

SneakySnake

A Fast and Accurate Universal Genome Pre-Alignment Filter for CPUs, GPUs, and FPGAs

Bioinformatics Journal, Volume 36

Authors:

Mohammed Alser^{1,2}, Taha Shahroodi¹,
Juan Gómez-Luna^{1,2}, Can Alkan⁴,
Onur Mutlu^{1,2,3,4}

1 Department of Computer Science, ETH Zurich

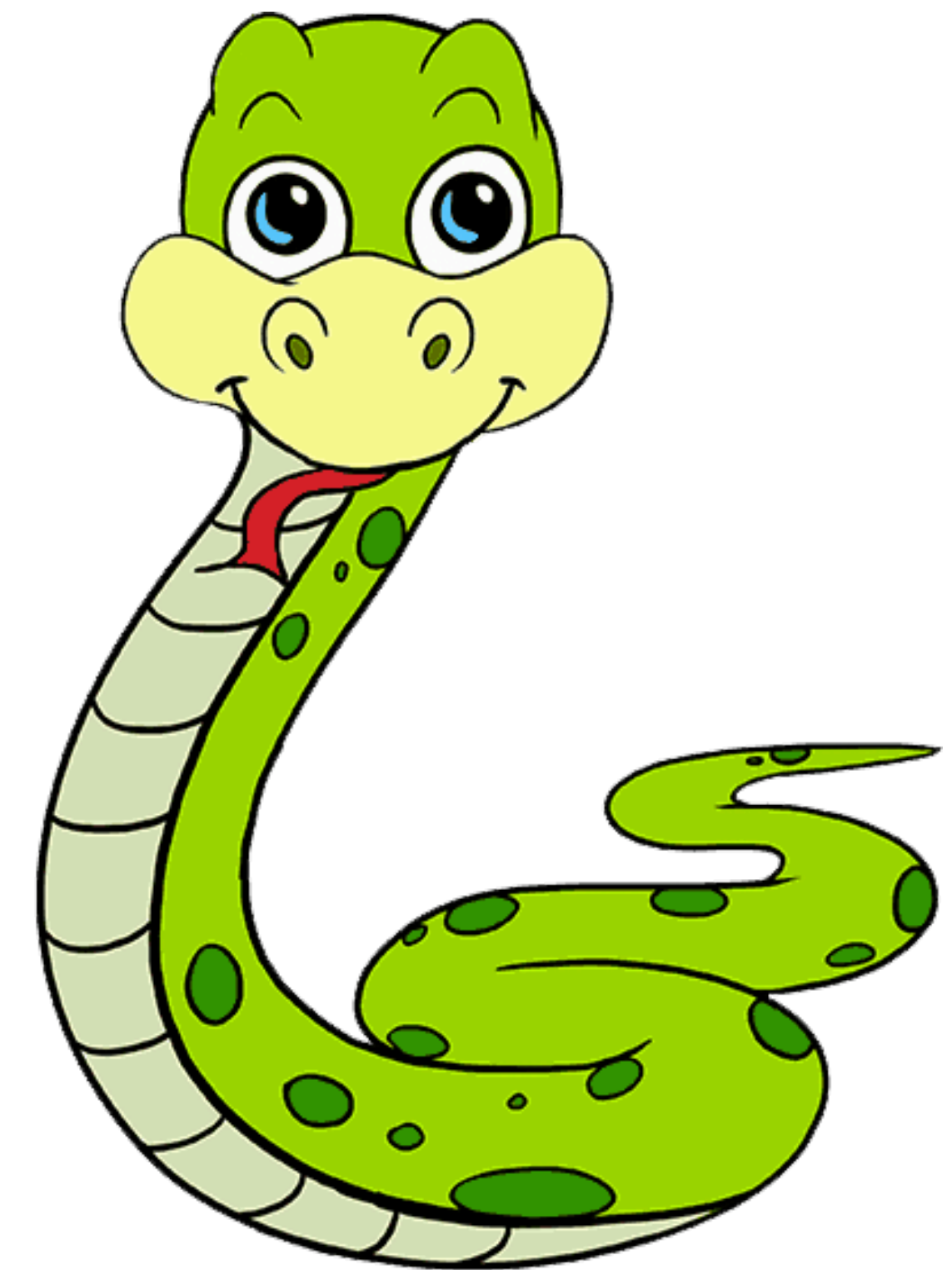
2 Department of Information Technology and Electrical Engineering, ETH Zurich

3 Department of Electrical and Computer Engineering, Carnegie Mellon University

4 Department of Computer Engineering, Bilkent University

Presented by:

Robert Veres



May 6, 2021

What is SneakySnake?

What is SneakySnake?

Subject Section

SneakySnake: A Fast and Accurate Universal Genome Pre-Alignment Filter for CPUs, GPUs, and FPGAs

**Mohammed Alser^{1,2,*}, Taha Shahroodi¹, Juan Gómez-Luna^{1,2},
Can Alkan^{4,*}, and Onur Mutlu^{1,2,3,4,*}**

¹Department of Computer Science, ETH Zurich, Zurich 8006, Switzerland

²Department of Information Technology and Electrical Engineering, ETH Zurich, Zurich 8006, Switzerland

³Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh 15213, PA, USA

⁴Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey

*To whom correspondence should be addressed.

Associate Editor: XXXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

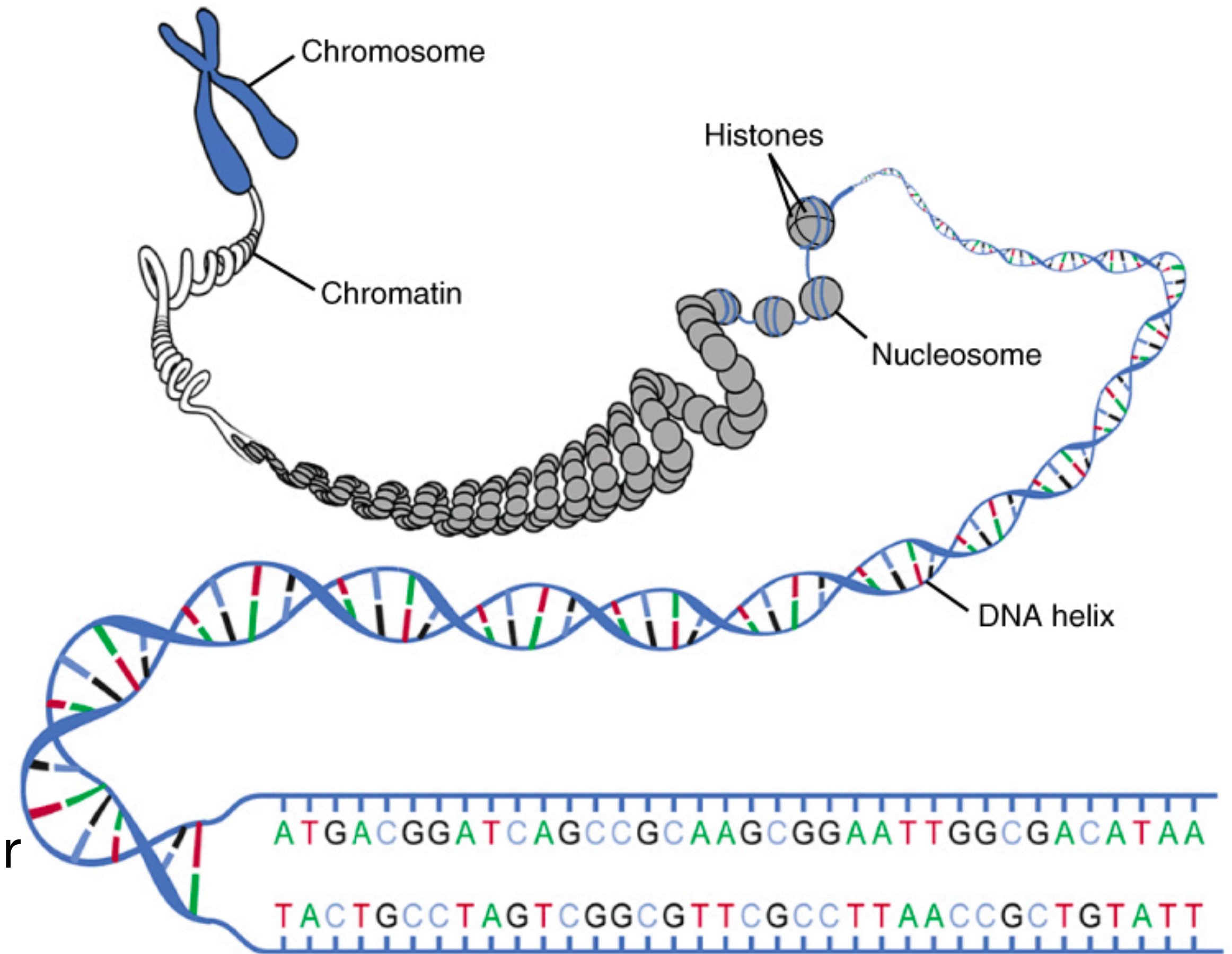
Abstract

Motivation: We introduce *SneakySnake*, a highly parallel and highly accurate pre-alignment filter that remarkably reduces the need for computationally costly sequence alignment. The key idea of SneakySnake is to reduce the *approximate string matching* (ASM) problem to the *single net routing* (SNR) problem in VLSI chip layout. In the SNR problem, we are interested in finding the optimal path that connects two terminals with the least routing cost on a special grid layout that contains obstacles. The SneakySnake algorithm quickly solves the SNR problem and uses the found optimal path to decide whether or not performing sequence alignment is necessary. Reducing the ASM problem into SNR also makes SneakySnake efficient to implement on CPUs, GPUs, and FPGAs.

Background

Recap from 10th grade biology

- Your most important attributes are written in your chromosomes.
(Eye colour, gender, but even your immune reactions to COVID)
- Your **chromosomes are** just very long strands of **DNA**
- If we can read your DNA, we can tell a lot more about you.
- If we could read the DNA of multiple people, we could **tell even more about you** after reading your DNA
 - ➡ We want to read the entire DNA of multiple people



Recap from 4th session

Seminar in Computer Architecture Meeting 4: GateKeeper

Dr. Mohammed Alser

ALSERM@ethz.ch

ETH Zürich

Spring 2021

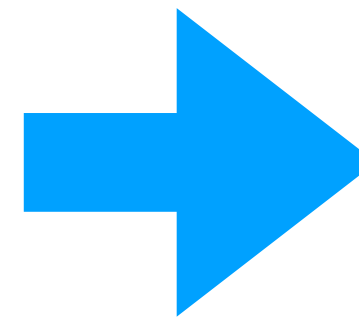
18 March 2021

Recap from 4th session

Seminar in Computer Architecture Meeting 4: GateKeeper

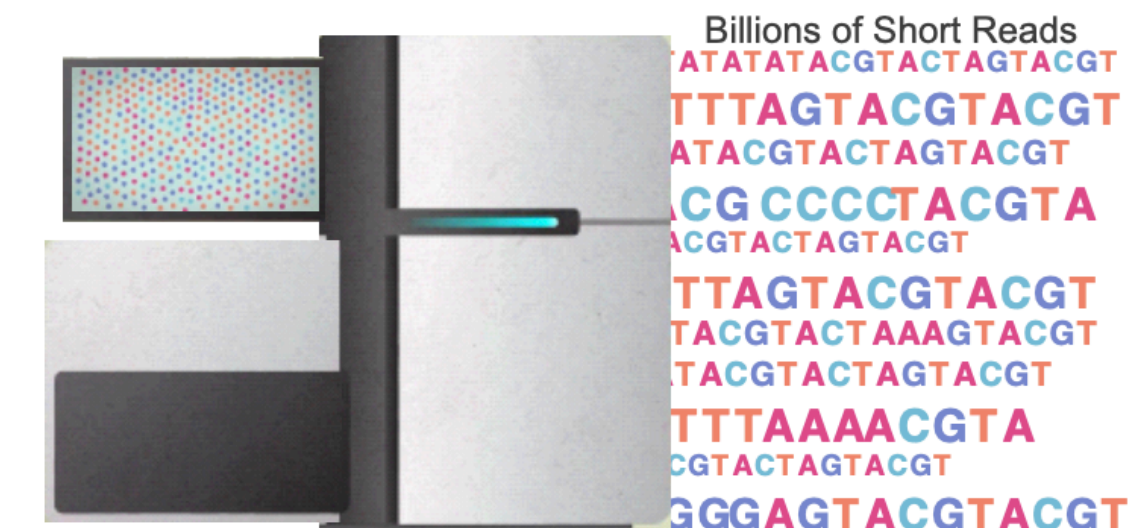
Dr. Mohammed Alser
ALSERM@ethz.ch

ETH Zürich
Spring 2021
18 March 2021



Genome Sequencer is a Chopper

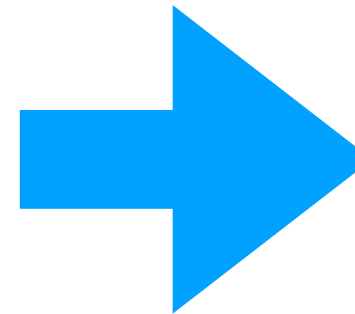
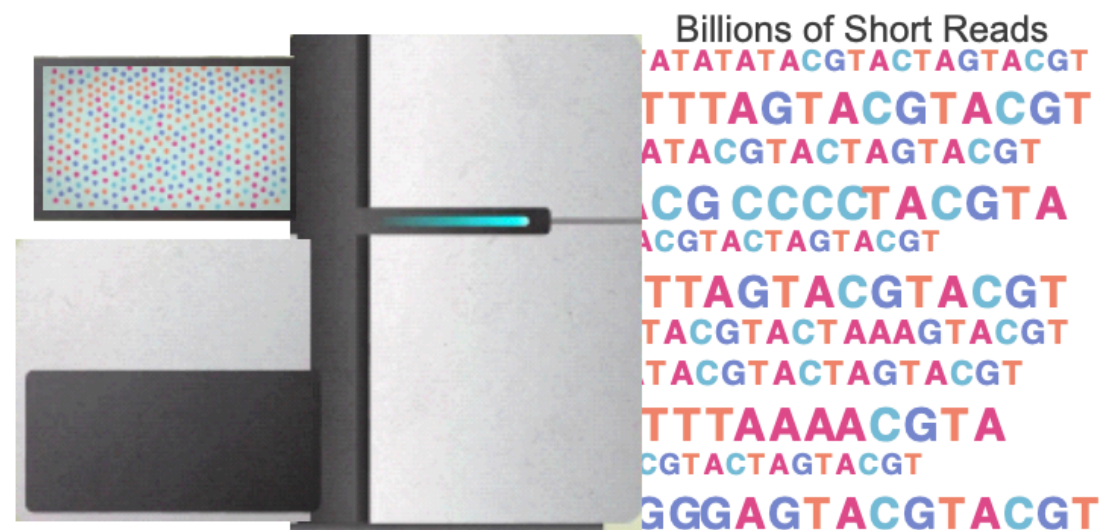
Regardless the sequencing machine,
reads still lack information about their order and location
(which part of genome they are originated from)



Recap from 4th session

Genome Sequencer is a Chopper

Regardless the sequencing machine,
reads still lack information about their order and location
(which part of genome they are originated from)



Solving the Puzzle

.FASTA file



Reference
genome

.FASTQ file



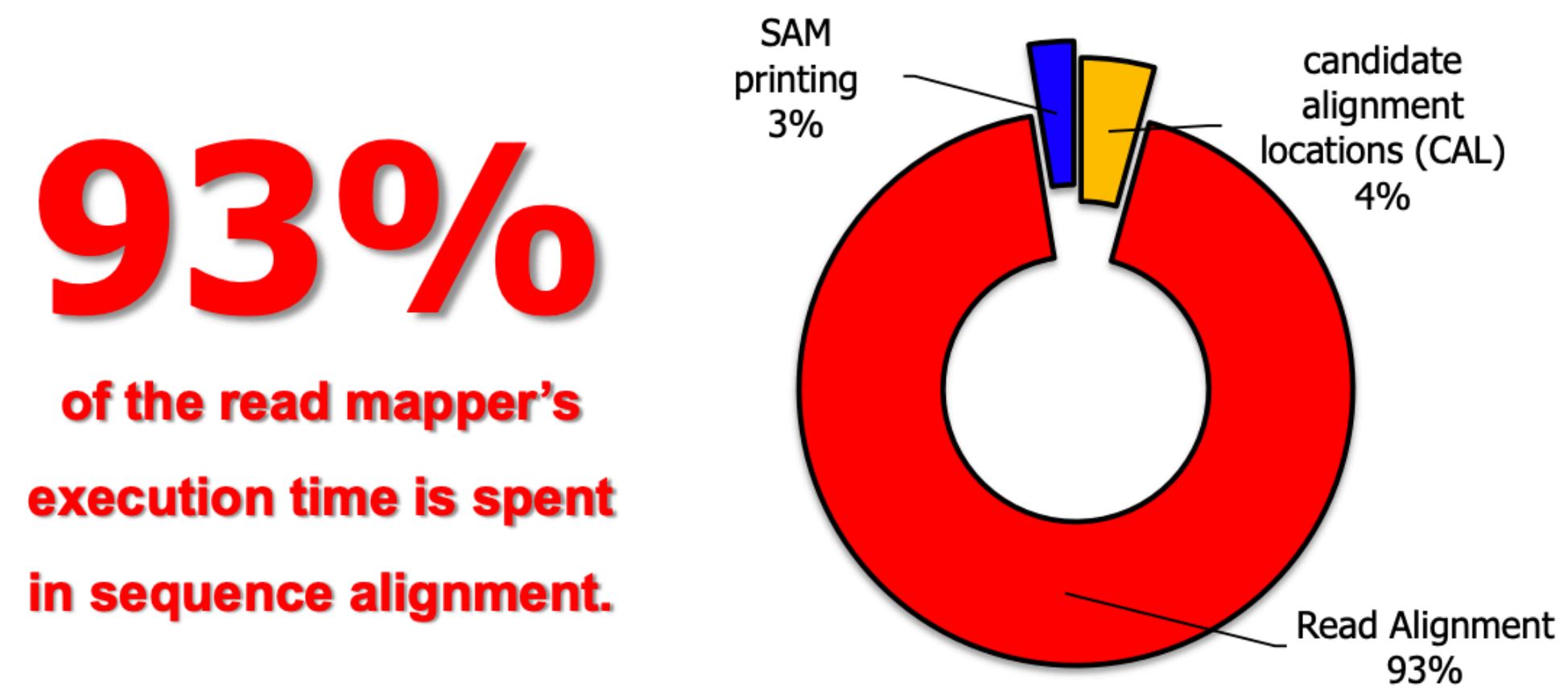
Reads

<https://www.pacb.com/smrt-science/smrt-sequencing/hifi-reads-for-highly-accurate-long-read-sequencing/>

Recap from 4th session

What Makes Read Mapper Slow?

Key Observation # 1



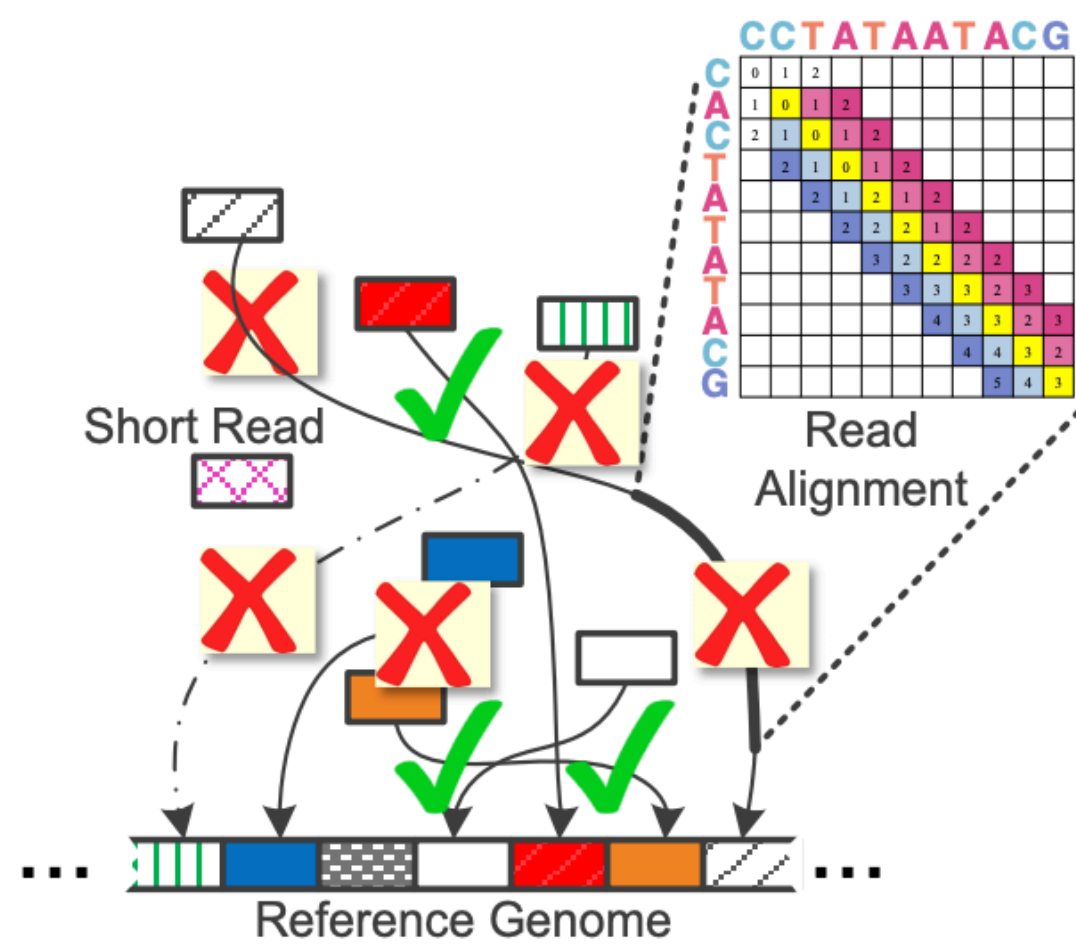
93%
of the read mapper's
execution time is spent
in sequence alignment.

Alser et al, Bioinformatics (2017)

Key observation #1

What Makes Read Mapper Slow? (cont'd)

Key Observation # 2



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

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

Key observation #2

Sequence-Alignment approach 1

- Most sequence alignment approaches are implemented as **dynamic programming** algorithms with **quadratic time complexity**
 - We can use a special **hardware** to **accelerate** the procedure
- e.g. SIMD capable processors used by Parasail or processing in memory Architecture such as GenASM

Sequence-Alignment approach 2

- Most sequence alignment approaches are implemented as **dynamic programming** algorithms with **quadratic time complexity**
- Introduce **pre-alignment filters** that reduce the need for DP by eliminating dissimilar strings

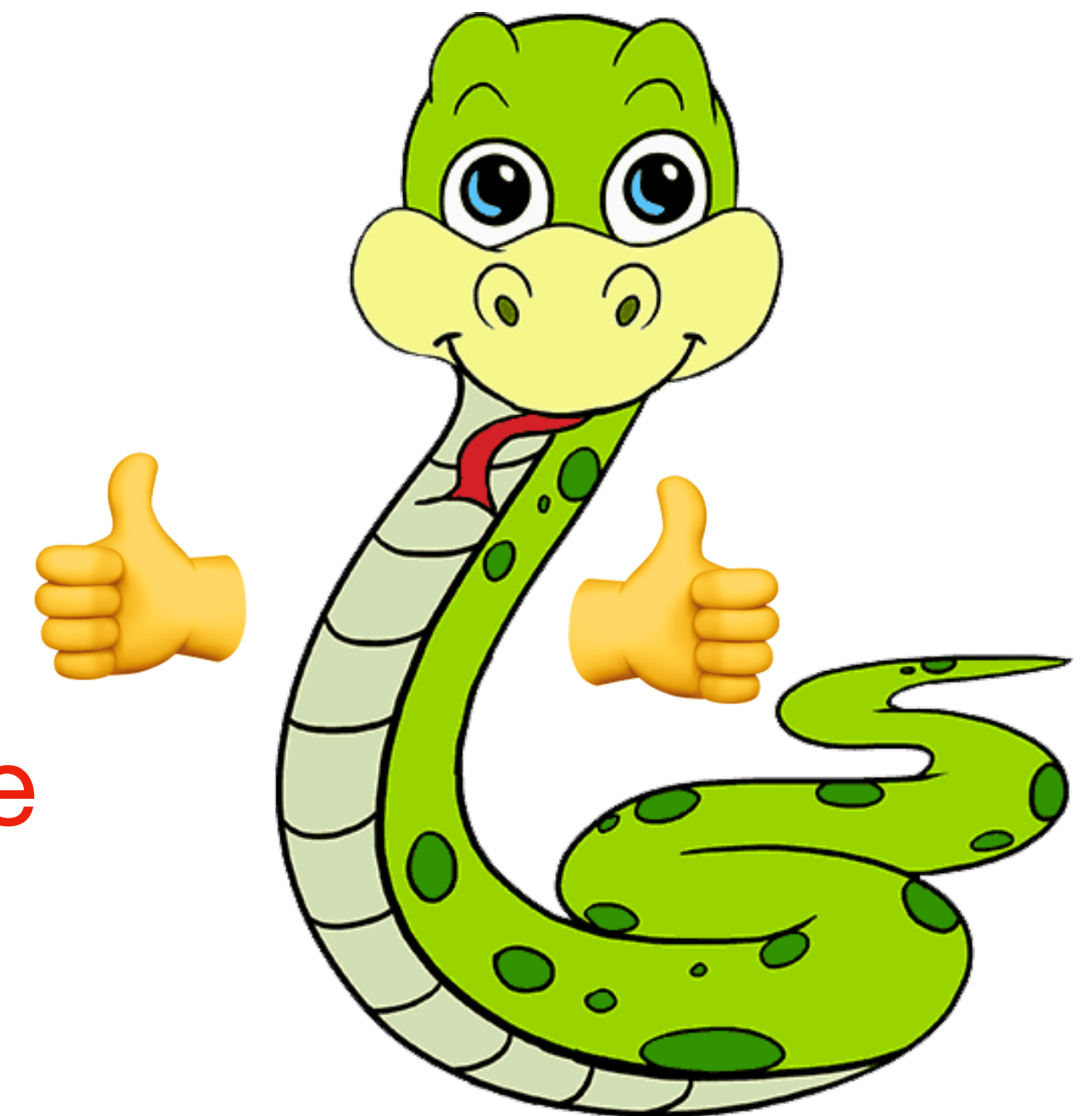
e.g. SHD or GateKeeper

However these are expensive and inaccurate

The goal of SneakySnake

- Highly accurate pre-alignment filter to help us distinguish between similar and dissimilar Strings, that we can ignore
- Should work for both short and long sequences
- Highly parallelizable
- Deployable on a lot of platforms

Eliminate dissimilar strings via solving the Approximate String matching problem



How does SneakySnake work?

How does SneakySnake work?

How does SneakySnake work?



Approximate String Matching (ASM)

How does SneakySnake work?



Approximate String Matching (ASM)



Single Net Routing (SNR)

How does SneakySnake work?

- Reduce ASM problem to SNR

How does SneakySnake work?

- Reduce ASM problem to SNR
- Solve the SNR problem

How does SneakySnake work?

- Reduce ASM problem to SNR
- Solve the SNR problem
- ???
- Profit!

Reducing ASM to SNR

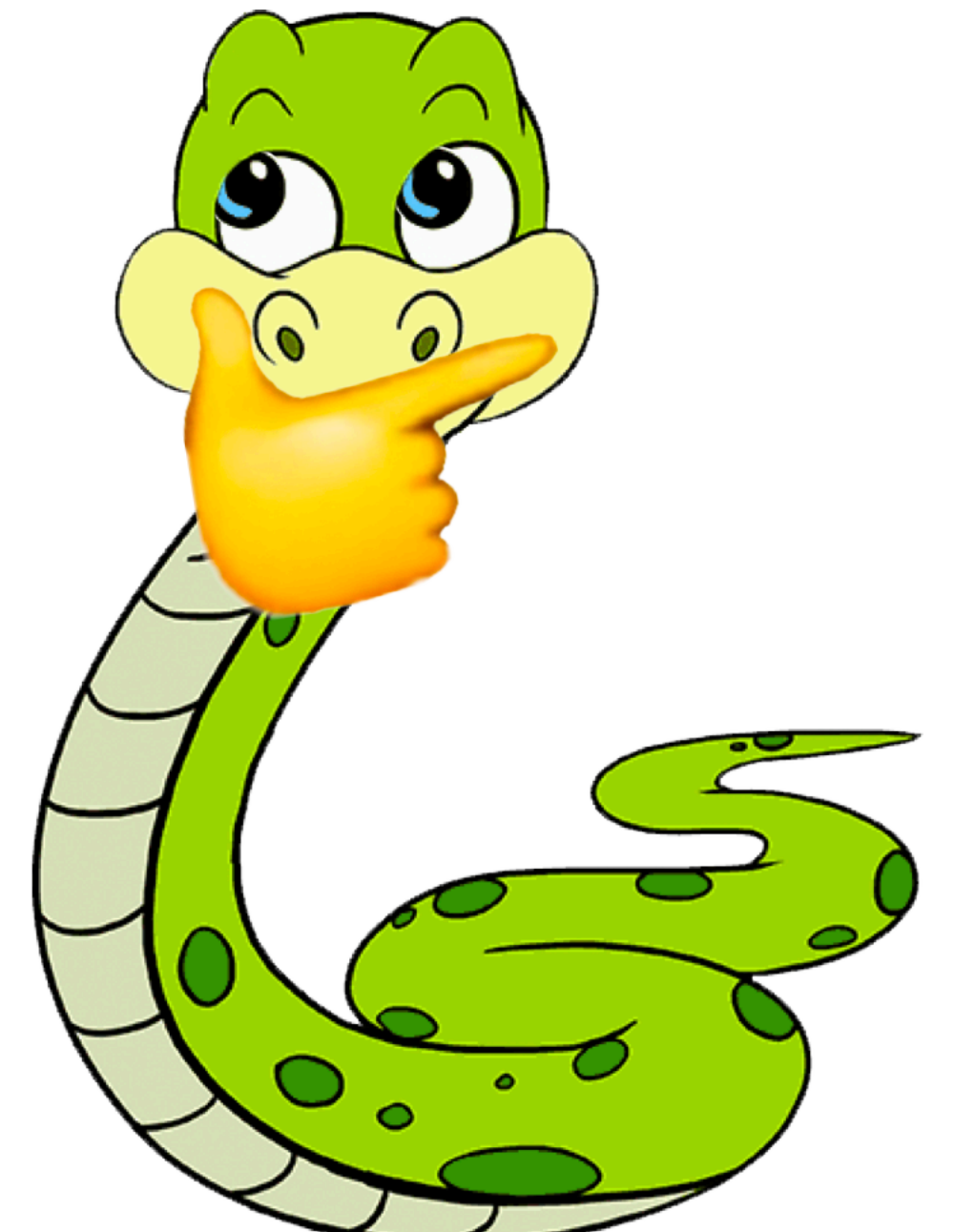
Step 1: Replace the DP-table with `chip-mazeTM`

Reducing ASM to SNR

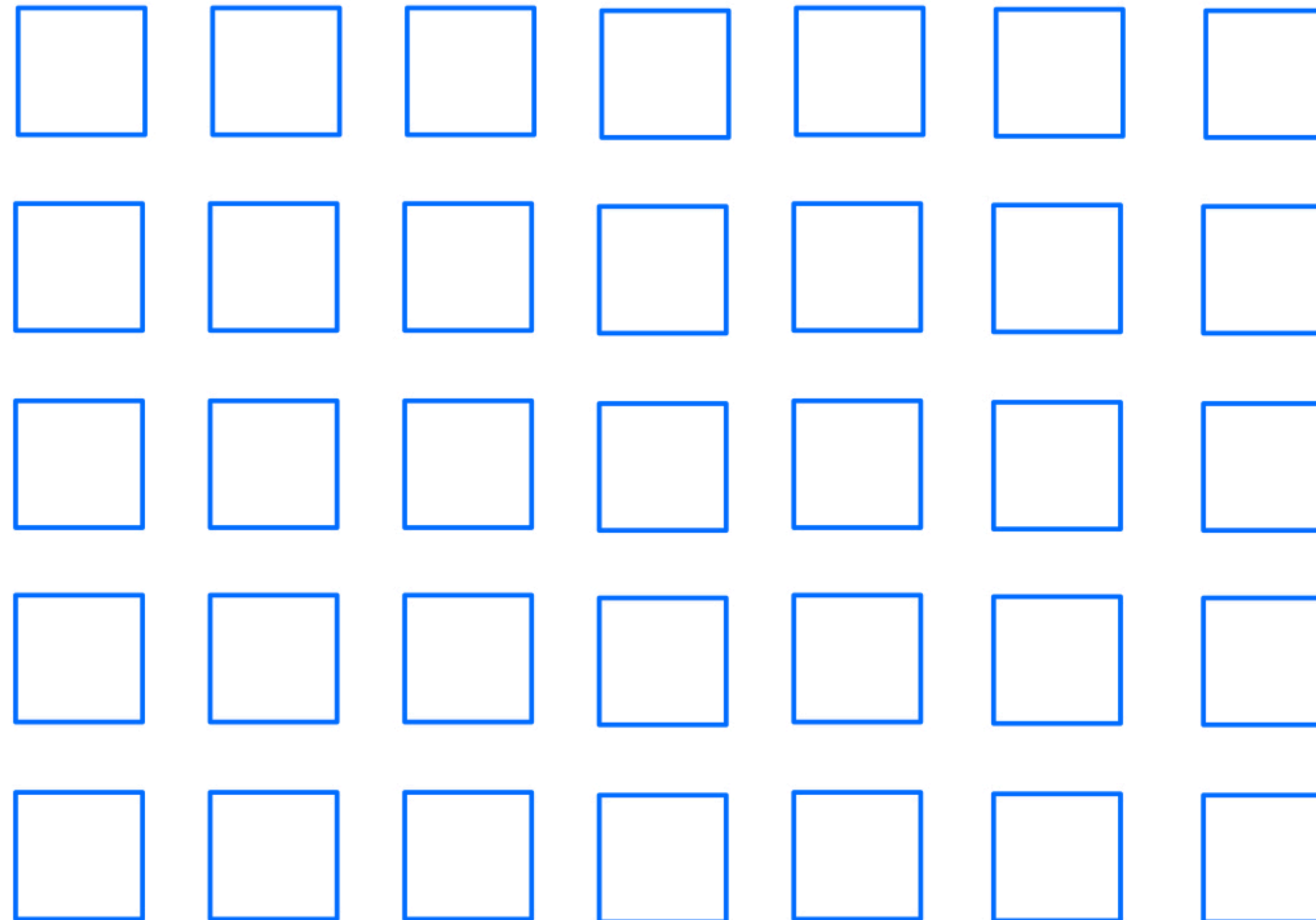
Step 1: Replace the DP-table with **chip-mazeTM**

Step 2: Find the number of differences between two sequences by solving the **SNR problem** in the chip-mazeTM

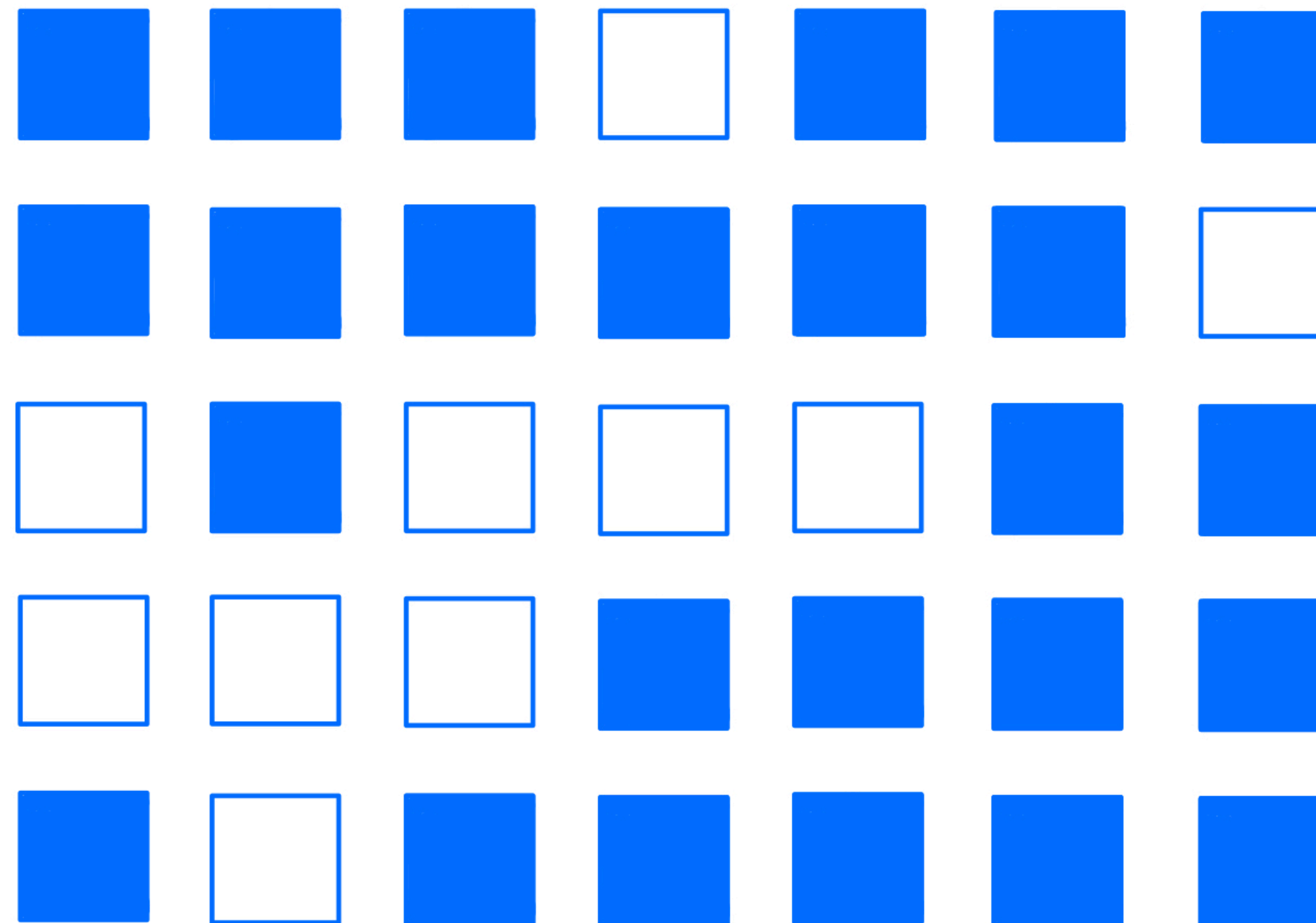
What are those?



The Single Net Routing problem

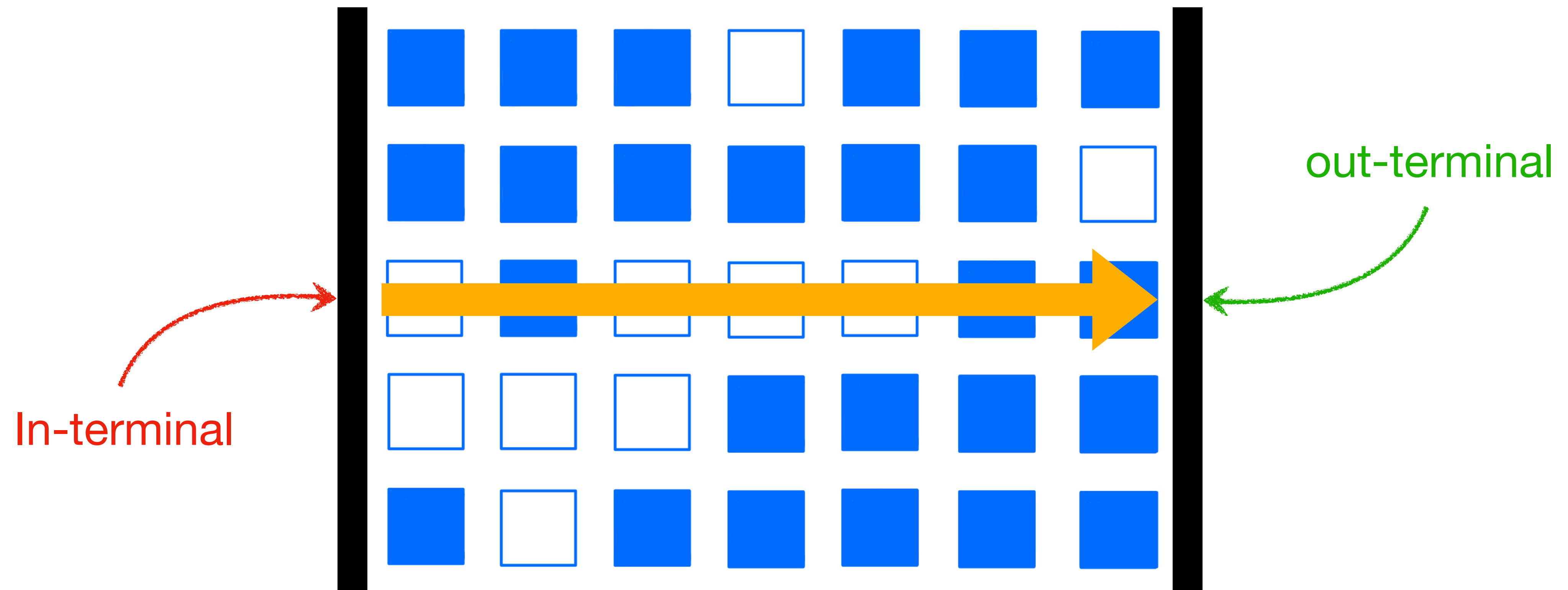


The Single Net Routing problem

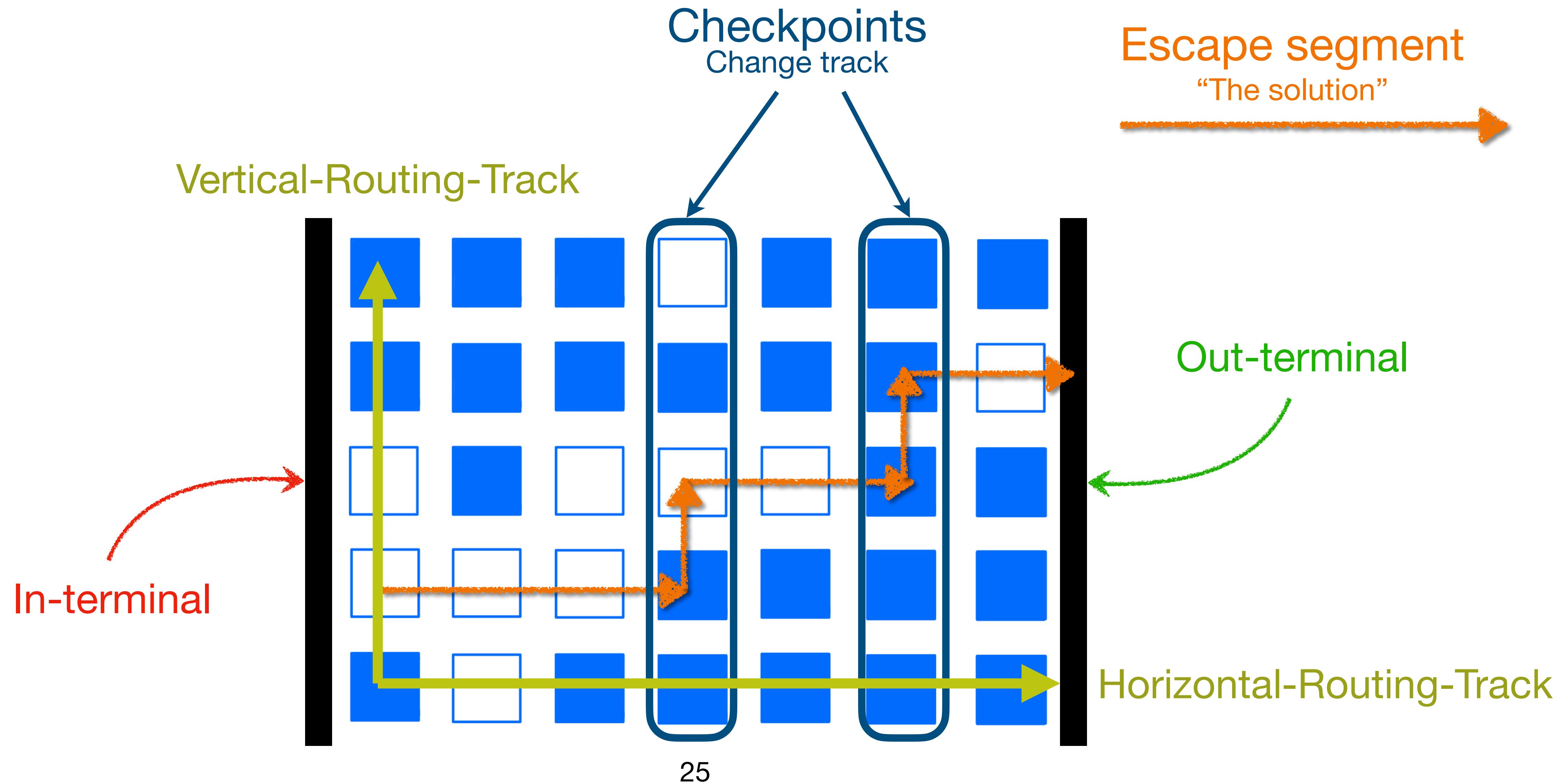


The Single Net Routing problem

Goal: getting from the in- to the end-terminal
with the least amount of obstacles possible



The Single Net Routing problem



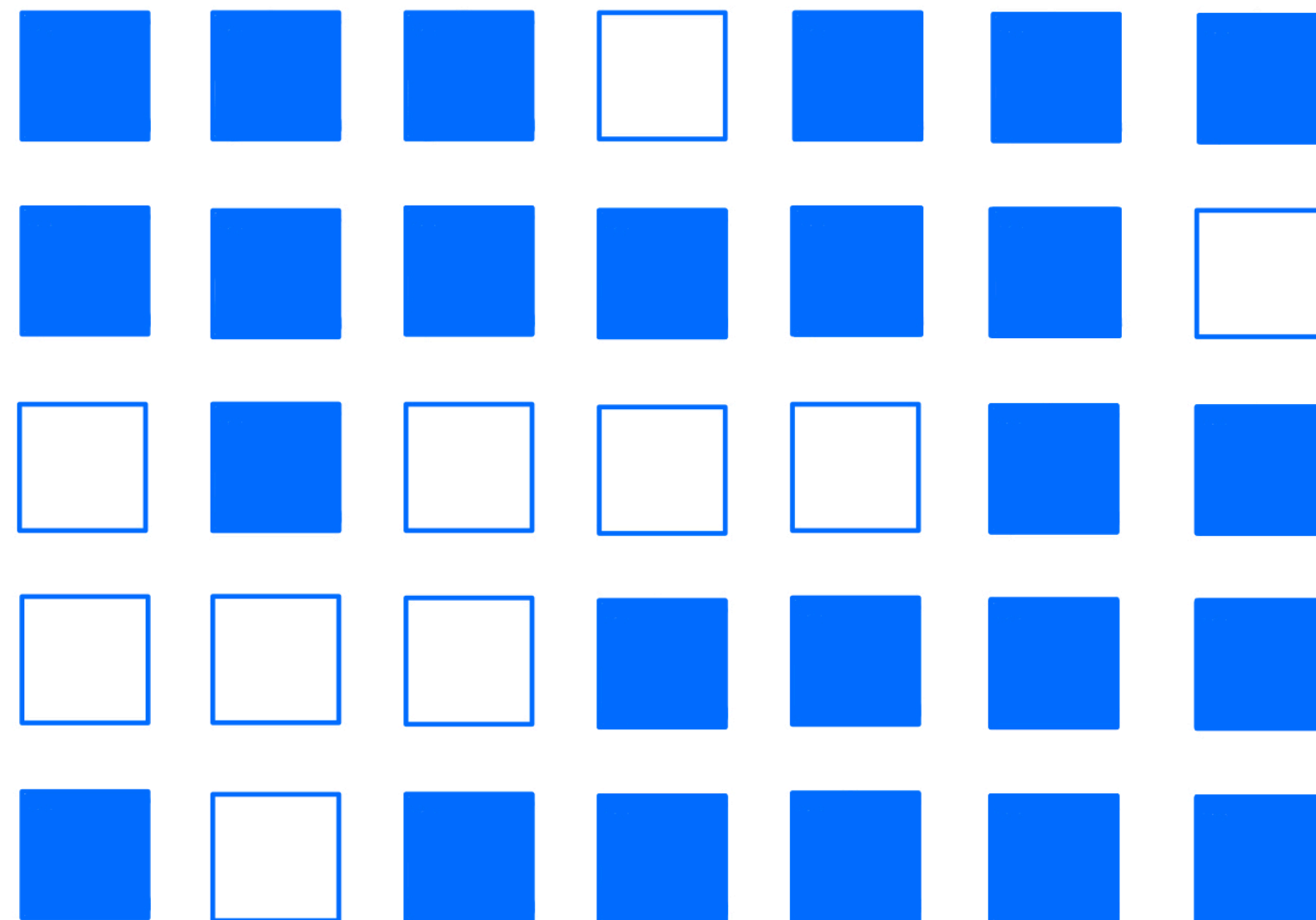
The chip-maze

Replace $(m+1) \times (m+1)$ matrix with $(2E+1) \times m$ where $Z_{i,j}$ is defined as:

$$Z[i, j] = \begin{cases} \square & \text{if } i = E + 1, Q[j] = R[j], \\ \square & \text{if } 1 \leq i \leq E, Q[j - i] = R[j], \\ \square & \text{if } i > E + 1, Q[j + i - E - 1] = R[j], \\ \blacksquare & \text{otherwise} \end{cases} \quad (1)$$

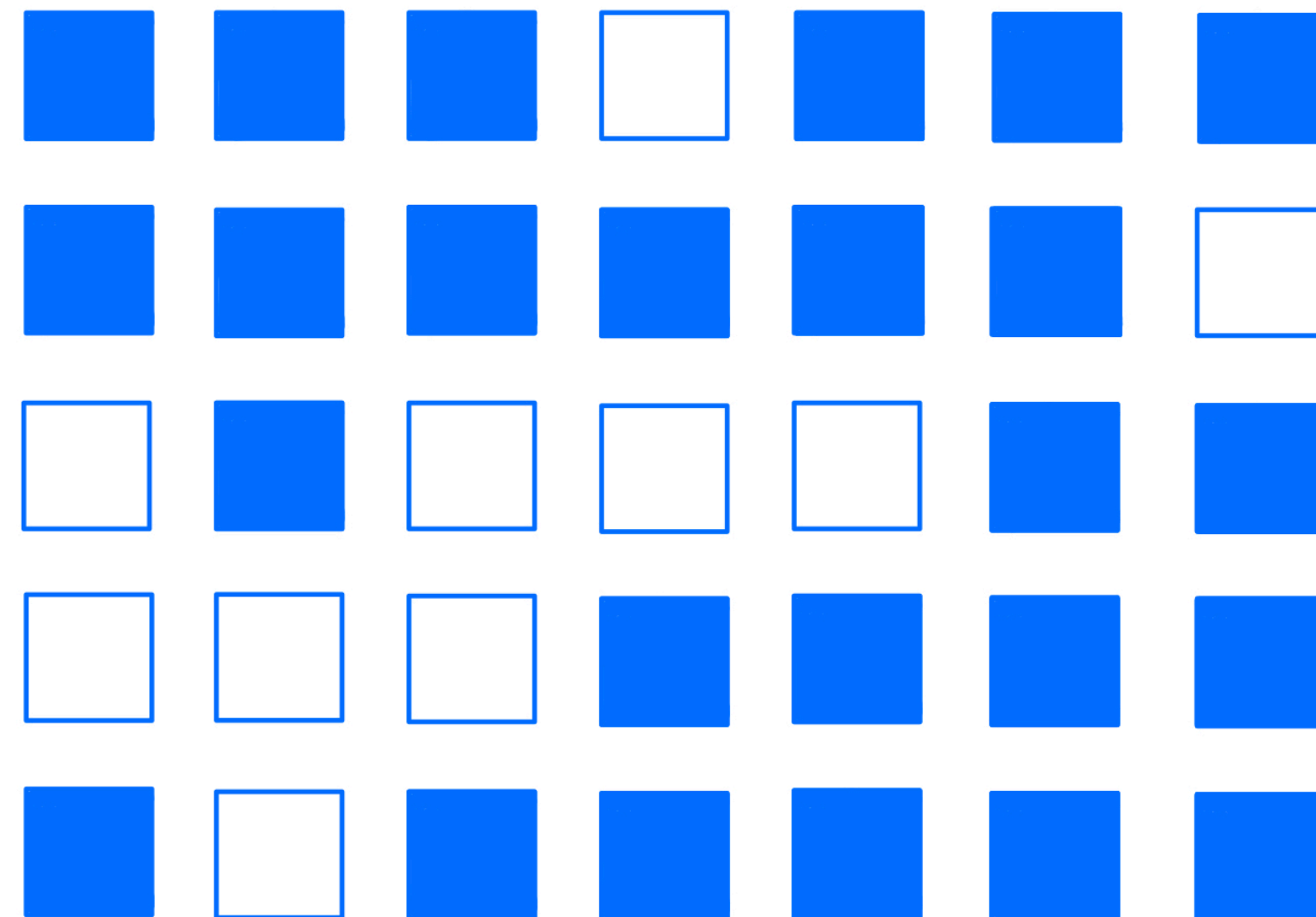
The chip-maze

Replace $(m+1) \times (m+1)$ matrix with $(2E+1) \times m$ where $Z_{i,j}$ is defined as:



The chip-maze

Replace $(m+1) \times (m+1)$ matrix with $(2E+1) \times m$ where $Z_{i,j}$ is defined as:



You can solve
the SNR
problem on this!

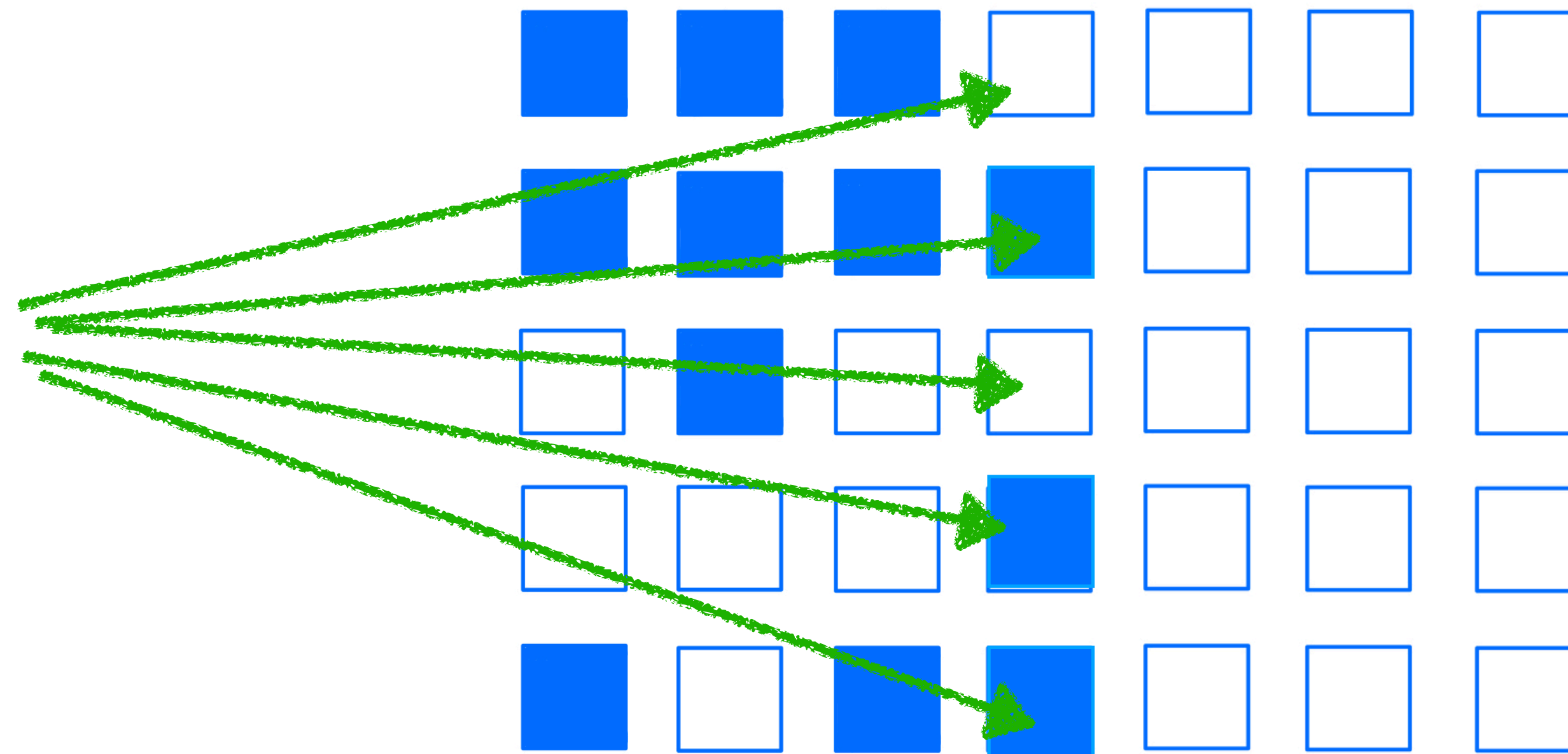
Creating the chip-maze

$$Z[i, j] = \begin{cases} \square & \text{if } i = E + 1, Q[j] = R[j], \\ \square & \text{if } 1 \leq i \leq E, Q[j - i] = R[j], \\ \square & \text{if } i > E + 1, Q[j + i - E - 1] = R[j], \\ \blacksquare & \text{otherwise} \end{cases} \quad (1)$$

ACC**C**GTA

AACCGTA

2. 1. 3. 4. 5.



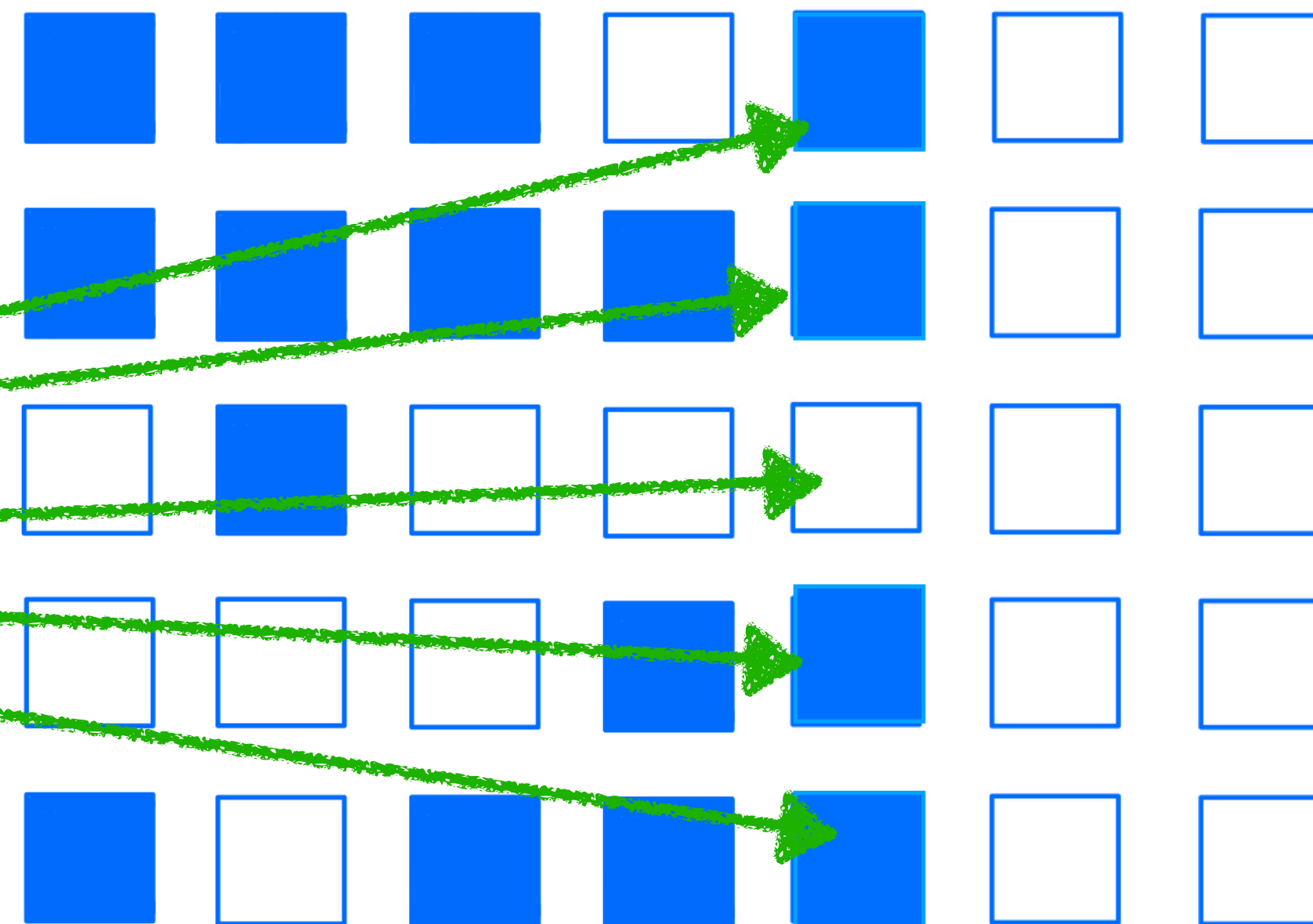
Creating the chip-maze

$$Z[i, j] = \begin{cases} \square & \text{if } i = E + 1, Q[j] = R[j], \\ \square & \text{if } 1 \leq i \leq E, Q[j - i] = R[j], \\ \square & \text{if } i > E + 1, Q[j + i - E - 1] = R[j], \\ \blacksquare & \text{otherwise} \end{cases} \quad (1)$$

ACCCGTA

AACCGTA

2. 1. 3. 4. 5.

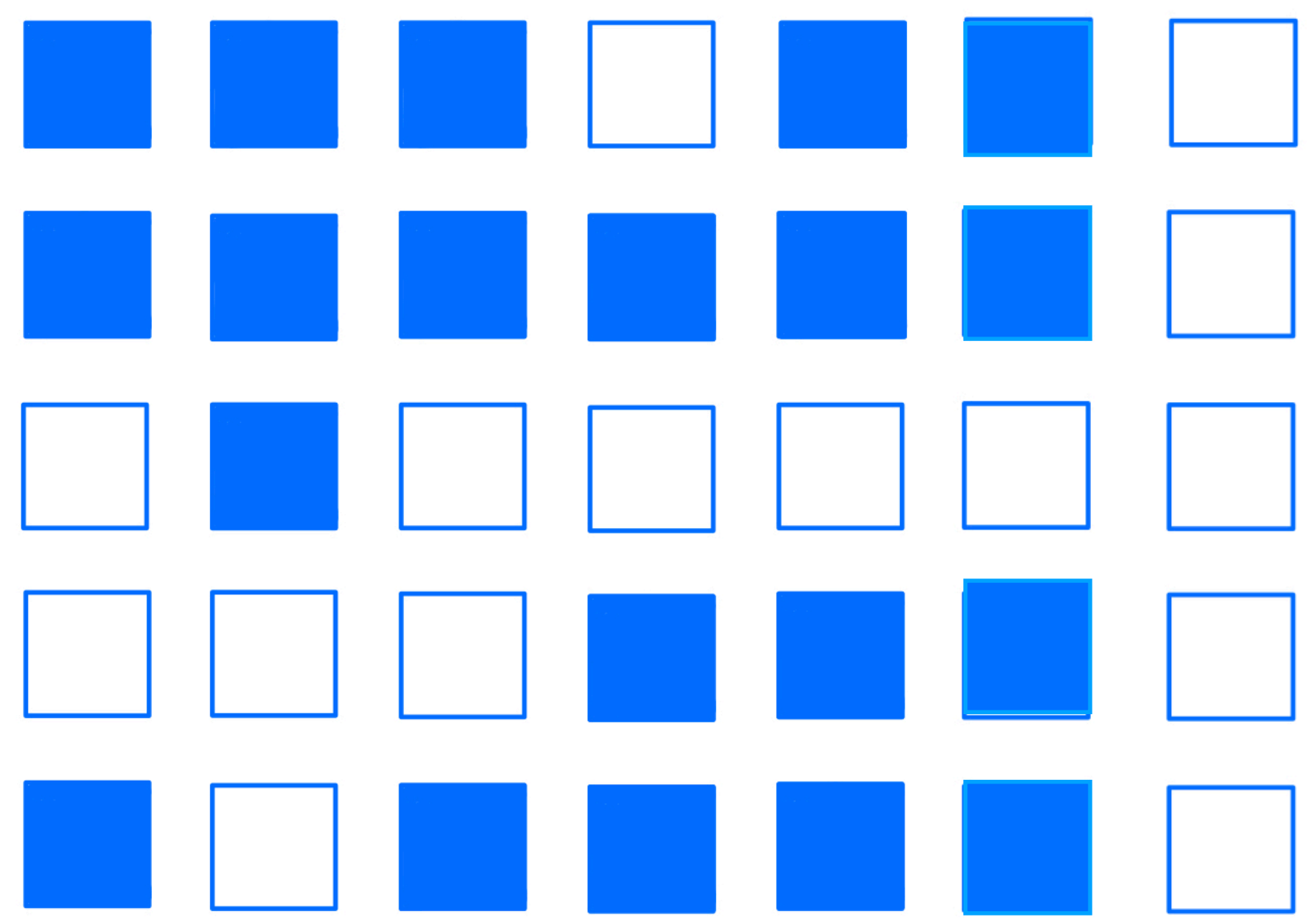


Creating the chip-maze

$$Z[i, j] = \begin{cases} \square & \text{if } i = E + 1, Q[j] = R[j], \\ \square & \text{if } 1 \leq i \leq E, Q[j - i] = R[j], \\ \square & \text{if } i > E + 1, Q[j + i - E - 1] = R[j], \\ \blacksquare & \text{otherwise} \end{cases} \quad (1)$$

ACCCGTA

AACCGTA
2. 1. 3. 4. 5.



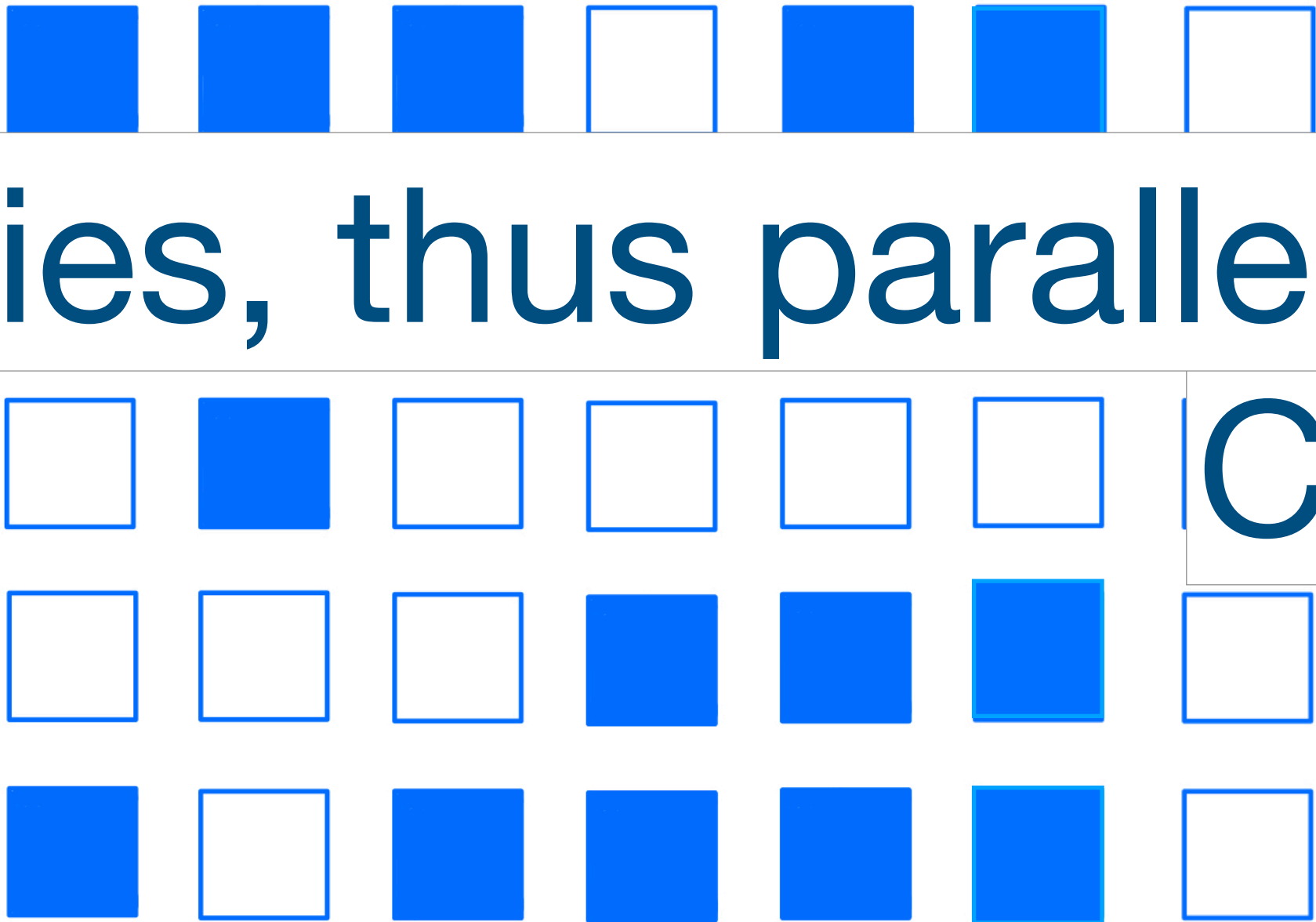
Creating the chip-maze

$$Z[i, j] = \begin{cases} \square & \text{if } i = E + 1, Q[j] = R[j], \\ \square & \text{if } 1 \leq i \leq E, Q[j - i] = R[j], \\ \square & \text{if } i > E + 1, Q[j + i - E - 1] = R[j], \\ \blacksquare & \text{otherwise} \end{cases} \quad (1)$$

No data dependencies, thus parallelisable


AACCGTA

2. 1. 3. 4. 5.



Cont'd

How does SneakySnake work?

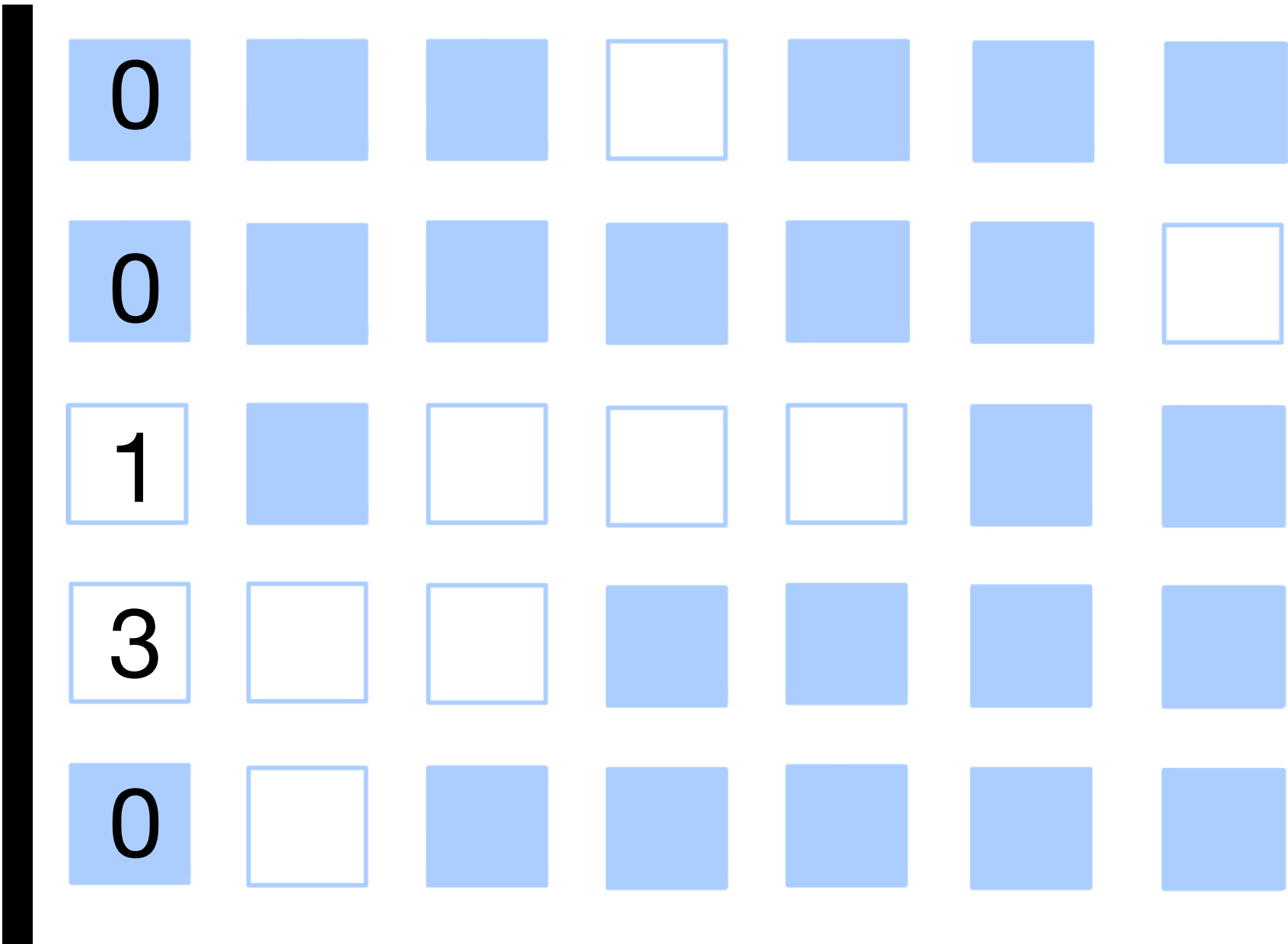
- Reduce ASM problem to SNR 
- Solve the SNR problem
- ???
- Profit!

Solving the Single Net Routing problem

Step 1: Select the **longest** escape segment

Step 2: Create a checkpoint

Step 3: Repeat step 2 and 3 until you reach the **end** or **threshold exceeded**



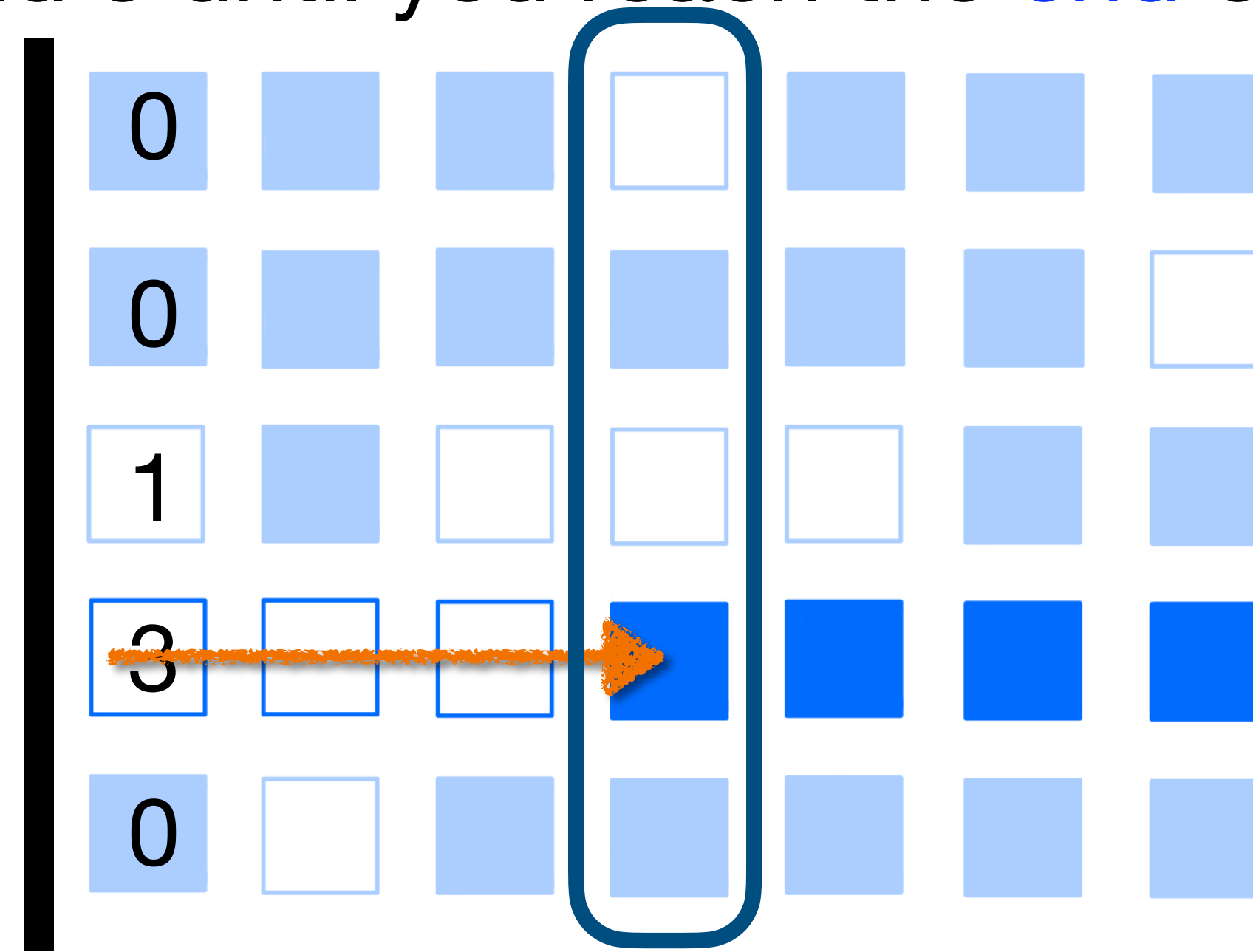
If length $Q \neq R$, deduct leading and trailing obstacles from the count of edits

Solving the Single Net Routing problem

Step 1: Select the **longest** escape segment

Step 2: Create a checkpoint

Step 3: Repeat step 2 and 3 until you reach the **end** or **threshold exceeded**



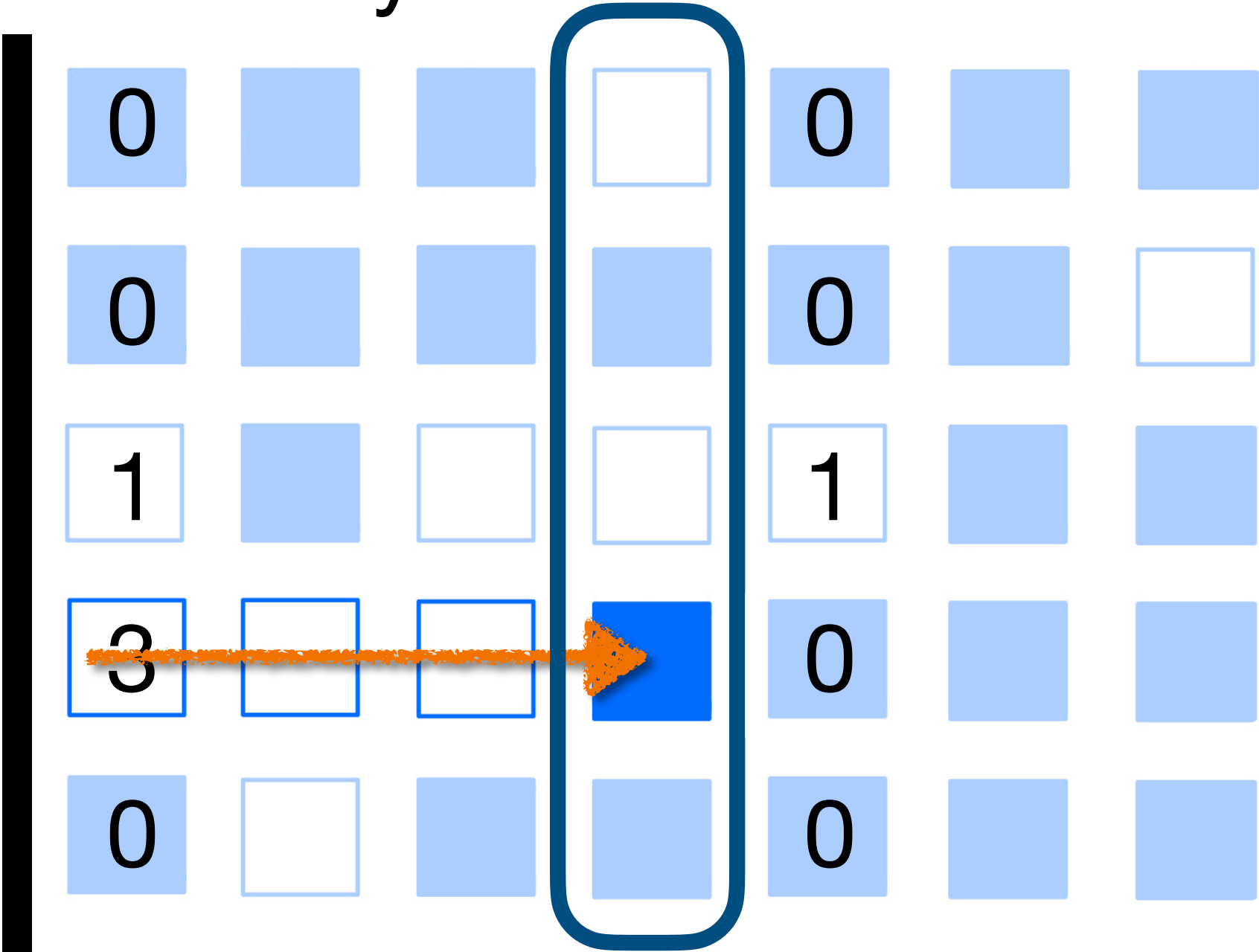
If length $Q \neq R$, deduct leading and trailing obstacles from the count of edits

Solving the Single Net Routing problem

Step 1: Select the **longest** escape segment

Step 2: Create a checkpoint

Step 3: Repeat step 2 and 3 until you reach the **end** or **threshold exceeded**



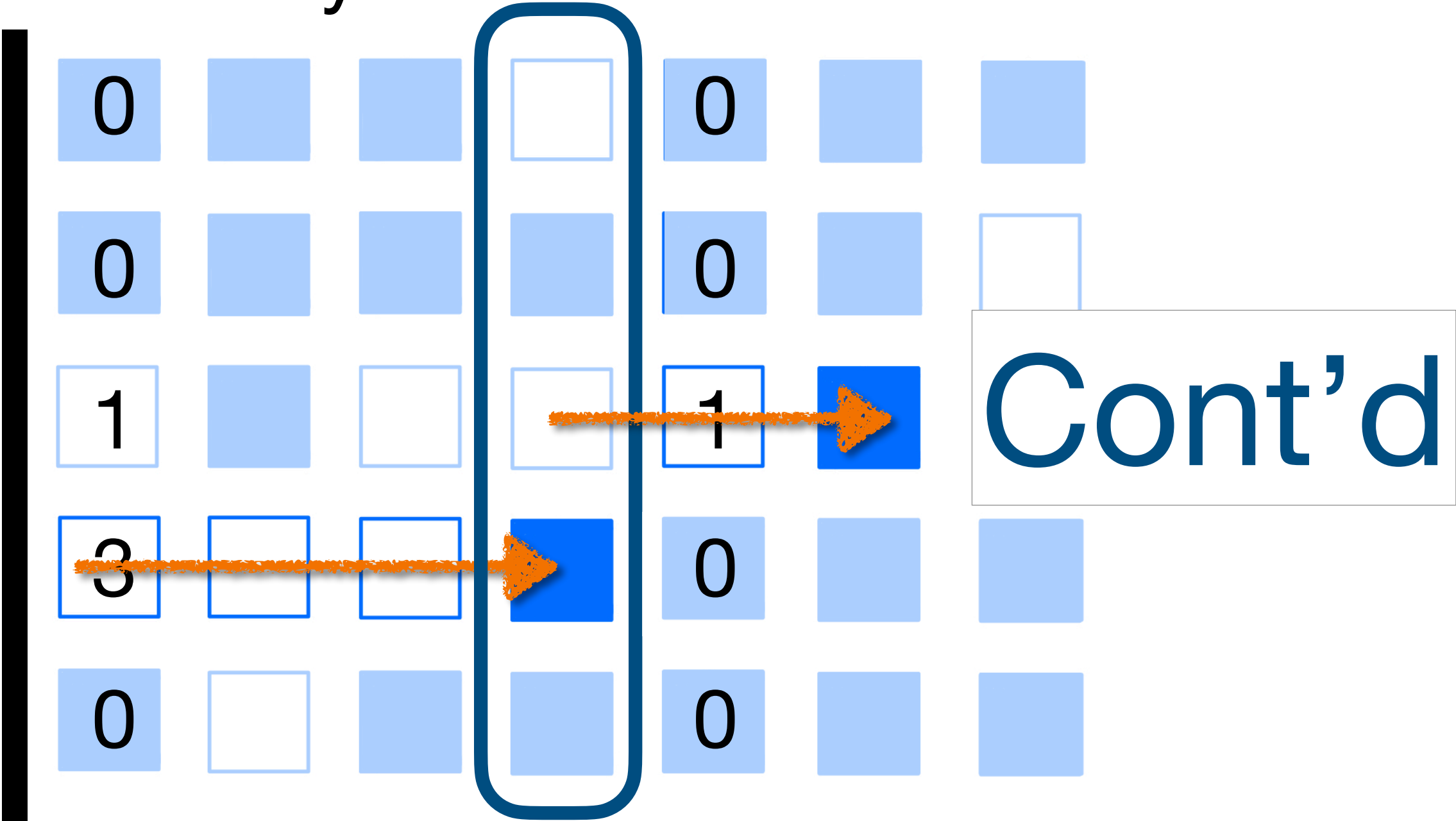
If length $Q \neq R$, deduct leading and trailing obstacles from the count of edits

Solving the Single Net Routing problem

Step 1: Select the **longest** escape segment

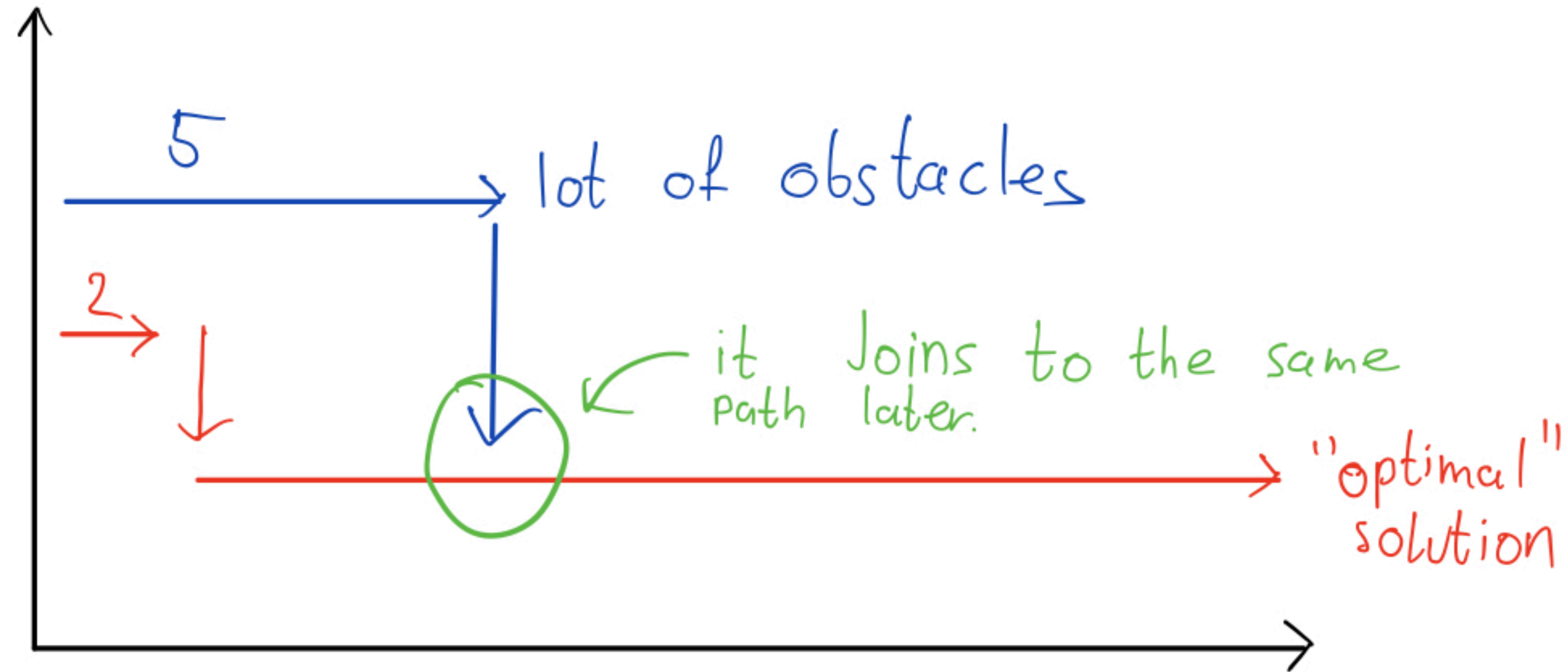
Step 2: Create a checkpoint

Step 3: Repeat step 2 and 3 until you reach the **end** or **threshold exceeded**

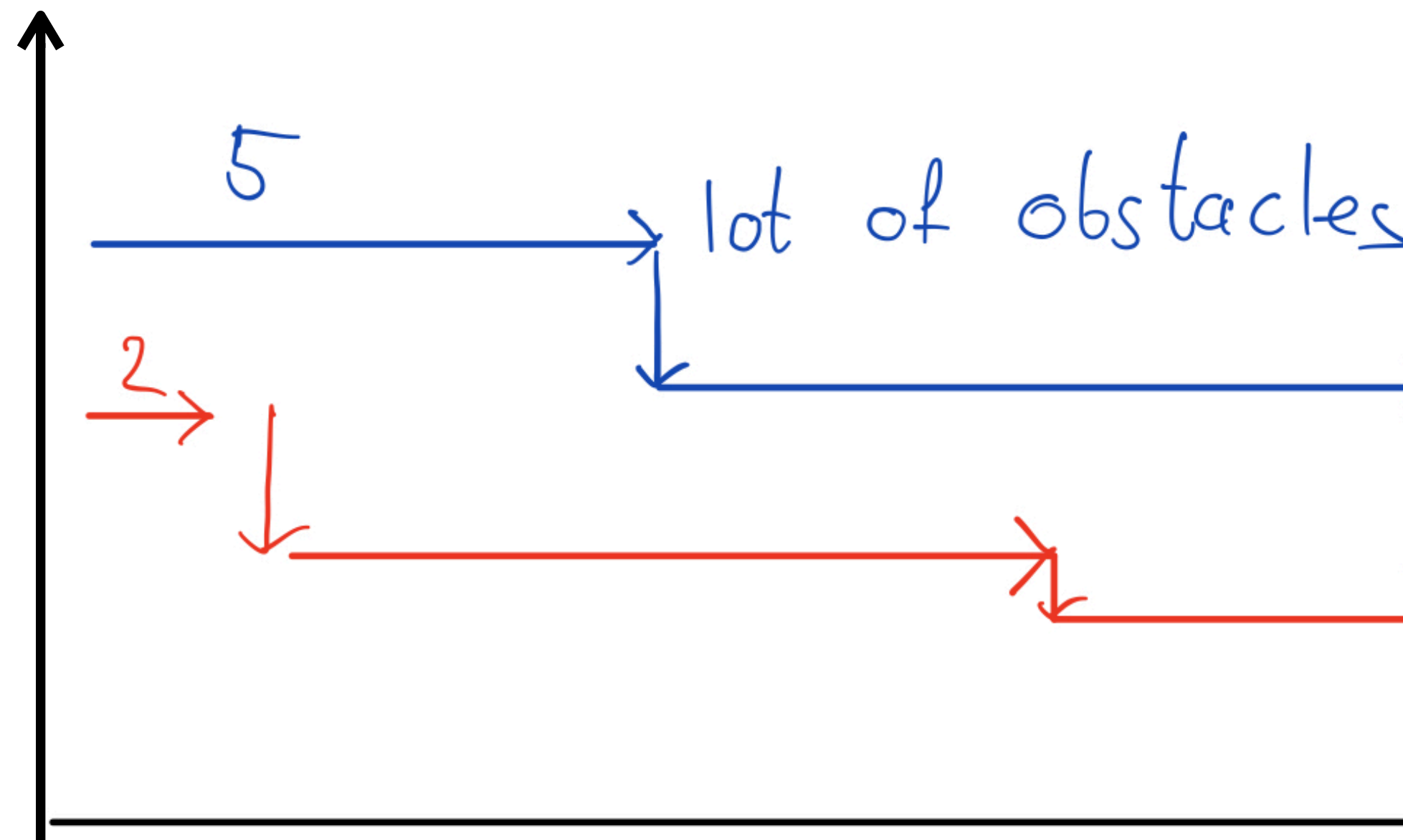


If length $Q \neq R$, deduct leading and trailing obstacles from the count of edits

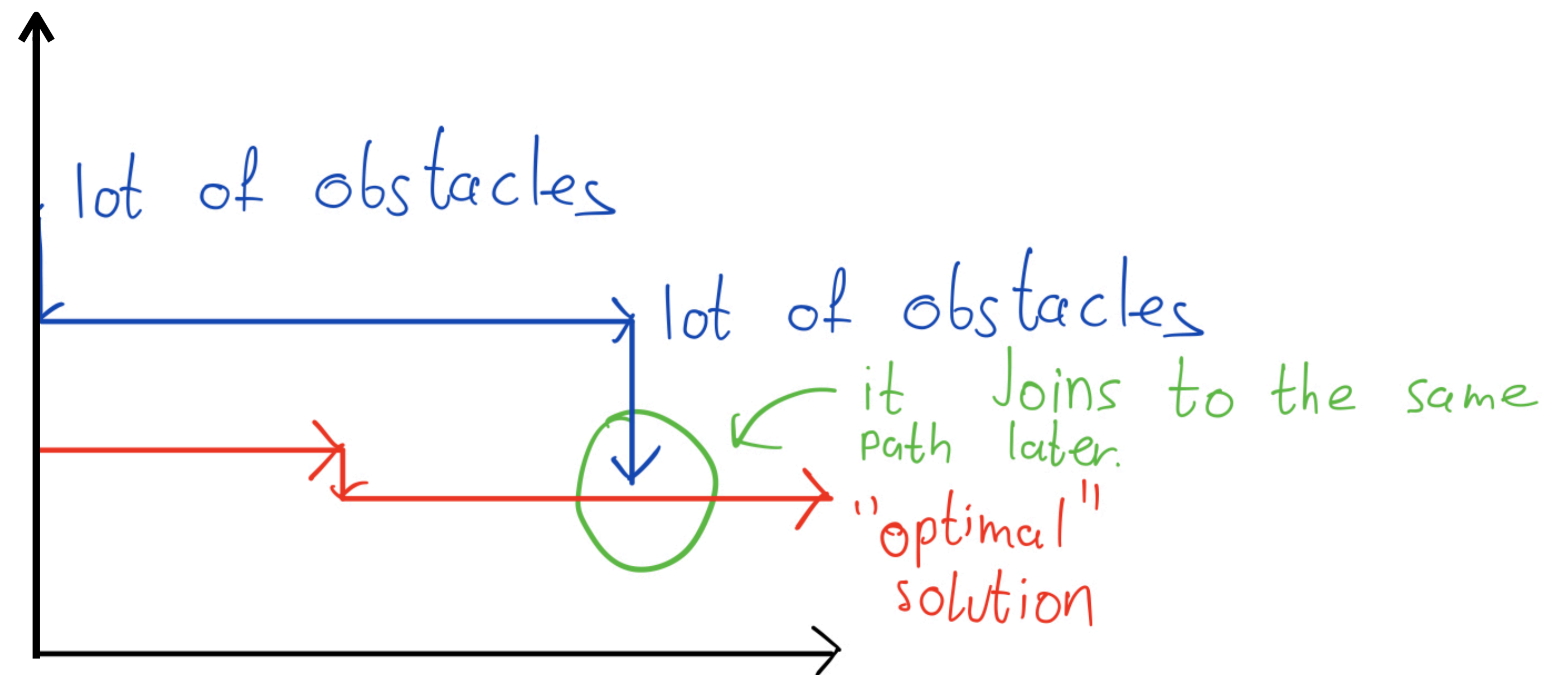
Sketch of optimality proof



Sketch of optimality proof



Sketch of optimality proof





If SneakySnake **doesn't join the optimal solution** at the next option, **we can shift** it to the next checkpoint or we reach the end (thus SneakySnake has less edits)

Solving the Single Net Rounting problem

In conclusion:

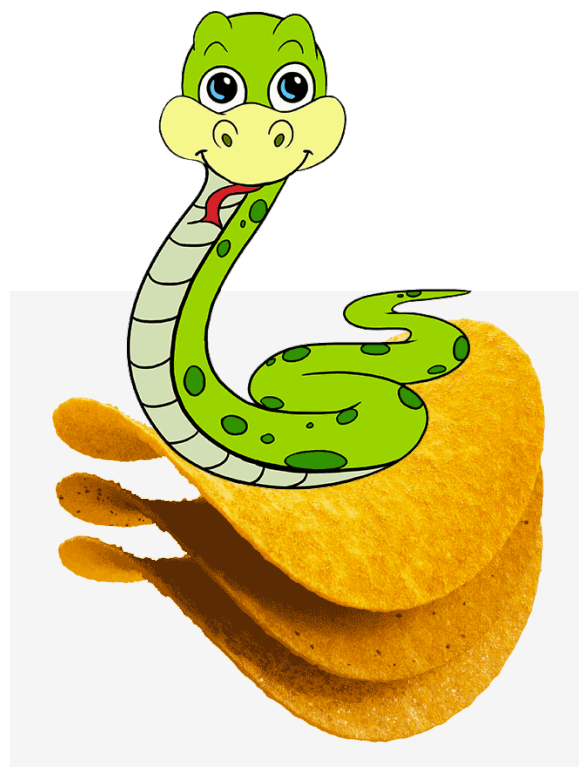
There is one or more Signal nets that connects the In and Out-Terminal \Rightarrow SneakySnake finds the Signal net with the least amount of obstacles possible.

How does SneakySnake work?

- Reduce ASM problem to SNR 
- Solve the SNR problem 
- ???
- Profit!

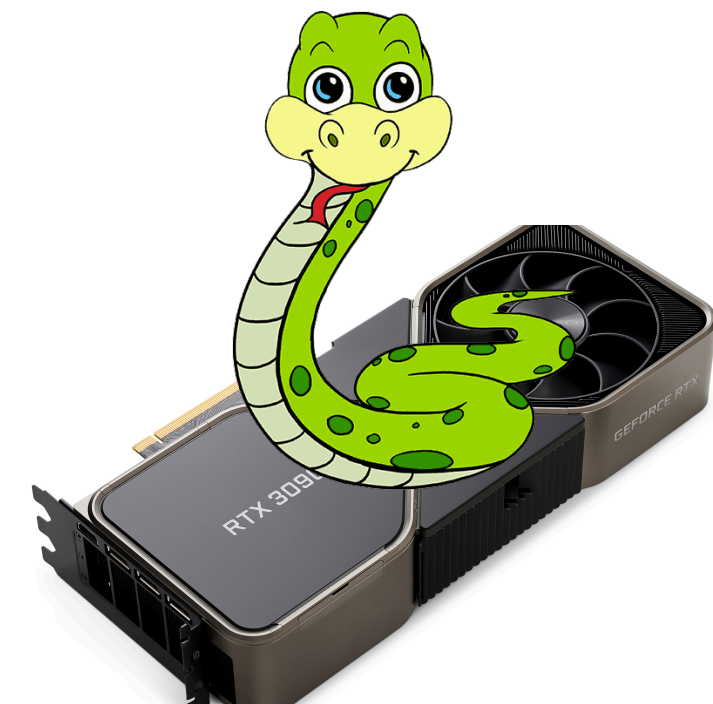
Different versions of SneakySnake

Snake-on-Chip



Exploits the advantages
of an **FPGA-Board**

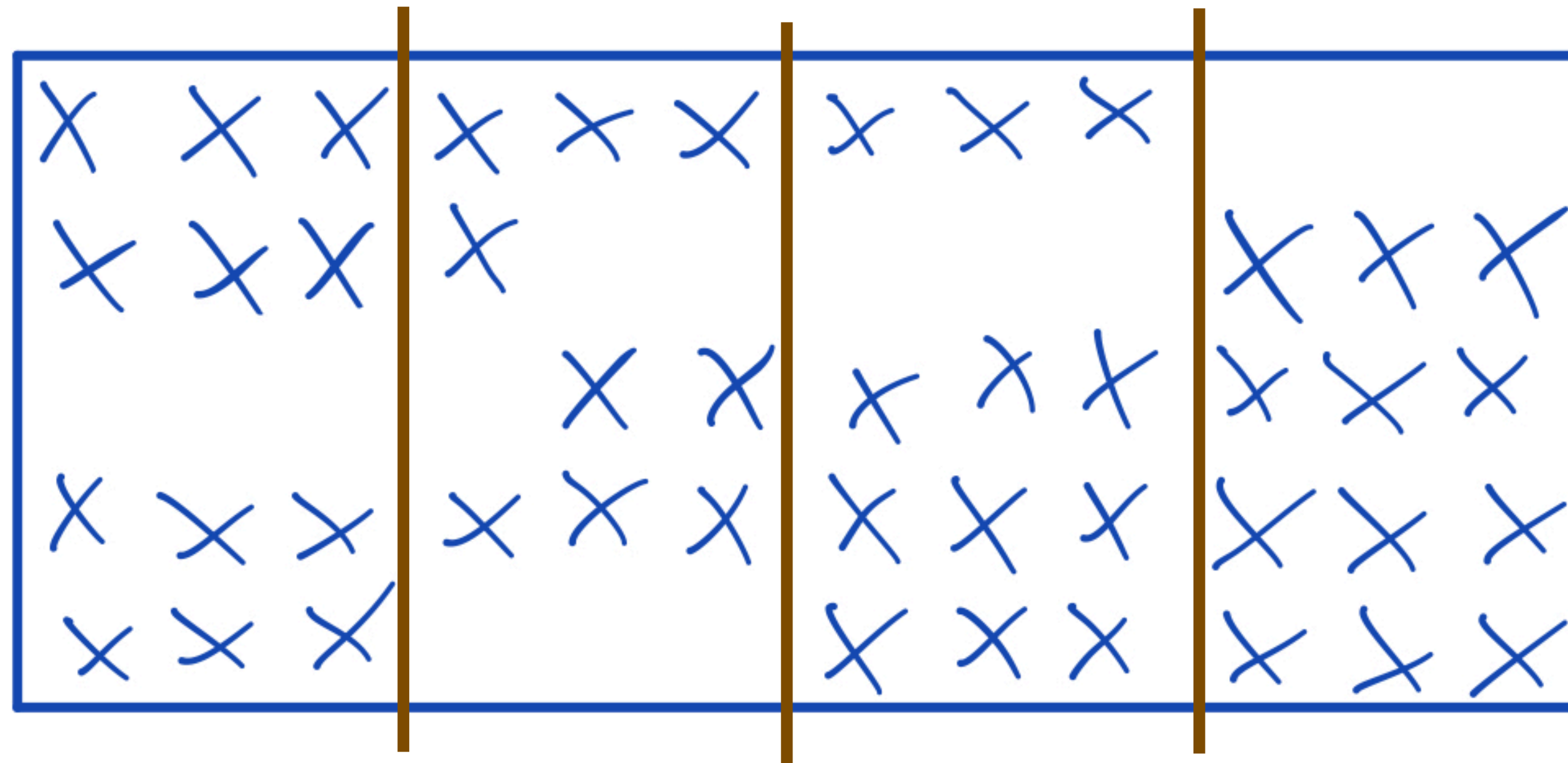
Snake-on-GPU



Exploits the advantages
of a **GPU**

Snake-on-Chip, idea

Solve $\text{multiple } (2E+1) \times t \text{ sized problems}$ instead of solving $\text{one } (2E+1) \times m \text{ SNR problem}$



Snake-on-Chip, benefits

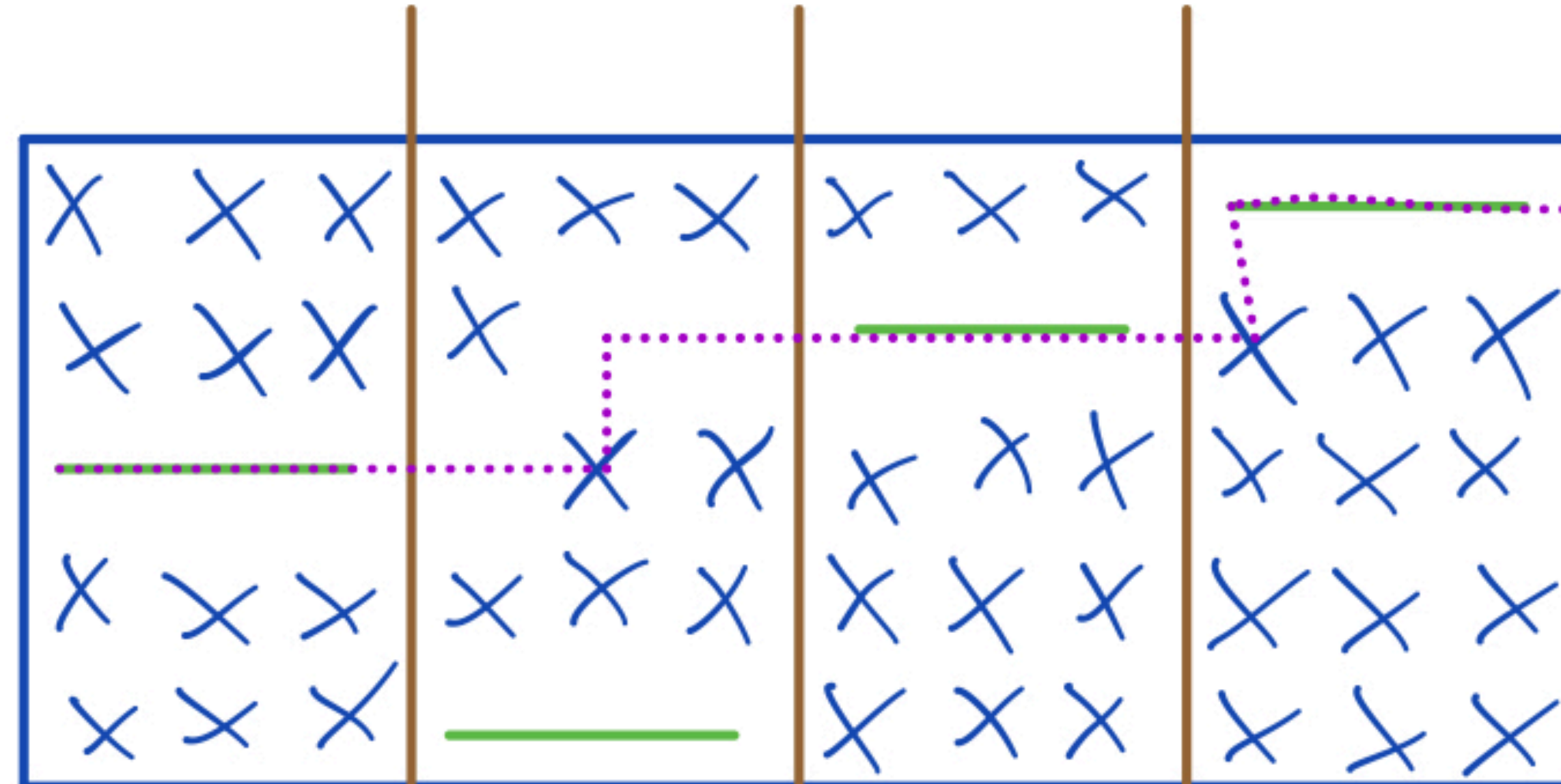
Benefits:

1. Smaller maze -> less amt. of possible solutions -> smaller LUT size
2. Easily scalable
3. Highly parallelisable (no Data dependency at all)!

Snake-on-Chip, problem

The idea: Divide and conquer

Solve multiple $(2E+1) \times t$ sized problems instead of solving one $(2E+1) \times m$ SNR problem



Actual solution: 2

What we get : 0

We can underestimate the optimal solution

➡ Similar strings get marked “more similar” as they are, but it’s ok.

Snake-on-Chip, problem

Benefits:

1. Smaller maze -> less amt. of possible solutions -> smaller LUT size
2. Easily scalable
3. Highly parallelisable (no Data dependency at all)!

But:

Less accurate! (However, it won't mark a similar string as dissimilar)

Snake-on-GPU

The idea: Exploit the amount of GPU threads to **solve multiple** SNR problems **at the same time**

- **Copy** reference and query into the GPU's **global memory**
- Each thread solves a **complete** SNR problem

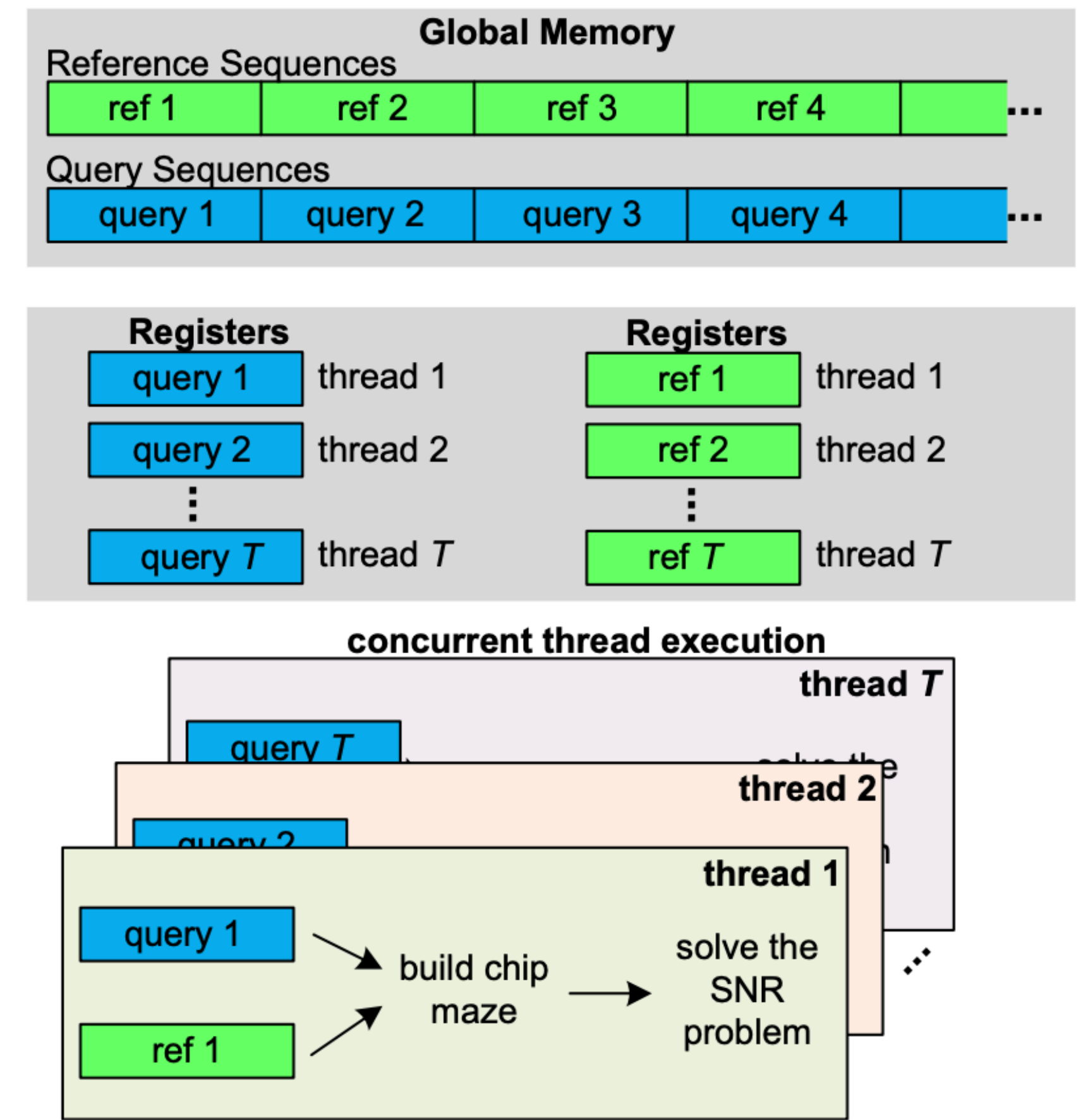


Fig 8

Different versions of SneakySnake

Comparison

Snake-on-Chip

- +Scalable and parallizable
- +More energy efficient than Snake-on-GPU
- More expensive and time consuming
- You can't configure the parameters after design time!!!

Snake-on-GPU

- +Easier to configure
- +Less expensive and time consuming
- +Scalable and parallizable
- Not as energy efficient as Snake-on-Chip

Result

Result

What we mostly care about is:

1. Filtering **accuracy**
2. Filtering **time** (short-and long sequences)
3. Effect on read-mapping

Dataset for accuracy test

How much
edits two similar
strings can have

Accession no.	ERR240727_1		SRR826471_1	
Source	https://www.ebi.ac.uk/ena/data/view/ERR240727		https://www.ebi.ac.uk/ena/data/view/SRR826471	
Sequence Length	100		250	
Sequencing Platform	Illumina HiSeq 2000		Illumina HiSeq 2000	
Dataset	100bp_1	100bp_2	250bp_1	250bp_2
mrFAST <i>e</i>	2	40	8	100
Amount of Edits	Low-edit	High-edit	Low-edit	High-edit

Table 3, Page 20

Dataset for accuracy test

- Each dataset contains 30 million real sequence pairs
- 2 different sequence length, 100 and 250 basepair (prefix 100bp or 250bp)
- We differentiate between low and high edits (suffix _1 or _2)

Filtering accuracy, goal

We want to know how many false accepts and false rejects we have

Goal is: **no false reject** and the less false accept the better.

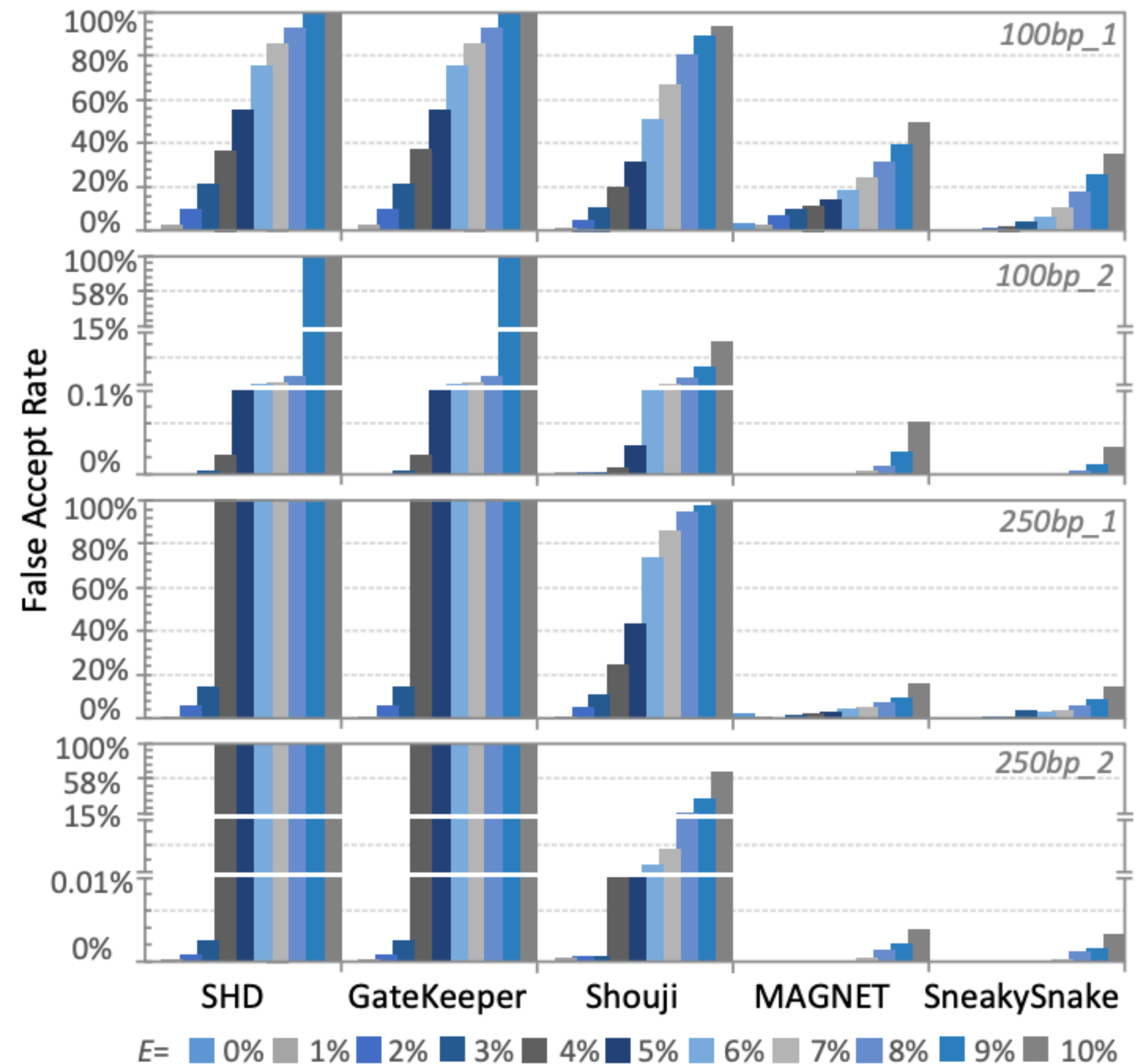
Definitions:

False accept := Two dissimilar string classified as similar by the algorithm

False reject := Two similar string classified as dissimilar by the algorithm

Filtering accuracy, false accepts

- SneakySnake has the **lowest false accept** rate
- All filters are **less accurate when they have less Edits** (i.e. **_1** databases are harder to tell than **_2**)
- SHD and GateKeeper is ineffective for $E \geq 8\%$



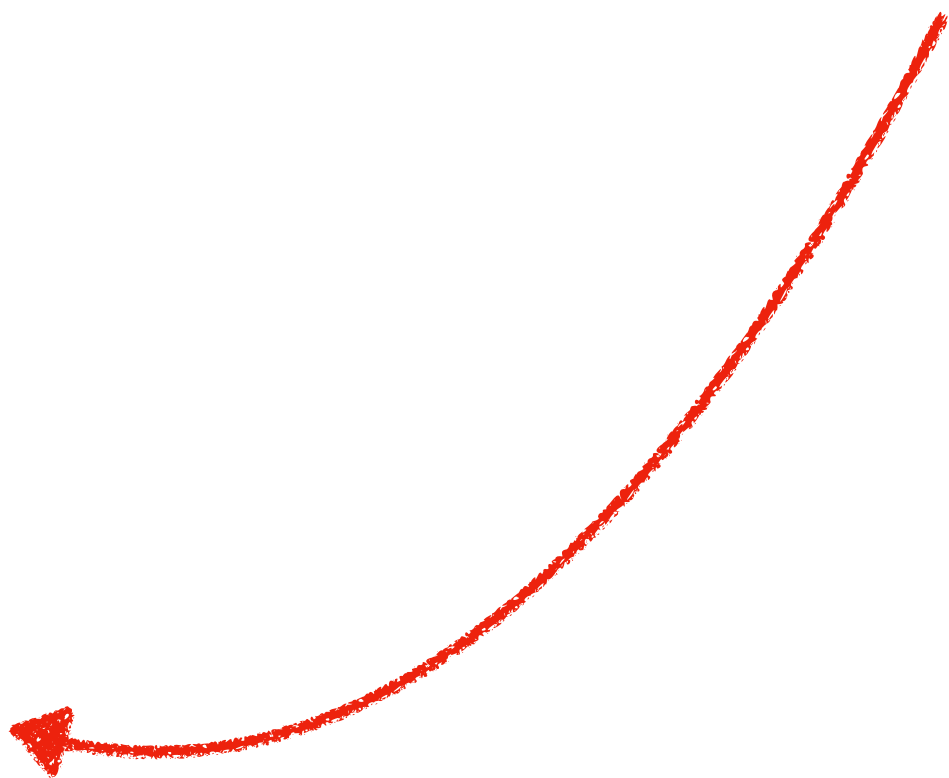
Filtering accuracy, false accepts

Each dataset contains **30 million** real sequence pairs.

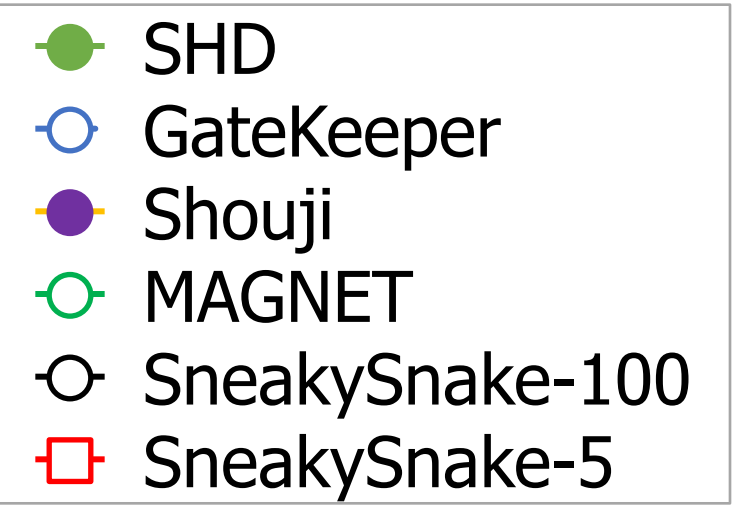
Accession no.	ERR240727_1
Read length (bp)	100
No. of reads	4 million
HTS	Illumina HiSeq 2000

Accession no.	ERR240727_1			
Dataset no.	1	2	3	4
mrFAST -e	2	3	5	40

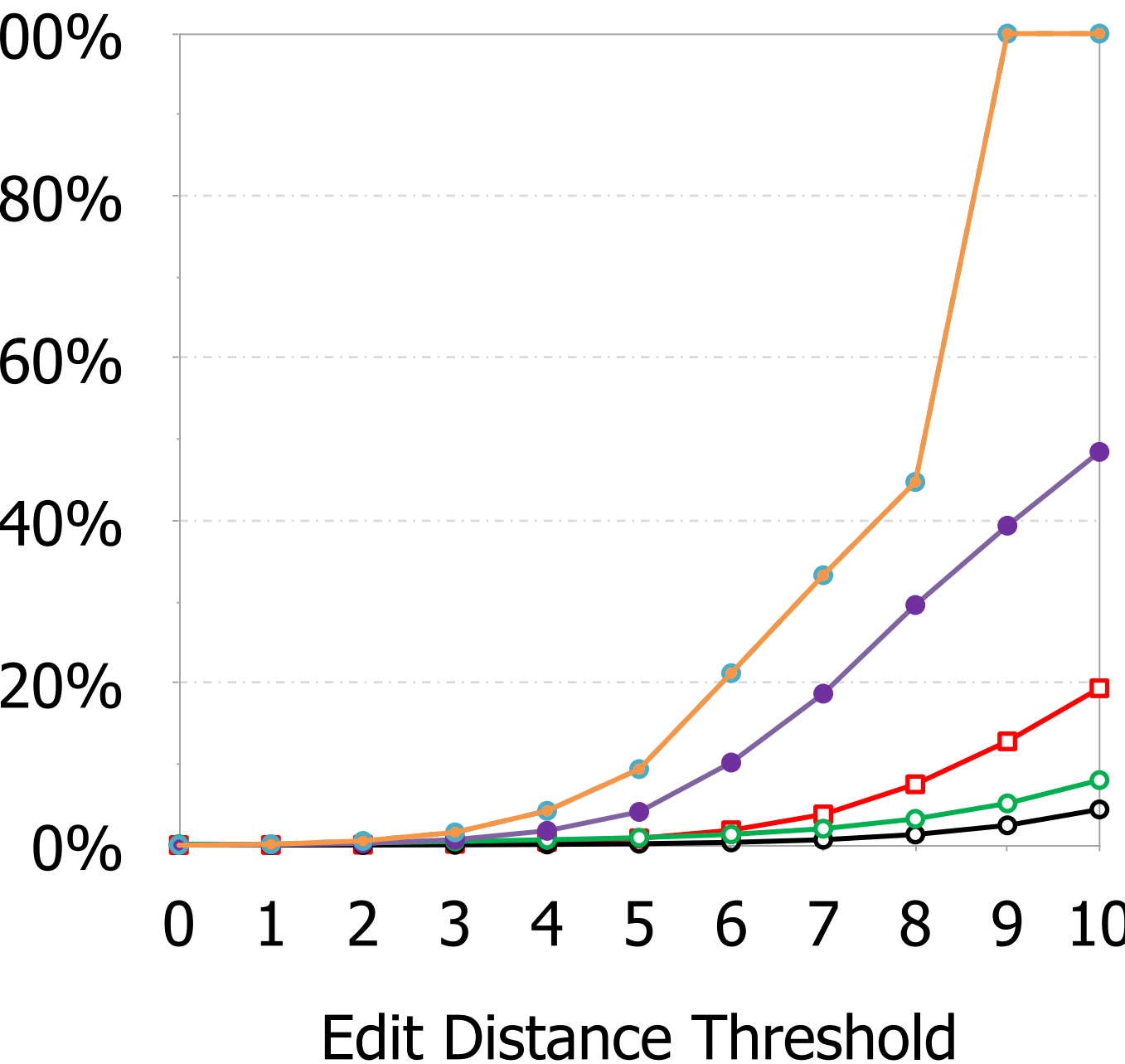
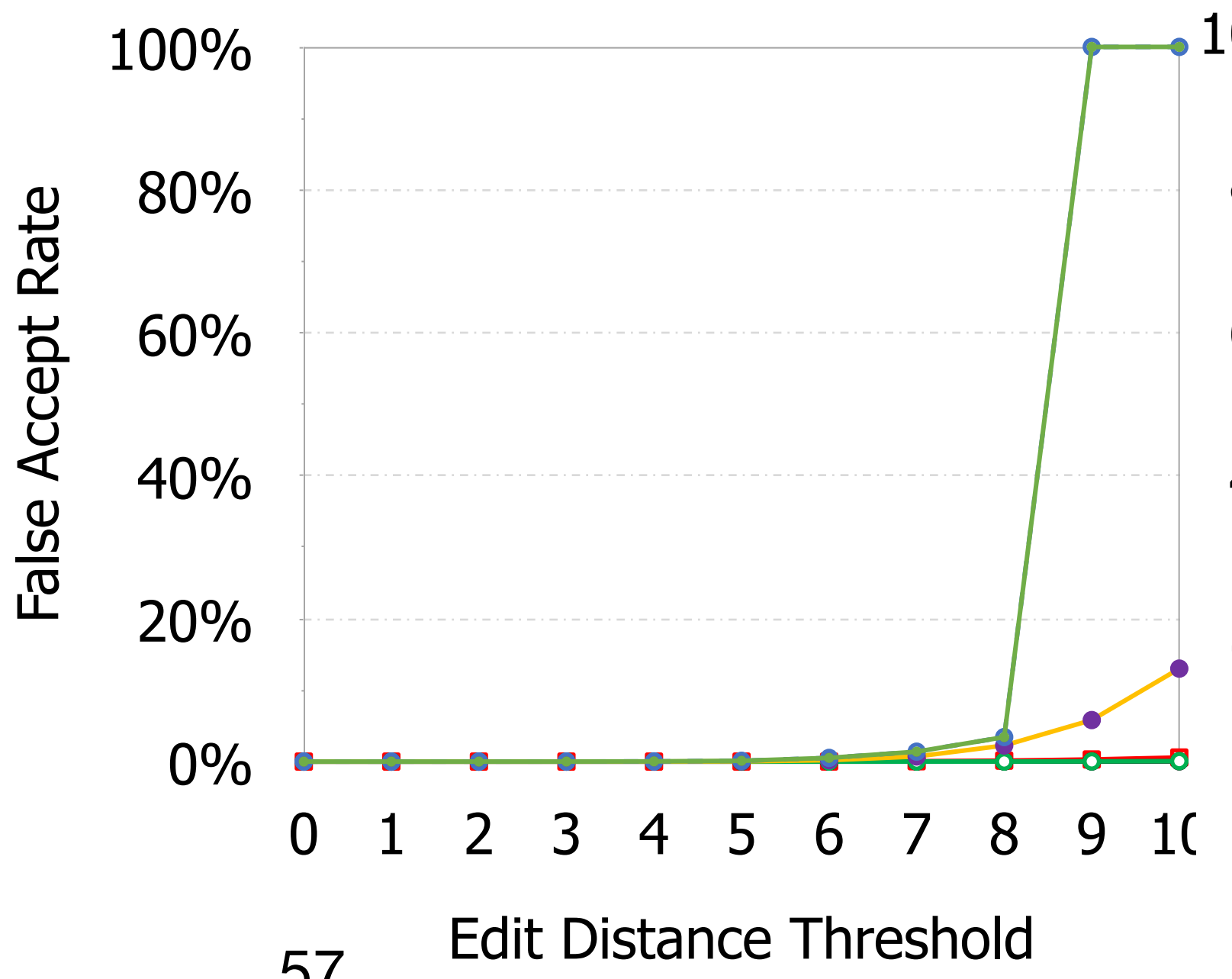
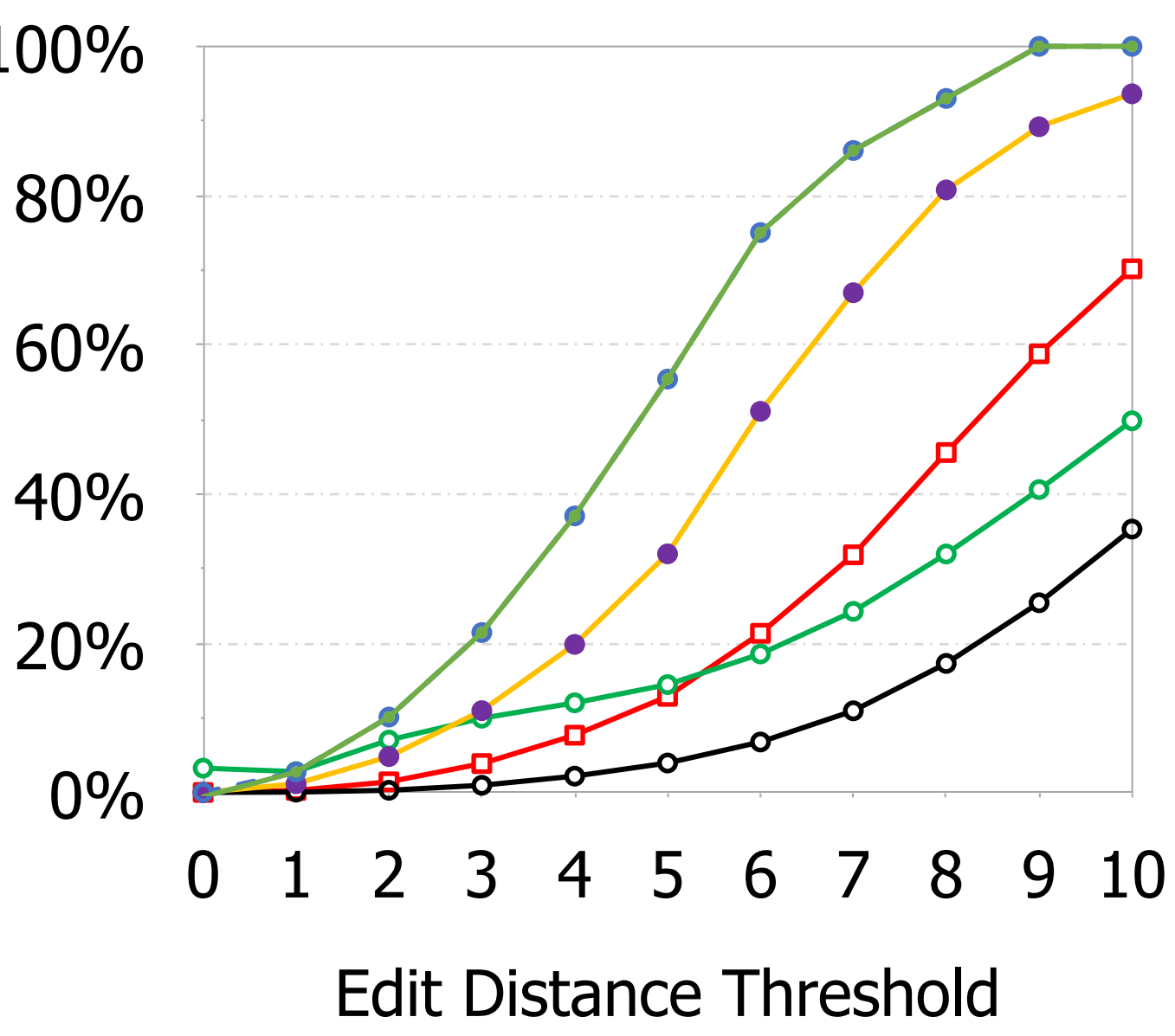
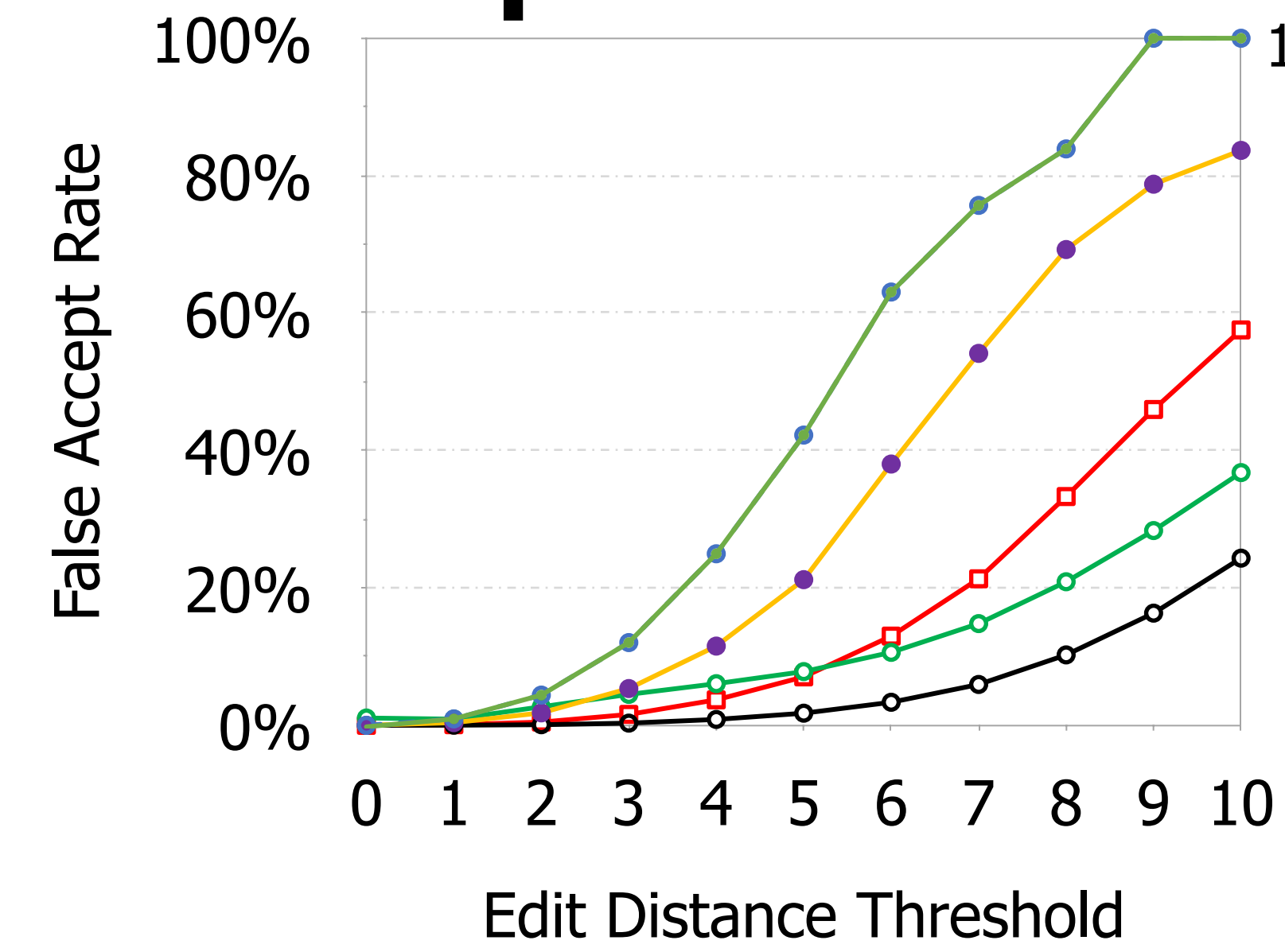
How much edits two similar strings can have



Filtering accuracy, false accepts



- SneakySnake eliminates, on average, up to **412x**, **40x**, and **20x** more incorrect mappings compared to GateKeeper, Shouji and MAGNET.



Filtering accuracy, false rejects

- SneakySnake has 0 false rejects
- This is nothing special, besides MAGNET they all have 0 false rejects

Number of falsely-rejected sequences of SneakySnake, Shouji, MAGNET, SHD, and GateKeeper across 4 real datasets. We use a wide range of edit distance thresholds (0%-10% of the sequence length) for sequence lengths of 100 and 250							
Dataset	E	Truly-Rejected		Falsely-Accepted			
		Edlib	SHD	GateKeeper	Shouji	MAGNET	SneakySnake
100bp_1	0	29'618'099	0	0	0	0	0
	1	28'654'158	0	0	0	0	0
	2	26'733'545	0	0	0	0	0
	3	24'404'404	0	0	0	0	0
	4	22'174'728	0	0	0	1	0
	5	20'178'692	0	0	0	0	0
	6	18'349'510	0	0	0	4	0
	7	16'592'199	0	0	0	19	0
	8	14'847'499	0	0	0	27	0
	9	13'105'320	0	0	0	41	0
	10	11'389'103	0	0	0	31	0
100bp_2	0	29'999'989	0	0	0	0	0
	1	29'999'982	0	0	0	0	0
	2	29'999'976	0	0	0	0	0
	3	29'999'973	0	0	0	0	0
	4	29'999'971	0	0	0	0	0
	5	29'999'966	0	0	0	0	0
	6	29'999'917	0	0	0	0	0
	7	29'999'823	0	0	0	0	0
	8	29'999'667	0	0	0	0	0
	9	29'999'289	0	0	0	0	0
	10	29'998'373	0	0	0	0	0
250bp_1	0	29'292'483	0	0	0	0	0
	2	28'537'758	0	0	0	0	0
	5	28'026'165	0	0	0	0	0
	7	27'638'582	0	0	0	1	0
	10	26'816'729	0	0	0	1	0
	12	26'137'224	0	0	0	9	0
	15	25'084'654	0	0	0	14	0
	17	24'449'131	0	0	0	23	0
	20	23'595'168	0	0	0	35	0
	22	23'040'384	0	0	0	42	0
	25	22'142'250	0	0	0	54	0
250bp_2	0	29'999'951	0	0	0	0	0
	2	29'999'837	0	0	0	0	0
	5	29'999'699	0	0	0	0	0
	7	29'999'625	0	0	0	0	0
	10	29'999'528	0	0	0	0	0
	12	29'999'480	0	0	0	0	0
	15	29'999'425	0	0	0	0	0
	17	29'999'377	0	0	0	0	0
	20	29'999'282	0	0	0	0	0
	22	29'999'158	0	0	0	0	0
	25	29'998'867	0	0	0	0	0

Filtering accuracy, conclusion

- SneakySnake is up to 3,141,100% more accurate than GateKeeper
- Also up to 2,060,200% more accurate than SHD
- And up to 64,000% more accurate than Shouji
- Also, when it comes to false rejects, it is 100% accurate
- ➡ SneakySnake is more accurate than other state-of-art pre alignment filters

Filtering time

Short sequences

- We can have the **same dataset** as in the filtering accuracy tests (100/250_1/2)
- We want to compare pre-alignment filters by using them with the **same sequence aligners**
- We want to **separate the tests** by CPU, GPU and FPGA based algorithms

Long sequences

- We need datasets with larger basepairs
- We want to compare runtime of sequence aligners **with and without** SneakySnake



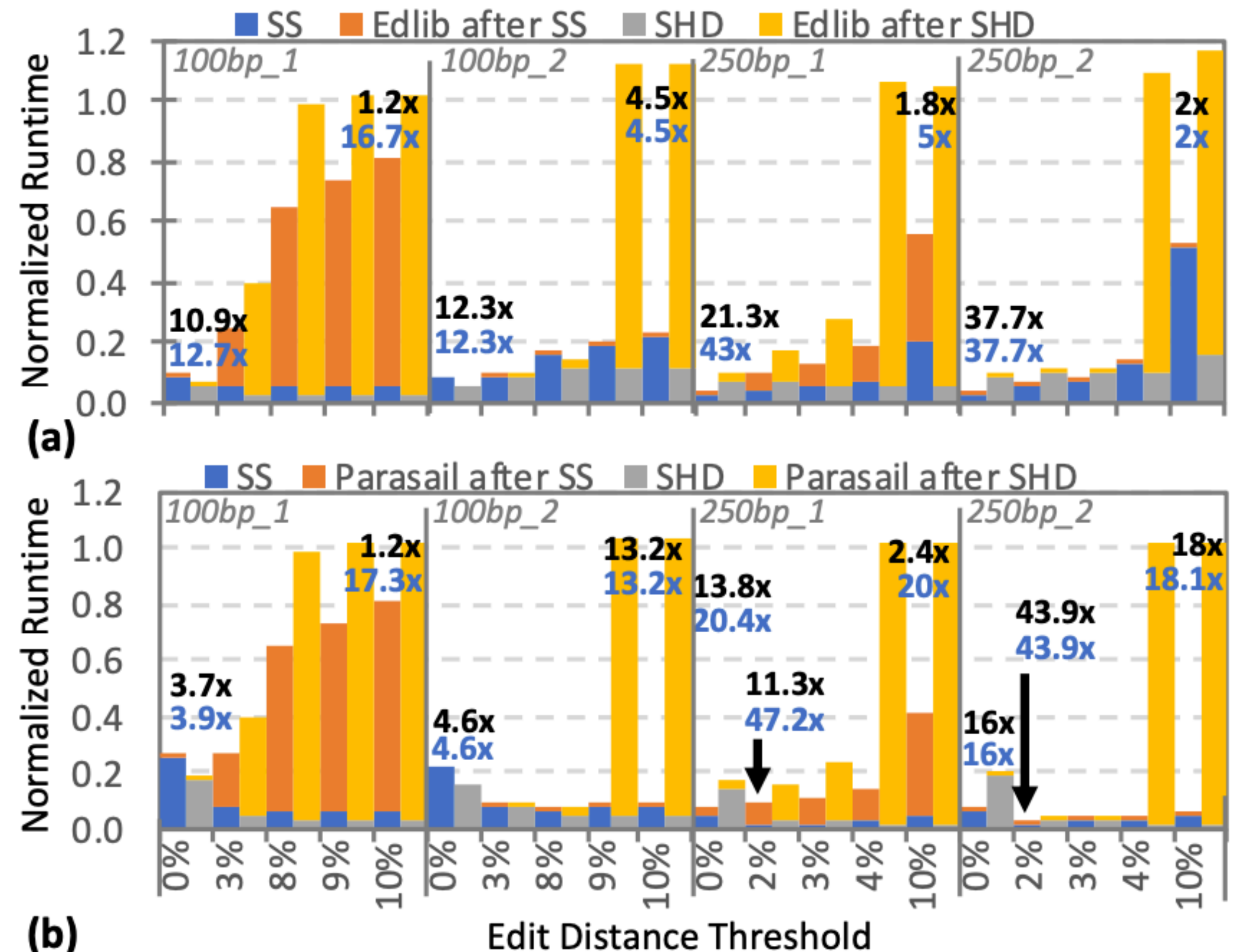
Spoiler: we'll learn that SneakySnake is the fastest pre-alignment filter so we only care about whether it makes sequence alignment faster

Filtering time, short sequences

- Dataset is the same as in Filtering accuracy test
- We use Edlib and Parasail, two state-of-art sequence aligners
- We use SHD (Shifted Hamming Distance) and SneakySnake for the CPU-based comparison
- We compare GateKeeper, Shouji, Snake-on-Chip and Snake-on-GPU for the FPGA/GPU-based speed test

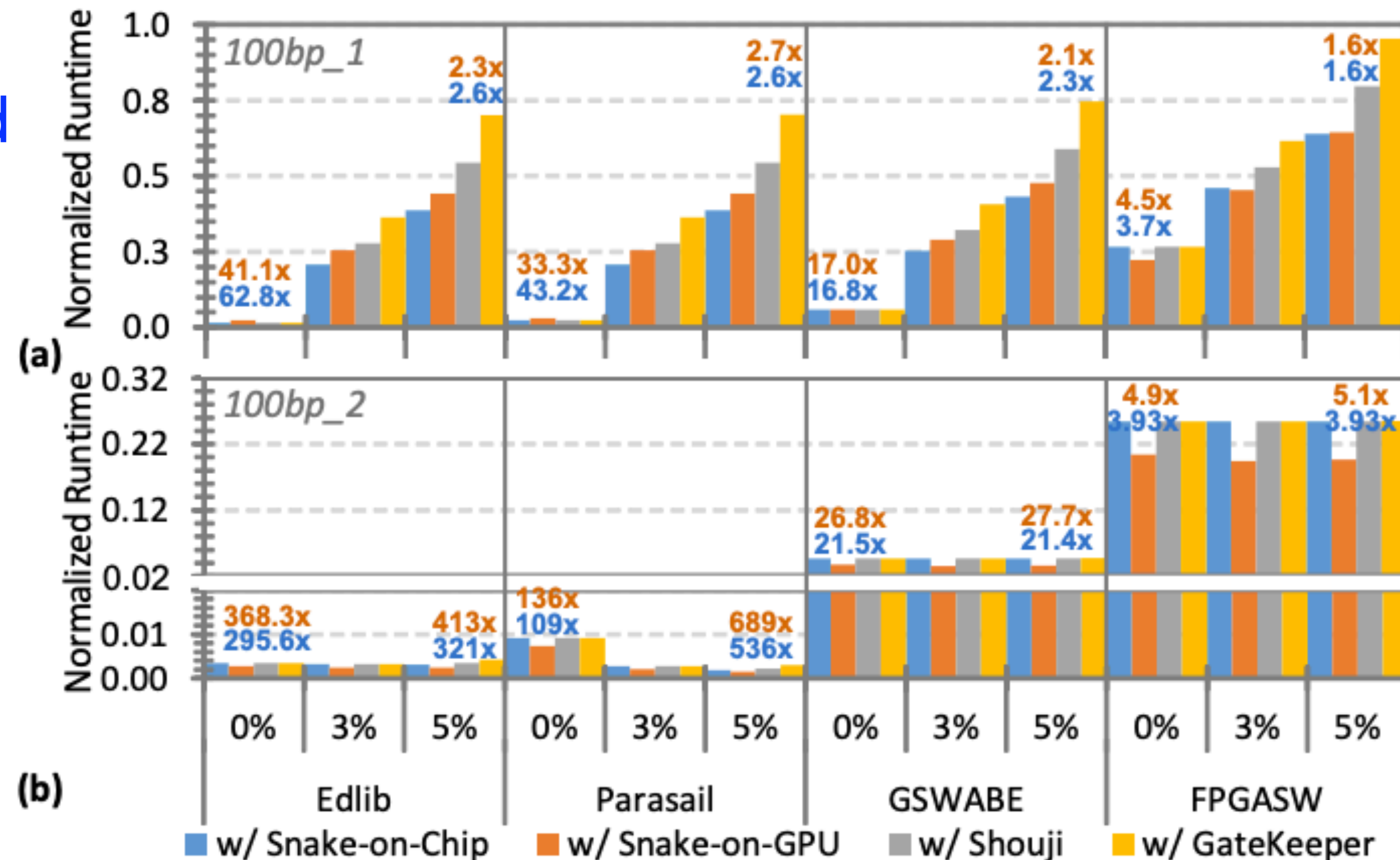
Filtering time, short sequences

- SneakySnake alone is **slower** than Shifted Hamming Distance
- But the **whole process** of sequence alignment gets **faster**
Because the sequence aligner has to compare more dissimilar strings with SHD



Short sequences, GPU/FPGA

- Runtime significantly reduced when using a variant of SneakySnake
- We can tell that Snake-on-Chip and Snake-on-GPU is faster than the CPU version of SneakySnake



Short sequences, conclusion

- SneakySnake is up to 790% - 3,900% faster than other CPU-based pre-alignment filters.
- Runtime of Edlib and Parasail reduced by up to 32,000% and up to 53,500% with Snake-on-Chip and by 41,200% and 68,800% with Snake-on-GPU
- SneakySnake is also up to 100% faster than Shouji and Gatekeeper
- Snake-on-GPU is 3,900% faster than SneakySnake (CPU based)
 - ➡ Snake-on-GPU and Snake-on-Chip is the fastest pre-alignment filter (over Edit-distance of $\leq 5\%$)

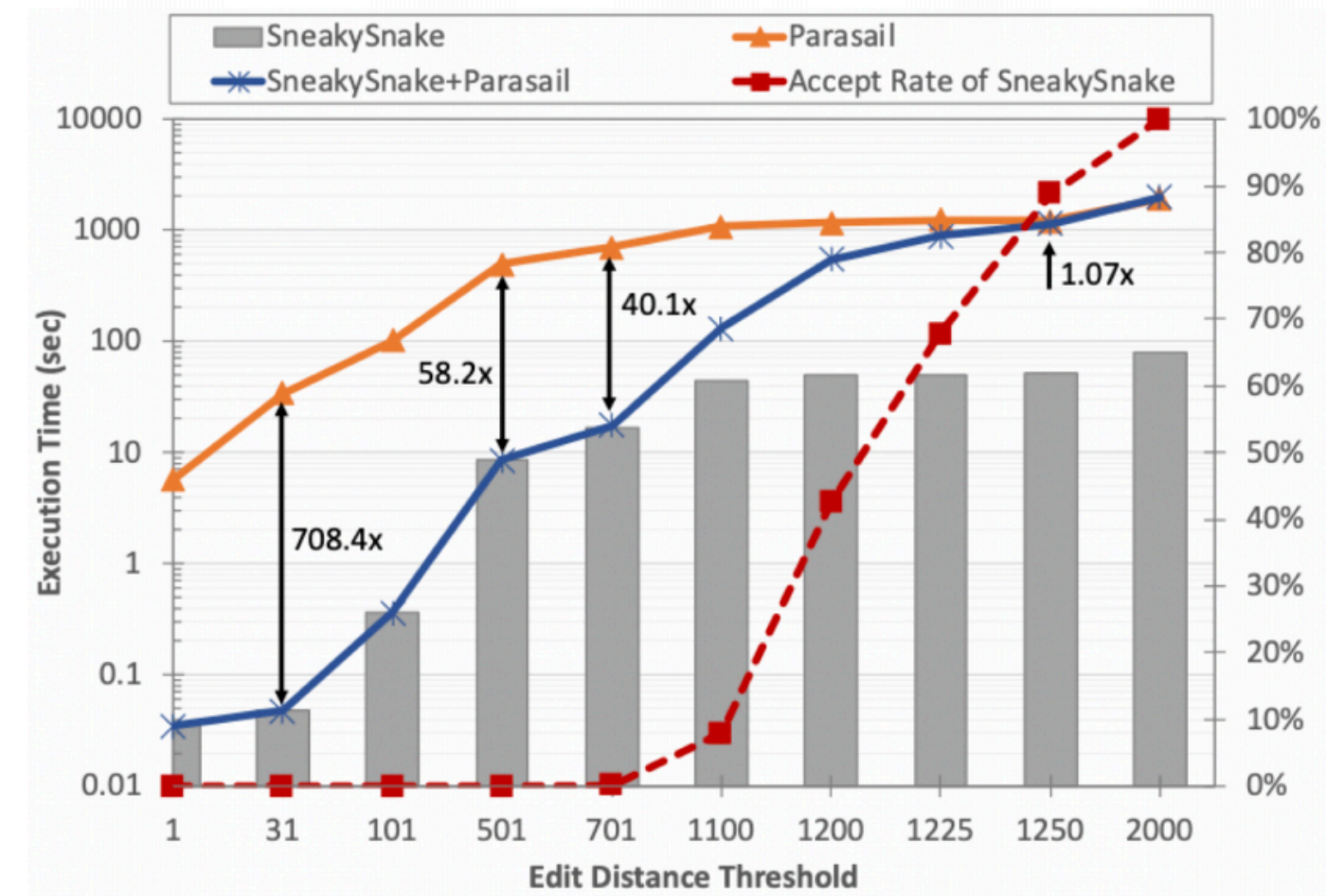
Long sequences, dataset

- Two datasets, (10Kbp, 100Kbp).
 - 10Kbp has 100,000 10,000 long base pair sequence
 - 100Kbp has 74,687 1,000,000 long base pair sequence
- We use [Parasail](#) and [KSW2](#), two state-of-art sequence aligners when it comes to longer sequences
- We look at the runtime of [SneakySnake](#), the sequence aligner and the two combined

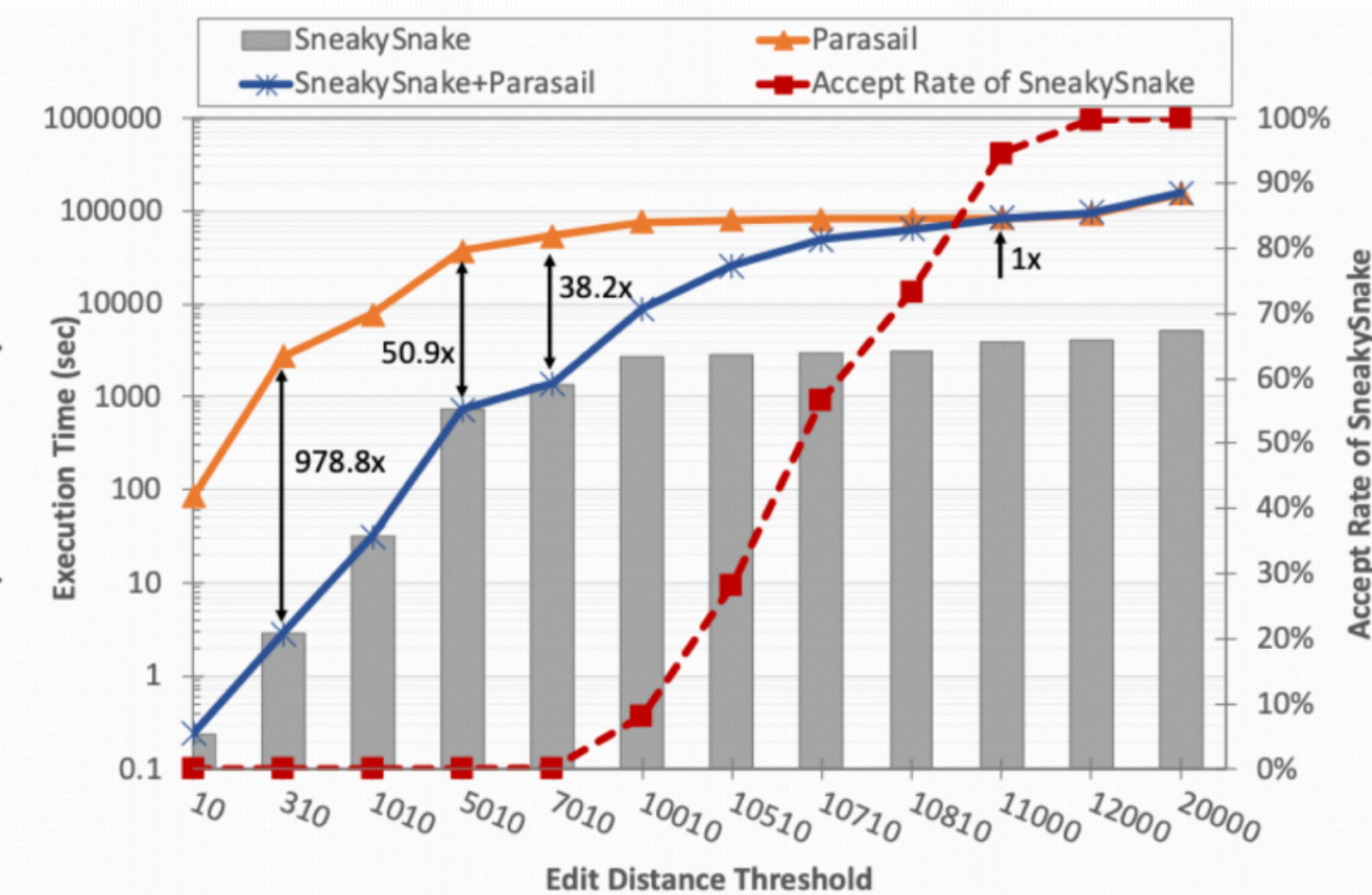
Filtering time, long sequences

- SneakySnake is faster than the sequence aligner alone, when the edit distance threshold is low
 - But it gets less and less significant as we increase the edit distance threshold
- Because the more strings are marked as similar the less of a help the pre-alignment filter is.

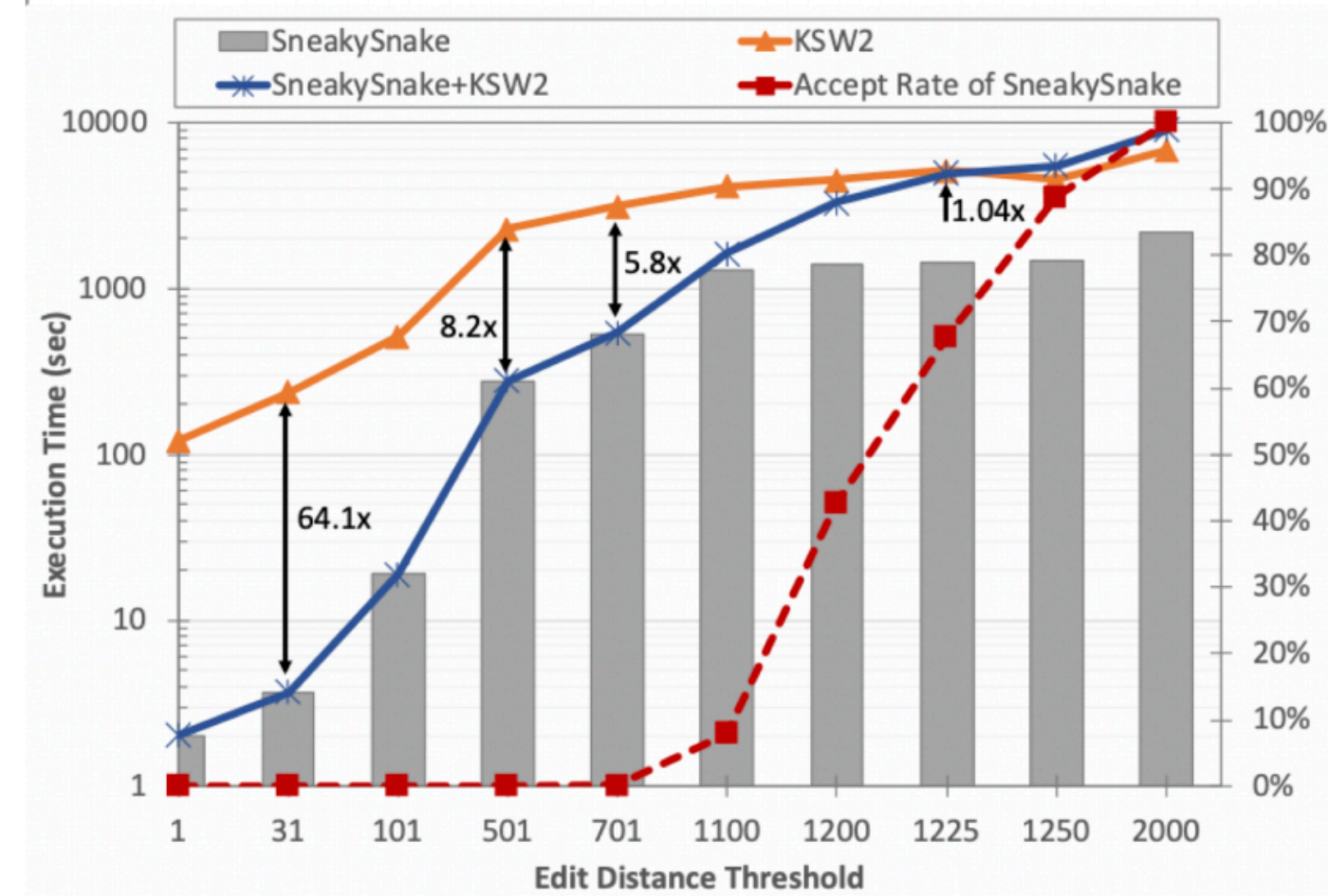
10K bp dataset



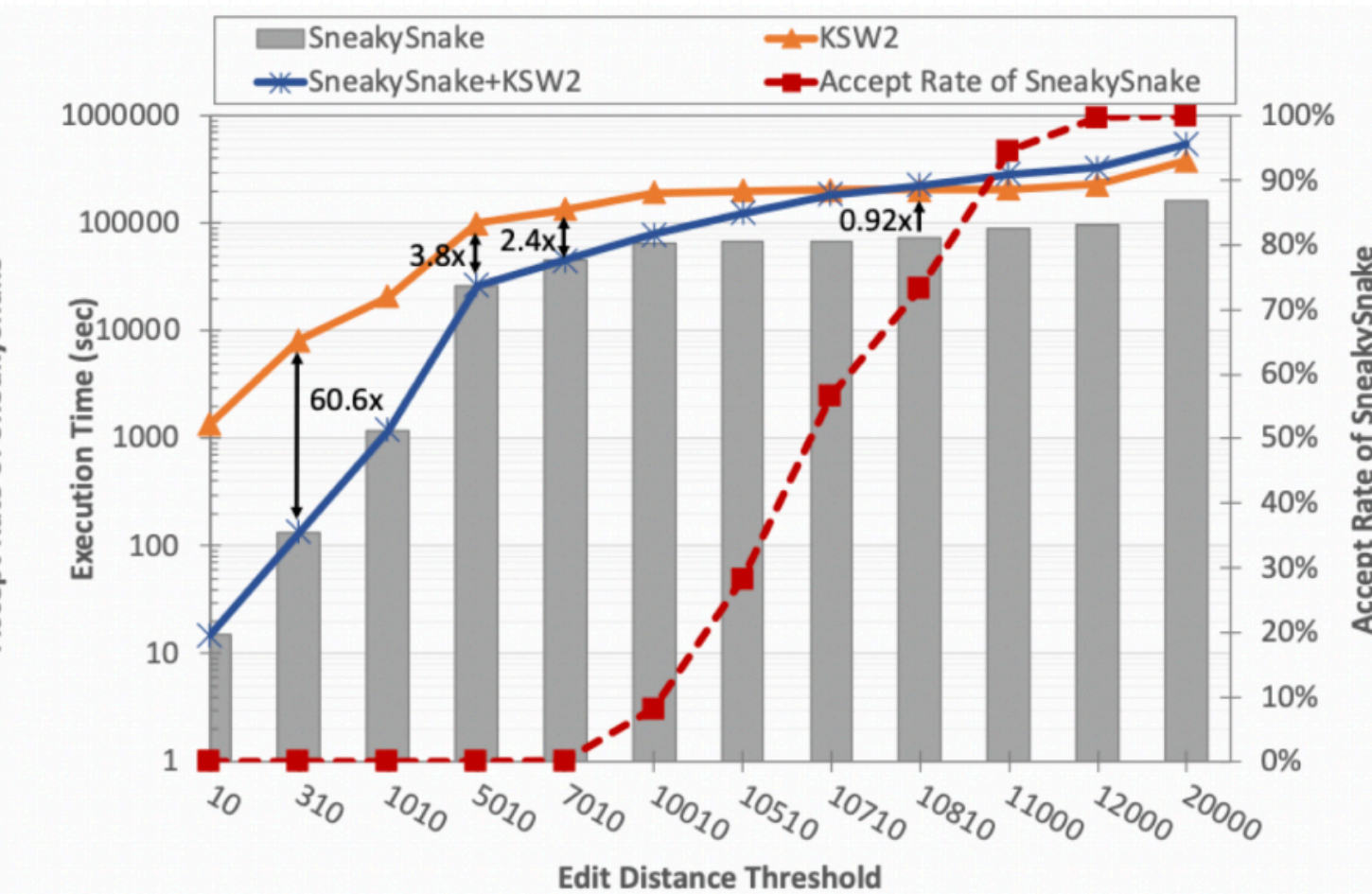
100K bp dataset



10K bp dataset



100K bp dataset



Long sequences, conclusion

- SneakySnake accelerates Parasail and KSW2 by 50,800-97,800% and 280-9,070%
- But it is only helping when SneakySnake filters out more than ~30% of the sequences
 - ➡ In most cases, it is beneficial to use SneakySnake as pre-alignment filter before applying the sequence aligner

Effect on read mapping

We want to **integrate SneakySnake** into a state-of-art read mapper

We use minimap2 as read mapper as it includes methods to speed up read mapping and it is parallelized.

- SneakySnake + minimap2's aligner is at least **6x faster than minimap2's approach**
- The **mapping time is reduced** from 418 seconds to 206
 ➡ We **reduced the speed of read mapping** with SneakySnake

Strengths

Strengths

- It's possible to sequence multiple samples parallel at the same time as it is **highly parallelisable**
- **Superior** to other approaches (and even to the state-of-art techniques) when it comes to **speed and accuracy**
- Available on CPU, GPU and FPGA thus **compatible with most sequence aligner**
- **First** pre-aligner that is **both software** designed and co-**hardware** designed
- Snake-on-Chip and Snake-on-GPU **exploit their architecture** in a very **effective** way, **without being dependent** from a given model or platform

Strengths (cont'd)

- We can **accelerate state-of-art** genome sequencer with **integrating SneakySnake** into state-of-art read mappers
- Very **simple**, easy-to-understand **solution**

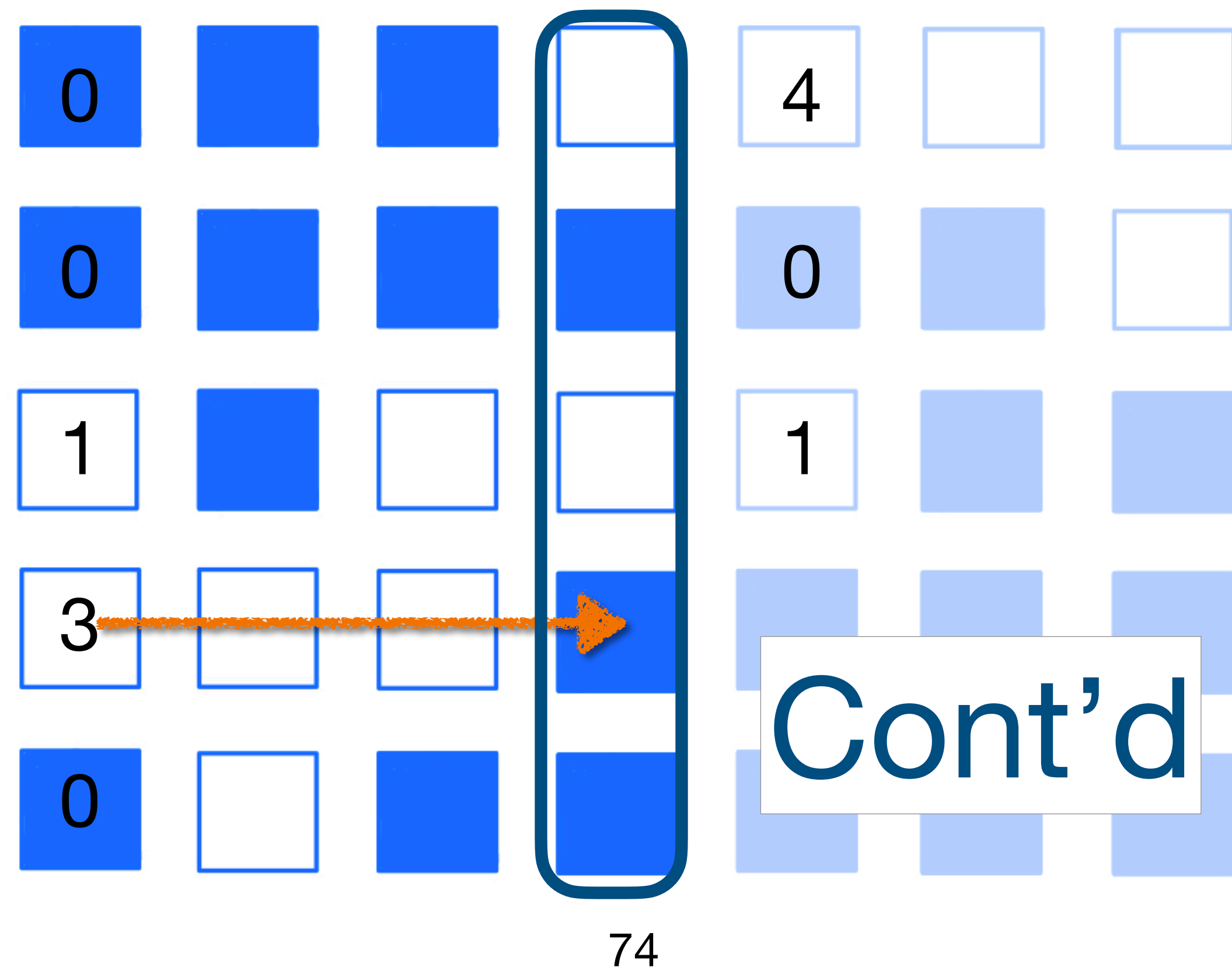
Weaknesses

Weaknesses

- If the higher HRT found an escape segment to the end, SneakySnake will still iterate through the lower HRTs

Weaknesses

- If the higher HRT found an escape segment to the end, **SneakySnake** will still iterate through the lower HRTs



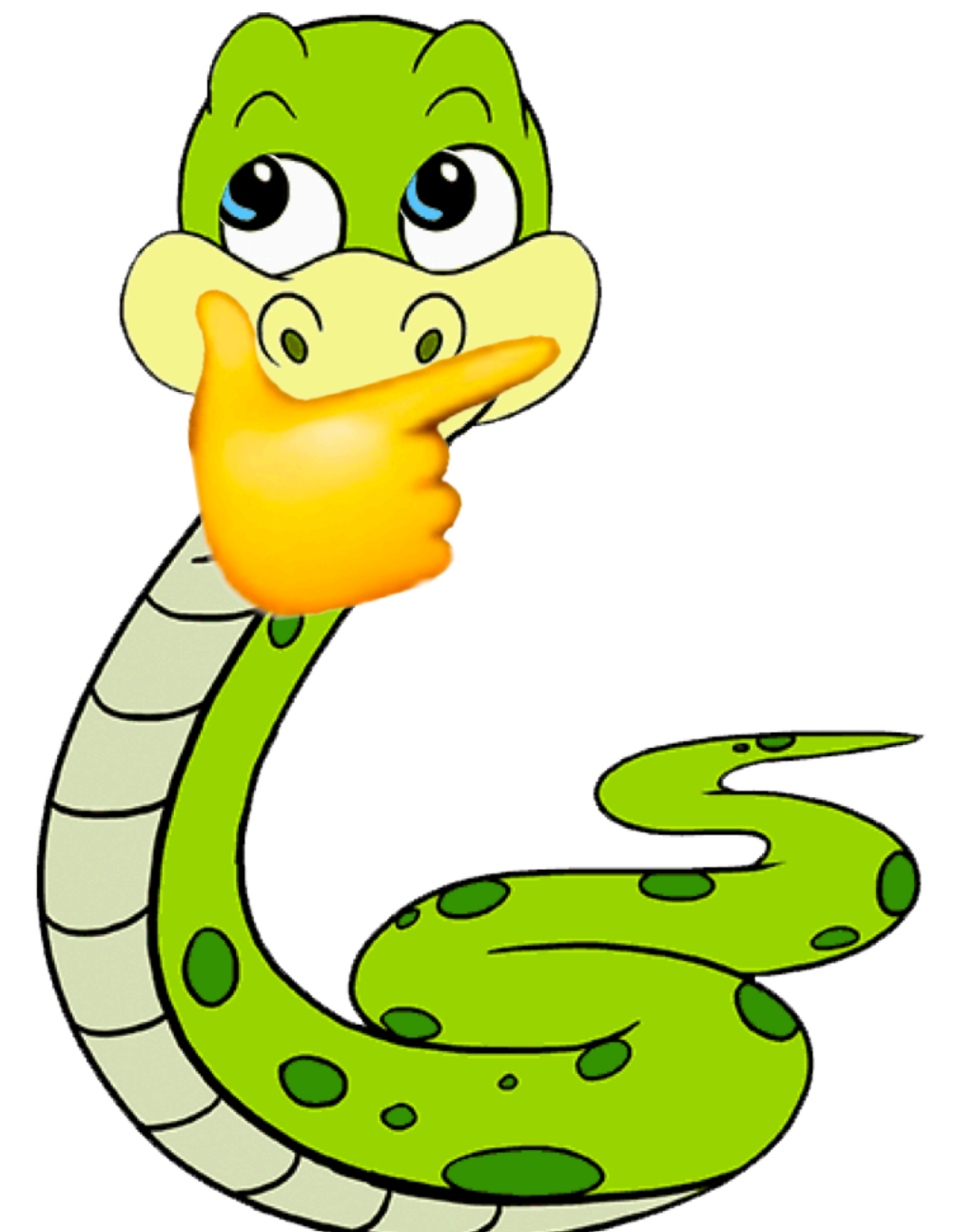
Weaknesses

- If the higher HRT found an escape segment to the end, **SneakySnake will still iterate through the lower HRTs**
- Snake-on-Chip is **more expensive and less accurate** than the other versions
- Sometimes definitions are unclear

Unclear definitions

$$Z[i, j] = \begin{cases} 0, & \text{if } i = E + 1, Q[j] = R[j], \\ 0, & \text{if } 1 \leq i \leq E, Q[j - i] = R[j], \\ 0, & \text{if } i > E + 1, Q[j + i - E - 1] = R[j], \\ 1, & \text{otherwise} \end{cases} \quad (1)$$

What if $j < i$?



Unclear definitions

```
printf("%zu - ", e);  
for (n = 0; n < (ReadLength) ; n++) {  
    if (n < e)  
        printf("1");  
    else if (ReadSeq[n-e] != RefSeq[n])  
        printf("1");  
    else if (ReadSeq[n-e] == RefSeq[n]) {  
        printf("0");  
    }  
}
```


Unclear definitions

$$Z[i, j] = \begin{cases} 1, & \text{if } j \leq i \text{ or } j + i - E - 1 > m \\ 0, & \text{if } i = E + 1, Q[j] = R[j], \\ 0, & \text{if } 1 \leq i \leq E, Q[j - i] = R[j], \\ 0, & \text{if } i > E + 1, Q[j + i - E - 1] = R[j], \\ 1, & \text{otherwise} \end{cases} \quad (1)$$

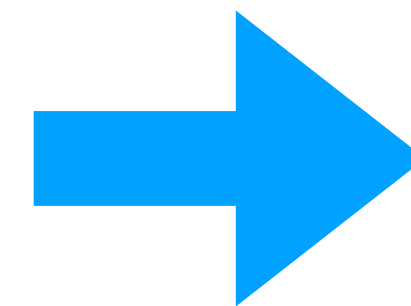
This tells the reader what happens for **index out of bound**

Unclear definitions

Do we create the maze in advance or not?

algorithm from ever searching backward for the longest escape segment. This leads to a signal net that has non-overlapping escape segments.

To achieve these two key objectives, the SneakySnake algorithm applies five effective steps. (1) The SneakySnake algorithm first constructs the chip maze using Equation 1. It then considers the first column of the chip maze as the first checkpoint, where the first iteration starts. (2) At each new checkpoint, the SneakySnake algorithm always selects the longest escape segment that allows the signal to travel as far forward as possible until it reaches an obstacle. For each row of the chip maze, it computes the length of the first horizontal segment of consecutive entries of value



alignment algorithm. Otherwise, the SneakySnake algorithm terminates without performing computationally expensive sequence alignment, since the differences between sequences is guaranteed to be $> E$.

To efficiently implement the SneakySnake algorithm, we use an implicit representation of the chip maze. That is, the SneakySnake algorithm starts computing on-the-fly one entry of the chip maze after another for each row until it faces an obstacle (i.e., $Z[i,j] = 1$) or it reaches the end of the current row. Thus, the entries that are actually calculated for each row of the chip maze are the entries that are located only between each checkpoint and the first obstacle, in each row, following this checkpoint, as we show in Fig. 2(c).

Weaknesses

- If the higher HRT found an escape segment to the end, **SneakySnake will still iterate through the lower HRTs**
- Snake-on-Chip is **more expensive and less accurate** than the other versions
- Sometimes definitions are unclear
- Result analysis of read mapping doesn't include any figures
- Requires a high knowledge about the topic to understand the paper
- Indexing starts at 1

Do you have any questions?

Discussion


We can convert **binary code** into **DNA** base pairs

www.nature.com/scientificreports

SCIENTIFIC REPORTS

OPEN

Demonstration of End-to-End Automation of DNA Data Storage

Christopher N. Takahashi¹, Bichlien H. Nguyen^{1,2}, Karin Strauss^{1,2} & Luis Ceze¹ 

Synthetic DNA has emerged as a novel substrate to encode computer data with the potential to be orders of magnitude denser than contemporary cutting edge techniques. However, even with the help of automated synthesis and sequencing devices, many intermediate steps still require expert laboratory technicians to execute. We have developed an automated end-to-end DNA data storage device to explore the challenges of automation within the constraints of this unique application. Our device encodes data into a DNA sequence, which is then written to a DNA oligonucleotide using a custom DNA synthesizer, pooled for liquid storage, and read using a nanopore sequencer and a novel, minimal preparation protocol. We demonstrate an automated 5-byte write, store, and read cycle with a modular design enabling expansion as new technology becomes available.

Storing information in DNA is an emerging technology with considerable potential to be the next generation storage medium of choice. Recent advances have shown storage capacity grow from hundreds of kilobytes to megabytes to hundreds of megabytes¹⁻³. Although contemporary approaches are book-ended with mostly automated synthesis⁴ and sequencing technologies (e.g., column synthesis, array synthesis, Illumina, nanopore, etc.), significant intermediate steps remain largely manual^{1-3,5}. Without complete automation in the write to store to read cycle of data storage in DNA, it is unlikely to become a viable option for applications other than extremely seldom read archival.

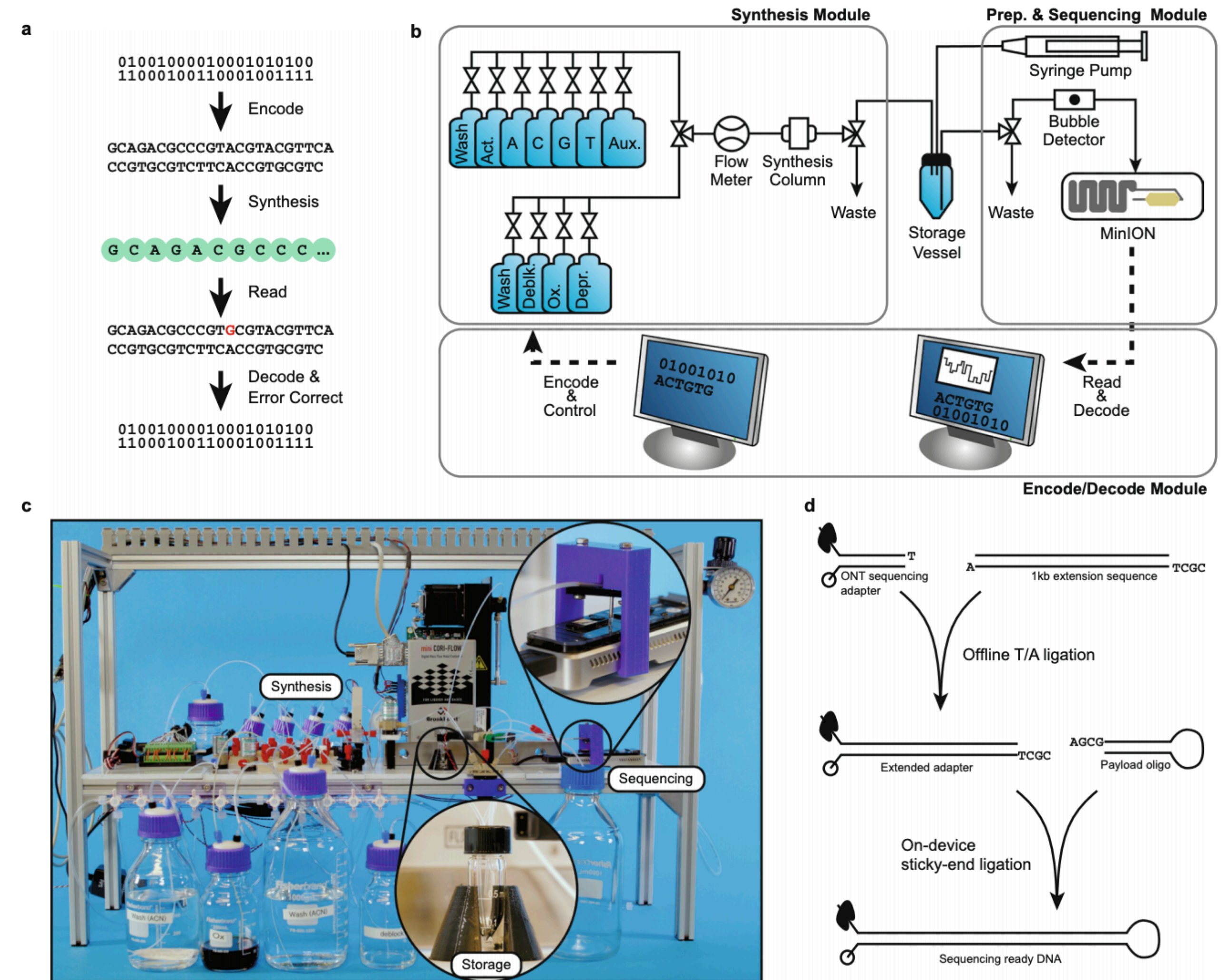
To demonstrate the practicality of integrating fluidics, electronics and infrastructure, and explore the challenges of full DNA storage automation, we developed the first full end-to-end automated DNA storage device. Our device is intended to act as a proof-of-concept that provides a foundation for continuous improvements, and as a first application of modules that can be used in future molecular computing research. As such, we adhered to specific design principles for the implementation: (1) maximize modularity for the sake of replication and reuse.

Received: 26 October 2018
Accepted: 5 March 2019
Published online: 21 March 2019

Discussion

We can convert **binary code** into **DNA** base pairs

They claim “*synthetic DNA has emerged as a novel substrate to encode computer data with the potential to be orders of magnitude denser than contemporary cutting edge techniques*”



Discussion

We can convert binary code into DNA base pairs

They claim “*synthetic DNA has emerged as a novel substrate to encode computer data with the potential to be orders of magnitude denser than contemporary cutting edge techniques*”

They tested it only with the String “HELLO”

They used **Parasail** for DNA alignment

DNA alignment. All DNA alignment was done using the parasail parasail_aligner command line tool¹⁸ with arguments -d -t 1 -O SSW -a sg_trace_striped_16 -o 8 -m NUC.4.4 -e 4. Alignments to the adapter sequence for decoding used the additional flag -c 20, while payload error analysis used flag -c 8.

Our system's write-to-read latency is approximately 21 h. The majority of this time is taken by synthesis, viz., approximately 305 s per base, or 8.4 h to synthesize a 99-mer payload and 12 h to cleave and deprotect the oligonucleotides at room temperature. After synthesis, preparation takes an additional 30 min, and nanopore reading and online decoding take 6 min.

Using this prototype system, we stored and subsequently retrieved the 5-byte message “HELLO” (01001000 01000101 01001100 01001100 01001111 in bits). Synthesis yielded approximately 1 mg of DNA, with approximately $4\mu\text{g} \approx 100\text{ pmol}$ retained for sequencing. Nanopore sequencing yielded 3469 reads, 1973 of which aligned to our adapter sequence. Of the aligned sequences, 30 had extractable payload regions. Of those, 1 was successfully decoded with a perfect payload. The remaining 29 payloads were rejected by the decoder for being irrecoverably corrupt.

Discussion

Does SneakySnake enable us to use DNA as memory storage?
If no, would it at least improve it?

Discussion

Does SneakySnake enable us to use DNA as memory storage?

If no, would it at least improve it?

- Helps a lot if small edit distance is possible
- Considering smaller sequences, the CPU based version is up to 40x faster than Parasail.
- In general, it would indeed accelerate the procedure, but still not fast enough to make DNA and other areas are lacking of performance as well

Discussion

Could we improve the runtime of Sequence-alignment instead of pre-filtering to make read mapping faster?

Discussion

Could we improve the time complexity of Sequence-alignment?

Unfortunately the **fastest sequence-alignment** is **proven** to be $O(m^2/\log m)$
(Otherwise we can solve 3-sat in less than $O(n^2)$)

➡ We need to decrease the speed of other areas.

Edit Distance Cannot Be Computed
in Strongly Subquadratic Time
(unless SETH is false)*

Arturs Backurs[†]
MIT

Piotr Indyk[‡]
MIT

Abstract

The edit distance (a.k.a. the Levenshtein distance) between two strings is defined as the minimum number of insertions, deletions or substitutions of symbols needed to transform one string into another. The problem of computing the edit distance between two strings is a classical computational task, with a well-known algorithm based on dynamic programming. Unfortunately, all known algorithms for this problem run in nearly quadratic time.

In this paper we provide evidence that the near-quadratic running time bounds known for the problem of computing edit distance might be tight. Specifically, we show that, if the edit distance can be computed in time $O(n^{2-\delta})$ for some constant $\delta > 0$, then the satisfiability of conjunctive normal form formulas with N variables and M clauses can be solved in time $M^{O(1)}2^{(1-\epsilon)N}$ for a constant $\epsilon > 0$. The latter result would violate the *Strong Exponential Time Hypothesis*, which postulates that such algorithms do not exist.

Discussion

SneakySnake has **no data dependencies**, but still needs to access a high amount of data

Could we improve SneakySnake's performance if we used in-memory processing or in-cache processing?

Discussion

Could we improve SneakySnake's performance if we used in-memory processing or in-cache processing?

This is the idea of **GenCache** (with GenAx, not SneakySnake), that uses **in-cache operations** to accelerate sequence aligners

They significantly reduced **energy consumption** (8.3x less) and **execution time** (5.8x less)

GenCache: Leveraging In-Cache Operators for Efficient Sequence Alignment

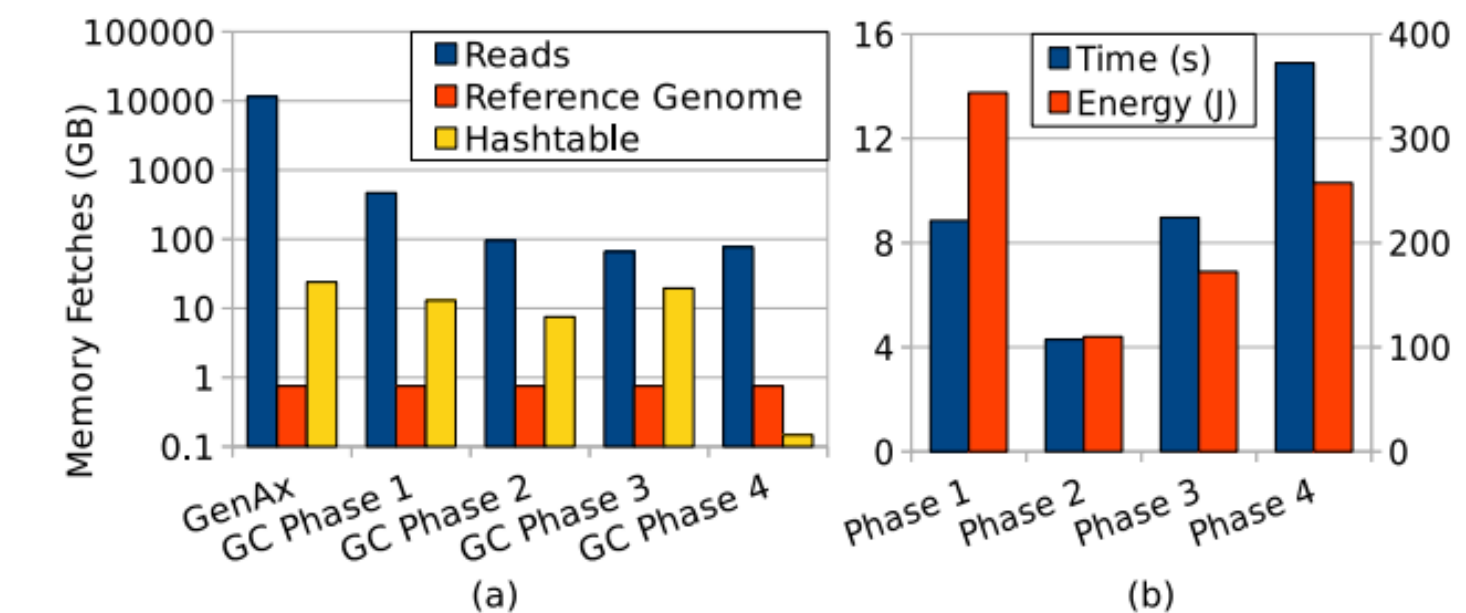


Figure 11: (a) Breakdown of time and energy by program phase. (b) Breakdown of memory accesses by program phase and data structure.

(4× speedup from in-cache operators and algorithm) than the sum of its parts (1.36× from algorithm alone and 1.62× from in-cache operators alone). The addition of the bloom filter alleviates the remaining memory bottleneck from hash table misses in Phases 1 and 2 to yield a 5.26× speedup over baseline GenAx.

Figure 10 shows an energy comparison (in terms of reads per mJ) and memory fetches for the same six configurations. Most of the energy reduction is from a reduction in memory accesses. The

Discussion

Could we improve SneakySnake's performance if we used in-memory processing or in-cache processing?

This is the idea of **GenCache** (with GenAx, not SneakySnake), that uses **in-cache operations** to accelerate sequence aligners

They significantly reduced **energy consumption** (8.3x less) and **execution time** (5.8x less)

They use a special architecture, SS would lose it's "independence"

GenCache: Leveraging In-Cache Operators for Efficient Sequence Alignment

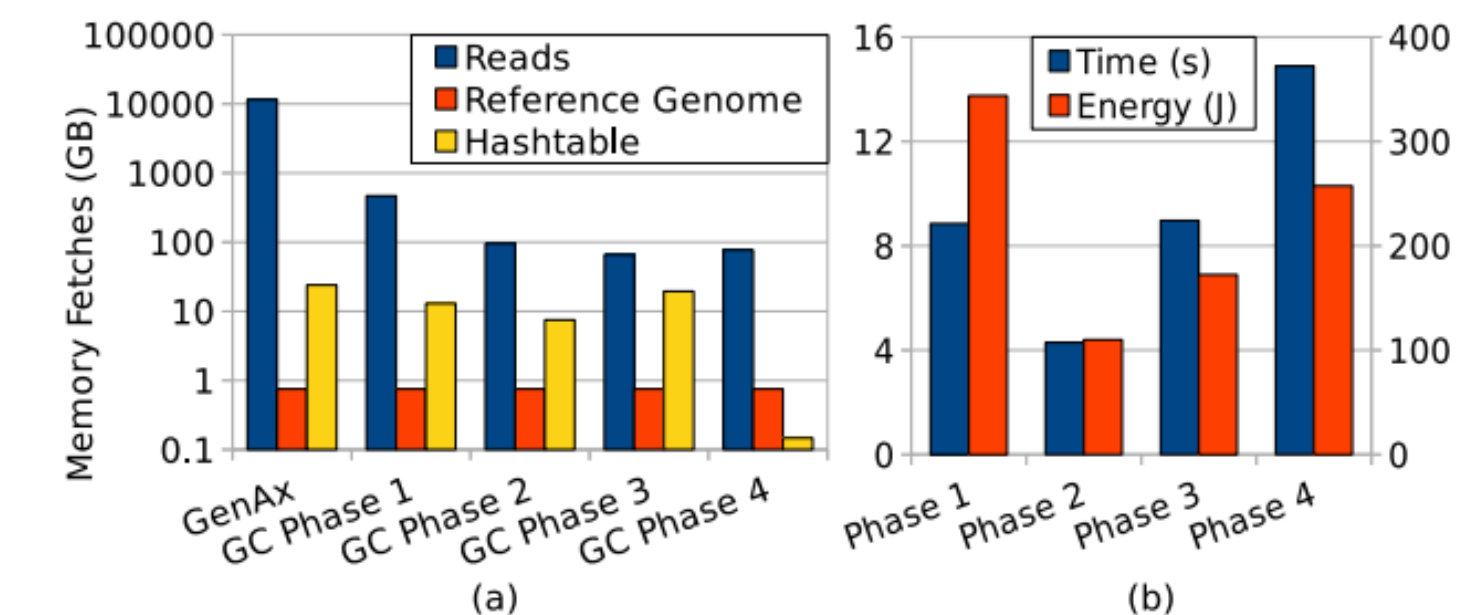


Figure 11: (a) Breakdown of time and energy by program phase. (b) Breakdown of memory accesses by program phase and data structure.

(4x speedup from in-cache operators and algorithm) than the sum of its parts (1.36x from algorithm alone and 1.62x from in-cache operators alone). The addition of the bloom filter alleviates the remaining memory bottleneck from hash table misses in Phases 1 and 2 to yield a 5.26x speedup over baseline GenAx.

Figure 10 shows an energy comparison (in terms of reads per mJ) and memory fetches for the same six configurations. Most of the energy reduction is from a reduction in memory accesses. The

Discussion

Could we do better if we sacrificed the “platform independence of SneakySnake”?

Discussion

Could we do better if we sacrificed the “platform independence of SneakySnake”?

GenASM creates a framework to **remove limitation** of Bitap (like GenAx) on current systems

They compare it to Shouji and provide 3.7x speedup, thus faster than SneakySnake **(only with 100bp)**

➡ Design of **special framework** could speed up SneakySnake

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

Genome sequence analysis has enabled significant advancements in medical and scientific areas such as personalized medicine, outbreak tracing, and the understanding of evolution. To perform genome sequencing, devices extract small random fragments of an organism's DNA sequence (known as reads). The first step of genome sequence analysis is a computational process known as read mapping. In read mapping, each fragment is matched to its potential location in the reference genome with the goal of identifying the original location of each read in the genome. Unfortunately, rapid genome sequencing is currently bottlenecked by the computational power and memory

amounts of genomics data at low cost [8, 118, 153], but are unable to extract an organism's complete DNA in one piece. Instead, these machines extract smaller random fragments of the original DNA sequence, known as *reads*. These reads then pass through a computational process known as *read mapping*, which takes each read, aligns it to one or more possible locations within the reference genome, and finds the matches and differences (i.e., *distance*) between the read and the reference genome segment at that location [6, 177]. Read mapping is the first key step in genome sequence analysis. State-of-the-art sequencing machines produce broadly one

We compare GenASM with the state-of-the-art FPGA-based pre-alignment filter for short reads, Shouji [9], using two datasets provided in [9]. When we compare Shouji (with maximum filtering units) and GenASM for the dataset with 100bp sequences, we find that **GenASM provides 3.7× speedup over Shouji**, while reducing power consumption by 1.7×. When we perform the same analysis with 250bp sequences, we find that GenASM does not provide speedup over Shouji, but reduces power consumption by 1.6×.

Different versions of SneakySnake

Comparison

Snake-on-Chip

- +Scalable and parallizable
- +More energy efficient than Snake-on-GPU
- More expensive and time consuming
- You can't configure the parameters after design time!!!

Snake-on-GPU

- +Easier to configure
- +Less expensive and time consuming
- +Scalable and parallizable
- Not as energy efficient as Snake-on-Chip

Discussion

As a lab offering genome sequencing, would you rather buy a sequencer based on Snake-on-Chip or Snake-on-GPU?

Discussion

As a lab offering genome sequencing, would you rather buy a sequencer based on Snake-on-Chip or Snake-on-GPU?

- ➡ Depends on how many customers we expect. Snake-on-GPU worth it if we can exploit the thousands of threads a GPU offers

Thank you for your attention!