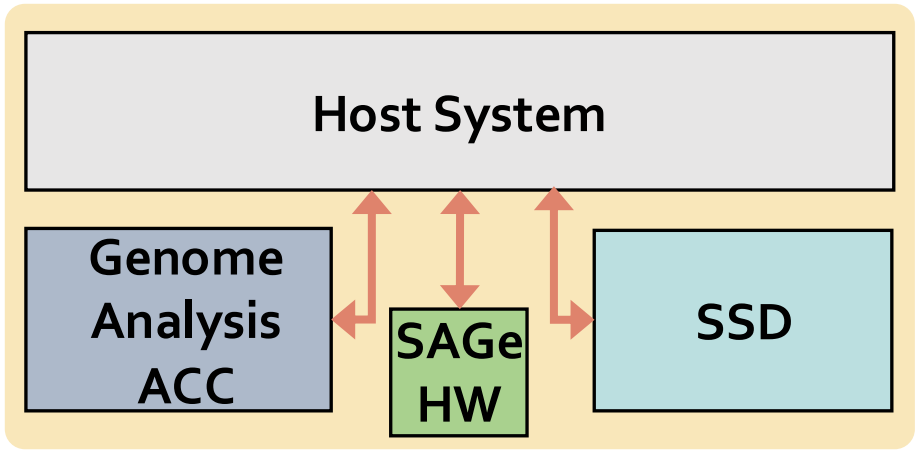
*An algorithm-architecture co-design for highly-compressed **s**torage and high-performance **a**ccess of large-scale **ge**nomic sequence data*

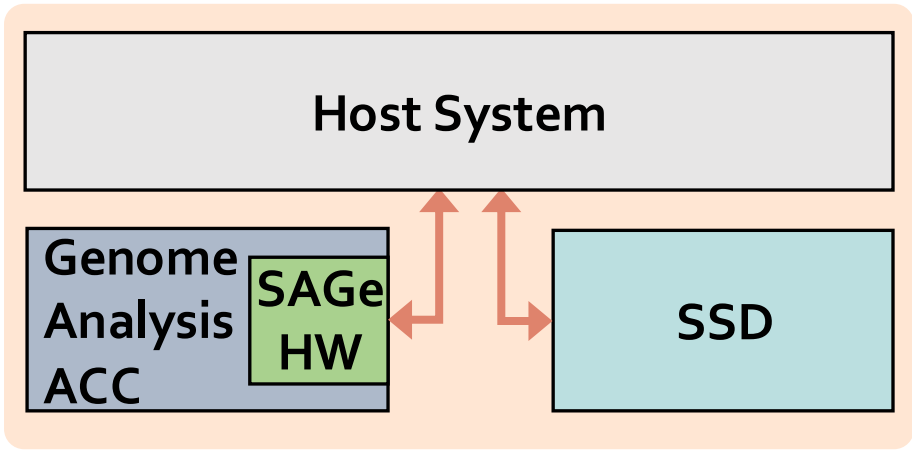
- **Key insight**: Information encoded in genomic data follows specific properties
  We leverage these properties in our algorithm-architecture co-design

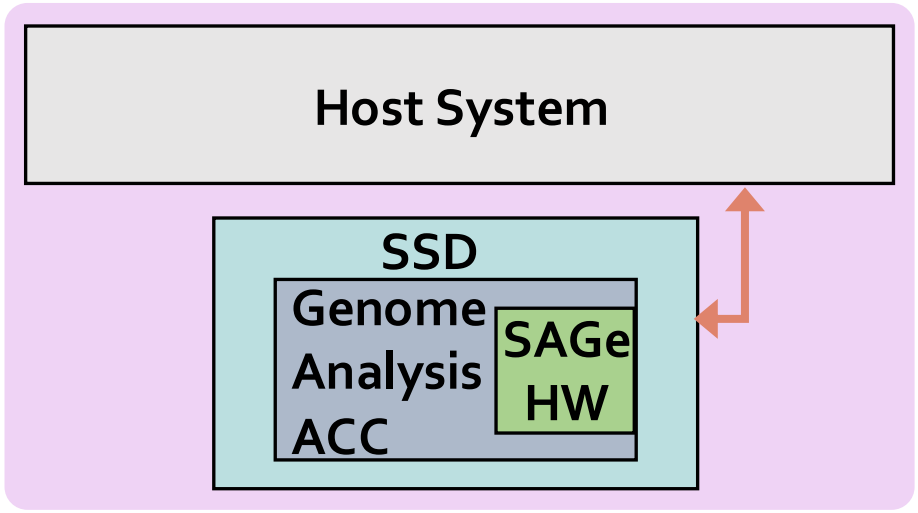


Case Studies of SAGe's Hardware Integration

Due to its lightweight design, SAGe can seamlessly integrate with genome analysis systems







Evaluation Methodology Overview (I)

Performance, Energy, and Power Analysis

Hardware Components

- Synthesized Verilog model for the hardware units
- MQSim *[Tavakkol+, FAST'18]* for SSD's internal operations
- Ramulator *[Kim+, CAL'15]* for SSD's internal DRAM

Software Components

Measure on a real system:

- AMD® EPYC® host CPU with 128 physical cores
- 1.5 TB host DRAM

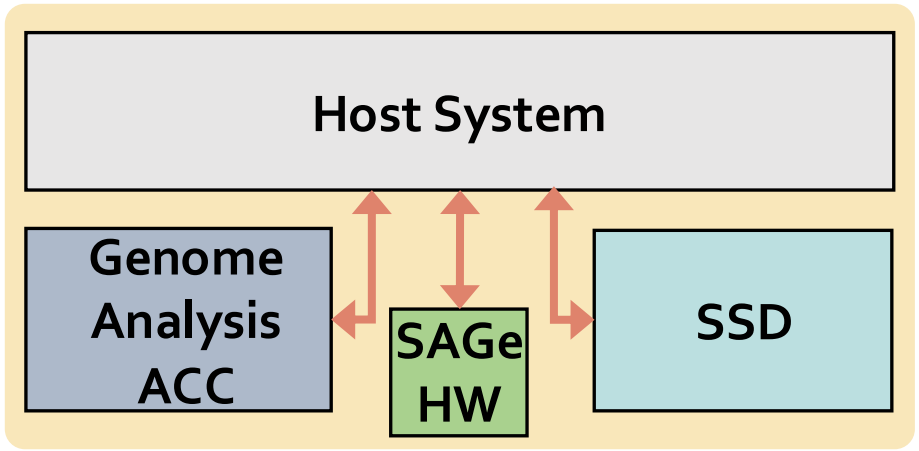
Datasets: Read sets (RS)

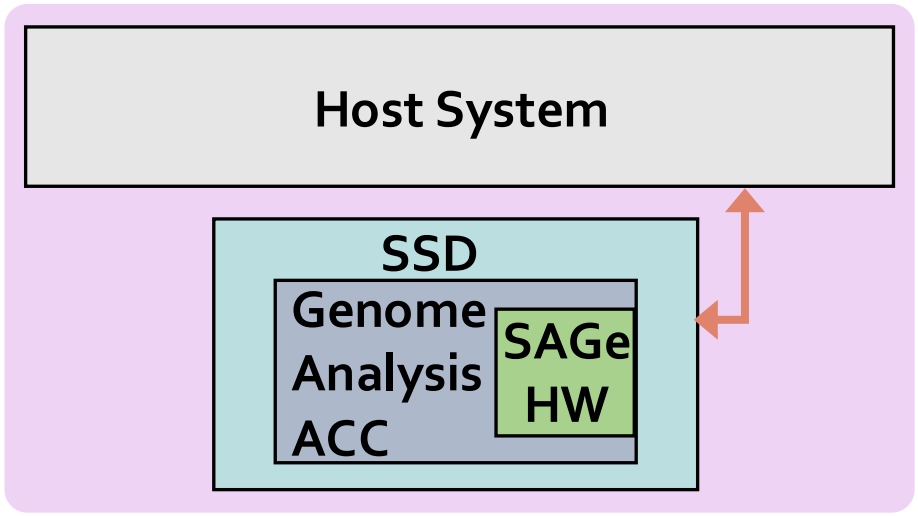
- From a variety of species (i.e., *H. sapiens*, *T. cacao*, *M. acuminata malaccensis*)
- Sequenced with different sequencing technologies (i.e., Illumina and Oxford Nanopore Technologies)

End-to-end performance of a genome analysis application, which includes both **data preparation** and **genome analysis**

Recall: Case Studies of SAGe's Hardware Integration

Due to its lightweight design, SAGe can seamlessly integrate with genome analysis systems



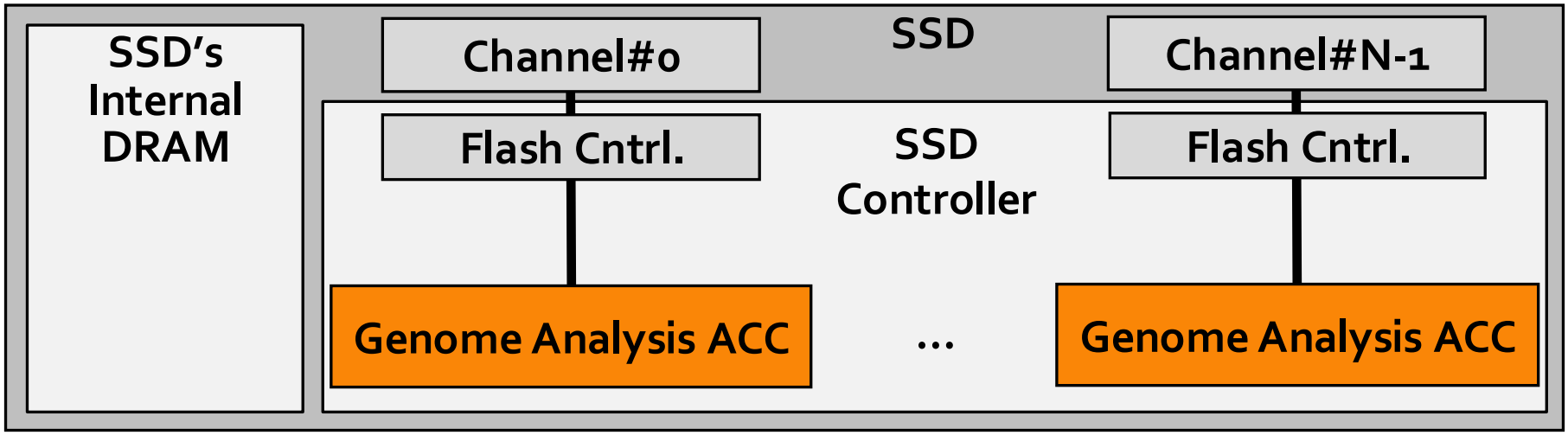


Evaluation Methodology Overview (II)

Genome Analysis

- A state-of-the-art hardware accelerator (GEM *[Chen+, TPDS'23]*), for read mapping

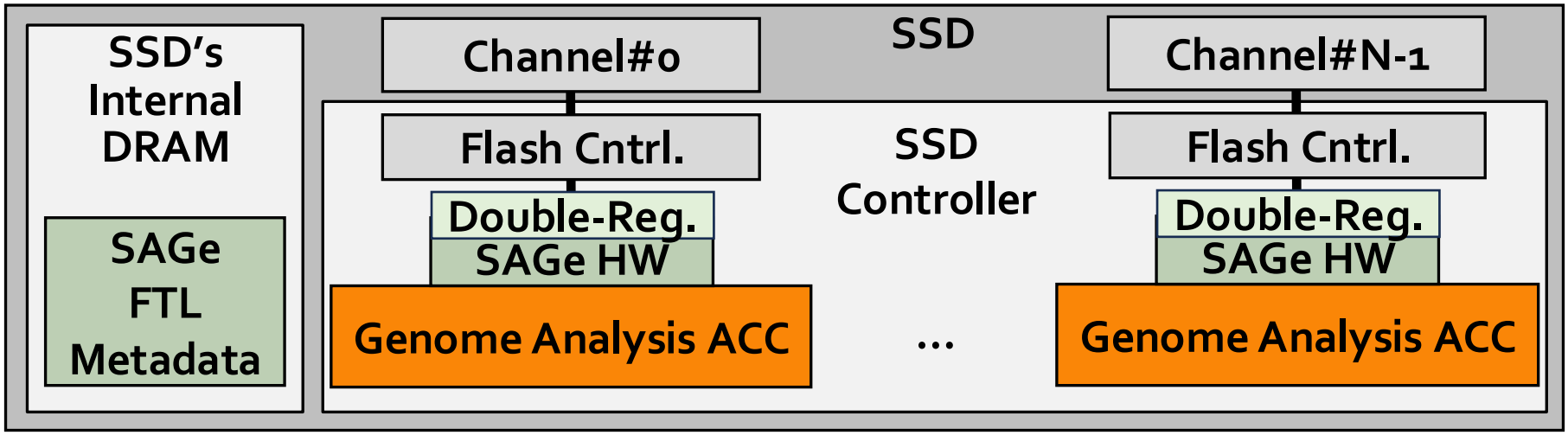
- Near-data processing (NDP) baseline: GenStore *[Mansouri Ghiasi+, ASPLOS'22]*
  - Alleviates the burden of moving and analyzing a large amount of low-reuse data
  - Implemented in the resource-constrained environment of the SSD



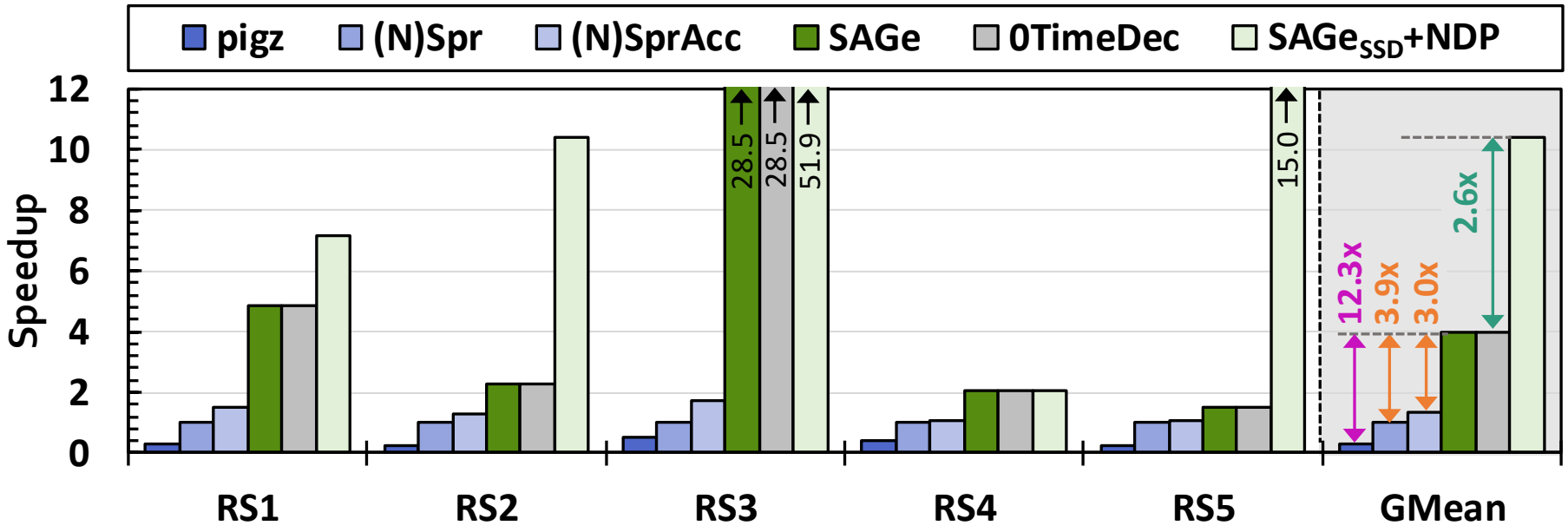
Evaluation Methodology Overview (II)

Data Preparation

- **pigz:** Commonly-used general-purpose software (*[Adler, JPL'15]*)
- **(N)Spr:** Genomics-specific software for short (Spring *[Chandak, Bioinformatics'18]*) and long reads (NanoSpring *[Meng+, Scientific Reports'23]*)
- **(N)SprAcc:** (N)Spring hardware-accelerated
- **0TimeDec:** An idealized decompressor (with *zero* decompression time), but inefficient for integration in resource-constrained environments
- **SAGe:** SAGe implemented as a standalone accelerator for data preparation
- **SAGe$_{SSD}$:** SAGe inside the SSD, for integration with the NDP analysis system



Speedup



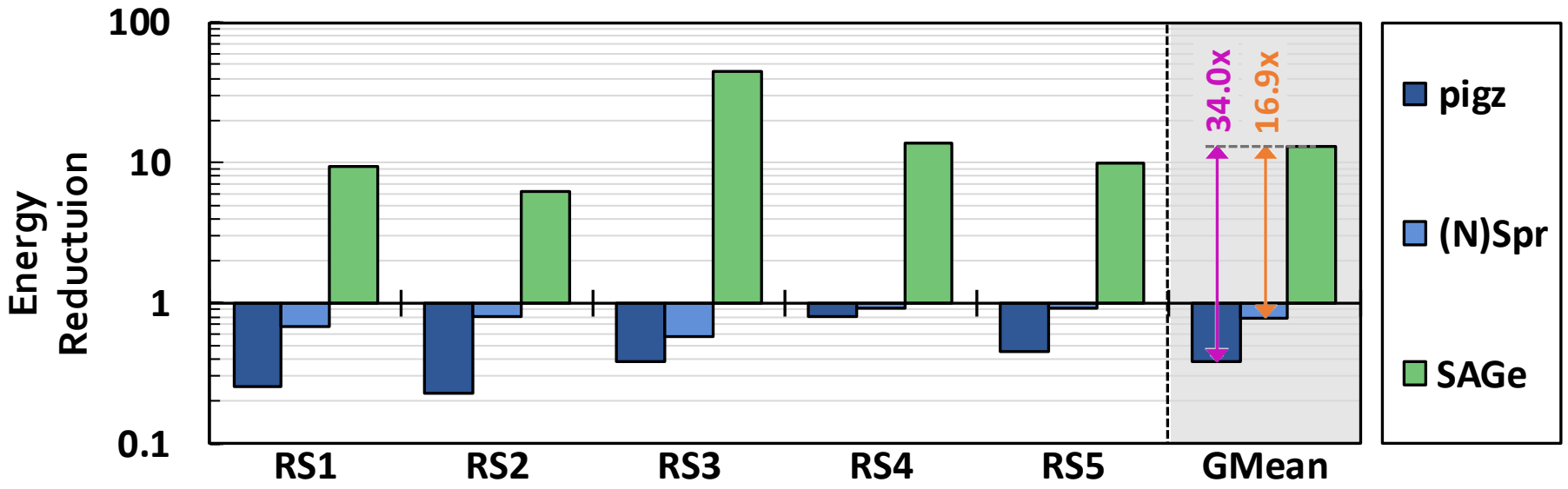
SAGe provides significant speedup over both general-purpose and genomics-specific baselines

SAGe achieves the same speedup as 0TimeDec

By efficiently integrating SAGe$_{SSD}$ with NDP SAGe$_{SSD}$+NDP leads to large speedups

Reduction in Energy Consumption

Normalized to (N)SprAcc (higher is better)



SAGe provides significant energy reduction over both general-purpose and genomics-specific baselines

Compression Ratio

2.9× better average compression ratio than general-purpose baseline

Comparable compression ratio to genomics-specific baseline
(with a modest 4.6% average reduction)

Area and Power

- Based on **Synthesis** of **SAGe's hardware units** using the Synopsys Design Compiler @ 22nm technology node

Logic unit	# of instances	Area [mm²]	Power [mW]
Comparator	1 per channel	0.000045	0.014
Read Construction Unit	1 per channel	0.000017	0.023
Double Registers	1 per channel	0.00020	0.035
Control Unit	1 per channel	0.000029	0.025
Total for an 8-channel SSD	-	0.002	0.77

SAGe's hardware units consume small area and power

Our Approach

*Designing customized **storage-centric systems** that efficiently*

① ***analyze** genomic and metagenomic data **inside the storage system***

1 GenStore [ASPLOS'22]
In-storage filtering
for genomic analysis

2 MegIS [ISCA'24]
Cooperative in-storage processing
for metagenomic analysis

3 GRAINS [ISCA'26]
Storage-aware algorithm-architecture co-design
for graph-based (meta)genomic analyses

② *enable **highly-compressed storage** and **high-performance access** of large-scale sequence data*

4 SAGe [HPCA'26]
Algorithm-architecture co-design
for storing and accessing sequence data

Conclusion

*By designing customized **storage-centric systems** that efficiently*

① ***analyze** genomic and metagenomic data **inside the storage system***

1 GenStore 2 MegIS 3 GRAINS

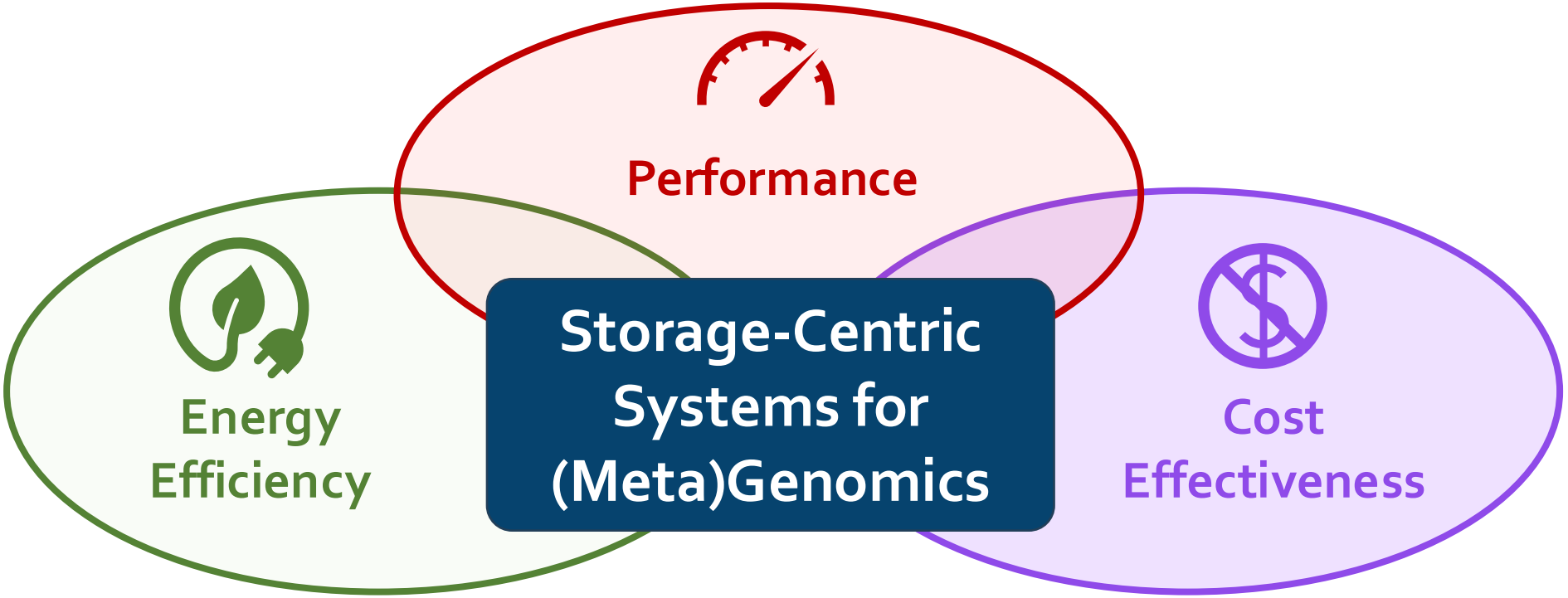
② *enable **highly-compressed storage** and **high-performance access** of large-scale sequence data*

4 SAGe

we can alleviate the overheads of

data movement *computation* *data preparation*

Putting It All Together



Make accurate (meta)genomic analysis more accessible for wider adoption

Future Directions

1 Advancing the capabilities of storage-centric computing for health and life sciences

- Extending the capabilities of the proposed designs for (meta)genomics
- Expanding to new areas: Transcriptomics, single-cell analysis
- Updating large-scale databases

2 Highly compressed storage and fast access of other biological data

- Raw genomic signals
- Transcriptomics, single-cell data

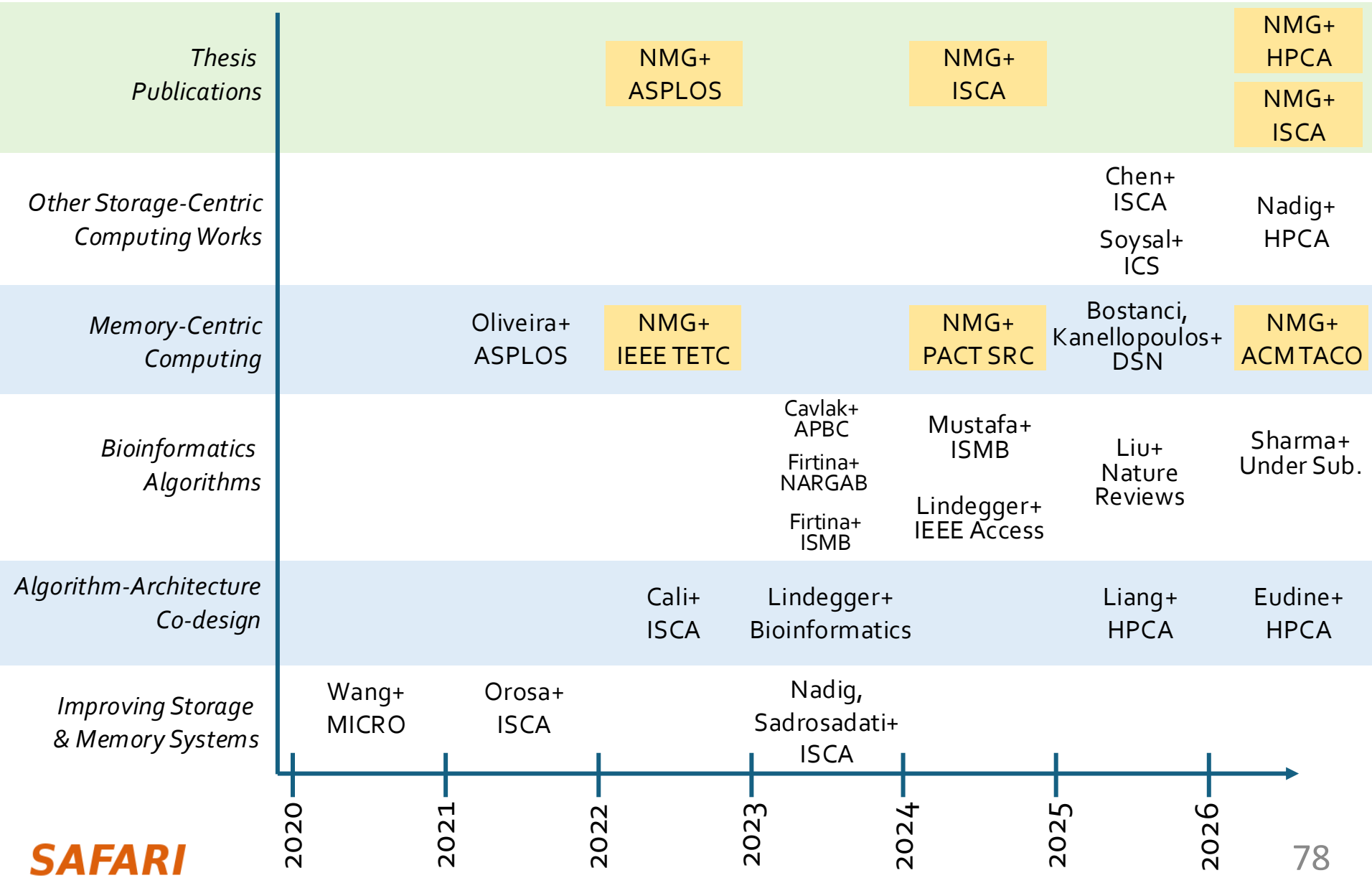
3 Alleviating the adoption challenges of storage-centric designs

- Privacy
- Applicability vs. cost tradeoffs

4 Facilitating portable (meta)genomics with storage-centric designs

My Involvements During PhD

First author publications



Acknowledgments

- **Advisor:** Onur Mutlu
- **Committee members:** Can Alkan, Reetuparna Das, Wen-mei Hwu, Jangwoo Kim, Yatish Turakhia
- **Chair:** Morteza Aramesh
- **Mentors:** Jisung Park, Mohammad Sadrosadati, Nandita Vijaykumar, Juan Gómez-Luna
- **Mentees:** Talu, Marc, Timur, Eirini, Akmuhammet
- **All SAFARI group members** (especially Konstantinos Sr., Rahul, Konstantina, Geraldo, Giray, Can, Damla, Rakesh, Jeremie, Nisa, Ataberk, Lois, Yaohua, Haiyu, Christina)
- **Industrial partners & sponsors**
- **Friends** (especially Mehrasa, Yasmin, Hashem, Julia, Vesna, Pascal)
- **Family:** Harun, Mehrangiz, Amir, Ali, Tuncel, Niyazi, Hazal, aunts, uncles, and cousins
  and in the loving memory of Khaleh Golshan and my grandmother, Narges

Storage-Centric System Designs for Enabling Fast, Efficient, and Low-Cost Genomic and Metagenomic Analyses

Nika Mansourighiasi

Doctoral Examination
31.03.2026

Advisor:
Onur Mutlu (ETH Zurich)

Co-Examiners:
Can Alkan (Bilkent University)
Reetuparna Das (University of Michigan)
Wen-mei Hwu (NVIDIA & University of Illinois Urbana-Champaign)
Jangwoo Kim (Seoul National University & MangoBoost)
Yatish Turakhia (University of California San Diego)

SAFARI

ETH zürich

Discussion

GenStore

MegIS

GRAINS

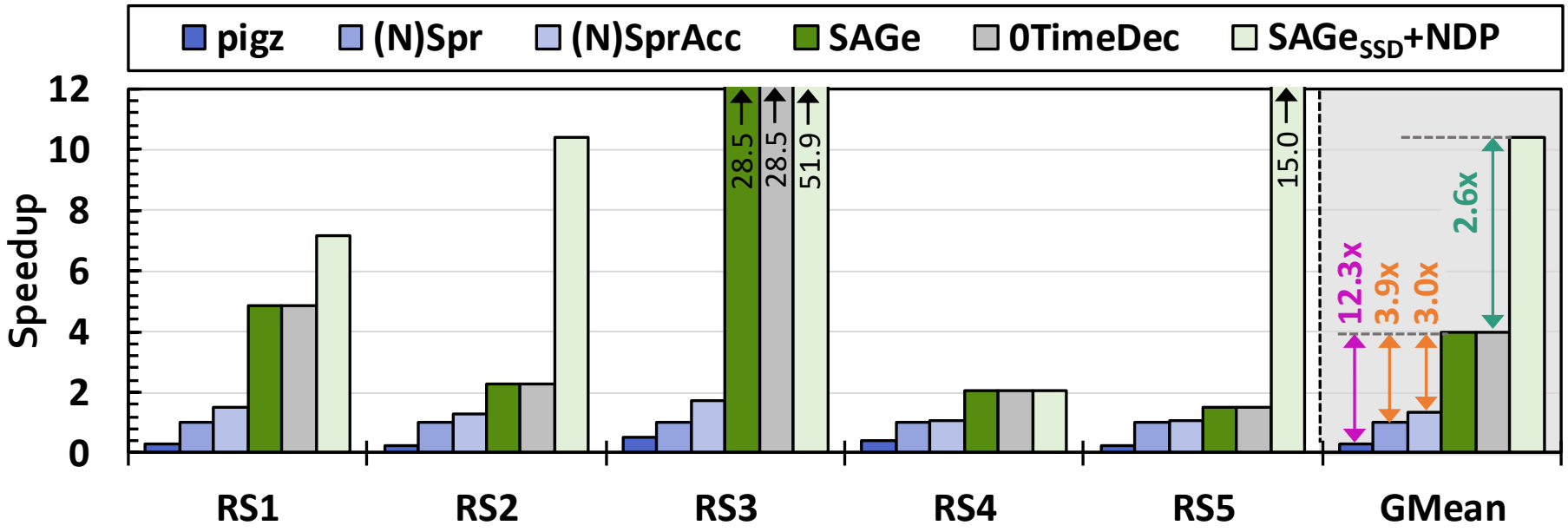
SAGe

Putting It All Together

- **Motivation for combining the proposed designs into a unified system**
  - Storage-centric computing on compressed data
  - Facilitating wider adoption

- **Design of the unified system**
  - Hardware units
    - Only 6% of the four ARM cores on an SSD controller
  - FTL
  - Interface commands
    - A single command initiates storage-centric acceleration, with a parameter selecting the specific mode (GenStore, MegIS, GRAINS, or SAGe)
    - Each acceleration design retains its own interface commands

Putting It All Together: SAGe + GenStore



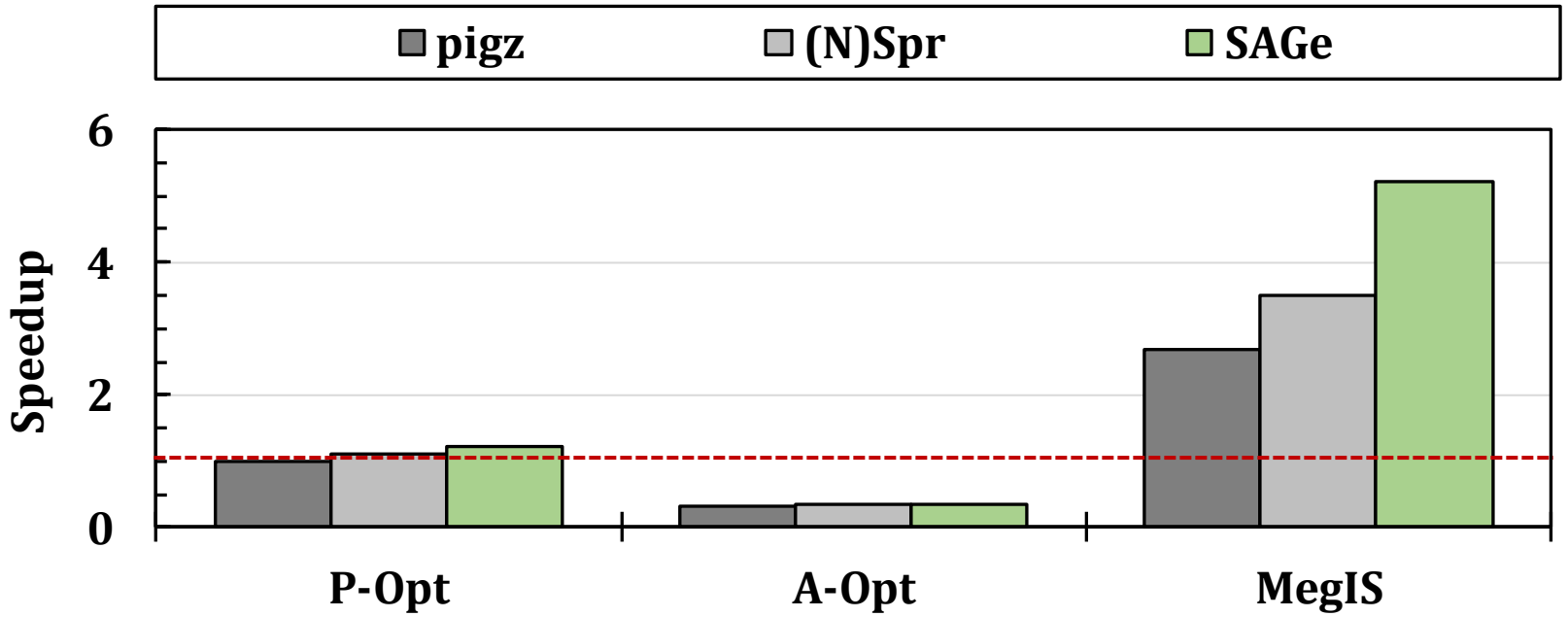
SAGe provides significant speedup over both general-purpose and genomics-specific baselines

SAGe achieves the same speedup as 0TimeDec

By efficiently integrating SAGe$_{SSD}$ with NDP SAGe$_{SSD}$+NDP leads to large speedups

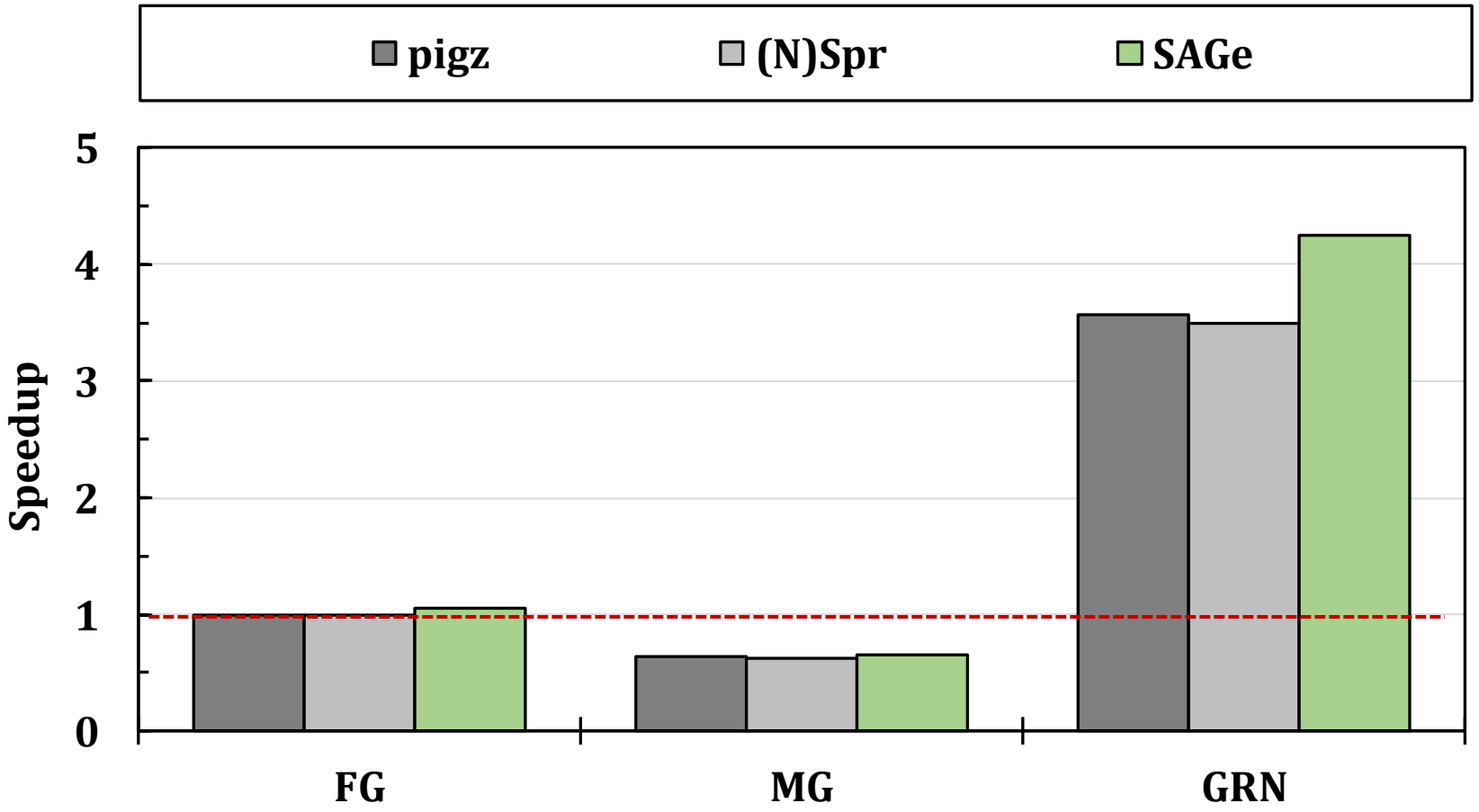
Putting It All Together: SAGe + MegIS

Speedup Normalized to P-Opt+pigz



Putting It All Together: SAGe + GRAINS

Speedup Normalized to FG+pigz



Compression Ratio for Metagenomic Data

Read Set	pigz	(N)Spr	SAGe
CAMI-L	3.53	21.54	32.53
CAMI-M	3.59	17.65	26.20
CAMI-H	3.54	7.35	9.53

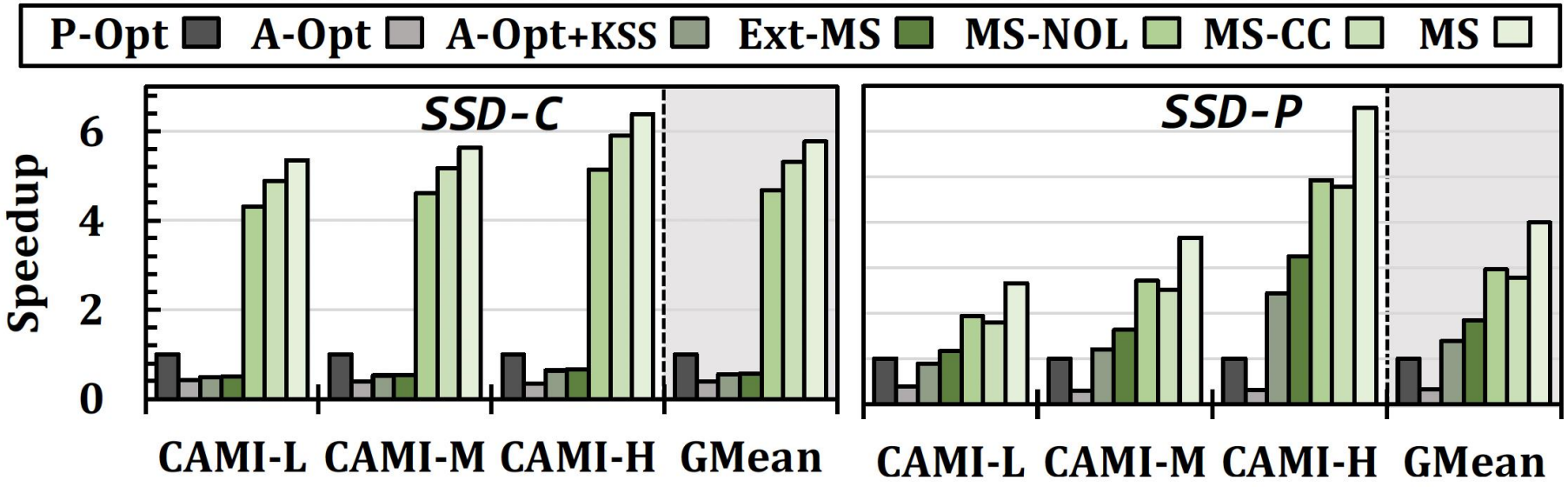
Better average compression ratio than general-purpose baseline

Comparable compression ratio to genomics-specific baseline

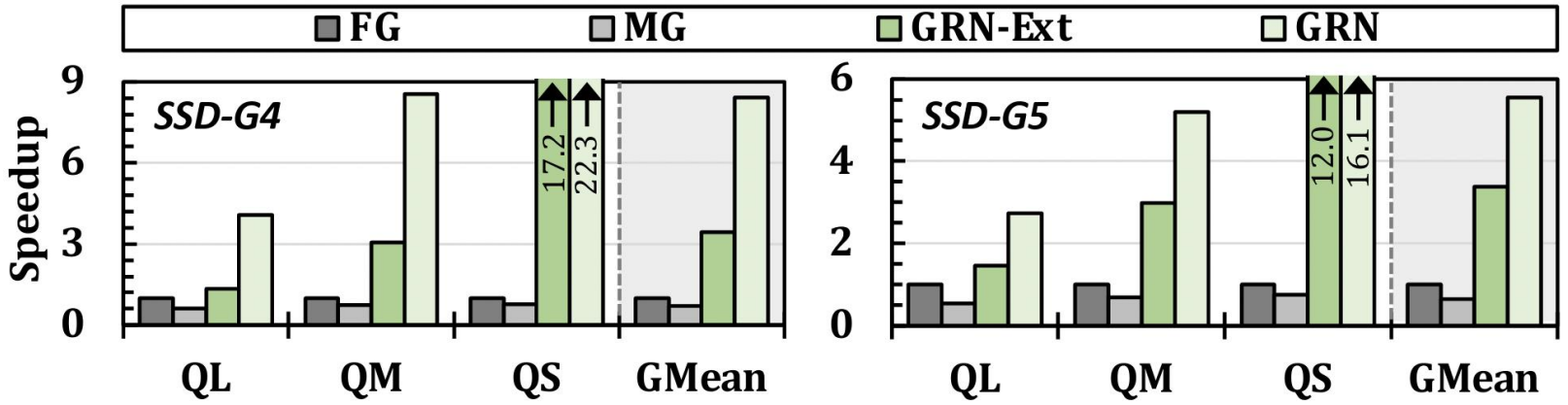
Benefits Outside the SSD

- **Key benefits of storage-centric designs outside storage**
  - Alleviating the burden of moving and analyzing low-reuse data
  - More storage-friendly execution pipelines: Efficient access patterns, and fewer random accesses
    - Efficient use of the available I/O bandwidth
    - Less reliance on large DRAM capacity
- **However, there are more benefits from analysis inside the storage**
  - Higher IFP/ISP bandwidth
  - Avoid moving low-reuse data
  - Does not rely on wider/more external interfaces

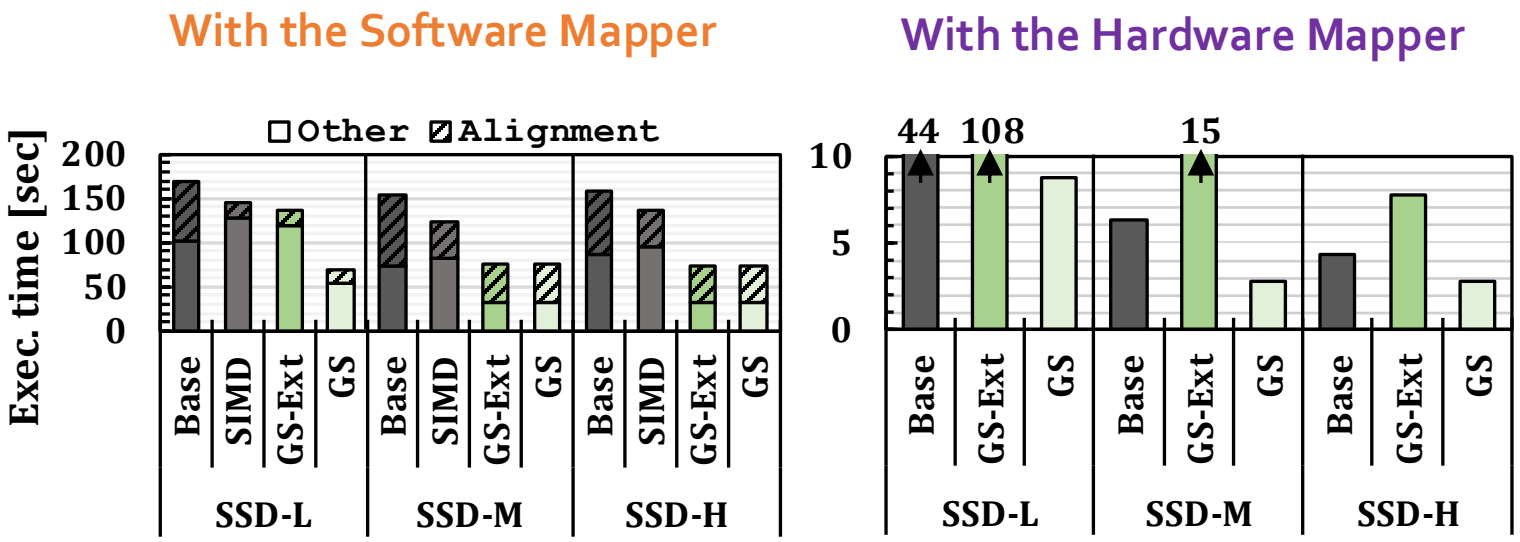
Benefits Outside the SSD: MegIS



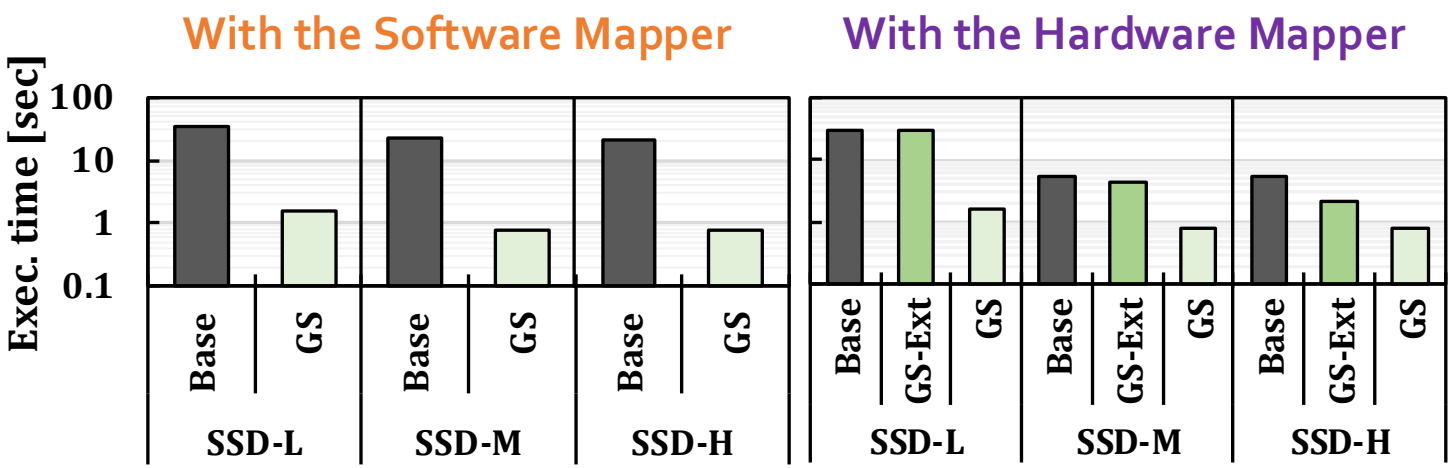
Benefits Outside the SSD: GRAINS



Benefits Outside the SSD: GenStore-EM



Benefits Outside the SSD: GenStore-NM



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

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

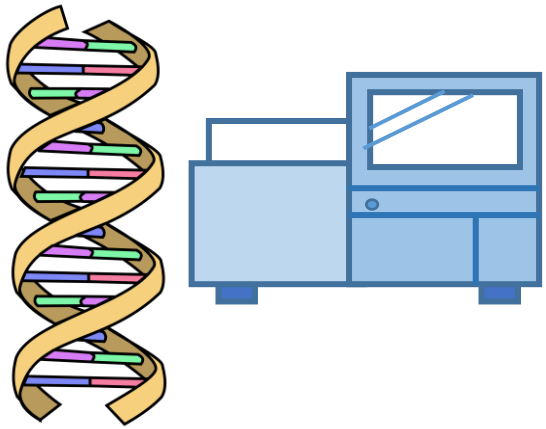
GenStore Discussion

- Main presentation
- Motivation details
- Long read use cases
- GenStore-NM detailed architecture
- Detailed results
- Effect of inputs

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

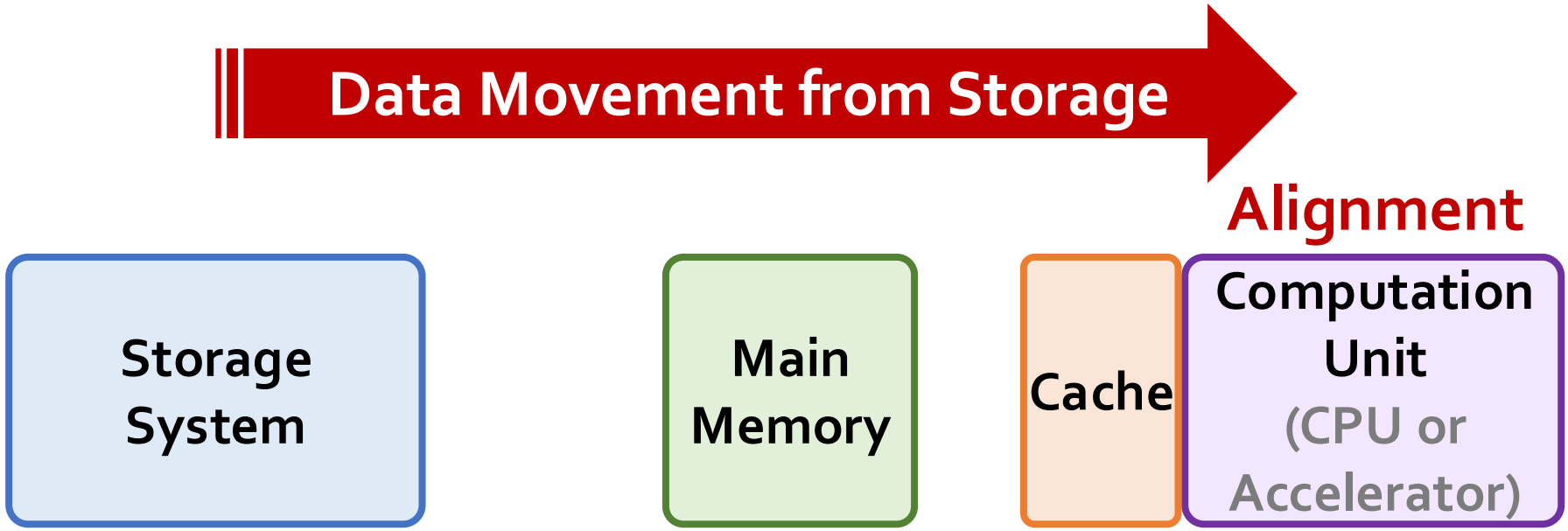


Genome Sequence Analysis

- **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

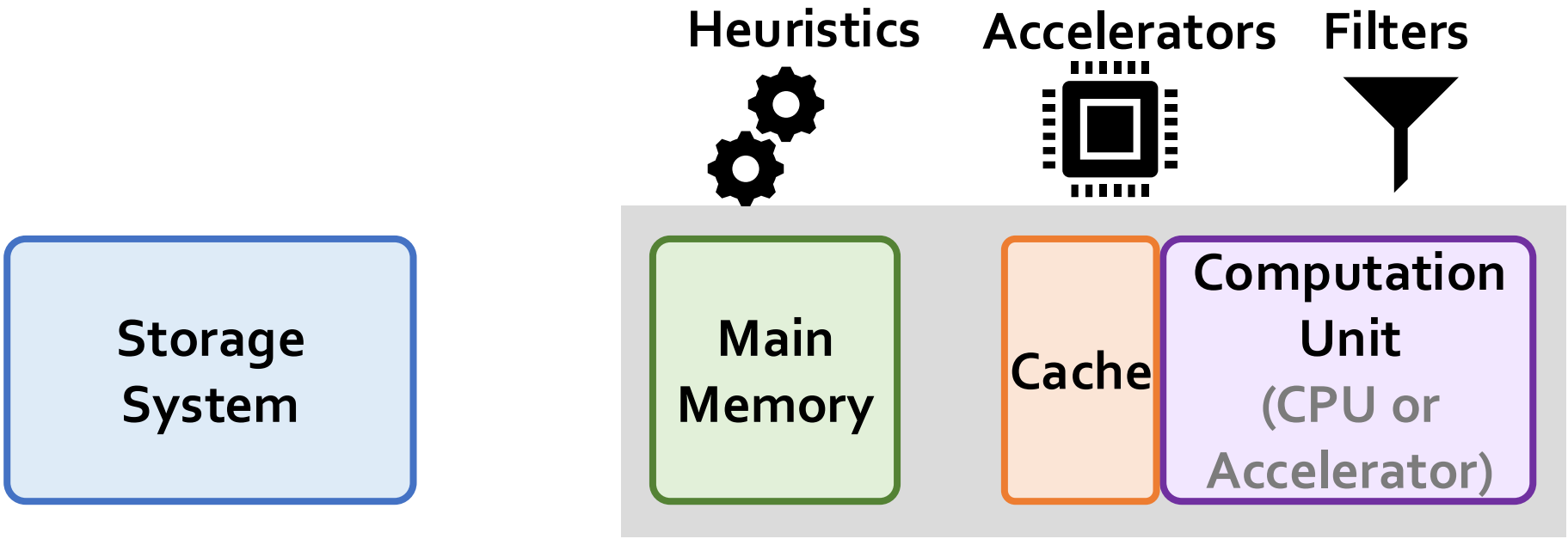
Genome Sequence Analysis



✗ Computation overhead

✗ Data movement overhead

Accelerating Genome Sequence Analysis

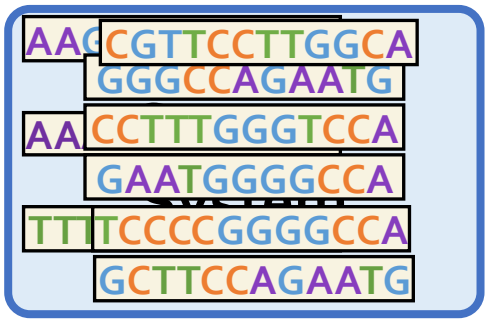


✓ Computation overhead

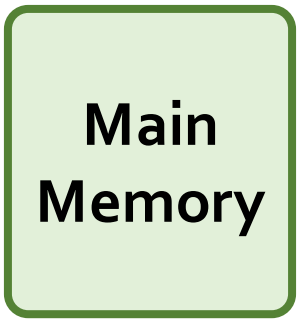
✗ Data movement overhead

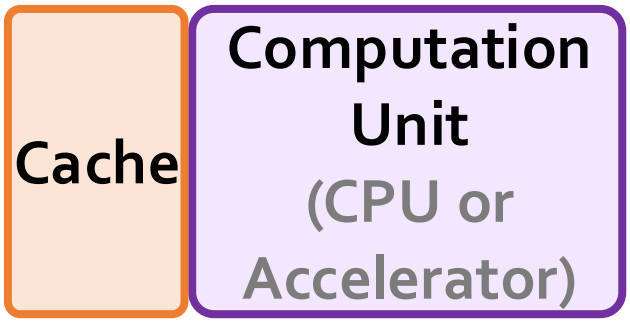
Key Idea

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



Filtered Reads





Exactly-matching reads

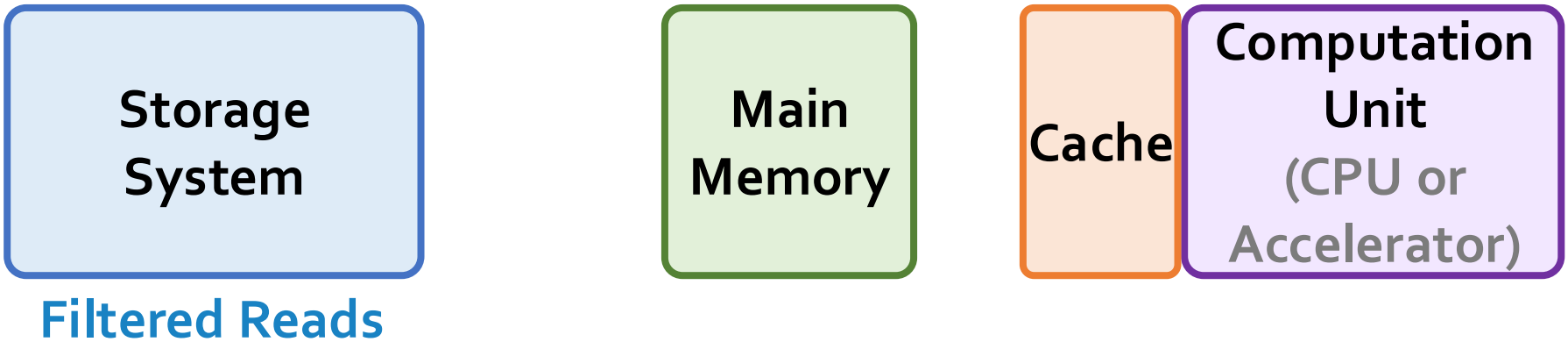
Do not need expensive approximate string matching during alignment

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*

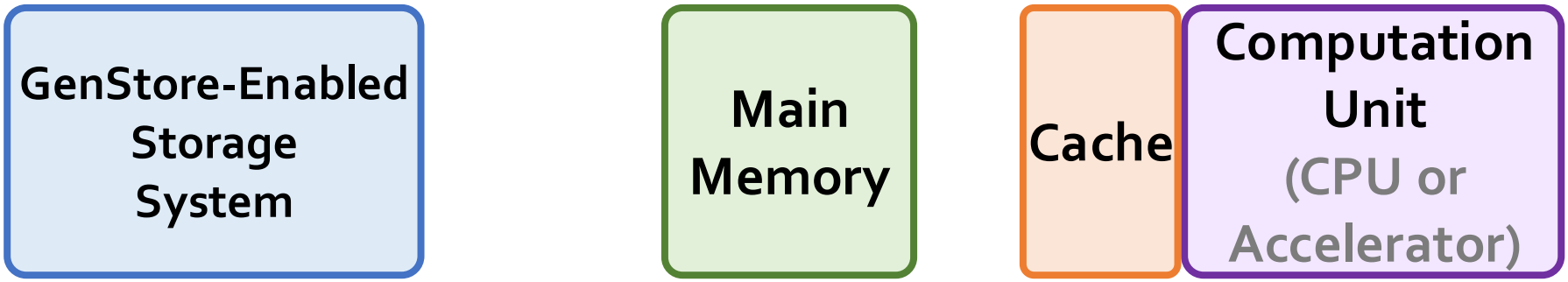


Read mapping workloads can exhibit different behavior

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

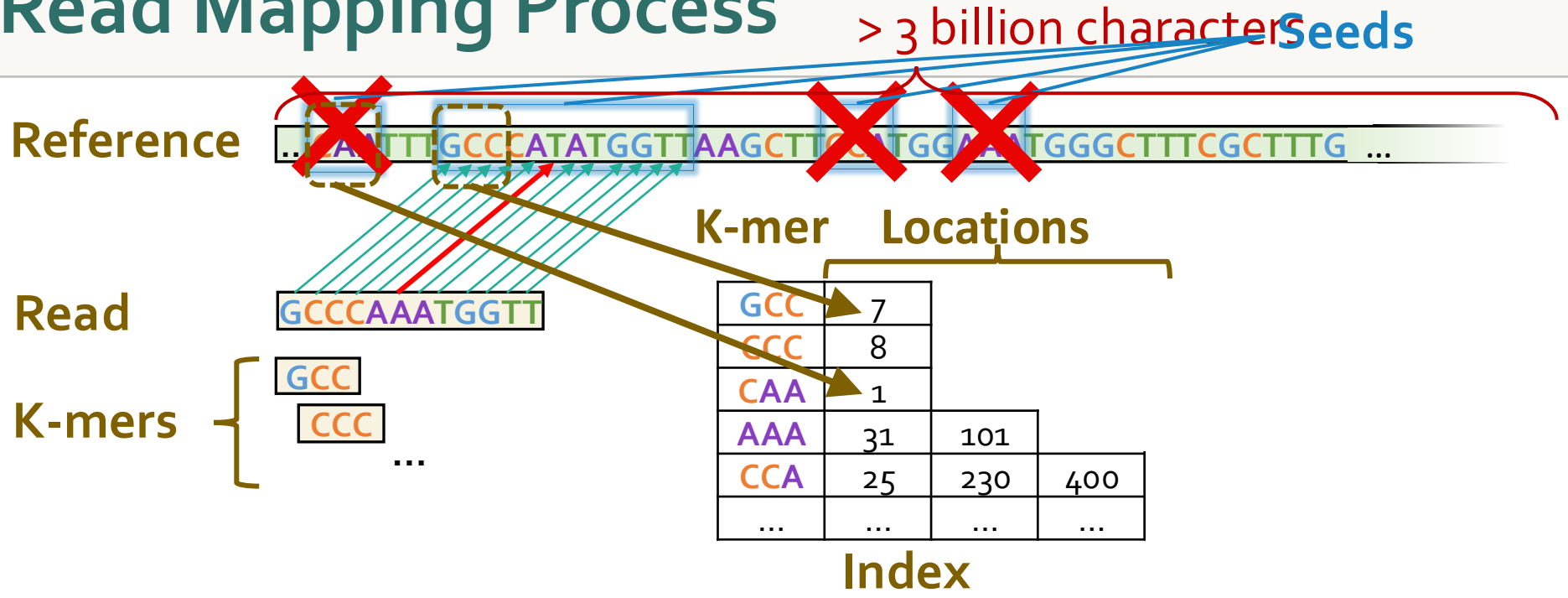
Motivation and Goal

GenStore

Evaluation

Conclusions

Read Mapping Process



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

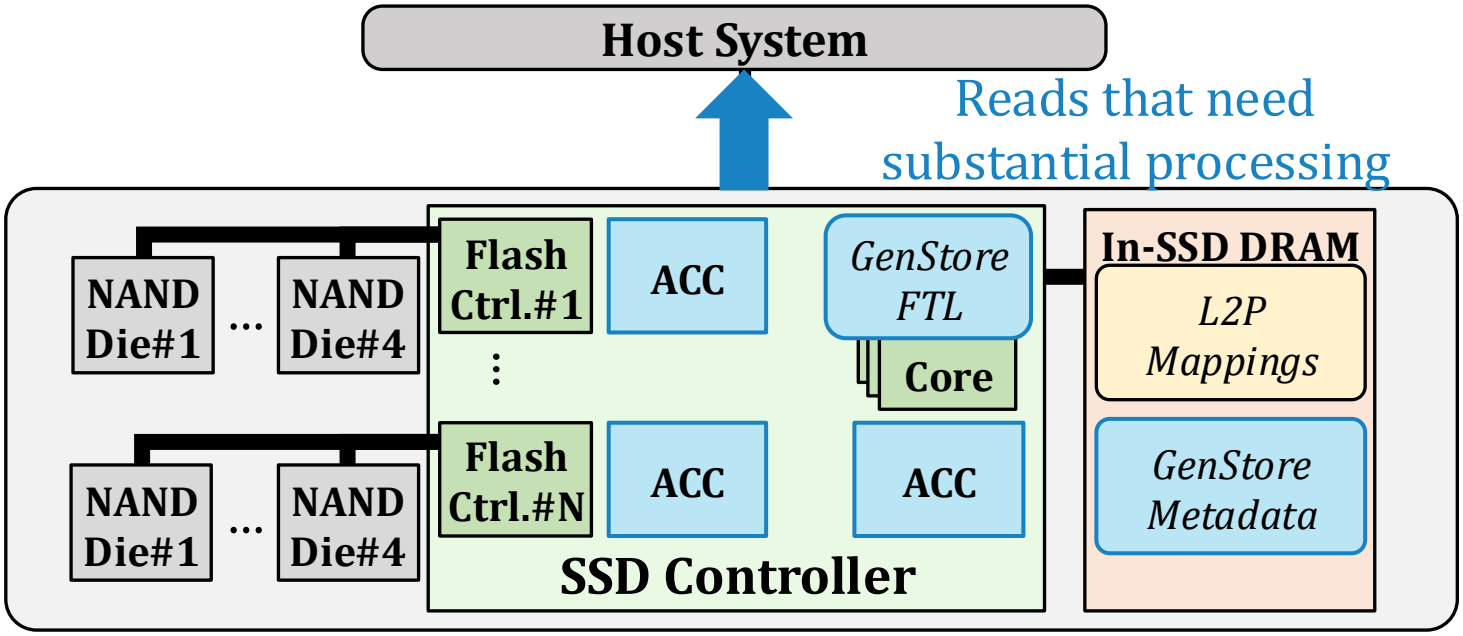
Evaluation

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 Exactly-Matching Reads

GenStore-NM for Non-Matching Reads

GenStore

GenStore-EM for Exactly-Matching Reads

GenStore-NM for Non-Matching Reads

GenStore-EM

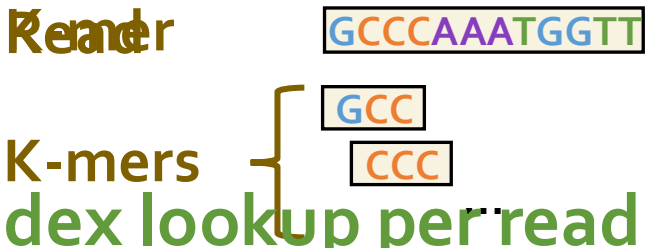
- Efficient in-storage filter for reads with at least one exact match in the reference genome
- Uses simple operations, without requiring alignment
- **Challenge:** large number of random accesses per read to the reference genome and its index

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

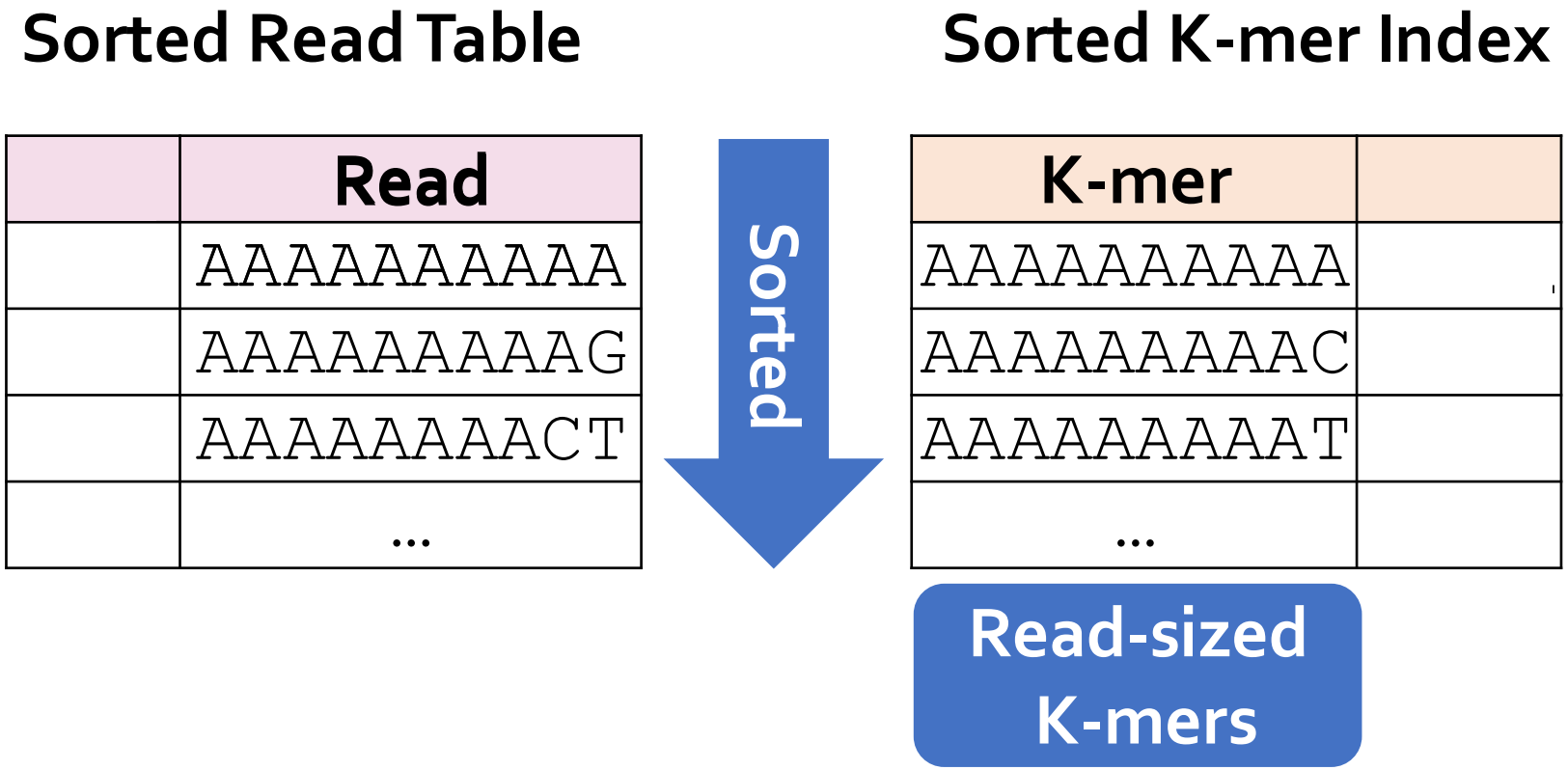


✓ Only one index lookup per read

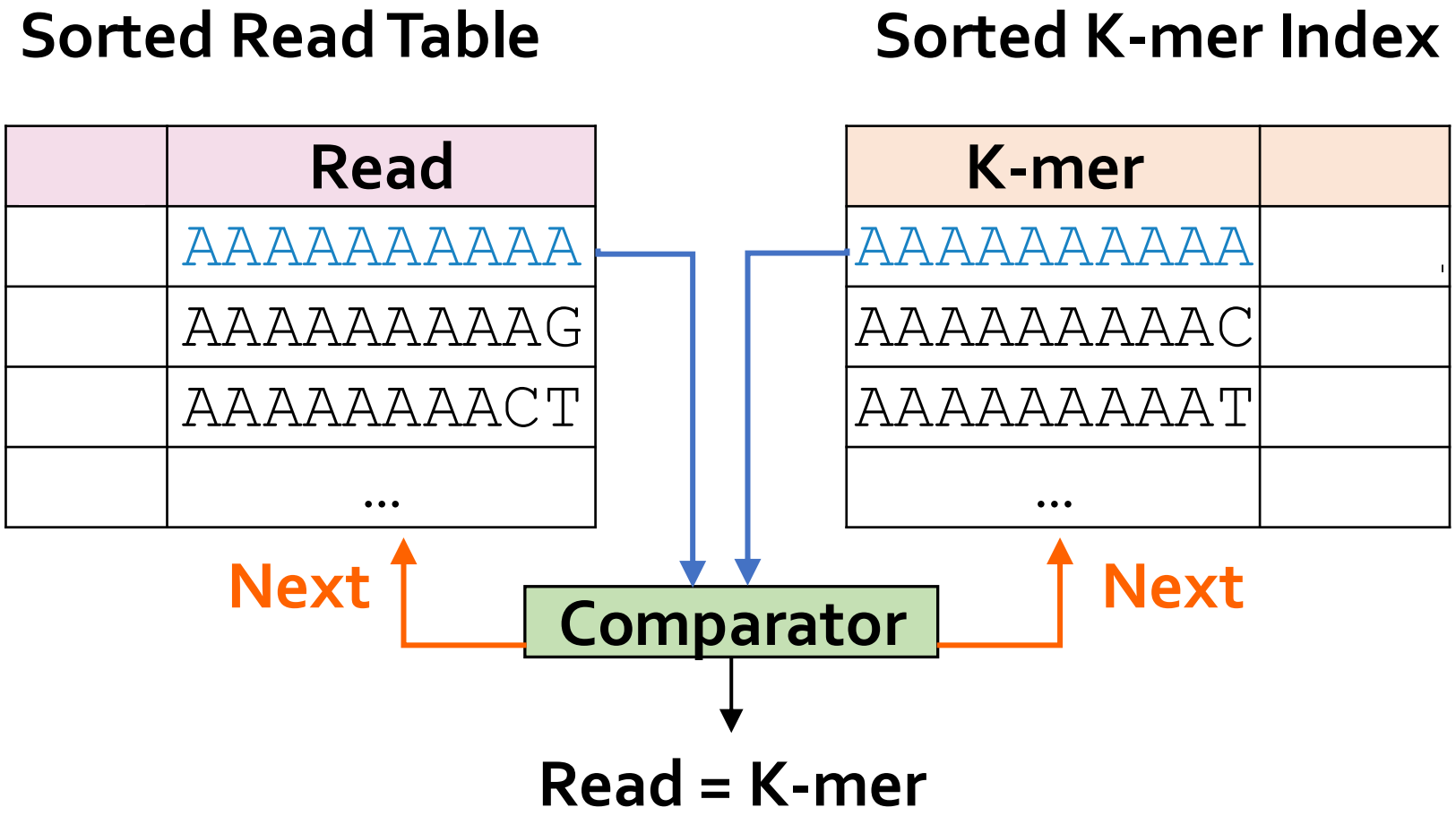
- **Sorted read-sized k-mers:** to avoid random accesses to the index

✓ Sequential scan of the read set and the index

GenStore-EM: Data Structures

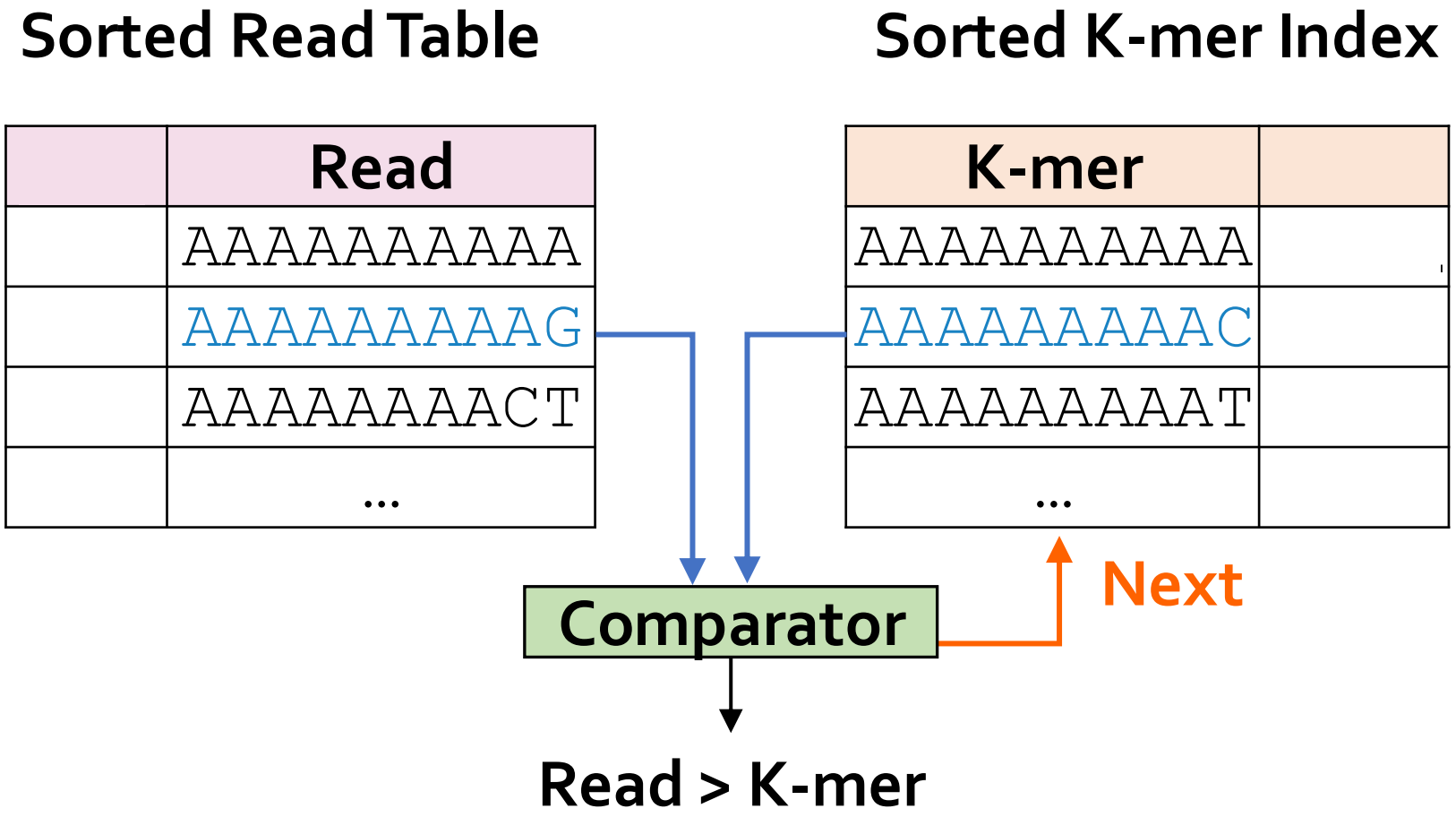


GenStore-EM: Finding a Match

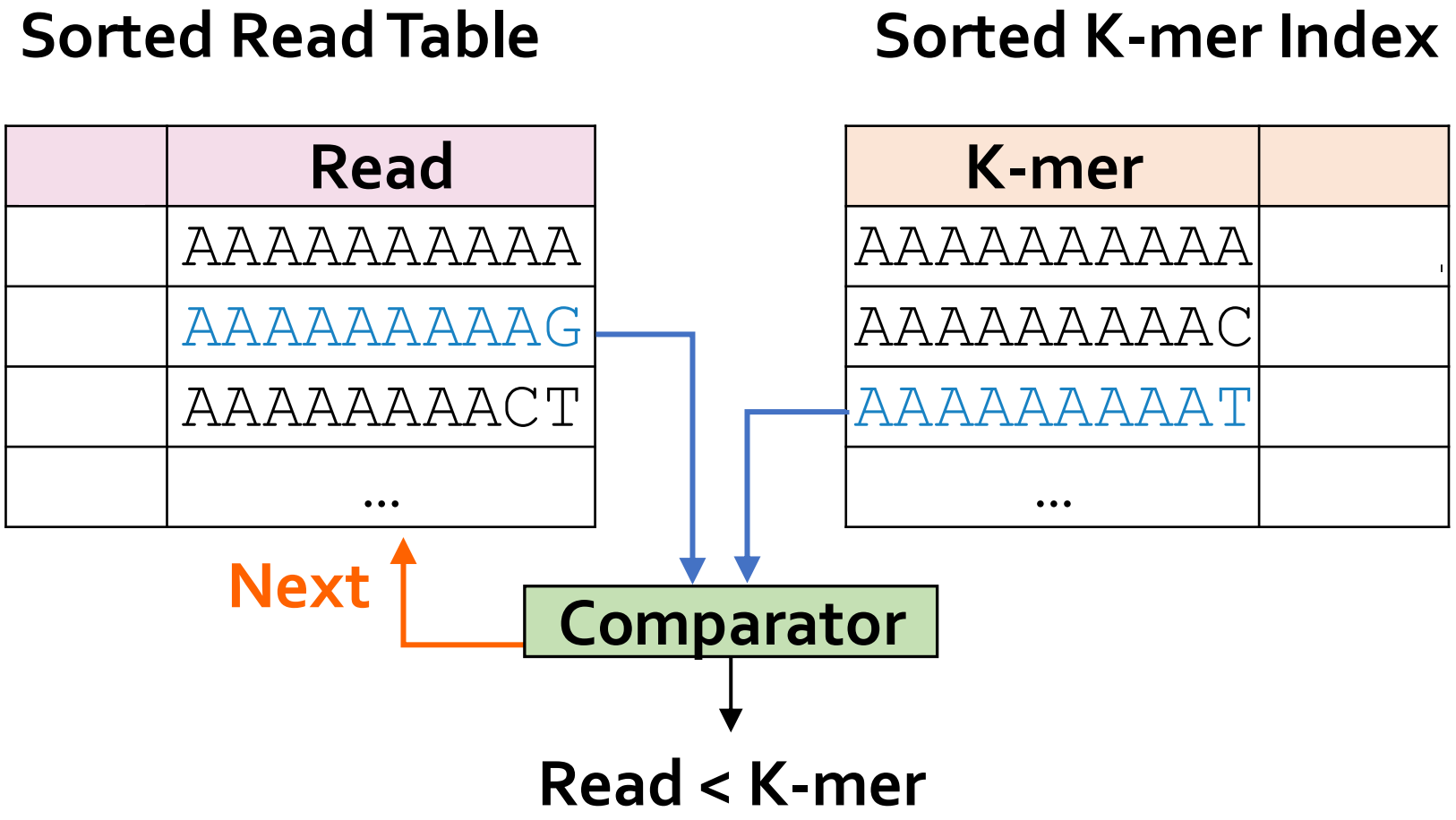


Exact match → Filter the read

GenStore-EM: Not Finding a Match

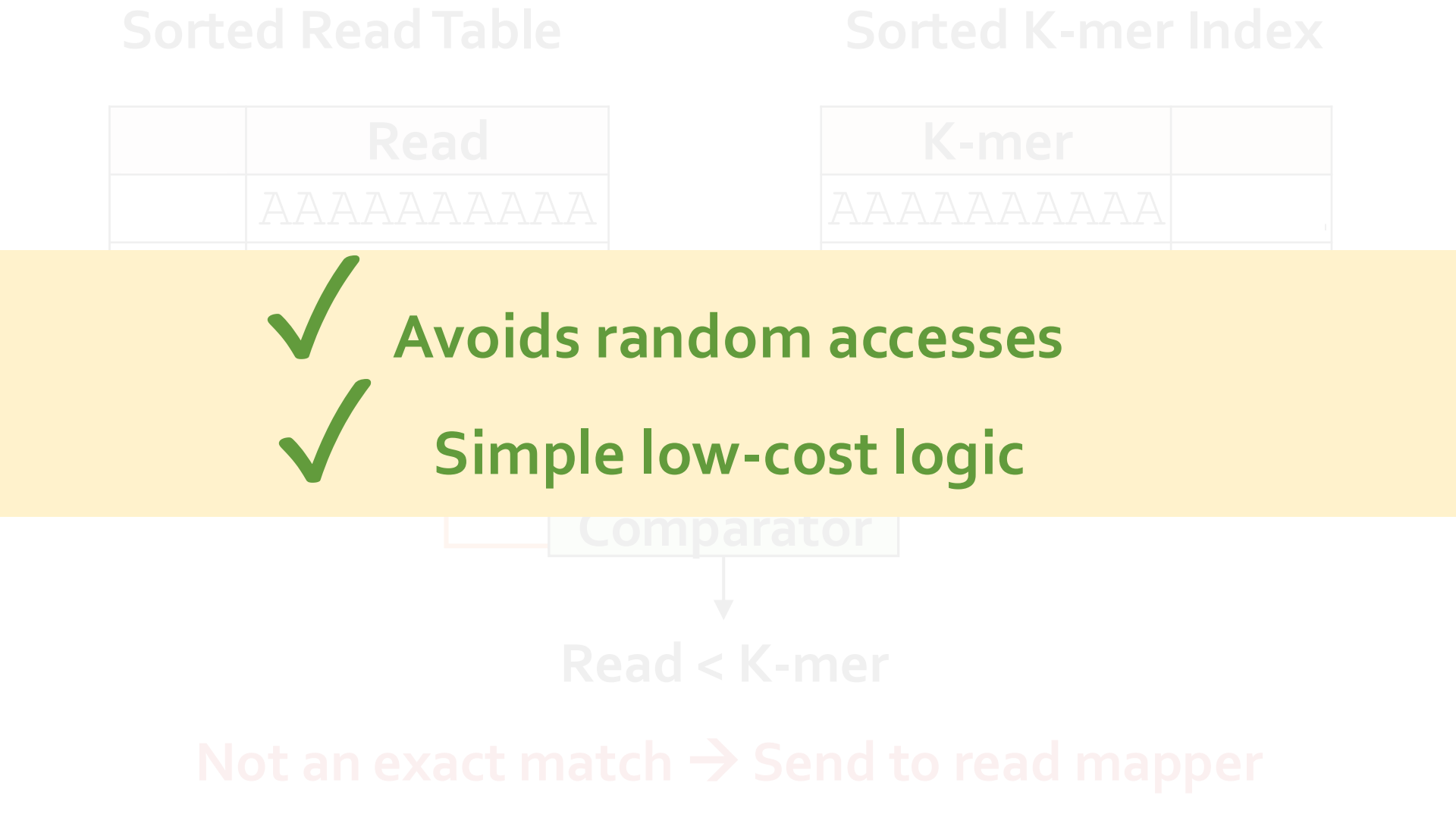


GenStore-EM: Not Finding a Match



Not an exact match → Send to read mapper

GenStore-EM: Not Finding a Match



✓ Avoids random accesses

✓ Simple low-cost logic

GenStore-EM: Optimization

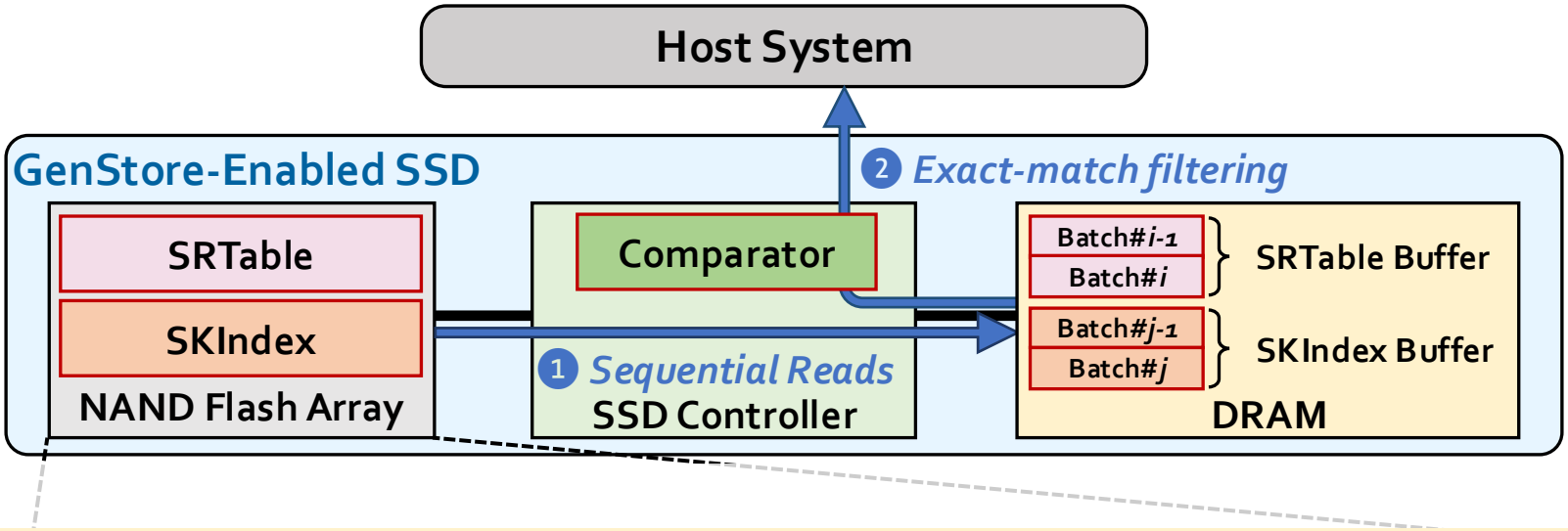
- Read-sized k-mer index takes up a large amount of space (126 GB for human index) due to the larger number of unique k-mers

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



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

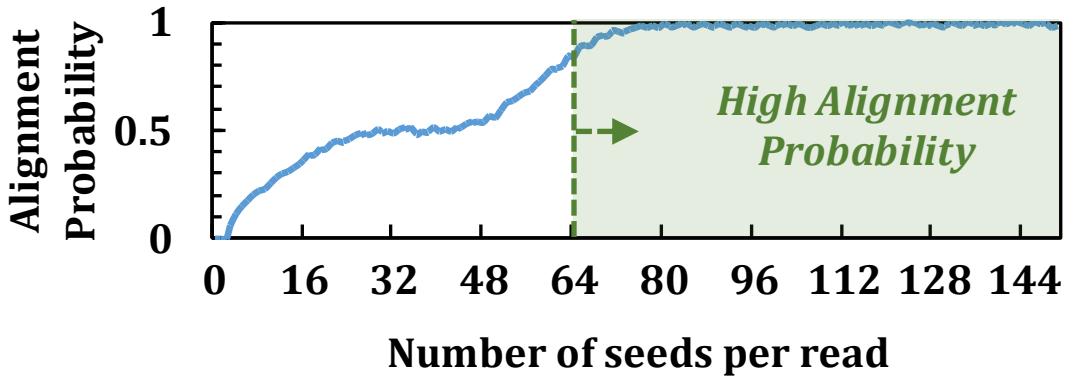
- **Challenge:** how to perform chaining inside the SSD

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

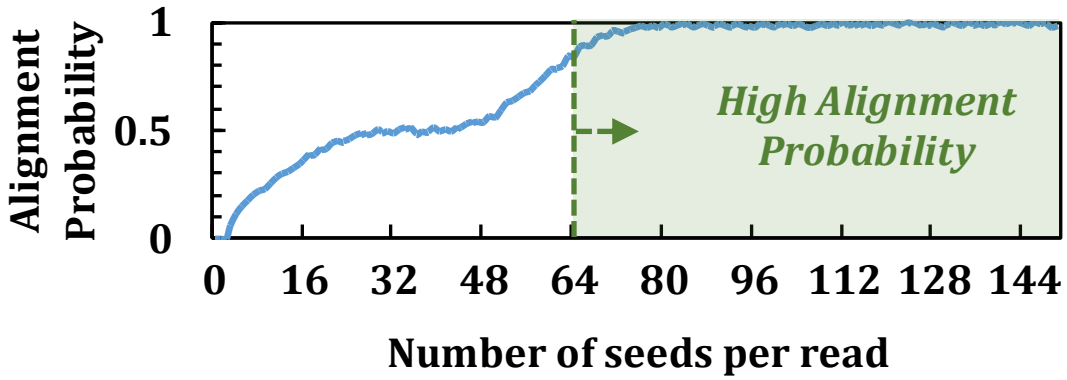


Reads with a sufficiently large number of seeds are very likely to align to the reference genome

✓ **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

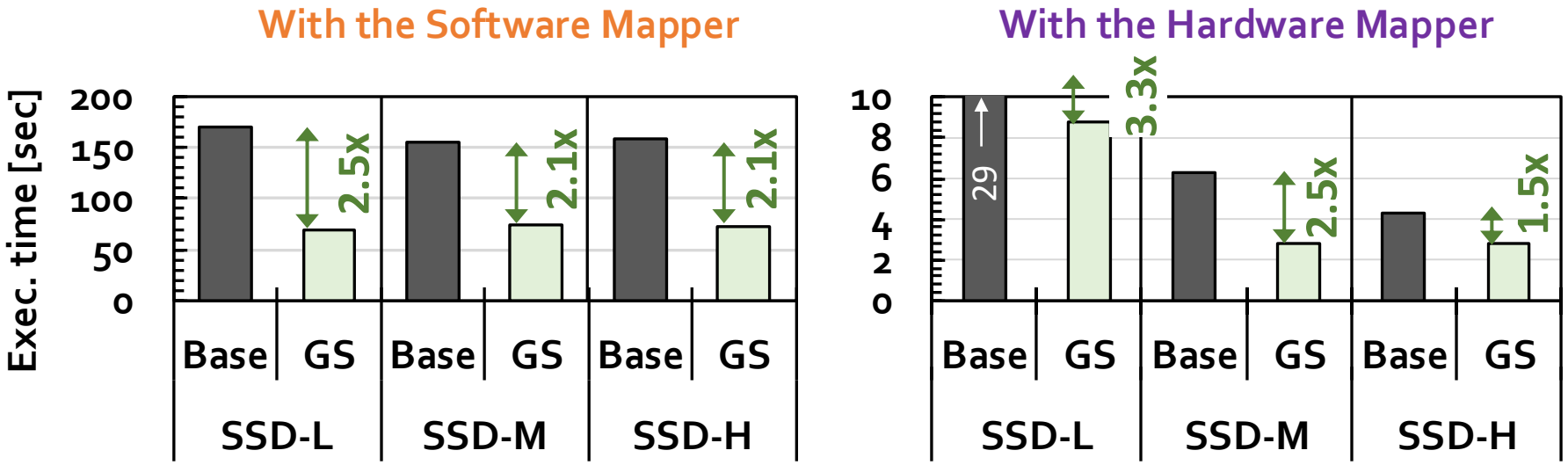
- **Base:** state-of-the-art software or hardware read mappers
  - Minimap2 [Bioinformatics'18]: software mapper for short and long reads
  - GenCache [MICRO'19]: hardware mapper for short reads
  - Darwin [ASPLOS'18]: hardware mapper for long reads
- **GS:** Base integrated with GenStore

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



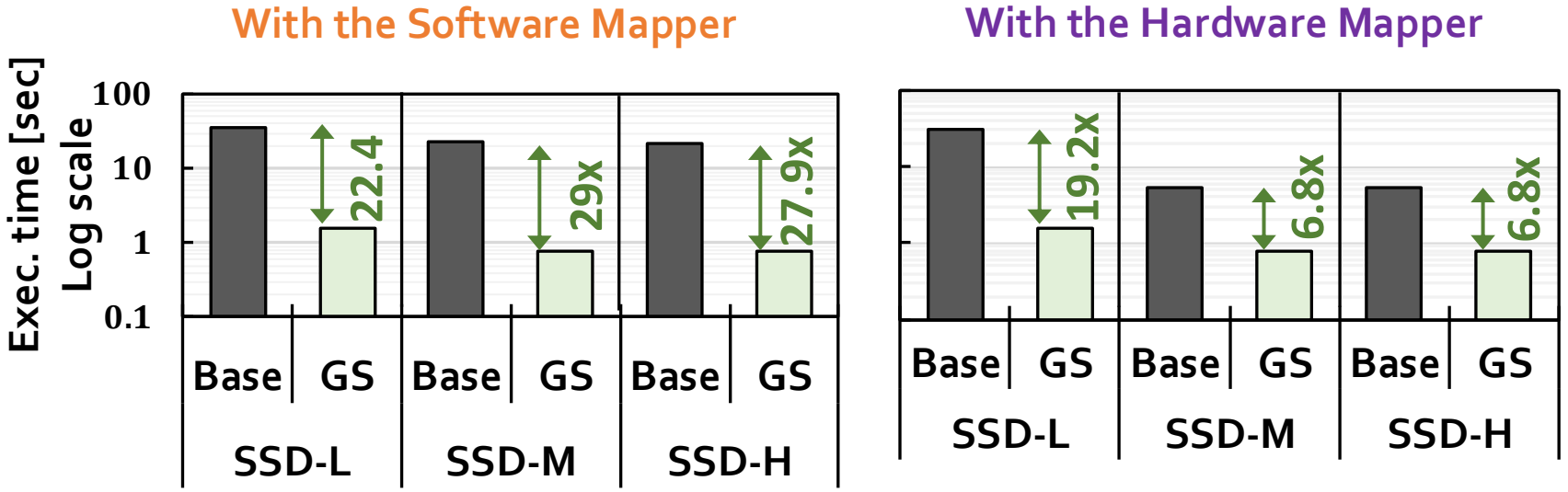
2.1× - 2.5× speedup compared to the software Base

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



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, 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

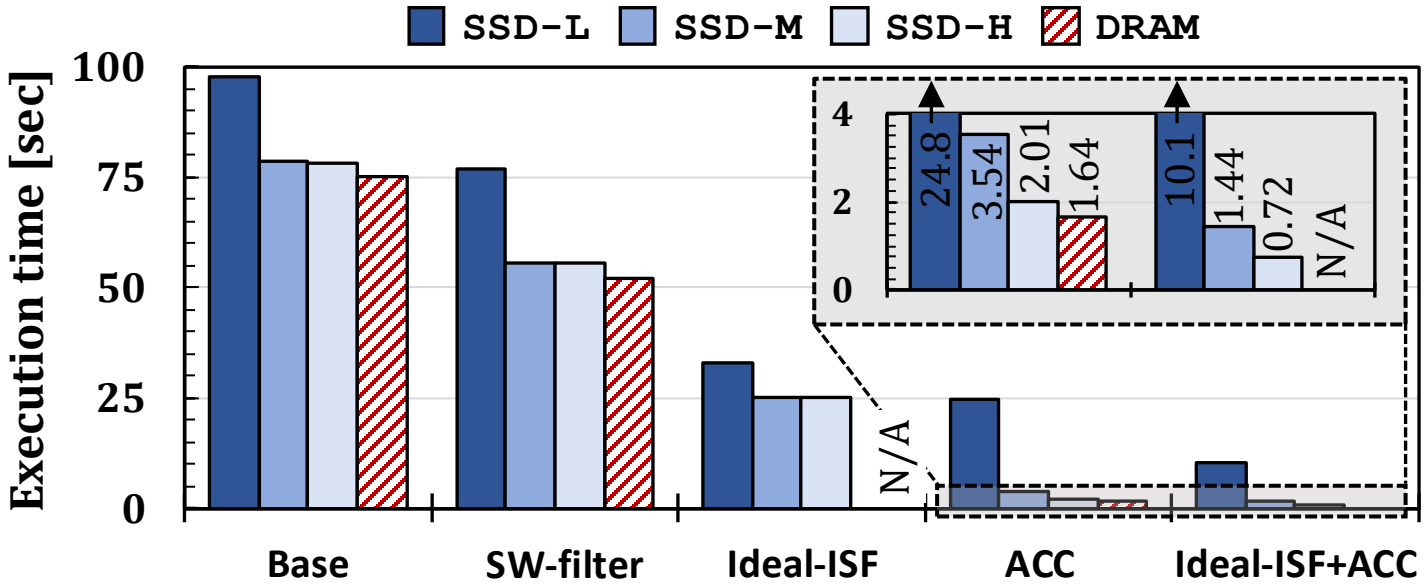
bionano GENOMICS

UNIVERSITY OF TORONTO

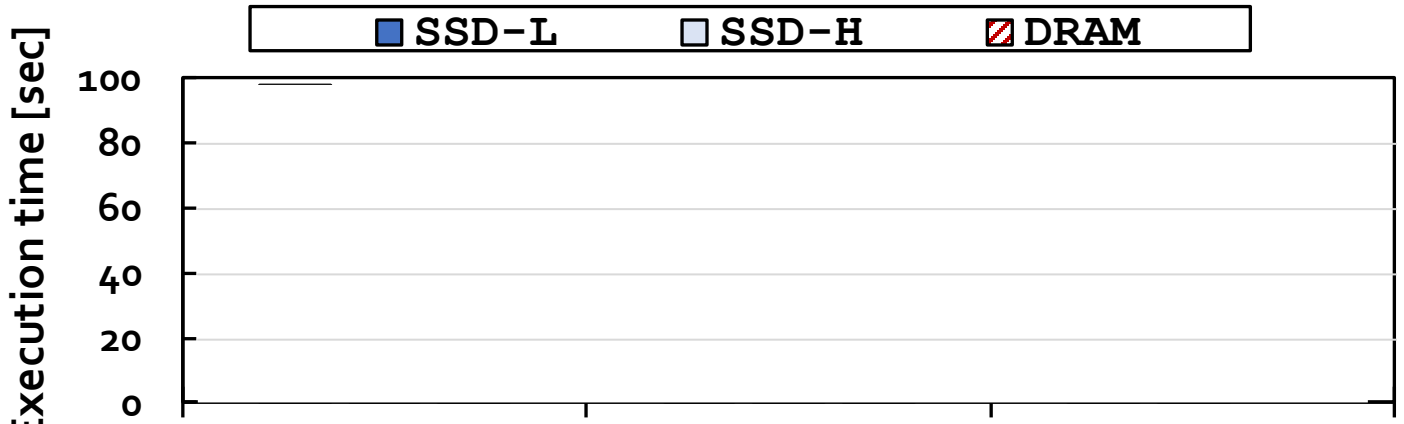
End-to-End Workflow of Genome Sequence Analysis

- 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
4
2
0
24.8
3.54
2.01
1.64
10.1
1.44
0.72
N/A
N/A
Base
SW-filter
Ideal-ISF
ACC
Ideal-ISF+ACC

Motivation



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

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

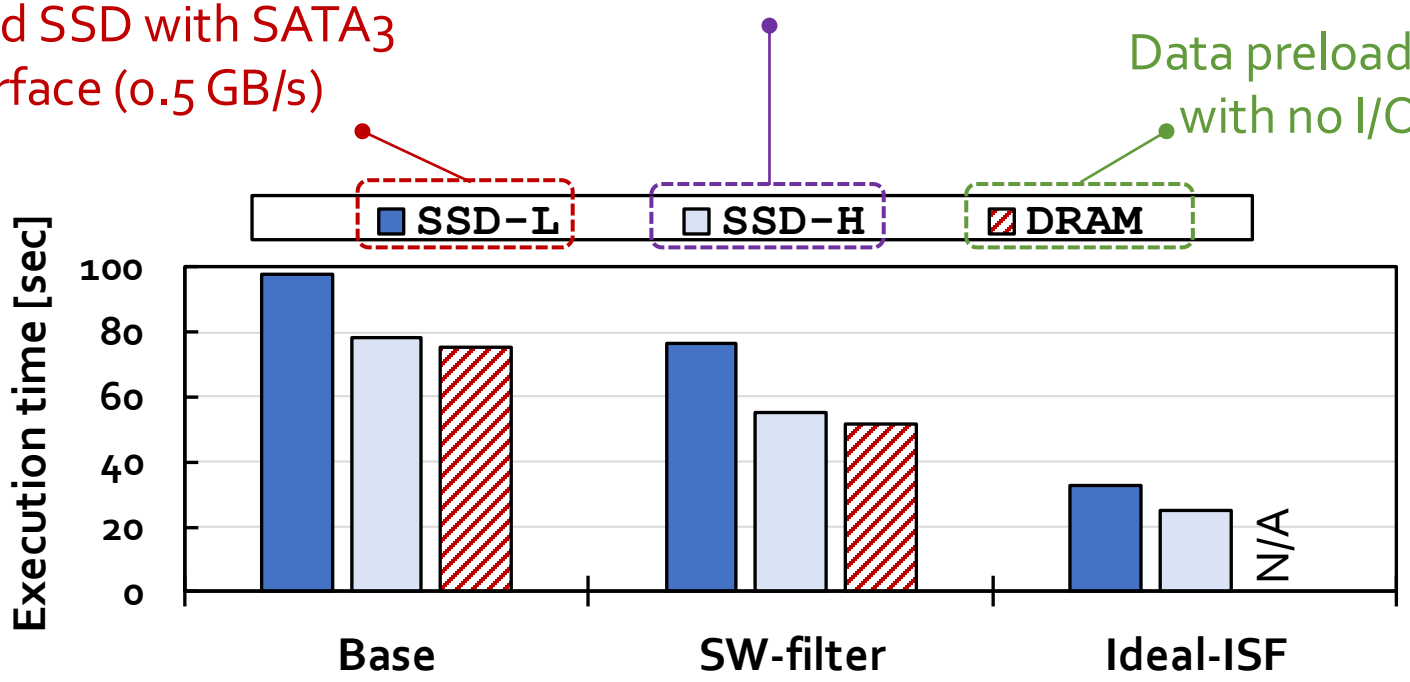
Base integrated with an ideal in-storage filter

Motivation

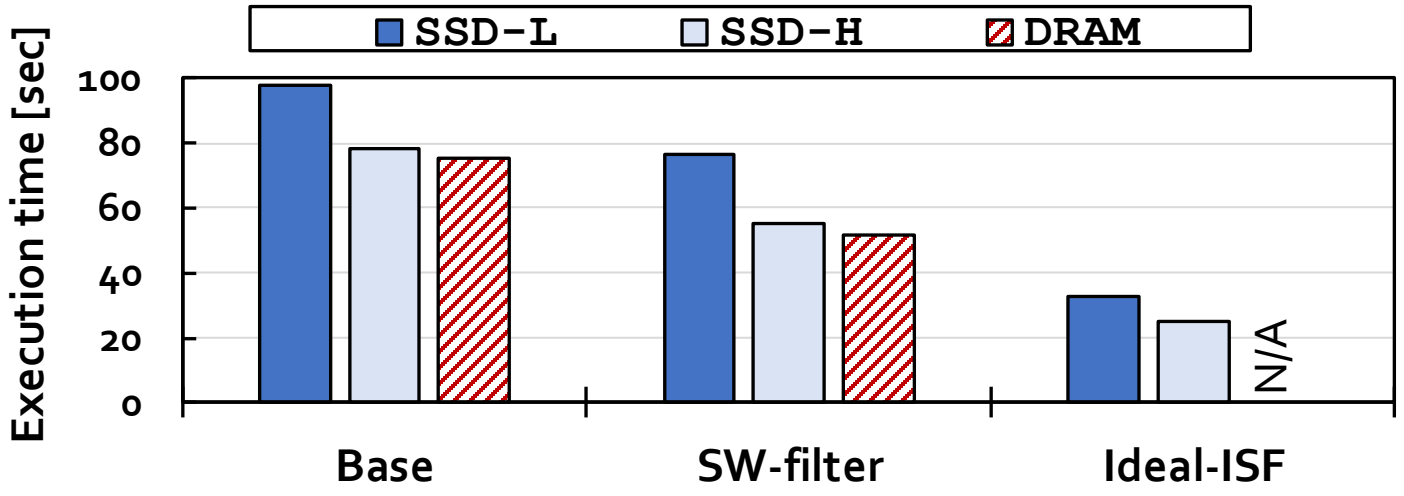
High-end SSD with PCIe Gen4 interface (7 GB/s)

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

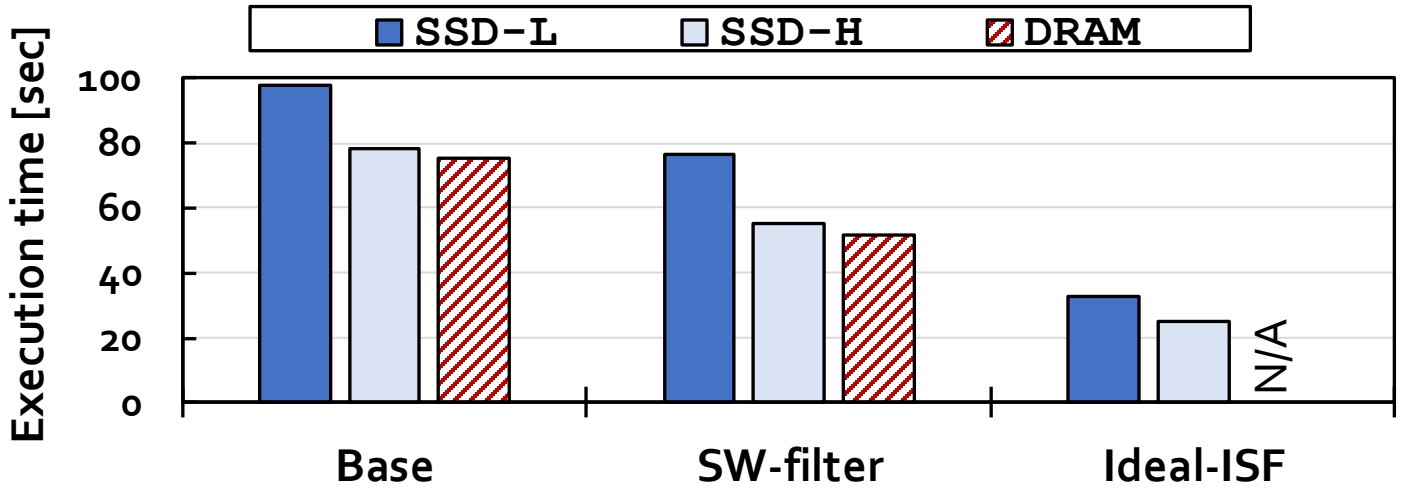


The ideal in-storage filter significantly improves performance by

1) **Reducing computation overhead**

2) **Reducing data movement overhead**

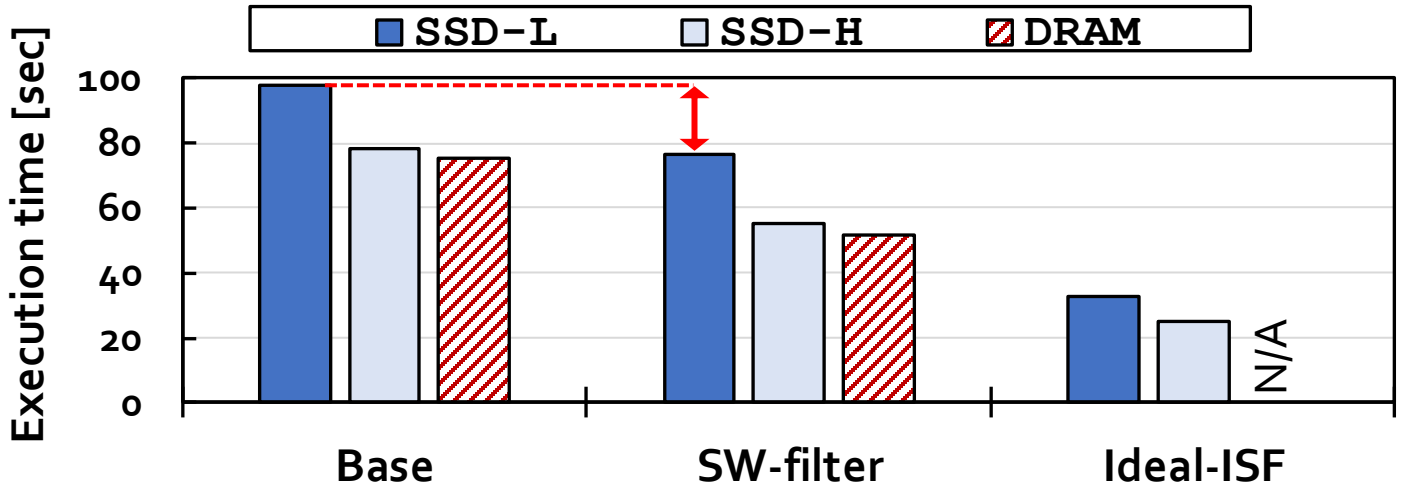
Overheads of Software Mappers



I/O has a significant impact on application performance

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

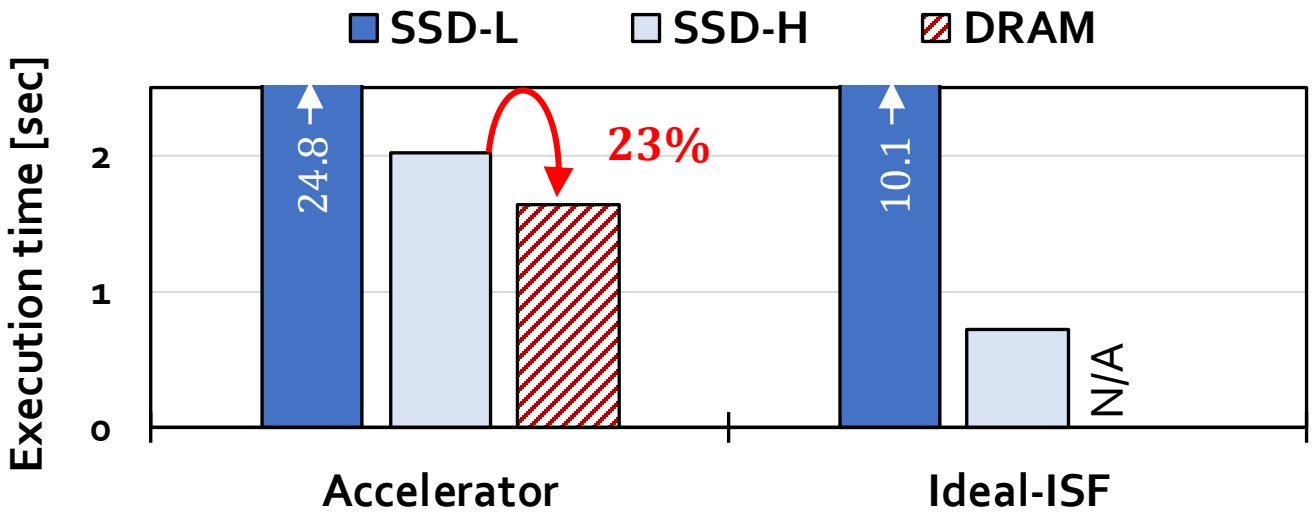
Overheads of Software Mappers



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.

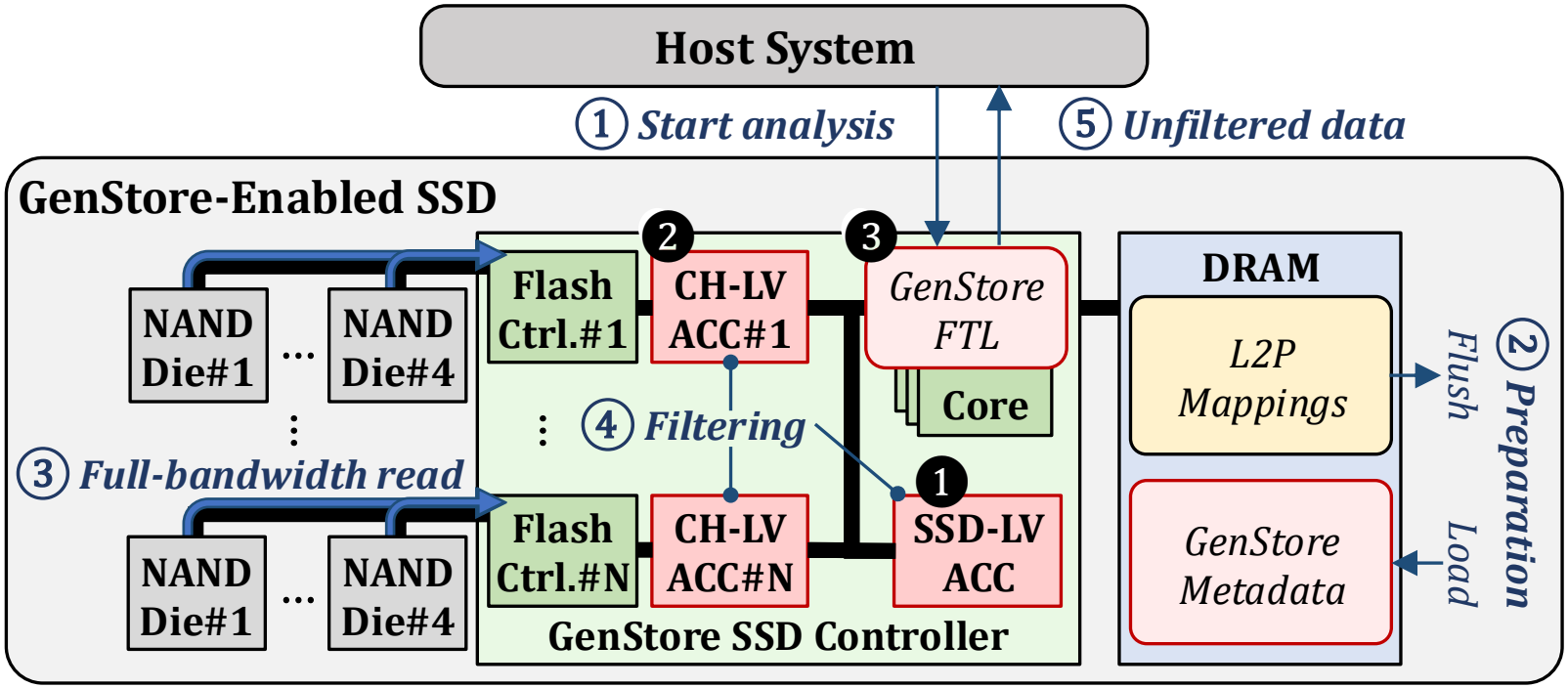
- Even if an ideal accelerator achieved a zero execution time, there would still exist the need to bring the data from storage to the accelerator.
  - 2.15x slower than the execution time that Ideal-ISF+ACC provides in our motivational analysis

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] HG002_ONT_20200204 (L) [158]	54 371	hg38 [144]	**47.4** **69.3**
Rapidly evolving samples	SRR5413248 (L) [157] SRR12423642 (S) [157]	1.69 0.466	NZ_NJEX02 [159] NC_045512.2 [160]	**60.0** **23.1**
No reference	SRR6767727 (L) [157] SRR9953689 (L) [157]	12.4 15.9	NZ_NJEX02 [159]	**0.35** **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
NAND Die#1
NAND Die#4
3 Full-bandwidth read
Flash Ctrl.#1
Flash Ctrl.#N
2
CH-LV ACC#1
CH-LV ACC#N
4 Filtering
3
GenStore FTL
Core
1
SSD-LV ACC
GenStore SSD Controller
DRAM
L2P Mappings
GenStore Metadata
2 Preparation
Flush
Load

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)

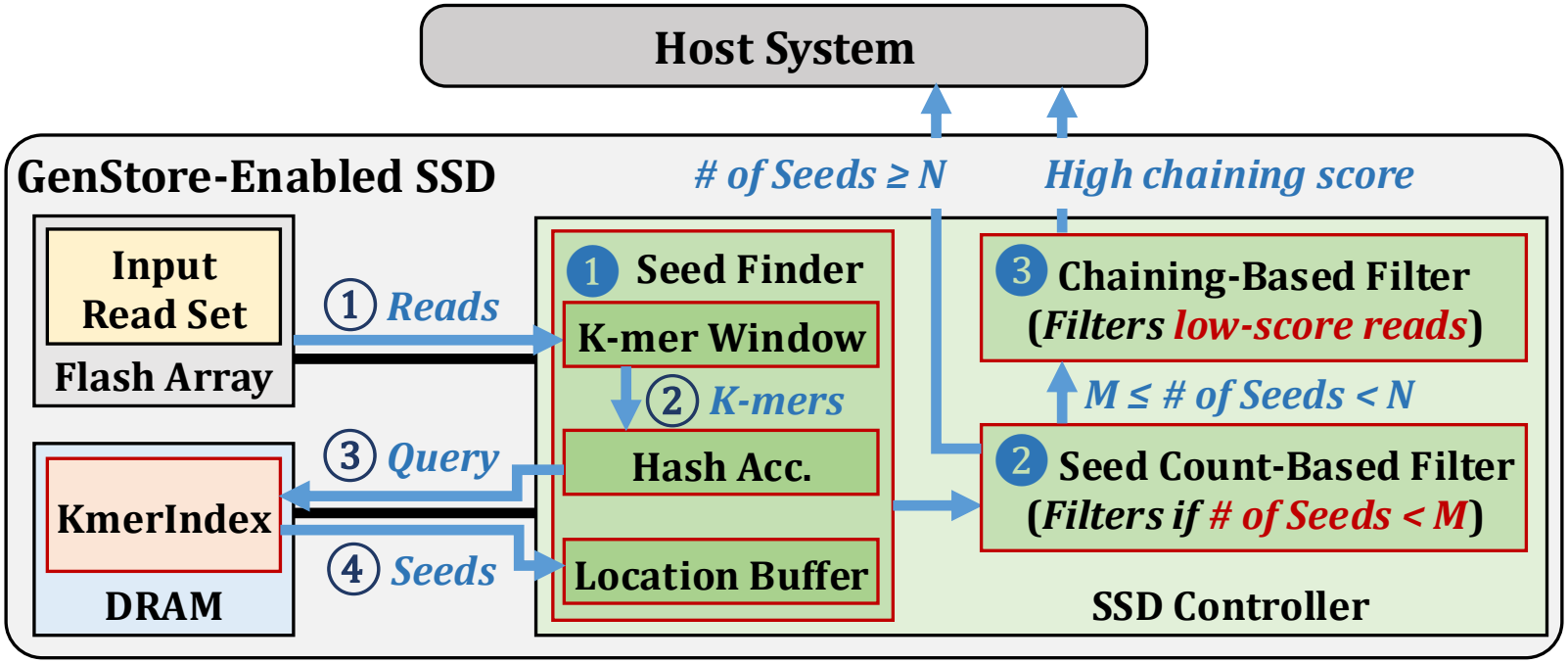
SSD Specs

- **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

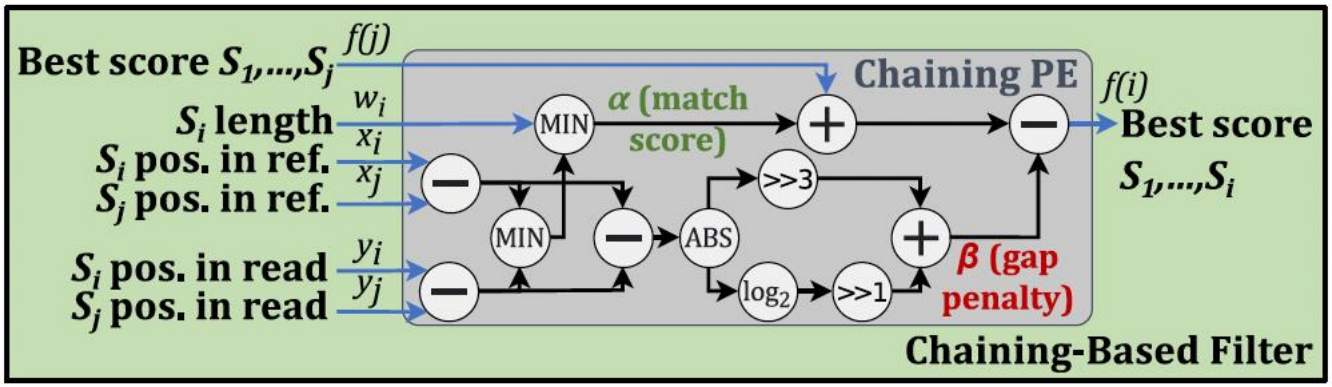
Evaluation Methodology

- **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

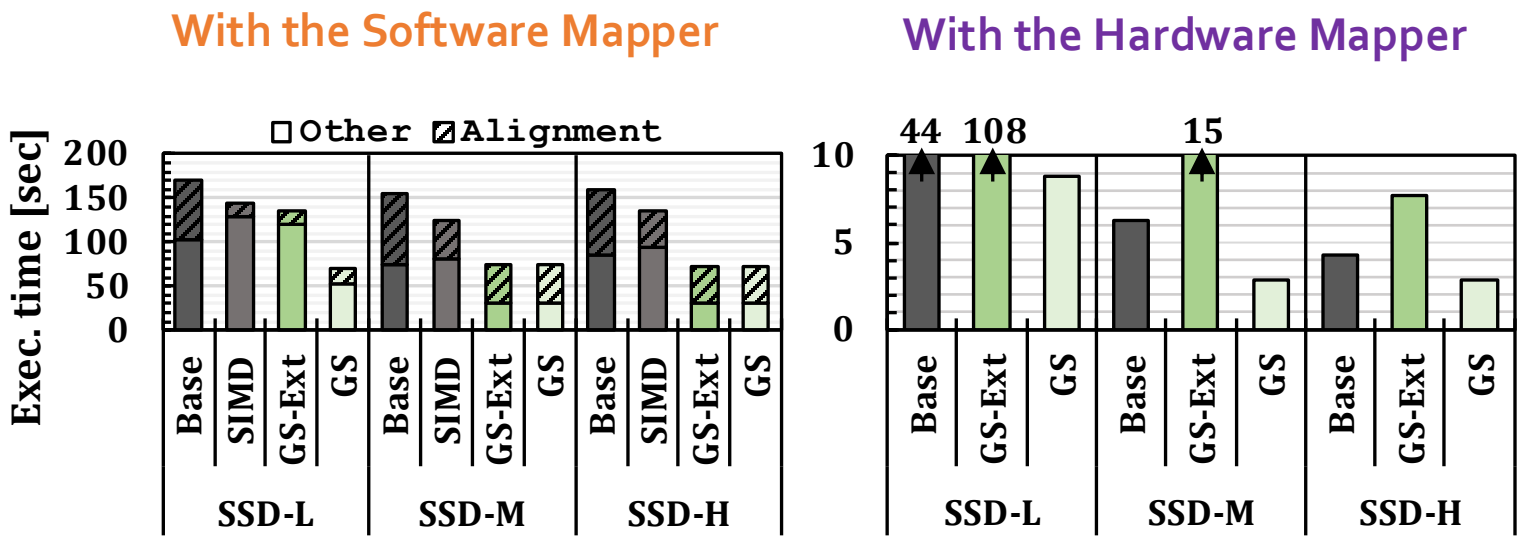
GenStore-NM


Host System
GenStore-Enabled SSD
# of Seeds ≥ N
High chaining score
Input Read Set
Flash Array
1 Reads
KmerIndex
DRAM
3 Query
4 Seeds
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



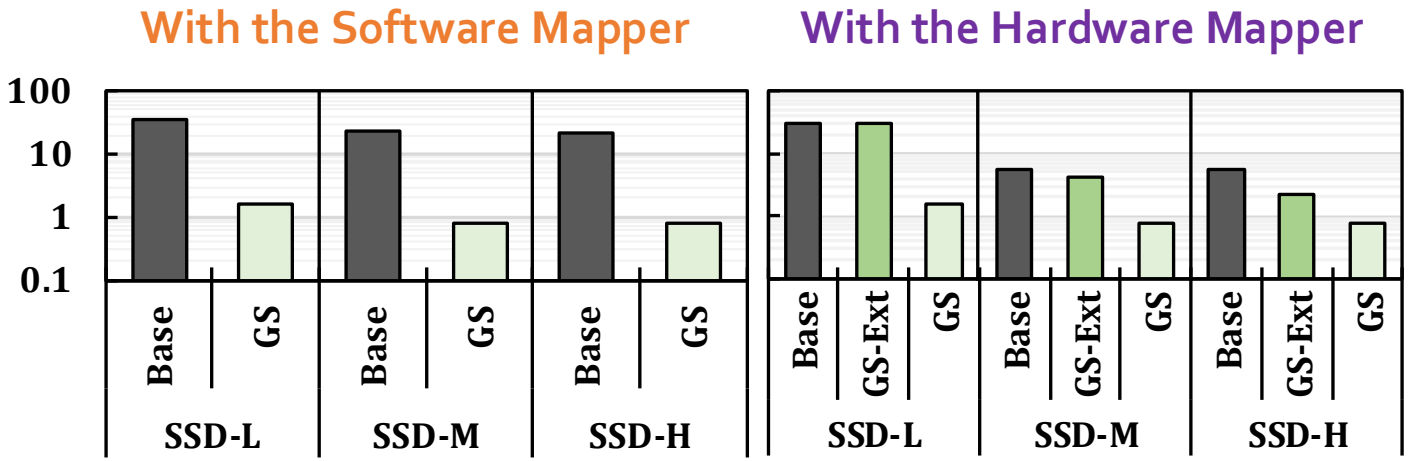
GenStore-EM



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.

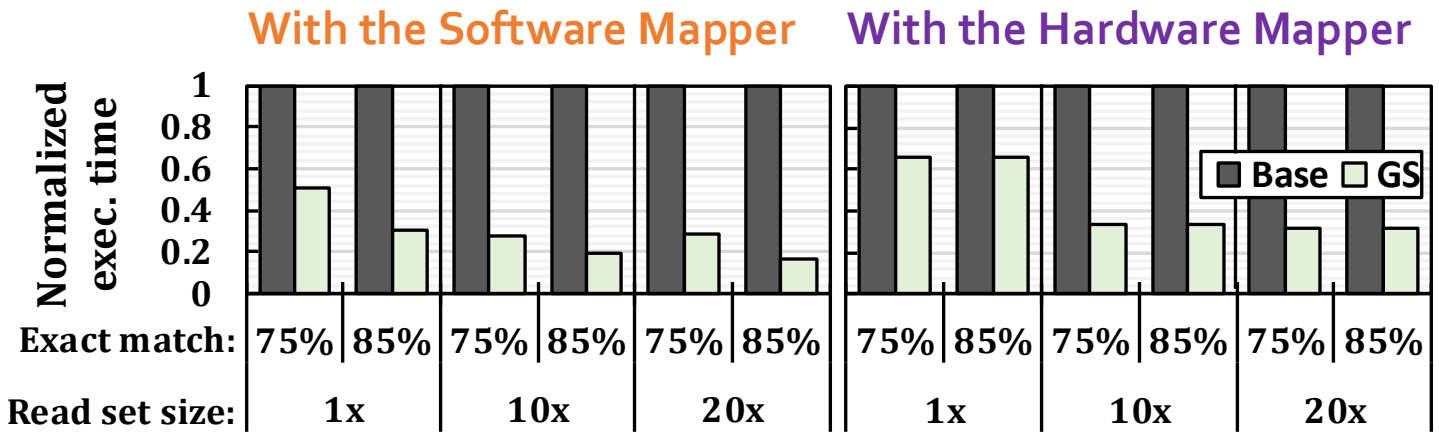
GenStore-NM



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

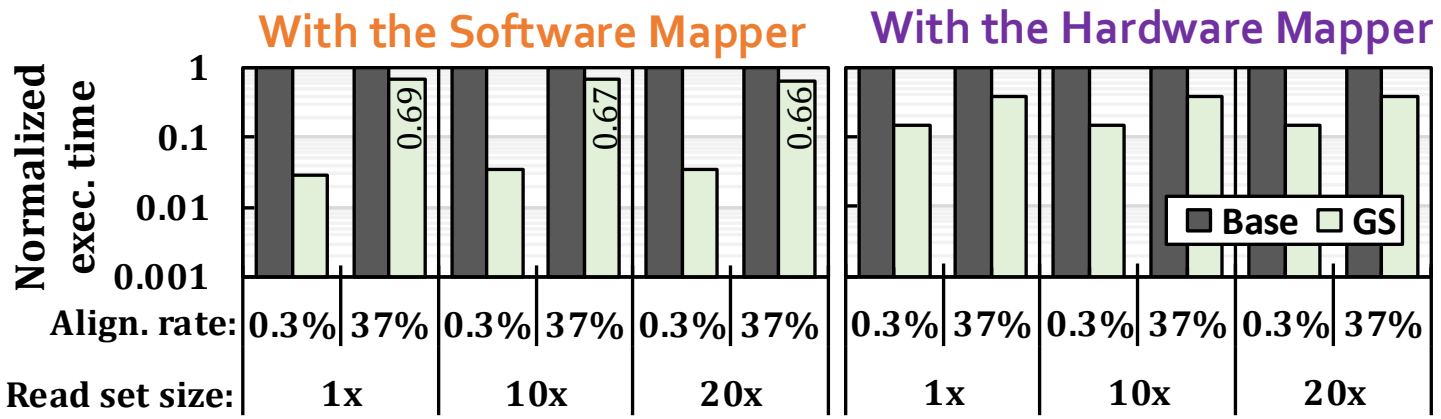
Effect of Inputs on GenStore-EM

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



Effect of Inputs on GenStore-NM

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



MegIS

High-Performance, Energy-Efficient, and Low-Cost Metagenomic Analysis with In-Storage Processing

Nika Mansouri Ghiasi

Mohammad Sadrosadati Harun Mustafa Arvid Gollwitzer Can Firtina

Julien Eudine Haiyu Mao Joël Lindegger Meryem Banu Cavlak

Mohammed Alser Jisung Park Onur Mutlu

SAFARI

ETH zürich POSTECH

MegIS Discussion

- Main presentation
- Detailed motivation
- Overview of MegIS's steps
- MegIS FTL
- Multi-sample analysis
- SSD configurations
- Detailed results

Outline

Background

Motivation and Goal

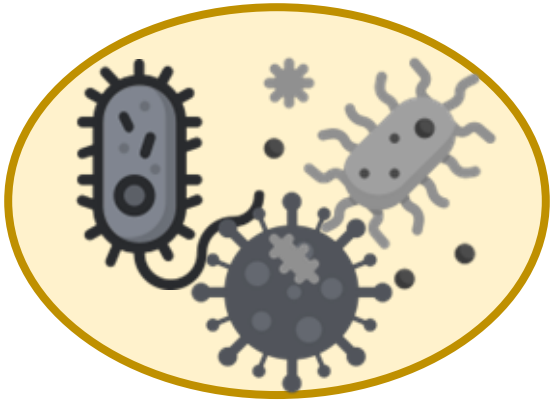
MegIS

Evaluation

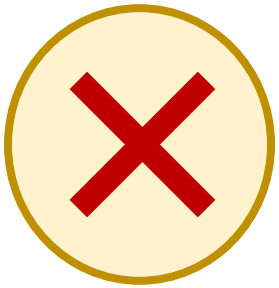
Conclusion

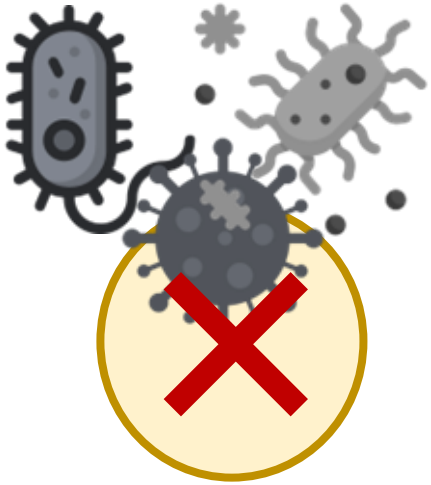
What is Metagenomics?

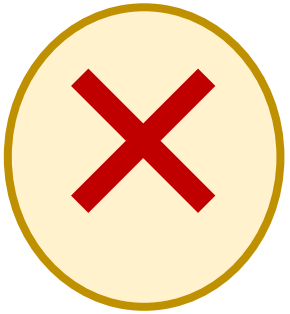
- ***Metagenomics:*** Study of genome sequences of **diverse organisms** within a **shared environment** (e.g., blood, ocean, soil)



- **Overcomes the limitations of traditional genomics**
  - Bypasses the need for analyzing individual species in isolation

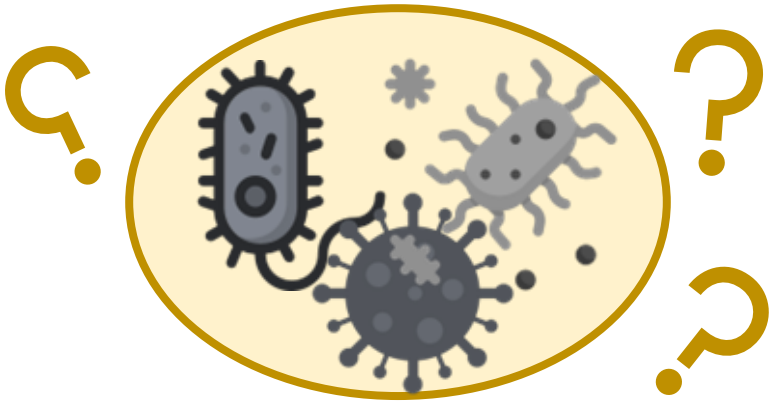






What is Metagenomics?

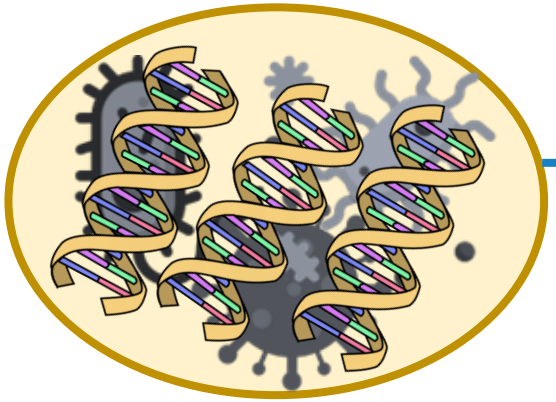
- ***Metagenomics:*** Study of genome sequences of **diverse organisms** within a **shared environment** (e.g., blood, ocean, soil)



Has led to groundbreaking advances

- Precision medicine
- Understanding microbial diversity of an environment
- Discovering early warnings of communicable diseases

Metagenomic Analysis



Metagenomic sample with species that are **not known** in advance

A large database containing information on **many species**

Preparation of Input Queries

Query K-mers

GCTCA

CTCAT

TCATG

...

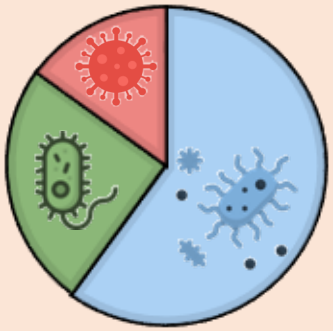
Presence/Absence Identification

V. cholerae

SARS-CoV-2

E. coli

Abundance Estimation



(e.g., > 100 TBs in emerging databases)

Outline

Background

Motivation and Goal

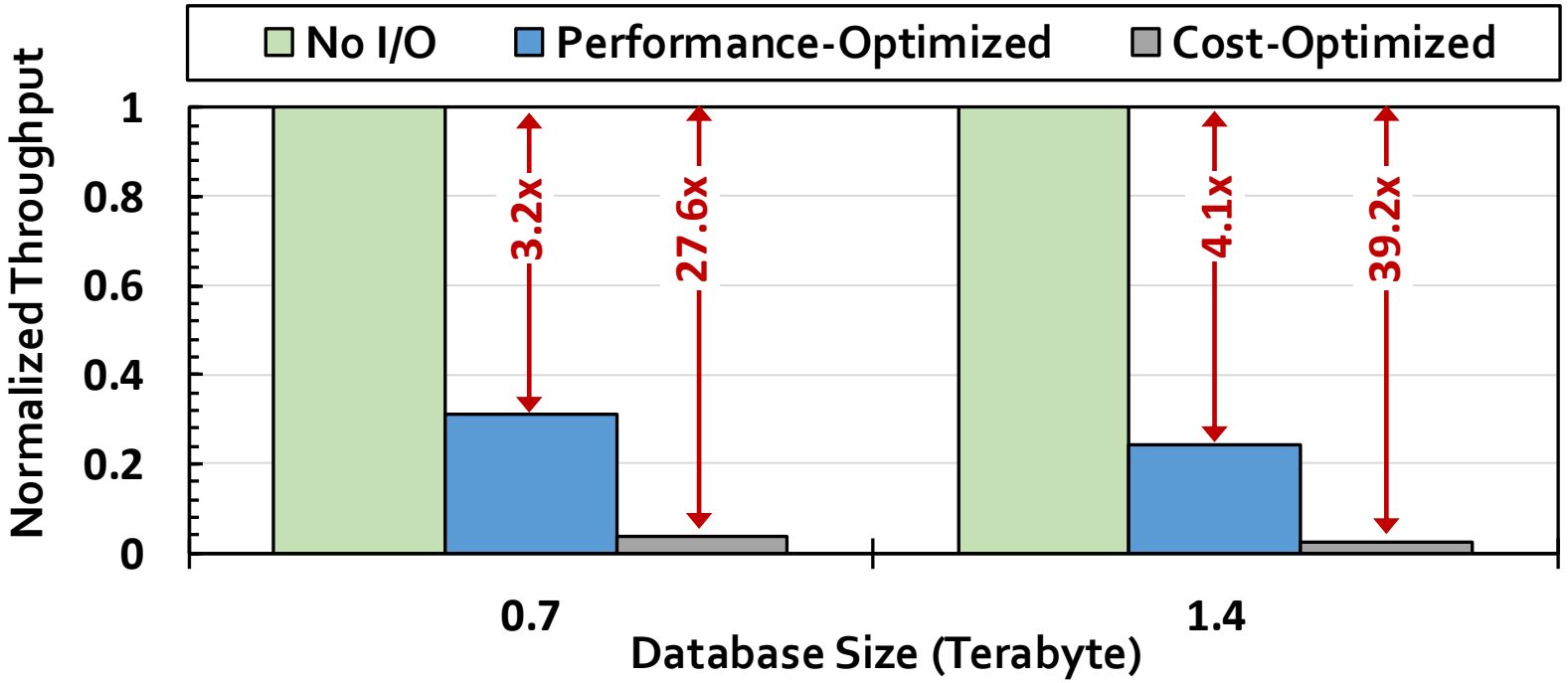
MegIS

Evaluation

Conclusion

Motivation

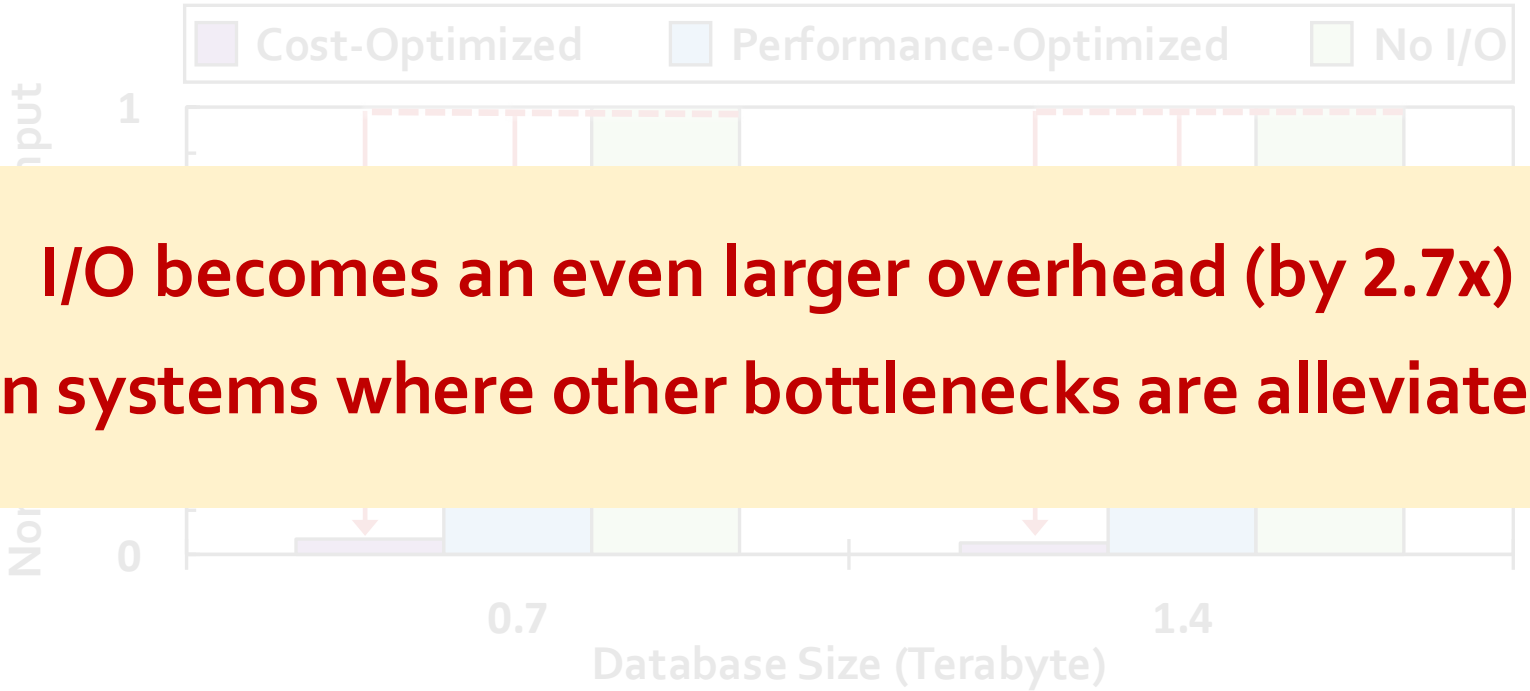
- Case study of the performance of metagenomic analysis tools
- With various state-of-the-art SSD configurations



I/O data movement causes significant performance overhead

Motivation

- Case study on the throughput of metagenomic analysis tools
- With Various state-of-the-art SSD configurations



I/O becomes an even larger overhead (by 2.7x) in systems where other bottlenecks are alleviated

I/O data movement causes significant performance overhead

I/O Overhead is Hard to Avoid

I/O overhead due to accessing **large**, **low-reuse** data is hard to avoid

Sampling techniques to shrink database sizes

[Wood+, Genome Biology'19], [Ounit+, BMC Genomics'15], [Kim+, Genome Research'16], …

✗ *Reduce accuracy to levels unacceptable for many use cases*

Keeping all data required by metagenomic analysis completely and always resident in main memory

✗ *Energy inefficient, costly, unscalable, and unsustainable*

- Database sizes **increase rapidly** (doubling every few months)
- Different analyses need **different databases**

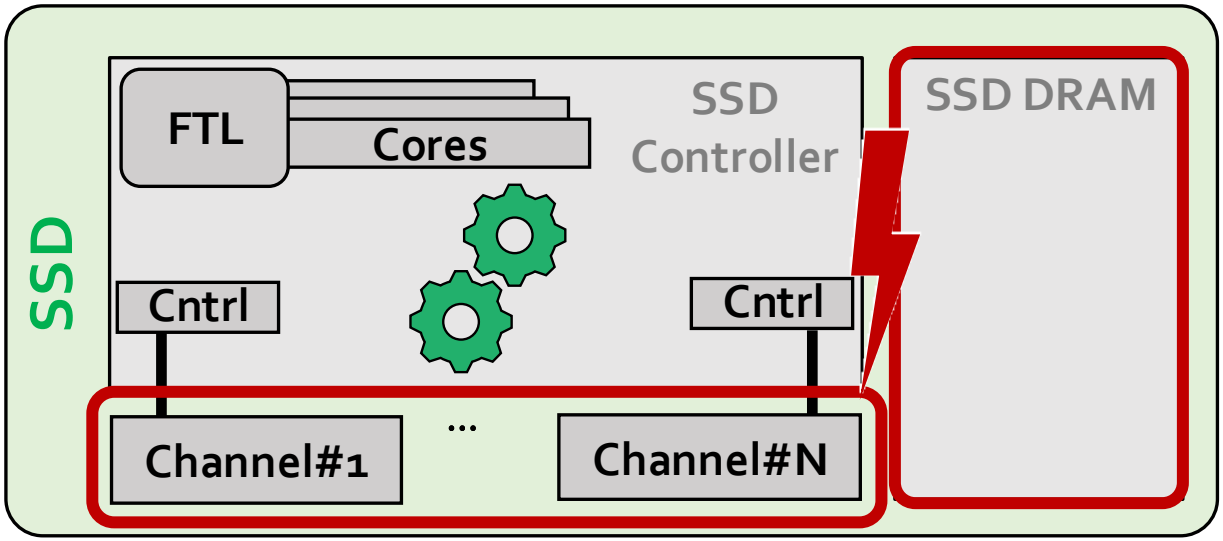
Our Goal

Improve metagenomic analysis ***performance***
by reducing large ***data movement overhead***
from the storage system
in a ***cost-effective*** *manner and with* ***high accuracy***

Challenges of In-Storage Processing

No metagenomic analysis tools can run in-storage due to SSD limits

- Long **latency of NAND flash** chips
- Limited **DRAM capacity** inside the SSD
- Limited **DRAM bandwidth** inside the SSD



Outline

Background

Motivation and Goal

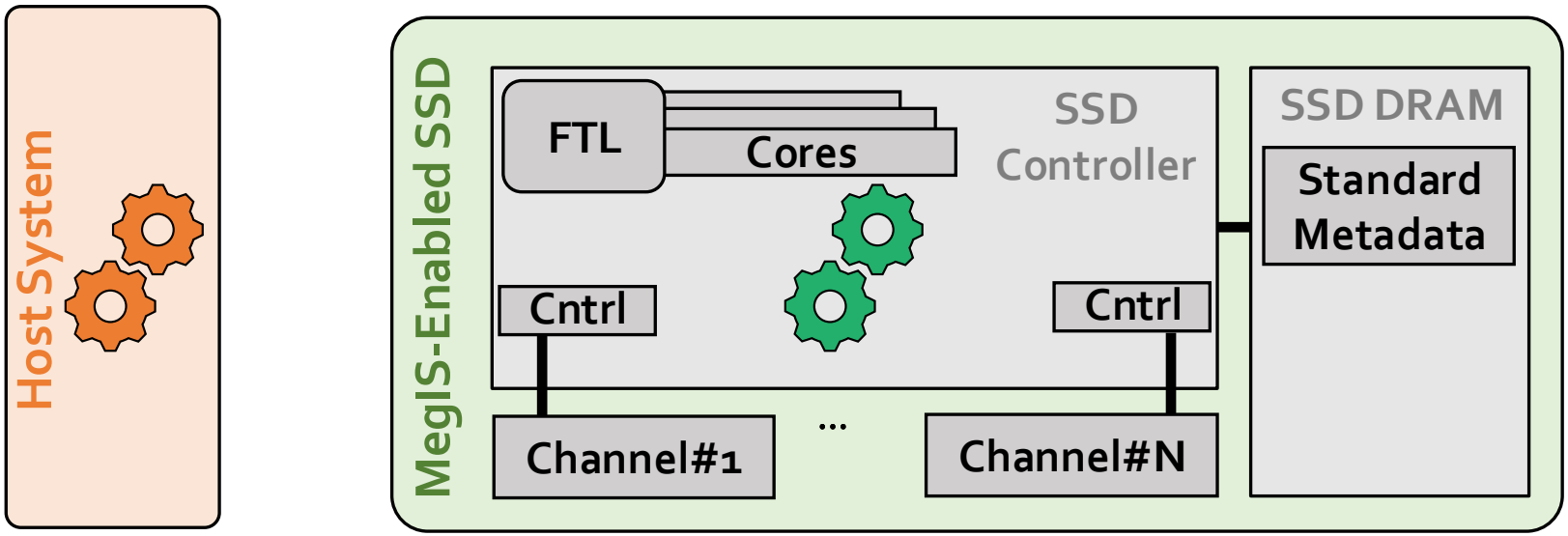
MegIS

Evaluation

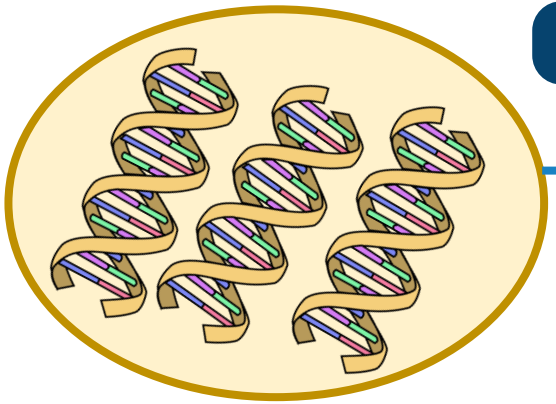
Conclusion

MegIS: Metagenomics In-Storage

- First in-storage system for *end-to-end* metagenomic analysis
- **Idea:** Cooperative in-storage processing for metagenomic analysis
  - Hardware/software co-design between



MegIS's Steps



Metagenomic sample with species that are **not known** in advance

A large database containing information on **many species**

Step 1

Preparation of Input Queries

Query K-mers: GCTCA, CTCAT, TCATG, ...

Step 2

Presence/Absence Identification

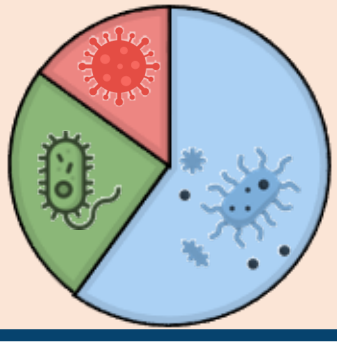
V. cholerae

SARS-CoV-2

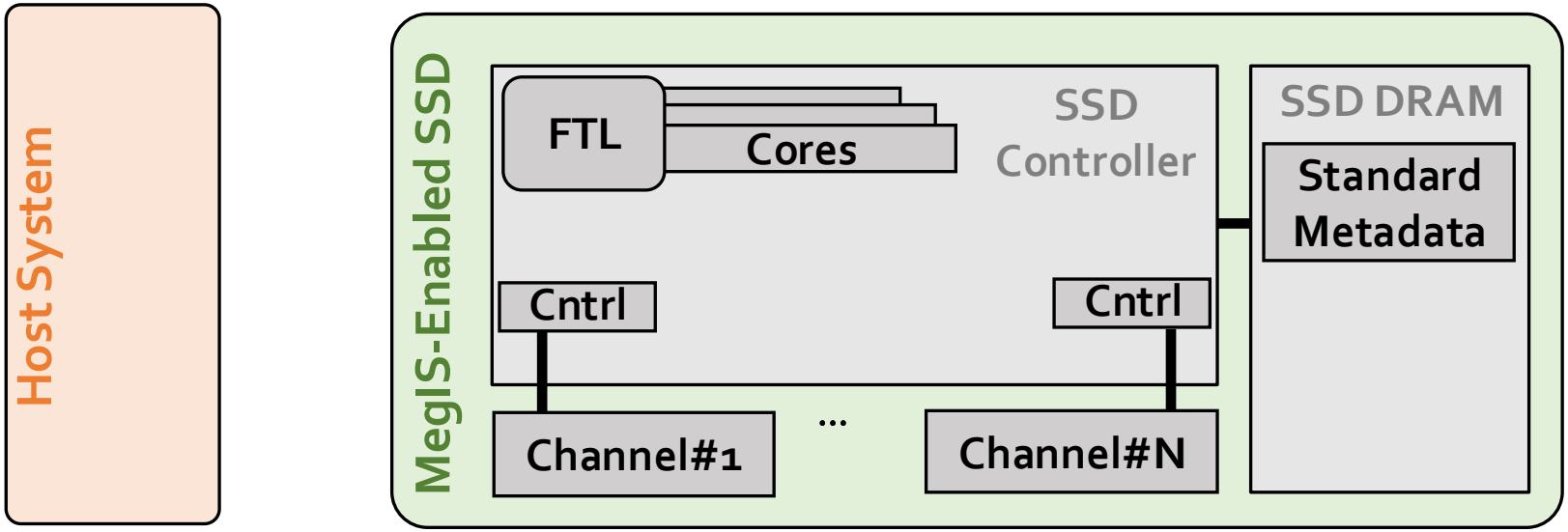
E. coli

Step 3

Abundance Estimation



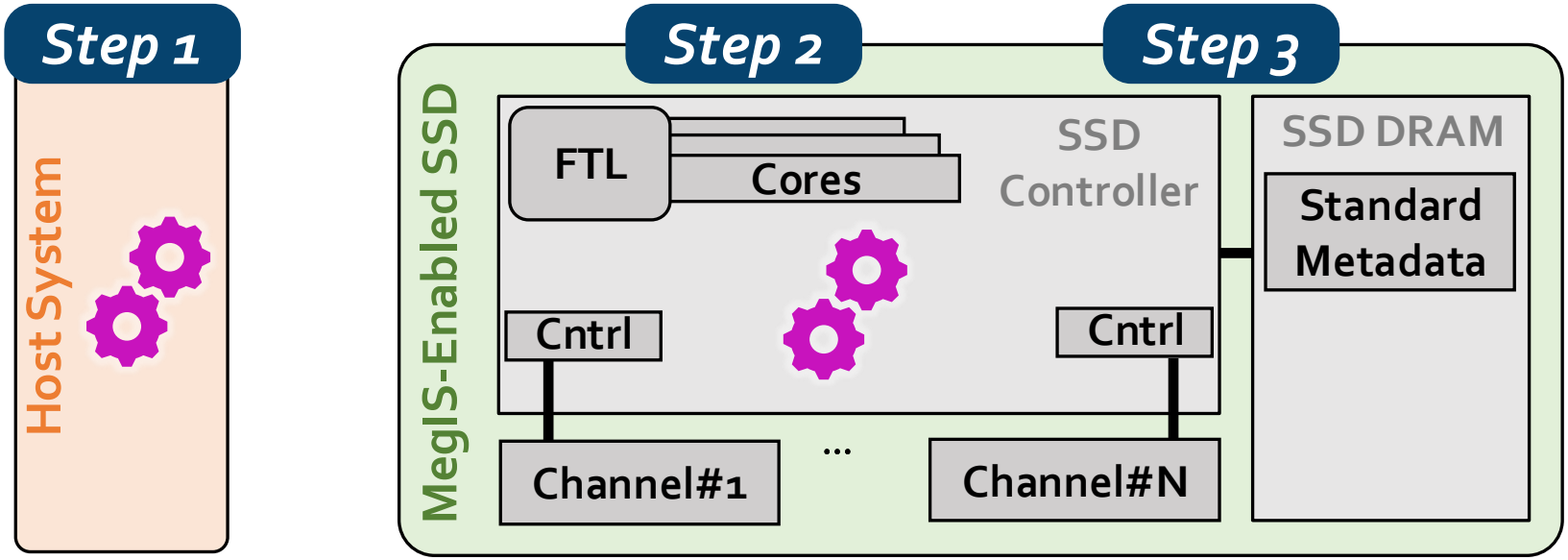
MegIS Hardware-Software Co-Design



MegIS Hardware-Software Co-Design

Task partitioning and mapping

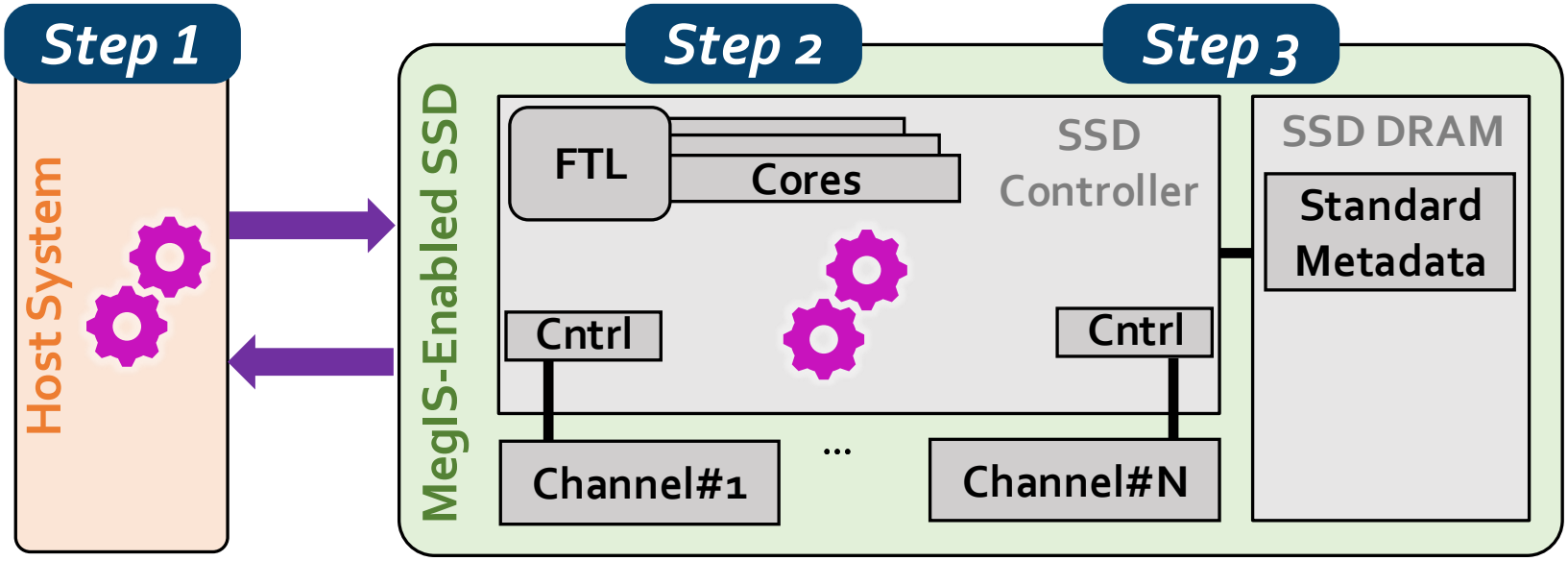
- *Each step executes in its most suitable system*



MegIS Hardware-Software Co-Design

Task partitioning and mapping
- *Each step executes in its most suitable system*

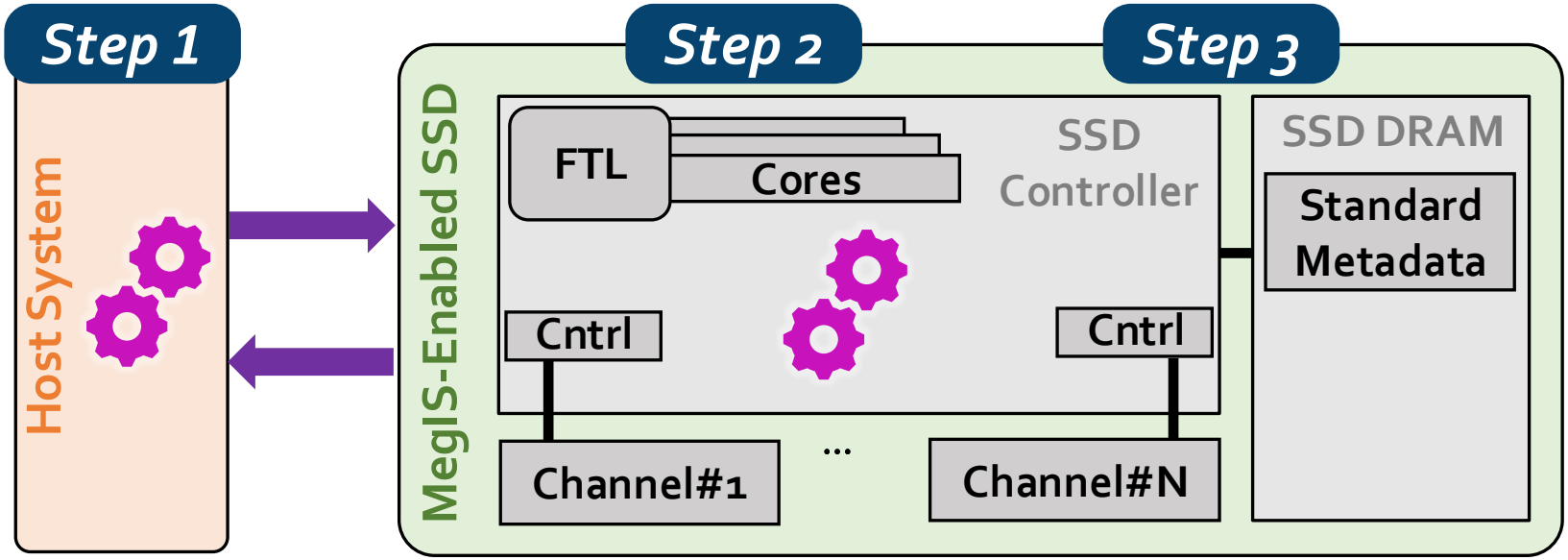
Data/computation flow coordination
- *Reduce communication overhead*
- *Reduce #writes to flash chips*



MegIS Hardware-Software Co-Design

Task partitioning and mapping
- *Each step executes in its most suitable system*

Data/computation flow coordination
- *Reduce communication overhead*
- *Reduce #writes to flash chips*

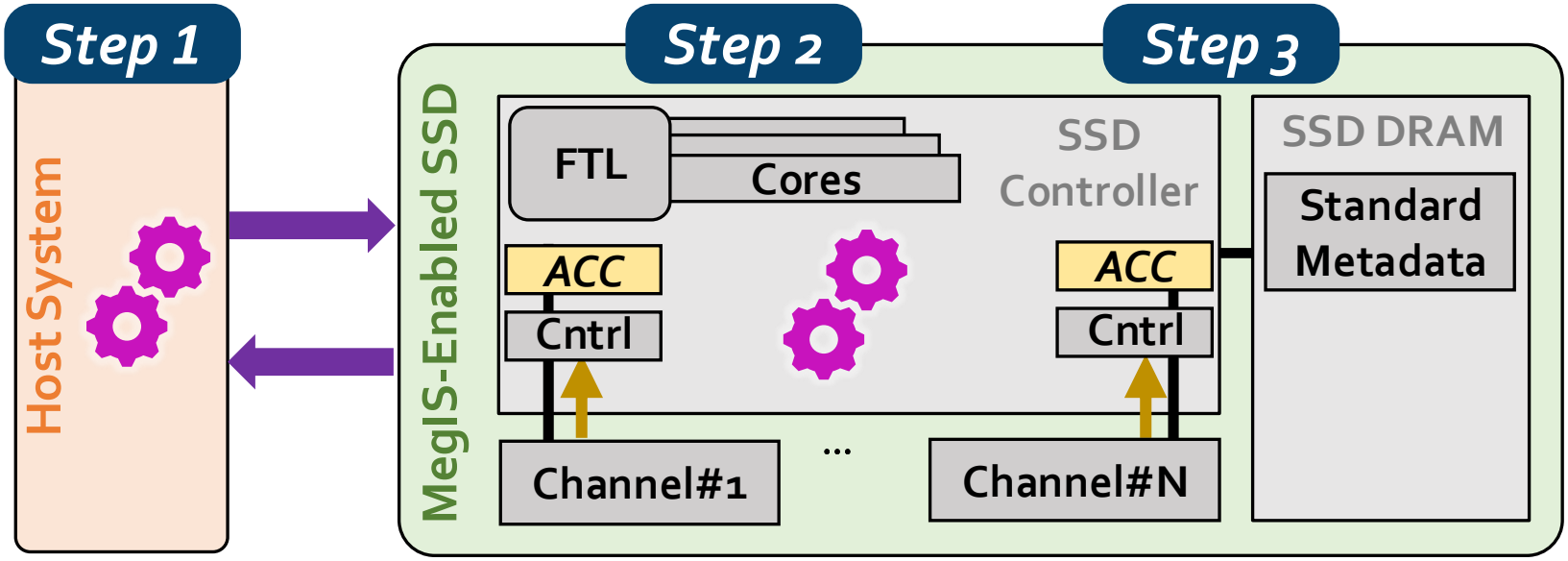


Storage-aware algorithms
- *Enable efficient access patterns to the SSD*

MegIS Hardware-Software Co-Design

Task partitioning and mapping
- *Each step executes in its most suitable system*

Data/computation flow coordination
- *Reduce communication overhead*
- *Reduce #writes to flash chips*



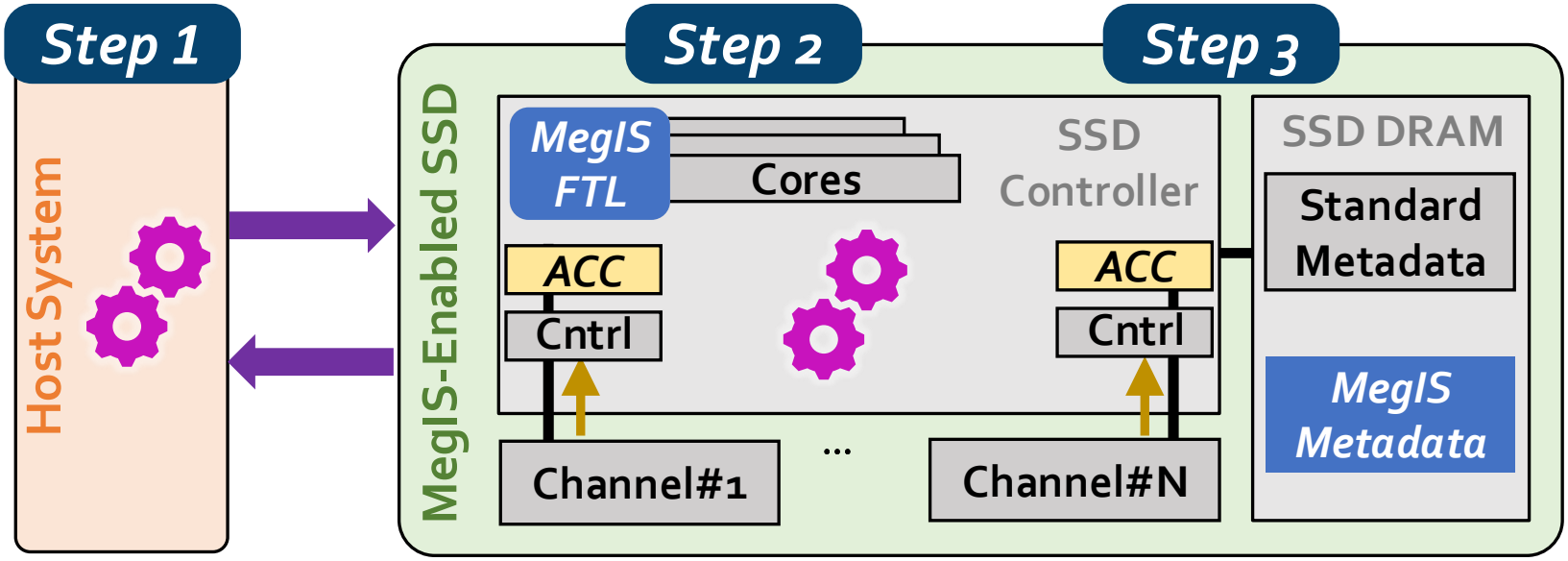
Storage-aware algorithms
- *Enable efficient access patterns to the SSD*

Lightweight in-storage accelerators
- *Minimize SRAM/DRAM buffer spaces needed inside the SSD*

MegIS Hardware-Software Co-Design

Task partitioning and mapping
- *Each step executes in its most suitable system*

Data/computation flow coordination
- *Reduce communication overhead*
- *Reduce #writes to flash chips*

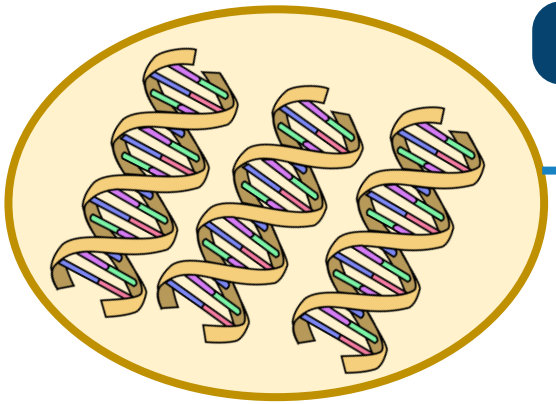


Storage-aware algorithms
- *Enable efficient access patterns to the SSD*

Lightweight in-storage accelerators
- *Minimize SRAM/DRAM buffer spaces needed inside the SSD*

Data mapping scheme and Flash Translation Layer (FTL)
- *Specialize to the characteristics of metagenomic analysis*
- *Leverage the SSD's full internal bandwidth*

Step 1 Overview



Metagenomic sample with species that are **not known** in advance

A large database containing information on **many species**

Step 1

Preparation of Input Queries

Query K-mers

GCTCA

CTCAT

TCATG

...

Step 2

Presence/Absence Identification

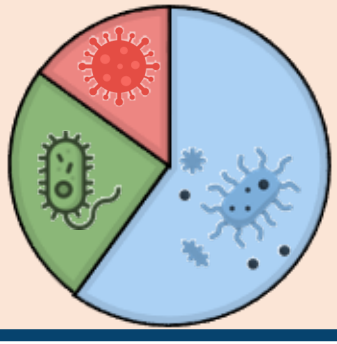
V. cholerae

SARS-CoV-2

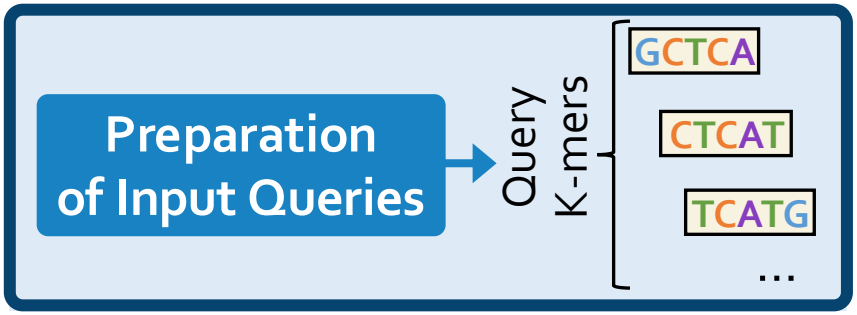
E. coli

Step 3

Abundance Estimation



Step 1 Overview



*MegIS employs **sorted data structures** to avoid expensive random accesses to the SSD*

- **Extract k-mers** from the sample
- **Sort** the k-mers (database is sorted offline)

MegIS executes Step 1 in the host system

- Benefits from **larger DRAM** and **more powerful computation**
- Incurs **fewer writes** to NAND flash chips (than processing this step in the SSD)
- Enables **overlapping** Step 1 with Step 2

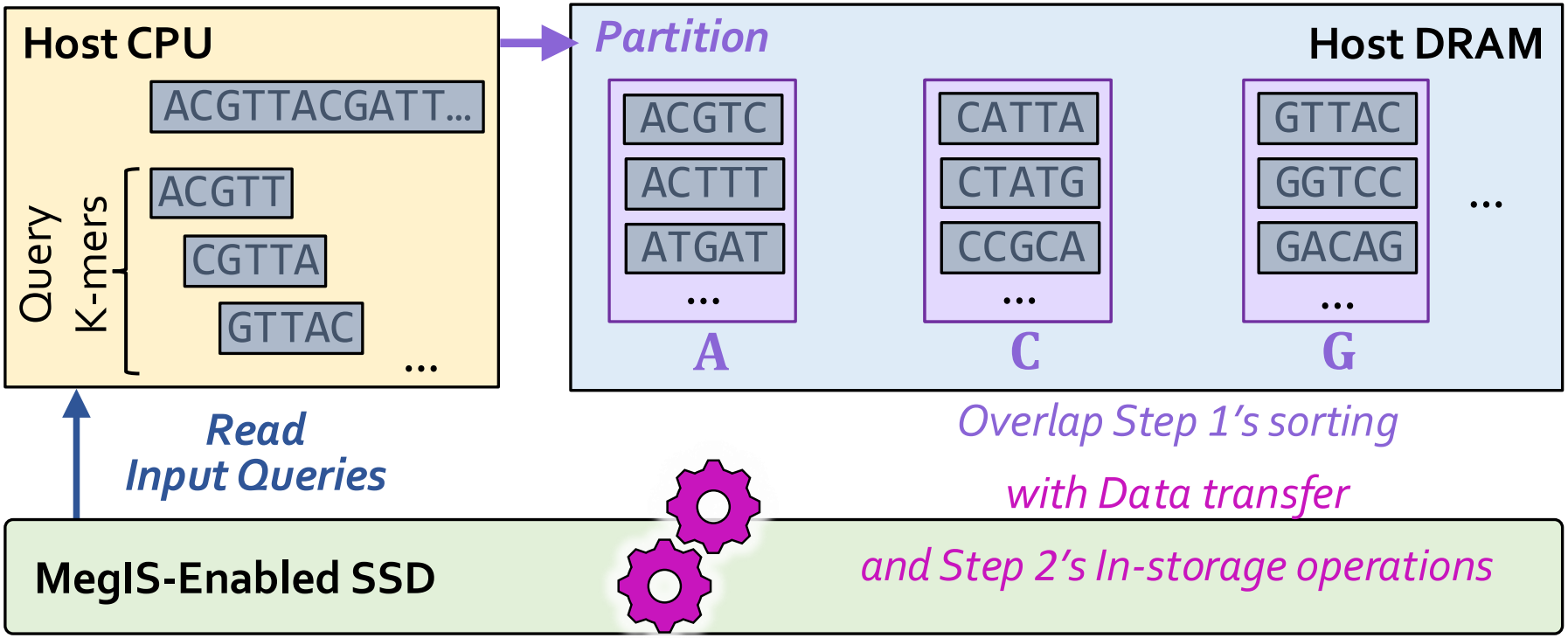
To execute Step 1 efficiently in the host system, MegIS needs to:

- Avoid significant overhead due to **data transfer time** between the steps
- Minimize **performance** and **lifetime** overheads *even* when host DRAM cannot hold all query k-mers

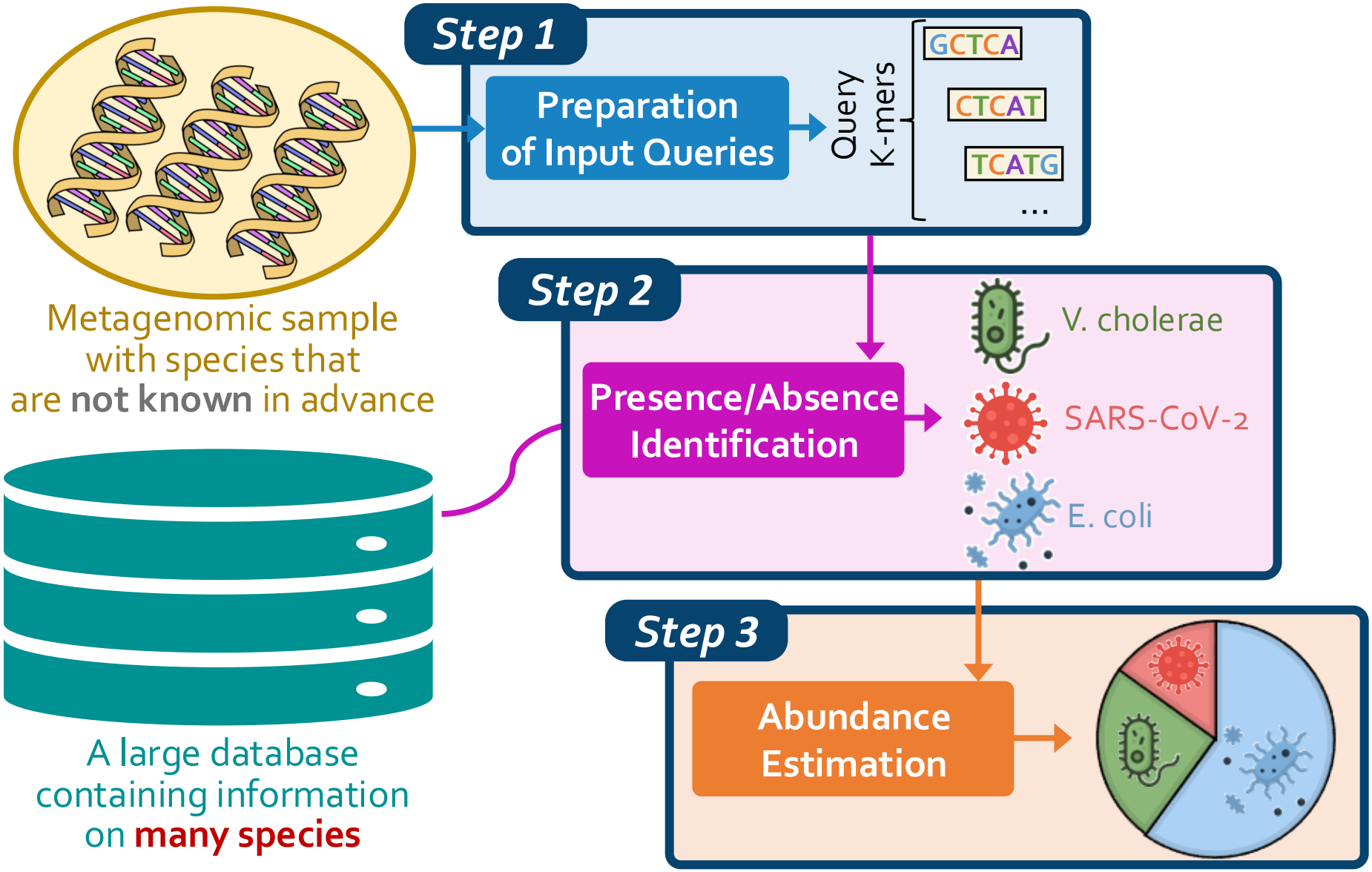
Step 1 Design

Divide k-mers into **independent partitions** by their alphabetical range

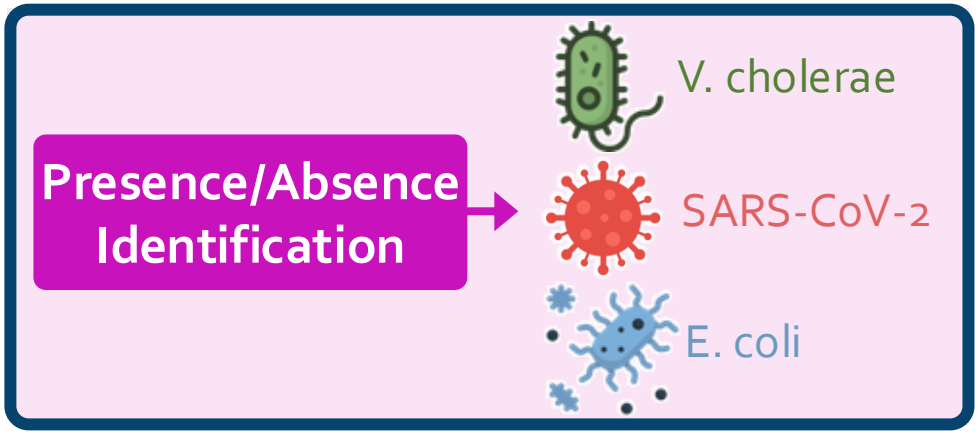
✓ **Can overlap operations on different partitions**



Step 2 Overview



Step 2 Overview



- **Identify the common k-mers** between the query k-mers and the database k-mers
- **Retrieve the species IDs** of the common k-mers

MegIS executes Step 2 in the SSD

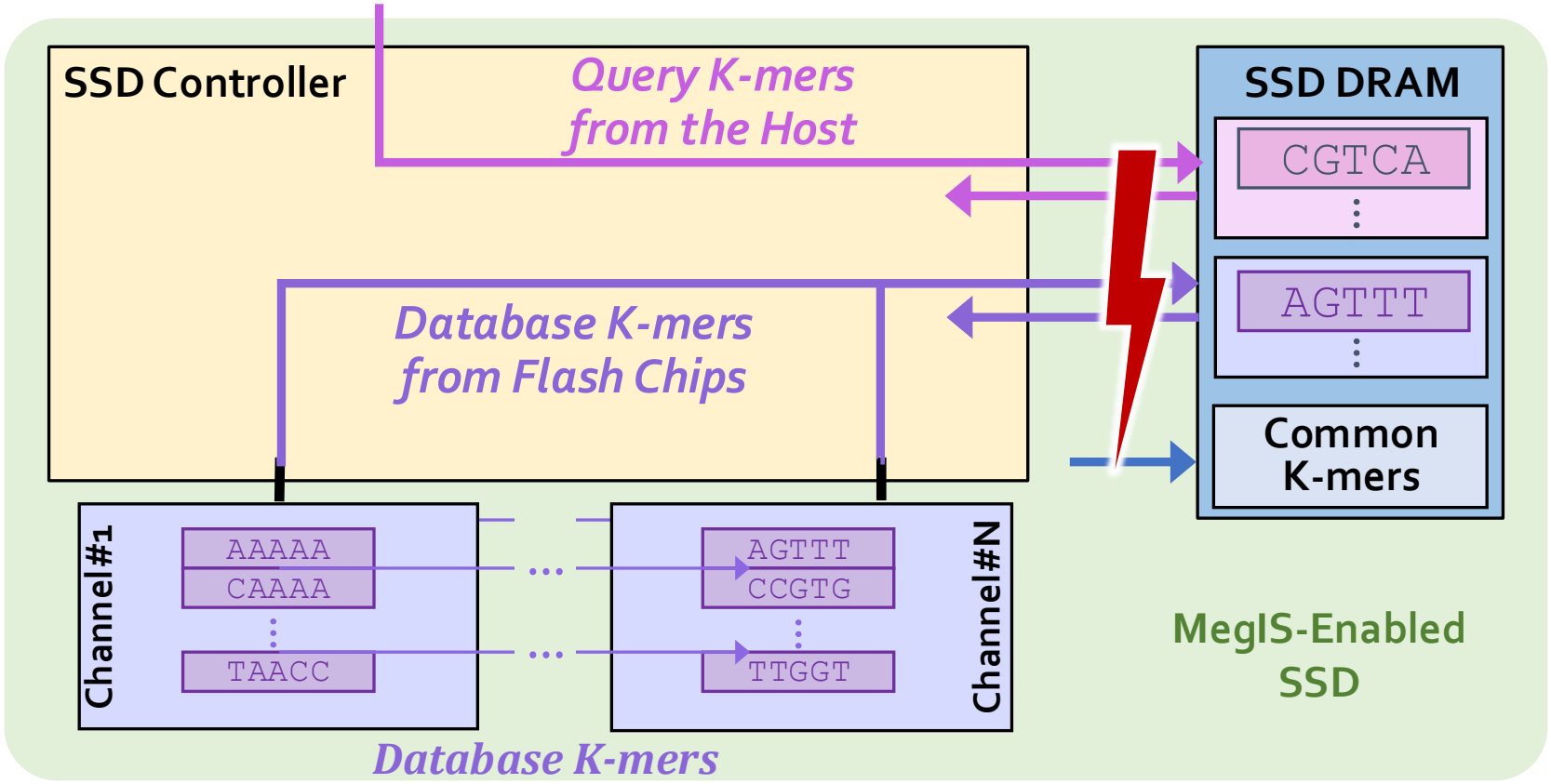
- Accesses **large data** with **low reuse**
- Involves **lightweight computation**

To execute Step 2 efficiently in the SSD, MegIS needs to:

- Leverage **internal bandwidth** efficiently
- Not require **expensive hardware inside the SSD** (e.g., large DRAM bandwidth/capacity and costly logic units)

Step 2 Design: Identifying the Common K-mers

- **Challenge:** Limited internal DRAM bandwidth

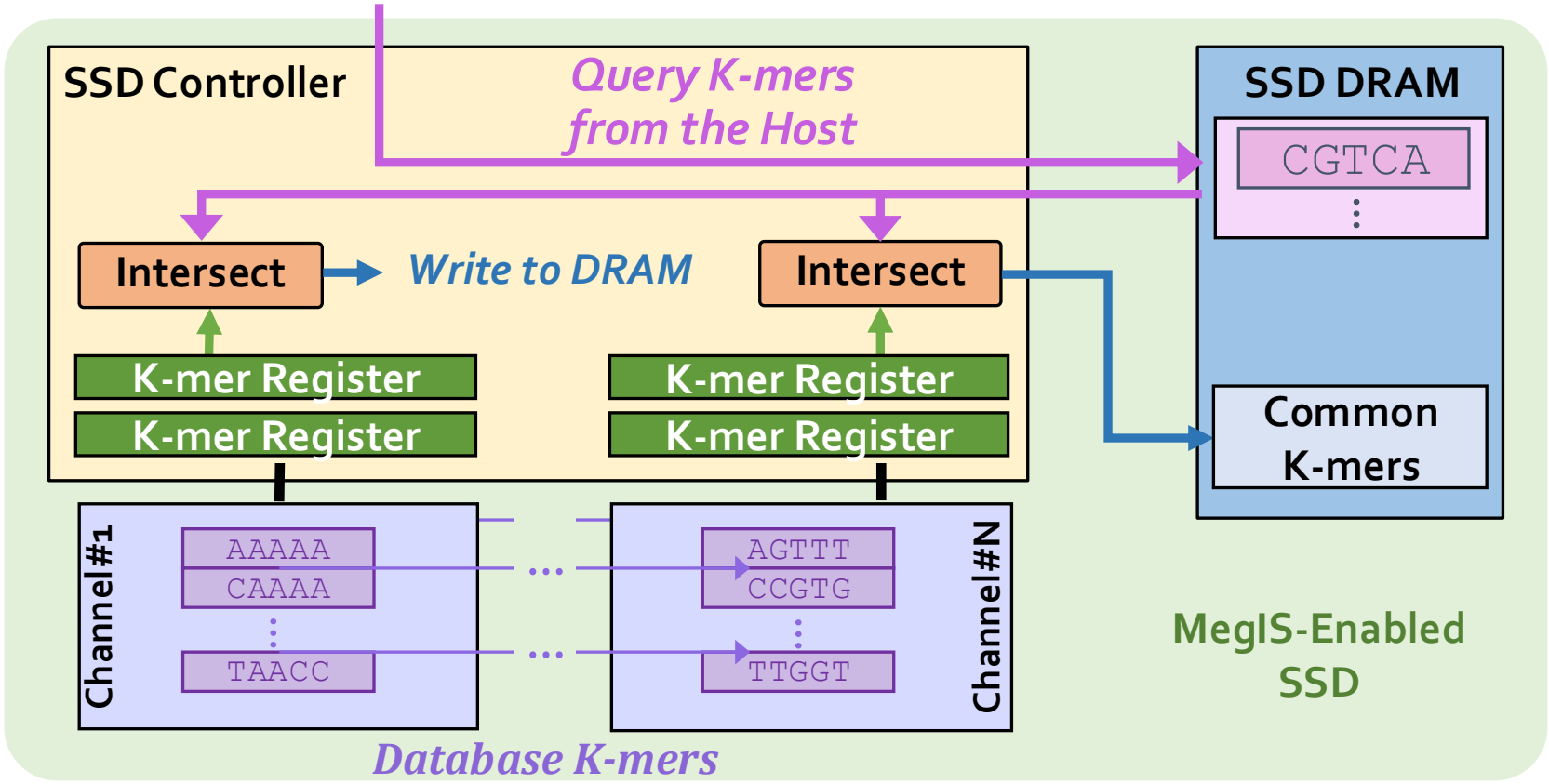


Step 2 Design: Identifying the Common K-mers

- **Challenge:** Limited internal DRAM bandwidth

✓ *Compute directly on the flash data streams* [Zou+, MICRO'22]

✓ *Reduce buffer size based on application features*

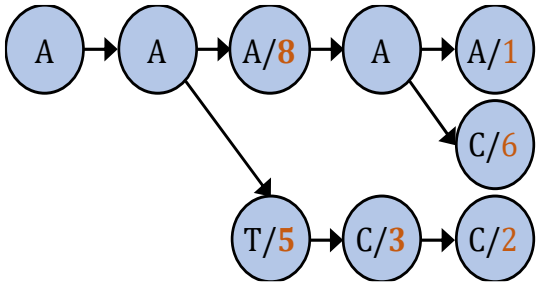


Step 2 Design: Retrieving the Species ID

- MegIS retrieves the species IDs of the **common k-mers** by looking up a **sketch database**

K-mer	
AAAAA	
AAAAC	
AATCC	
…	

Space-Inefficient



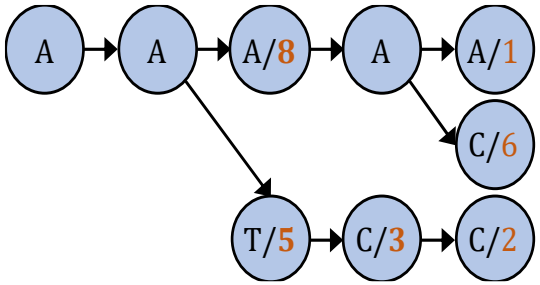
Space-Efficient

✗ ***Slow inside the SSD due to long NAND flash latency***

Step 2 Design: Retrieving the Species ID

- MegIS retrieves the species IDs of the **common k-mers** by looking up a **sketch database**

K-mer	ID
AAAAA	1,5
AAAAC	6
AATCC	2, 9
...	...



K-mer Sketch Streaming is much more suitable for in-storage processing due to its streaming accesses

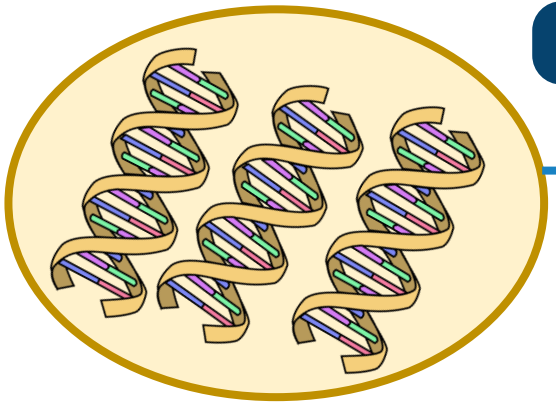
Step 2 Design: Retrieving the Species ID

- MegIS retrieves the species IDs of the **common k-mers** by looking up a **sketch database**

Design details are in the paper

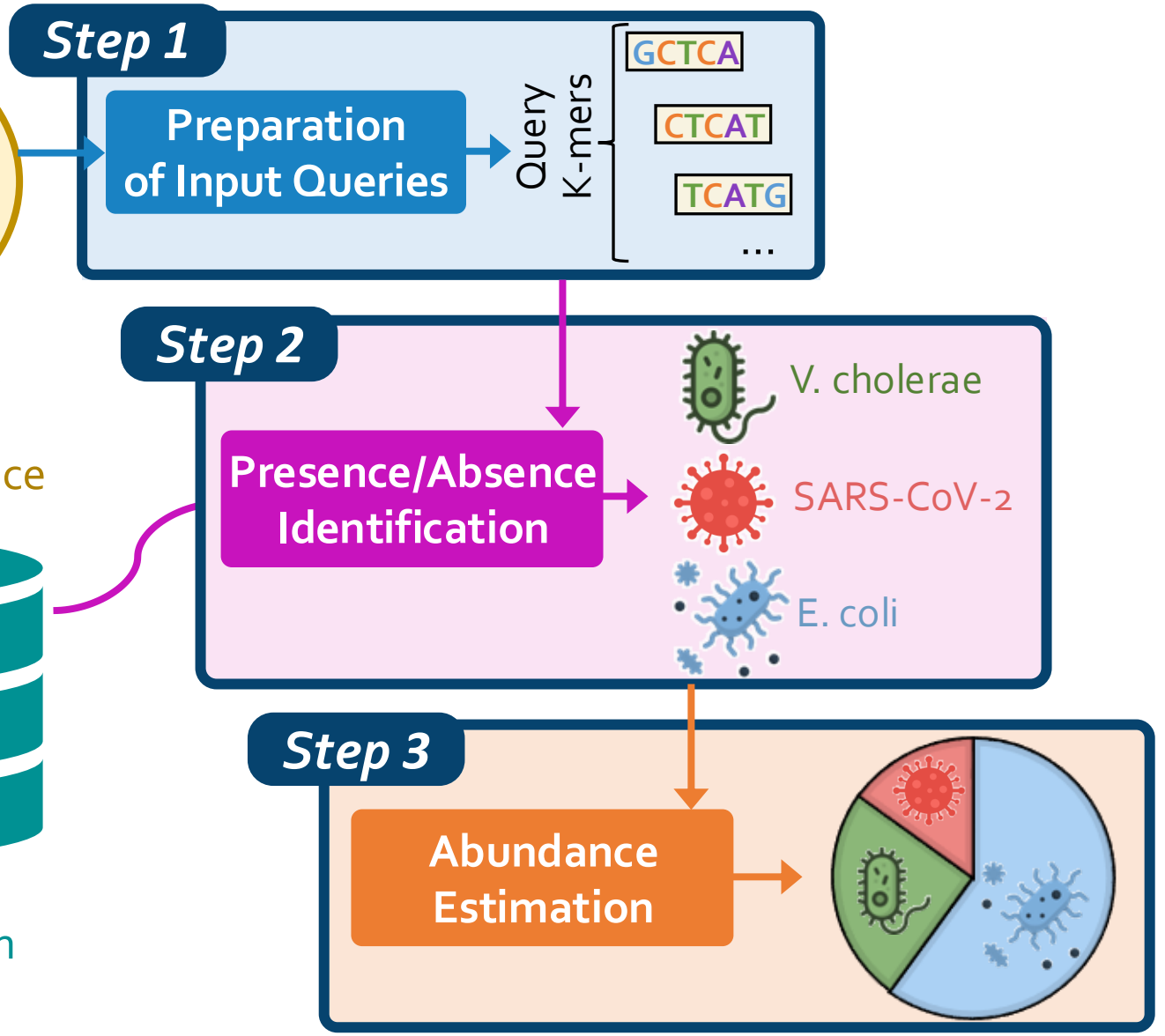
K-mer Sketch Streaming is much more suitable for in-storage processing due to its streaming accesses

Step 3

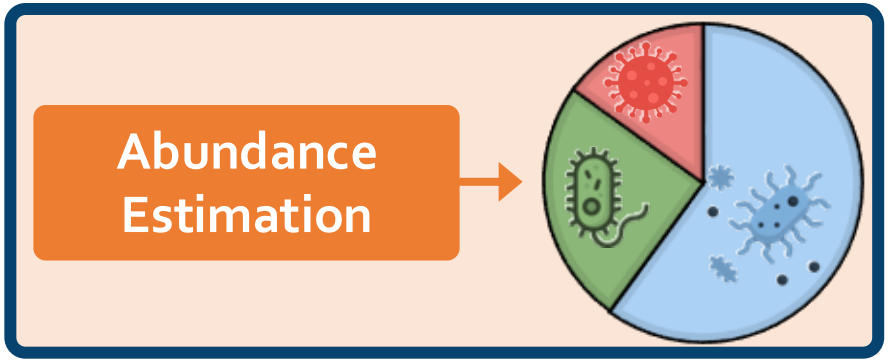


Metagenomic sample with species that are **not known** in advance

A large database containing information on **many species**



Step 3



MegIS performs additional analysis on species identified in the sample to estimate their abundance

MegIS can flexibly integrate with different approaches

1. **Lightweight statistical approaches**: Directly uses the output of Step 2
2. **More accurate and costly read mapping:** MegIS facilitates integration by preparing mapping indexes in the SSD

K-mer	Loc.
ATT	14
CCA	9
GCT	5
...	...

Reference Index Organism A

K-mer	Loc.
AAG	2
CCA	21
TGC	4
...	...

Reference Index Organism B

Merge

K-mer	Loc.
AAG	1002
ATT	14
CCA	9, 1021
GCT	5
TGC	1004
...	...

Unified Reference Index

Step 3 and MegIS FTL are in the paper

Outline

Background

Motivation and Goal

MegIS

Evaluation

Conclusion

Evaluation Methodology Overview (I)

Performance, Energy, and Power Analysis

Hardware Components

- Synthesized Verilog model for the in-storage accelerators
- MQSim *[Tavakkol+, FAST'18]* for SSD's internal operations
- Ramulator *[Kim+, CAL'15]* for SSD's internal DRAM

Software Components

Measure on a real system:

- AMD® EPYC® CPU with 128 physical cores
- 1-TB DRAM

Baseline Comparison Points

- **Performance-optimized software**, Kraken2 *[Genome Biology'19]*
- **Accuracy-optimized software**, Metalign *[Genome Biology'20]*
- **PIM hardware-accelerated tool** (using processing-in-memory), Sieve *[ISCA'21]*

SSD Configurations

- **SSD-C:** with SATA3 interface (0.5 GB/s sequential read bandwidth)
- **SSD-P:** with PCIe Gen4 interface (7 GB/s sequential read bandwidth)

Evaluation Methodology Overview (II)

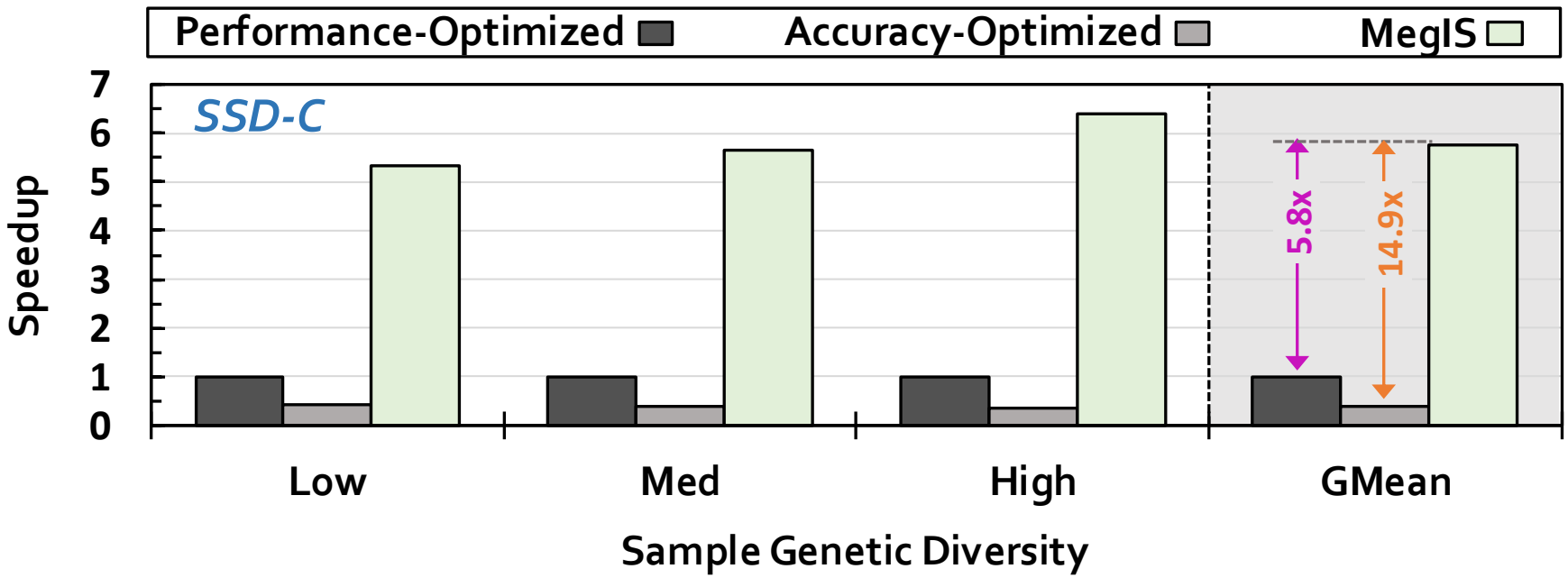
Metagenomic Analysis Task

- Finding species present in the sample
- Analysis of the abundance estimation task is in the paper

Metagenomic Samples

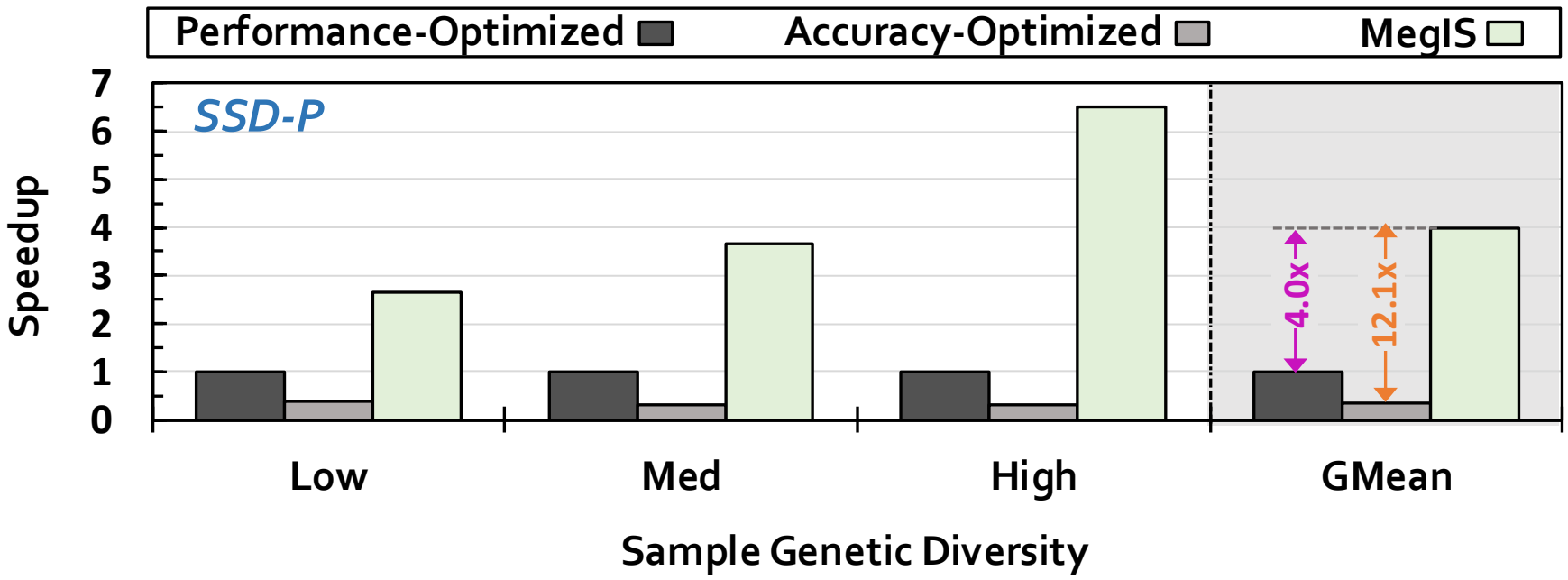
- With varying degrees of genetic diversity
  - Low
  - Medium
  - High

Speedup over Software (with Cost-Optimized SSD)



MegIS provides significant speedup over both Performance-Optimized and Accuracy-Optimized baselines

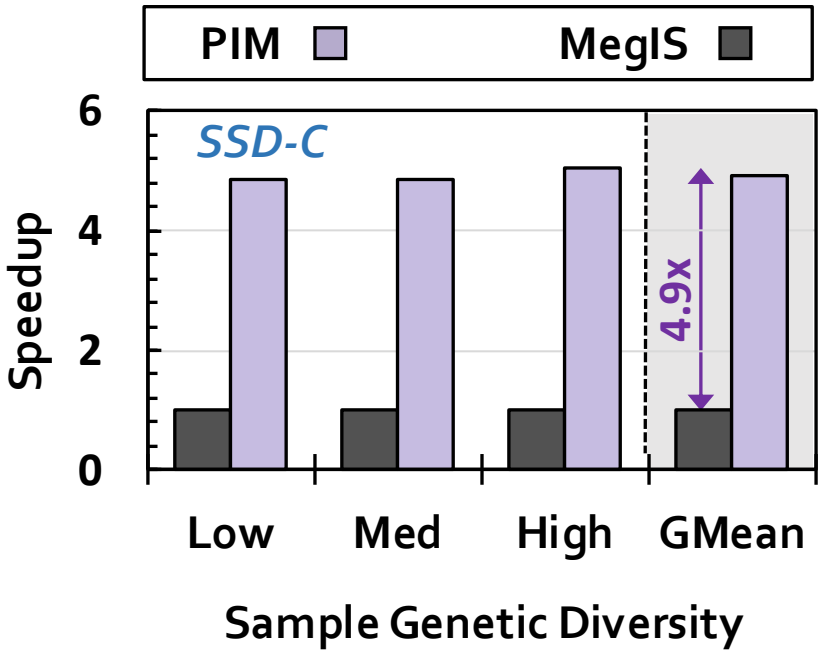
Speedup over Software (with Performance-Optimized SSD)

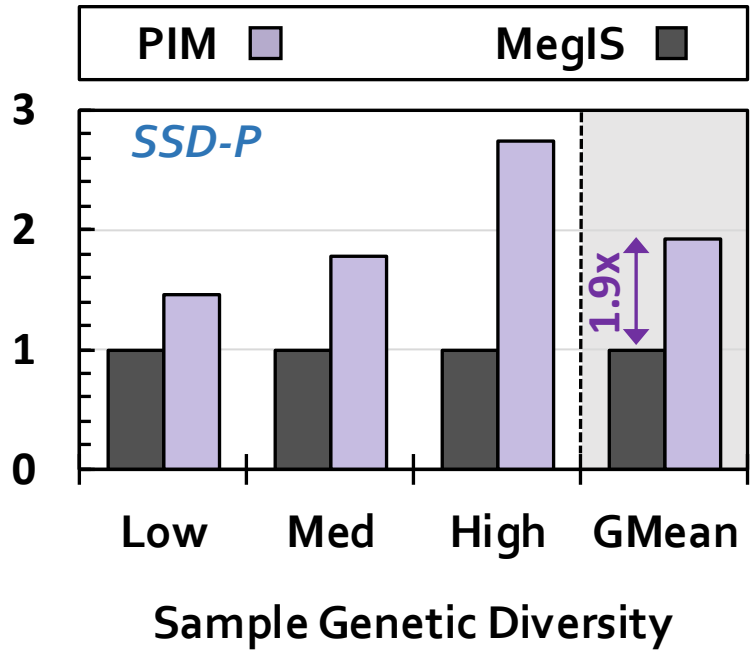


MegIS provides significant speedup over both Performance-Optimized and Accuracy-Optimized baselines

MegIS improves performance on both cost-optimized and performance-optimized SSDs

Speedup over the PIM Hardware Baseline

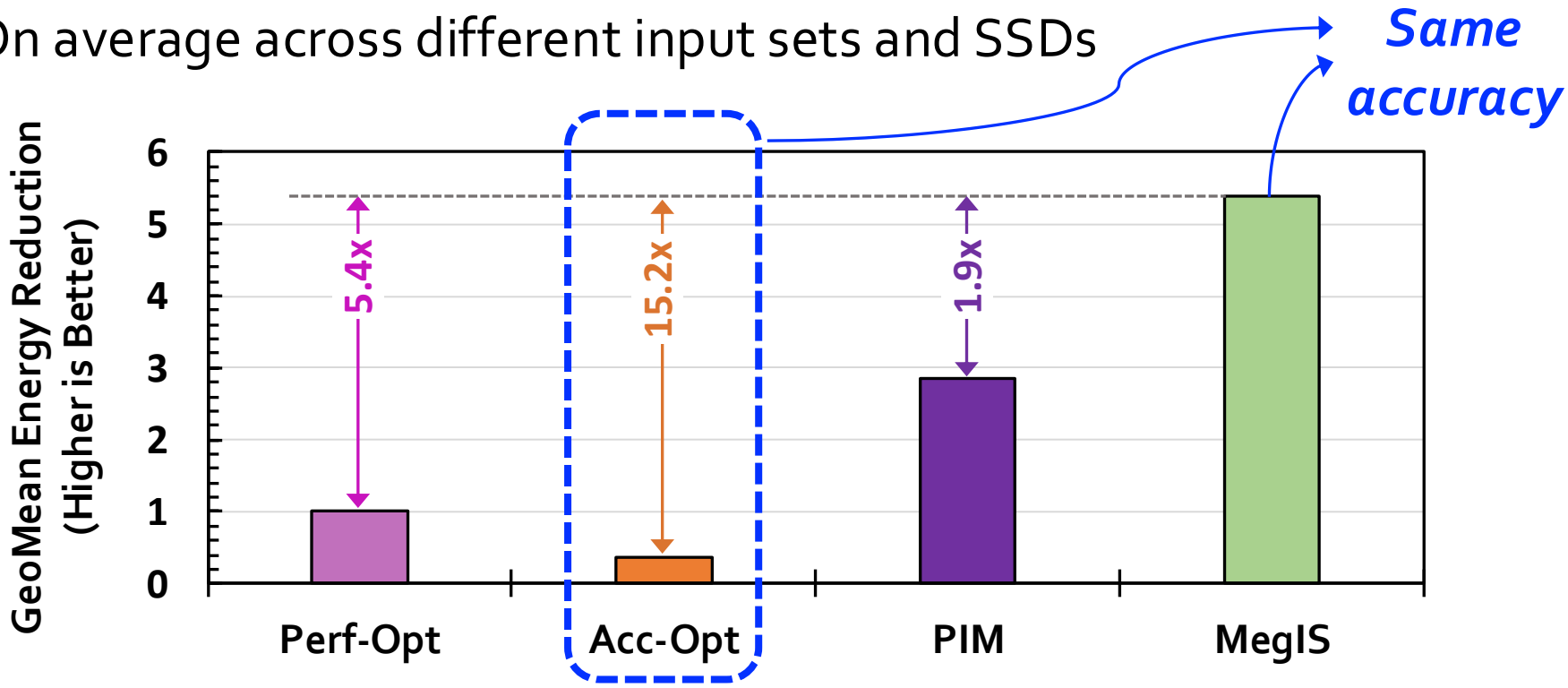




MegIS provides significant speedup over the PIM baseline

Reduction in Energy Consumption

- On average across different input sets and SSDs



MegIS provides significant energy reduction over the Performance-Optimized, Accuracy-Optimized, and PIM baselines

Accuracy, Area, and Power

Accuracy

- **Same accuracy** as the **accuracy-optimized** baseline
- **Significantly higher accuracy** than the **performance-optimized** and **PIM** baselines
  - 4.6 – 5.2× higher F1 score
  - 3 – 24% lower L1 norm error

Area and Power

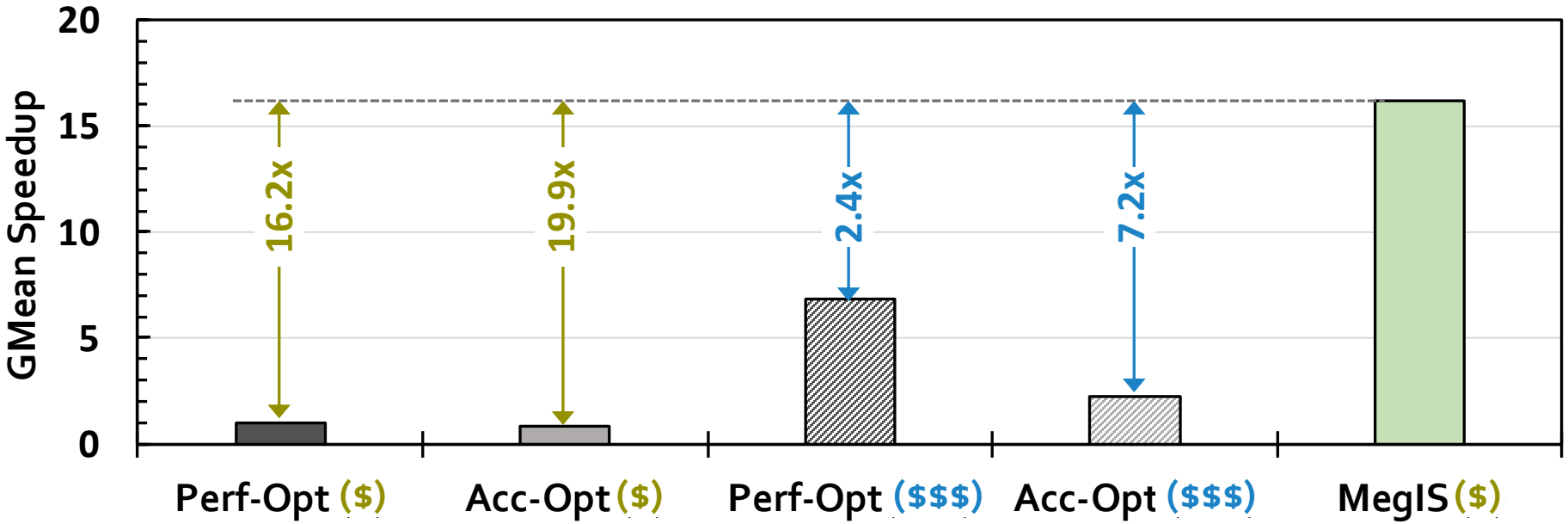
Total for an 8-channel SSD:

- **Area:** 0.04 mm^2
- **Power:** 7.658 mW

*(Only **1.7%** of the area and **4.6%** of the power consumption of three ARM Cortex R4 cores in an SSD controller)*

System Cost-Efficiency

- **Cost-optimized system ($):** With SSD-C and 64-GB DRAM
- **Performance-optimized system ($$$):** With SSD-P and 1-TB DRAM

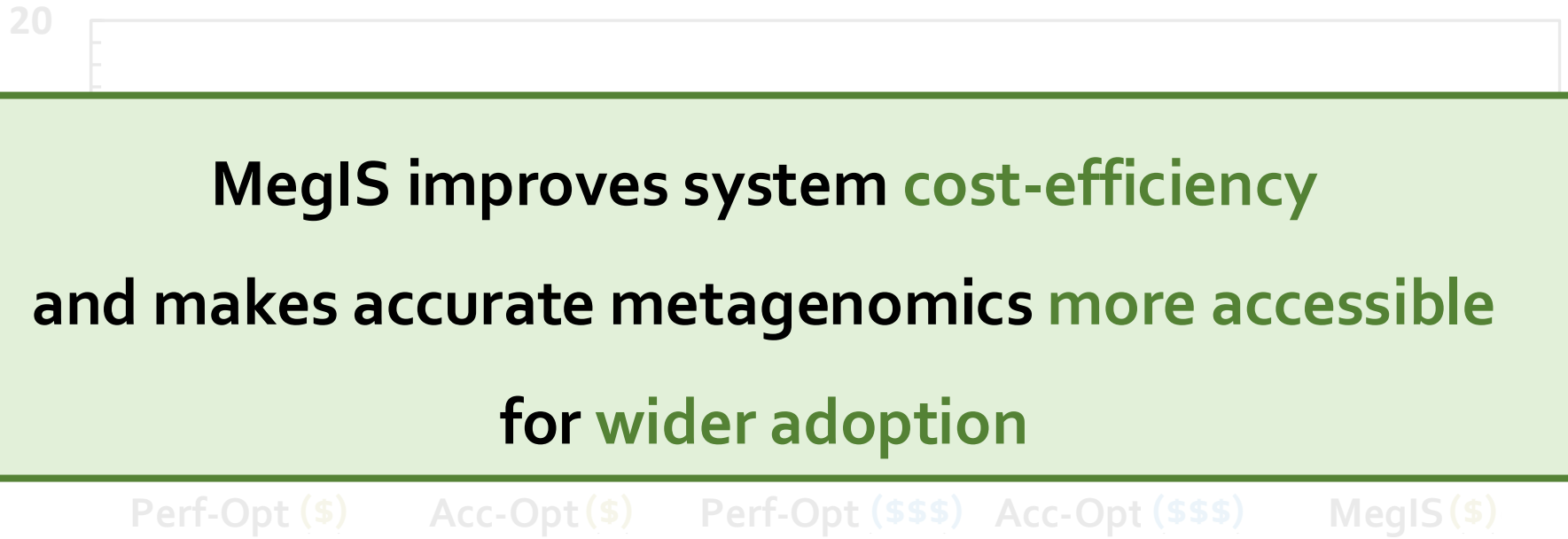


MegIS outperforms the baselines

***even* when running on a much less costly system**

System Cost-Efficiency

- **Cost-optimized system ($):** With SSD-C and 64-GB DRAM
- **Performance-optimized system ($$$):** With SSD-P and 1-TB DRAM



MegIS improves system cost-efficiency

and makes accurate metagenomics more accessible

for wider adoption

MegIS outperforms the baselines

even when running on a much less costly system

More in the Paper

- MegIS's performance when running in-storage processing operations on the **cores existing in the SSD controller**
- MegIS's performance when using the same accelerators **outside SSD**
- **Sensitivity analysis with varying**
  - Database sizes
  - Memory capacities
  - #SSDs
  - #Channels
  - #Samples
- MegIS's performance for **abundance estimation**

More in the Paper

MegIS: High-Performance, Energy-Efficient, and Low-Cost Metagenomic Analysis with In-Storage Processing

Nika Mansouri Ghiasi[1] Mohammad Sadrosadati[1] Harun Mustafa[1] Arvid Gollwitzer[1]
Can Firtina[1] Julien Eudine[1] Haiyu Mao[1] Joël Lindegger[1] Meryem Banu Cavlak[1]
Mohammed Alser[1] Jisung Park[2] Onur Mutlu[1]
[1]ETH Zürich [2]POSTECH

- Database sizes
- Memory capacities
- #SSDs
- #Channels
- #Samples

- MegIS's performance for abundance estimation

https://arxiv.org/abs/2406.19113

Outline

Background

Motivation and Goal

MegIS

Evaluation

Conclusion

Conclusion

Metagenomic analysis suffers from
significant storage I/O data movement overhead

MegIS

The *first* **in-storage processing** system for *end-to-end* metagenomic analysis
Leverages and orchestrates **processing inside** and **outside** the storage system

Improves performance

2.7×–37.2× over **performance-optimized** software
6.9×–100.2× over **accuracy-optimized** software
1.5×–5.1× over **hardware-accelerated PIM** baseline

High accuracy

Same as accuracy-optimized
4.8× higher F1 score
over **performance-optimized/PIM**

Reduces energy consumption

5.4× over **performance-optimized** software
15.2× over **accuracy-optimized** software
1.9× over **hardware-accelerated PIM** baseline

Small area/power

Area: **0.04 mm2**
Power: **7.658 mW**

MegIS

High-Performance, Energy-Efficient, and Low-Cost Metagenomic Analysis with In-Storage Processing

https://arxiv.org/abs/2406.19113

SAFARI

ETH zürich

POSTECH

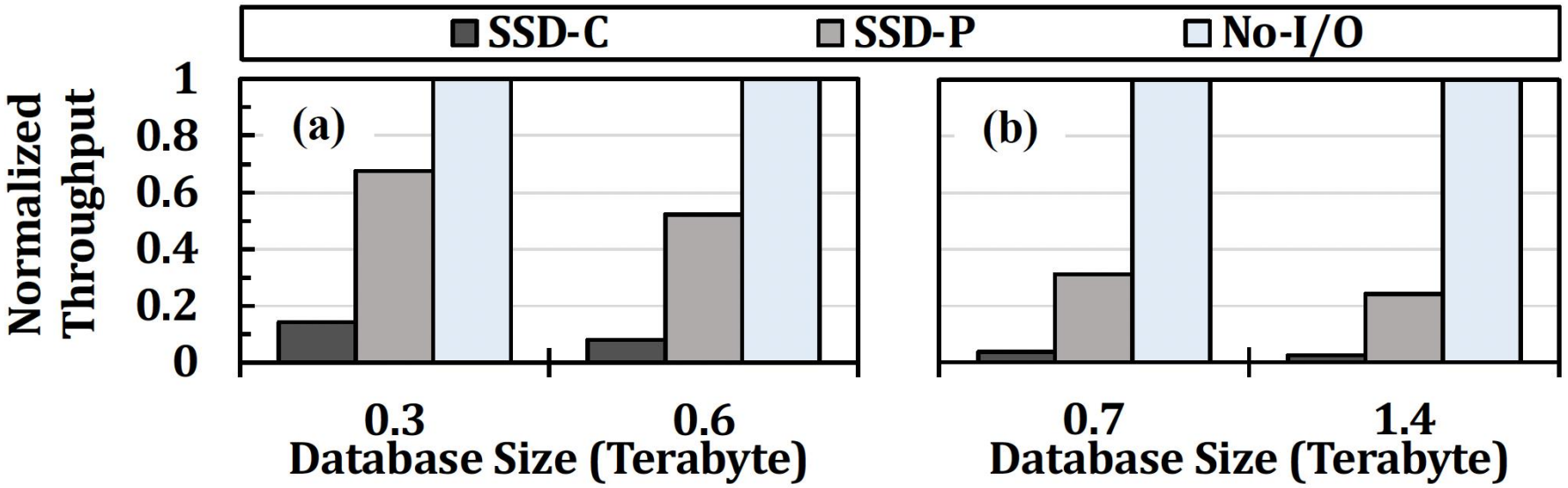
Backup Slides

Motivational Analysis

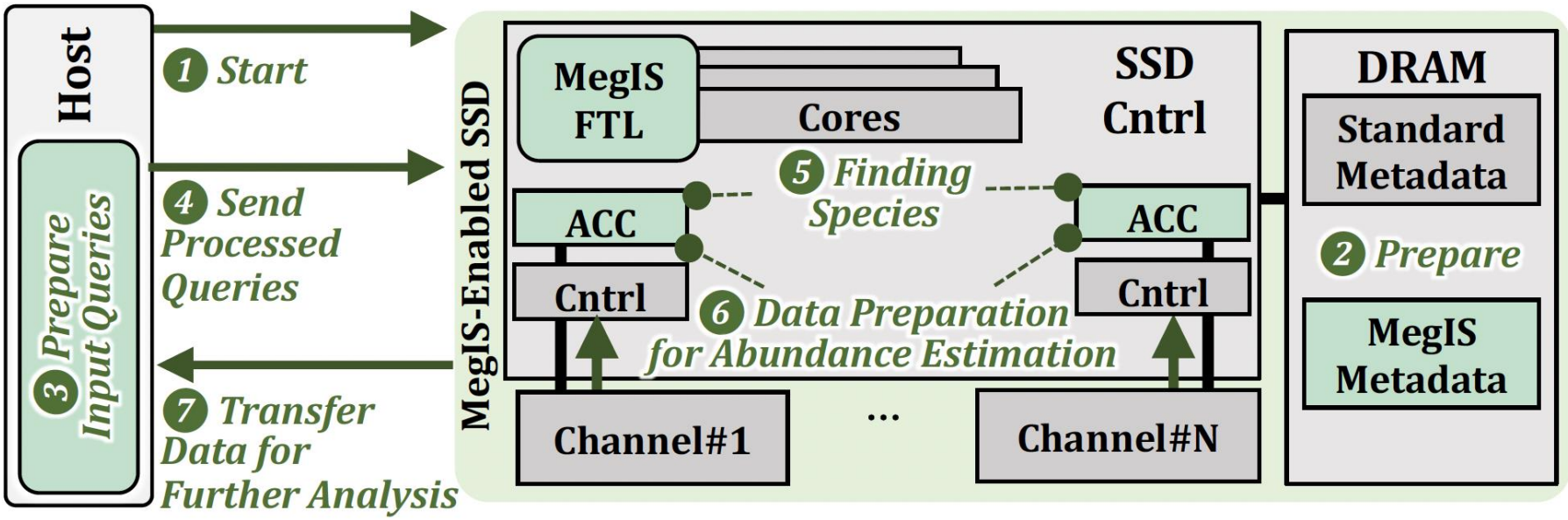
Database access patterns

(a)Random Query

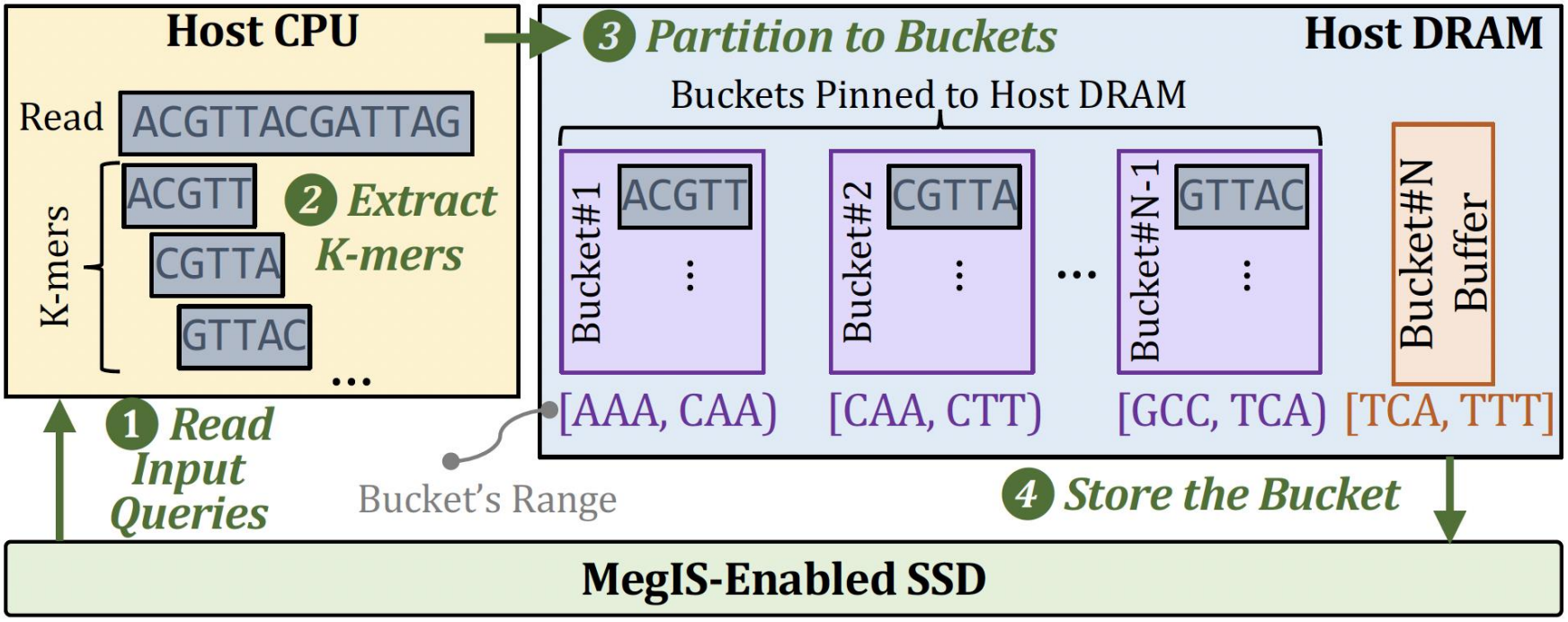
(b)Streaming Query



Overview of MegIS's Steps



More Details on Step 1



K-mer Sketch Data Structures

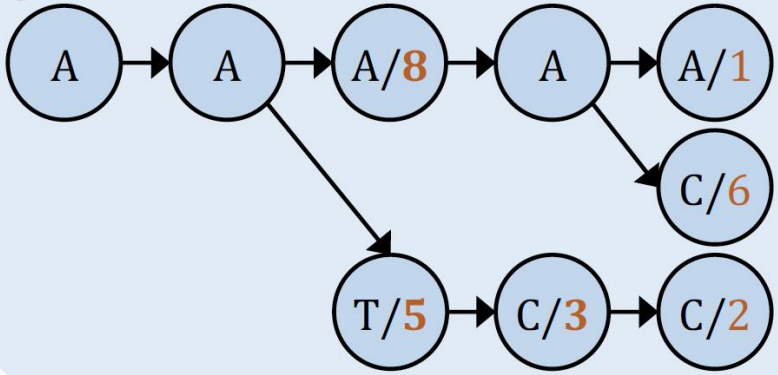
a Baseline K-mer Sketch Tables

5-mer	ID
AAAAA	1
AAAAC	6
AATCC	2
...	...

4-mer	ID
AAAA	1, 6
AATC	2, **3**
...	...

3-mer	ID
AAA	1, 6, **8**
AAT	2,3, **5**
...	...

b Ternary Search Tree



c K-mer Sketch Streaming Tables

5-mer	ID
AAAAA	1
AAAAC	6
AATCC	2
...	...

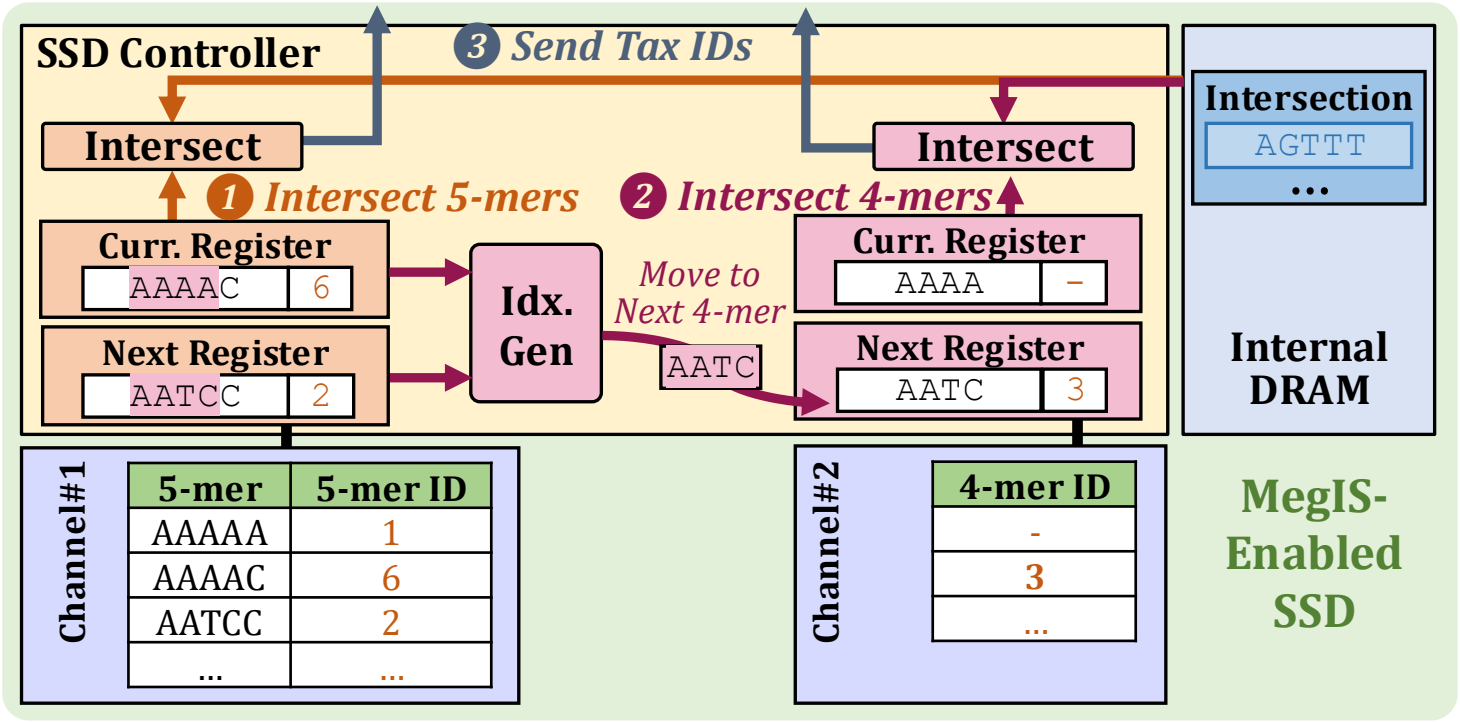
4-mer *i*

4-mer *i+1*

4-mer ID
-
3
...

3-mer ID
8
5
...

K-mer Sketch Streaming Hardware Design



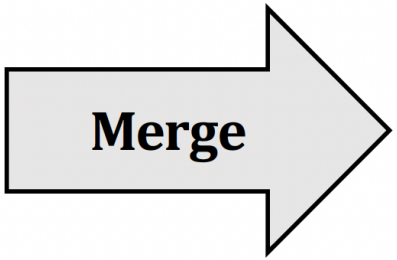
Index Generation in Step 3

K-mer	Loc.
ATT	14
CCA	9
GCT	5
…	…

Reference Index Organism A

K-mer	Loc.
AAG	2
CCA	21
TGC	4
…	…

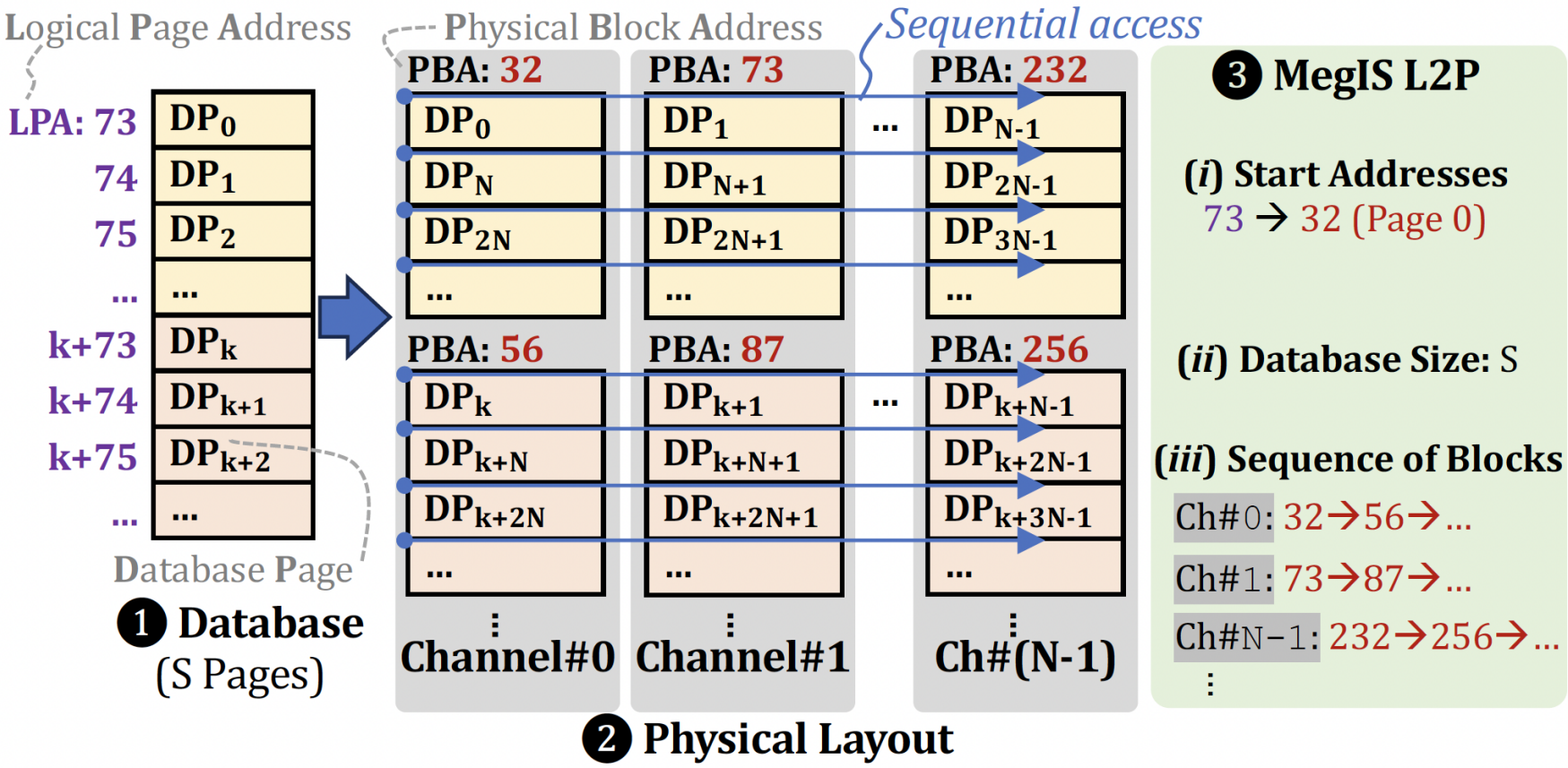
Reference Index Organism B



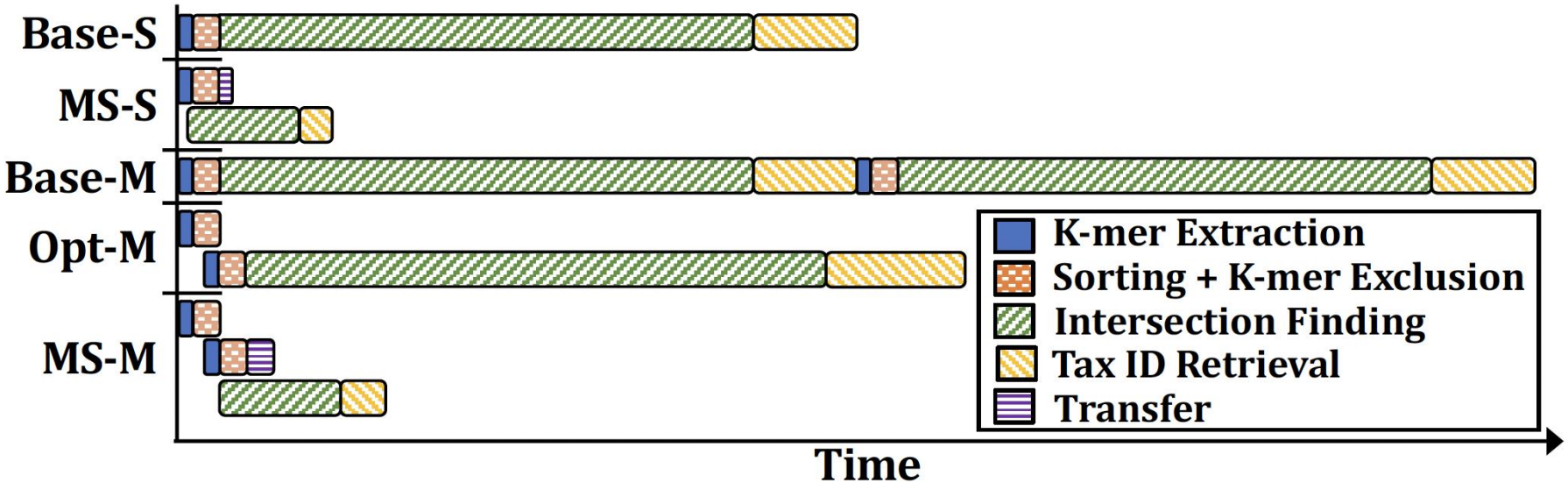
K-mer	Loc.
AAG	1002
ATT	14
CCA	9, 1021
GCT	5
TGC	1004
…	…

Unified Reference Index

MegIS FTL


Logical Page Address
Physical Block Address
Sequential access
LPA: 73
74
75
...
k+73
k+74
k+75
...
DP0
DP1
DP2
...
DPk
DPk+1
DPk+2
...
Database Page
1 Database
(S Pages)
PBA: 32
PBA: 73
PBA: 232
DP0
DP1
...
DPN-1
DPN
DPN+1
DP2N-1
DP2N
DP2N+1
DP3N-1
PBA: 56
PBA: 87
PBA: 256
DPk
DPk+1
DPk+N-1
DPk+N
DPk+N+1
DPk+2N-1
DPk+2N
DPk+2N+1
DPk+3N-1
Channel#0
Channel#1
Ch#(N-1)
2 Physical Layout
3 MegIS L2P
(i) Start Addresses
73 → 32 (Page 0)
(ii) Database Size: S
(iii) Sequence of Blocks
Ch#0: 32→56→...
Ch#1: 73→87→...
Ch#N-1: 232→256→...

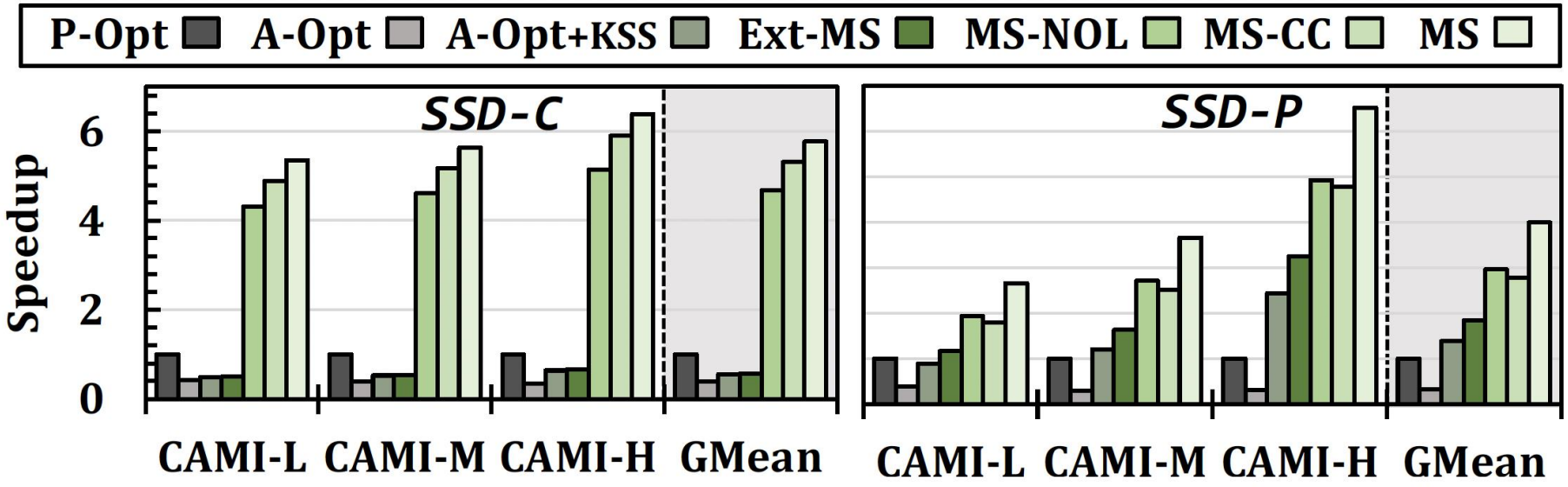
Multi-Sample Analysis



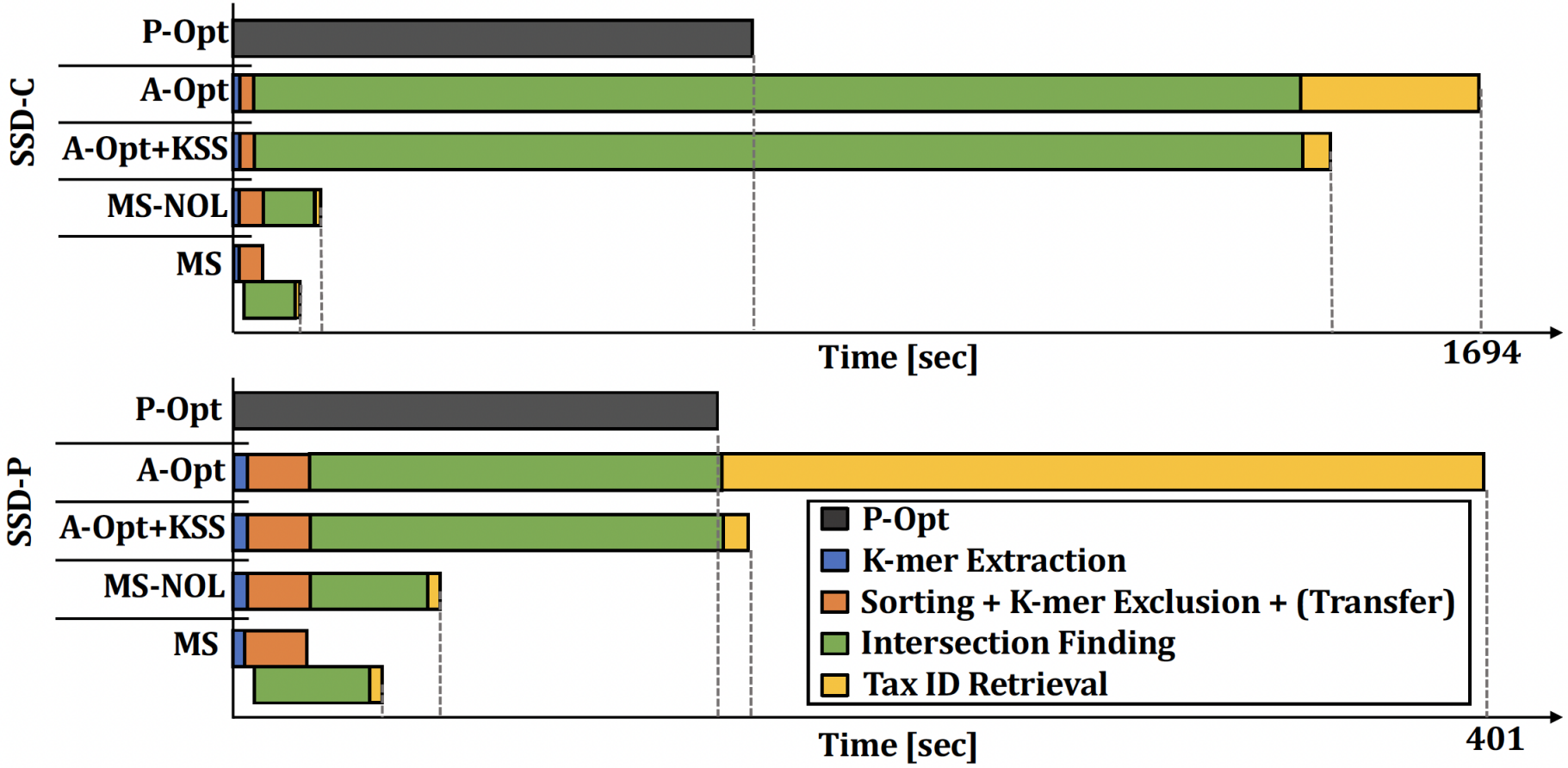
SSD Configurations

Specification	SSD-C	SSD-P
General	48-WL-layer 3D TLC NAND flash-based SSD 4 TB capacity, 4 GB internal LPDDR4 DRAM [226]	
Bandwidth (BW)	600 MB/s interface BW (SATA3); 560 MB/s sequential-read BW 1.2-GB/s channel I/O rate	8 GB/s interface BW (4-lane PCIe Gen4); 7 GB/s sequential-read BW 1.2-GB/s channel I/O rate
NAND Config	8 channels, 8 dies/channel, 4 planes/dies, 2,048 blocks/plane, 196 WLs/block, 16 KiB/page *(4/8/16 channels in Fig. 17)*	16 channels, 8 dies/channel, 2 planes/dies, 2,048 blocks/plane, 196 WLs/block, 16 KiB/page *(8/16/32 channels in Fig. 17)*
Latencies	Read (tR): 52.5 μs, Program (tPROG): 700 μs	
Embedded Cores	3 ARM Cortex-R4 cores [86]	4 ARM Cortex-R4 cores [86]

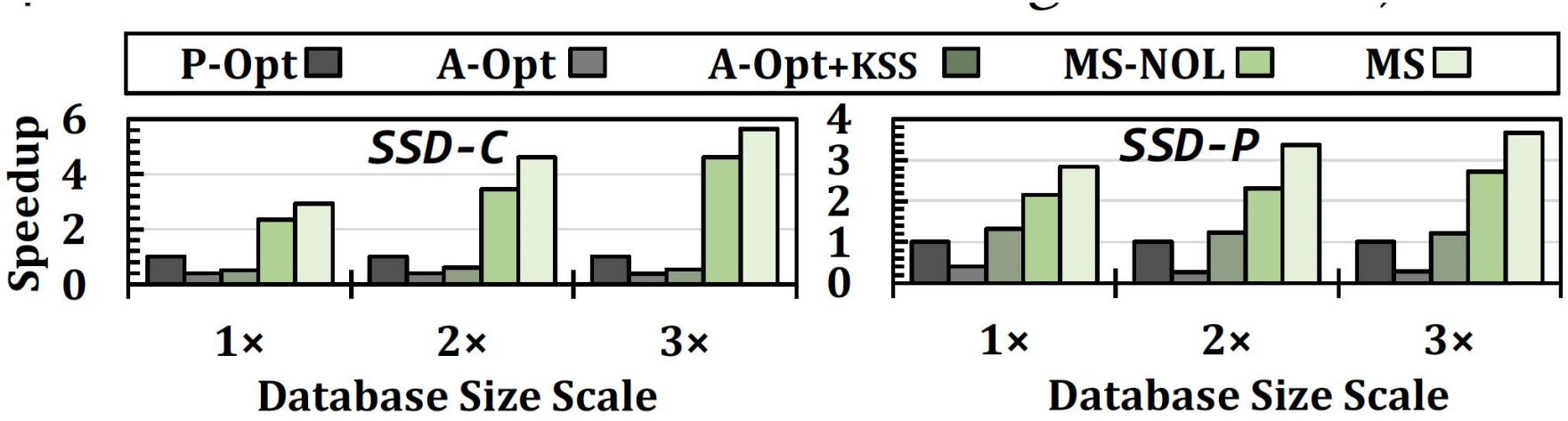
Impact of Different Optimizations



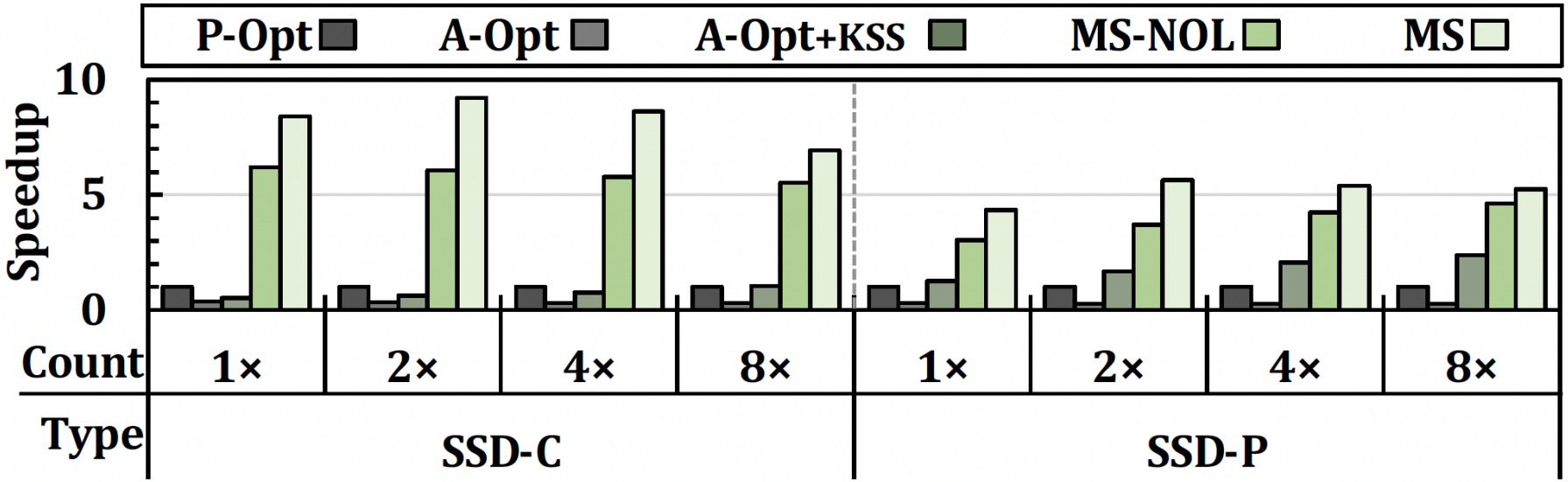
Impact of Different Optimizations



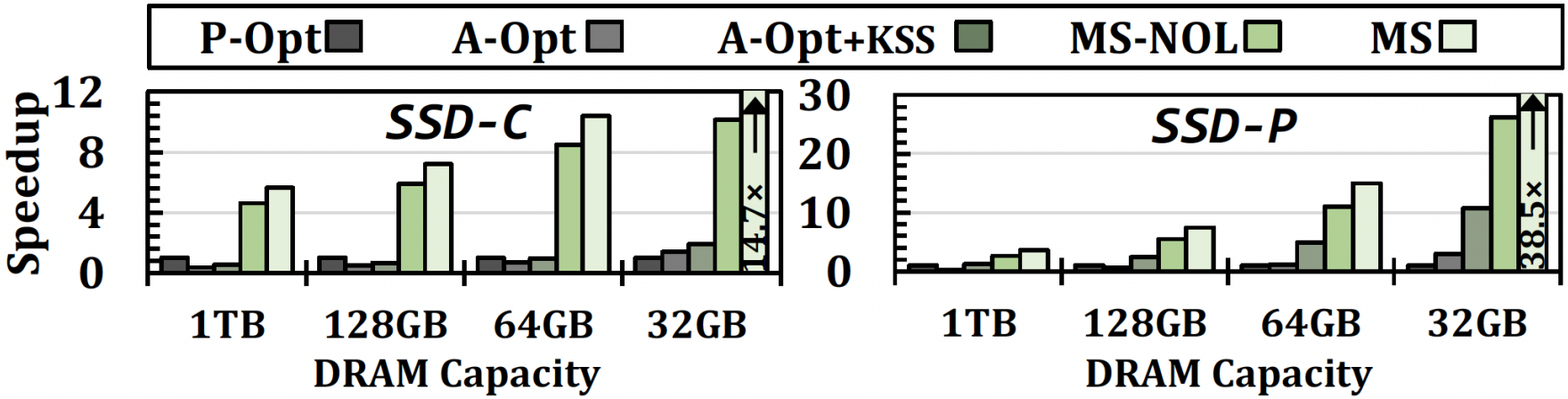
Speedup with Different Database Sizes



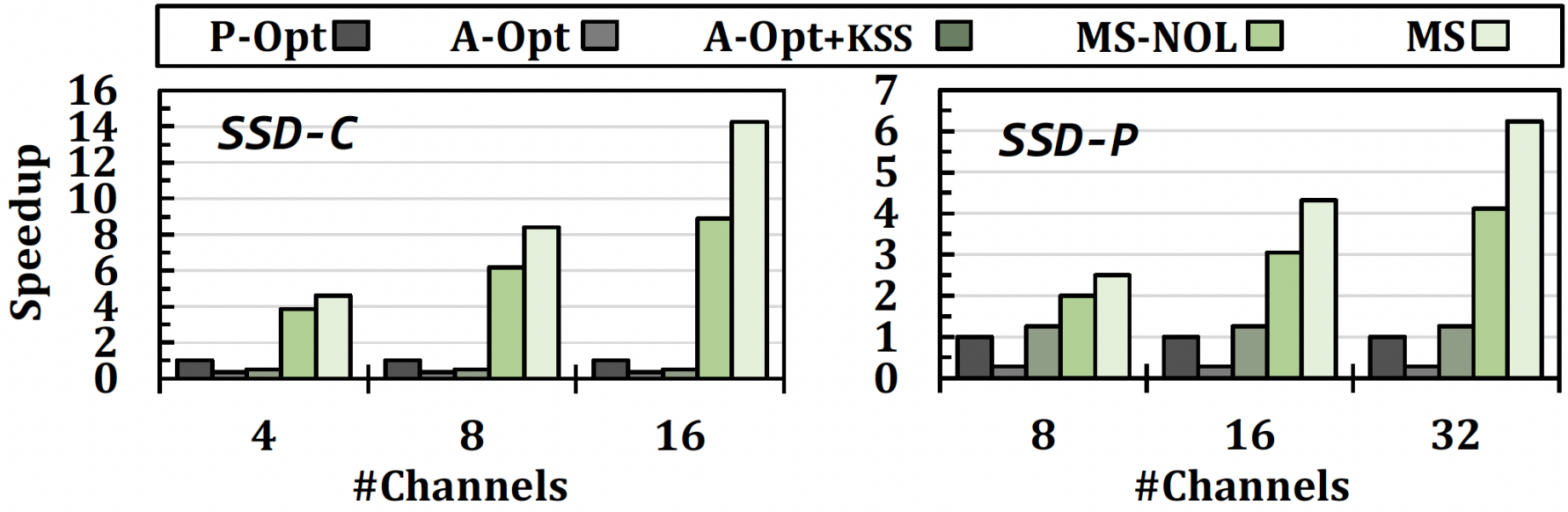
Speedup with Different #SSDs



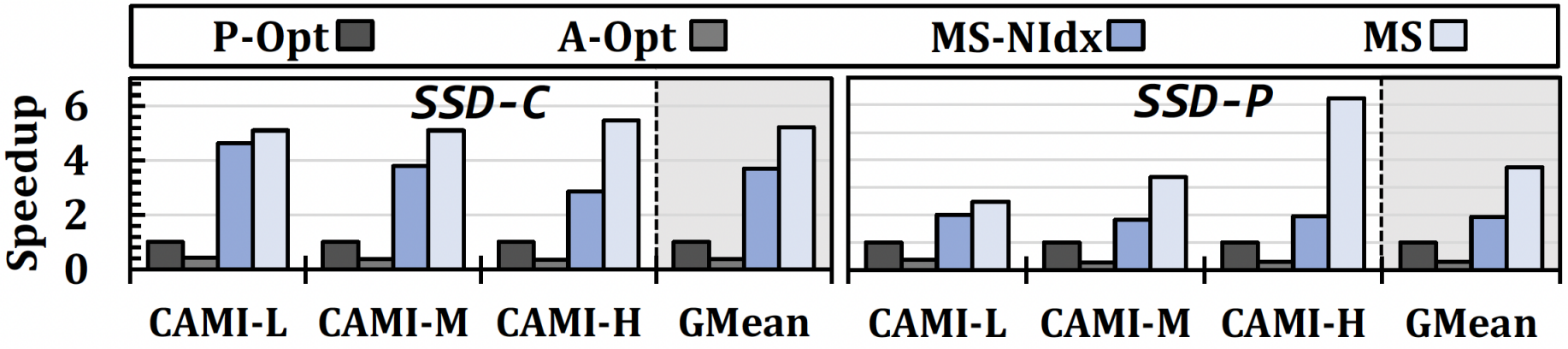
Speedup with Different Main Memory Capacities



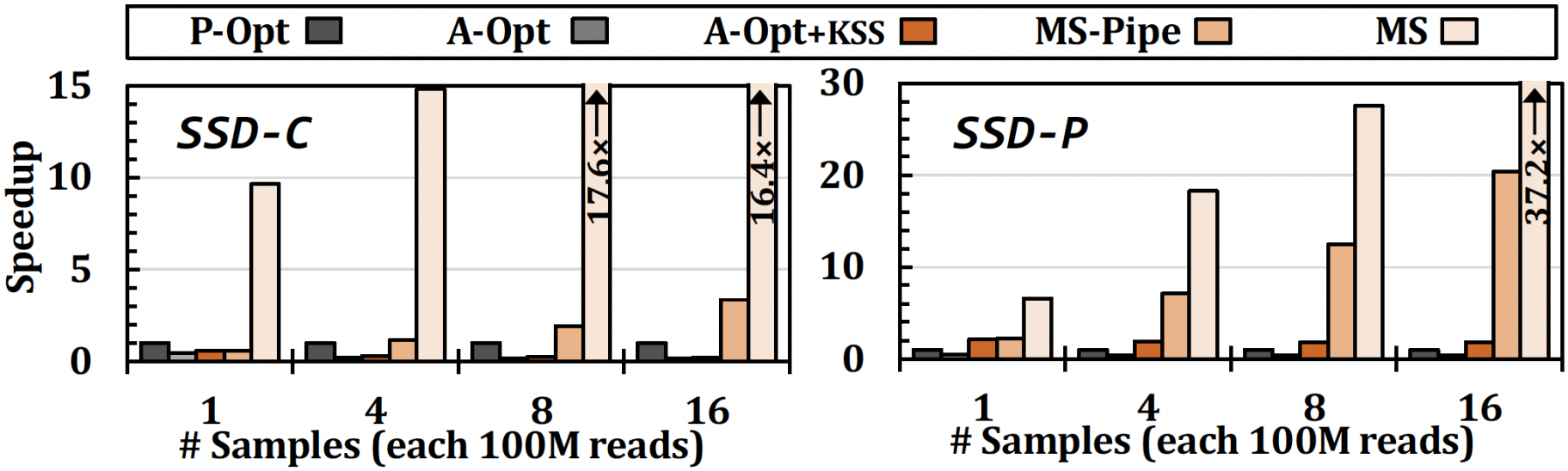
Speedup with Varying SSD Internal Bandwidth



Speedup of Abundance Estimation



Multi-Sample Use Case



Area and Power

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

Logic Unit	# of instances	Area [mm²]	Power [mW]
Intersect (120-bit)	1 per channel	0.001361	0.284
k-mer Registers (2 x 120-bit)	1 per channel	0.002821	0.645
Index Generator (64-bit)	1 per channel	0.000272	0.025
Control Unit	1 per SSD	0.000188	0.026
Total for an 8-channel SSD	-	*0.04*	*7.658*

Only 1.7% of the area of three 28-nm ARM Cortex R4 cores in a SATA SSD controller

GRAINS

Enabling High-Performance and Low-Cost Graph-Based Genome Analysis via Storage-Aware Algorithm-Architecture Co-Design

Nika Mansouri Ghiasi

Harun Mustafa Talu Güloglu Rakesh Nadig Konstantina Koliogeorgi

Susana Rebolledo Ruiz Marc Rautmann Furkan Eris

Mohammad Sadrosadati Jisung Park Onur Mutlu

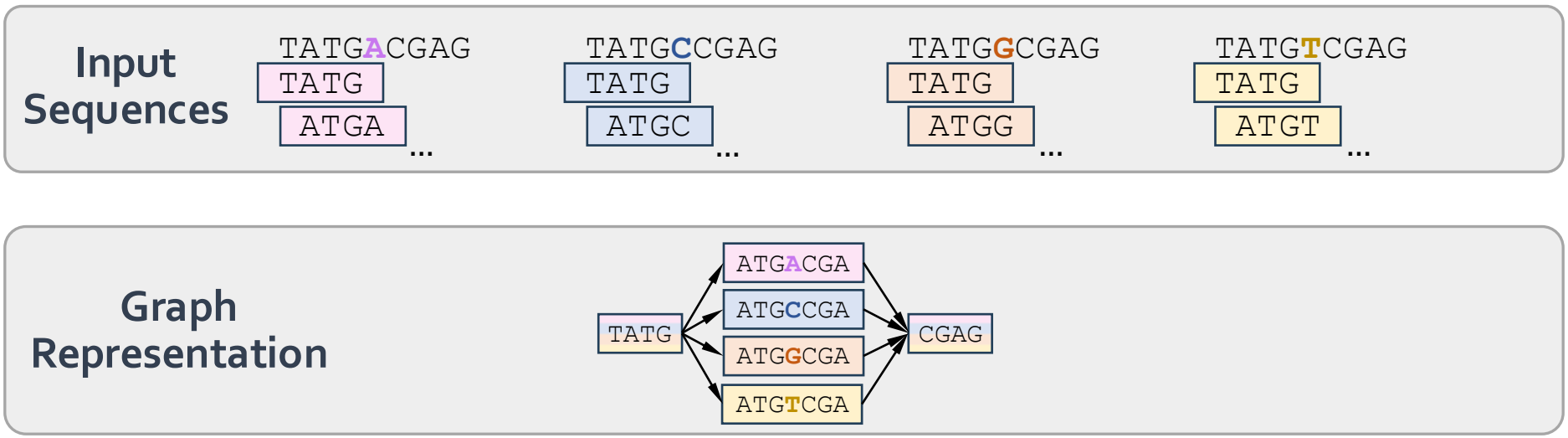
SAFARI

ETH zürich POSTECH

GRAINS Discussion

- Main presentation
- Unique characteristics of genome graphs
- De Bruijn Graphs
- Graph-based genome analysis
- SSD Configurations
- Detailed results

Large-Scale Genome Graphs



More powerful and efficient representations of genomic data

✓ Greater expressive power

✓ More compact representations of massive, population-scale databases

Unique Characteristics of Genome Graphs

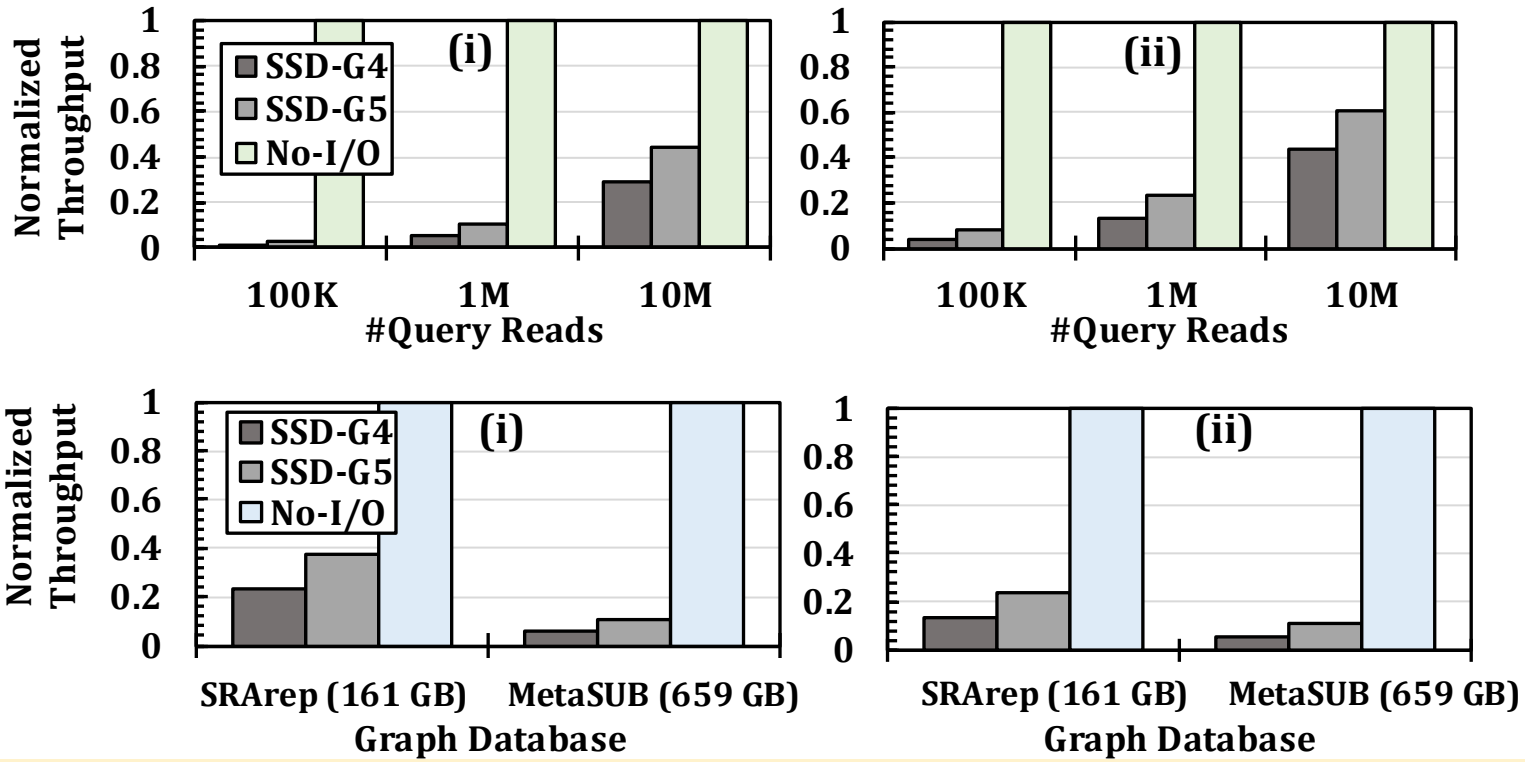
- **Different Node and Edge Encodings**
  - Traditional graphs must explicitly store O(N) to O(N^2) edges
  - DBGs store only nodes or edges and the other is derived implicitly
  - Enabled by the inherent overlap structure of encoded sequences

- **Different Structural Characteristics**
  - In/out-degrees capped at 4 (A, C, G, T), regardless of graph size
  - Power-law hub optimizations from conventional graphs are inapplicable
  - Necessitates fundamentally different optimization approaches

- **Different Access Pattern Optimizations**
  - Compressed data structures enforce node orderings ➔ no reordering possible
  - Structural coupling between queries and graph enables unique locality optimizations
  - Query reads sharing k-mer substrings map to nearby index regions

Motivation

- Case study of the performance of genome graph analysis tools
- With various state-of-the-art SSD configurations and data sizes



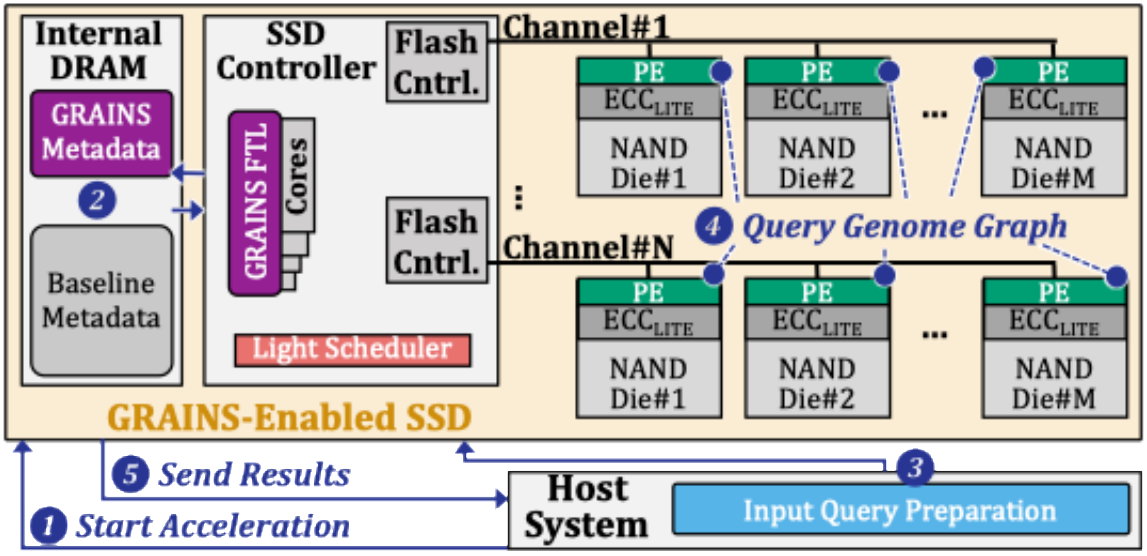
I/O data movement causes significant performance overhead

Goal

Improve the performance and energy-efficiency of graph-based genome analysis by alleviating its data movement overheads from storage in a cost-effective manner.

GRAINS

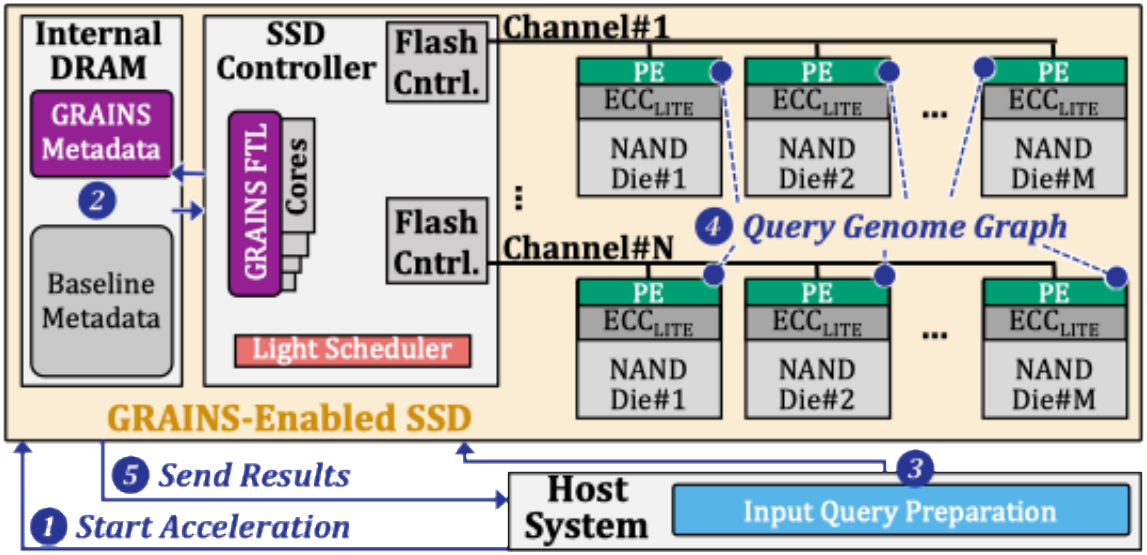
- The first system for analysis with large-scale genome graphs in storage
- Enabled via storage-aware algorithm architecture co-design to
  - Make the pipelines **more storage-friendly**
  - Further improve performance, energy-efficiency, and cost-effectiveness via **in-storage (ISP) and in-flash (IFP) processing**



GRAINS Hardware-Software Co-Design

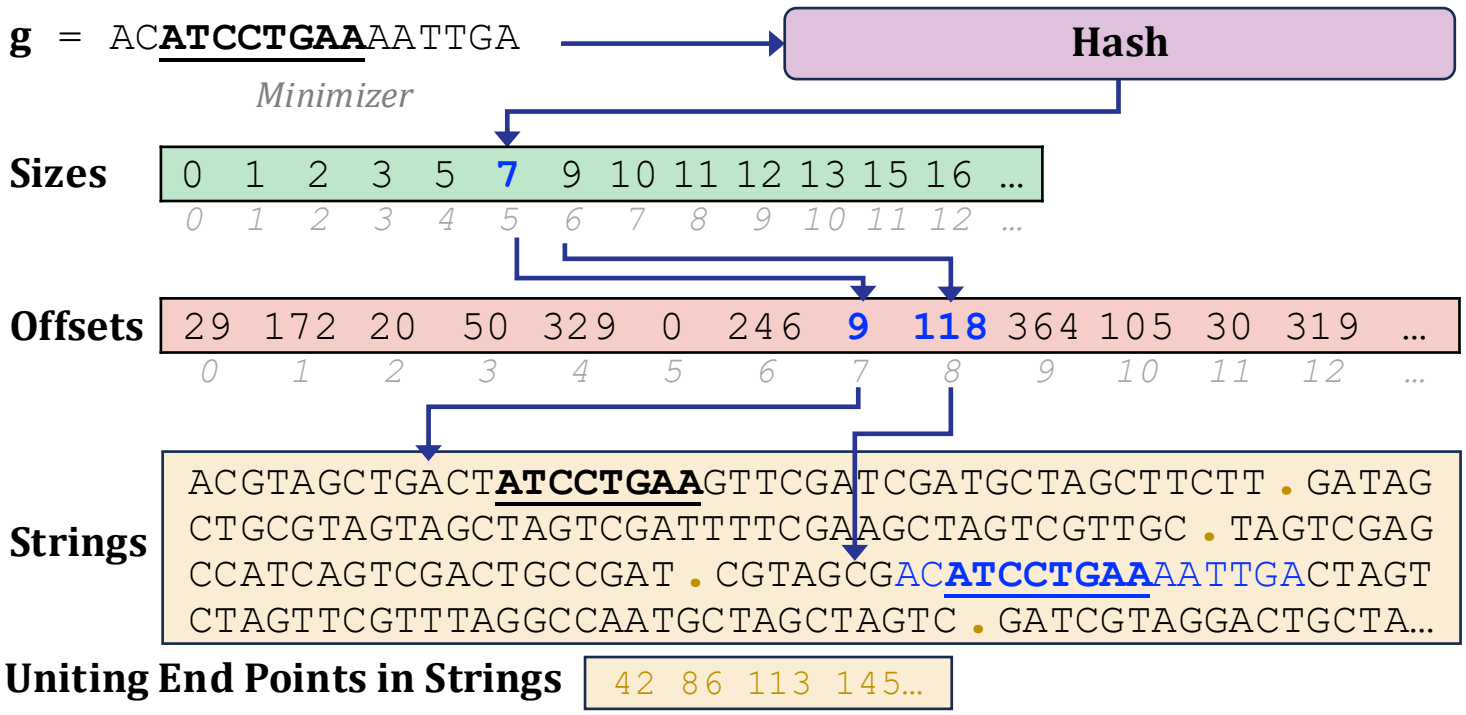
A new batching technique and execution flow, that reduces #random accesses

ISP and IFP to avoid transferring low-reuse/unused pages and bandwidth waste

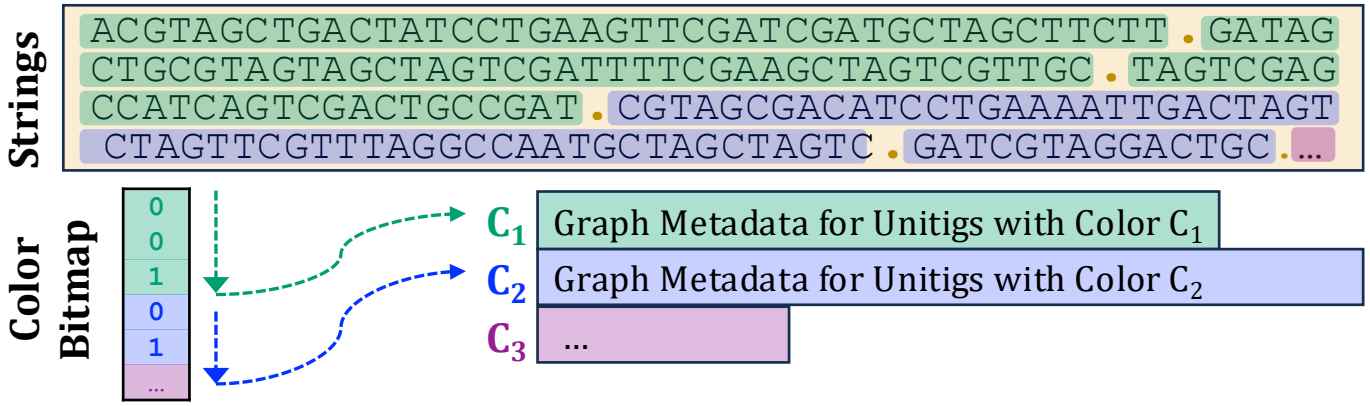


An effective, yet lightweight, scheduling technique, enabled by repurposing the existing SSD structures in a new way to enable leveraging the full parallelism of all flash dies during IFP

Overview of Graph Structure and K-Mer Queries

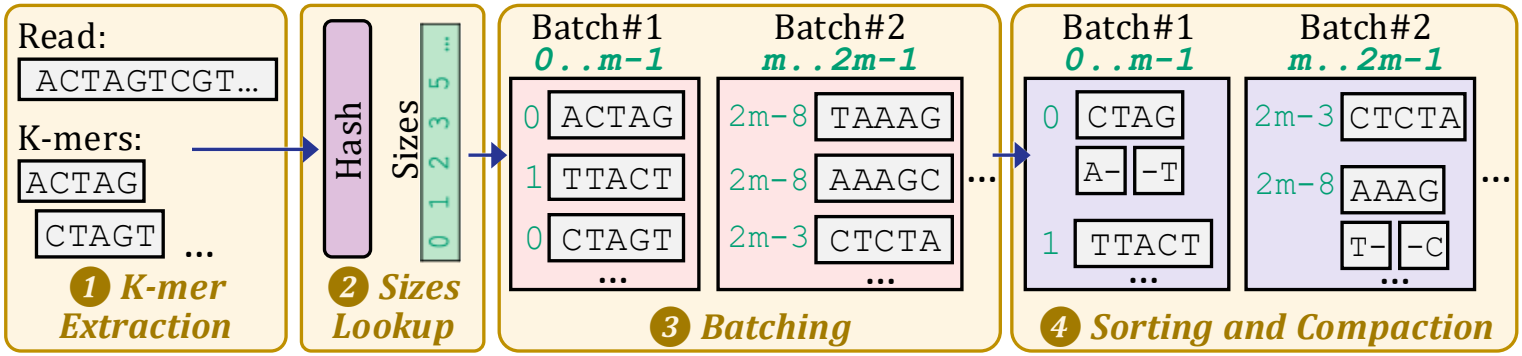


Overview of Graph Color Encoding



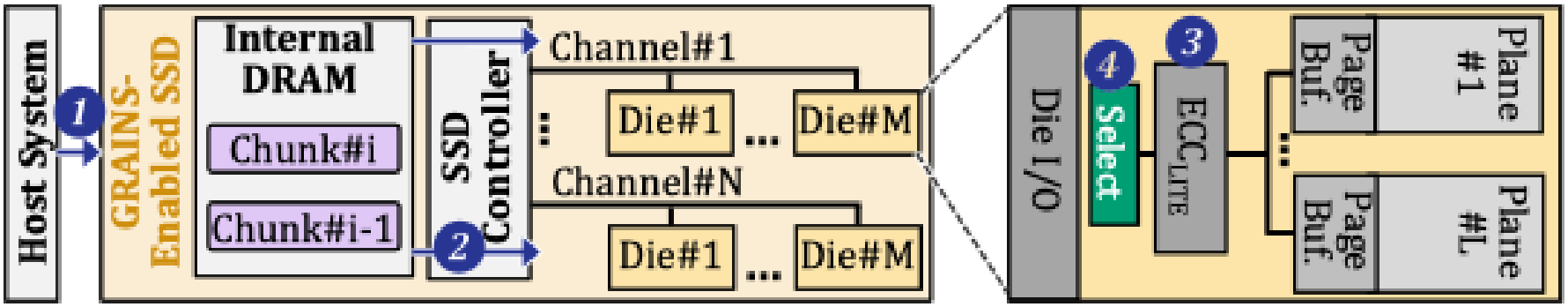
Step 1: Query Processing

- Performed in the host system to avoid excessive writes to NAND flash dies during k-mer extraction
- Leverages the key property of the underlying data structures: The small Sizes



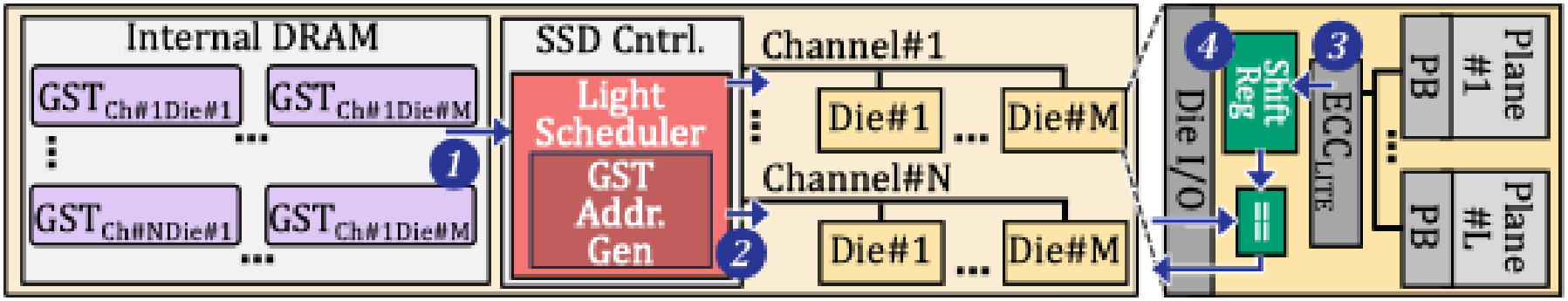
Step 2.1: Accessing Graph Offsets

- Performed in the storage system to avoid the data movement overhead of a large amount of low re-use data



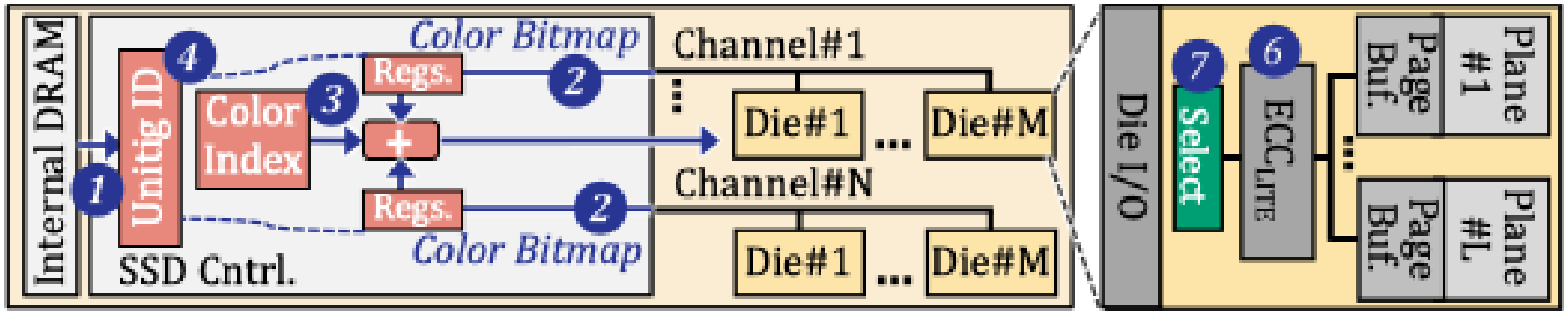
Step 2.2: Access Graph Strings

- Performed in the storage system to avoid the data movement overhead of a large amount of low re-use data
- Lightweight in-storage scheduler to enable full die-level parallelism
  - Small L2P data mapping ➔ Leverage internal DRAM to store per-die scheduling table
  - Ensures all requests to different dies and planes can be issues concurrently



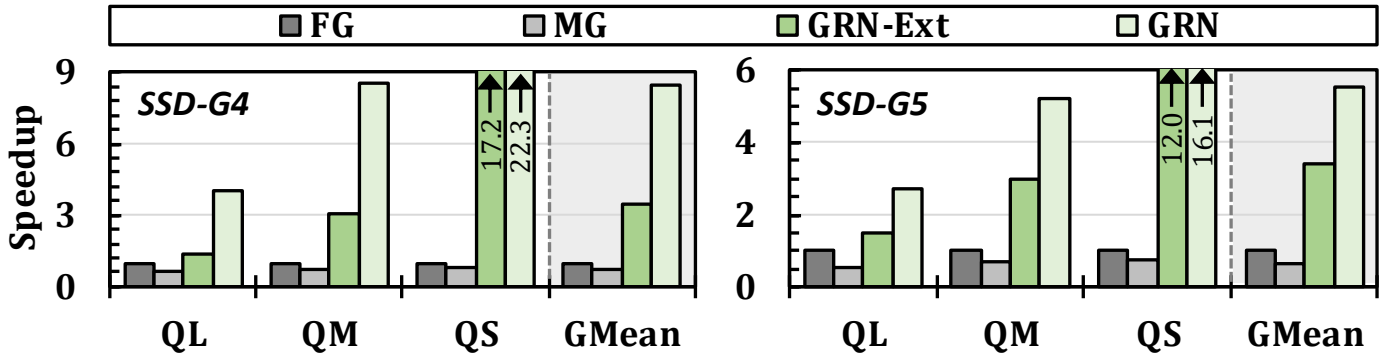
Step 2.3: Accessing Graph Colors

- Performed in the storage system to avoid the data movement overhead of a large amount of low re-use data
- Streams through color bitmaps sequentially
- Sends the colors to the host system in a pipelined manner



Performance

- Performance of k-mer set lookups
- Across different tools and storage configurations

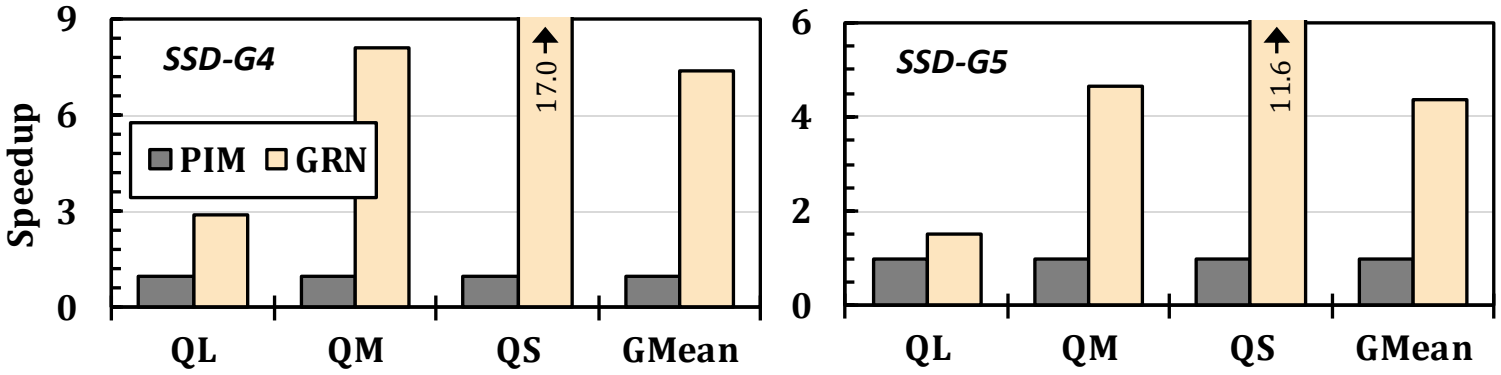


On systems with SSD-G4 (SSD-G5)
GRN outperforms FG and MG by 8.4× (8.5×) and 11.9× (8.5×), on average across all inputs, respectively

GRN provides 2× larger average speedup over GRN-Ext

Comparison to PIM

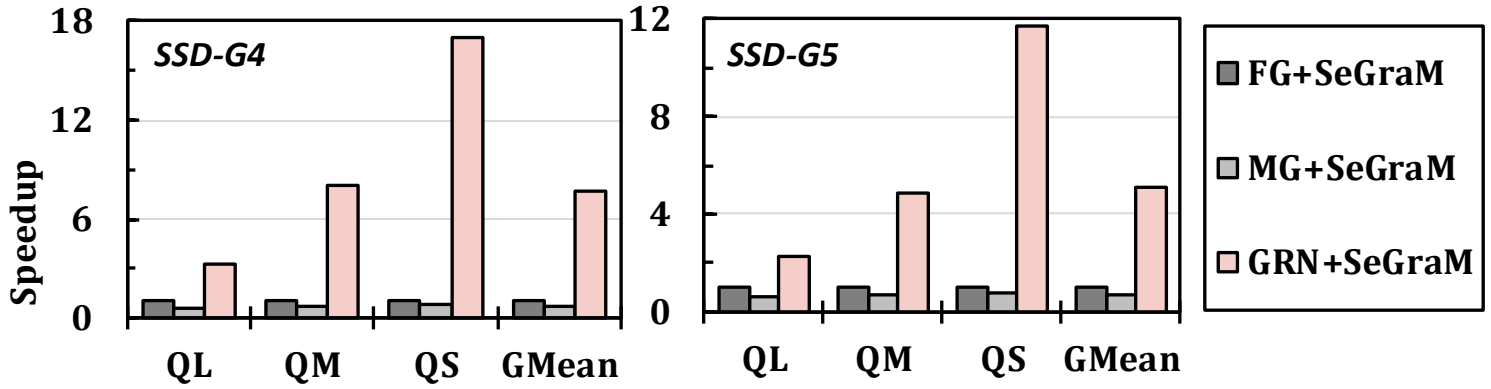
- Performance of k-mer set lookups
- Compared to an idealized PIM system without memory overheads for graph lookups



on the system with SSD-G4 (SSD-G5),
GRN provides 7.4× (4.3×) average speedup over PIM

Performance of Alignment-Based Mapping

- GRAINS integrated with a state-of-the-art sequence-to-graph alignment accelerator, SeGraM
- GRAINS finds the relevant parts of the graph and passes them to SeGraM to go through alignment



SeGraM+GRN provides 6.2× and 9.0× average speedup over SeGraM+FG and SeGraM+MG

Area and Power

- Based on **Synthesis** of **GRAINS's hardware units** using the Synopsys Design Compiler @ 22nm technology node

- ECC_{LITE} area based on the original paper [AiF, ISCA'25]

Logic unit	Area [mm^2]	Power [mW]
On SSD Controller	0.0025	0.21
On Die (ECC_{LITE})	0.036	18.04
On Die (Others)	0.000093	0.01

On-controller logic units are 0.7% of the four cores on an SSD controller

On-die logic area is mainly (99%) for ECC_{LITE},
which takes only 0.2% of the flash chip's area
and fits well within its power budget

GRAINS

Enabling High-Performance and Low-Cost Graph-Based Genome Analysis via Storage-Aware Algorithm-Architecture Co-Design

Nika Mansouri Ghiasi

Harun Mustafa Talu Güloglu Rakesh Nadig Konstantina Koliogeorgi

Susana Rebolledo Ruiz Marc Rautmann Furkan Eris

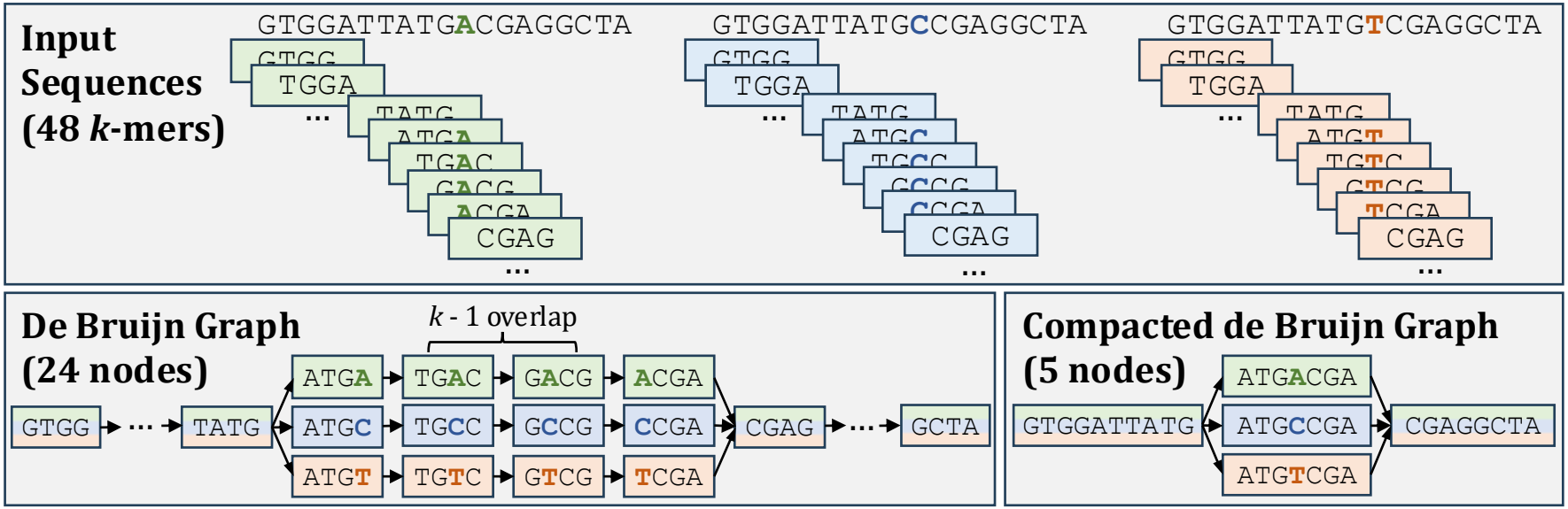
Mohammad Sadrosadati Jisung Park Onur Mutlu

SAFARI

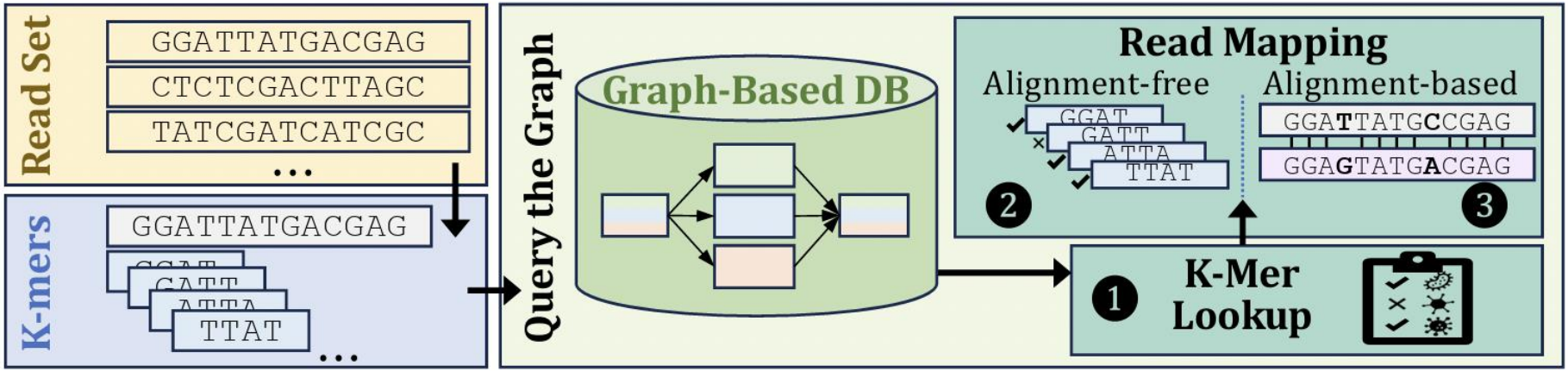
ETH zürich POSTECH

Backup Slides

De Bruijn Graphs



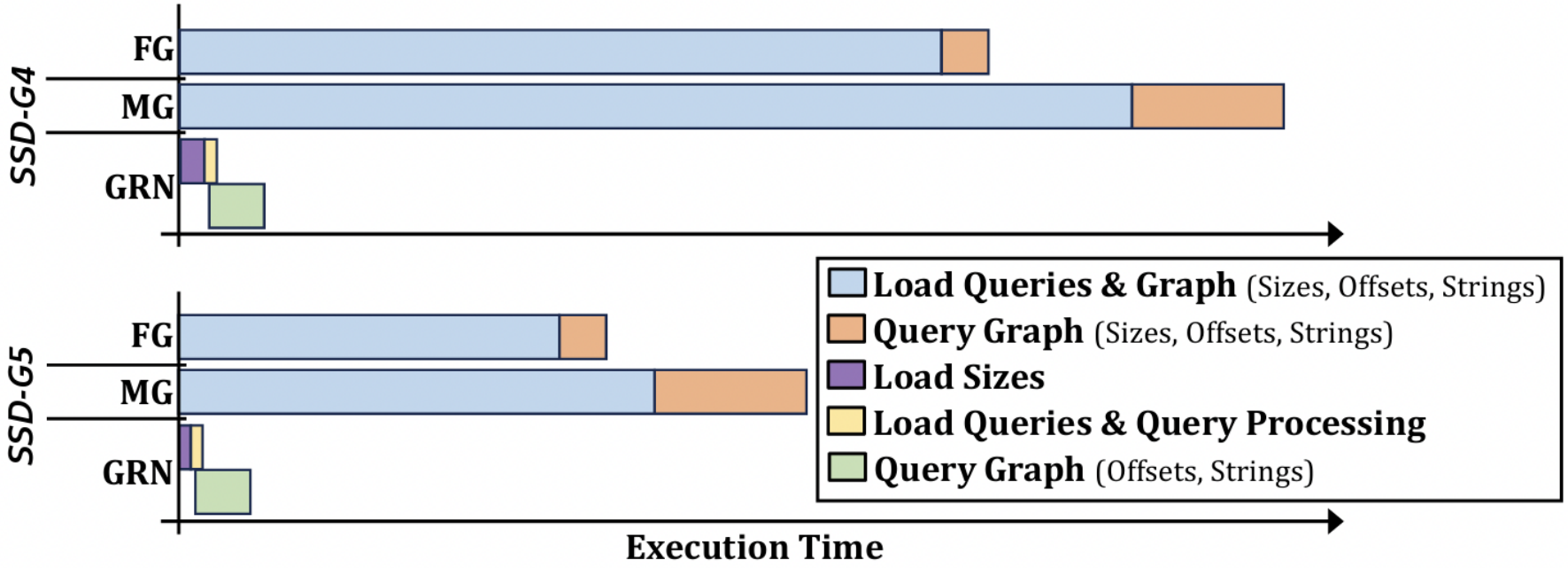
Graph-Based Genome Analysis



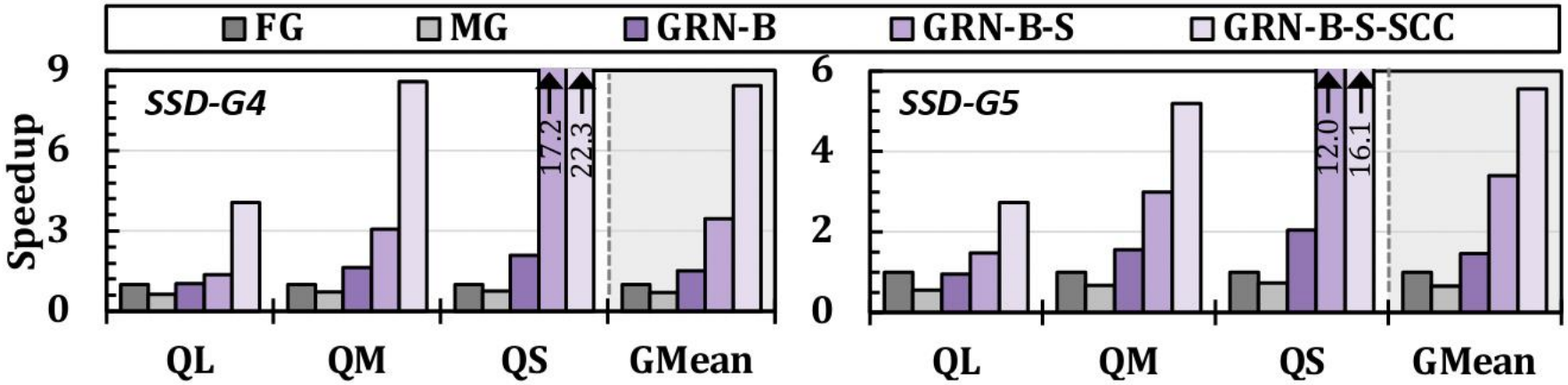
SSD Configurations

Specification	SSD-G4	SSD-G5
General	TLC NAND flash-based SSD 4 TB capacity, 4 GB internal DRAM [218]	
Bandwidth (BW)	PCIe Gen4 interface; 7 GB/s sequential-read BW; 1.2-GB/s channel I/O rate	PCIe Gen5 interface; 14.8 GB/s sequential-read BW; 2.4-GB/s channel I/O rate
NAND Config	16 channels, 8 dies/channel, 4 planes/die	

Execution Time Breakdown



Ablation Analysis



Speedup with Different #SSDs

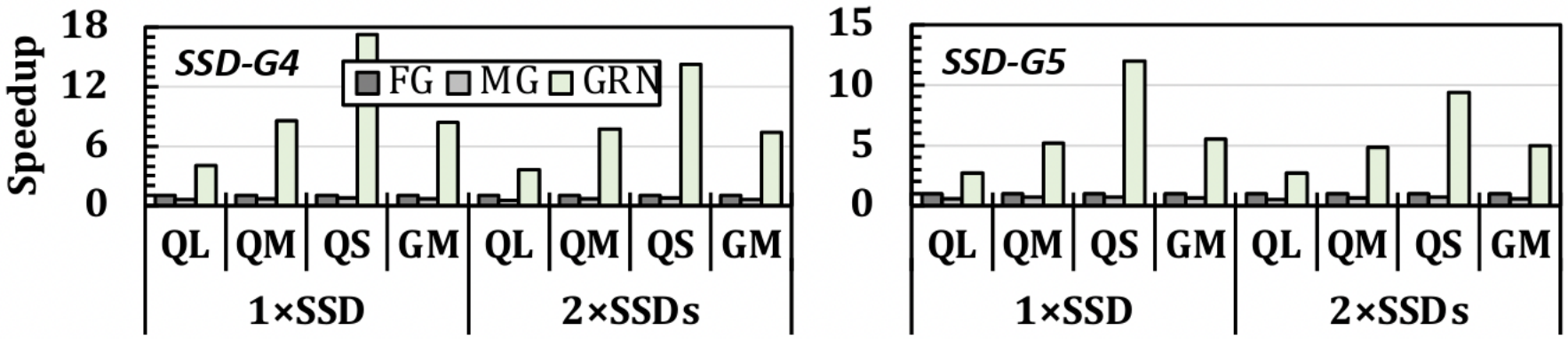


Fig. 17: Speedups with different numbers of SSDs.

A Lightweight Algorithm-Architecture Co-Design for Mitigating the Data Preparation Bottleneck in Large-Scale Genome Sequence Analysis

Nika Mansouri Ghiasi

Talu Güloglu Harun Mustafa Can Firtina Konstantina Koliogeorgi

Konstantinos Kanellopoulos Haiyu Mao Rakesh Nadig

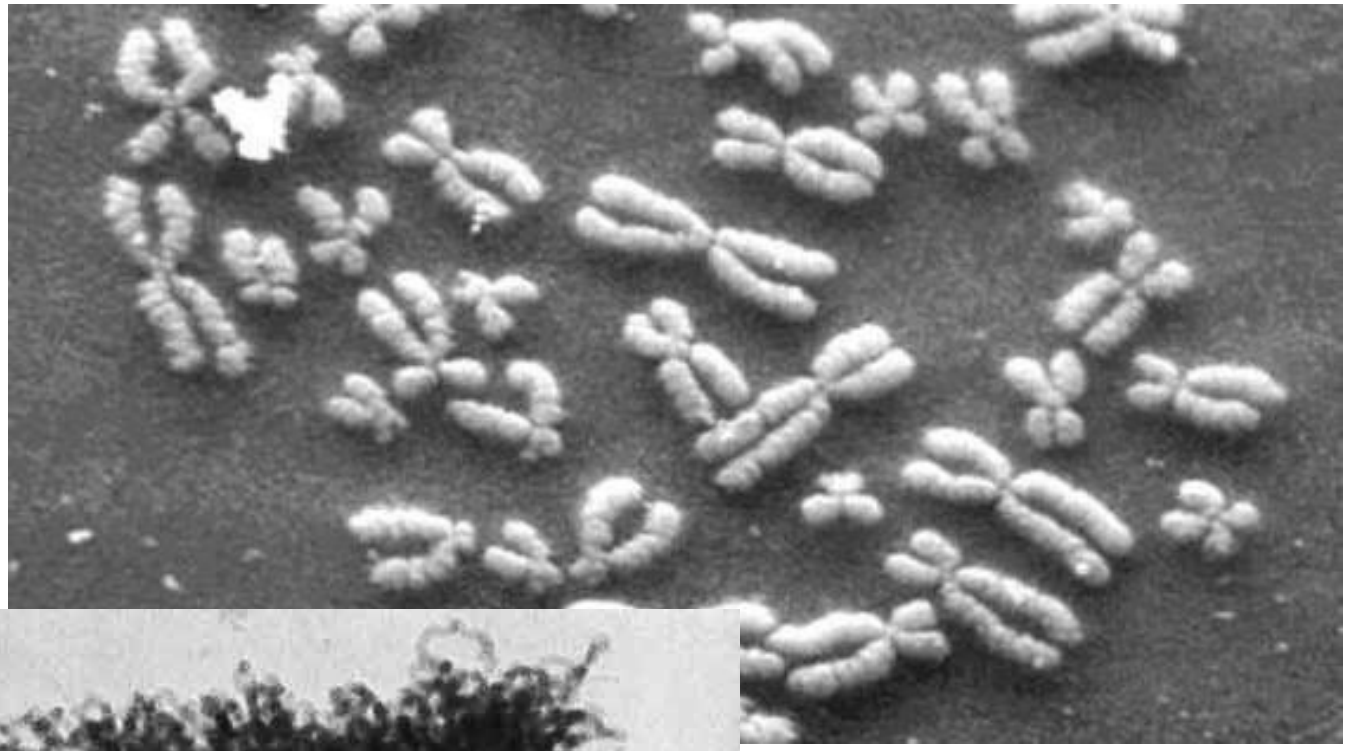
Mohammad Sadrosadati Jisung Park Onur Mutlu

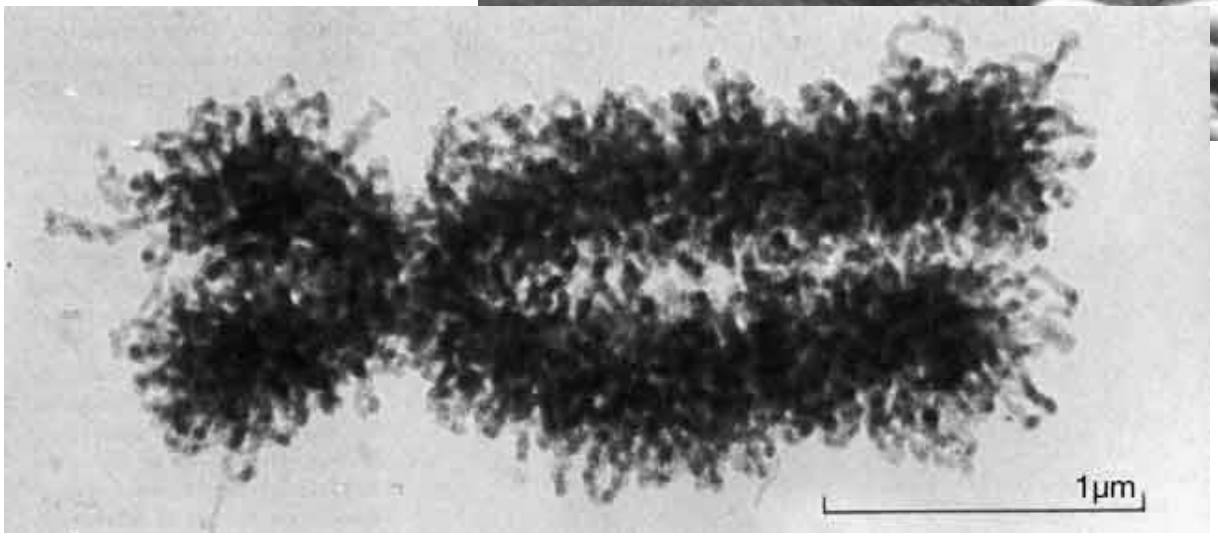
SAFARI

ETH zürich UNIVERSITY OF MARYLAND KING'S College LONDON POSTECH

Applications of Genome Sequence Analysis

DNA Under Electron Microscope





human chromosome #12
from a HeLa cell

Digital Format for Analysis

CCTCCTCAGTGCCACCCAGCCCACTGGCAGCTCCCA
AACAGGCTCTTATTAAAACACCCTGTTCCCTGCCCC
TTGGAGTGAGGTGTCAAGGACCTAAACTAAAAAAAA
AAAAAGAAAAAGAAAAGAAAAAGAATTTAAAATTTA
AGTAATTCTTTGAAAAAAACTAATTTCTAAGCTTCT
TCATGTCAAGGACCTAATGTGCTAAACAGCACTTTT
TTGACCATTATTTTGGATCTGAAAGAAATCAAGAAT
AAATGAAGGACTTGATACATTGGAAGAGGAGAGTCA
AGGACCTACAGAAAAAAAAAAAAAAGAAAAAGAAAA
GAAAAAGAATTTAAAATTTAAGTAATTCTTTGAAAA
AAACTAATTTCTAAGCTTCTTATGTCAAGGACCTAA
TGTCTGTGTTGCAGGTCTTCTTGCATCTTCCCTGTC

High-Throughput Sequencers

Illumina MiSeq

Pacific Biosciences Sequel II

Oxford Nanopore PromethION

Oxford Nanopore MinION

Illumina NovaSeq 6000

Pacific Biosciences RS II

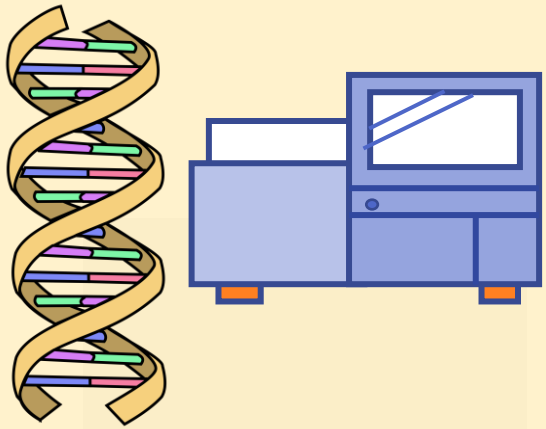
Oxford Nanopore GridION

… and more! All produce data with different properties.

High-Throughput Sequencers

Current sequencing technologies cannot sequence very long DNA molecules end-to-end

Instead, they extract smaller fragments of the original DNA sequence, known as reads

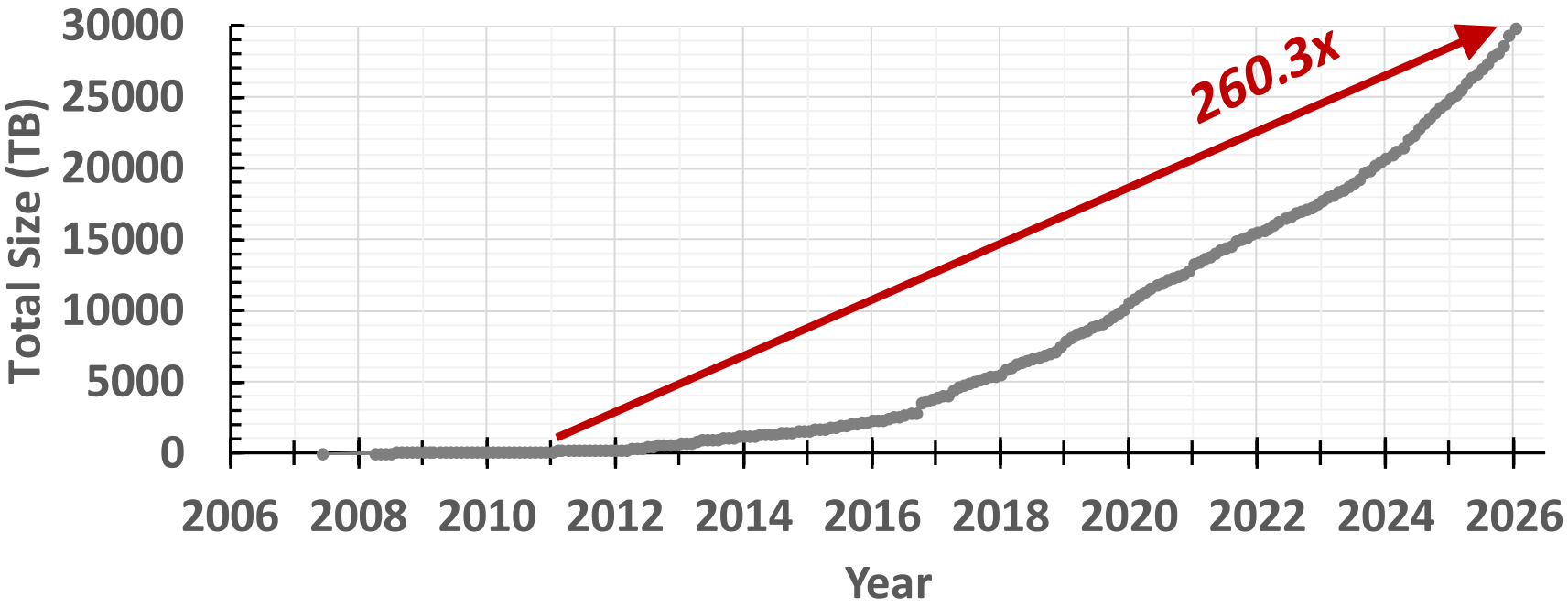


Genomic Data is Growing at a Fast Pace

Important Role in Many Fields

Significant Drop in Sequencing Cost

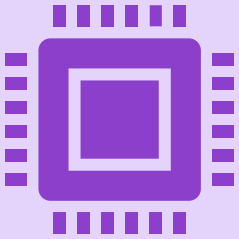
Fast growth in genomic data size



Public Genomic Sequence Reads Data Size in Sequence Read Archive (SRA) Over Years

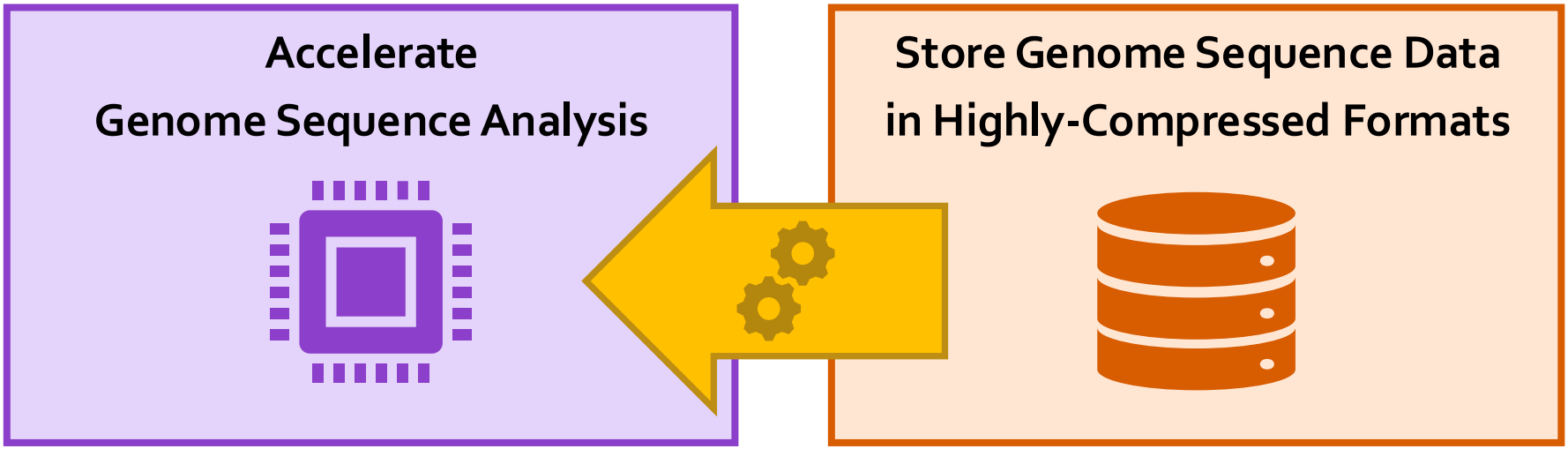
Large Efforts in Handling Fast-Growing Genomic Data

Accelerate
Genome Sequence Analysis



Store Genome Sequence Data
in Highly-Compressed Formats

Data Preparation Bottleneck

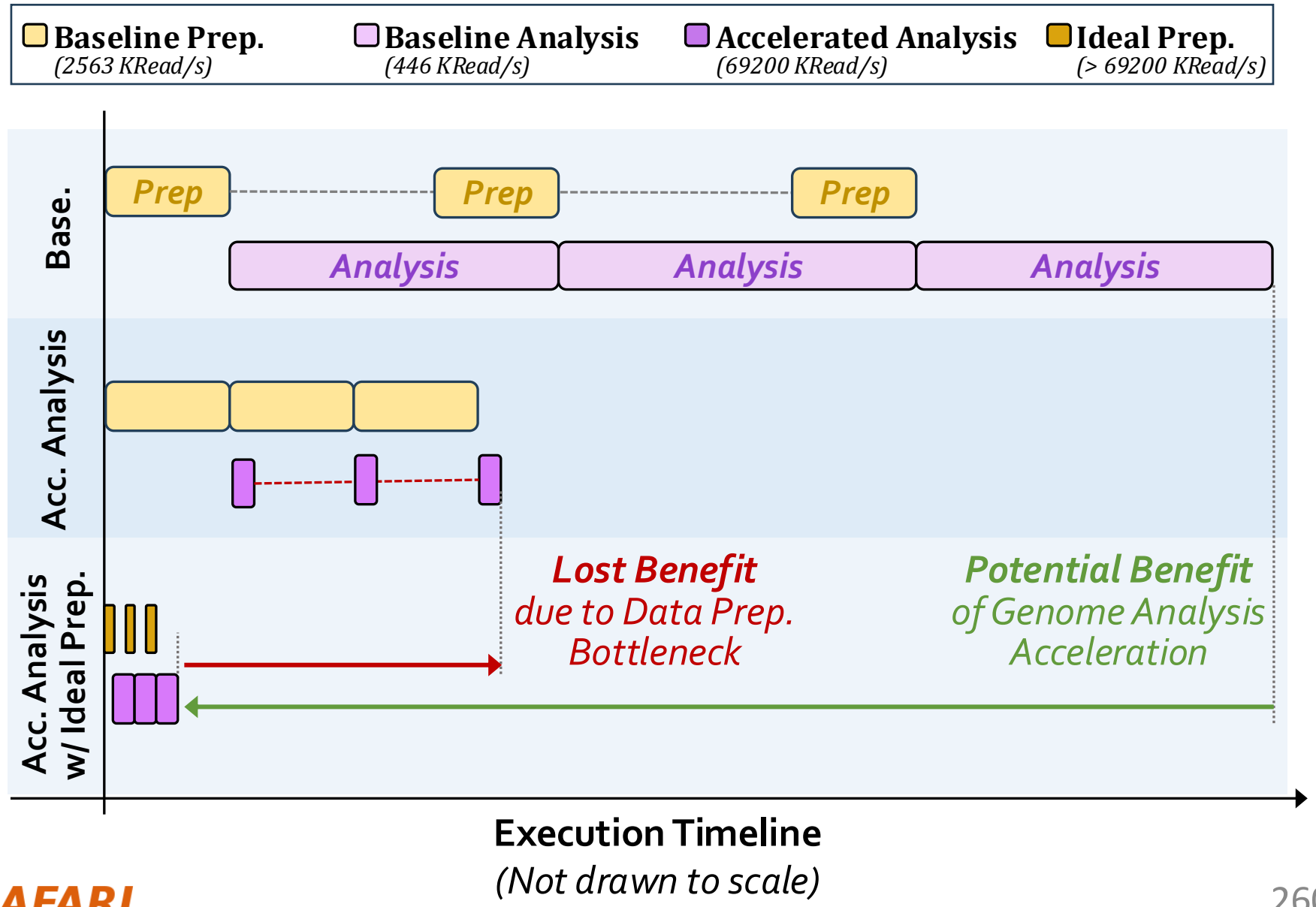


Genomic data is

decompressed *and* ***formatted*** *first*

before an accelerator can operate on it

Effect of the Data Preparation Bottleneck



Mitigating Data Preparation Bottleneck

We need to effectively mitigate the data preparation bottleneck while meeting four key requirements:

High performance

High energy efficiency

High compression ratio

Low overhead

Meeting all these requirements is challenging

Even more challenging in genome analysis systems used in hardware resource-constrained environments

*An algorithm-architecture co-design for highly-compressed **s**torage and high-performance **a**ccess of large-scale **ge**nomic sequence data*

- **Key insight**: information encoded in genomic data follows specific properties
- We leverage these properties to co-design
  - A lossless (de)compression algorithm
  - Hardware for decompression with lightweight operations
  - Storage data layout & Interface commands to access data

Higher performance *(by 3.0×–32.1×)*

Higher energy efficiency *(by 13.0×–34.0×)*

High compression ratio

Low overhead

Outline

Background

Motivation and Goal

SAGe

Evaluation

Conclusion

Genomics-Specific Compression

Read Set *(A set of reads from a sample)*

ATACGTAGAAAAAGT
TCGATGCTTGCATAA
AGCAAATGTACGATG
CATAAGTCGATAGTT
AAAAAGTCGATGCTT
...

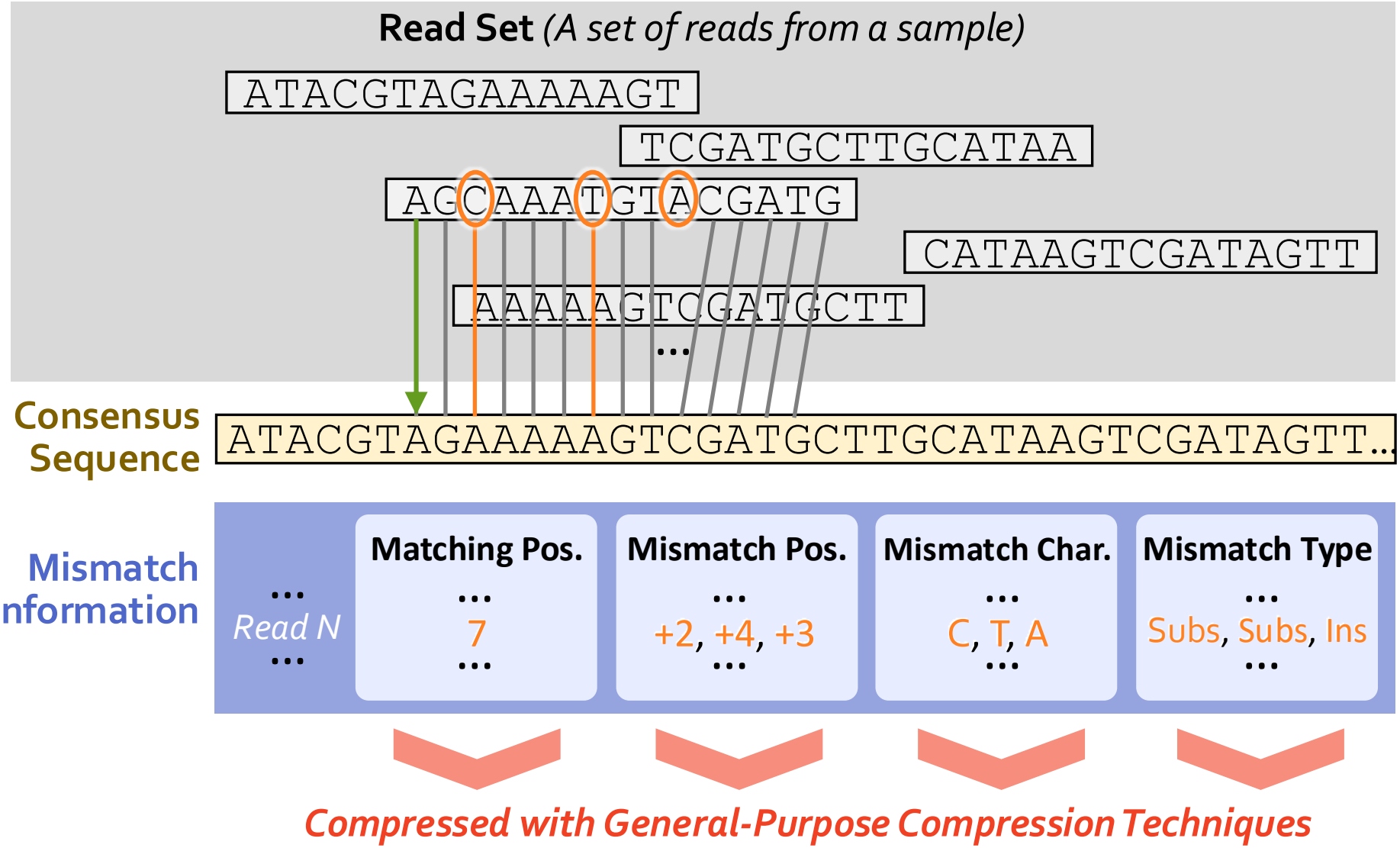
Consensus Sequence

ATACGTAGAAAAAGTCGATGCTTGCATAAGTCGATAGTT...

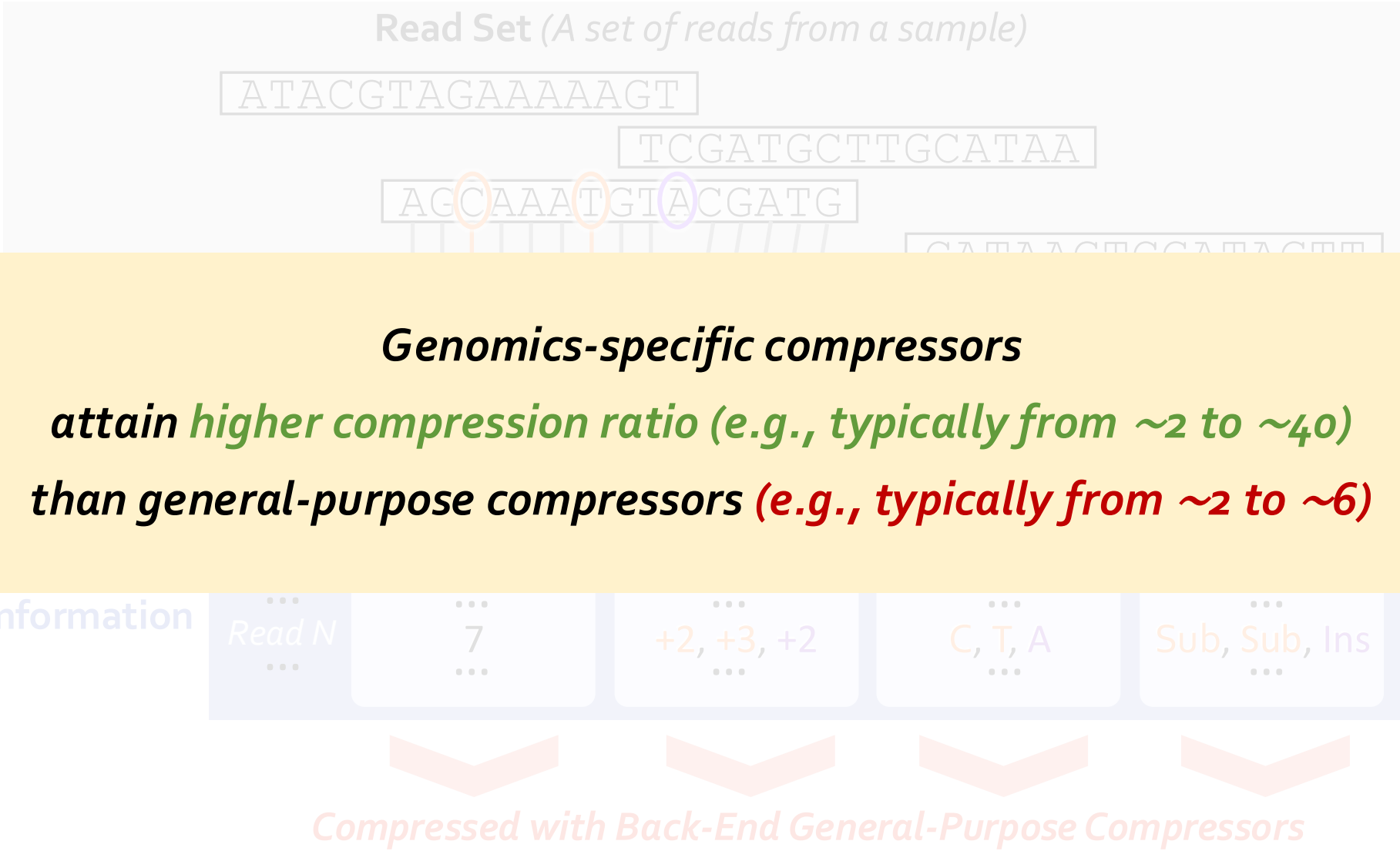
Mismatch Information

Genomics-Specific Compression

Genomics-Specific Compression



Genomics-Specific Compression



Genomics-specific compressors

attain higher compression ratio (e.g., typically from ~2 to ~40)

than general-purpose compressors (e.g., typically from ~2 to ~6)

Outline

Background

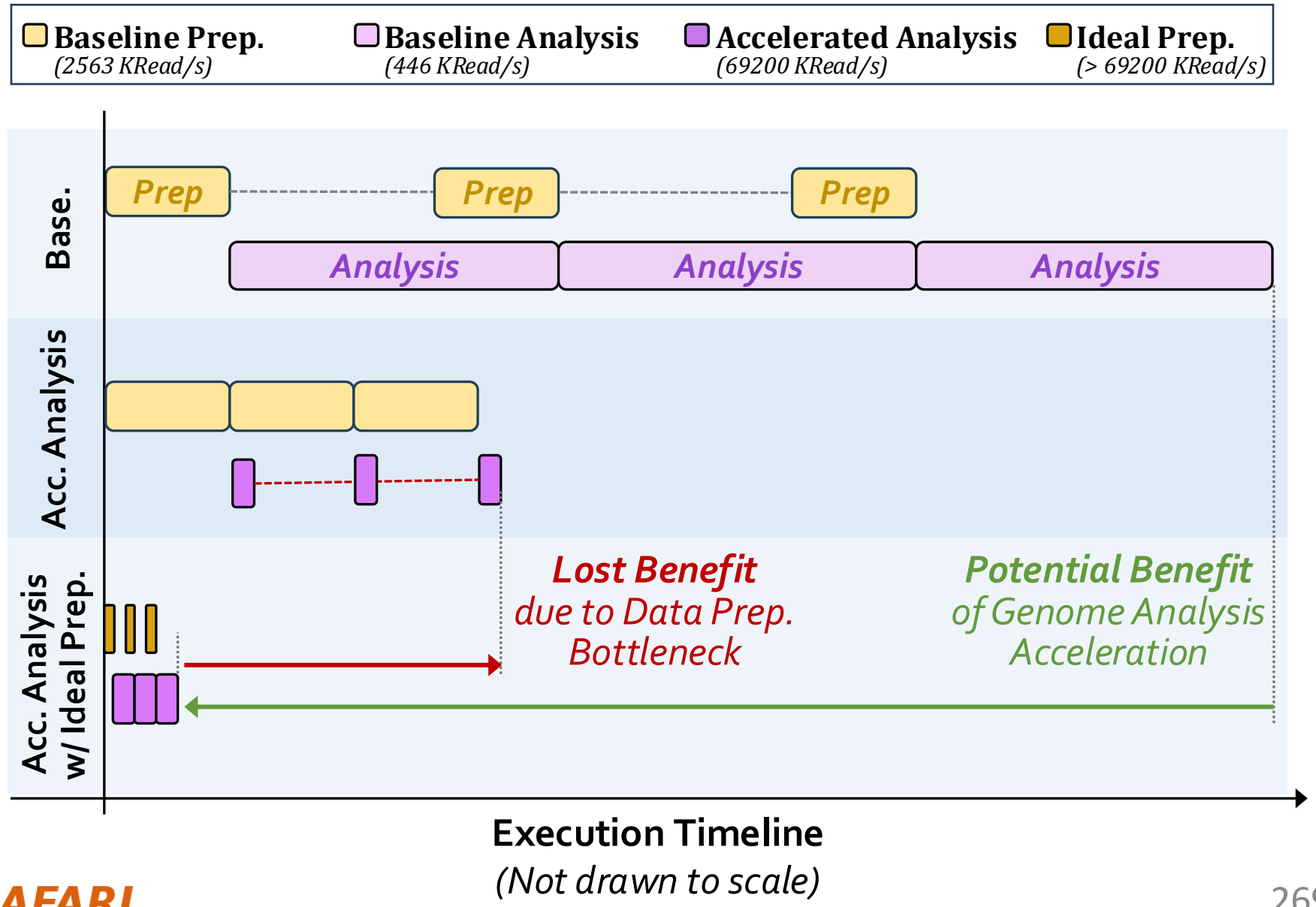
Motivation and Goal

SAGe

Evaluation

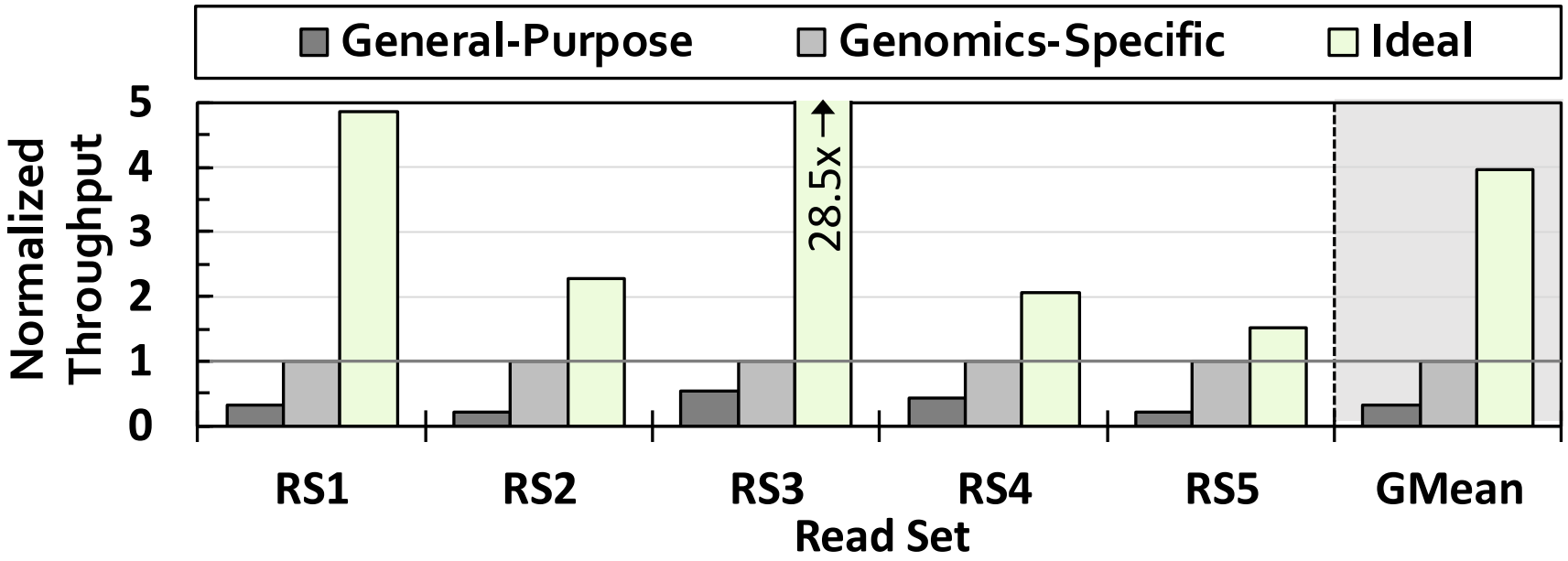
Conclusion

Effect of the Data Preparation Bottleneck



Data Preparation Bottleneck

- End-to-end performance of a genome analysis application, which includes both **data preparation** and **genome analysis**
  - Genome Analysis: GEM *[Chen+, IEEE TPDS'23]*, a state-of-the-art hardware accelerator for read mapping (a fundamental task in genome analysis)



Data preparation greatly limits end-to-end performance

Challenges

- Mitigating the data preparation bottleneck is challenging when aiming to achieve **high performance**, **energy efficiency**, **high compression ratio**, & **low overhead**

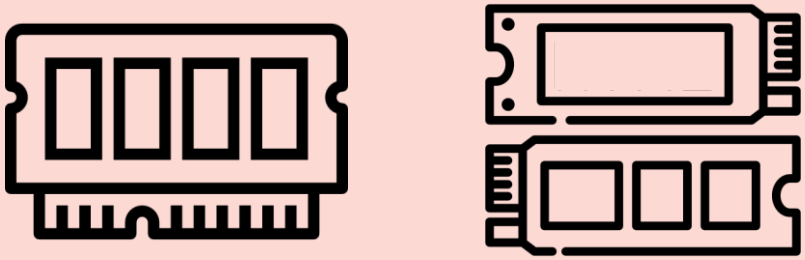
- This is because prior approaches rely on
  - **Expensive computational units**
  - **Large buffers, or large DRAM bandwidth or capacity**

- This challenge is exacerbated in genome analysis systems used in hardware **resource-constrained environments**

Portable Systems

Near-Data Processing Systems

(e.g., in Main Memory or Storage System)



Our Goal

Effectively mitigate the data preparation bottleneck, while meeting four key requirements:

High performance

Energy efficiency

High compression ratio

Low overhead

Outline

Background

Motivation and Gaol

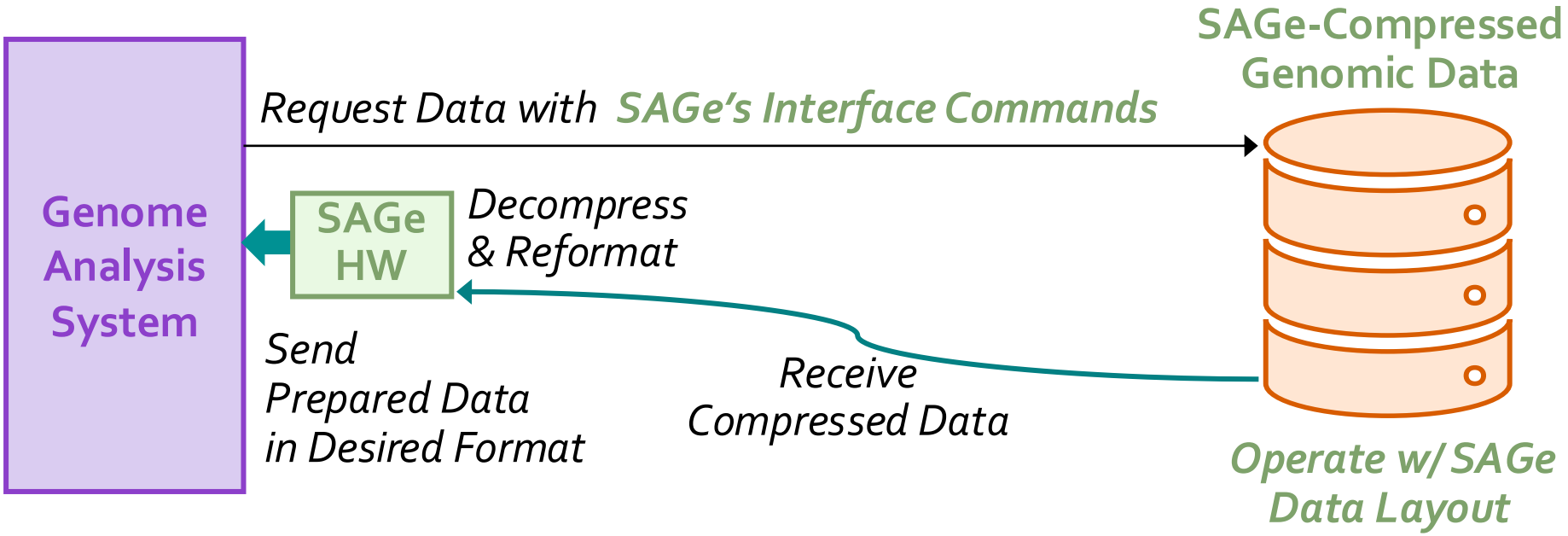
SAGe

Evaluation

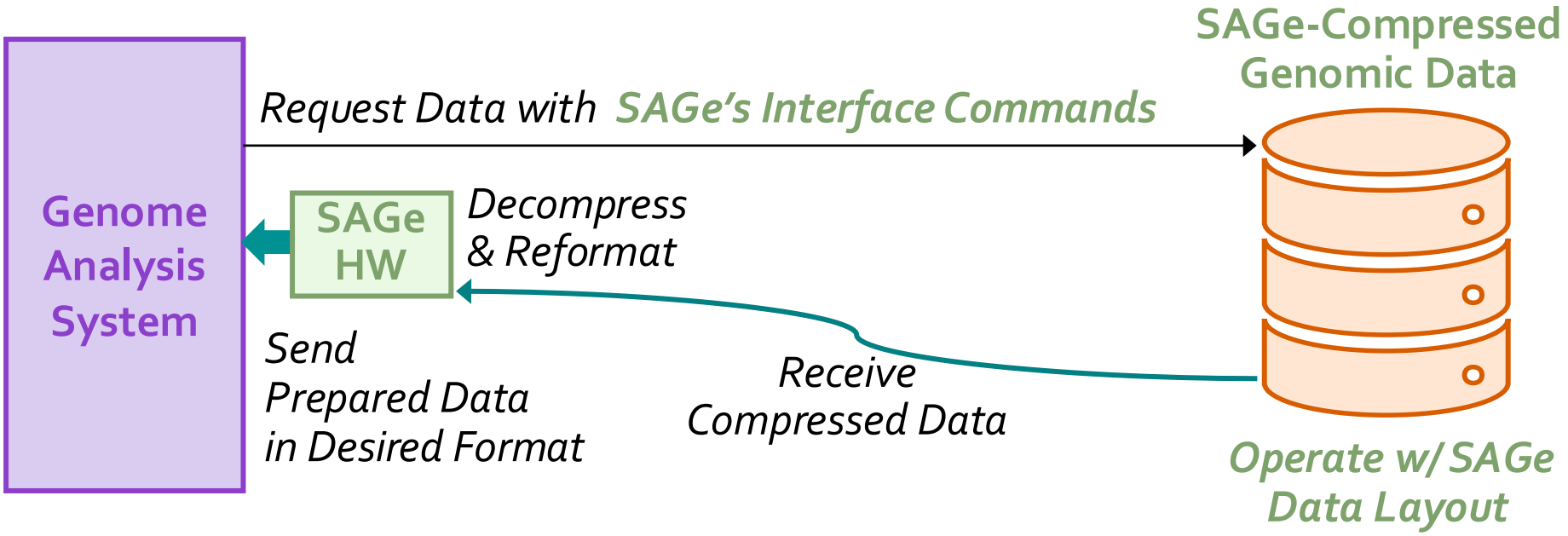
Conclusion

*An algorithm-architecture co-design for highly-compressed **s**torage and high-performance **a**ccess of large-scale **ge**nomic sequence data*

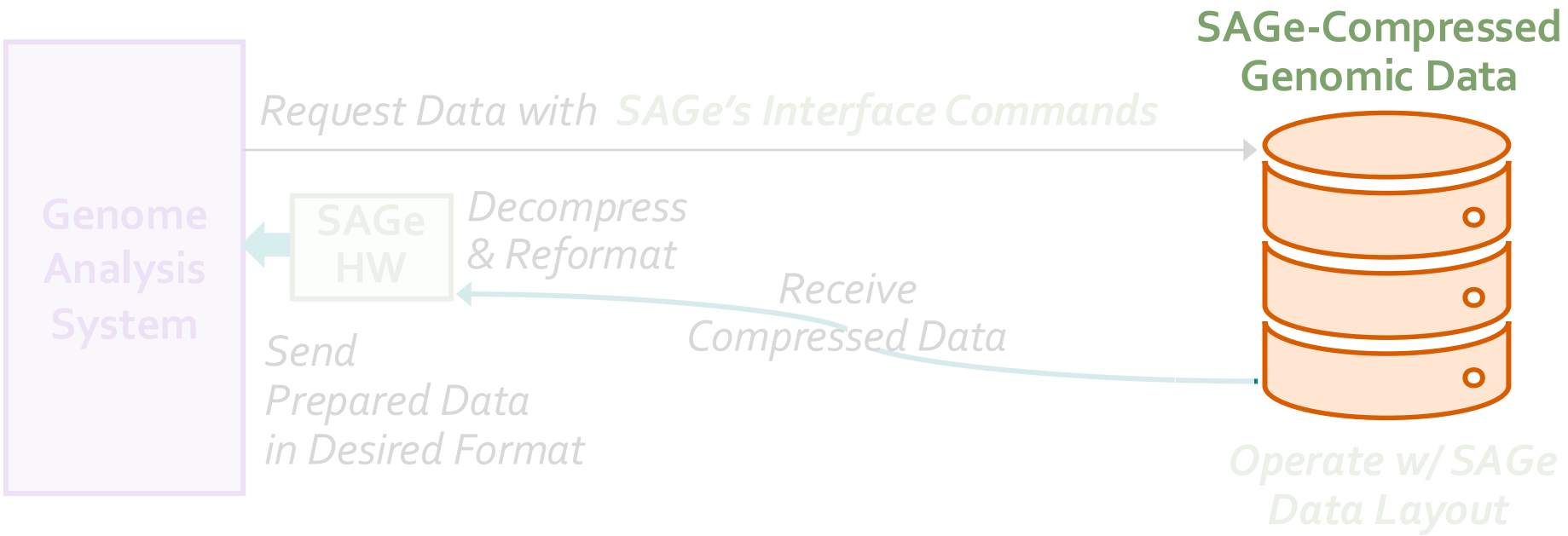
- **Key insight**: information encoded in genomic data follows specific properties
  We leverage these properties in our algorithm-architecture co-design



SAGe's Algorithm Architecture Co-Design



SAGe's Algorithm Architecture Co-Design



Compression Algorithm and Data Structures

Consensus Sequence

ATACGTAGAAAAAGTCGATGCTTGCATAAGTCGATAGTT...

Mismatch Information

	Matching Pos.	Mismatch Pos.	Mismatch Base	Mismatch Type
Read 1	7	1, 0, 9	C, T, A	Subs, Subs, Ins
Read 2	10	1, 1, 0	A, T, T	Ins, Subs, Ins
...	...	...	...	...

Compression Algorithm and Data Structures

Consensus Sequence

ATACGTAGAAAAAGTCGATGCTTGCATAAGTCGATAGTT...

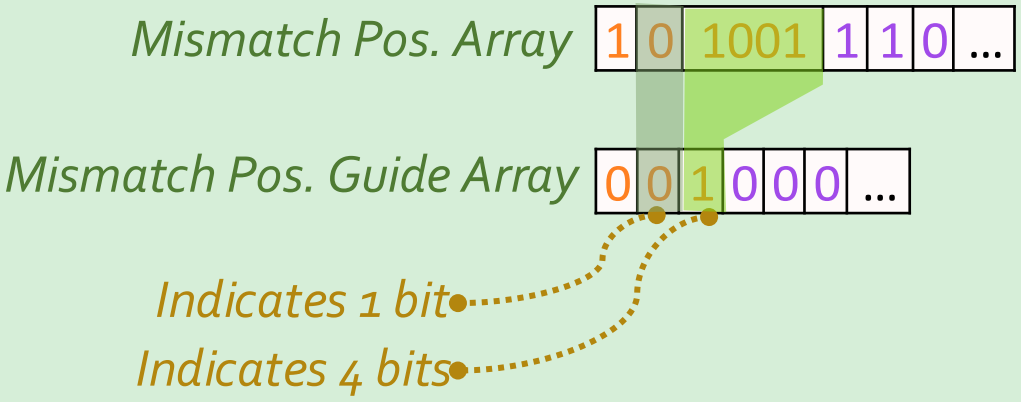
Mismatch Information

	Matching Pos.	Mismatch Pos.	Mismatch Base	Mismatch Type
Read 1	7	1, 0, 9	C, T, A	Subs, Subs, Ins
Read 2	10	1, 1, 0	A, T, T	Ins, Subs, Ins
...	...	...	...	...

1 Sequentially Encode Mismatch Information in Lightweight Structures

1	0	9	1	1	0	
0001	0000	1001	0001	0001	0000	...

2 Tune the Bit Counts Based on Read Set Properties

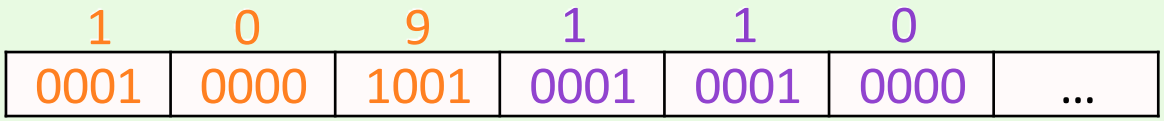


Compression Algorithm and Data Structures

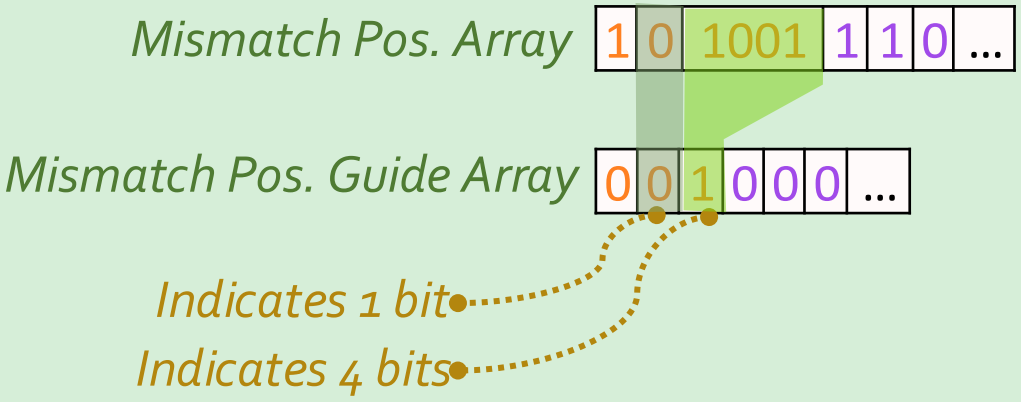
This is effective since the mismatch information in genomic datasets tends to follow specific properties

and by using a limited set of bit widths tailored to these properties, SAGe significantly reduces data size

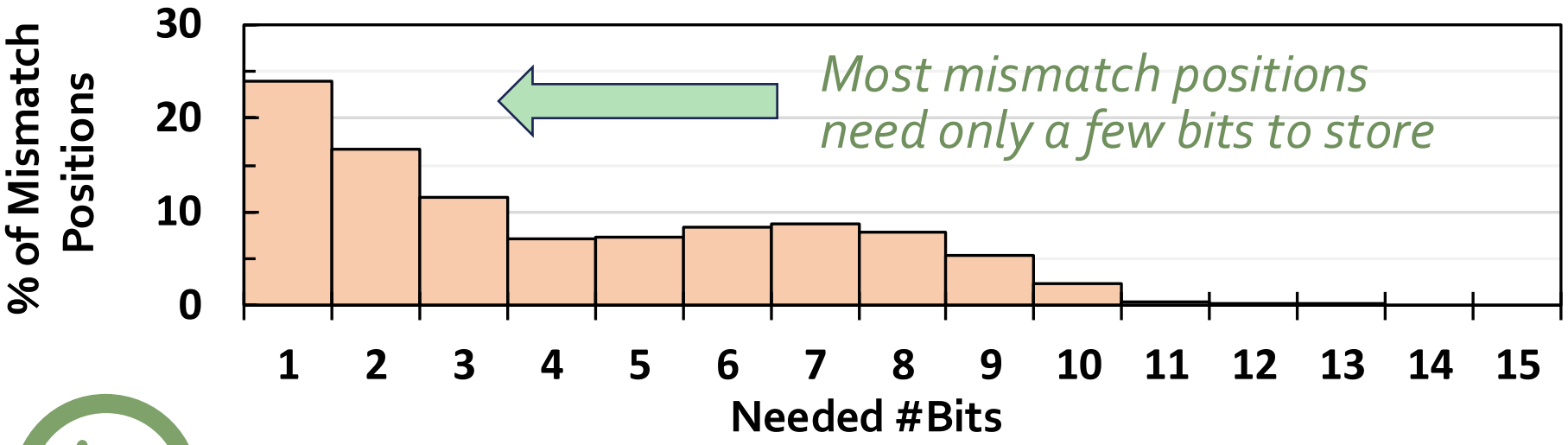
① **Sequentially Encode Mismatch Information in Lightweight Structures**



② **Tune the Bit Counts Based on Read Set Properties**



Example Properties of Mismatch Information



Based on the information encoded in histograms of ***each read set***,

SAGe systematically searches for the **best bit-counts for arrays and guide arrays**

that **minimize the total encoded size** for that specific read set

Example Properties of Mismatch Information

Observations regarding properties of

other parts of the mismatch information

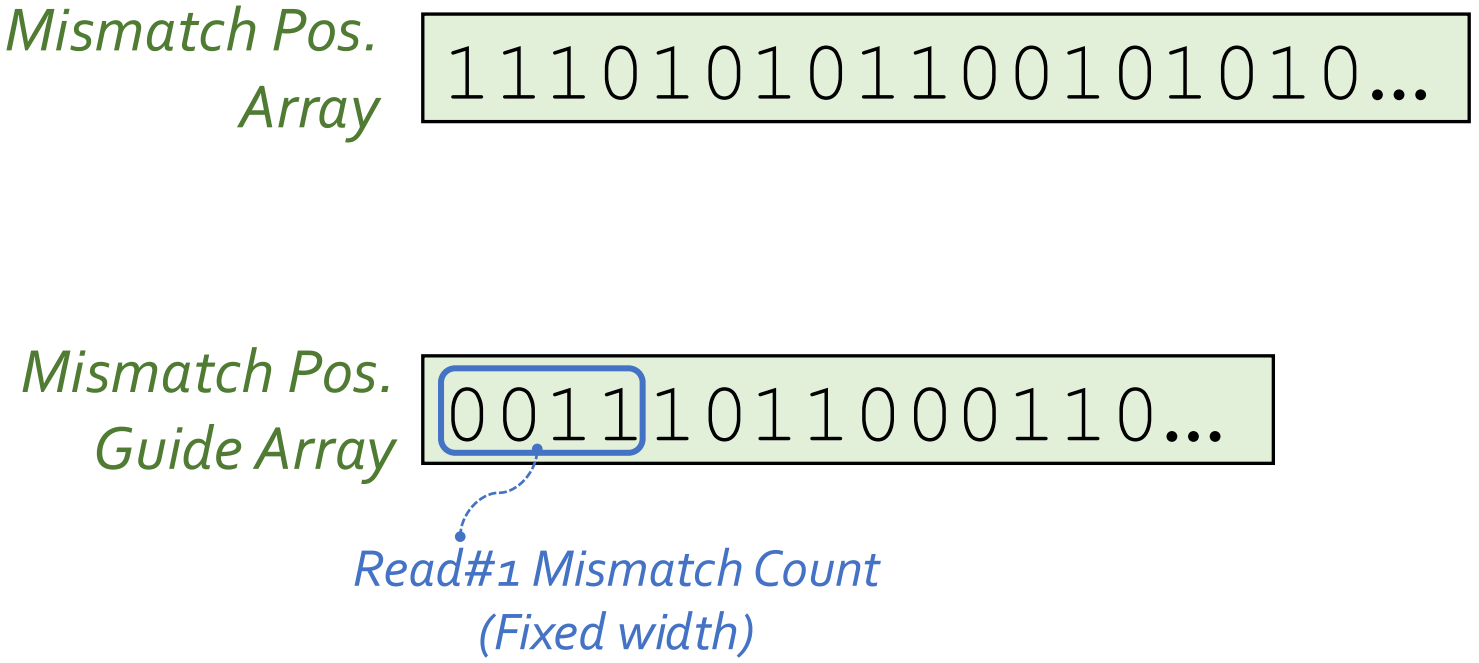
are in the paper

Based on the information encoded in histograms of
each read set,

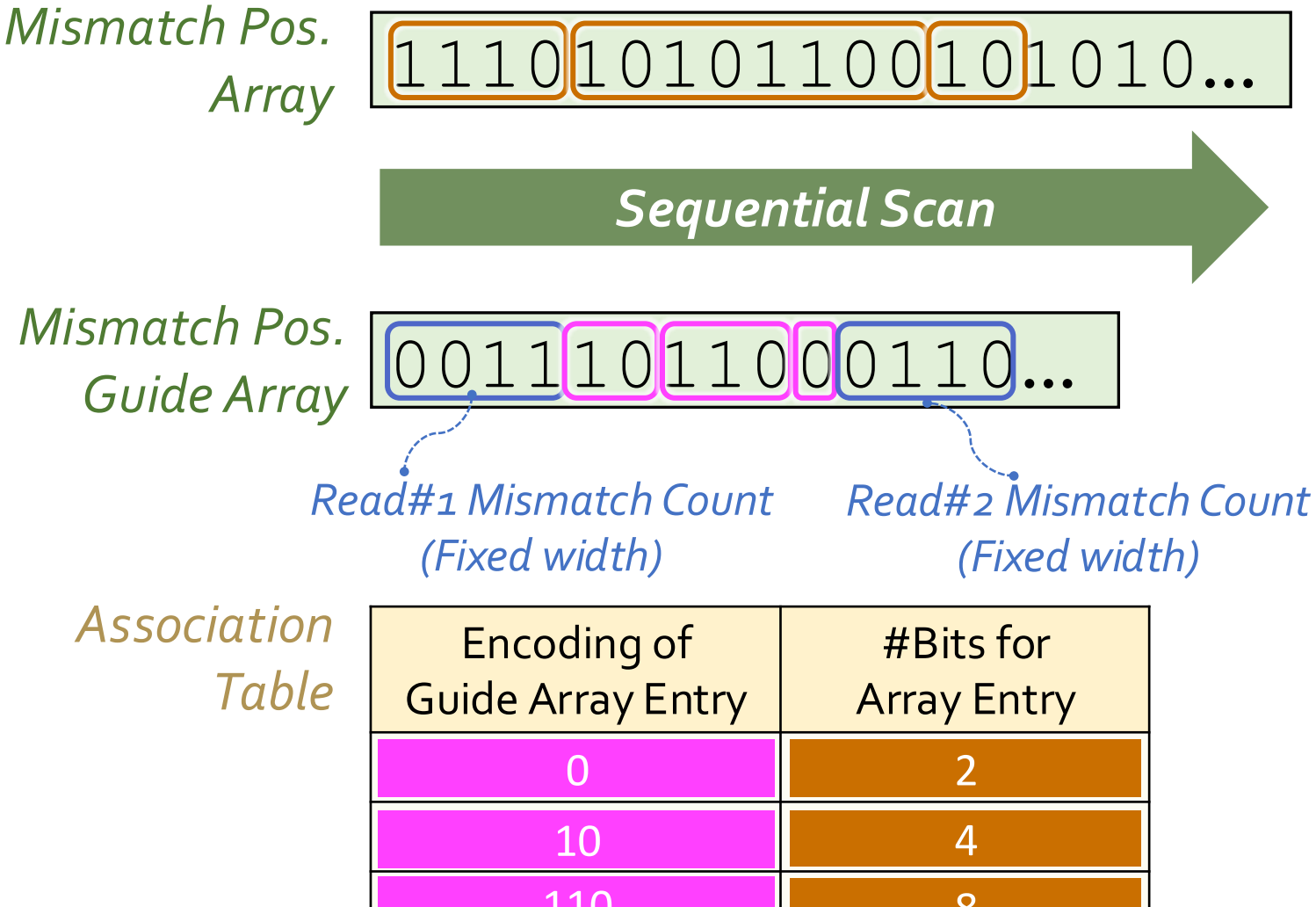
SAGe systematically searches for
the **best bit-counts for arrays and guide arrays**

that **minimize the total encoded size** for that specific read set

Example Walkthrough of Decompression



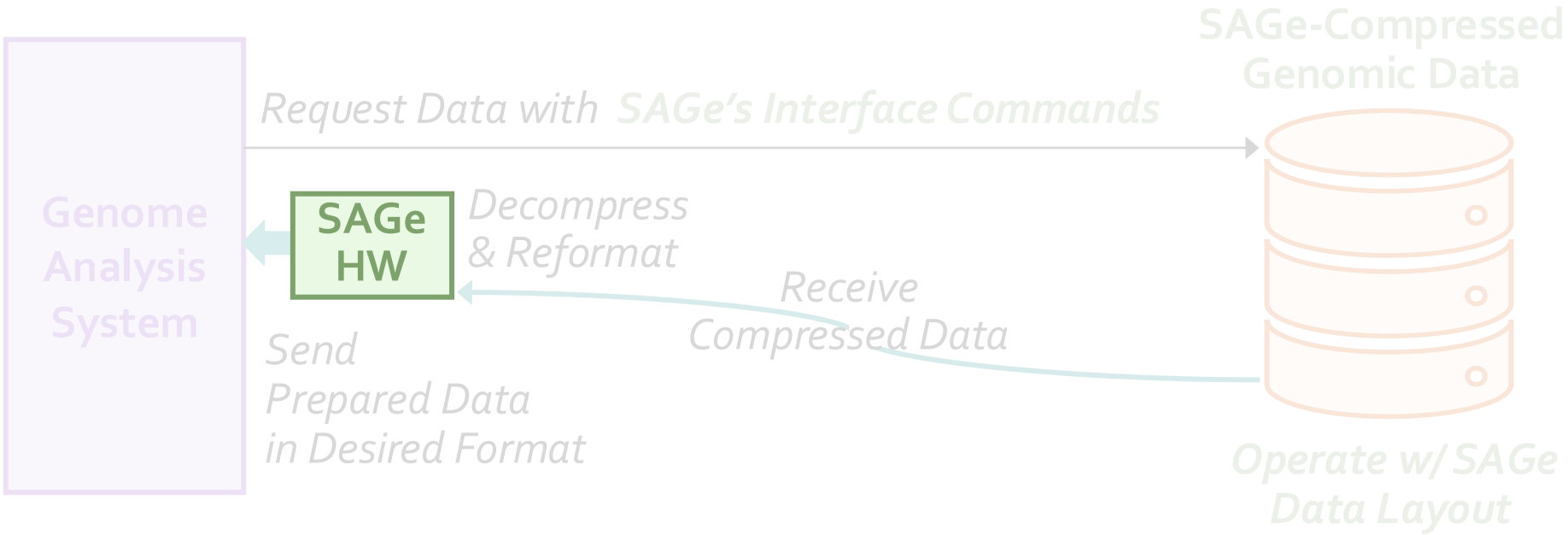
Example Walkthrough of Decompression



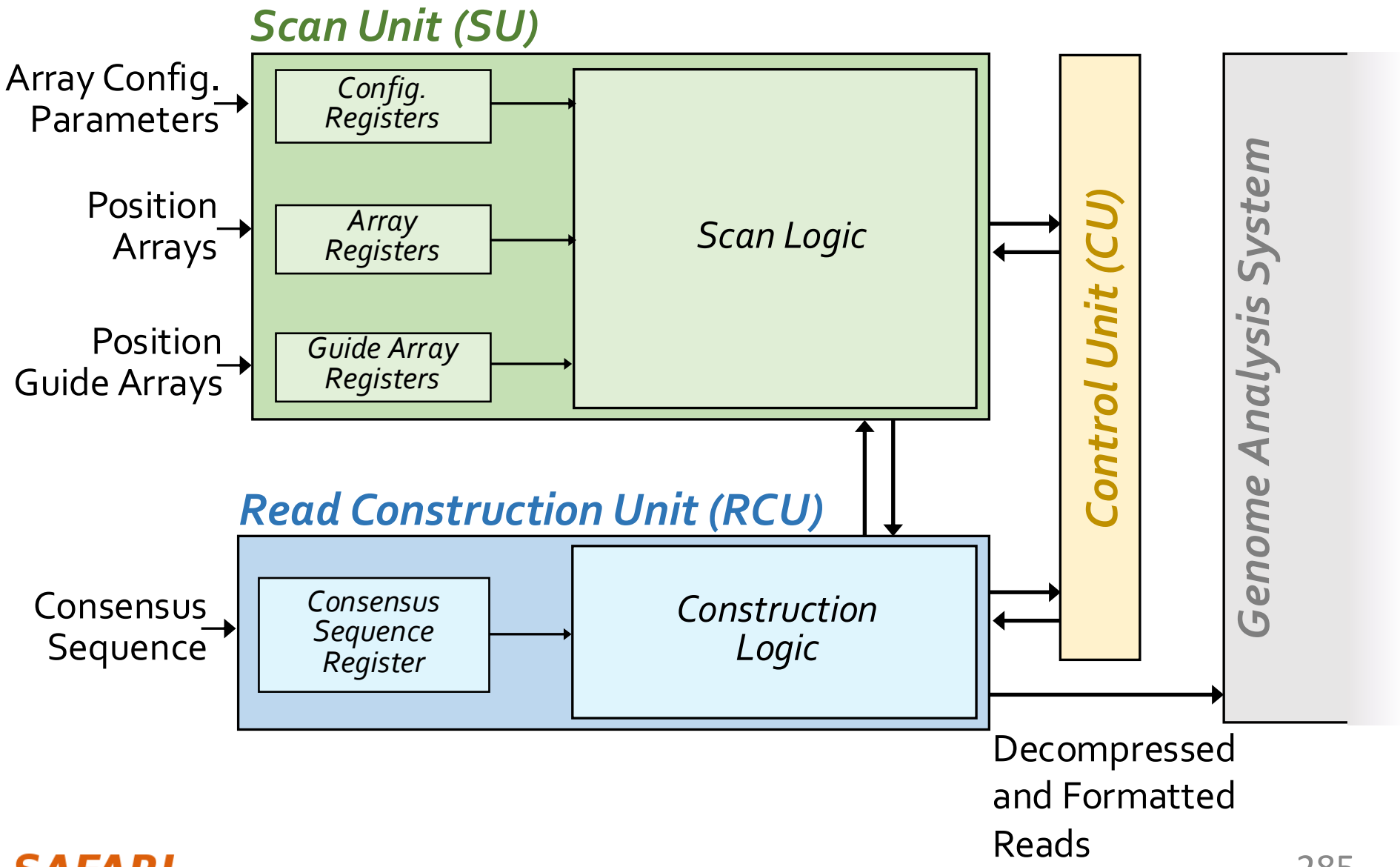
Association Table

Encoding of Guide Array Entry	#Bits for Array Entry
0	2
10	4
110	8

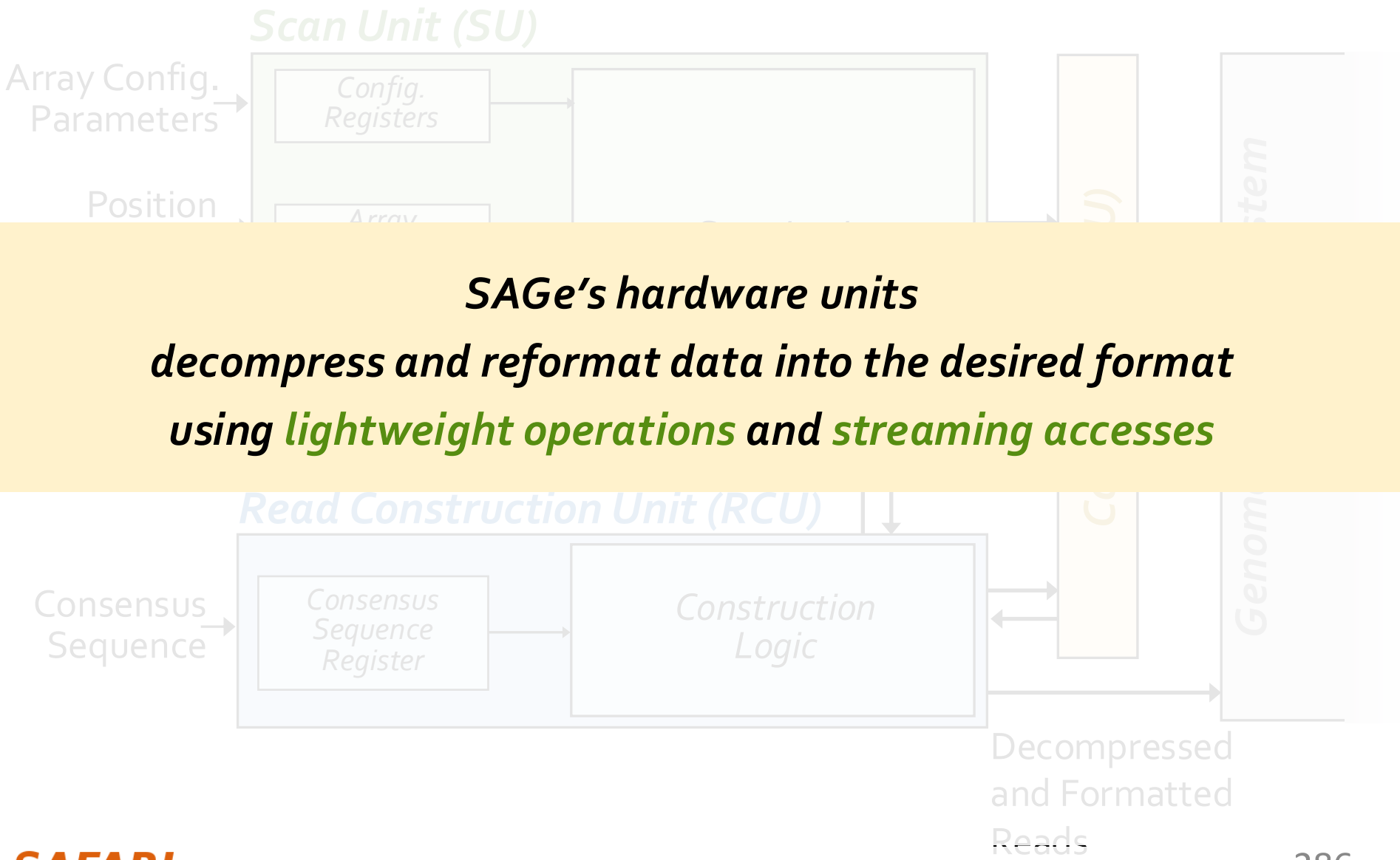
SAGe's Algorithm Architecture Co-Design



Hardware for Data Preparation



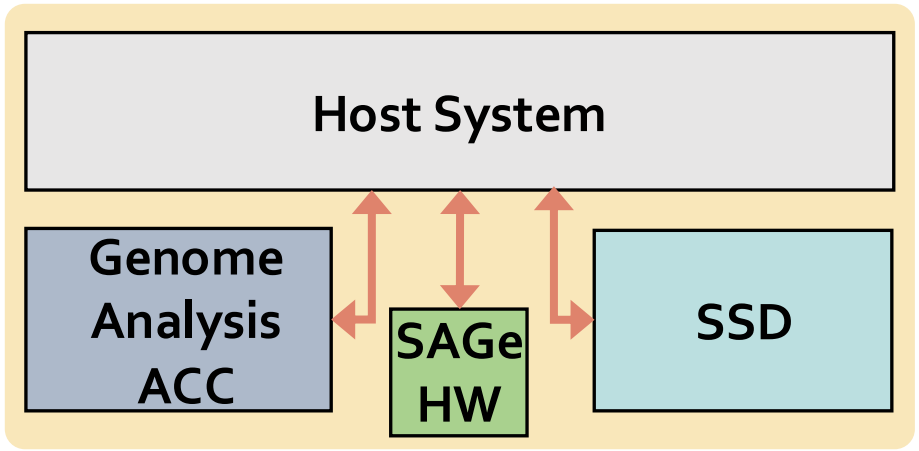
Hardware for Data Preparation

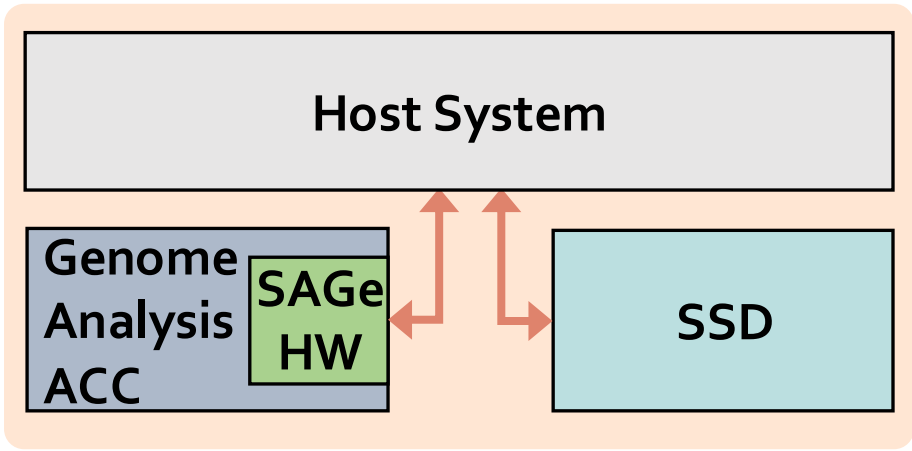


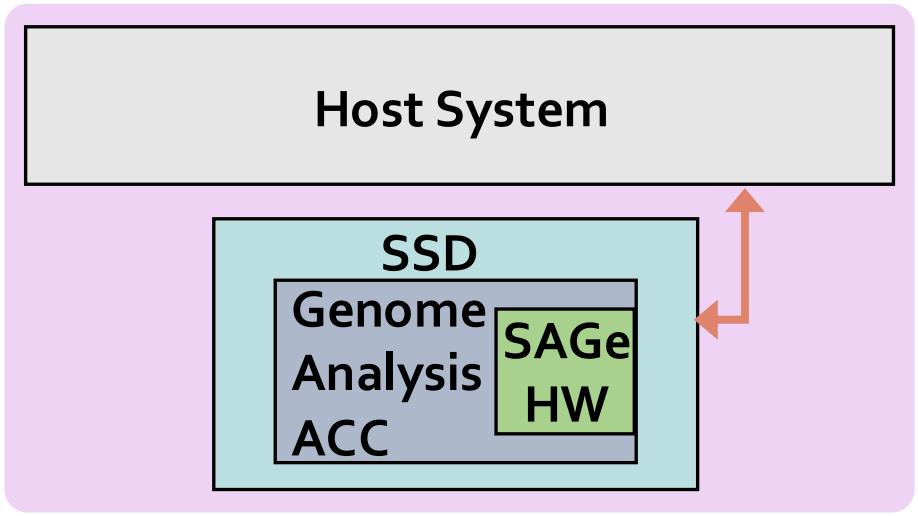
SAGe's hardware units decompress and reformat data into the desired format using lightweight operations and streaming accesses

Case Studies of SAGe's Hardware Integration

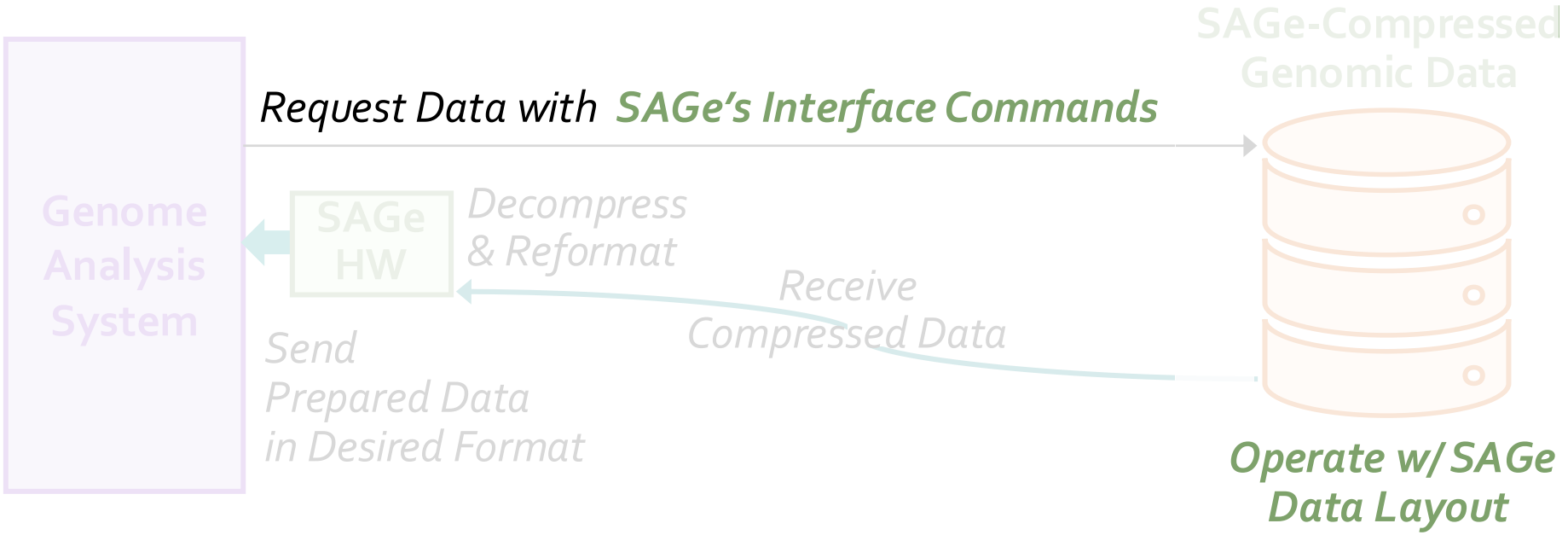
Due to its lightweight design, SAGe can seamlessly integrate with genome analysis systems







SAGe's Algorithm Architecture Co-Design



Details in the paper

More in the Paper

SAGe: A Lightweight Algorithm-Architecture Co-Design for Mitigating the Data Preparation Bottleneck in Large-Scale Genome Sequence Analysis

Nika Mansouri Ghiasi[1] Talu Güloglu[1] Harun Mustafa[1] Can Firtina[1,2]
Konstantina Koliogeorgi[1] Konstantinos Kanellopoulos[1] Haiyu Mao[3] Rakesh Nadig[1]
Mohammad Sadrosadati[1] Jisung Park[4] Onur Mutlu[1]

[1]ETH Zürich [2]University of Maryland [3]King's College London [4]POSTECH

https://arxiv.org/abs/2504.03732

Outline

Background

Motivation and Gaol

SAGe

Evaluation

Conclusion

Evaluation Methodology Overview (I)

Performance, Energy, and Power Analysis

Hardware Components

- Synthesized Verilog model for the hardware units
- MQSim *[Tavakkol+, FAST'18]* for SSD's internal operations
- Ramulator *[Kim+, CAL'15]* for SSD's internal DRAM

Software Components

Measure on a real system:

- AMD® EPYC® CPU with 128 physical cores
- 1 TB DRAM

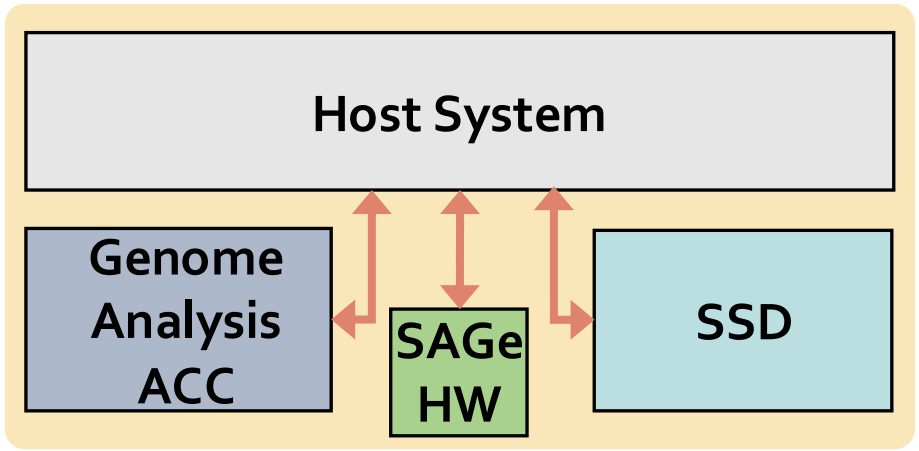
Datasets: Read sets (RS)

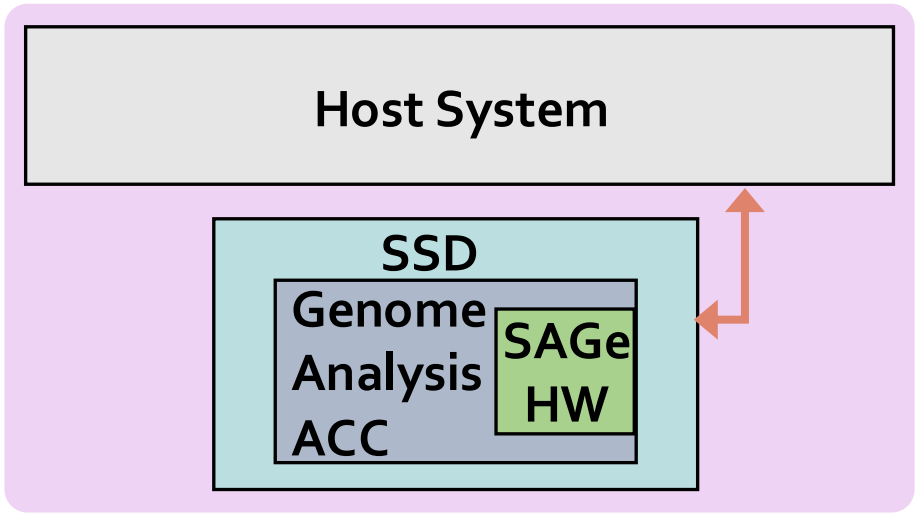
- From a variety of species (i.e., *H. sapiens*, *T. cacao*, *M. acuminata malaccensis*)
- Sequenced with different sequencing technologies (i.e., Illumina and Oxford Nanopore Technologies)

End-to-end performance of a genome analysis application, which includes both **data preparation** and **genome analysis**

Recall: Case Studies of SAGe's Hardware Integration

Due to its lightweight design, SAGe can seamlessly integrate with genome analysis systems

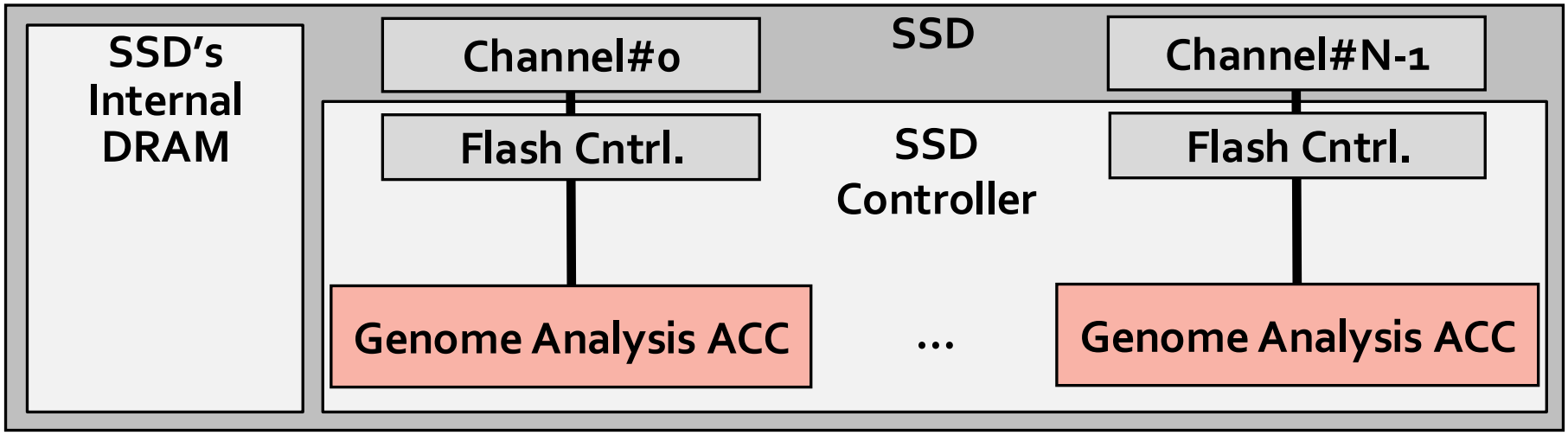




Evaluation Methodology Overview (II)

Genome Analysis

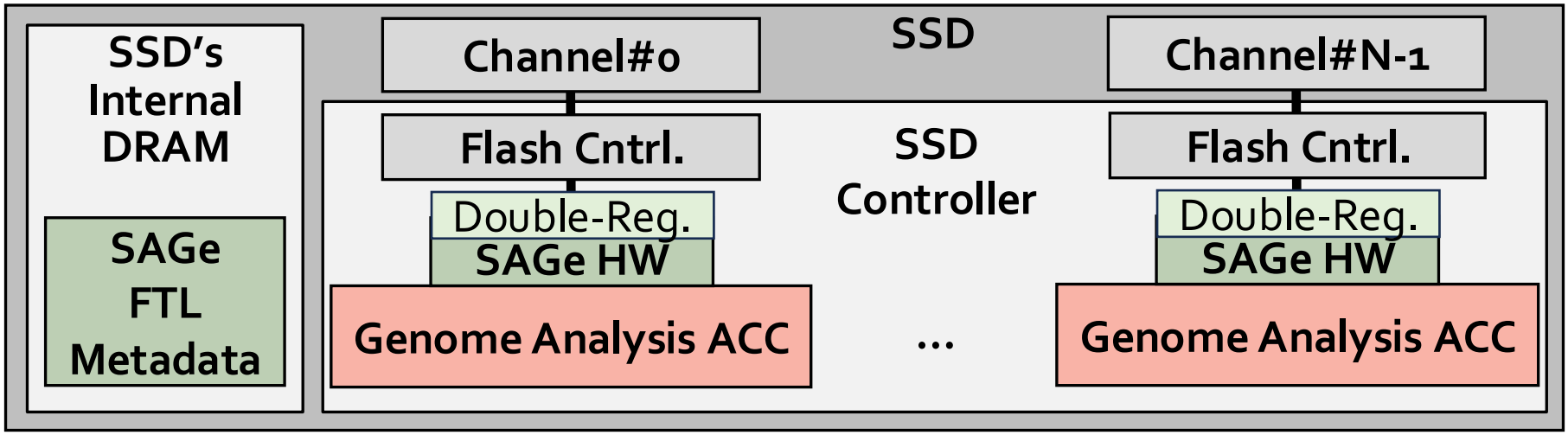
- A state-of-the-art hardware accelerator (GEM *[Chen+, TPDS'23]*), for read mapping
- A state-of-the-art NDP accelerator (GenStore *[Mansouri Ghiasi+, ASPLOS'22]*)
  - Filters reads that do not require expensive computation in the SSD ➔ Alleviates the burden of moving and analyzing a large amount of low-reuse data from the rest of the system
  - Implemented in the resource-constrained environment of the SSD



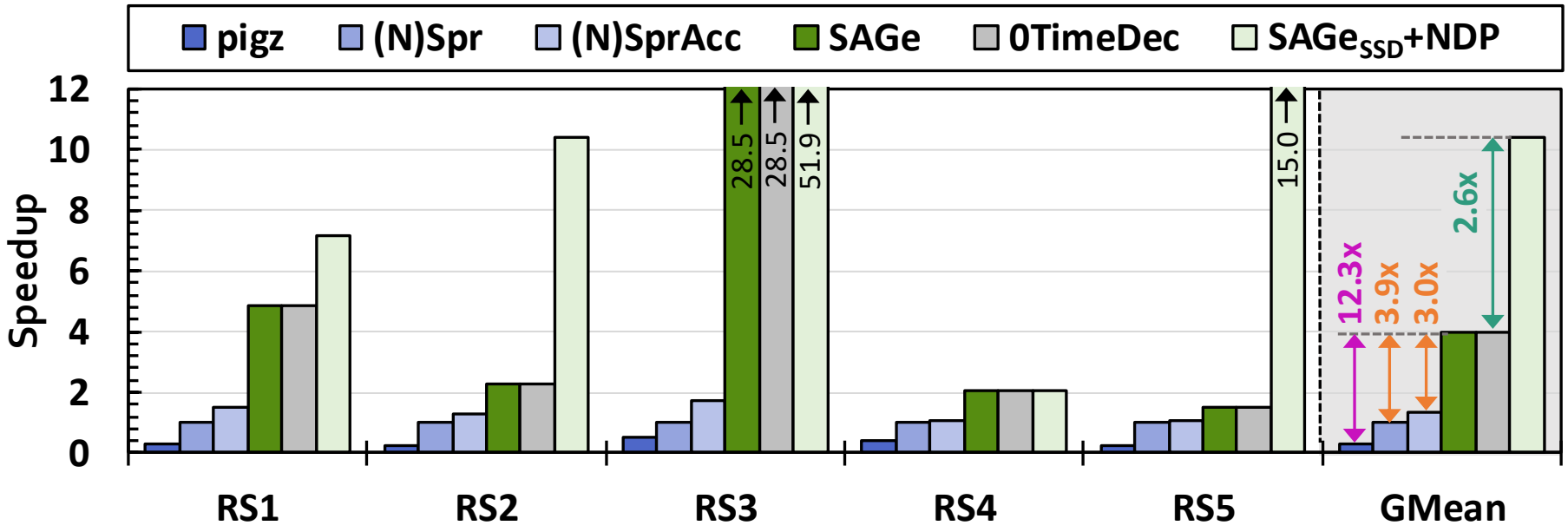
Evaluation Methodology Overview (II)

Data Preparation

- **pigz:** Commonly-used general-purpose software (*[Adler, JPL'15]*)
- **(N)Spr:** Genomics-specific software for short (Spring *[Chandak, Bioinformatics'18]*) and long reads (NanoSpring *[Meng+, Scientific Reports'23]*)
- **(N)SprAcc:** (N)Spring hardware-accelerated
- **0TimeDec:** an idealized decompressor (with *zero* decompression time), but inefficient for integration in resource-constrained environments
- **SAGe:** SAGe implemented as an standalone accelerator for data preparation
- **$SAGe_{SSD}$:** SAGe inside the SSD, for integration with the NDP analysis system



Speedup



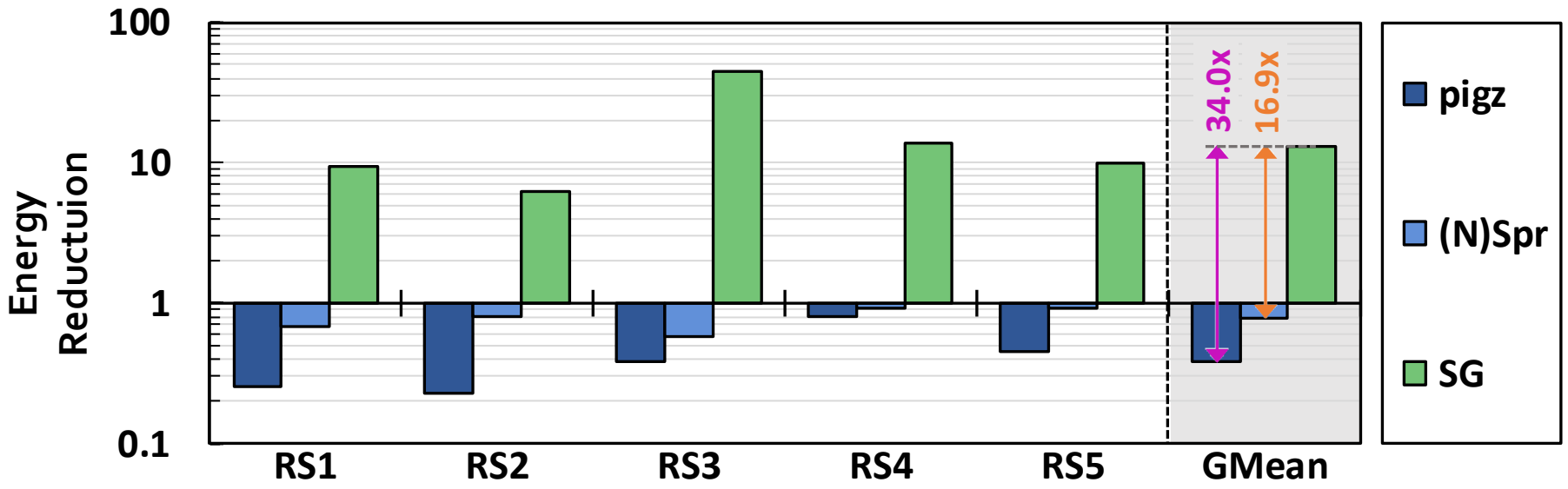
SAGe provides significant speedup over both general-purpose and genomics-specific baselines

SAGe achieves the same speedup as 0TimeDec

By efficiently integrating SAGe$_{SSD}$ with NDP SAGe$_{SSD}$+NDP leads to large speedups

Reduction in Energy Consumption

Normalized to (N)SprAcc (higher is better)



SAGe provides significant energy reduction over both general-purpose and genomics-specific baselines

Compression Ratio

Read Set	Uncompressed Size (MB)	Compression Ratio		
		pigz	(N)Spr	SAGe
RS1	5000	3.4	24.8	22.8
RS2	79000	12.5	40.2	36.8
RS3	4000	3.4	7.2	7.1
RS4	12000	3.9	4.8	4.5
RS5	88400	3.5	7.6	7.8
RS6	9600	3.27	18.5	18.9

2.9× better average compression ratio than general-purpose baseline

comparable compression ratio to genomics-specific baseline
(with a modest 4.6% average reduction)

Compression Ratio for Metagenomic Data

Read Set	pigz	(N)Spr	SAGe
CAMI-L	3.53	21.54	32.53
CAMI-M	3.59	17.65	26.20
CAMI-H	3.54	7.35	9.53

Better average compression ratio than general-purpose baseline

Comparable compression ratio to genomics-specific baseline

Area and Power

- Based on **Synthesis** of **SAGe's hardware units** using the Synopsys Design Compiler @ 22nm technology node

Logic unit	# of instances	Area [mm²]	Power [mW]
Comparator	1 per channel	0.000045	0.014
Read Construction Unit	1 per channel	0.000017	0.023
Double Registers	1 per channel	0.00020	0.035
Control Unit	1 per channel	0.000029	0.025
Total for an 8-channel SSD	-	0.002	0.77

SAGe's hardware units consume small area and power

More in the Paper

- Observations on different parts of **mismatch information**
  - Matching positions
  - Mismatch bases and types
  - Mismatch counts
- SAGe's performance in **software**
- Sensitivity analysis with **varying the number of SSD**
- Impact of **each of SAGe's optimizations** on compression ratio
- SAGe's performance for **quality scores**
- SAGe's **resource requirements** compared to other (de)compression accelerators
- SAGe's compression time

More in the Paper

SAGe: A Lightweight Algorithm-Architecture Co-Design for Mitigating the Data Preparation Bottleneck in Large-Scale Genome Sequence Analysis

Nika Mansouri Ghiasi[1] Talu Güloglu[1] Harun Mustafa[1] Can Firtina[1,2]
Konstantina Koliogeorgi[1] Konstantinos Kanellopoulos[1] Haiyu Mao[3] Rakesh Nadig[1]
Mohammad Sadrosadati[1] Jisung Park[4] Onur Mutlu[1]

[1]ETH Zürich [2]University of Maryland [3]King's College London [4]POSTECH

https://arxiv.org/abs/2504.03732

Outline

Background

Motivation and Gaol

SAGe

Evaluation

Conclusion

Conclusion

Data preparation bottleneck
greatly limits end-to-end performance of genome analysis

SAGe

An algorithm-architecture co-design for highly-compressed storage and high-performance access of large-scale genomic sequence data

Improves performance

Improves the end-to-end performance of two state-of-the-art genome analysis accelerators by 3.0×–32.1×

High compression ratio

2.9× better than general-purpose baseline and comparable to genomics-specific baseline (with a modest 4.6% average reduction)

Reduces energy consumption

Improves the end-to-end energy efficiency of two state-of-the-art genome analysis accelerators by 13.0×–34.0×

Low overhead

Area: 0.002 mm^2

Power: 0.77 mW

A Lightweight Algorithm-Architecture Co-Design for Mitigating the Data Preparation Bottleneck in Large-Scale Genome Sequence Analysis

Nika Mansouri Ghiasi

Talu Güloglu Harun Mustafa Can Firtina Konstantina Koliogeorgi

Konstantinos Kanellopoulos Haiyu Mao Rakesh Nadig

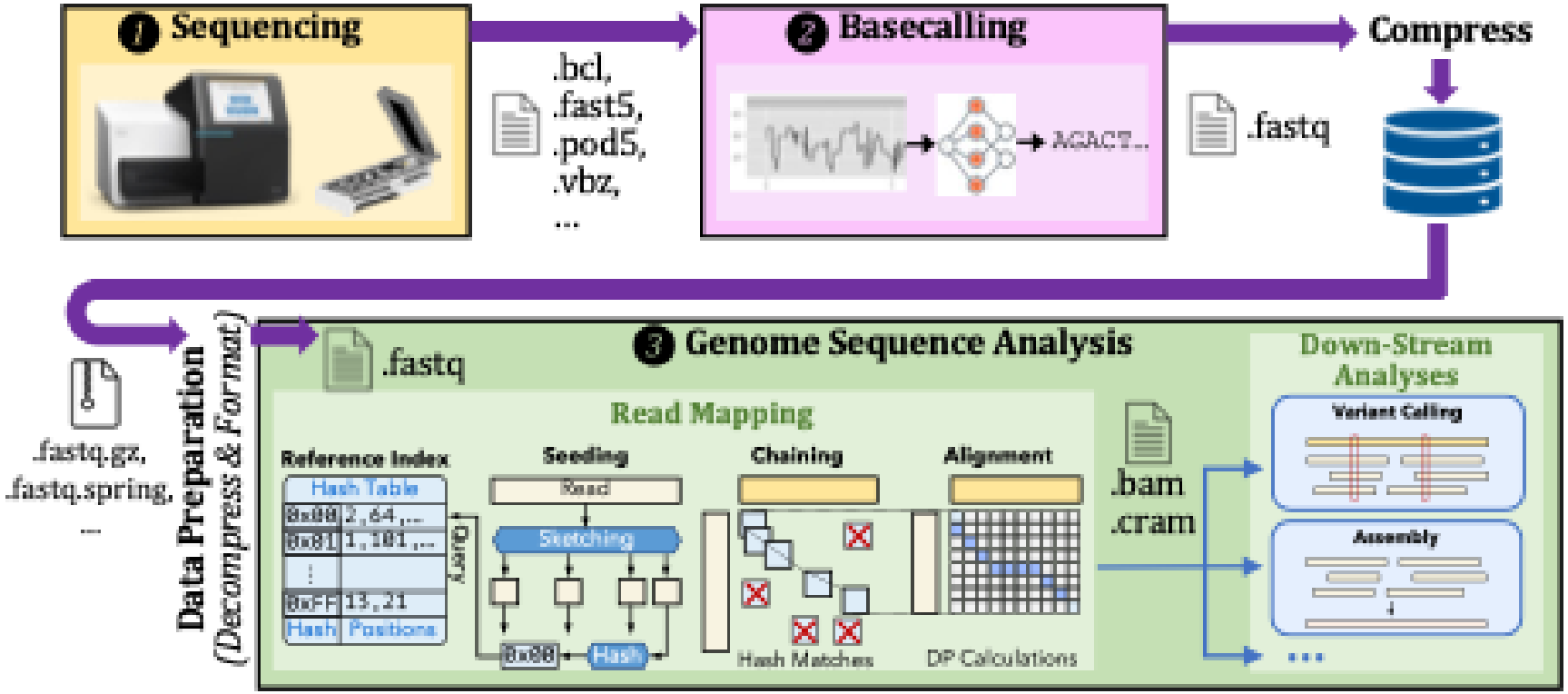
Mohammad Sadrosadati Jisung Park Onur Mutlu

SAFARI

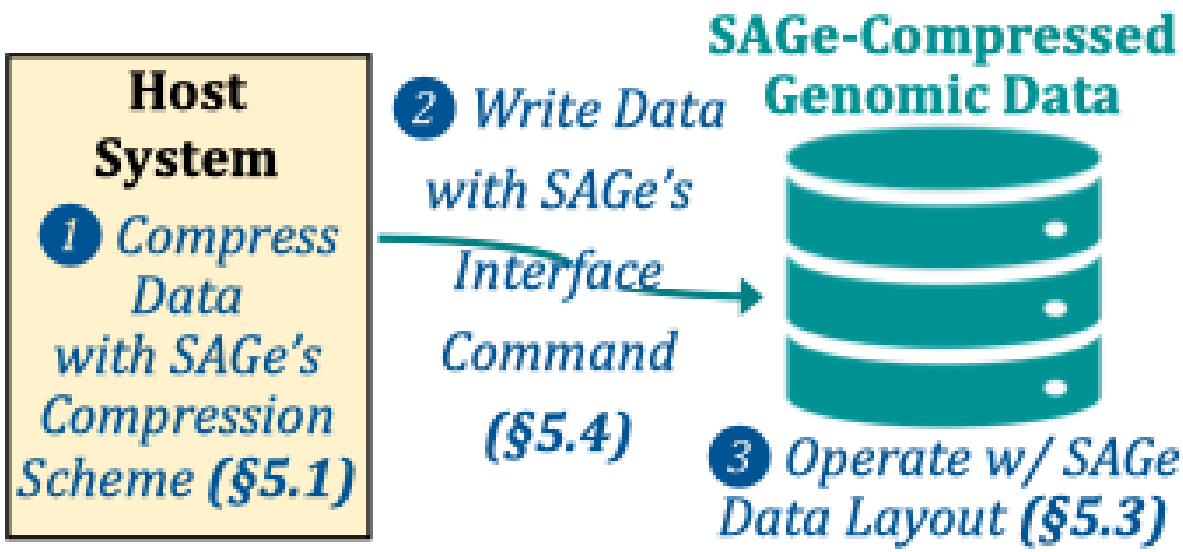
ETH zürich UNIVERSITY OF MARYLAND KING'S College LONDON POSTECH

Backup Slides

Genomic Workflow

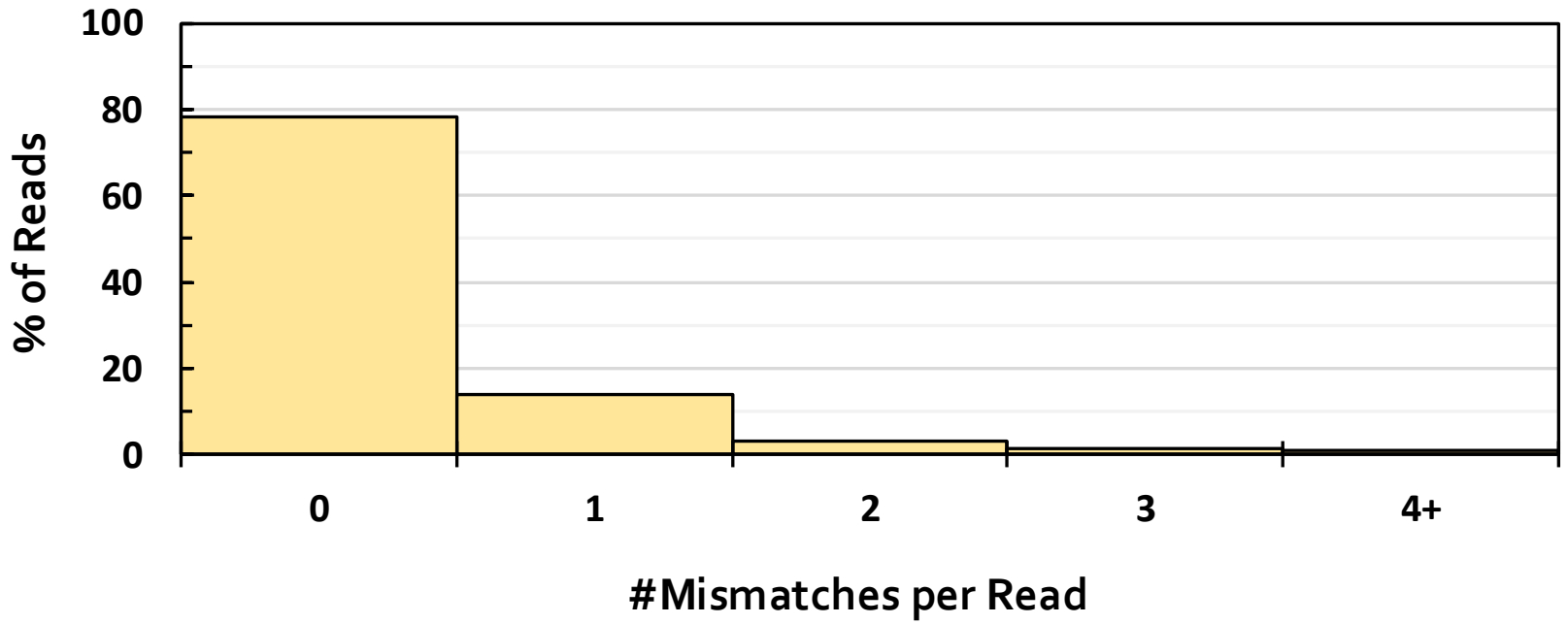


SAGe Compression High-Level Overview



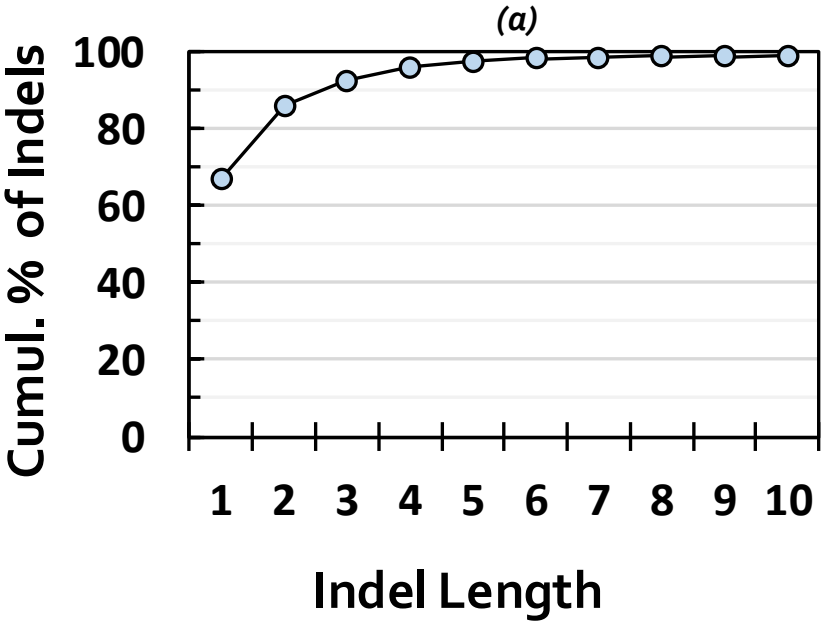
Properties of Mismatch Information (I)

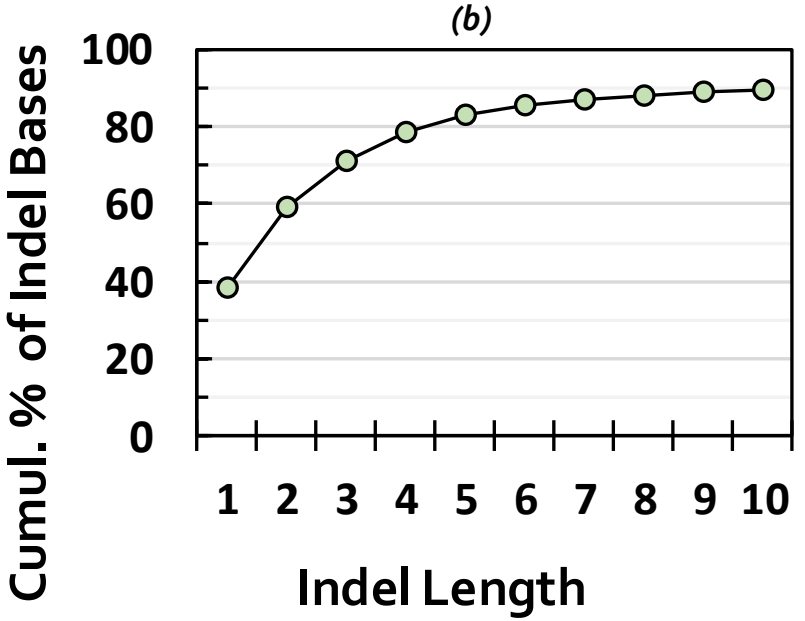
Distribution of mismatch counts per short read



Properties of Mismatch Information (II)

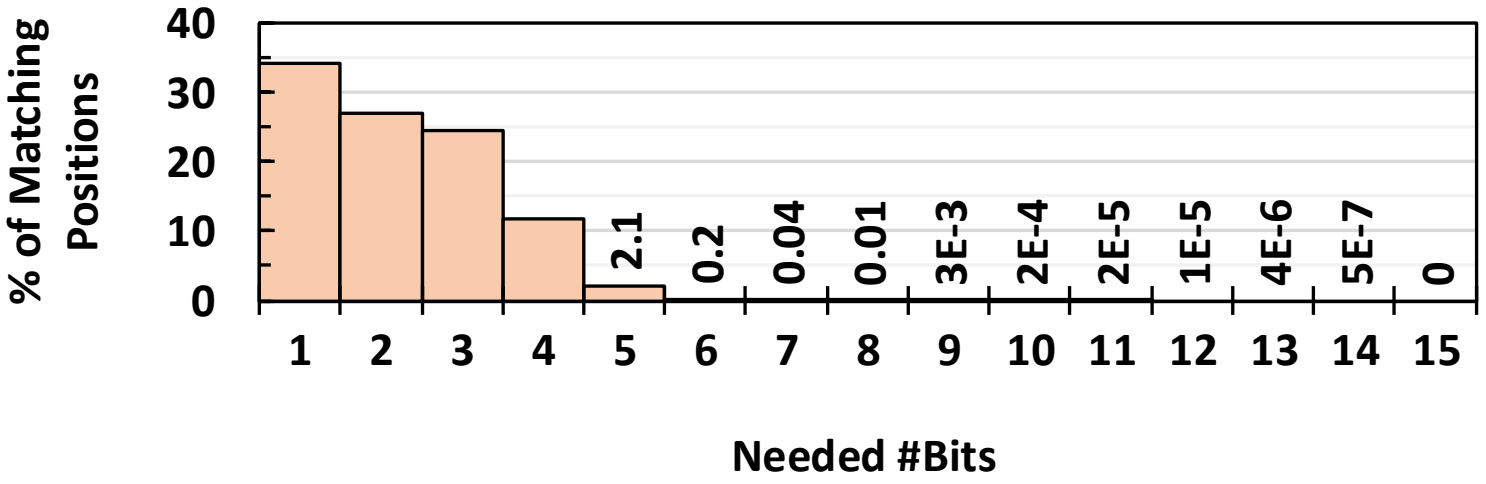
Cumulative distribution of
(a) indel block lengths and
(b) #bases in indel blocks of different lengths





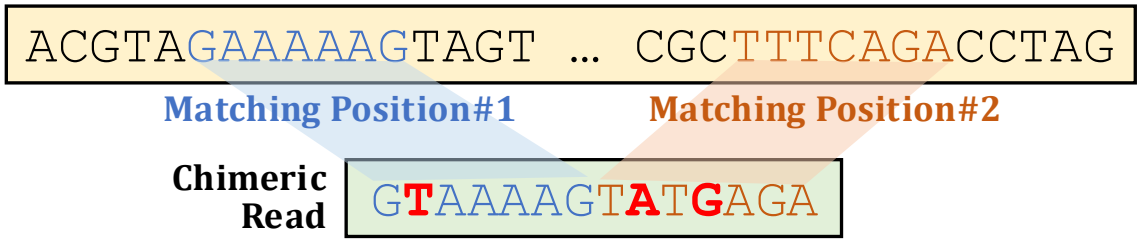
Properties of Mismatch Information (III)

Distribution of #bits needed to store the delta-encoded matching position.



Example of a chimeric read

Chimeric reads: reads with sequences joined from different regions of the genome due to sequencing/library preparation errors, or structural variations



Tuning Arrays and Guide Arrays

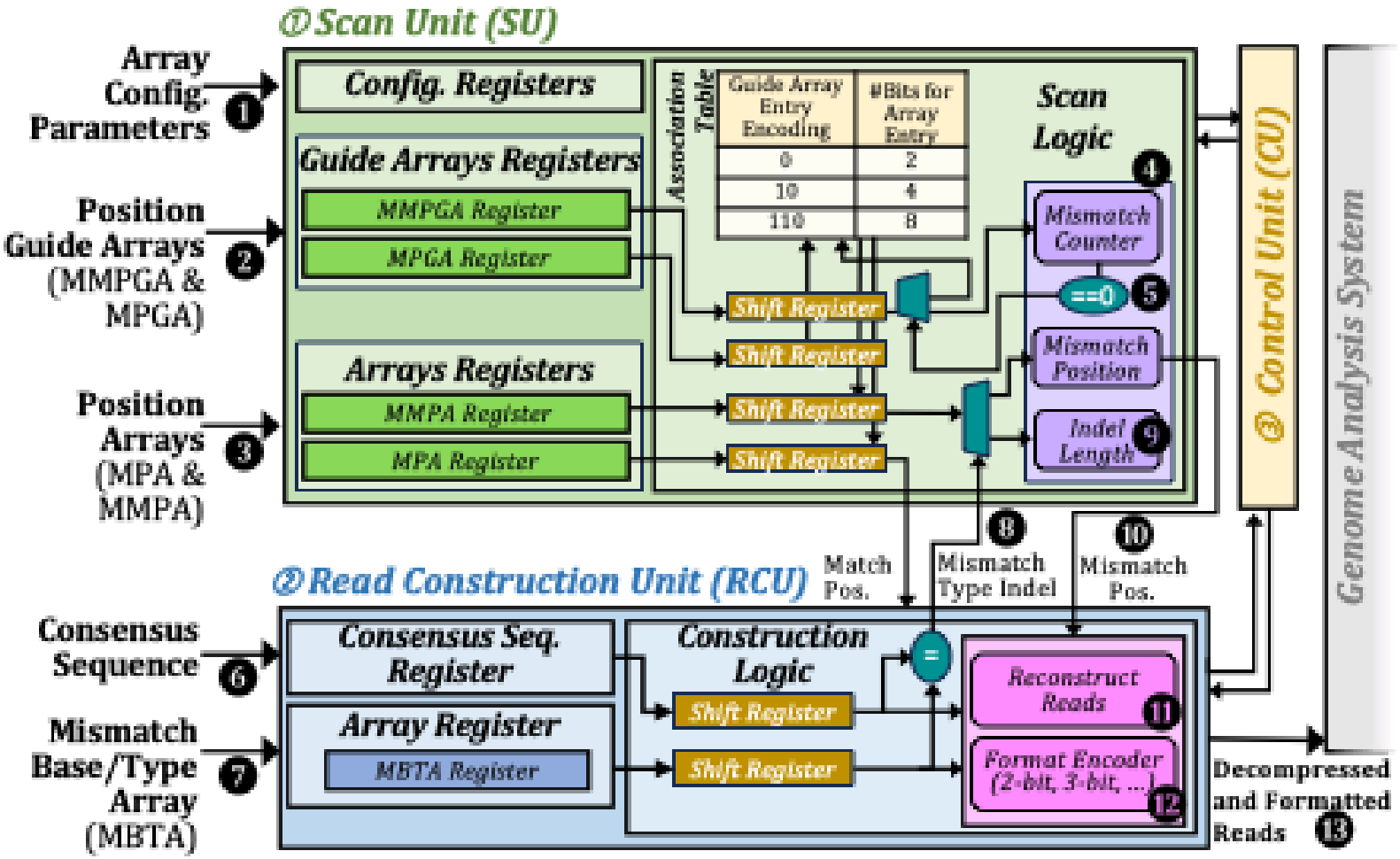
Algorithm 1 Tuning Bit Counts

Input: Histogram of mismatch position bit counts H, where $|H| \leq 32$, and a tuning convergence threshold ε.

Output: List of bit counts W for encoding mismatch positions.

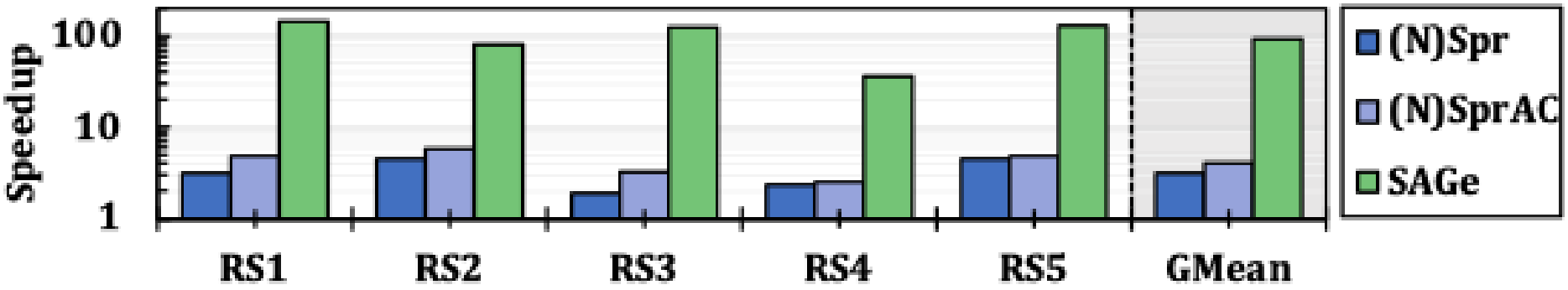
$\ell_{\text{min}} \leftarrow \infty$ ▷ *Initialize min. encoding size*
$x_0 \leftarrow 0$
for all $d \in \{1, \ldots, 8\}$ **do** ▷ *Find optimal $|W|$.*
  $\ell_{\text{last}} \leftarrow \ell_{\text{min}}$ ▷ *Best result from previous d values*
  for all $(x_1, \ldots, x_d) \in |H|^d$ s.t. $x_0 < x_1 < \cdots < x_d$ **do**
    $\ell \leftarrow$ total length of encoded mismatch positions and guide arrays s.t. positions with bit counts $\in (x_i, x_{i+1}]$ are encoded with x_{i+1} bits.
    if $\ell < \ell_{\text{min}}$ **then**
      $\ell_{\text{min}} = \ell$
      $W \leftarrow (x_1, \ldots, x_d)$
  if $(\ell_{\text{last}} - \ell_{\text{min}})/\ell_{\text{min}} < \varepsilon$ **then**
    break ▷ *Exit loop when ℓ_{min} converges, typically at $d < 8$*
return W

Hardware for Data Preparation



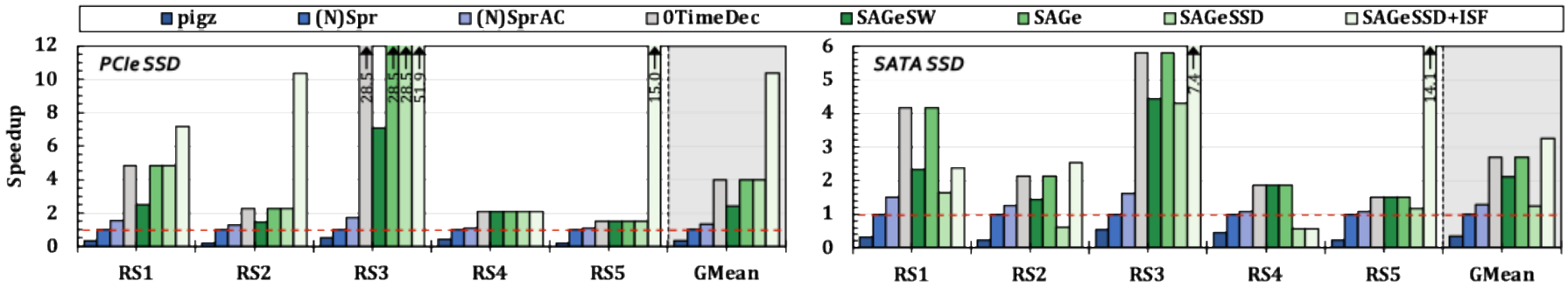
Data Preparation Performance

Only data preparation throughput, normalized to pigz

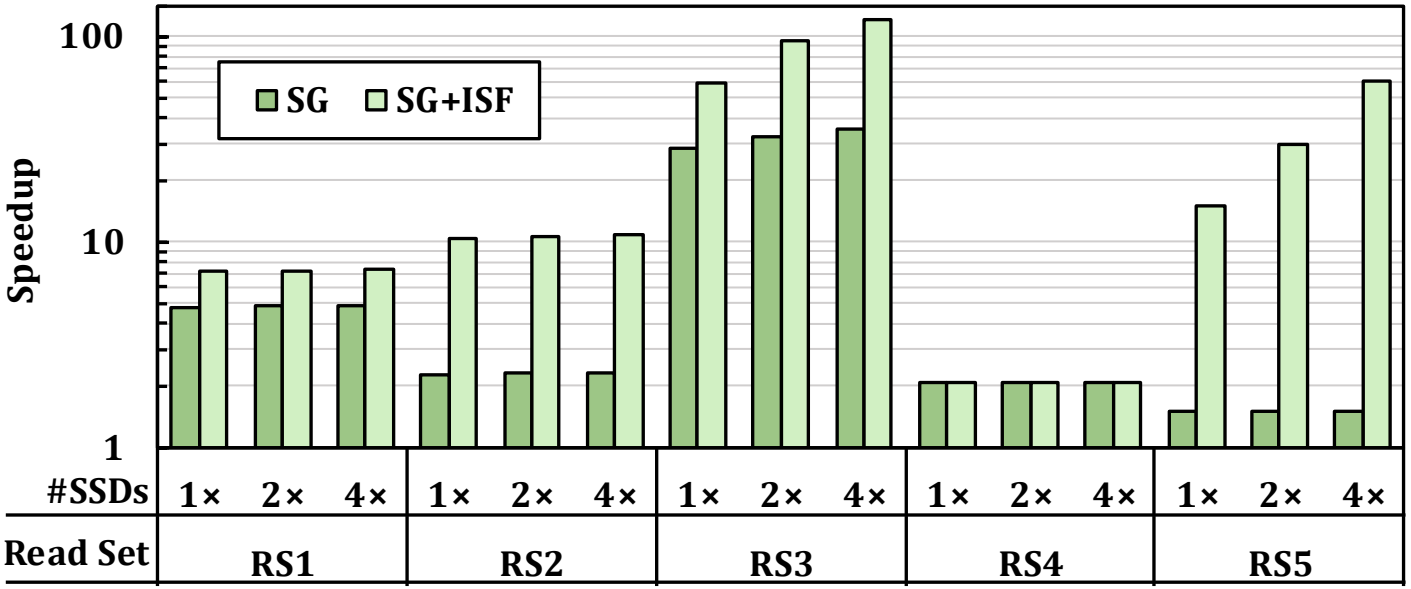


End-to-End Performance

Speedup, with different SSDs



Performance with Multiple SSDs



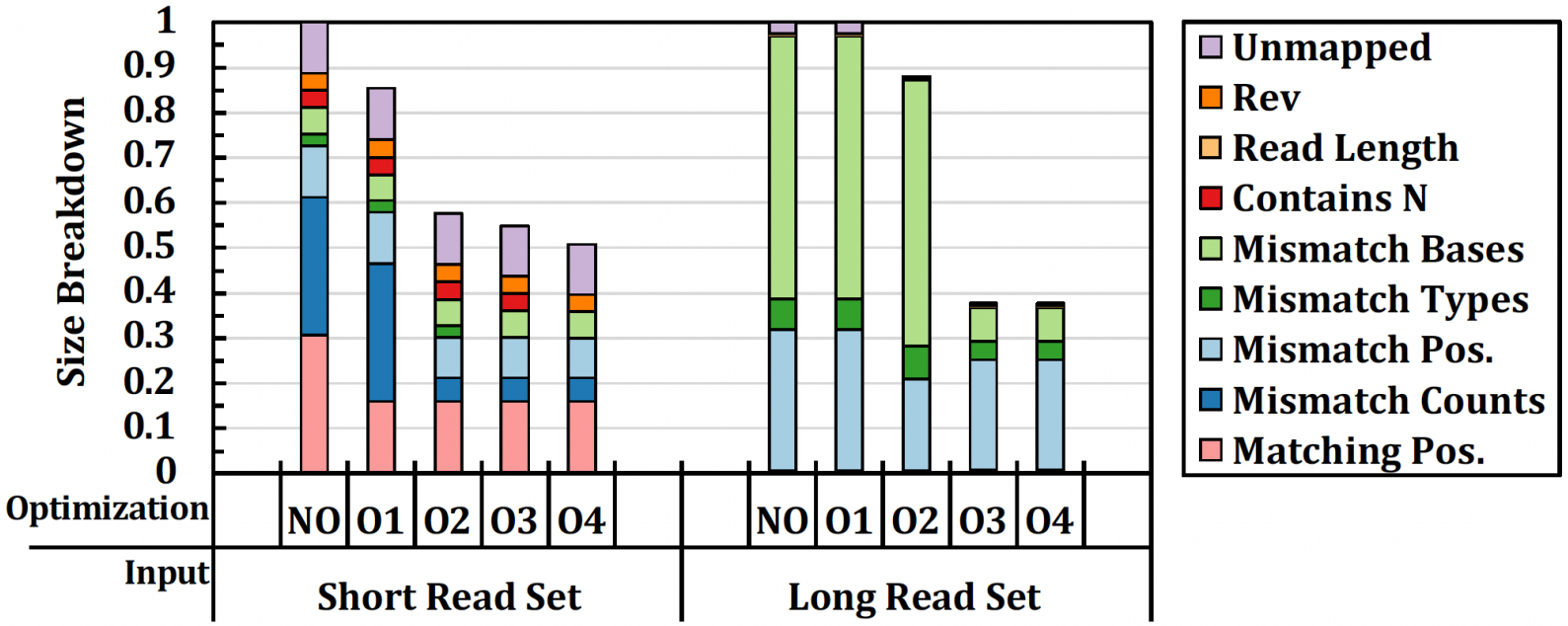
Compression Ratio

Table 2: Compression ratios for different read sets.

Label	Read Set (Short, Long)	Uncomp. Size (MB) DNA+Qual	Compression Ratio					
			PigZ [167]		(Nano)Spring [48]		SAGe	
			DNA	Qual.	DNA	Qual.	DNA	Qual.
RS1	SRR870667_2 [342] (S)	10 000	3.39	2.23	24.8	2.80	22.8	2.80
RS2	ERR194146_1 [343] (S)	158 000	12.5	2.49	40.2	3.4	36.8	3.4
RS3	SRR2052419_1 [344] (S)	8 000	3.41	3.45	7.2	5.07	7.1	5.07
RS4	PAO89685_sampled [345] (L)	24 000	3.93	1.79	4.8	2.19	4.5	2.19
RS5	ERR5455028 [346] (L)	176 800	3.5	1.57	7.6	1.82	7.8	1.82

Effect of Different SAGe Optimizations

NO: no optimization on the raw mismatch information
O1: matching position optimization added to NO
O2: mismatch positions and count optimizations added to O1
O3: mismatching base and type optimizations added to O2
O4: corner case optimizations added to O3



Resource Requirements

Tool	Genomics-specific	Avg. Comp. Ratio	End-to-end?	Hardware Requirements	Memory Footprint	Decomp. Throughput (GB/s)
nvCOMP (DEFLATE) [347]	×	5.3	✓	GPU (NVIDIA A100 [348])	1.5 GB	50
Xilinx/AMD GZIP Engine [349]	×	5.3	✓	FPGA (AMD Alveo U50 [350])	80 KB	0.7
xz [274]	×	6.7	✓	CPU (AMD® EPYC® 7742 CPU [324] with 128 physical cores)	13 GB	0.6
HW-accelerated ZStandard [351]	×	6.7	✓	ASIC ($1.89\ mm^2$ in a 14nm technology node)	2–64 KB	3.9
GPUFastqLZ [53]	✓	5.8	✓	GPU (A GPU server with four NVIDIA Tesla V100 GPUs [352])	N/A[14]	7.8
repaq [54]	✓	17.1	×	FPGA (AMD ALVEO U200 [353])	16 GB	N/A
Spring [43] & NanoSpring [48]	✓	16.9	✓	CPU (AMD® EPYC® 7742 CPU [324] with 128 physical cores)	26 GB	0.7
SAGe	✓	15.8	✓	ASIC ($0.002\ mm^2$ in a 22nm technology node)	128 B	75.4

Compression Time