String Searching Algorithms

Robert Horvick
SOFTWARE ENGINEER

@bubbafat www.roberthorvick.com

Overview

String searching overview

Naive search algorithm

- Algorithm
- Performance

Boyer-Moore-Horspool algorithm

- Algorithm
- Performance

Demo: Search and replace

```
public interface IStringSearchAlgorithm
{
    IEnumerable<ISearchMatch> Search(string pattern, string toSearch);
}
```

Search Algorithm

The IStringSearchAlgorithm defines the function, Search, which will be called to find all of the matches of the search string (pattern) within the input (toSearch) string.

```
public interface ISearchMatch
{
    int Start { get; }
    int Length { get; }
}
```

Search Matches

The ISearchMatch interface defines the index and length of a search match in the input string.

```
string pattern = "fox";

string toSearch = "The quick brown fox jumps over the lazy dog";

foreach(ISearchMatch match in algorithm.Search(pattern, toSearch))

{

    ...

}
```

Example Usage

The Search method of the IStringSearchAlgorithm instance (algorithm) returns each of the matches as an enumeration of ISearchMatch objects.

Search Example

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse sodales, enim id lobortis consectetur, neque lacus ultricies nisl, at feugiat.

Start	Length

Search Example – Finding “is”

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse sodales, enim id lobortis consectetur, neque lacus ultricies nisl, at feugiat.

Start	Length
44	2

Search Example – Finding “is”

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse sodales, enim id lobortis consectetur, neque lacus ultricies nisl, at feugiat.

Start	Length
44	2
64	2

Search Example – Finding “is”

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse sodales, enim id lobortis consectetur, neque lacus ultricies nisl, at feugiat.

Start	Length
44	2
64	2
92	2

Search Example – Finding “is”

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse sodales, enim id lobortis consectetur, neque lacus ultricies nisl, at feugiat.

Start	Length
44	2
64	2
92	2
131	2

Naive Search

Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            Match Found!
    }
}
```

Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```

T	H	E		C	A	T		C	H	A	S	E	D		A		D	O	G

Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```

T	H	E		C	A	T		C	H	A	S	E	D		A		D	O	G

A	T

Naive Search Algorithm

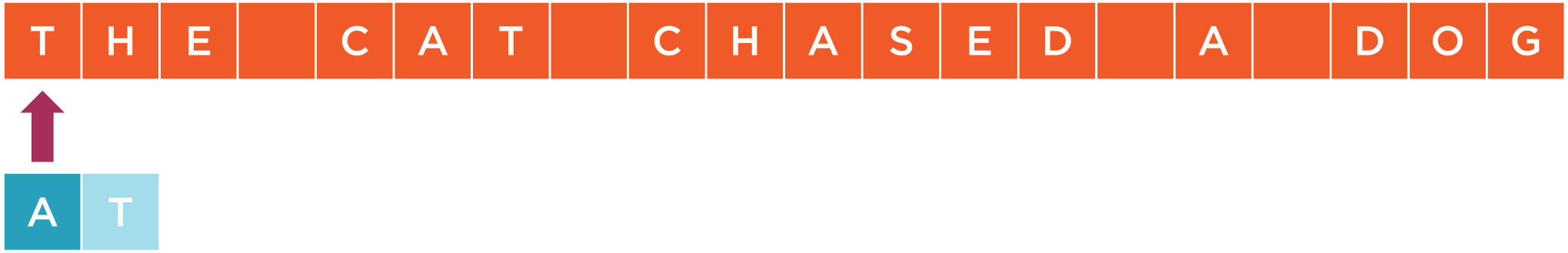
```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```

T	H	E		C	A	T		C	H	A	S	E	D		A		D	O	G

A	T

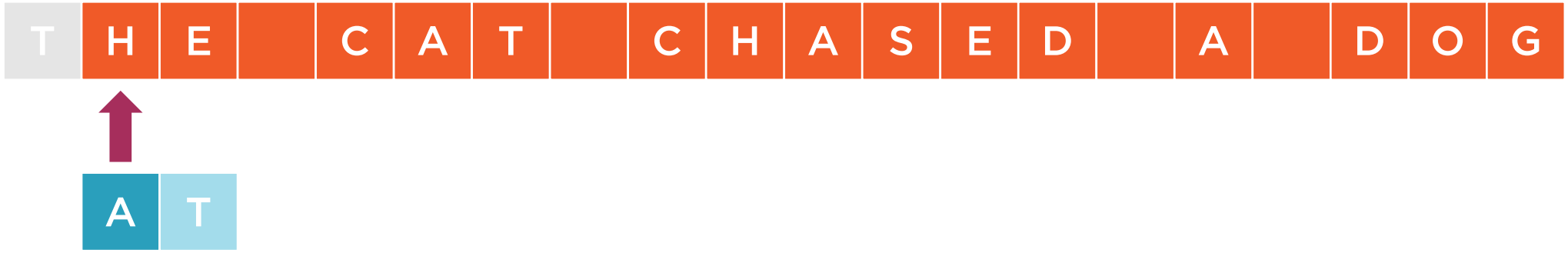
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



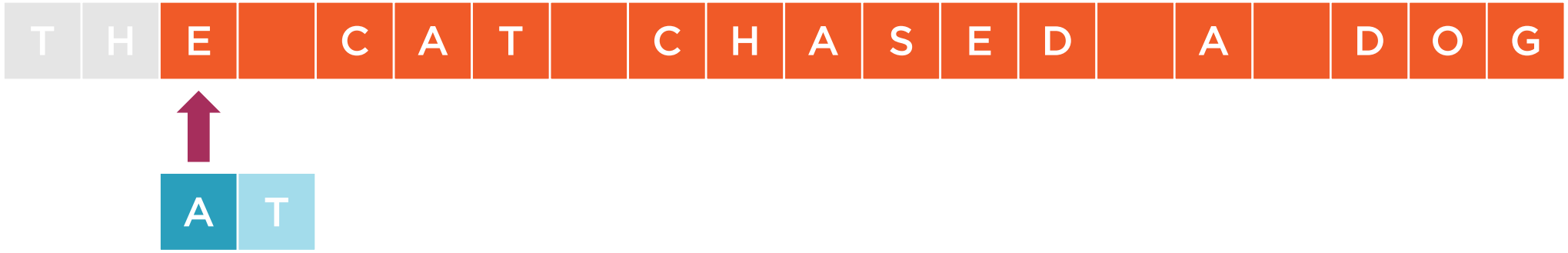
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



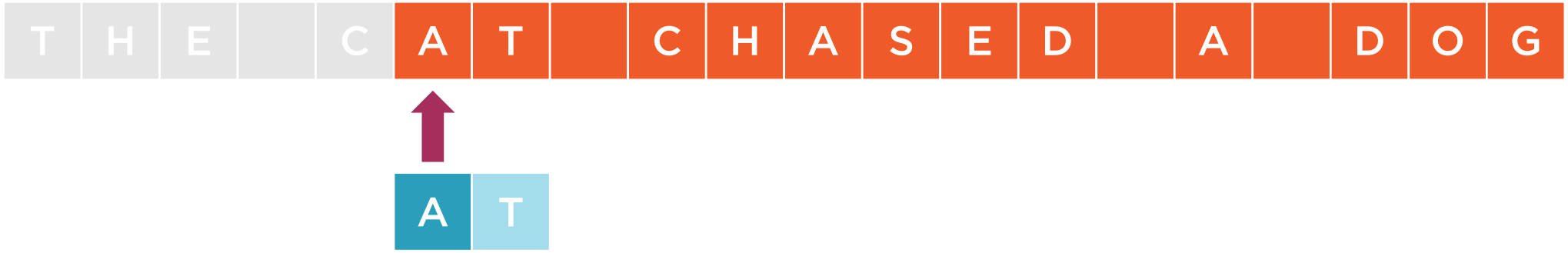
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



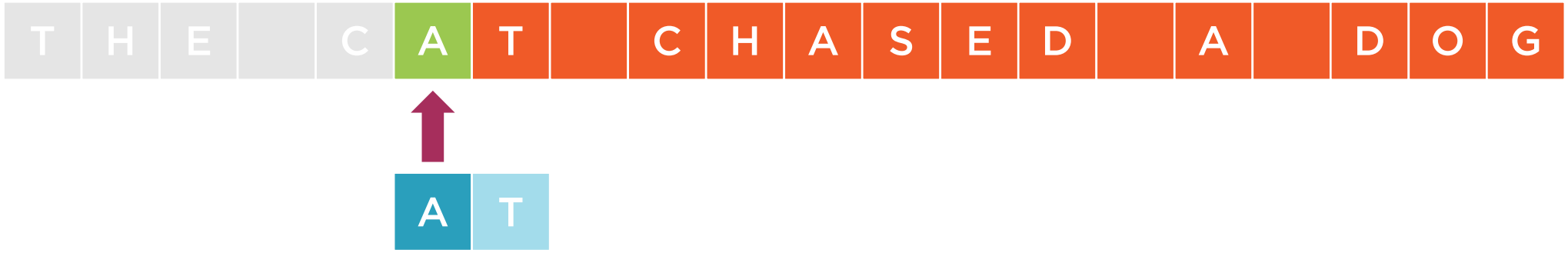
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



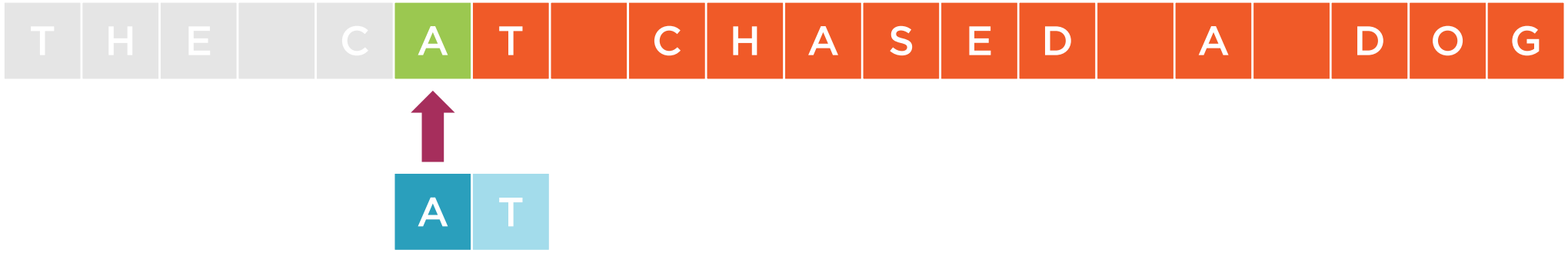
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



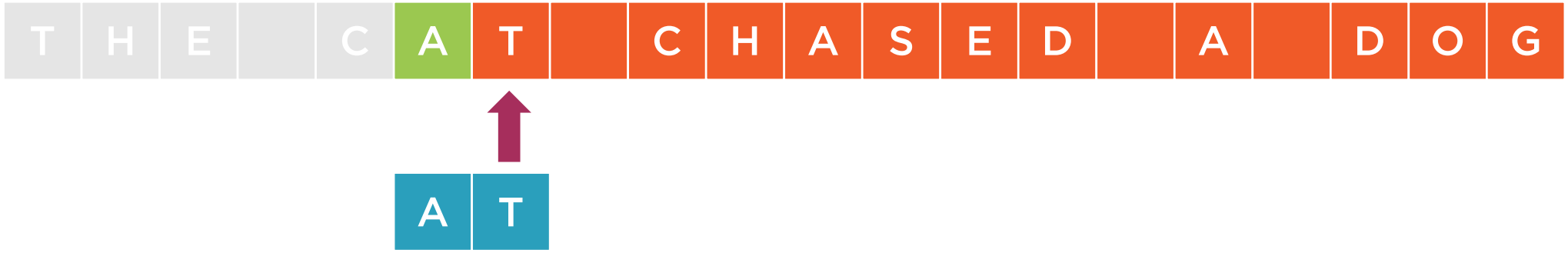
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



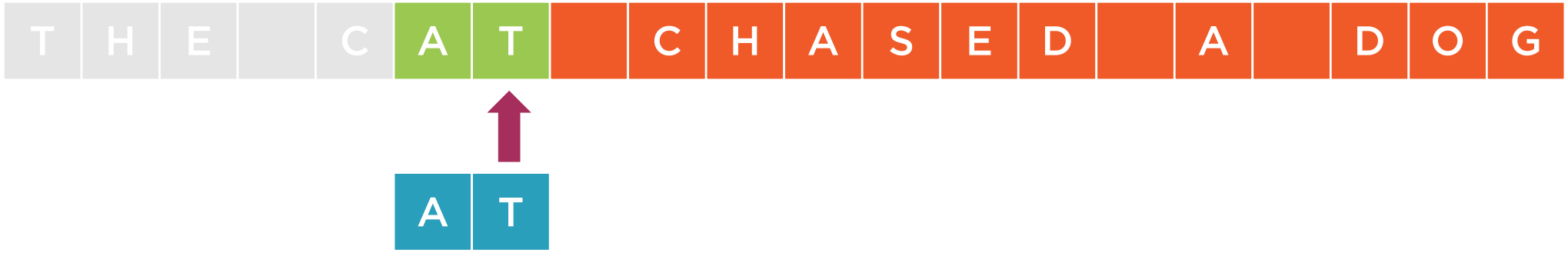
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



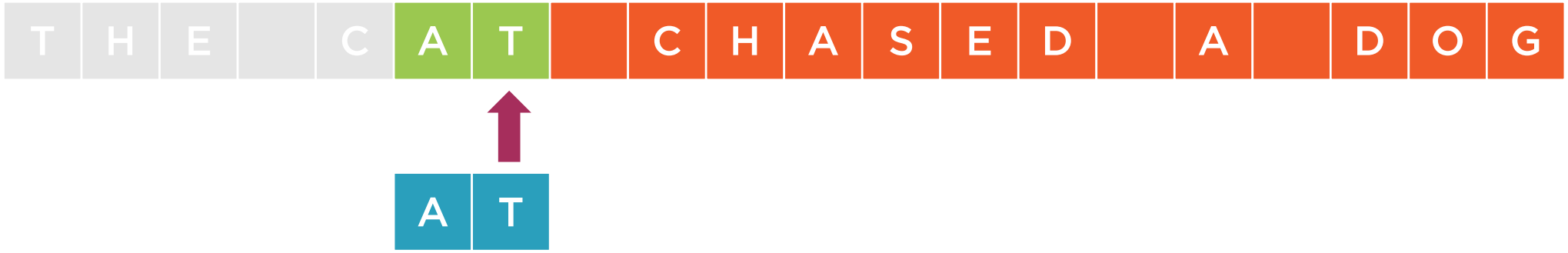
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



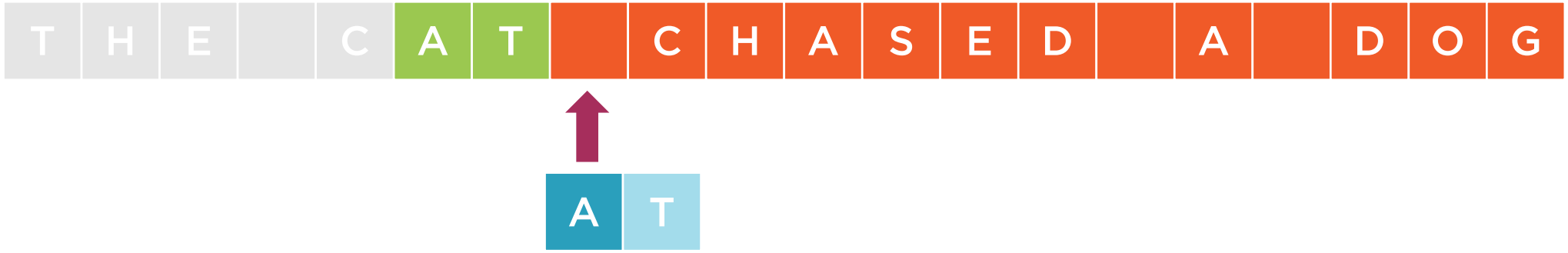
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



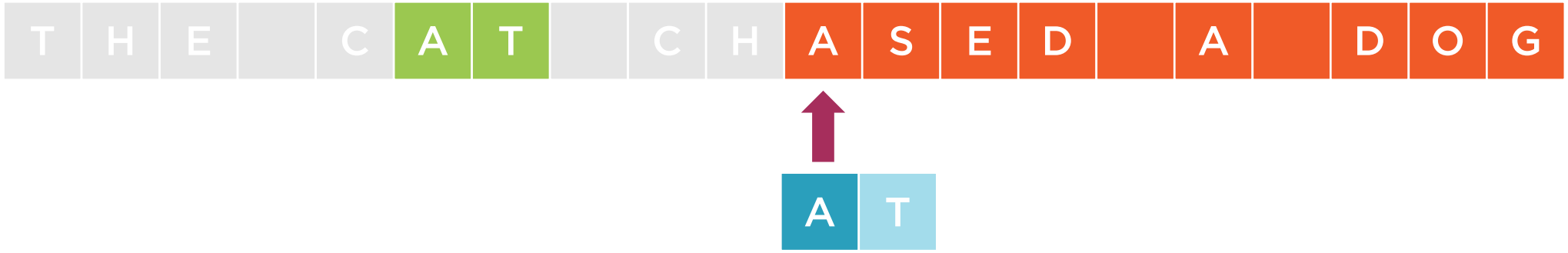
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



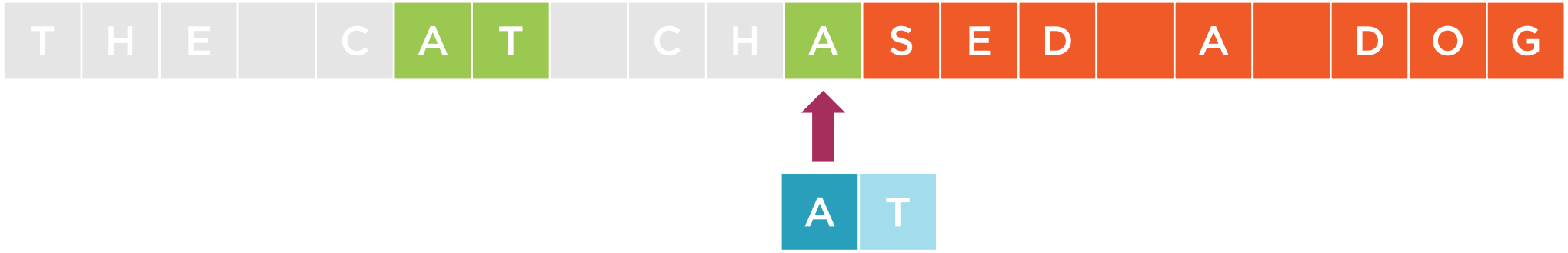
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



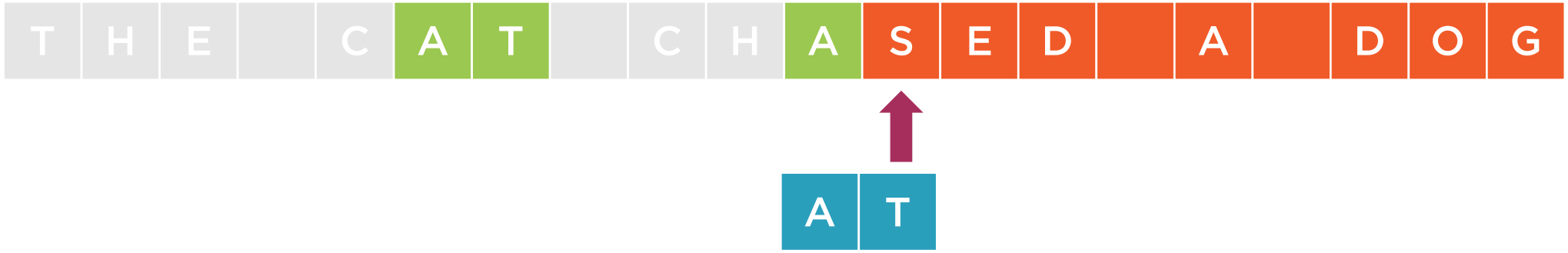
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



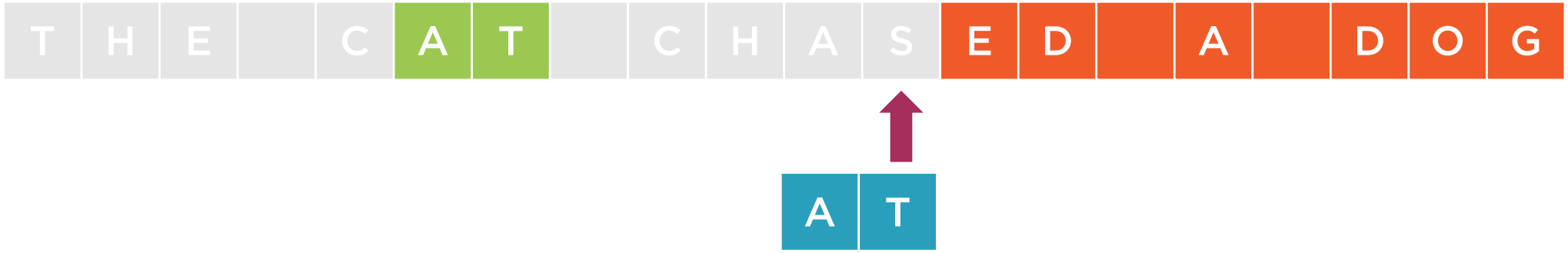
Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```

Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```

Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```

Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```

Naive Search Algorithm

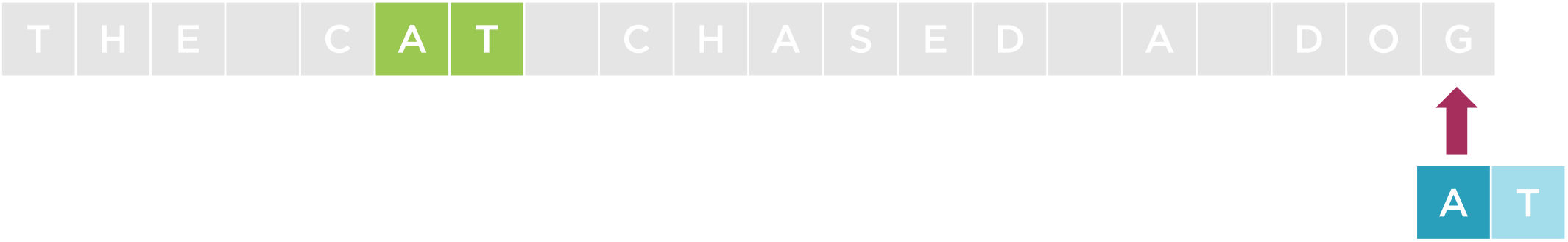
```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```

Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```

Naive Search Algorithm

```
for (startIndex = 0; startIndex < toSearch.Length; startIndex++) {

    matchCount = 0

    while toSearch[startIndex + matchCount] == pattern[matchCount] {
        matchCount++
        if pattern.Length == matchCount
            // Match Found!
            startIndex += matchCount - 1
    }
}
```



Naive Search Performance

O(n+m)	**O(n+m) average performance**
O(nm)	**O(nm) worst case performance**
?	**Requires no pre-processing. Appropriate for small inputs.**

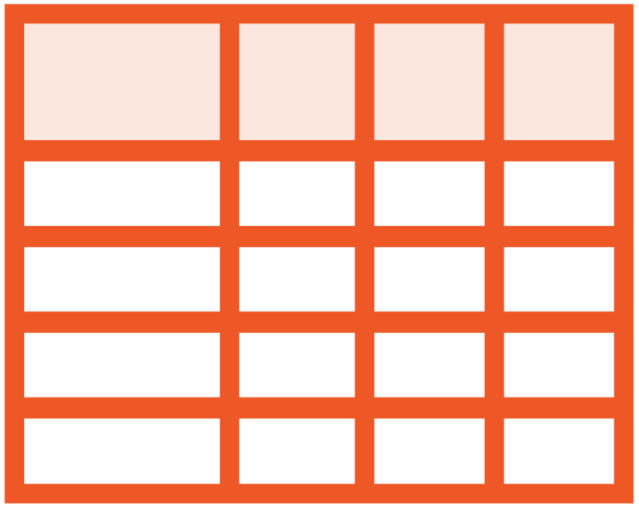
CODE: Naive Search

Boyer-Moore-Horspool

Boyer-Moore-Horspool

A 2-stage searching algorithm that uses a table that contains the length to shift when a bad match occurs.

Boyer-Moore-Horspool



Stage 1

Pre-process the string to find to build a bad match table

Stage 2

The string to find is search right-to-left using the bad match table to skip ahead at a mismatch

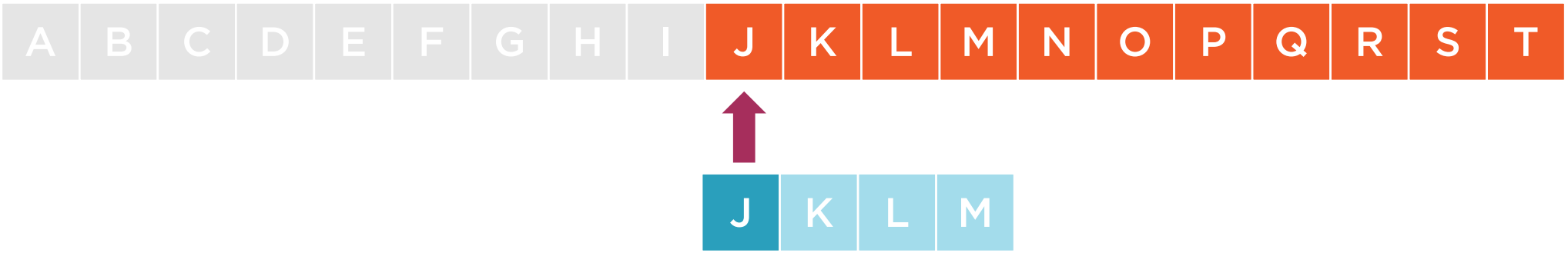
Boyer-Moore-Horspool Algorithm

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

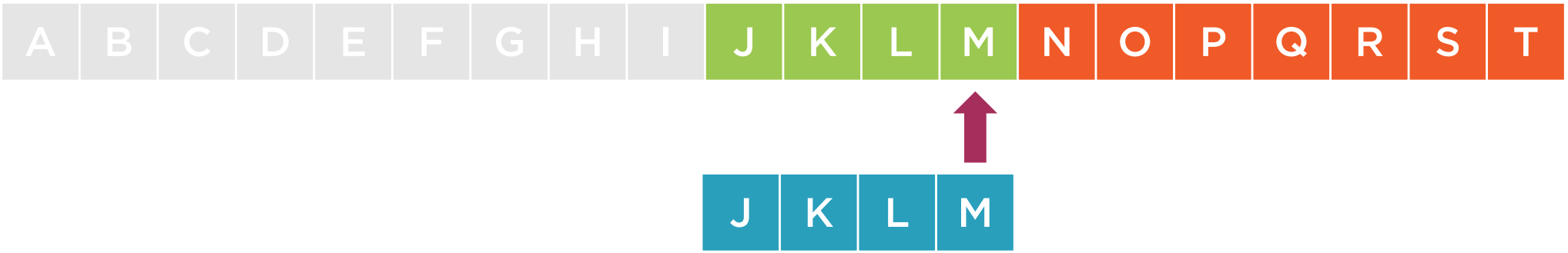
J	K	L	M

Boyer-Moore-Horspool Algorithm

Boyer-Moore-Horspool Algorithm



Boyer-Moore-Horspool Algorithm



Boyer-Moore-Horspool Algorithm

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

J	K	L	M

Boyer-Moore-Horspool Algorithm

Boyer-Moore-Horspool Algorithm

Boyer-Moore-Horspool Algorithm

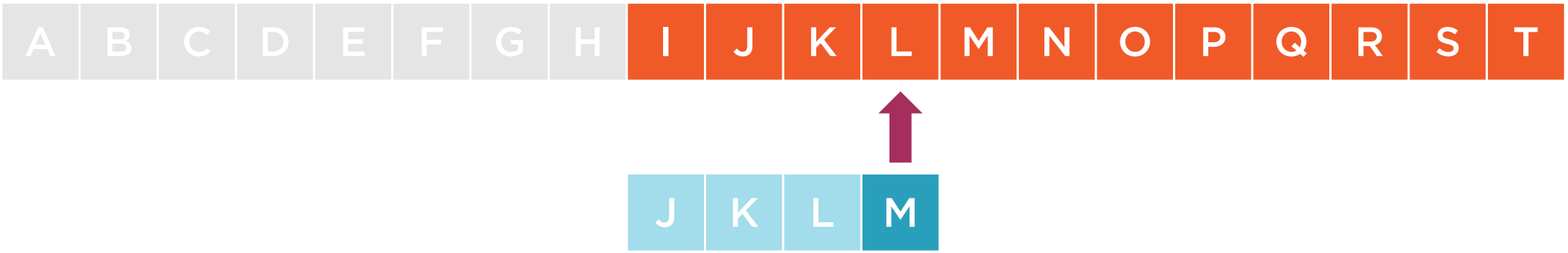
Boyer-Moore-Horspool Algorithm

Boyer-Moore-Horspool Algorithm

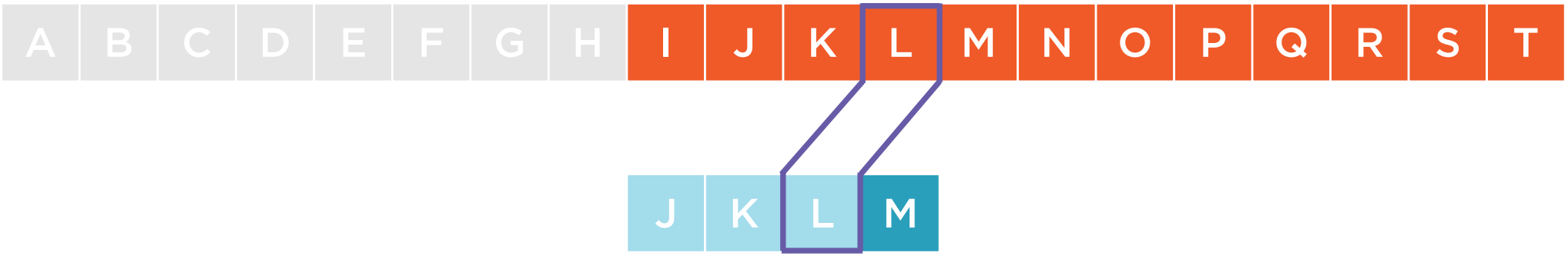
Boyer-Moore-Horspool Algorithm

Boyer-Moore-Horspool Algorithm

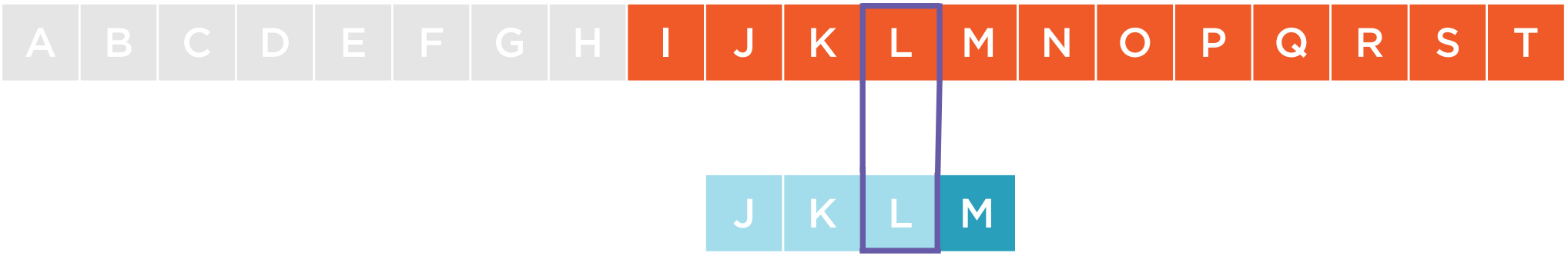
Boyer-Moore-Horspool Algorithm



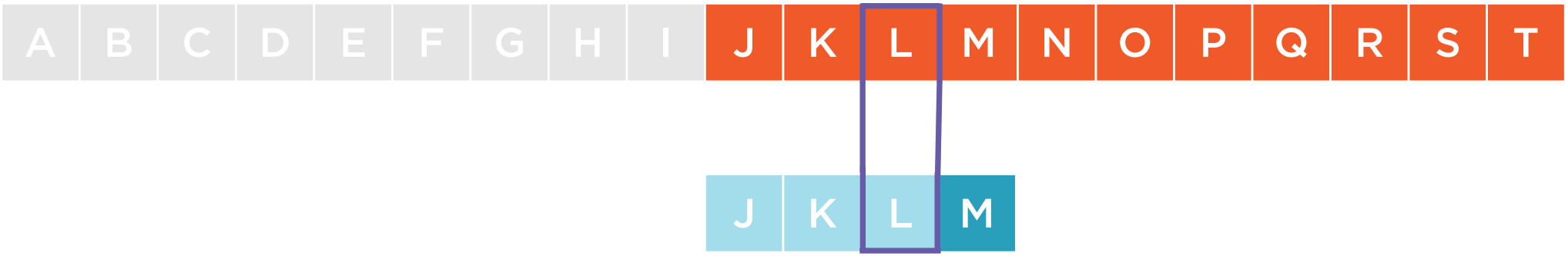
Boyer-Moore-Horspool Algorithm



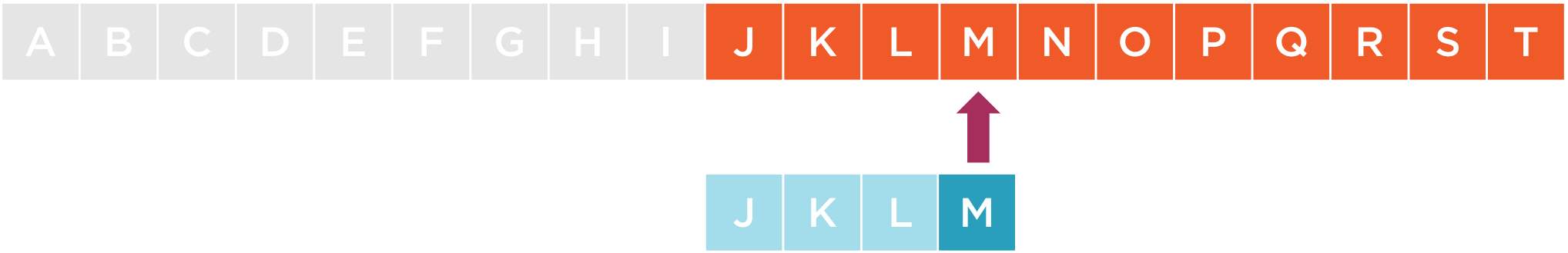
Boyer-Moore-Horspool Algorithm



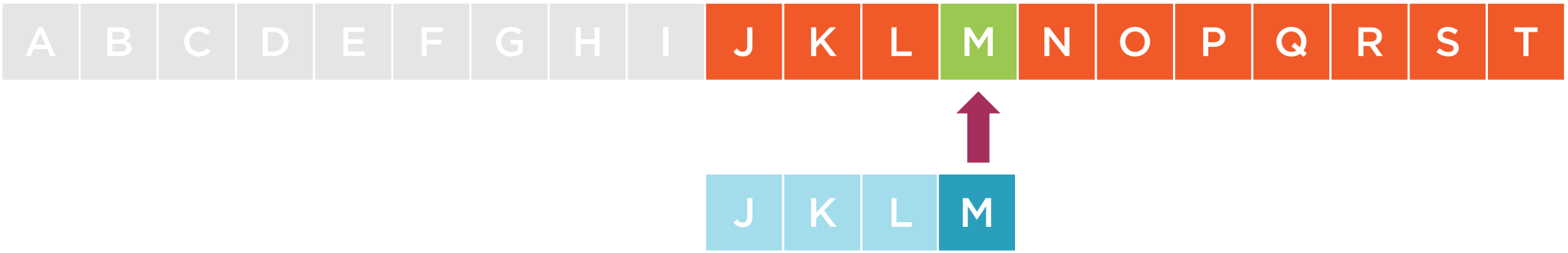
Boyer-Moore-Horspool Algorithm



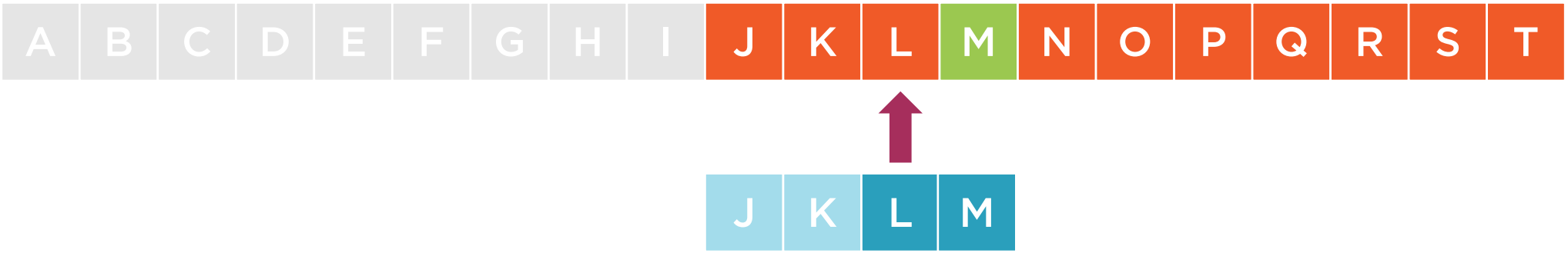
Boyer-Moore-Horspool Algorithm



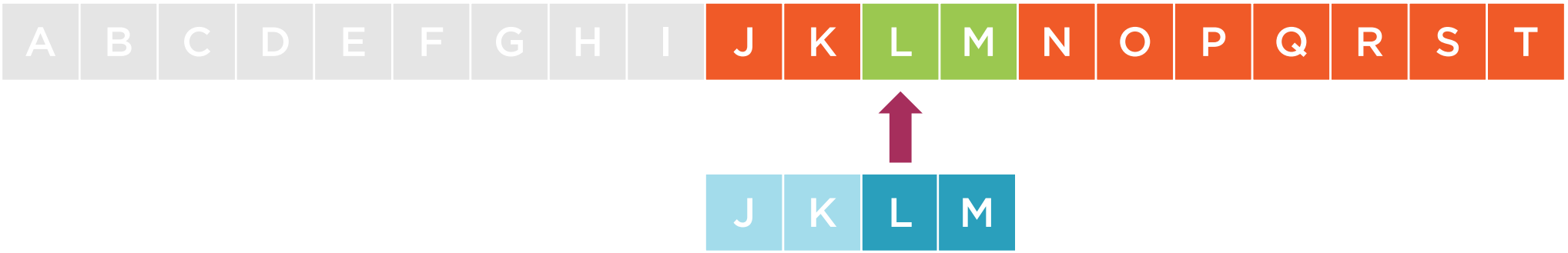
Boyer-Moore-Horspool Algorithm



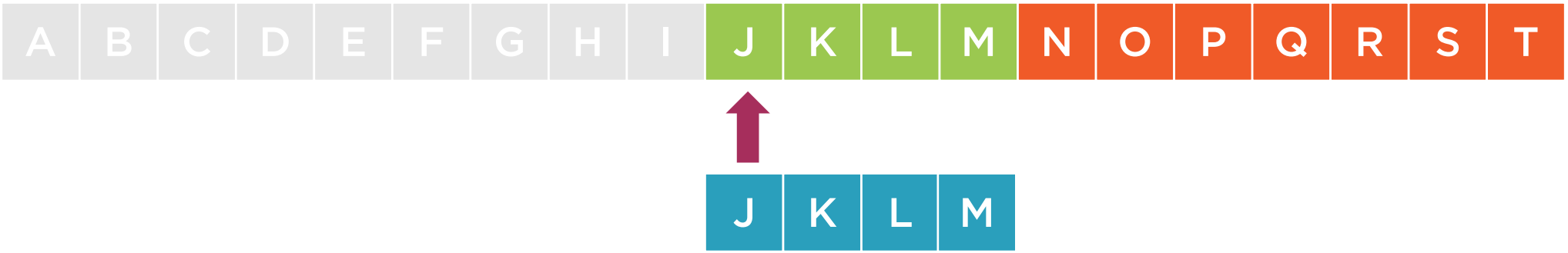
Boyer-Moore-Horspool Algorithm



Boyer-Moore-Horspool Algorithm



Boyer-Moore-Horspool Algorithm



Boyer-Moore-Horspool Algorithm

Boyer-Moore-Horspool Algorithm

Boyer-Moore-Horspool Algorithm

Boyer-Moore-Horspool Algorithm

Bad Match Table

```
class BadMatchTable {
  readonly int missing;
  readonly Dictionary<int, int> table;

  public BadMatchTable(string pattern) {
    missing = pattern.Length;
    table = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++) {
      table[pattern[i]] = pattern.Length - i - 1;
    }
  }

  public int this[int index] {
    get {
      return table.GetValueOrDefault(index, missing);
    }
  }
}
```

- Default value for items not in table
- The table of offsets to shift

- Initialize the table from the pattern

- For each character in the pattern
- Set the offset when that character is seen

- Provide a table lookup accessor

```
class BadMatchTable {
  readonly int missing;
  readonly Dictionary<int, int> table;

  public BadMatchTable(string pattern) {
    missing = pattern.Length;
    table = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++) {
      table[pattern[i]] = pattern.Length - i - 1;
    }
  }

  public int this[int index] {
    get {
      return table.GetValueOrDefault(index, missing);
    }
  }
}
```

T R U T H

```
class BadMatchTable {
  readonly int missing;
  readonly Dictionary<int, int> table;

  public BadMatchTable(string pattern) {
    missing = pattern.Length;
    table = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++) {
      table[pattern[i]] = pattern.Length - i - 1;
    }
  }

  public int this[int index] {
    get {
      return table.GetValueOrDefault(index, missing);
    }
  }
}
```

T	R	U	T	H

Index	Value

```
class BadMatchTable {
  readonly int missing;
  readonly Dictionary<int, int> table;

  public BadMatchTable(string pattern) {
    missing = pattern.Length;
    table = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++) {
      table[pattern[i]] = pattern.Length - i - 1;
    }
  }

  public int this[int index] {
    get {
      return table.GetValueOrDefault(index, missing);
    }
  }
}
```

T	R	U	T	H

Index	Value
?	5

```
class BadMatchTable {
  readonly int missing;
  readonly Dictionary<int, int> table;

  public BadMatchTable(string pattern) {
    missing = pattern.Length;
    table = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++) {
      table[pattern[i]] = pattern.Length - i - 1;
    }
  }

  public int this[int index] {
    get {
      return table.GetValueOrDefault(index, missing);
    }
  }
}
```

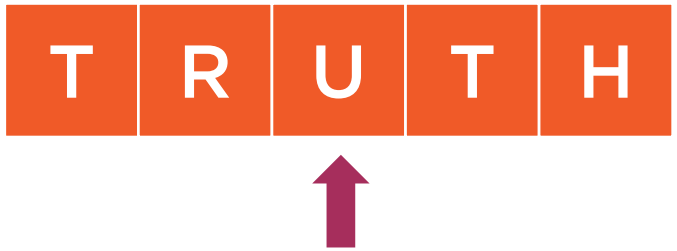
Index	Value
?	5
T	4

```
class BadMatchTable {
  readonly int missing;
  readonly Dictionary<int, int> table;

  public BadMatchTable(string pattern) {
    missing = pattern.Length;
    table = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++) {
      table[pattern[i]] = pattern.Length - i - 1;
    }
  }

  public int this[int index] {
    get {
      return table.GetValueOrDefault(index, missing);
    }
  }
}
```

T	R	U	T	H

Index	Value
?	5
T	4
R	3

```
class BadMatchTable {
  readonly int missing;
  readonly Dictionary<int, int> table;

  public BadMatchTable(string pattern) {
    missing = pattern.Length;
    table = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++) {
      table[pattern[i]] = pattern.Length - i - 1;
    }
  }

  public int this[int index] {
    get {
      return table.GetValueOrDefault(index, missing);
    }
  }
}
```



Index	Value
?	5
T	4
R	3
U	2

```
class BadMatchTable {
  readonly int missing;
  readonly Dictionary<int, int> table;

  public BadMatchTable(string pattern) {
    missing = pattern.Length;
    table = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++) {
      table[pattern[i]] = pattern.Length - i - 1;
    }
  }

  public int this[int index] {
    get {
      return table.GetValueOrDefault(index, missing);
    }
  }
}
```

Index	Value
?	5
T	4
R	3
U	2

```
class BadMatchTable {
  readonly int missing;
  readonly Dictionary<int, int> table;

  public BadMatchTable(string pattern) {
    missing = pattern.Length;
    table = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++) {
      table[pattern[i]] = pattern.Length - i - 1;
    }
  }

  public int this[int index] {
    get {
      return table.GetValueOrDefault(index, missing);
    }
  }
}
```

Index	Value
?	5
T	4
R	3
U	2

```
class BadMatchTable {
  readonly int missing;
  readonly Dictionary<int, int> table;

  public BadMatchTable(string pattern) {
    missing = pattern.Length;
    table = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++) {
      table[pattern[i]] = pattern.Length - i - 1;
    }
  }

  public int this[int index] {
    get {
      return table.GetValueOrDefault(index, missing);
    }
  }
}
```

Index	Value
?	5
T	1
R	3
U	2

```
class BadMatchTable {
  readonly int missing;
  readonly Dictionary<int, int> table;

  public BadMatchTable(string pattern) {
    missing = pattern.Length;
    table = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++) {
      table[pattern[i]] = pattern.Length - i - 1;
    }
  }

  public int this[int index] {
    get {
      return table.GetValueOrDefault(index, missing);
    }
  }
}
```

Index	Value
?	5
T	1
R	3
U	2

```
class BadMatchTable {
  readonly int missing;
  readonly Dictionary<int, int> table;

  public BadMatchTable(string pattern) {
    missing = pattern.Length;
    table = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++) {
      table[pattern[i]] = pattern.Length - i - 1;
    }
  }

  public int this[int index] {
    get {
      return table.GetValueOrDefault(index, missing);
    }
  }
}
```

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

T	H	E		T	R	U	T	H		I	S		O	U	T		T	H	E	R	E

Boyer-Moore-Horspool Algorithm

T	H	E		T	R	U	T	H		I	S		O	U	T		T	H	E	R	E

T	R	U	T	H

Boyer-Moore-Horspool Algorithm

T	H	E		T	R	U	T	H		I	S		O	U	T		T	H	E	R	E

T	R	U	T	H

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

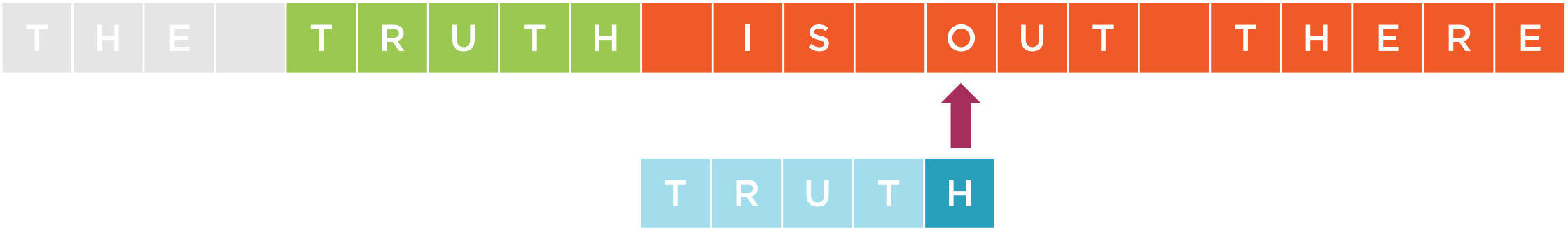
Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

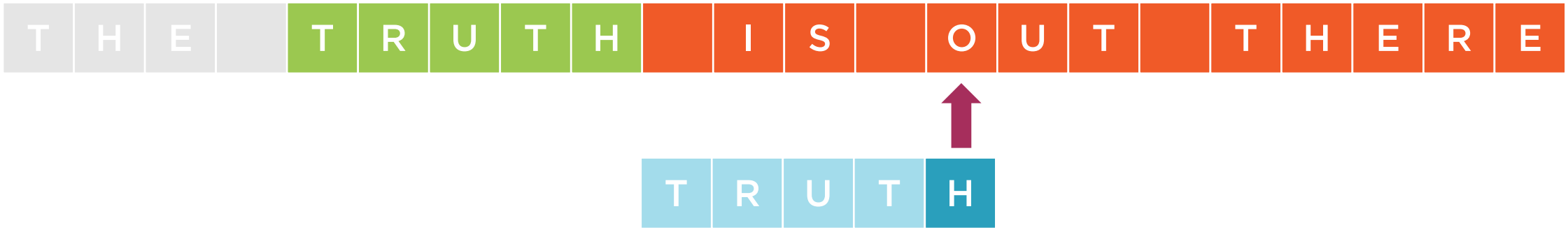
Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm



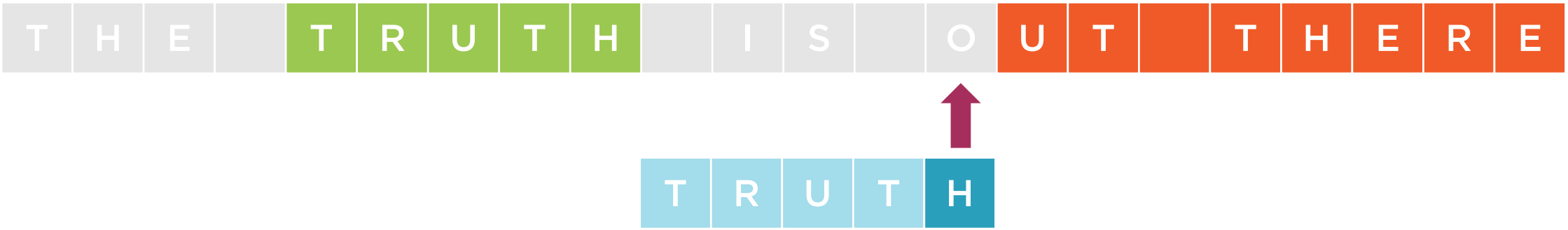
Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm



Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm



Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Performance

O(n/m) | **O(n) average performance**

O(nm) | **O(nm) worst case performance**

? | **The larger the bad match table the better the performance**

Demo

Review Boyer-Moore-Horspool Algorithm

Demo: Text search and replace tool