GenGraph: A Hardware Acceleration Framework for Sequence-to-Graph Mapping

Damla Senol Cali *et al.*

https://damlasenolcali.github.io

TECHCON'21 – September 14, 2021

Carnegie Mellon ETH zürich Bilkent University

intel UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN SAFARI SRC Semiconductor Research Corporation

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

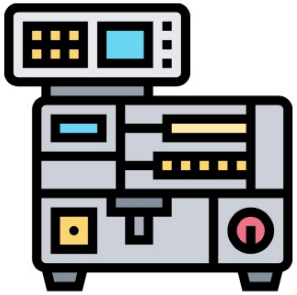
DNA

- **Challenges:**
  - There is no sequencing machine that takes long DNA as an input, and gives the complete sequence as output
  - Sequencing machines extract small randomized fragments of the original DNA sequence

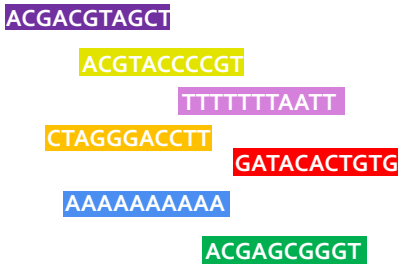
Genome Sequencing (cont'd.)

Sample Collection

Chopped DNA fragments



Sequencing



Sequenced reads

Large DNA molecule

Preparation

Genome Sequence Analysis

Sequencing Technologies

Oxford Nanopore (ONT)

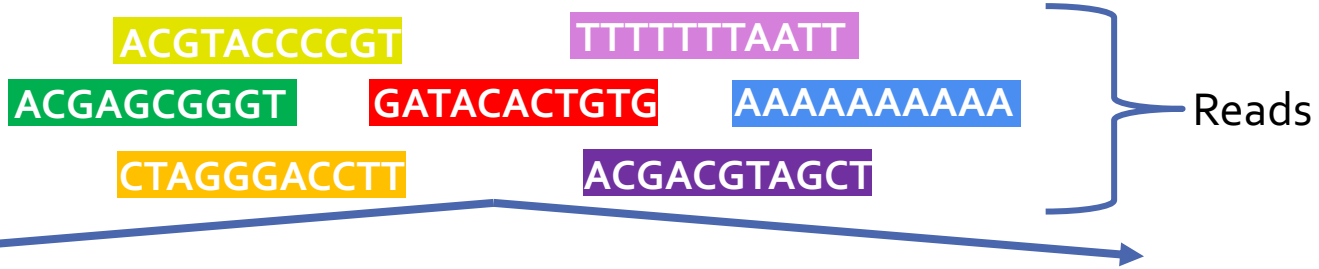
PacBio

Illumina

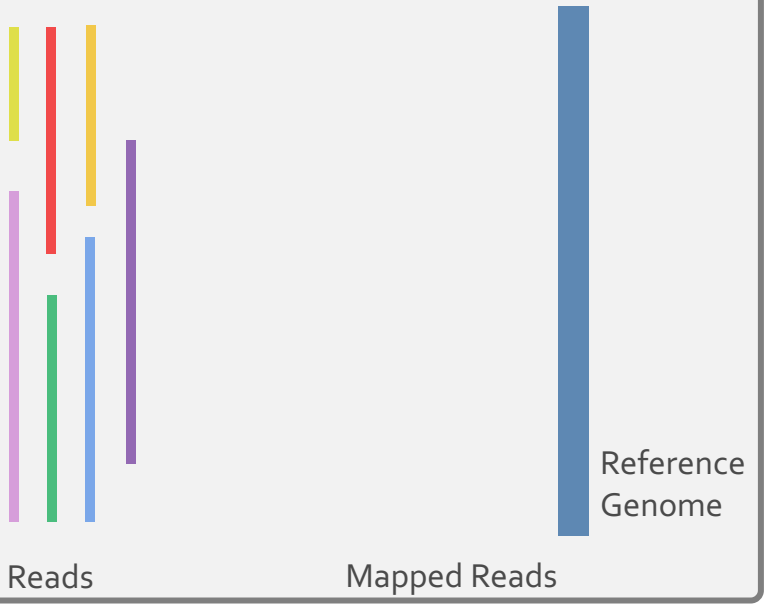
Short reads: **a few hundred base pairs** and **error rate of ~0.1%**

Long reads: **thousands to millions of base pairs** and **error rate of 5–10%**

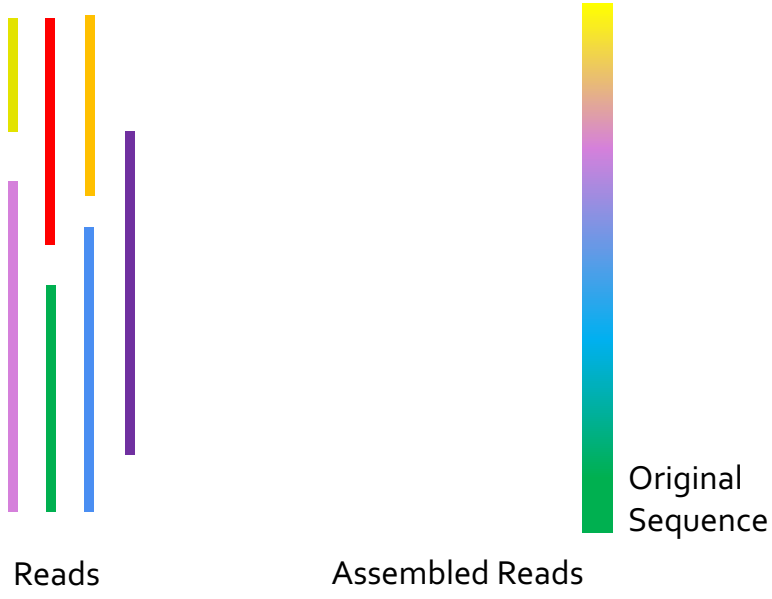
Genome Sequence Analysis



Read Mapping, method of aligning the reads against the reference genome in order to **detect matches and variations.**



***De novo* Assembly,** method of merging the reads in order to **construct** the original sequence.



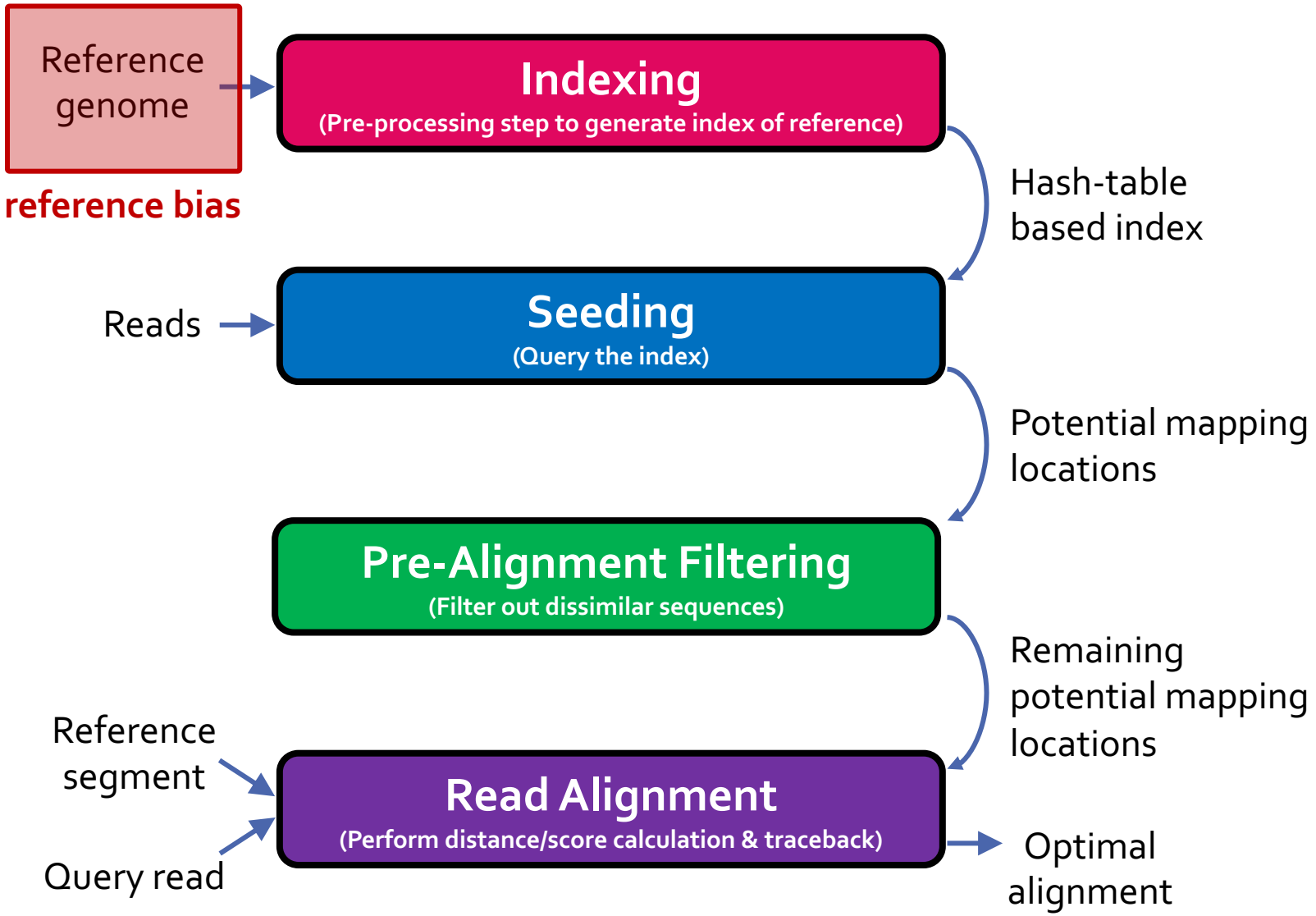
GSA with Read Mapping

- **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

Read Mapping Pipeline



Genome Graphs

Genome graphs:

- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACGTACGT

Genome Graphs

Genome graphs:

- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACG**T**ACGT

Reference #2: ACG**G**ACGT

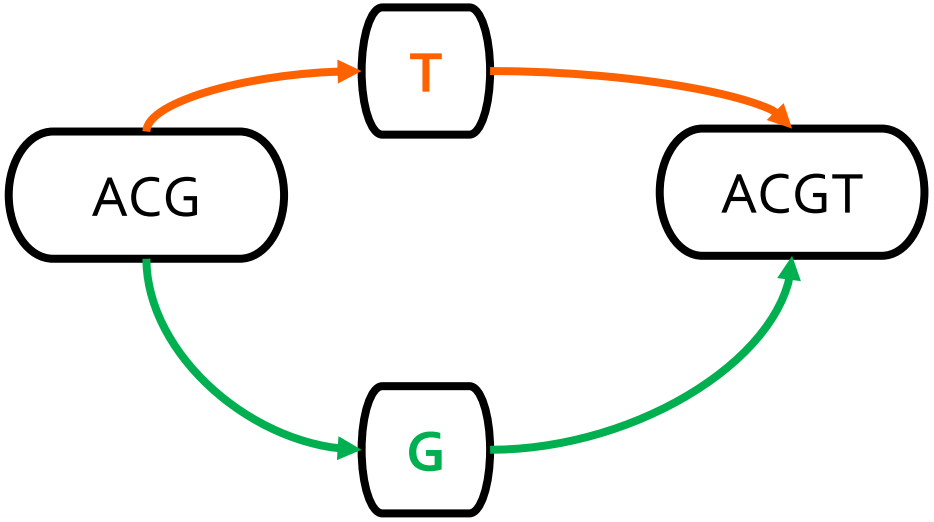
Genome Graphs

Genome graphs:

- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACG**T**ACGT

Reference #2: ACG**G**ACGT



Genome Graphs

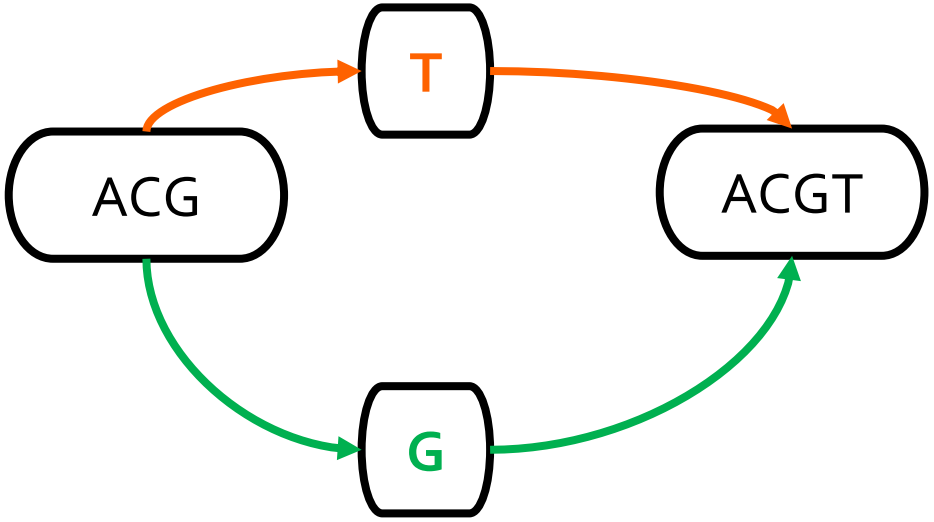
Genome graphs:

- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACG**T**ACGT

Reference #2: ACG**G**ACGT

Reference #3: ACG**TT**ACGT



Genome Graphs

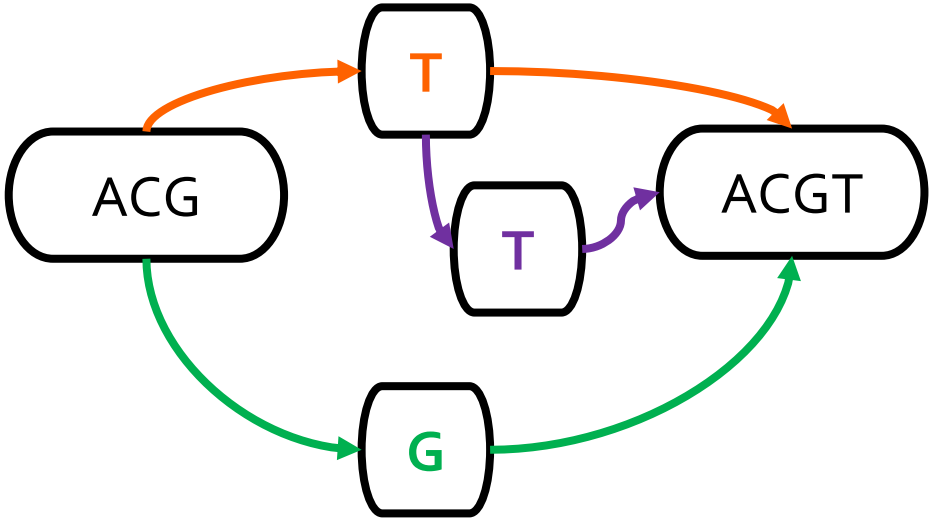
Genome graphs:

- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACGTACGT

Reference #2: ACGGACGT

Reference #3: ACGTTACGT



Genome Graphs

Genome graphs:

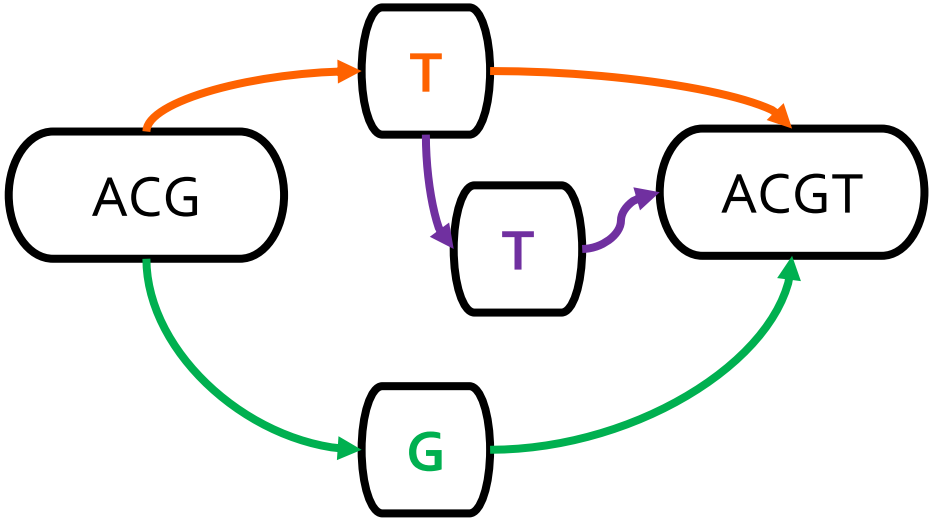
- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACGTACGT

Reference #2: ACGGACGT

Reference #3: ACGTTACGT

Reference #4: ACGACGT



Genome Graphs

Genome graphs:

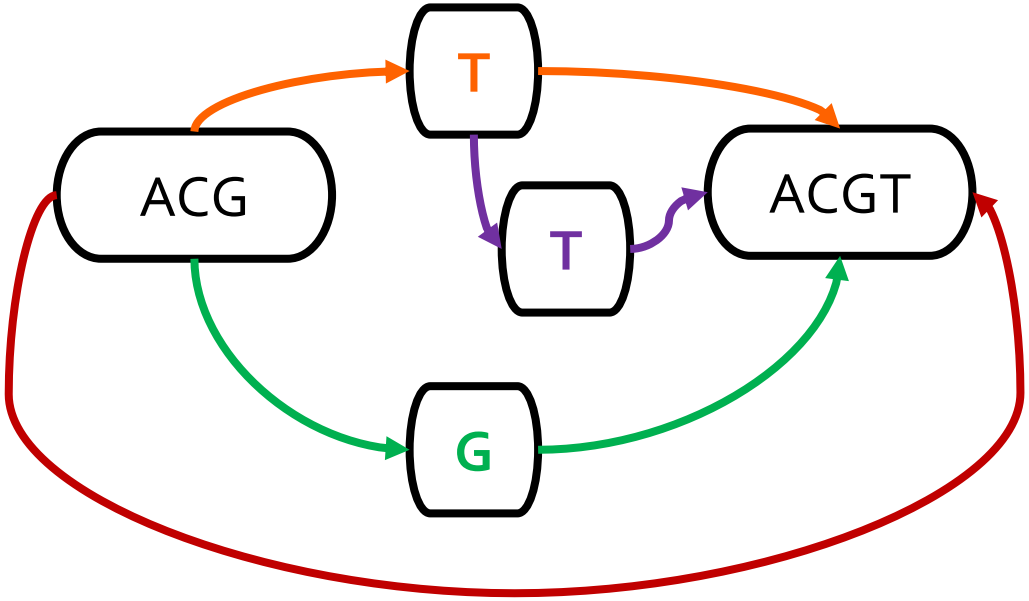
- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACGTACGT

Reference #2: ACGGACGT

Reference #3: ACGTTACGT

Reference #4: ACGACGT



Problem & Motivation

- Traditional read mapping causes reference bias
- Aligning sequences to graphs is a newer field and only a few software tools exist for graph-based GSA
- Graph-based analysis exacerbates mapping's bottlenecks
- Hardware acceleration of sequence-to-graph mapping: important but unexplored research problem

GenGraph: First Graph Mapping Accelerator

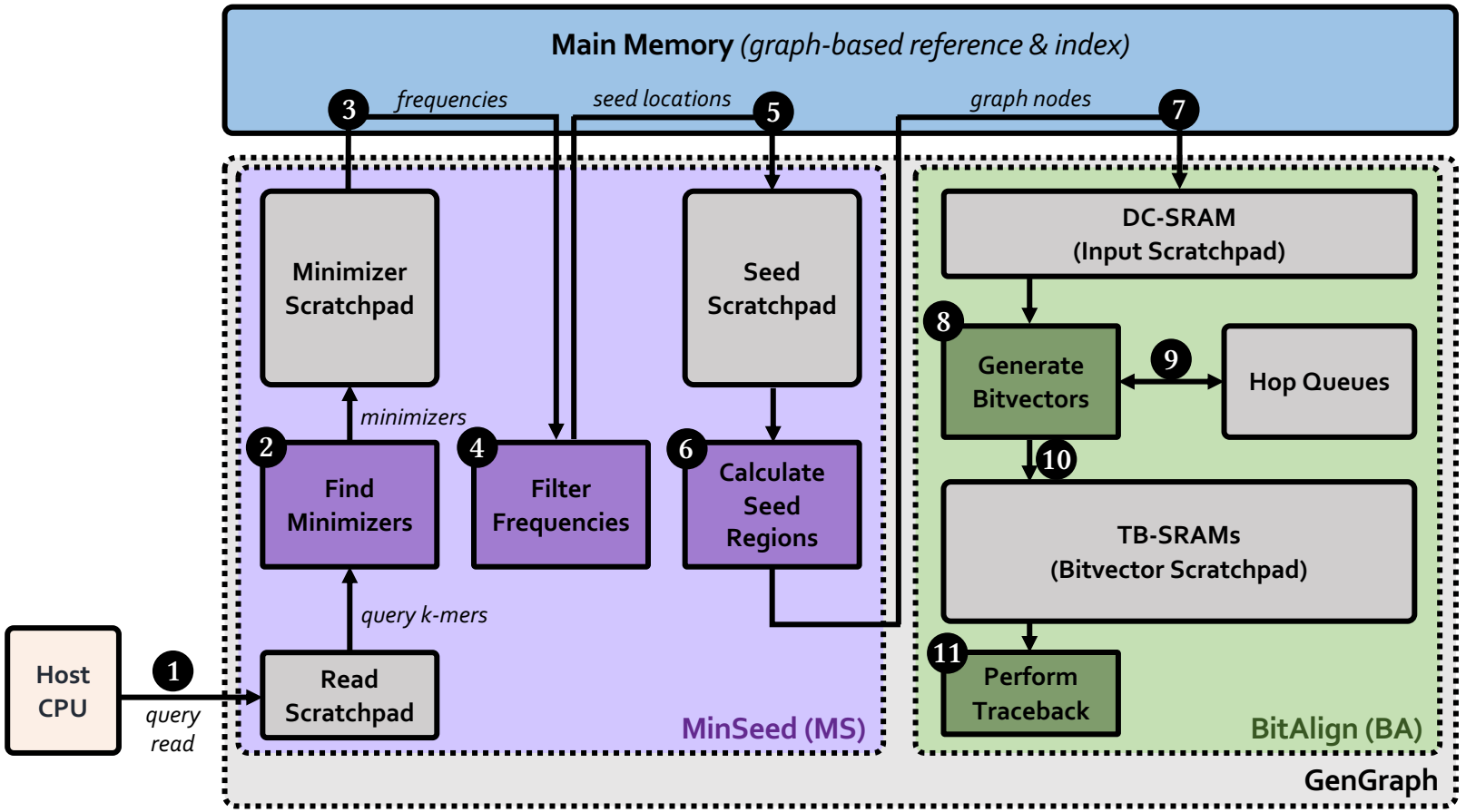
Our Goal:

Design high-performance, scalable, power- and area-efficient hardware accelerators that alleviate bottlenecks in both the seeding and alignment steps of sequence-to-graph mapping with support for both short and long reads

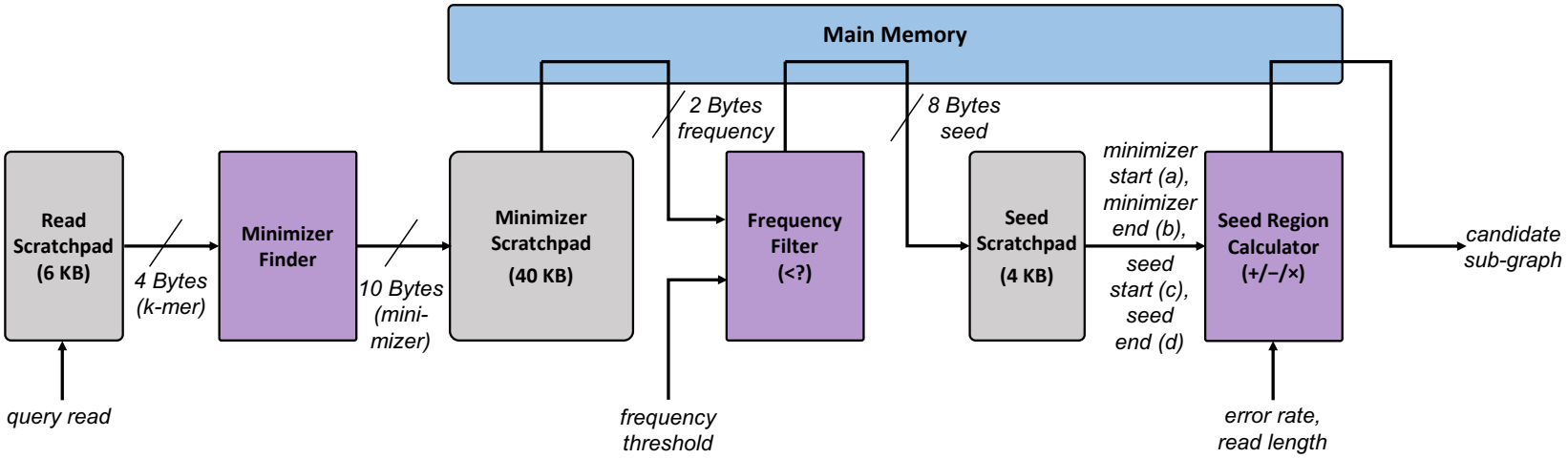
GenGraph:

- *MinSeed:* The *first* minimizer-based seeding hardware
- *BitAlign:* The *first* sequence-to-graph alignment hardware based on modified GenASM algorithms and accelerators

Overview of GenGraph



MinSeed HW



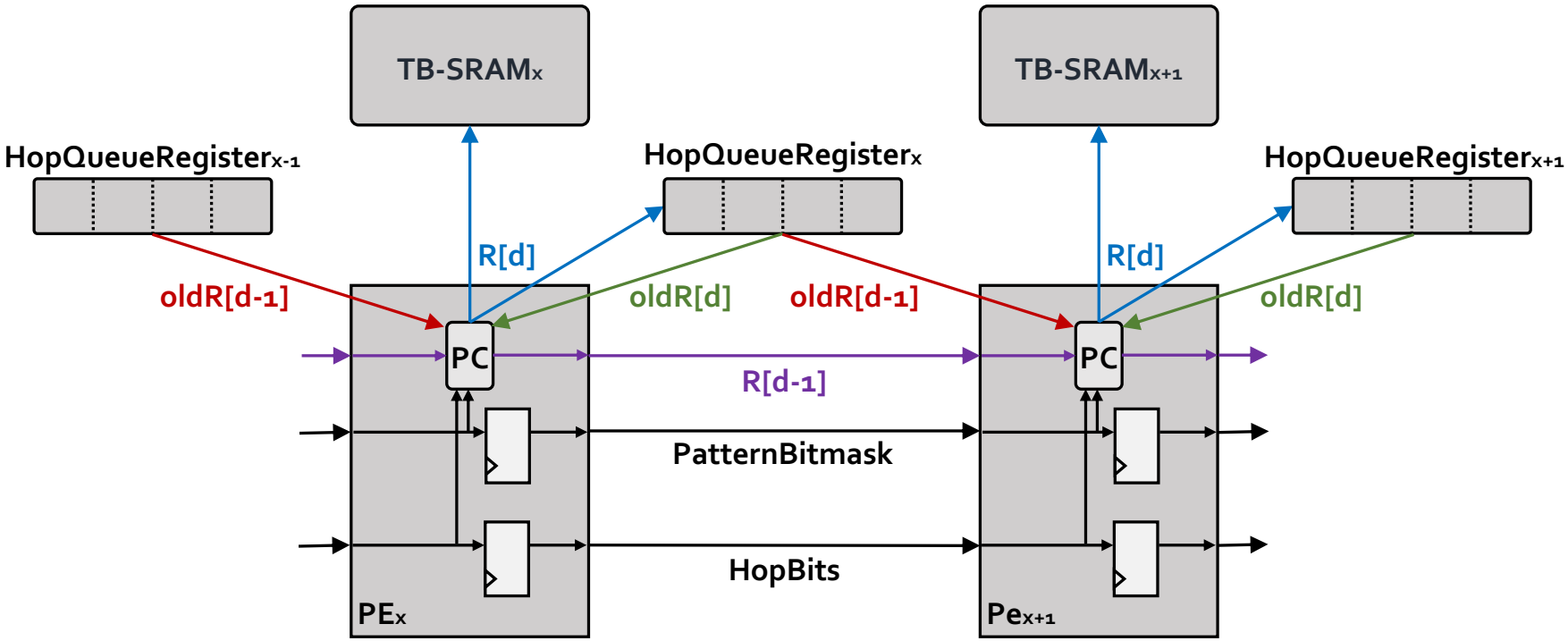
(1) **Three computation modules** responsible for finding the minimizers, filtering the frequencies of minimizers, and finding the associated regions of every seed location

(2) **Three scratchpads** for storing the query read, its minimizers, and seed locations

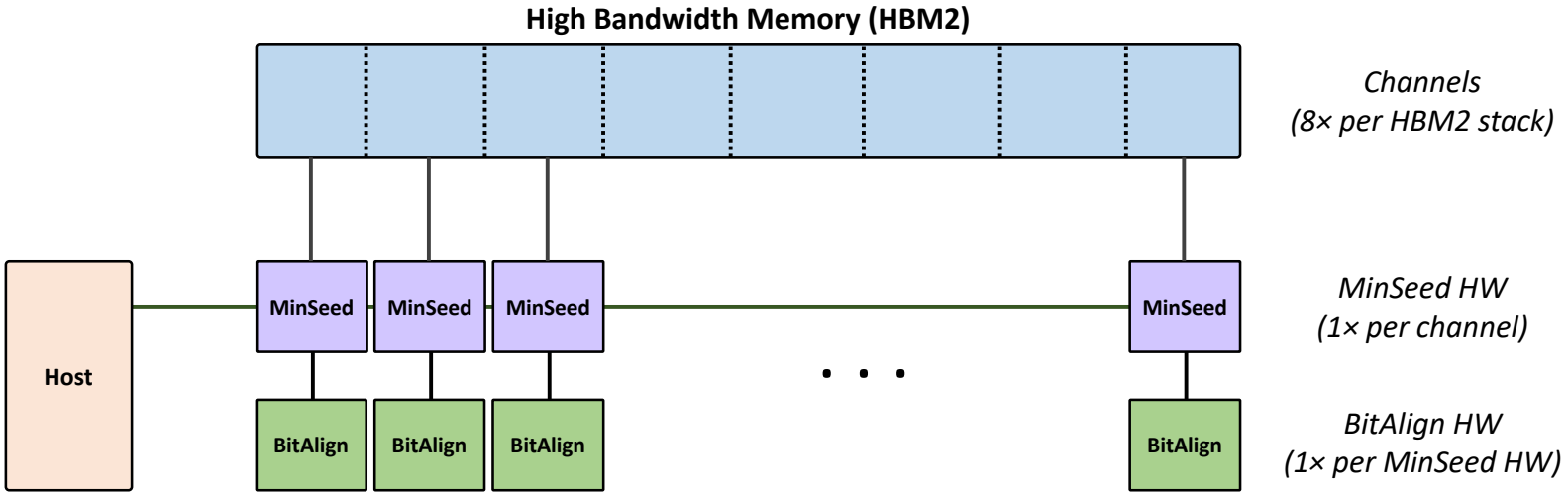
(3) **The memory interface**, which handles the frequency, seed location, and subgraph accesses

BitAlign HW

- **Linear cyclic systolic array-based accelerator**
- ***Hop queue registers*** to incorporate the hops by feeding the bitvectors of non-neighbor characters/nodes



Overall System of GenGraph



- A single GenGraph consists of **8 MinSeed modules** that exploit data-level parallelism when performing seeding
- Each MinSeed module has **exclusive access to one HBM2E channel**
- Each MinSeed module is connected to **a single BitAlign module**
- We **hide the latency of MinSeed** when performing seeding while running sequence-to-graph alignment with BitAlign

Use Cases of GenGraph

(1) End-to-End Sequence-to-Graph Mapping

- The whole GenGraph design (MinSeed + BitAlign) should be executed
- We support both short and long reads

(2) Sequence-to-Graph Alignment

- BitAlign can be executed by itself without the need of an initial seeding tool/accelerator
- BitAlign can also be used for sequence-to-sequence alignment since it is a special and simpler variant of sequence-to-graph alignment

(3) Seeding

- MinSeed only can be used as the seeding module for both graph-based mapping and linear traditional mapping
- MinSeed is orthogonal to be coupled with any alignment tool or accelerator

Evaluation Methodology

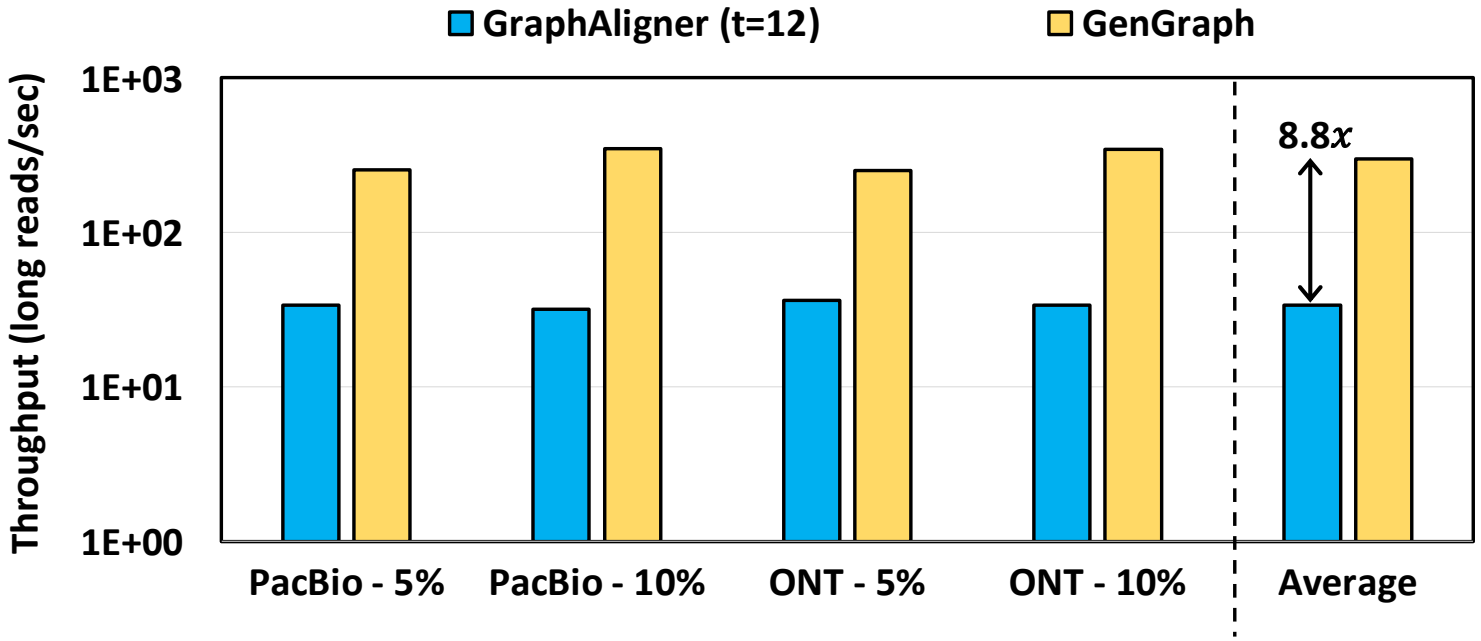
- We evaluate GenGraph using:
  - **Synthesized** SystemVerilog models of the MinSeed and BitAlign accelerator datapaths
  - **Simulation-** and **spreadsheet-based** performance modeling
- **4 x 24GB HBM2E stacks, each with 8 independent channels**
  - 1 MinSeed and 1 BitAlign HW per each channel **(32 in total)**
- Baseline tools:
  - **GraphAligner** and **vg** for sequence-to-graph mapping
  - **PaSGAL** for sequence-to-graph alignment
  - **Darwin**, **GenAx**, and **GenASM** for sequence-to-sequence alignment
- **Simulated datasets** for both short and long reads

Key Results – Area & Power

- Based on our **synthesis** of **MinSeed** and **BitAlign** accelerator datapaths using the Synopsys Design Compiler with a **28nm** process (**@ 1GHz):**

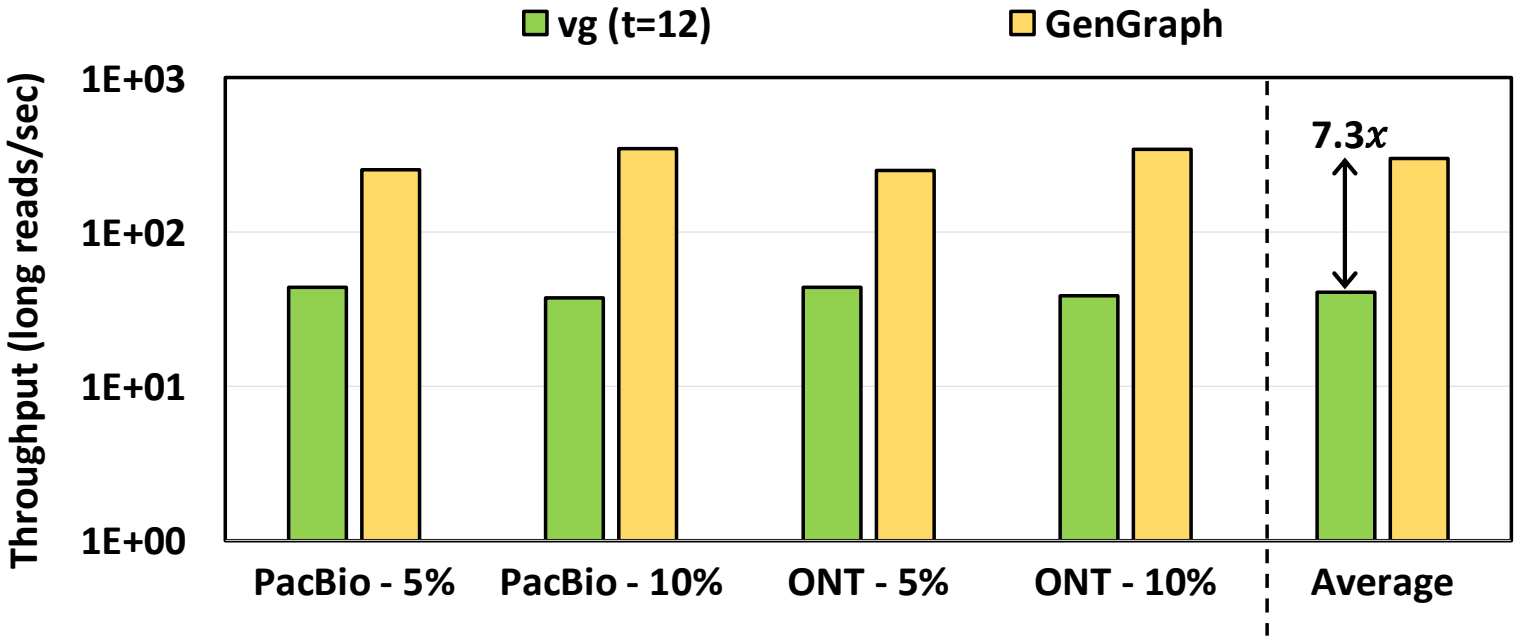
Component	Area (mm^2)	Power (mW)
MinSeed – Logic	0.017	10.8
Read Scratchpad (6 KB)	0.009	1.9
Minimizer Scratchpad (40 KB)	0.061	6.9
Seed Scratchpad (4 KB)	0.006	2.5
BitAlign – DC Logic with HopQueueRegisters (64 PEs)	0.393	378.0
BitAlign – TB Logic	0.020	2.7
Input Scratchpad (DC-SRAM; 24 KB)	0.034	8.4
Bitvector Scratchpad (TB-SRAMs; 128 KB)	0.233	115.1
Total – 1 x GenGraph	**0.773**	**526.3 (0.5 W)**
Total – 8 x GenGraph	**6.184**	**4210.4 (4.2 W)**
Total – 32 x GenGraph	**24.736**	**16841.6 (16.8 W)**

Key Results – GenGraph with Long Reads (I)



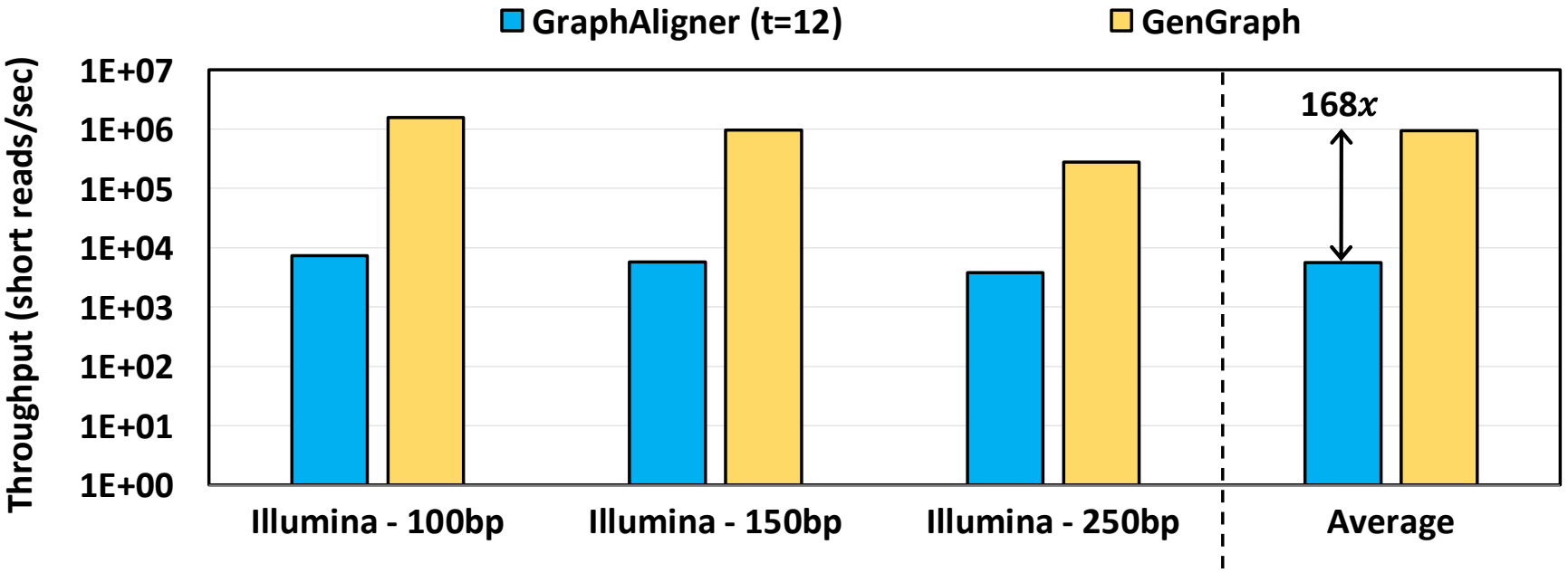
GenGraph provides **8.8× throughput improvement** over GraphAligner's 12-thread execution, while **reducing the power consumption by 4.9×**

Key Results – GenGraph with Long Reads (II)



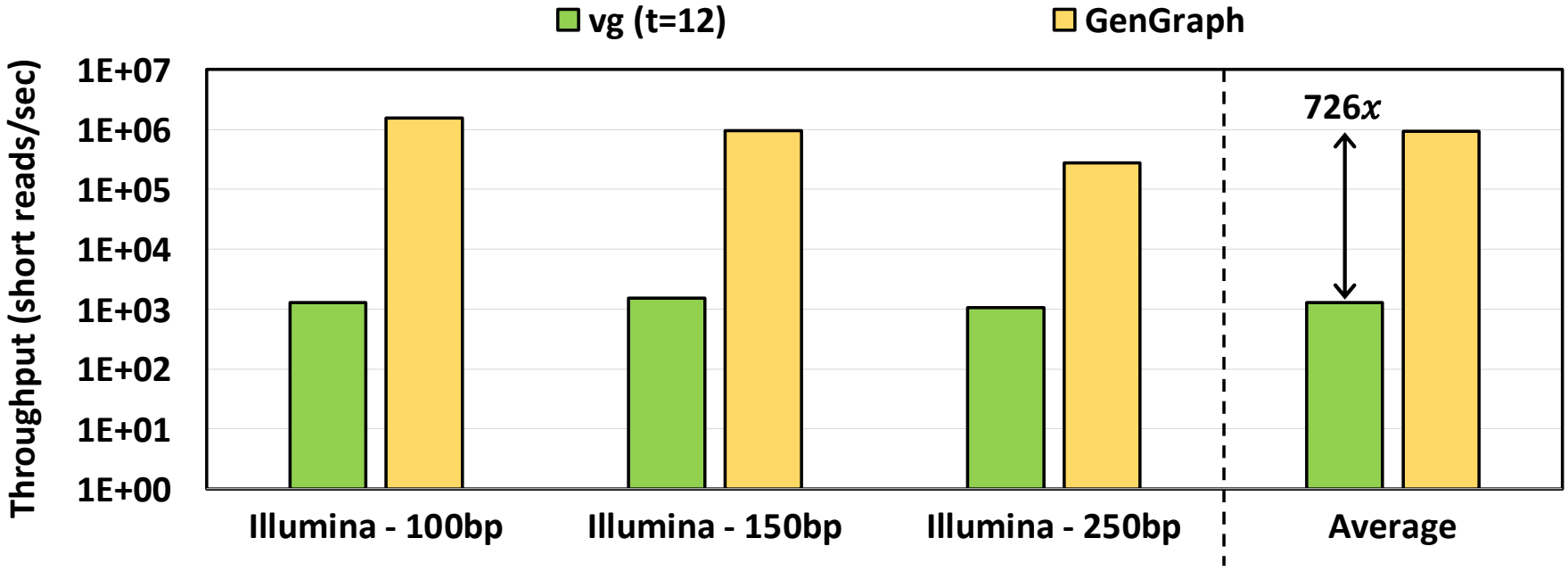
GenGraph provides **7.3× throughput improvement** over vg's 12-thread execution, while **reducing the power consumption by 6.5×**

Key Results – GenGraph with Short Reads (I)



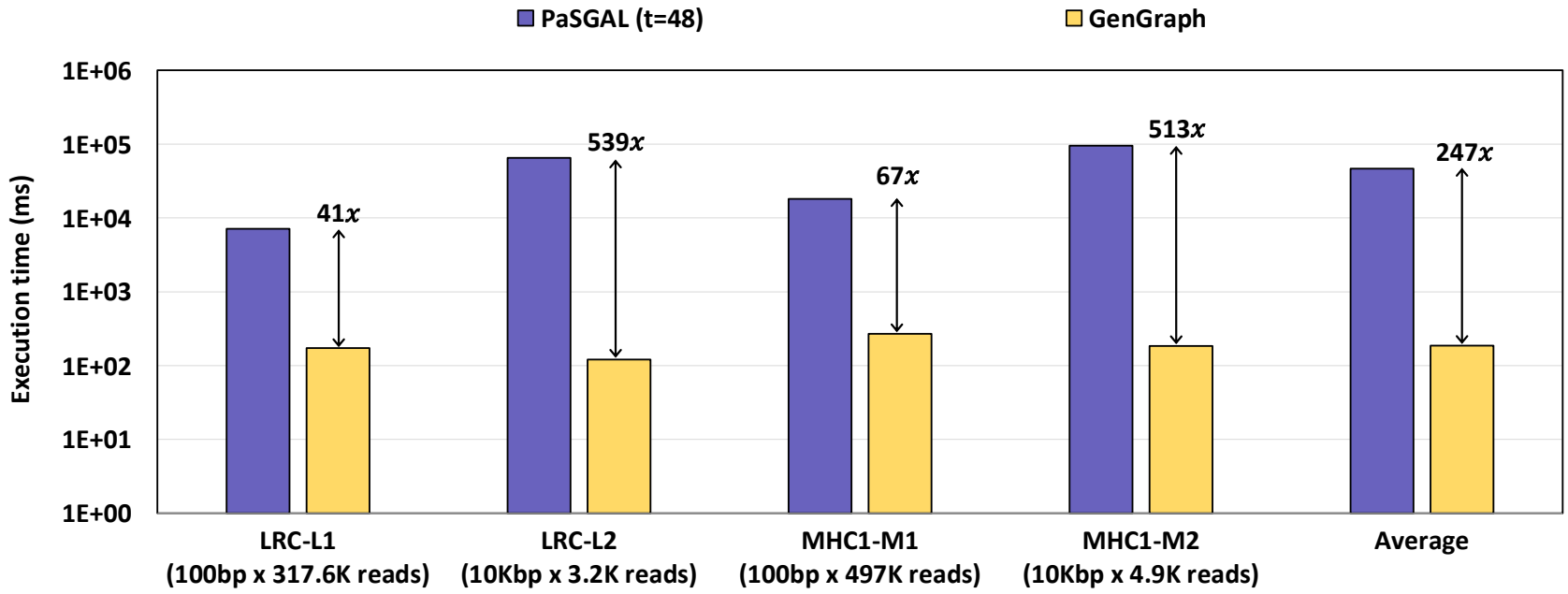
GenGraph provides **168× throughput improvement** over GraphAligner's 12-thread execution, while **reducing the power consumption by 4.7×**

Key Results – GenGraph with Short Reads (II)



GenGraph provides **726× throughput improvement** over vg's 12-thread execution, while **reducing the power consumption by 4.9×**

Key Results – BitAlign (Graph Alignment)



BitAlign provides **41×-539× speedup** over the 48-thread AVX512-supported execution of PaSGAL

Key Results – BitAlign (Linear Alignment)

- BitAlign can be used for both sequence-to-sequence alignment and sequence-to-graph alignment
  - The cost of more functionality: Extra hop queue registers in BitAlign
  - However, we do *not* sacrifice any performance
- **For long reads (over GACT of Darwin and GenASM):**
  - 4.8× and 1.2× throughput improvement,
  - 1.9× and 5.2× higher power consumption, and
  - 1.4× and 2.3× higher area overhead
- **For short reads (over SillaX of GenAx and GenASM):**
  - 2.4× and 1.3× throughput improvement

Conclusion

Problem:

- Traditional read mapping causes reference bias
- Aligning sequences to graphs is a newer field and only a few software tools exist for graph-based GSA
- Graph-based analysis exacerbates mapping's bottlenecks
- Hardware acceleration of sequence-to-graph mapping: important but unexplored research problem

Key Contributions:

- GenGraph: *First* acceleration framework for sequence-to-graph mapping
  - MinSeed: *First* minimizer-based seeding accelerator
  - BitAlign: *First* sequence-to-graph alignment accelerator based upon our new bitvector-based, highly-parallel algorithm

Key Results: GenGraph and BitAlign provide significant speedups compared to the software baselines, while reducing the power consumption

GenGraph: A Hardware Acceleration Framework for Sequence-to-Graph Mapping

Damla Senol Cali *et al.*

https://damlasenolcali.github.io

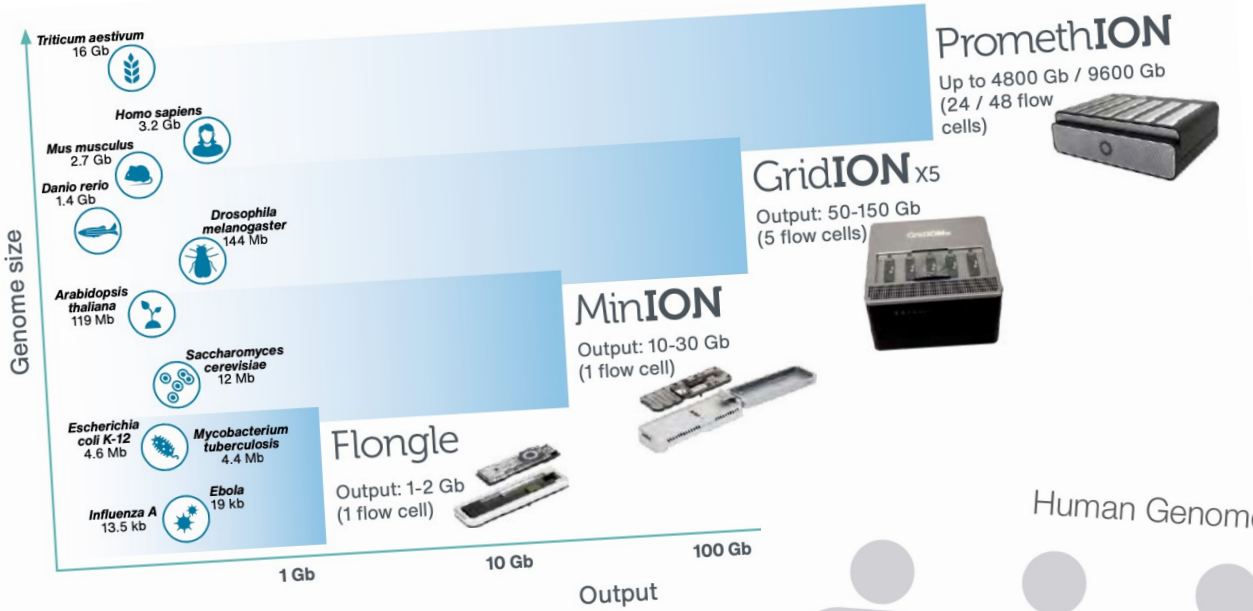
TECHCON'21 – September 14, 2021

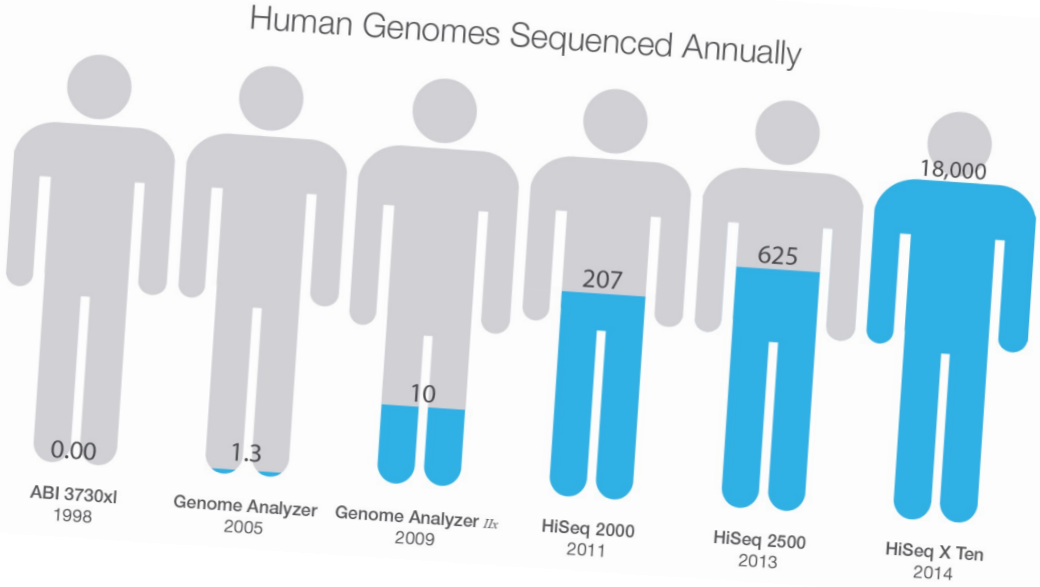
This work was supported in part by Semiconductor Research Corporation (SRC).

Backup Slides

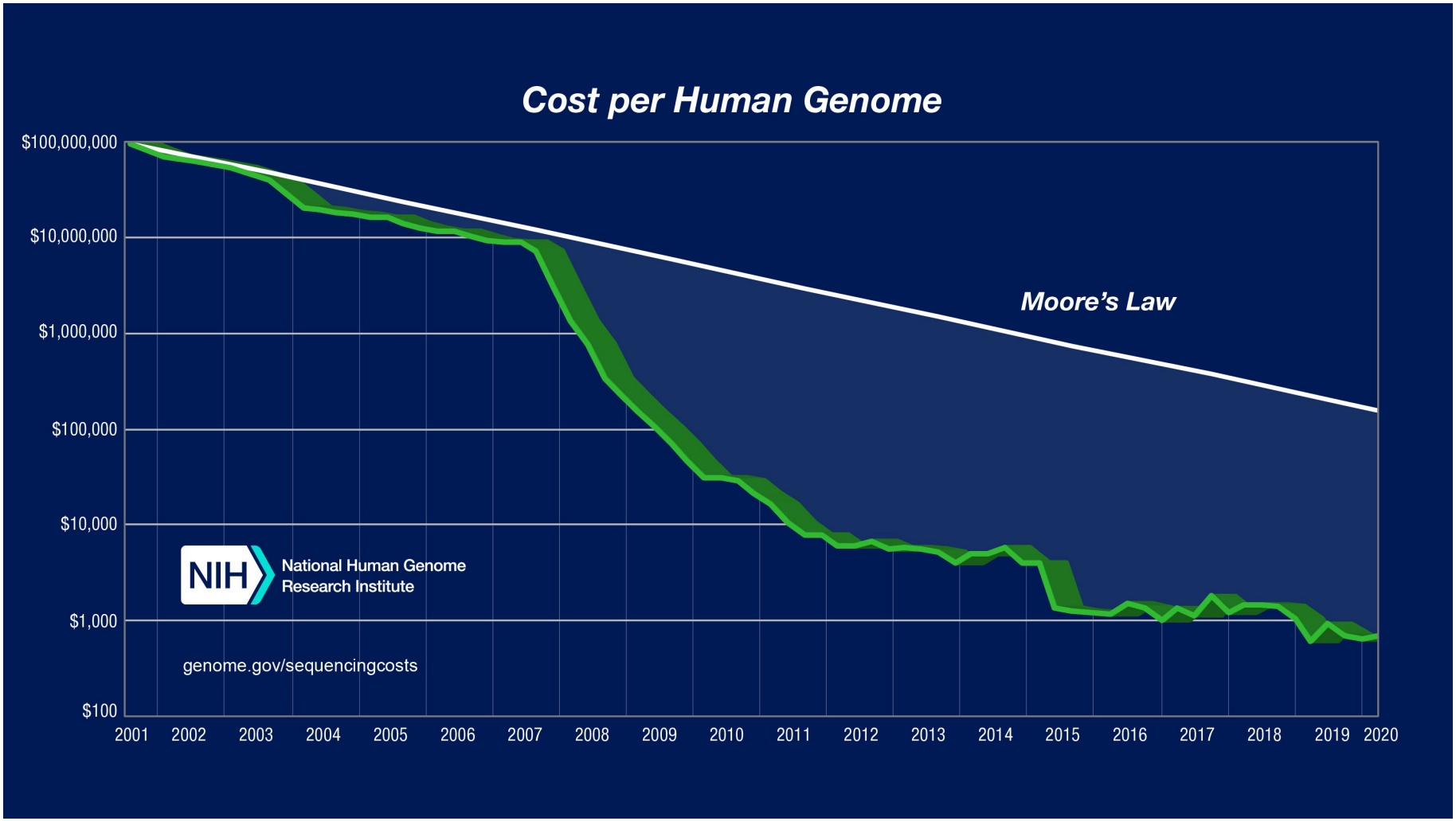
(Sequencing)

Current State of Sequencing



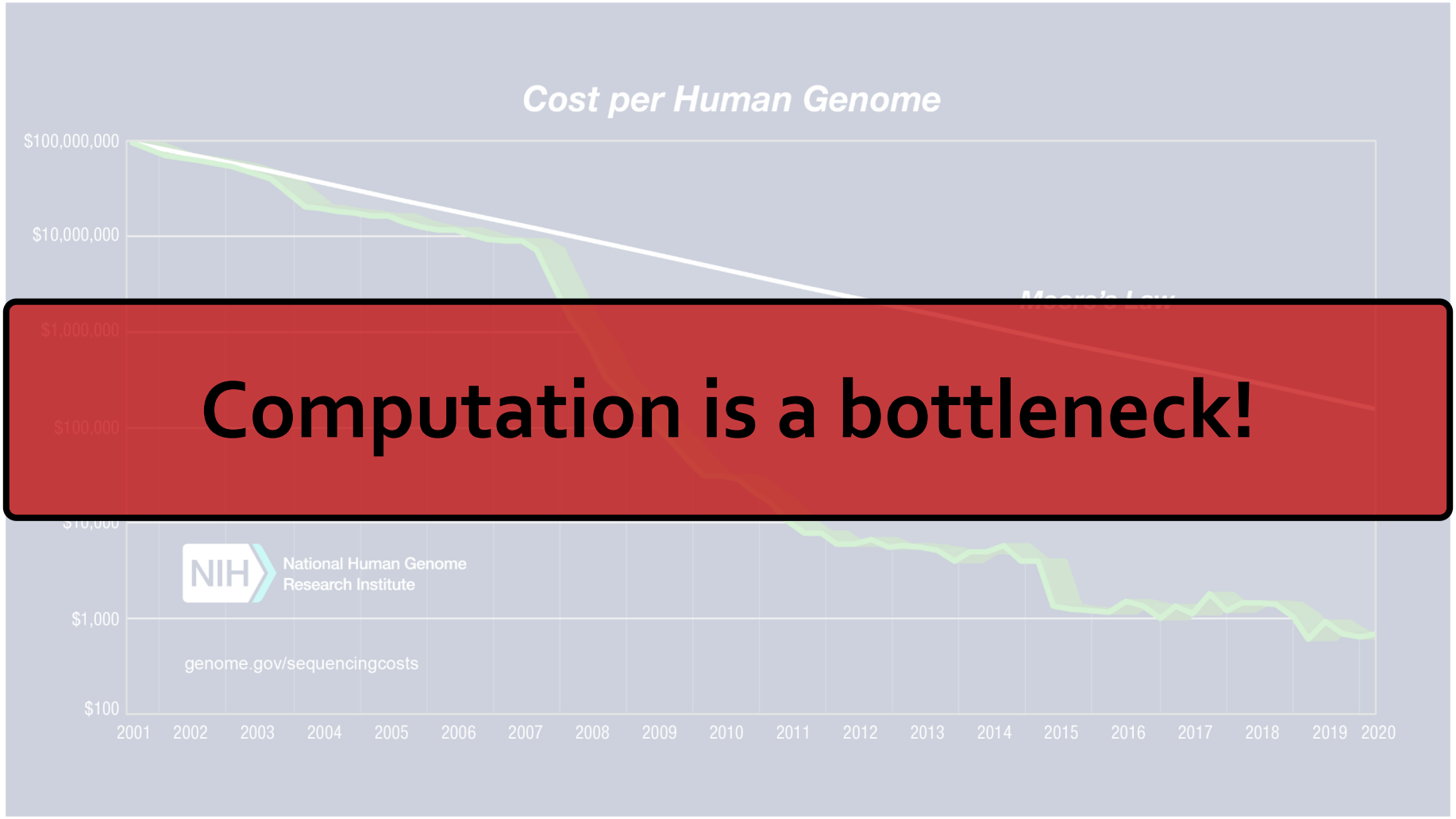


Current State of Sequencing (cont'd.)



*From NIH (https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data)

Current State of Sequencing (cont'd.)

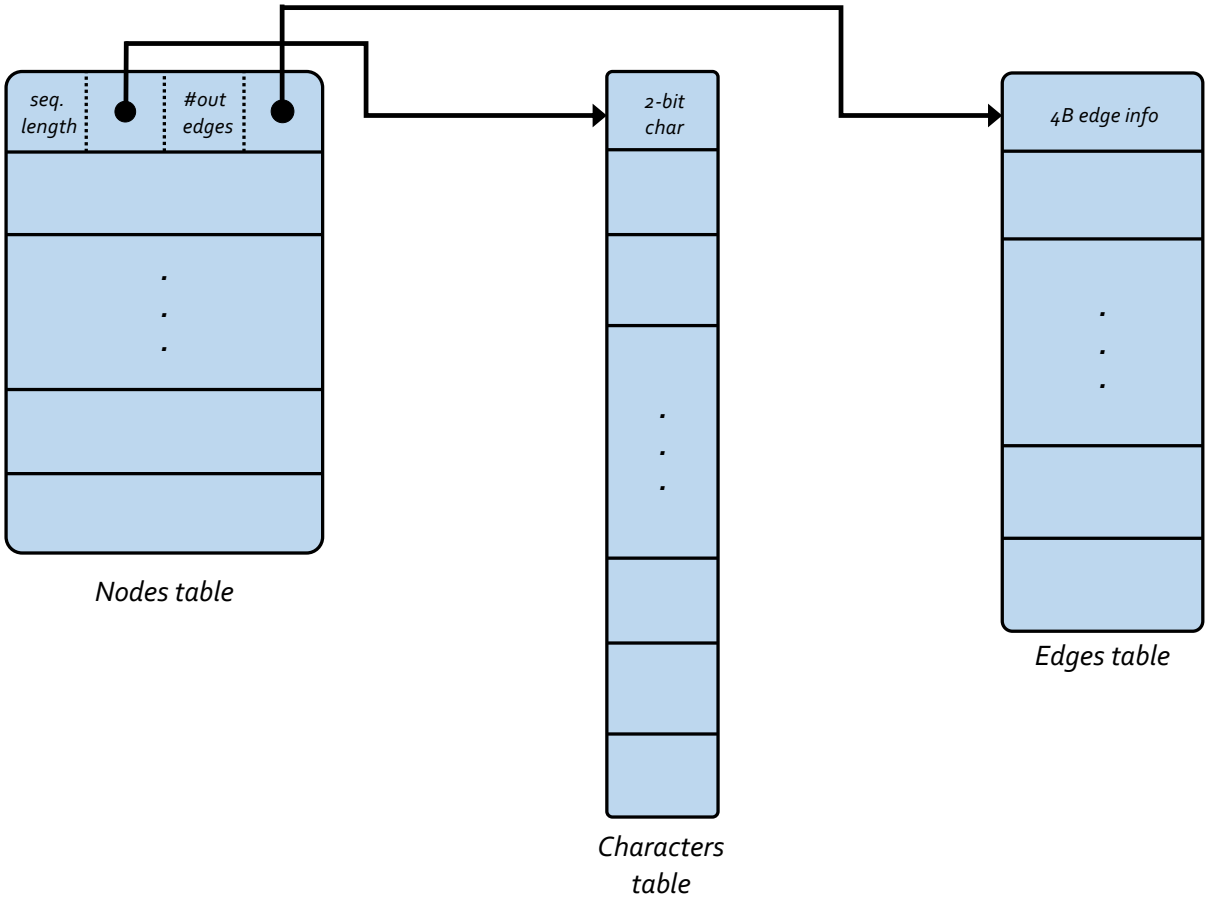


Computation is a bottleneck!

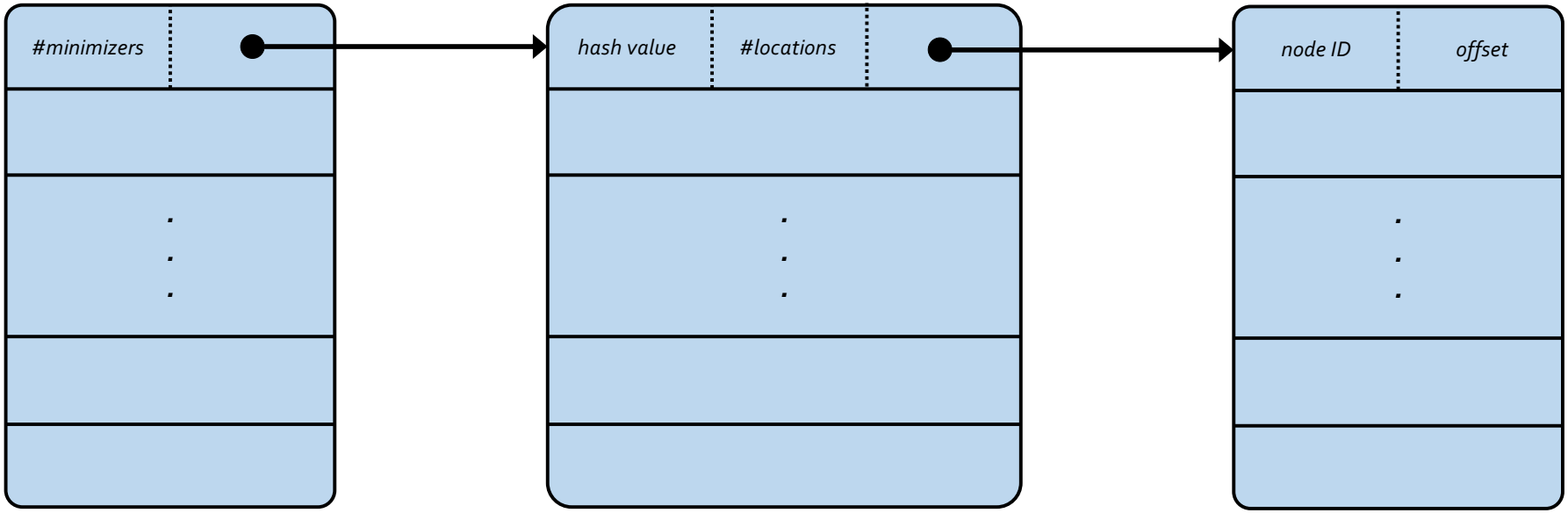
*From NIH (https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data)

Backup Slides
(GenGraph)

GenGraph – Graph Structure



GenGraph – Index Structure



First-level: Buckets

Second-level: Minimizers

Third-level: Locations

Minimizers

Position	1	2	3	4	5	6	7
Sequence	A	G	T	A	G	C	A
Full set of k-mers with minimizer in **red**	A	G	T				
		G	T	A			
			T	A	G		
				A	**G**	**C**	
					G	C	A

BitAlign Algorithm

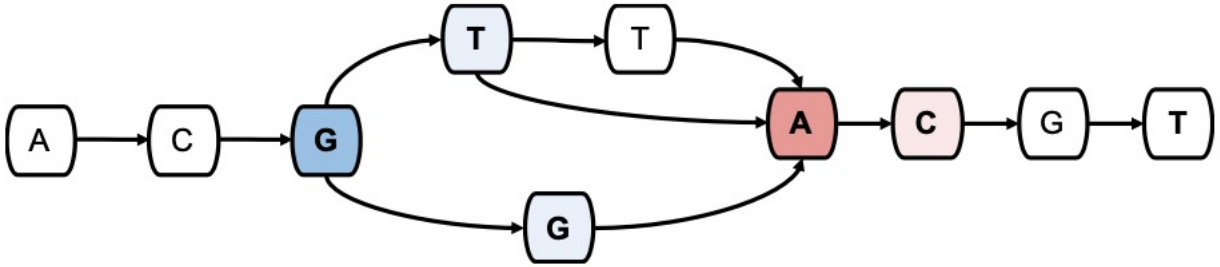
Algorithm 1 BitAlign Algorithm

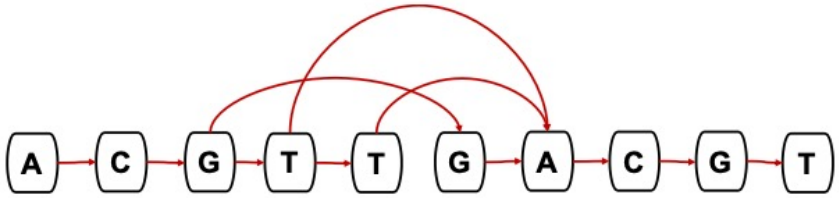
Inputs: `graph-nodes` (reference), `pattern` (query), `k` (edit distance threshold)

Outputs: `editDist` (minimum edit distance), `CIGARstr` (traceback output)

```
1: n ← length of linearized reference subgraph
2: m ← length of query pattern
3: PM ←genPatternBitmasks(pattern)   ▷ pre-process the pattern
4:
5: allR[n][d] ← 111.111      ▷ init R[d] bitvectors for all characters
6:
7: for i in (n-1):-1:0 do          ▷ iterate over each graph node
8:    curChar ← graph-nodes[i].char
9:    curPM ← PM[curChar]          ▷ retrieve the pattern bitmask
10:
11:   R0 ← 111...111             ▷ status bitvector for exact match
12:   for j in graph-nodes[i].successors do
13:      R0 ← ((R[j][0]<<1) | curPM) & R0
14:   allR[i][0] ← R0
15:
16:   for d in 1:k  do
17:      I ← (allR[i][d-1]<<1)                          ▷ insertion
18:      Rd ← I                        ▷ status bitvector for d errors
19:      for j in graph-nodes[i].successors do
20:         D ← allR[j][d-1]                            ▷ deletion
21:         S ← allR[j][d-1]<<1                      ▷ substitution
22:         M ← (allR[j][d]<<1) | curPM                 ▷ match
23:         Rd ← D & S & M & Rd
24:      allR[i][d] ← Rd
25: <editDist, CIGAR> ← traceback(allR, graph-nodes,
26: pattern)
```

GenGraph – Hops



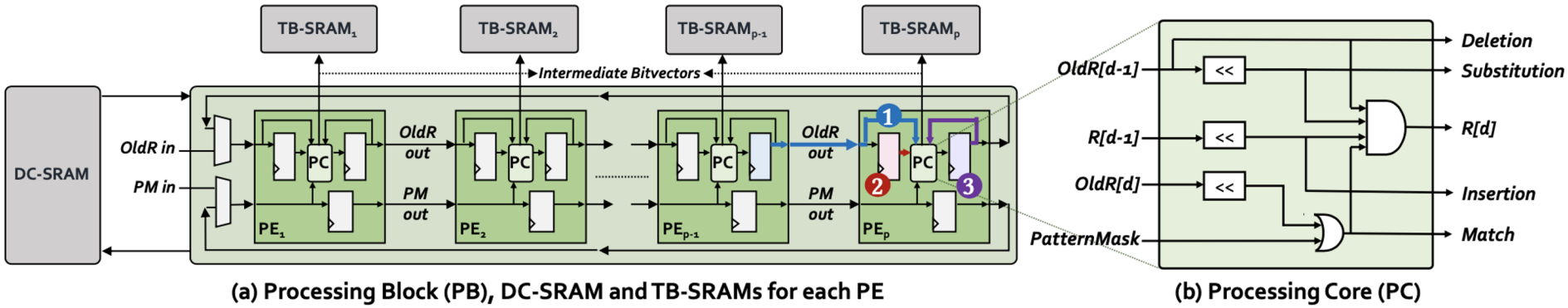


NodeID	1	2	3	4	5	6	7	8	9	10	
HopBits	0	0	0	0	0	0	0	0	0	0	1
	1	0	0	0	0	0	0	0	0	0	2
	0	1	0	0	0	0	0	0	0	0	3
	0	0	1	0	0	0	0	0	0	0	4
	0	0	0	1	0	0	0	0	0	0	5
	0	0	1	0	0	0	0	0	0	0	6
	0	0	0	1	1	1	0	0	0	0	7
	0	0	0	0	0	0	1	0	0	0	8
	0	0	0	0	0	0	0	1	0	0	9
	0	0	0	0	0	0	0	0	1	0	10

Recall: GenASM-DC's HW Design

Cycle#	Thread$_1$ *R0/4*	Thread$_2$ *R1/5*	Thread$_3$ *R2/6*	Thread$_4$ *R3/7*
#1	T0-R0	–	–	–
#2	T1-R0	T0-R1	–	–
#3	T2-R0	T1-R1	T0-R2	–
#4	T3-R0	T2-R1	T1-R2	T0-R3
#5	T0-R4	T3-R1	T2-R2	T1-R3
#6	T1-R4	T0-R5	T3-R2	T2-R3
#7	T2-R4	T1-R5	T0-R6	T3-R3
#8	T3-R4	T2-R5	T1-R6	T0-R7
#9	–	T3-R5	T2-R6	T1-R7
#10	–	–	T3-R6	T2-R7
#11	–	–	–	T3-R7

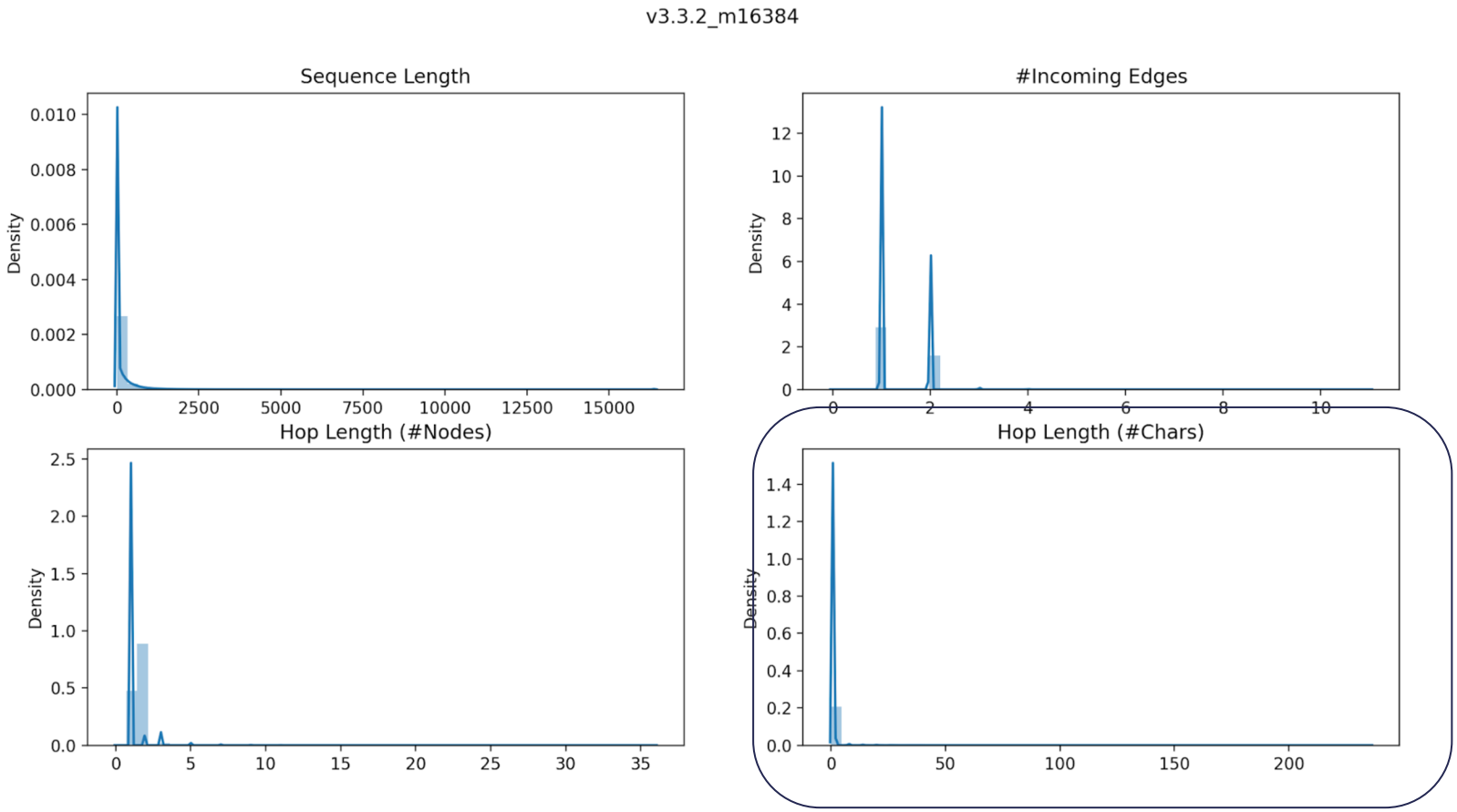
```
deletion (D) ← oldR[d-1]
substitution (S) ← (oldR[d-1]<<1)
insertion (I) ← (R[d-1]<<1)
match (M) ← (oldR[d]<<1) | curPM
```



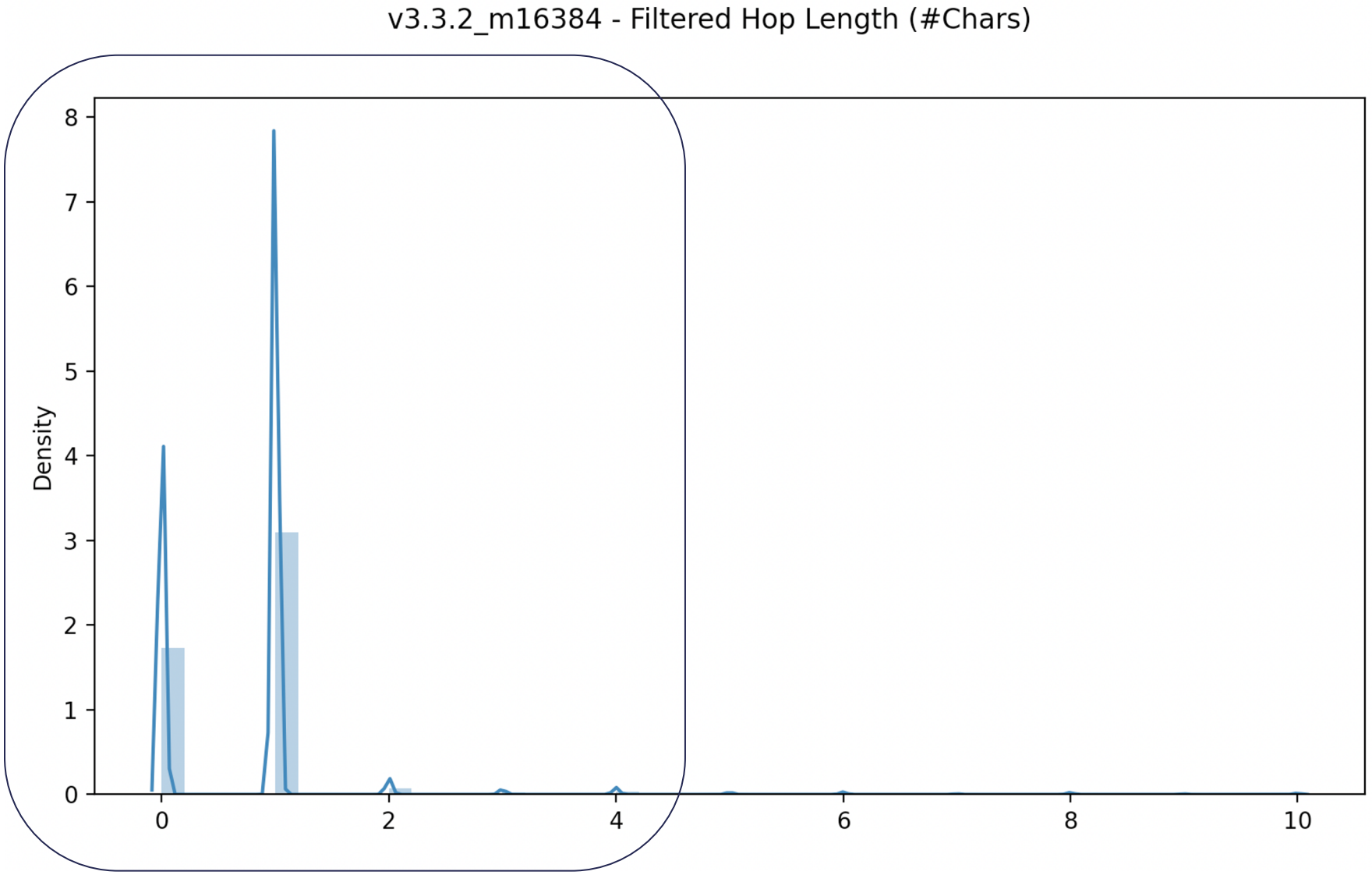
(a) Processing Block (PB), DC-SRAM and TB-SRAMs for each PE

(b) Processing Core (PC)

BitAlign – Hop Length Dist Plots



Hop Length Dist Plots (cont'd.)



DP-based Graph Alignment

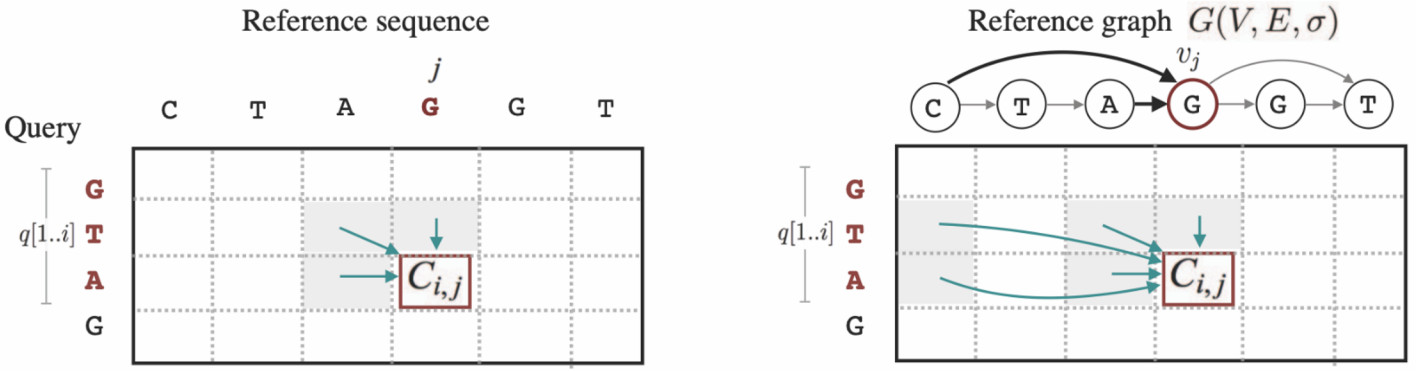


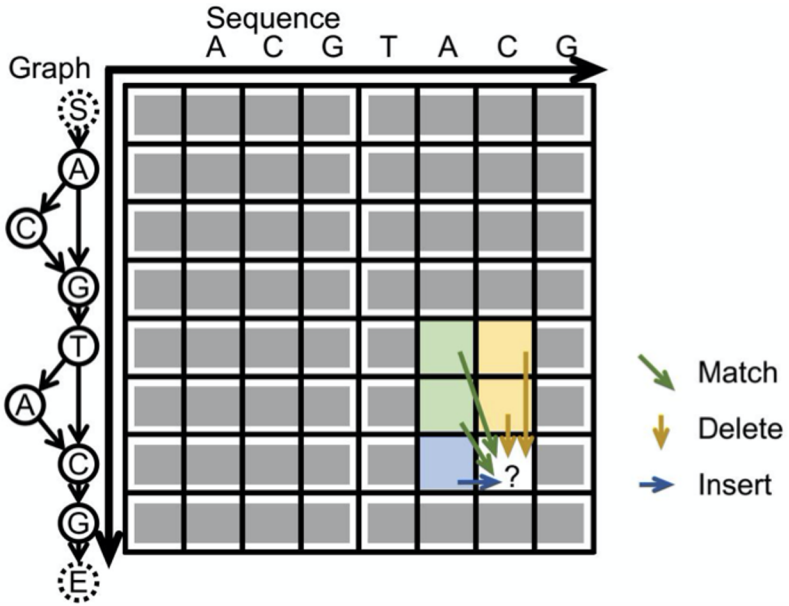
Fig. 2: Example to illustrate difference between Smith-Waterman sequence to sequence alignment and sequence to DAG alignment procedures.

$$C_{0,j} = 0$$

$$C_{i,j} = \max \begin{cases} 0 & \\ \Delta_{i,j} & \\ C_{i-1,k} + \Delta_{i,j} & \forall k : (v_k, v_j) \in E \\ C_{i,k} - \Delta_{ins} & \forall k : (v_k, v_j) \in E \\ C_{i-1,j} - \Delta_{del} & \end{cases} \quad (1)$$

From *PaSGAL paper*

DP-based Graph Alignment (cont'd.)



"abPOA processes all the vectors in a row-by-row manner following the partial order of the graph. During the DP process, for "match" and "delete" operations (diagonal and vertical moves in the DP matrix), all scores stored in each SIMD vector can be updated in parallel as they only rely on scores in the predecessor rows. For "insert" operations (horizontal moves in the DP matrix), sequential non-parallel updating of scores in the same SIMD vector is needed, as the score of each cell depends on the score of the cell on the left."

From *abPOA paper*

Backup Slides
(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*

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

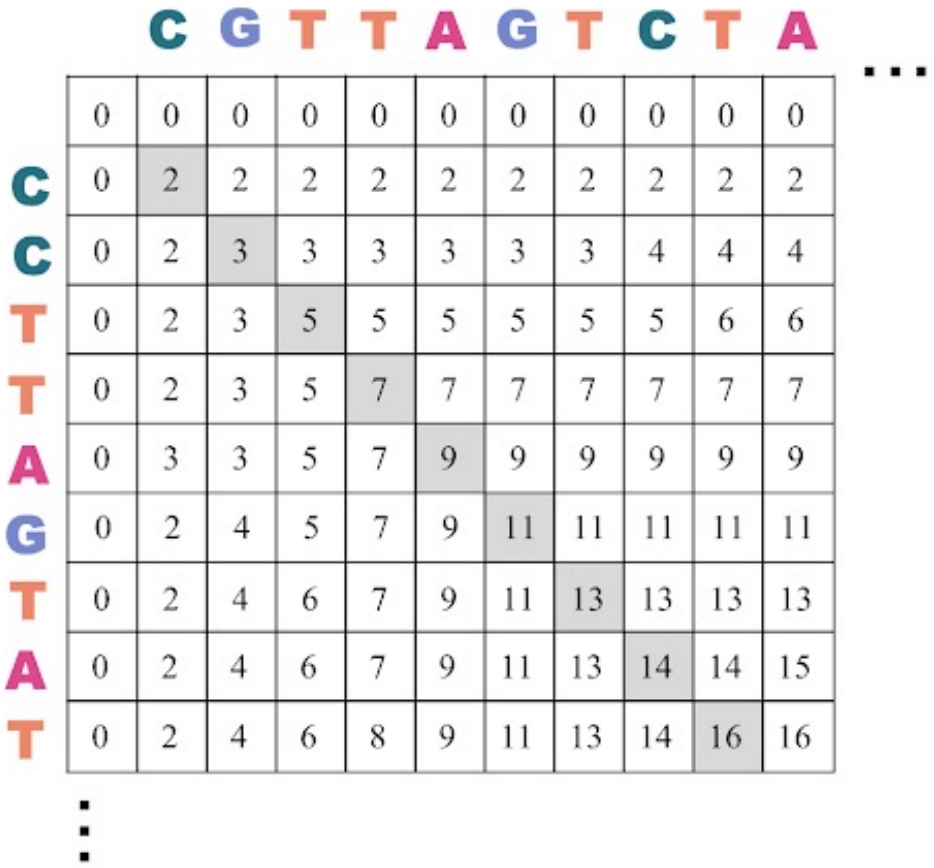
- **GenASM:** *First* ASM acceleration framework for GSA
  - Based upon the *Bitap* algorithm
    - Uses fast and simple bitwise operations to perform ASM
  - Modified and extended ASM algorithm
    - Highly-parallel Bitap with long read support
    - Novel bitvector-based algorithm to perform *traceback*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

Approximate String Matching

- Sequenced genome may not exactly map to the reference genome due to genetic variations and sequencing errors

- **Approximate string matching (ASM):**
  - Detect the differences and similarities between two sequences
  - In genomics, ASM is required to:
    - Find the *minimum edit distance* (i.e., total number of differences)
    - Find the *optimal alignment* with a *traceback* step
      - Sequence of matches, substitutions, insertions and deletions, along with their positions
  - Usually implemented as a dynamic programming (DP) based algorithm

DP-based ASM



		C	G	T	T	A	G	T	C	T	A	...
	0	0	0	0	0	0	0	0	0	0	0	
C	0	2	2	2	2	2	2	2	2	2	2	
C	0	2	3	3	3	3	3	3	4	4	4	
T	0	2	3	5	5	5	5	5	5	6	6	
T	0	2	3	5	7	7	7	7	7	7	7	
A	0	3	3	5	7	9	9	9	9	9	9	
G	0	2	4	5	7	9	11	11	11	11	11	
T	0	2	4	6	7	9	11	13	13	13	13	
A	0	2	4	6	7	9	11	13	14	14	15	
T	0	2	4	6	8	9	11	13	14	16	16	
⋮												

Commonly-used algorithm for ASM in genomics…

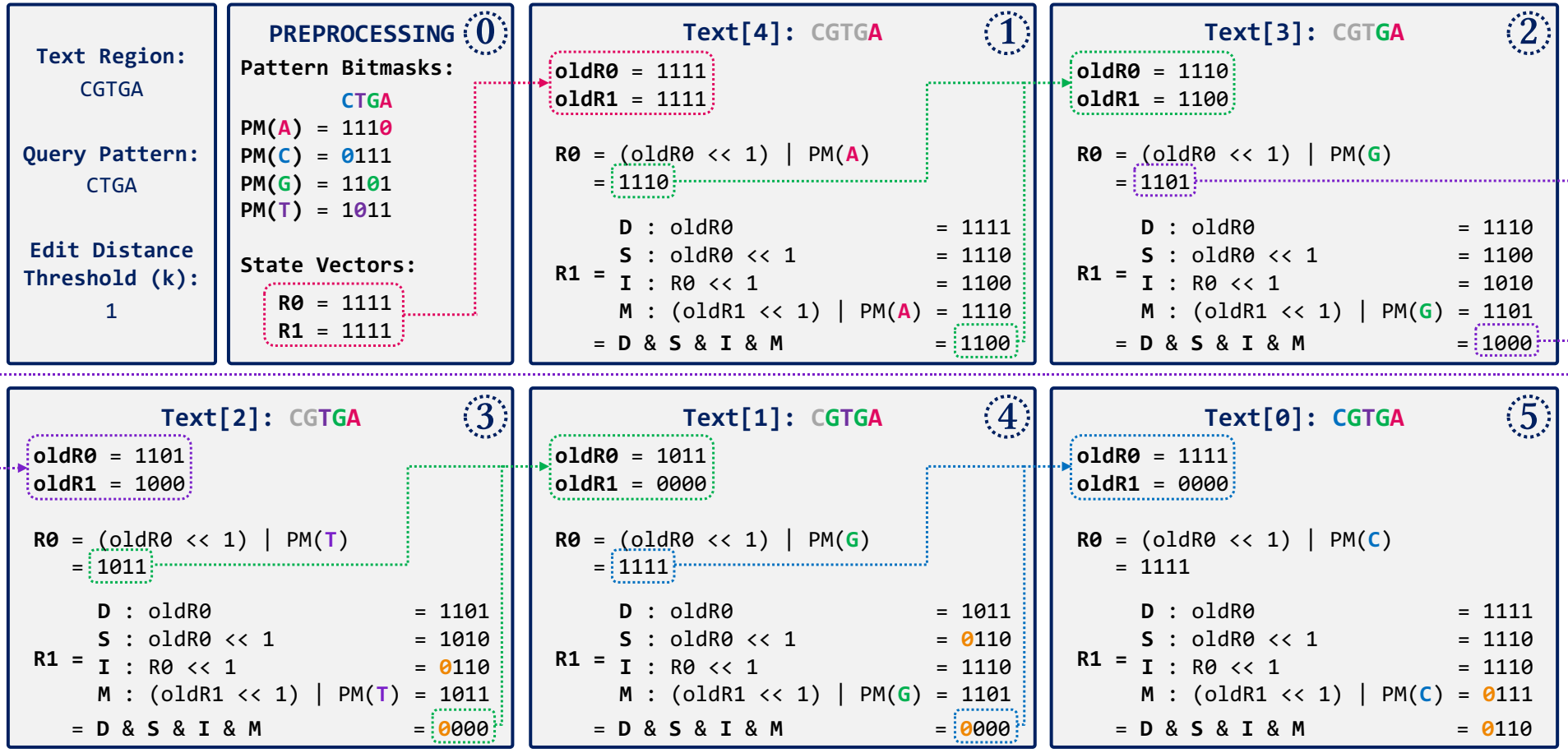
…with quadratic time and space complexity

Bitap Algorithm

- Bitap[1,2] performs ASM with fast and simple bitwise operations
  - Amenable to efficient hardware acceleration
  - Computes the **minimum edit distance** between a **text** (e.g., reference genome) and a **pattern** (e.g., read) with a maximum of ***k*** errors
- **Step 1: Pre-processing (per pattern)**
  - Generate a pattern bitmask (PM) for each character in the alphabet (A, C, G, T)
  - Each PM indicates if character exists at each position of the pattern
- **Step 2: Searching (Edit Distance Calculation)**
  - Compare all characters of the text with the pattern by using:
    - Pattern bitmasks
    - Status bitvectors that hold the partial matches
    - Bitwise operations

[1] R. A. Baeza-Yates and G. H. Gonnet. "A New Approach to Text Searching." *CACM,* 1992.
[2] S. Wu and U. Manber. "Fast Text Searching: Allowing Errors." *CACM,* 1992.

Example for the Bitap Algorithm



Limitations of Bitap

1) **Data Dependency Between Iterations:**
   - Two-level data dependency forces the consecutive iterations to take place sequentially

Bitap Algorithm (cont'd.)

- **Step 2: Edit Distance Calculation**

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.)

- **Step 2: Edit Distance Calculation**

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.

Limitations of Bitap

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

Bitap Algorithm (cont'd.)

- **Step 2: Edit Distance Calculation**

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]

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

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

Check MSB of R[d]:

If 1, no match.

If 0, match with *d* many errors.

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 **SW**
    - 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 **HW**

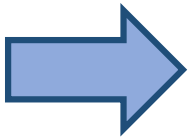
GenASM Algorithm

- **GenASM-DC Algorithm:**
  - Modified Bitap for Distance Calculation
  - Extended for efficient long read support
  - Besides bit-parallelism that Bitap has, extended for parallelism:
    - Loop unrolling
    - Text-level parallelism

- **GenASM-TB Algorithm:**
  - Novel Bitap-compatible TraceBack algorithm
  - Walks through the intermediate bitvectors (match, deletion, substitution, insertion) generated by GenASM-DC
  - Follows a divide-and-conquer approach to decrease the memory footprint

Loop Unrolling in GenASM-DC

Cycle#	Thread$_1$ *R0/1/2/..*
#1	T0-R0
…	…
#8	T0-R7
#9	T1-R0
…	…
#16	T1-R7
#17	T2-R0
…	…
#24	T2-R7
#25	T3-R0
…	…
#32	T3-R7



Cycle#	Thread$_1$ *R0/4*	Thread$_2$ *R1/5*	Thread$_3$ *R2/6*	Thread$_4$ *R3/7*
#1	T0-R0	–	–	–
#2	T1-R0	T0-R1	–	–
#3	T2-R0	T1-R1	T0-R2	–
#4	T3-R0	T2-R1	T1-R2	T0-R3
#5	T0-R4	T3-R1	T2-R2	T1-R3
#6	T1-R4	T0-R5	T3-R2	T2-R3
#7	T2-R4	T1-R5	T0-R6	T3-R3
#8	T3-R4	T2-R5	T1-R6	T0-R7
#9	–	T3-R5	T2-R6	T1-R7
#10	–	–	T3-R6	T2-R7
#11	–	–	–	T3-R7

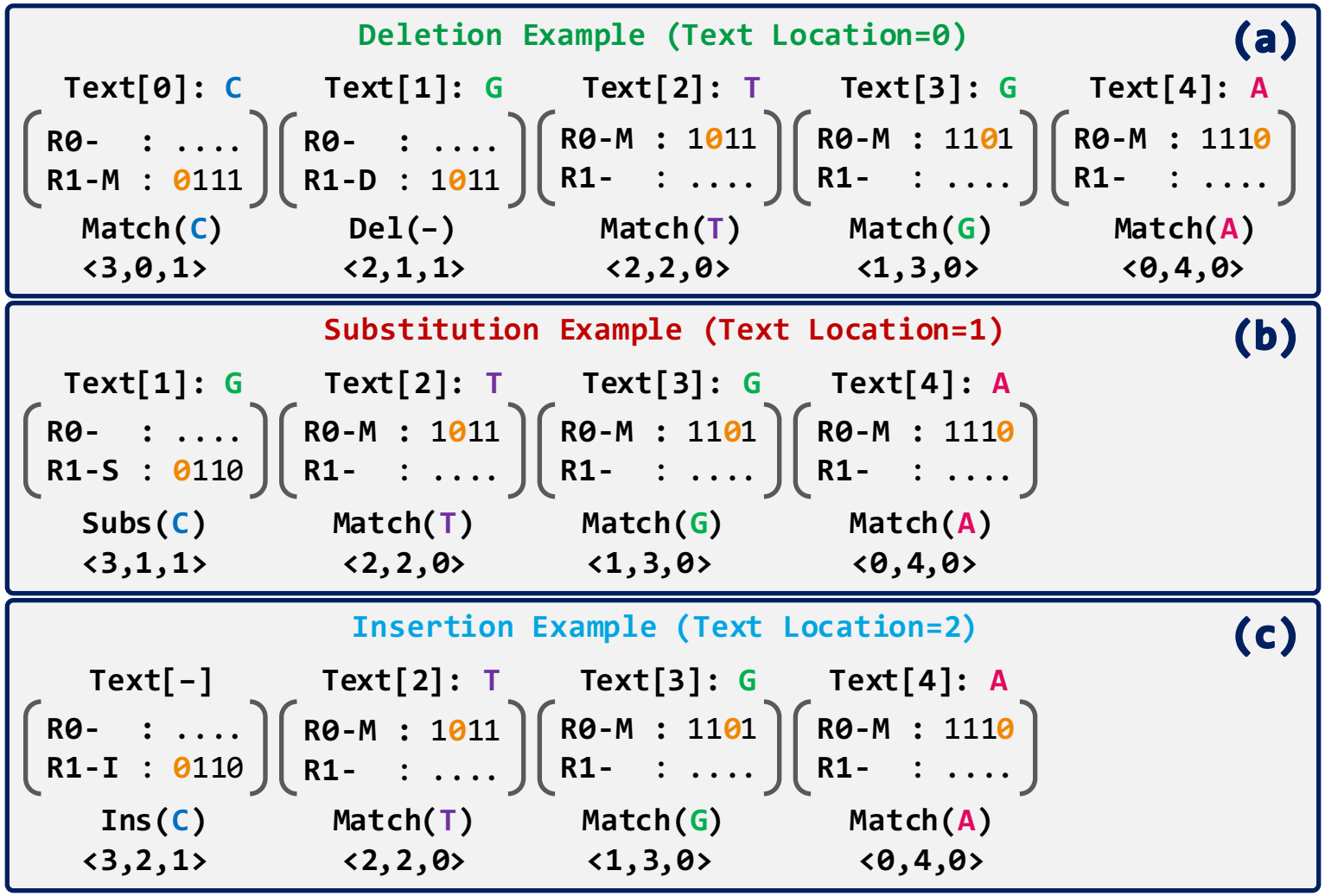
data ***written to memory***

data ***read from memory***

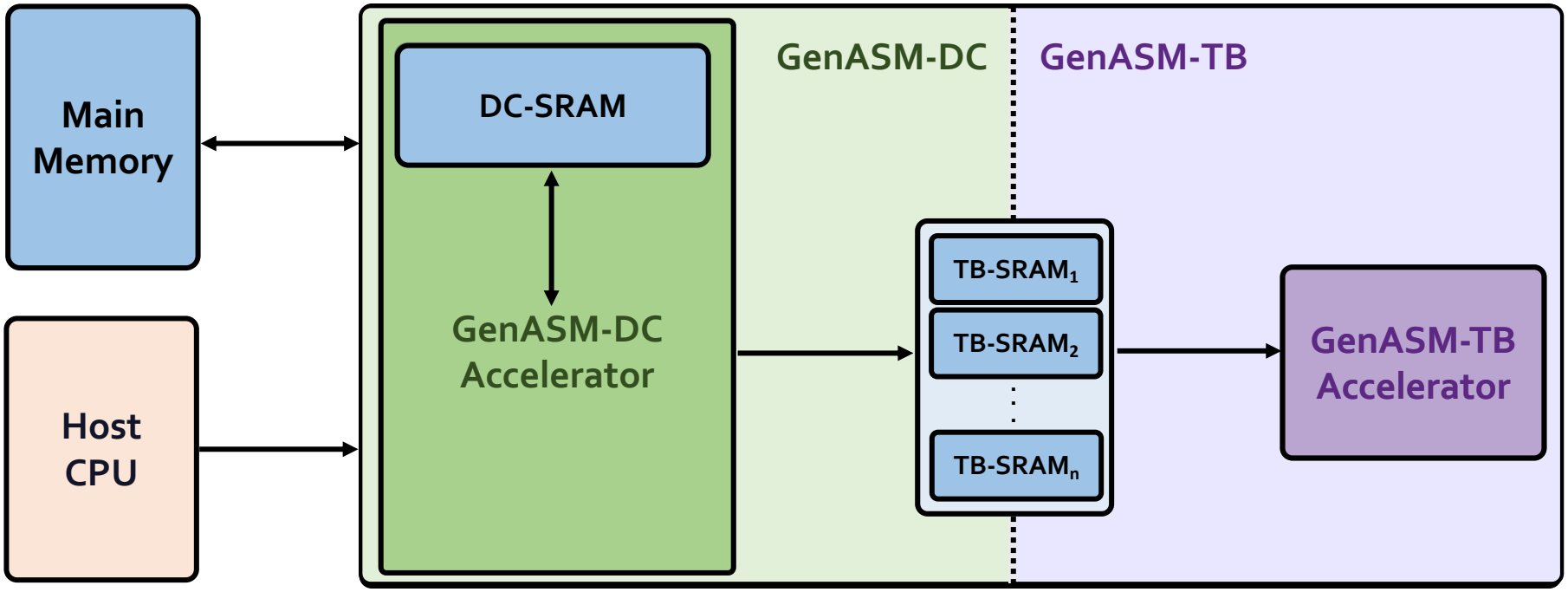
target cell (R_d)

cells target cell depends on ($oldR_d$, R_{d-1}, $oldR_{d-1}$)

Traceback Example with GenASM-TB



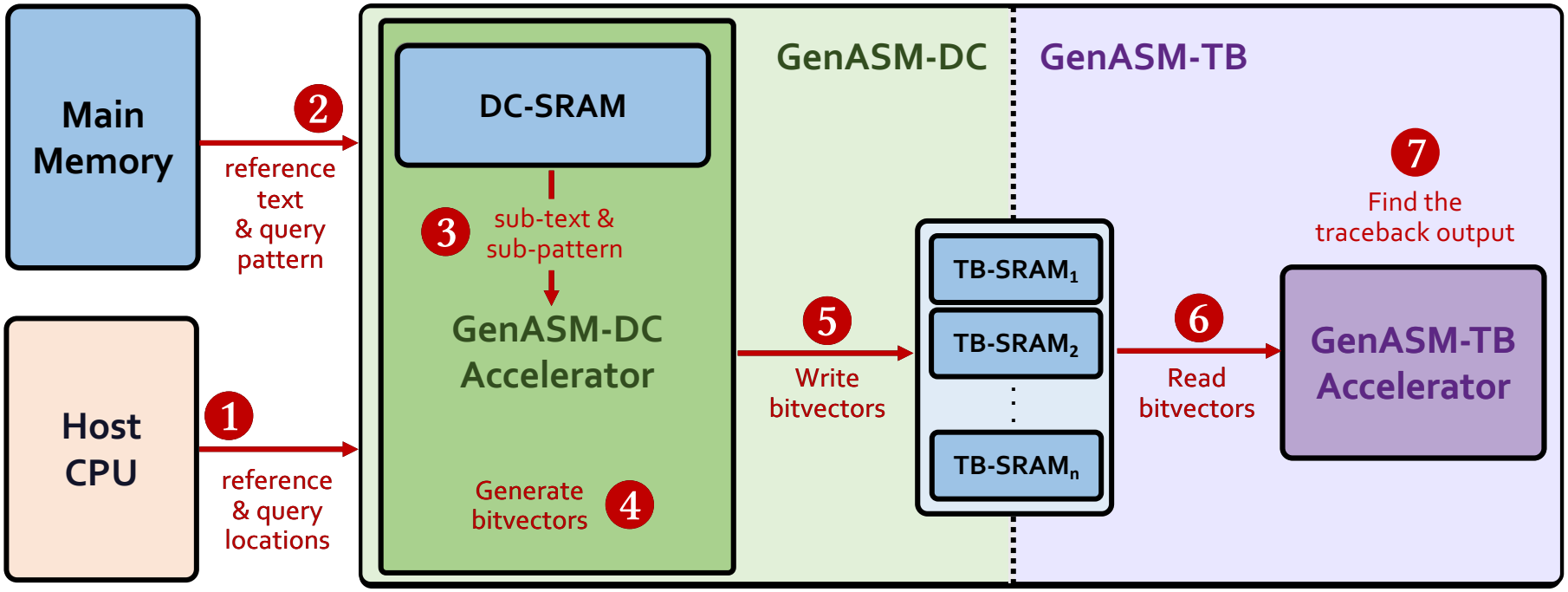
GenASM Hardware Design



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

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

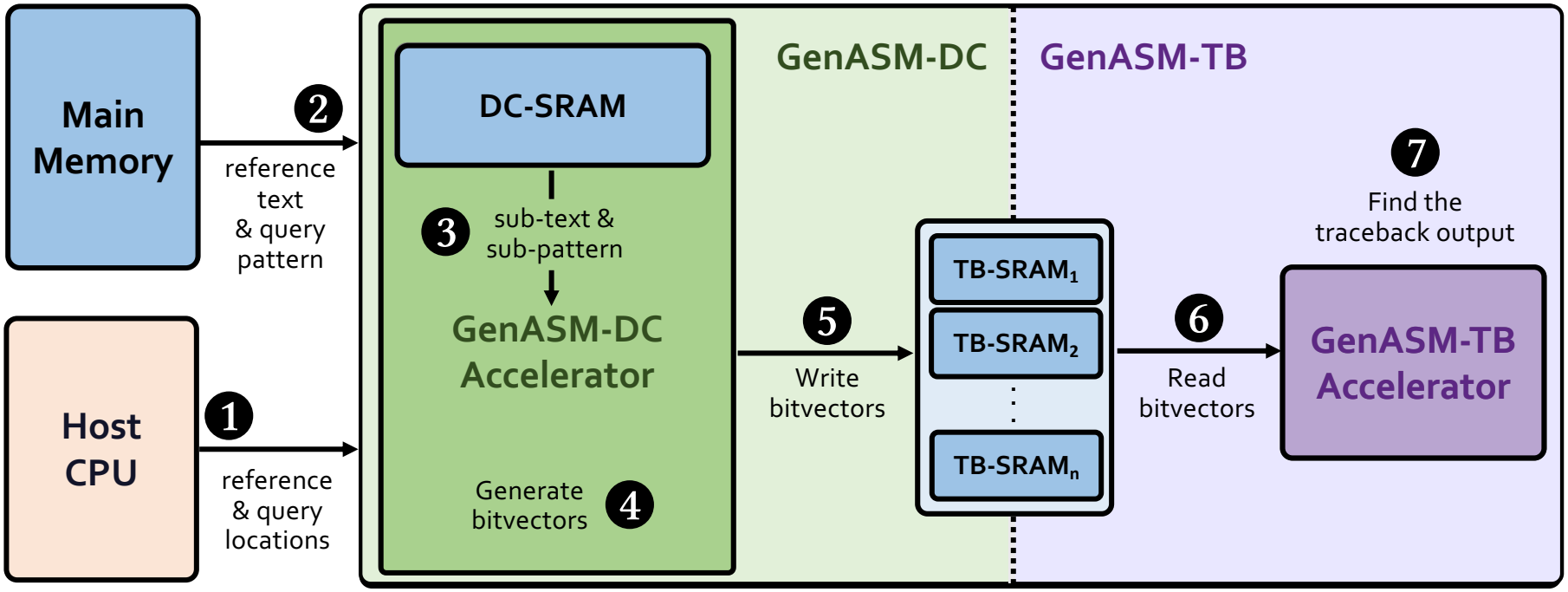
GenASM Hardware Design



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:

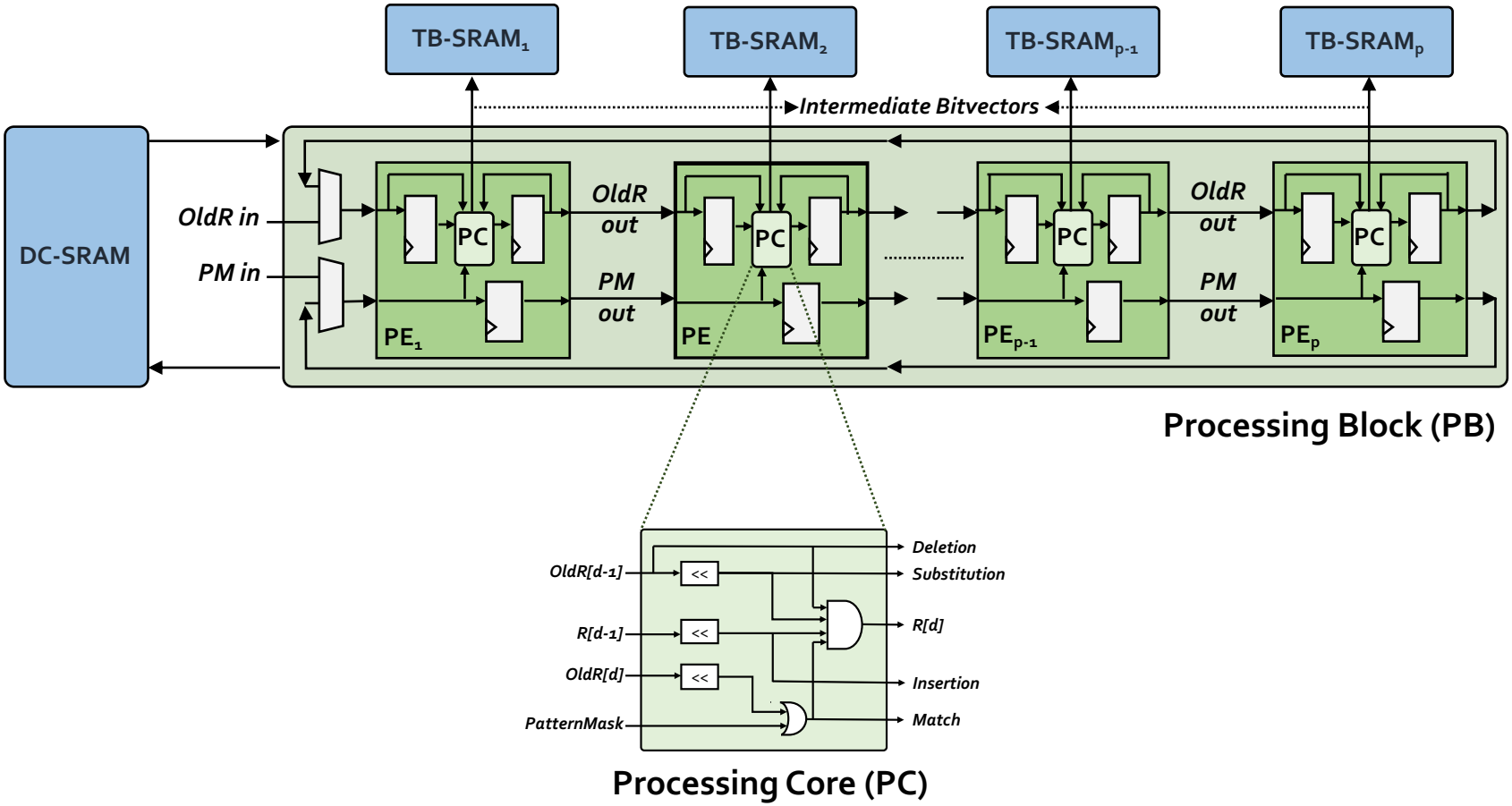
→ Match the rate of computation with memory capacity and bandwidth

→ Achieve high performance and power efficiency

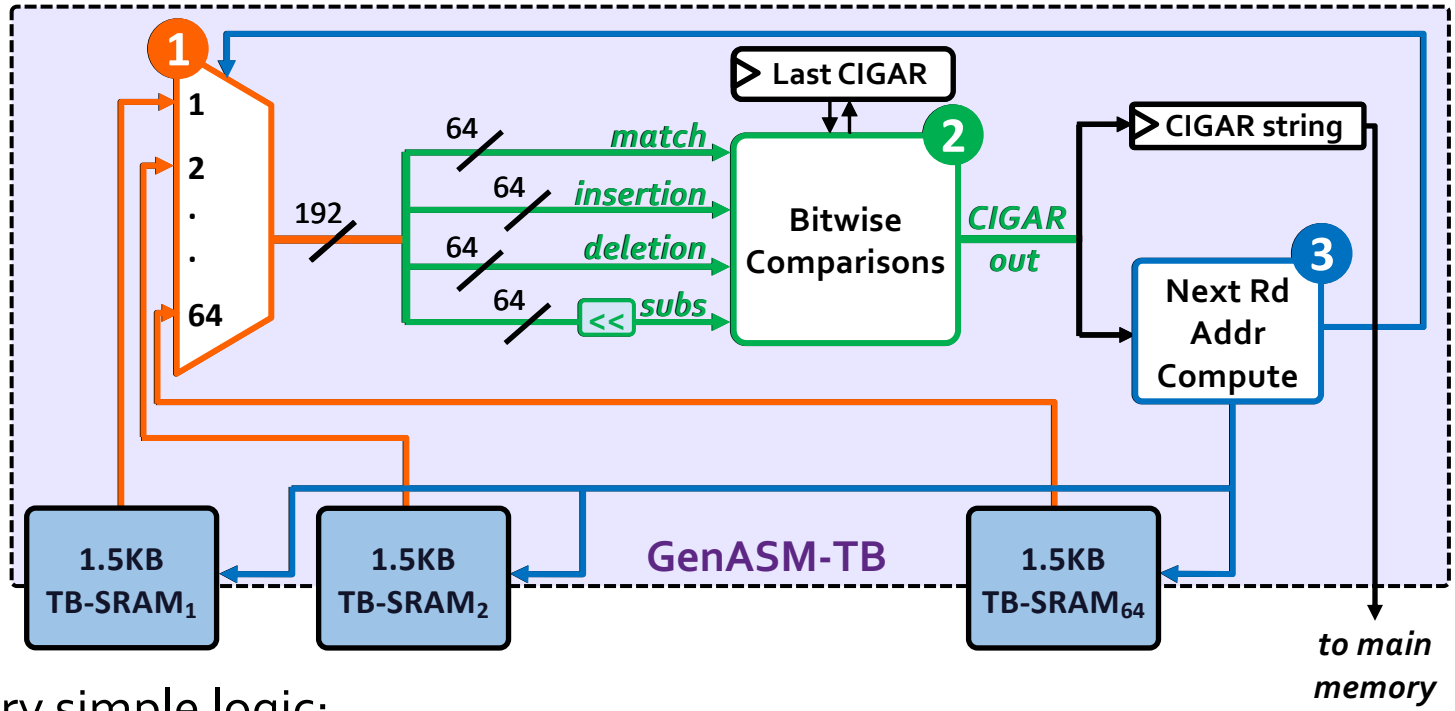
→ Scale linearly in performance with the number of parallel compute units that we add to the system

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

Use Cases of GenASM

(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

- We also discuss other possible use cases of GenASM in our paper:
  - Read-to-read overlap finding, hash-table based indexing, whole genome alignment, generic text search

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

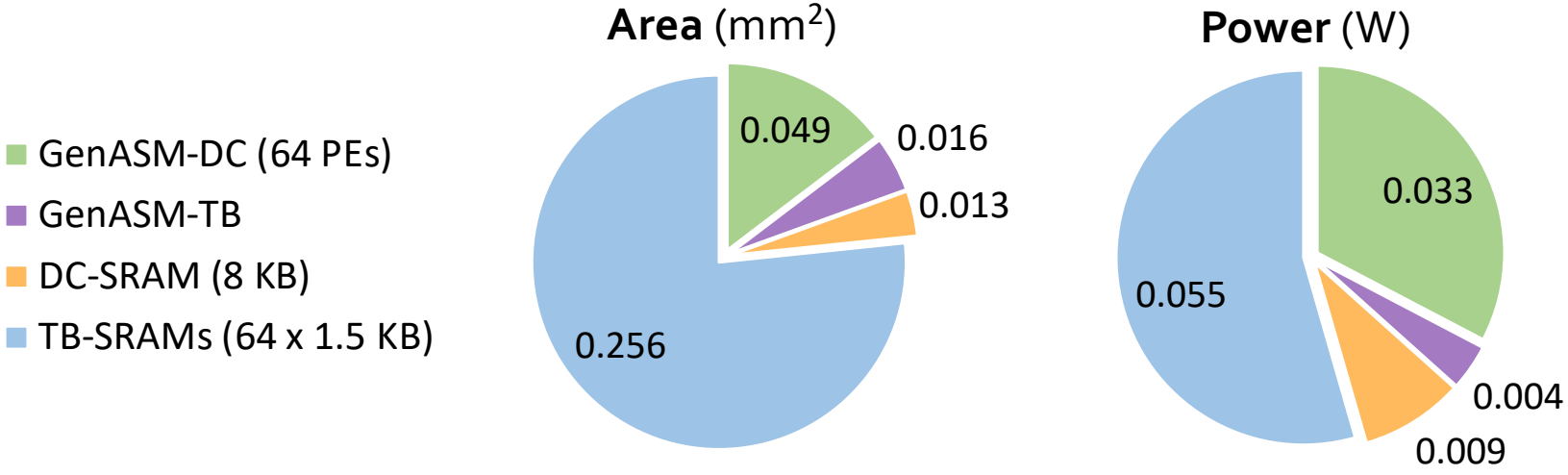
Evaluation Methodology (cont'd.)

- **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

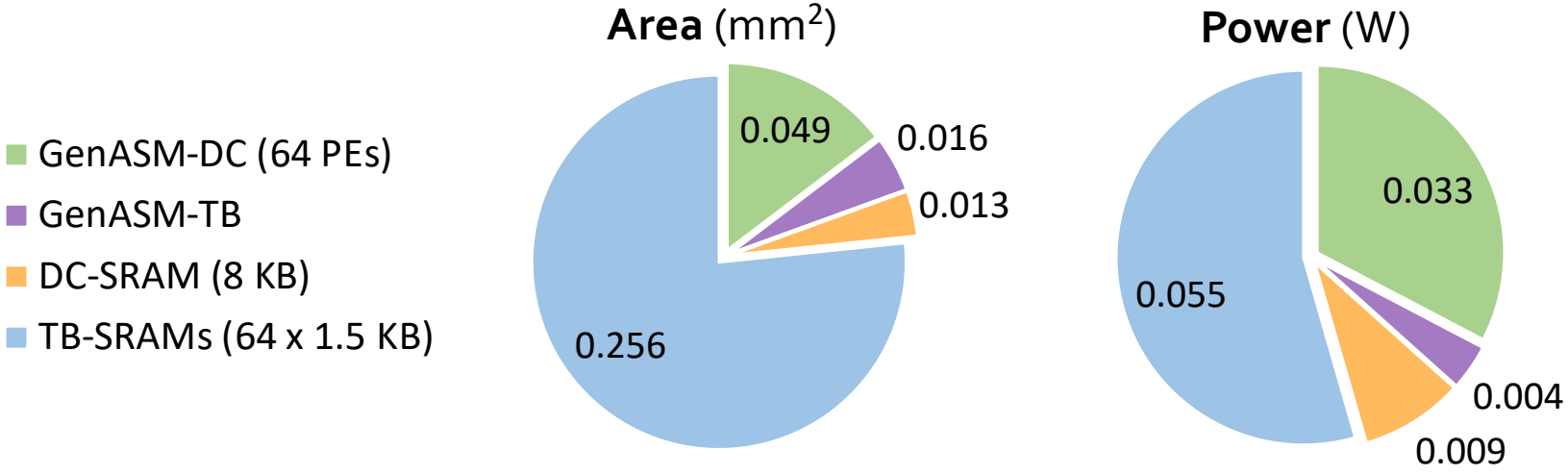
- Based on our **synthesis** of **GenASM-DC** and **GenASM-TB** accelerator datapaths using the Synopsys Design Compiler with a **28nm** process:
  - Both GenASM-DC and GenASM-TB operate **@ 1GHz**



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

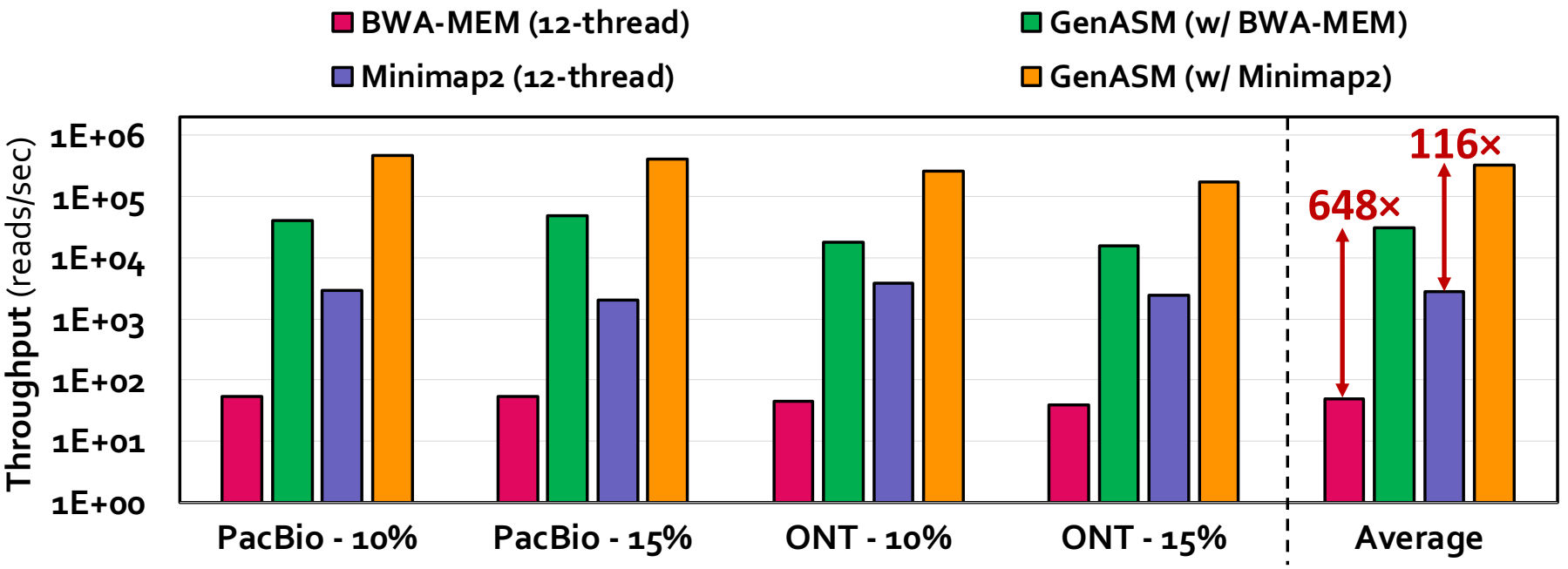
(2) Pre-Alignment Filtering for Short Reads

- 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

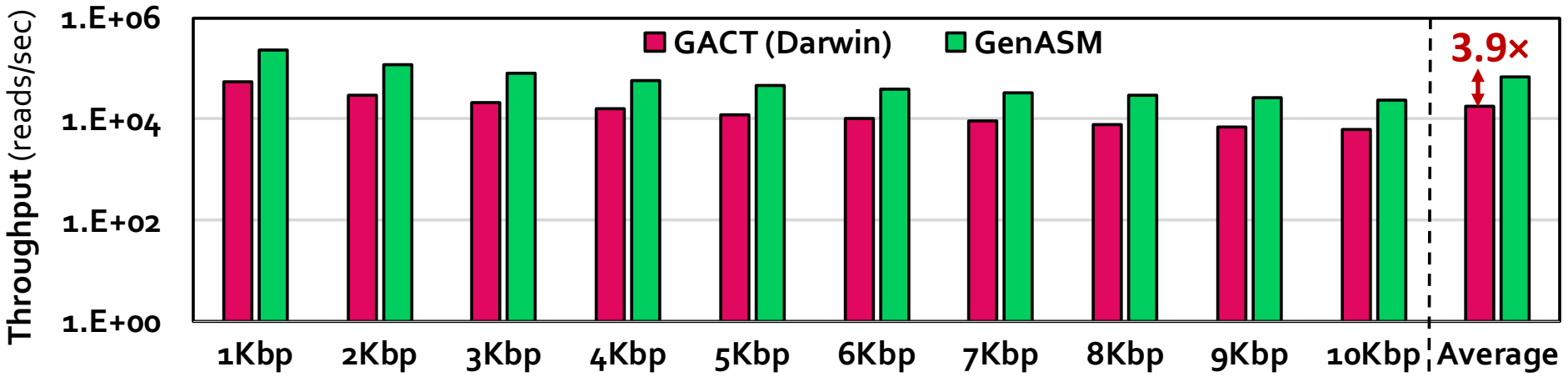
Key Results – Use Case 1 (Long Reads)



SW

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

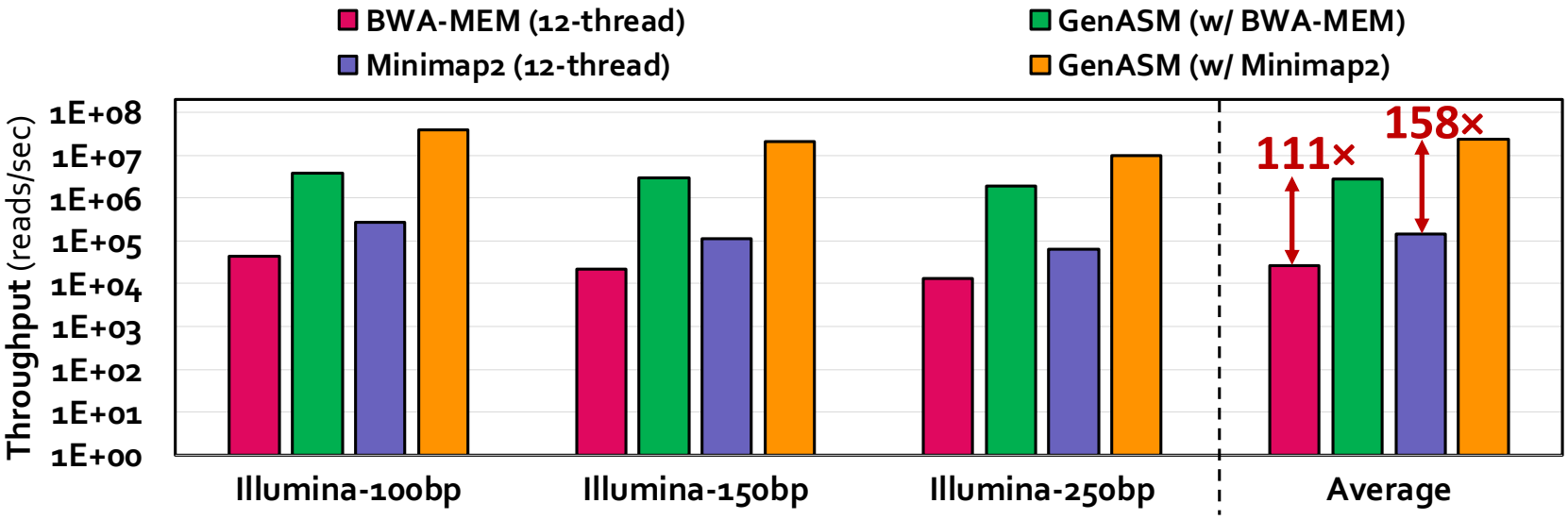
Key Results – Use Case 1 (Long Reads)



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

Key Results – Use Case 2

(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 2

- Compared to Shouji:
  - **3.7×** speedup
  - **1.7×** less power consumption
  - **False accept rate of 0.02%** for GenASM vs. 4% for Shouji
  - **False reject rate of 0%** for both GenASM and Shouji

HW

GenASM is **more efficient in terms of both speed and power consumption**, while **significantly improving the accuracy** of pre-alignment filtering

Key Results – Use Case 3

(1) Read Alignment Step of Read Mapping

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

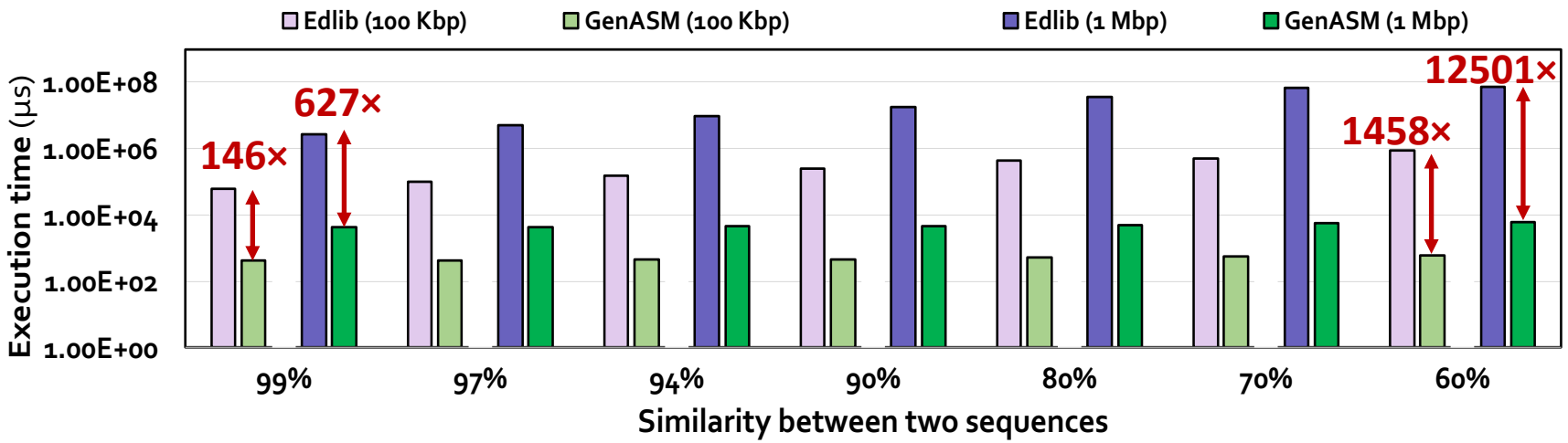
(2) Pre-Alignment Filtering for Short Reads

- 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 3



SW

GenASM provides **146 – 1458×** and **627 – 12501× speedup, while reducing power consumption by 548×** and **582×** for 100Kbp and 1Mbp sequences, respectively, compared to Edlib

HW

GenASM provides **9.3 – 400× speedup** over ASAP, while consuming **67× less power**

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

Summary of GenASM

- **Problem:**
  - Genome sequence analysis is bottlenecked by the computational power and memory bandwidth limitations of existing systems
  - This bottleneck is particularly an issue for *approximate string matching*

- **Key Contributions:**
  - GenASM: An approximate string matching (ASM) acceleration framework to accelerate multiple steps of genome sequence analysis
    - *First* to enhance and accelerate Bitap for ASM with genomic sequences
    - *Co-design* of our modified scalable and memory-efficient algorithms with low-power and area-efficient hardware accelerators
    - Evaluation of three different use cases: read alignment, pre-alignment filtering, edit distance calculation

- **Key Results:** GenASM is significantly more efficient for all the three use cases (in terms of throughput and throughput per unit power) than state-of-the-art software and hardware baselines

GenGraph: A Hardware Acceleration Framework for Sequence-to-Graph Mapping

Damla Senol Cali *et al.*

https://damlasenolcali.github.io

TECHCON'21 – September 14, 2021

This work was supported in part by Semiconductor Research Corporation (SRC).