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

Published in 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

Presented by Alexander Frey

General Outline

- Summary of the paper
- Strength
- Weaknesses
- Discussion

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

GRIM-Filter Outline

Motivation and Goal

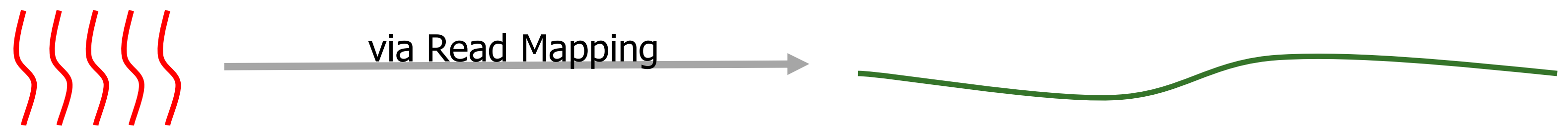
- **Sequencing**: determine the [A,C,G,T] series in DNA strand
- Today’s machines sequence short strands (**reads**)
  - Reads are on the order of 100 –2M base pairs (**bp**)
  - The human genome is approximately 3 billion bp
- Therefore genomes are cut into reads, which are sequenced independently, 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

SAFARI

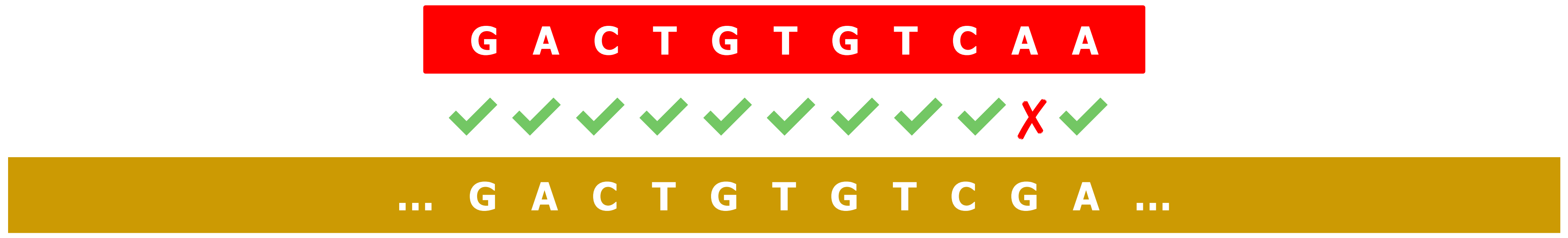
Background: Read Mappers

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**



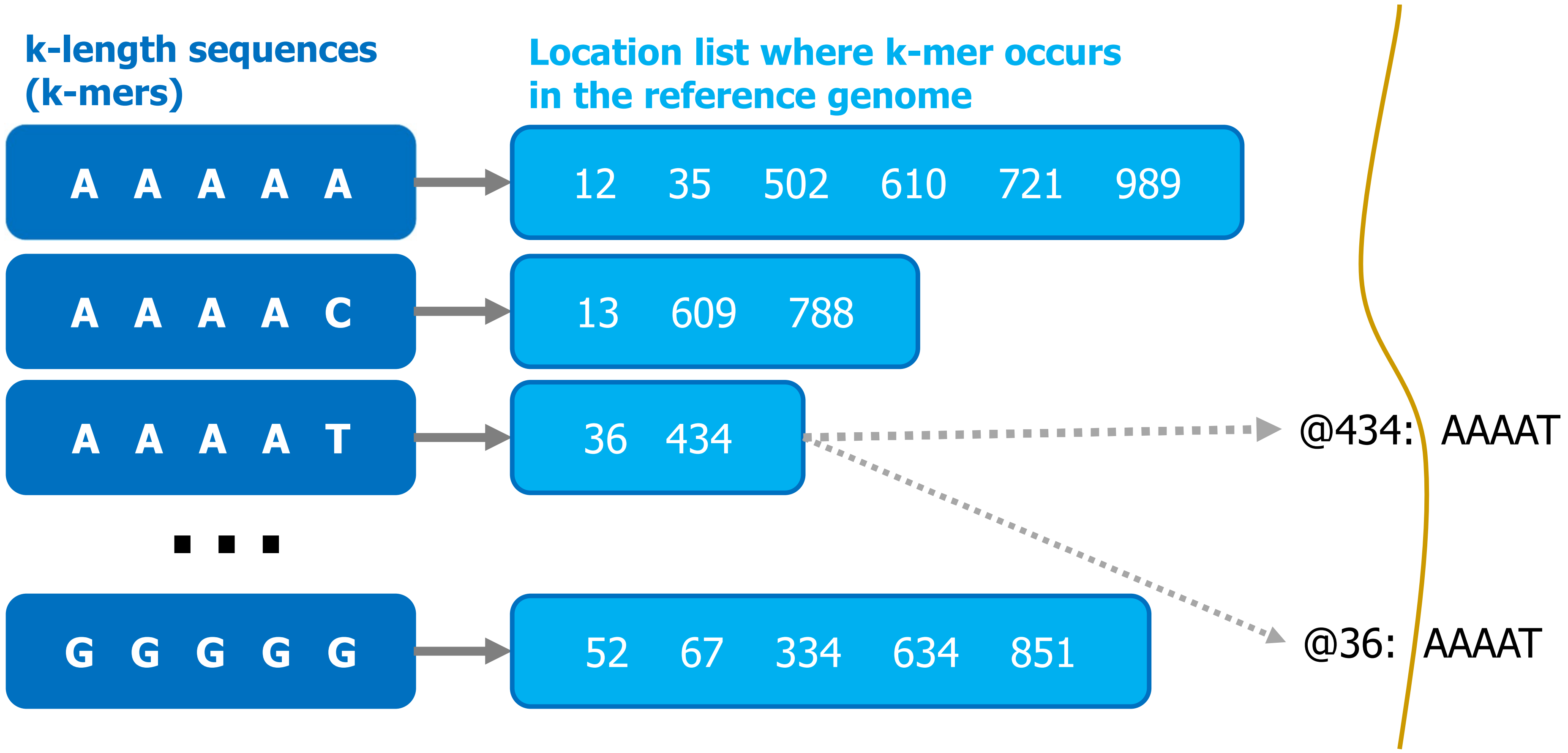
We can use a hash table to help quickly map the reads!

SAFARI

GRIM-Filter Outline

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

SAFARI

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

- M t f

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

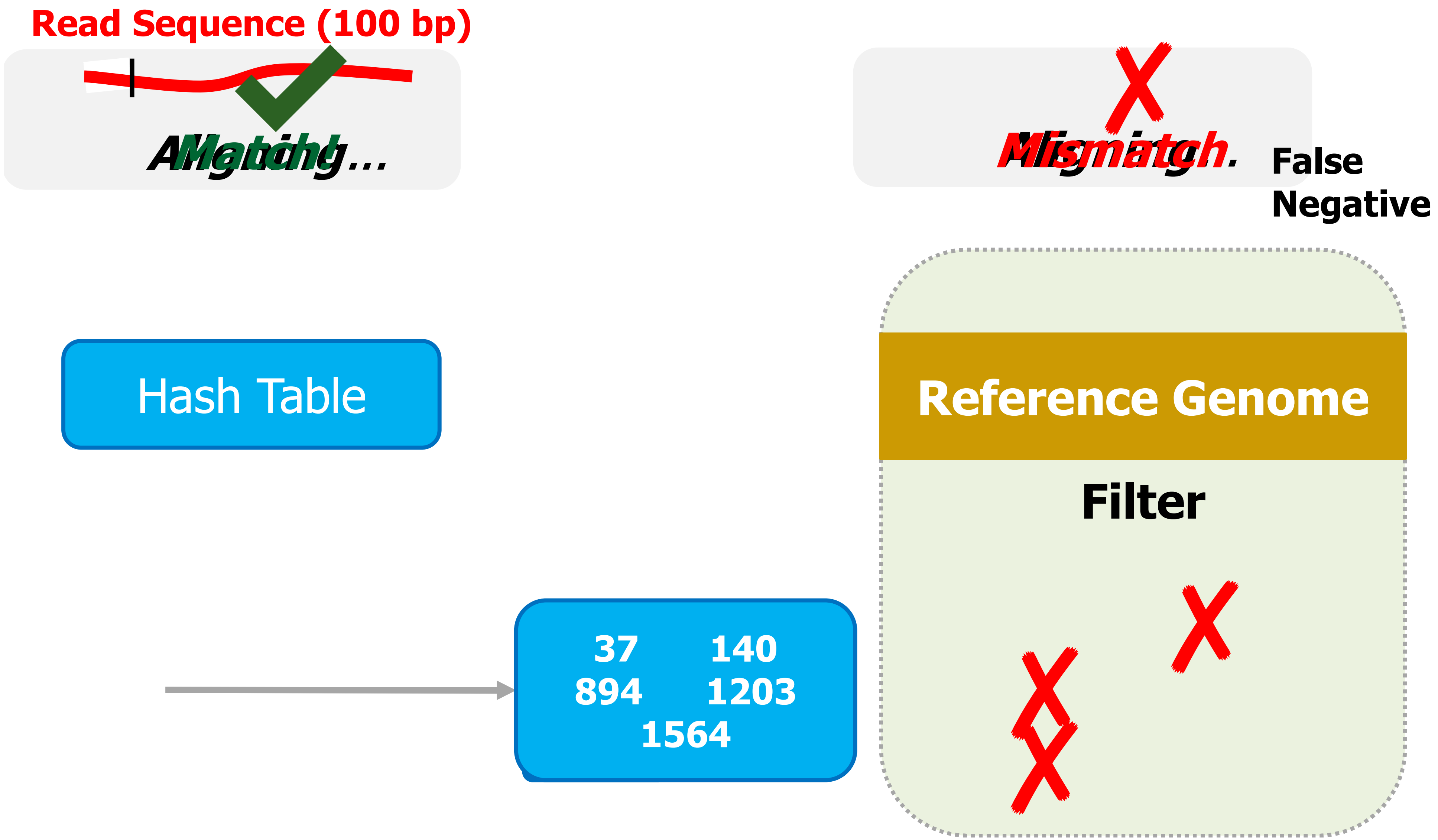
out mismatches quickly

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

SAFARI

GRIM-Filter Outline

Hash Tables in Read Mapping



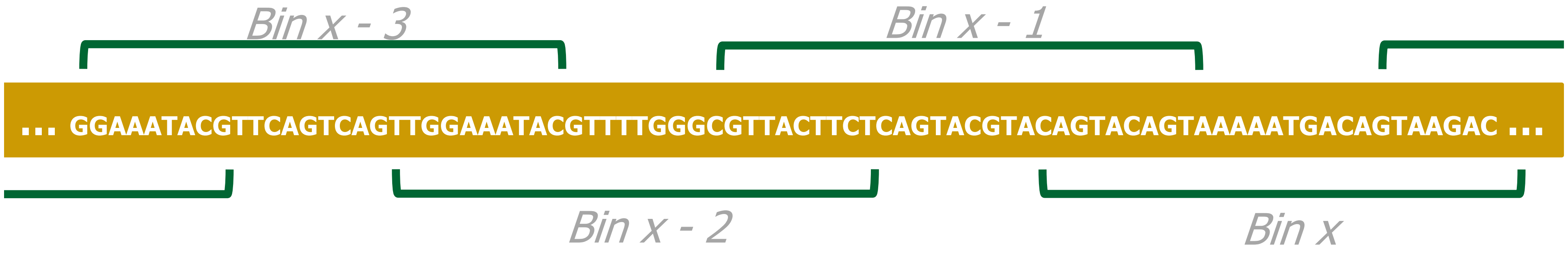
GRIM-Filter Outline

Our Proposal: GRIM-Filter

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

GRIM-Filter: Bins

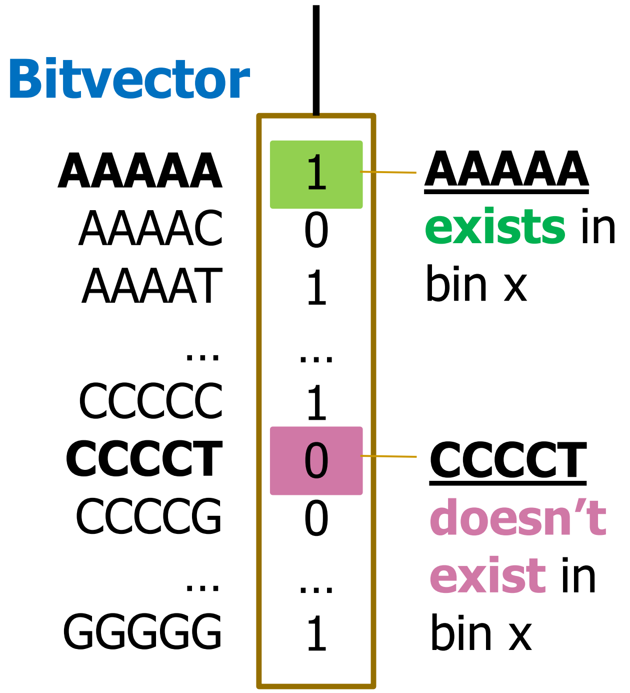
- We partition the genome into large sequences (**bins**).



Bin x - 3
Bin x - 1
... GGAAATACGTTCAGTCAGTTGGAAATACGTTTTGGGCGTTACTTCTCAGTACGTACAGTACAGTAAAAATGACAGTAAGAC ...
Bin x - 2
Bin x

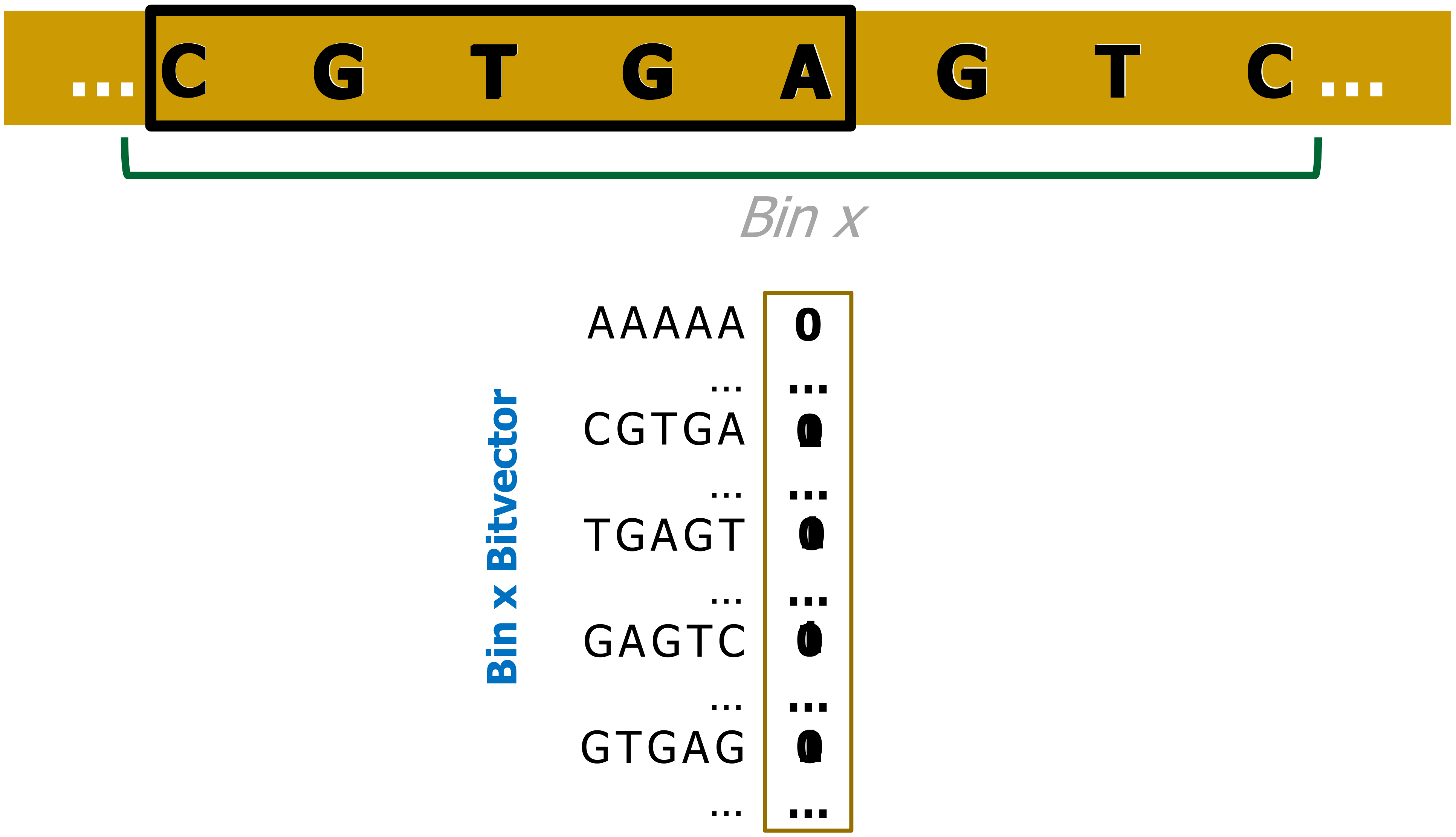
- Represent each bin with a **bitvector** that holds the occurrence of all permutations of a small string (**token**) in the bin

- To account for matches that straddle bins, we employ overlapping bins
  - A read will now always completely fall within a single bin



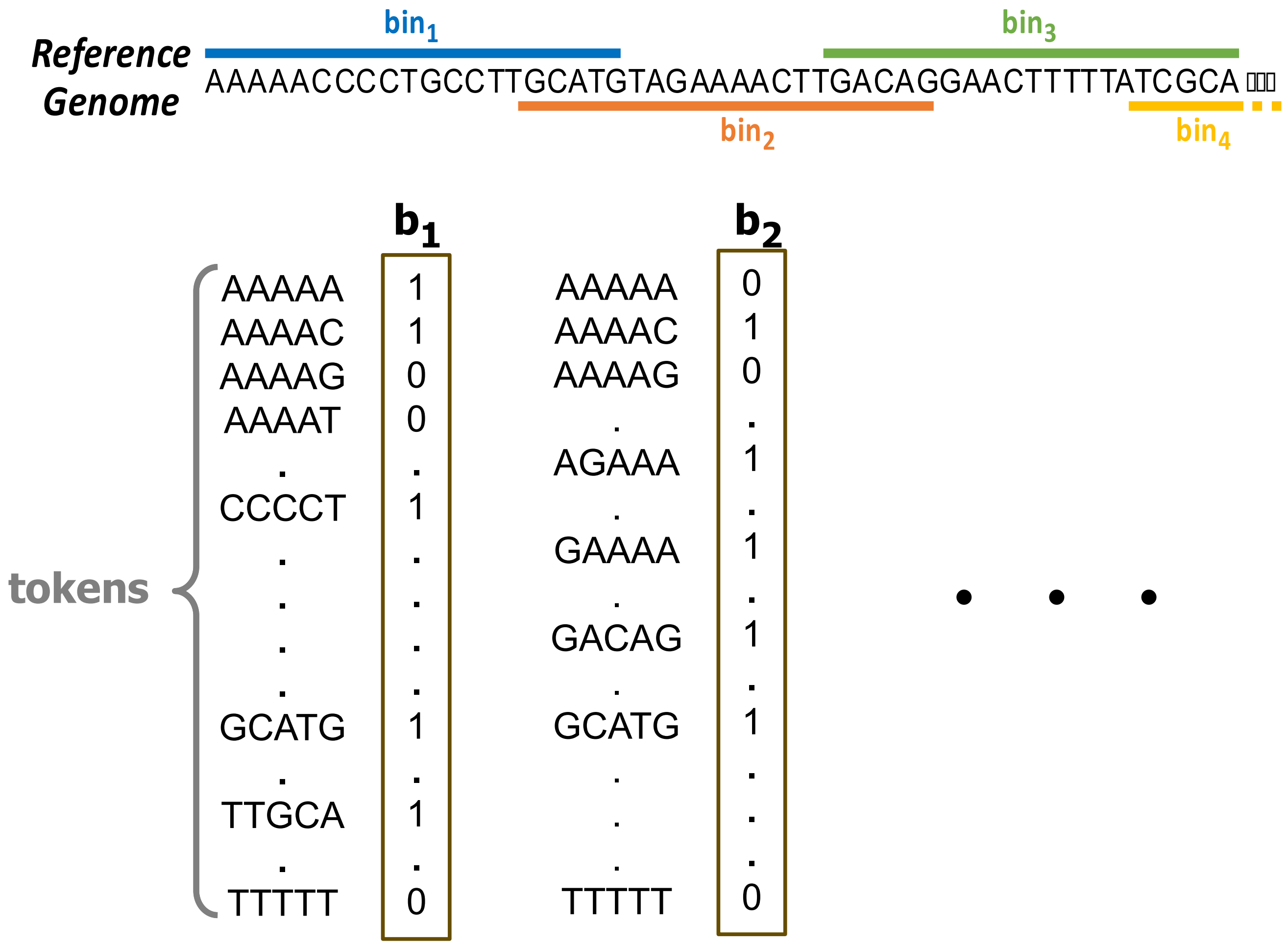
SAFARI

GRIM-Filter: Bitvectors



SAFARI

GRIM-Filter: Bitvectors



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

For **bin size** ~450, and **n** = 5, **memory footprint** ~3.8 GB

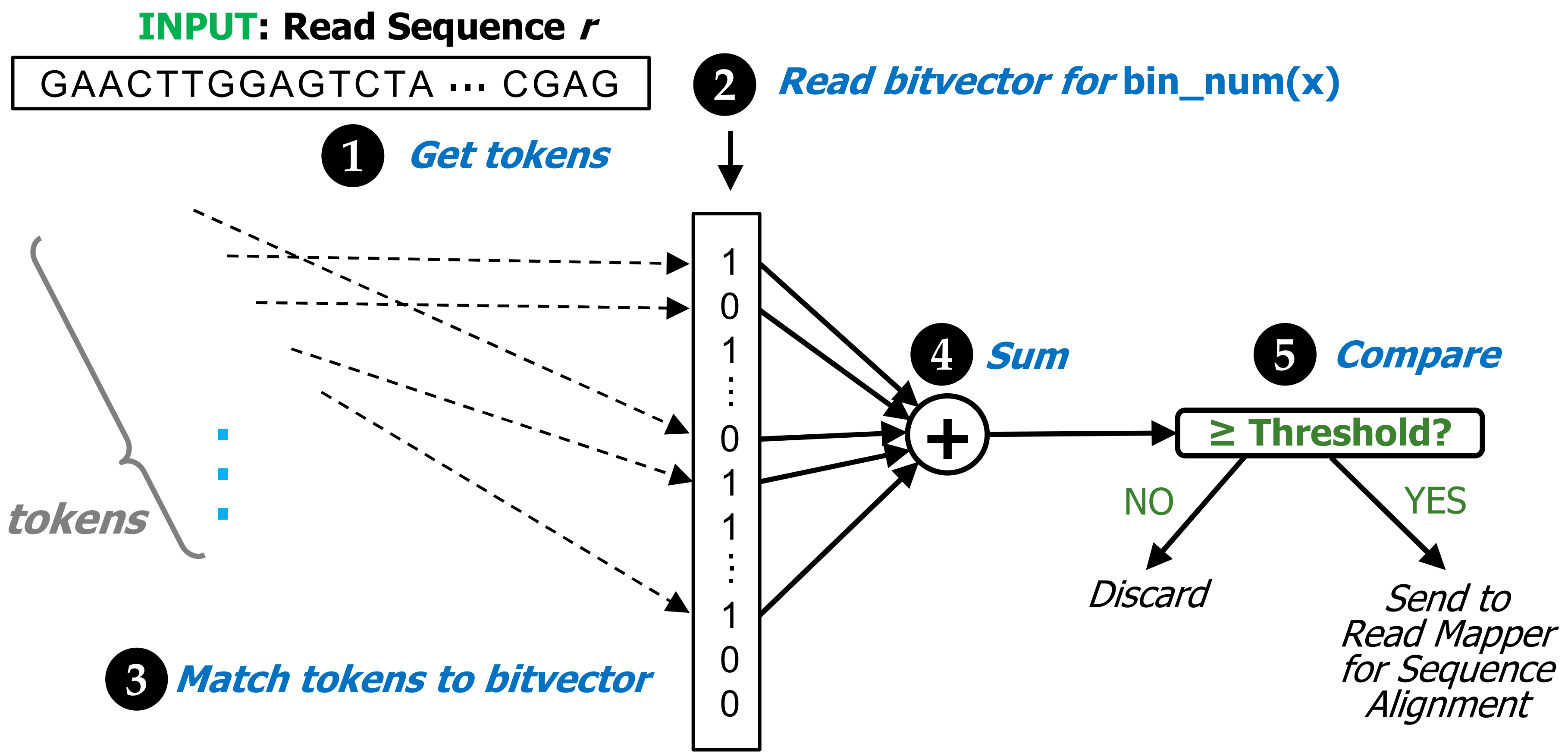
SAFARI

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



SAFARI

Our Proposal: GRIM-Filter

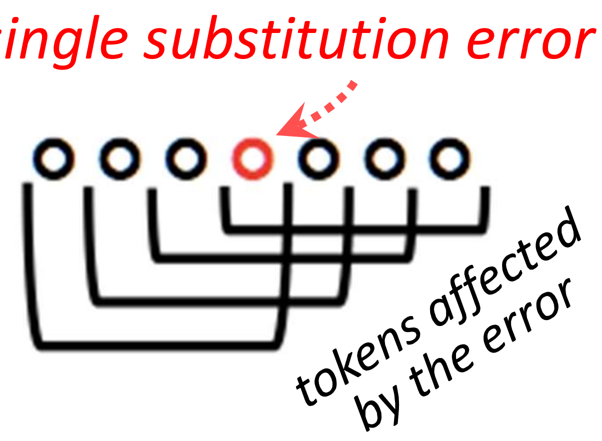
1. Data Structures: Bins & Bitvectors

2. Checking a Bin

3. Integrating GRIM-Filter into a Mapper

GRIM-Filter: Error Tolerance

$$\text{Threshold} = \overbrace{\text{read\_length} - (n-1)}^{\textit{total number of tokens in a read}} - \underbrace{n \times \underbrace{\lceil \text{read\_length} \times e \rceil}_{\textit{number of errors allowed per read}}}_{\textit{maximum number of tokens that could contain errors}}$$



one substitution error affects four tokens when n = 4

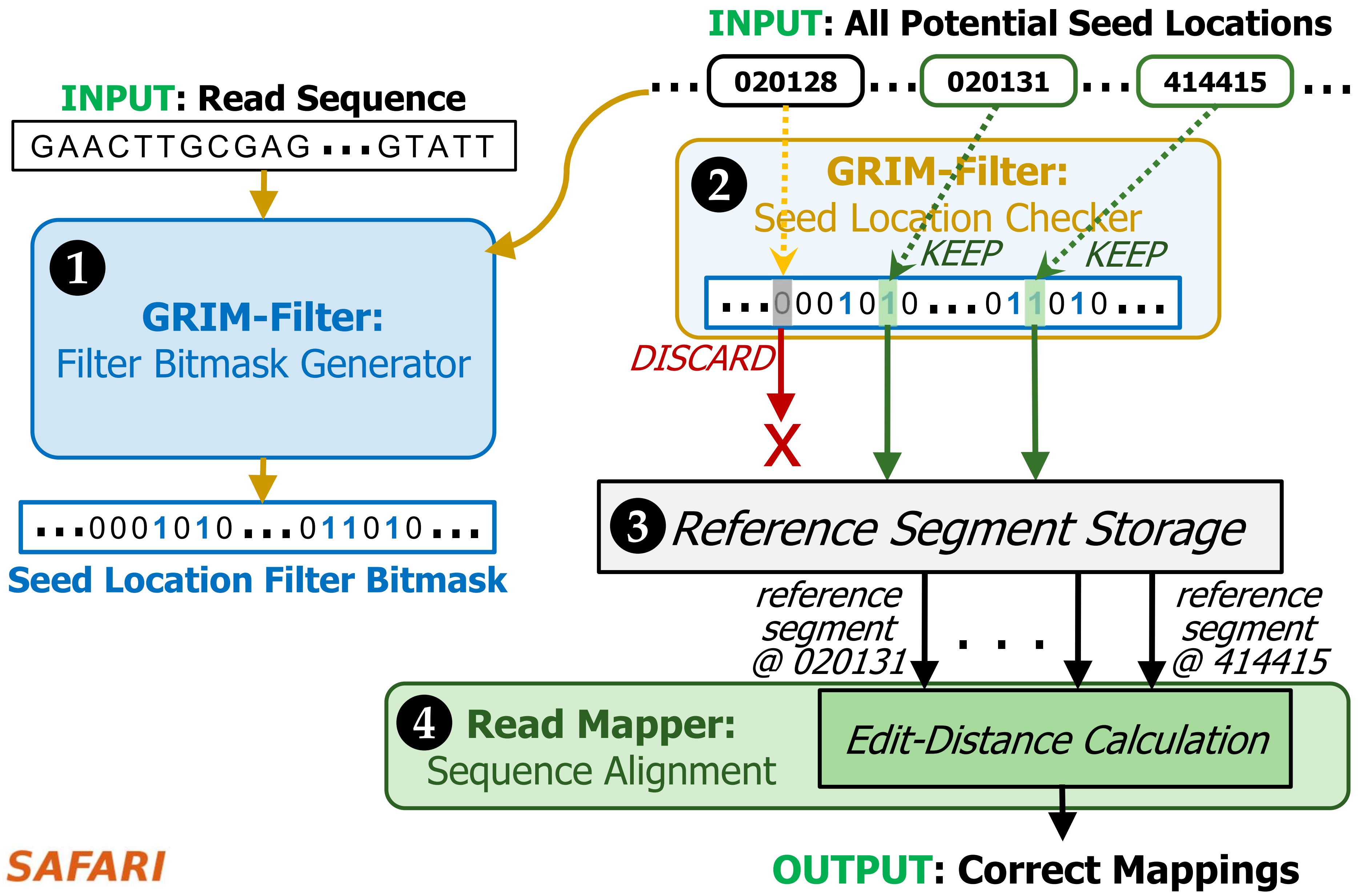
GRIM-Filter can support different error tolerances by simply changing the threshold value

More details in the paper

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



SAFARI

GRIM-Filter Outline

Key Properties of GRIM-Filter

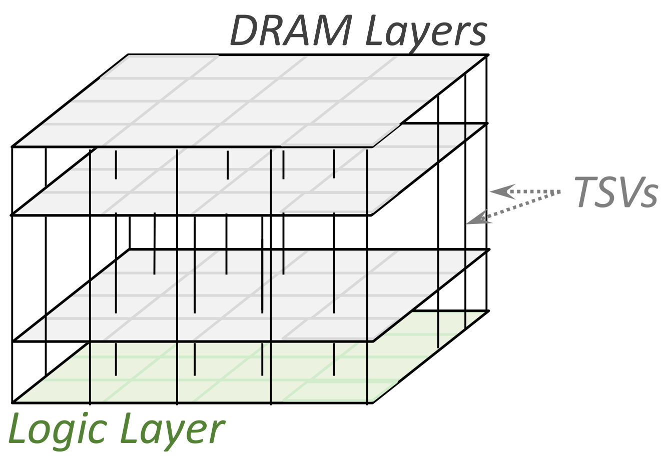
1. **Simple Operations**:
   - To check a given bin, find the **sum** of all bits corresponding to each token in the read
   - **Compare** against threshold to determine whether to align

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

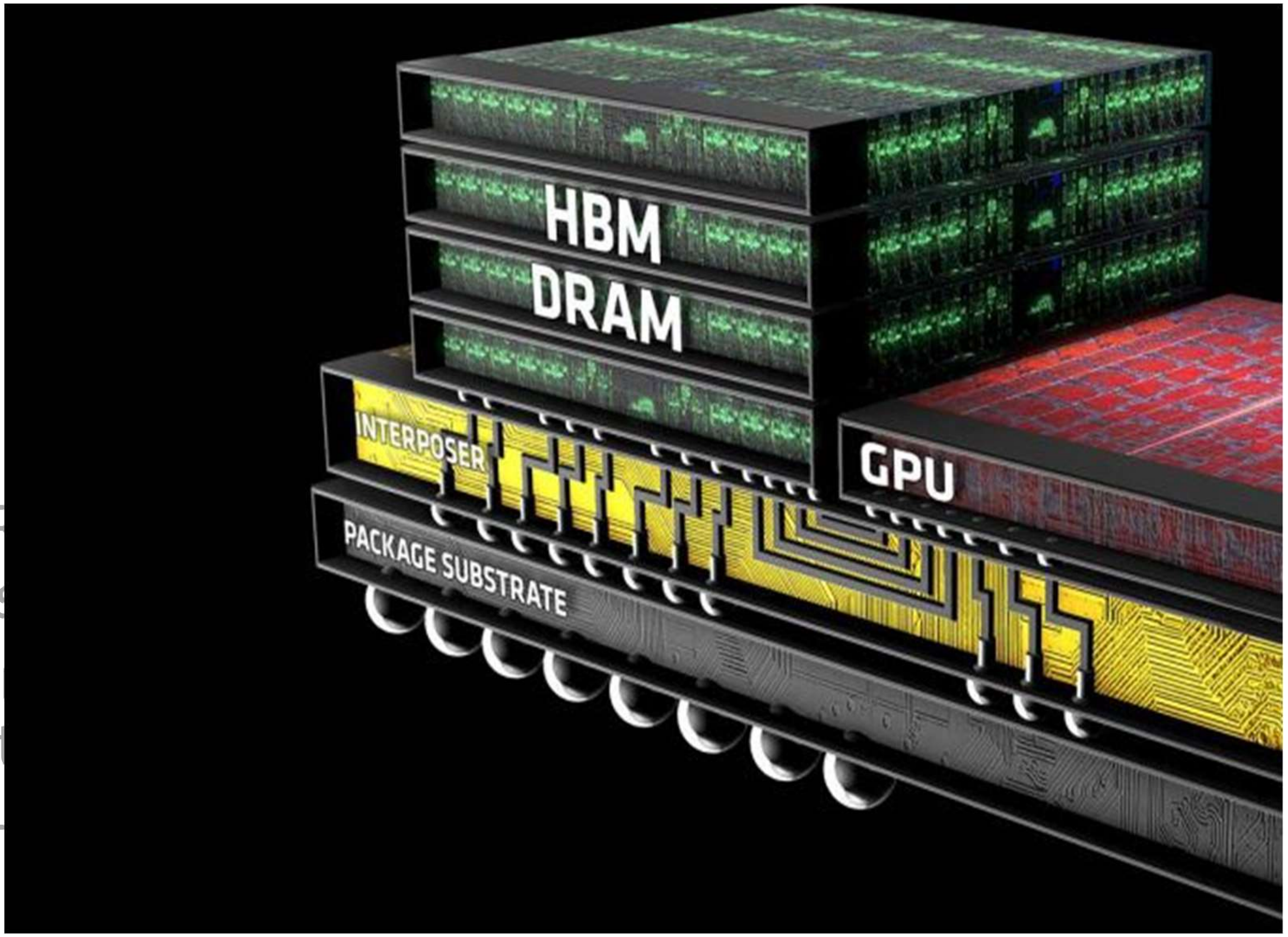


- 3D-Stacked DRAM architecture has **extremely high bandwidth** as well as a stacked customizable logic layer
  - Logic Layer enables **Processing-in-Memory**, offloading computation to this layer and alleviating the memory bus
  - Embed GRIM-Filter operations into **DRAM logic layer** and appropriately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

SAFARI

3D-Stacked Memory

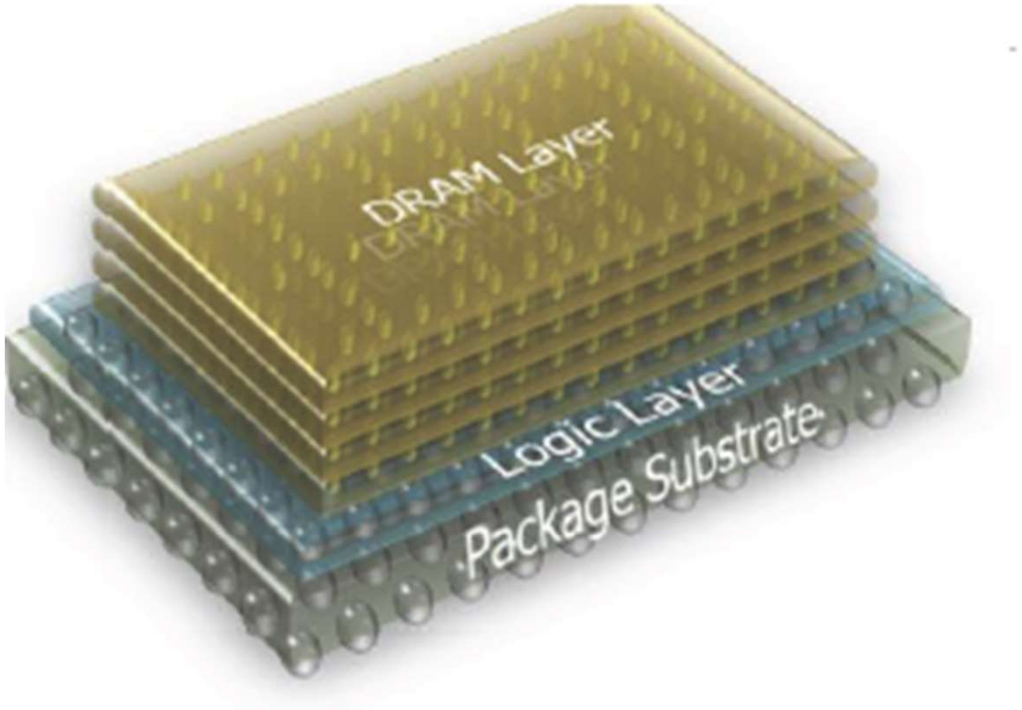
- 3D-Stacked DF
  bandwidth as
  - Logic Layer e
    computation t
  - Embed GRIM-
    appropriately



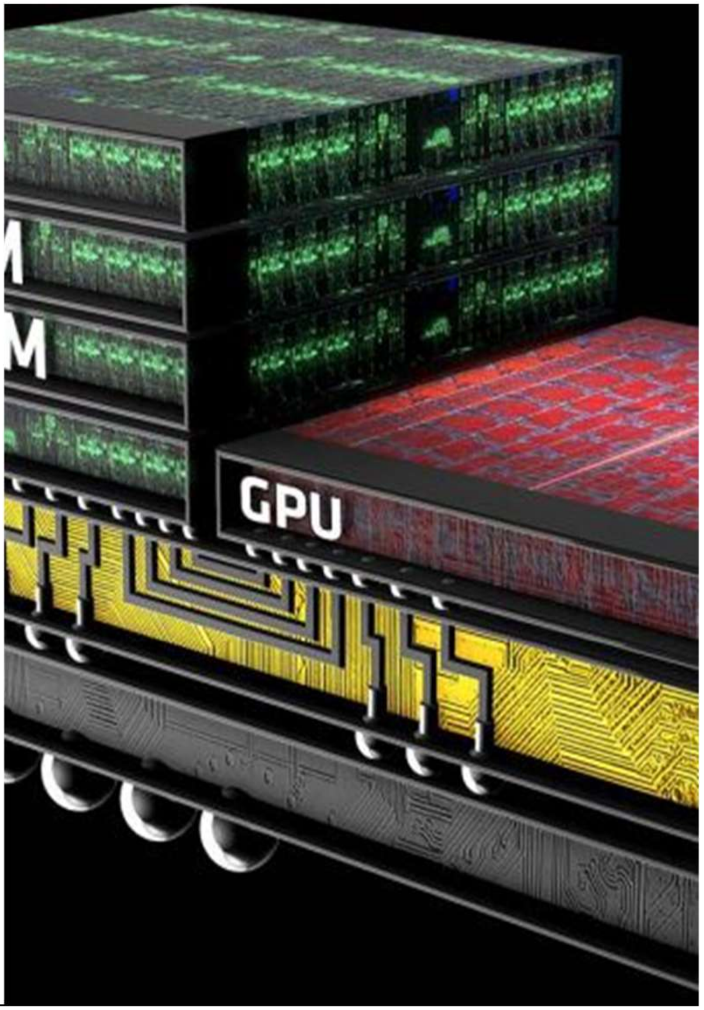
http://i1-news.softpedia-static.com/images/news2/Micron-and-Samsung-Join-Force-to-Create-Next-Gen-Hybrid-Memory-2.png

3D-Stacked Memory

Micron's HMC



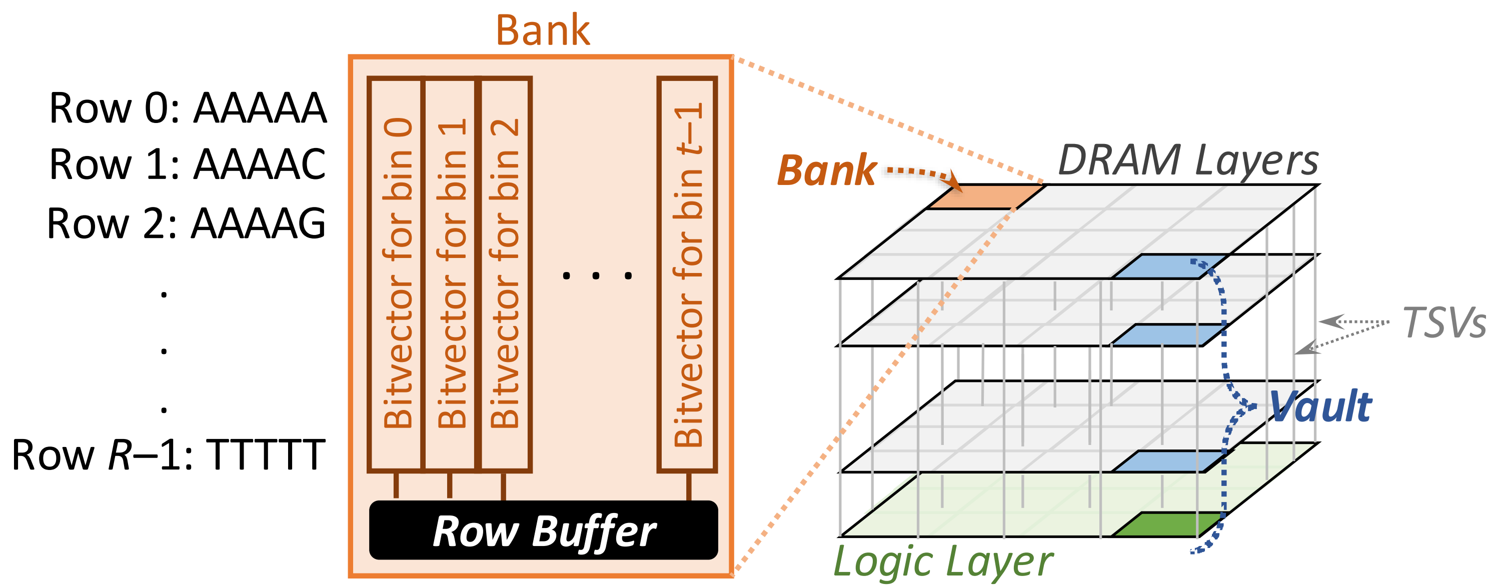
Micron has working demonstration components



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

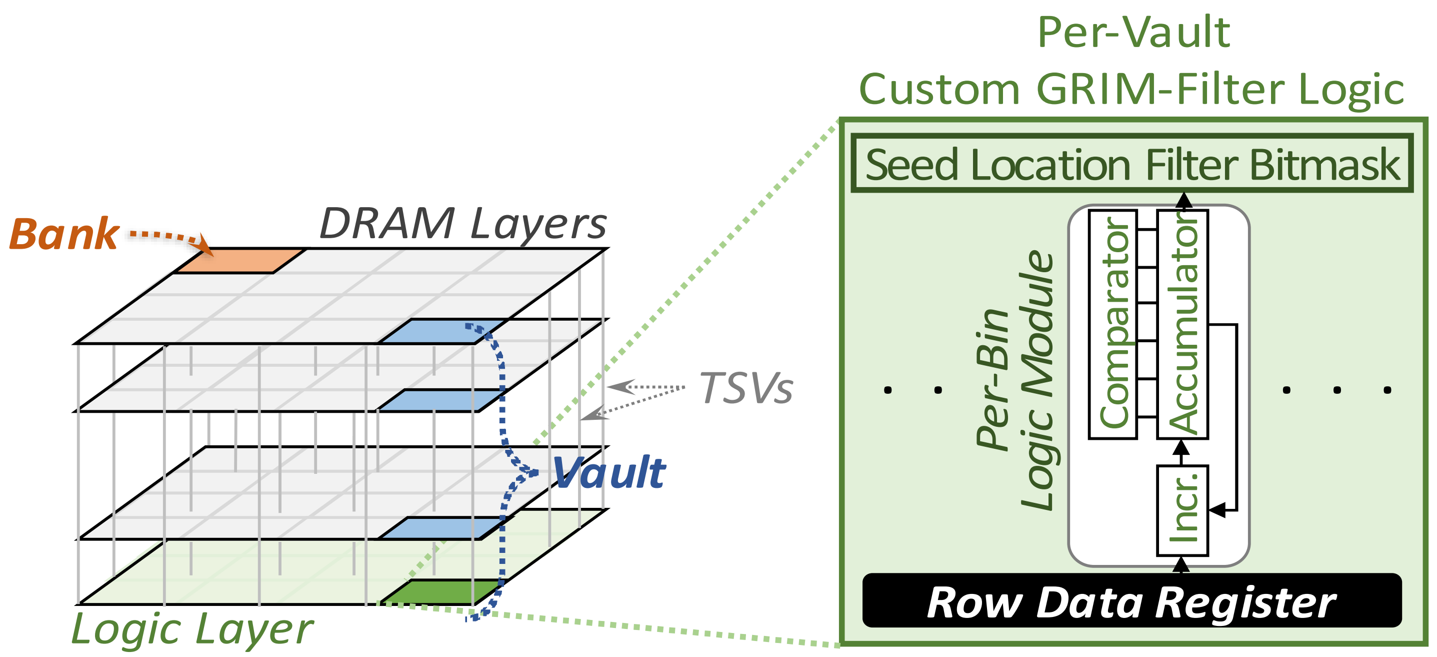


- Each DRAM layer is organized as an array of **banks**
  - A **bank** is an array of cells with a row buffer to transfer data

- The layout of bitvectors in a bank enables filtering many bins in parallel

SAFARI

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

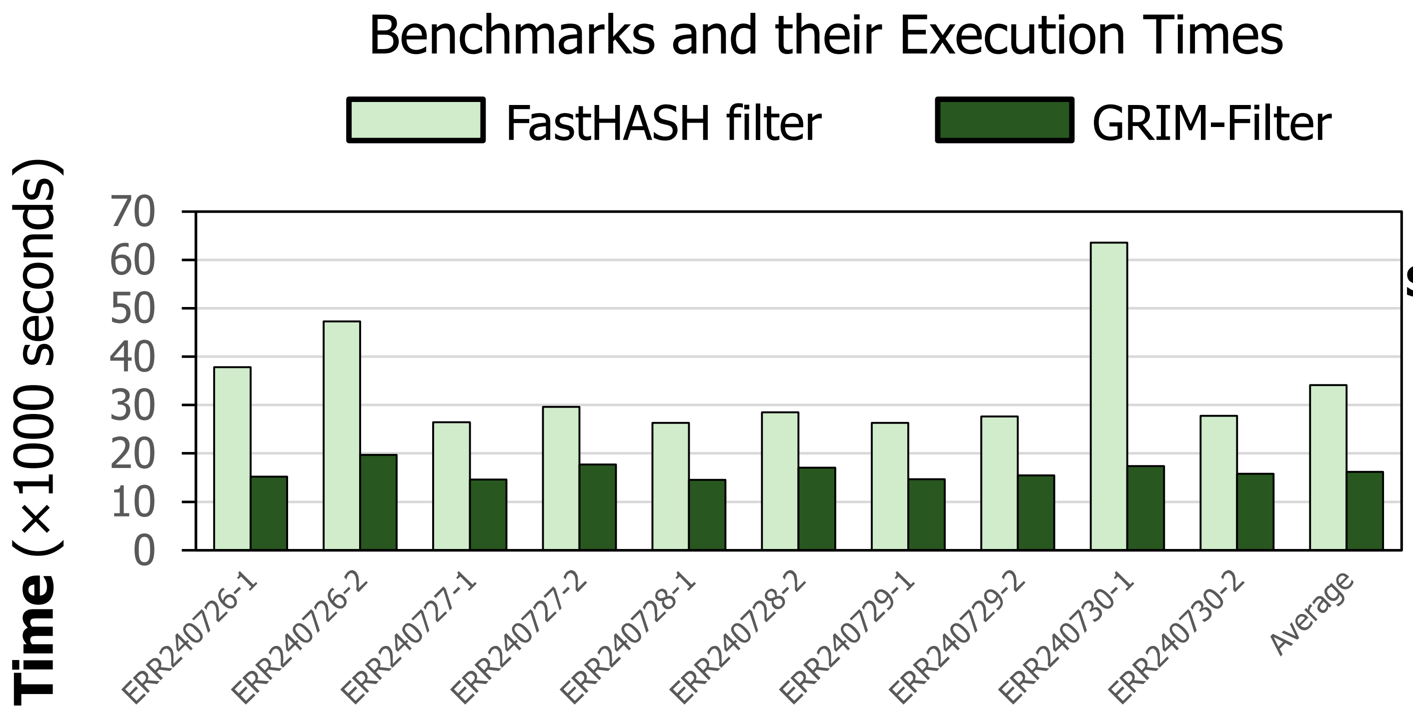
GRIM-Filter Outline

SAFARI

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



Sequence Alignment Error Tolerance (*e*)

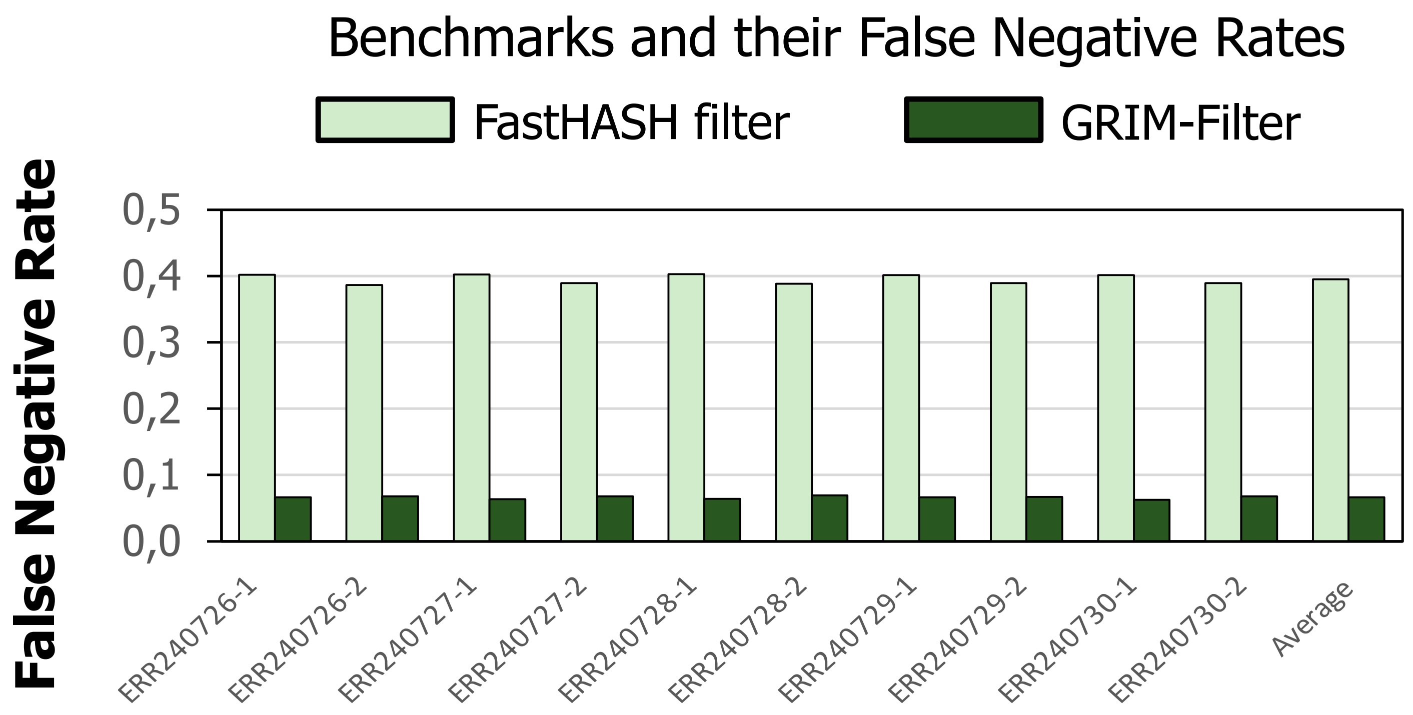
***e* = 0.05**

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



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

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

General Outline

- Summary of the paper
- Strength
- Weaknesses
- Discussion

Strength

- We have a novel idea to improve DNA read mapping
- Could be used if PIM was possible today
- Sensitivity is changeable if needed (different applications possible)
- Good for high error
- Giving background information and future research ideas
- The evaluation of GRIM is clearly defined and explained
- open-source

General Outline

- Summary of the paper
- Strength
- Weaknesses
- Discussion

Weaknesses

- GRIM is evaluated on a Hashing-based read mapper
- How do we input/output data while computing
- In the runtime results we do not count the transformation from reference genome to bit-vector.

General Outline

- Summary of the paper
- Strength
- Weaknesses
- Discussion

SAFARI

Discussion

- Is there a possibility to have a smaller memory footprint?

Discussion

- Is there a possibility to have a smaller memory footprint?
  - Yes, we can take every n token in the reference Genome and multiply the length of the bin span by n.
    - This move could impact our false positive rate of our filter. We can do this since the bit-vector density remains unchanged.

Discussion

- How could we mitigate the problems of slightly longer input parameters or inconsistent lengths?

Discussion

- How could we mitigate the problems of slightly longer input parameters or inconsistent lengths?
  - A small change in the length of the read would not impact Grim since we simply change the threshold.
  - Another possibility ,if the read gets long, is to change the bit-vector span.
  - If the length of the read is to long for all other changes we can cut it in half

Discussion

- Is there a possibility of implementing GRIM efficiently in GPU/FPGA?

Discussion

- Is there a possibility of implementing GRIM efficiently in GPU/FPGA?
  - It is highly unlikely that GRIM would work in a reasonable amount of time since we need a high bandwidth which the FPGA / GPU cannot support.
  - Be 400 GB/s bandwidth(bus bandwidth not internal bandwidth, I assume the bandwidth similar and that we run on 4 Mherz)
  - =>838 bits per cycle
  - how do we store the bits efficiently in the GPU?

Discussion

- With long reads is Grim a good option in this configuration? If not could we change it?

Discussion

- With long reads is Grim a good option in this configuration? If not could we change it?
  - No it is not we would need around 70 Banks to hold 1 read (286 720 Banks required if we want to use the 4096 bandwidth).
  - What could we change? we could change the interconnection between bins, change the span of bins, and allow for other filtering possibilities

Discussion

- Can we change the fundamental working procedure to accommodate for long reads?

Discussion

- Can we change the fundamental working procedure to accommodate for long reads?
  - We could change the filtering system of Grim, we know that our DNA has a specific order of garbage and useful DNA sequences.
    - If we know these in the reference Genome and our genome with the locations we found we could filter some of our locations out since their pattern of useful and garbage DNA is not the one of our read
    - In conjunction with the first 1,000 -10,000 bp we could find reasonable amount locations.
  - Another possibility is the cut the reads
    - save their location in relation to each other and use smaller reads combined with CKS/AF we could filter more thoroughly
  - Another possibility is to allow long reads to be cut in multiple pieces and to use Grim filter on those shorter reads. We could allow for independent or dependent thresholds.
  - => we would need 4 GRIM filters for a cut which results in 4 smaller read

Discussion

- Can we speedup GRIM?

SAFARI

Discussion

- Can we speedup GRIM?
  - Since GRIM speed is dependent on the bandwidth of 3D-stacked memory it would be possible to speed up Grim by increasing the bandwidth of the 3D-stacked memory
  - Another possibility could be to check every n token from the input read(token)
  - Another possibility would be the balancing of the runtimes of GRIM and of read alignment since currently the bottleneck is GRIM.