An opinionated guide to node embeddings

Anton Tsitsulin

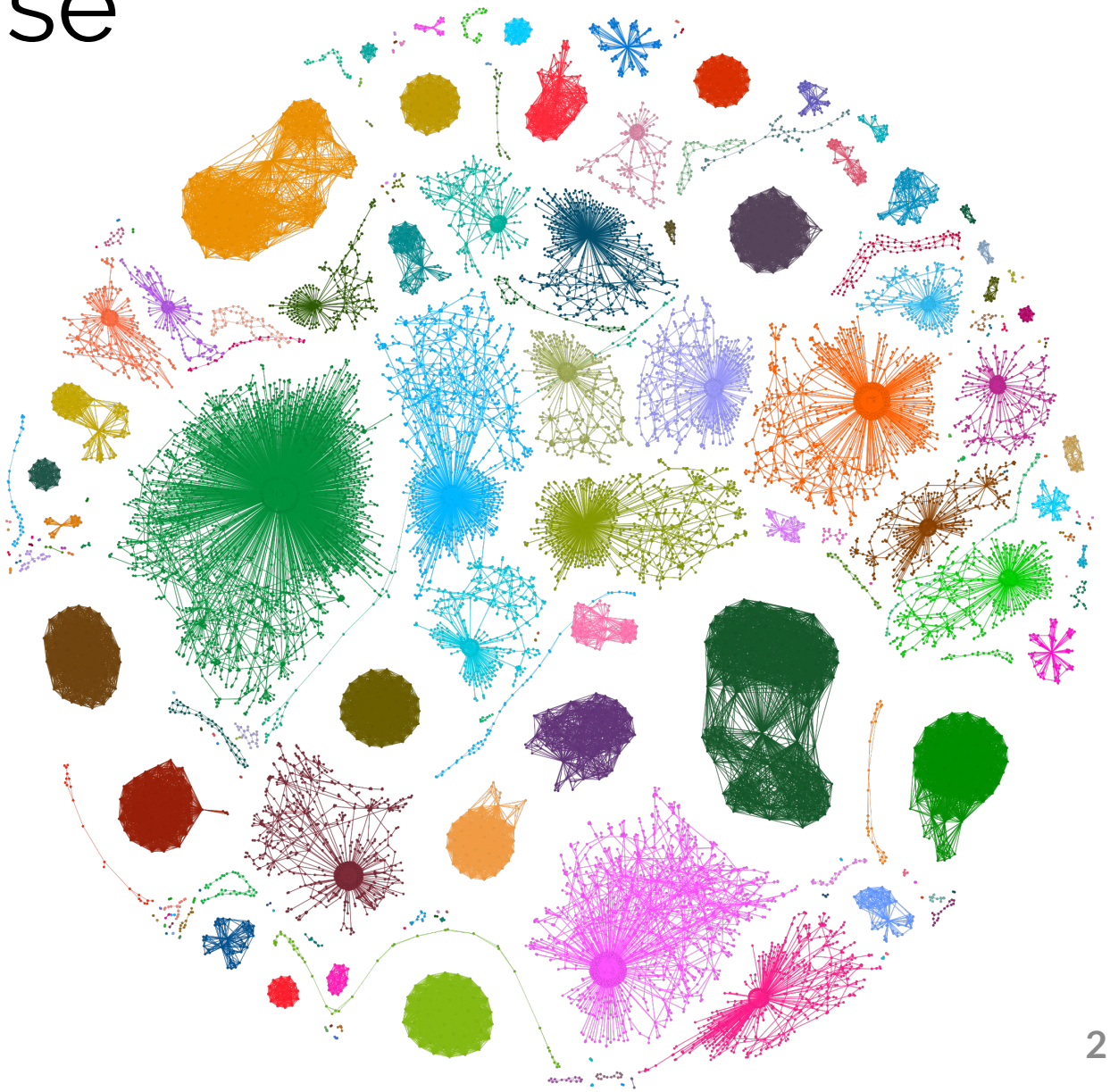
Ph.D. student @ University of Bonn

Student Researcher @ Google

Graph world is diverse

Different **domains**:

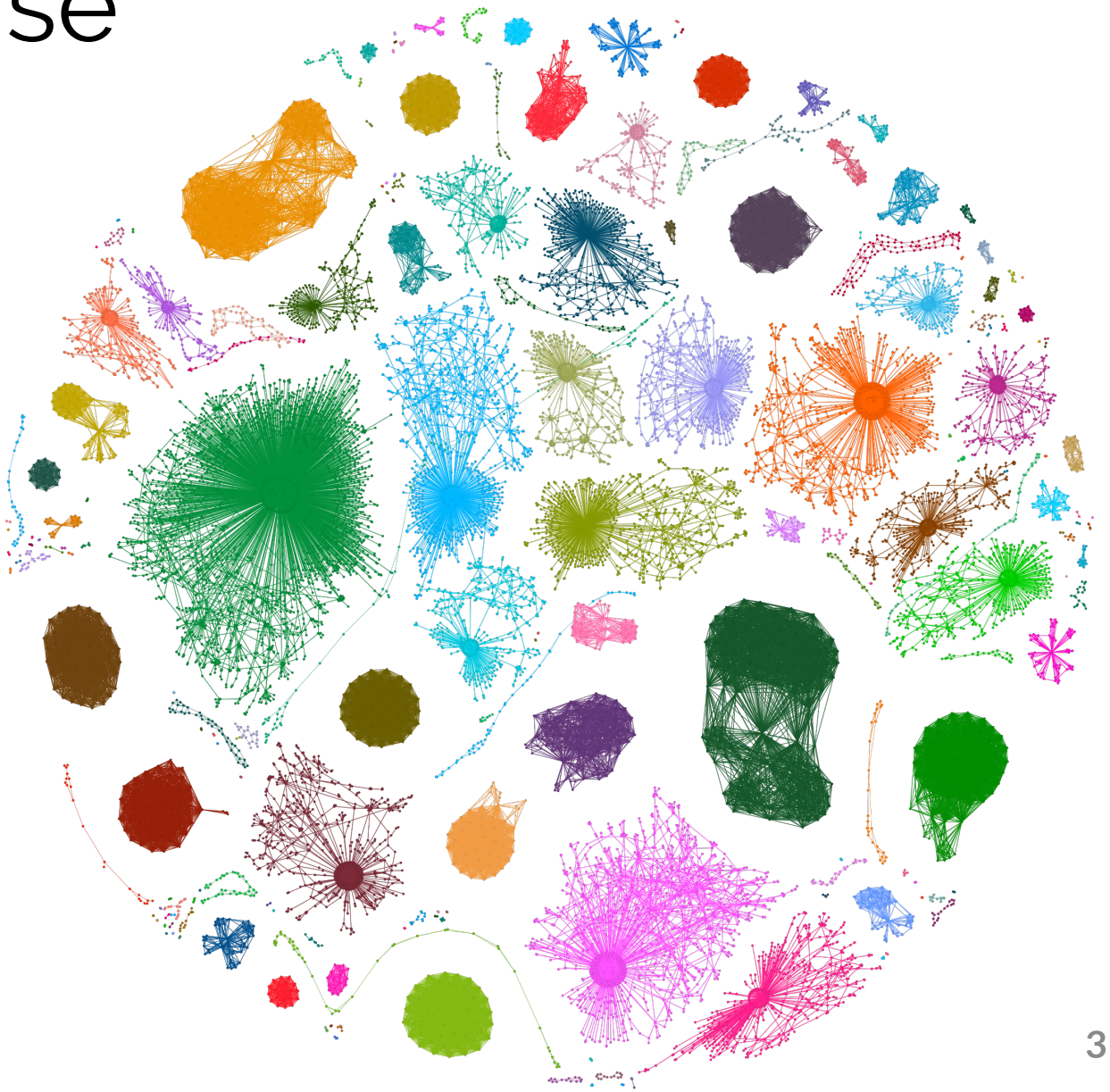
- Information
- Social
- Biological
- Transportation
- …



Graph world is diverse

Different **graph types**:

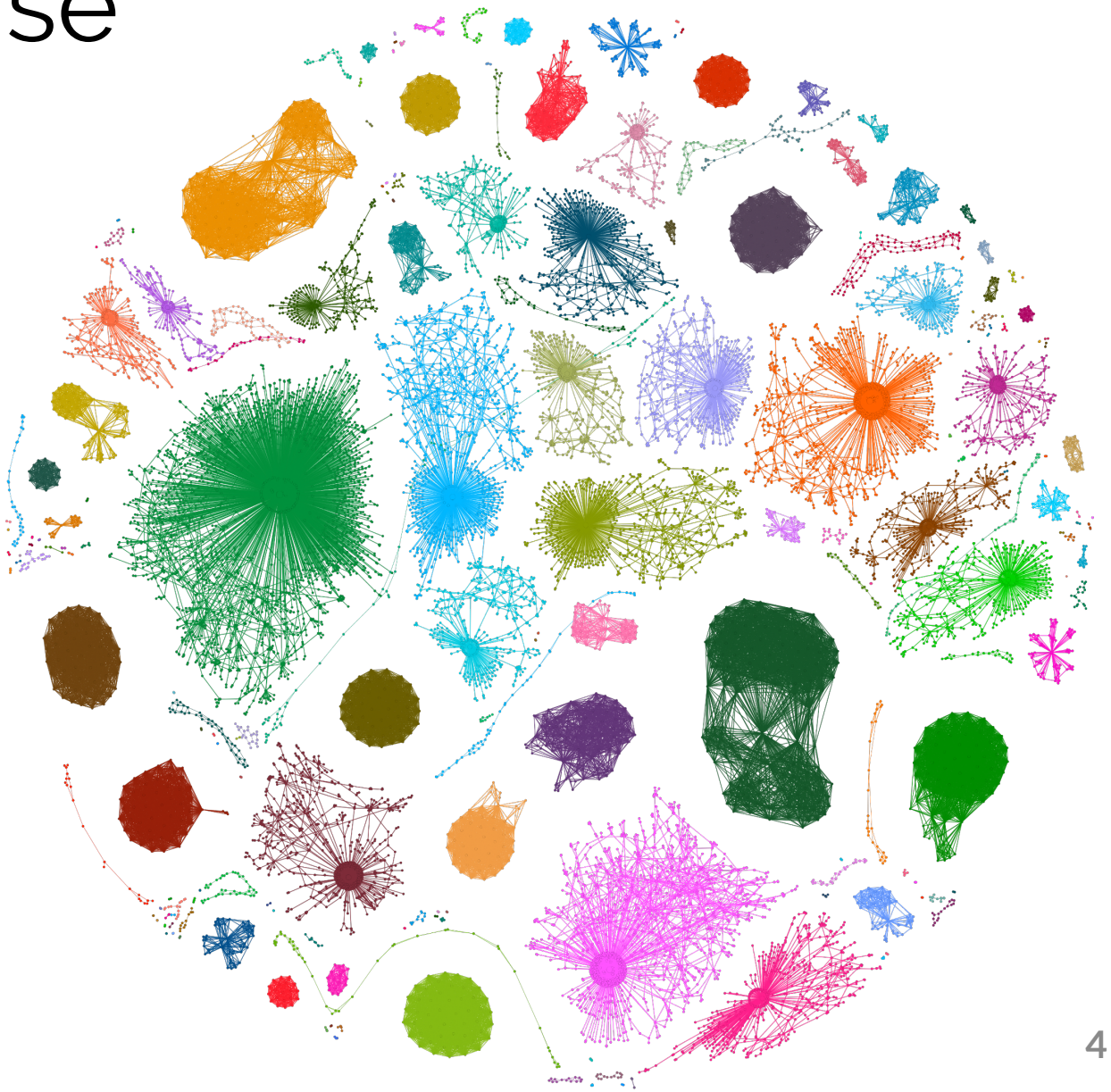
- (Un)directed
- (Un)weighted
- Temporal
- Heterogeneous
- …



Graph world is diverse

Different **modalities**:

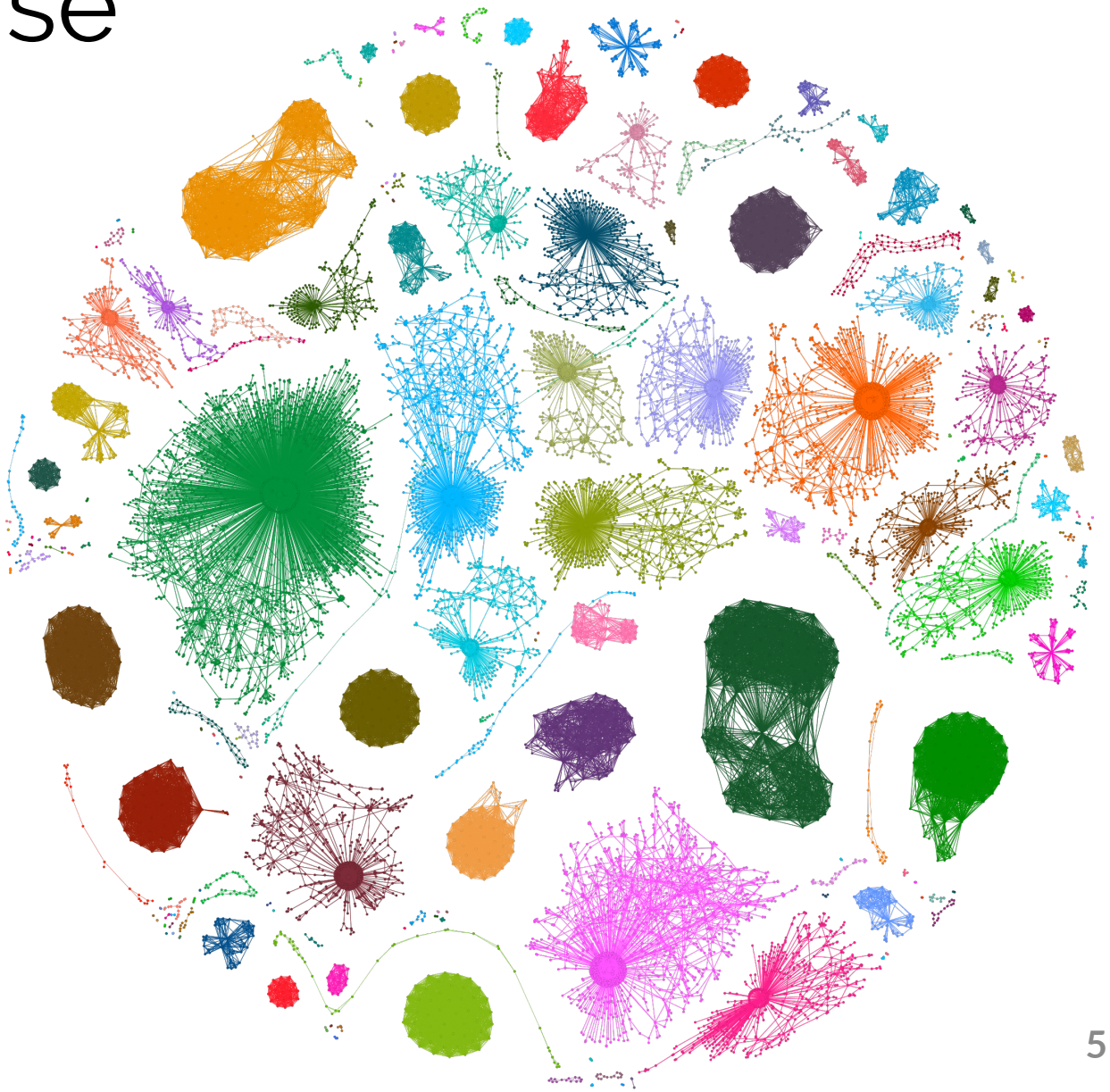
- Nodes
- Edges
- Motifs
- Subgraphs
- Whole graphs
- …



Graph world is diverse

Different **tasks**:

- Classification
- Clustering
- Anomaly detection
- …



Graph world is diverse

Domains	Graph Types	Modalities	Tasks
Information Social Biological Transportation	(Un)directed (Un)weighted Temporal Heterogeneous	Nodes Edges Subgraphs Whole graphs	Classification Clustering Anomaly detection

Graph world is diverse

Domains

Information
Social
Biological
Transportation

Graph Types

(Un)directed
(Un)weighted
Temporal
Heterogeneous

Modalities

Nodes
Edges
Subgraphs
Whole graphs

Tasks

Classification
Clustering
Anomaly detection

↓

Embeddings

Graph world is diverse

Domains

Information
Social
Biological
Transportation

Graph Types

(Un)directed
(Un)weighted
Temporal
Heterogeneous

Modalities

Nodes
Edges
Subgraphs
Whole graphs

Tasks

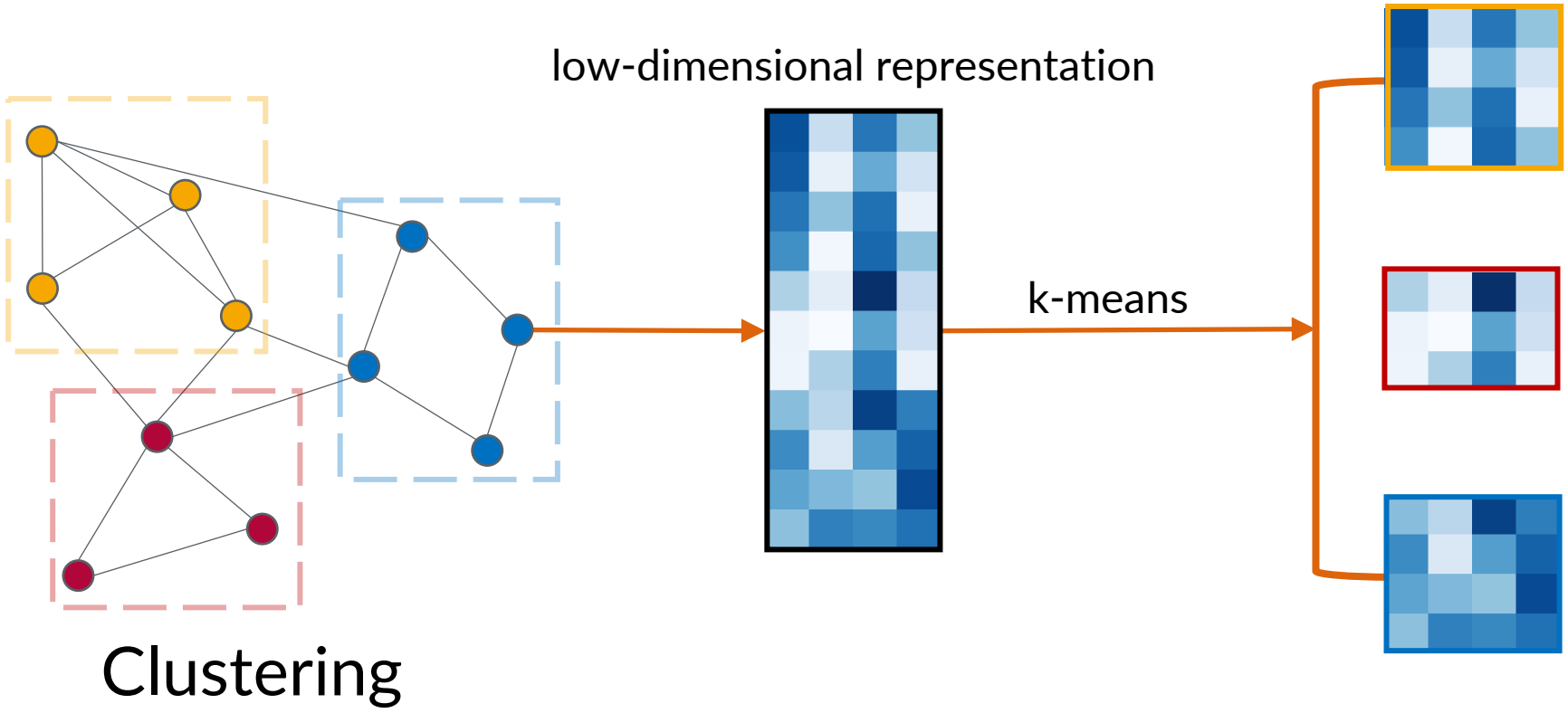
Classification
Clustering
Anomaly detection

↓

Embeddings

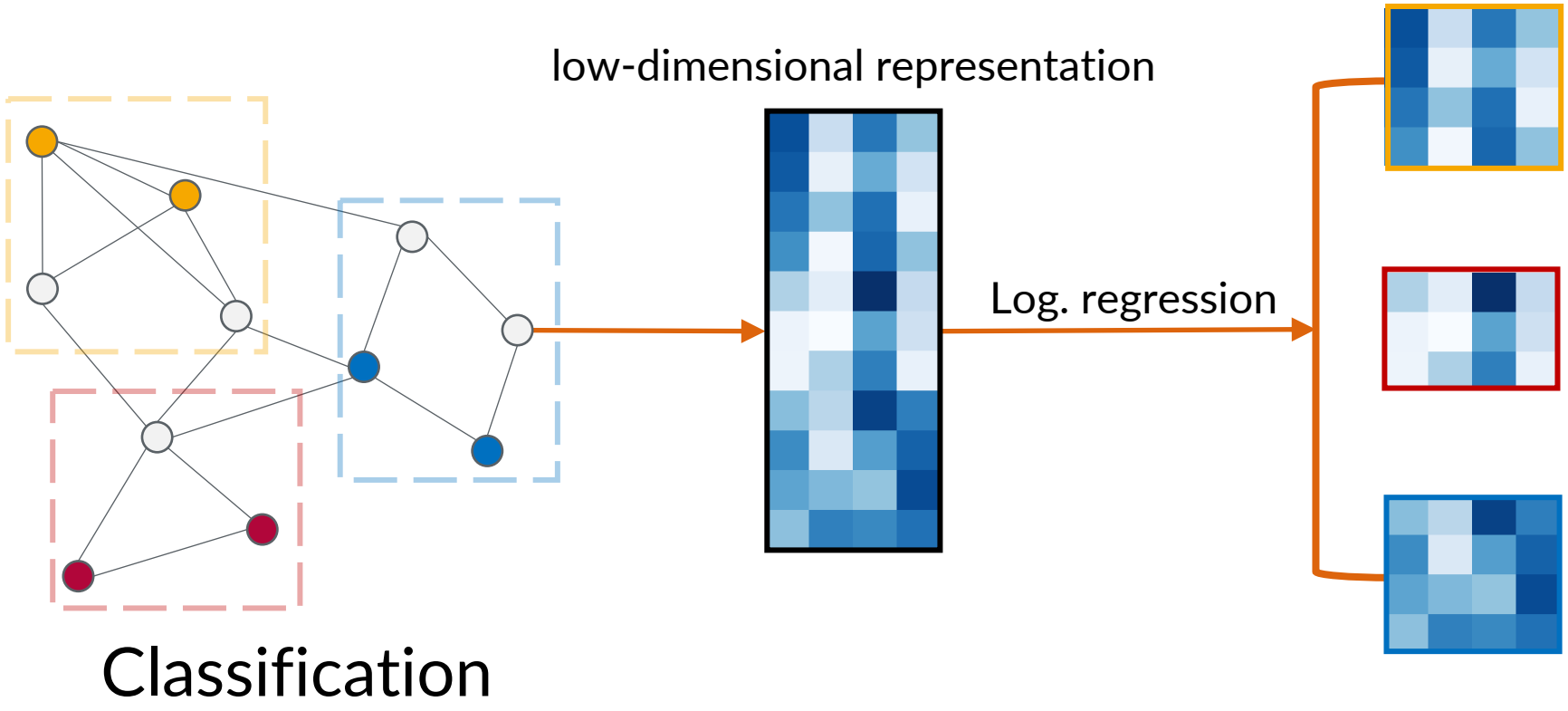
Why representations?

We have fast & good algorithms for mining vector data…

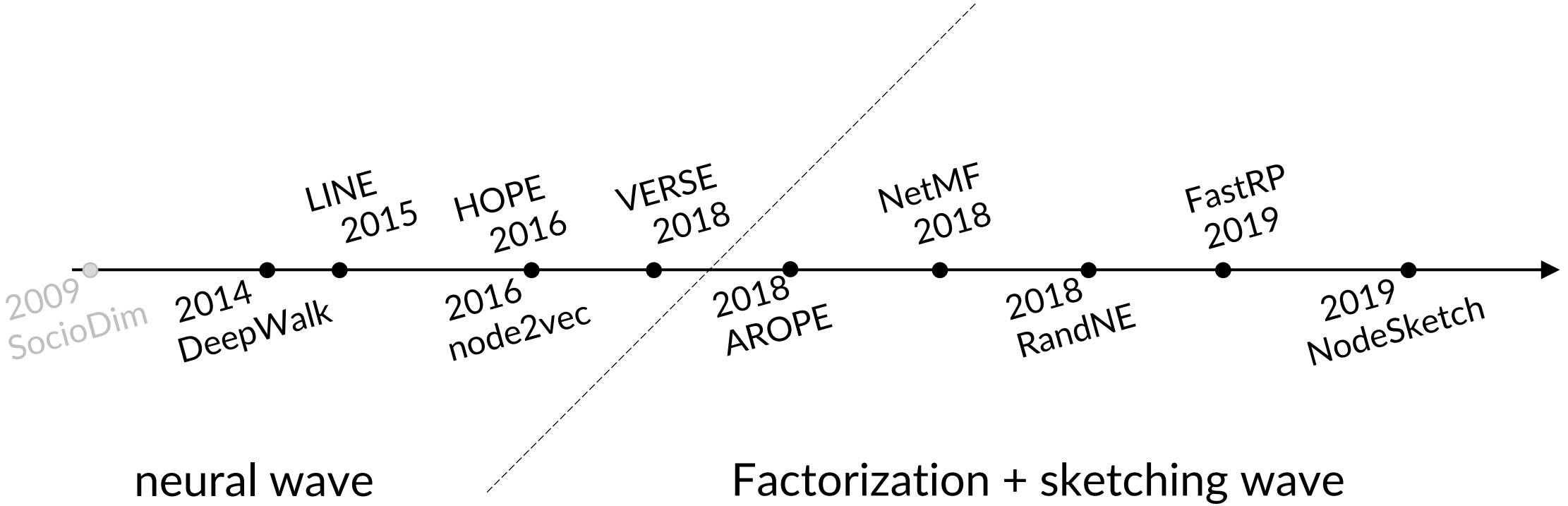


Why representations?

We have fast & good algorithms for mining vector data…



A brief history of node embeddings



Algorithm

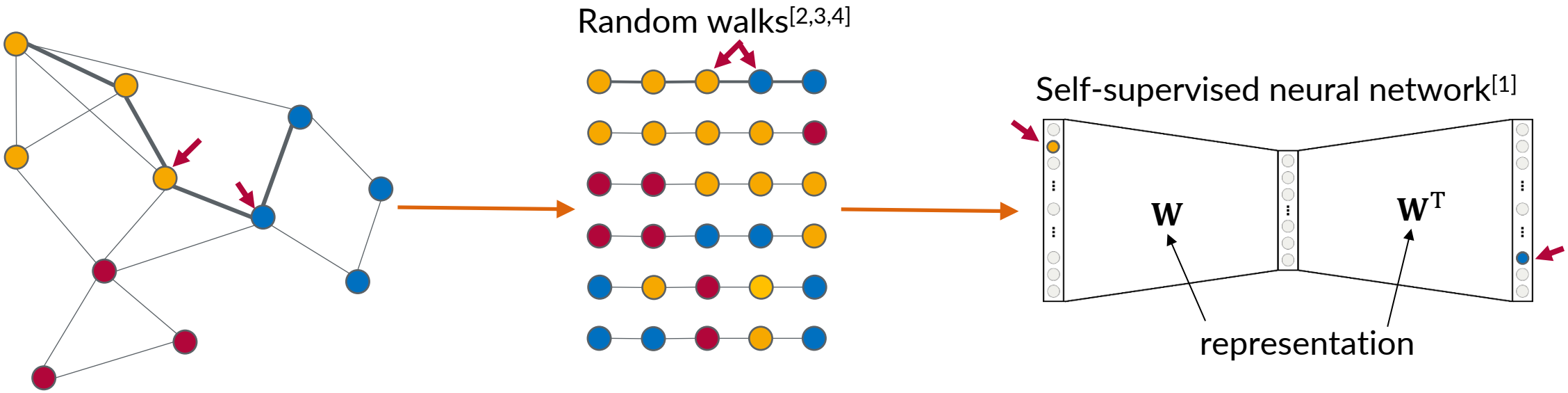
1-sentence summary of the contribution

Overview and inner workings of the algorithm

Neural node embeddings

Anatomy of a neural embedding

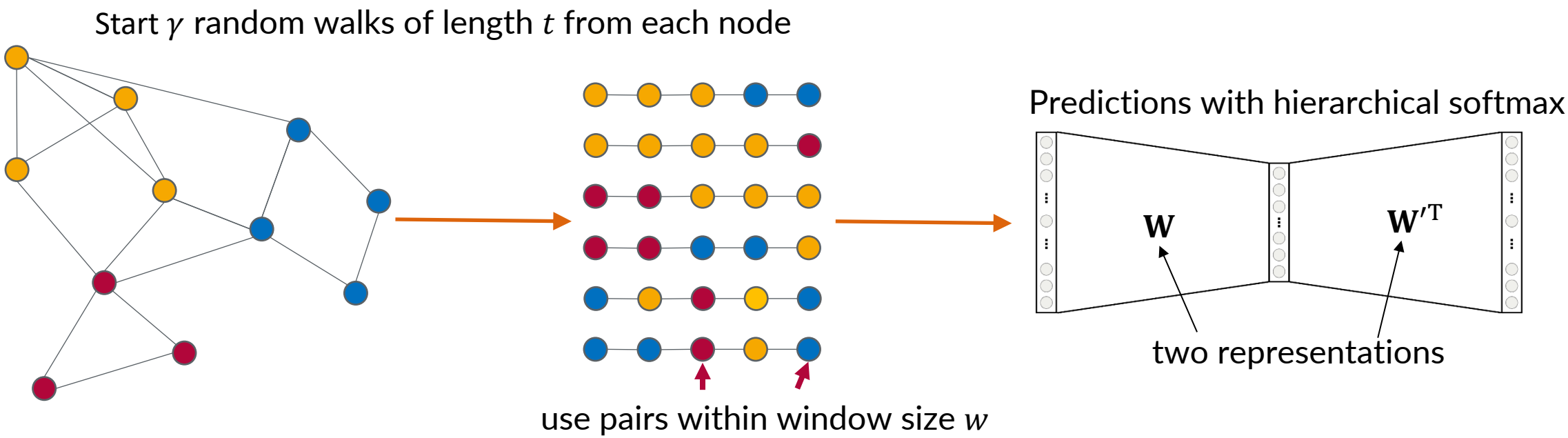
Nodes in random walks ≈ words in sentences → use word2vec



[1] Efficient Estimation of Word Representations in Vector Space. Mikolov et al., NIPS 2013
[2] DeepWalk: Online Learning of Social Representations. Perozzi et al., KDD 2014
[3] node2vec: Scalable Feature Learning for Networks. Grover & Leskovec, KDD 2016
[4] VERSE: Versatile Graph Embeddings from Similarity Measures. Tsitsulin et al., WWW 2018

DeepWalk: algorithm overview

"Nodes in random walks ≈ words in sentences → use word2vec"



DeepWalk: asymptotics and practice

In practice, $\gamma = 80$, $t = 80$, $w = 10$, meaning $80 * 80 * n$ of "text"

NB: never change w

If you lower w, increase γ and t

Parameter meaning is not trivial :(

DeepWalk: asymptotics and practice

Python implementation generates all walks and calls word2vec

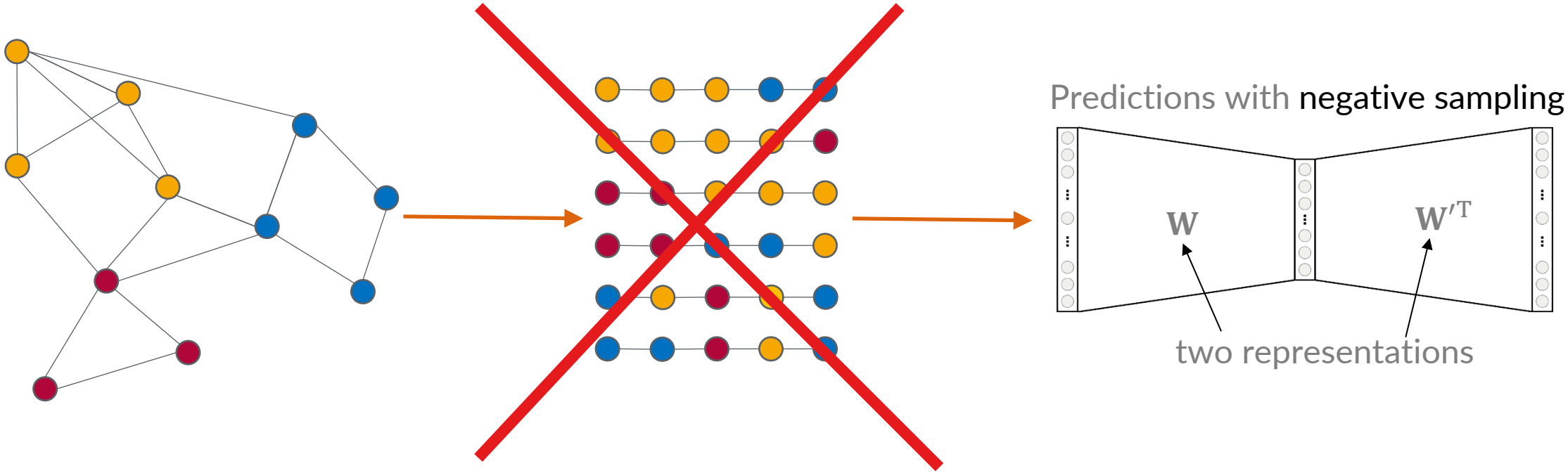
I wrote a C++ implementation that does not store extra walks

Optimization is still $O(d * \log n)$ at each step :(

Practical limitations: ~3M nodes

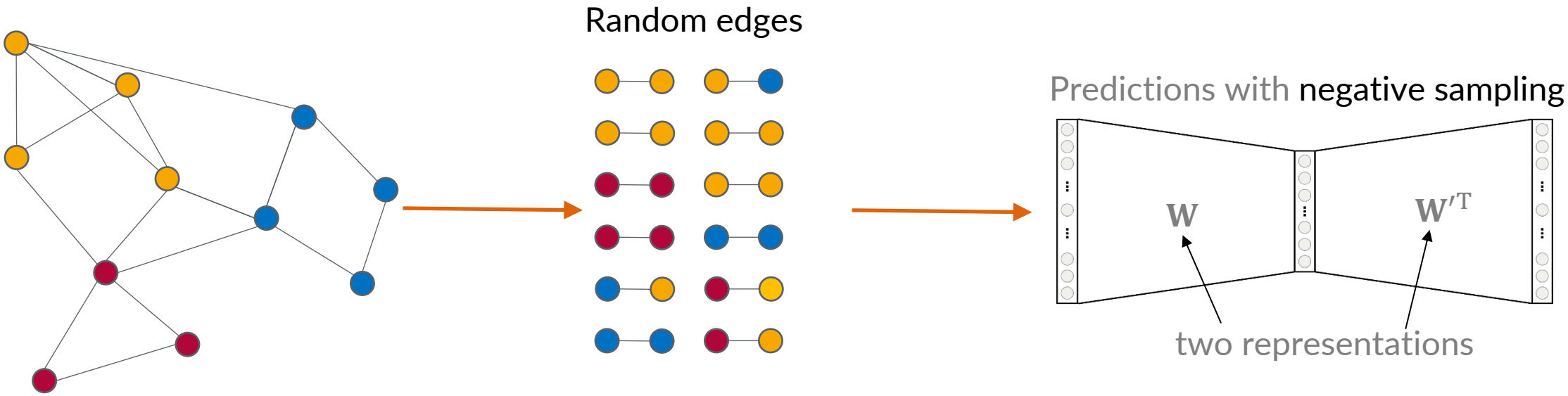
LINE: algorithm overview

"Why use random walks when edges do the trick"



LINE: algorithm overview

"Why use random walks when edges do the trick"



LINE: Large-scale information network embedding. Tang et al., WWW 2015

LINE: asymptotics and practice

Simple & fast algorithm, **not great** on downstream tasks :(

NB: Set the total # of samples T proportional to # of edges

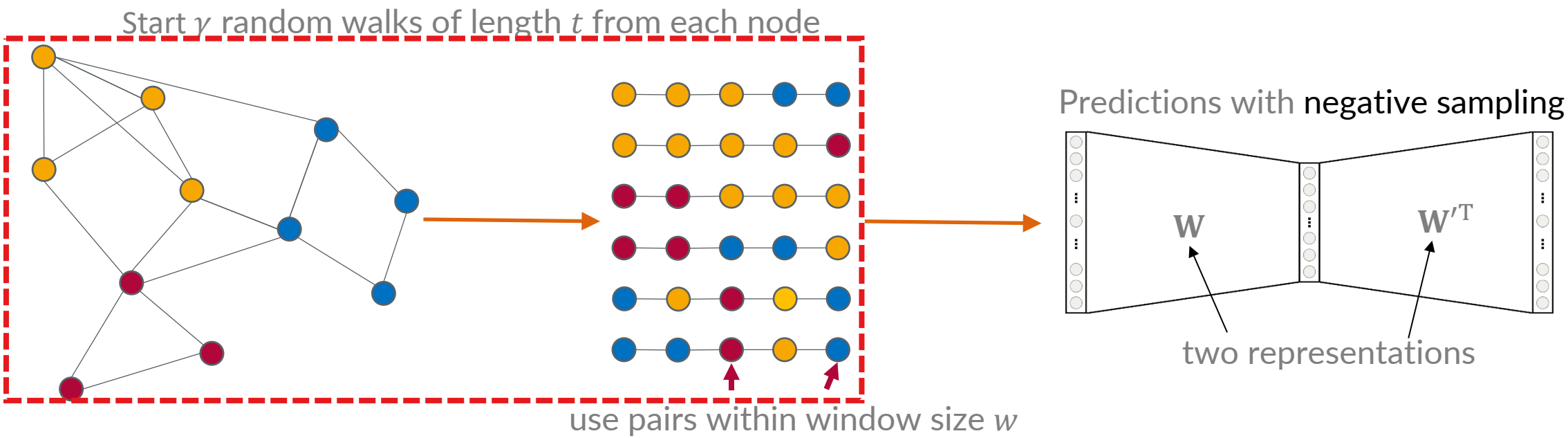
Optimization is $O(dn)$:)

Authors' C++ implementation works well

Practical limitations: ~10M nodes

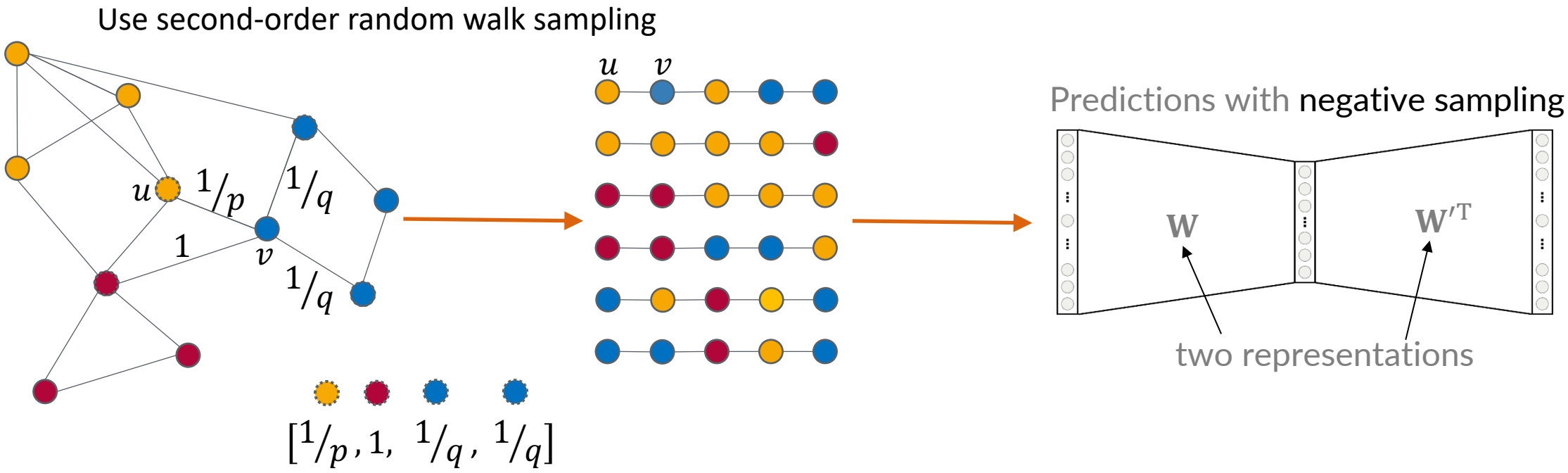
Node2vec: algorithm overview

"Let's add two more parameters (p, q) to DeepWalk"



Node2vec: algorithm overview

"Let's add two more parameters (p, q) to DeepWalk"



Node2vec: scalable feature learning for networks. Grover & Leskovec, KDD 2016

Node2vec: myth 1

Myth: parameters (p, q) are related to BFS and DFS

Reality: parameters (p, q) are related to triangles $\approx$ clusters

Low $q \rightarrow$ explore intra-cluster information

High $q \rightarrow$ explore inter-cluster information

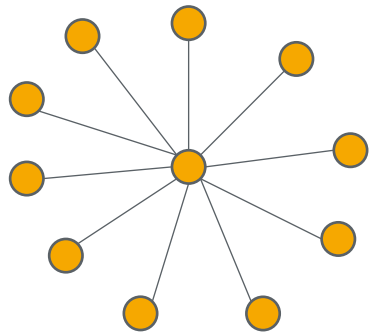
Node2vec: myth 2

Myth: node2vec is a scalable algorithm

Reality: second-order random walks are worst-case $O(n^2)$

Worst case is a star graph

(or any graph with very high-degree nodes)



Node2vec: asymptotics and practice

NB: comparisons in the paper are misleading ($\gamma = 10$ for all methods)

In the paper, $\gamma = 10$, $t = 80$, $w = 10$, hyperparameter search for (p, q)

Setting $\gamma = 10$ gives worse results, please use $\gamma = 80$

Tuning (p, q) is not beneficial on most graphs

Node2vec: asymptotics and practice

Python implementation generates all walks and calls word2vec

I wrote a C++ implementation that does not store extra walks

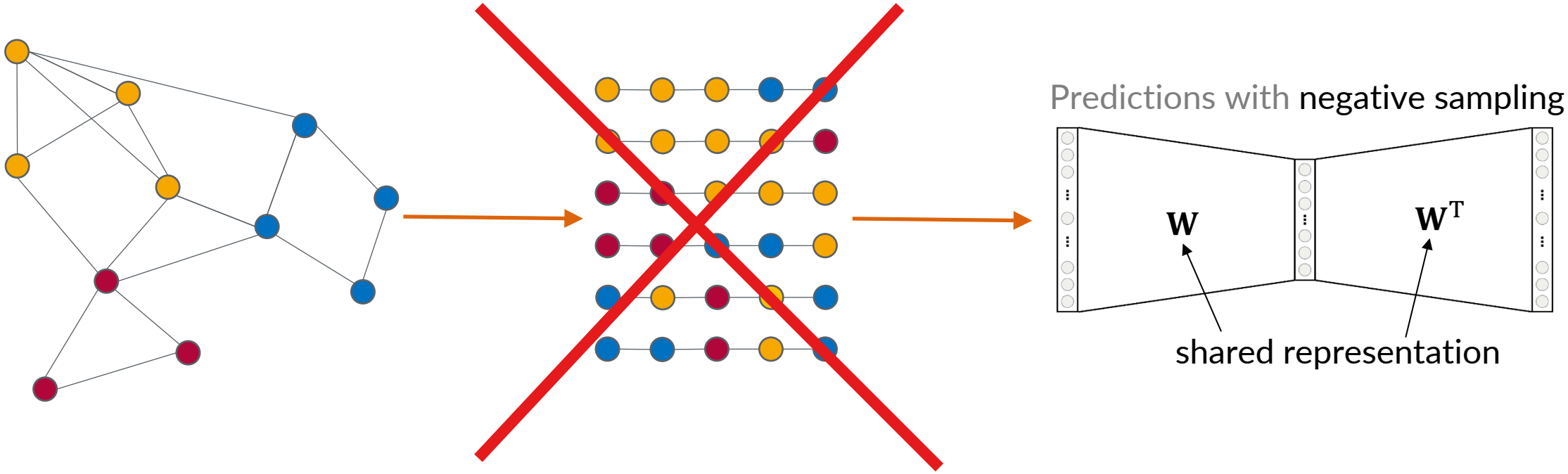
Preprocessing can be $O(n^2)$:(

Optimization is $O(d)$:)

Practical limitations: ~500k nodes if you are lucky, if not, ~50k

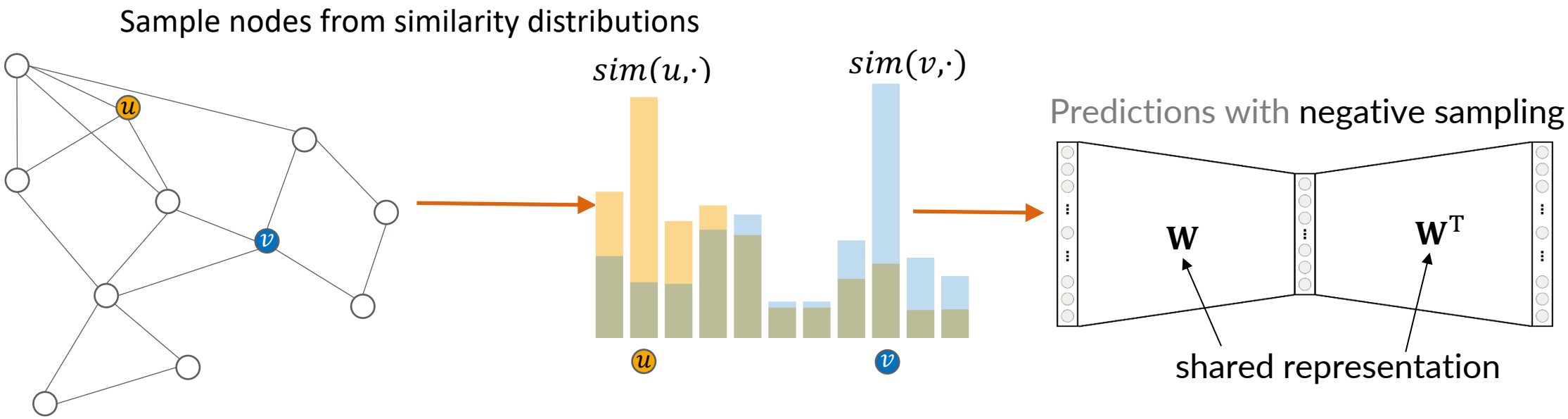
VERSE: algorithm overview

"Random walks define a similarity distribution"



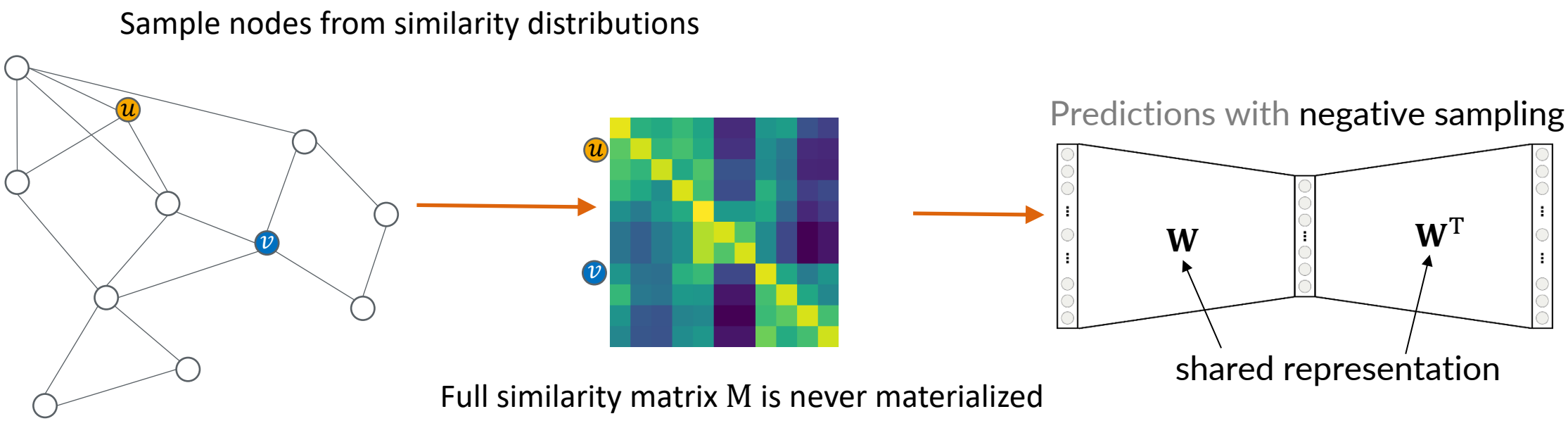
VERSE: algorithm overview

"Random walks define a similarity distribution"



VERSE: algorithm overview

"Random walks define a similarity distribution"



VERSE: interpretation of DeepWalk

DeepWalk random walks ~= Personalized PageRank

PPR parameter $\alpha = \frac{w-2}{w+1}$ for DeepWalk's w

We can now measure the quality of embedding directly :)

1 parameter instead of 3 or 5

VERSE: asymptotics and practice

Simple & fast algorithm, good on symmetric link prediction :)

NB: if edges' information is asymmetric, try using two matrices $\mathbf{W}, \mathbf{W'^T}$

Optimization is $O(d)$:)

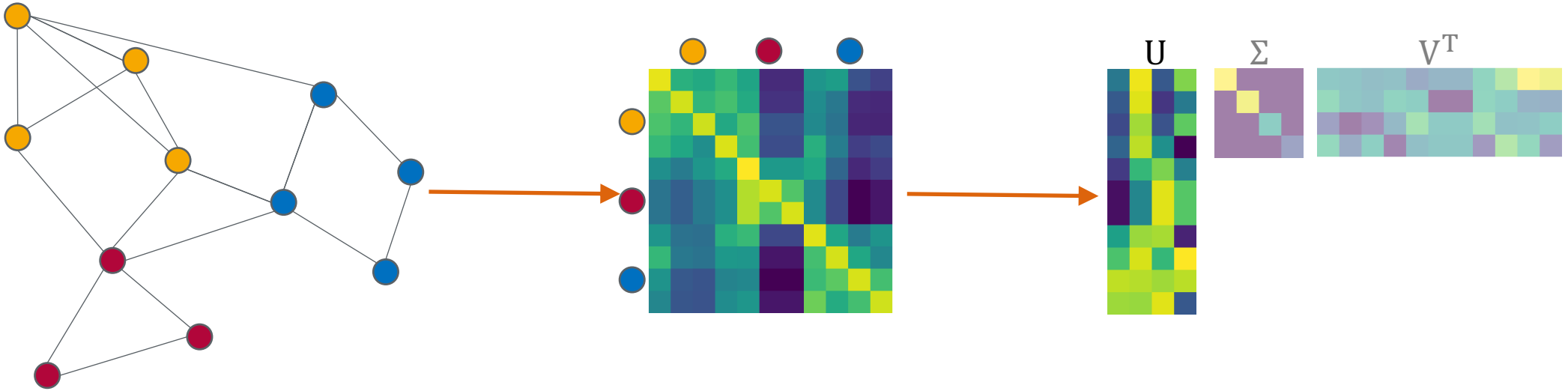
Authors' C++ implementation works well

Practical limitations: ~10M nodes

Factorization embeddings

Anatomy of a factorization embedding

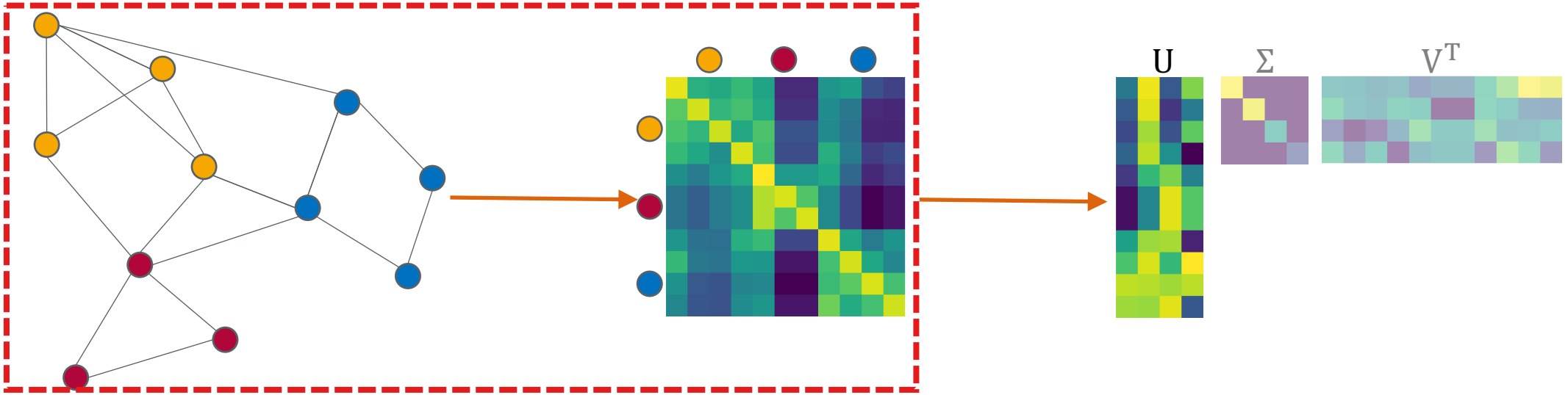
Construct a similarity matrix → do SVD



[1] Asymmetric transitivity preserving graph embedding. Ou et al., KDD 2016
[2] Arbitrary-order proximity preserved network embedding. Zhang et al., KDD 2018
[3] Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec. Qiu et al., WSDM 2018

HOPE: algorithm overview

Do SVD on an implicit similarity matrix with sparse updates

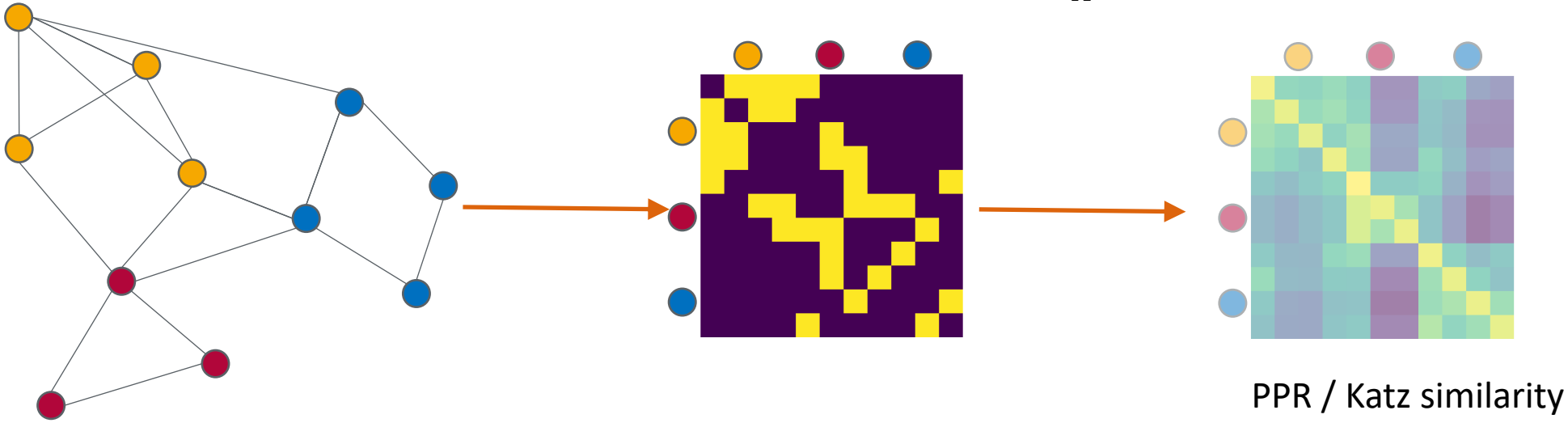


Asymmetric transitivity preserving graph embedding. Ou et al., KDD 2016

HOPE: algorithm overview

Do SVD on an implicit similarity matrix with sparse updates

JDGSVD decomposes dense similarity matrices of the form $M_A^{-1}M_B$



PPR / Katz similarity

HOPE: asymptotics and practice

Does not work well for classification/link prediction :(

Horrible MATLAB code :(

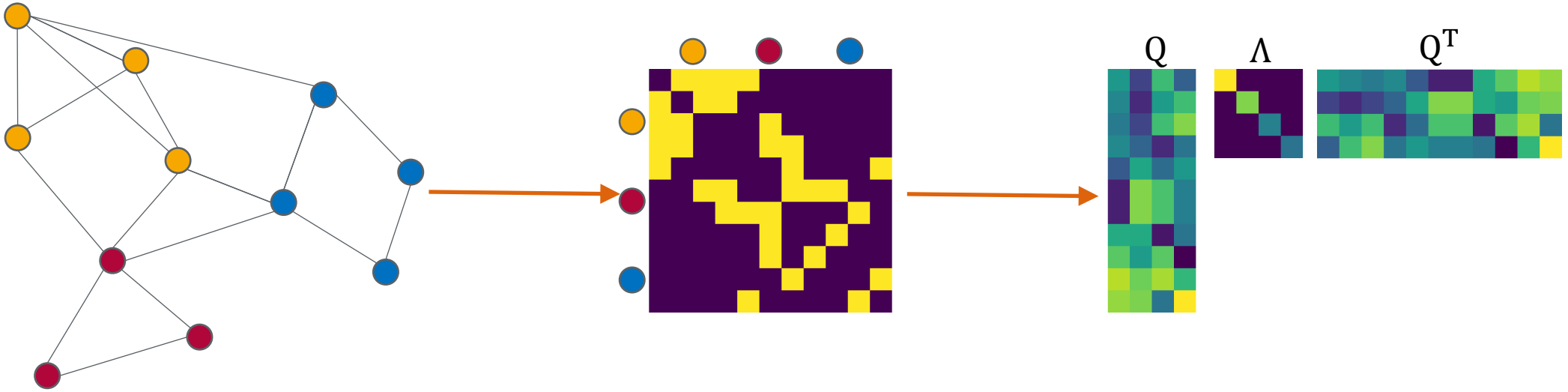
Overall complexity is ~ $O(d * m * L)$:)

Practical limitations: ~5M nodes + MATLAB license

AROPE: algorithm overview

Do eigen decomposition on the adjacency matrix, scale the λ's

Idea: $f(A) \approx Qf(\Lambda)Q^T$, works with Katz similarity



AROPE: asymptotics and practice

Does not work well for classification/link prediction :(

Easy to implement :)

Overall complexity is ~ $O(d * m + d^2 n)$:)

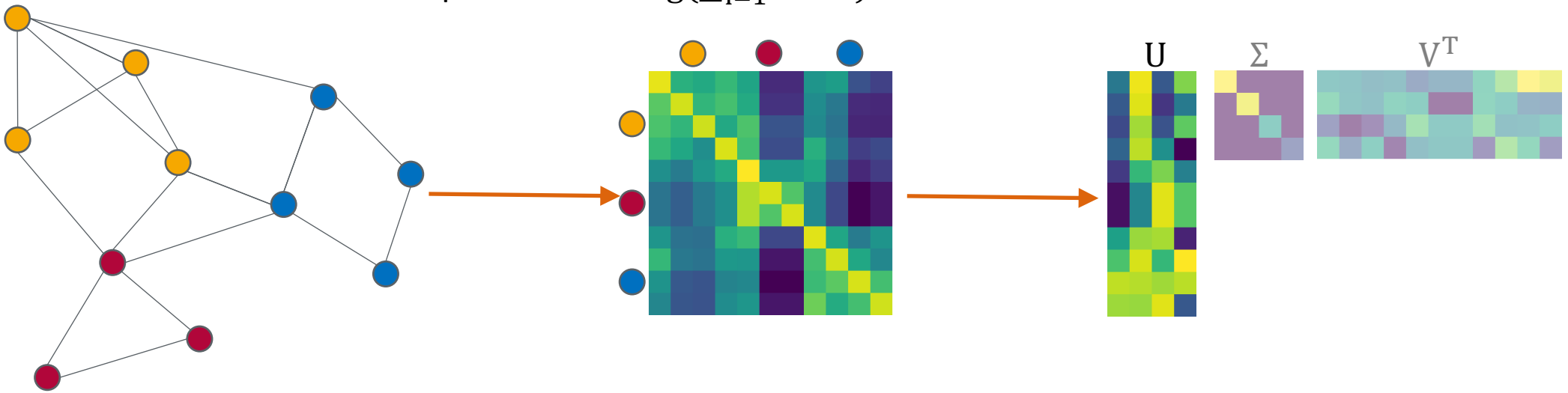
Actual speed depends on the eigensolver

Practical limitations: ~5M nodes

NetMF: algorithm overview

Let's decompose DeepWalk's similarity matrix

DeepWalk's $M \approx \log(\sum_{i=1}^{10} D^{-1}A)$



NetMF: asymptotics and practice

Matrix $\mathbf{M}$ is dense, limiting scalability

A bridge between factorization and neural methods

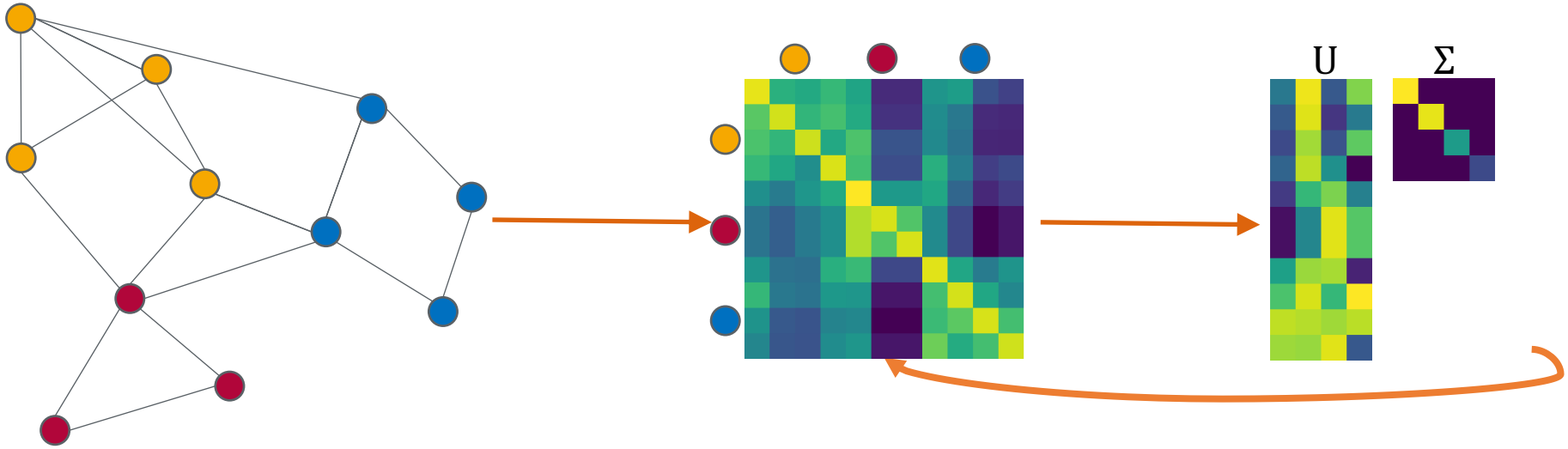
Overall complexity is ~ $O(n^3)$ ٩(ර益ර)۶

Practical limitations: ~10k nodes

Sketch-based embeddings

Anatomy of a sketch-based embedding

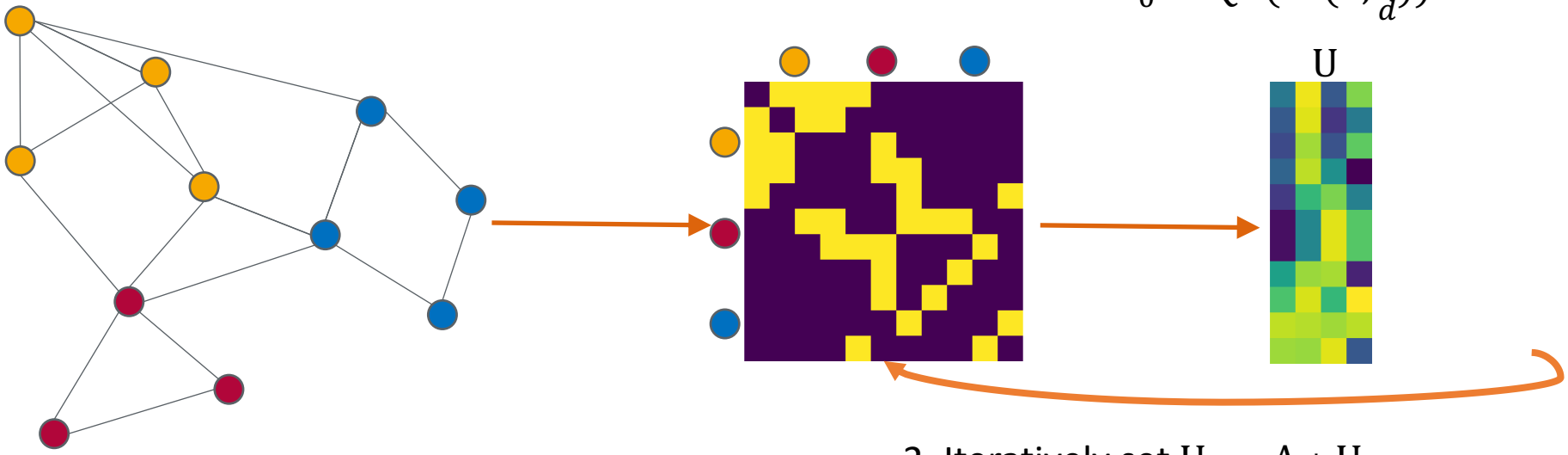
Iteratively approximate a higher-order embedding



[1] Billion-scale Network Embedding with Iterative Random Projection. Zhang et al., ICDM 2018
[2] Fast and Accurate Network Embeddings via Very Sparse Random Projection. Chen et al., CIKM 2018
[3] NodeSketch: Highly-Efficient Graph Embeddings via Recursive Sketching. Yang et al., KDD 2019

RandNE: algorithm overview

Approximate high-order M by iterative random projections



1. Start with $\mathrm{U}_0 = \mathrm{QR}(\mathcal{N}(0, \frac{1}{d}))$

2. Iteratively set $\mathrm{U}_\mathrm{i} = \mathrm{A} * \mathrm{U}_{\mathrm{i}-1}$

3. Final embedding is a weighted sum $\mathrm{U} = \alpha_0 \mathrm{U}_0 + \alpha_1 U_1 + \cdots$

RandNE: asymptotics and practice

NB: For decent downstream performance need to tune α_i

Typically, we only need 3 multiplications ($i = 3$)

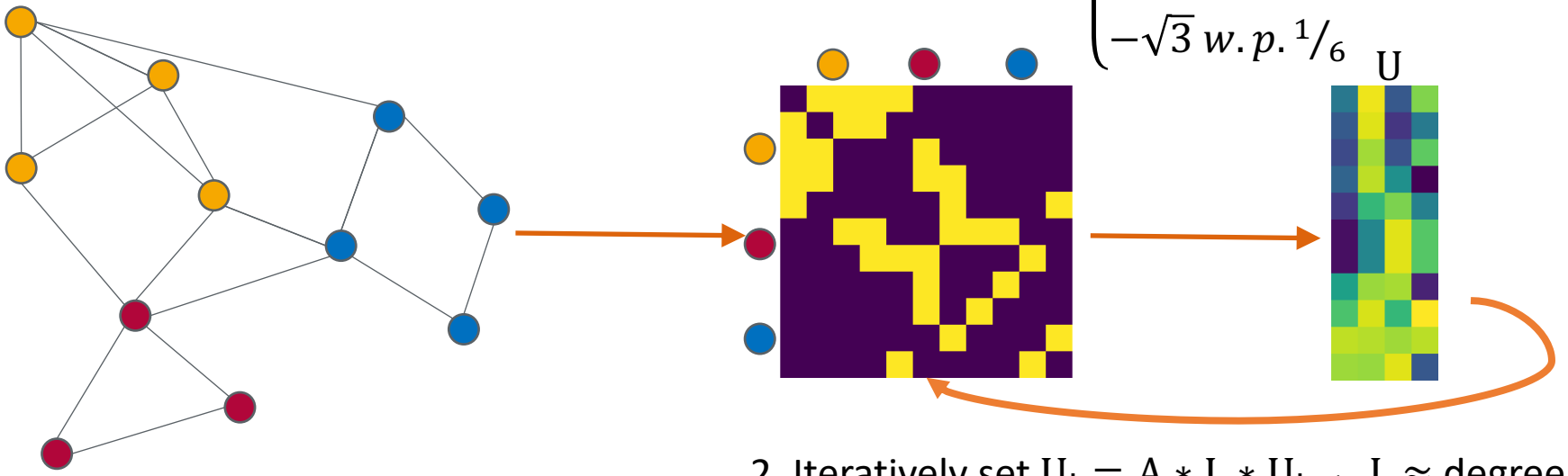
Overall complexity is ~ $O(m + d^2 n)$:)

Python and MATLAB code available

Practical limitations: ~1B nodes (need to store all U's)

FastRP: algorithm overview

Approximate high-order M by iterative random projections



1. Start with $\mathrm{U}_0 = \begin{cases} \sqrt{3}\ w.p.\ 1/6 \\ 0\ w.p.\ 1/3 \\ -\sqrt{3}\ w.p.\ 1/6 \end{cases}$

2. Iteratively set $\mathrm{U_i} = \mathrm{A} * \mathrm{L} * \mathrm{U_{i-1}}$, $\mathrm{L} \approx$ degree normalization

3. Final embedding is a weighted sum $\mathrm{U} = \alpha_0 \mathrm{U}_0 + \alpha_1 U_1 + \cdots$

Fast and Accurate Network Embeddings via Very Sparse Random Projection. Chen et al., CIKM 2018

FastRP: asymptotics and practice

NB: For good downstream performance need to tune α_i, β

Typically, we only need 4 multiplications ($i = 4$)

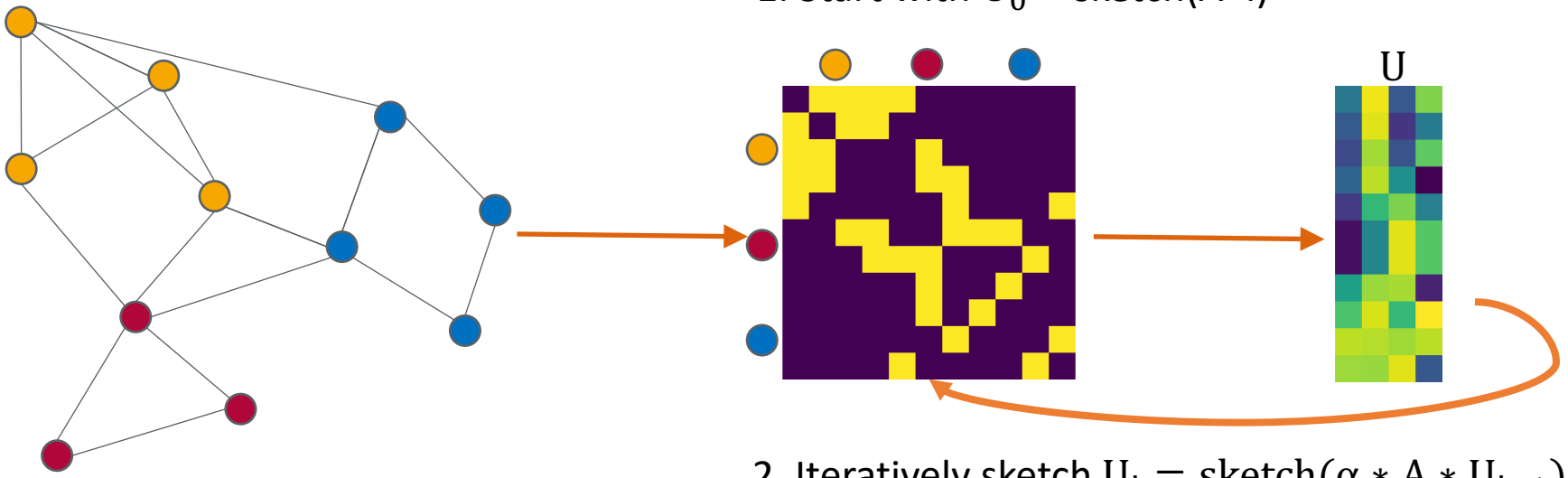
Overall complexity is ~ $O(m)$:)

Python code available

Practical limitations: ~1B nodes (need to store all U's)

NodeSketch: algorithm overview

Approximate high-order M by iterative sketching



FastRP: asymptotics and practice

NB: For good downstream performance need to tune α

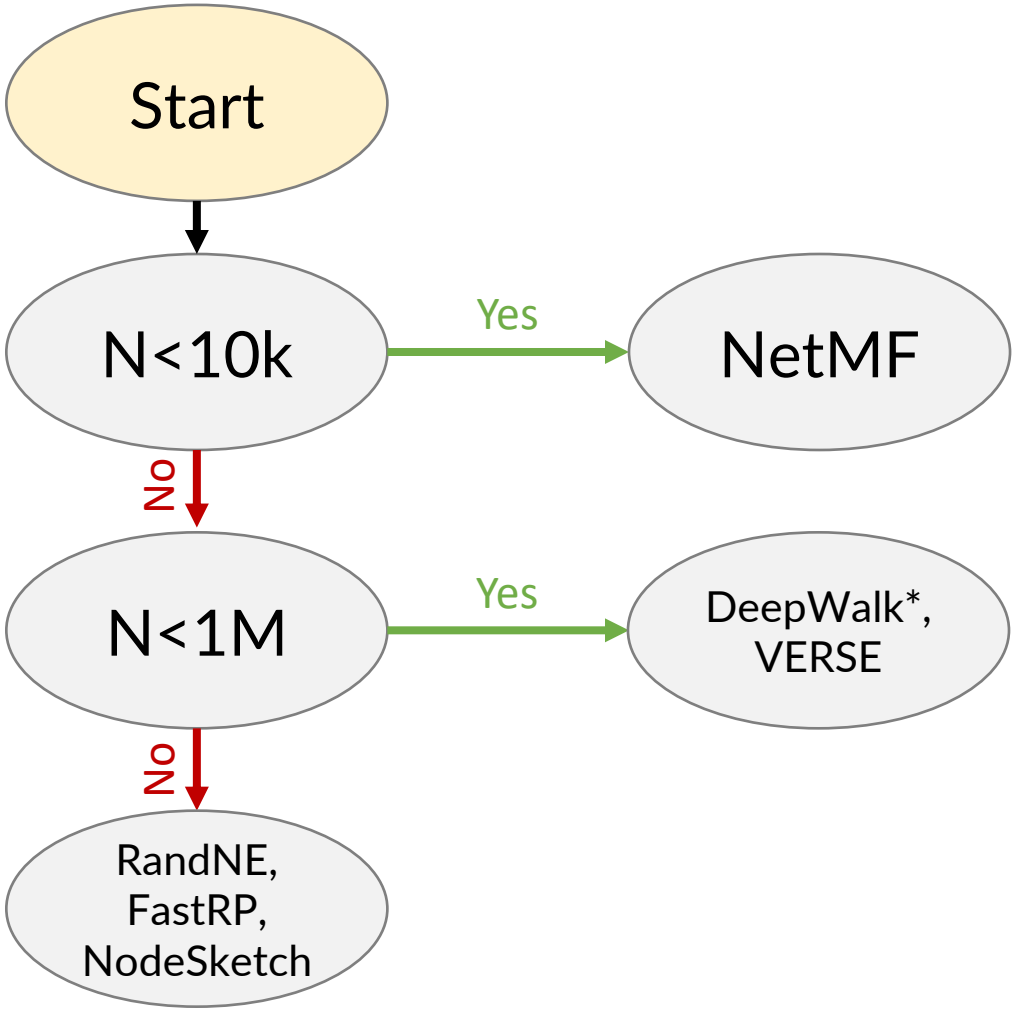
Typically, we only need 5 hashes $k = 5$

Overall complexity is ~ $O(mk)$:)

Python / C++ code available

Practical limitations: ~1B nodes (need to tune α)

Choosing the right algorithm



Edge embeddings

Operator	Result
Average	$(\mathbf{a} + \mathbf{b})/2$
Concat	$[\mathbf{a}_1, \ldots, \mathbf{a}_d, \mathbf{b}_1, \ldots, \mathbf{b}_d]$
Hadamard	$[\mathbf{a}_1 * \mathbf{b}_1, \ldots, \mathbf{a}_d * \mathbf{b}_d]$
Weighted L1	$[\lvert\mathbf{a}_1 - \mathbf{b}_1\rvert, \ldots, \lvert\mathbf{a}_d - \mathbf{b}_d\rvert]$
Weighted L2	$[(\mathbf{a}_1 - \mathbf{b}_1)^2, \ldots, (\mathbf{a}_d - \mathbf{b}_d)^2]$

Table 2.3: Vector operators used for link-prediction task for each $u, v \in V$ and corresponding embeddings $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$.

NB: choose operator depending on the algorithm

Questions?

twitter	twitter.com/tsitsulin_
website	tsitsul.in/talks/ods ← presentation
write me	anton@tsitsul.in