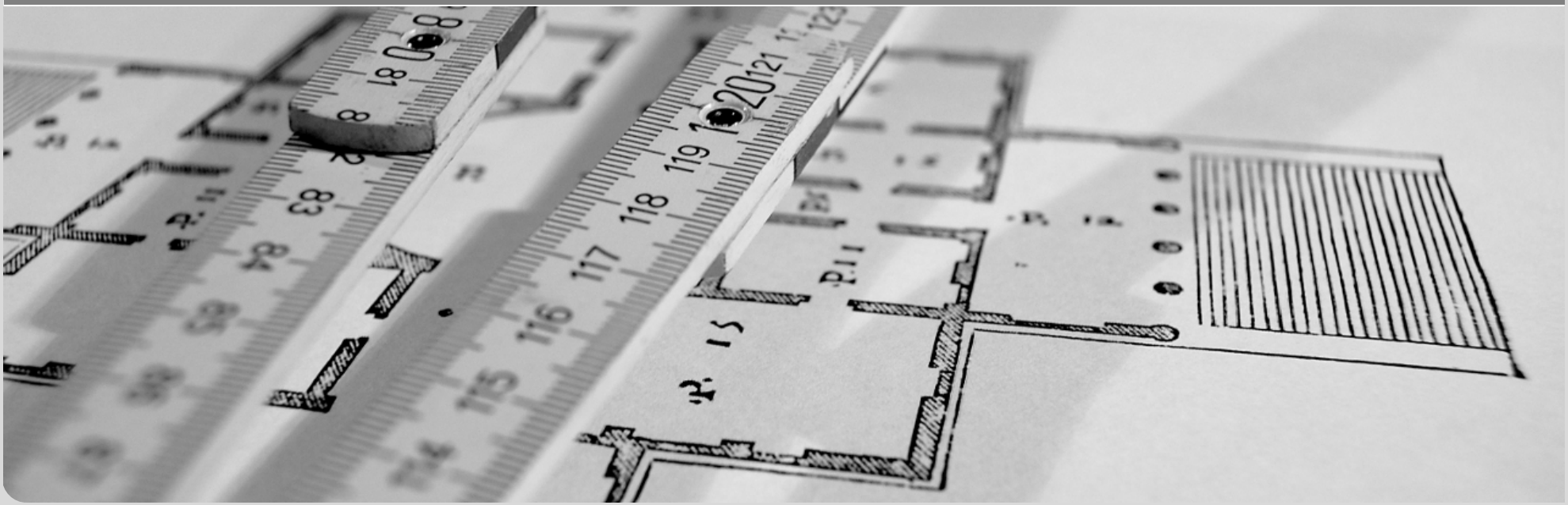
Karlsruher Institut für Technologie

Support Vector Machines via Multilevel Label Propagation

Bachelorarbeitsabschlußpräsentation, Betreuer: Sebastian Schlag, Christian Schulz

Matthias Schmitt | 11.07.2018

INSTITUT FÜR THEORETISCHE INFORMATIK, ALGORITHMIK II



Gliederung

Motivation

“Machine learning explores the study and construction of algorithms that can learn from and make predictions on data.”

Motivation

“Data is the oil of the 21st century.”

Klassifizierung

Training: Datenpunkte $X_i \ldots X_n$ und zugehörige Klassen $y_i \ldots y_n$

Beispiel: `forest` Datensatz
Features - Höhe, Gefälle, Sonnenstunden, Bodentyp ...
Klasse - eine von 7 Baumarten

Ziel: neue Datenpunkte klassifizieren, also y_{n+1} für X_{n+1} bestimmen

Support Vector Machine

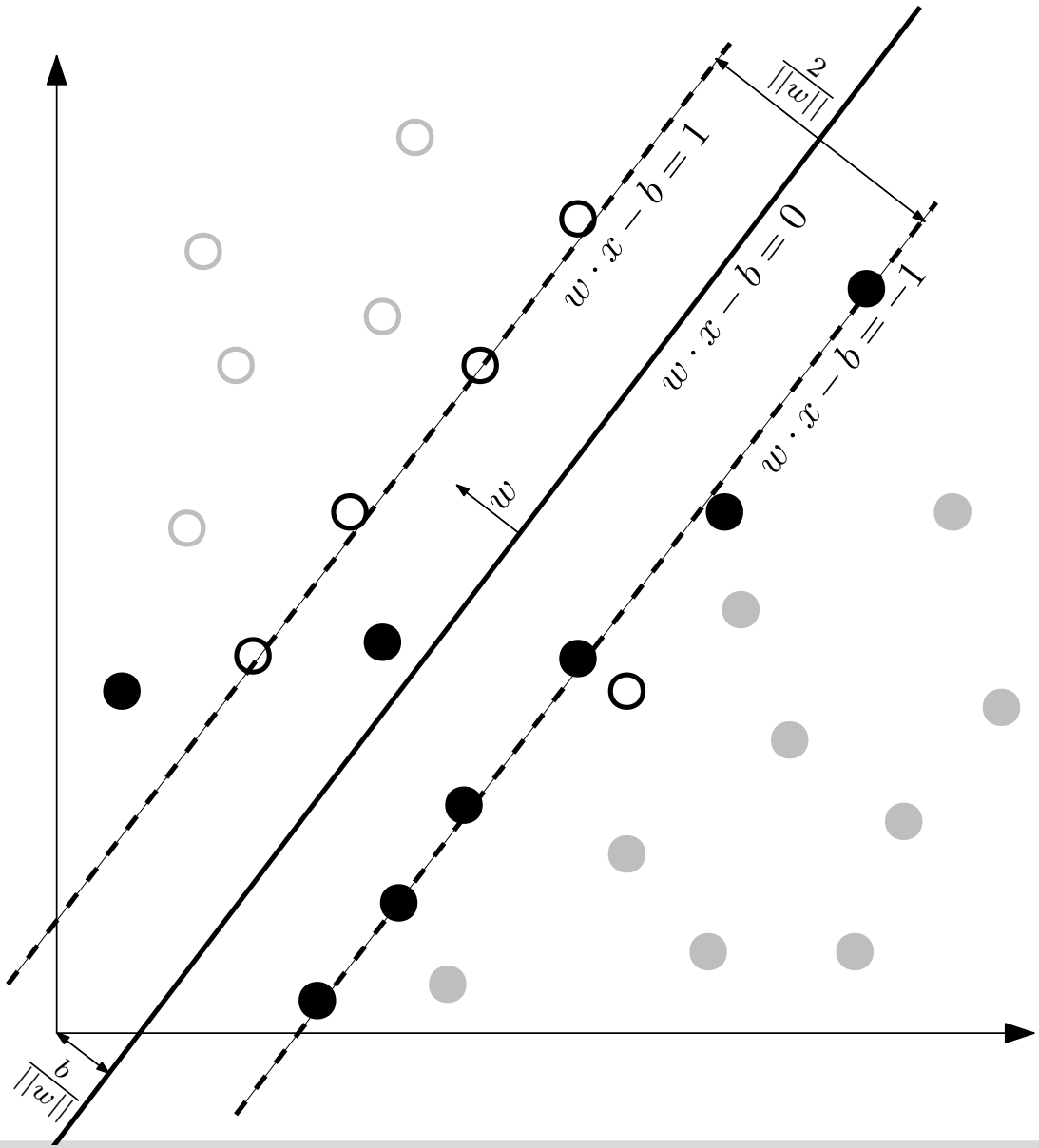
Hyperebene finden, die die Klassen mit größtmöglichem Abstand trennt

$$\begin{aligned} \text{minimiere} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{unter} \quad & y_i(w \cdot x_i - b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \end{aligned}$$

neue Datenpunkte anhand der Hyperebene klassifizieren

$$y_{n+1} = \text{sign}(w \cdot x_{n+1} - b)$$

Support Vector Machine



Support Vector Machine

Problem: nicht linear separierbare Klassen sind nicht mit einer Hyperebene zu trennen

Kernel Trick

- linearer Klassifizierer lernt nicht lineare Entscheidungsgrenze

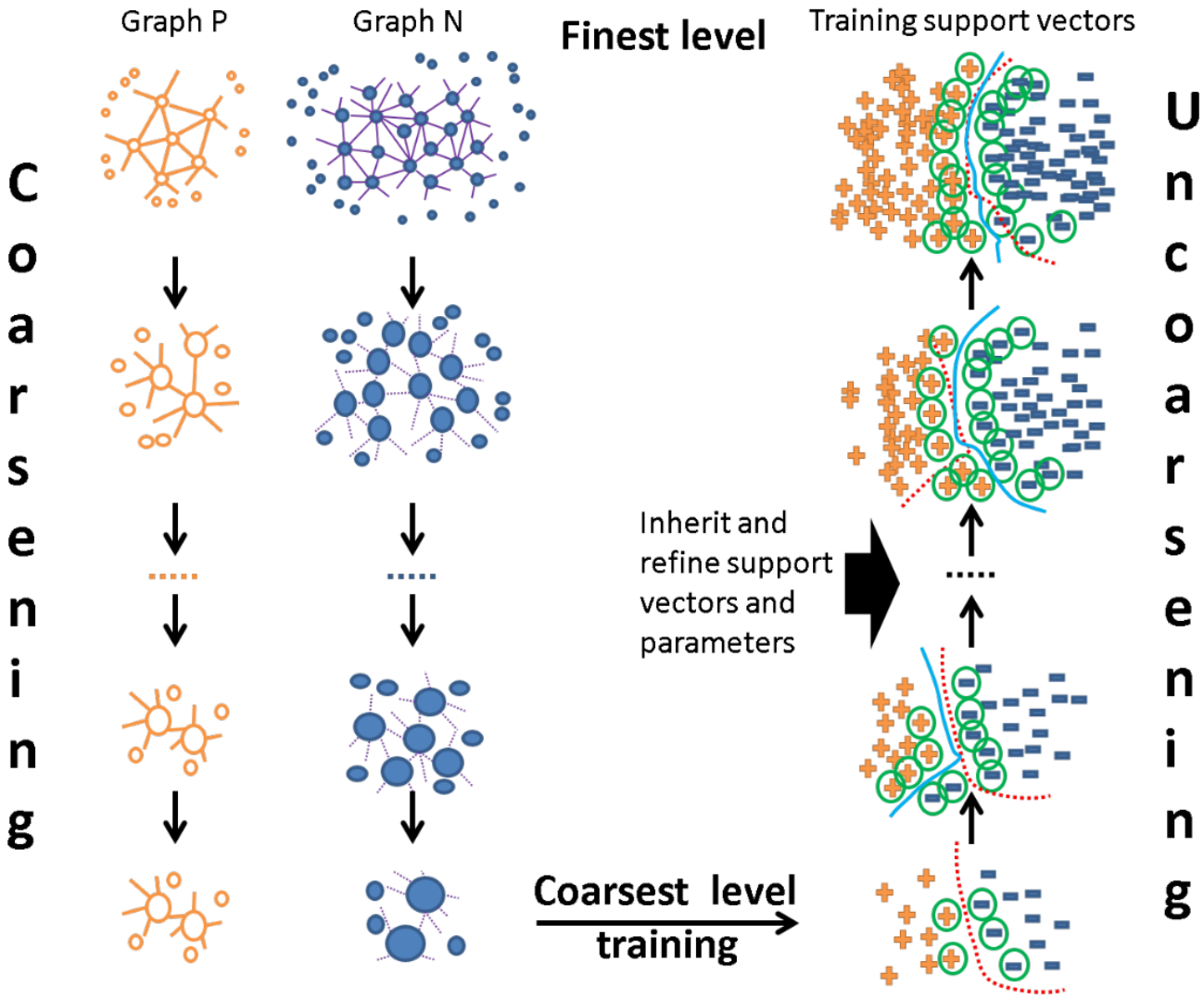
Support Vector Machine

Zeitkompletxität

- zwischen $O(n^2)$ und $O(n^3)$
- für optimale Parameter viele Modelle trainieren
- Problem mit großen Datensätzen
  (Hunderttausend bis Millionen Datenpunkte)

$\Longrightarrow$ multilevel Ansatz

Multilevel Support Vector Machine



aus *"Engineering fast multilevel support vector machines"* (2018) by Sadrfaridpour et.al.

Multilevel Support Vector Machine

mlsvm-IIS (2015)

- iterative independent set

mlsvm-AMG (2016)

- algebraic multigrid

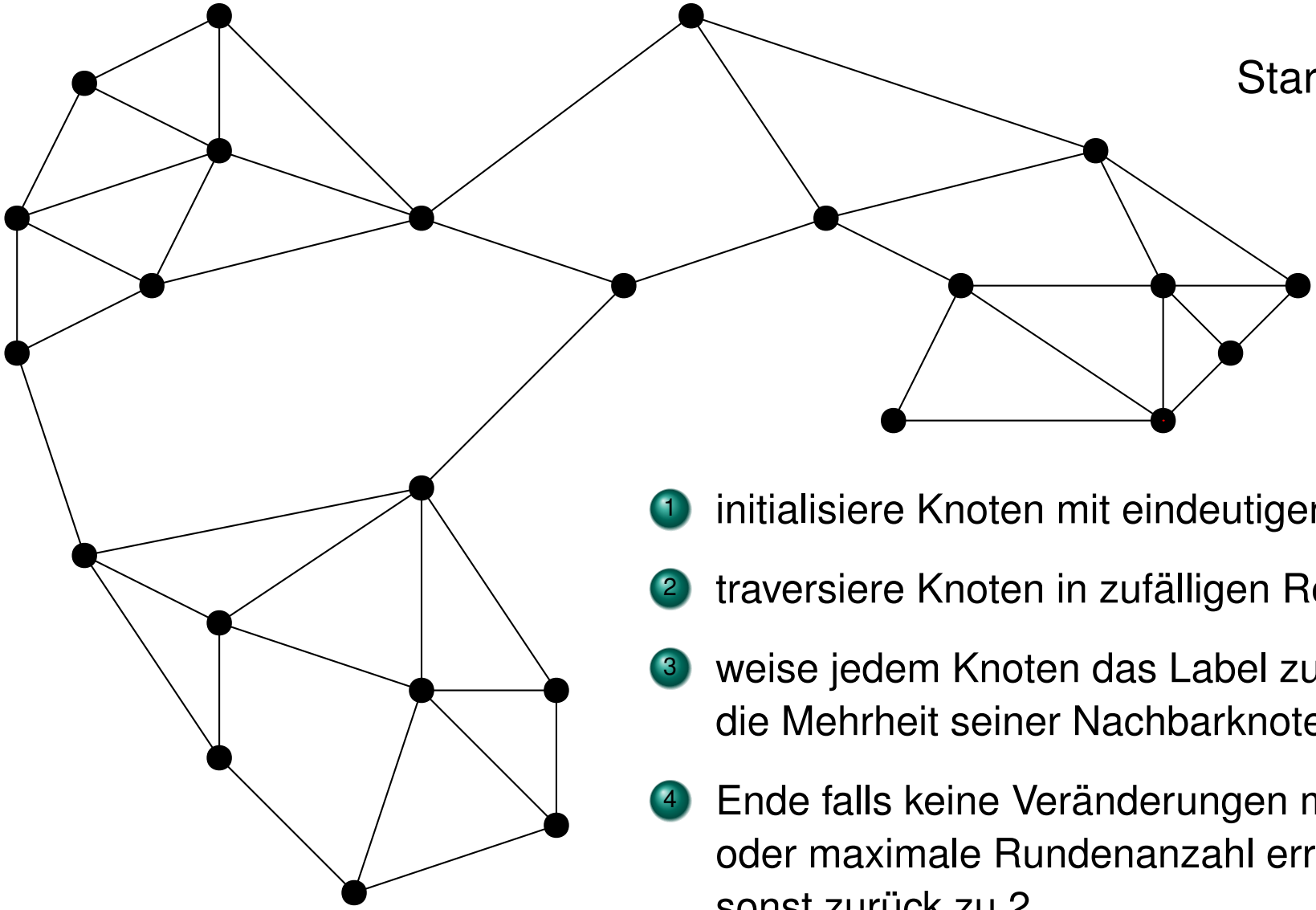
“Engineering fast multilevel support vector machines” (2018)

- Verfeinerung des multilevel Prozesses

Label Propagation Algorithmus (2007)

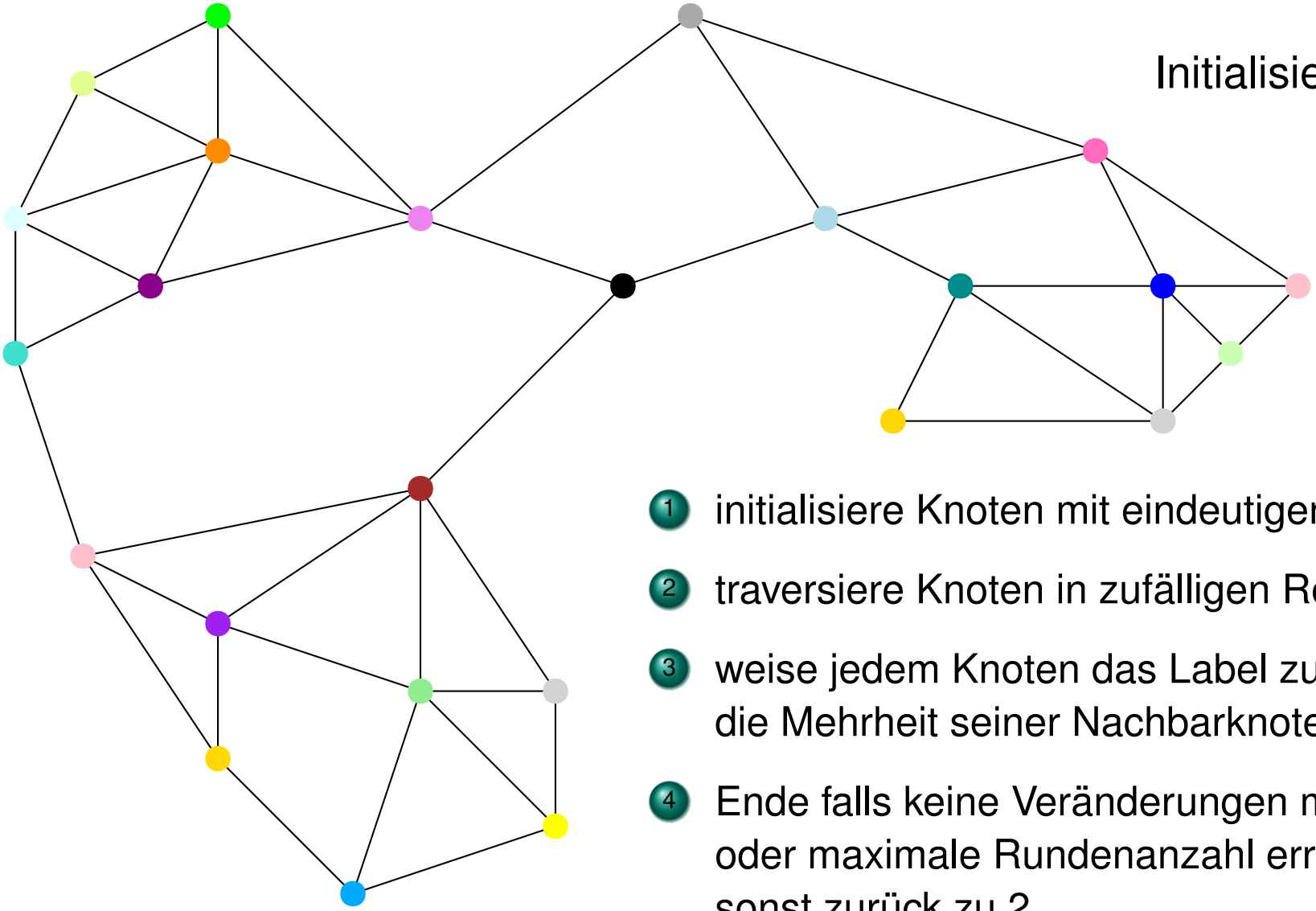
fast linearzeit Algorithmus zur Community-Erkennung

Label Propagation Algorithmus (2007)



1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

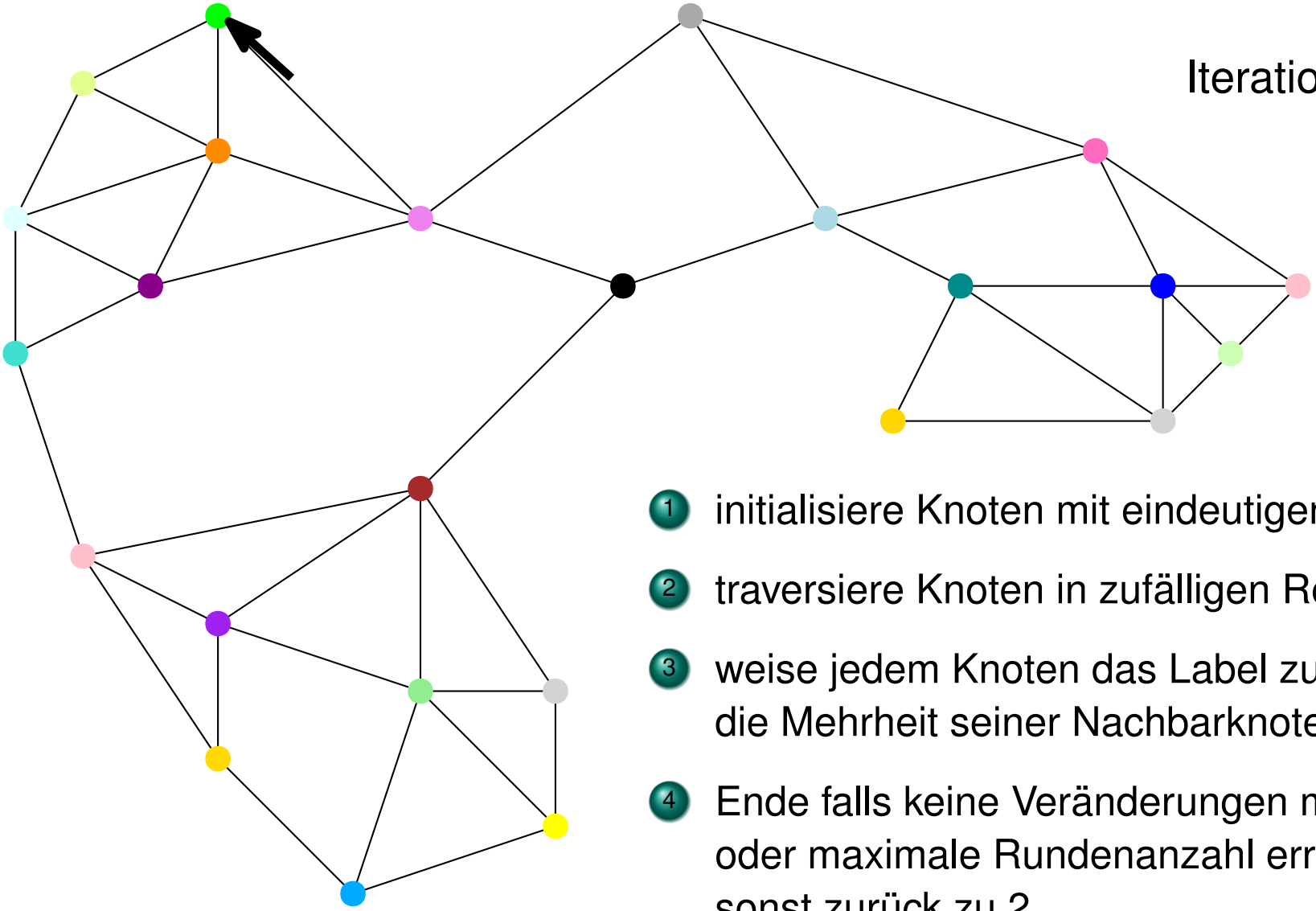
Label Propagation Algorithmus (2007)



Initialisierung

1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

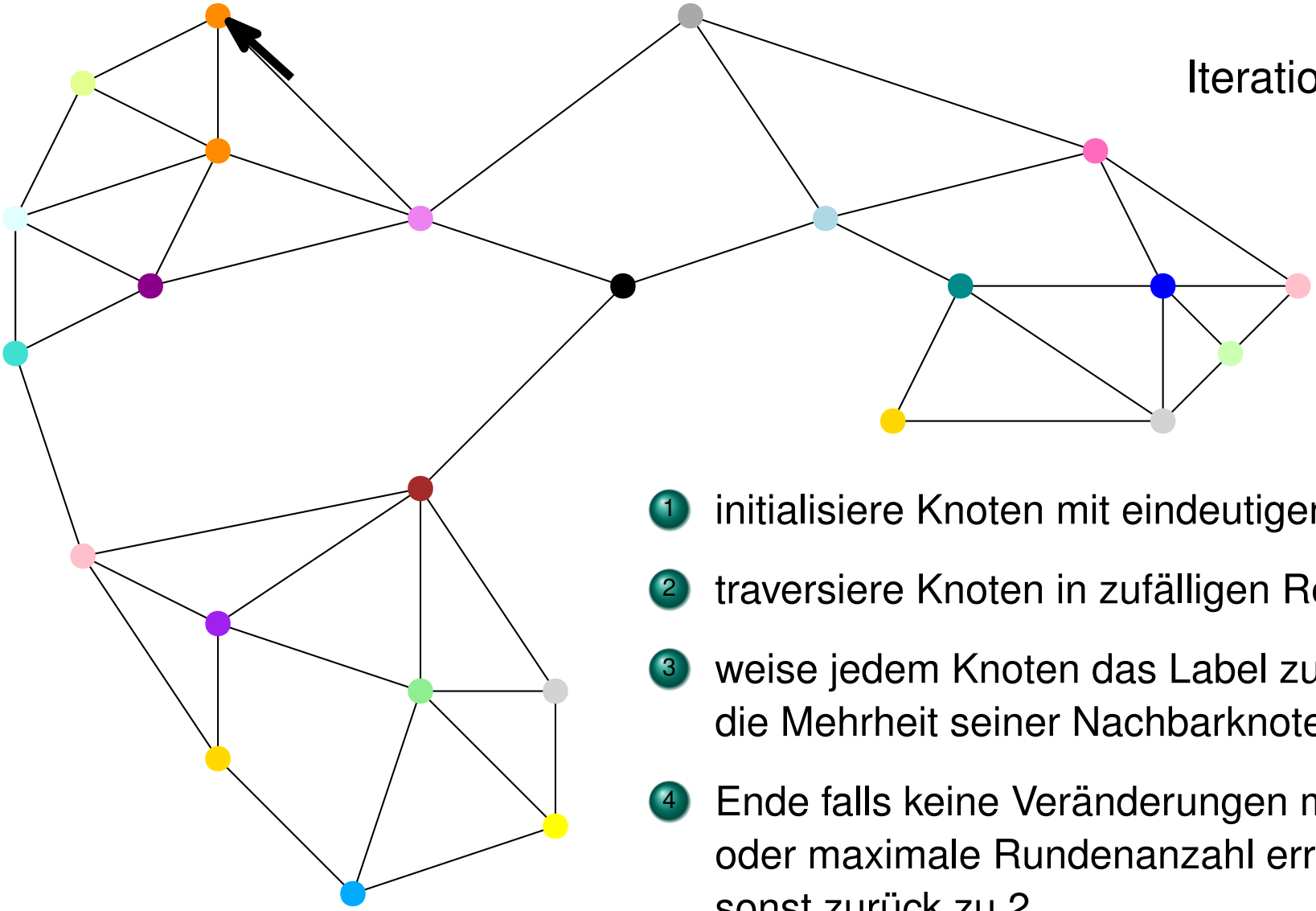
Label Propagation Algorithmus (2007)



Iteration 1

1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

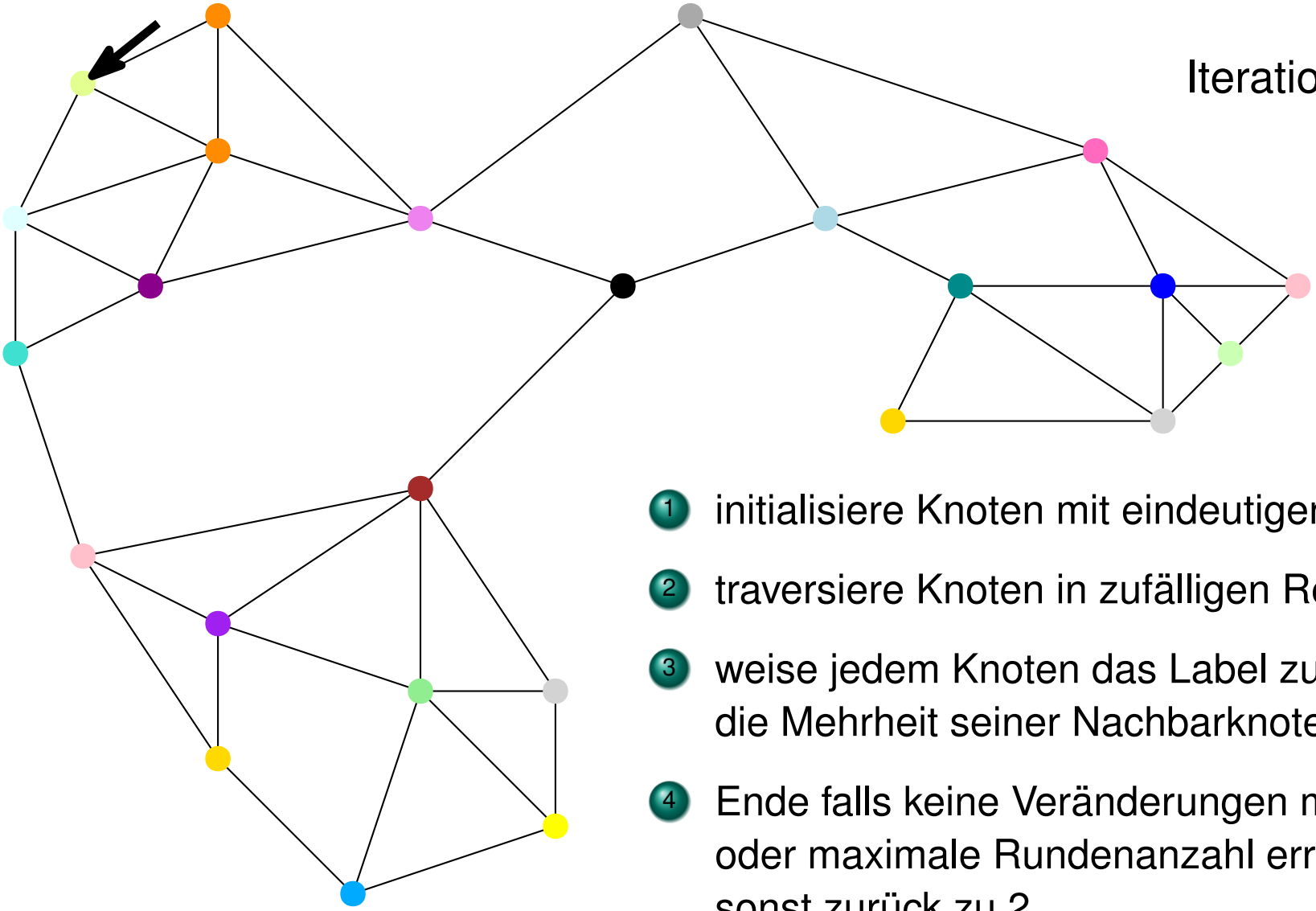
Label Propagation Algorithmus (2007)



Iteration 1

1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

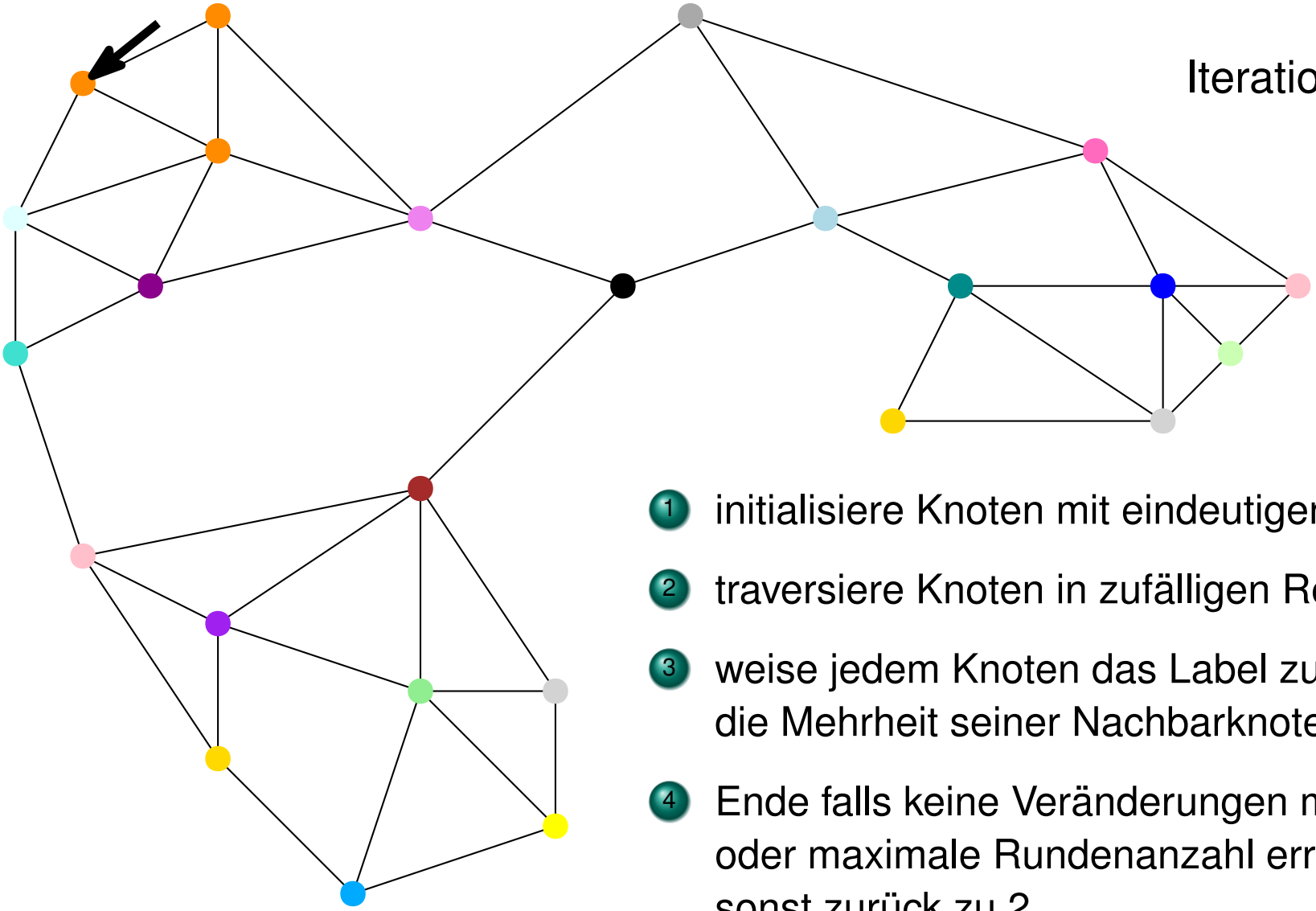
Label Propagation Algorithmus (2007)



Iteration 1

1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

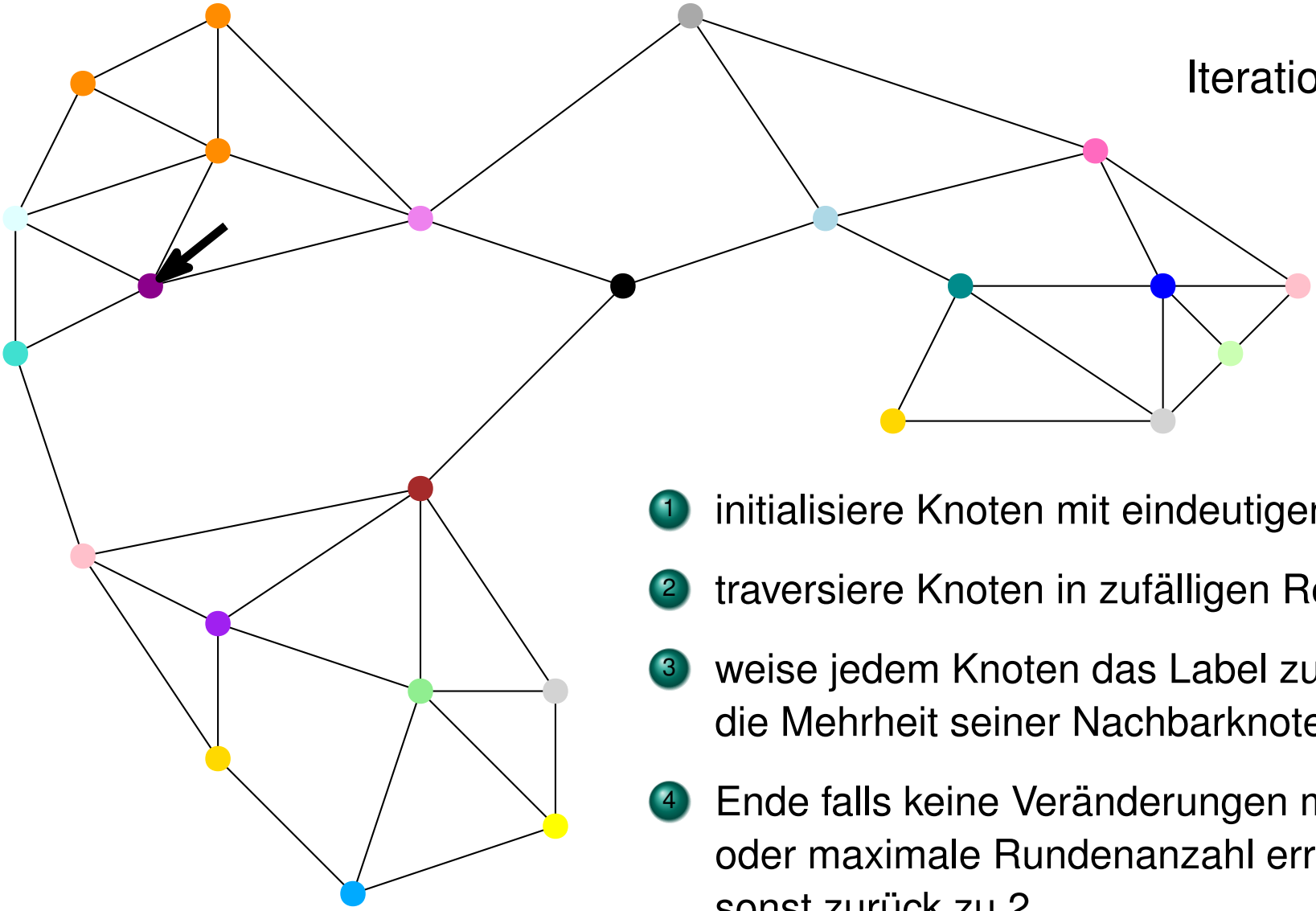
Label Propagation Algorithmus (2007)



Iteration 1

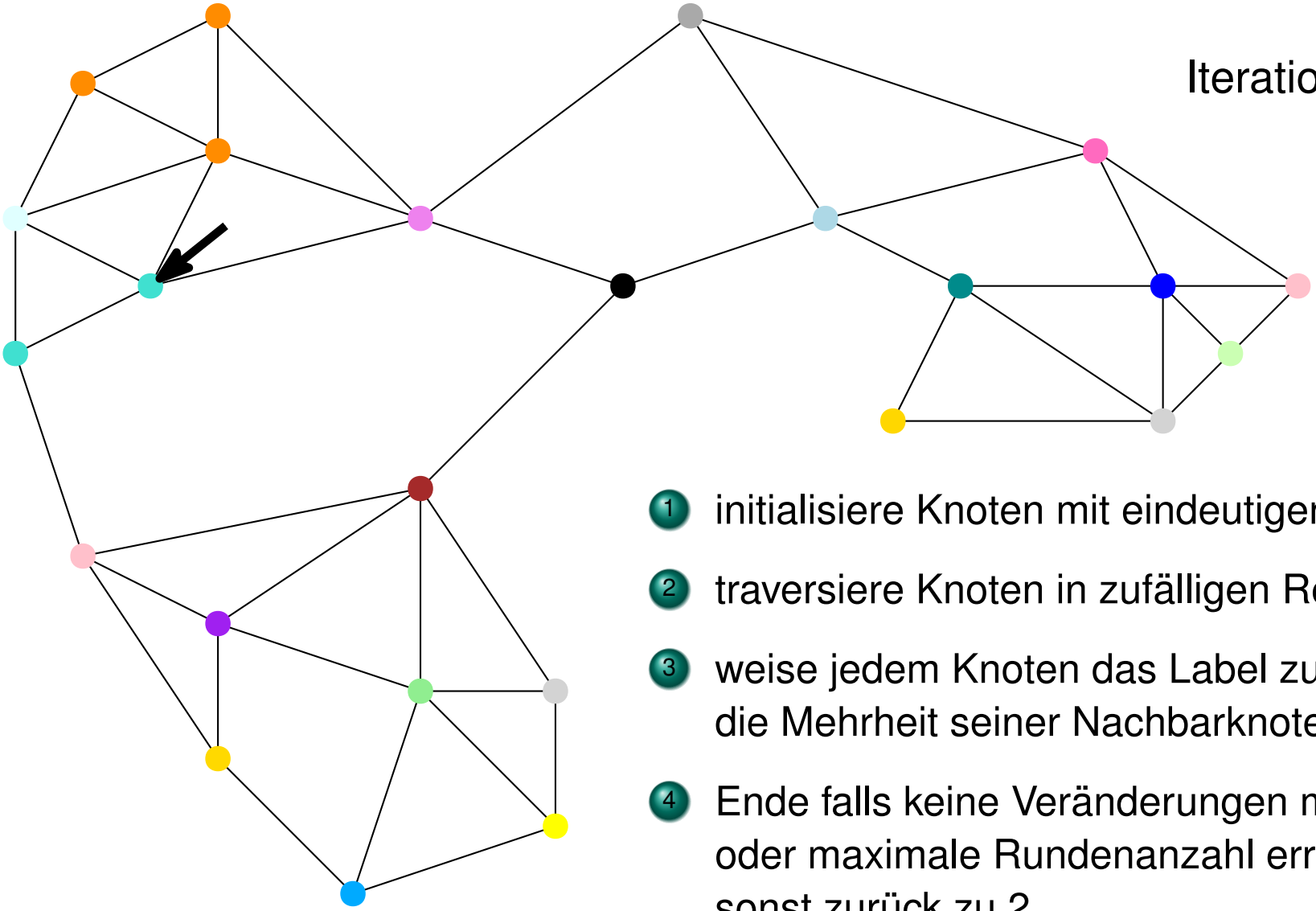
1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

Label Propagation Algorithmus (2007)



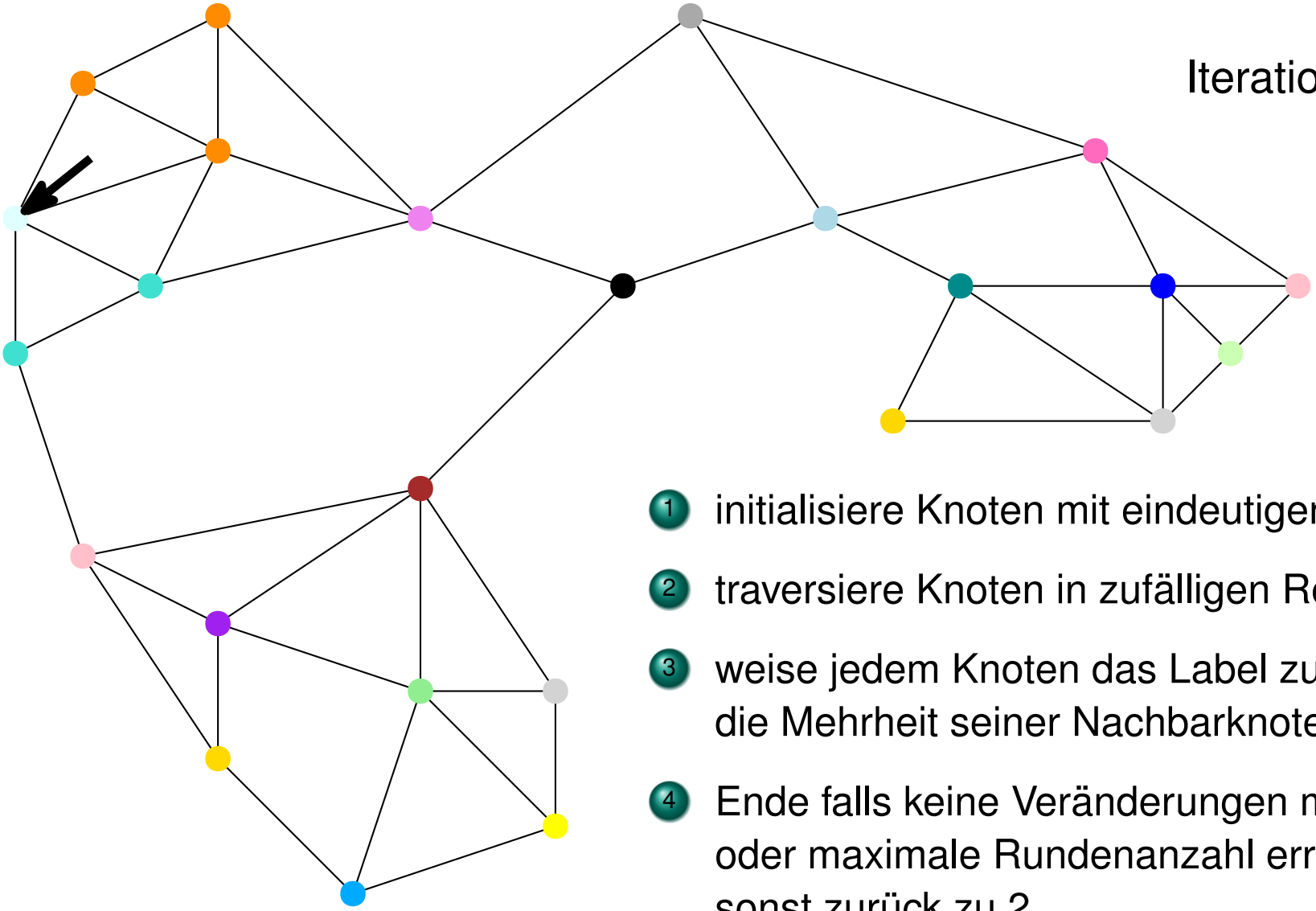
1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

Label Propagation Algorithmus (2007)



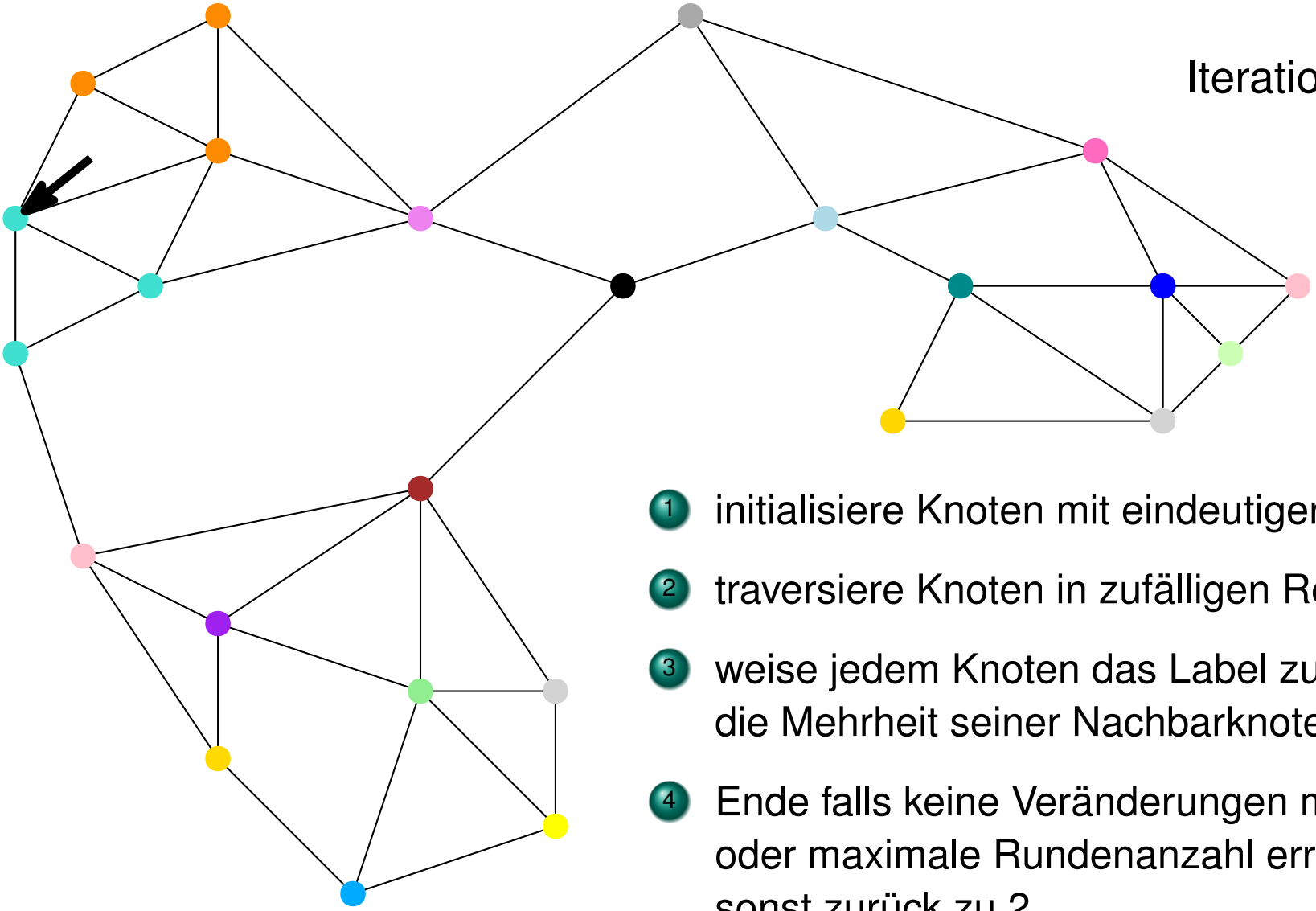
1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

Label Propagation Algorithmus (2007)



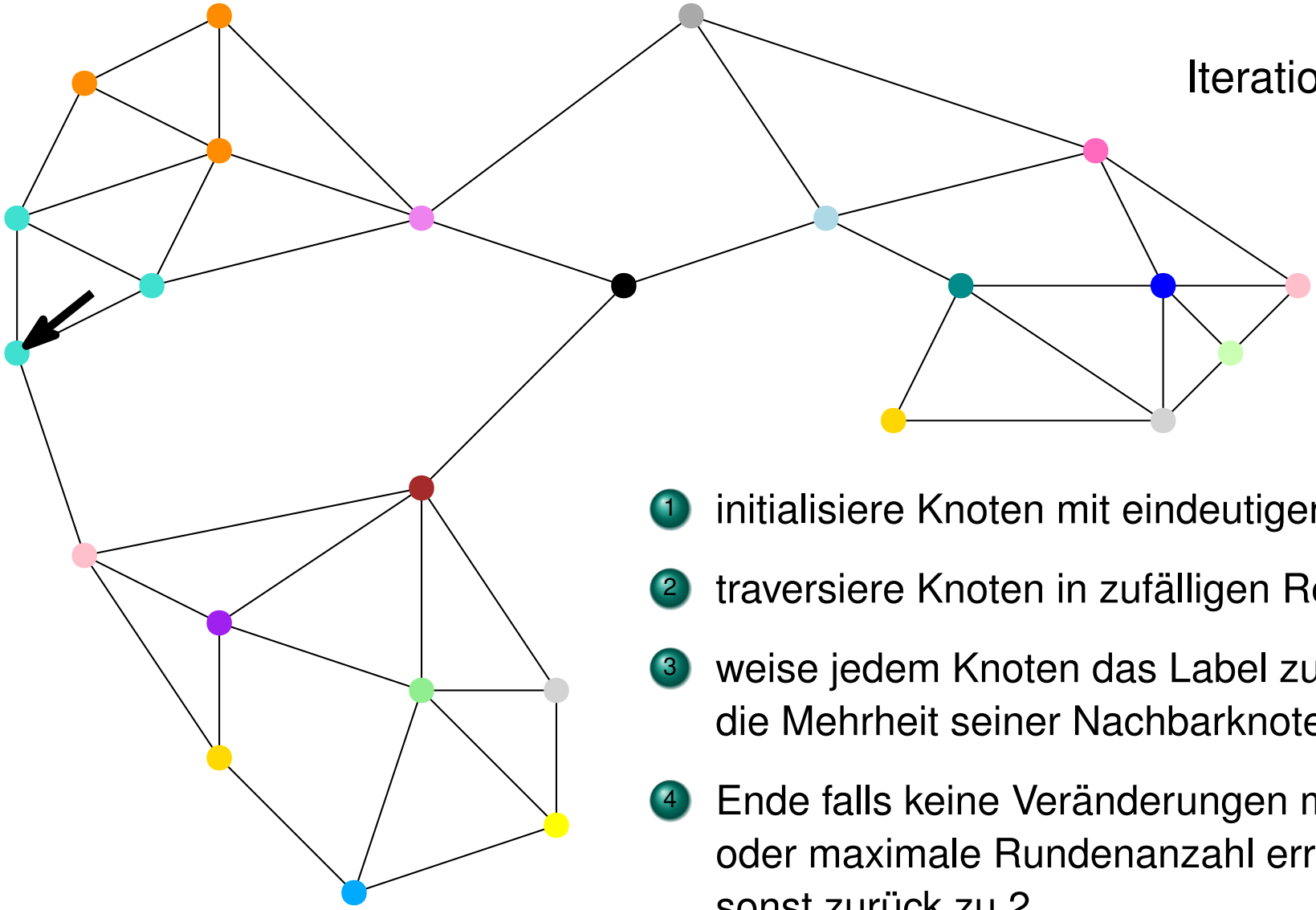
1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

Label Propagation Algorithmus (2007)



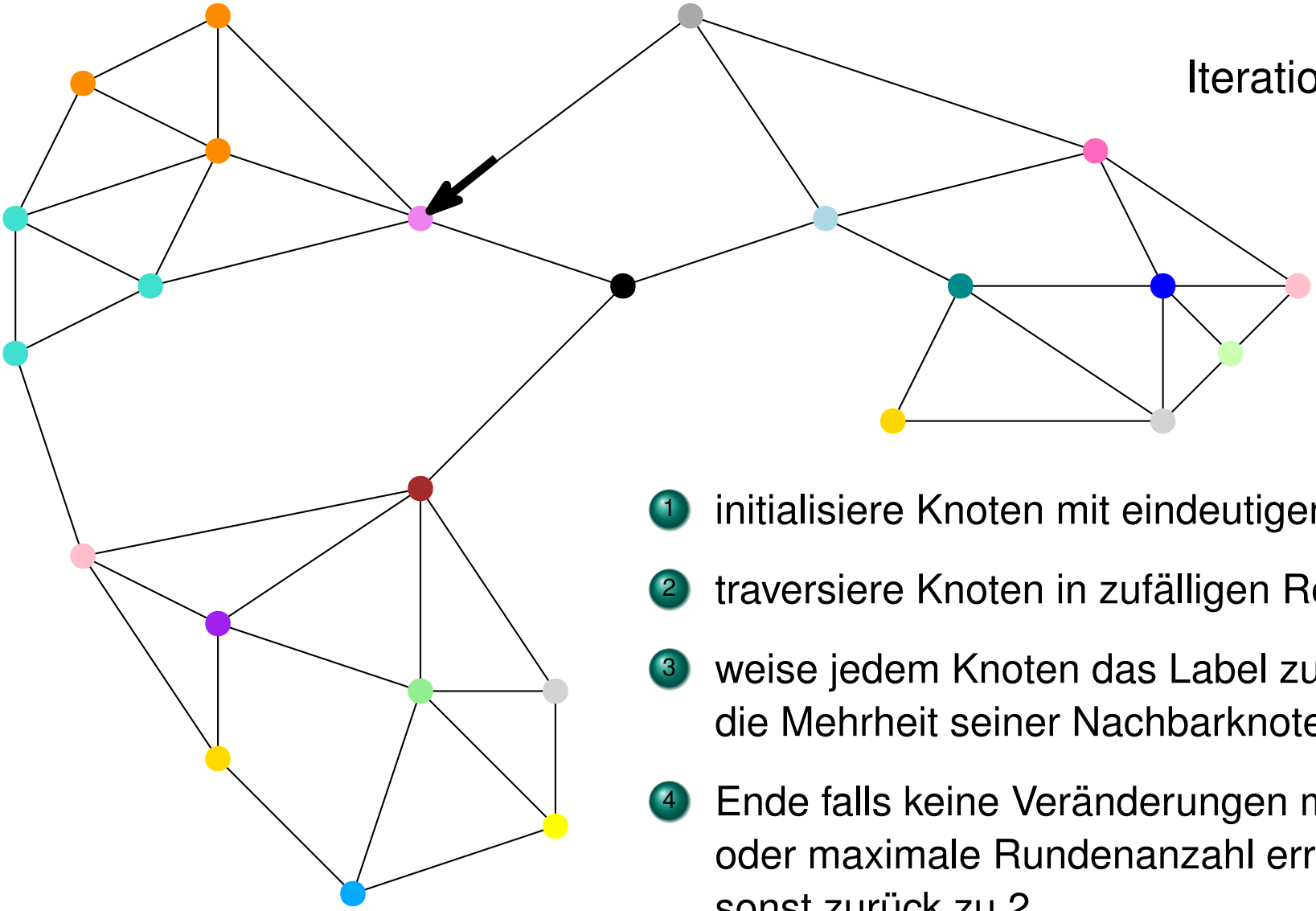
1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

Label Propagation Algorithmus (2007)



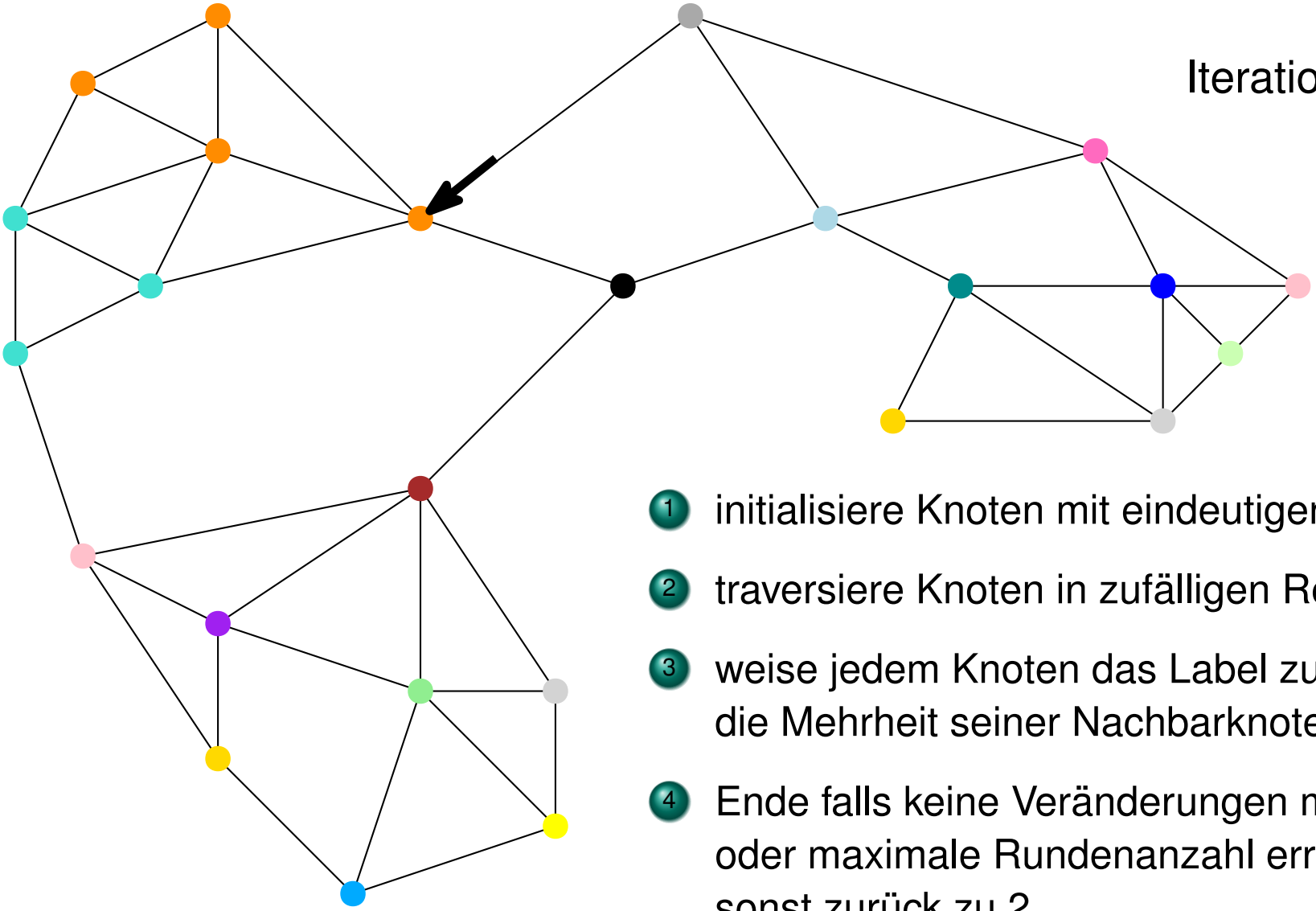
1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

Label Propagation Algorithmus (2007)



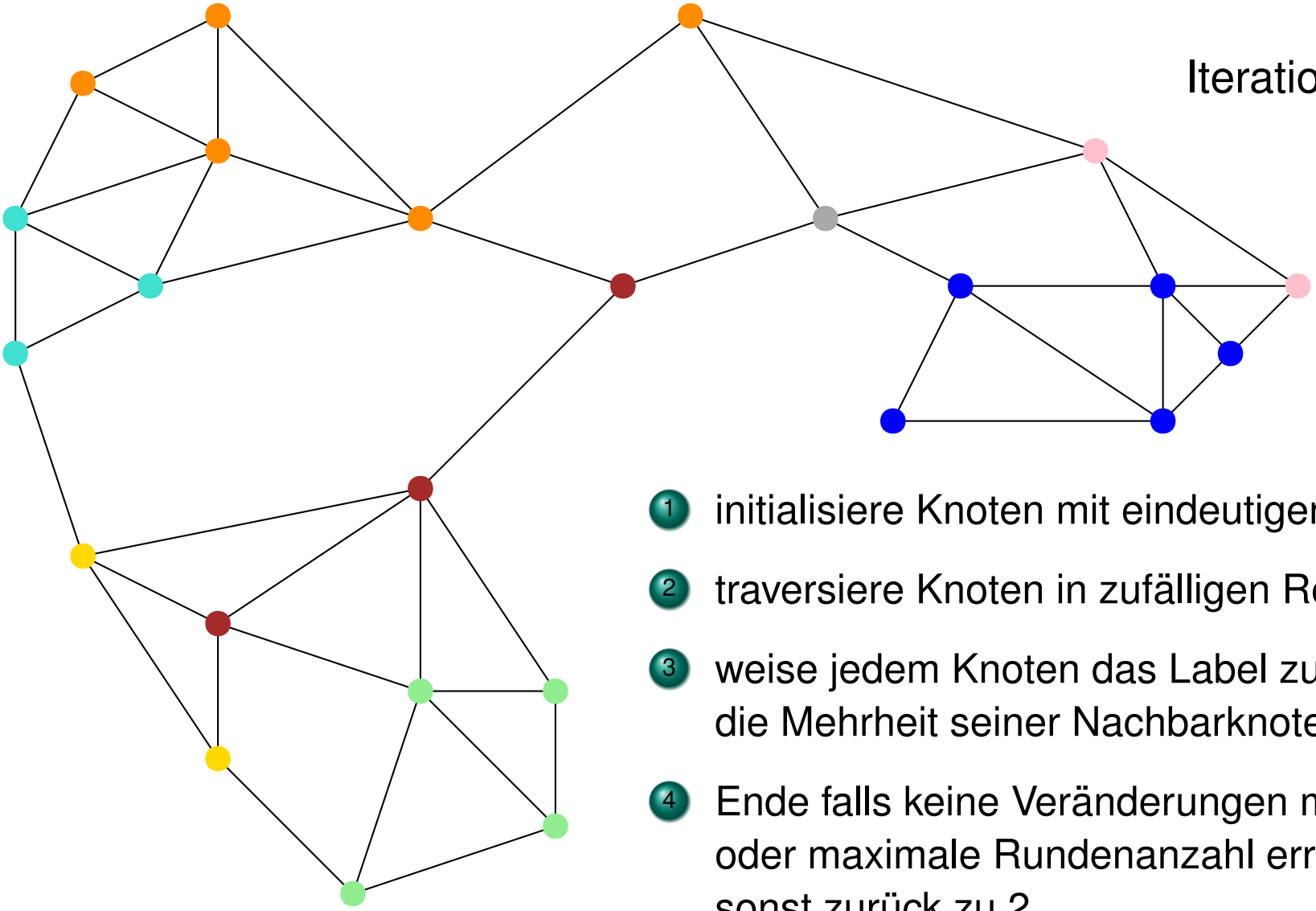
1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

Label Propagation Algorithmus (2007)



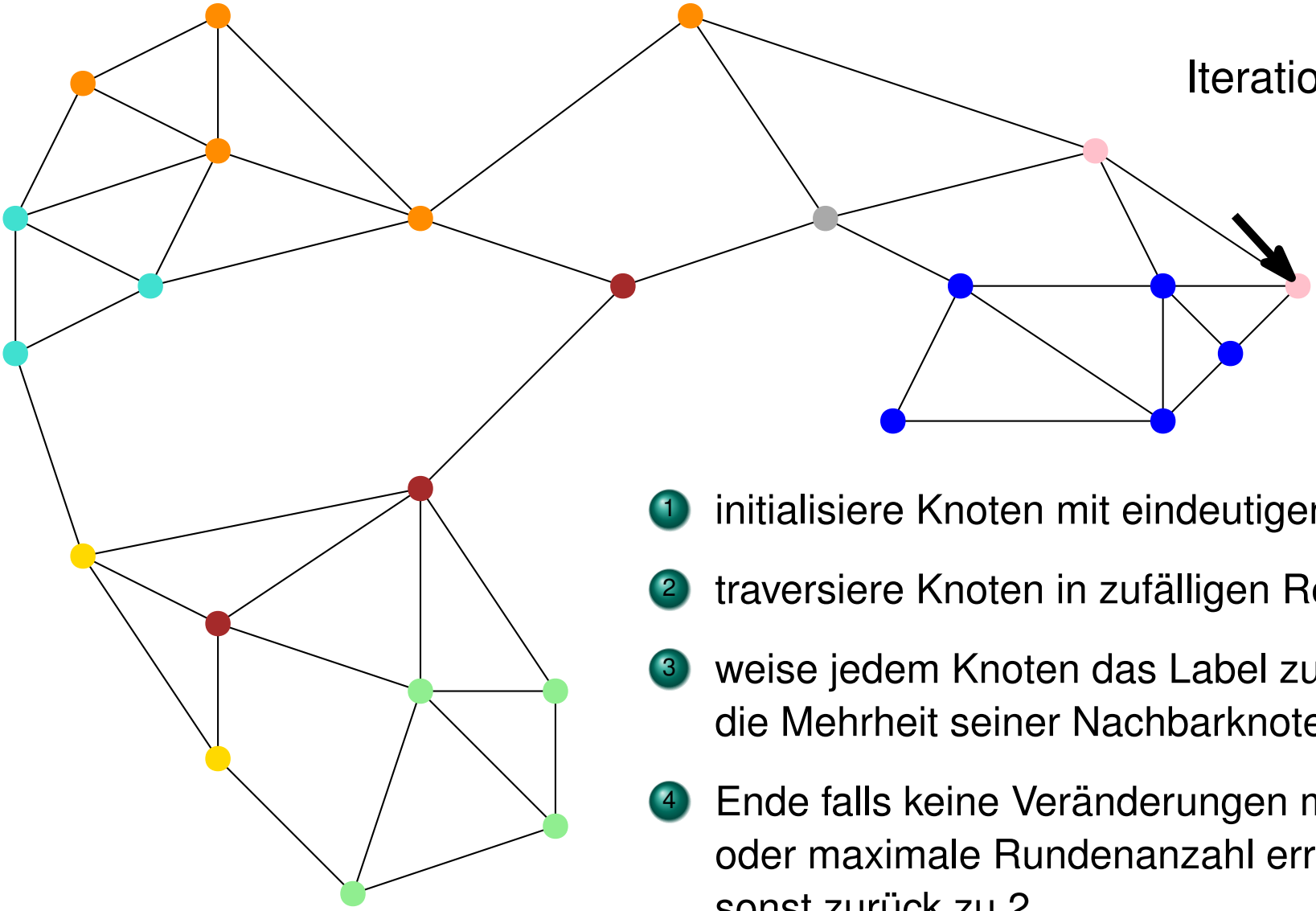
1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

Label Propagation Algorithmus (2007)



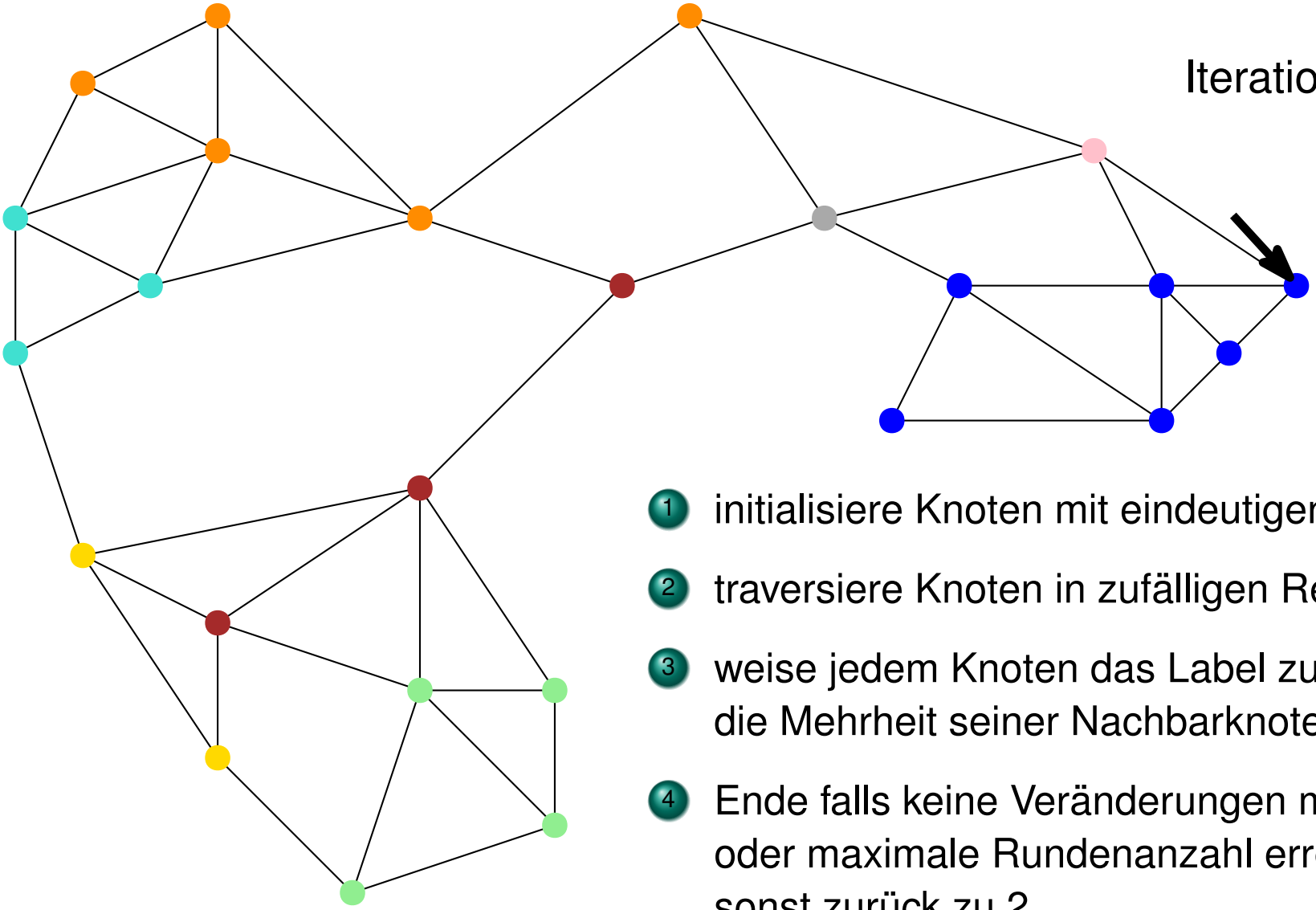
1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

Label Propagation Algorithmus (2007)



1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

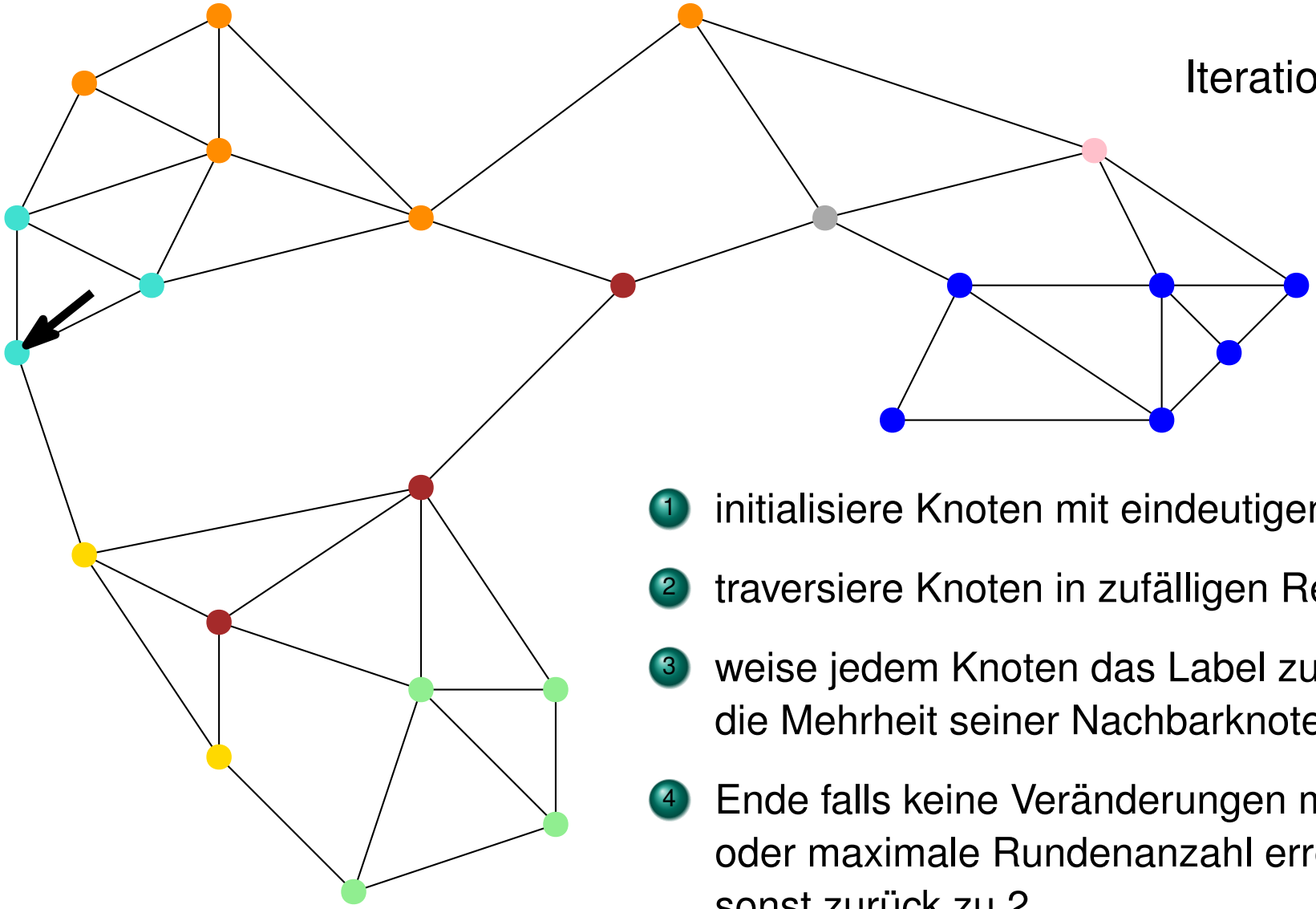
Label Propagation Algorithmus (2007)



Iteration 2

1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

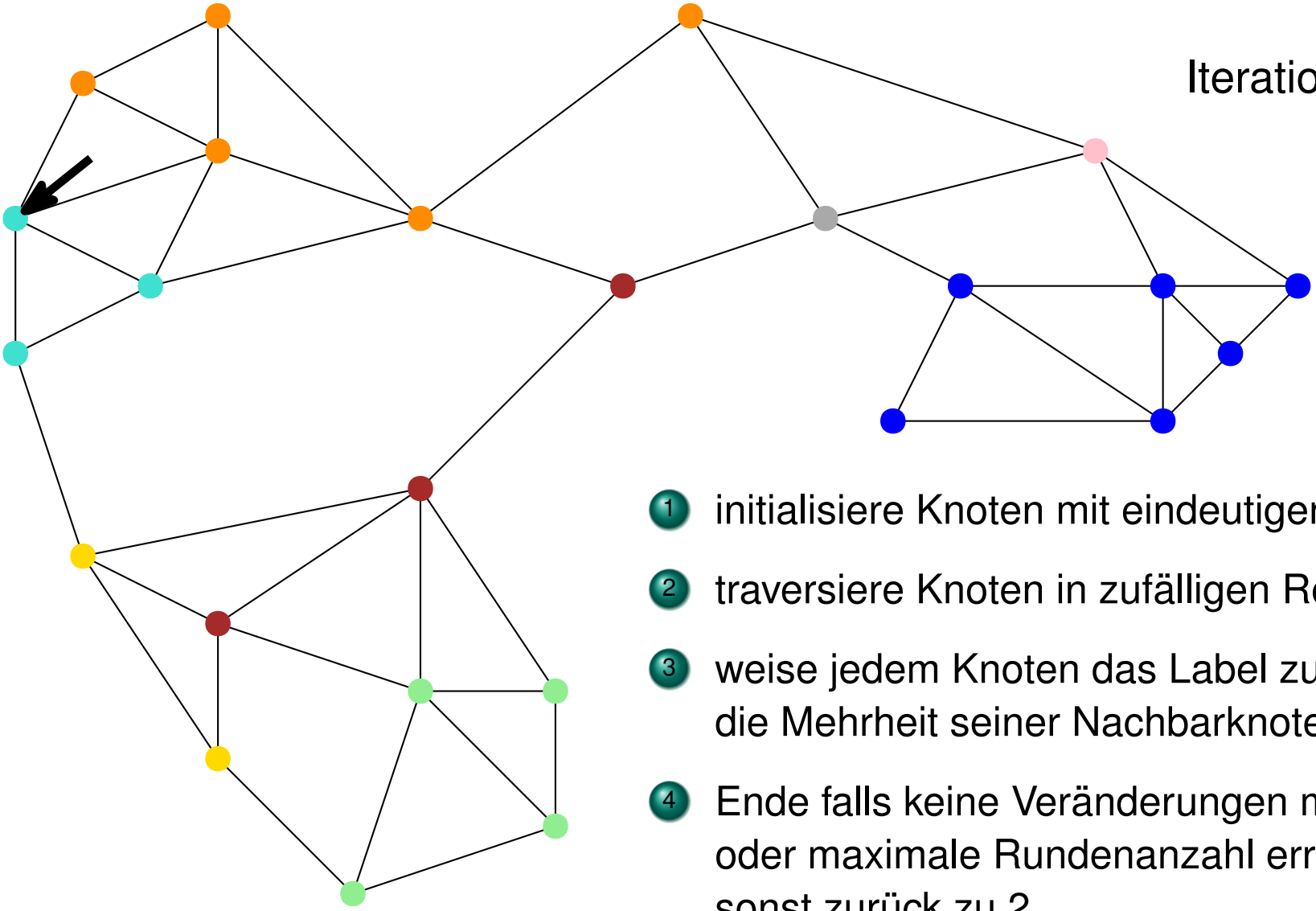
Label Propagation Algorithmus (2007)



Iteration 2

1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

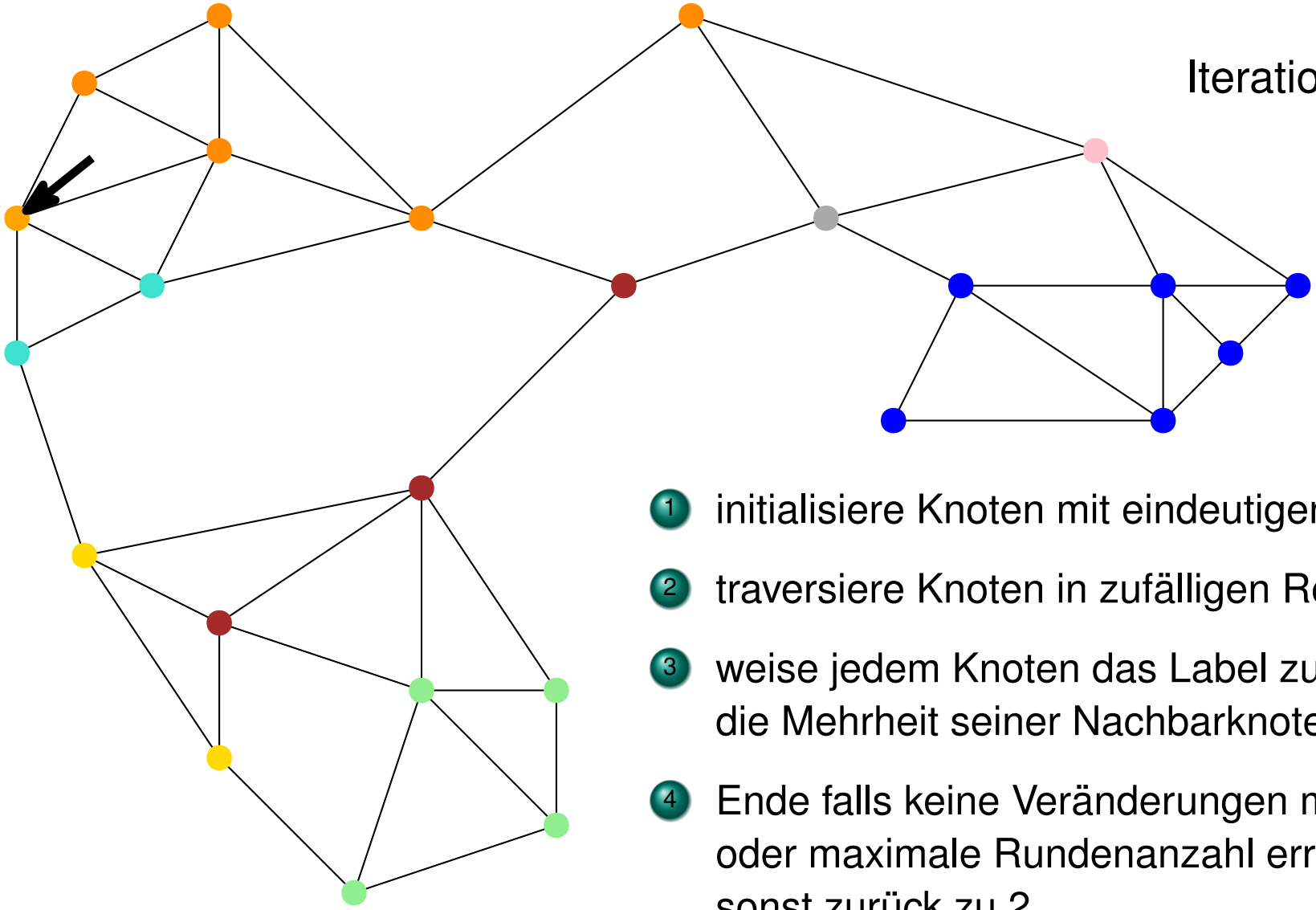
Label Propagation Algorithmus (2007)



Iteration 2

1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

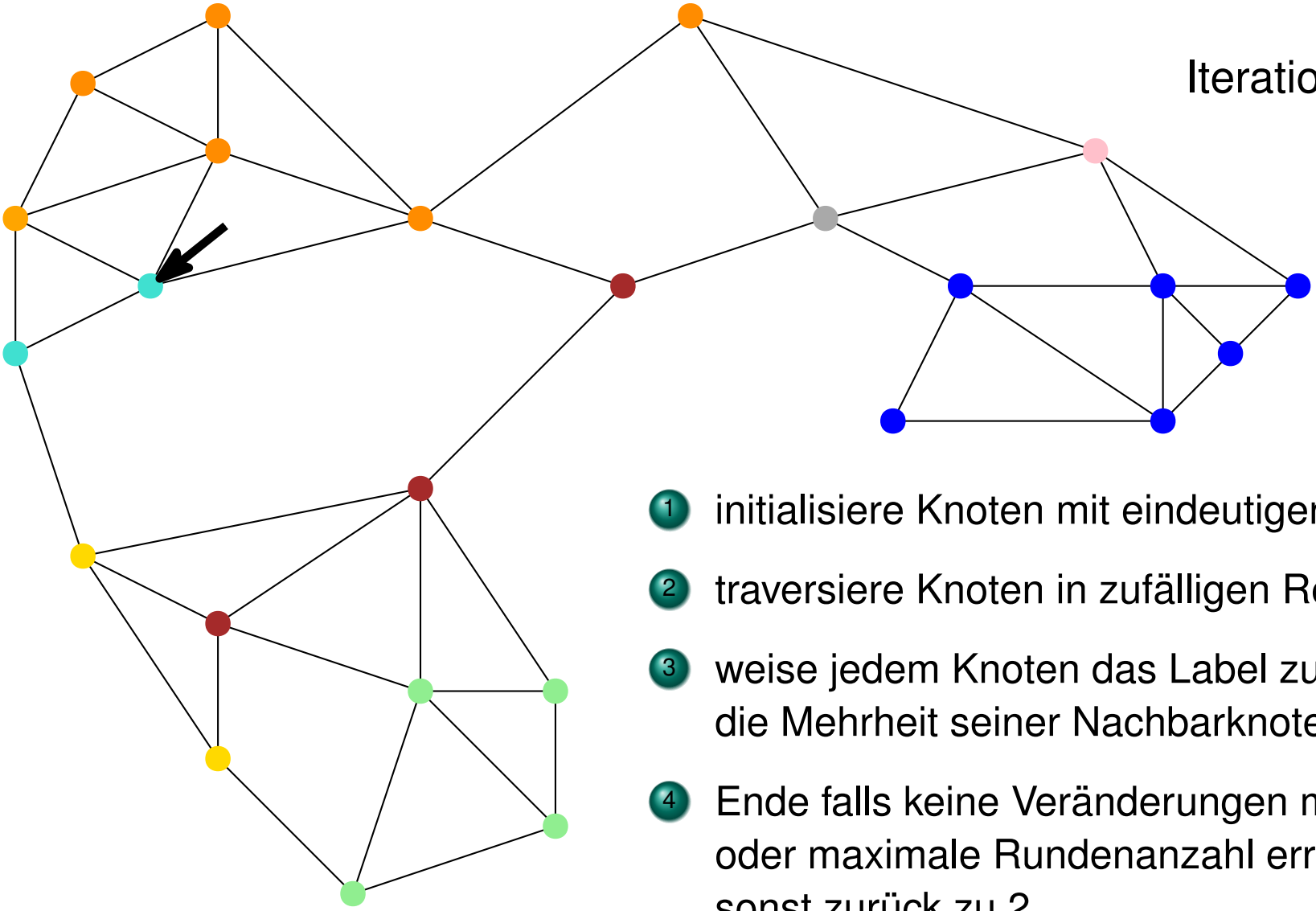
Label Propagation Algorithmus (2007)



Iteration 2

1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

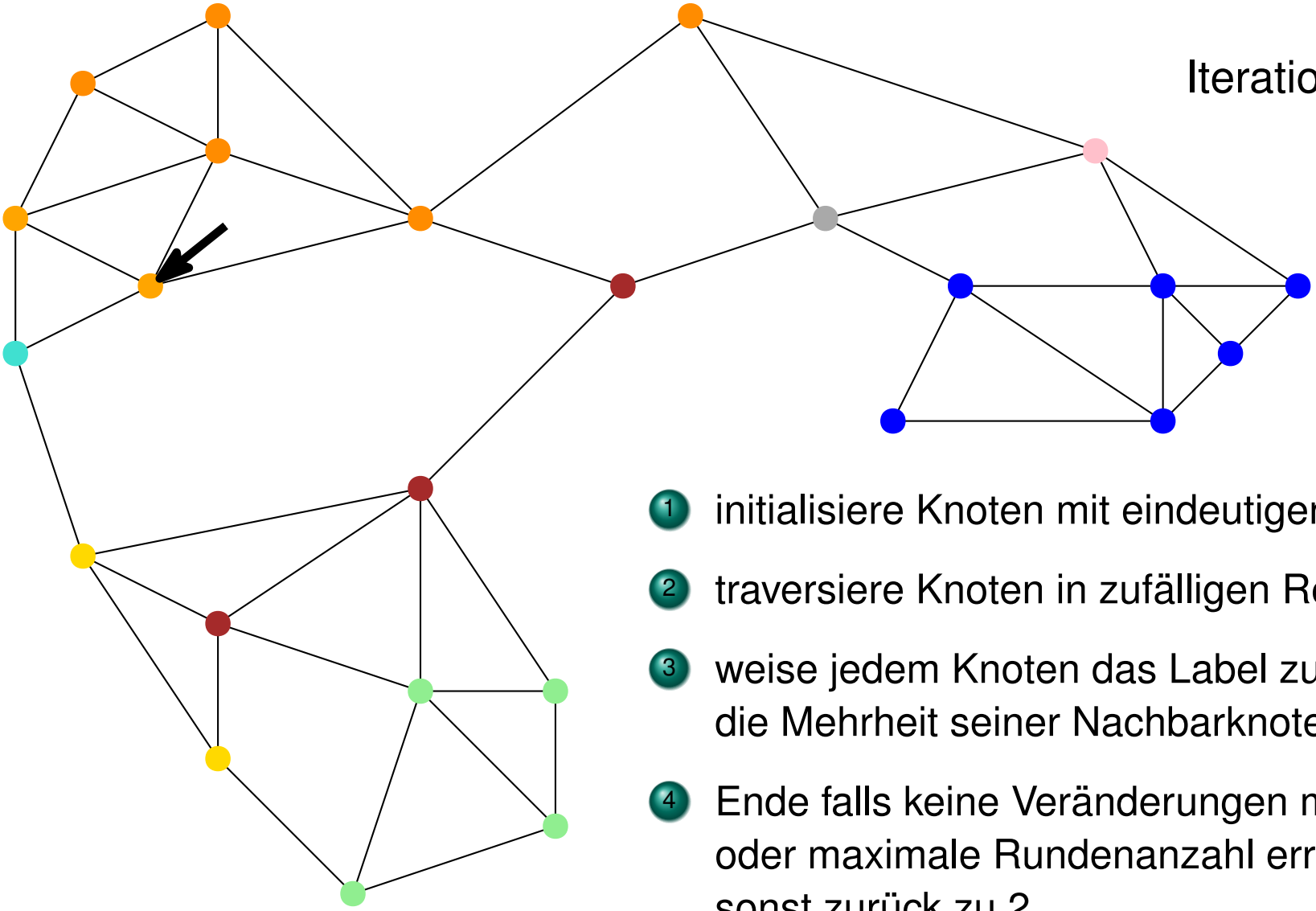
Label Propagation Algorithmus (2007)



Iteration 2

1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

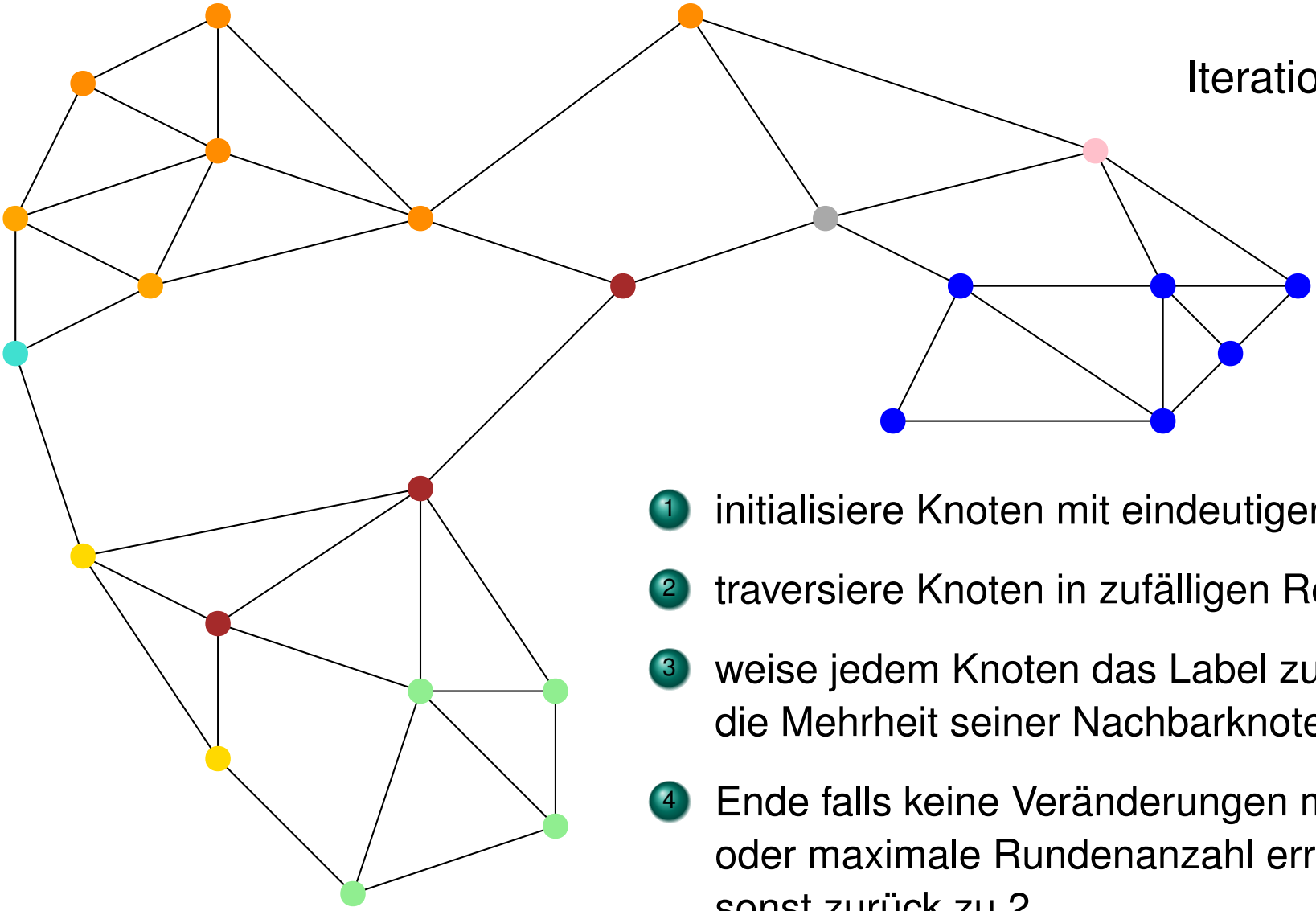
Label Propagation Algorithmus (2007)



Iteration 2

1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

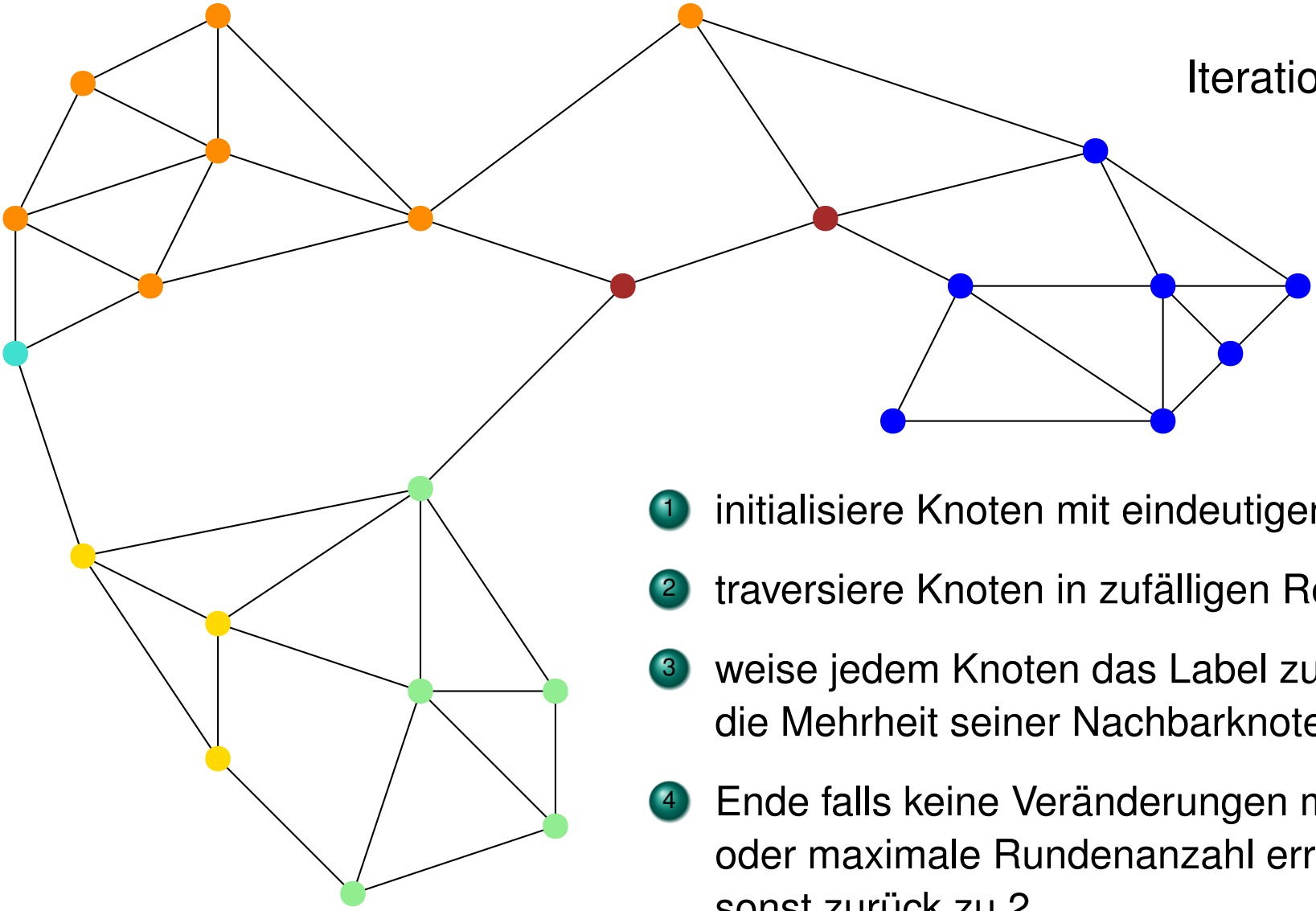
Label Propagation Algorithmus (2007)



Iteration 2

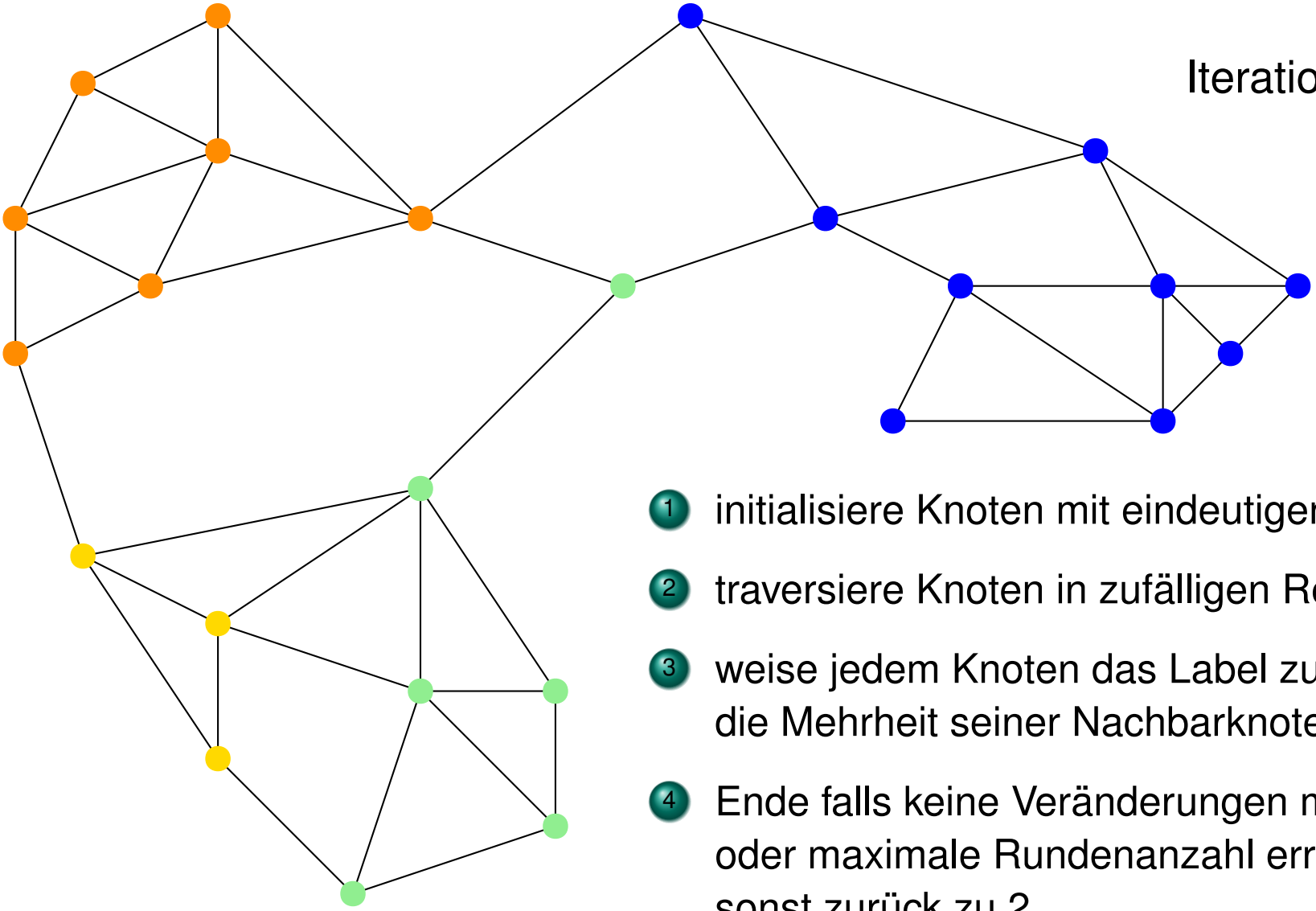
1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

Label Propagation Algorithmus (2007)



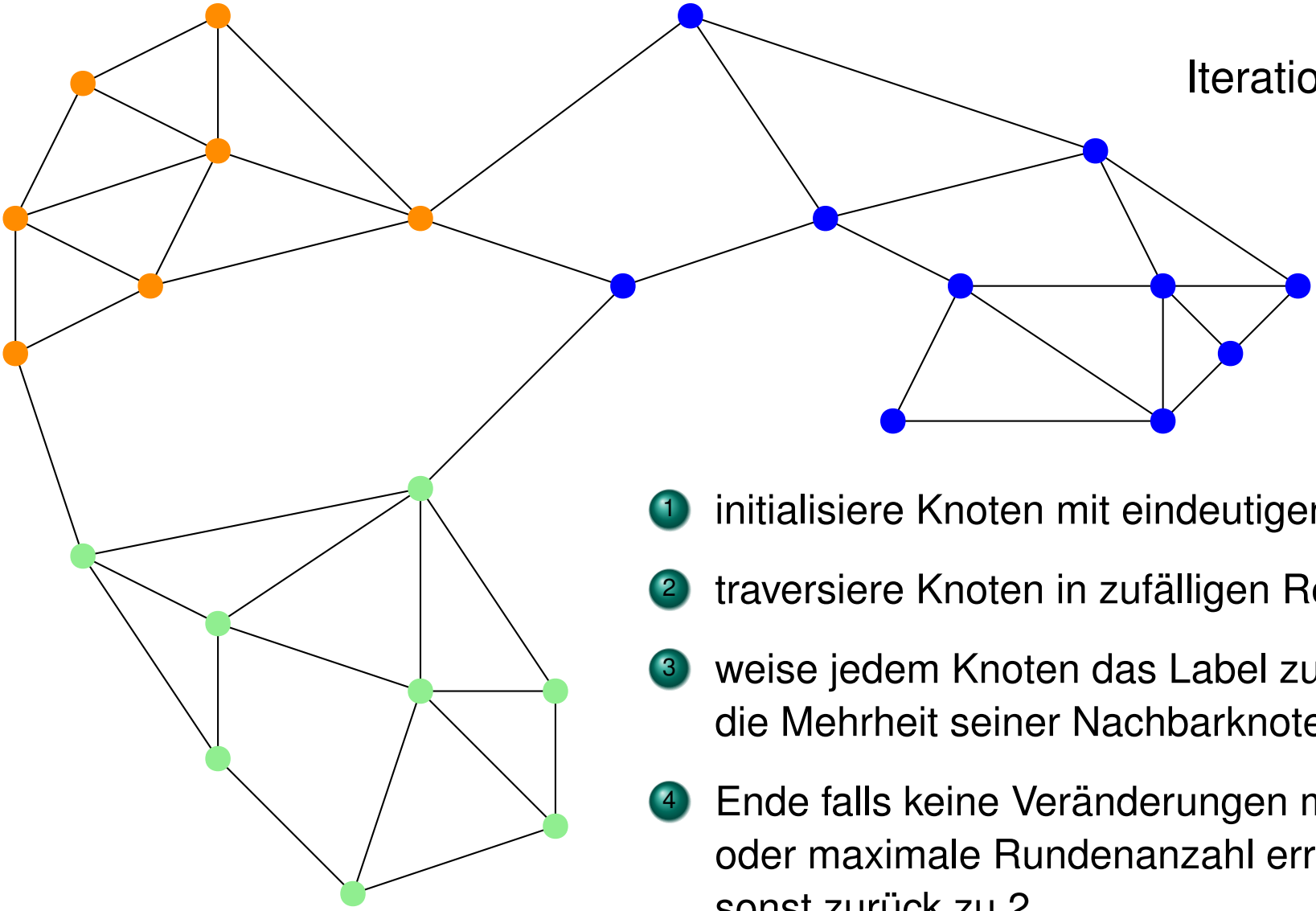
1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

Label Propagation Algorithmus (2007)



1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

Label Propagation Algorithmus (2007)



Iteration 4

1. initialisiere Knoten mit eindeutigen Labeln
2. traversiere Knoten in zufälligen Reihenfolge
3. weise jedem Knoten das Label zu, welches die Mehrheit seiner Nachbarknoten besitzt
4. Ende falls keine Veränderungen mehr oder maximale Rundenanzahl erreicht, sonst zurück zu 2

KaMLSVM

Algorithm 1: Überblick

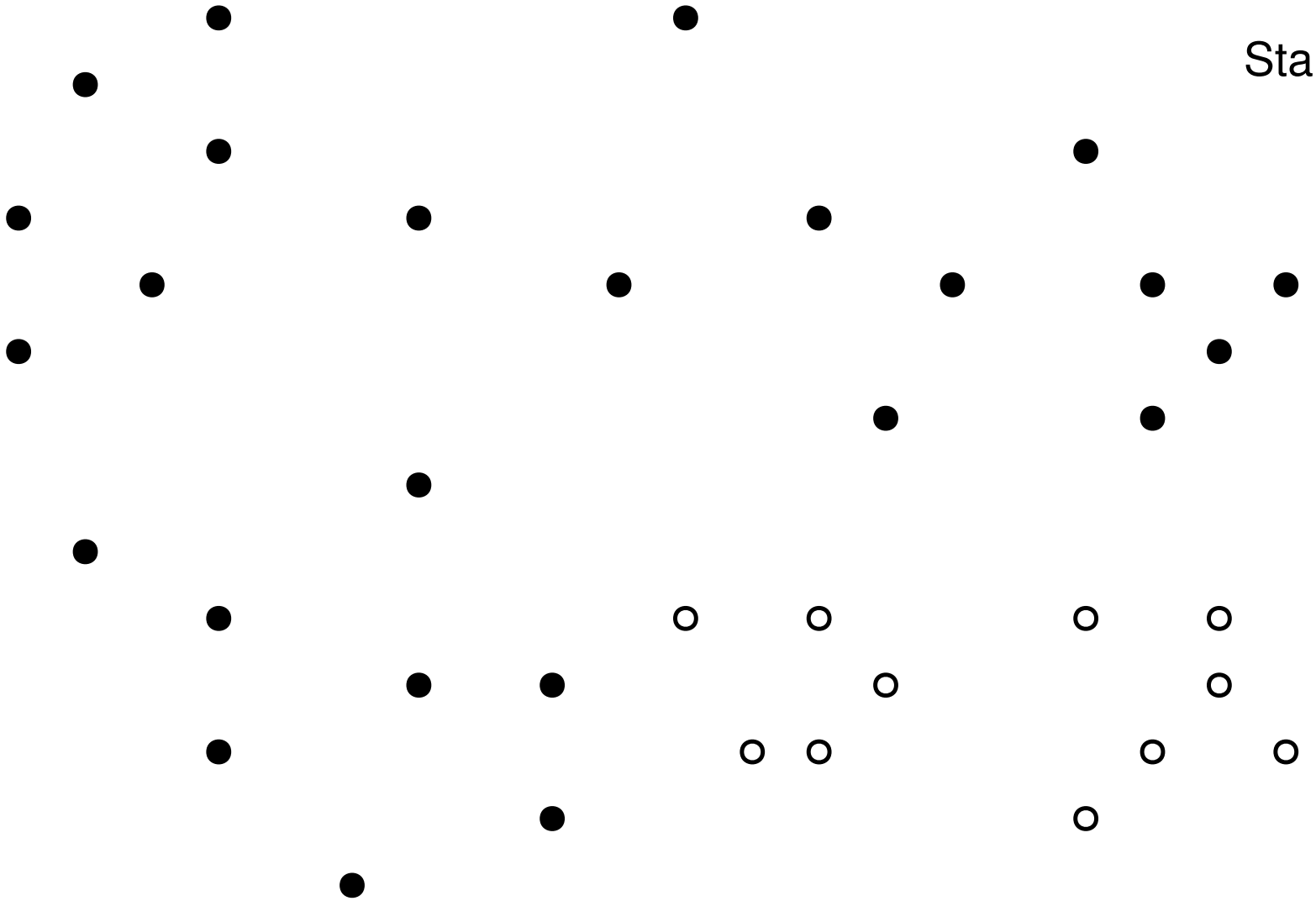
```
1  preprocess data
2  build k-fold instances
3  foreach k-fold instance do
4      use a fraction of the training data as validation data
5      Coarsening
6      Initial Training
7      while levels in the hierarchies do
8          Refinement
9      use the best trained model of all levels as final model
10     evaluate the final model with the test data
11 average the results of the k-folds
```

Coarsening

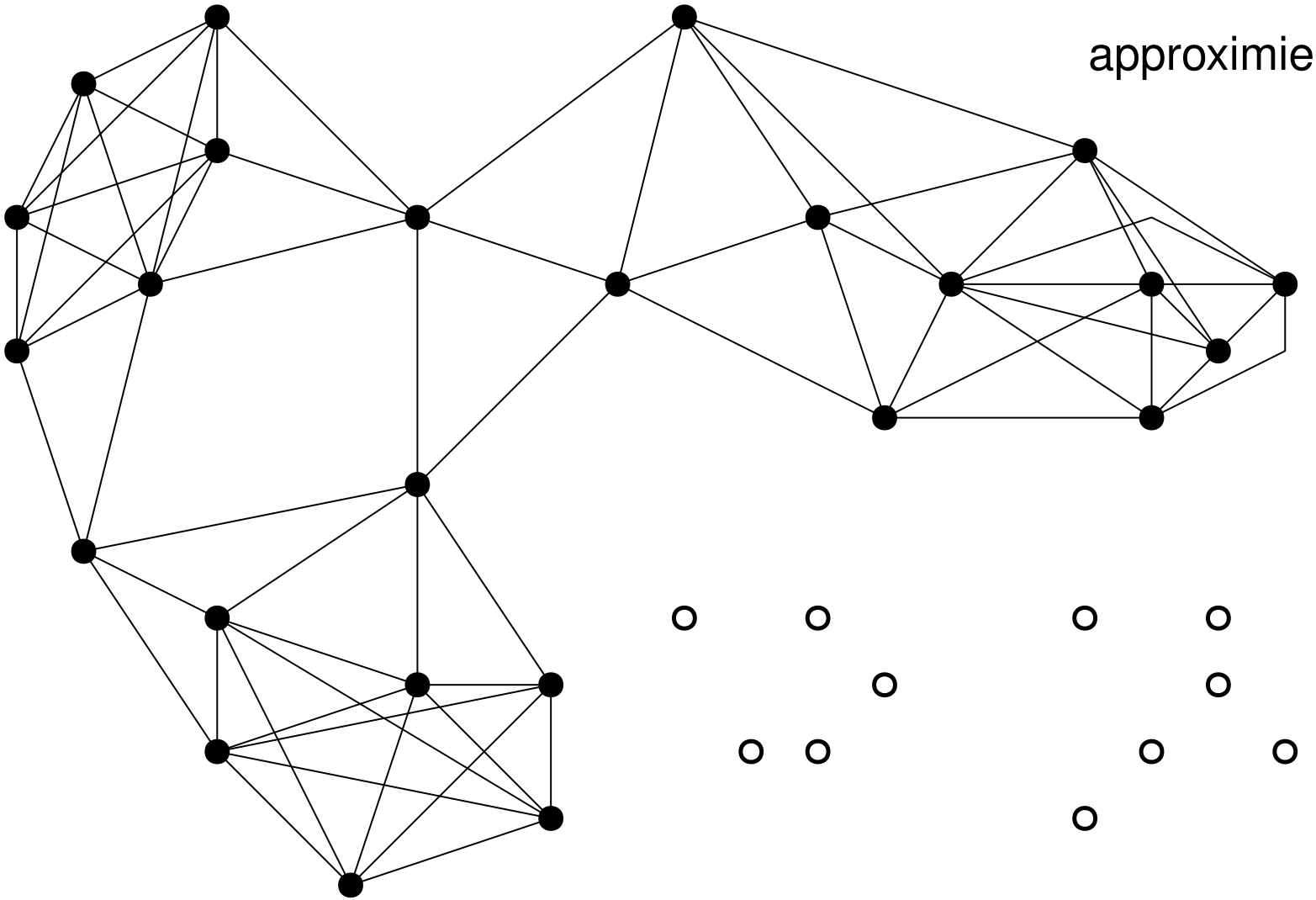
positive und negative Klasse separat verarbeiten

1. approximierten k-nearest neighbor Graph aufbauen (FLANN)
2. Communities finden via Label Propagation
3. Communities contracten $\Longrightarrow$ Graph vergröbern
4. neues Problem in der Hierarchie speichern
5. solange Graph noch nicht klein genug zurück zu 2

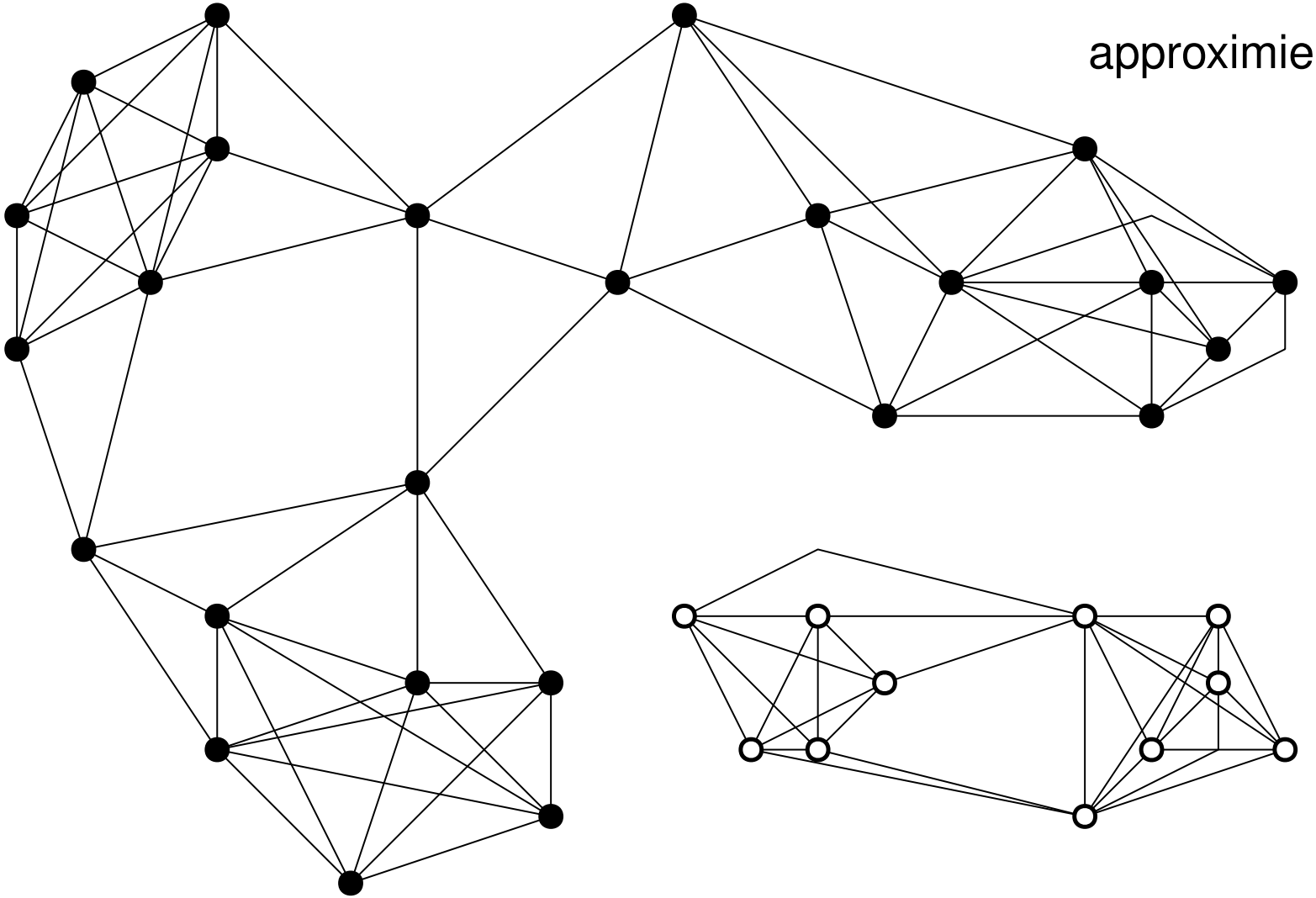
Coarsening - Beispiel



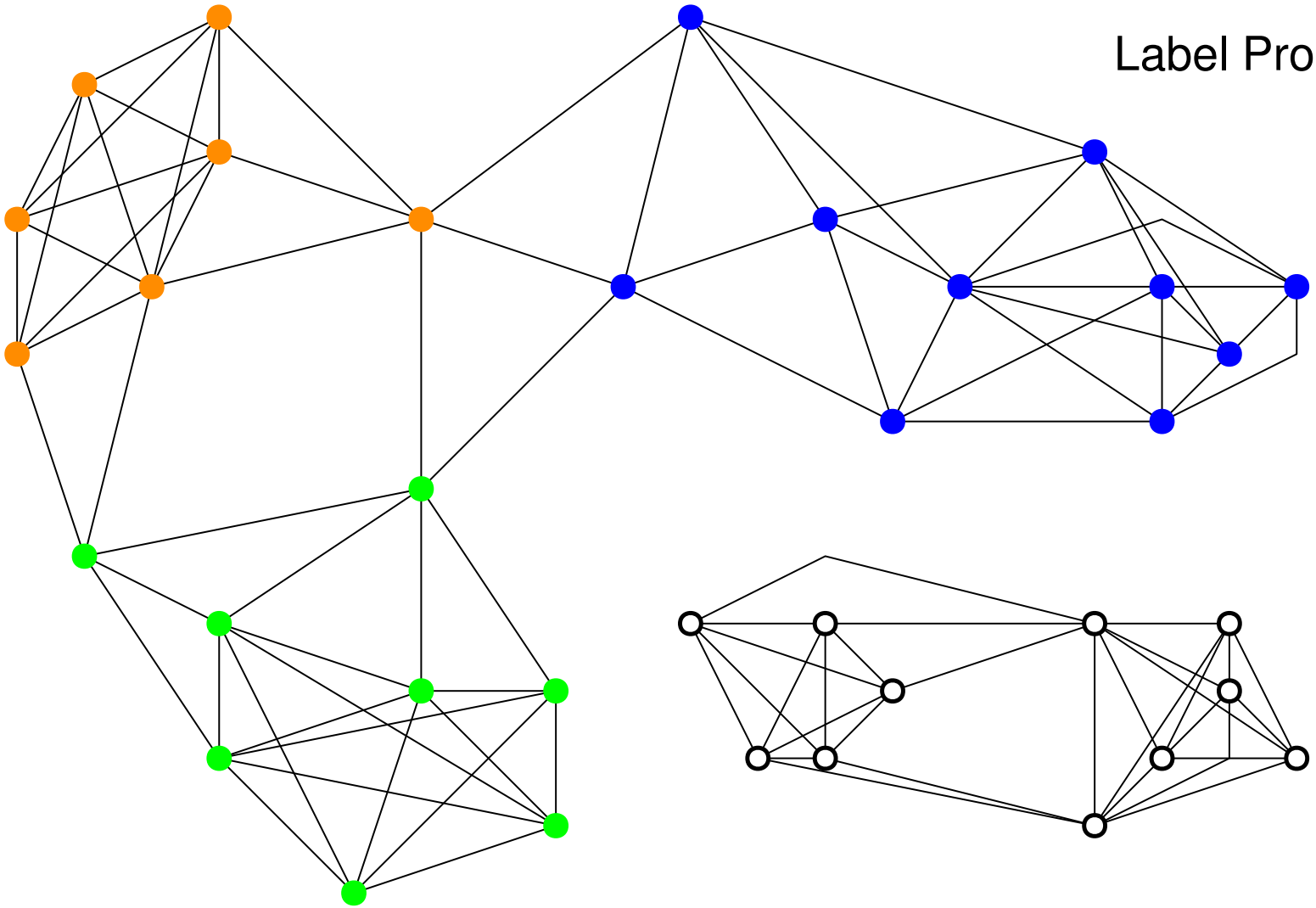
Coarsening - Beispiel



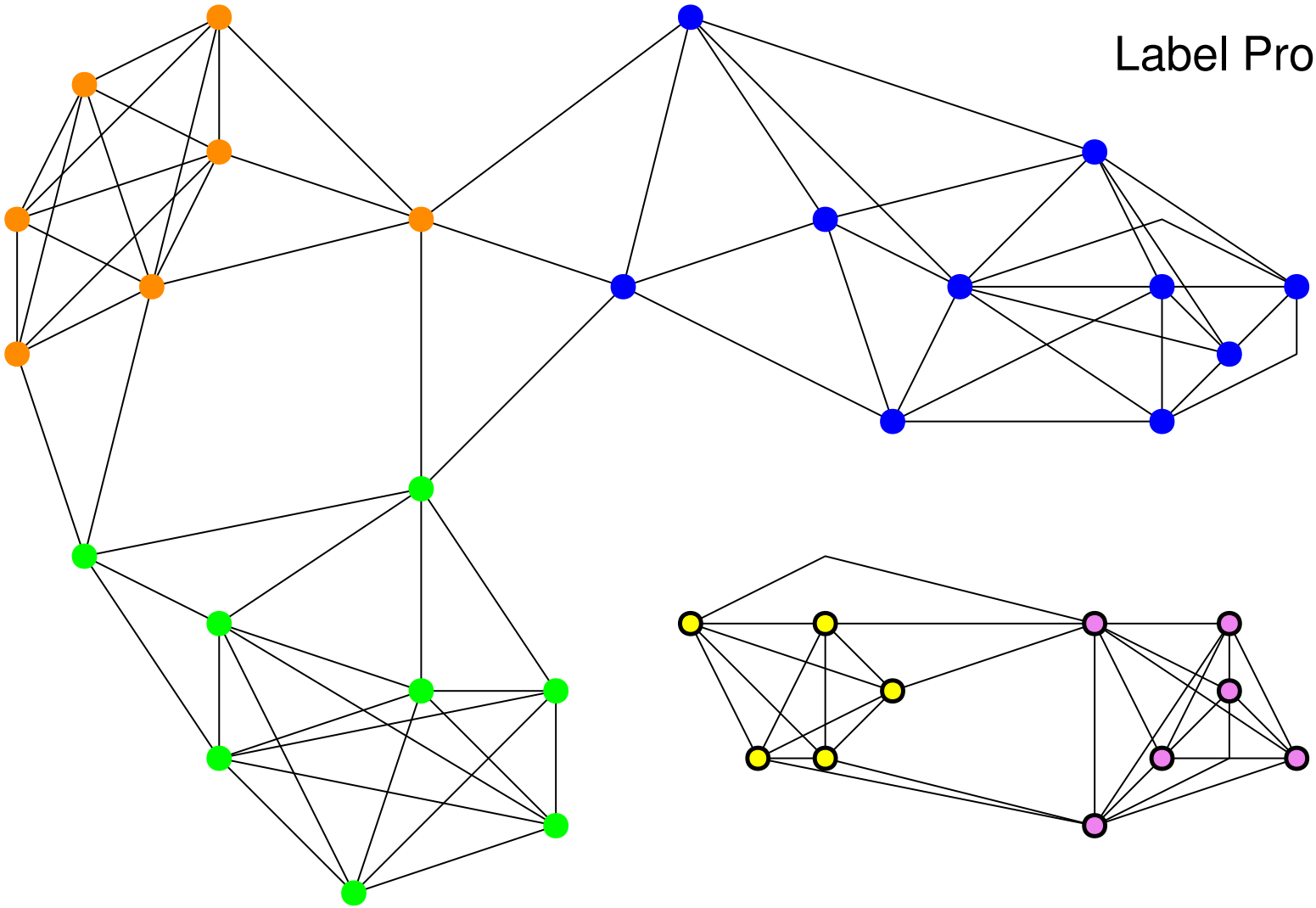
Coarsening - Beispiel



Coarsening - Beispiel

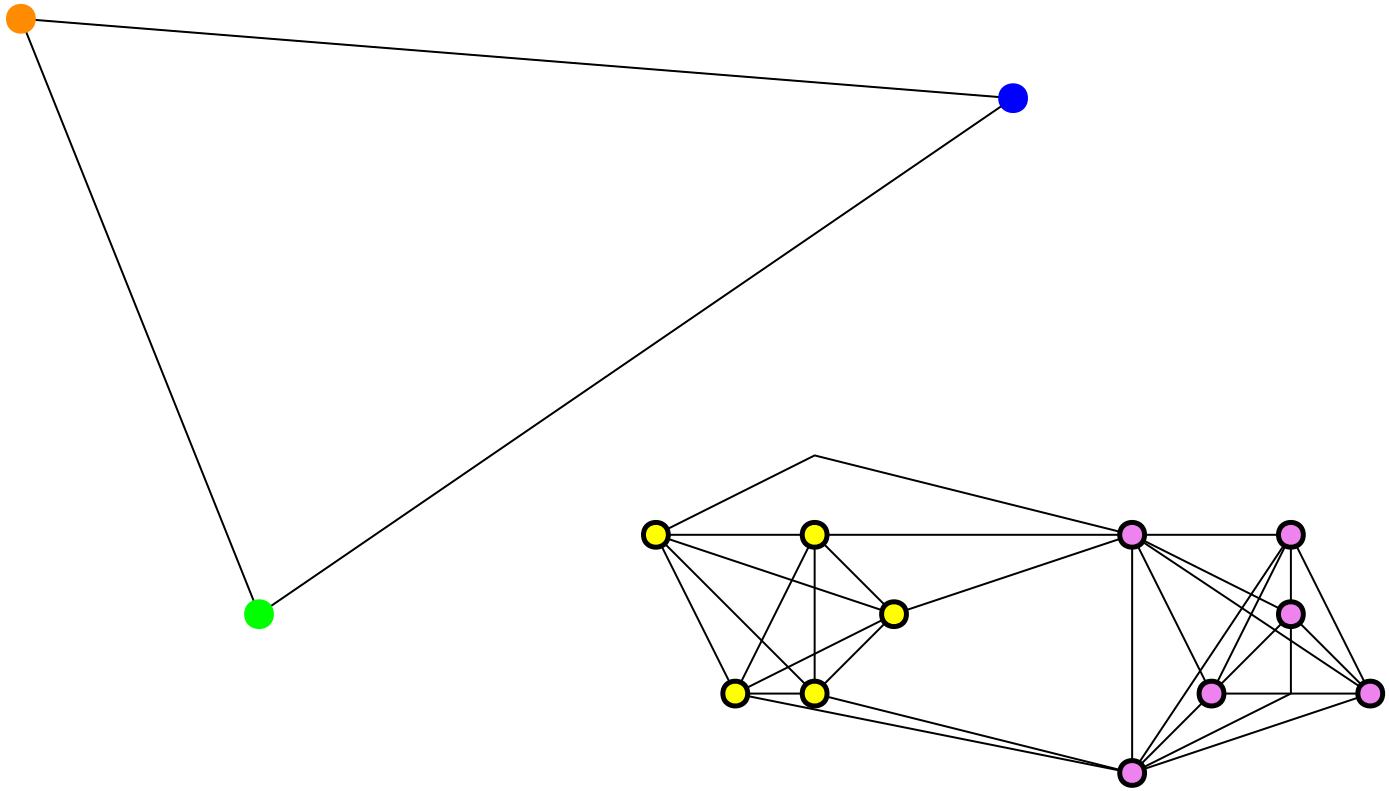


Coarsening - Beispiel



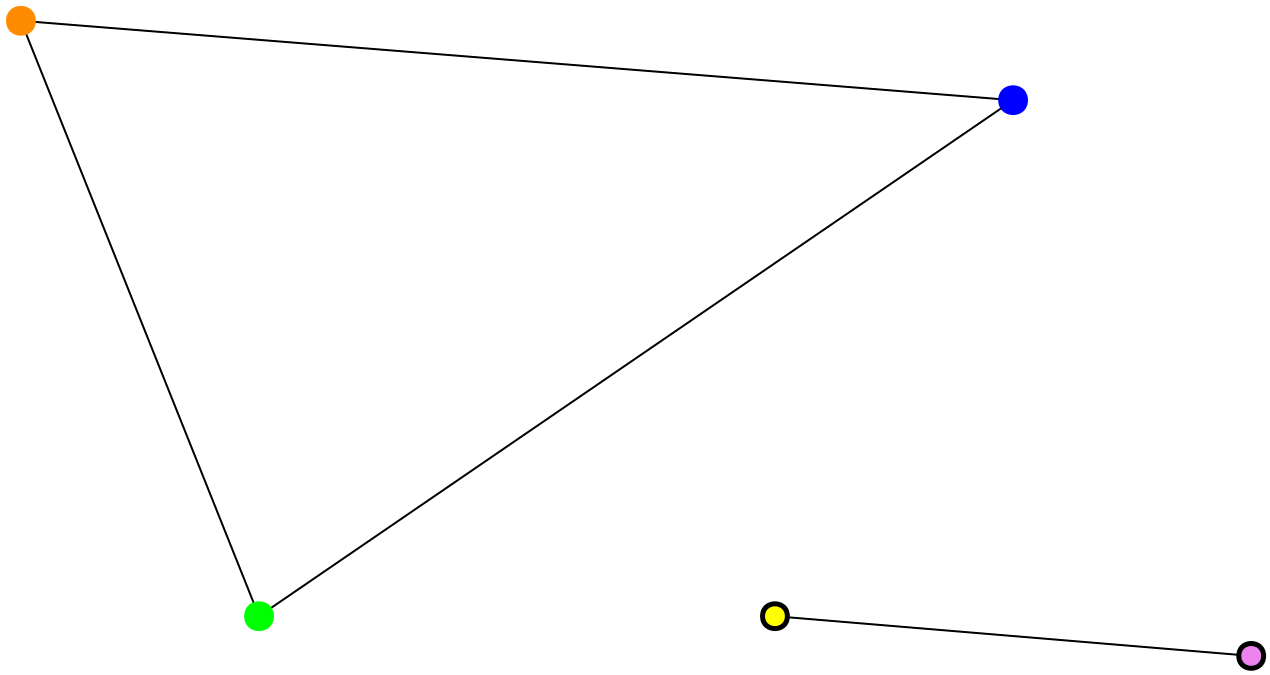
Coarsening - Beispiel

Contraction



Coarsening - Beispiel

Contraction



Initiales Training

Training auf dem gröbsten Graphen

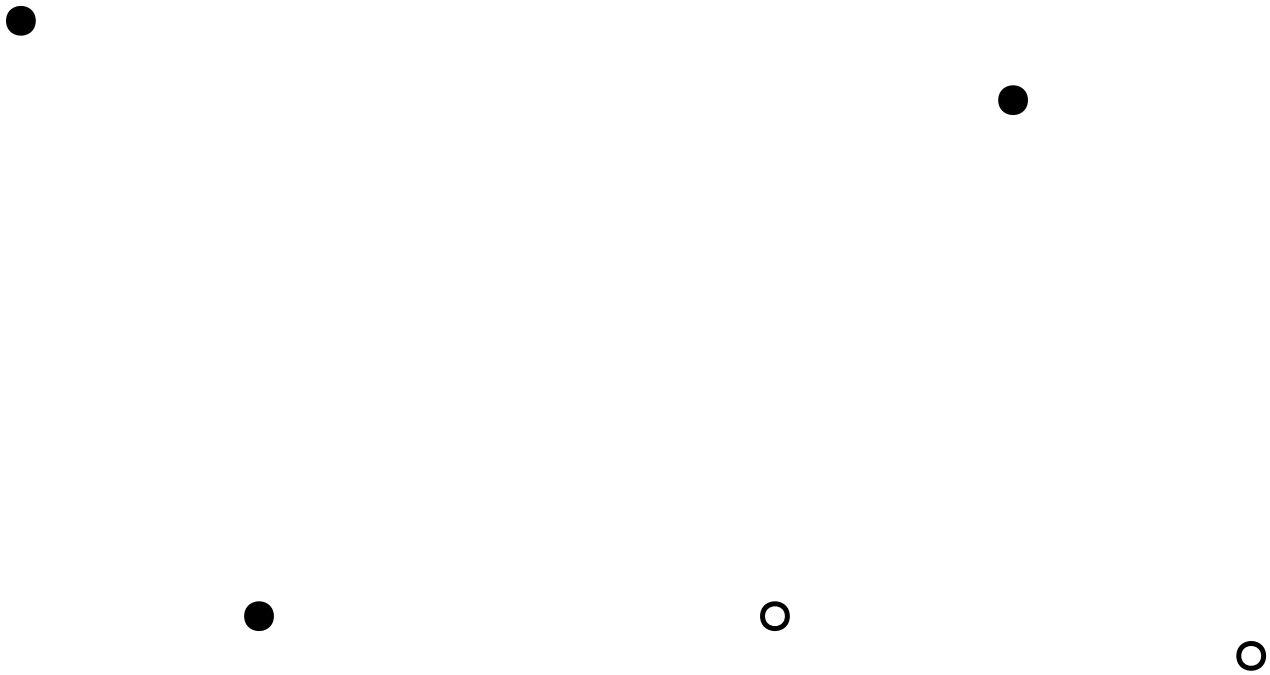
Parametersuche

- Grid Search
- Uniform Design
- 2-stufig

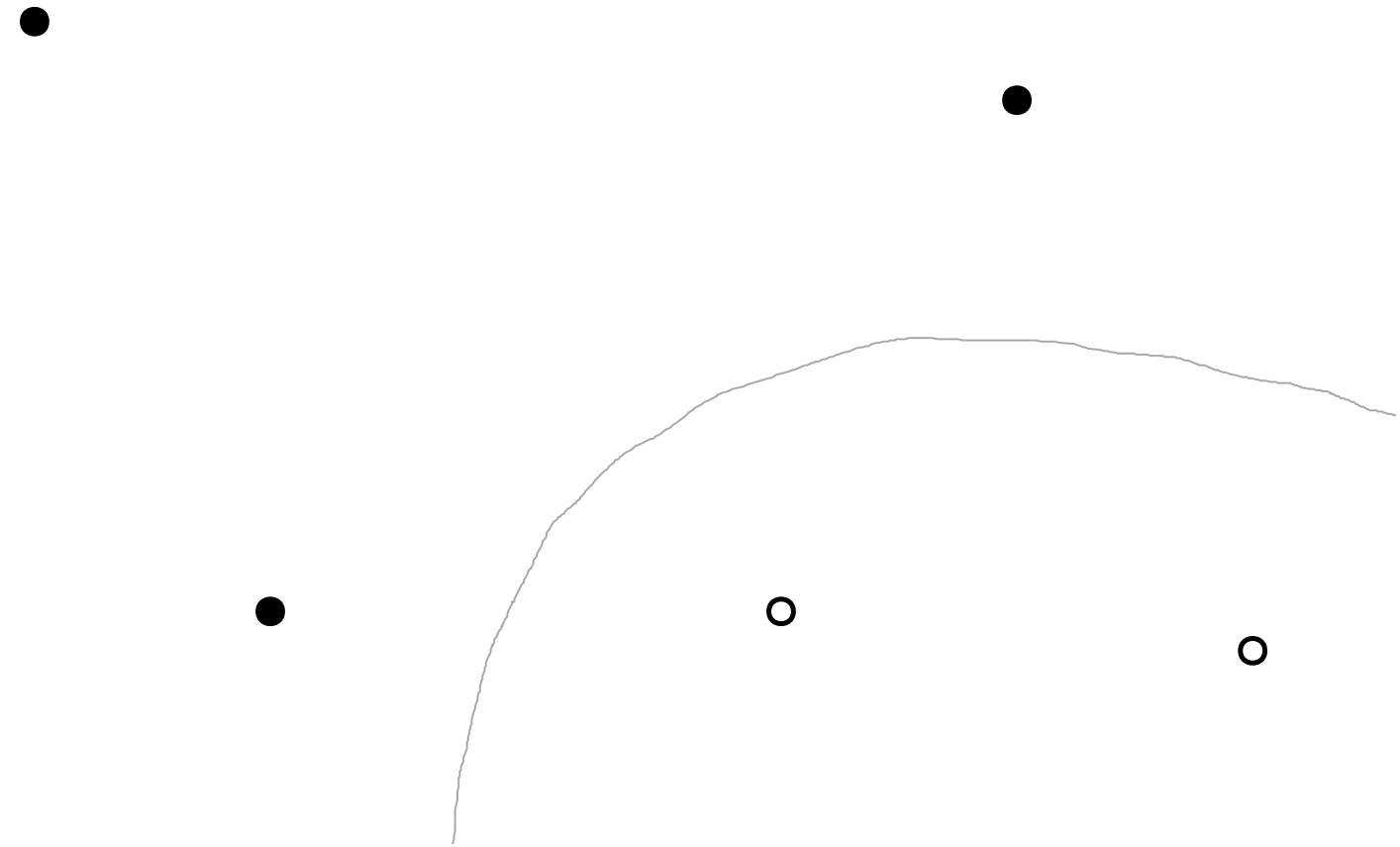
LibSVM zugrunde liegende SVM Bibliothek

Testen auf Validierungsdatensatz um beste Parameter zu identifizieren

Initiales Training - Beispiel



Initiales Training - Beispiel



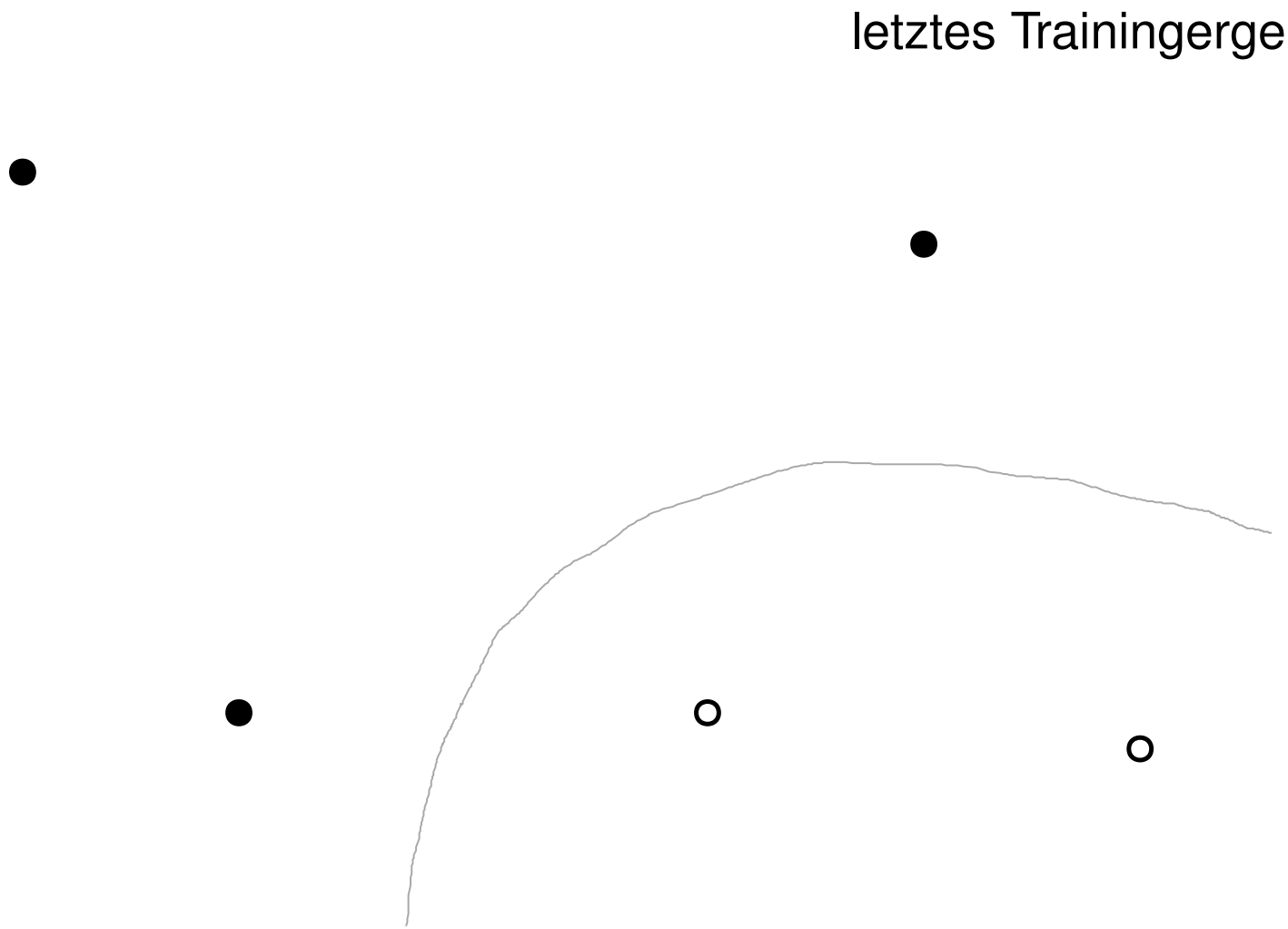
Refinement

Ziel: weitere Level der Hierarchie nutzen & Ergebnisse wiederverwenden

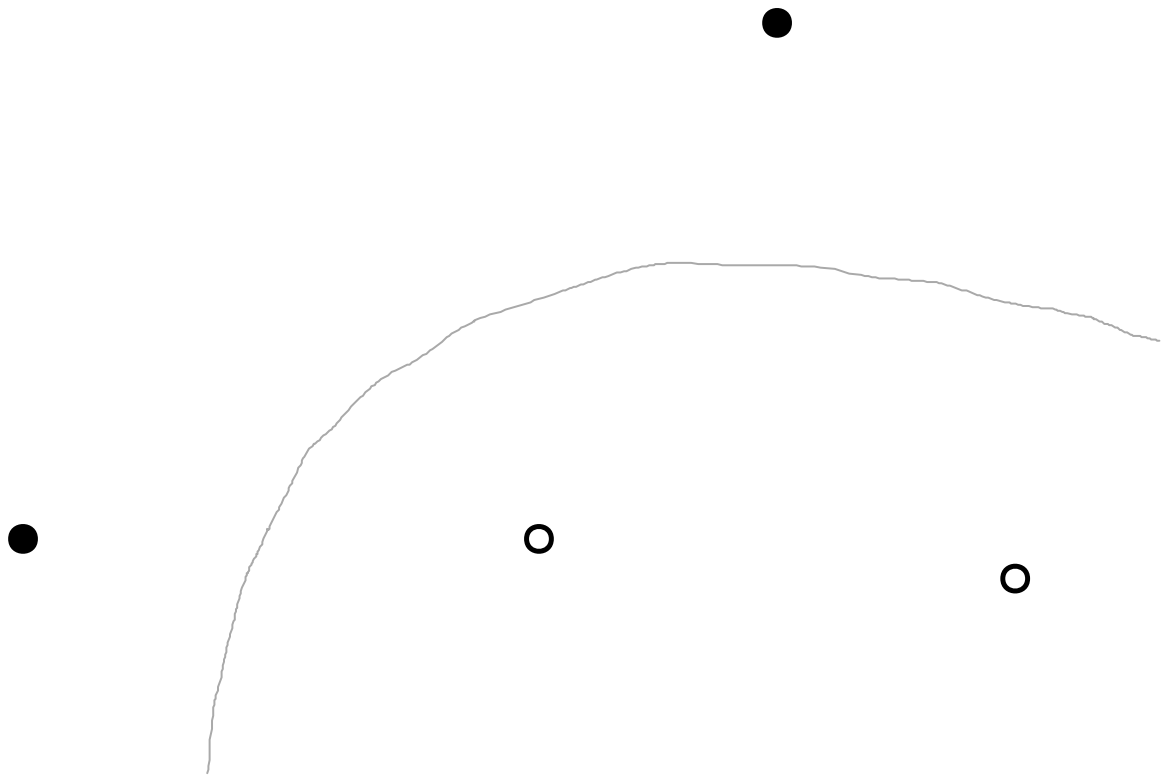
rekursiv auf jedem Level trainieren:

- nur Support Vektoren aus vorherigem Level “nach oben” projizieren
- Erkenntnisse über Parameter des vorherigem Levels wiederverwenden

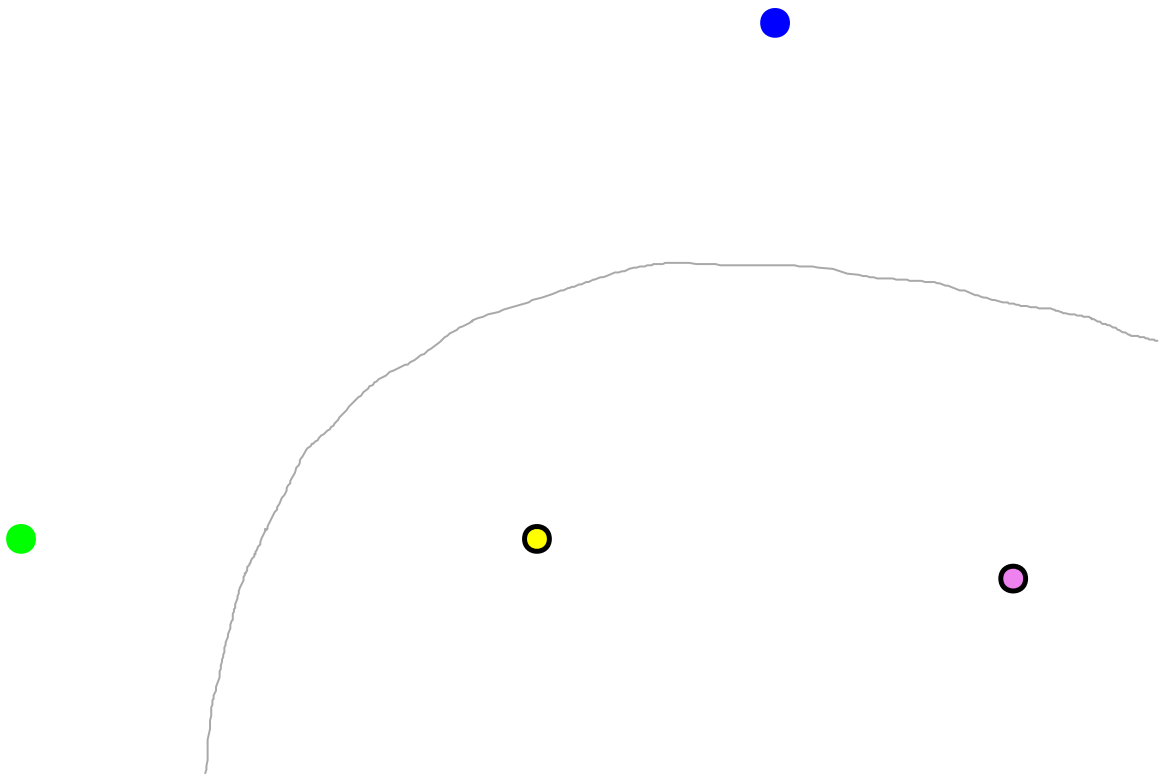
Refinement - Beispiel



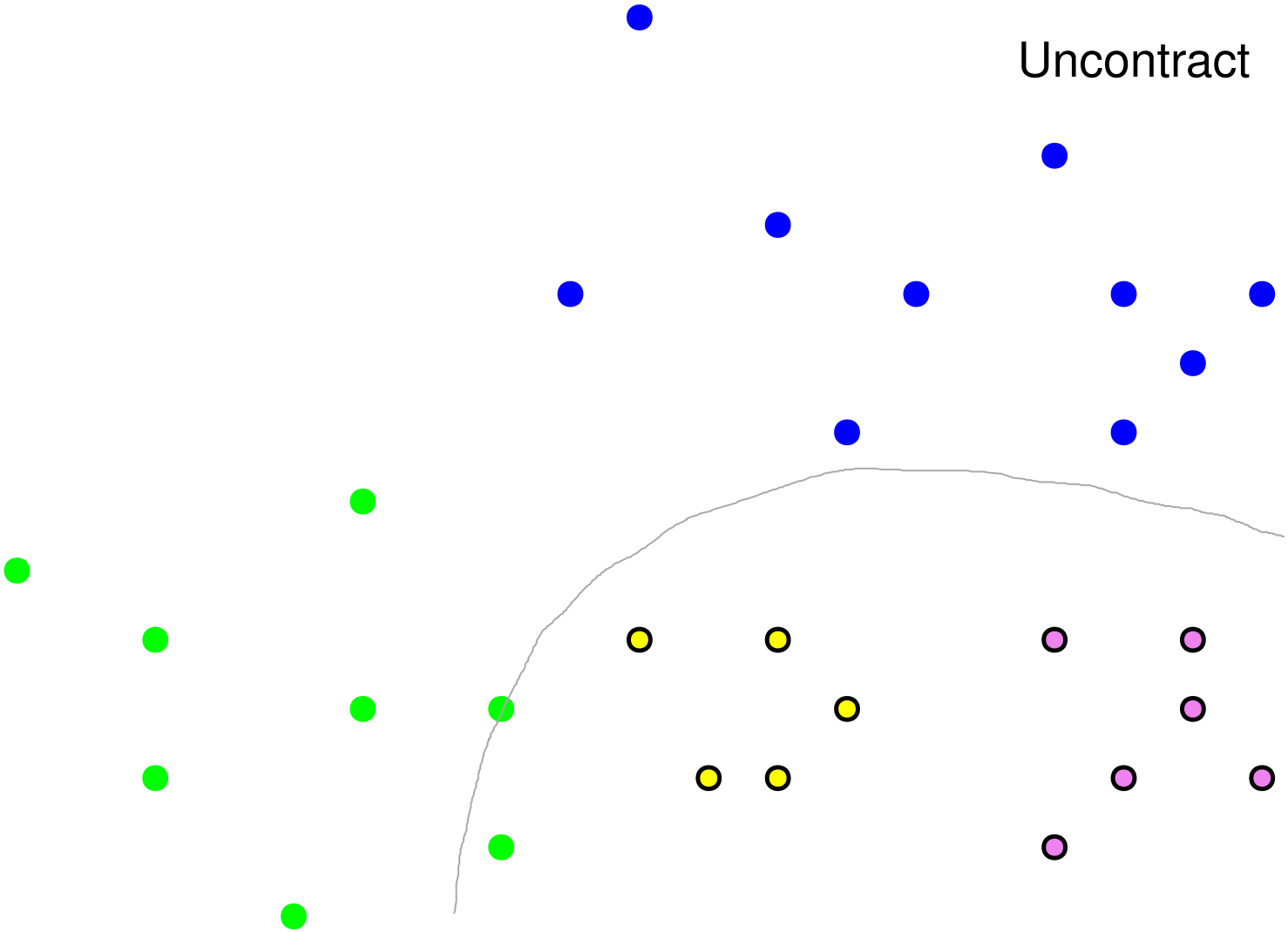
Refinement - Beispiel



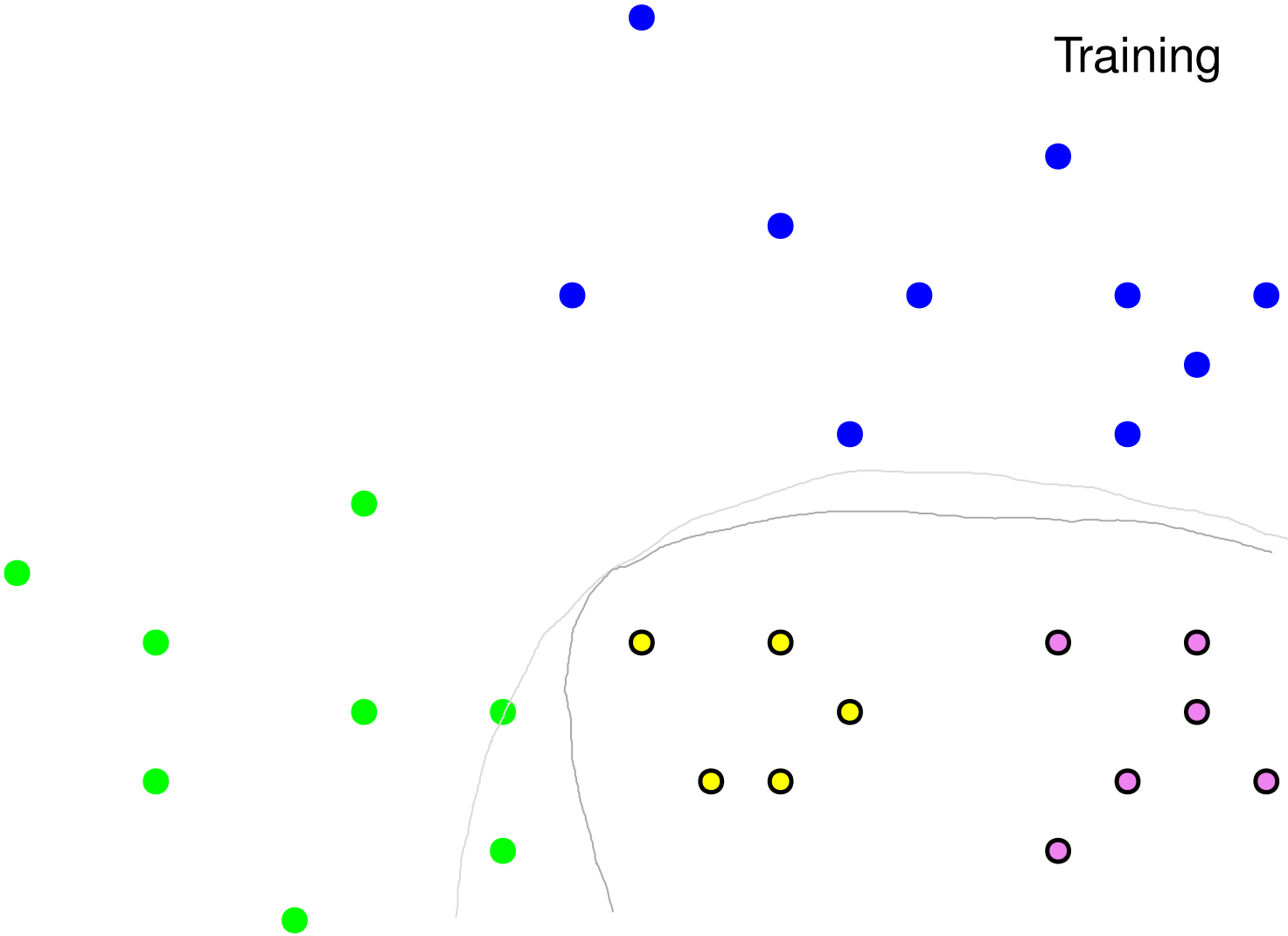
Refinement - Beispiel



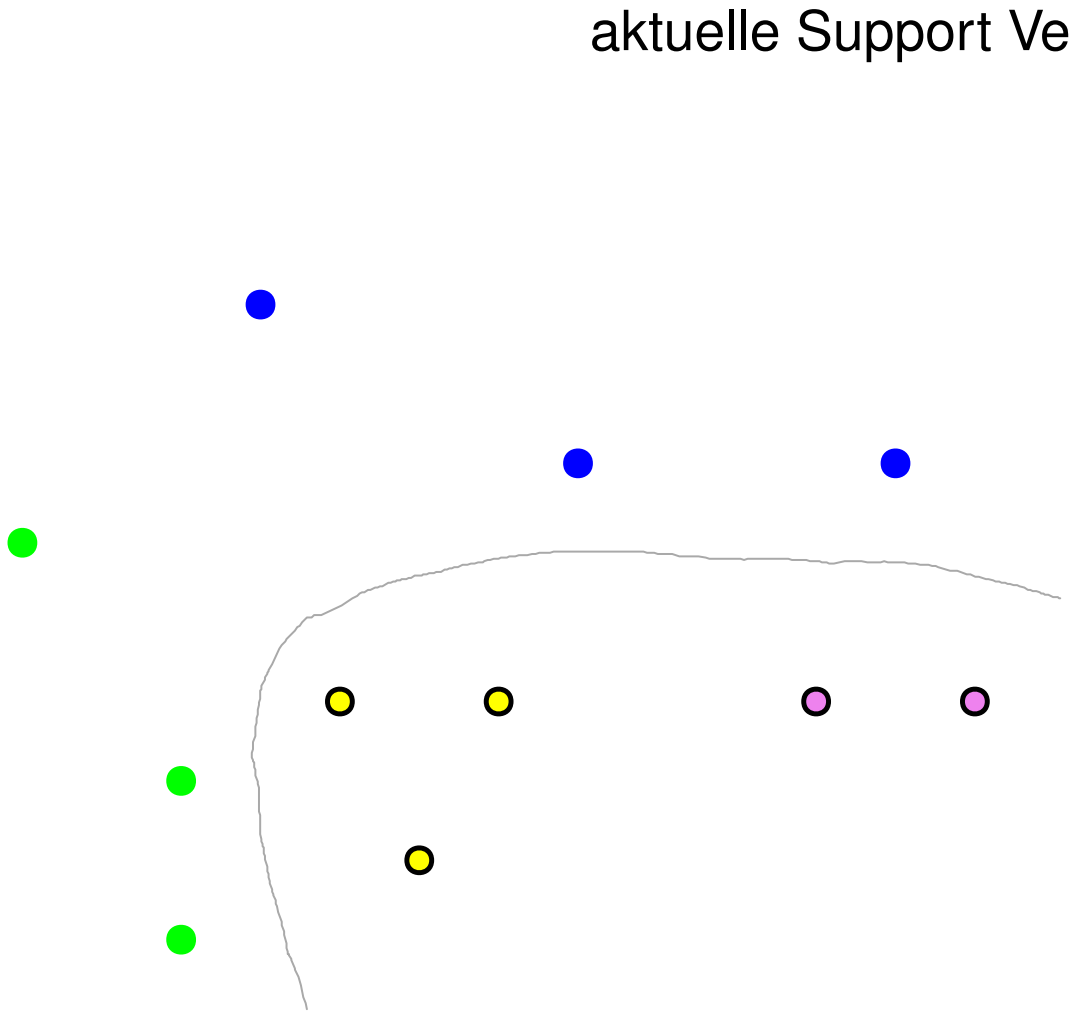
Refinement - Beispiel



Refinement - Beispiel



Refinement - Beispiel



Instanzen

Name	Größe	Features	C^+	C^-	Balance
Advertisement	3.279	1.558	459	2.820	0,86
APS failure	76.000	170	1.375	74.625	0,98
Buzz	140.707	77	27.775	112.932	0,80
Census	299.285	41	18.568	280.717	0,94
Cod-rna	59.535	8	19.845	39.690	0,67
EEG Eye State	14.980	14	6.723	8.257	0,55
Forest (Class 1)	581.012	54	221.840	369.172	0,64
Forest (Class 2)	581.012	54	283.301	297.711	0,51
Forest (Class 3)	581.012	54	35.754	369.172	0,94
Forest (Class 4)	581.012	54	2.747	578.265	1,00
Forest (Class 5)	581.012	54	9.493	571.519	0,98
Forest (Class 6)	581.012	54	17.367	563.645	0,97
Forest (Class 7)	581.012	54	20.510	560.502	0,96
Hypothyroid	3.919	21	240	3.679	0,94
Isolet (Class A)	6.919	617	240	5.998	0,96
Letter (Class A)	20.000	16	786	19.266	0,96
Letter (Class B)	20.000	16	766	19.266	0,96
Letter (Class H)	20.000	16	734	19.266	0,96
Letter (Class Z)	20.000	16	734	19.266	0,96
Musk (Clean)	6.598	166	1.017	5.581	0,85
Nursery	12.960	8	4.320	8.640	0,67
Protein	145.751	74	1.296	144.455	0,99
Ringnorm	7.400	20	3.664	3.736	0,50
Skin	245.057	3	50.859	194.198	0,79
Sleep (Class 1)	105.908	13	9.052	96.856	0,91
Twonorm	7.400	20	3.703	3.697	0,50

Klassifizierungsqualität

Dataset	mlsvm-AMG		KaMLSVM	
	ACC	G-mean	ACC	G-mean
Advertisement	0,91	0,78	0,94	**0,80**
APS failure	0,94	**0,94**	0,94	0,93
Buzz	0,94	**0,95**	0,94	0,94
Census	0,74	0,80	0,84	**0,83**
Cod-rna	0,94	**0,95**	0,94	0,94
EEG Eye State	0,78	**0,78**	0,78	0,77
Forest (Class 1)	0,73	0,75	0,80	**0,80**
Forest (Class 2)	0,73	0,73	0,80	**0,80**
Forest (Class 3)	0,90	0,94	0,93	**0,95**
Forest (Class 4)	0,94	0,97	0,97	**0,98**
Forest (Class 5)	0,73	0,79	0,90	**0,89**
Forest (Class 6)	0,82	0,89	0,90	**0,94**
Forest (Class 7)	0,83	0,87	0,89	**0,93**
Hypothyroid	0,93	**0,94**	0,97	0,85
Isolet (Class A)	0,97	**0,99**	0,99	**0,99**
Letter (Class A)	0,99	**0,98**	0,97	0,96
Letter (Class B)	0,96	**0,95**	0,94	0,94
Letter (Class H)	0,96	0,88	0,90	**0,91**
Letter (Class Z)	0,96	**0,97**	0,96	0,96
Musk (Clean)	0,92	0,92	0,97	**0,96**
Nursery	1,00	**1,00**	1,00	**1,00**
Protein	0,93	0,92	0,96	**0,93**
Ringnorm	0,98	**0,98**	0,97	0,97
Skin	0,99	0,99	1,00	**1,00**
Sleep (Class 1)	0,65	0,64	0,71	**0,66**
Twonorm	0,97	**0,97**	0,97	**0,97**

Zeitvergleich

Zusammenfassung

- neue Mulitilevel SVM mit Label Propagation
- vergleichbare Klassifizierungsqualität
- Speed-Up von mindestens zwei und bis zu 20

Ausblick

- Coarsening Abbruchkriterium verbessern
- Parallelisierung

Vielen Dank

Algorithm 2: Überblick

```
1  preprocess data
2  build k-fold instances
3  foreach k-fold instance do
4      use a fraction of the training data as validation data
5      build a k-nearest neighbor graph for C+ and C−
6      contract the graphs recursively and build the hierarchies
7      Initial Training
8      while levels in the hierarchies do
9          train a SVM model on the SV of the previous level
10         evaluate on the validation data
11     use the best trained model of all levels as final model
12     evaluate the final model with the test data
13 average the results of the k-folds
```

Algorithm 3: Initial Training

Input: $\mathbf{C}_c^+$, $\mathbf{C}_c^-$, VD

Output: $\text{model}_{best}, \text{SV}_{best}, C_{best}, \gamma_{best}$

```
1  evalList := list of evaluated SVM models and parameters
2  params_1 ← UD sweep around initial position
3  foreach (C, γ) ∈ params_1 do
4      (model, SV) ← train SVM on C_c^+ ∪ C_c^- using C, γ
5      res ← evaluate model on VD
6      evalList.add((res,model,SV,C,γ))
7  (C_good, γ_good) ← evalList.getEntryWithBestResult()
8  params_2 ← UD sweep around C_good and γ_good
9  foreach (C, γ) ∈ params_2 do
10     (model, SV) ← train SVM on C_c^+ ∪ C_c^- using C, γ
11     res ← evaluate model on VD
12     evalList.add((res,model,SV,C,γ))
13 (model_best, SV_best, C_best, γ_best) ← evalList.getEntryWithBestResult()
```

Klassifikationsqualitätskriterien

accuracy (ACC), *sensitivity* (SN), *specificity* (SP), *G-mean*

$$ACC = \frac{TP + TN}{FP + TN + TP + FN}$$
$$SN = \frac{TP}{TP + FN}$$
$$SP = \frac{TN}{TN + FP}$$
$$G\text{-}mean = \sqrt{SP * SN}$$

TP true positives, correctly classified points of $\mathbf{C}^+$
FN false negatives, wrongly classified points of $\mathbf{C}^+$
TN true negatives, correctly classified points of $\mathbf{C}^-$
FP false positives, wrongly classified points of $\mathbf{C}^-$

Refinement Verbesserung

Dataset	initial		final		KaMLSVM	mlsvm-AMG
	ACC	G-mean	ACC	G-mean	Levels	Levels
Advertisement	0,84	0,87	0,94	0,80	2	3
APS failure	0,96	0,93	0,94	0,93	4	4
Buzz	0,94	0,93	0,94	0,94	4	5
Census	0,76	0,81	0,84	0,83	5	5
Cod-rna	0,93	0,94	0,94	0,94	4	4
EEG Eye State	0,65	0,64	0,78	0,77	3	3
Forest (Class 1)	0,76	0,75	0,80	0,80	5	5
Forest (Class 2)	0,75	0,75	0,80	0,80	5	5
Forest (Class 3)	0,94	0,94	0,93	0,95	5	5
Forest (Class 4)	0,94	0,97	0,97	0,98	5	5
Forest (Class 5)	0,83	0,88	0,90	0,89	5	5
Forest (Class 6)	0,89	0,94	0,90	0,94	5	5
Forest (Class 7)	0,95	0,92	0,90	0,93	5	5
Hypothyroid	0,97	0,87	0,97	0,85	2	3
Isolet (Class A)	0,80	0,89	0,99	0,99	2	3
Letter (Class A)	0,95	0,95	0,97	0,96	3	4
Letter (Class B)	0,93	0,93	0,94	0,94	3	4
Letter (Class H)	0,93	0,89	0,90	0,91	3	4
Letter (Class Z)	0,95	0,95	0,96	0,96	3	4
Musk (Clean)	0,93	0,86	0,97	0,96	2	3
Nursery	1,00	1,00	1,00	1,00	2	3
Protein	0,90	0,92	0,96	0,93	3	5
Ringnorm	0,80	0,80	0,97	0,97	2	3
Skin	1,00	1,00	1,00	1,00	5	5
Sleep (Class 1)	0,89	0,36	0,71	0,66	3	4
Twonorm	0,97	0,97	0,97	0,97	2	3

Zeitanteil