DAMDID/RCDL'2018

Data Analytics and Management in Data Intensive Domains

Moscow, Russia, October 9–12, 2018

An Efficient Subsequence Similarity Search on Modern Intel Many-core Processors for Data Intensive Applications

Yana Kraeva and Mikhail Zymbler

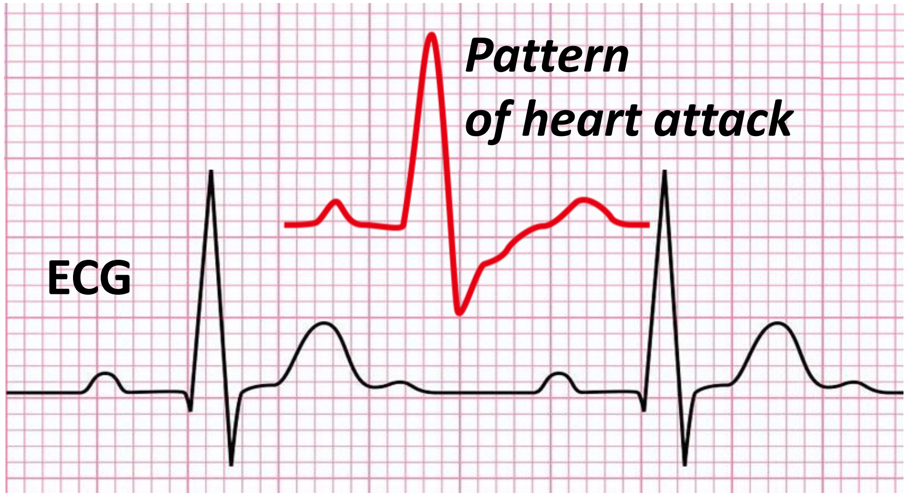
South Ural State University, Chelyabinsk, Russia

kraevaya@susu.ru, mzym@susu.ru

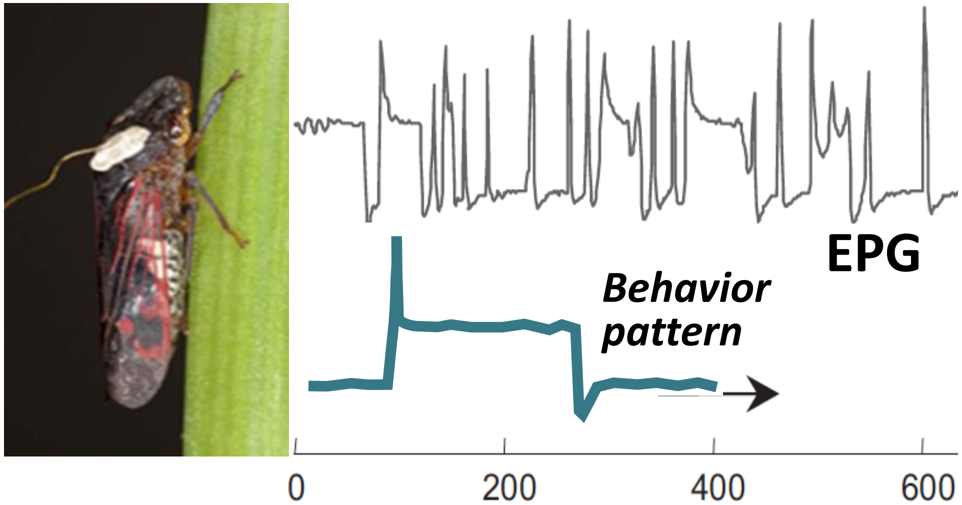
This work was financially supported by the Russian Foundation for Basic Research (grant No. 17-07-00463), by Act 211 Government of the Russian Federation (contract No. 02.A03.21.0011) and by the Ministry of education and science of Russian Federation (government order 1.9624.2017/7.8).

Similarity search: typical applications

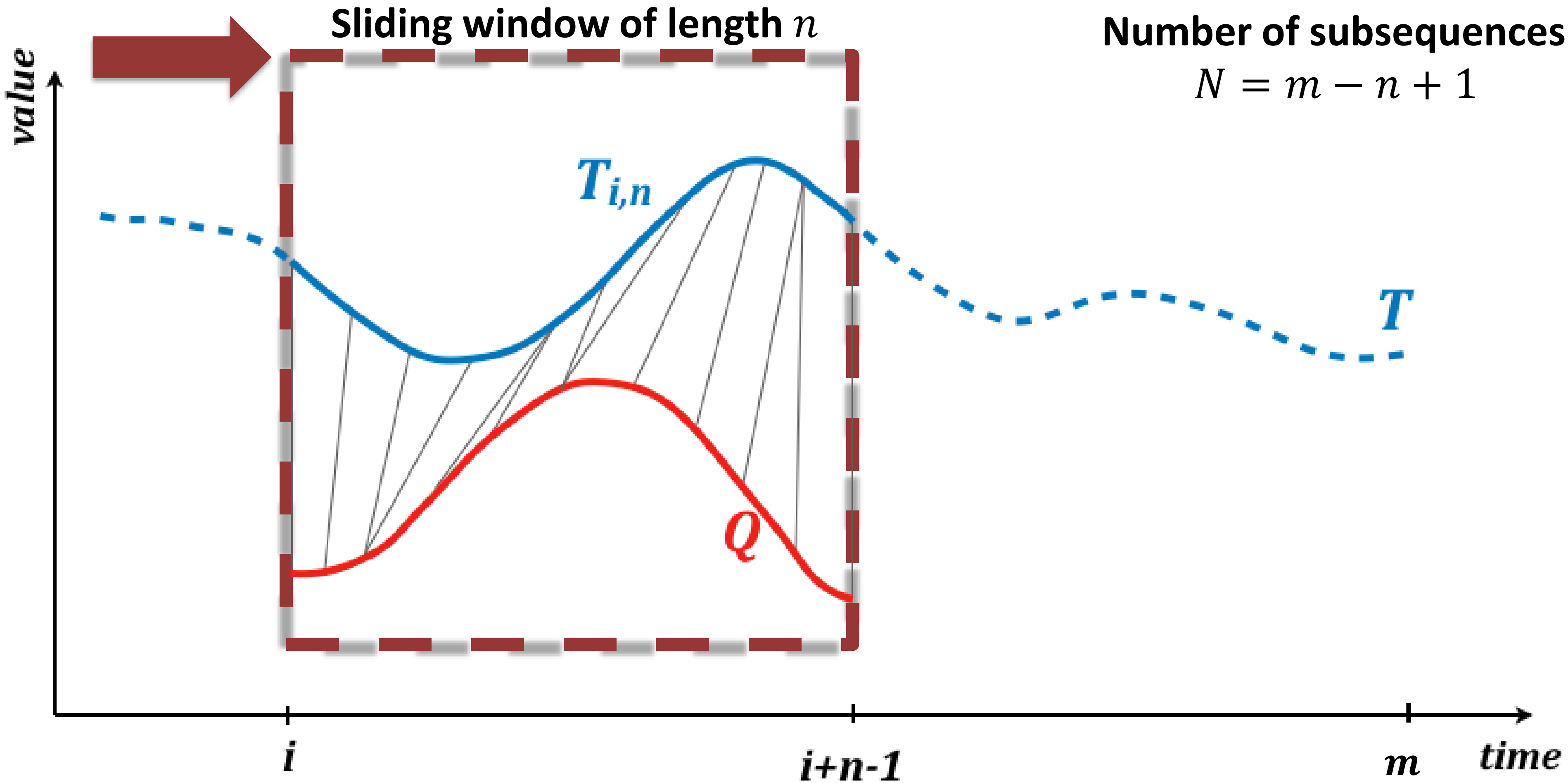
Medicine:
search for heart attack patterns



Entomology:
studying behavior of insects



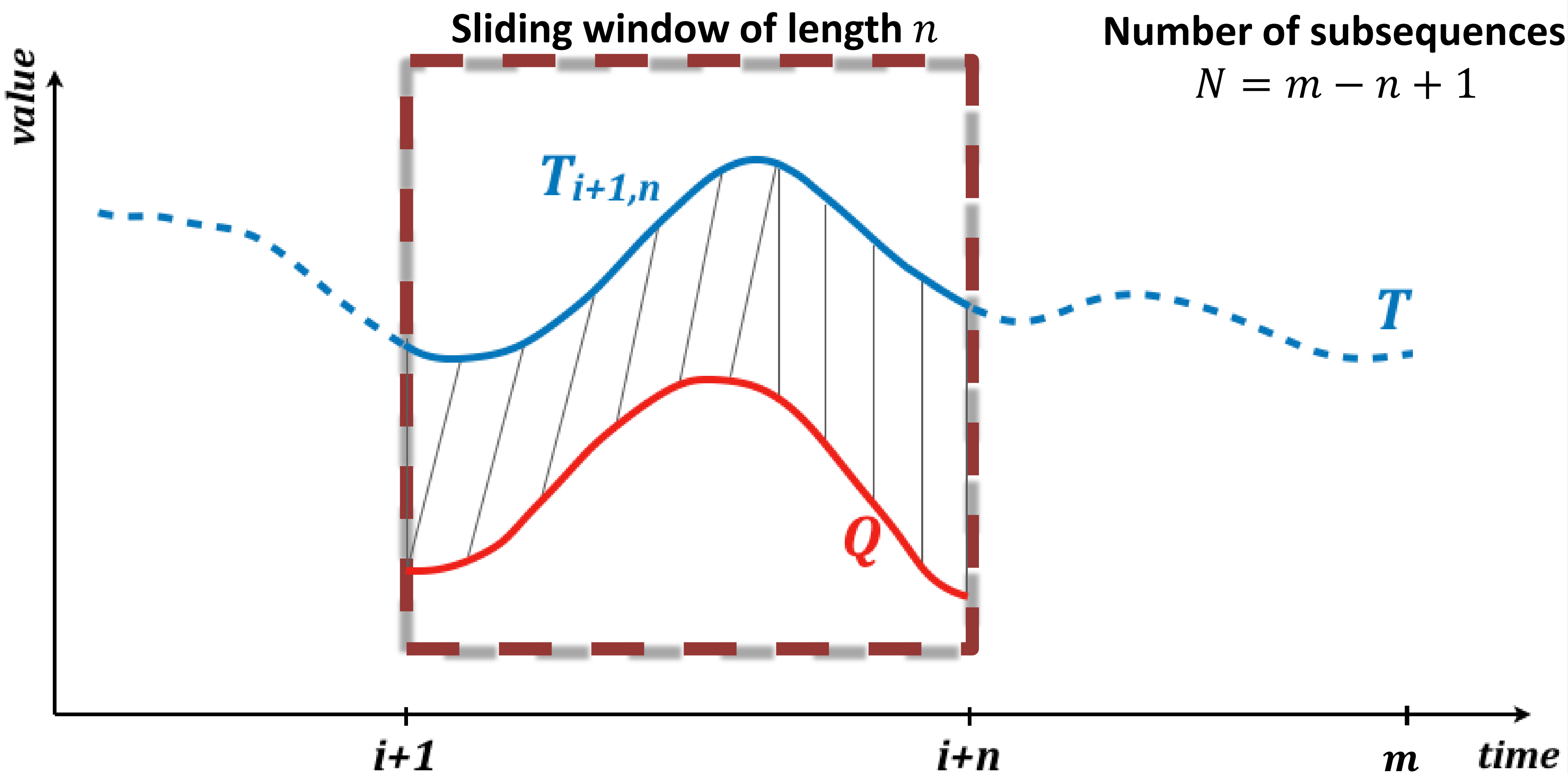
Subsequence similarity search



Number of subsequences
$N = m - n + 1$

$$\exists\, i > 1\ \forall\, j < N \quad DTW\left(Q, T_{i,n}\right) < DTW\left(Q, T_{j,n}\right)$$

Subsequence similarity search

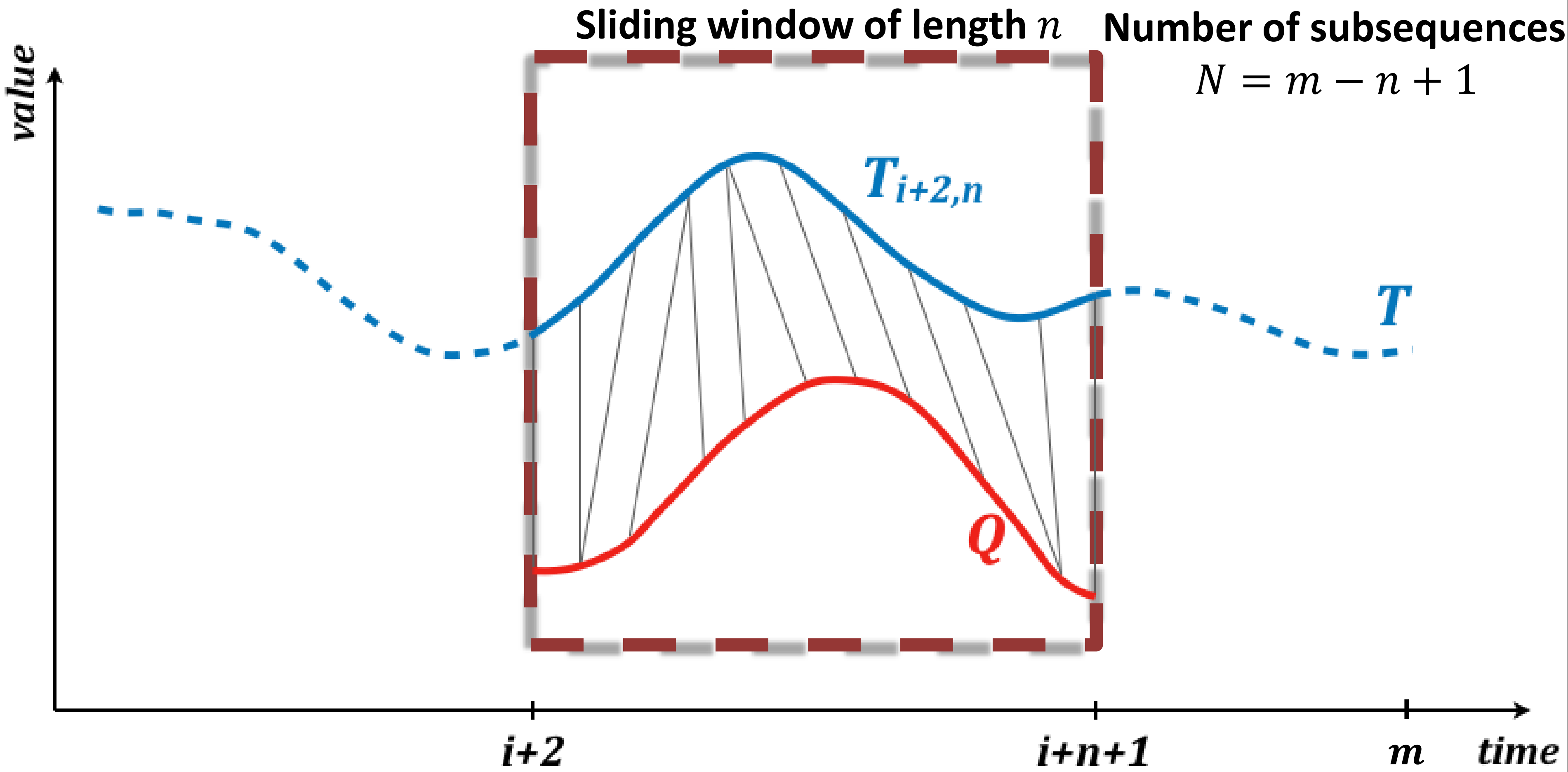


Number of subsequences

$$N = m - n + 1$$

$$\exists\, i > 1 \;\forall\, j < N \quad DTW\left(Q, T_{i,n}\right) < DTW\left(Q, T_{j,n}\right)$$

Subsequence similarity search



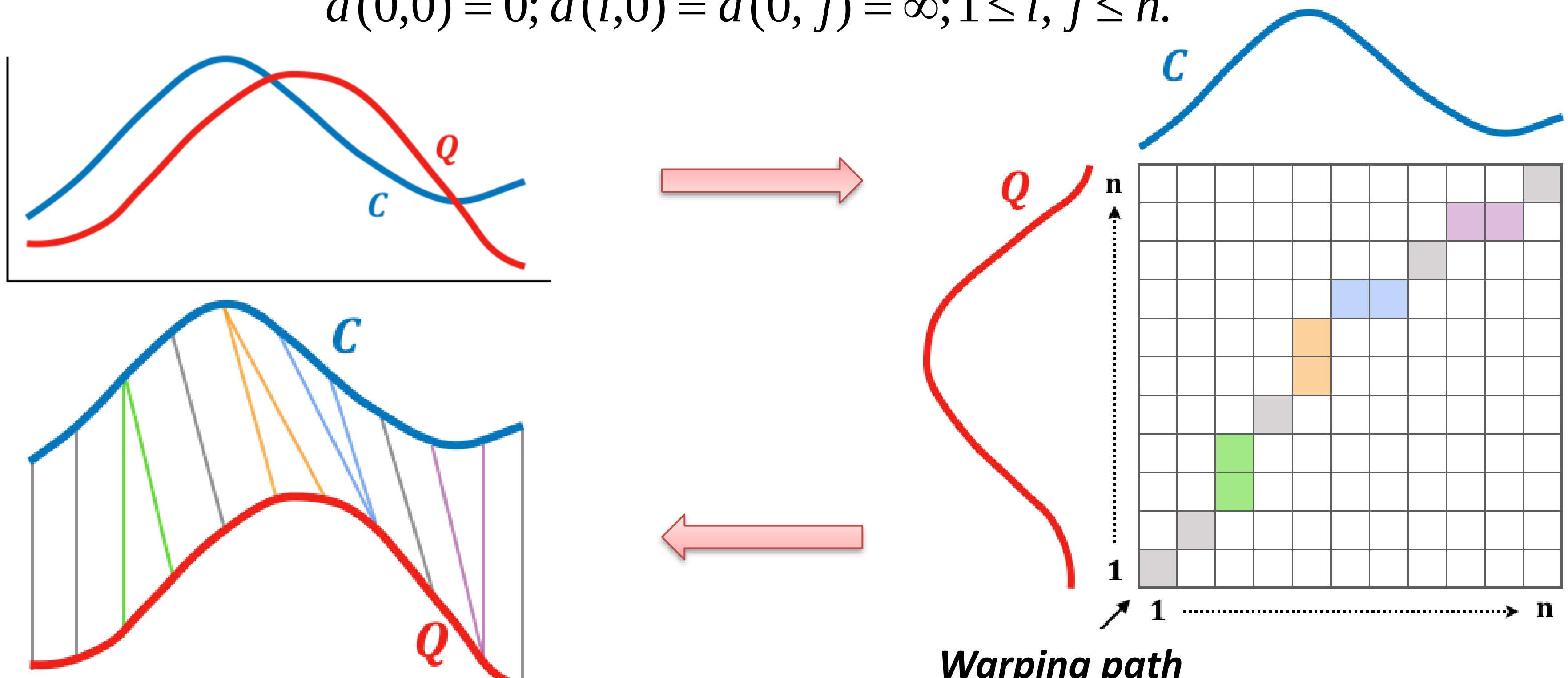
$$\exists\, i > 1\ \forall j < N \quad DTW\left(Q, T_{i,n}\right) < DTW\left(Q, T_{j,n}\right)$$

Dynamic Time Warping measure

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

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

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

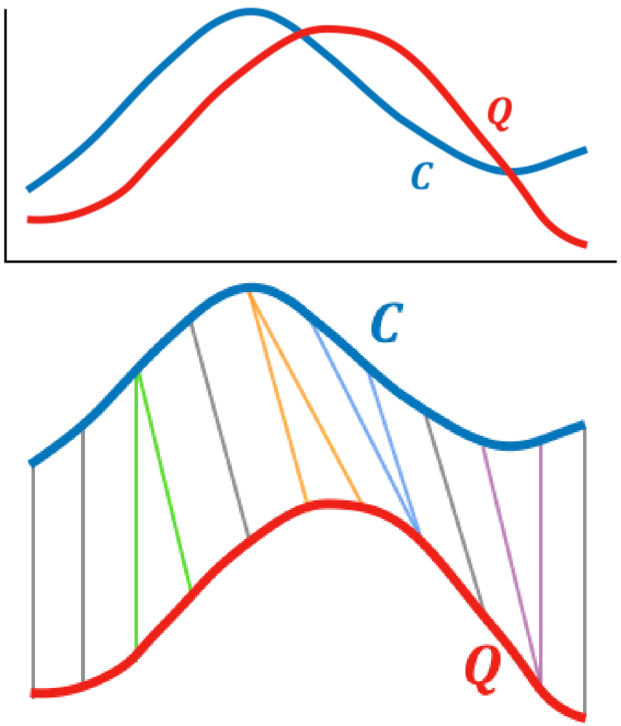


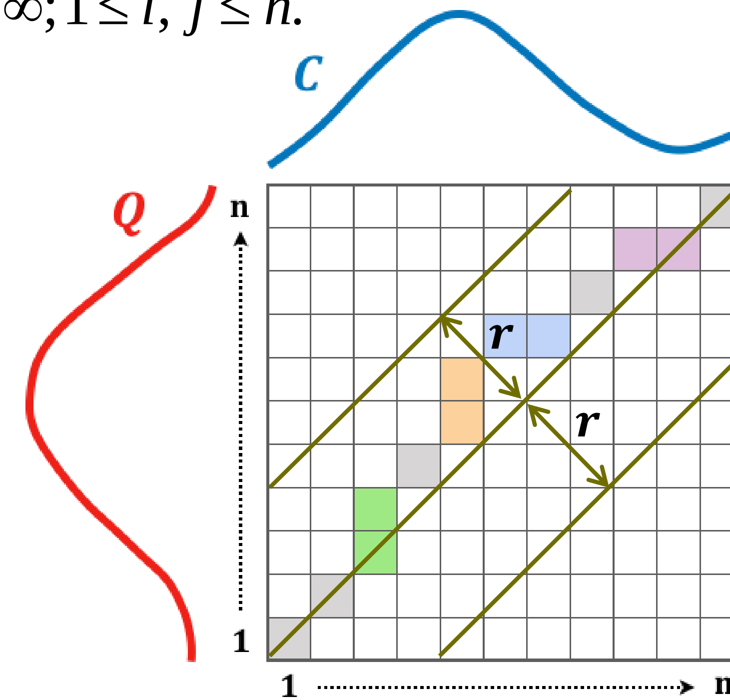
Dynamic Time Warping measure

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

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

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



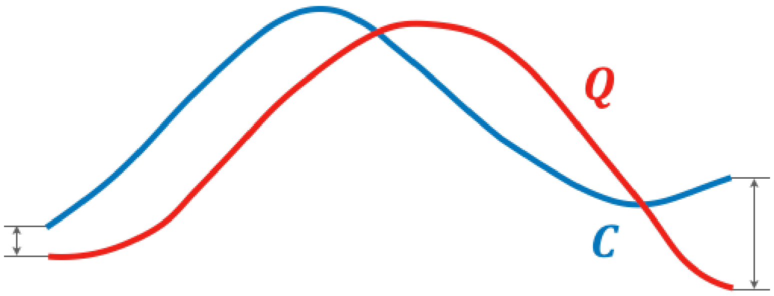


Warping constraint (Sako–Chiba band)

UCR-DTW: serial similarity search

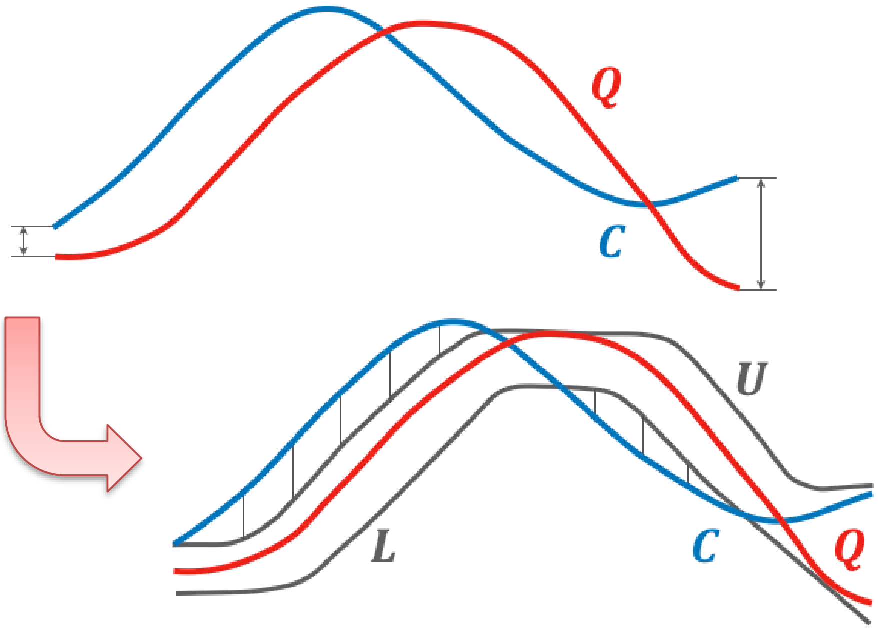
- Z-normalization
  - The mean and standard deviation of query and each candidate subsequence must be normalized 0 and 1, respectively
  - $\hat{t}_i = \frac{t_i - \mu}{\sigma}, \quad \mu = \frac{1}{m}\sum_{i=1}^{m} t_i, \quad \sigma = \frac{1}{m}\sqrt{\sum_{i=1}^{m} {t^2}_i - \mu^2}$
- Lower bounding
  - Cascade of cheap-to-compute lower bounds (LB) helps to prune clearly dissimilar candidates
  - If LB has exceeded the threshold, the DTW measure will exceed the threshold as well

Lower bounding



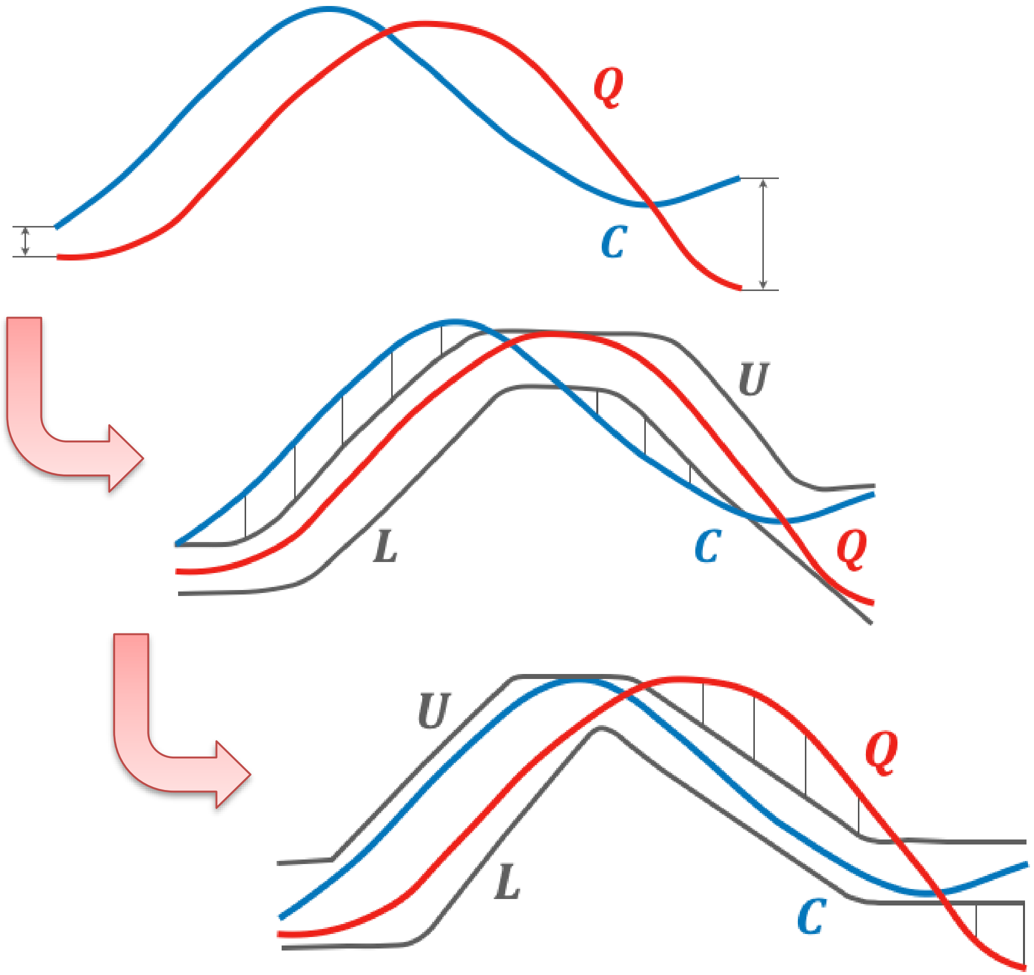
Lower Bound	Complexity
$LB_{Kim}(Q,C) = (q_1 - c_1)^2 + (q_n - c_n)^2$	$O(1)$

Lower bounding



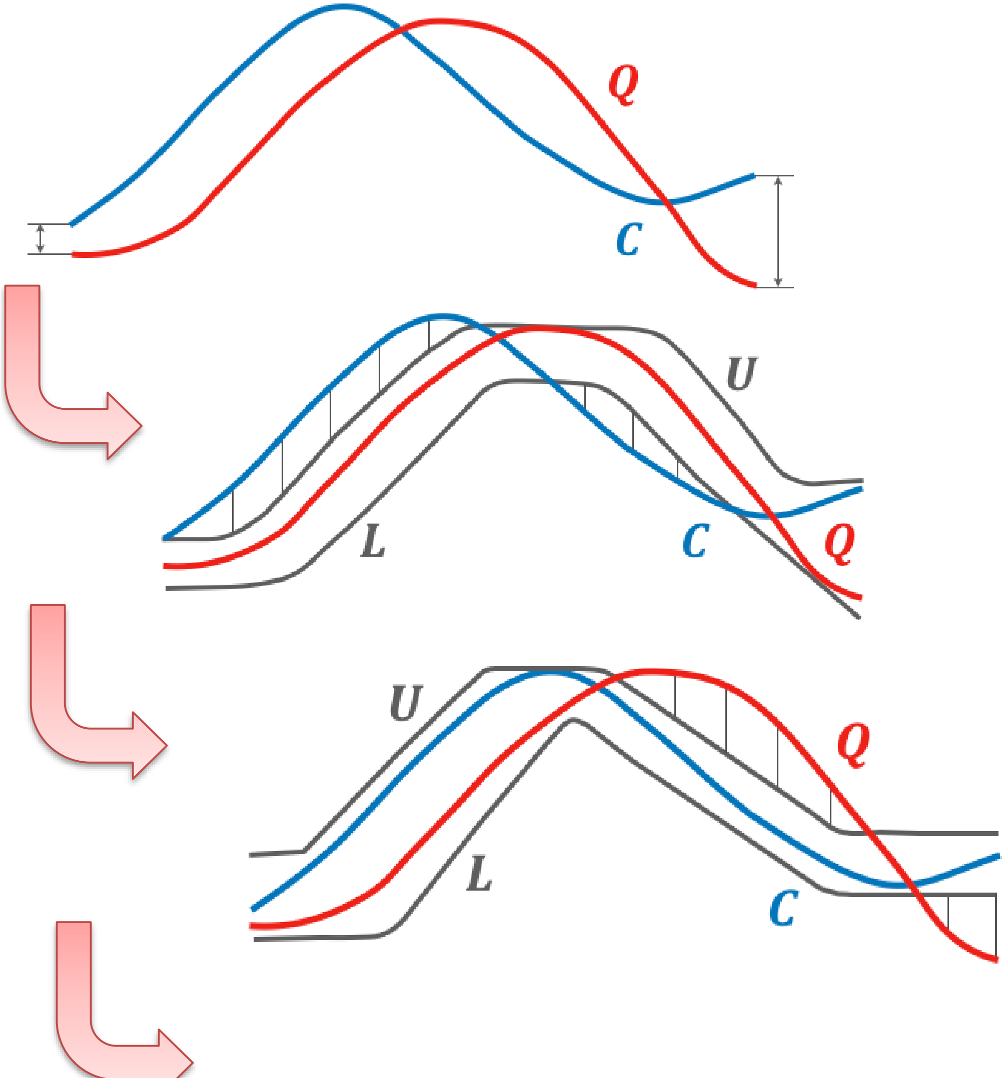
Lower Bound	Complexity
$LB_{Kim}(Q,C) = (q_1 - c_1)^2 + (q_n - c_n)^2$	$O(1)$
$LB_{Keogh}EC(Q,C) = \sum_{i=1}^{n} \begin{cases} (c_i - u_i)^2, & c_i > u_i \\ (c_i - \ell_i)^2, & c_i < \ell_i \\ 0, & otherwise \end{cases}$ $u_i = \max_{i-r \le k \le i+r} q_k$ $\ell_i = \min_{i-r \le k \le i+r} q_k$	$O(n)$

Lower bounding



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

Lower bounding

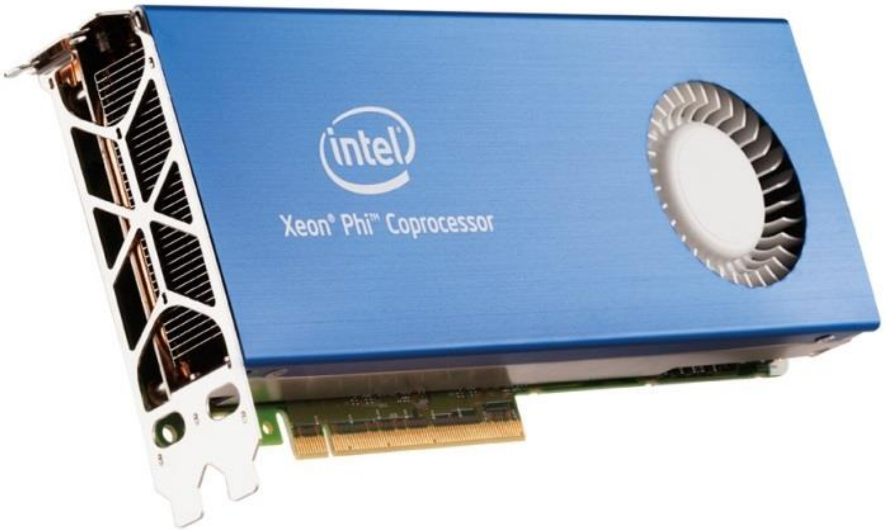


$$best\_so\_far = min(DTW(Q,C), bsf_{prev})$$

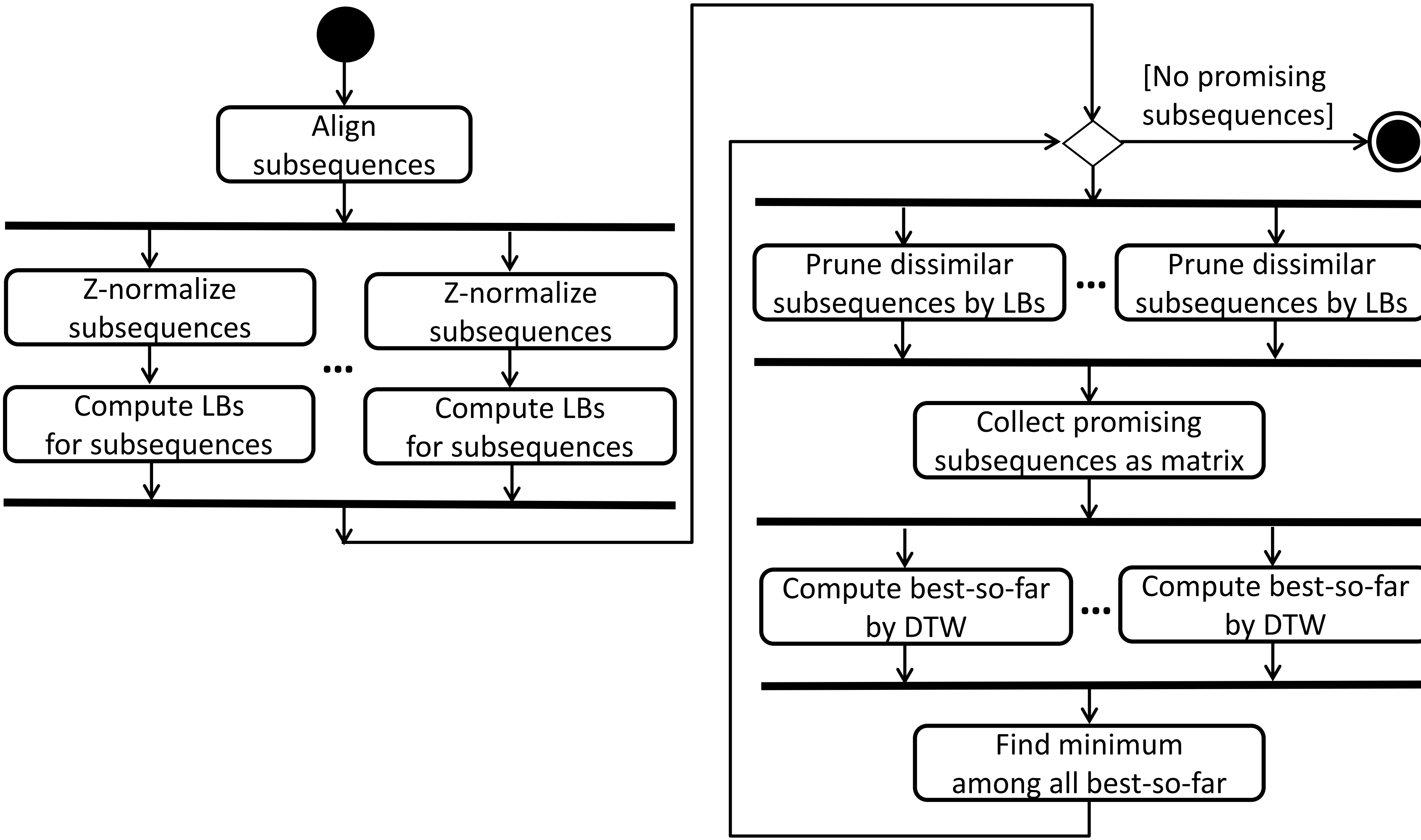
Lower Bound	Complexity
$LB_{Kim}(Q,C) = (q_1 - c_1)^2 + (q_n - c_n)^2$	$O(1)$
$LB_{Keogh}EC(Q,C) = \sum_{i=1}^{n} \begin{cases} (c_i - u_i)^2, & c_i > u_i \\ (c_i - \ell_i)^2, & c_i < \ell_i \\ 0, & otherwise \end{cases}$ $u_i = \max_{i-r \le k \le i+r} q_k$ $\ell_i = \min_{i-r \le k \le i+r} q_k$	$O(n)$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

Intel Xeon Phi many-core systems

- **Up to 72 compute cores**, each core provides
  - relatively weak computing power than the Intel Xeon CPU core
  - 512-bit wide **vector processing unit (VPU) and SIMD instructions**
- Based on **x86 architecture** and supports IDE and programming models for Intel CPUs

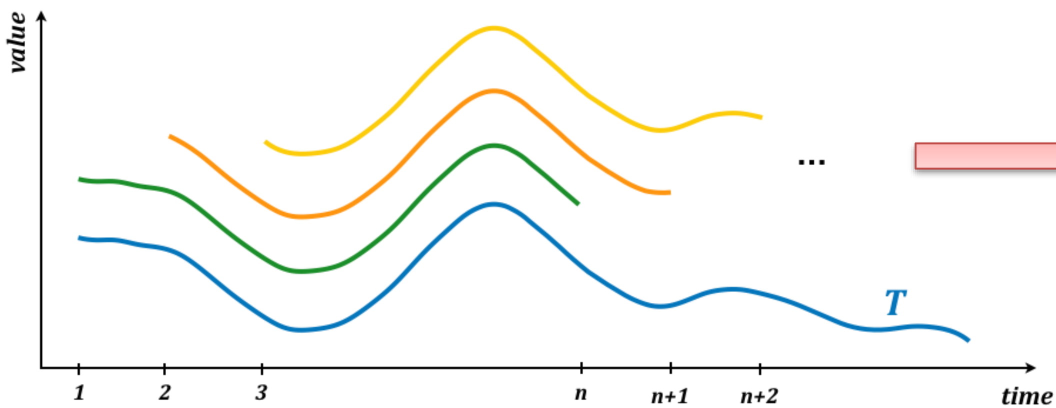


PhiBestMatch: novel algorithm for Intel MIC

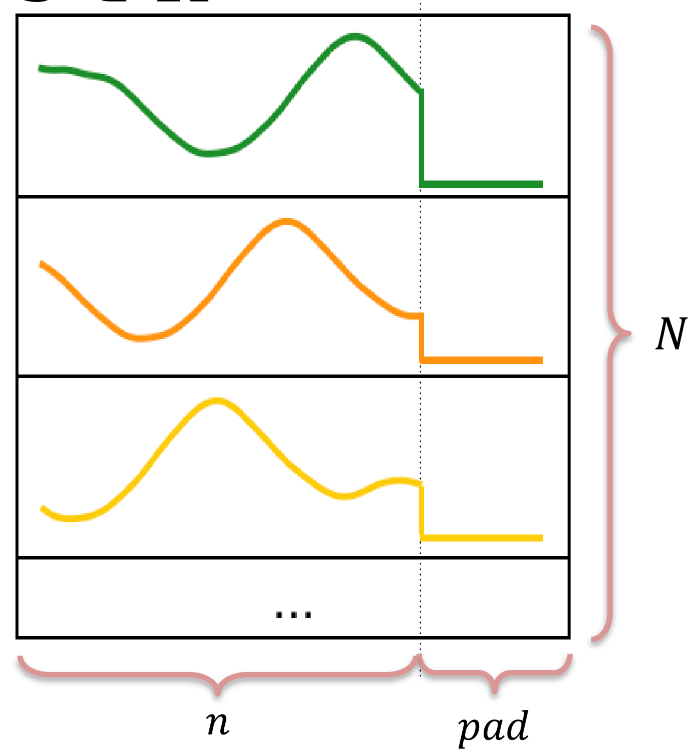


Subsequence matrix

$T \in \mathbb{R}^m$

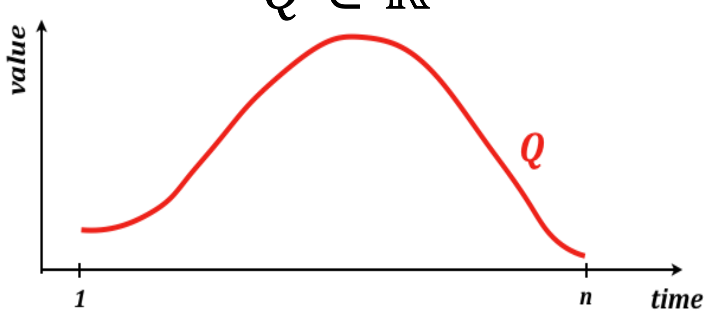


$\boldsymbol{S} \in \mathbb{R}^{N \times (n+pad)}$

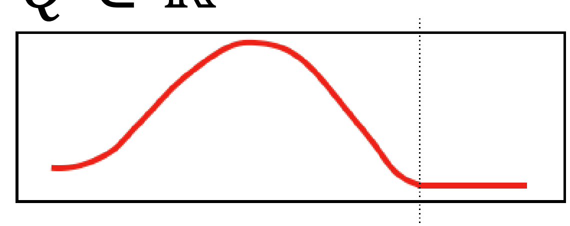


Data alignment: $(n + pad) \vdots width_{VPU}$

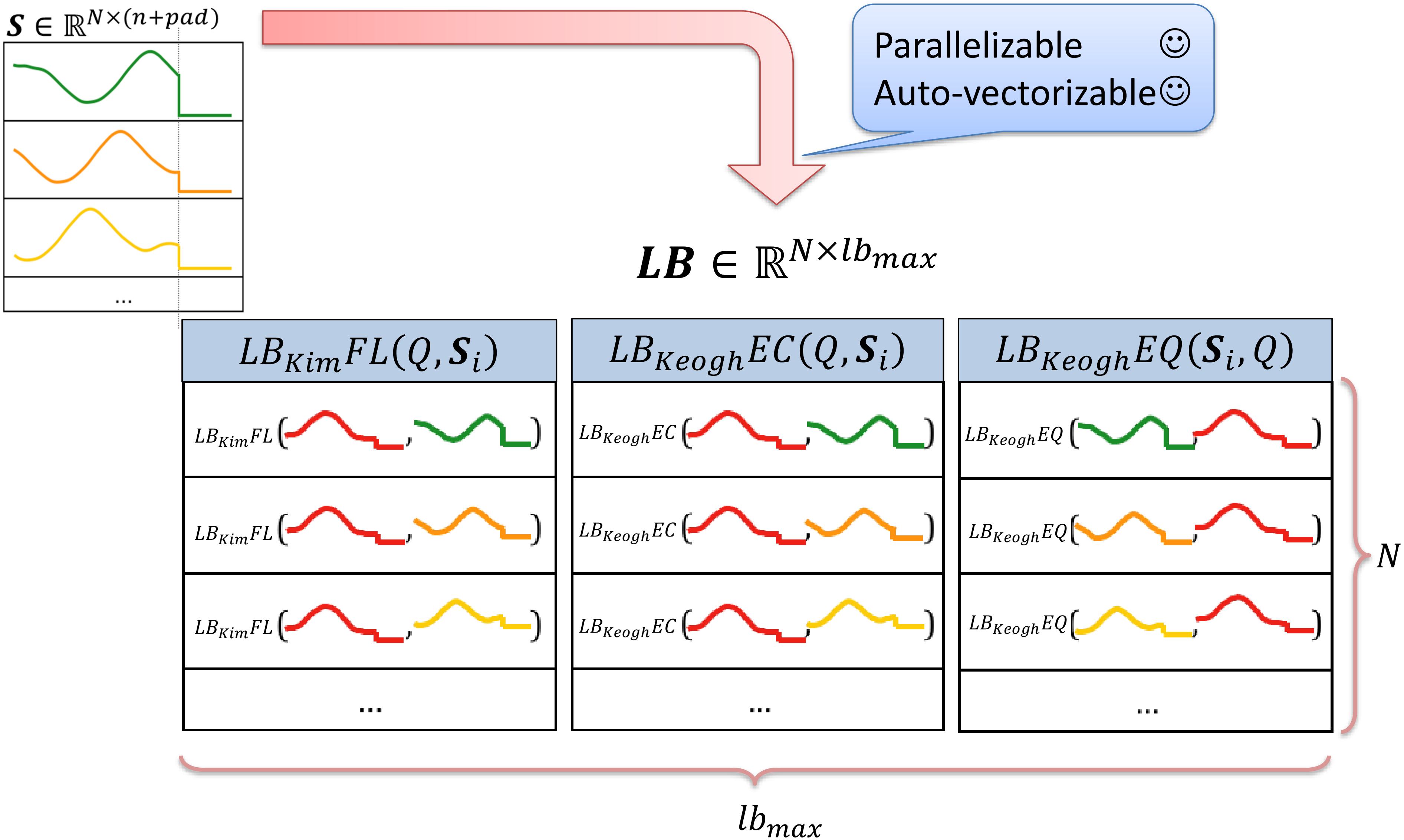
$Q \in \mathbb{R}^n$



$Q \in \mathbb{R}^{n+pad}$

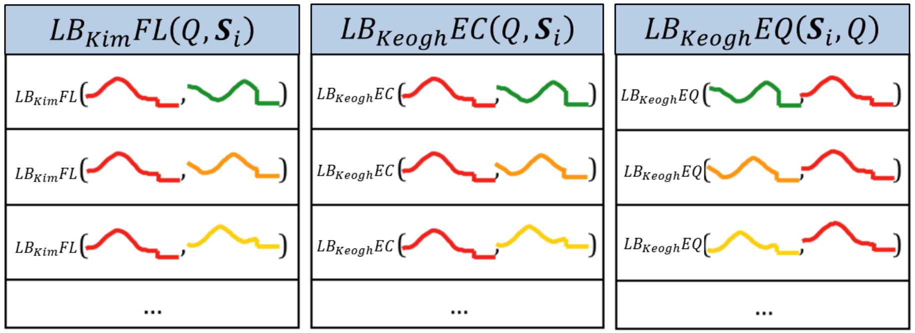


Matrix of Lower Bounds



Bitmap matrix

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



Parallelizable ☺
Auto-vectorizable☺

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

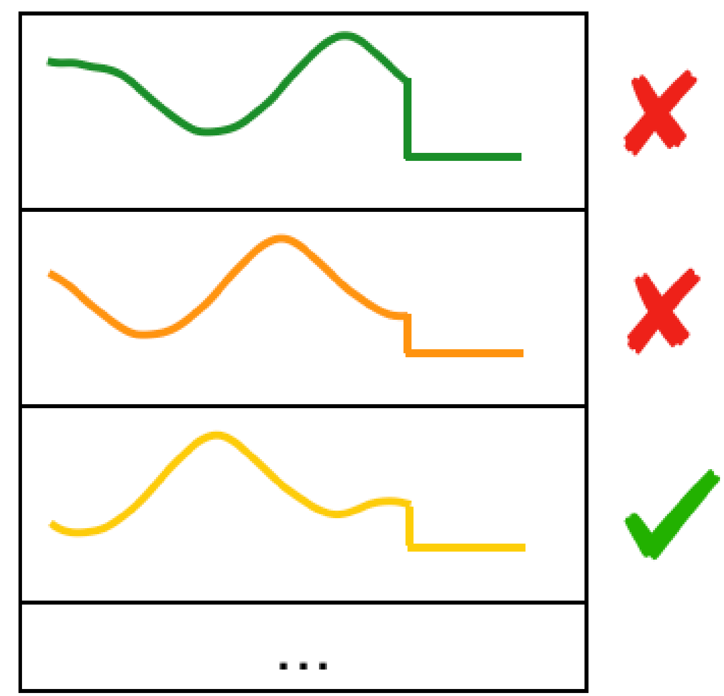
$LB_{Kim}FL(Q, \boldsymbol{S}_i) < bsf$	$LB_{Keogh}EC(Q, \boldsymbol{S}_i) < bsf$	$LB_{Keogh}EQ(\boldsymbol{S}_i, Q) < bsf$
True	False	False
True	True	False
True	True	True
...	...	...

Pruning clearly dissimilar candidates

Parallelizable ☺
Auto-vectorizable☺

$LB_{Kim}FL(Q,\boldsymbol{S}_i) < bsf$		$LB_{Keogh}EC(Q,\boldsymbol{S}_i) < bsf$		$LB_{Keogh}EQ(\boldsymbol{S}_i,Q) < bsf$		
True	$\wedge$	False	$\wedge$	False	$=$	**False**
True	$\wedge$	True	$\wedge$	False	$=$	**False**
True	$\wedge$	True	$\wedge$	True	$=$	**True**
...		...		...		...

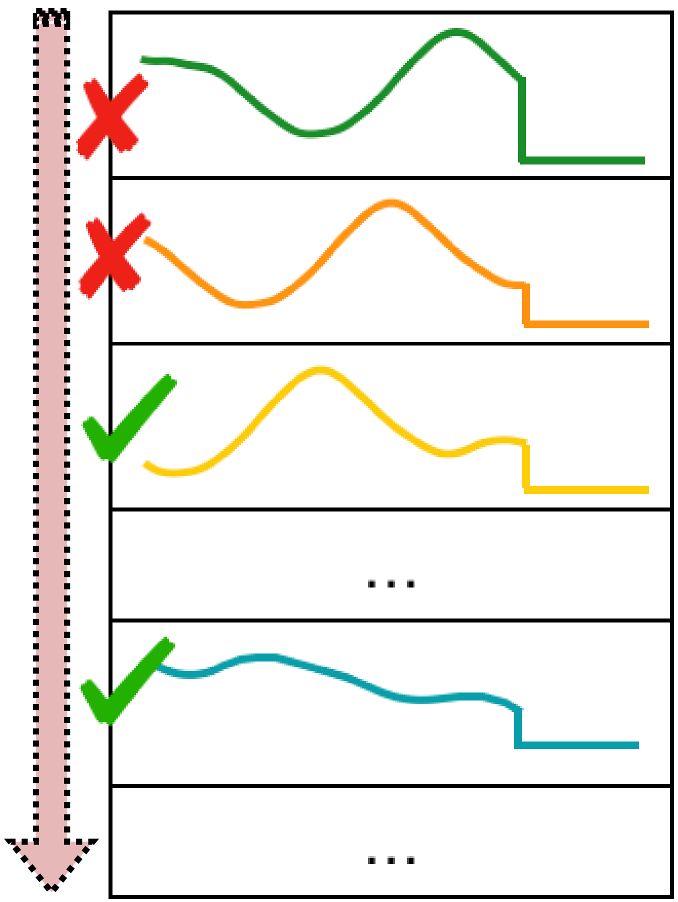
$\boldsymbol{S} \in \mathbb{R}^{N\times(n+pad)}$

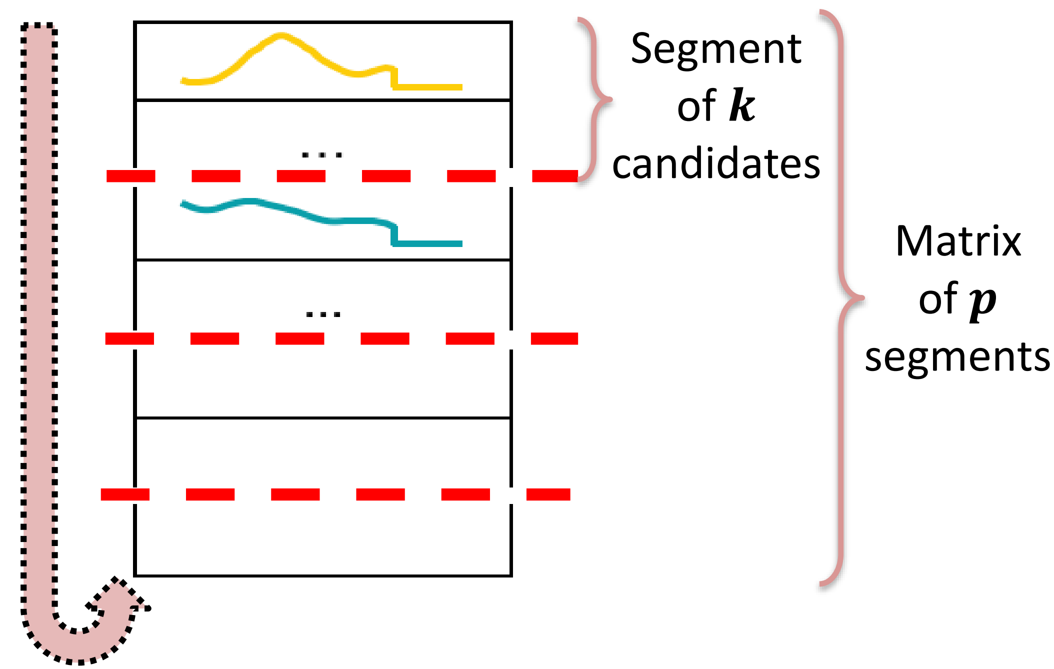


Matrix of promising candidates

Serial ☹

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





Balancing of threads load:

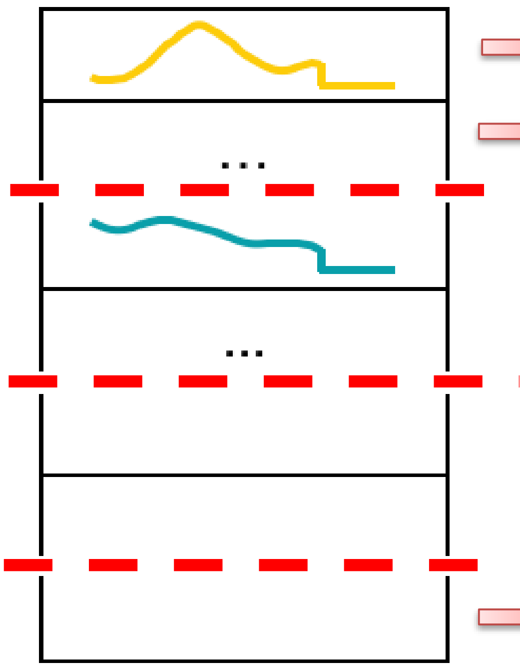
$\boldsymbol{p} \ll m$ is number of threads

$\boldsymbol{k}$ is parameter (e.g. $10, 100, 1\,000, \ldots$)

Computing best-so-far

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

Parallelizable ☺
Not auto-vectorizable ☹



$bsf_1 \leftarrow DTW(C_1, Q)$

$bsf_2 \leftarrow DTW(C_2, Q)$

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

$$bsf \leftarrow min\left(bsf_{prev}, \min_{1 \le i \le k \cdot p} bsf_i\right)$$

Partly auto-vectorizable ☺

Computing 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]
}
```

Not auto-vectorizable ☹
but could be rewritten
with auto-vectorizable part ☺

Computing 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], )
   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]
}
```

Auto-vectorizable part ☺

Not auto-vectorizable part ☹

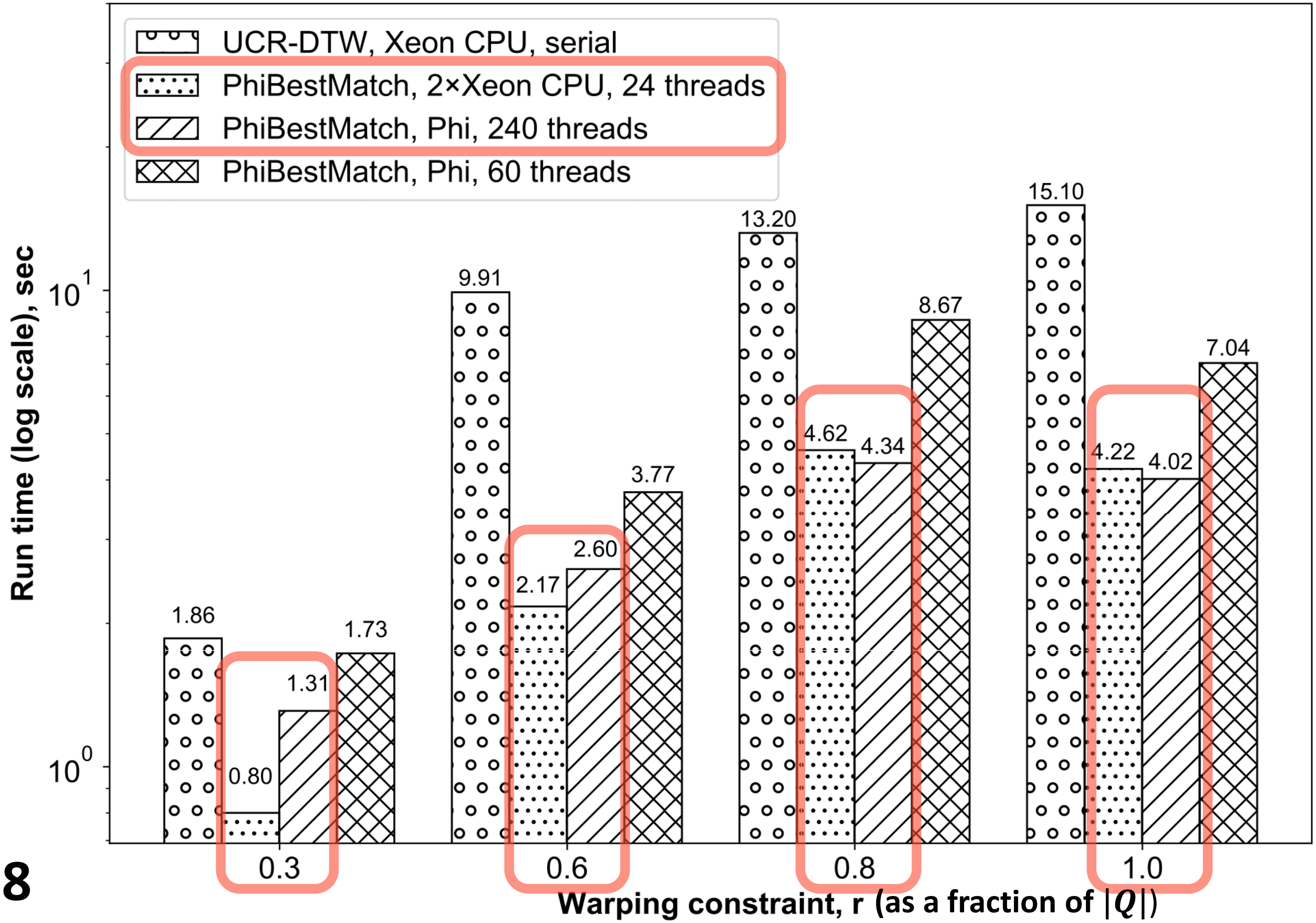
Experiments

- Hardware

Feature \ Device	2×Intel Xeon X5680	Intel Xeon Phi SE10X
# of physical cores	2×6	61
Hyper threading factor	2	4
# of logical cores	24	244
Frequency, GHz	3.33	1.1
Peak performance, TFLOPS	0.371	1.076

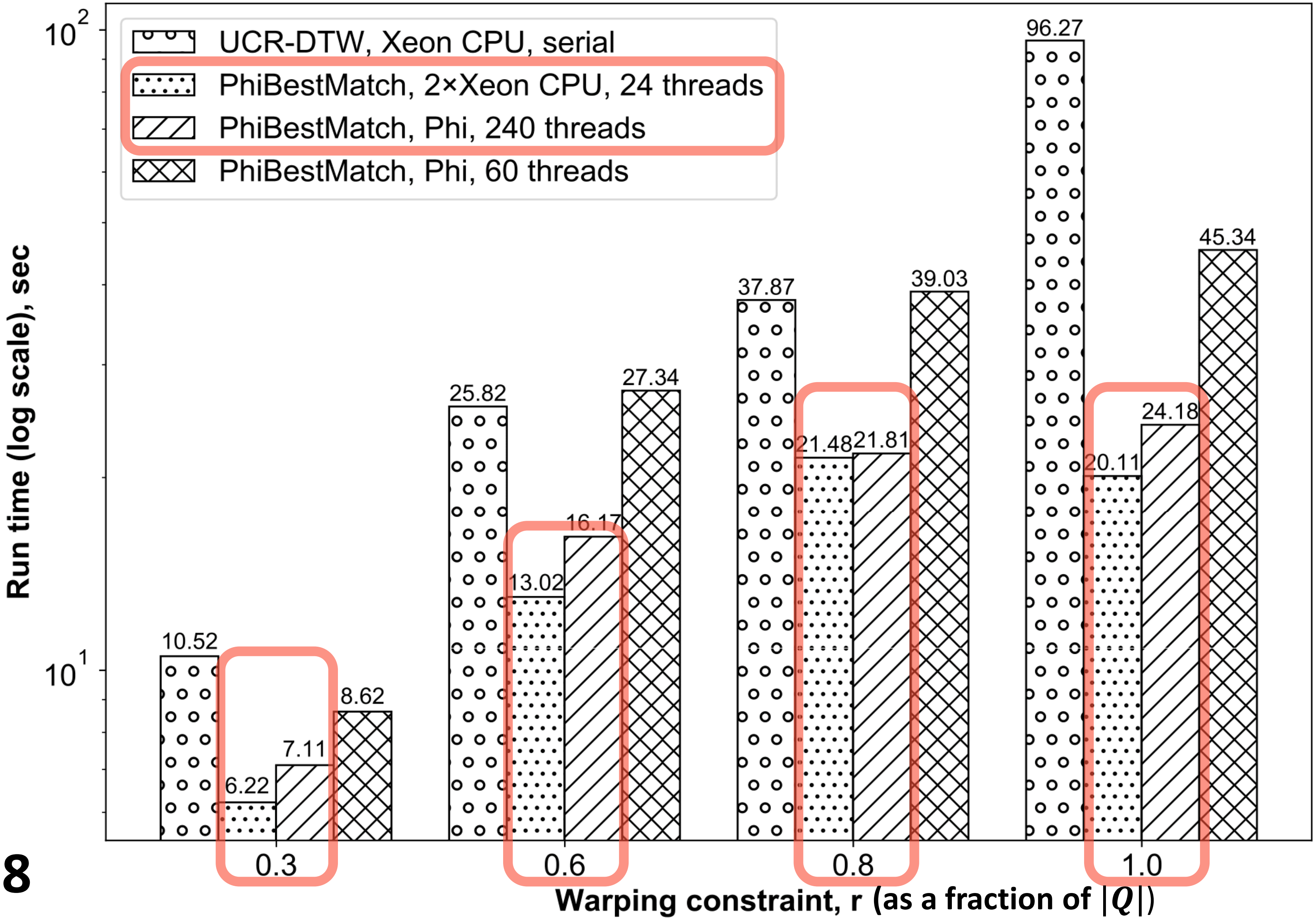
- Datasets
  - Random walk (synthetic): $|T|=10^6$
  - EPG (entomology): $|T|=2.5*10^5$
- Measures
  - Performance, sec.
  - Speedup, $s(p) = {}^{t_1}/_{t_p}$
  - Parallel efficiency, $e(p) = {}^{s(p)}/_{p}$

Performance: Random Walk dataset



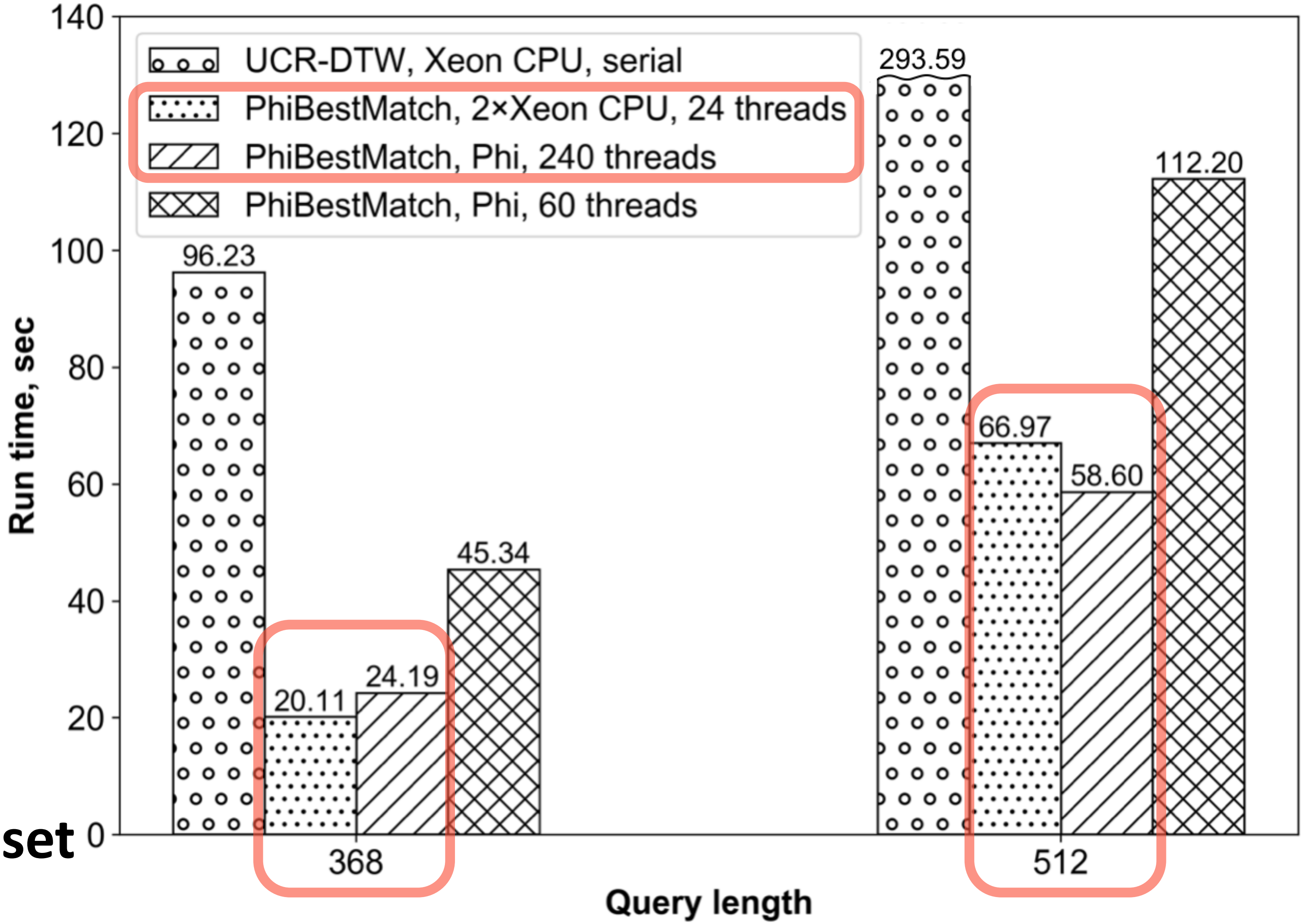
$|Q|$=128

Performance: EPG dataset



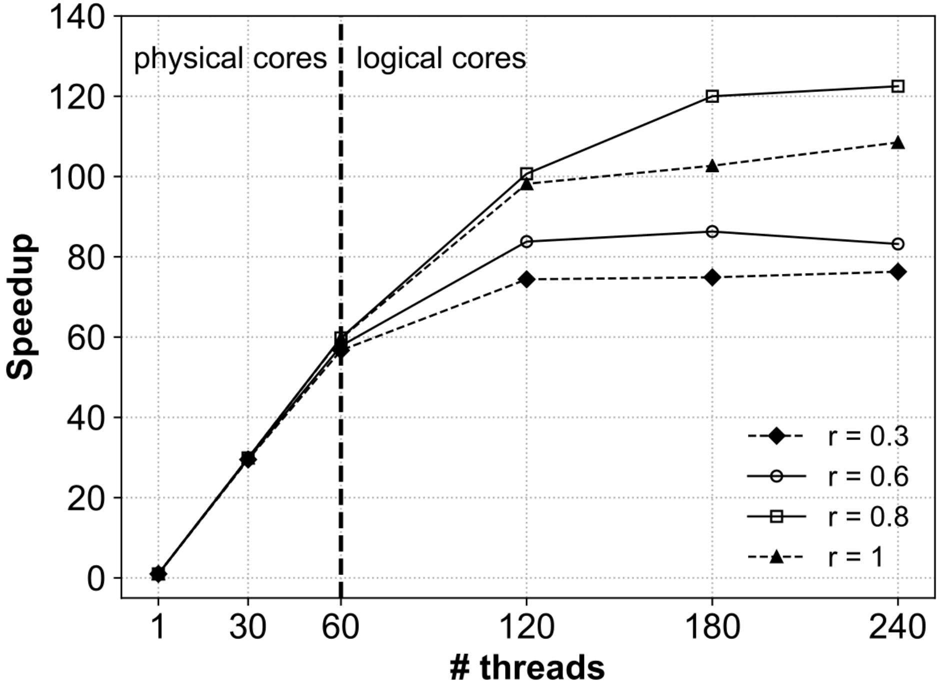
$|Q|$=368

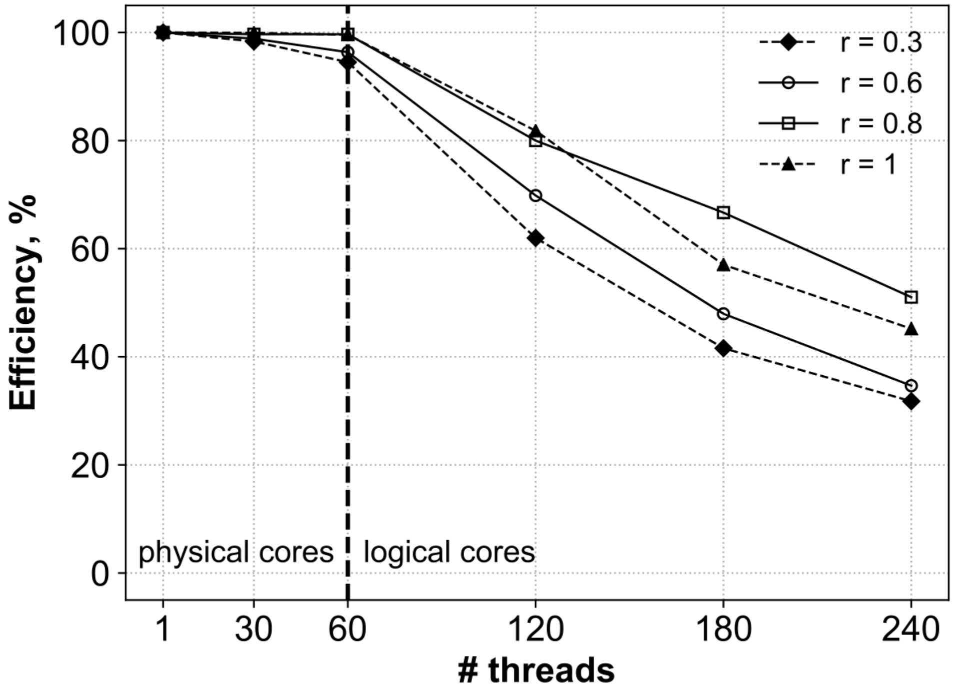
Performance: impact of query length



EPG dataset
r=1.0

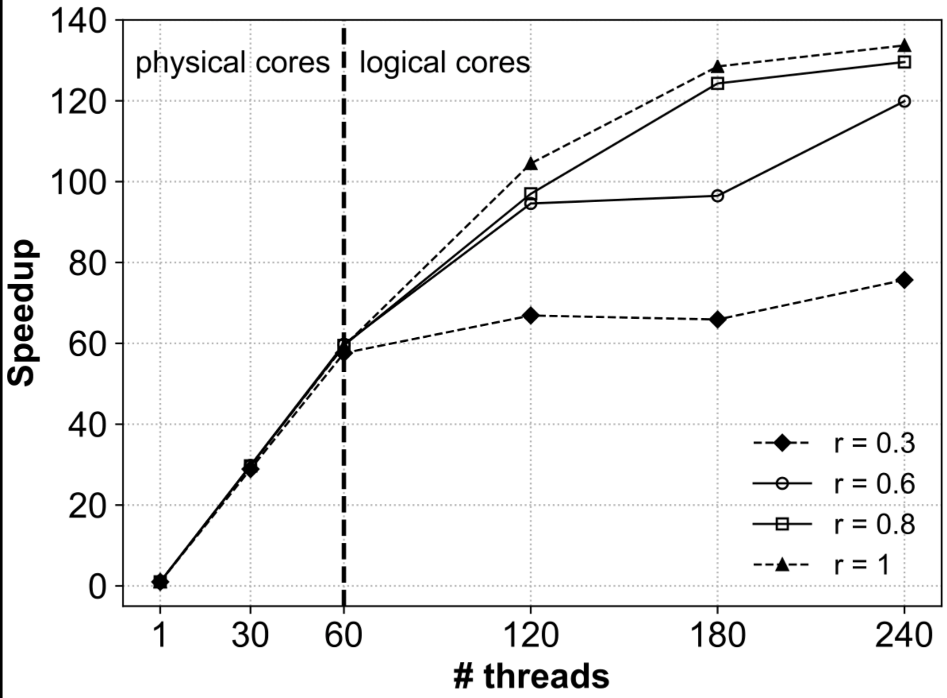
Scalability: Random Walk dataset

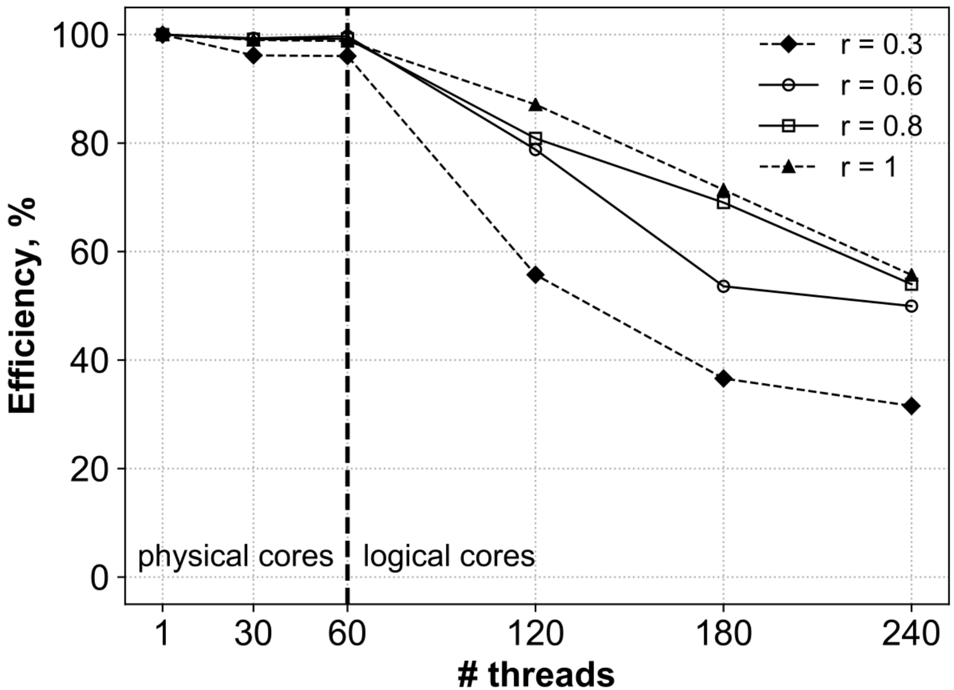




r denotes a fraction of $|Q|$

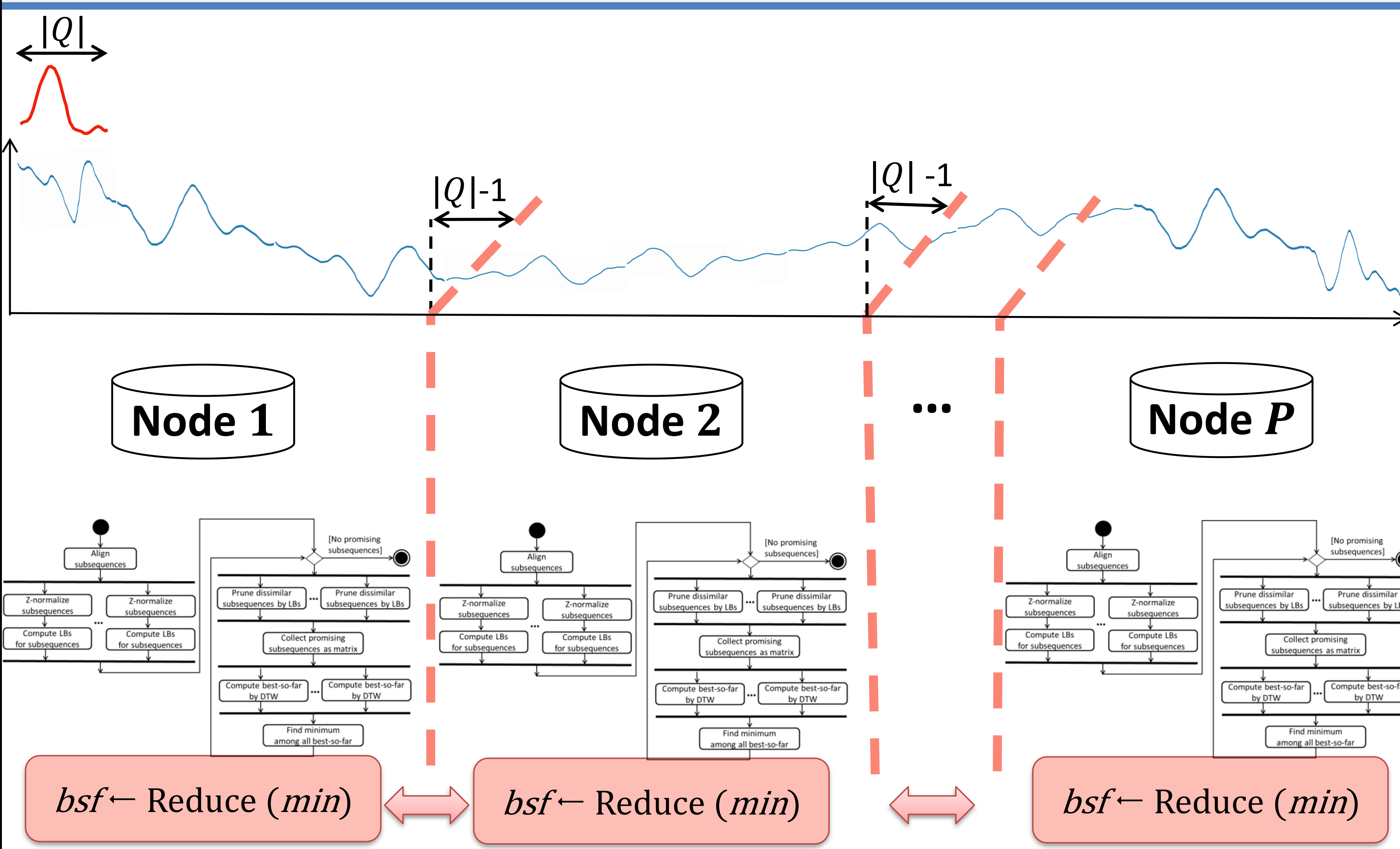
Scalability: EPG dataset





***r* denotes a fraction of |Q|**

Further work: cluster version



Conclusions

- We have proposed a novel parallel algorithm for subsequence similarity search for Intel many-core systems
- We have conducted experiments on synthetic and real data that show good scalability of our algorithm

Thank you for paying attention! Questions?

Mikhail Zymbler

mzym@susu.ru