ILD software/analysis meeting

# Vertex Finder with Deep Learning

Kyushu University
Kiichi Goto,
Taikan Suehara, Kiyotomo Kawagoe

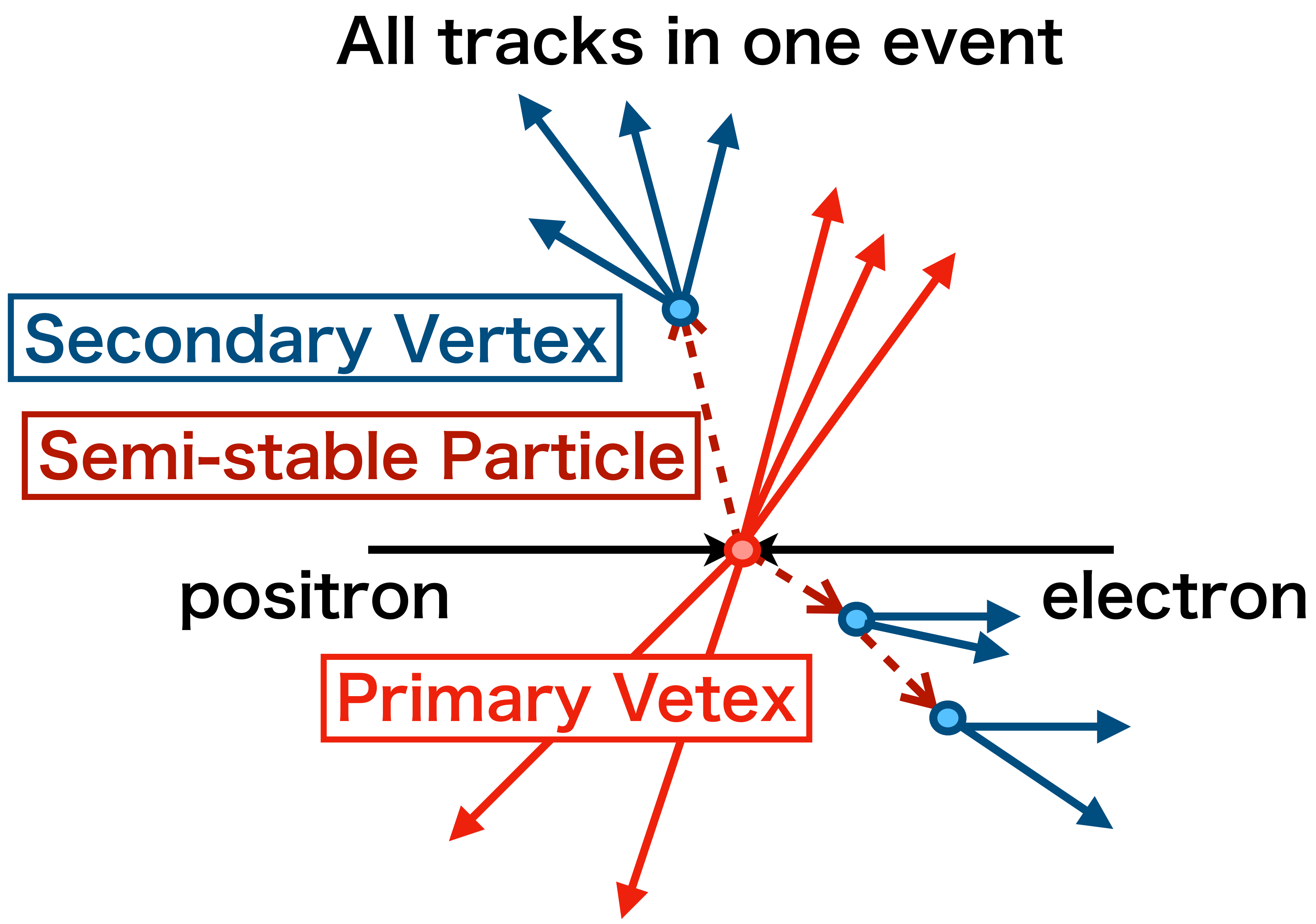
# Contents

# 1. Introduction

## Vertex Finder

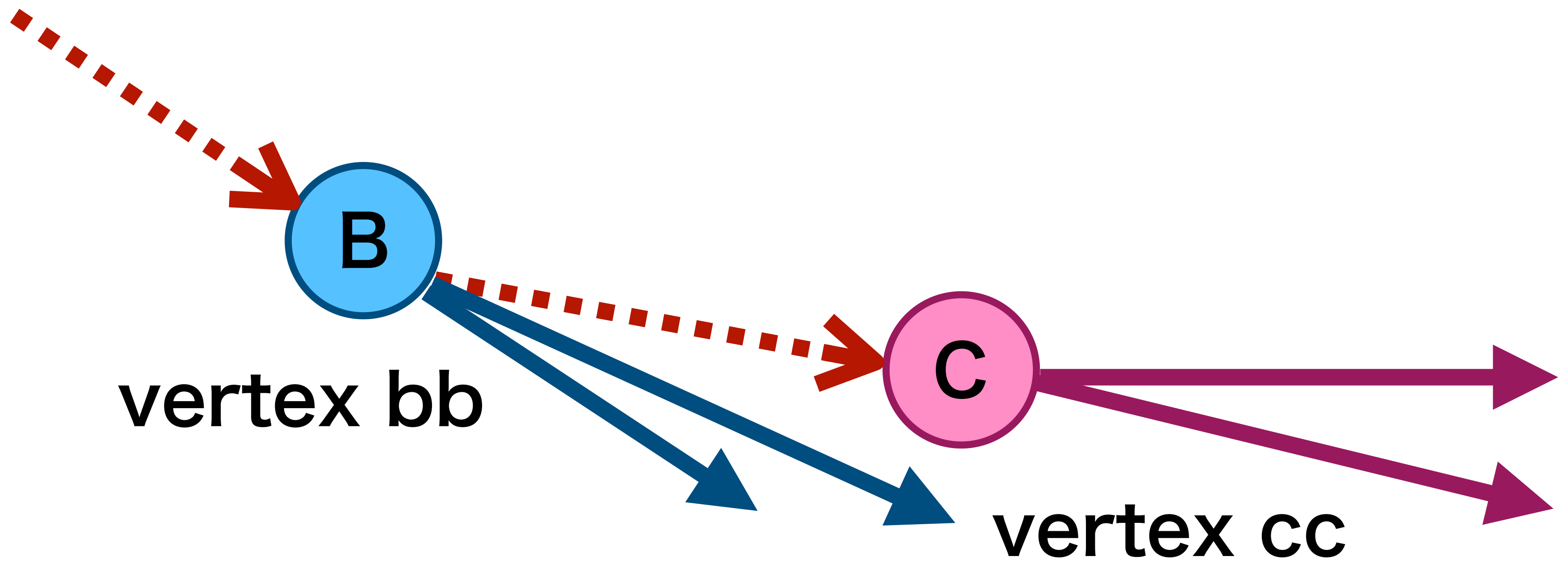
- Purpose : To know which vertex does the track come from

- Now, in ILC the "LCFIPlus" is used for "Vertex Finding"
  - Evaluate based on some thresholds tuned by human : Cut base analysis
    ➡ Improve using deep learning




- Data property (used in this study)
  - Using Monte Carlo simulation data that the final state is $b\bar{b}$
  - The labels of training data are created from MC truth
  - Using following variables as track information (22 variables)
    - Helix parameters (d0, z0, $\phi$, $\omega$, tan$\lambda$)
    - Charge
    - Energy

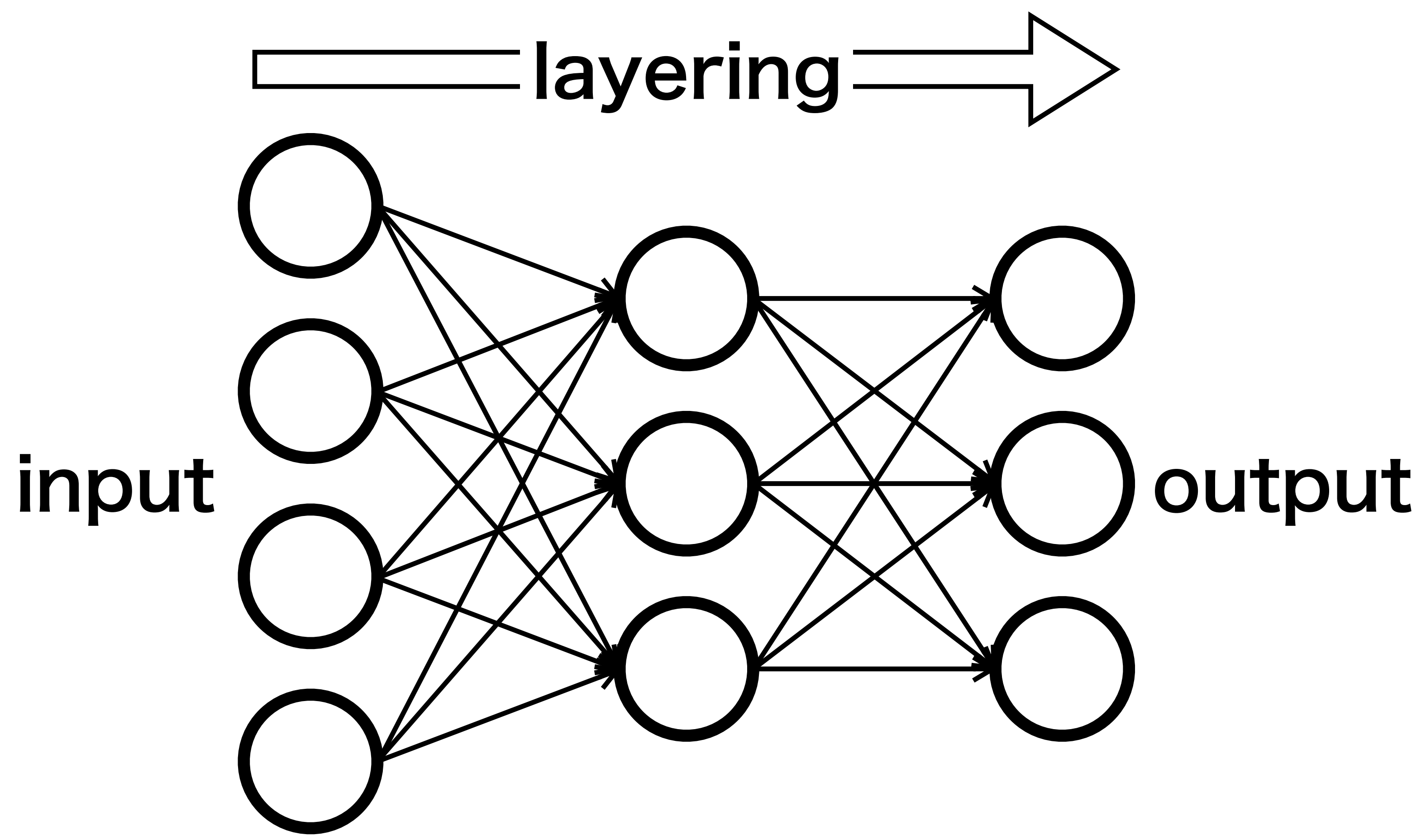
When the final state is $b\bar{b}$,
Secondary Vertex of flavor c are generated



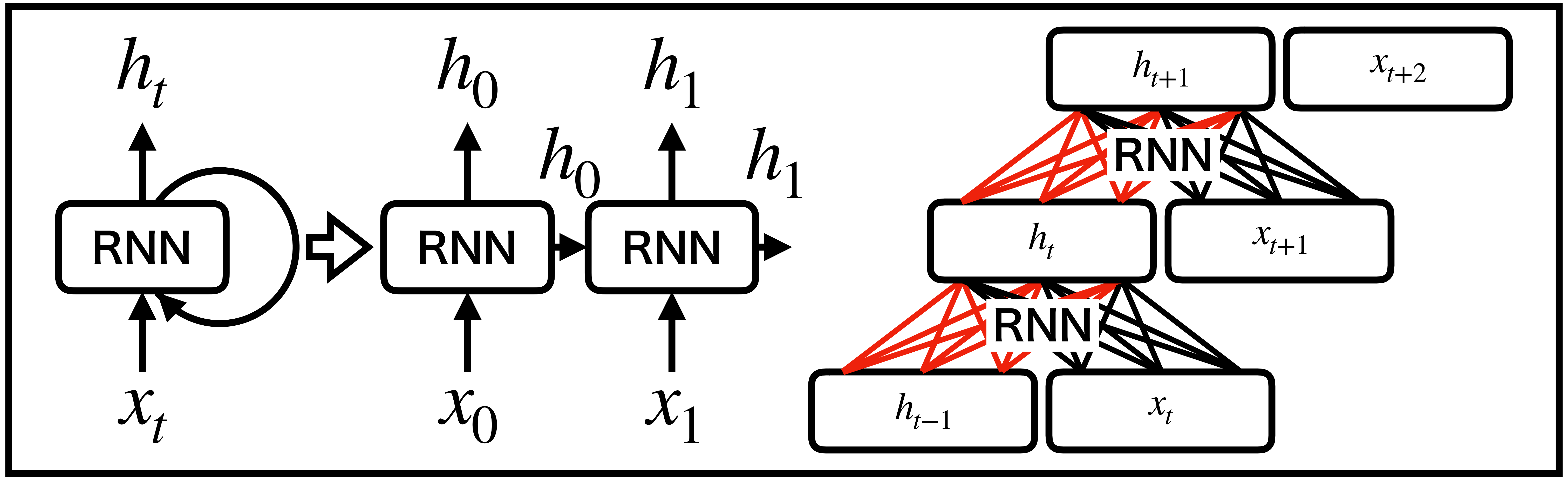
# 1. Introduction

## Deep Learning

- One of the Machine Learning technologies
- Basically it is "Supervised Learning", and can solve "Classification" and "Regression"
  - Supervised Learning : pattern recognition based on training data
    - fitting output to training by weight updating
- Complex (non-linear) problems can be solved by "layering" simple (linear) networks
- Recently, various practical networks are provided




Recurrent Neural Networks : can process the series data

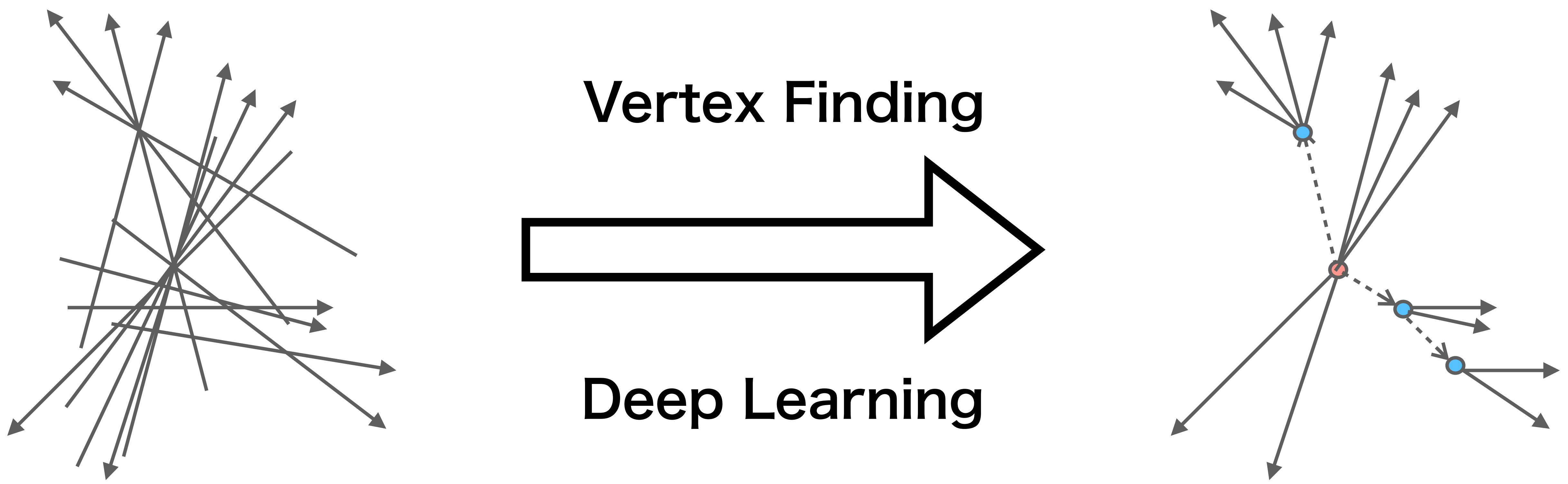


# 2. Networks

## Networks

- Finding the vertex using two networks

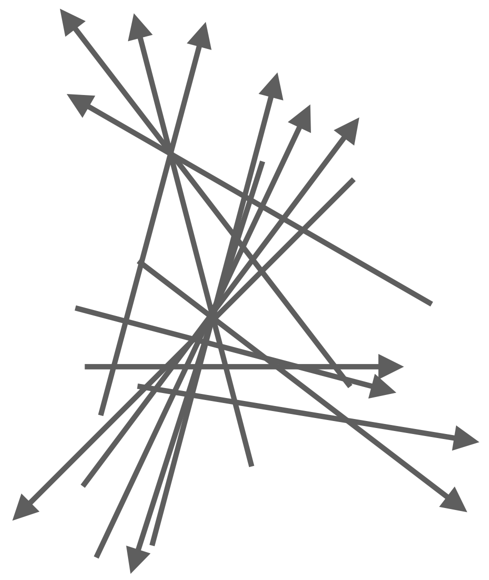
1. The network for track pairs
   - Classify the track pairs to the vertex classes
   - Input : "Track pair (Two tracks)"
   - Output : "Not connected" "Primary Vertex" "Secondary Vertex"
   - ➡ Search for the "seed" of the vertex
2. The network for any number of tracks
   - Determine whether any number of tracks are connected
   - Input : "The seed of the vertex" "Any number of tracks"
   - Output : "Connected" "Not connected"
   - ➡ Reconstruct the vertex with adding the tracks to the seed evaluated by "network 1"



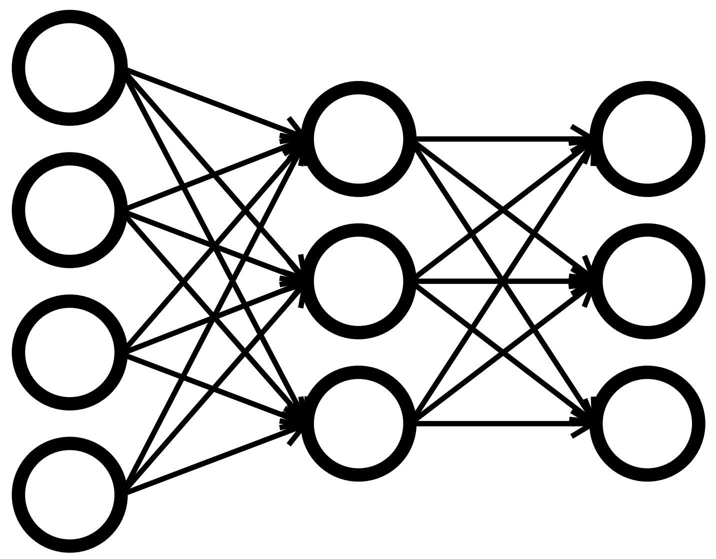
# 2. Networks

## Networks

1. The network for track pairs (seed finding)



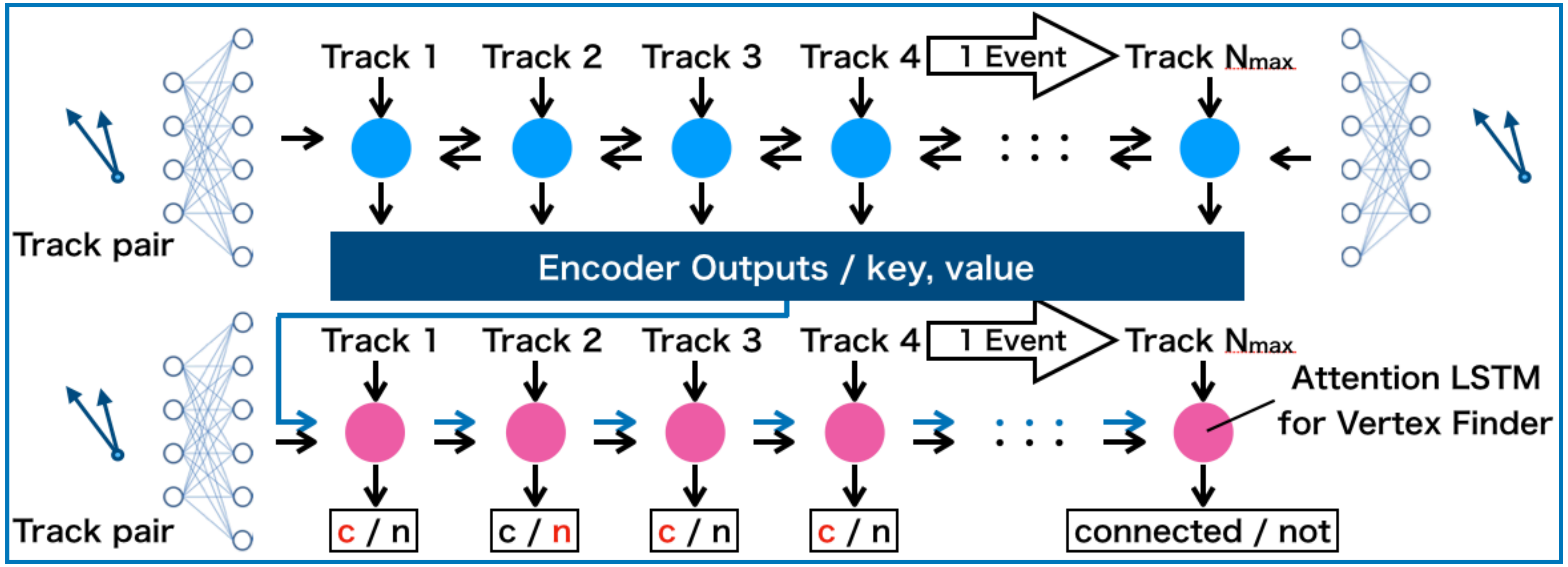
All combination of track pairs
[Track 0, Track 1],
[Track 0, Track 2],
[Track 0, Track 3], ...



Primary Vertex
Secondary Vertex
Not connected

2. The network for any number of tracks (vertex production)

Using
connected pair
(Primary and
Secondary)
as the initial state

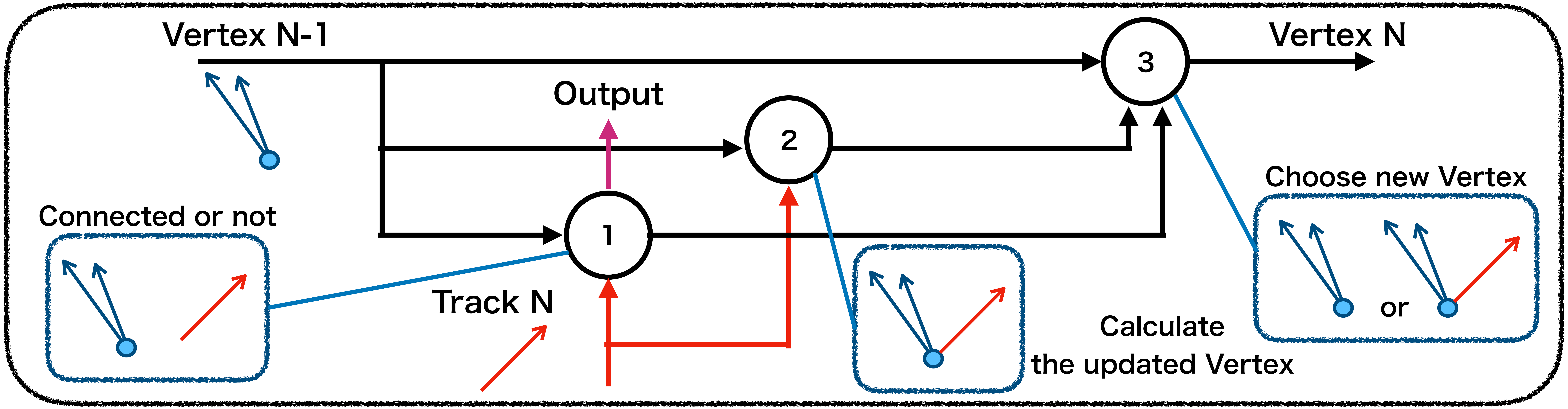


# 2. Networks

## LSTM for Vertex Finder

- The track data are not "sequential data" ➡ extend the "LSTM"
  - LSTM (Long short-term memory) : One of the DL techniques, it can process the series data

1. Determine whether the Track N is connected to the Vertex N-1 $h_N = \sigma(D_h[\sigma(W_o t_N + R_o V_{N-1}) \cdot \tanh(V_{N-1})])$
2. Calculate the updated Vertex with the Track N $U_N = \sigma(W_i t_N + R_i V_{N-1}) \cdot \tanh(W_z t_N + R_z V_{N-1}) + \sigma(W_f t_N + R_f V_{N-1}) \cdot V_{N-1}$
3. In "1", if it is determined to be connected, the updated Vertex,
   if it is determined to be not connected, the Vertex N-1 is adopted as the Vertex N $V_N = (1 - h_N)V_{N-1} + h_N U_N$

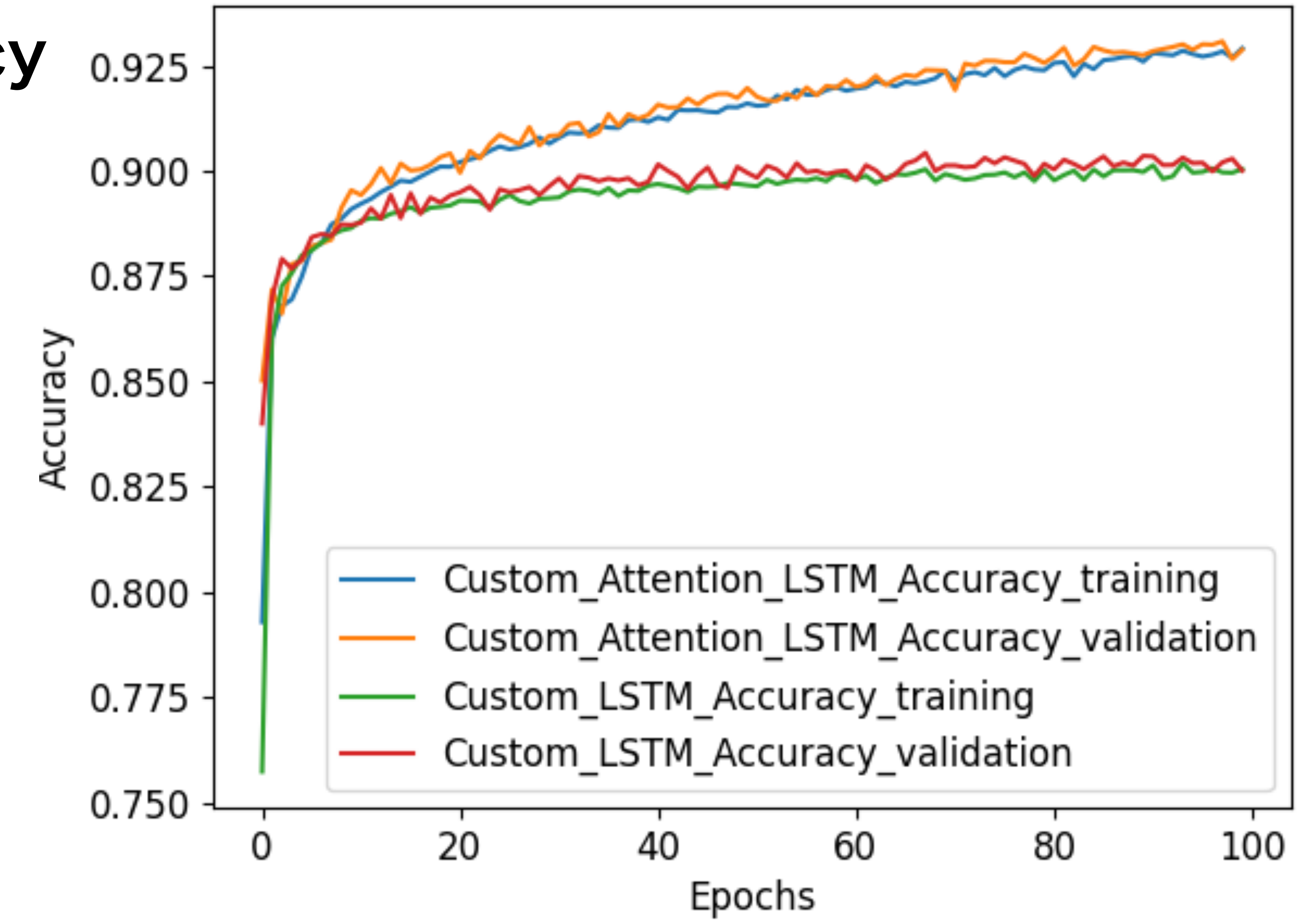


# 2. Networks

## Training and Performance

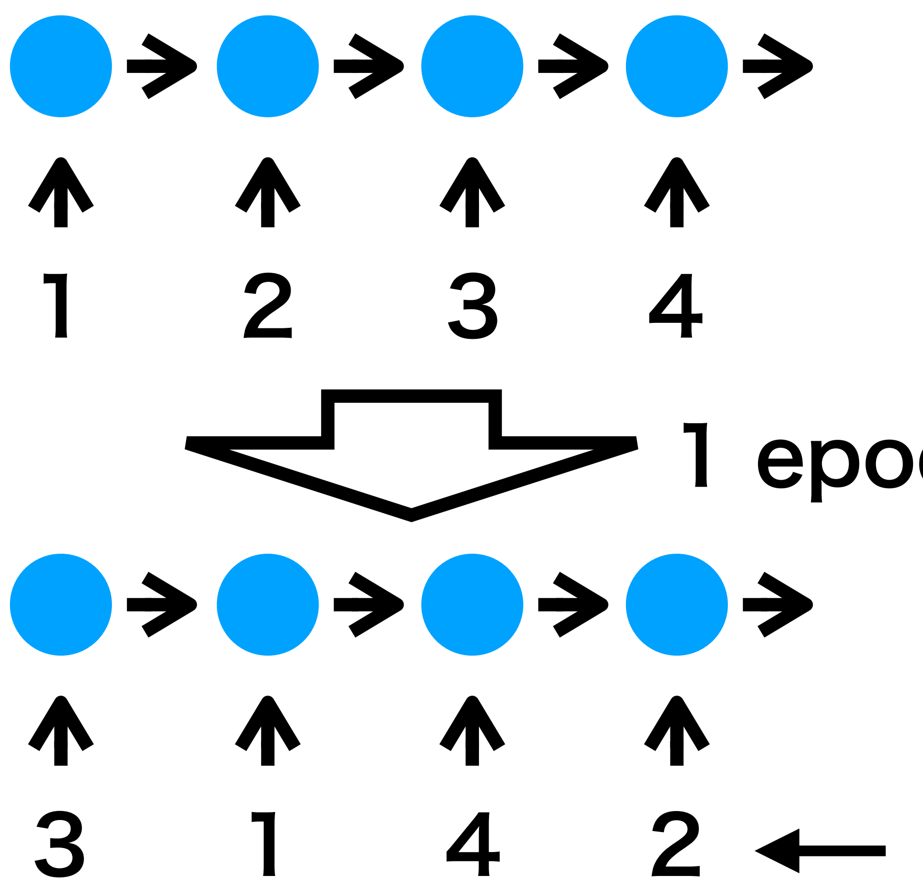
- Loss function : binary cross entropy
- Optimization/Learning rate : Adam/0.001
  - ‣ The method of weight update and step width
- The number of training (Epoch) : 100 epochs
- Batch size : 32
  - ‣ The number of samples per a weight update
- ‣ Hardware : TITAN RTX × 2

Accuracy




- 20000 Event (1159547 samples) ➡ Randomly chosen 50000 samples per a epoch
  - ‣ Create one training sample per a seed

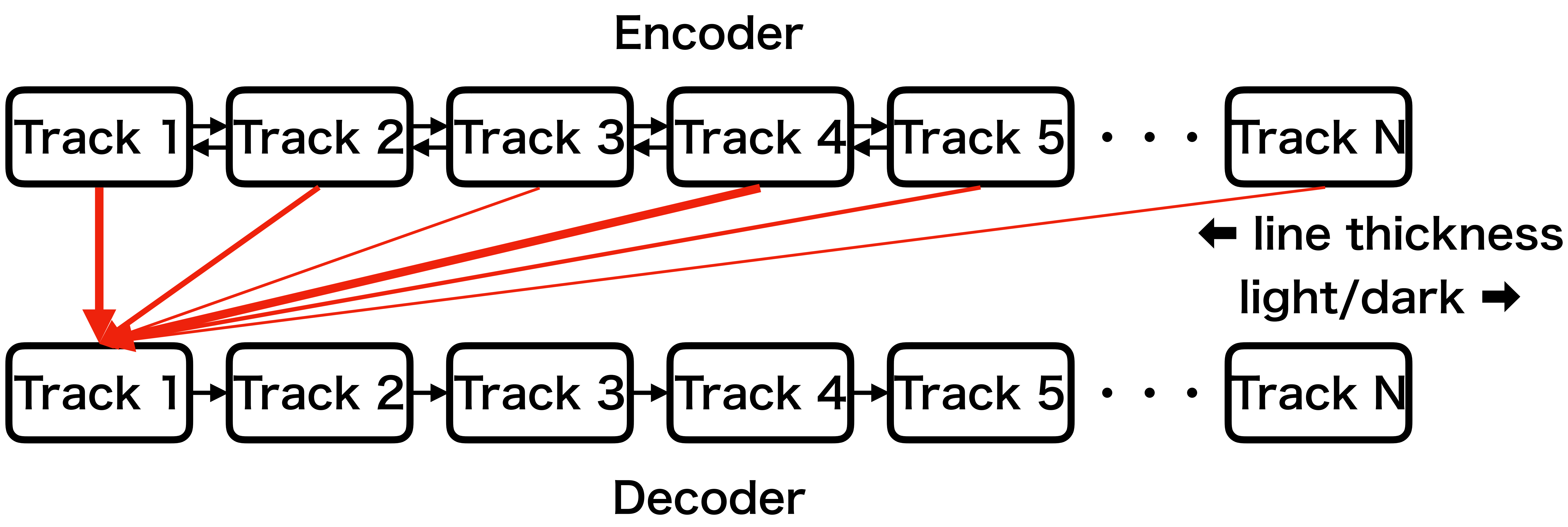
- Zero padding and masked
  - "Zero padding" by the maximum number of tracks for all events
    "Masked" not to influence the training
  - Actually, decoder can process any tracks
- Shuffle the sequence of tracks
  - Since tracks are not sequential in nature,
    we shuffle the order of the tracks at each epoch

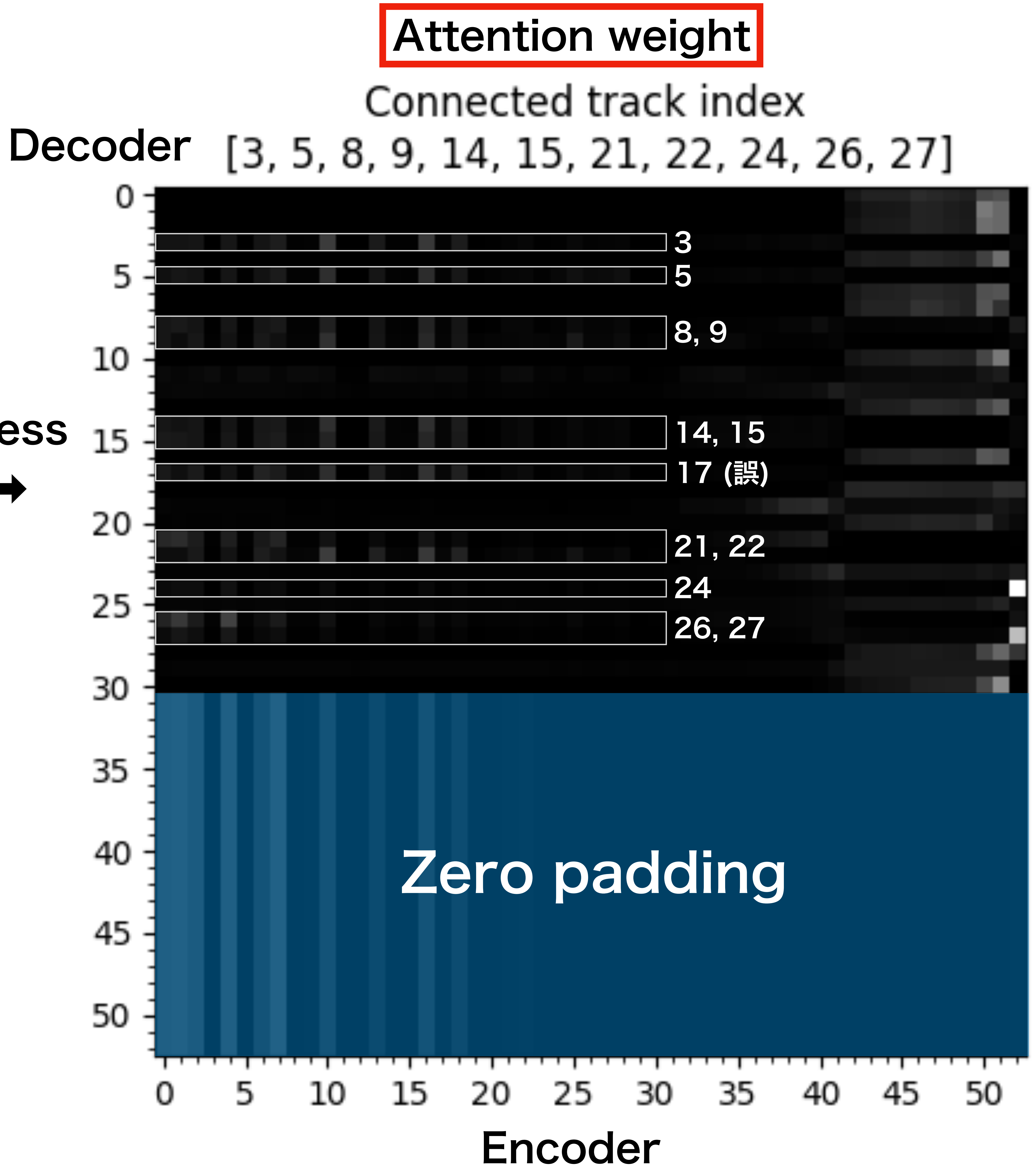


# 2. Networks

## Attention

- I want each track to give "attention" to all tracks in an event



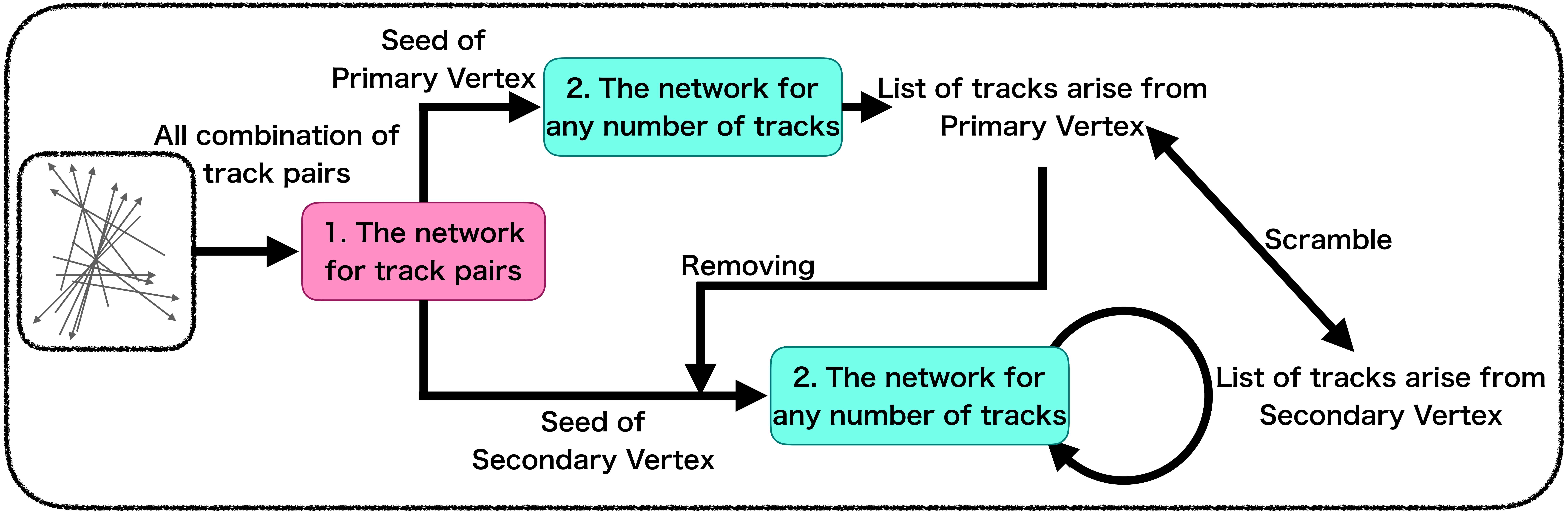

- Tracks predicted to be connected can be received information from tracks
- it is necessary to study "which track" it is getting information from

# 3. Vertex Finding

## Algorithm for Vertex Finding with Deep Learning

- Vertex finding with two networks

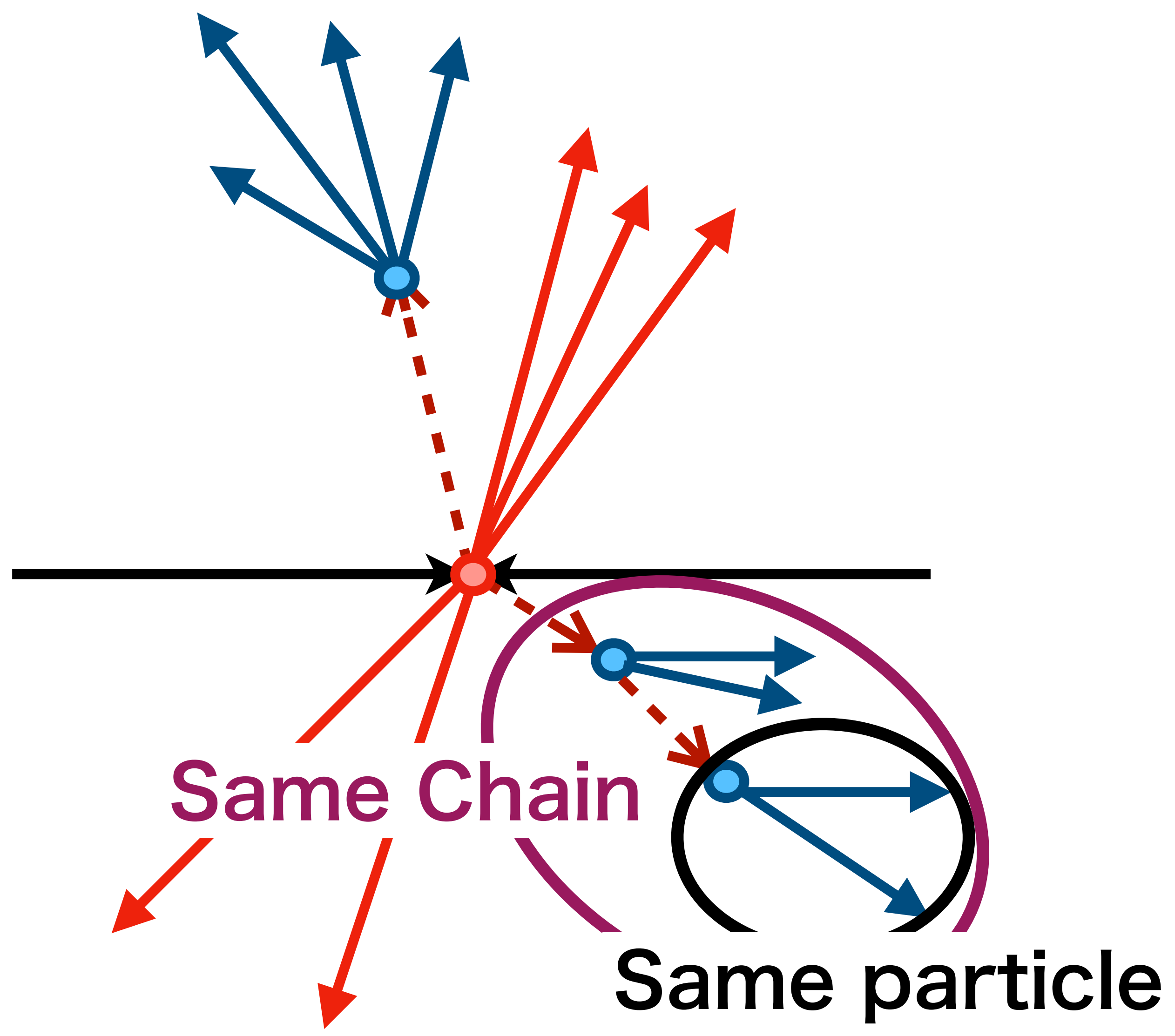
1. The network for track pairs
2. The network for any number of tracks



# 3. Vertex Finding

## Comparison with LCFIPlus

- Comparison with performance of LCFIPlus
  - T. Suehara, T. Tanabe, LCFIPlus: A framework for jet analysis in linear collider studies, Nuclear Instruments and Methods in Physics Research A 808 (2016) 109-116
- Items
  - Labeled Primary by MC, predicted Secondary Vertex by DL
  - Labeled Others by MC, predicted Secondary Vertex by DL
    - Others : They are not Primary, Bottom, Charm
  - Labeled Bottom by MC, predicted Secondary Vertex by DL
    - Rate of the tracks chosen from same chain
    - Rate of the tracks chosen from same particle
  - Labeled Charm by MC, predicted Secondary Vertex by DL
    - Rate of the tracks chosen from same chain
    - Rate of the tracks chosen from same particle




This study@average 100 events

| | Primary | Others | Bottom | Charm |
|---|---|---|---|---|
| Second | 0.9% | 10.6% | 67.1% | 72.9% |
| Chain | | | 63.8% | 69.6% |
| Particle | | | 34.5% | 43.4% |

LCFIPlus

| | Primary | Others | Bottom | Charm |
|---|---|---|---|---|
| Second | 0.6% | 2.5% | 57.5% | 64.3% |
| Chain | | 1.9% | 56.6% | 63.4% |
| Particle | | 1.2% | 32.2% | 38.9% |

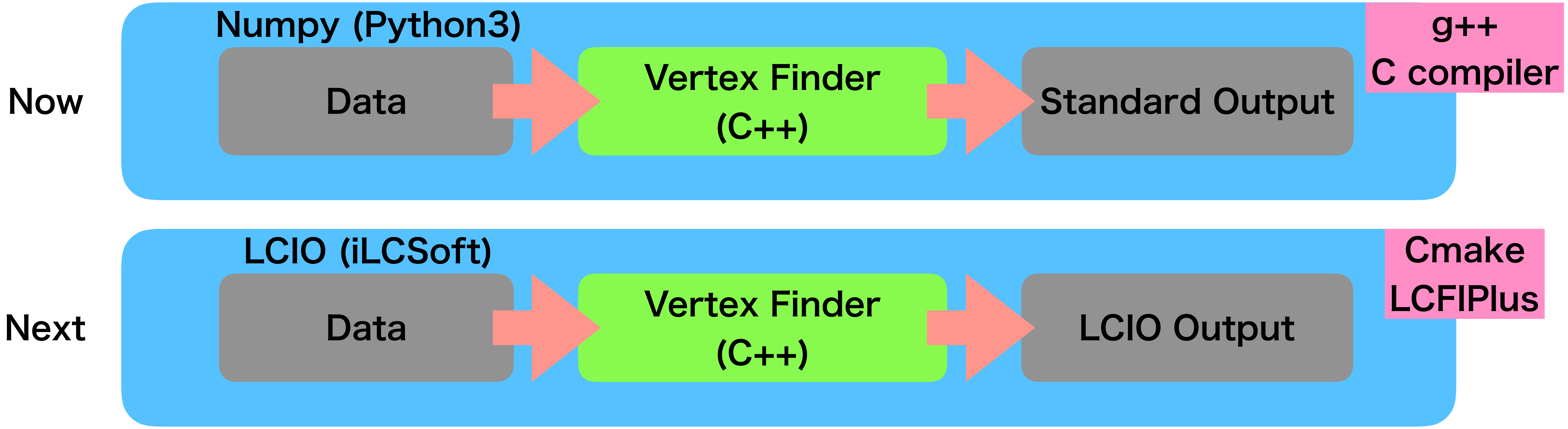
# 4. Inference with C++

## For Evaluation in LCFIPlus

- I want to show the performance of Flavor Tagging with my Vertex Finder
  ➡ I need to run these networks in LCFIPlus

- LCFIPlus are written with "C++" and "Cmake", but the networks are constructed by "Python 3"

- Now I'm implementing the vertex finding with "C++"
  - Completed running the networks from C++
    - Almost same performance with Python could be obtained



# 5. Summary and Next step

- I'm constructing the networks for vertex finding
- I extend the LSTM for vertex finding and produced the vertex
- I can become to compare the performance with LCFIPlus
    - Purity is little bit bad, but Efficiency is improved

- Now I try to run the my Vertex Finder in LCFIPlus

This study@average 100 events

| | Primary | Others | Bottom | Charm |
|---|---|---|---|---|
| Second | 0.9% | 10.6% | 67.1% | 72.9% |
| Chain | | | 63.8% | 69.6% |
| Particle | | | 34.5% | 43.4% |

LCFIPlus

| | Primary | Others | Bottom | Charm |
|---|---|---|---|---|
| Second | 0.6% | 2.5% | 57.5% | 64.3% |
| Chain | | 1.9% | 56.6% | 63.4% |
| Particle | | 1.2% | 32.2% | 38.9% |

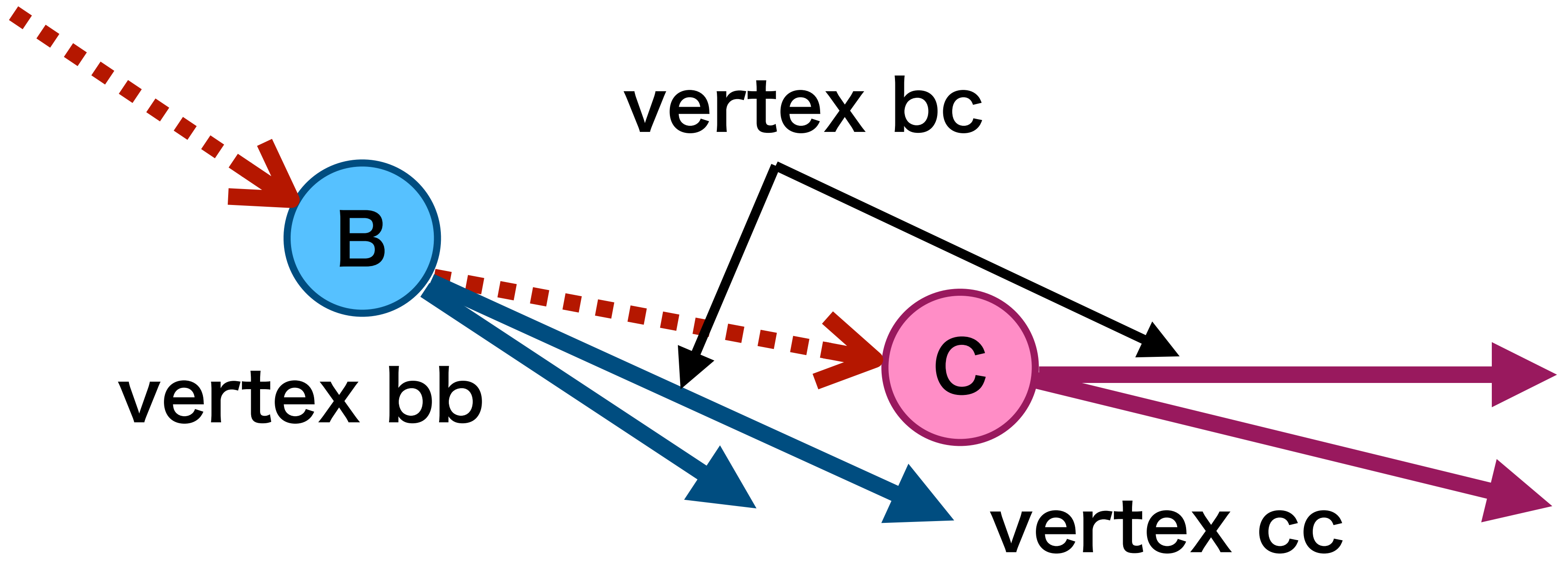
# Backup

# 1. Introduction

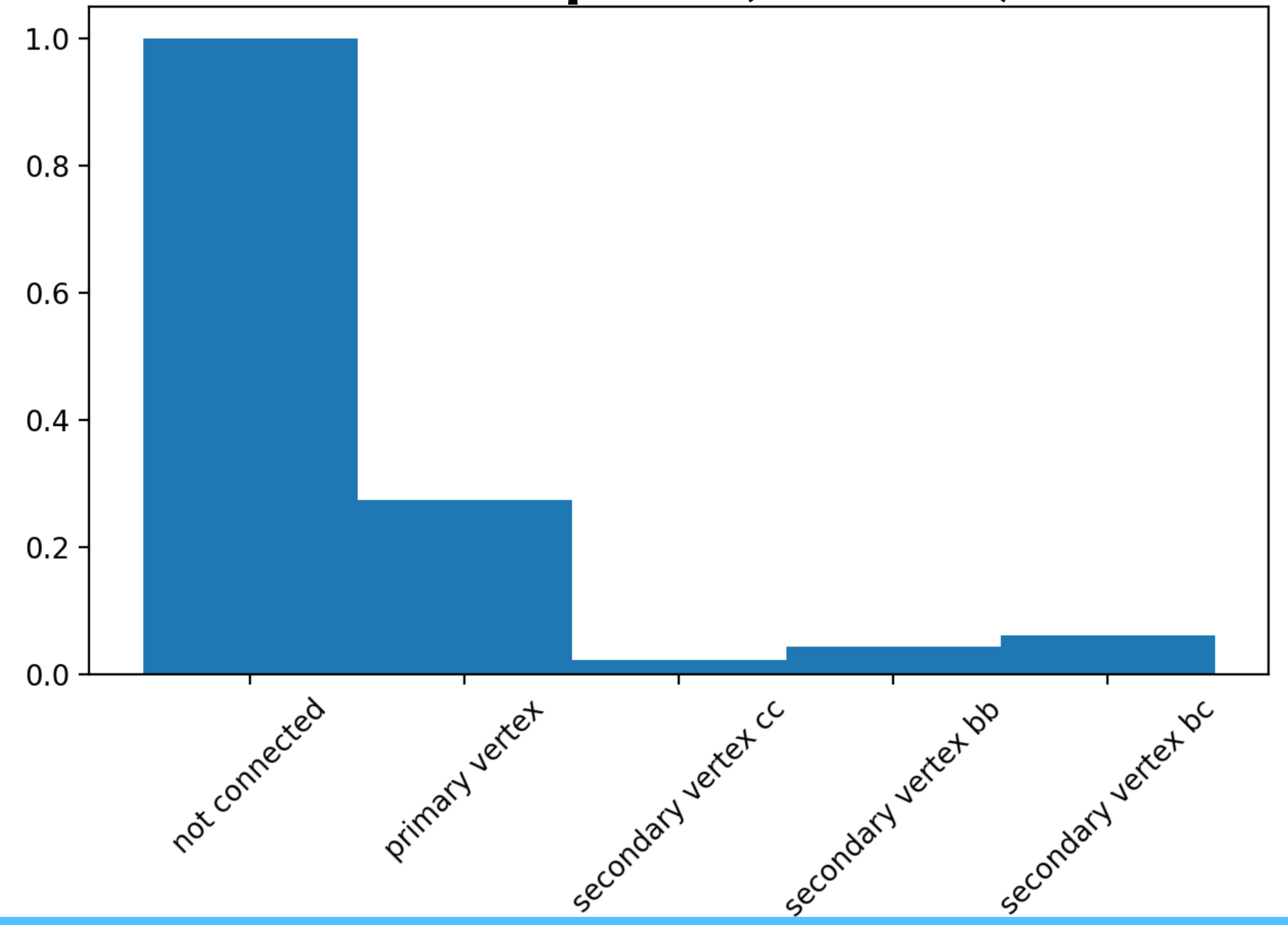
## Data property

- Using simulation data that the final state is $b\bar{b}$
- The labels of training data are created from Monte Carlo
- Using following variables as track information (22 variables)
  - Position and Momentum (Helix), Covariance Matrix
  - Charge
  - Energy

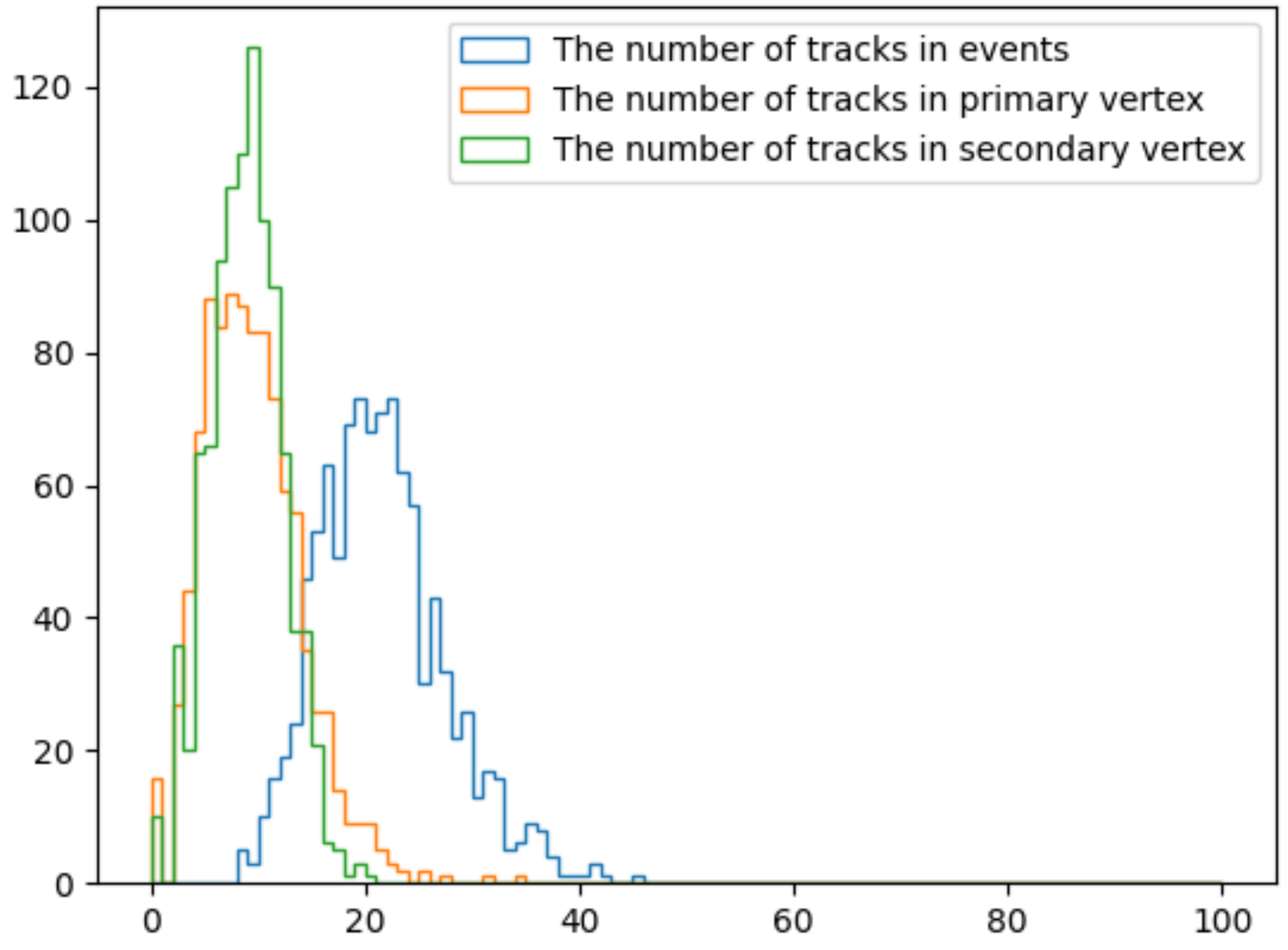
When the final state is $b\bar{b}$,
Secondary Vertex of flavor c are generated



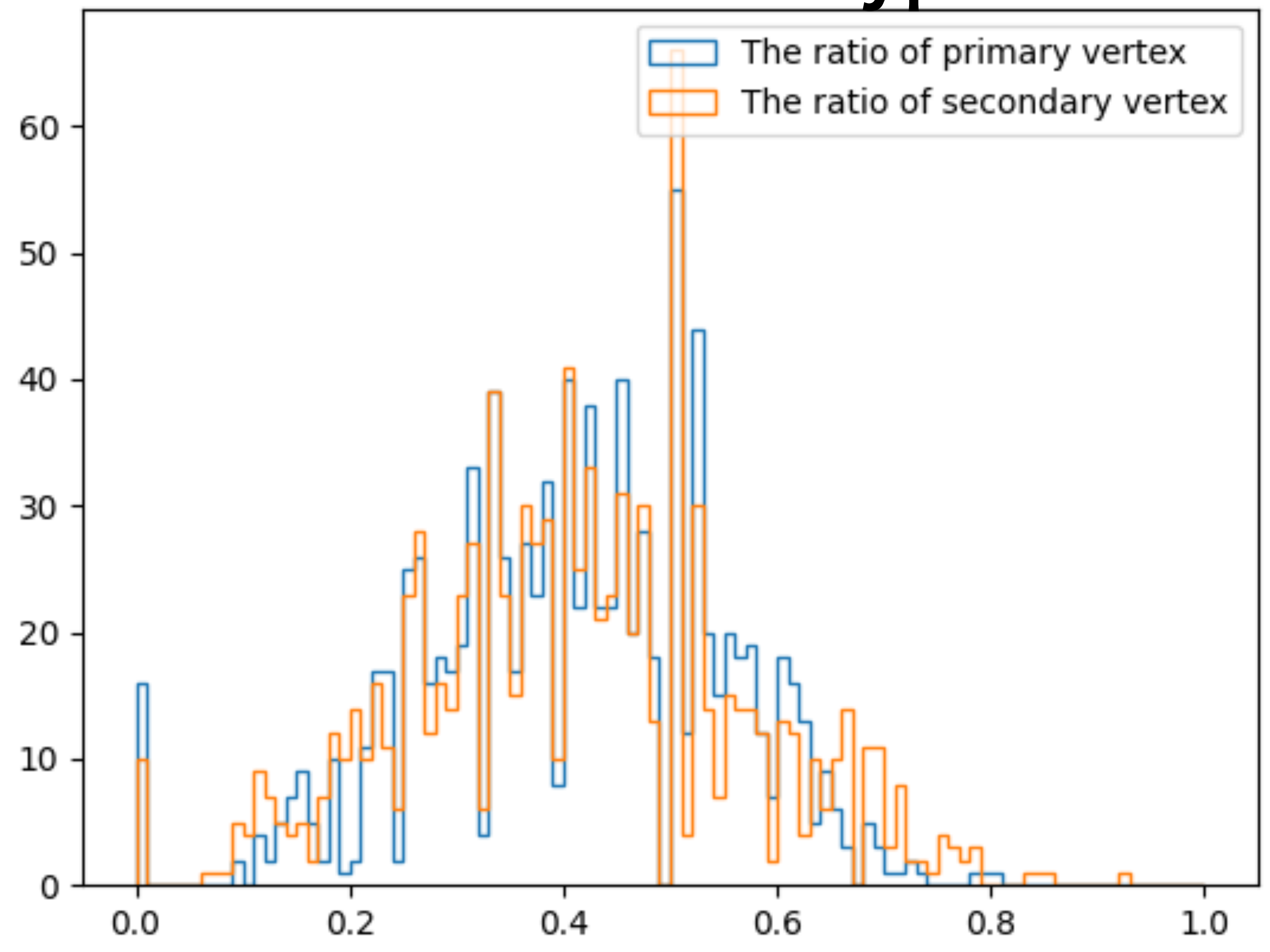

The ratio of the number of samples for the track pair (vertex) class




The number of tracks in a event

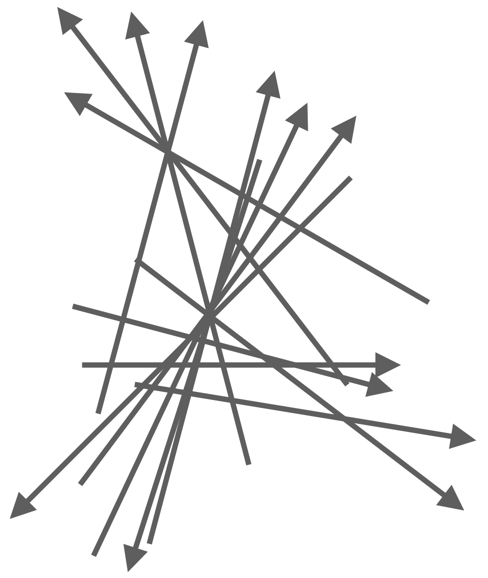



The rate of the number of tracks in each vertex type



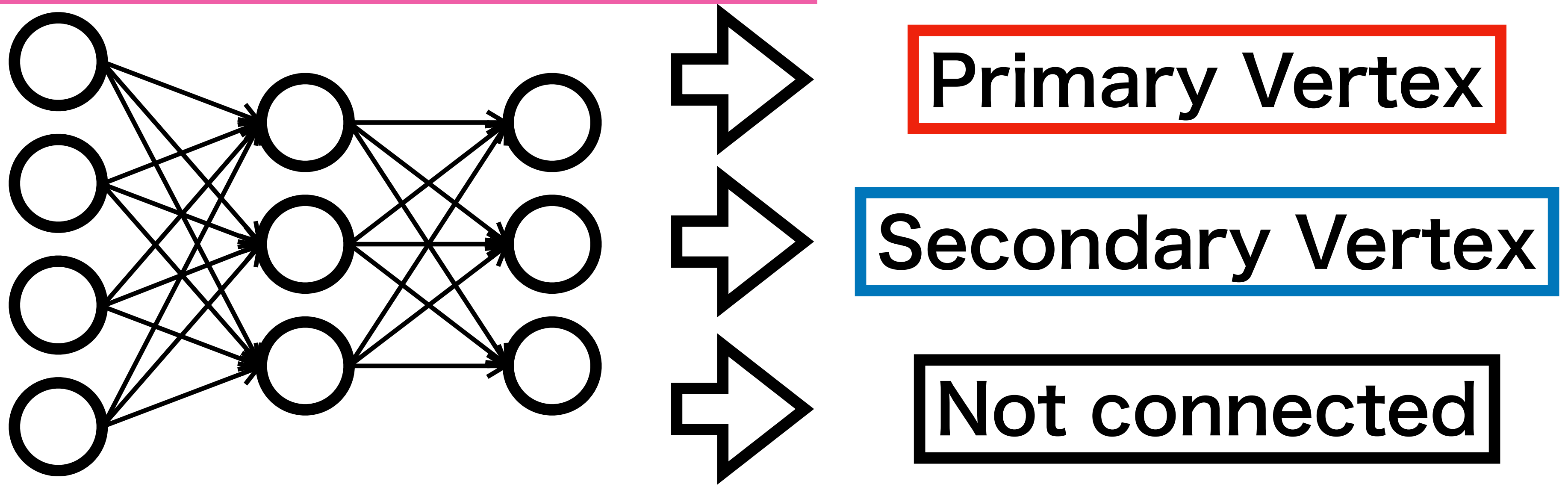
# 2. Networks

## Overview



All combination of track pairs
[Track 0, Track 1],
[Track 0, Track 2],
[Track 0, Track 3], …

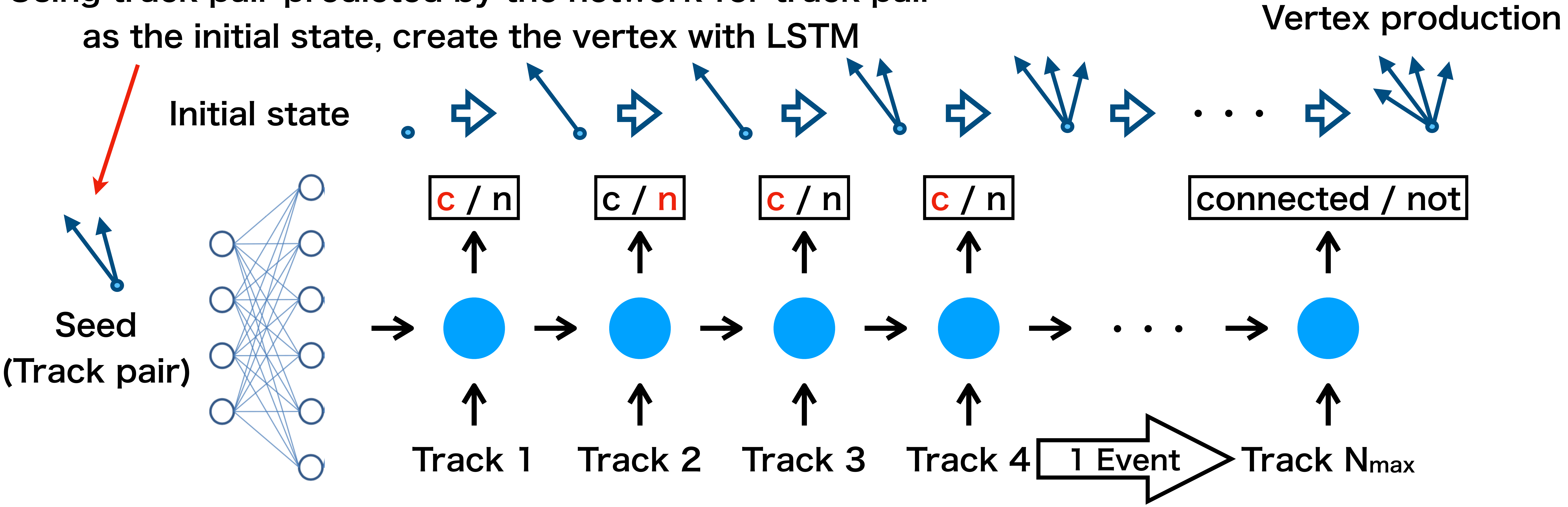
The network for track pair




The network for any number of tracks

Using track pair predicted by the network for track pair as the initial state, create the vertex with LSTM

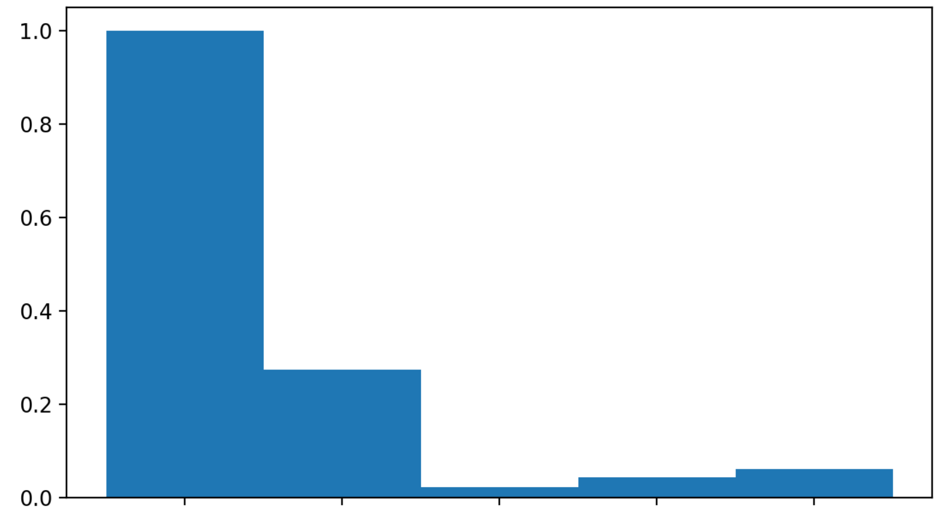
Using connected pair (Primary and Secondary) as the initial state

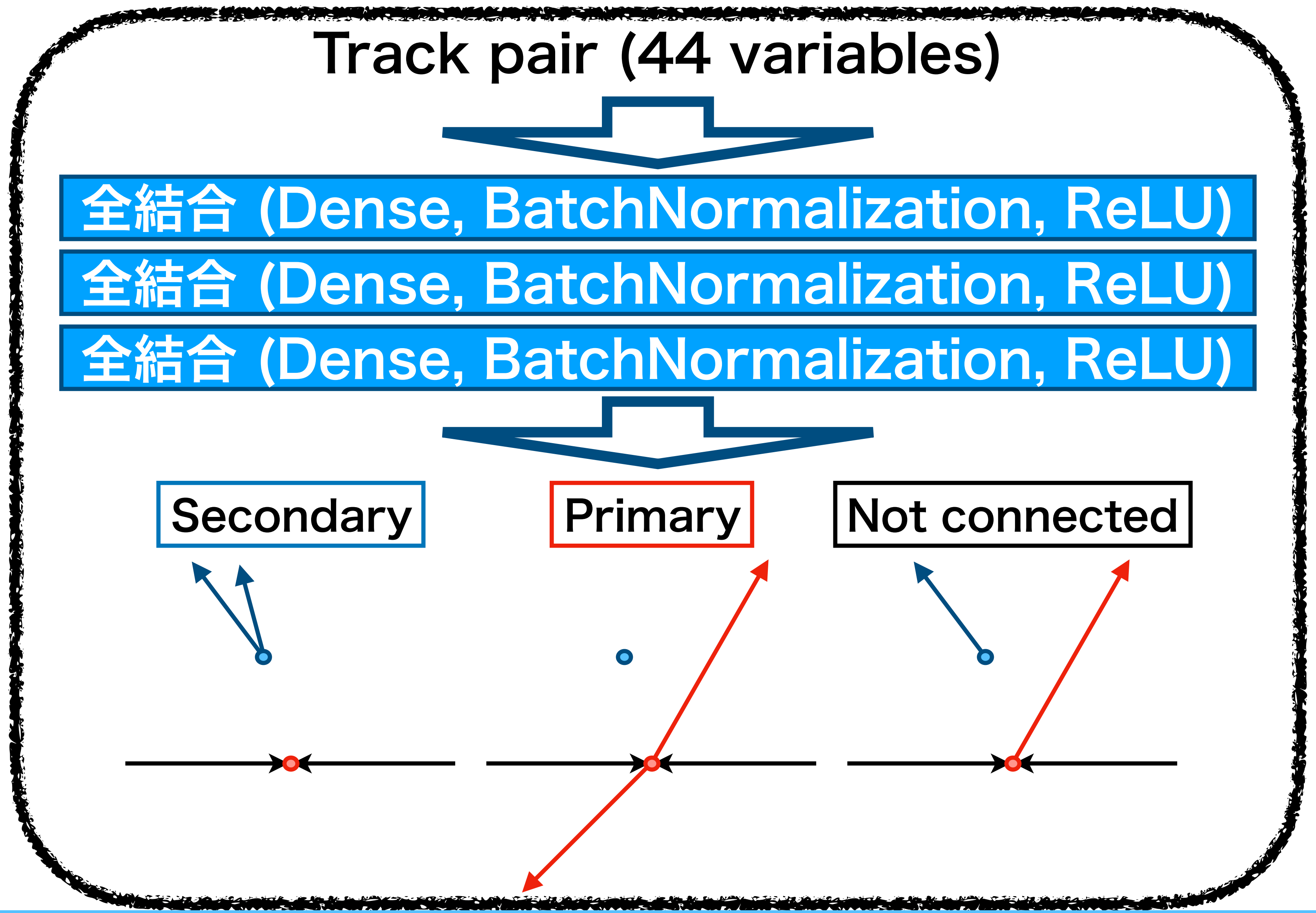


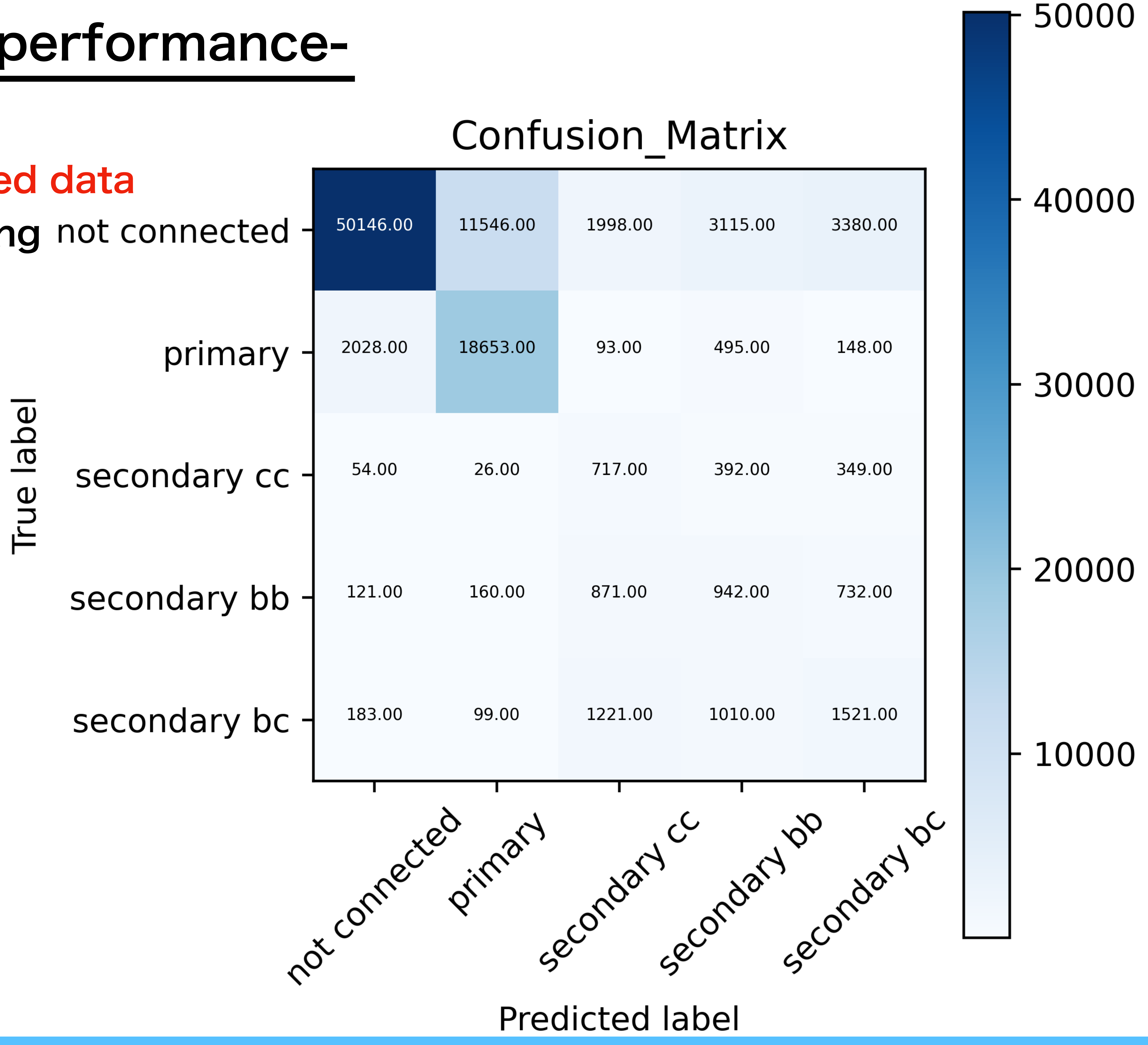
# 2. Networks

## The model for track pairs -structure and performance-

- Use the simply network
- Weighted the loss function because of the imbalanced data
  - Loss function : Evaluation function using to training



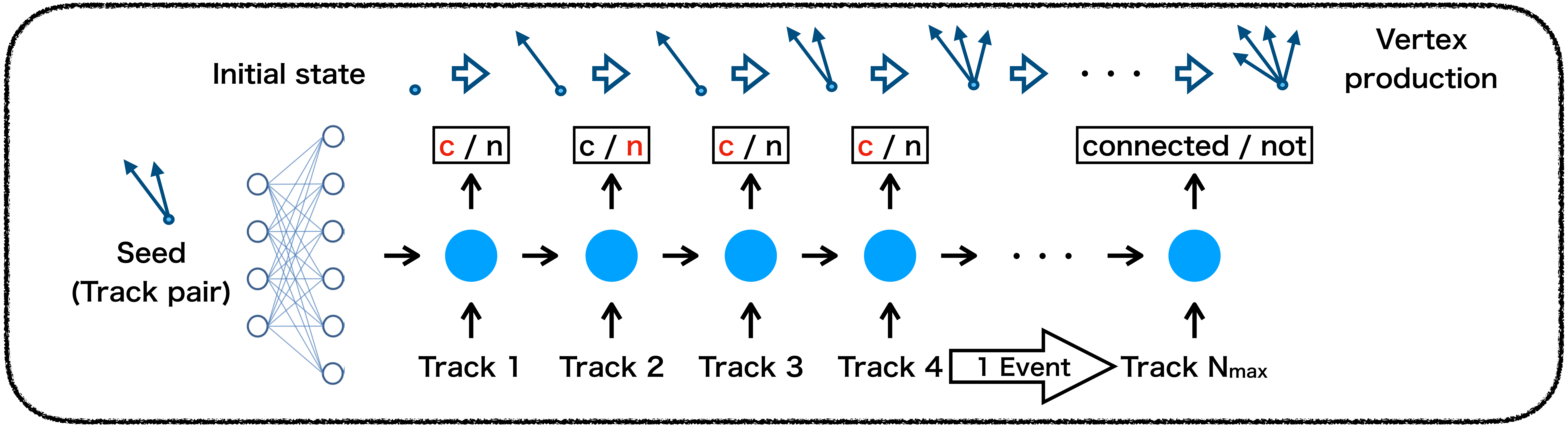



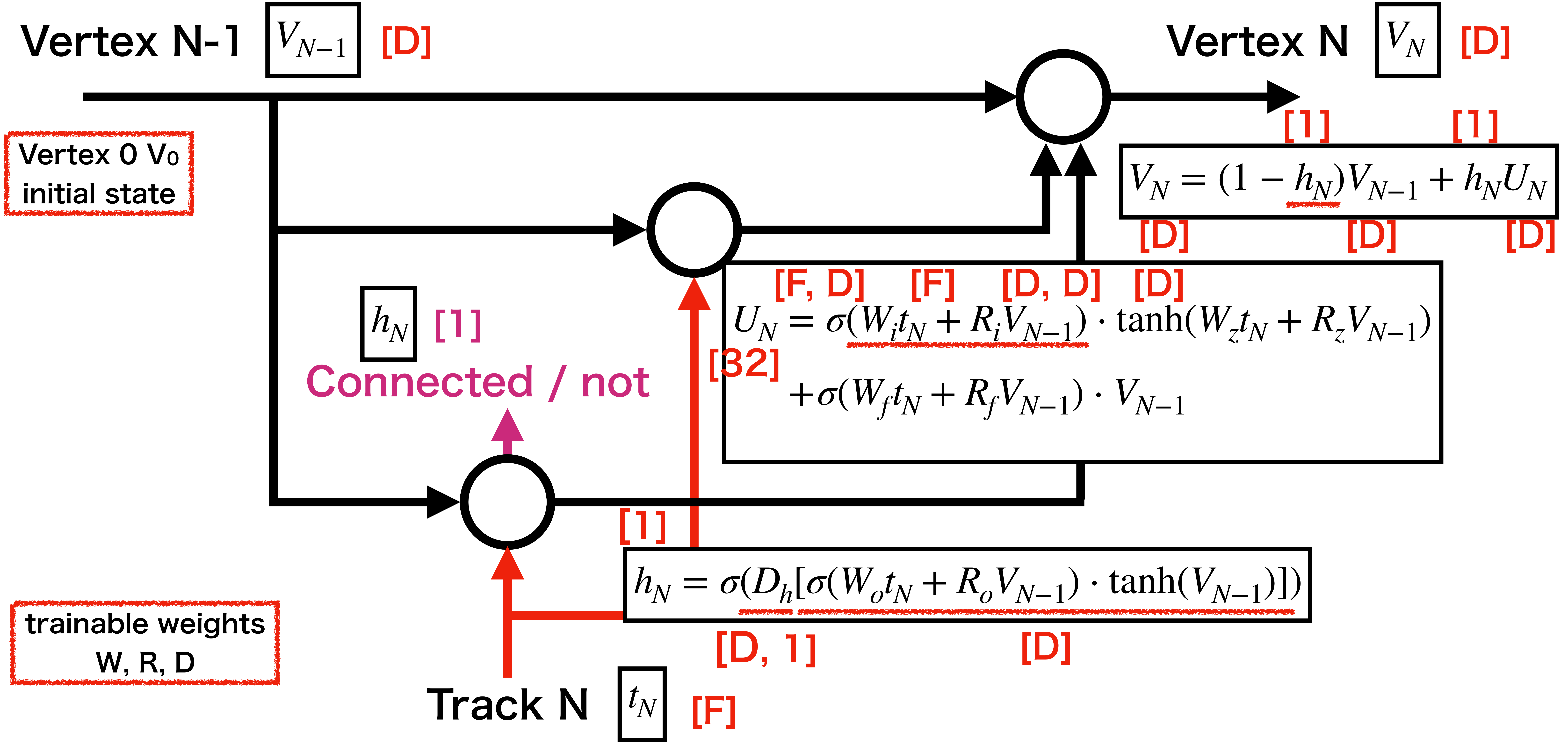
# 2. Networks

## Approach using the LSTM (Long short-term memory)

- I want to construct the network that can process over two tracks
- Points
  - The number of tracks included a event are different
  - The number of vertices included a event are different
  - ➡ Networks for variable length (Recurrent Neural Networks) are required
- Whether tracks are connected to the initial state (seed) : Trainable initial state



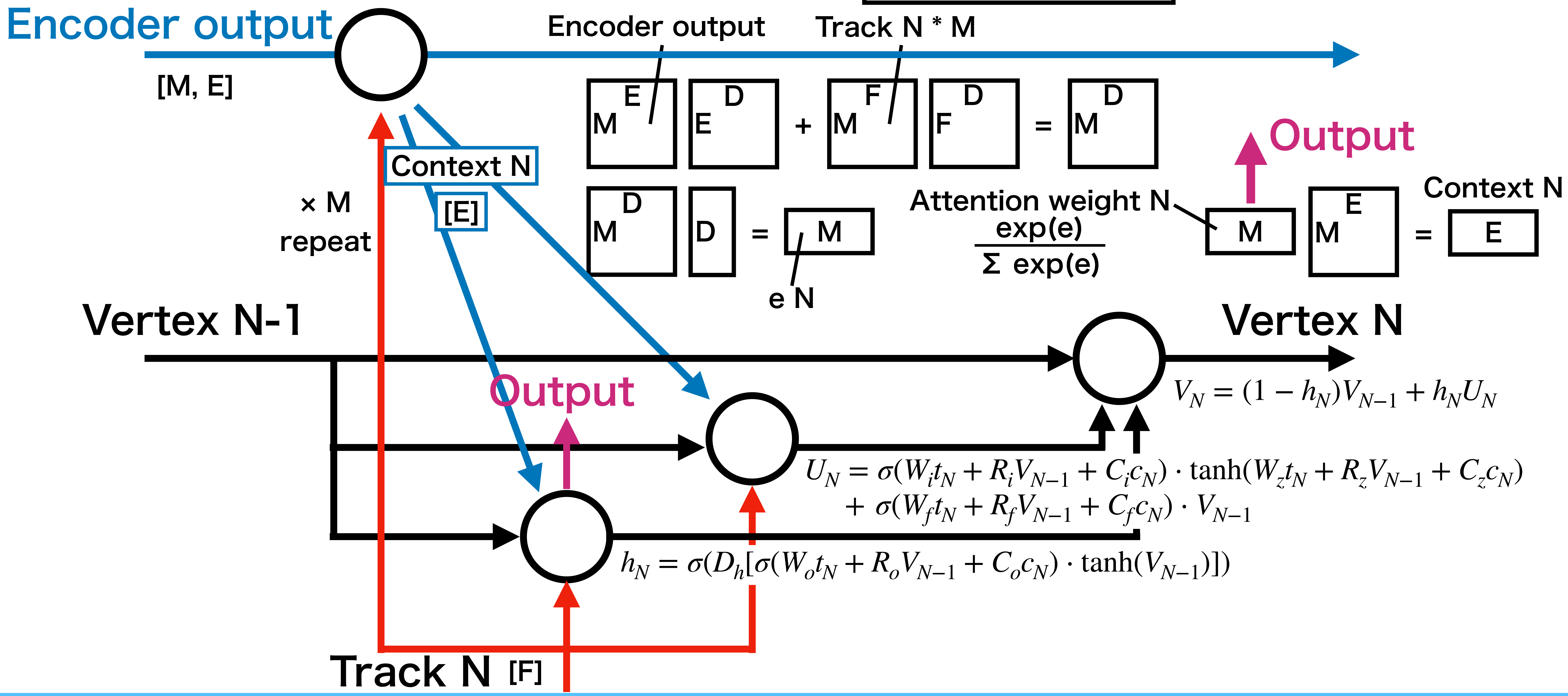
# 2. Networks



Vertex N-1
$V_{N-1}$
[D]
Vertex N
$V_N$
[D]
Vertex 0 V0
initial state
[1]
[1]
$V_N = (1 - h_N)V_{N-1} + h_N U_N$
[D]
[D]
[D]
[F, D]
[F]
[D, D]
[D]
$U_N = \sigma(W_i t_N + R_i V_{N-1}) \cdot \tanh(W_z t_N + R_z V_{N-1})$
$+\sigma(W_f t_N + R_f V_{N-1}) \cdot V_{N-1}$
[32]
$h_N$
[1]
Connected / not
[1]
$h_N = \sigma(D_h[\sigma(W_o t_N + R_o V_{N-1}) \cdot \tanh(V_{N-1})])$
[D, 1]
[D]
trainable weights
W, R, D
Track N
$t_N$
[F]

# 2. Networks

## Attention LSTM for Vertex Finder




$V_N = (1 - h_N)V_{N-1} + h_N U_N$

$U_N = \sigma(W_i t_N + R_i V_{N-1} + C_i c_N) \cdot \tanh(W_z t_N + R_z V_{N-1} + C_z c_N)$
$+ \sigma(W_f t_N + R_f V_{N-1} + C_f c_N) \cdot V_{N-1}$

$h_N = \sigma(D_h[\sigma(W_o t_N + R_o V_{N-1} + C_o c_N) \cdot \tanh(V_{N-1})])$

# 3. Vertex Finding

All (31) tracks

[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 27 28 29 30]

True Others

[1, 13, 22, 29]

```
True Primary Vertex
[3, 4, 6, 7, 8, 11, 12, 15, 16, 18, 19, 20, 21, 23, 25, 27, 28, 30]
Predict Primary Vertex
[3, 4, 6, 7, 8, 11, 12, 15, 16, 18, 19, 20, 21, 23, 25, 27, 28, 29, 30]
True Secondary Vertex Chain 1
cc : [0, 2, 14]
bb : [5, 10, 17]
one track : []
True Secondary Vertex Chain 2
cc : [24, 26]
bb : []
one track : [9]
Predict Secondary Vertex 0
[24, 26]
Predict Secondary Vertex 1
[2, 10]
Predict Secondary Vertex 2
[5, 17]
Predict Secondary Vertex 3
[0, 14]
------------------------------------------------------------------------------------------
MC Primary / Reco SV : 0.0
MC Others  / Reco SV : 0.0
MC Bottom  / Reco SV : 1.0 Same Chain : 1.0 Same Particle : 0.6666666666666666
MC Charm   / Reco SV : 1.0 Same Chain : 1.0 Same Particle : 0.8
------------------------------------------------------------------------------------------
```