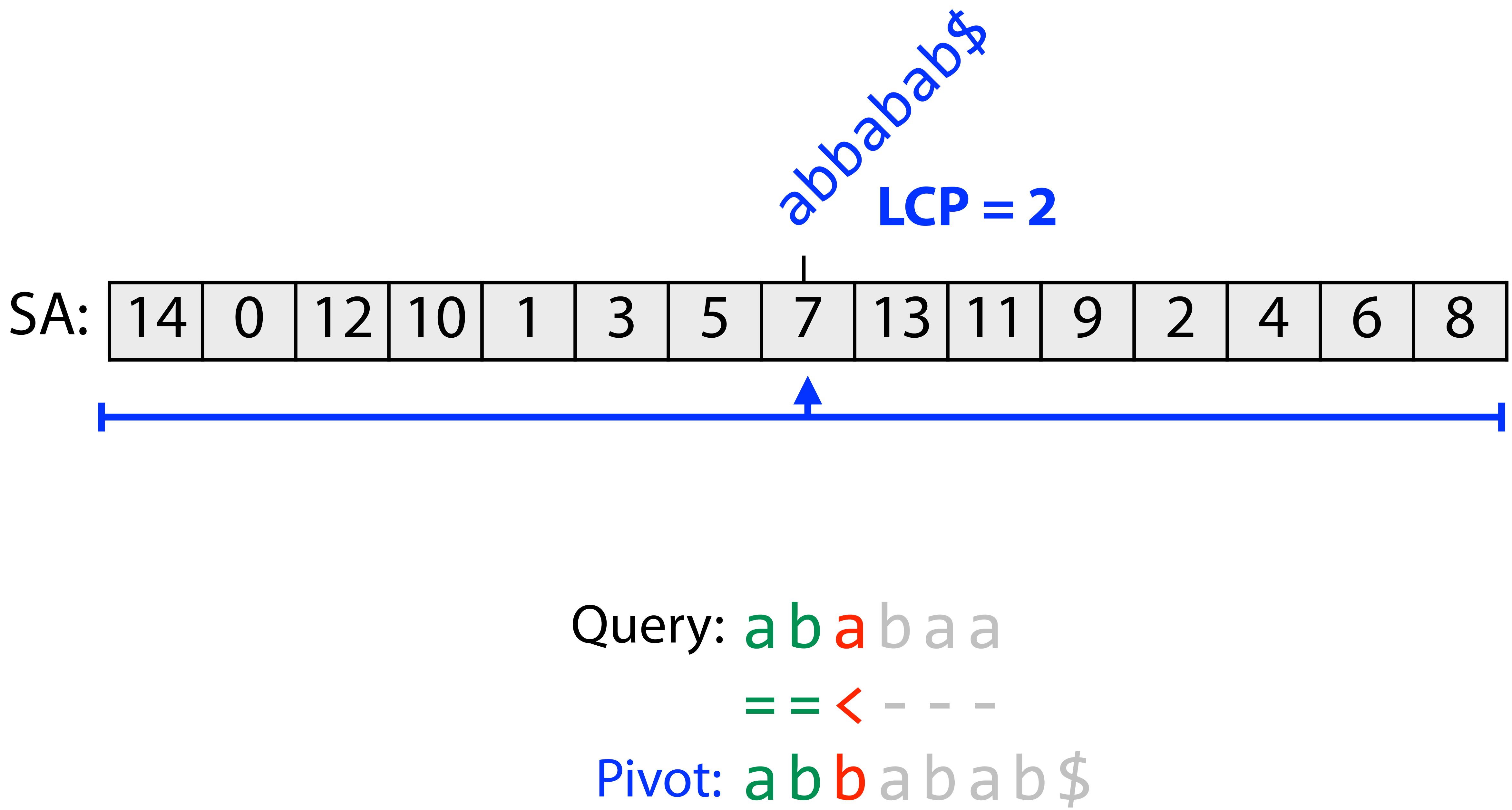
Suffix Arrays: maximum skipping

Ben Langmead

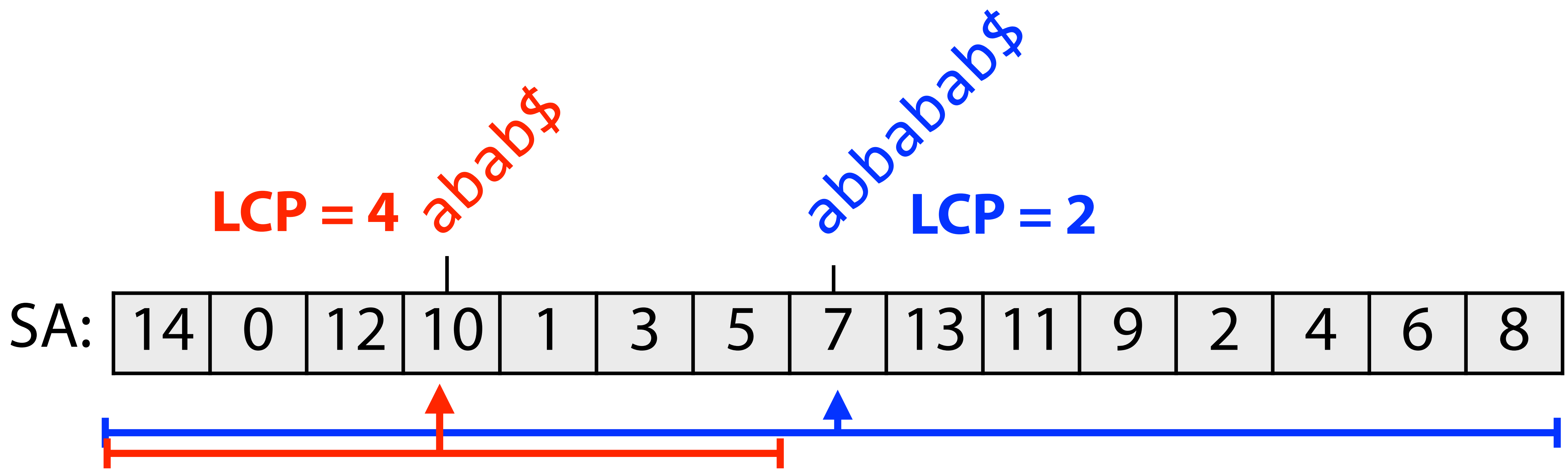
Johns Hopkins Whiting School of Engineering

Department of Computer Science

Suffix array: querying



Suffix array: querying

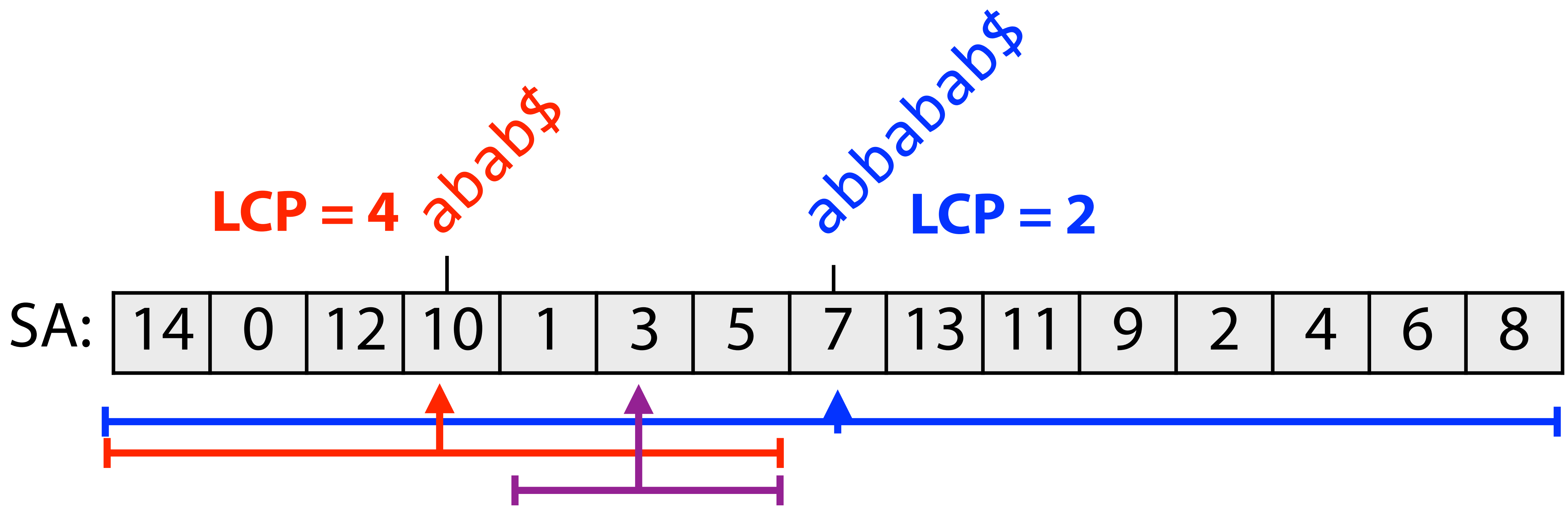


Query: a b a b a a

= = = = >

Pivot: a b a b $

Suffix array: querying



Query: a b a b a a

= = = = = <

Pivot: a b a b a b b a b a b $

Skip! Skip!

Min-LCP skipping uses what we **learn** about LCPs to skip character comparisons

We can also **precompute** common prefixes between suffixes for even more skipping!

Suffix array

Terminology: length of common prefix between a query string and a suffix is an **LCP**

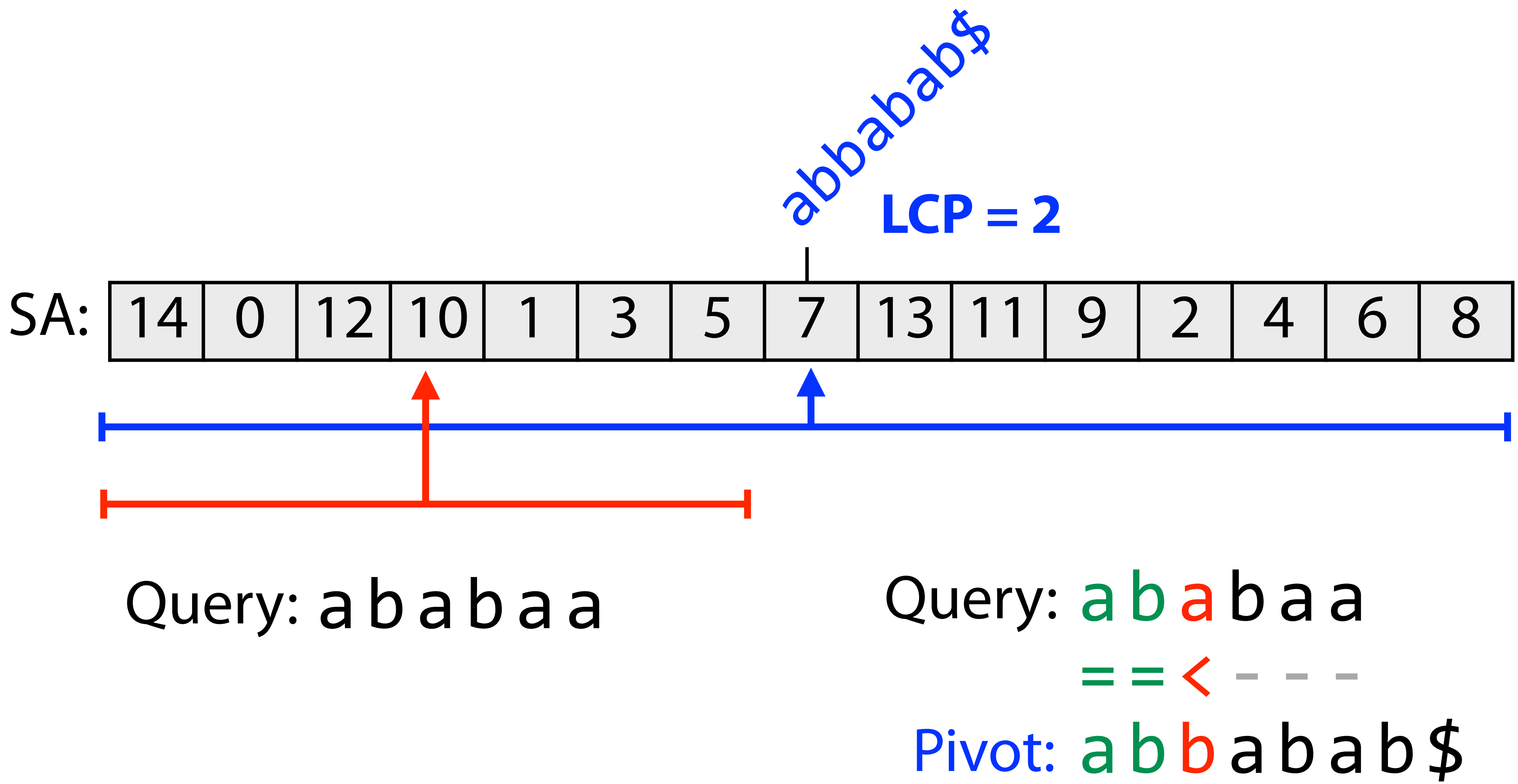
Longest Common Prefix

Length of common prefix between two suffixes of the same string is an **LCE**

Longest Common Extension

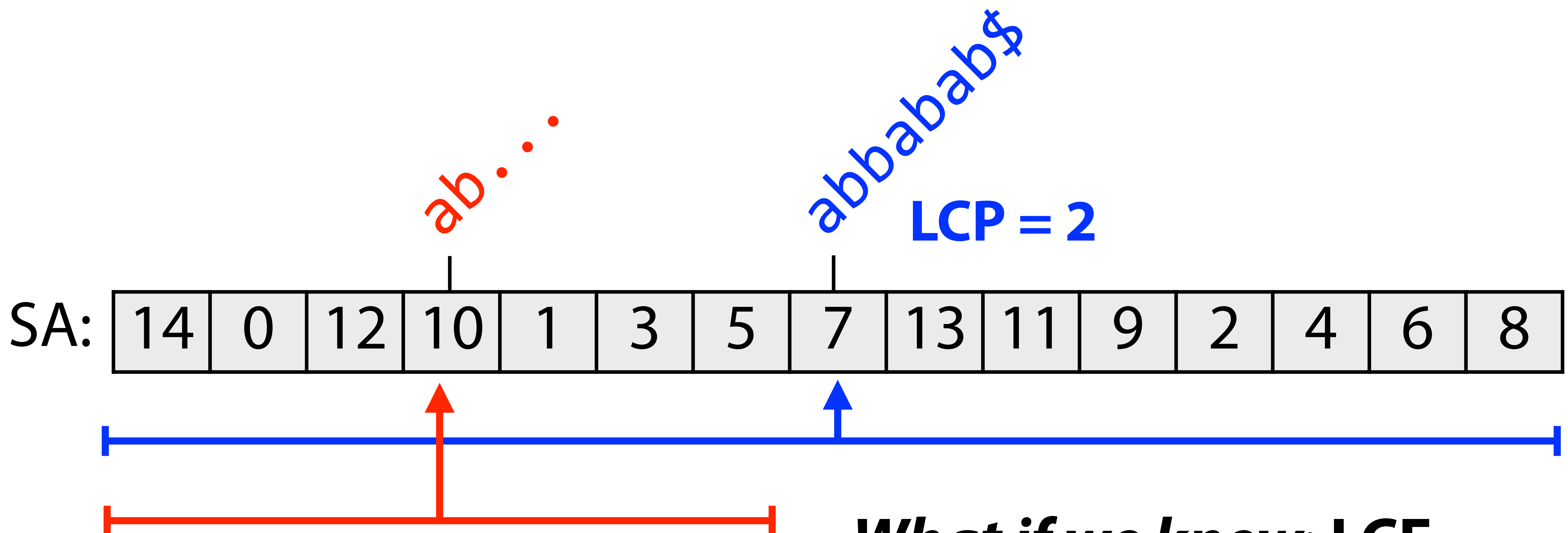
We learn about **LCPs** during binary search;
we can precompute **LCEs**

Suffix array



Suffix array

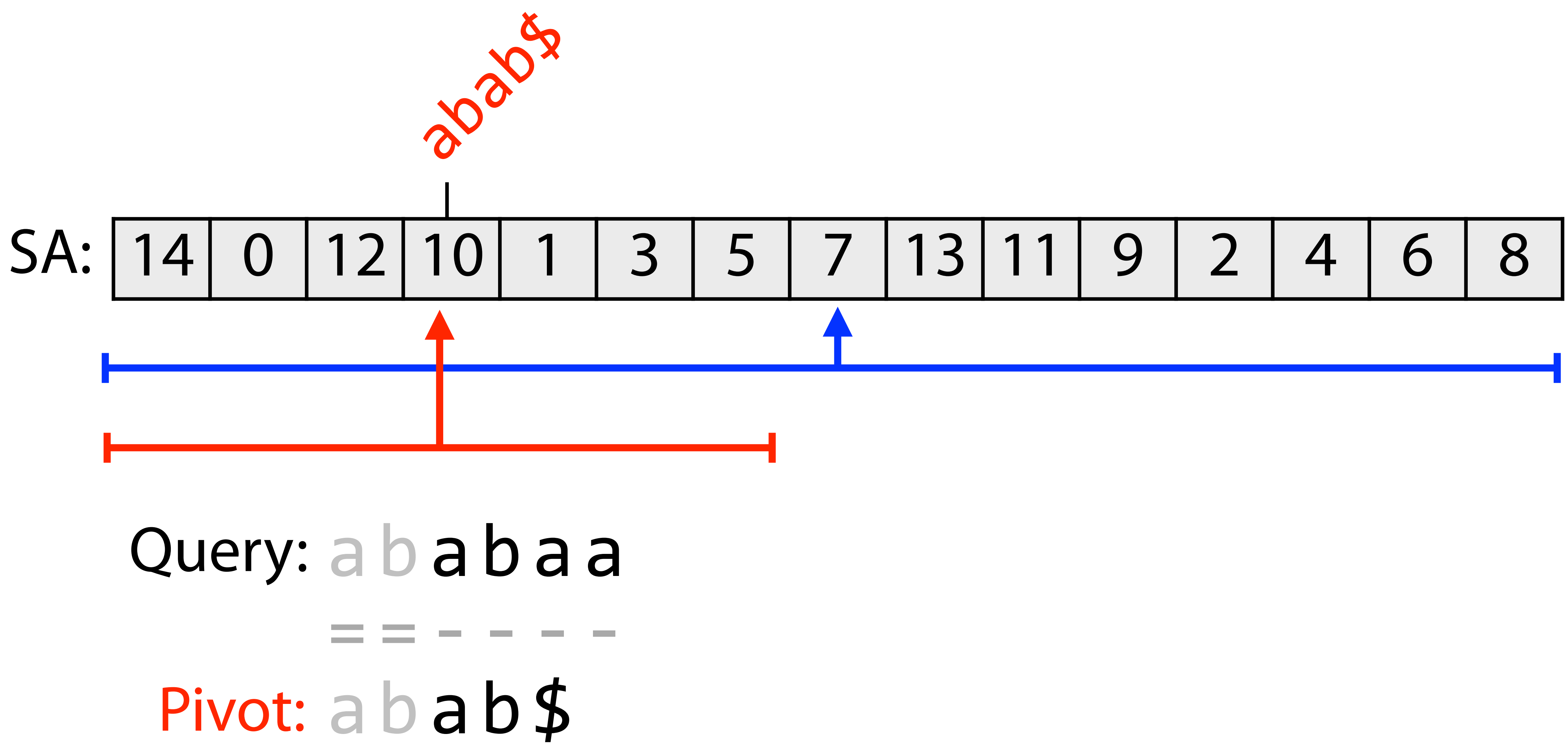
We can **skip** the first 2 character comparisons between query and new pivot!



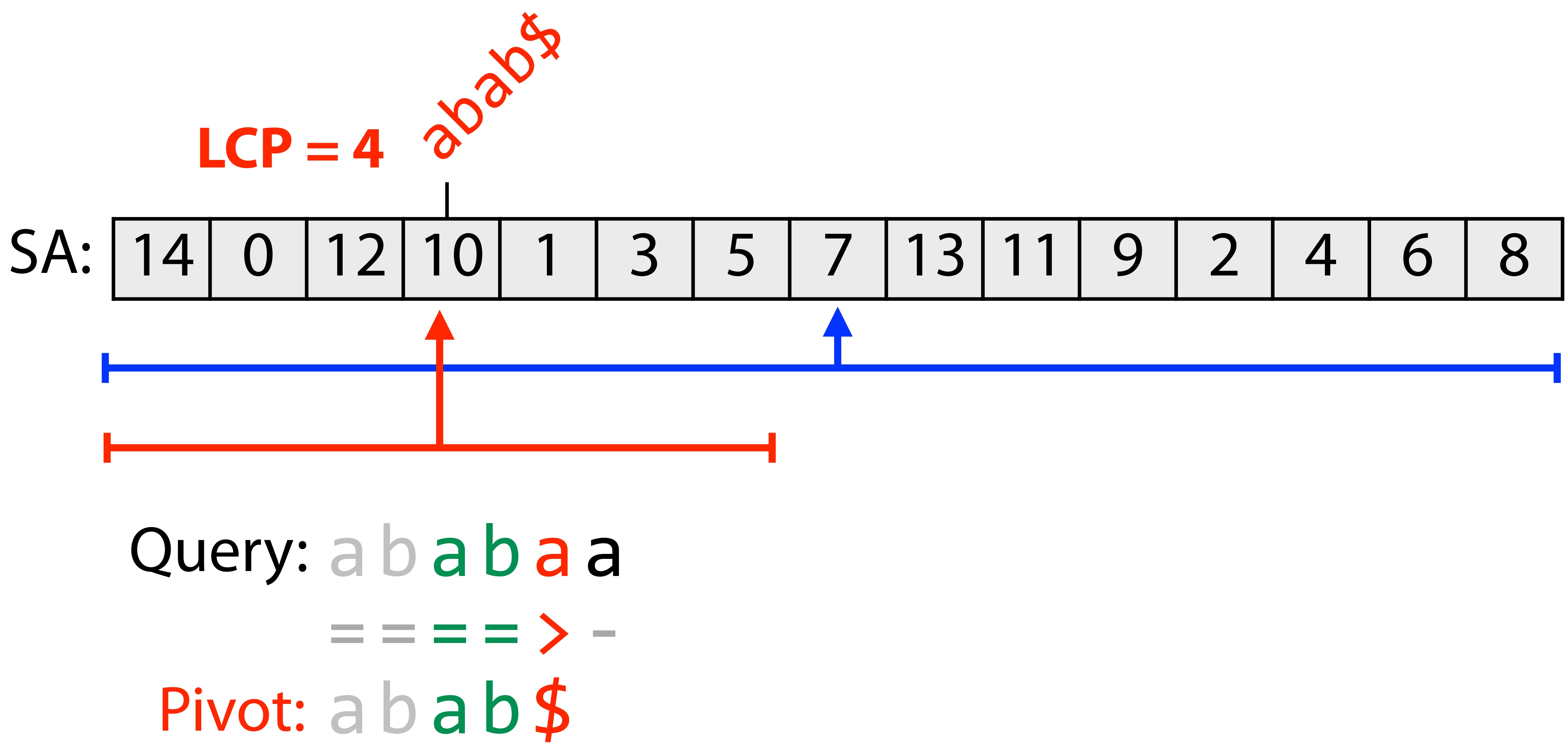
Query: a b a b a a

What if we knew: **LCE** between the two pivots is 2

Suffix array

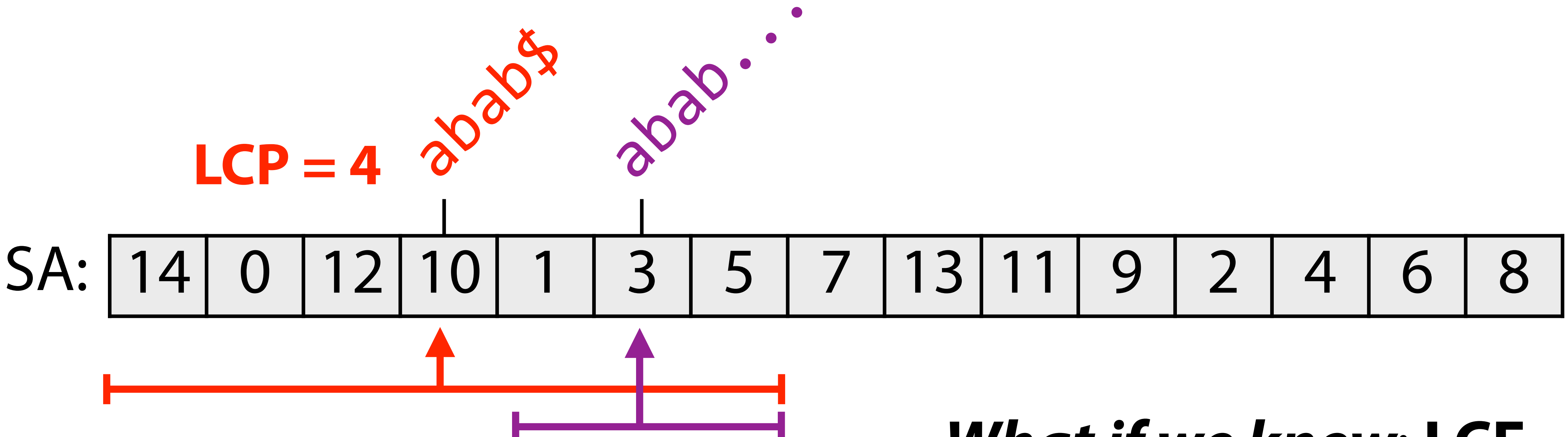


Suffix array



Suffix array

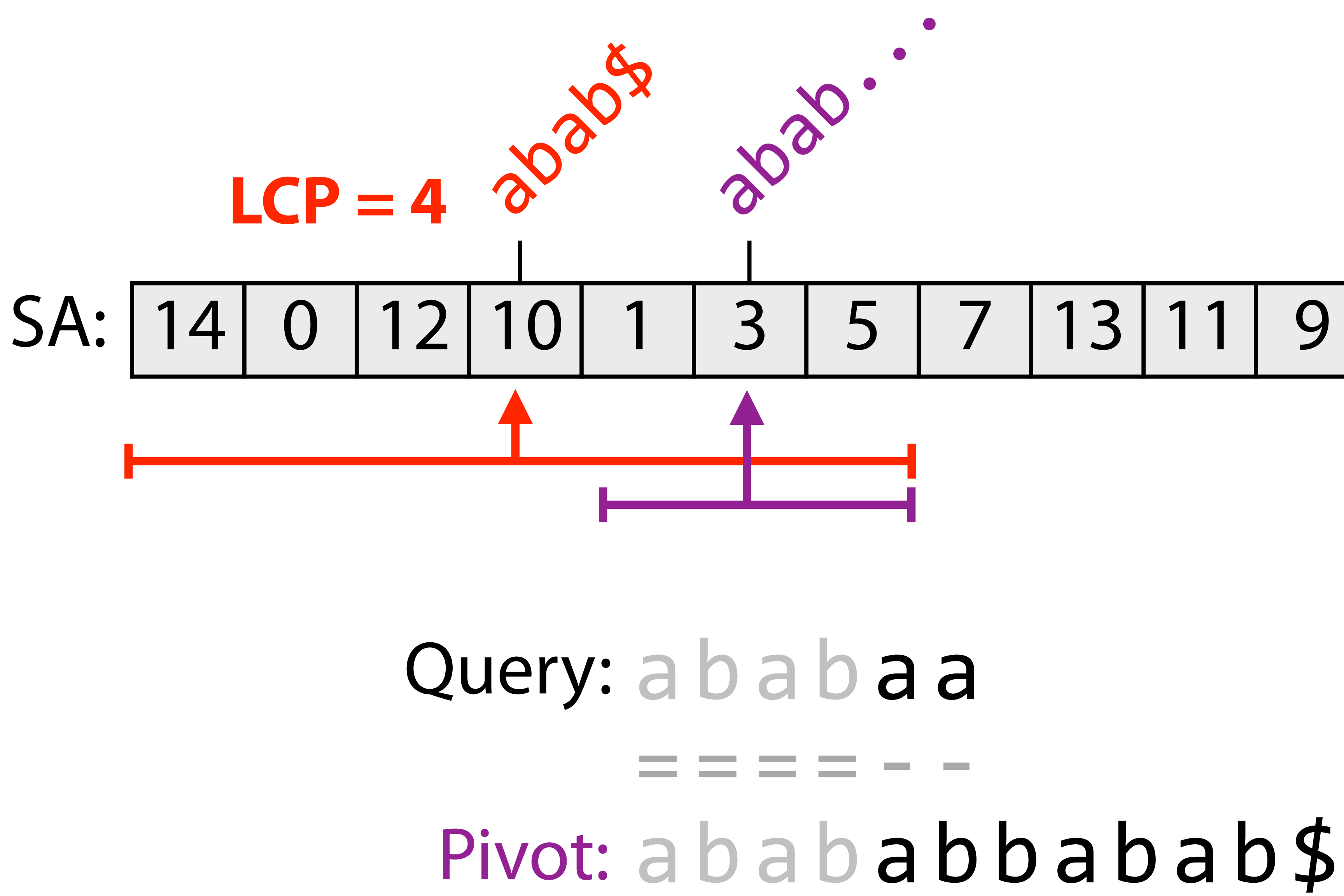
Skip first 4 character comparisons between query and new pivot!



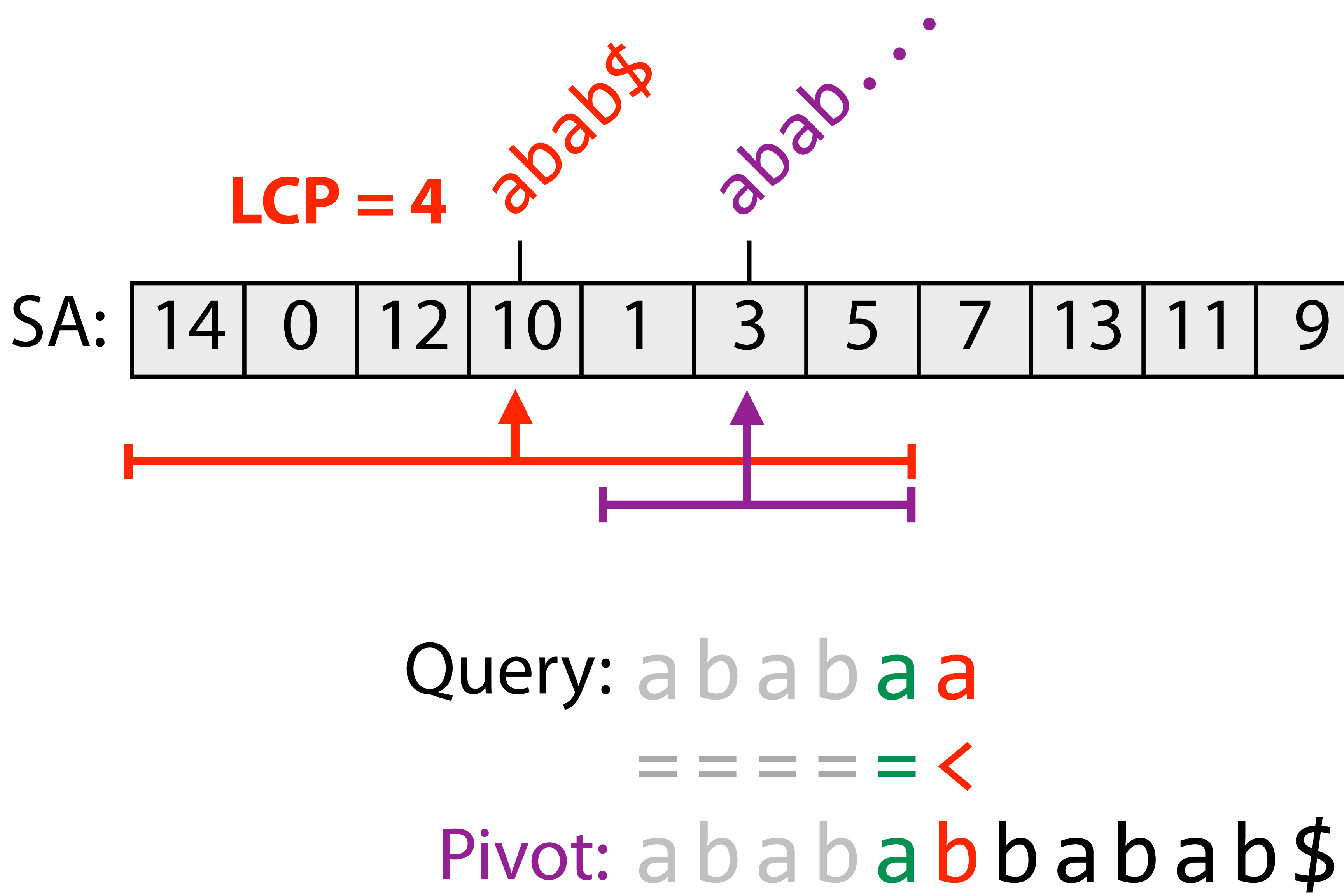
What if we knew: **LCE** between the two pivots is 4

Query: a b a b a a

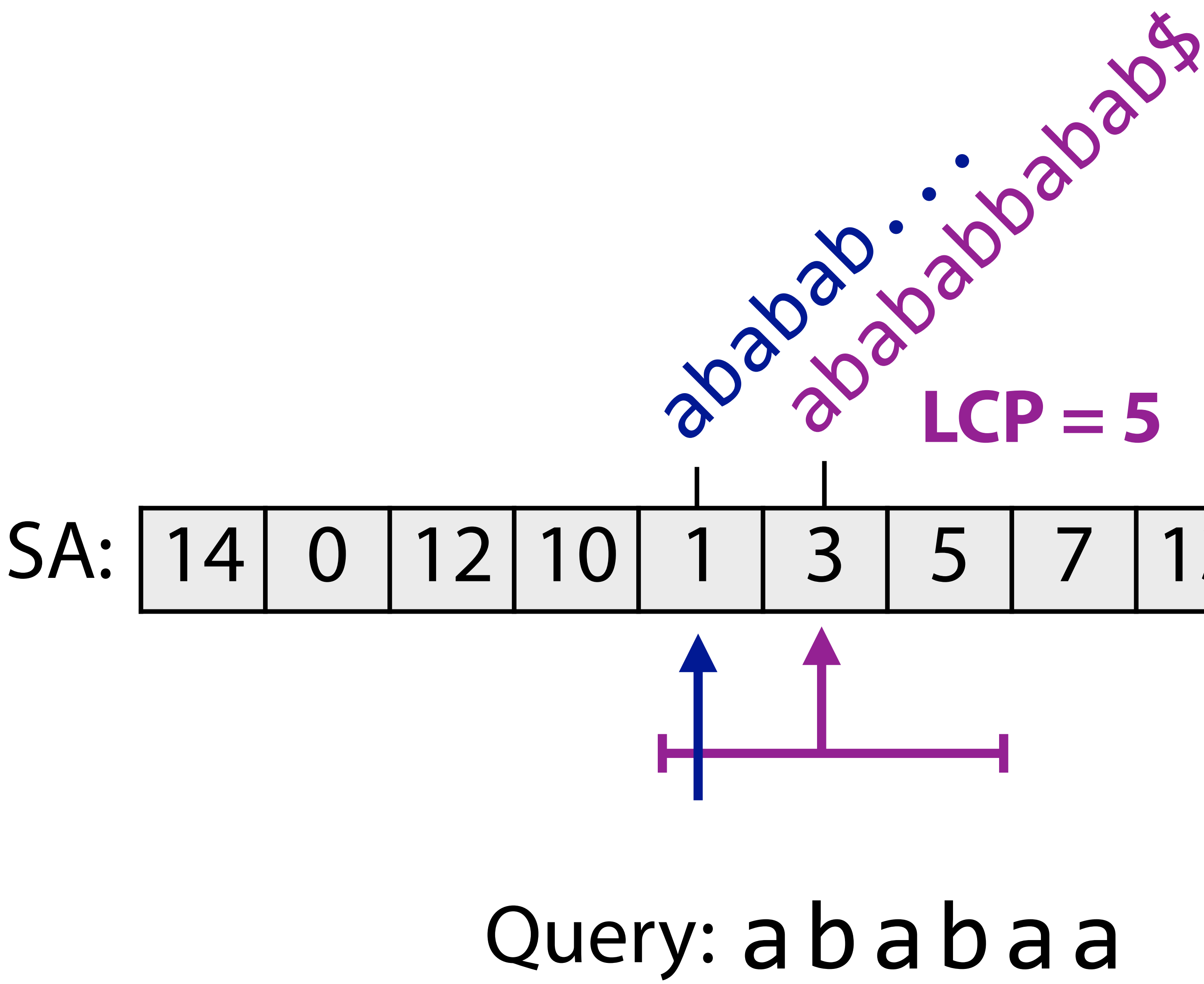
Suffix array



Suffix array



Suffix array



What if we knew: **LCE** between the two pivots is 6

If query was less than the previous pivot, it must be less than this pivot too; no character comparisons needed

Suffix array

No skipping

```
a b a b a a
= = < - - -
a b b a b a b $
```

```
a b a b a a
= = = = > -
a b a b $
```

```
a b a b a a
= = = = = <
a b a b a b b a b a b $
```

```
a b a b a a
= = = = = <
a b a b a b a b b a b a b $
```

Min-LCP skipping

```
a b a b a a
= = < - - -
a b b a b a b $
```

```
a b a b a a
= = = = > -
a b a b $
```

```
a b a b a a
= = = = = <
a b a b a b b a b a b $
```

```
a b a b a a
= = = = = <
a b a b a b a b b a b a b $
```

Suffix array

Min-LCP skipping

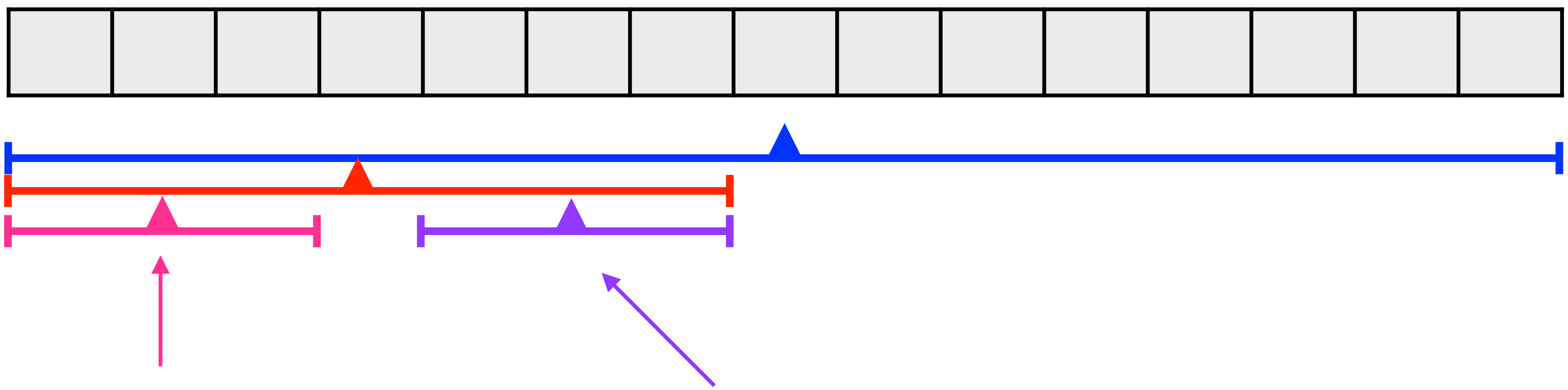
```
a b a b a a
= = < - - -
a b b a b a b $

a b a b a a
= = = = > -
a b a b $

a b a b a a
= = = = = <
a b a b a b b a b a b $

a b a b a a
= = = = = <
a b a b a b a b b a b a b $
```

Max skipping

```
a b a b a a
= = < - - -
a b b a b a b $

a b a b a a
= = = = > -
a b a b $

a b a b a a
= = = = = <
a b a b a b b a b a b $

a b a b a a
= = = = = <
a b a b a b a b b a b a b $
```

Suffix array: less-than case

Say we know the query is less than the previous pivot with some LCP. We also know the LCE *between* the pivots.



LCP < LCE

Query must also be **less** than next pivot; recurse left

New LCP is same as old LCP

New LCE is between red & new pivots

LCP > LCE

Query must be **greater** than next pivot; recurse right

New LCP is same as old LCP

New LCE is between blue & new pivots

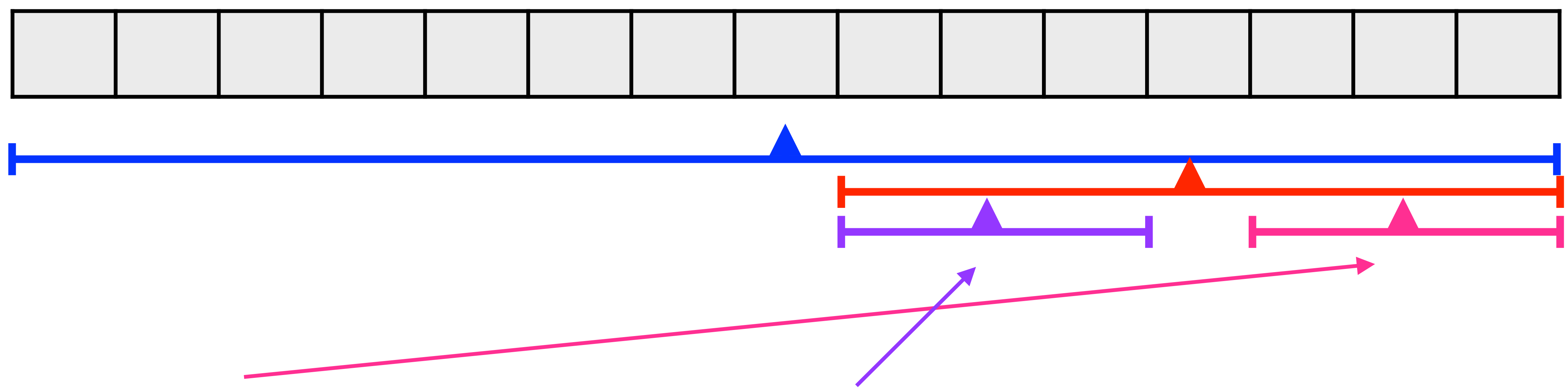
LCP = LCE

Skip first LCP characters, then continue comparisons, updating LCP and deciding recursion as usual.

New LCE will involve red pivot

Suffix array: greater-than case

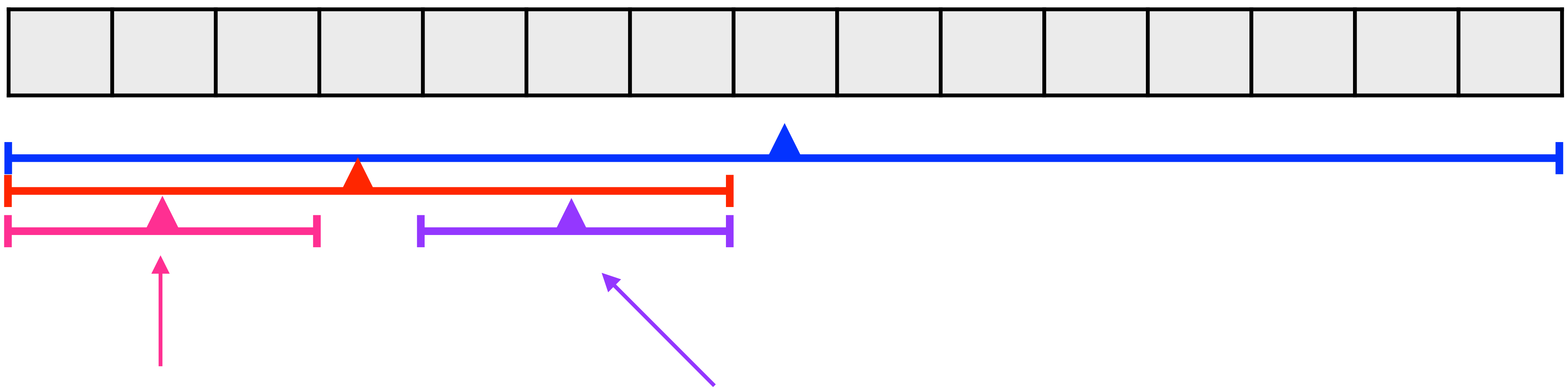
Say we know the query is **greater** than the previous pivot with some LCP. We also know the LCE *between* the pivots.



LCP < LCE	LCP > LCE	LCP = LCE
Query must also be **greater** than next pivot; recurse right	Query must be **less** than next pivot; recurse left	Skip first LCP characters, then continue comparisons, updating LCP and deciding recursion as usual.
New LCP is same as old LCP	New LCP is same as old LCP	New LCE will involve red pivot
New LCE is between red & new pivots	New LCE is between blue & new pivots	

Suffix array: less-than case

Say we know the query is less than the previous pivot with some LCP. We also know the LCE *between* the pivots.



LCP < LCE

Query must also be **less** than next pivot; recurse left

New LCP is same as old LCP

New LCE is between red & new pivots

LCP > LCE

Query must be **greater** than next pivot; recurse right

New LCP is same as old LCP

New LCE is between blue & new pivots

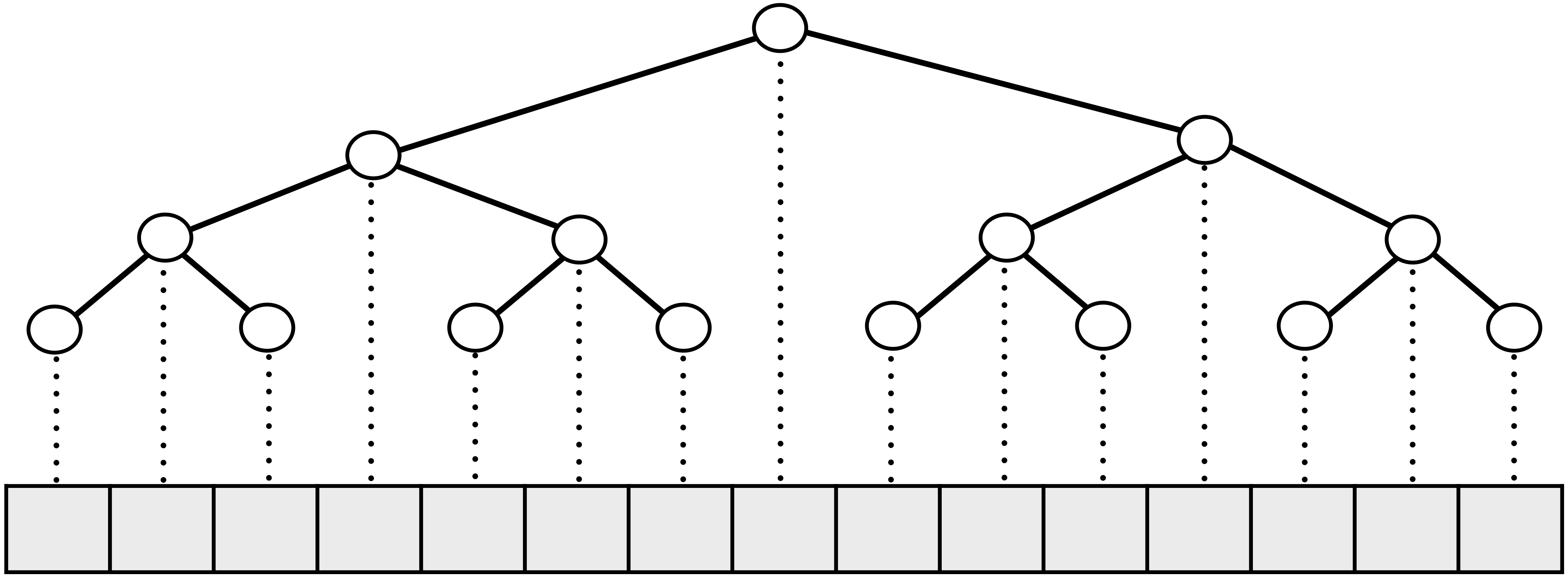
LCP = LCE

Skip first LCP characters, then continue comparisons, updating LCP and deciding recursion as usual.

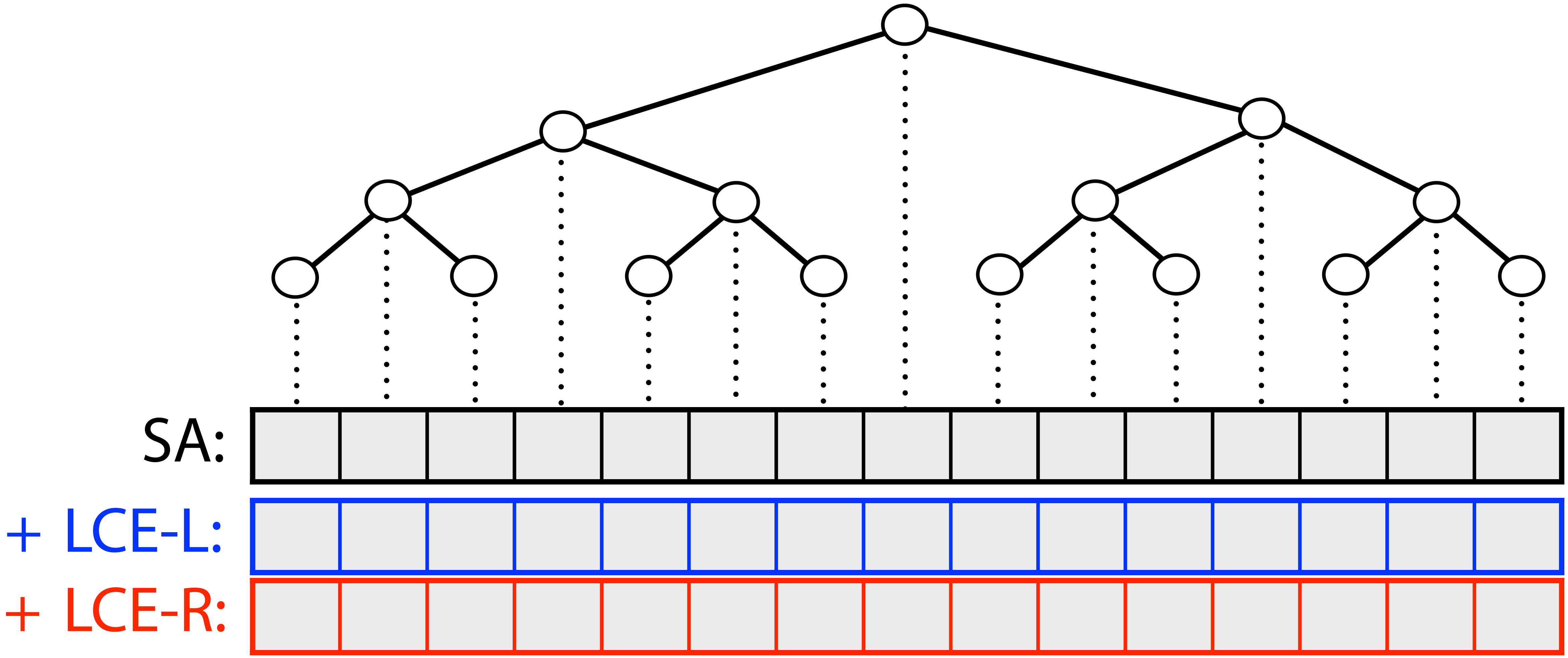
New LCE will involve red pivot

Suffix array

We must precompute all possible LCEs that might be needed in the previous computation

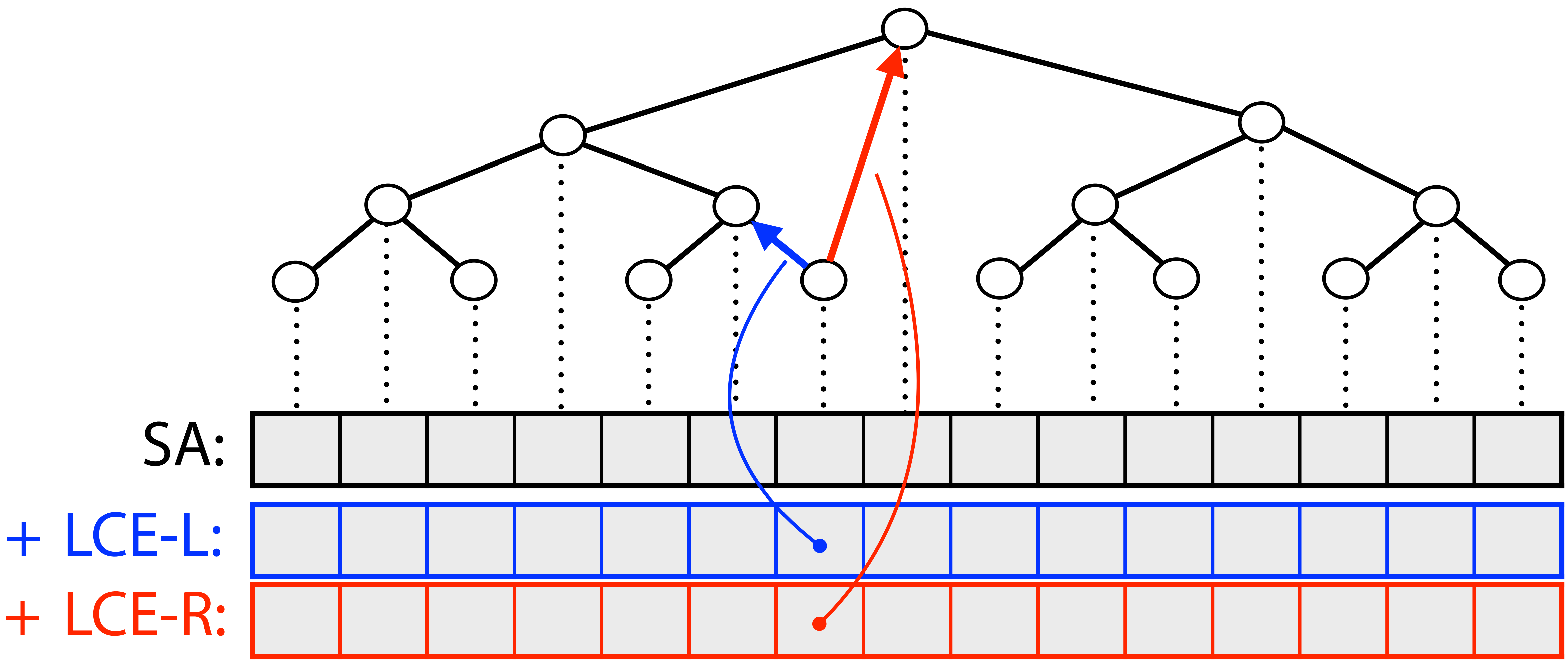


Suffix array



Store pre-computed LCEs at nodes (pivots), conditioned on whether previous pivot was to the left or right

Suffix array



Store pre-computed LCEs at nodes (pivots), conditioned on whether previous pivot was to the left or right

Approximately triples size of the data structure (!)

Suffix array

Like naive & min-LCP strategies, max skipping performs $O(\log m)$ bisections

Once a character of P matches it "stays matched." Comparisons in a given round don't look back farther than the last mismatch.

Our query time bound therefore improves from $O(n \log m)$ to $O(n + \log m)$

Max skipping

```
a b a b a a
= = < - - -
a b b a b a b $

a b a b a a
= = = = > -
a b a b $

a b a b a a
= = = = = <
a b a b a b b a b a b $

a b a b a a
= = = = = <
a b a b a b a b b a b a b $
```

Suffix array: summary

Naive binary search on suffix array takes $O(n \log m)$ time, in contrast to $O(n)$ for suffix tree query

Min-LCP skipping helps, using what we learned in previous rounds to skip character comparisons

Requires no additional space beyond SA + T

Not $O(n + \log m)$, but efficient in practice

Max skipping additionally stores ~2m pre-computed LCEs, tripling structure size, but improving time to $O(n + \log m)$

Suffix array: summary

Whether query takes $O(n \log m)$ or $O(n + \log m)$ time, there's no escaping the $\log m$; the price of binary search!