# EventEmbedding: Visual Segmentation and Stage Analysis of Event Sequence Data

# Introduction

- Background and Motivation
  - Event sequence data is very commonplace.
  - Stage analysis is beneficial in many application scenarios, such as electronic health data, user-behavior analysis…
  - Stage analysis is a challenging problem, because in most of the existing techniques, the duration of the stages are fixed, and the number of stages are pre-determined. However, we usually do not have those aprior information.

# Introduction

- The specific problem you want to address
  - Summarizing event sequences according to the evolution pattern of stages
  - Developing a system which allows dynamic stage modification through interaction
  - Visual comparing on different event occurrence patterns of the same stages
- Is it a new problem or old problem?
  - The idea of stage analysis is old, and especially commonplace in analyzing electronic health record. But traditional stage analysis requires a given number of stages and is not able to incorporate user's feedback into stage segmentation.
  - It is new to apply visualization technique to solve this problem

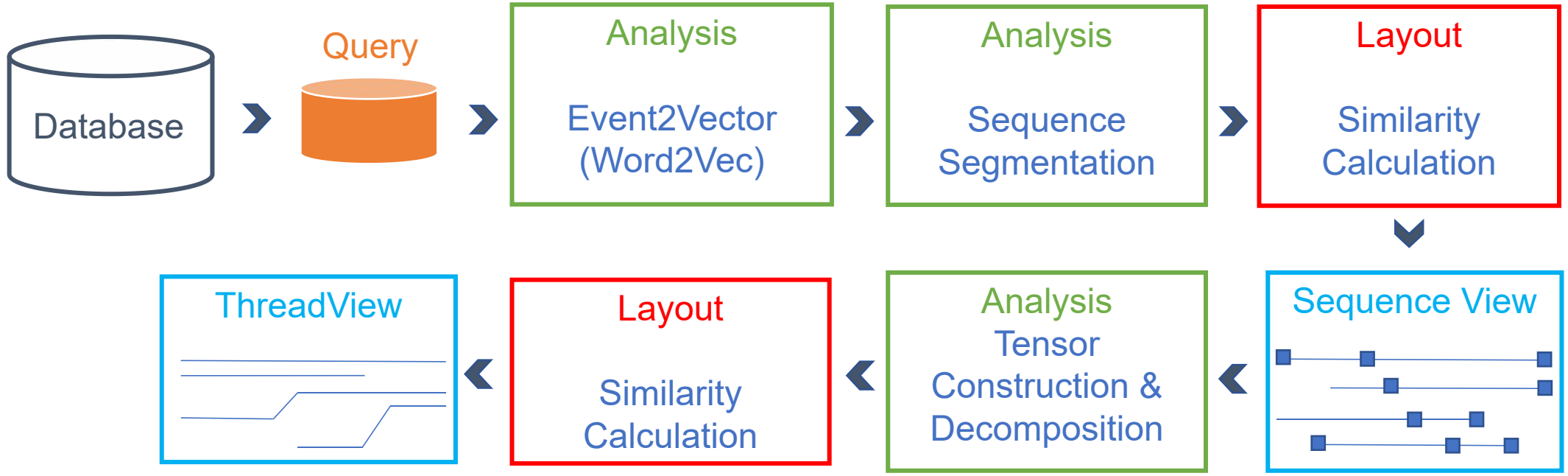
# Important Queries

- General patterns:
  - Diverse circumstances of stage distribution
    - Some sequences are divided into 7 stages, while others are divided into 5
    - Some individual directly jump into stage 3
    - Etc..
  - Different stage transition time
- Detailed patterns(Event related):
  - Key events for stage segmentation
  - Most frequent events in different stages
  - Various circumstances in the same stage
    - Different event frequency
  - Etc..

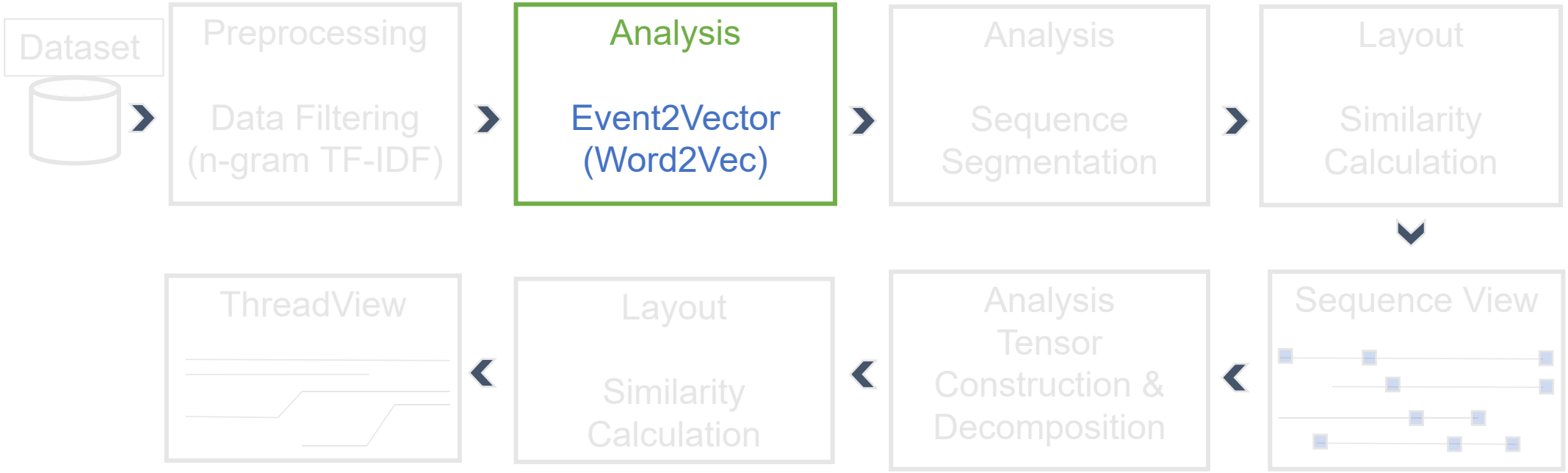
# Contributions

- A novel sequence segmentation method based on event embedding which can efficiently divide raw sequence into several stages.
- An interactive visualization system which aggregate subsequences in different stages and allow users to include a prior knowledge and make stage modification.

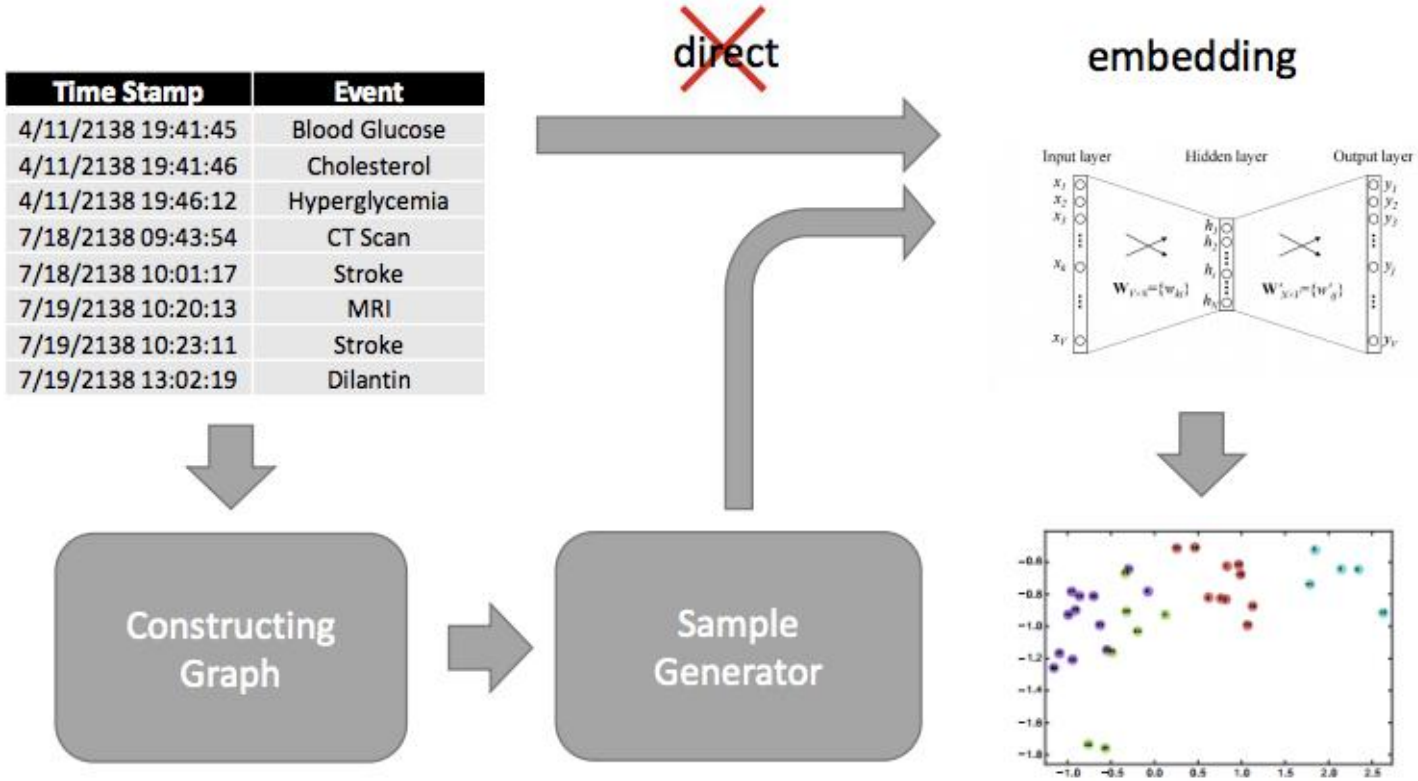
# System Pipeline



# System Pipeline



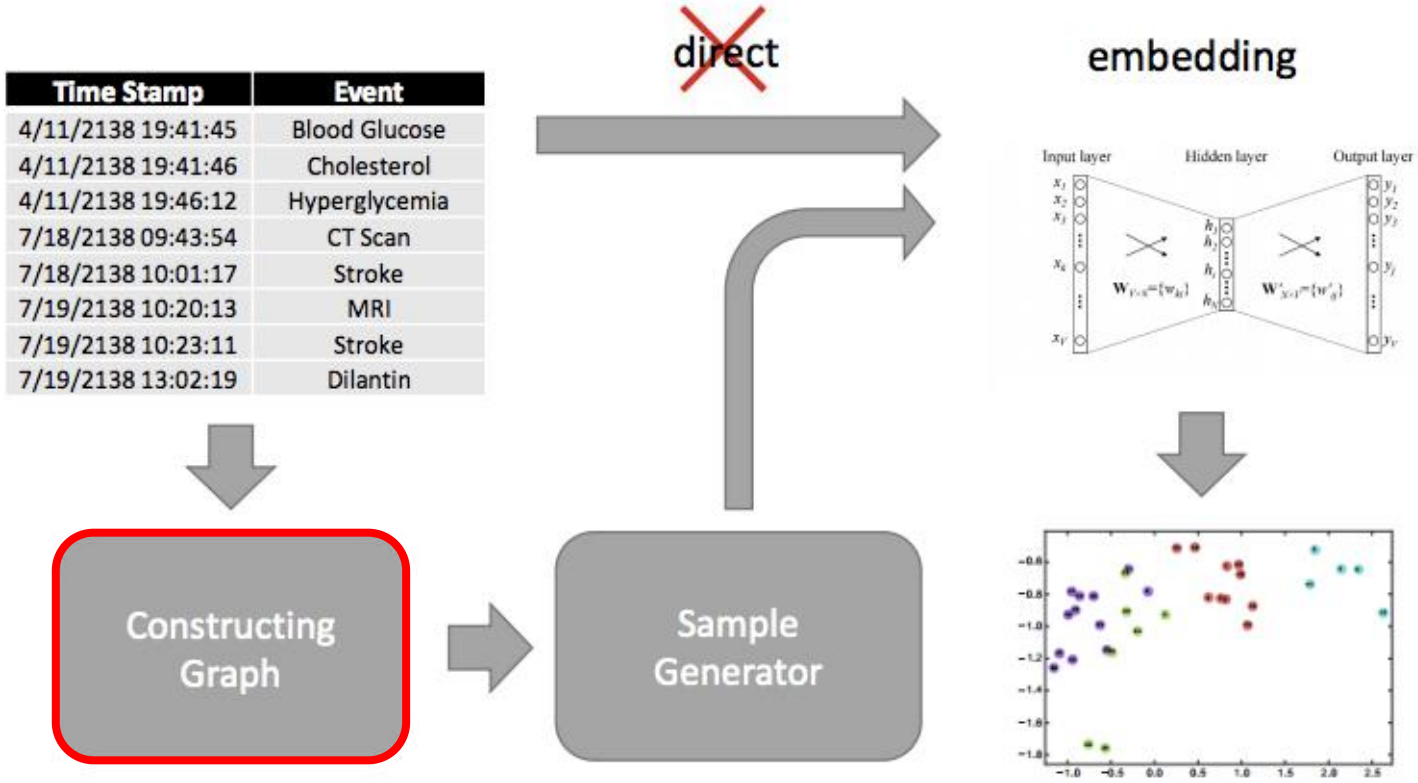
# Design Details:Event2Vec




| Time Stamp | Event |
|---|---|
| 4/11/2138 19:41:45 | Blood Glucose |
| 4/11/2138 19:41:46 | Cholesterol |
| 4/11/2138 19:46:12 | Hyperglycemia |
| 7/18/2138 09:43:54 | CT Scan |
| 7/18/2138 10:01:17 | Stroke |
| 7/19/2138 10:20:13 | MRI |
| 7/19/2138 10:23:11 | Stroke |
| 7/19/2138 13:02:19 | Dilantin |

**Fig. 2.** Framework of event2vec

# Design Details:Event2Vec




| Time Stamp | Event |
|---|---|
| 4/11/2138 19:41:45 | Blood Glucose |
| 4/11/2138 19:41:46 | Cholesterol |
| 4/11/2138 19:46:12 | Hyperglycemia |
| 7/18/2138 09:43:54 | CT Scan |
| 7/18/2138 10:01:17 | Stroke |
| 7/19/2138 10:20:13 | MRI |
| 7/19/2138 10:23:11 | Stroke |
| 7/19/2138 13:02:19 | Dilantin |

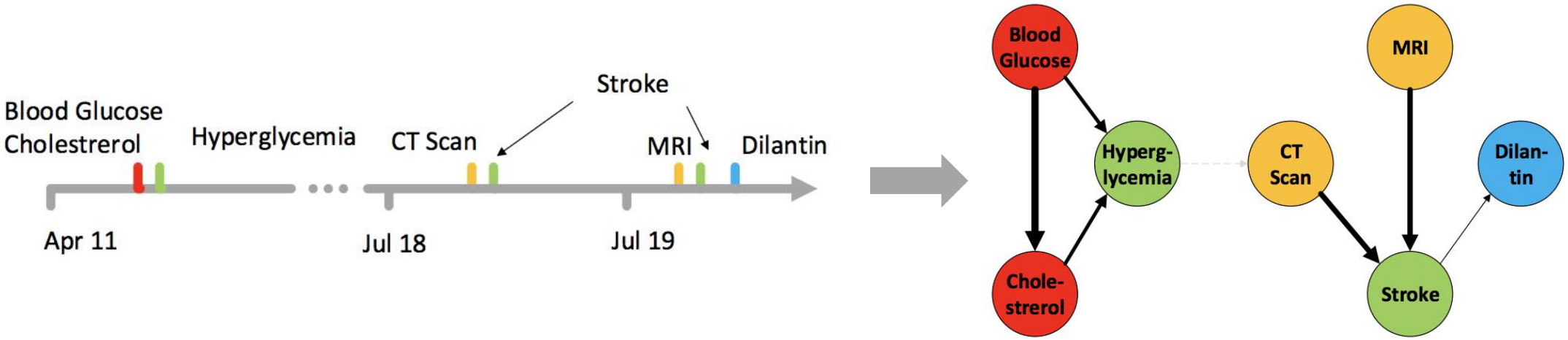
**Fig. 2.** Framework of event2vec

# Event2Vec: Event Connection Graph

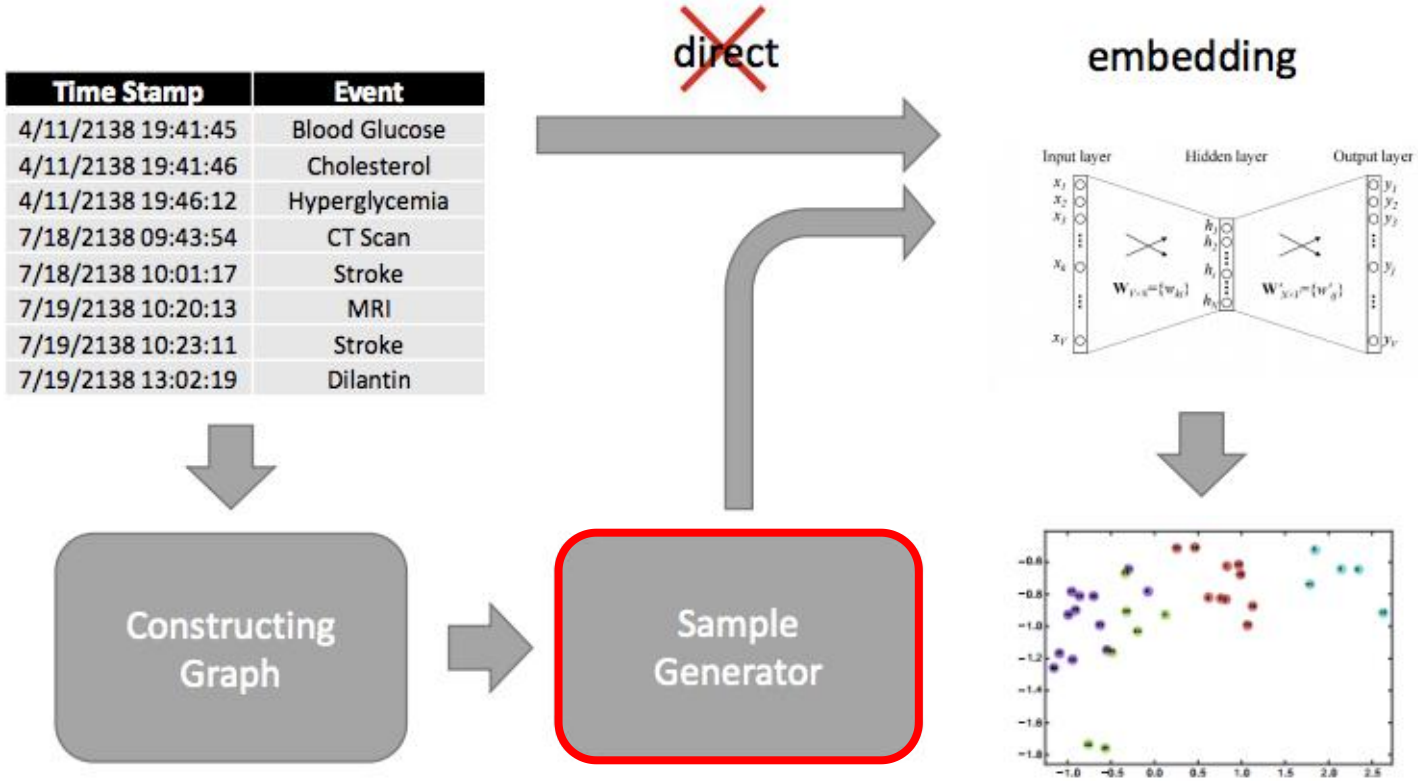
**Definition 4.** *(Event Connection Graph): Let $G =< V, E >$ be a directed weighted graph constructed from $S$, where each vertex $e_i$ in $V$ represents an unique event from Eve and edges in $E$ represent relations extracted from temporal event sequence. Weight of the directed edge from node $e_i$ to node $e_j$ is calculated by:*

$$G_{ij} = \sum_{1 \leq i < j \leq N} 1\{S(t_1) = e_i\} \wedge 1\{S(t_2) = e_j\} \delta(t_2 - t_1) \tag{1}$$

*where $1\{\cdot\}$ is set to 1 if its argument is true, and $\delta(x)$ is a function mapping time interval to the relation measurement of two events.*



# Design Details:Event2Vec




| Time Stamp | Event |
|---|---|
| 4/11/2138 19:41:45 | Blood Glucose |
| 4/11/2138 19:41:46 | Cholesterol |
| 4/11/2138 19:46:12 | Hyperglycemia |
| 7/18/2138 09:43:54 | CT Scan |
| 7/18/2138 10:01:17 | Stroke |
| 7/19/2138 10:20:13 | MRI |
| 7/19/2138 10:23:11 | Stroke |
| 7/19/2138 13:02:19 | Dilantin |

**Fig. 2.** Framework of event2vec

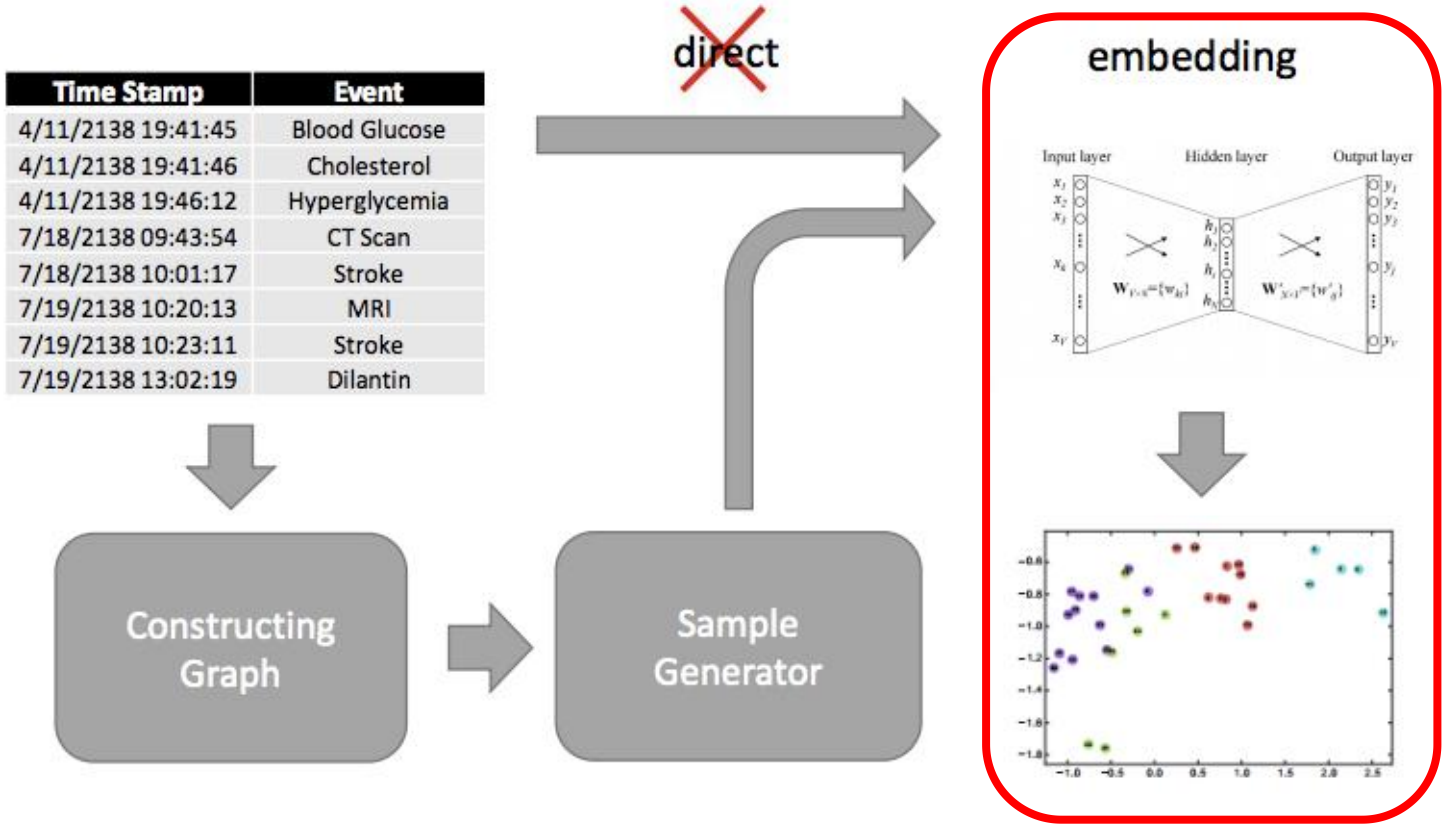
# Event2Vec: Sample Generator

- We incorporate Probabilistic Walk into the previously constructed event graph. It is a stochastic process starting with a single randomly chosen event e_i. And one of the neighbors of e_i denoted as e_j would be added based on a probability in direct proportion to the edge weight between e_i and e_j . The probability of choosing event e_j after event e_i is:

$$P(i \rightarrow j) = W_{ij} / \sum_{i} W_{ij}$$

- Then we apply standard Event2Vector method to the generated samples.

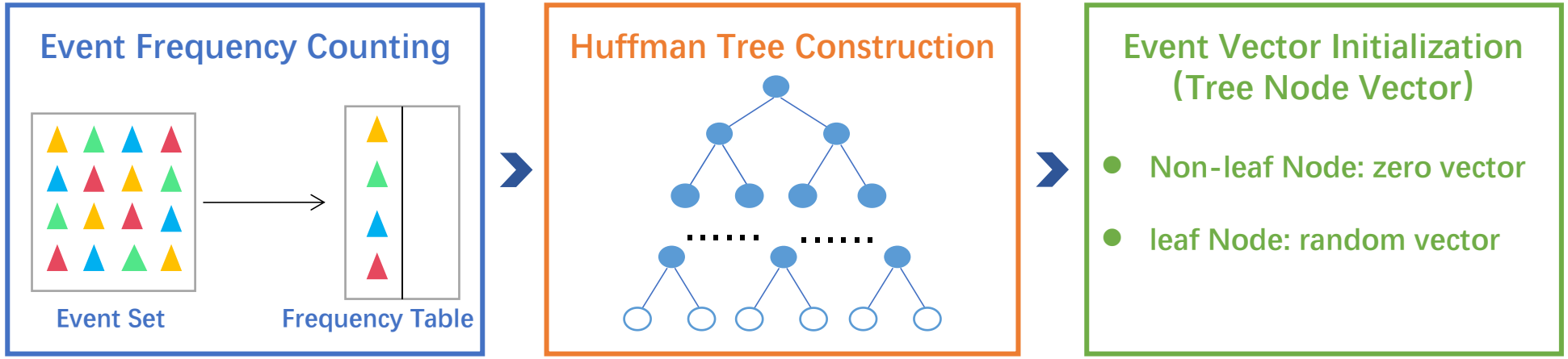
# Design Details:Event2Vec




| Time Stamp | Event |
|---|---|
| 4/11/2138 19:41:45 | Blood Glucose |
| 4/11/2138 19:41:46 | Cholesterol |
| 4/11/2138 19:46:12 | Hyperglycemia |
| 7/18/2138 09:43:54 | CT Scan |
| 7/18/2138 10:01:17 | Stroke |
| 7/19/2138 10:20:13 | MRI |
| 7/19/2138 10:23:11 | Stroke |
| 7/19/2138 13:02:19 | Dilantin |

**Fig. 2.** Framework of event2vec

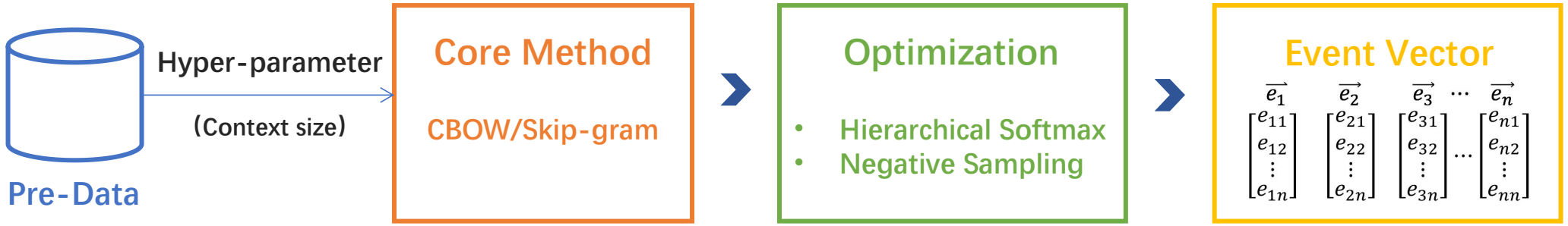
# Design Details: Event2Vec




Event Set: contains all events of sequences.

Huffman Tree: constructs by the frequency of event, leaf node represents event, non-leaf node generates by leaf nodes, its frequency equals to the sum of child nodes.The event with Lower frequency is farther away from the root node.

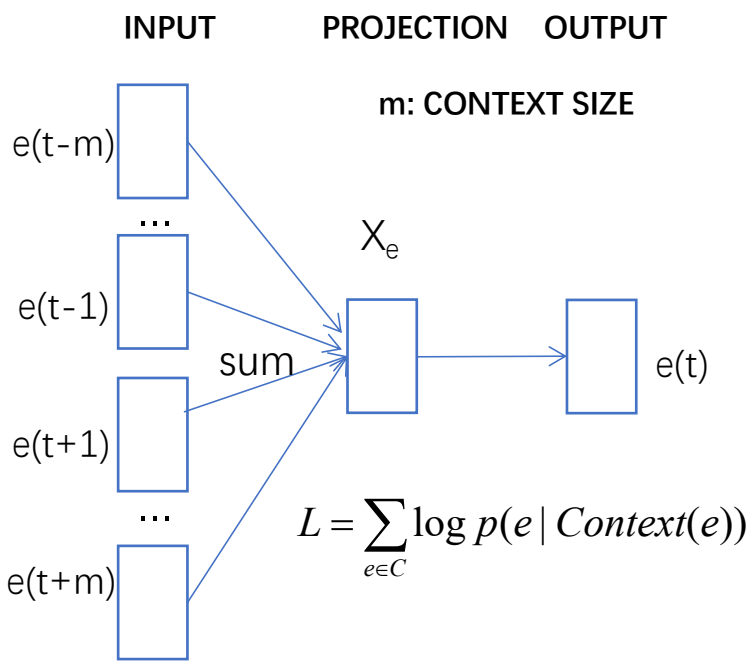
# Design Details: Event2Vec




- Huffman Code: 1 represents nagative sample, 0 represents positive sample.

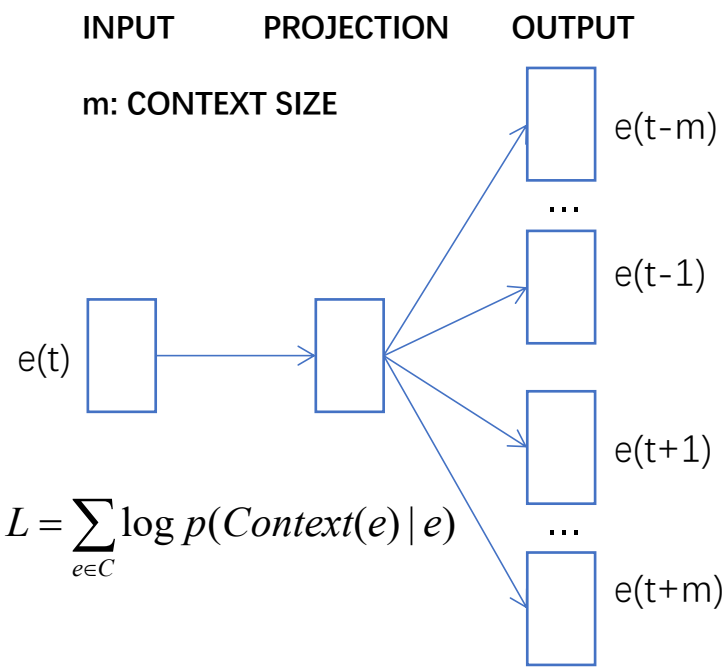
# Design Details: Event2Vec

- Core Method: maximizing log-likelihood function




$$L = \sum_{e \in C} \log p(e \mid Context(e))$$

CBOW

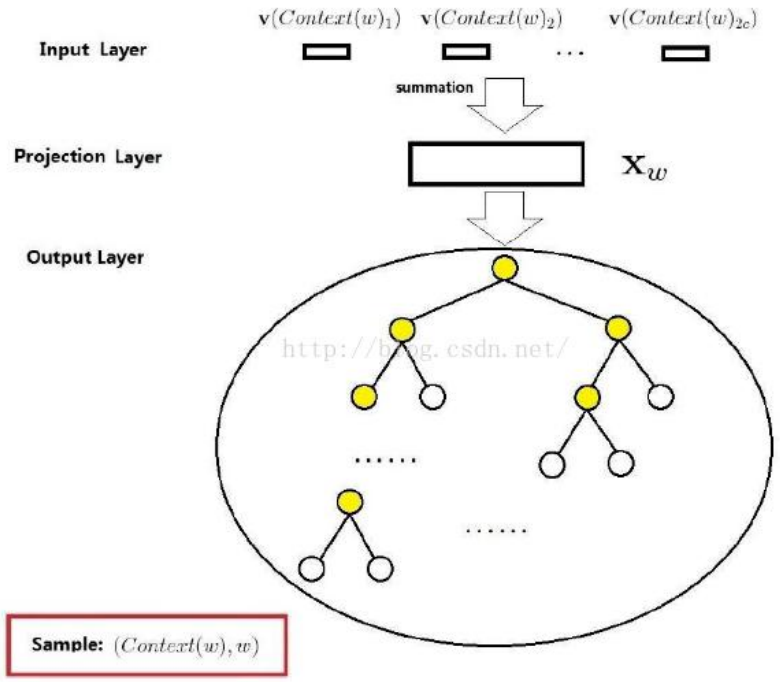



$$L = \sum_{e \in C} \log p(Context(e) \mid e)$$

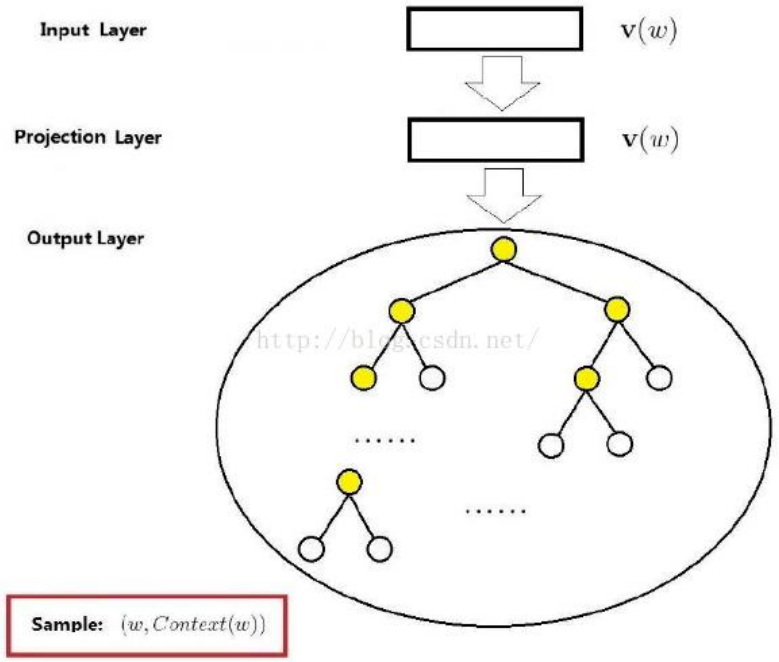
Skip-gram

# Design Details: Event2Vec

- Optimization: Hierarchical Softmax




CBOW




Skip-gram

**Huffman Calculation**

- CBOW

$$p(w \mid Context(w)) = \prod_{j=2}^{l^w} p(d_j^w \mid x_w, \theta_{j-1}^w)$$

- Skip-gram

$$p(Context(w) \mid w) = \prod_{u \in Context(w)} p(u \mid w)$$

$$p(u \mid w) = \prod_{j=2}^{l^u} p(d_j^u \mid v(w), \theta_{j-1}^u)$$

**Event Vector Updation**

- CBOW

$$v(\tilde{w}) = v(\tilde{w}) + \eta \sum_{j=2}^{l^w} \frac{\partial L(w, j)}{\partial x_w}, \tilde{w} \in Context(w)$$

- Skip-gram

$$v(w) = v(w) + \eta \sum_{u \in Context(w)} \sum_{j=2}^{l^w} \frac{\partial L(w, u, j)}{\partial v(w)}$$

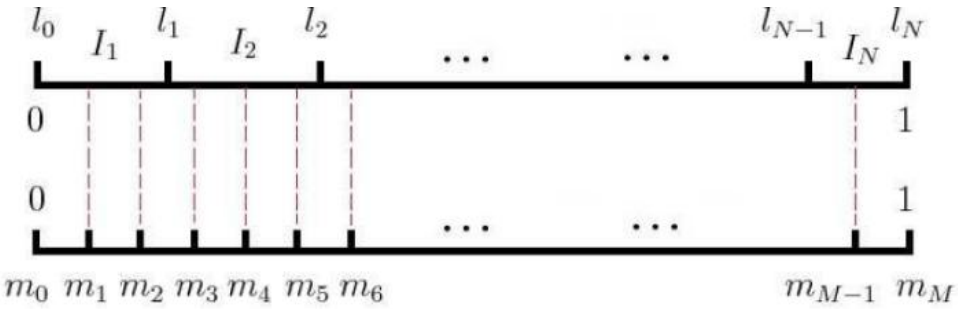
# Design Details: Event2Vec

- Optimization: Negative Sampling

  The length of Segment L is 1,equally divided into M segments.

$$\mathrm{len}(e) = \frac{frequency(e)^{3/4}}{\sum_{u \in E} frequency(u)^{3/4}}$$

smoothing strategy




As for the context of event e, e is positive sample, (context(e),e), others are negative sample.

- Maximizing likelihood function

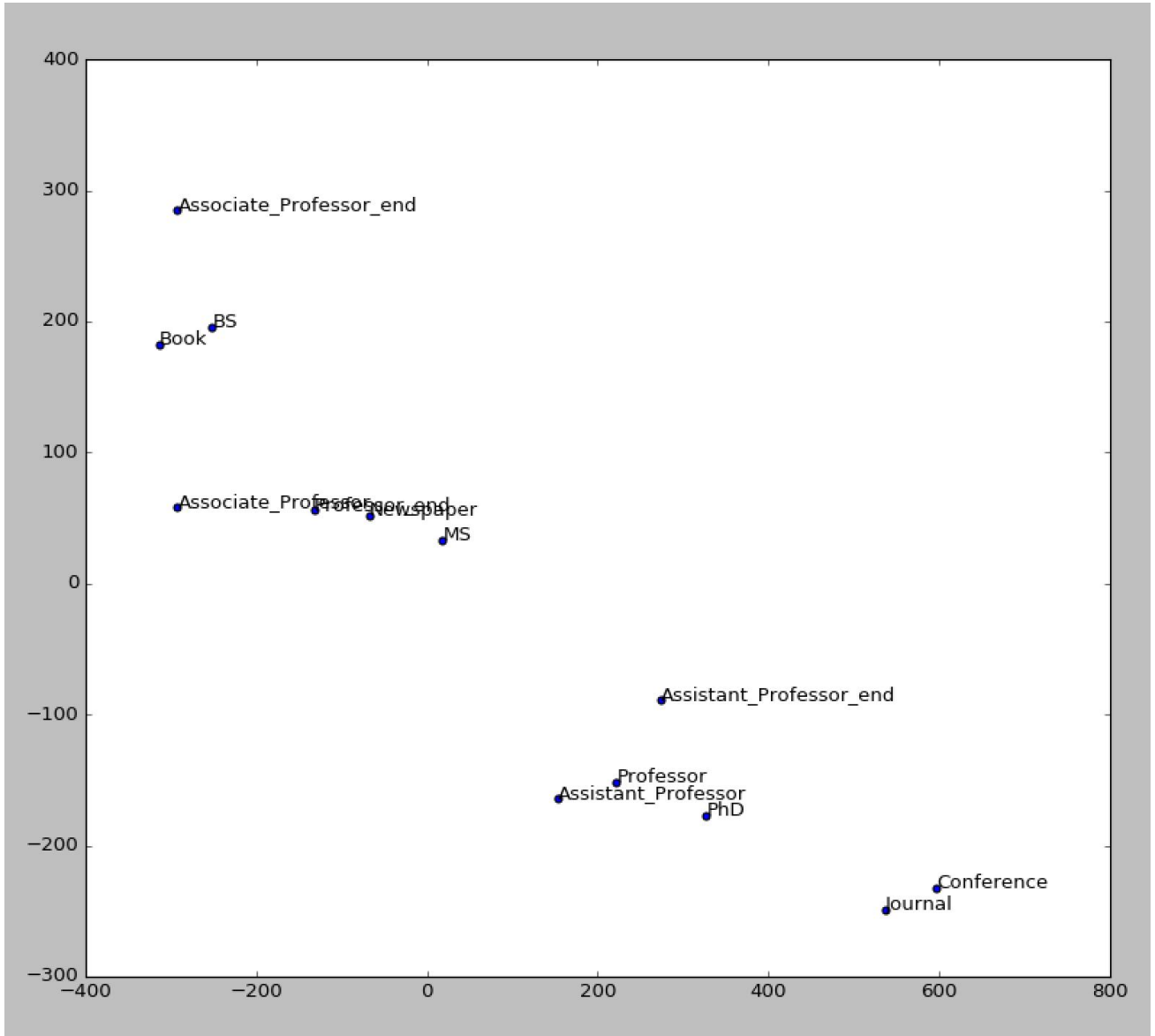
$$L = \sum_{i=0}^{neg} y_i \log(\sigma(x_{w_i}^T, \theta^{w_i})) + (1 - y_i)\log(1 - \sigma(x_{w_i}^T, \theta^{w_i}))$$

- Sampling Rule

  Randomly generate r integers from 1 to M-1, corresponding samples are the positions in the segment L pointed by r integers.

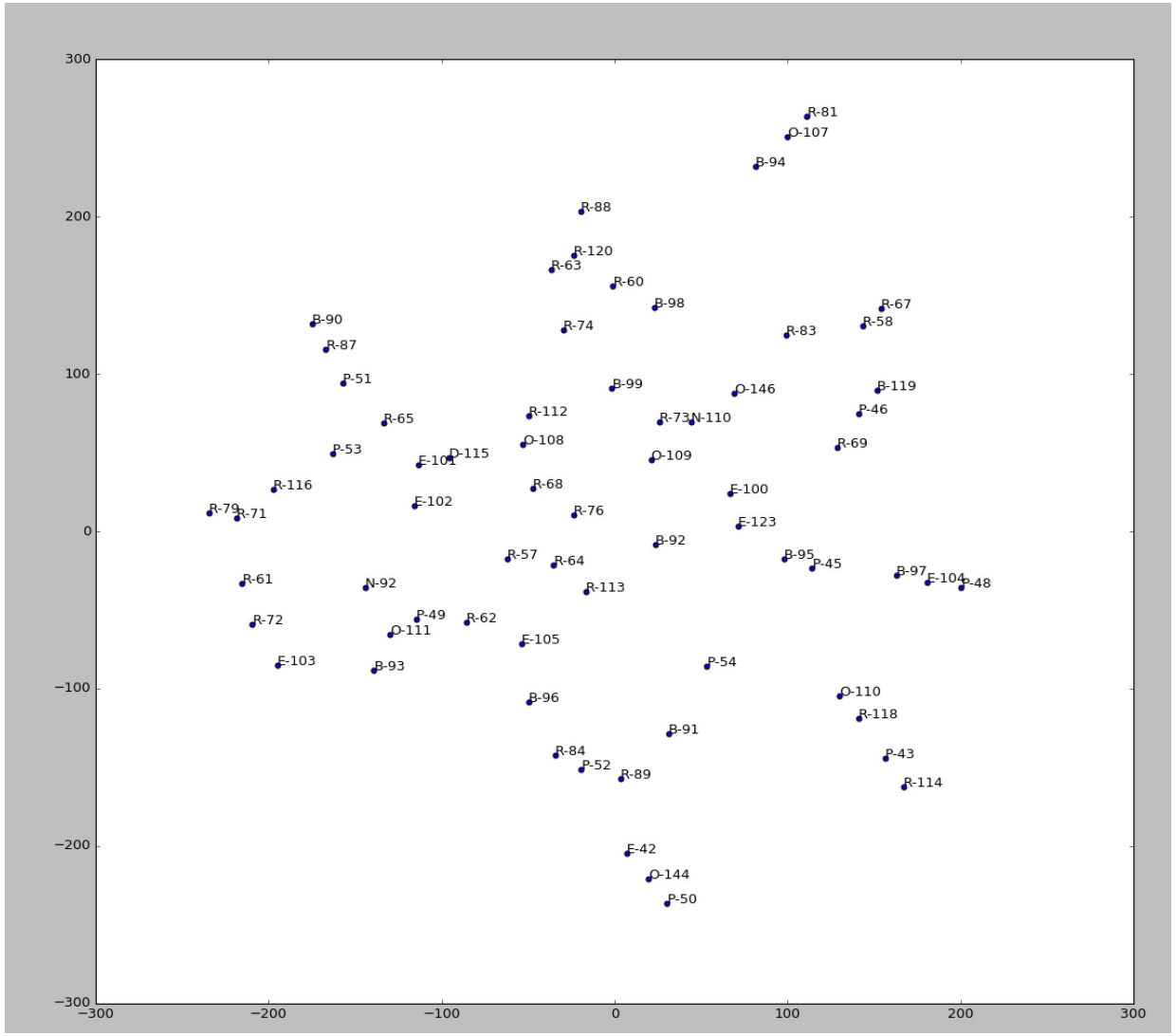
# Preliminary Results

- TSNE of Professor Dataset:



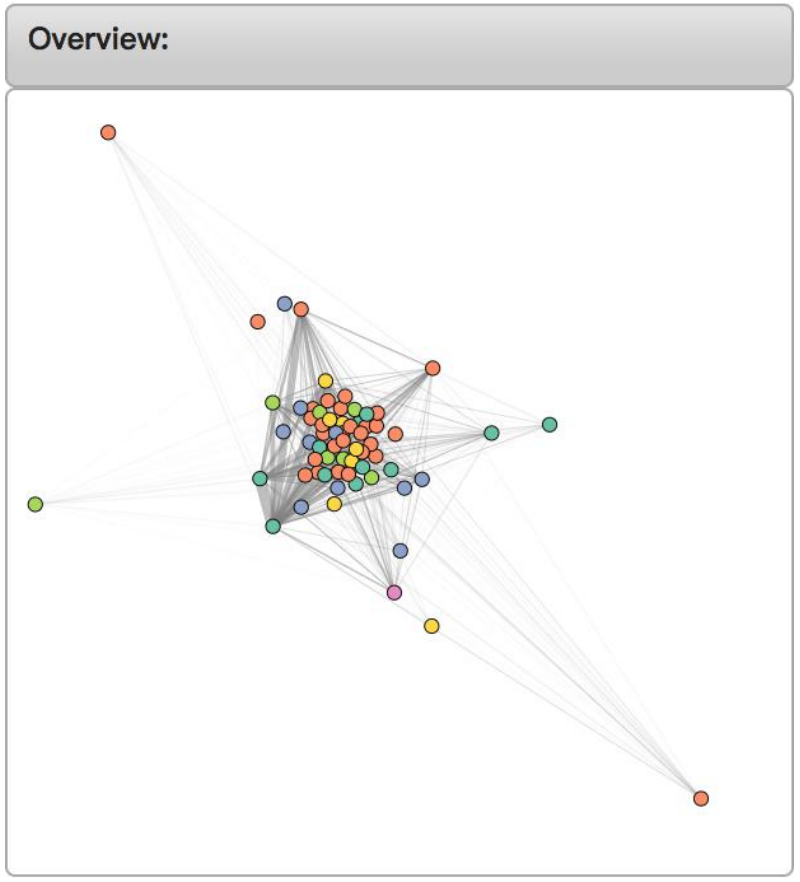
# Preliminary Results

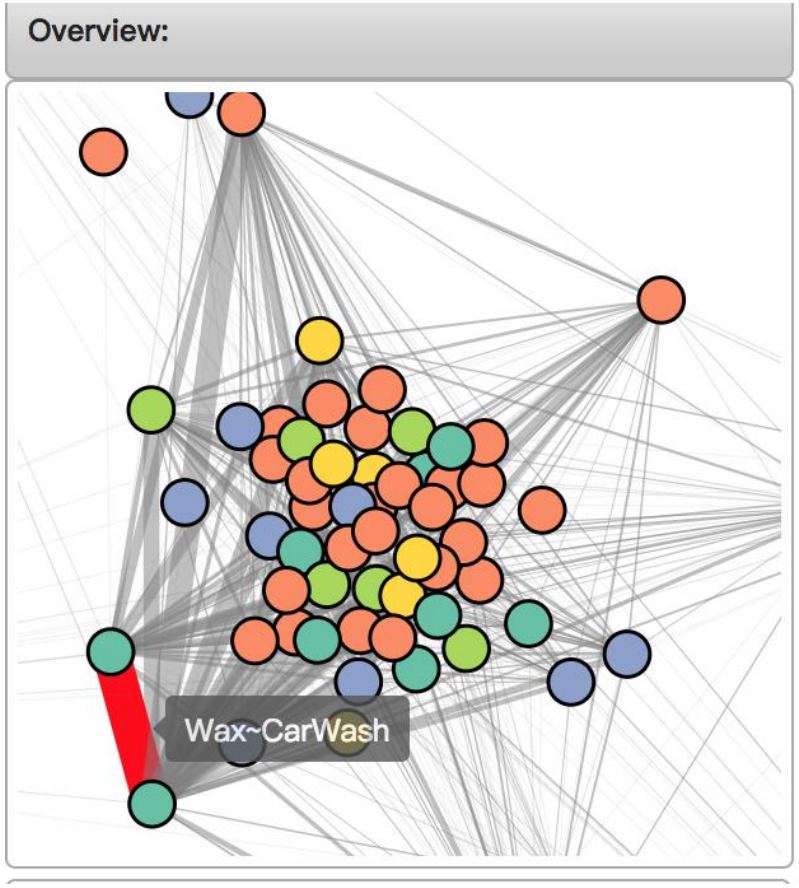
- TSNE of Car Maintenance Dataset:

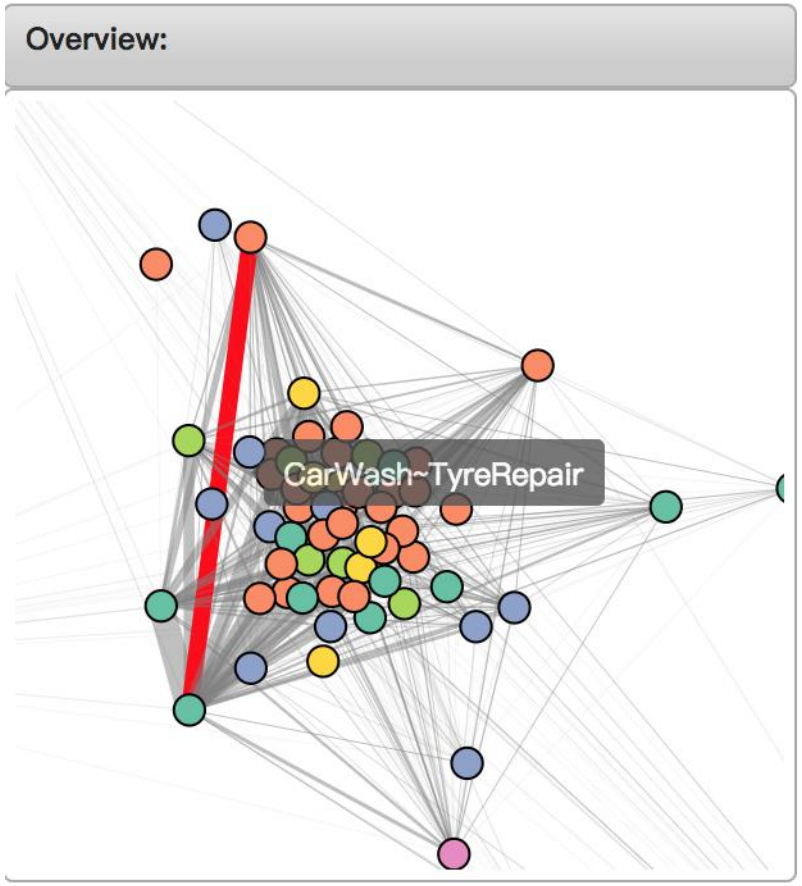


# Preliminary Results

- TSNE View of Car Maintenance Dataset:



# Design Details: Sequence Segmentation

- For different sequences

$$\begin{array}{ll} \mathbf{C_1} & e_1 \rightarrow e_3 \rightarrow e_7 \rightarrow \cdots \rightarrow e_{130} \\ \mathbf{C_2} & e_2 \rightarrow e_3 \rightarrow e_{10} \rightarrow \cdots \rightarrow e_{119} \\ & \vdots \\ \mathbf{C_m} & e_5 \rightarrow e_7 \rightarrow e_{11} \rightarrow \cdots \rightarrow e_{80} \end{array}$$

$$\Rightarrow$$

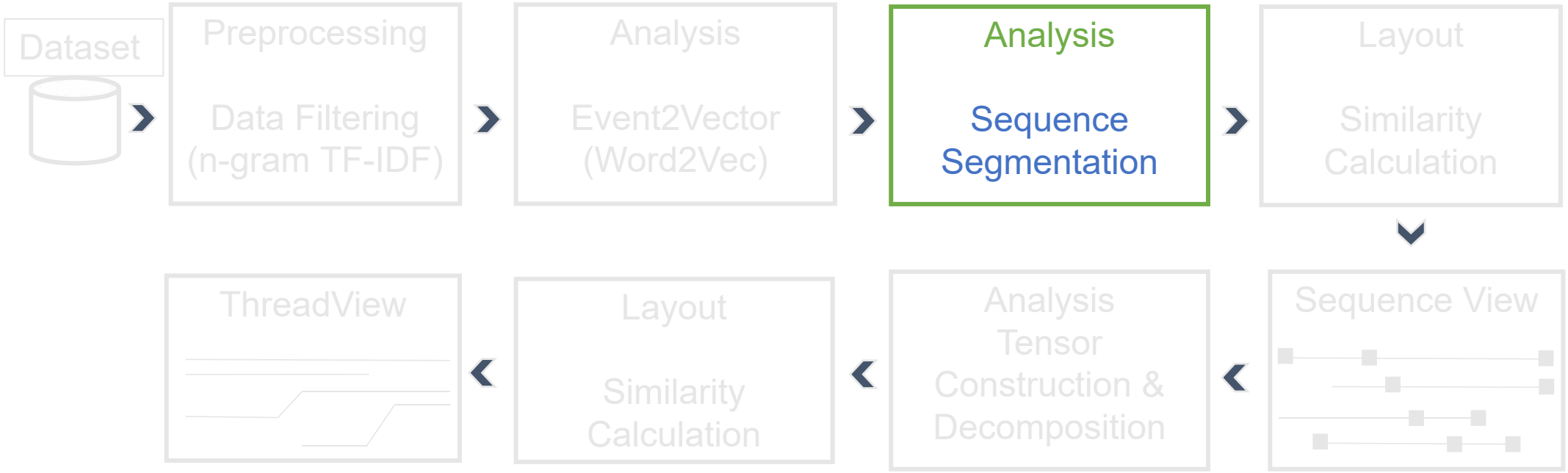
$$\begin{array}{ll} \mathbf{C_1} & e_1 \rightarrow e_3 \mid e_7 \rightarrow \cdots \mid e_{130} \\ \mathbf{C_2} & e_2 \rightarrow e_3 \rightarrow e_{10} \mid \cdots \rightarrow e_{119} \\ & \vdots \\ \mathbf{C_m} & e_5 \mid e_7 \rightarrow e_{11} \mid \cdots \rightarrow e_{80} \end{array}$$

We can calculate a similarity between each event transition, according to the Euclidean distance between event vectors.

Transitions with similarities under x will be cut into neighborhood stages.

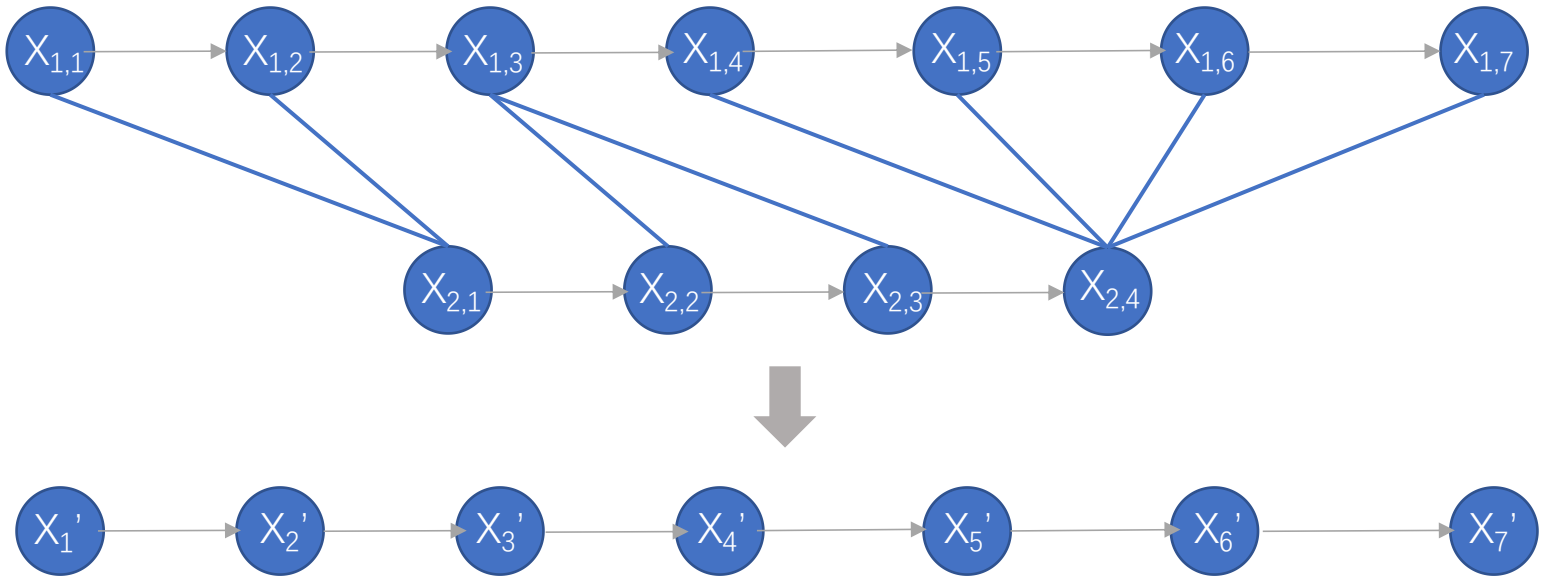
The value x can be initially given by the system and adjusted by the user.

# System Pipeline



# Sequence Segmentation: Aggregation

- We first aggregate all sequence into an average sequence by applying DTW (Dynamic Time Warping)



# Sequence Segmentation

- Then we divide the average sequence into different segments using Greedy Segmentation with iterative relaxation.

### 2.3.1 *Greedy Segmentation*

The greedy segmentation approach builds up a segmentation into $K$ segments by greedily inserting new boundaries at each step to minimize the aggregate score:

$$s^0 = \{N\} \tag{4}$$

$$s^{t+1} = \underset{i \in [1,N)}{\arg\min}\, C(s^t \cup \{i\}) \tag{5}$$

until the desired number of splits is reached. Many published text segmentation algorithms are greedy in nature, including the original C99 algorithm [3].

### 2.3.3 *Iterative Relaxation*

Inspired by the popular Lloyd algorithm for $k$-means, we attempt to retain the computational benefit of the greedy segmentation approach, but realize additional performance gains by iteratively refining the segmentation. Since text segmentation problems require contiguous blocks of text, a natural scheme for relaxation is to try to move each segment boundary optimally while keeping the edges to either side of it fixed:

$$\begin{aligned} s_k^{t+1} &= \underset{l \in (s_{k-1}^t,\ s_{k+1}^t)}{\overset{[\![\cdot]\!]}{\arg\min}} \big(\sigma(0, s_1^t) \oplus \cdots \\ &\quad \oplus \sigma(s_{k-1}^t, l) \oplus \sigma(l, s_{k+1}^t) \oplus \cdots \oplus \sigma(s_{K-1}^t, s_K^t)\big) \quad (8) \\ &= \underset{l \in (s_{k-1}^t,\ s_{k+1}^t)}{\overset{[\![\cdot]\!]}{\arg\min}} S\left(s^t - \{s_k^t\} \cup \{l\}\right) \quad (9) \end{aligned}$$

We will see in practice that by 20 iterations it has typically converged to a fixed point very close to the optimal dynamic programming segmentation.

Alemi A A, Ginsparg P. Text segmentation based on semantic word embeddings[J]. arXiv preprint arXiv:1503.05543, 2015.

# Sequence Segmentation

- The score function is determined through assuming that the content vector in each putative segment is constant, and all word vectors in the segment are drawn from the same content vector *c*.
- This determines the Content Vector Segmentation (CVS) algorithm, based on the score function

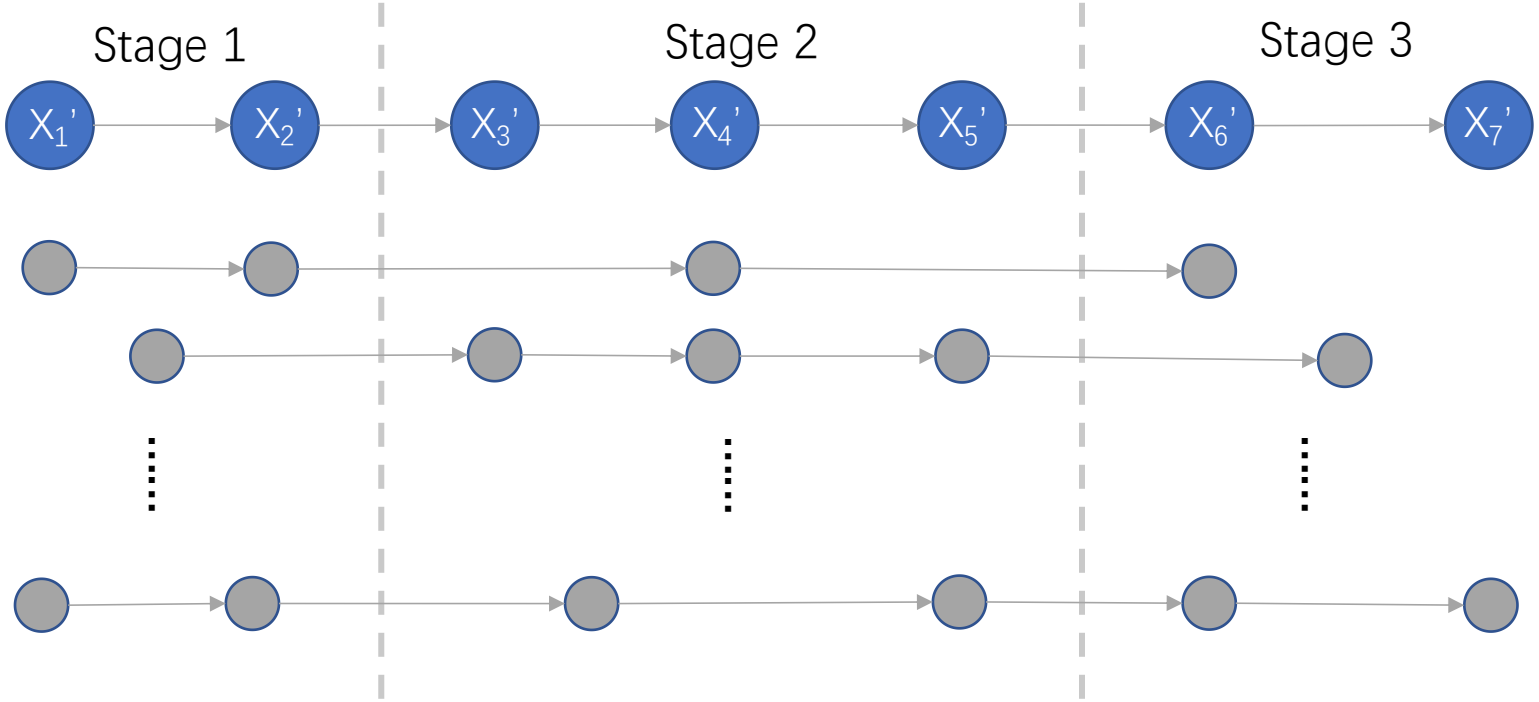
$$\sigma(i,j) = \sum_{i \le l < j} \sum_{k} w_{lk} c_k(i,j) \ .$$

- The score σ(i, j) for a segment(i, j) is the sum of the dot products of the word vectors w(lk) with the maximum likelihood content vector c(i, j) for the segment, with components given by

$$c_k(i,j) = \mathrm{sign}\left( \sum_{i \le l < j} w_{l,k} \right) \frac{1}{\sqrt{D}} \ .$$

# Sequence Segmentation

- After segmenting the average sequence, we separate each sequence from the aggregated sequence. By now, we have the stage segmentation of all sequences.

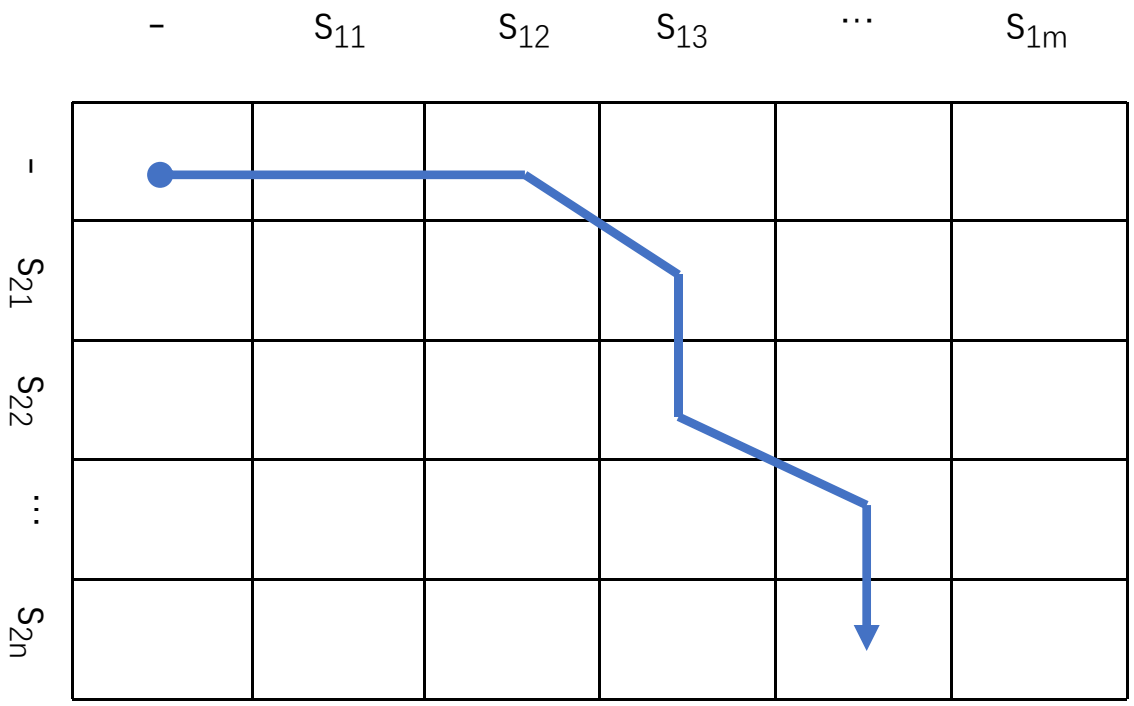


# Design Details: Stage Alignment

- After Stage Segmentation, we have

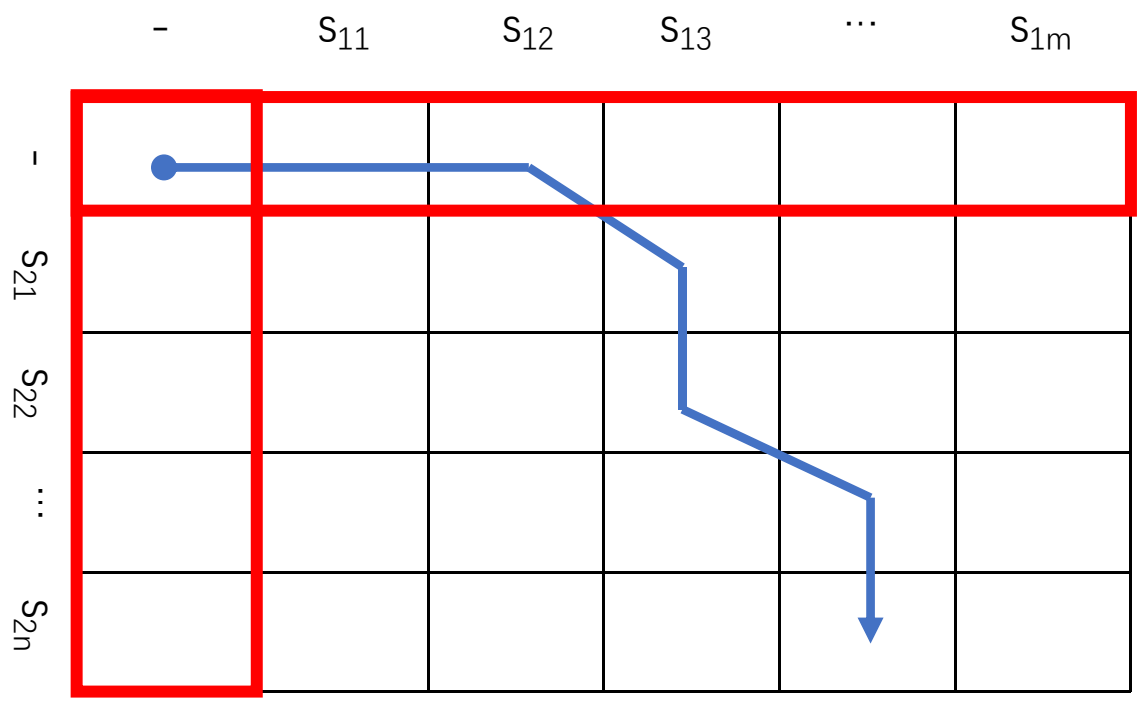
$$
\begin{array}{ll}
\mathbf{C_1} & S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow \cdots \rightarrow S_{c1} \\
\mathbf{C_2} & S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow \cdots \rightarrow S_{c2} \\
 & \vdots \\
\mathbf{C_m} & S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow \cdots \rightarrow S_{cm}
\end{array}
$$

Plan for similarity measurement between stages:
1.Euclidean distance(computational inefficient)
**2.Number of element in longest common subsequence (preferred)**
3.Number of intersect events(temporal information not considered)



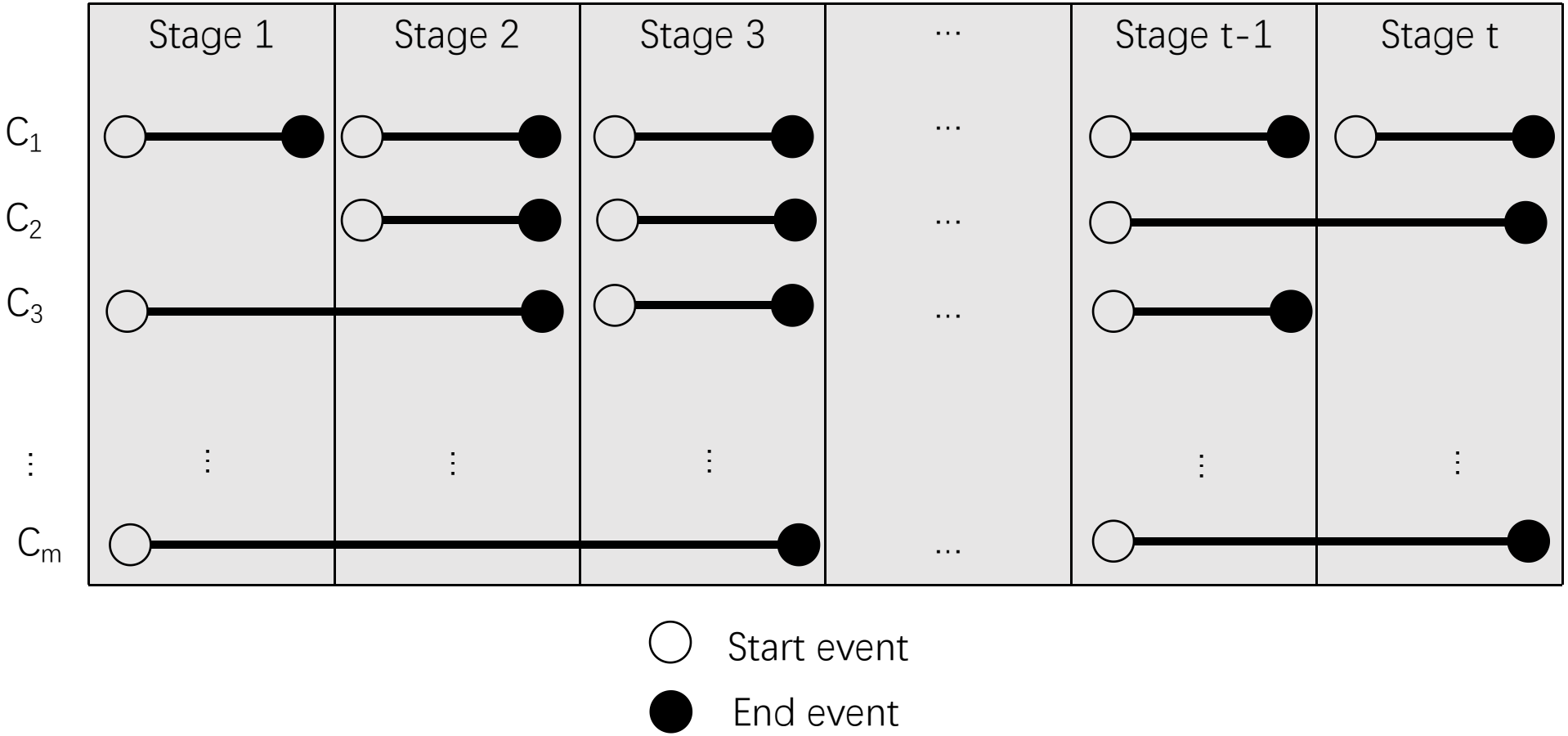

Multiple Sequence Alignment(MSA)
Can be implemented using **Clustalw**

# Design Details: Stage Alignment



# Design Details: Stage Alignment

- After stage segmentation and alignment, we have sequences:



# Visualization: Sequence View

- Information to be displayed in this view:
  - Start/end event in each stage
  - Stage duration-time from start event to end event
  - Proportion of different event types in the middle
  - Most frequent event in the middle

# Design Details : Query

Table 1

| EventID (string) | PatientID (string) | Time (timestamp) |
|---|---|---|
| | | |

Table 2

| PatientID (string) | ItemSeq (string) | EventSeq (string) | Gender (M/F) | BD (timestamp) |
|---|---|---|---|---|
| | | | | |

Table 3

| EventID (string) | EventType (string) | EventCode (string) |
|---|---|---|
| | | |

# Design Details : Query

Input

Event Pattern  (StringType “ABC”)

Event Type (StringType “A”)

Patient ID (StringType “01”)

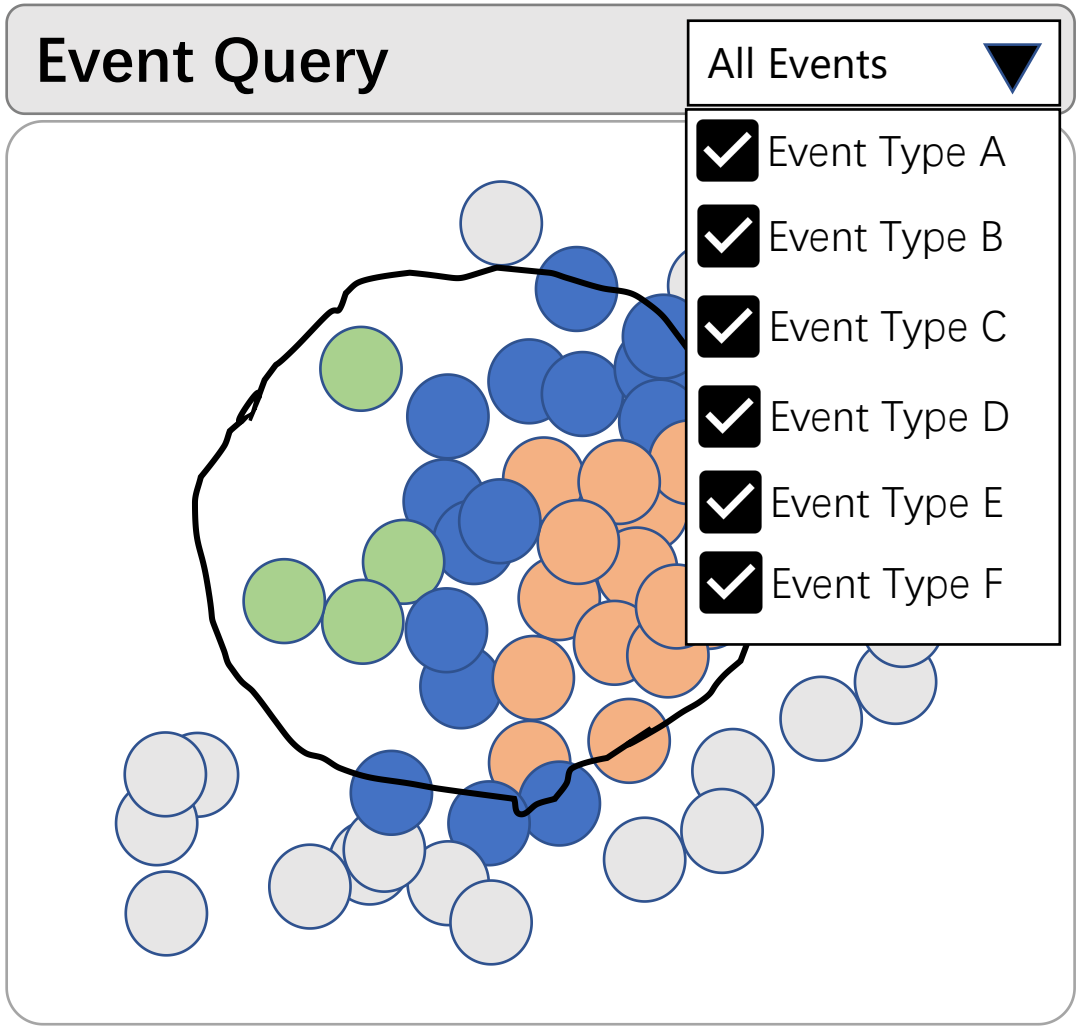
Output

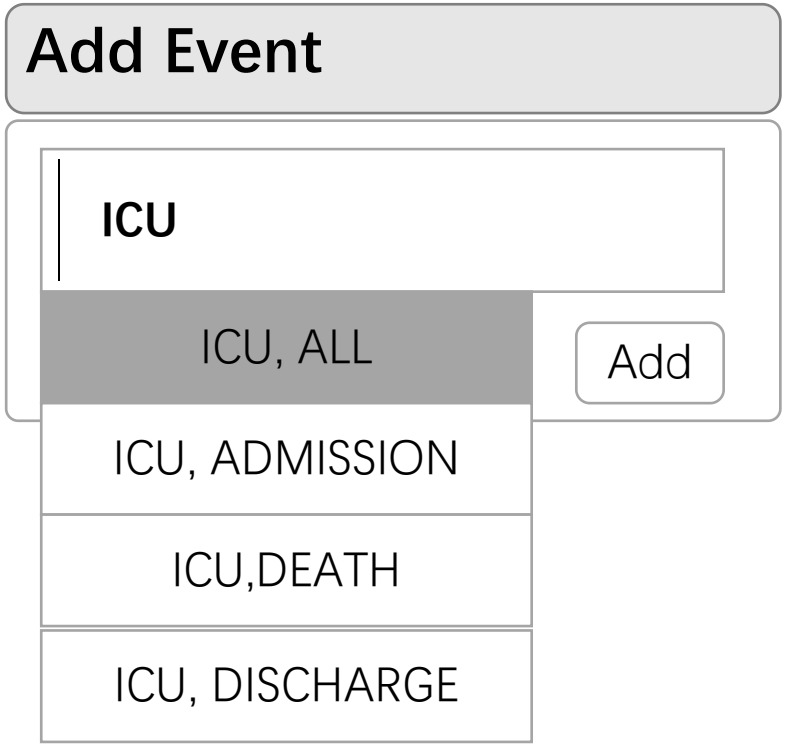
[ { patient } , { } , ……]

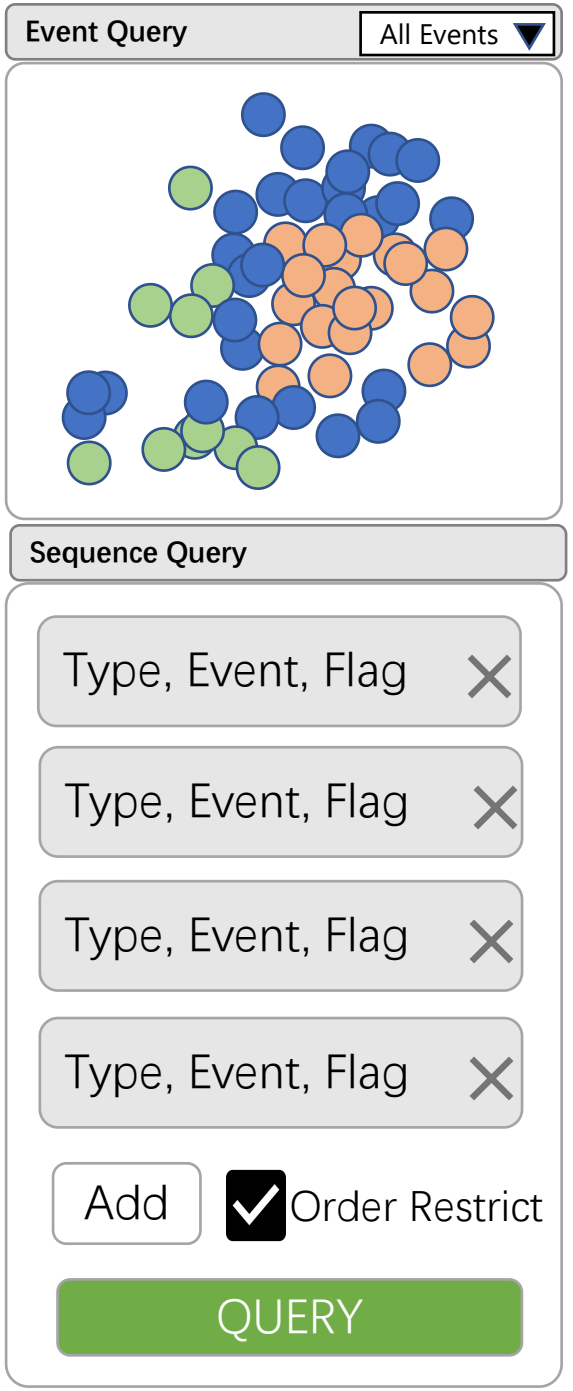
{sequence:[{ event } , { } , …. ] , gender: , BD: }

{ EventID: , EventType: , EventCode: , Time: }

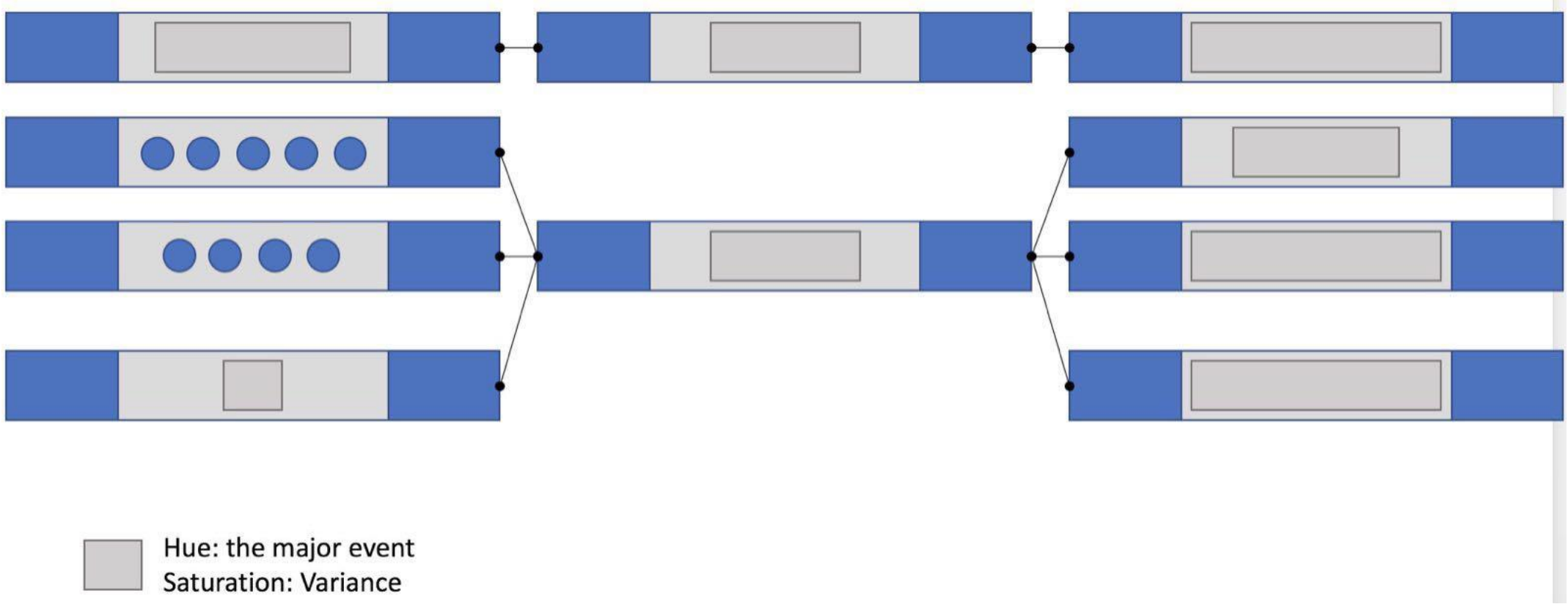
# Design Details: Query



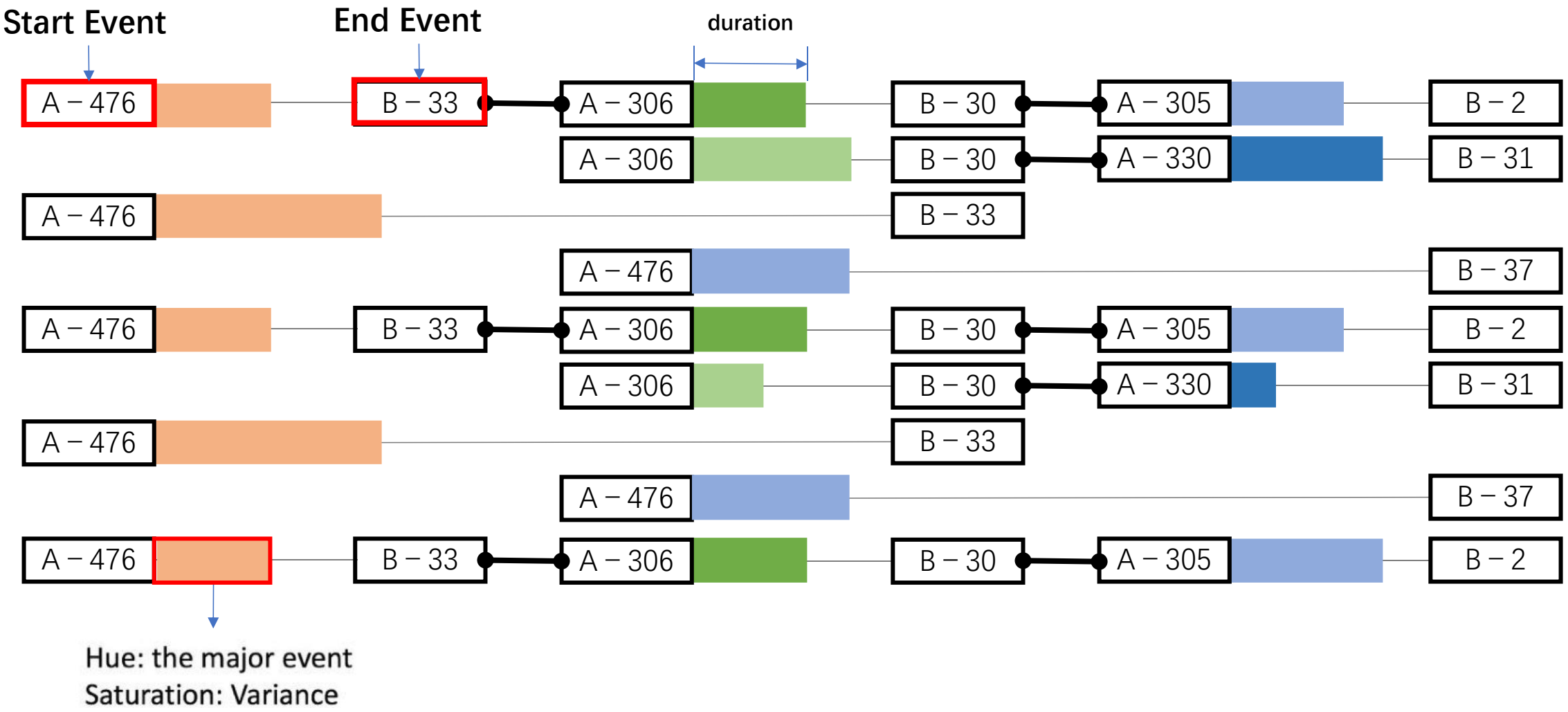



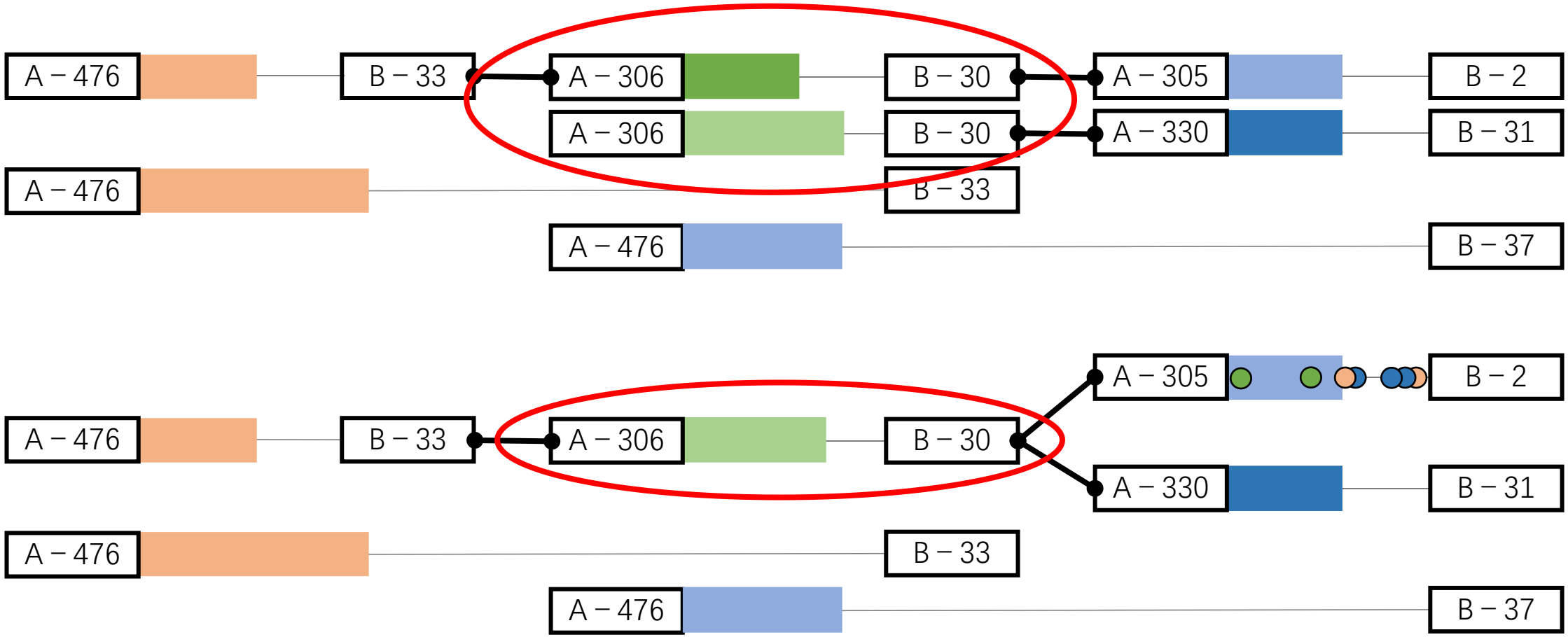
# Visualizatoin: Sequence View



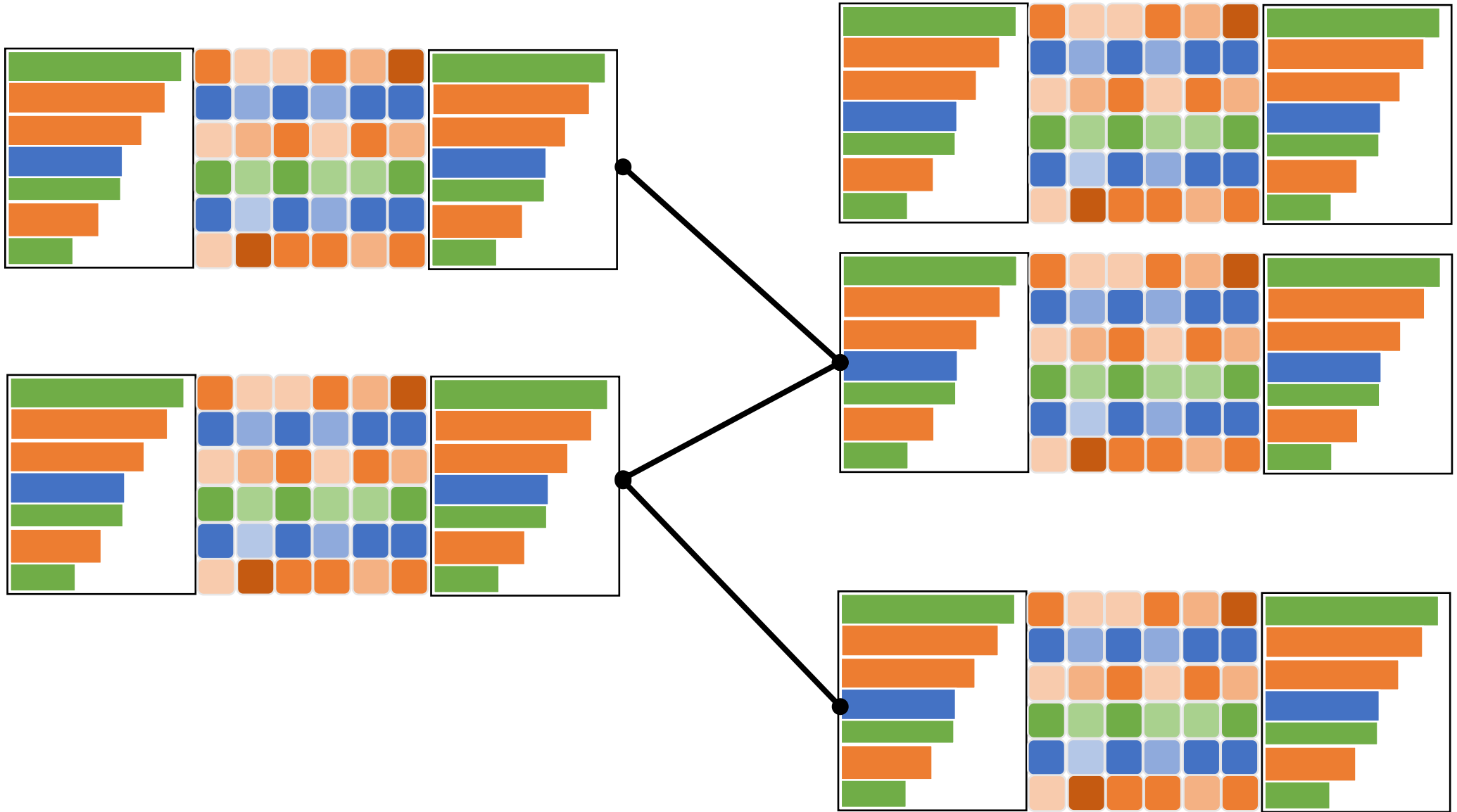
# Design Details: Sequence View



# Design Details: Sequence View



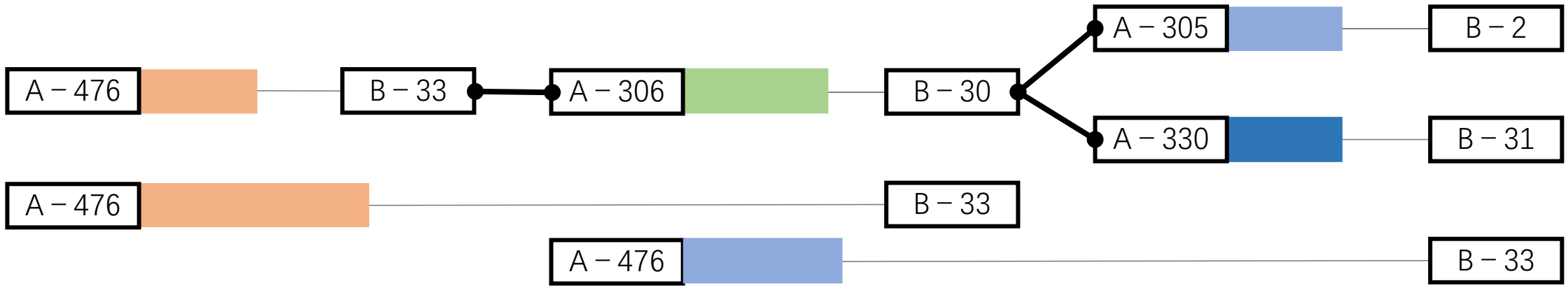
# Design Details: Cluster View



# Design Details: Stage Transition View

# Design Details: Interaction(1)

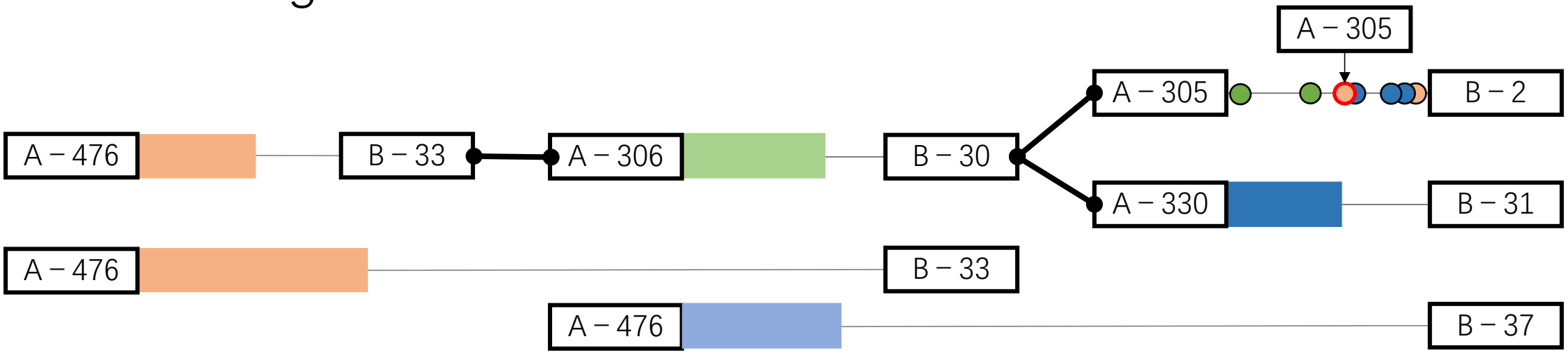
- Manual Alignment




- What will happen after user-feedback?
  - Recalculate alignment before and after manipulated segment.
  - Recalculate Layout.
  - Move.

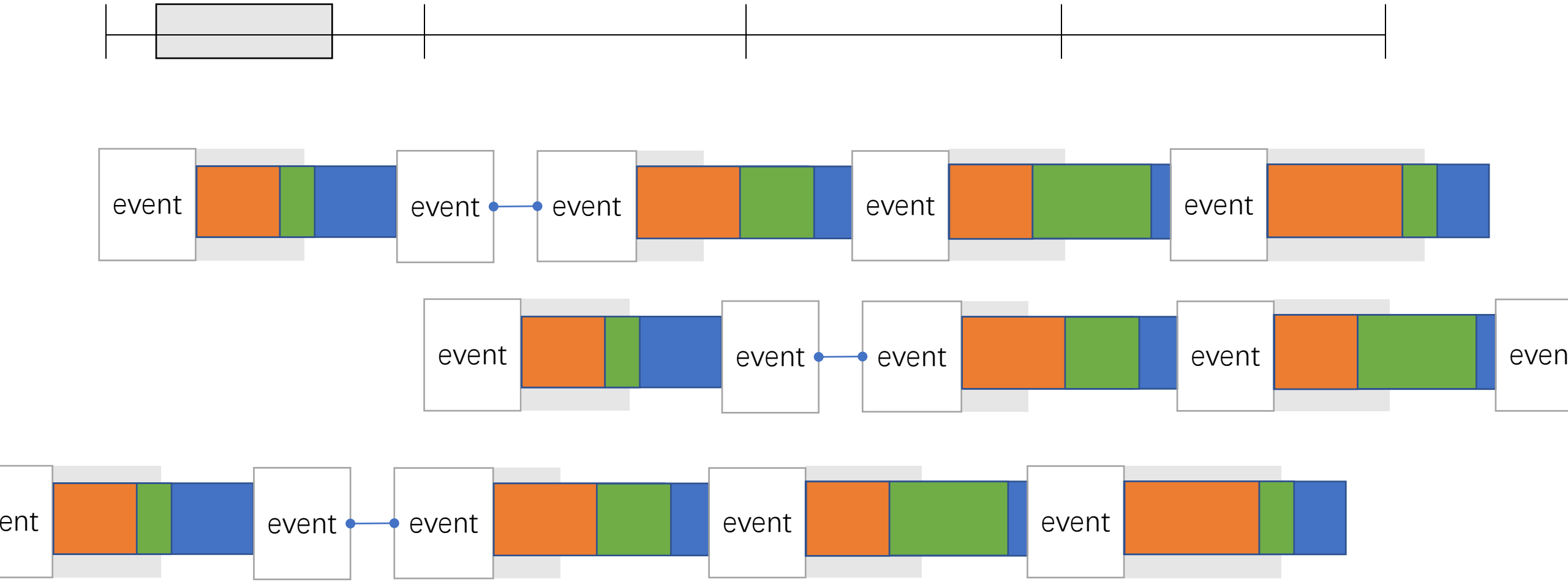
# Design Details: Interaction(1)

- Manual Segmentation

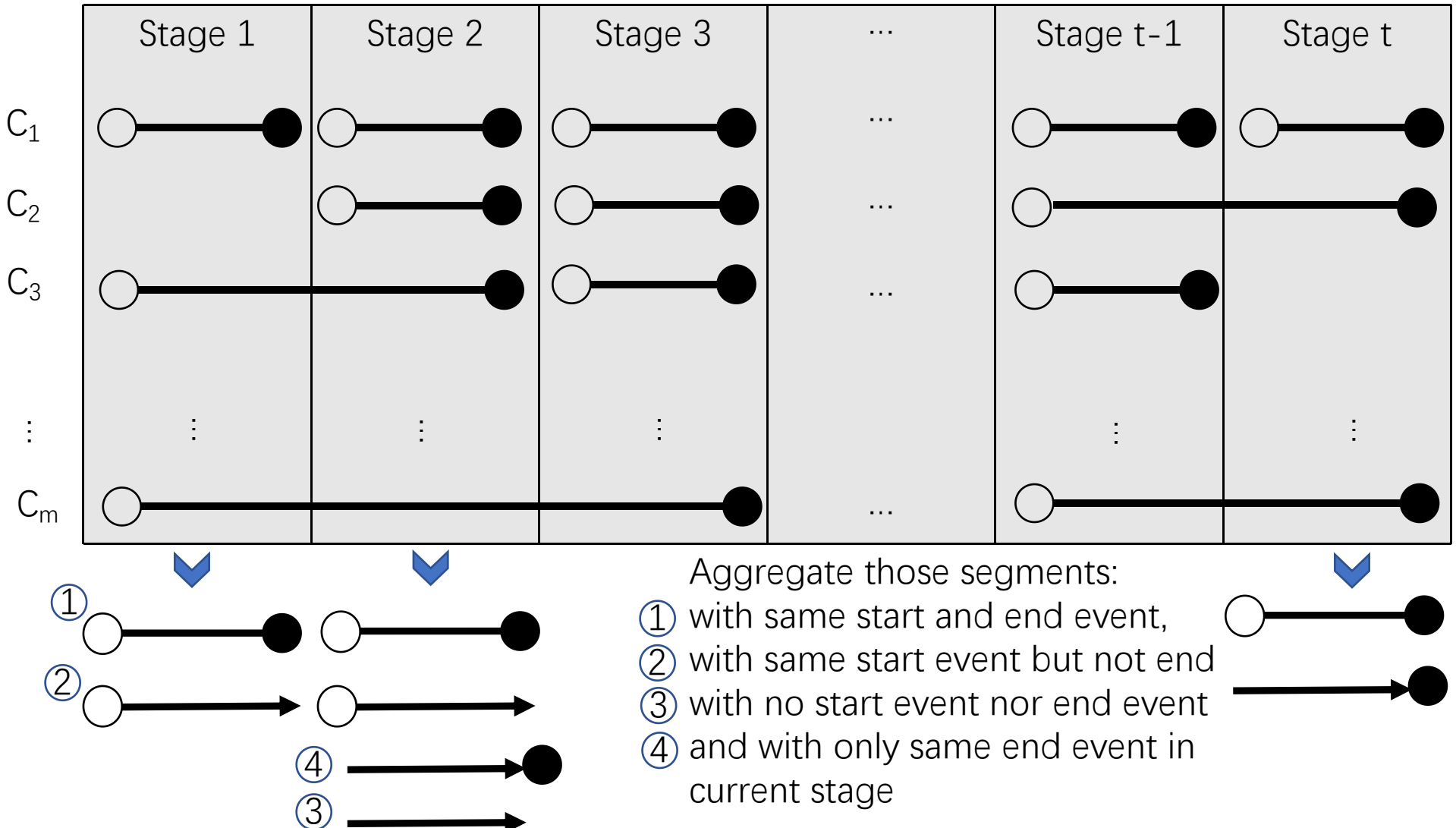



- What will happen after user-feedback?
  - Insert a new cut to original sequence
  - Recalculate sequence alignment
  - Recalculate layout
  - Move

# Design Details: Sequence View



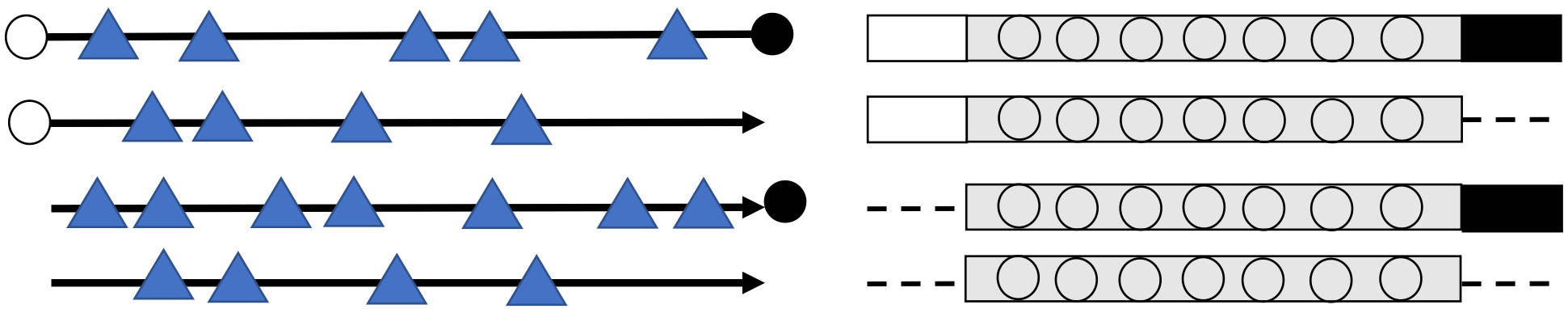
# Design Details: Segment Aggregation

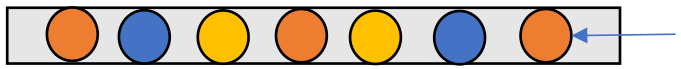



Aggregate those segments:

① with same start and end event,
② with same start event but not end
③ with no start event nor end event
④ and with only same end event in current stage

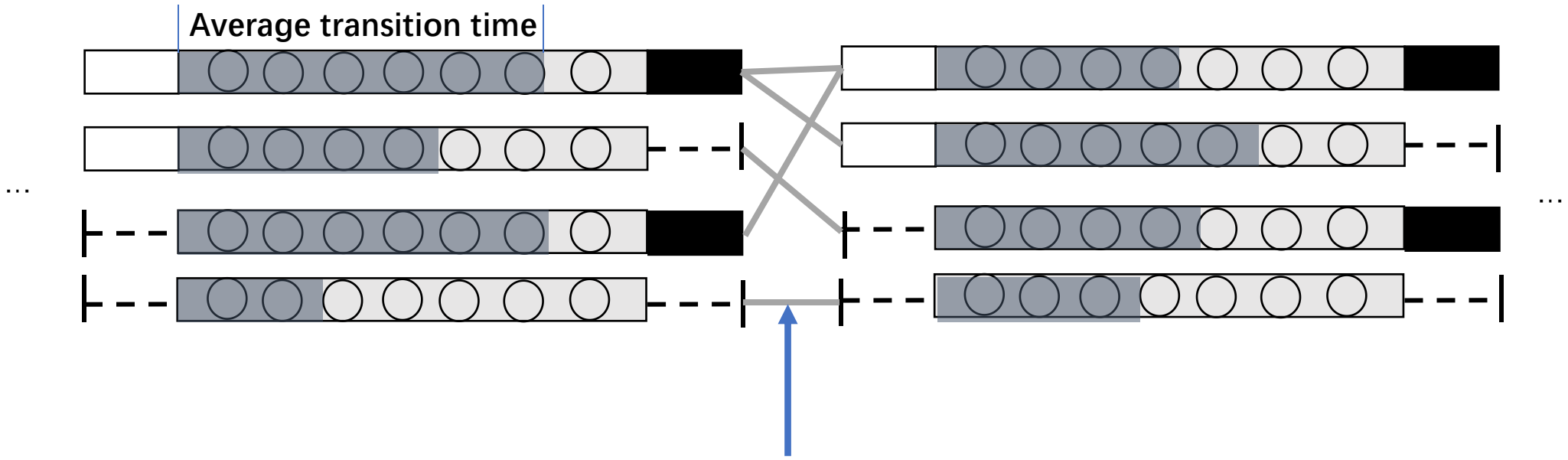
# Design Details: Statistical Analysis





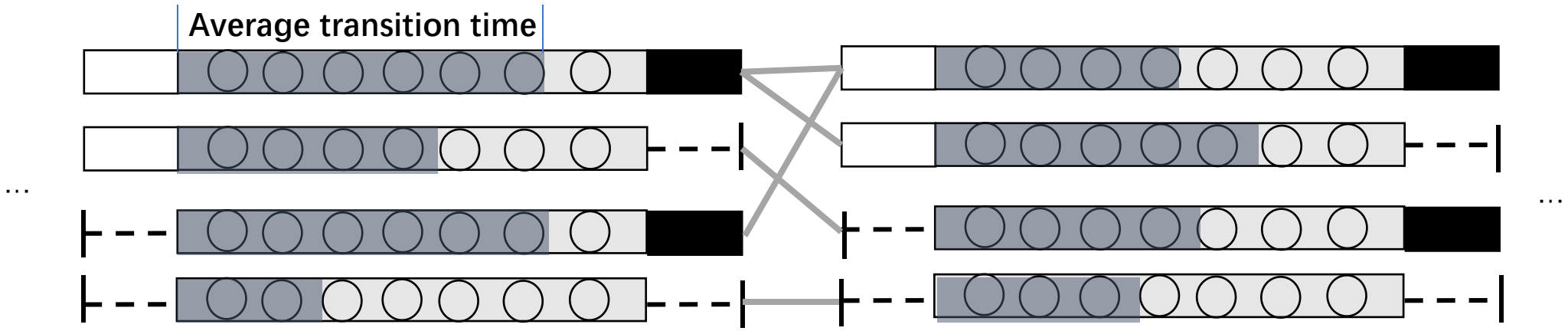
Events with high occurrence frequency. The size of nodes encodes frequency, color encodes event type

# Design Details: Visualization




**Sankey Diagram**
Link width is proportional to the number of sequences

# Design Details: Layout Algorithm



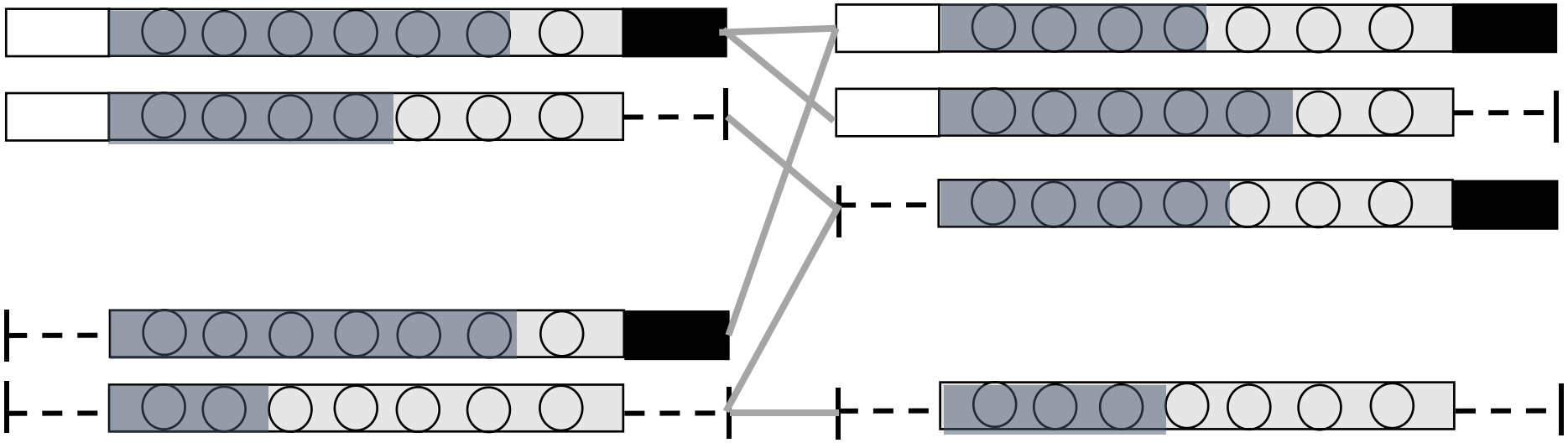

$$min \sum_{t} \alpha \sum_{i<j} w_{ij}(t) \left\| y_i(t) - y_j(t) \right\|^2 + (1-\alpha) \sum_{i} \sum_{j} m\theta_{ij} \left\| y_i(t) - y_j(t-1) \right\|^2$$

Cluster by segment similarity

Reduce visual clutter in sankey diagram

# Design Details: Interaction(1)

- Thread Regrouping
  - Similar to old version
  - Users use slider bar to re-group segments in different stages
  - Expected result:



# Design Details: Interaction(2)

- Stage Modification
  - Set a specific event x as the start of a particular stage
    - Plan 1: For subsequences ahead of current stage, rerun all processes from stage segmentation. (Will take time, computational complexity unknown)
    - Plan 2: Abandon subsequence before event x in the current stage to avoid huge layout rearrangement.
    - Plan 3: Still thinking.