Locating All Maximal Approximate Runs in a String

Mika Amit, Maxime Crochemore and Gad M. Landau

CPM 2013, Germany

Exact Runs

- An exact run is a non-empty substring of T, that can be divided into identical adjacent non-overlapping substrings: $u^t \cdot u_1$

$$T = \ldots\ a\ b\ c\ a\ b\ c\ a\ b\ c\ a\ b\ c\ a\ b\ \ldots$$

u = abc
(period)

u_1=ab

period length = $|u|$ = p

Exact runs

- An exact run is a non-empty substring of T, that can be divided into identical adjacent non-overlapping substrings: $u^t \cdot u_1$

$$T = \ldots\ a\ b\ c\ a\ b\ c\ a\ b\ c\ a\ b\ c\ a\ b\ \ldots$$

period = abc

- A maximal exact run is an exact run that cannot be extended

Related Work

- Given a string T of length n, detecting all occurrences of exact runs in T can be done in $O(n \log n)$-time
  - Crochemore, 1981
  - Apostolico & Preparata, 1983
  - Main & Lorentz, 1984

RELATED WORK

- Given a string T of length n, detecting all exact runs in T can be done in $O(n)$-time
  - Crochemore, 1983
  - Main, 1989
  - Kolpakov and Kucherov, 1999

Approximate Runs

- Finding approximate runs is sometimes more sensible than finding exact runs
- There are many different measurements and definitions of approximate runs
- The difference is in measuring the differences between the periods

Related Work

- Landau and Schmidt, 1993
- Sim, Iliopoulos, Park and Smyth, 1999
- Landau, Schmidt and Sokol, 2001
- Kolpakov and Kucherov, 2001
- Amir, Eisenberg and Levy, 2012
- ...

Approximate Runs - V.1

- Given a string T and a number k,
find a substring of T that can be divided into adjacent non-overlapping substrings: $u^{t} \cdot u_1$,
such that the difference between every two **adjacent** periods does not exceed k

T = ... a b d a b c a d c d d c d e c ...

1-MAR

(Maximal Approximate Run)

(4-MAR)

Approximate Runs - v.2

- Given a string T and a number k,
find a substring of T that can be divided into adjacent non-overlapping substrings: $u^t \cdot u_1$,
such that the difference between **every** two periods does not exceed k

T = ... a b d a b c a d c d b c d b c ...

2-MAR

Approximate Runs - v.3

- Given a string T and a number k, find a substring of T that can be divided into adjacent non-overlapping substrings: $u^t \cdot u_1$, such that the difference between each period and a **consensus** substring does not exceed k

T = ... a b d a b c a d c a d c d b c ...

1-MAR

u = abc

(4-MAR)

Approximate Runs - our version

- Given a string T and a number k,
  find a substring of T that can be divided into adjacent non-overlapping substrings: $u^t \cdot u_1$,
  such that the **sum** of all differences between each period and a **consensus** substring $\leq k$

T = ... a b c a b c a b c a b c a b c ...

3-MAR

u = abc

modified letters

Mismatch vs. Modified Letter

T = …a b d a b d a b c a b d a b d…

mismatch   mismatch

Mismatch vs. Modified Letter

2 mismatches → 1 modified letter

T = …a b d a b d a b $\boxed{\text{d}}$ a b d a b d…

mismatch

Mismatch vs. Modified Letter

2 mismatches ⇨ 1 modified letter

1 mismatch ⇨ $\frac{n}{2p}$ modified letters

$T = \ldots a\ b\ d\ a\ b\ d\ a\ b\ c\ a\ b\ c\ a\ b \ldots$

mismatch

Mismatch vs. Modified Letter

2 mismatches $\Rightarrow$ 1 modified letter

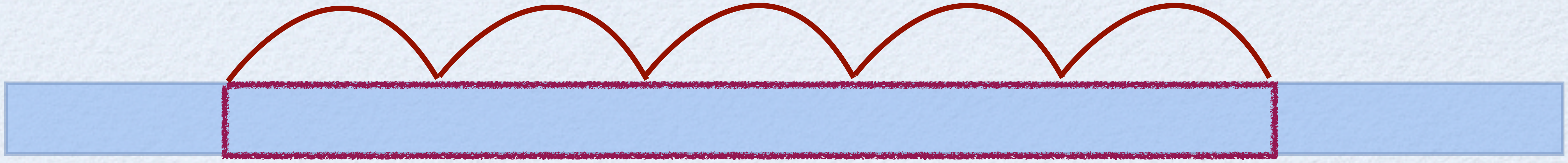
1 mismatch $\Rightarrow$ $n/_{2p}$ modified letters

T = … a b d a b d a b $\boxed{d}$ a b $\boxed{d}$ a b …

Approximate Runs - Problem Definition

Input: a string T, of length n, and a number k

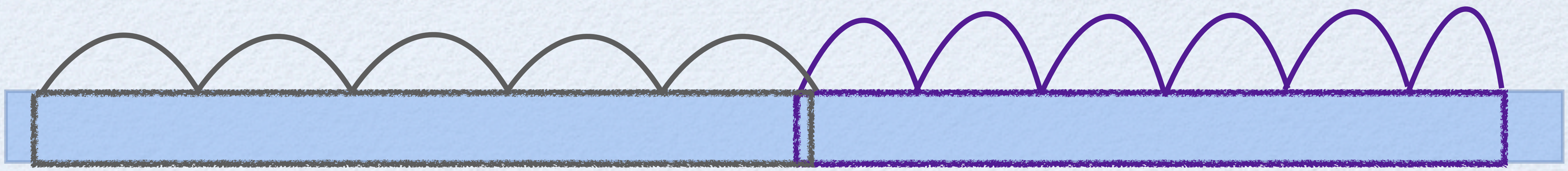
Output: for each period size $1 \leq p \leq n/2$,
find all occurrences of k-MARs with period length p in T



Approximate Runs - Problem Definition

Input: a string T, of length n, and a number k

Output: for each period size $1 \leq p \leq n/2$,
find all occurrences of k-MARs with period length p in T



Locating all k-mars algorithms

- An $O(n^2)$ algorithm
  (uses Parikh matrix)
- An improved $O(n \log n k^3)$ algorithm
  (uses Main & Lorentz ('84) technique)
- An efficient $O(n \log n k^2 \log k)$ algorithm

AN O(N²) ALGORITHM

- Using Parikh matrix (p=3, k=2)

T = a b d a b c a b d a c d a c …

i

|Σ|

a	1			2			2			3				
b		1			2		1	3		1	3			
c			1			1			1		1			
d						1			2					
Win	a	b	c	a	b	c	a	b	d	a	b			

“winner” letter

p

#modified letters =

AN O(N²) ALGORITHM

- Using Parikh matrix (p=3, k=2)

T = a b d a b d a b d a b d a c …

l r

a	1			2			2			3		3	3		
b		1			2		1			1	3		1	2	
c			1			1			1		1			2	
d						1			2						
Win	a	b	c	a	b	c	a	b	d	a	b	d	a	b	

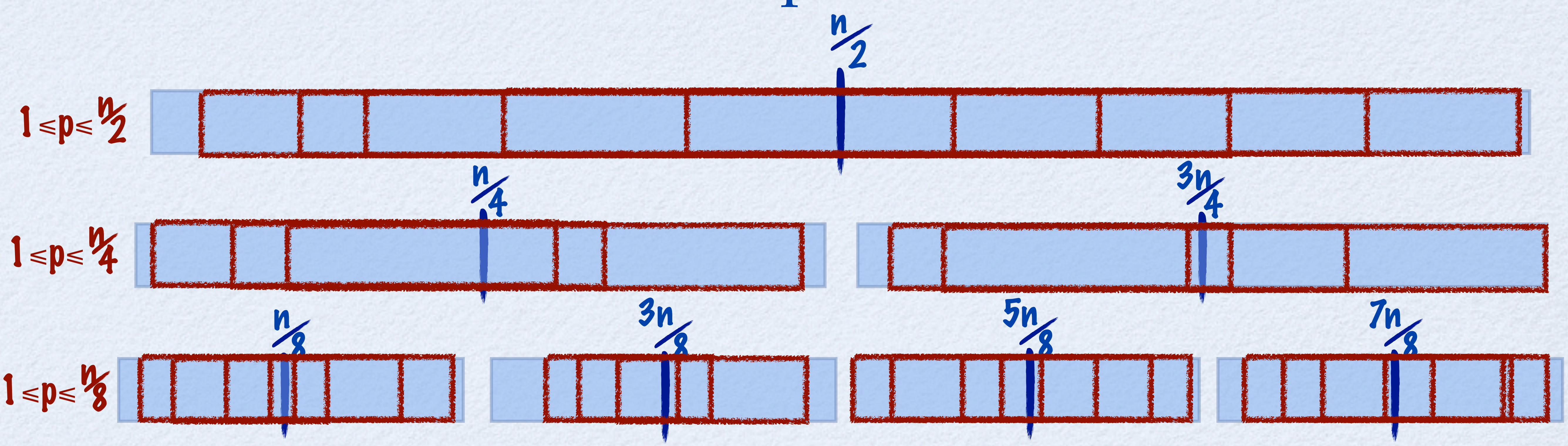
#modified letters = 2

AN $O(N^2)$ ALGORITHM

- Time complexity using Parikh matrix:
  - for each period size $1 \leqslant p \leqslant n/2$
  - find all k-MARs of size p in $O(n)$ time
  - **Total**: $O(n^2)$ time (for constant alphabet)

AN IMPROVED ALGORITHM

- Divide and Conquer (Main & Lorentz):
  - for each period size **p**
  - find all k-MARs with period size **p** that contain the middle position



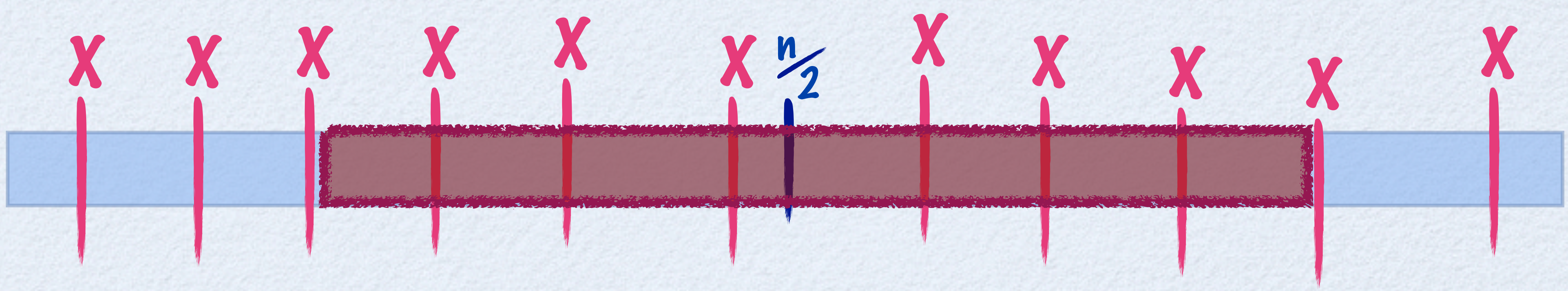
Some observations..

2 mismatches ⇒ 1 modified letter

T = ... a b d a b d a b d a b d a b d ...

Observation 1:

A **k-MAR** can contain at most **2k+1** mismatches

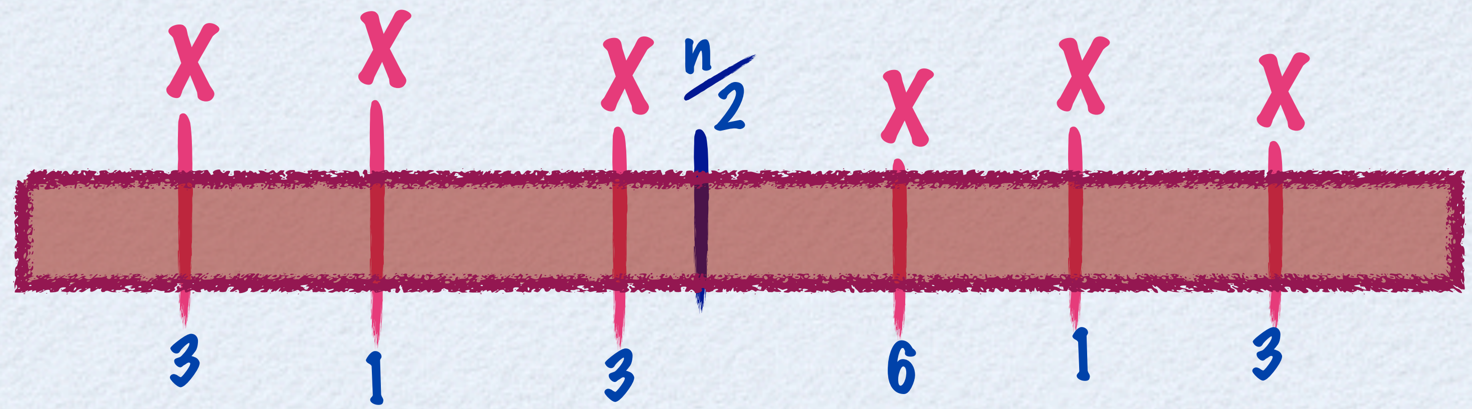


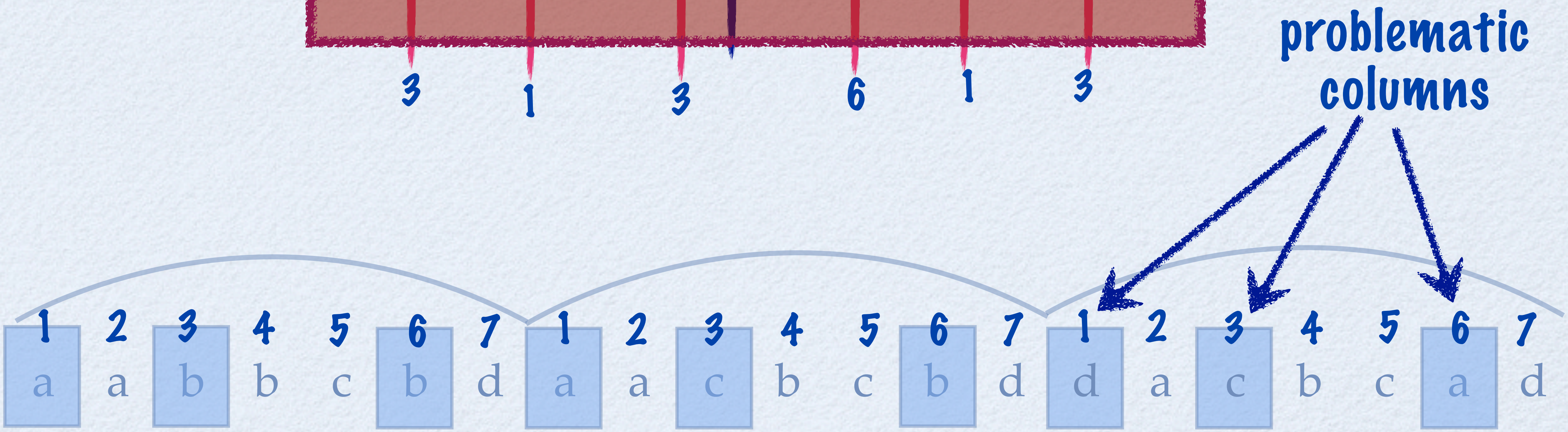
Some observations..

<u>Observation 2:</u>

A **k-MAR** can contain at most **O(k)** problematic columns for each period size

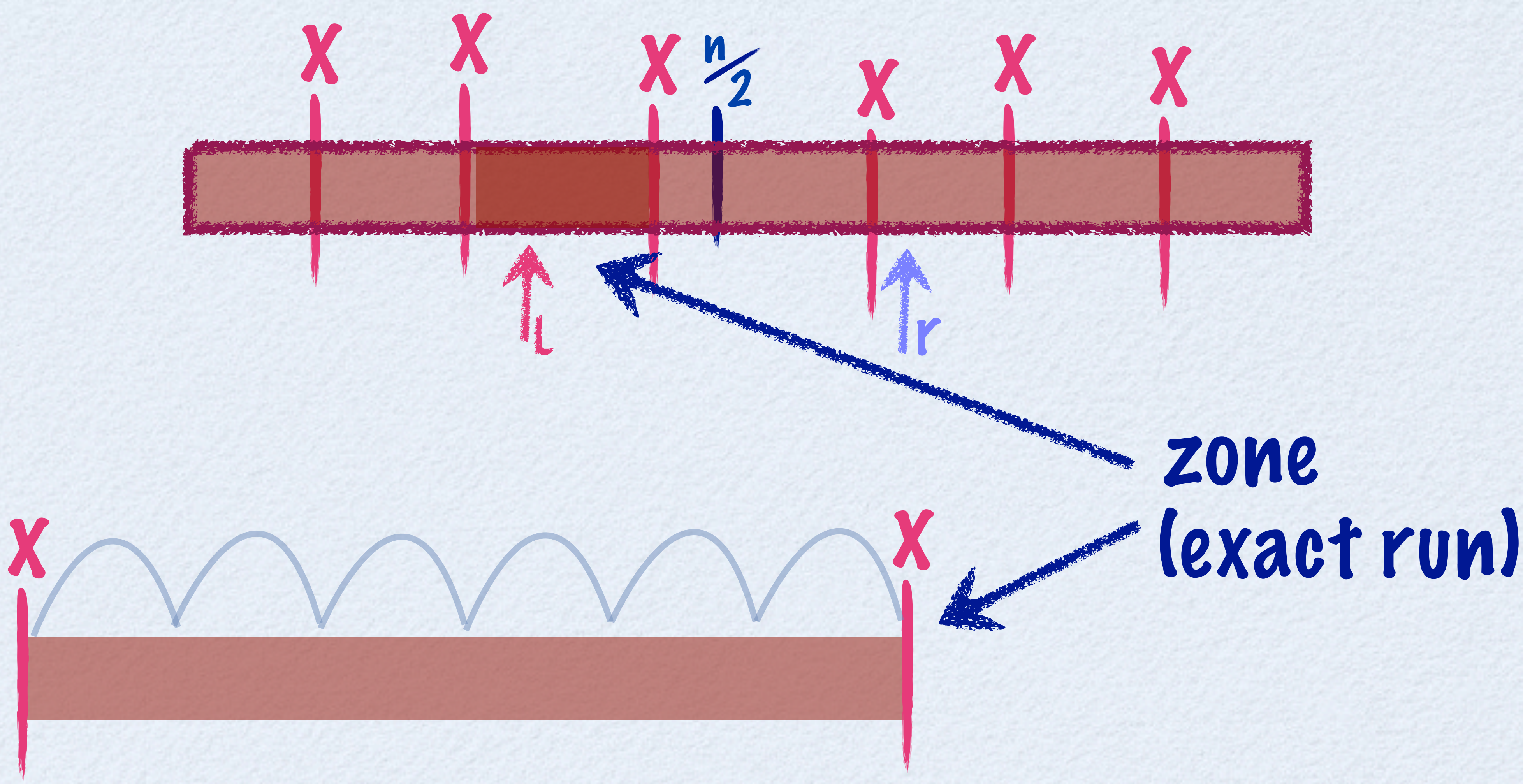
p=7, k=1





There are O(k) problematic columns

Some observations..

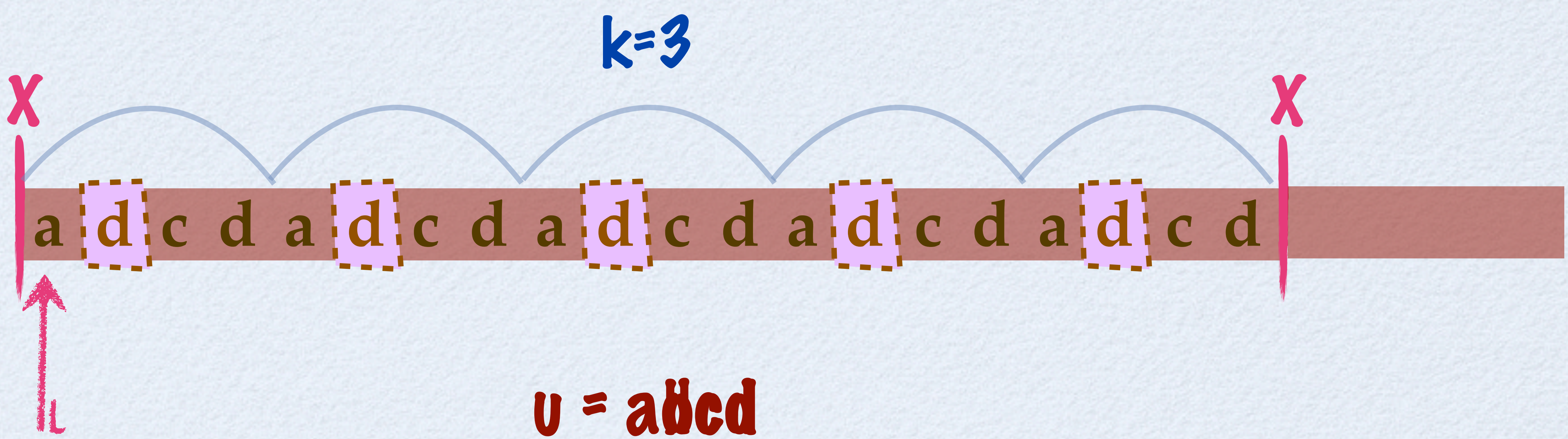


There are O(k) zones

Some observations..

<u>Observation 3:</u>

A **k-MAR** can either start on the **leftmost position** of a zone or on the **rightmost k+1 periods** of it

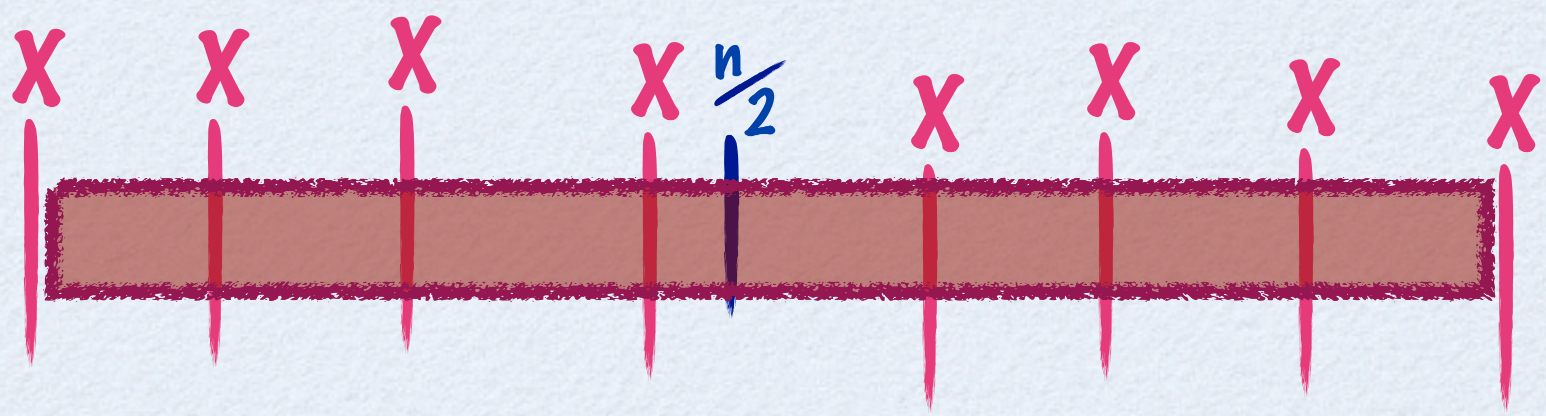


On each zone there are at most $O(k^2)$ positions to visit

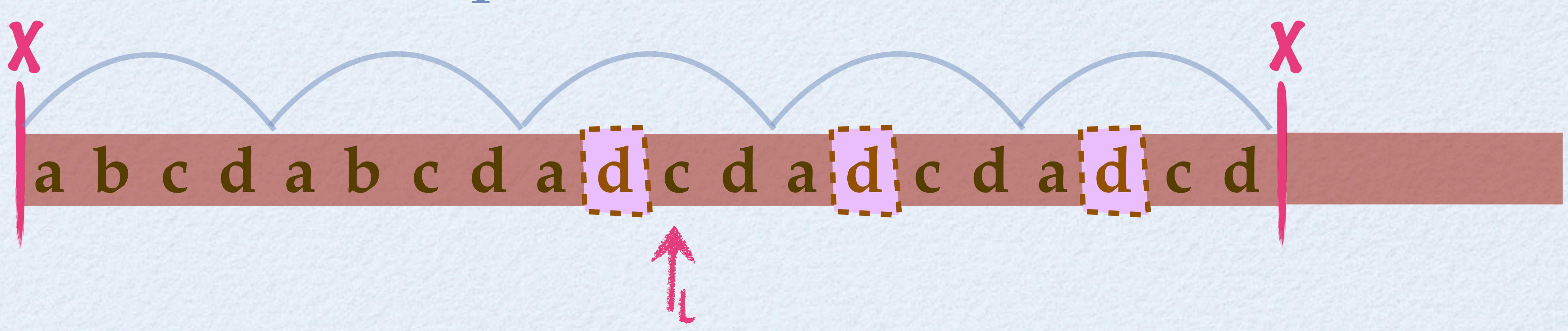
AN IMPROVED ALGORITHM

Given a period length **p**, and a text **T**:

* Find **4k+2** mismatch positions:



* On each zone visit *only* problematic columns in the last **k+1** periods



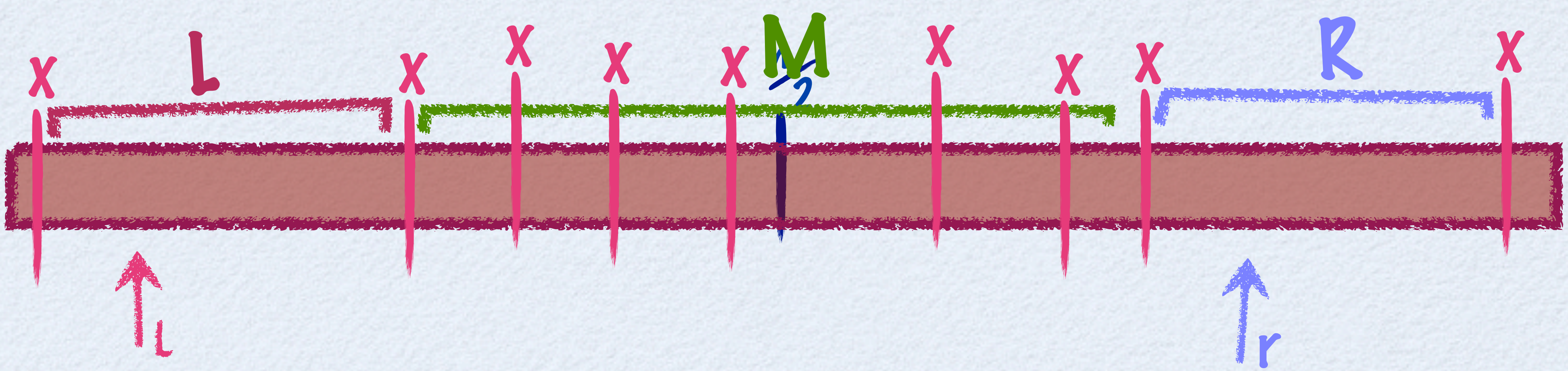
AN IMPROVED ALGORITHM

- Time complexity of iteration 1:
  - for all period length $1 \leq p \leq n/2$
  - find all **k-MARs** that contain position $n/2$
    1. find $O(k)$ mismatch positions in $O(k)$ time
    2. for each one of the $O(k)$ zones:
       visit only $O(k^2)$ positions in $O(k^3)$ time
- Total of $O(nk^3)$ for iteration 1
- **Total** of $O(n\log nk^3)$ time for the entire algorithm

The Efficient Algorithm

Observation:

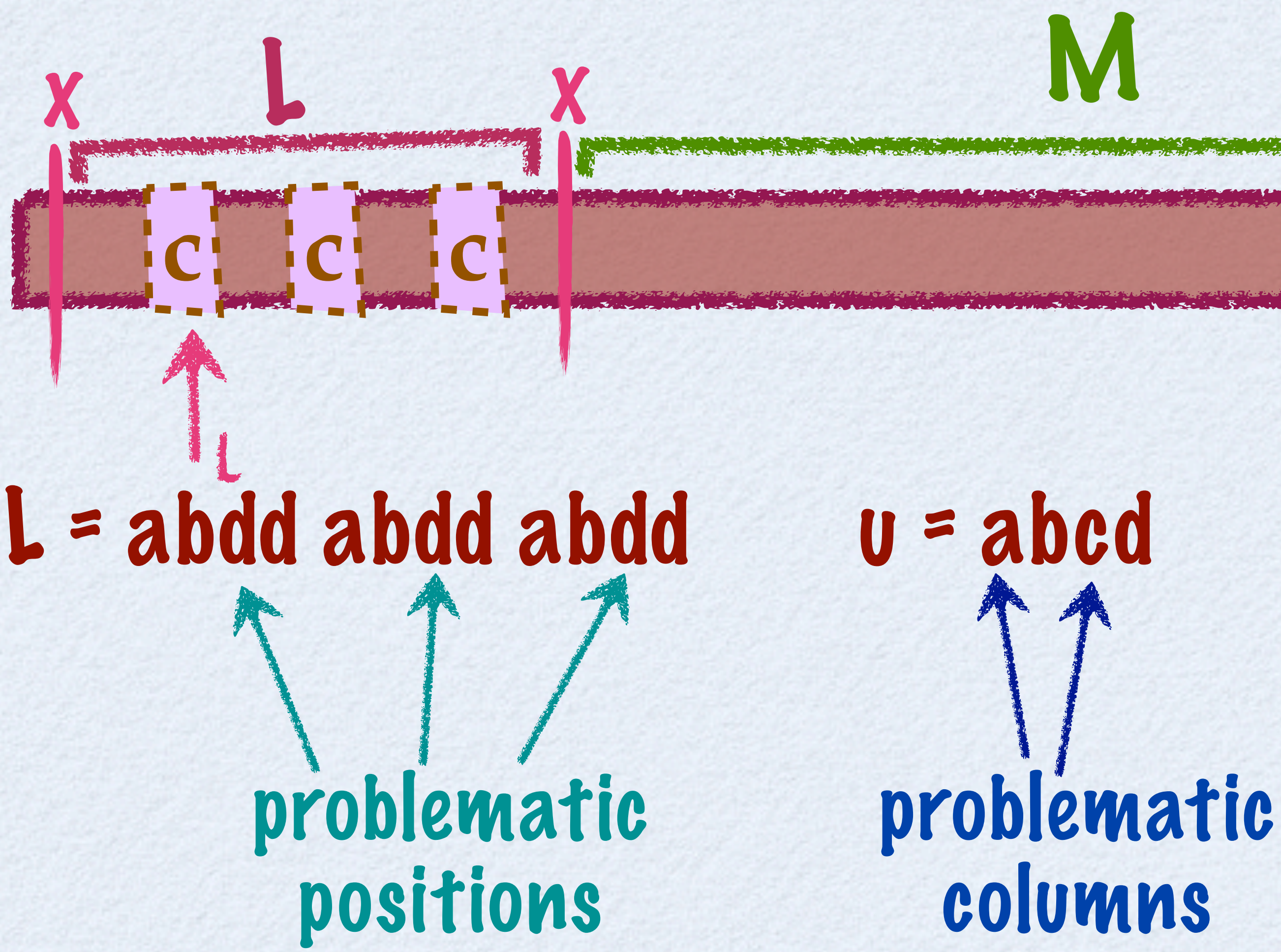
Not all **$O(k^2)$** positions in a zone need to be visited



THE EFFICIENT ALGORITHM

Observation 3:

Not all **$O(k^2)$** positions in a zone need to be visited

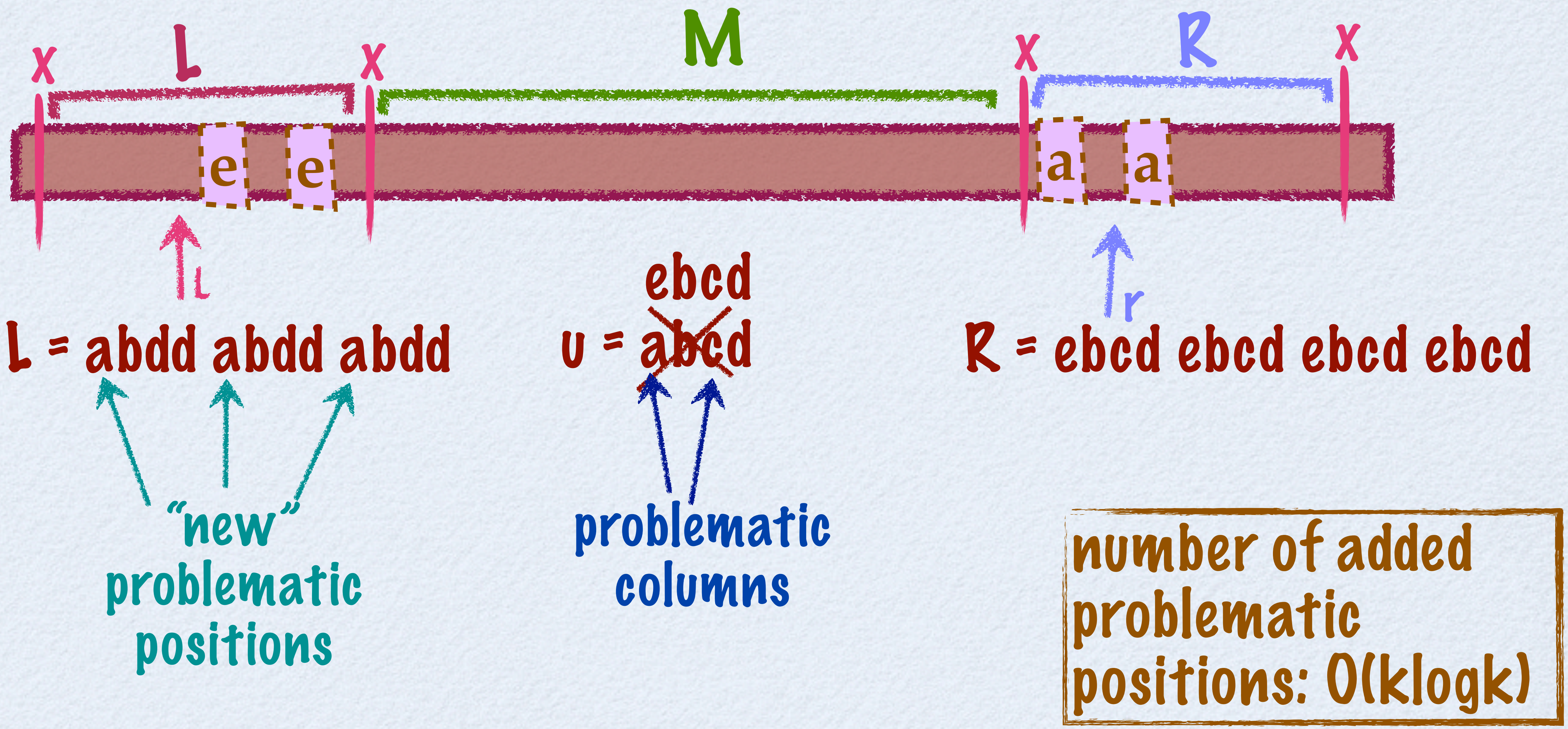


initial number of problematic positions: $O(k)$

THE EFFICIENT ALGORITHM

Observation 3:

Not all **O(k²)** positions in a zone need to be visited



THE EFFICIENT ALGORITHM

- Time complexity of iteration 1:
  * for all period length $1 \leq p \leq n/2$
  * find all **k-MARs** that contain position $n/2$
    1. find $O(k)$ mismatch positions in $O(k)$ time
    2. for each one of the $O(k)$ zones:
       visit $O(k\log k)$ positions in $O(k^2\log k)$ time
- Total $O(nk^2\log k)$ for iteration 1
- **Total** $O(n\log n k^2\log k)$ time for the entire algorithm

Thank you!