

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

Damla Senol Cali

Staff Software Engineer, Hardware Acceleration
Bionano Genomics

Email: damlasenolcali@gmail.com

Website: <https://damlasenolcali.github.io>

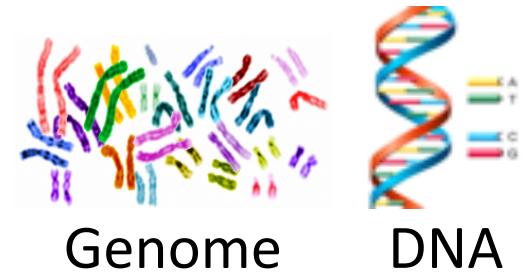
SAFARI Live Seminar

November 7, 2021

Genome Sequencing

- **Genome sequencing:** Enables us to determine the order of the DNA sequence in an organism's genome

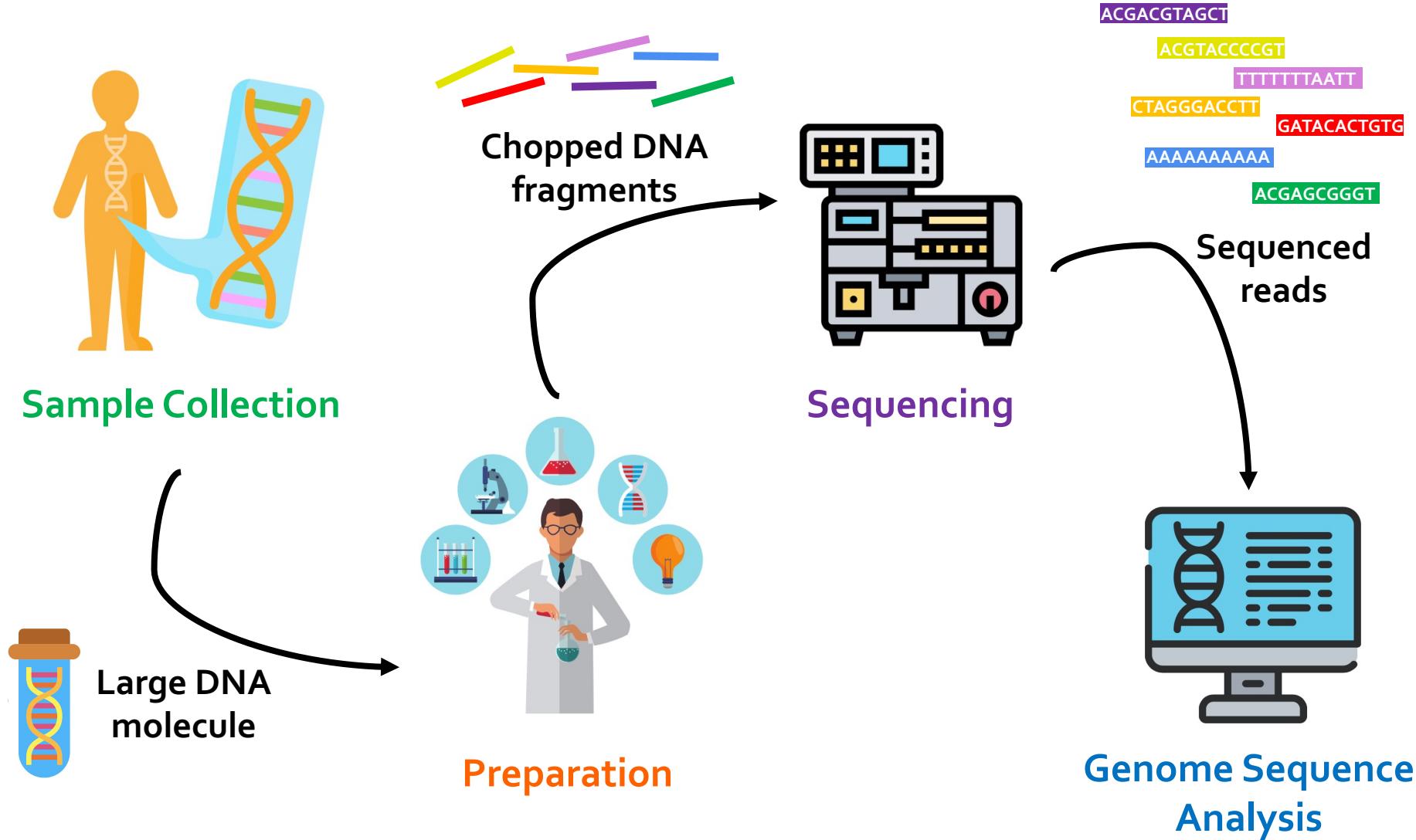
- Plays a **pivotal role** in:
 - Personalized medicine
 - Outbreak tracing
 - Understanding of evolution



- **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.)



Sequencing Technologies



Oxford Nanopore
(ONT)



PacBio

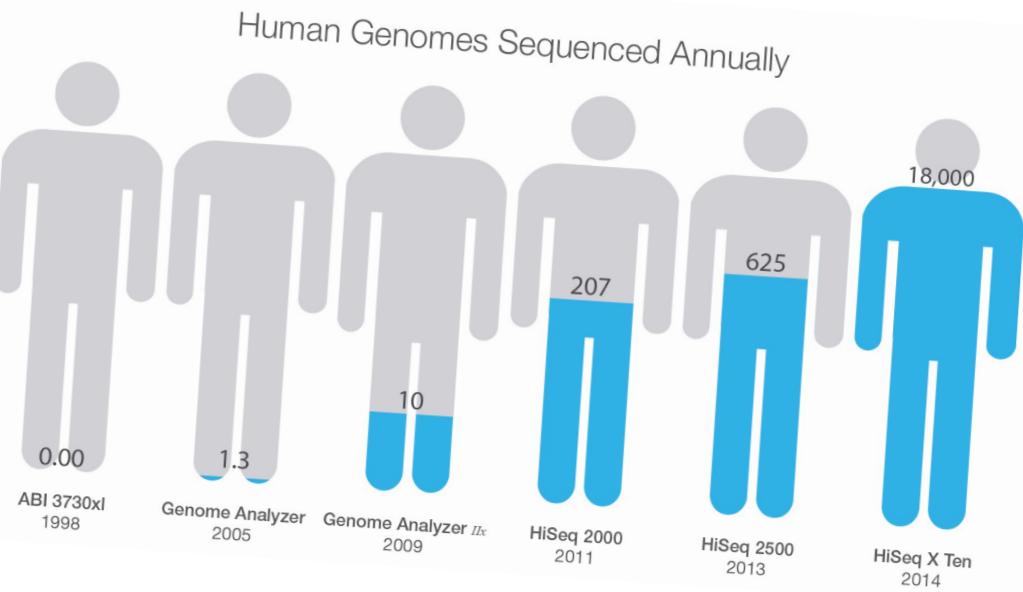
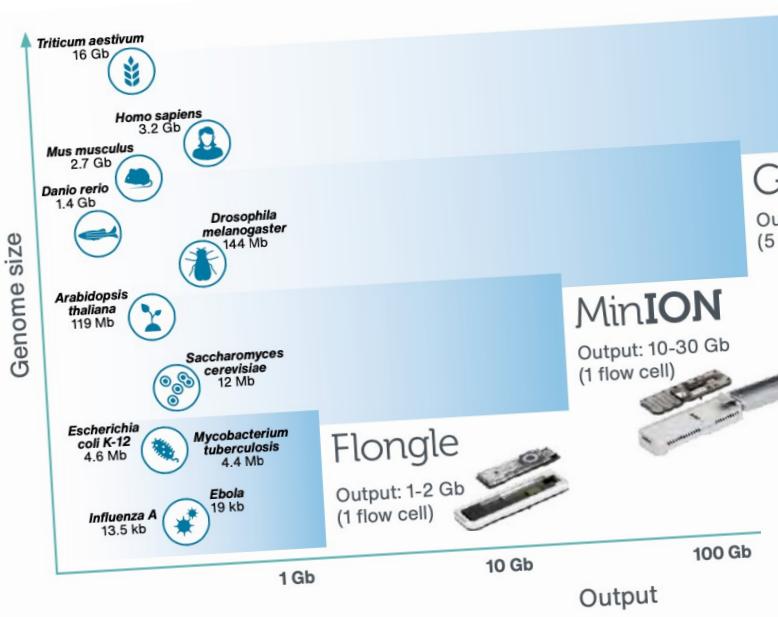
Illumina



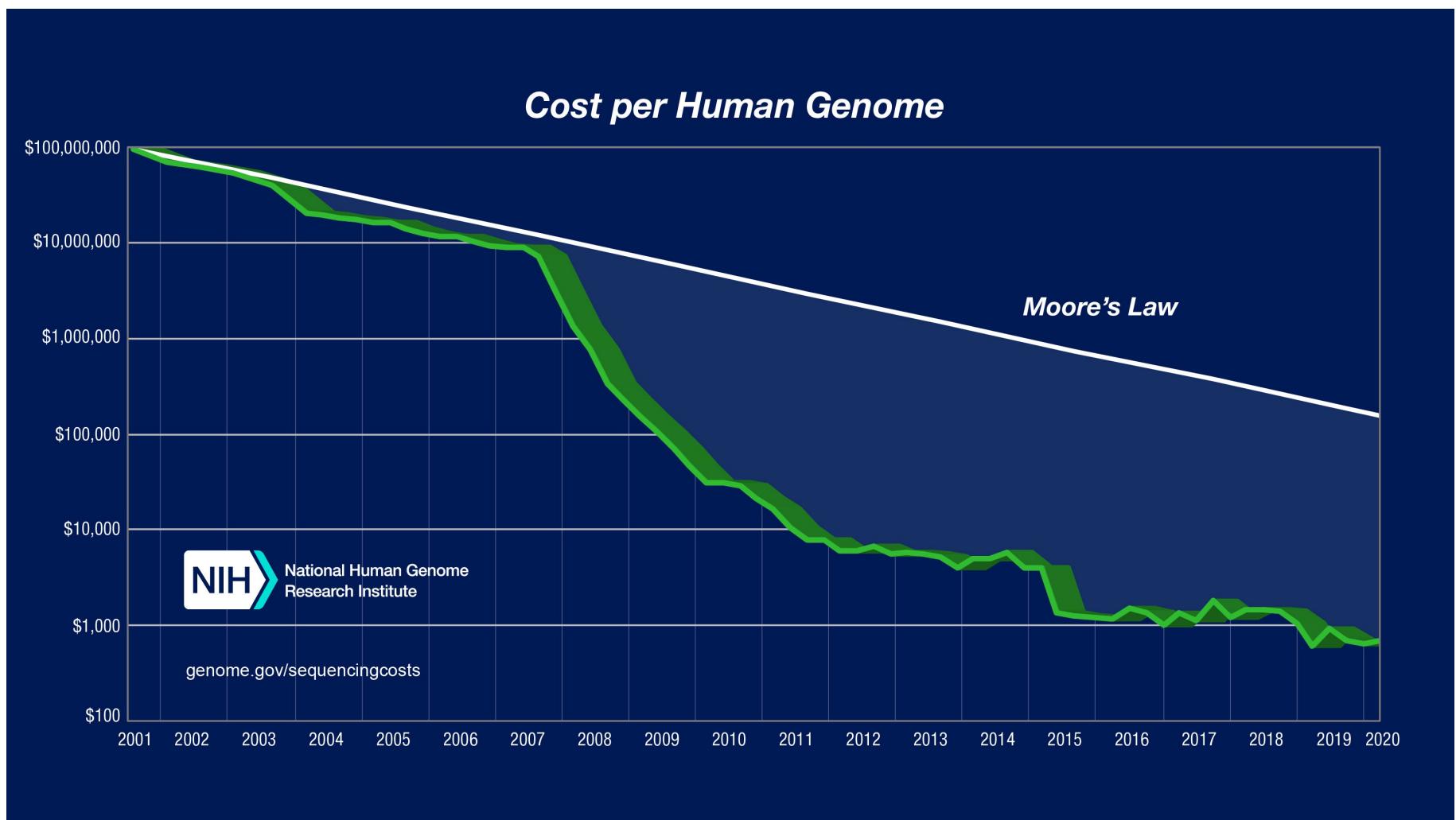
***Short reads:* a few hundred base pairs and error rate of ~0.1%**

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

Current State of Sequencing

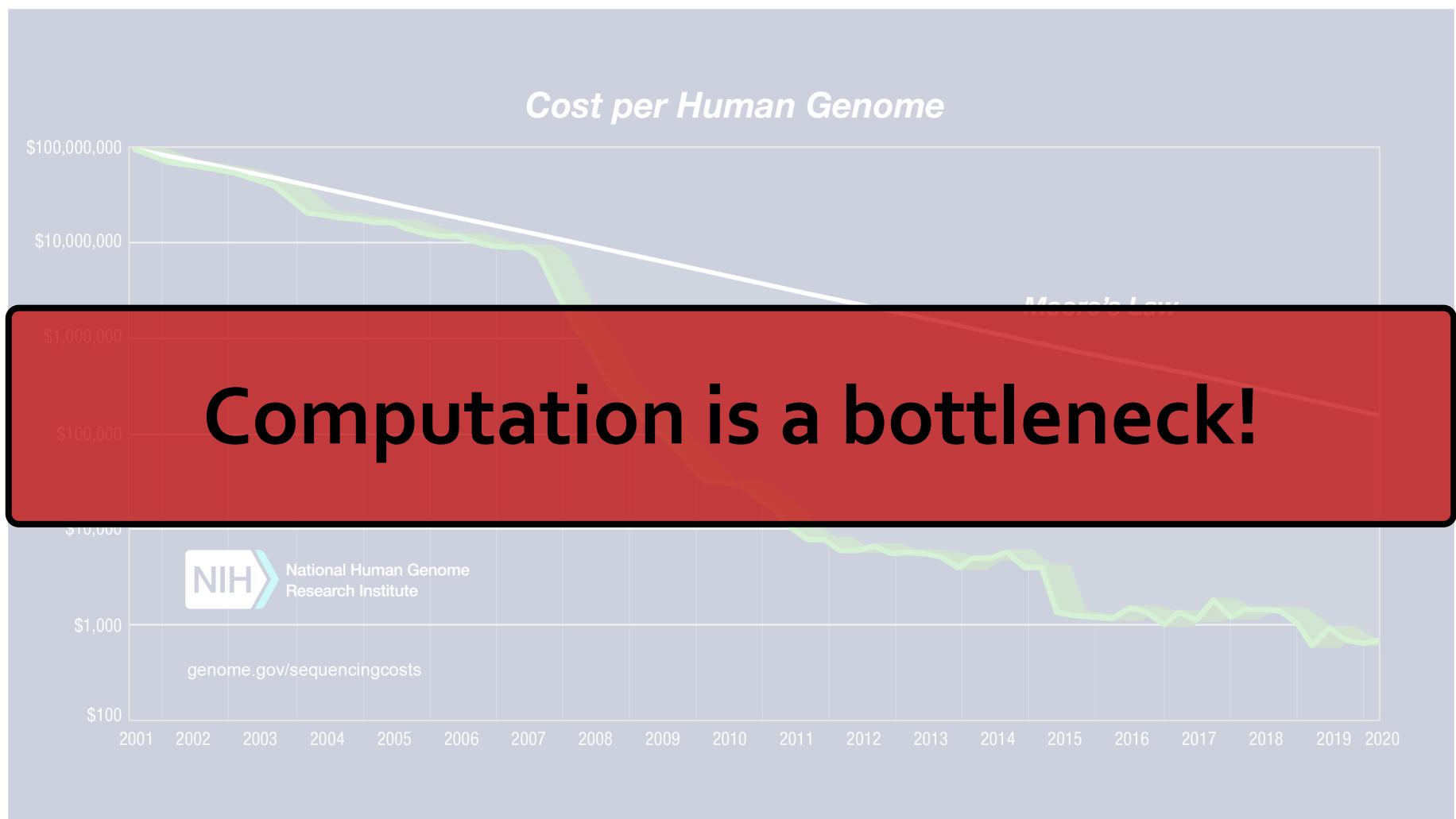


Current State of Sequencing (cont'd.)



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

Current State of Sequencing (cont'd.)



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

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

Research Statement

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

Research Contributions

Bottleneck analysis of genome assembly pipeline for long reads

[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]

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

[Ongoing]

Research Contributions

Bottleneck analysis of genome assembly pipeline for long reads

[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]

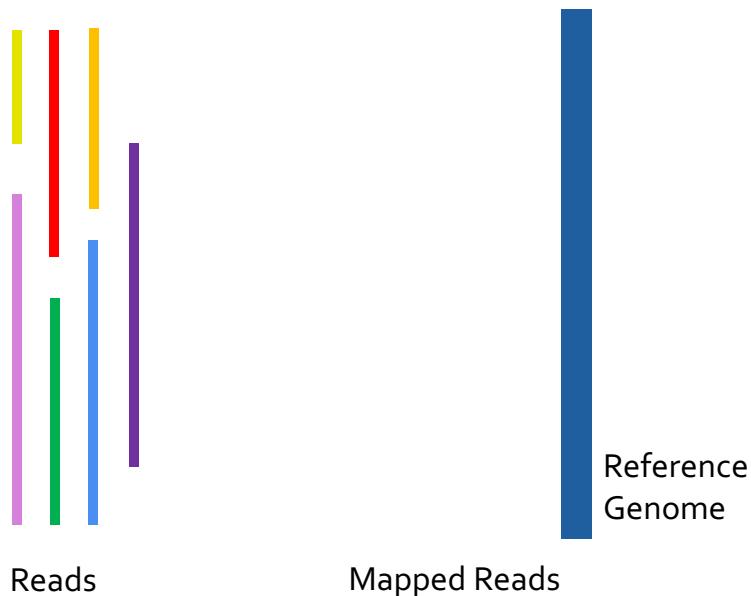
SeGraM: Hardware acceleration framework for sequence-to-graph mapping

[Ongoing]

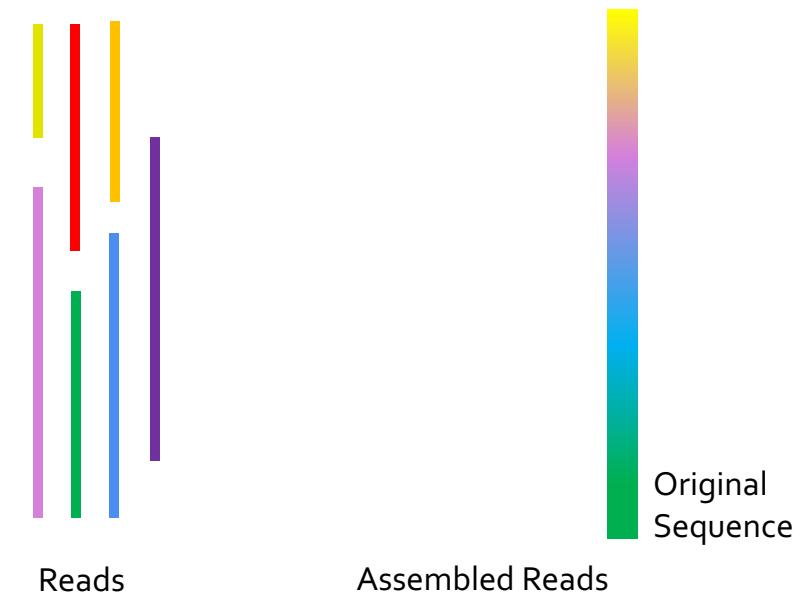
Genome Sequence Analysis



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

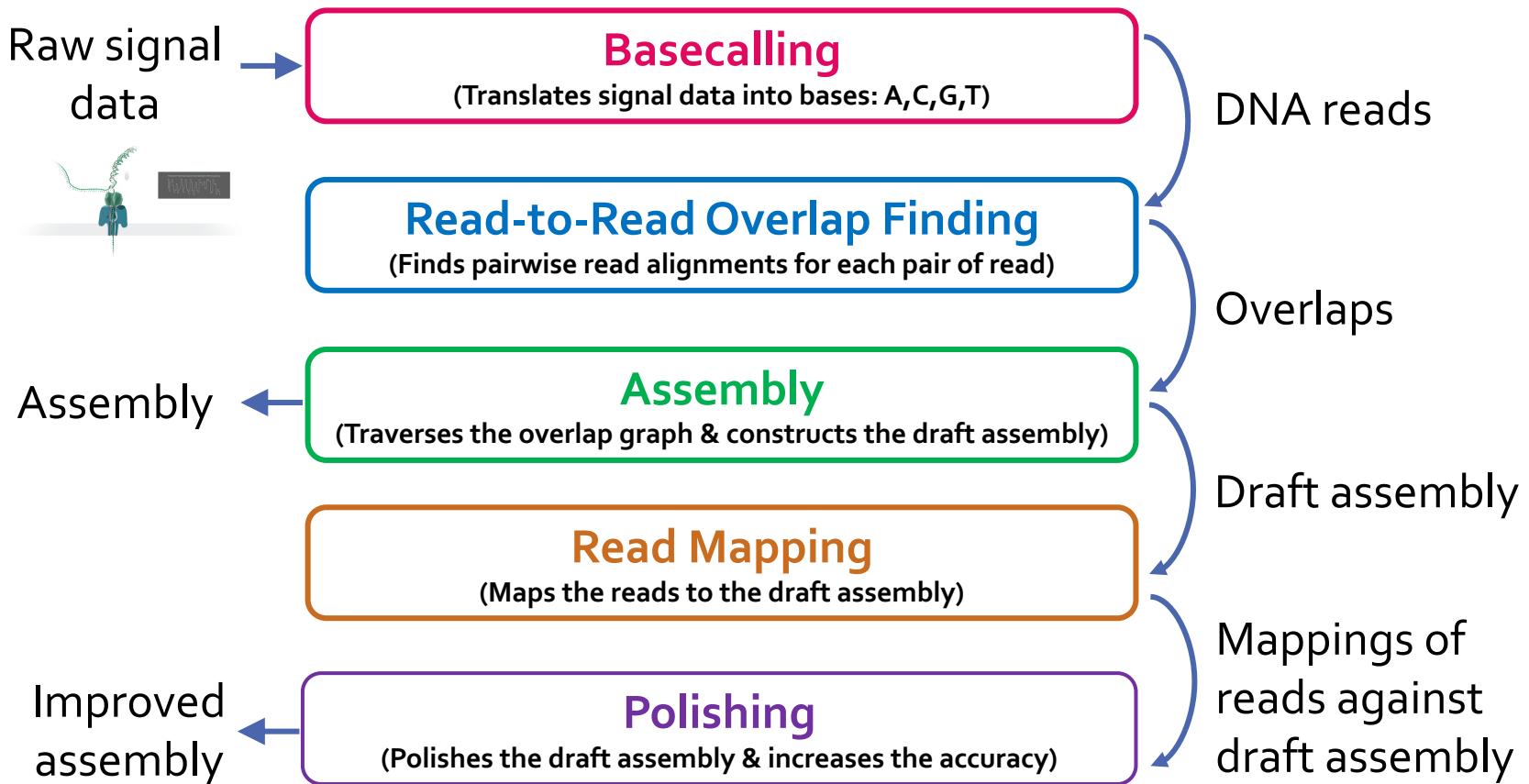


De novo Assembly, method of merging the reads in order to **construct** the original sequence.



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**

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

Nanopore Sequencing & Tools [BiB 2018]

Damla Senol Cali, Jeremie S. Kim, Saugata Ghose, Can Alkan, and Onur Mutlu,
"Nanopore Sequencing Technology and Tools for Genome Assembly:
Computational Analysis of the Current State, Bottlenecks and Future
Directions"

Briefings in Bioinformatics, April 2018.

Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions

**Damla Senol Cali^{1,*}, Jeremie S. Kim^{1,3}, Saugata Ghose¹, Can Alkan^{2*}
and Onur Mutlu^{3,1*}**

¹Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA

²Department of Computer Engineering, Bilkent University, Bilkent, Ankara, Turkey

³Department of Computer Science, Systems Group, ETH Zürich, Zürich, Switzerland

Research Contributions

Bottleneck analysis of genome assembly pipeline for long reads

[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]

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

[Ongoing]

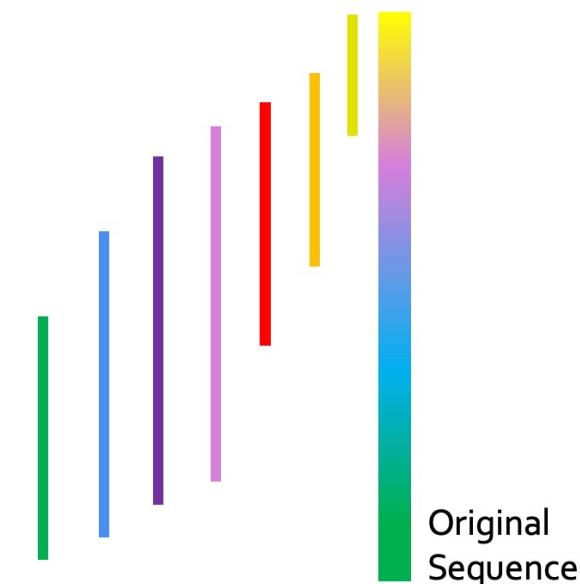
Recall: Genome Sequence Analysis



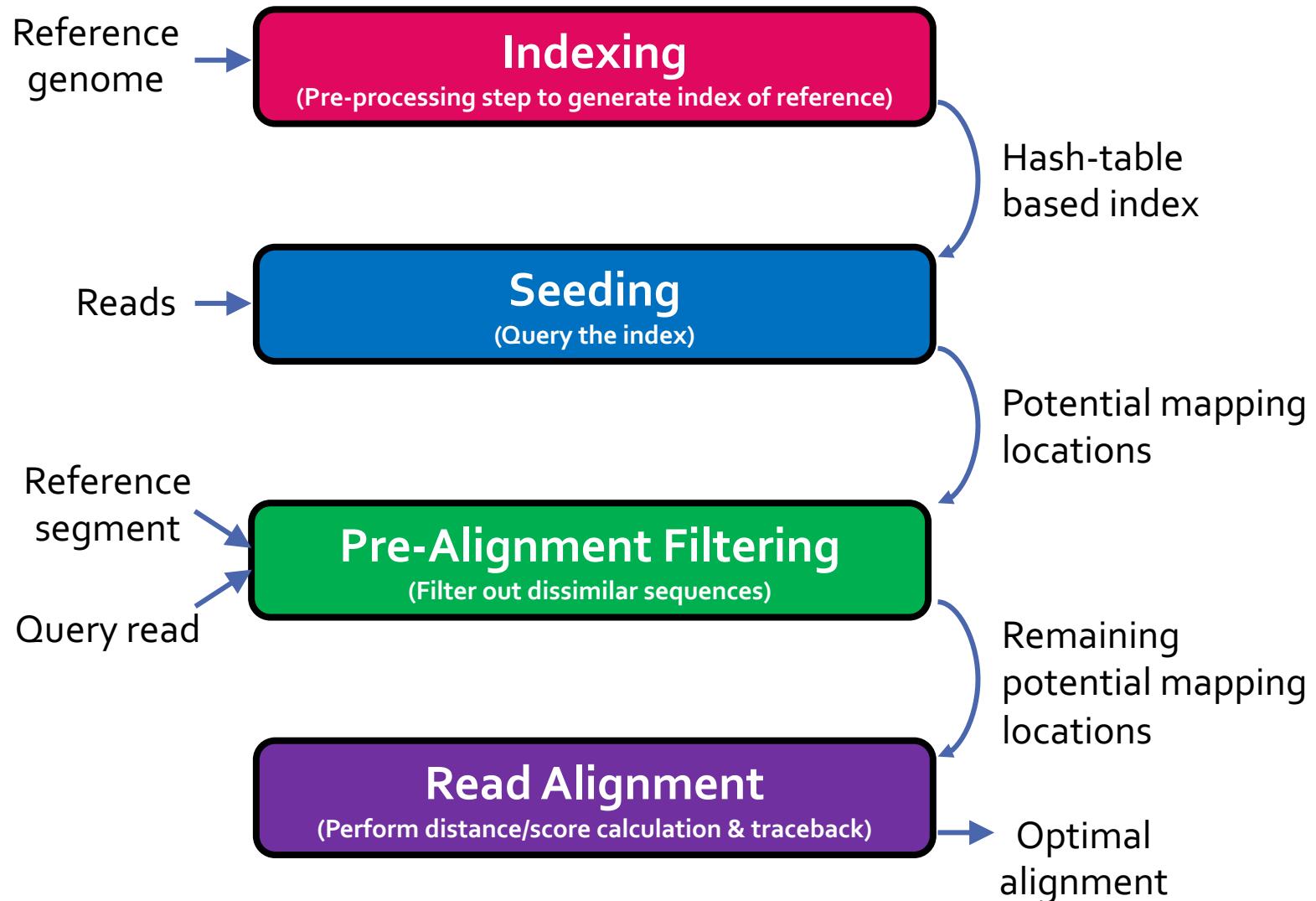
Read Mapping, method of aligning the reads against the reference genome in order to **detect matches and variations**.



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**

Reference: AAA T G T T T A T G C T A C T G
Read: AAA T G T T T A T G C T A C T G

deletion *substitution* *insertion*

- **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	
C	0	0	0	0	0	0	0	0	0	0	0
C	0	2	2	2	2	2	2	2	2	2	2
T	0	2	3	3	3	3	3	3	4	4	4
T	0	2	3	5	5	5	5	5	5	6	6
A	0	2	3	5	7	7	7	7	7	7	7
G	0	3	3	5	7	9	9	9	9	9	9
T	0	2	4	5	7	9	11	11	11	11	11
A	0	2	4	6	7	9	11	13	13	13	13
T	0	2	4	6	7	9	11	13	14	14	15
	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

$R[0] = (\text{oldR}[0] \ll 1) | \text{PM}[\text{char}]$

 For $d = 1 \dots k$:

 deletion = oldR[d-1]

 substitution = oldR[d-1] << 1

 insertion = R[d-1] << 1

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

$R[d] = \text{deletion} \& \text{mismatch} \& \text{insertion} \& \text{match}$

 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

$R[0] = (\text{oldR}[0] \ll 1) | \text{PM}[\text{char}]$

For $d = 1 \dots k$:

deletion $= \text{oldR}[d-1]$

substitution $= \text{oldR}[d-1] \ll 1$

insertion $= R[d-1] \ll 1$

match $= (\text{oldR}[d] \ll 1) | \text{PM}[\text{char}]$

$R[d] = \text{deletion} \& \text{mismatch} \& \text{insertion} \& \text{match}$

Check MSB of $R[d]$:

If 1, no match.

If 0, match with d many errors.

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

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

$R[0] = (\text{oldR}[0] \ll 1) | \text{PM}[\text{char}]$

For $d = 1 \dots k$:

deletion $= \text{oldR}[d-1]$

substitution $= \text{oldR}[d-1] \ll 1$

insertion $= R[d-1] \ll 1$

match $= (\text{oldR}[d] \ll 1) | \text{PM}[\text{char}]$

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

$R[d] = \text{deletion} \& \text{mismatch} \& \text{insertion} \& \text{match}$

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

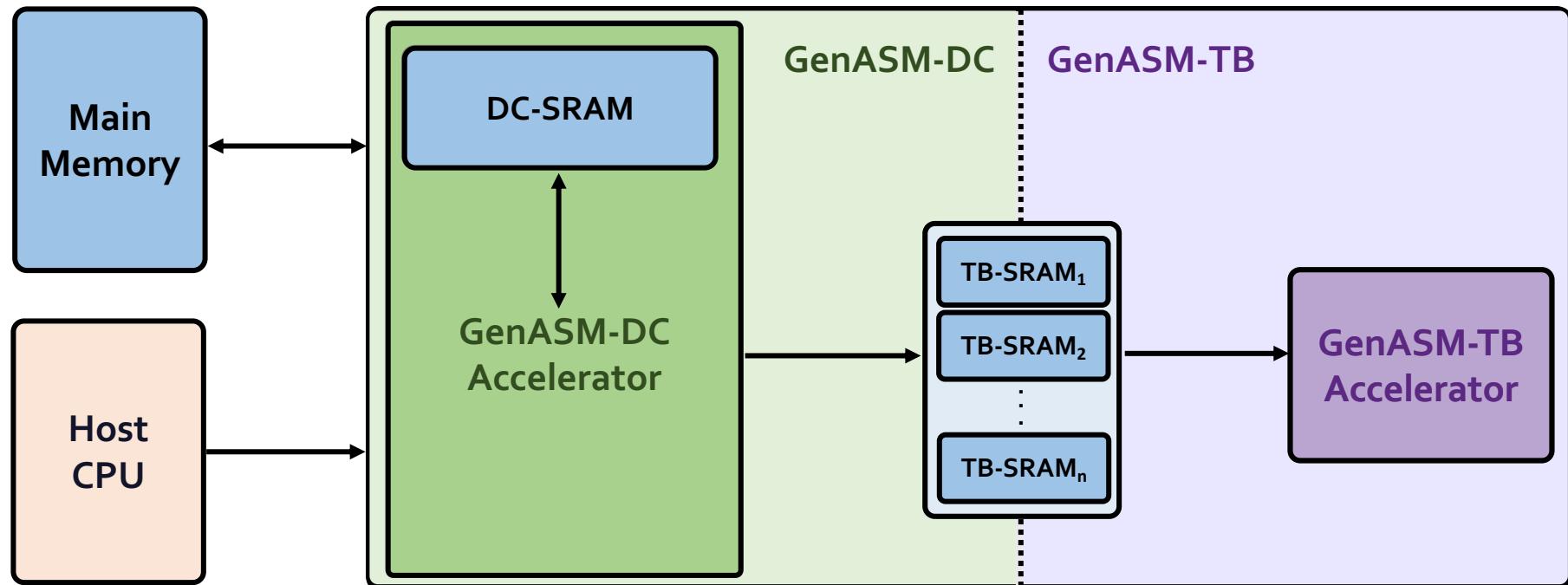
SW

- 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

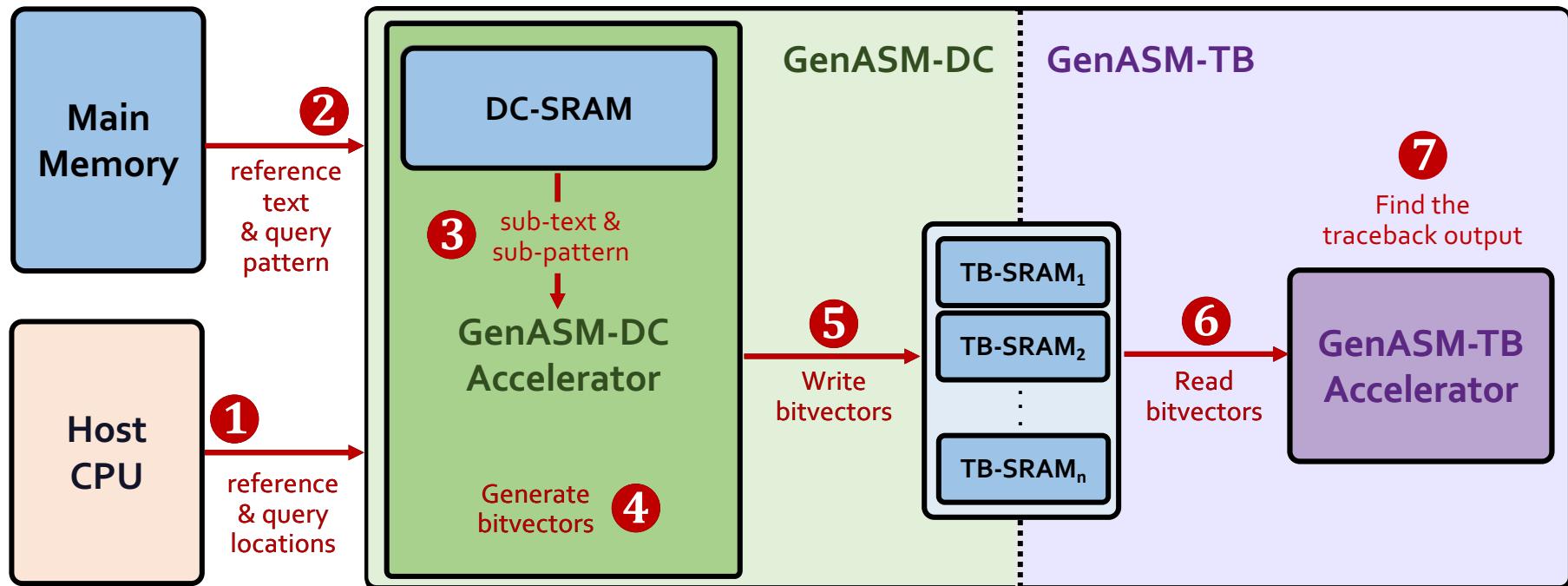
GenASM Hardware Design



GenASM-DC:
generates bitvectors
and performs edit
Distance Calculation

GenASM-TB:
performs TraceBack
and assembles the
optimal alignment

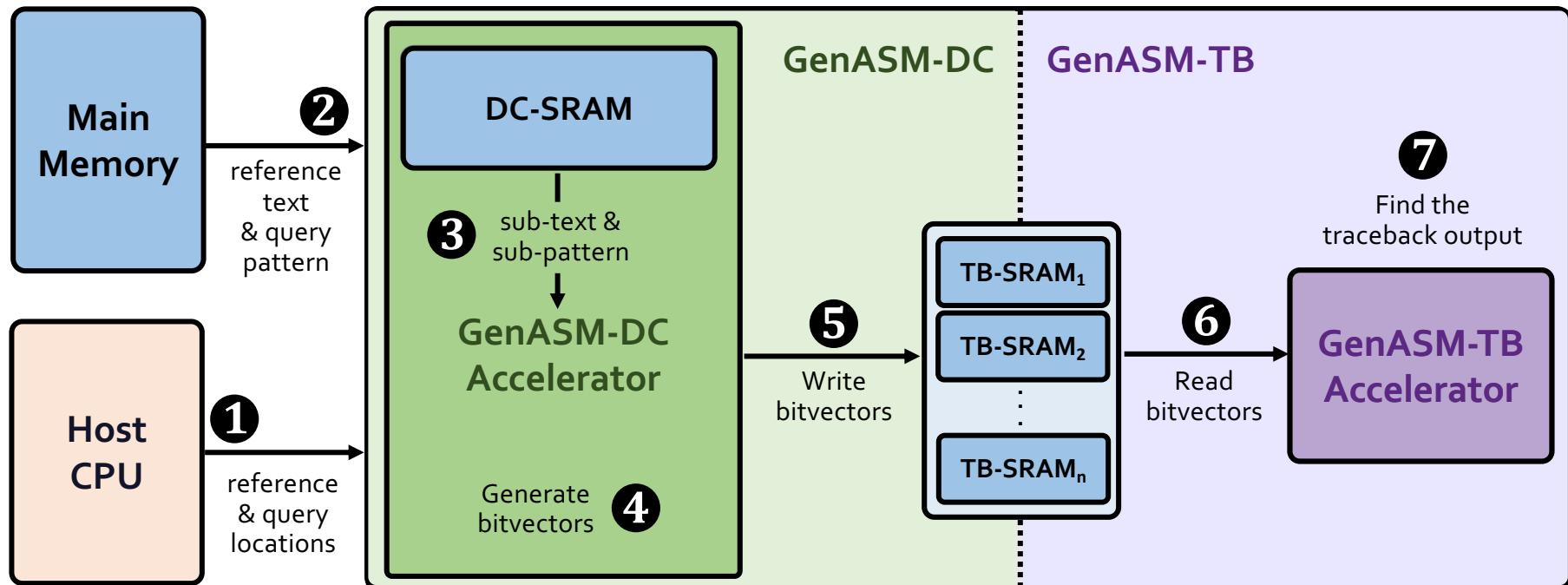
GenASM Hardware Design



GenASM-DC:
generates bitvectors
and performs edit
Distance Calculation

GenASM-TB:
performs TraceBack
and assembles the
optimal alignment

GenASM Hardware Design

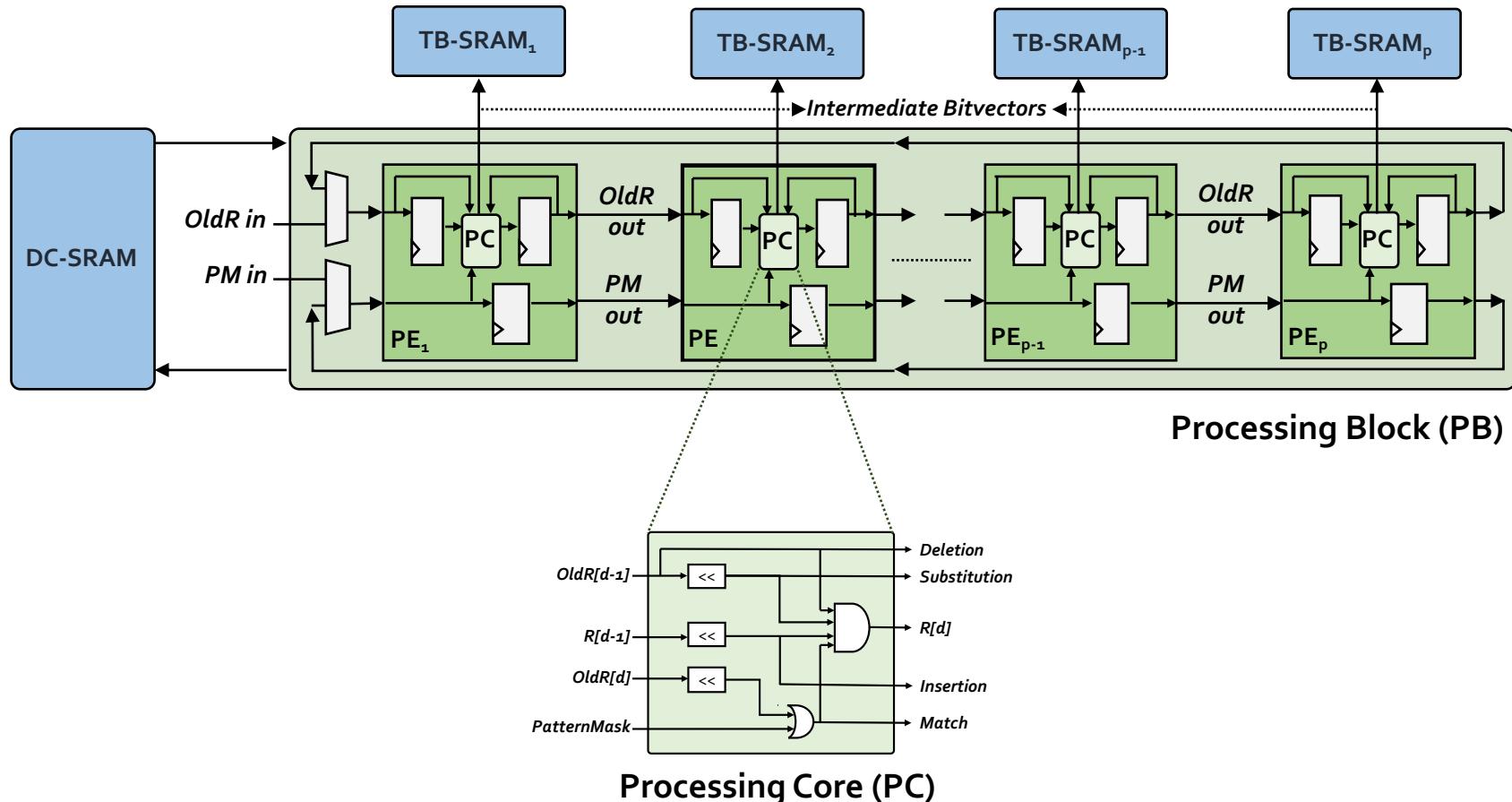


Our *specialized compute units* and *on-chip SRAMs* help us to:

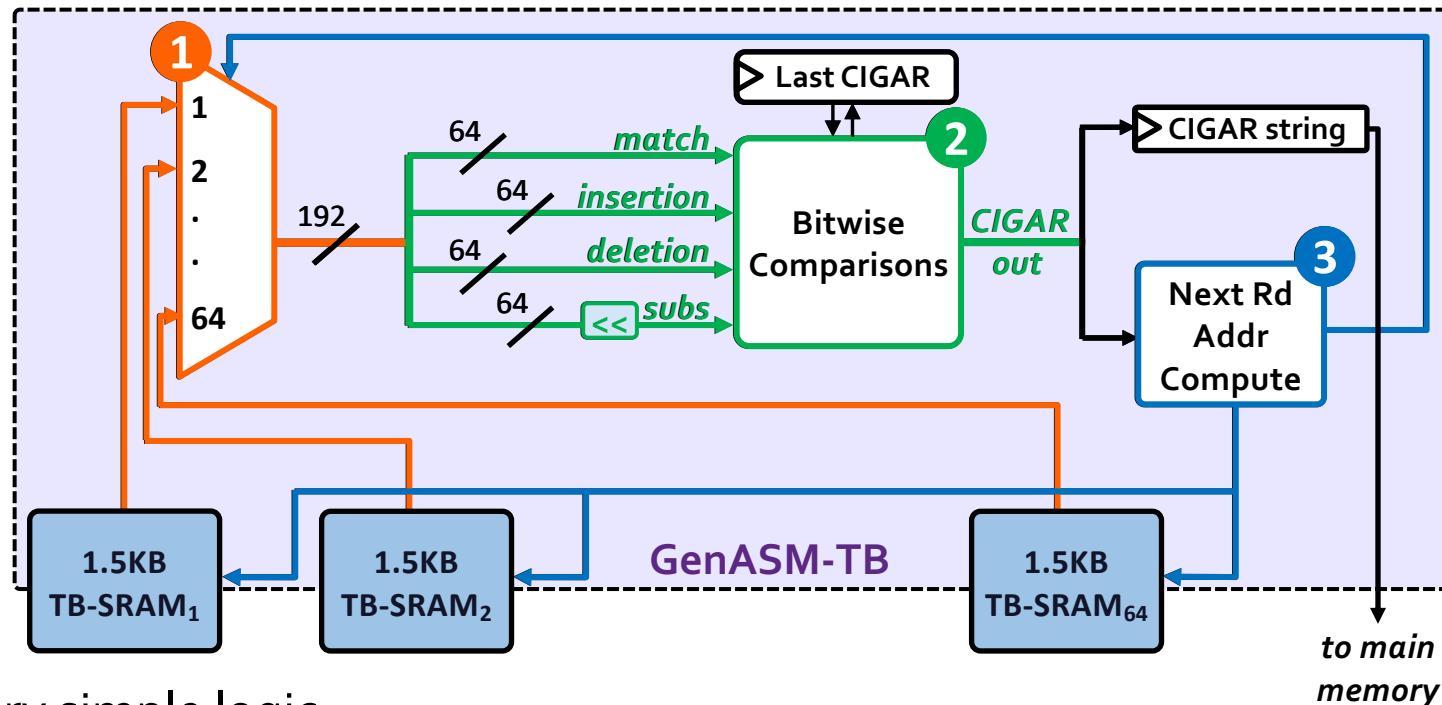
- Match **the rate of computation** with **memory capacity and bandwidth**
 - Achieve **high performance and power efficiency**
 - Scale **linearly in performance** with
the number of parallel compute units that we add to the system

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

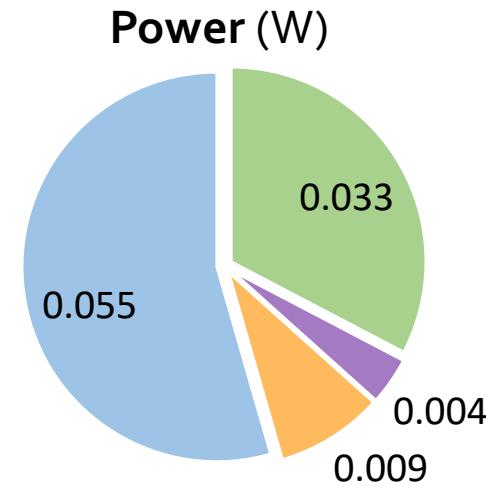
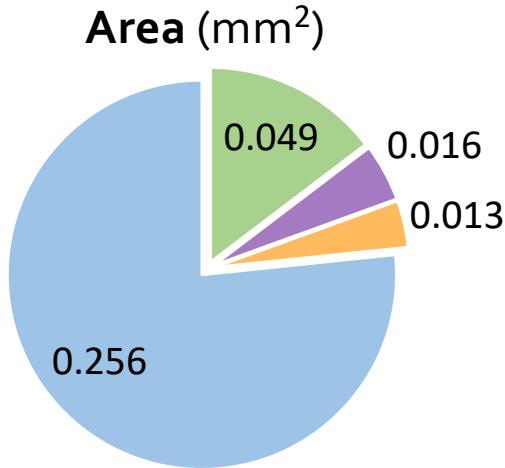
- For Use Case 3: Edit Distance Calculation, we compare GenASM with:
 - Edlib (state-of-the-art SW)
 - Using two 100Kbp and 1Mbp sequences with similarity ranging between 60%-99%

 - 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**

- GenASM-DC (64 PEs)
- GenASM-TB
- DC-SRAM (8 KB)
- TB-SRAMs (64 x 1.5 KB)

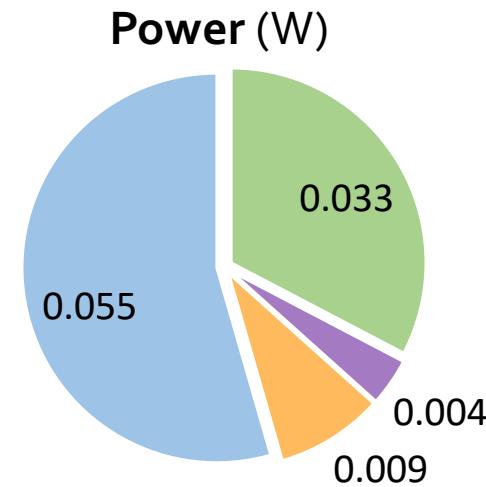
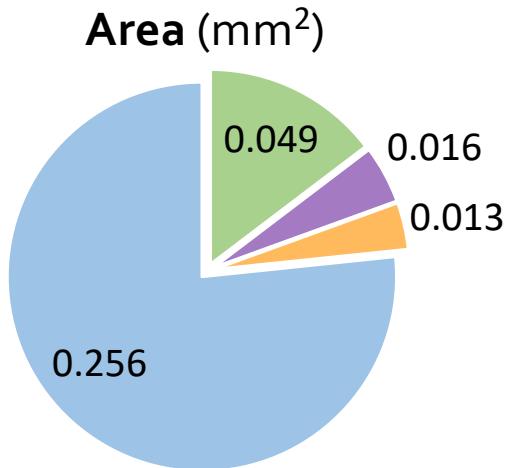


Total (1 vault):	0.334 mm ²	0.101 W
Total (32 vaults):	10.69 mm ²	3.23 W
% of a Xeon CPU core:	1%	1%

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-DC (64 PEs)
- GenASM-TB
- DC-SRAM (8 KB)
- TB-SRAMs (64 x 1.5 KB)



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

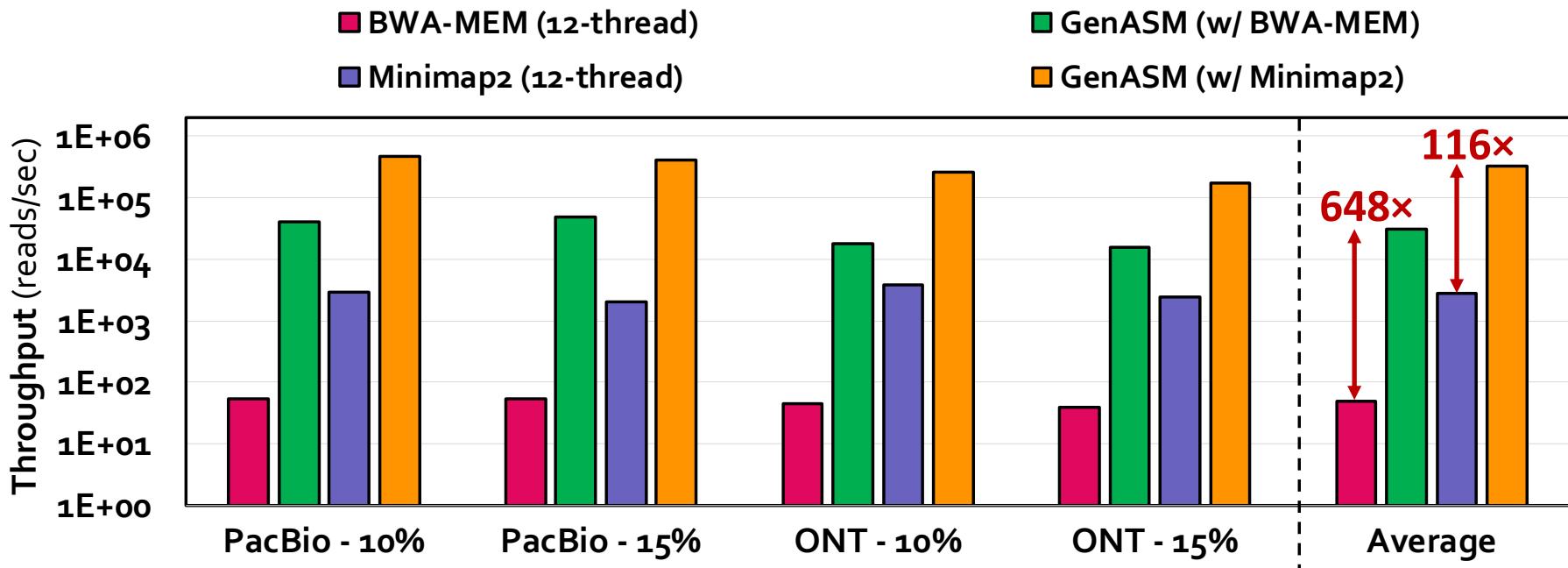
(2) Pre-Alignment Filtering for Short Reads

- 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

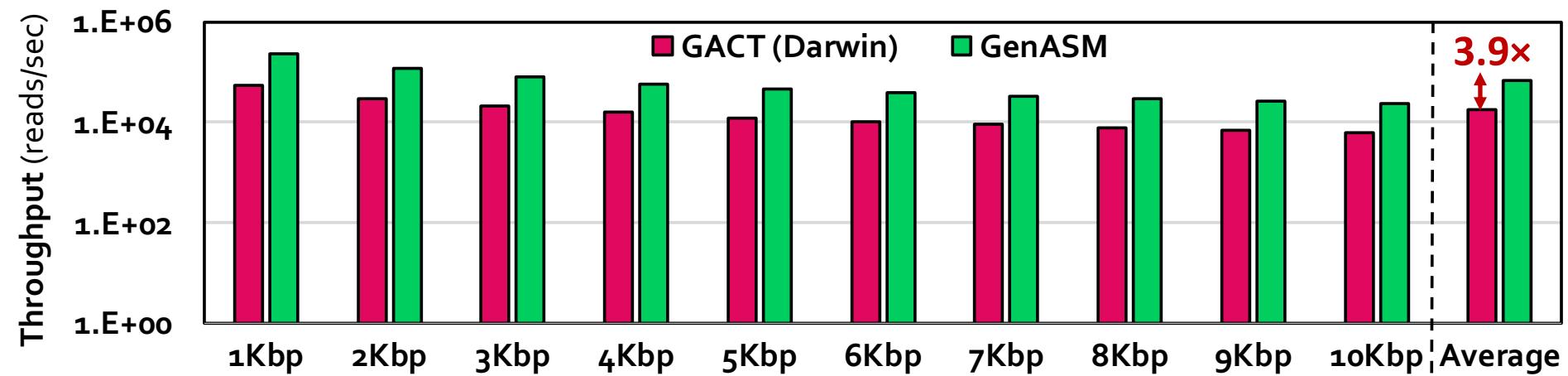
Key Results – Use Case 1 (Long Reads)



SW

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

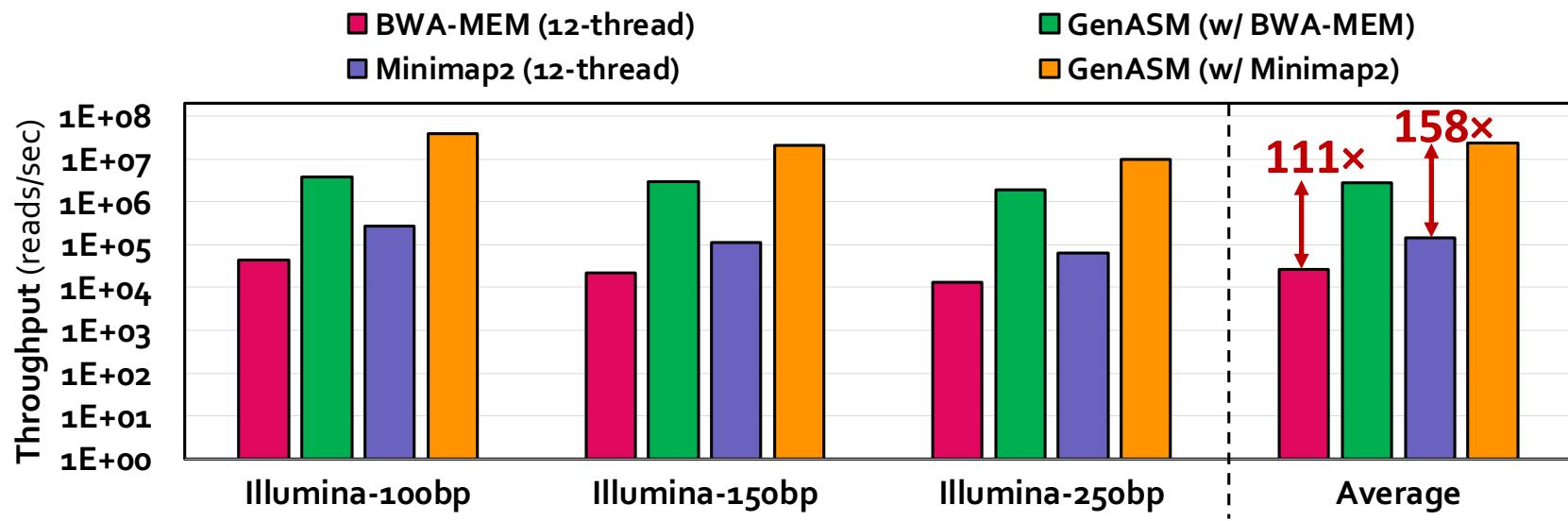
Key Results – Use Case 1 (Long Reads)



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 \times** and **158 \times** speedup over
12-thread runs of BWA-MEM and Minimap2,
while **reducing power consumption by 33 \times and 31 \times**

HW

GenASM provides **1.9 \times** 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:
 - **3.7x** speedup
 - **1.7x** 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

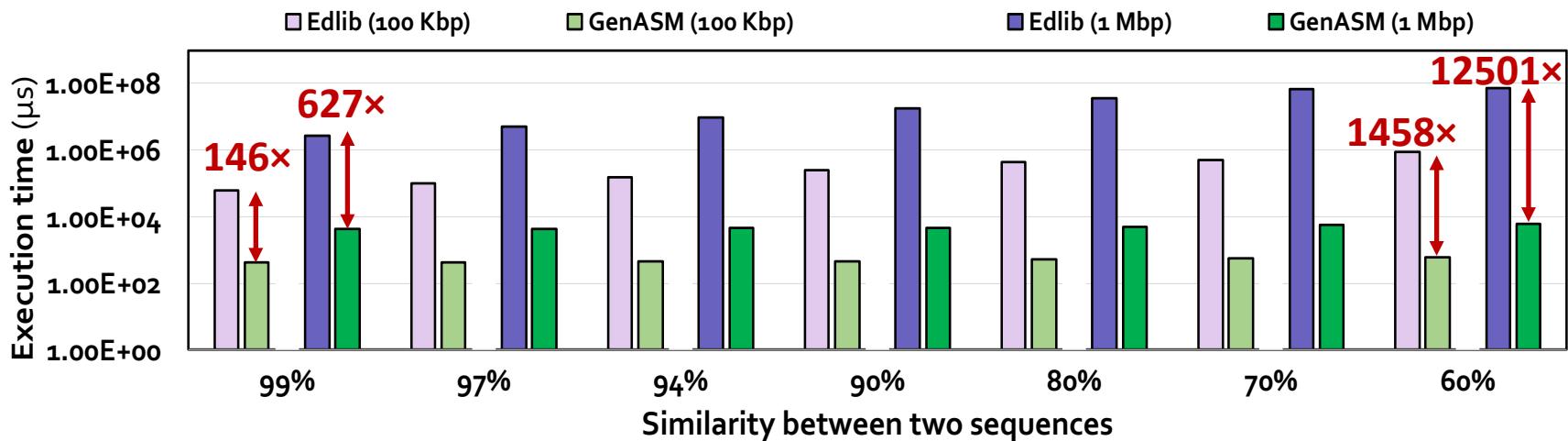
(2) Pre-Alignment Filtering for Short Reads

- 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× 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

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*

Research Contributions

Bottleneck analysis of genome assembly pipeline for long reads

[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]

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

[Ongoing]

BitMAC: FPGA-based GenASM

Our Goal:

Map GenASM accelerators to an FPGA with HBM2,
where **HBM2 offers high memory bandwidth** and

FPGA resources offer high parallelism by instantiating
multiple copies of the GenASM accelerators

- ❑ **Re-modified GenASM algorithms** for a better mapping to the FPGA resources
- ❑ **Intra-level parallelism** by instantiating multiple processing elements (PEs) for the DC execution
- ❑ **Inter-level parallelism** by running multiple independent GenASM executions in parallel

Key Findings

- Based on the FPGA resources, **the complete BitMAC design:**
 - Each BitMAC accelerator contains a DC accelerator with 16 PEs, a TB accelerator, an FSM, and 13.2KB of M20Ks
 - 4 BitMAC accelerators connected to each pseudo-channel (128 in total)
 - Clocked at 200MHz

- BitMAC provides:
 - **136x – 761x speedup** over the state-of-the-art CPU baselines
 - **6.8x – 19.4x speedup** over the state-of-the-art GPU baseline

Key Findings (cont'd.)

- BitMAC has:
 - **64% logic utilization** and **90% on-chip memory utilization**
 - Total power consumption of **48.9W**, where **59%** accounts for the M20Ks
- **Bottlenecked by the amount of on-chip memory** (i.e., M20Ks)
- **Cannot saturate the high bandwidth** that multiple HBM2 stacks on the FPGA provide
- Need (1) **algorithm-level modifications** to decrease the amount of data that need to be stored in M20Ks, and (2) **newer FPGA chips that provide a higher amount of on-chip memory capacity**

Research Contributions

Bottleneck analysis of genome assembly pipeline for long reads

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

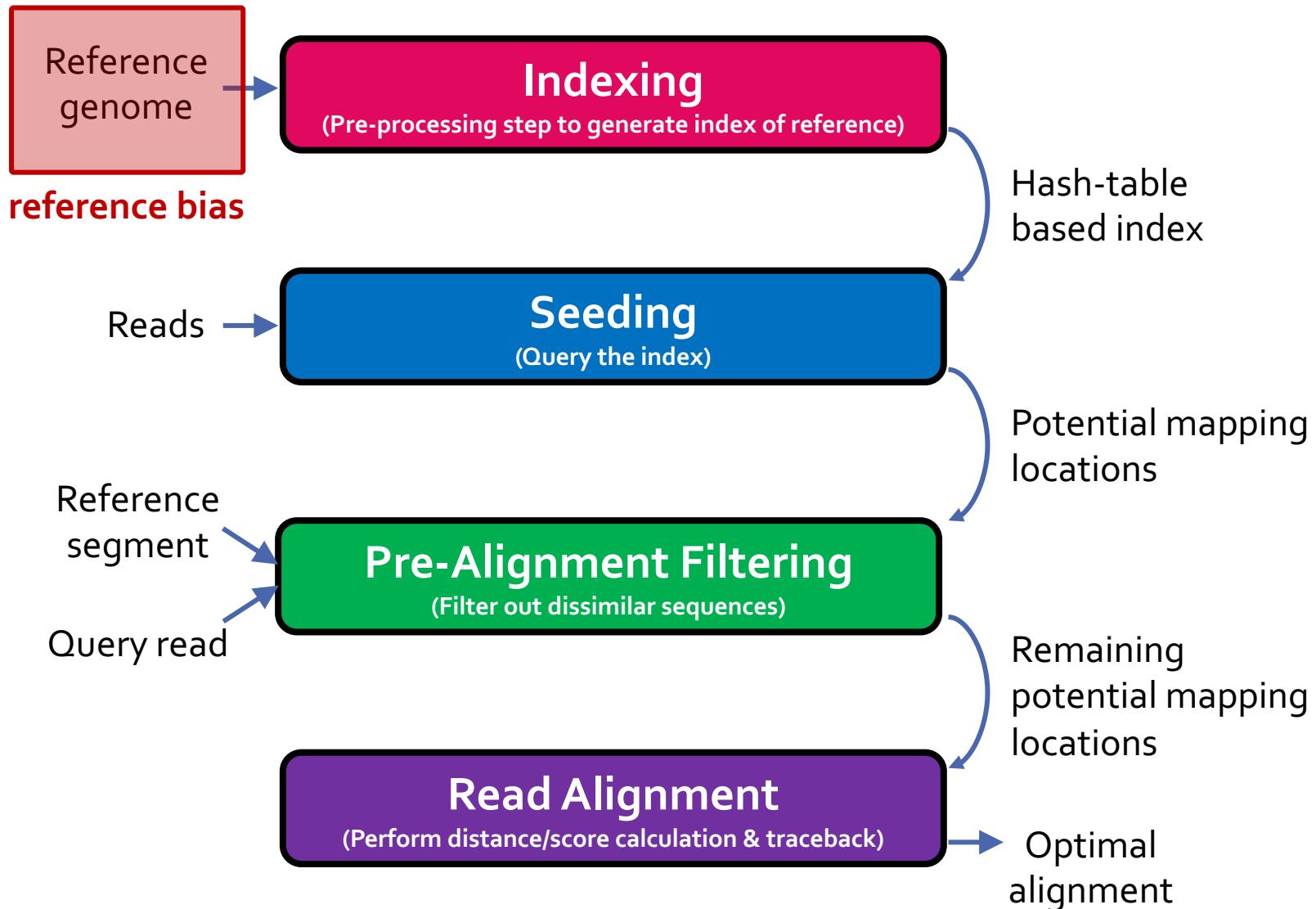
BitMAC: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Ongoing]

SeGraM: 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

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

ACGTACGT

Reference #2: ACGGACGT

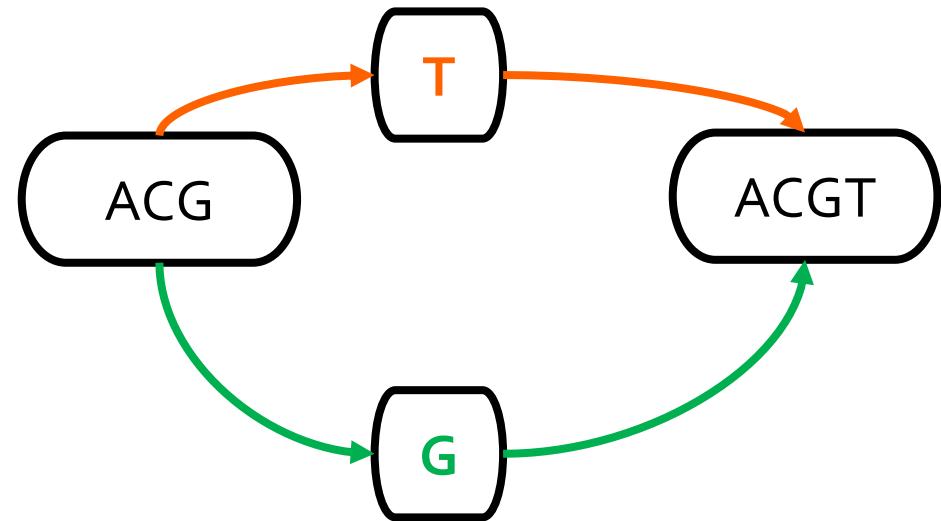
Genome Graphs

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: ACGTACGT

Reference #2: ACGGACGT



Genome Graphs

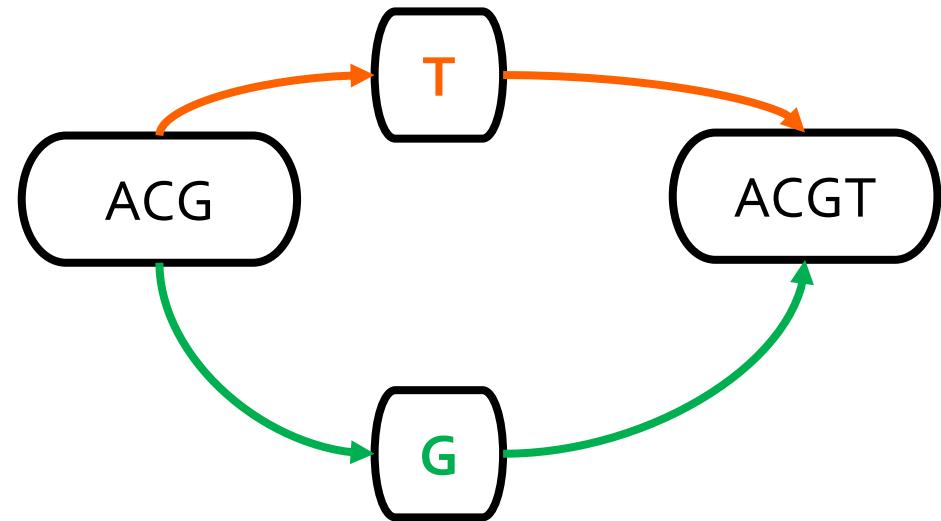
Genome graphs:

- ❑ 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: ACGTACGT

Reference #2: ACGGACGT

Reference #3: ACGTTACGT



Genome Graphs

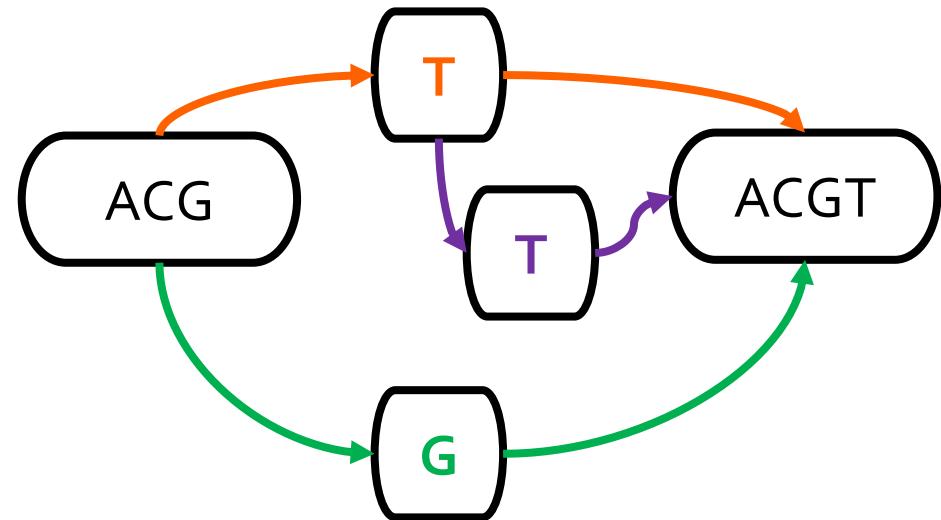
Genome graphs:

- ❑ 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



Genome Graphs

Genome graphs:

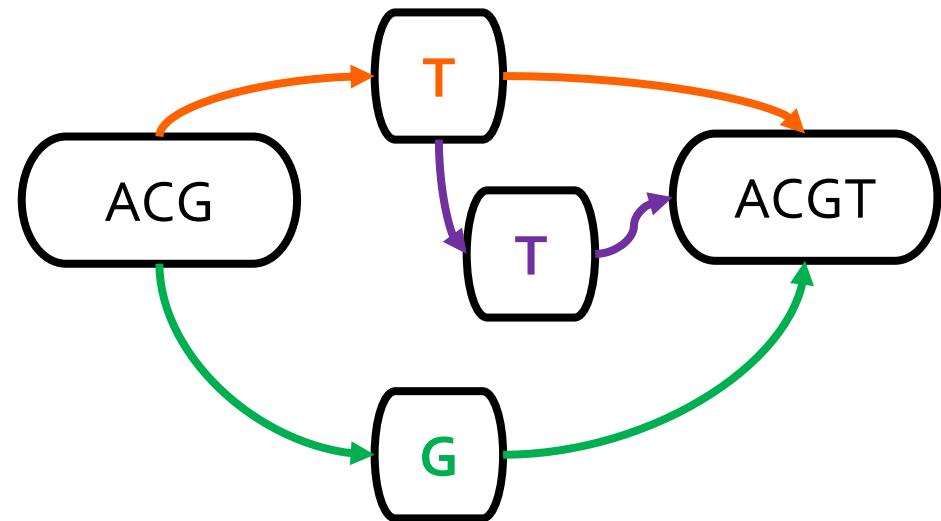
- ❑ Include the reference genome together with genetic variations
- ❑ Provide a compact representation
- ❑ Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACGTACGT

Reference #2: ACGGACGT

Reference #3: ACGTTACGT

Reference #4: ACGACGT



Genome Graphs

Genome graphs:

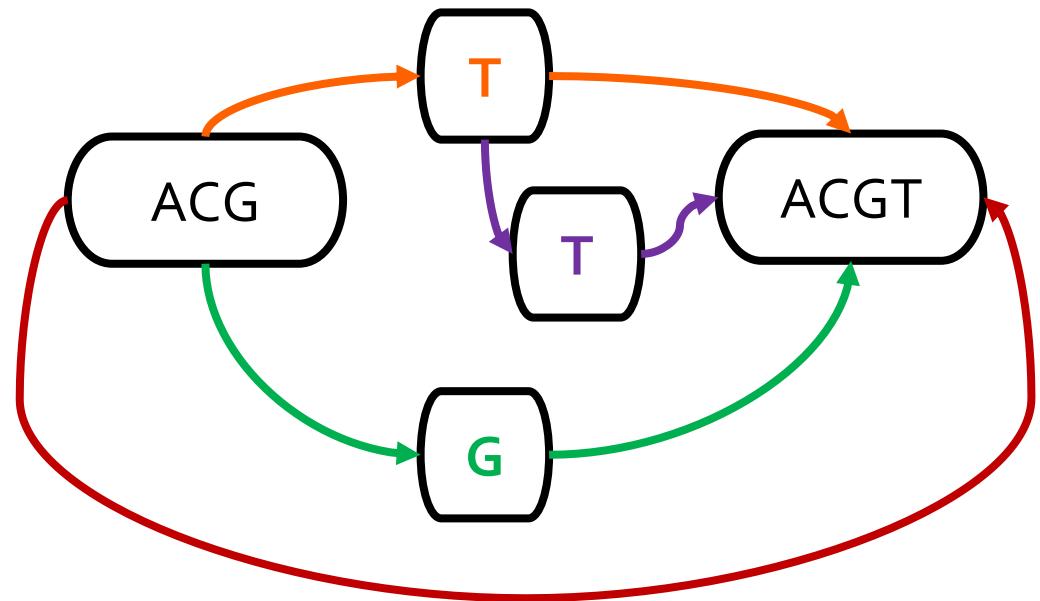
- ❑ Include the reference genome together with genetic variations
- ❑ Provide a compact representation
- ❑ Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

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

SeGraM: First Graph Mapping Accelerator

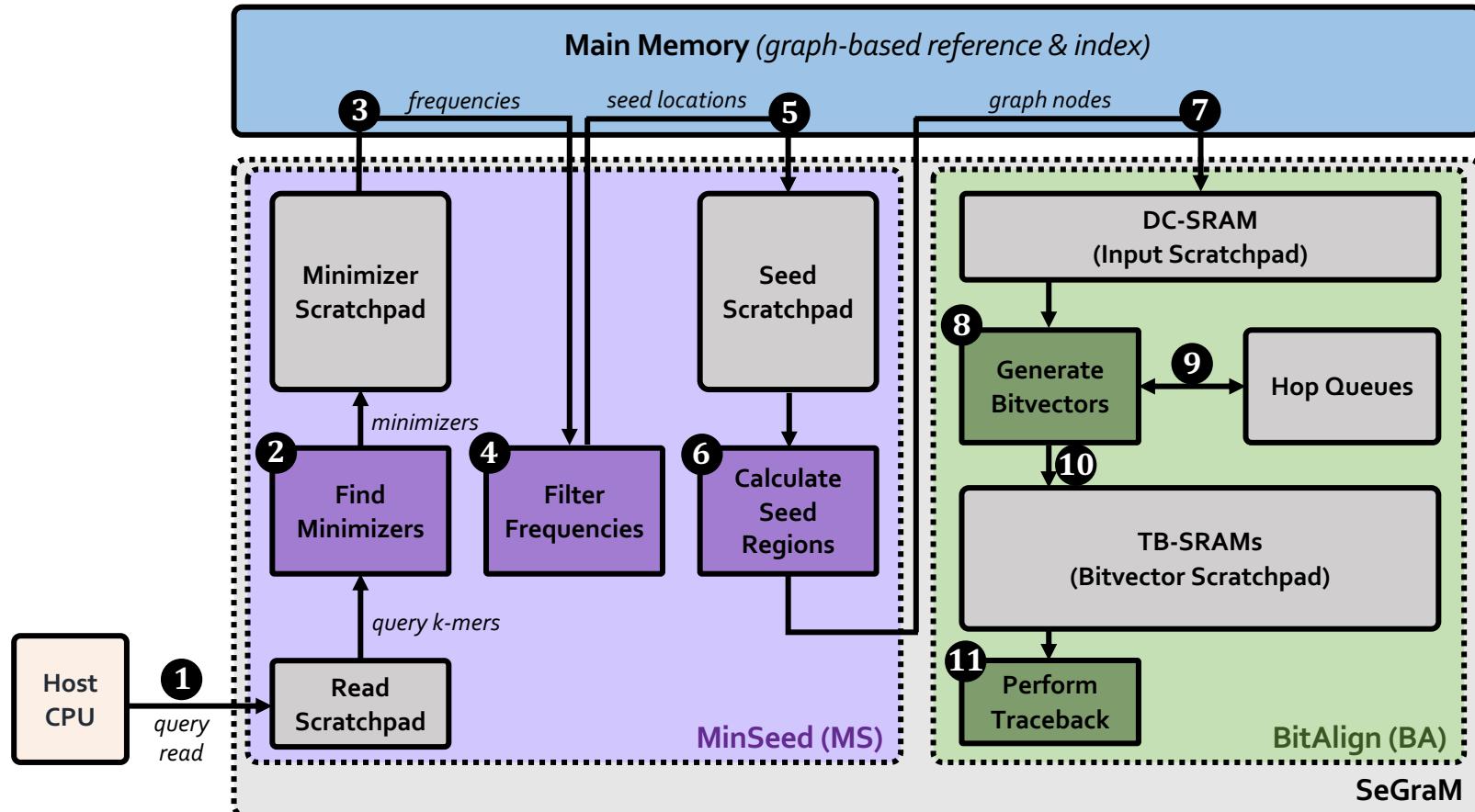
Our Goal:

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**

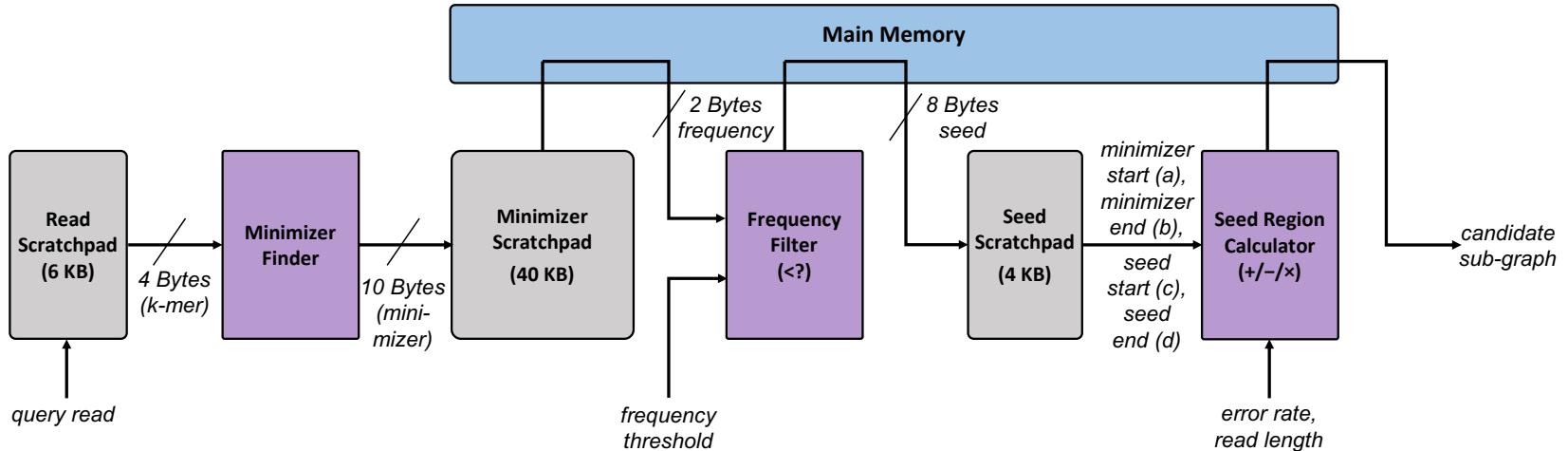
SeGraM:

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

Overview of SeGraM



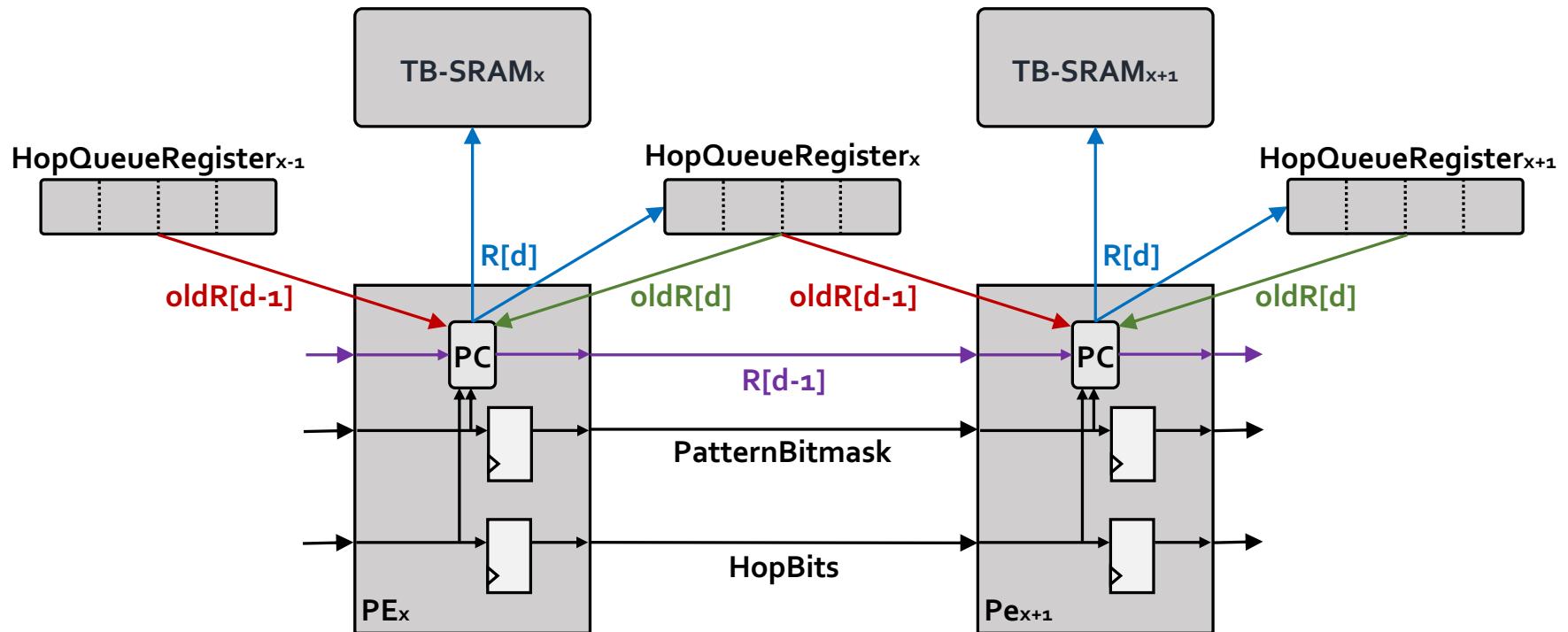
MinSeed HW



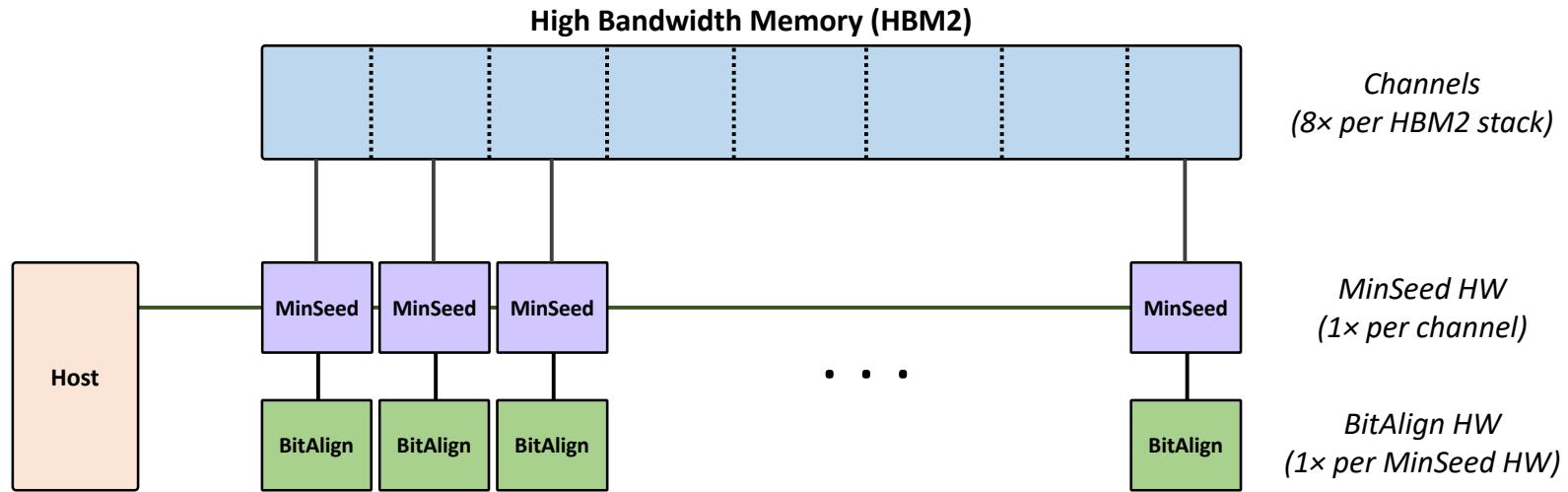
- (1) **Three computation modules** responsible for finding the minimizers, filtering the frequencies of minimizers, and finding the associated regions of every seed location
- (2) **Three scratchpads** for storing the query read, its minimizers, and seed locations
- (3) **The memory interface**, which 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 SeGram



- ❑ A single SeGram 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 SeGram

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

- The whole SeGram 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

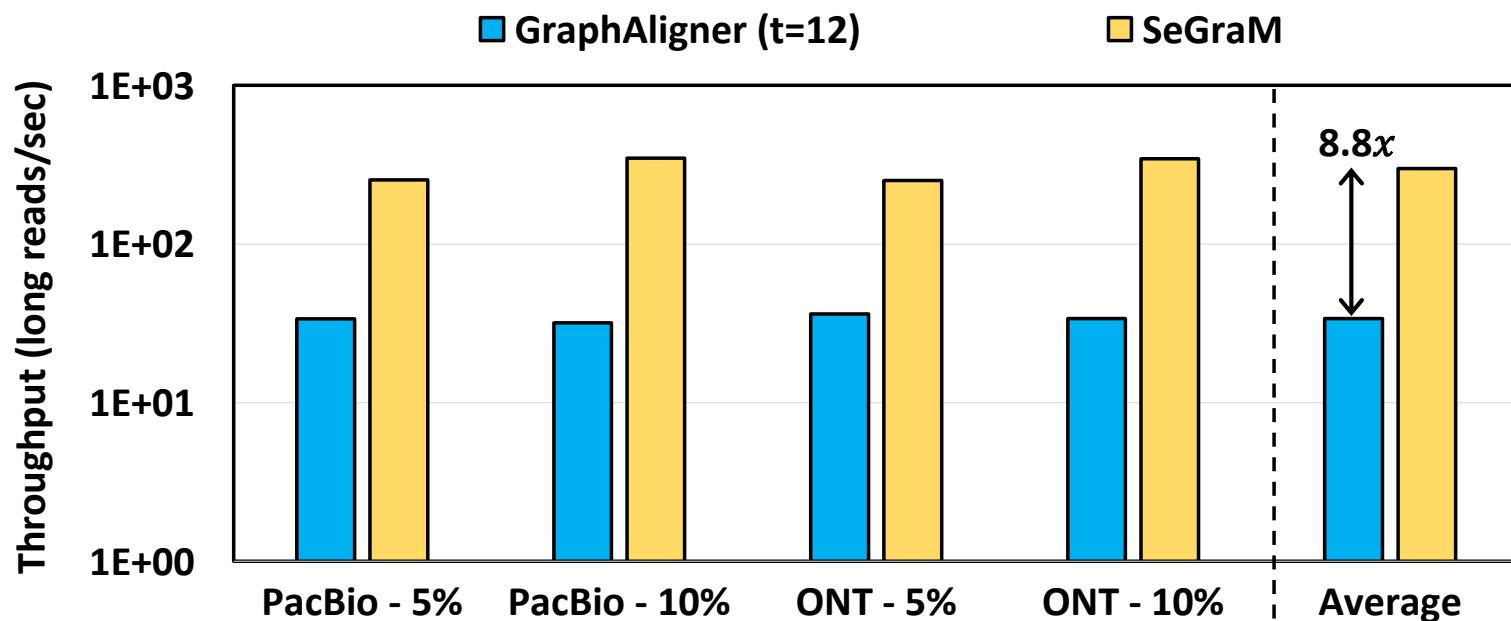
- We evaluate SeGram using:
 - **Synthesized** SystemVerilog models of the MinSeed and BitAlign accelerator datapaths
 - **Simulation-** and **spreadsheet-based** performance modeling
- **4 x 24GB HBM2E stacks, each with 8 independent channels**
 - 1 MinSeed and 1 BitAlign HW per each channel (**32 in 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**):

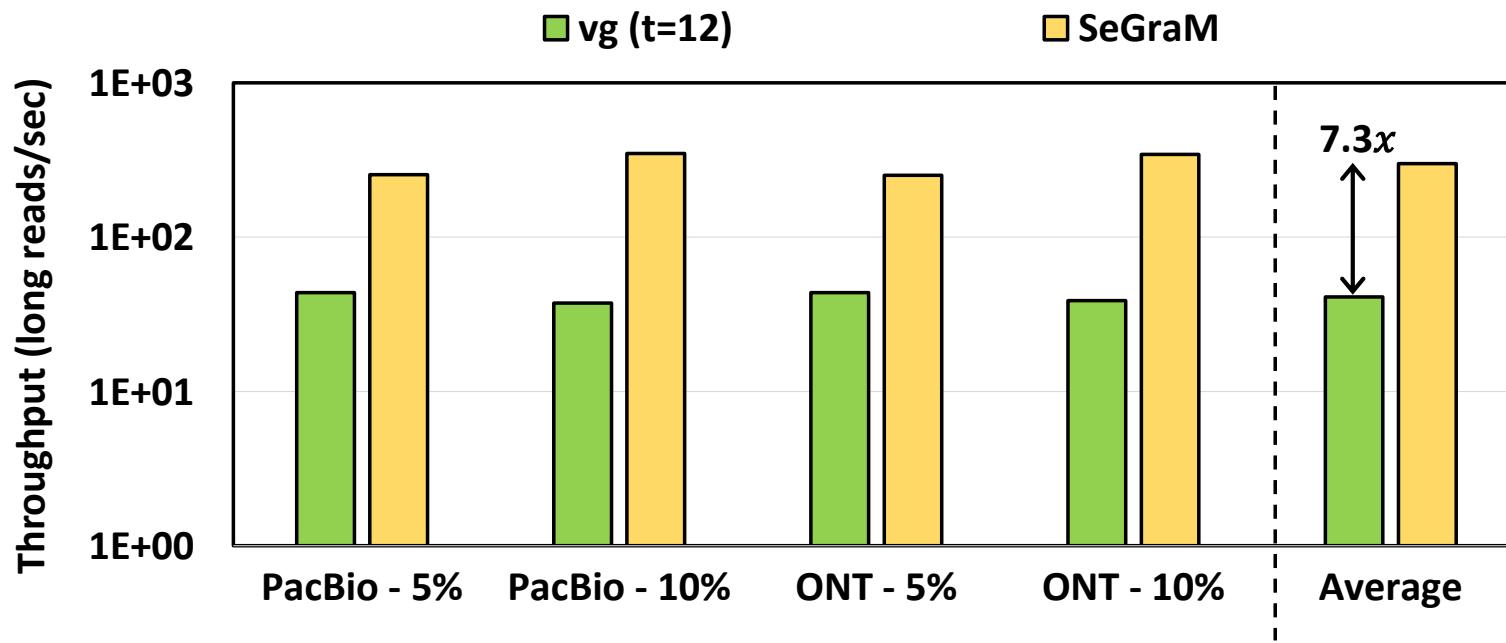
Component	Area (mm ²)	Power (mW)
MinSeed – Logic	0.017	10.8
Read Scratchpad (6 KB)	0.009	1.9
Minimizer Scratchpad (40 KB)	0.061	6.9
Seed Scratchpad (4 KB)	0.006	2.5
BitAlign – DC Logic with HopQueueRegisters (64 PEs)	0.393	378.0
BitAlign – TB Logic	0.020	2.7
Input Scratchpad (DC-SRAM; 24 KB)	0.034	8.4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

Key Results – SeGraM with Long Reads (I)



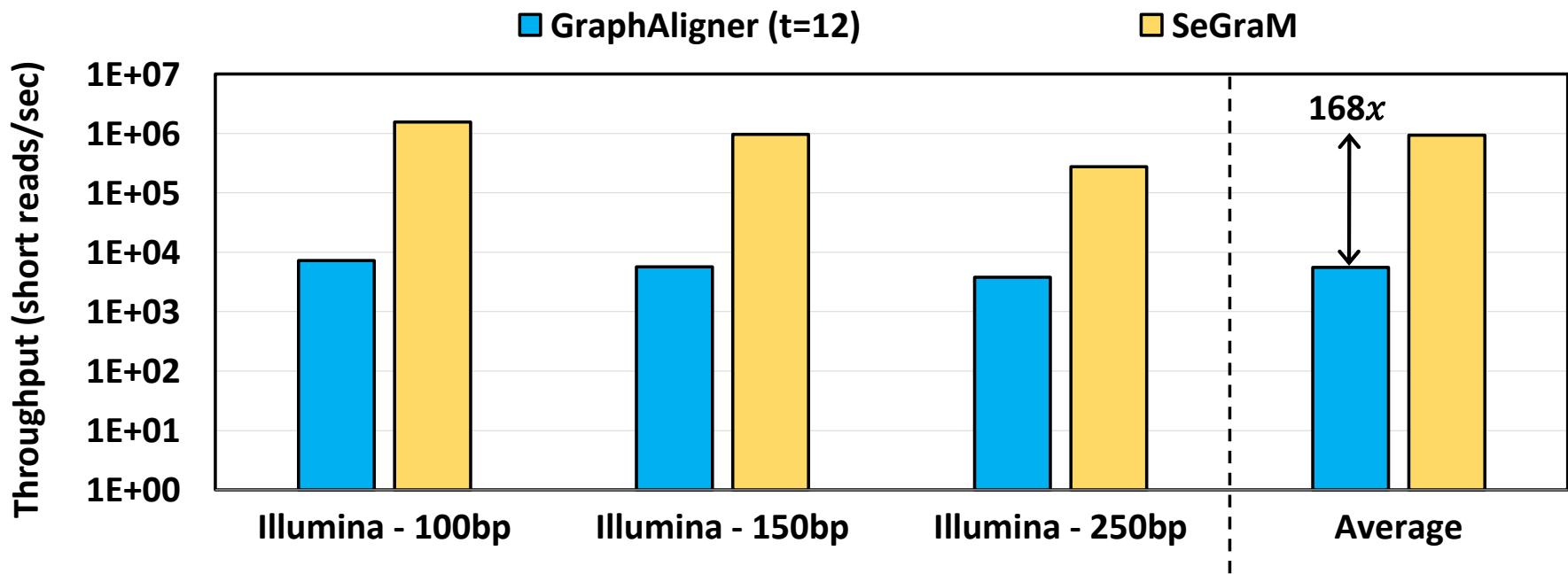
SeGraM provides **8.8x throughput improvement** over GraphAligner's 12-thread execution, while **reducing the power consumption by 4.9x**

Key Results – SeGraM with Long Reads (II)



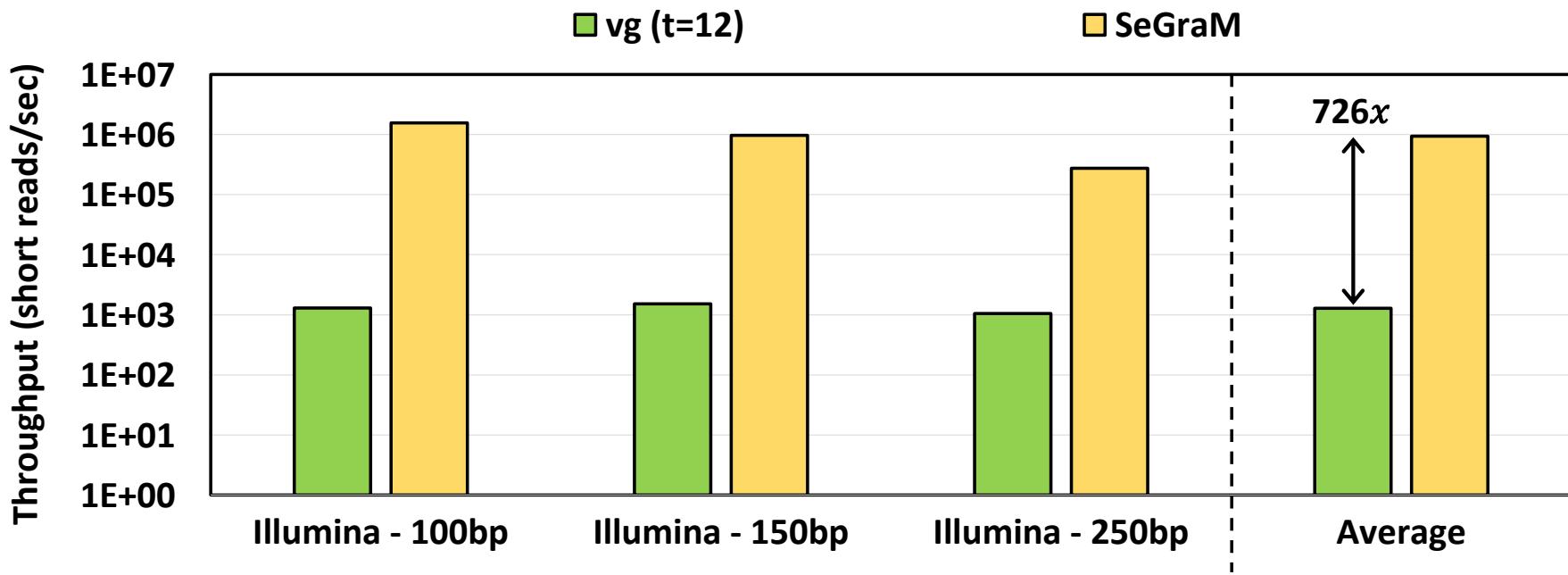
SeGraM provides **7.3x throughput improvement** over vg's 12-thread execution,
while **reducing the power consumption by 6.5x**

Key Results – SeGraM with Short Reads (I)



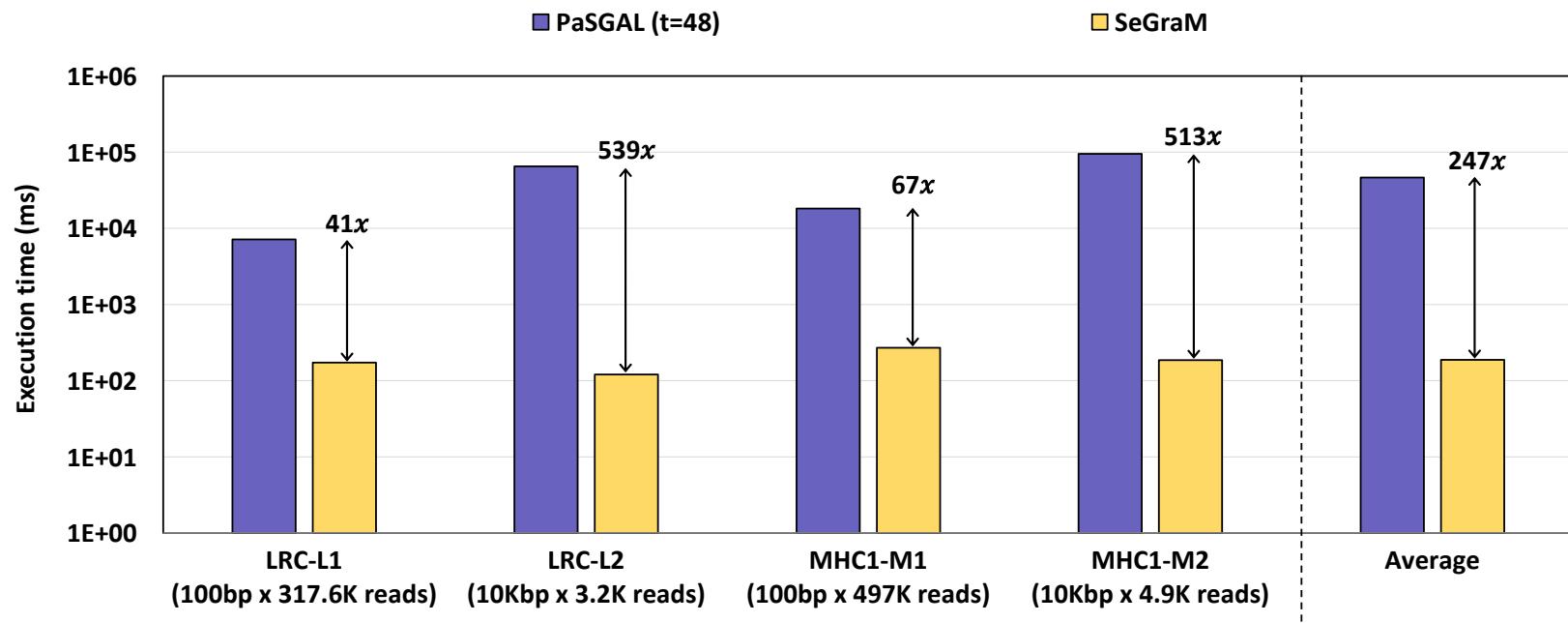
SeGraM provides **168x throughput improvement** over
GraphAligner's 12-thread execution,
while **reducing the power consumption by 4.7x**

Key Results – SeGraM with Short Reads (II)



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

Key Results – BitAlign (Graph Alignment)



BitAlign provides **41x-539x 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

Summary of SeGram

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:

- **SeGram**: *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: SeGram and BitAlign provide **significant speedups** compared to the software baselines, while **reducing the power consumption**

Research Contributions

Bottleneck analysis of genome assembly pipeline for long reads

[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]

SeGraM: 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 accelerated 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 genome assembly pipeline for long reads

[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]

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

[Ongoing]

Other Publications @ SAFARI

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.

Future Work

- Incorporating a Filtering Approach for SeGraM
- GenASM-based Algorithm for Sequence Alignment with the Affine Gap Model
- End-to-End Acceleration of the Mapping Pipeline
- Accelerating Assembly with Long Reads

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

Damla Senol Cali

Staff Software Engineer, Hardware Acceleration
Bionano Genomics

Email: damlasenolcali@gmail.com

Website: <https://damlasenolcali.github.io>

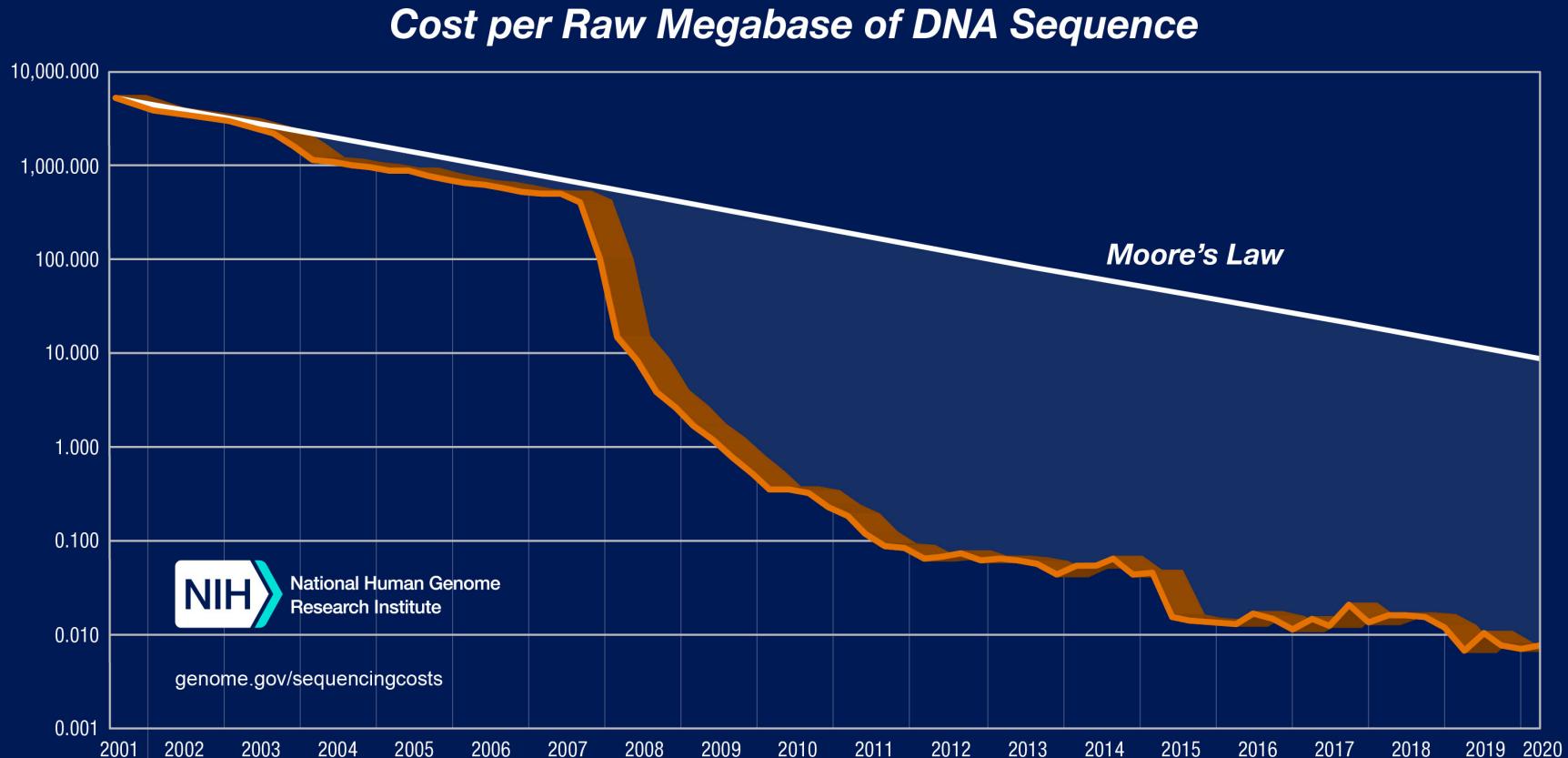
SAFARI Live Seminar

November 7, 2021

Backup Slides

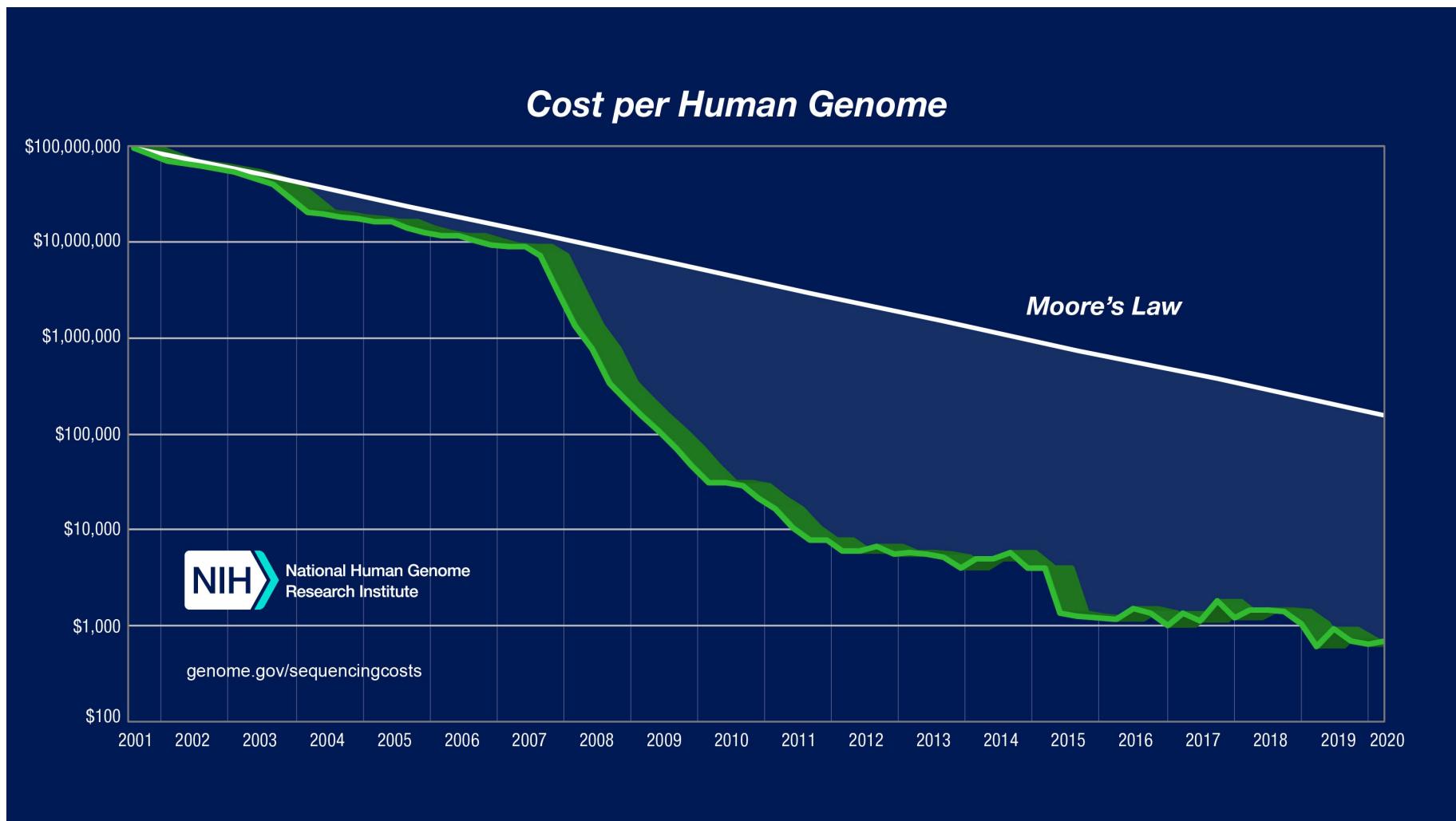
(Sequencing)

Cost of Sequencing



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

Cost of Sequencing (cont'd.)



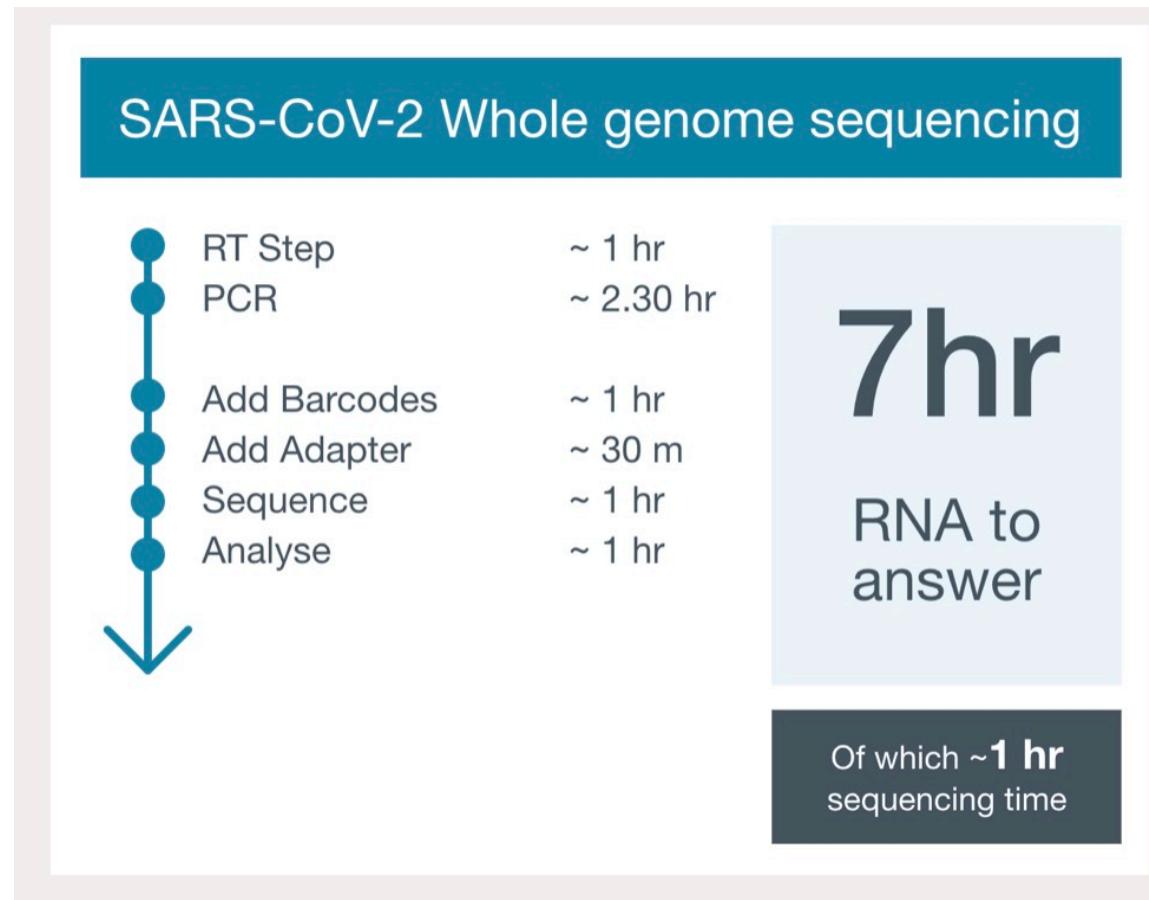
*From NIH (<https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data>)

Sequencing of SARS-CoV-2

Why genome sequencing and sequence data analysis are important?

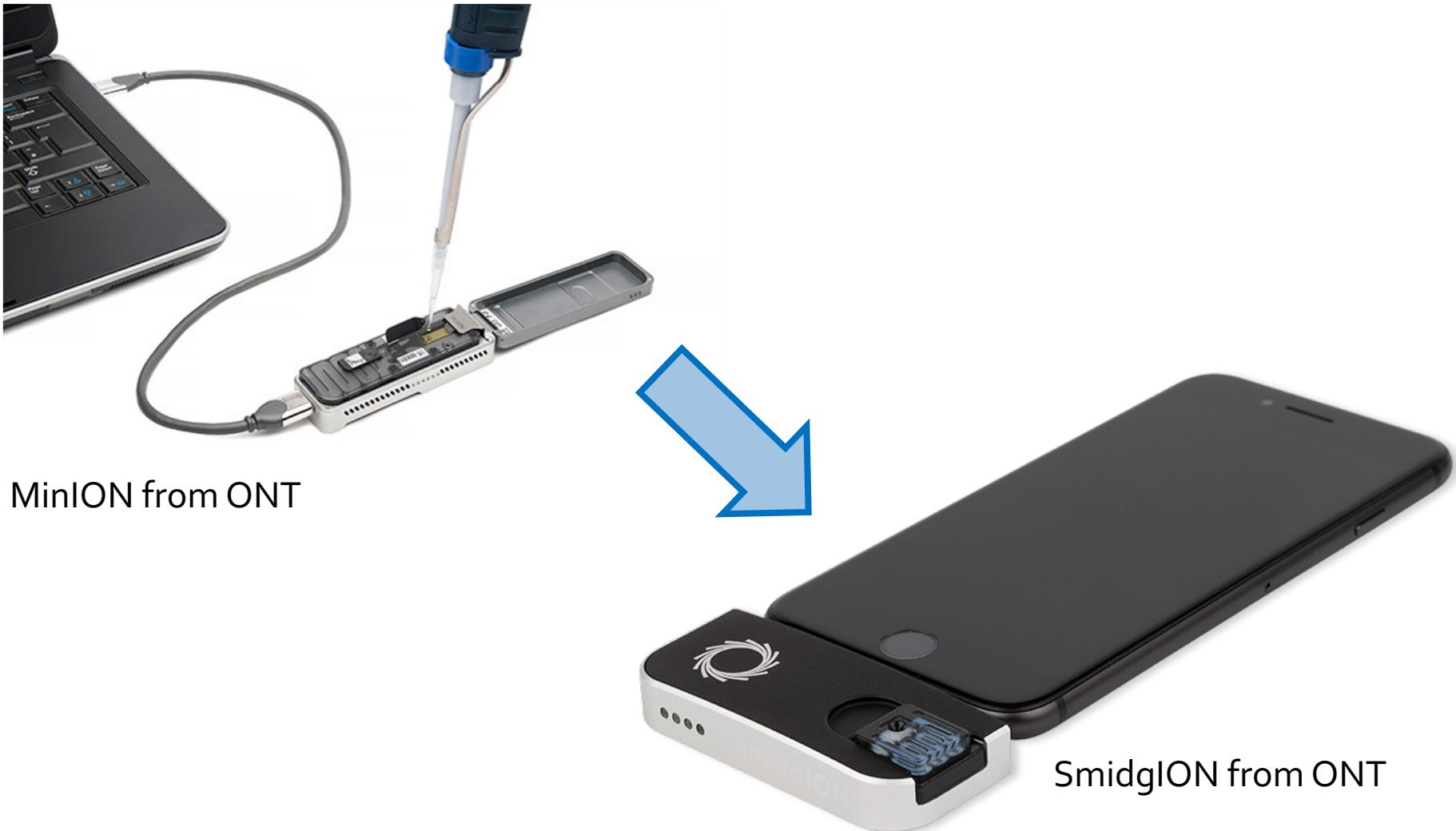
- To detect the virus from a human sample
- To understand the sources and modes of transmission of the virus
- To sequence the genome of the virus itself, COVID-19, in order to track the mutations in the virus
- To explore the genes of infected patients
 - To understand why some people get more severe symptoms than others
 - To help with the development of new treatments

COVID-19 Research with ONT



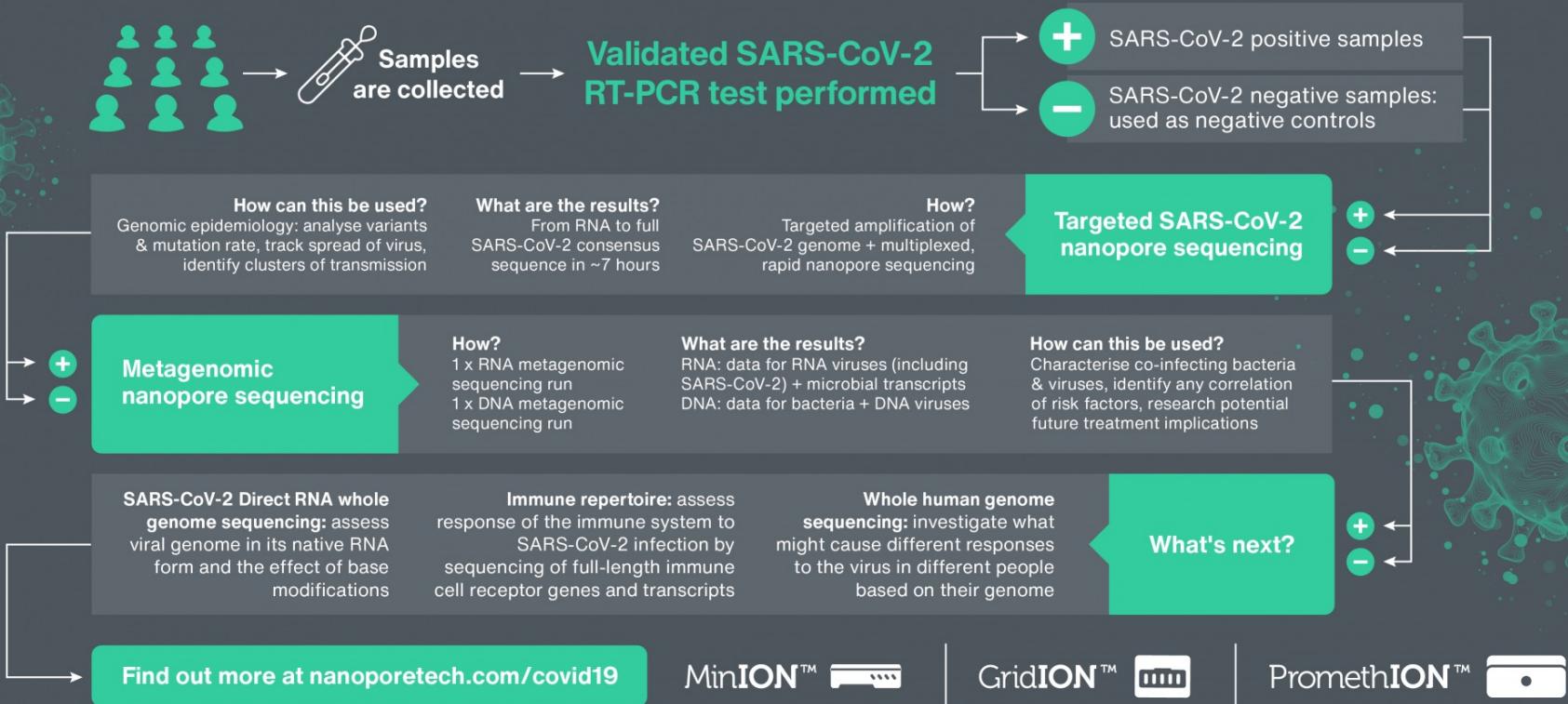
- From ONT (<https://nanoporetech.com/covid-19/overview>)

Future of Genome Sequencing & Analysis



COVID-19 Research with ONT (cont'd.)

How are scientists using nanopore sequencing to research COVID-19?



Oxford Nanopore Technologies, the Wheel icon, GridION, PromethION and MinION are registered trademarks of Oxford Nanopore Technologies in various countries. © 2020 Oxford Nanopore Technologies. All rights reserved. Oxford Nanopore Technologies' products are currently for research use only IG_1061[EN]_V1_03April2020

- From ONT (<https://nanoporetech.com/covid-19/overview>)

Backup Slides

(Nanopore)

Nanopore Sequencing Technology

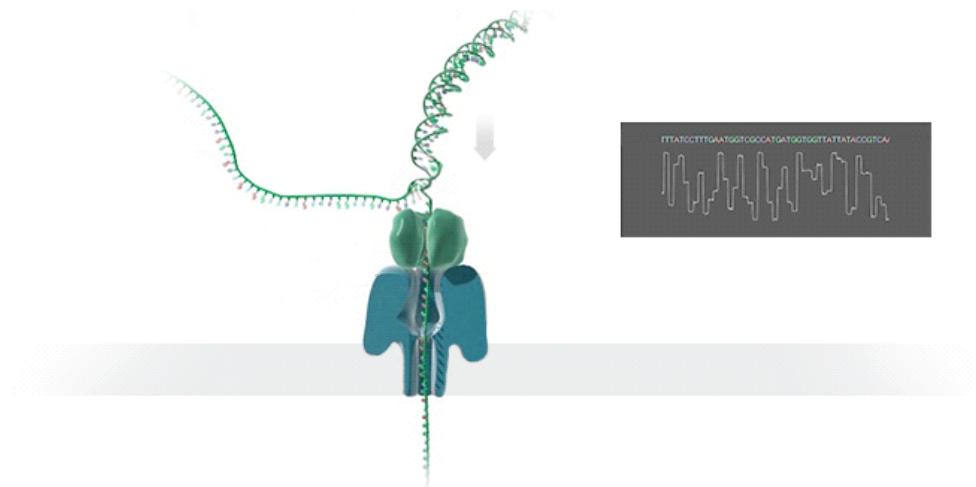
- **Nanopore sequencing** is an emerging and a promising single-molecule DNA sequencing technology.

- First nanopore sequencing device, **MinION**, made commercially available by **Oxford Nanopore Technologies (ONT)** in **May 2014**.
 - Inexpensive
 - Long read length (>882Kbp)
 - Produces data in real time
 - Pocket-sized and portable

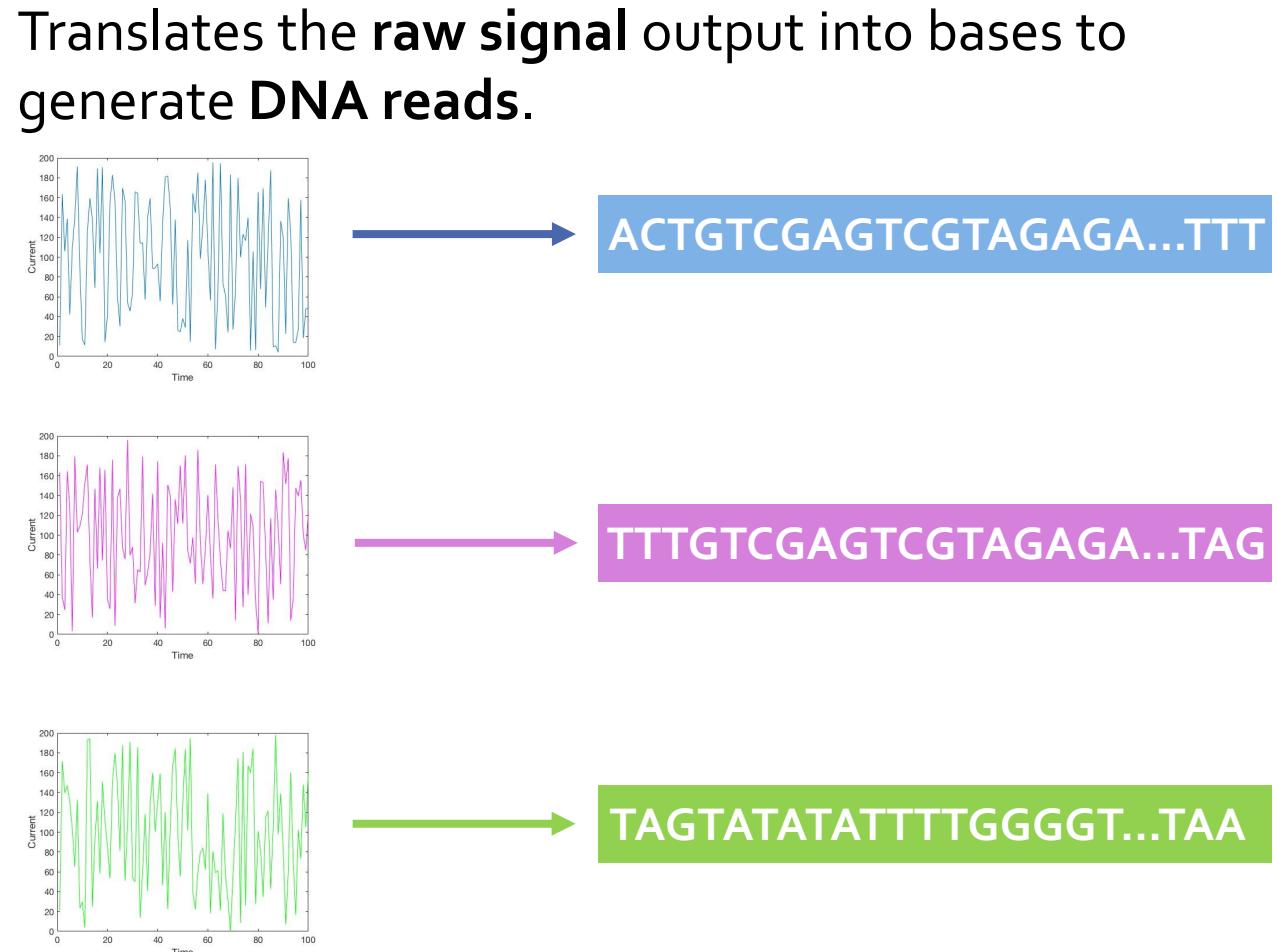
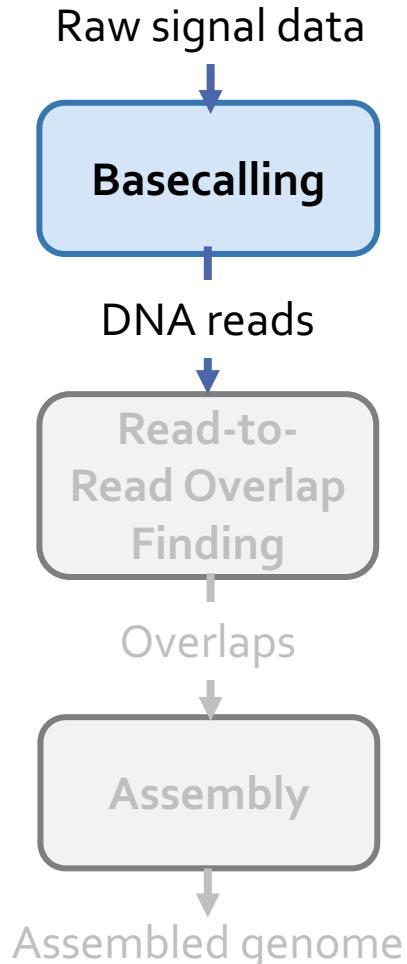


Nanopore Sequencing

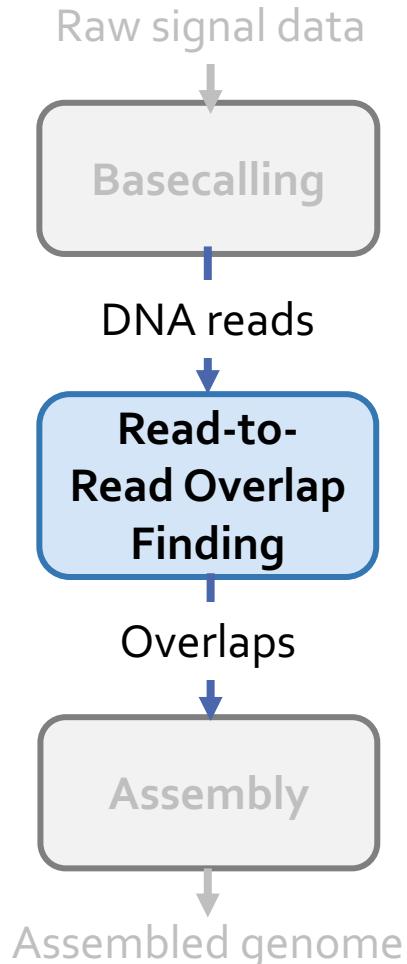
- **Nanopore** is a nano-scale hole.
- In nanopore sequencers, an **ionic current** passes through the nanopores.
- When the DNA strand passes through the nanopore, the sequencer measures the **change in current**.
- This change is used to identify the bases in the strand with the help of **different electrochemical structures** of the different bases.



Step 1: Basecalling



Step 2: Read-to-Read Overlap Finding



Read-to-read overlap

- is a **common sequence between two reads**, and
- occurs when the matched regions of these reads **originate from the same part of the complete genome**.

ACTGTCGAGTCGT...TTT



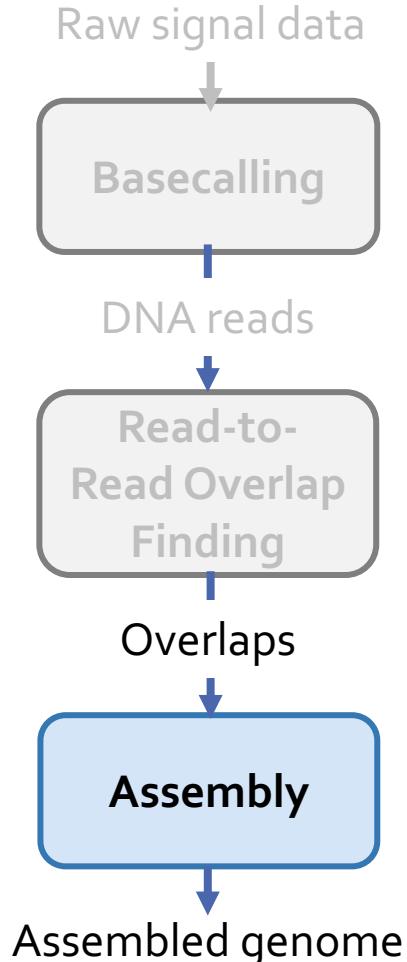
TTTGTCTGAGTCGT...ACT



ACTTATATATTTT...TTT

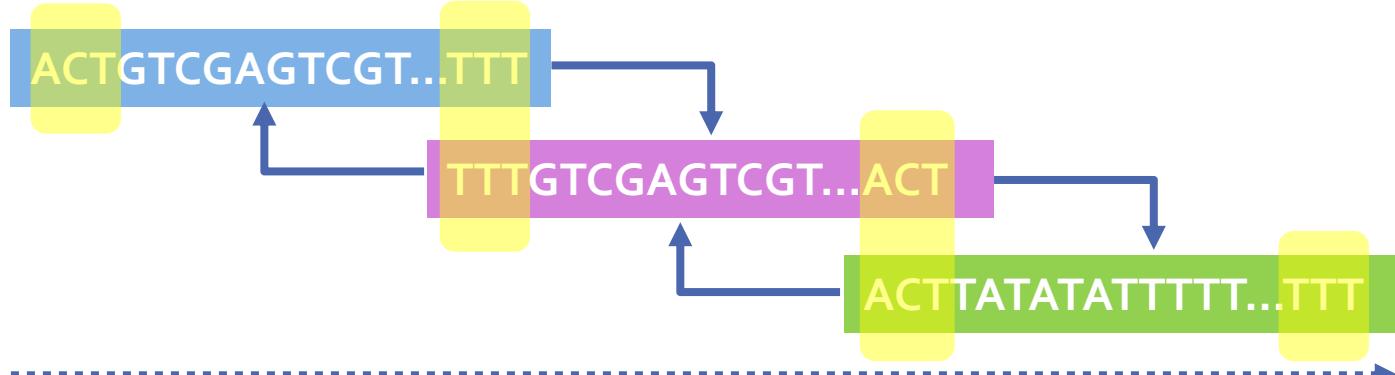
ACTTATATATTTT...TTT

Step 3: Assembly



Assembly algorithms,

- generate an **overlap graph** with the overlaps from the previous step,
- **traverse** this graph, then
- **construct** the assembled genome.



ACTGTCGAGTCGT...TTT GTCGAGTCGT...ACT TATATATT...TTT
ACTTATATATT...TTT GTCGAGTCGT...ACT GTCGAGTCGT...TTT

Which one is correct?

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**

Experimental Methodology

Basecalling

Tool	Strategy
Metricor	RNN
Nanonet	RNN
Scrappie	RNN
Nanocall	HMM
DeepNano	RNN

Read-to-Read Overlap Finding

Tool	Strategy
GraphMap	k-mer similarity
Minimap	Minimizer similarity

Assembly

Tool	Strategy
Canu	OLC with error correction
Miniasm	OLC without error correction

Read Mapping

Tool	Strategy
BWA-MEM	Burrows–Wheeler Transform
GraphMap	k-mer similarity
Minimap	Minimizer similarity

Polishing

Tool	Strategy
Nanopolish	HMM
Racon	Partial order alignment graph

Nanopore Basecalling Tools

- ❑ Metrichor
 - ONT's cloud-based basecaller
 - Uses recurrent neural networks (**RNN**) for basecalling
- ❑ Nanonet
 - ONT's offline and open-source alternative for Metrichor
 - Uses **RNN** for basecalling
- ❑ Scrappie
 - ONT's newest basecaller that explicitly addresses basecalling errors in homopolymer regions
- ❑ Nanocall [David+, Bioinformatics 2016]
 - Uses Hidden Markov Models (**HMM**) for basecalling
- ❑ DeepNano [Boža+, PloS One 2017]
 - Uses **RNN** for basecalling

Read-to-Read Overlap Finding Tools

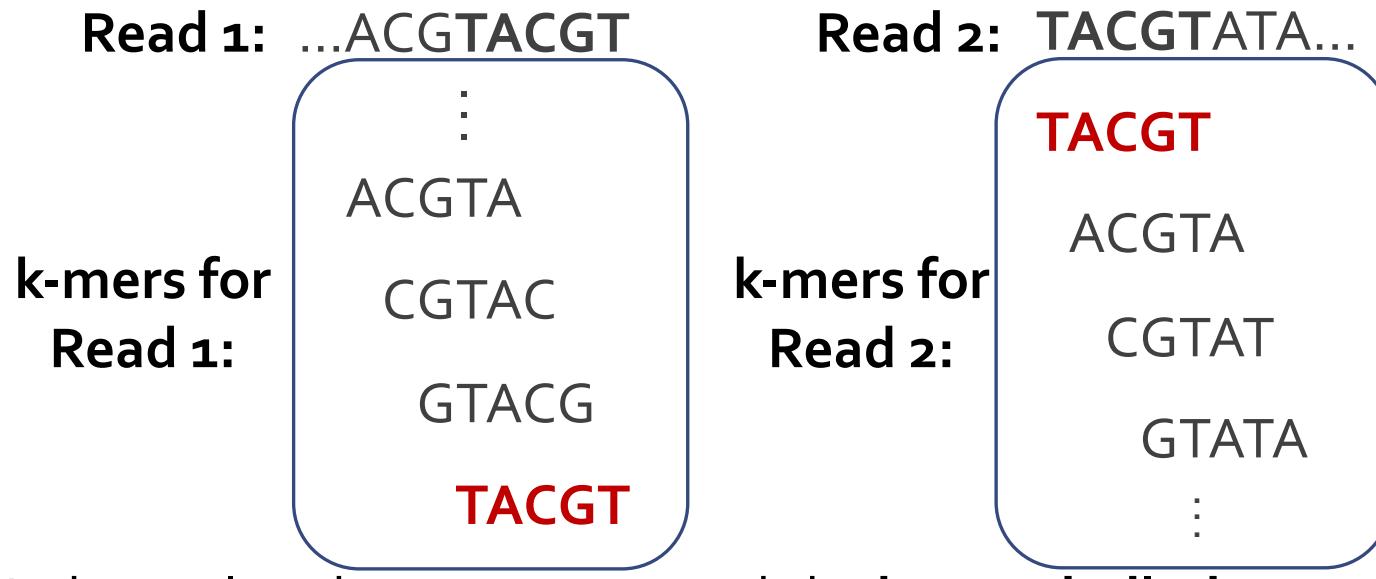
- GraphMap [Sovic' +, Nature Communications 2016]
 - First partitions the entire read data set into k -length substrings (i.e., *k-mers*), and then stores them in a hash table with the positions.
 - Detects the overlaps by finding the *k-mer similarity* between any two given reads, using the generated hash table.

- Minimap [Li+, Bioinformatics 2016]
 - Partitions the entire read data set into k -mers, but instead of creating a hash table for the full set of k -mers, finds the minimum representative set of k -mers, called *minimizers*, and creates a hash table with only these minimizers.
 - Finds the overlaps between two reads by finding *minimizer similarity*.

GraphMap vs. Minimap

□ GraphMap

- Finds **k-mers** and store them in hash table with the positions.

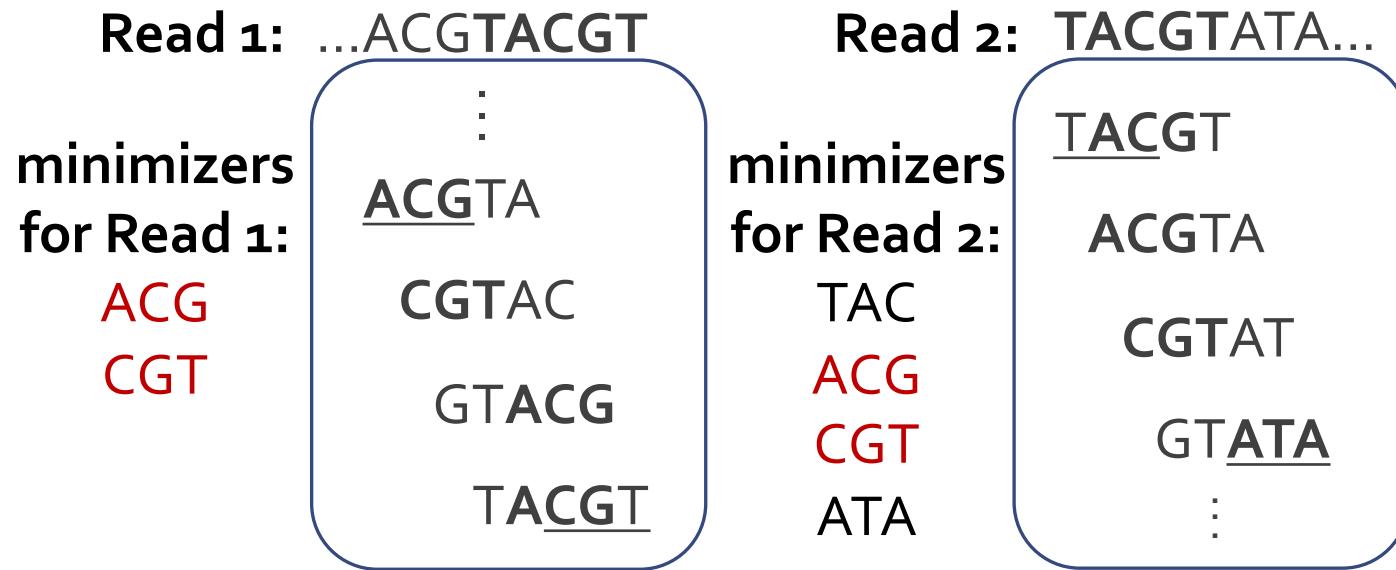


- Finds overlaps between two reads by **k-mer similarity**.

GraphMap vs. Minimap

□ Minimap

- Finds minimum representative set of k-mers, i.e. **minimizers** and store them in hash table, instead of storing all k-mers.



- Finds overlaps between two reads by **minimizer similarity**.

Assembly Tools

- Canu [Koren+, Genome Research 2017]
 - Performs **error-correction** as the initial step of its own pipeline
 - Improves the accuracy of the bases in the reads
 - Computationally-expensive
 - After the error-correction step, finds overlaps between **corrected reads** and constructs a draft assembly
- Miniasm [Li+, Bioinformatics 2016]
 - Skips the error-correction step, and constructs the draft assembly from the **uncorrected read overlaps** computed in the previous step.
 - **Lowers computational cost** but the accuracy of the draft assembly depends directly on the **accuracy of the uncorrected basecalled reads**.

Read Mapping & Polishing Tools

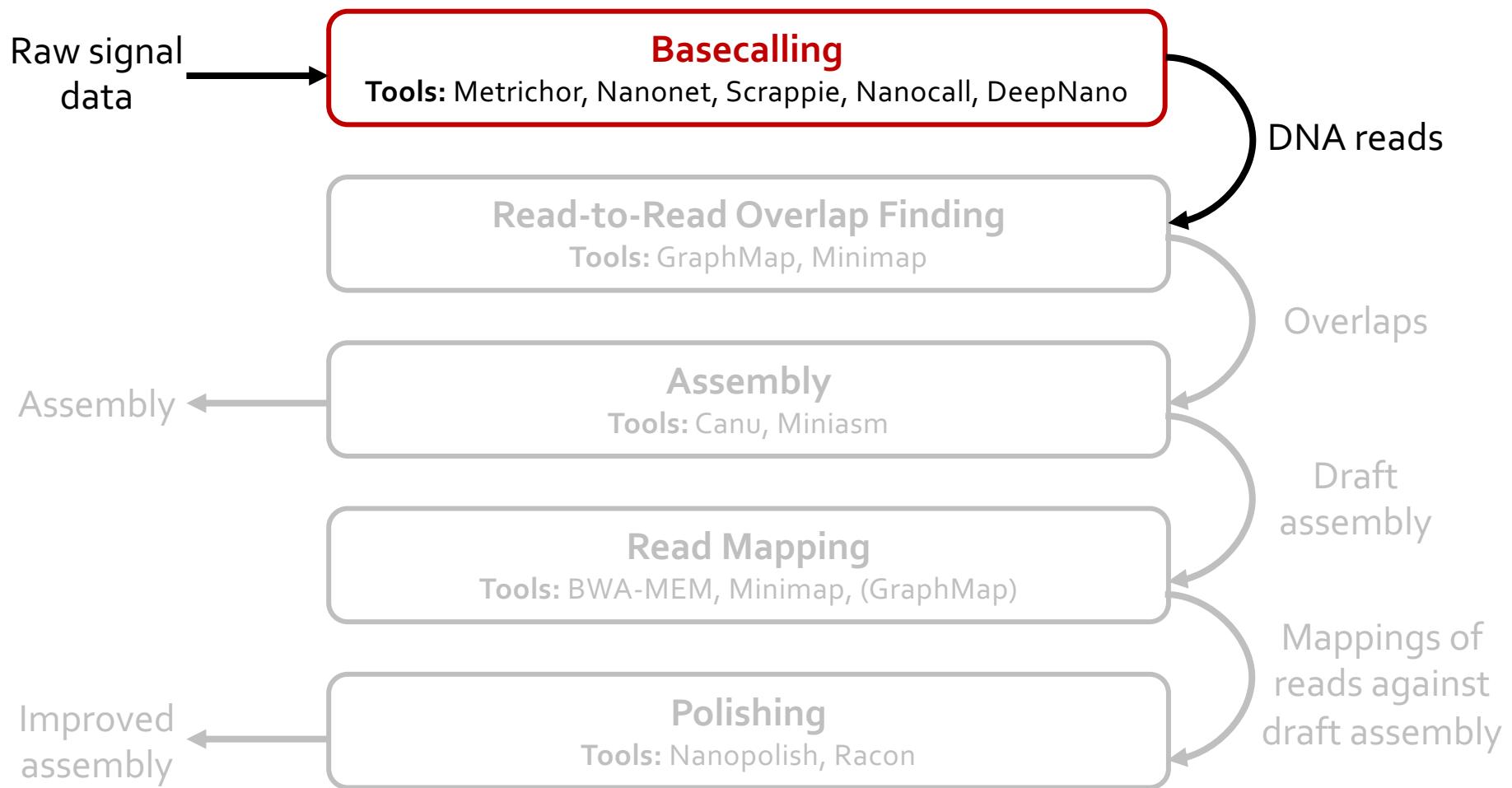
❑ Read Mapping tools

- BWA-MEM [Li, arXiv 2013]
 - Commonly used long-read mapper
- GraphMap and Minimap (from Step 2)

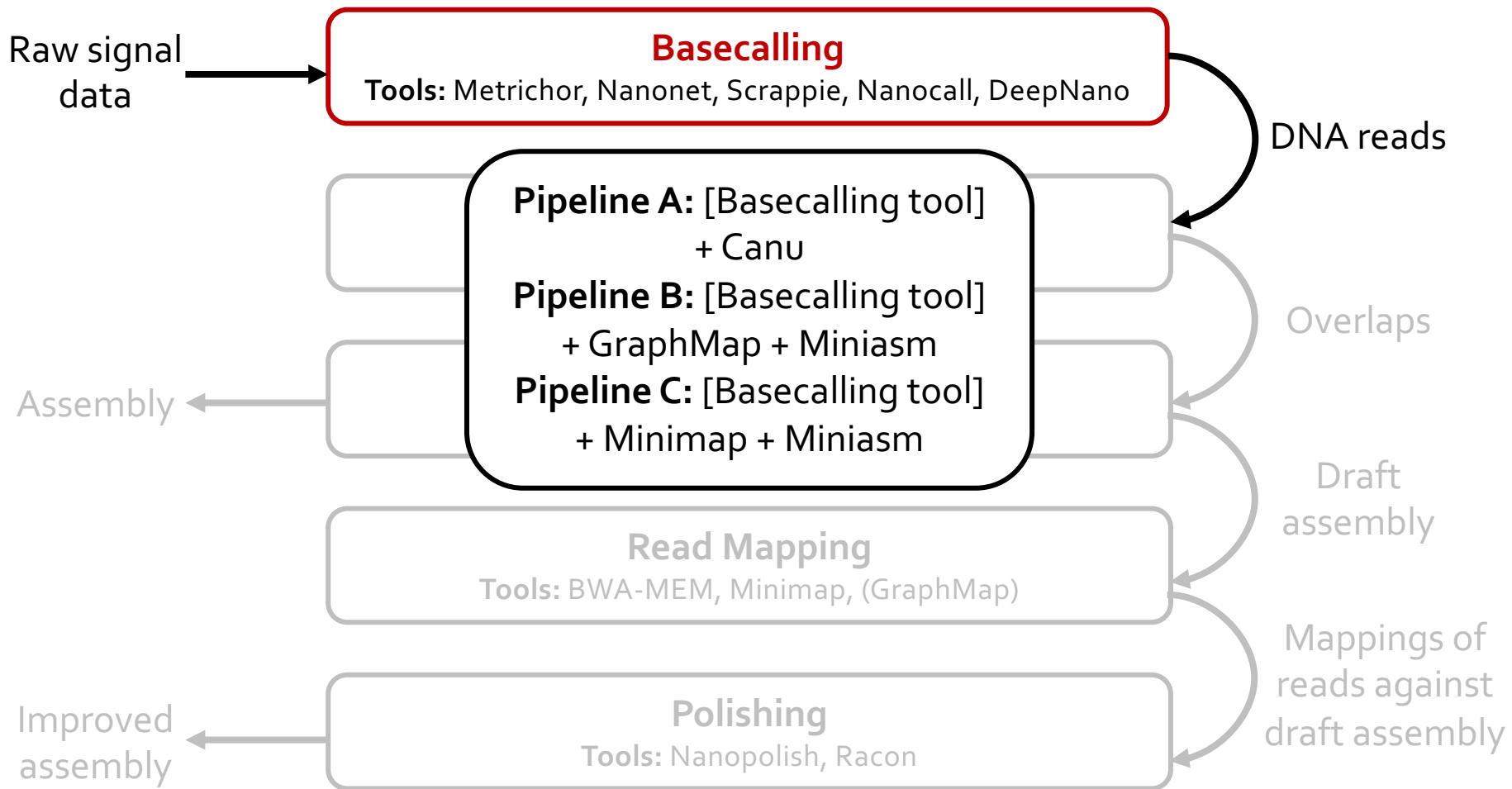
❑ Polishing tools

- Nanopolish [Loman+, Nature Methods 2015]
 - HMM-based approach for polishing
- Racon [Vaser+, Genome Research 2017]
 - Alignment graph-based approach for polishing

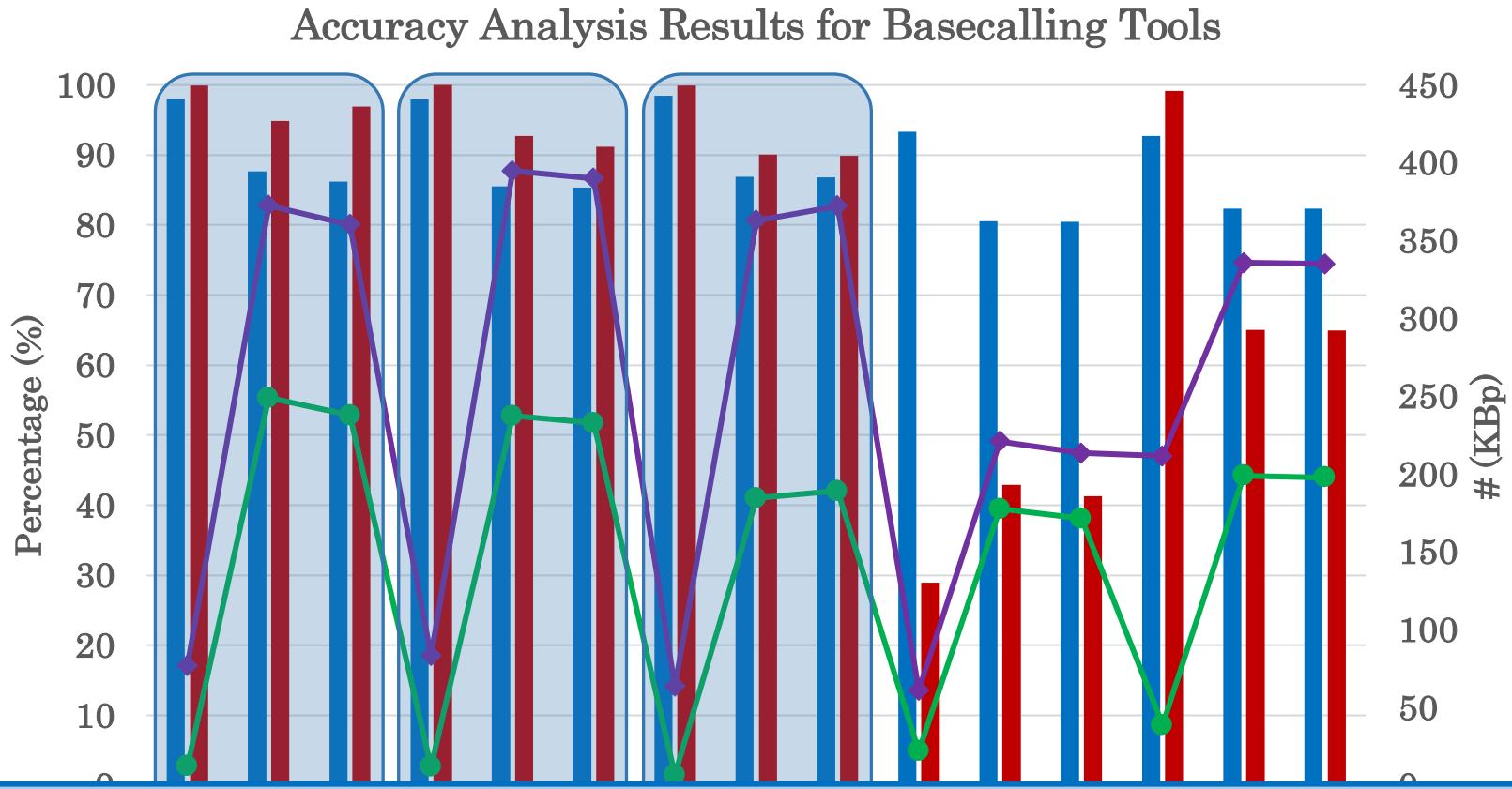
Nanopore Genome Assembly Pipeline



Nanopore Genome Assembly Pipeline

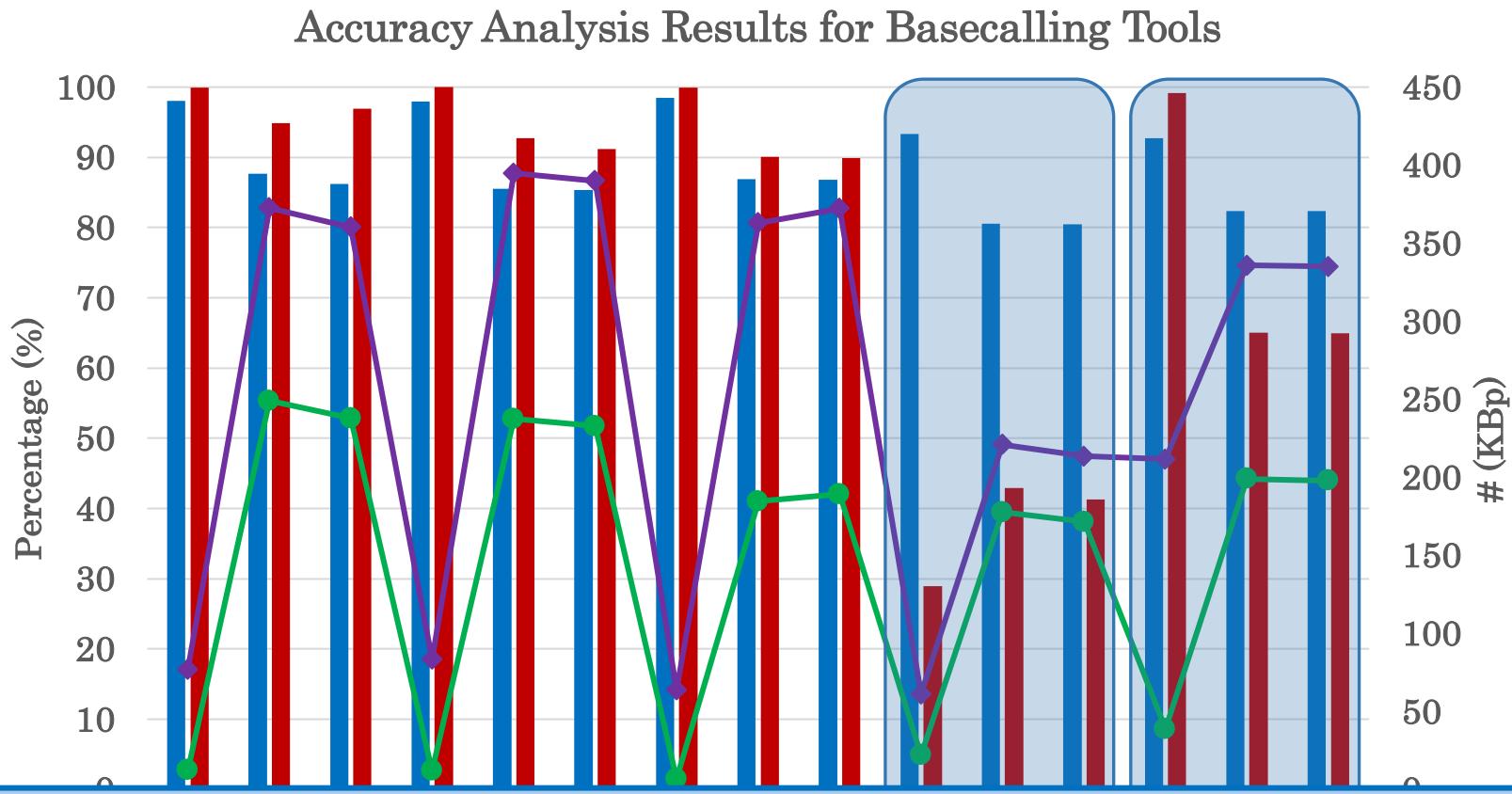


Basecalling – Accuracy



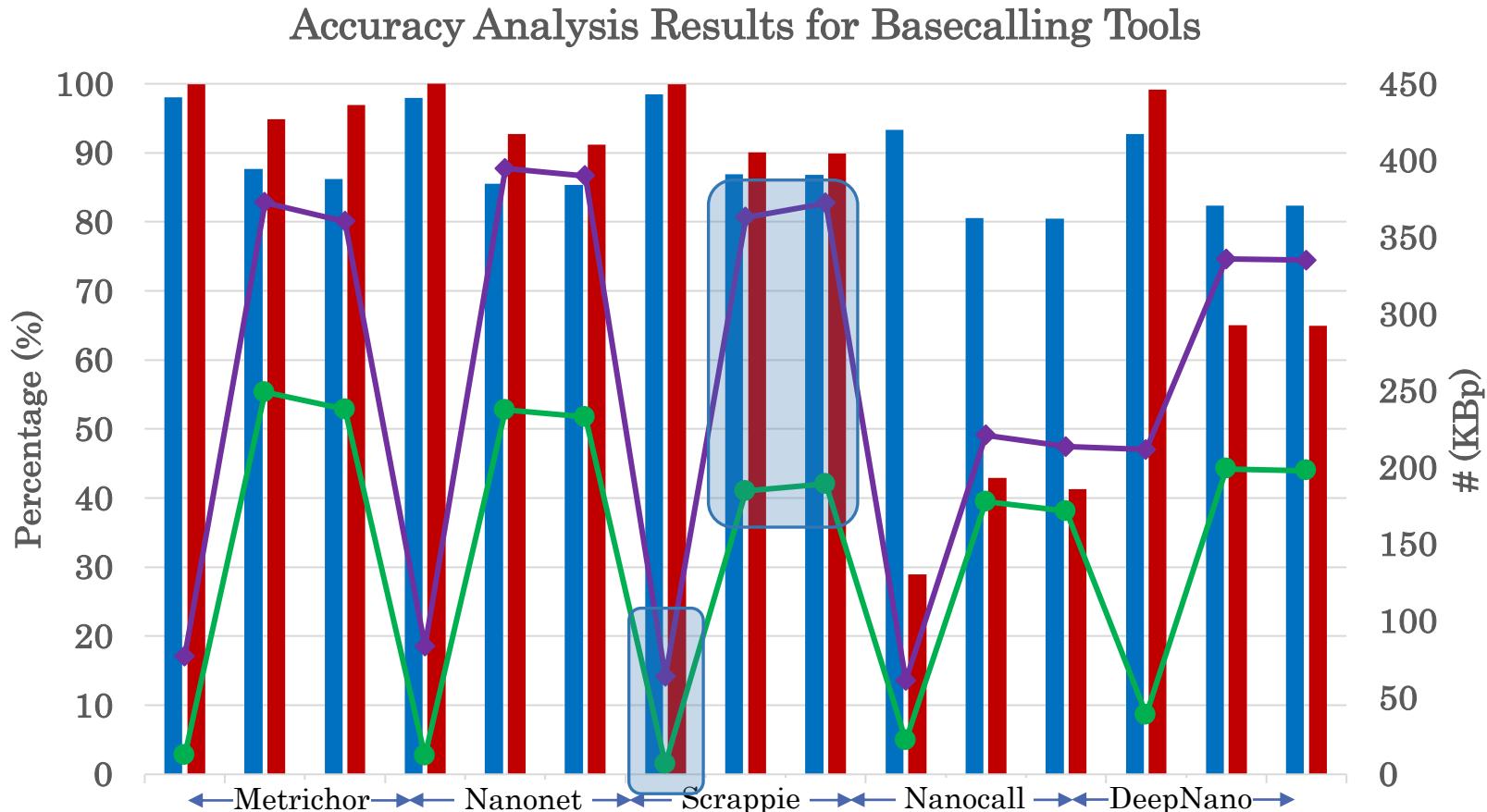
Observation 1-a: Metrichor, Nanonet and Scrappie have similar identity and coverage trends among all of the evaluated scenarios.

Basecalling – Accuracy



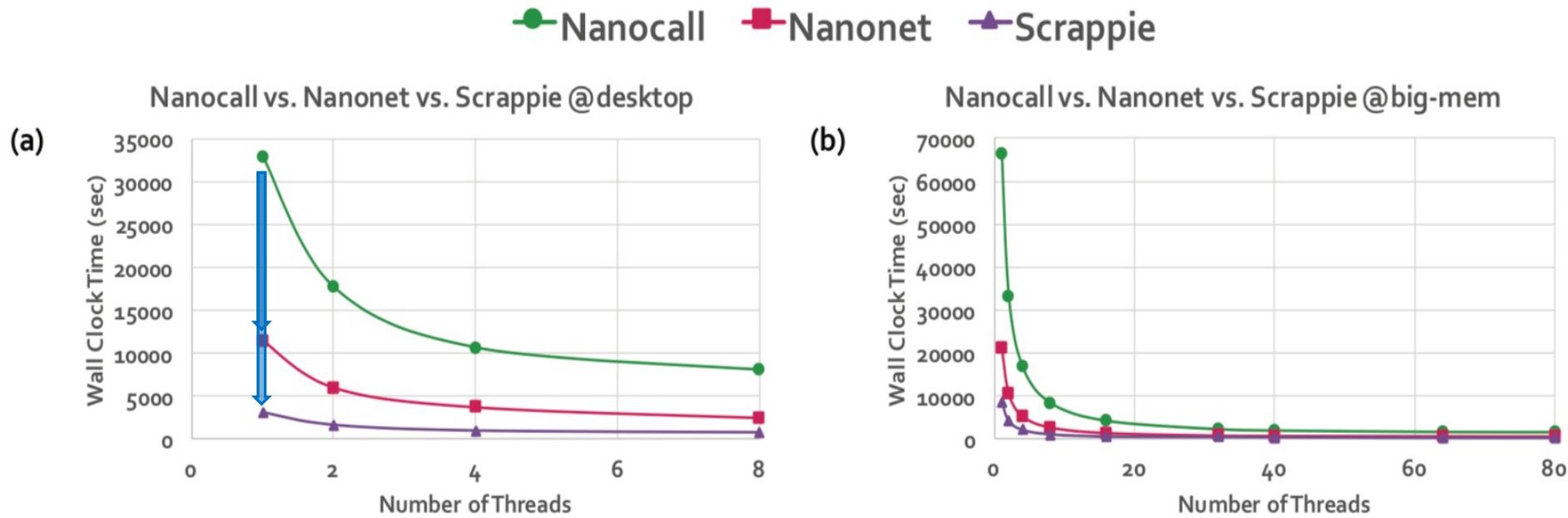
Observation 1-b: However, Nanocall and DeepNano cannot reach these three basecallers' accuracies: they have lower identity and lower coverage.

Basecalling – Accuracy



Observation 1-c: *Scrappie has the highest accuracy with the lowest number of mismatches and indels.*

Basecalling – Speed

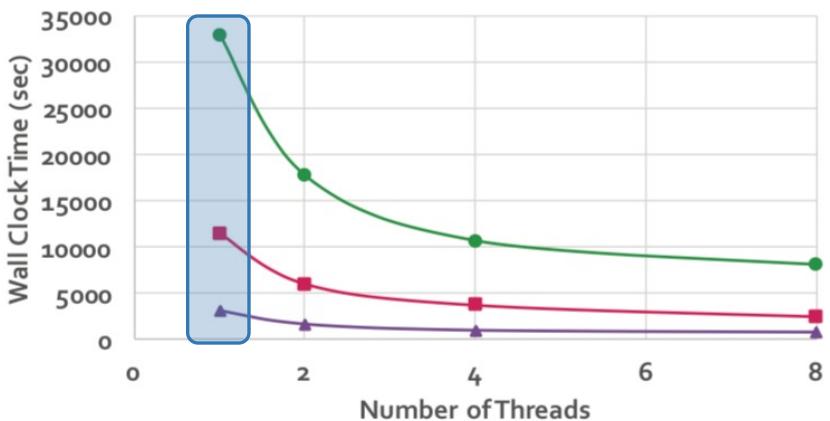


Observation 2: RNN-based basecallers, Nanonet and Scrappie are faster than HMM-based basecaller, Nanocall.

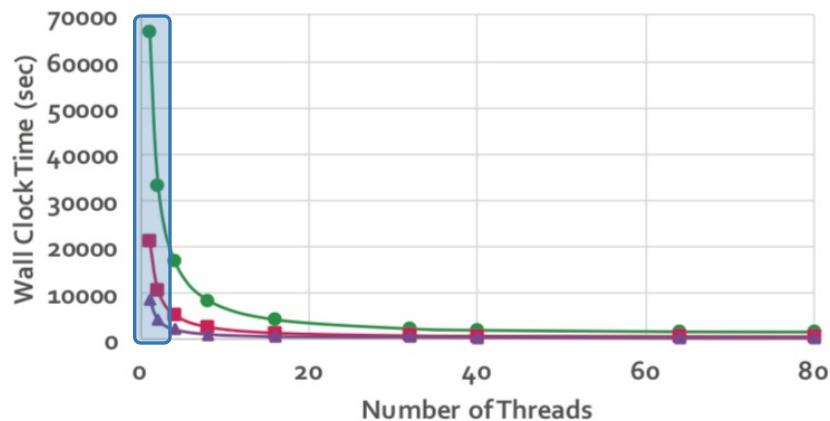
Basecalling – Speed

● Nanocall ■ Nanonet ▲ Scrappie

Nanocall vs. Nanonet vs. Scrappie @desktop



Nanocall vs. Nanonet vs. Scrappie @big-mem

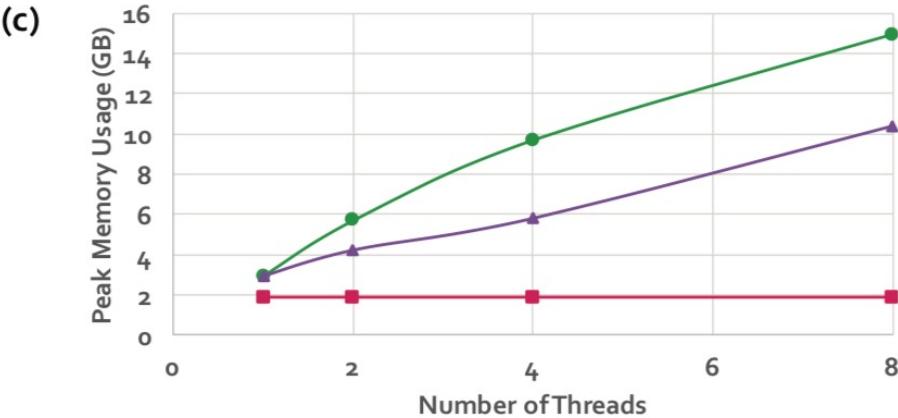


Observation 3: When $\#threads=1$, desktop is approximately 2x faster than big-mem because of desktop's higher CPU frequency. It is an indication that all of these three tools are computationally expensive.

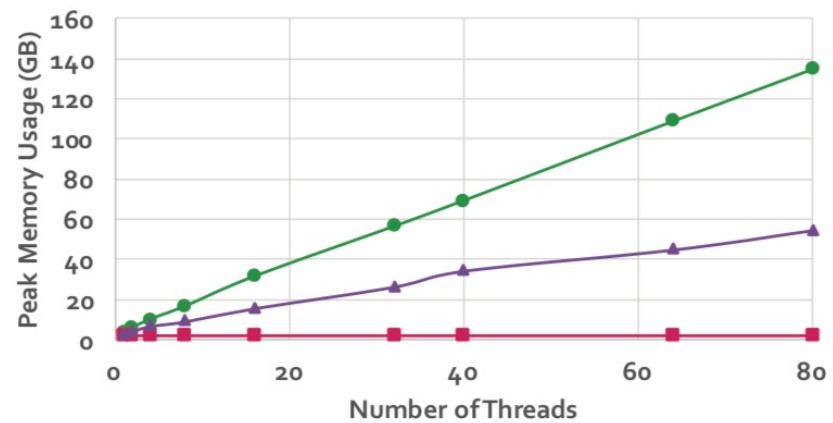
Basecalling – Memory

● Nanocall ■ Nanonet ▲ Scrappie

Nanocall vs. Nanonet vs. Scrappie @desktop



Nanocall vs. Nanonet vs. Scrappie @big-mem

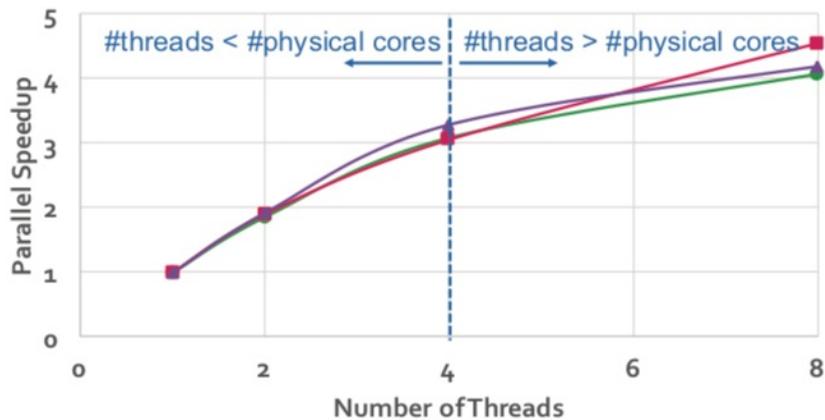


Observation 4: *Scrappie and Nanocall have a linear increase in memory usage when number of threads increases. In contrast, Nanonet has a constant memory usage for all evaluated thread units.*

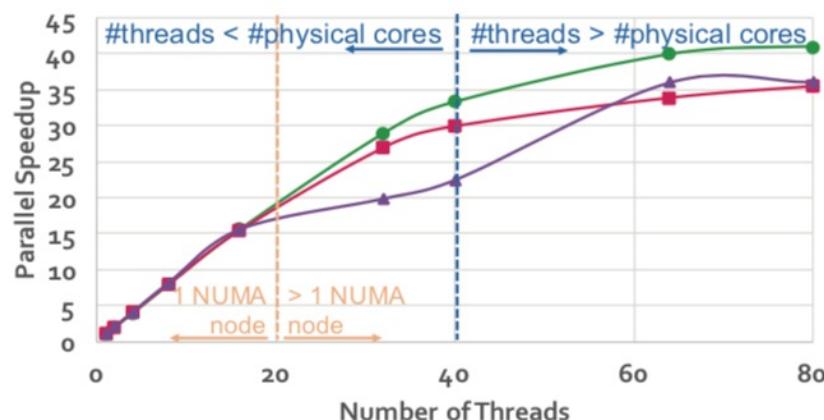
Basecalling – Speedup

● Nanocall ■ Nanonet ▲ Scrappie

Nanocall vs. Nanonet vs. Scrappie @desktop



Nanocall vs. Nanonet vs. Scrappie @big-mem



Observation 5: When the number of threads exceeds the number of physical cores, the simultaneous multithreading overhead prevents continued linear speedup of Nanonet, Scrappie and Nanocall because of the CPU-intensive workload of these tools.

Basecalling – Key Observations

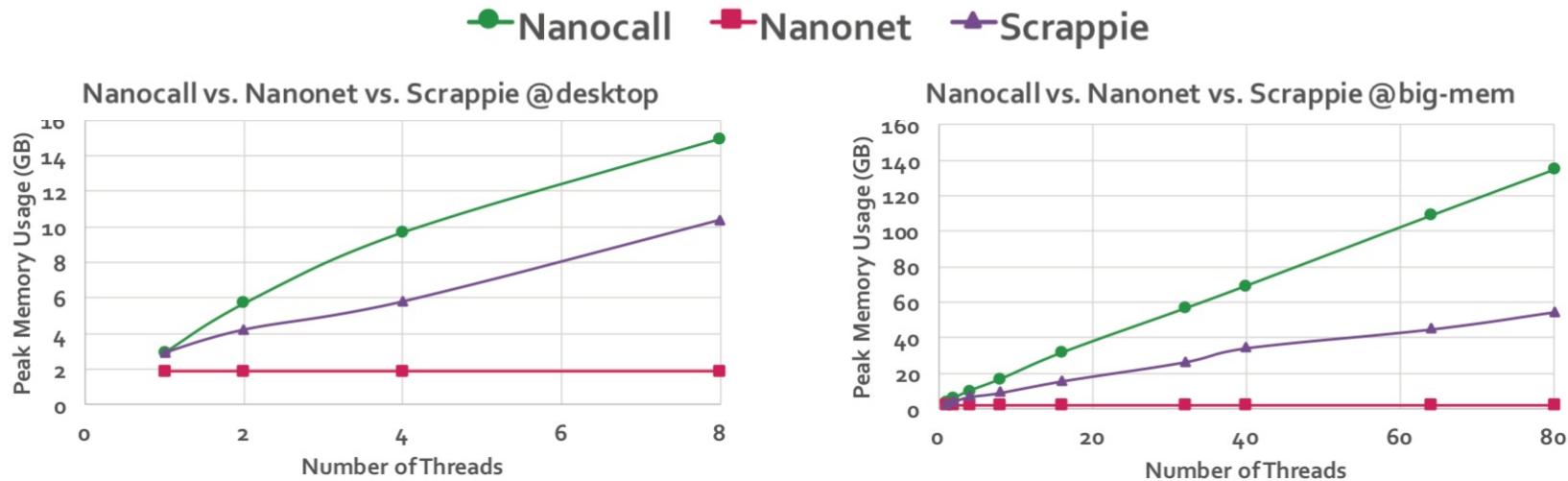
Accuracy:

- ❑ ONT's basecallers (i.e., Metrichor, Nanonet and Scrappie) have similar identity and coverage trends among all of the evaluated scenarios. However, other two basecallers (i.e., Nanocall and DeepNano) cannot reach these three basecallers' accuracies: they have lower identity and lower coverage
- ❑ Scrappie has the highest accuracy with the **lowest number of mismatches and indels**

Performance:

- ❑ RNN-based basecallers are faster than HMM-based basecaller

Basecalling – Key Observations



Memory Usage:

- Scappie and Nanocall have a linear increase in memory usage when number of threads increases. In contrast, Nanonet has a constant memory usage for all evaluated thread units

Scalability:

- Data sharing between threads degrades the parallel speedup of Nanonet when cores from multiple NUMA nodes take role in the computation

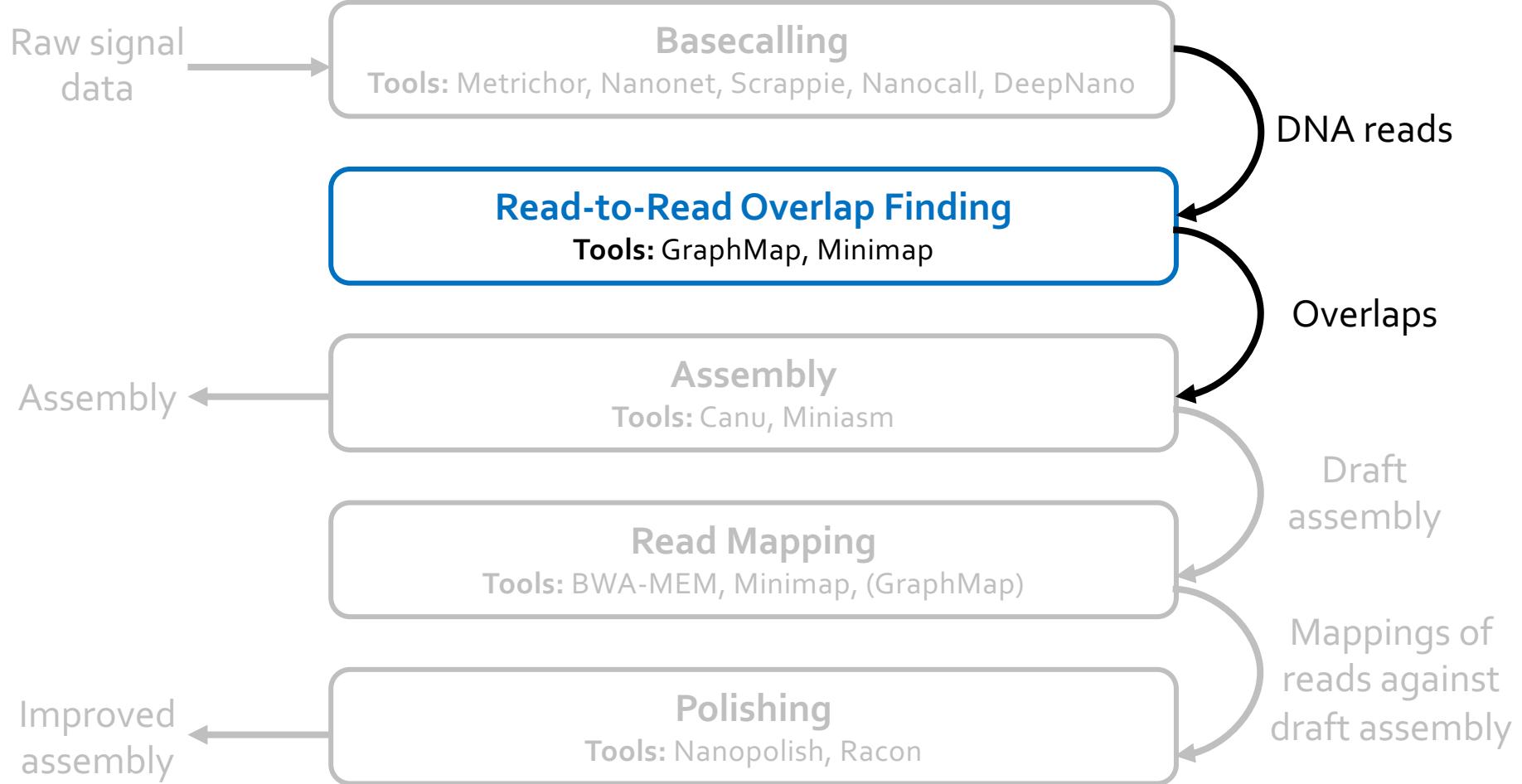
Basecalling – Summary

- The choice of the tool for the basecalling step plays an important role to overcome the high error rates of nanopore sequencing technology.

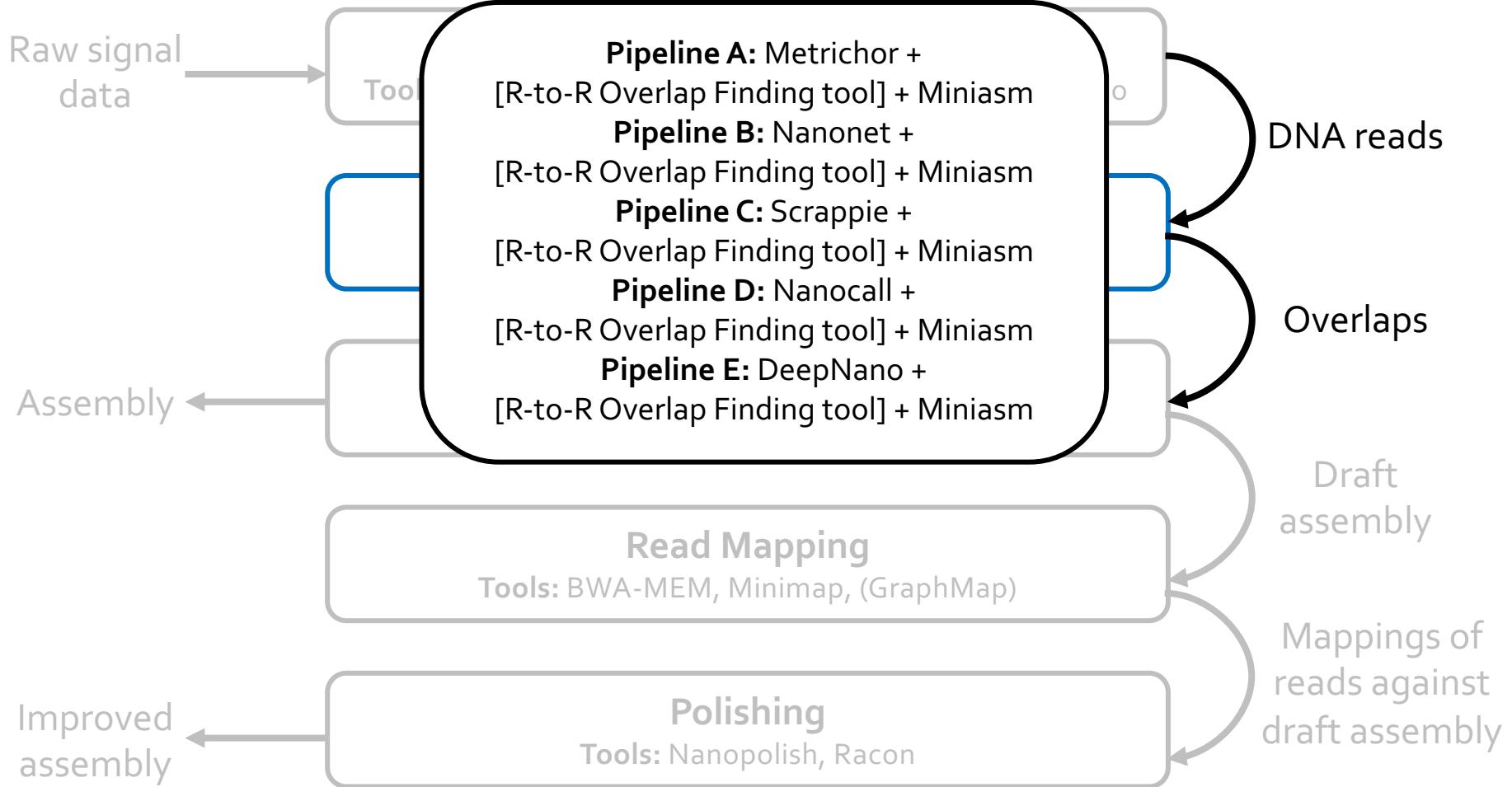
- Basecalling with RNNs (e.g. Metrichor, Nanonet, Scrappie) provides higher accuracy and higher speed than basecalling with HMMs.

- The newest basecaller of ONT, Scrappie, also has the potential to overcome the homopolymer basecalling problem.

Nanopore Genome Assembly Pipeline

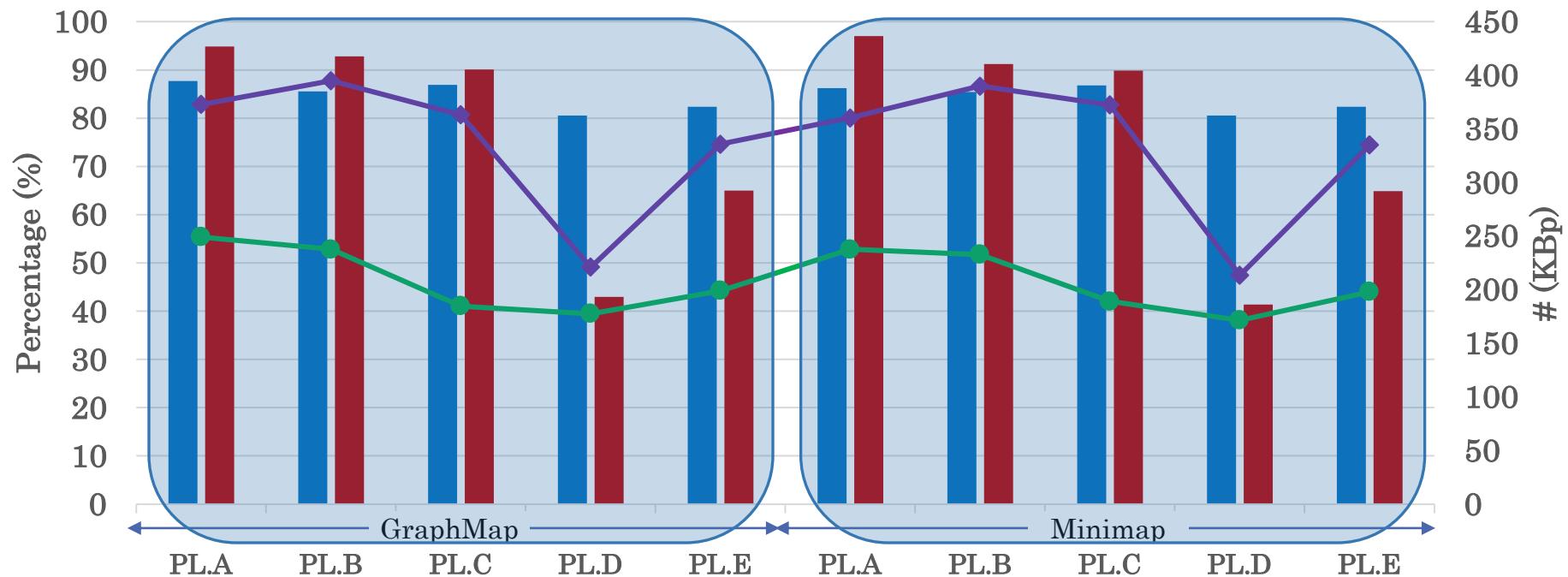


Nanopore Genome Assembly Pipeline



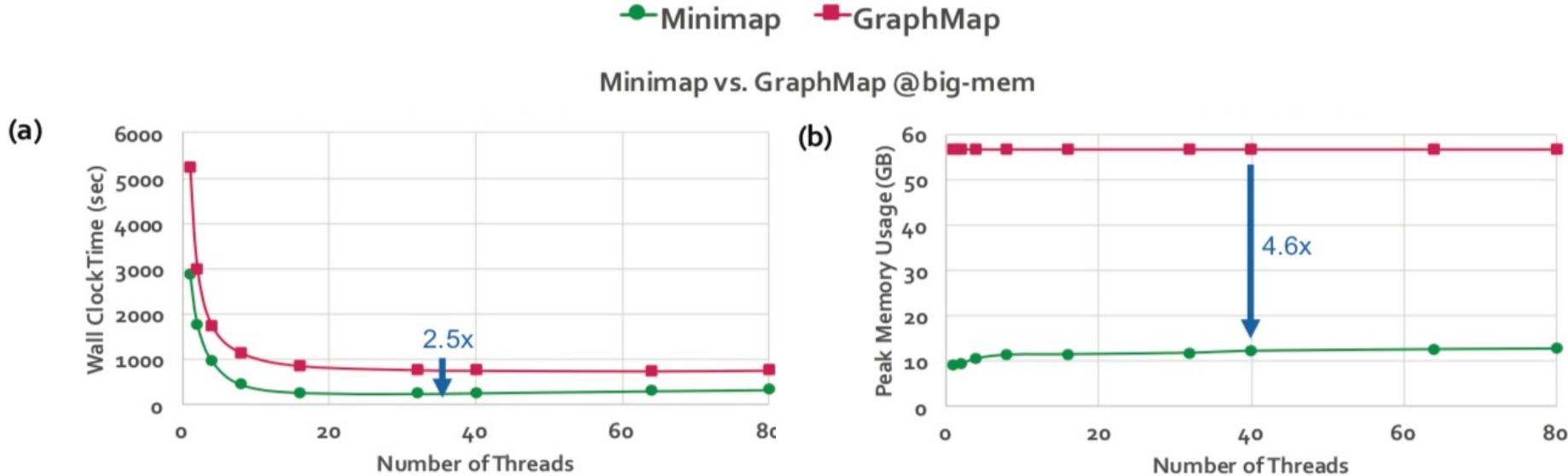
R-to-R Overlap Finding – Accuracy

Accuracy Analysis Results for Read-to-Read Overlap Finding Tools



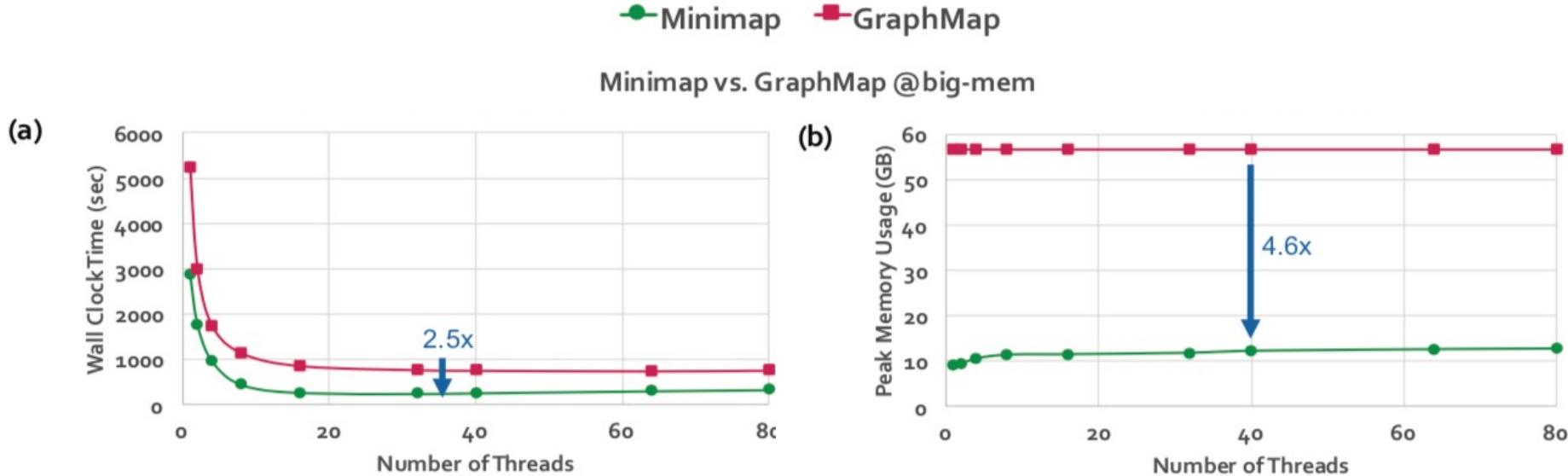
Observation 5: Pipelines with GraphMap or Minimap end up with similar accuracy results.

R-to-R Overlap Finding – Performance



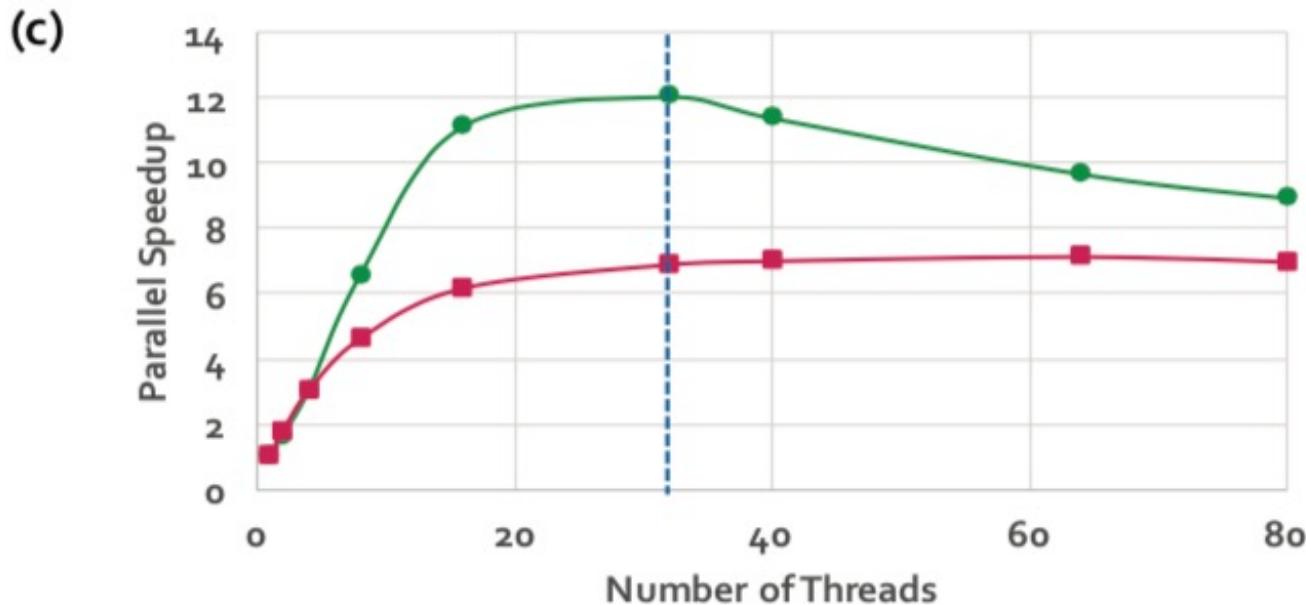
Observation 6: *The memory usage of both GraphMap and Minimap is dependent on the hash table size but independent of number of threads. Minimap requires 4.6x less memory than GraphMap, on average.*

R-to-R Overlap Finding – Performance

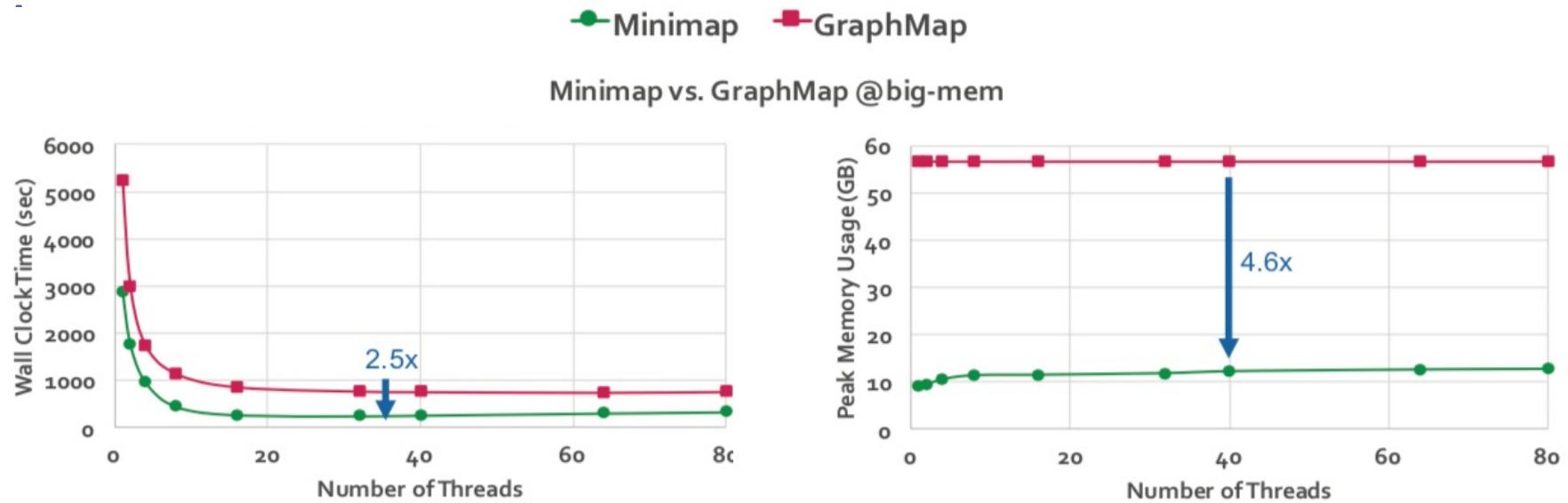


Observation 7: *Minimap is 2.5x faster than GraphMap, on average. Since in Minimap, the size of dataset that needs to be scanned is greatly shrunk by storing minimizers instead of k-mers, it performs much less computation than GraphMap.*

R-to-R Overlap Finding – Speedup



R-to-R Overlap Finding – Key Observations



Memory Usage:

- The memory usage of both GraphMap and Minimap is dependent on the hash table size but independent of number of threads. Minimap requires 4.6x less memory than GraphMap, on average.

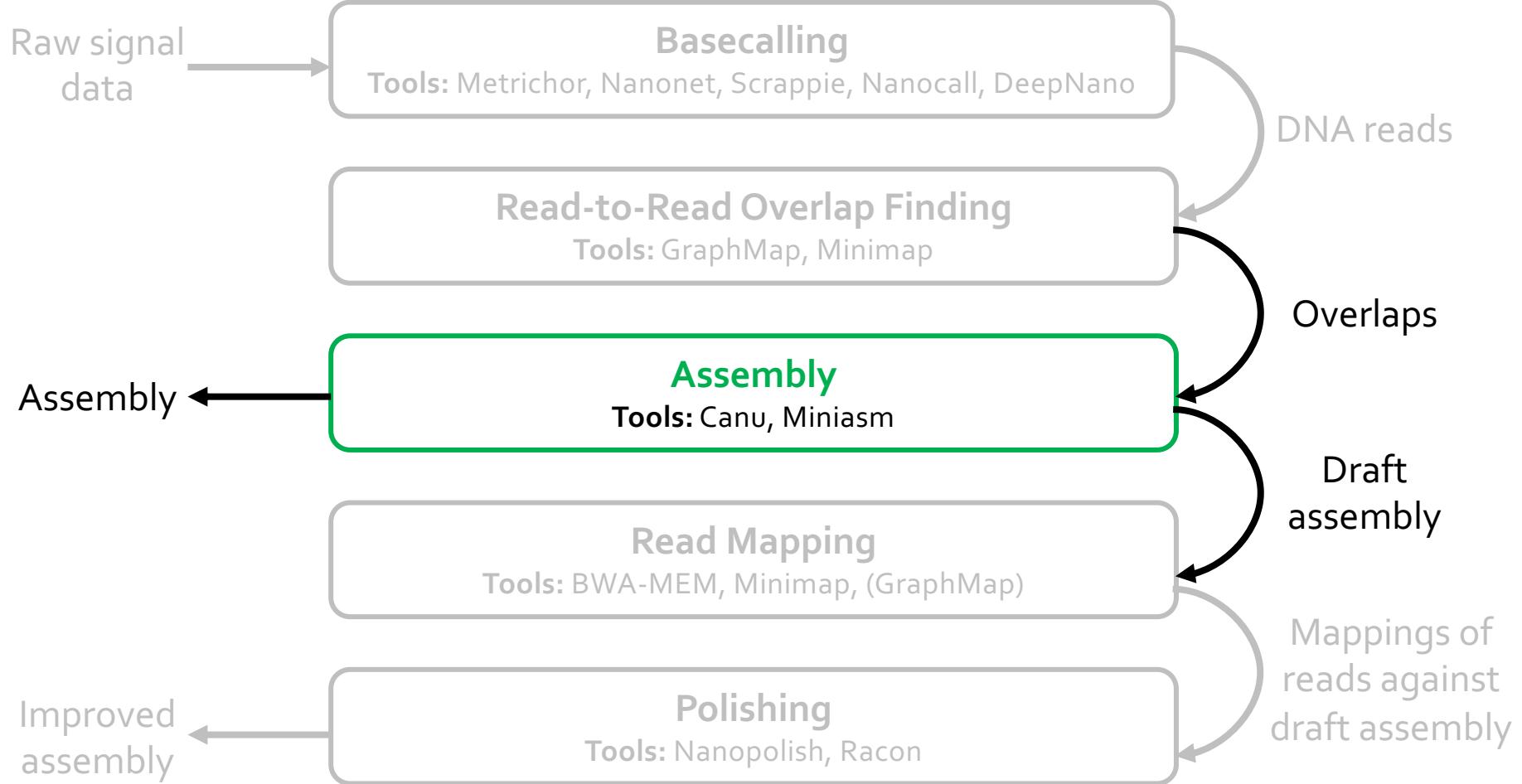
Performance:

- Minimap is 2.5x faster than GraphMap, on average.

R-to-R Overlap Finding – Summary

- Storing minimizers instead of all k -mers, as done by Minimap, does *not* affect the overall accuracy of the first three steps of the pipeline.
- By storing minimizers, Minimap has a much lower memory usage and thus much higher performance than GraphMap.

Nanopore Genome Assembly Pipeline



Assembly – Accuracy & Performance

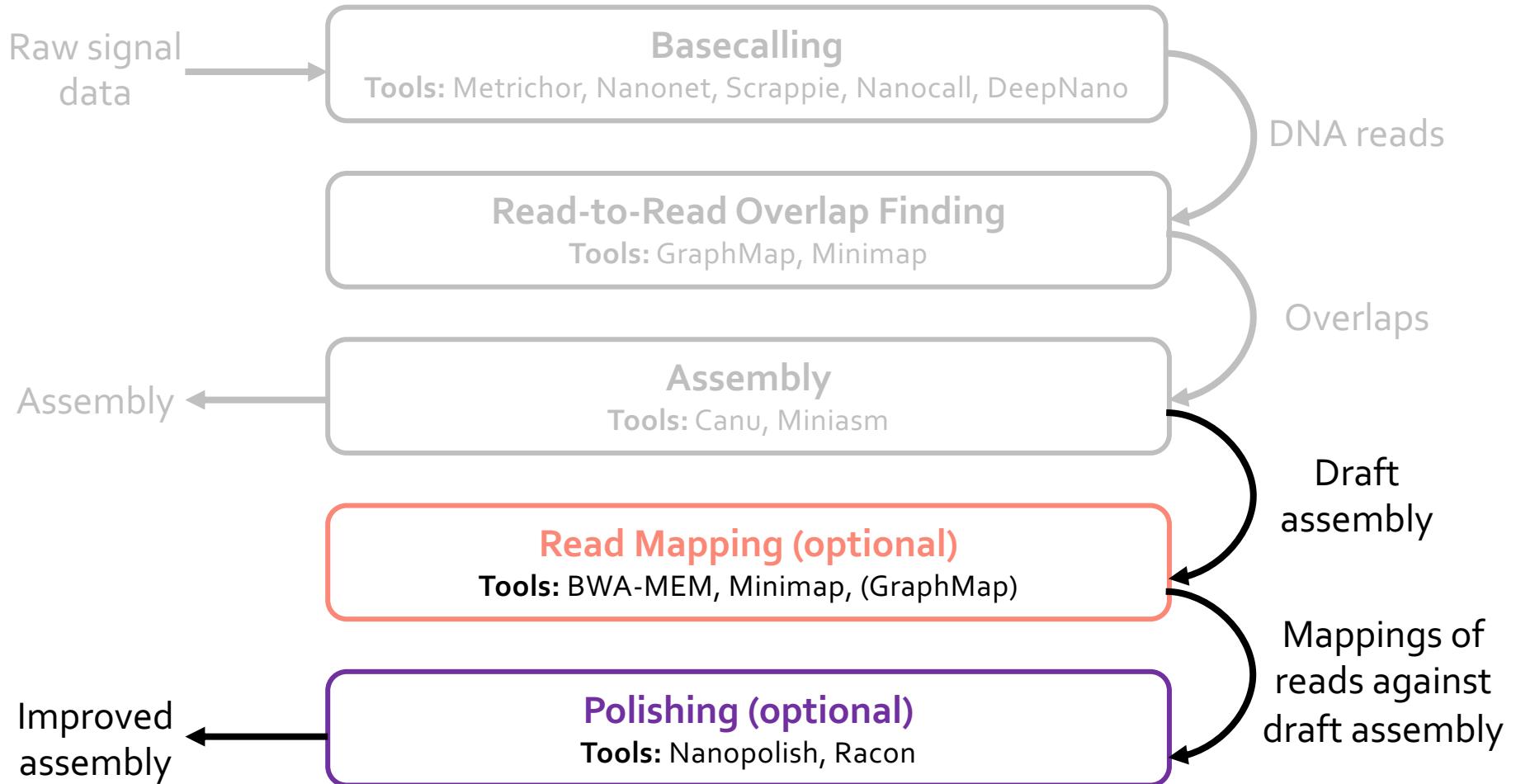
Observation 8: *Canu provides higher accuracy than Miniasm, with the help of the error-correction step that is present in its own pipeline. On average, Canu provides 96.1% identity whereas Miniasm provides 84.4% identity.*

Observation 9: *Canu is much more computationally intensive and greatly (i.e., by 1096.3x) slower than Miniasm, because of its very expensive error-correction step.*

Assembly – Summary

- There is a trade-off between accuracy and performance when deciding on the appropriate tool for the assembly step.
- Canu produces highly accurate assemblies, but it is resource intensive and slow. In contrast, Miniasm is a fast assembler, but it cannot produce as accurate draft assemblies as Canu.
- Miniasm can potentially be used for fast initial analysis and then further polishing can be applied in the next step to produce higher-quality assemblies.

Nanopore Genome Assembly Pipeline



Read Mapping & Polishing – Accuracy

Observation 11: *Both Nanopolish and Racon significantly increase the accuracy of the draft assemblies.*

For example, Nanopolish increases the identity and coverage of the draft assembly generated with the Metrichor+Minimap+Miniasm pipeline from 87.71% and 94.85%, respectively, to 92.33% and 96.31%. Similarly, Racon increases them to 97.70% and 99.91%, respectively.

Observation 12: *For Racon, the choice of read mapper does not affect the accuracy of the polishing step.*

Read Mapping & Polishing – Speed

Observation 13: *Nanopolish is computationally much more intensive and thus greatly slower than Racon.*

Nanopolish runs take days to complete whereas Racon runs take minutes. This is mainly because Nanopolish works on each base individually, whereas Racon works on the windows. Since each window is much longer (i.e., 20kb) than a single base, the computational workload is greatly smaller in Racon.

Observation 14: *BWA-MEM is computationally more expensive than Minimap.*

Although the choice of BWA-MEM and Minimap for the read mapping step does not affect the accuracy of the polishing step, these two tools have a significant difference in performance.

Read Mapping & Polishing – Summary

- ❑ Further polishing can significantly increase the accuracy of the assemblies.
- ❑ Pipelines with Minimap and Racon can provide a significant speedup compared with the pipelines with BWA-MEM and Nanopolish, while resulting with high-quality consensus sequences.

Nanopore Sequencing & Tools [BiB 2018]

Damla Senol Cali, Jeremie S. Kim, Saugata Ghose, Can Alkan, and Onur Mutlu,
**"Nanopore Sequencing Technology and Tools for Genome Assembly:
Computational Analysis of the Current State, Bottlenecks and Future
Directions."**

Briefings in Bioinformatics, April 2018.

Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions

**Damla Senol Cali^{1,*}, Jeremie S. Kim^{1,3}, Saugata Ghose¹, Can Alkan^{2*}
and Onur Mutlu^{3,1*}**

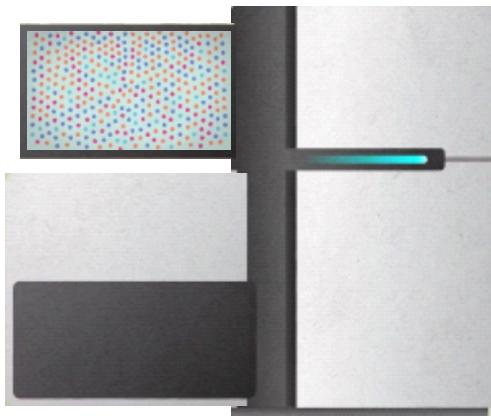
¹Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA

²Department of Computer Engineering, Bilkent University, Bilkent, Ankara, Turkey

³Department of Computer Science, Systems Group, ETH Zürich, Zürich, Switzerland

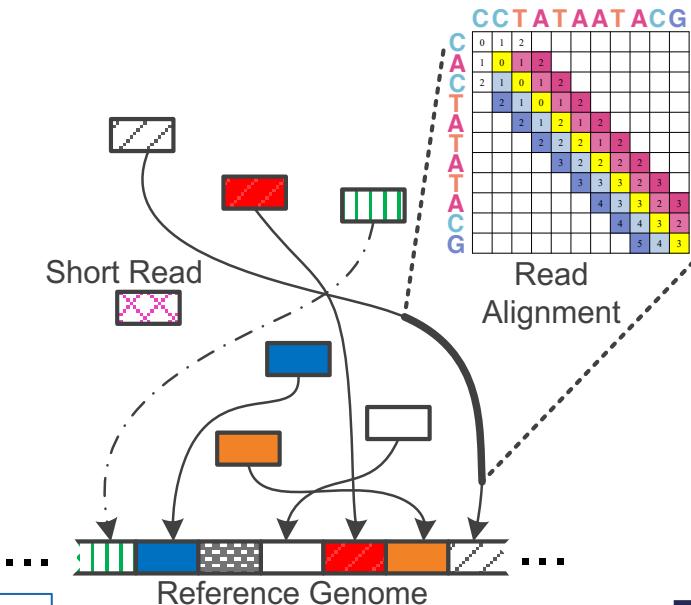
Backup Slides

(GenASM)



Billions of Short Reads

```
ATATATAACGTACTAGTACGT  
TTTAGTACGTACGT  
ATACGTACTAGTACGT  
ACG CCCCTACGTA  
ACGTACTAGTACGT  
TTAGTACGTACGT  
TACGTACTAAAGTACGT  
TACGTACTAGTACGT  
TTTAAAAACGTA  
CGTACTAGTACGT  
GGGAGTACGTACGT
```



1 Sequencing

Genome Analysis

2 Read Mapping

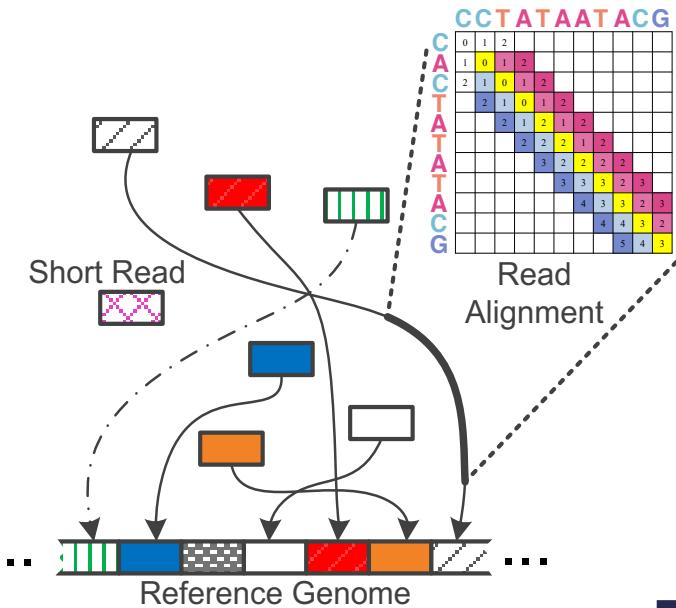
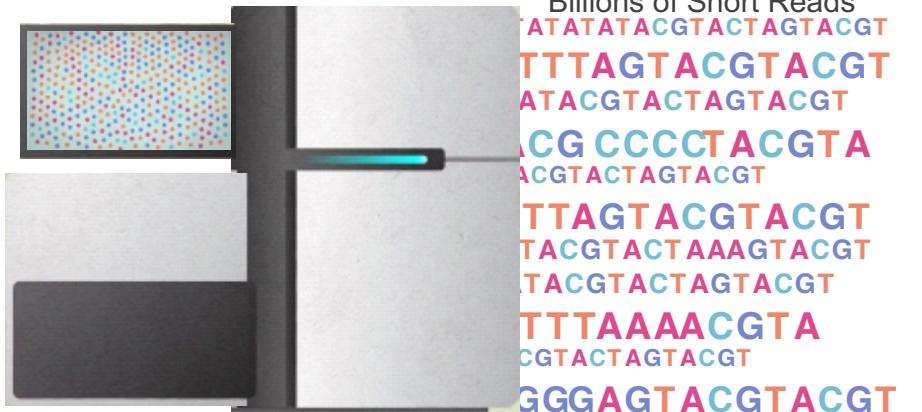
reference: TTTATCGCTTCATGACGCAG

read1:	ATCGC ATCC
read2:	TATCGC ATC
read3:	CATCC ATGA
read4:	CGCTTCCAT
read5:	CCATGACGC
read6:	TTCCATGAC



3 Variant Calling

Scientific Discovery 4

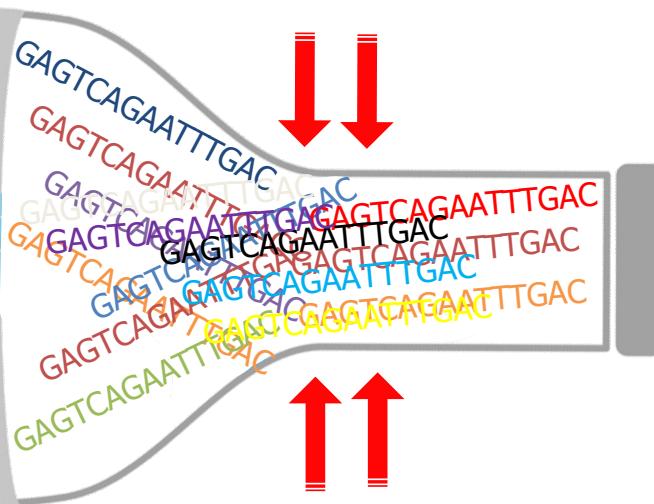


1 Sequencing

2 Read Mapping

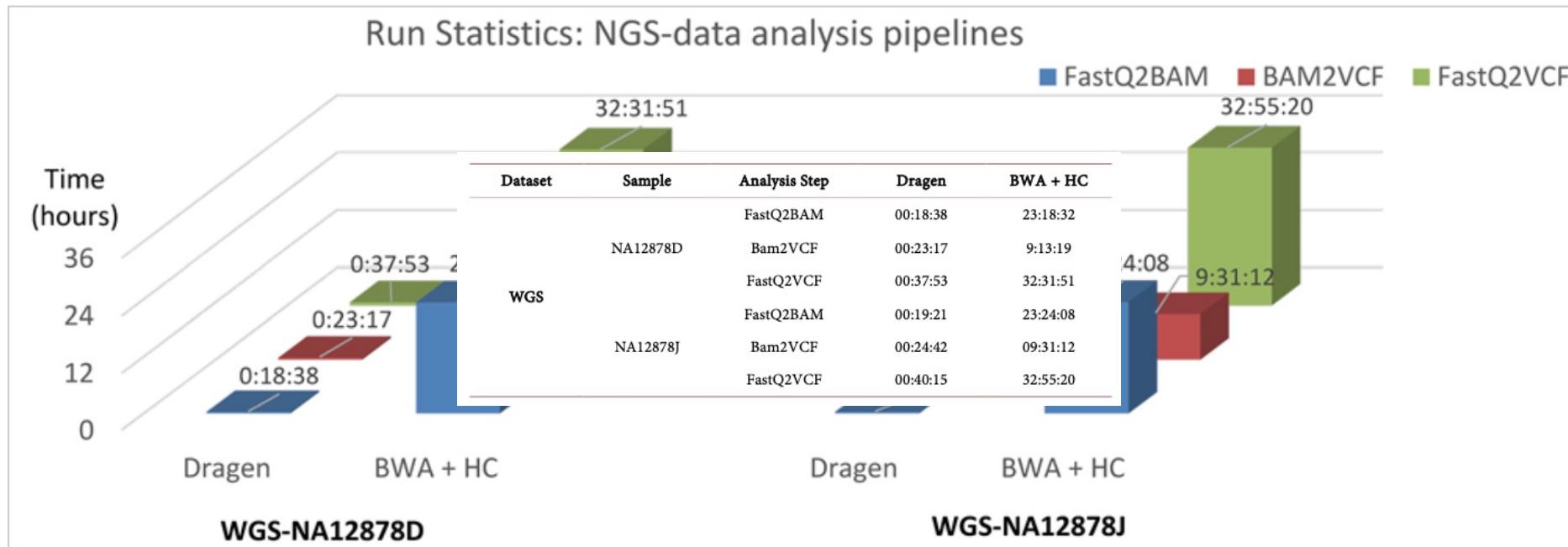
Bottlenecked in Mapping!!

Illumina HiSeq4000
300 M
bases/min



on average
2 M
bases/min
(0.6%)

Illumina's DRAGEN



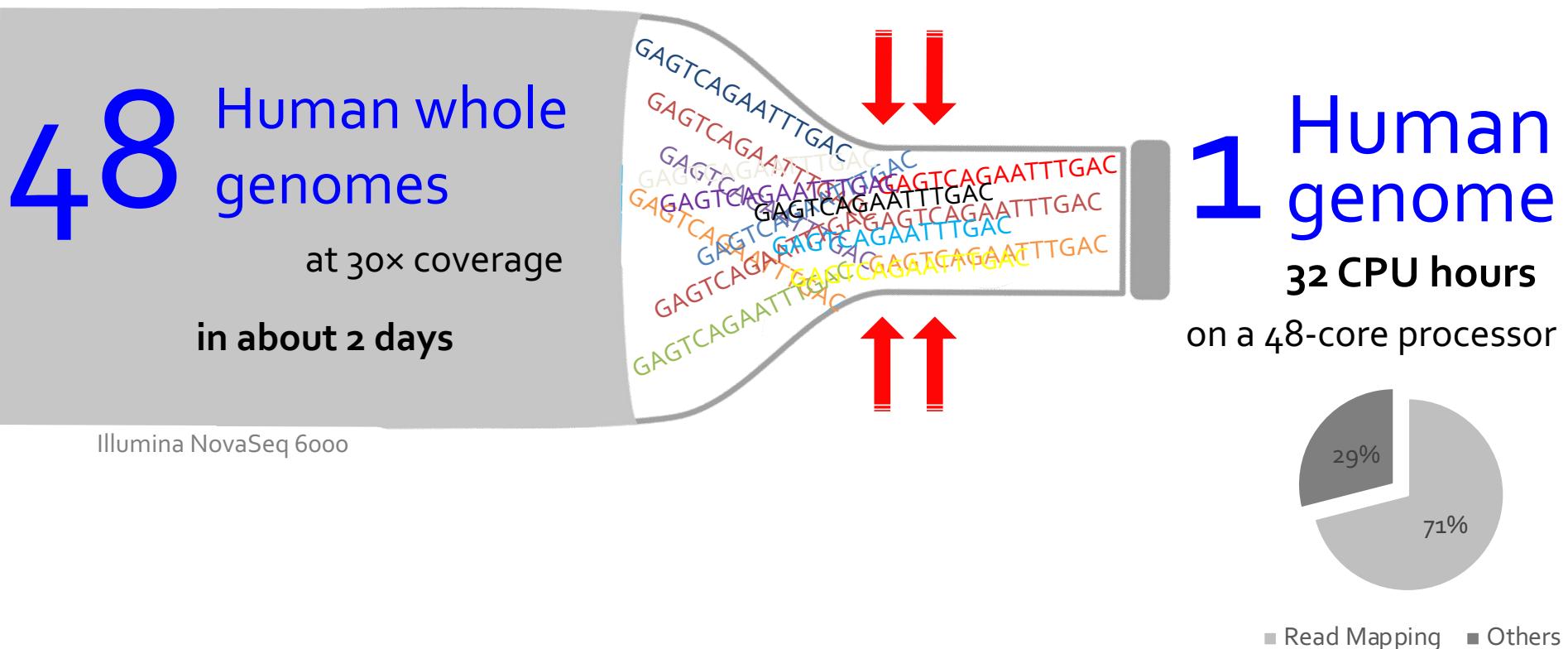
Goyal+, "[Ultra-fast next generation human genome sequencing data processing using DRAGENTM bio-IT processor for precision medicine](#)", *Open Journal of Genetics*, 2017.

Illumina's DRAGEN

Dataset	Sample	Analysis Step	Dragen	BWA + HC
WGS	NA12878D	FastQ2BAM	00:18:38	23:18:32
		Bam2VCF	00:23:17	9:13:19
		FastQ2VCF	00:37:53	32:31:51
	NA12878J	FastQ2BAM	00:19:21	23:24:08
		Bam2VCF	00:24:42	09:31:12
		FastQ2VCF	00:40:15	32:55:20

Goyal+, "[Ultra-fast next generation human genome sequencing data processing using DRAGENTM bio-IT processor for precision medicine](#)", *Open Journal of Genetics*, 2017.

Bottlenecked in Read Mapping!!

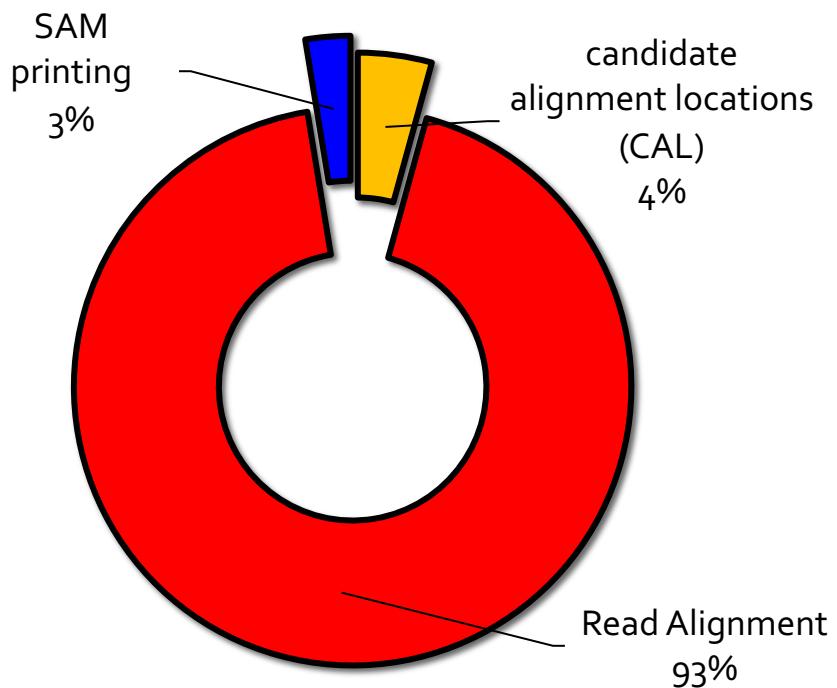


Goyal+, "[Ultra-fast next generation human genome sequencing data processing using DRAGENTM bio-IT processor for precision medicine](#)", *Open Journal of Genetics*, 2017.

What Makes Read Mapper Slow?

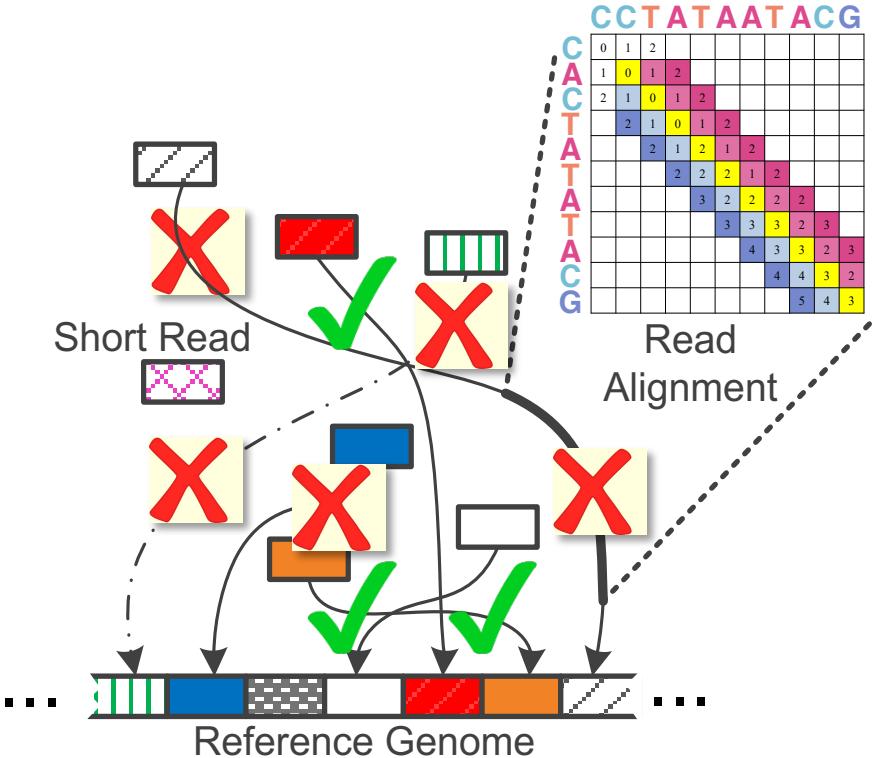
93%

of the read mapper's
execution time is spent in
read alignment.



Alser et al, Bioinformatics (2017)

What Makes Read Mapper Slow? (cont'd.)



98%
of candidate locations
have high dissimilarity
with a given read.

Cheng *et al*, BMC bioinformatics (2015)
Xin *et al*, BMC genomics (2013)

What Makes Read Mapper Slow? (cont'd.)

- ❑ **Quadratic-time** dynamic-programming algorithm

Enumerating all possible prefixes

- **Data dependencies** limit the computation parallelism

Processing row (or column) after another

- **Entire matrix** is computed even though strings can be dissimilar.

Number of differences is computed only at the backtracking step.

		N	E	T	H	E	R	L	A	N	D	S
	0	1	2	3	4	5	6	7	8	9	10	11
S	1	1	2	3	4	5	6	7	8	9	10	10
W	2	2	2	3	4	5	6	7	8	9	10	11
I	3	3	3	3	4	5	6	7	8	9	10	11
T	4	4	4	3	4	5	6	7	8	9	10	11
Z	5	5	5	4	4	5	6	7	8	9	10	11
E	6	6	5	5	5	4	5	6	7	8	9	10
R	7	7	6	6	6	5	4	5	6	7	8	9
L	8	8	7	7	7	6	5	4	5	6	7	8
A	9	9	8	8	8	7	6	5	4	5	6	7
N	10	9	9	9	9	8	7	6	5	4	5	6
D	11	10	10	10	10	9	8	7	6	5	4	5



Approximate String Matching (ASM)

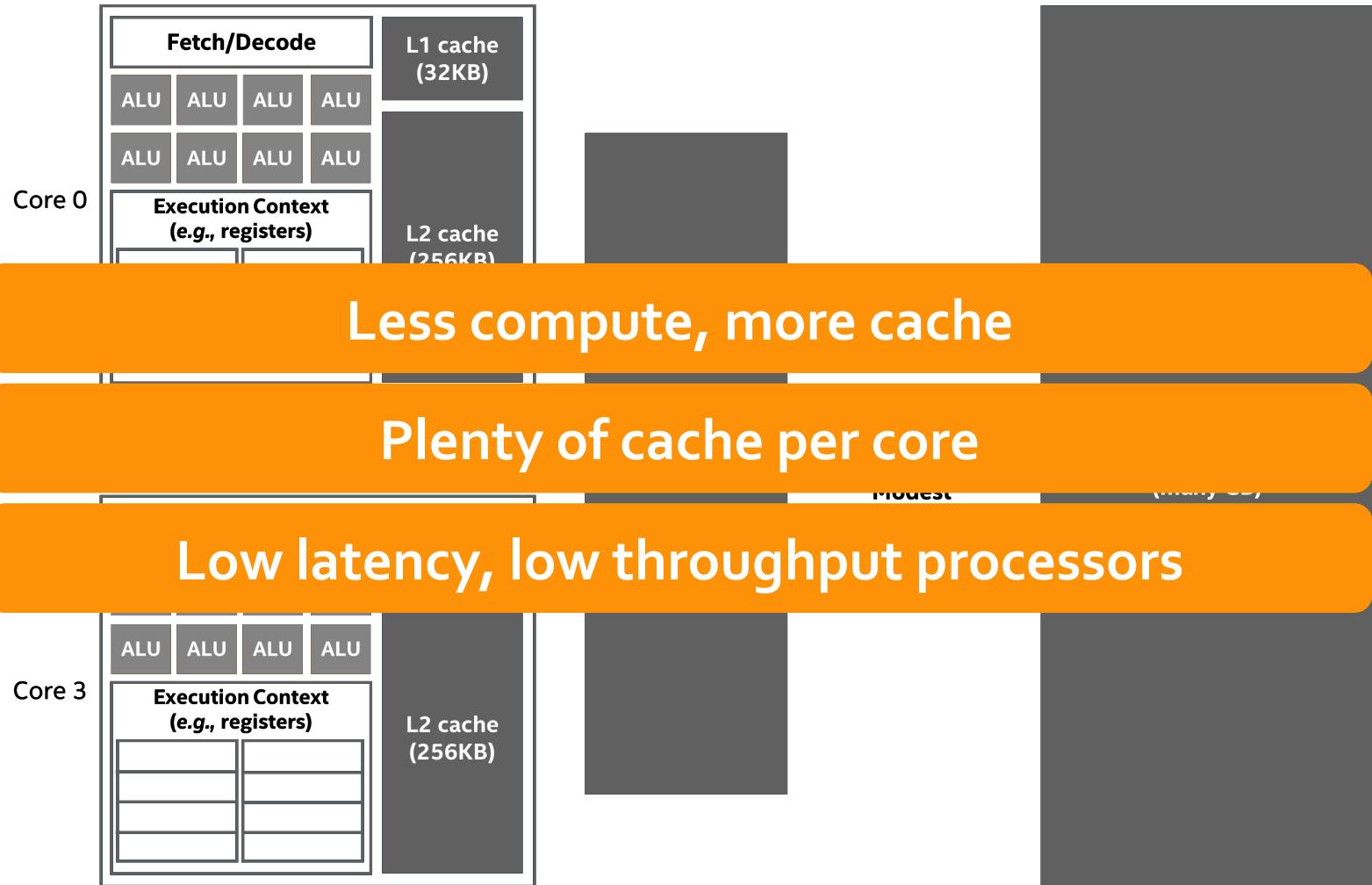
Approximate string matching algorithms:

- *Smith-Waterman (SW)* algorithm [Smith+, Advances in Applied Mathematics 1981]
 - Dynamic programming (DP) algorithm, with quadratic time and space complexity
 - Common algorithm used by read mappers
- *Myers' bitvector* algorithm [Myers, Journal of the ACM 1999]
 - Transformed version of SW algorithm into bitvectors and bitwise operations
- *Bitap* algorithm [Baeza-Yates+, Communications of the ACM 1992]
 - [Wu+, Communications of the ACM 1992] extended *Bitap* to perform approximate string matching
 - Bitvectors and bitwise operations

We have focused on the *Bitap* algorithm.

→ Reason: *Bitap* algorithm can perform ASM with **fast and simple bitwise operations**, which makes it amenable to efficient hardware acceleration.

CPU Systems

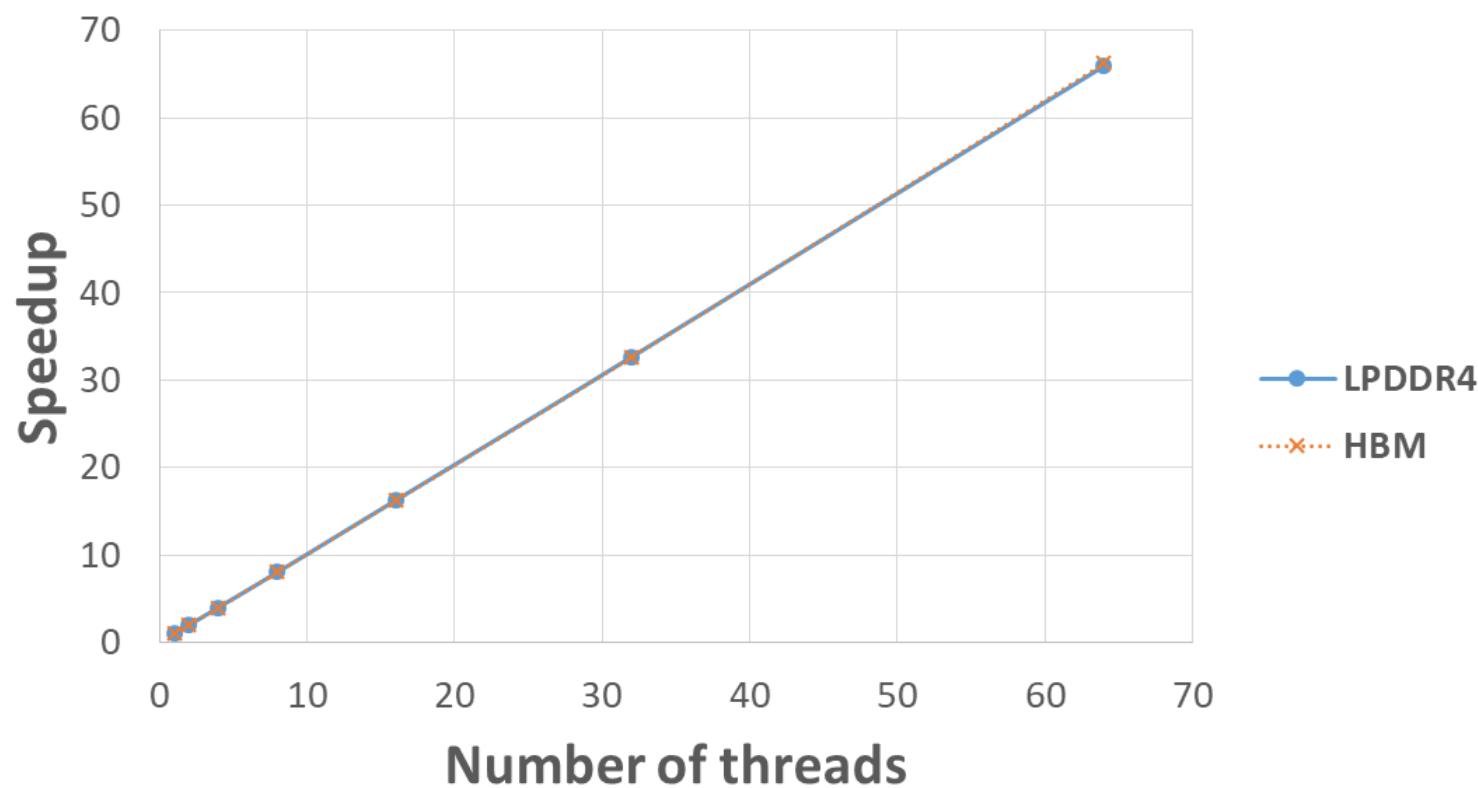


Evaluation Methodology (CPU-bitap)

- Vtune analysis on a real system
 - System Configuration:
 - Intel Core i5-6600K CPU @ 3.50GHz (Skylake)
 - Single socket, 4 physical cores, 1 thread per core
 - 32KB L1 private caches, 256KB L2 private caches, 6MB shared LLC
 - 32GB main memory
 - Analysis Details:
 - HPC performance characterization
 - Hardware events for MPKIs and cache hit/miss rates of each level of cache
 - Hotspot analysis

- Gem5 + Ramulator Simulations
 - Gem5 Configuration:
 - CPU type: O3 (detailed)
 - Number of cores= Number of threads = 1, 2, 4, 8, 16, 32, 64
 - Private L1 size = 64KB each
 - Private L2 size = 512KB each
 - Shared L3 size = # cores * 1MB
 - Main memory type = LPDDR4 vs. HBM
 - Main memory size = 16GB
 - Analysis Details:
 - Execution-driven simulation
 - Scalability, memory-intensity (cache usage, memory bandwidth, and memory latency) and possible bottlenecks analysis
 - With and without L2/L3 caches

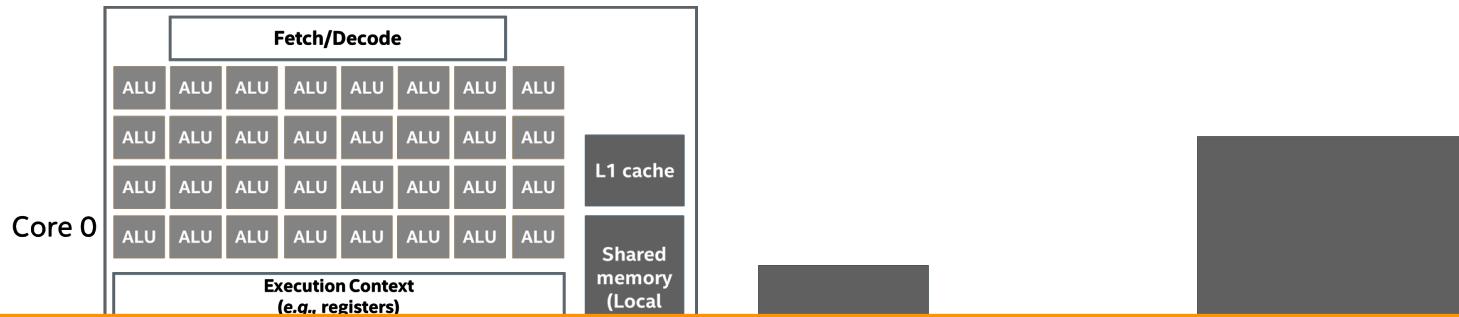
Results (CPU-bitap)



Results (CPU-bitap)

- *CPU-bitap* is very **compute-intensive** and not memory-bound.
- Lots of computation for one byte of data movement
 - L₁-MPKI: 0.196, L₂-MPKI: 0.086, LLC-MPKI: 0.037, and
 - Very high L₁-hit rate (99.895%)
- Adding more cores provides a **linear speedup**
- Since **the working set fits within the registers and the L₁ cache and the number of memory requests is very low**:
 - No performance difference without L₂ and L₃ caches
 - No performance difference between LPDDR4 or HBM as the memory

GPU Systems



More compute, less cache

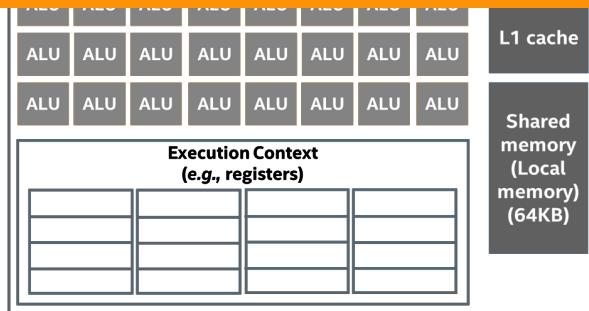
Less cache per core

8 GB

1-4 GB

High latency, high throughput processors

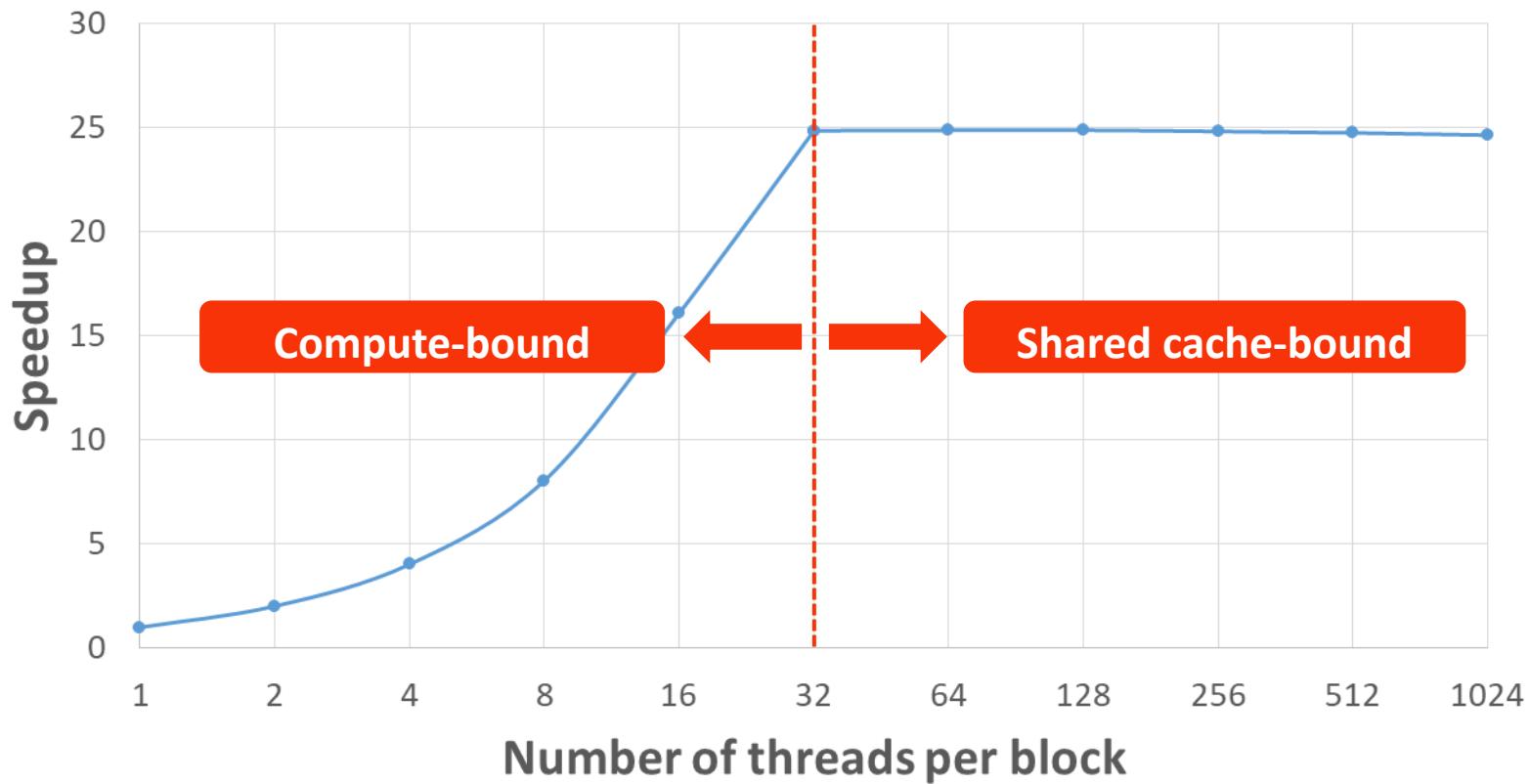
Core 19



Evaluation Methodology (GPU-bitap)

- nvprof analysis on a real system
 - System Configuration:
 - Nvidia Titan V GPU (Volta)
 - 80 multiprocessors * 64 CUDA cores per MP = 5120 CUDA cores
 - L2 cache size = 4.5MB
 - Warp size = 32
 - 12GB HBM2 memory
 - Analysis Details:
 - Events:
 - Elapsed and active cycles
 - Metrics:
 - Branch and warp execution efficiency
 - L2 read/write transactions and throughput
 - DRAM read/write transactions and throughput
 - Stalls (*i.e.*, instruction fetch, execution dependency, memory dependency, and busy compute pipeline)

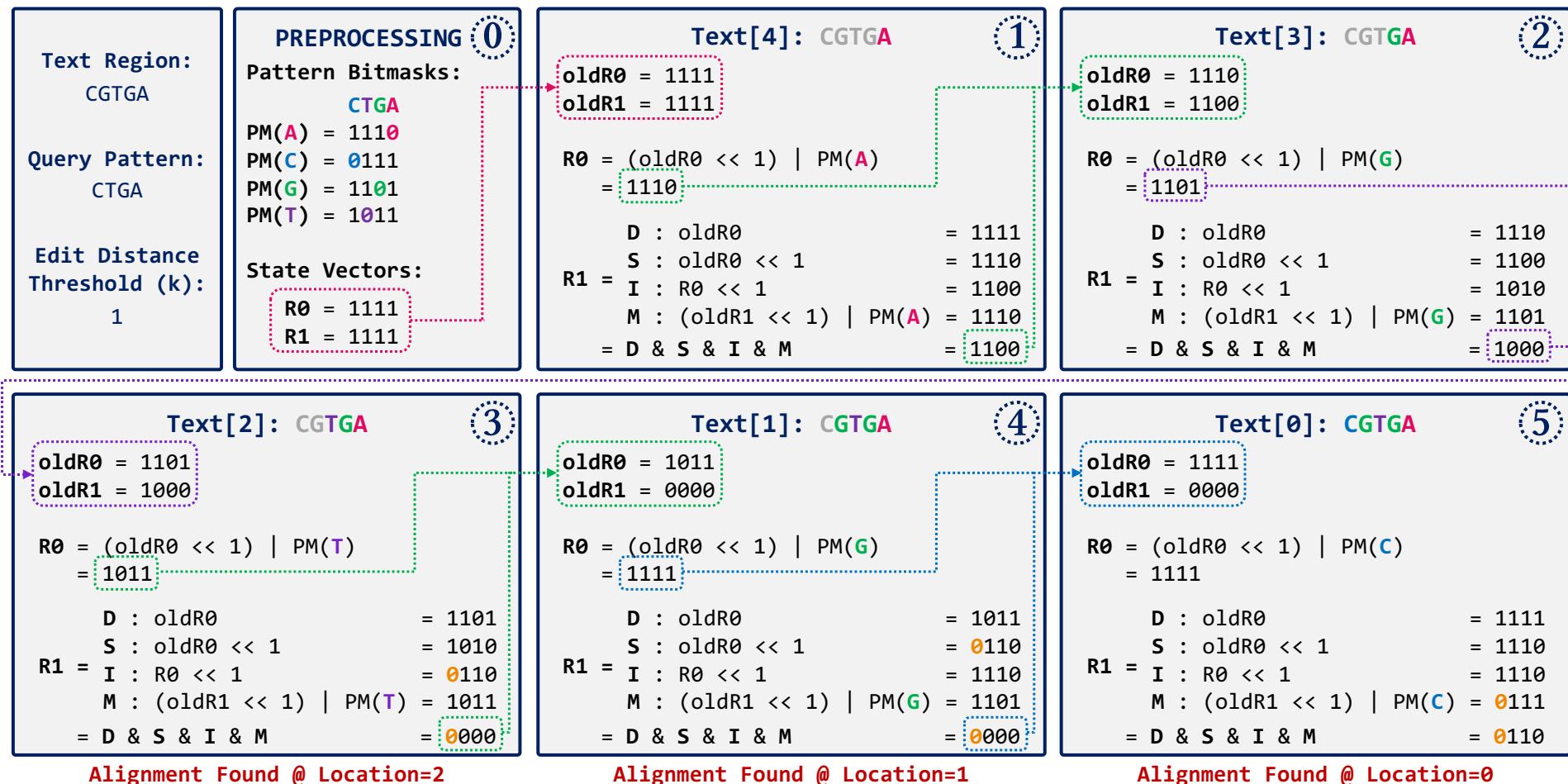
Results (GPU-bitap)



Results (GPU-bitap)

- From 1 thread per block to 32 threads per block,
 - ❑ *GPU-bitap* is **compute-bound**, and
 - ❑ Warp execution efficiency increases from 3% to 100%, linearly.
- *GPU-bitap* is **shared cache-bound** (*i.e.*, on-GPU L2 cache-bound) after number of threads per block reaches 32.
 - ❑ Small number of registers → not enough to hold the frequently used data
 - ❑ Number of L2 read transactions stops decreasing and becomes stable
- Bottlenecks:
 - ❑ Shared memory and L2 cache accesses
 - ❑ Destructive interference of threads

Example for the Bitap Algorithm



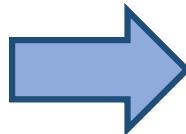
GenASM Algorithm

- **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 Ro/1/2/..
#1	To-Ro
...	...
#8	To-R ₇
#9	T ₁ -Ro
...	...
#16	T ₁ -R ₇
#17	T ₂ -Ro
...	...
#24	T ₂ -R ₇
#25	T ₃ -Ro
...	...
#32	T ₃ -R ₇



Cycle#	Thread ₁ Ro/4	Thread ₂ R ₁ /5	Thread ₃ R ₂ /6	Thread ₄ R ₃ /7
#1	To-Ro	-	-	-
#2	T ₁ -Ro	To-R ₁	-	-
#3	T ₂ -Ro	T ₁ -R ₁	To-R ₂	-
#4	T ₃ -Ro	T ₂ -R ₁	T ₁ -R ₂	To-R ₃
#5	To-R ₄	T ₃ -R ₁	T ₂ -R ₂	T ₁ -R ₃
#6	T ₁ -R ₄	To-R ₅	T ₃ -R ₂	T ₂ -R ₃
#7	T ₂ -R ₄	T ₁ -R ₅	To-R ₆	T ₃ -R ₃
#8	T ₃ -R ₄	T ₂ -R ₅	T ₁ -R ₆	To-R ₇
#9	-	T ₃ -R ₅	T ₂ -R ₆	T ₁ -R ₇
#10	-	-	T ₃ -R ₆	T ₂ -R ₇
#11	-	-	-	T ₃ -R ₇



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

Deletion Example (Text Location=0)

(a)

Text[0]: C	Text[1]: G	Text[2]: T	Text[3]: G	Text[4]: A
$\begin{cases} R0- : \dots \\ R1-M : 0111 \end{cases}$	$\begin{cases} R0- : \dots \\ R1-D : 1011 \end{cases}$	$\begin{cases} R0-M : 1011 \\ R1- : \dots \end{cases}$	$\begin{cases} R0-M : 1101 \\ R1- : \dots \end{cases}$	$\begin{cases} R0-M : 1110 \\ R1- : \dots \end{cases}$
Match(C) <3,0,1>	Del(-) <2,1,1>	Match(T) <2,2,0>	Match(G) <1,3,0>	Match(A) <0,4,0>

Substitution Example (Text Location=1)

(b)

Text[1]: G	Text[2]: T	Text[3]: G	Text[4]: A
$\begin{cases} R0- : \dots \\ R1-S : 0110 \end{cases}$	$\begin{cases} R0-M : 1011 \\ R1- : \dots \end{cases}$	$\begin{cases} R0-M : 1101 \\ R1- : \dots \end{cases}$	$\begin{cases} R0-M : 1110 \\ R1- : \dots \end{cases}$
Subs(C) <3,1,1>	Match(T) <2,2,0>	Match(G) <1,3,0>	Match(A) <0,4,0>

Insertion Example (Text Location=2)

(c)

Text[-]	Text[2]: T	Text[3]: G	Text[4]: A
$\begin{cases} R0- : \dots \\ R1-I : 0110 \end{cases}$	$\begin{cases} R0-M : 1011 \\ R1- : \dots \end{cases}$	$\begin{cases} R0-M : 1101 \\ R1- : \dots \end{cases}$	$\begin{cases} R0-M : 1110 \\ R1- : \dots \end{cases}$
Ins(C) <3,2,1>	Match(T) <2,2,0>	Match(G) <1,3,0>	Match(A) <0,4,0>

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^{◊†▽}

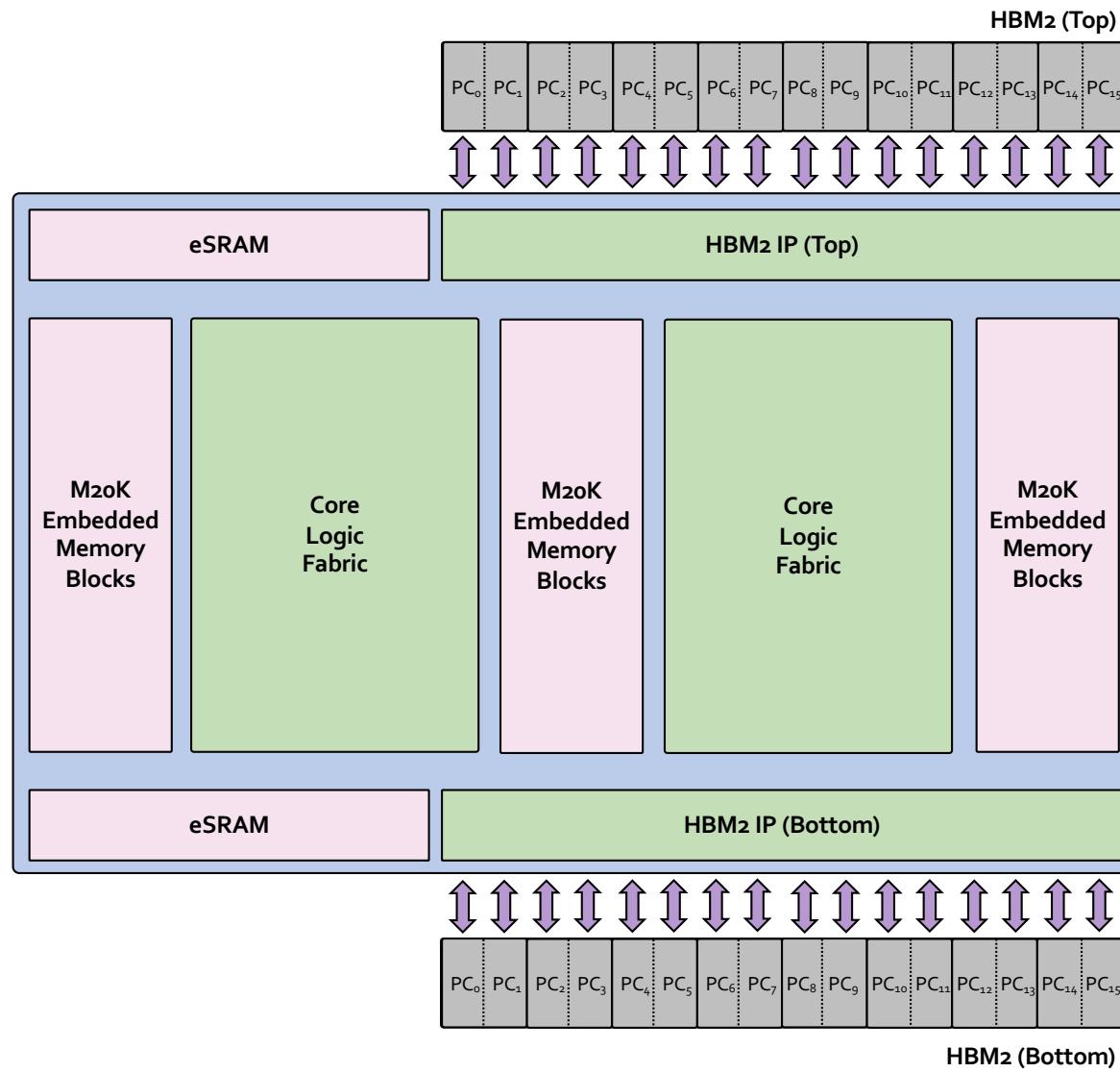
[†]*Carnegie Mellon University* [✉]*Processor Architecture Research Lab, Intel Labs* [▽]*Bilkent University* [◊]*ETH Zürich*

[‡]*Facebook* [○]*King Mongkut's University of Technology North Bangkok* ^{*}*University of Illinois at Urbana-Champaign*

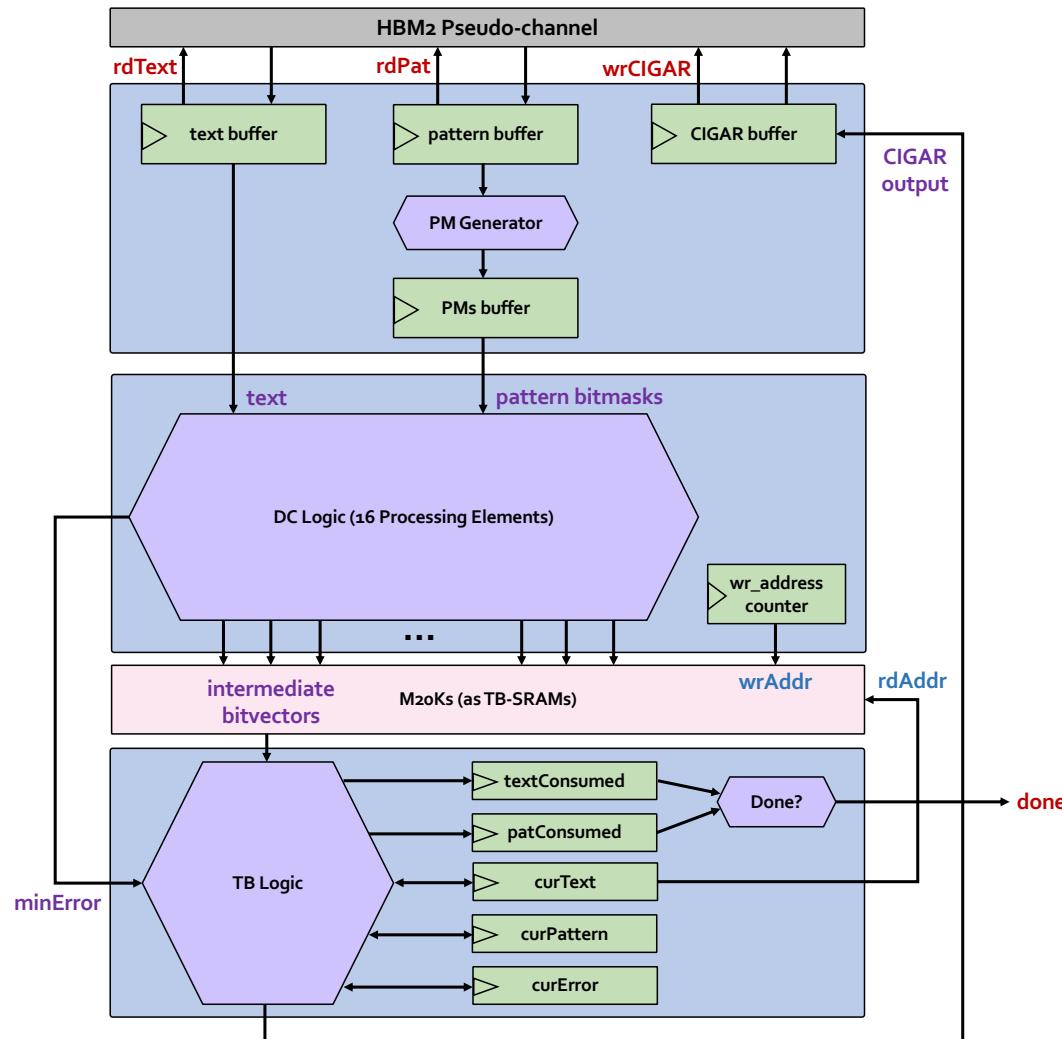
Backup Slides

(BitMAc)

Intel Stratix 10 MX

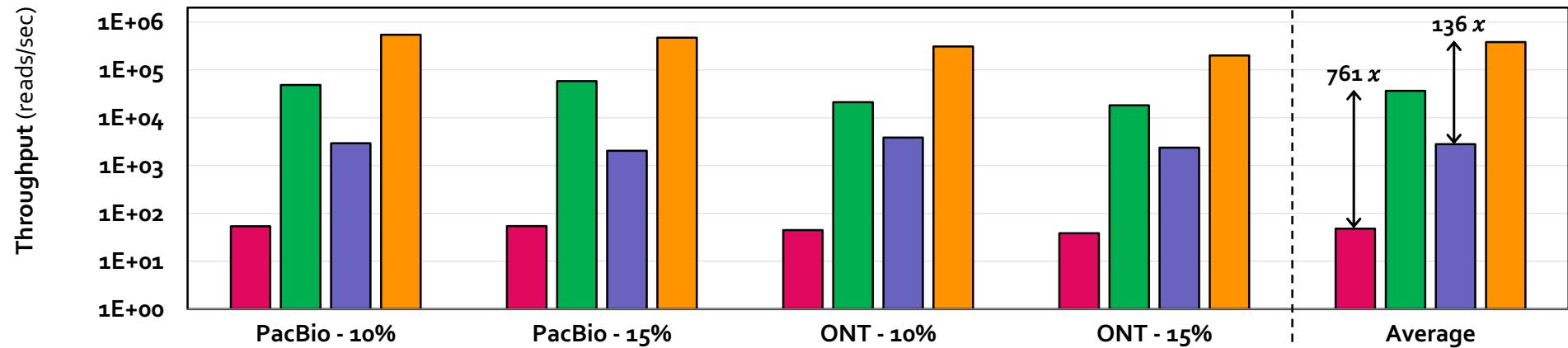


BitMAC Design

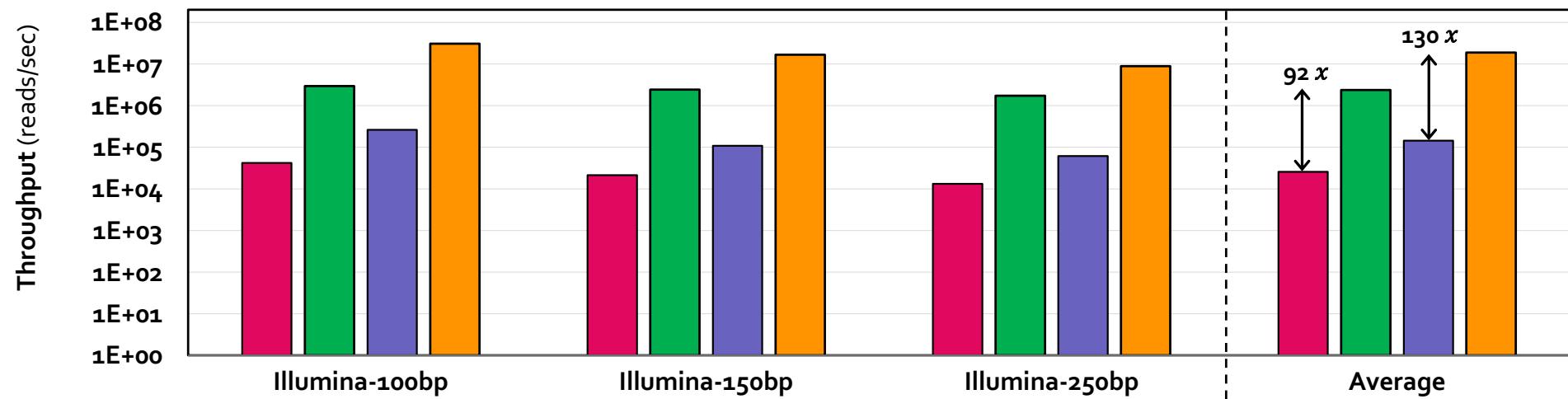


BitMAC – Results

■ BWA-MEM (t=12) ■ BitMAC (w/ pairs from BWA-MEM) ■ Minimap2 (t=12) ■ BitMAC (w/ pairs from Minimap2)



■ BWA-MEM (t=12) ■ BitMAC (w/ pairs from BWA-MEM) ■ Minimap2 (t=12) ■ BitMAC (w/ pairs from Minimap2)



BitMAC – Results

Component	Dynamic On-Chip Power Dissipation	Total On-Chip Power Dissipation
DC Logic (16 PEs)	128.57 mW	
TB Logic	10.24 mW	
FSM Logic	3.15 mW	
M2oKs	211.61 mW	
Other	15.72 mW	
Total – 1 BitMAC Accelerator	369.29 mW (0.4 W)	6043.24 mW (6.0 W)
Total – 32 BitMAC Accelerators (1 per each pseudo-channel)	11569.92 mW (11.6 W)	17234.67 mW (17.2 W)
Total – 128 BitMAC Accelerators (4 per each pseudo-channel)	43042.90 mW (43 W)	48935.65 mW (48.9 W)

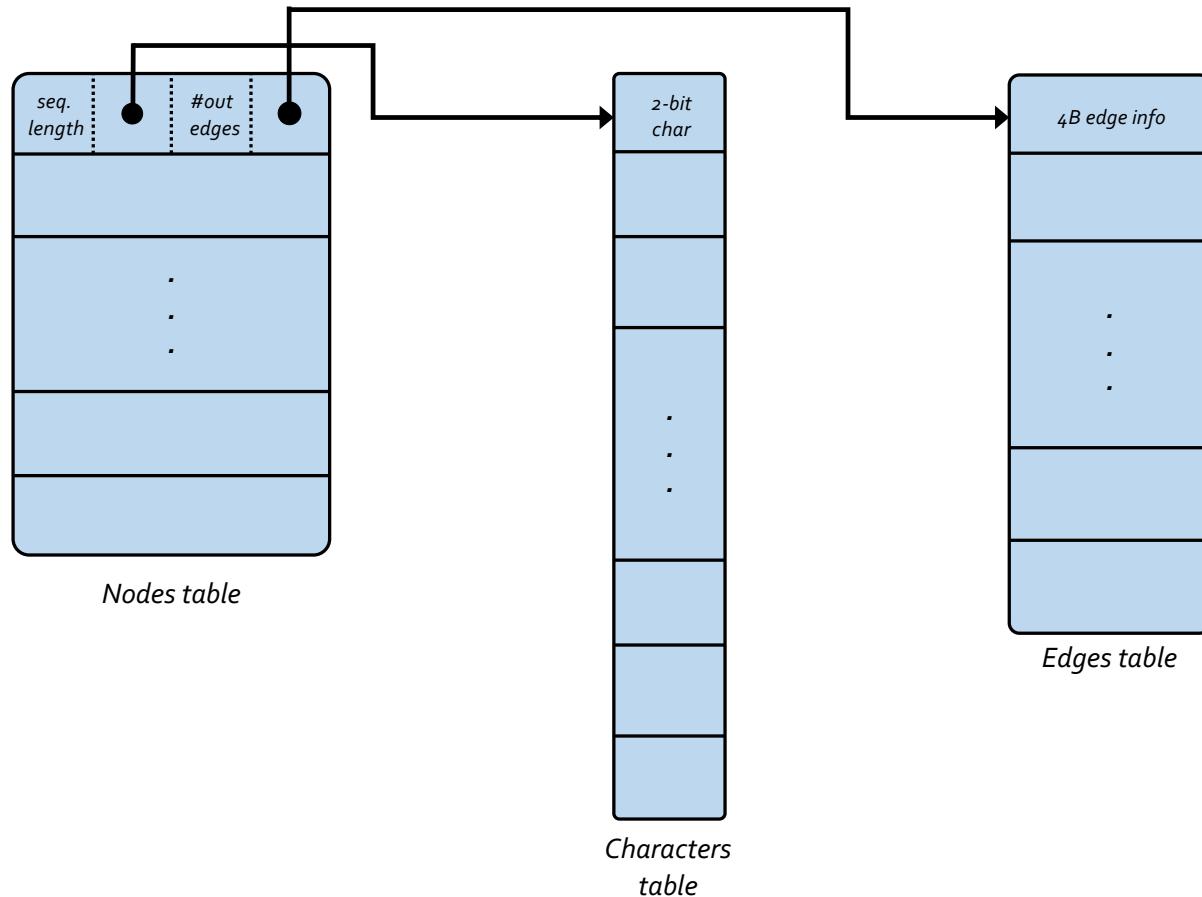
BitMAC – Results

Configuration	Logic Utilization	M2oK	eSRAM	DSP
1 BitMAC Accelerator	0.5%	0.7%	0%	0%
32 BitMAC Accelerators (1 per each pseudo-channel)	17.7%	22.4%	0%	0%
128 BitMAC Accelerators (4 per each pseudo-channel)	64.3%	89.7%	0%	0%

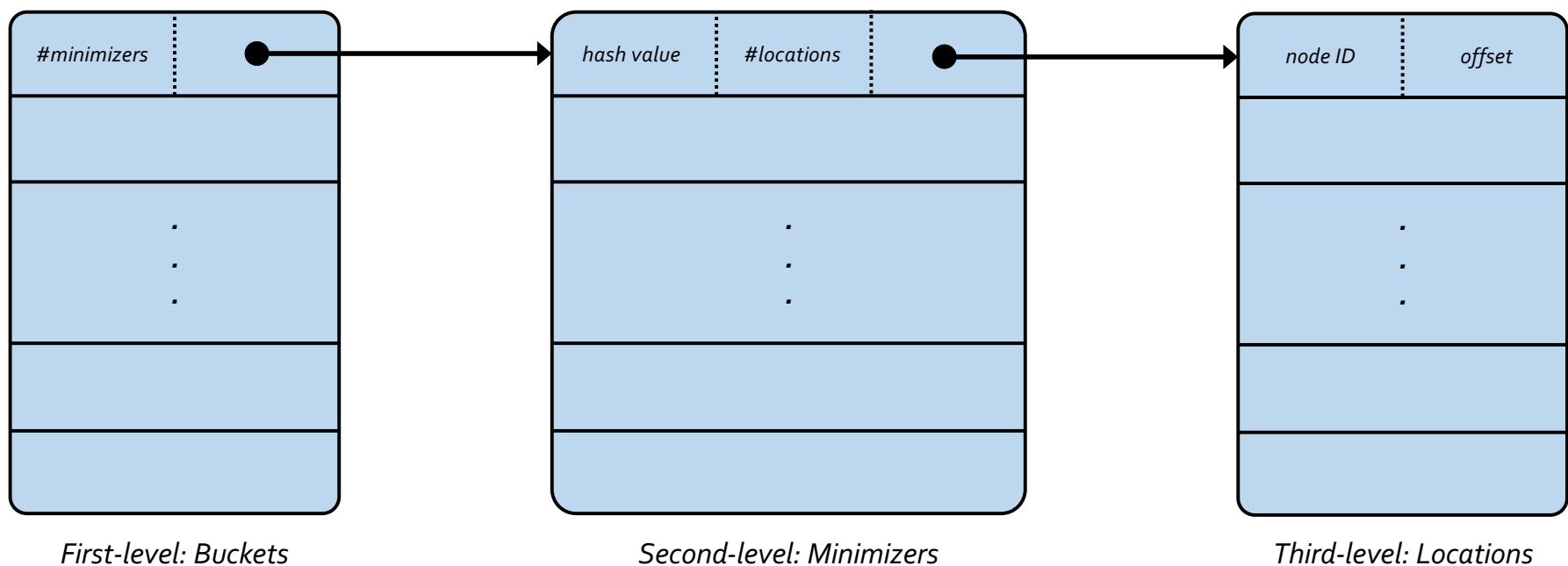
Backup Slides

(SeGram)

SeGram – Graph Structure



SeGraM – Index Structure



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

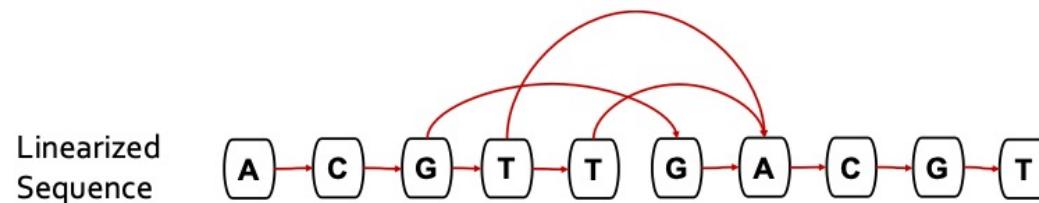
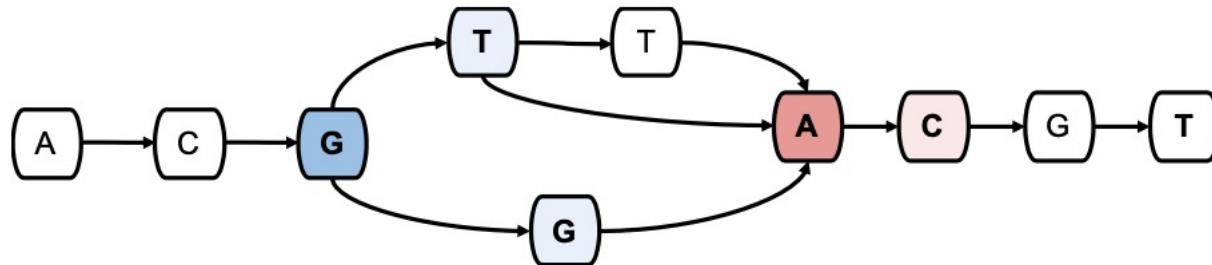
Algorithm 1 BitAlign Algorithm

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)
```

SeGraM – Hops



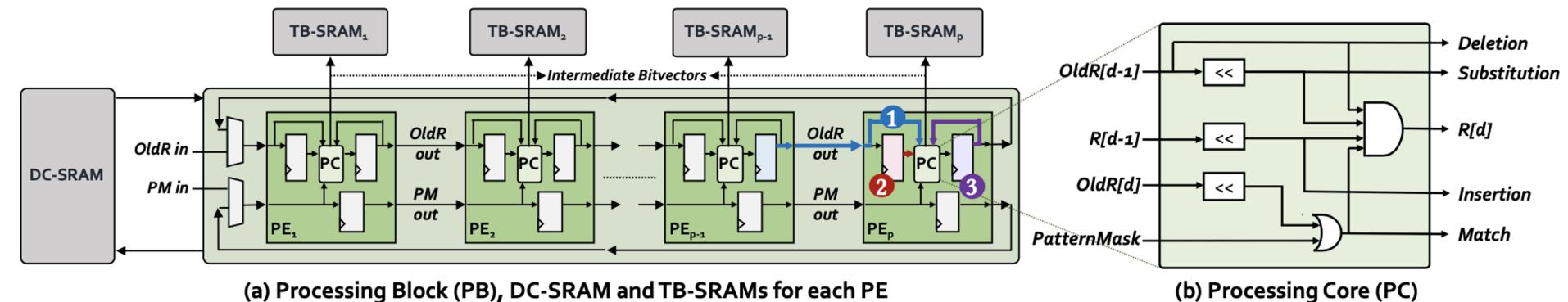
NodeID	1	2	3	4	5	6	7	8	9	10	
HopBits	0	0	0	0	0	0	0	0	0	0	1
	1	0	0	0	0	0	0	0	0	0	2
	0	1	0	0	0	0	0	0	0	0	3
	0	0	1	0	0	0	0	0	0	0	4
	0	0	0	1	0	0	0	0	0	0	5
	0	0	1	0	0	0	0	0	0	0	6
	0	0	0	1	1	1	0	0	0	0	7
	0	0	0	0	0	0	1	0	0	0	8
	0	0	0	0	0	0	0	1	0	0	9
	0	0	0	0	0	0	0	0	1	0	10

Recall: GenASM-DC's HW Design

Cycle#	Thread ₁ R _{0/4}	Thread ₂ R _{1/5}	Thread ₃ R _{2/6}	Thread ₄ R _{3/7}
#1	To-Ro	-	-	-
#2	T ₁ -Ro	To-R ₁	-	-
#3	T ₂ -Ro	T ₁ -R ₁	To-R ₂	-
#4	T ₃ -Ro	T ₂ -R ₁	T ₁ -R ₂	To-R ₃
#5	To-R ₄	T ₃ -R ₁	T ₂ -R ₂	T ₁ -R ₃
#6	T ₁ -R ₄	To-R ₅	T ₃ -R ₂	T ₂ -R ₃
#7	T ₂ -R ₄	T ₁ -R ₅	To-R ₆	T ₃ -R ₃
#8	T ₃ -R ₄	T ₂ -R ₅	T ₁ -R ₆	To-R ₇
#9	-	T ₃ -R ₅	T ₂ -R ₆	T ₁ -R ₇
#10	-	-	T ₃ -R ₆	T ₂ -R ₇
#11	-	-	-	T ₃ -R ₇

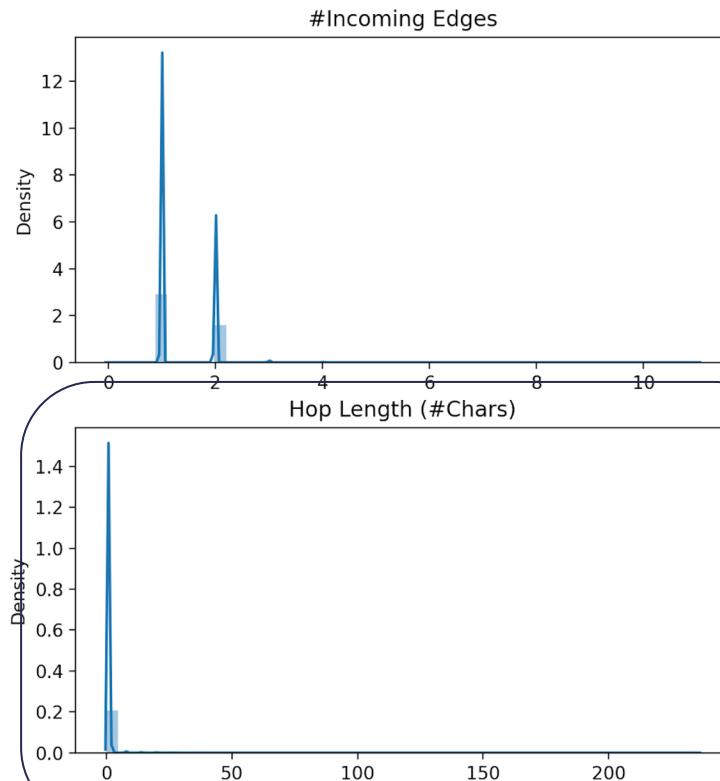
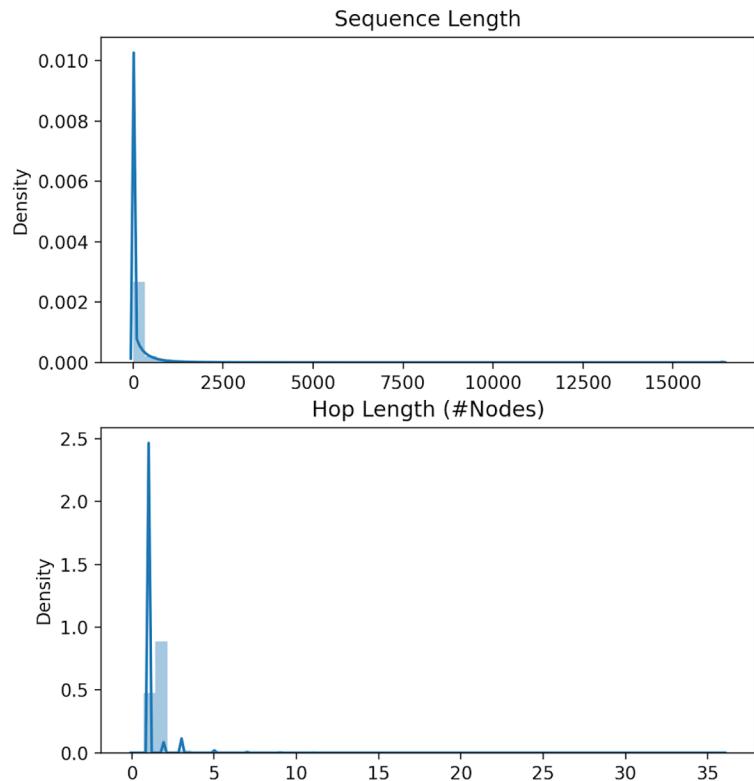
```

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



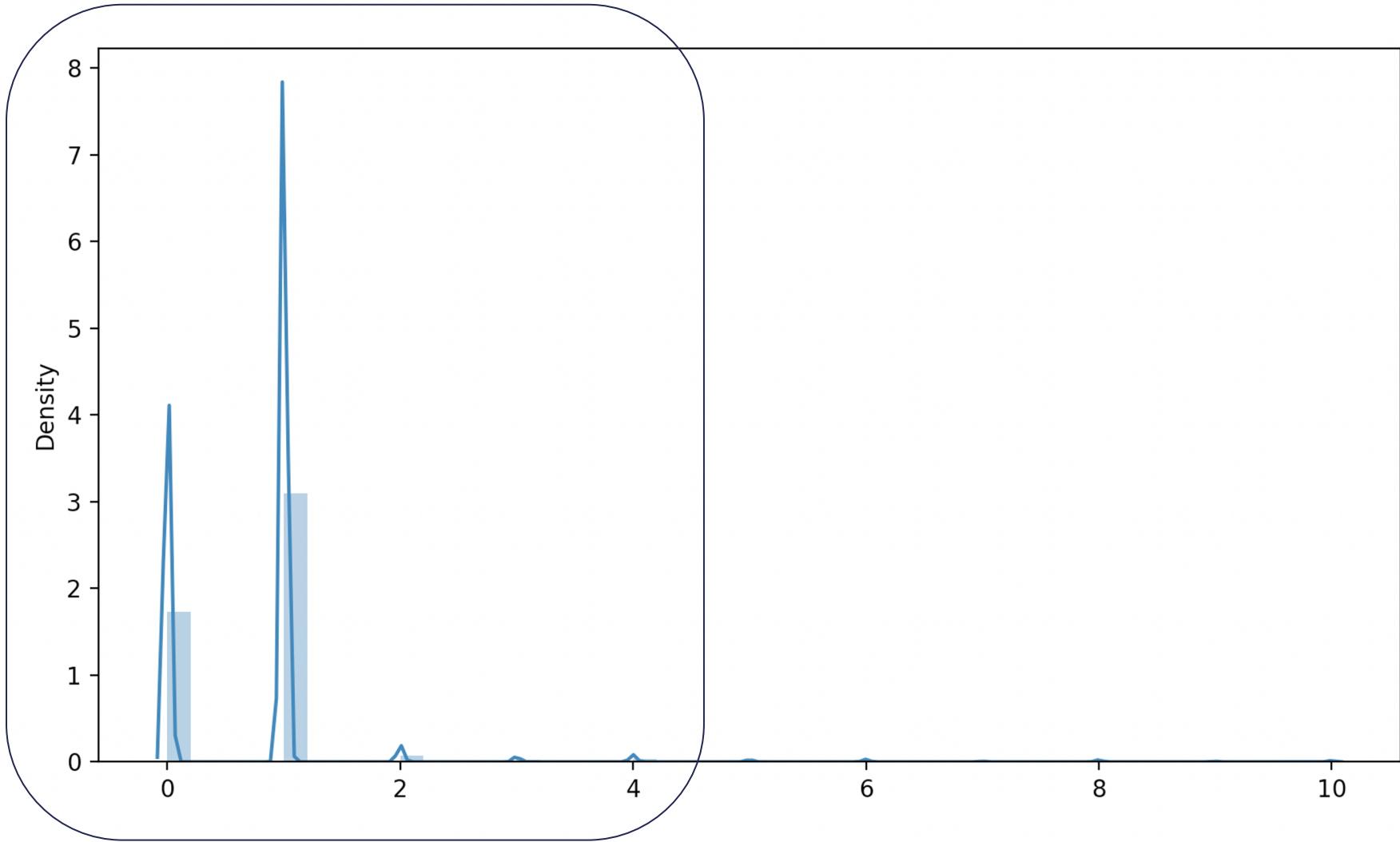
BitAlign – Hop Length Dist Plots

v3.3.2_m16384



Hop Length Dist Plots (cont'd.)

v3.3.2_m16384 - Filtered Hop Length (#Chars)



DP-based Graph Alignment

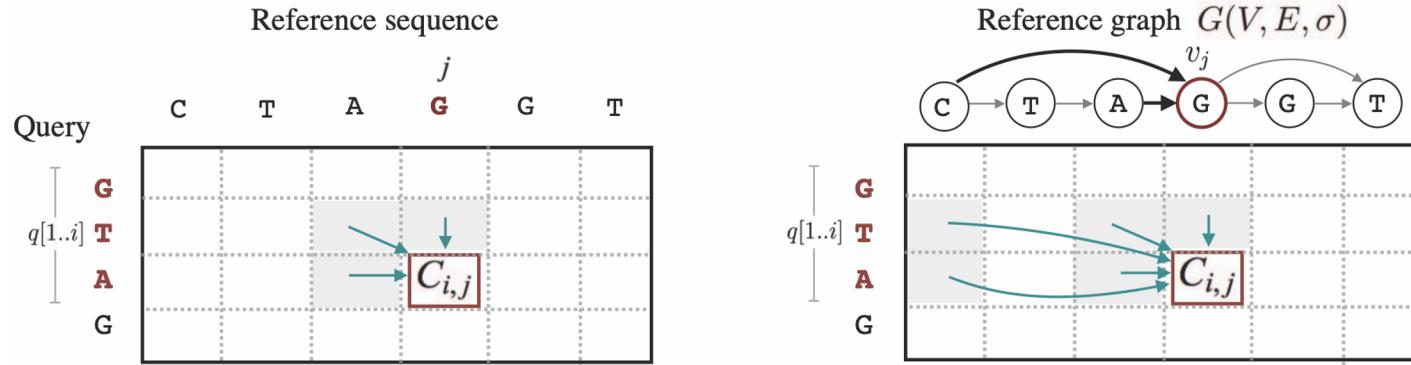


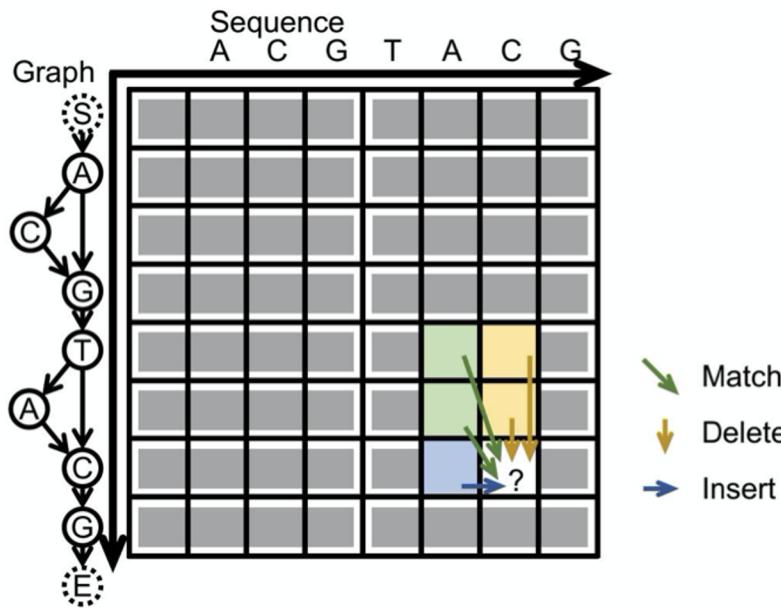
Fig. 2: Example to illustrate difference between Smith-Waterman sequence to sequence alignment and sequence to DAG alignment procedures.

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

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

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](#)

"A region of a yeast genome variation graph"
from [vg paper](#) [Garrison et al., Nature Biotechnology, 2018]

