Naive Substring Search
(Algorithms and Data Structures)

Naive Substring Search

- we would like to construct an algorithm thats capable of finding a **P** pattern in a given **S** string or text
- **brute-force search** is the naive approach
- keep iterating through the text and if there is a mismatch we shift the pattern one step to the right

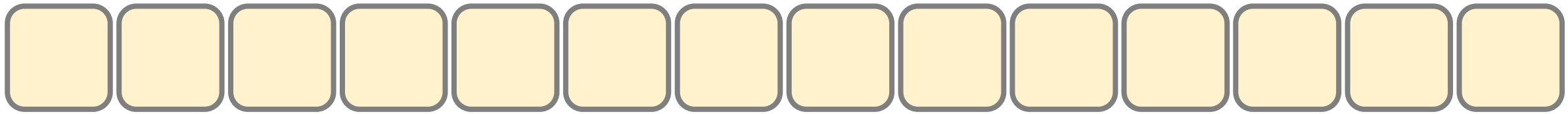
Naive Substring Search

- we would like to construct an algorithm thats capable of finding a **P** pattern in a given **S** string or text
- **brute-force search** is the naive approach
- keep iterating through the text and if there is a mismatch we shift the pattern one step to the right

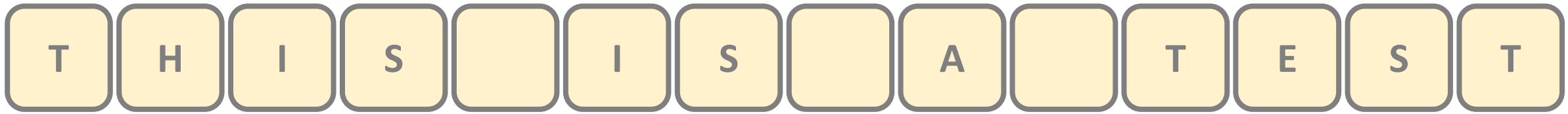
Naive Substring Search

- not so efficient especially when there are lots of matching prefixes
- main problem is that we need backup for every mismatch
- if there is a mismatch we jump back to the next character
- there are lot of comparisons – **O(NM)** where **N** is the length of text and **M** is the length of the pattern
- **linear running time** would be better

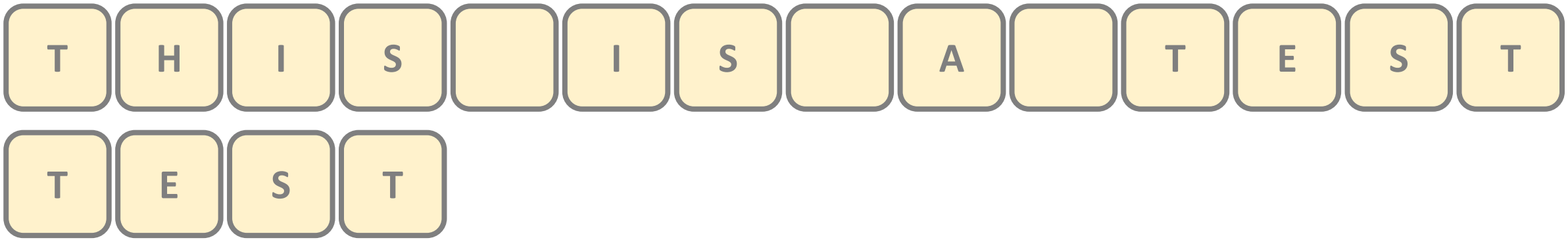
Naive Substring Search



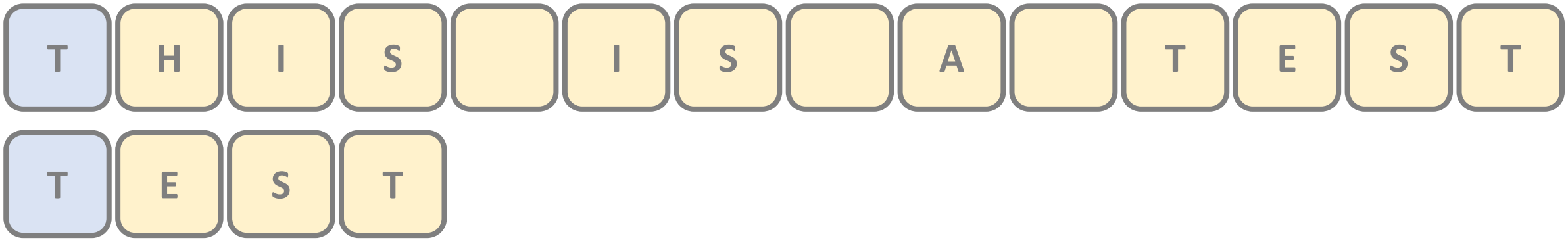
Naive Substring Search



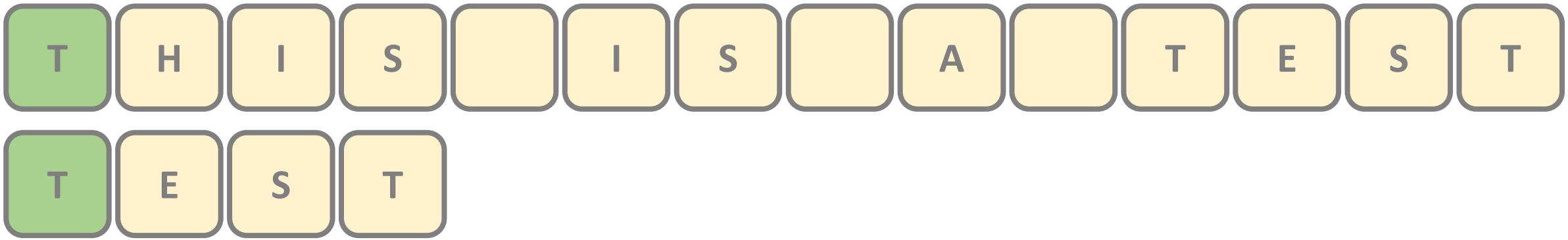
Naive Substring Search



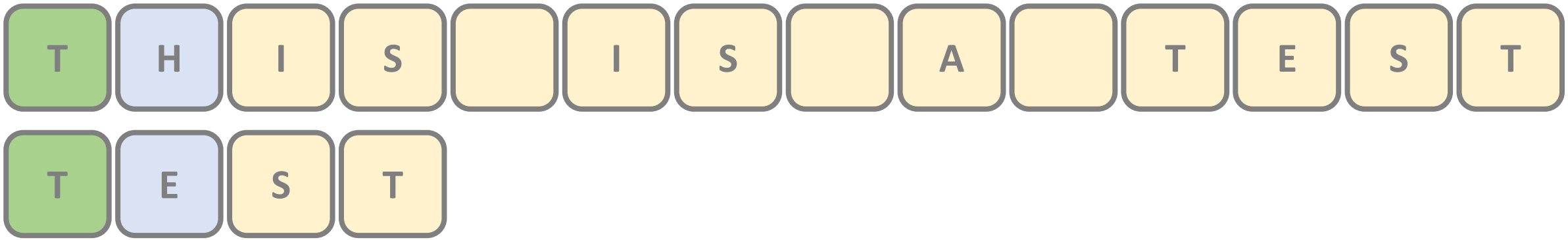
Naive Substring Search



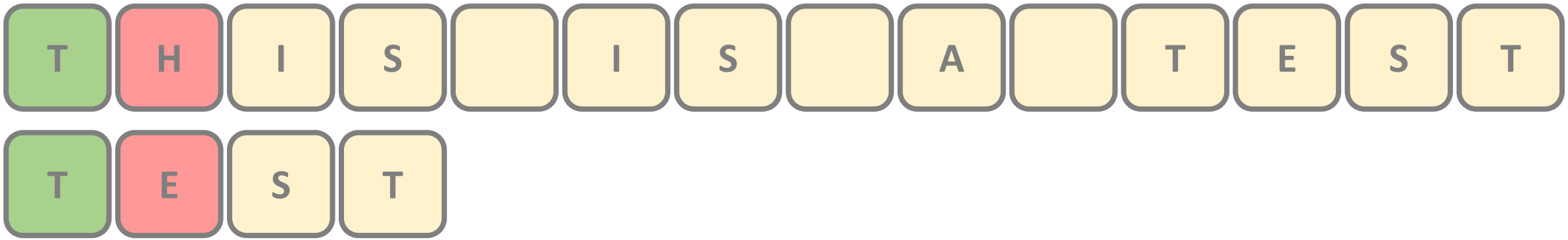
Naive Substring Search



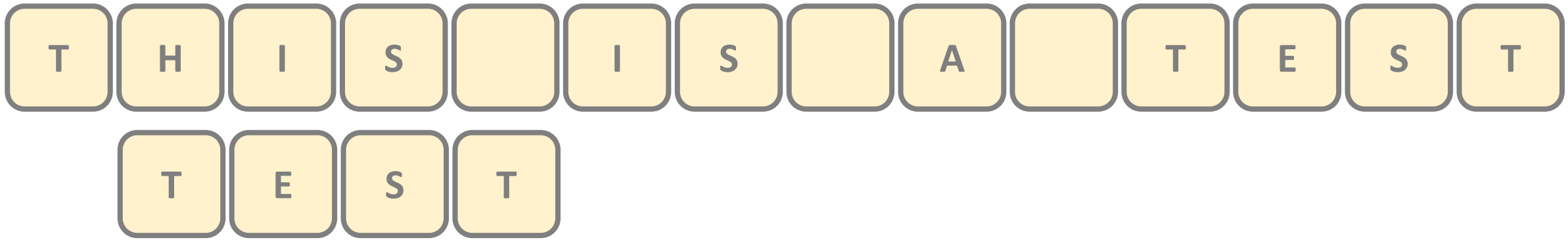
Naive Substring Search



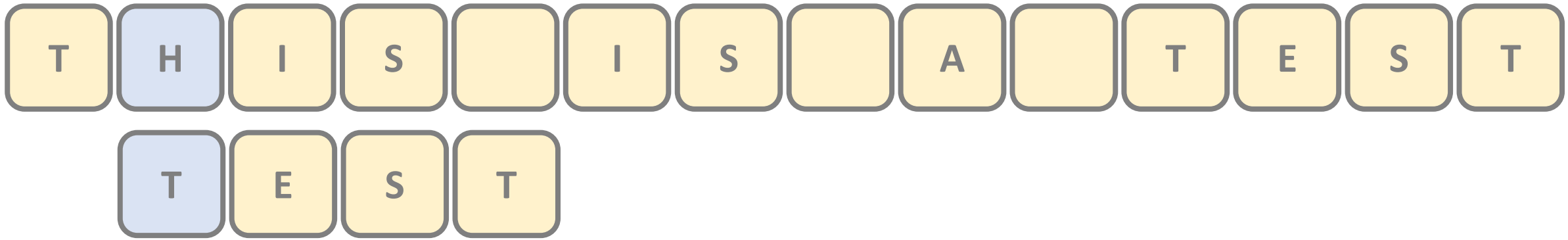
Naive Substring Search



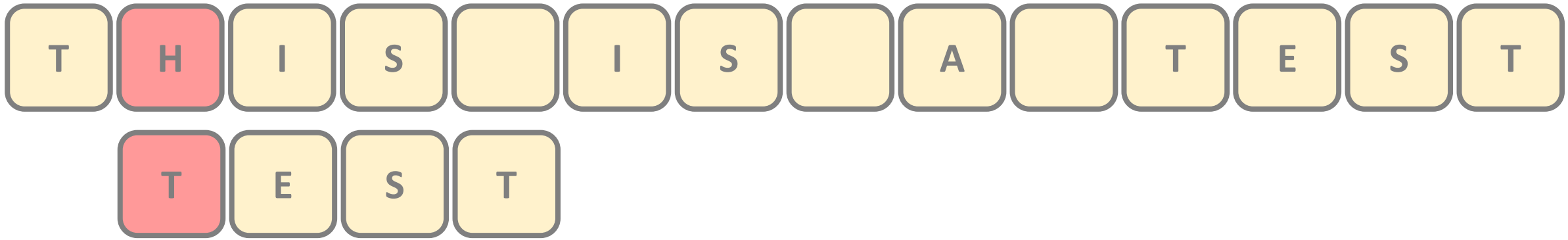
Naive Substring Search



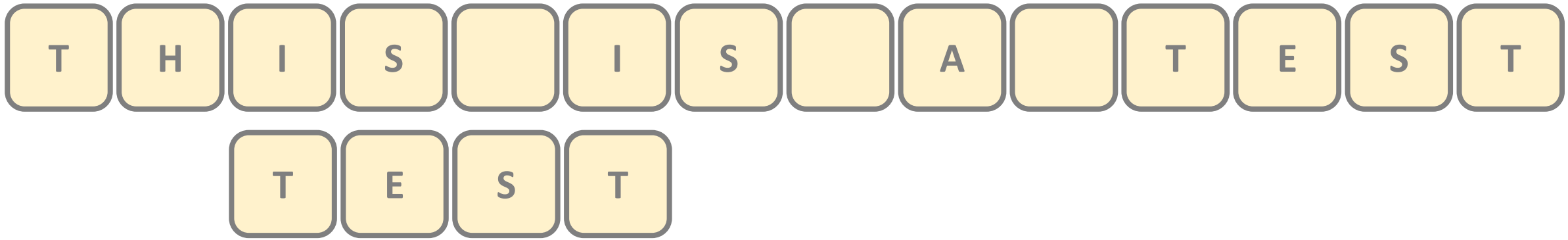
Naive Substring Search



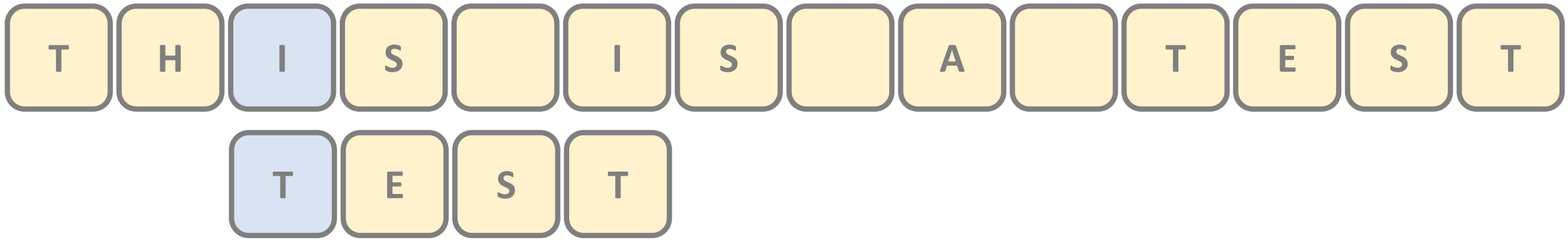
Naive Substring Search



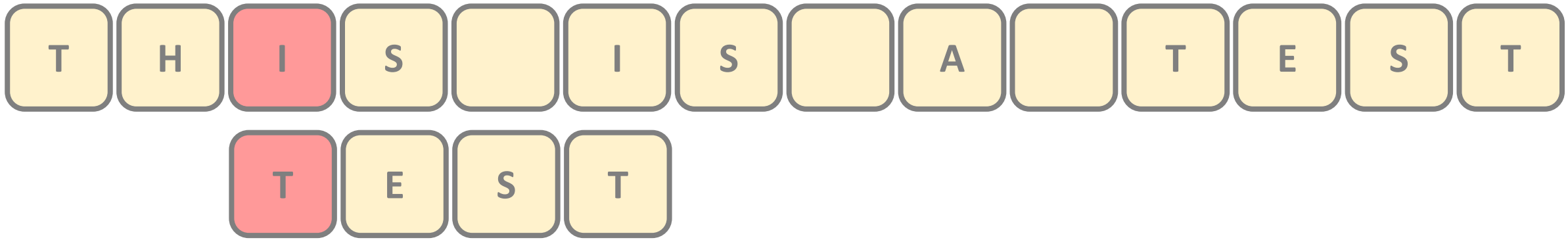
Naive Substring Search



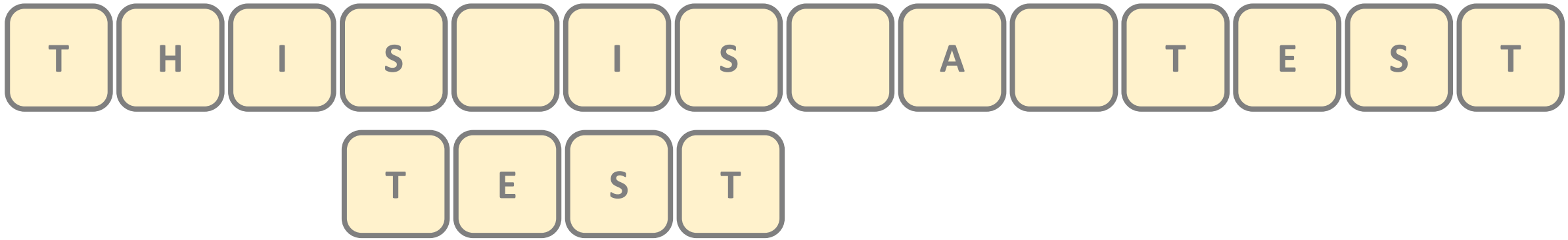
Naive Substring Search



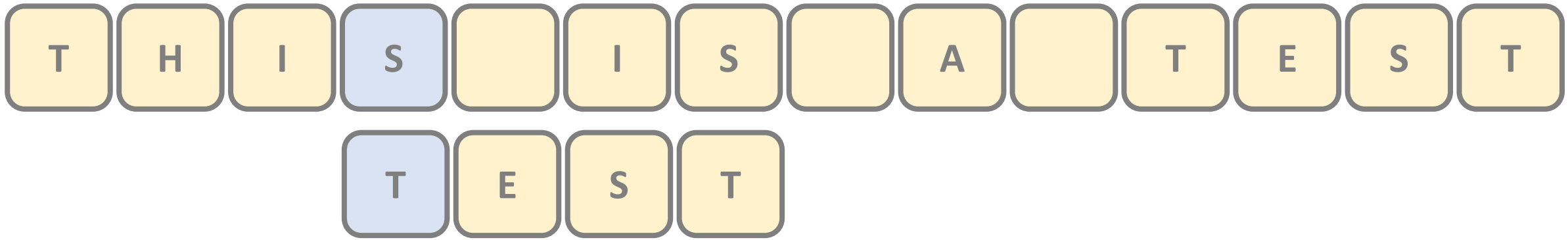
Naive Substring Search



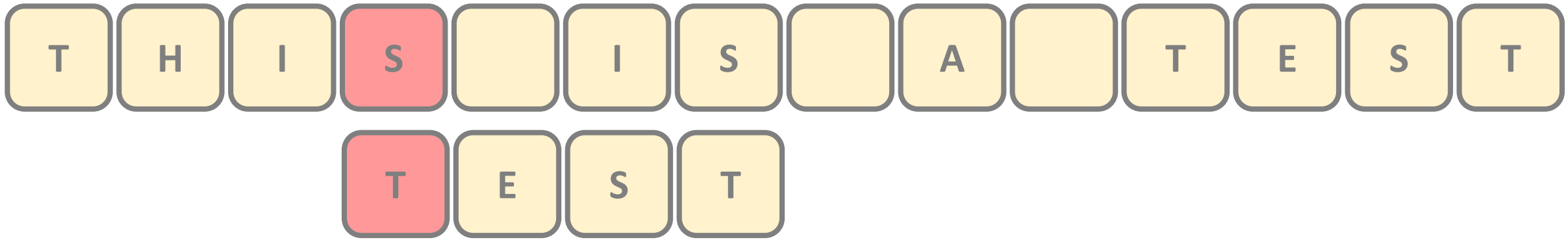
Naive Substring Search



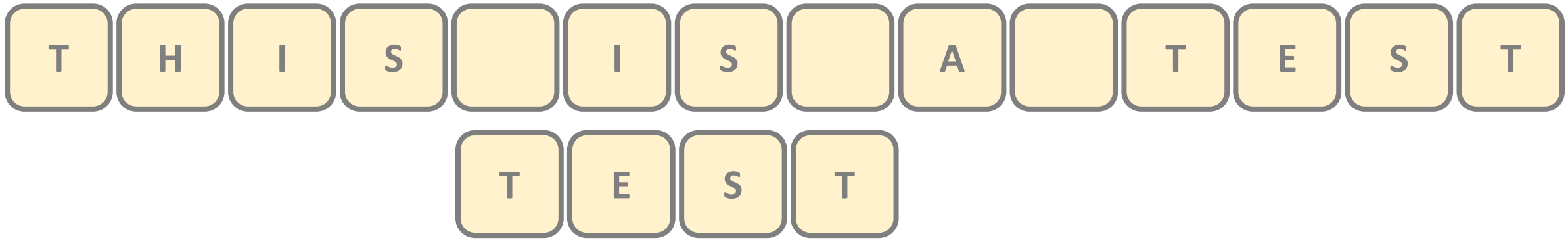
Naive Substring Search



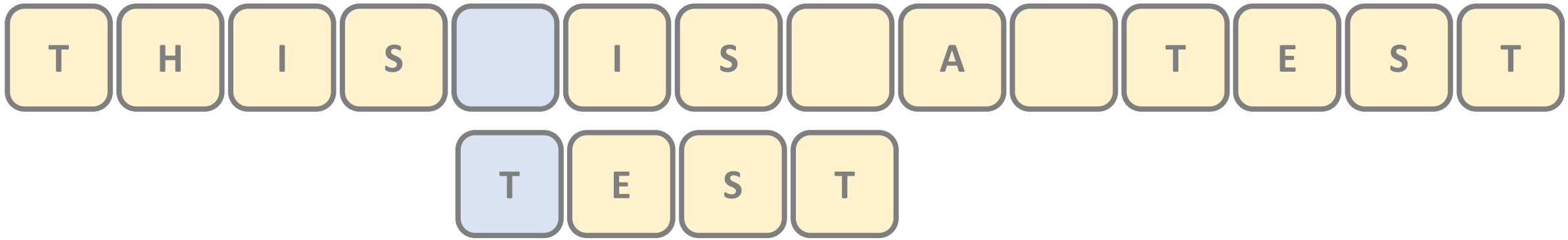
Naive Substring Search



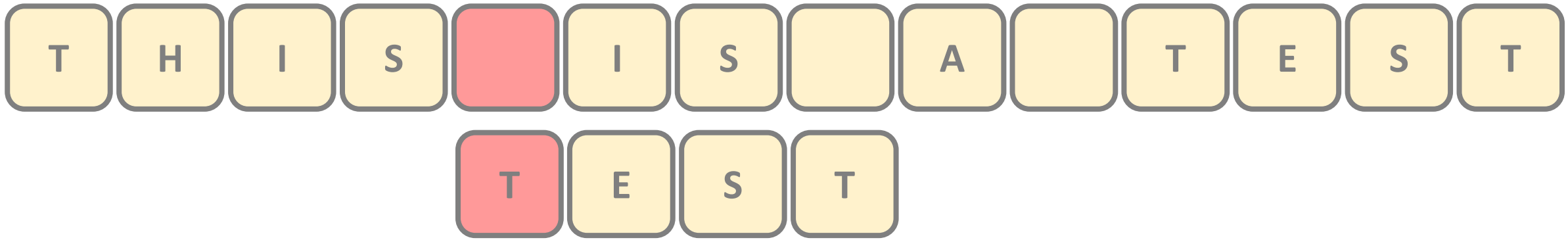
Naive Substring Search



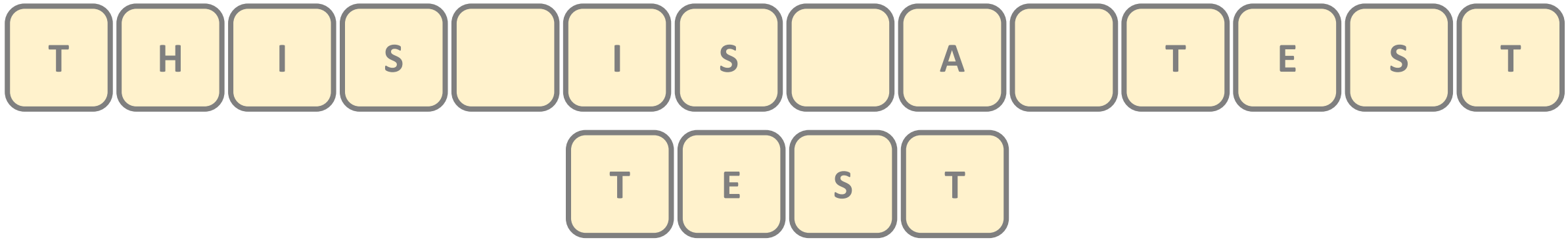
Naive Substring Search



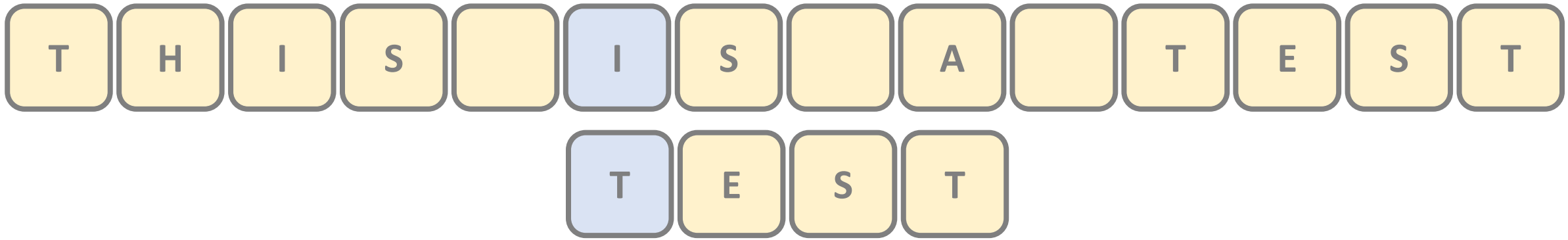
Naive Substring Search



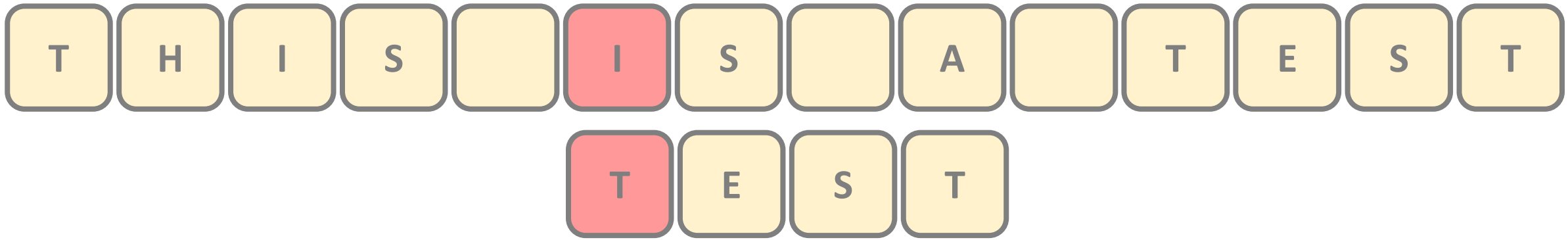
Naive Substring Search



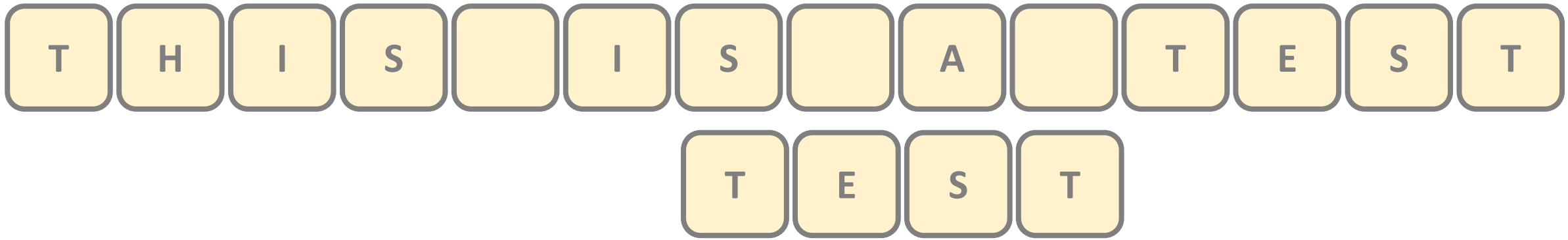
Naive Substring Search



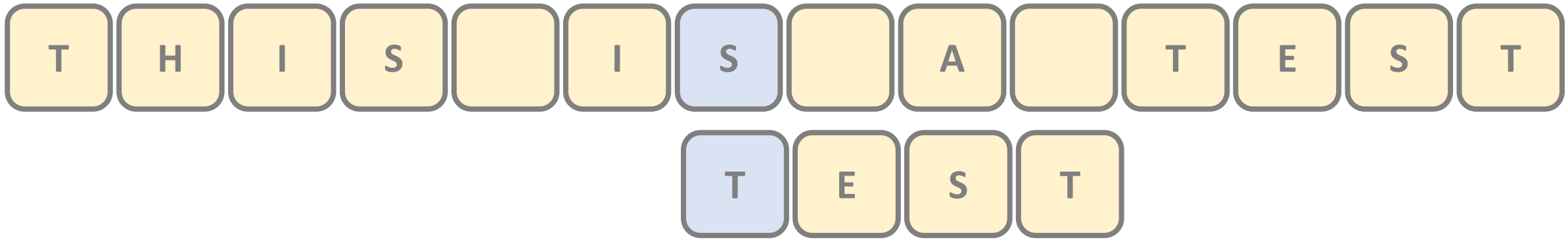
Naive Substring Search



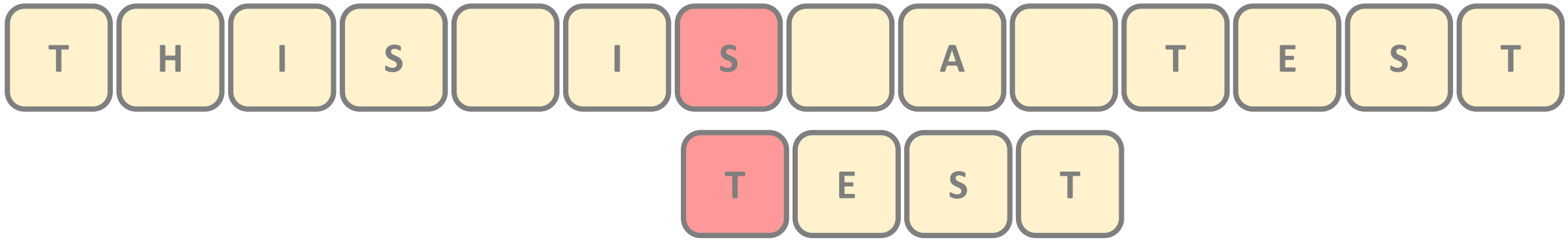
Naive Substring Search



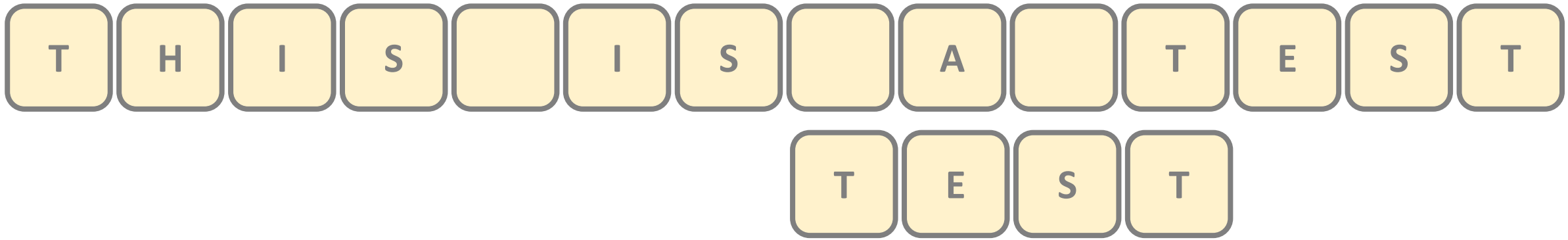
Naive Substring Search



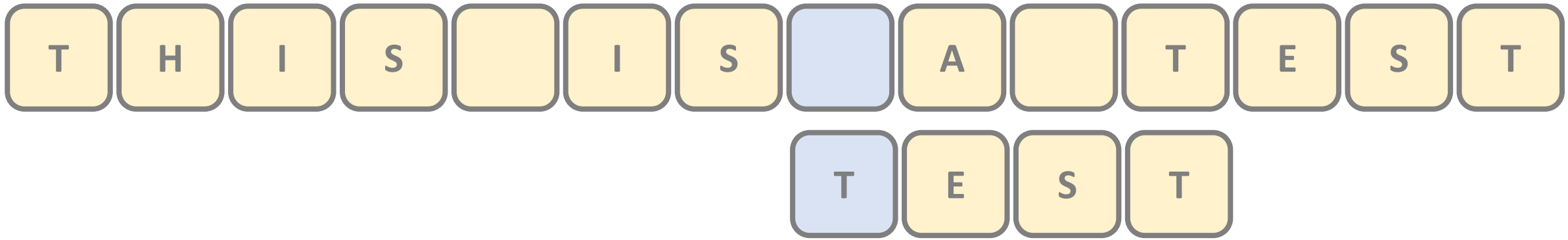
Naive Substring Search



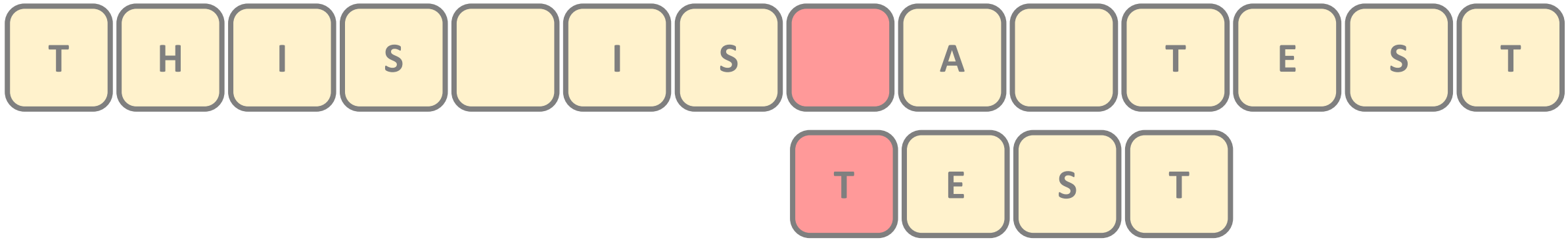
Naive Substring Search



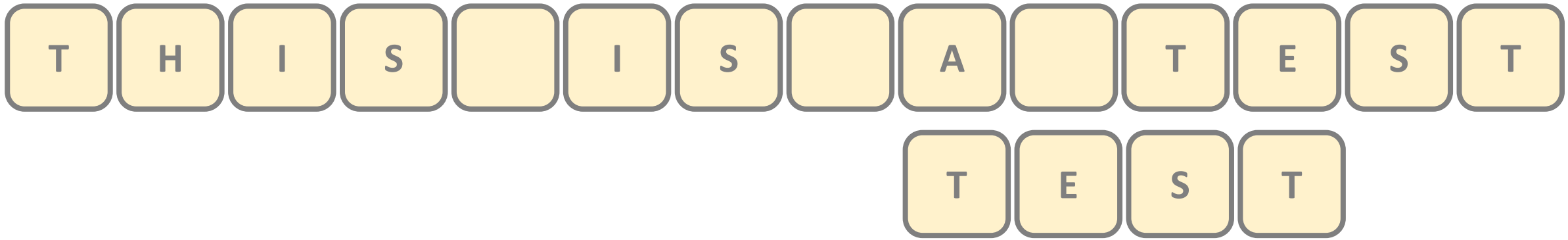
Naive Substring Search



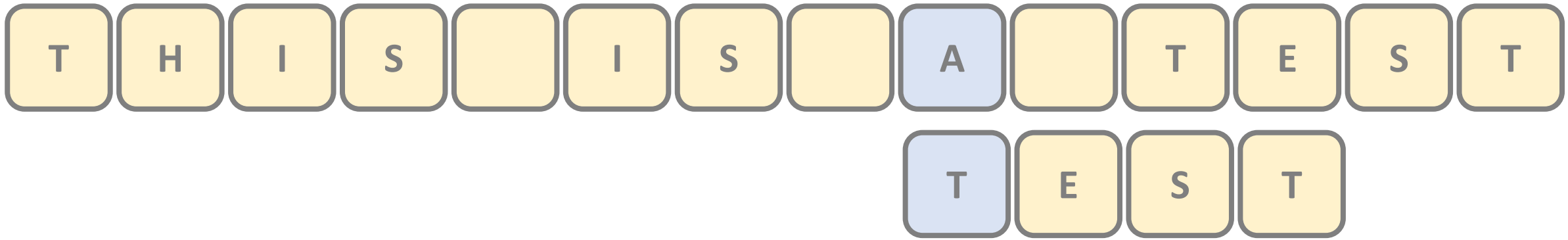
Naive Substring Search



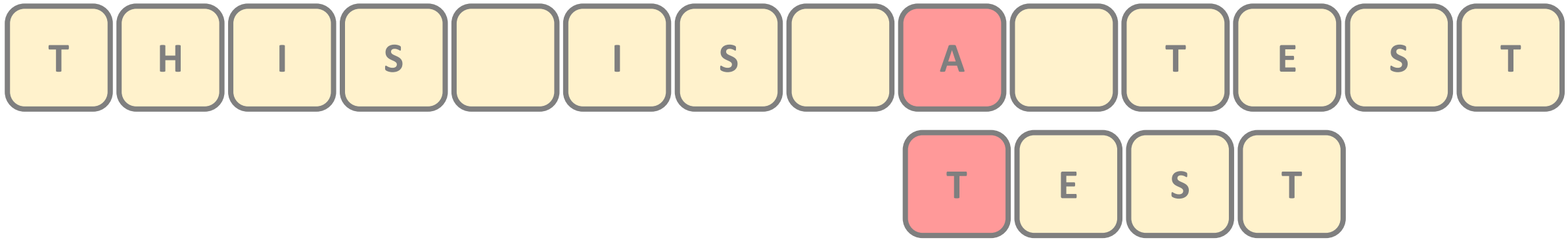
Naive Substring Search



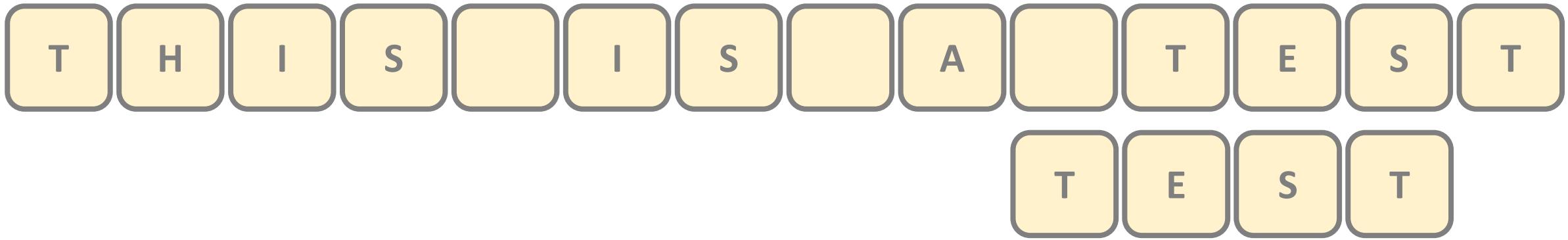
Naive Substring Search



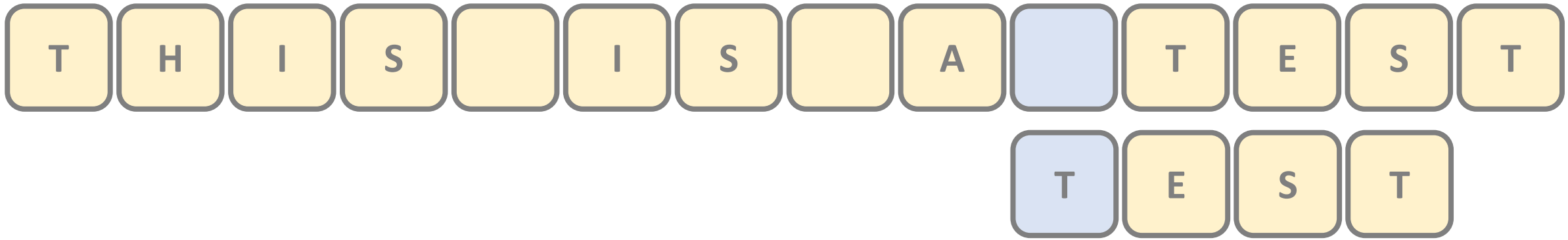
Naive Substring Search



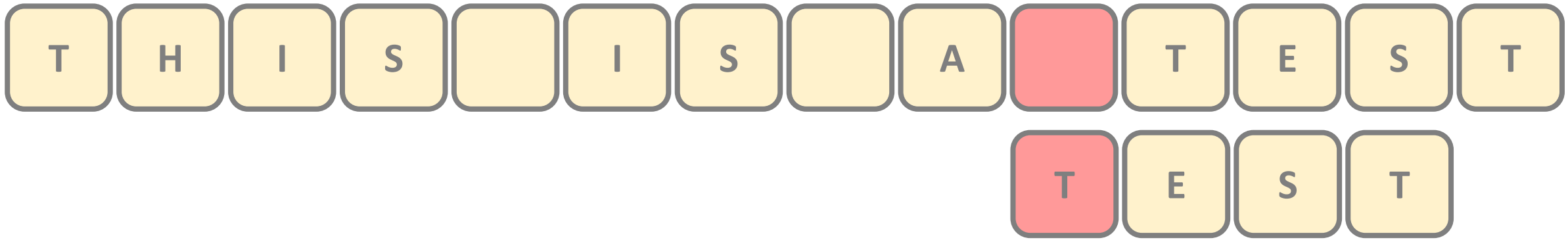
Naive Substring Search



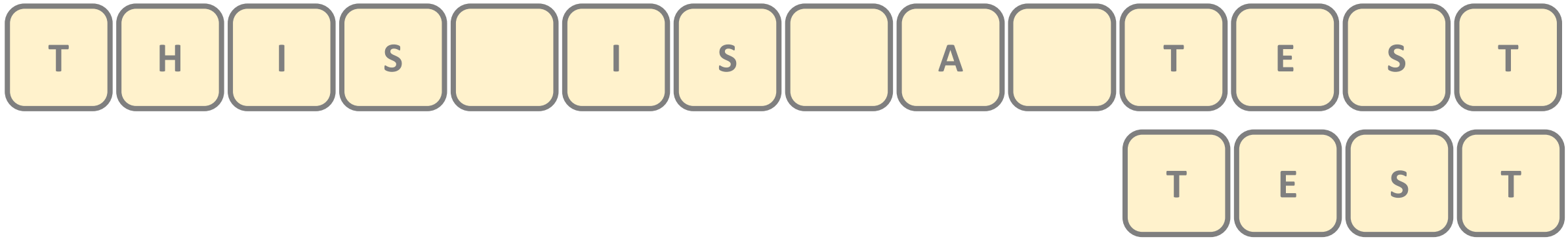
Naive Substring Search



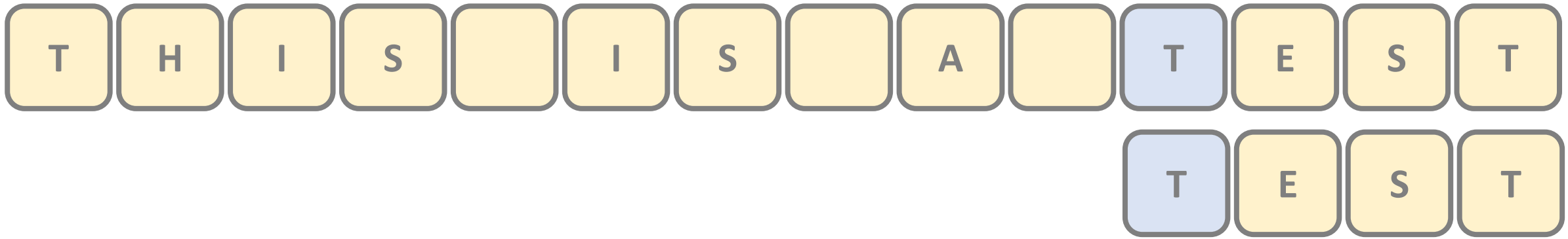
Naive Substring Search



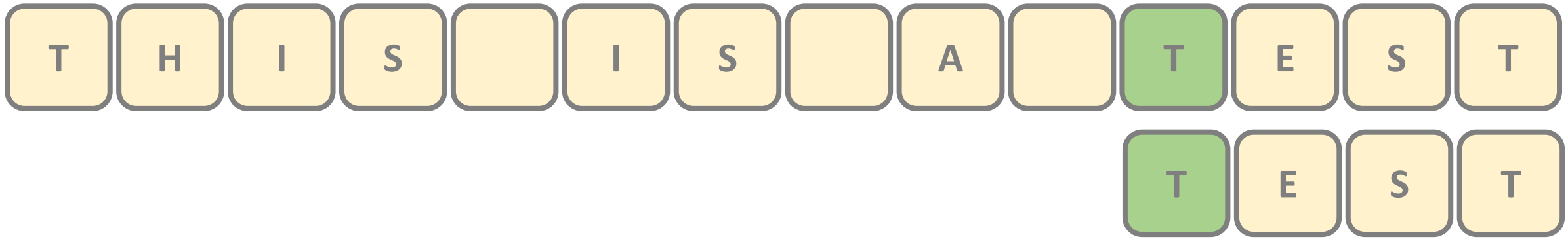
Naive Substring Search



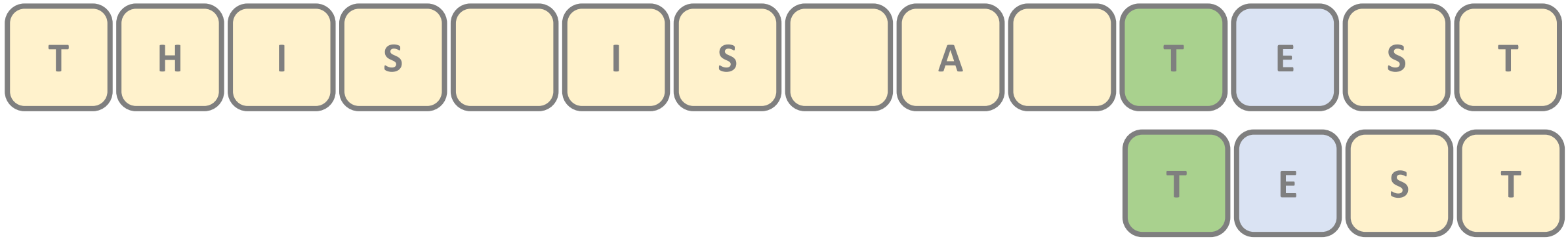
Naive Substring Search



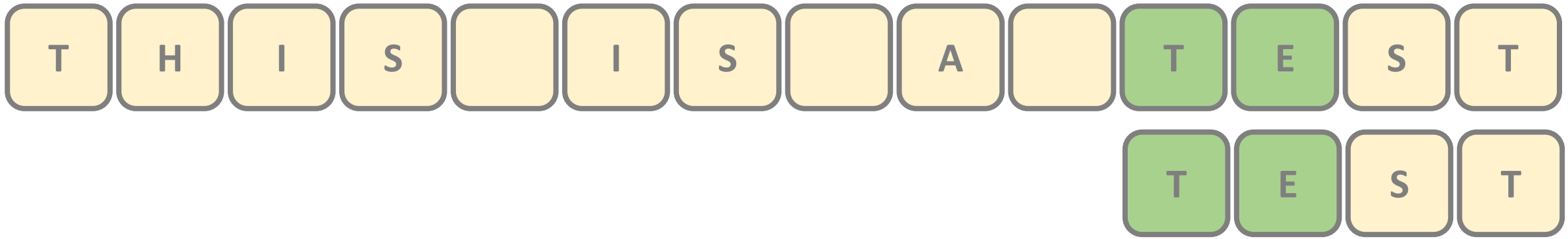
Naive Substring Search



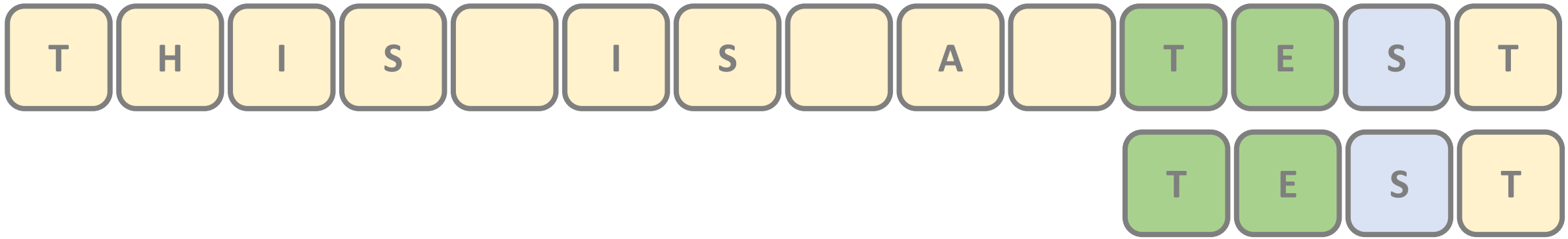
Naive Substring Search



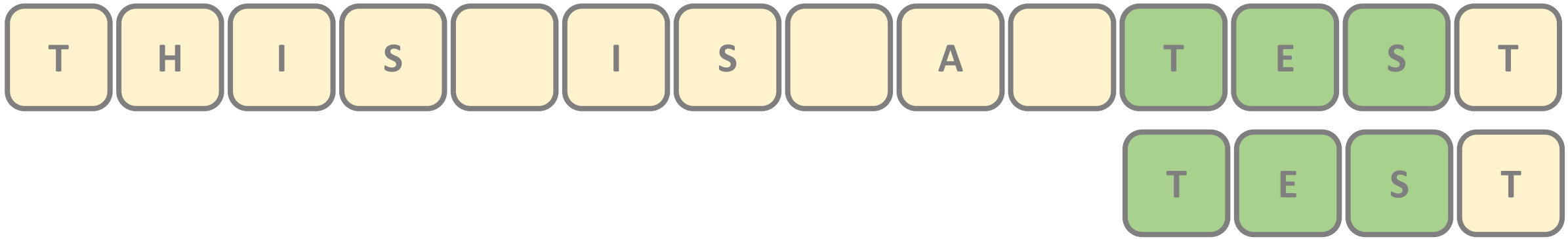
Naive Substring Search



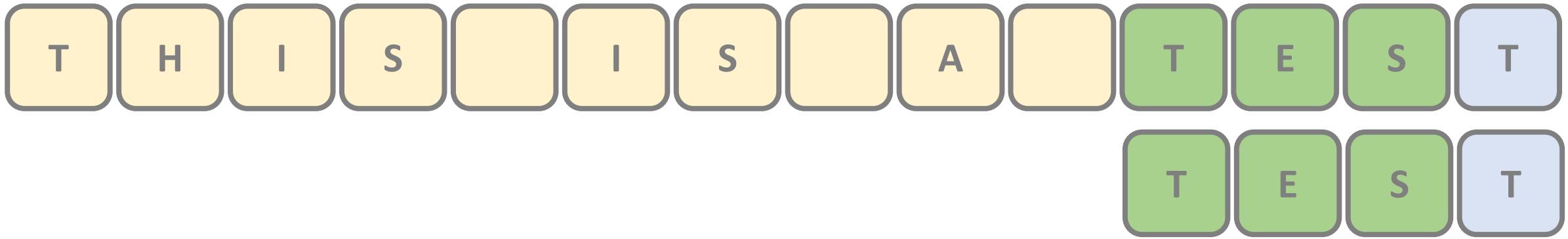
Naive Substring Search



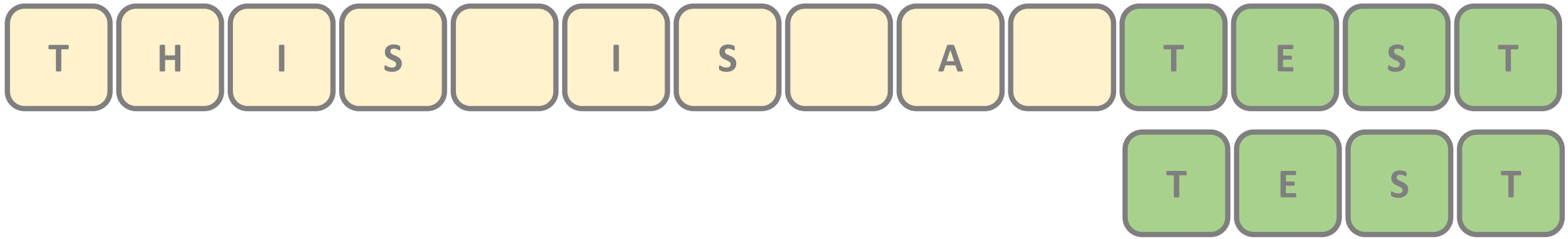
Naive Substring Search



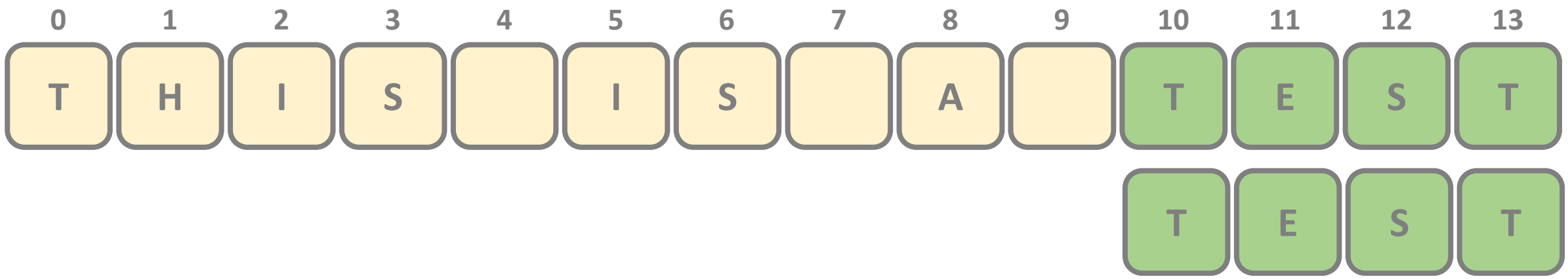
Naive Substring Search



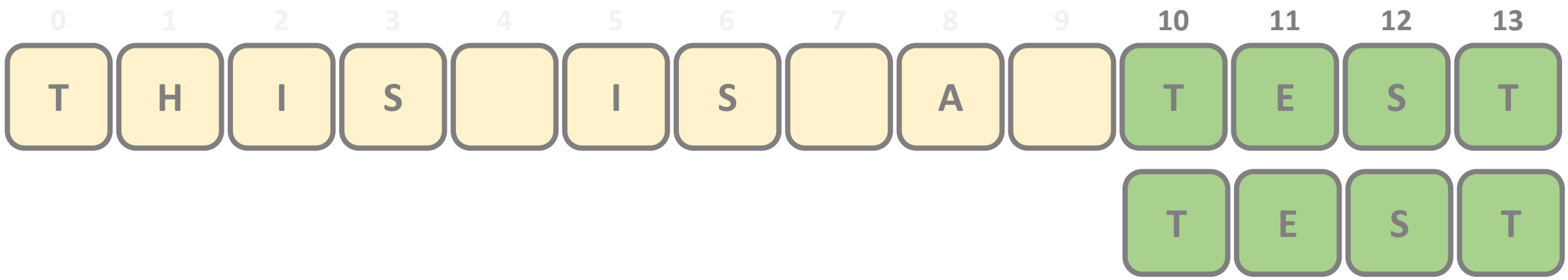
Naive Substring Search



Naive Substring Search



Naive Substring Search

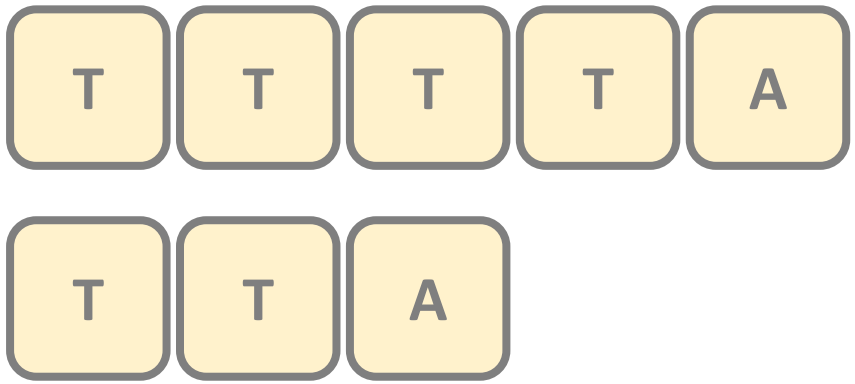


Naive Substring Search - Problem

*the **worst-case scenario***
occurs when there are several
*matching **prefixes***

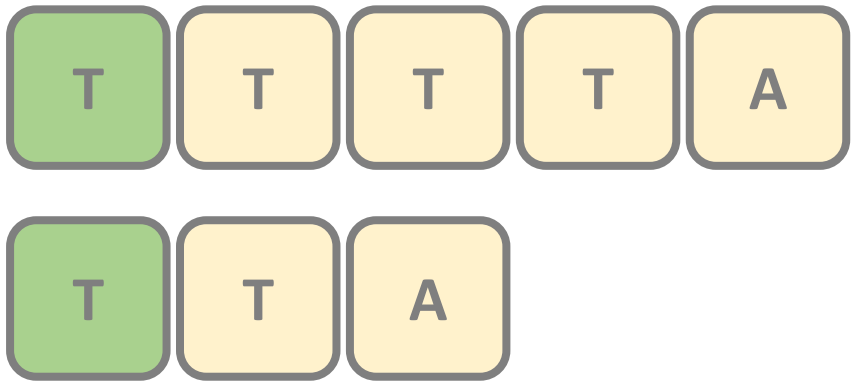
Naive Substring Search - Problem

*the **worst-case scenario** occurs when there are several matching **prefixes***



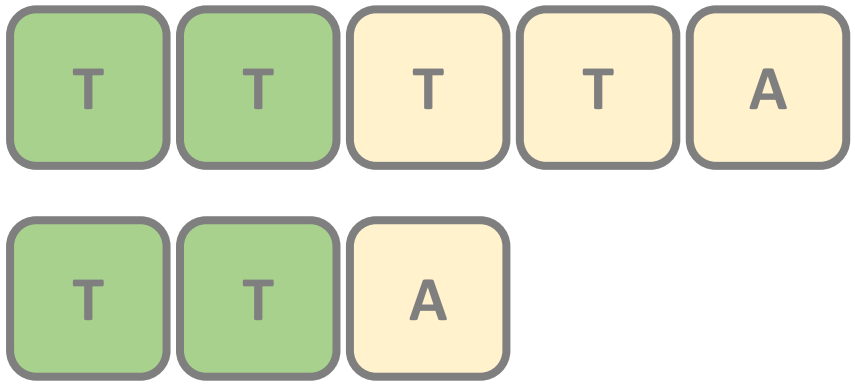
Naive Substring Search - Problem

*the **worst-case scenario** occurs when there are several matching **prefixes***



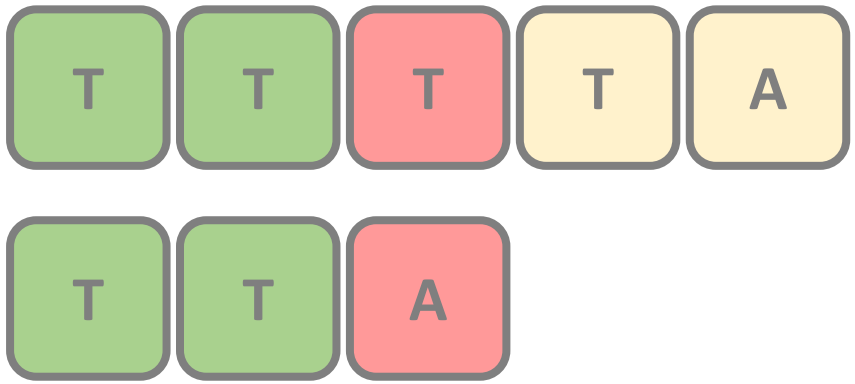
Naive Substring Search - Problem

*the **worst-case scenario** occurs when there are several matching **prefixes***



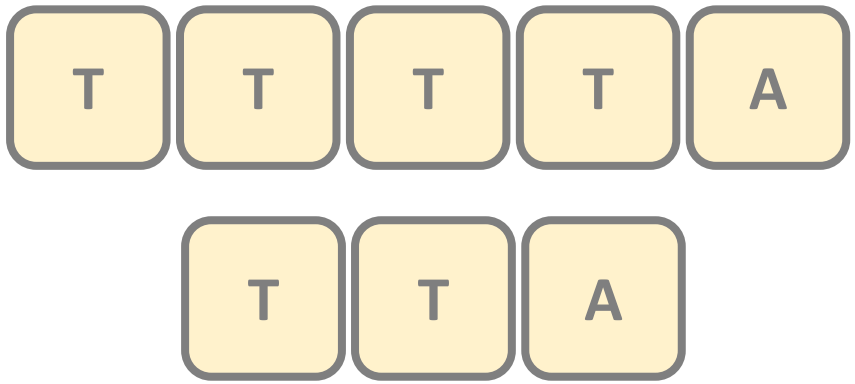
Naive Substring Search - Problem

*the **worst-case scenario** occurs when there are several matching **prefixes***



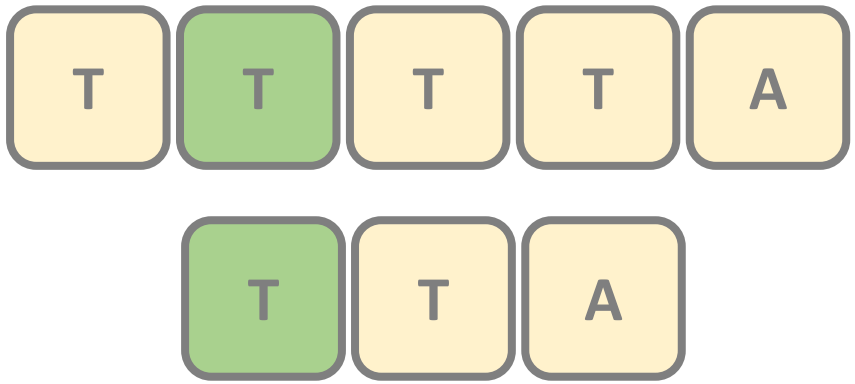
Naive Substring Search - Problem

*the **worst-case scenario** occurs when there are several matching **prefixes***



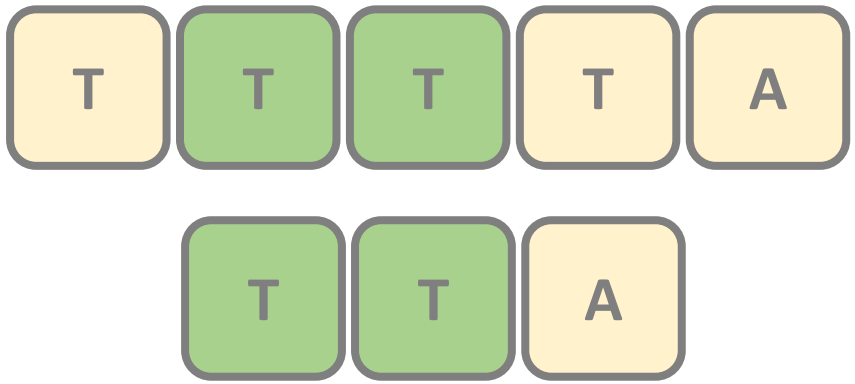
Naive Substring Search - Problem

*the **worst-case scenario** occurs when there are several matching **prefixes***

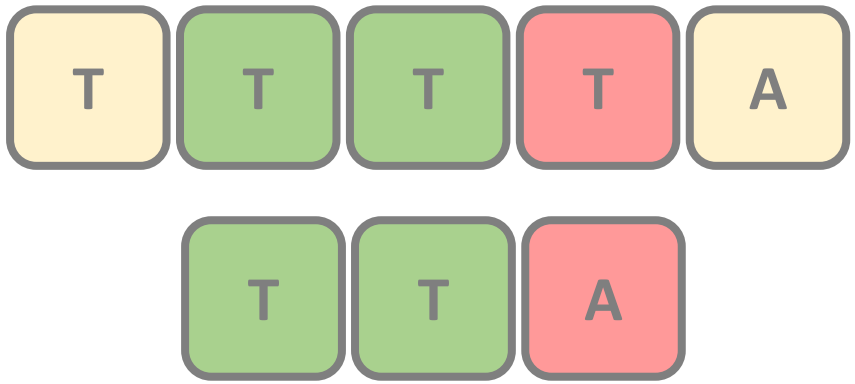


Naive Substring Search - Problem

*the **worst-case scenario** occurs when there are several matching **prefixes***



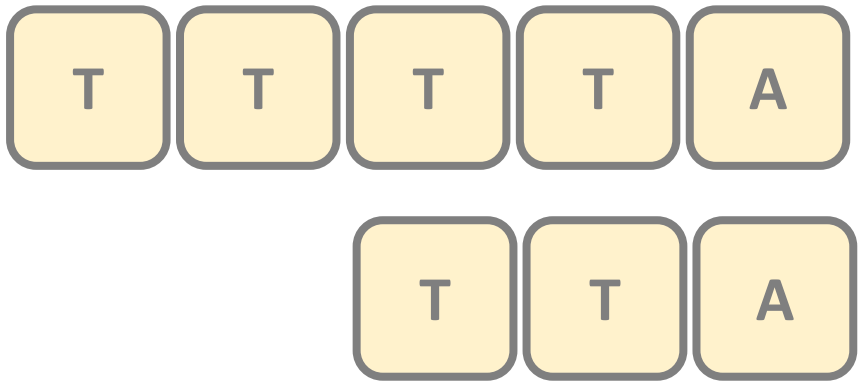
Naive Substring Search - Problem



*the **worst-case scenario** occurs when there are several matching **prefixes***

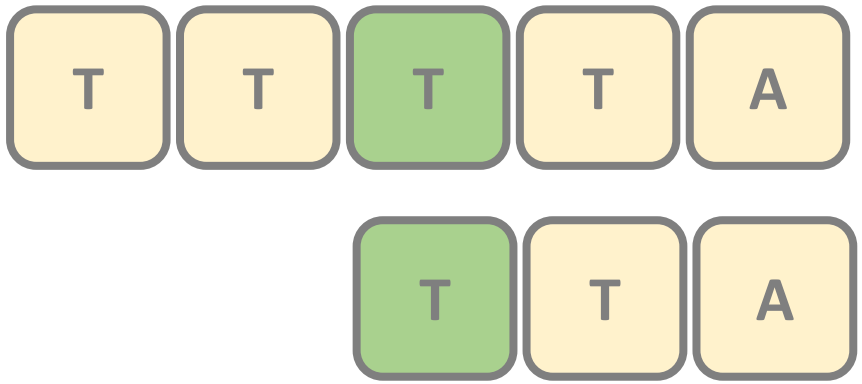
Naive Substring Search - Problem

*the **worst-case scenario** occurs when there are several matching **prefixes***



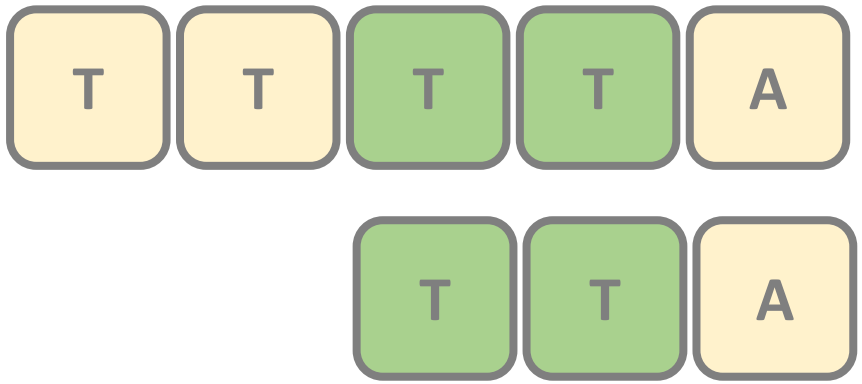
Naive Substring Search - Problem

*the **worst-case scenario** occurs when there are several matching **prefixes***



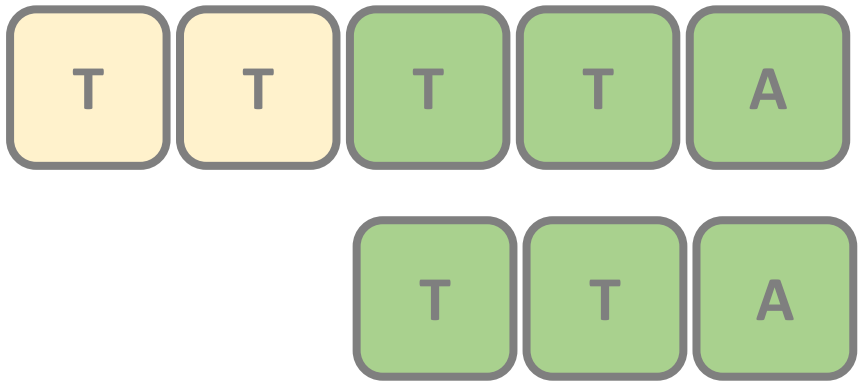
Naive Substring Search - Problem

*the **worst-case scenario** occurs when there are several matching **prefixes***



Naive Substring Search - Problem

*the **worst-case scenario** occurs when there are several matching **prefixes***



Boyer-Moore Substring Search
(Algorithms and Data Structures)

Boyer-Moore Substring Search

- we have considered brute-force substring search algorithm
- the problem that is have to make **too many comparisons** when there are matching prefixes
- **IS IT POSSIBLE TO SHIFT THE PATTERN MULTIPLE STEPS?**

Boyer-Moore Substring Search

This is exactly the logic behind **Boyer-Moore algorithm**

1.) REVERSE COMPARISON OF CHARACTERS

- we start comparing the letters starting with the last ones

2.) BAD CHARACTER HEURISTIC

- we can skip multiple characters of the original text by using the so-called **bad character heuristic**
- first we have to preprocess the pattern in **O(M)** where **M** is the length of the pattern

Boyer-Moore Substring Search

→ the original **Boyer-Moore** algorithm uses **2** heuristics

(**bad match table** and **suffix table**)

→ this is how we can gurantee that we always make the optimal number of shifts of the pattern

→ **Boyer-Moore-Horspool** algorithm uses the bad match table exclusively (it is working fine as well)

Bad Match Table

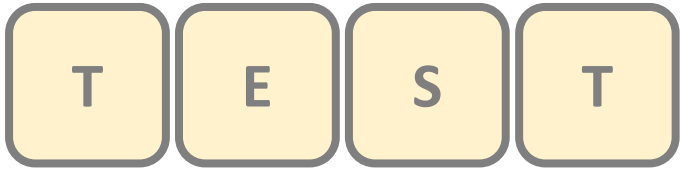
→ we construct a **table of the characters** out of the pattern

→ make sure the table does not contains repetitive characters

max(1 , pattern length – actual index – 1)

→ we iterate over the pattern (except for the last character) and compute the values to the bad match table and **keep updating the old values** for the same characters

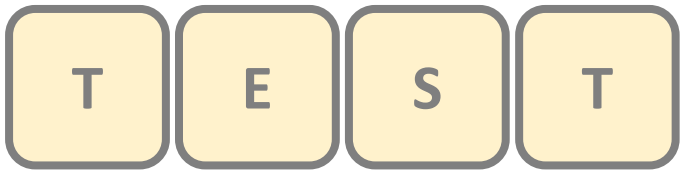
Bad Match Table



*first we have to **preprocess** the pattern in **O(M)** running time*

Bad Match Table

*first we have to **preprocess** the pattern in **O(M)** running time*

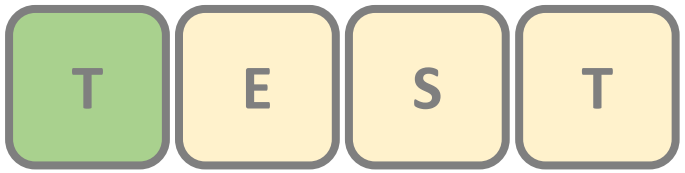


max(1 , pattern length – actual index – 1)

LETTERS	T	E	S	*
VALUES				

Bad Match Table

*first we have to **preprocess** the pattern in **O(M)** running time*



max(1 , pattern length – actual index – 1)

LETTERS	T	E	S	*
VALUES	3			

Bad Match Table

*first we have to **preprocess** the pattern in **O(M)** running time*

max(1 , pattern length – actual index – 1)

LETTERS	T	E	S	*
VALUES	3	2		

Bad Match Table

*first we have to **preprocess** the pattern in **O(M)** running time*

max(1 , pattern length – actual index – 1)

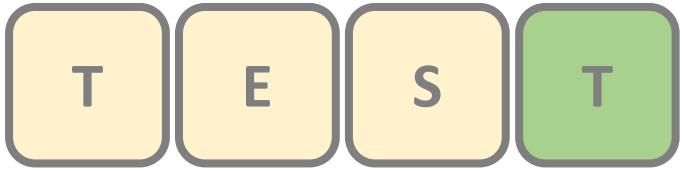
LETTERS	T	E	S	*
VALUES	3	2	1	

Bad Match Table

*last character is **special***

***1.)** if it is **not unique** then we can apply the formula (value is **1**)*

***2.)** if it is a **unique character** then we do not bother with it (it is the * case)*

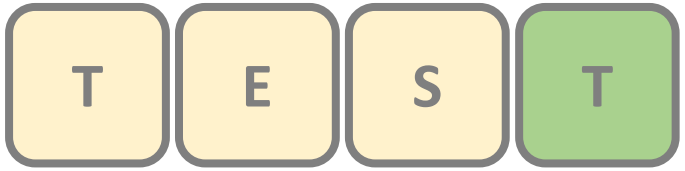


max(1 , pattern length – actual index – 1)

LETTERS	T	E	S	*
VALUES	3	2	1	

Bad Match Table

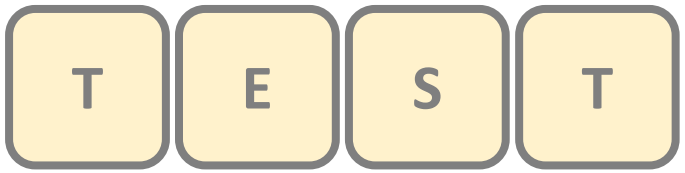
*first we have to **preprocess** the pattern in **O(M)** running time*



max(1 , pattern length – actual index – 1)

LETTERS	T	E	S	*
VALUES	1	2	1	

Bad Match Table

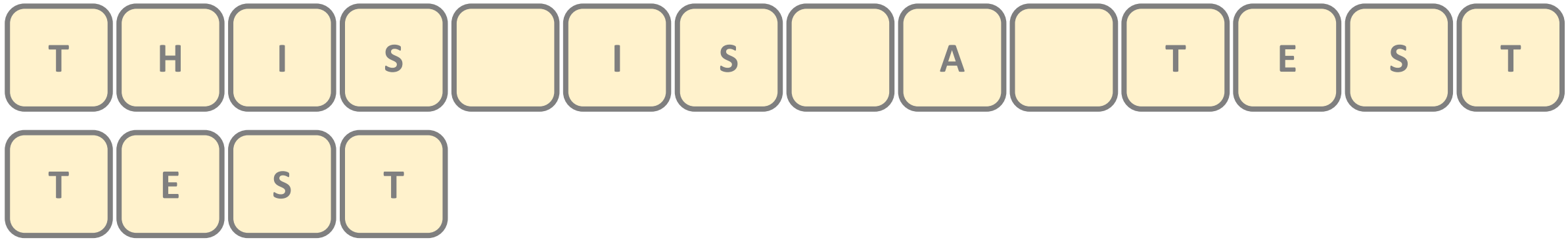


*first we have to **preprocess** the pattern in **O(M)** running time*

*this * represents all other characters with value **length of the pattern***

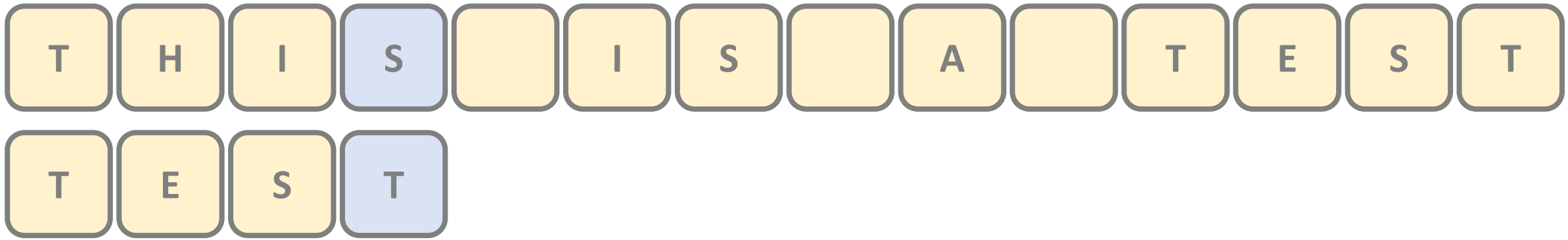
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



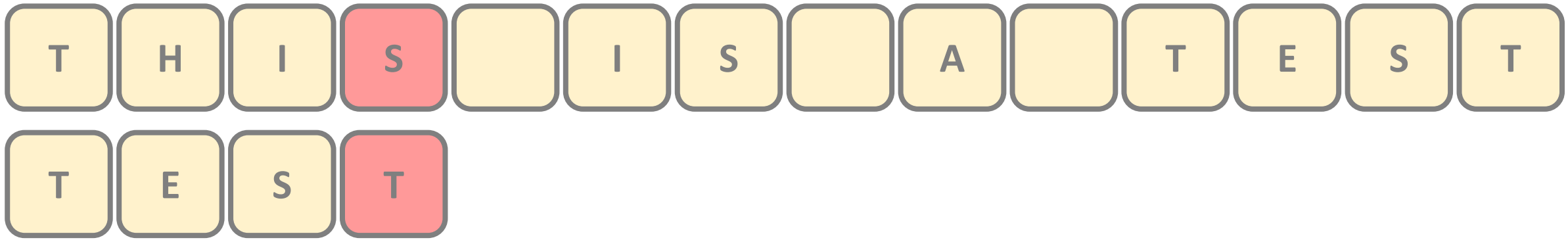
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



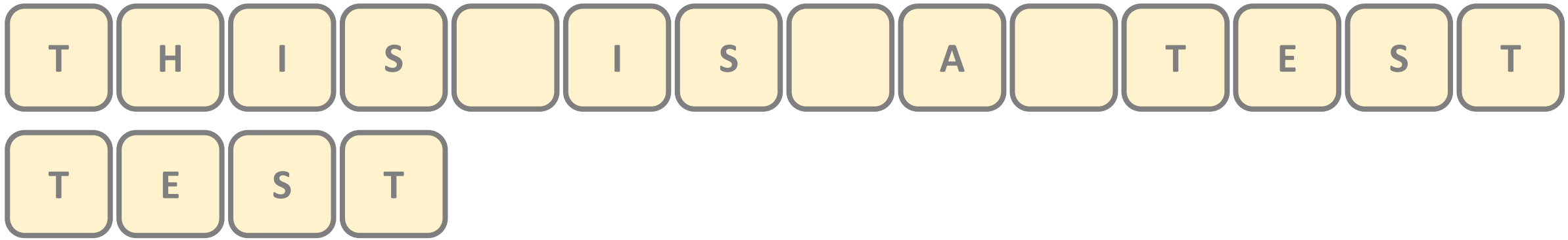
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



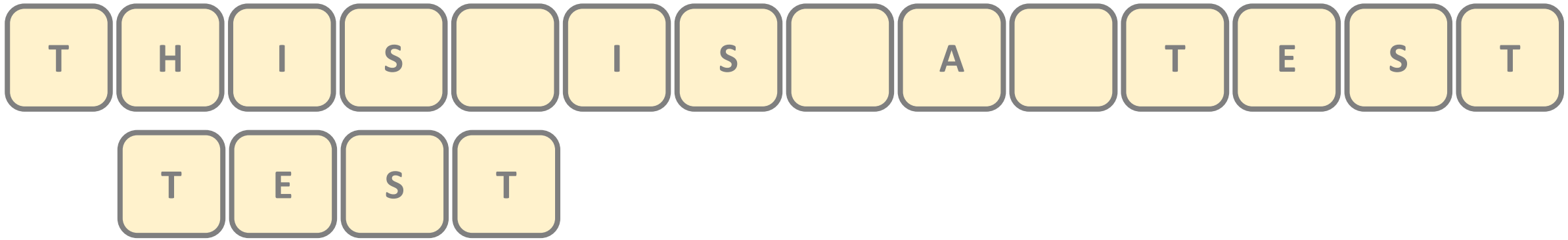
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



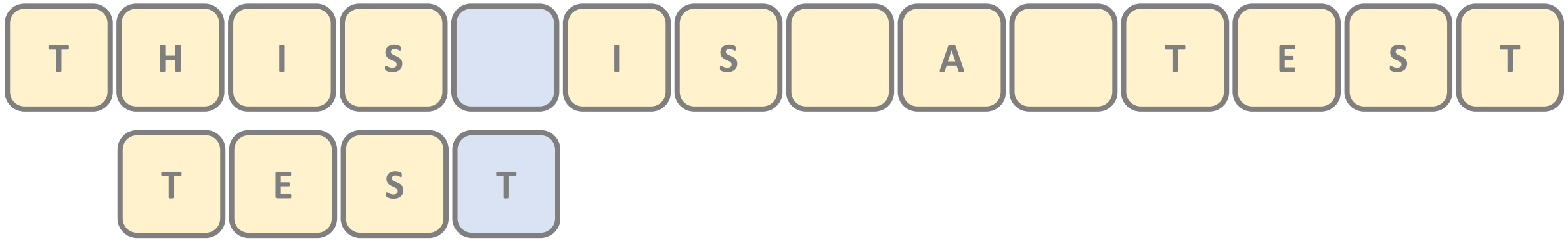
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



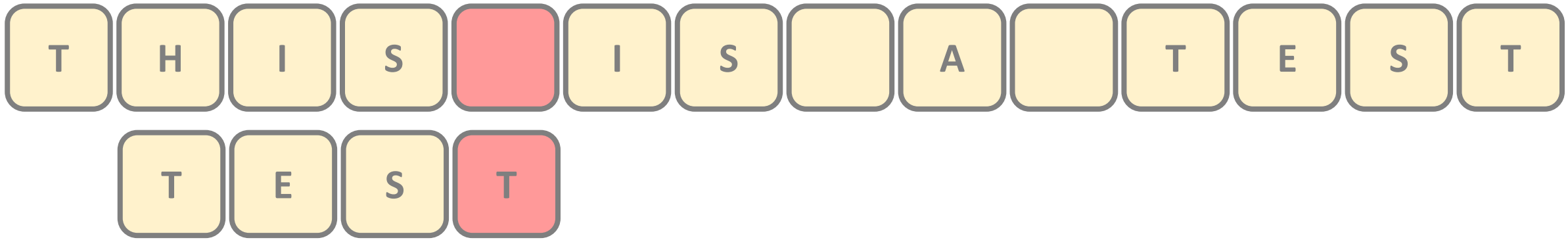
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



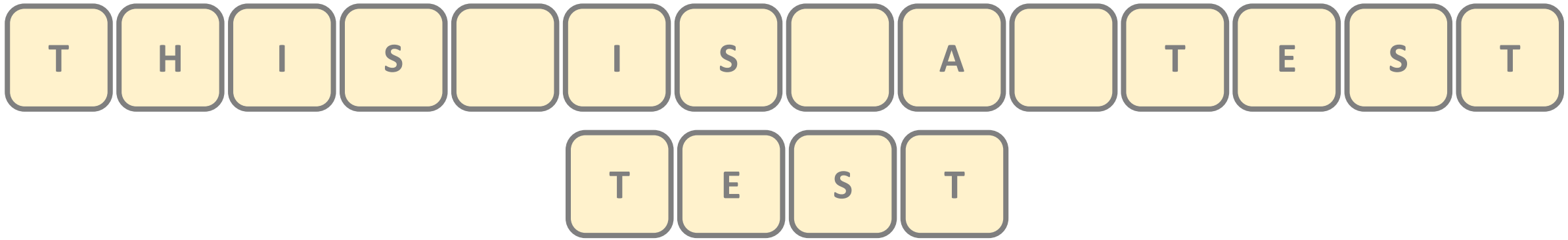
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



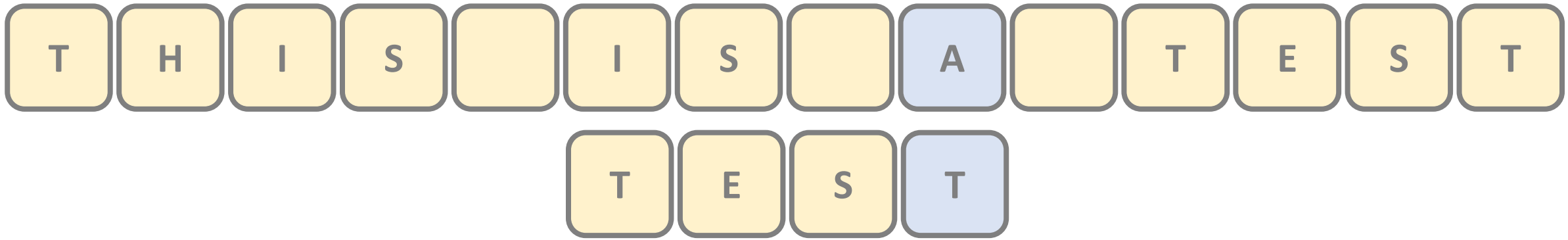
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



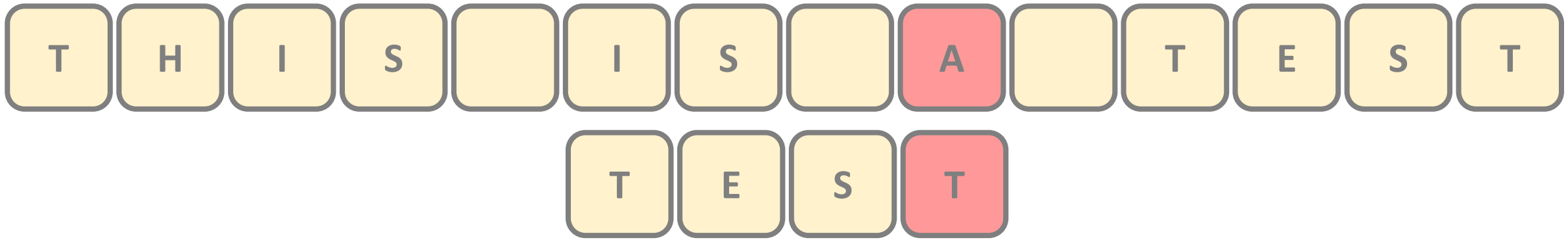
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



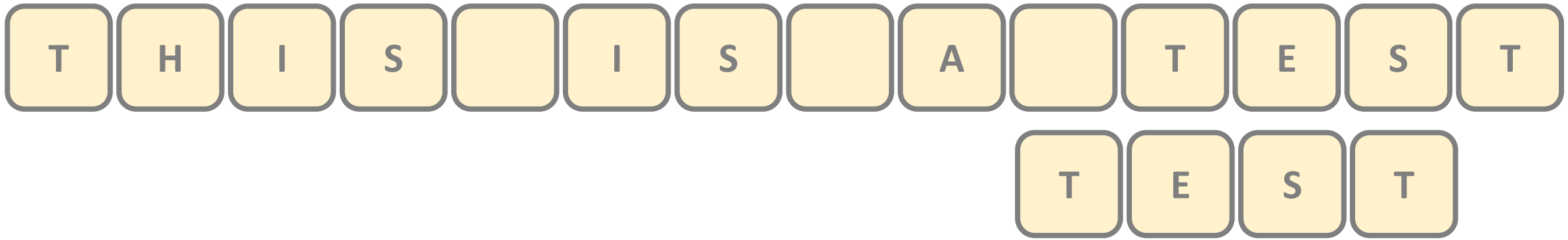
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



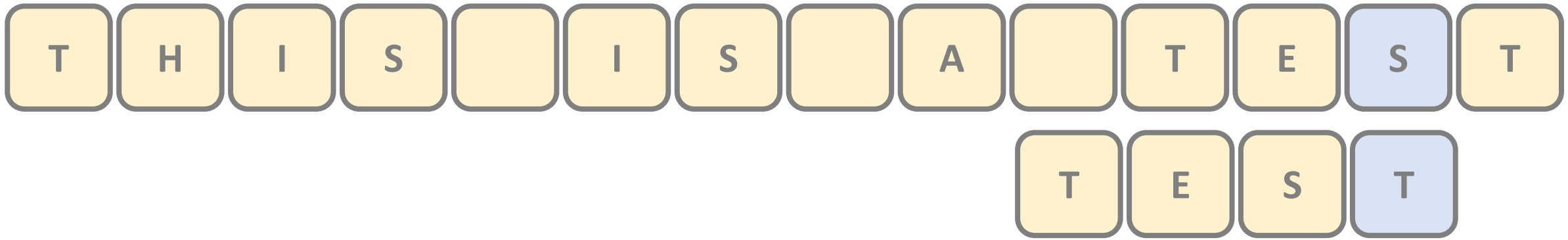
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



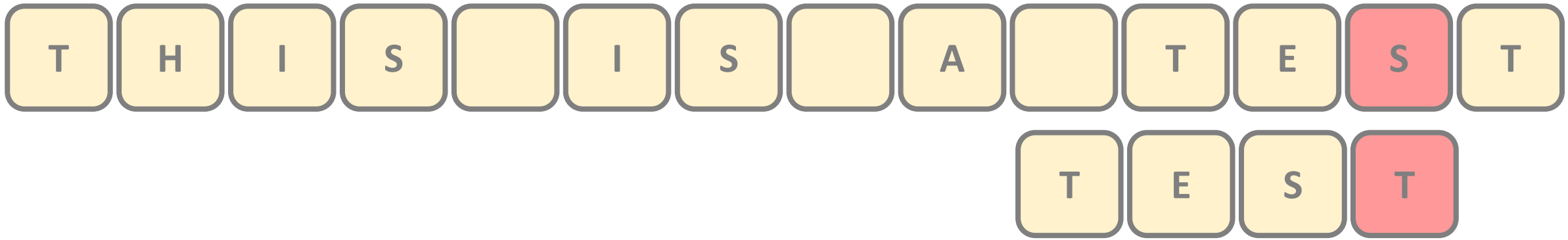
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



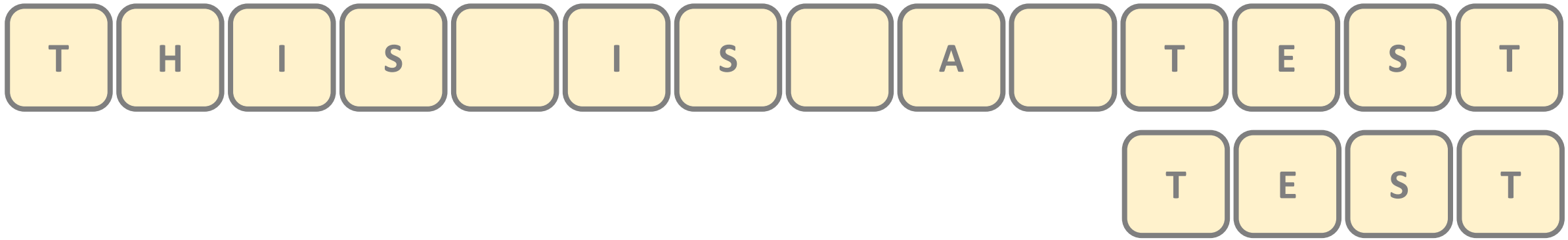
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



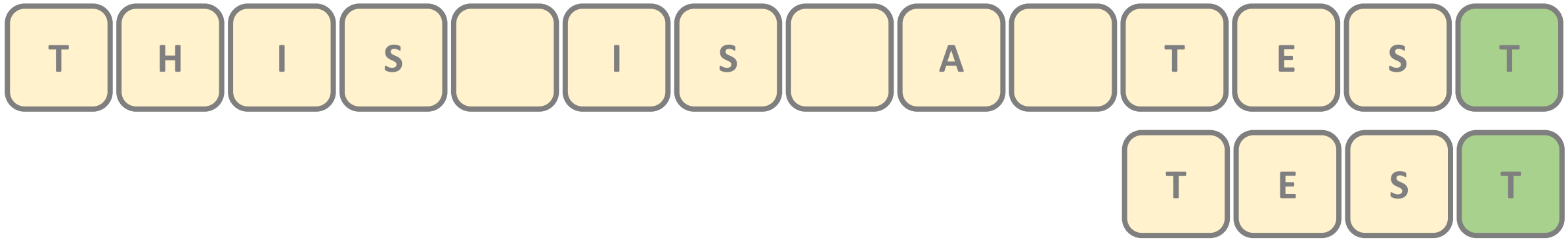
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



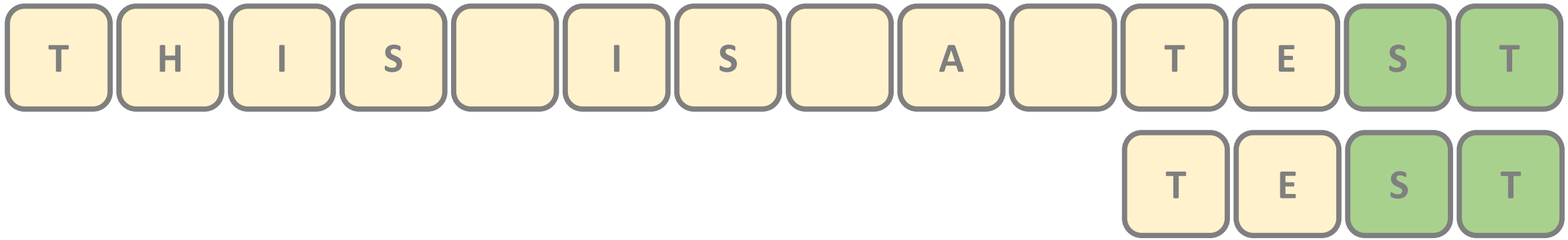
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



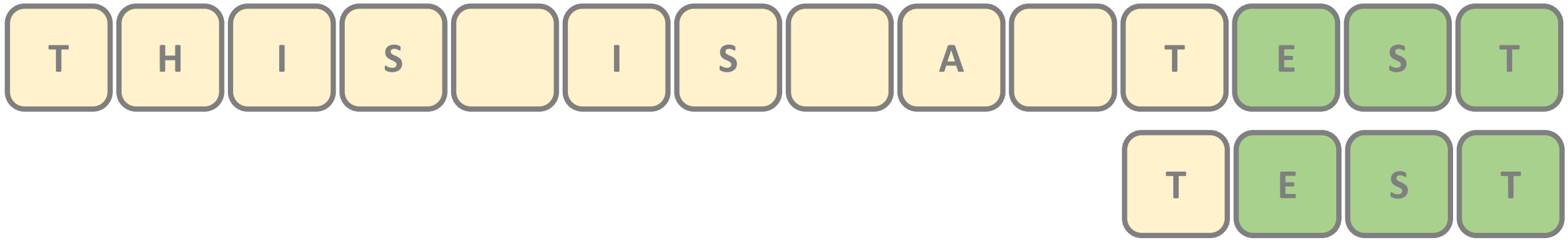
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



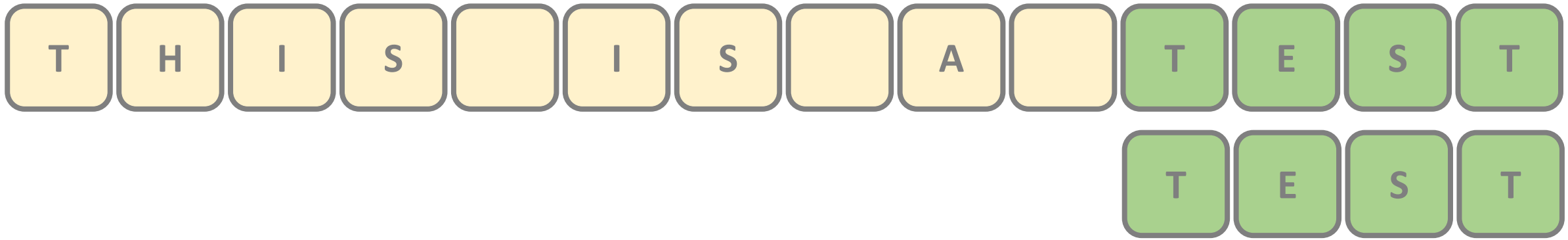
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search



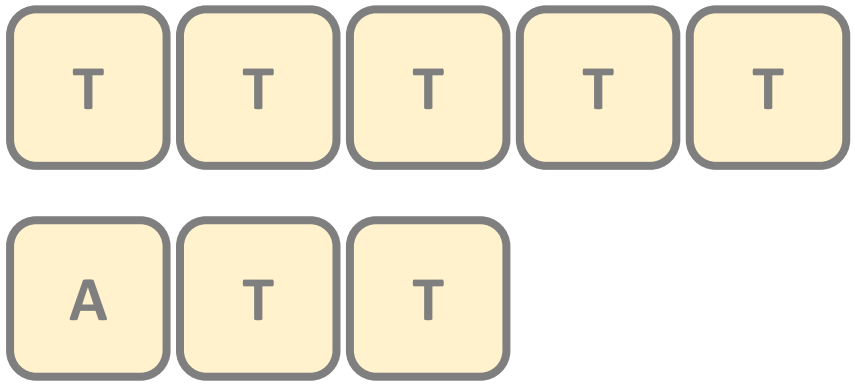
LETTERS	T	E	S	*
VALUES	1	2	1	4

Boyer-Moore Substring Search

- this approach turns out to be **quite efficient**
- mismatched character heuristics takes about **N/M** character comparisons where **M** is the length of the pattern and **N** is the length of the text
- it is not even linear – it is **sublinear**
- so the longer the pattern the faster the algorithm becomes
- it has **O(M+N)** average-case running time but the worst-case running time is still **O(MN)**

Boyer-Moore Substring Search

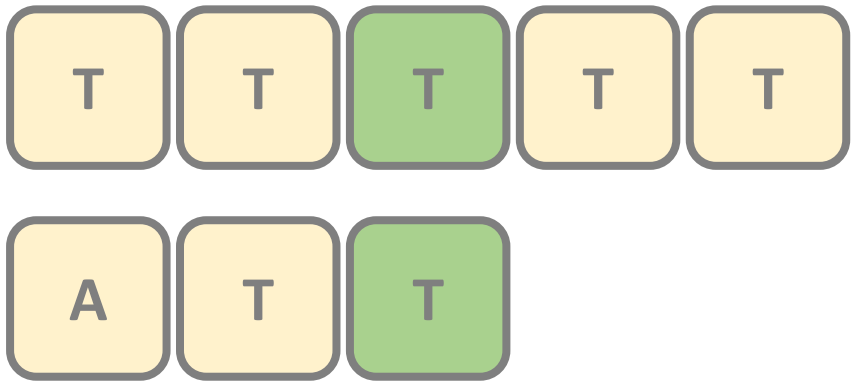
*the **worst-case scenario** occurs when there are several matching **suffixes***



LETTERS	A	T	*
VALUES	2	1	3

Boyer-Moore Substring Search

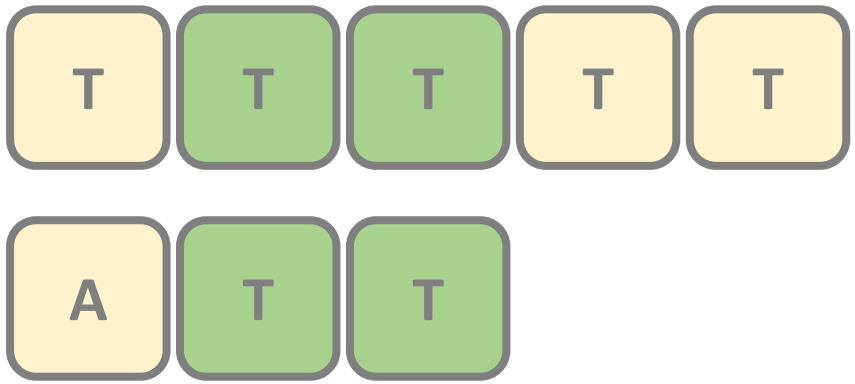
*the **worst-case scenario** occurs when there are several matching **suffixes***



LETTERS	A	T	*
VALUES	2	1	3

Boyer-Moore Substring Search

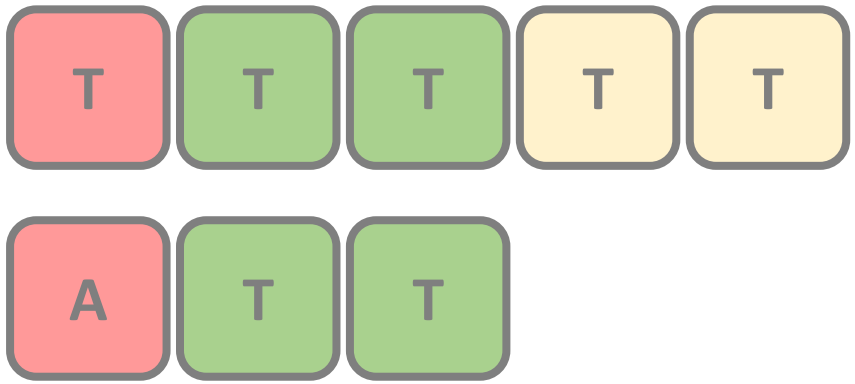
*the **worst-case scenario** occurs when there are several matching **suffixes***



LETTERS	A	T	*
VALUES	2	1	3

Boyer-Moore Substring Search

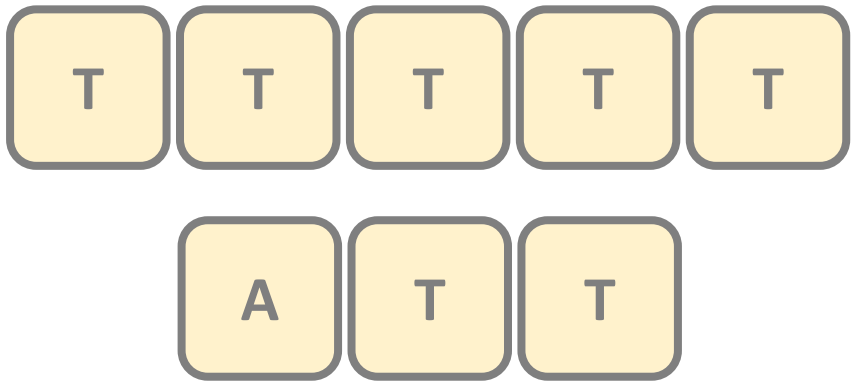
*the **worst-case scenario** occurs when there are several matching **suffixes***



LETTERS	A	T	*
VALUES	2	1	3

Boyer-Moore Substring Search

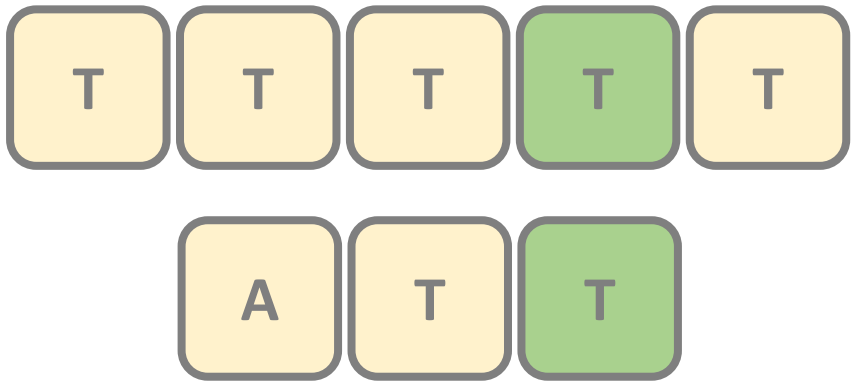
*the **worst-case scenario** occurs when there are several matching **suffixes***



LETTERS	A	T	*
VALUES	2	1	3

Boyer-Moore Substring Search

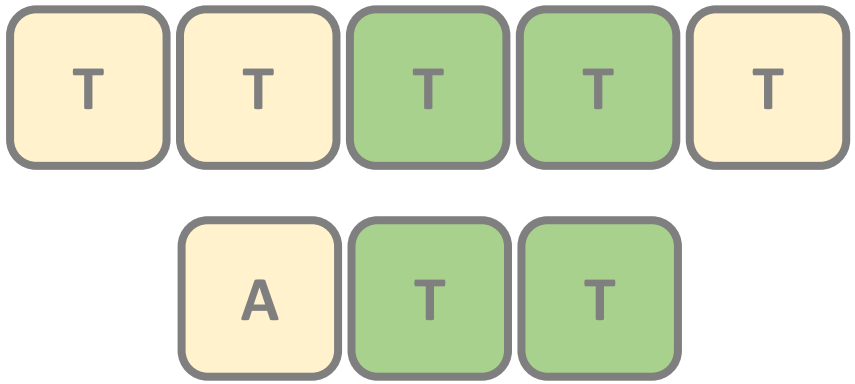
*the **worst-case scenario** occurs when there are several matching **suffixes***



LETTERS	A	T	*
VALUES	2	1	3

Boyer-Moore Substring Search

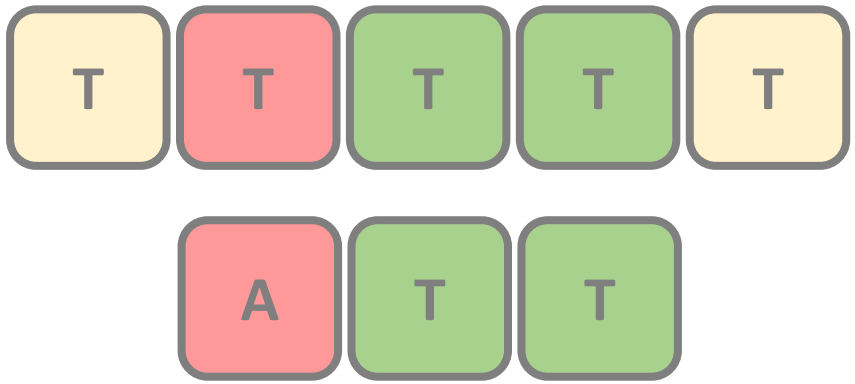
*the **worst-case scenario** occurs when there are several matching **suffixes***



LETTERS	A	T	*
VALUES	2	1	3

Boyer-Moore Substring Search

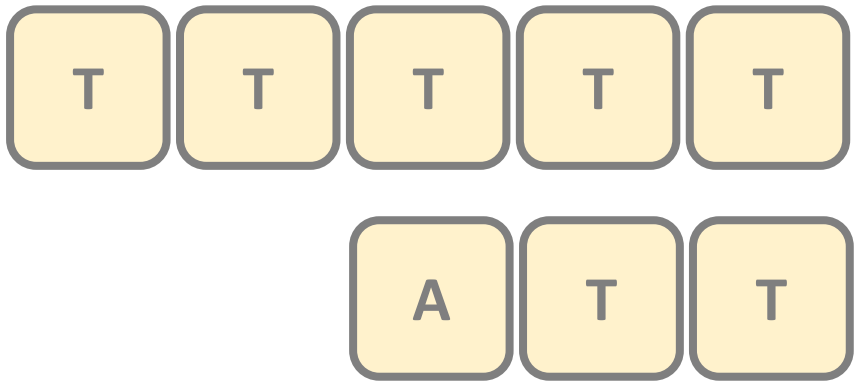
*the **worst-case scenario** occurs when there are several matching **suffixes***



LETTERS	A	T	*
VALUES	2	1	3

Boyer-Moore Substring Search

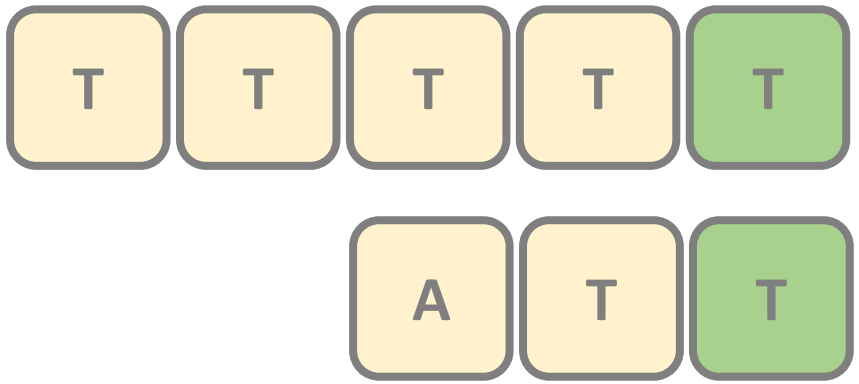
*the **worst-case scenario** occurs when there are several matching **suffixes***



LETTERS	A	T	*
VALUES	2	1	3

Boyer-Moore Substring Search

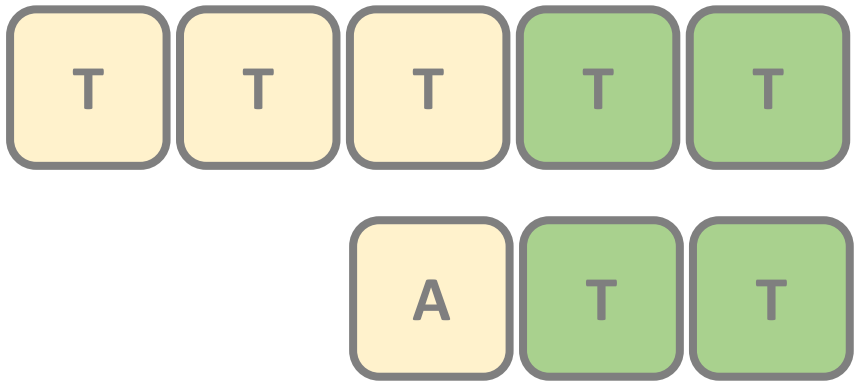
*the **worst-case scenario** occurs when there are several matching **suffixes***



LETTERS	A	T	*
VALUES	2	1	3

Boyer-Moore Substring Search

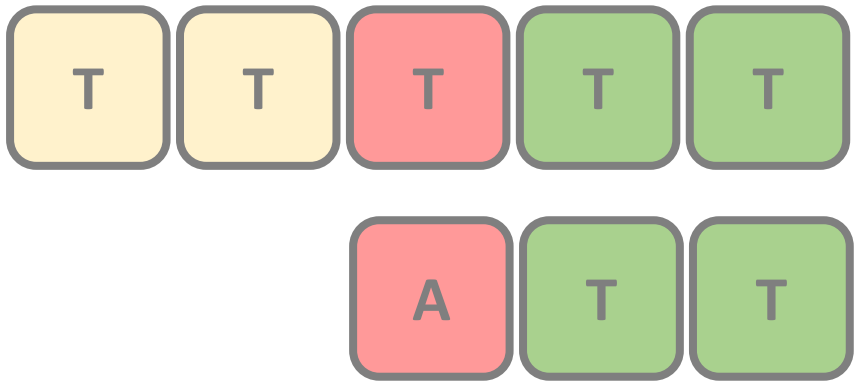
*the **worst-case scenario** occurs when there are several matching **suffixes***



LETTERS	A	T	*
VALUES	2	1	3

Boyer-Moore Substring Search

*the **worst-case scenario** occurs when there are several matching **suffixes***



LETTERS	A	T	*
VALUES	2	1	3