

# P&S Accelerating Genomics

## Lecture 9: GRIM-Filter

Jeremie S. Kim

ETH Zurich

Fall 2022

15 December 2022

# ***GRIM-Filter:***

*Fast seed location filtering in DNA read mapping  
using processing-in-memory technologies*

**Jeremie S. Kim,**

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



**Carnegie Mellon**



TOBB  
UNIVERSITY OF  
ECONOMICS AND TECHNOLOGY

**SAFARI**

**ETH zürich**

# Executive Summary

- **Genome Read Mapping** is a very important problem and is the first step in genome analysis
- Read Mapping is an **approximate string matching** problem
  - Find the best fit of 100 character strings into a 3 billion character dictionary
  - **Alignment** is currently the best method for determining the similarity between two strings, but is **very expensive**
- We propose an algorithm called **GRIM-Filter**
  - Accelerates read mapping by reducing the number of required alignments
  - GRIM-Filter can be accelerated using **processing-in-memory**
    - Adds simple logic into **3D-Stacked memory**
    - Uses high internal memory bandwidth to perform parallel filtering
- GRIM-Filter with processing-in-memory delivers a **3.7x speedup**

# GRIM-Filter Outline

## 1. Motivation and Goal

## 2. Background Read Mappers

a. Hash Table Based

b. Hash Table Based with Filter

## 3. Our Proposal: GRIM-Filter

## 4. Mapping GRIM-Filter to 3D-Stacked Memory

## 5. Results

## 6. Conclusion

# Motivation and Goal

- **Sequencing**: determine the [A,C,G,T] series in DNA strand
- Today's machines sequence short strands (**reads**)
  - Reads are on the order of 100 – 20k base pairs (**bp**)
  - The human genome is approximately 3 billion bp
- Therefore genomes are cut into reads, which are sequenced independently, and then reconstructed
  - **Read mapping** is the first step in analyzing someone's genome to detect predispositions to diseases, personalize medicine, etc.
- **Goal**: We want to **accelerate** end-to-end performance of **read mapping**

# GRIM-Filter Outline

1. Motivation and Goal

**2. Background: Read Mappers**

a. Hash Table Based

b. Hash Table Based with Filter

3. Our Proposal: GRIM-Filter

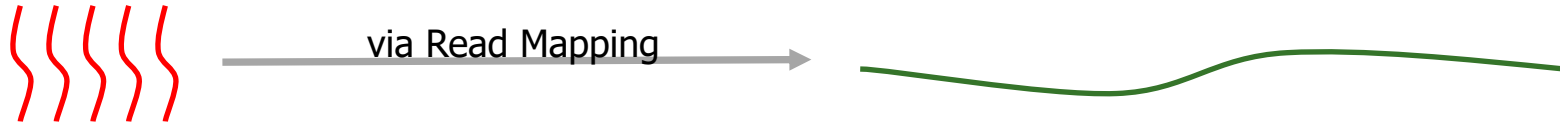
4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

6. Conclusion

# Background: Read Mappers

We now have **sequenced reads** and want a **full genome**



We map **reads** to a known **reference genome** (>99.9% similarity across humans) with some minor errors allowed

Because of high similarity, long sequences in **reads** perfectly match in the **reference genome**



We can use a hash table to help quickly map the **reads**!

# GRIM-Filter Outline

1. Motivation and Goal

**2. Background: Read Mappers**

**a. Hash Table Based**

b. Hash Table Based with Filter

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

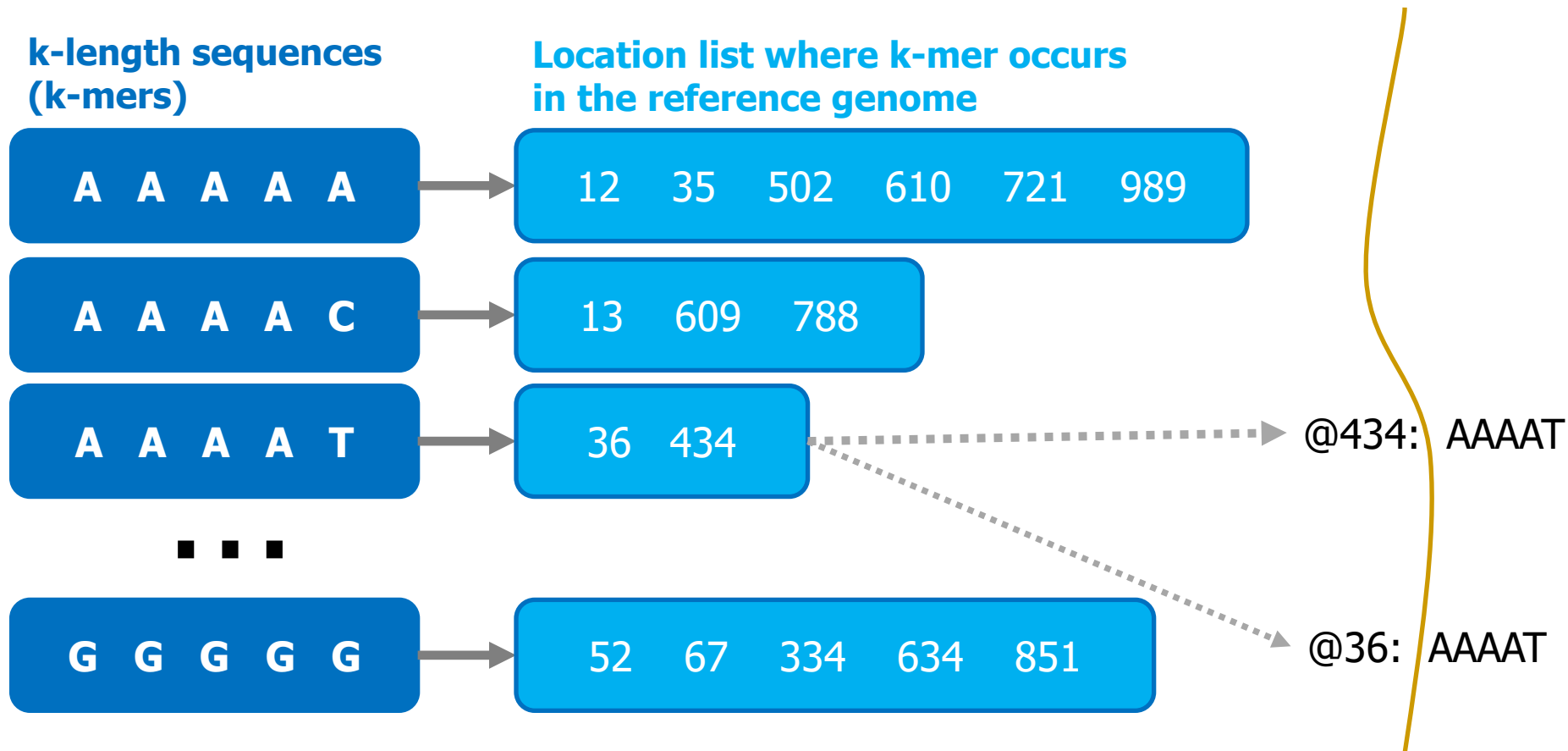
5. Results

6. Conclusion



# Generating Hash Tables

To map any reads, generate a **hash table** per **reference genome**.



**We can query the table with substrings from reads to quickly find a list of possible mapping locations**

# Hash Tables in Read Mapping

Read Sequence (100 bp)

**99.9%** of locations  
result in a **mismatch**

Hash Table

Reference Genome

**We want to filter these out  
so we do not waste time  
trying to align them**

# Location Filtering

- **Alignment** is **expensive** and requires the use of  $O(n^2)$  dynamic programming algorithm
  - We need to align millions to billions of reads

Our goal is to accelerate **read mapping** by improving the **filtering** step

- Both methods are used by mappers today, but **filtering** has replaced alignment as the bottleneck [Xin+, BMC Genomics 2013]

# GRIM-Filter Outline

1. Motivation and Goal

**2. Background: Read Mappers**

a. Hash Table Based

**b. Hash Table Based with Filter**

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

6. Conclusion

# Hash Tables in Read Mapping

Read Sequence (100 bp)



**Matching...**

**Mismatch.**

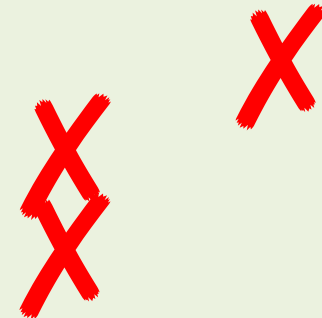
False  
Negative

Hash Table

37 140  
894 1203  
1564

Reference Genome

Filter



# GRIM-Filter Outline

1. Motivation and Goal

2. Background: Read Mappers

a. Hash Table Based

b. Hash Table Based with Filter

**3. Our Proposal: GRIM-Filter**

4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

6. Conclusion

# Our Proposal: GRIM-Filter

1. **Data Structures: Bins & Bitvectors**
2. Checking a Bin
3. Integrating GRIM-Filter into a Mapper

# GRIM-Filter: Bins

- We partition the genome into large sequences (**bins**).



- Represent each bin with a **bitvector** that holds the occurrence of all permutations of a small string (**token**) in the bin
- To account for matches that straddle bins, we employ overlapping bins
  - A read will now always completely fall within a single bin

## Bitvector

<b>AAAAA</b>	1	<b>AAAAA</b> exists in bin x
AAAAC	0	
AAAAT	1	
...	...	
CCCCC	1	
<b>CCCCT</b>	0	<b>CCCCT</b> doesn't exist in bin x
CCCCG	0	
...	...	
GGGGG	1	



# GRIM-Filter: Bitvectors

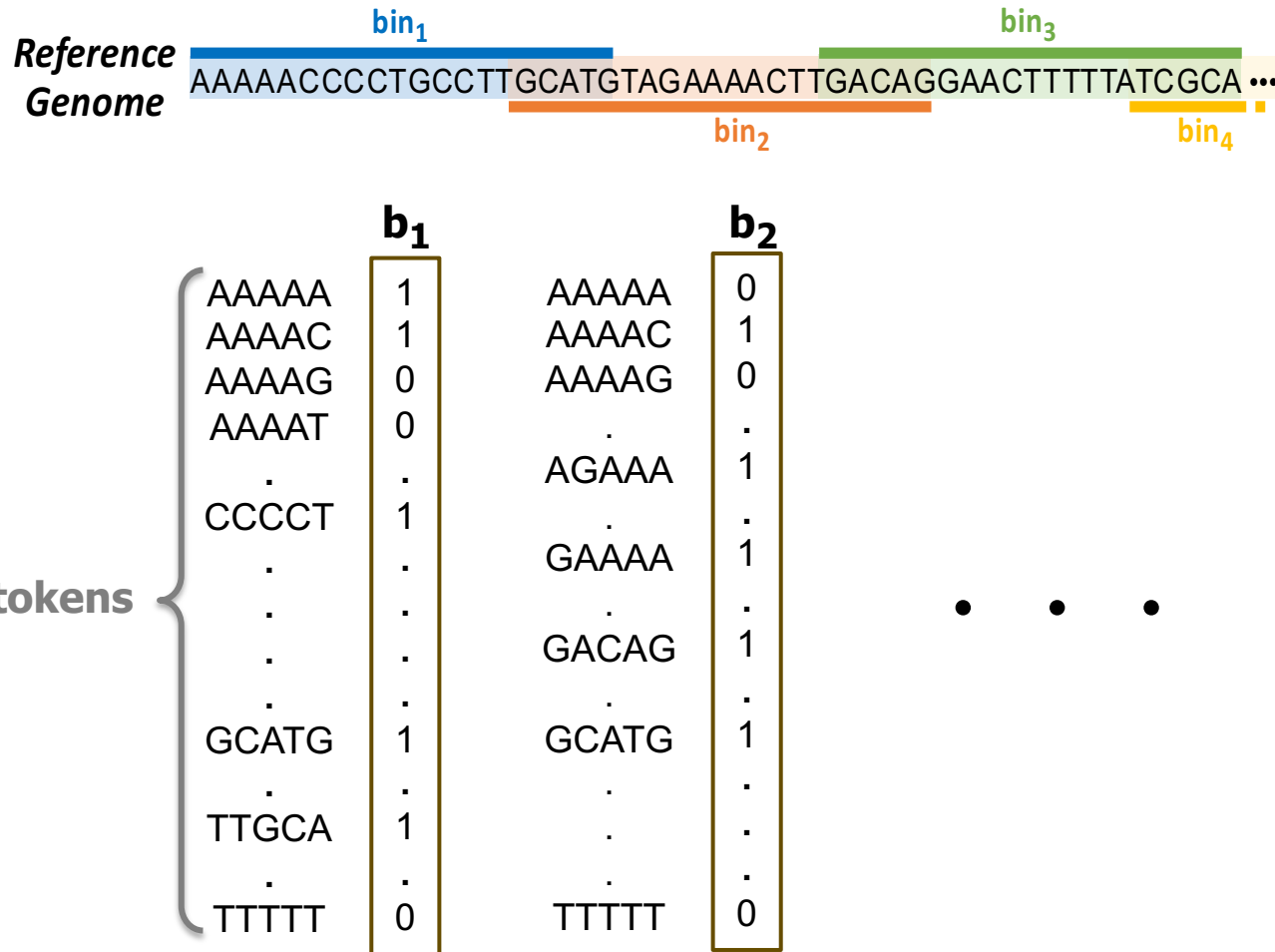


*Bin x*

**Bin x Bitvector**

AAAAA	<b>0</b>
...	...
CGTGA	<b>0</b>
...	...
TGAGT	<b>0</b>
...	...
GAGTC	<b>0</b>
...	...
GTGAG	<b>0</b>
...	...

# GRIM-Filter: Bitvectors



Storing all bitvectors requires  $4^n * t$  bits in memory, where  $t$  = number of bins.

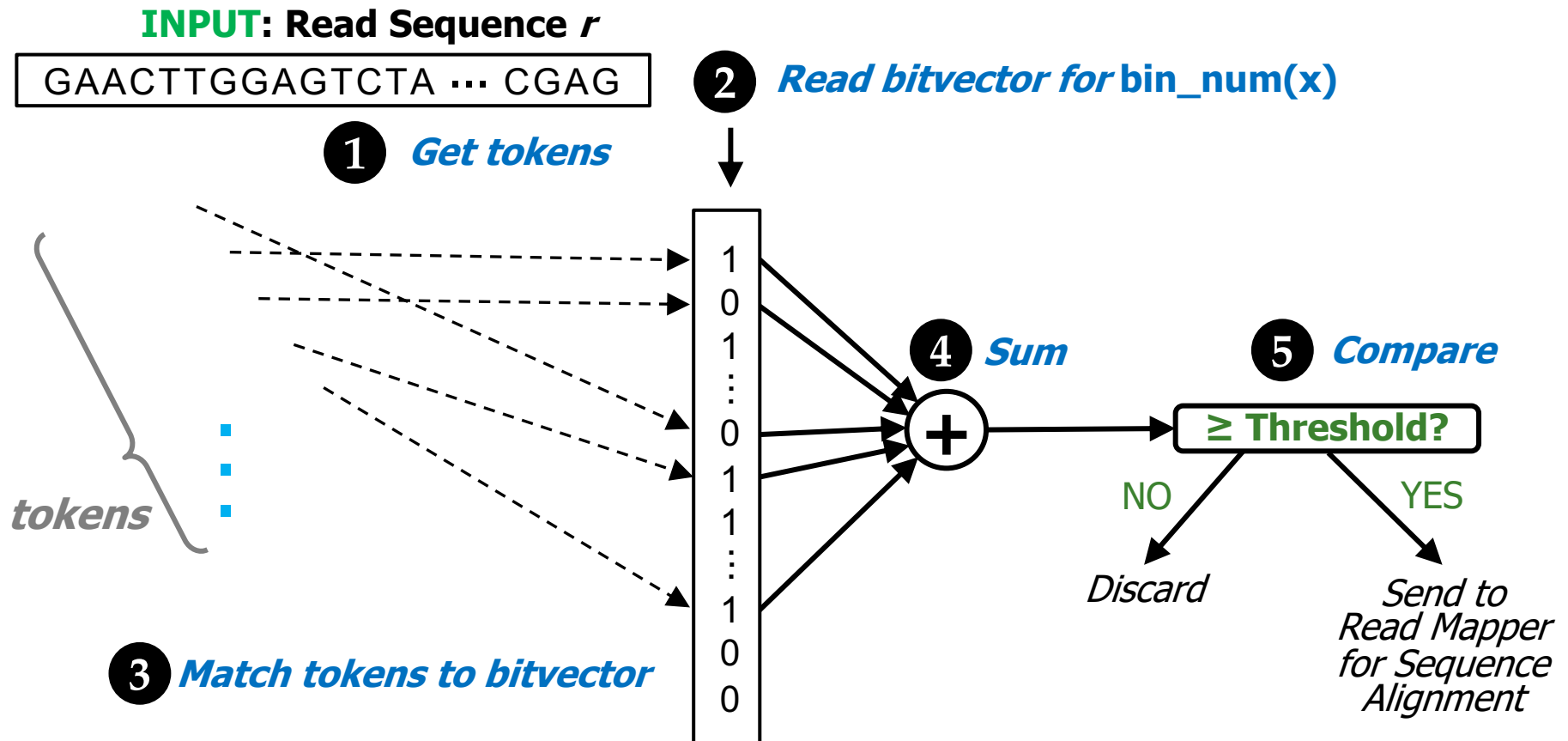
For **bin size**  $\sim 200$ , and **n** = 5, **memory footprint**  $\sim 3.8$  GB

# Our Proposal: GRIM-Filter

1. Data Structures: Bins & Bitvectors
2. **Checking a Bin**
3. Integrating GRIM-Filter into a Mapper

# GRIM-Filter: Checking a Bin

How GRIM-Filter determines whether to **discard** potential match locations in a given bin **prior** to alignment



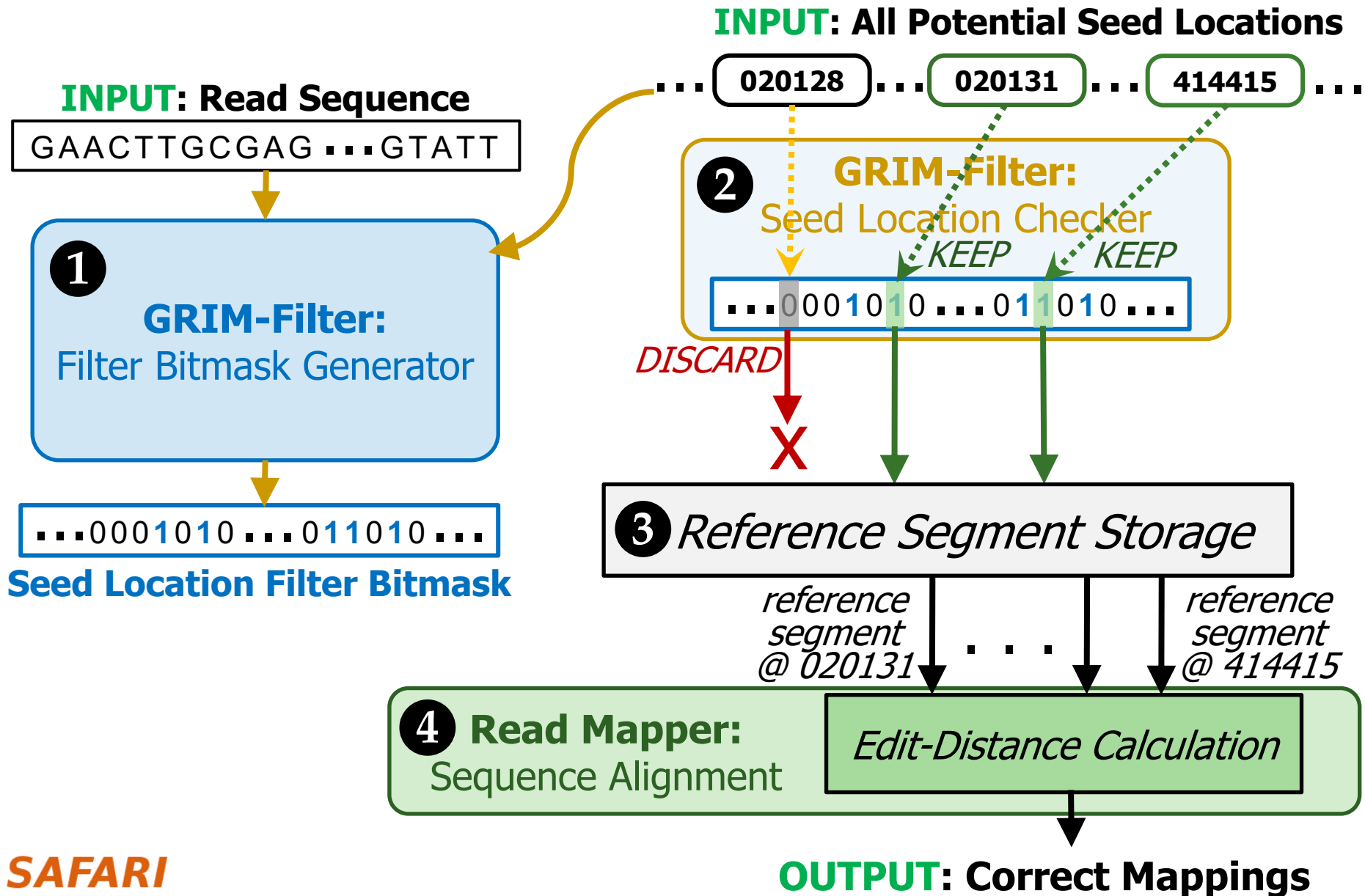
# Our Proposal: GRIM-Filter

1. Data Structures: Bins & Bitvectors
2. Checking a Bin
3. Integrating GRIM-Filter into a Mapper

# Our Proposal: GRIM-Filter

1. Data Structures: Bins & Bitvectors
2. Checking a Bin
3. **Integrating GRIM-Filter into a Mapper**

# Integrating GRIM-Filter into a Read Mapper



# GRIM-Filter Outline

1. Motivation and Goal

2. Background: Read Mappers

a. Hash Table Based

b. Hash Table Based with Filter

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

6. Conclusion



# Key Properties of GRIM-Filter

## 1. Simple Operations:

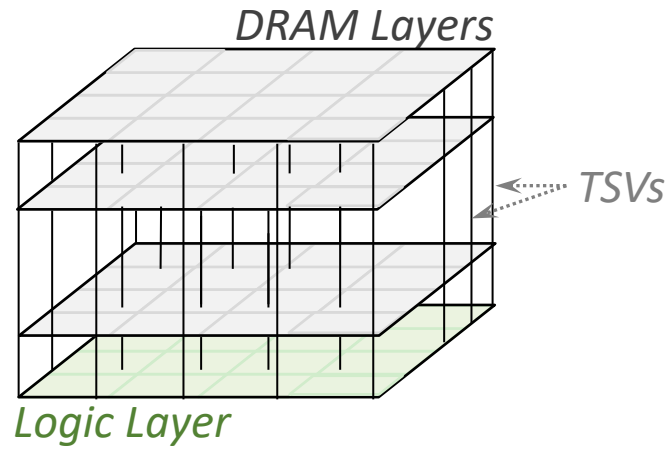
- To check a given bin, find the **sum** of all bits corresponding to each token in the read
- **Compare** against threshold to determine whether to align

## 2. Highly Parallel: Each bin is operated on independently and there are many many bins

## 3. Memory Bound: Given the frequent accesses to the large bitvectors, we find that GRIM-Filter is memory bound

**These properties together make GRIM-Filter a good algorithm to be run in 3D-Stacked DRAM**

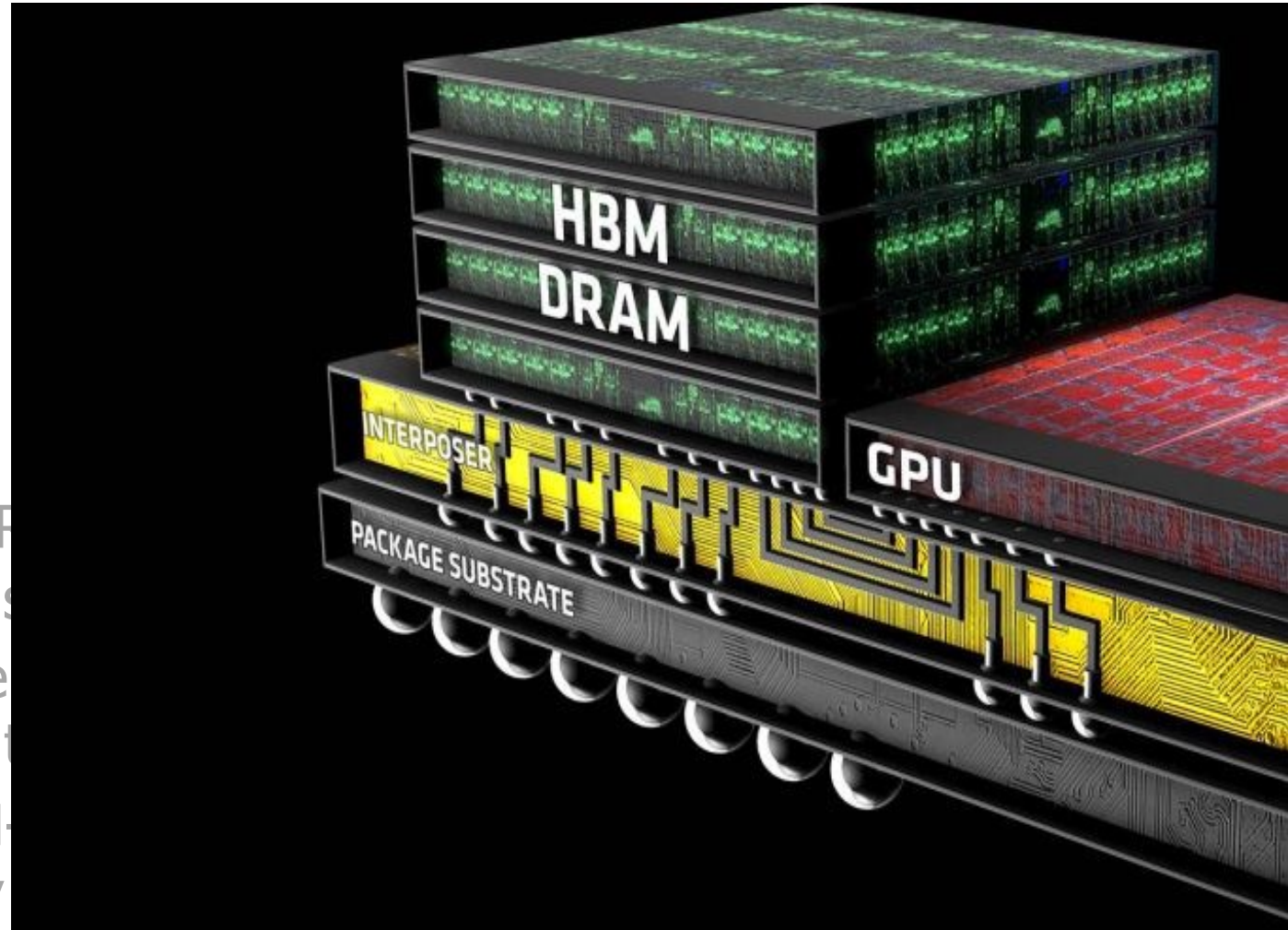
# 3D-Stacked Memory



- 3D-Stacked DRAM architecture has **extremely high bandwidth** as well as a stacked customizable logic layer
  - Logic Layer enables **Processing-in-Memory**, offloading computation to this layer and alleviating the memory bus
  - Embed GRIM-Filter operations into **DRAM logic layer** and appropriately distribute bitvectors throughout memory

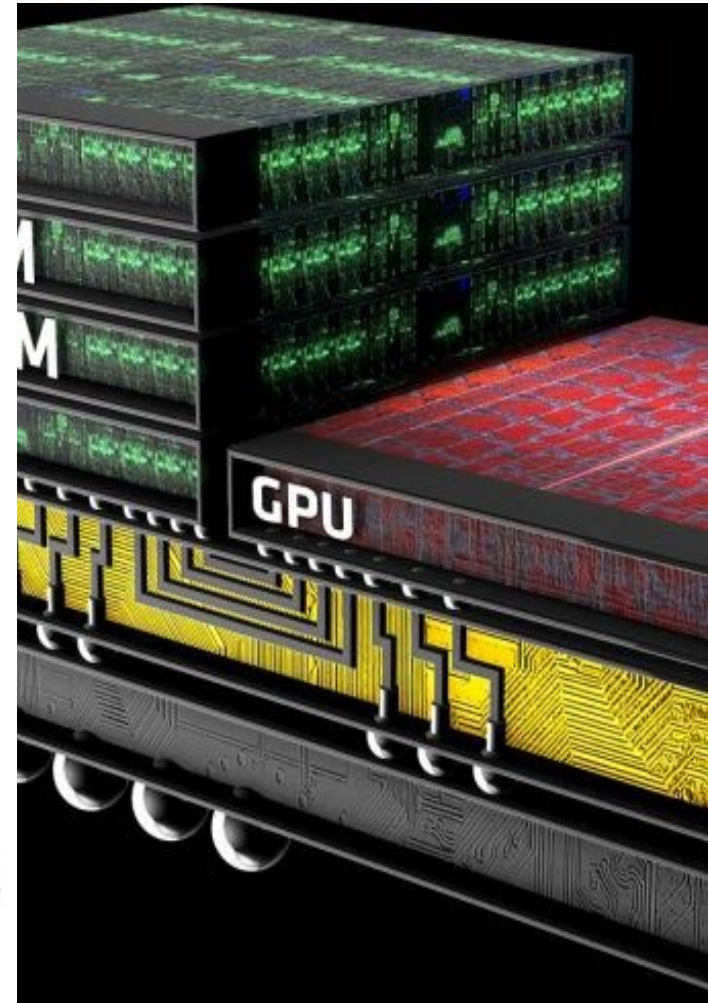
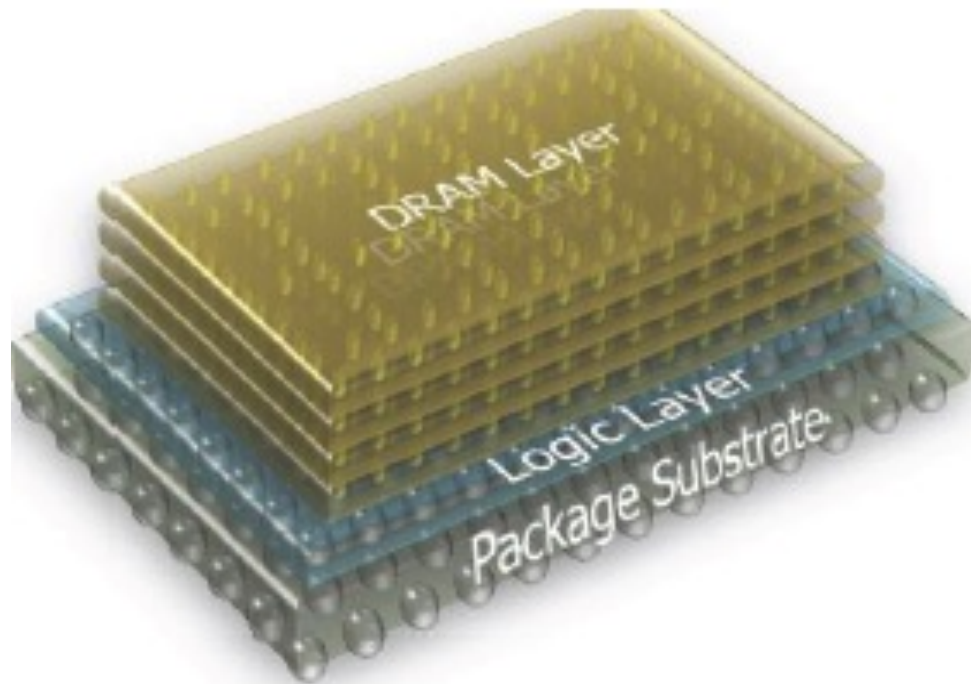
# 3D-Stacked Memory

- 3D-Stacked DRAM provides **bandwidth** as high as 100 GB/s
  - Logic Layer enables computation to be performed directly on the memory
  - Embed GRIMM architecture appropriately



# 3D-Stacked Memory

## Micron's HMC

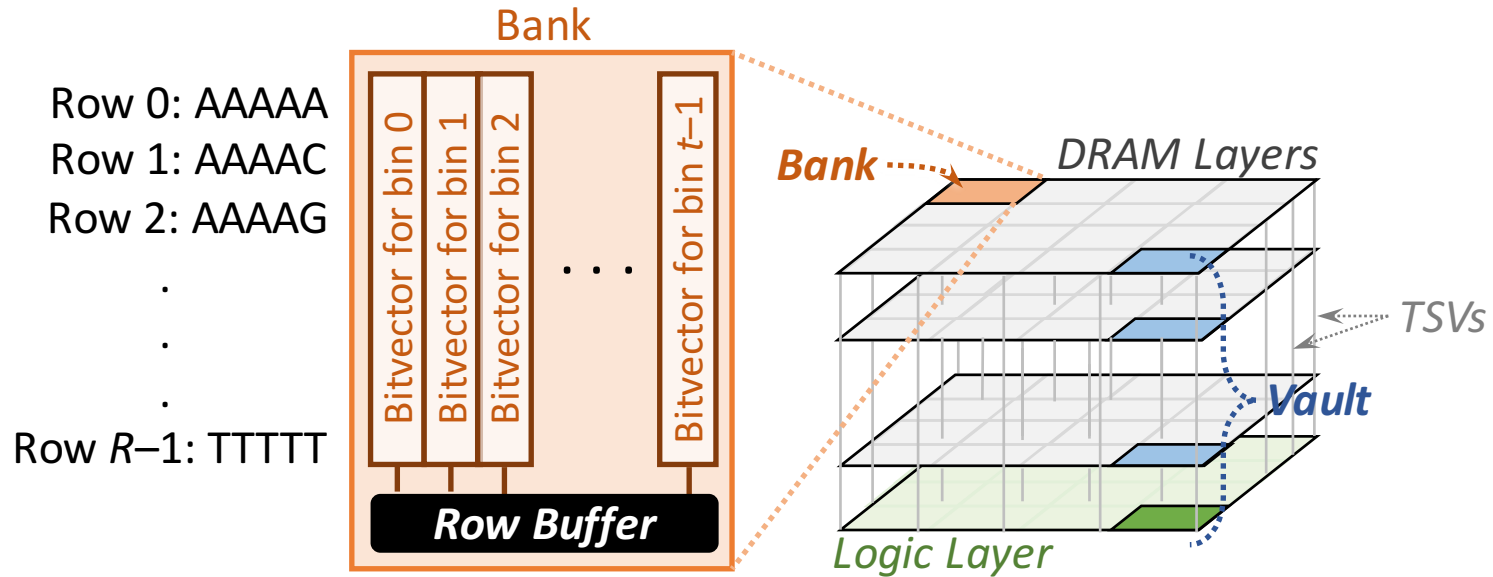


Micron has working demonstration components

[http://images.anandtech.com/doci/9266/HBMCar\\_678x452.jpg](http://images.anandtech.com/doci/9266/HBMCar_678x452.jpg)

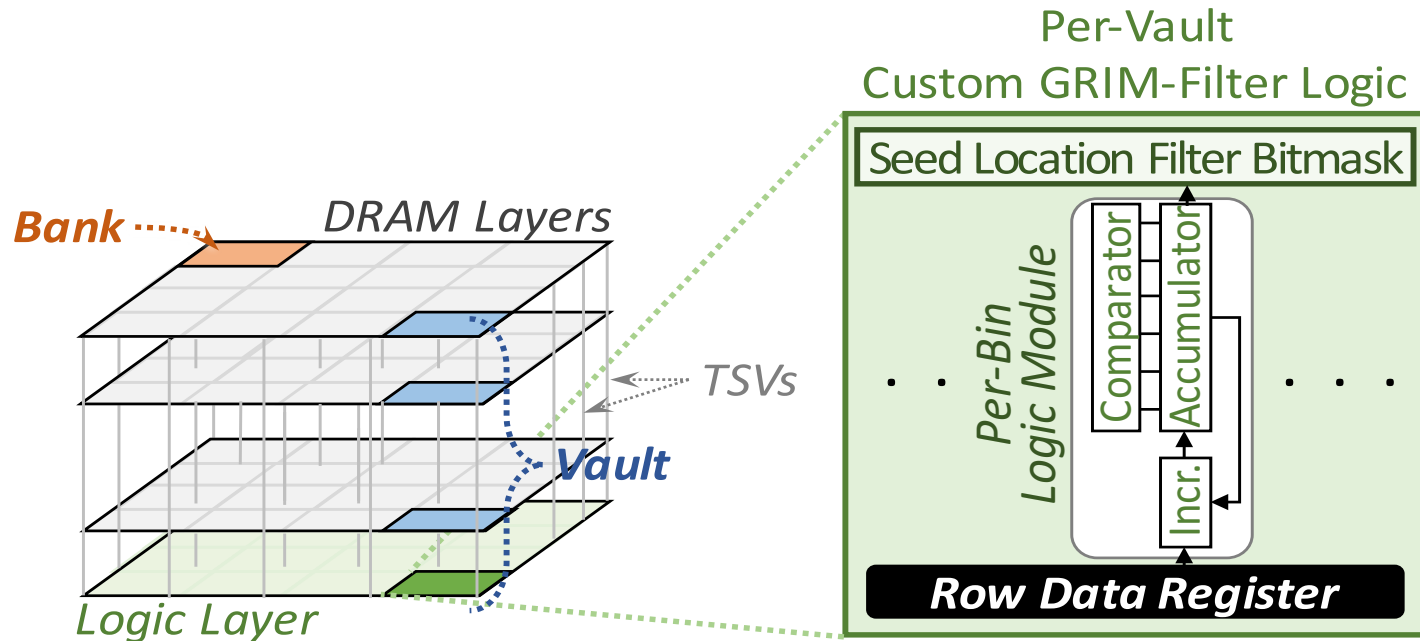
<http://i1-news.softpedia-static.com/images/news2/Micron-and-Samsung-Join-Force-to-Create-Next-Gen-Hybrid-Memory-2.png>

# GRIM-Filter in 3D-Stacked DRAM



- Each DRAM layer is organized as an array of **banks**
  - A **bank** is an array of cells with a row buffer to transfer data
- The layout of bitvectors in a bank enables filtering many bins in parallel

# GRIM-Filter in 3D-Stacked DRAM



- Customized logic for accumulation and comparison per genome segment
  - Low area overhead, simple implementation
  - For HBM2, we use 4096 incrementer LUTs, 7-bit counters, and comparators in logic layer

**Details are in the paper**

# GRIM-Filter Outline

1. Motivation and Goal

2. Background: Read Mappers

a. Hash Table Based

b. Hash Table Based with Filter

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

**5. Results**

6. Conclusion

# Methodology

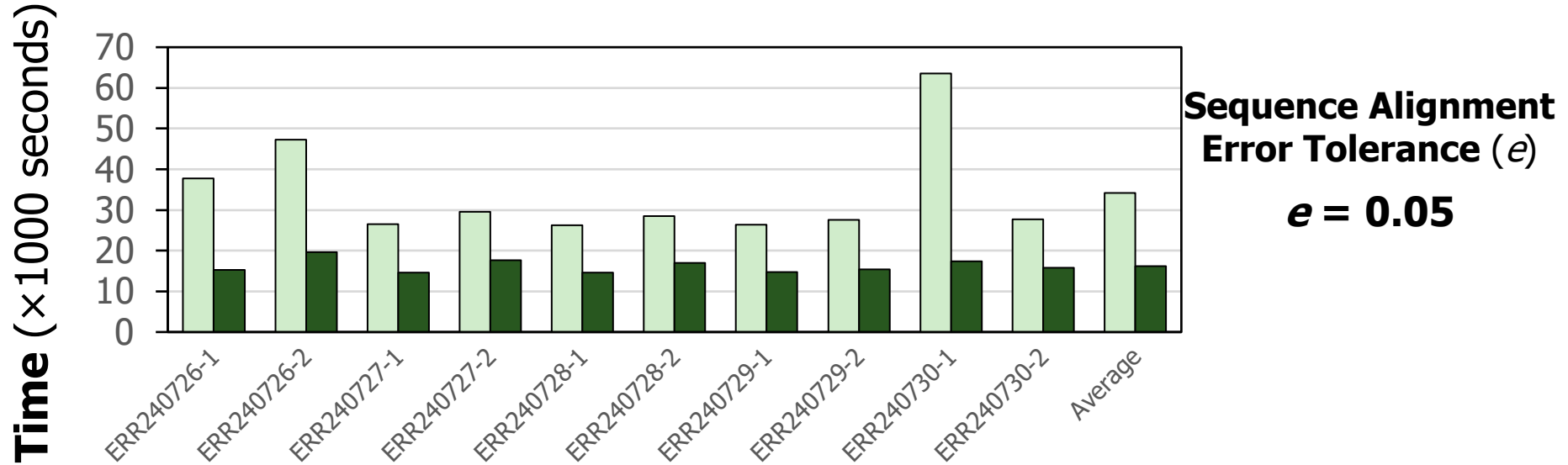
- Performance simulated using an in-house 3D-Stacked DRAM simulator
- Evaluate 10 real read data sets (From the 1000 Genomes Project)
  - Each data set consists of 4 million reads of length 100
- Evaluate two key metrics
  - Performance
  - False negative rate
    - The fraction of locations that pass the filter but result in a mismatch
- Compare against a state-of-the-art filter, FastHASH [Xin+, BMC Genomics 2013] when using mrFAST, but **GRIM-Filter can be used with ANY read mapper**



# GRIM-Filter Performance

Benchmarks and their Execution Times

FastHASH filter GRIM-Filter



**1.8x-3.7x performance benefit across real data sets**

**2.1x average performance benefit**

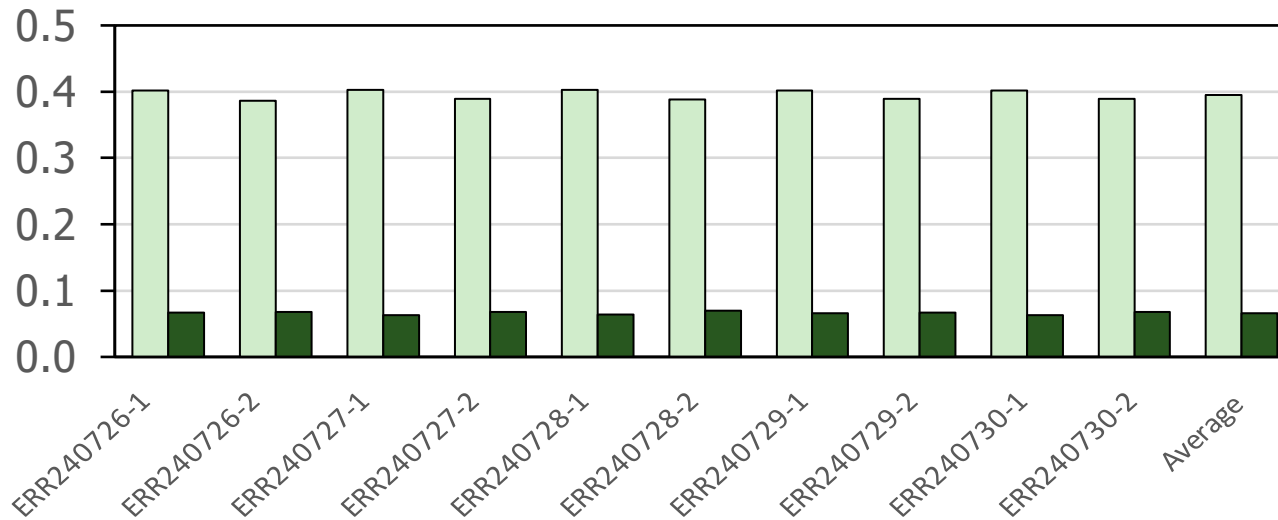
**GRIM-Filter gets performance due to its hardware-software co-design**

# GRIM-Filter False Negative Rate

Benchmarks and their False Negative Rates

FastHASH filter GRIM-Filter

False Negative Rate



Sequence Alignment  
Error Tolerance ( $e$ )

$e = 0.05$

**5.6x-6.4x False Negative reduction across real data sets**

**6.0x average reduction in False Negative Rate**

**GRIM-Filter utilizes more information available in the read to filter**

# Other Results in the Paper

- Sensitivity of execution time and false negative rates to error tolerance of string matching
- Read mapper execution time breakdown
- Sensitivity studies on the filter
  - Token Size
  - Bin Size
  - Error Tolerance

# GRIM-Filter Outline

1. Motivation and Goal

2. Background: Read Mappers

a. Hash Table Based

b. Hash Table Based with Filter

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

**6. Conclusion**

# Conclusion

We propose an **in-memory filtering algorithm** to **accelerate end-to-end read mapping** by reducing the number of required alignments

## Key ideas:

- Introduce a **new representation** of coarse-grained segments of the reference genome
- Use **massively-parallel in-memory operations** to identify read presence within each coarse-grained segment

## Key contributions and results:

- Customized filtering algorithm for 3D-Stacked DRAM
- Compared to the previous best filter
  - We observed **1.8x-3.7x read mapping speedup**
  - We observed **5.6x-6.4x fewer false negatives**

**GRIM-Filter is a universal filter** that can be applied to any read mapper

# ***GRIM-Filter:***

*Fast seed location filtering in DNA read mapping  
using processing-in-memory technologies*

**Jeremie S. Kim,**

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



**Carnegie Mellon**



TOBB  
UNIVERSITY OF  
ECONOMICS AND TECHNOLOGY

**SAFARI**

**ETH zürich**

# P&S Accelerating Genomics

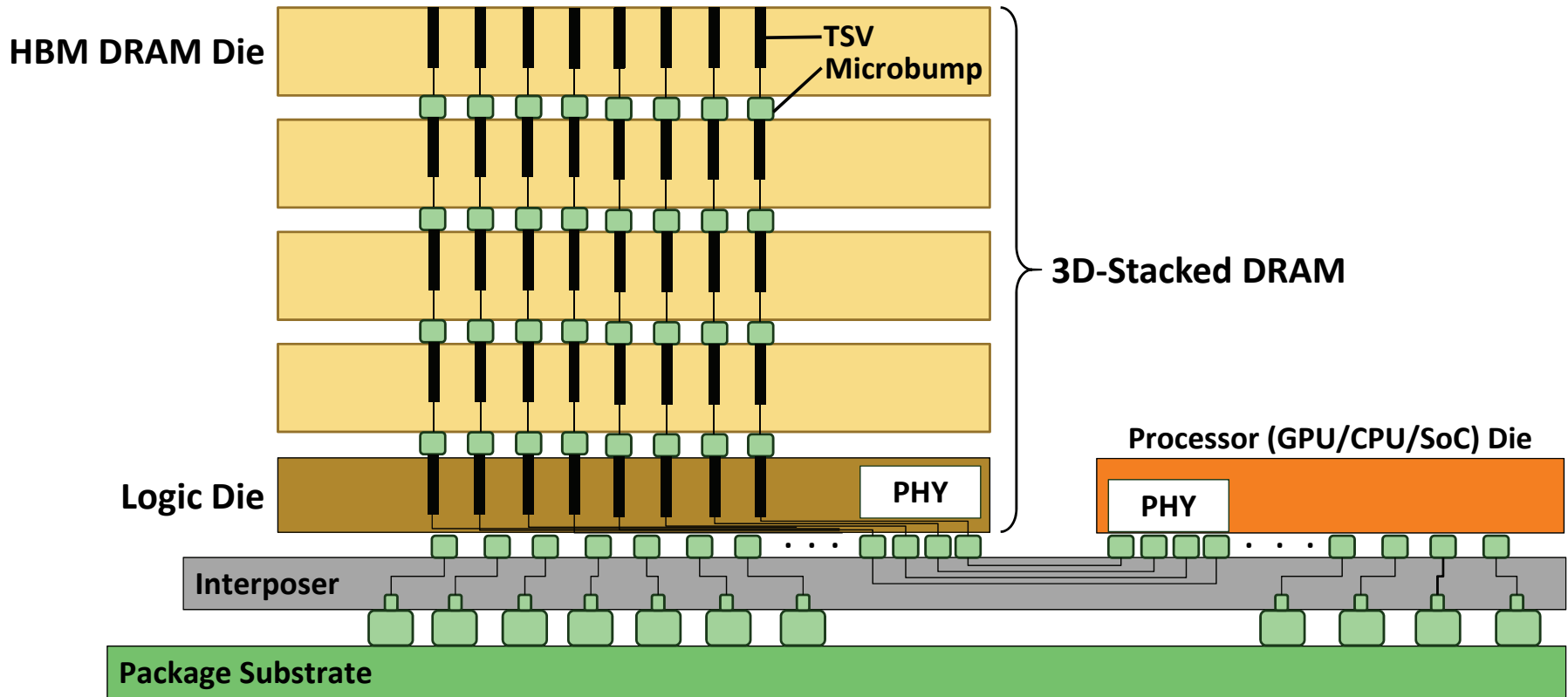
## Lecture 9: GRIM-Filter

Jeremie S. Kim

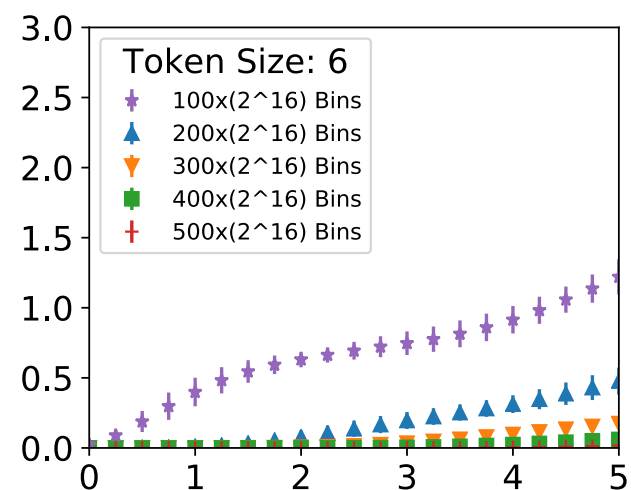
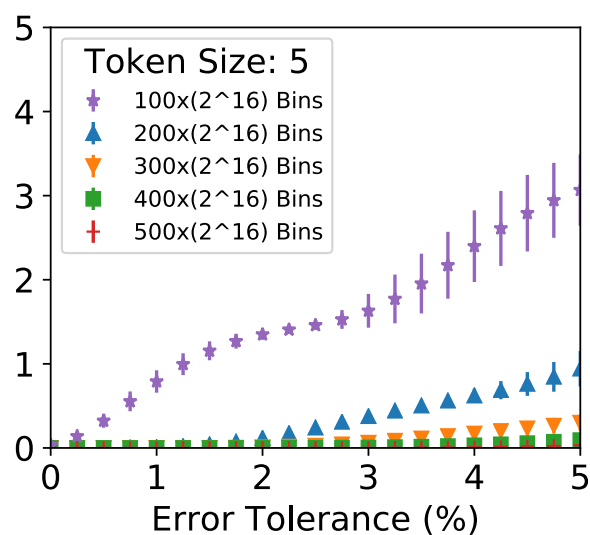
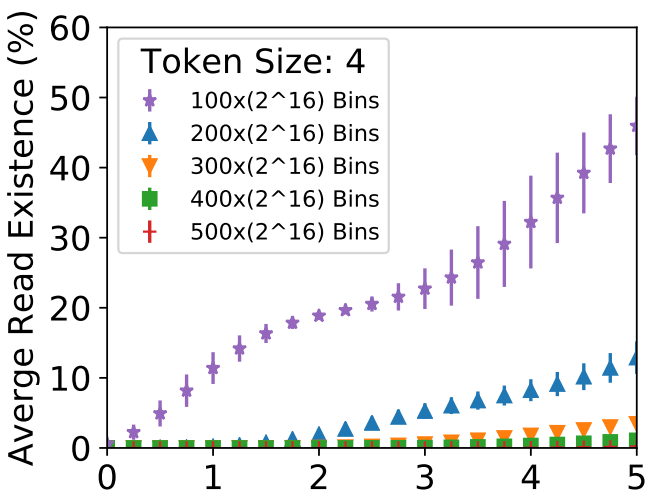
ETH Zurich

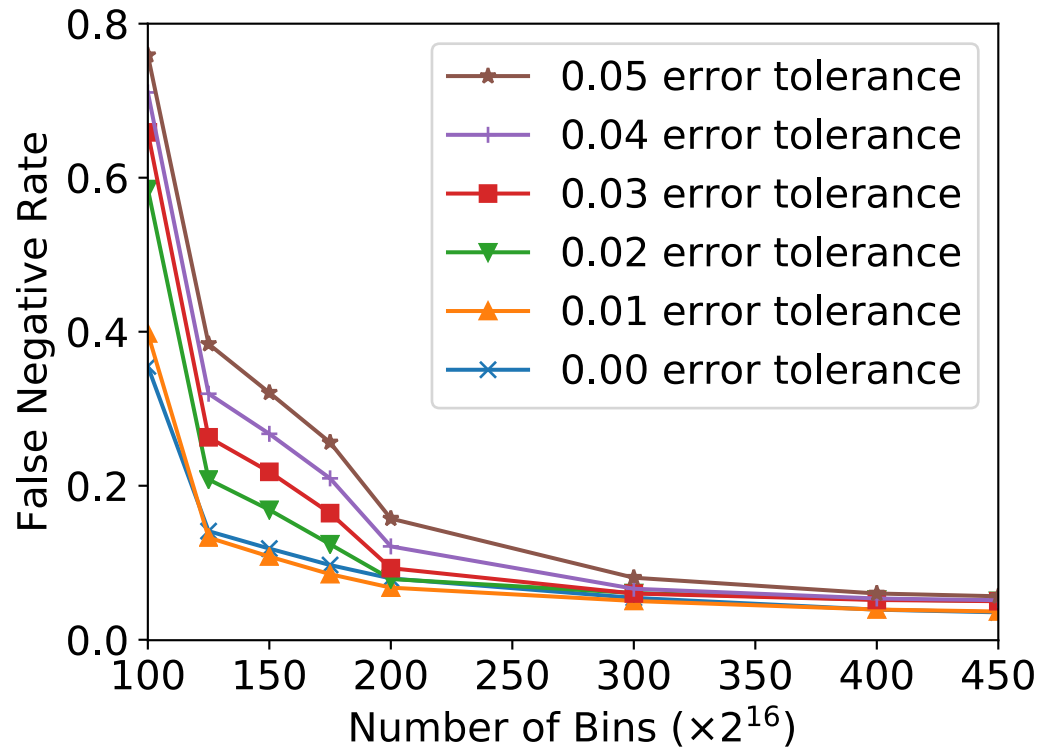
Fall 2022

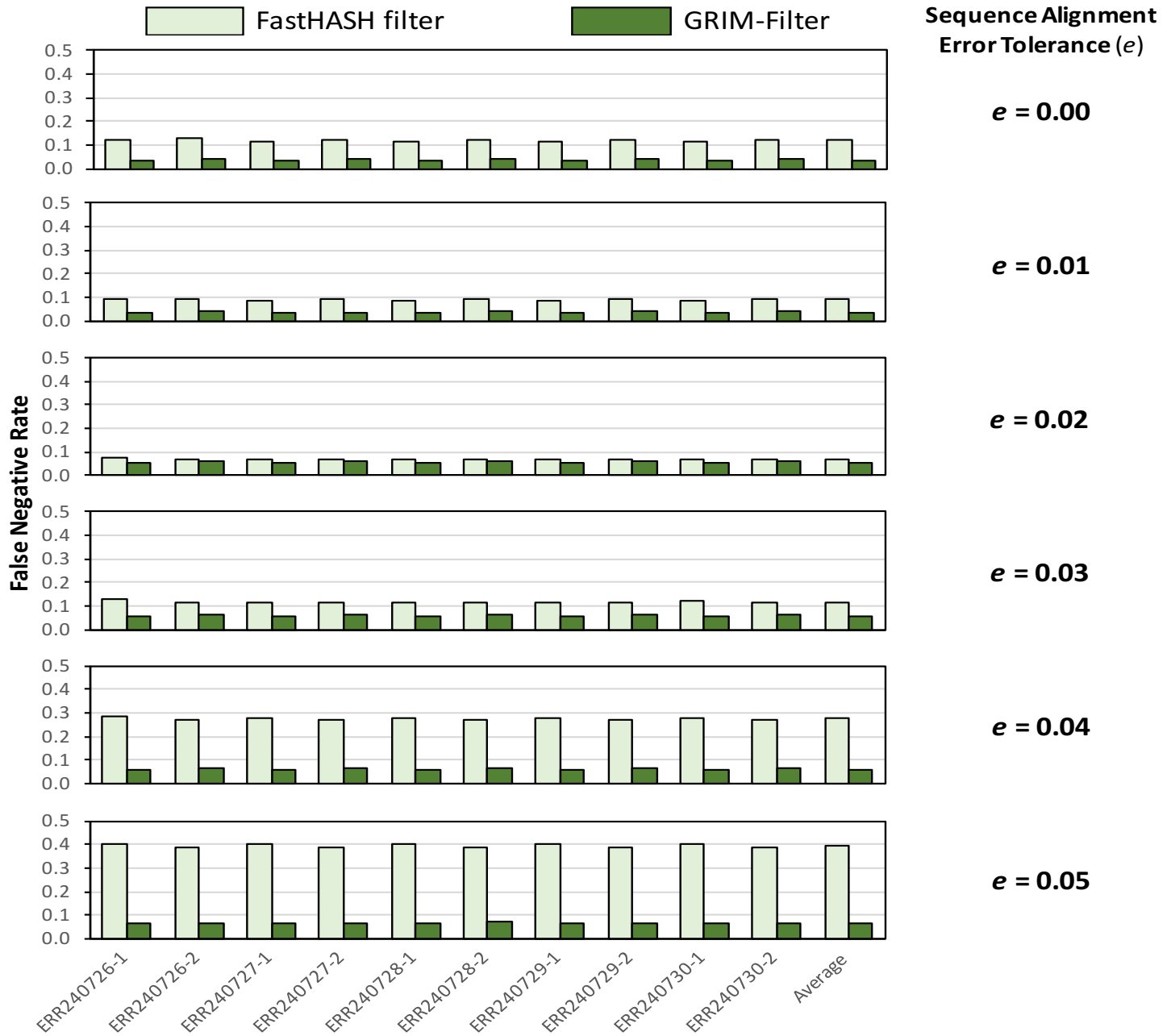
15 December 2022

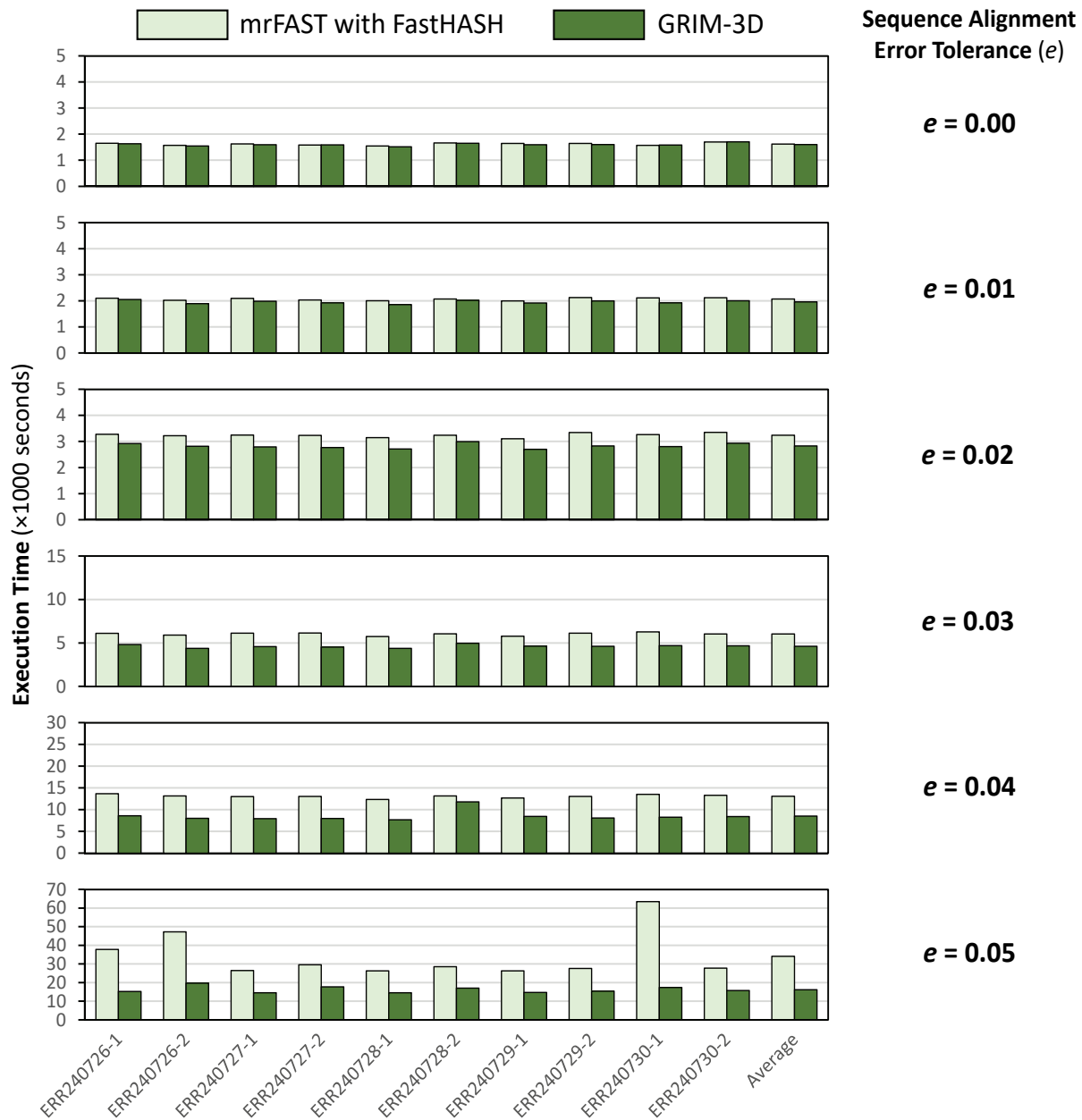












# GRIM-Filter: Error Tolerance

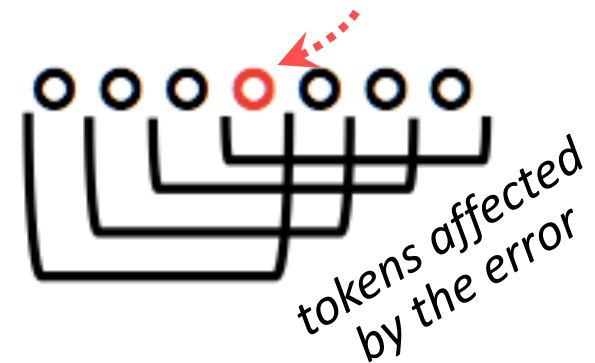
*total number of tokens in a read*

$$\text{Threshold} = \text{read\_length} - (n-1) - n \times \lceil \text{read\_length} \times e \rceil$$

*maximum number of tokens that could contain errors*

*number of errors allowed per read*

*single substitution error*



*one substitution error affects four tokens when  $n = 4$*

**GRIM-Filter can support different error tolerances by simply changing the threshold value**

**More details in the paper**