# Constant-Time Word-Size String-Matching

**D. Breslauer, L. Gasienec, R. Grossi**

# Main points

- New approach to Packed String Matching by reducing the problem to Word-Size String Matching [FSTTCS 2011].

- Efficient Emulation of Word-Size String Matching in the word-RAM model:

  arithmetic, logical, shift operations.

- Some new bit tricks in the word-RAM model.

# String Matching Problem

## INPUT:

pattern $X$ of $m$ symbols in $\Sigma$ [pattern preprocessing]
text $T$ of $n$ symbols in $\Sigma$ [text processing]

## OUTPUT:

positions $i$ s.t. $X = T[i...i+m-1]$

Can we say anything new?

# String Matching Problem

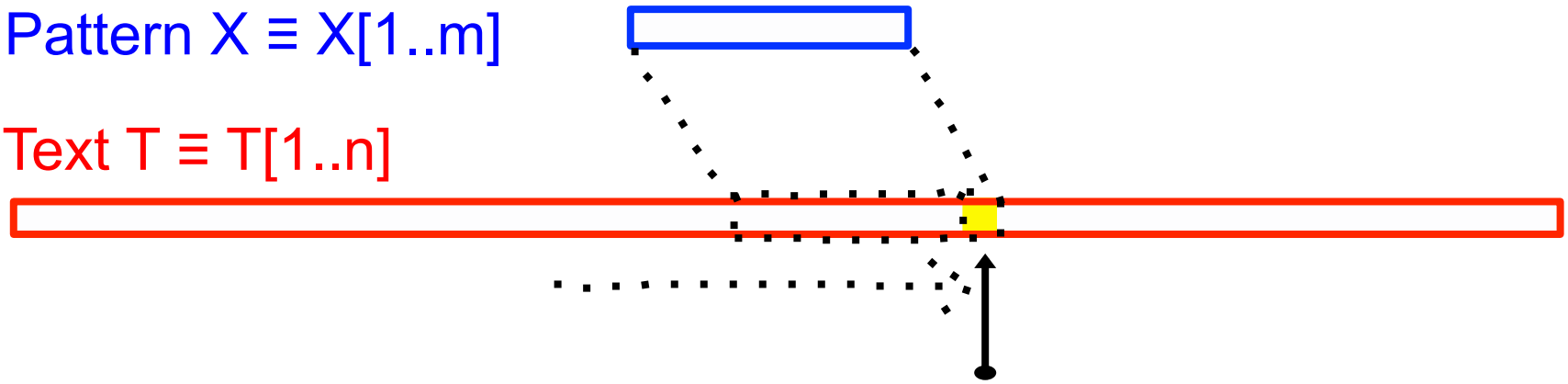
Knuth-Morris-Pratt

Boyer-Moore

Karp-Rabin

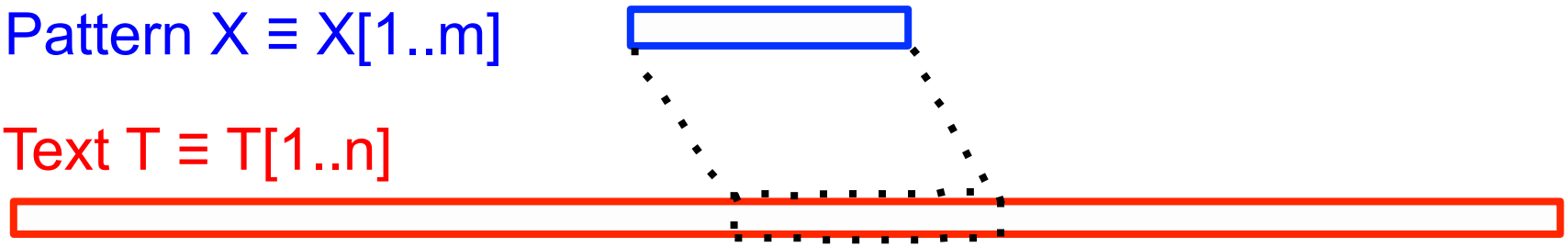
O(m+n) time solution
[over 85 algs in Faro-Lecroq's survey]

# Real-time string matching

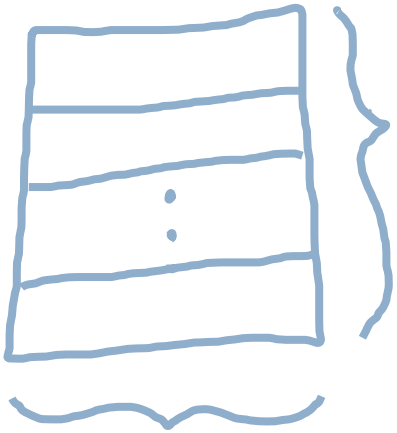



O(1) worst-case time to answer
after reading the text symbol

# Constant-space string matching




(a.k.a. in-place)



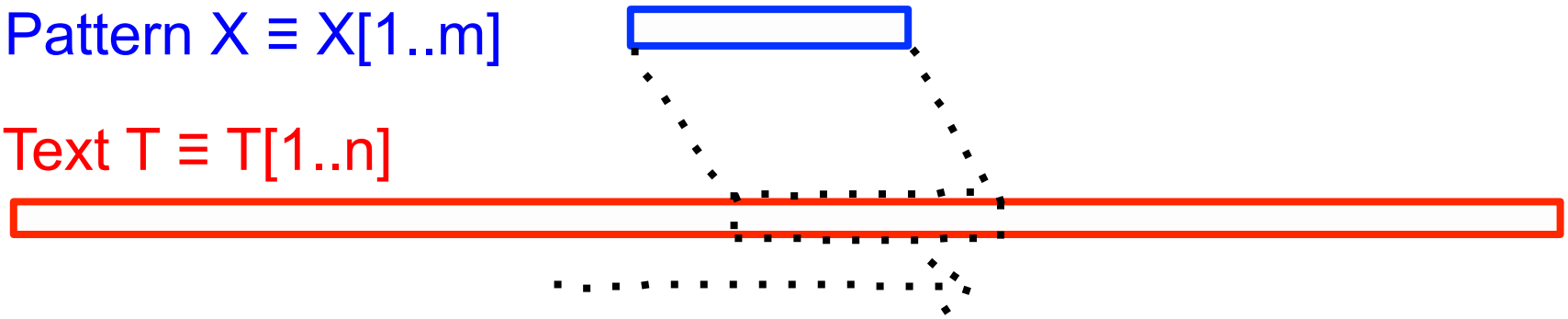

O(1) working space
apart from that
required by X and T

# More related work

- Galil '81: real-time string matching
- Galil, Seiferas '83: constant space
- Karp, Rabin '87: randomized constant space real-time
- Crochemore, Perrin '91: constant space
- Gasieniec, Plandowski, Rytter '95: constant space
- Gasienec, Kolpakov '04: real-time + sublinear space (extends GPR'95)

  more papers [Crochemore, Rytter '91,'95] [Crochemore '92] [...]
- Porat, Porat '09: randomized streaming, O(log m) space, no real-time
- Breslauer, Galil '10: randomized real-time streaming, O(log m) space
- Breslauer, Grossi, Mignosi ‘11: O(1) space, real-time version of Crochemore-Perrin

# Packed String Matching




α symbols packed in a word:
bulk comparison in O(1) time

**Theory:** word-RAM w bits per word

**Your laptop:**
- α characters per word [ $\alpha = w / \log_2 \Sigma = 64/8 = 8$]
- richer instruction set: SSE4.2 & AVX PCMPESTRM

# History of packed string matching

- mentioned in KMP & BM
- several practical approaches (not discussed here)
- New: packed version of Breslauer, Grossi, Mignosi

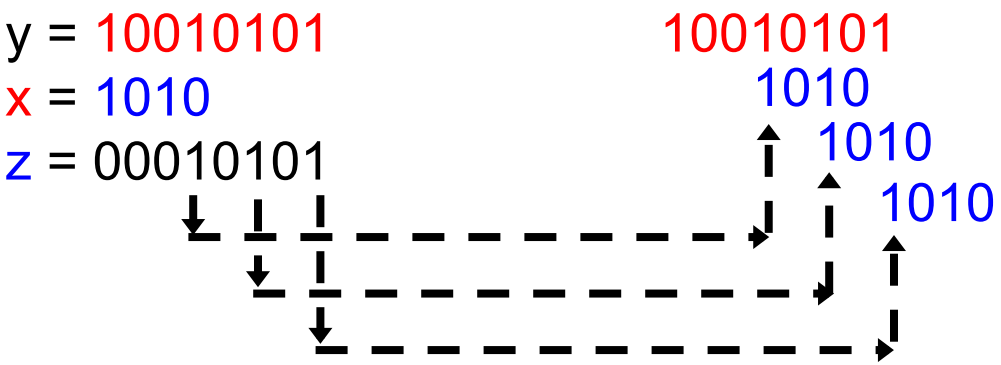
| Time | Space | Reference |
|---|---|---|
| $O(\frac{n}{\log_{\lvert\Sigma\rvert} n} + n^{\varepsilon}m + occ)$ | $O(n^{\varepsilon}m)$ | Fredriksson [24, 25] |
| $O(\frac{n}{\log_{\lvert\Sigma\rvert} n} + m + occ)$ | $O(n^{\varepsilon} + m)$ | Bille [10] |
| $O(\frac{n}{\alpha} + \frac{n}{m} + m + occ)$ | $O(m)$ | Belazzougui [8] |
| $O(\frac{n}{\alpha} + \frac{m}{\alpha} + occ)$ | $O(1)$ | using WSSM and WSLM |

- Ben-Kiki, Bille, Gasieniec, Grossi, Weimann [FSTTCS11]

# Use two special $AC^0$ instructions

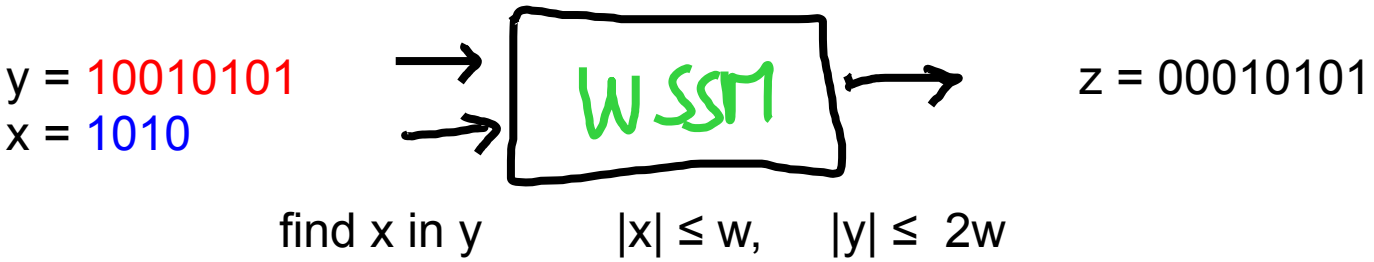
WSSM: word-size string matching [text processing]

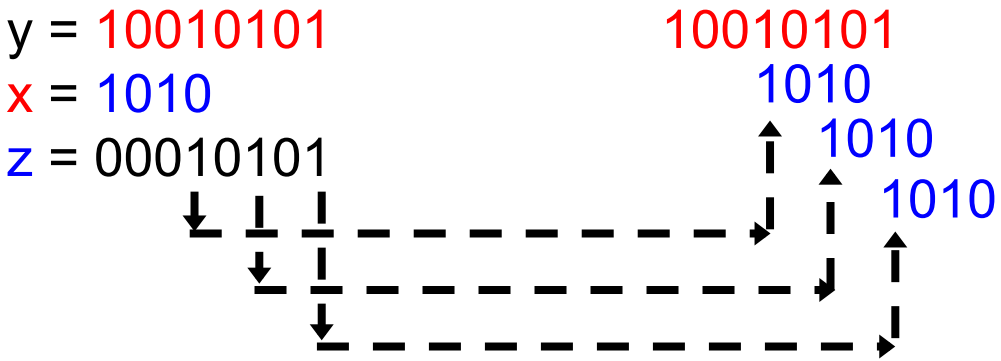
find x in y $|x| \le w$, $|y| \le 2w$




WSLM: word-size lex-max suffix [pattern preprocessing]

# O(1)-time WSSM black box:



# Experiments in SMART [FSTTCS11]

Intel Sandy Bridge, SSE (Streaming SIMD Extension), AVX (Advanced Vector Extension)

SMART (String MAtching Research Tool) by Faro and Lecroq [library of > 85 algorithms]

| 2 | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|
| **SSECP** 4.44 | **SSECP** 4.57 | UFNDMQ4 4.99 | BNDMQ4 4.23 | BNDMQ4 3.83 | LBNDM 3.91 | BNDMQ4 3.71 |
| SKIP 4.80 | RF 5.07 | **SSECP** 5.00 | SBNDMQ4 4.31 | BNDMQ6 3.86 | BNDMQ4 3.94 | HASH5 3.83 |
| SO 4.84 | BM 5.33 | FSBNDM 5.05 | UFNDMQ4 4.31 | SBNDMQ4 3.95 | SBNDMQ4 3.96 | HASH8 3.93 |
| FNDM 4.94 | BNDMQ2 5.46 | SBNDMQ2 5.08 | UFNDMQ6 4.47 | SBNDMQ6 3.97 | BNDMQ6 3.97 | HASH3 3.94 |
| FSBNDM 5.03 | BF 5.58 | BNDMQ2 5.13 | SBNDMQ6 4.57 | UFNDMQ4 4.00 | HASH5 3.98 | BNDMQ6 3.97 |
| | | | 23 **SSECP** 5.00 | 27 **SSECP** 5.29 | 39 **SSECP** 4.88 | 42 **SSECP** 4.73 |
| **SSECP** 4.28 | **SSECP** 4.49 | BNDMQ2 4.42 | SBNDMQ2 4.08 | UFNDMQ2 3.75 | SBNDMQ4 3.67 | BNDMQ4 3.70 |
| FFS 4.88 | SVM1 4.84 | SBNDMQ2 4.48 | UFNDMQ2 4.08 | BNDMQ4 3.79 | BNDMQ4 3.72 | SBNDMQ4 3.71 |
| GRASPM 4.93 | SBNDMQ4 4.85 | SBNDM 4.59 | SBNDMQ4 4.10 | SBNDMQ4 3.80 | UFNDMQ4 3.80 | HASH5 3.75 |
| BR 5.14 | BOM2 4.95 | SBNDM2 4.59 | SBNDM2 4.13 | UFNDMQ4 3.80 | BNDMQ2 3.89 | UFNDMQ4 3.77 |
| BWW 5.14 | EBOM 5.25 | UFNDMQ2 4.69 | BNDMQ2 4.14 | BNDMQ2 3.89 | SBNDM2 3.96 | HASH8 3.80 |
| | | 13 **SSECP** 5.00 | 22 **SSECP** 5.08 | 35 **SSECP** 4.77 | 39 **SSECP** 4.77 | 45 **SSECP** 4.76 |

- SSECP our implementation, performs well for a wide range of parameters
- algorithms that skips characters are faster than SSECP for long patterns

# What if we use the plain word-RAM?

WSSM & WSLM Emulation in the word-RAM

| Time | Space | Emulation |
|---|---|---|
| $O(\frac{n \log \alpha}{\alpha} + occ)$ | $O(1)$ | bit-parallel WSSM no pre-processing |
| $O(\frac{n}{\alpha} + \alpha + occ)$ | $O(\alpha)$ | bit-parallel WSSM pre-processing |
| $O(\frac{m}{\log_{\|\Sigma\|} n})$ | $O(n^\epsilon)$ | four Russian WSLM table lookup |

slowdown factor $\log \alpha$

# Attempts to simulating the WSSM on the word-RAM

- reduce the problem to binary convolution ($AC^0$)
- simulate the convolution using int. multiplication (not $AC^0$) and padding each bit by log α bits
- use deterministic sampling (from parallel random access machine algorithms) to reduce padding to log log α

## Example

Padding the pattern 101 and the text 01101010 (padding bits are in gray)

$$p = 010001, t = 0001010001000100$$

$$\overline{p} = 000100, \overline{t} = 0100001000100001$$

Doing standard integer multiplication on these vectors we get that:

$$p \times \overline{t} = 1000101001000100001$$

$$\overline{p} \times t = 0000101000100010000$$

Adding these we get the mismatch vector:

$$(p \times \overline{t}) + (\overline{p} \times t) = 1\ 00\ 10\ 10\ 00\ 11\ 00\ 11\ 00\ 01$$

Replacing each field (two bits) by the number it holds gives:

$$(p \times \overline{t}) + (\overline{p} \times t) = 1022030301$$

Taking the $n = 8$ least significant bits gives the mismatch vector 22030301

# Exploit PRAM algorithms!!!

- Galil '85: O(log n) optimal PRAM
- Vishkin ‘85: O(log n) WITNESSES eliminate alphabet dependence
- Breslauer, Galil ‘89: O(log log n) optimal
- Vishking ‘90: O(log* n) optimal – DETERMINISTIC SAMPLES slower preprocessing
- Breslauer, Galil ’91: ῼ(log log n) lower bound
- Galil ’92: O(1) time – slower pattern preprocessing
- Goldberg and Zwick ‘94: better preprocessing
- Cole, Crochemore, Galil, Gasieniec, Hariharan, Muthukrishnan, Park, Rytter ‘93: higher dimension better preprocesing O(1) randomized
- Czumaj, Galil, Gasieniec, Park, Plandowski ’95: EREW & CREW
- Crochemore, Galil, Gasieniec, Park, Rytter ‘97:. O(1) randomized

# Our New Ideas: Witnesses, Samples, Slobs

Slob z is a prefix of the pattern x:

1. za and zb appear in x, a≠b
2. za only occurs as prefix in first period

x = 10100101 period(x) = 5

za = **101**

zb = **100**

# Slobs: simple parallel algorithm

1. Initially n candidate occurrences in text y
2. Compare to {a,b} to get n/(|z|+1) candidates
3. Compare to za to get n/period candidates
4. Compare to period and count periods

x = 10100101 period(x) = 5

y = 01100100100101001010

# Slobs: simple parallel algorithm

1. Initially n candidate occurrences in text y
2. Compare to {a,b} to get n/(|z|+1) candidates
3. Compare to za to get n/period candidates
4. Compare to period and count periods

x = 10100101 period(x) = 5

y = 01100100100101001010

# Slobs: simple parallel algorithm

1. Initially n candidate occurrences in text y
2. Compare to {a,b} to get n/(|z|+1) candidates
3. Compare to za to get n/period candidates
4. Compare to period and count periods

x = 10100101 period(x) = 5

y = 0110010010010100101

# Slobs: simple parallel algorithm

1. Initially n candidate occurrences in text y
2. Compare to {a,b} to get n/(|z|+1) candidates
3. Compare to za to get n/period candidates
4. Compare to period and count periods

x = **10**1**0**0101 period(x) = 5

y = 011001001**10**100**10**1

# Slobs: simple parallel algorithm

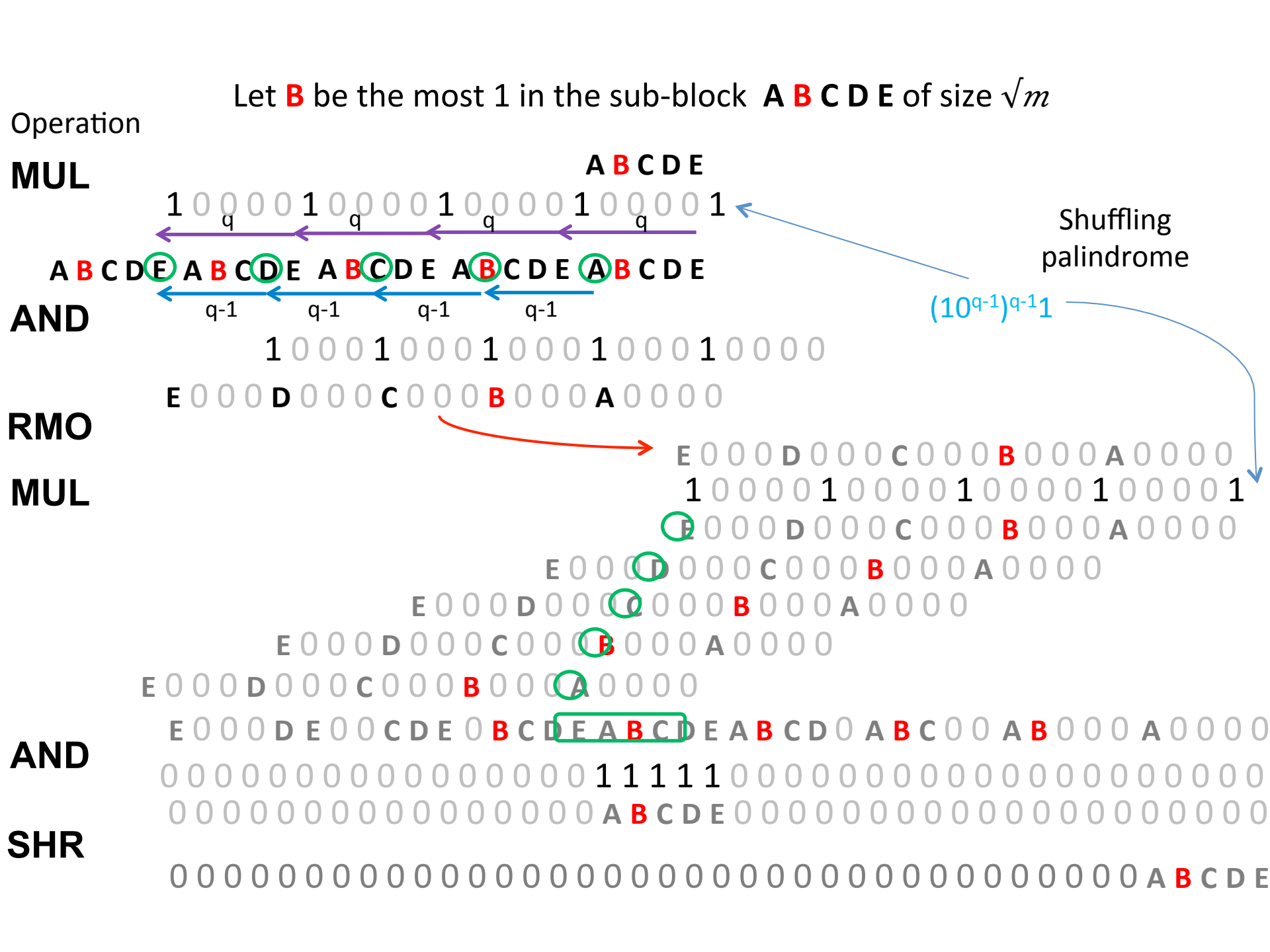
1. Initially n candidate occurrences in text y
2. Compare to {a,b} to get n/(|z|+1) candidates
3. Compare to za to get n/period candidates
4. Compare to period and count periods

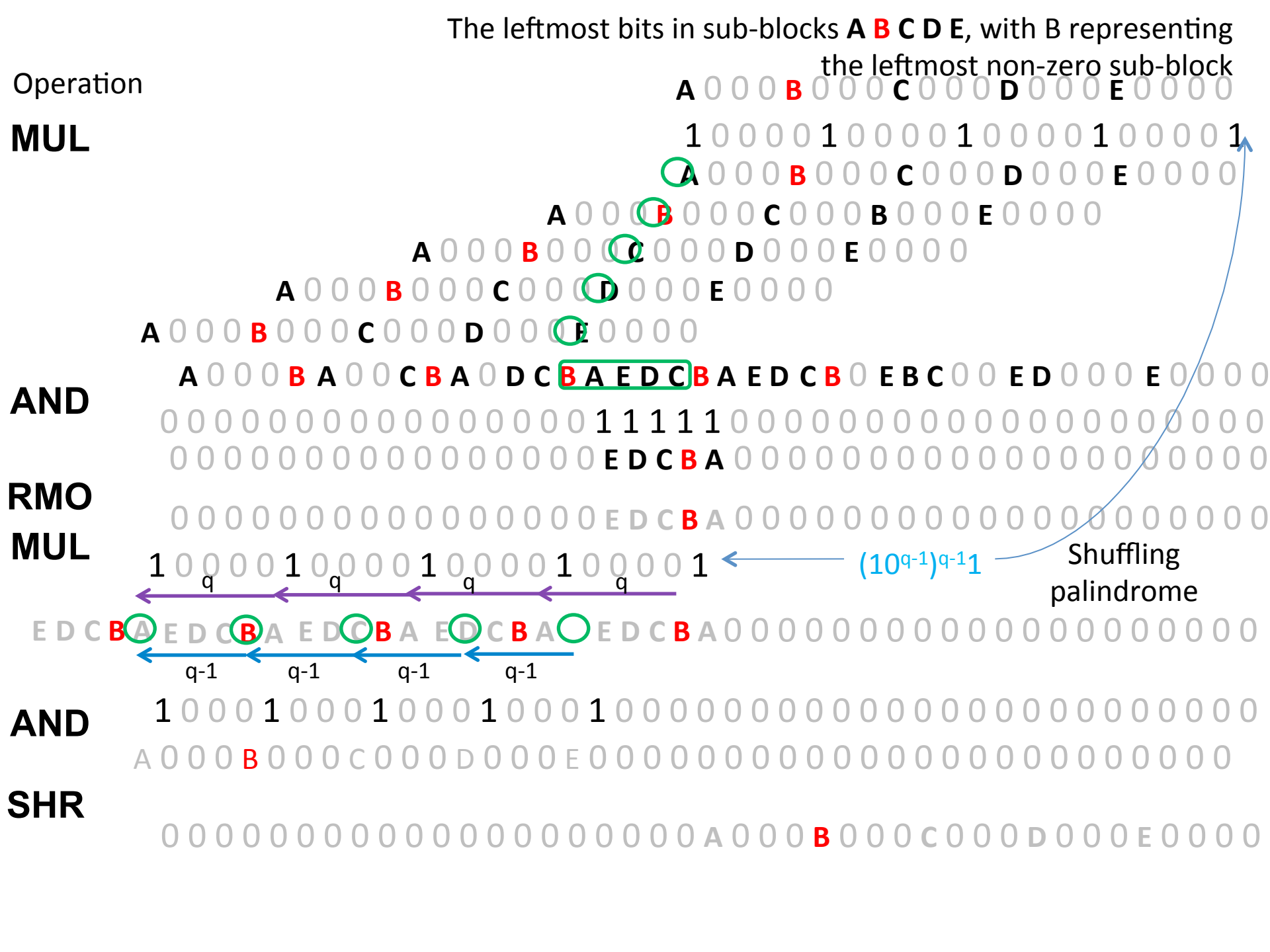
x = **10**1**0**0101 period(x) = 5

y = 011001001001**0**1**0**0101

# Bitwise Tricks and Techniques

- New chapter in Knuth's book published 2011
- Addition carry bits flow in one direction: easy to find rightmost least significant bits, harder to find leftmost most significant bits
- We have tricks to find most significant bits as well ☺


Let B be the most 1 in the sub-block A B C D E of size $\sqrt{m}$
Operation
MUL
AND
RMO
MUL
AND
SHR
Shuffling palindrome
$(10^{q-1})^{q-1}1$


The leftmost bits in sub-blocks A B C D E, with B representing the leftmost non-zero sub-block
Operation
MUL
AND
RMO
MUL
AND
SHR
q
q-1
$(10^{q-1})^{q-1}1$
Shuffling palindrome

# Further work

- Preprocessing
  - Packed preprocessing (WSLM): lex-max suffix
  - Word-size preprocessing (small word size, time)
  - Parallel random access machine preprocessing (constant size alphabet)

Questions?