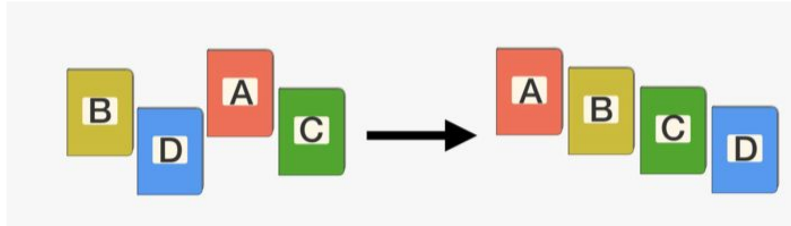
Parallel Pairwise Operations on Data Stored in DNA: Sorting, Shifting and Searching

Tonglin Chen, Arnav Solanki, Marc Riedel

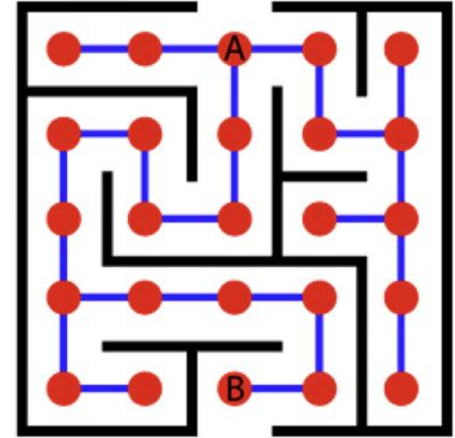
University of Minnesota
Driven to Discover℠

CS 101: Sorting, Searching, Shifting



Sorting

Applications: string processing, genomics, combinatorics, ...
Binary data: majority operations, thresholding operations in neural networks.



Searching

Applications: genomics, data mining, ...
Binary data: pattern matching, ...

Shifting

Applications: image processing, ...
Binary data: multiplying/dividing, scaling, activation functions in neural networks.

Agenda

- Background
  - DNA Storage via Nicks.
  - “In-Memory” Computation: SIMD operations.
- Technical Details
  - Encoding binary data.
  - Transforming bits, pairwise, in parallel.
- Applications
  - Parallel Binary Sorting
  - Parallel Searching
  - Parallel Shifting (omitted)
- Summary

Data Storage: Conventional Approach

Nucleotides: $\{A, C, T, G\}$

DNA: string of nucleotides

Digital files

Binarization

Binary code

01111001 01101111
01110101 00100000
01110010 01101111
01100011 01101011

Encoding

Nucleobase code

CAGGATCA CAGTAGAG
CAGGTACA CCGCTCTCT
CAGGTCAA CAGTAAGG
CAGCTCGC CAGTACGG

Synthesis

DNA

Sequencing

Decoding

Reading

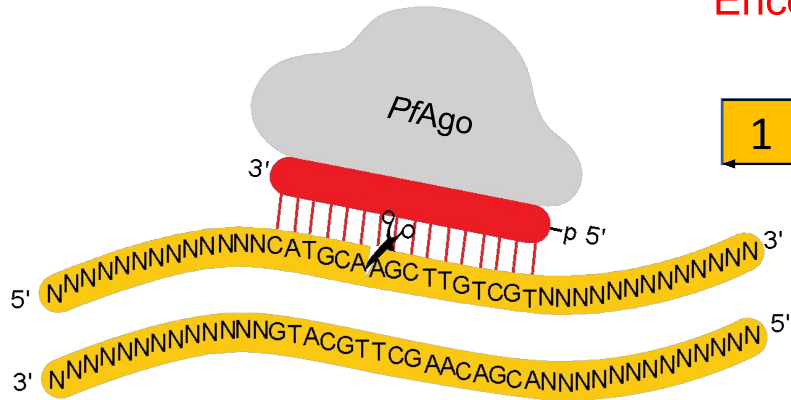
Our storage modality: "Nicks"

Gene editing with CRISPR/Cas9 or PfAgo

Simplest form: a cut represents a 1;
absence of a cut a 0.

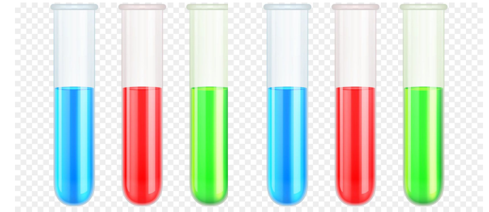
Our storage modality: "Nicks"

Gene editing with CRISPR/Cas9 or PfAgo



Encode data by a pattern of cuts, followed by heating.

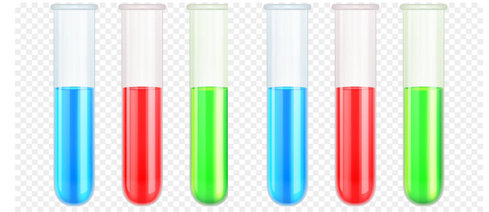
Parallelism with Nick-Based Displacement



0	1	1	0	1	1
1	1	1	1	1	1
0	0	0	0	0	0
0	1	0	0	1	0
0	0	1	0	0	1
1	1	0	1	1	0
0	1	0	0	1	0
0	0	1	0	0	1

- Single-instruction applied to multiple data (SIMD).
- A single common “instruction” can initiate a sequence of computation on many (currently 10s, in the future millions) of “registers”.
- Instruction is a single synthesized strand. Registers are copies of identical DNA nicked to encode different values (so a vector or a matrix.)

Parallelism with Nick-Based Displacement

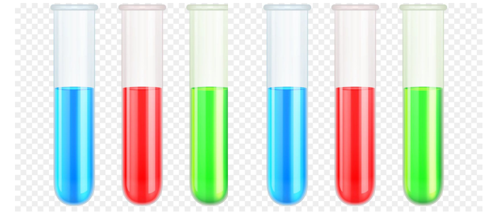


Two levels of parallelism:

1. Bit-level Parallelism: instructions applied to all bits in array at once.
2. Data-level Parallelism: same instructions can be applied to different data in different arrays at once.

0	1	1	0	1	1
1	1	1	1	1	1
0	0	0	0	0	0
0	1	0	0	1	0
0	0	1	0	0	1
1	1	0	1	1	0
0	1	0	0	1	0
0	0	1	0	0	1

Parallelism with Nick-Based Displacement



Two levels of parallelism:

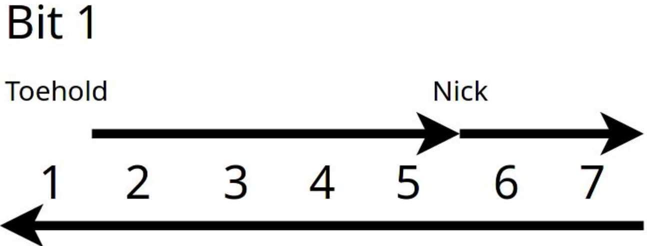
1. Bit-level Parallelism: instructions applied to all bits in array at once.
2. Data-level Parallelism: same instructions can be applied to different data in different arrays at once.

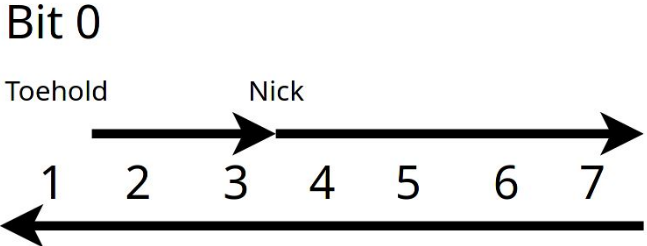
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	0	0	1	0
0	1	1	0	1	1
0	1	1	0	1	1
1	1	1	1	1	1
1	1	1	1	1	1

Wang, Boya, Cameron Chalk, and David Soloveichik. "SIMD|| DNA: single instruction, multiple data computation with DNA strand displacement cascades." *International Conference on DNA Computing and Molecular Programming*. Springer, Cham, 2019.

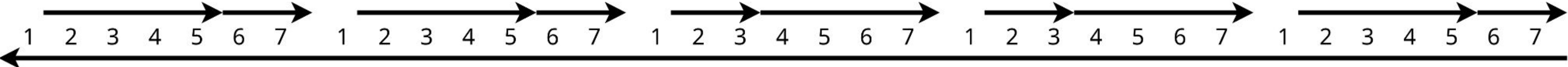
Encoding

➔ How do we represent bits in DNA cells?



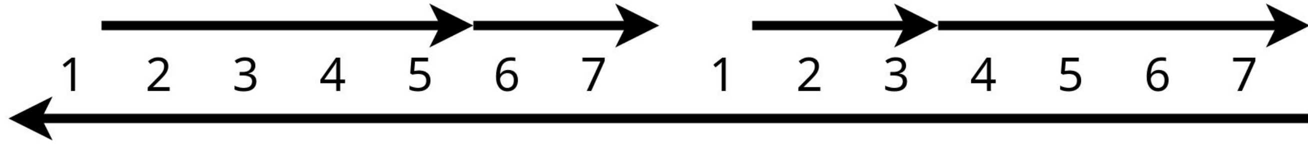


➔ Example: register with 5 cells (1,1,0,0,1)



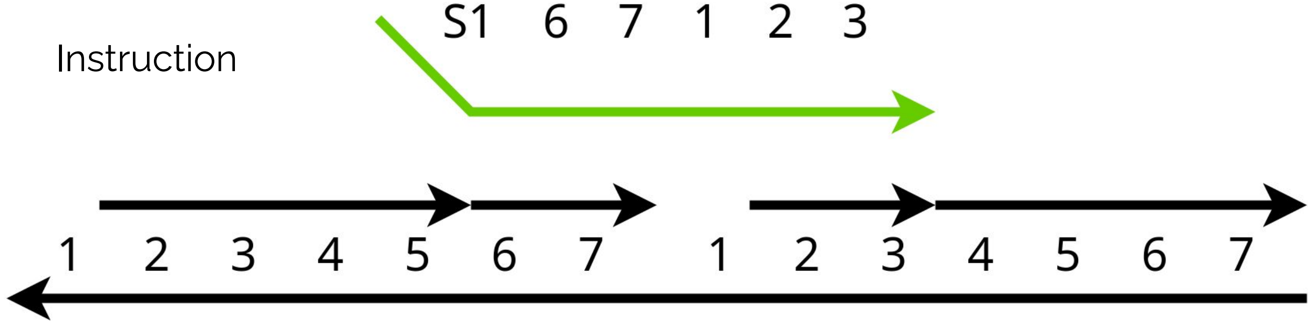
Example: From (1,0) to (0,1)

Original: we have pair (1, 0) here.



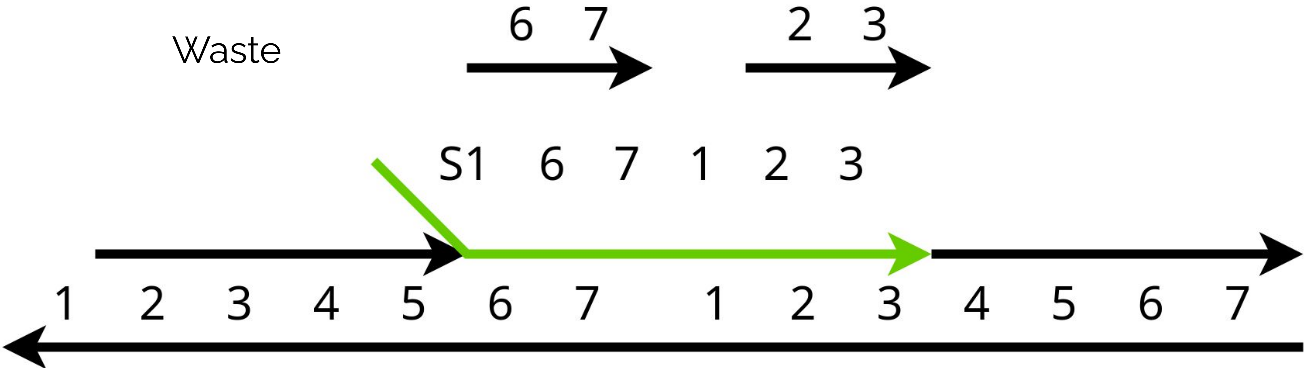
Example: From (1,0) to (0,1)

Step 1: Add strand S1 that covers domains 6 7 1 2 3. Strands (6 7) and (2 3) are displaced



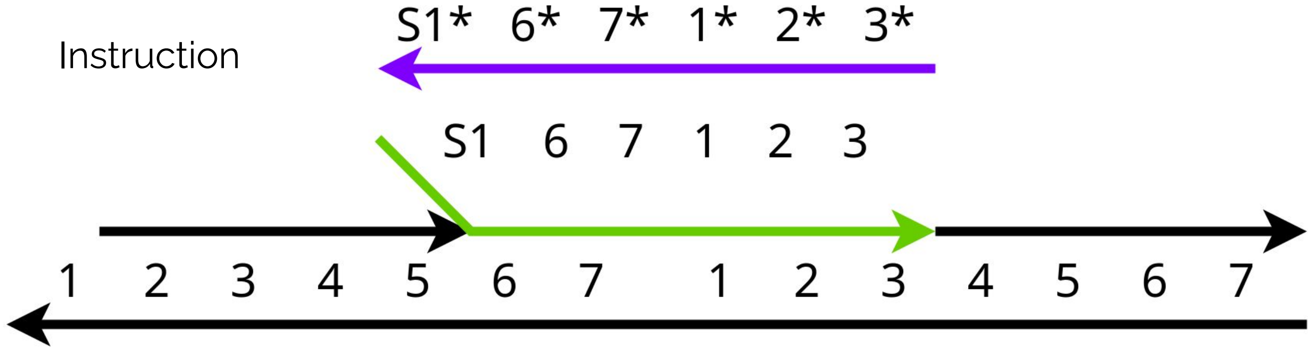
Example: From (1,0) to (0,1)

Step 1: Add strand S1 that covers domains 6 7 1 2 3. Strands (6 7) and (2 3) are displaced



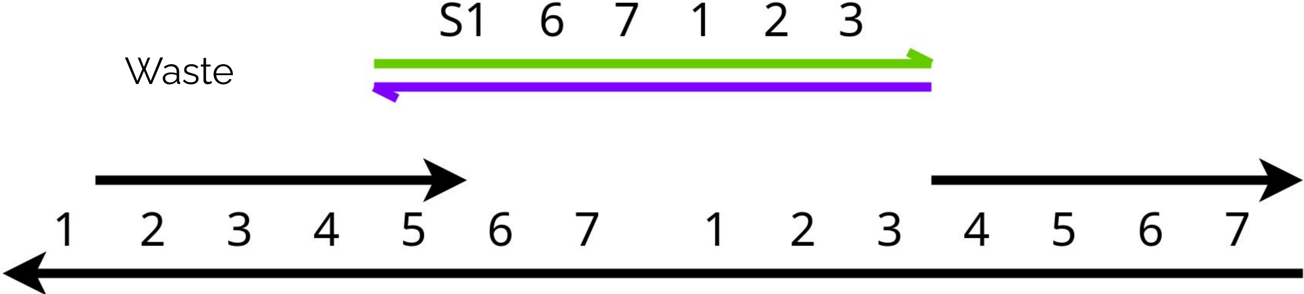
Example: From (1,0) to (0,1)

Step 2: Extract S1 with S1* (complementary strand). S1 and S1* forms a waste complex.



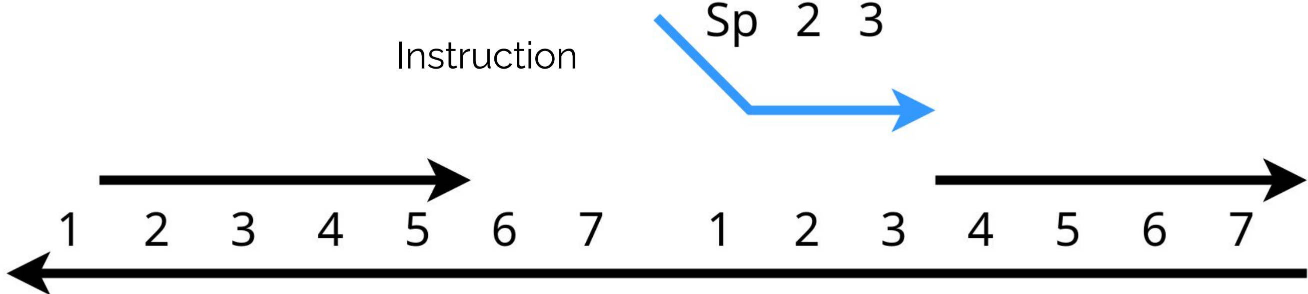
Example: From (1,0) to (0,1)

Step 2: Extract S1 with S1* (complementary strand). S1 and S1* forms a waste complex.



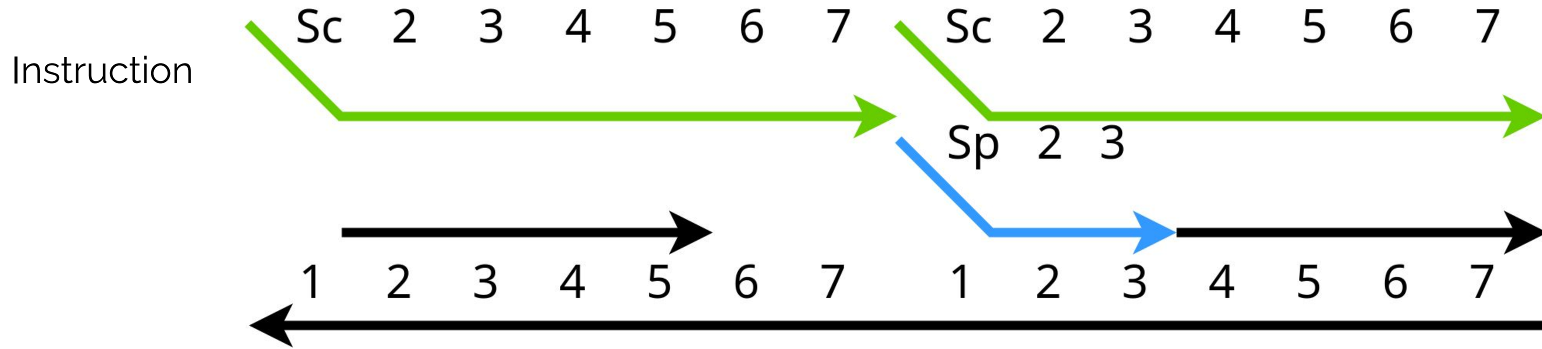
Example: From (1,0) to (0,1)

Step 3: “protect” second cell (in this case bit 0). Then we can focus on rewriting first cell.



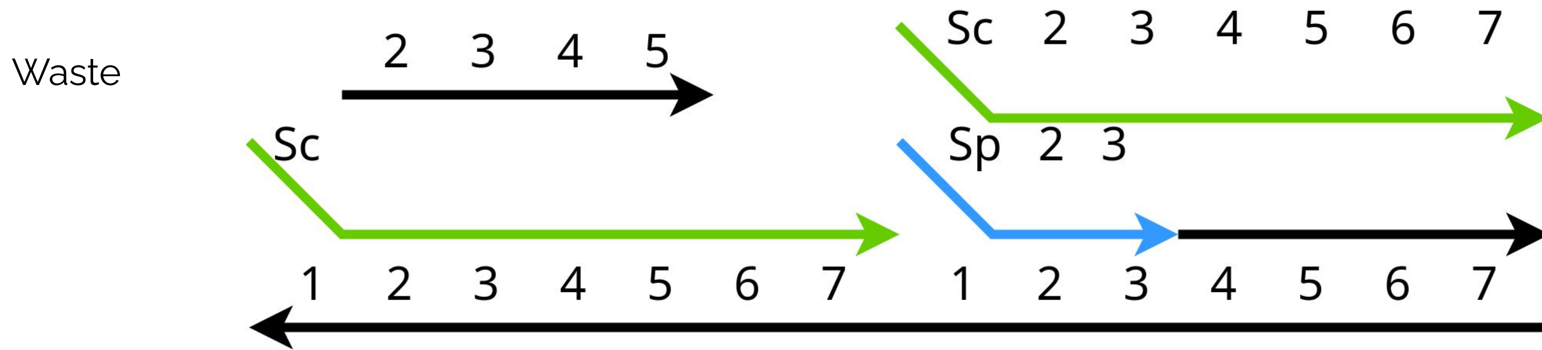
Example: From (1,0) to (0,1)

Step 4: Cover domains 2-7.



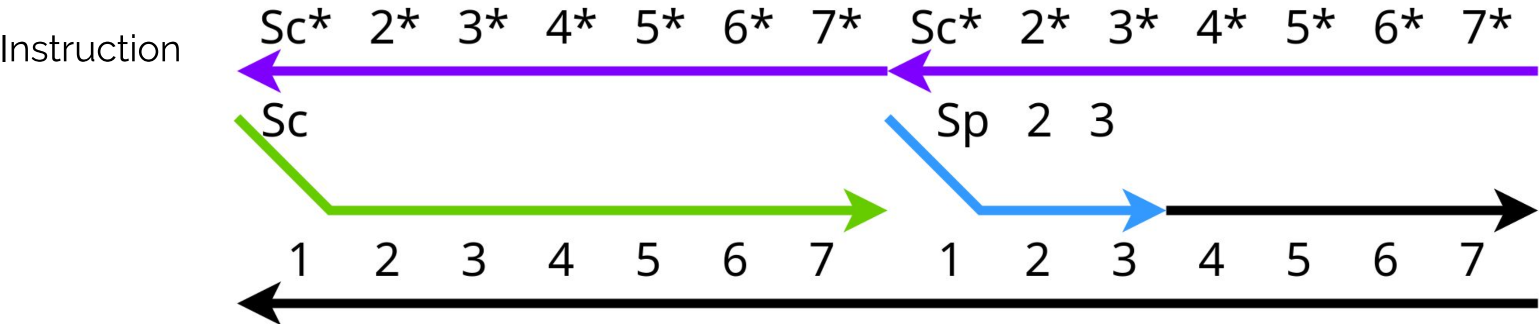
Example: From (1,0) to (0,1)

Step 4: Cover domains 2-7.



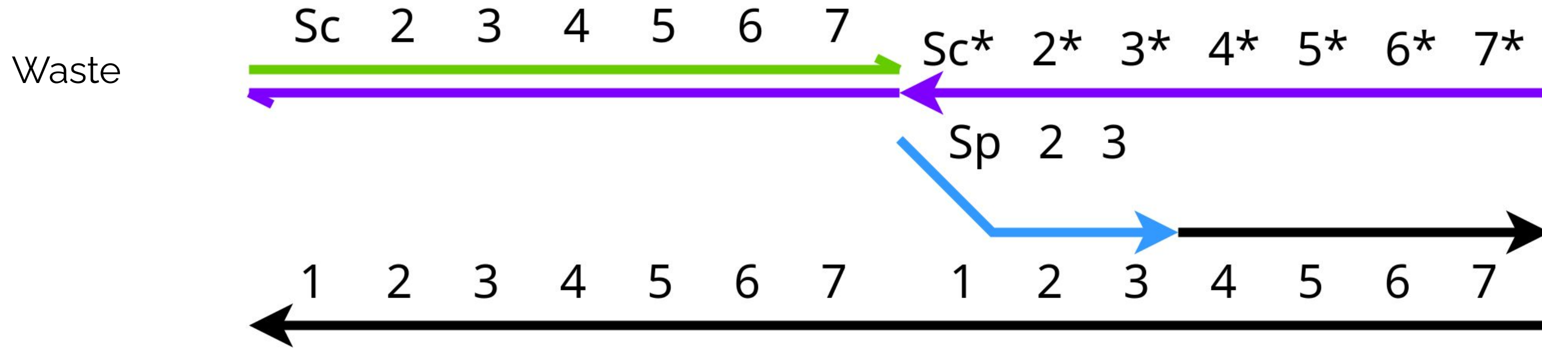
Example: From (1,0) to (0,1)

Step 5: remove cover strand.



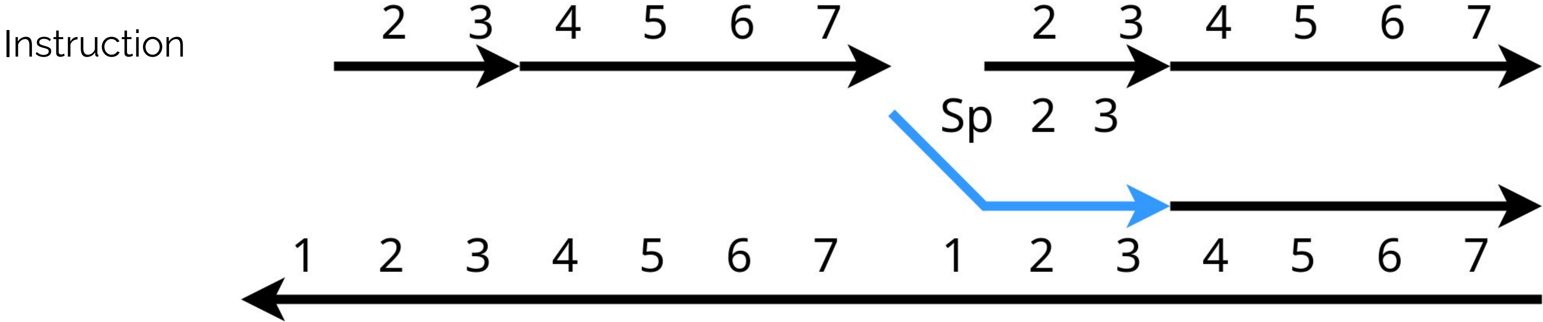
Example: From (1,0) to (0,1)

Step 5: remove cover strand.



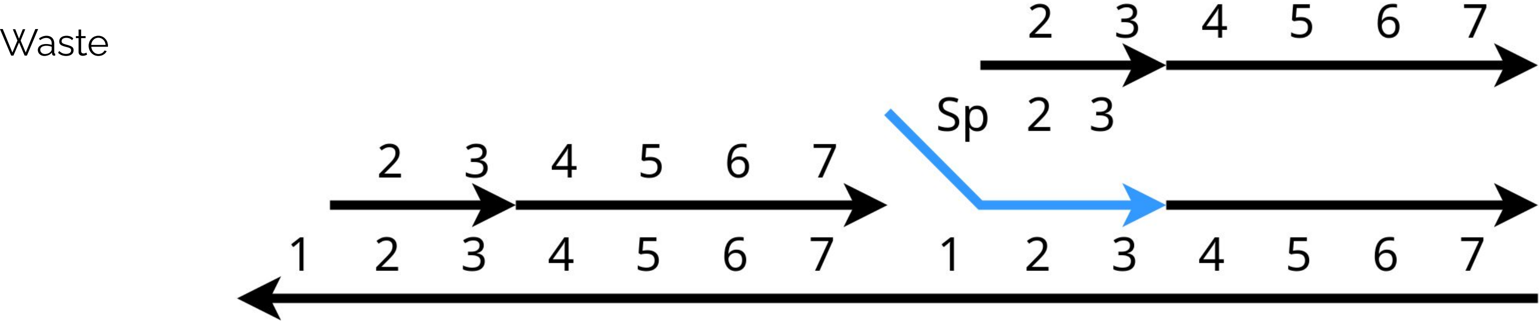
Example: From (1,0) to (0,1)

Step 6: write bit **0**.



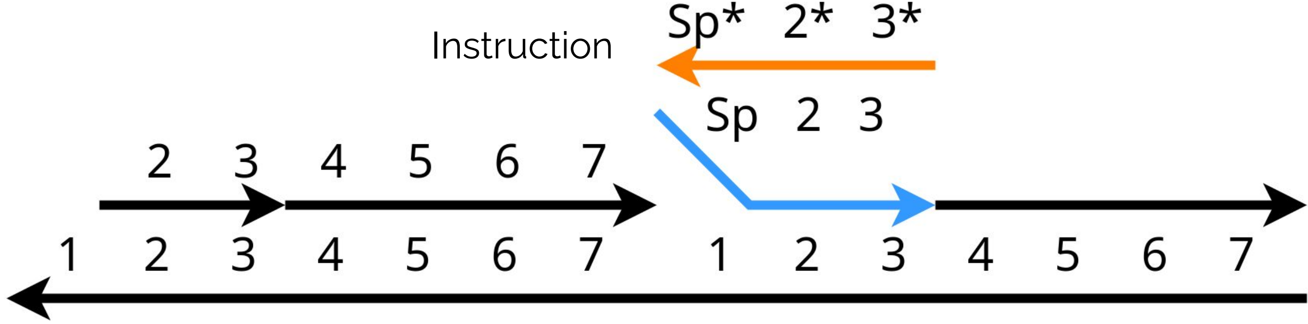
Example: From (1,0) to (0,1)

Step 6: write bit 0.



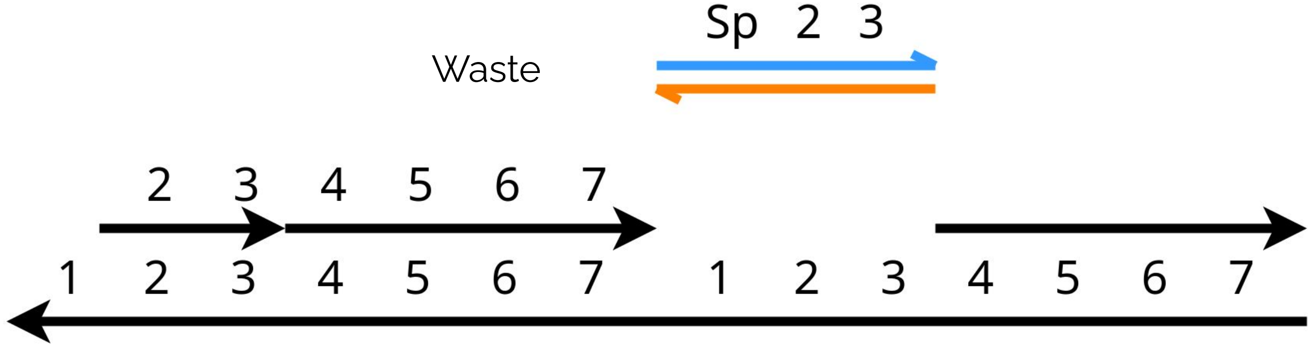
Example: From (1,0) to (0,1)

Step 7: release the protection on the second cell.



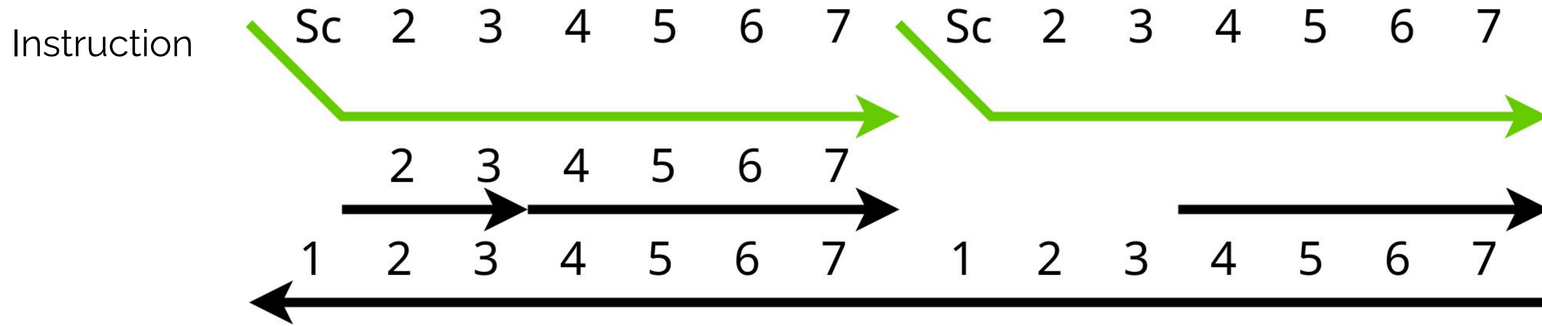
Example: From (1,0) to (0,1)

Step 7: release the protection on second cell.



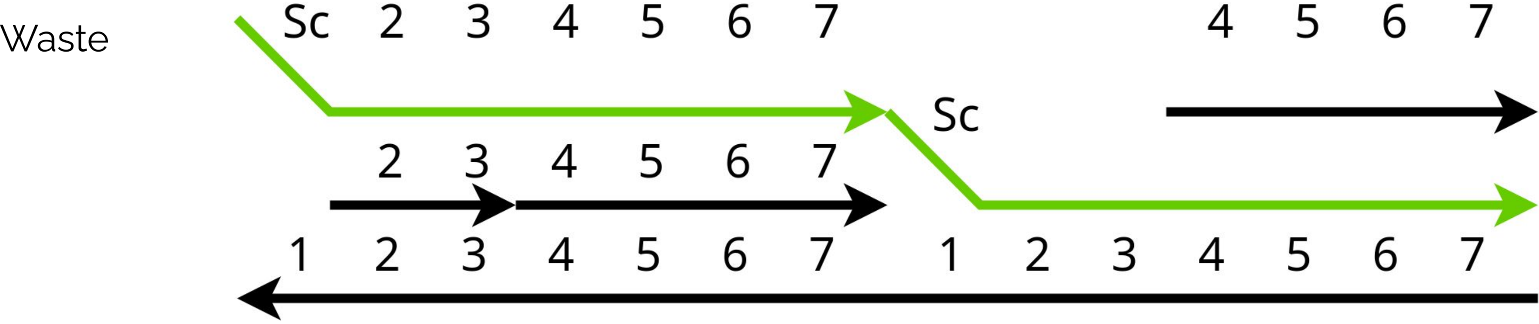
Example: From (1,0) to (0,1)

Step 8: cover domains **2-7**.



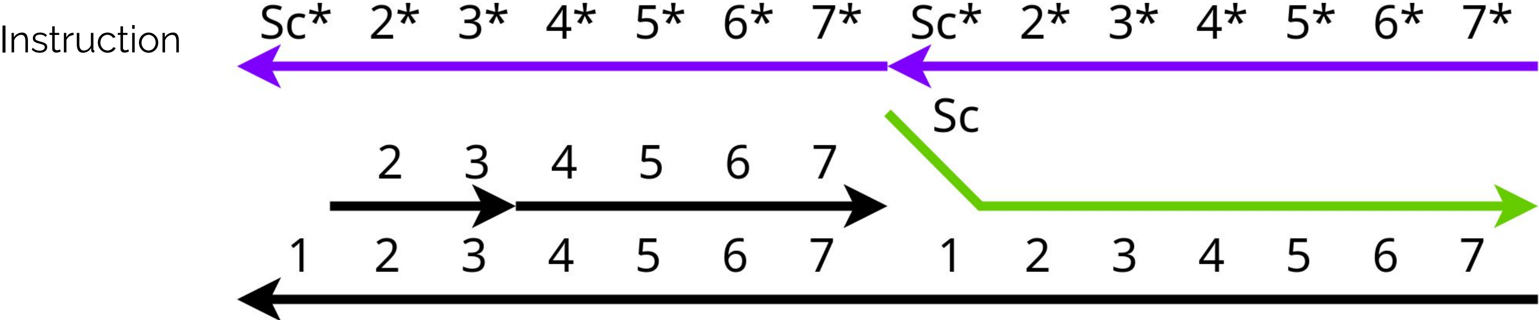
Example: From (1,0) to (0,1)

Step 8: cover domains **2-7**.



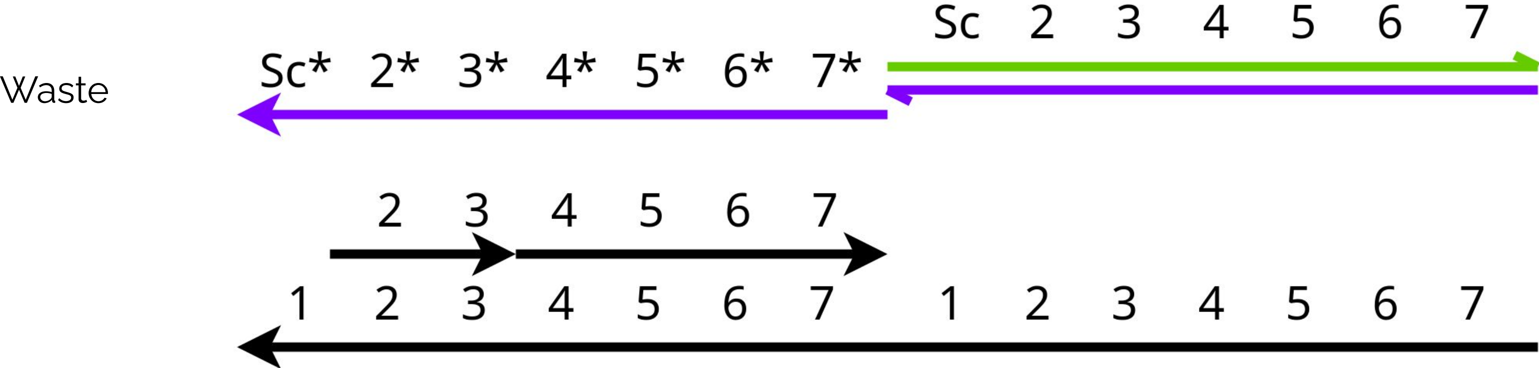
Example: From (1,0) to (0,1)

Step 9: release the cover.



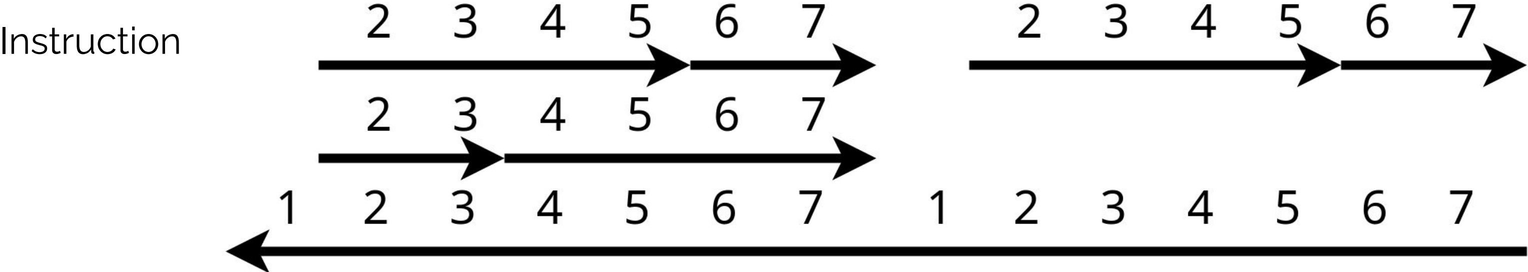
Example: From (1,0) to (0,1)

Step 9: release the cover



Example: From (1,0) to (0,1)

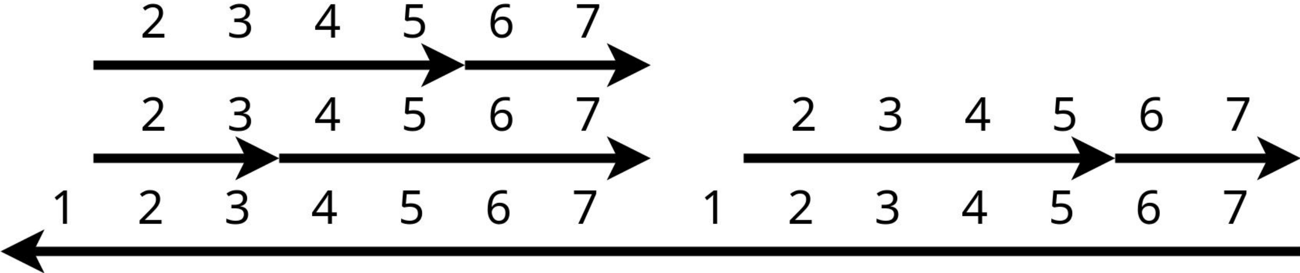
Step 10: write bit 1.



Example: From (1,0) to (0,1)

Step 10: write bit 1 .

Waste

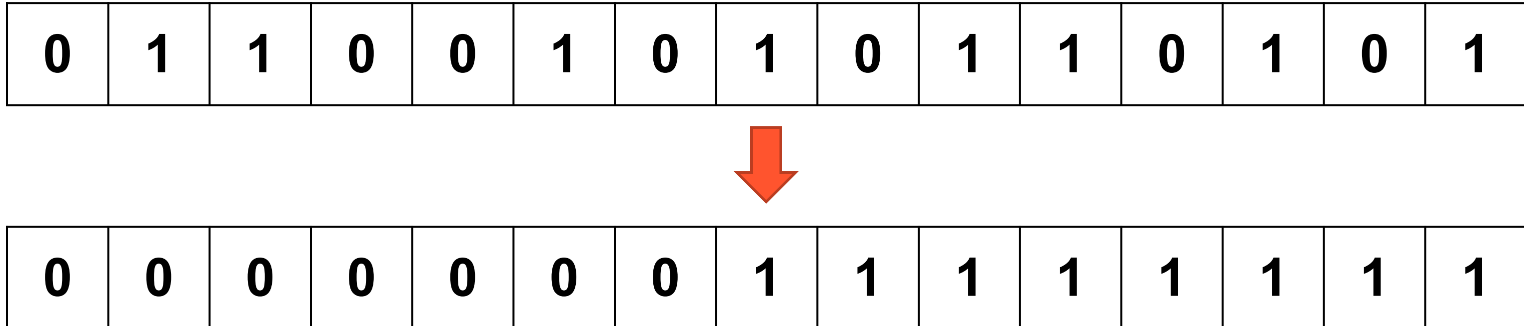


Pairwise Parallel Operations

- Common instructions that operate on a specific pair of bits in a register.
  - 4 possible pairs - (0,0), (0,1), (1,0), and (1,1).
  - Example 1 - identify all cells containing 0 that are followed by cells containing 1 - identify (0,1)
  - Example 2 - convert all pairs (1,0) into (1,1)
  - Must be careful about using random access memory -- are the domain sequences unique or not? Unique sequence allows for specific pair targeting, but requires more instruction strands

Binary Bubble Sorting

- One of the most basic computing tasks: rearrange a list of items into ascending/descending order.
- Serial sorting algorithms take approximately $n\log_2 n$ steps to sort n items.
- Parallel sorting algorithms take approximately n parallel steps to sort n items.

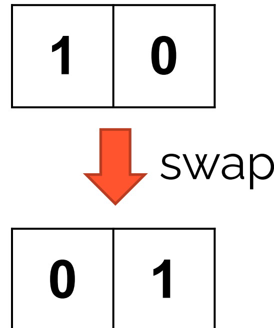


Binary Bubble Sorting

Repeat n times

For every pair of bits (*overlapping*)

If pair is (1, 0), swap to (0, 1)



Binary Bubble Sorting

Starting list

0	1	1	0	0	1	0	1	0	1	1	0	1	0	1

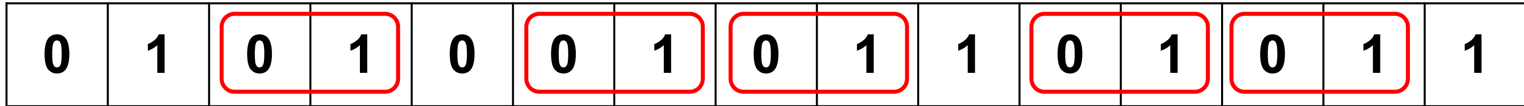
Binary Bubble Sorting

Locate all (1,0) pairs

0	1	1	0	0	1	0	1	0	1	1	0	1	0	1

Binary Bubble Sorting

Swap to (0,1) pairs



Binary Bubble Sorting

Locate all (1,0) pairs

0	1	0	1	0	0	1	0	1	1	0	1	0	1	1

Binary Bubble Sorting

Swap to (0,1) pairs

0	0	1	0	1	0	0	1	1	0	1	0	1	1	1

Binary Bubble Sorting

Locate all (1,0) pairs

0	0	1	0	1	0	0	1	1	0	1	0	1	1	1

Binary Bubble Sorting

Swap to (0,1) pairs

0	0	0	1	0	1	0	1	0	1	0	1	1	1	1

Binary Bubble Sorting

Locate all (1,0) pairs

0	0	0	1	0	1	0	1	0	1	0	1	1	1	1

Binary Bubble Sorting

Swap to (0,1) pairs

0	0	0	0	1	0	1	0	1	0	1	1	1	1	1

Binary Bubble Sorting

Locate all (1,0) pairs

0	0	0	0	1	0	1	0	1	0	1	1	1	1	1

Binary Bubble Sorting

Swap to (0,1) pairs

0	0	0	0	0	1	0	1	0	1	1	1	1	1	1

Binary Bubble Sorting

Locate all (1,0) pairs

0	0	0	0	0	1	0	1	0	1	1	1	1	1	1

Binary Bubble Sorting

Swap to (0,1) pairs

0	0	0	0	0	0	1	0	1	1	1	1	1	1	1

Binary Bubble Sorting

Locate all (1,0) pairs

0	0	0	0	0	0	1	0	1	1	1	1	1	1	1

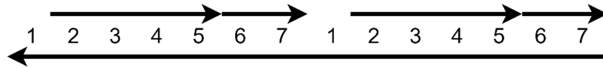
Binary Bubble Sorting

Swap to (0,1) pairs, fully sorted

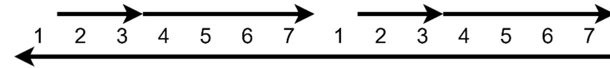
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Other Pairwise operations - (1,1) and (0,0)

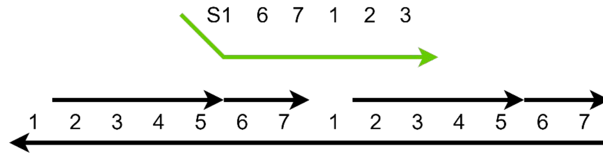
Original: (1,1)

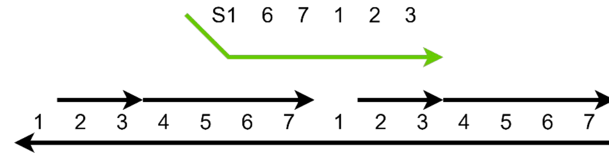


Original: (0,0)

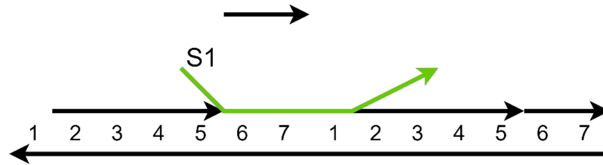


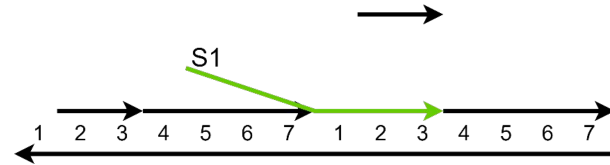
Other Pairwise operations - (1,1) and (0,0)



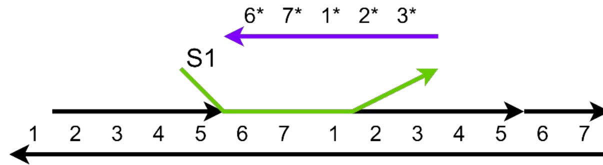


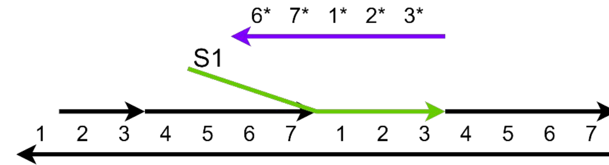
Other Pairwise operations - (1,1) and (0,0)



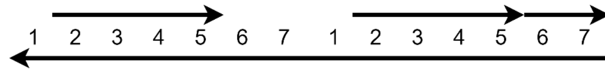


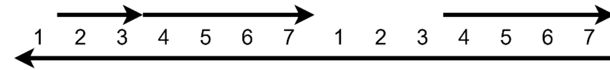
Other Pairwise operations - (1,1) and (0,0)



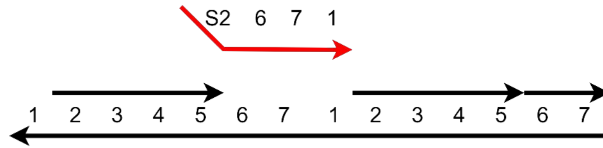


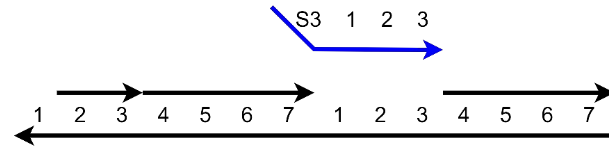
Other Pairwise operations - (1,1) and (0,0)



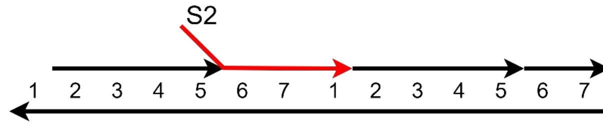


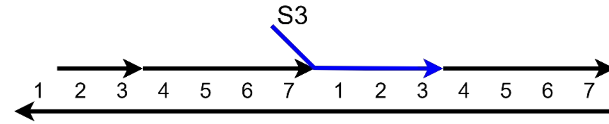
Other Pairwise operations - (1,1) and (0,0)





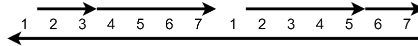
Other Pairwise operations - (1,1) and (0,0)



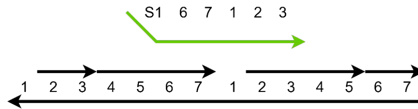


Other Pairwise operations - (0,1)

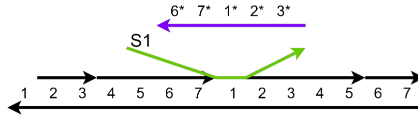
Original: (0,1)



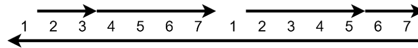
Ins 1: Identifying pair (1,0)



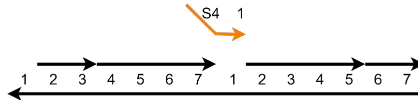
Ins 2: Detaching S1 from non (1,0) pairs



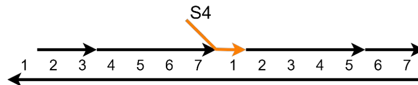
Wash away waste



Ins 3: Identifying pair (0,1)



Result



Parallel Searching - Example 1

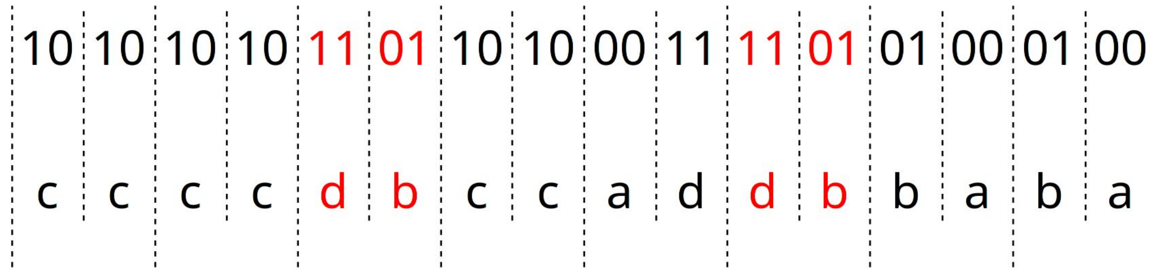
Query = 1101, String = 10101010110110100011110101000100

10	10	10	10	11	01	10	10	00	11	11	01	01	00	01	00

First level: a = 00, b = 01, c = 10, d = 11

Parallel Searching - Example 1

Query = 1101, String = 10101010110110100011110101000100

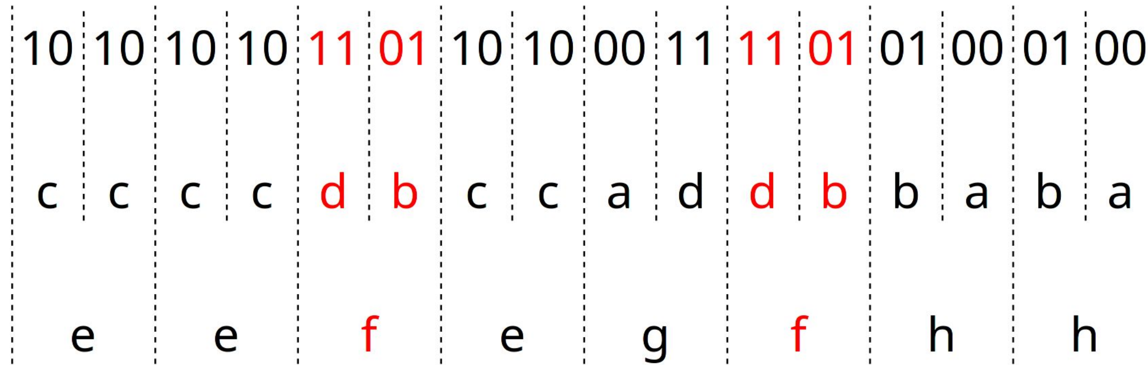


First level: a = 00, b = 01, c = 10, d = 11

Second level: e = cc, f = db, g = ad, h = ba

Parallel Searching - Example 1

Query = 1101, String = 10101010110110100011110101000100



First level: a = 00, b = 01, c = 10, d = 11

Second level: e = cc, f = db, g = ad, h = ba

Note that query = 1101 = db = f.

Parallel Searching - Example 2

Substring does not always start on multiples of query length!

Query = 1011, String = 10101010110110100011110101000100

Solution: Create copies, each with 0 to N-1 bits truncated at the start

10 | 10 | 10 | 10 | 11 | 01 | 10 | 10 | 00 | 11 | 11 | 01 | 01 | 00 | 01 | 00

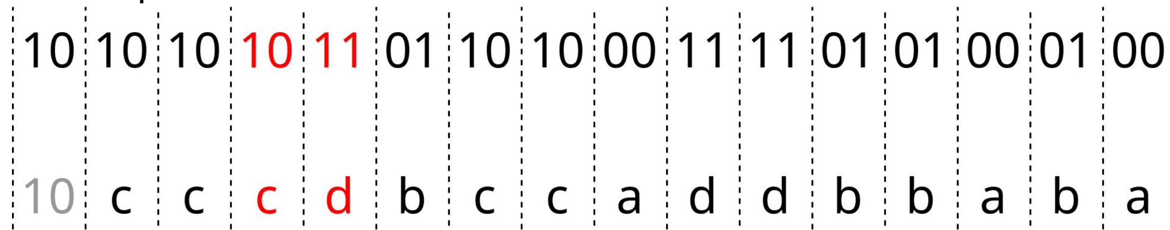
First level: a = 00, b = 01, c = 10, d = 11

Parallel Searching - Example 2

Substring does not always start on multiples of query length!

Query = 1011, String = 10101010110110100011110101000100

Solution: Create copies, each with 0 to N-1 bits truncated at the start



First level: a = 00, b = 01, c = 10, d = 11

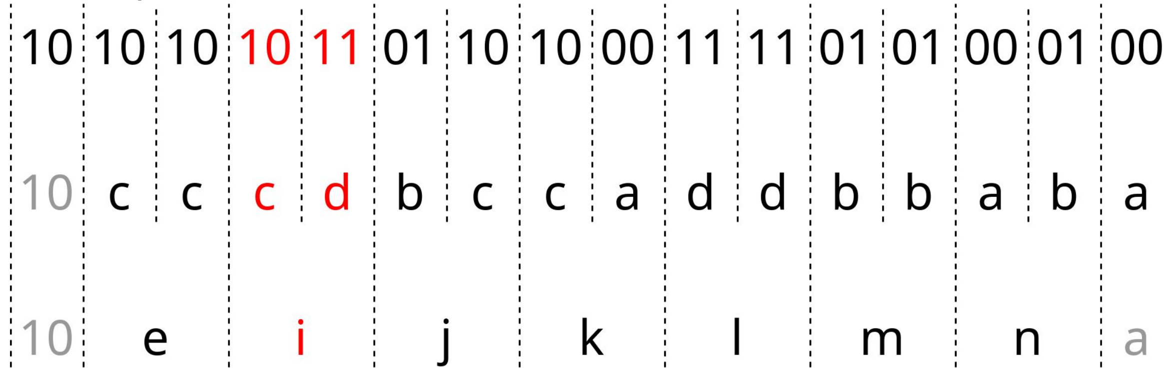
Second level: e = cc, f = db, g = ad, h = ba, i = cd, j = bc, k = ca, l = dd, m = bb, n = ab

Parallel Searching - Example 2

Substring does not always start on multiples of query length!

Query = 1011, String = 10101010110110100011110101000100

Solution: Create copies, each with 0 to N-1 bits truncated at the start

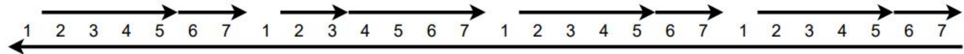


First level: a = 00, b = 01, c = 10, d = 11

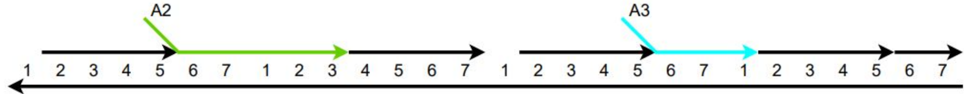
Second level: e = cc, f = db, g = ad, h = ba, i = cd, j = bc, k = ca, l = dd, m = bb, n = ab

Found the query with i

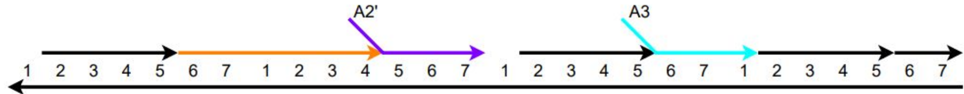
Parallel Searching - Example in DNA



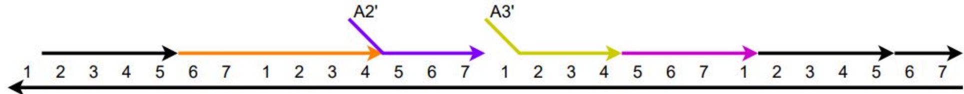
(a) Initial Sequence 1011



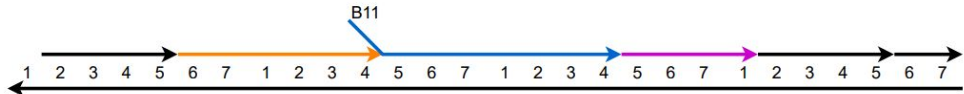
(b) Identifier A_2 captures first pair 10, A_3 captures second pair 11



(c) covering the domain 1 between the two bit pairs



(d) Rewrite the content in the pair so that new identifiers are close to the middle



(e) Two identifier strands replaced by a single identifier if there is a perfect match

Complexity of Search

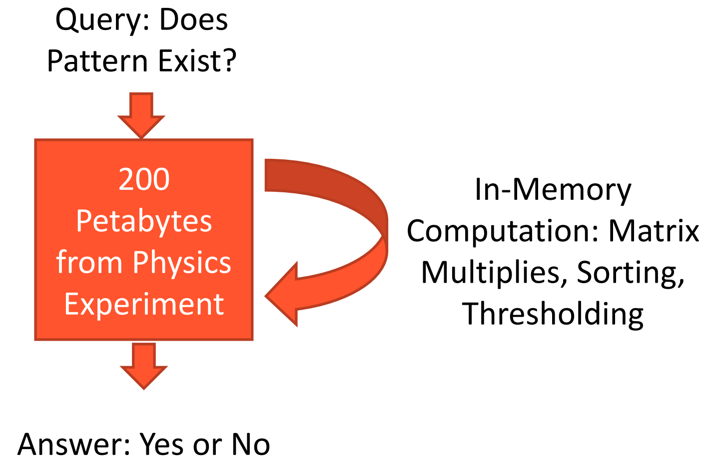
N: length of query string.
M: length data string.

- ➔ Number of levels: O(logN)
- ➔ Number of sequential steps: O(N)
  - ◆ At level i:
    - at most $\frac{n}{2^i}$ pairs of symbols
    - at most 2^{2^i} distinct pairs
  - ◆ first two levels requires fewer steps

Nick-based In-Memory Computing

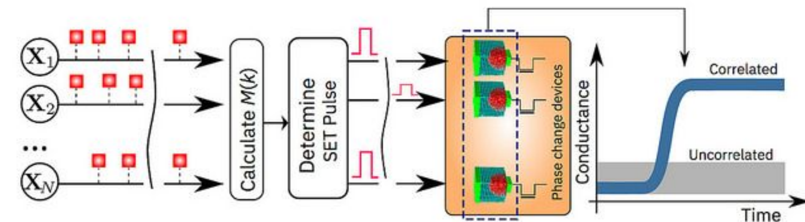
Objectives:

- Leverage the high-density of storage with effective computation.
- Perform "computation in memory" to reduce I/O operations.
- Integrate storage with data-intensive algorithms, such as machine learning.



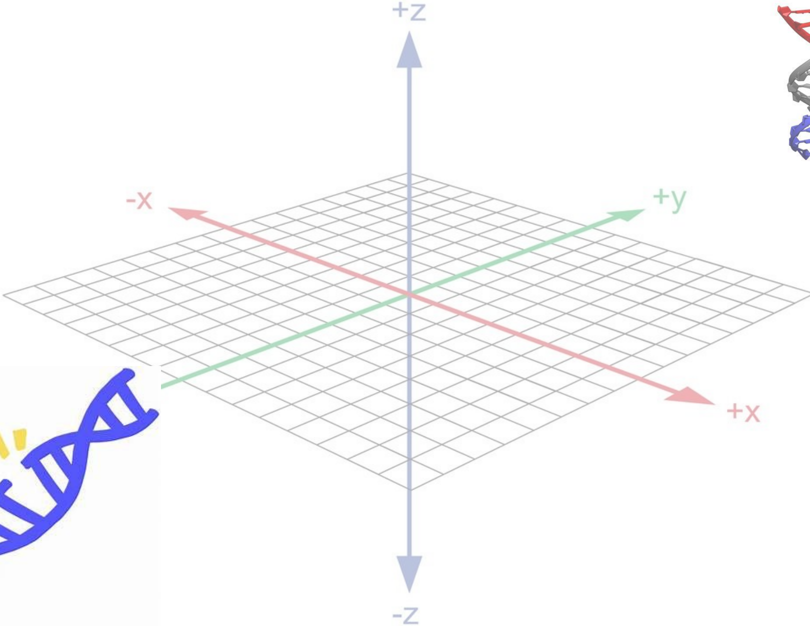
Motivation:

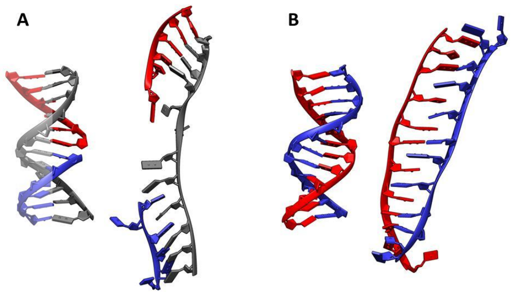
- Techniques such as data aggregation and could reduce the I/O requirements.
- The paradigm might be most effective for applications that generate large volumes of static data.
- Perform SQL-like, Database-like queries on large volumes of data.



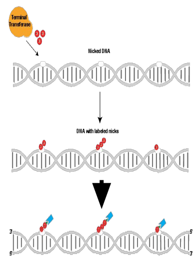
Multidimensional Data Storage with DNA

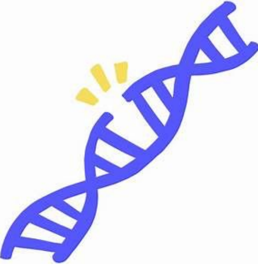
Sequence dimension



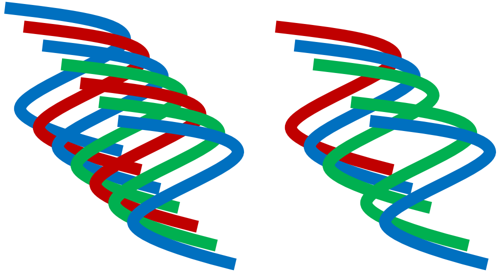


“Random dimension”





Topological dimension



Concentration dimension

Acknowledgement

UNIVERSITY
OF MINNESOTA
Driven to Discover®

UNIVERSITY OF
ILLINOIS
URBANA-CHAMPAIGN

TEXAS
The University of Texas at Austin

Q & A