Move structure, part 3: Constant-time queries

Ben Langmead

For original Keynote files email me ben.langmead@gmail.com

Move structure: splitting

We can $O(1)$-bound the # of scans with "splitting"

My explanation follows that of Brown, Gagie & Rossi:

RLBWT Tricks

Nathaniel K. Brown
Faculty of Computer Science, Dalhousie University, NS, Canada

Travis Gagie
Faculty of Computer Science, Dalhousie University, NS, Canada

Massimiliano Rossi
Department of Computer and Information Science and Engineering, University of Florida, FL, USA

Abstract

Until recently, most experts would probably have agreed we cannot backwards-step in constant time with a run-length compressed Burrows-Wheeler Transform (RLBWT), since doing so relies on rank queries on sparse bitvectors and those inherit lower bounds from predecessor queries. At ICALP '21, however, Nishimoto and Tabei described a new, simple and constant-time implementation. For a permutation π, it stores an $O(r)$-space table — where r is the number of positions i where either $i = 0$ or $\pi(i+1) \neq \pi(i)+1$ — that enables the computation of successive values of $\pi(i)$ by table look-ups and linear scans. Nishimoto and Tabei showed how to increase the number of rows in the table to bound the length of the linear scans such that the query time for computing $\pi(i)$ is constant while maintaining $O(r)$-space.

Nathaniel K. Brown, Travis Gagie, and Massimiliano Rossi. RLBWT Tricks. In 20th International Symposium on Experimental Algorithms (SEA 2022). Leibniz International Proceedings in Informatics (LIPIcs), Volume 233, pp. 16:1-16:16

Move structure: splitting

We saw an example that needed 9 scans

T `row_row_roo__our_boatrou_oou_row__our_boa__ow_oow_ror_your_boat$`

L `twoarrruw___wwuwrooo___obbbr__rro_y_r_roruuuo__t__$aaooooooooooo_`

P `1111100111001011111001001100110101111111111001101101101000000001`

Q `1100100101111010111010011001111100000000100111011111111100111011`

Move structure: splitting

We saw an example that needed 9 scans

T `row_row_roo__our_boatrou_oou_row__our_boa__ow_oow_ror_your_boat$`

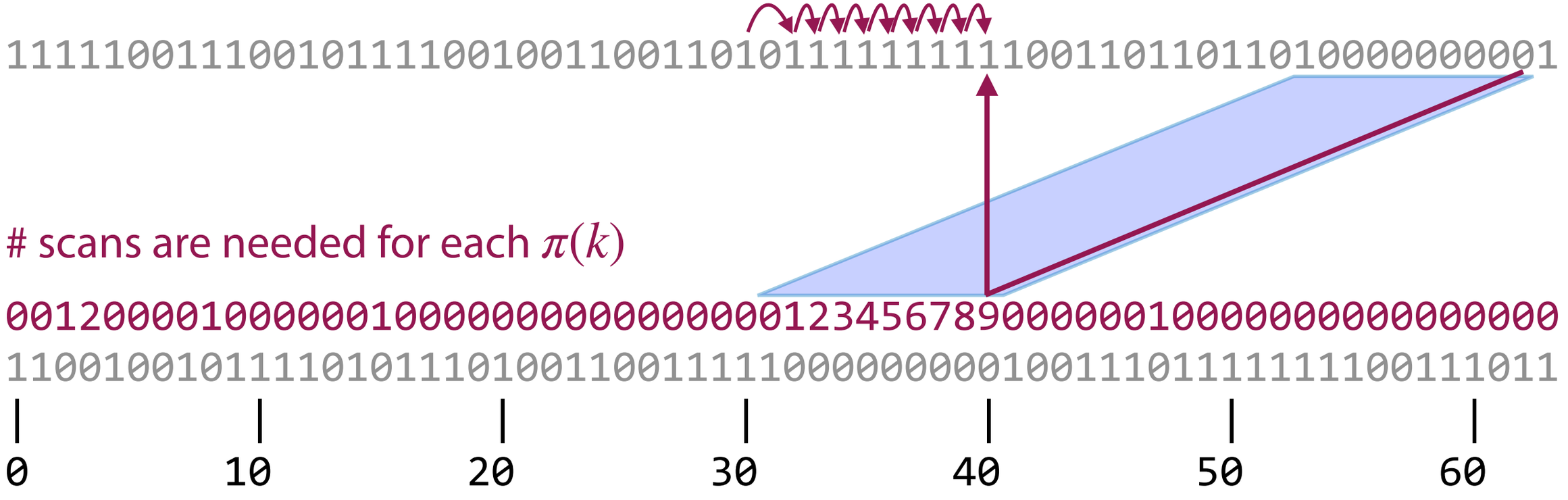
L `twoarrruw___wwuwrooo___obbbr__rro_y_r_roruuuo__t__$aaooooooooooo_`

P `1111100111001011111001001100110101111111111001101101101000000001`

scans are needed for each $\pi(k)$

`0012000010000001000000000000000012345678900000010000000000000000`

Q `1100100101111010111010011001111100000000100111011111111100111011`



Move structure: splitting

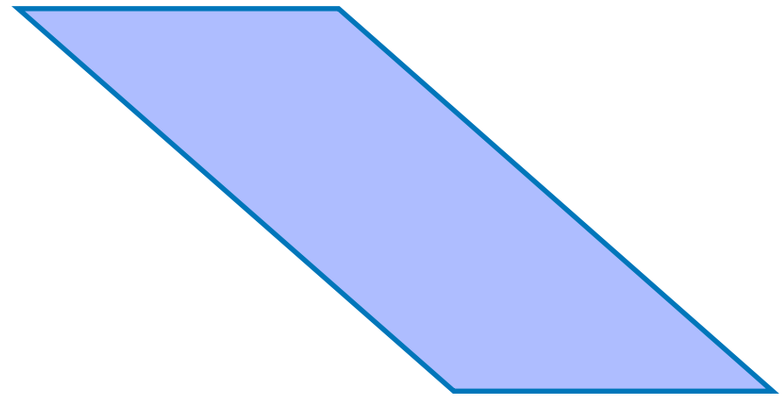
Idea: **split** runs until all Q-runs have weight $\leq$ a constant threshold

We'll use 4 as the threshold, then generalize later

Move structure: splitting

Splitting a run amounts to setting more bits in P & Q

P `1100100000000100100100100100000000100101100000100100000000000100`



Q `1100100000100100100100100000000000010010000000011010010000000100`

Move structure: splitting

Splitting a run amounts to setting more bits in P & Q

P 1100100000000100100100100100000000100101100000100100000000000100

0000000000000000000000000111111110000012222220000000000000000000

Q 1100100000100100100100100000000000100100000000110100100000000100

Move structure: splitting

Splitting a run amounts to setting more bits in P & Q

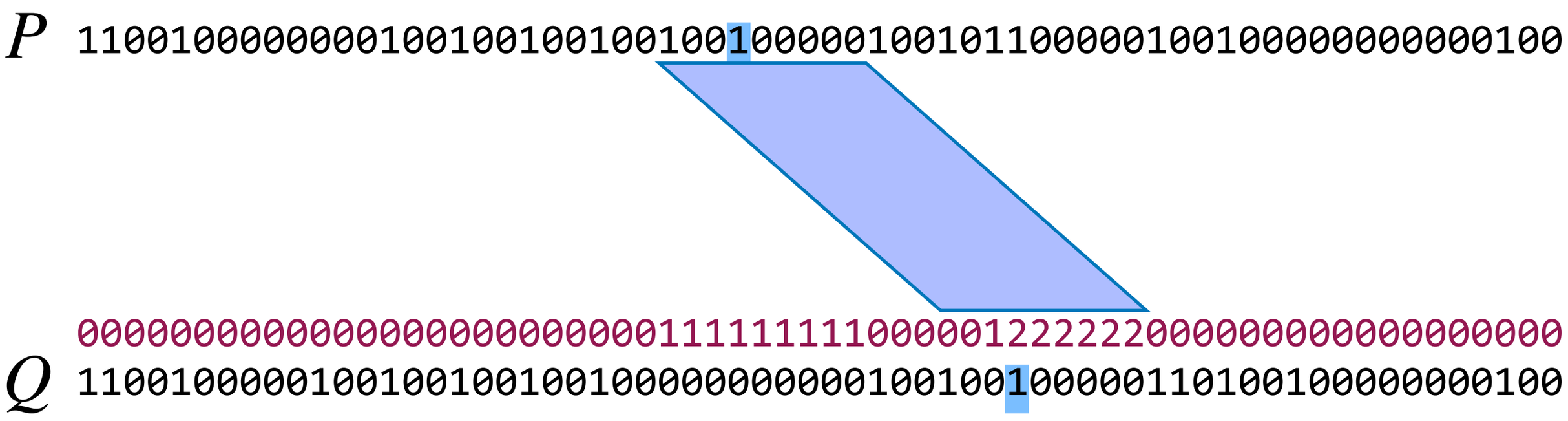
P 1100100000000100100100100100000000100101100000100100000000000100

0000000000000000000000000111111110000012222220000000000000000000

Q 1100100000100100100100100000000000100100100000011010010000000100

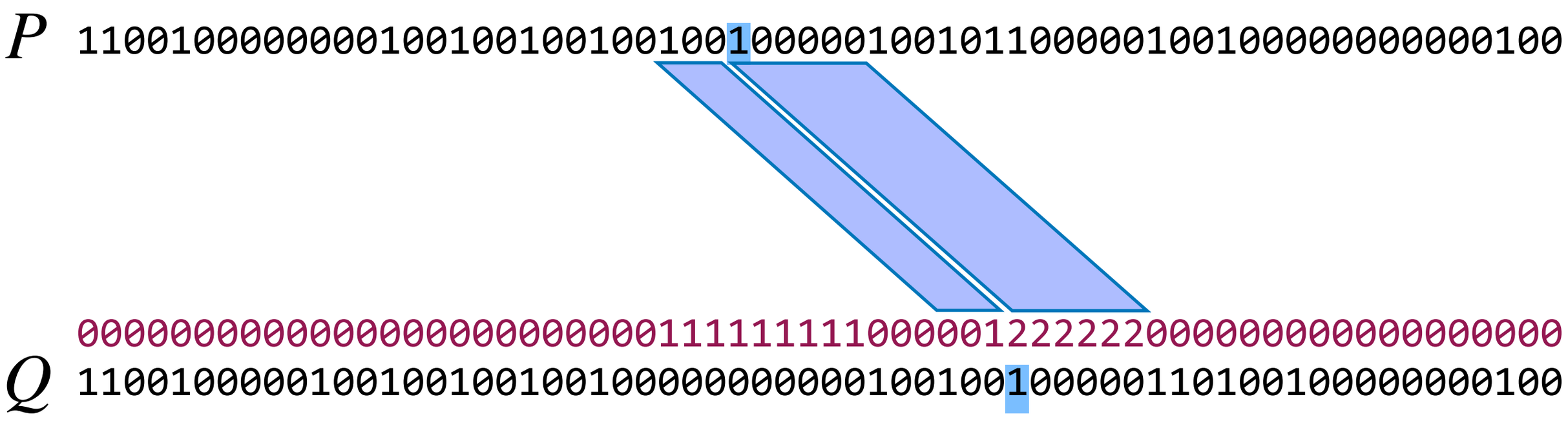
Move structure: splitting

Splitting a run amounts to setting more bits in P & Q



Move structure: splitting

Splitting a run amounts to setting more bits in P & Q



Move structure: splitting

Splitting a run amounts to setting more bits in P & Q

P 1100100000000100100100100100100000010010110000010010000000000100

Were 2s!

00000000000000000000000001111111100000100000000000000000000000

Q 1100100000100100100100100000000000100100100000110100100000000100

Price: the M table has one more row than before

Move structure: splitting

Until all Q-runs have weight < 4:

Pick a Q-run with weight ≥ 4

Set a bit in Q so this becomes 2 smaller runs, one of weight 2, another of weight ≥ 2

Move structure: splitting

Until all Q-runs have weight < 4:

Pick a Q-run with weight ≥ 4

Set a bit in Q so this becomes 2 smaller runs, one of weight 2, another of weight ≥ 2

E.g.:

P ...100101101100100 10...

Q ...100100000000000 10...

Move structure: splitting

Until all Q-runs have weight < 4:

Pick a Q-run with weight ≥ 4

Set a bit in Q so this becomes 2 smaller runs, one of weight 2, another of weight ≥ 2

E.g.:

P ...100101101100100 10...

Q ...100100100000000 10...

Move structure: splitting

Until all Q-runs have weight < 4:

Pick a Q-run with weight ≥ 4

Set a bit in Q so this becomes 2 smaller runs, one of weight 2, another of weight ≥ 2

Set corresponding bit of P. I.e., if we set bit k of Q, also set bit $\pi^{-1}(k)$ of P

Move structure: splitting

Until all Q-runs have weight < 4:

Pick a Q-run with weight ≥ 4

Set a bit in Q so this becomes 2 smaller runs, one of weight 2, another of weight ≥ 2

Set corresponding bit of P. I.e., if we set bit k of Q, also set bit $\pi^{-1}(k)$ of P

Is this guaranteed to finish? Is it guaranteed that the final # Q-runs will be O(initial # Q-runs)?

Move structure

P ...100101101100100 10...
Q ...100100100000000 10...

Until all Q-runs have weight < 4:

Pick a Q-run with weight ≥ 4

Set a bit in Q so this becomes 2 smaller runs, one of weight 2, another of weight ≥ 2

Also set $\pi^{-1}(k)$ in P, where k is the offset of the bit set in Q

Proof idea: each iteration

(a) increases # set bits in P by 1

(b) increases # of Q-runs with weight ≥ 2 by at least 1

Move structure: splitting

Each iteration

(a) increases # set bits in P by 1

(b) increases # of Q-runs with weight ≥ 2 by at least 1

If the initial # Q-runs is r, iterating our algorithm r times results in there being :

(a) $2r$ set bits in P

(b) $\geq r$ Q-runs with weight ≥ 2

Must stop here (or earlier); no Q-runs of weight 4 or greater can remain

Corollary: we at most double the number of set bits / M rows

Move structure: splitting

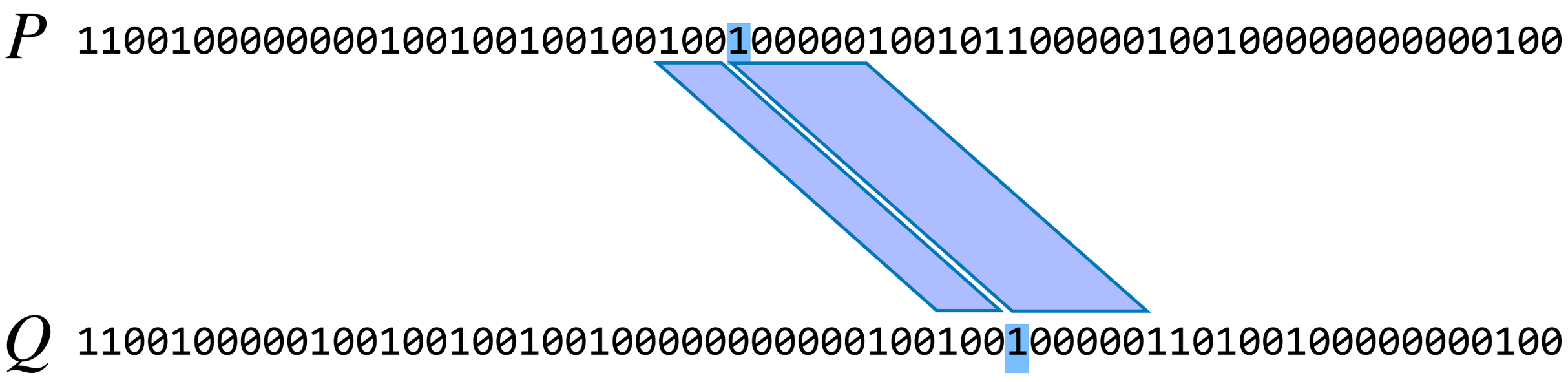
p	ℓ	π	ξ
0	1	46	12
1	3	34	8
4	9	52	13
13	3	19	5
16	3	1	1
19	3	16	4
22	3	61	14
25	9	37	9
34	3	49	13
37	2	47	12
39	1	0	0
40	6	4	2
46	3	13	3
49	12	22	2
61	3	10	6

p	ℓ	π	ξ
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	3	37	9
28	6	40	11
34	3	49	13
37	2	47	12
39	1	0	0
40	6	4	2
46	3	13	3
49	12	22	2
61	3	10	6

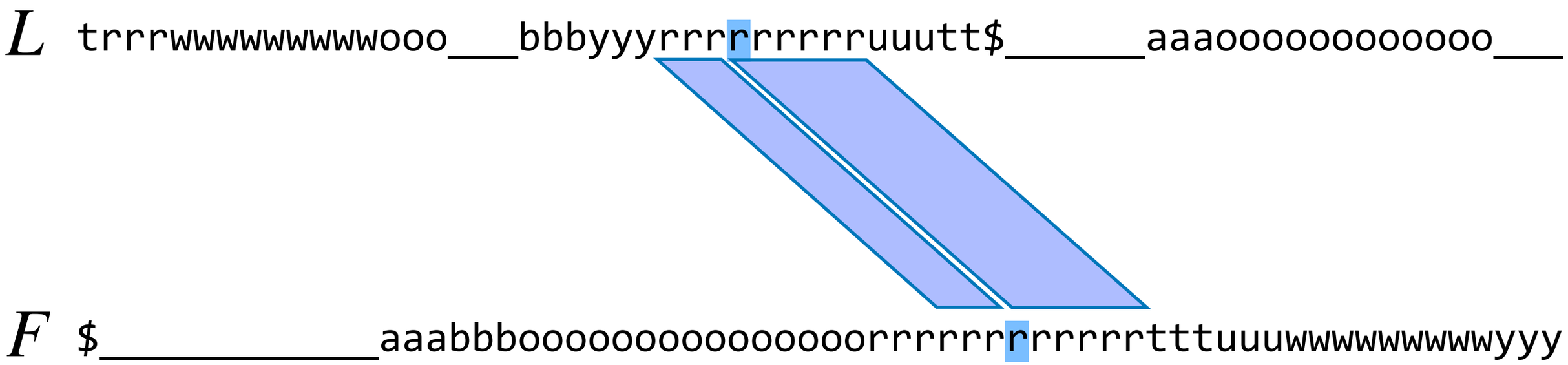
P 1100100000000100100100100100100000010010110000010010000000000100

Q 1100100000100100100100100000000000100100100000011010010000000100

Move structure: splitting



Move structure: splitting



Move structure

Argument generalizes from "weight < 4" to "weight $< 2d$" for a $d \geq 2$ we can pick

For some $d \geq 2$

j : # splits so far

x : # Q-runs of weight $\geq d$ so far

r : original # P-runs

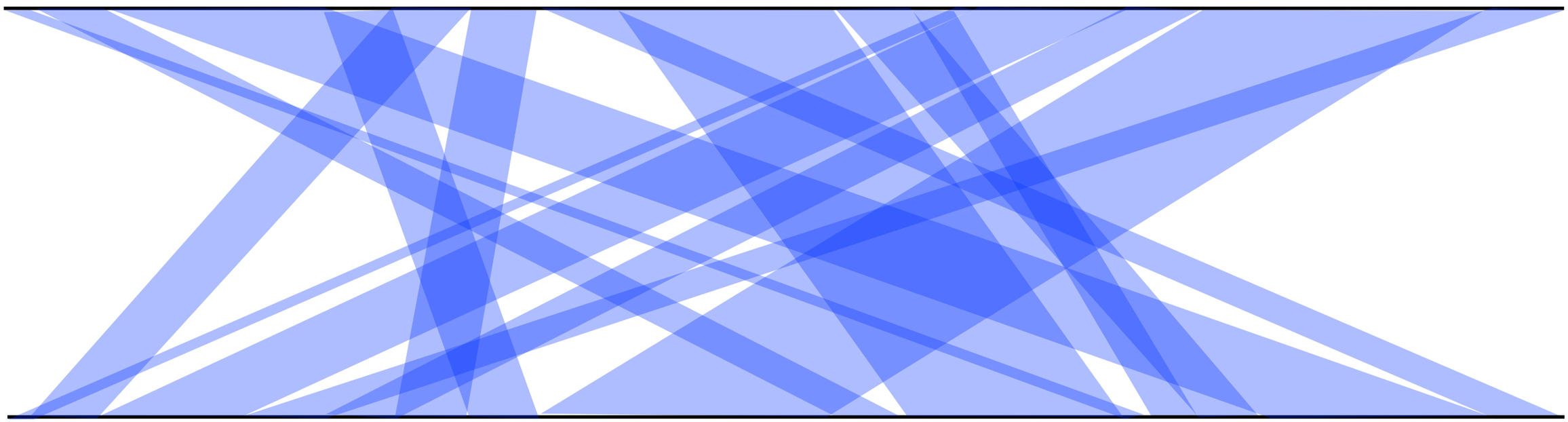
$$dj \leq dx \leq r + j \qquad r \geq dj - j = j(d-1)$$

$$j \leq \frac{r}{d-1}$$

iterations until all Q-runs have weight $< 2d$ can't exceed $\frac{r}{d-1}$

Nathaniel K. Brown, Travis Gagie, and Massimiliano Rossi. RLBWT Tricks. In 20th International Symposium on Experimental Algorithms (SEA 2022). Leibniz International Proceedings in Informatics (LIPIcs), Volume 233, pp. 16:1-16:16

Move structure



Works in $O(r)$ space & $O(1)$* query time

* For a query chain of any length $\pi^x(k)$, we need to begin with knowledge of both the initial k **and the initial** i

Proof v1

We start by setting $P_0 = P$ & $Q_0 = Q$

Say at some point we have P_i and $Q_i = \{\pi(i) : i \in P_i\}$

If there **do not** exist $q, q' \in Q_i$ such that q is the predecessor of q' in Q_i and $|[q, q') \cap P_i| \geq 2d$, then we return $P' = P_i$ and $Q' = Q_i$

Otherwise, we choose some such q and q'

Proof v1

We choose the $(d+1)$st largest element p in $[q, q') \cap P_i$.

We update $P_{i+1} = P_i \cup \{\pi^{-1}(p)\}$

and $Q_{i+1} = Q_i \cup p$

Since $q < p < q'$, we have $p \notin Q_i$ and $\pi^{-1}(p) \notin P_i$

Therefore $|P_{i+1}| = |P_i| + 1$ and, by induction,

$$|P_{i+1}| = P + i + 1$$

Proof v1

Let E_i be the set of intervals $[u, u')$ such that $u, u' \in Q_i$, u is the predecessor of u' in Q_i and $|\,[u, u') \cap P_i\,| \geq d$

Let E_{i+1} be the same for Q_{i+1}, P_{i+1}

Since $E_{i+1} = \left(E_i \setminus \{[q, q')\}\right) \cup \{[q, p), [p, q')\}$, we have

$|\,E_{i+1}\,| = |\,E_i\,| + 1$ and, by induction:

$|\,E_{i+1}\,| \geq i + 1$

Proof v1

Since intervals in E_{i+1} are disjoint and each contain at least d elements of P_{i+1}, we have

$$|P_{i+1}| \geq d \cdot |E_{i+1}| \geq d \cdot (i+1)$$

$$|P| + i + 1 = |P_{i+1}| \geq d \cdot |E_{i+1}| \geq d \cdot (i+1)$$

$$|P| + 1 - d \geq (d-1)i$$