BIO306: Bioinformatics

Lecture 6

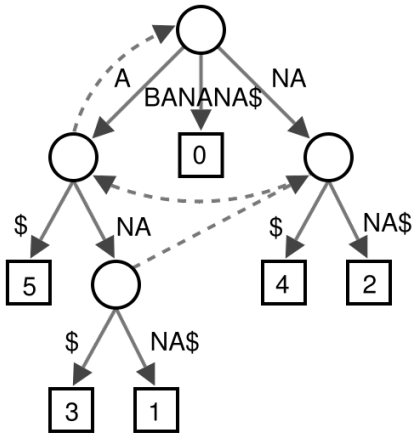
Reads Mapping and output format

Wenfei JIN PhD
jinwf@sustc.edu.cn
Department of Biology, SUSTech

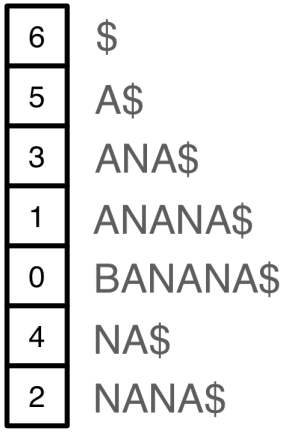
Indexing genome

- Genomes and reads are too large for direct approaches like dynamic programming

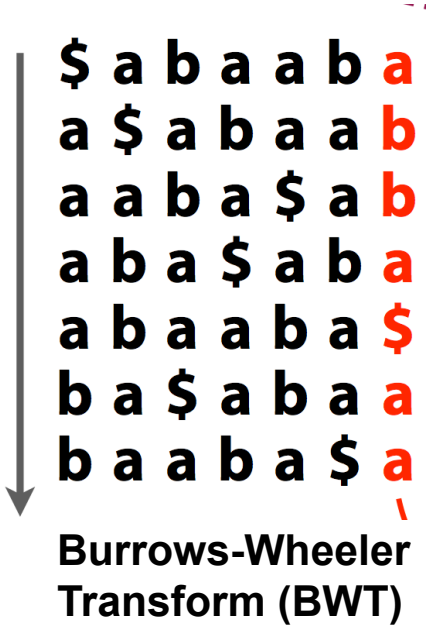
Data structures for indexing



Suffix tree



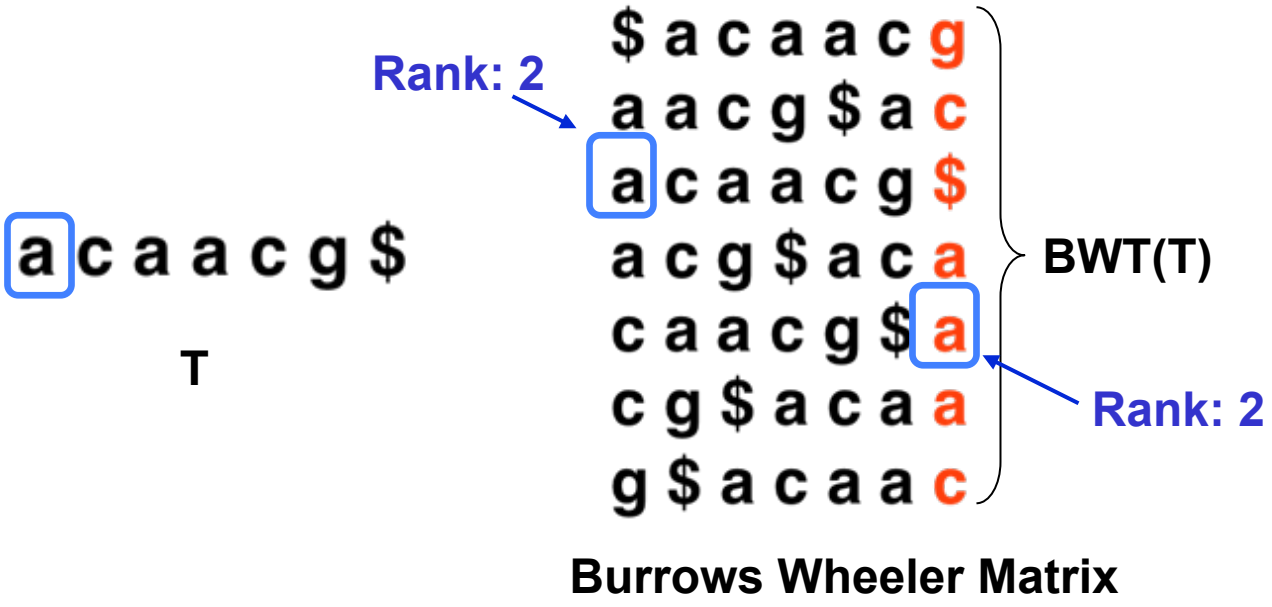
Suffix array



Burrows-Wheeler Transform (BWT)

LF mapping

- i^{th} occurrence of a character in Last column is same text occurrence as the i^{th} occurrence in First column



Recreate T from BWT

- To recreate T from BWT(T), repeatedly apply rule:
  T = **BWT**[**LF**(i)] + **T**; i = **LF**(i)
  - Where **LF**(i) maps row i to row whose first character corresponds to i's last per LF Mapping

FM Index

- Ferragina & Manzini propose "FM Index" based on BWT
- Observed:
  - LF Mapping also allows *exact matching* within T
  - **LF**(i) can be made fast with *checkpointing*
  - …and more (see FOCS paper)

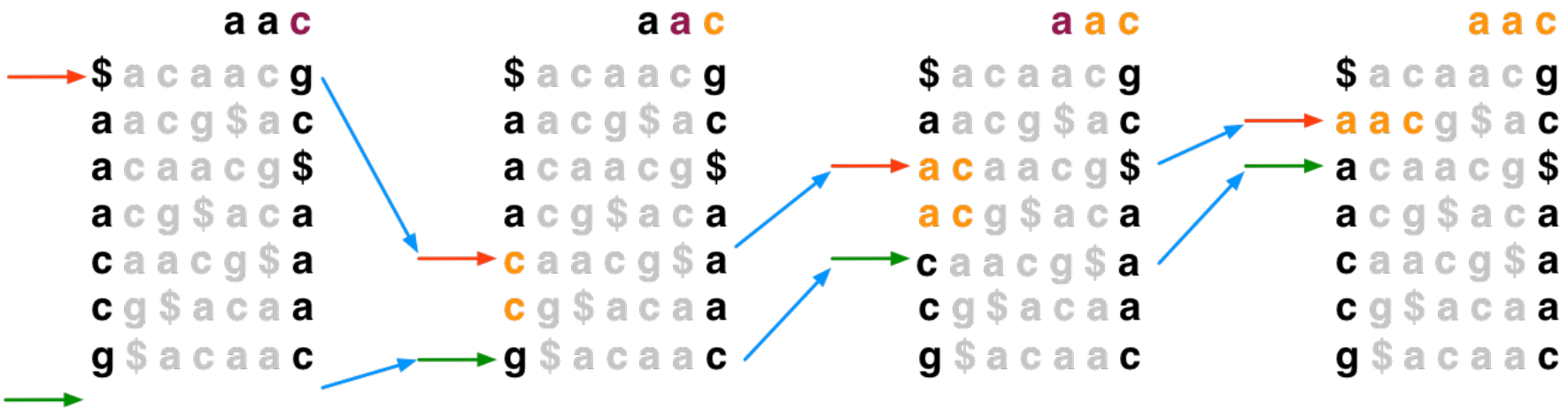
- Ferragina P, Manzini G: Opportunistic data structures with applications. *FOCS. IEEE Computer Society; 2000.*
- Ferragina P, Manzini G: An experimental study of an opportunistic index. *SIAM symposium on Discrete algorithms*. Washington, D.C.; 2001.

Exact Matching with FM Index

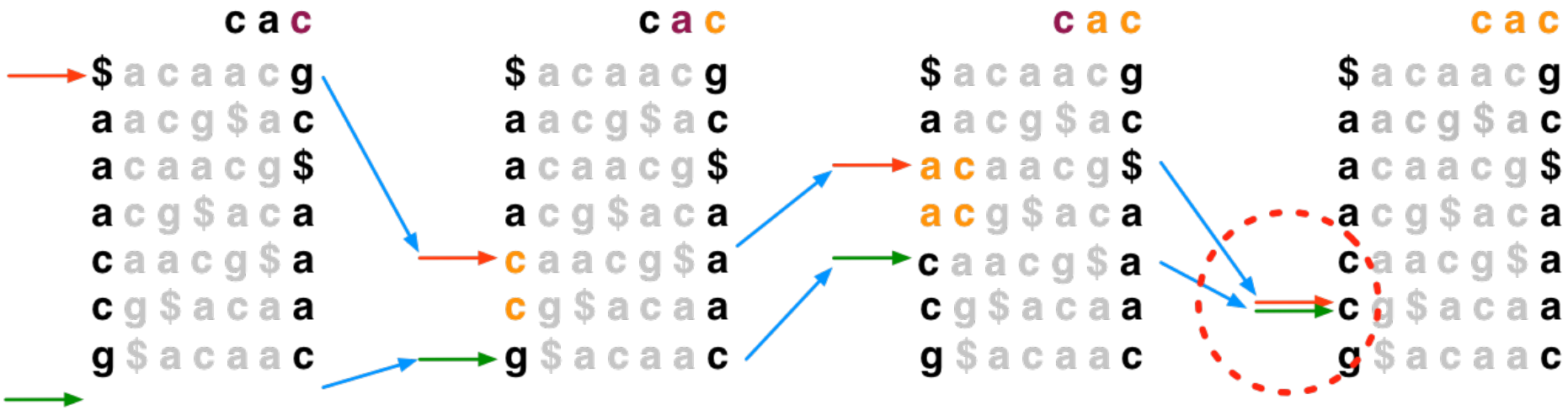
- To match Q in T using BWT(T), repeatedly apply rule:

  top = LF(top, qc); bot = LF(bot, qc)

  – Where **qc** is the next character in Q (right-to-left) and **LF**(i, **qc**) maps row i to the row whose first character corresponds to i's last character *as if it were* ***qc***



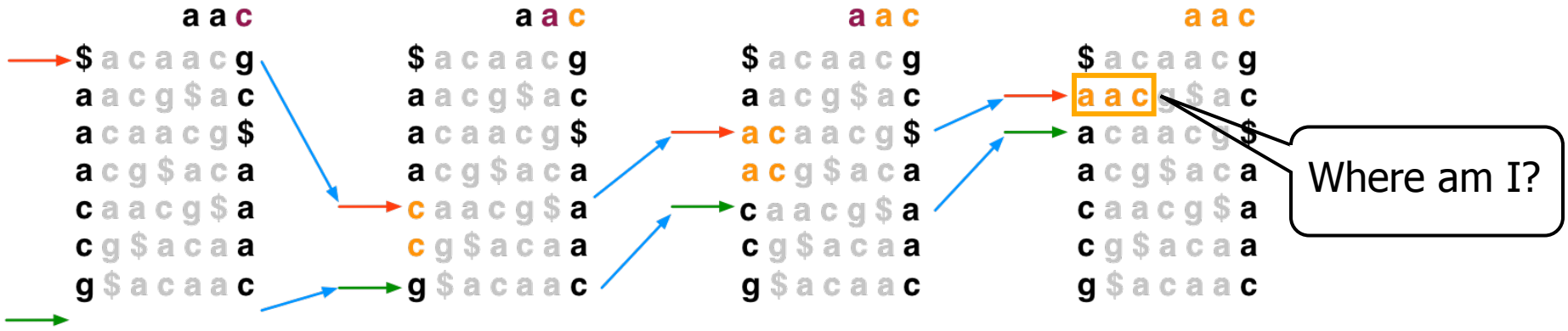
Exact Matching with FM Index



- If range becomes empty (**top** = **bot**) the query suffix (and therefore the query) does not occur in the text

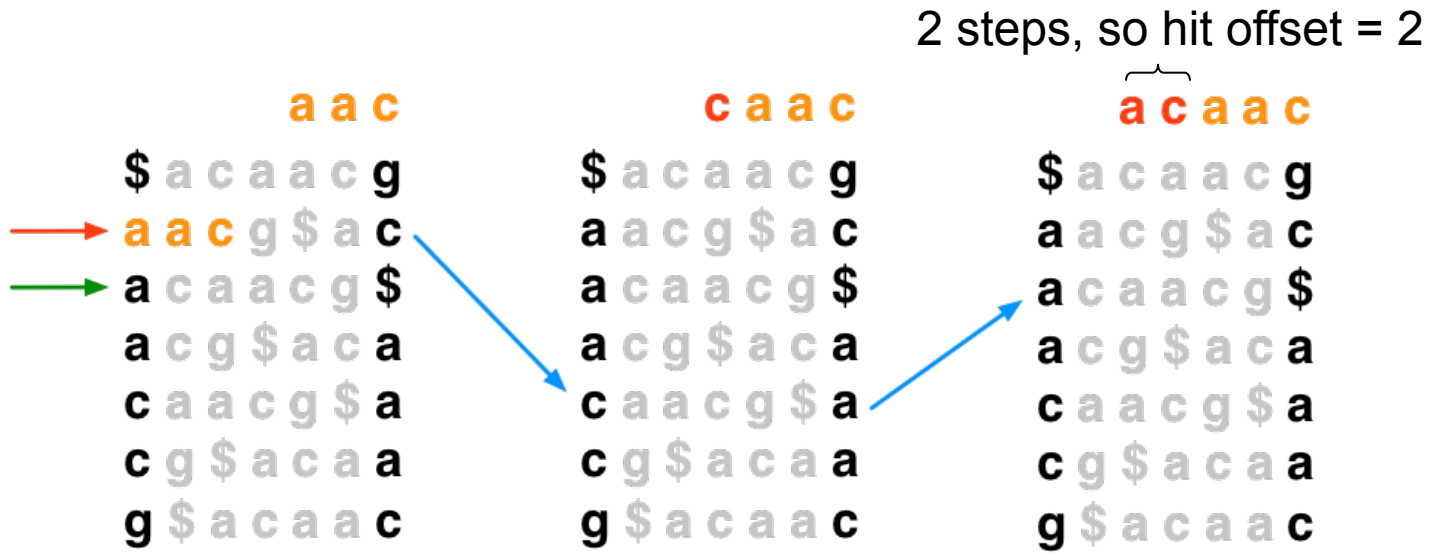
Rows to Reference Positions

- Once we know a row contains a legal alignment, how do we determine its position in the reference?



Rows to Reference Positions

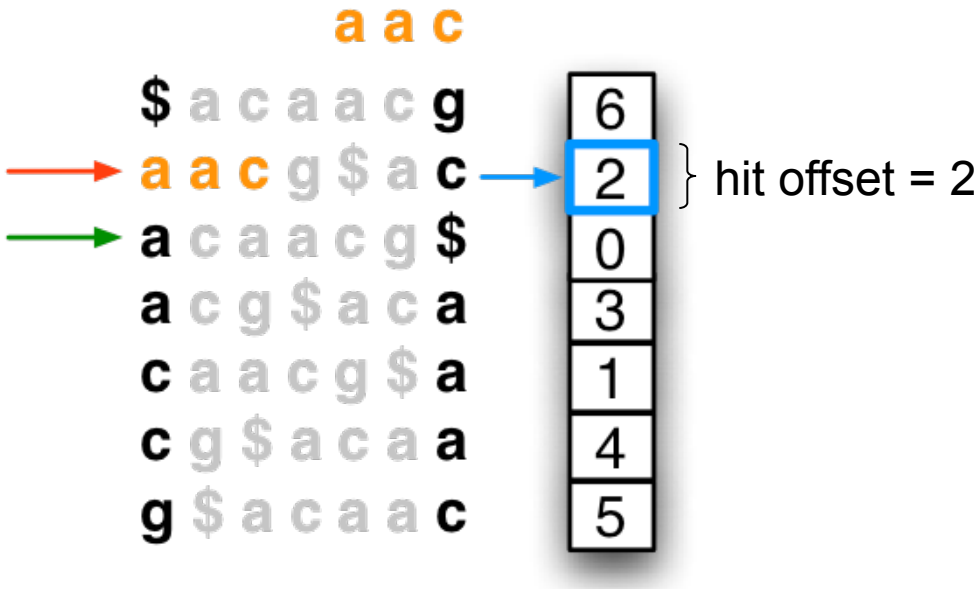
- Naïve solution 1: Use “walk-left” to walk back to the beginning of the text; number of steps = offset of hit



- Linear in length of text in general – too slow

Rows to Reference Positions

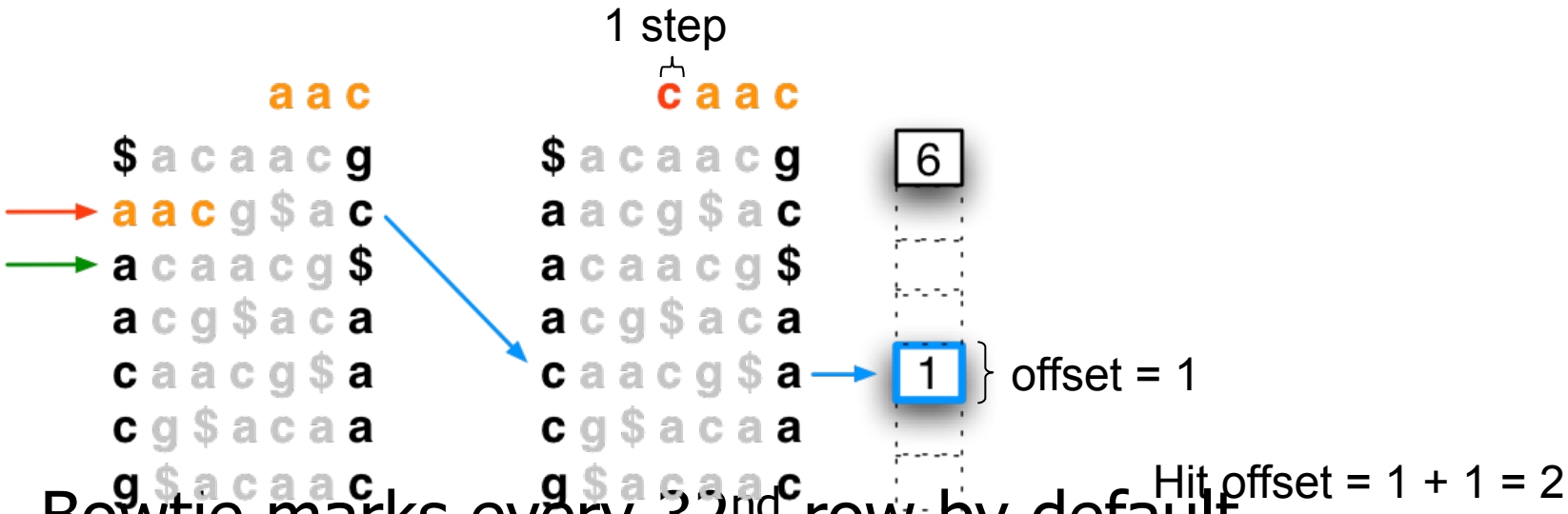
- Naïve solution 2: Keep whole suffix array in memory. Finding reference position is a lookup in the array.



- Suffix array is ~12 gigabytes for human – too big

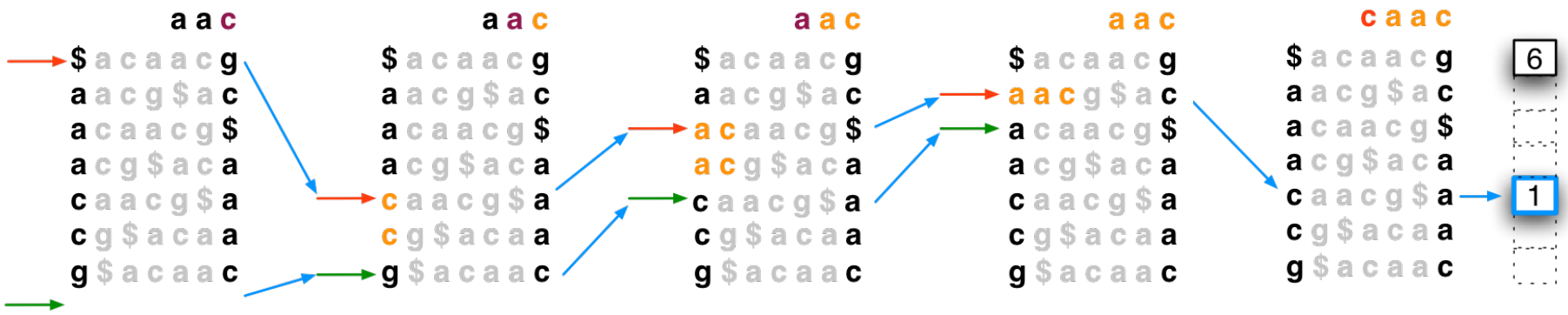
Rows to Reference Positions

- Hybrid solution: Store *sample* of suffix array; "walk left" to next sampled ("marked") row to the left
  - Due to Ferragina and Manzini



- Bowtie marks every 32nd row by default (configurable)

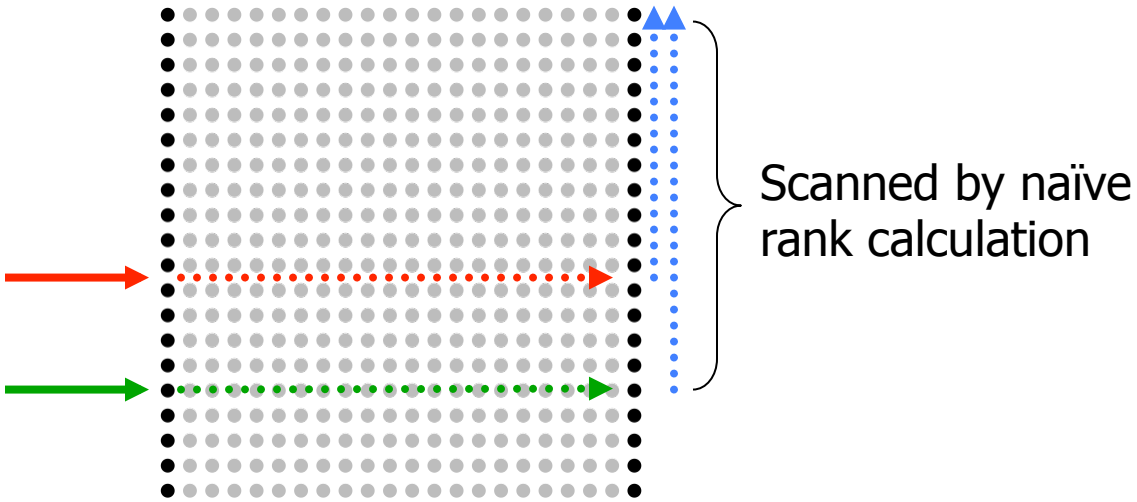
Put It All Together



- Algorithm concludes: “aac” occurs at offset 2 in “acaacg”

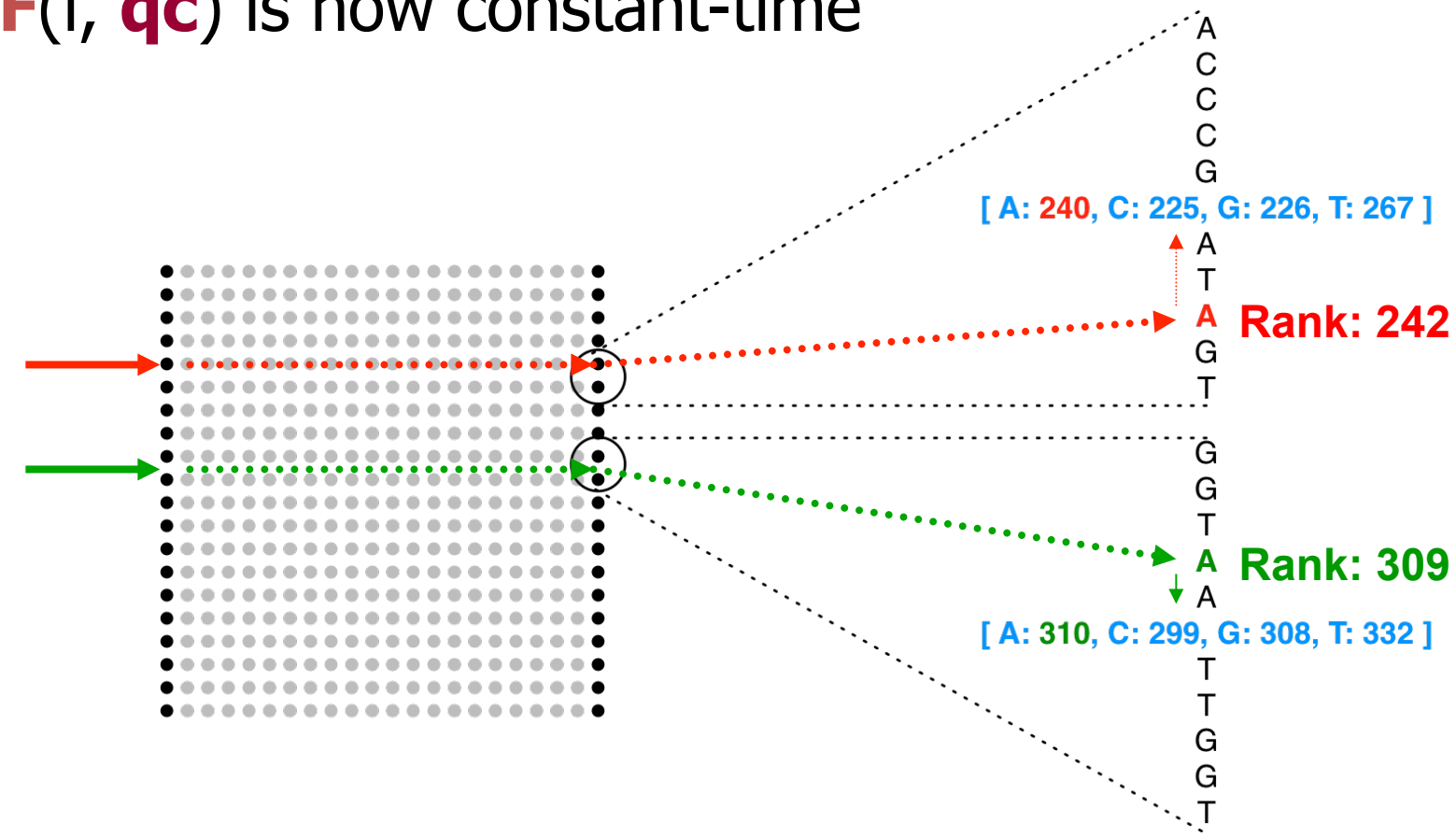
Checkpointing in FM Index

- **LF**(i, **qc**) must determine the *rank* of **qc** in row i
- Naïve way: count occurrences of **qc** in all previous rows
  - This **LF**(i, **qc**) is linear in length of text – too slow



Checkpointing in FM Index

- Solution: pre-calculate cumulative counts for A/C/G/T up to periodic **checkpoints** in BWT
- **LF**(i, **qc**) is now constant-time



FM Index is Small

Components of the FM Index:

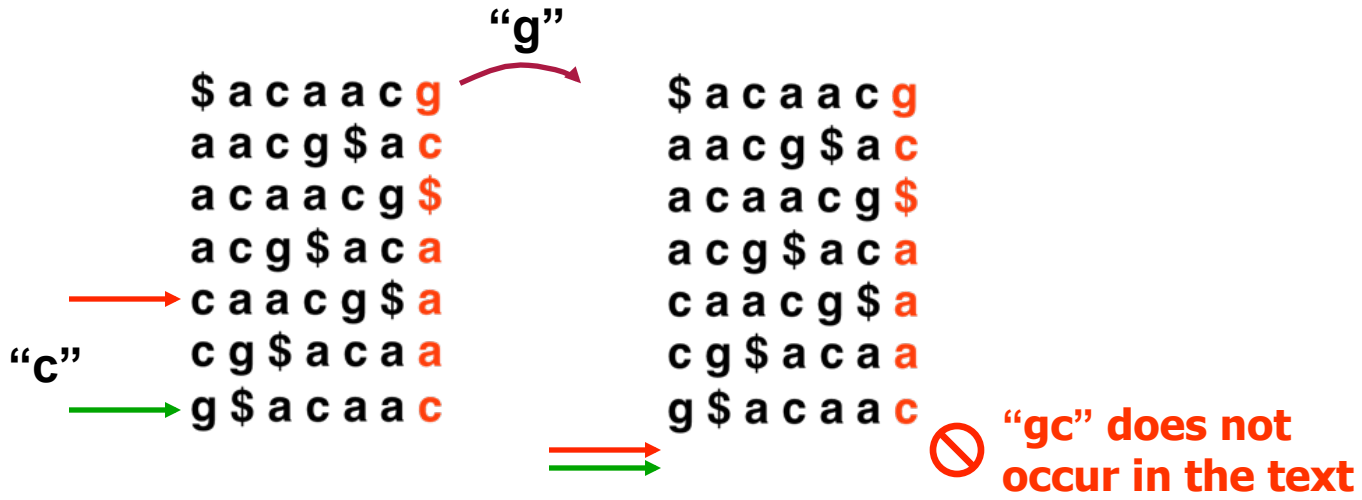
First column (*F*):	$\sim \lvert \Sigma \rvert$ integers
Last column (*L*):	*m* characters
SA sample:	$m \cdot a$ integers, where *a* is fraction of rows kept
Checkpoints:	$m \times \lvert \Sigma \rvert \cdot b$ integers, where *b* is fraction of rows checkpointed

Example: DNA alphabet (2 bits per nucleotide), *T* = human genome, $a = 1/32$, $b = 1/128$

First column (*F*):	16 bytes
Last column (*L*):	2 bits * 3 billion chars = 750 MB
SA sample:	3 billion chars * 4 bytes/char / 32 = ~ 400 MB
Checkpoints:	*3 billion* * 4 bytes/char / *128* = ~ 100 MB
	Total < 1.5 GB

Backtracking

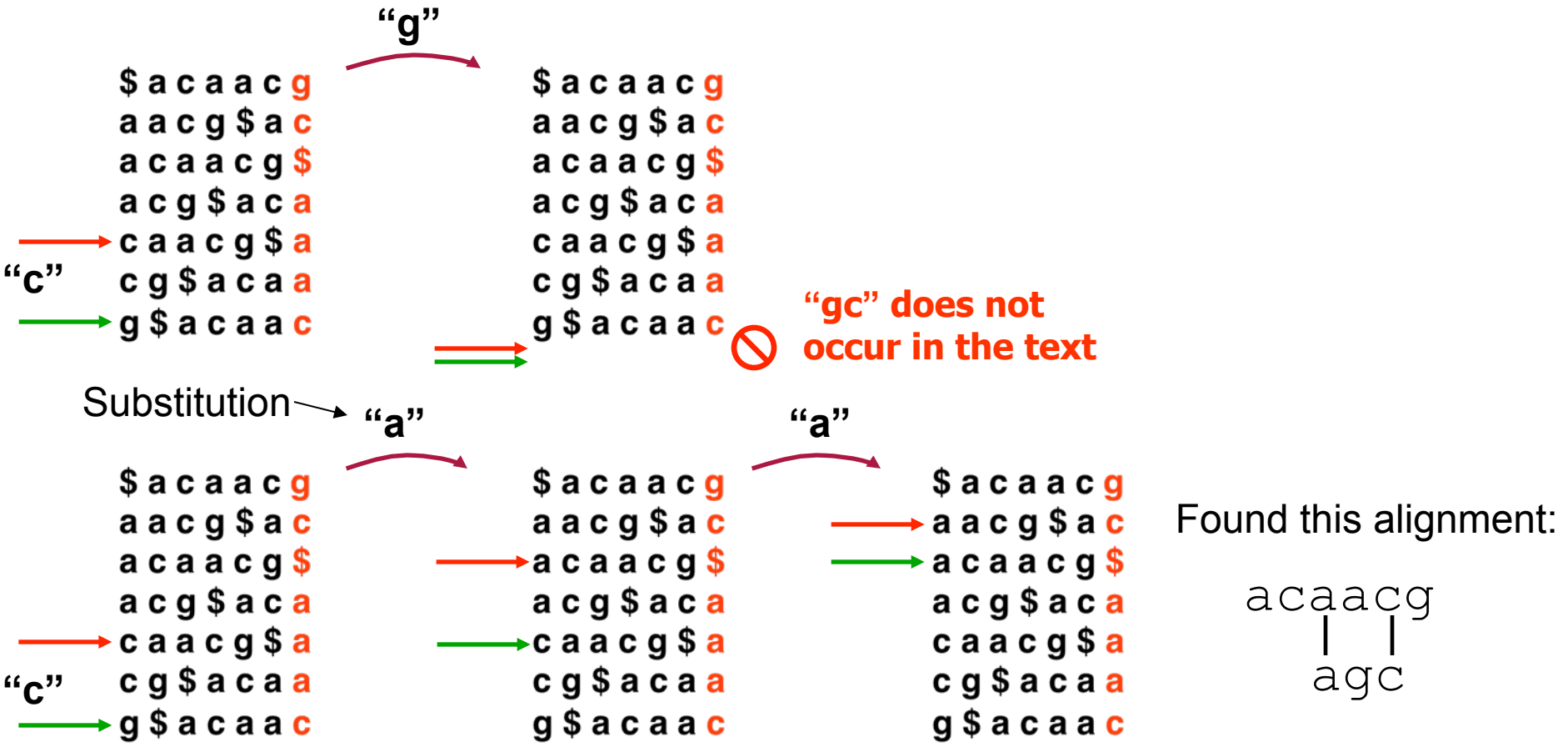
- Consider an attempt to find Q = “agc” in T = “acaacg”:



- Instead of giving up, try to “backtrack” to a previous position and try a different base

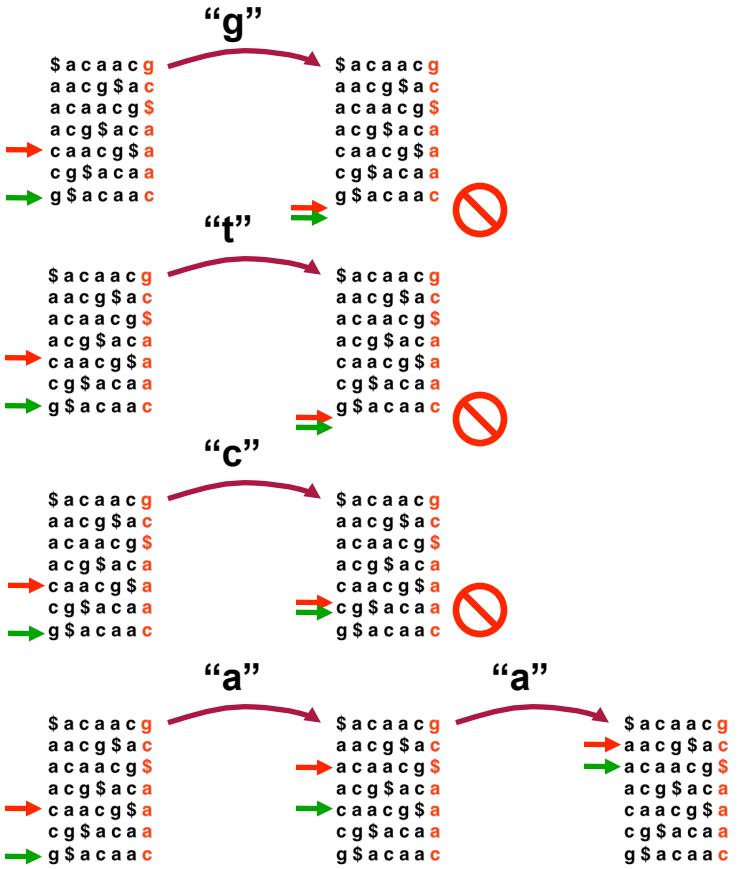
Backtracking

- Backtracking attempt for Q = “agc”, T = “acaacg”:



Backtracking

- May not be so lucky

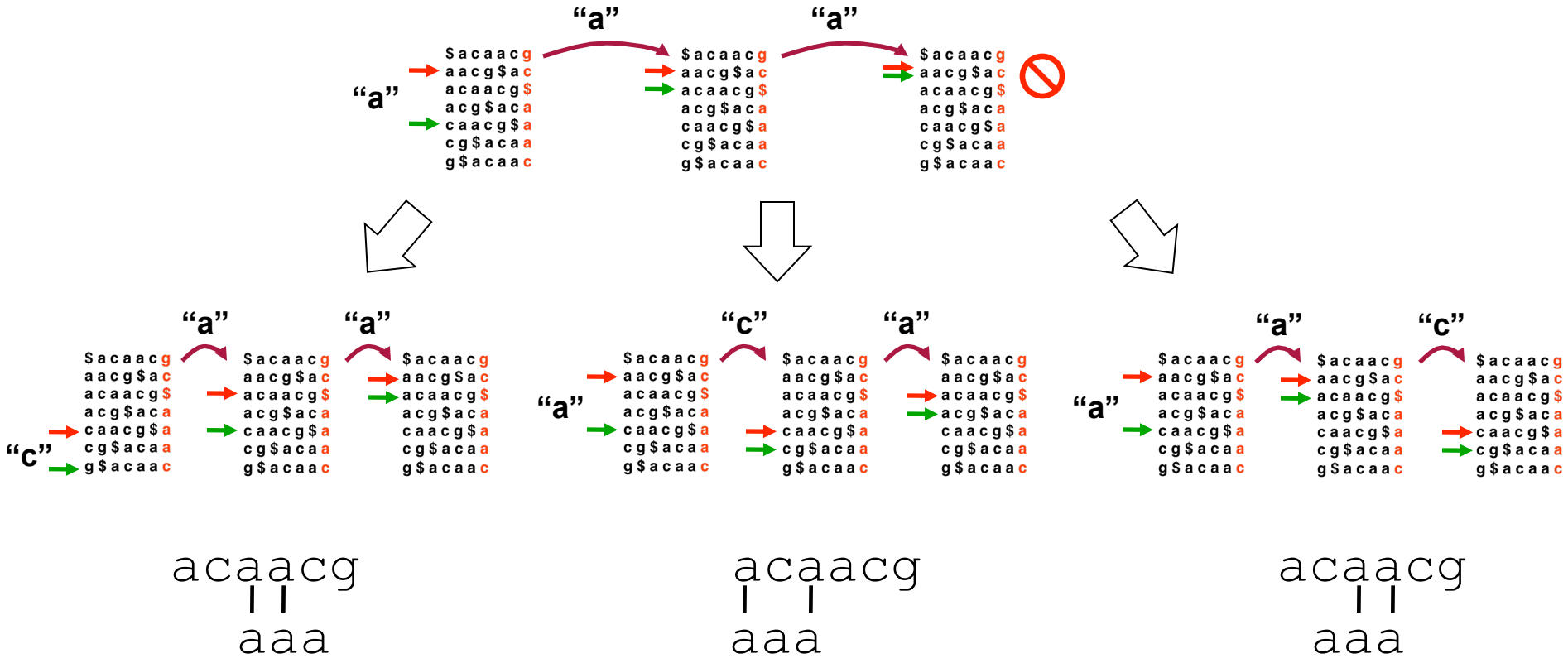


Found this alignment (eventually):

```
acaacg
 | |
 agc
```

Backtracking

- Relevant alignments may lie along multiple paths
  - E.g., Q = “aaa”, T = “acaacg”



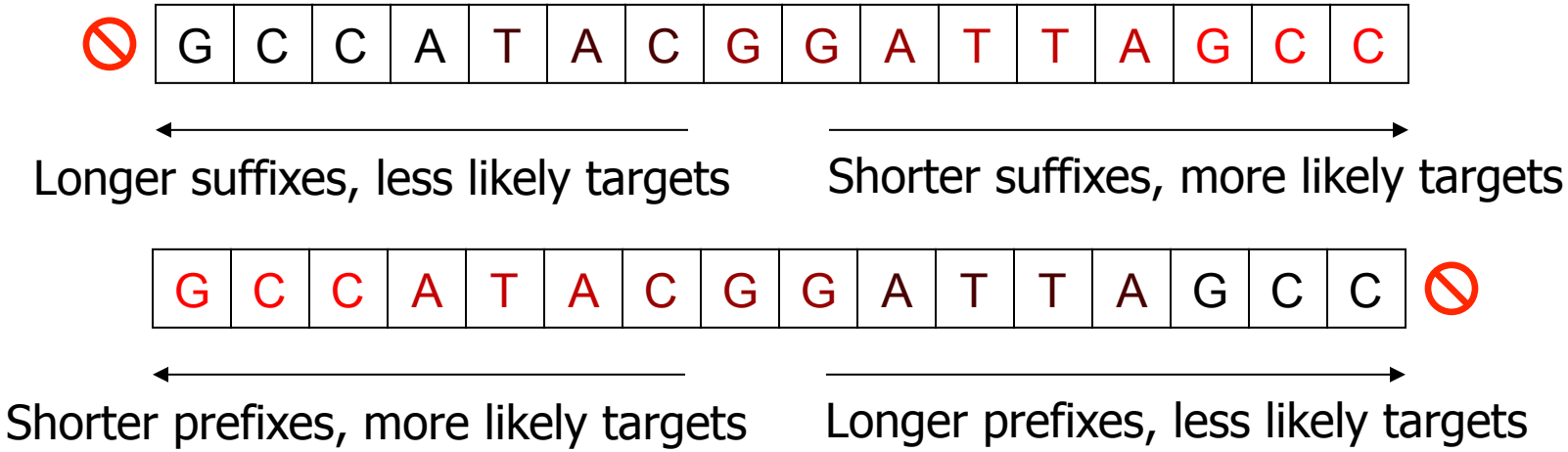
Excessive Backtracking

- Bowtie only backtracks if it can make progress, i.e., if **top** ≠ **bot** after the backtrack
  - Rightmost positions are likeliest targets because shorter suffixes are likeliest to occur "by chance"

- When >1 mismatch is allowed, such backtracks can easily dominate running time and make search slow

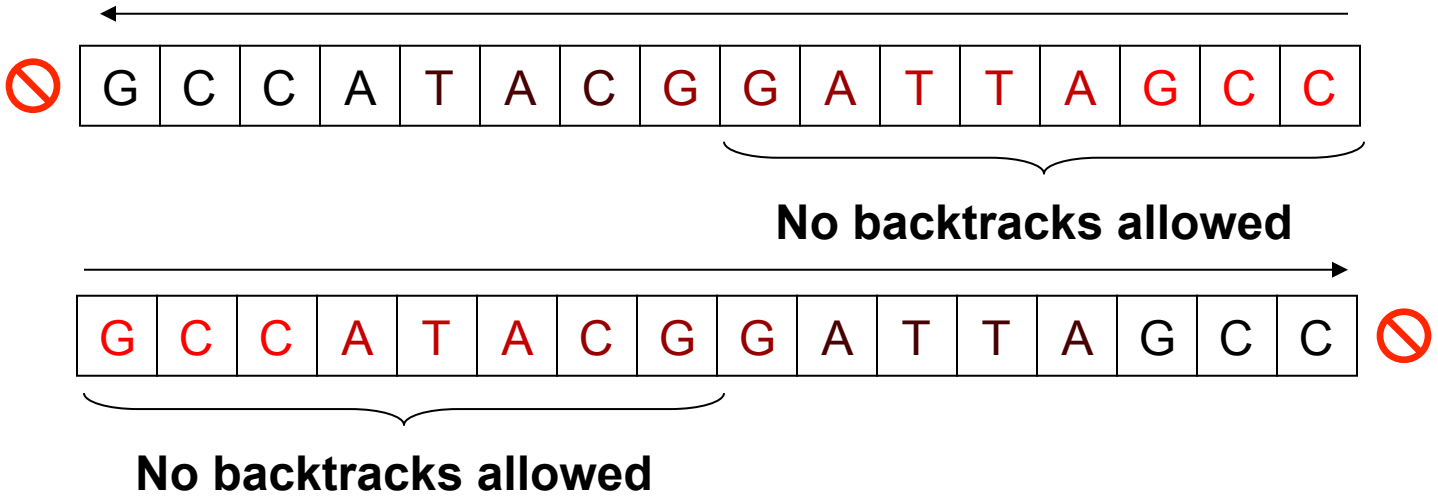
Excessive Backtracking

- Solution: Double indexing
- We've considered matching from right to left, but what if left-to-right were possible too?

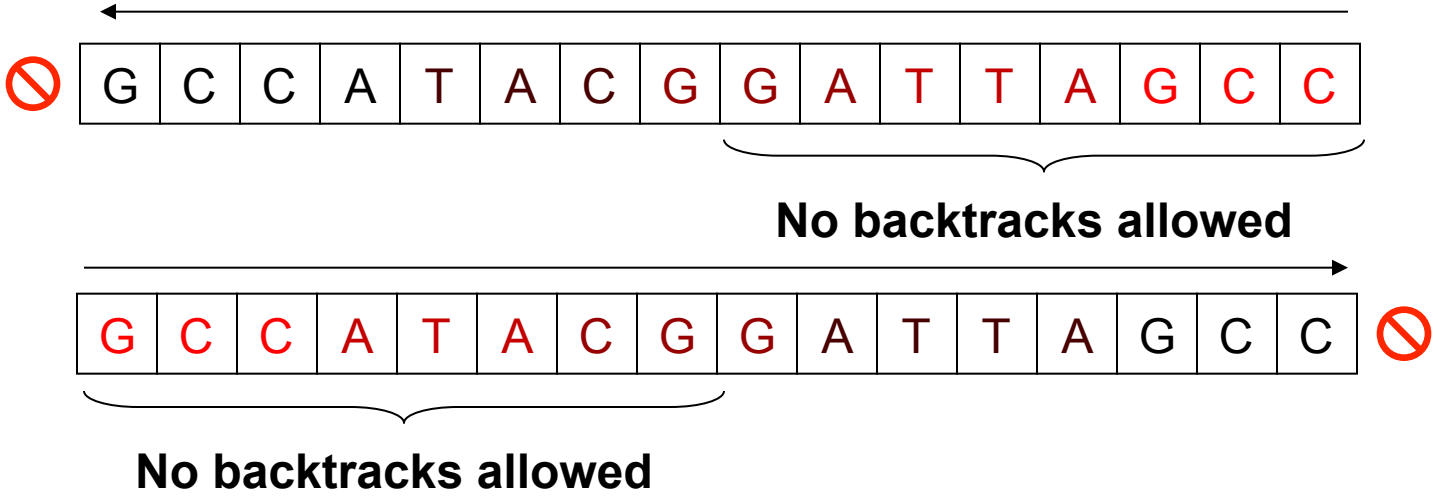


Excessive Backtracking

- Suggests a multi-stage scheme that minimizes excessive backtracking in reddest regions
  - Workflow for up to 1-mismatch that matches in both directions & disallows backtracks in reddest regions:



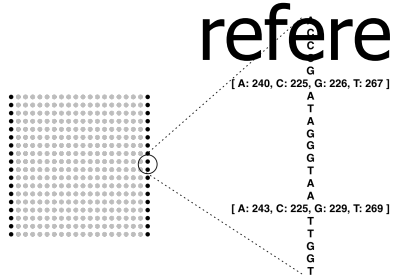
Excessive Backtracking



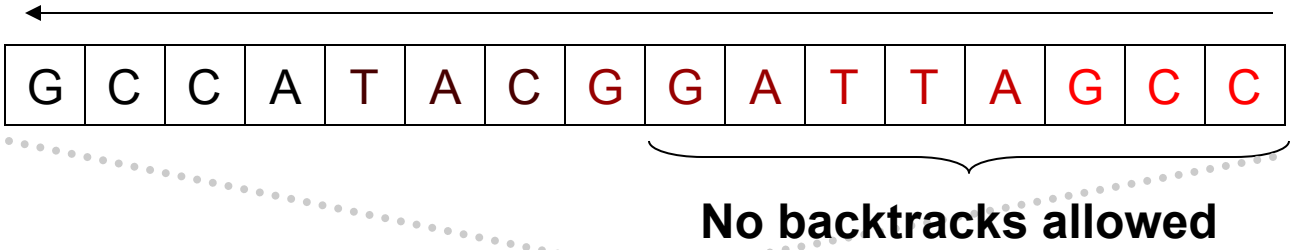
- Minimizes backtracks by disallowing backtracks in reddest regions
- Maintains full sensitivity by matching in both directions

Excessive Backtracking

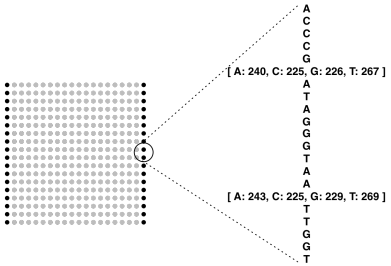
- But how to match left-to-right?
- Double indexing:
  - Reverse read and use “mirror index”: index for reference with sequence reversed



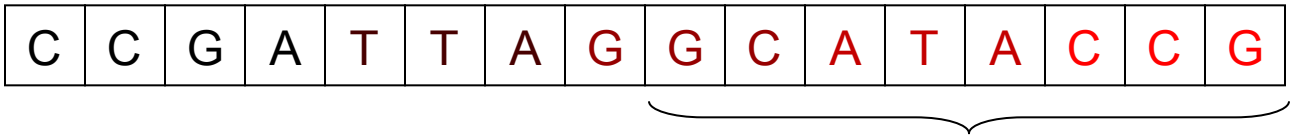
Forward Index



No backtracks allowed



Mirror Index



No backtracks allowed

Coverage/Depth

- Coverage/depth is how many times that your data covers the genome (on average)

$$C=N*L/G$$

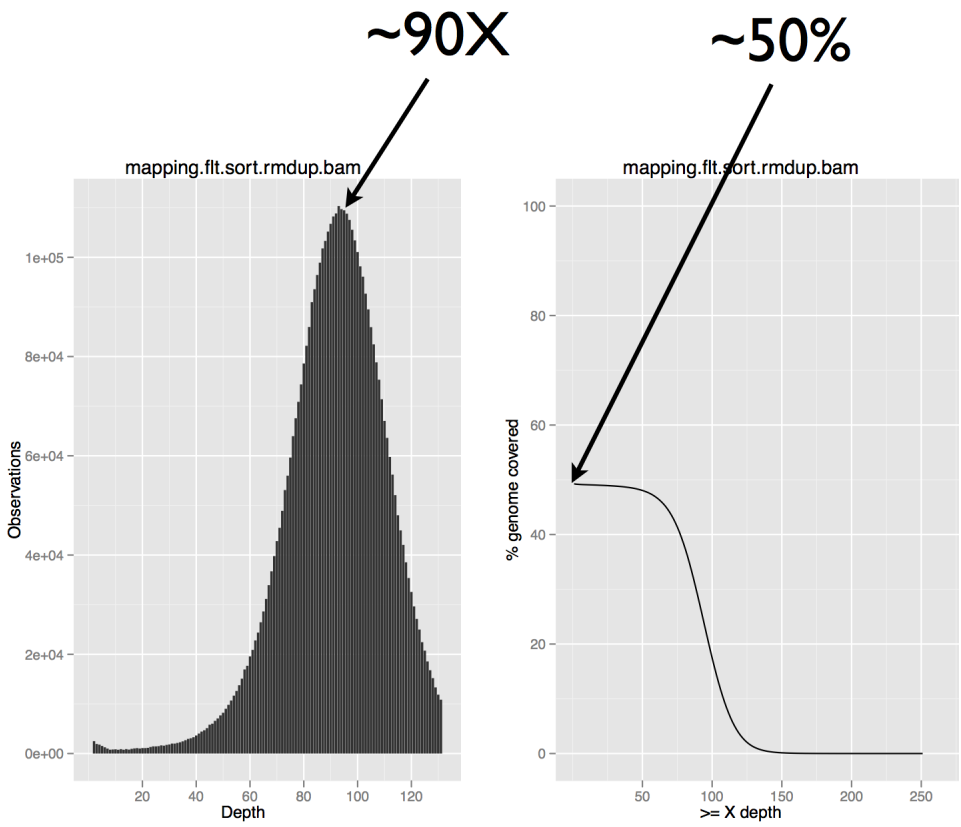
- Example:
  - N : number of reads: 10M
  - L: Read length: 50
  - G: Genome size: 5 M bases
  - C = 10*50/5 = 100X

- On average there are 100 reads covering each position in the genome

Actual depth

- We aligned reads to the genome - how much do we actually cover?

- Avg. depth ~ 90X
- Range from 0-250X
- Only 50% of the genome was covered with reads



Single vs. Paired alignment

- Always get paired end reads (if possible)
- Can map across repeats
- Less mapping errors
- Unmapped read can be “rescued” by a good aligning mate

Output: SAM/BAM format

- Sequence Alignment/Map format (SAM)
- BAM = Binary SAM and zipped (compressed format of SAM)
- Two sections
  - Header: All lines start with "@"
  - Alignments: All other lines

Thank you for your attention