Carnegie Mellon

SAFARI

Accelerating Genome Sequence Analysis via Efficient Hardware/Algorithm Co-Design

Damla Senol Cali

Carnegie Mellon University

(dsenol@andrew.cmu.edu)

June 24, 2021

Job Talk

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

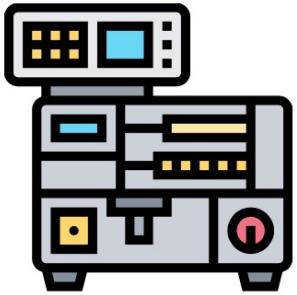
DNA

- **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

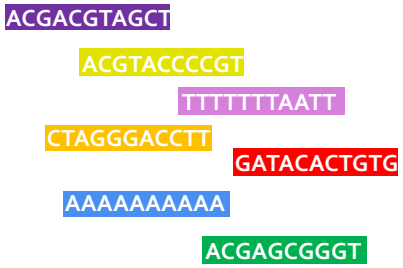
Genome Sequencing (cont'd.)

Sample Collection

Chopped DNA fragments



Sequencing



Sequenced reads

Large DNA molecule

Preparation

Genome Sequence Analysis

Sequencing Technologies

Oxford Nanopore (ONT)

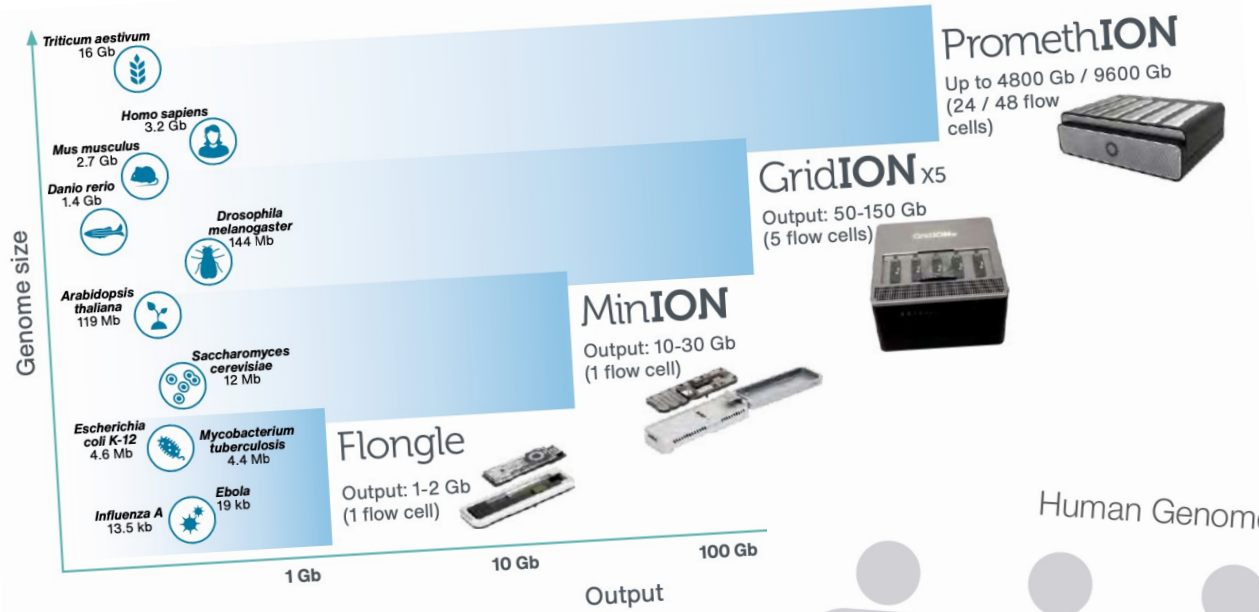
PacBio

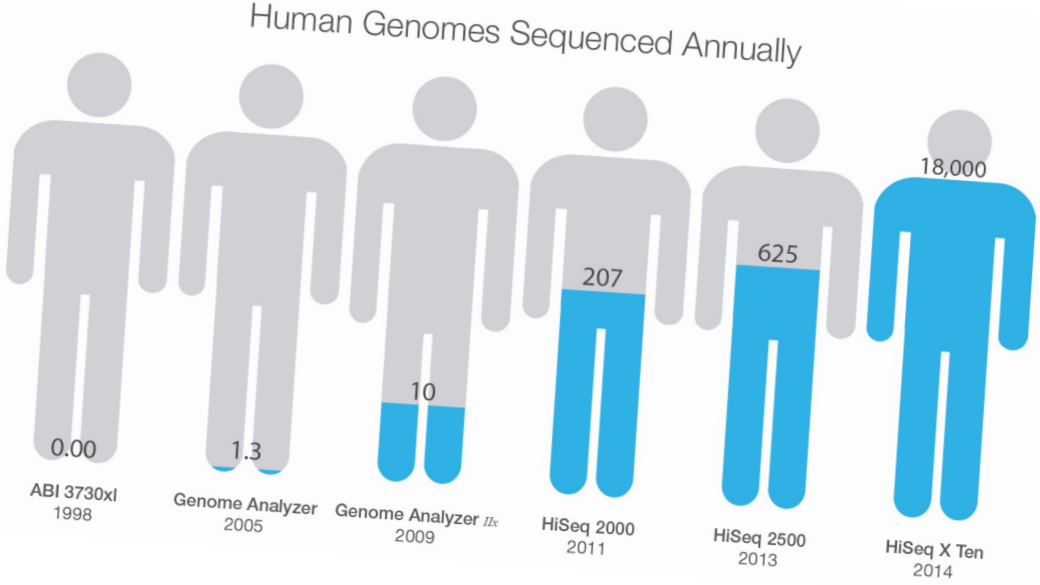
Illumina

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

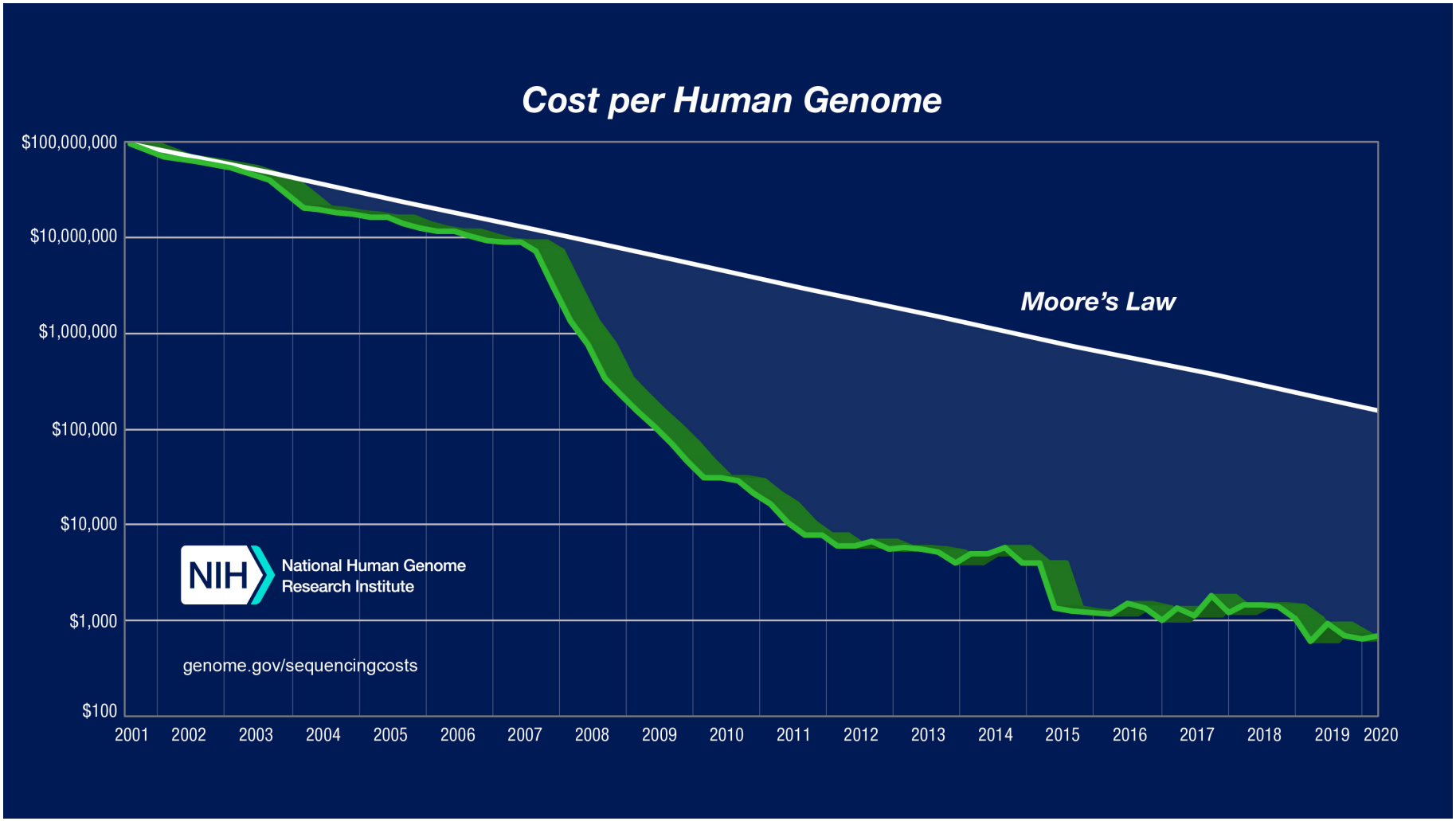
Long reads: **thousands to millions of base pairs** and **error rate of 5–10%**

Current State of Sequencing



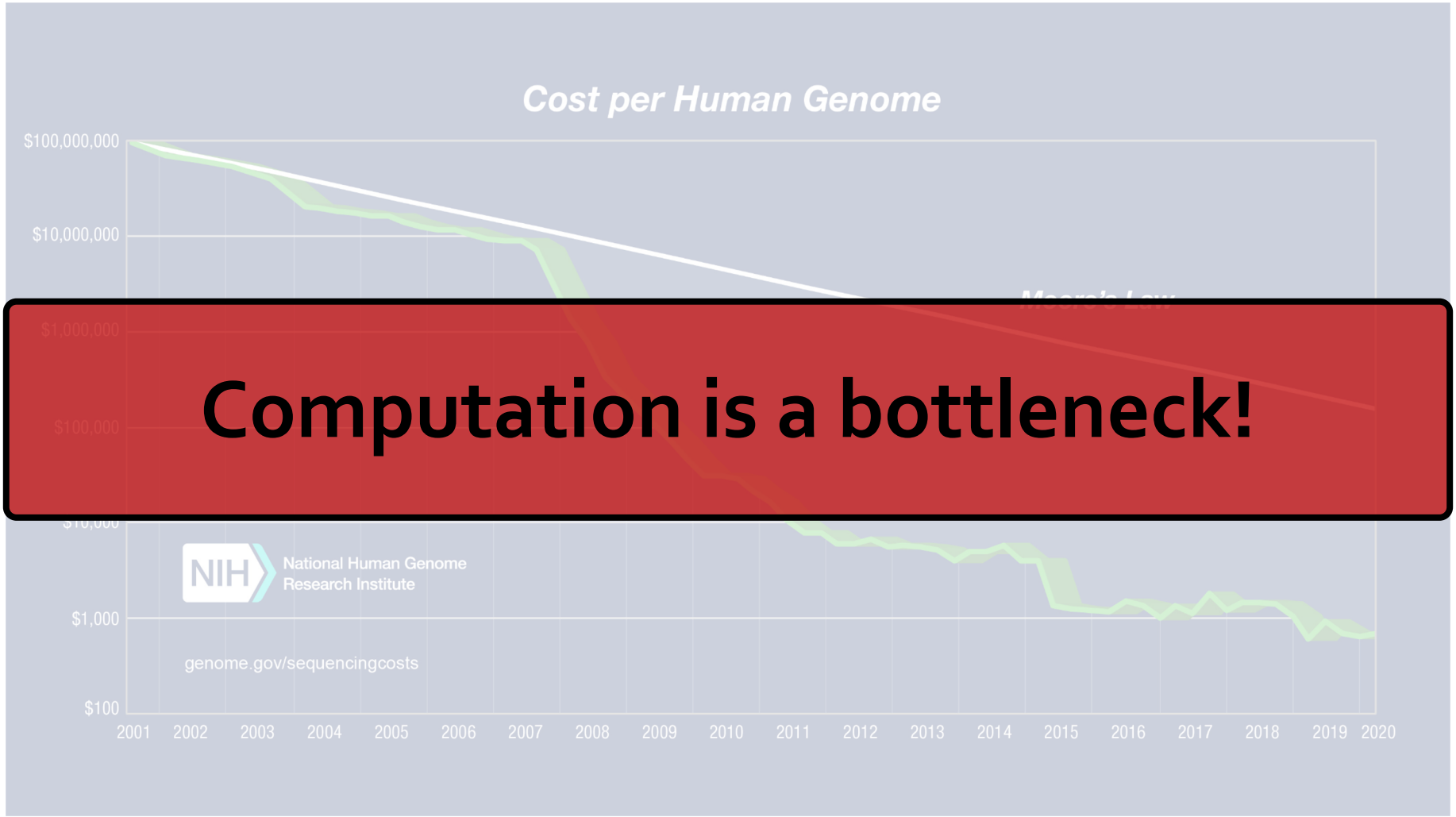


Current State of Sequencing (cont'd.)



*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)

Future of Genome Sequencing & Analysis

MinION from ONT

SmidgION from ONT

Problem Statement

Rapid genome sequence analysis is currently bottlenecked by **the computational power and memory bandwidth limitations of existing systems**, as many of the steps in genome sequence analysis must process **a large amount of data**

Our Goal & Approach

- **Our Goal:**

  Accelerating genome sequence analysis by **efficient hardware/algorithm co-design**

- **Our Approach:**
  1. Analyze the multiple steps and the associated tools in the genome sequence analysis pipeline,
  2. Expose the tradeoffs between accuracy, performance, memory usage and scalability, and
  3. Co-design fast and efficient algorithms along with scalable and energy-efficient customized hardware accelerators for the key bottleneck steps of the pipeline

Outline

Bottleneck analysis of long read assembly

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Ongoing]

GenGraph: Hardware acceleration framework for sequence-to-graph mapping

[Ongoing]

Outline

Bottleneck analysis of long read assembly

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

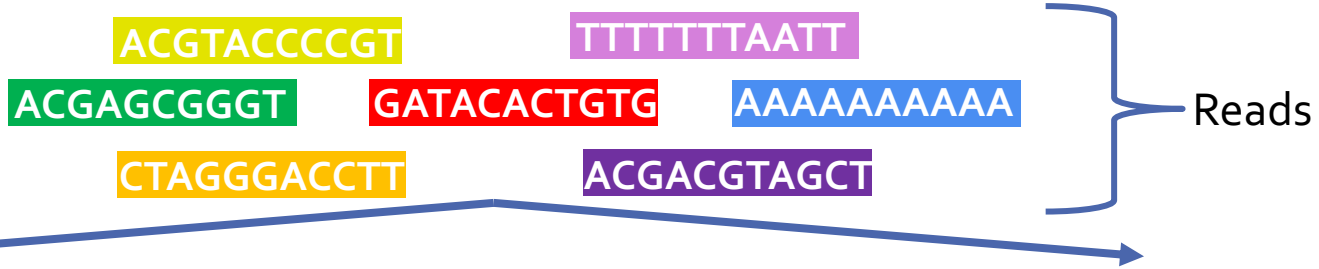
BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Ongoing]

GenGraph: Hardware acceleration framework for sequence-to-graph mapping

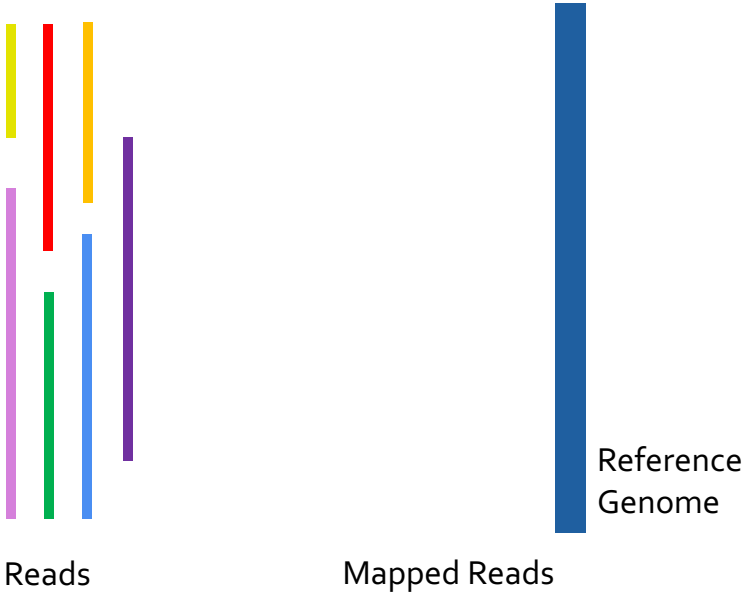
[Ongoing]

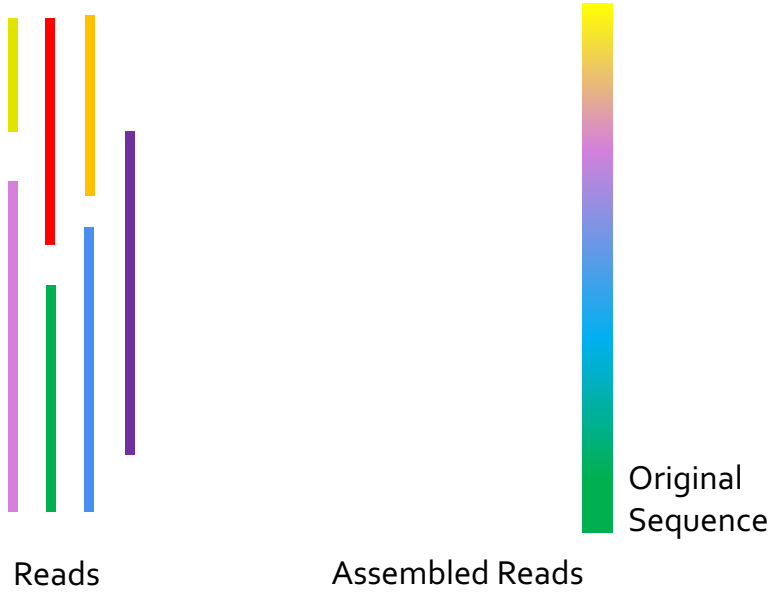
Genome Sequence Analysis



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

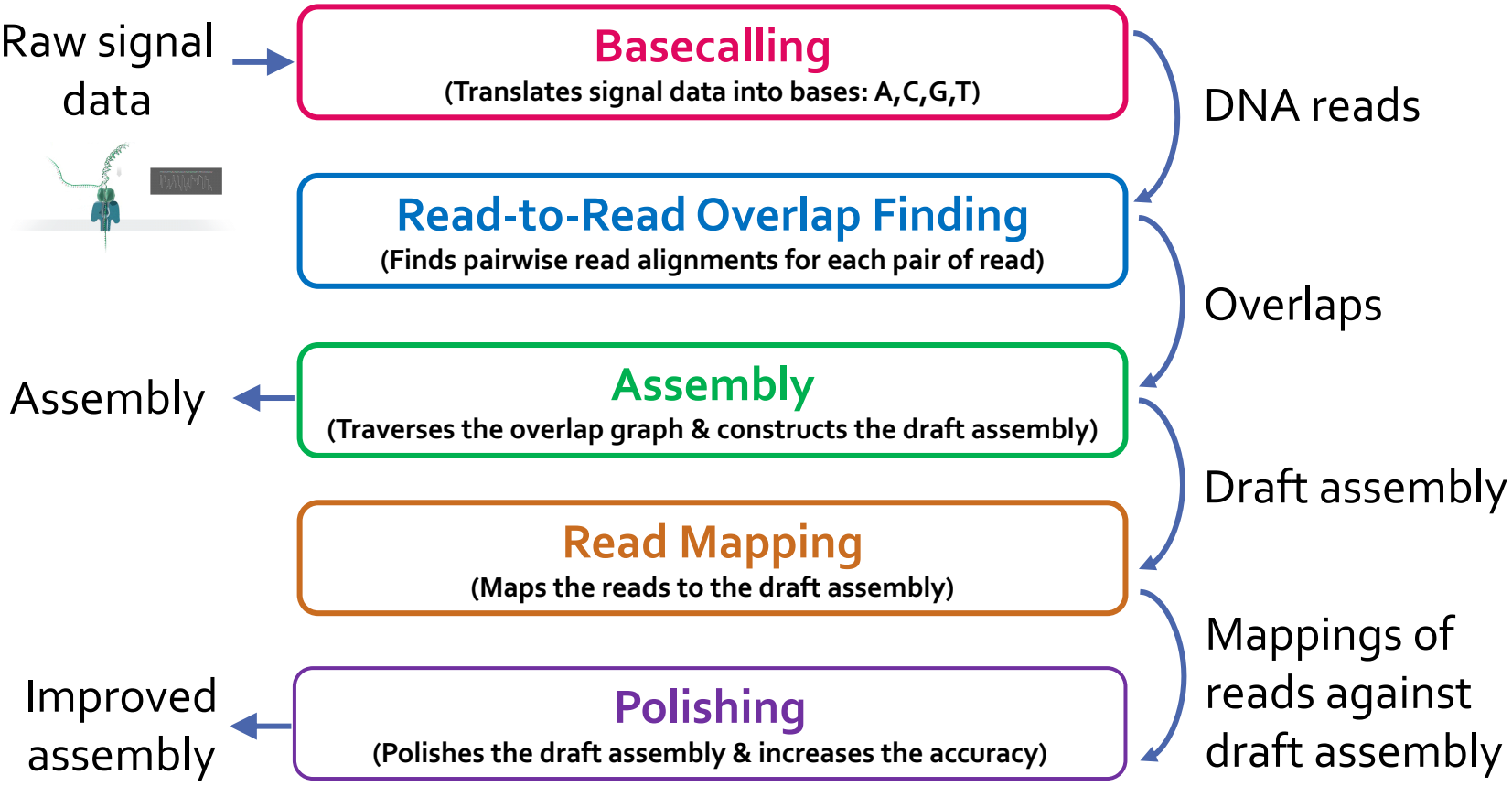
***De novo* Assembly,** method of merging the reads in order to **construct** the original sequence.





Genome Assembly Pipeline Using Long Reads

- With the emergence of long read sequencing technologies, *de novo* assembly becomes a promising way of constructing the original genome.



Our Contributions

- Analyze the tools in multiple dimensions: accuracy, performance, memory usage, and scalability
- Reveal new bottlenecks and trade-offs
- First study on bottleneck analysis of nanopore sequence analysis pipeline on real machines
- Provide guidelines for practitioners
- Provide guidelines for tool developers

Experimental Methodology

Name	Model	CPU specifications	Main memory specifications
System 1	40-core Intel® Xeon® E5-2630 v4 CPU @ 2.20GHz	20 physical cores 2 threads per core 40 logical cores with hyper-threading**	128GB DDR4 2 channels, 2 ranks/channel Speed: 2400MHz
System 2 (desktop)	8-core Intel® Core i7-2600 CPU @ 3.40GHz	4 physical cores 2 threads per core 8 logical cores with hyper-threading**	16GB DDR3 2 channels, 2 ranks/channel Speed: 1333MHz
System 3 (big-mem)	80-core Intel® Xeon® E7-4850 CPU @ 2.00GHz	40 physical cores 2 threads per core 80 logical cores with hyper-threading**	1TB DDR3 8 channels, 4 ranks/channel Speed: 1066MHz

Experimental Methodology (cont'd.)

Accuracy Metrics

- Average Identity : Percentage **similarity** between the assembly and the reference genome
- Coverage: **Ratio of the #aligned bases** in the reference genome to the length of reference genome
- Number of mismatches: Total number of **single-base differences** between the assembly and the reference genome
- Number of indels: Total number of **insertions and deletions** between the assembly and the reference genome

Performance Metrics

- Wall clock time
- Peak memory usage
- Parallel speedup

} Using **/usr/bin/time** & **perf**

Key Findings

- Laptops are becoming a popular platform for running genome assembly tools, as the portability of a laptop makes it a good fit for in-field analysis
  - Greater memory constraints
  - Lower computational power
  - Limited battery life

- Memory usage is an important factor that greatly affects the performance and the usability of the tool
  - Data structure choices that increase the memory requirements
  - Algorithms that are not cache-efficient
  - Not keeping memory usage in check with the number of threads

- Scalability of the tool with the number of cores is an important requirement. However, parallelizing the tool can increase the memory usage
  - Not dividing the input data into batches
  - Not limiting the memory usage of each thread
  - Dividing the dataset instead of the computation between simultaneous threads

Key Findings

Goal 1: High-performance and low-power

- Memory usage is an important factor that greatly affects the performance and the usability of the tool
  - Data structure choices that increase the memory requirements
  - Algorithms that are not cache-efficient
  - Not keeping memory usage in check with the number of threads

- Scalability of the tool with the number of cores is an important requirement. However, parallelizing the tool can increase the memory usage
  - Not dividing the input data into batches
  - Not limiting the memory usage of each thread
  - Dividing the dataset instead of the computation between simultaneous threads

Key Findings

Goal 1:
High-performance and low-power

Goal 2:
Memory-efficient

- Scalability of the tool with the number of cores is an important requirement. However, parallelizing the tool can increase the memory usage
  - Not dividing the input data into batches
  - Not limiting the memory usage of each thread
  - Dividing the dataset instead of the computation between simultaneous threads

Key Findings

Goal 1:
High-performance and low-power

Goal 2:
Memory-efficient

Goal 3:
Scalable/highly-parallel

Outline

Bottleneck analysis of long read assembly

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

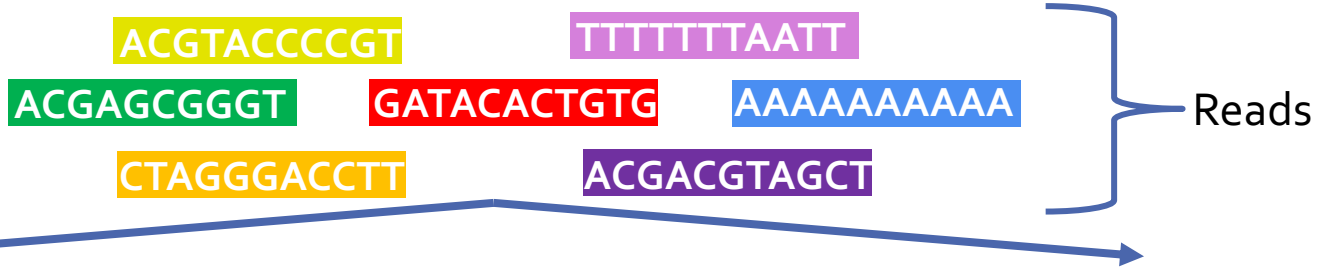
BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Ongoing]

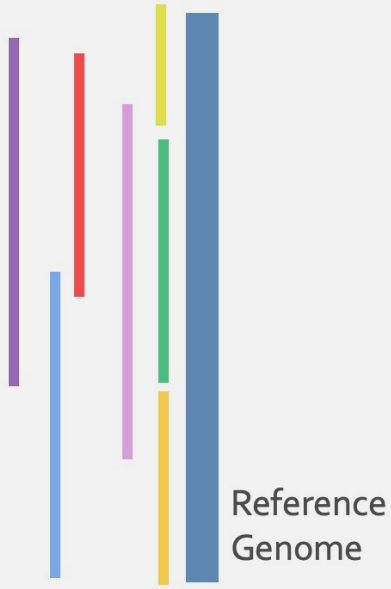
GenGraph: Hardware acceleration framework for sequence-to-graph mapping

[Ongoing]

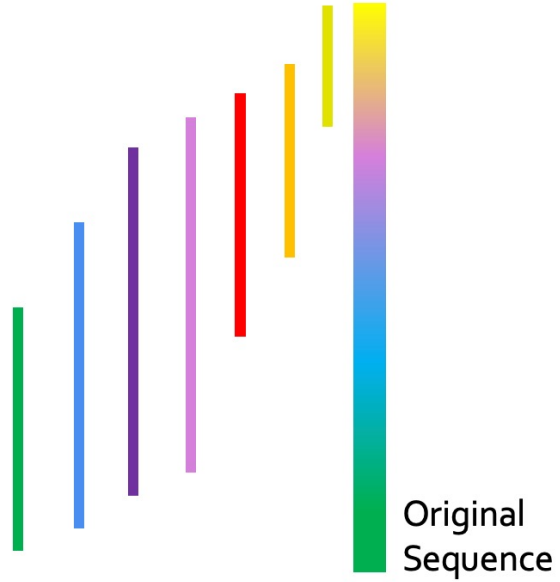
Recall: Genome Sequence Analysis



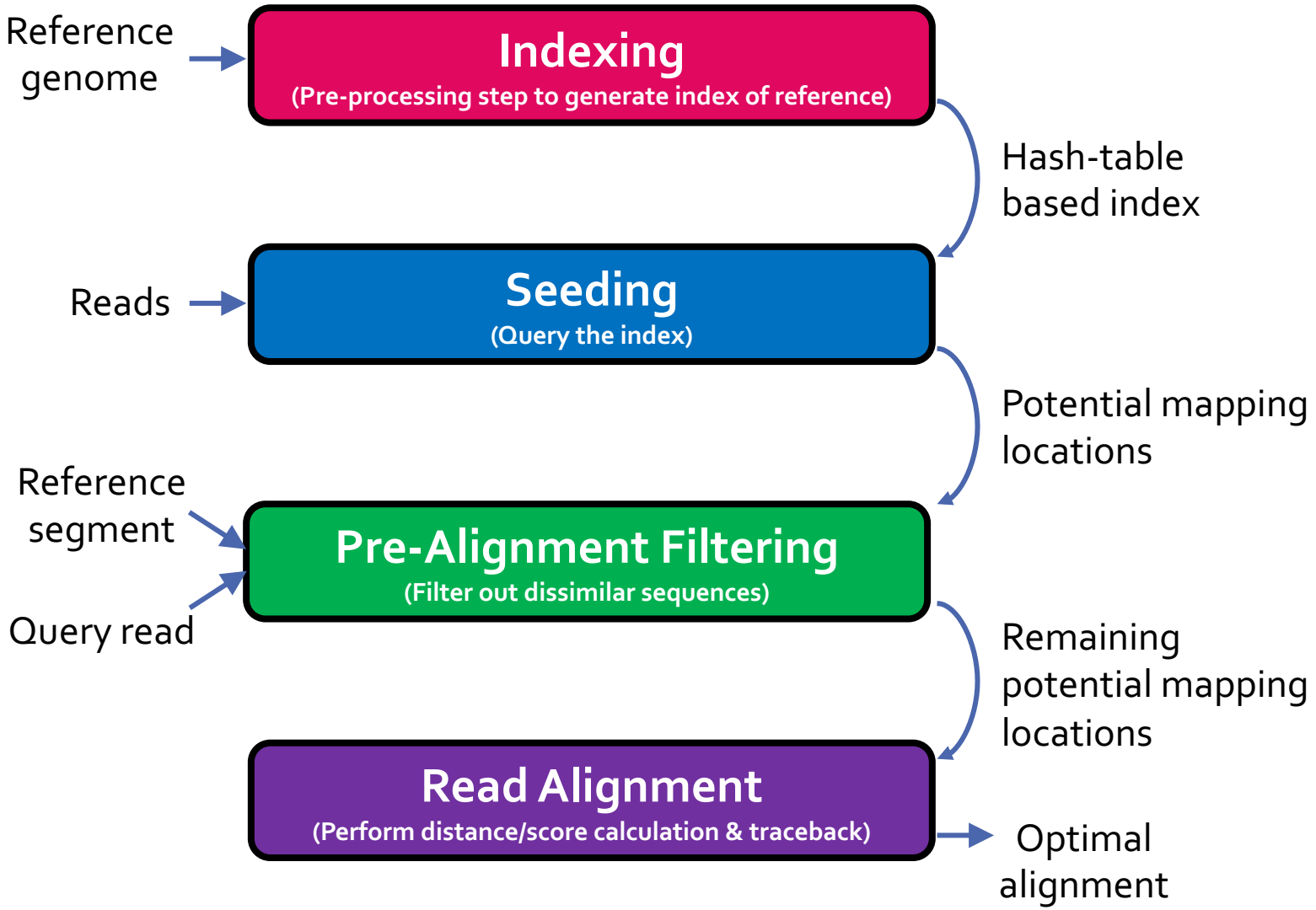
Read Mapping, method of aligning the reads against the reference genome in order to **detect matches and variations.**



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



Read Mapping Pipeline



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

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

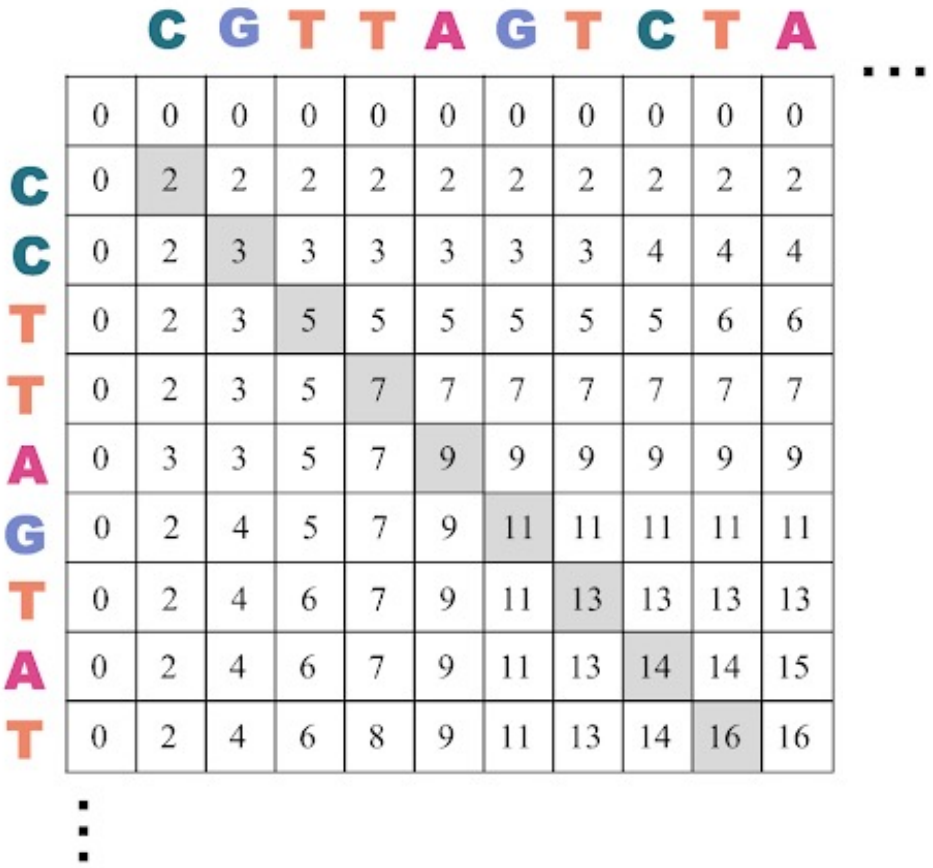
- **GenASM:** *First* ASM acceleration framework for GSA
  - Based upon the *Bitap* algorithm
    - Uses fast and simple bitwise operations to perform ASM
  - Modified and extended ASM algorithm
    - Highly-parallel Bitap with long read support
    - Novel bitvector-based algorithm to perform *traceback*
  - 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.

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):

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.

Large number of iterations

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]

  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

Algorithm

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
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

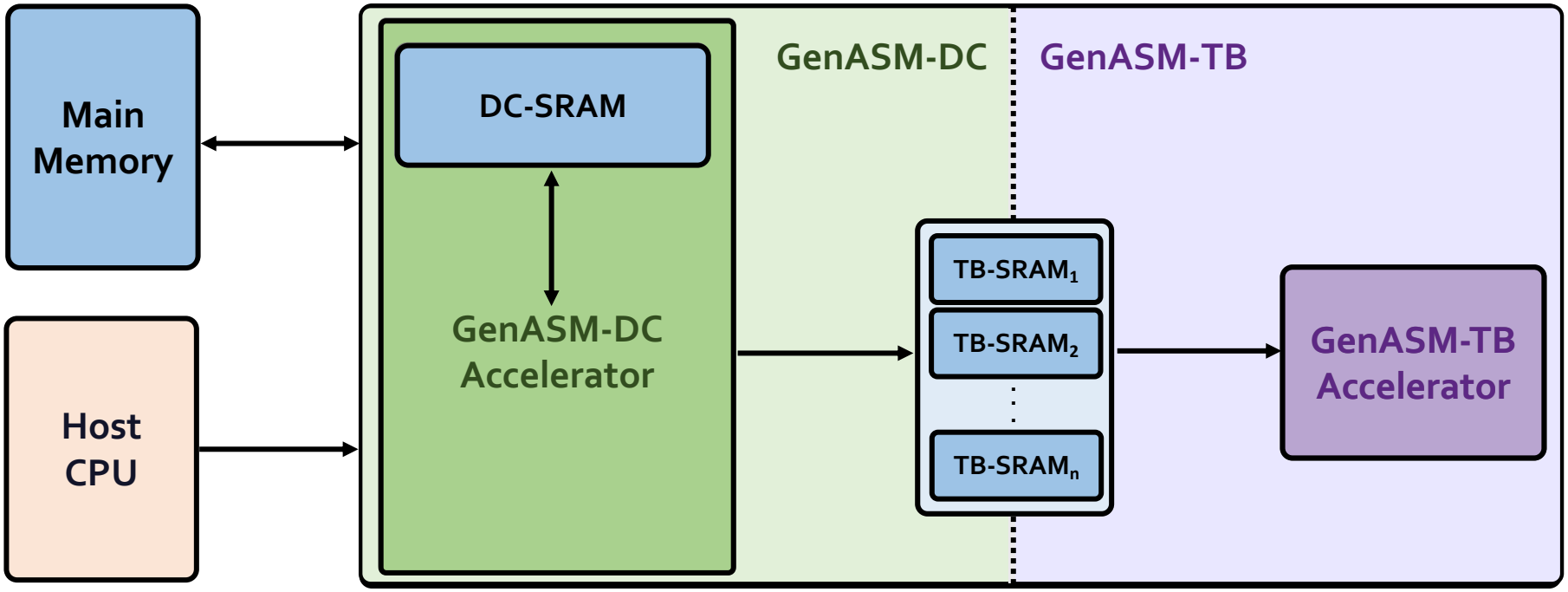
Hardware

4) **Limited Compute Parallelism:**
   - 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**

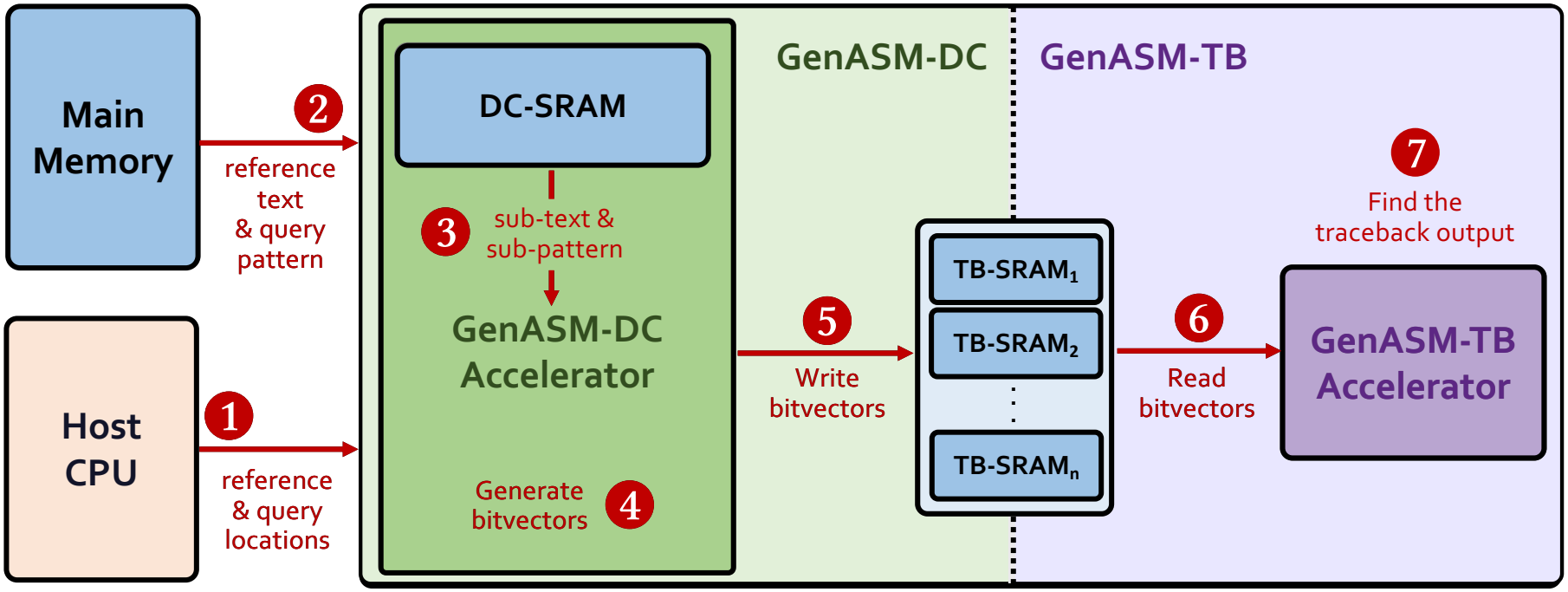
GenASM Hardware Design



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

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

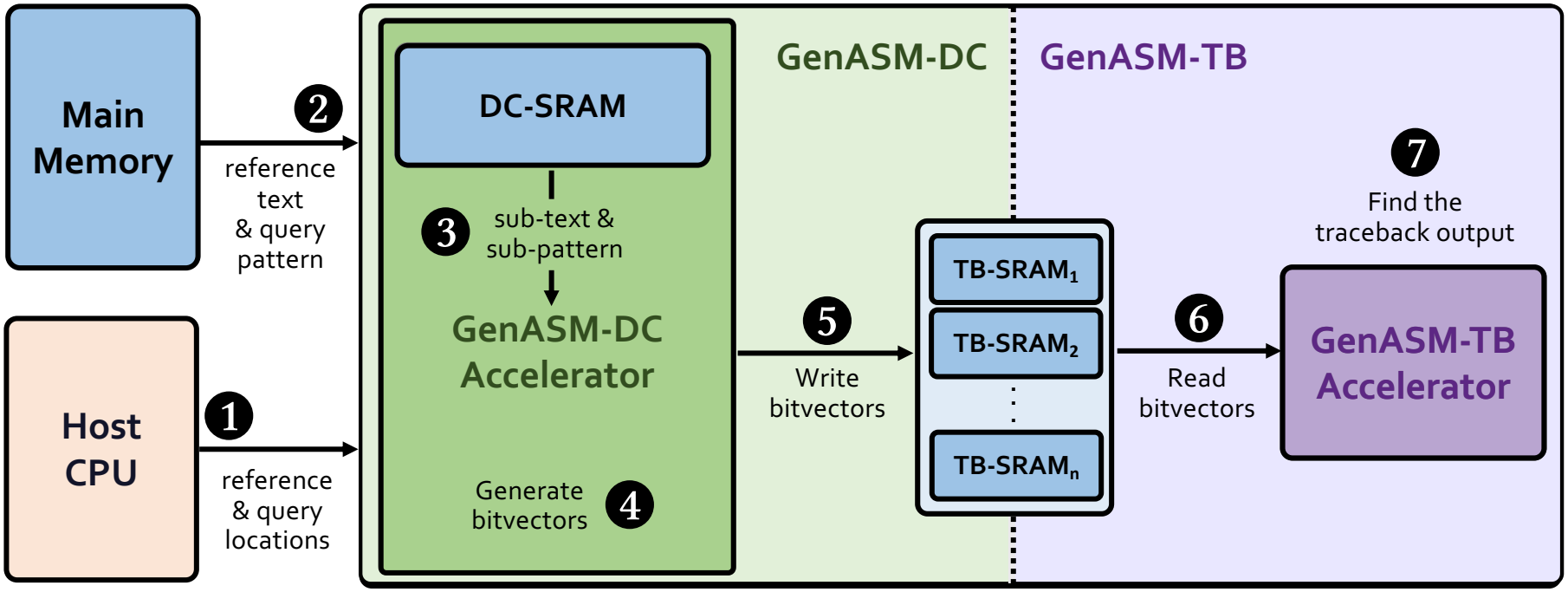
GenASM Hardware Design



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:

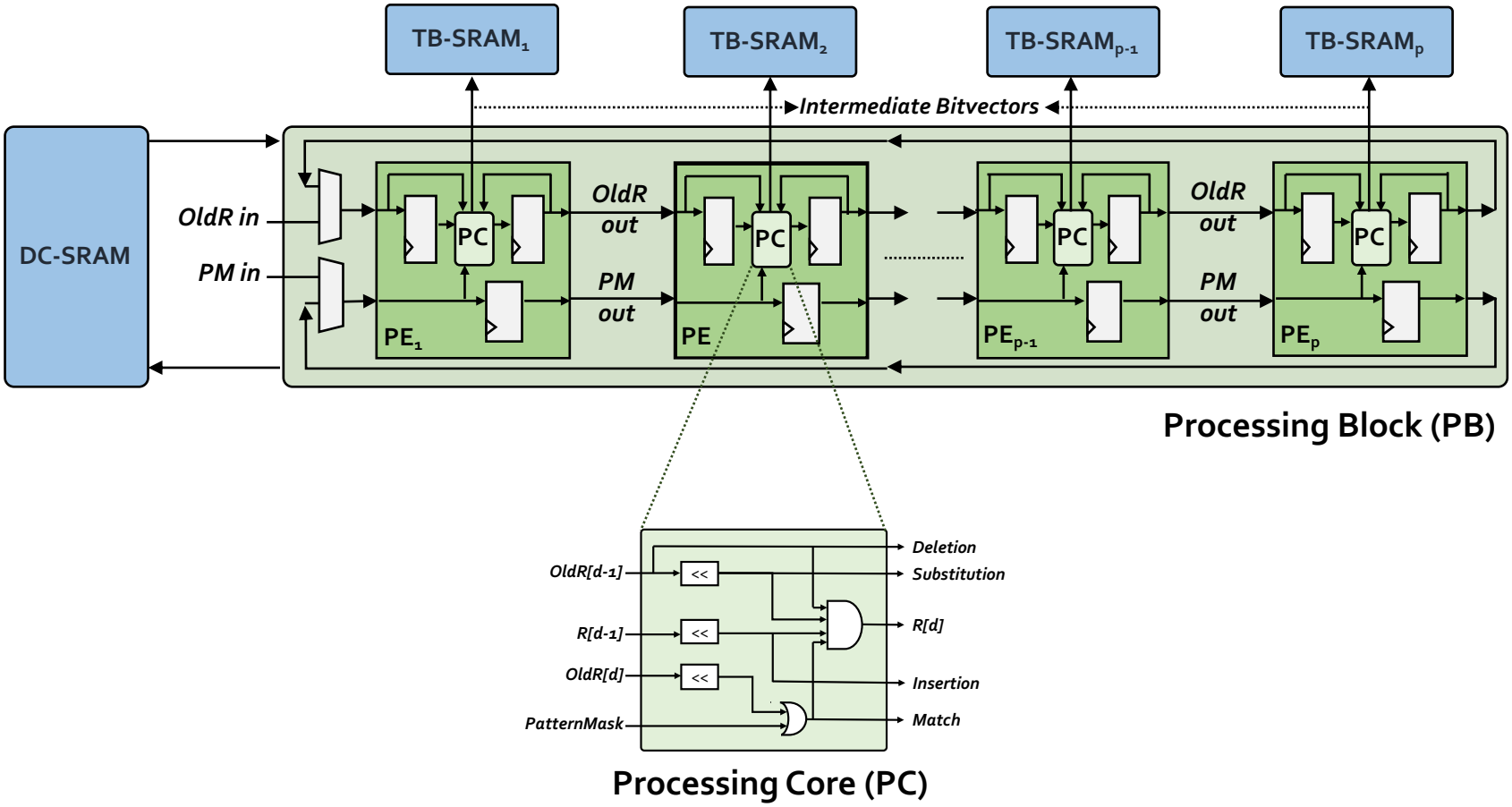
→ Match the rate of computation with memory capacity and bandwidth

→ Achieve high performance and power efficiency

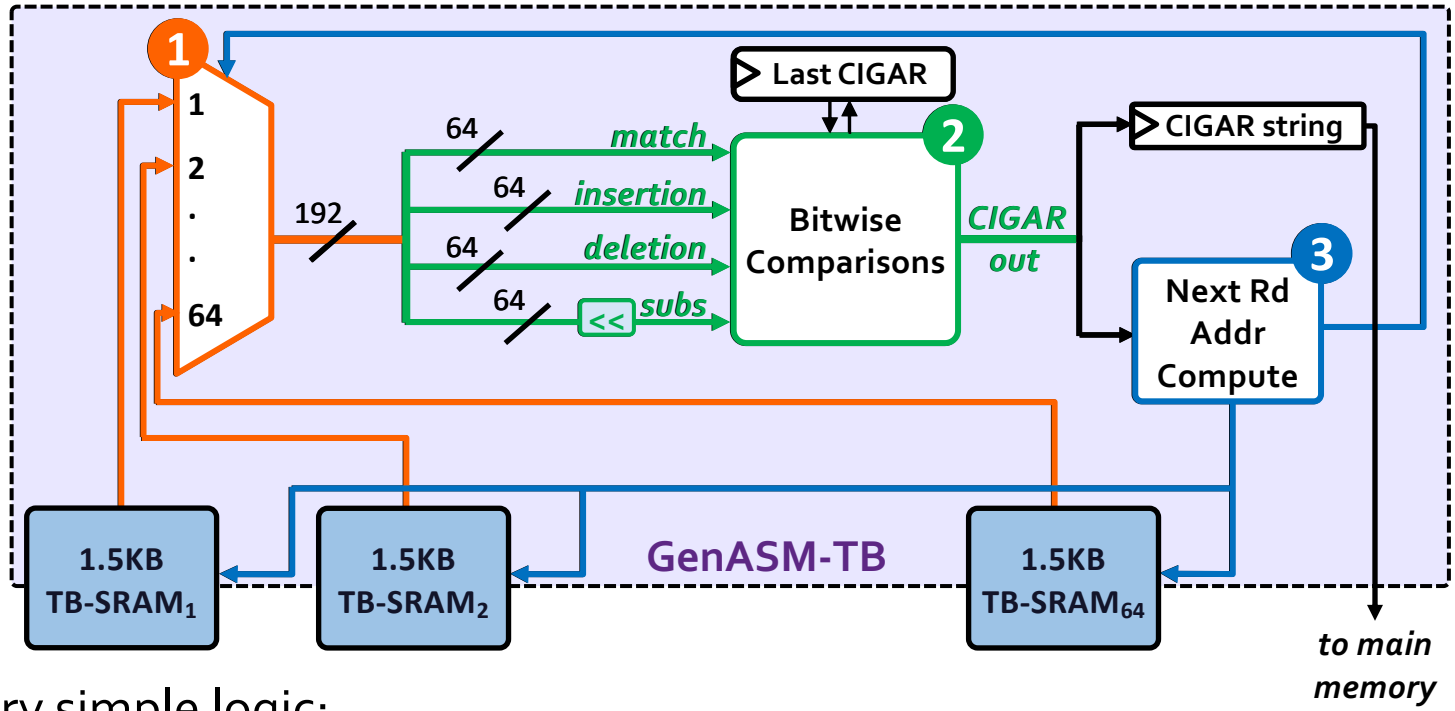
→ Scale linearly in performance with the number of parallel compute units that we add to the system

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

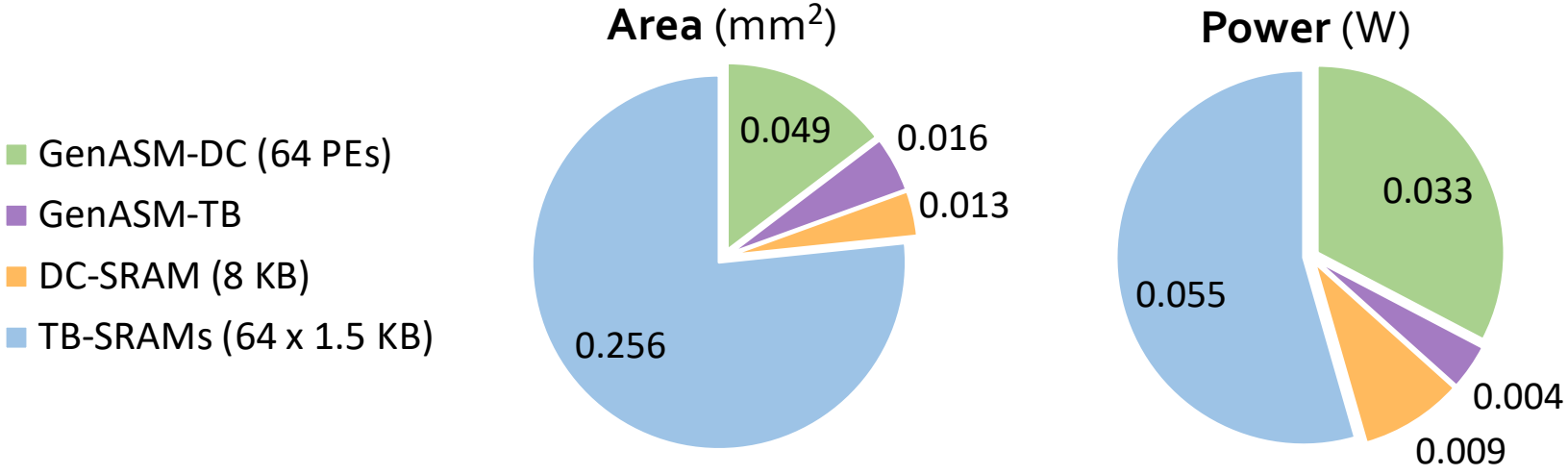
Evaluation Methodology (cont'd.)

- **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

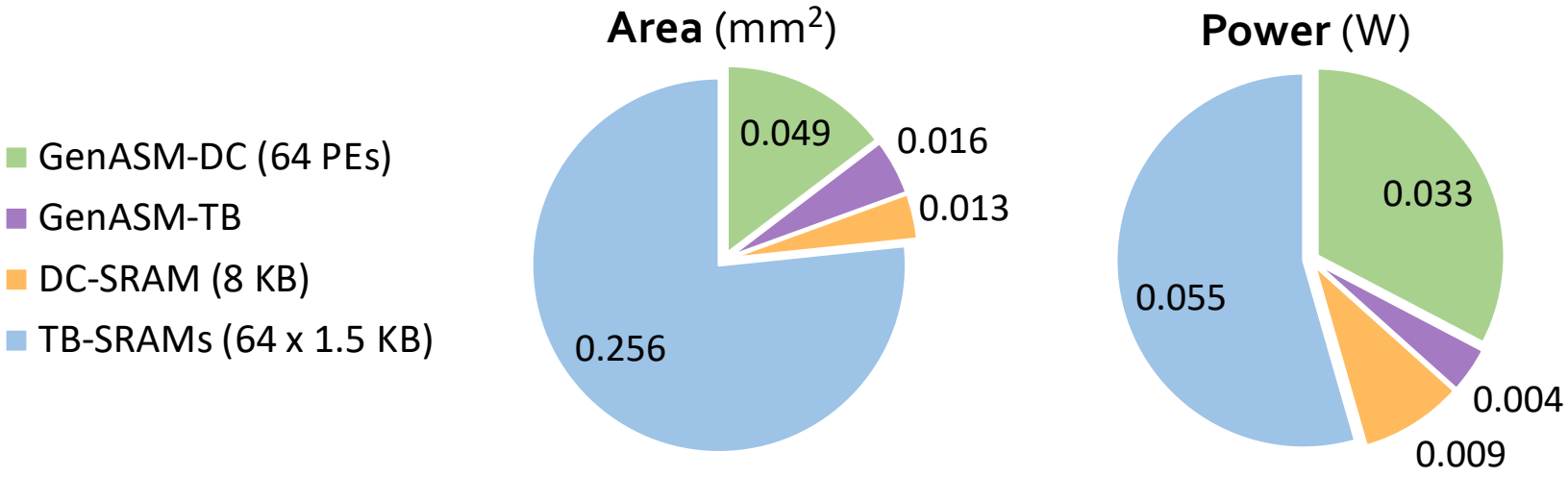
- Based on our **synthesis** of **GenASM-DC** and **GenASM-TB** accelerator datapaths using the Synopsys Design Compiler with a **28nm** process:
  - Both GenASM-DC and GenASM-TB operate **@ 1GHz**



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

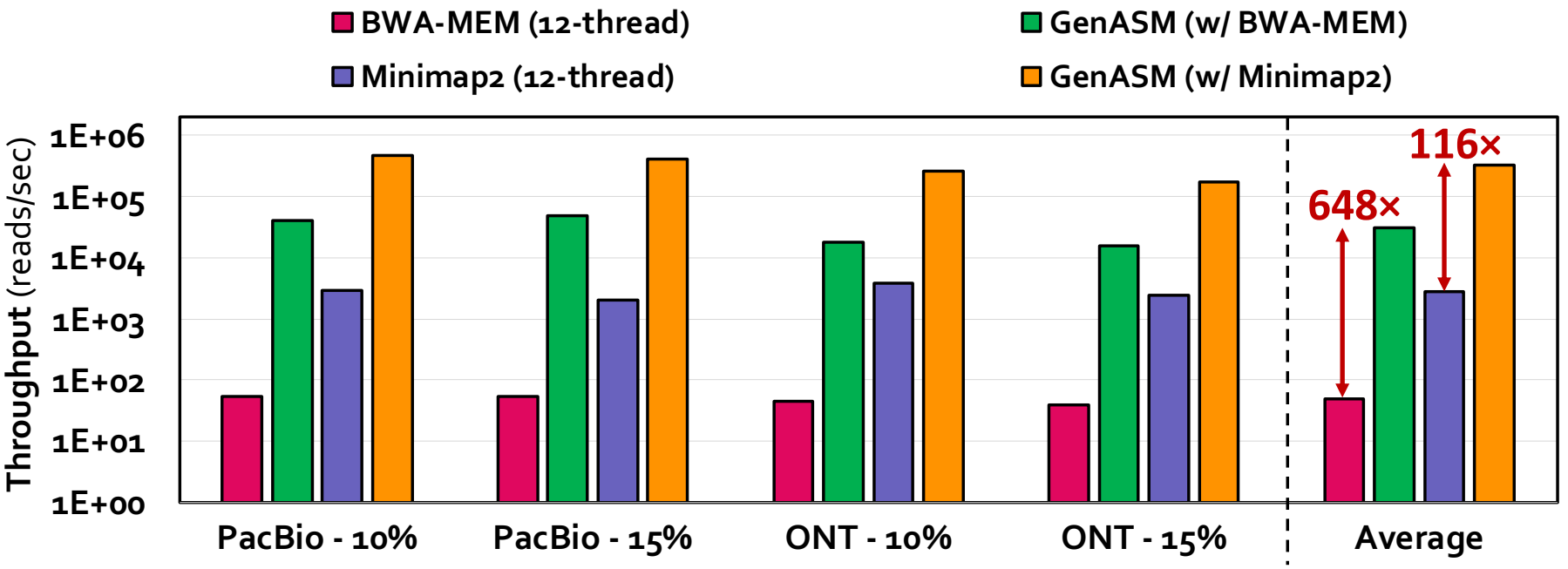
(2) Pre-Alignment Filtering for Short Reads

- 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

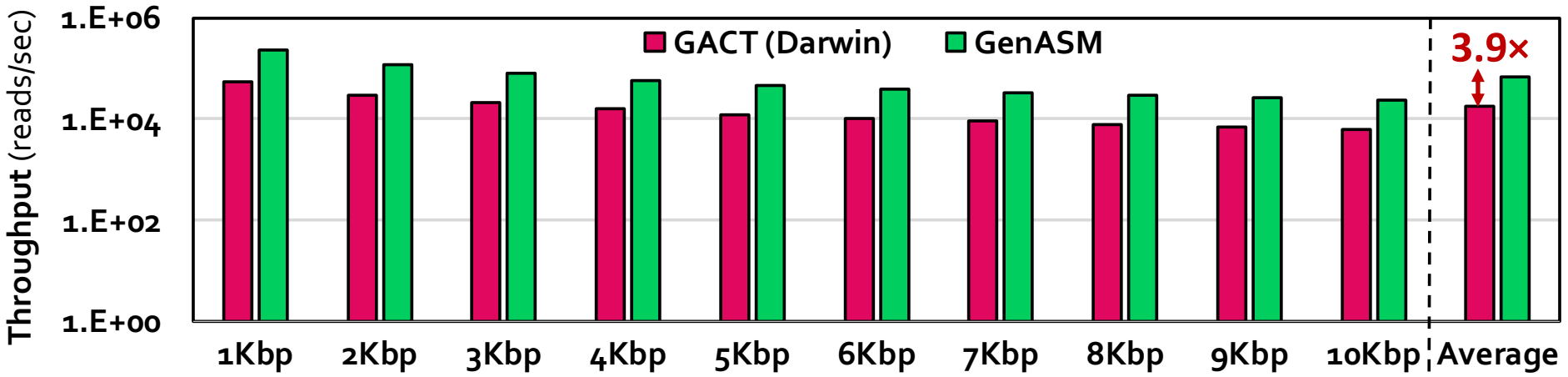
Key Results – Use Case 1 (Long Reads)



SW

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

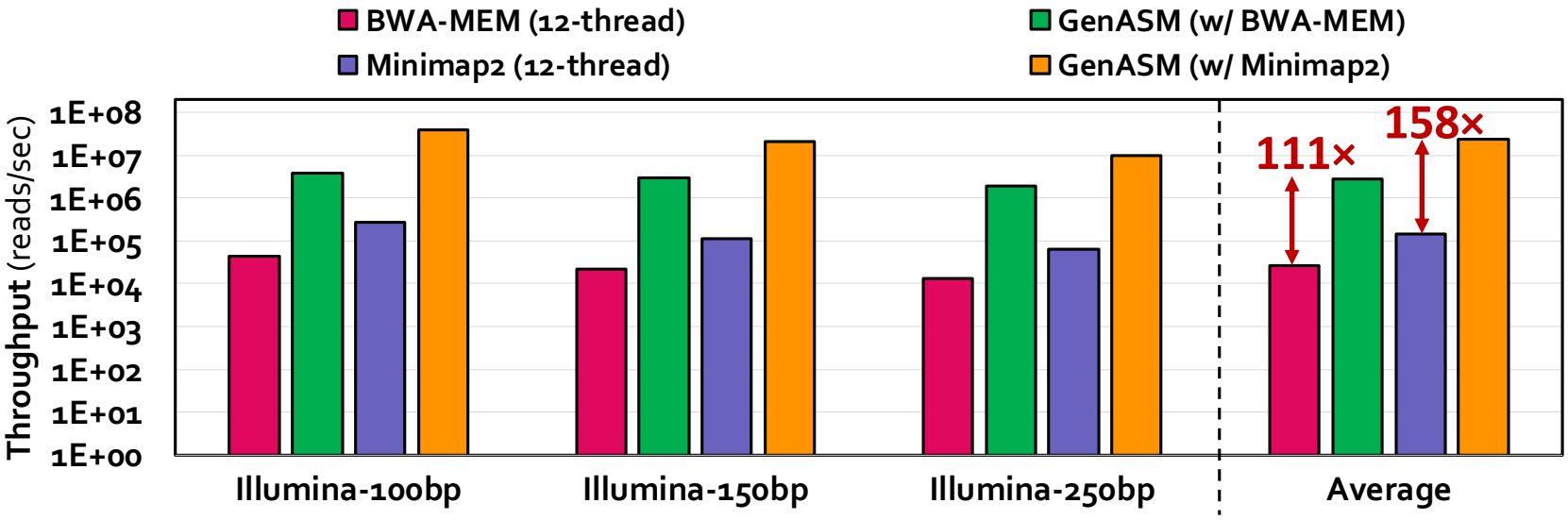
Key Results – Use Case 1 (Long Reads)



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

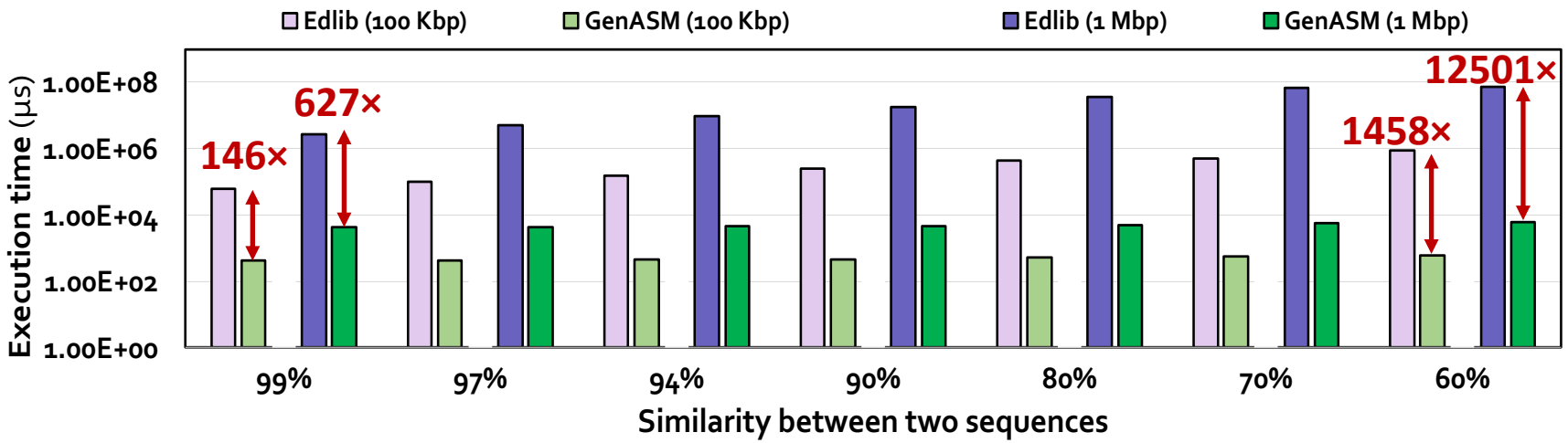
(2) Pre-Alignment Filtering for Short Reads

- 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

Outline

Bottleneck analysis of long read assembly

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Ongoing]

GenGraph: Hardware acceleration framework for sequence-to-graph mapping

[Ongoing]

BitMAc: FPGA-based GenASM

Goal: Implement and map GenASM-DC and GenASM-TB to an FPGA with HBM2 and demonstrate end-to-end application acceleration

Key Idea: HBM2 offers high bandwidth and FPGA resources offer high parallelism by instantiating multiple copies of GenASM accelerators

Key Findings:

- Due to high amount of data needs to be saved for TB, we are bottlenecked by the amount of on-chip memory we have
- We cannot saturate the high bandwidth that multiple HBM2 stacks that are on package provide
- Thus, we need
  - Algorithm-level modifications to decrease the amount of data that need to be written, and
  - New FPGA chips, which has a higher amount of on-chip memory capacity and bandwidth

Outline

Bottleneck analysis of long read assembly

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

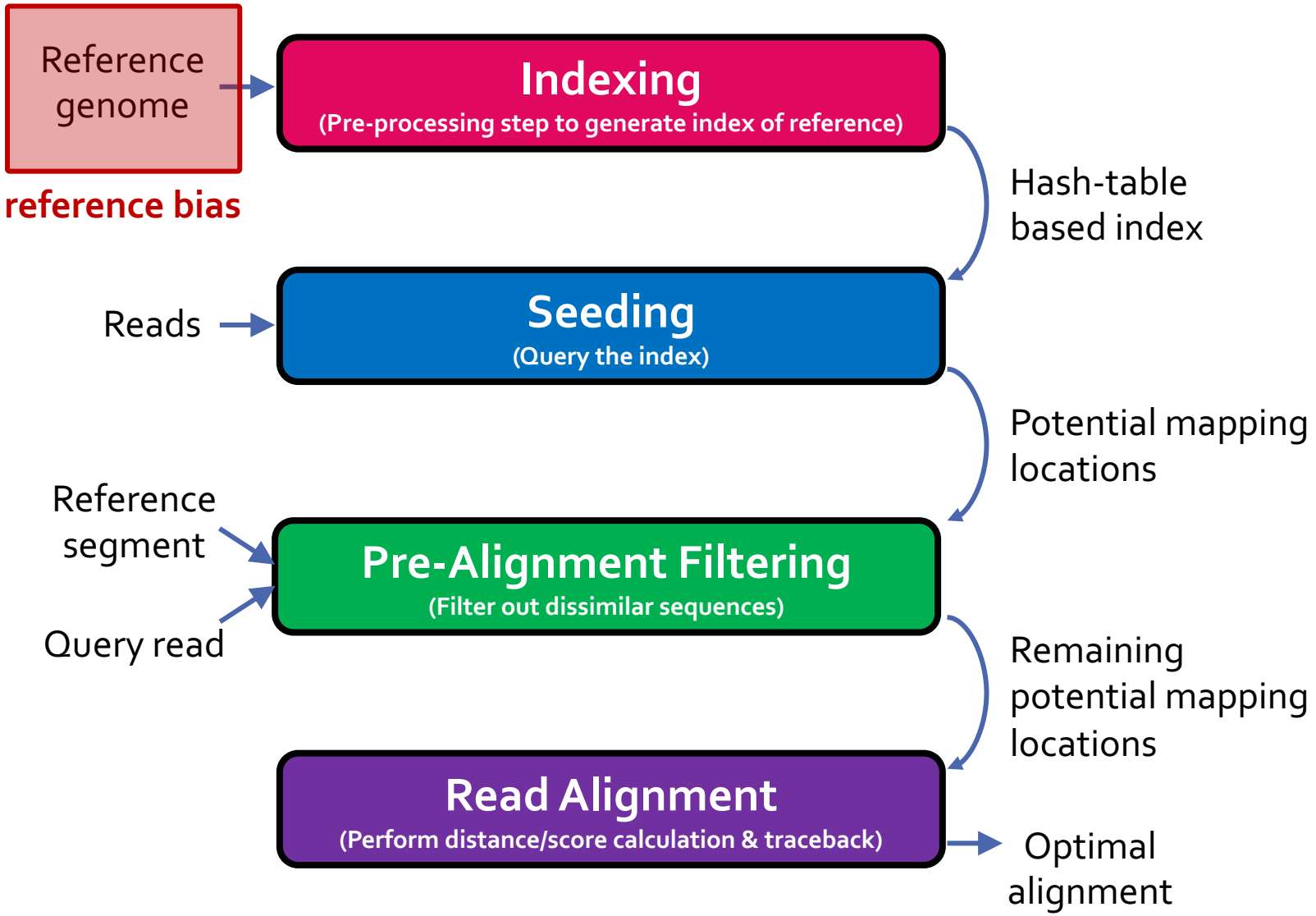
BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Ongoing]

GenGraph: Hardware acceleration framework for sequence-to-graph mapping

[Ongoing]

Recall: 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: ACGTACGT

Reference #2: ACGGACGT

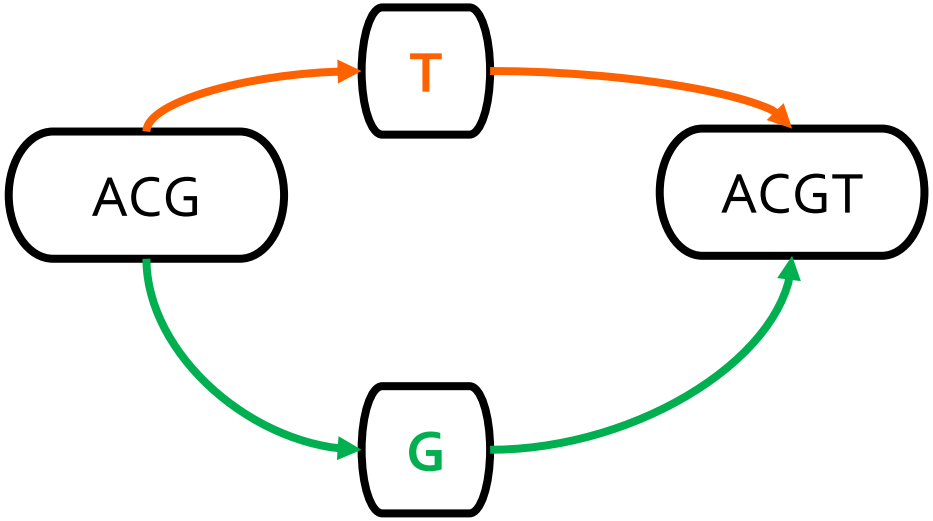
Genome Graphs

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

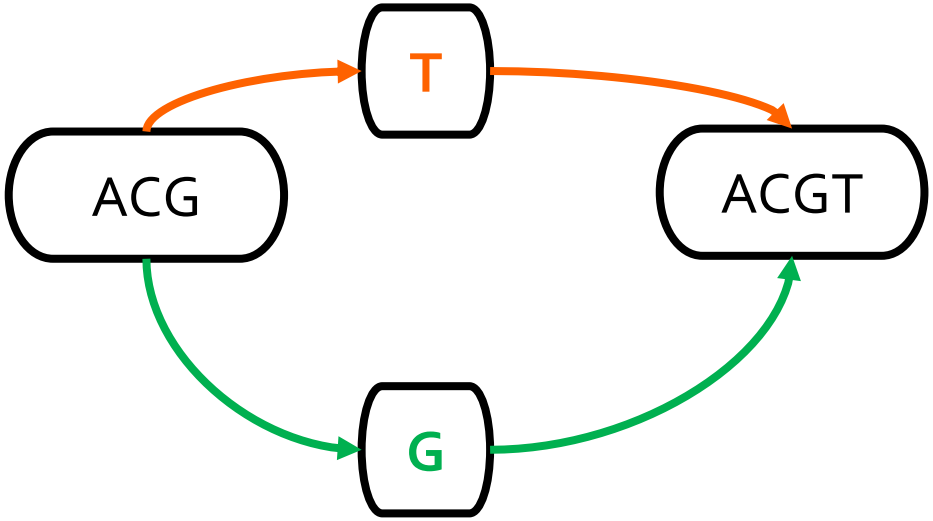
Genome graphs:

- 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

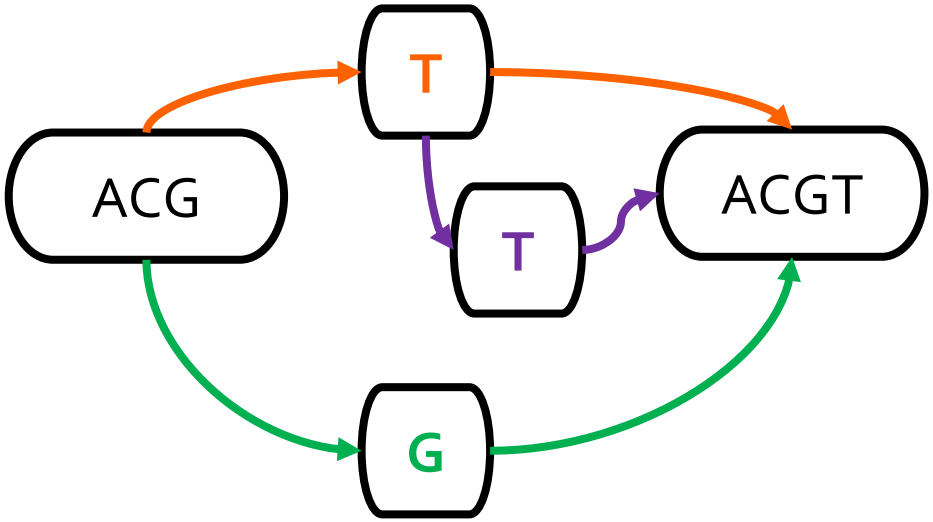
Genome graphs:

- 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:

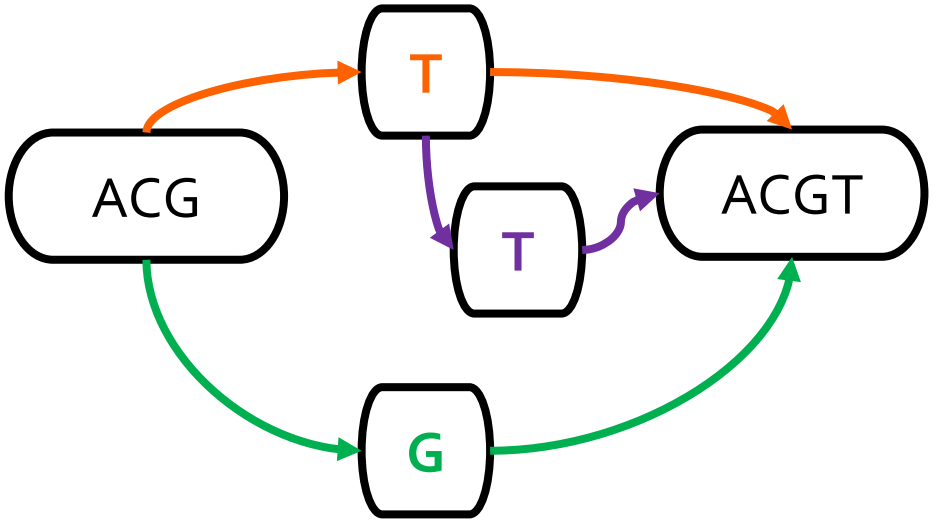
- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACGTACGT

Reference #2: ACGGACGT

Reference #3: ACGTTACGT

Reference #4: ACGACGT



Genome Graphs

Genome graphs:

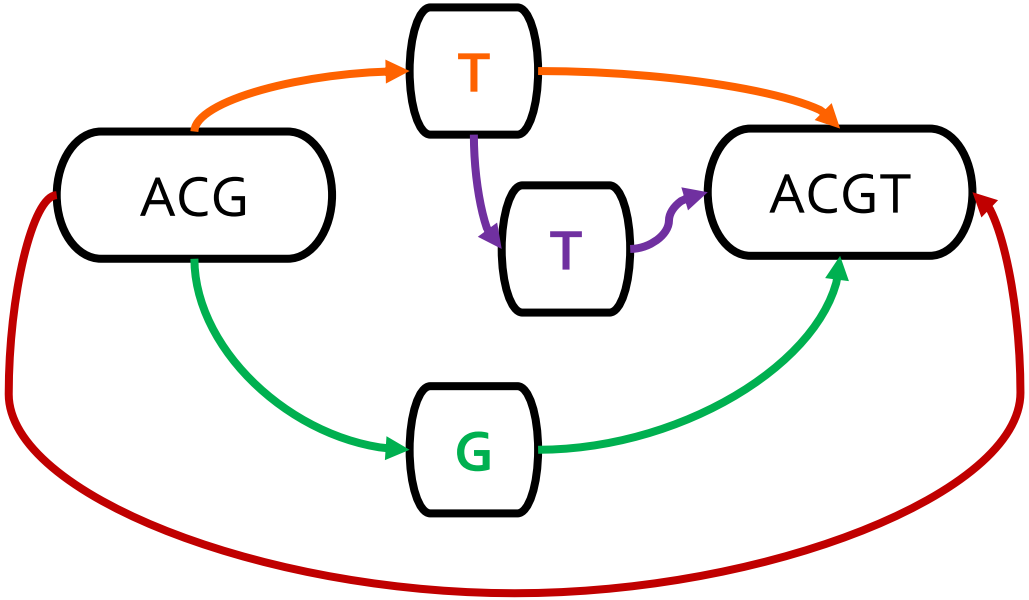
- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACGTACGT

Reference #2: ACGGACGT

Reference #3: ACGTTACGT

Reference #4: ACGACGT



GenGraph

Motivation:

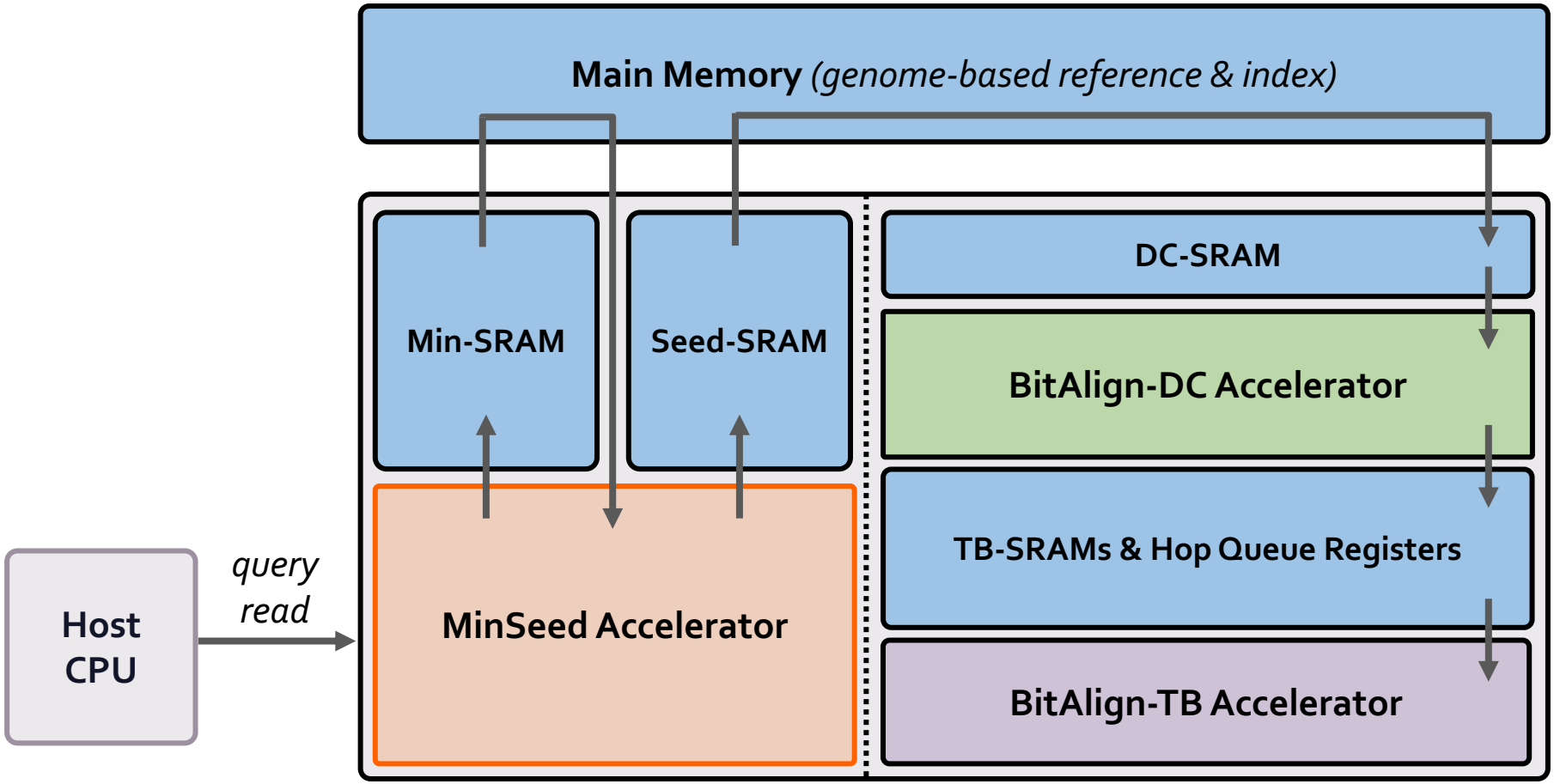
- Traditional read mapping causes reference bias
- Aligning sequences to graphs is a newer field and practical tools only start to emerge
- HW acceleration of sequence-to-graph mapping: important but unexplored research problem

Goal: Design an accelerator framework for sequence-to-graph mapping, which provides high performance and high accuracy

Our Approach:

- *BitAlign:* Modified GenASM algorithms and HW accelerators for sequence-to-graph alignment
- *MinSeed:* The first minimizer-based seeding hardware

Overview of GenGraph



MinSeed:
performs **min**imizer-based **seed**ing

BitAlign:
performs **bit**vector-based sequence-to-graph **align**ment

Outline

Bottleneck analysis of long read assembly

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Ongoing]

GenGraph: Hardware acceleration framework for sequence-to-graph mapping

[Ongoing]

Conclusion

Rapid genome sequence analysis is bottlenecked by **the computational power and memory bandwidth limitations of existing systems**, as many of the steps in genome sequence analysis must process **a large amount of data**

Conclusion (cont'd.)

Genome sequence analysis can be accelarated by co-designing **fast and efficient algorithms** along with **scalable and energy-efficient customized hardware accelerators** for the key bottleneck steps of the pipeline

Conclusion (cont'd.)

Bottleneck analysis of long read assembly

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Ongoing]

GenGraph: Hardware acceleration framework for sequence-to-graph mapping

[Ongoing]

Other Publications in CMU

FPGA-based Near-Memory Acceleration of Modern Data-Intensive Applications (IEEE Micro, 2021)

Gagandeep Singh, Mohammed Alser, Damla Senol Cali, Dionysios Diamantopoulos, Juan Gomez-Luna, Henk Corporaal, and Onur Mutlu.

Accelerating Genome Analysis: A Primer on an Ongoing Journey (IEEE Micro, 2020)

Mohammed Alser, Zulal Bingol, Damla Senol Cali, Jeremie S. Kim, Saugata Ghose, Can Alkan, and Onur Mutlu.

Apollo: A Sequencing-Technology-Independent, Scalable, and Accurate Assembly Polishing Algorithm (Bioinformatics, 2020)

Can Firtina, Jeremie S. Kim, Mohammed Alser, Damla Senol Cali, A. Ercument Cicek, Can Alkan, and Onur Mutlu.

Demystifying Workload–DRAM Interactions: An Experimental Study (ACM SIGMETRICS, 2019)

Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu.

GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies (BMC Genomics, 2018)

Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu.

Carnegie Mellon

SAFARI

Accelerating Genome Sequence Analysis via Efficient Hardware/Algorithm Co-Design

Damla Senol Cali

Carnegie Mellon University

(dsenol@andrew.cmu.edu)

June 24, 2021

Job Talk