Marius

Machine Learning over Billion-Edge Graphs 10x Faster and 5x Cheaper

Theo Rekatsinas | thodrek@cs.wisc.edu

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

The team

Student Leads


Jason Mohoney

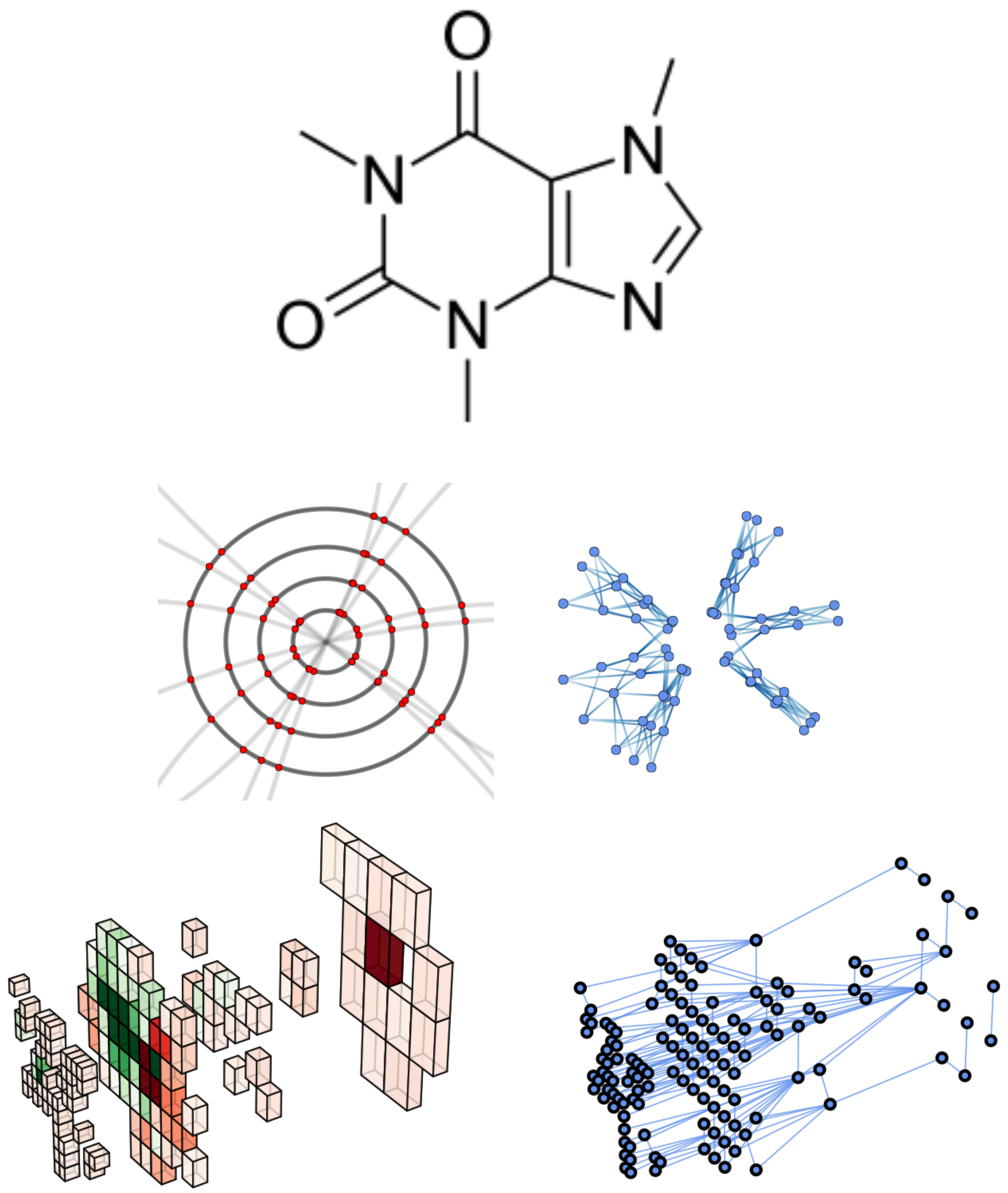

Roger Waleffe

Project Leads

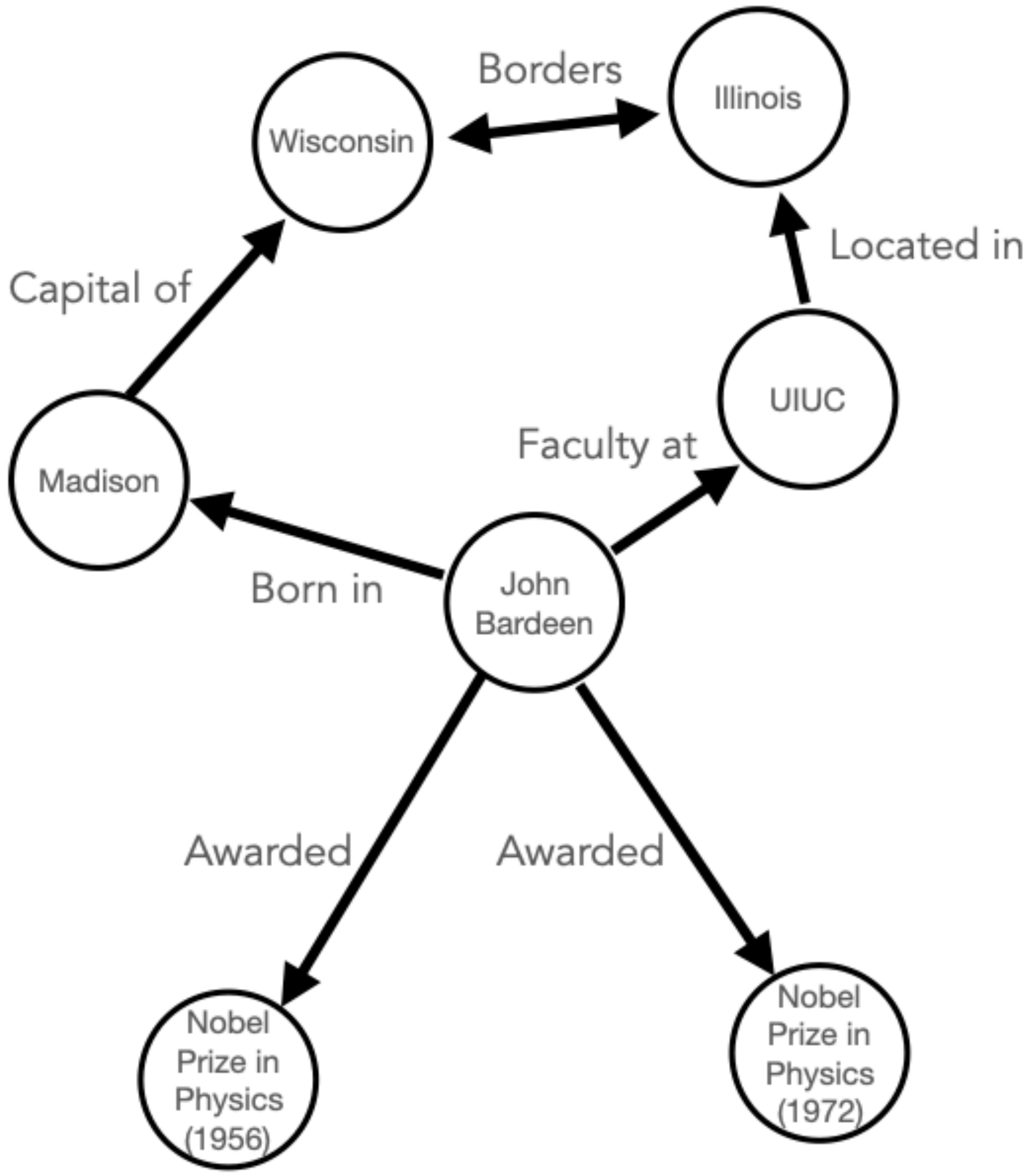

Prof. Theo Rekatsinas


Prof. Shivaram Venkataraman

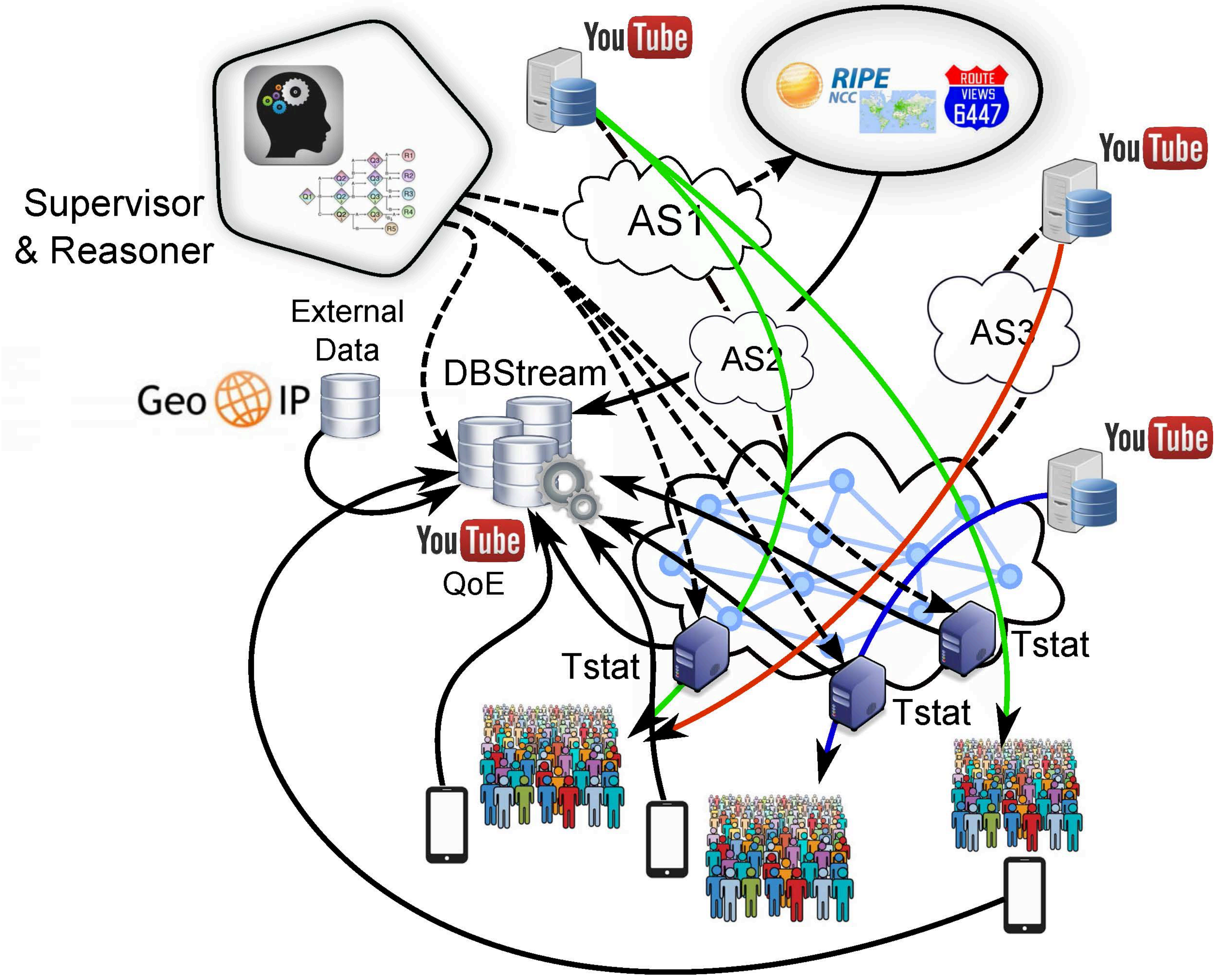
The universality of semantic structure



Scientific graphs



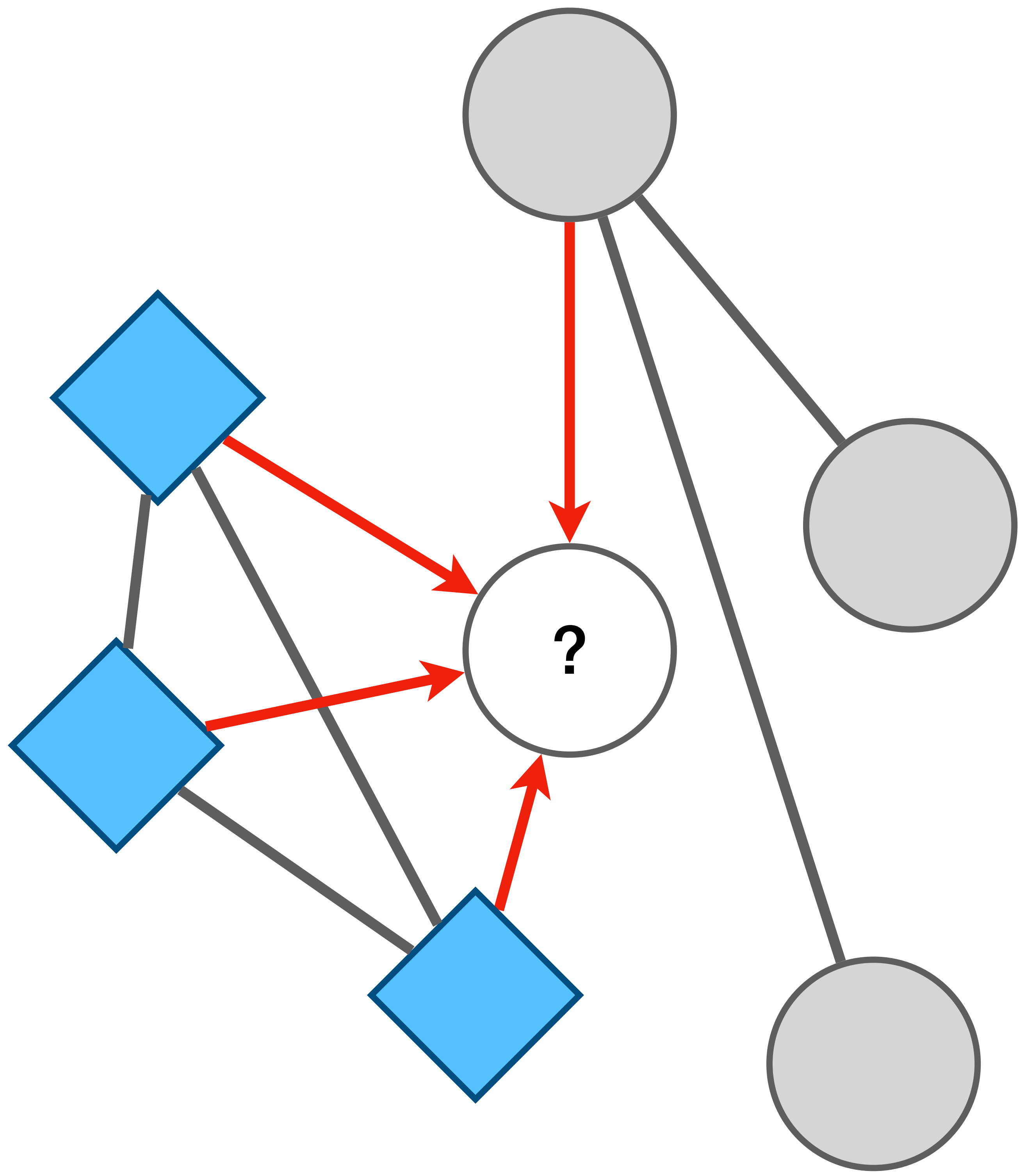
Knowledge Graphs



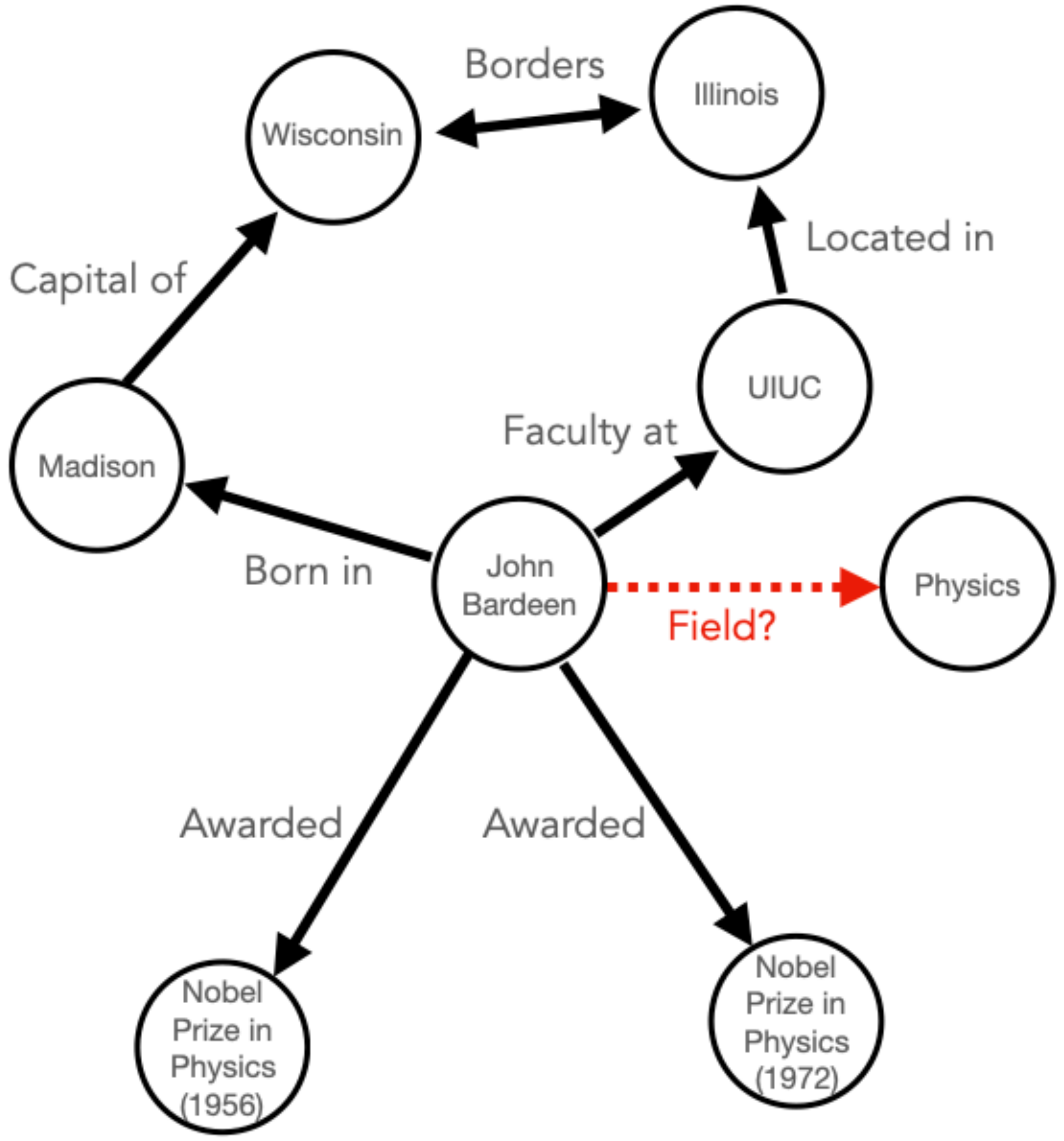
Server Logs

Graphs are **universal representations** of rich semantics about entities (nodes) and their relationships (edges)

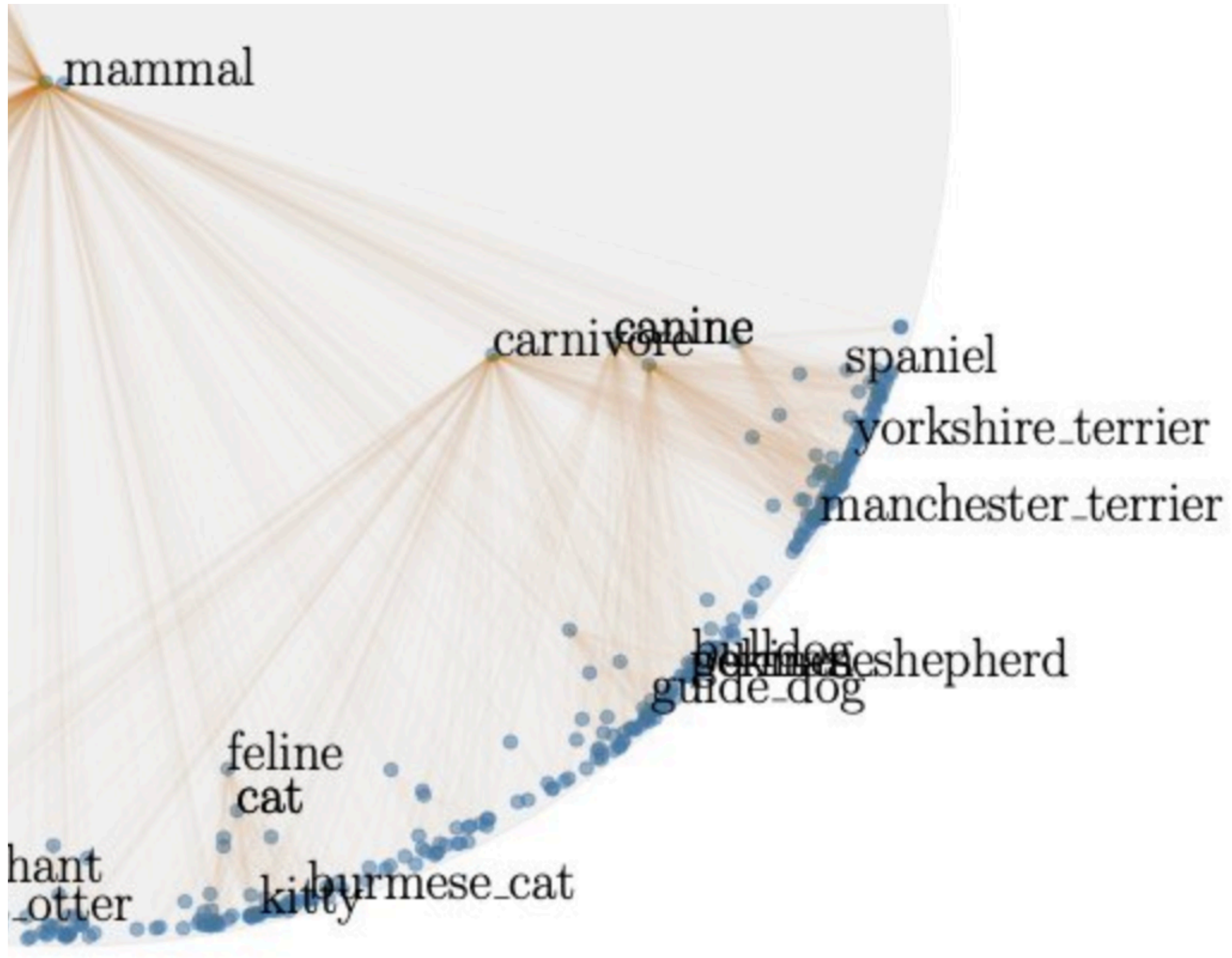
Harnessing the power of structure



Node classification



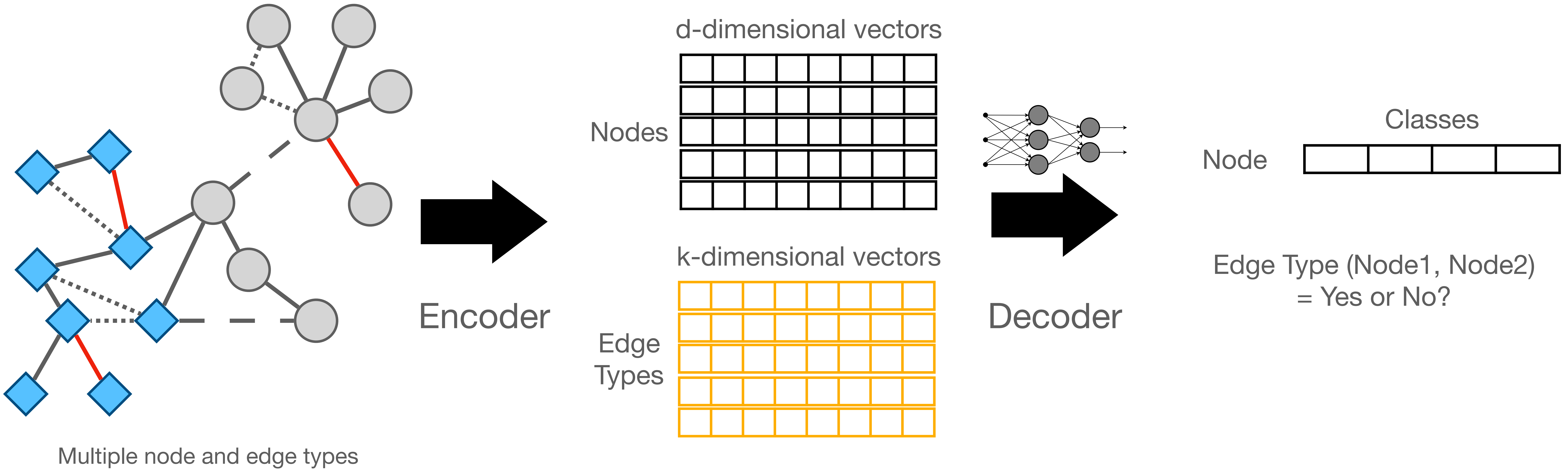
Link prediction



Related-entities prediction

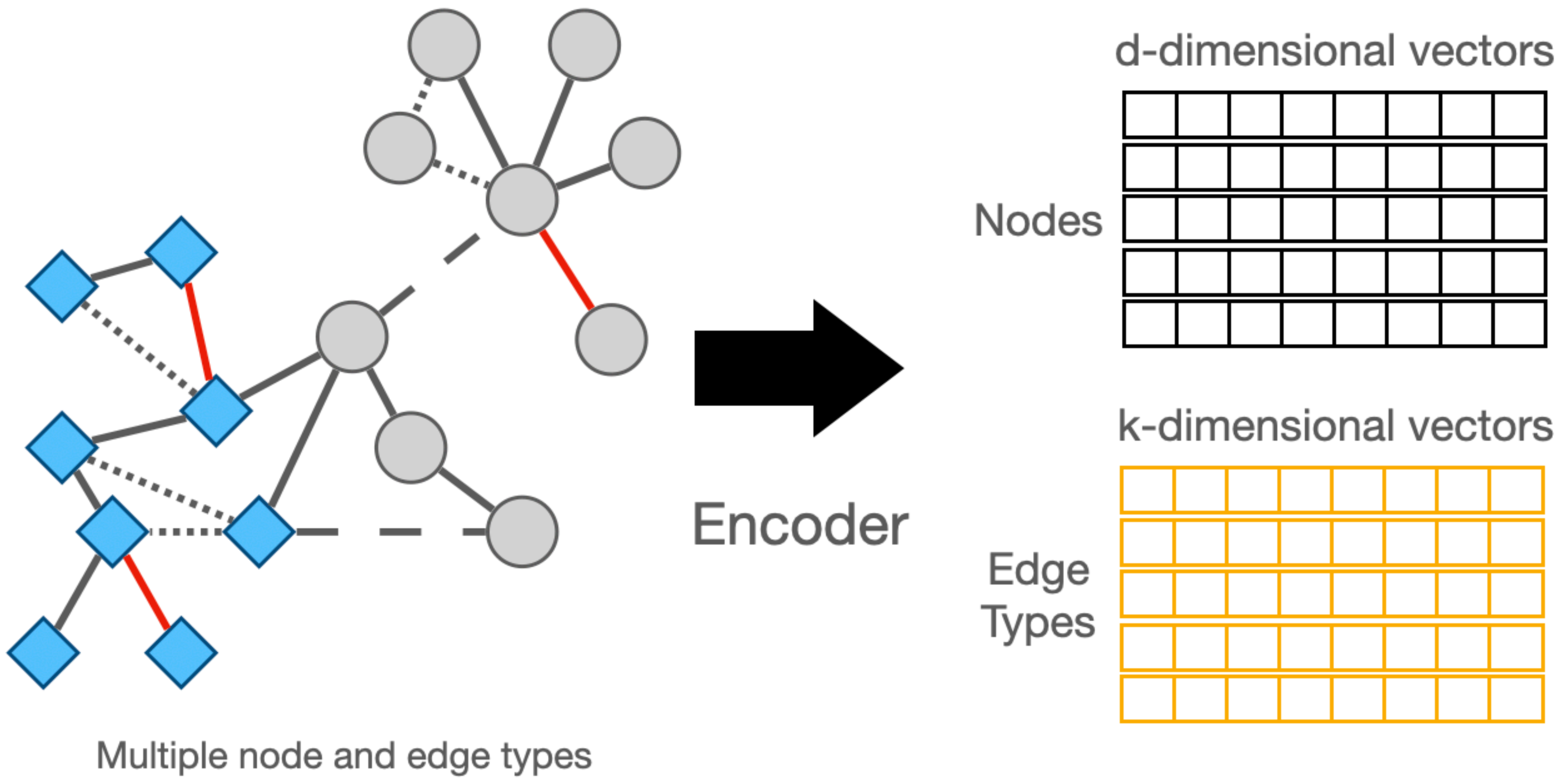
Reasoning requires operating over relational structured data

Modern Machine Learning over graphs



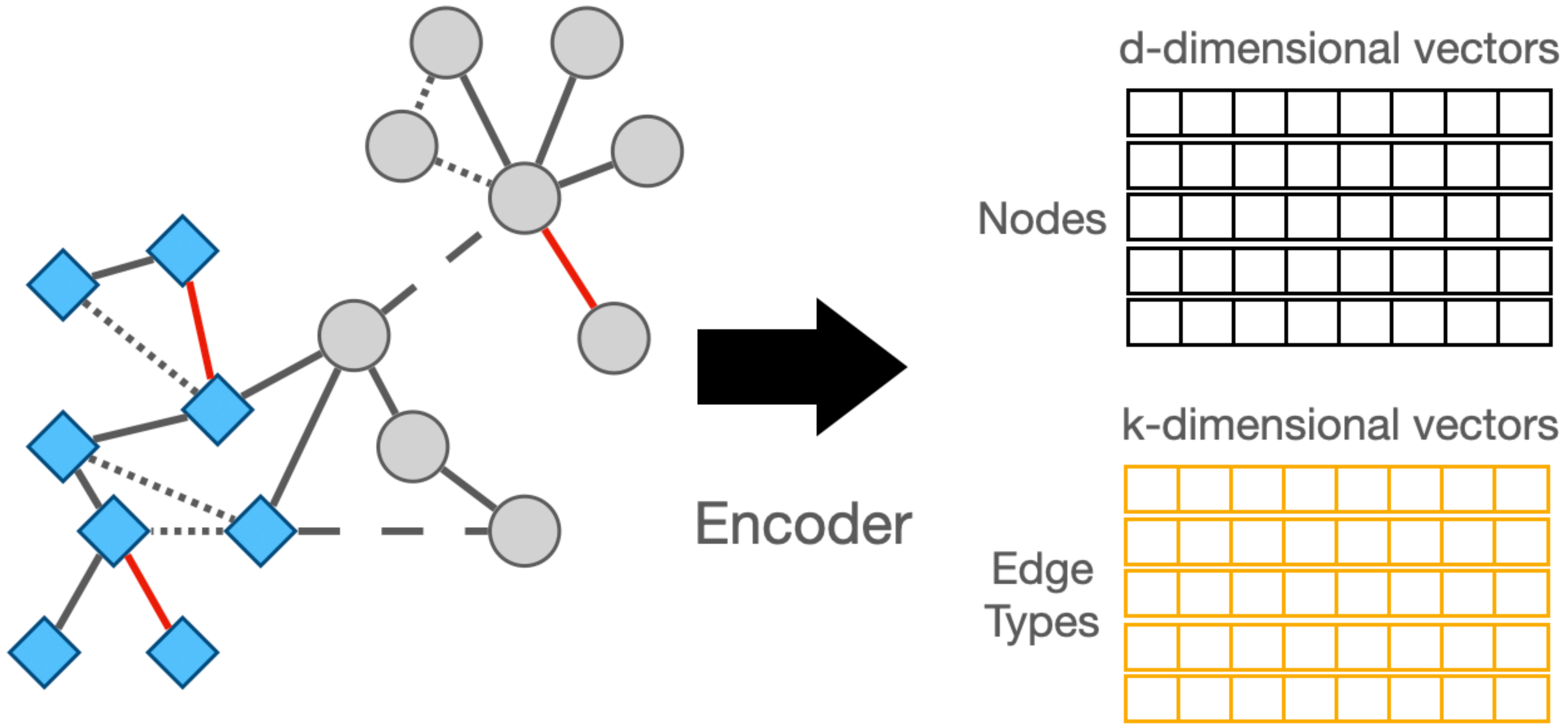
Learned **vector representations** of nodes and edges are key to deep graph learning

Graph learning is memory- and IO-bound



Graphs introduce irregular access patterns

Graph learning is memory- and IO-bound



Multiple node and edge types

Example: Learning Graph Embeddings

Training requires iterating over all edges and retrieving/updating embedding vectors

Training Process

```
// E ordered randomly
for (s, r, d) in E:

    // compute loss of model for an edge
    computeLoss(s, r, d)

    // apply updates to embeddings of edge
    update(s, r, d)
```

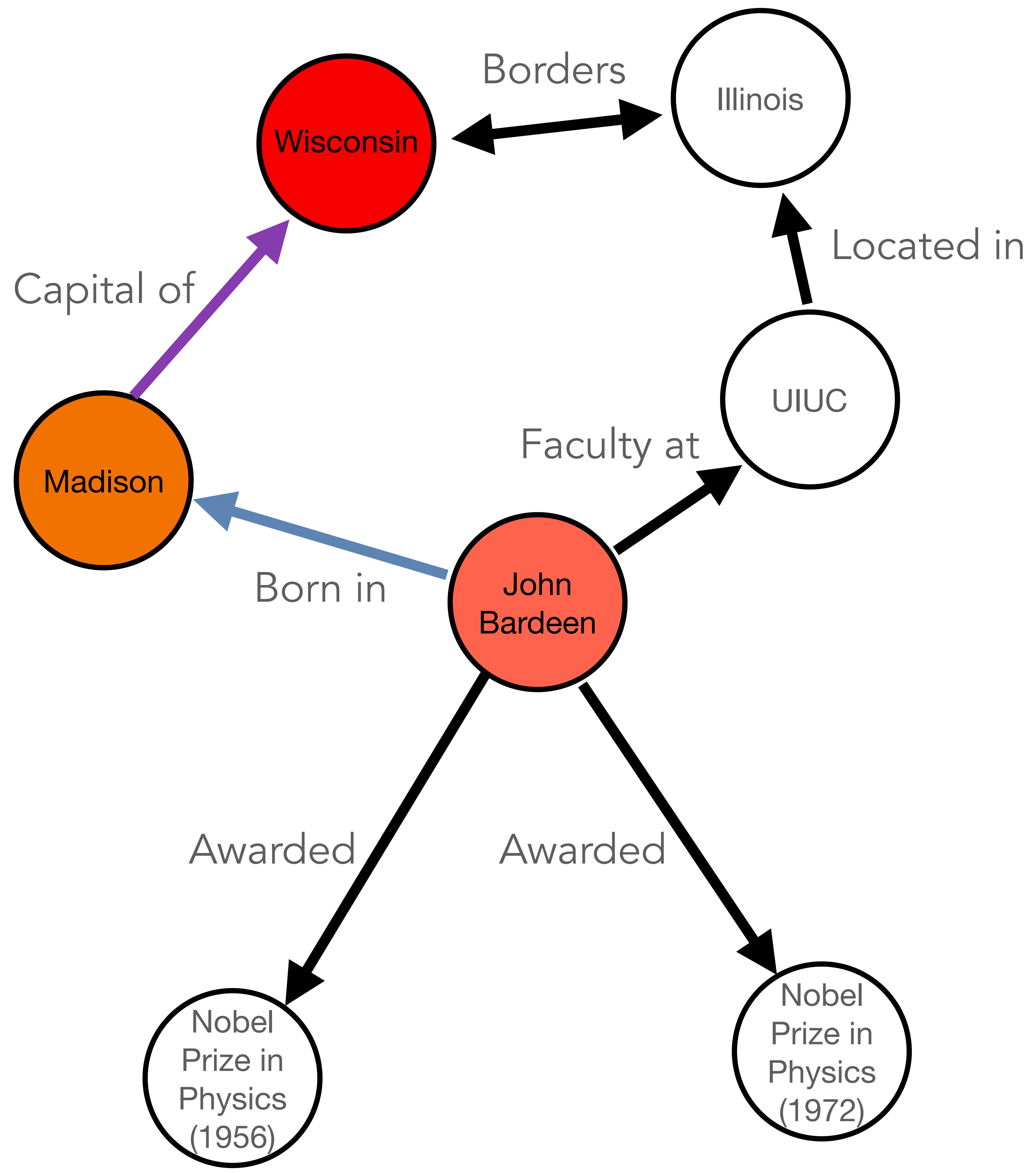
Graph learning is memory- and IO-bound

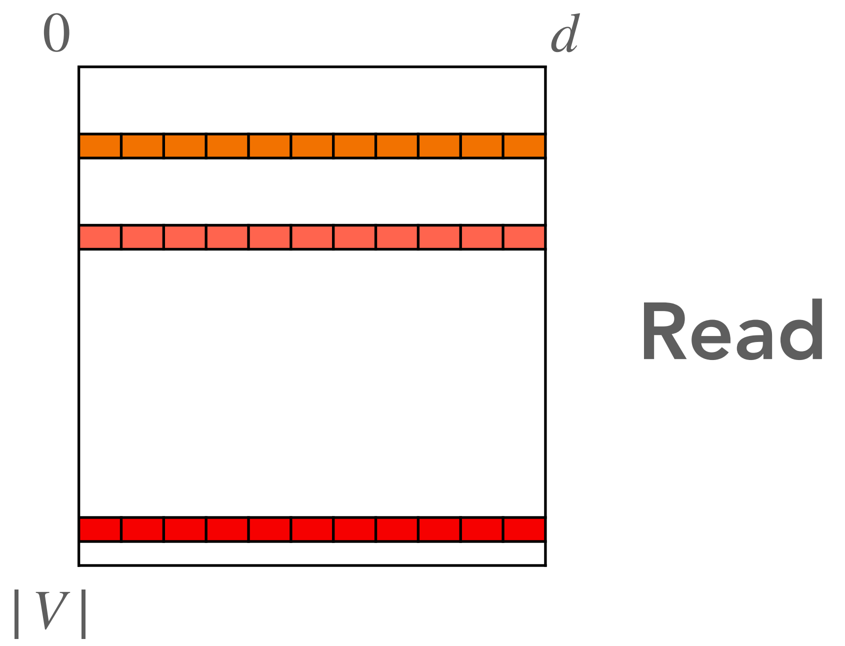
Example: Learning Graph Embeddings

Training requires iterating over all edges and retrieving/updating embedding vectors

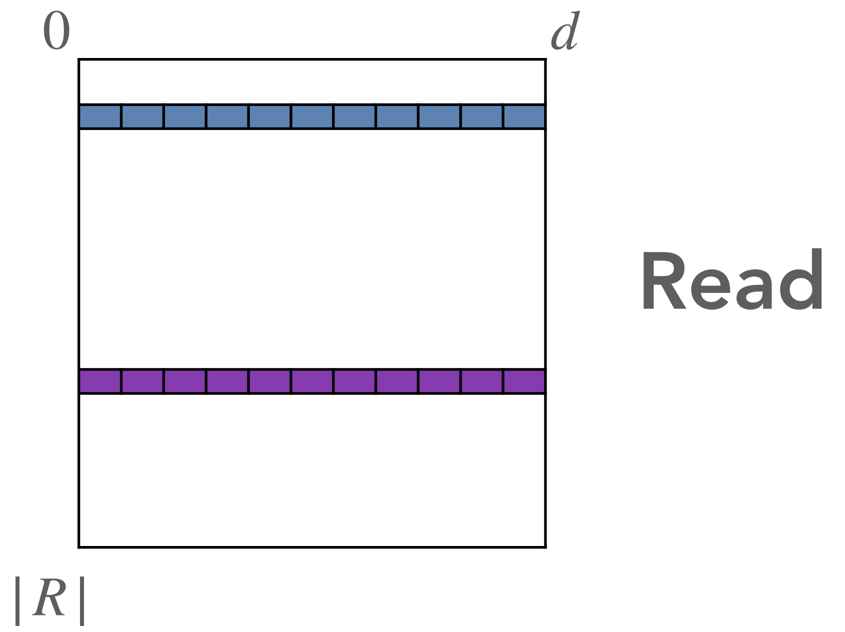
Training Process

```
// E ordered randomly
for (s, r, d) in E:

    // compute loss of model for an edge
    computeLoss(s, r, d)

    // apply updates to embeddings of edge
    update(s, r, d)
```





Node embeddings Table



Edge-type embeddings Table

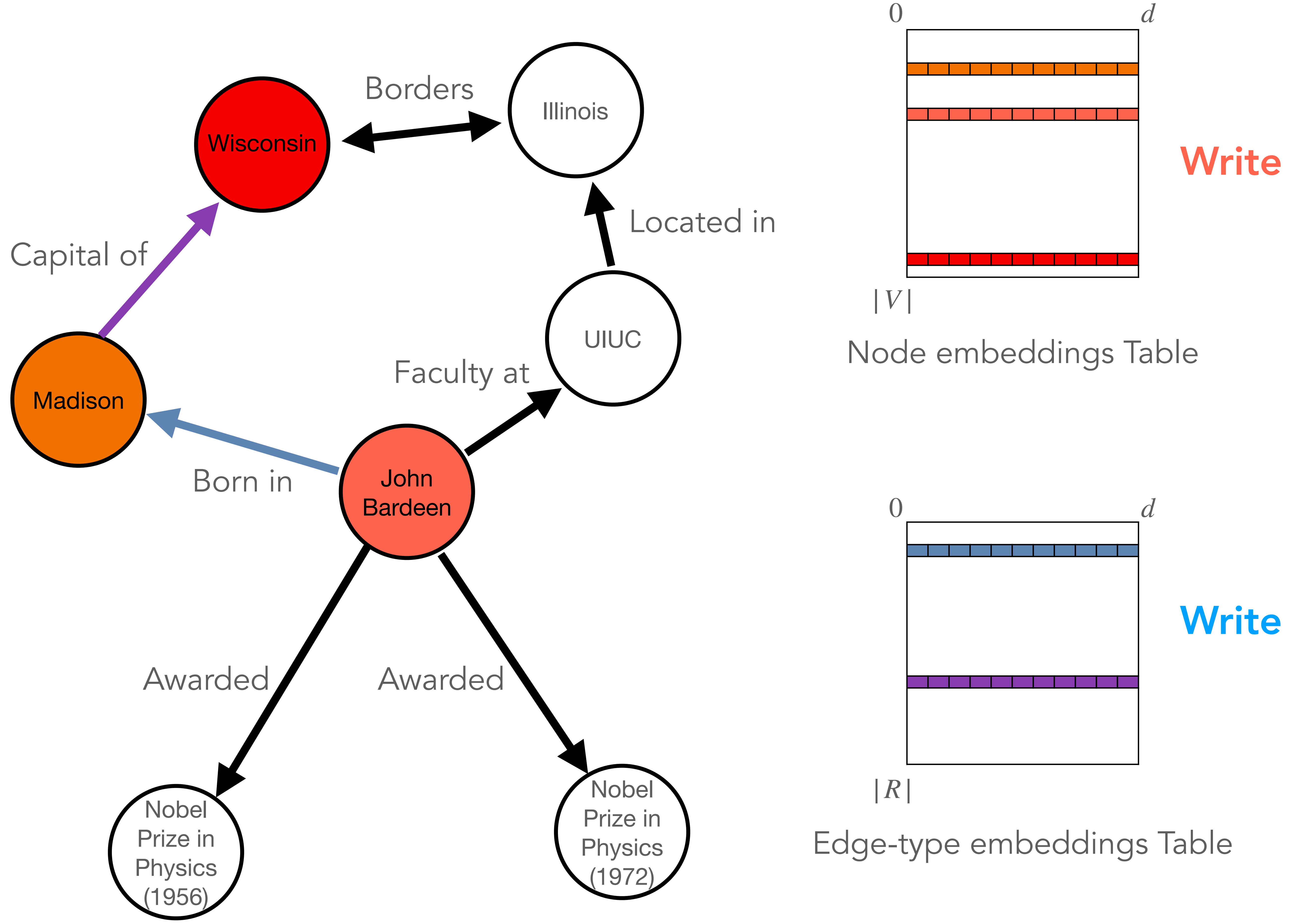
Graph learning is memory- and IO-bound

Example: Learning Graph Embeddings

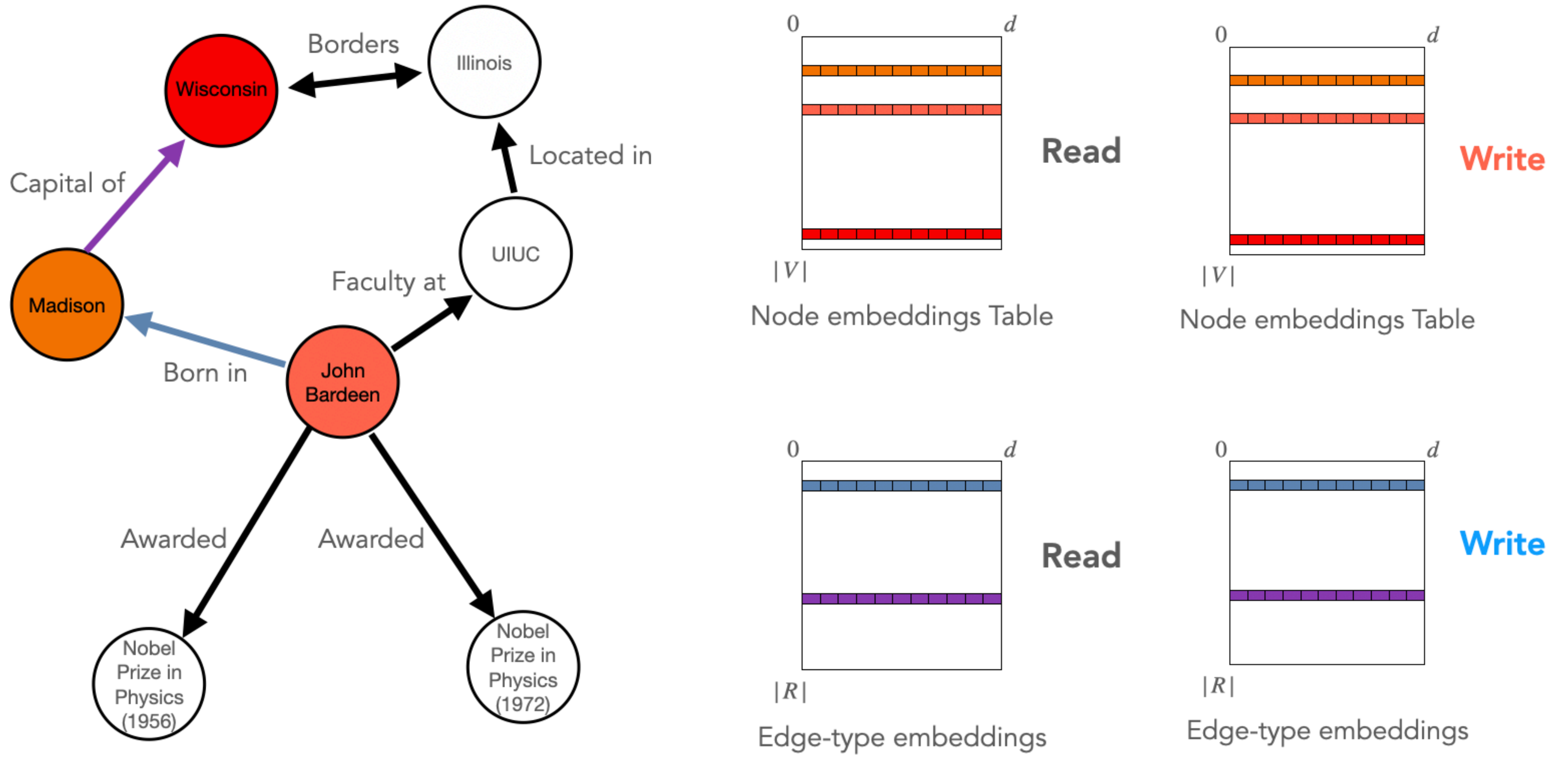
Training requires iterating over all edges and retrieving/updating embedding vectors

Training Process

```
// E ordered randomly
for (s, r, d) in E:

    // compute loss of model for an edge
    computeLoss(s, r, d)

    // apply updates to embeddings of edge
    update(s, r, d)
```



Graph learning is memory- and IO-bound



Freebase86m:

- 338 million edges, 86 million nodes, 15,000 edge types
- Size of node embedding table for d = 400:

$\boxed{\text{86 million x 400 x 4 bytes = 138 GB}}$

AWS P3.2xLarge instance:

- 16 GB GPU Memory
- 64 GB CPU Memory

Embedding tables **do not fit** in GPU memory

Moving embeddings to compute

marius

1. Store embeddings in CPU memory and transfer to GPU(s)
   - Bottlenecked by transfer overheads
   - Limited scalability

DGL

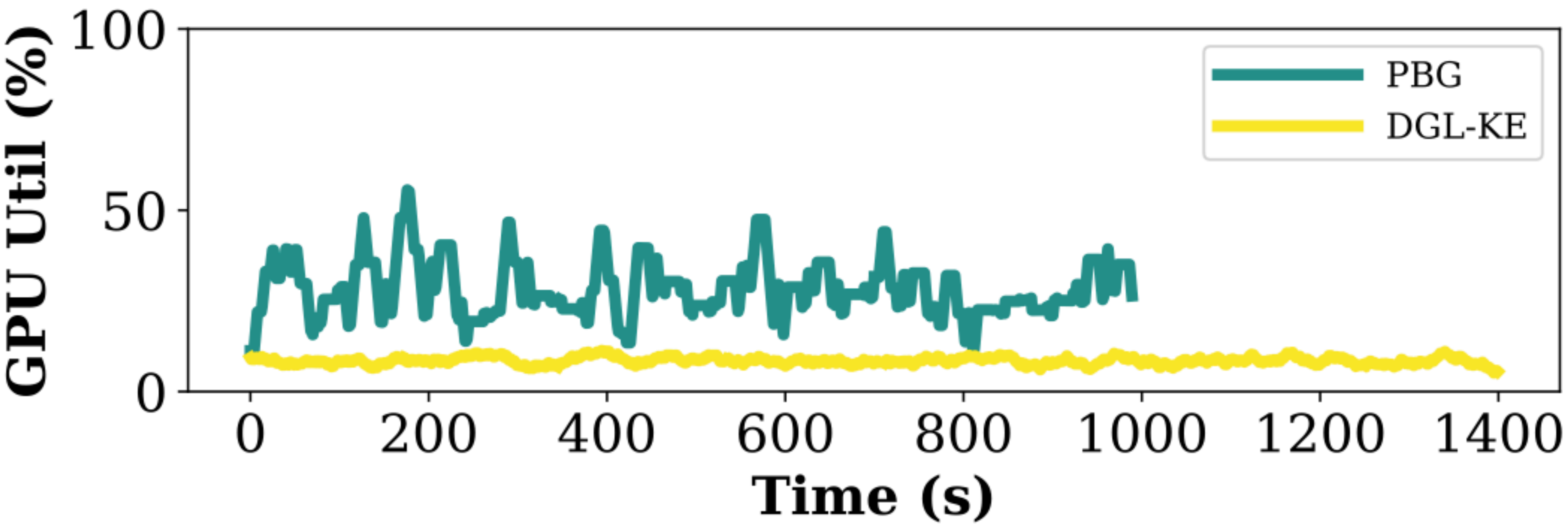
2. Partition node embeddings and store on disk
   - Limited by disk throughput

PyTorch Big-Graph (PBG)

3. Distribute embeddings across multiple machines
   - Bottlenecked by transfer overheads
   - Expensive

PBG & DGL

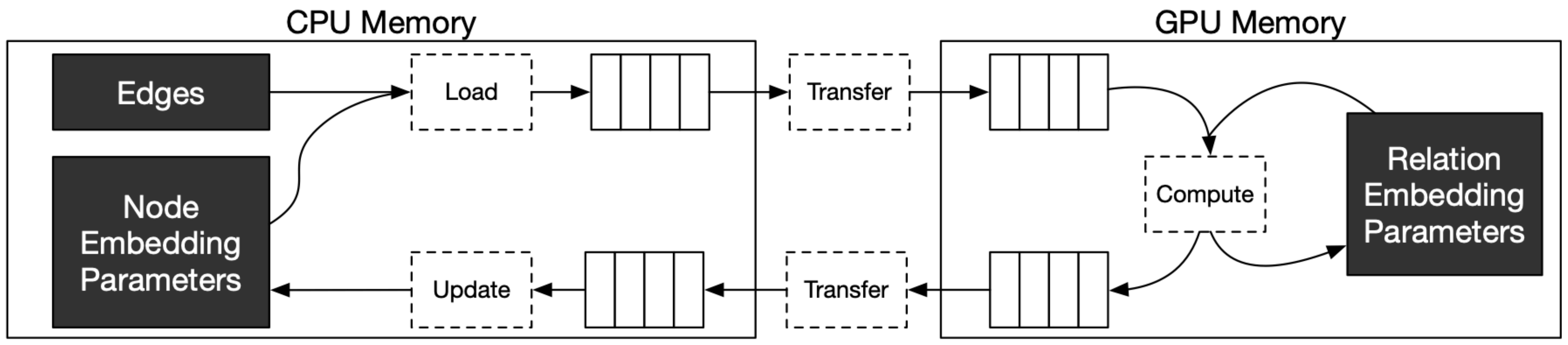
Moving embeddings to compute

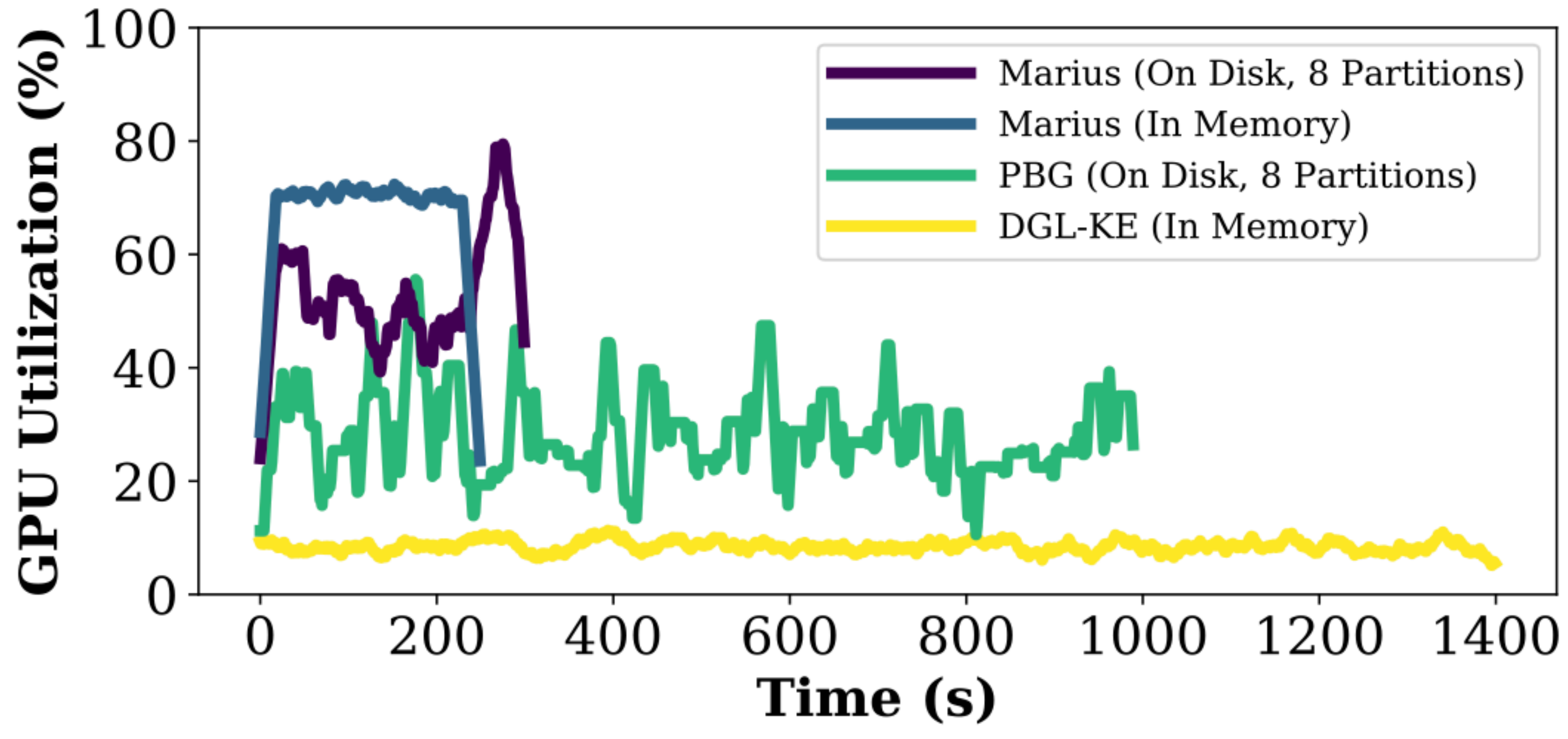


The **key bottleneck** when training graph learning models **is data movement**

Marius: Scalable graph learning

Learning Massive Graph Embeddings on a Single Machine, OSDI'2021
Find more at: **marius-project.org**





Pipelining and a novel data replacement policy allow Marius to maximize resource utilization of the entire memory hierarchy (including disk, CPU, and GPU memory)

Achieves graph learning over billion edge graphs **in a single machine**

Marius: Scalable graph learning

System	Model	MRR	Hits		Time
			@1	@10	
PBG	Dot	.313	.239	.451	5h15m
DGL-KE	Dot	.220	.153	.385	35h3m
Marius	Dot	.310	.236	.445	**3h28m**

*MRR: mean reciprocal rank (higher is better)

Measuring time-to-reconstruction-accuracy for
Dot-Product graph embeddings over the Twitter graph
(41.6M nodes and 1.5B edges)

Marius can be **10x faster** than competing methods in a single box

Marius: Scalable graph learning

System	Deployment	Epoch Time (s)	Per Epoch Cost ($)
Marius	1-GPU	288	**.248**
DGL-KE	2-GPUs	761	1.29
DGL-KE	4-GPUs	426	1.45
DGL-KE	8-GPUs	220	1.50
DGL-KE	Distributed	1237	1.69
PBG	1-GPU	1005	.85
PBG	2-GPUs	430	.73
PBG	4-GPUs	330	1.12
PBG	8-GPUs	273	1.86
PBG	Distributed	1199	1.64

Per-epoch runtime and monetary cost ($) for embedding the Freebase Knowledge Graph (86M nodes and 338M edges)

Marius can be **5x cheaper** than competing methods; single-box (1GPU) Marius has comparable runtime with multi-GPU solutions marius

Open-source Marius

PyTorch
Compatible

Installation from source with Pip

1. Install latest version of PyTorch for your CUDA version:

   Linux:

   - CUDA 10.1: `python3 -m pip install torch==1.7.1+cu101 -f https://download.pytorch.org/whl/torch_stable.html`
   - CUDA 10.2: `python3 -m pip install torch==1.7.1`
   - CPU Only: `python3 -m pip install torch==1.7.1+cpu -f https://download.pytorch.org/whl/torch_stable.html`

   MacOS:

   - CPU Only: `python3 -m pip install torch==1.7.1`

2. Clone the repository `git clone https://github.com/marius-team/marius.git`
3. Build and install Marius `cd marius; python3 -m pip install .`

docker

Marius in Docker

Marius can be deployed within a docker container. Here is a sample ubuntu dockerfile (loca `examples/docker/dockerfile`) which contains the necessary dependencies preinstalled f

Building and running the container

Build an image with the name `marius` and the tag `example` :
`docker build -t marius:example -f examples/docker/dockerfile examples/docker`

Marius

Search docs

CONTENTS

Introduction
Quick Start
Build
System Overview
Configuration
IO Format
Training
Models
Loss Functions
Evaluation
Storage Backends

API

Batch
Buffer
Config
DataSet
Datatypes
Decoder
Encoder
Evaluator
IO
Logger
Marius
Model
Ordering
Pipeline
Storage
Trainer
Util

» Batch

View page source

Batch

class `Batch`

Contains metadata, edges and embeddings for a single batch.

Subclassed by PartitionBatch

Public Functions

`Batch`(bool *train*)

Constructor

`~Batch`()

void `localSample`()

Destructor Construct additional negative samples and neighborhood information from the batch

void `accumulateUniqueIndices`()

Populates the unique_<>_indices tensors

void `embeddingsToDevice`(int *device_id*)

Transfers embeddings, optimizer state, and indices to specified device

void `prepareBatch`()

Populates the src_pos_embeddings, dst_post_embeddings, relation_embeddings, src_neg_embeddings, and dst_neg_embeddings tensors for model computation

void `accumulateGradients`()

Accumulates gradients into the unique_node_gradients and unique_relation_gradients tensors, and applies optimizer update rule to create the unique_node_gradients2 and unique_relation_gradients2 tensors

void `embeddingsToHost`()

Transfers gradients and embedding updates to host

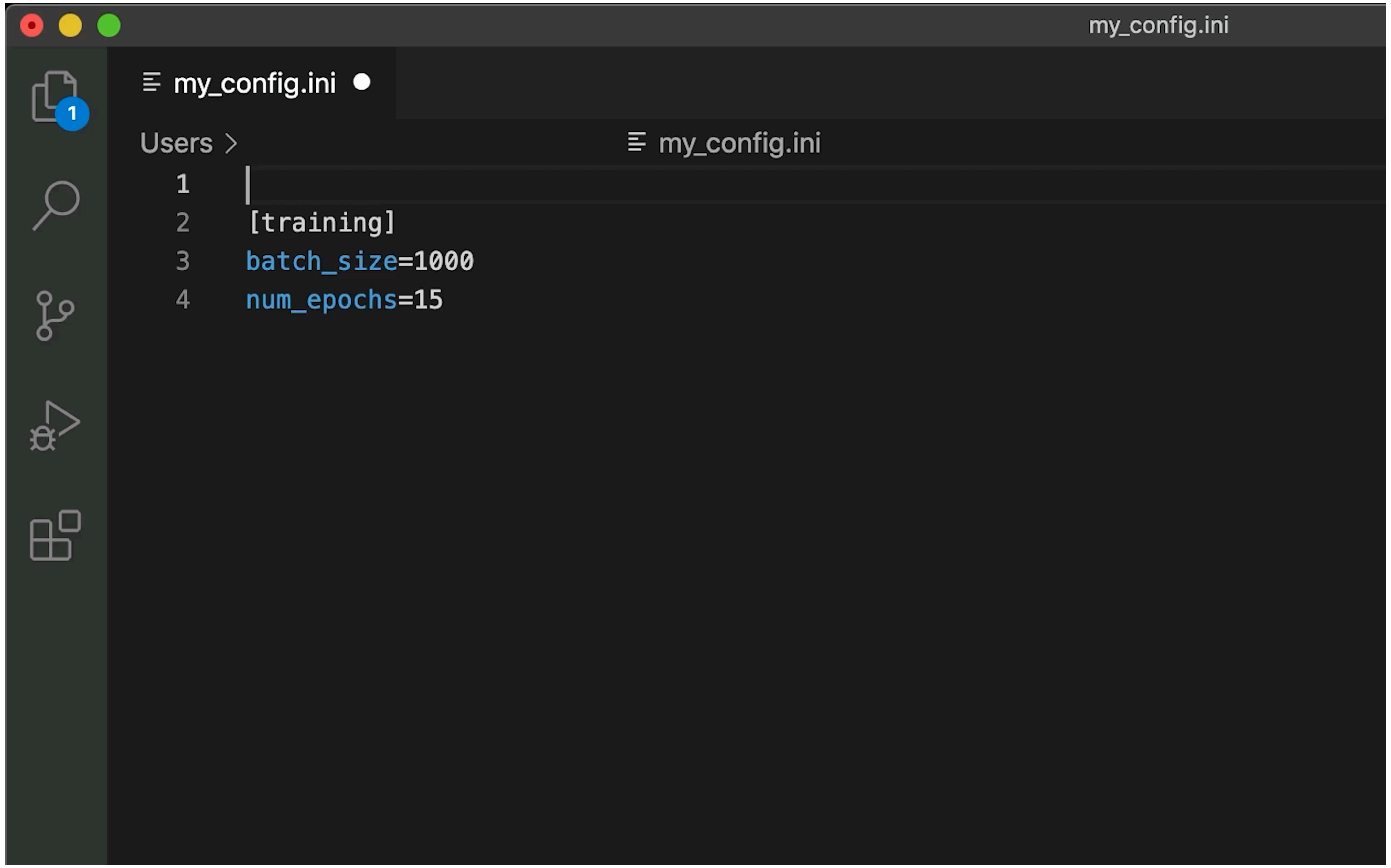
Released at:
marius-project.org

Apache-2.0 License

Using Marius

Config-based development

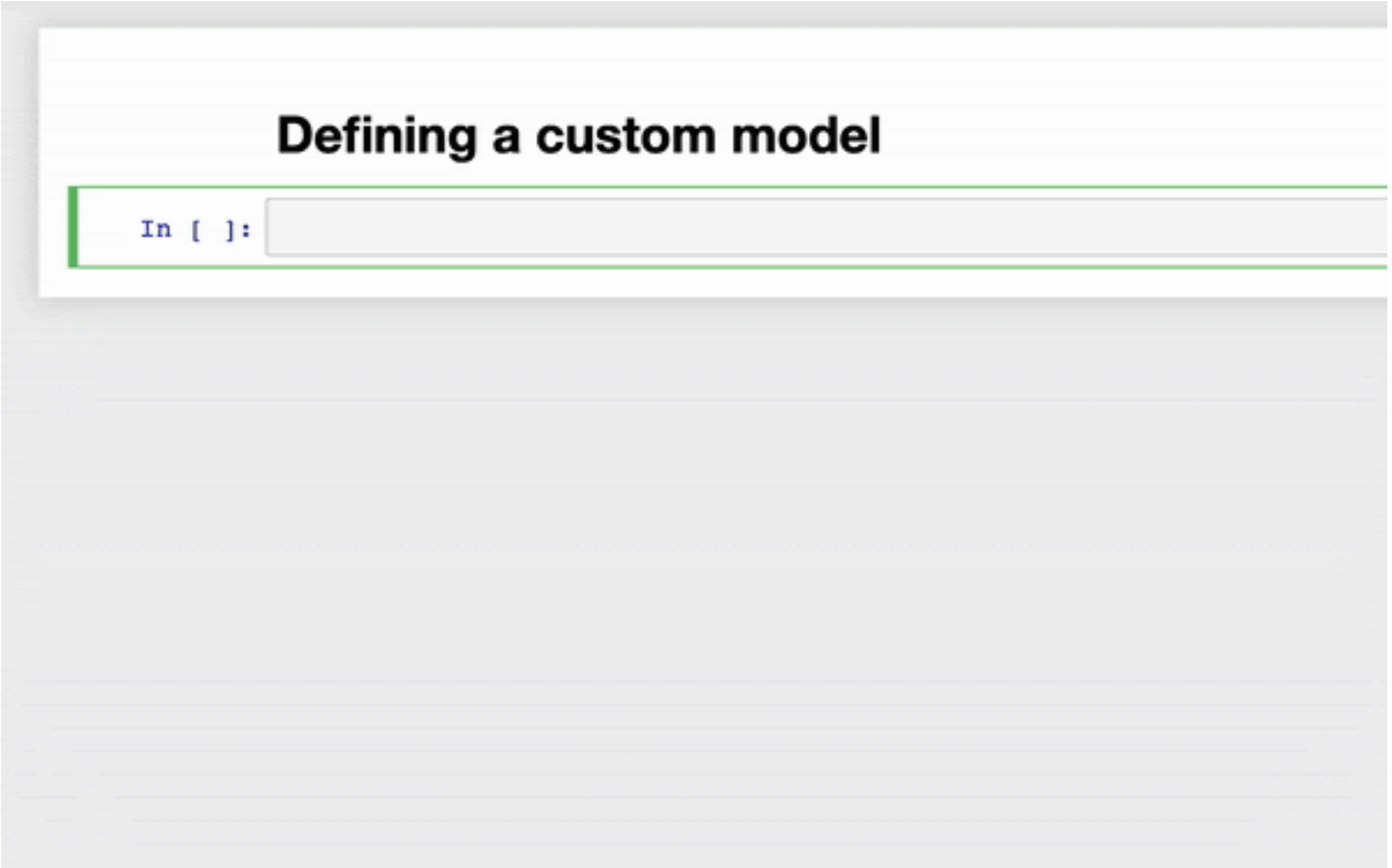
- No-code paradigm: running Marius only requires a simple configuration file
- Customize parameters, defaults provided if not specified
- Easy to run from command line



Using Marius

Extensible

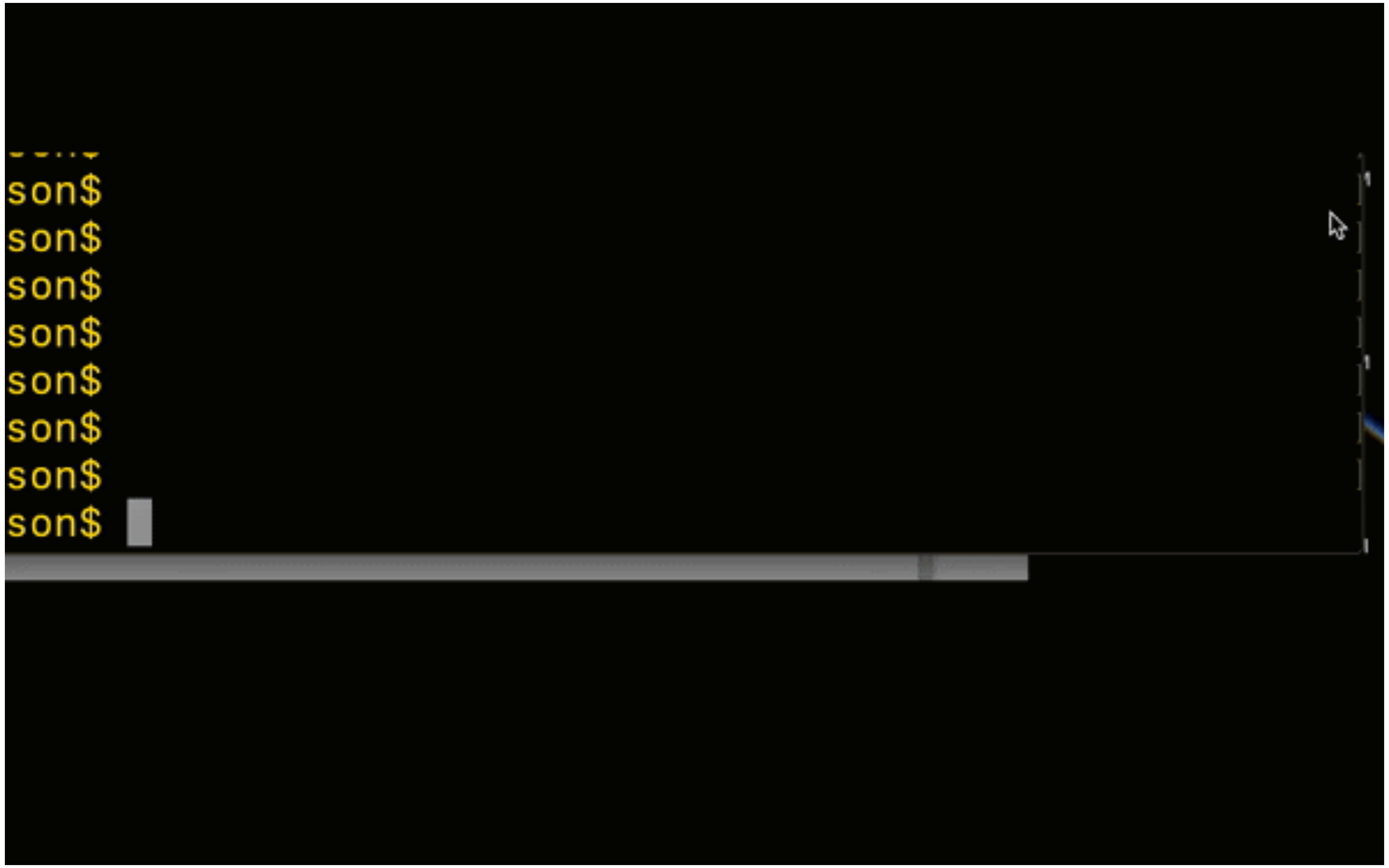
- Features a Python API
- Write custom models
- High-degree of control and customization



Using Marius

Interoperability

- Multiple data converters to transform raw data into the Marius input format
- Support for conversion of TSV, CSV, Parquet file formats
- Output embeddings can be converted to commonly used types such as PyTorch tensors



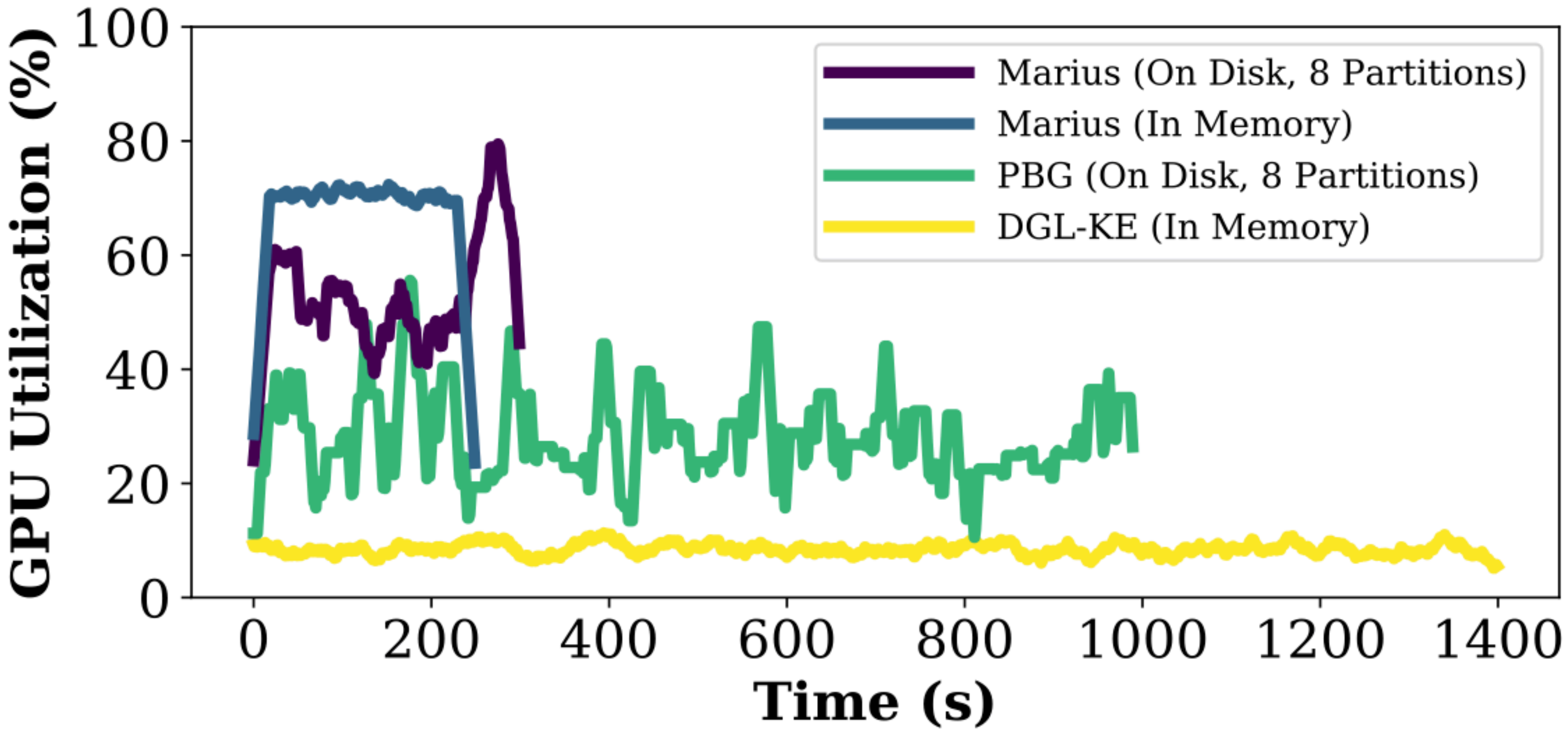
Key innovation in Marius

Method

- Use pipelining and async IO hide data movement
- Utilize the full memory hierarchy with a partition buffer
- **Minimize IO with Buffer-aware Edge Traversal Algorithm (BETA)**

Results

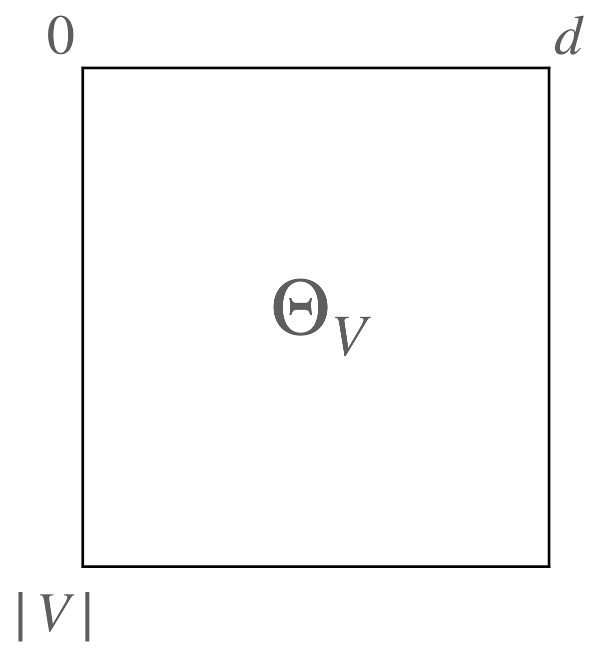
- 10x reduction in runtime vs. DGL-KE on Twitter
- 3.7x runtime reduction vs. PBG on Freebase86m
- 2x higher utilization than PBG, 6-8x higher utilization than DGL-KE



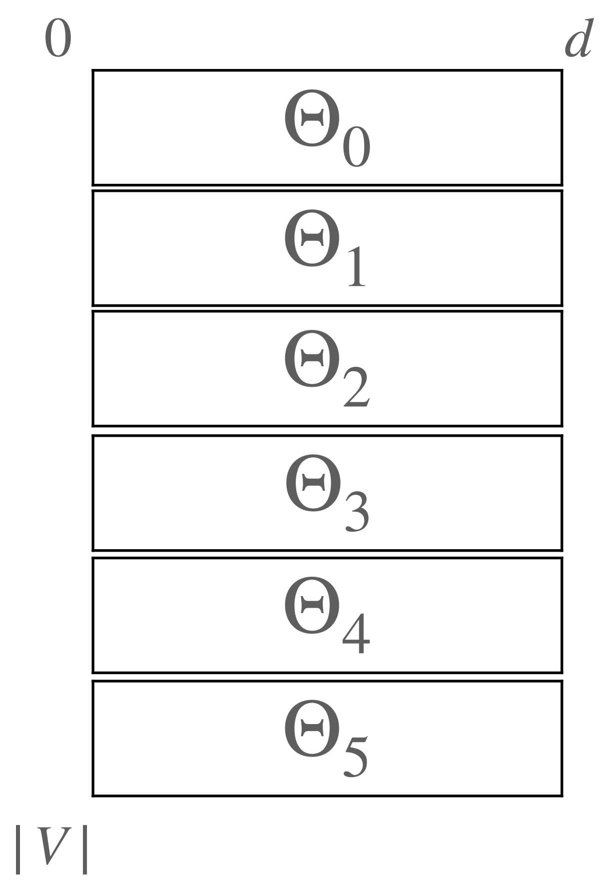
Partition-based processing

Node Embedding Partitions

Node embeddings are partitioned uniformly into p disjoint partitions.



Node embedding table

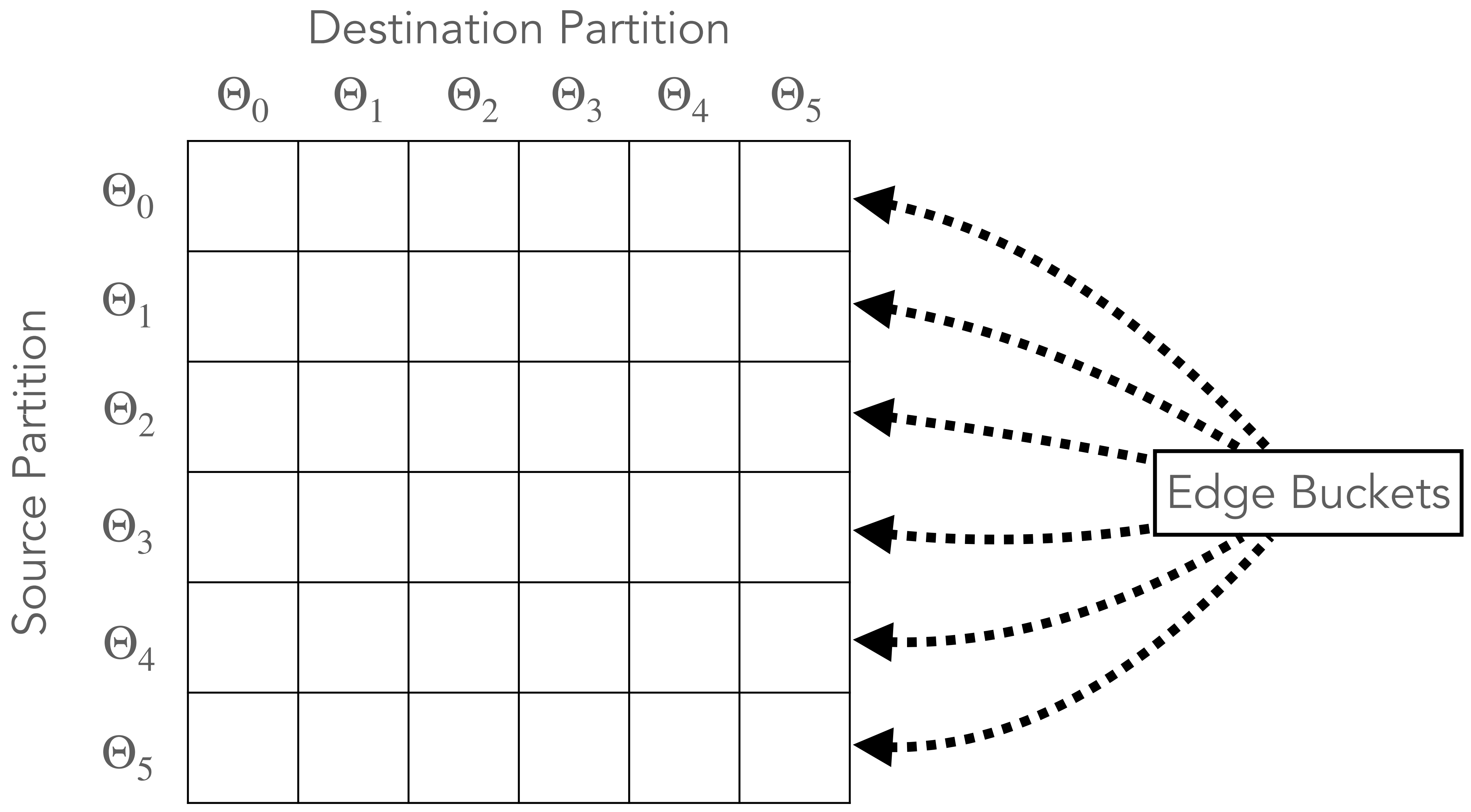


Partitioned node embedding table (p = 6)

Edge Buckets

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

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



Global Adjacency Matrix

Edge bucket ordering and IO

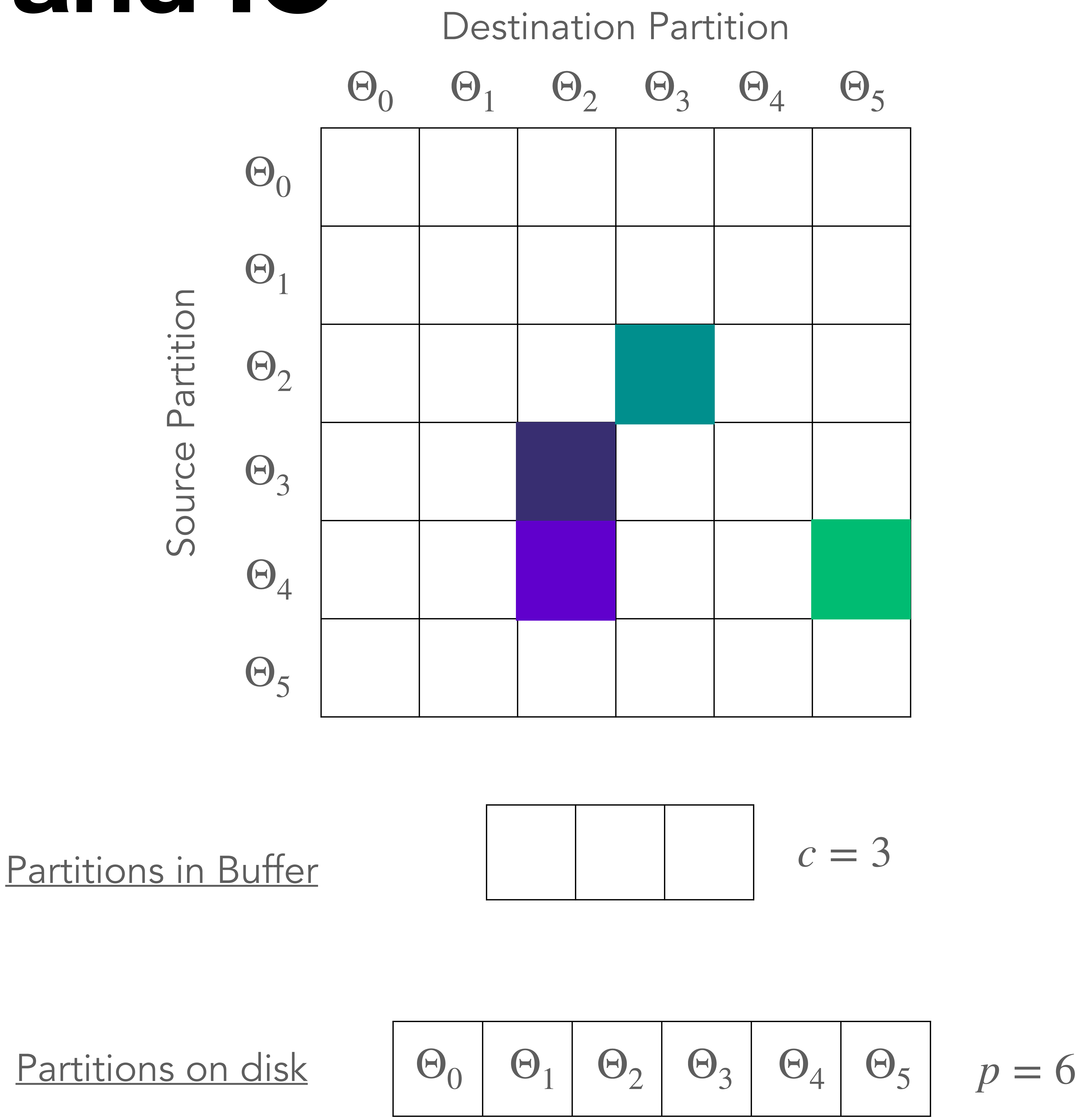
The order in which edge buckets are processed has an impact on IO

Example: After processing edge bucket (3, 2)

Processing (2, 3): Requires no extra swaps

Processing (2, 4): Requires one swap

Processing (4, 5): Requires two swaps



Edge bucket ordering and IO

Random Ordering ~23 swaps

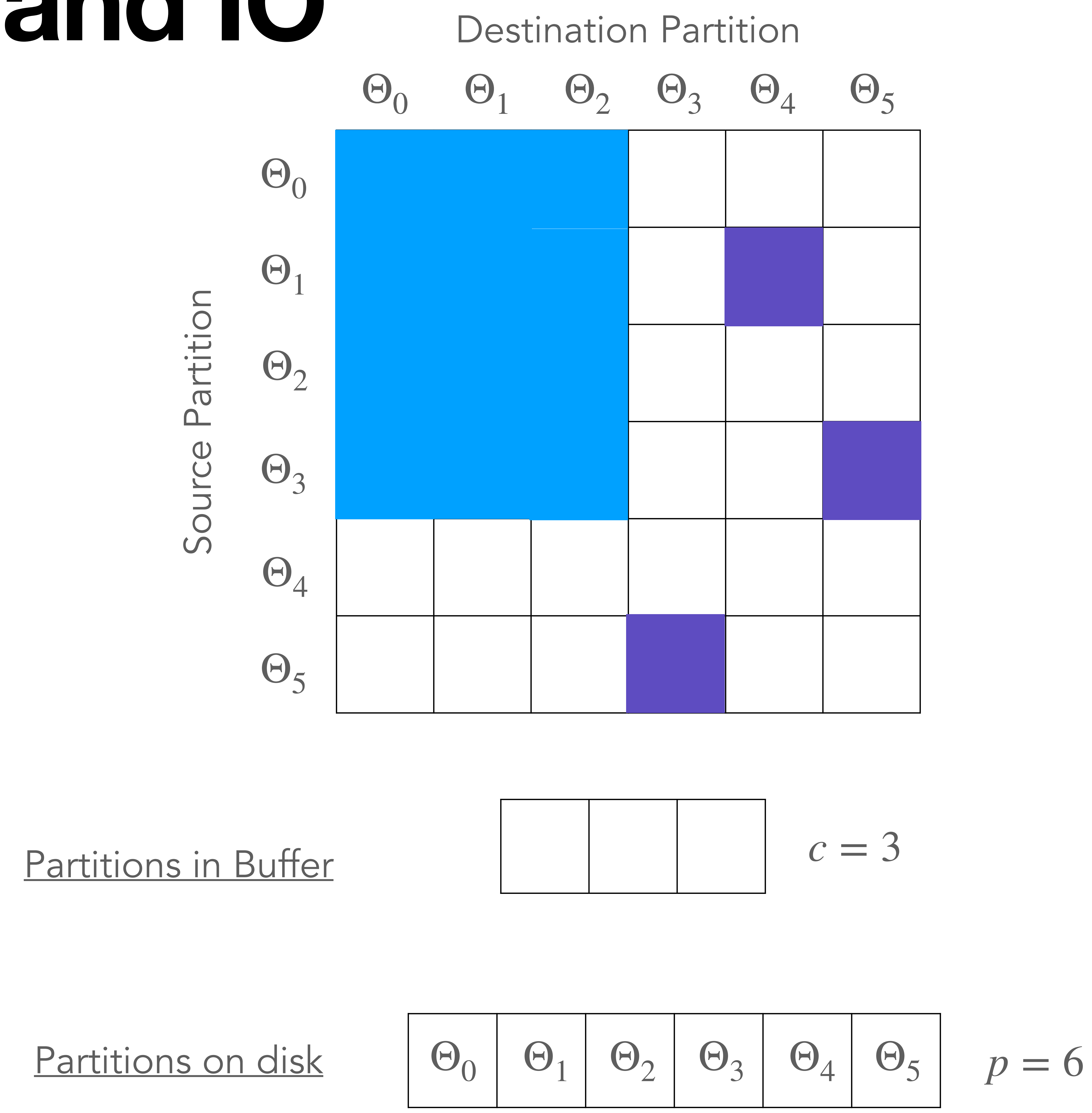
Locality-aware Ordering 12 swaps

We show a Lower Bound

Can never process more than 2c - 1 edge buckets per swap

$$\left\lceil \frac{p^2 - c^2}{2c - 1} \right\rceil = \left\lceil \frac{6^2 - 3^2}{2 * 3 - 1} \right\rceil = 6$$

We propose an ordering which is close to this bound



Buffer-aware Edge Traversal Algorithm (BETA)

BETA Ordering

1. Randomly initialize buffer

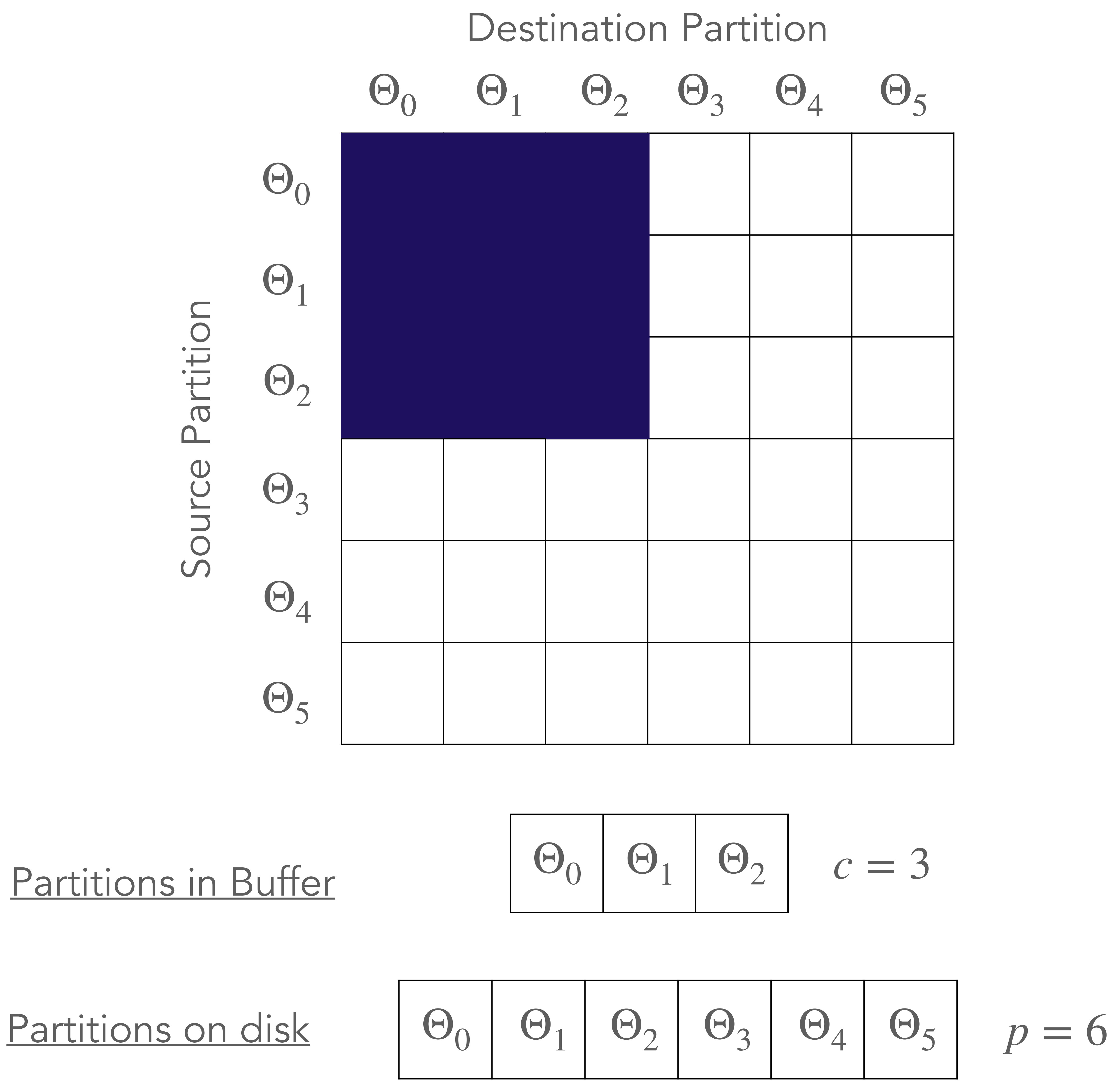
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets

3. Fix a new c - 1 partitions and repeat until all edge buckets have been processed

Buffer-aware Edge Traversal Algorithm (BETA)

BETA Ordering

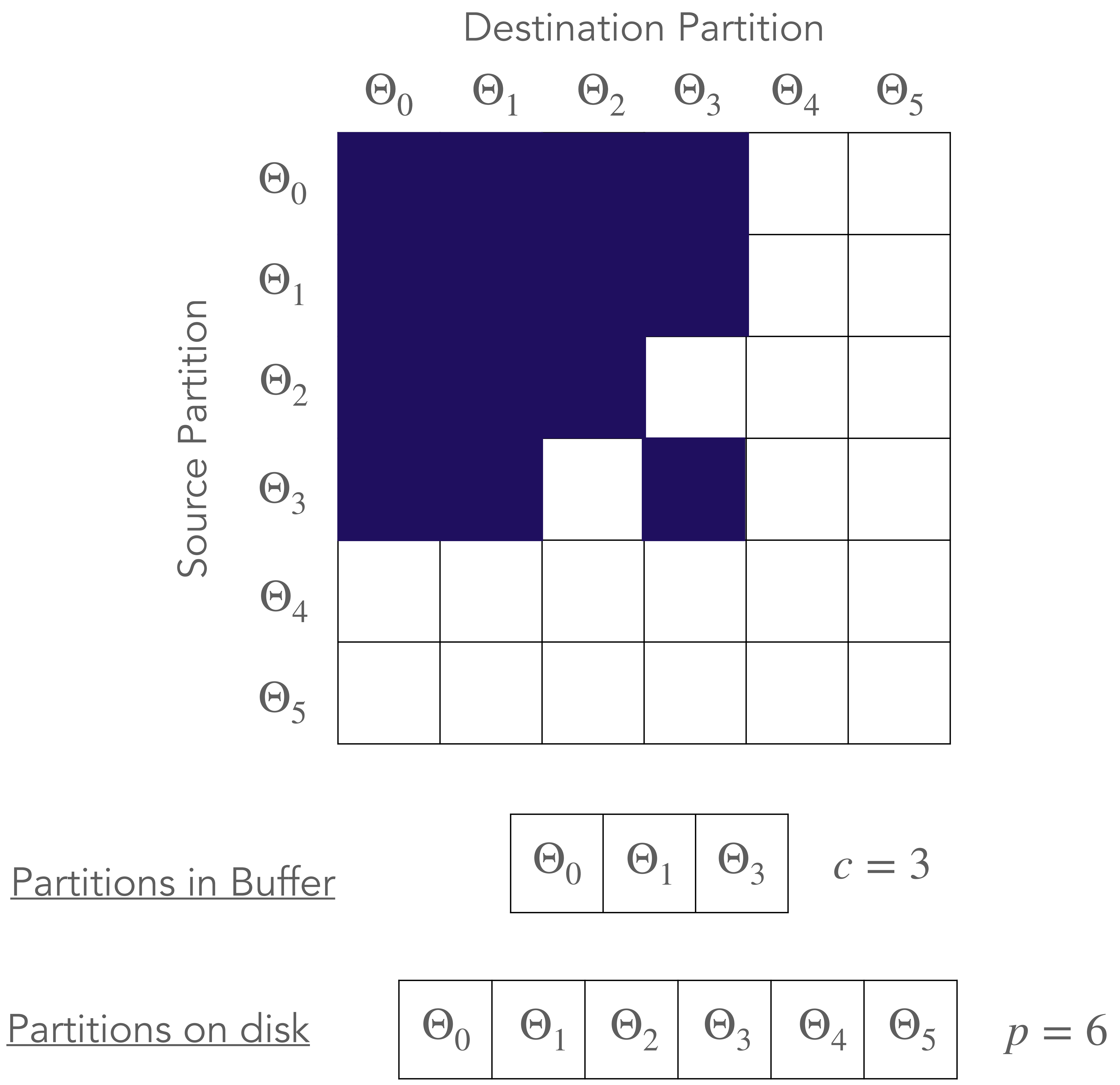
1. **Randomly initialize buffer**
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new c - 1 partitions and repeat until all edge buckets have been processed



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



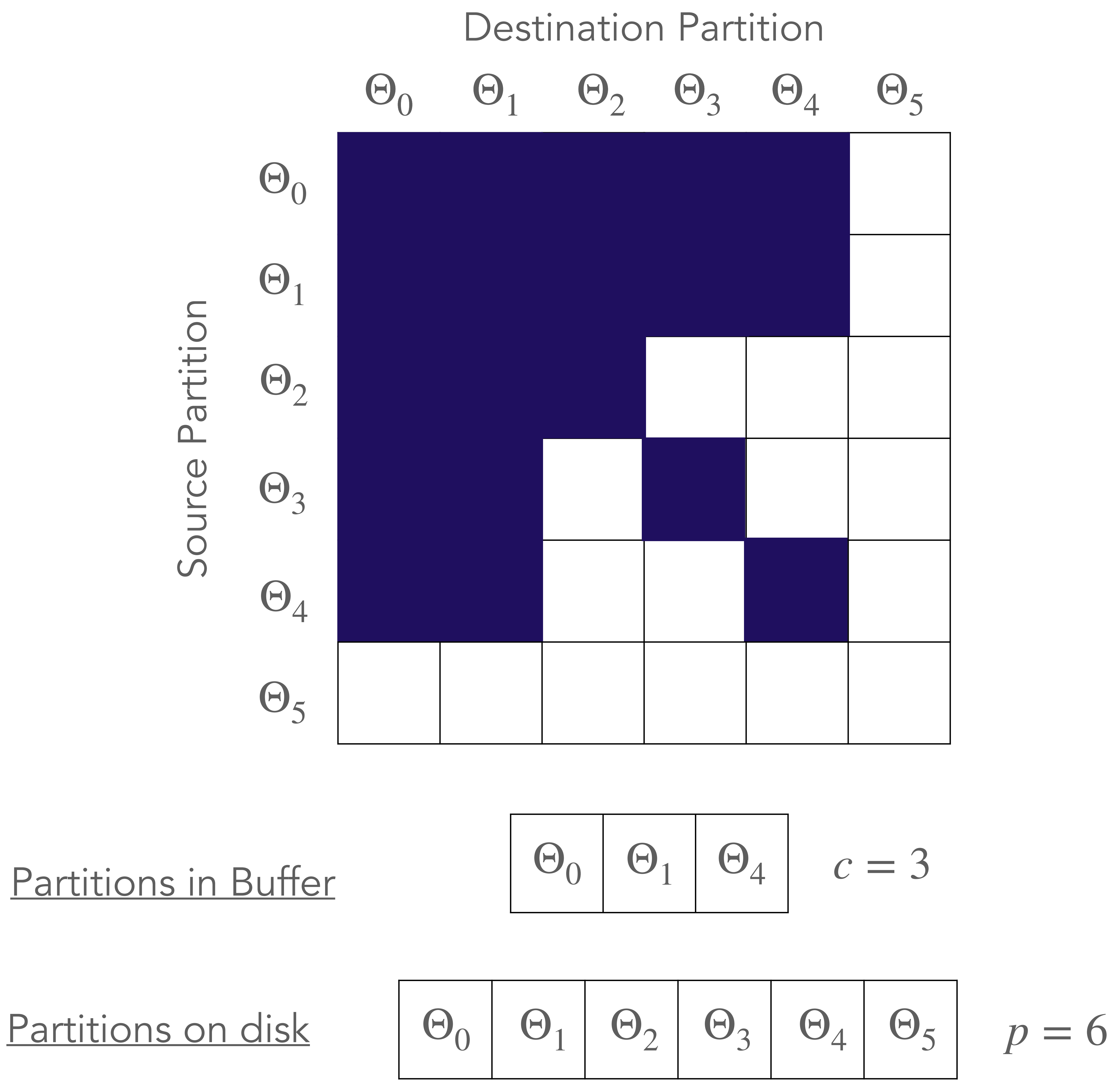
Buffer-aware Edge Traversal Algorithm (BETA)

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



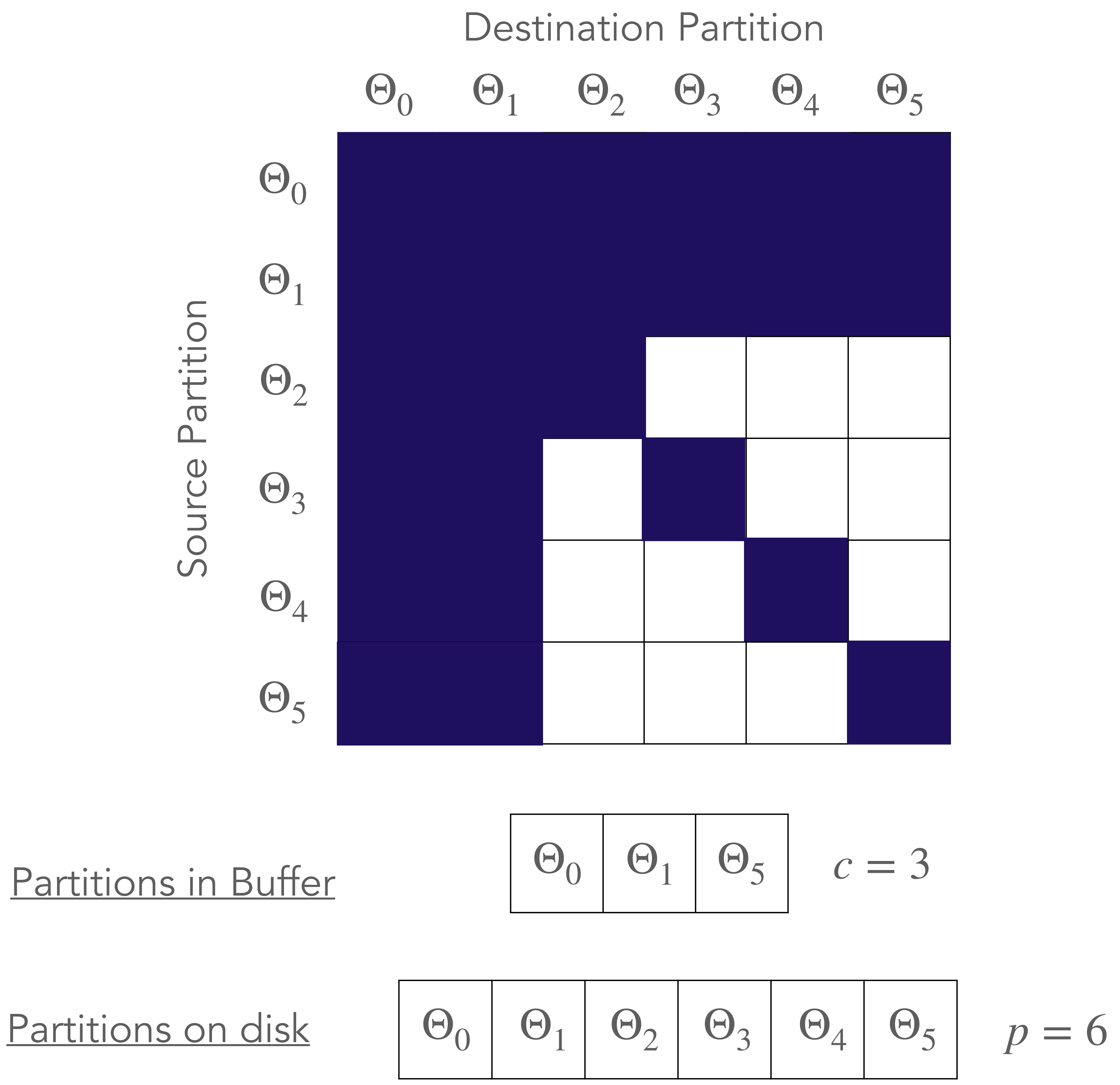
Buffer-aware Edge Traversal Algorithm (BETA)

BETA Ordering

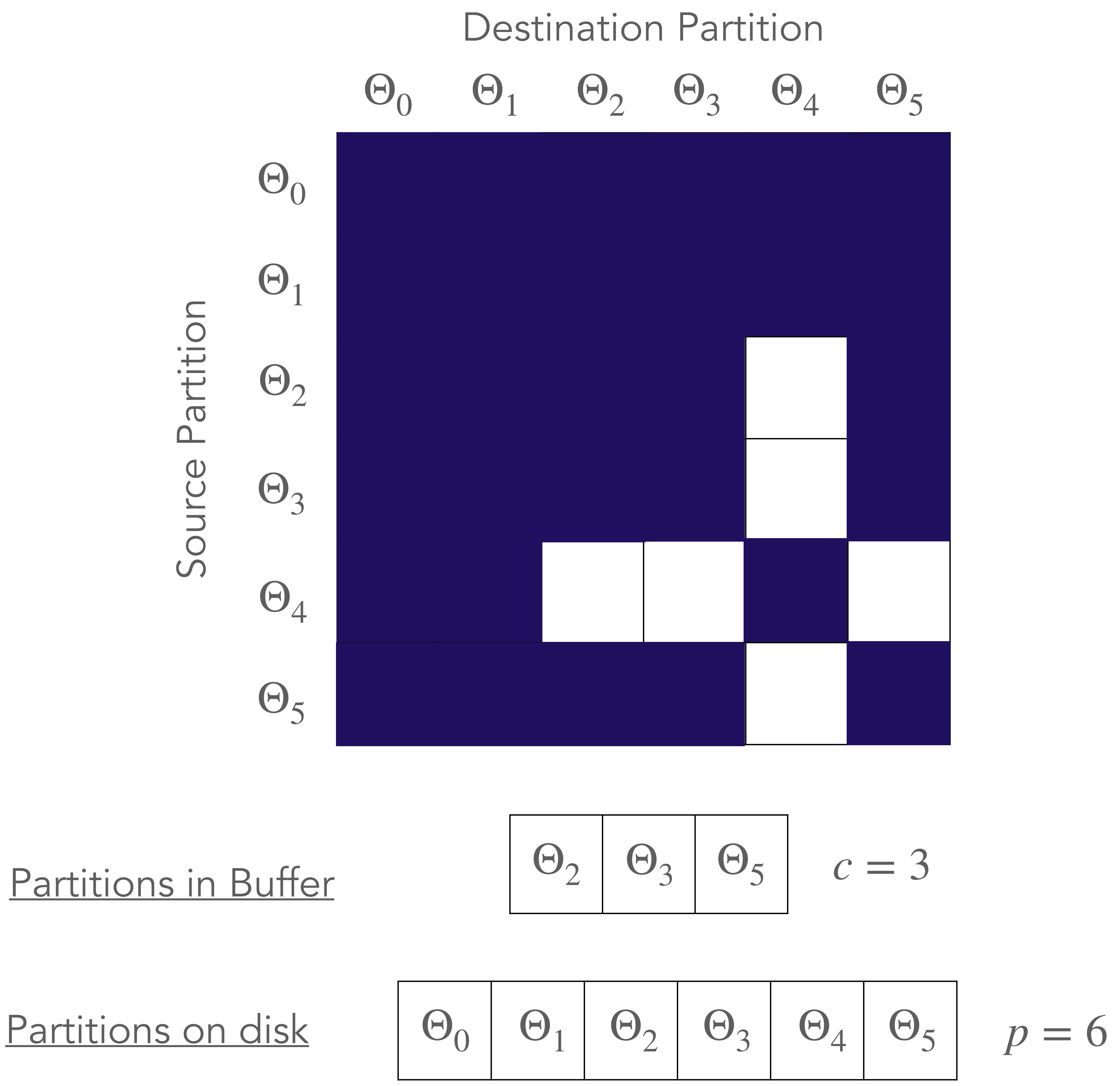
1. Randomly initialize buffer

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



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**

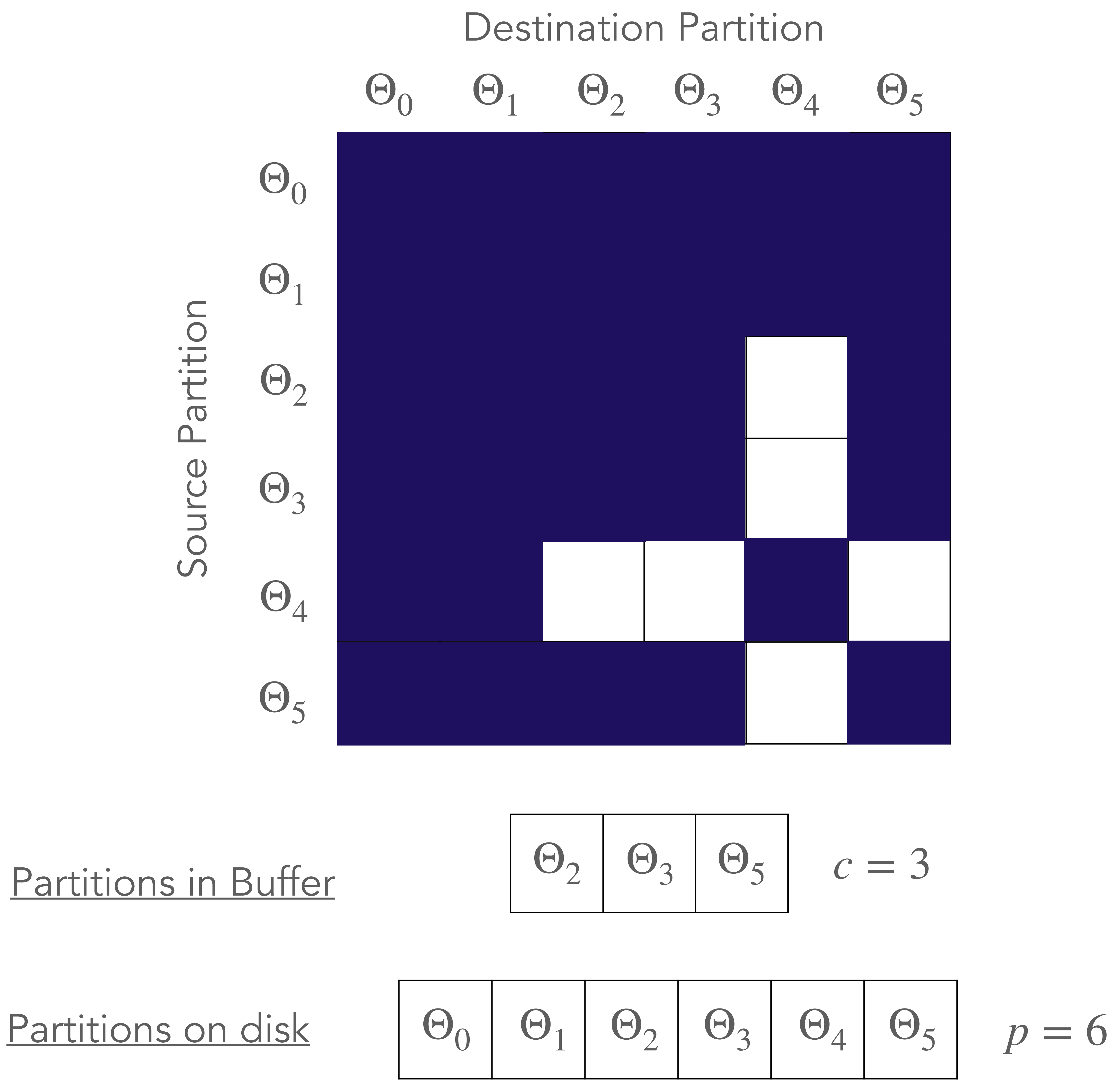
Buffer-aware Edge Traversal Algorithm (BETA)

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



BETA ordering gives 7 swaps (6 is the lower bound)

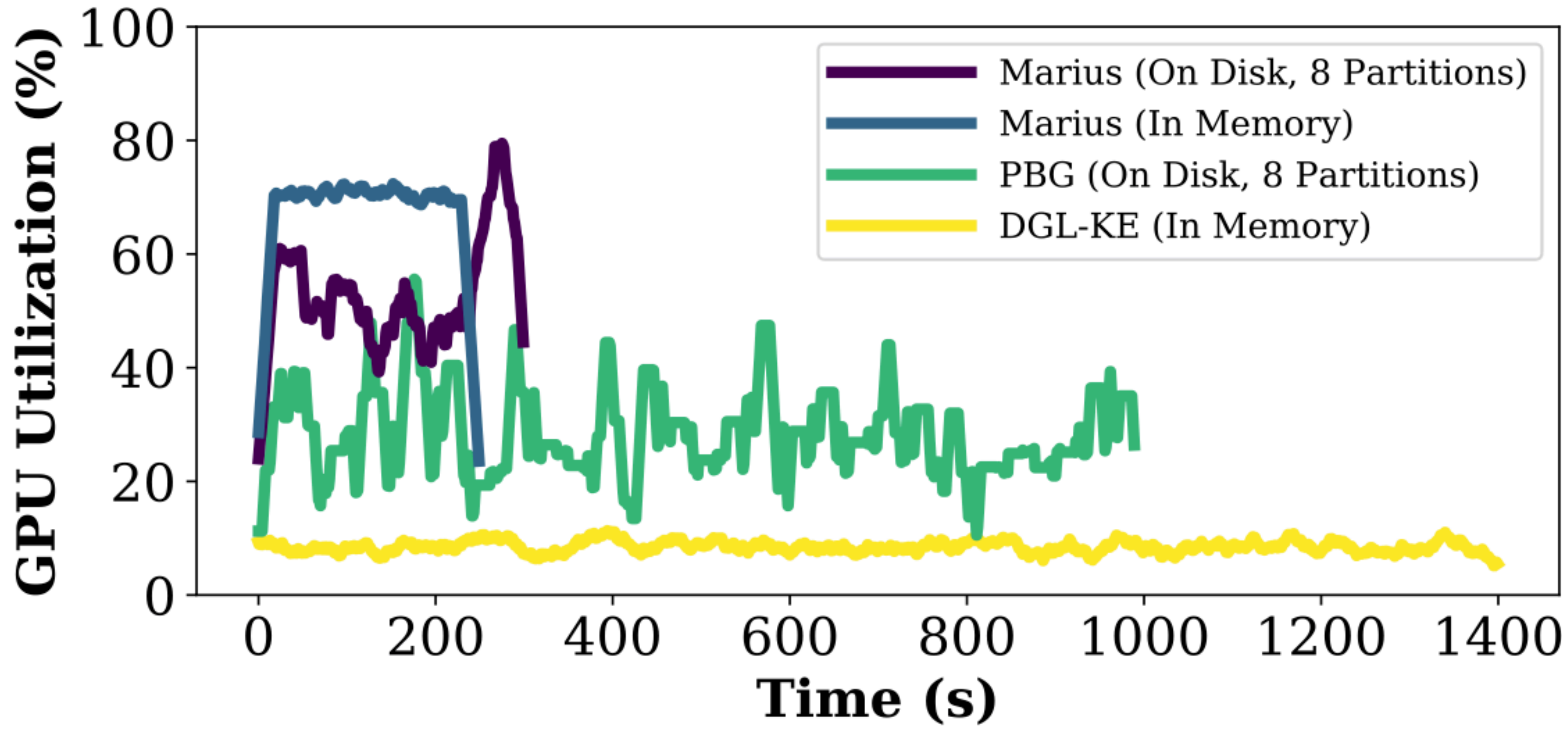
BETA ordering enables high GPU utilization

<u>Method</u>

- Use pipelining and async IO hide data movement
- Utilize the full memory hierarchy with a partition buffer
- **Minimize IO with Buffer-aware Edge Traversal Algorithm (BETA)**

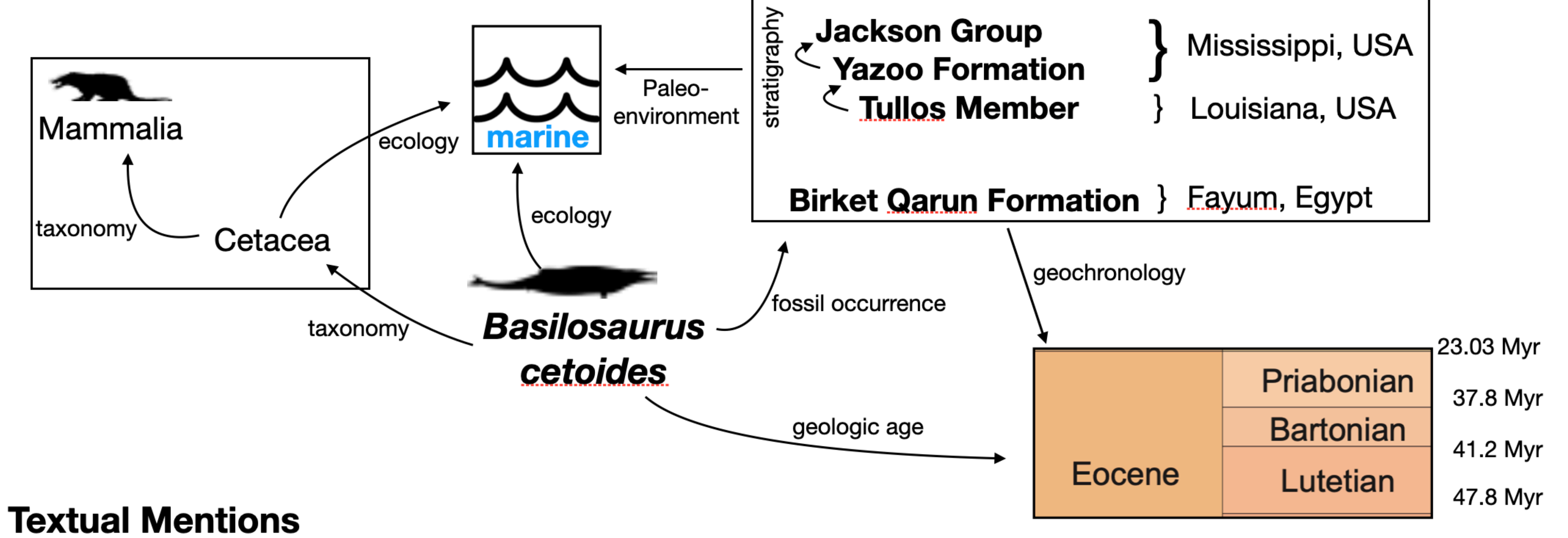
<u>Results</u>

- 10x reduction in runtime vs. DGL-KE on Twitter
- 3.7x runtime reduction vs. PBG on Freebase86m
- 2x higher utilization than PBG, 6-8x higher utilization than DGL-KE



Use-case: Construction of Scientific Knowledge Graphs

Simplified Knowledge Base



Textual Mentions

Fossils from an extinct toothed (**Archaeocete**) whale, ***Basilosaurus cetoides***, were found in a surface exposure of the **Pachuta Marl Member** of the **late Eocene Yazoo Clay** near the Matherville community in **Wayne County, Mississippi**.

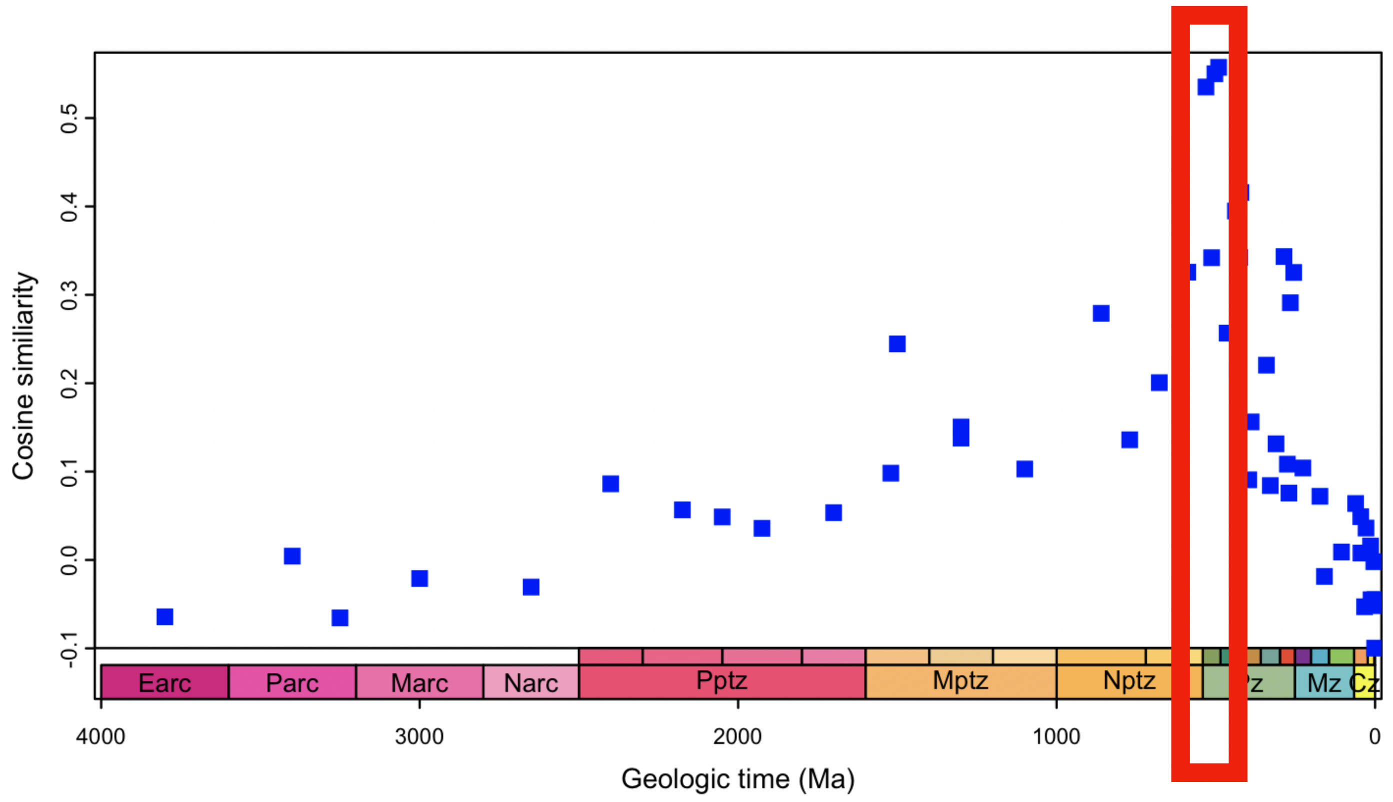
The **Yazoo Clay Formation** makes up the upper half of the **Jackson Group** in the central **Gulf Coastal Plain**, representing deposition during the TAGC4.3 **marine** transgression.

Analogical Reasoning Example

Basilosaurus —geochronology→ **Eocene**

Wheeler Shale Trilobites —geochronology→ **?**

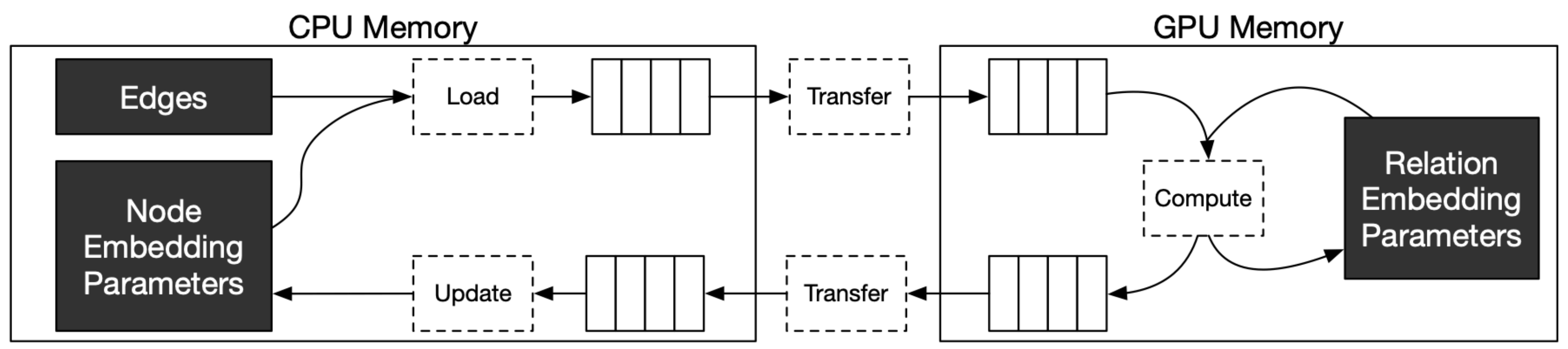
Elrathia kingii, famed trilobite of the Wheeler Shale.

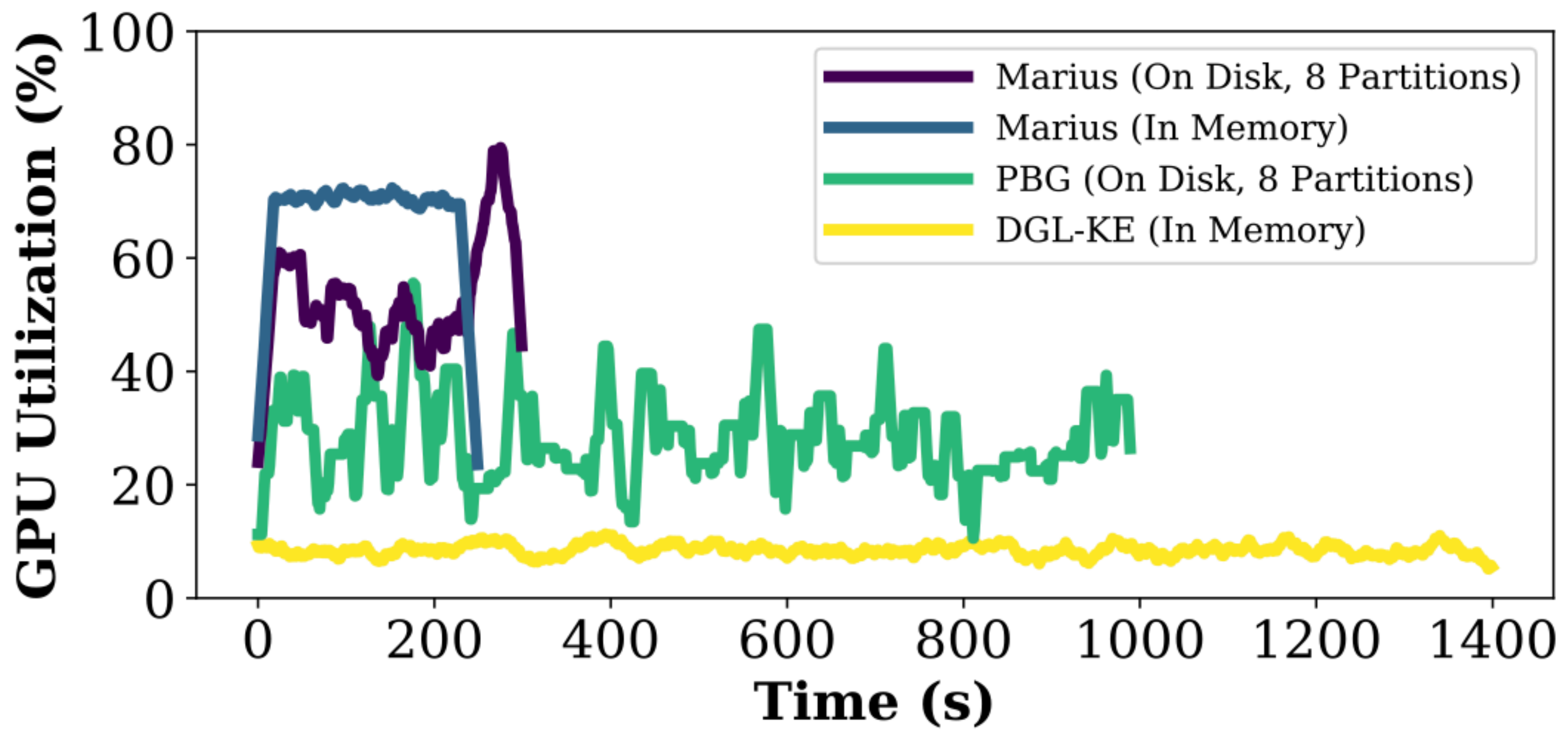


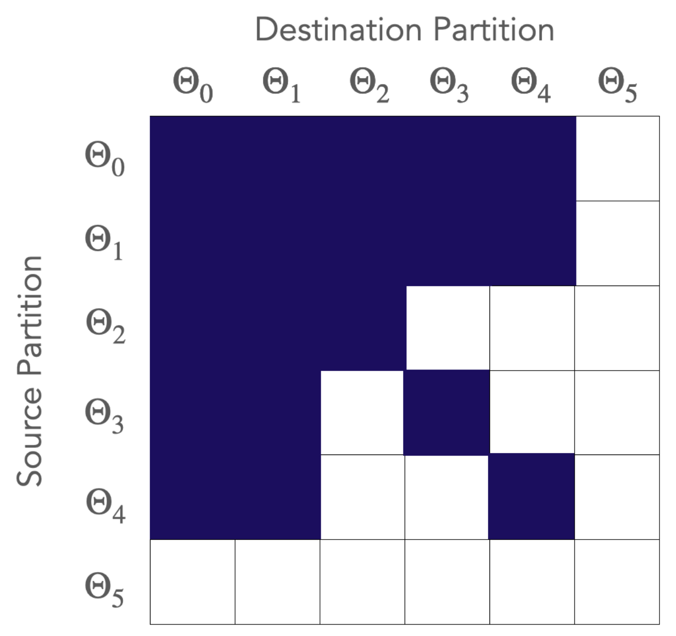
Joint embeddings of text and existing knowledge graphs to enable analogical reasoning and knowledge completion in any domain

Marius: Scalable graph learning

Learning Massive Graph Embeddings on a Single Machine, OSDI'2021







Can never process more than 2c - 1 edge buckets per swap

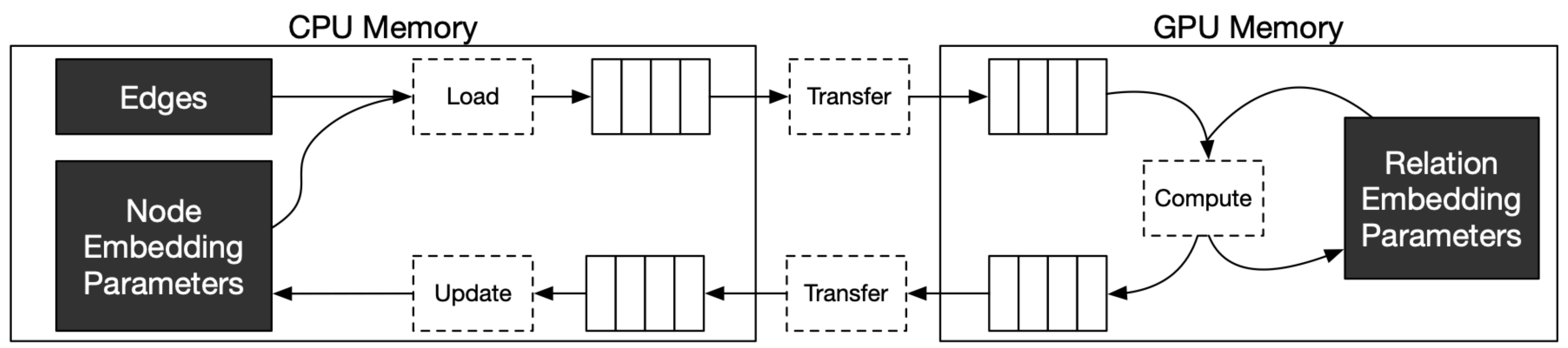
$$\lceil \frac{p^2 - c^2}{2c - 1} \rceil = \lceil \frac{6^2 - 3^2}{2 * 3 - 1} \rceil = 6$$

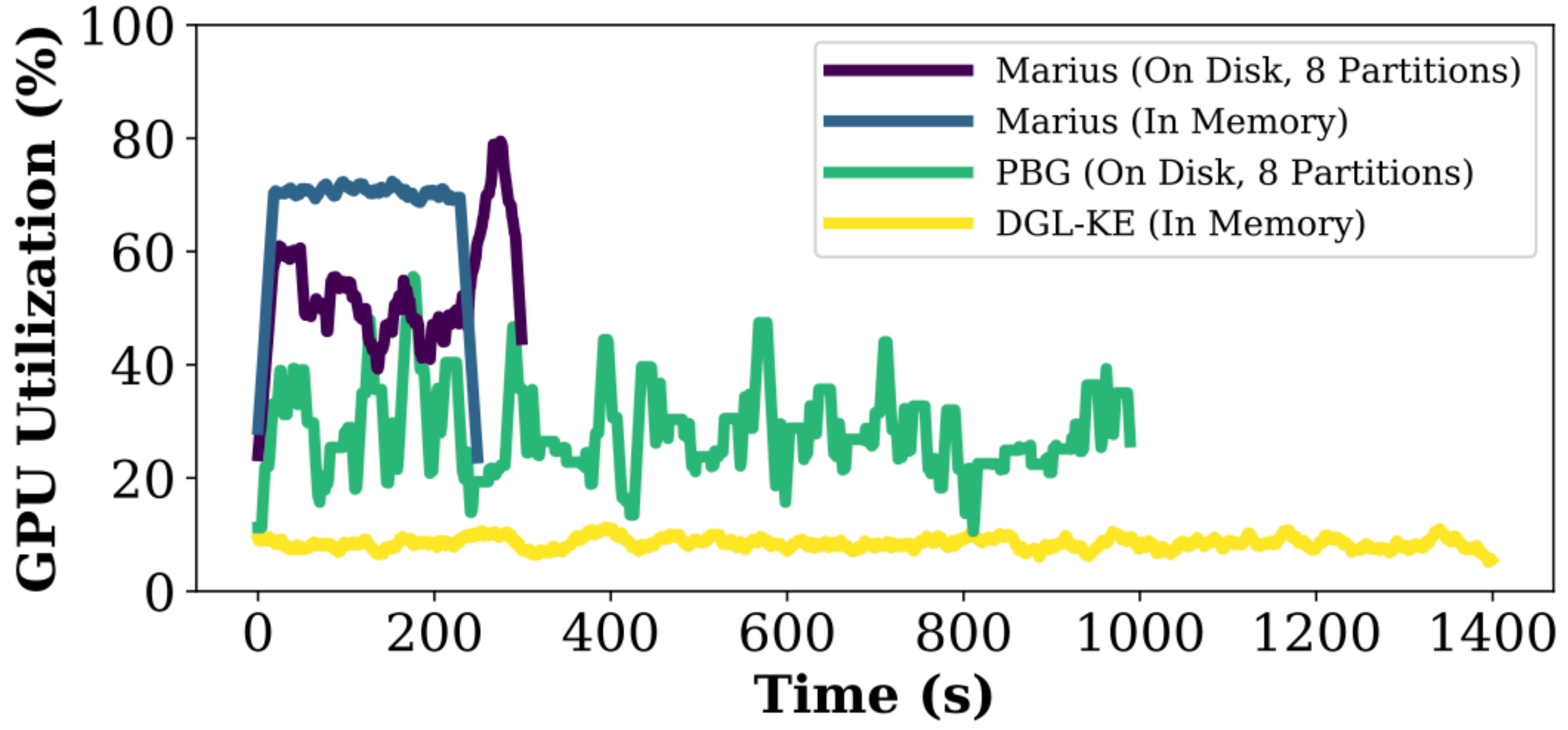
Marius achieves graph learning over billion-edge graphs
10x faster and 5x cheaper than competing solutions

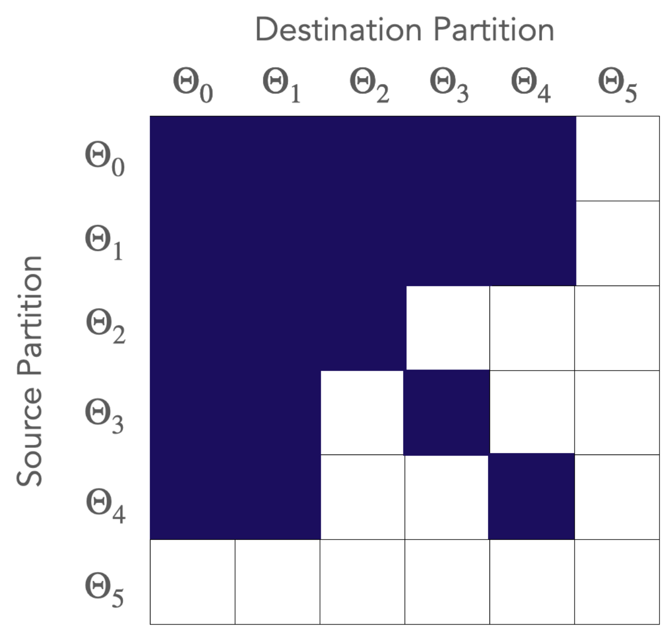
Find more at: marius-project.org

Marius: Scalable graph learning

Learning Massive Graph Embeddings on a Single Machine, OSDI'2021







Can never process more than 2c - 1 edge buckets per swap

$$\left\lceil \frac{p^2 - c^2}{2c - 1} \right\rceil = \left\lceil \frac{6^2 - 3^2}{2 * 3 - 1} \right\rceil = 6$$

Marius achieves graph learning over billion-edge graphs
10x faster and 5x cheaper than competing solutions

Find more at: marius-project.org

Thank you!
@thodrek

The case of exploiting the full memory stack

d	Size	Partitions	MRR	Runtime (Epoch)
20	13.6 GB	-	.698	4m
50	34.4 GB	-	.722	4.8m
100	68.8 GB	32	.726	12.1m
400	275.2 GB	32	.731	92.4m
800	550.4 GB	64	.731	396m

*MRR: mean reciprocal rank (higher is better)

Per-epoch runtime and reconstruction-accuracy as we increase the embedding size for Freebase (86M nodes and 338M edges)

Higher-dimensional embeddings can lead to higher accuracy