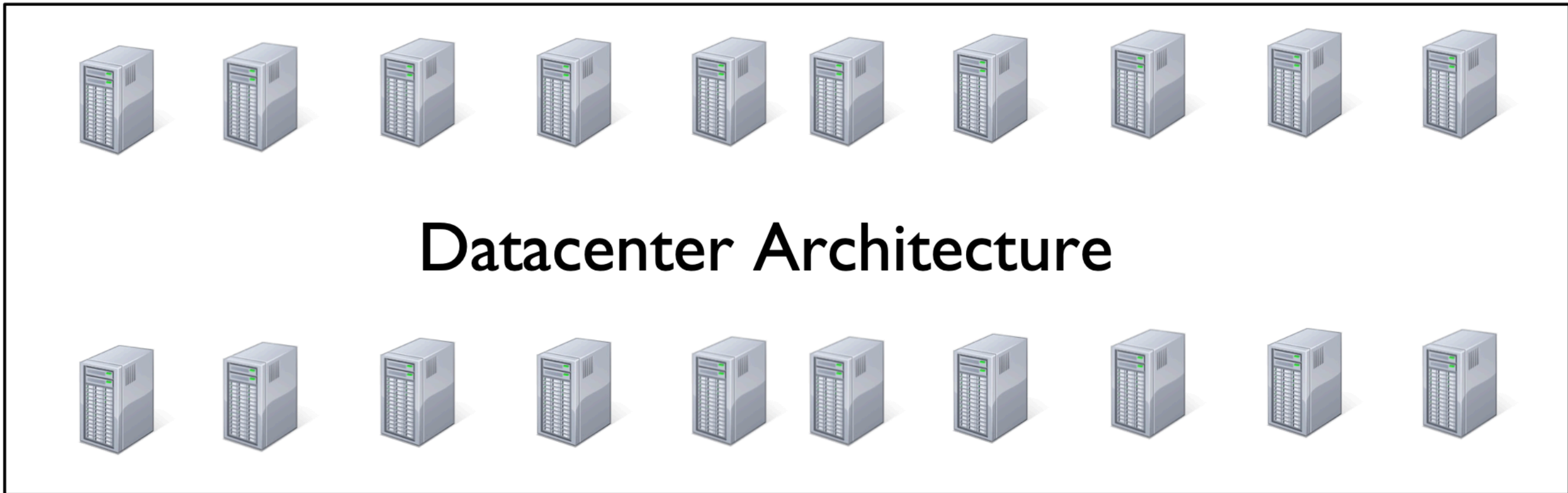


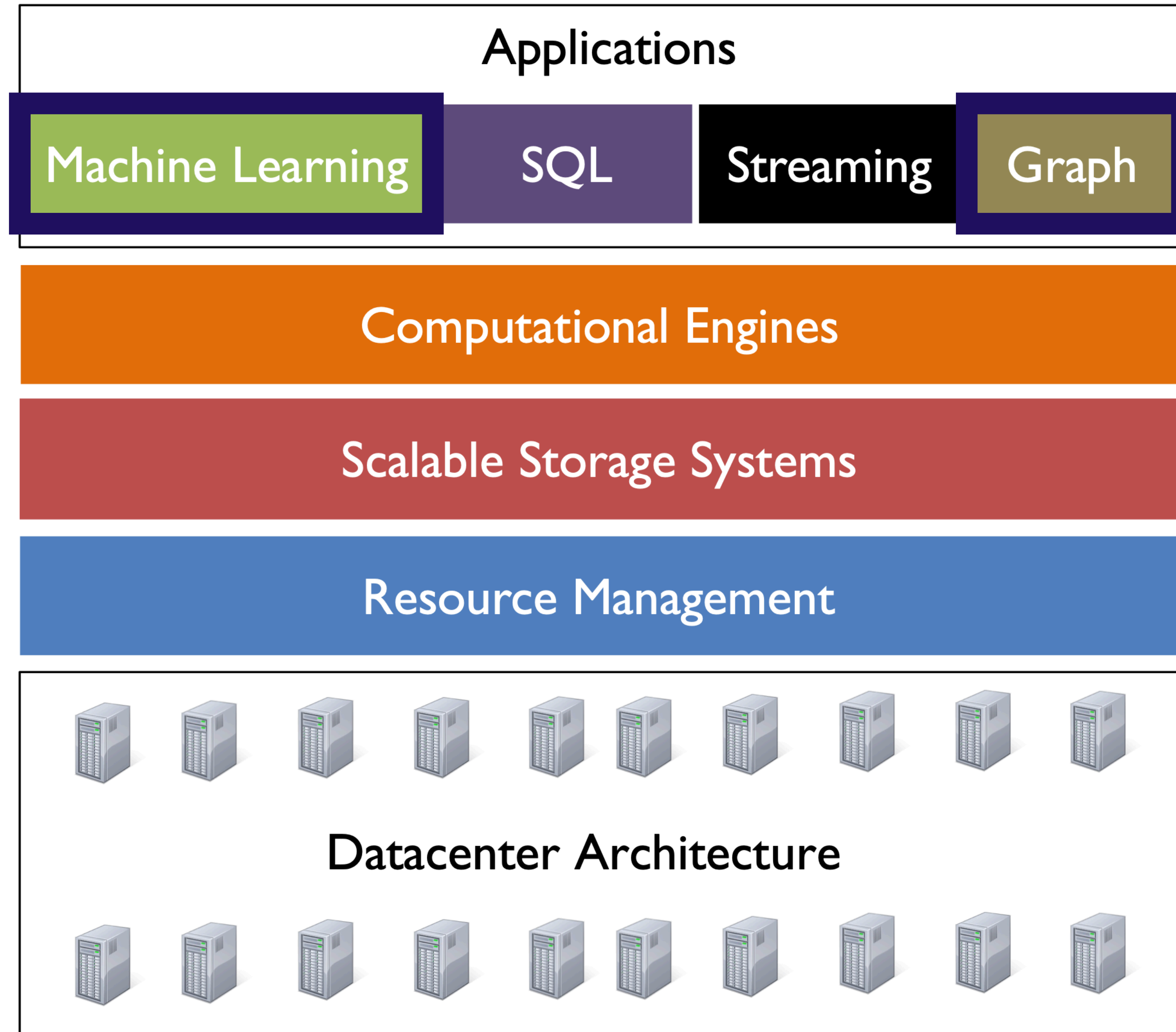
# CS 744: Marius

Jason Mohoney  
Spring 2025

# Administrivia

- Shivaram gone for the week.
- Project check-ins pushed back. **Due April 7th**





# Graph Machine Learning

A uniquely challenging workload

## Graph Processing

Skewed Access

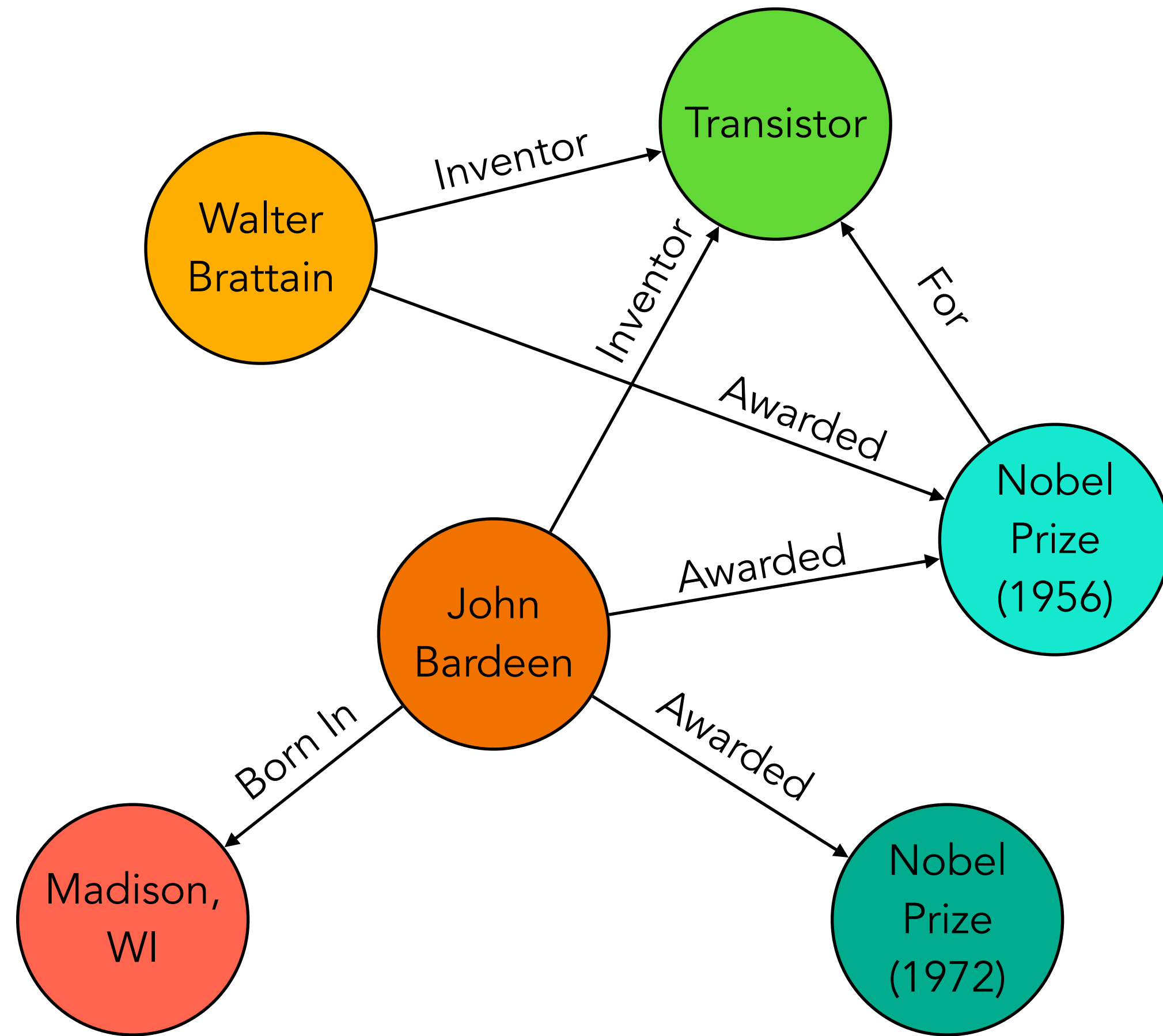
Random Access

## Machine Learning

Compute Heavy

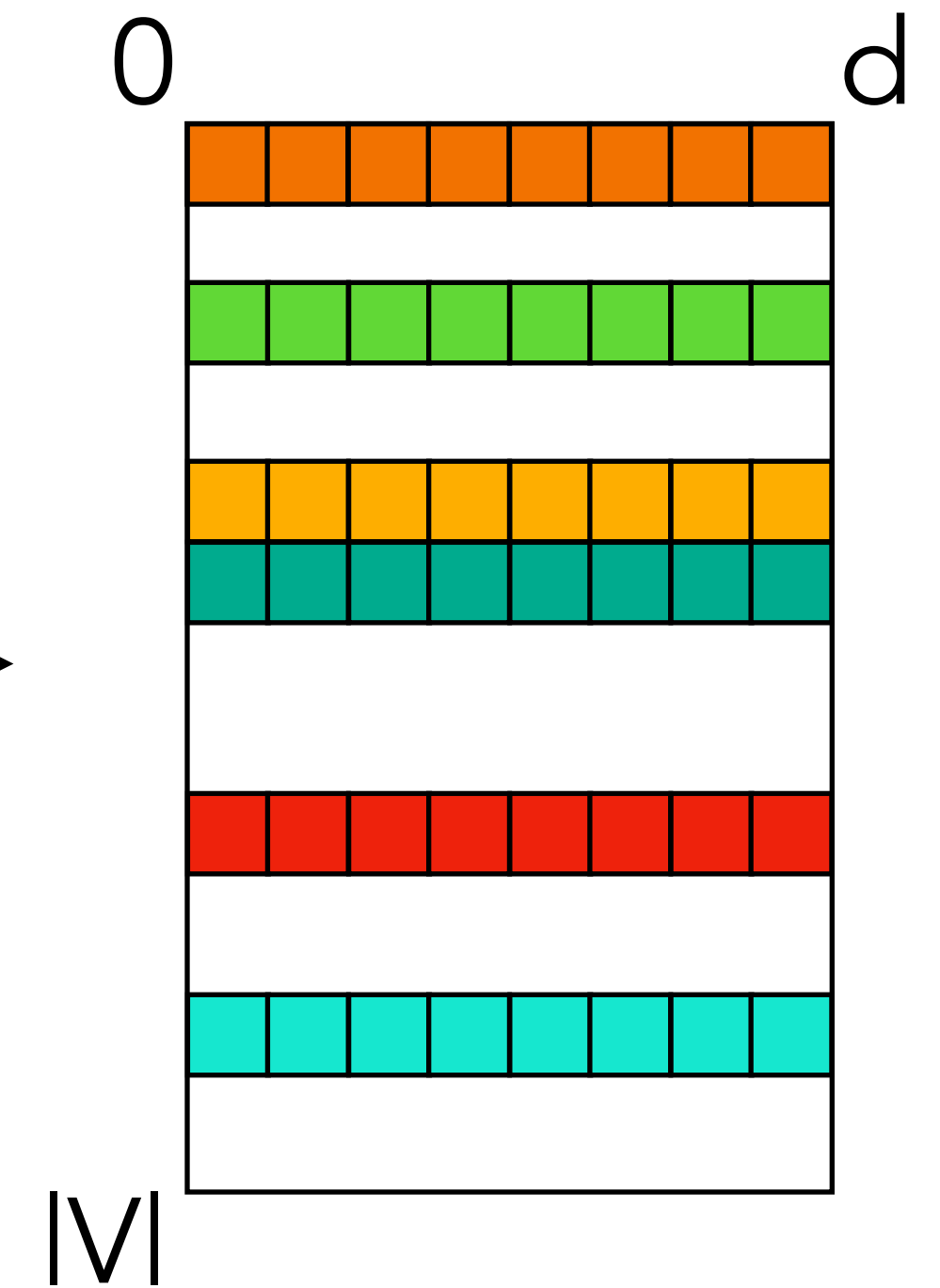
Memory Heavy

# Graph Embedding



Knowledge Graph:  $G = (V, R, E)$

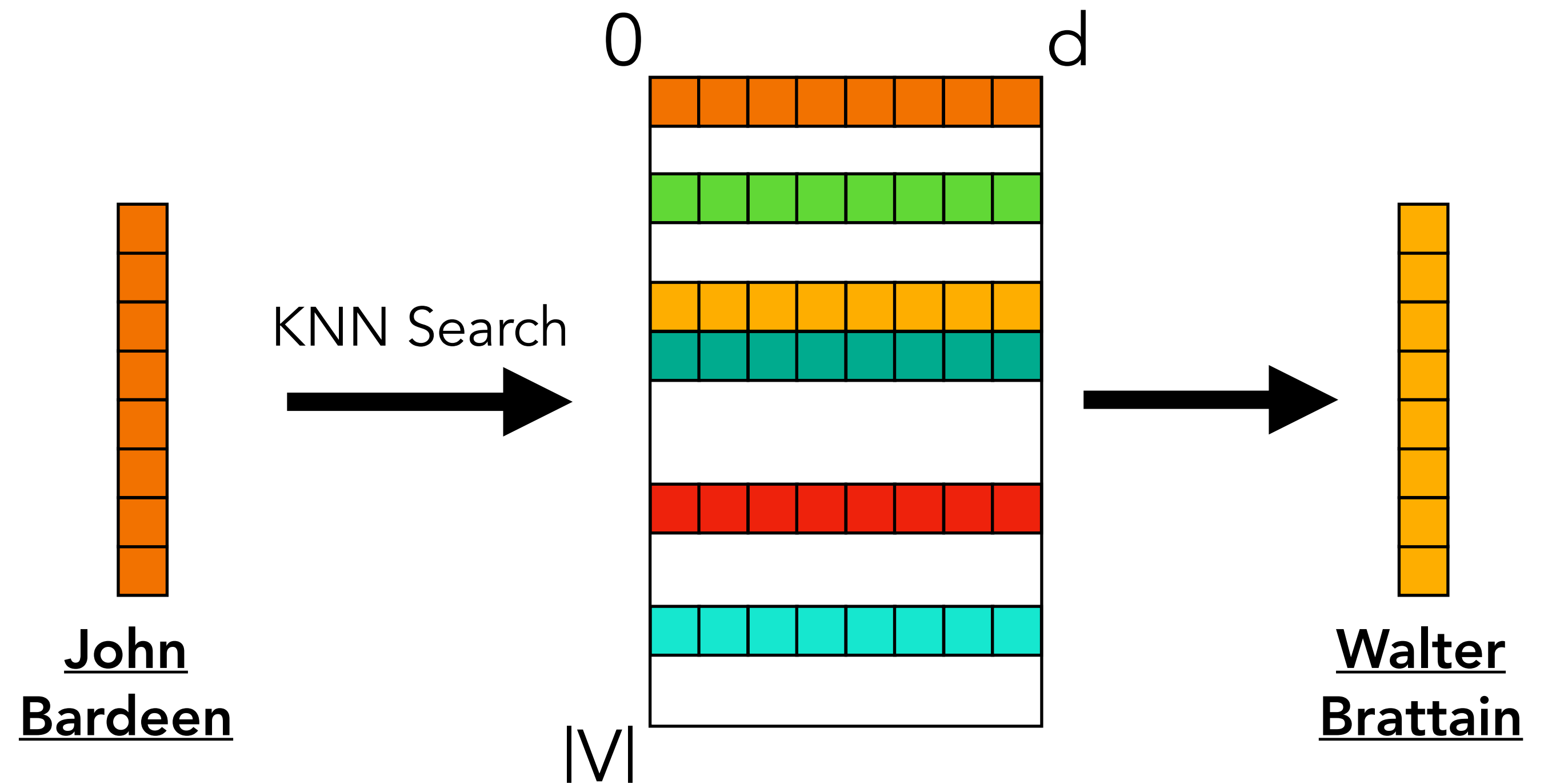
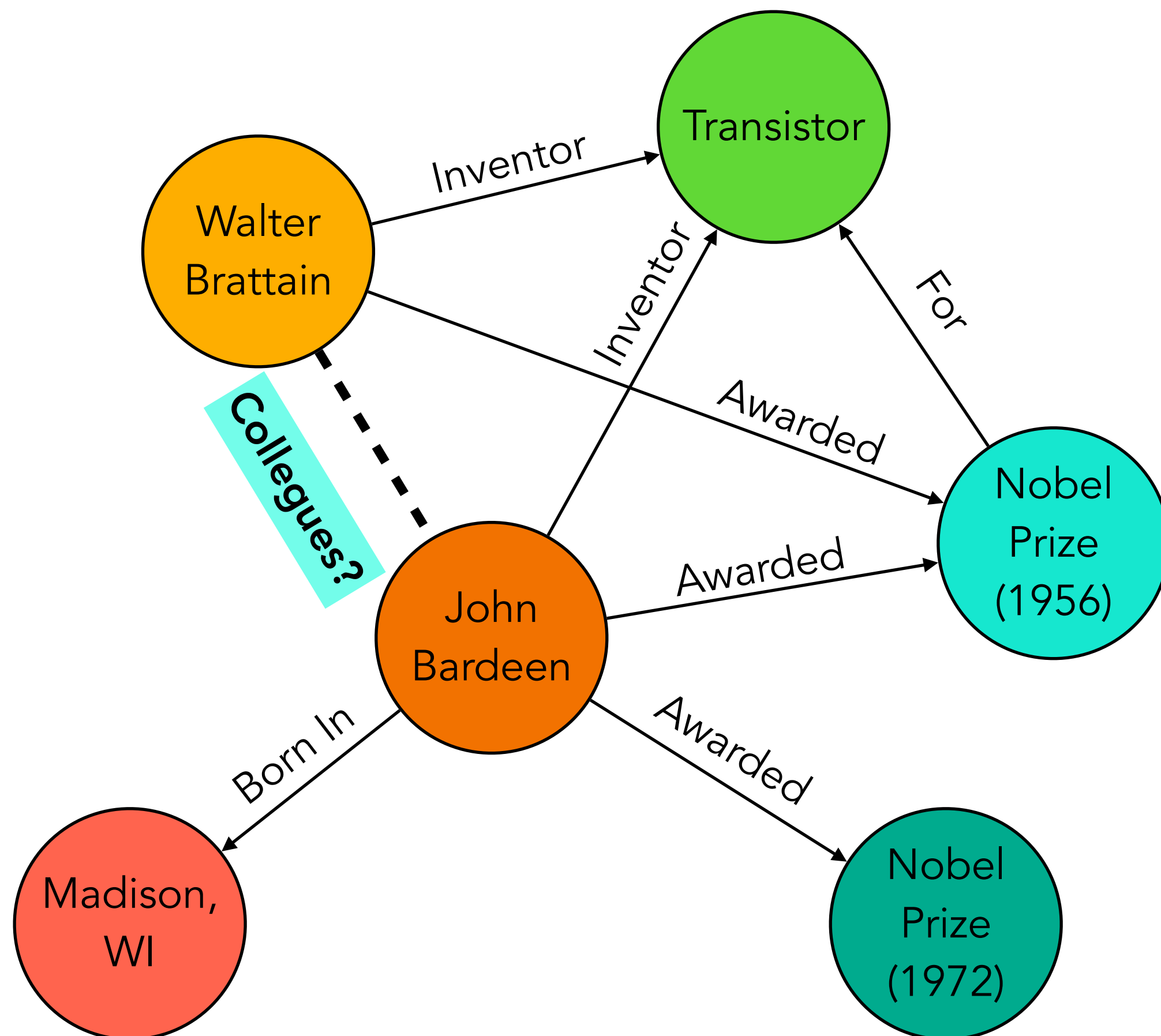
Learn Representations!



Node Embedding Table

# Application: Link Prediction

Goal: Predict missing edges in the graph



# Training Graph Embeddings

Contrastive Learning over edges

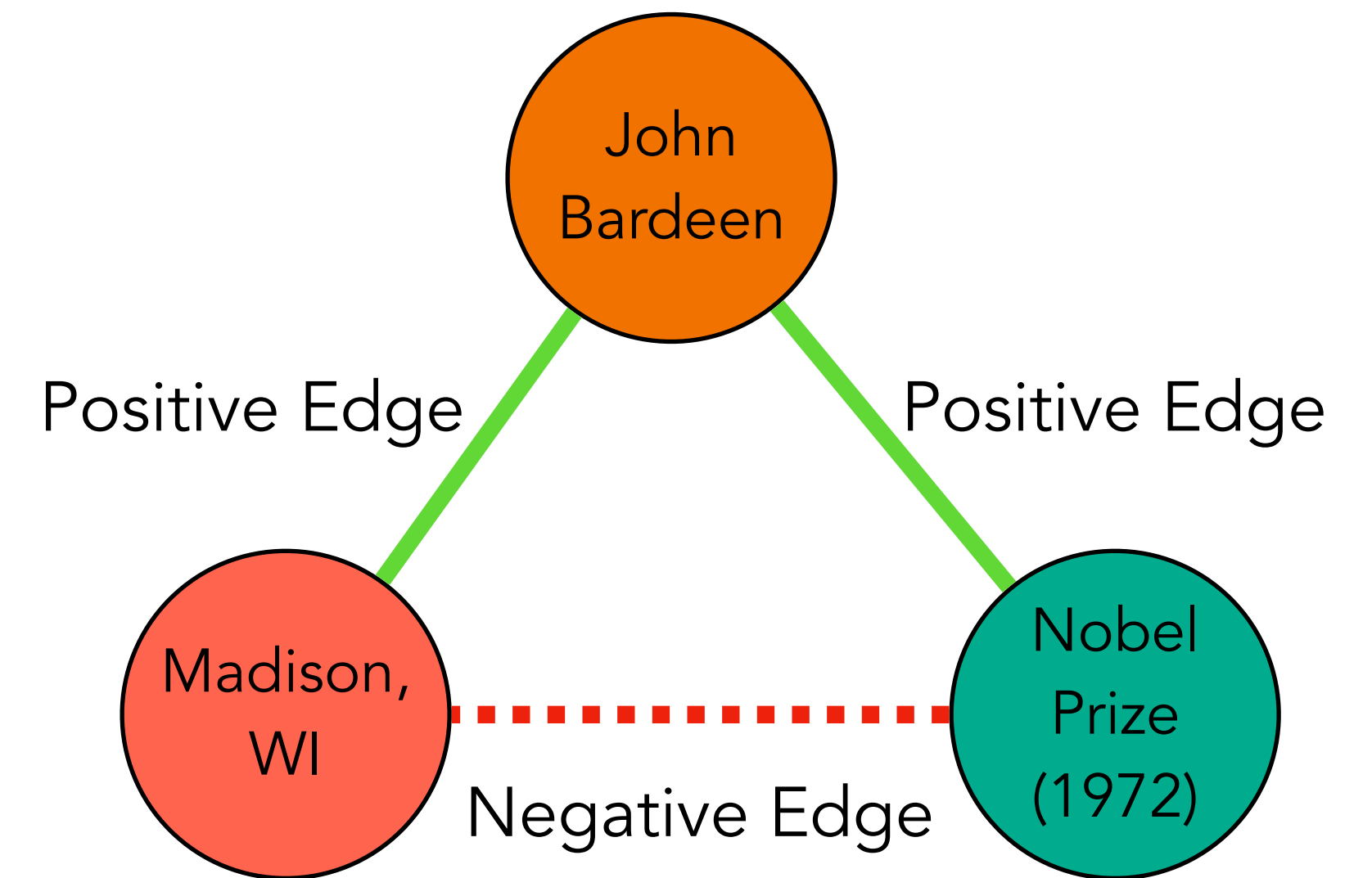
**Maximize** score for **positive** edges:  $e \in E$

**Minimize** score for **negative** edges:  $e' \notin E$

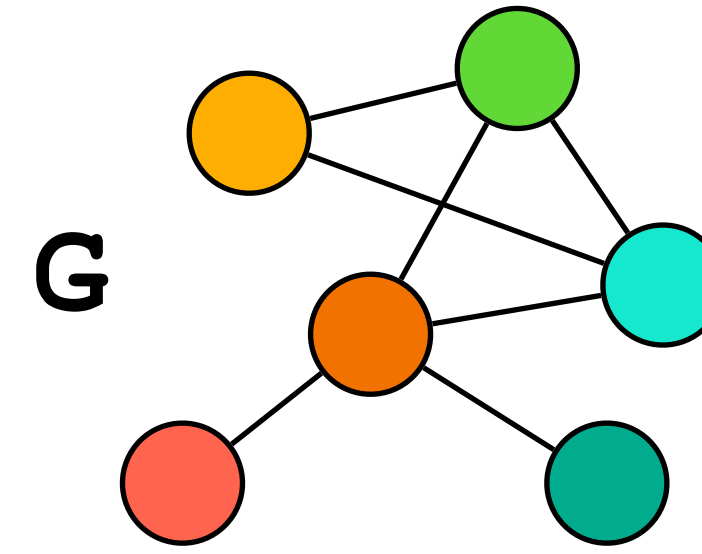
$$\mathcal{L} = \sum_{e \in E} \sum_{e' \in S'_e} MSE(f(e), f(e'))$$

DistMult: Score Function

$$f(e) = (\theta_s \odot \theta_r) \cdot \theta_d$$



# Training Algorithm



```
 $\Theta$  = initEmbeddings(G)
```

```
for i range(num_batches):
```

```
    B = getBatch(i)
```

```
     $\theta$  = getEmbeddings(B,  $\Theta$ )
```

```
     $\Phi$  = computeGrads(B,  $\theta$ )
```

```
    updateEmbeddings( $\Theta$ ,  $\Phi$ )
```

# Training Algorithm

$\Theta = \text{initEmbeddings}(G)$

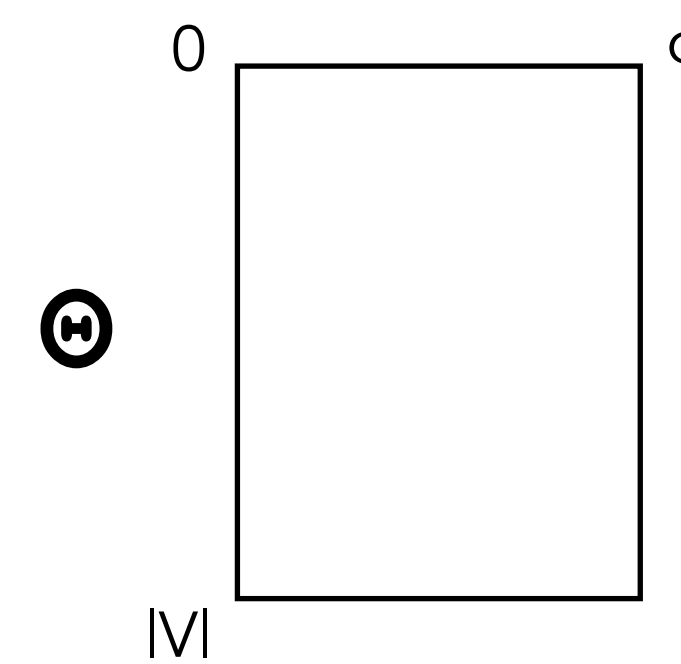
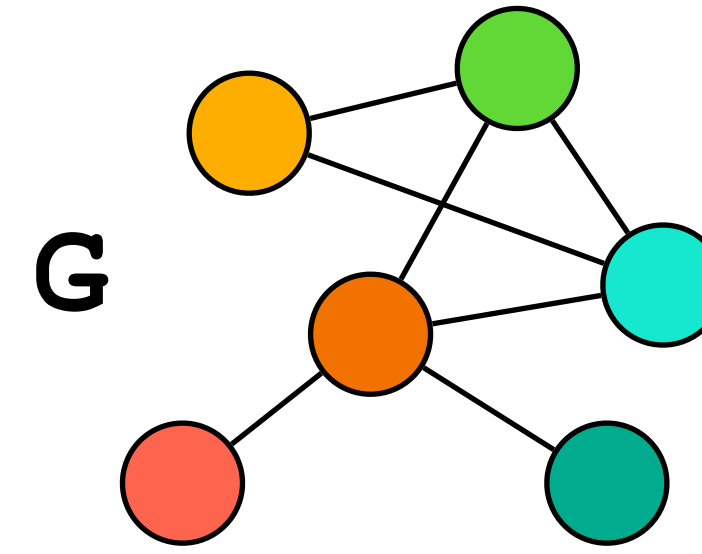
for  $i$  range(num\_batches):

$B = \text{getBatch}(i)$

$\theta = \text{getEmbeddings}(B, \Theta)$

$\Phi = \text{computeGrads}(B, \theta)$

$\text{updateEmbeddings}(\Theta, \Phi)$



# Training Algorithm

```
 $\Theta = \text{initEmbeddings}(G)$ 
```

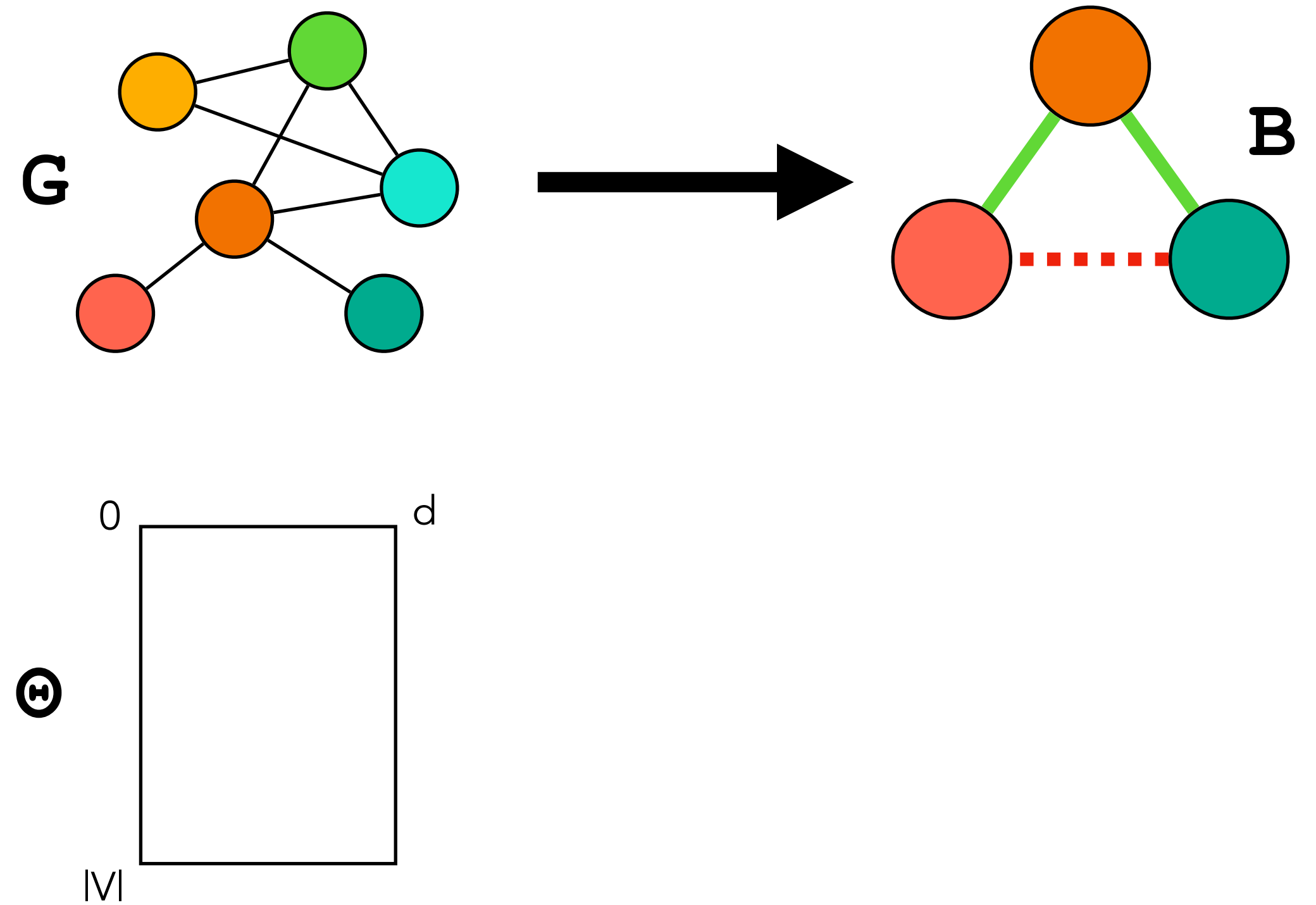
```
for i range(num_batches):
```

```
    B = getBatch(i)
```

```
     $\theta = \text{getEmbeddings}(B, \Theta)$ 
```

```
     $\Phi = \text{computeGrads}(B, \theta)$ 
```

```
     $\text{updateEmbeddings}(\Theta, \Phi)$ 
```



# Training Algorithm

```
 $\Theta = \text{initEmbeddings}(G)$ 
```

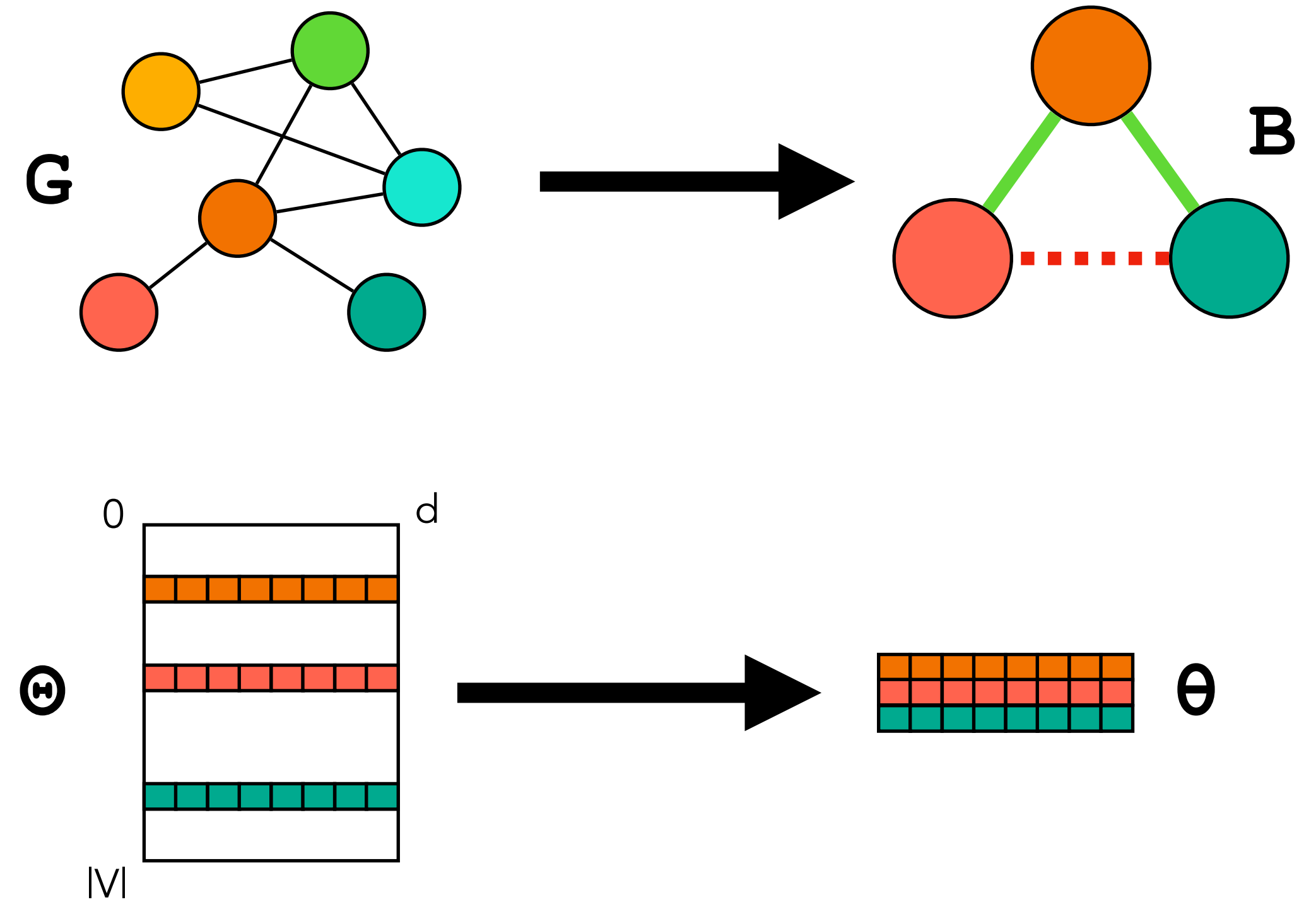
```
for i range(num_batches):
```

```
    B = getBatch(i)
```

```
     $\theta = \text{getEmbeddings}(B, \Theta)$ 
```

```
     $\Phi = \text{computeGrads}(B, \theta)$ 
```

```
    updateEmbeddings( $\Theta, \Phi$ )
```



# Training Algorithm

```
 $\Theta = \text{initEmbeddings}(G)$ 
```

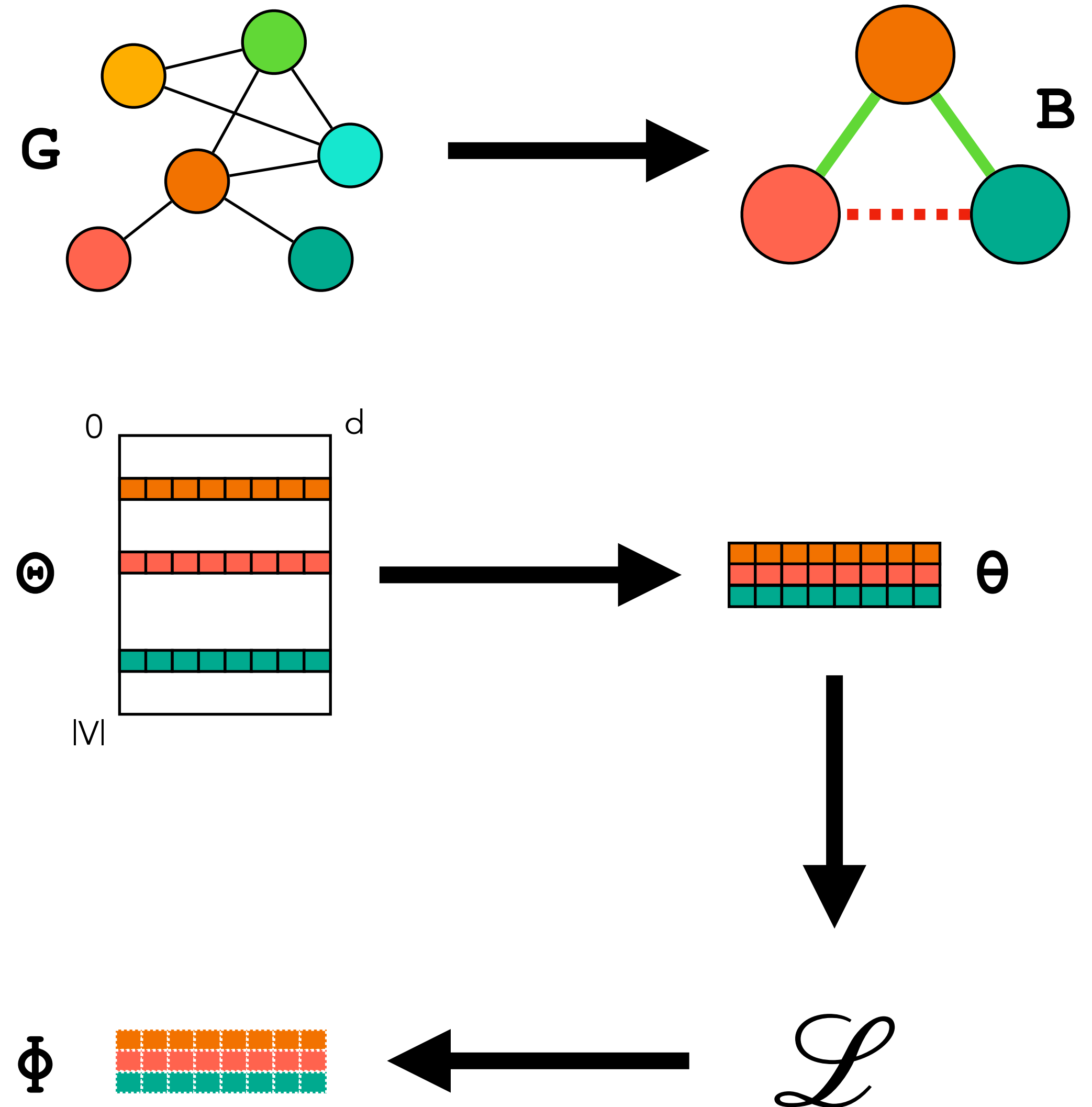
```
for i range(num_batches):
```

```
    B = getBatch(i)
```

```
     $\theta = \text{getEmbeddings}(B, \Theta)$ 
```

```
     $\Phi = \text{computeGrads}(B, \theta)$ 
```

```
    updateEmbeddings( $\Theta, \Phi$ )
```



# Training Algorithm

```
 $\Theta = \text{initEmbeddings}(G)$ 
```

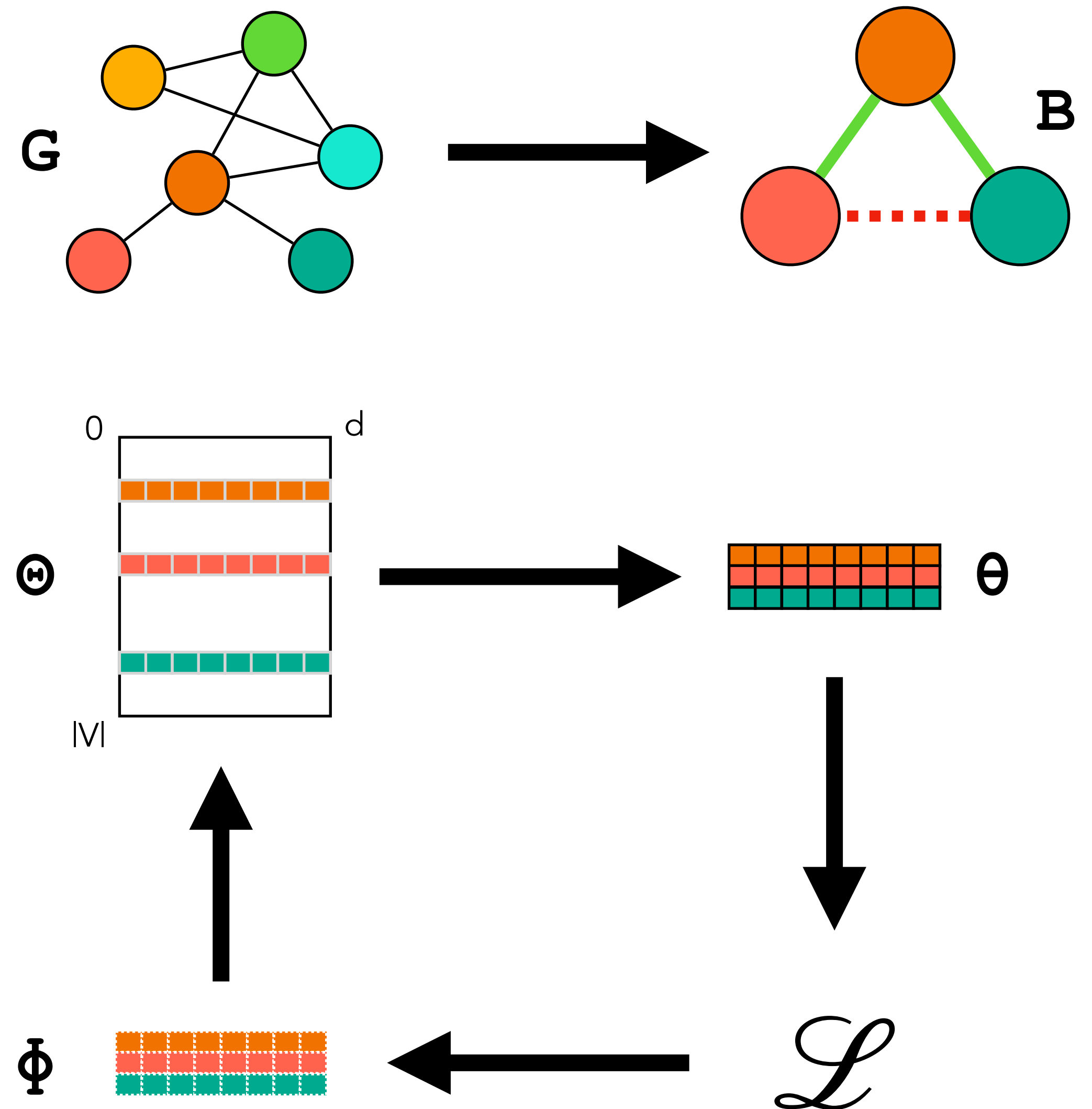
```
for i range(num_batches):
```

```
    B = getBatch(i)
```

```
     $\theta = \text{getEmbeddings}(B, \Theta)$ 
```

```
     $\phi = \text{computeGrads}(B, \theta)$ 
```

```
    updateEmbeddings( $\Theta, \phi$ )
```



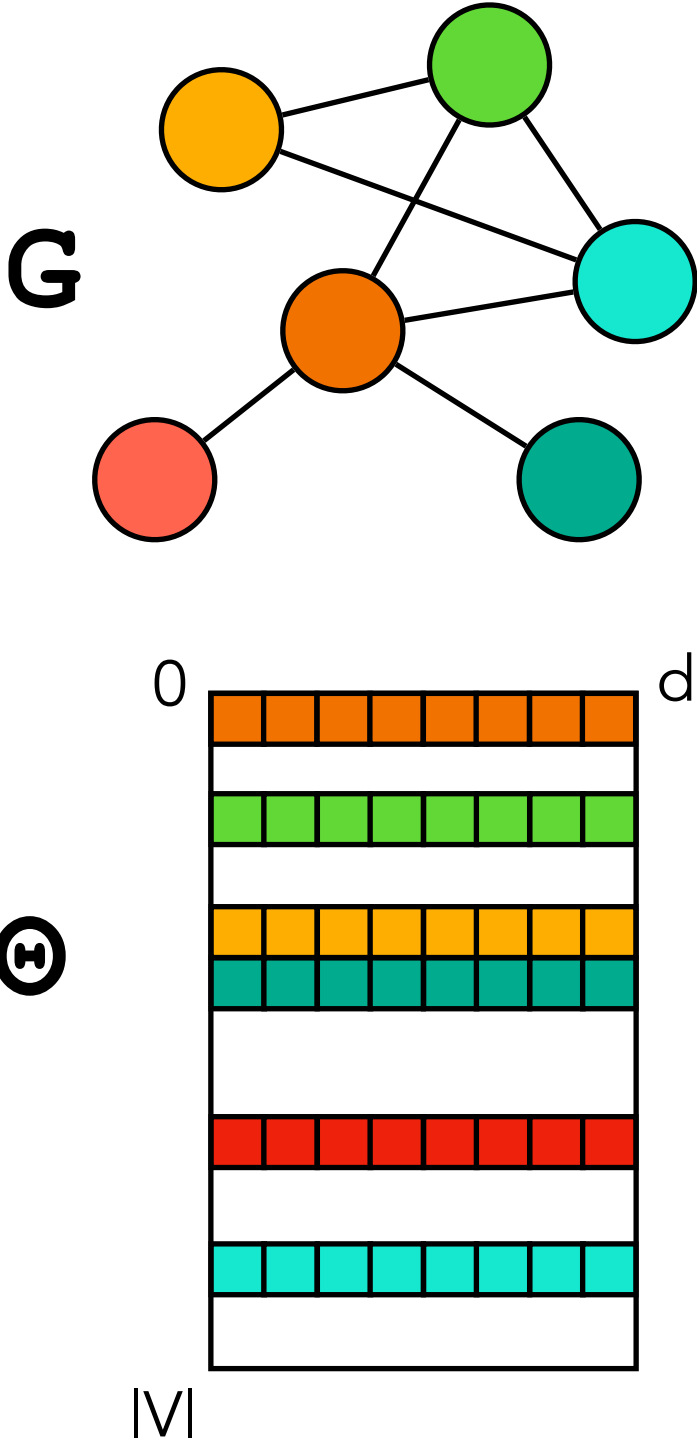
# Challenge: Scale

Example: Hyperlink-2012

Link structure of the internet circa 2012

$|V| = 3.5$  billion

$|E| = 128$  billion edges



Edge List Size  
 $|E| * 2 * 4 \text{ bytes} = \underline{1TB}$

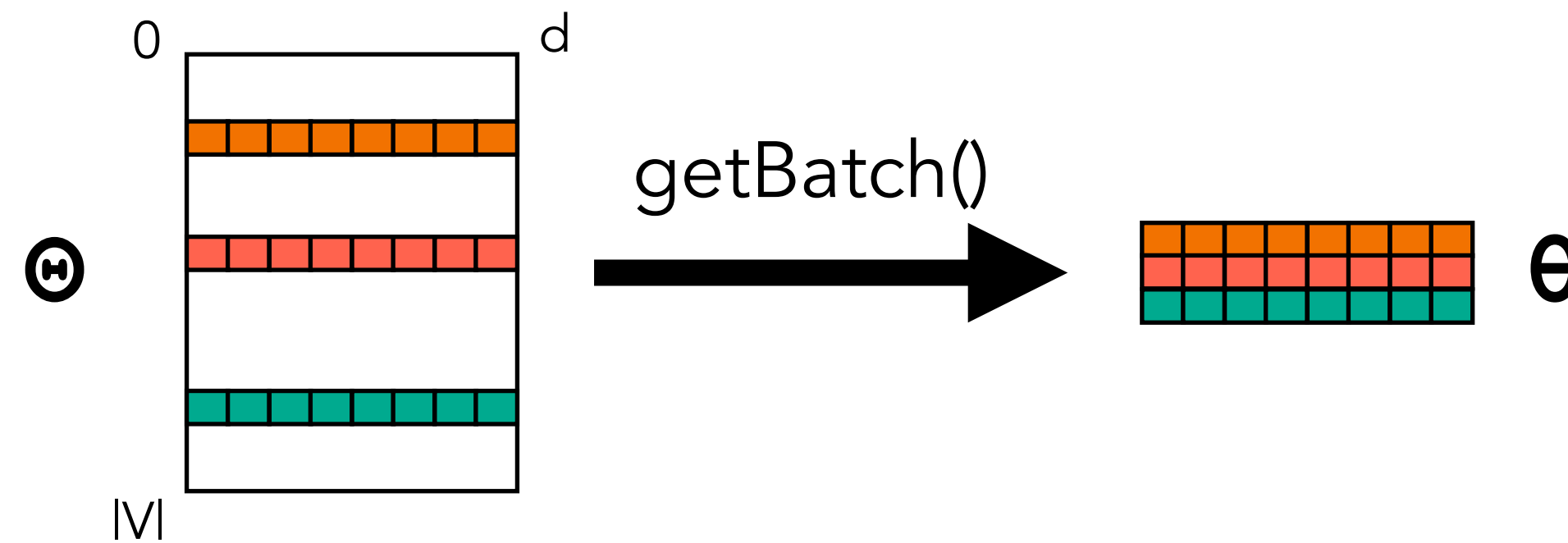
Embedding Table Size  
 $d=200$   
 $|V| * d * 4 \text{ bytes} = \underline{2.8TB}$

Large number of edges => Large amount of model computation => Need GPUs!

But edges and embeddings exceed GPU memory capacity!

# Challenge: Random Access

Need to perform random access of the embedding table when forming batches



Okay if embeddings are in GPU or CPU memory, but prohibitively slow on disk!

# Prior Solutions Bottlenecked by Data Movement

DGL-KE (Amazon)

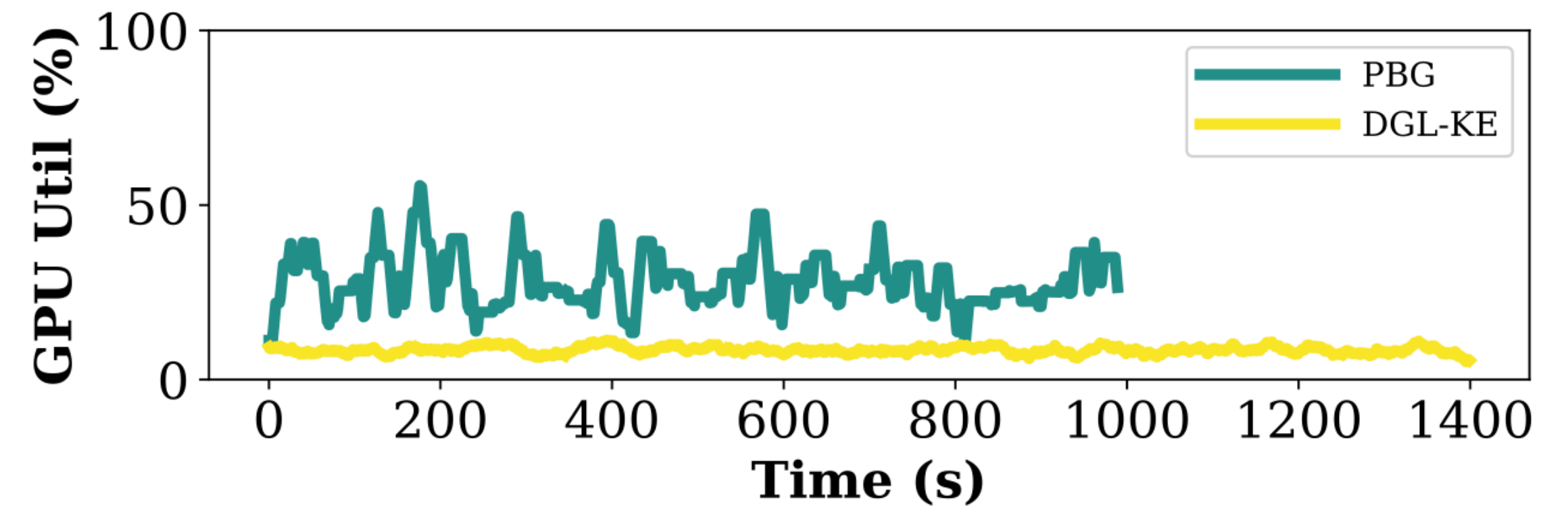


Figure 1: The GPU utilization of DGL-KE and PBG for one training epoch of ComplEx embeddings on the Freebase86m knowledge graph.

---

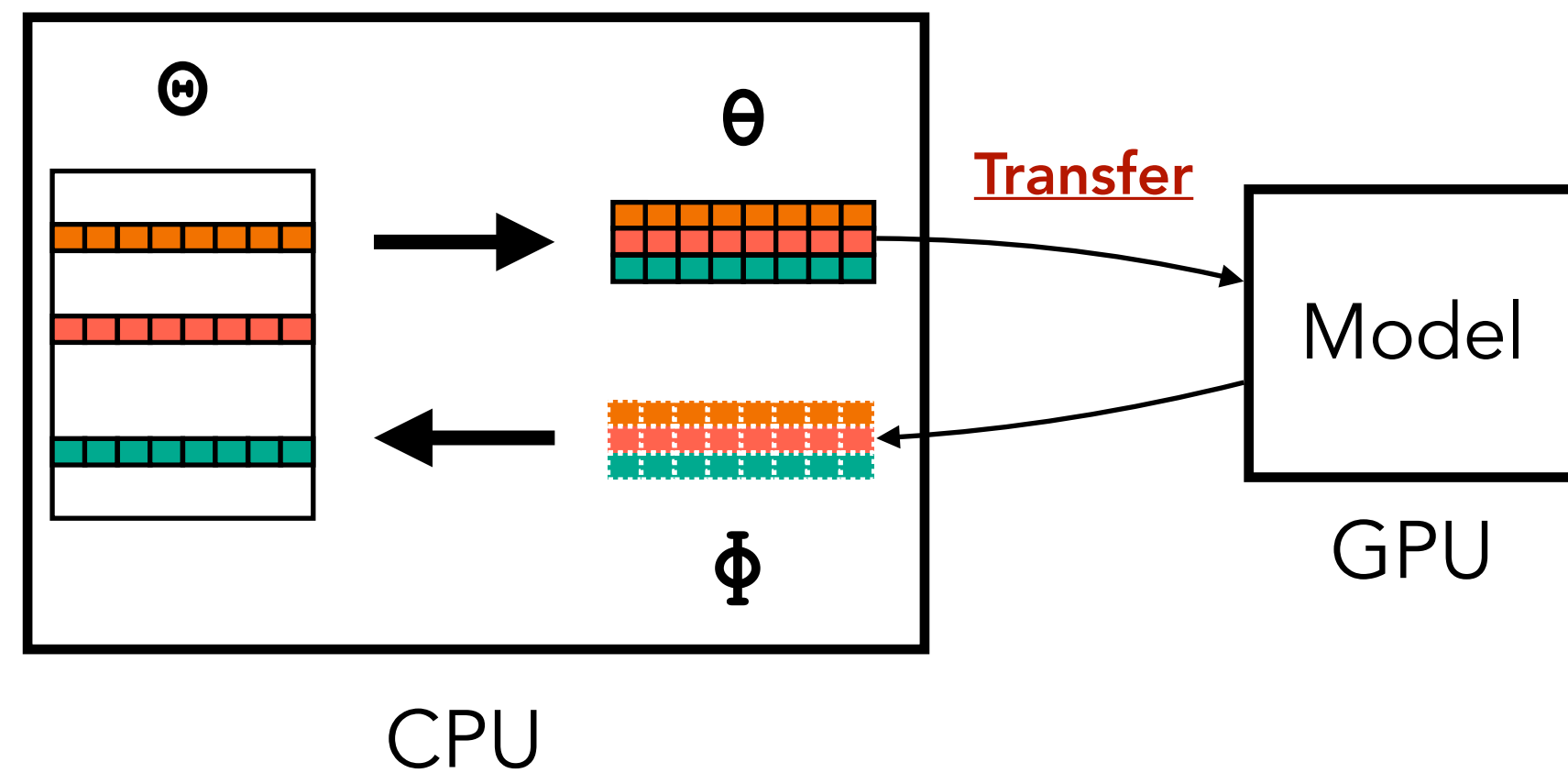
PBG (Facebook)

Why such poor utilization?

# Prior Solutions Bottlenecked by Data Movement

## DGL-KE (Amazon)

Synchronous training with batch prep on CPU



---

## PBG (Facebook)

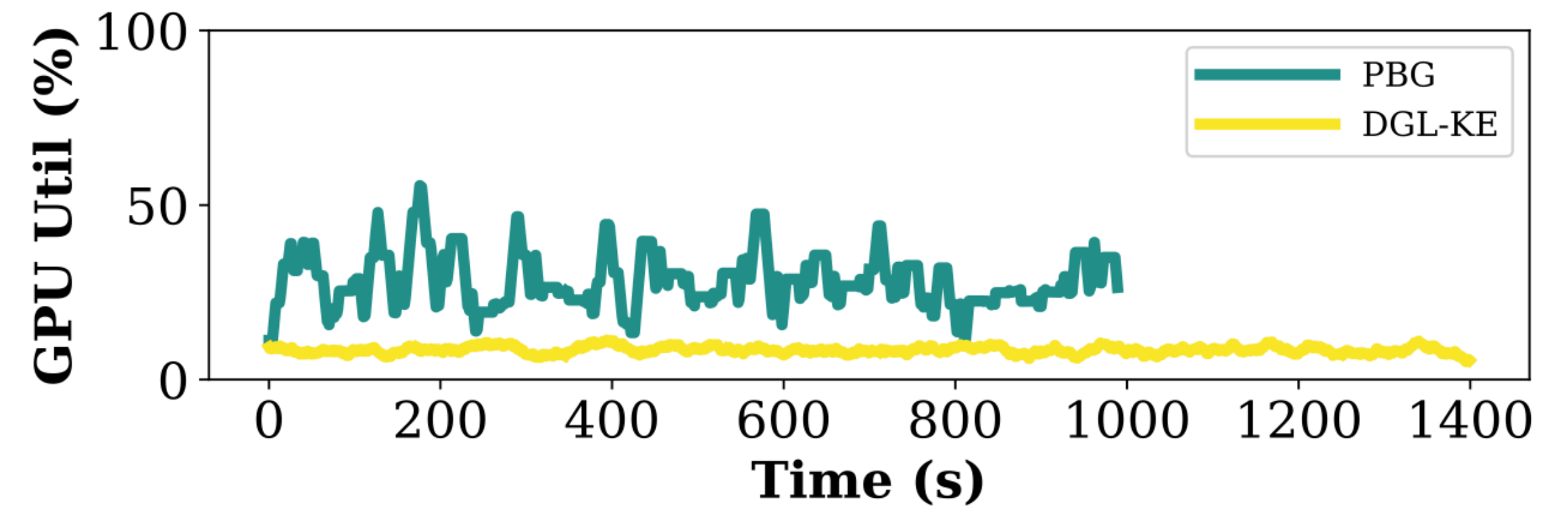


Figure 1: The GPU utilization of DGL-KE and PBG for one training epoch of ComplEx embeddings on the Free-base86m knowledge graph.

Why such poor utilization?

# Prior Solutions Bottlenecked by Data Movement

## DGL-KE (Amazon)

Synchronous training with batch prep on CPU

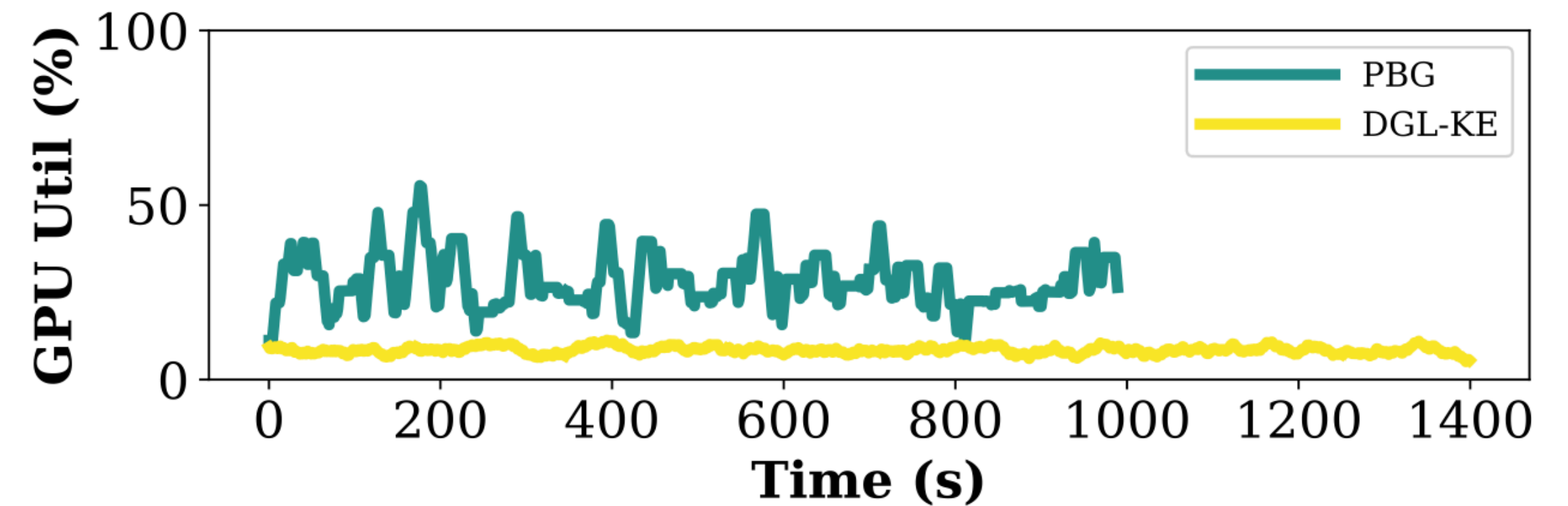
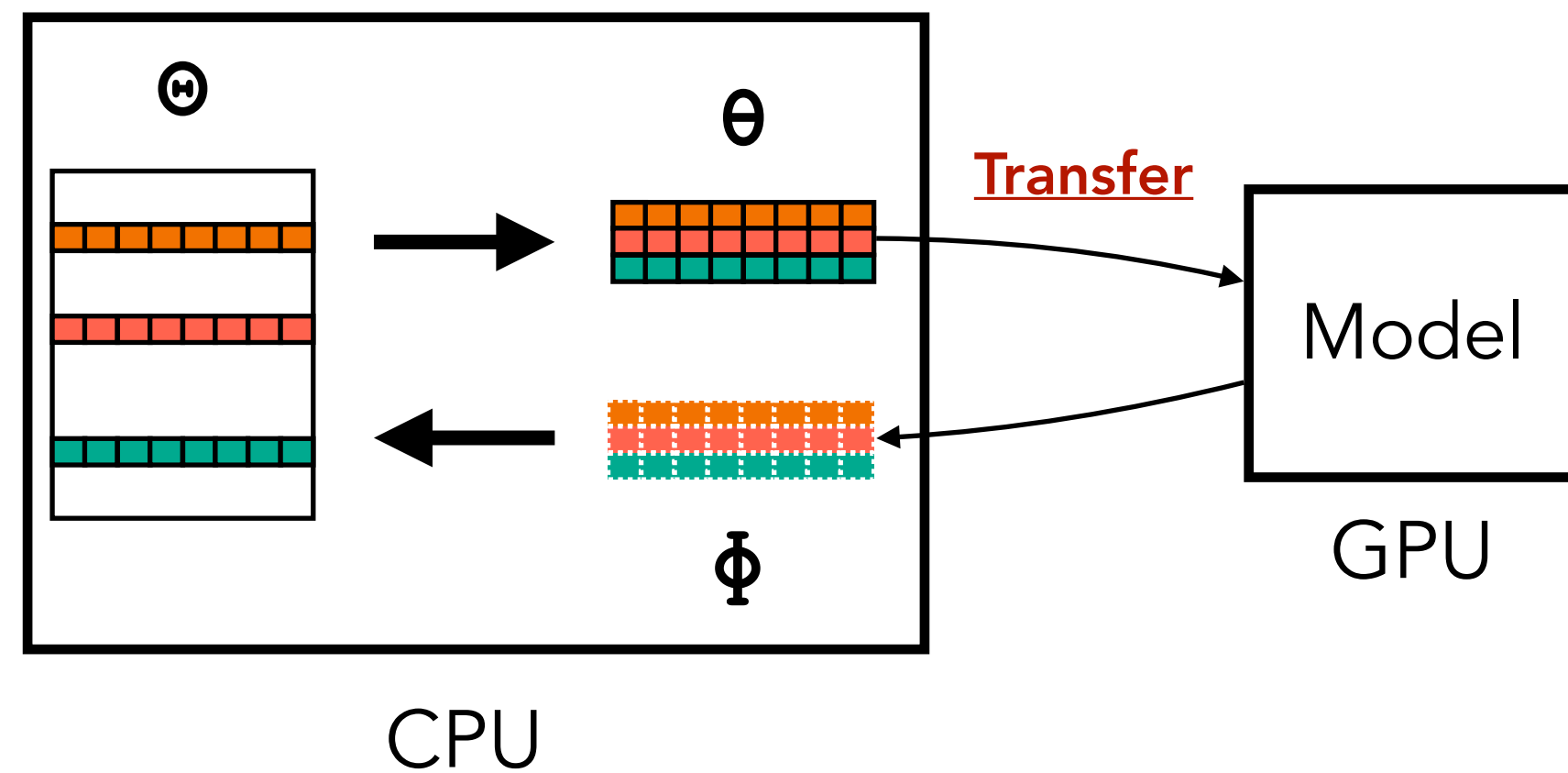
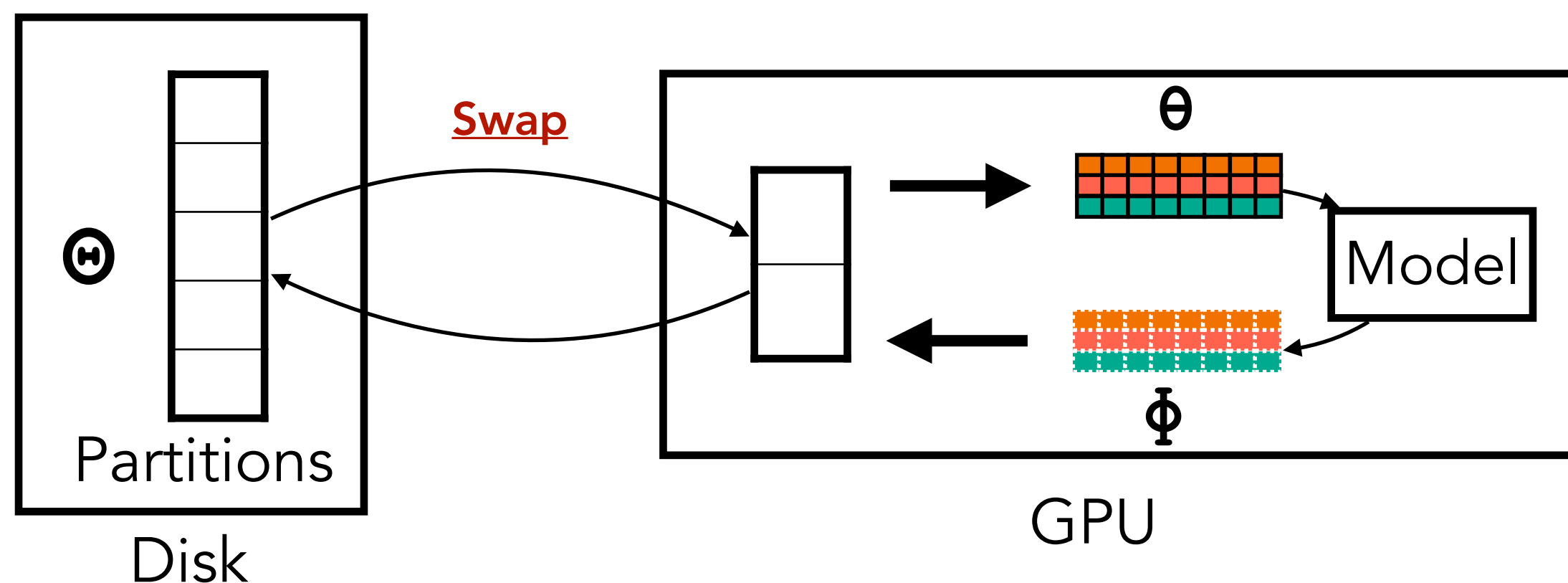


Figure 1: The GPU utilization of DGL-KE and PBG for one training epoch of ComplEx embeddings on the Freebase86m knowledge graph.

## PBG (Facebook)

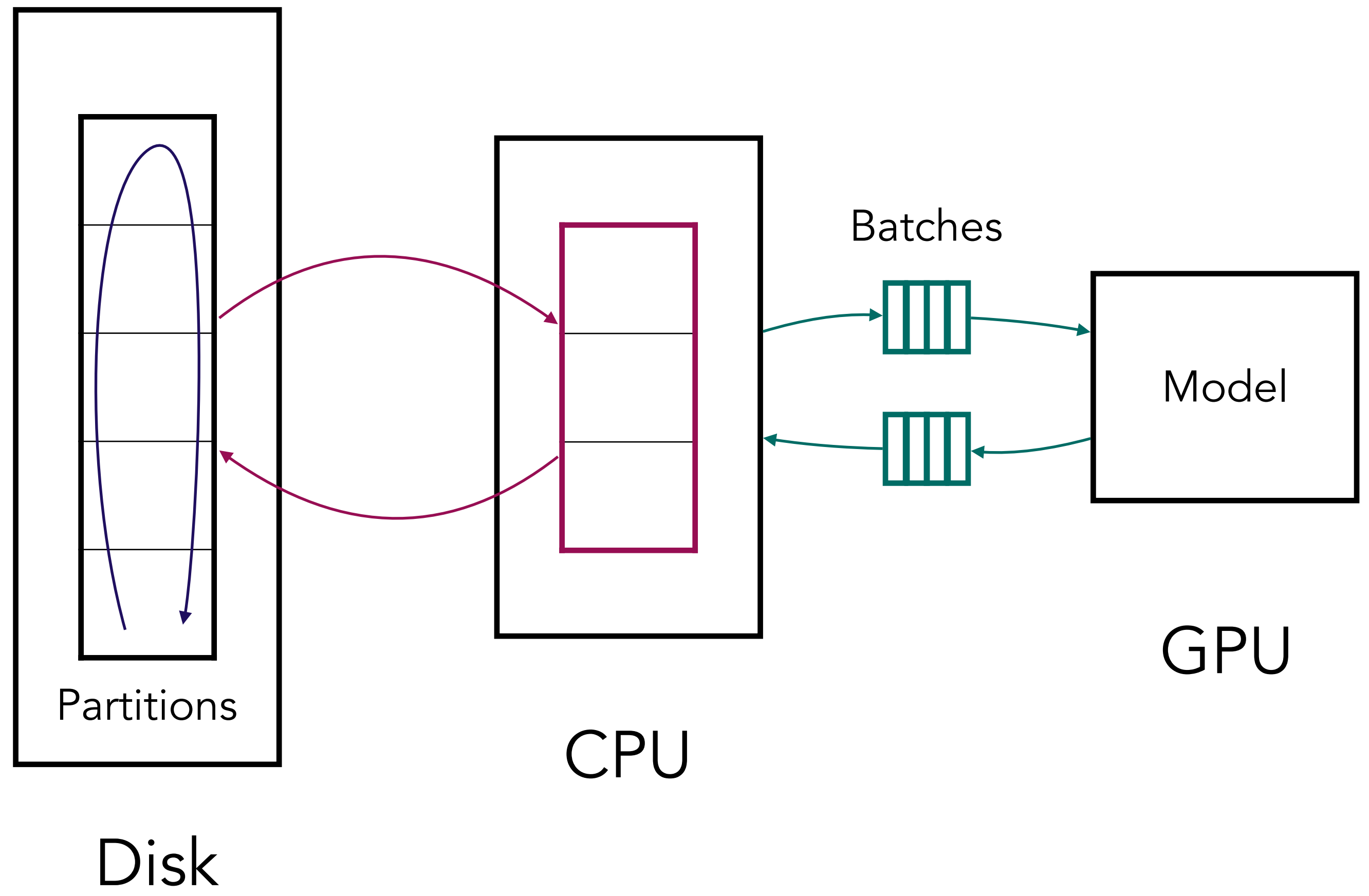
Partitioned synchronous training with batch prep on GPU



Why such poor utilization?

# Marius

- 1. Mitigate CPU-GPU transfer stalls through **pipelined training**
- 2. Mitigate Disk-CPU transfer stalls through **partition buffer**
- 3. Minimize Disk IO through **BETA ordering**



# Marius

1. Mitigate CPU-GPU transfer stalls through **pipelined training**
2. Mitigate Disk-CPU transfer stalls through **partition buffer**
3. Minimize Disk IO through **BETA ordering**

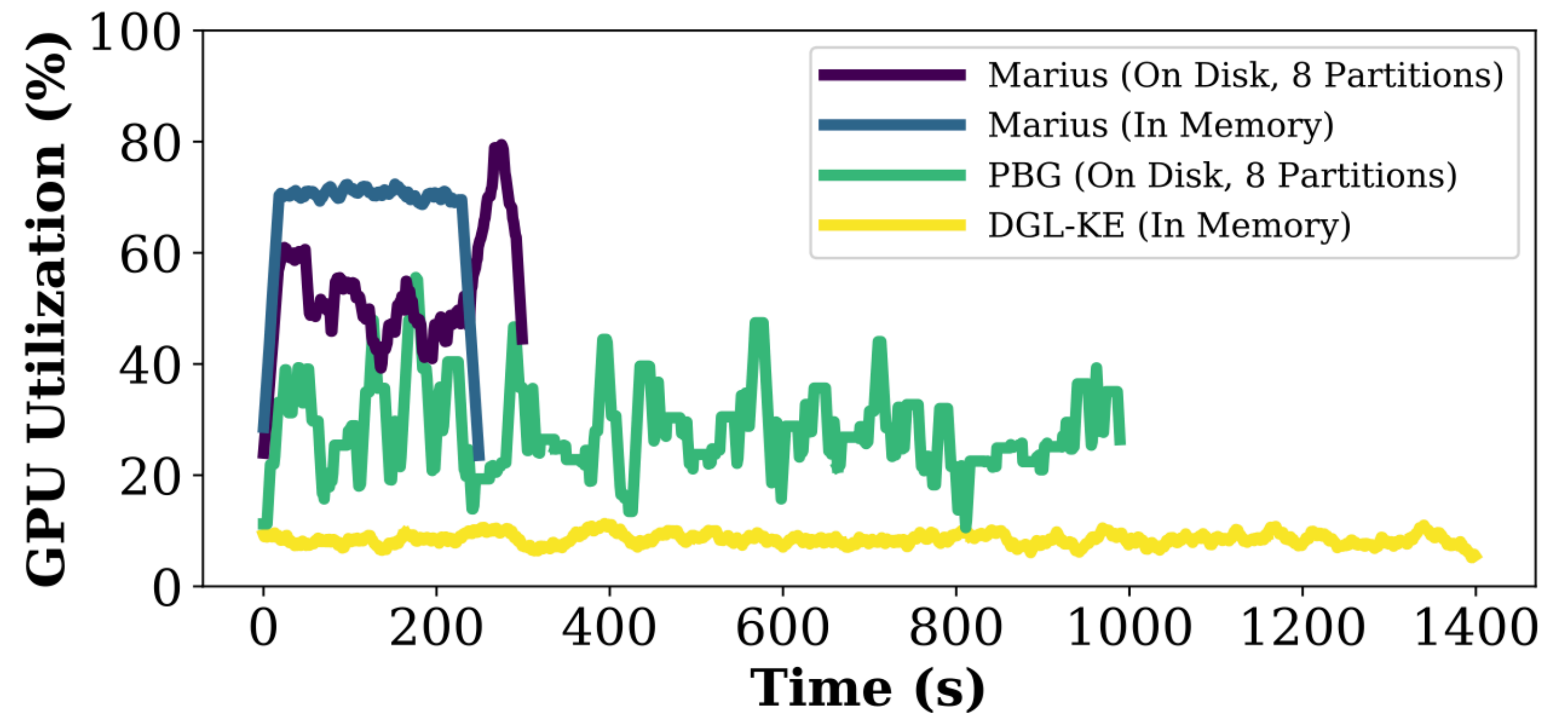
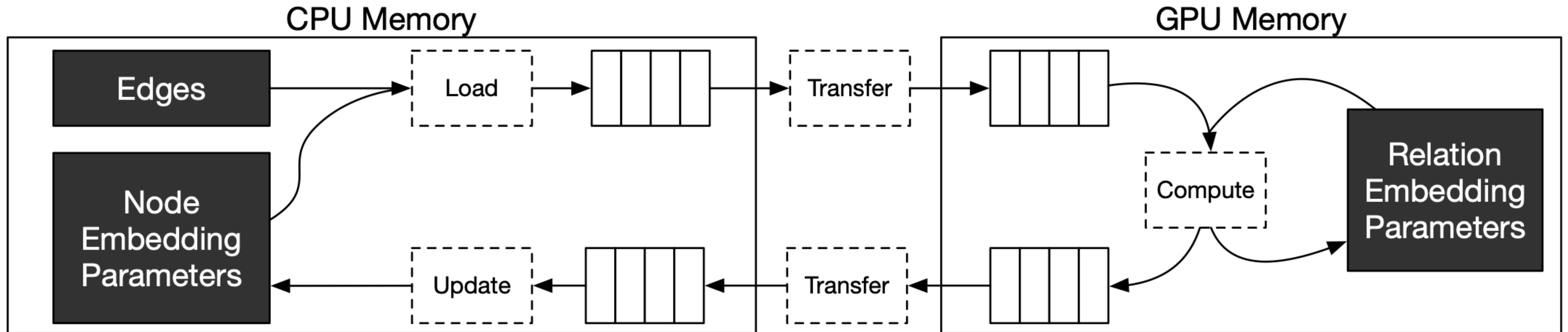
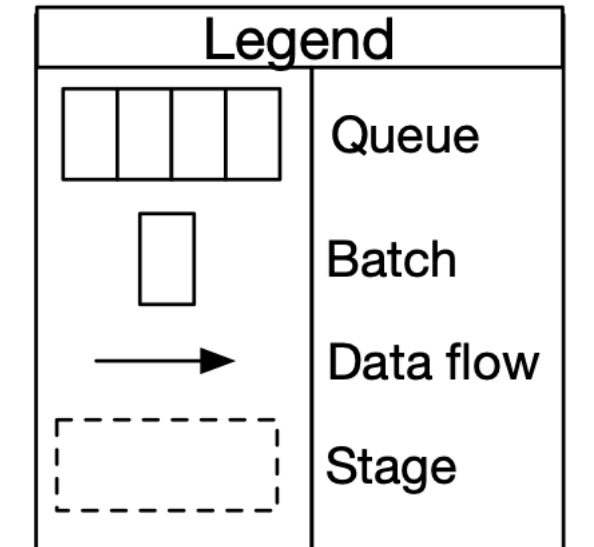


Figure 8: GPU utilization of Marius, DGL-KE and PBG during a single epoch of training  $d = 50$  embeddings on Freebase86m. Utilization is smoothed over 25 seconds.

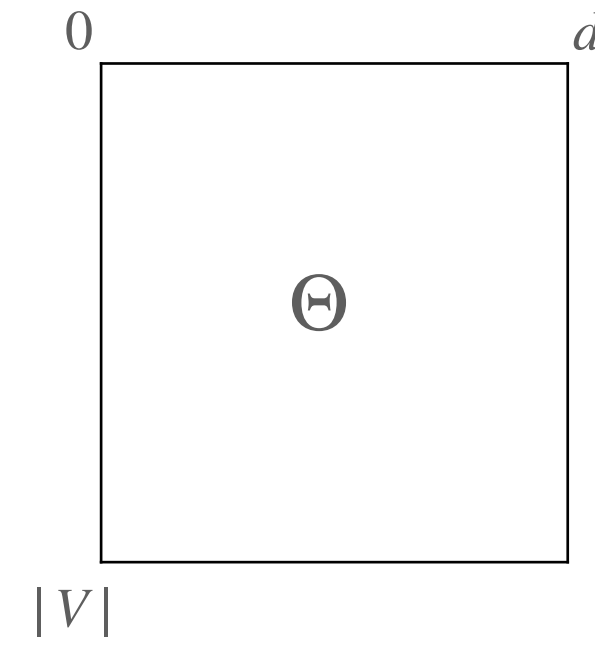
# Pipelined Architecture



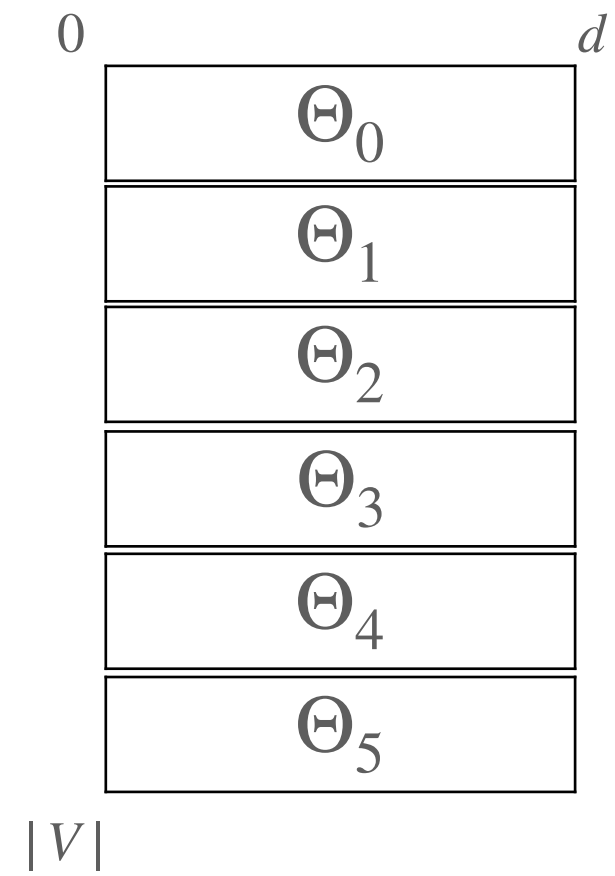
# Out-of-core Training

## Node Embedding Partitions

Node embeddings are partitioned uniformly into  $p$  disjoint partitions.



Node embedding table



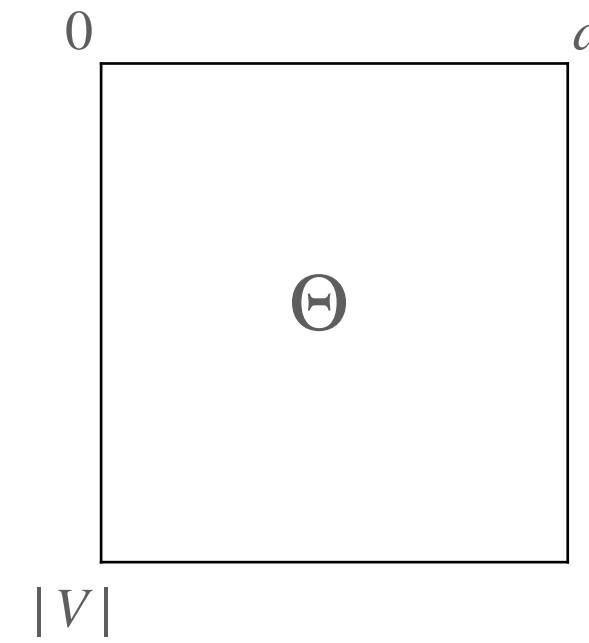
Partitioned node embedding table ( $p = 6$ )

---

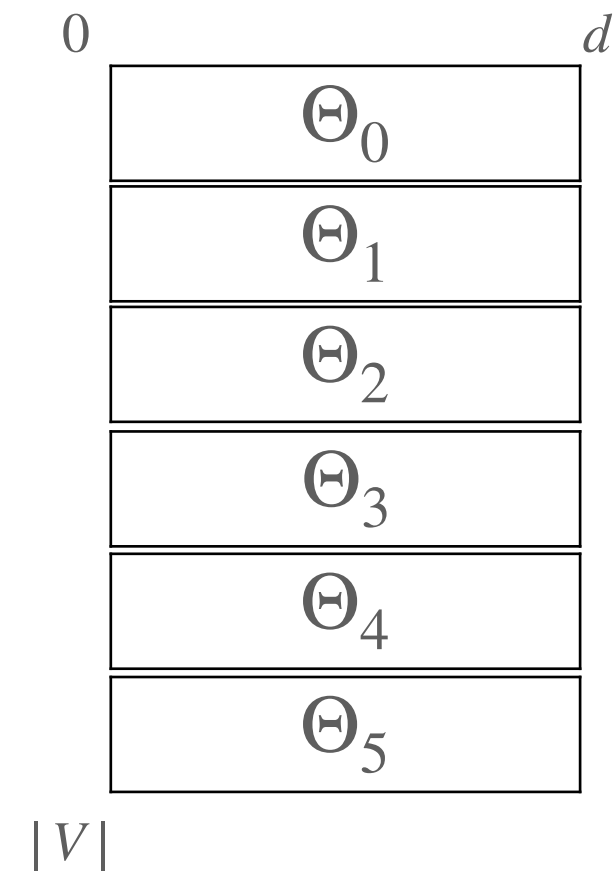
# Out-of-core Training

## Node Embedding Partitions

Node embeddings are partitioned uniformly into  $p$  disjoint partitions.



Node embedding table

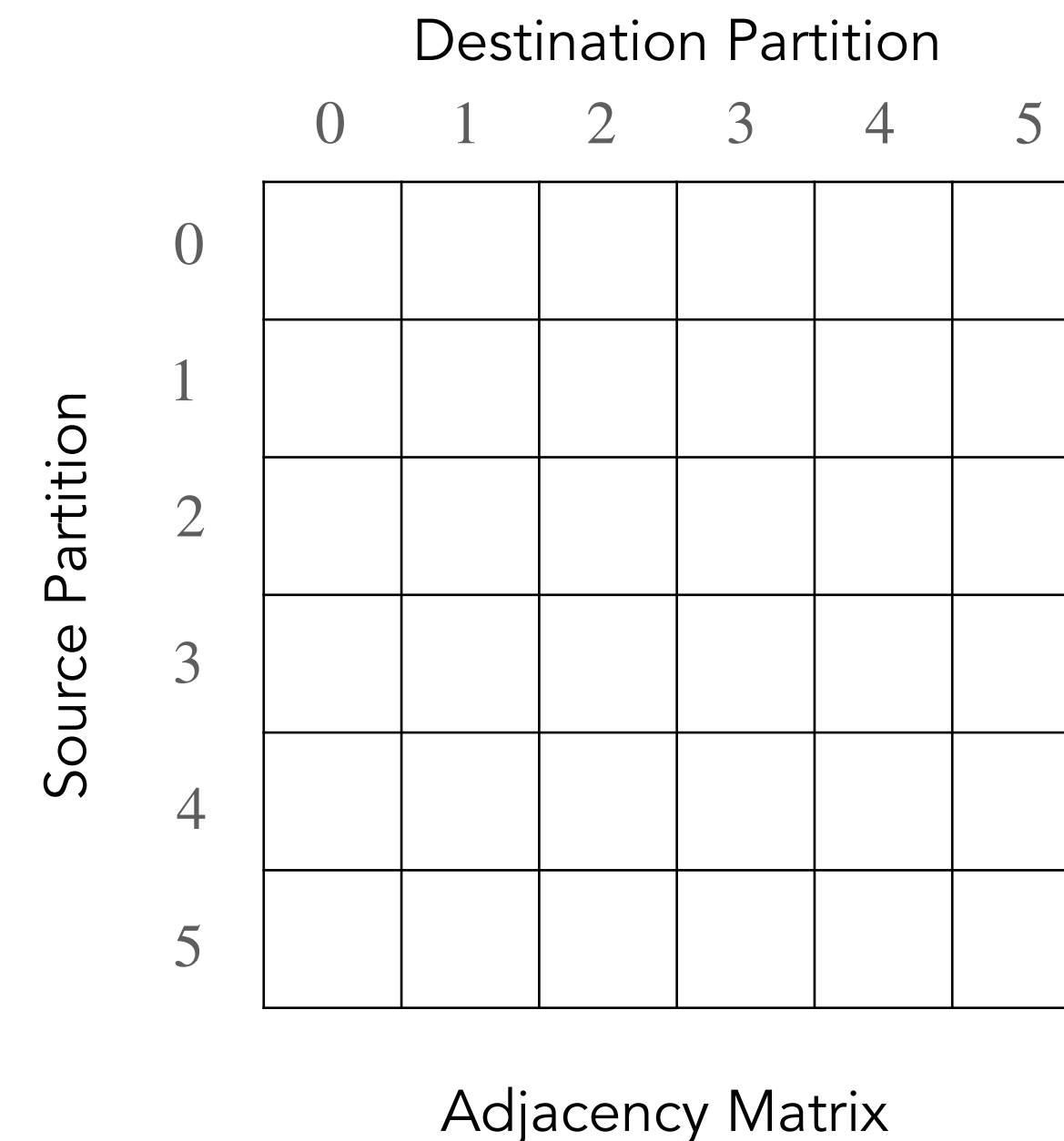


Partitioned node embedding table ( $p = 6$ )

## Edge Buckets

Edge bucket  $(i,j)$  contains all edges with a source in partition  $i$  and a destination in partition  $j$

**To train, we need to iterate over all edges, we need to iterate over all edge buckets**



Adjacency Matrix

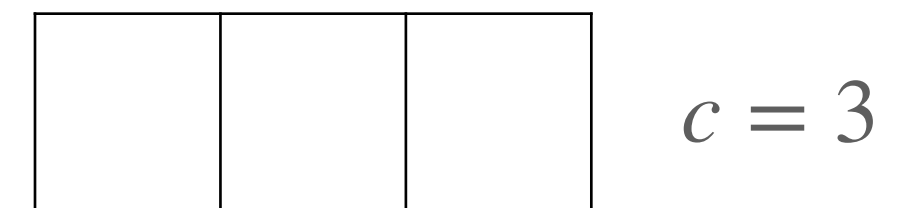
# Buffer-aware Edge Traversal Algorithm (BETA)

## BETA Ordering

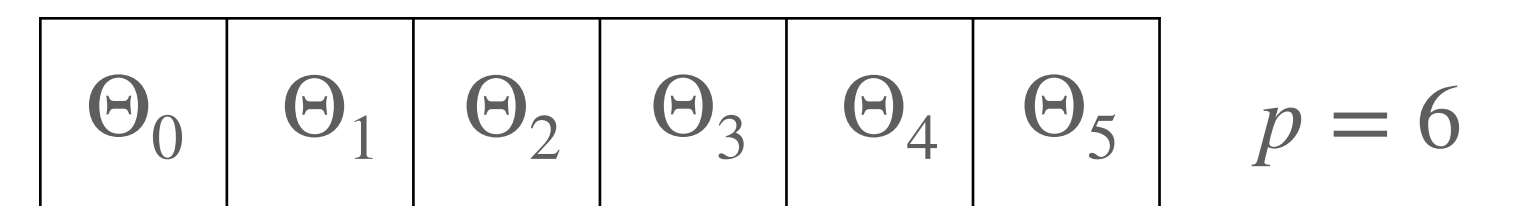
1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new  $c - 1$  partitions and repeat until all edge buckets have been processed

		Destination Partition					
		0	1	2	3	4	5
Source Partition	0						
	1						
	2						
	3						
	4						
	5						

Partitions in Buffer



Partitions on disk



# Buffer-aware Edge Traversal Algorithm (BETA)

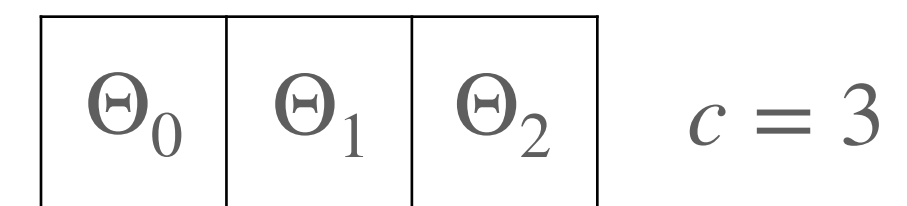
## BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new  $c - 1$  partitions and repeat until all edge buckets have been processed

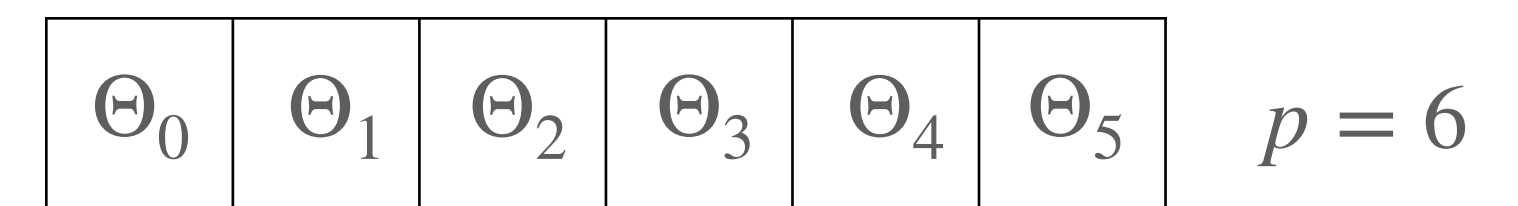
0 swaps\*

		Destination Partition					
		0	1	2	3	4	5
Source Partition	0						
	1						
	2						
	3						
	4						
	5						

Partitions in Buffer



Partitions on disk



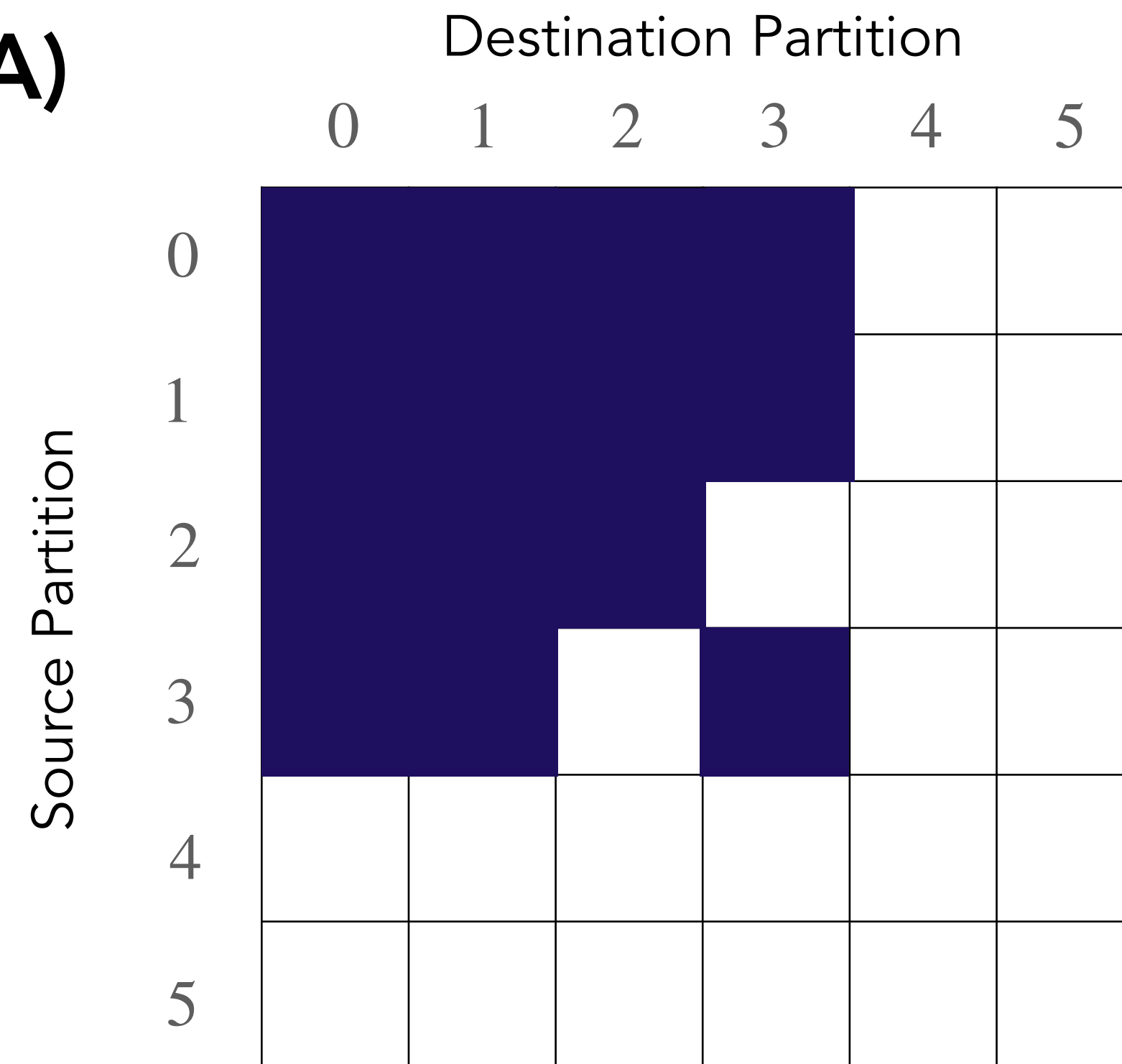
\* Not counting initialized buffer

# Buffer-aware Edge Traversal Algorithm (BETA)

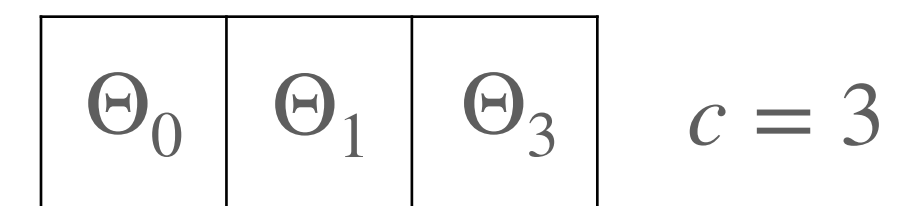
## BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new  $c - 1$  partitions and repeat until all edge buckets have been processed

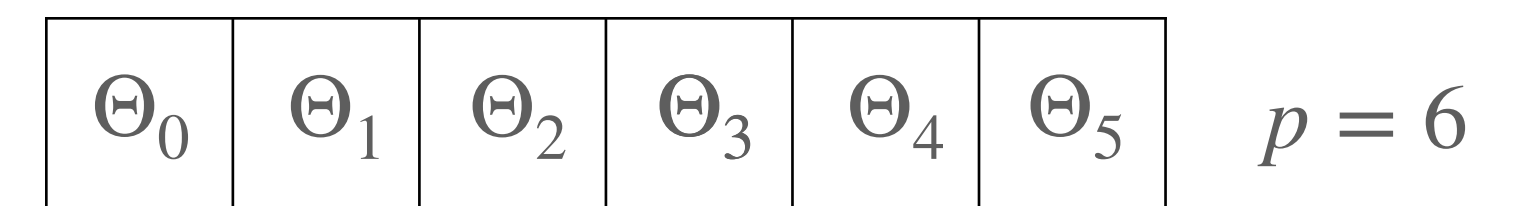
1 swap



Partitions in Buffer



Partitions on disk

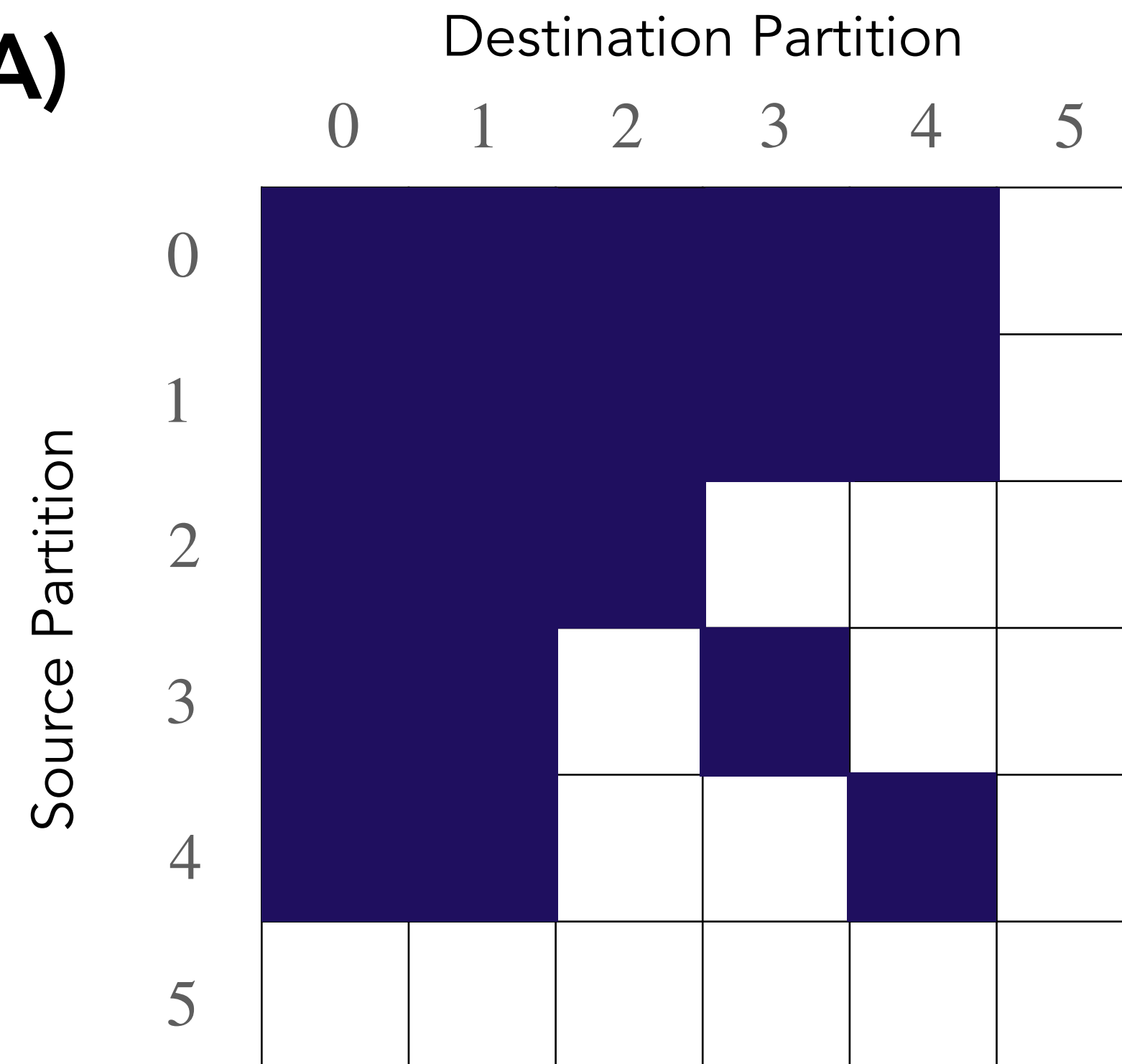


# Buffer-aware Edge Traversal Algorithm (BETA)

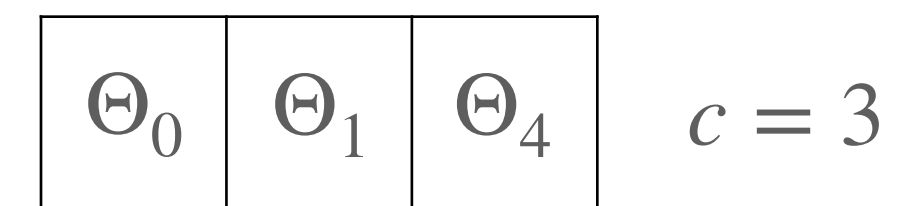
## BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new  $c - 1$  partitions and repeat until all edge buckets have been processed

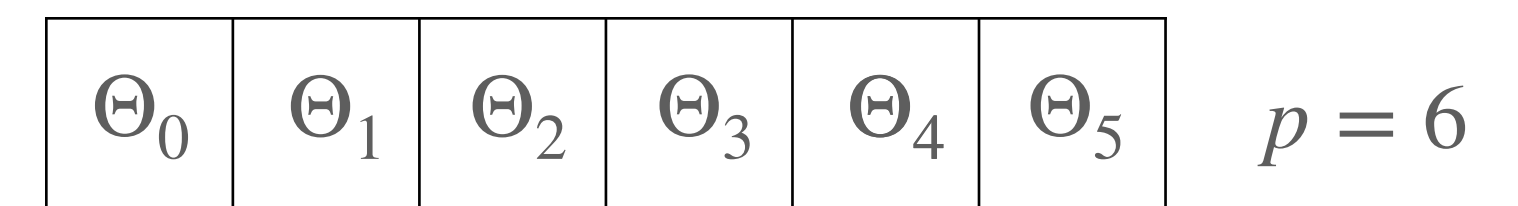
2 swaps



Partitions in Buffer



Partitions on disk

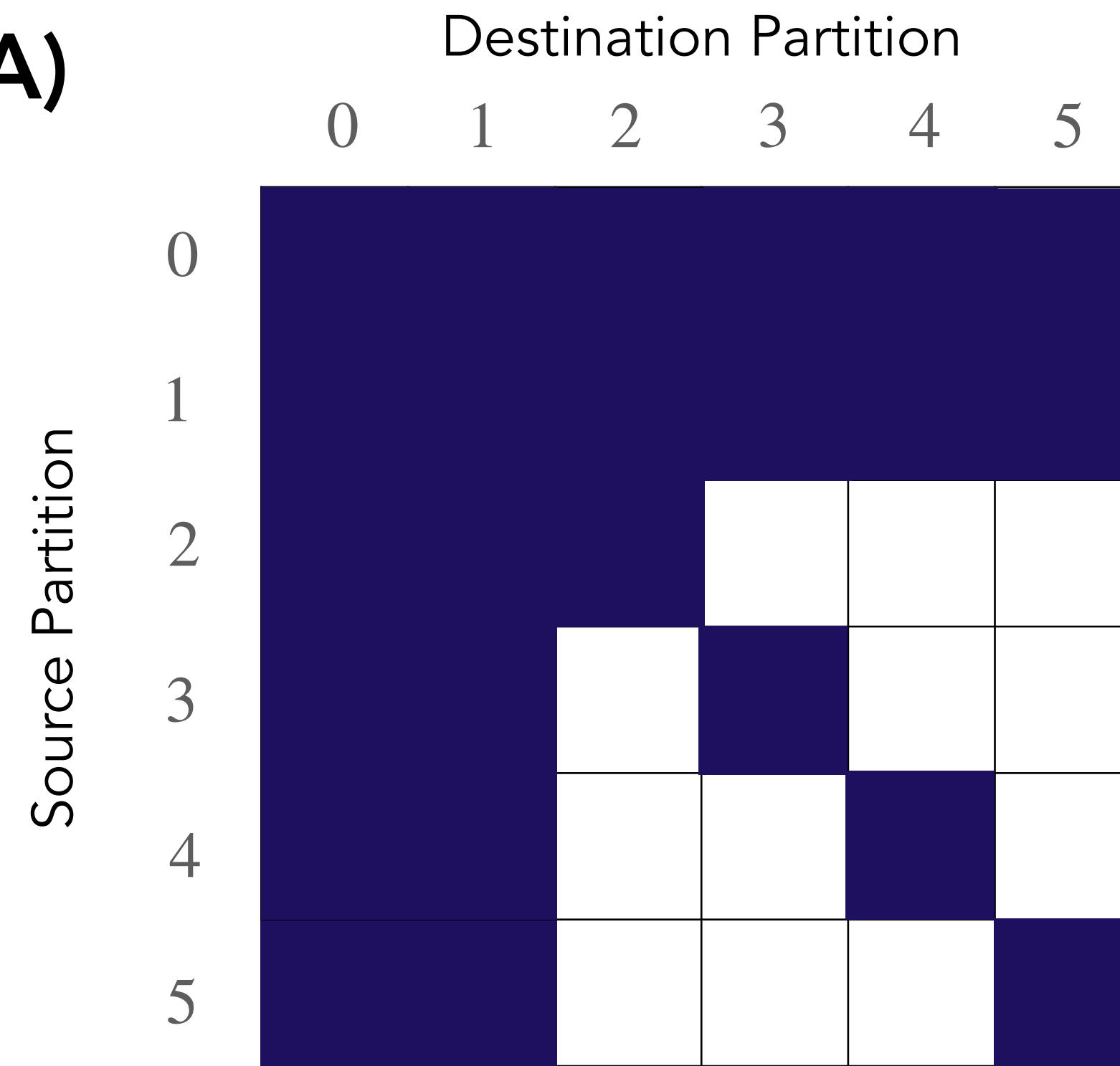


# Buffer-aware Edge Traversal Algorithm (BETA)

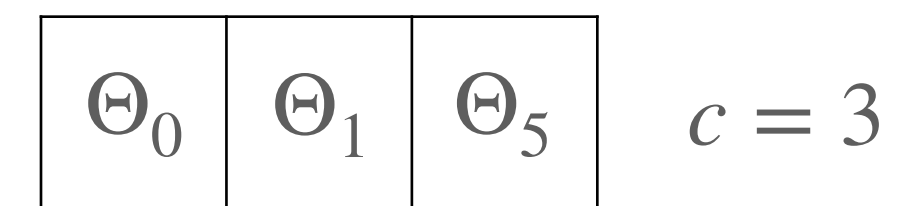
## BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new  $c - 1$  partitions and repeat until all edge buckets have been processed

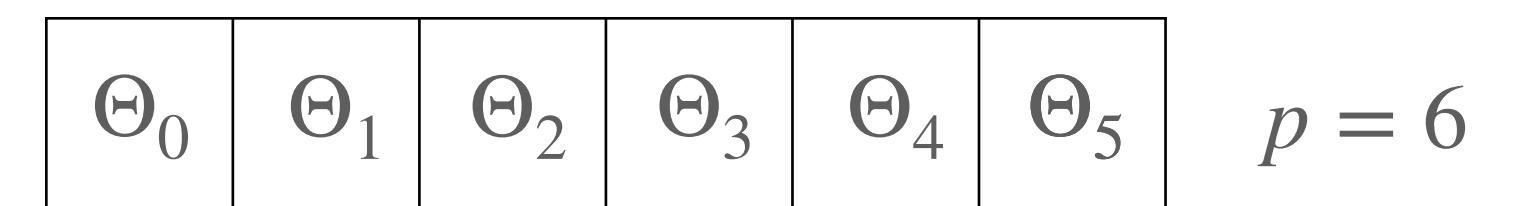
3 swaps



Partitions in Buffer



Partitions on disk

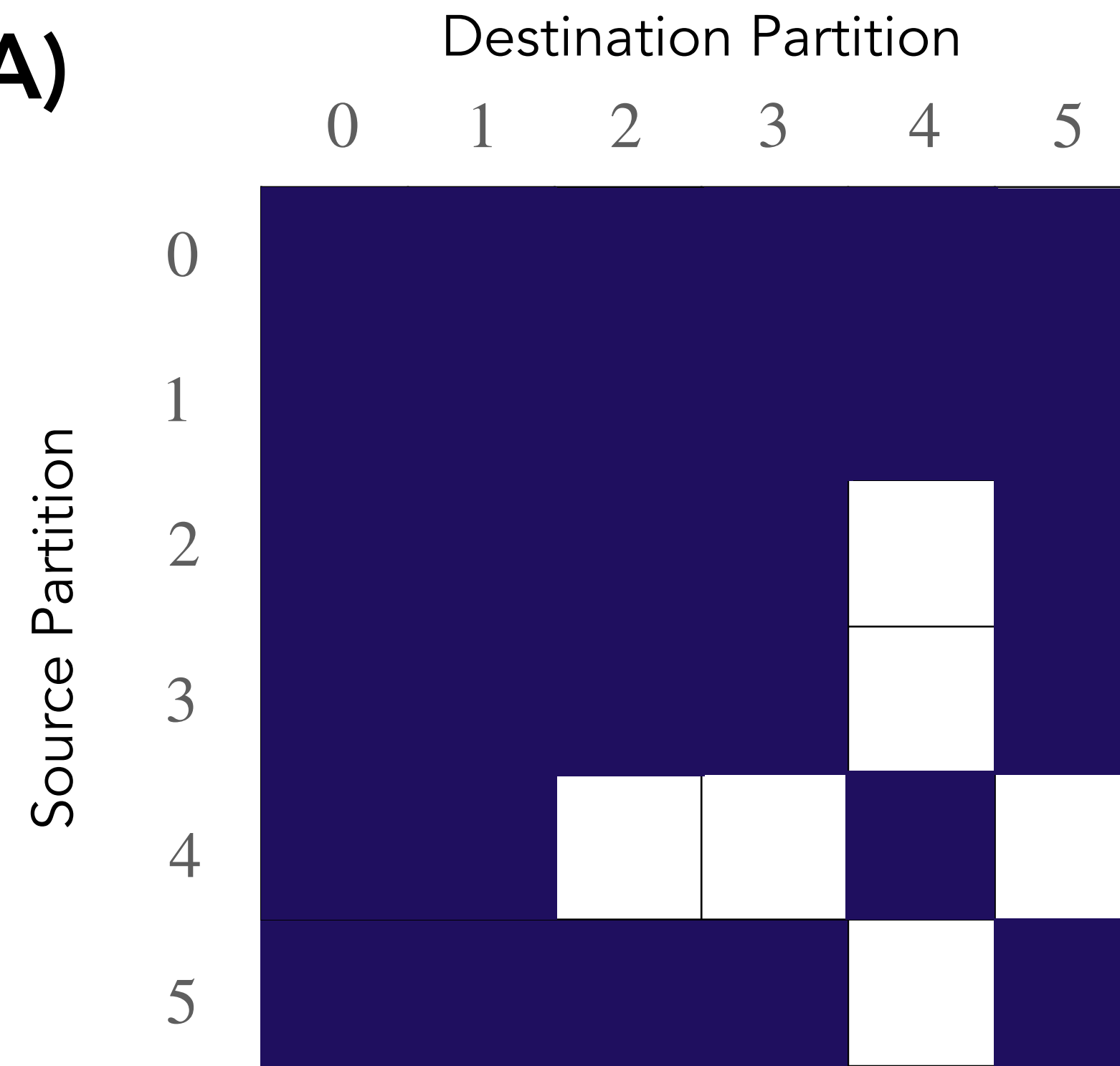


# Buffer-aware Edge Traversal Algorithm (BETA)

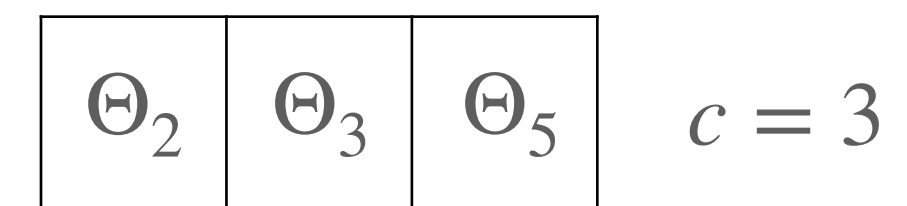
## BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new  $c - 1$  partitions and repeat until all edge buckets have been processed

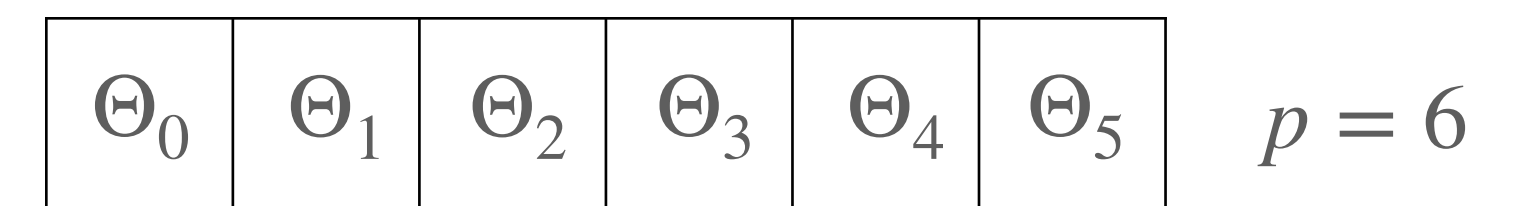
5 swaps



Partitions in Buffer



Partitions on disk

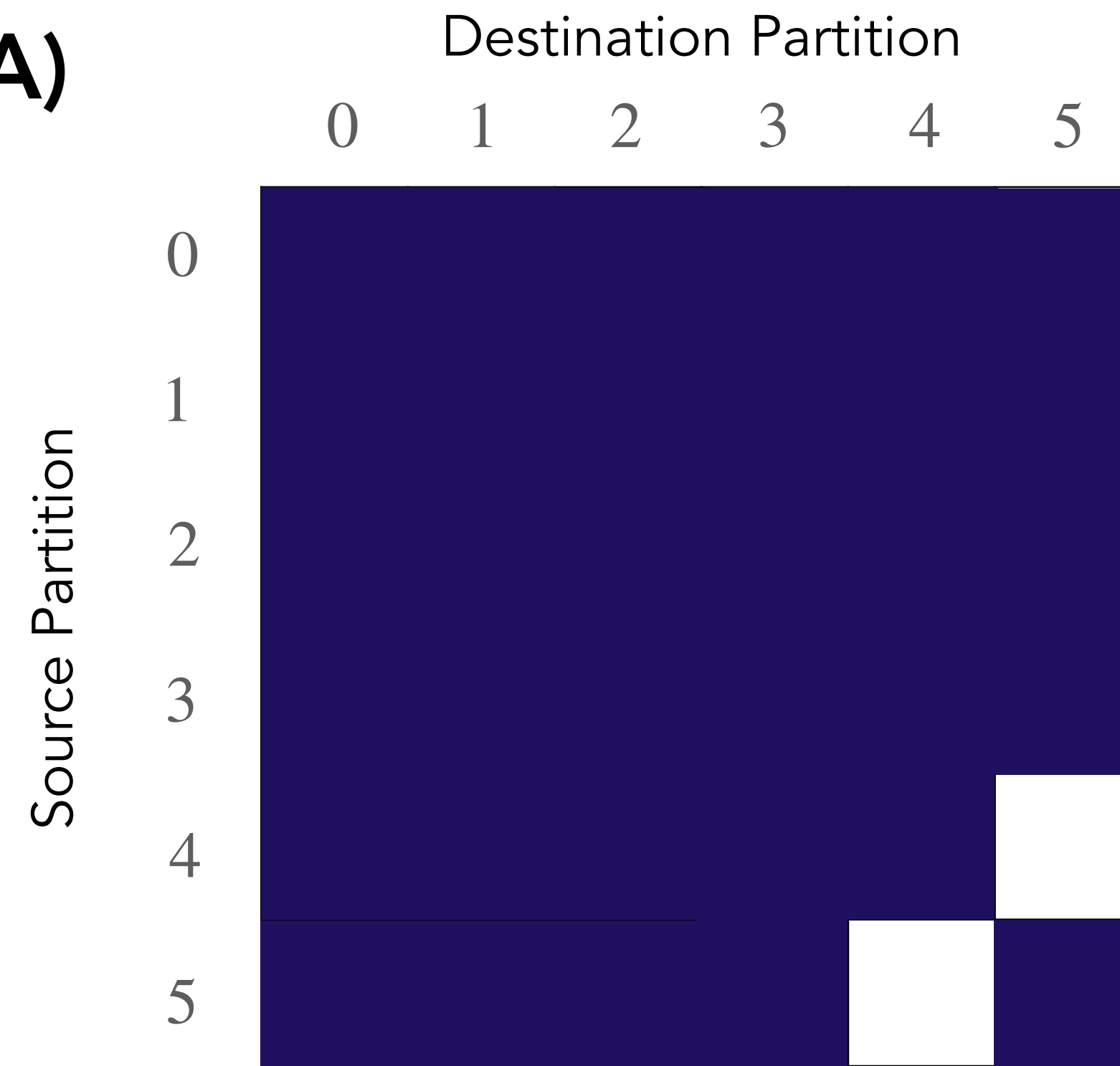


# Buffer-aware Edge Traversal Algorithm (BETA)

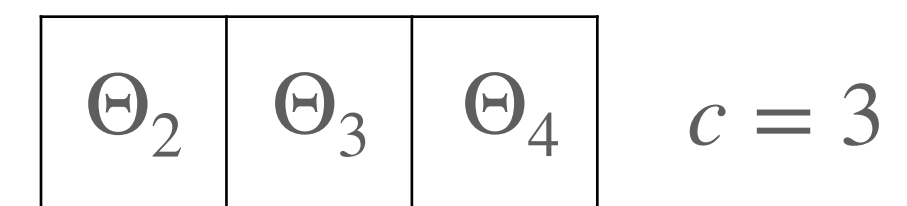
## BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new  $c - 1$  partitions and repeat until all edge buckets have been processed

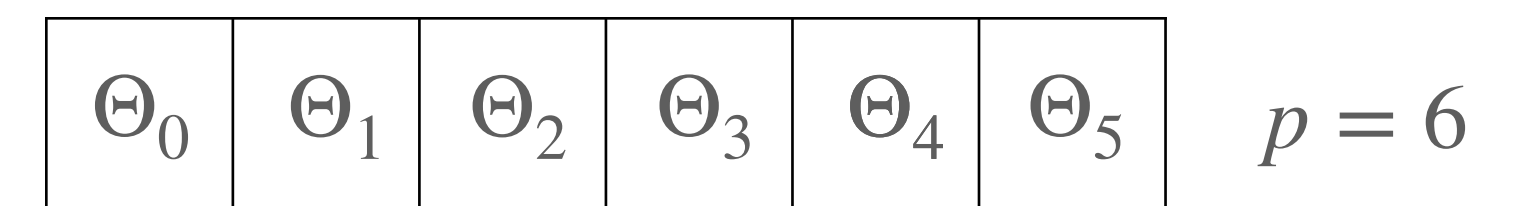
6 swaps



Partitions in Buffer



Partitions on disk

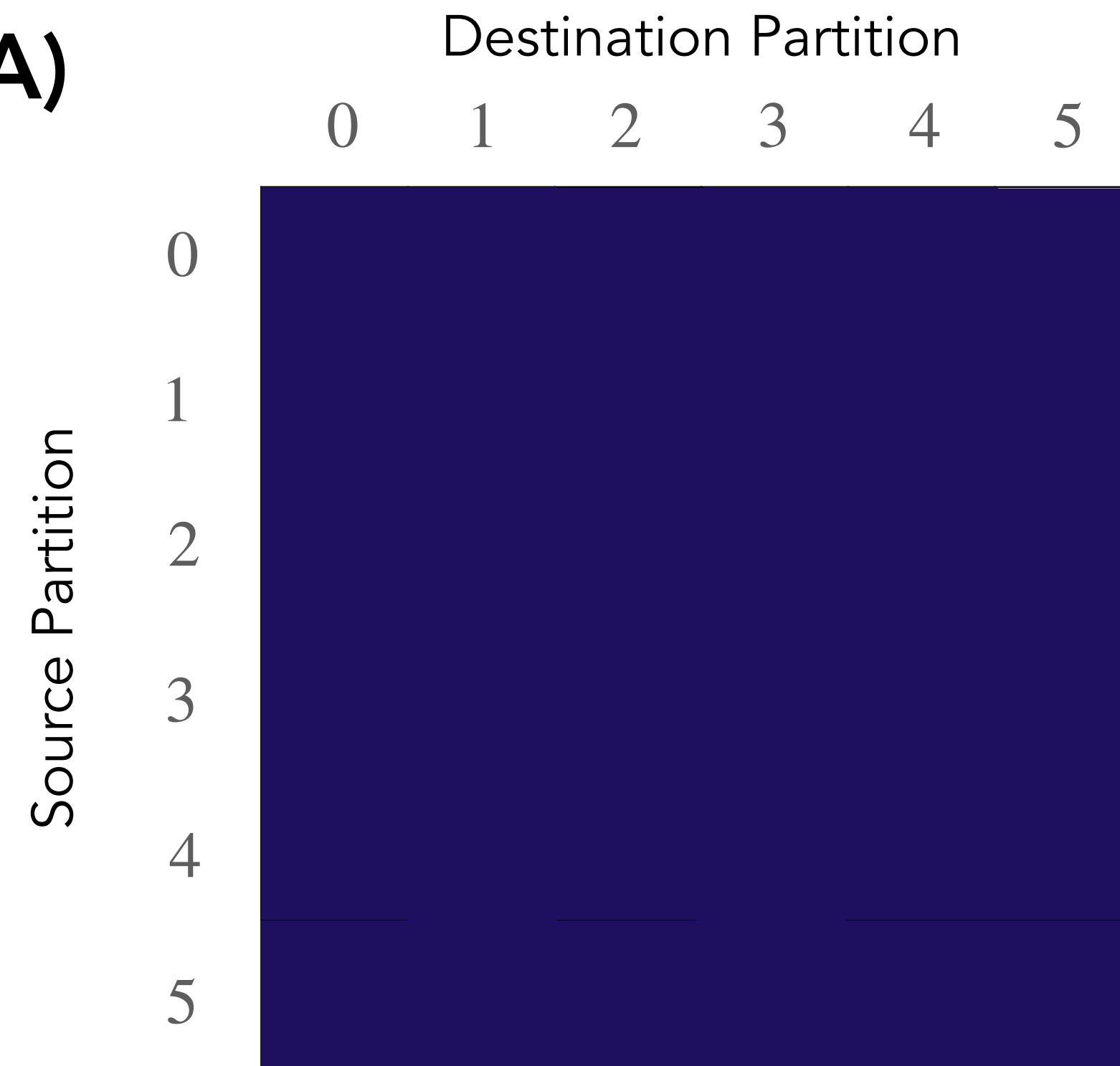


# Buffer-aware Edge Traversal Algorithm (BETA)

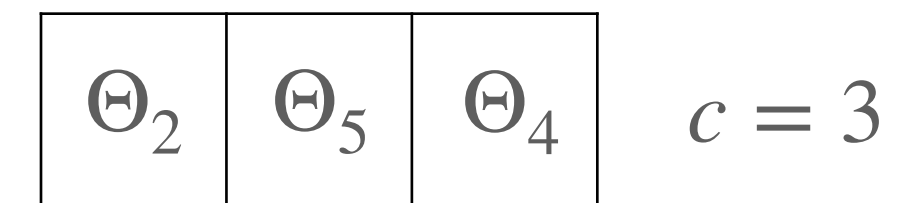
## BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new  $c - 1$  partitions and repeat until all edge buckets have been processed

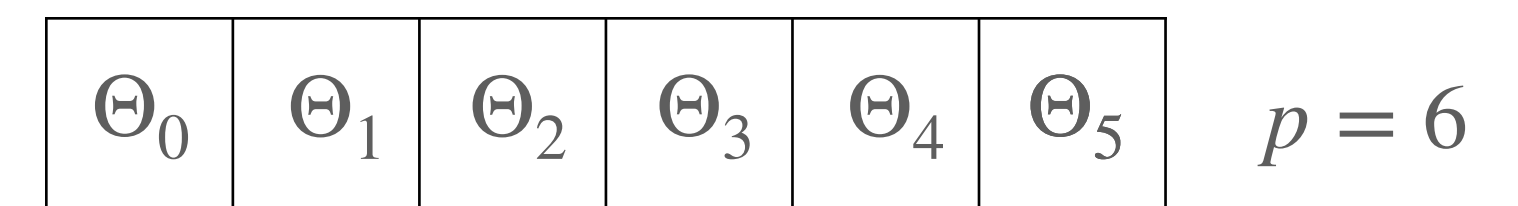
7 swaps



Partitions in Buffer



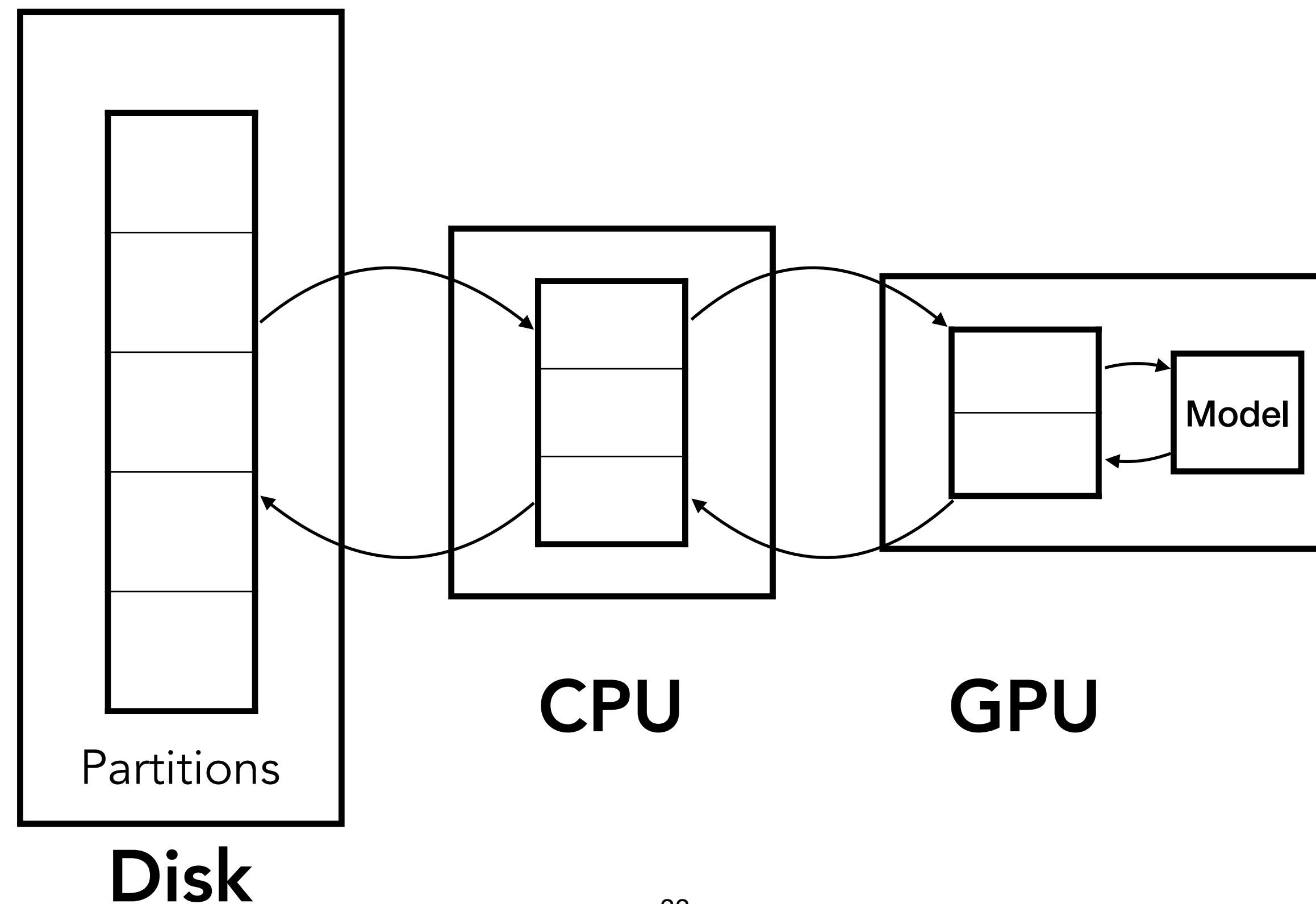
Partitions on disk



# Discussion

Consider a three-tiered design where partitions are cached in the **CPU and GPU**.

1. What are the advantages & disadvantages of this design?
2. What changes are needed to the training algorithm?



# Discussion

**1. What are the advantages & disadvantages of the three-tier design?**

**2. What changes are needed to the training algorithm?**

# Next Steps

- Next week: Vector Databases
- Project check-ins **due April 7th**