String Algorithms and Data Structures

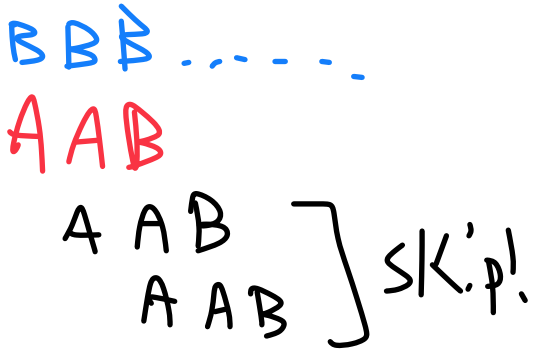
Boyer-Moore

CS 199-225

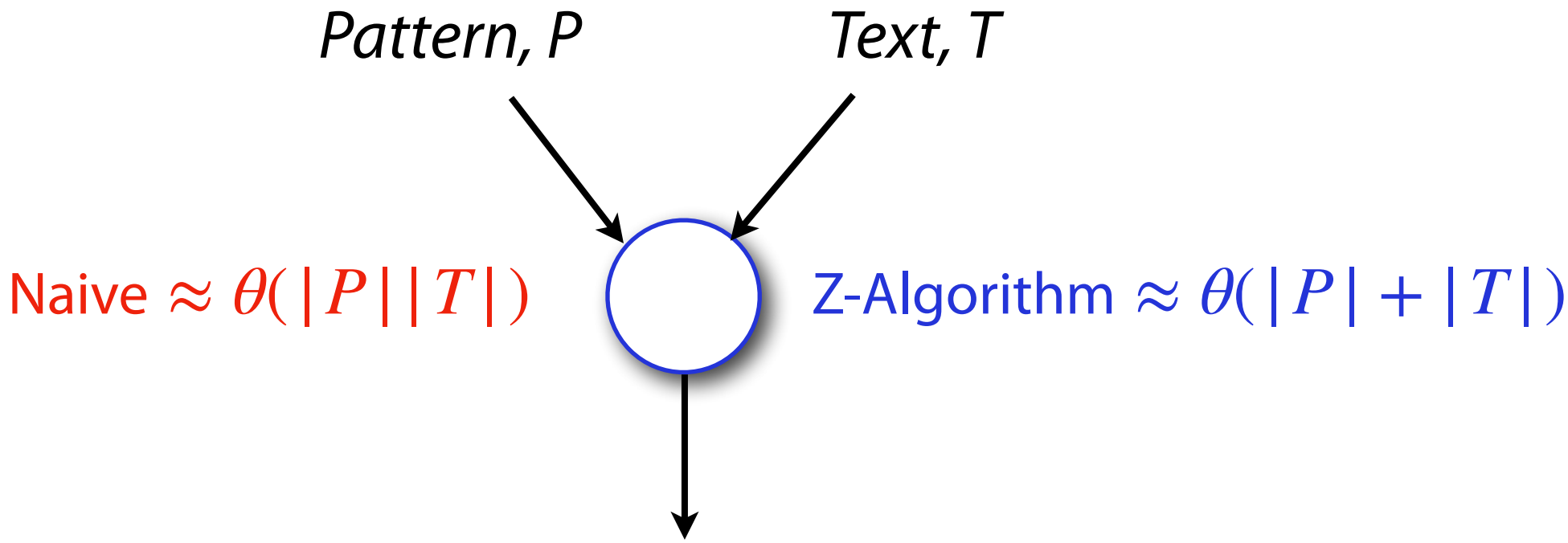
Brad Solomon

February 16, 2026

Department of Computer Science



Exact Pattern Matching w/ Z-algorithm



'instances': An exact, full length copy

Why continue?

The Z-algorithm is:

- The Z-algorithm is: $O(|P| + |T|)$ time
- An alphabet-independent solution

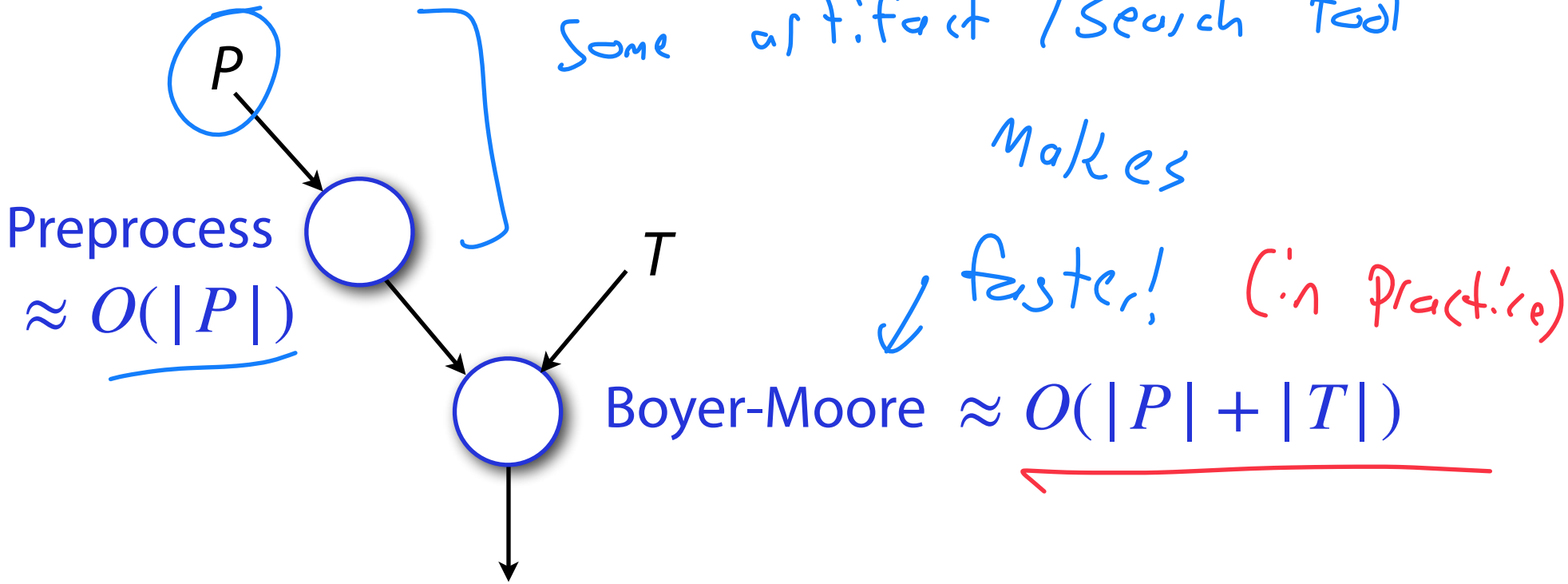
The Z-algorithm is less good at:

- Searching for a ***set*** of patterns (Aho-Corasick)
- Running in *sub-linear** time (Boyer-Moore)

* — in practice, not theory

Exact pattern matching w/ Boyer-Moore

Boyer Moore **preprocesses** the pattern



Find instances of *P* in *T*

'instances': An exact, full length copy

Boyer-Moore

Z-Alg Learning from past to speed up present

Intuition: Learn from alignments to avoid others

Boyer Moore looks at current to skip future work

P: c a t

T: c a r l c a r r i e d t h e c a t

c a t ----------►

0 1 2 3 4 5 6 7 8 9 …

What does this alignment tell us?

Boyer-Moore

Intuition: Learn from alignments to avoid others

P: c a t

T: c a r l c a r r i e d t h e c a t

c a t ----------→

0 1 2 3 4 5 6 7 8 9 …

What does this alignment tell us?

1) Our pattern doesn't match at this alignment

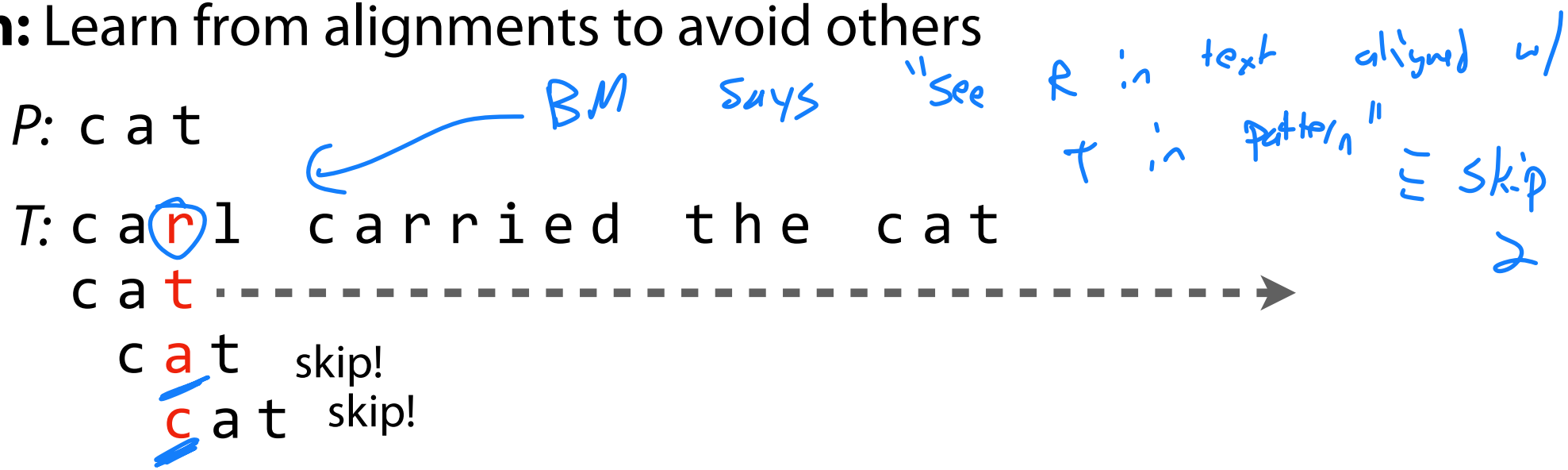
Boyer-Moore

Intuition: Learn from alignments to avoid others

What does this alignment tell us?

Boyer-Moore

Intuition: Learn from alignments to avoid others

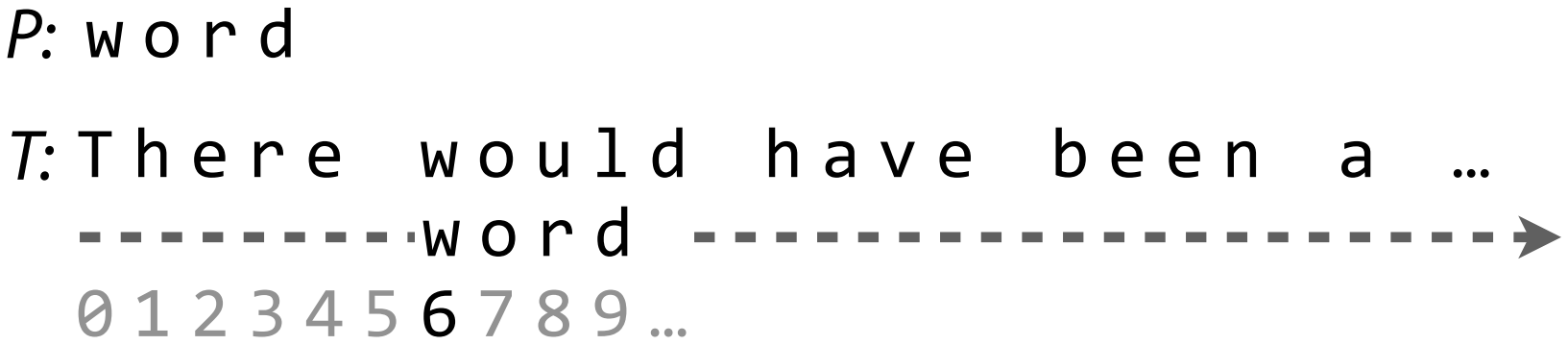


What does this alignment tell us?

2) Our pattern doesn't match at *later* alignments

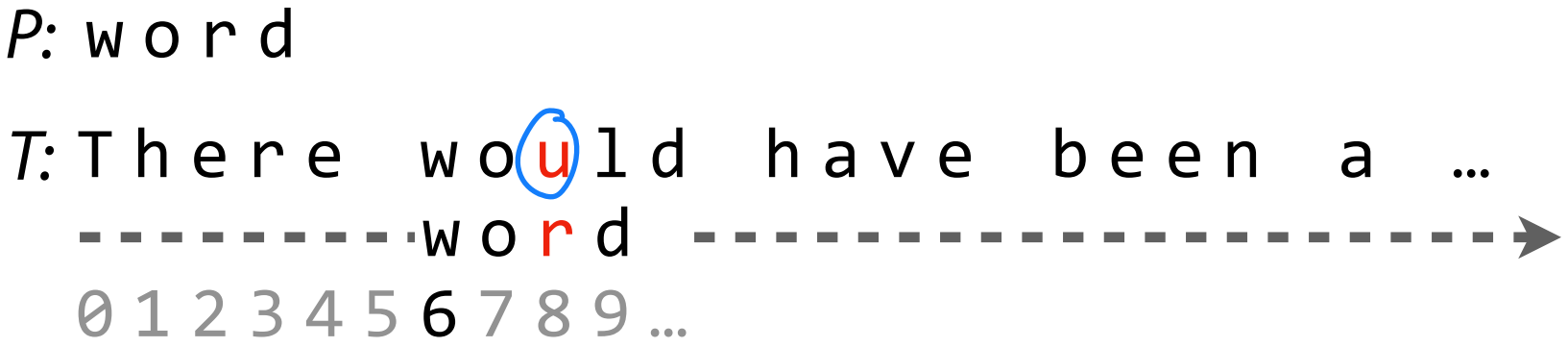
Boyer-Moore

Intuition: Learn from alignments to avoid others



Boyer-Moore

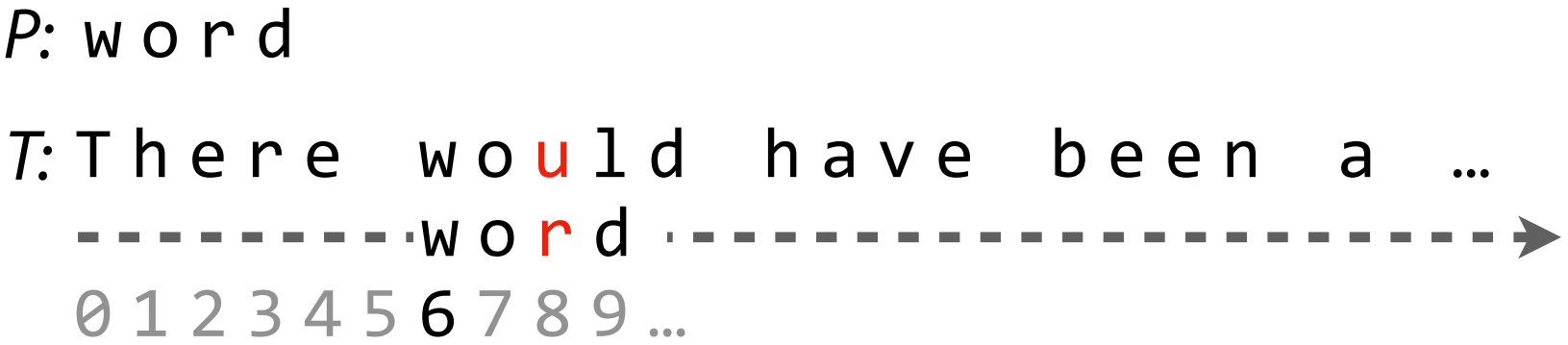
Intuition: Learn from alignments to avoid others



How many alignments can we skip?

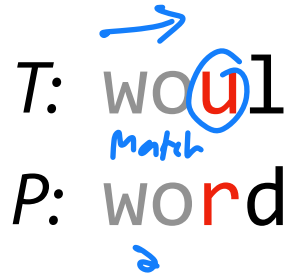
Boyer-Moore

Intuition: Learn from alignments to avoid others



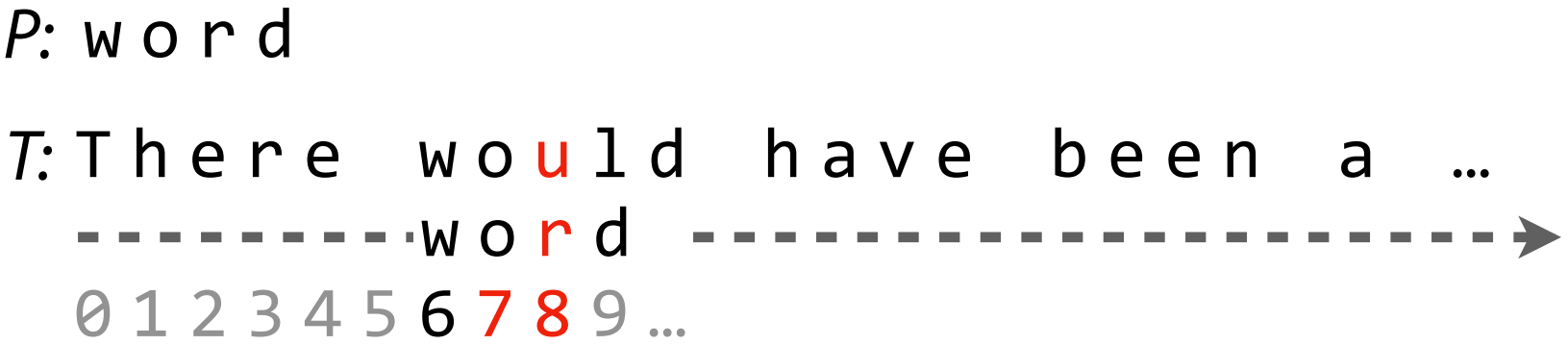
How many alignments can we skip?

1) Our pattern doesn't match at this alignment



Boyer-Moore

Intuition: Learn from alignments to avoid others

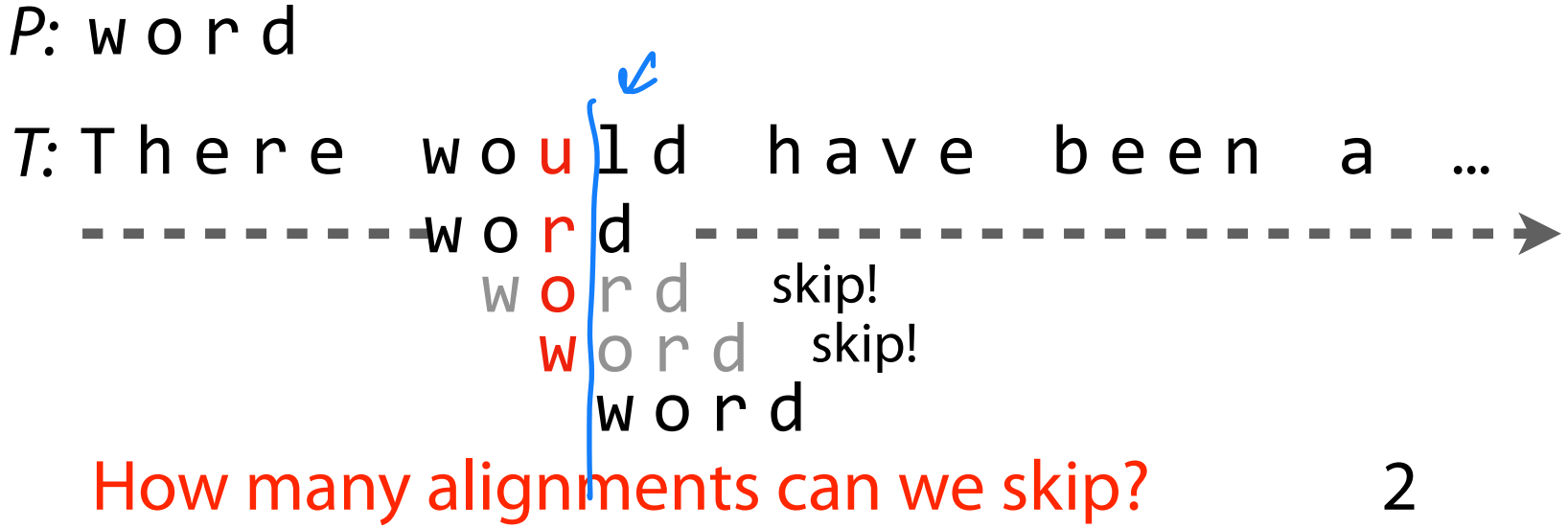


How many alignments can we skip? 2

2) Our pattern doesn't match at *later* alignments

Boyer-Moore

Intuition: Learn from alignments to avoid others



2) Our pattern doesn't match at *later* alignments

Boyer-Moore

Intuition: Learn from alignments to avoid others

How many alignments can we skip?

Boyer-Moore

Intuition: Learn from alignments to avoid others

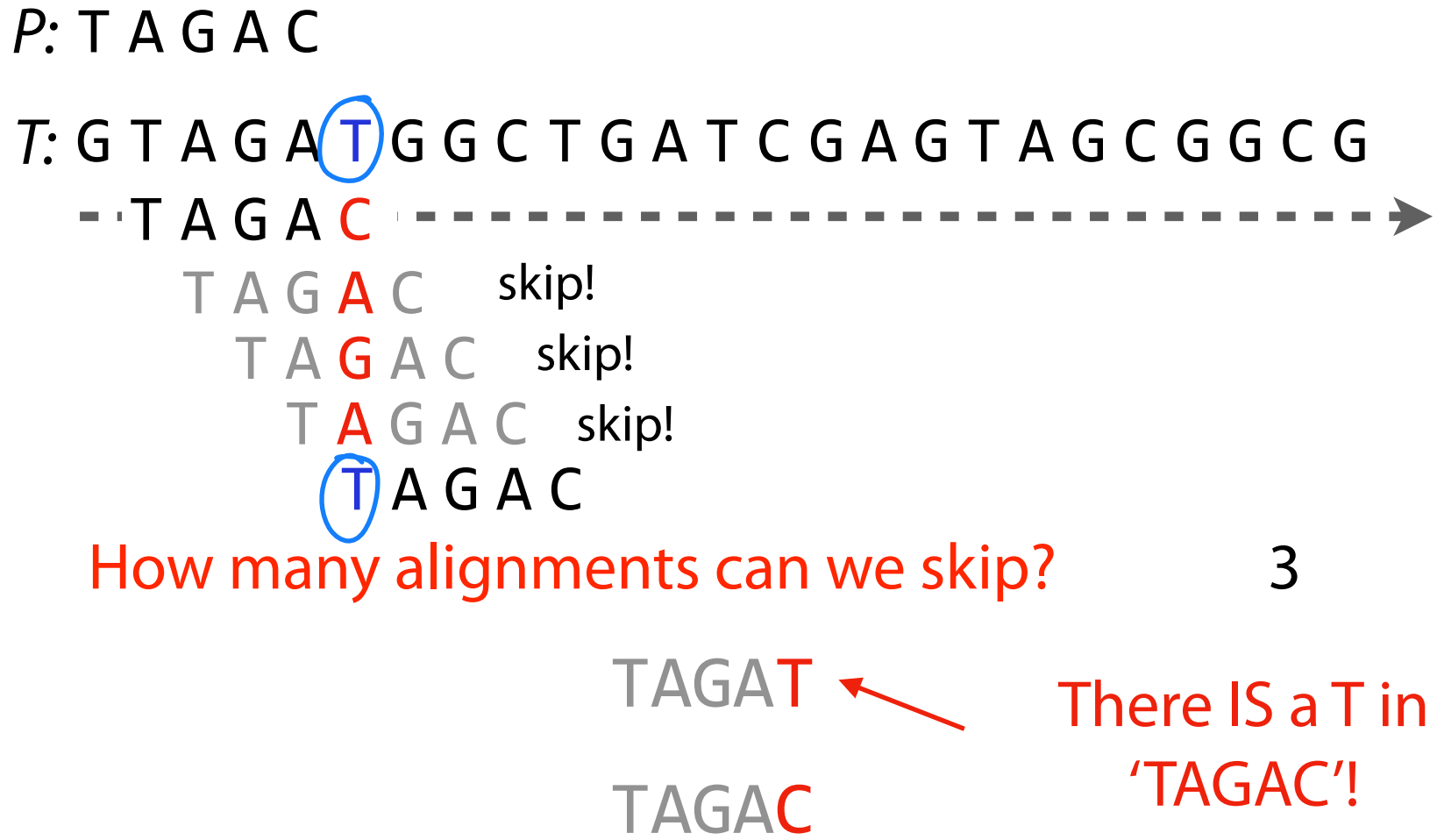
How many alignments can we skip? 3

TAGAT ← There IS a T in 'TAGAC'!

TAGAC

Boyer-Moore

Intuition: Learn from alignments to avoid others



Boyer-Moore

Intuition: Learn from alignments to avoid others

P: A A B B B

T: A A A B A B A A A A A A A A A A A A A A A A A A

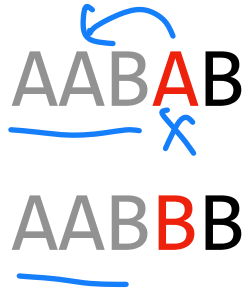
A A B B B

How many alignments can we skip?

Boyer-Moore

Intuition: Learn from alignments to avoid others

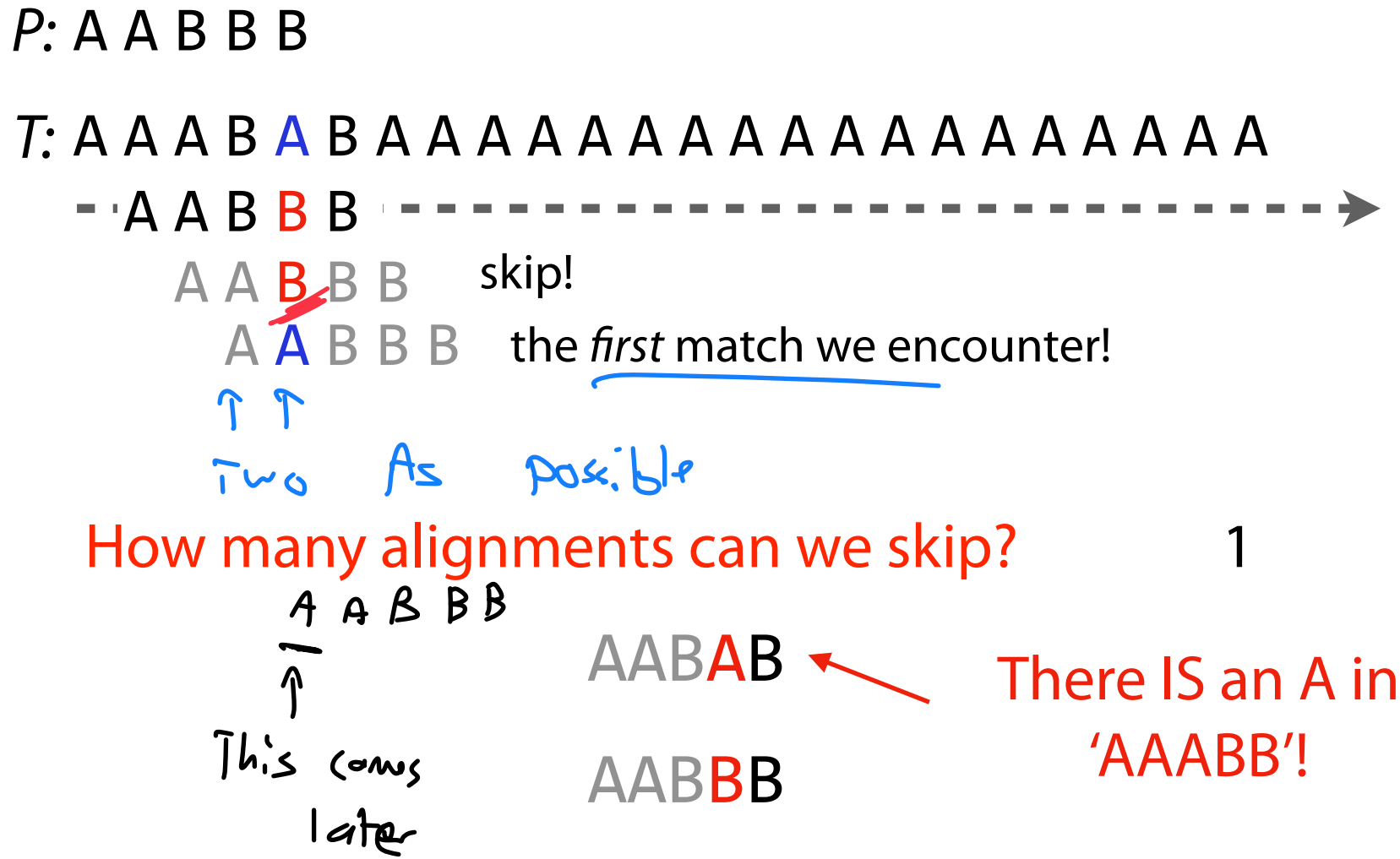
How many alignments can we skip? 1



There IS an A in 'AAABB'!

Boyer-Moore

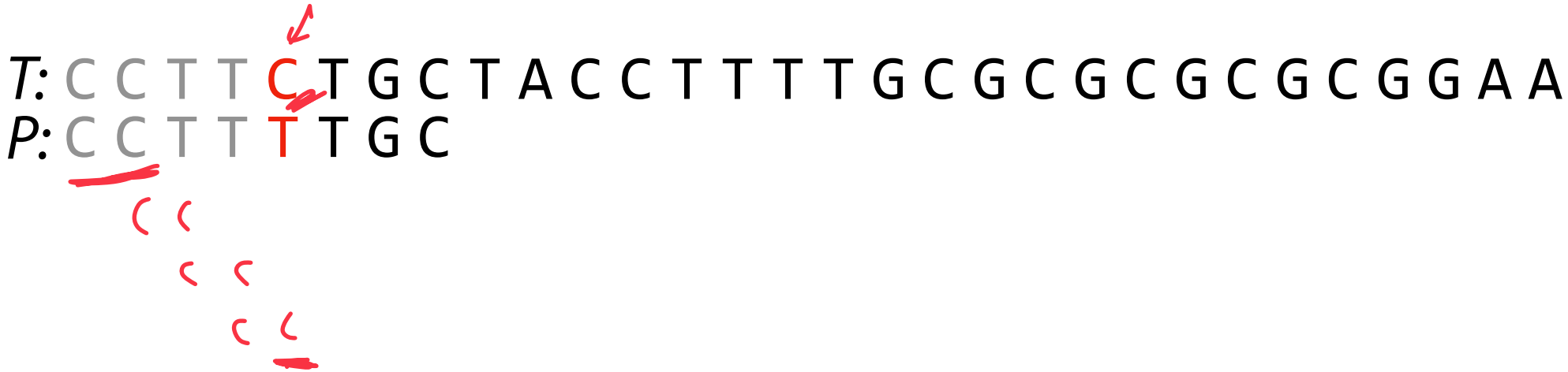
Intuition: Learn from alignments to avoid others



Boyer-Moore: Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) *P* moves past mismatched character.

Step 1:



Boyer-Moore: Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) *P* moves past mismatched character.

Step 1:

Boyer-Moore: Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) *P* moves past mismatched character.

Step 1:

T: C C T T C T G C T A C C T T T T G C G C G C G C G G A A

P: C C T T T T G C

Case (a)

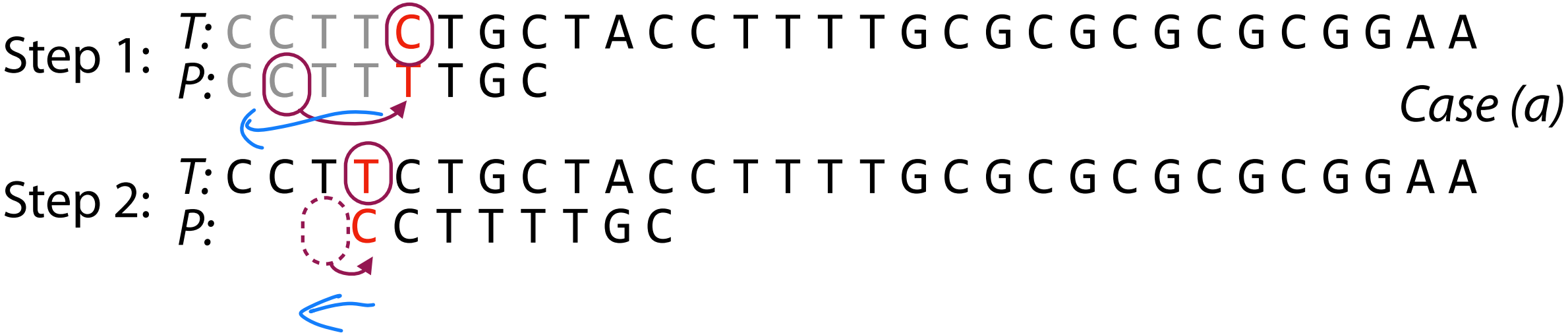
Step 2:

T: C C T T C T G C T A C C T T T T G C G C G C G C G G A A

P: ↑ × × C C T T T T G C

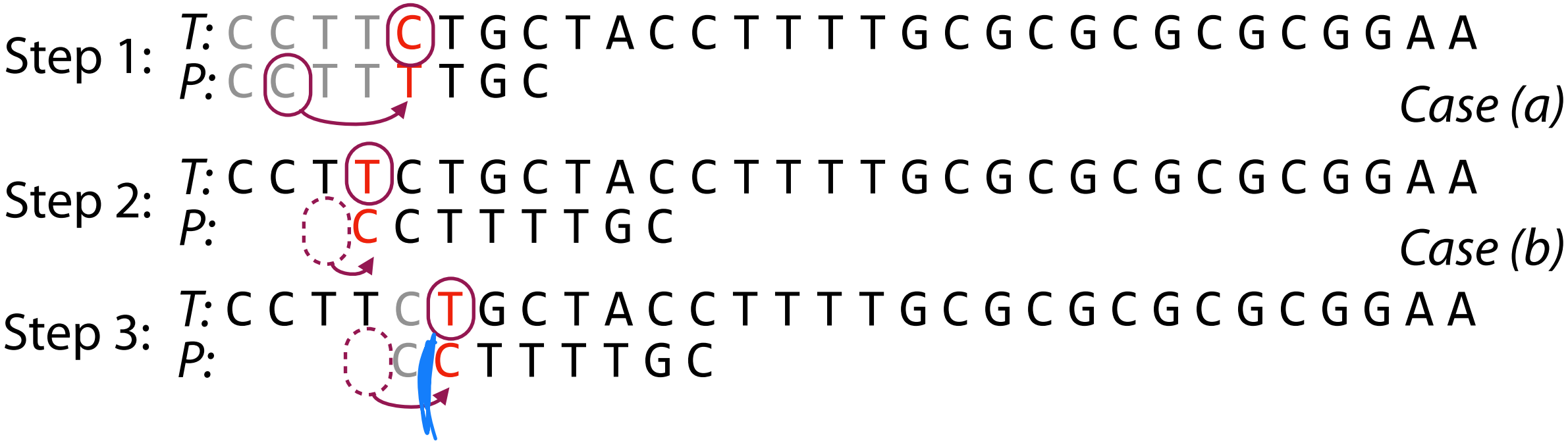
Boyer-Moore: Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) *P* moves past mismatched character.



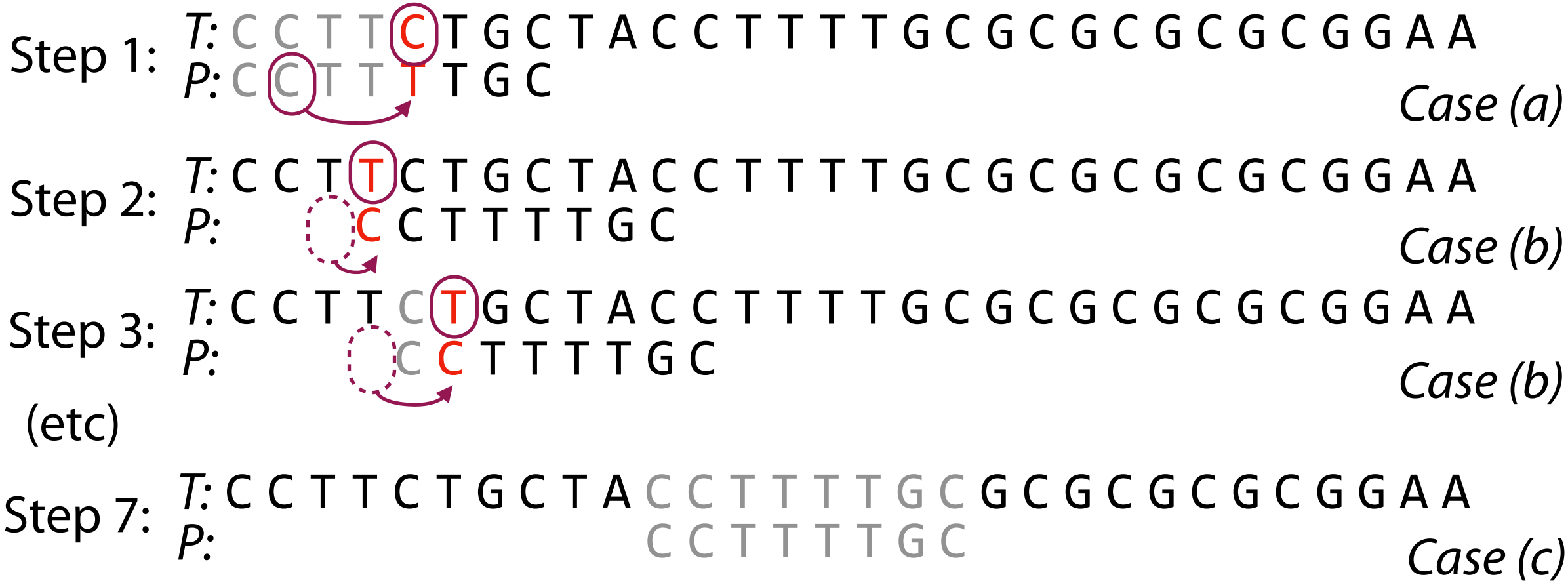
Boyer-Moore: Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) *P* moves past mismatched character.



Boyer-Moore: Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) *P* moves past mismatched character. (c) If no mismatch, don't skip



Boyer-Moore: Bad Character rule

Step 1:

T: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A

P: C C T T T T G C

C C T T T T G C

C C T T T T G C

skip!

Step 2:

T: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A

P: C C T T T T G C

C C T T T T G C

skip nothing?

Step 3:

T: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A

P: C C T T T T G C

C C T T T T G C

skip!

Boyer-Moore: Bad Character rule

Which of the following alignments skips the most?

Join Code: 225

A)

T: TATAT…

P: TAGAC

B)

T: TTGAT…

P: TAGAC

C)

T: TAGAT…

P: TAGAC

D)

T: TAGTT…

P: TAGAC

Boyer-Moore: Bad Character rule improvement

↳ A proposed heuristic improvement

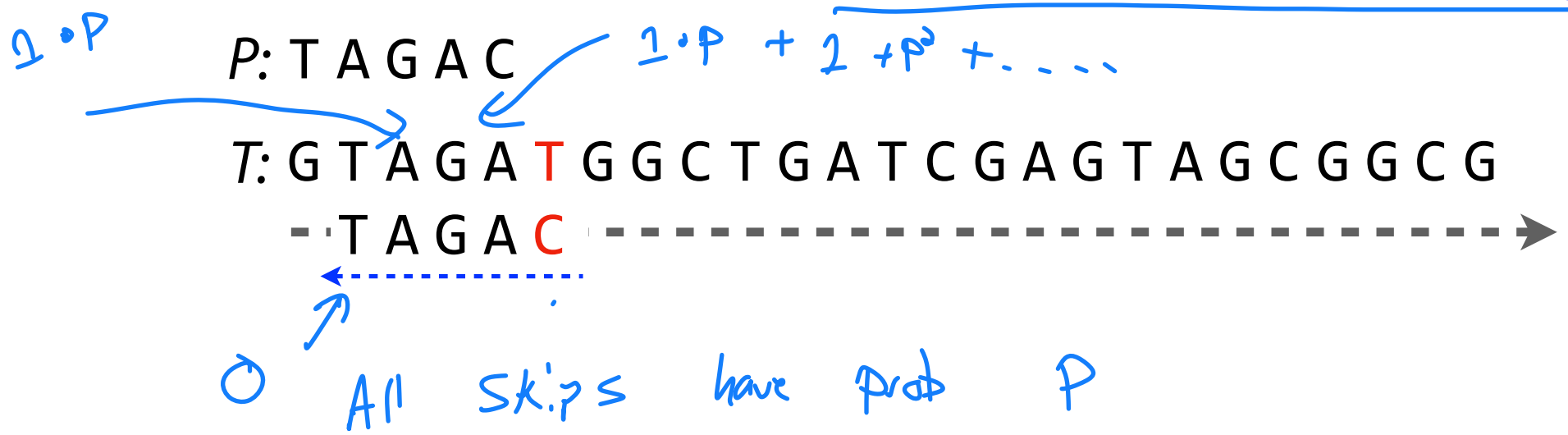
Continue to test alignment from left-to-right

↑ characters at end of string are most useful* (potentially)

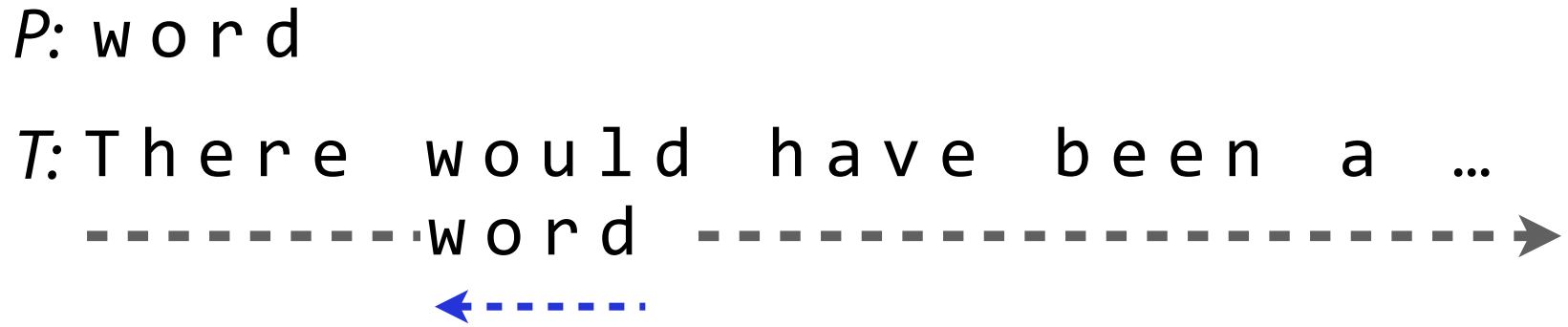
Boyer-Moore: Bad Character rule improvement

Continue to test alignment from left-to-right

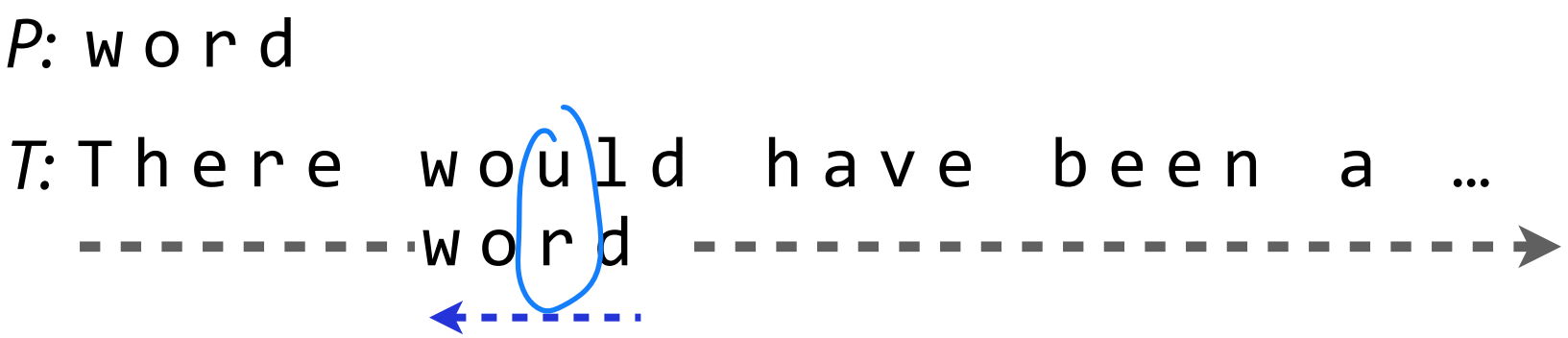
… but compare ***characters*** from right to left.



Right-to-left-scanning w/ BC Rule



Right-to-left-scanning w/ BC Rule

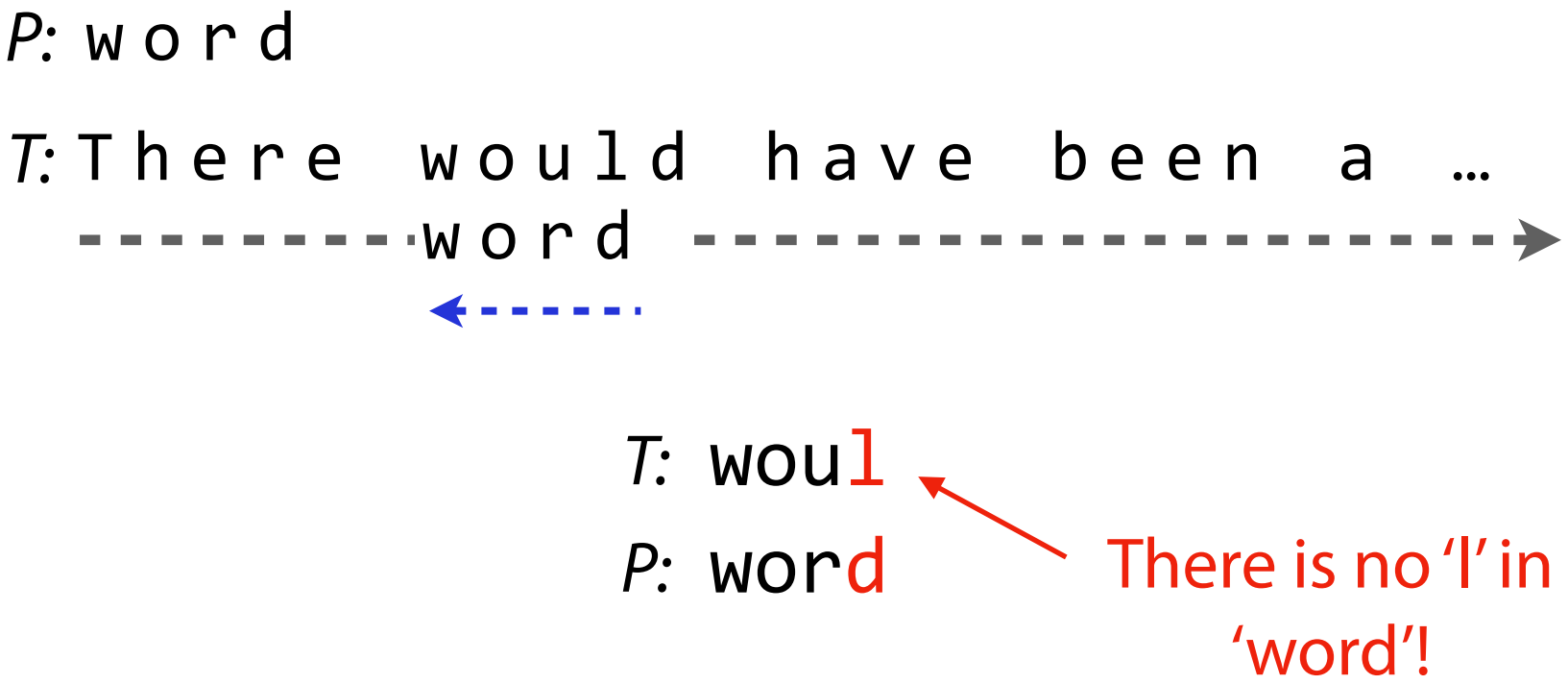


T: woul

P: word

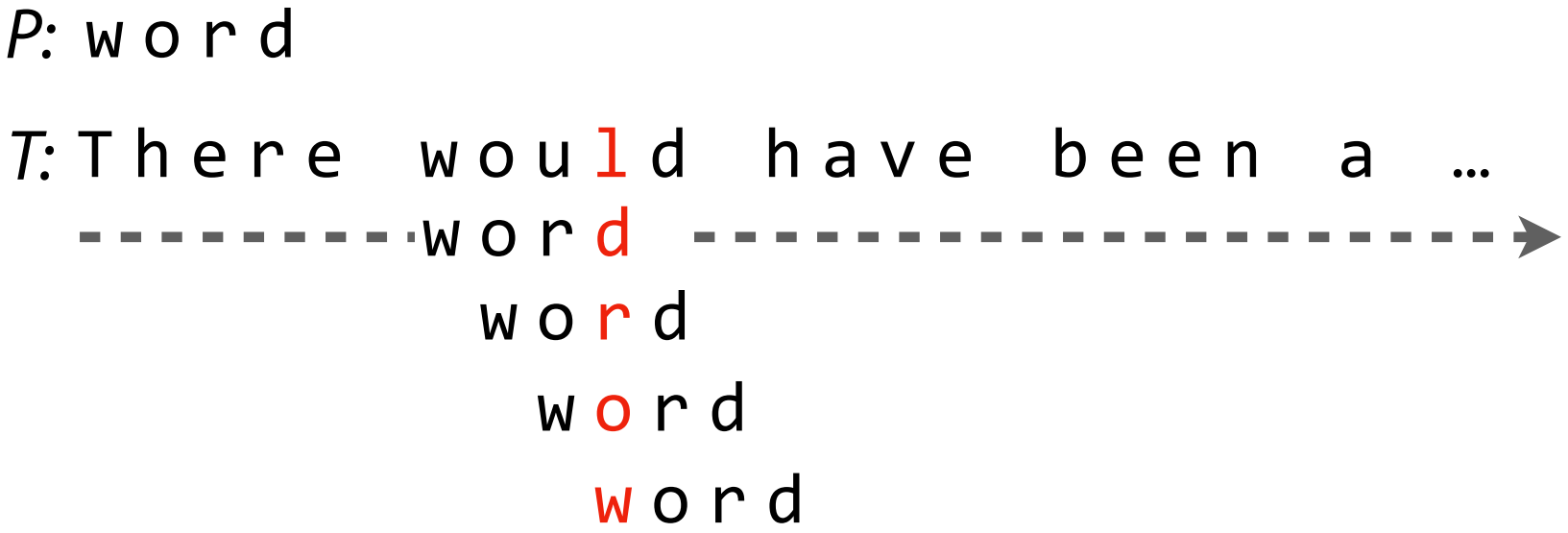
There is no 'l' in 'word'!

Right-to-left-scanning w/ BC Rule



How many alignments do we skip?

Right-to-left-scanning w/ BC Rule



How many alignments do we skip? 3

Right-to-left-scanning w/ BC Rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) *P* moves past mismatched character. (c) If no mismatch, don't skip

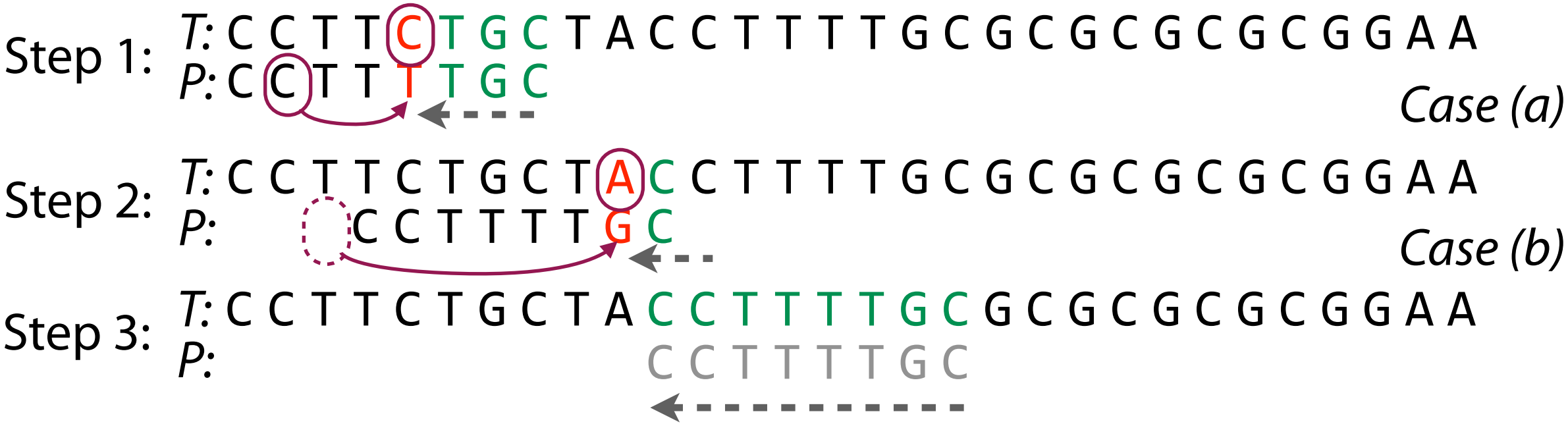
Step 1:

Right-to-left-scanning w/ BC Rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) *P* moves past mismatched character. (c) If no mismatch, don't skip

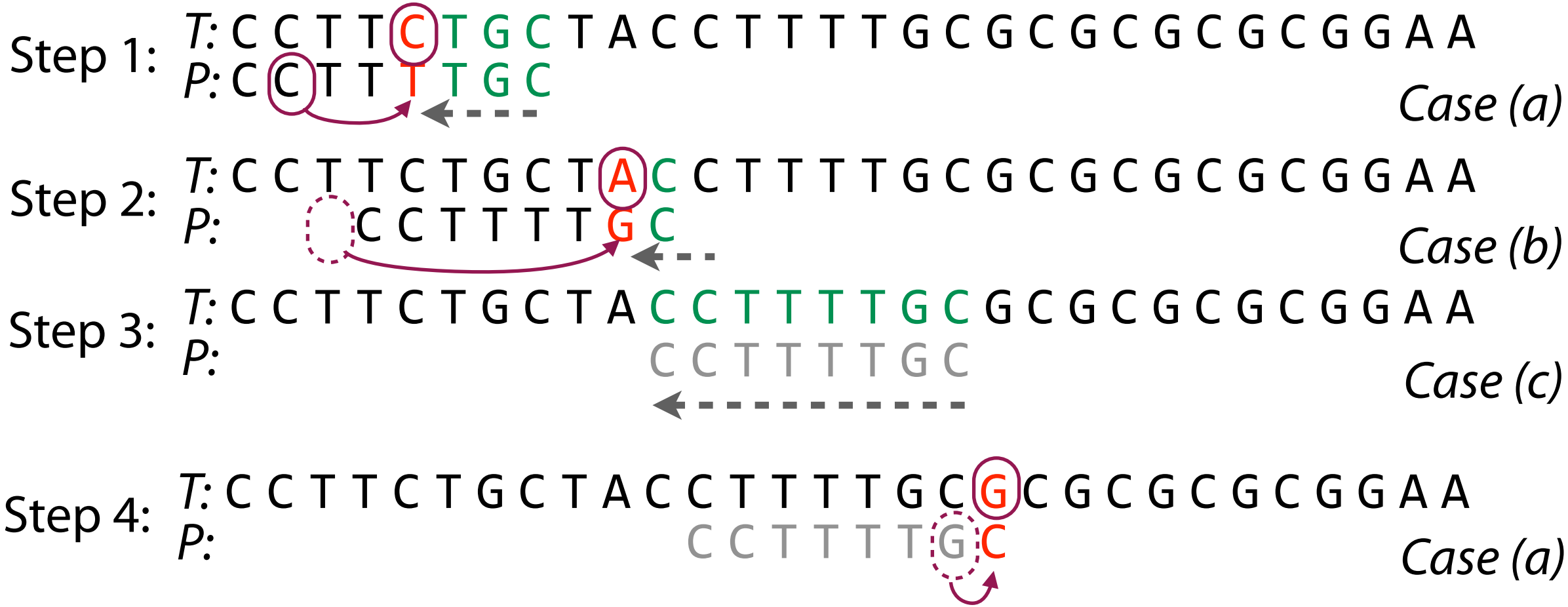
Right-to-left-scanning w/ BC Rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) *P* moves past mismatched character. (c) If no mismatch, don't skip



Right-to-left-scanning w/ BC Rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) *P* moves past mismatched character. (c) If no mismatch, don't skip



Right-to-left-scanning w/ BC Rule

Step 1:
T: C C T T C T G C T A C C T T T T G C G C G C G C G G A A
P: C C T T T T G C

Step 2:
T: C C T T C T G C T A C C T T T T G C G C G C G C G G A A
P: C C T T T T G C

0 1 2 3 4 5 6 7 8 9 10

Step 3:
T: C C T T C T G C T A C C T T T T G C G C G C G C G G A A
P: C C T T T T G C

Up to step 3, we skipped 8 alignments

C C T T C T G C T A C C T T T T G C G C G C G C G G A A

5 characters in *T* were *never* looked at

C C T T C T G C T A C C T T T T G C G C G C G C G G A A

Right-to-left-scanning w/ BC Rule

Learn from character comparisons to skip pointless alignments

1. When we hit a mismatch *c*, move *P* along until *c* becomes a match (or *P* moves past *c*)

"Bad character rule"

2. Try alignments in one direction, but do character comparisons in *opposite* direction

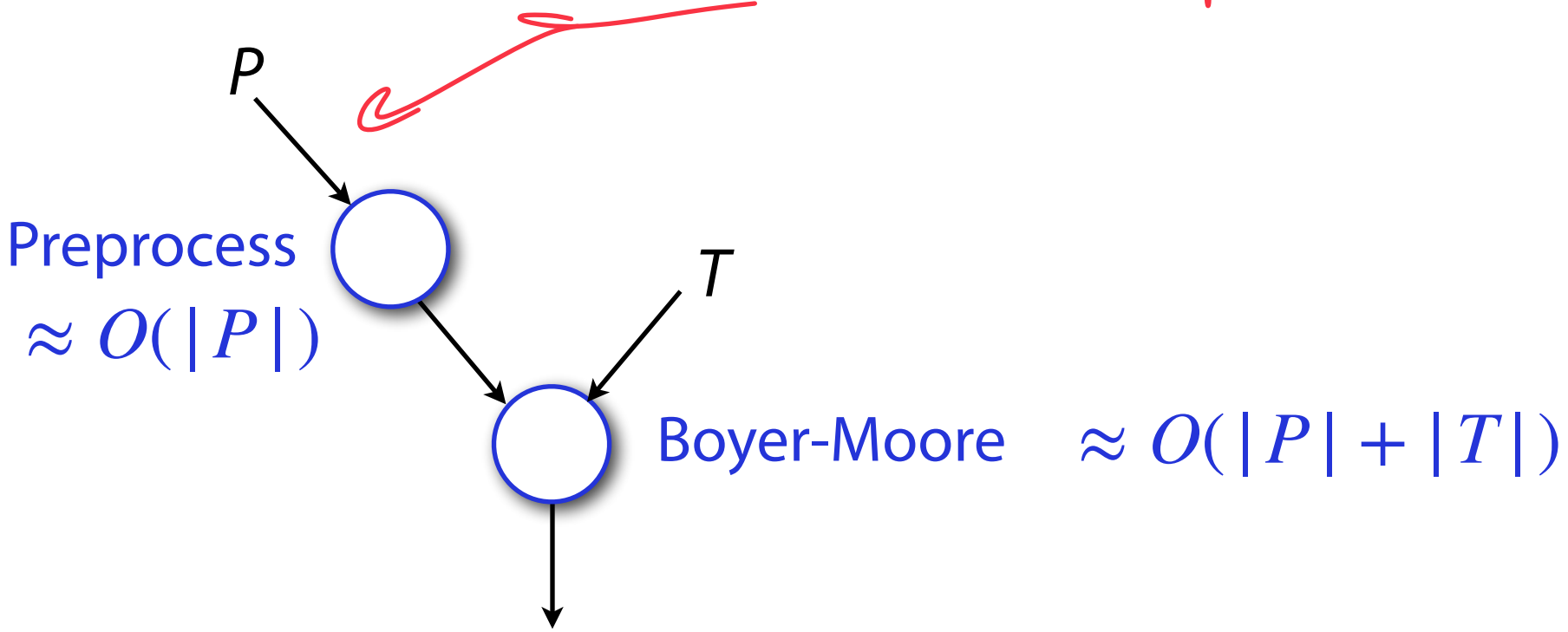
"Right-to-left scanning"

How do we put the first two rules in practice?

Boyer, RS and Moore, JS. "A fast string searching algorithm." *Communications of the ACM* 20.10 (1977): 762-772.

Exact pattern matching w/ Boyer-Moore

Boyer Moore **preprocesses** the pattern & the alphabet



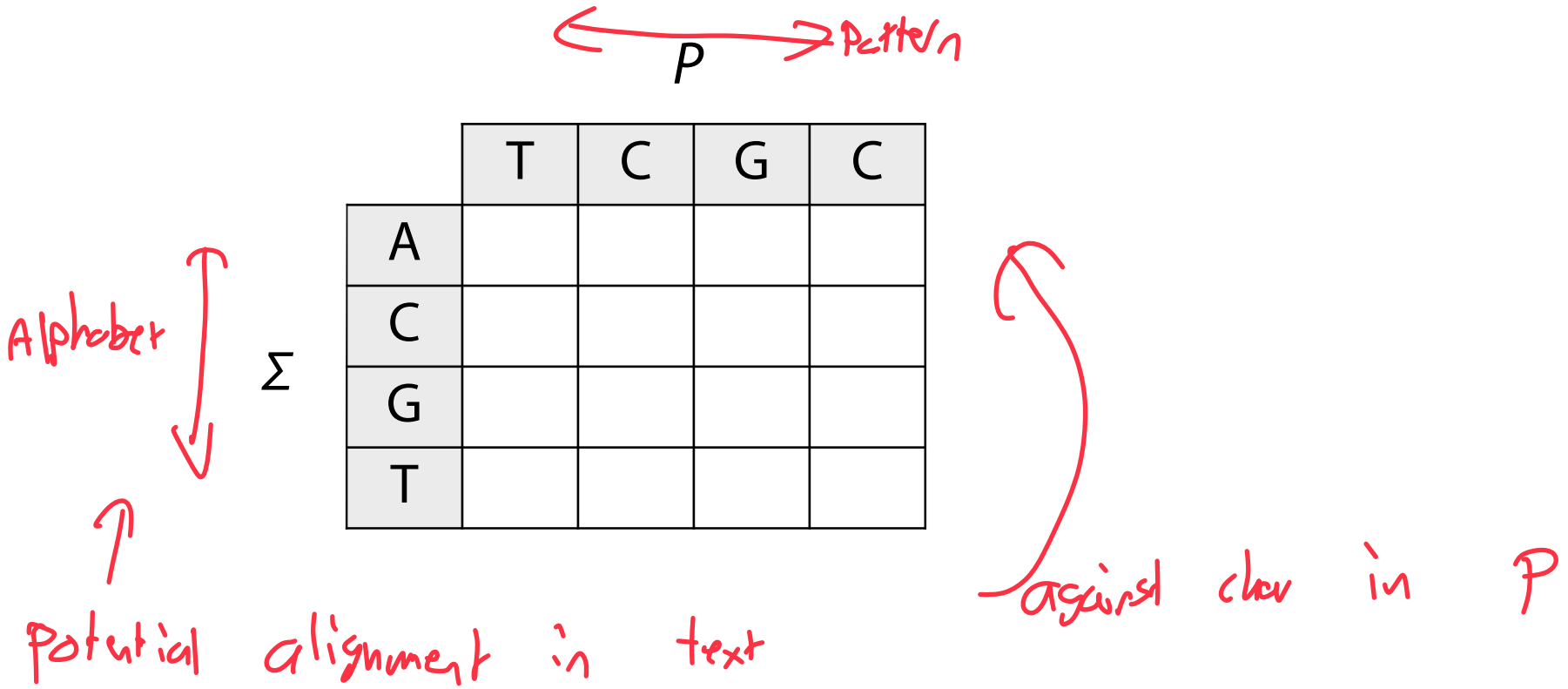
Find instances of *P* in *T*

'instances': An exact, full length copy

Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: *P*: T C G C *Σ*: A C G T

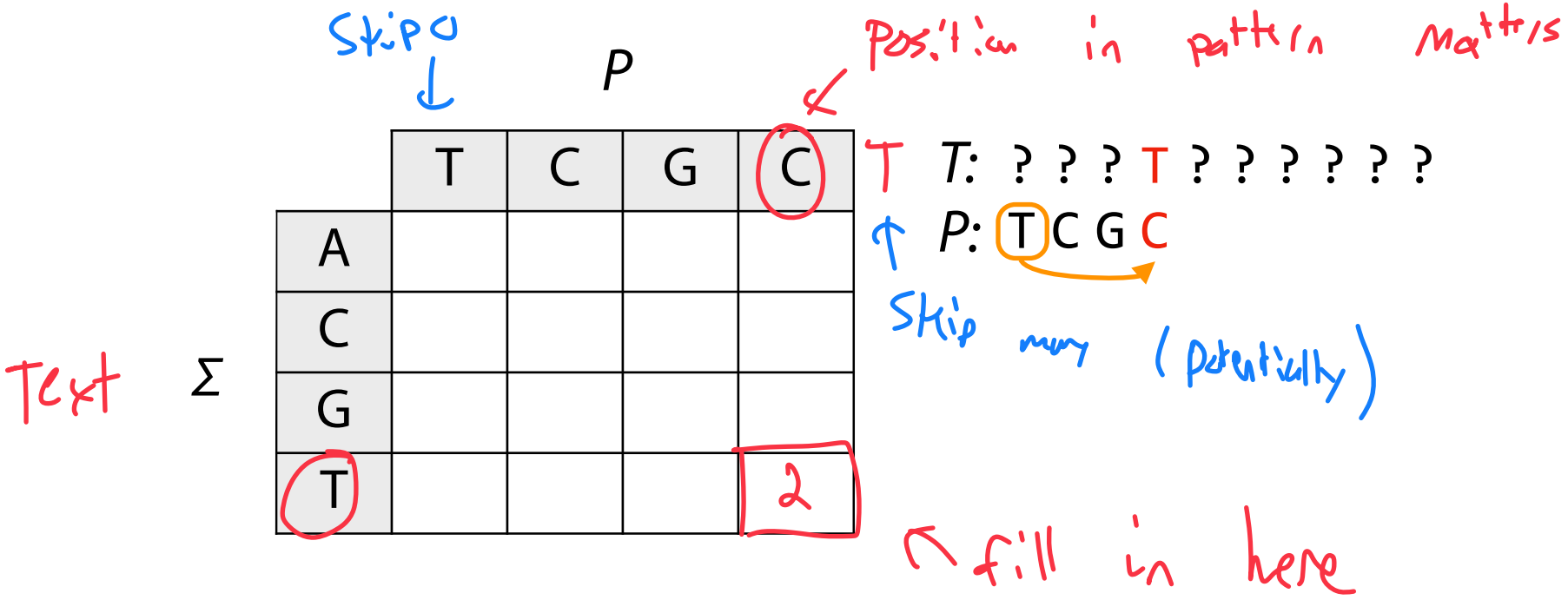
The goal is to produce a table which tracks *skips*



Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: *P*: T C G C *Σ*: A C G T

The goal is to produce a table which tracks *skips*



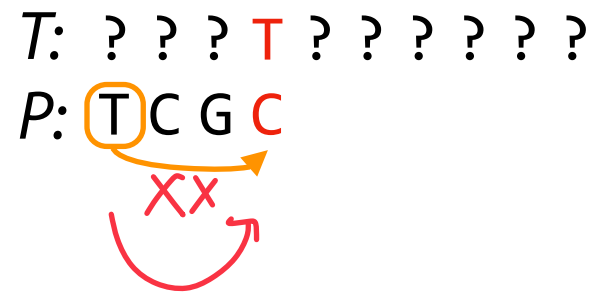
Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: *P:* T C G C *Σ:* A C G T

The goal is to produce a table which tracks *skips*

P

Σ	T	C	G	C
A				
C				
G				
T				2



Boyer-Moore: BC rule preprocessing

Join Code: 225

Preprocessing requires two args: *P*: T C G C *Σ*: A C G T

The goal is to produce a table which tracks *skips*

This is specific matching character by index

Text gives me row | ε letter

P

Σ	T	C	G	C	
A				3	0
C					1
G					2
T				2	3
	X	Y	Z	Q	

T: ? ? ? A ? ? ? ? ? ?
P: T C G C

T C G C X
T C G C X
T C G C X
T C G C

Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: *P:* T C G C *Σ:* A C G T

The goal is to produce a table which tracks *skips*

P

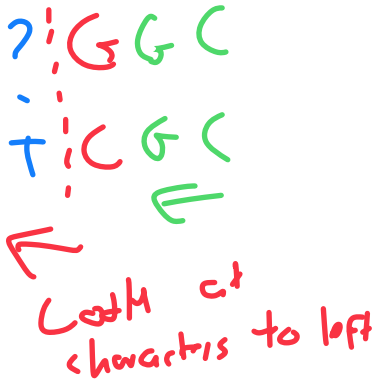
Σ	T	C	G	C
A				3
C				
G				
T				2

T: ? ? ? A ? ? ? ? ? ?
P: T C G C

Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: *P:* T C G C *Σ:* A C G T

The goal is to produce a table which tracks *skips*



P

Σ	T	C	G	C
A	0	1	2	3
C	0	-	0	-
G	0	1	-	0
T	-	0	1	2

T: ? ? A ? ? ? ? ? ? ?
P: T C G C

T: ? ? C ? ? ? ? ? ?
P: T C G C

T: ? ? G ? ? ? ? ? ?
P: T C G C

This is not a mismatch
−1

T: ? ? T ? ? ? ? ? ?
P: T C G C

Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: *P:* B A B A A A B *Σ:* A B

Pattern

Σ		B	A	B	A	A	A	B
	A							
	B							

Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: *P*: B A B A A A B *Σ*: A B

For each character p in pattern *P*

For each character c in alphabet *Σ*

Find the closest previous instance of p (to the left of c).

$O(|P| \cdot |\Sigma|)$

if $\leftarrow O(1)$

Pattern

Σ		B	A	B	A	A	A	B
	A	0	-1					
	B	-1	0					

Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: *P*: B A B A A A B *Σ*: A B

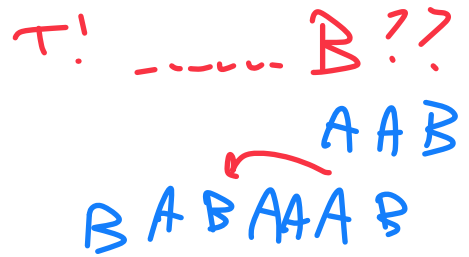
For each character p in pattern *P*

For each character c in alphabet *Σ*

Find the closest previous instance of p (to the left of c).

Pattern

Σ	B	A	B	A	A	A	B
A	0	-1	0	-1	-1	-1	
B	-1	0	-1	0	1	2	



Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: *P:* B A B A A A B *Σ:* A B

For each character p in pattern *P*

For each character c in alphabet *Σ*

Find the closest previous instance of p (to the left of c).

Pattern

Σ		B	A	B	A	A	A	B
	A	0	-1	0	-1	-1	-1	0
	B	-1	0	-1	0	1	2	-1

Assignment 4: a_bmoore

Learning Objective:

Implement preprocessing of patterns with Boyer-Moore*

Observe Boyer-Moore* efficiency *as a heuristic*

Consider: Optimal preprocessing is $\theta(|P||\Sigma|)$. Can you code it?

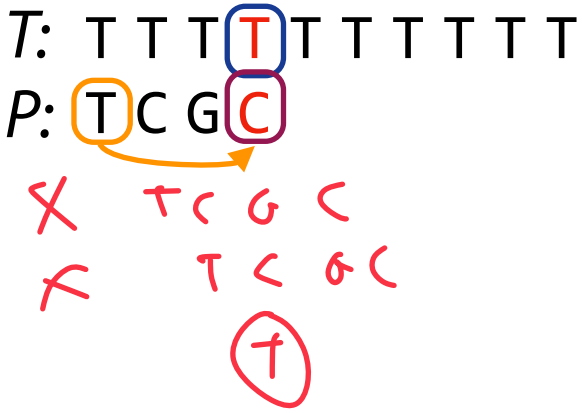
Boyer-Moore: Using the BC Table

Try alignments from left-to-right and match characters from right-to-left

When we encounter a mismatch, skip the calculated number of alignments

P

Σ	T	C	G	C
A	0	1	2	3
C	0	-	0	-
G	0	1	-	0
T	-	0	1	2



O(1) lookup in a table

Boyer-Moore: Using the BC Table

Try alignments from left-to-right and match characters from right-to-left

When we encounter a mismatch, skip the calculated number of alignments

P

Σ	T	C	G	C
A	0	1	2	3
C	0	-	0	-
G	0	1	-	0
T	-	0	1	2

T: G G G G G G G G G G

P: T C G C

Boyer-Moore: Using the BC Table

Try alignments from left-to-right and match characters from right-to-left

When we encounter a mismatch, skip the calculated number of alignments

P

Σ	T	C	G	C
A	0	1	2	3
C	0	-	0	-
G	0	1	-	0
T	-	0	1	2

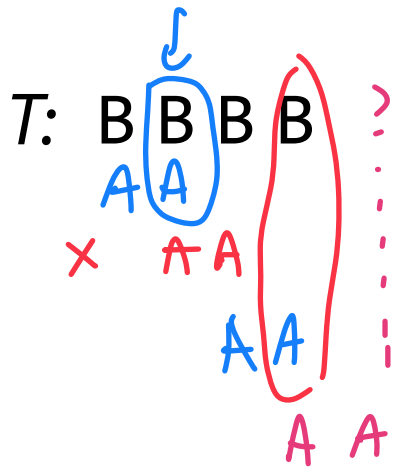
T: A A T C A A T A G C
P: T C G C

Boyer-Moore: Tracking total skips

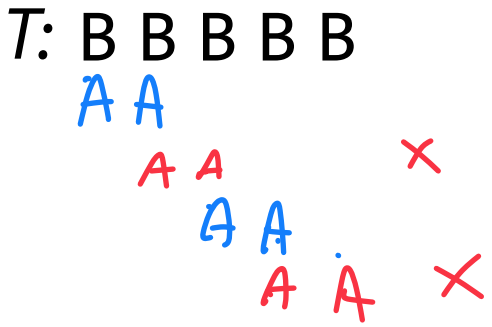
Join Code: 225

P

Σ	A	A
A	-1	-1
B	0	1



1 skip only!



2 skips

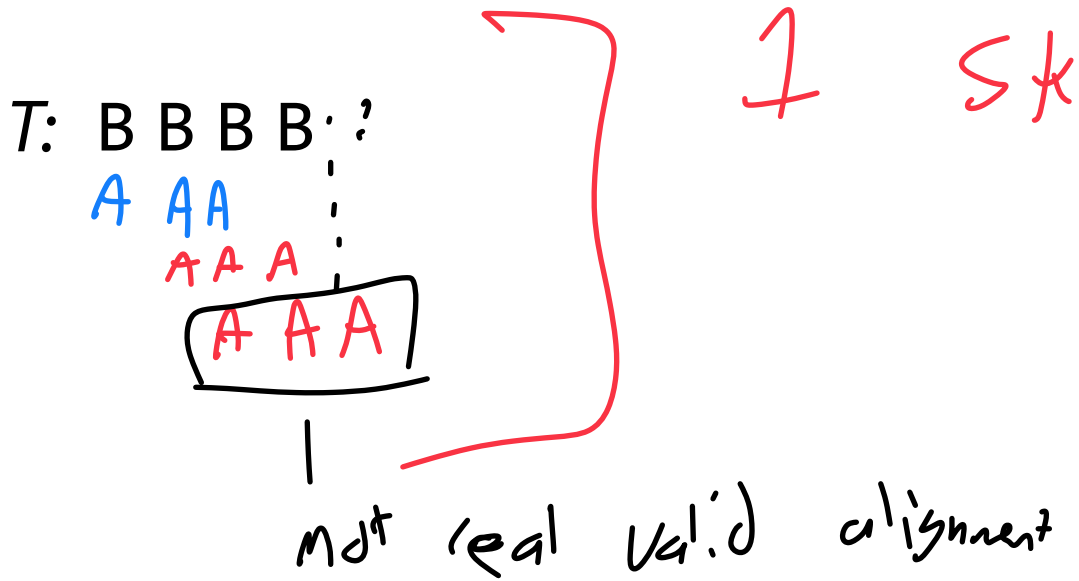
T: B B B B B B

2 skips!

Boyer-Moore: Tracking total skips

P

Σ		A	A	A
	A	-1	-1	-1
	B	0	1	2



Count while aware of edge cases

↳ only count valid alignments which are skipped

Assignment 4: a_bmoore

Learning Objective:

Implement preprocessing of patterns with Boyer-Moore*

Observe Boyer-Moore* efficiency *as a heuristic*

Consider: Our Boyer-Moore is theoretically slower than Z-algorithm.

But is it slower in practice? What is our total character comparisons?

A complete bonus lecture!

A better Boyer-Moore

Learn from character comparisons to skip pointless alignments

1. When we hit a mismatch *c*, move *P* along until *c* becomes a match (or *P* moves past *c*)

"Bad character rule"

2. Try alignments in one direction, but do character comparisons in *opposite* direction

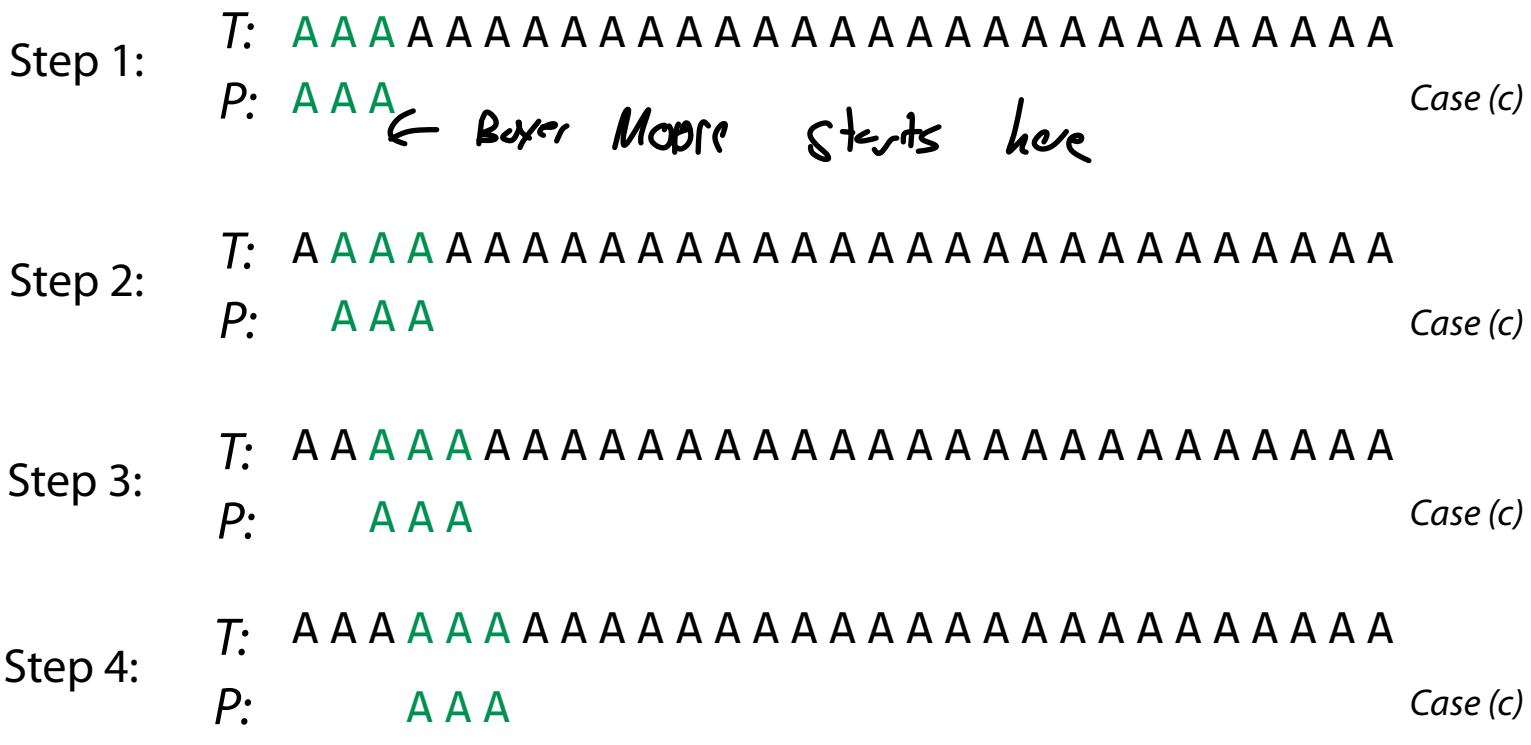
"Right-to-left scanning"

Is this $O(|P| + |T|)$? No!

Boyer, RS and Moore, JS. "A fast string searching algorithm." *Communications of the ACM* 20.10 (1977): 762-772.

Worst-Case Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) *P* moves past mismatched character. (c) If there was no mismatch, don't skip



Using just bad character, $O(|P||T|)$

A better Boyer-Moore

The complete Boyer-Moore algorithm, ***with all refinements***, is $O(|P| + |T|)$.

Refinements include:

- "strong" good suffix rule
- Galil rule

We will be covering the 'weak' good suffix rule

If interested in refinements, see Gusfield textbook (syllabus) or contact me for details

"Weak" Good Suffix rule

Intuition: Learn from alignments to avoid others

What does this alignment tell us?

Look for "AC" elsewhere in my pattern

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others

We only want to look at alignments that are ***at least as good*** as our current alignment

“Weak” Good Suffix rule

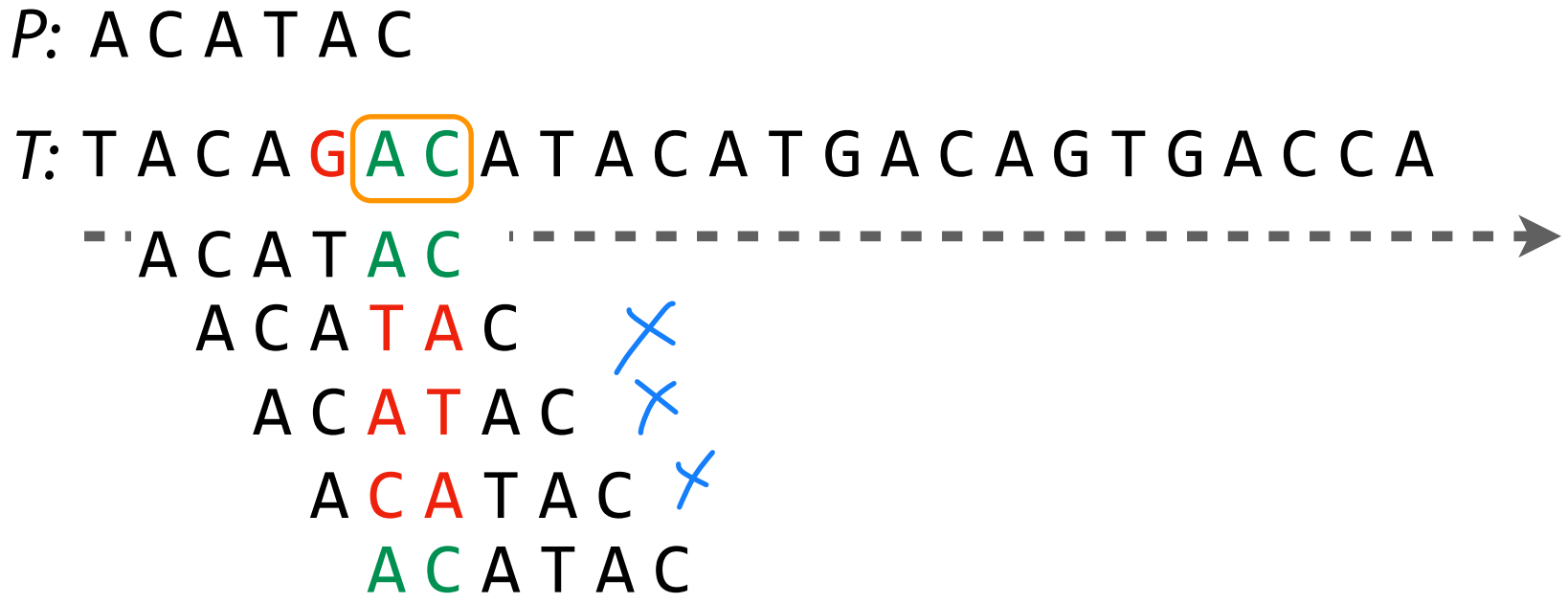
Intuition: Learn from alignments to avoid others

What does partial match (the suffix ‘AC’) tell us?

Any alignment that overlaps this region of the text must match the suffix! So we can look for another ‘AC’ somewhere in the pattern!

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others



Any alignment that overlaps this region of the text must match the suffix! So we can look for another ‘AC’ somewhere in the pattern!

“Weak” Good Suffix rule

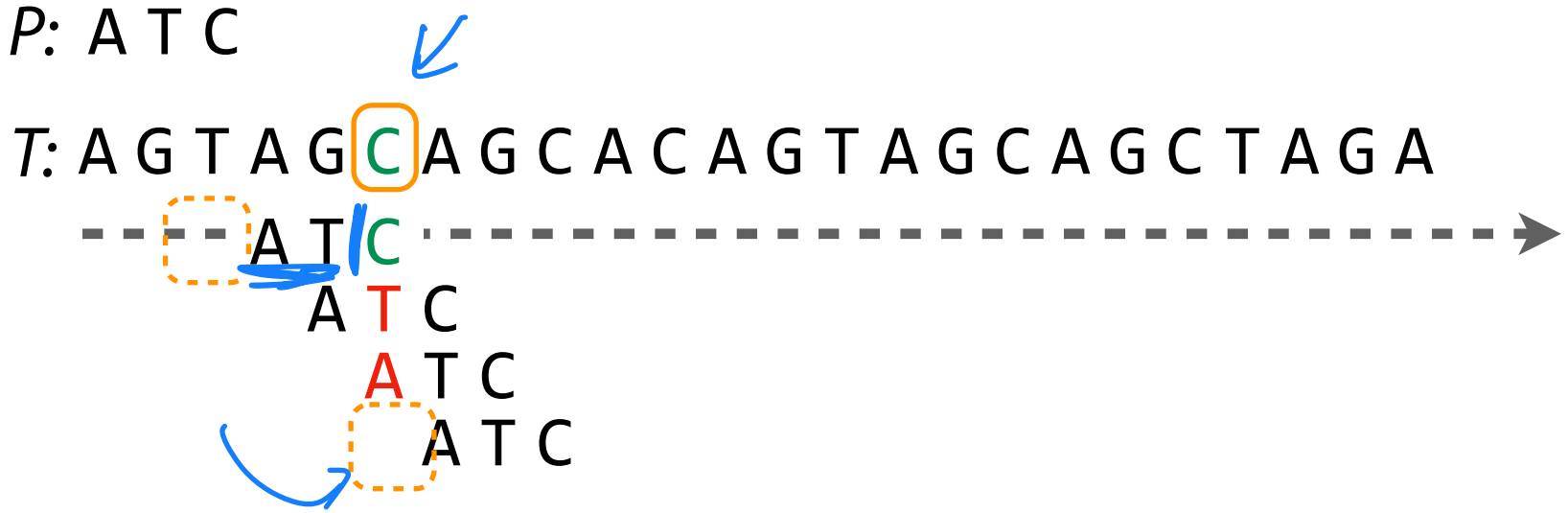
Intuition: Learn from alignments to avoid others

How many alignments do we skip? 3

Any alignment that overlaps this region of the text must match the suffix! So we can look for another ‘AC’ somewhere in the pattern!

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others



Any alignment that overlaps this region of the text must match the suffix! So we can look for another _____ somewhere in the pattern!

How many alignments do we skip?

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others

Any alignment that overlaps this region of the text must match the suffix! So we can look for another **C** somewhere in the pattern!

How many alignments do we skip? 2

“Weak” Good Suffix rule

Join Code: 225

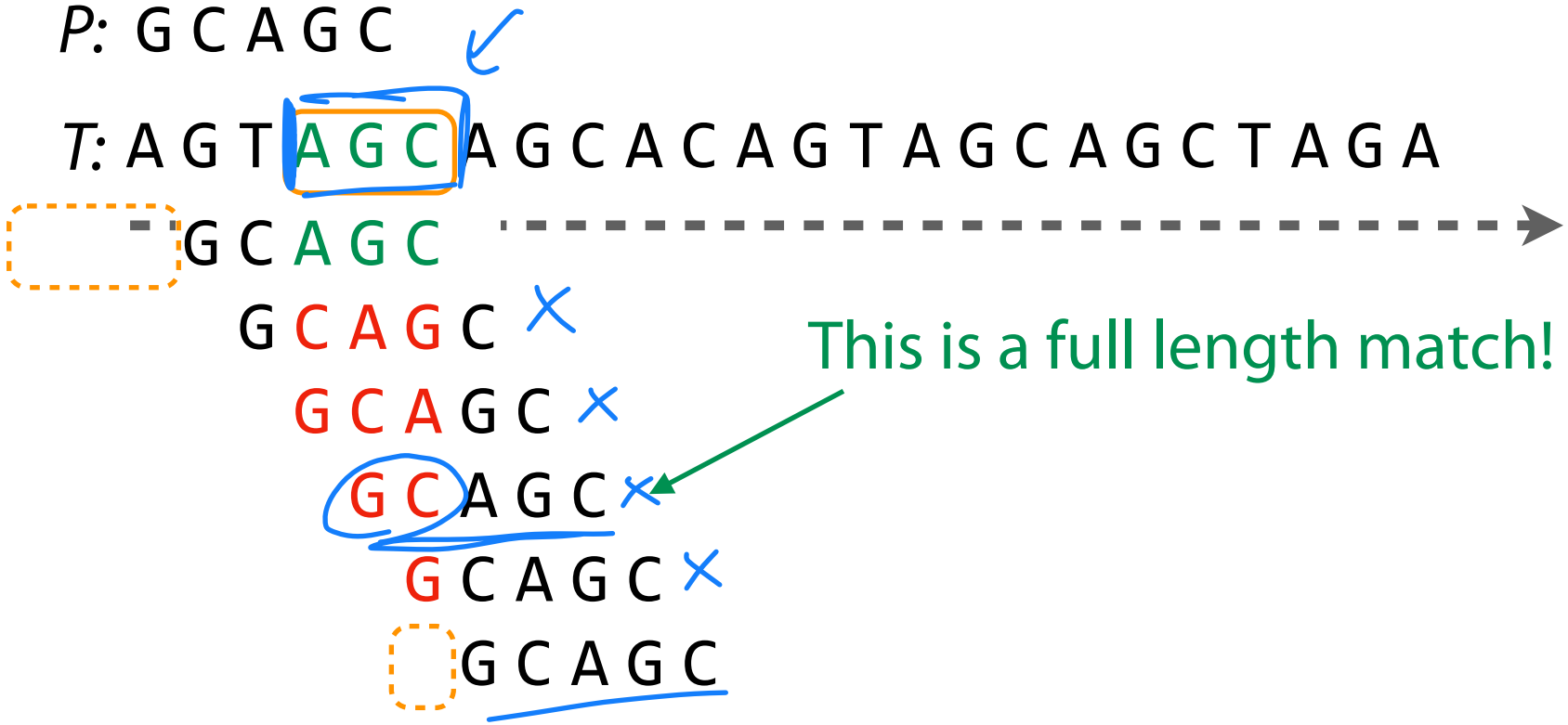
Intuition: Learn from alignments to avoid others

Any alignment that overlaps this region of the text must match the suffix! So we can look for another A G C somewhere in the pattern!

How many alignments do we skip?

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others



How many alignments do we skip?

“Weak” Good Suffix rule

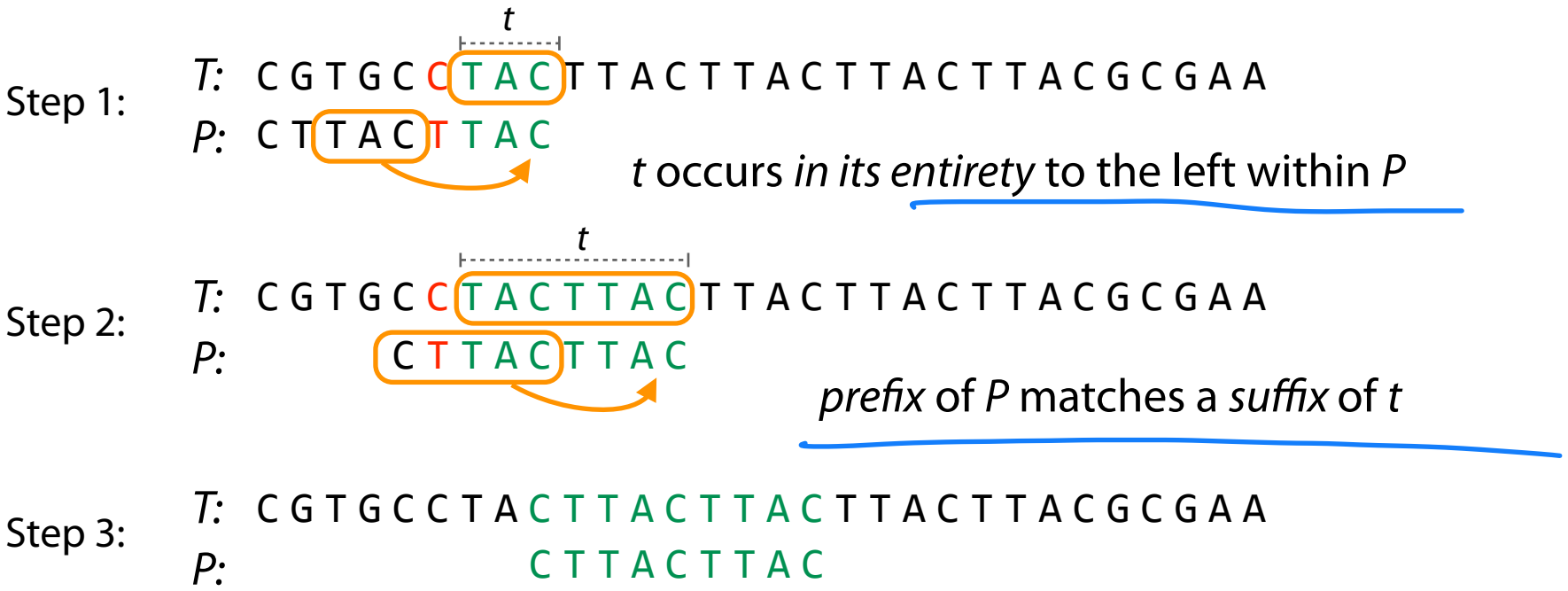
Intuition: Learn from alignments to avoid others

Any alignment that overlaps this region of the text must match the suffix … *or have a prefix-suffix partial match!*

How many alignments do we skip? 2

"Weak" Good Suffix rule

Let *t* = longest suffix match at alignment; skip until (a) we find another ***instance*** of *t or* (b) *P* moves past *t*



An ***instance*** of *t* is either a full match to the left within *P*
or a *prefix* of *P* matches a *suffix* of *t*

Boyer-Moore: Putting it together

How to combine bad character and good suffix rules?

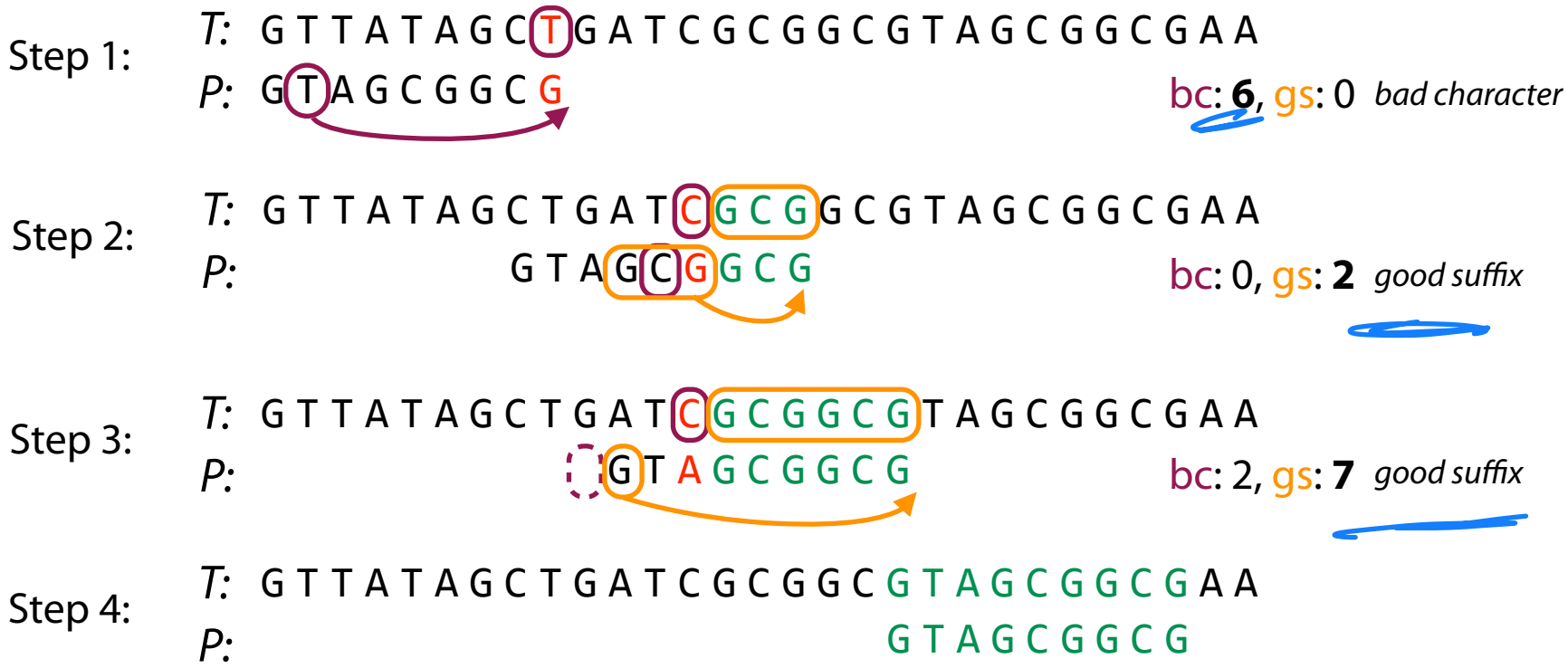
How many characters does bad character skip? 2 characters

How many characters does good suffix skip? 7 characters

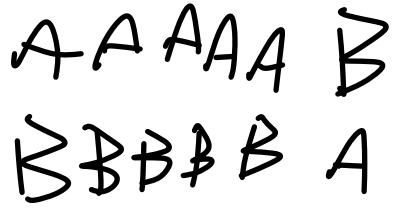
Take the maximum (7)!

Boyer-Moore: Putting it together

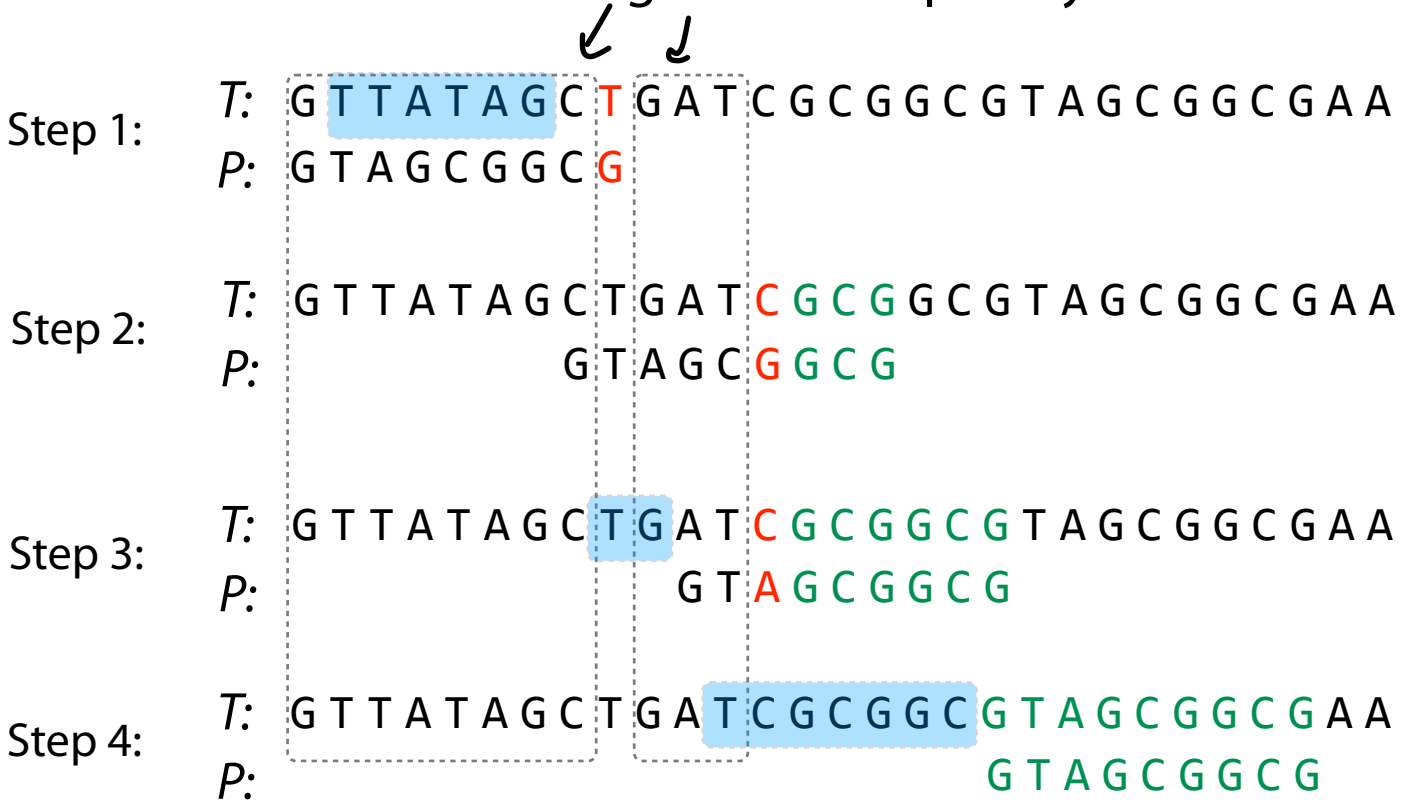
Use bad character or good suffix rule, *whichever skips more*



Boyer-Moore: Putting it together



11 characters of *T* ignored completely!



Skipped 15 alignments

Boyer-Moore

Learn from character comparisons to skip pointless alignments

1. When we hit a mismatch *c*, move *P* along until *c* becomes a match (or *P* moves past *c*) — "Bad character rule"

2. Try alignments in one direction, but do character comparisons in *opposite* direction — "Right-to-left scanning"

3. When we move *P* along, make sure characters that matched in the last alignment also match in the next alignment — "Good suffix rule"

Boyer, RS and Moore, JS. "A fast string searching algorithm." *Communications of the ACM* 20.10 (1977): 762-772.