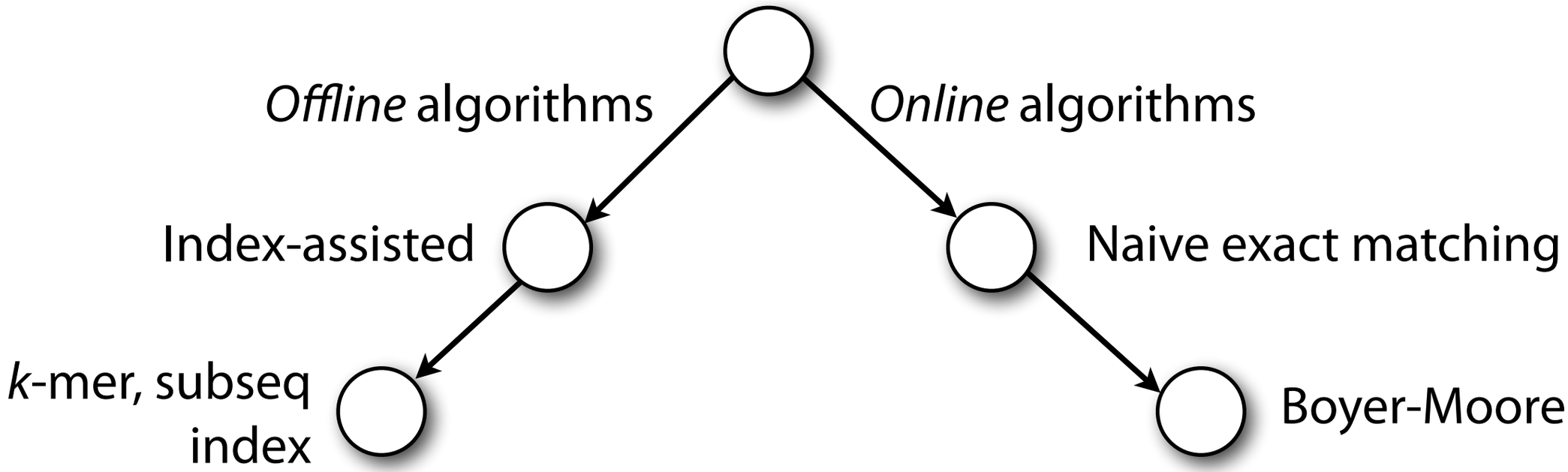
Approximate matching



All for exact matching

Approximate matching

Read

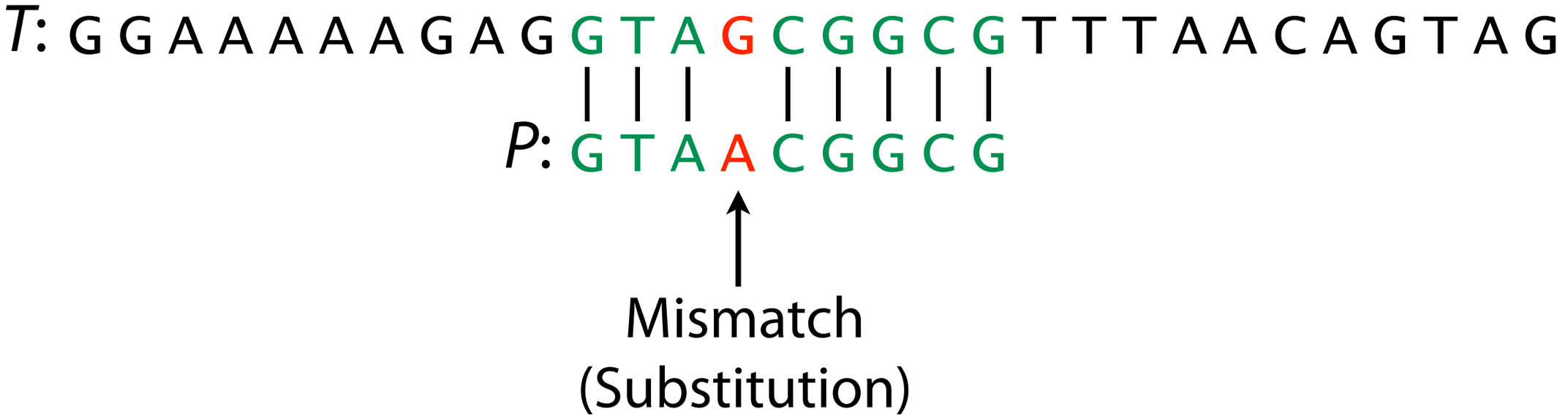
CTCAAACTCCTGACCTTTGGTGATCCACCCGCCTAGGCCTTC

Reference

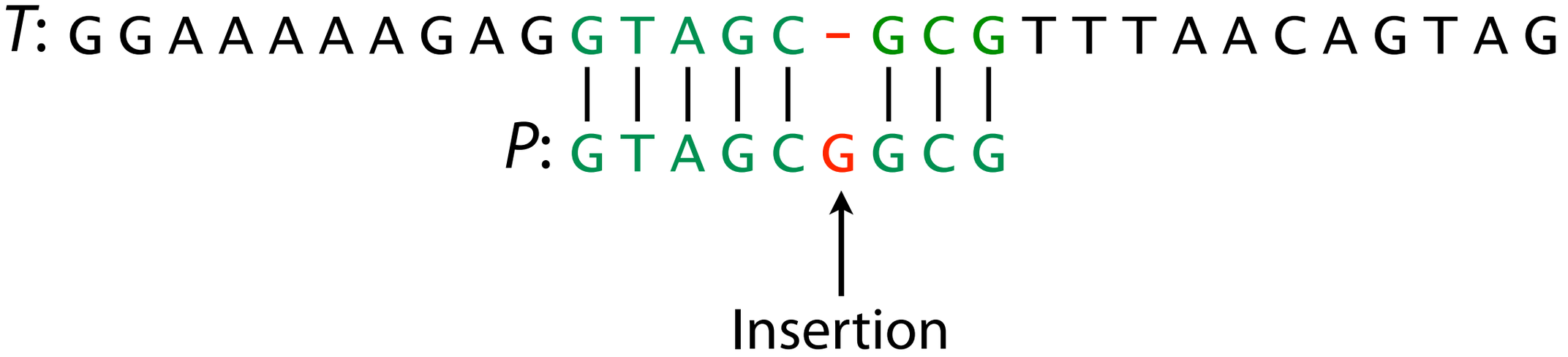
Differences between read and reference occur because of...

1. Sequencing error
2. Natural variation

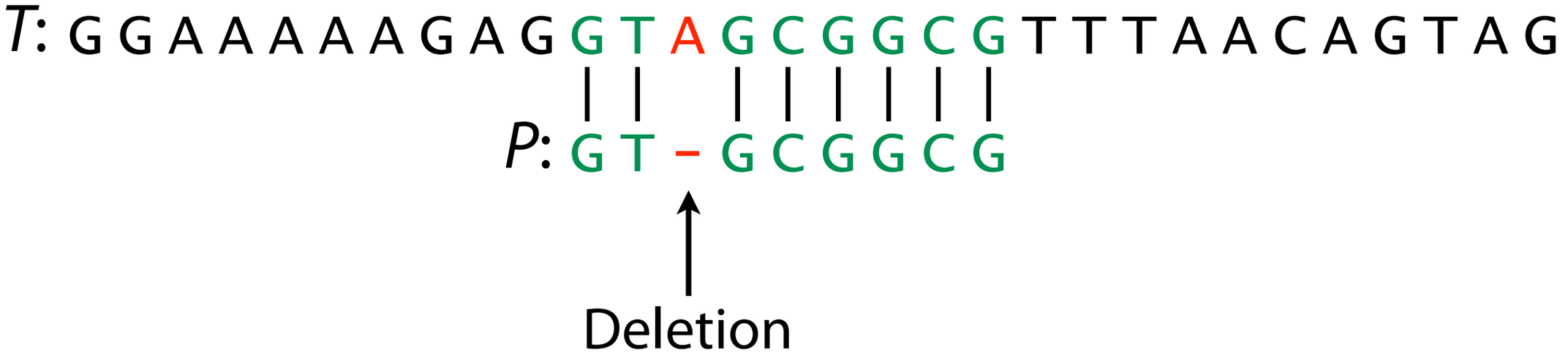
Approximate matching



Approximate matching



Approximate matching



Hamming distance

For X & Y where $|X| = |Y|$, *hamming distance* =
minimum # substitutions needed to turn one into the other

Hamming distance = 3

Edit distance

(AKA Levenshtein distance)

For *X* & *Y*, *edit distance* = minimum # edits (substitutions, insertions, deletions) needed to turn one into the other

```
X: T G G C C G C G C A A A A A C A G C
   | |   | | | | | | | | |   | | | |
Y: T G A C C G C G C A A A A - C A G C
```

Edit distance = 2

```
X: G C G T A T G C G G C T A - A C G C
   | |   | | | | | | | | |   | | |
Y: G C - T A T G C G G C T A T A C G C
```

Edit distance = 2

Approximate matching

```
def naive(p, t):
    occurrences = []
    for i in xrange(len(t) - len(p) + 1): # loop over alignments
        match = True
    for j in xrange(len(p)):              # loop over characters
        if t[i+j] != p[j]:                # compare characters
            match = False                 # mismatch; reject alignment
            break
    if match:
      occurrences.append(i)               # all chars matched; record
return occurrences
```

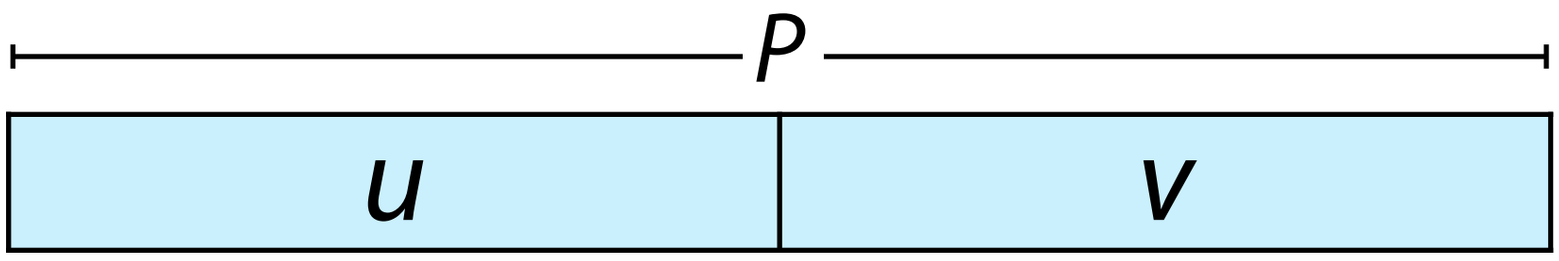
Approximate matching

```
def naiveHamming(p, t, maxDistance):
    occurrences = []
    for i in xrange(len(t) - len(p) + 1): # loop over alignments
        nmm = 0
        match = True
        for j in xrange(len(p)):          # loop over characters
            if t[i+j] != p[j]:            # compare characters
                nmm += 1                  # mismatch
                if nmm > maxDistance:
                    break                 # exceeded max hamming dist
        if nmm <= maxDistance:
          occurrences.append(i)           # approximate match
return occurrences
```

Approximate matching

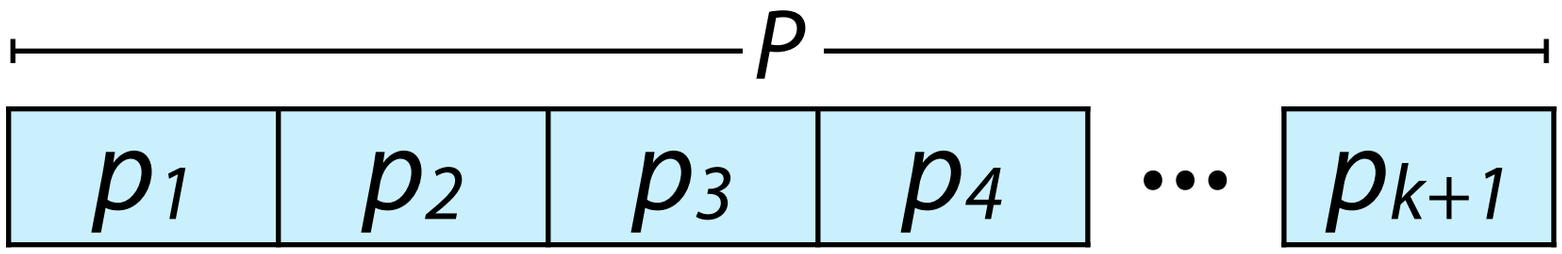
Wanted: way to apply exact matching algorithms to approximate matching problems

Approximate matching



If *P* occurs in *T* with 1 edit, then *u* or *v* appears with no edits

Approximate matching



If *P* occurs in *T* with up to *k* edits, then at least one of $p_1, p_2, ..., p_{k+1}$ must appear with 0 edits

Approximate matching

Approximate matching

Approximate Boyer-Moore performance

	Boyer-Moore, exact			Boyer-Moore, ≤1 mismatch with pigeonhole			Boyer-Moore, ≤2 mismatches with pigeonhole		
	# character comparisons	wall clock time	# matches	# character comparisons	wall clock time	# matches	# character comparisons	wall clock time	# matches
P: "tomorrow" **T**: Shakespeare's complete works	786 K	1.91s	17	3.05 M	7.73 s	24	6.98 M	16.83 s	382
P: 50 nt string from Alu repeat* **T**: Human reference (hg19) chromosome 1	32.5 M	67.21 s	336	107 M	209 s	1,045	171 M	328 s	2,798

* GCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGG