Clustering II

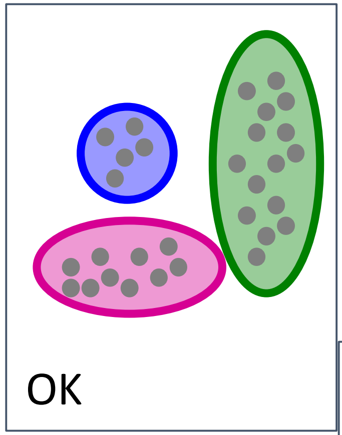
The CURE Algorithm

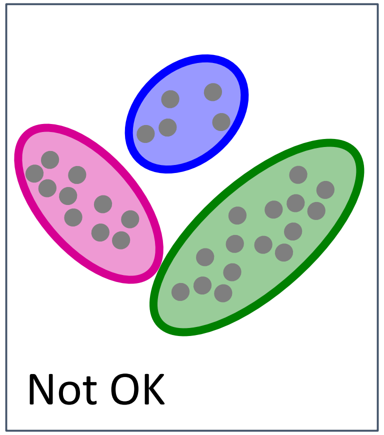
Recap

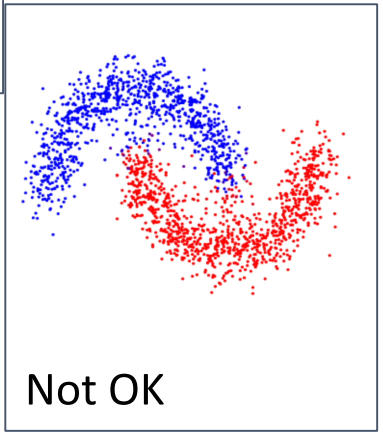
- Transition (Random Walk) Matrix
- Convergence of Random Walk
- Graph Laplacian,
- Graph Spectra, Eigen gap,
- Structure of the Web,
- PageRank,
- Topic-Specific PageRank
- **PageRank with Restarts**
- **Hierarchical Clustering**
- **K-means**
- **BFR algorithm**

The CURE Algorithm

- **Problem with BFR? Any ideas?**
  - Assumes clusters are normally distributed in each dimension
  - And axes are fixed – ellipses at an angle are ***not OK***







- **CURE (Clustering Using REpresentatives):**
  - Assumes a Euclidean distance
  - Allows clusters to assume any shape
  - **Underlying idea: Uses a collection of representative points to represent clusters**
    - Instead of a centroid and std. deviation as in BFR

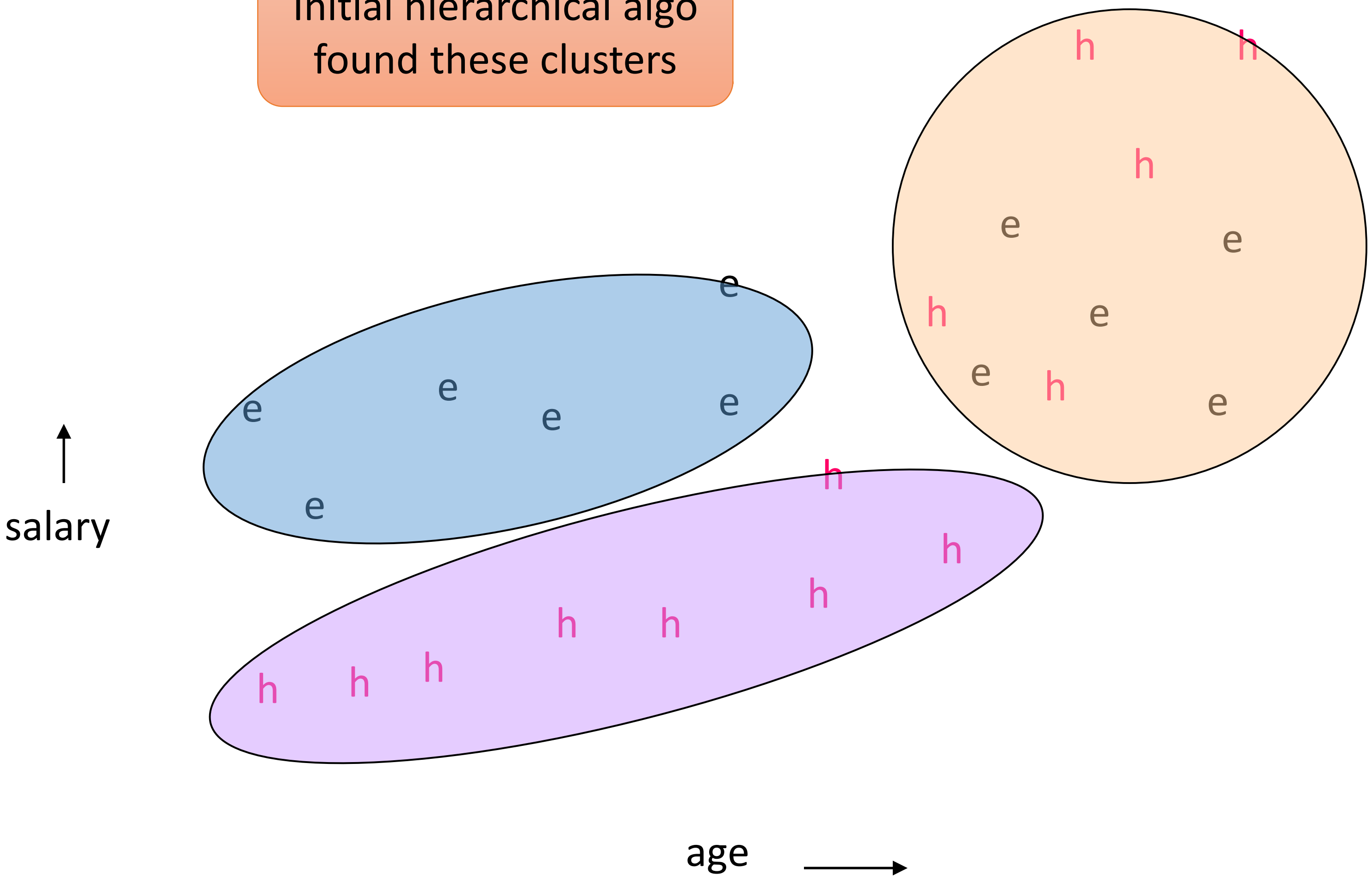
Starting CURE

Two Pass algorithm. Pass 1:

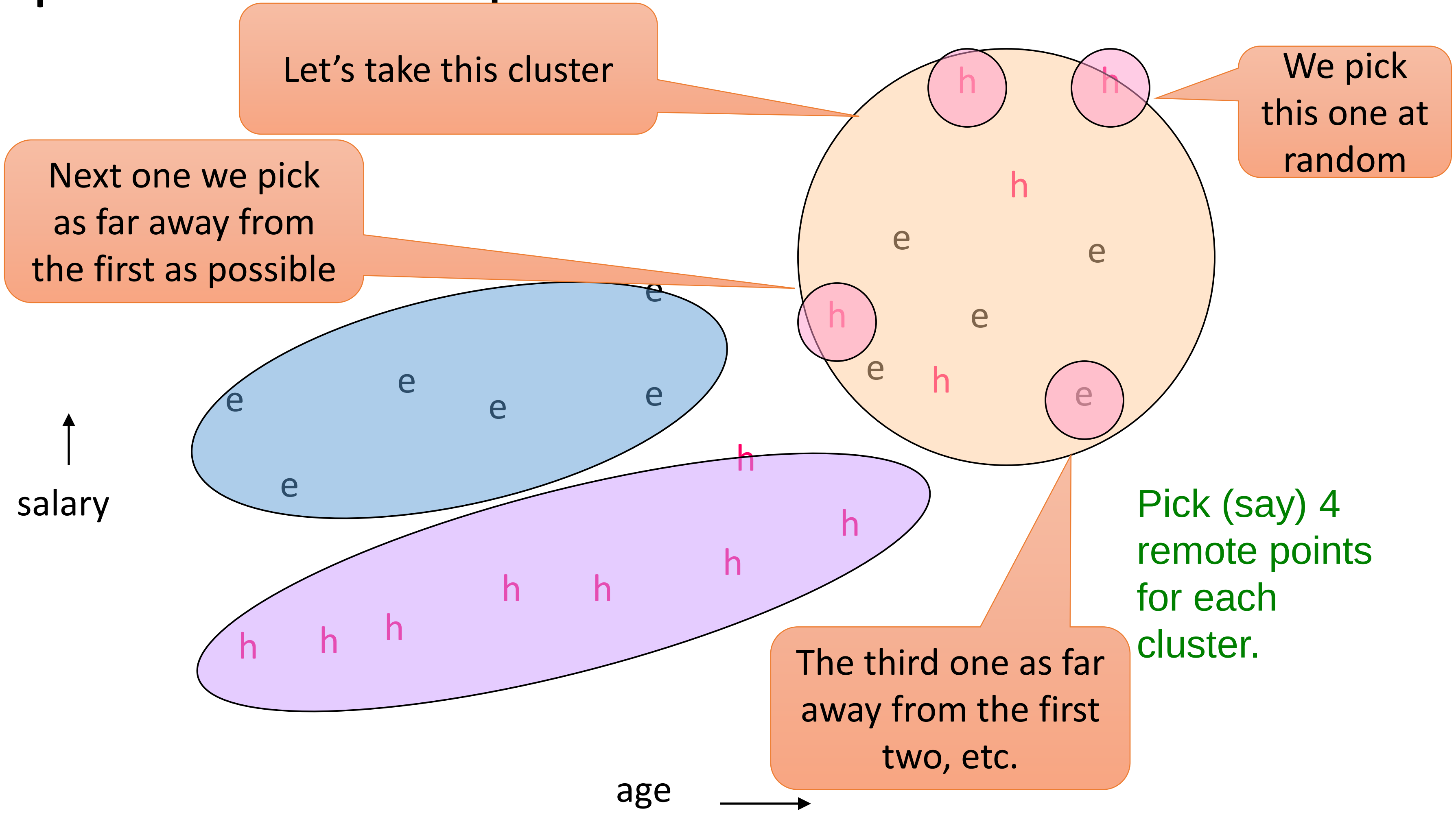
- **0) Pick a random sample of points that fit in main memory**
- **1) Initial clusters:**
  - Cluster these points e.g., hierarchically – group nearest points/clusters
    - Hierarchical algos can find clusters of any shape that way
- **2) Pick representative points:**
  - For each cluster, **pick *k* sample points**, as dispersed as possible (as discussed in previous techniques)
  - From the sample, pick **representatives (synthetic points)** by moving them (say) 20% toward the centroid of the cluster

Example: Initial Clusters

Initial hierarchical algo found these clusters

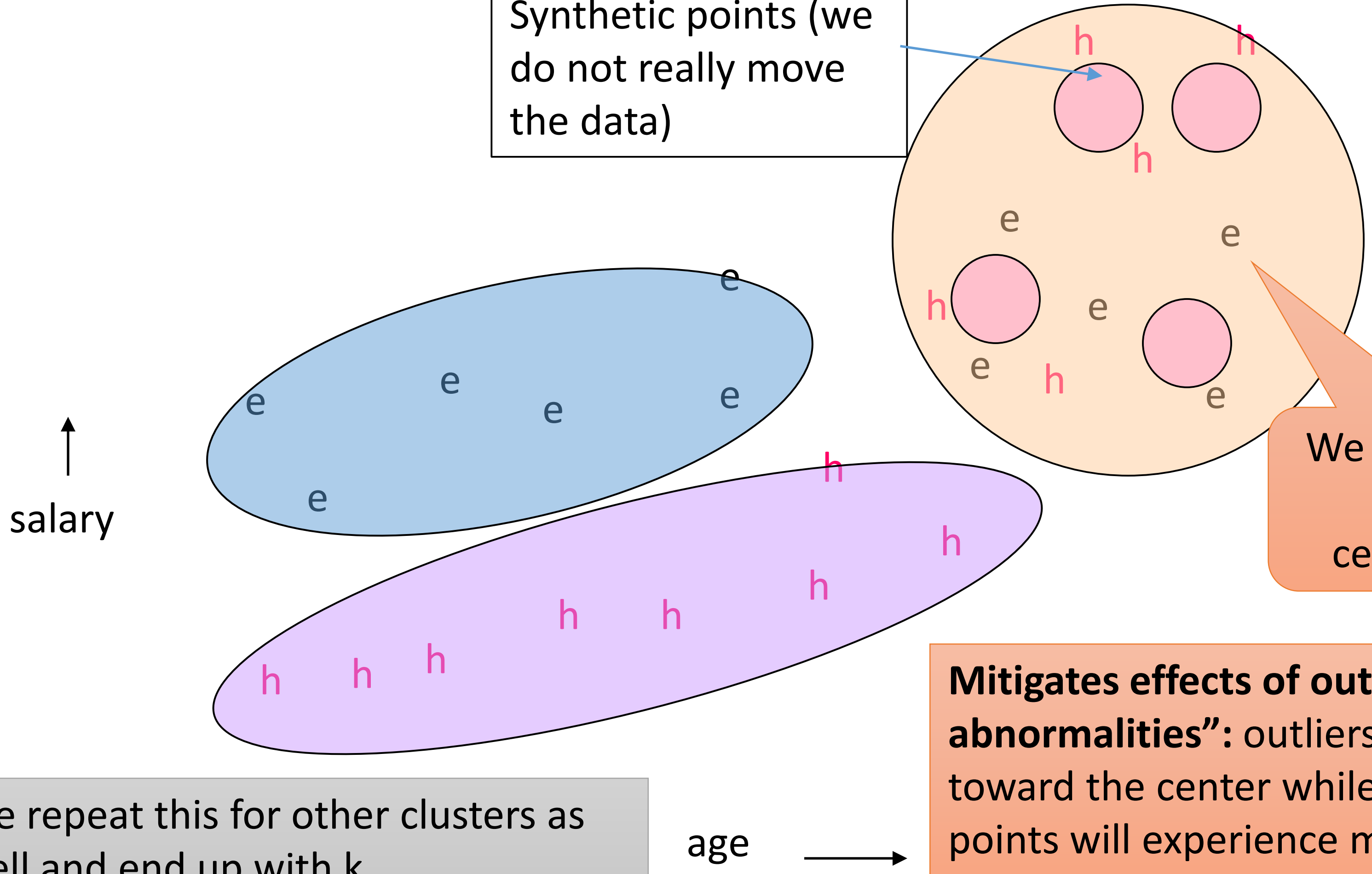


Example: Pick Dispersed Points



Example: Pick Dispersed Points

Synthetic points (we do not really move the data)

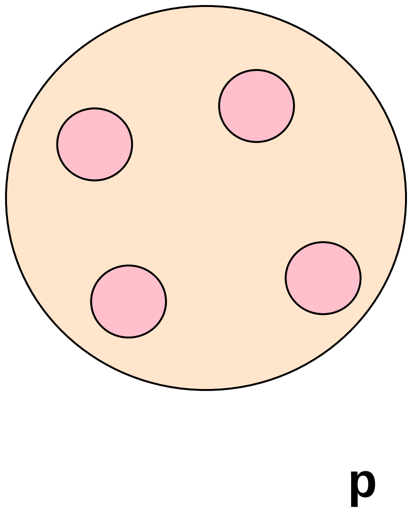


We move e.g, 20% toward the centroid. WHY?

Mitigates effects of outliers/"surface abnormalities": outliers will move toward the center while "normal" points will experience minimal shifts.

We repeat this for other clusters as well and end up with k representative (synthetic) points for each cluster

Finishing CURE



Pass 2:

- Now, rescan the whole dataset and visit each point ***p*** in the data set
- **Place it in the "closest cluster"**
  - Normal definition of "closest":
    Find the closest representative to ***p*** and assign it to representative's cluster

Summary

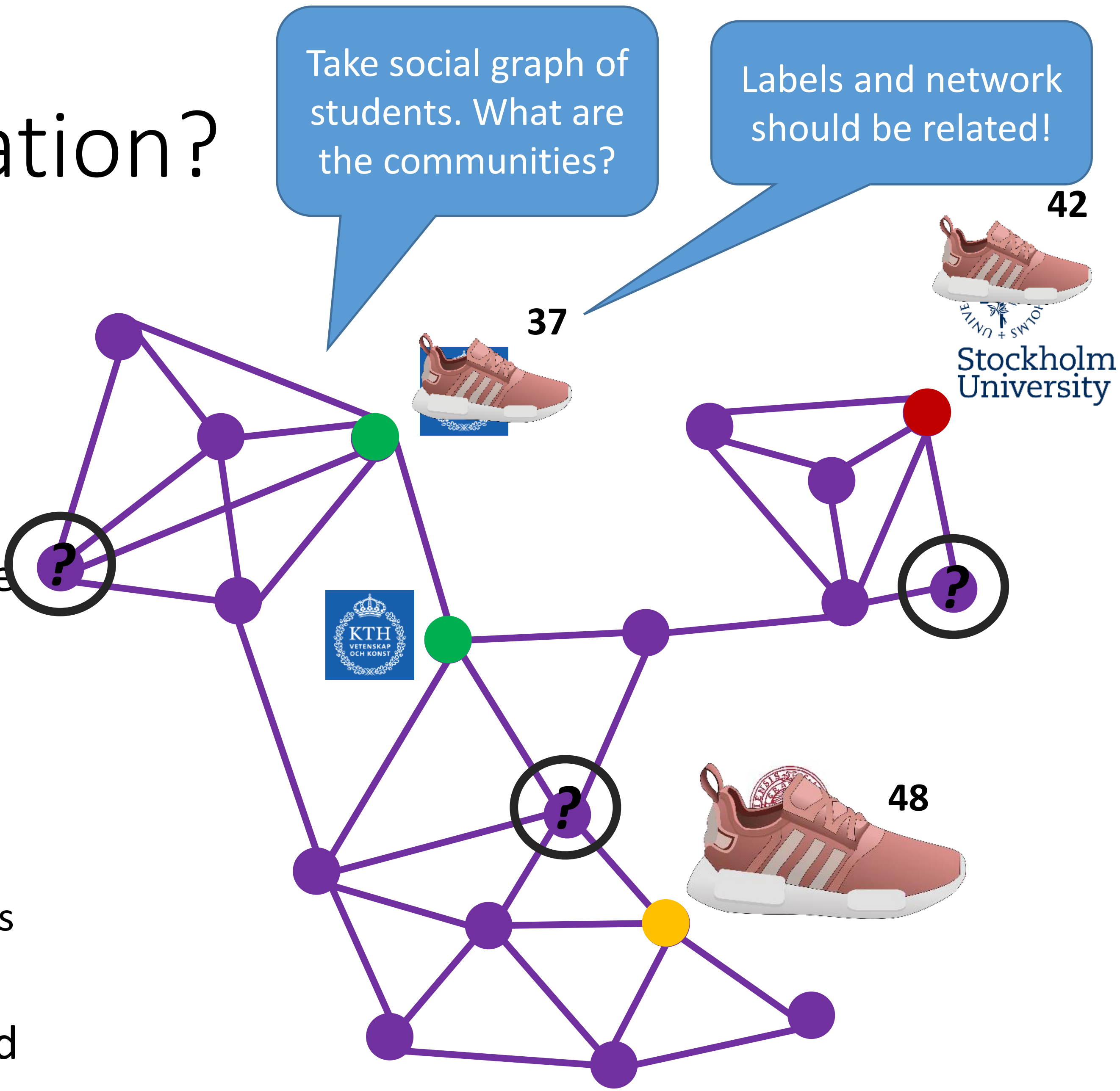
- **Goal of Clustering:** Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***
- **Algorithms:**
  - Agglomerative **hierarchical clustering**:
    - Centroid and clustroid
    - Issues with scalability!
  - ***k*-means:**
    - Initialization, picking *k*
    - Number of rounds to converge?
  - **BFR**
    - One pass algo! Strong assumptions on the data!
  - **CURE**
    - Overcomes limitations of BFR

Kahoot! time

Label Propagation and Graph Spectra

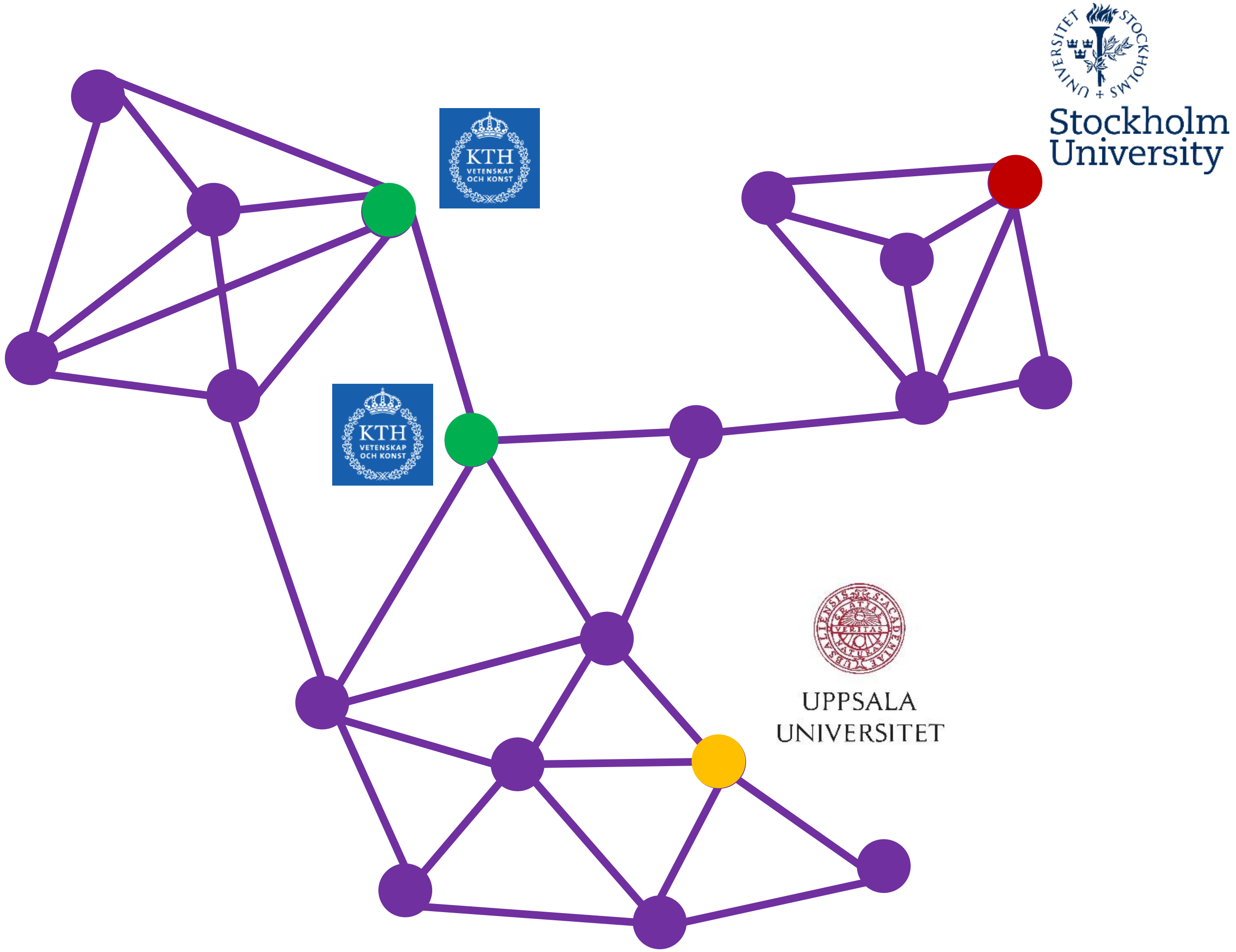
What is Label Propagation?

- **Label propagation**
  - Some nodes are labeled (ground truth)
    - Categorical/numeric/binary values
  - Task: Predict/Label for the rest of the nodes in the graph
  - Key assumption:
    - We assumed linked nodes are correlated (e.g., homophily, influence)
    - Labels propagate only on edges
    - You should have “enough” initial labels
  - Semi-supervised Learning
  - Why don’t we just go for a supervised learning?



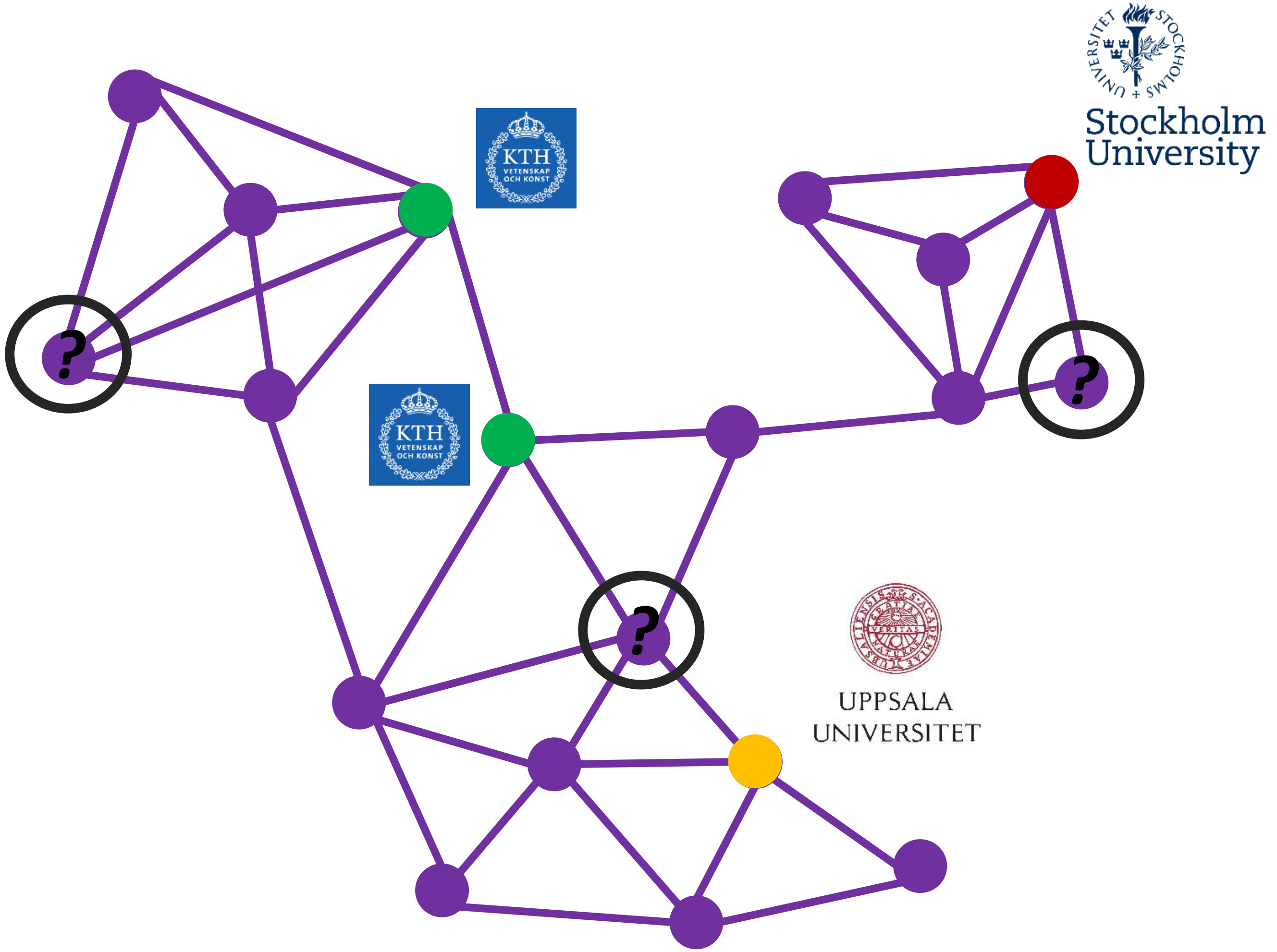
Network Classification

- **Label propagation**
  - Given graph G with nodes V, where
    - Nodes V_l given labels Y_l
    - Nodes V_u do not have labels
  - We need to find Y_u
  - Labels can be
    - Binary
    - Multi-class
    - Real values
  - Why?
    - Labelling/annotating data is expensive,
    - Small amounts of labelled data, large amounts of unlabelled data.



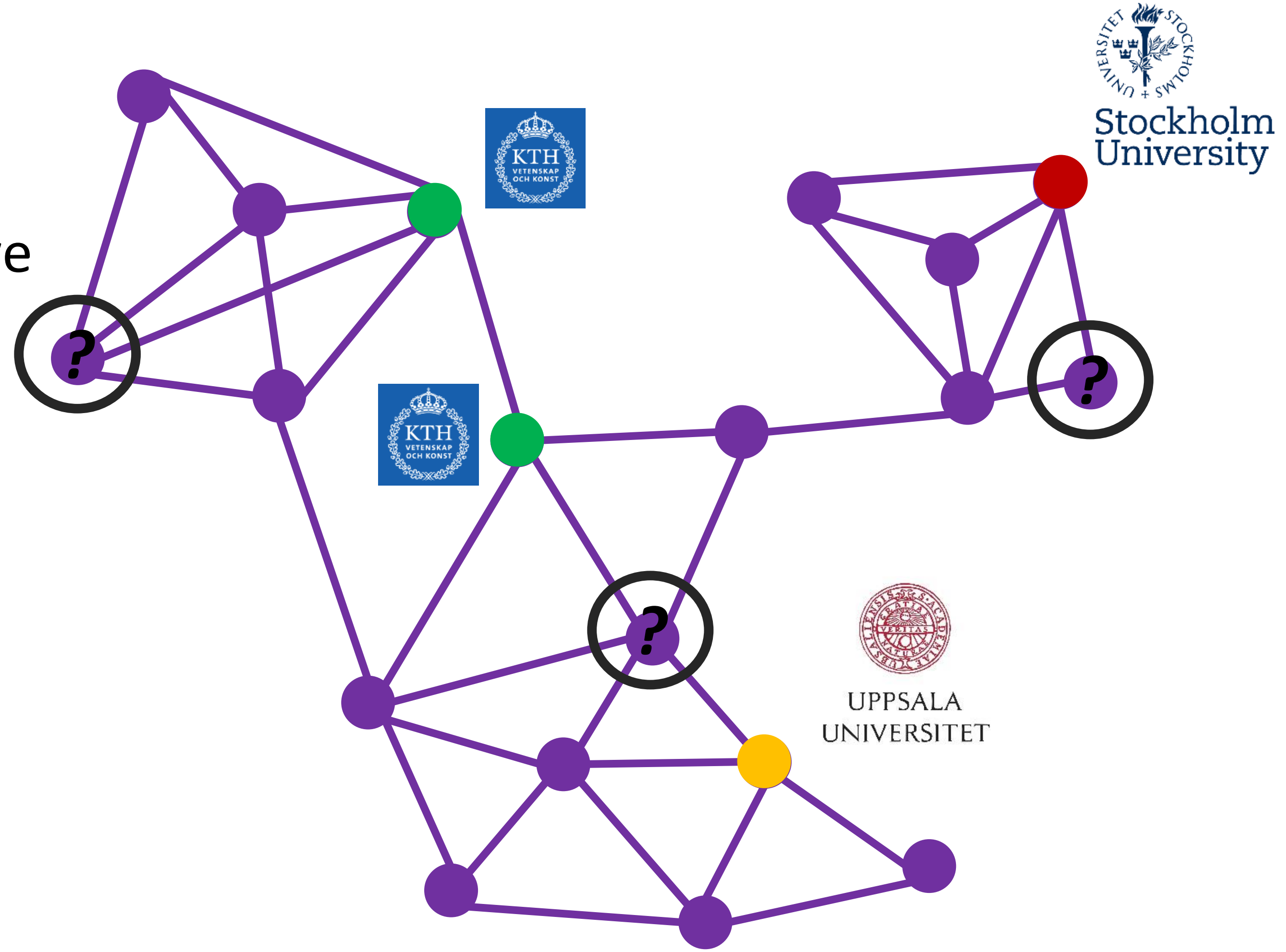
How to predict the labels?

- **Ideas?**
- Example:
  - Look at the local neighborhood
    - See what's the dominant label.
    - Or mean, Or average, or any other ML classifier.
  - Adopt that label.
  - What if your neighborhood does not have any labels?
    - Wait until it "propagates" to you.



Label Propagation

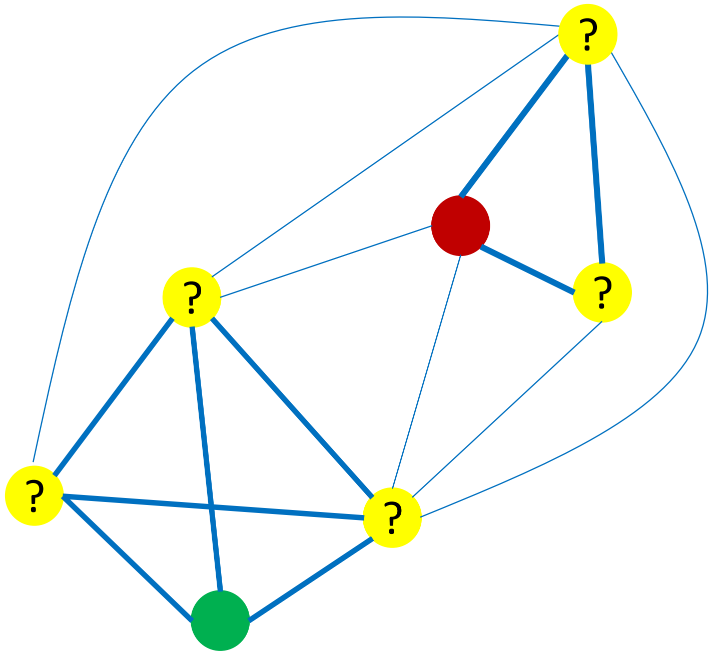
- We start with a graph and partially labeled nodes
  - If we do not have a graph, then we construct it by linking data points based on their **similarity**.
- We apply label propagation through that graph to predict the rest of the nodes
- Typical – semisupervised learning using graphs.



Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In ICML, 2003.

Label Propagation through Random-Walks with absorbing states

- **Idea:**
  - You start **many random walkers** from an **unlabeled** node
  - Keep random walking until you **hit a labeled node**.
    - Then your random walker gets stuck there – it's a trap!
  - Check which random walkers got stuck in which labelled node.
    - E.g., out of 100 random walkers 85 got stuck in Green and 15 at red.
  - **Adopt** the dominant label
    - i.e., we want to calculate a probability for a given node that a random walker will end up in a particular label.
  - Can we do that through matrix multiplication?
    - Remember Page rank?
    - So what should be our P so that it represents the above idea?



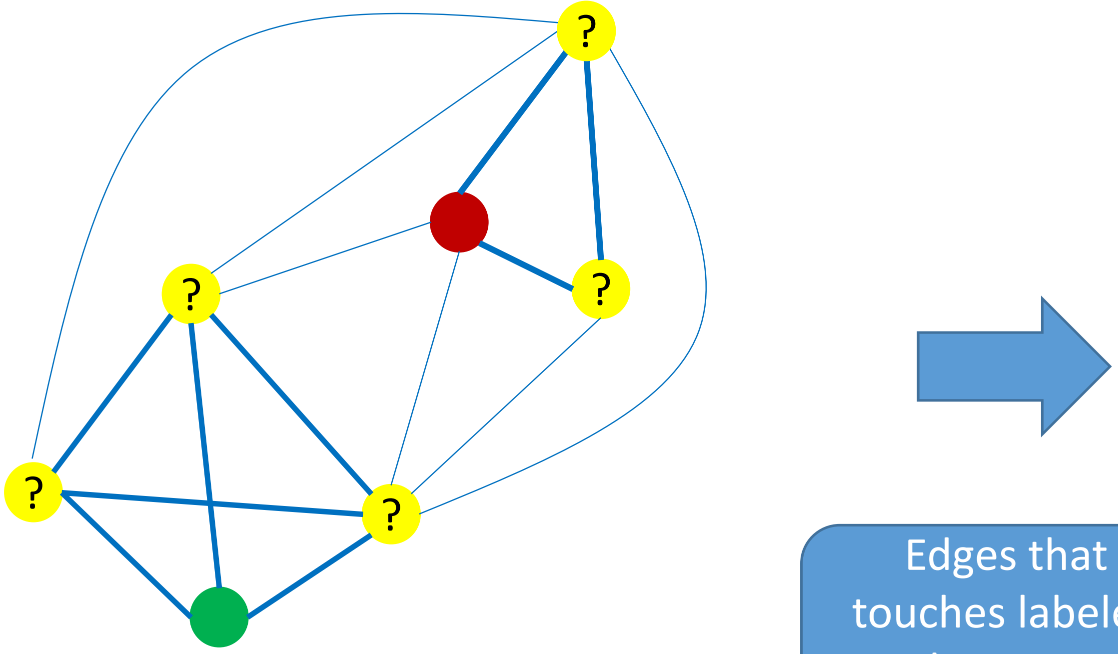
$$Y = P^{\infty} Y_{old}$$

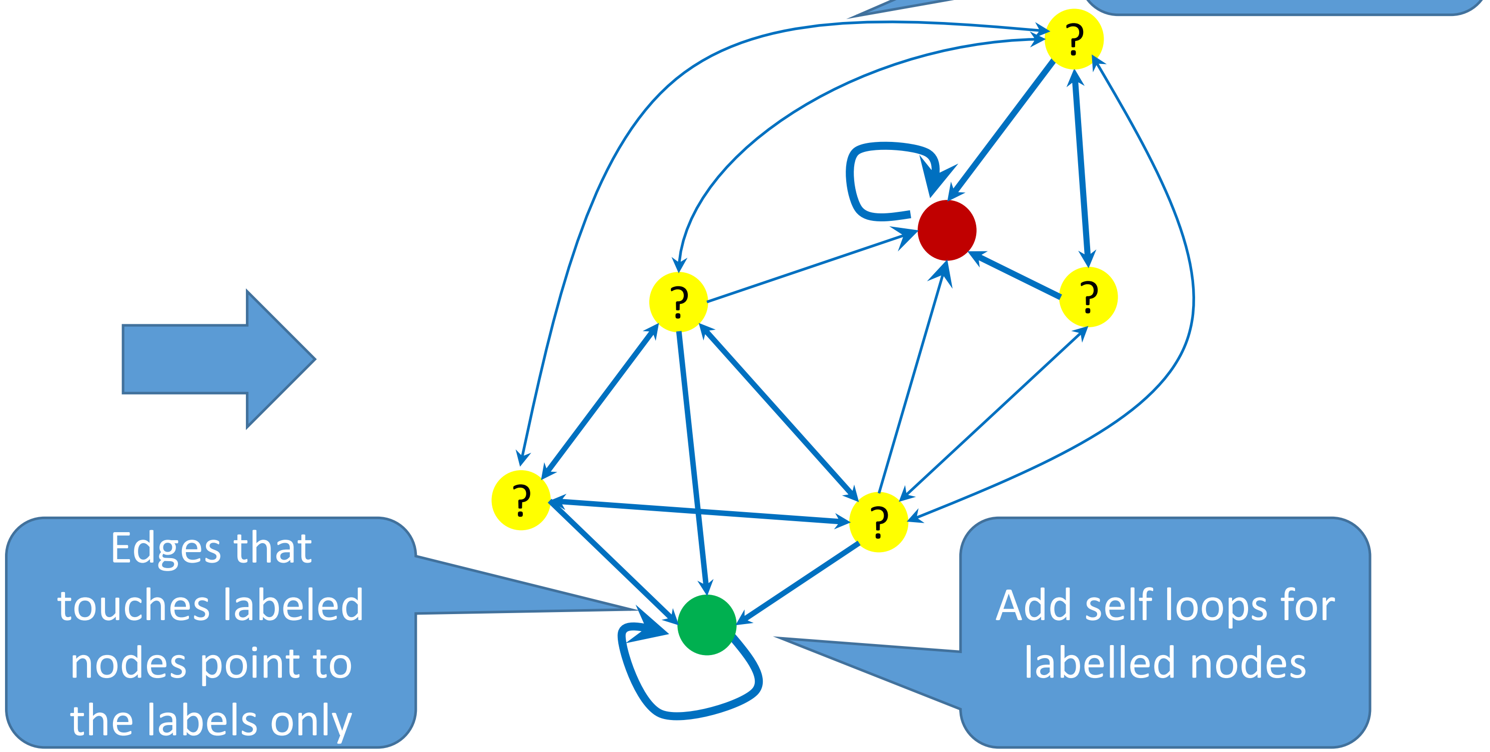
Labeled

Unlabelled

Changing the graph

- How to represent “getting stuck” on a particular label?

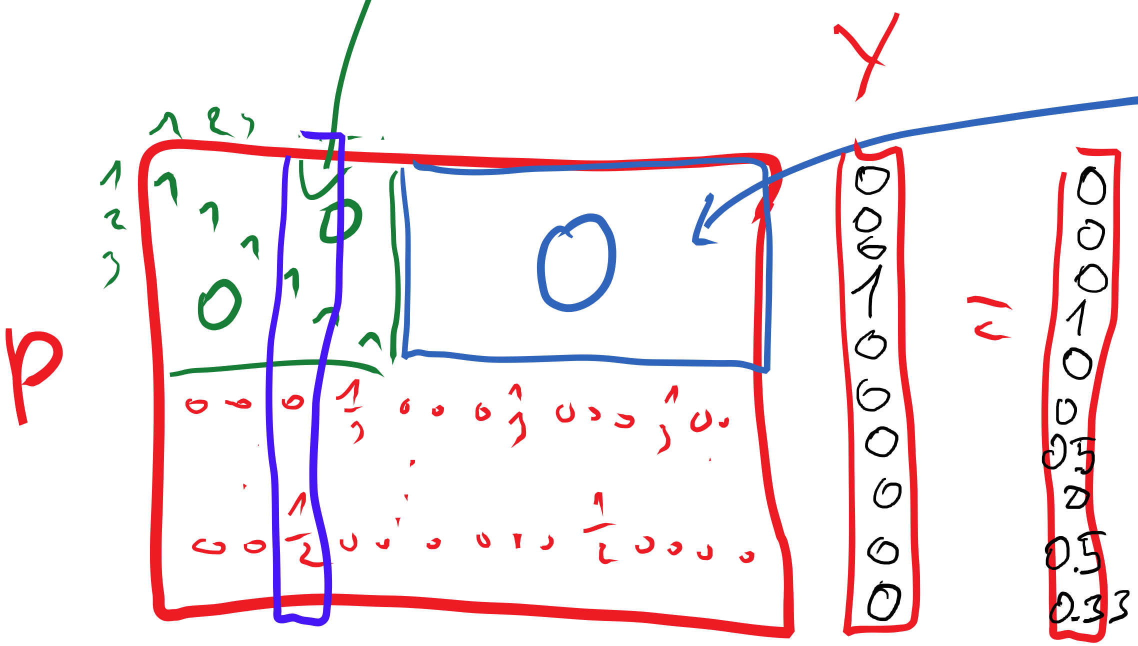


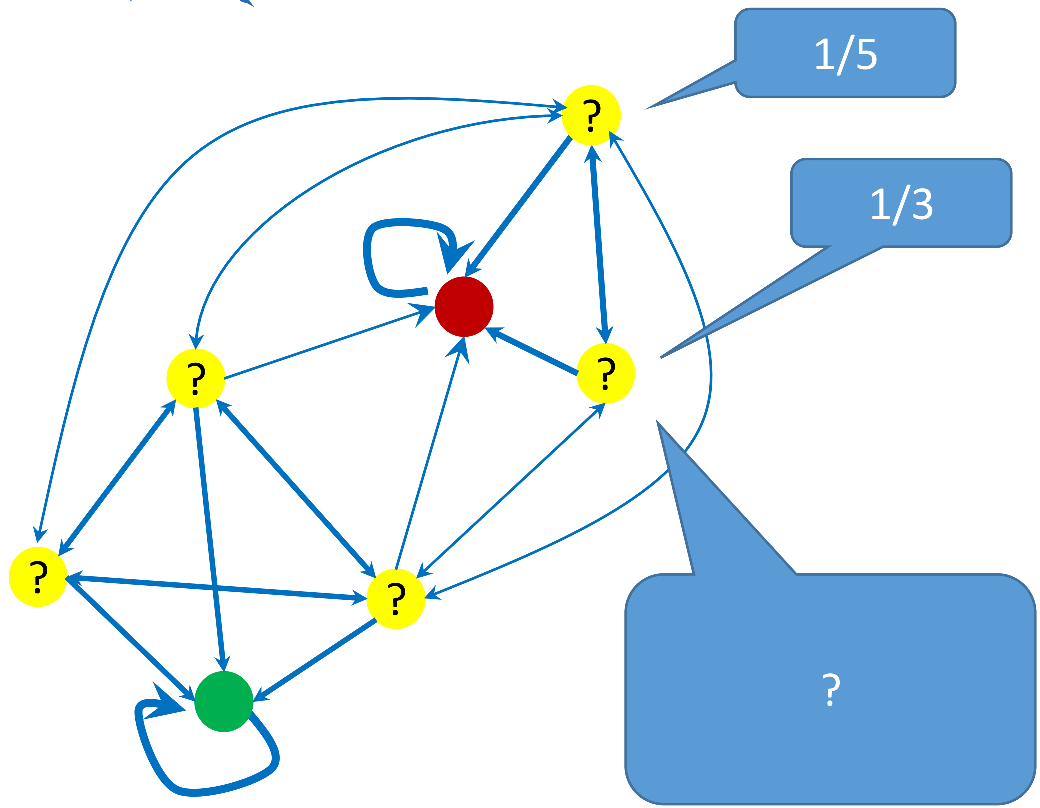


Absorption Matrix P

from labeld to labeled

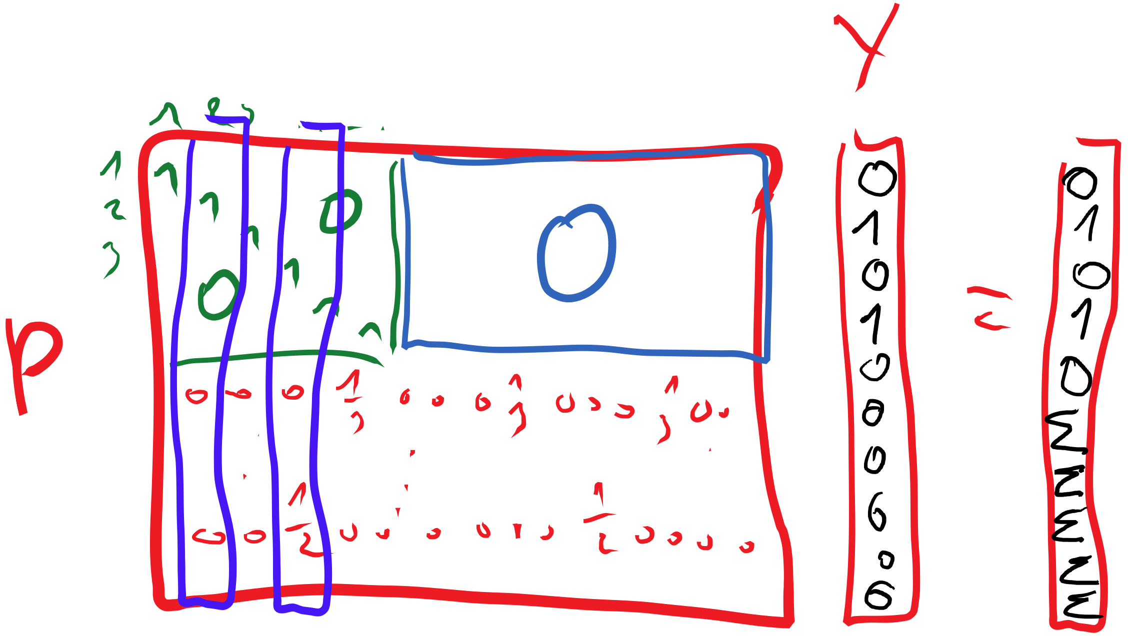
from labeled to unlabeled

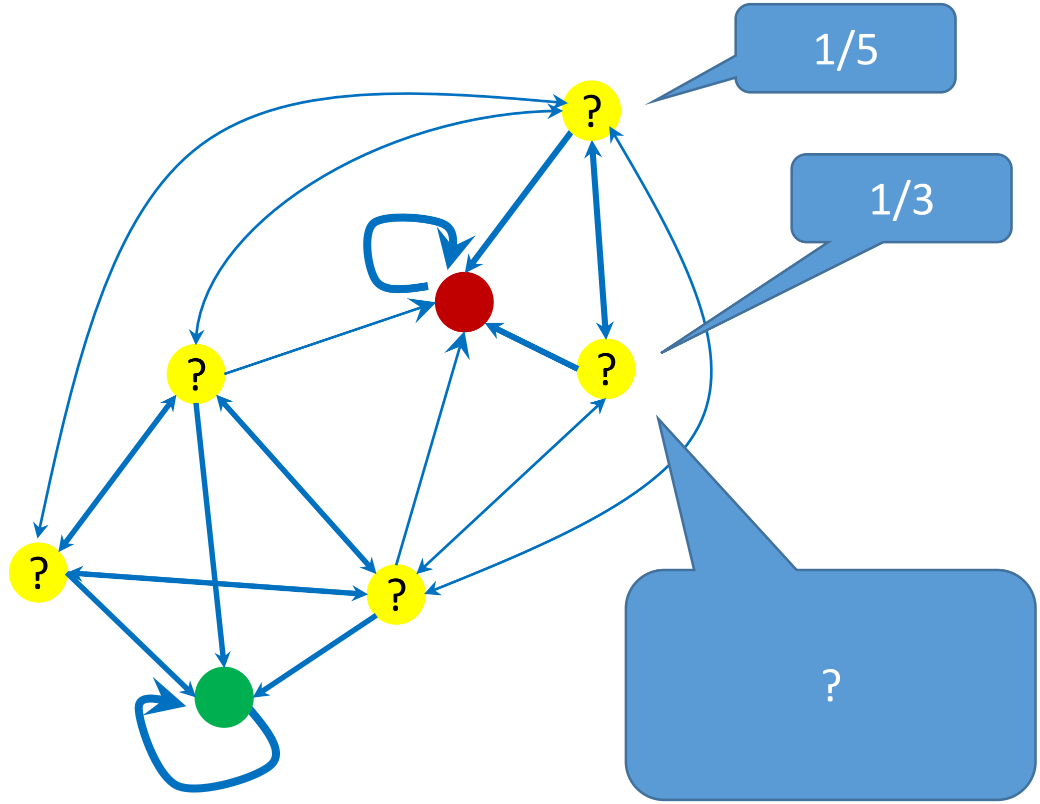




- Row-stochastic matrix P
- What is a chance to hit labelled node with a Random Walk with TTL=1?

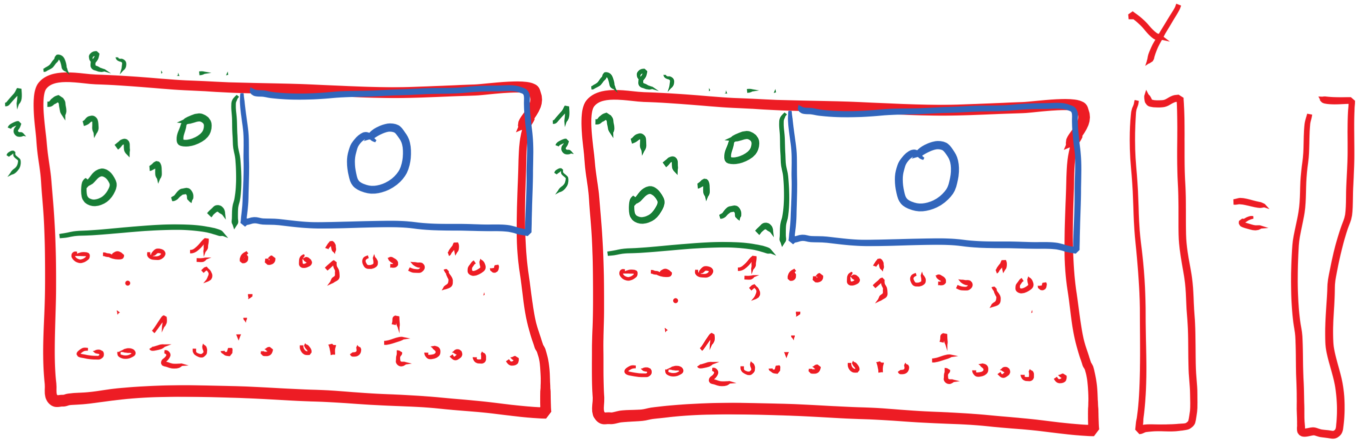
Absorption Matrix P – multiple labels of the same "color"



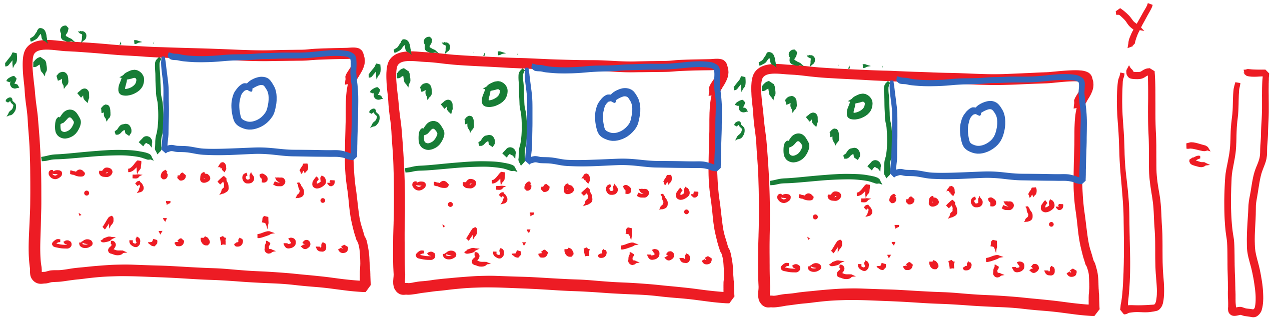


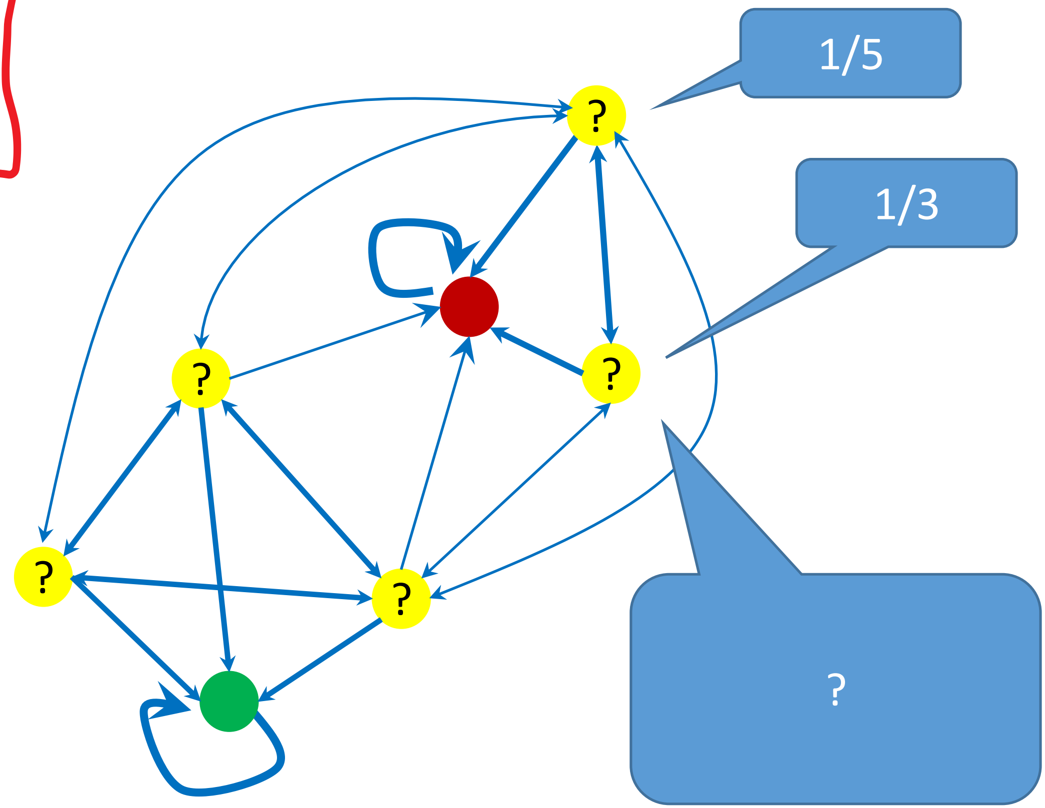
- Row-stochastic matrix P
- What is a chance to hit labelled node with a Random Walk with TTL=1?

What is a chance to hit labelled node with a Random Walk with TTL=2?

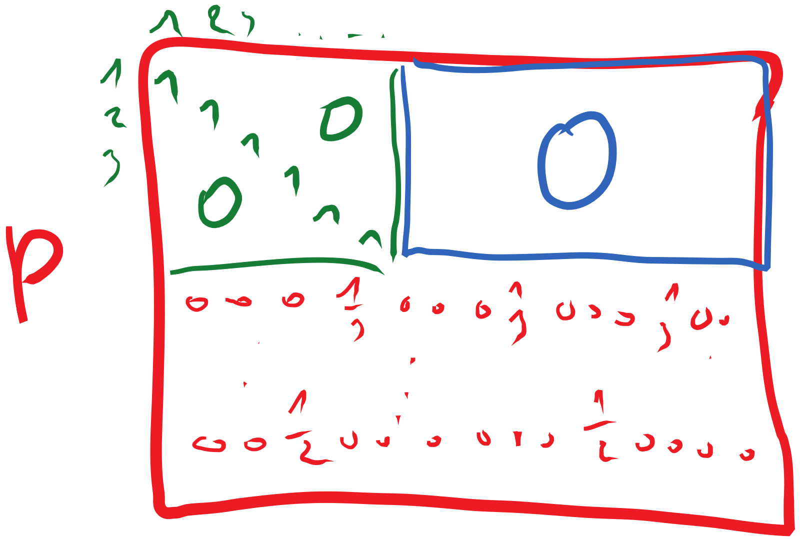


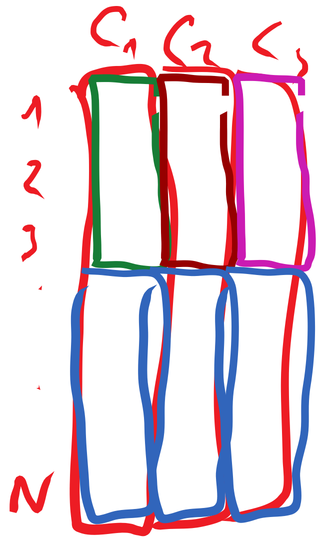
- What is a chance to hit labelled node with a Random Walk with TTL=3?

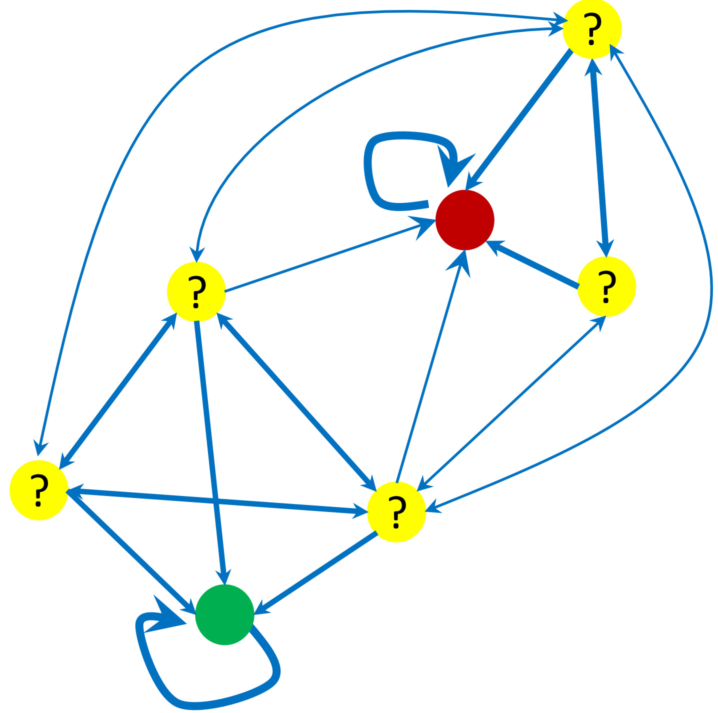




Dealing with labels of several "colors"







$$P \quad \begin{array}{c|cc} & N_1 \;\dots\dots\; N_l & \dots\dots\; N_n \\ \hline N_1 \;\cdot\;\cdot\; N_l & P_{ll} & P_{lu} \\ \cdot\;\cdot\; N_n & P_{ul} & P_{uu} \end{array}$$

$$Y \quad \begin{array}{c|c} & C_1 \;\dots\dots\; C_k \\ \hline N_1 \;\cdot\; N_l & Y_l \\ \cdot\;\cdot\; N_n & Y_u \end{array}$$

How (does) it converge?

- What happens to $\mathbf{P}^{\infty}$?

$$P \quad \begin{array}{cc} & \begin{array}{ccccc} N_1 & \ldots & N_l & \ldots & N_n \end{array} \\ \begin{array}{c} N_1 \\ \vdots \\ N_l \\ \vdots \\ N_n \end{array} & \begin{bmatrix} P_{ll} & P_{lu} \\ P_{ul} & P_{uu} \end{bmatrix} \end{array}$$

$$P= \quad \begin{array}{cc} & \begin{array}{ccccc} N_1 & \ldots & N_l & \ldots & N_n \end{array} \\ \begin{array}{c} N_1 \\ \vdots \\ N_l \\ \vdots \\ N_n \end{array} & \begin{bmatrix} I & 0 \\ \textit{something} & \textit{something} \end{bmatrix} \end{array}$$

$$\mathbf{P}^{\infty} = \quad \begin{array}{cc} & \begin{array}{ccccc} N_1 & \ldots & N_l & \ldots & N_n \end{array} \\ \begin{array}{c} N_1 \\ \vdots \\ N_l \\ \vdots \\ N_n \end{array} & \begin{bmatrix} I & 0 \\ (I-P_{uu})^{-1}P_{ul} & 0 \end{bmatrix} \end{array}$$

$(I-P_{uu})^{-1}P_{ul}$

What will be the labels?

$$\underbrace{\begin{array}{cc} & \begin{array}{ccc} C_1 & \cdots & C_k \end{array} \\ \begin{array}{c} N_1 \\ \vdots \\ N_l \\ \vdots \\ N_n \end{array} & \begin{bmatrix} Y_l \\ \end{bmatrix} \end{array}}_{Y} = \underbrace{\begin{array}{cc} & \begin{array}{ccccc} N_1 & \cdots & N_l & \cdots & N_n \end{array} \\ \begin{array}{c} N_1 \\ \vdots \\ N_l \\ \vdots \\ N_n \end{array} & \begin{bmatrix} I & 0 \\ (I-P_{uu})^{-1}P_{ul} & 0 \end{bmatrix} \end{array}}_{\mathbf{P}^{\infty}} \underbrace{\begin{array}{cc} & \begin{array}{ccc} C_1 & \cdots & C_k \end{array} \\ \begin{array}{c} N_1 \\ \vdots \\ N_l \\ \vdots \\ N_n \end{array} & \begin{bmatrix} Y_l \\ Y_u \end{bmatrix} \end{array}}_{Y_{\text{initial}}}$$

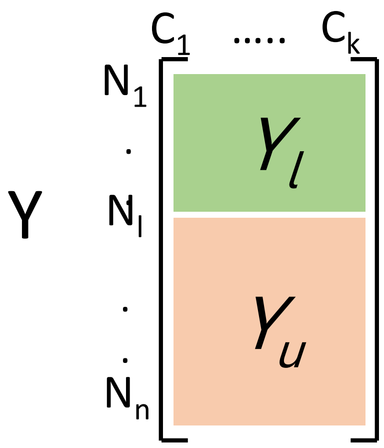
Probabilities that a random walker from a node i will get stuck on a label Cj

The only “problem“ is to calculate this matrix inverse.

Matrix Y

Y is a matrix, holding the soft probabilities of each instance

- $N_1, N_2, \ldots, N_n$ represent instances (i.e., nodes in graph)
- $C_1, C_2, \ldots, C_k$ represent the possible label
- Y_{ab} represent the probability of N_a being labeled as C_b



- For final labelling one has to account for "class proportions"!
  - Often highly disablanced labels.
  - Class balance might be known apriori
    - Probabilities can be weighted with the estimated size proportion of that category.
    - Class mass normalization!

$$
Y \quad
\begin{array}{c}
 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ \\ 100
\end{array}
\begin{array}{cc}
\text{Spam} & \text{Non-spam} \\
1 & 0 \\
1 & 0 \\
0 & 1 \\
0 & 1 \\
0.2 & 0.8 \\
0.4 & 0.6 \\
 & \vdots \\
0.7 & 0.3
\end{array}
\begin{array}{c}
 \\ \\ \\ \\ \\ \text{non-spam} \\ \text{non-spam} \\ \\ \text{spam}
\end{array}
$$

Zhu, Xiaojin, and Zoubin Ghahramani. "Learning from labeled and unlabeled data with label propagation." (2002)

Alternative Solution (original Label Propagation)

- Alternative solution on the "original graph", i.e., original random walk matrix
  - $D_{ii} = \Sigma_j A_{ij}$
  - $P = D^{-1} A$
- Converges to the same thing

Initialize matrix P and Y:
Use labeled data to initialize $Y^{(0)}$
(set Y_u rows to random values)

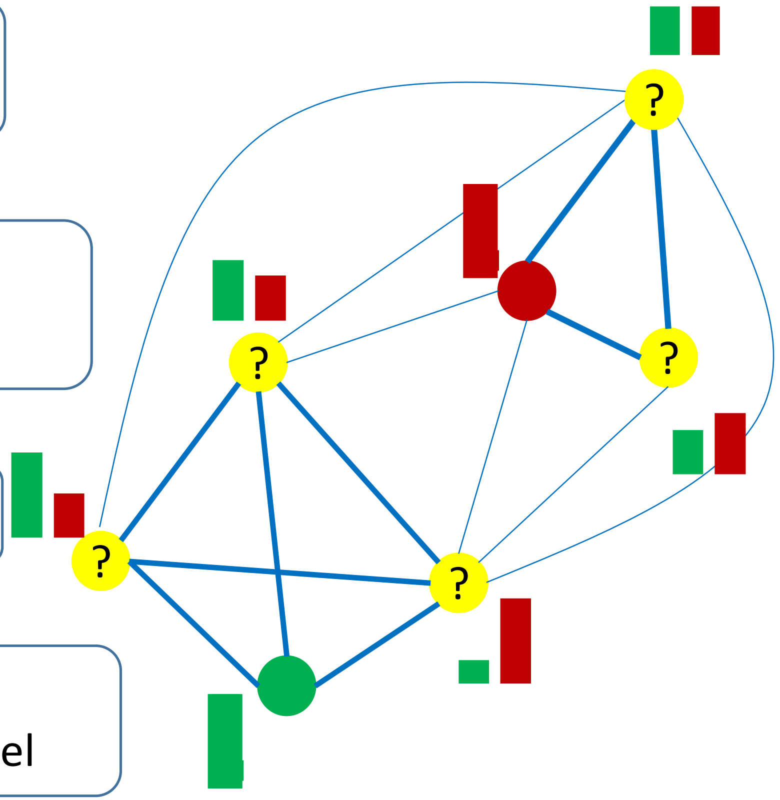
Propagate labels:
$Y^{(t+1)} = P\, Y^{(t)}$

Repeat till convergence

Reinforce labeled sources:
Set Y_l to initial probabilities

Row Normalize $Y^{(t+1)}$

Labeling approach:
e.g., use the most likely label



Optimization: Smoothing P using PageRank

- Creating a uniform transition matrix T, such that:

$$t_{ij} = \frac{1}{N}$$

- Update the probability transition matrix P, such that:

$$\mathbf{P} = \epsilon T + (1 - \epsilon)\,\mathbf{P}\,,$$

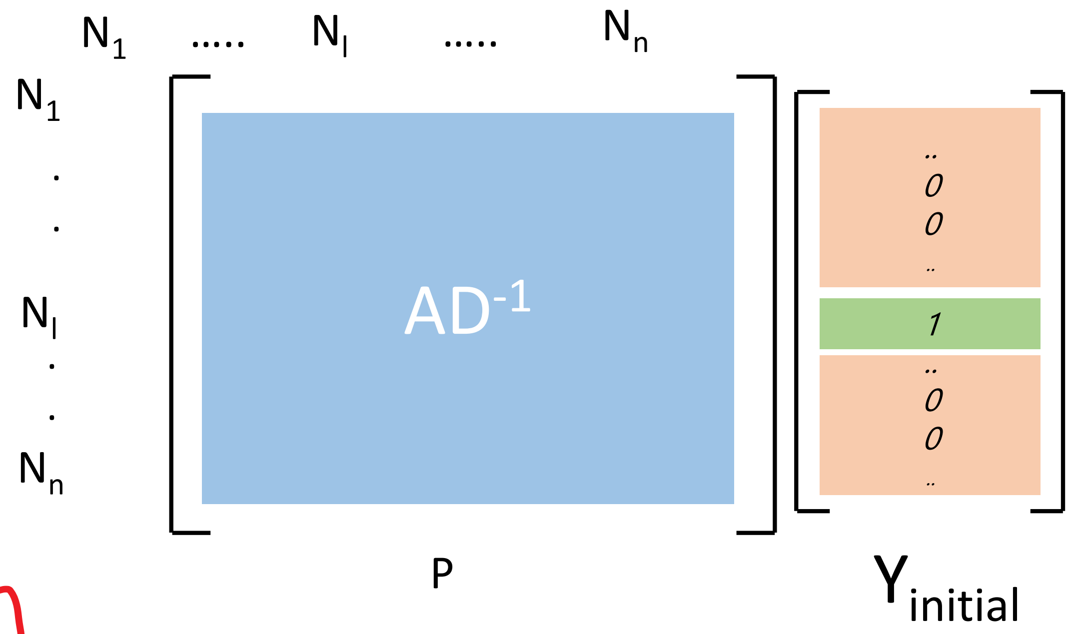
where ϵ is a parameter in $(0,1)$

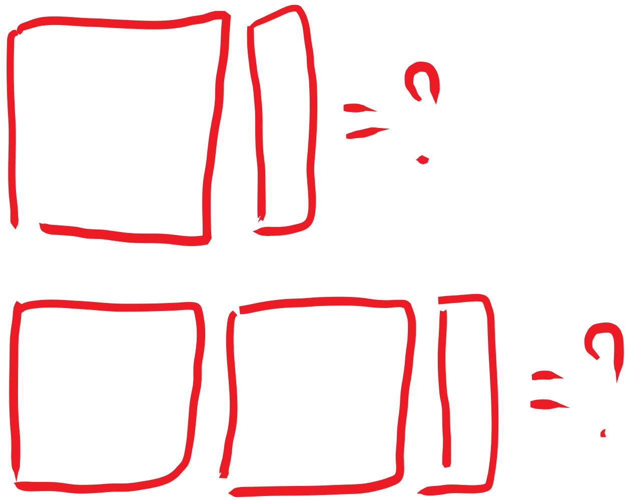
Things to remember

- **Label propagation: Network Classification**
  - Initial labeling is very important for having meaningful classes
  - Initially labelled nodes can never change their label!
    - Is it good or bad?
      - What if you have noisy labelled data?
  - That brings us to Diffusion
- **Label propagation: Community Detection**

Finding Communities with Diffusion

- A – Adjacency matrix of undirected graph G
- D – Degree matrix of graph G
- P – random walk Matrix $P = AD^{-1}$
- Let's start with a single label $Y_{initial}$
- What happens for $P*Y_{initial}$?
  - $P*P*Y_{initial}$?

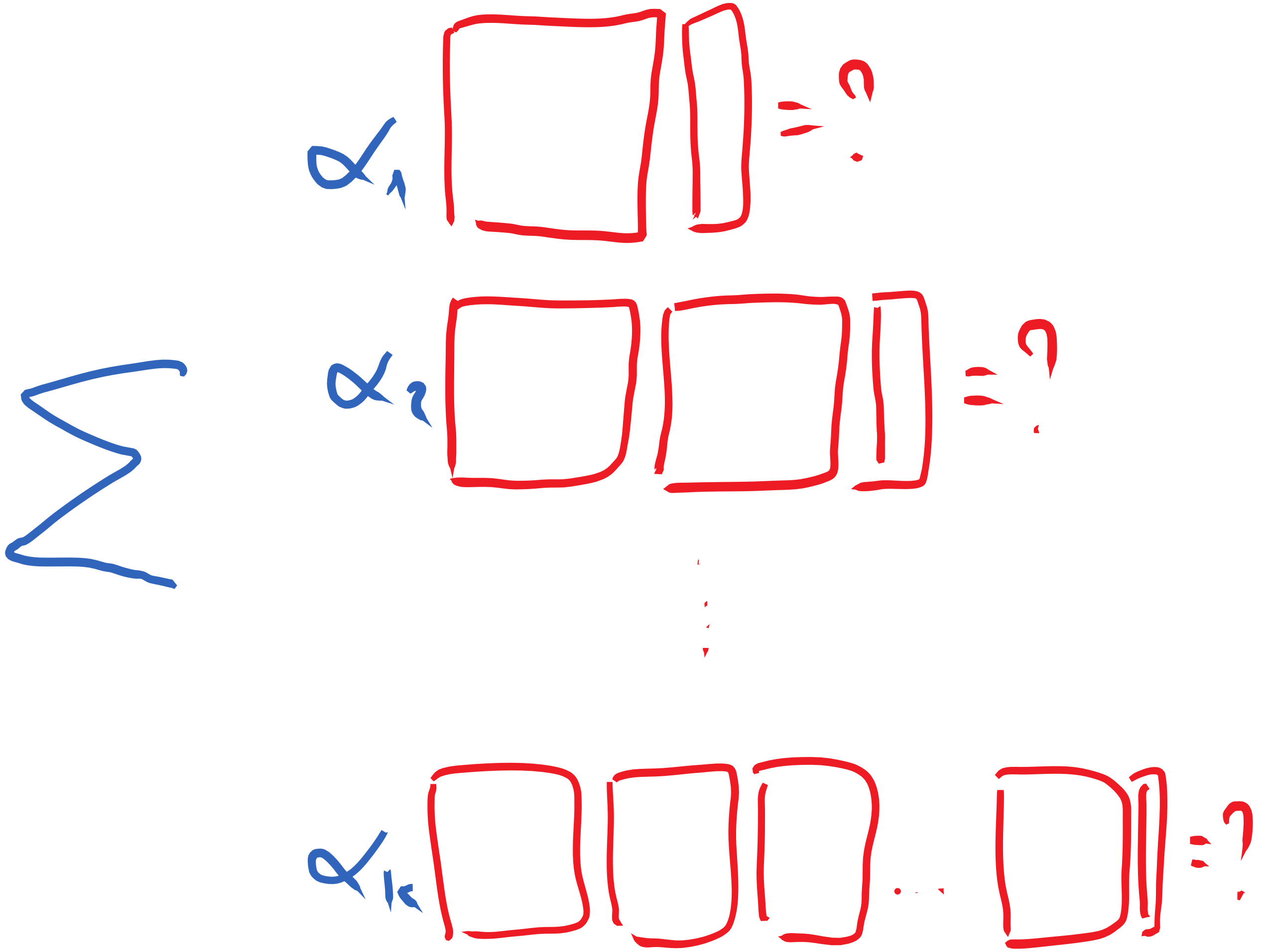




Finding Communities with Diffusion

- What happens for P*Y$_{\text{initial}}$?
  - P*P*Y$_{\text{initial}}$?
  - P*…*P*Y$_{\text{initial}}$?
- Graph diffusion is a sum:
  - $\mathbf{Y}_{\boldsymbol{dif}} = \sum_{k=0}^{\infty} \alpha_k \mathbf{P}^{\boldsymbol{k}} \mathbf{Y}_{\boldsymbol{initial}}$
    - Alpha provides decaying wegiht $\sum_{k=0}^{\infty} \alpha_k = 1$
  - High values in Y$_{\text{dif}}$ would indicate the belonging to a specific cluster
- Best known instance – personalized PageRank diffusion.

$$(1-\alpha)\sum_{k=0}^{\infty} \alpha^k \mathbf{P}^{\boldsymbol{k}} \mathbf{Y}_{\boldsymbol{initial}}$$



Andersen et al. Local Graph Partitioning using PageRank Vectors, 2006
Kloster et al. Heat Kernel Based Community Detection, 2016

Label Spreading (Zhou 2004)

- Input: Graph G(V; E)
- A – Adjacency matrix of graph G
- D – Degree matrix of graph G
- L – normalized Laplacian $L = D^{-½} A\, D^{-½}$

- Steps of the algo:
  - Initialize $Y_{initial}$ (non labelled nodes – zeros)
  - Repeat
    - $Y_{t+1} \leftarrow \alpha L\, Y_t + (1-\alpha)\, Y_{initial}$
  - until Y_t converges;

- If your matrix is not too big and you can calculate the inverse then the solution:
  - $Y_{converged} = (1-\alpha)(I - \alpha\, L)^{-1}\, Y_{initial}$

As if you'd restart from the initial seeds and "reinforce" initial rounds.

Zhou et al. Learning with local and global consistency, NIPS 2004