P&S Genomics

Lecture 8a: GenASM

Joël Lindegger

ETH Zürich

Spring 2023

27 April 2023

Genome Sequencing

- **Genome sequencing:** Enables us to determine the order of the DNA sequence in an organism's genome
  - Plays a pivotal role in:
    - Personalized medicine
    - Outbreak tracing
    - Understanding of evolution

Genome DNA

- Modern genome sequencing machines extract smaller randomized fragments of the original DNA sequence, known as **reads**
  - *Short reads:* a few hundred base pairs, error rate of ~0.1%
  - *Long reads:* thousands to millions of base pairs, error rate of 10–15%

Genome Sequence Analysis

- **Read mapping:** *First key step* in genome sequence analysis (GSA)
  - Aligns reads to one or more possible locations within the reference genome, and
  - Finds the matches and differences between the read and the reference genome segment at that location

- Multiple steps of read mapping require ***approximate string matching***
  - Approximate string matching (ASM) enables read mapping to account for sequencing errors and genetic variations in the reads

- Bottlenecked by the computational power and memory bandwidth limitations of existing systems

GenASM: ASM Framework for GSA

Our Goal:

Accelerate approximate string matching
by designing a fast and flexible framework,
which can accelerate *multiple steps* of genome sequence analysis

- **GenASM:** *First* ASM acceleration framework for GSA
  - Based upon the *Bitap* algorithm
    - Uses fast and simple bitwise operations to perform ASM
  - Modified and extended ASM algorithm
    - Highly-parallel Bitap with long read support
    - Novel bitvector-based algorithm to perform *traceback*
  - Co-design of our modified scalable and memory-efficient algorithms with low-power and area-efficient hardware accelerators

Use Cases & Key Results

(1) Read Alignment

- **116×** speedup, **37×** less power than **Minimap2** (state-of-the-art **SW**)
- **111×** speedup, **33×** less power than **BWA-MEM** (state-of-the-art **SW**)
- **3.9×** better throughput, **2.7×** less power than **Darwin** (state-of-the-art **HW**)
- **1.9×** better throughput, **82%** less logic power than **GenAx** (state-of-the-art **HW**)

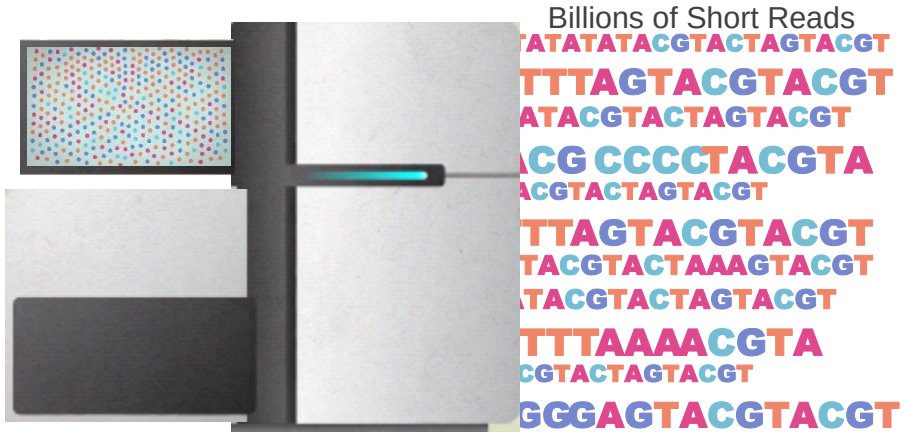
(2) Pre-Alignment Filtering

- **3.7×** speedup, **1.7×** less power than **Shouji** (state-of-the-art **HW**)

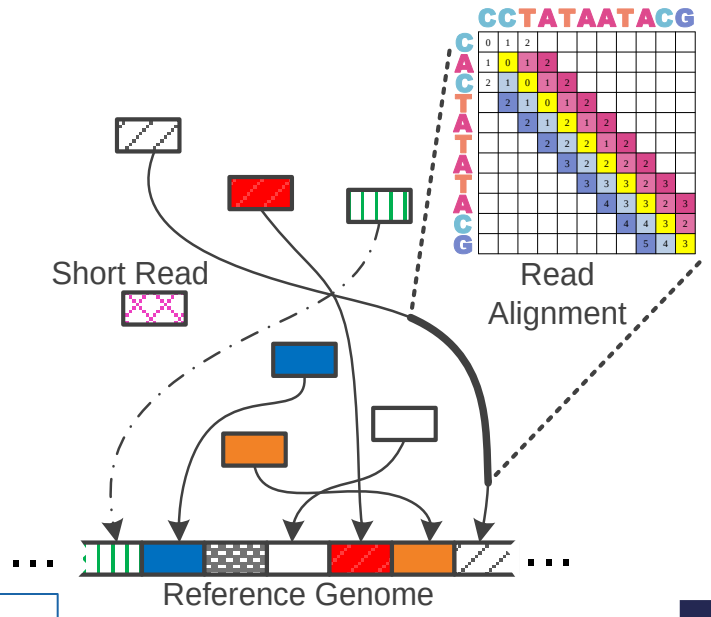
(3) Edit Distance Calculation

- **22–12501×** speedup, **548–582×** less power than **Edlib** (state-of-the-art **SW**)
- **9.3–400×** speedup, **67×** less power than **ASAP** (state-of-the-art **HW**)

Source: Prof. Onur Mutlu's lecture slides



1 Sequencing



2 Read Mapping

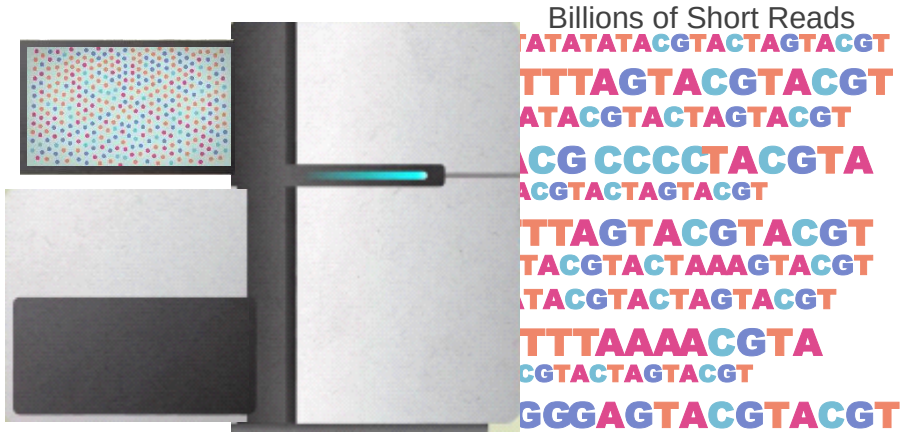
Genome Analysis

```
reference: TTTATCGCTTCCATGACGCAG
read1:       ATCGCATCC
read2:      TATCGCATC
read3:           CATCCATGA
read4:         CGCTTCCAT
read5:               CCATGACGC
read6:             TTCCATGAC
```

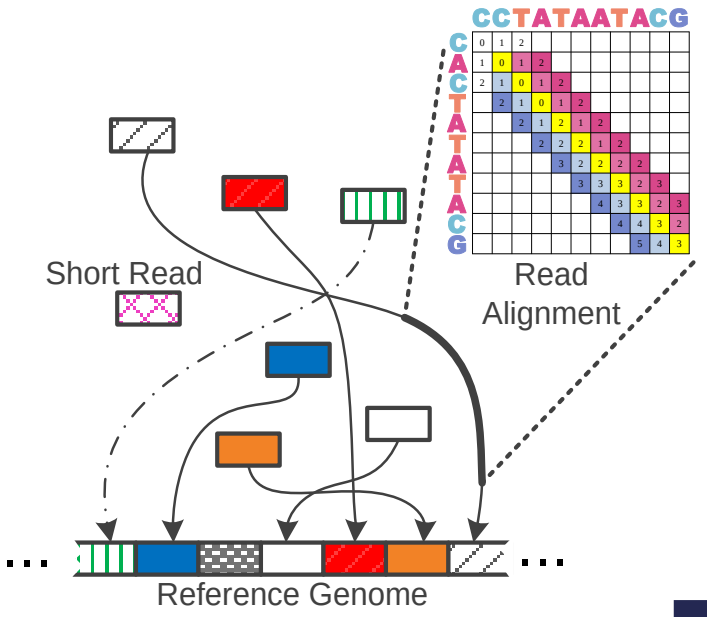
3 Variant Calling

4 Scientific Discovery

Source: Prof. Onur Mutlu's lecture slides

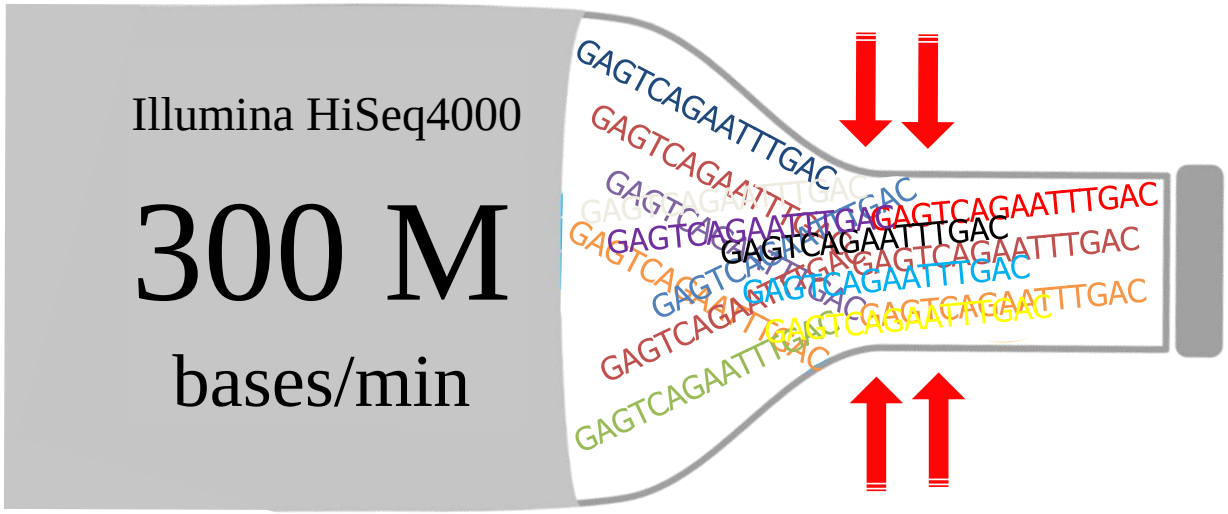


1 Sequencing



Read Mapping 2

Bottlenecked in Mapping!!



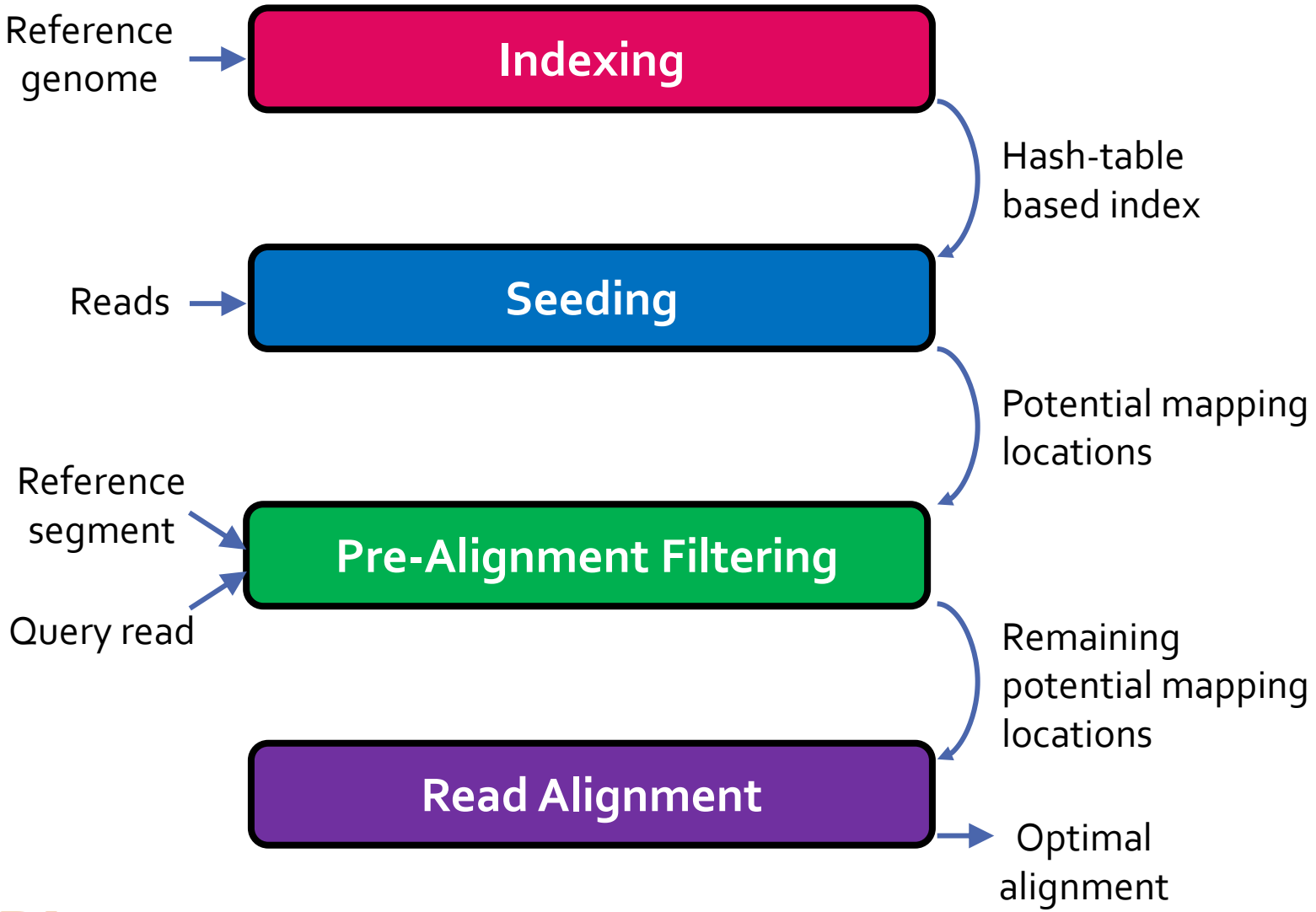
on average

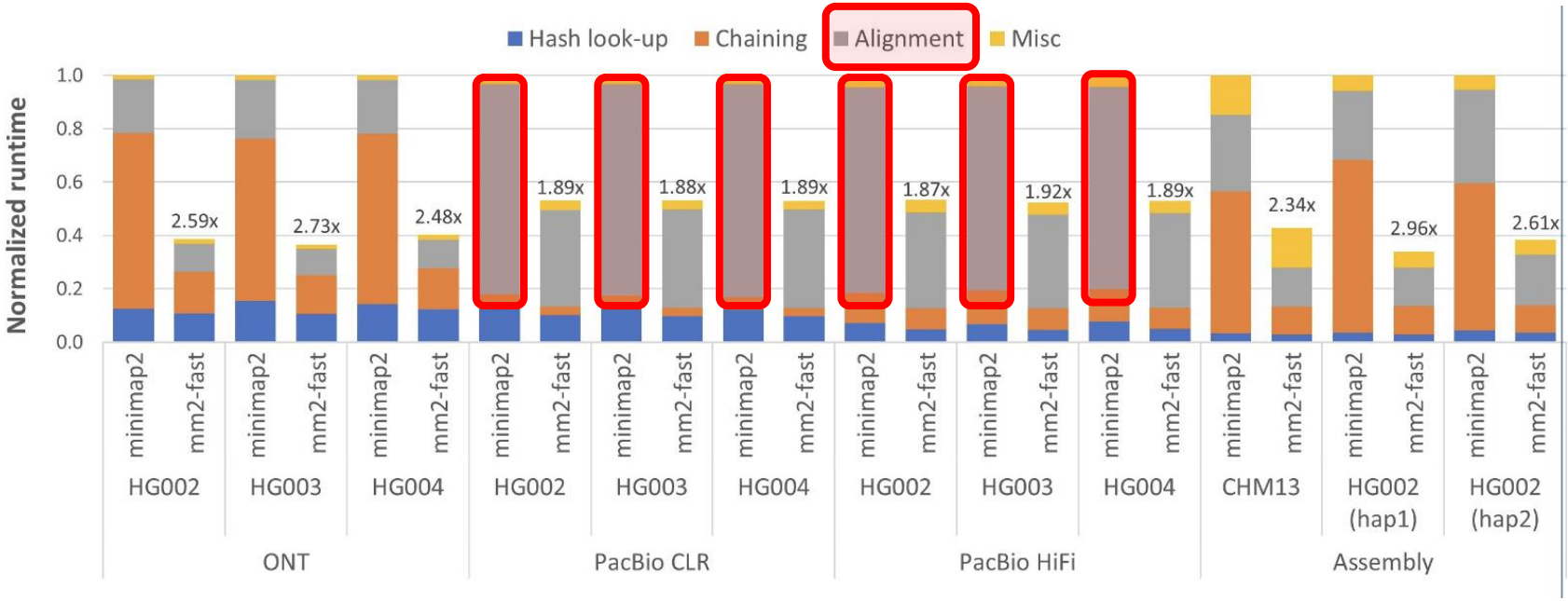
2 M

bases/min

(0.6%)

Read Mapping





Read alignment is often the bottleneck in read mapping

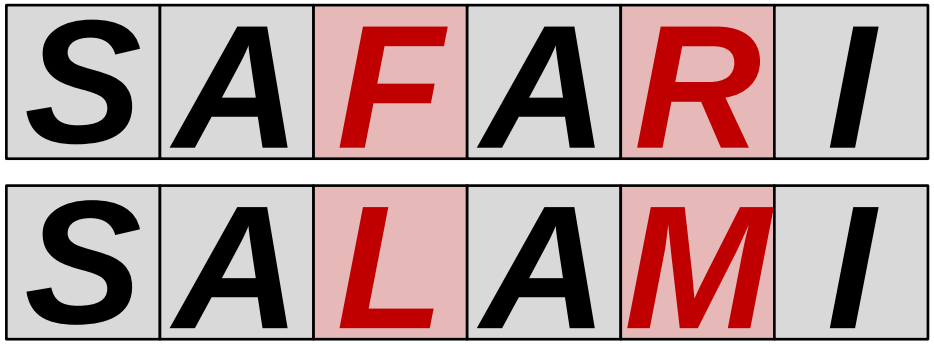
Pairwise Sequence Alignment (PSA)

- Compare a pair of strings
- while allowing substitutions, insertions, and deletions

S	A	F	A	R	I
S	A	L	A	M	I

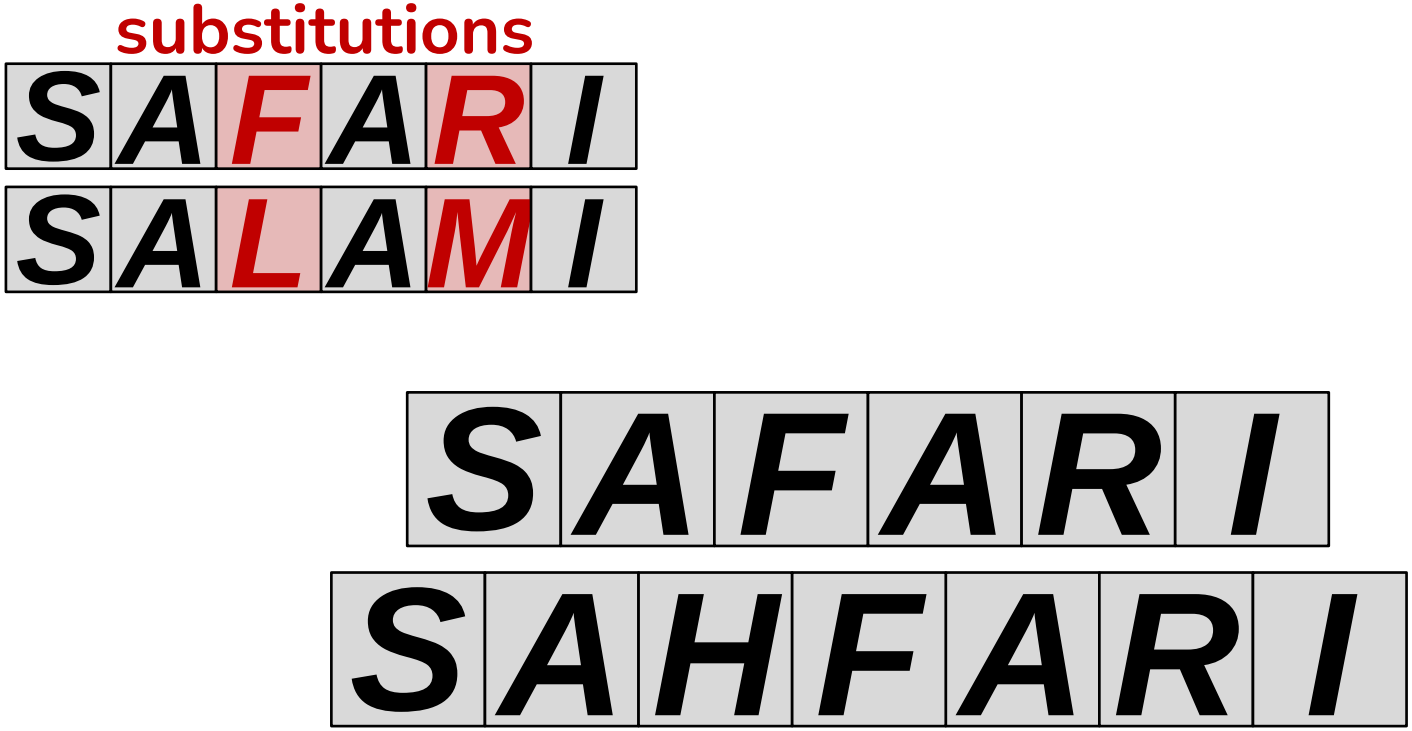
Pairwise Sequence Alignment (PSA)

- Compare a pair of strings
- while allowing **substitutions**, insertions, and deletions



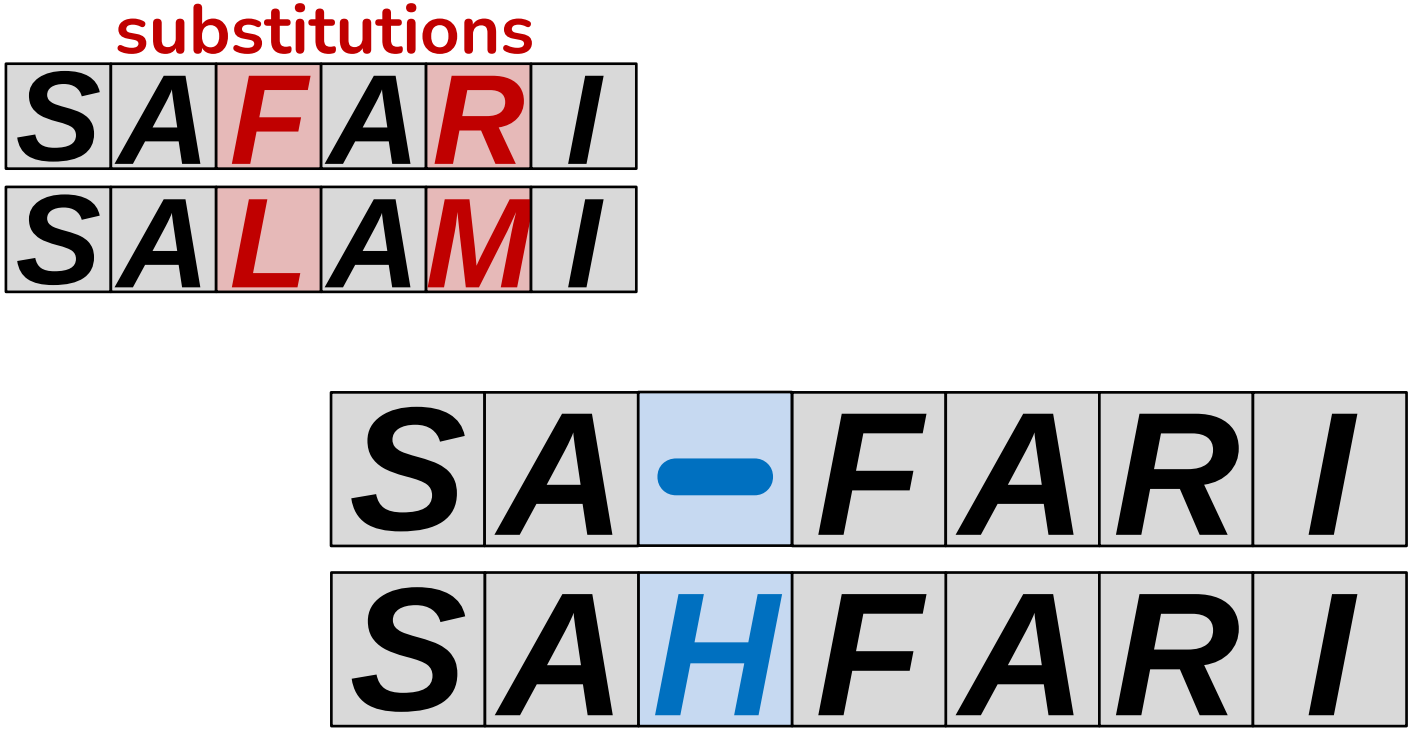
Pairwise Sequence Alignment (PSA)

- Compare a pair of strings
- while allowing substitutions, insertions, and deletions



Pairwise Sequence Alignment (PSA)

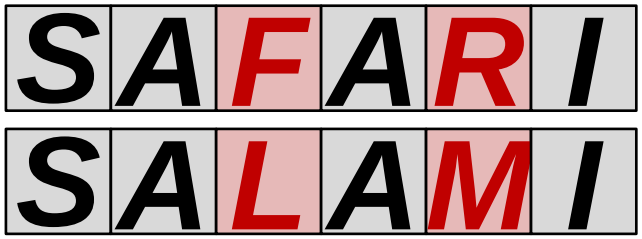
- Compare a pair of strings
- while allowing substitutions, insertions, and deletions



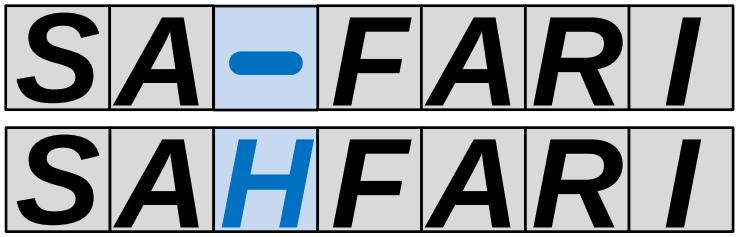
Pairwise Sequence Alignment (PSA)

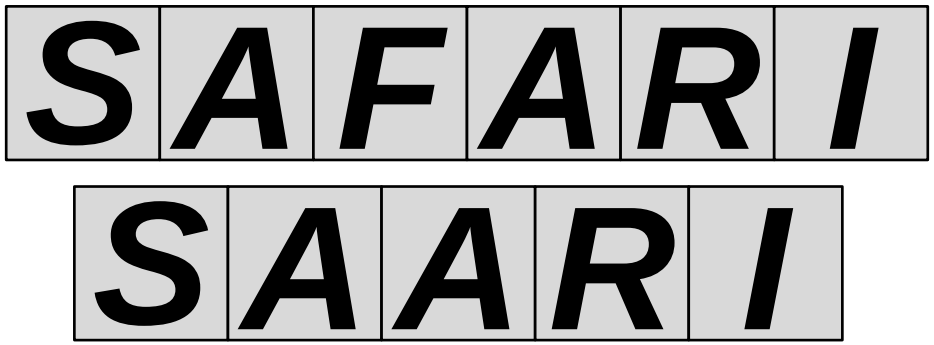
- Compare a pair of strings
- while allowing substitutions, insertions, and deletions

substitutions



insertions





Pairwise Sequence Alignment (PSA)

- Compare a pair of strings
- while allowing substitutions, insertions, and deletions

substitutions

S	A	F	A	R	I
S	A	L	A	M	I

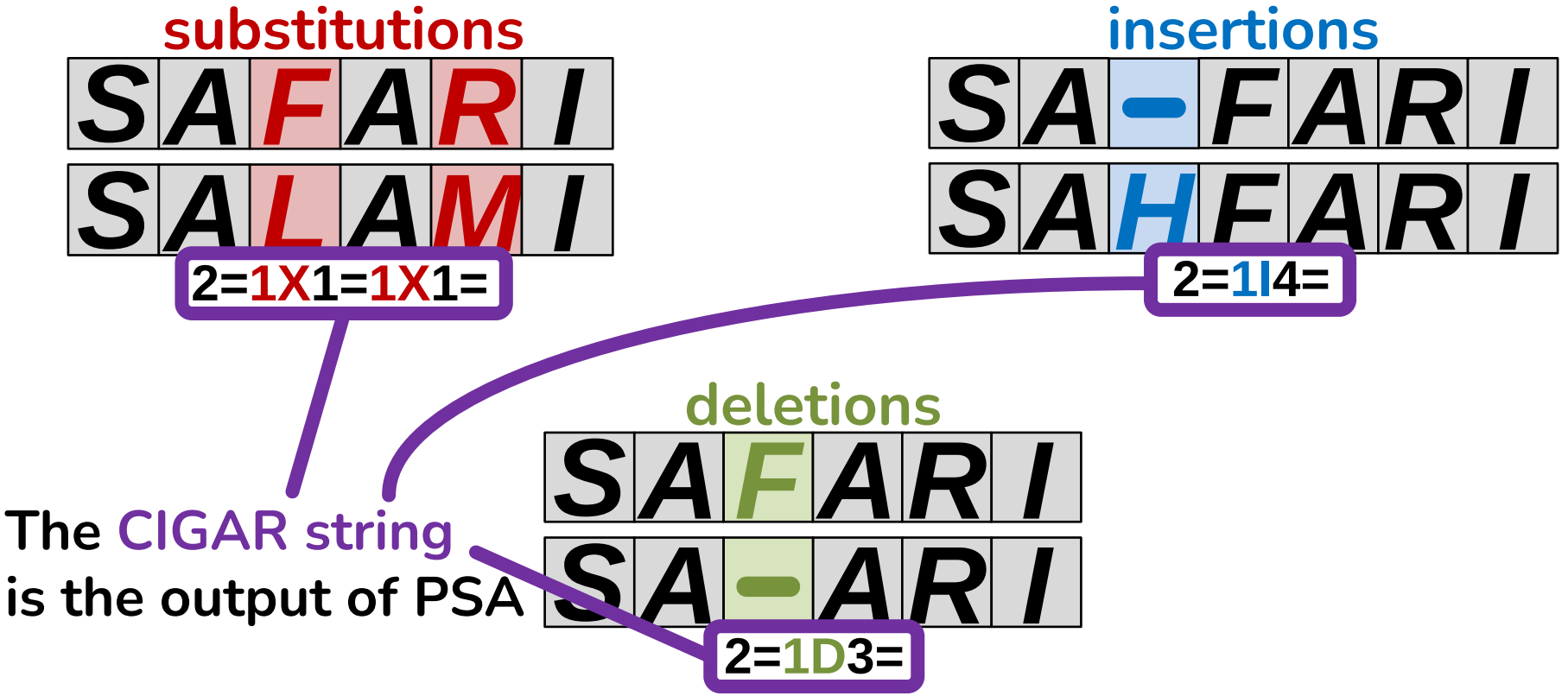
insertions

S	A	–	F	A	R	I
S	A	H	F	A	R	I

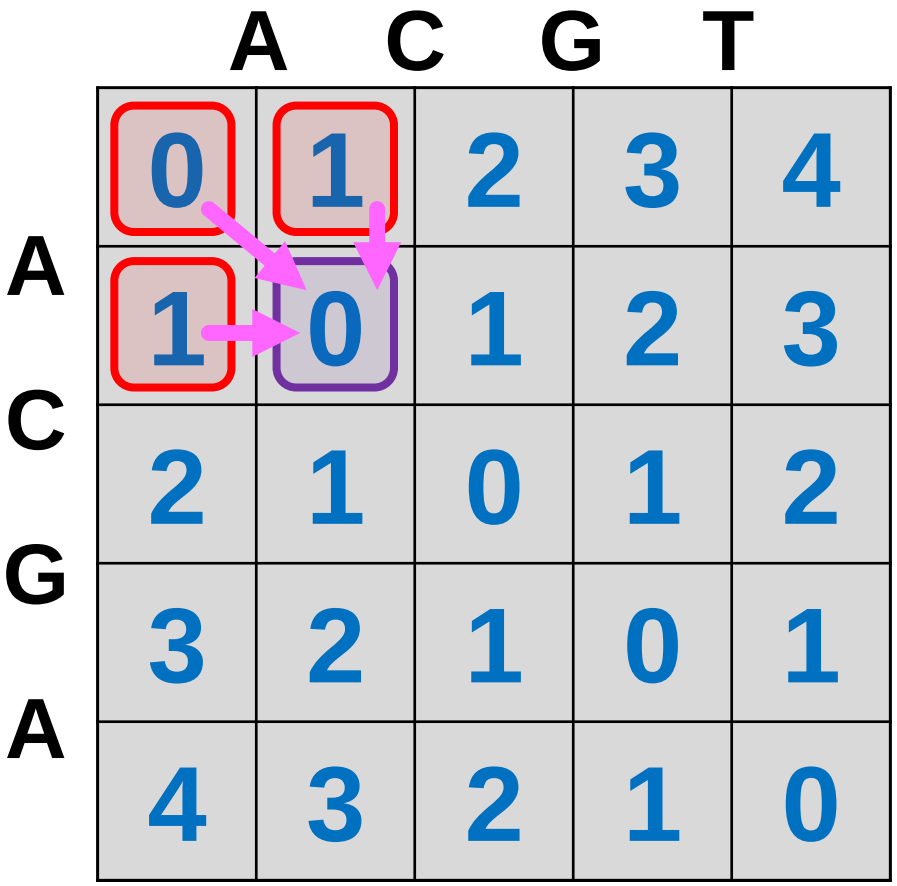
S	A	F	A	R	I
S	A	–	A	R	I

Pairwise Sequence Alignment (PSA)

- Compare a pair of strings
- while allowing substitutions, insertions, and deletions
- The total number of edits should be minimal



Arithmetic Dynamic Programming for PSA



Needleman-Wunsch Smith-Waterman-Gotoh, WFA, ...
Next entry is calculated from three neighbors
using arithmetic operations

The Bitap Algorithm

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	1010	1100	1110	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000

Bitap

Next entry is calculated from three neighbors using bitwise operations

Particularly efficient in hardware

		A	C	G	T
		0	1	2	3	4
A		1	0	1	2	3
C		2	1	0	1	2
G		3	2	1	0	1
A		4	3	2	1	0

Needleman-Wunsch Smith-Waterman-Gotoh, WFA, ...

Next entry is calculated from three neighbors using arithmetic operations

Bitap Algorithm (cont'd.)

For each character of the text (char):

Large number of iterations

Copy previous R bitvectors as oldR

R[0] = (oldR[0] << 1) | PM [char]

For d = 1…k:

deletion = oldR[d-1]

substitution = oldR[d-1] << 1

insertion = R[d-1] << 1

match = (oldR[d] << 1) | PM [char]

R[d] = deletion & mismatch & insertion & match

Check MSB of R[d]:

If 1, no match.

If 0, match with *d* many errors.

Bitap Algorithm (cont'd.)

For each character of the text (char):

Copy previous R bitvectors as oldR

R[0] = (oldR[0] << 1) | PM [char]

For d = 1…k:

deletion = oldR[d-1]

substitution = oldR[d-1] << 1

insertion = R[d-1] << 1

match = (oldR[d] << 1) | PM [char]

Data dependency between iterations (i.e., no parallelization)

R[d] = deletion & mismatch & insertion & match

Check MSB of R[d]:

If 1, no match.

If 0, match with *d* many errors.

Bitap Algorithm (cont'd.)

For each character of the text (char):

Copy previous R bitvectors as oldR

R[0] = (oldR[0] << 1) | PM [char]

For d = 1…k:

deletion = oldR[d-1]

substitution = oldR[d-1] << 1

insertion = R[d-1] << 1

match = (oldR[d] << 1) | PM [char]

R[d] = deletion & mismatch & insertion & match

Check MSB of R[d]:

If 1, no match.

If 0, match with *d* many errors.

Does *not* store and process these intermediate bitvectors to find the optimal alignment (i.e., no traceback)

Limitations of Bitap

1) Data Dependency Between Iterations: **Algorithm**

- Two-level data dependency forces the consecutive iterations to take place sequentially

2) No Support for Traceback:

- Bitap does not include any support for optimal alignment identification

3) No Support for Long Reads:

- Each bitvector has a length equal to the length of the pattern
- Bitwise operations are performed on these bitvectors

4) Limited Compute Parallelism: **Hardware**

- Text-level parallelism
- Limited by the number of compute units in existing systems

5) Limited Memory Bandwidth:

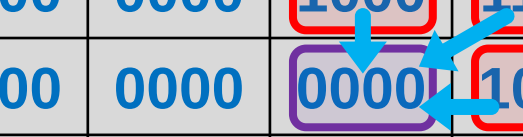
- High memory bandwidth required to read and write the computed bitvectors to memory

GenASM: ASM Framework for GSA

- Approximate string matching (ASM) acceleration framework based on the Bitap algorithm
- *First* ASM acceleration framework for genome sequence analysis
- We overcome the five limitations that hinder Bitap's use in genome sequence analysis:
  - Modified and extended ASM algorithm
    - Highly-parallel Bitap with long read support
    - Novel bitvector-based algorithm to perform *traceback*
  - Specialized, low-power and area-efficient hardware for both modified Bitap and novel traceback algorithms

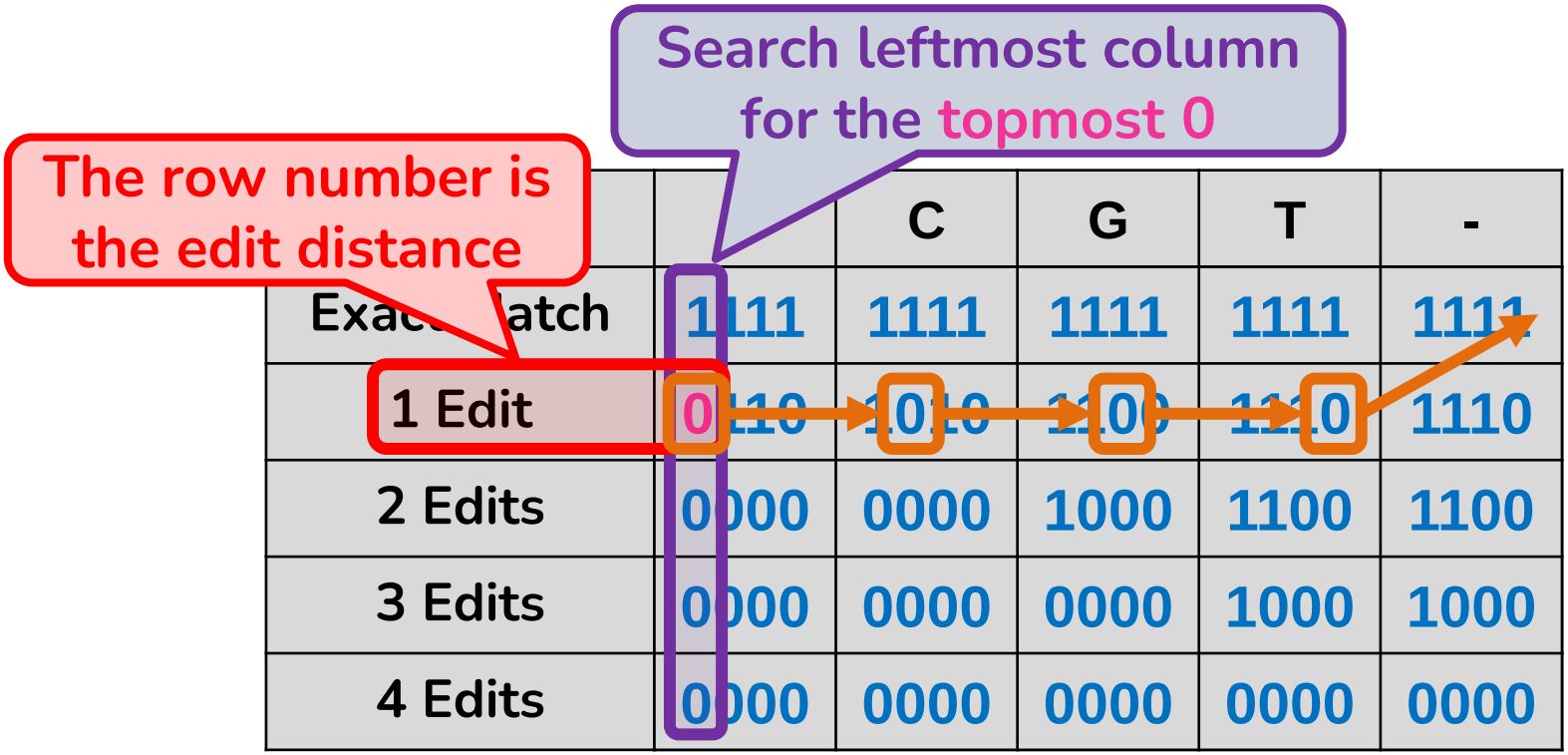
The GenASM-DC Algorithm

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	1010	1100	1110	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000



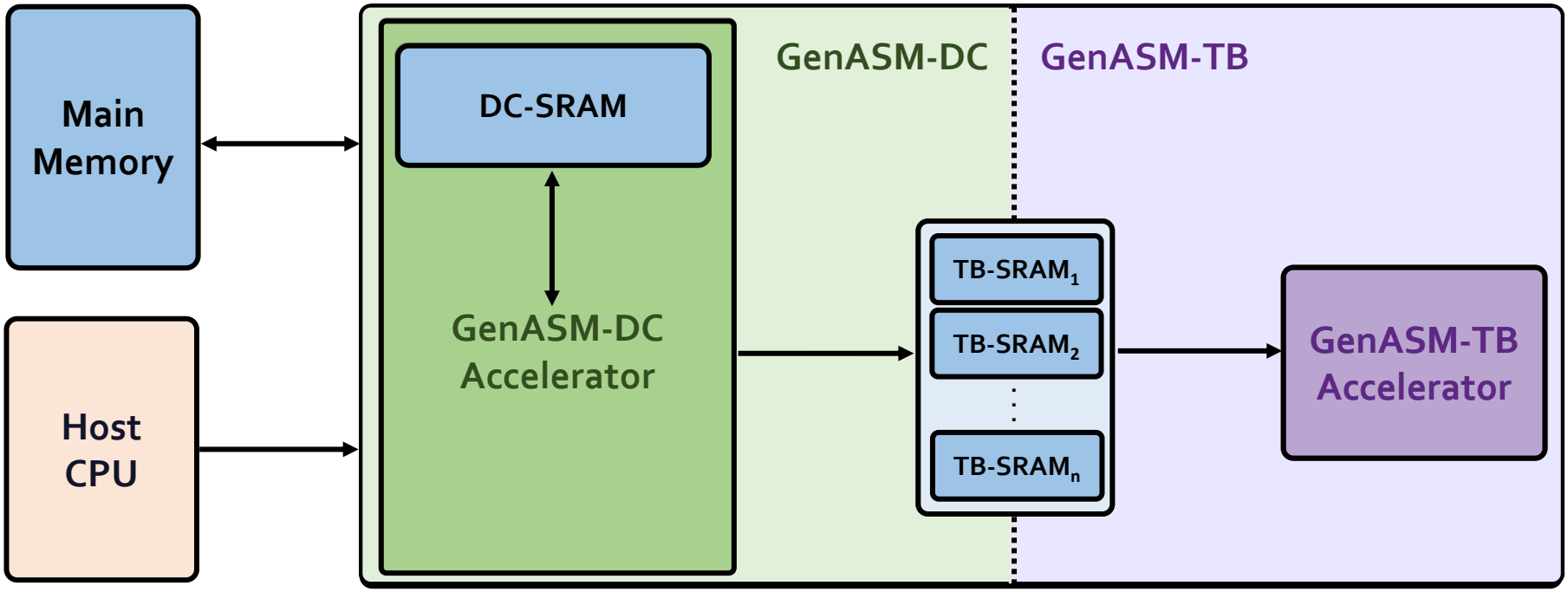
GenASM-DC stores all computed bitvectors for later "traceback"

The GenASM-TB Algorithm



Traceback obtains the CIGAR string by backtracking the origin of the topmost 0 in the leftmost column.

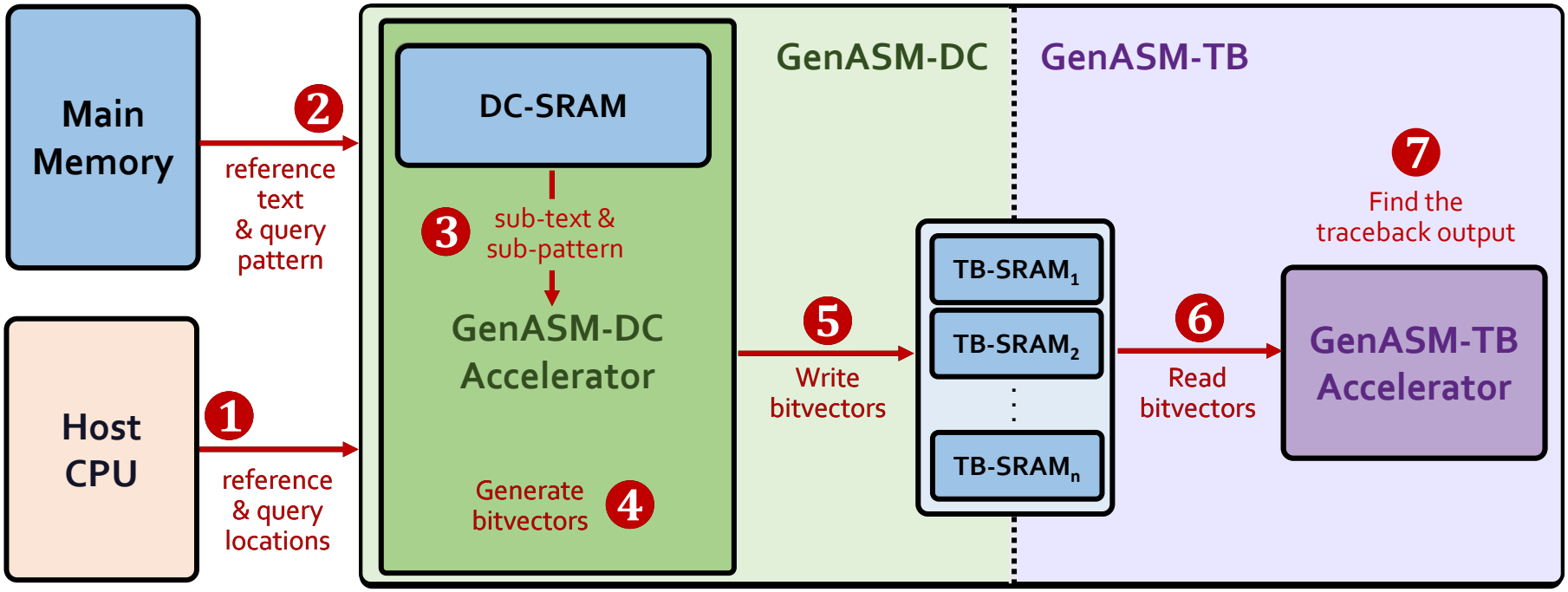
GenASM Hardware Design



GenASM-DC:
generates bitvectors and performs edit **D**istance **C**alculation

GenASM-TB:
performs **T**race**B**ack and assembles the optimal alignment

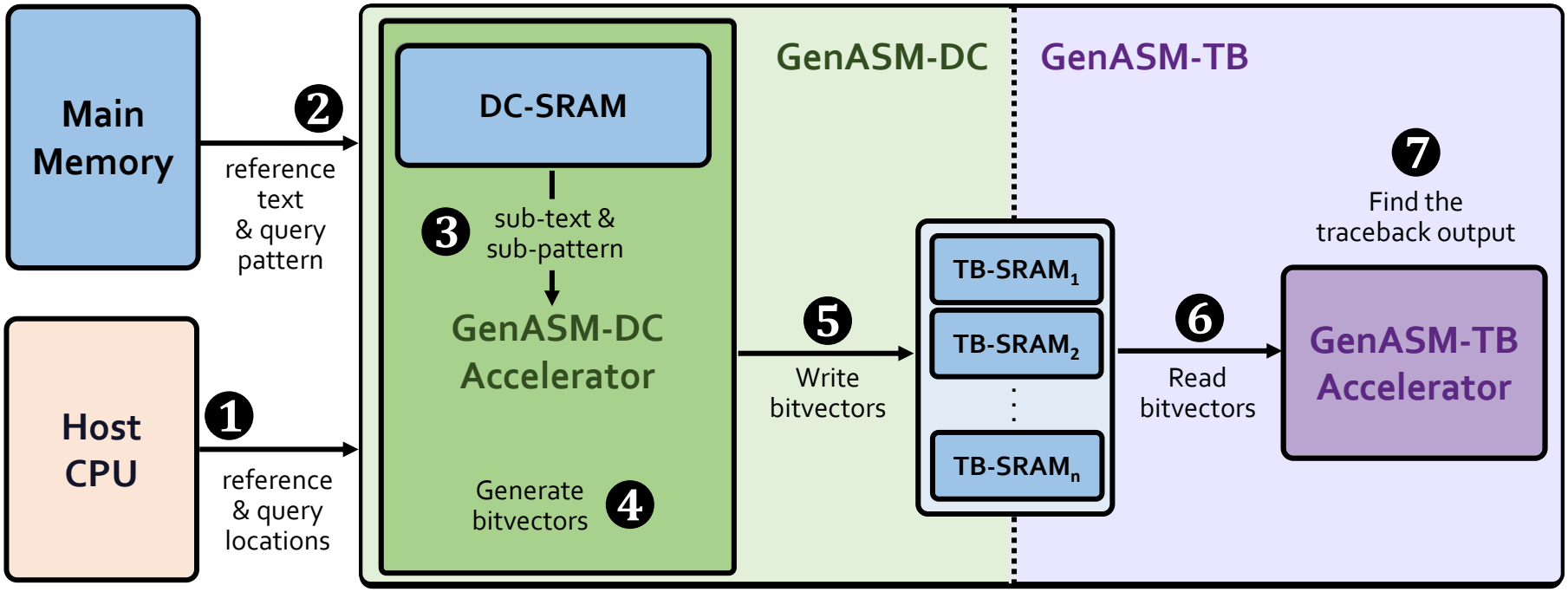
GenASM Hardware Design



GenASM-DC:
generates bitvectors and performs edit **D**istance **C**alculation

GenASM-TB:
performs **T**race**B**ack and assembles the optimal alignment

GenASM Hardware Design



Our *specialized compute units* and *on-chip SRAMs* help us to:

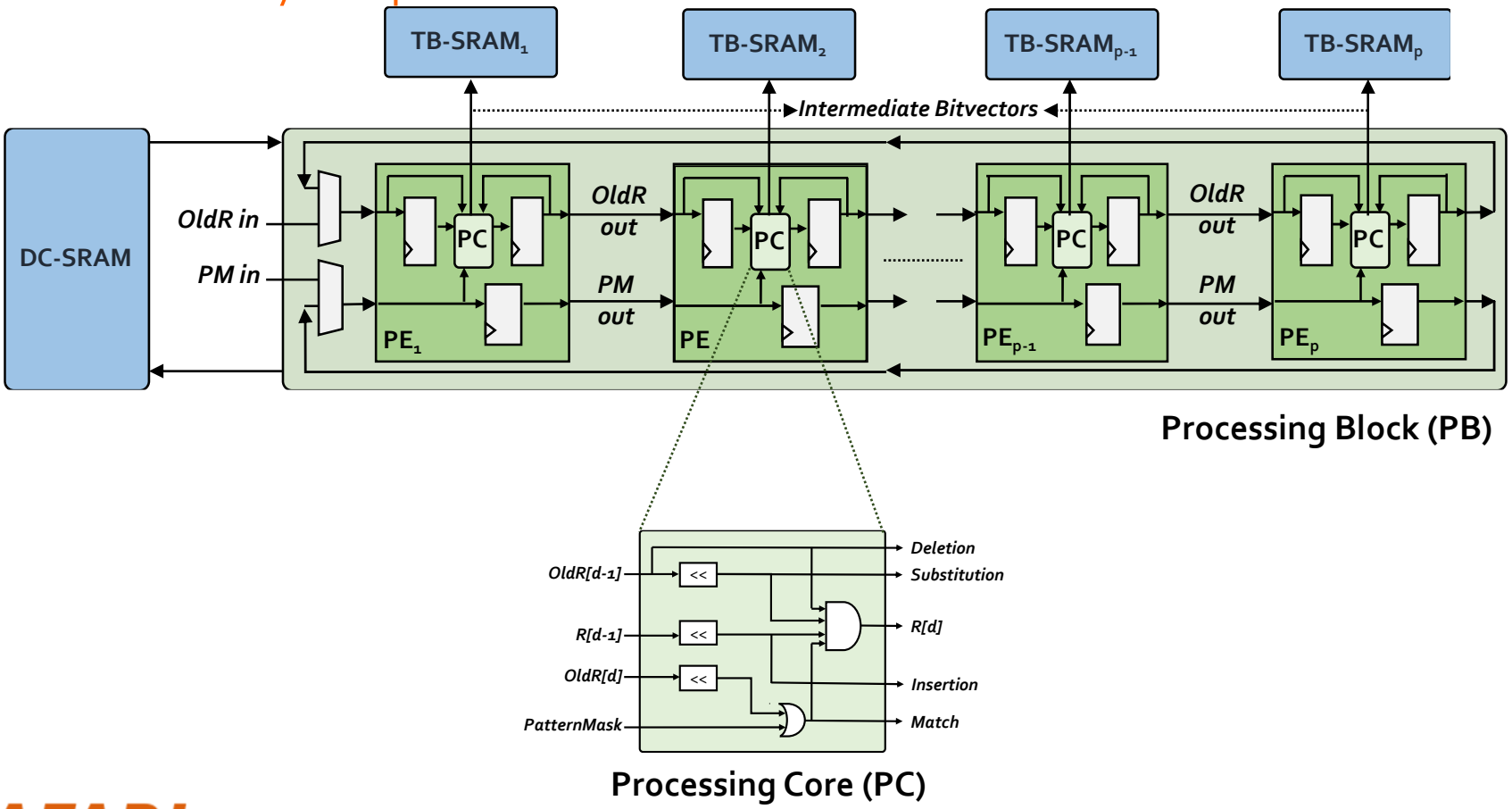
→ Match the rate of computation with memory capacity and bandwidth

→ Achieve high performance and power efficiency

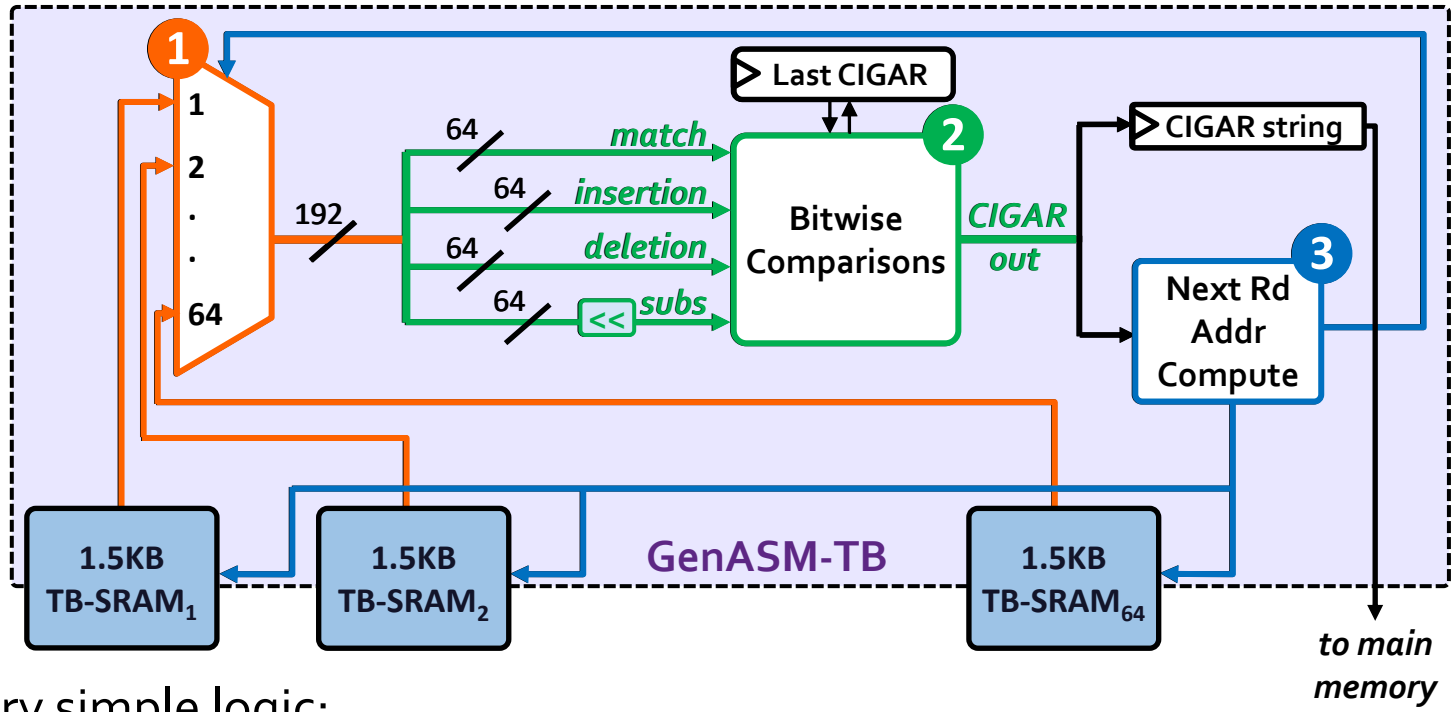
→ Scale linearly in performance with the number of parallel compute units that we add to the system

GenASM-DC: Hardware Design

- **Linear cyclic systolic array** based accelerator
  - Designed to maximize parallelism and minimize memory bandwidth and memory footprint



GenASM-TB: Hardware Design



- Very simple logic:
  - 1 Reads the bitvectors from one of the TB-SRAMs using the computed address
  - 2 Performs the required bitwise comparisons to find the traceback output for the current position
  - 3 Computes the next TB-SRAM address to read the new set of bitvectors

Evaluation Methodology

- We evaluate GenASM using:
  - Synthesized SystemVerilog models of the GenASM-DC and GenASM-TB accelerator datapaths
  - Detailed simulation-based performance modeling

- 16GB HMC-like 3D-stacked DRAM architecture
  - 32 vaults
  - 256GB/s of internal bandwidth, clock frequency of 1.25GHz
  - In order to achieve high parallelism and low power-consumption
  - Within each vault, the logic layer contains a GenASM-DC accelerator, its associated DC-SRAM, a GenASM-TB accelerator, and TB-SRAMs.

Evaluation Methodology (cont'd.)

	SW Baselines	HW Baselines
Read Alignment	Minimap2[1] BWA-MEM[2]	GACT (Darwin)[3] SillaX (GenAx)[4]
Pre-Alignment Filtering	–	Shouji[5]
Edit Distance Calculation	Edlib[6]	ASAP[7]

[1] H. Li. "Minimap2: Pairwise Alignment for Nucleotide Sequences." In *Bioinformatics,* 2018.
[2] H. Li. "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM." In *arXiv,* 2013.
[3] Y. Turakhia et al. "Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly." In *ASPLOS,* 2018.
[4] D. Fujiki et al. "GenAx: A genome sequencing accelerator." In *ISCA*, 2018.
[5] M. Alser. "Shouji: A fast and efficient pre-alignment filter for sequence alignment." In *Bioinformatics,* 2019.
[6] M. Šošić et al. "Edlib: A C/C++ library for fast, exact sequence alignment using edit distance." In *Bioinformatics,* 2017.
[7] S.S. Banerjee et al. "ASAP: Accelerated short-read alignment on programmable hardware." In *TC*, 2018.

Evaluation Methodology (cont'd.)

- **For Use Case 1: Read Alignment**, we compare GenASM with:
  - Minimap2 and BWA-MEM (state-of-the-art **SW**)
    - Running on Intel® Xeon® Gold 6126 CPU (12-core) operating @2.60GHz with 64GB DDR4 memory
    - Using two simulated datasets:
      - Long ONT and PacBio reads: 10Kbp reads, 10-15% error rate
      - Short Illumina reads: 100-250bp reads, 5% error rate
  - GACT of Darwin and SillaX of GenAx (state-of-the-art **HW**)
    - Open-source RTL for GACT
    - Data reported by the original work for SillaX
    - GACT is best for long reads, SillaX is best for short reads

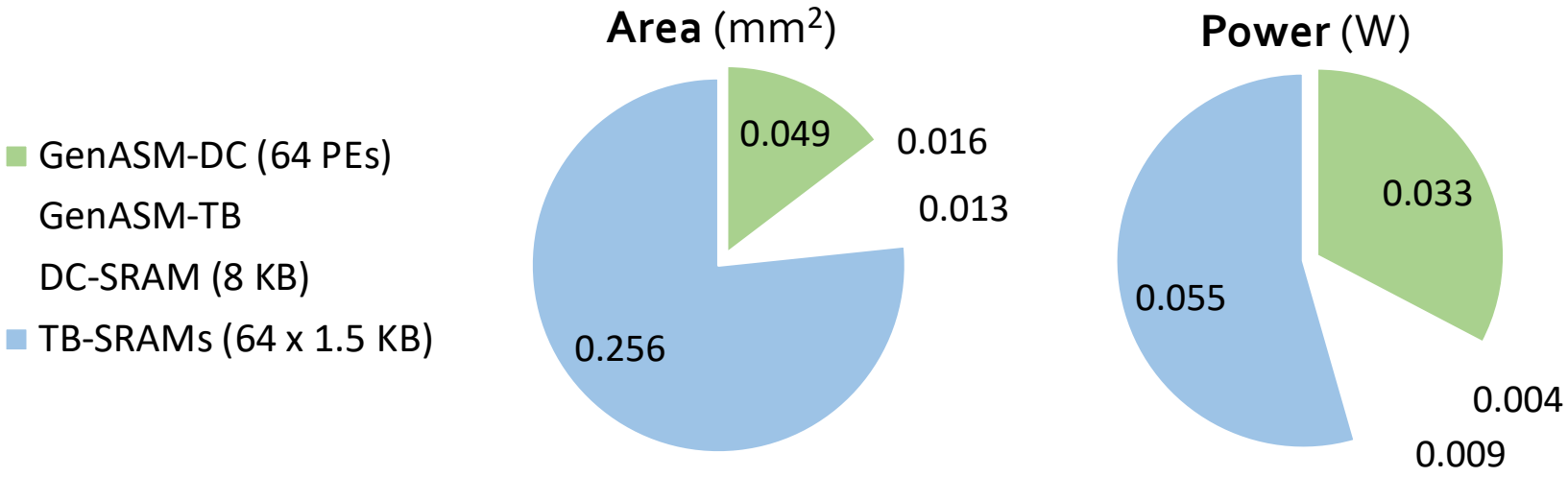
Evaluation Methodology (cont'd.)

- **For Use Case 2: Pre-Alignment Filtering,** we compare GenASM with:
  - Shouji (state-of-the-art **HW** – FPGA-based filter)
    - Using two datasets provided as test cases:
      - 100bp reference-read pairs with an edit distance threshold of 5
      - 250bp reference-read pairs with an edit distance threshold of 15

- **For Use Case 3: Edit Distance Calculation,** we compare GenASM with:
  - Edlib (state-of-the-art **SW**)
    - Using two 100Kbp and 1Mbp sequences with similarity ranging between 60%-99%
  - ASAP (state-of-the-art **HW** – FPGA-based accelerator)
    - Using data reported by the original work

Key Results – Area and Power

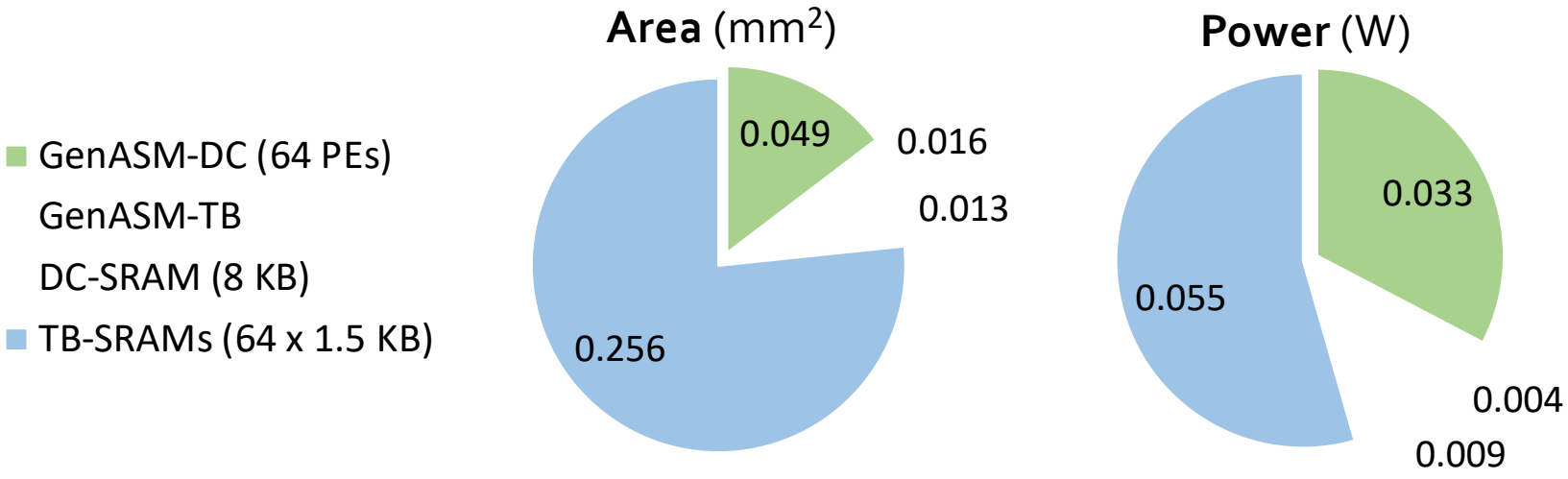
- Based on our **synthesis** of **GenASM-DC** and **GenASM-TB** accelerator datapaths using the Synopsys Design Compiler with a **28nm** LP process:
  - Both GenASM-DC and GenASM-TB operate **@ 1GHz**



Total (1 vault):	0.334 mm^2	0.101 W
Total (32 vaults):	10.69 mm^2	3.23 W
% of a Xeon CPU core:	**1%**	**1%**

Key Results – Area and Power

- Based on our **synthesis** of **GenASM-DC** and **GenASM-TB** accelerator datapaths using the Synopsys Design Compiler with a **28nm** LP process:
  - Both GenASM-DC and GenASM-TB operate **@ 1GHz**



GenASM has low area and power overheads

Key Results – Use Case 1

(1) Read Alignment Step of Read Mapping

- Find the optimal alignment of how reads map to candidate reference regions

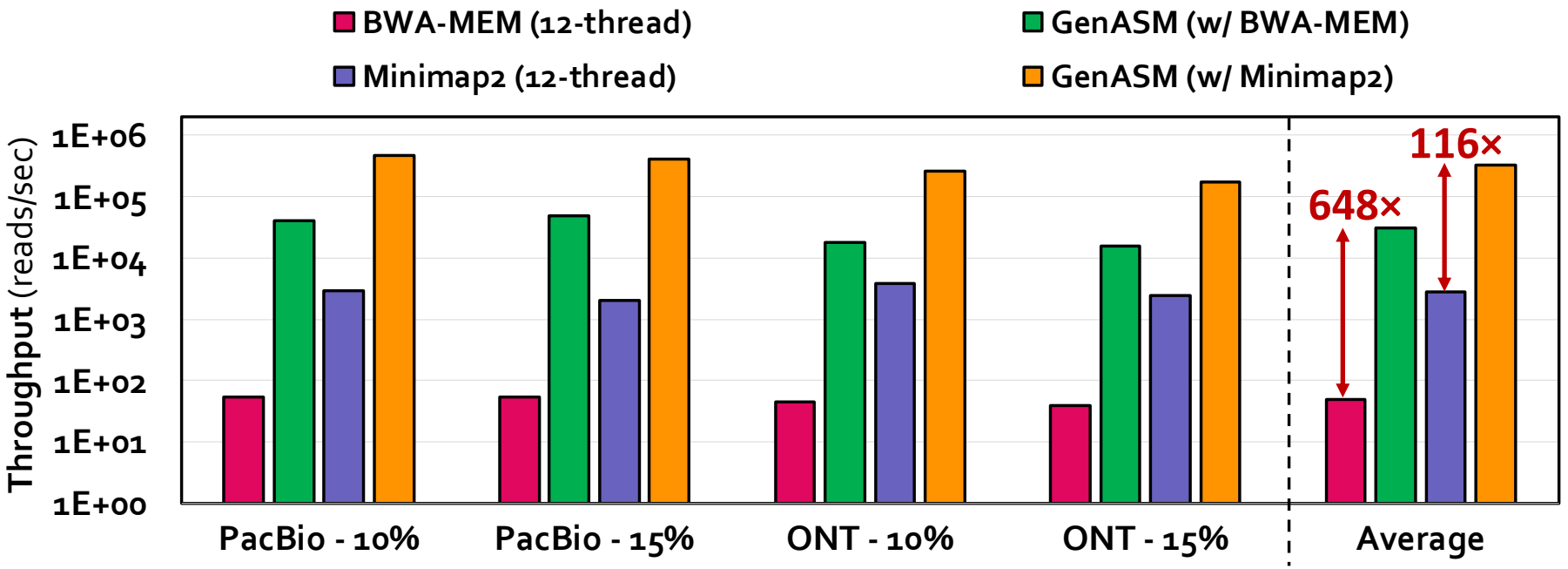
(2) Pre-Alignment Filtering for Short Reads

- Quickly identify and filter out the unlikely candidate reference regions for each read

(3) Edit Distance Calculation

- Measure the similarity or distance between two sequences

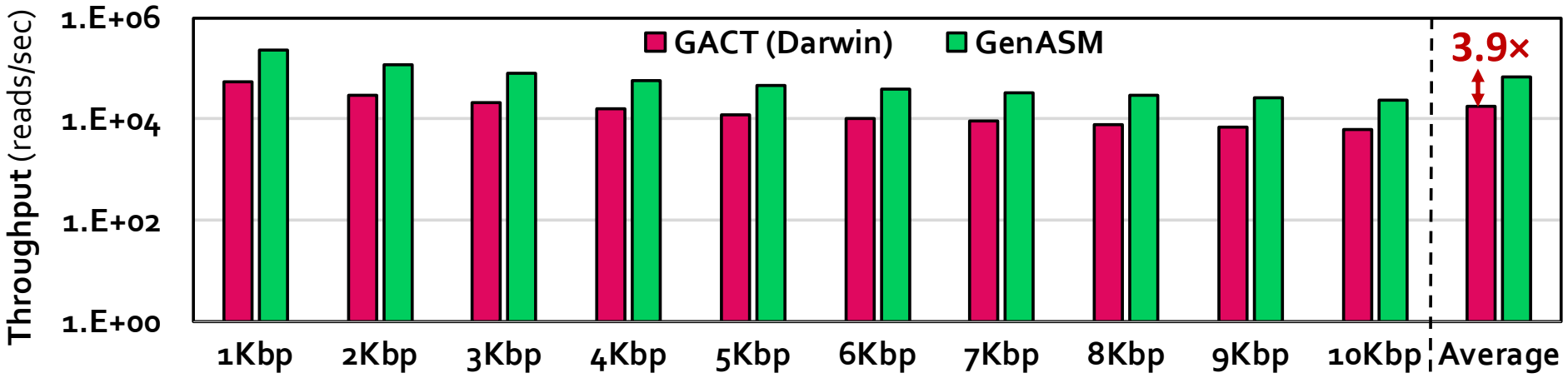
Key Results – Use Case 1 (Long Reads)



SW

GenASM **achieves 648× and 116× speedup** over 12-thread runs of BWA-MEM and Minimap2, while **reducing power consumption by 34× and 37×**

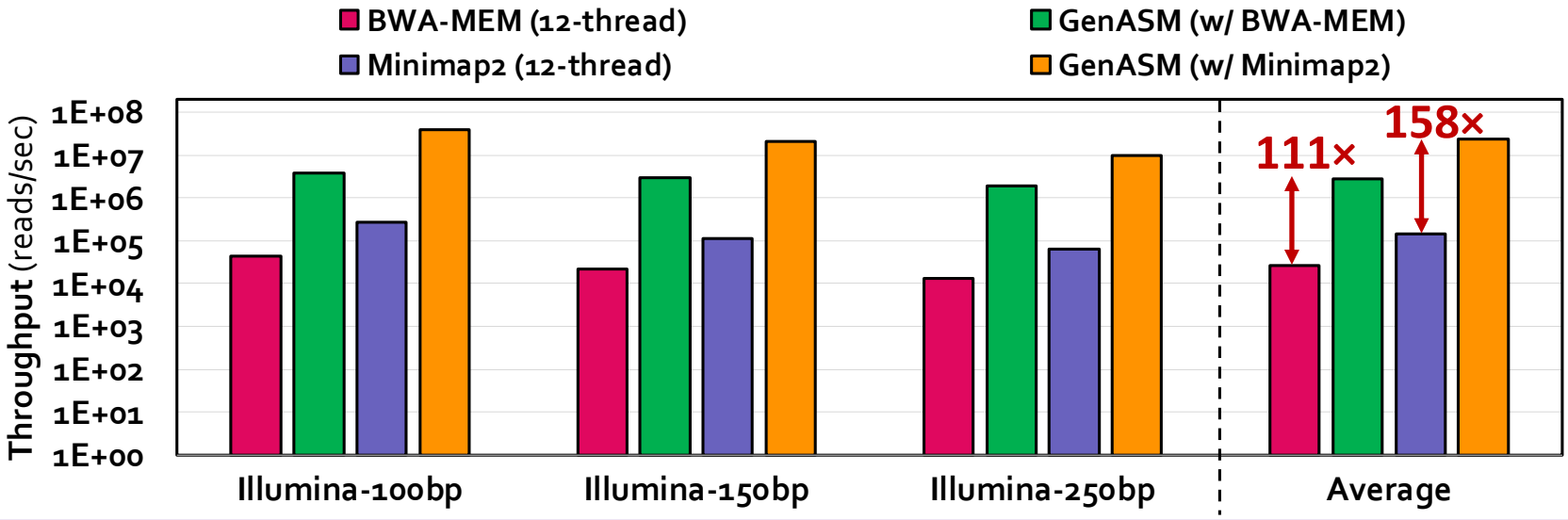
Key Results – Use Case 1 (Long Reads)



HW

GenASM provides **3.9× better throughput**,
6.6× the throughput per unit area, and
10.5× the throughput per unit power,
compared to GACT of Darwin

Key Results – Use Case 1 (Short Reads)



SW

GenASM **achieves 111× and 158× speedup** over 12-thread runs of BWA-MEM and Minimap2, while **reducing power consumption by 33× and 31×**

HW

GenASM provides **1.9× better throughput** and uses **63% less logic area** and **82% less logic power**, compared to SillaX of GenAx

Additional Details in the Paper

- Details of the **GenASM-DC and GenASM-TB algorithms**
- **Big-O analysis** of the algorithms
- Detailed explanation of **evaluated use cases**
- **Evaluation methodology details**
  (datasets, baselines, performance model)
- **Additional results** for the three evaluated use cases
- **Sources of improvements in GenASM**
  (algorithm-level, hardware-level, technology-level)
- Discussion of **four other potential use cases** of GenASM

GenASM [MICRO 2020]

Damla Senol Cali, Gurpreet S. Kalsi, Zulal Bingol, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, Anant Nori, Allison Scibisz, Sreenivas Subramoney, Can Alkan, Saugata Ghose, and Onur Mutlu,

"GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis"

*Proceedings of the 53rd International Symposium on Microarchitecture (**MICRO**)*, Virtual, October 2020.

GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis

Damla Senol Cali[†⋈] Gurpreet S. Kalsi[⋈] Zülal Bingöl[▽] Can Firtina[◇] Lavanya Subramanian[‡] Jeremie S. Kim[◇†]
Rachata Ausavarungnirun[⊙] Mohammed Alser[◇] Juan Gomez-Luna[◇] Amirali Boroumand[†] Anant Nori[⋈]
Allison Scibisz[†] Sreenivas Subramoney[⋈] Can Alkan[▽] Saugata Ghose[⋆†] Onur Mutlu[◇†▽]
[†]*Carnegie Mellon University* [⋈]*Processor Architecture Research Lab, Intel Labs* [▽]*Bilkent University* [◇]*ETH Zürich*
[‡]*Facebook* [⊙]*King Mongkut's University of Technology North Bangkok* [⋆]*University of Illinois at Urbana–Champaign*

P&S Genomics

Lecture 8a: GenASM

Joël Lindegger

ETH Zürich

Spring 2023

27 April 2023

P&S Genomics

Lecture 8b: Scrooge

Joël Lindegger

ETH Zürich

Spring 2023

27 April 2023

Scrooge

A Fast and Memory-Frugal Genomic Sequence Aligner for CPUs, GPUs, and ASICs

Joël Lindegger

Damla Senol Cali, Mohammed Alser, Juan Gómez-Luna,
Nika Mansouri Ghiasi, Onur Mutlu

April 14th 2023
RECOMB-Seq

ETH zürich

SAFARI
SAFARI Research Group

Our Goals

Build a practical and efficient implementation of the GenASM algorithm for multiple computing platforms

Compete with state-of-the-art pairwise sequence aligners like Edlib, KSW2, and BiWFA

Scrooge

Three novel algorithmic improvements which address inefficiencies in the GenASM algorithm

Efficient open-source implementations for CPUs and GPUs

Key Results

Scrooge consistently outperforms GenASM

- **2.1x speedup over GenASM on CPU**
- **5.9x speedup over GenASM on GPU**
- **3.6x better area efficiency than GenASM as an ASIC**

Scrooge consistently outperforms state-of-the-art CPU and GPU baselines, including KSW2, Edlib, and BiWFA

SAFARI

Outline

Analysis of GenASM

ASIC [Senol Cali+]

Application **S**pecific **I**ntegrated **C**ircuit

CPU

GPU

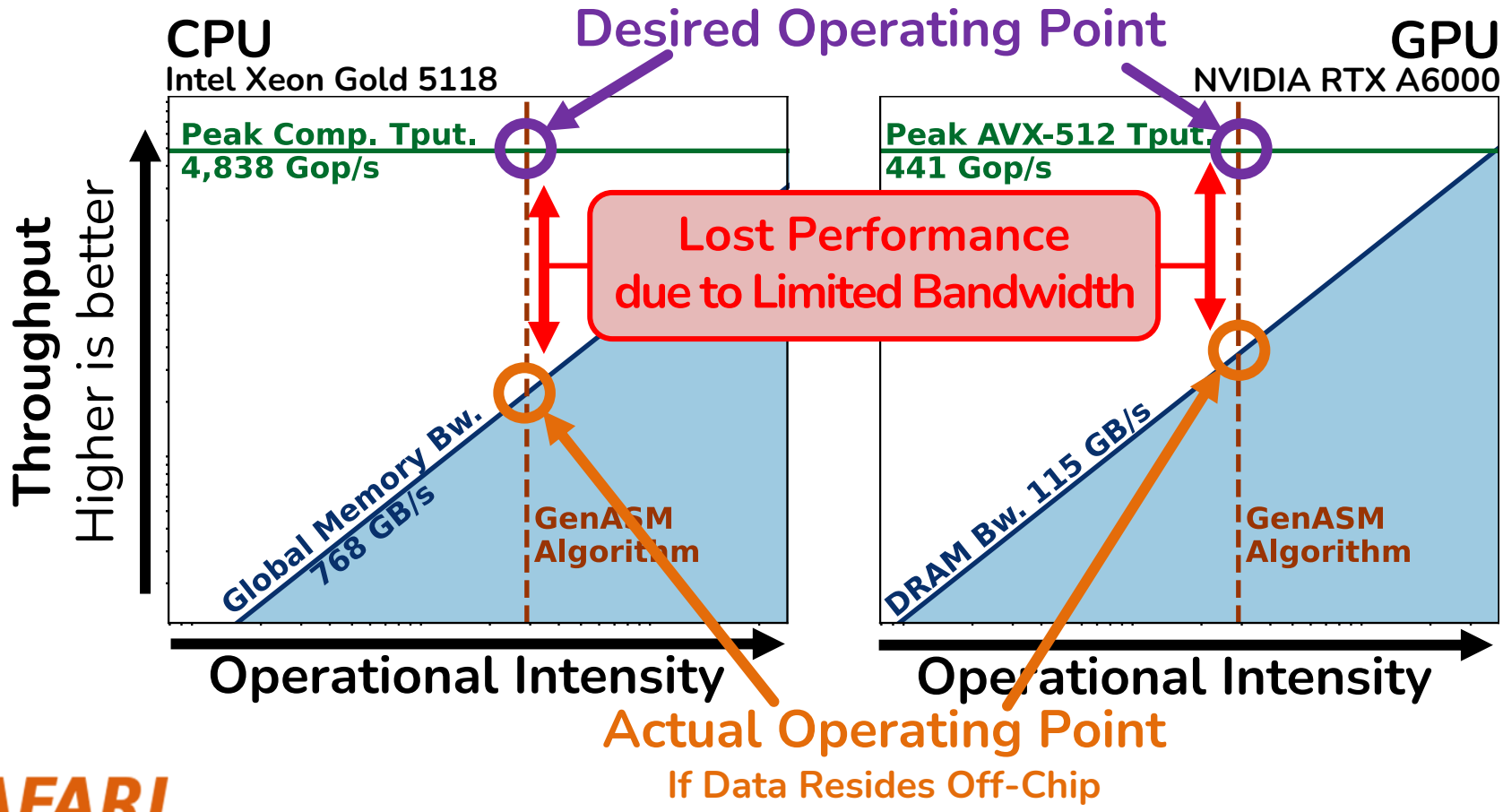
Can we do better?

Is GenASM suitable to commodity hardware?

[Senol Cali+], "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis", MICRO 2020

Roofline Analysis of GenASM

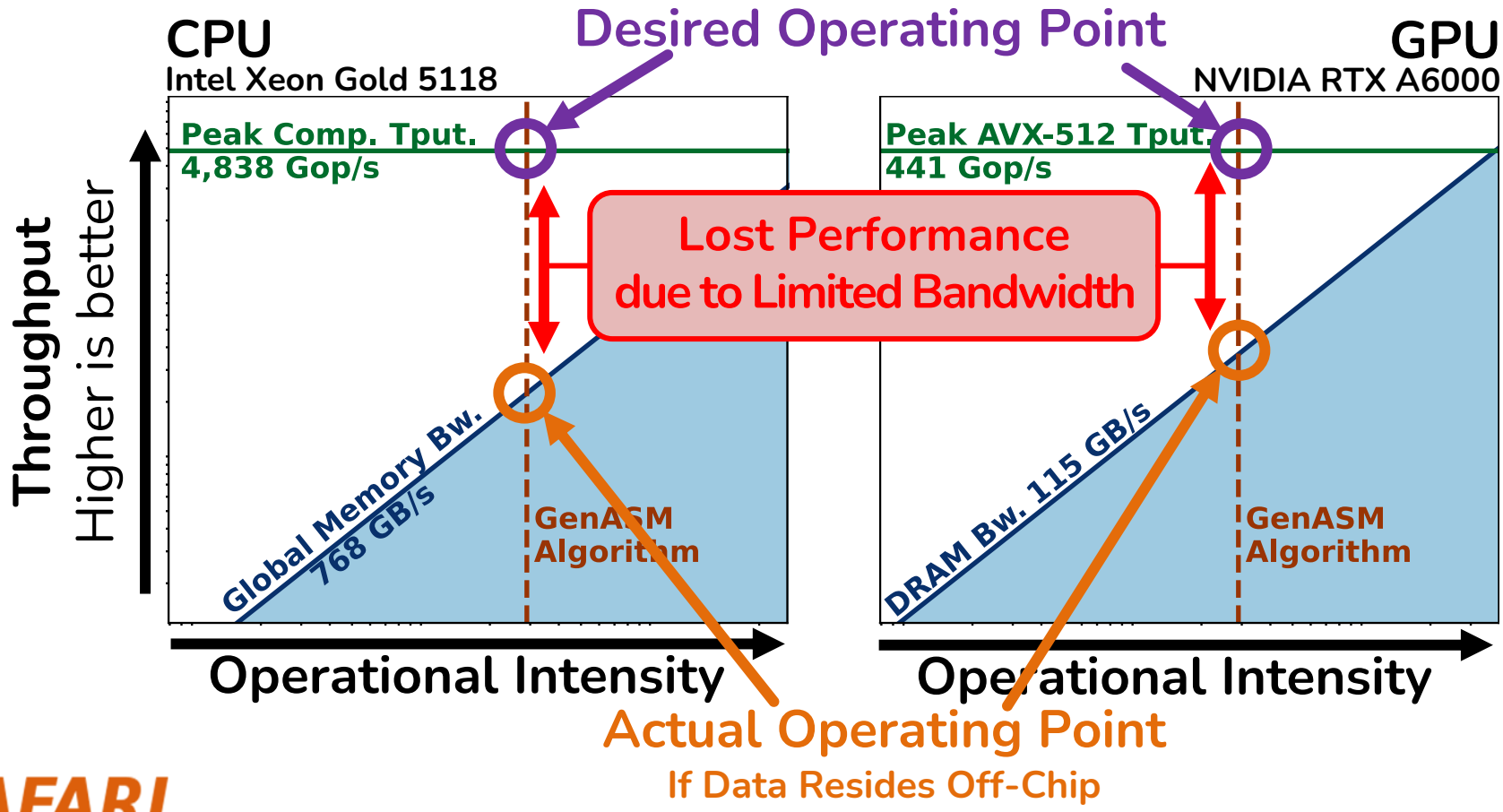
- Does commodity hardware have enough memory bandwidth for the GenASM algorithm?



Roofline Analysis of GenASM

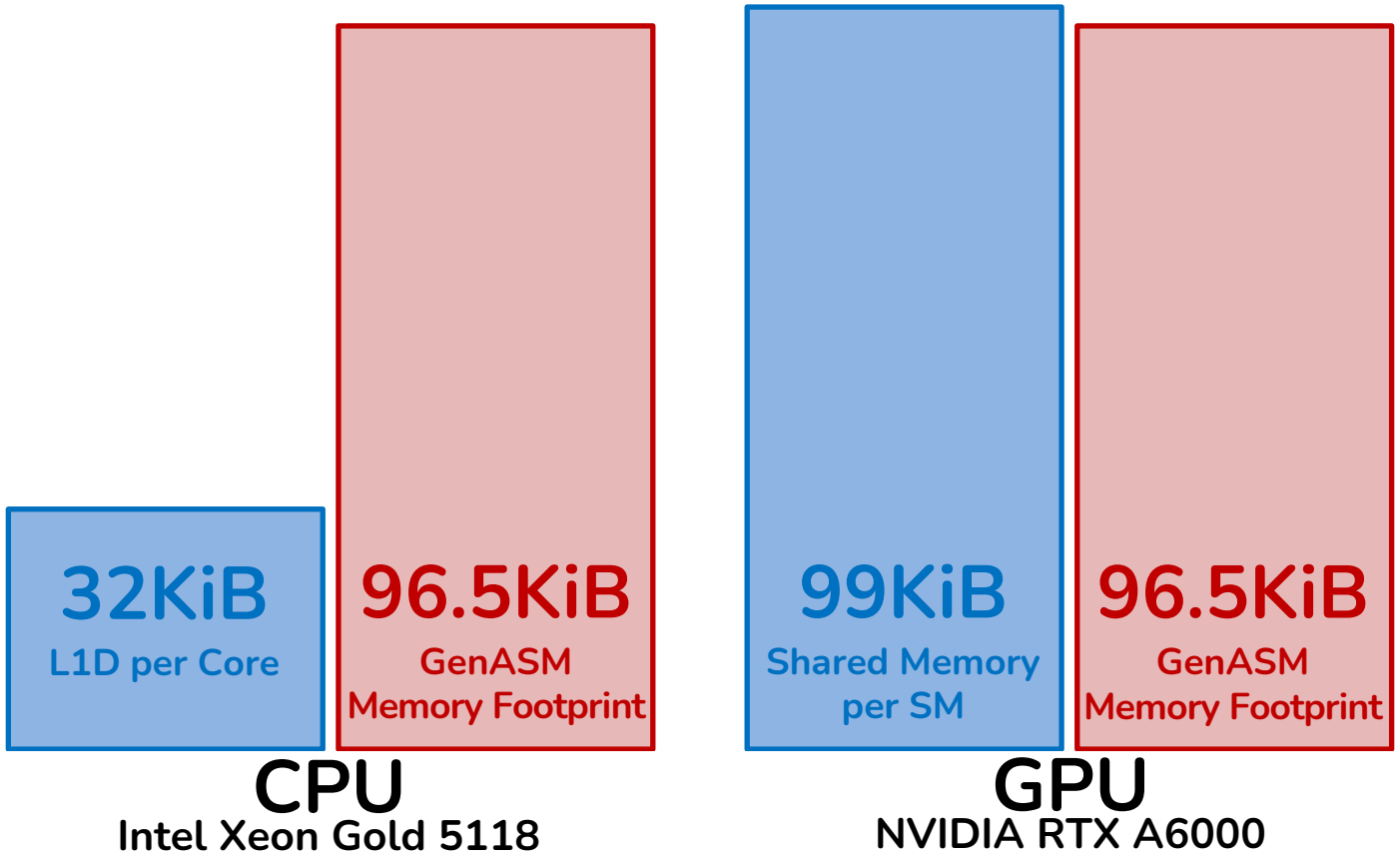
Inefficiency #1

GenASM cannot saturate commodity hardware with computation due to too much data movement



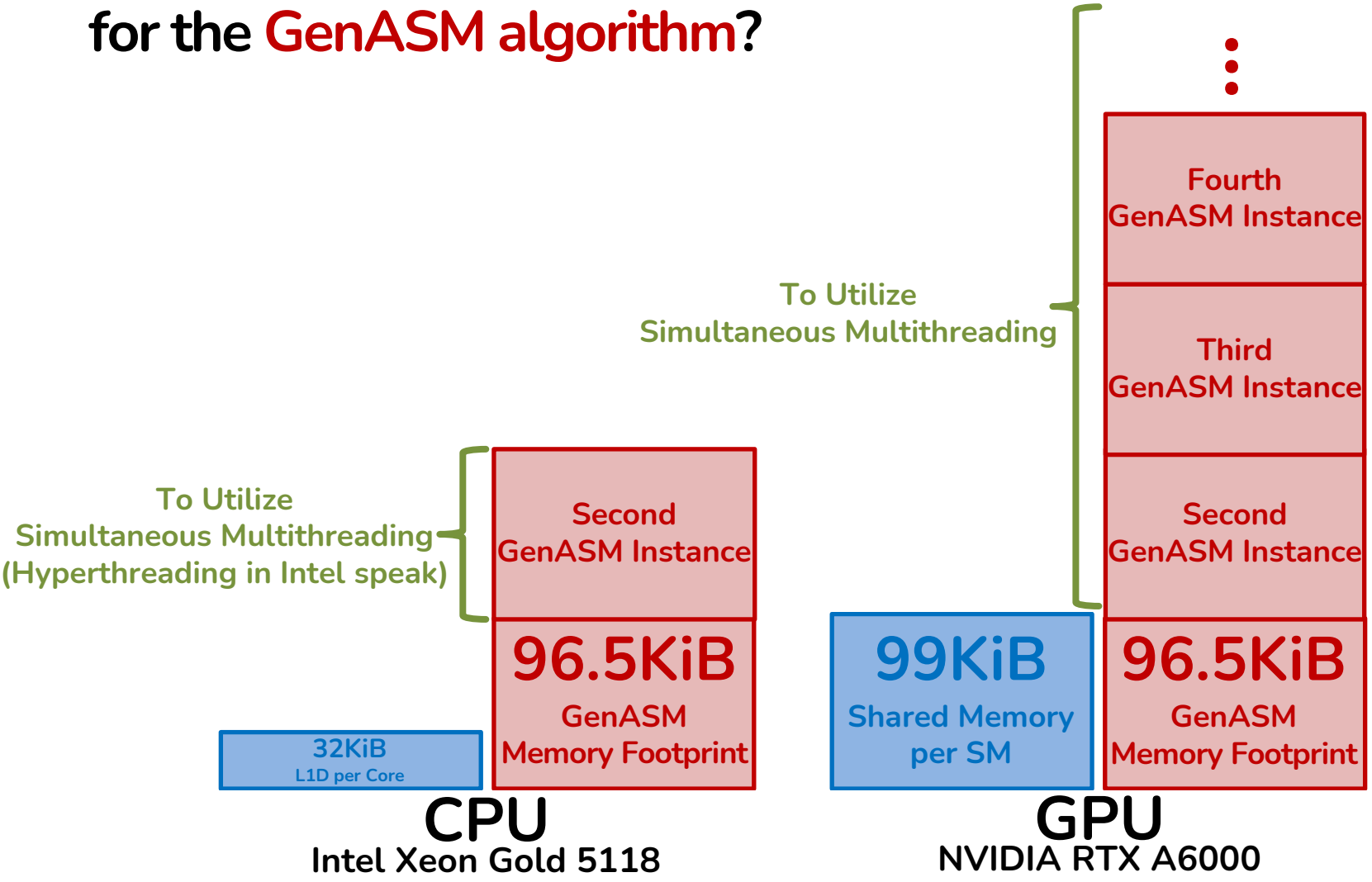
Memory Footprint Analysis of GenASM

- Does commodity hardware have enough on-chip memory for the GenASM algorithm?



Memory Footprint Analysis of GenASM

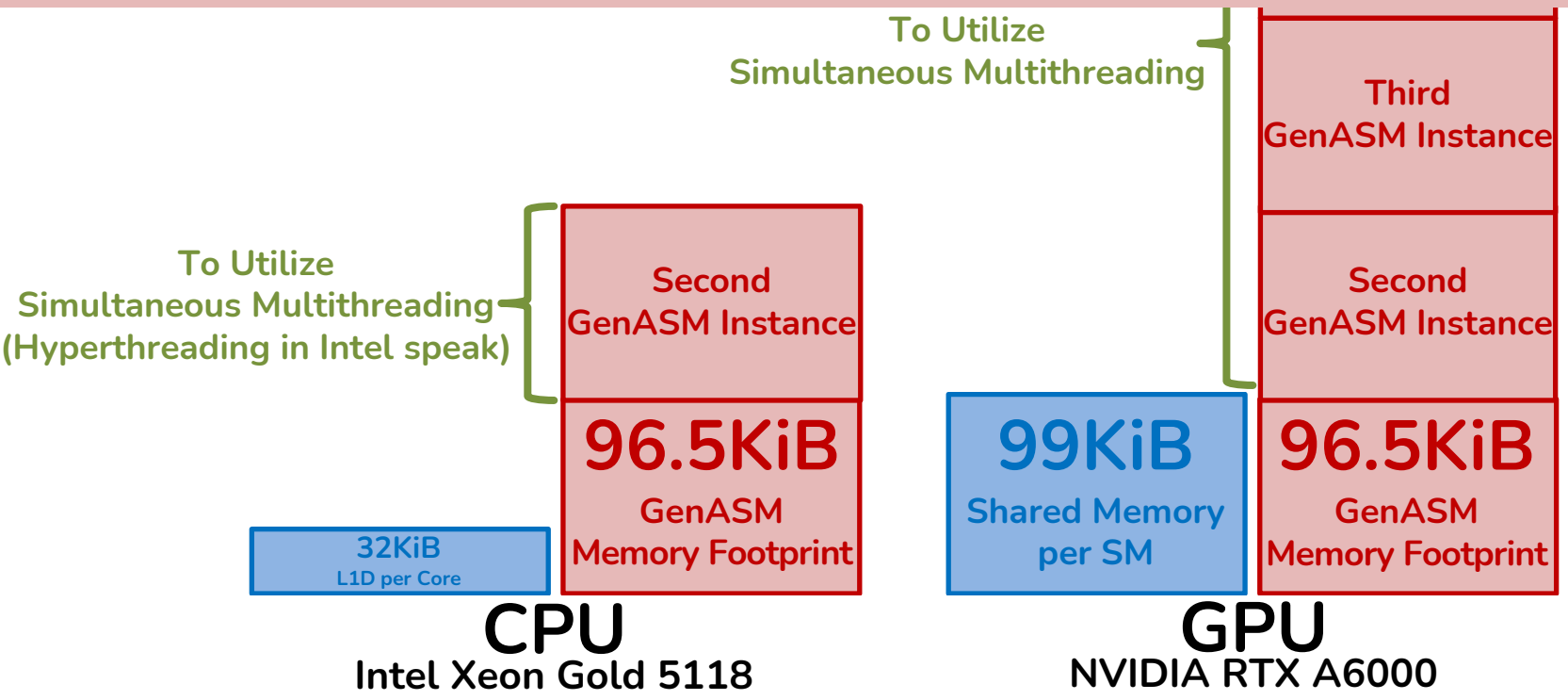
- Does commodity hardware have enough on-chip memory for the GenASM algorithm?



Memory Footprint Analysis of GenASM

Inefficiency #2

GenASM has a large memory footprint, especially when multiple instances are kept in memory for simultaneous multithreading



Unnecessary Work in GenASM

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	1010	1100	1110	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000

Cannot be Reached by Traceback

Inefficiency #3

GenASM does unnecessary work by computing DP cells which cannot be reached by Traceback

Inefficiencies in GenASM

1. Large memory bandwidth requirement

2. Large memory footprint

3. Unnecessary work

Outline

Scrooge Algorithm

Memory Improvements

reduce the memory footprint and data movement

SENE
Store Entries, not Edges

DENT
Discard Entries, not Used by Traceback

Efficiency Improvement

eliminates the unnecessary work

ET
Early Termination

Scrooge Algorithm

Memory Improvements

reduce the memory footprint and data movement

SENE
Store Entries, not Edges

DENT
Discard Entries, not Used by Traceback

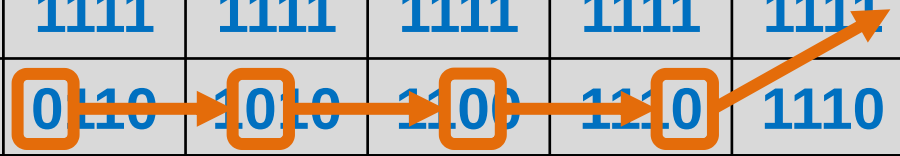
Efficiency Improvement

eliminates the unnecessary work

ET
Early Termination

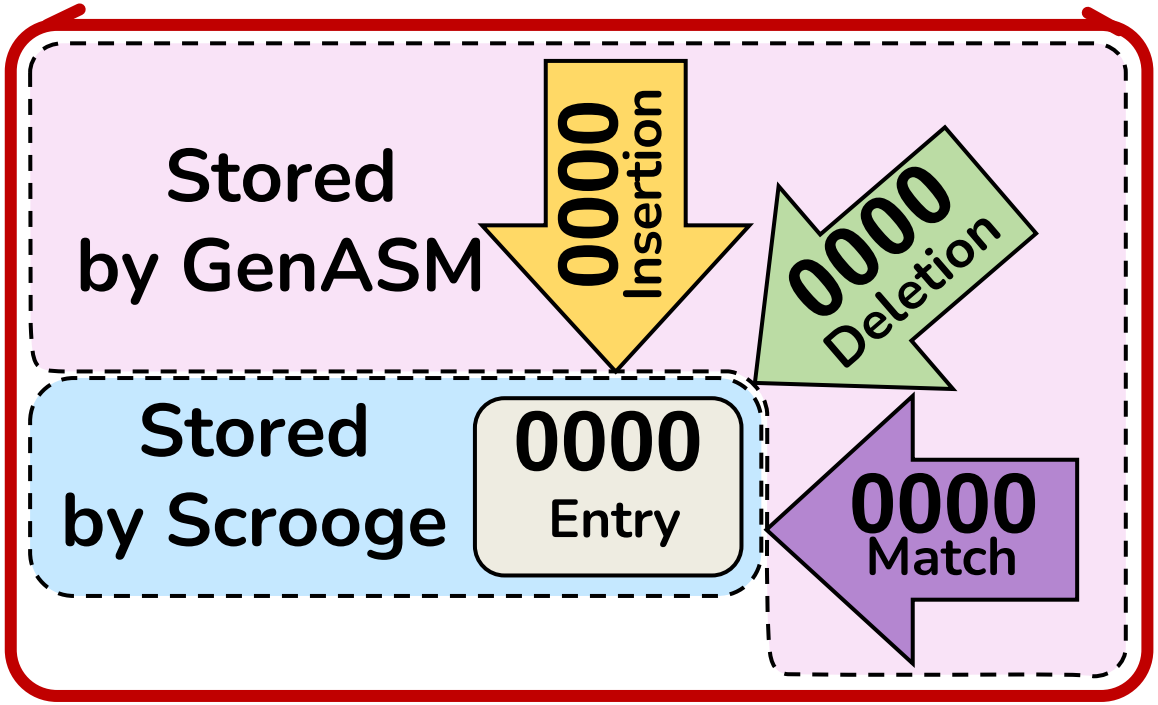
SENE: Store Entries, Not Edges

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	1010	1100	1110	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000



SENE: Store Entries, Not Edges

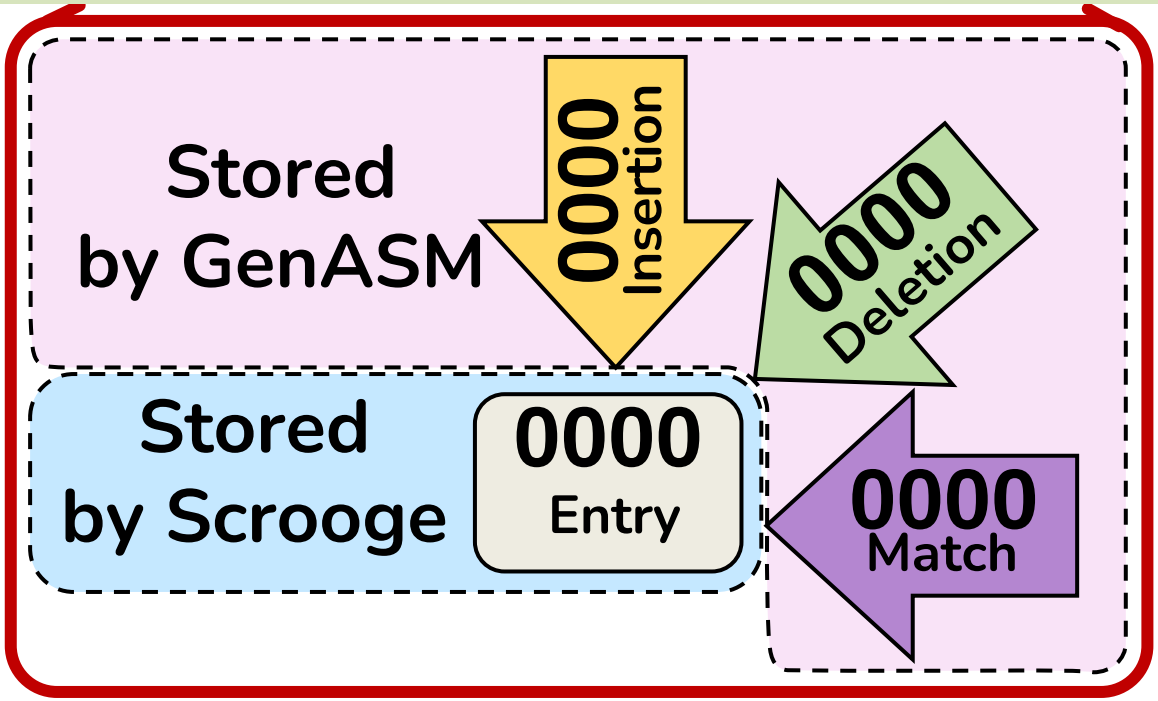
Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	0110	1110	1110	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000



SENE: Store Entries, Not Edges

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	0110	1110	1110	1110
2 Edits	0000	0000	1000	1100	1100

SENE results in a **3x reduction** in **memory footprint** and **data movement**



Scrooge Algorithm

Memory Improvements

reduce the memory footprint and data movement

SENE
Store Entries, not Edges

DENT
Discard Entries, not Used by Traceback

Efficiency Improvement

eliminates the unnecessary work

ET
Early Termination

DENT: Discard Entries Not Used by Traceback

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	1010	1100	1110	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000

Traceback is confined due to the "windowing heuristic"

Remaining bits need to be computed, but not stored

DENT results in a 4x reduction in memory footprint and data movement

Scrooge Algorithm

Memory Improvements

reduce the memory footprint and data movement

SENE
Store Entries, not Edges

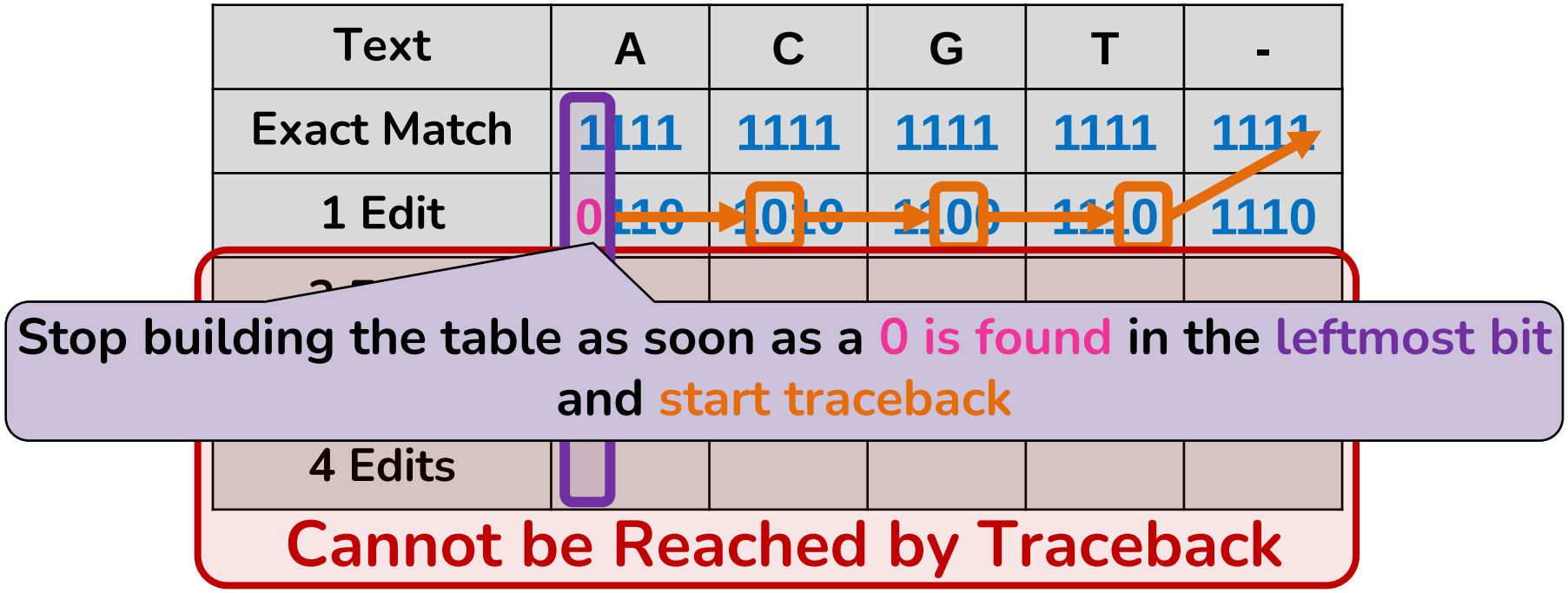
DENT
Discard Entries, not Used by Traceback

Efficiency Improvement

eliminates the unnecessary work

ET
Early Termination

ET: Early Termination



ET eliminates the unnecessary work

on average, at least 25% of cells are unnecessary

Outline

Scrooge CPU & GPU Implementations

- **We provide efficient open-source implementations of the Scrooge algorithm for CPUs and GPUs**
  - Easy-to-use library interface

- **CPU version**
  - C++
  - OpenMP for multithreading

- **GPU version**
  - C++
  - NVIDIA GPUs
    - CUDA 11.1
    - Compute capability 7.0+

Scrooge on GitHub

README.md

Scrooge: A fast and memory-frugal genomic sequence aligner for CPUs, GPUs and ASICs

Scrooge is a fast pairwise genomic sequence aligner. It efficiently aligns short and long genomic sequence pairs on multiple computing platforms. It is based on the GenASM algorithm (Senol Cali+, 2020), and adds multiple algorithmic improvements that significantly improve the throughput and resource efficiency for CPUs, GPUs and ASICs. For long reads, the CPU version of Scrooge achieves a 20.1x, 1.7x, and 2.1x speedup over KSW2, Edlib, and a CPU implementation of GenASM, respectively. The GPU version of Scrooge achieves a 4.0x 80.4x, 6.8x, 12.6x and 5.9x speedup over the CPU version of Scrooge, KSW2, Edlib, Darwin-GPU, and a GPU implementation of GenASM, respectively. We estimate an ASIC implementation of Scrooge to use 3.6x less chip area and 2.1x less power than a GenASM ASIC while maintaining the same throughput.

This repository contains Scrooge's CPU and GPU implementations, and several evaluation scripts. We describe Scrooge in our paper on arXiv and in Bioinformatics.

3 forks

Report repository

Releases

No releases published

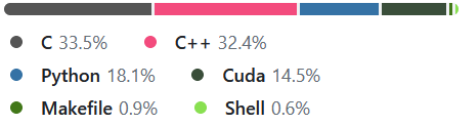
Create a new release

Packages

No packages published

Publish your first package

Languages



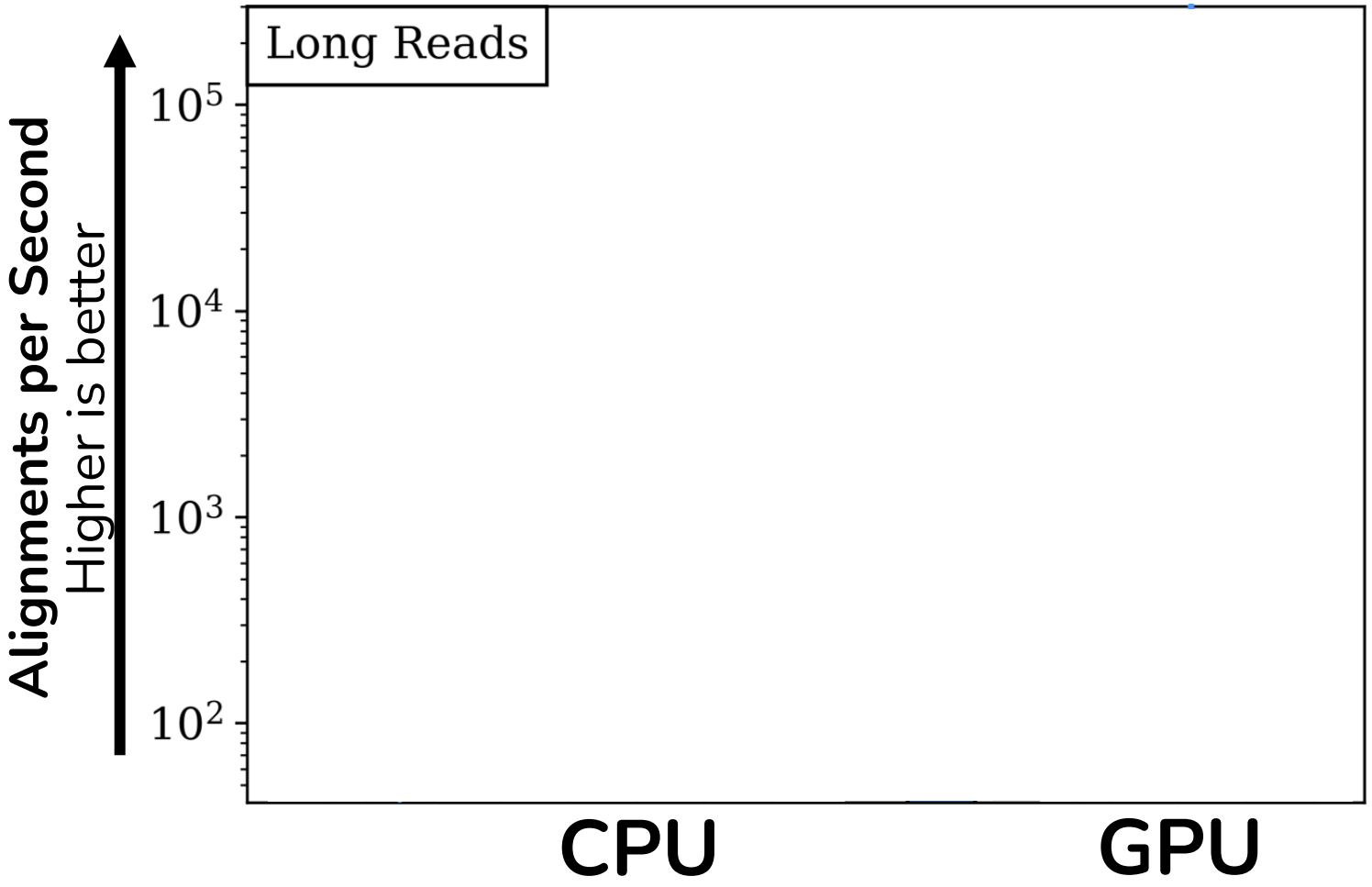
Scrooge on GitHub

Outline

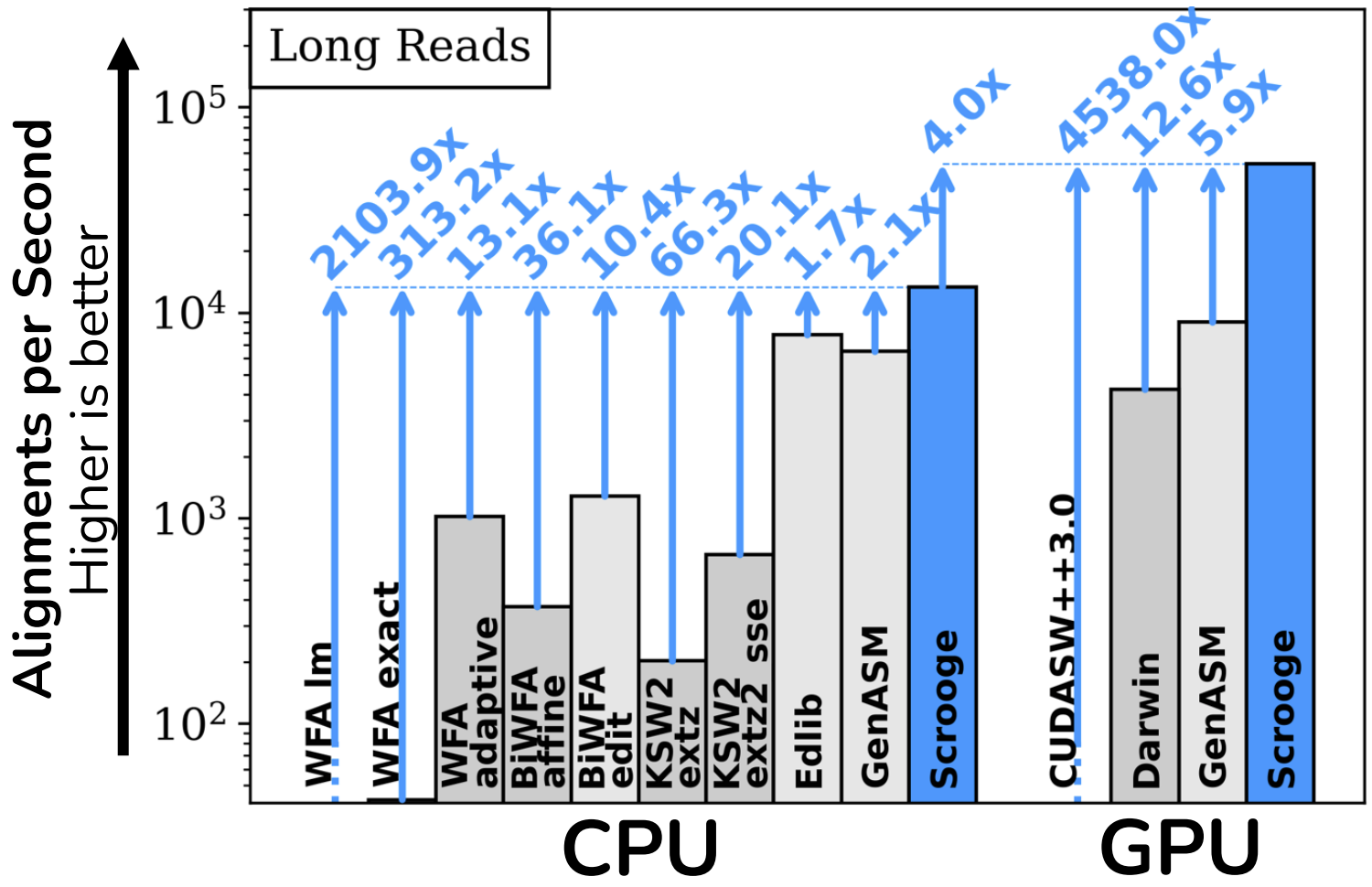
Methodology

- **Datasets**
  - Long reads
    - Simulated with PBSIM2 from the human reference genome GRCh38.p13
    - Chained with minimap2 to obtain 138,929 candidate pairs
  - Short reads
    - Illumina reads from SRR13278681
    - Chained with minimap2 to obtain 9,612,222 candidate pairs
- **CPU: dual-socket Intel Xeon Gold 5118**
  - 2× 12 physical cores, 2× 24 logical cores @ 3.2GHz
  - 196GiB DDR4 RAM
- **GPU: NVIDIA RTX A6000**
- **ASIC**
  - 28nm logic synthesis from [Senol Cali+]
  - SRAM numbers from CACTI 7

Long Read Throughput

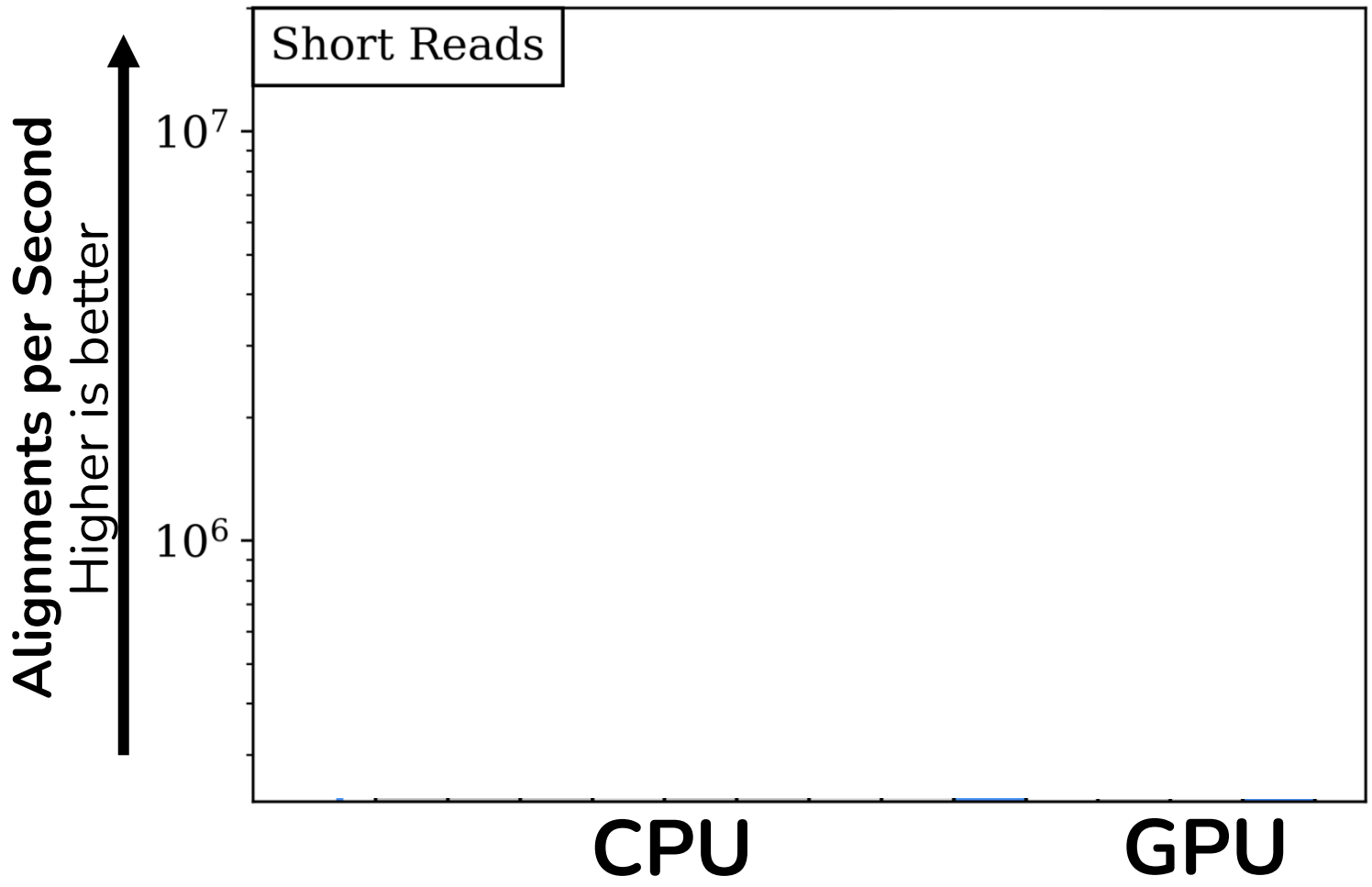


Long Read Throughput

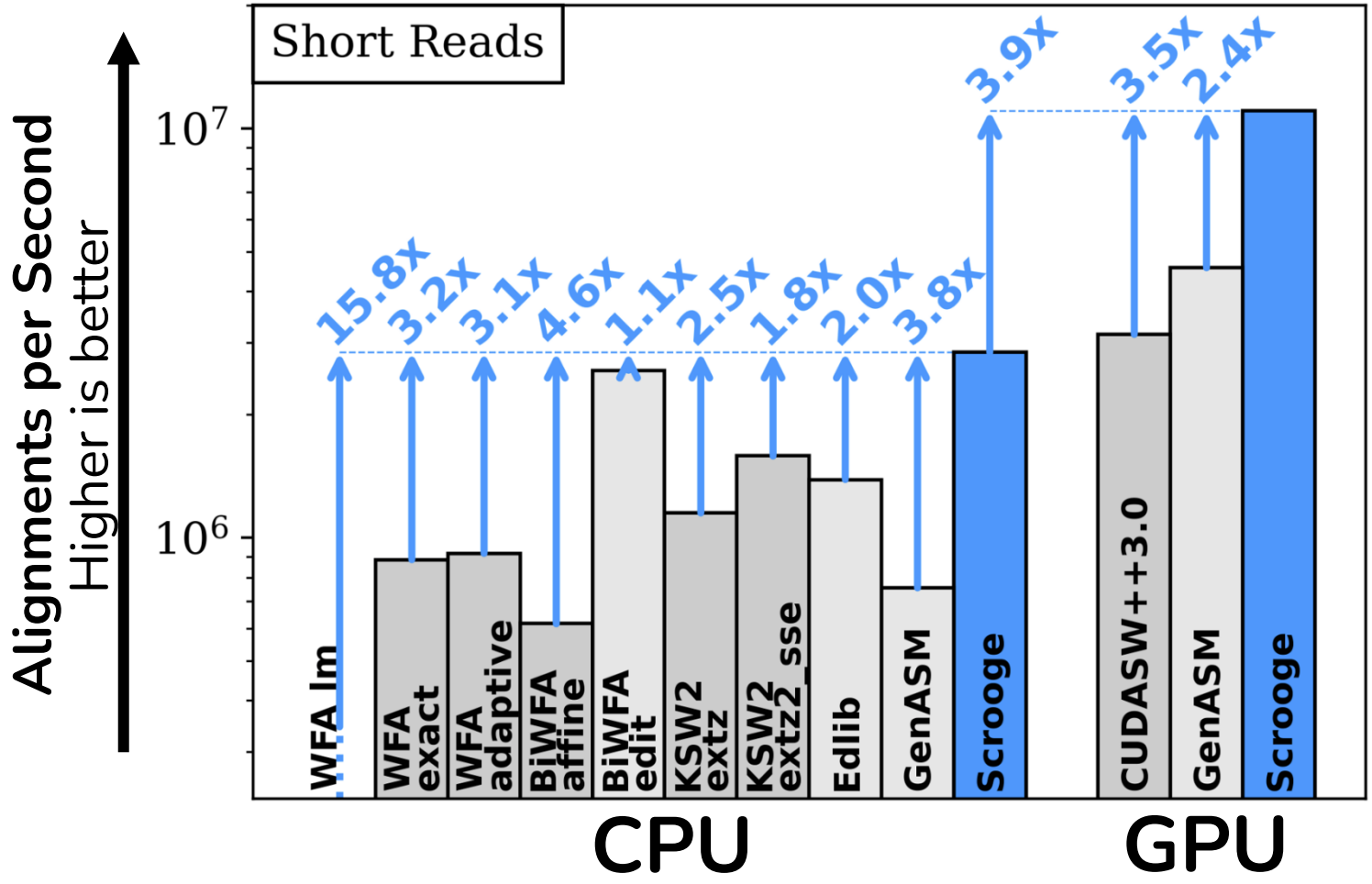


For long reads, Scrooge outperforms GenASM by 2.1x on CPU and 5.9x on GPU

Short Read Throughput



Short Read Throughput



For short reads, Scrooge outperforms GenASM by 3.8x on CPU and 2.4x on GPU

ASIC Results

Scrooge introduces
no significant computation overheads
over a GenASM ASIC

Scrooge's on-chip memory is much cheaper than GenASM's
due to the memory footprint and bandwidth reductions
(uses 18x less chip area and 18x less power)

Scrooge uses 3.6x less chip area
and 2.1x less power than a GenASM ASIC

More in the Paper: Evaluation

- Throughput sensitivity to each algorithmic improvement
- Thread scaling results
- Rigorous accuracy analysis
- Sensitivity analysis of throughput and accuracy
- ASIC breakdown

More in the Paper

Bioinformatics

ISCB INTERNATIONAL SOCIETY FOR COMPUTATIONAL BIOLOGY

Issues Advance articles Submit ▾ Alerts About ▾

Article Navigation

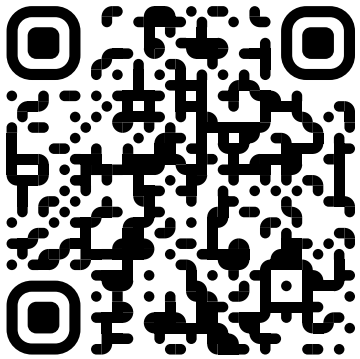
JOURNAL ARTICLE ACCEPTED MANUSCRIPT

Scrooge: A Fast and Memory-Frugal Genomic Sequence Aligner for CPUs, GPUs, and ASICs

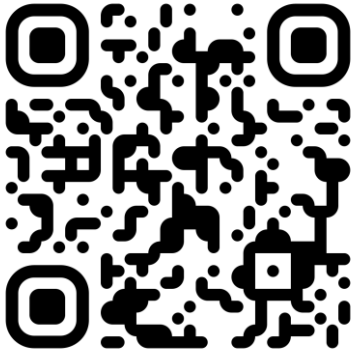
Joël Lindegger ✉, Damla Senol Cali, Mohammed Alser, Juan Gómez-Luna, Nika Mansouri Ghiasi, Onur Mutlu ✉

Bioinformatics, btad151, https://doi.org/10.1093/bioinformatics/btad151

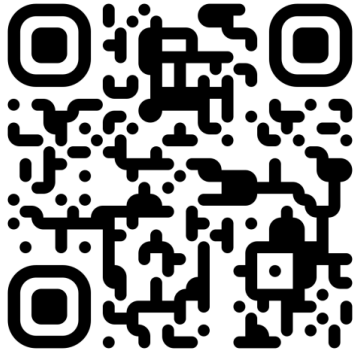
Published: 24 March 2023 **Article history** ▾



Scrooge in Bioinformatics



Scrooge on arXiv



Scrooge on GitHub

Outline

Conclusion

Motivation	**Pairwise sequence alignment (PSA)** is computationally costly and common step in bioinformatics pipelines. **GenASM** is a promising candidate for efficient PSA. For example, its ASIC implementation is up to **10,000x faster** than prior software aligners.
Goals	• Build a **practical** and **efficient implementation** of the GenASM algorithm for **multiple computing platforms** • **Compete** with **state-of-the-art pairwise sequence aligners** like Edlib, KSW2, and BiWFA
Scrooge	•**Three novel algorithmic improvements** address GenASM's inefficiencies •Efficient **open-source CPU** and **GPU** implementations
Key Results	Scrooge **consistently outperforms GenASM** • **2.1x speedup** over GenASM on CPU • **5.9x speedup** over GenASM on GPU • **3.6x** better **area efficiency** than GenASM on ASIC Scrooge **consistently outperforms state-of-the-art CPU and GPU baselines**, including KSW2, Edlib, and BiWFA

P&S Genomics

Lecture 8b: Scrooge

Joël Lindegger

ETH Zürich

Spring 2023

27 April 2023

Backup Slides

ASIC Breakdown

Scrooge has insignificant computation overheads

Significant resource savings from memory footprint and bandwidth reductions

ASIC Implementation	Area (mm^2)					Power (W)				
	DC Logic	TB Logic	DC SRAM	TB SRAM	total	DC Logic	TB Logic	DC SRAM	TB SRAM	total
GenASM	0.049	0.016	0.013	0.256	0.334	0.033	0.004	0.009	0.055	0.101
Scrooge	0.049	0.016	0.013	0.014	0.093	0.033	0.004	0.009	0.003	0.049

Scrooge uses 3.6x less chip area and 2.1x less power than a GenASM ASIC

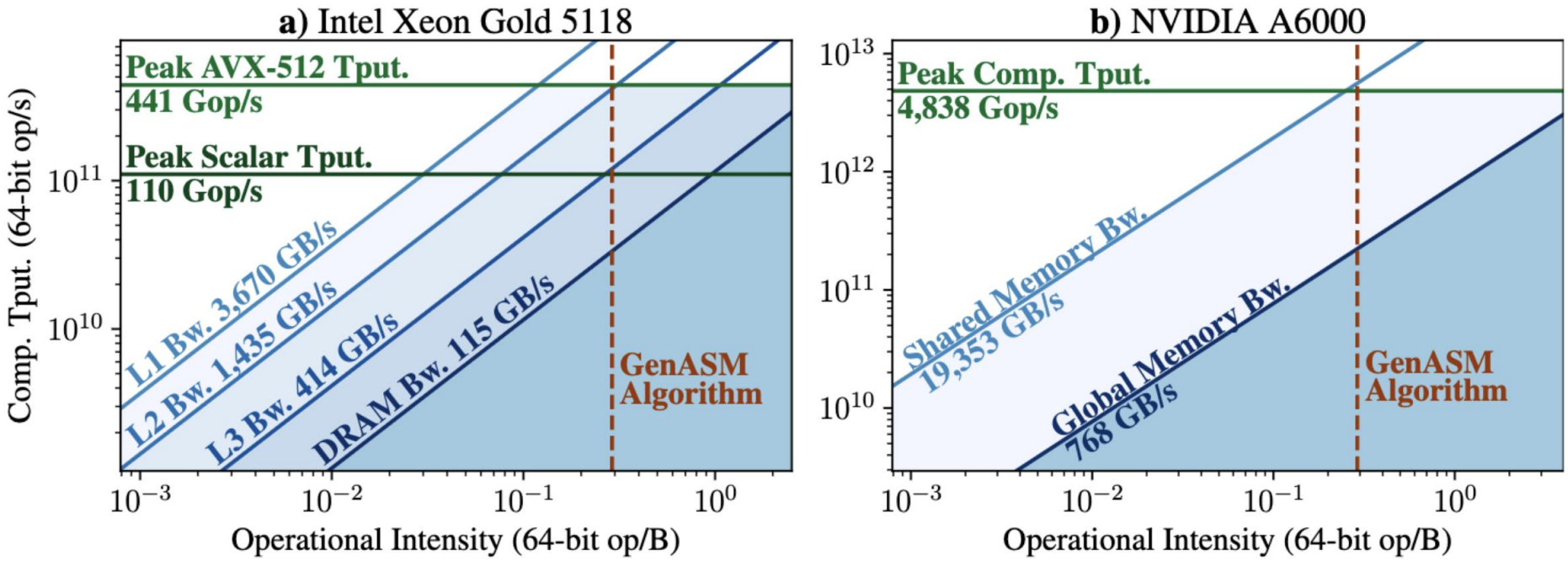
GenASM-DC Algorithm

Algorithm 1 GenASM-DC Algorithm

Inputs: text, pattern, k
Outputs: editDist

```
1:  n ← LENGTH(text)
2:  m ← LENGTH(pattern)
3:  PM ← BUILDPATTERNMASKS(pattern)
4:
5:  R[n][d] ← 11...1 ≪ d                    ▷ Initialize for all 0 ≤ d ≤ k
6:
7:  for i in (n − 1) : −1 : 0 do
8:      char ← text[i]
9:      curPM ← PM[char]
10:
11:     R[i][0] ← (R[i + 1][0] ≪ 1) | curPM          ▷ exact match
12:     for d in 1 : k do
13:         I ← R[i][d − 1] ≪ 1                      ▷ insertion
14:         D ← R[i + 1][d − 1]                      ▷ deletion
15:         S ← R[i + 1][d − 1] ≪ 1                  ▷ substitution
16:         M ← (R[i + 1][d] ≪ 1) | curPM            ▷ match
17:         R[i][d] ← I & D & S & M
18:
19: editDist ← arg min_d {MSB(R[0][d]) = 0}
```

Fulls Roofline Models

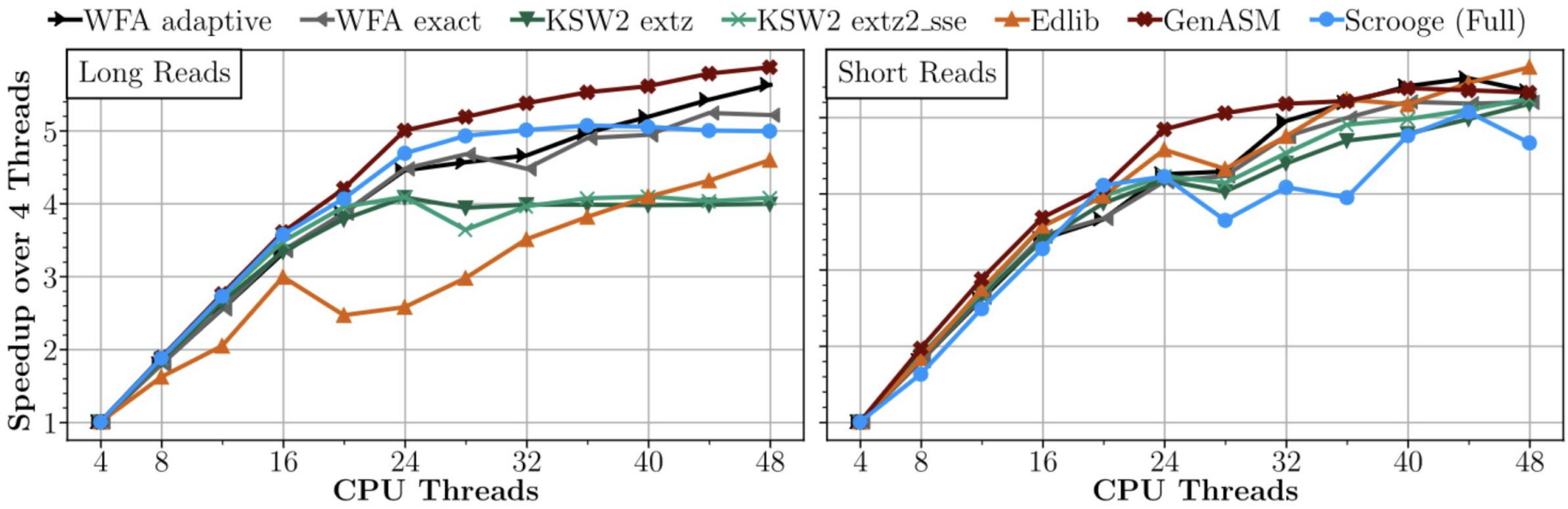


Bitvector Interpretation

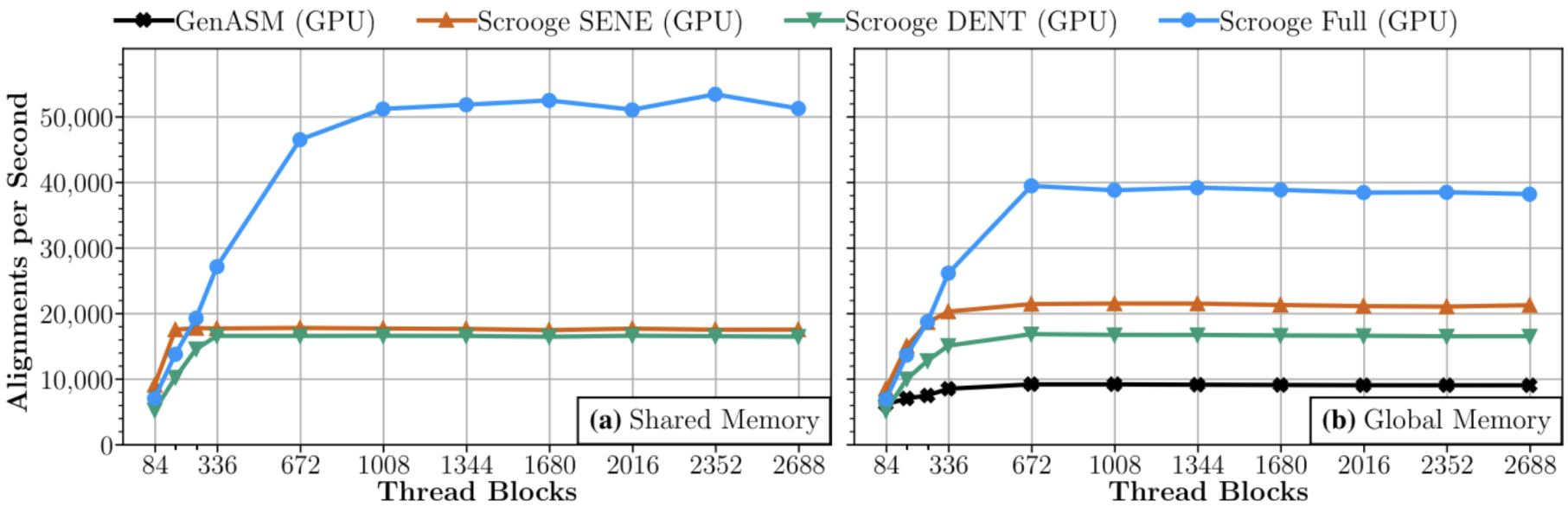
Theorem 1 *The entries (bitvectors) of* R *can be interpreted as follows:*

$$j\text{-th bit of } R[i][d] = 0 \iff distance\big(\texttt{text}[i:n), \texttt{pattern}[j:m)\big) \leq d$$

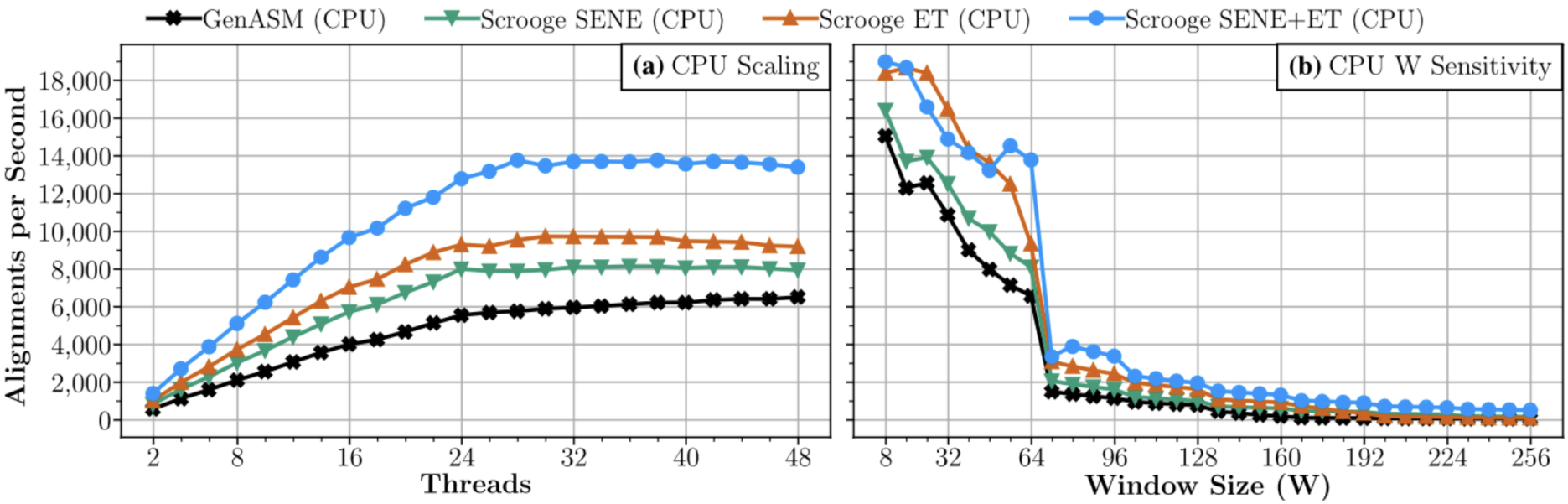
CPU Thread Scaling



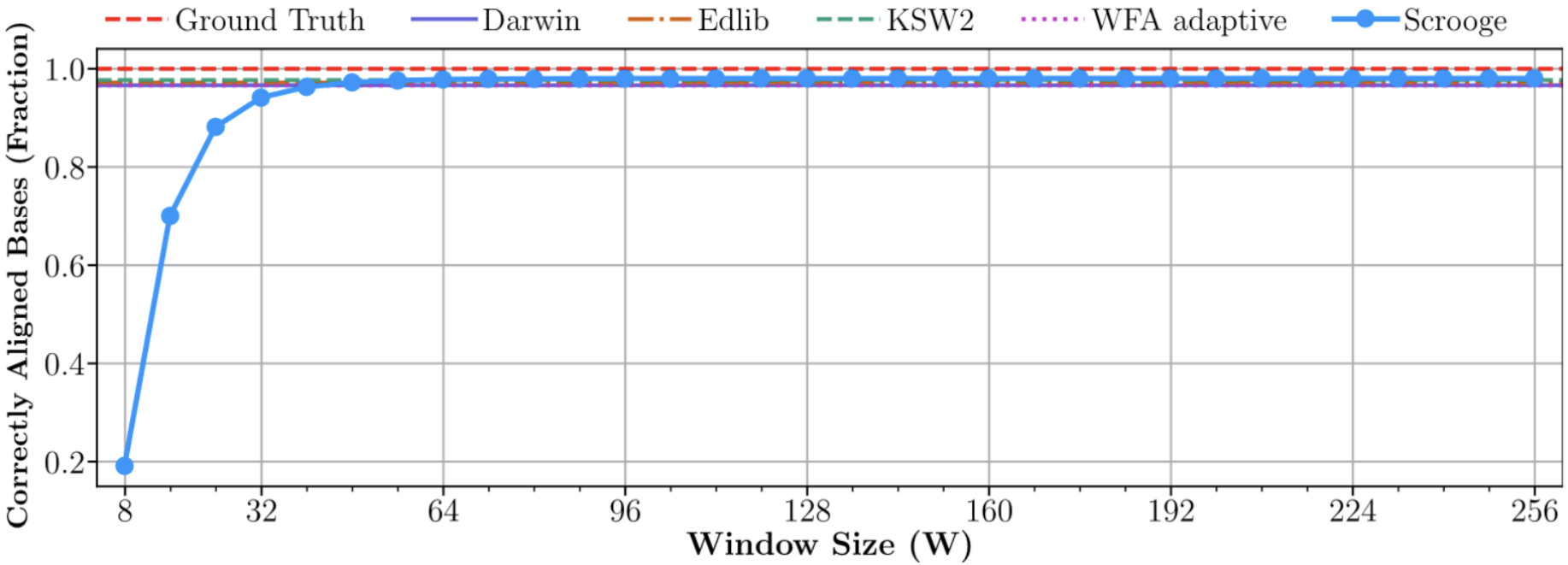
GPU Thread Scaling



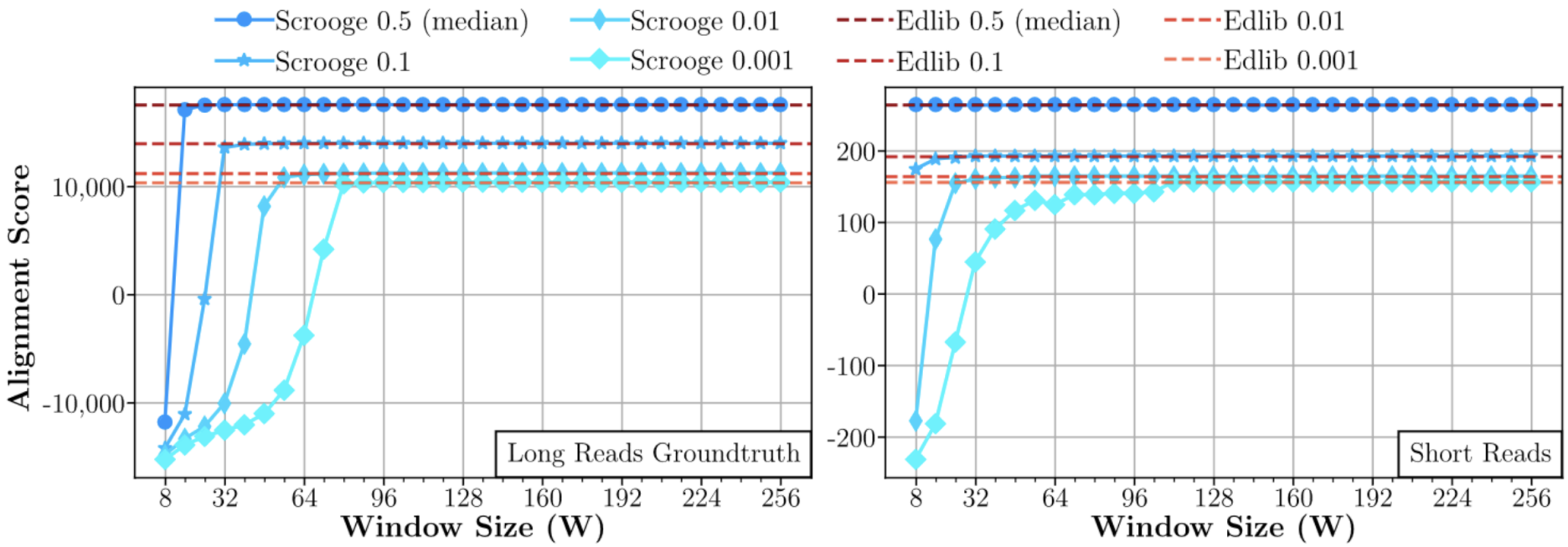
CPU Optimization Sensitivity



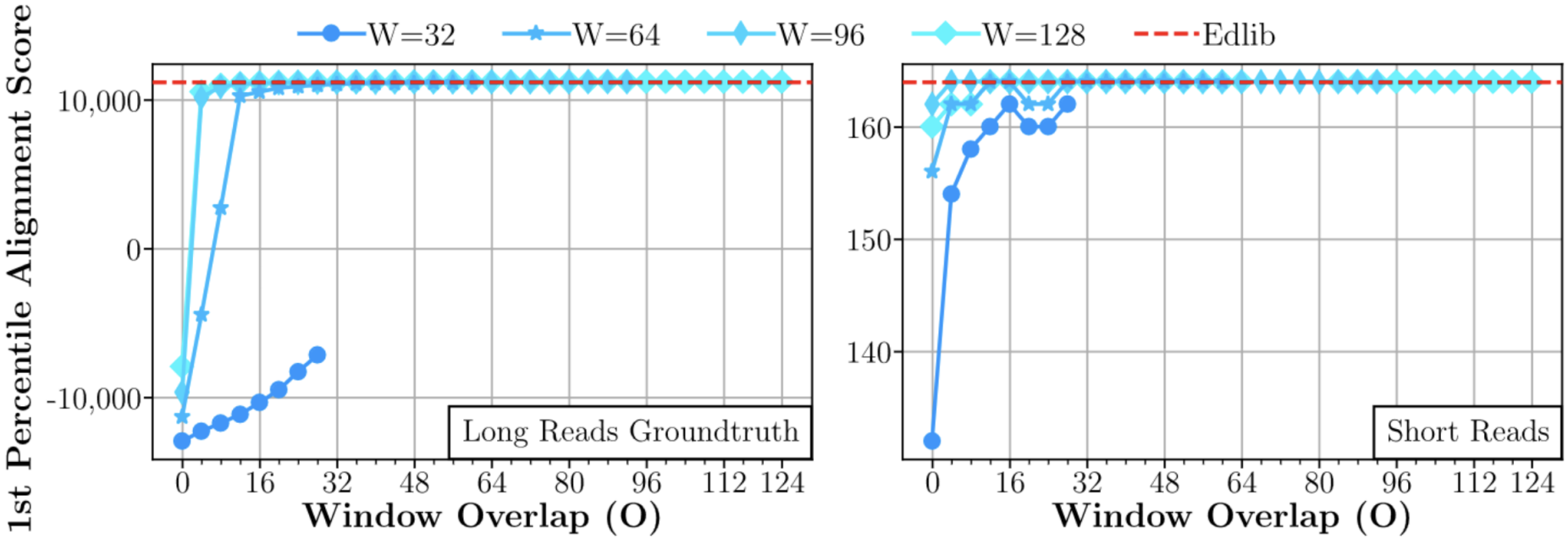
Accuracy Comparison



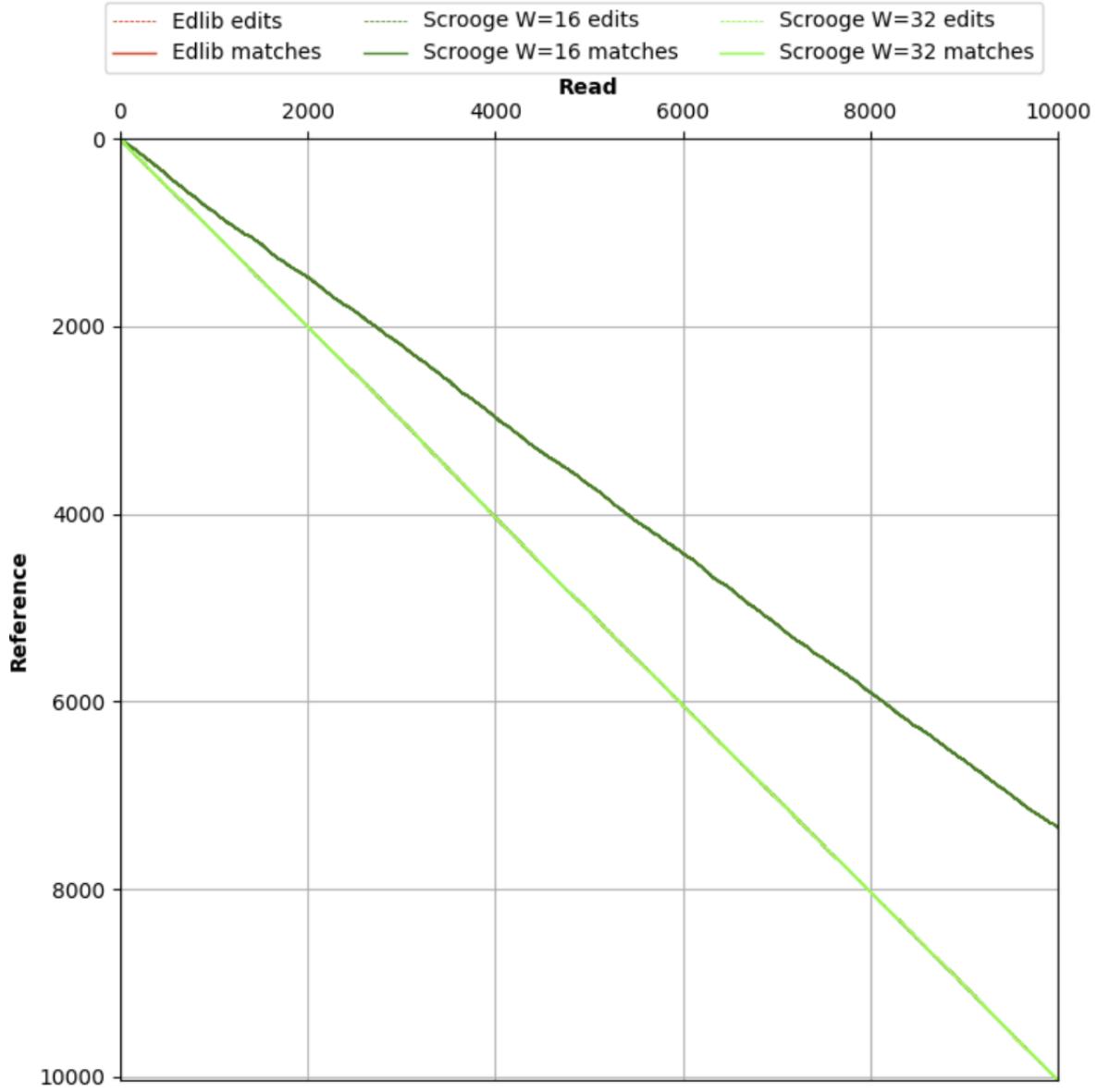
Accuracy Sensitivity to Window Size W



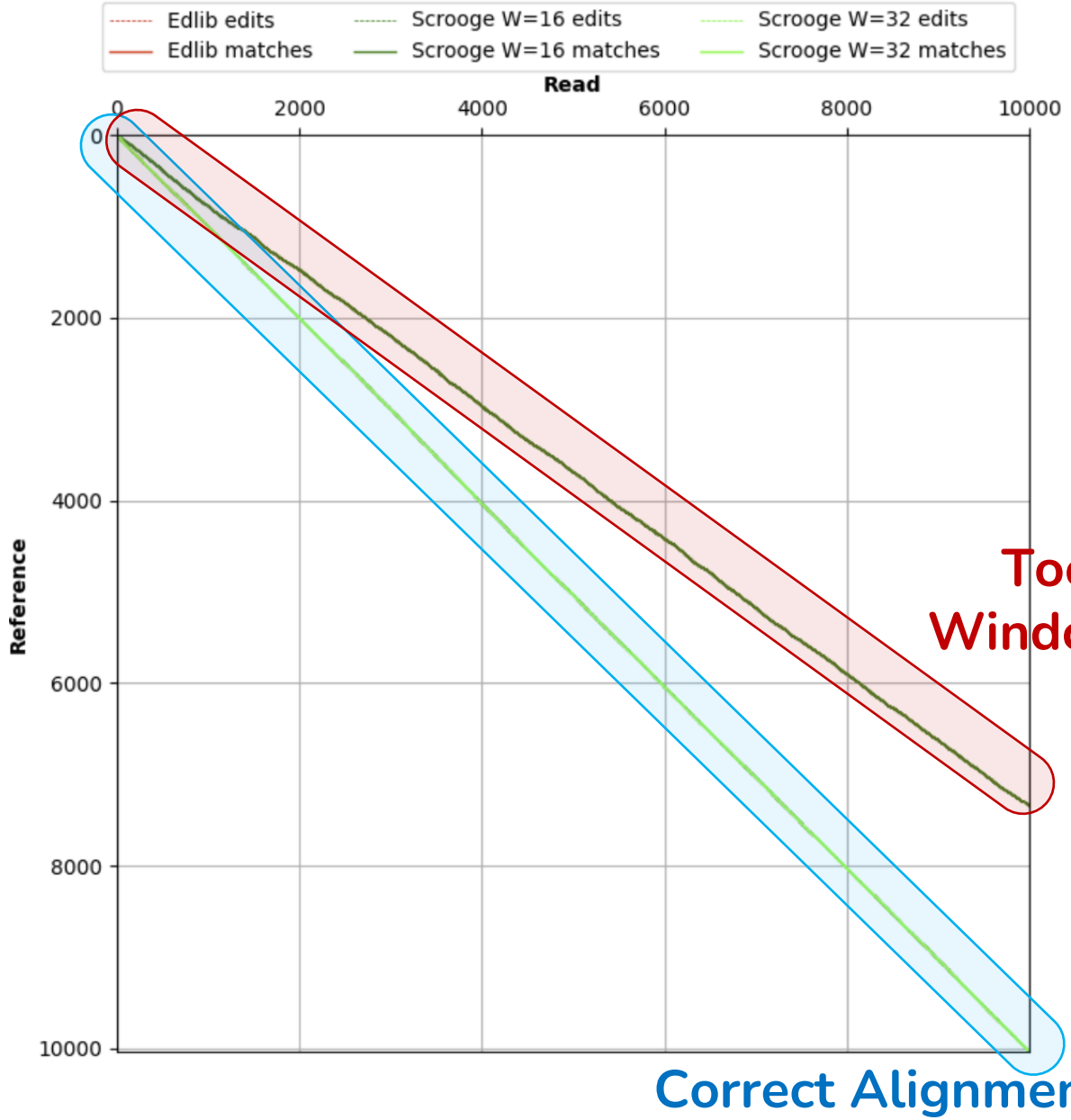
Accuracy Sensitivity to Window Overlap O



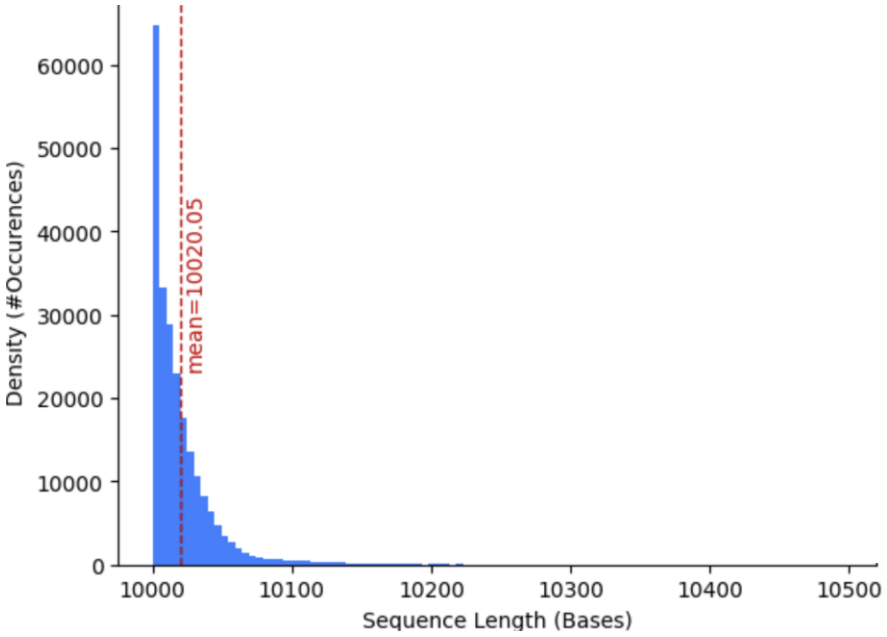
Failure Mode for Too Small Window Size W

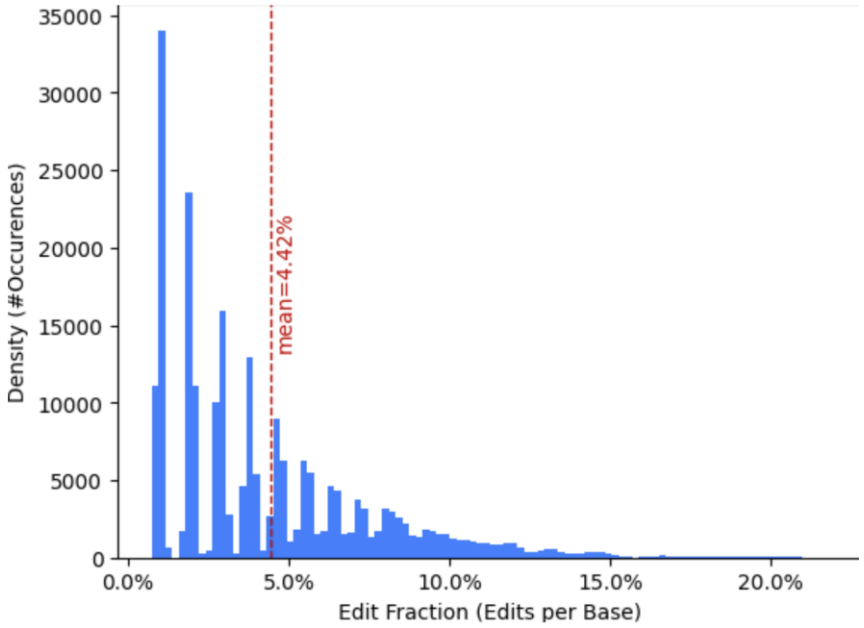


Failure Mode for Too Small Window Size W

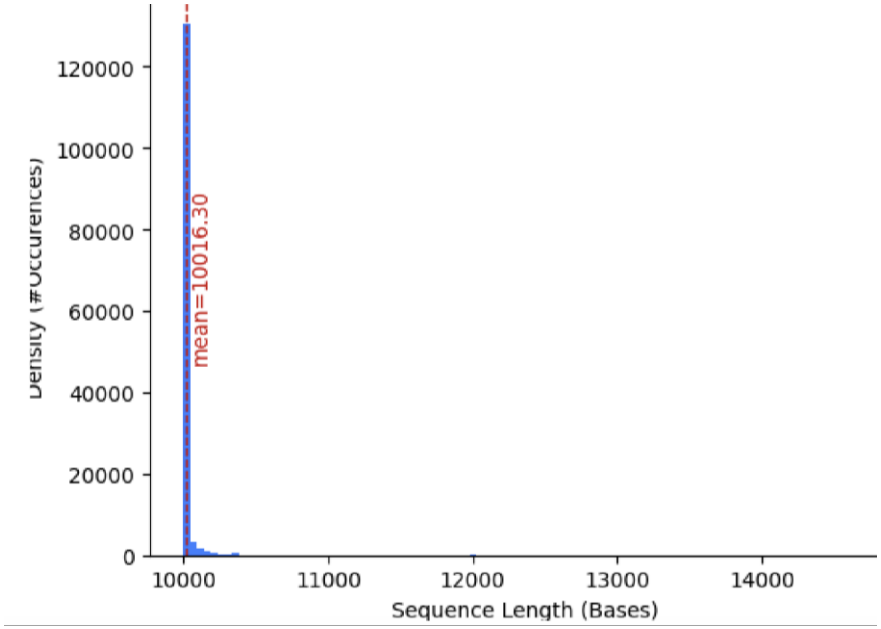


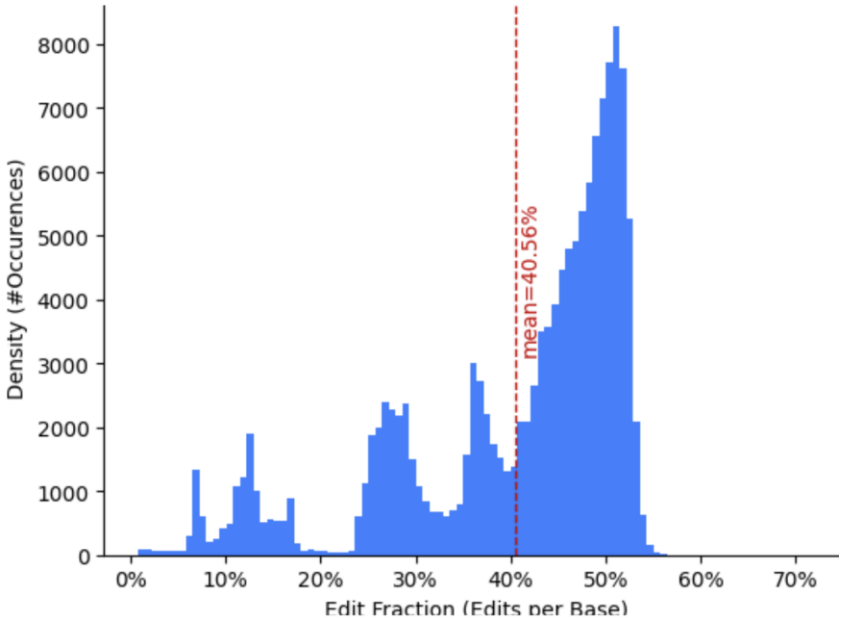
Long Read Dataset (Ground Truth)





Long Read Dataset





Short Read Dataset