Move structure in practice: memory and speed

Ben Langmead

r-index critique

Match query for length-m string

	Time	Space
r-index	$O(m)$	$O(r \log \log_w(\sigma + n/r))$
	$O(m \log \log_w(\sigma + n/r))$	$O(r)$

Takaaki Nishimoto and Yasuo Tabei. Optimal-Time Queries on BWT-Runs Compressed Indexes. In 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021). Leibniz International Proceedings in Informatics (LIPIcs), Volume 198, pp. 101:1-101:15

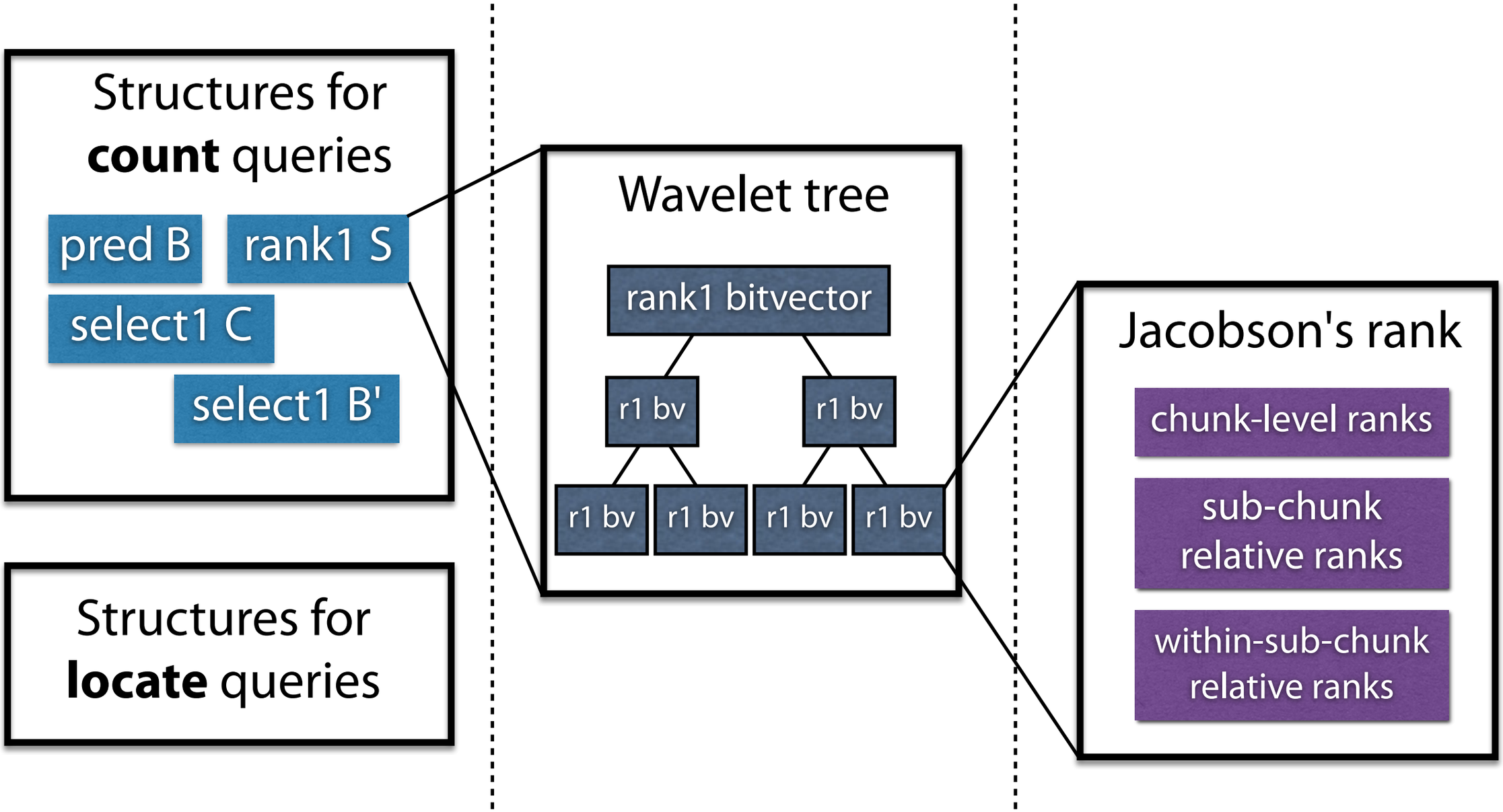
r-index critique

Match query for length-m string

	Time	Space
r-index	$O(m)$	$O(r \log \log_w(\sigma + n/r))$
	$O(m \log \log_w(\sigma + n/r))$	$O(r)$
Move	$O(m)$	$O(r)$

Takaaki Nishimoto and Yasuo Tabei. Optimal-Time Queries on BWT-Runs Compressed Indexes. In 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021). Leibniz International Proceedings in Informatics (LIPIcs), Volume 198, pp. 101:1-101:15

Another r-index critique



Another r-index critique

Main memory

https://en.wikipedia.org/wiki/George_Peabody_Library

Cache memory

https://unsplash.com/photos/books-and-pencil-on-wooden-table-Q8HfuO9udts

For *recently* or *frequently* used data

Processor

Another r-index critique

Main memory

https://en.wikipedia.org/wiki/George_Peabody_Library

When needed data was accessed recently, or is near data that was accessed recently, we can get it quickly from cache

Cache memory

https://unsplash.com/photos/books-and-pencil-on-wooden-table-Q8HfuO9udts

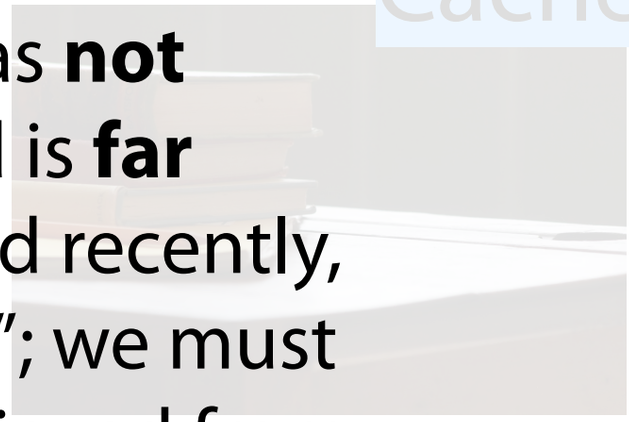
Processor

Another r-index critique

Main memory

https://en.wikipedia.org/wiki/George_Peabody_Library

Cache memory

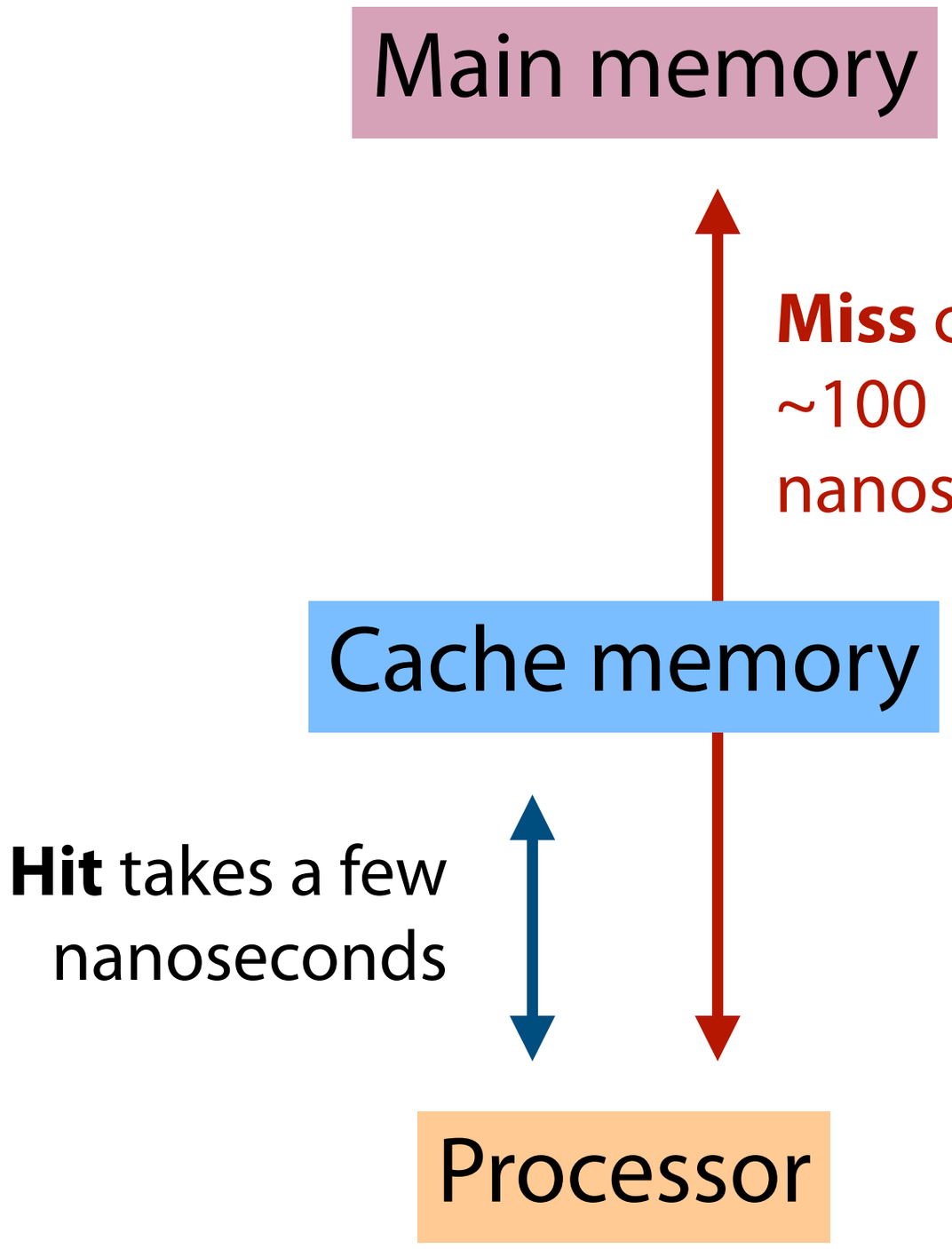


https://unsplash.com/photos/books-and-pencil-on-wooden-table-Q8HfuO9udts

When needed data was **not** accessed recently, and is **far** from any data accessed recently, we have a "cache miss"; we must wait as the data is retrieved from main memory

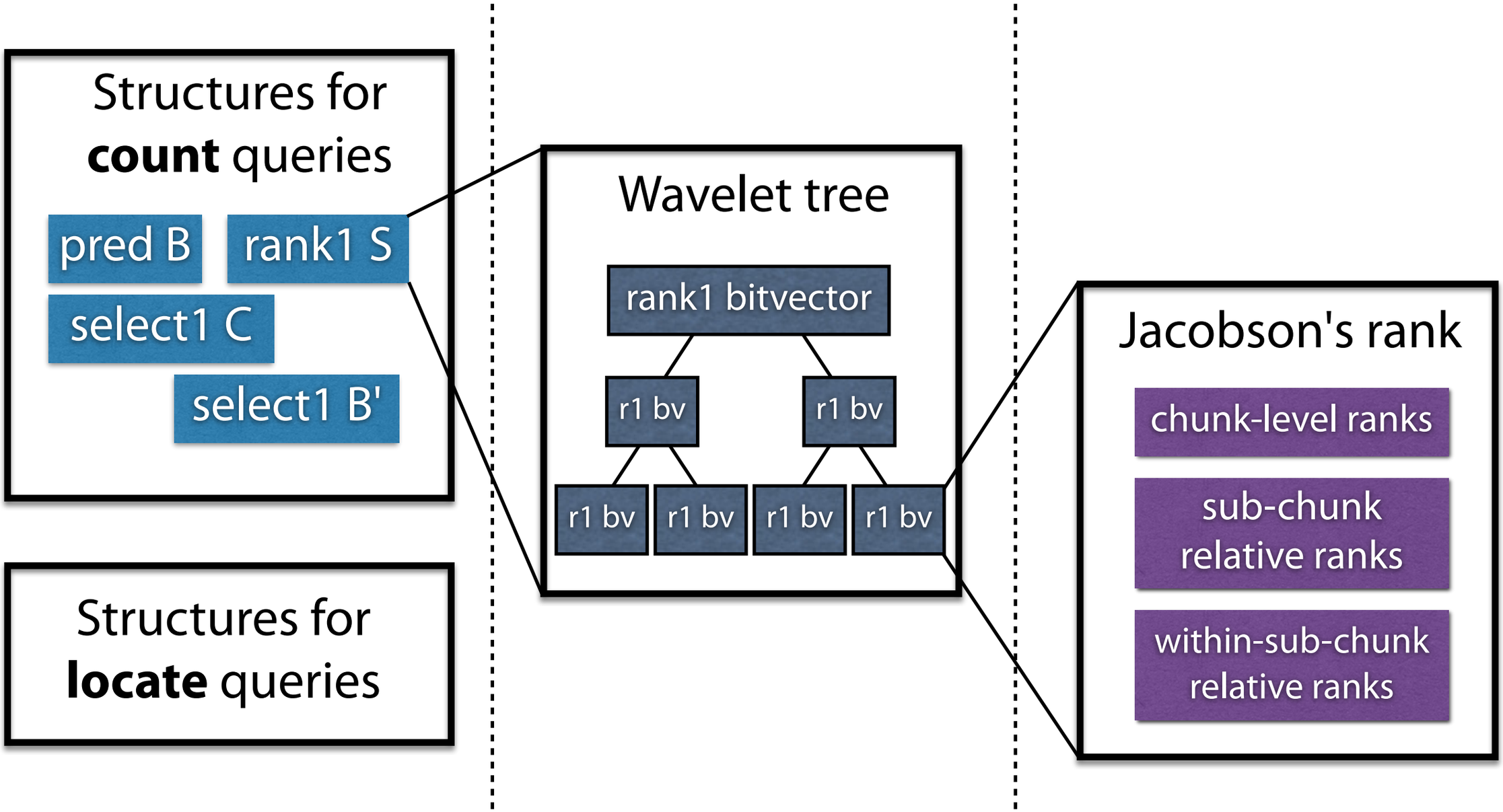
Processor

Another r-index critique



An algorithm whose memory accesses mostly hit in the cache has good "locality of reference"

Another r-index critique



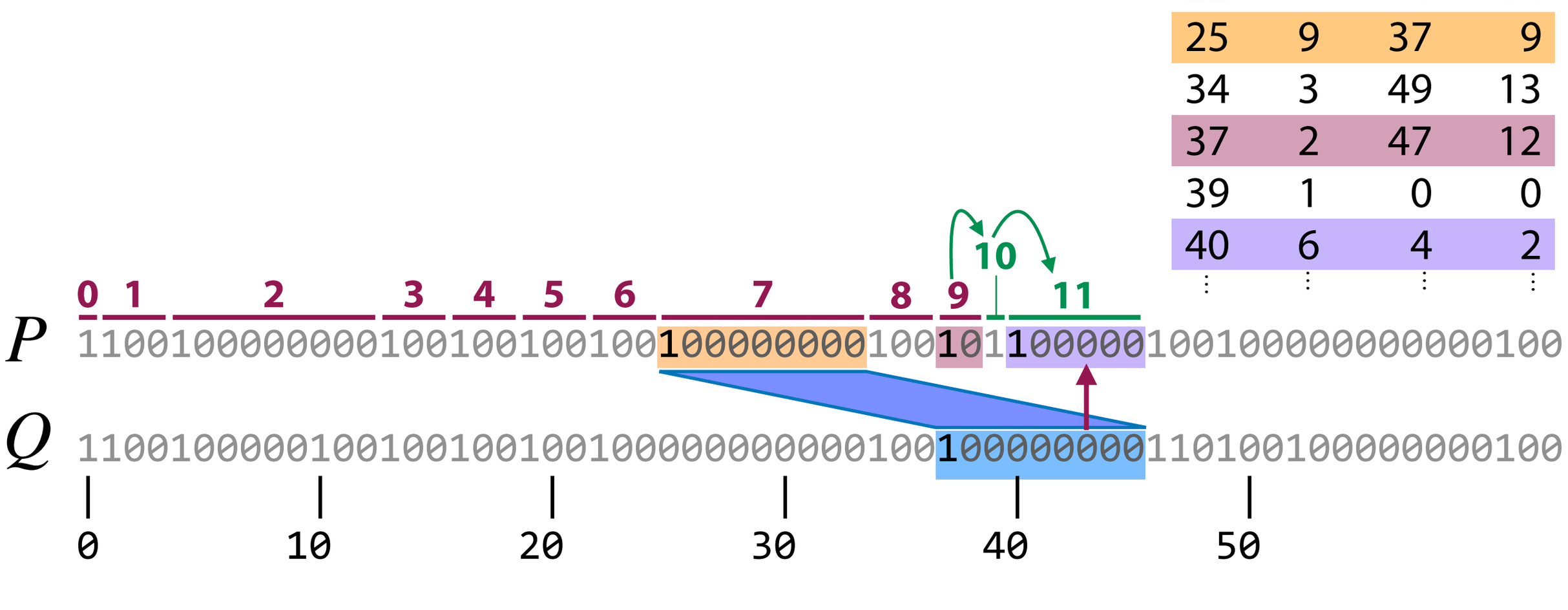
Accesses to many parts of many disparate structures means:
poor locality → many cache misses → lower speed

Query reminder

First we make a big move to $M[i] \,.\, \xi$

Then we make smaller moves to compute the final i'

$\langle\, p,$	$\ell,$	$\pi,$	$\xi\,\rangle$
0	1	46	12
1	3	34	8
4	9	52	13
13	3	19	5
16	3	1	1
19	3	16	4
22	3	61	14
25	9	37	9
34	3	49	13
37	2	47	12
39	1	0	0
40	6	4	2
⋮	⋮	⋮	⋮



Query reminder

Function $\pi(k, i)$**:**

$k' \leftarrow M[i].\pi + (k - M[i].p)$

$i' \leftarrow M[i].\xi$

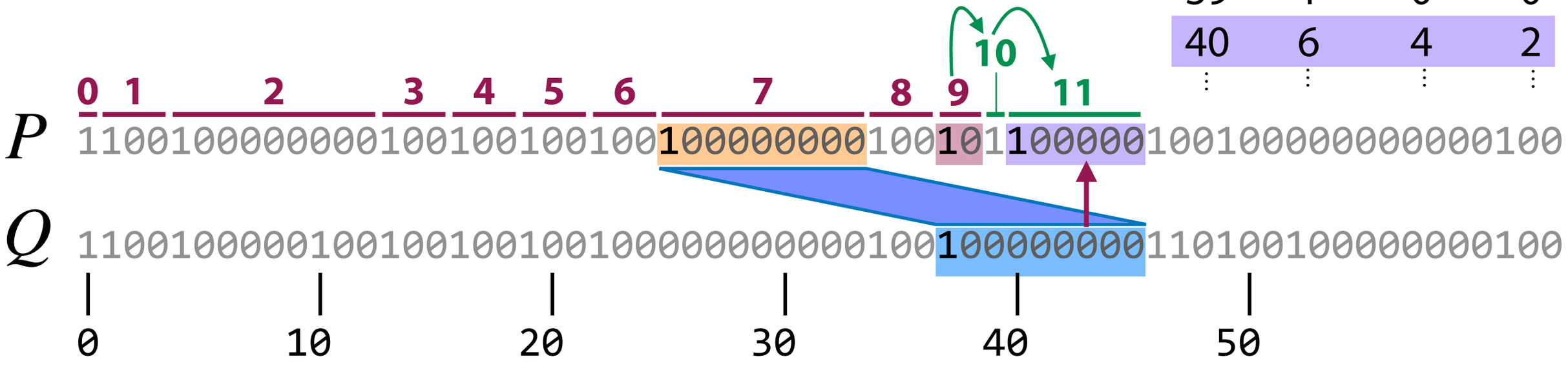
while $k' \geq (M[i'].p + M[i'].\ell)$ **do**

$i' \leftarrow i' + 1$

end

return $\langle k', i' \rangle$

$\langle\, p,$	$\ell,$	$\pi,$	$\xi\,\rangle$
0	1	46	12
1	3	34	8
4	9	52	13
13	3	19	5
16	3	1	1
19	3	16	4
22	3	61	14
25	9	37	9
34	3	49	13
37	2	47	12
39	1	0	0
40	6	4	2
⋮	⋮	⋮	⋮



Idea: hide memory latency

Function $\pi(k, i)$**:**

$k' \leftarrow M[i].\pi + (k - M[i].p)$

$i' \leftarrow M[i].\xi$

while $k' \geq (M[i'].p + M[i'].\ell)$ **do**

$i' \leftarrow i' + 1$

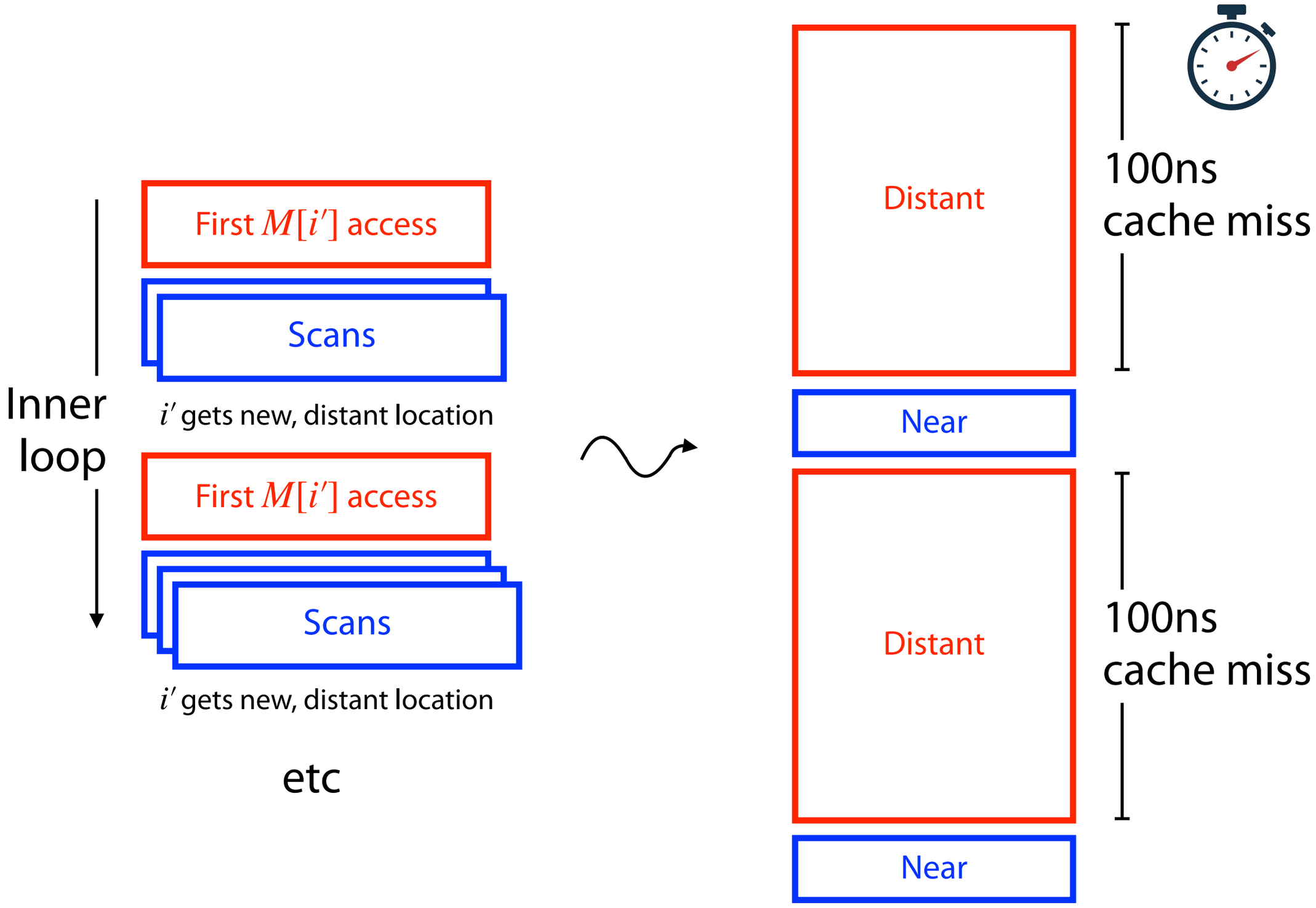
end

return $\langle k', i' \rangle$

First access to destination row of move table jumps to a distant memory address; **cache miss**

Later iterations scan sequentially along rows; **cache hits**

Idea: hide memory latency



Idea: prefetching

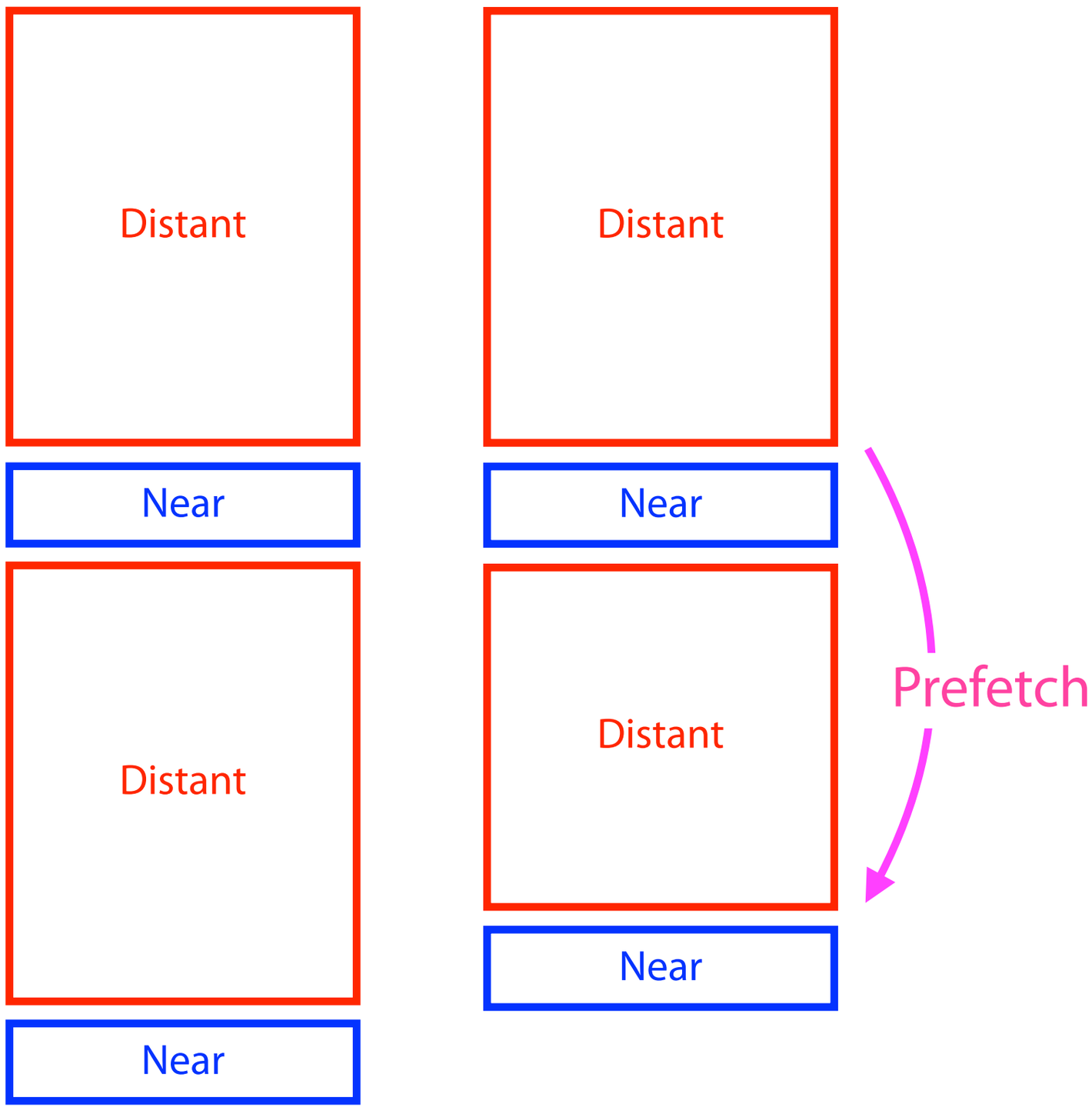
```
__builtin_prefetch (const void *addr)
```

Asynchronous request to retrieve data & install it in the cache

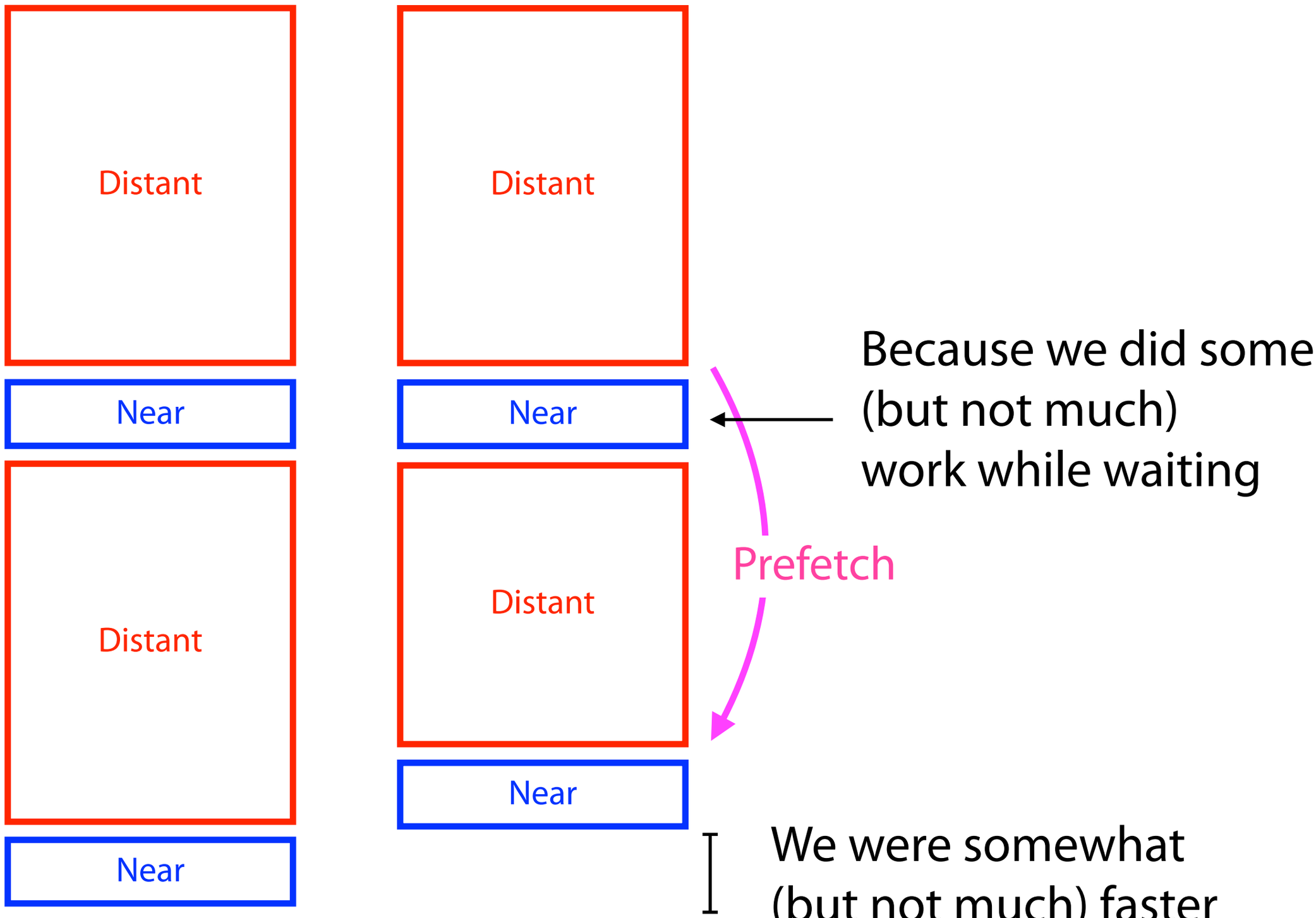
Does *asynchronously* what would otherwise have happened *synchronously*

Why ***asynchronously***? Because sometimes we can do productive work while we wait.

Idea: prefetching



Idea: prefetching



Idea: interleaving queries

Usually we have **many queries** (sequencing reads) to match

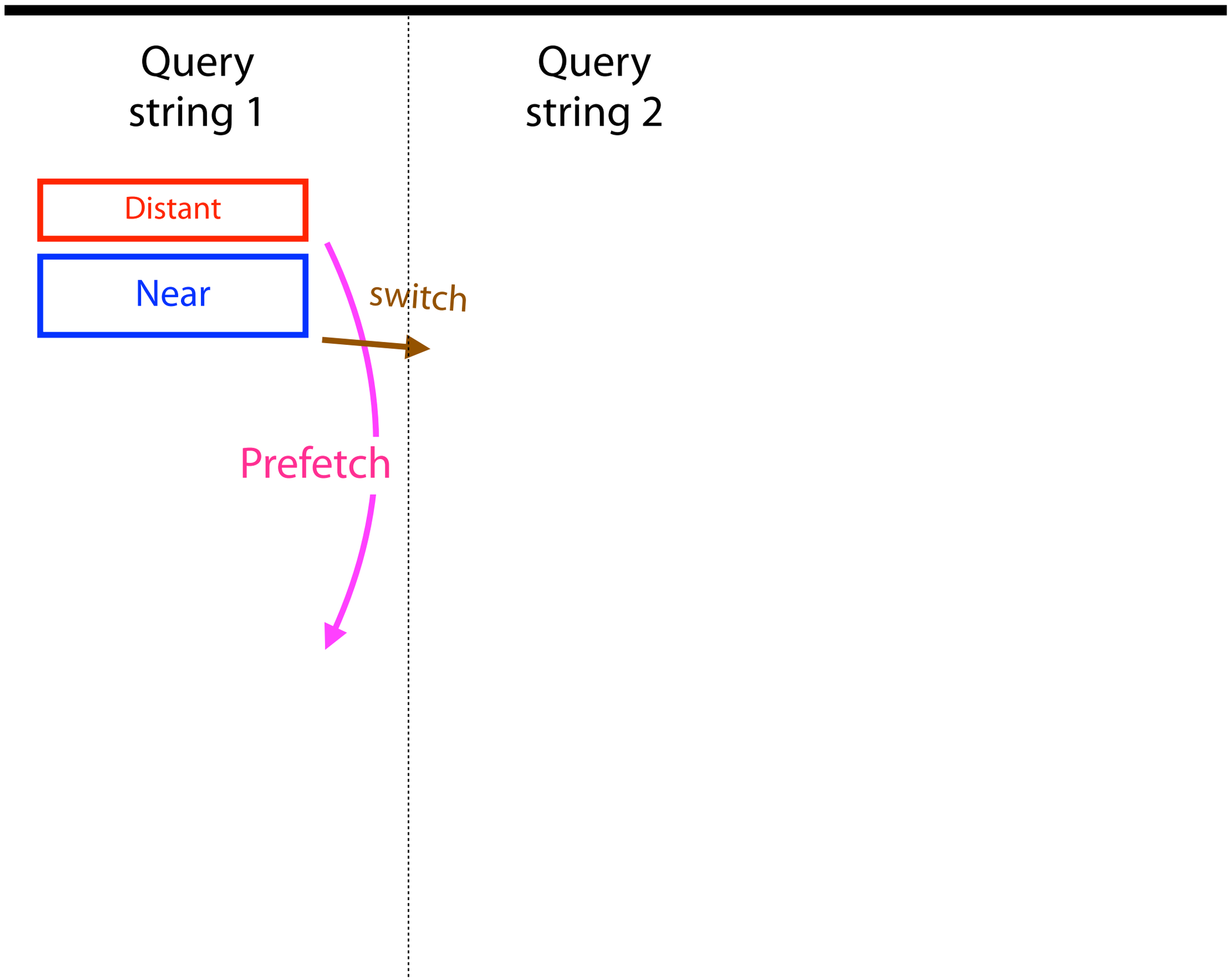
Image: Illumina Inc.

```
@Read1
AAAGGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGA
TCTGTTCYCTAAACGAACTTTAAAATCTGTGTGGCTGTCACTCGGC
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read2
TAGATCTGTTCYCTAAACGAACTTTAAAATCTGTGTGGCTGTCACTCGGCTGCA
TGCTTAGTGCACTCACGCAGTATAATTAATAACTAATTACTGTCGT
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read3
TGCATGCTTAGTGCACTCACGCAGTATAATTAATAACTAATTACTGTCGTTGAC
AGGACACGAGTAACTCGTCTATCTTCTGCAGGCTGCTTACGGTTTC
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read4
TGACAGGACACGAGTAACTCGTCTATCTTCTGCAGGCTGCTTACGGTTTCGTCC
GTGTTGCAGCCGATCATCAGCACATCTAGGTTTCGTCCGGGTGTGA
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read5
GTCCGTGTTGCAGCCGATCATCAGCACATCTAGGTTTCGTCCGGGTGTGACCGA
AAGGTAAGATGGAGAGCCTTGTCCCTGGTTTCAACGAGAAAACACA
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read6
CCGAAAGGTAAGATGGAGAGCCTTGTCCCTGGTTTCAACGAGAAAACACACGTC
CAACTCAGTTTGCCTGTTTTACAGGTTCGCGACGTGCTCGTACGTG
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read7
CGTCCAACTCAGTTTGCCTGTTTTACAGGTTCGCGACGTGCTCGTACGTGGCTT
TGGAGACTCCGTGGAGGAGGTCTTATCAGAGGCACGTCAACATCTT
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read8
GCTTTGGAGACTCCGTGGAGGAGGTCTTATCAGAGGCACGTCAACATCTTAAAG
ATGGCACTTGTGGCTTAGTAGAAGTTGAAAAAGGCGTTTTGCCTCA
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read9
AAAGATGGCACTTGTGGCTTAGTAGAAGTTGAAAAAGGCGTTTTGCCTCAACTT
GAACAGCCCTATGTGTTCATCAAACGTTCGGATGCTCGAACTGCAC
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read10
ACTTGAACAGCCCTATGTGTTCATCAAACGTTCGGATGCTCGAACTGCACCTCA
TGGTCATGTTATGGTTGAGCTGGTAGCAGAACTCGAAGGCATTCAG
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
```

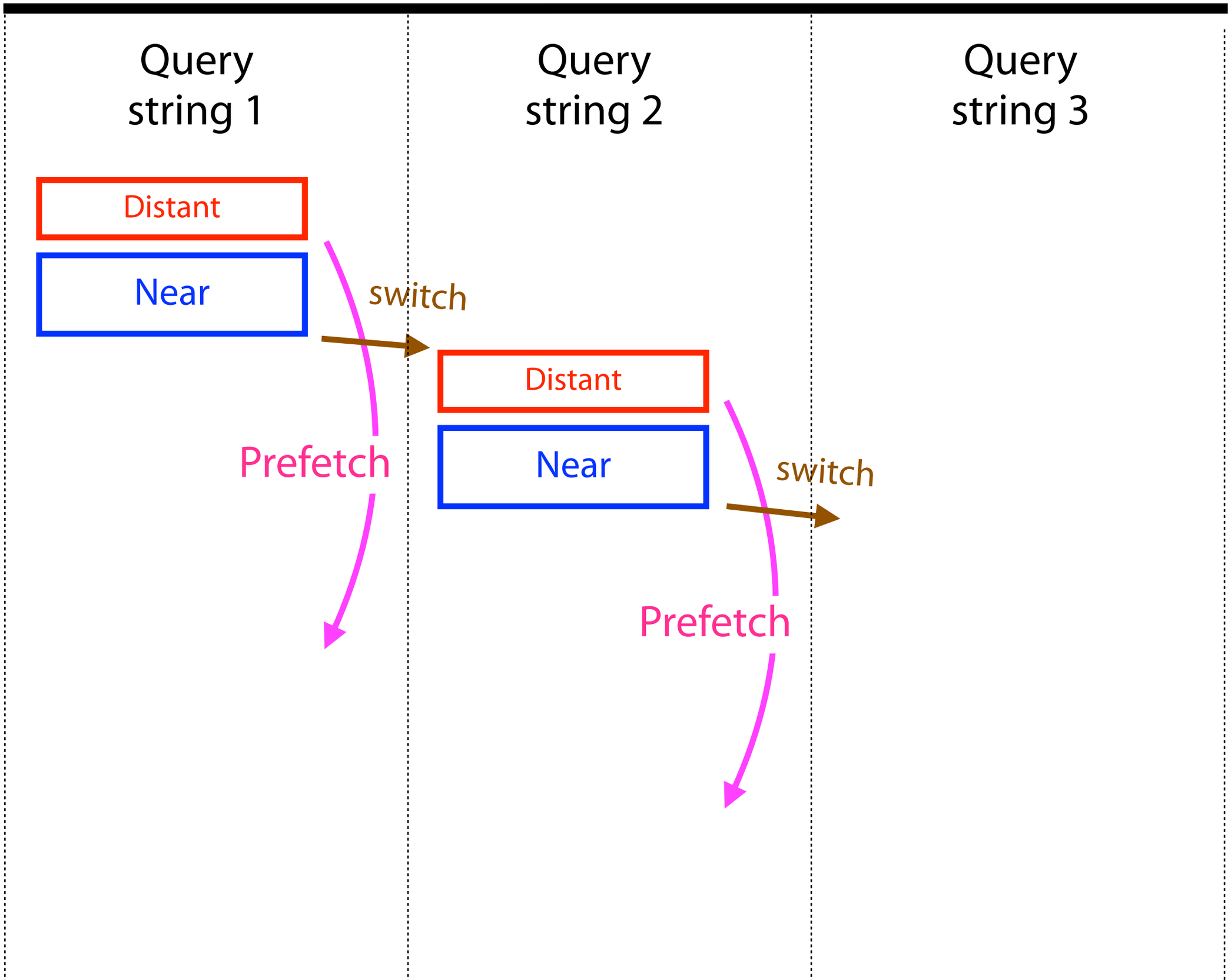
```
@Read11
CTCATGGTCATGTTATGGTTGAGCTGGTAGCAGAACTCGAAGGCATTCAGTACG
GTCGTAGTGGTGAGACACTTGGTGTCCTTGTCCCTCATGTGGGCGA
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read12
TACGGTCGTAGTGGTGAGACACTTGGTGTCCTTGTCCCTCATGTGGGCGAAATA
CCAGTGGCTTACCGCAAGGTTCTTCTTCGTAAGAACGGTAATAAAG
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read13
AATACCAGTGGCTTACCGCAAGGTTCTTCTTCGTAAGAACGGTAATAAAGGAGC
TGGTGGCCATAGTTACGGCGCCGATCTAAAGTCATTTGACTTAGGC
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read14
GAGCTGGTGGCCATAGTTACGGCGCCGATCTAAAGTCATTTGACTTAGGCGACG
AGCTTGGCACTGATCCTTATGAAGATTTTCAAGAAAACTGGAACAC
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read15
GACGAGCTTGGCACTGATCCTTATGAAGATTTTCAAGAAAACTGGAACACTAAA
CATAGCAGTGGTGTTACCCGTGAACTCATGCGTGAGCTTAACGGAG
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read16
TAAACATAGCAGTGGTGTTACCCGTGAACTCATGCGTGAGCTTAACGGAGGGGC
ATACACTCGCTATGTCGATAACAACTTCTGTGGCCCTGATGGCTAC
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read17
GGGCATACACTCGCTATGTCGATAACAACTTCTGTGGCCCTGATGGCTACCCTC
TTGAGTGCATTAAAGACCTTCTAGCACGTGCTGGTAAAGCTTCATG
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read18
CCTCTTGAGTGCATTAAAGACCTTCTAGCACGTGCTGGTAAAGCTTCATGCACT
TTGTCCGAACAACTGGACTTTATTGACACTAAGAGGGGTGTATACT
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read19
CACTTTGTCCGAACAACTGGACTTTATTGACACTAAGAGGGGTGTATACTGCTG
CCGTGAACATGAGCATGAAATTGCTTGGTACACGGAACGTTCTGAA
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@Read20
GCTGCCGTGAACATGAGCATGAAATTGCTTGGTACACGGAACGTTCTGAAAAGA
GCTATGAATTGCAGACACCTTTTGAAATTAAATTGGCAAAGAAATT
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
```

• • •

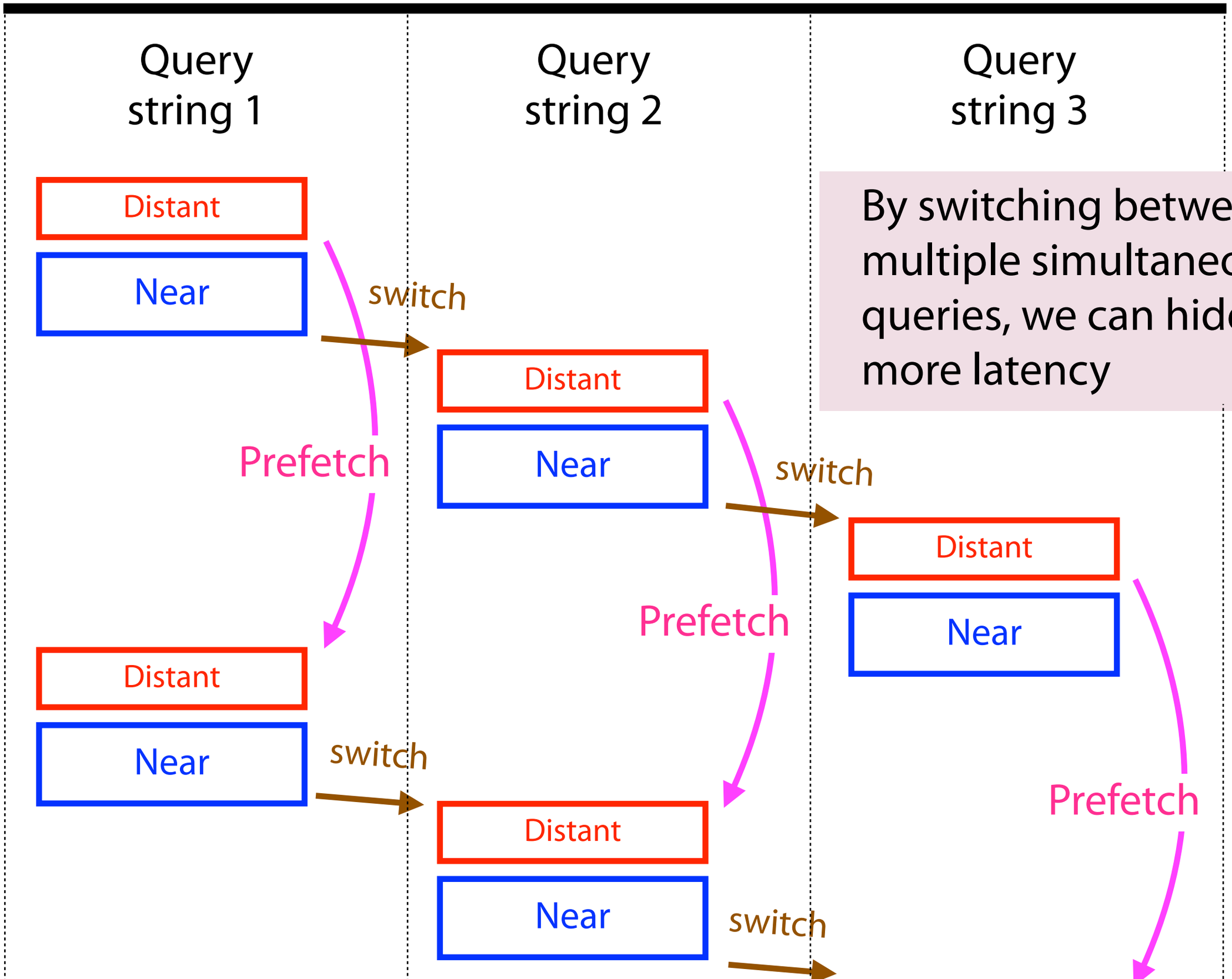
Idea: interleaving queries



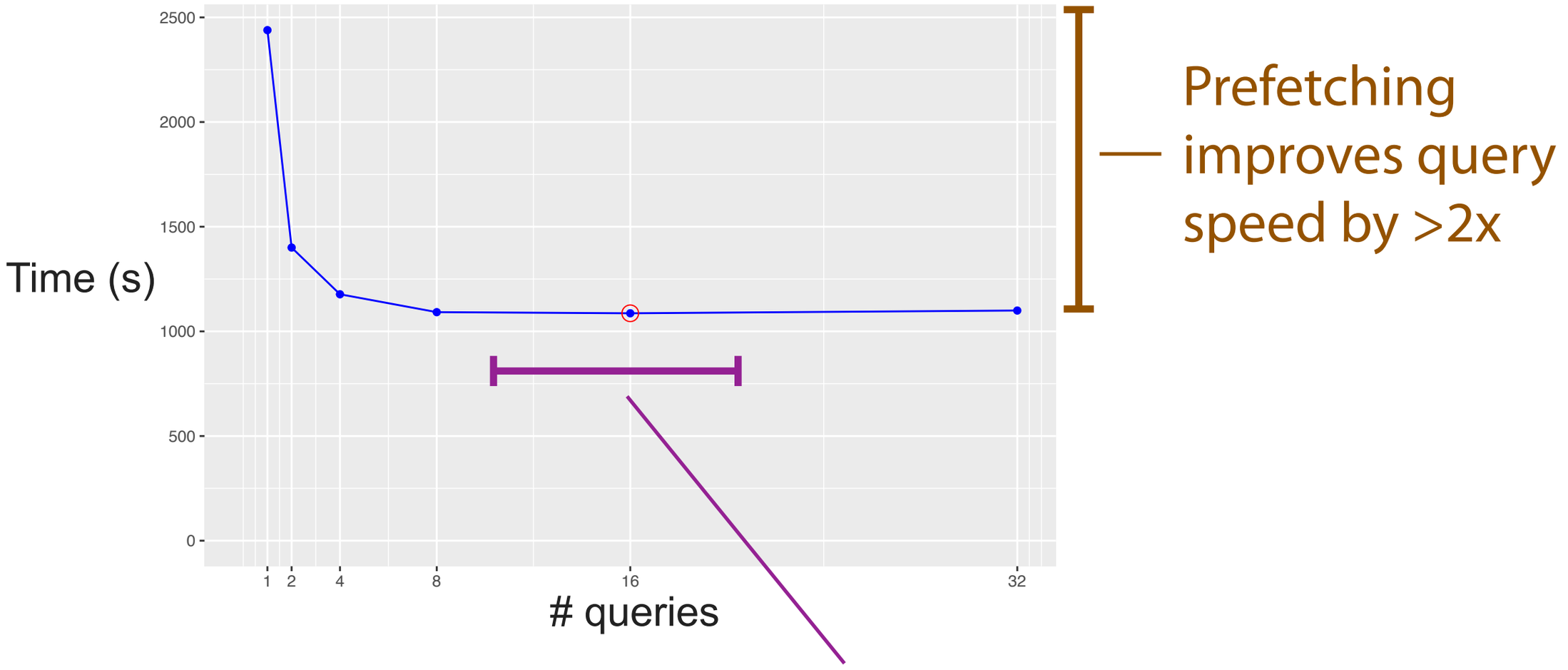
Idea: interleaving queries



Idea: interleaving queries



Speed improvement



Zakeri, M., Brown, N. K., Ahmed, O. Y., Gagie, T., & Langmead, B. (2024). Movi: a fast and cache-efficient full-text pangenome index. *iScience*, 27(12).