Алгоритм поиска диссонансов временного ряда

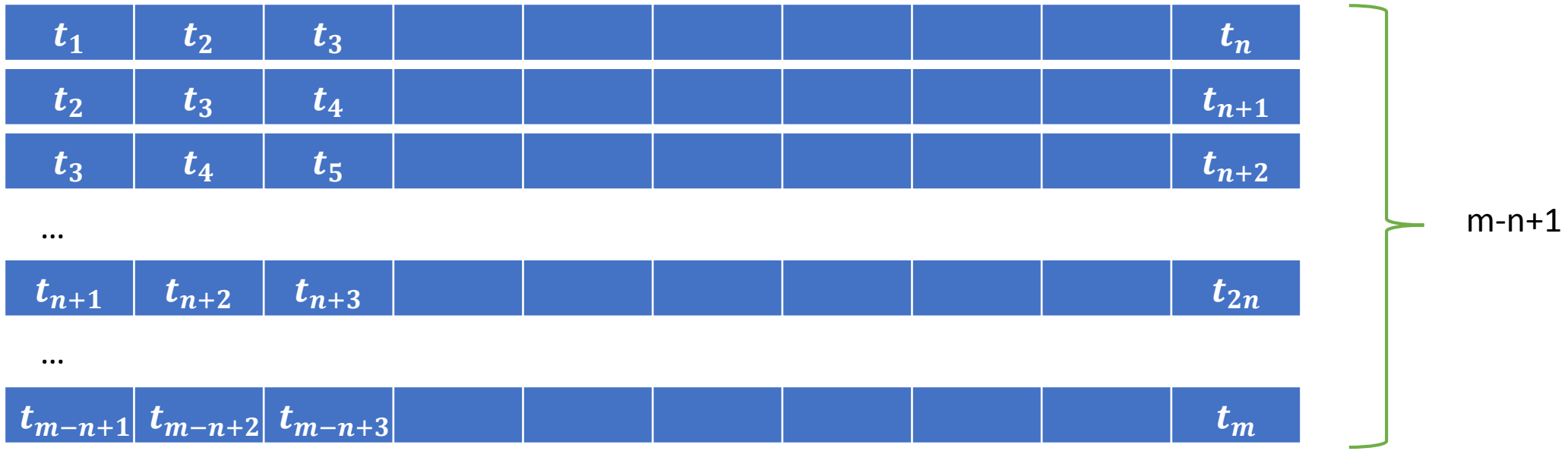
Алгоритм:

1. Подготовка (выбор эвристики) – подбор порядка подачи подпоследовательностей, при котором возможно быстро отбрасывать неподходящие подпоследовательности.
2. Поиск диссонансов – перебор упорядоченных подпоследовательностей временного ряда с поиском наибольшего расстояния до ближайшего соседа

Входные данные

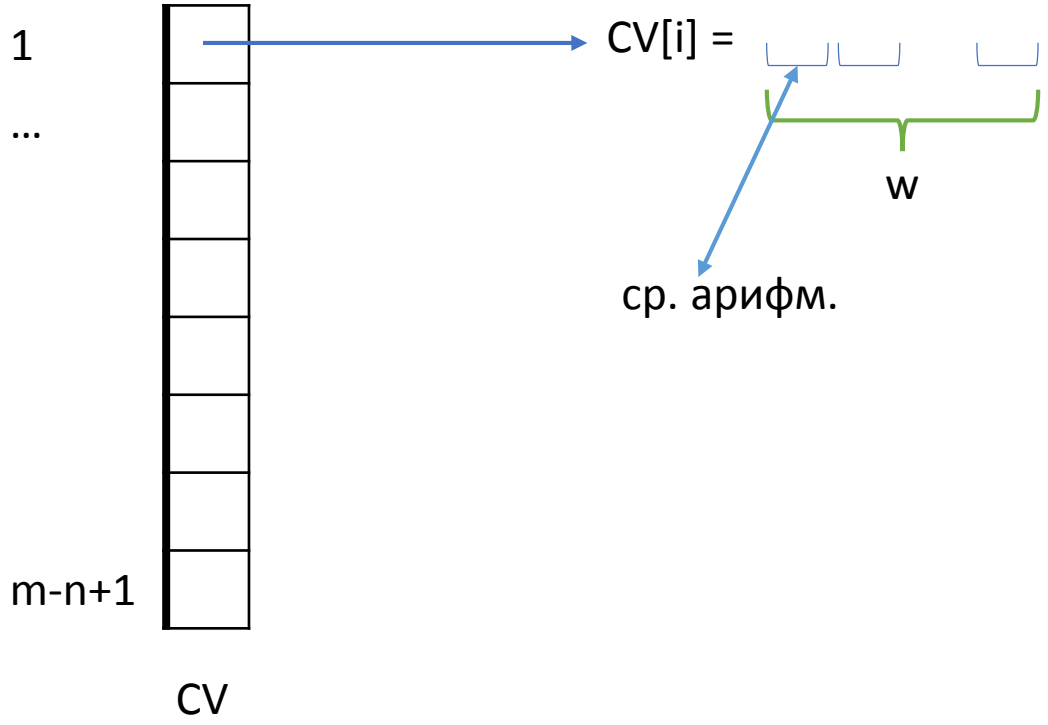
- T – временной ряд ($t_i, 1 \leq i \leq m$)
- m – длина ряда
- n – длина подпоследовательности
- C – множество подпоследовательностей
- w – длина слова (в SAX аппроксимации подпоследовательностей), $1 \leq w \leq n, n \; mod \; w = 0$
- Lookup table (LT) – для точек разделения в SAX

Хранение подпоследовательностей временного ряда



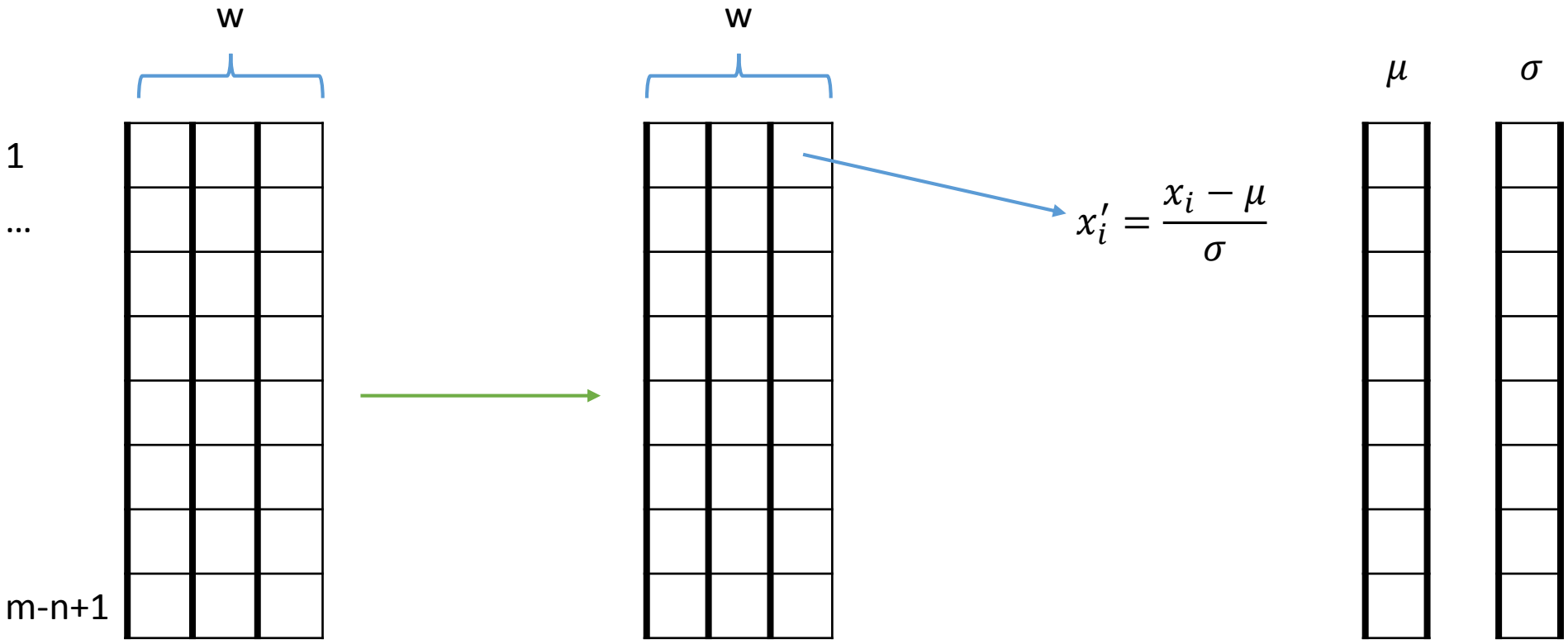
1. Подбор эвристики

1.1 Кусочная аппроксимация (PAA-представление)



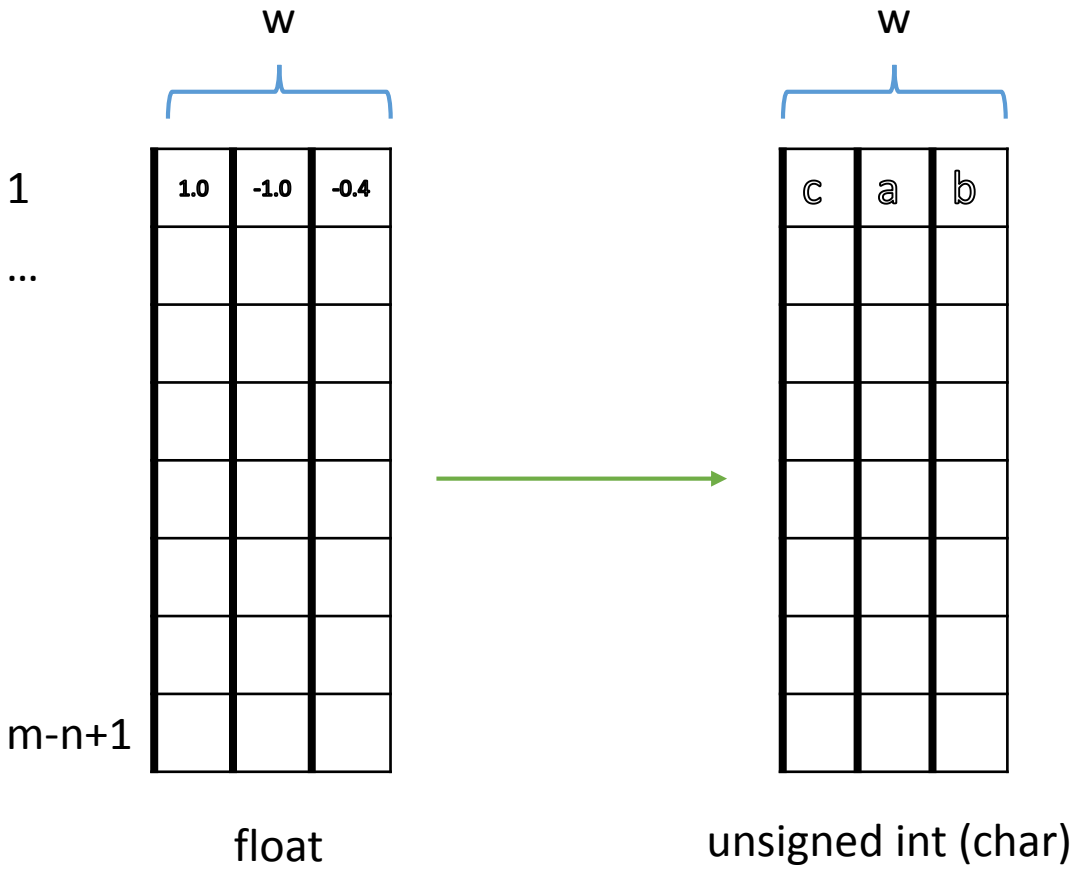
1. Подбор эвристики

1.2 z-нормализация PAA-представления



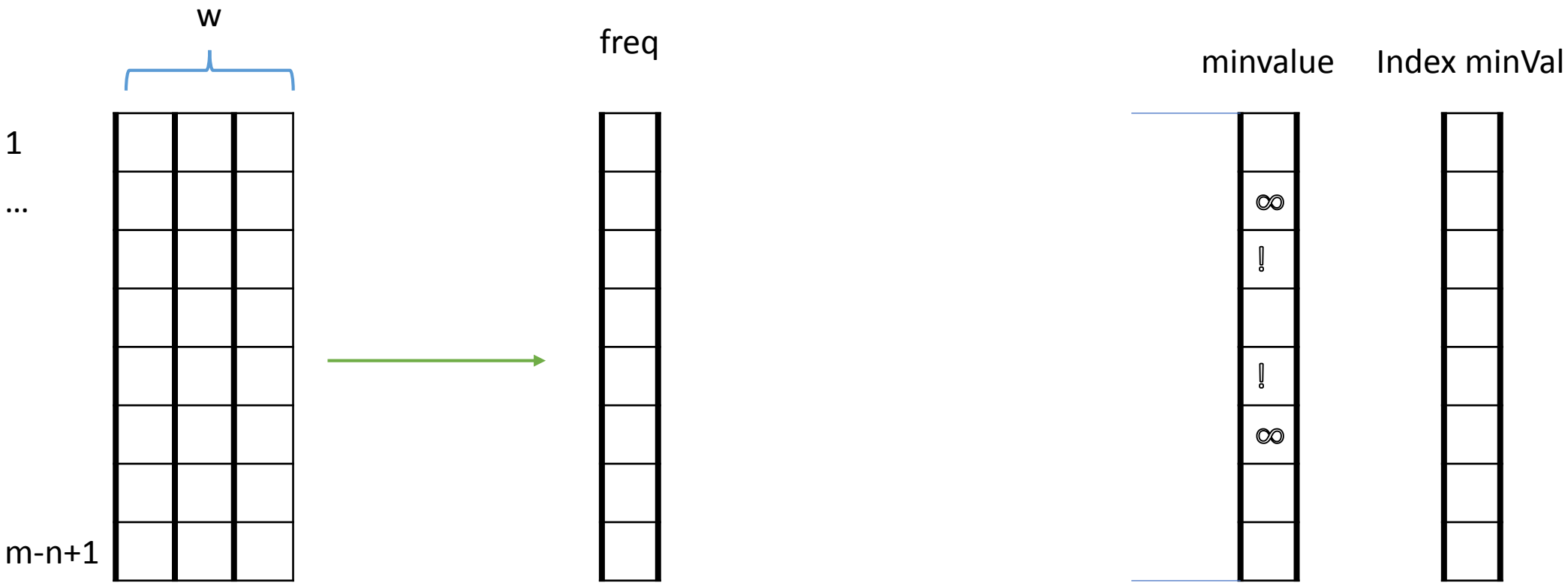
1. Подбор эвристики

1.3 Кодирование с помощью lookup table



1. Подбор эвристики

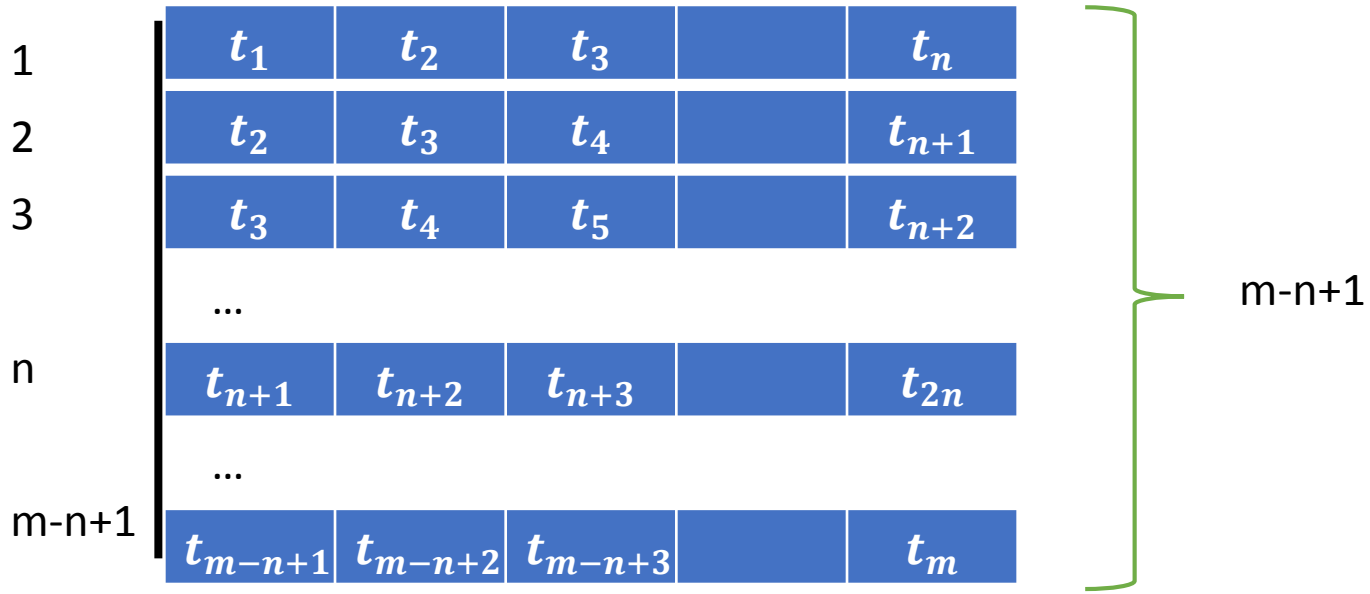
1.4 Подсчет частот, нахождение мин. значения



Упорядочивание множества подпоследовательностей

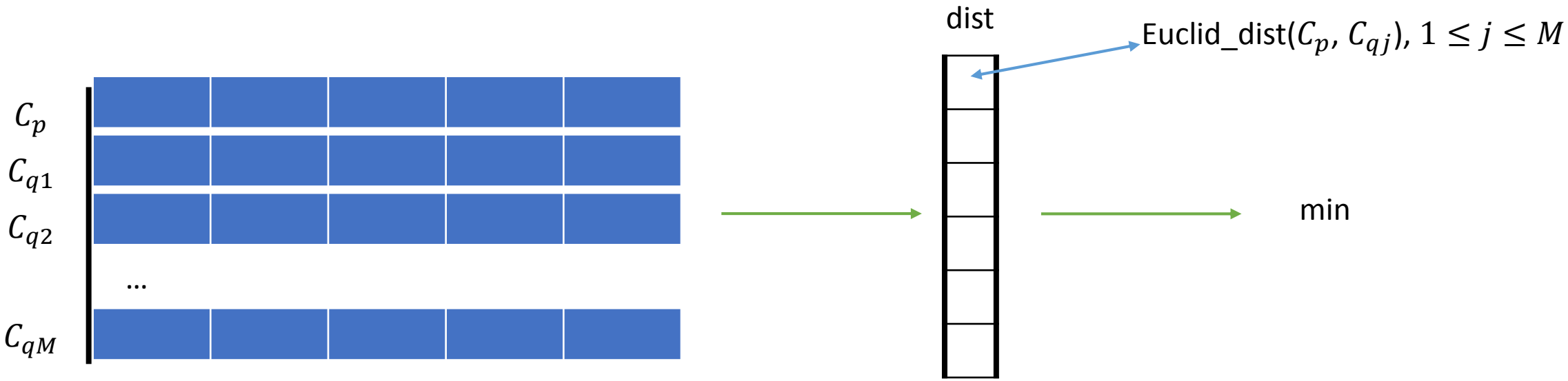
Index minVal

2
3
n



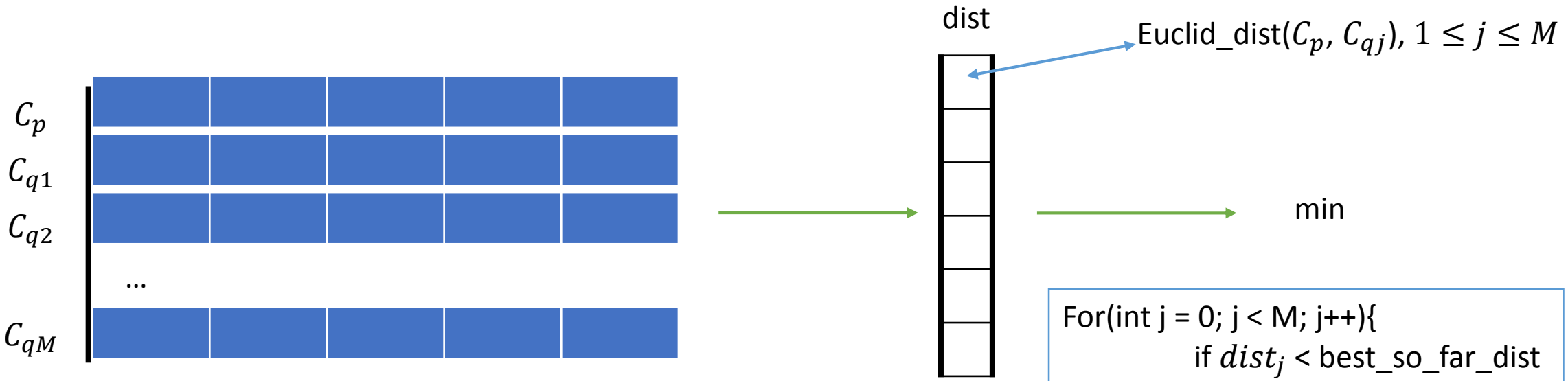
2. Поиск диссонансов

2.1 Нахождение оптимальной best_so_far_dist (для подпоследовательностей - предположительных диссонансов)



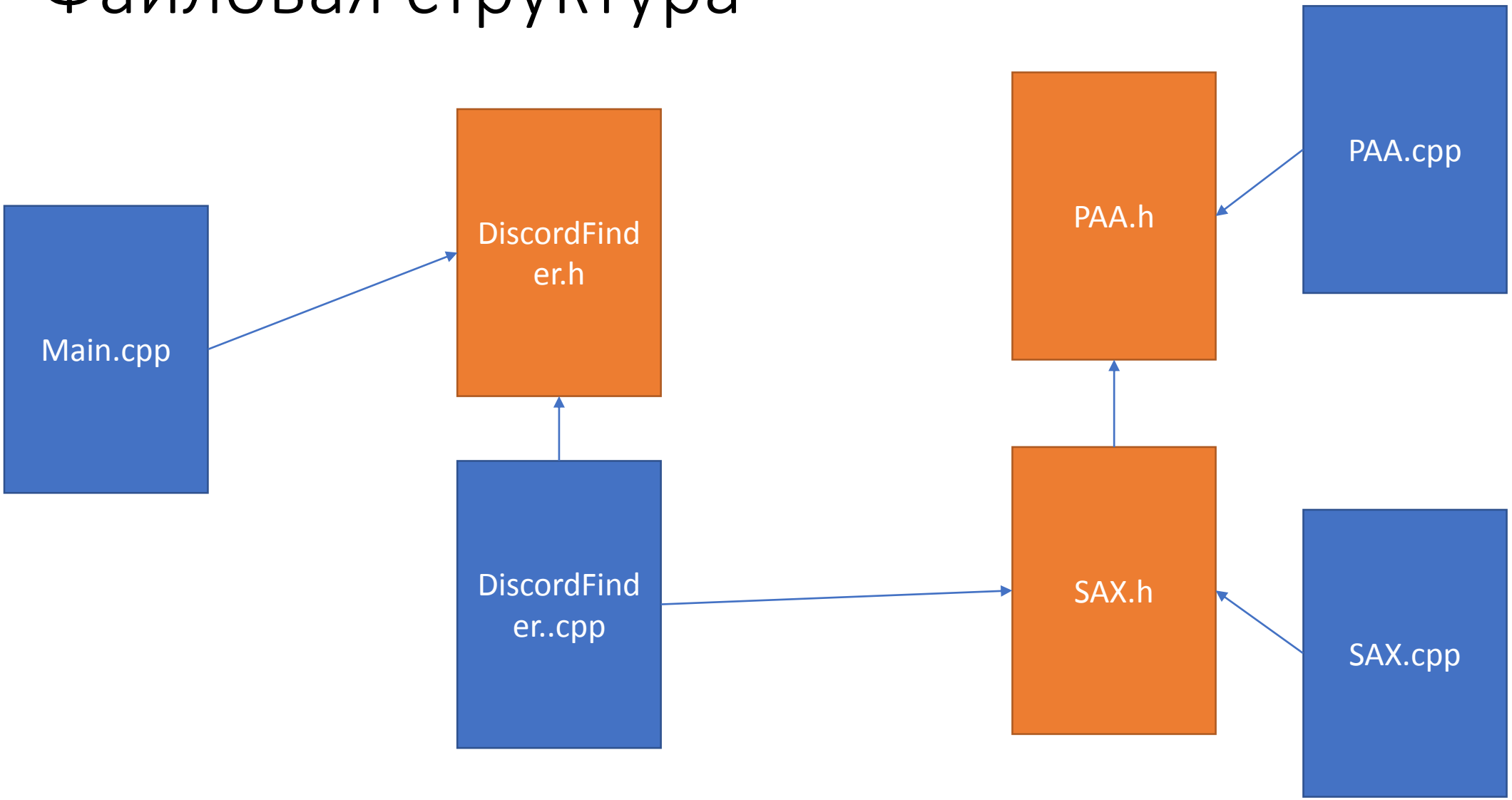
2. Поиск диссонансов

2.2 Нахождение расстояния до ближайшего соседа для оставшихся подпоследовательностей




```
For(int j = 0; j < M; j++){
        if dist_j < best_so_far_dist
                -> не подходит
}
If(min > best_so_far_dist){
        best_so_far_dist = min;
        best_so_far_pos = p;
}
```

Файловая структура



Модульная структура (TODO)

```
class PAA {
protected:
double * const vector;
int length;
int* counts;
public:
PAA(int len);
void apply(double * timeSeries, int length);
double * const getVector();
int getLength();
double getIndex(int i);
};
```

```
class SAX{
    size_t m_window_size;
    size_t m_string_size;
    size_t m_alphabet_size;
    double m_baseline_mean;
    double m_baseline_stdev;
    vector<double> m_cutpoints;
    bool m_trained;};

void saxify(Iter it, const Iter end,
vector<int> *syms);
```

```
Utils{
    inline void fill_cutpoints(size_t
alphabet_size, vector<double>
*cutpoints)
```