

Научный семинар по информационным технологиям

Челябинск, ЮУрГУ

14.12.2018

# Параллельный алгоритм поиска похожих подпоследовательностей временного ряда для кластерных систем с узлами Intel Xeon Phi

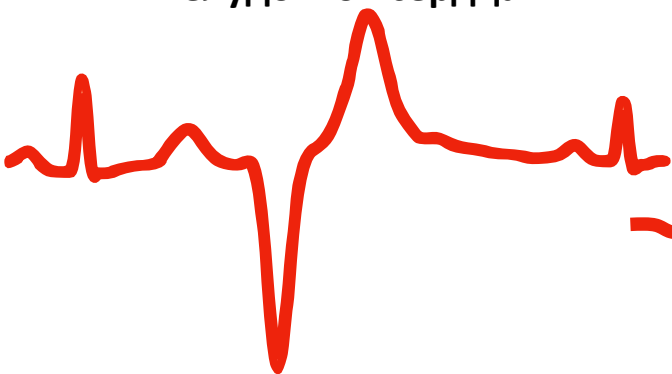
Я.А. Краева, М.Л. Цымблер

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 17-07-00463), Правительства РФ в соответствии с Постановлением № 211 от 16.03.2013 (соглашение № 02.A03.21.0011) и Министерства образования и науки РФ (государственное задание 2.7905.2017/8.9).

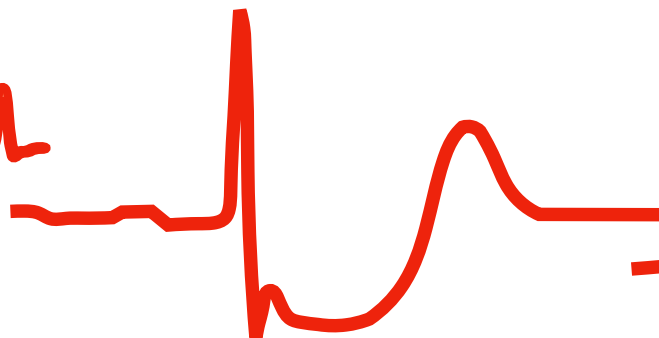
# Медицина: паттерны в ЭКГ



**Преждевременное сокращение  
желудочков сердца**



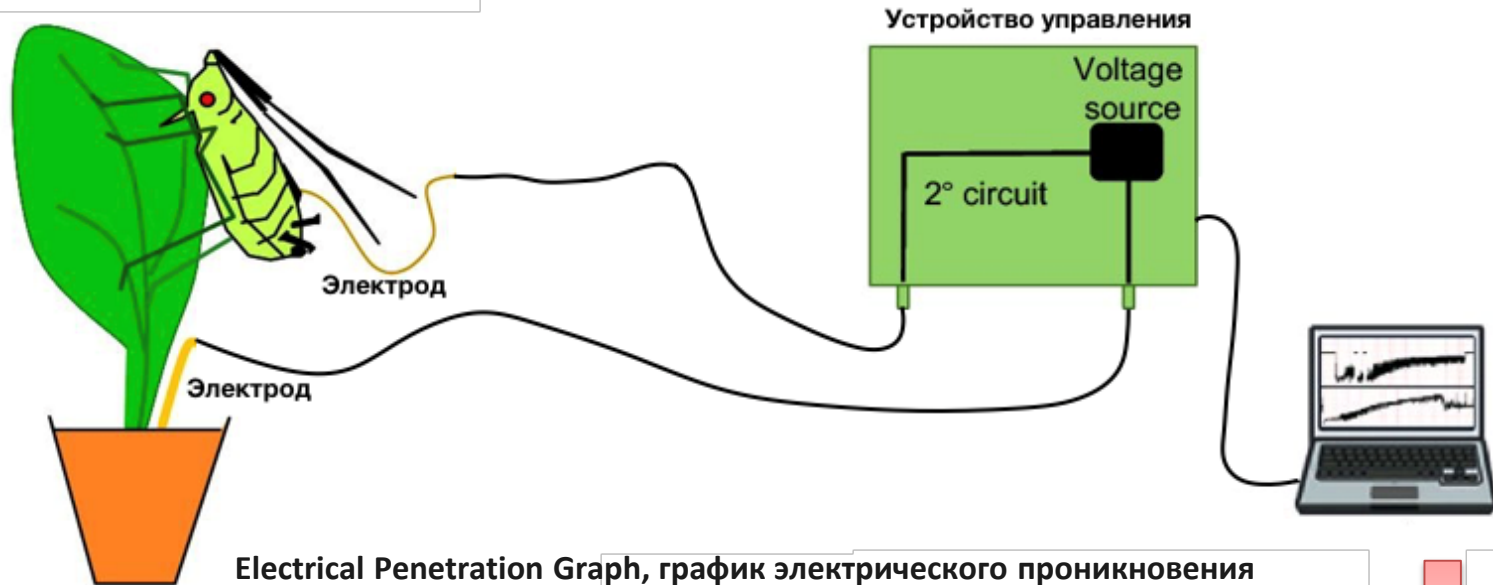
**Инфаркт**



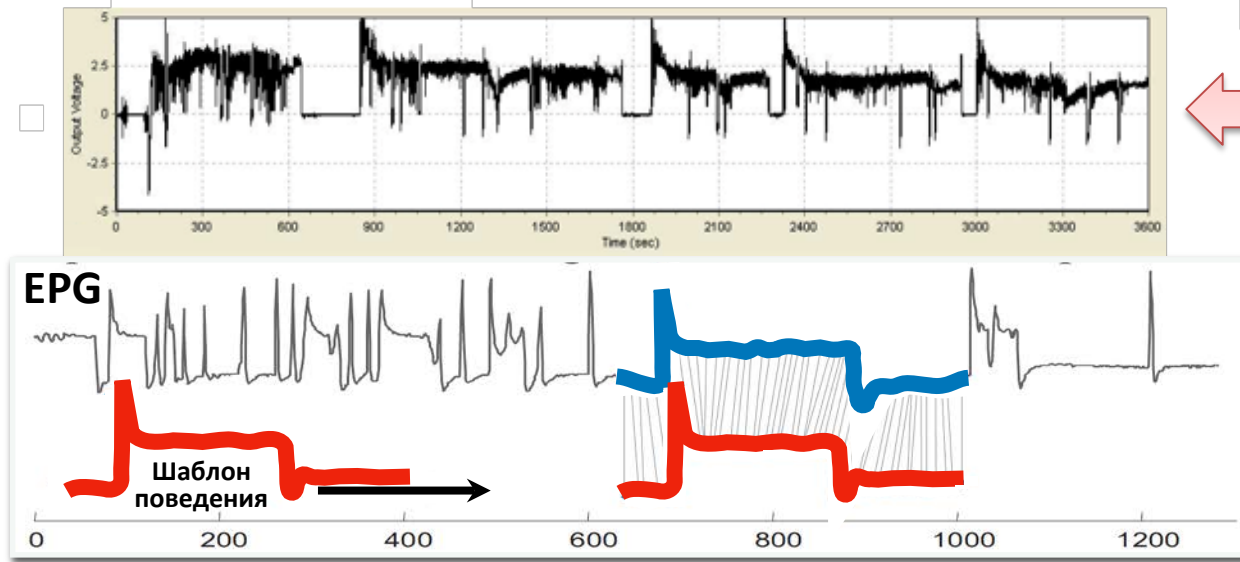
**Гиперкалиемия**



# Энтомология: поведение насекомых



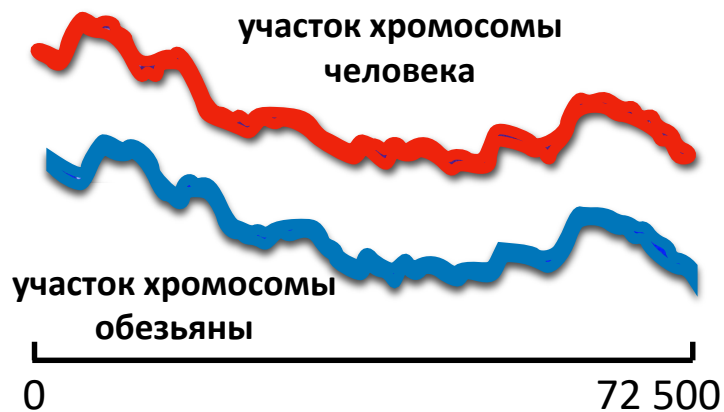
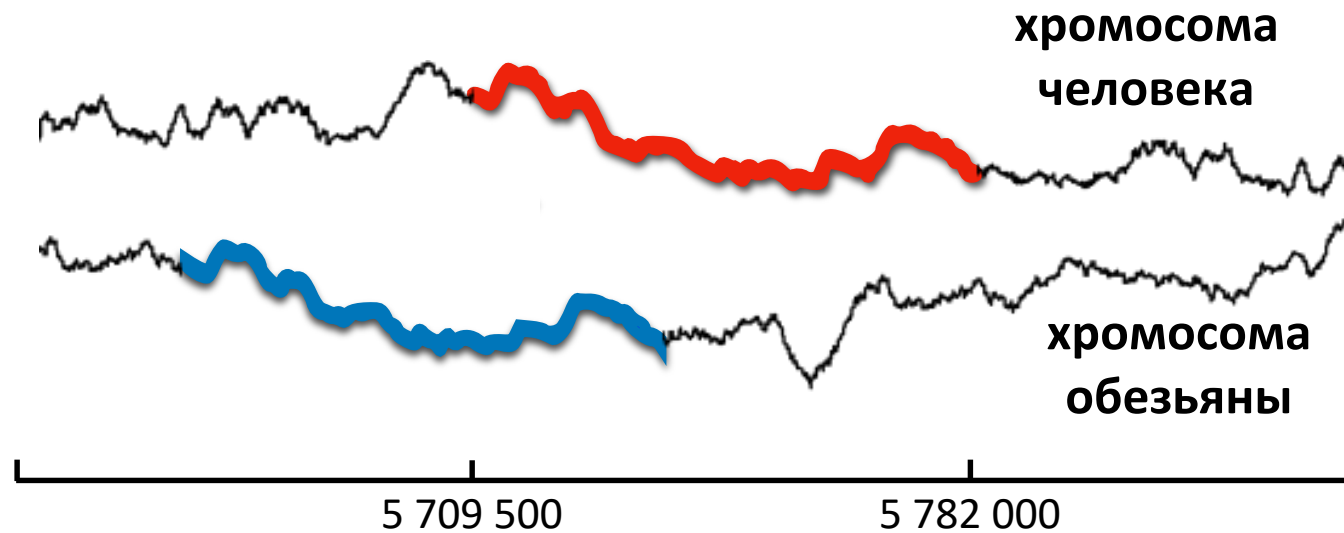
Electrical Penetration Graph, график электрического проникновения



# Генетика: геном приматов vs геном человека

Хромосома человека: **GTCAAT...AAGAGATTTG**

Хромосома обезьяны: **GGCAAT...ACAGATTTGA**



мартышка



гibbon



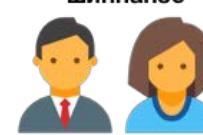
орангутанг



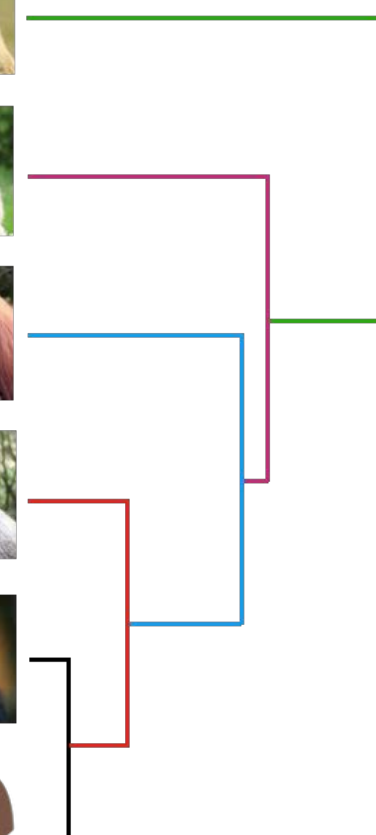
горилла



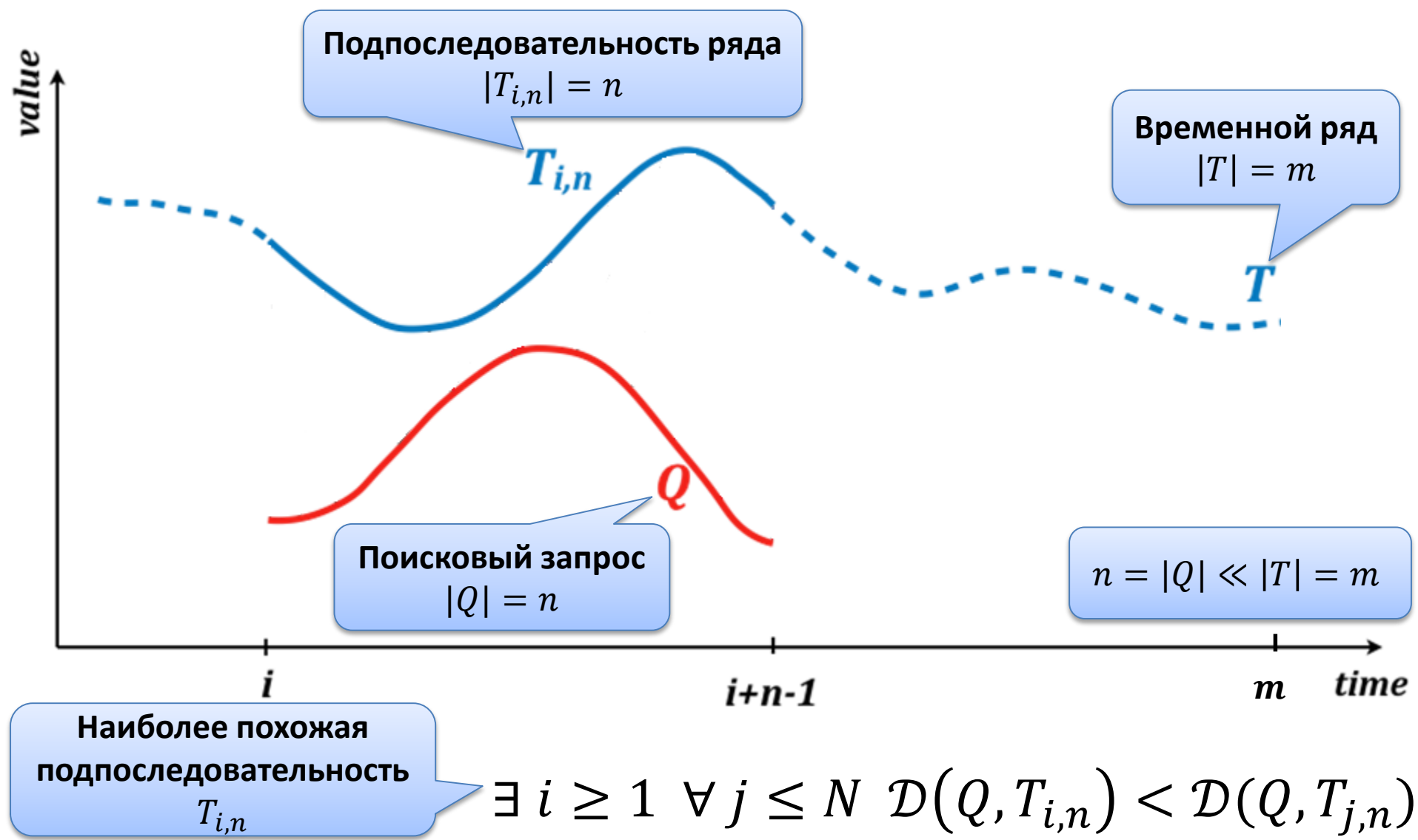
шинпанзе



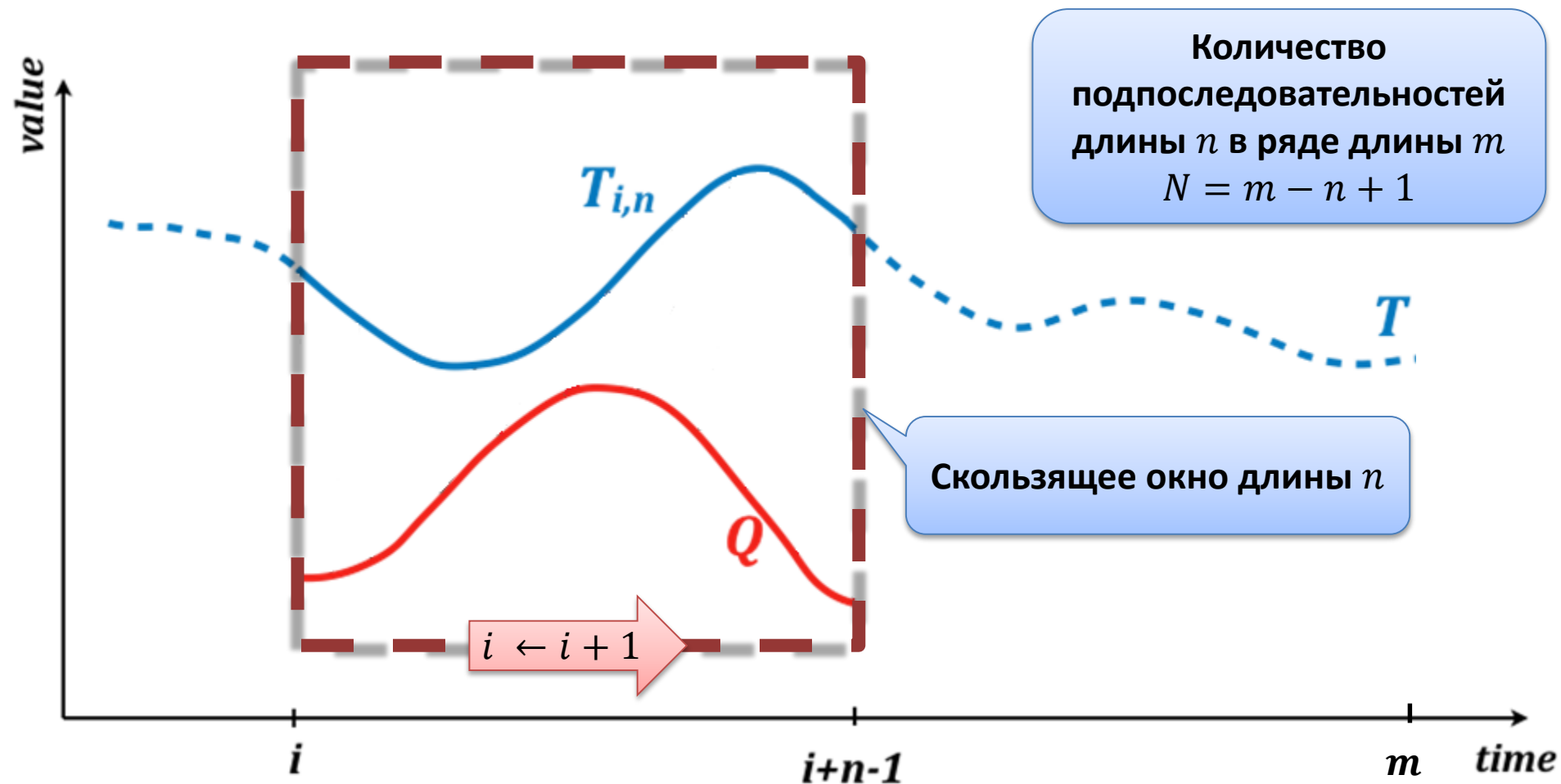
человек



# Поиск подпоследовательностей



# Поиск на базе скользящего окна



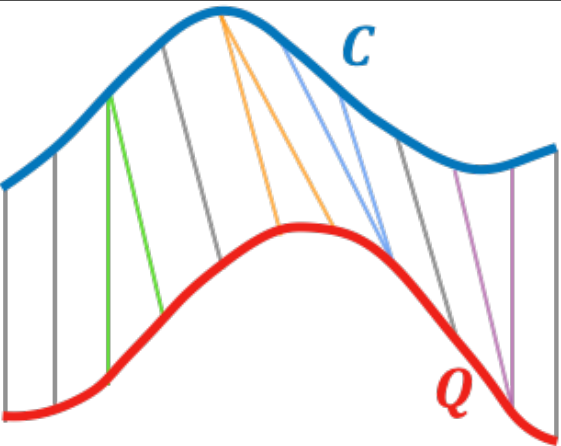
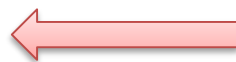
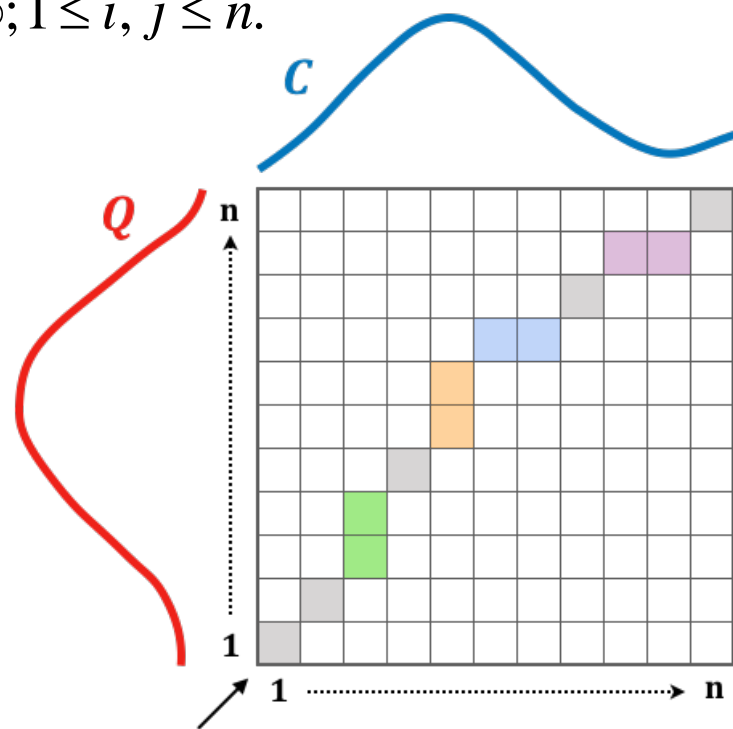
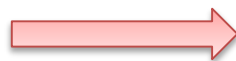
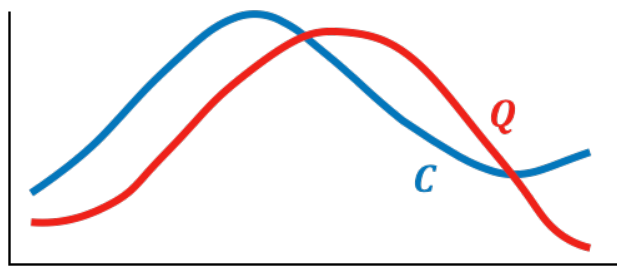
$$\exists i \geq 1 \forall j \leq N \quad \mathcal{D}(Q, T_{i,n}) < \mathcal{D}(Q, T_{j,n})$$

# Мера схожести DTW (Dynamic Time Warping)

$$DTW(Q, C) = d(n, n),$$

$$d(i, j) = (q_i - c_j)^2 + \min \begin{cases} d(i-1, j) \\ d(i, j-1) \\ d(i-1, j-1), \end{cases}$$

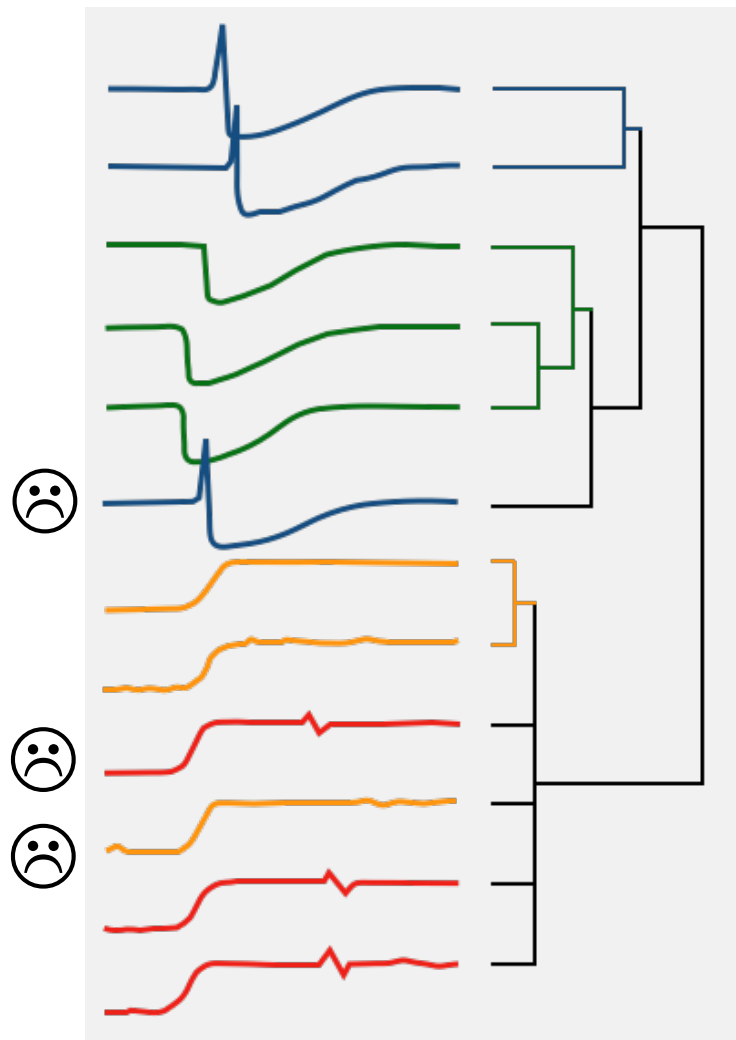
$$d(0,0) = 0; d(i,0) = d(0, j) = \infty; 1 \leq i, j \leq n.$$



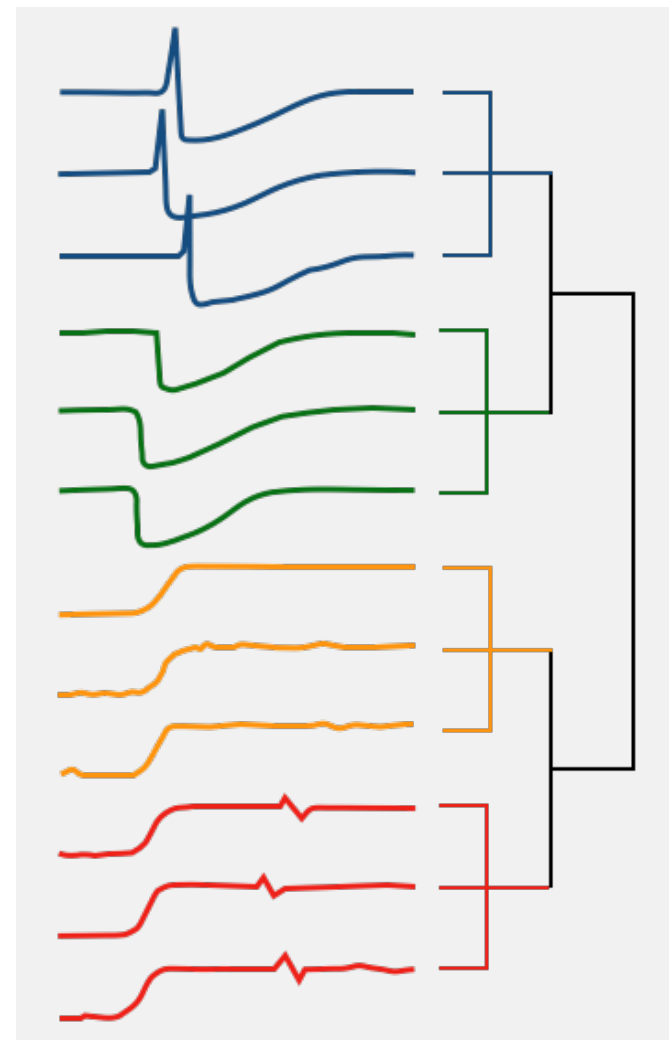
Путь трансформации

# Евклидово расстояние vs мера DTW

## Метрика Евклида



## Мера DTW



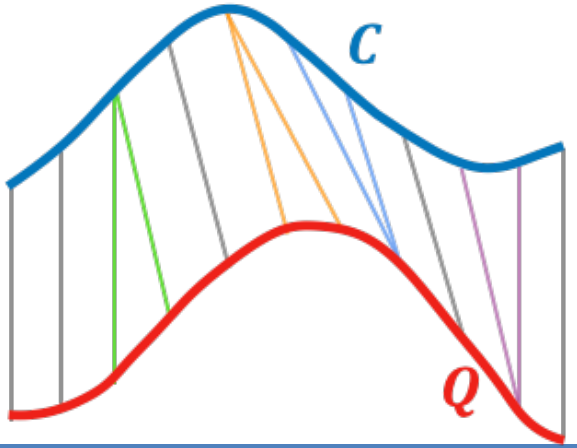
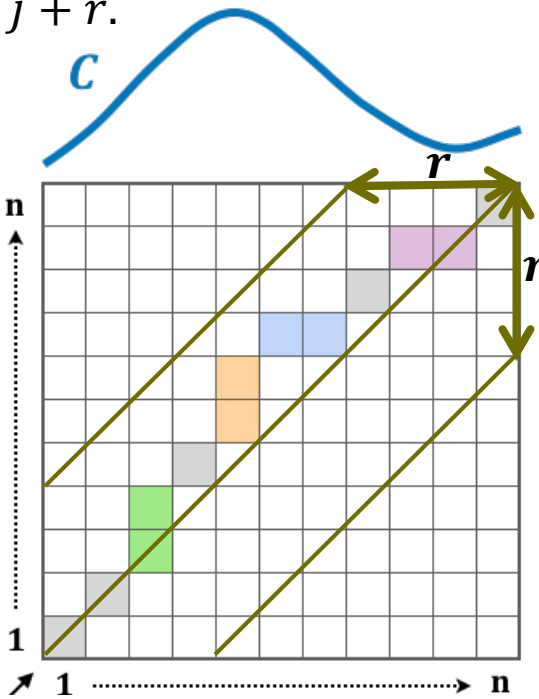
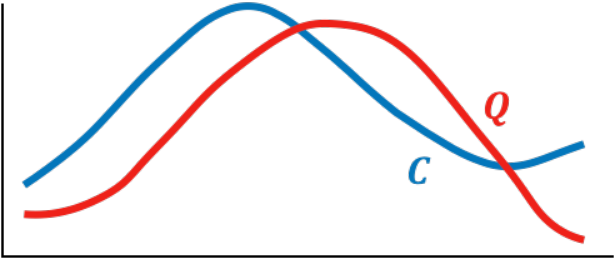


# Огрубление схожести vs сложность вычислений

$$DTW(Q, C) = d(n, n),$$

$$d(i, j) = (q_i - c_j)^2 + \min \begin{cases} d(i-1, j) \\ d(i, j-1) \\ d(i-1, j-1), \end{cases}$$

$$d(0,0) = 0; d(i,0) = d(0, j) = \infty; j - r \leq i \leq j + r.$$



**Полоса Сако-Чиба**

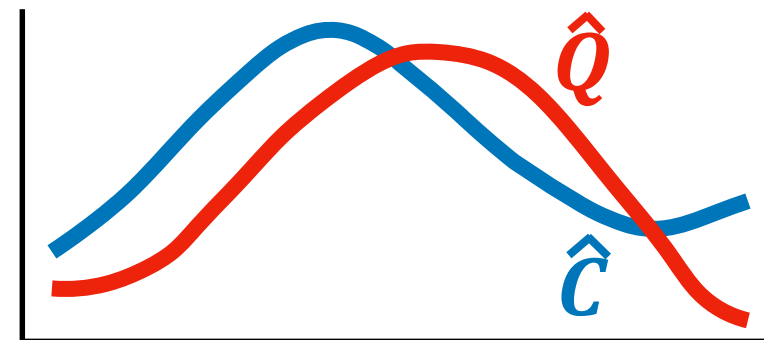
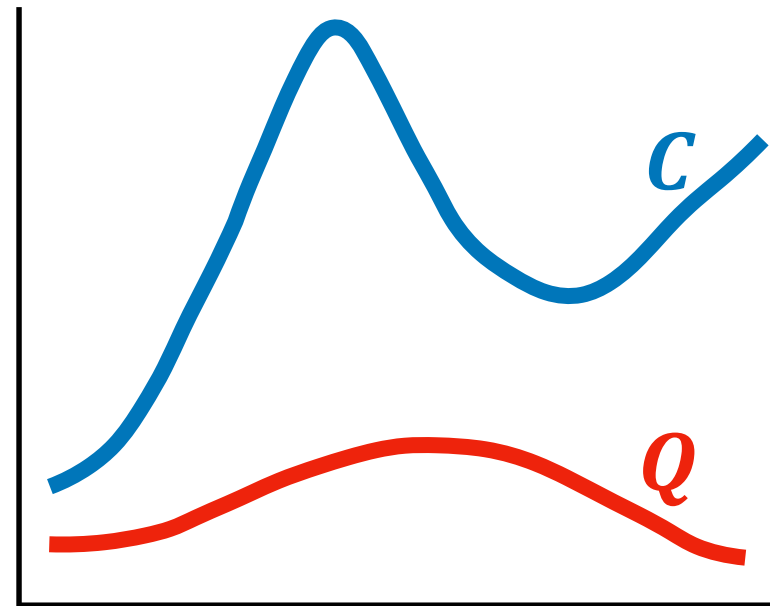
# Алгоритм UCR-DTW\*: принципы

- Z-нормализация подпоследовательностей и запроса
- Отбрасывание заведомо непохожих подпоследовательностей с помощью каскада нижних границ схожести

\* Rakthanmanon T., et al. Searching and mining trillions of time series subsequences under Dynamic Time Warping. ACM SIGKDD, 2012. pp. 262-270.

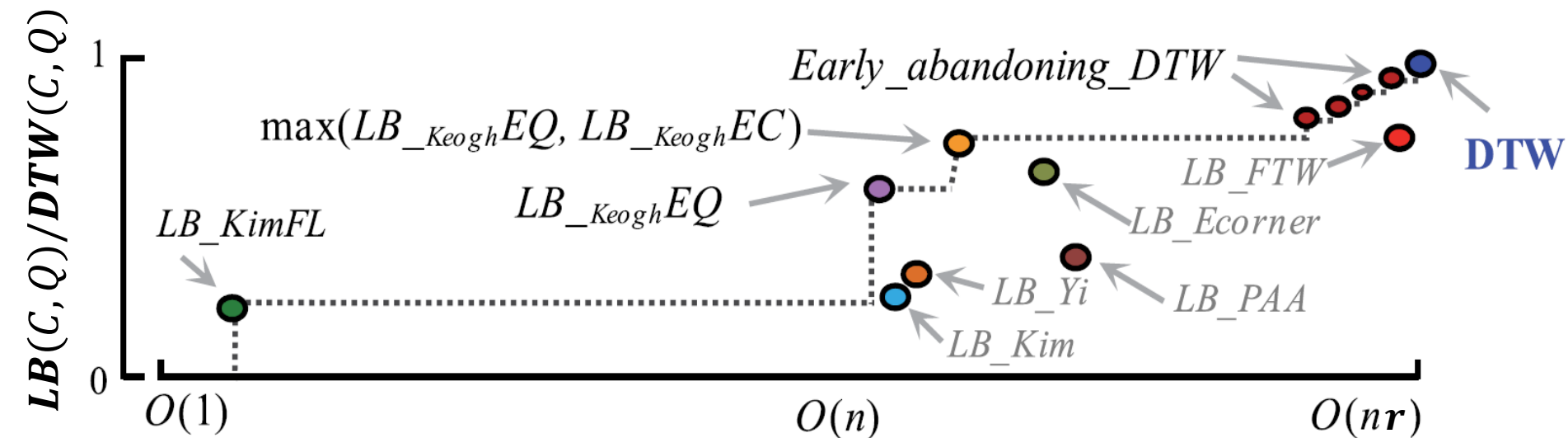
# Z-нормализация

- $\hat{C} = (\hat{t}_1, \hat{t}_2, \dots, \hat{t}_n)$
- $\hat{t}_i = \frac{t_i - \mu}{\sigma}$
- $\mu = \frac{1}{n} \sum_{i=1}^n t_i$
- $\sigma^2 = \frac{1}{n} \sum_{i=1}^n t_i^2 - \mu^2$
- Необходимо для сравнения рядов, которые отличны по амплитуде.  
После z-нормализации среднее арифметическое ряда приблизительно равно 0, среднее квадратичное отклонение близко к 1.

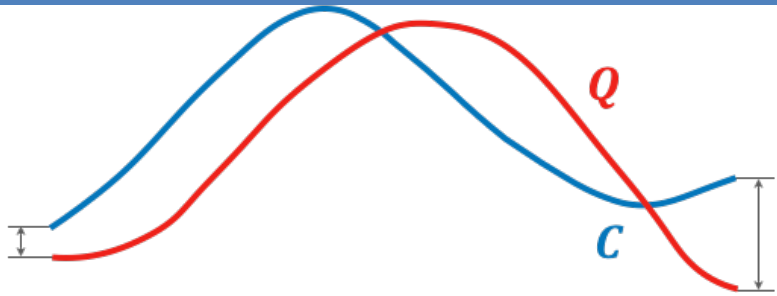


# Нижняя граница схожести (LB, lower bound)

- Функция, которая имеет вычислительную сложность меньше, чем DTW и задает пороговое значение (*best-so-far, bsf*) для DTW
- Если  $LB(T_{i,n}, Q) > bsf$ , то DTW можно не вычислять (подпоследовательность заведомо не похожа на запрос)
- $bsf_0 = \infty$ ,  $bsf_{next} = \min(DTW(Q, C), bsf_{prev})$

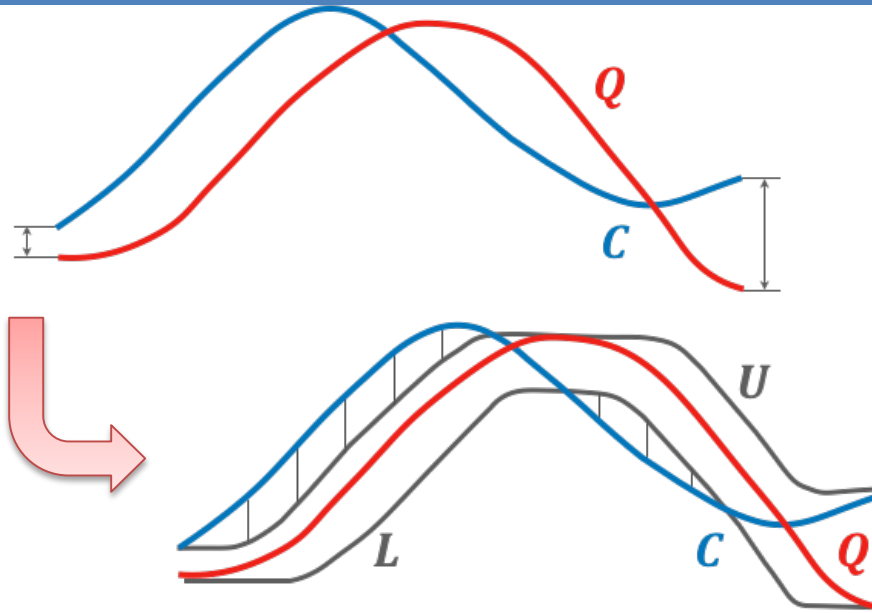


# Отбрасывание: нижняя граница $LB_{Kim}FL$



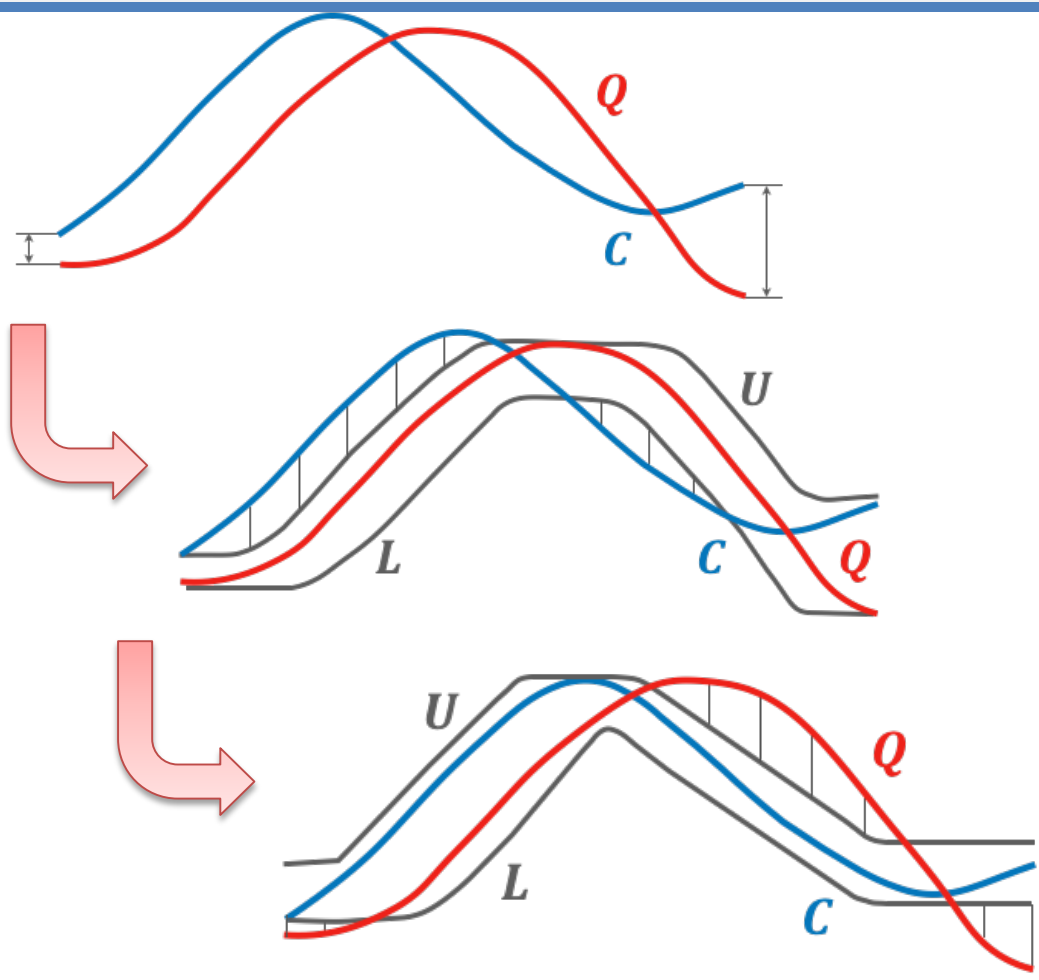
Нижняя граница	Сложность
$LB_{Kim}FL(Q, C) = (q_1 - c_1)^2 + (q_n - c_n)^2$	$O(1)$

# Отбрасывание: нижняя граница $LB_{Keogh}^{EC}$



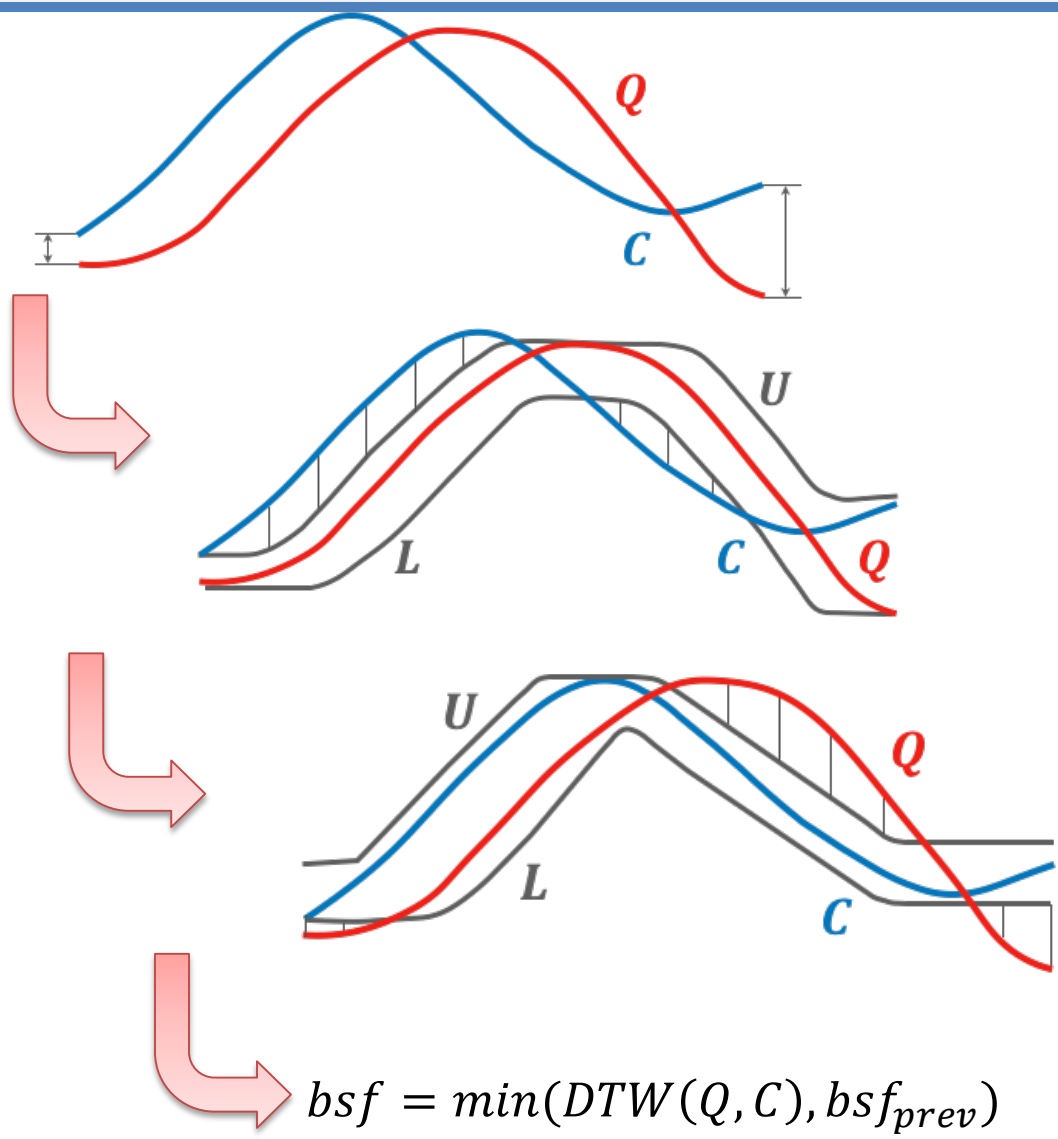
Нижняя граница	Сложность
$LB_{Kim}^{FL}(Q, C) = (q_1 - c_1)^2 + (q_n - c_n)^2$	$O(1)$
$LB_{Keogh}^{EC}(Q, C) = \sum_{i=1}^n \begin{cases} (c_i - u_i)^2, & c_i > u_i \\ (c_i - \ell_i)^2, & c_i < \ell_i \\ 0, & \text{otherwise} \end{cases}$ $u_i = \max_{i-r \leq k \leq i+r} q_k$ $\ell_i = \min_{i-r \leq k \leq i+r} q_k$	$O(n)$

# Отбрасывание: нижняя граница $LB_{KeoghEQ}$



Нижняя граница	Сложность
$LB_{KimFL}(Q, C) = (q_1 - c_1)^2 + (q_n - c_n)^2 O(1)$	$O(1)$
$LB_{KeoghEC}(Q, C) = \sum_{i=1}^n \begin{cases} (c_i - u_i)^2, & c_i > u_i \\ (c_i - \ell_i)^2, & c_i < \ell_i \\ 0, & otherwise \end{cases}$ $u_i = \max_{i-r \leq k \leq i+r} q_k$ $\ell_i = \min_{i-r \leq k \leq i+r} q_k$	$O(n)$
$LB_{KeoghEQ}(Q, C) = LB_{KeoghEC}(C, Q)$	$O(n)$

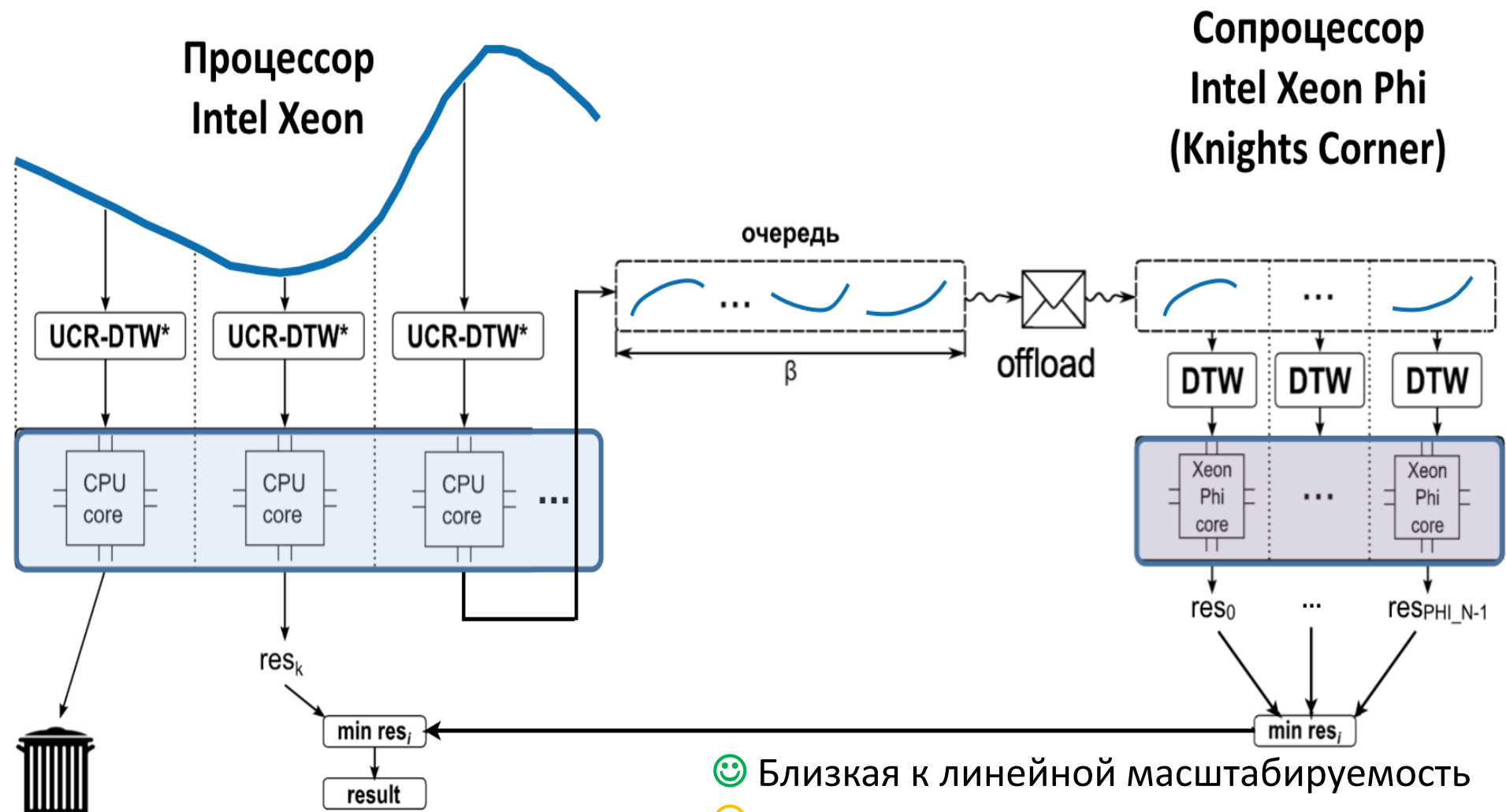
# Отбрасывание: DTW



Нижняя граница	Сложность
$LB_{Kim}FL(Q, C) = (q_1 - c_1)^2 + (q_n - c_n)^2$	$O(1)$
$LB_{Keogh}EC(Q, C) = \sum_{i=1}^n \begin{cases} (c_i - u_i)^2, & c_i > u_i \\ (c_i - \ell_i)^2, & c_i < \ell_i \\ 0, & otherwise \end{cases}$ $u_i = \max_{i-r \leq k \leq i+r} q_k$ $\ell_i = \min_{i-r \leq k \leq i+r} q_k$	$O(n)$
$LB_{Keogh}EQ(Q, C) = LB_{Keogh}EC(C, Q)$	$O(n)$
$DTW(Q, C)$	$O(nr)$



# Алгоритм для кластера на базе Phi KNC

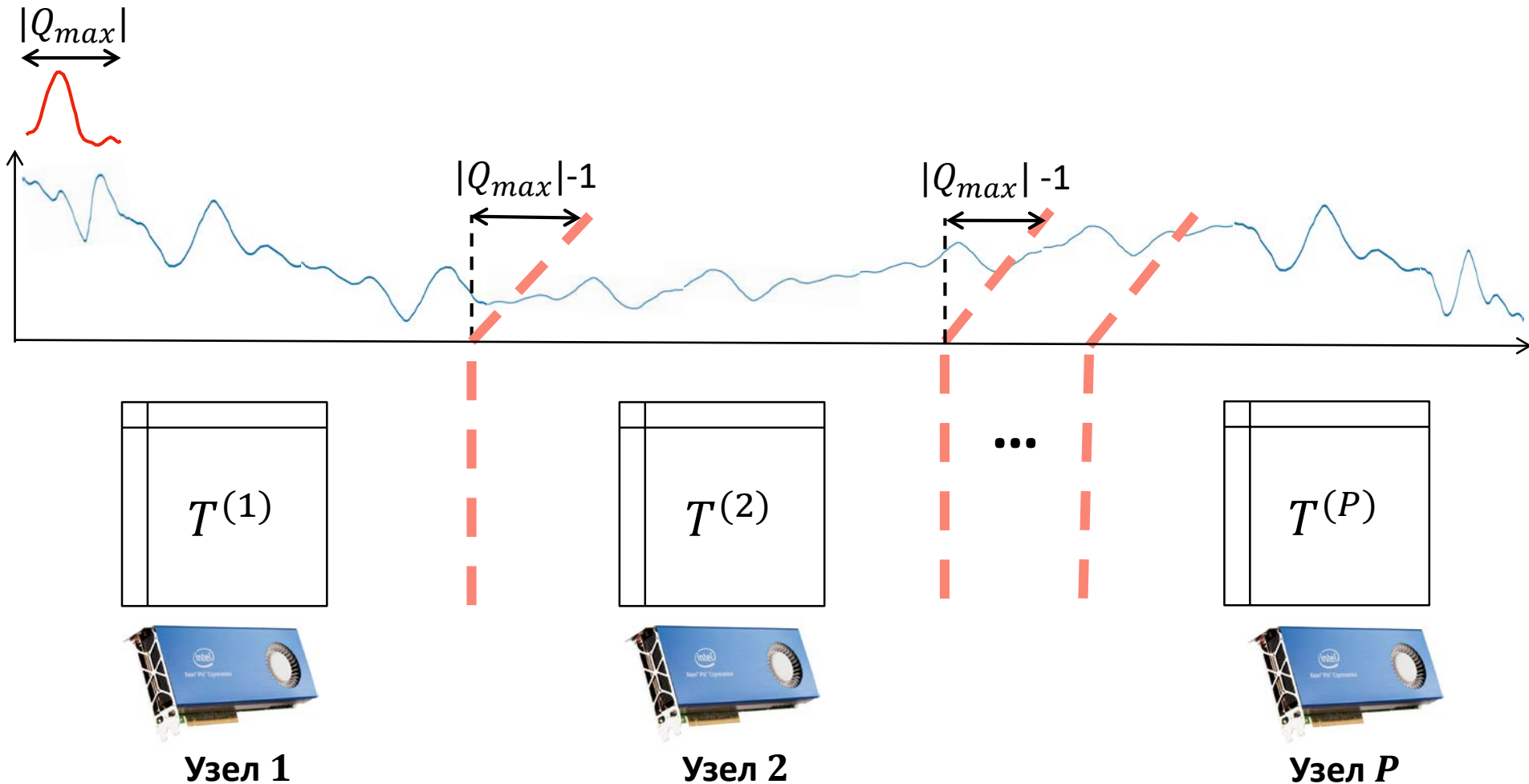


- 😊 Близкая к линейной масштабируемость
- 😄 при длине запроса в десятки тысяч точек
- 😞 **Схема не работает для Xeon Phi KNL**

# Алгоритм PhiBestMatch: принципы

- Аппаратная платформа
  - Вычислительный кластер с узлами Intel Xeon Phi (KNL)
- Параллелизм на уровне узлов кластера
  - Фрагментация временного ряда по узлам кластера
- Параллелизм на уровне одного узла
  - Избыточные вычисления и структуры данных для максимального использования возможностей векторной обработки Intel Xeon Phi
- Масштабирование
  - Каждый фрагмент размещается в оперативной памяти своего узла кластера, при необходимости в кластер добавляется еще один узел

# Фрагментация временного ряда



# Алгоритм PhiBestMatch: кластер

Узел 1

...

Узел  $P$

$myRank \leftarrow \text{MPI\_Comm\_rank}()$

$N \leftarrow |T^{(myRank)}| - n + 1$

$subseq_{rnd} \leftarrow T_{\text{random}(1..N), n}^{(myRank)}$

$bsf \leftarrow \text{DTW}(subseq_{rnd}, Q)$

$processed \leftarrow N$

**ПОДГОТОВИТЬ**

**repeat**

**УЛУЧШИТЬ** ( $T^{(myRank)}, bsf, bestmatch$ )

$fragDone \leftarrow (processed = 0)$

$bsf \leftarrow \text{MPI\_Allreduce}(bsf, \text{MPI\_MIN})$

$Stop \leftarrow \text{MPI\_Allreduce}(fragDone, \text{MPI\_AND})$

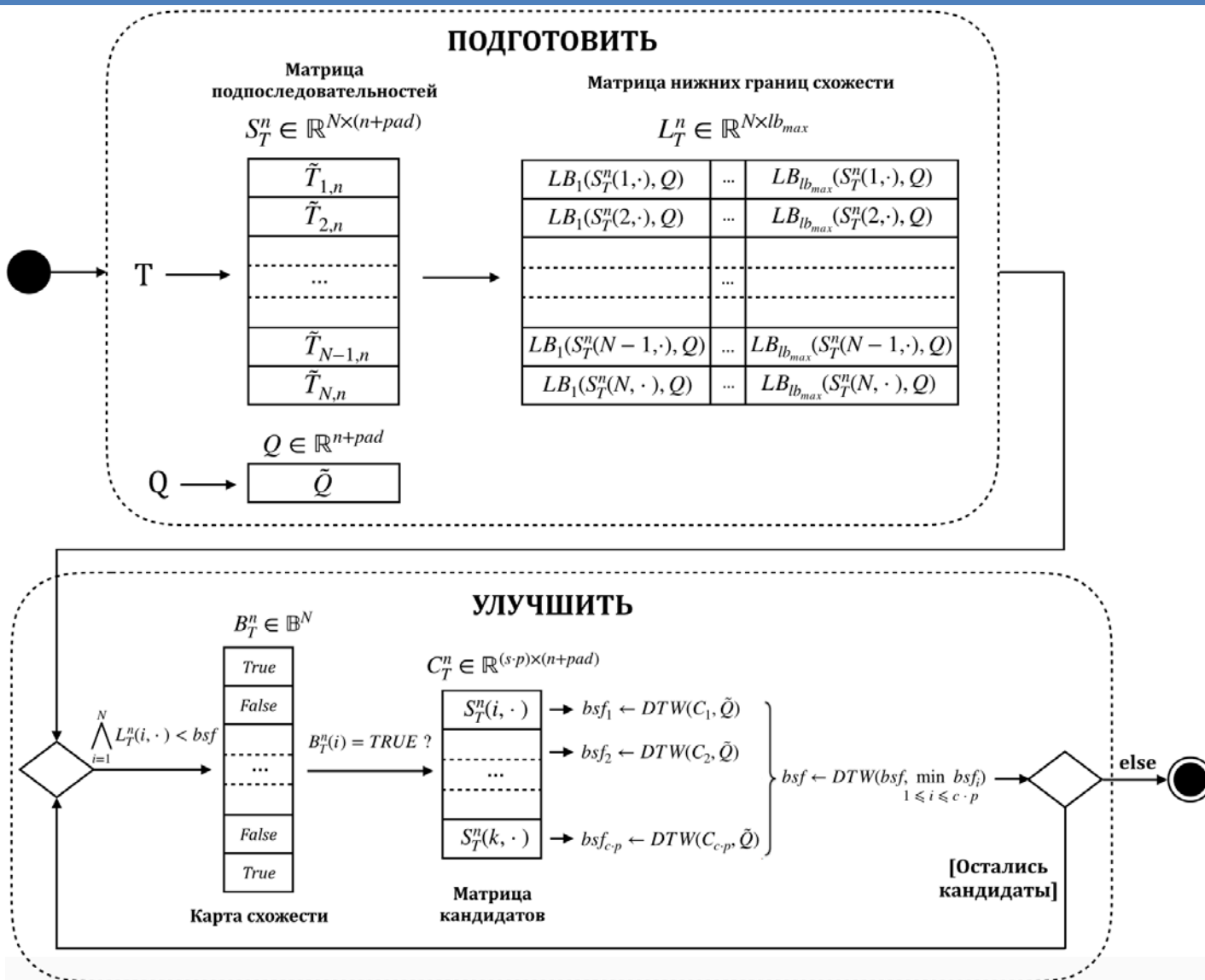
**until not Stop**

min

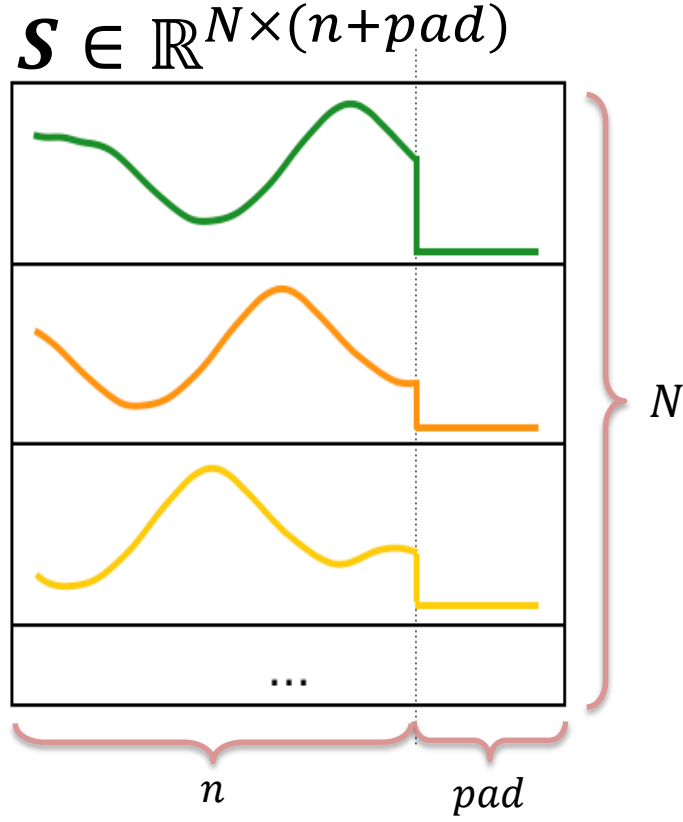
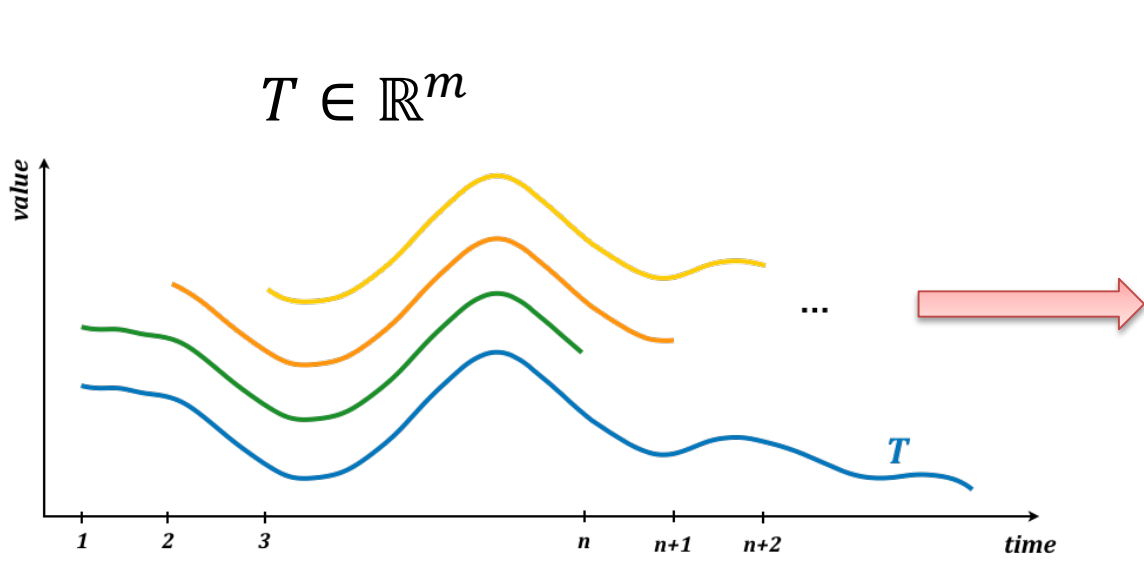
and



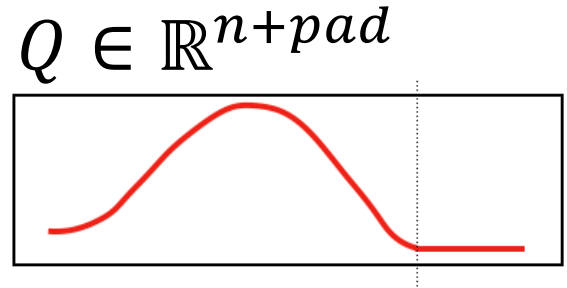
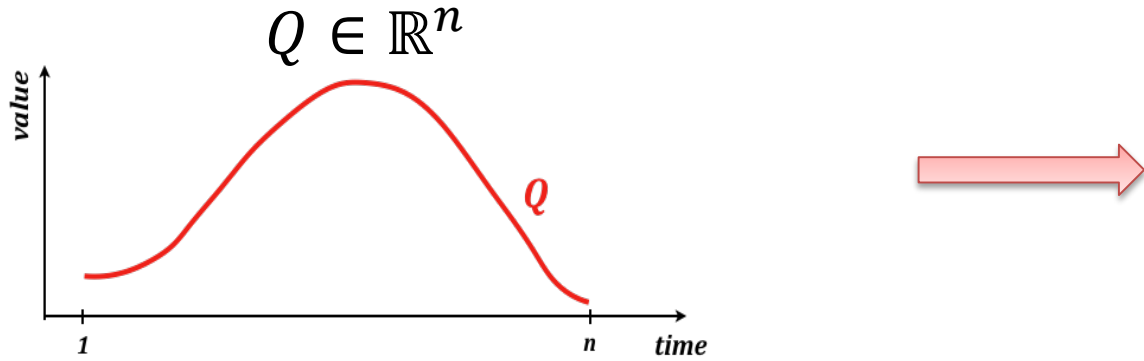
# Алгоритм PhiBestMatch: данные



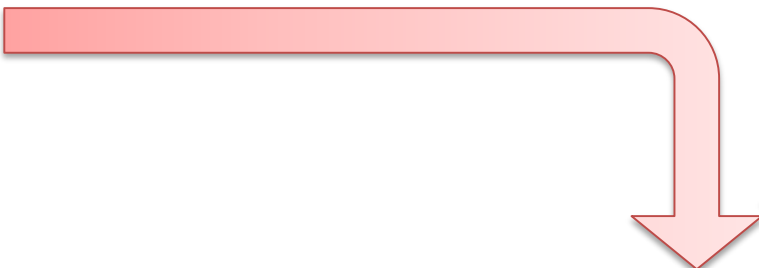
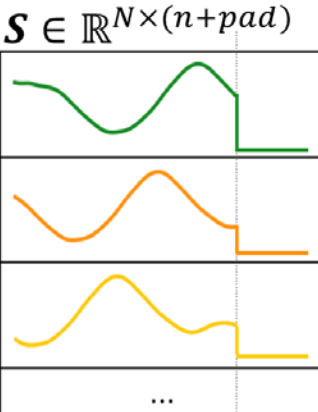
# Матрица подпоследовательностей



**Выравнивание:**  $(n + pad) : width_{VPU}$



# Матрица нижних границ



Распараллеливается 😊  
Авто-векторизуется 😊

$L \in \mathbb{R}^{N \times lb_{max}}$

$LB_{Kim}FL(Q, S_i)$	$LB_{Keogh}EC(Q, S_i)$	$LB_{Keogh}EQ(S_i, Q)$
$LB_{Kim}FL$ (red, green)	$LB_{Keogh}EC$ (red, green)	$LB_{Keogh}EQ$ (green, red)
$LB_{Kim}FL$ (red, orange)	$LB_{Keogh}EC$ (red, orange)	$LB_{Keogh}EQ$ (orange, red)
$LB_{Kim}FL$ (red, yellow)	$LB_{Keogh}EC$ (red, yellow)	$LB_{Keogh}EQ$ (yellow, red)
...	...	...

N

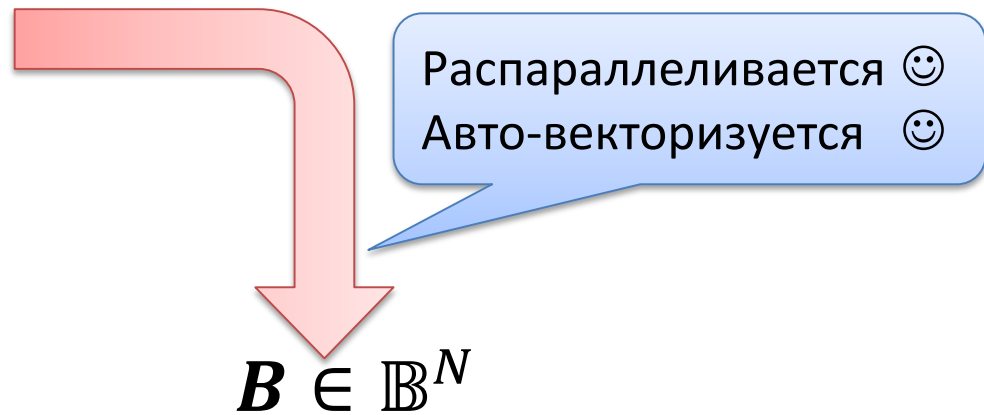
$lb_{max}$



# Карта схожести

$$L \in \mathbb{R}^{N \times lb_{max}}$$

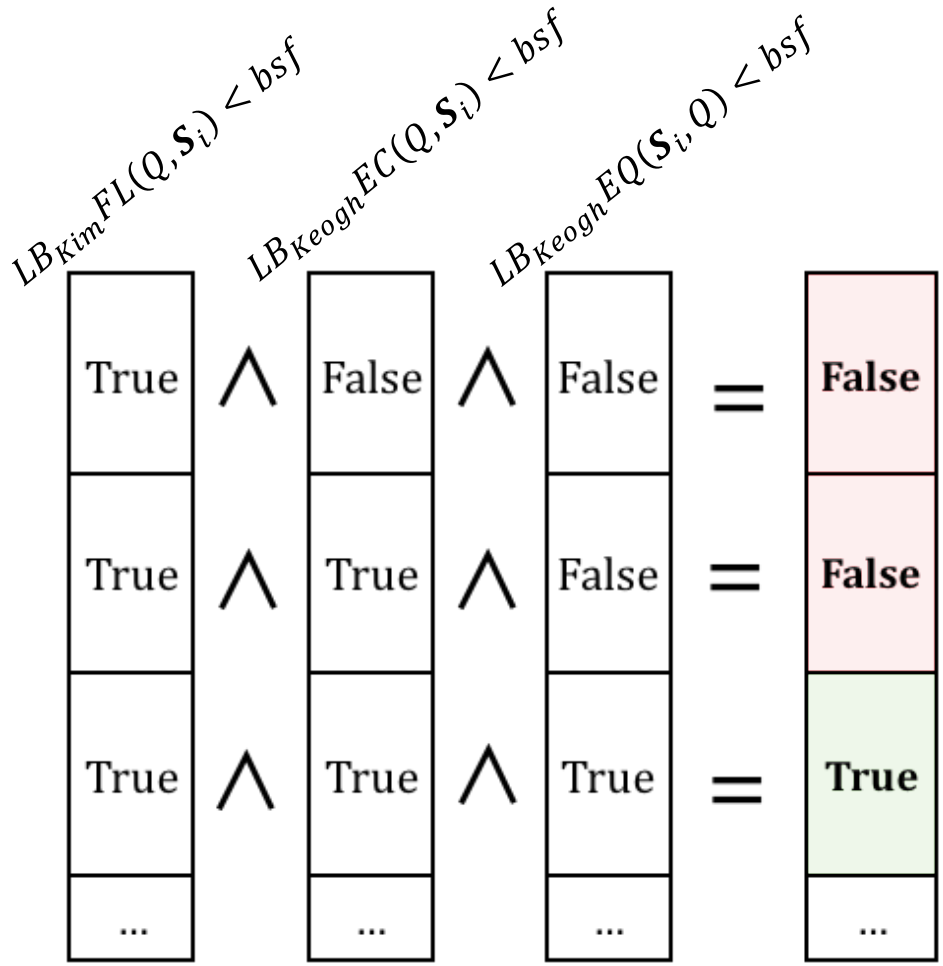
$LB_{Kim}FL(Q, S_i)$	$LB_{Keogh}EC(Q, S_i)$	$LB_{Keogh}EQ(S_i, Q)$
...	...	...



$$B \in \mathbb{B}^N$$

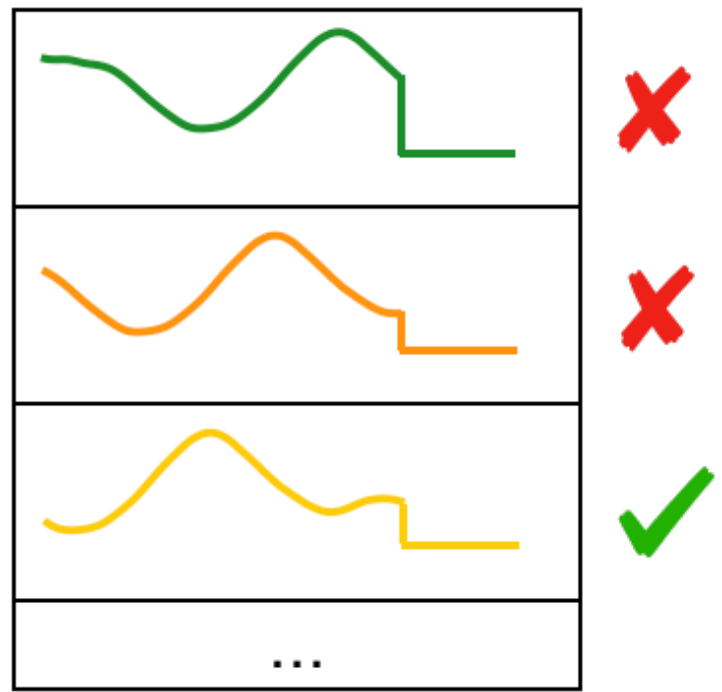
$\bigwedge_{j=1}^{lb_{max}} (L_T^n(i, j) < bsf)$
False
True
True
...

# Отбрасывание непохожих подпослед-тей



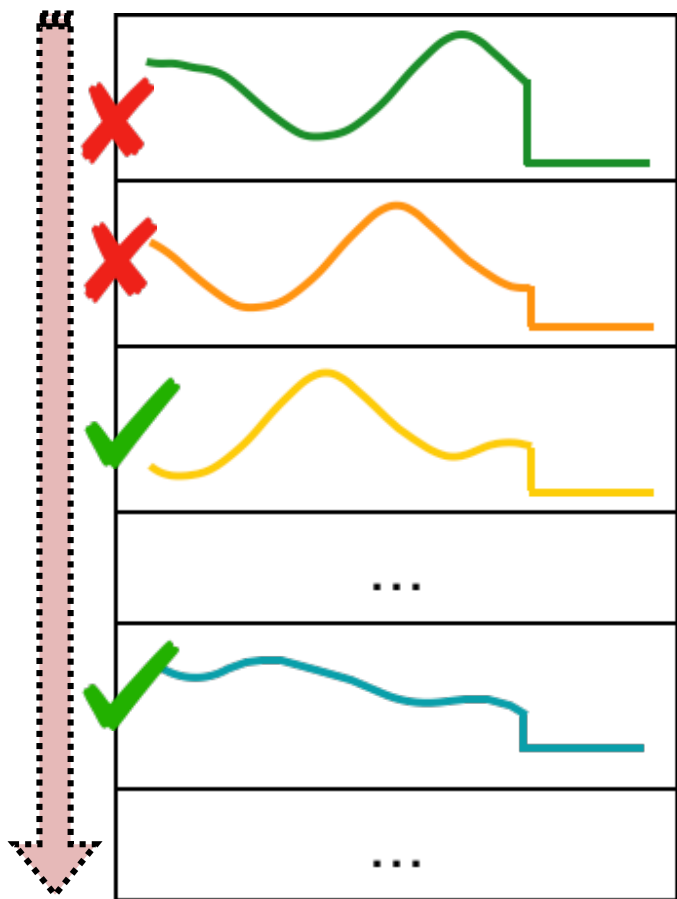
Распараллеливается 😊  
 Авто-векторизуется 😊

$S \in \mathbb{R}^{N \times (n+pad)}$



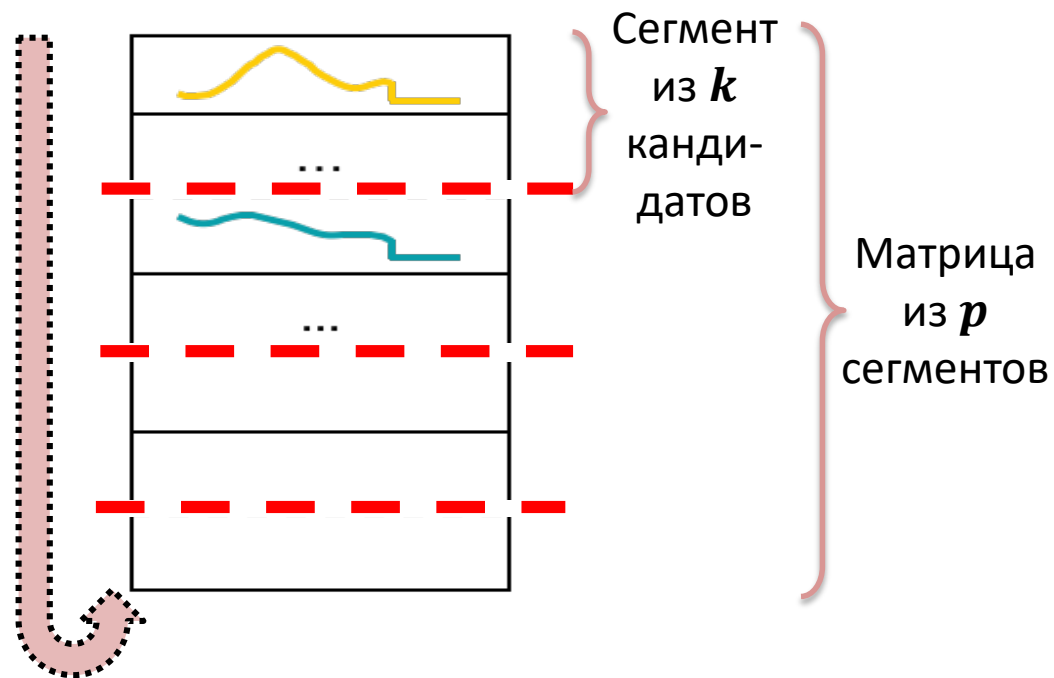
# Матрица кандидатов

$$S \in \mathbb{R}^{N \times (n+pad)}$$



Не распараллеливается ☹️

$$C \in \mathbb{R}^{(k \cdot p) \times (n+pad)}$$



**Балансировка загрузки нитей:**

$p \ll t$  – количество нитей

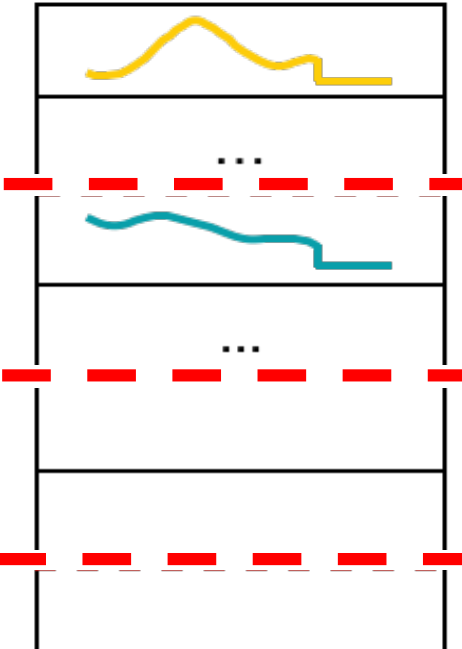
$k$  – параметр (10, 100, 1 000, ...)

# Вычисление best-so-far

$$C \in \mathbb{R}^{(k \cdot p) \times (n + pad)}$$

Распараллеливается ☺

Не авто-векторизуется ☹


$$\rightarrow bsf_1 \leftarrow DTW(C_1, Q)$$

$$\rightarrow bsf_2 \leftarrow DTW(C_2, Q)$$

$$\rightarrow bsf_{k \cdot p} \leftarrow DTW(C_{k \cdot p}, Q)$$

$$\rightarrow bsf \leftarrow \min(bsf_{prev}, \min_{1 \leq i \leq k \cdot p} bsf_i)$$

Частично авто-векторизуется ☺

# Вычисление меры DTW

```
double DTW(a: array [1..m], b: array [1..m], r: int) {  
    cost := array [1..m]  
    cost_prev := array [1..m]  
  
    for i := 1 to m  
        cost[i] = infinity  
        cost_prev[i] = infinity  
  
    cost_prev[1] = dist(a[1], b[1])  
  
    for j := max(2, i-r) to min(m, i+r)  
        cost_prev[j] := cost_prev[j-1] + dist(a[1], b[j])  
  
    for i := 2 to m  
        for j := max(1, i-r) to min(m, i+r)  
            c := d(a[i], b[j])  
            cost[j] := c + min(cost[j-1], cost_prev[j-1], cost_prev[j])  
            swap(cost, cost_prev)  
  
    return cost_prev[m]  
}
```

Не авто-векторизуется, ☹️  
но может быть переписано  
с авто-векторизуемой частью 😊

# Вычисление меры DTW

```
double DTW(a: array [1..m], b: array [1..m], r: int) {
  cost := array [1..m]
  cost_prev := array [1..m]
  for i := 1 to m
    cost[i] = infinity
    cost_prev[i] = infinity
  cost_prev[1] = dist(a[1], b[1])
  for j := max(2, i-r) to min(m, i+r)
    cost_prev[j] := cost_prev[j-1] + dist(a[1], b[j])
  for i := 2 to m
    for j := max(1, i-r) to min(m, i+r)
      cost[j] = min(cost_prev[j-1], cost_prev[j])
    for j := max(1, i-r) to min(m, i+r)
      c := dist(a[i], b[j])
      cost[j] := c + min(cost[j-1], cost[j])
    swap(cost, cost_prev)
  return cost_prev[m]
}
```

Авто-векторизуемая часть 😊

Не авто-векторизуемая часть 😞

# Эксперименты

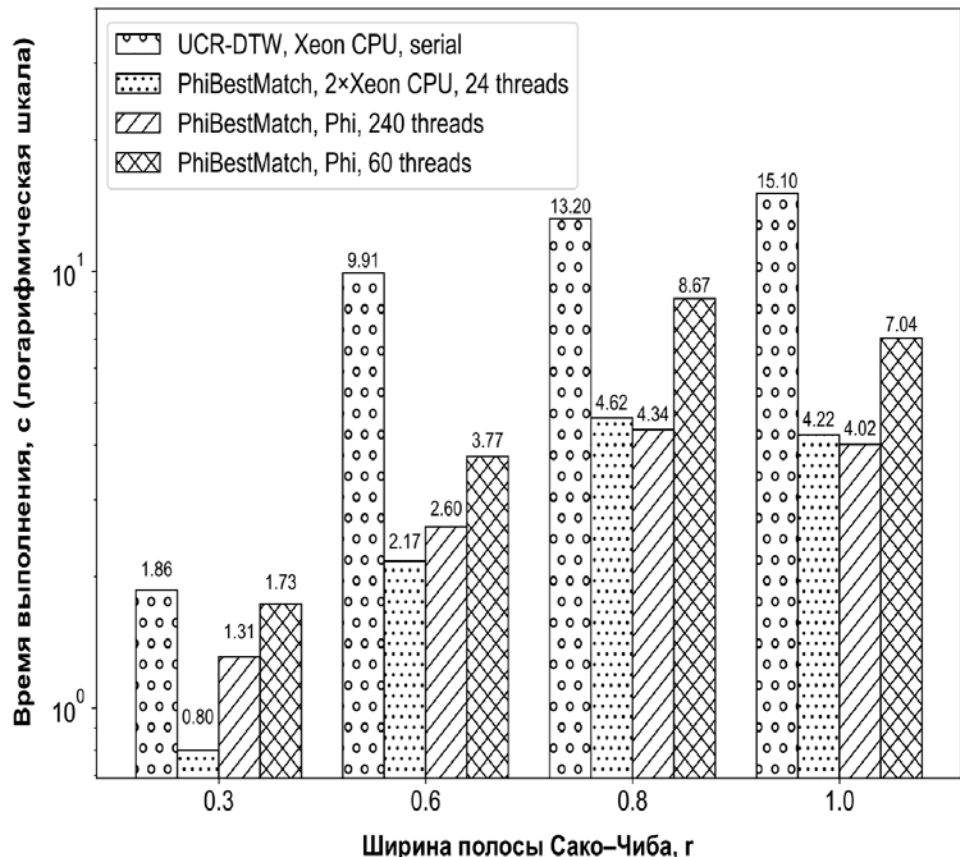
- Аппаратная платформа

Характеристика \ Устройство	Торнадо ЮУрГУ		ЦКП СО РАН
	2×Intel Xeon X5680	Intel Xeon Phi SE10X (KNC)	Intel Xeon Phi 7290 (KNL)
К-во физ. ядер	2×6	61	72
Гиперпоточность	2×	4×	4×
К-во лог. ядер	24	244	288
Частота, GHz	3.33	1.1	1.5
Память, Gb	8	8	16
Пиковая производительность, TFLOPS	0.371	1.076	3.456

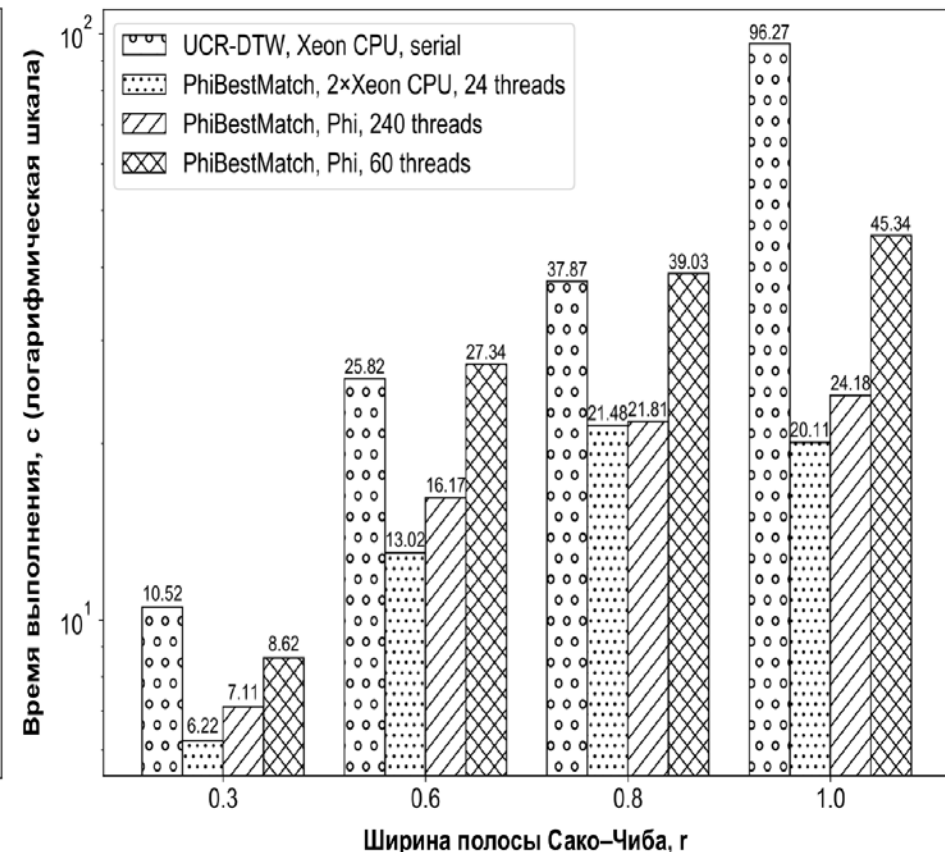
- Наборы данных
  - Random Walk
  - EPG
  - ECG

# Влияние параметра $r$ (на одном узле кластера)

## Ряд Random Walk ( $|T|=10^6$ , $|Q|=128$ )



## Ряд EPG ( $|T|=2.5 \cdot 10^5$ , $|Q|=368$ )

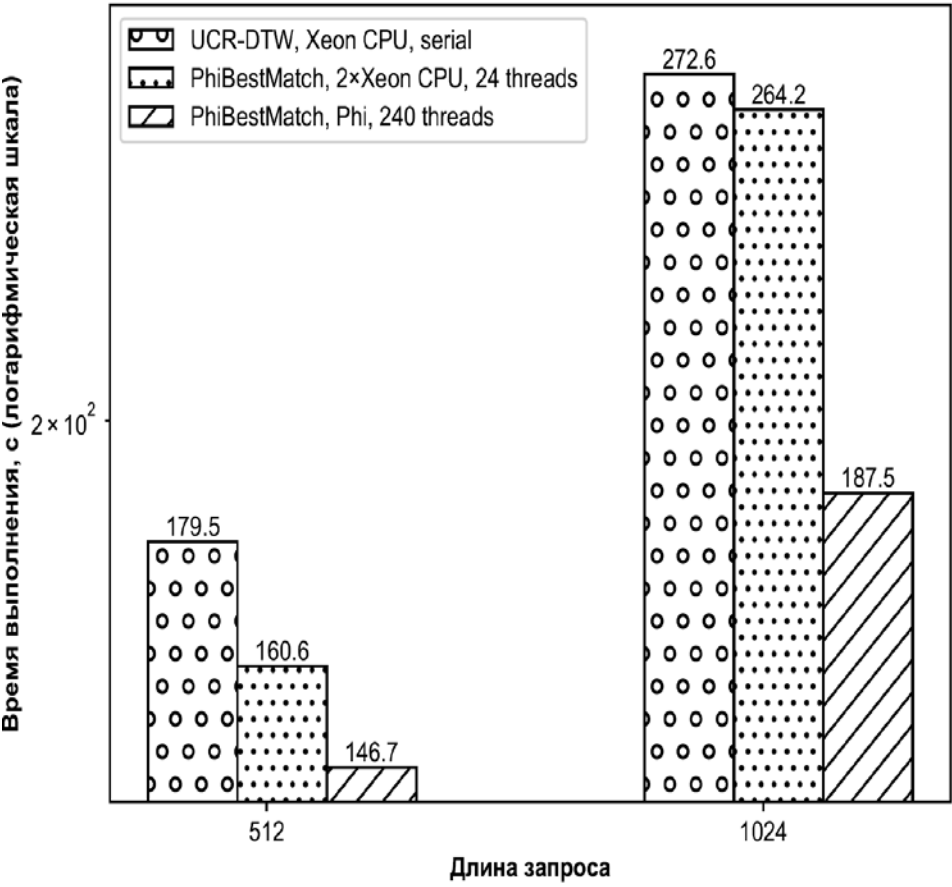


PhiBestMatch более эффективен, когда требуется поиск подпоследовательностей с большей точностью при определении схожести

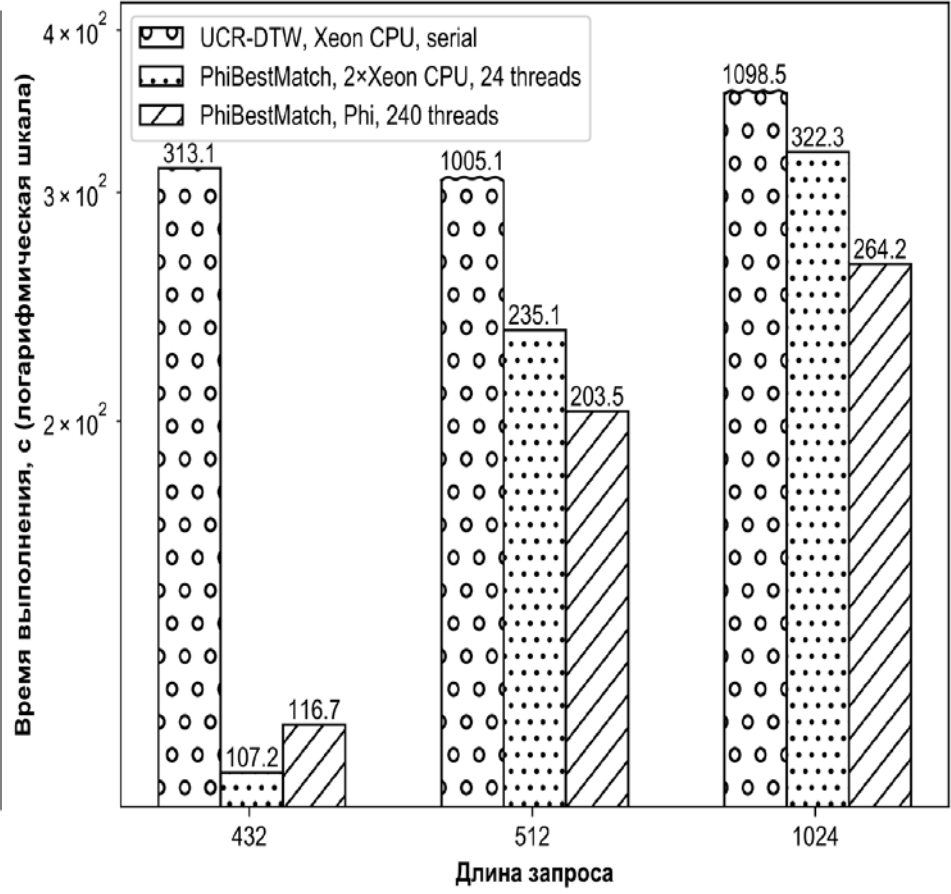


# Влияние длины запроса (на одном узле кластера)

Ряд Random Walk ( $|T|=10^6, r=0.8$ )



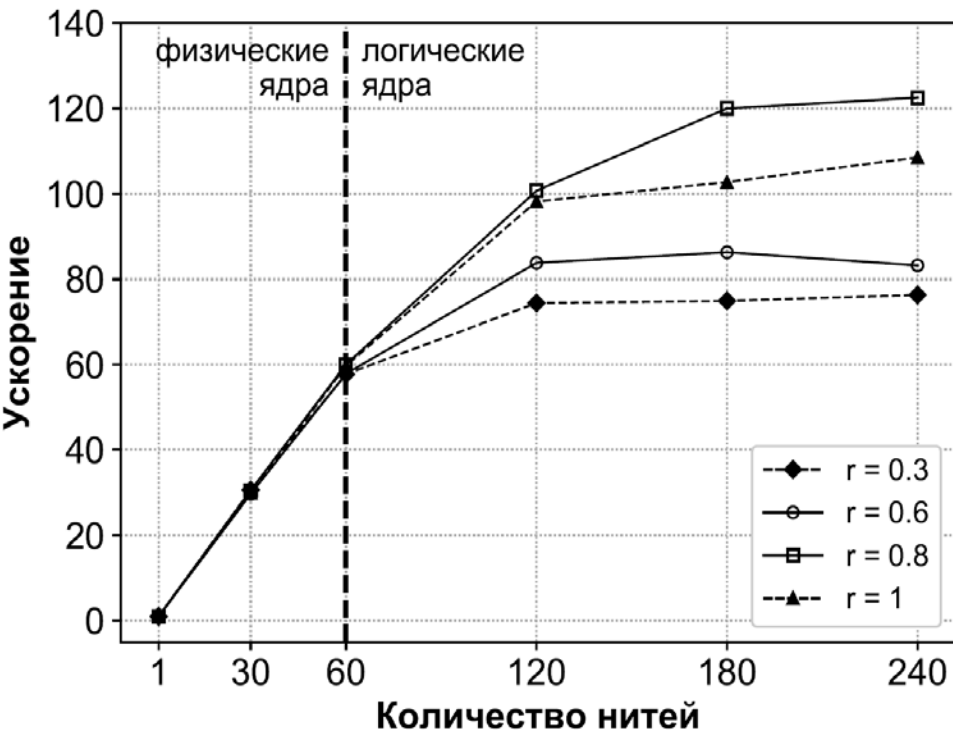
Ряд ЭКГ ( $|T|=10^6, r=0.8$ )



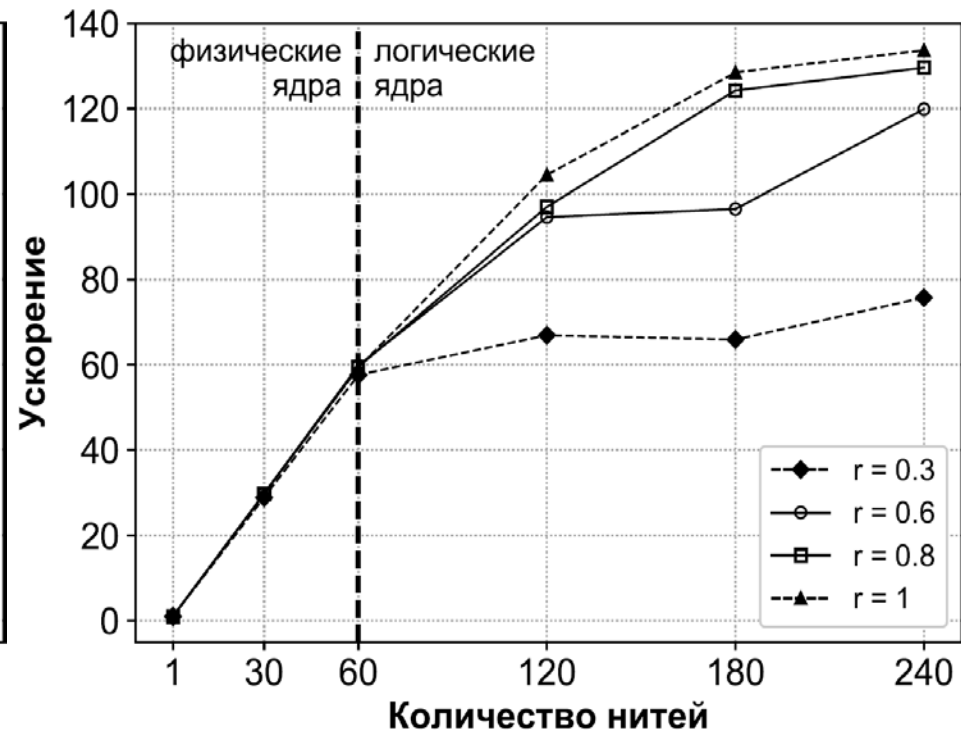
PhiBestMatch более эффективен, когда требуется поиск подпоследовательностей большей длины

# Ускорение (на одном узле кластера)

Ряд Random Walk ( $|T|=10^6$ ,  $|Q|=128$ )



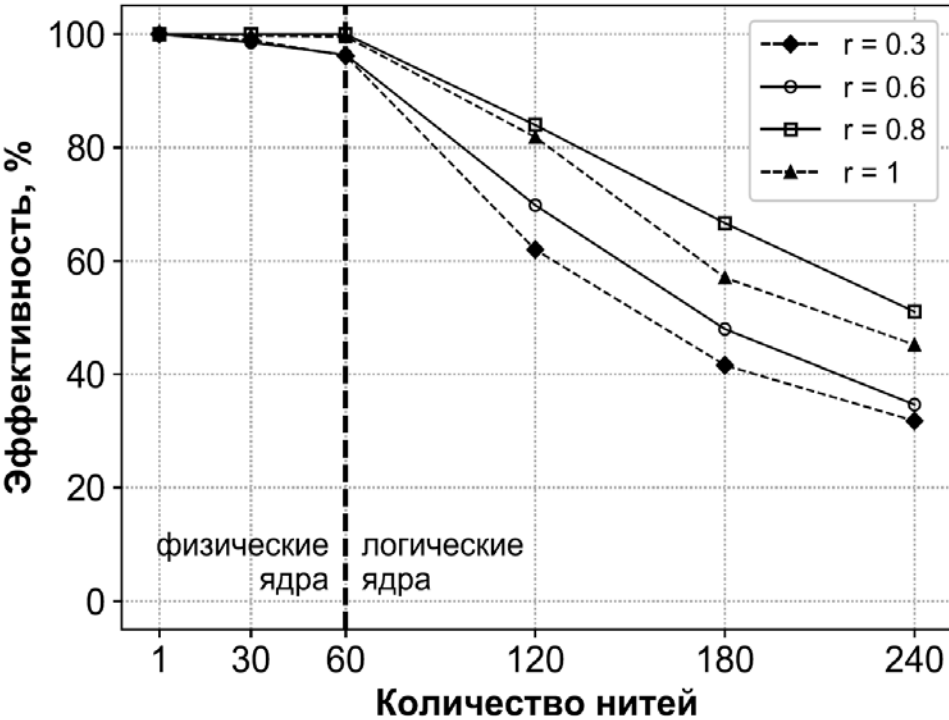
Ряд EPG ( $|T|=2.5 \cdot 10^5$ ,  $|Q|=368$ )



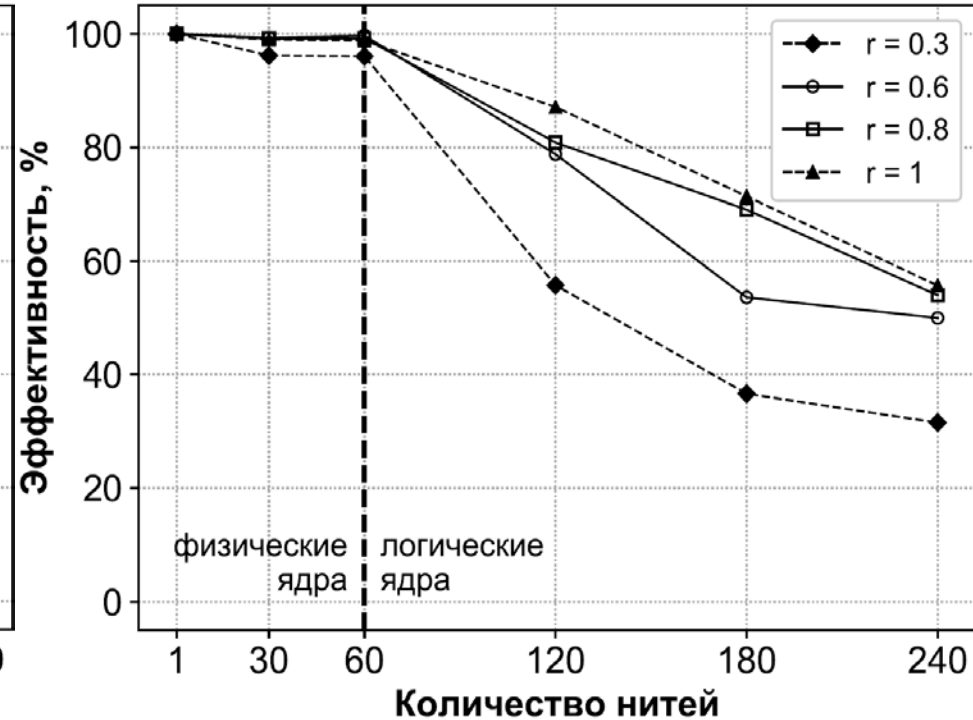
PhiBestMatch дает линейное ускорение, когда количество нитей совпадает с количеством физических ядер Intel Xeon Phi.  
При большем количестве нитей ускорение становится сублинейным

# Параллельная эффективность (на одном узле кластера)

Ряд Random Walk ( $|T|=10^6$ ,  $|Q|=128$ )



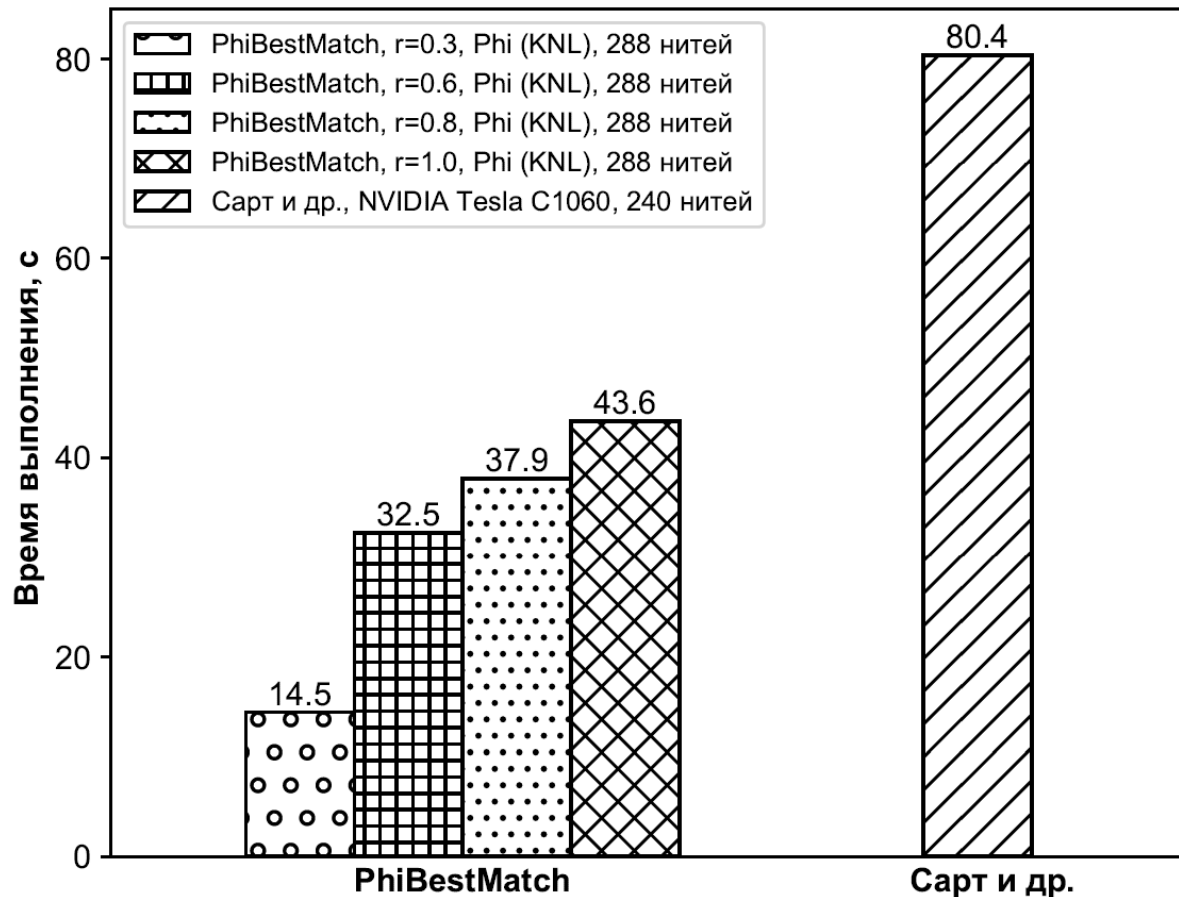
Ряд EPG ( $|T|=2.5 \cdot 10^5$ ,  $|Q|=368$ )



PhiBestMatch дает близкую к 100% эффективность, когда количество нитей совпадает с количеством физических ядер Intel Xeon Phi. При большем количестве нитей эффективность падает до 50%

# Сравнение с GPU (на одном узле кластера)

Ряд EPG ( $|T|=1.5 \cdot 10^6$ ,  $|Q|=368$ )



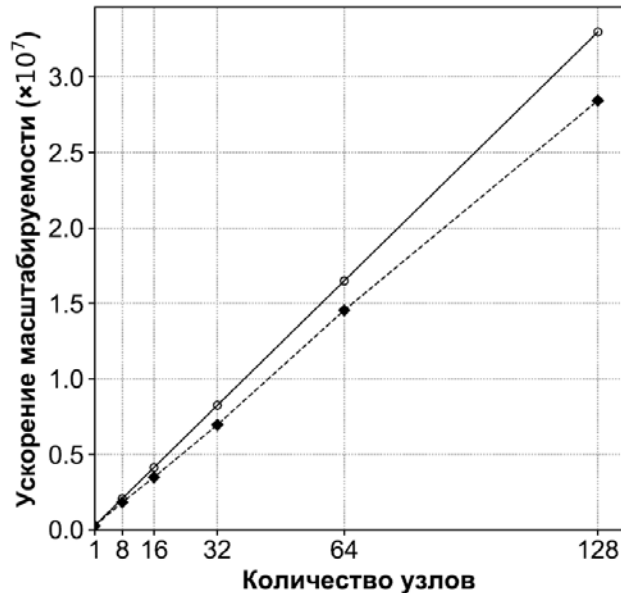
Устройство	Intel Xeon Phi 7290	NVIDIA Tesla C1060
Характеристика		
К-во физич. ядер	72	240
Гиперпоточность	4x	
К-во логич. ядер	288	
Частота, GHz	1.5	1.3
Пик. пр., TFLOPS	3.456	0.93

Sart D., et al. Accelerating Dynamic Time Warping subsequence search with GPUs and FPGAs. ICDM. IEEE, 2010. pp. 1001–1006.

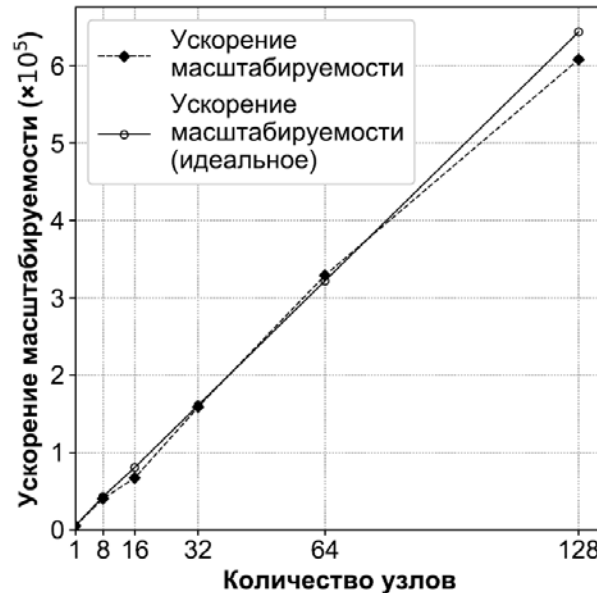
# Ускорение масштабируемости (на кластере)

Ряд Random Walk ( $|T^{(i)}|=10^6$ )

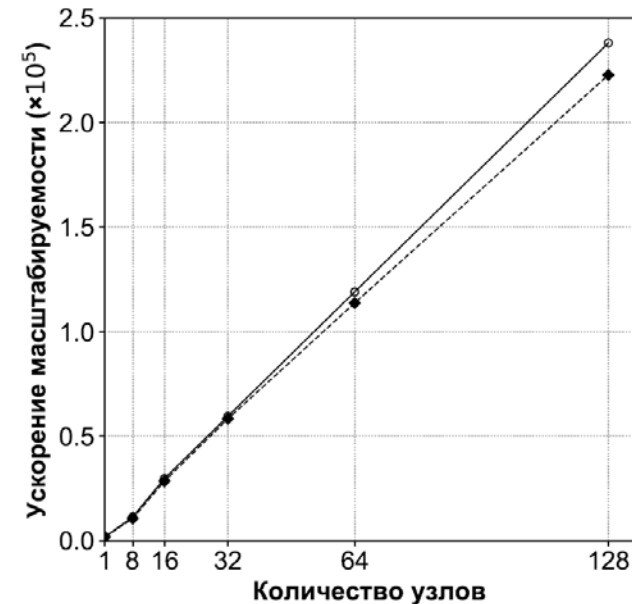
$|Q|=128$



$|Q|=512$



$|Q|=1024$



$$S_{scaled} = \frac{p \cdot m}{t_p(p \cdot m)}$$

$p$  – кол-во узлов

$m$  – объем данных (длина ряда)

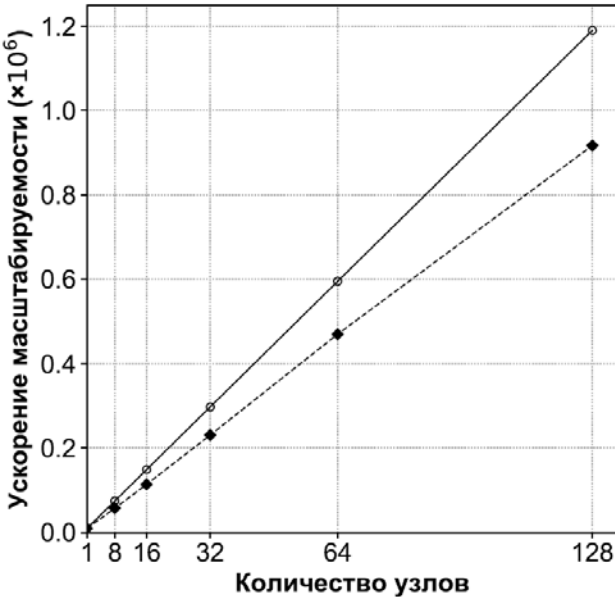
$t_p(p \cdot m)$  – время обработки ряда длины  $p \cdot m$  на  $p$  узлах

PhiBestMatch показывает ускорение масштабируемости, близкое к линейному (при увеличении длины поискового запроса)

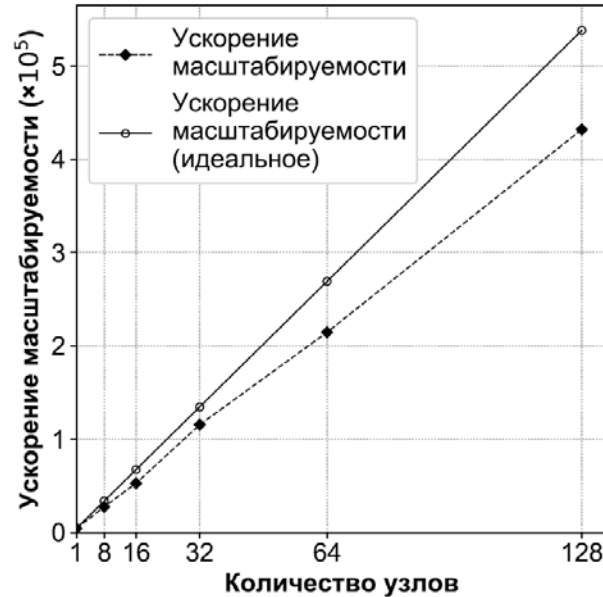
# Ускорение масштабируемости (на кластере)

Ряд ЭКГ ( $|T^{(i)}|=10^6$ )

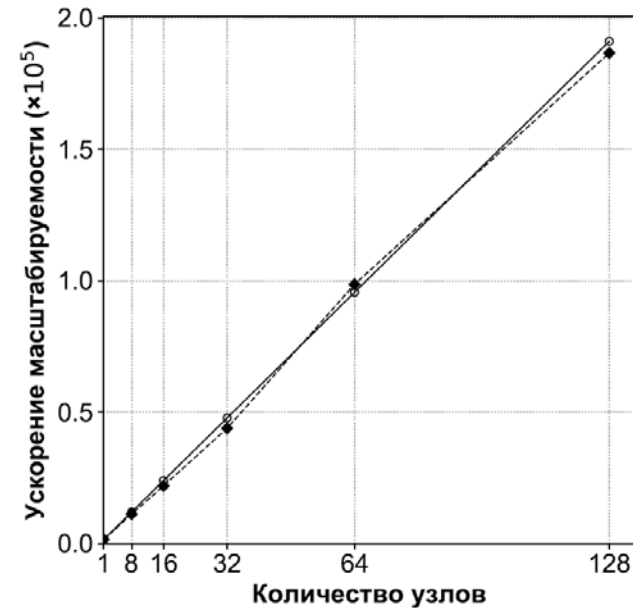
$|Q|=432$



$|Q|=512$



$|Q|=1024$



$$S_{scaled} = \frac{p \cdot m}{t_p(p \cdot m)}$$

$p$  – кол-во узлов

$m$  – объем данных (длина ряда)

$t_p(p \cdot m)$  – время обработки ряда длины  $p \cdot m$  на  $p$  узлах

PhiBestMatch показывает ускорение масштабируемости, близкое к линейному (при увеличении длины поискового запроса)

# Сравнение с распределенным алгоритмом

- Набор данных
  - Random Walk ( $|T|=2.2 \cdot 10^8$ ,  $|Q|=128$ )
- Платформа
  - Торнадо ЮУрГУ: 220 узлов
  - Кластер Spark: 6 узлов с Intel Xeon E3-1200v2 3.10 ГГц

Производительность алгоритма на указанной платформе, с				Ускорение относительно алгоритма UCR-DTW	
Intel Xeon Phi		Кластер Spark (6 узлов)			
PhiBestMatch	UCR-DTW	Шабиб и др.	UCR-DTW		
0.69	74.43	32	137	107.87	4.28

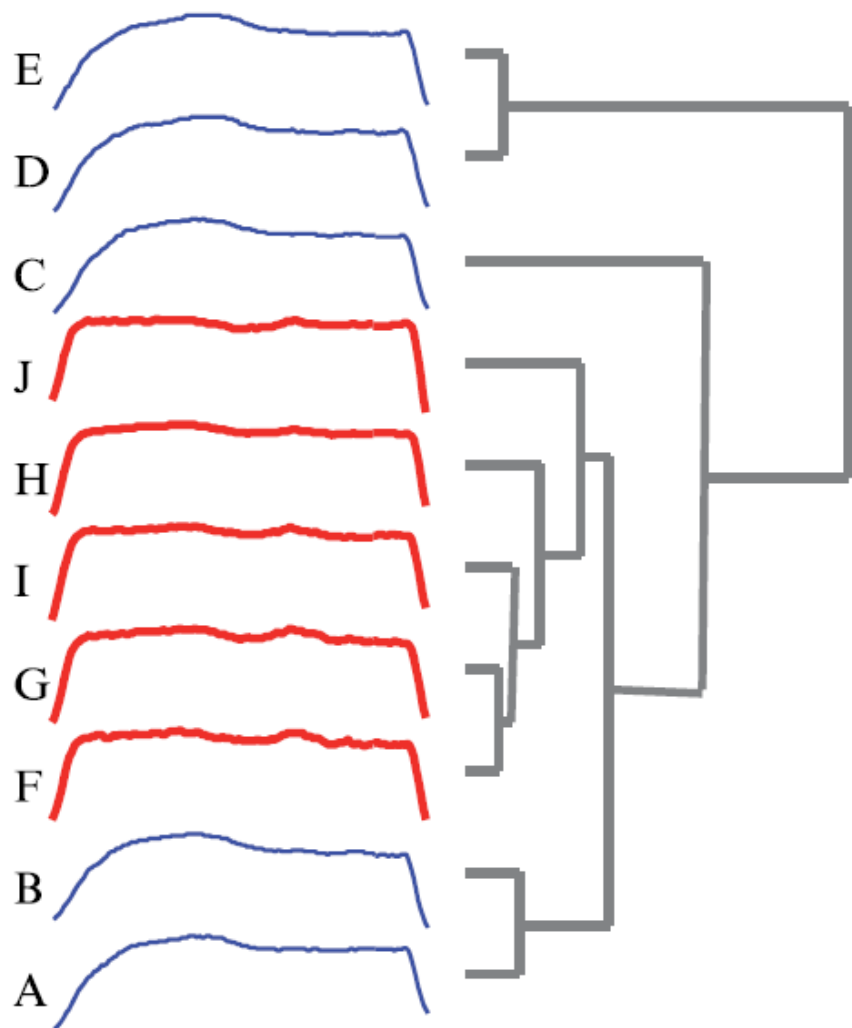
Shabib A., Narang A., Niddodi C.P. et al. Parallelization of searching and mining time series data using Dynamic Time Warping. 2015 Int. Conf. on Advances in Computing, Communications and Informatics. IEEE, 2015. pp. 343–348.

# PhiBestMatch: за и против

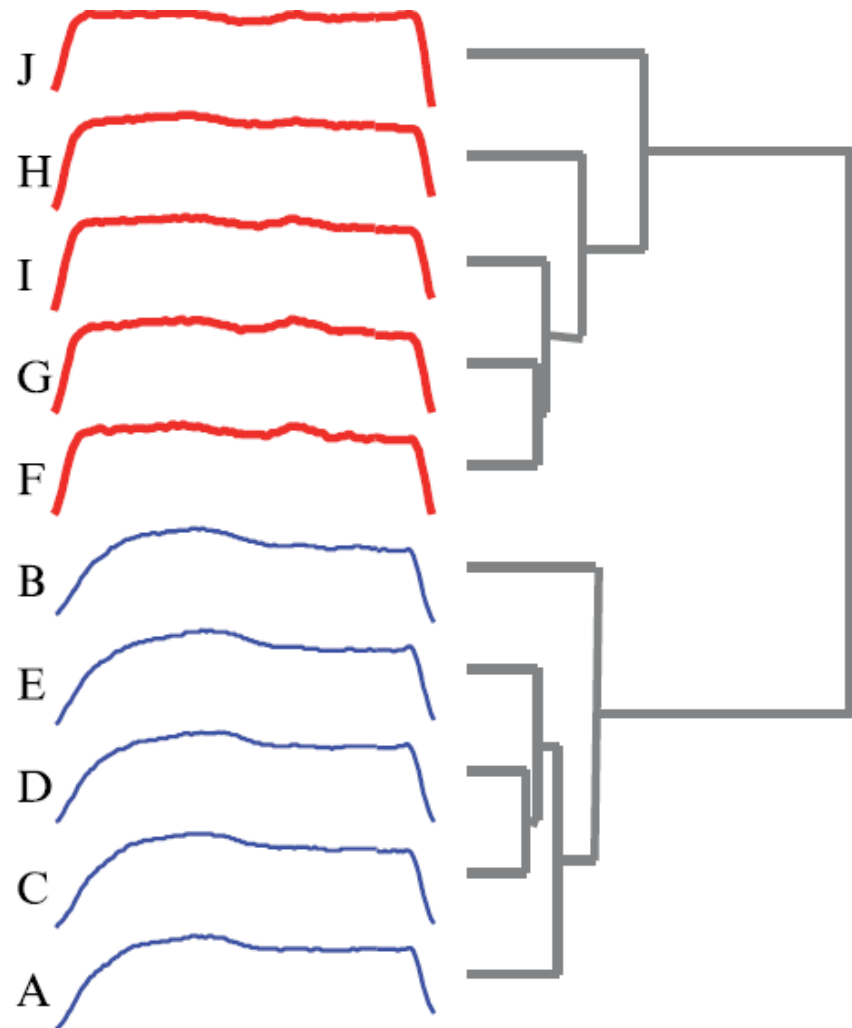
- 😊 Хорошая масштабируемость и выгода от использования Intel Xeon Phi при условиях
  - длина образца поиска  $n \geq 128$  (в среднем)
  - точность при определении схожести  $r \geq 0.5n$  (в среднем)
  - длина фрагмента/ряда  $|T^{(i)}| \geq 10^6$
- 😞 Большая пространственная сложность:  $O(mn)$
- 😐 При раннем существенном улучшении оценки схожести *bsf* предварительные вычисления будут существенно избыточными



# Необходимость z-нормализации



Unnormalized Euclidean Distance

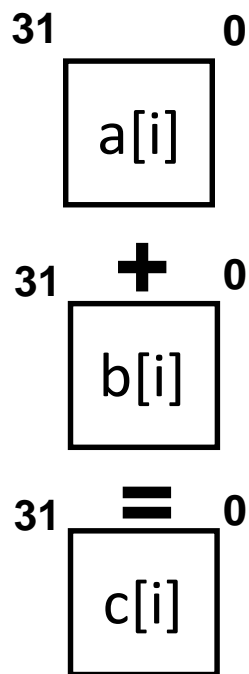


Normalized Euclidean Distance

# Векторизация

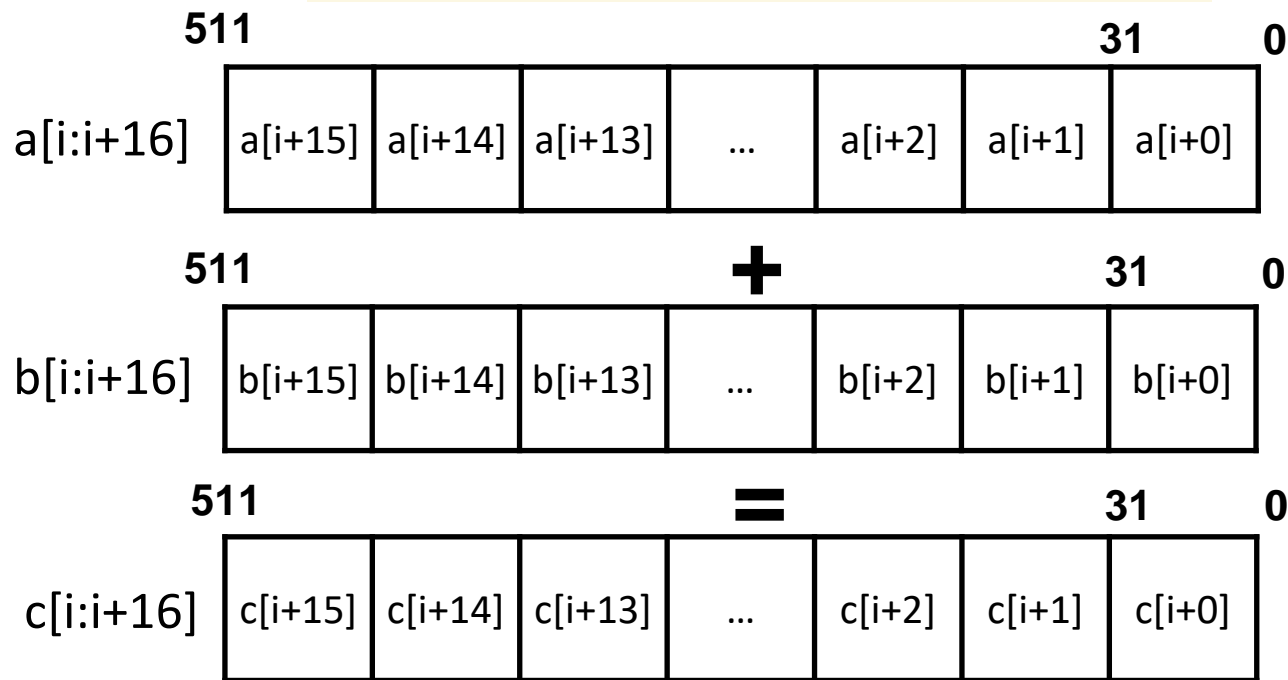
## Скалярная операция

```
float a[N], b[N], c[N];  
for (i = 0; i < N; i++)  
    c[i] = a[i] + b[i];
```



## Векторная операция

```
float a[N] __attribute__((aligned(64)));  
float b[N] __attribute__((aligned(64)));  
float c[N] __attribute__((aligned(64)));  
for (i = 0; i < N; i+=16)  
    c[i:i+16] = a[i:i+16] + b[i:i+16];
```



# Выравнивание данных

