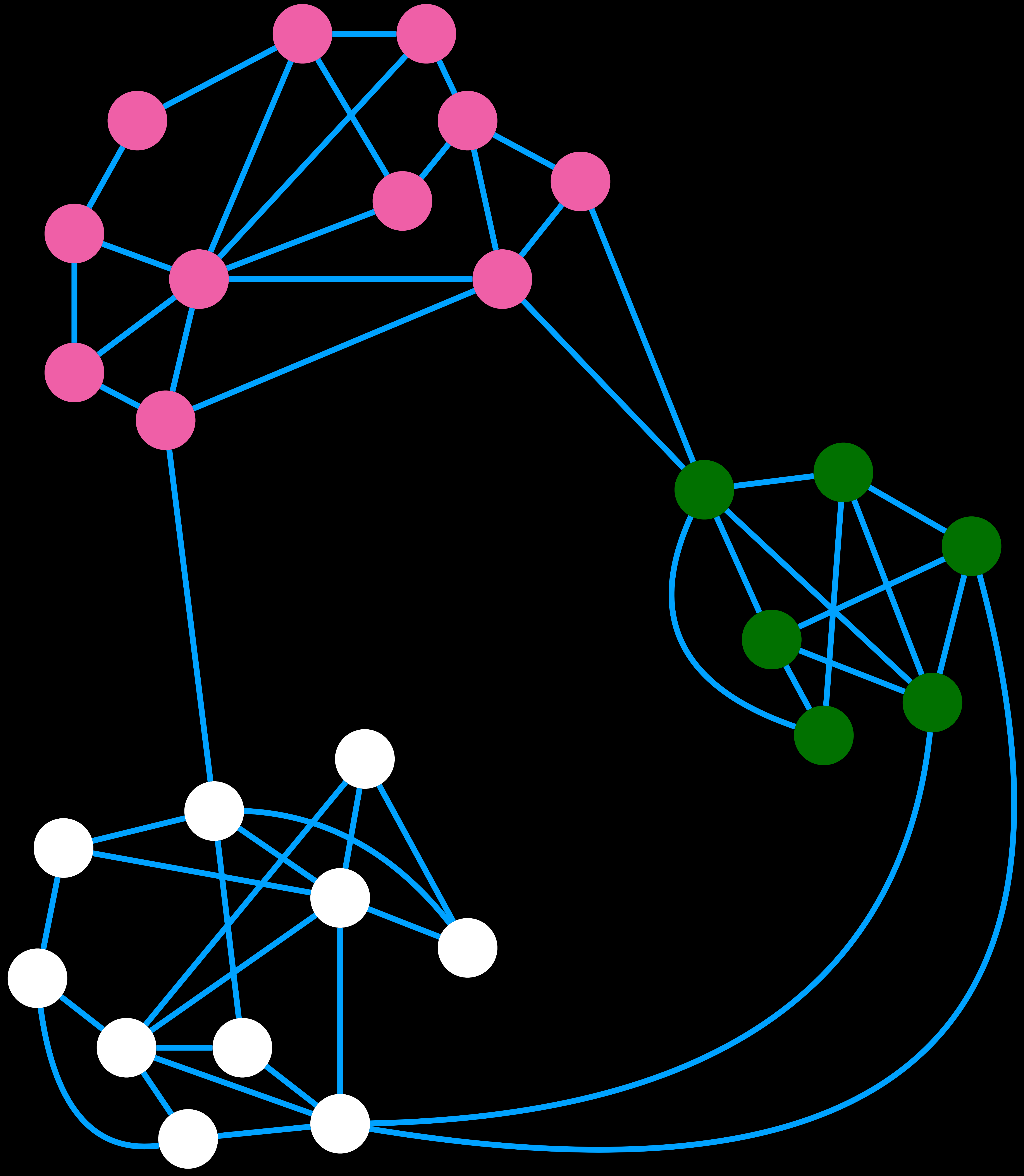
Sublinear-Time Clustering Oracle for Signed Graphs

Stefan Neumann (KTH) and Pan Peng (USTC)
@StefanResearch

ICML'22

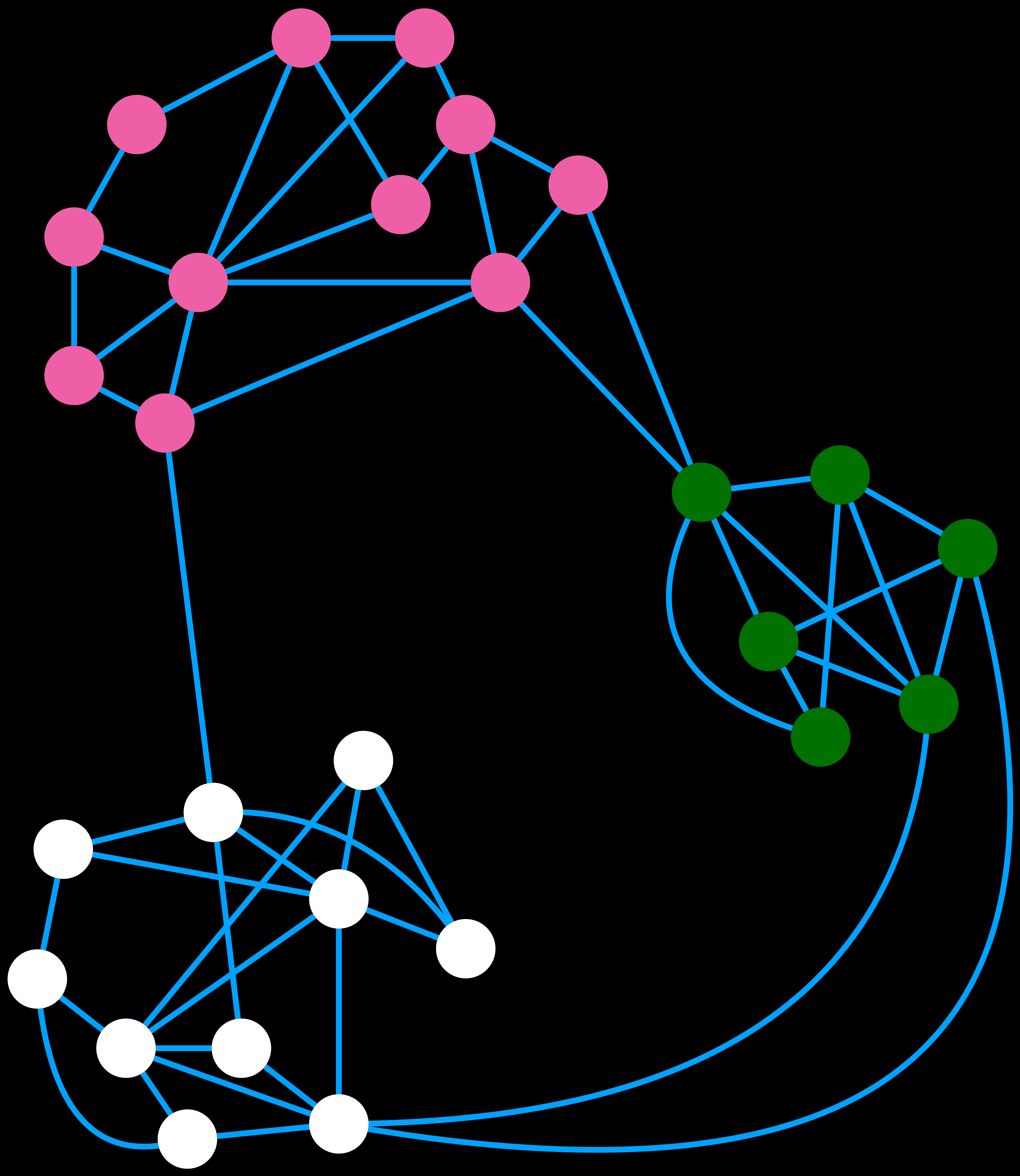
Graph Clustering

- Ubiquitous task in machine learning and data science



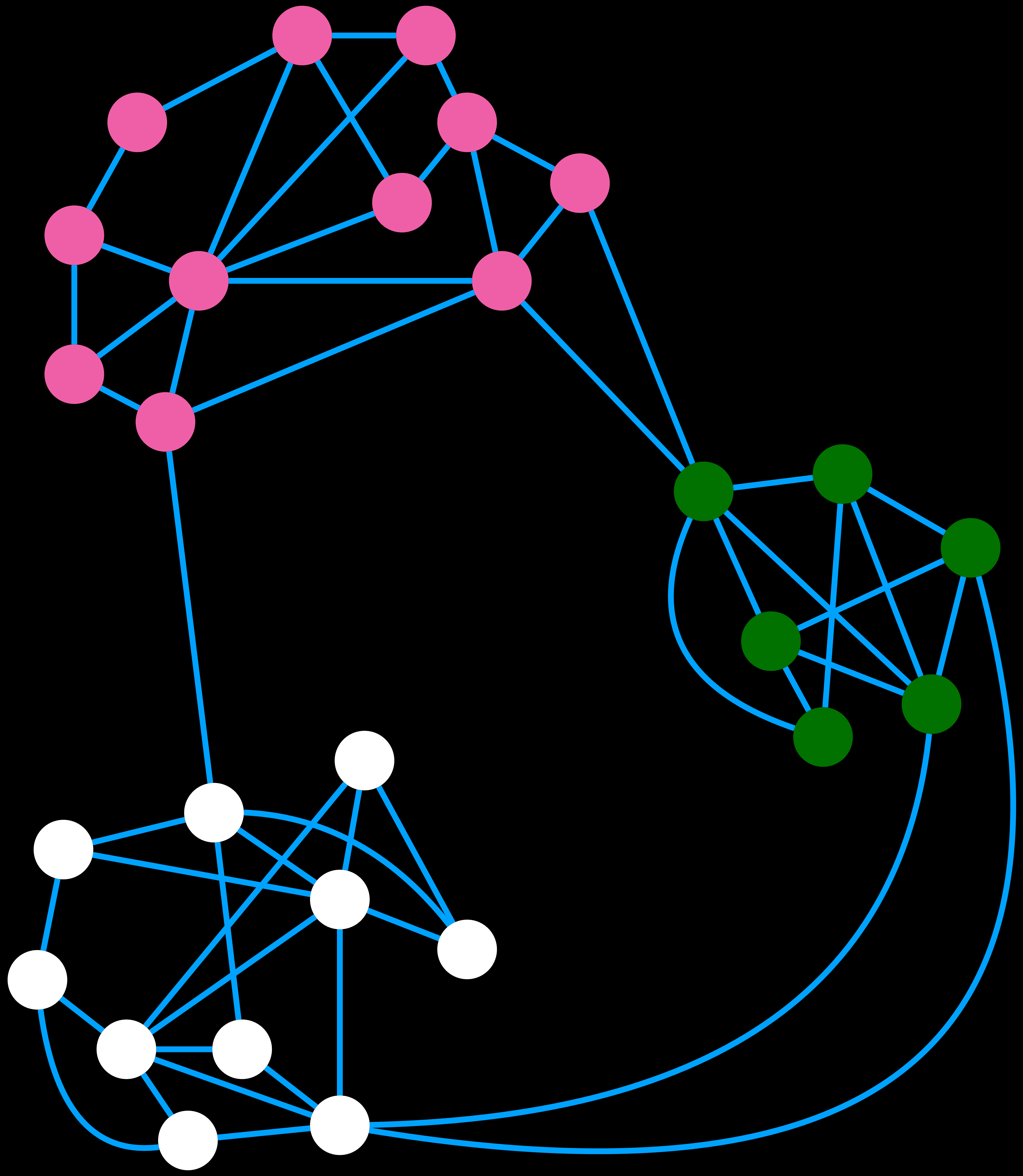
Graph Clustering

- Ubiquitous task in machine learning and data science
- *Application:*
  Find communities in (social) networks



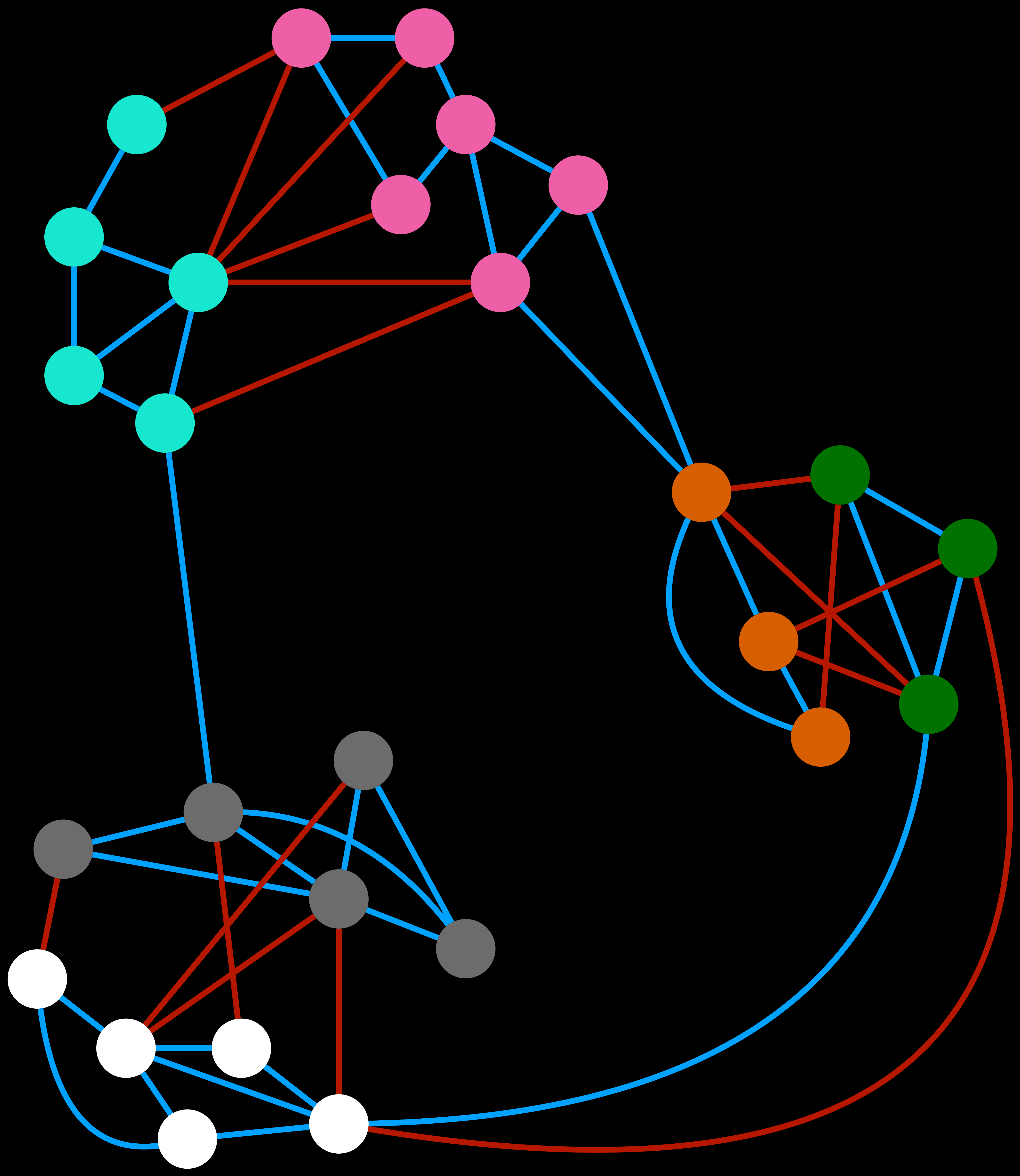
Graph Clustering

- Ubiquitous task in machine learning and data science
- *Application:*
  Find communities in (social) networks
- Typically studied for *unsigned* graphs



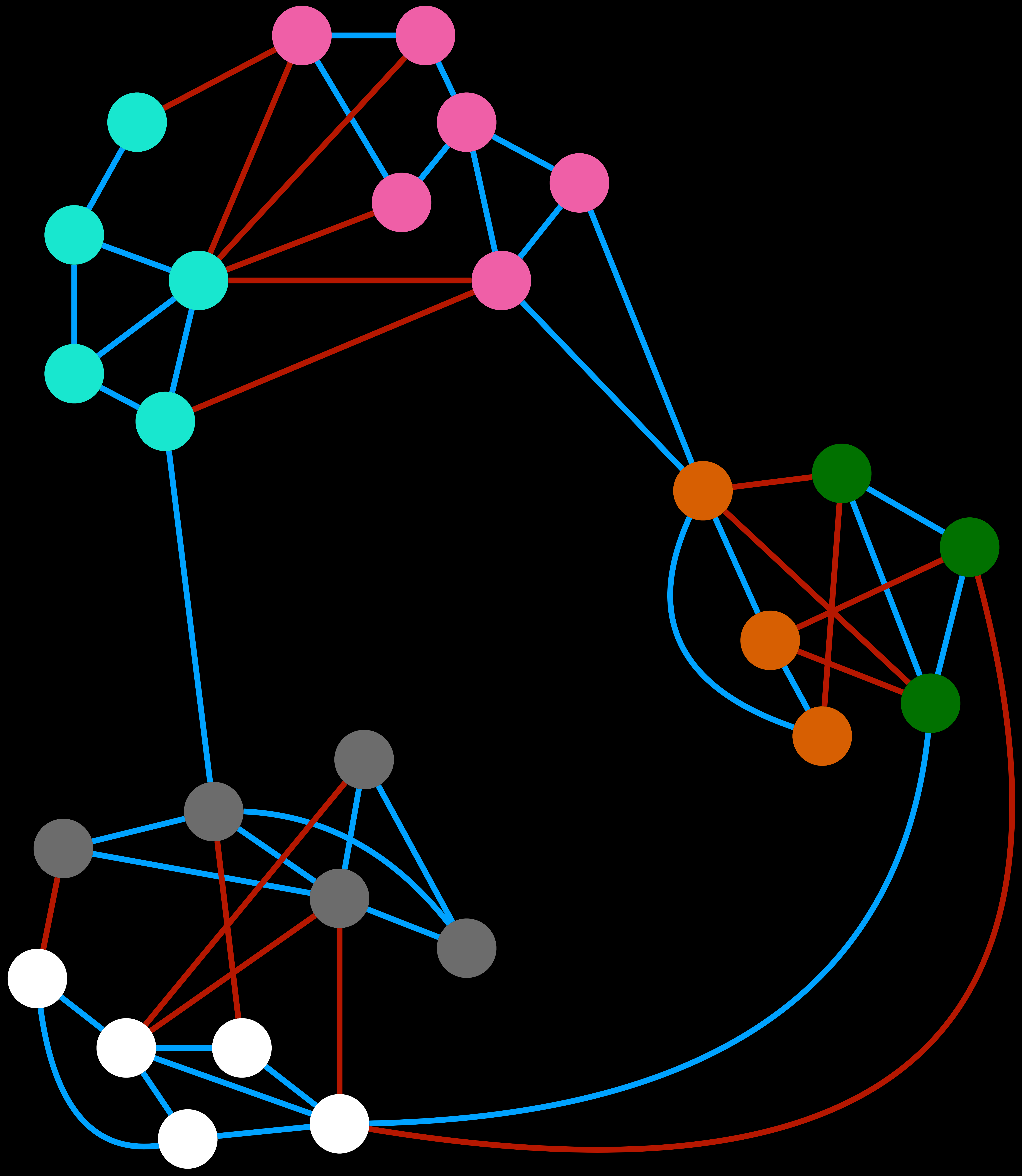
Signed Graphs

- Social networks can be seen as *signed* networks



Signed Graphs

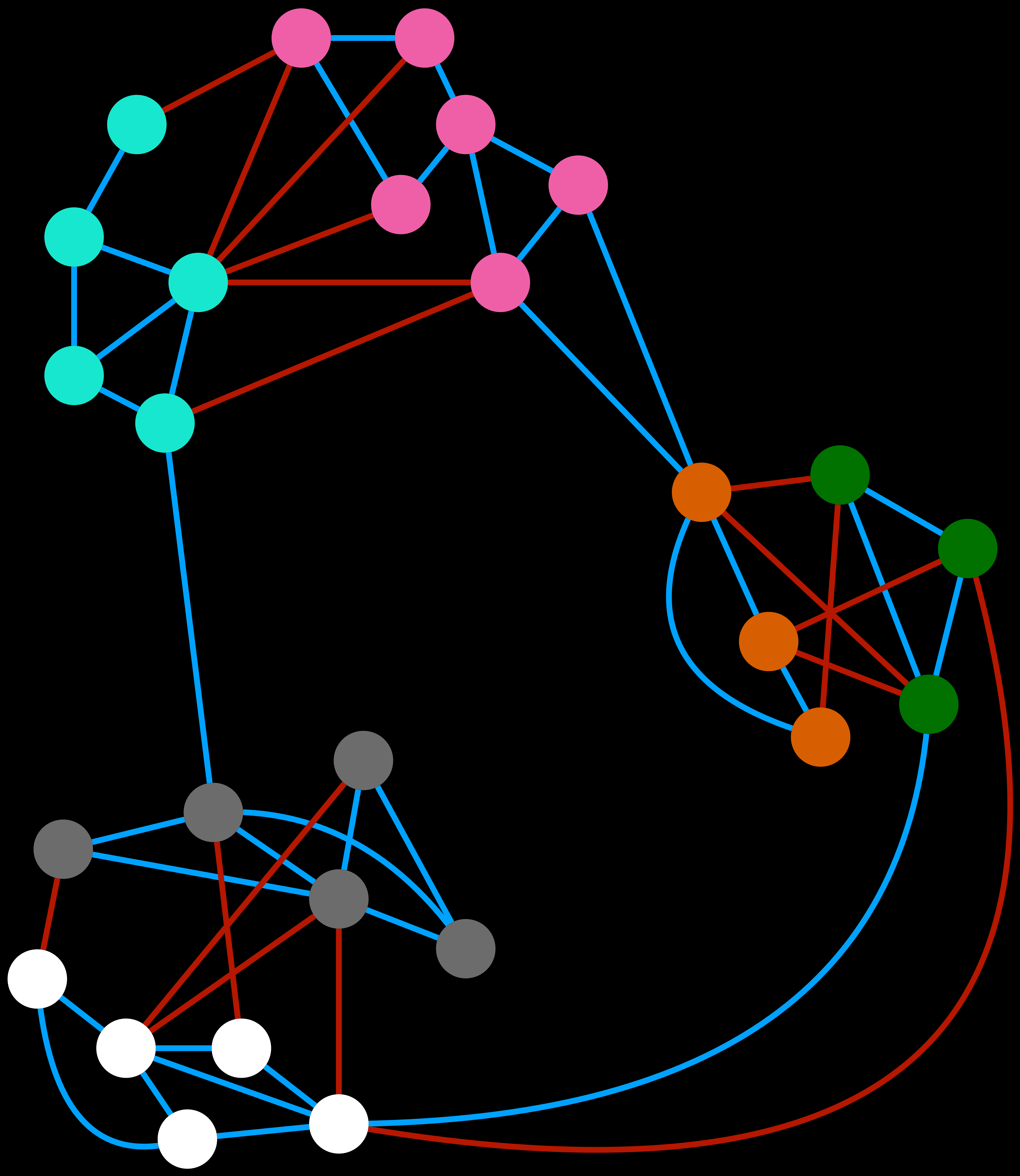
- Social networks can be seen as *signed* networks
- Each edge has a sign + or – indicating whether



Signed Graphs

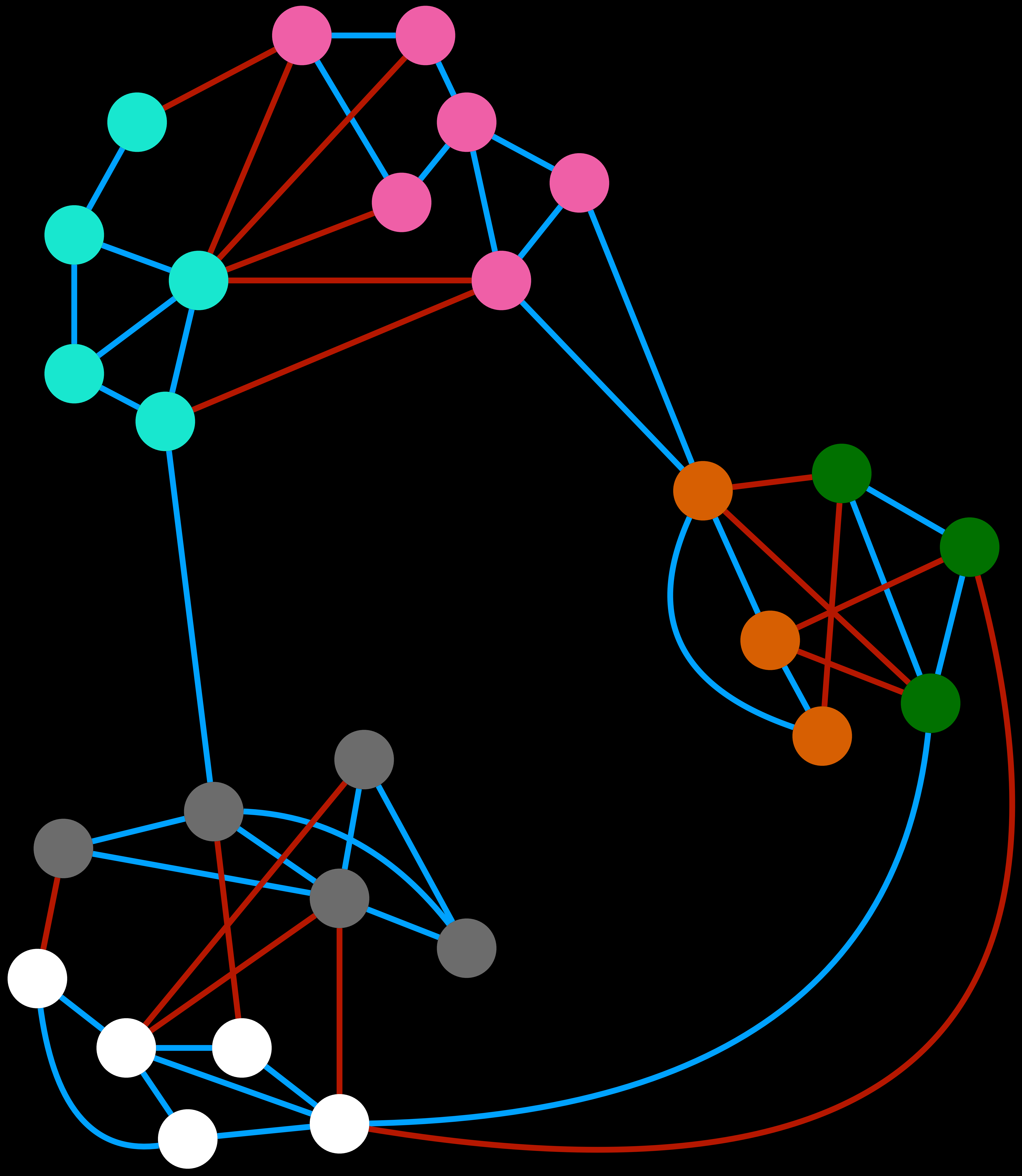
- Social networks can be seen as *signed* networks
- Each edge has a sign + or – indicating whether

  ➡ interaction was positive +



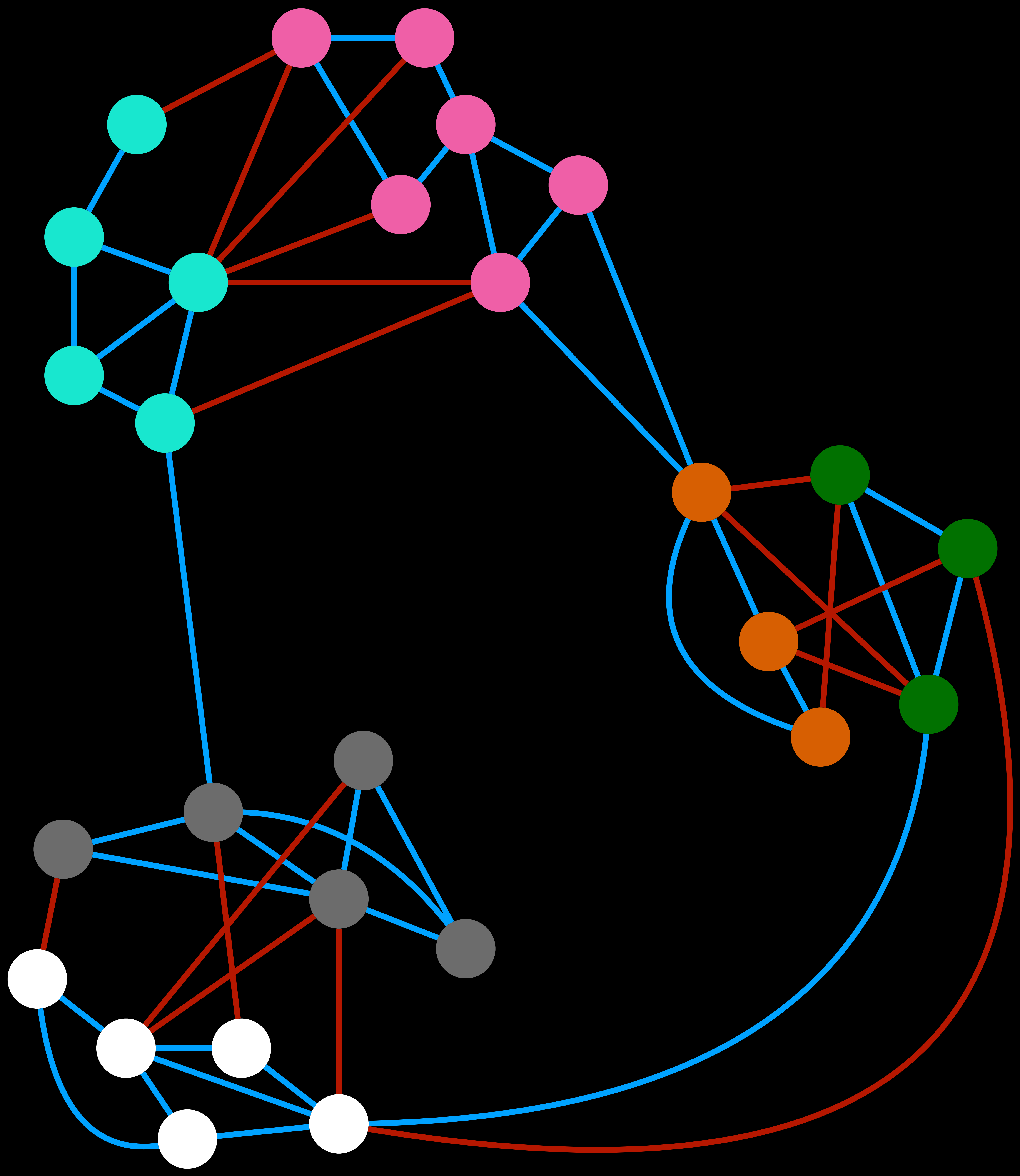
Signed Graphs

- Social networks can be seen as *signed* networks
- Each edge has a sign + or – indicating whether
  - ➡ interaction was positive +
  - ➡ or negative –



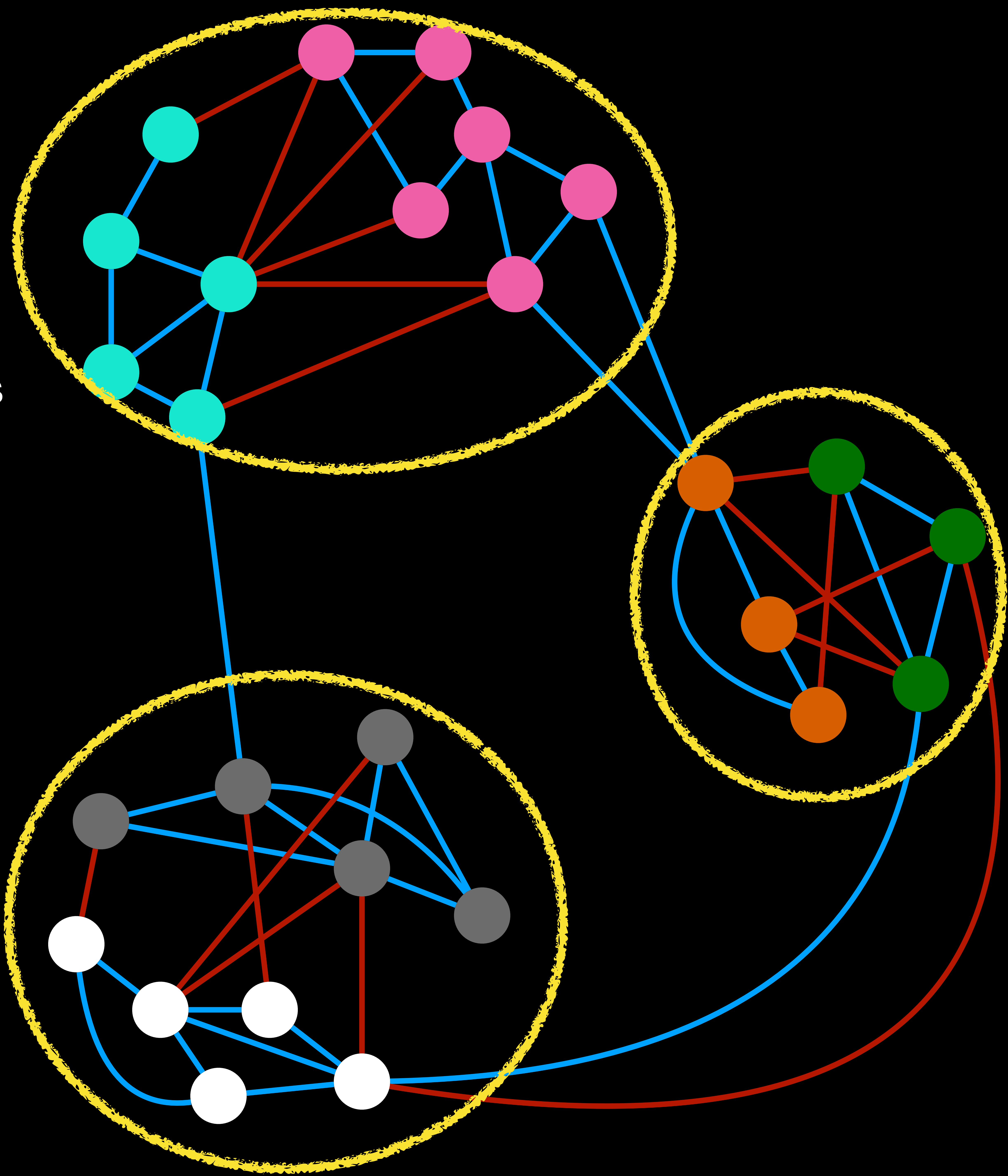
Signed Graphs

- Social networks can be seen as *signed* networks
- Each edge has a sign + or – indicating whether
  - ➡ interaction was positive +
  - ➡ or negative –
- Allows to *detect conflicting groups* in social networks



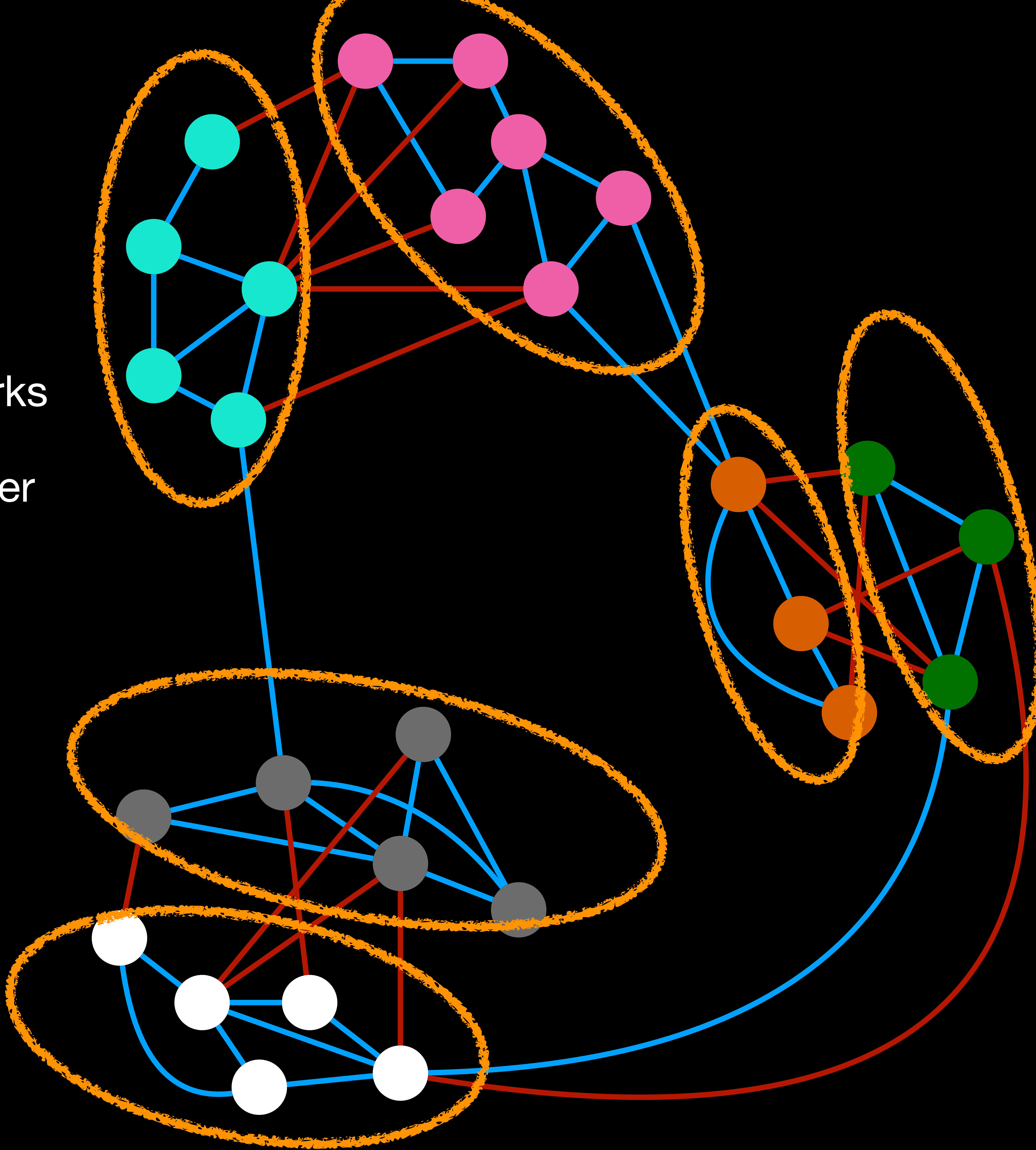
Signed Graphs

- Social networks can be seen as *signed* networks
- Each edge has a sign + or – indicating whether
  - ➡ interaction was positive +
  - ➡ or negative –
- Allows to *detect conflicting groups* in social networks



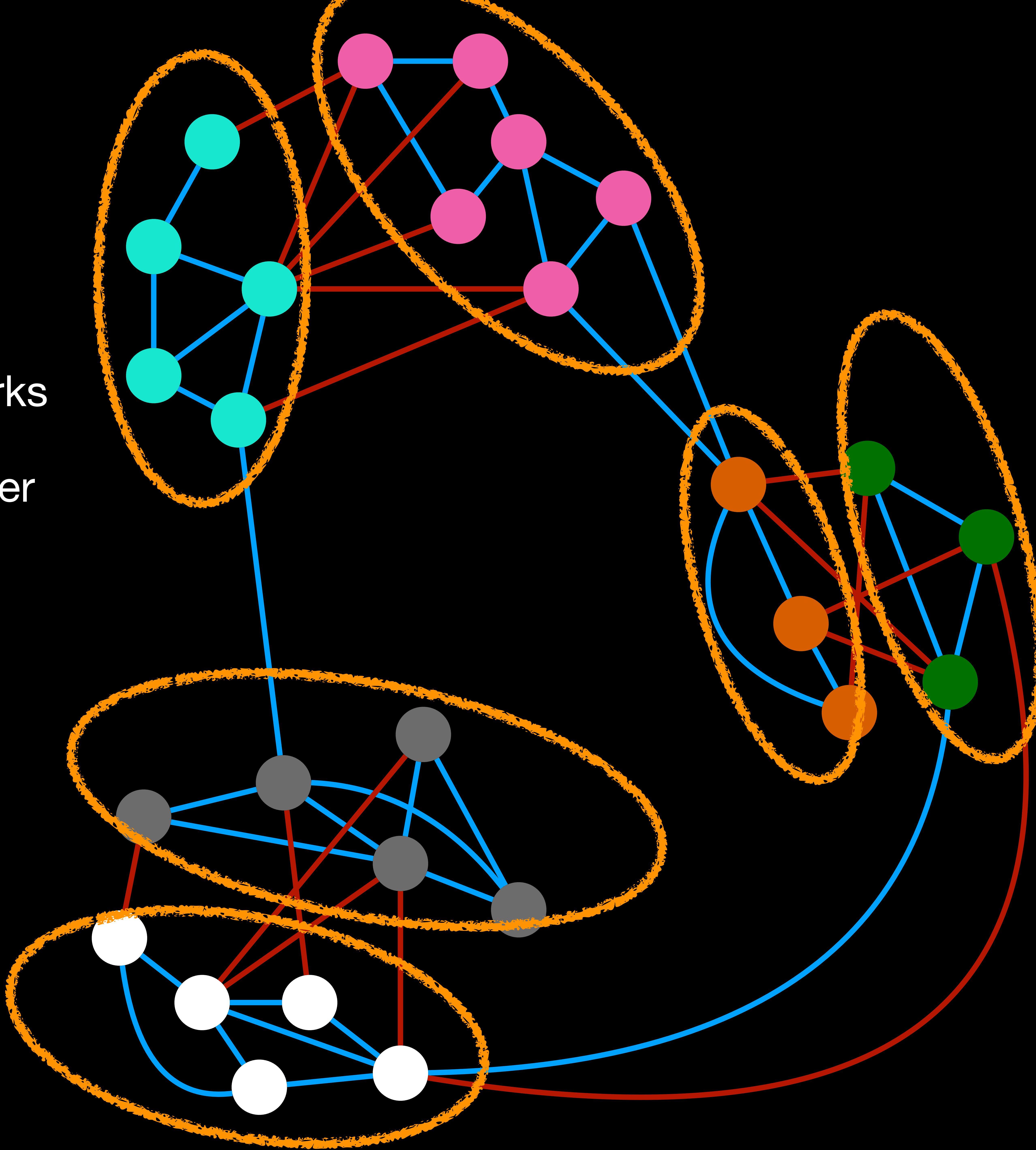
Signed Graphs

- Social networks can be seen as *signed* networks
- Each edge has a sign + or – indicating whether
  - ➡ interaction was positive +
  - ➡ or negative –
- Allows to *detect conflicting groups* in social networks



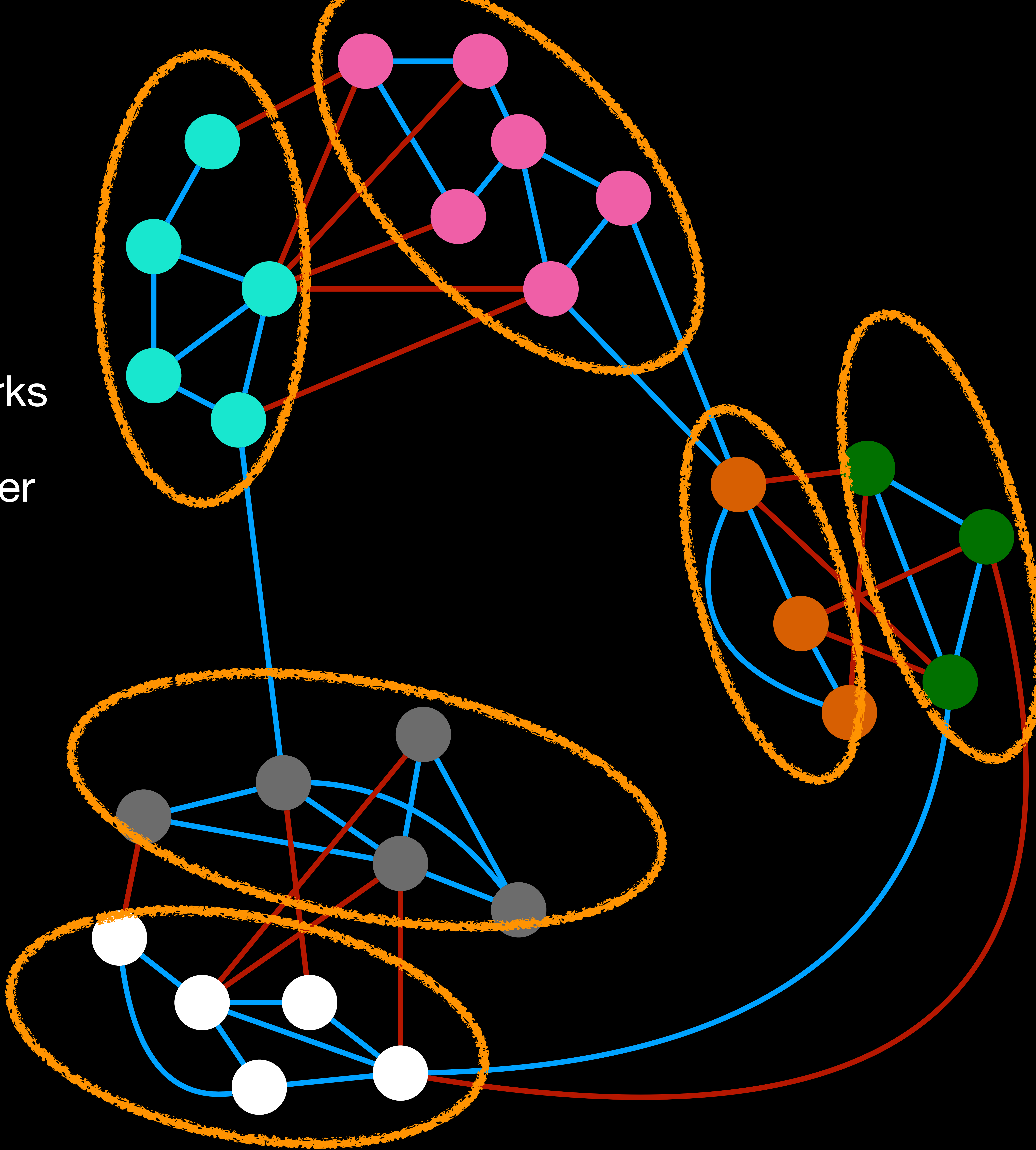
Signed Graphs

- Social networks can be seen as *signed* networks
- Each edge has a sign + or – indicating whether
  - ➡ interaction was positive +
  - ➡ or negative –
- Allows to *detect conflicting groups* in social networks
  - ➡ Democrats vs Republicans



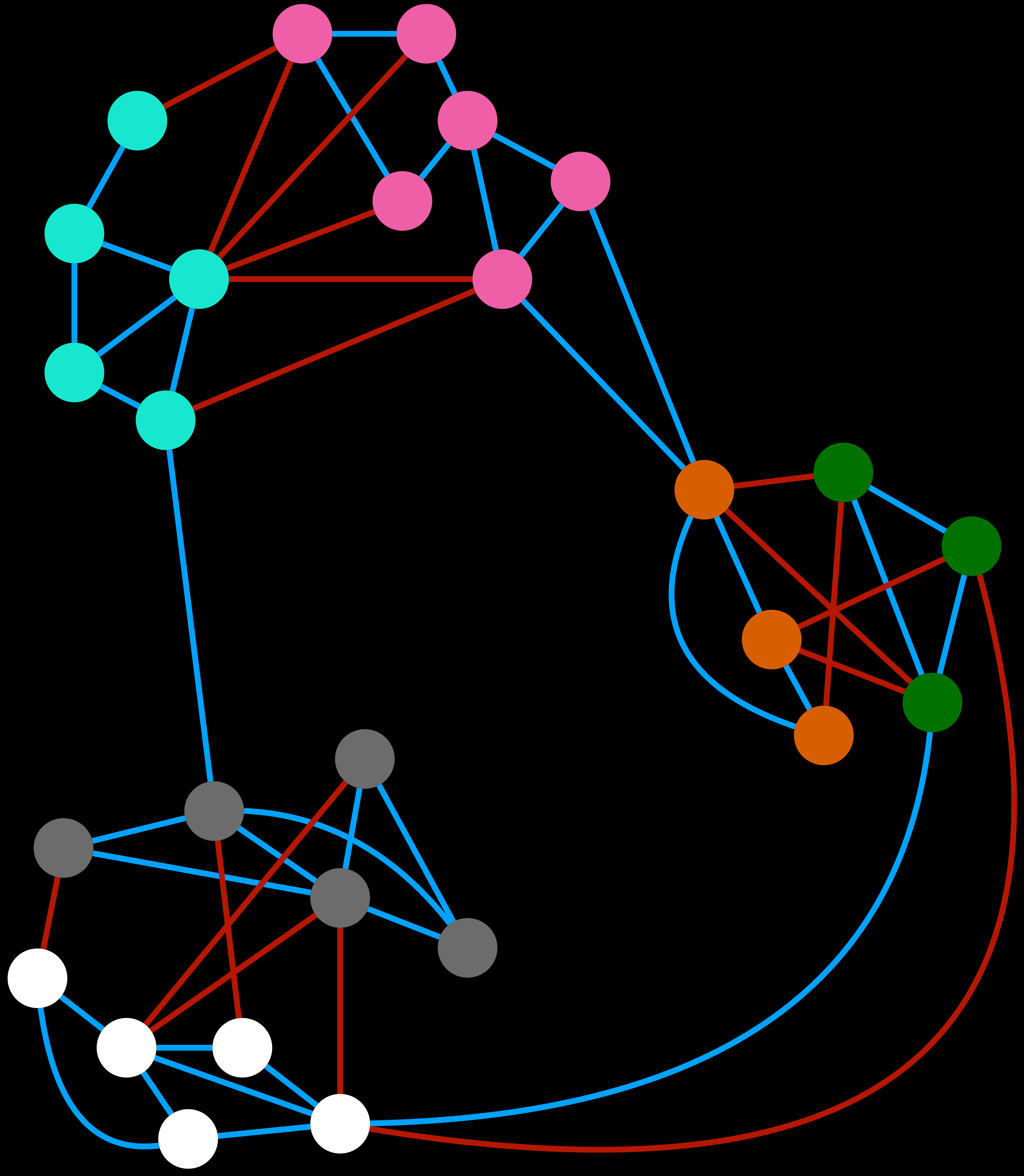
Signed Graphs

- Social networks can be seen as *signed* networks
- Each edge has a sign + or – indicating whether
  - ➡ interaction was positive +
  - ➡ or negative –
- Allows to *detect conflicting groups* in social networks
  - ➡ Democrats vs Republicans
  - ➡ analyze trust in Bitcoin networks



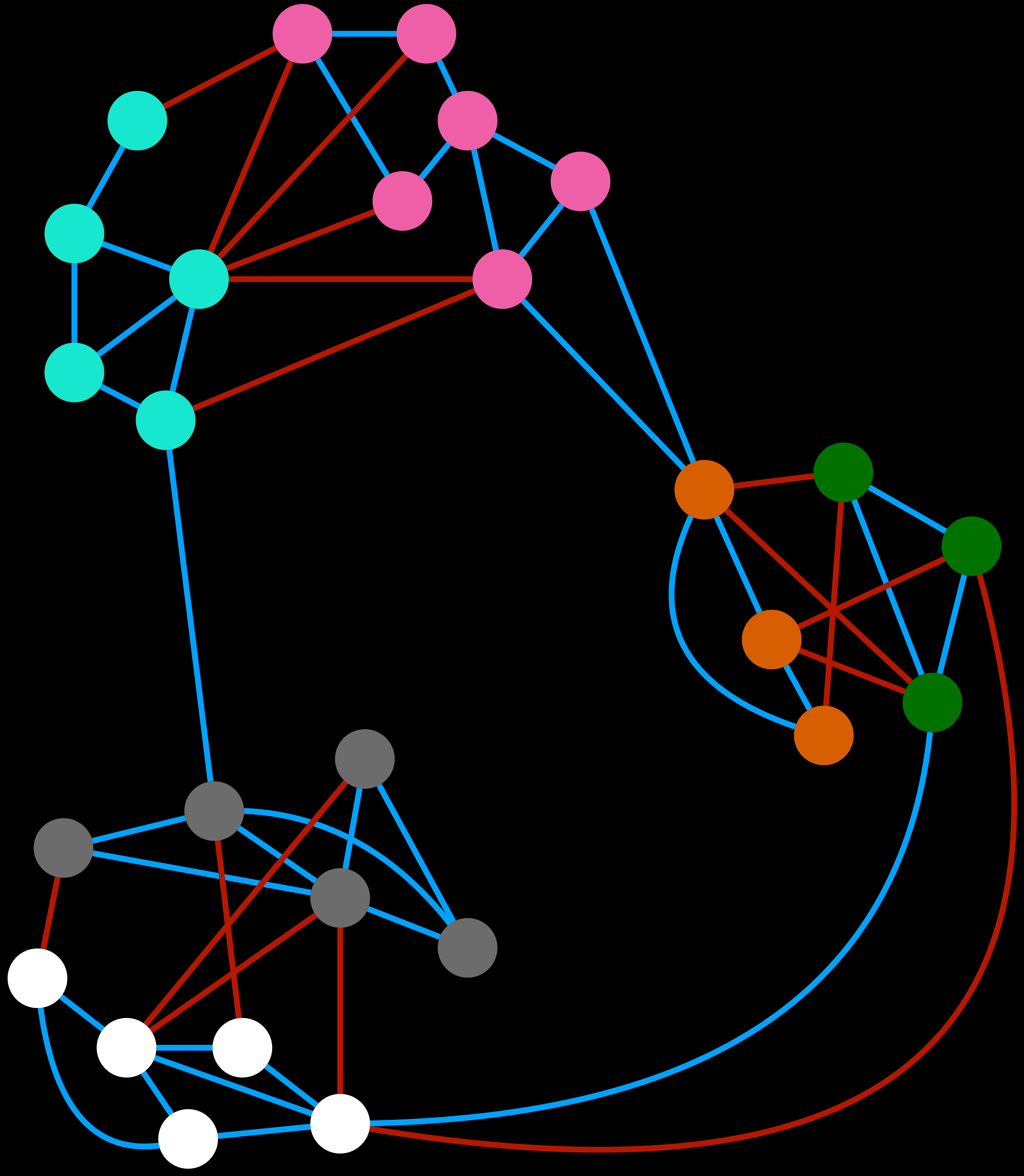
Sublinear-Time Algorithms

- Real-world graphs are huge and clustering the entire graph is often infeasible



Sublinear-Time Algorithms

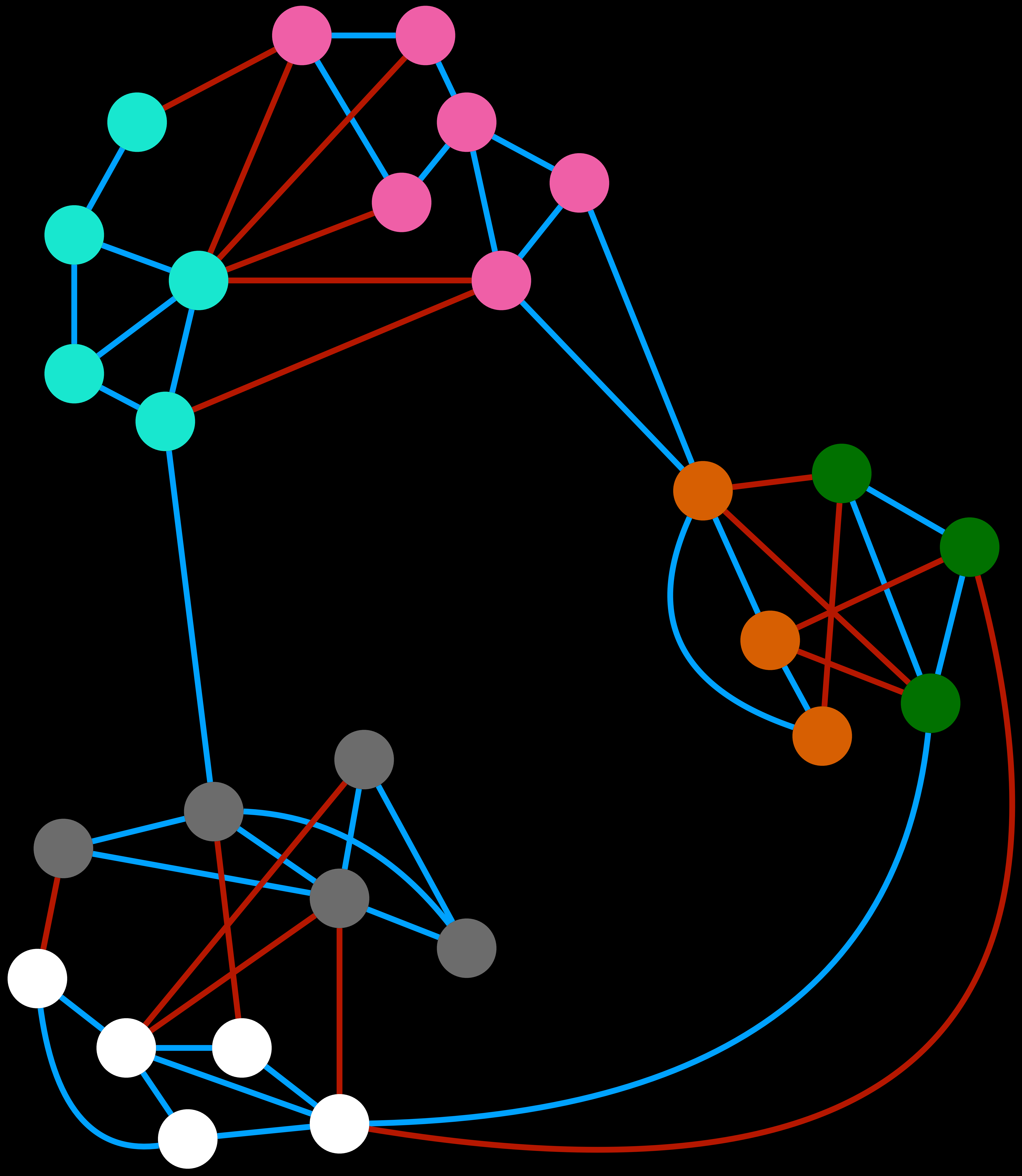
- Real-world graphs are huge and clustering the entire graph is often infeasible
- Often we do not have access to the full graph (e.g., Twitter)



Sublinear-Time Algorithms

- Real-world graphs are huge and clustering the entire graph is often infeasible
- Often we do not have access to the full graph (e.g., Twitter)

➡ **We cannot read the full graph**

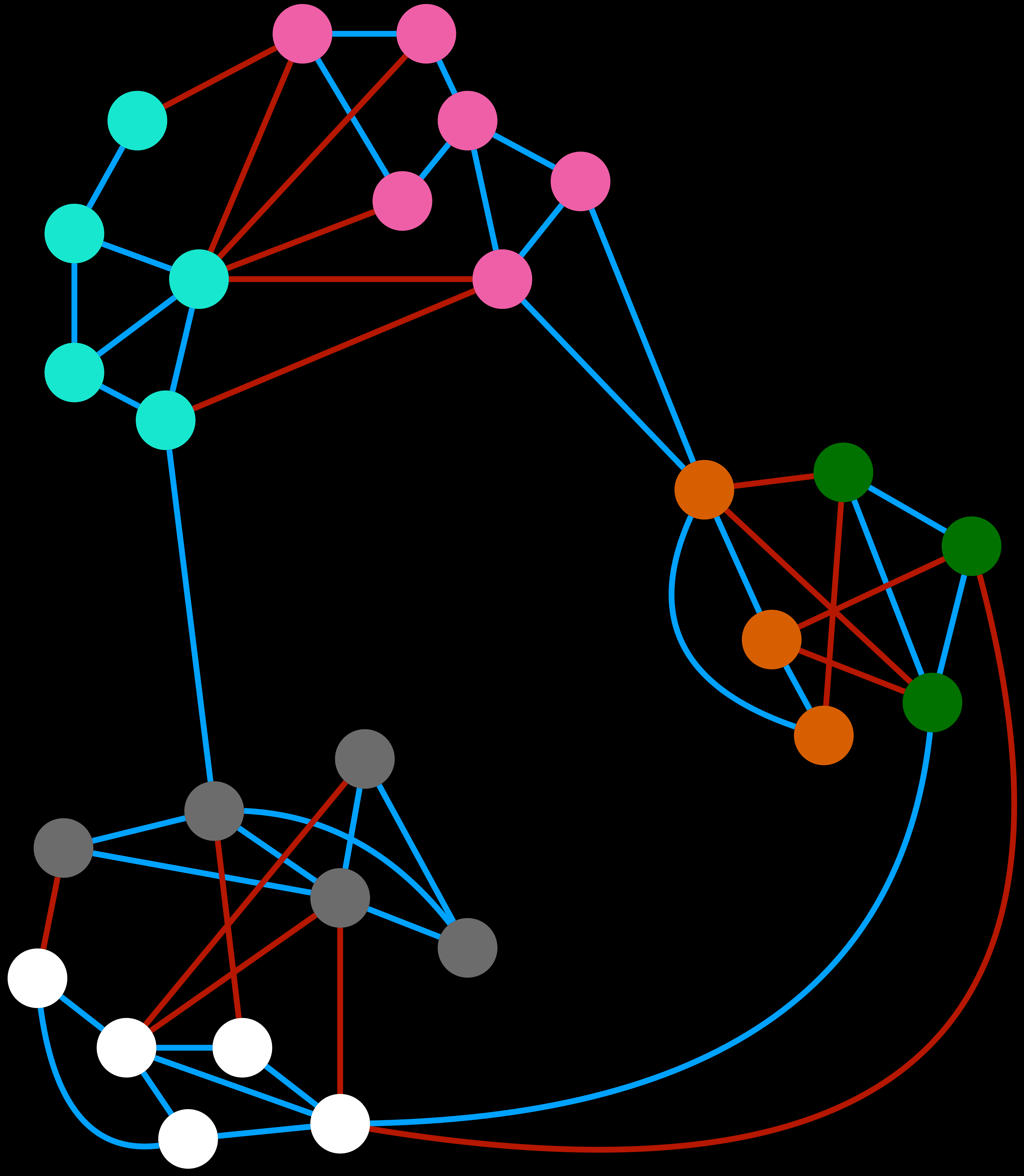


Sublinear-Time Algorithms

- Real-world graphs are huge and clustering the entire graph is often infeasible
- Often we do not have access to the full graph (e.g., Twitter)

➡ **We cannot read the full graph**

- We assume query-access to the graph, i.e., we can sample random vertices and random neighbors

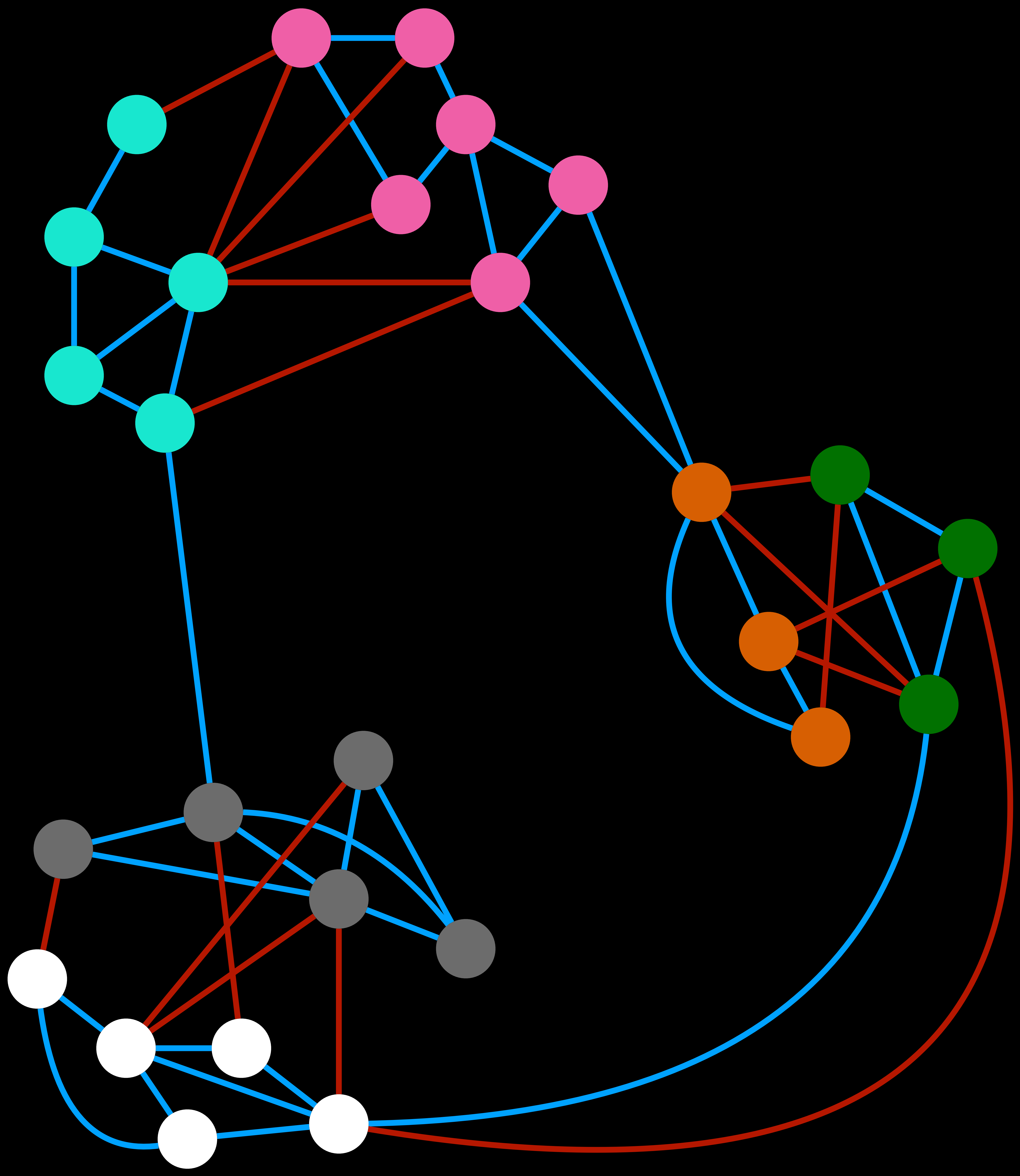


Sublinear-Time Algorithms

- Real-world graphs are huge and clustering the entire graph is often infeasible
- Often we do not have access to the full graph (e.g., Twitter)

➡ **We cannot read the full graph**

- We assume query-access to the graph, i.e., we can sample random vertices and random neighbors
- In many applications we need the cluster information only for a few vertices

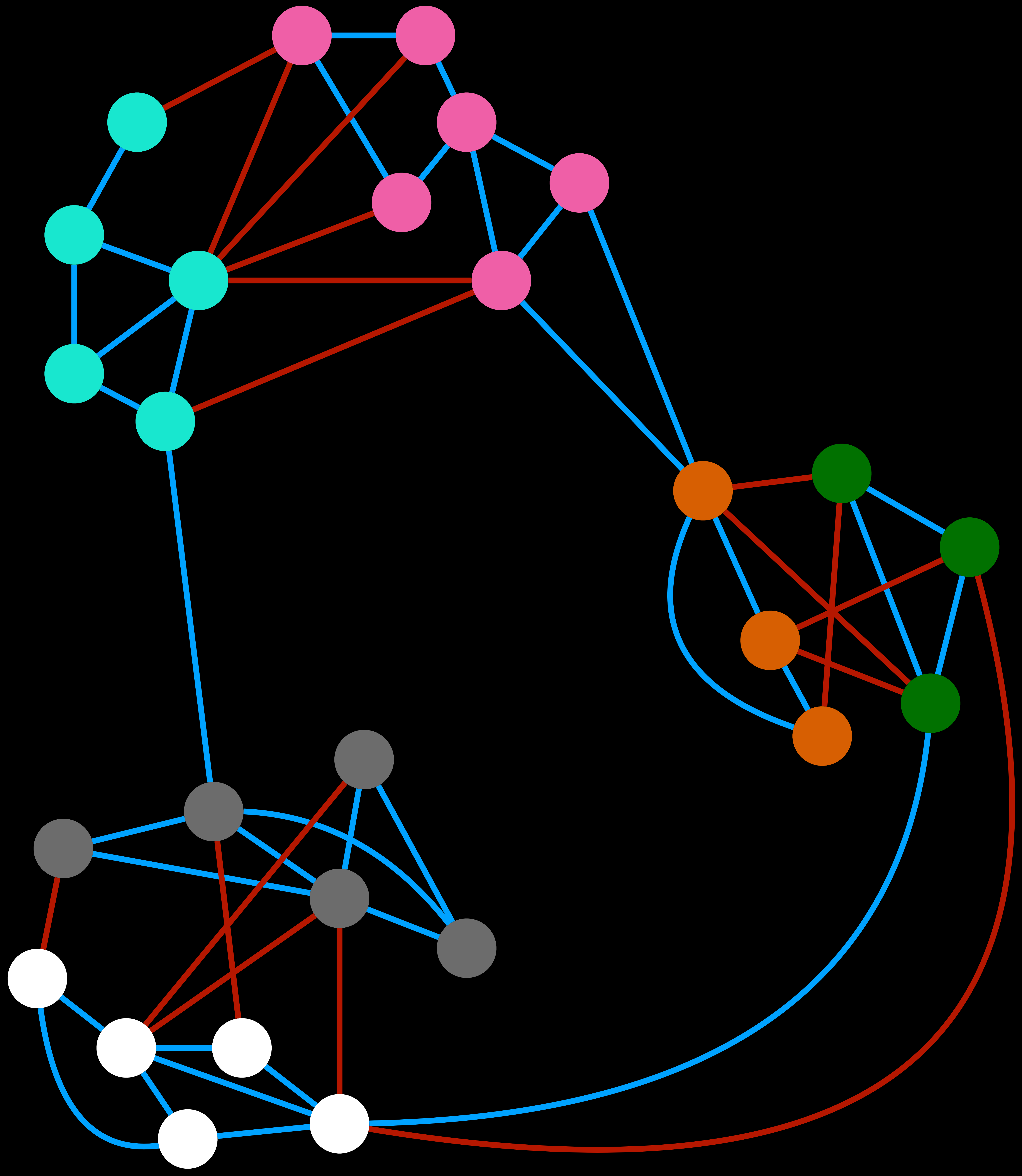


Sublinear-Time Algorithms

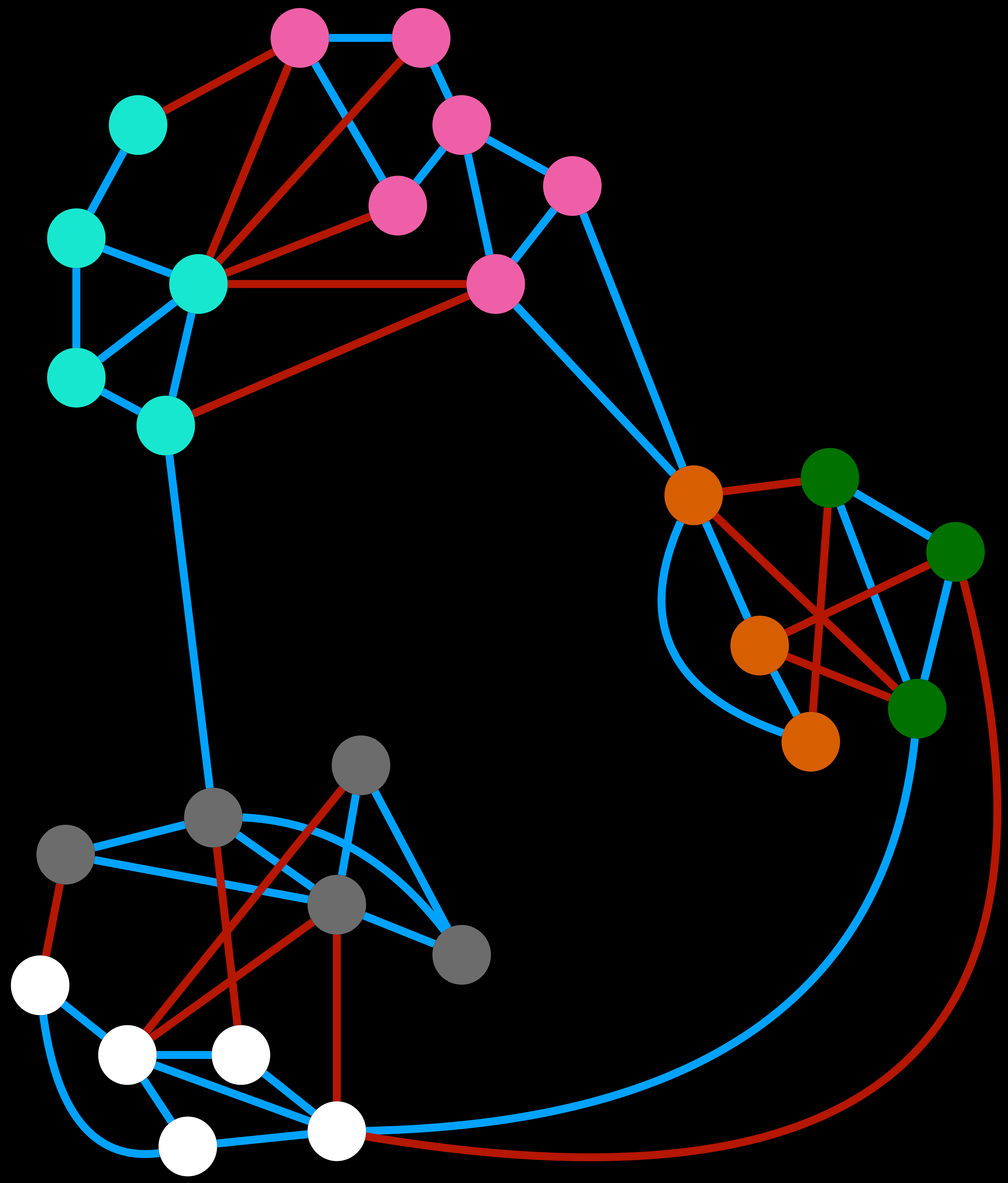
- Real-world graphs are huge and clustering the entire graph is often infeasible
- Often we do not have access to the full graph (e.g., Twitter)

➡ **We cannot read the full graph**

- We assume query-access to the graph, i.e., we can sample random vertices and random neighbors
- In many applications we need the cluster information only for a few vertices
- Must be very fast and space efficient

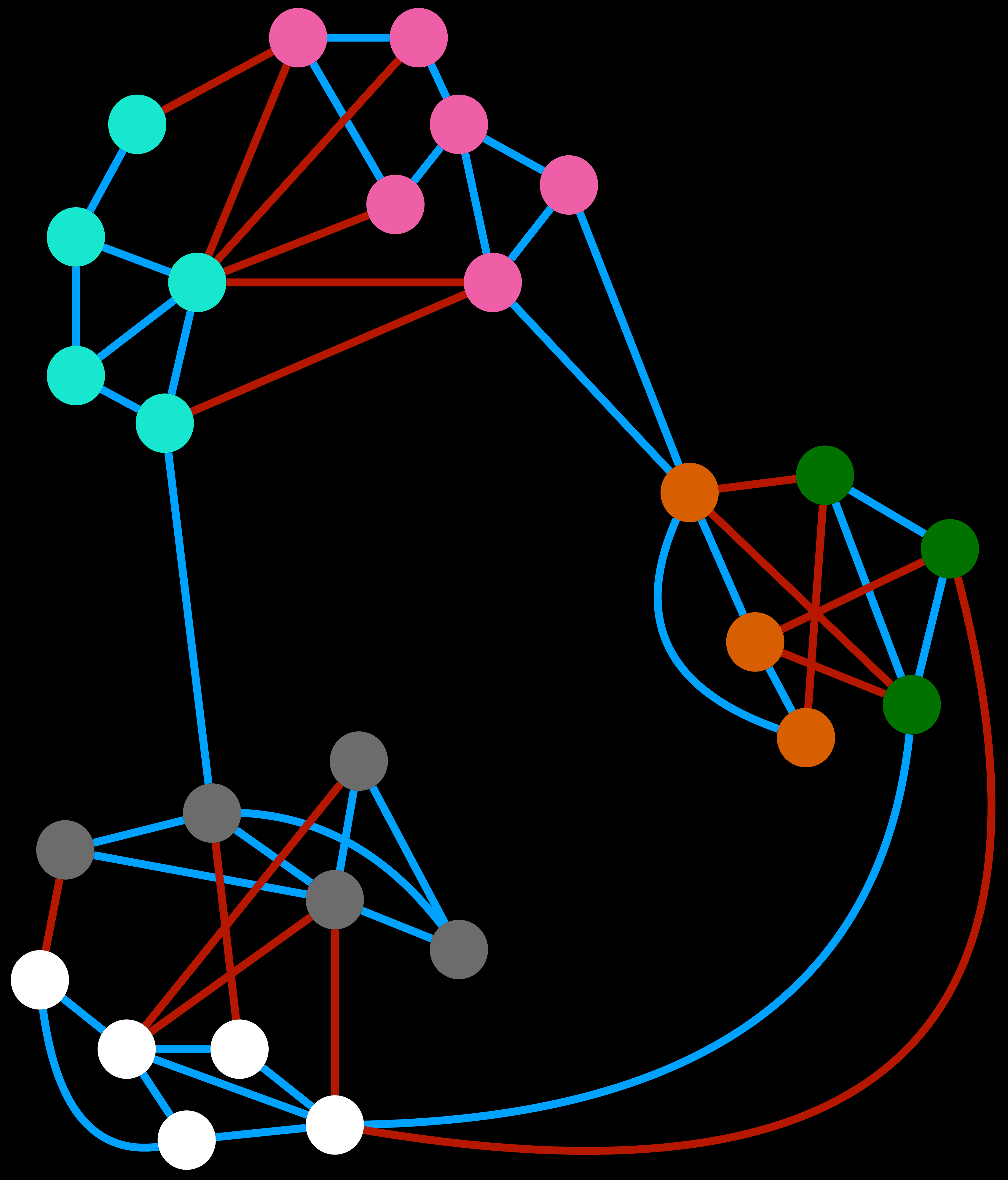


Our Signed Oracle



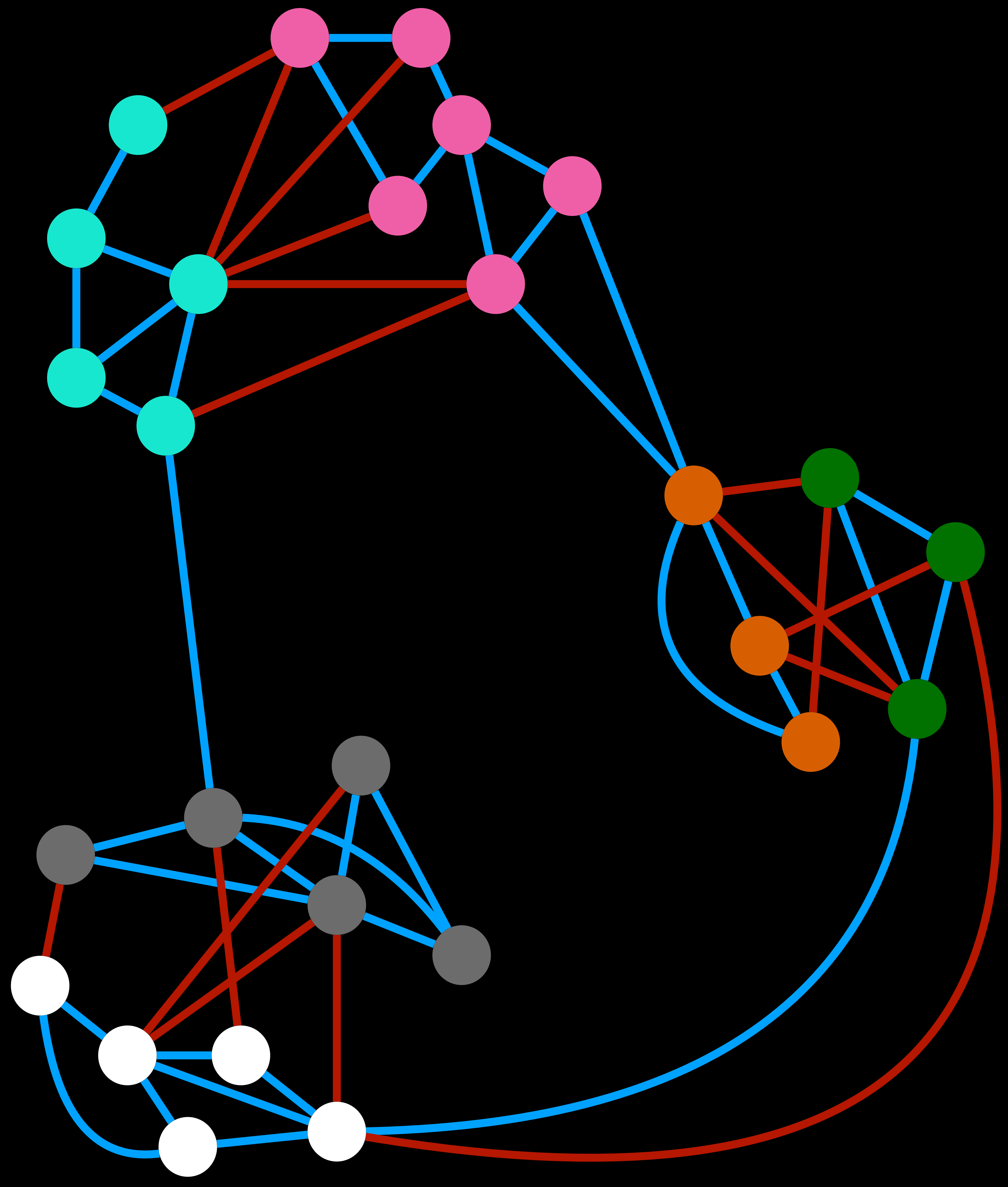
Our Signed Oracle

- Our oracle data structure allows the query:



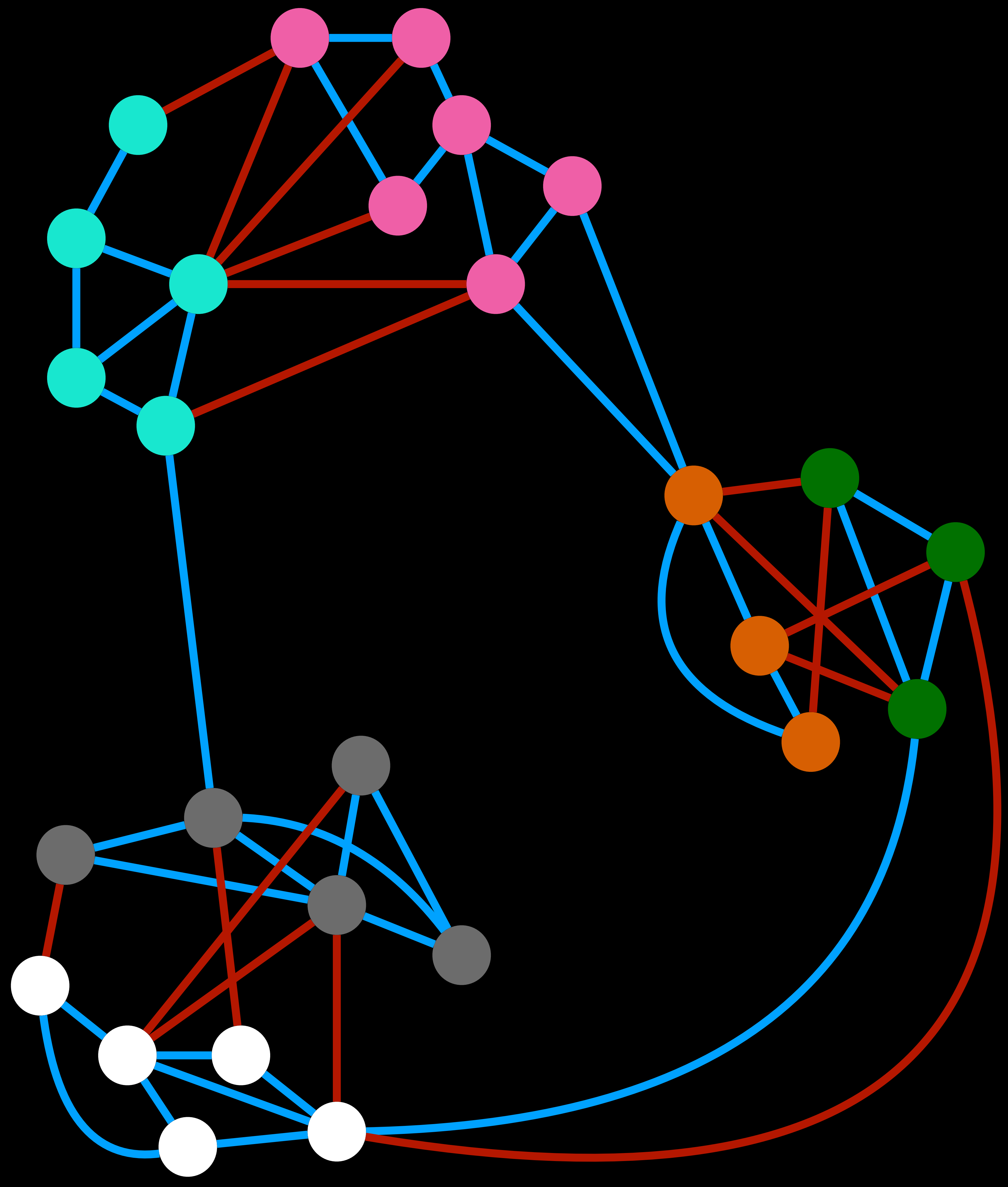
Our Signed Oracle

- Our oracle data structure allows the query:
  - Given a vertex u,
    which cluster does u belong to?



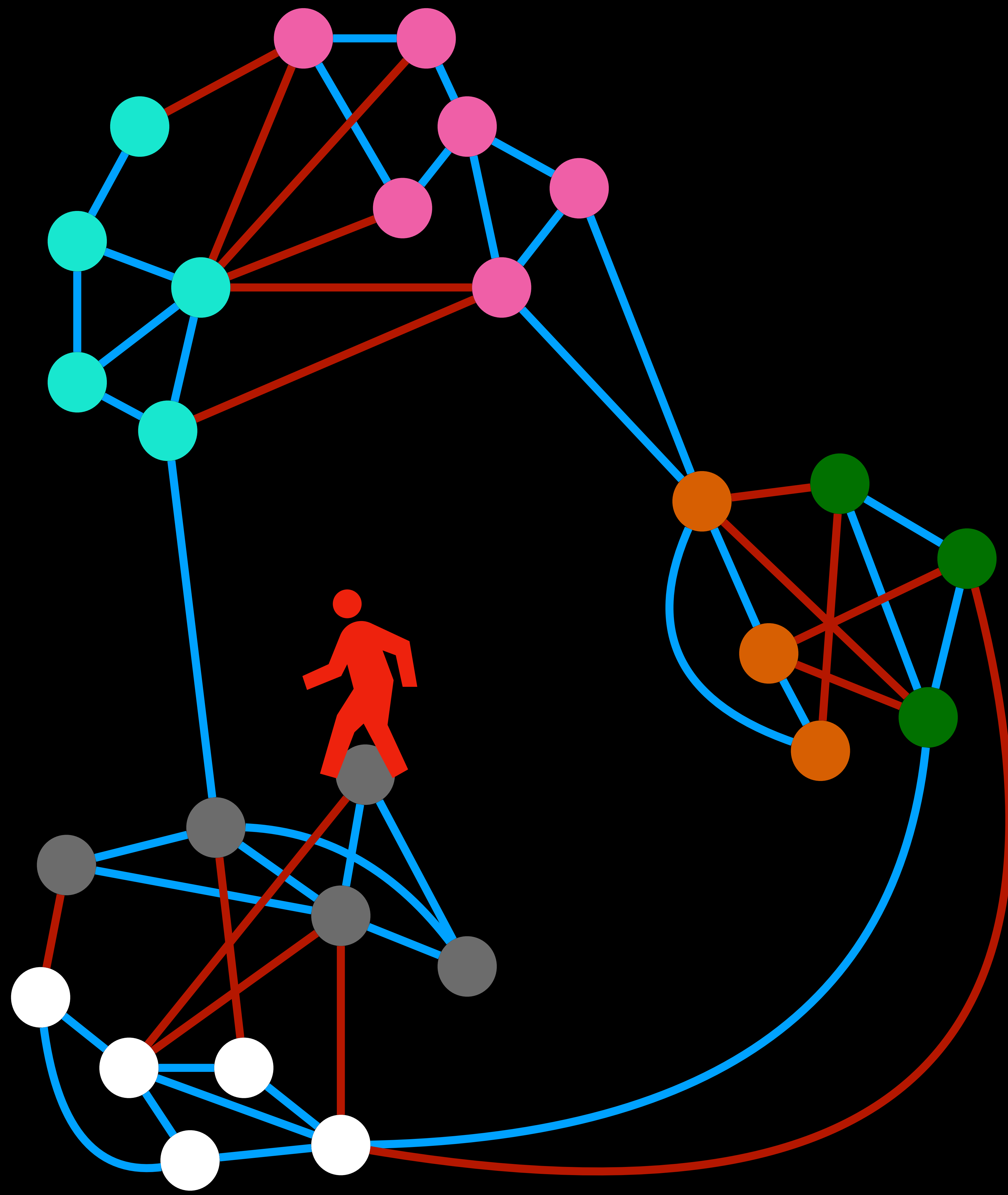
Our Signed Oracle

- Our oracle data structure allows the query:
  - Given a vertex u,
    which cluster does u belong to?
- To classify a vertex,
  we perform a small number of signed random walks



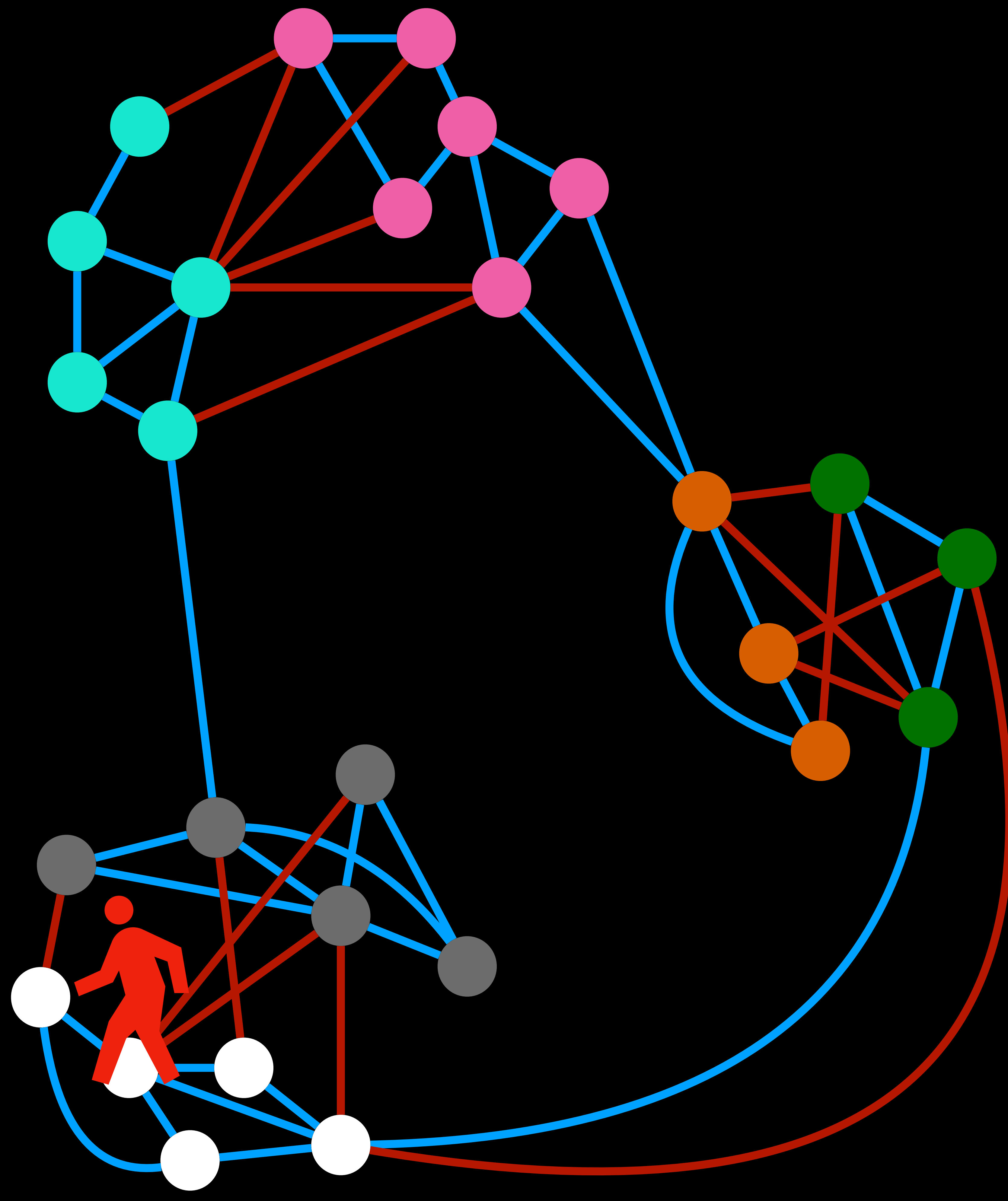
Our Signed Oracle

- Our oracle data structure allows the query:
  - Given a vertex u,
    which cluster does u belong to?
- To classify a vertex,
  we perform a small number of signed random walks



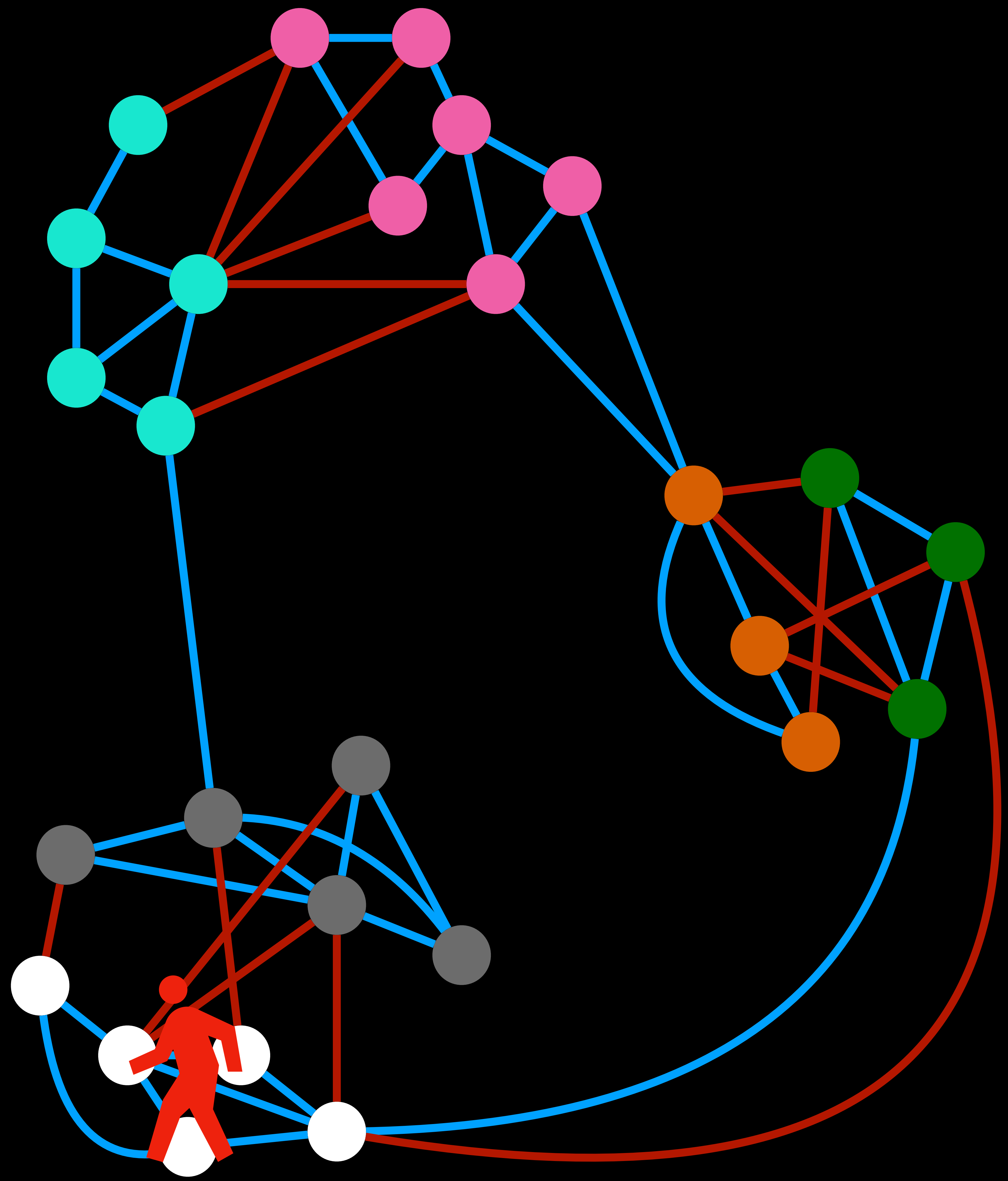
Our Signed Oracle

- Our oracle data structure allows the query:
    - Given a vertex u,
      which cluster does u belong to?
- To classify a vertex,
  we perform a small number of signed random walks



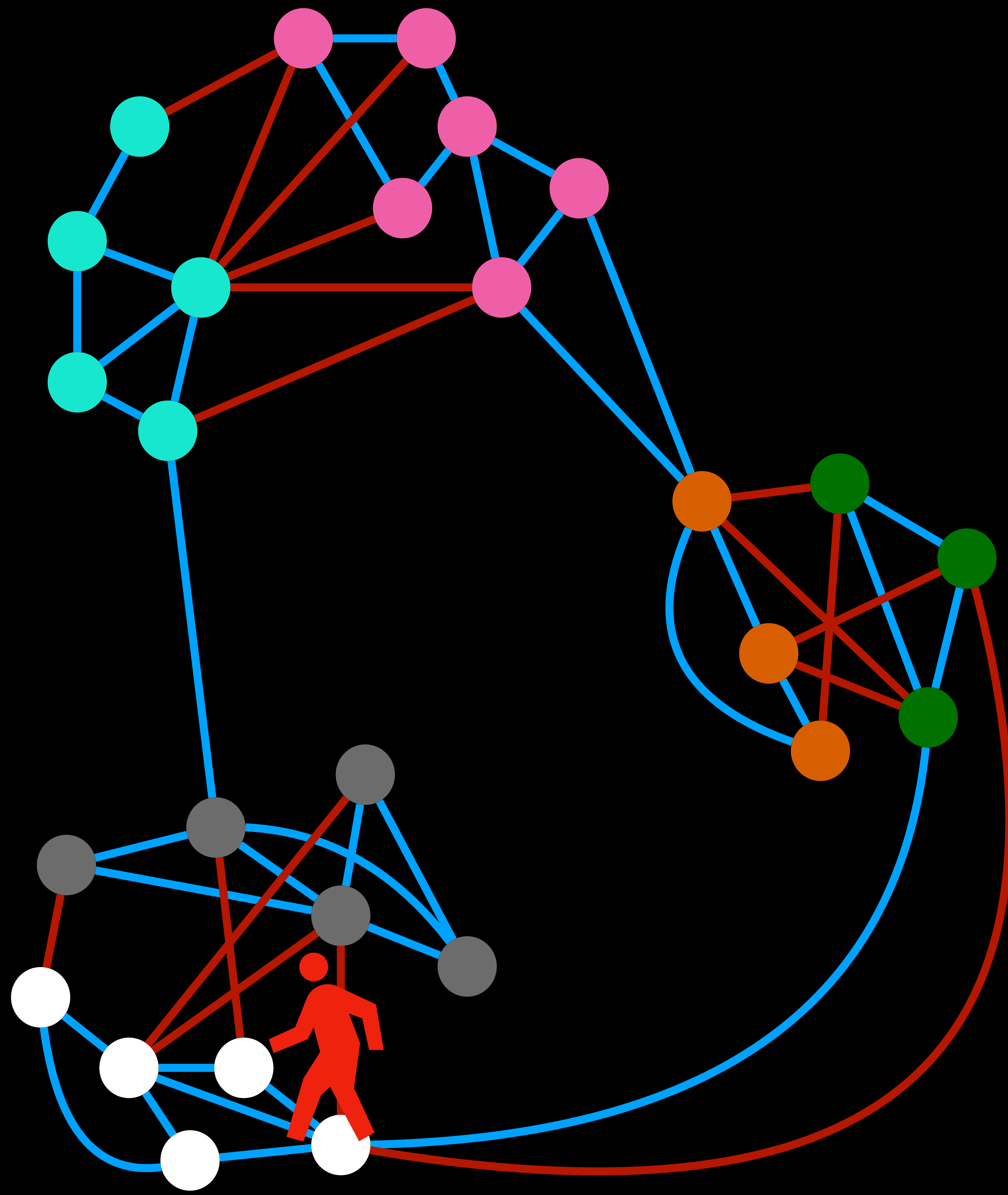
Our Signed Oracle

- Our oracle data structure allows the query:
    - Given a vertex u,
      which cluster does u belong to?
- To classify a vertex,
  we perform a small number of signed random walks



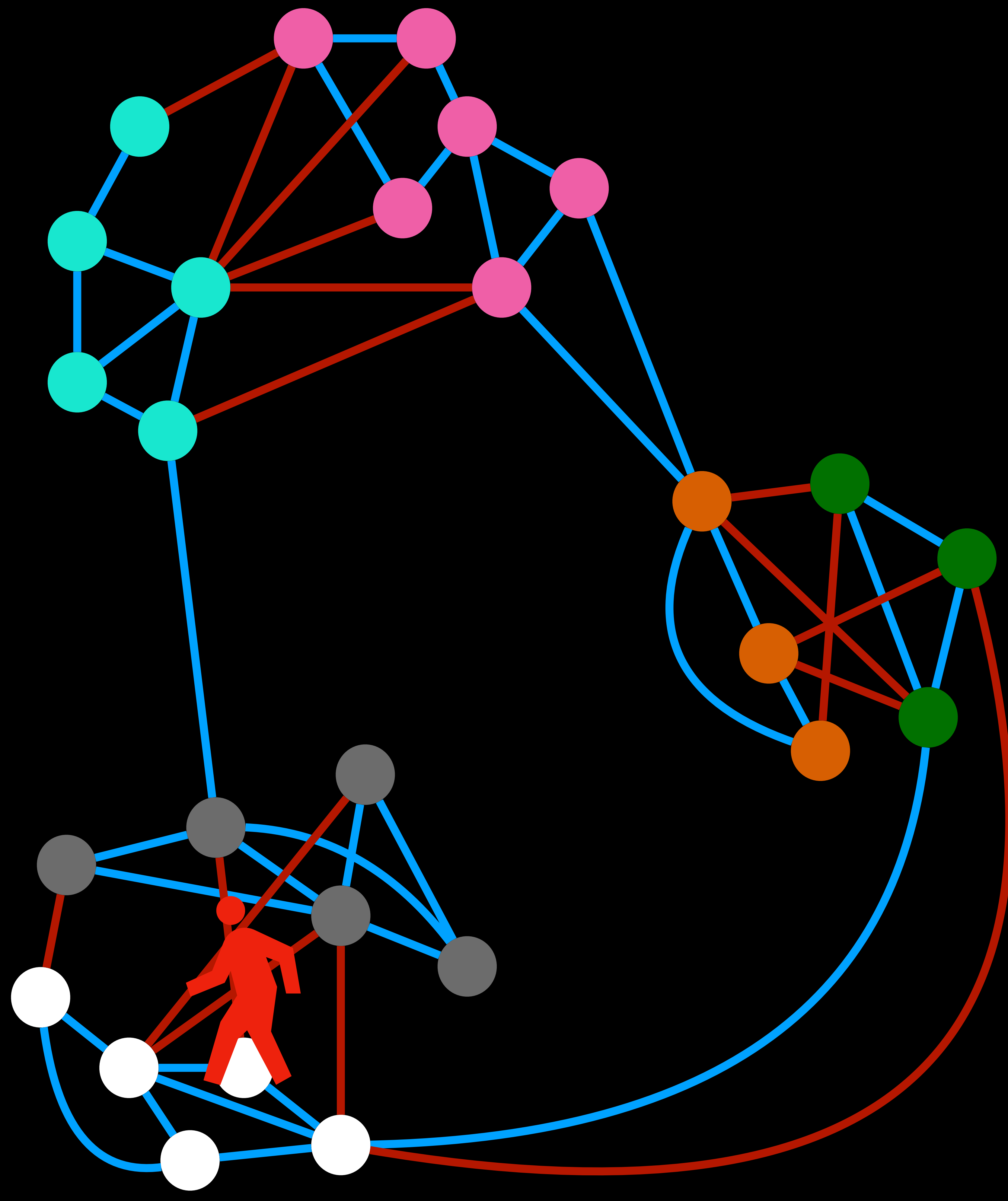
Our Signed Oracle

- Our oracle data structure allows the query:
  - Given a vertex u,
    which cluster does u belong to?
- To classify a vertex,
  we perform a small number of signed random walks



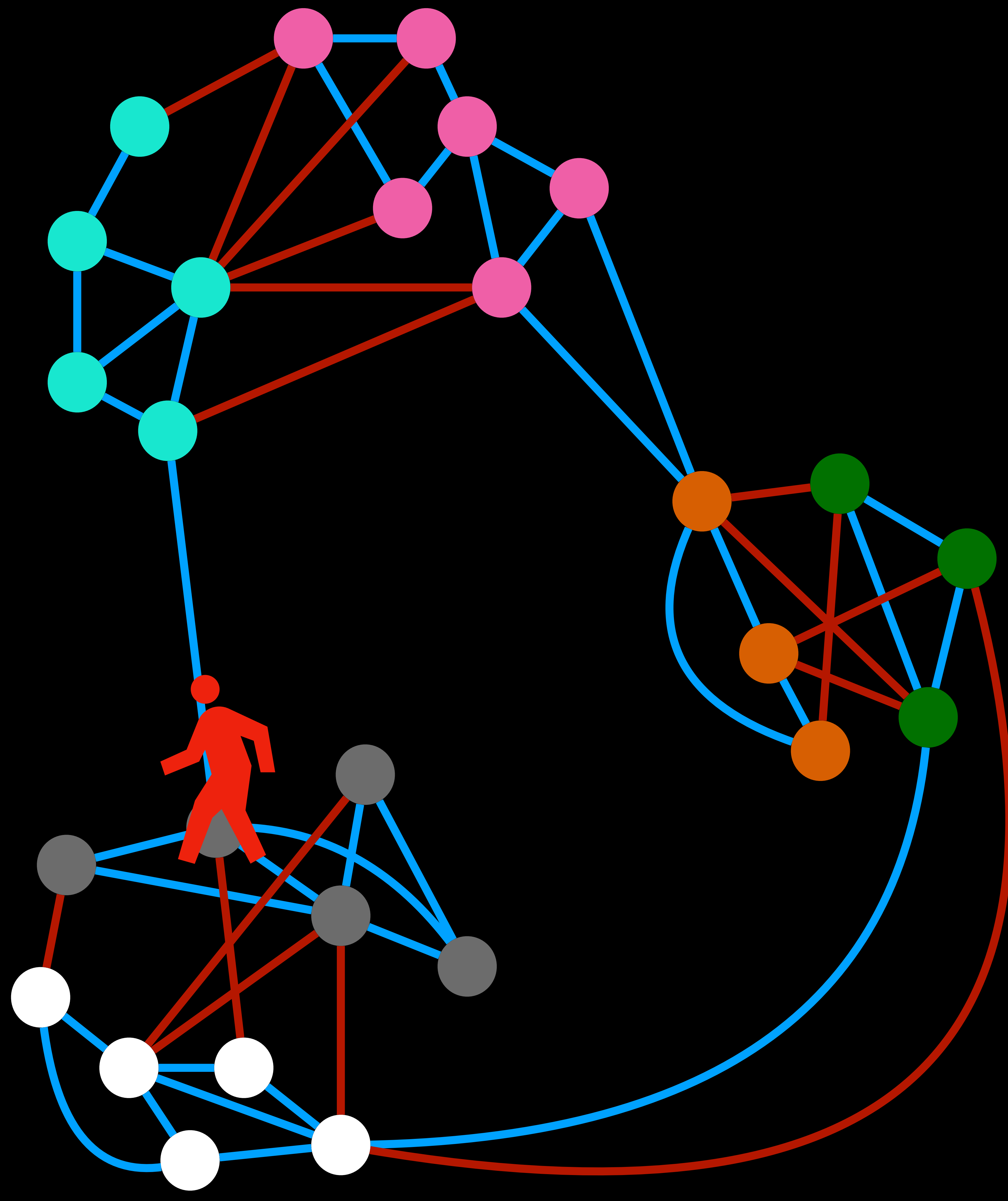
Our Signed Oracle

- Our oracle data structure allows the query:
  - Given a vertex u,
    which cluster does u belong to?
- To classify a vertex,
  we perform a small number of signed random walks



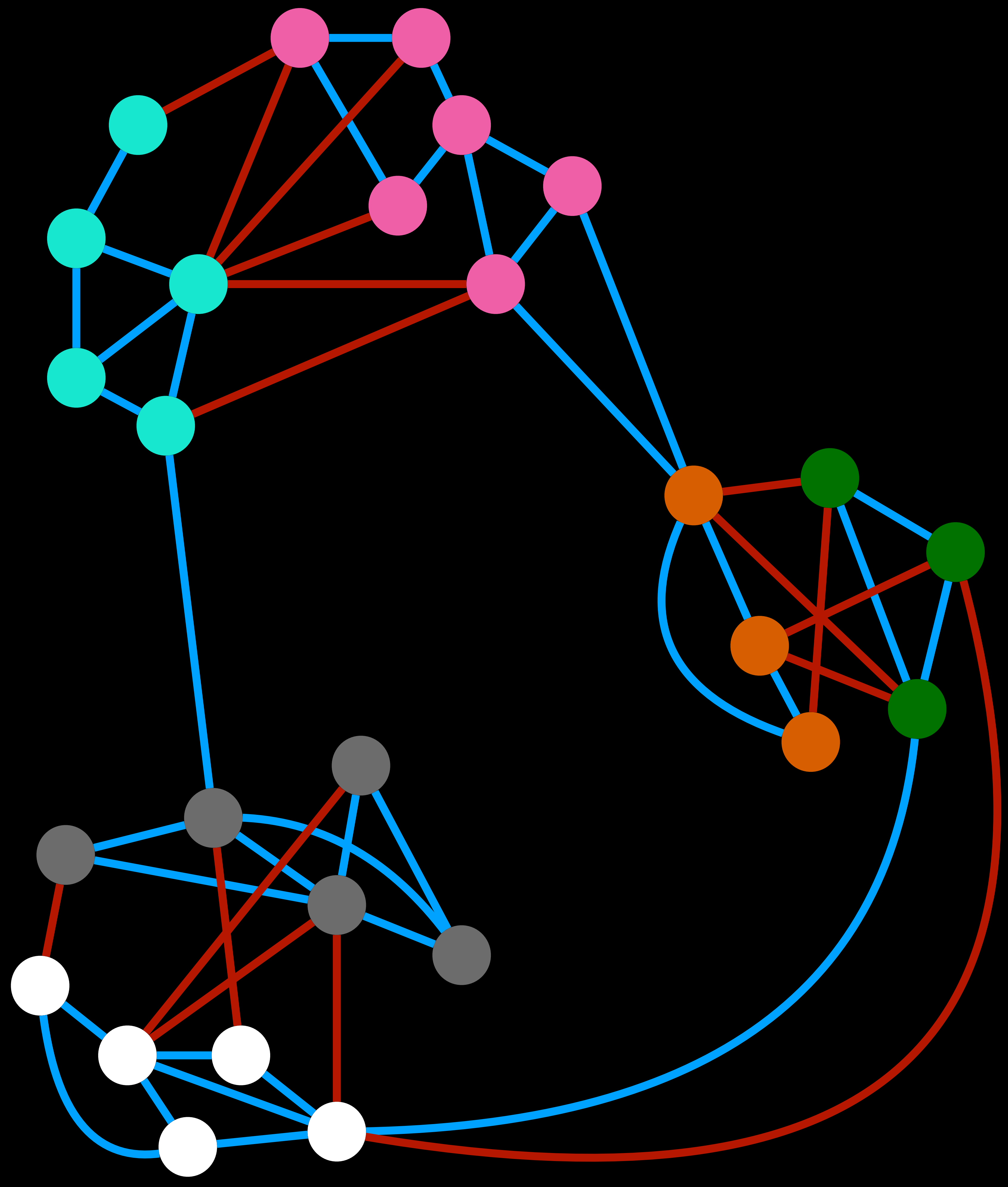
Our Signed Oracle

- Our oracle data structure allows the query:
  - Given a vertex u,
    which cluster does u belong to?
- To classify a vertex,
  we perform a small number of signed random walks



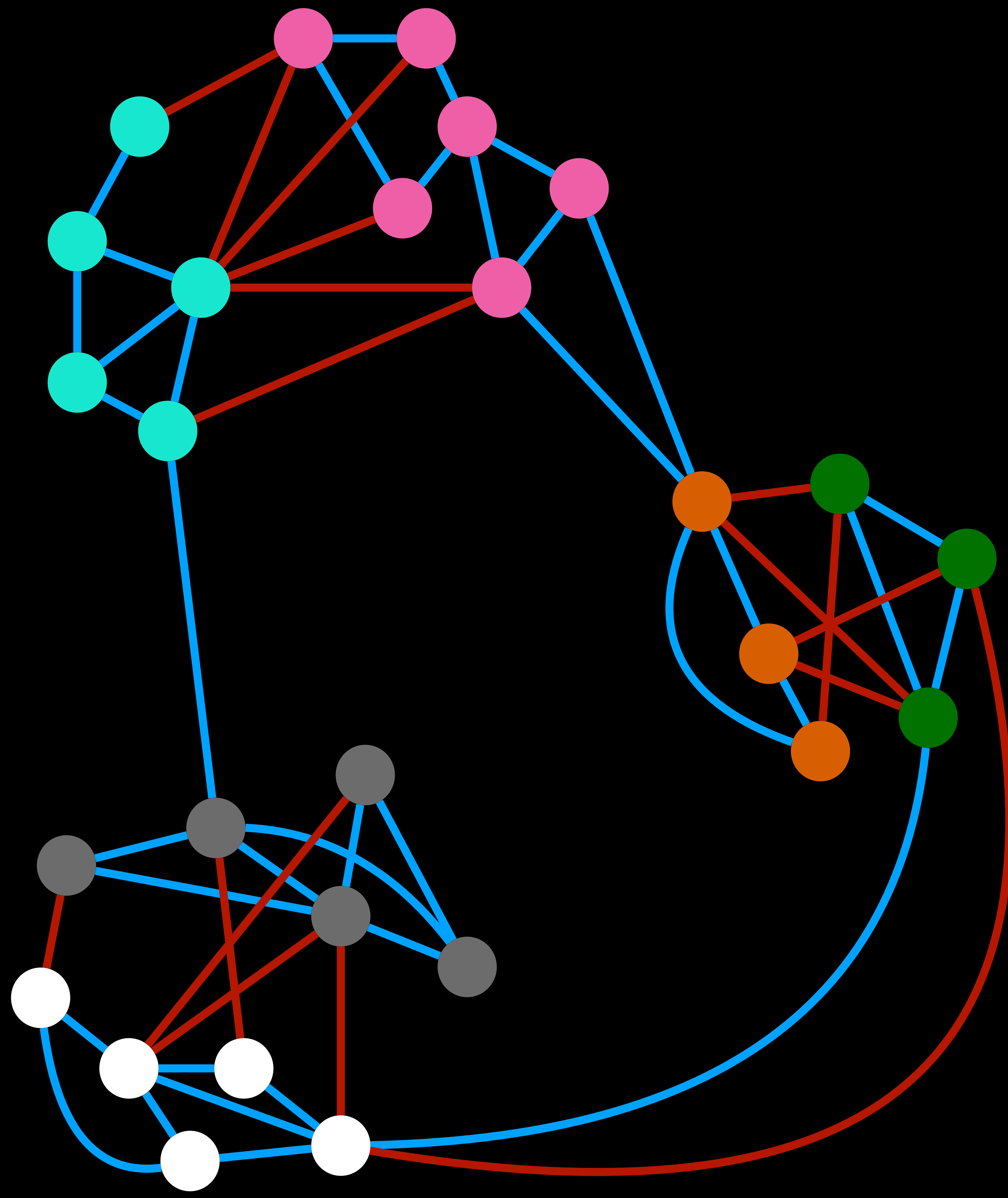
Our Signed Oracle

- Our oracle data structure allows the query:
  - Given a vertex u,
    which cluster does u belong to?
- To classify a vertex,
  we perform a small number of signed random walks



Our Signed Oracle

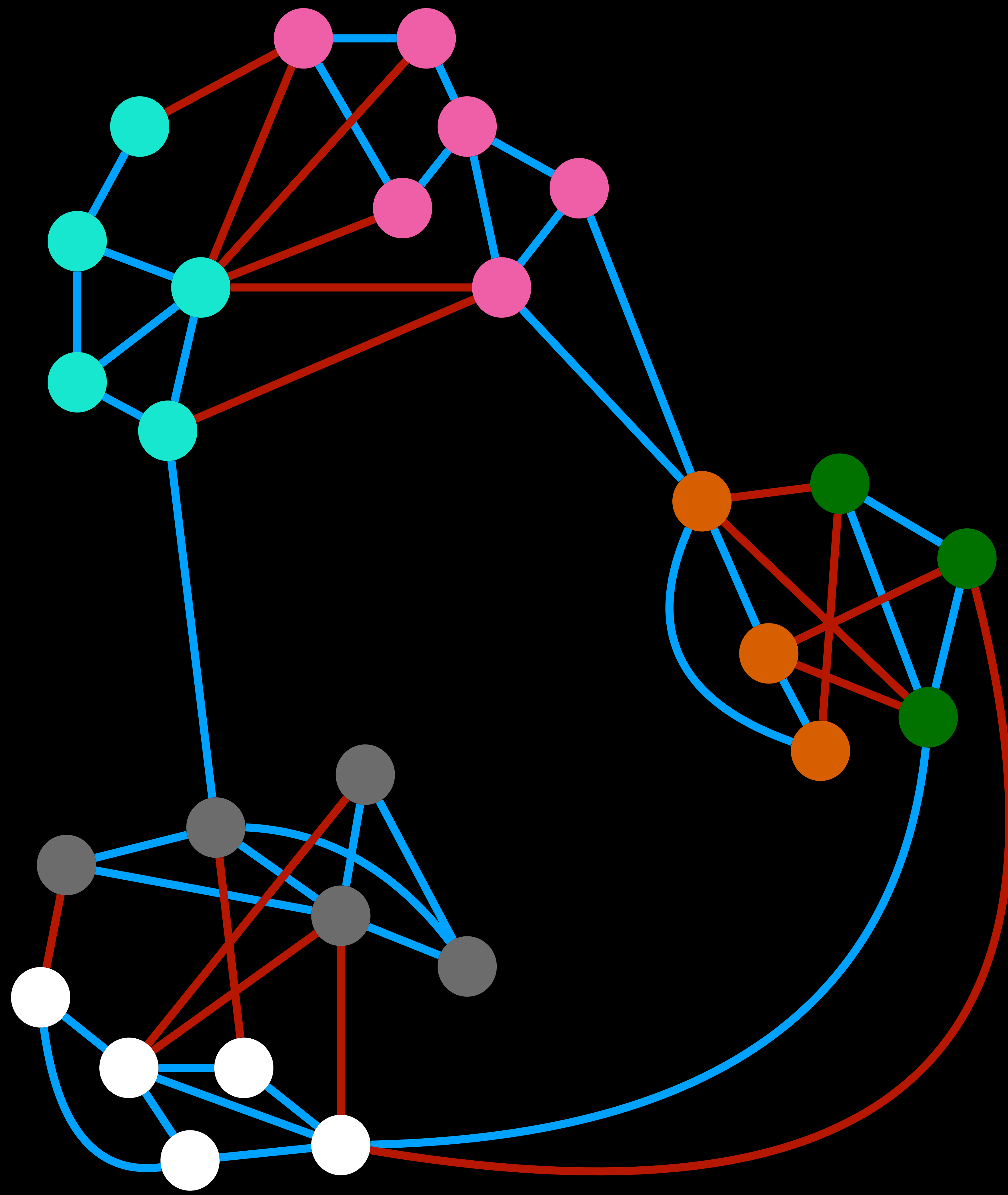
- Our oracle data structure allows the query:
  - Given a vertex u,
    which cluster does u belong to?
- To classify a vertex,
  we perform a small number of signed random walks
- Each vertex can be classified in sublinear time and space



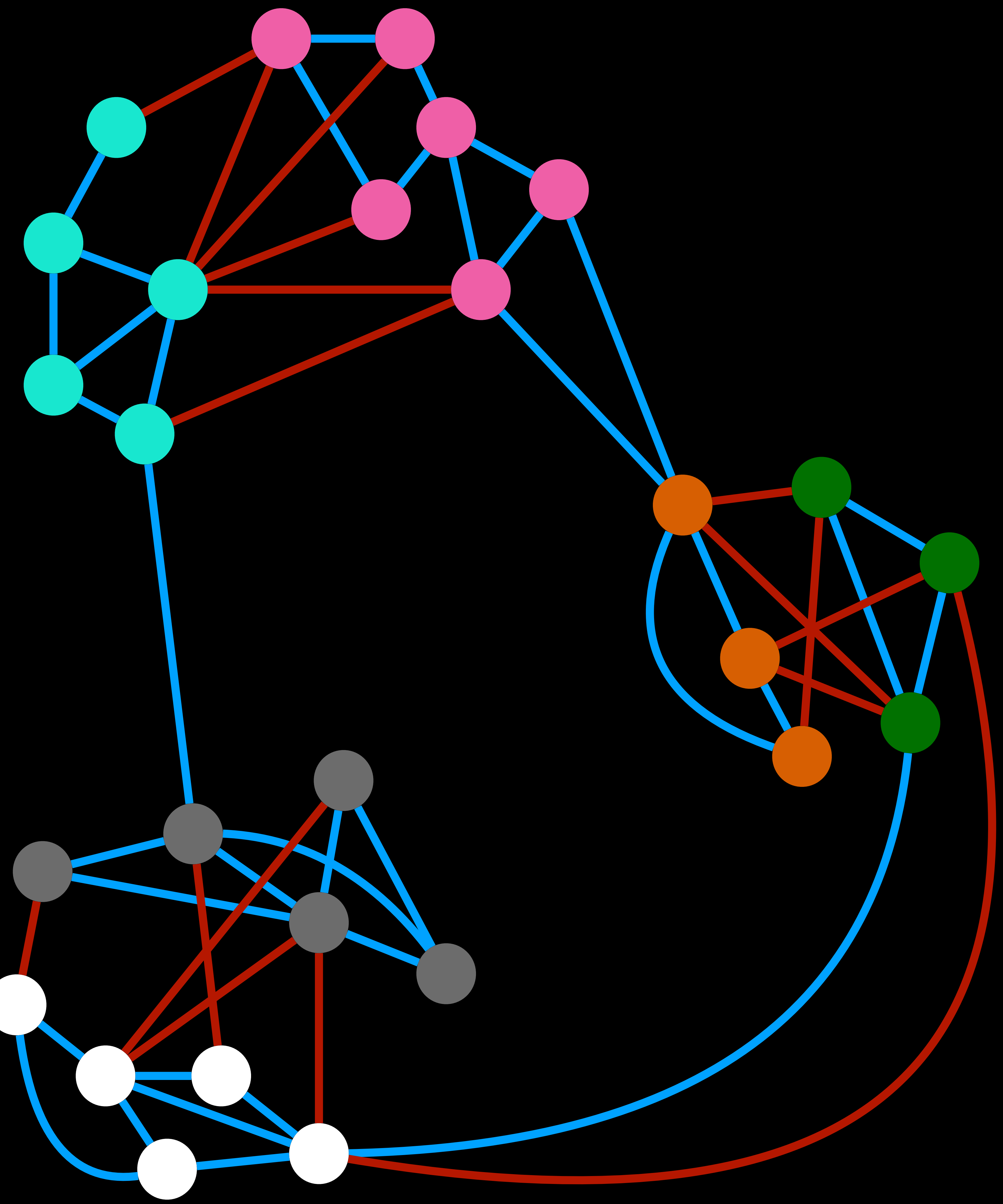
Our Signed Oracle

- Our oracle data structure allows the query:
  - Given a vertex u,
    which cluster does u belong to?
- To classify a vertex,
  we perform a small number of signed random walks
- Each vertex can be classified in sublinear time and space

  ➡ Very efficient when only a few vertices need to be classified

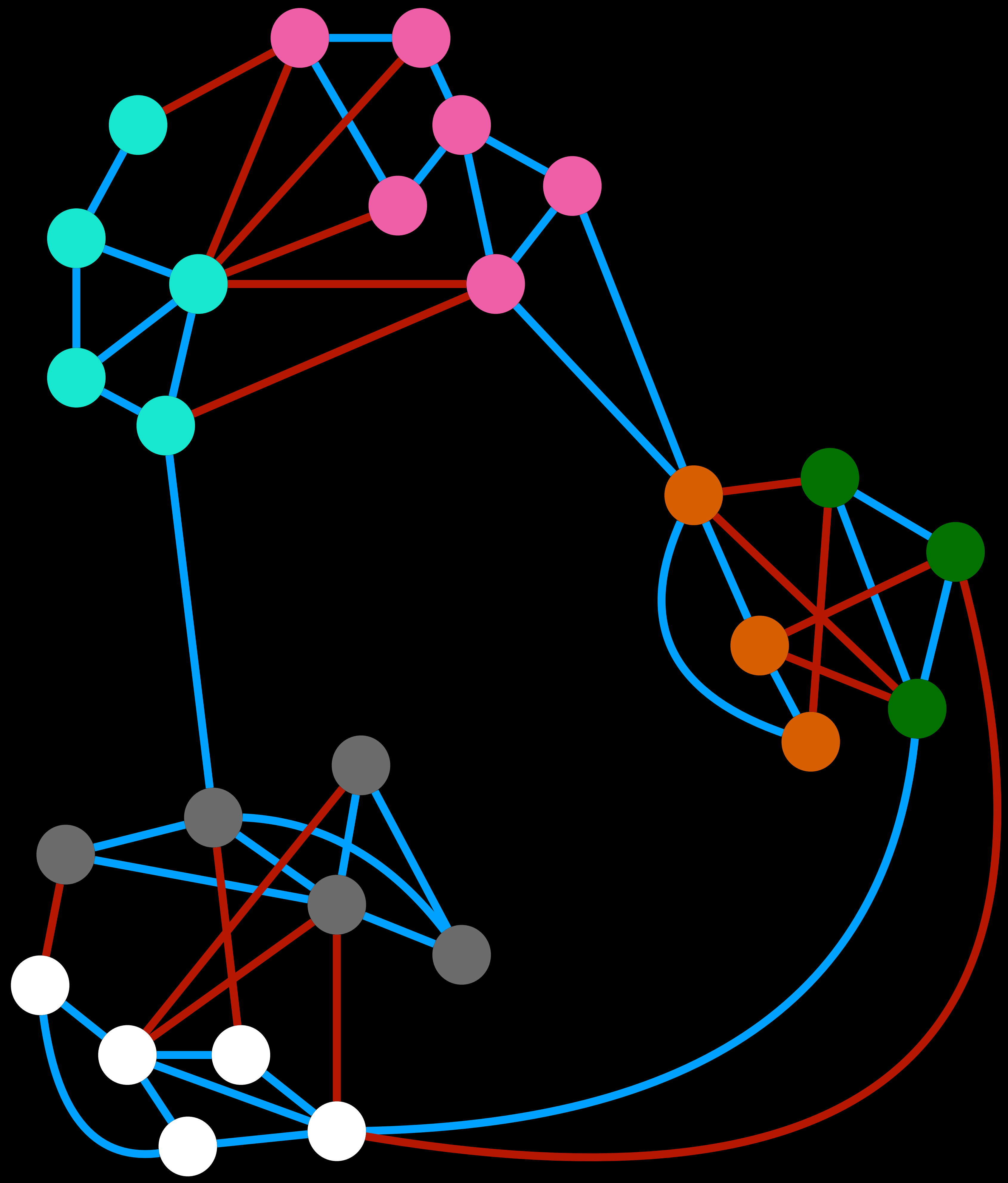


We Can Provably Recover Planted Clusters



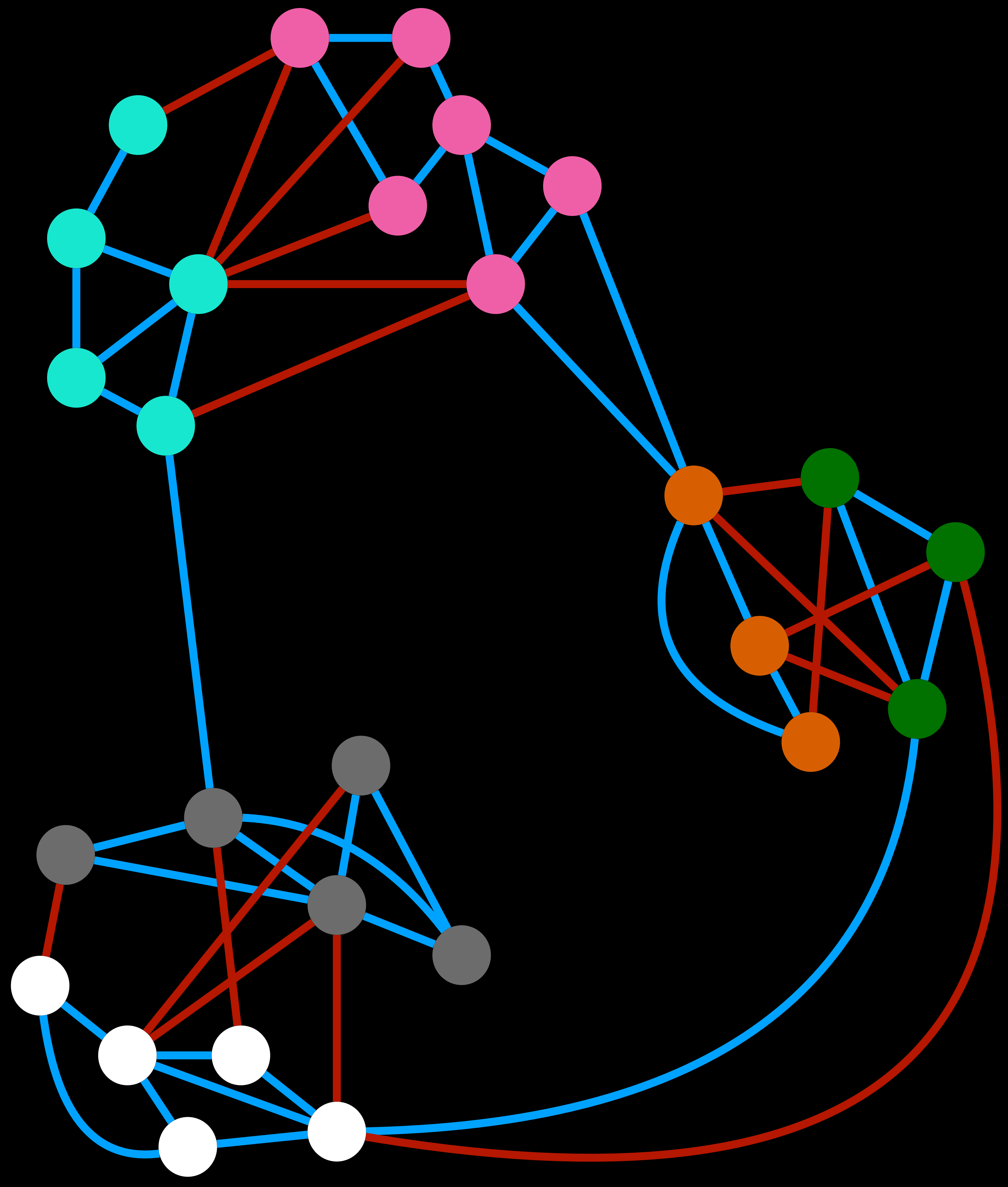
We Can Provably Recover Planted Clusters

- Consider graph with n vertices



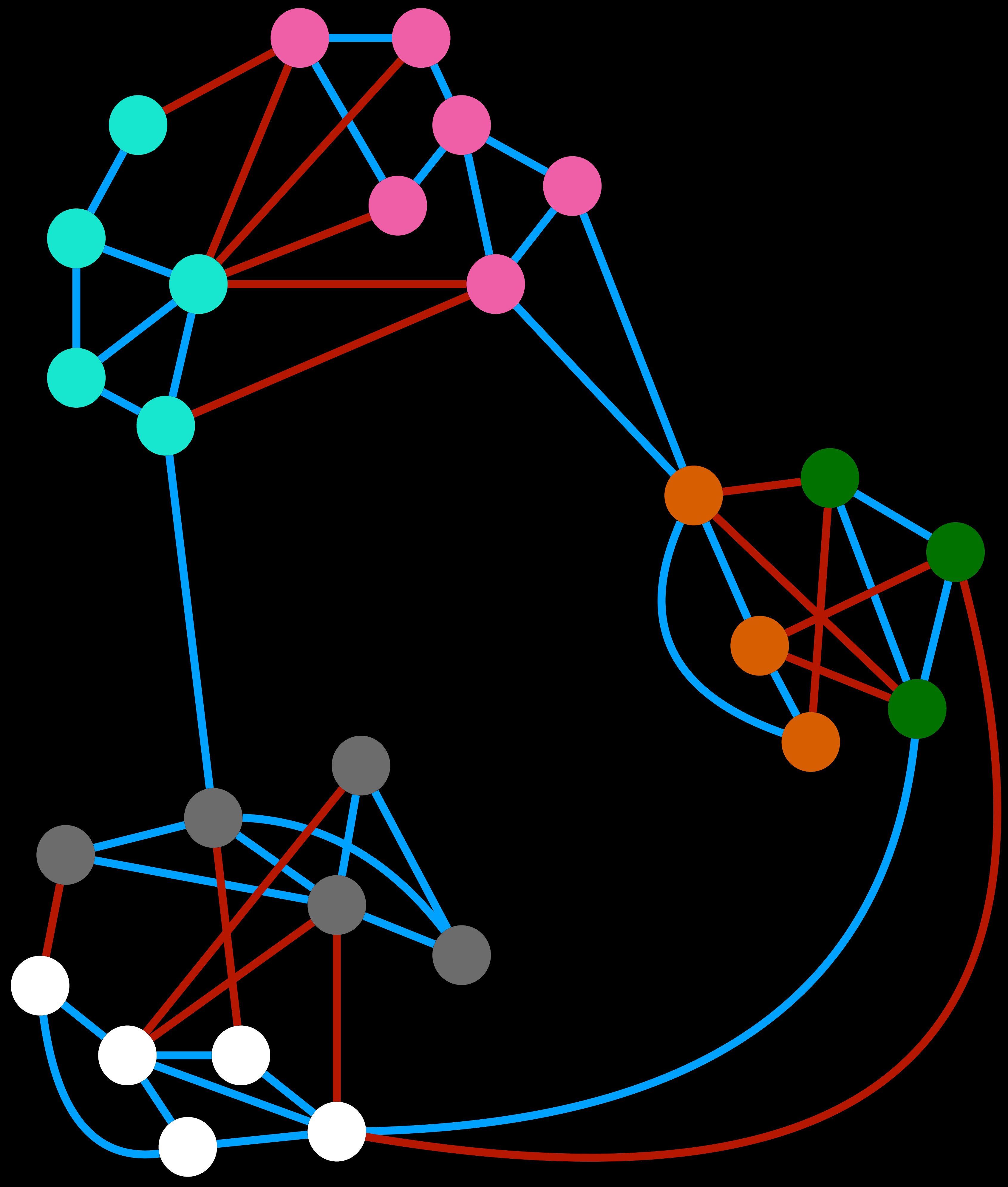
We Can Provably Recover Planted Clusters

- Consider graph with n vertices
- Assumptions:
  bounded degrees, $\tilde{O}(1)$ clusters, “nice” planted clusters



We Can Provably Recover Planted Clusters

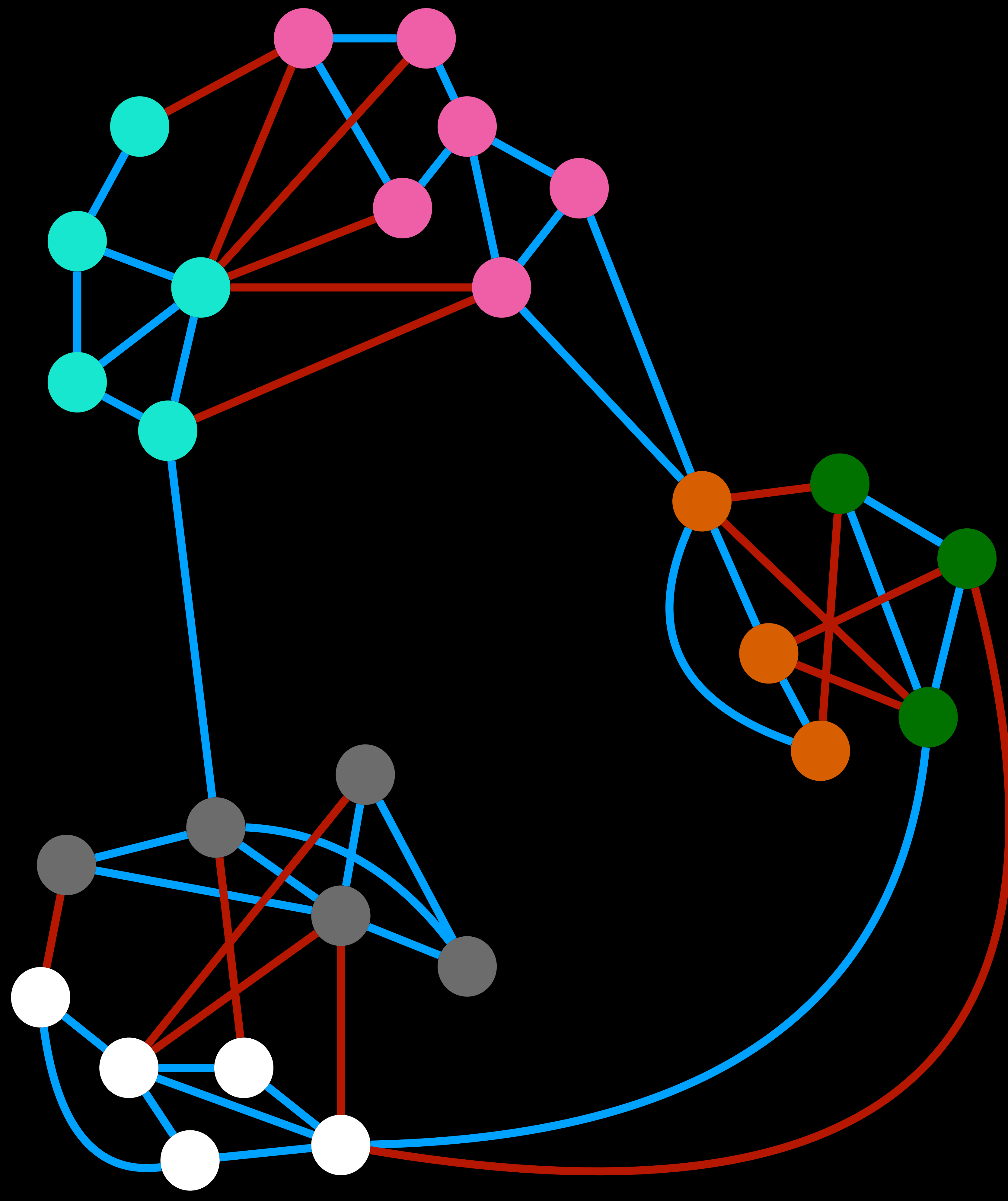
- Consider graph with n vertices
- Assumptions:
  bounded degrees, $\tilde{O}(1)$ clusters, “nice” planted clusters
- Theorem (informal):
  We can return the cluster index of a vertex with query time $\tilde{O}(\sqrt{n})$.
  The answer is $(1-\varepsilon)$-close to the planted clustering with probability at least 90%.



We Can Provably Recover Planted Clusters

- Consider graph with n vertices
- Assumptions:
  bounded degrees, $\tilde{O}(1)$ clusters, “nice” planted clusters
- Theorem (informal):
  We can return the cluster index of a vertex with query time $\tilde{O}(\sqrt{n})$.
  The answer is $(1-\varepsilon)$-close to the planted clustering with probability at least 90%.

➡ Gives theoretical analysis for *random walks with signs* and provides new results for spectral graph theory of signed graphs

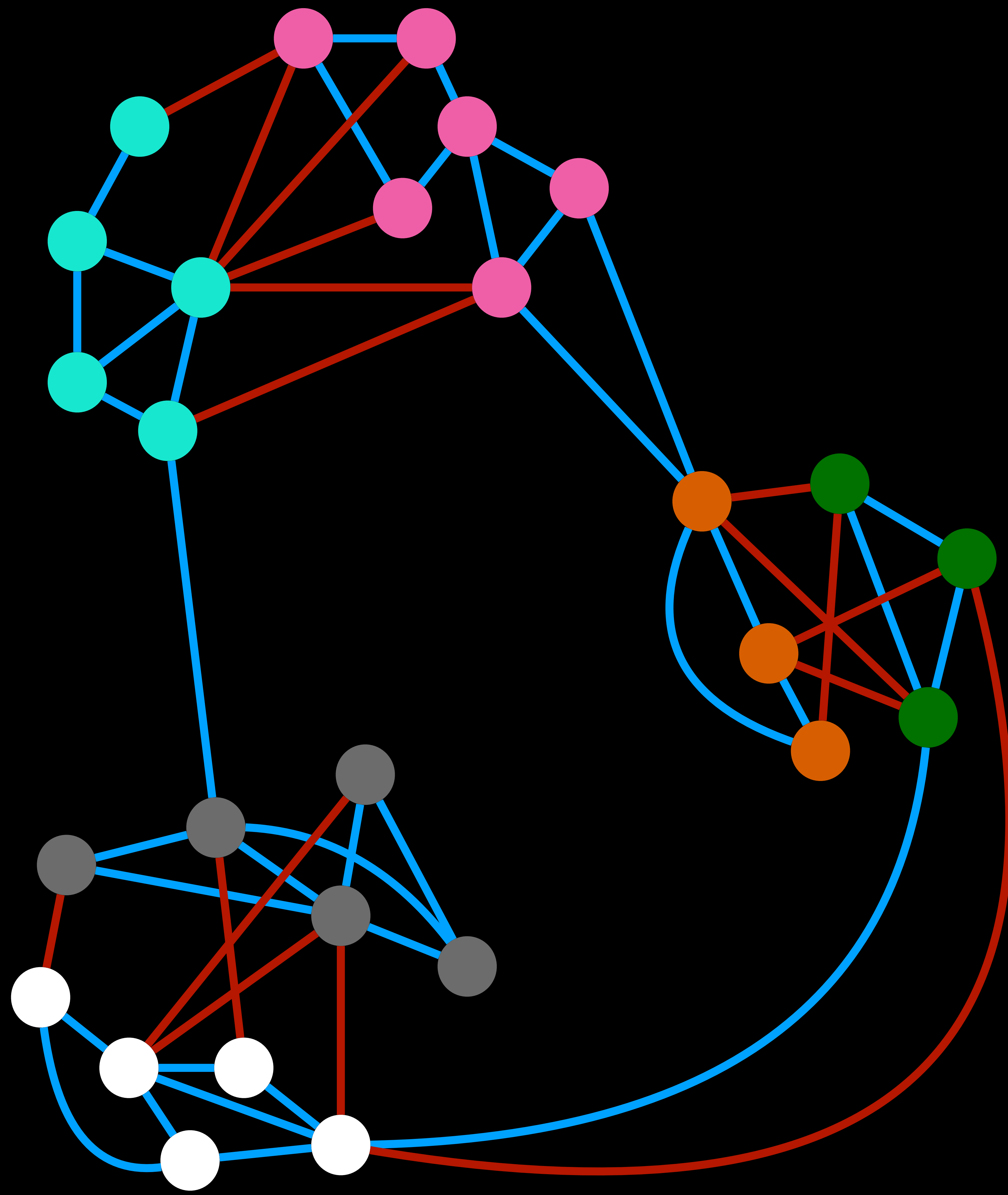


We Can Provably Recover Planted Clusters

- Consider graph with n vertices
- Assumptions:
  bounded degrees, $\tilde{O}(1)$ clusters, "nice" planted clusters
- Theorem (informal):
  We can return the cluster index of a vertex with query time $\tilde{O}(\sqrt{n})$.
  The answer is $(1-\varepsilon)$-close to the planted clustering with probability at least 90%.

  ➡ Gives theoretical analysis for *random walks with signs* and provides new results for spectral graph theory of signed graphs

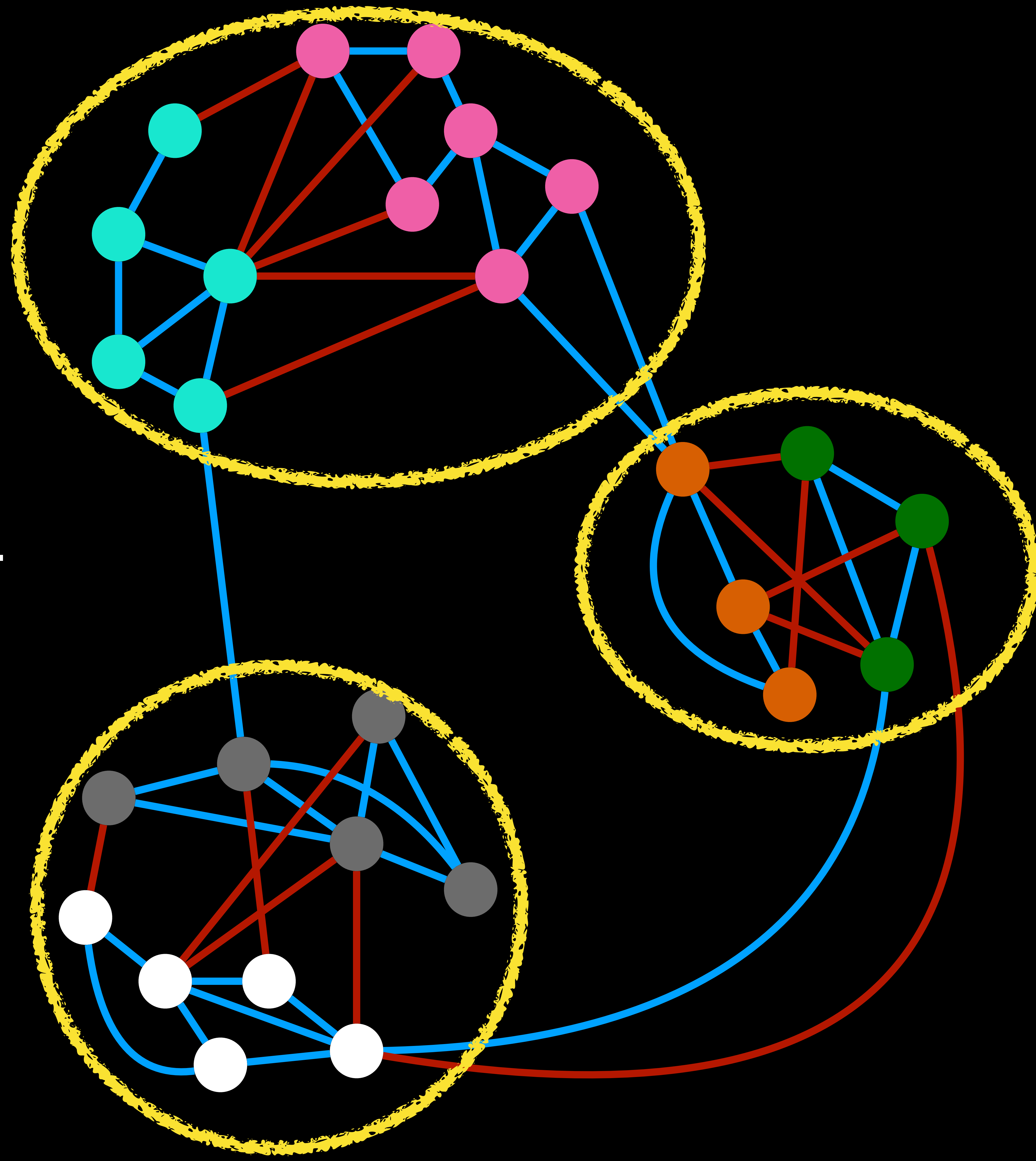
- Theoretical analysis for identifying conflicts, but not the sides



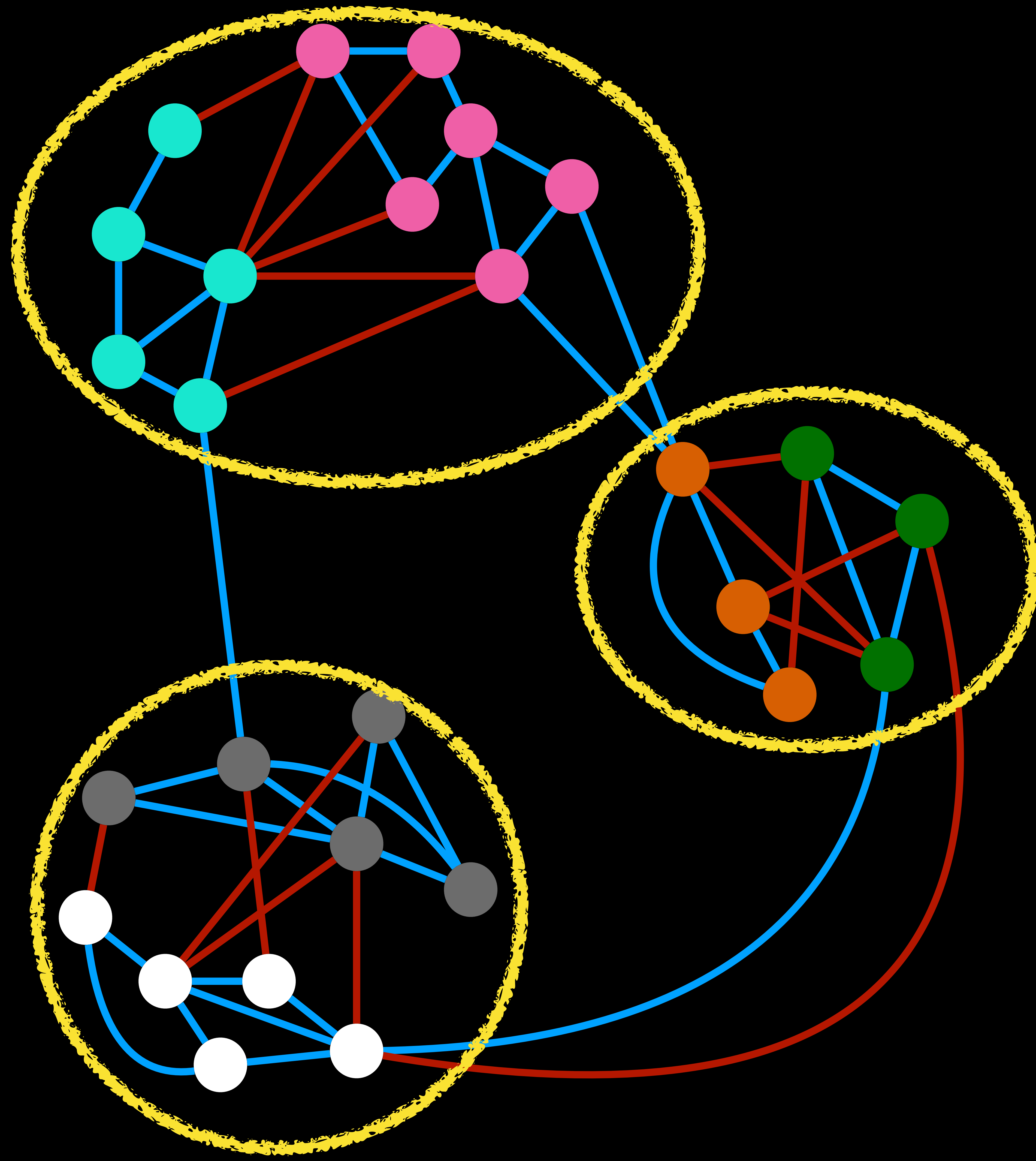
We Can Provably Recover Planted Clusters

- Consider graph with n vertices
- Assumptions:
  bounded degrees, $\tilde{O}(1)$ clusters, “nice” planted clusters
- Theorem (informal):
  We can return the cluster index of a vertex with query time $\tilde{O}(\sqrt{n})$.
  The answer is $(1-\varepsilon)$-close to the planted clustering with probability at least 90%.

  ➡ Gives theoretical analysis for *random walks with signs* and provides new results for spectral graph theory of signed graphs
- Theoretical analysis for identifying conflicts, but not the sides

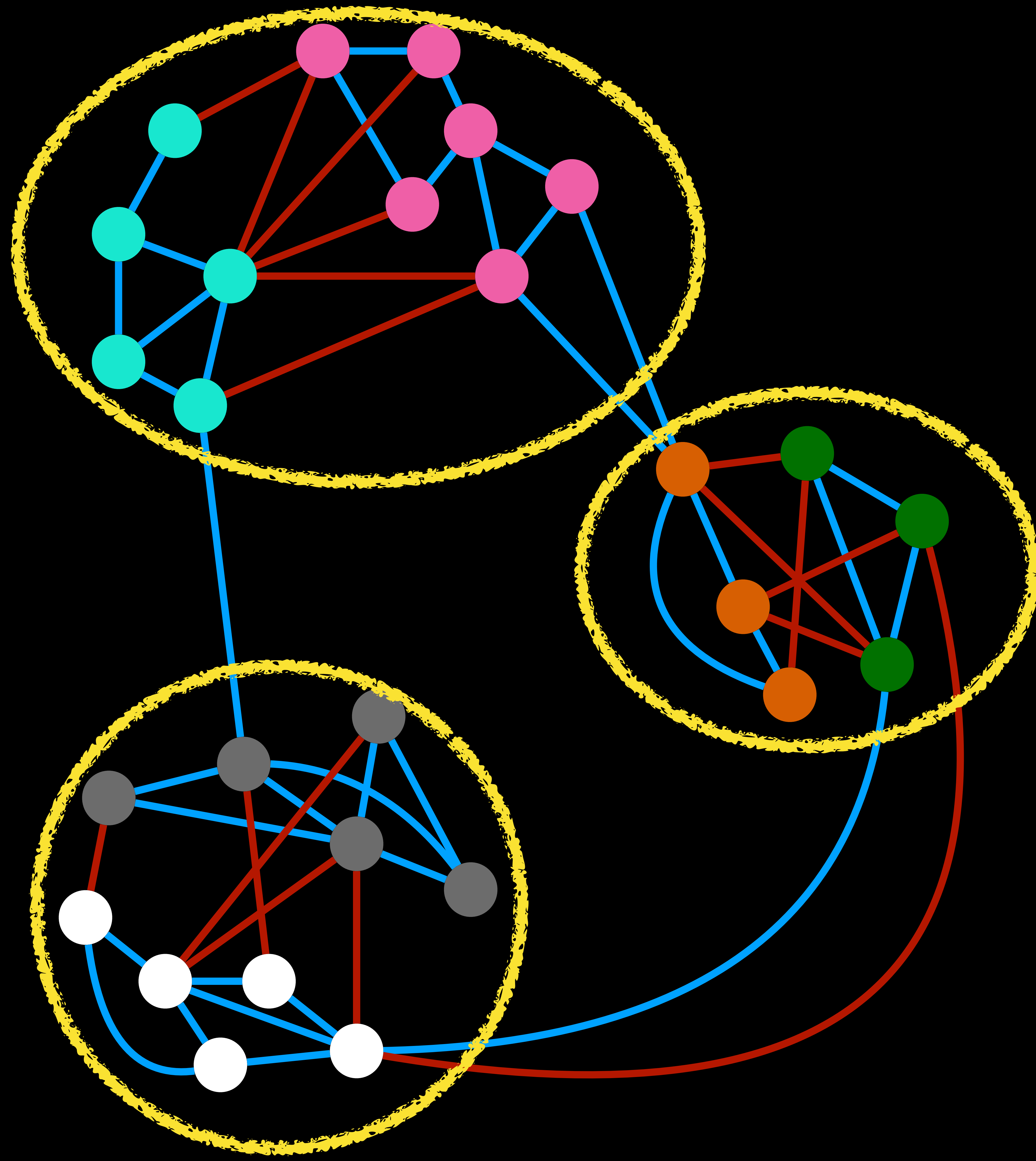


Also Highly Practical



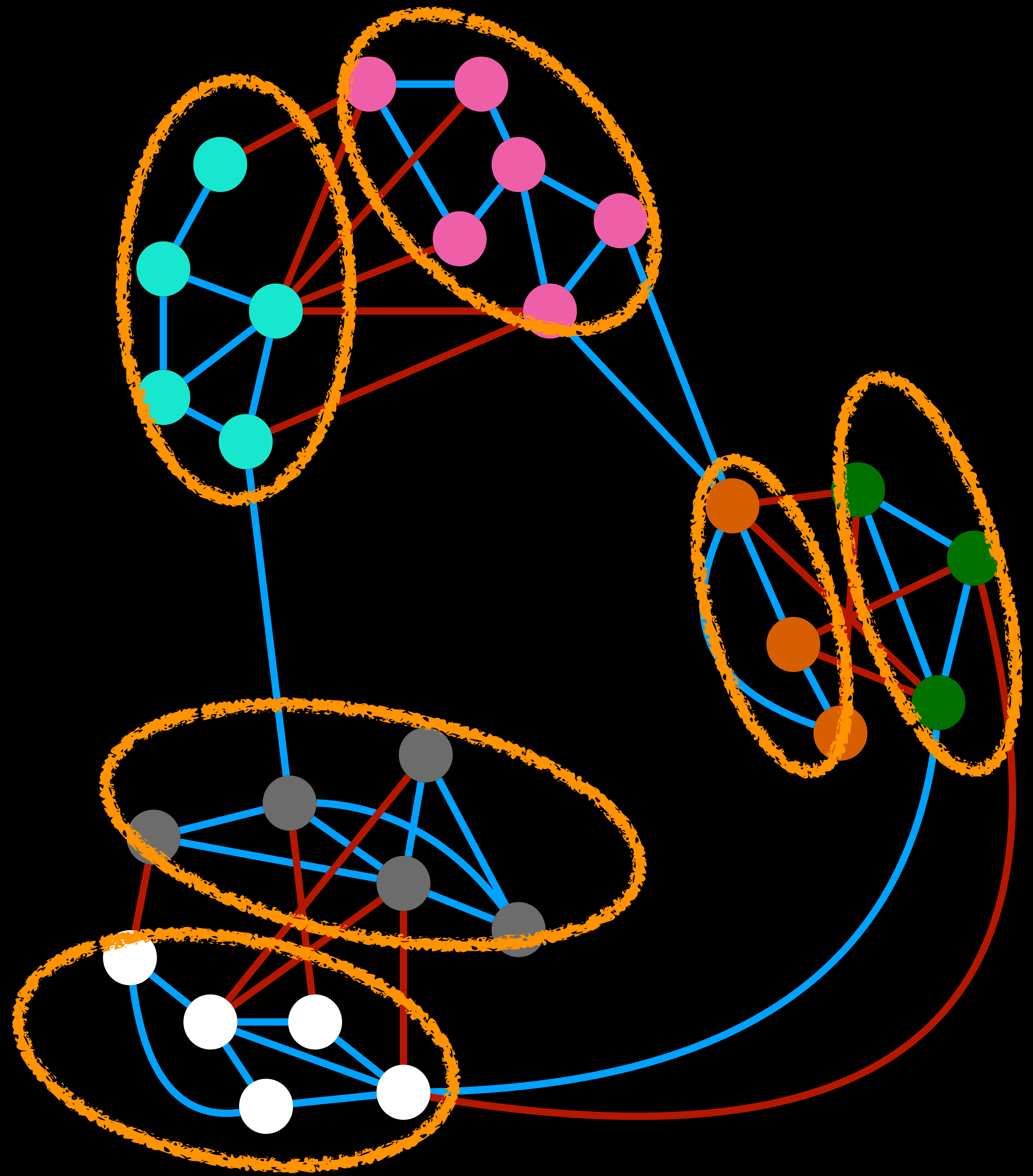
Also Highly Practical

- In practice, we can identify the conflict groups!



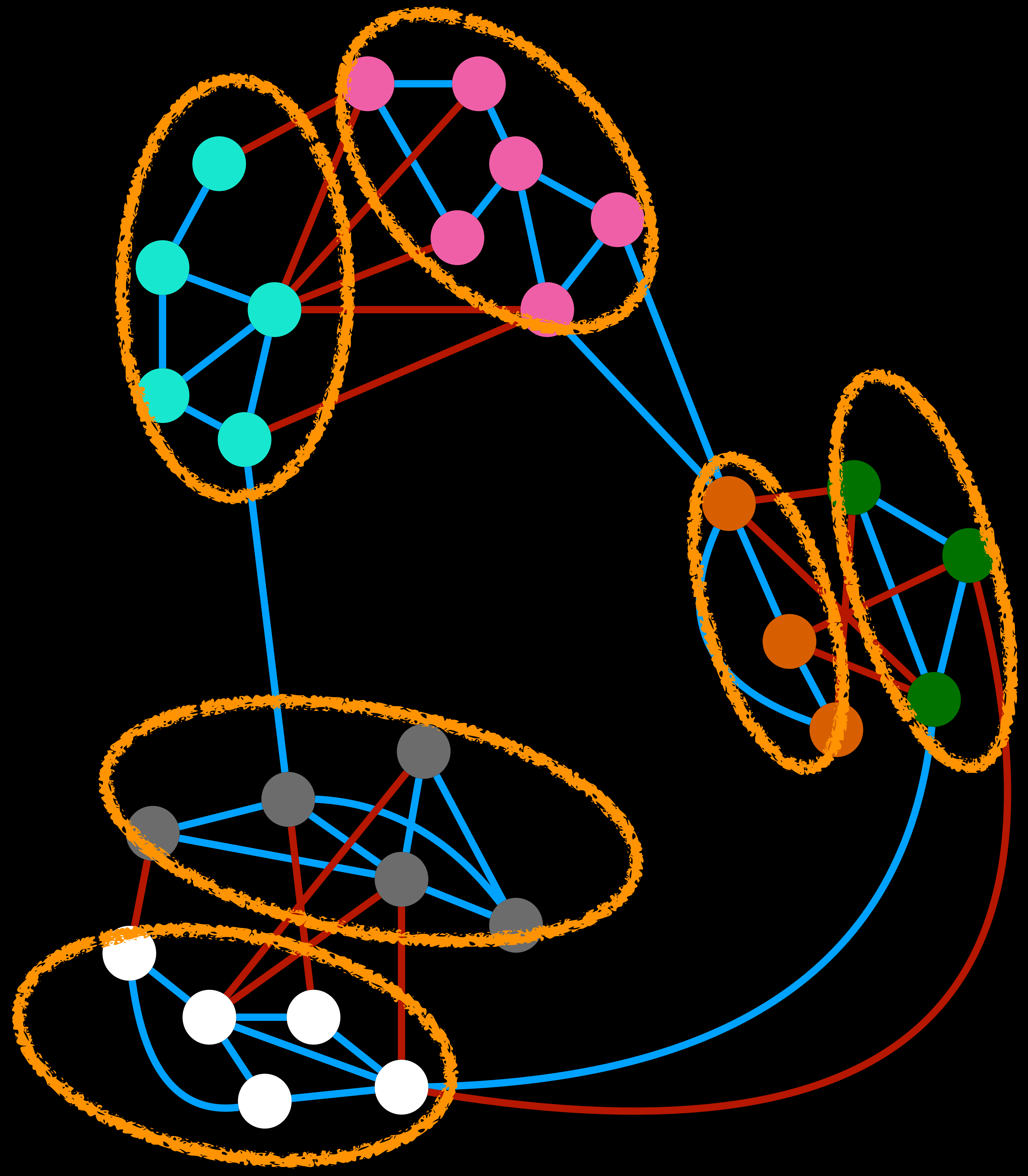
Also Highly Practical

- In practice, we can identify the conflict groups!



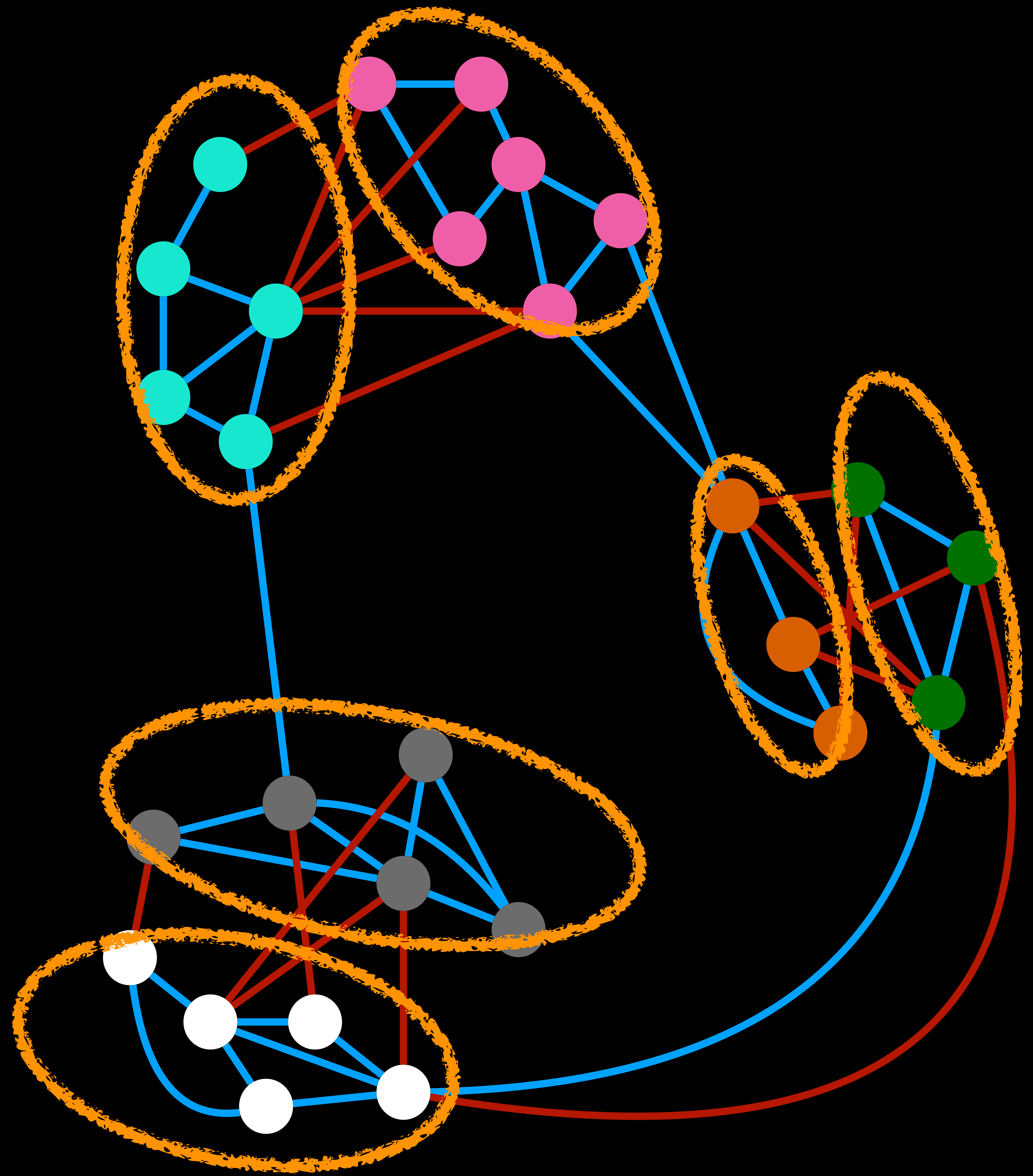
Also Highly Practical

- In practice, we can identify the conflict groups!
- For large clusters,
  our oracle has higher accuracy than baselines



Also Highly Practical

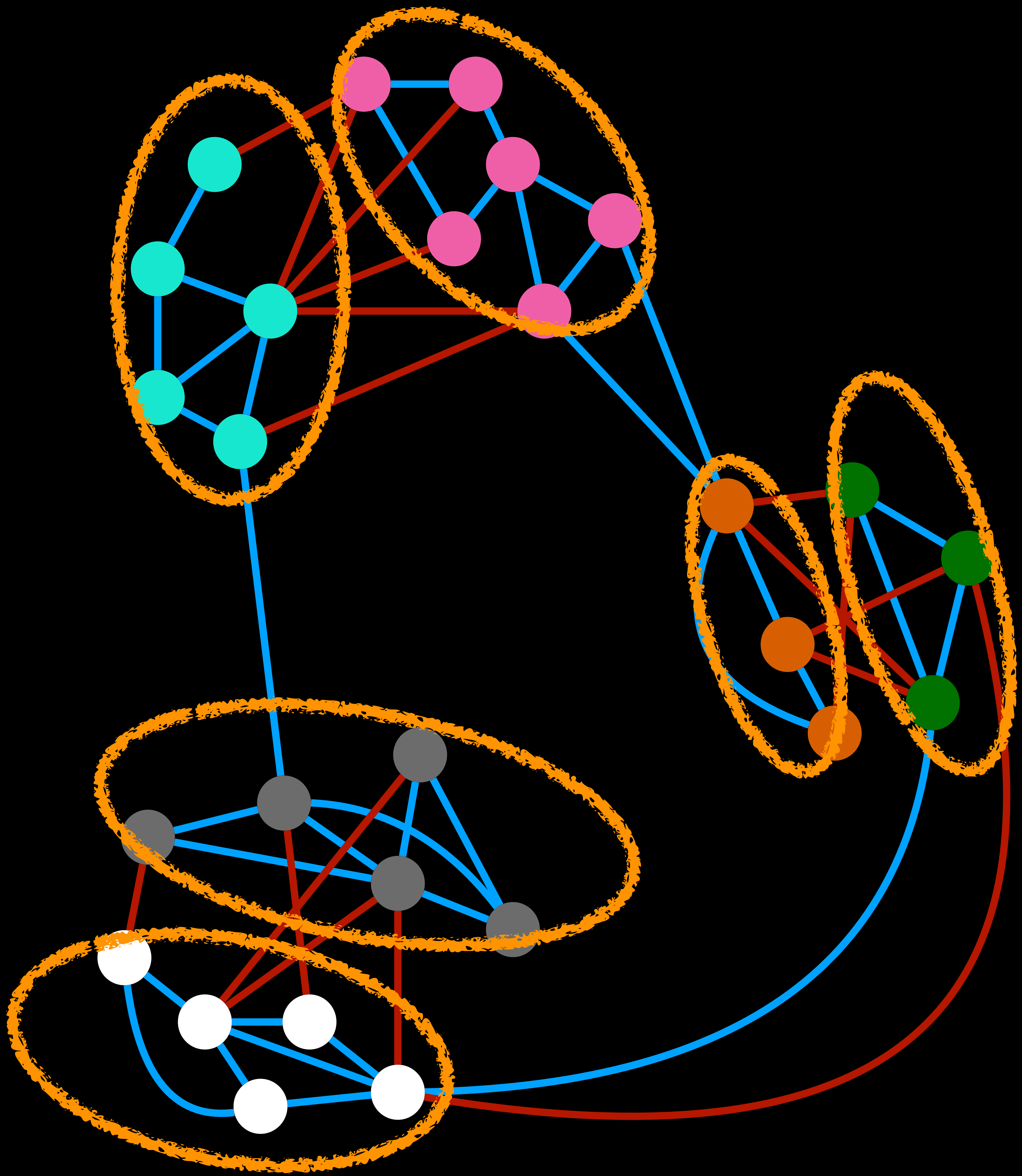
- In practice, we can identify the conflict groups!
- For large clusters,
  our oracle has higher accuracy than baselines
- Much faster than global methods if we only need to classify a small number of vertices!



Also Highly Practical

- In practice, we can identify the conflict groups!
- For large clusters,
  our oracle has higher accuracy than baselines
- Much faster than global methods if we only need to classify a small number of vertices!

  ➡ Each query takes ≤1.5 seconds on a graph with 250k vertices and 3M edges

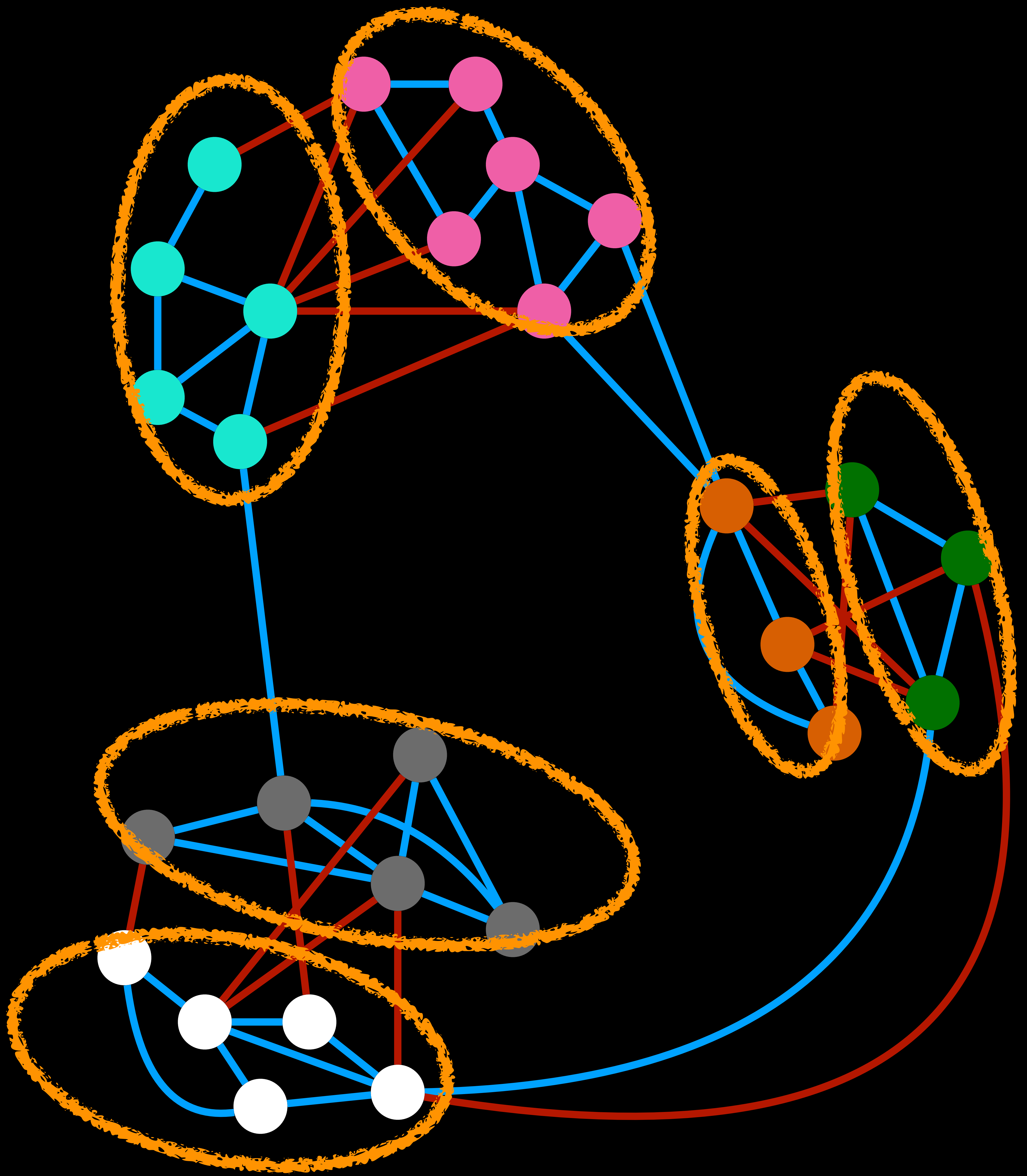


Also Highly Practical

- In practice, we can identify the conflict groups!
- For large clusters,
  our oracle has higher accuracy than baselines
- Much faster than global methods if we only need to classify a small number of vertices!

  ➡ Each query takes ≤1.5 seconds on a graph with 250k vertices and 3M edges

  ➡ Easy to parallelize

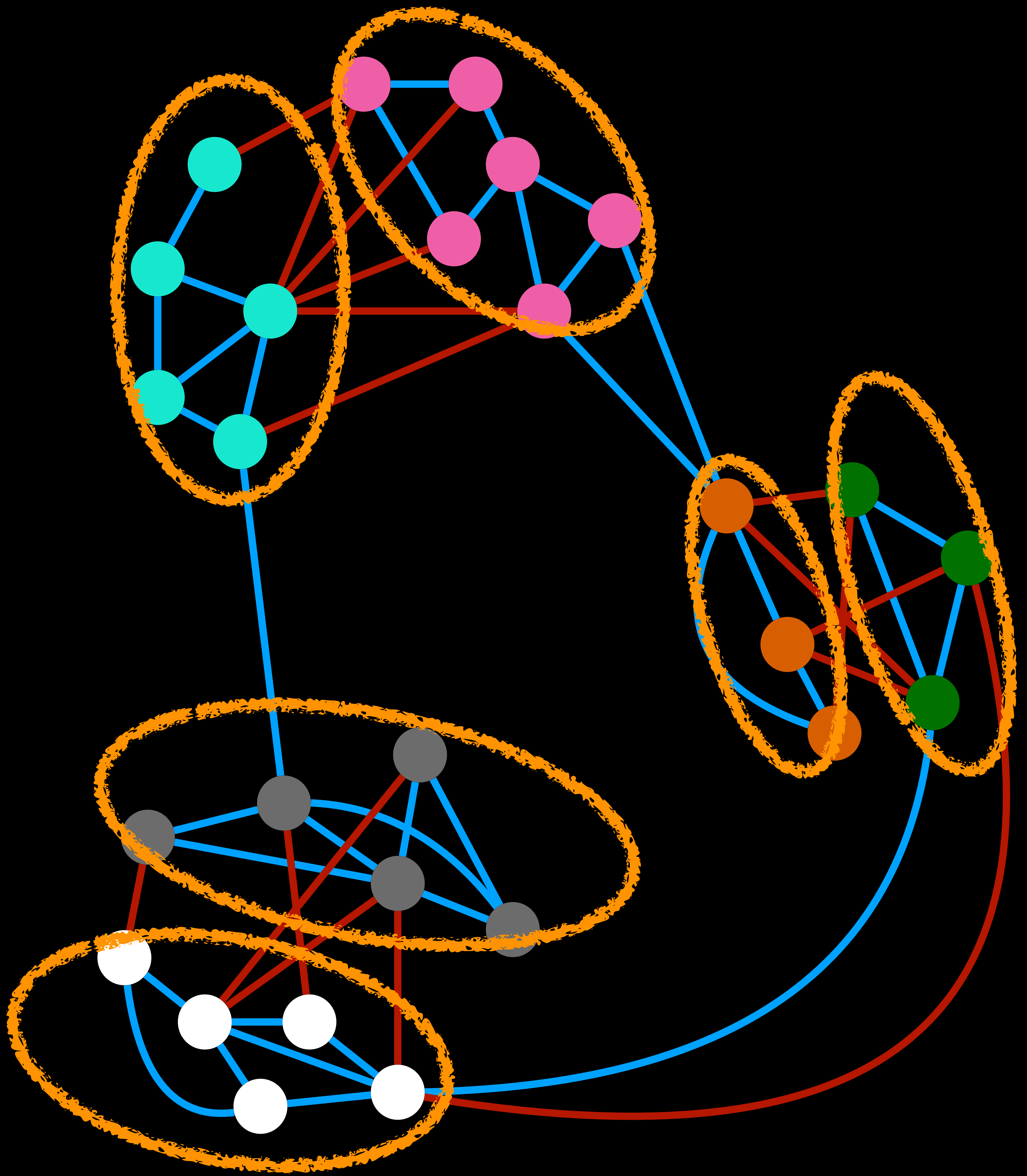


Also Highly Practical

- In practice, we can identify the conflict groups!
- For large clusters,
  our oracle has higher accuracy than baselines
- Much faster than global methods if we only need to classify a small number of vertices!

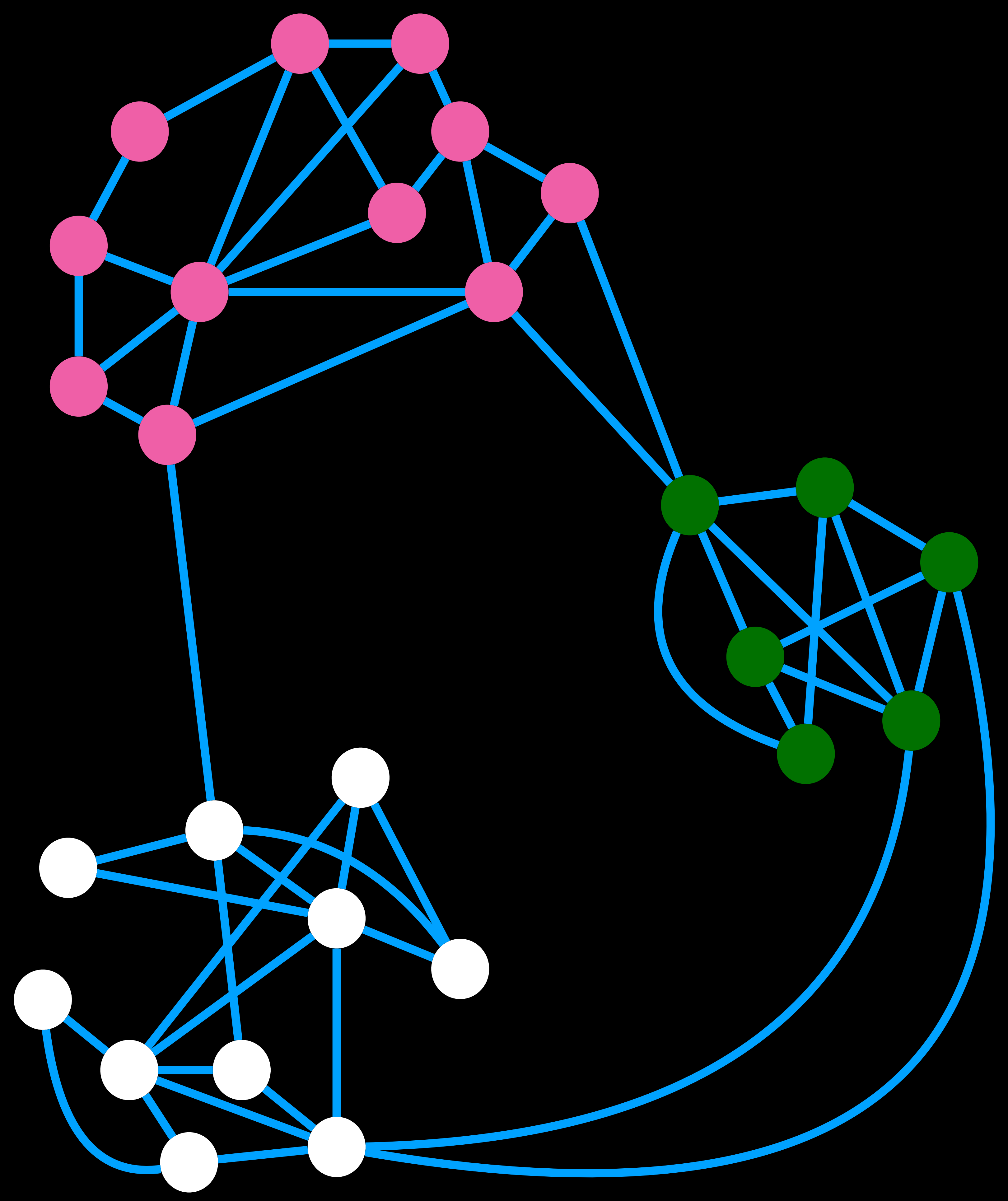
  ➡ Each query takes ≤1.5 seconds on a graph with 250k vertices and 3M edges

  ➡ Easy to parallelize
- We provide a new signed graph dataset with large ground-truth communities

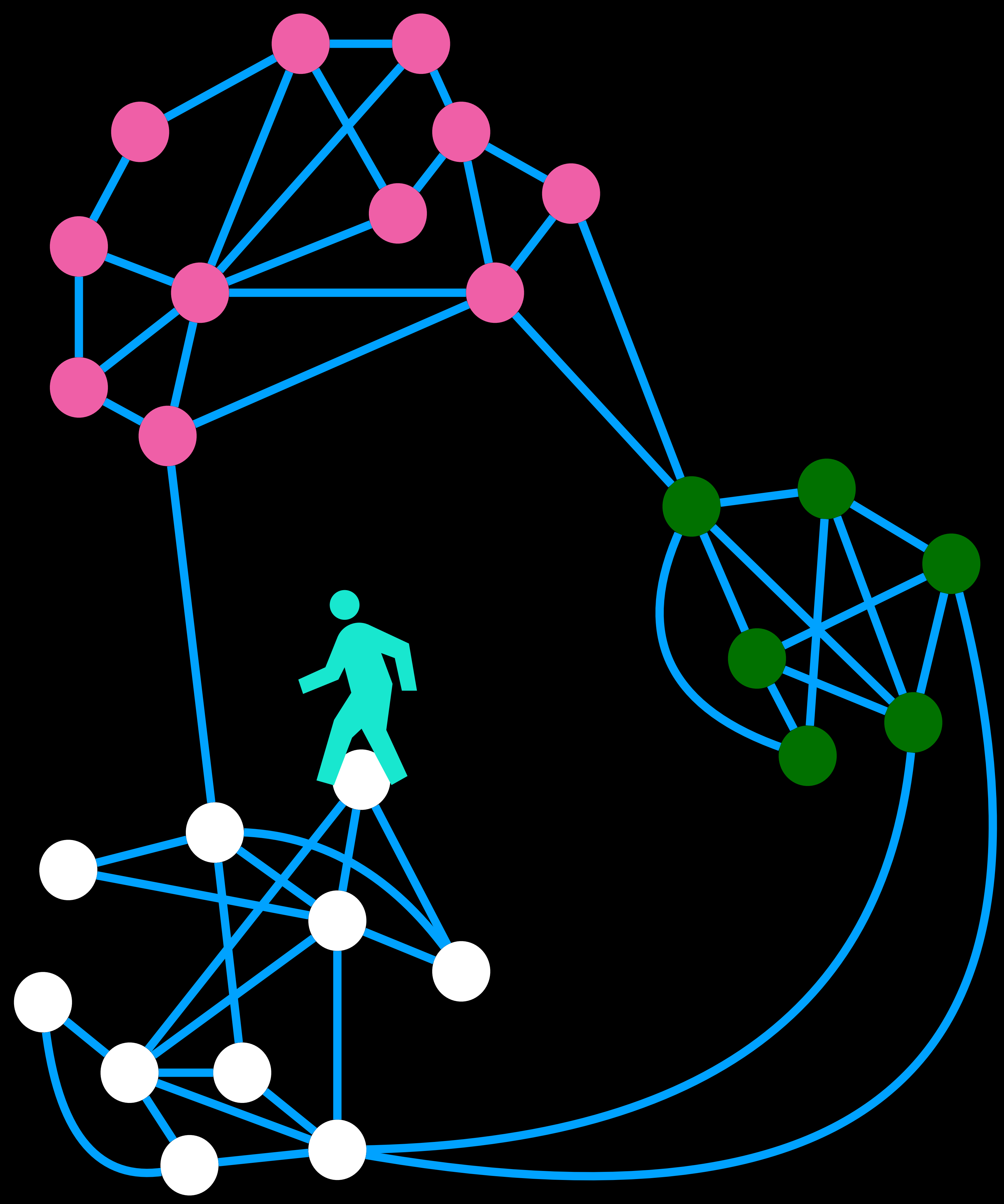


How Does the Oracle Work?

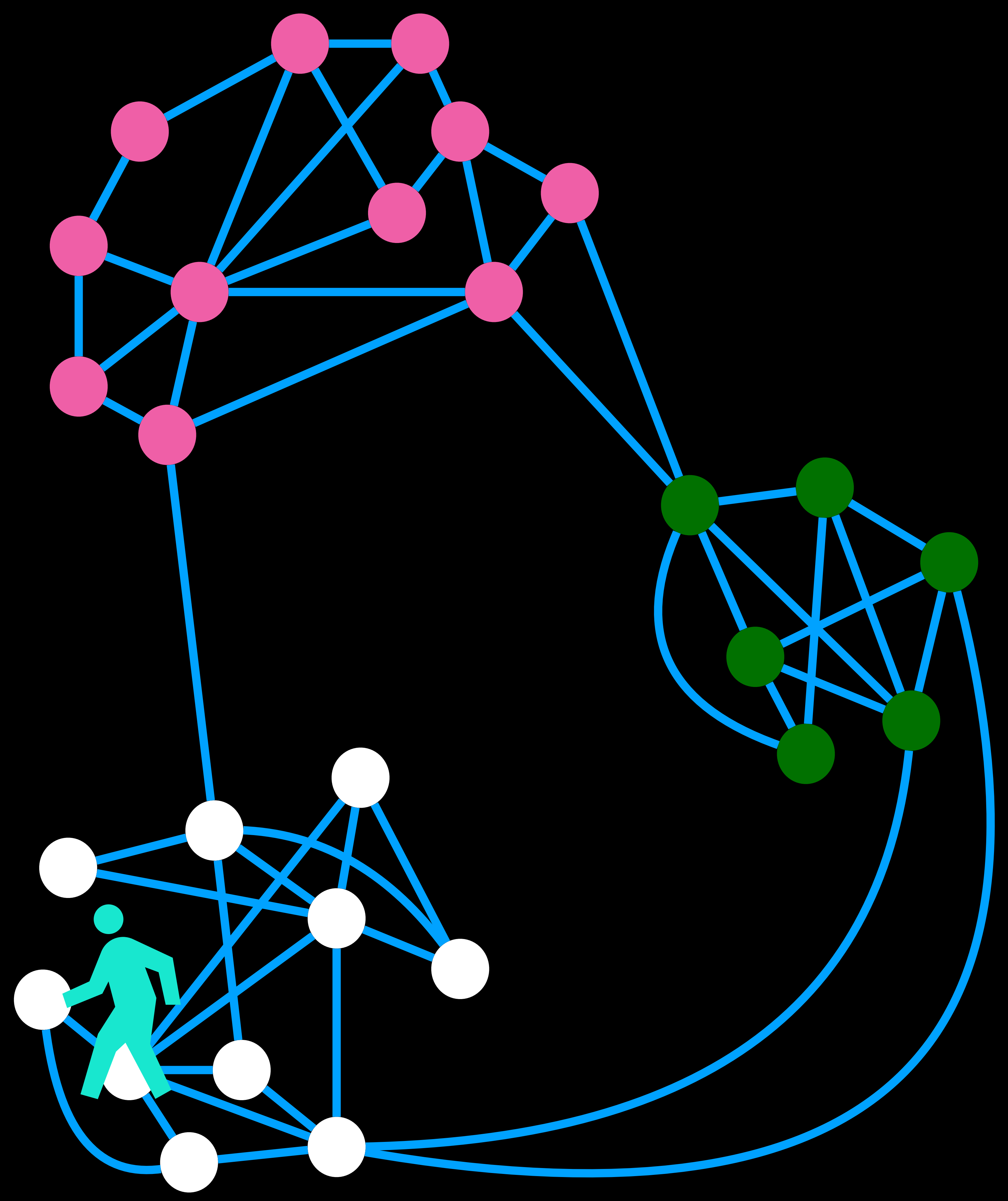
Unsigned Graphs



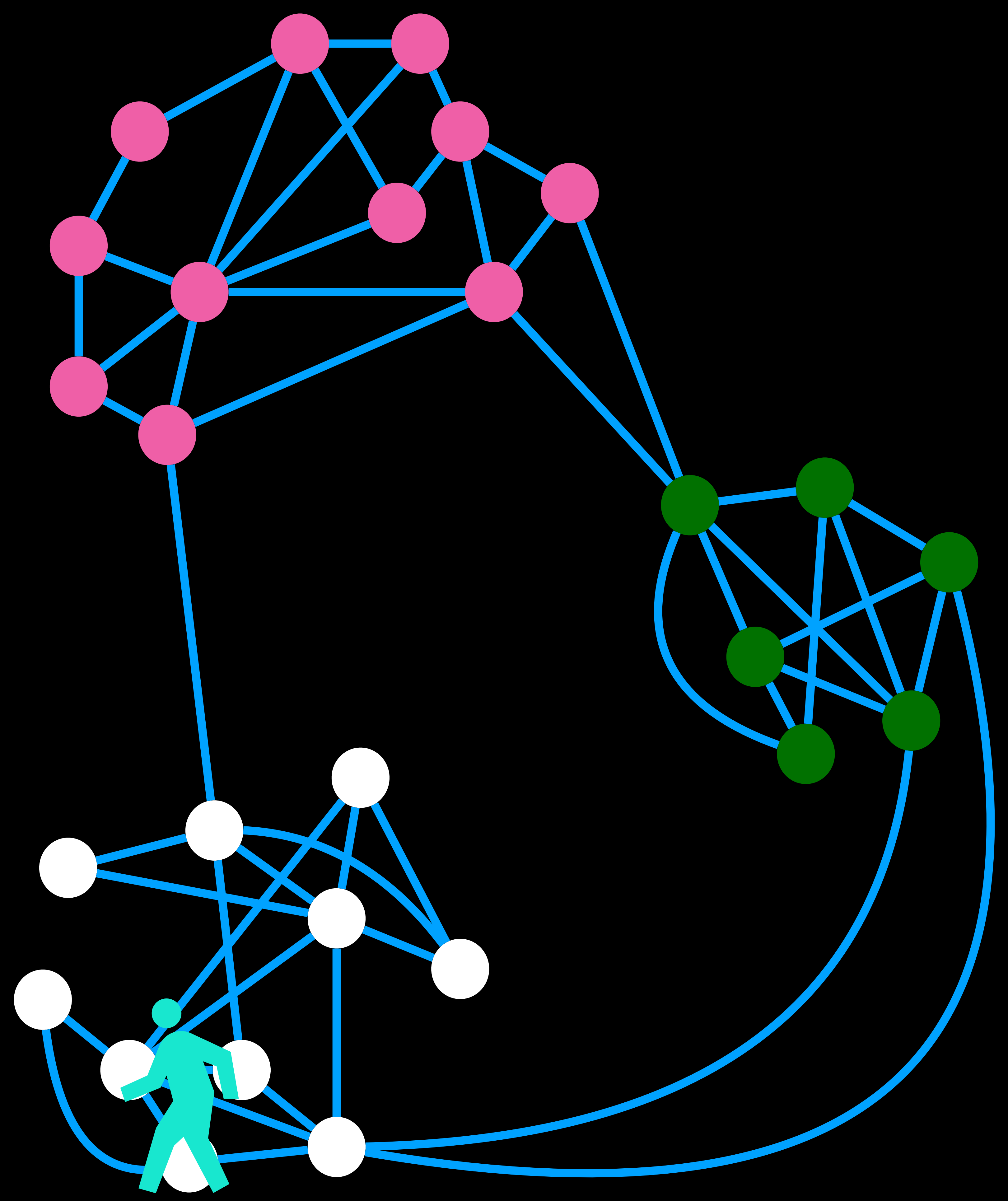
Unsigned Graphs



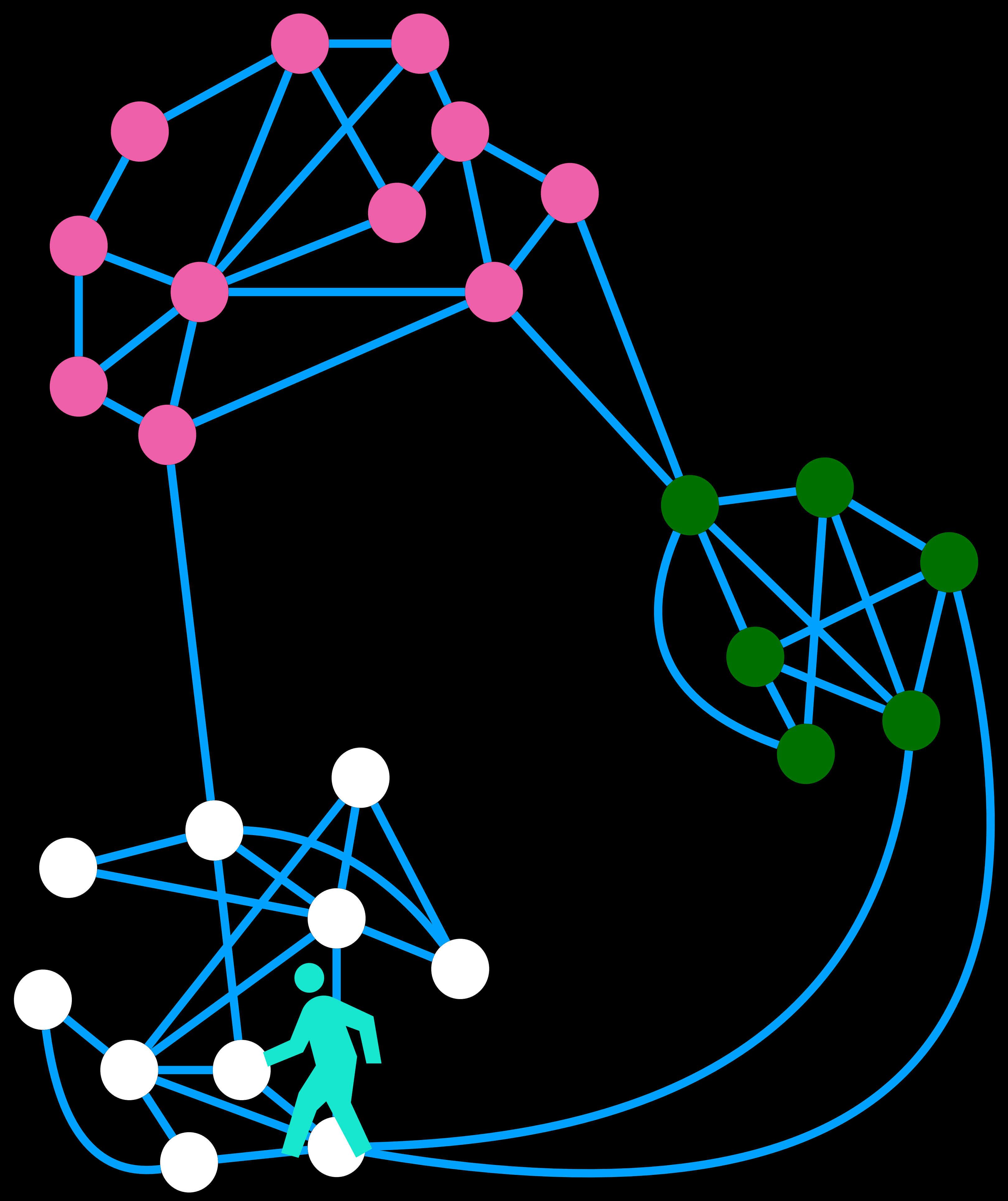
Unsigned Graphs



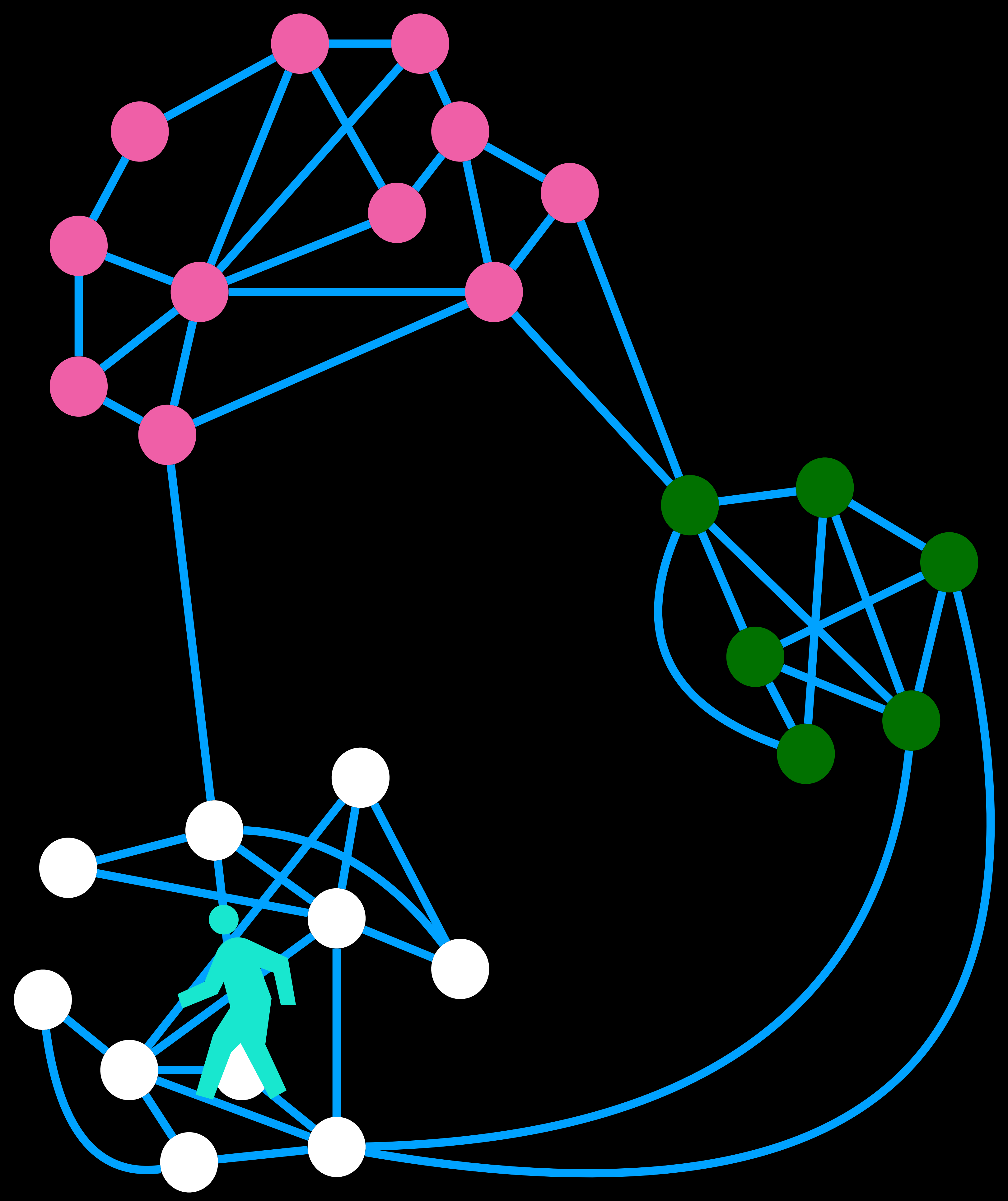
Unsigned Graphs



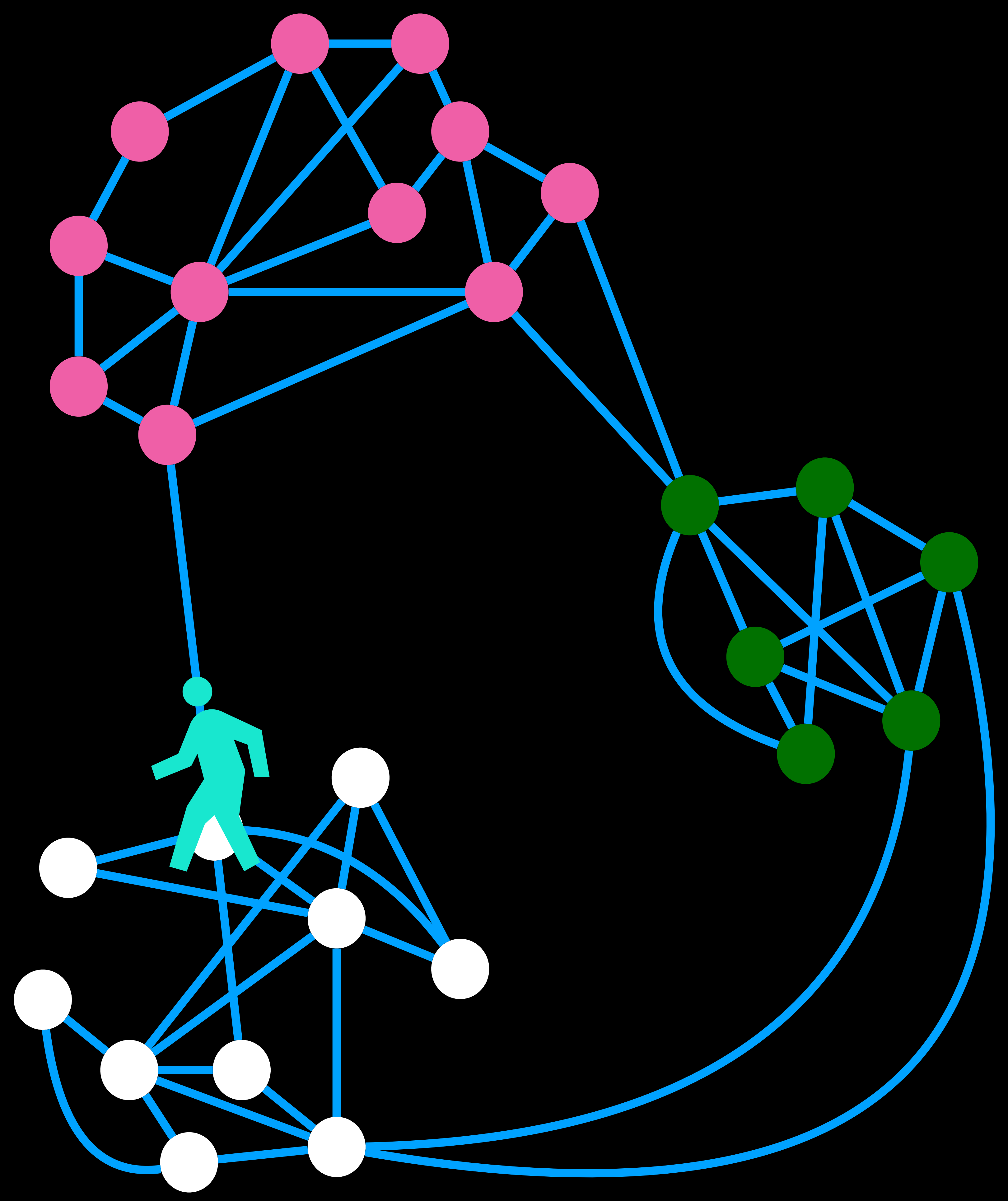
Unsigned Graphs



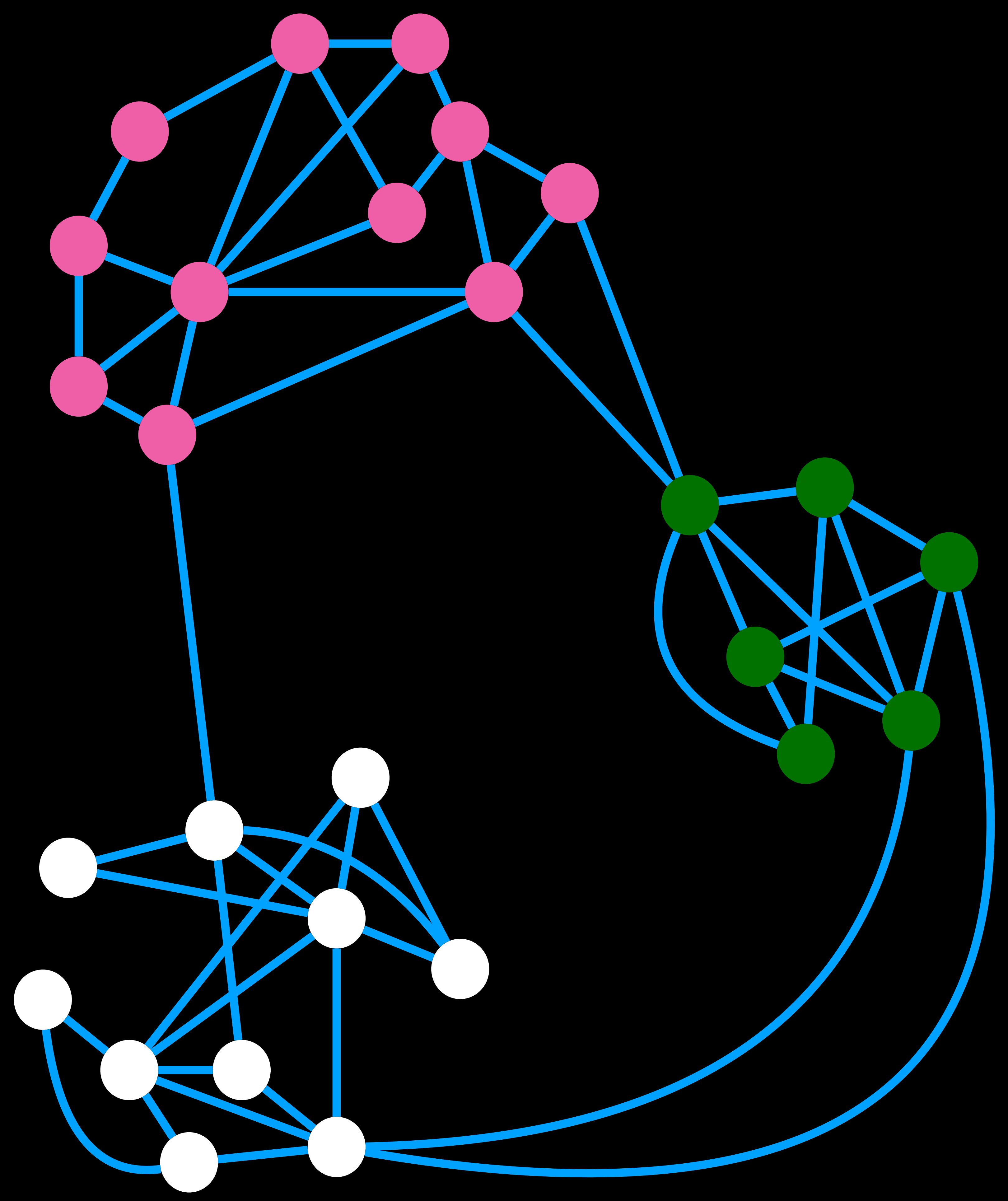
Unsigned Graphs



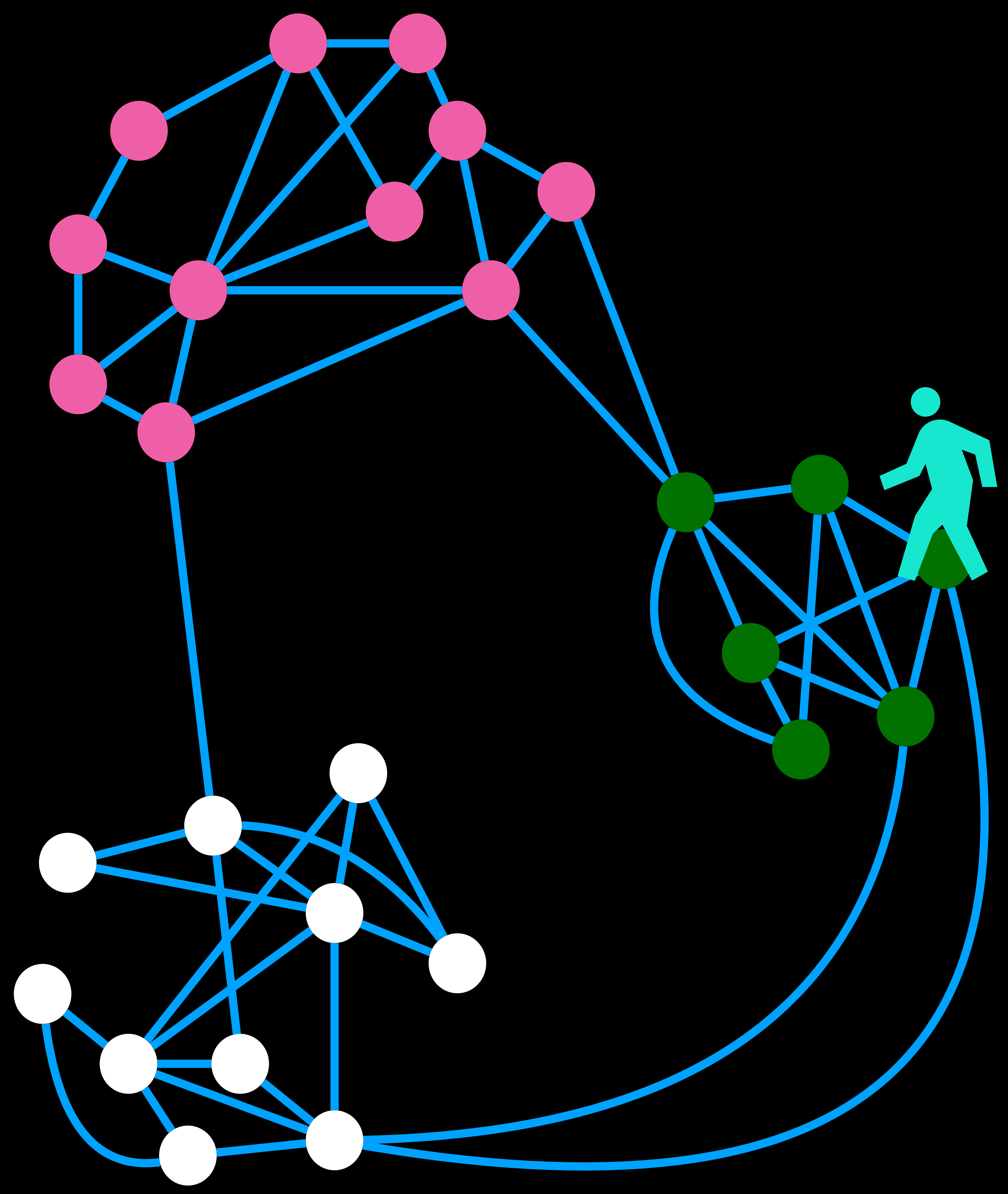
Unsigned Graphs



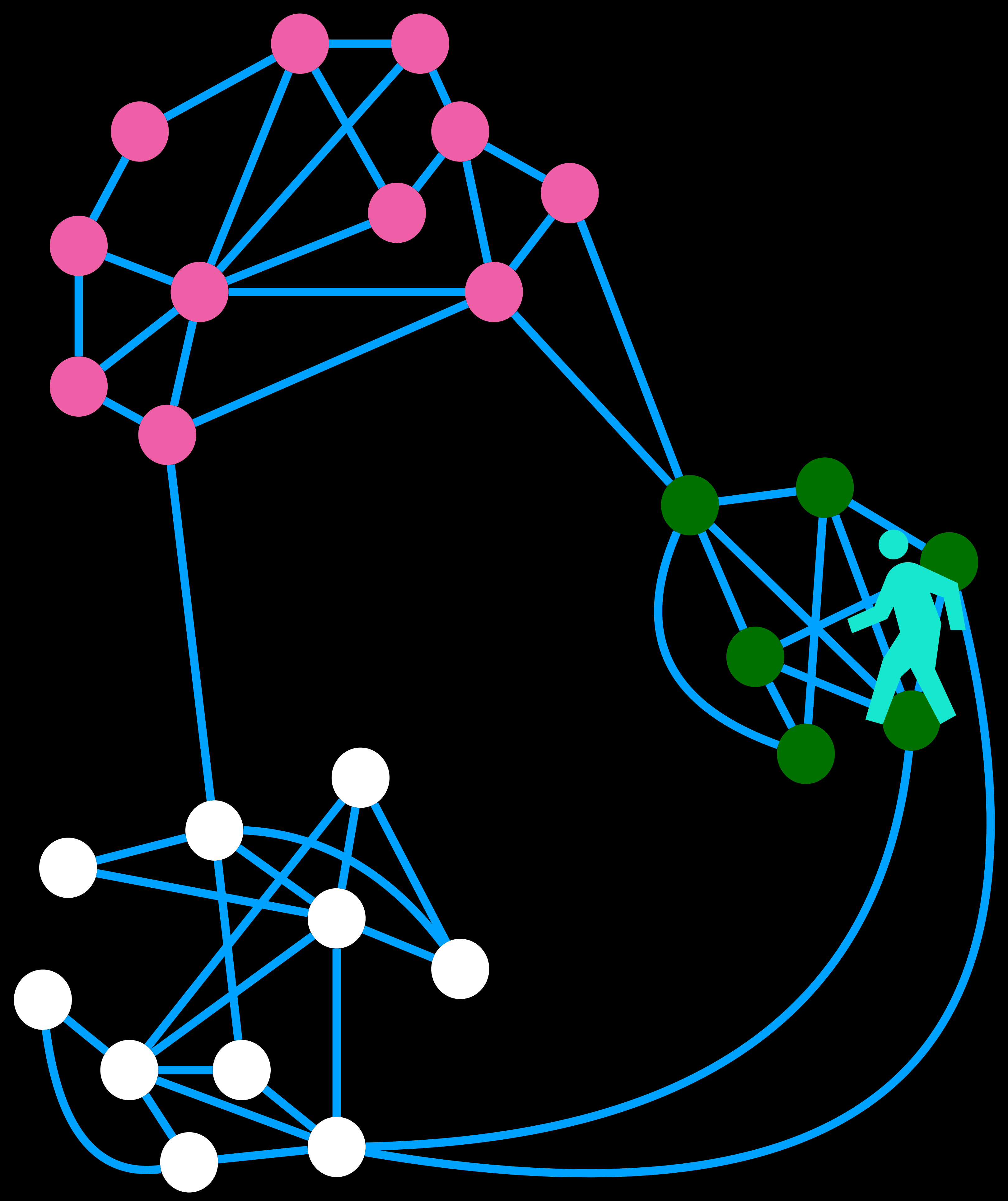
Unsigned Graphs



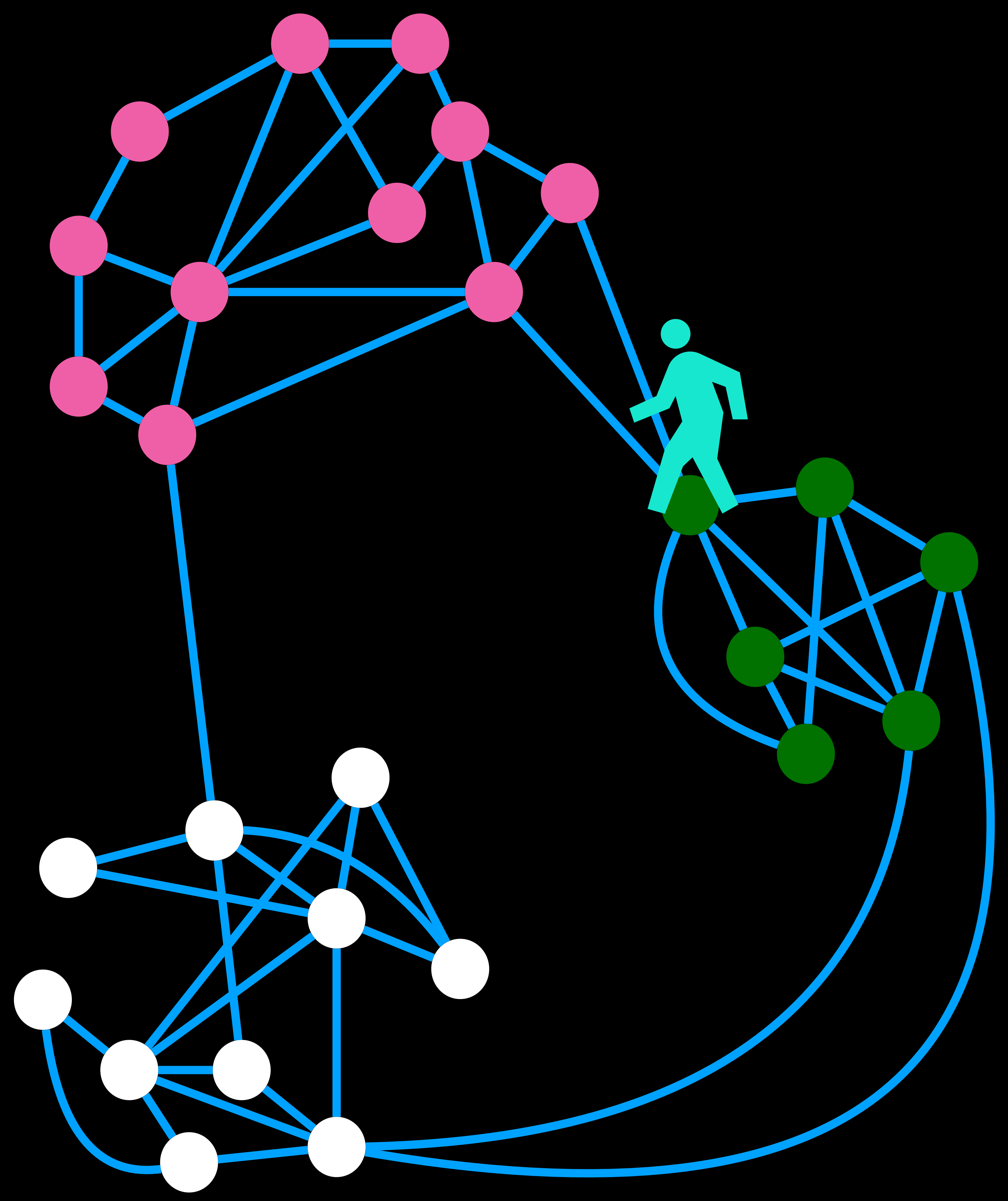
Unsigned Graphs



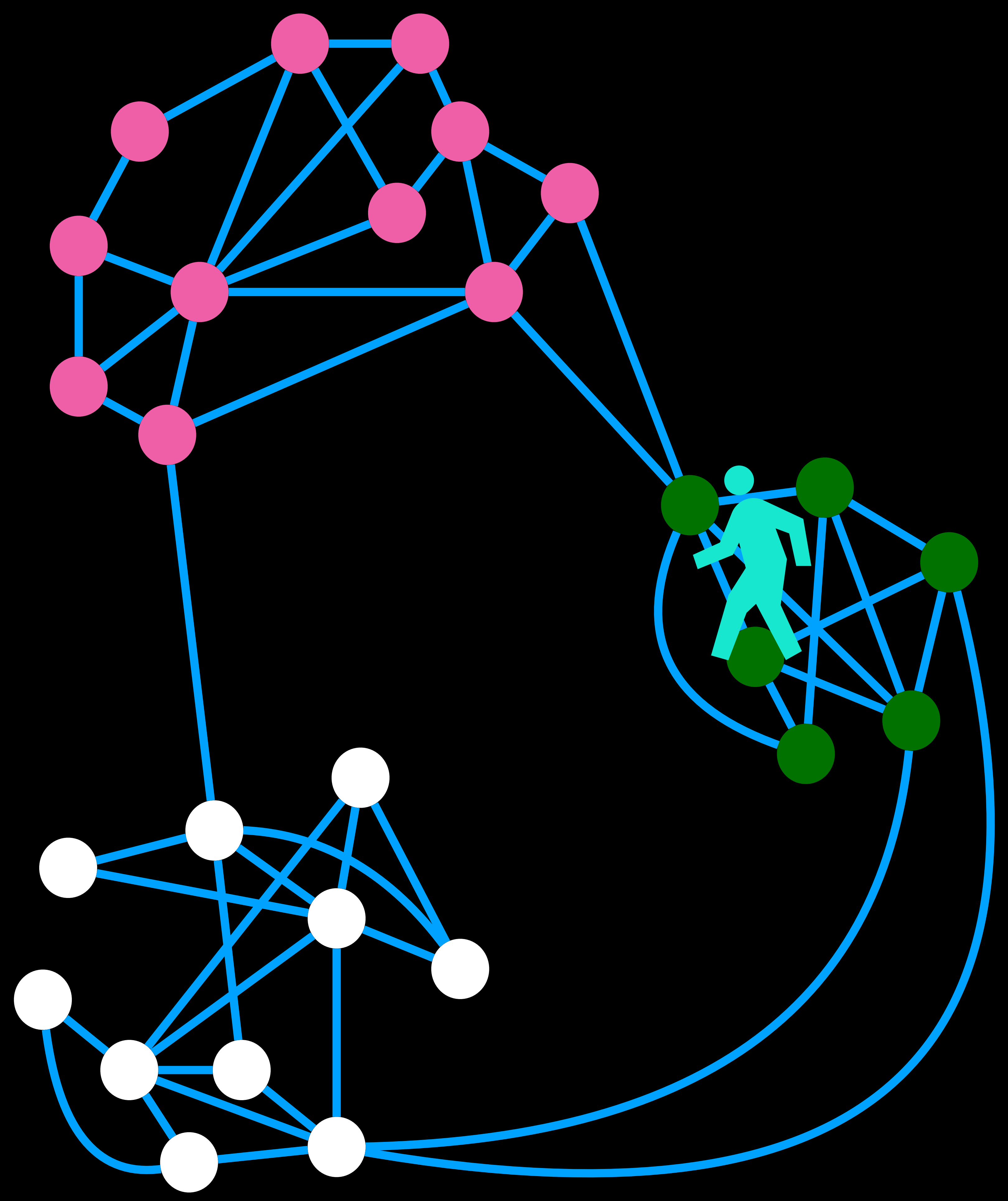
Unsigned Graphs



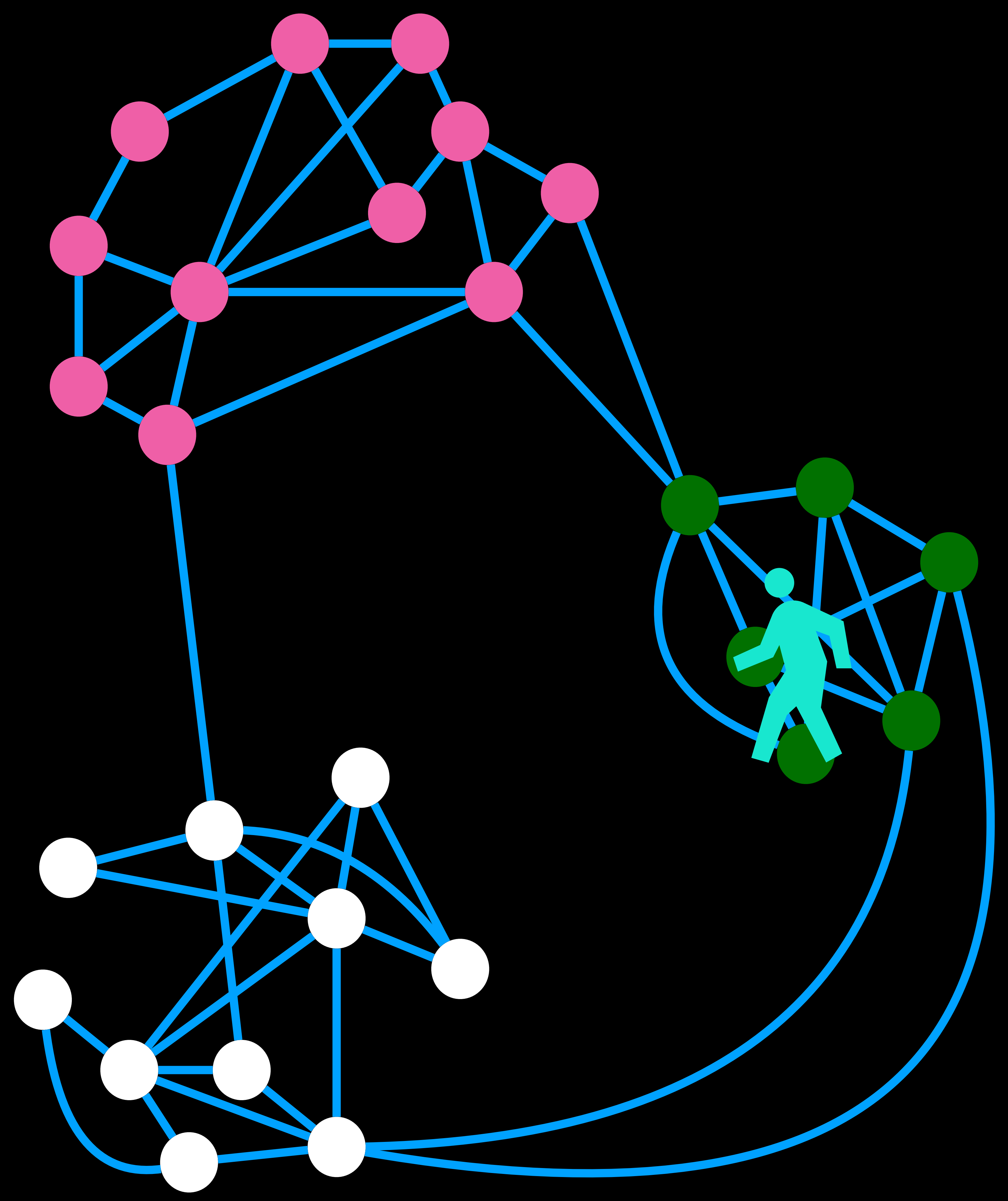
Unsigned Graphs



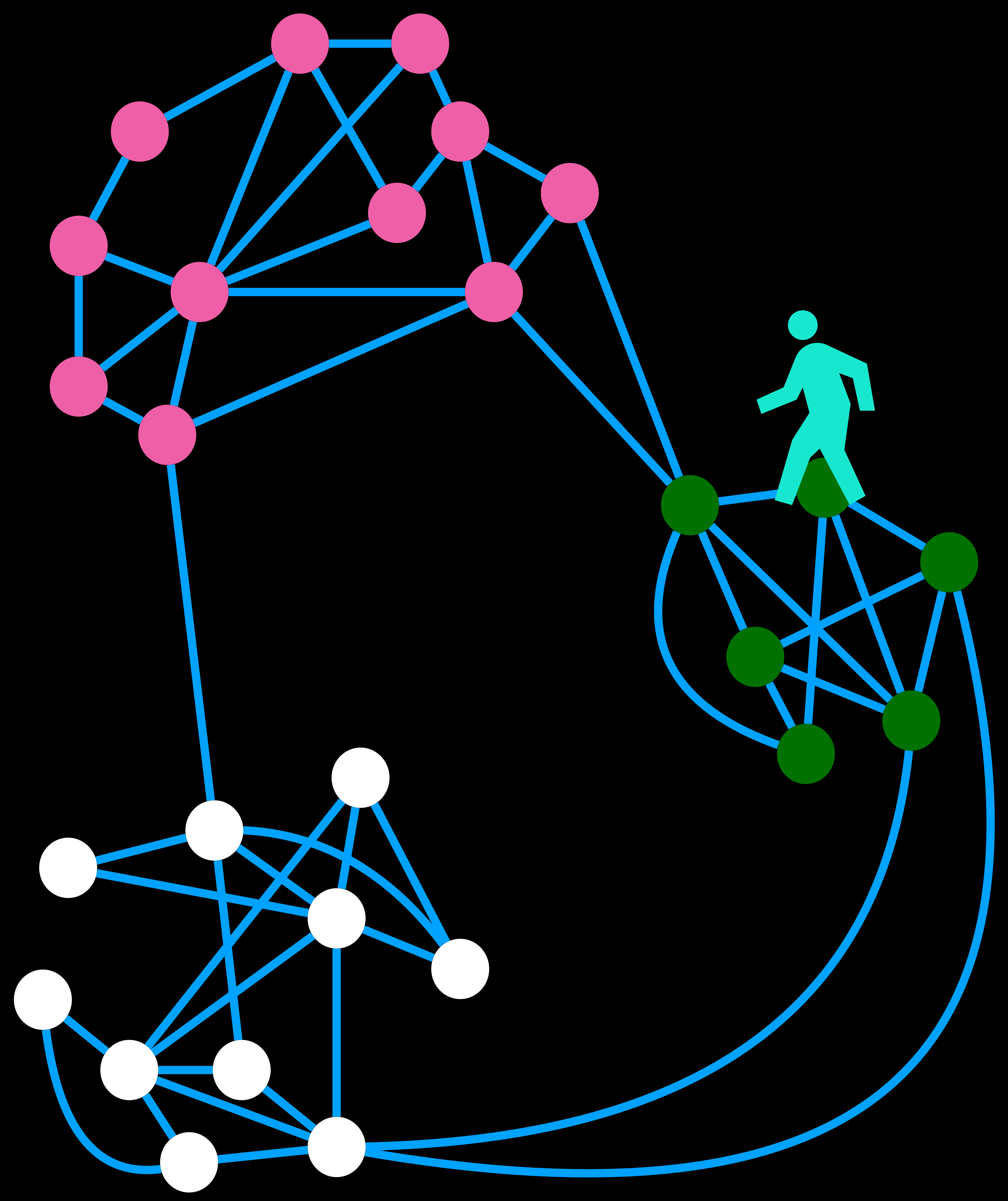
Unsigned Graphs



Unsigned Graphs

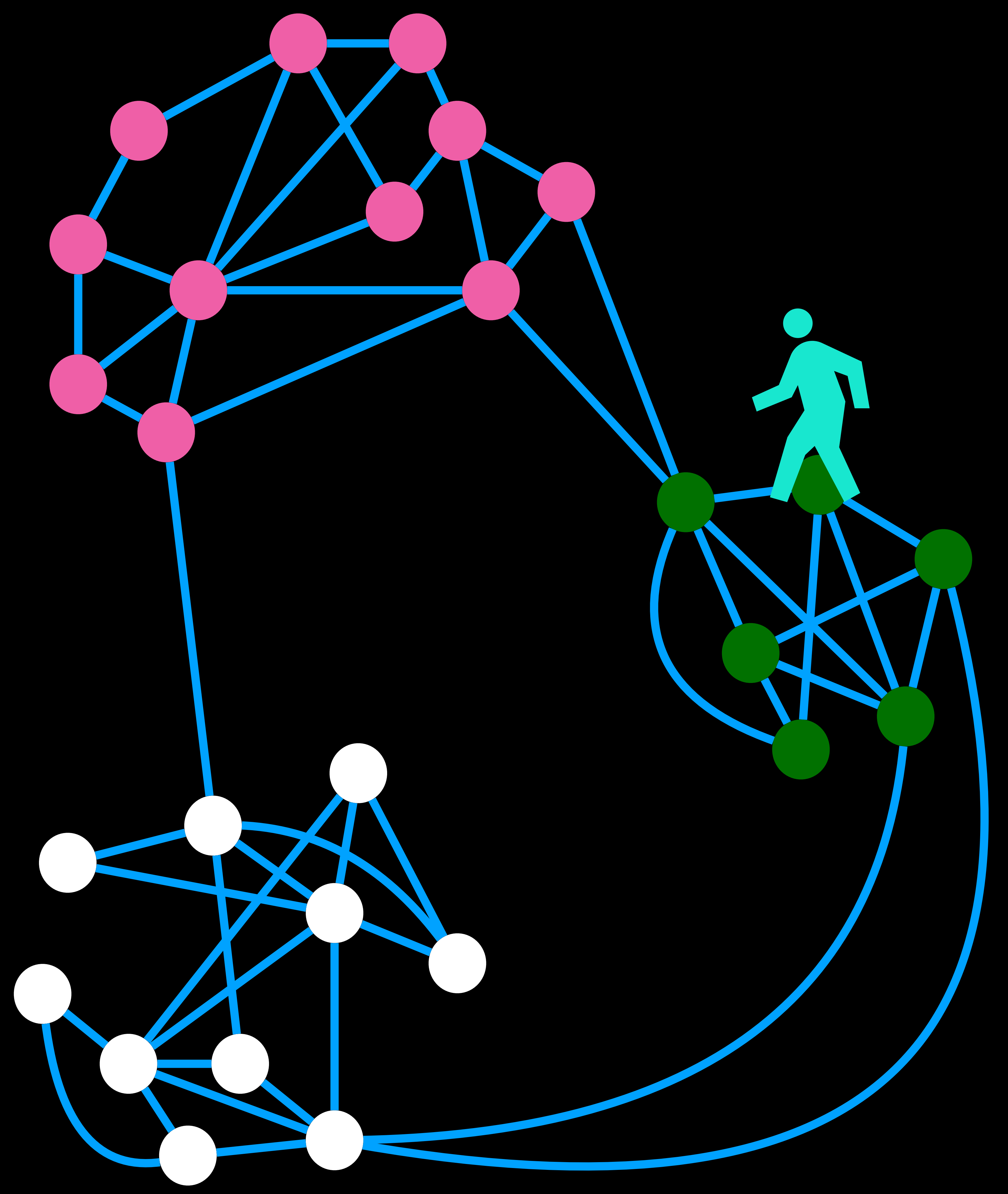


Unsigned Graphs



Unsigned Graphs

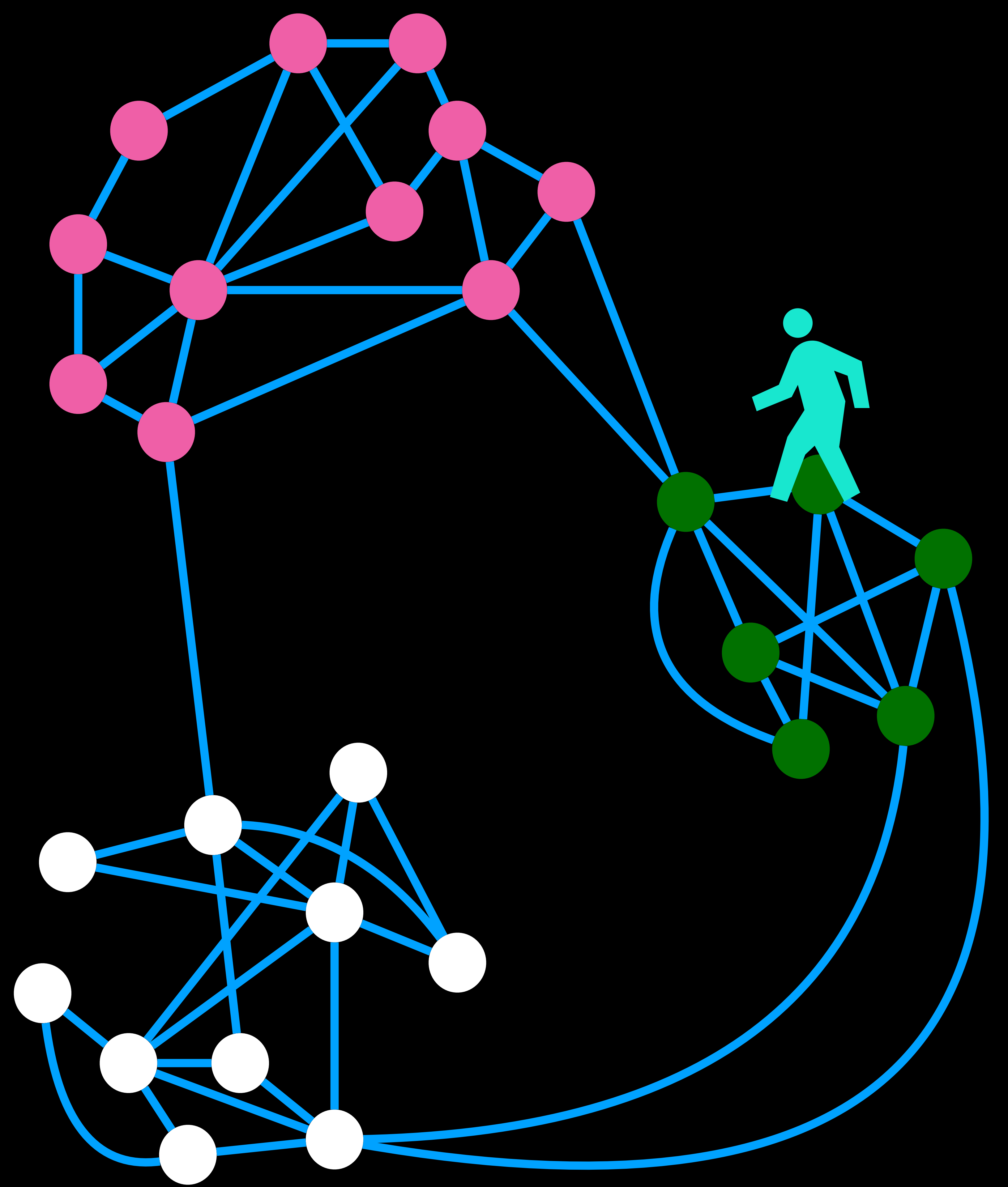

Computing communities in large networks using random walks
P Pons, M Latapy - International symposium on computer and information …, 2005 - Springer
Dense subgraphs of sparse graphs (communities), which appear in most real-world complex networks, play an important role in many contexts. Computing them however is generally …
☆ Save 99 Cite Cited by 2065 Related articles All 43 versions



Unsigned Graphs

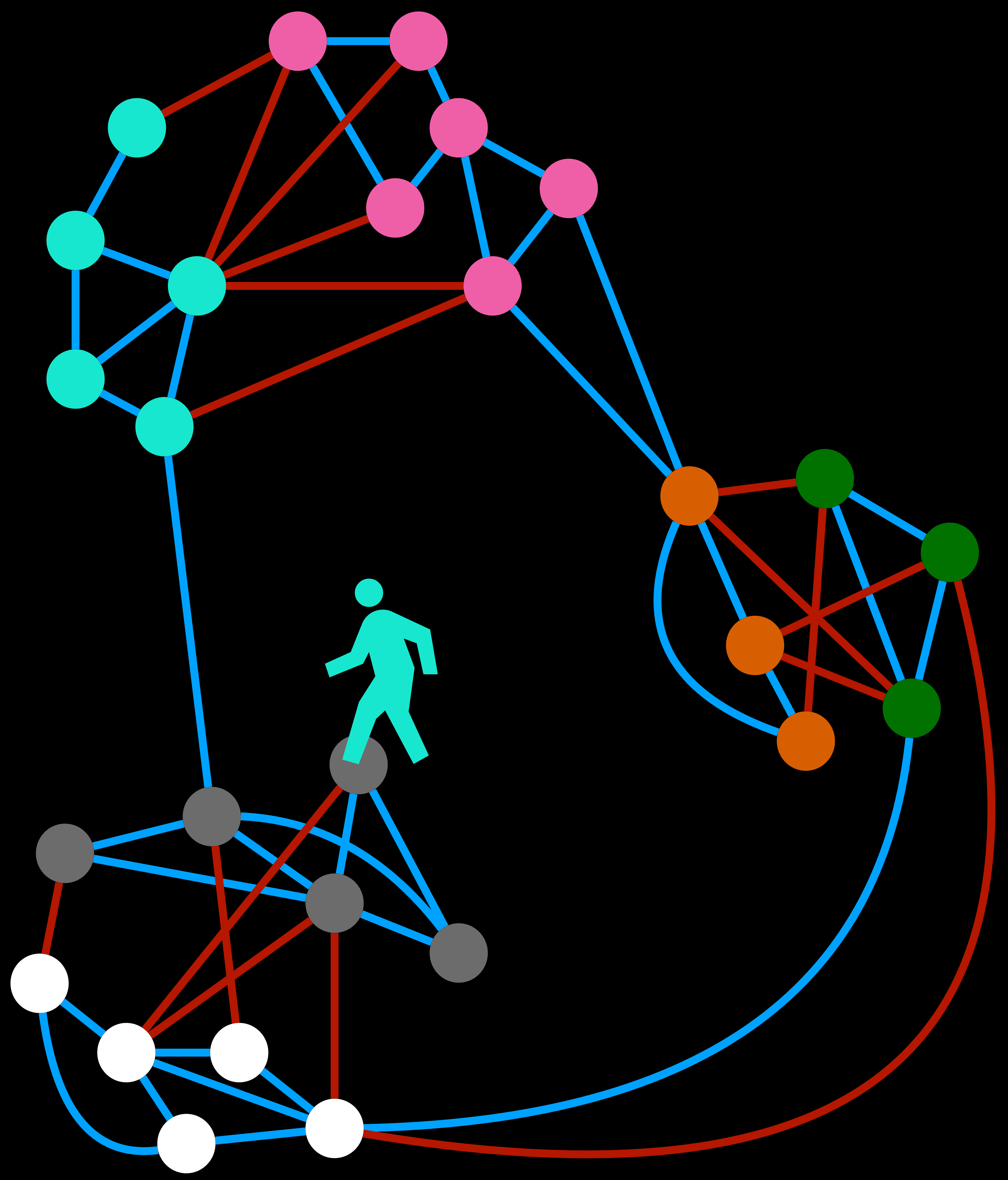
Computing communities in large networks using random walks
P Pons, M **Latapy** - International symposium on computer and information …, 2005 - Springer
Dense subgraphs of sparse graphs (communities), which appear in most real-world complex networks, play an important role in many contexts. Computing them however is generally …
☆ Save 99 Cite Cited by 2065 Related articles All 43 versions

Testing cluster structure of graphs
A Czumaj, P Peng, C Sohler - Proceedings of the forty-seventh annual …, 2015 - dl.acm.org
We study the problem of recognizing the cluster structure of a graph in the framework of property testing in the bounded degree model. Given a parameter ε, a d-bounded degree graph is defined to be (k, φ)-clusterable, if it can be partitioned into no more than k parts, such that the (inner) conductance of the induced subgraph on each part is at least φ and the (outer) conductance of each part is at most cd, k ε 4 φ 2, where cd, k depends only on d, k. Our main result is a sublinear algorithm with the running time~ O (√ n· poly (φ, k, 1/ε)) that …
☆ Save 99 Cite Cited by 35 Related articles All 7 versions

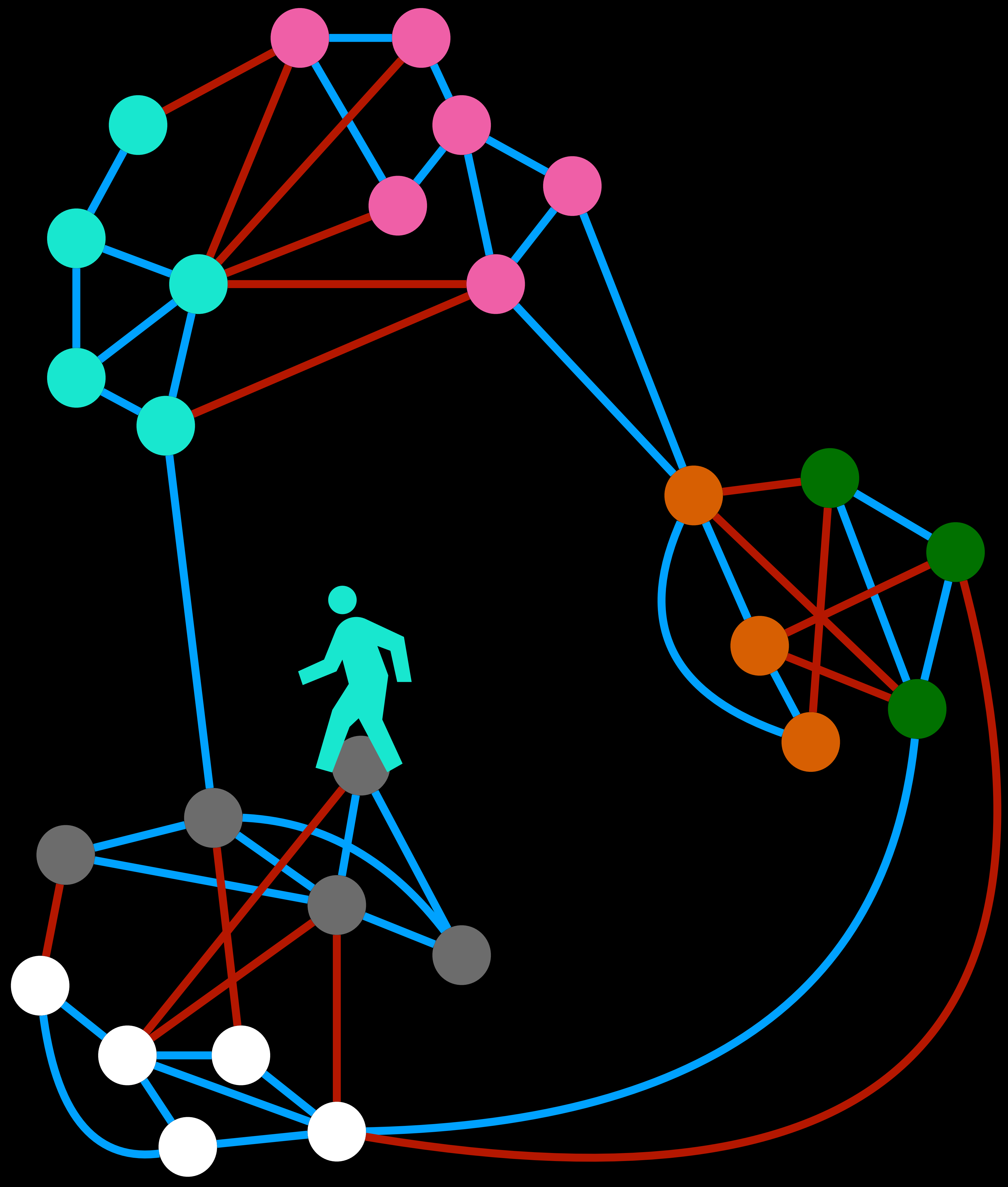


Signed Graphs

What changes when we have signs?

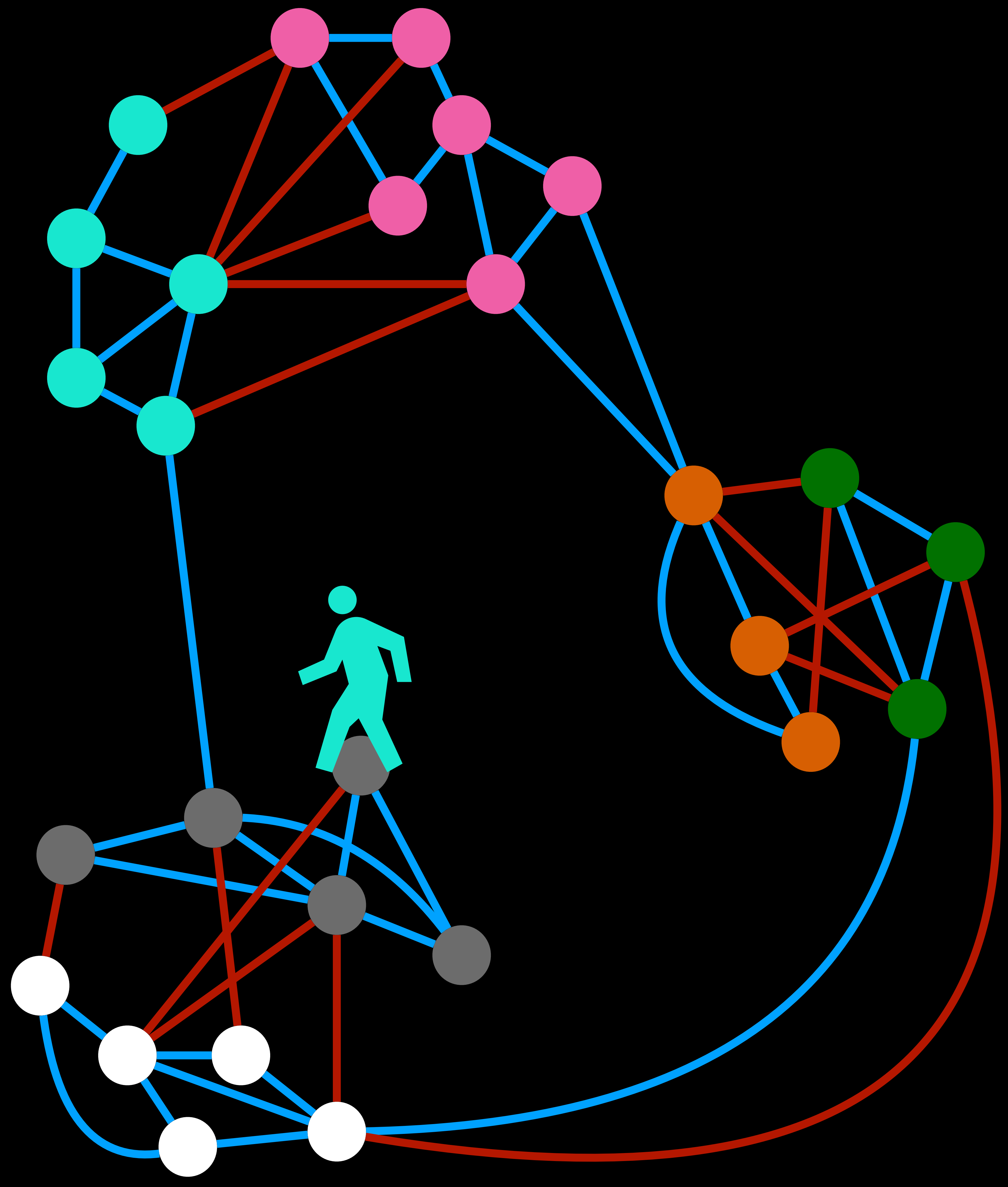


Random Walks With Signs



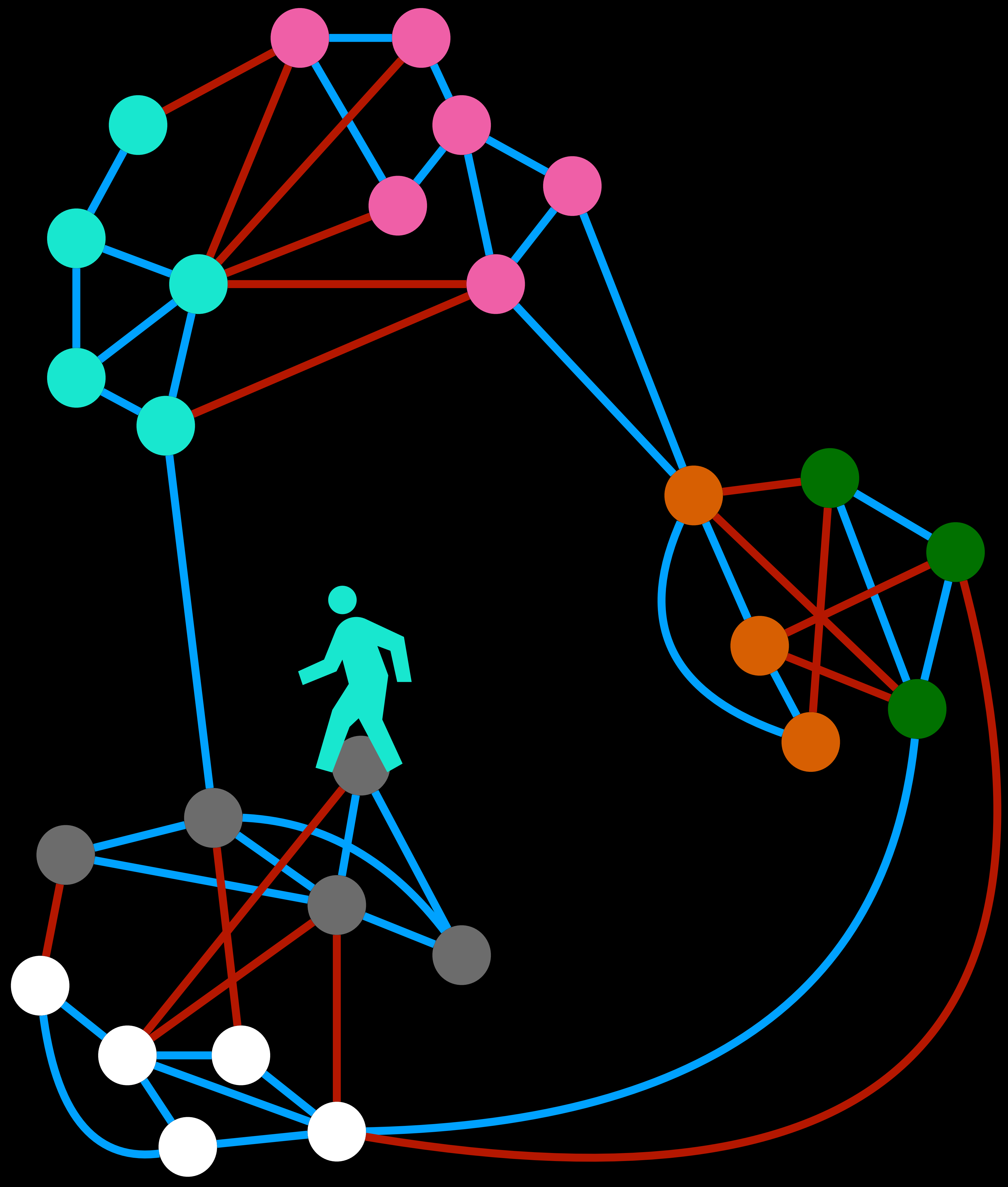
Random Walks With Signs

- Initialize sign $s = +$



Random Walks With Signs

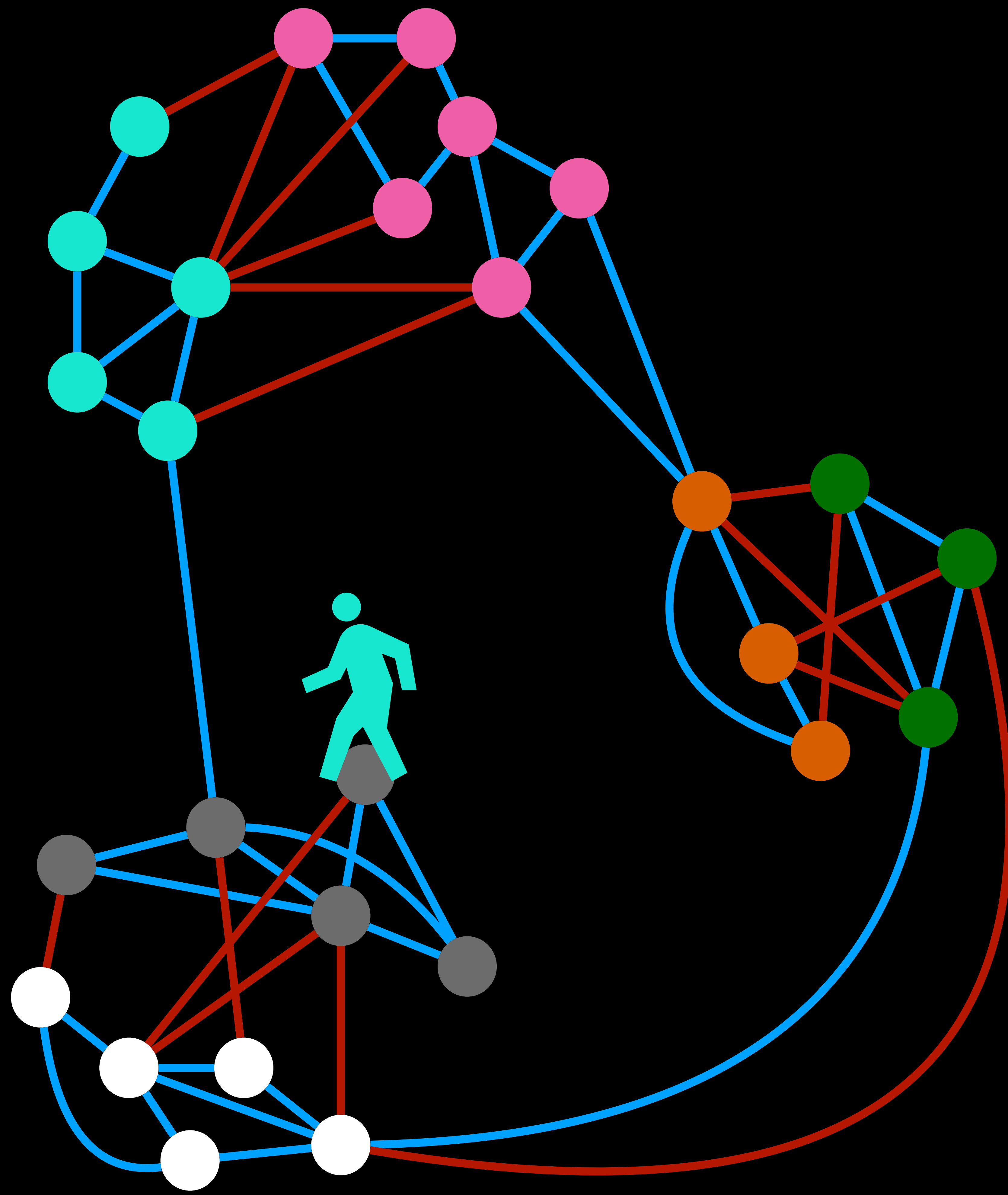
- Initialize sign $s = +$
- When traversing an edge, multiply s with the edge sign



Random Walks With Signs

- Initialize sign $s = +$
- When traversing an edge, multiply s with the edge sign

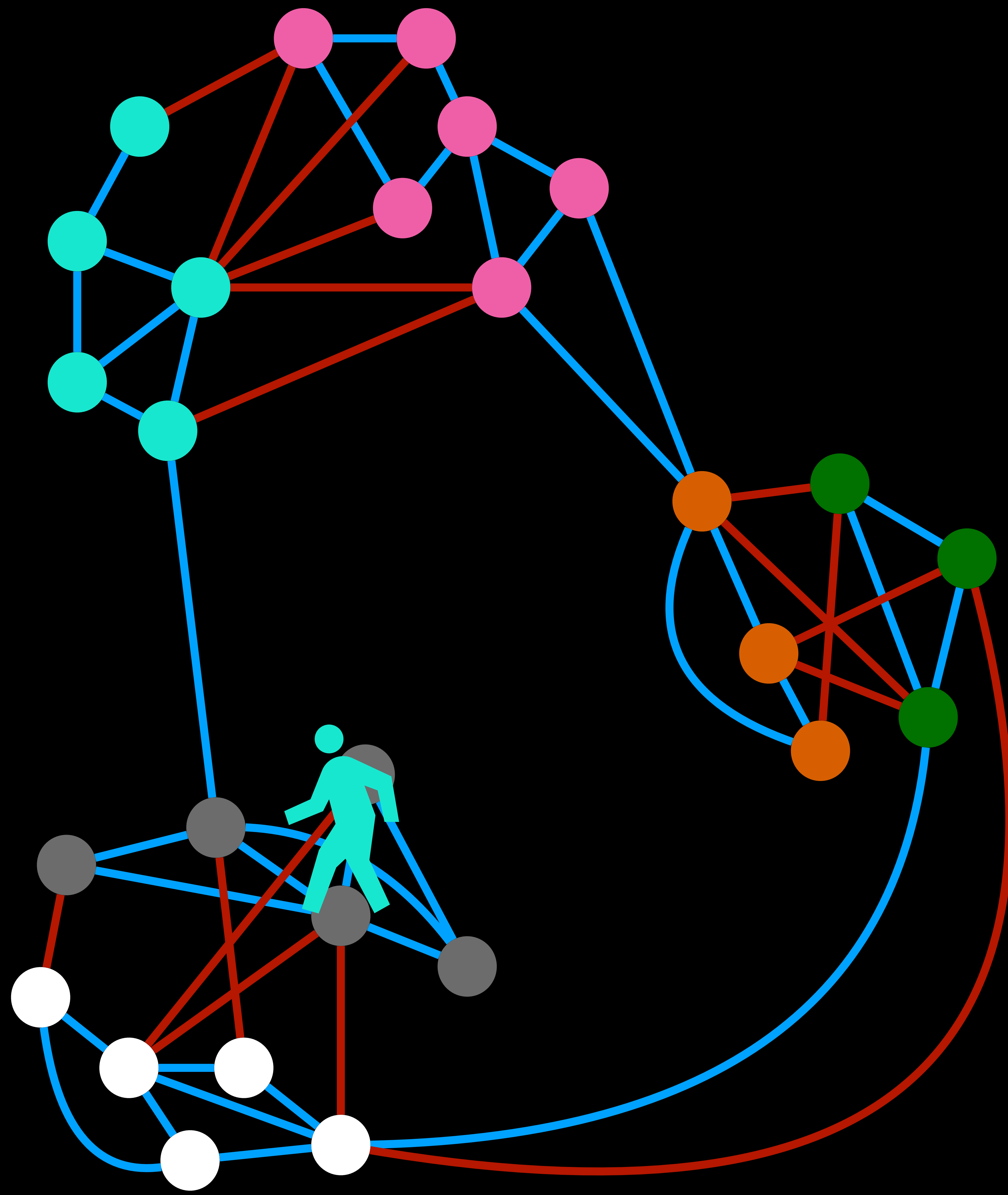
➡ the friend of my friend is my friend
if $s = +$ and we traverse a positive edge, new sign $s = +$



Random Walks With Signs

- Initialize sign $s = +$
- When traversing an edge, multiply s with the edge sign

➡ the friend of my friend is my friend
if $s = +$ and we traverse a positive edge, new sign $s = +$

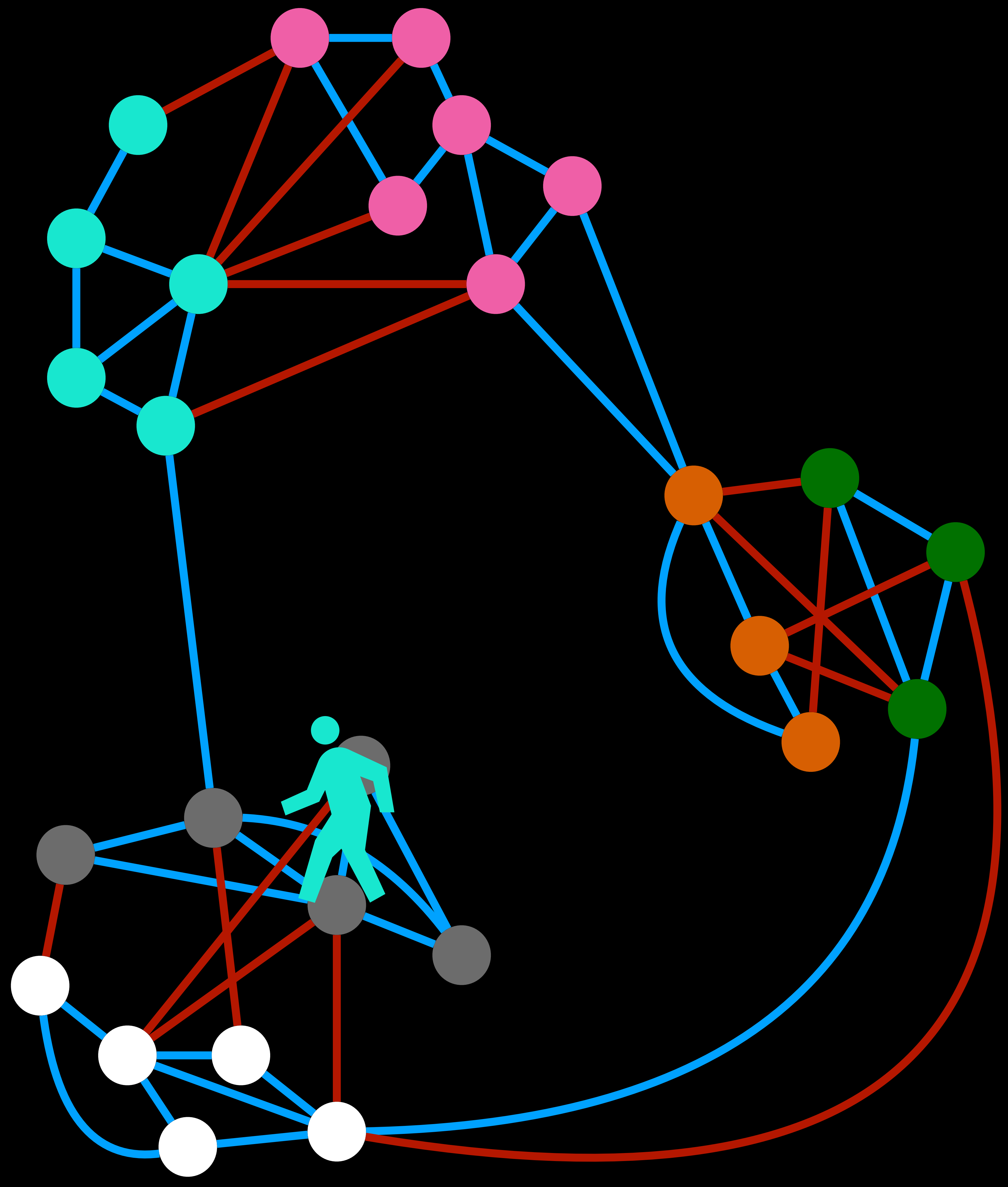


Random Walks With Signs

- Initialize sign $s = +$
- When traversing an edge, multiply s with the edge sign

➡ the friend of my friend is my friend
if $s = +$ and we traverse a positive edge, new sign $s = +$

➡ the enemy of my friend is my enemy
if $s = +$ and we traverse a negative edge, new sign $s = -$

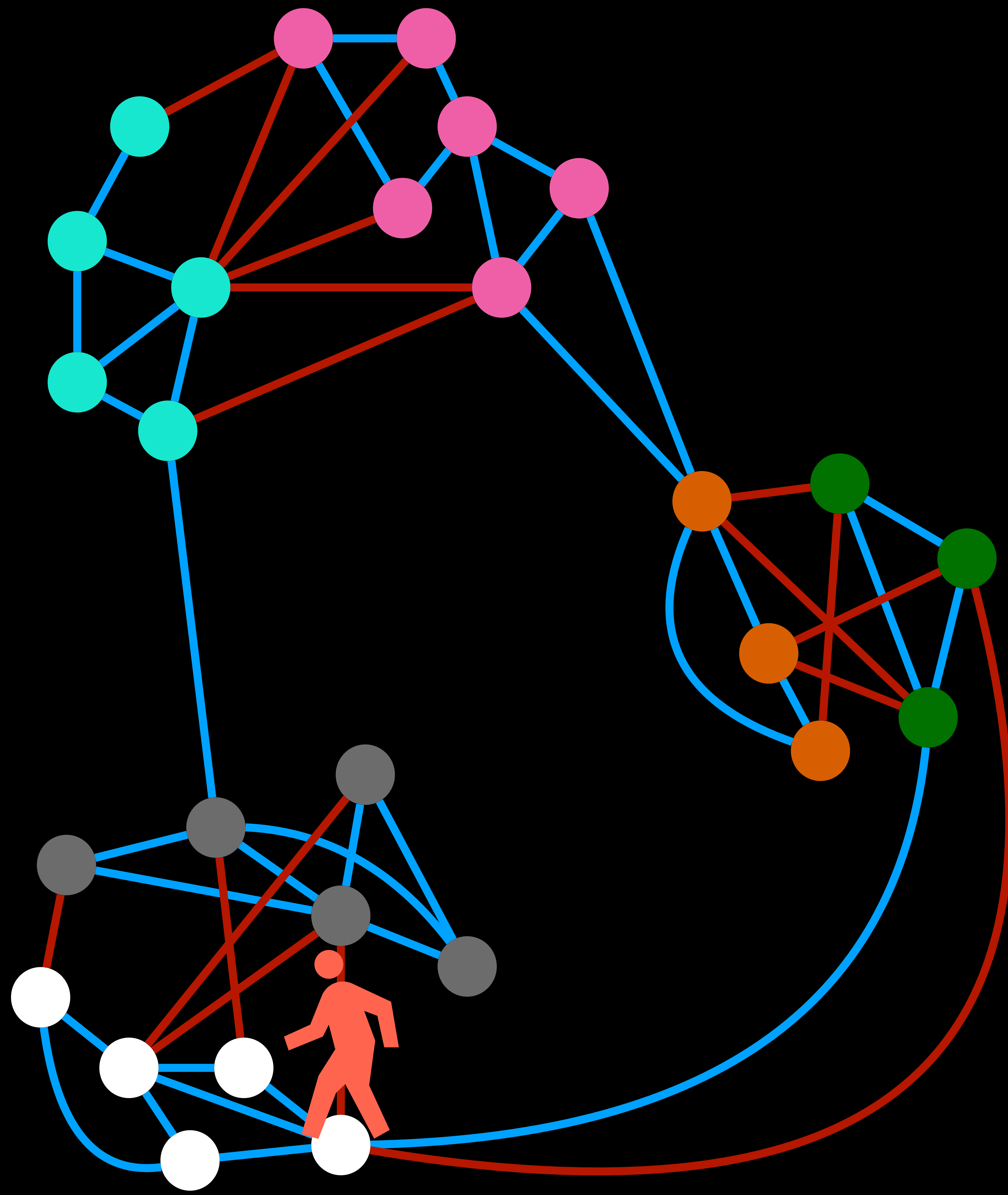


Random Walks With Signs

- Initialize sign $s = +$
- When traversing an edge, multiply s with the edge sign

➡ the friend of my friend is my friend
if $s = +$ and we traverse a positive edge, new sign $s = +$

➡ the enemy of my friend is my enemy
if $s = +$ and we traverse a negative edge, new sign $s = -$



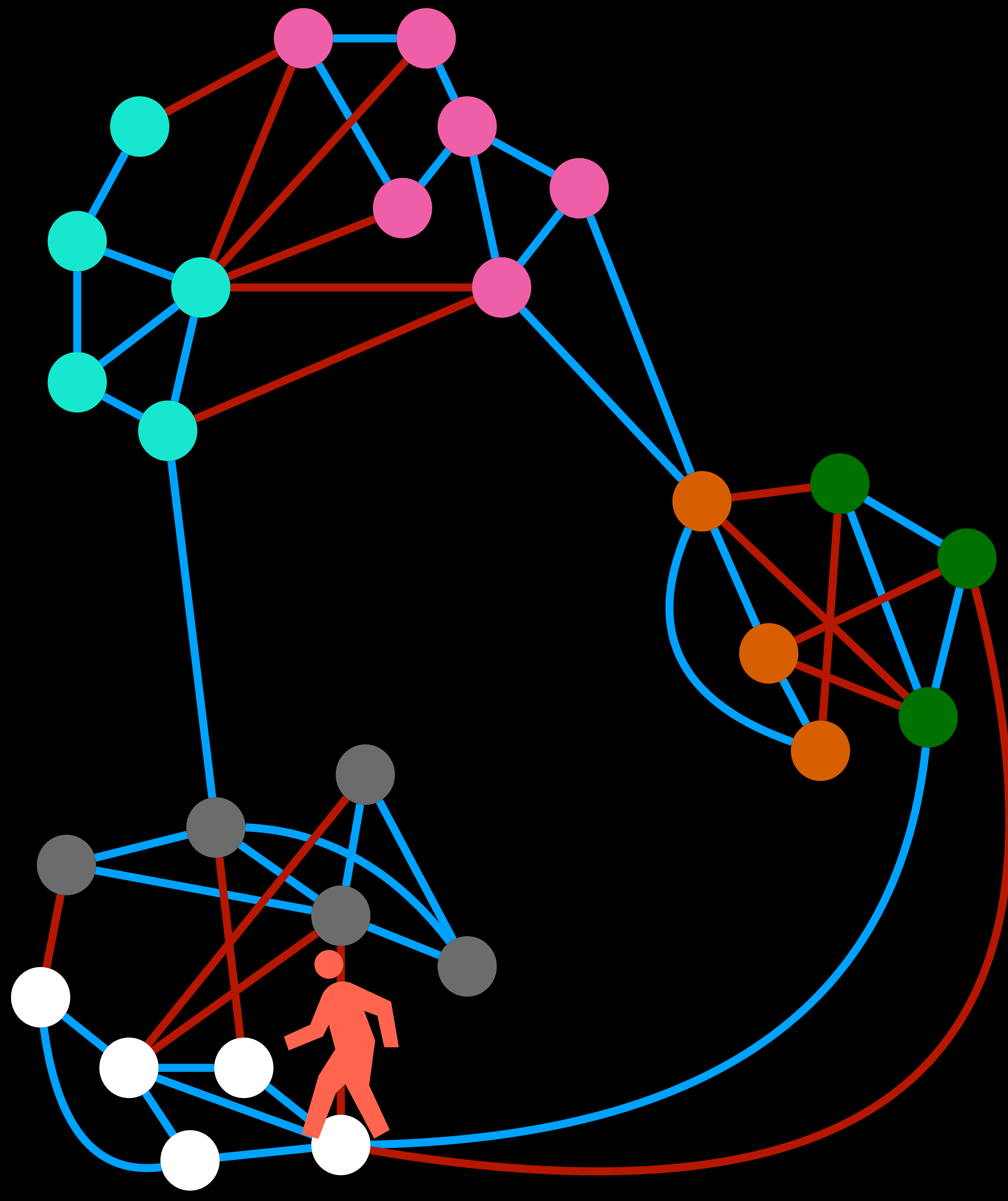
Random Walks With Signs

- Initialize sign $s = +$
- When traversing an edge, multiply s with the edge sign

  ➡ the friend of my friend is my friend
  if $s = +$ and we traverse a positive edge, new sign $s = +$

  ➡ the enemy of my friend is my enemy
  if $s = +$ and we traverse a negative edge, new sign $s = -$

  ➡ the friend of my enemy is my enemy
  if $s = -$ and we traverse a positive edge, new sign $s = -$



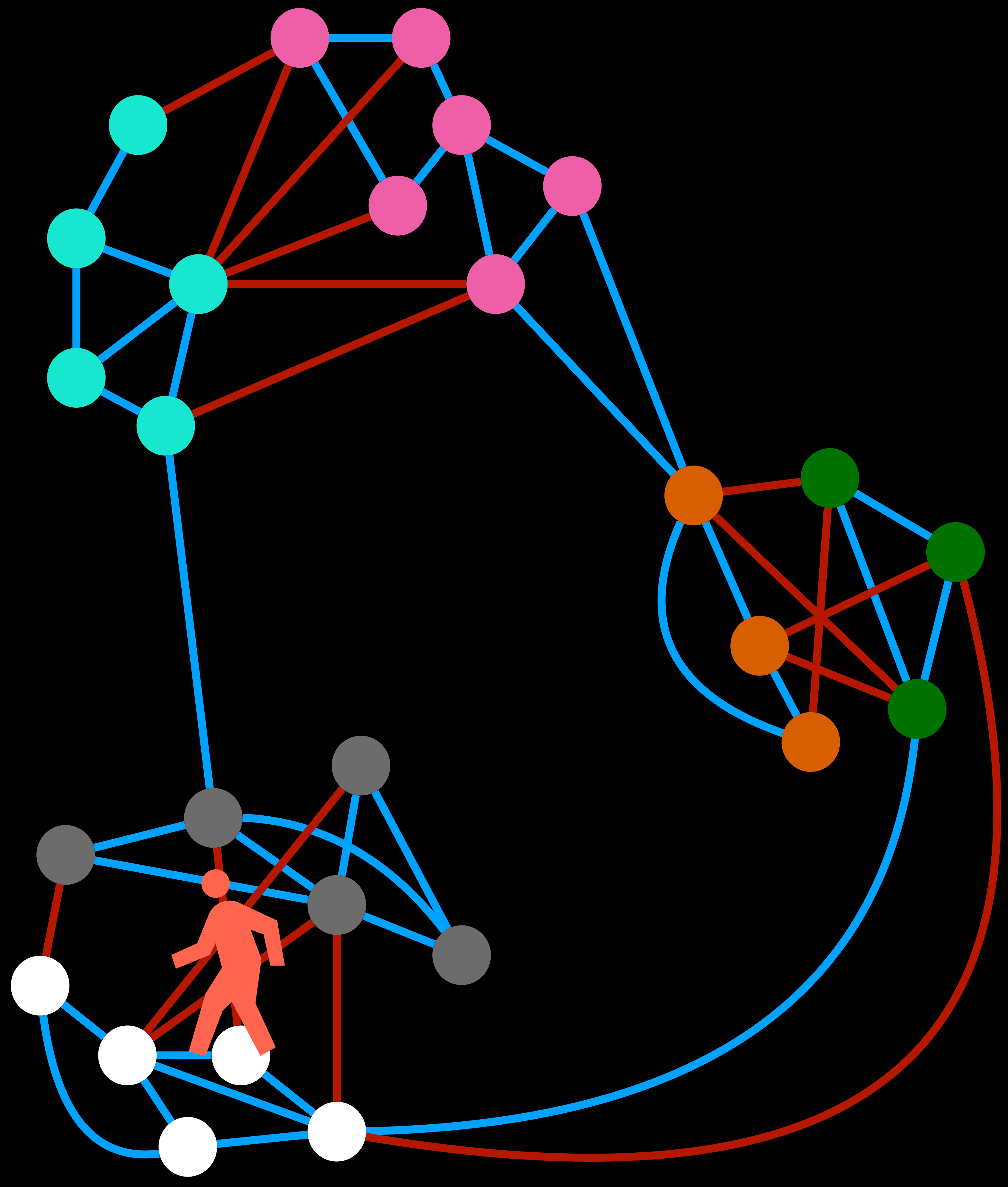
Random Walks With Signs

- Initialize sign $s = +$
- When traversing an edge, multiply s with the edge sign

➡ the friend of my friend is my friend
if $s = +$ and we traverse a positive edge, new sign $s = +$

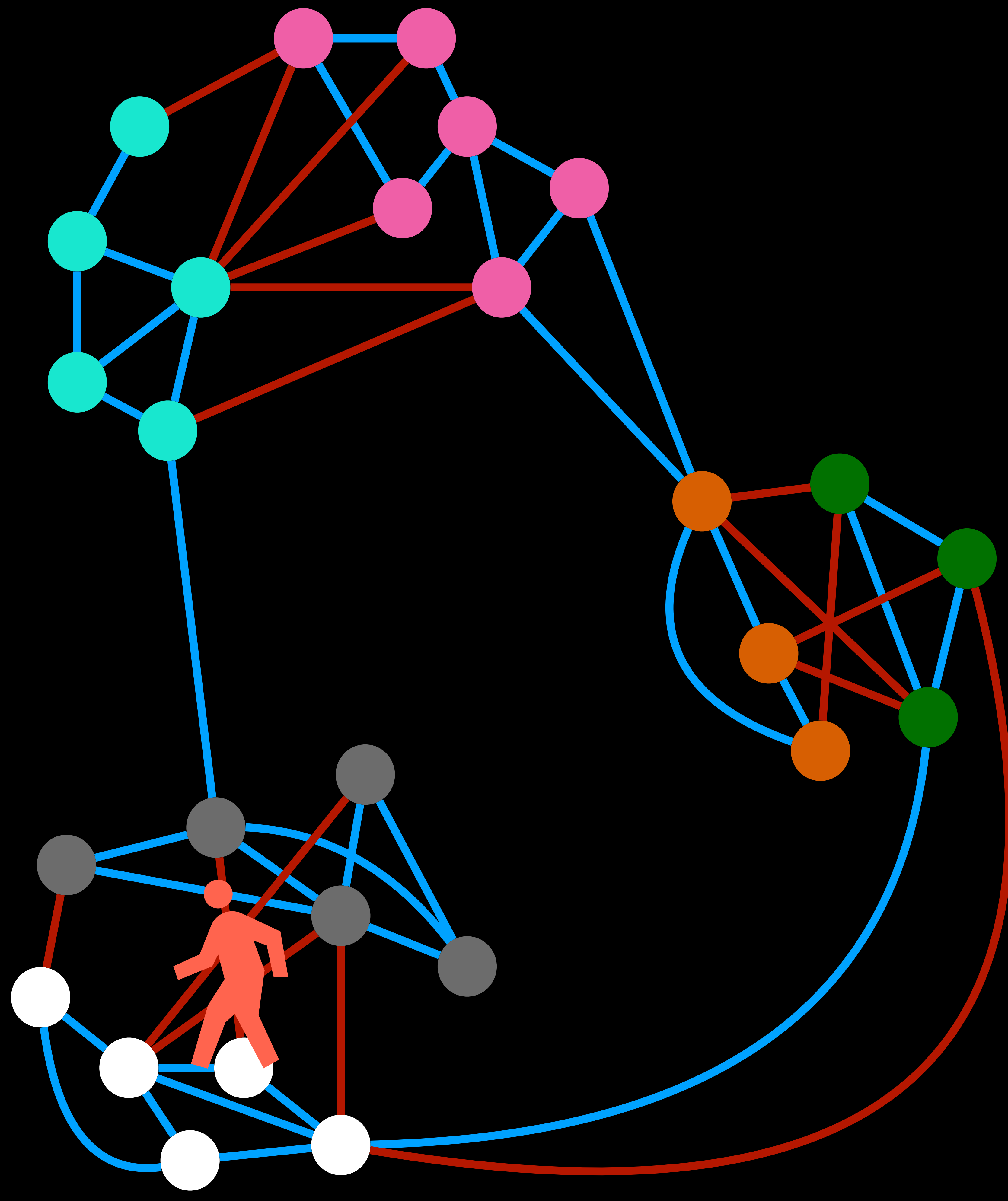
➡ the enemy of my friend is my enemy
if $s = +$ and we traverse a negative edge, new sign $s = -$

➡ the friend of my enemy is my enemy
if $s = -$ and we traverse a positive edge, new sign $s = -$



Random Walks With Signs

- Initialize sign $s = +$
- When traversing an edge, multiply s with the edge sign
  - ➡ the friend of my friend is my friend
    if $s = +$ and we traverse a positive edge, new sign $s = +$
  - ➡ the enemy of my friend is my enemy
    if $s = +$ and we traverse a negative edge, new sign $s = -$
  - ➡ the friend of my enemy is my enemy
    if $s = -$ and we traverse a positive edge, new sign $s = -$
  - ➡ the enemy of my enemy is my friend
    if $s = -$ and we traverse a negative edge, new sign $s = +$



Random Walks With Signs

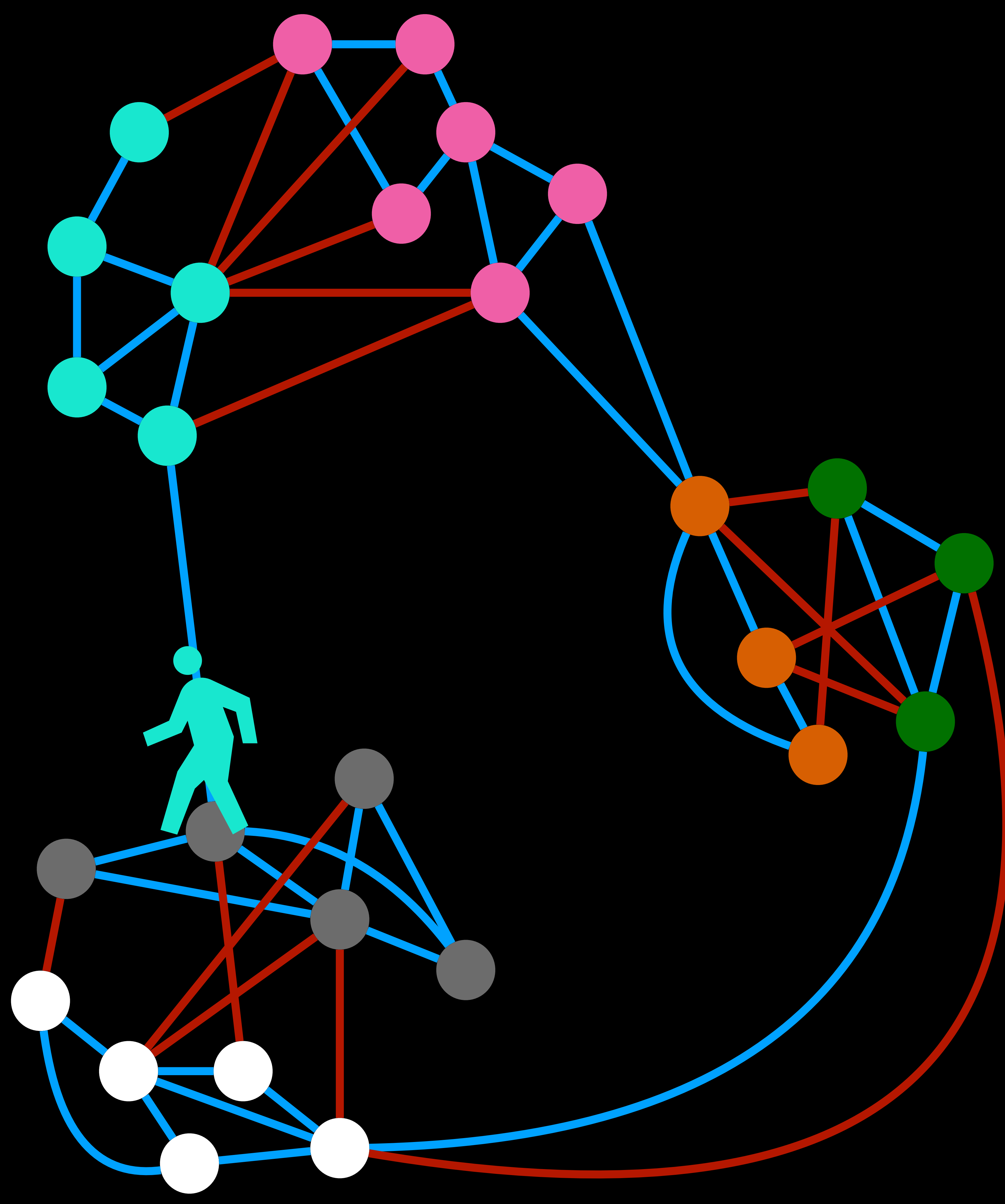
- Initialize sign $s = +$
- When traversing an edge, multiply s with the edge sign

➡ the friend of my friend is my friend
if $s = +$ and we traverse a positive edge, new sign $s = +$

➡ the enemy of my friend is my enemy
if $s = +$ and we traverse a negative edge, new sign $s = -$

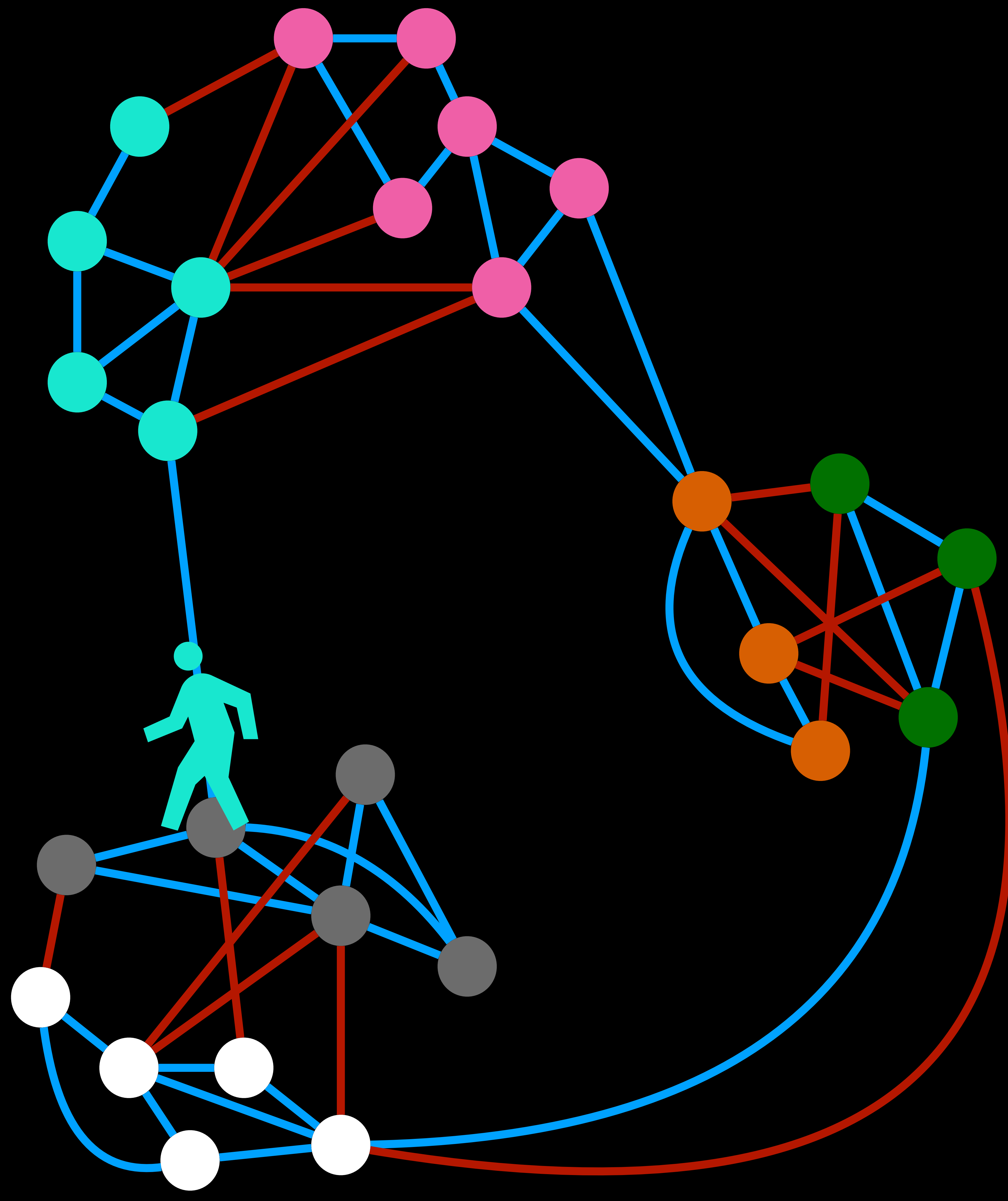
➡ the friend of my enemy is my enemy
if $s = -$ and we traverse a positive edge, new sign $s = -$

➡ the enemy of my enemy is my friend
if $s = -$ and we traverse a negative edge, new sign $s = +$



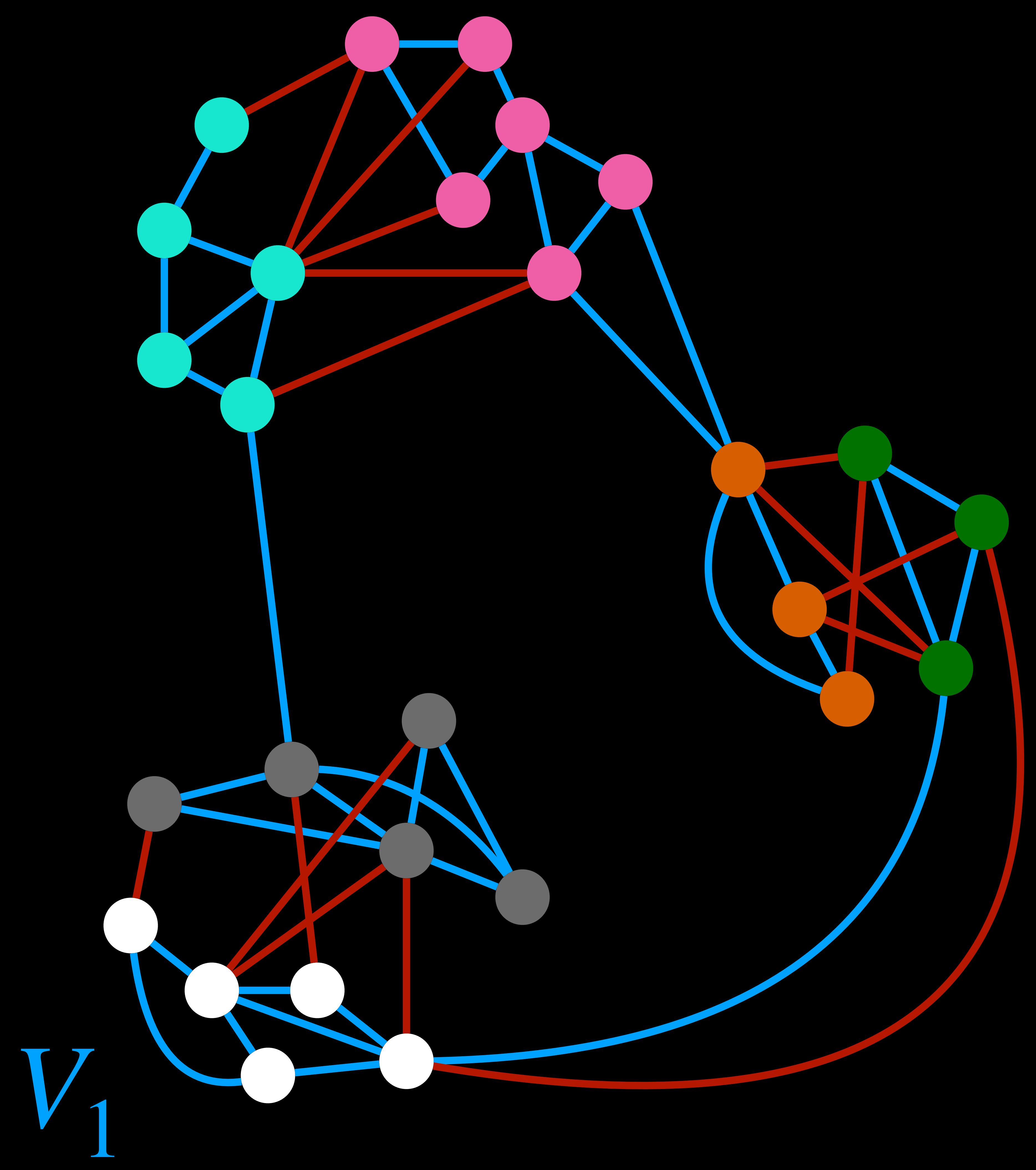
Random Walks With Signs

- Initialize sign $s = +$
- When traversing an edge, multiply s with the edge sign
  - ➡ the friend of my friend is my friend
    if $s = +$ and we traverse a positive edge, new sign $s = +$
  - ➡ the enemy of my friend is my enemy
    if $s = +$ and we traverse a negative edge, new sign $s = -$
  - ➡ the friend of my enemy is my enemy
    if $s = -$ and we traverse a positive edge, new sign $s = -$
  - ➡ the enemy of my enemy is my friend
    if $s = -$ and we traverse a negative edge, new sign $s = +$
- We do not make any changes to the randomness of the walk



Some Intuition

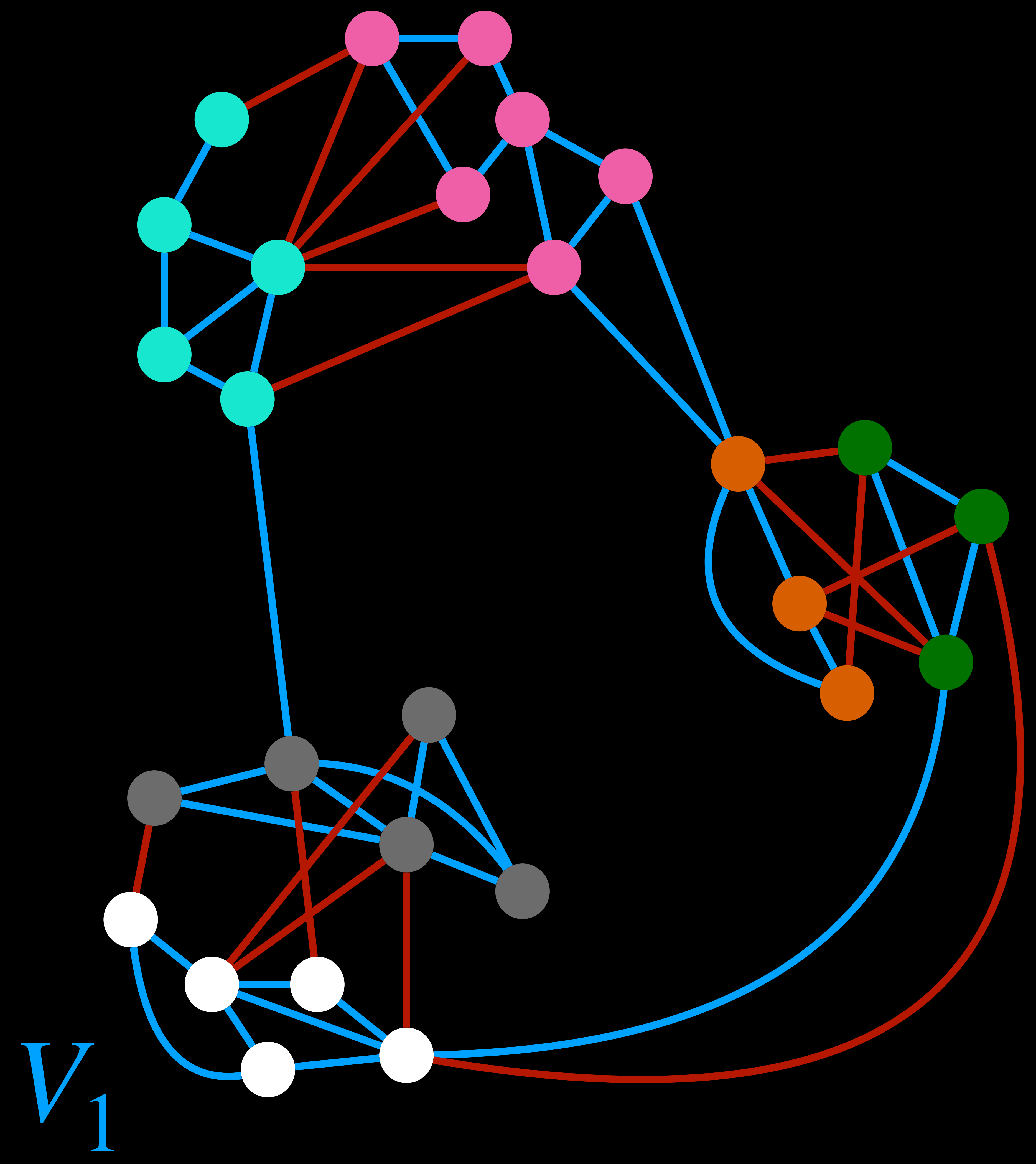
What signs of random walks do we expect?



Some Intuition

What signs of random walks do we expect?

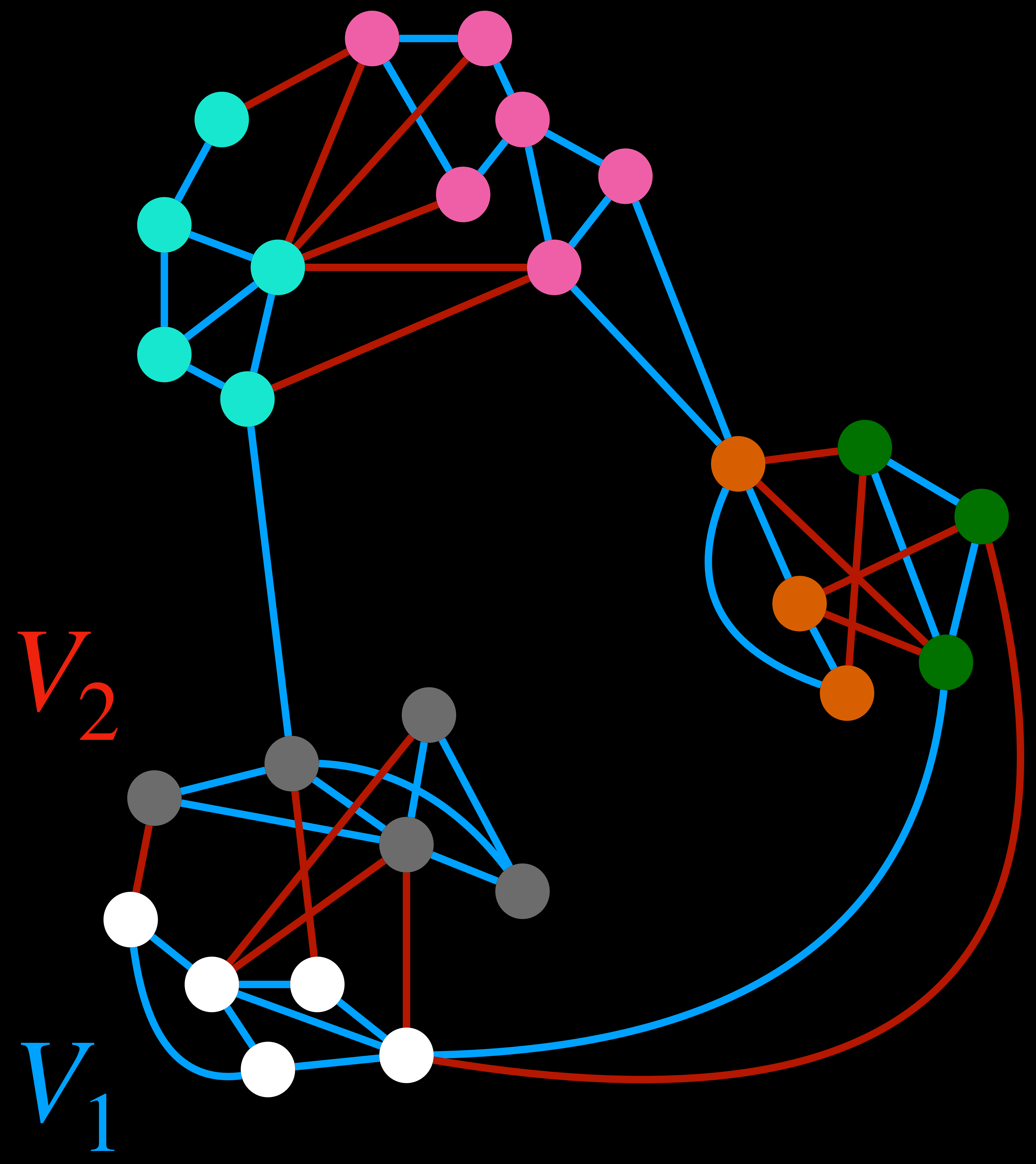
- If we start in V_1 and end in V_1, sign should be $+$



Some Intuition

What signs of random walks do we expect?

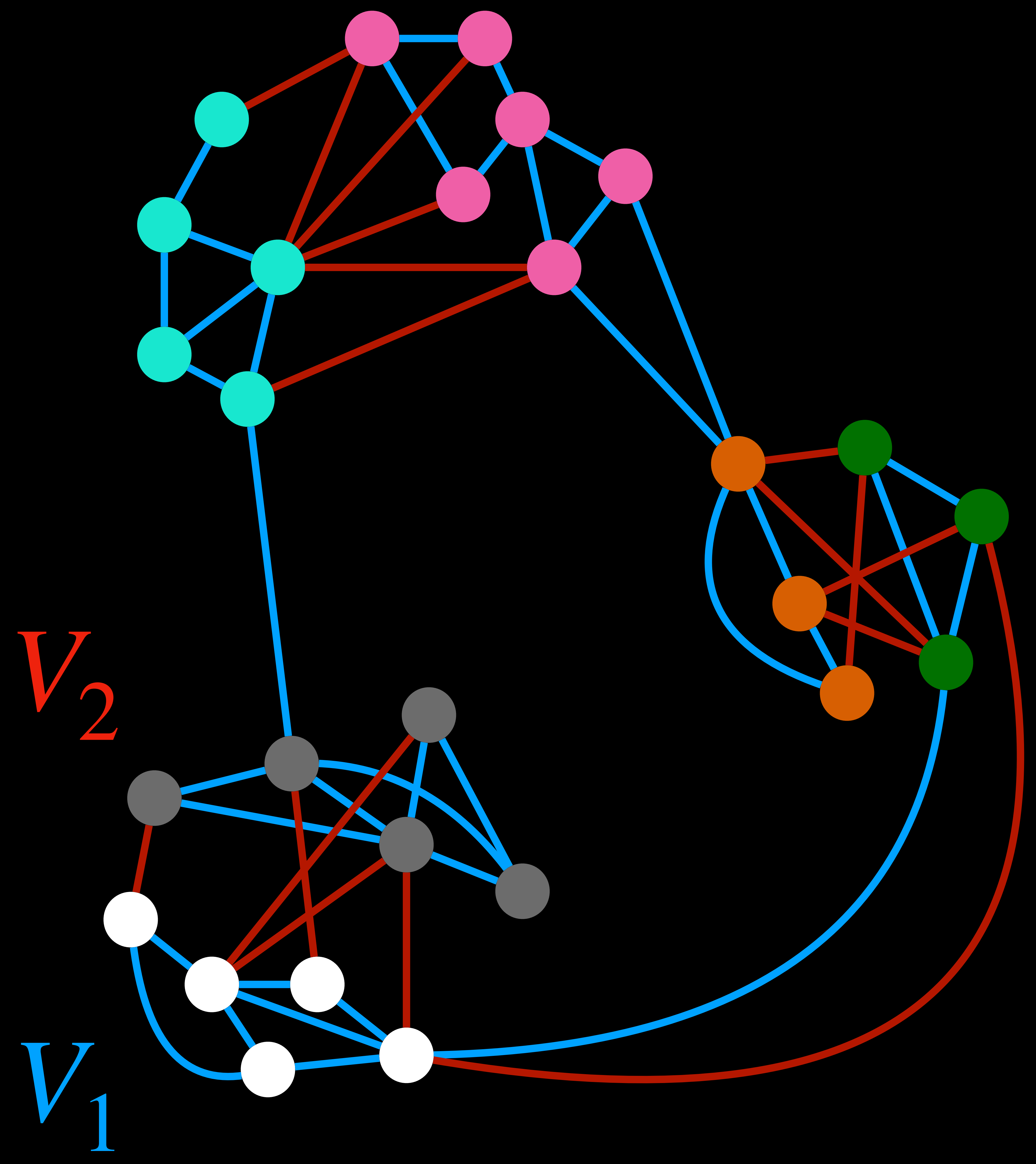
- If we start in V_1 and end in V_1, sign should be $+$
- If we start in V_1 and end in V_2, sign should be $-$



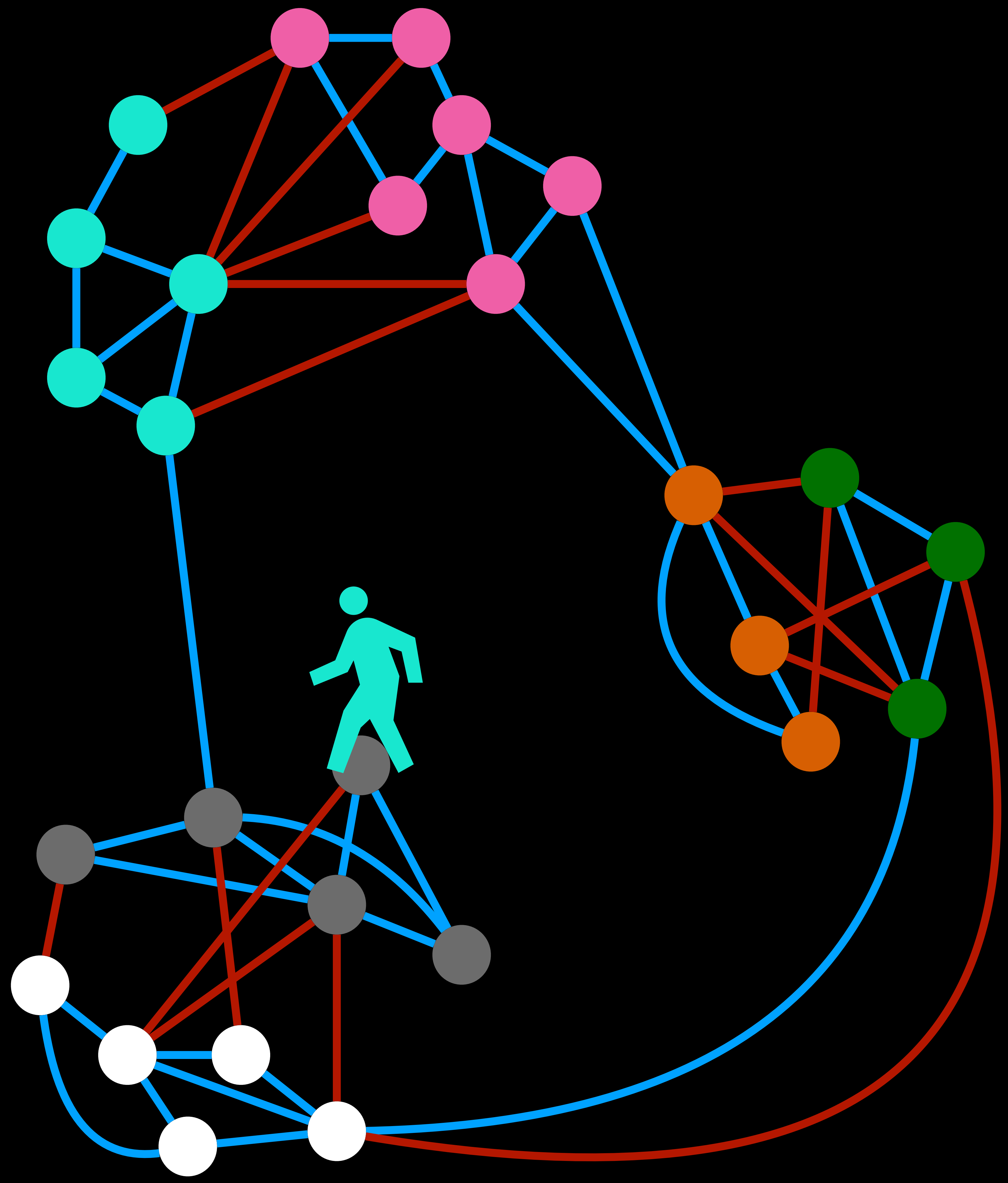
Some Intuition

What signs of random walks do we expect?

- If we start in V_1 and end in V_1, sign should be $+$
- If we start in V_1 and end in V_2, sign should be $-$
- If we start in V_1 then the probability to end in $V\backslash(V_1 \cup V_2)$ should be very small

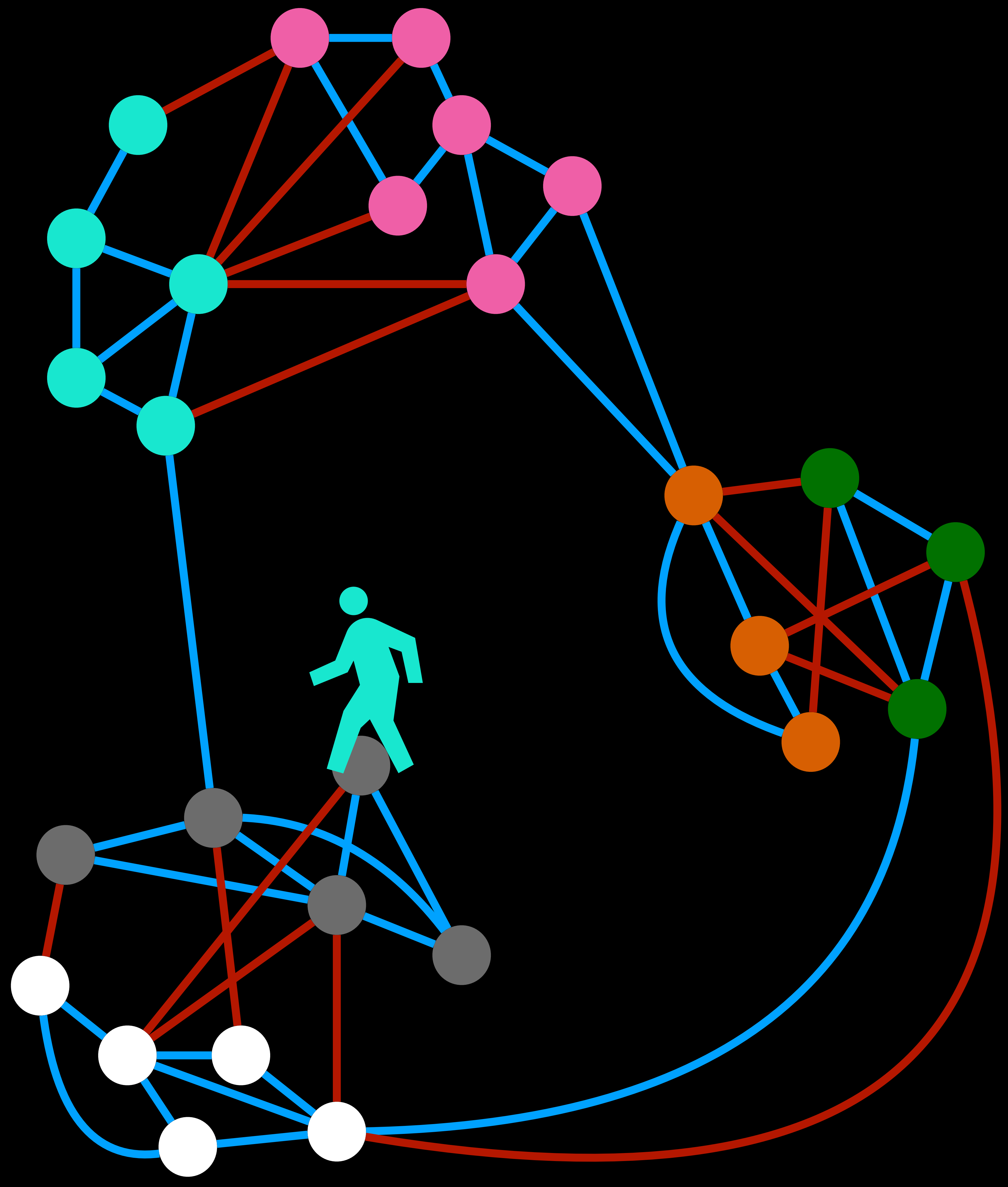


Community Vectors



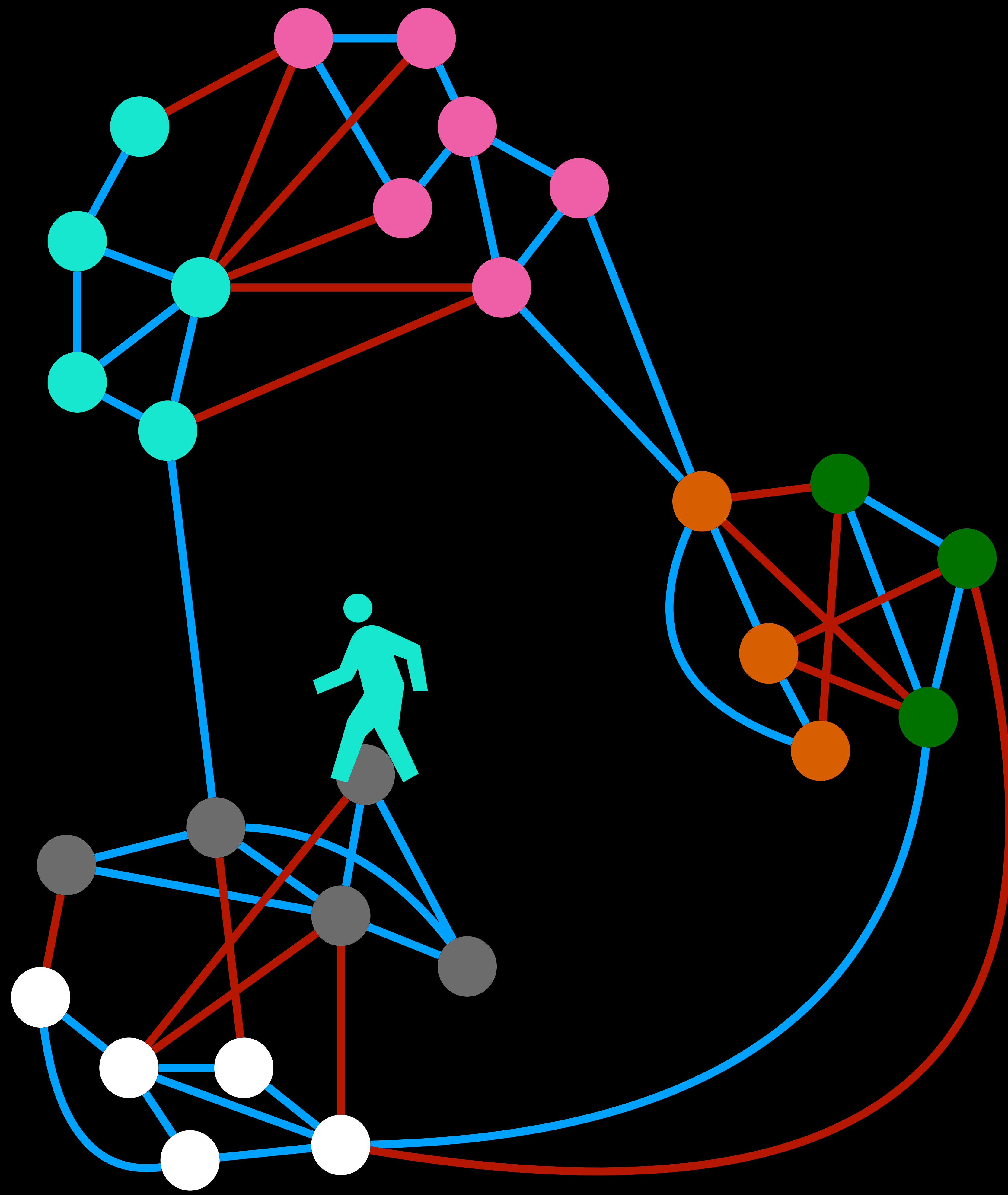
Community Vectors

- SignedCommunityVector(u):



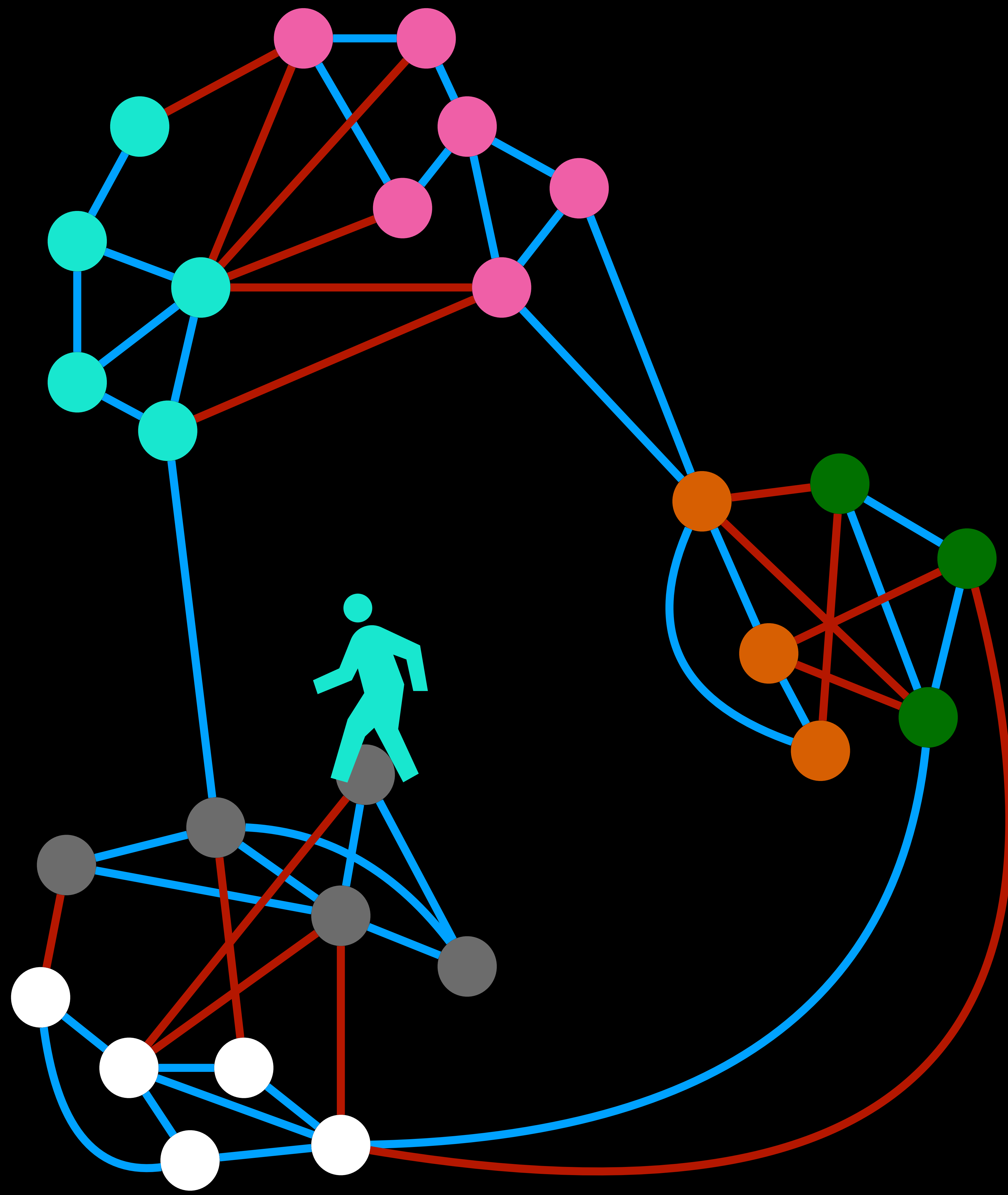
Community Vectors

- SignedCommunityVector(u):
  - Initialize an empty vector $m \in \mathbb{R}^n$



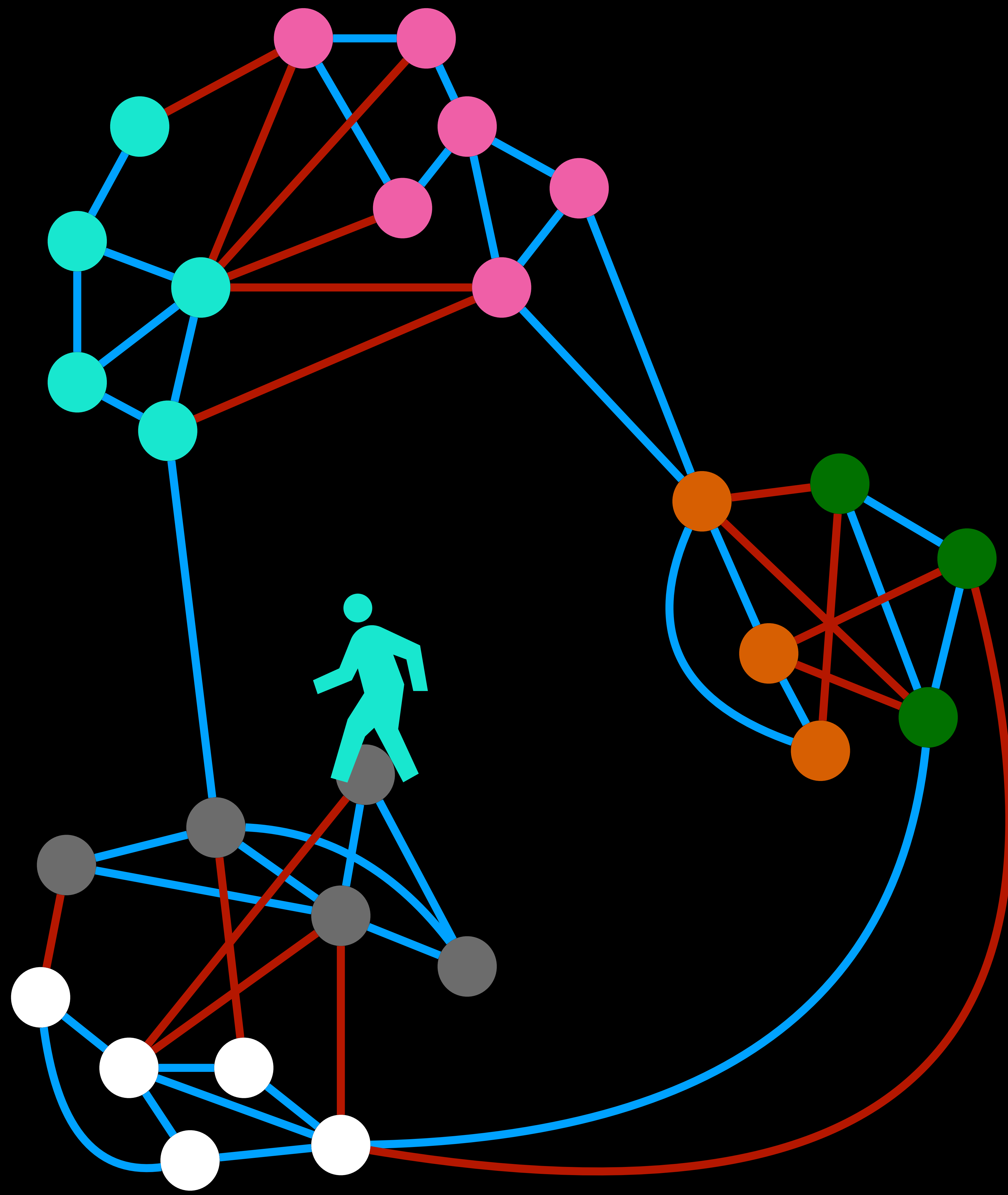
Community Vectors

- SignedCommunityVector(u):
  - Initialize an empty vector $m \in \mathbb{R}^n$
  - Perform $\sqrt{n}$ random walks of length $\log n$ starting at u



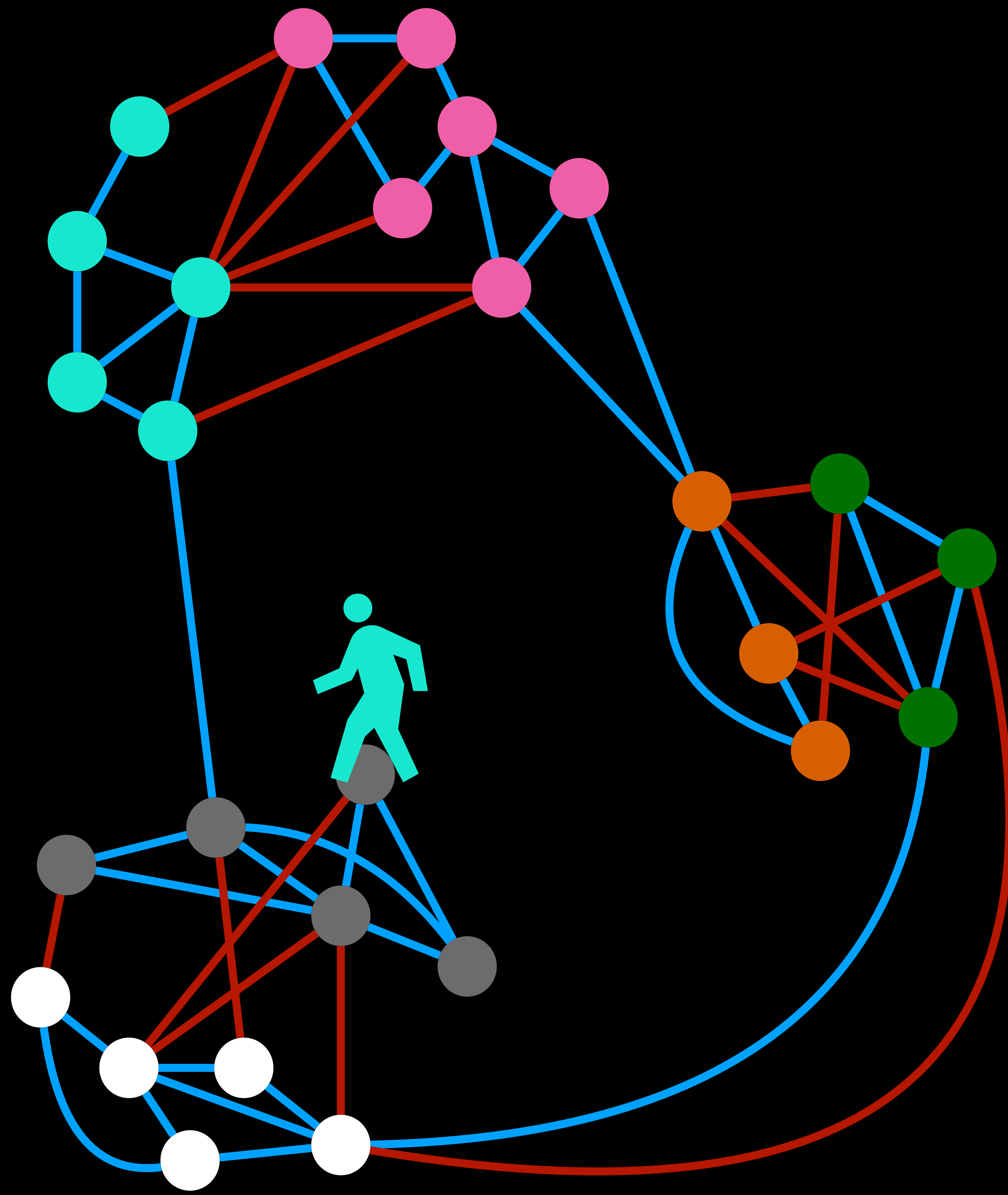
Community Vectors

- SignedCommunityVector(u):
    - Initialize an empty vector $m \in \mathbb{R}^n$
    - Perform $\sqrt{n}$ random walks of length $\log n$ starting at u
    - For each $w \in V$:



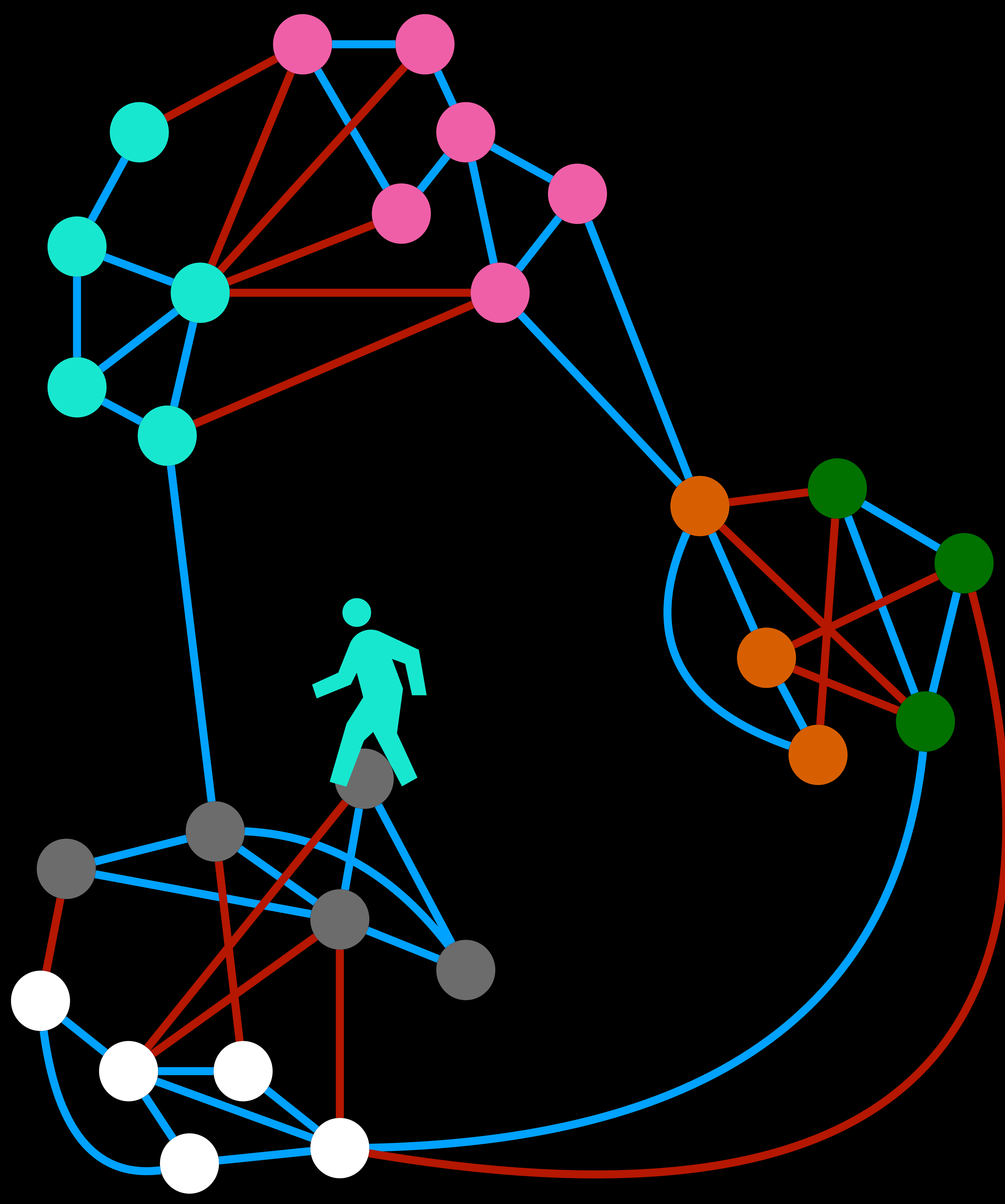
Community Vectors

- SignedCommunityVector(u):
  - Initialize an empty vector $m \in \mathbb{R}^n$
  - Perform $\sqrt{n}$ random walks of length $\log n$ starting at u
  - For each $w \in V$:
    - $m_+(w) \leftarrow$ fraction of random walks that end at w with sign $+$



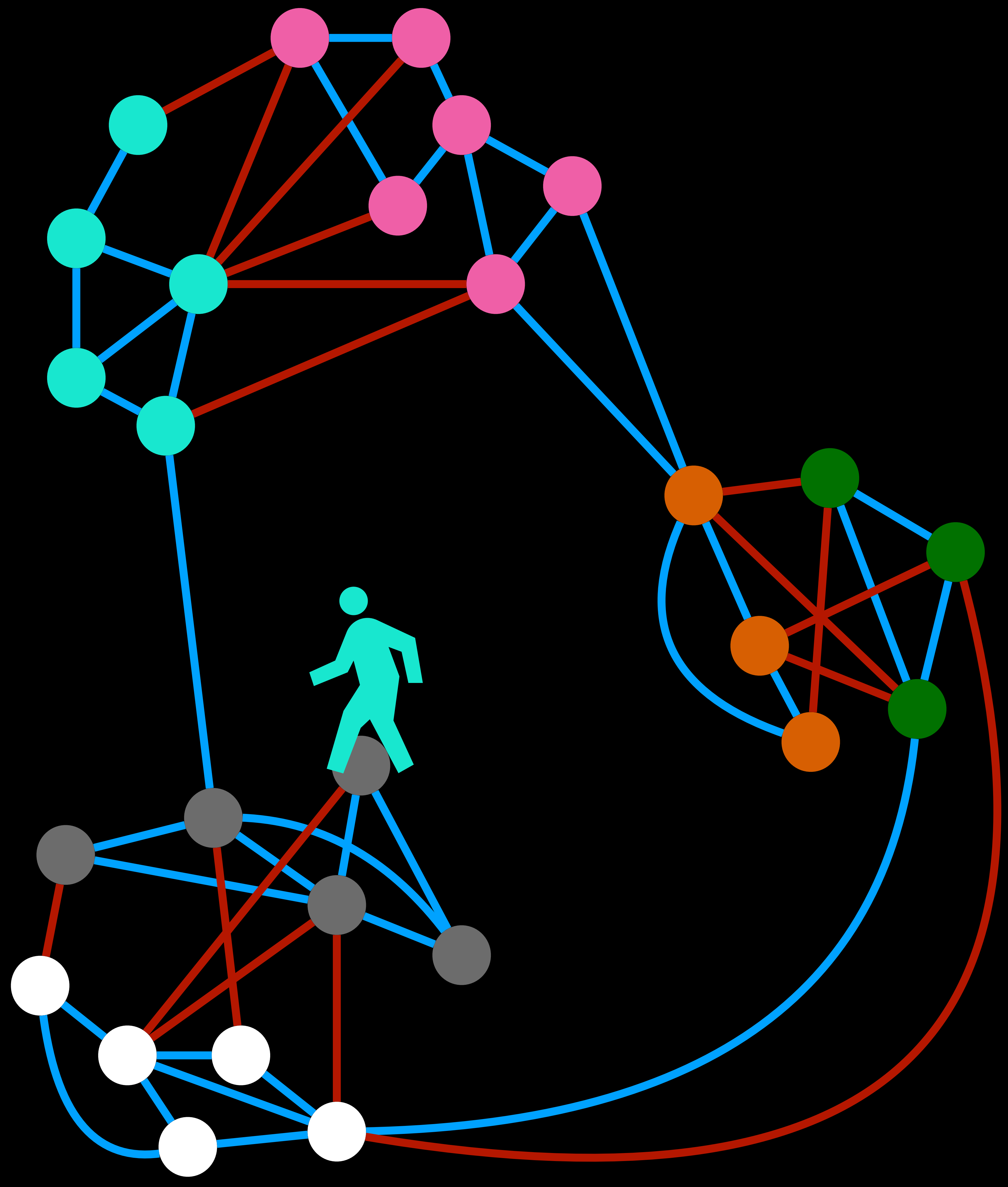
Community Vectors

- SignedCommunityVector(u):
  - Initialize an empty vector $m \in \mathbb{R}^n$
  - Perform $\sqrt{n}$ random walks of length $\log n$ starting at u
  - For each $w \in V$:
    - $m_+(w) \leftarrow$ fraction of random walks that end at w with sign $+$
    - $m_-(w) \leftarrow$ fraction of random walks that end at w with sign $-$



Community Vectors

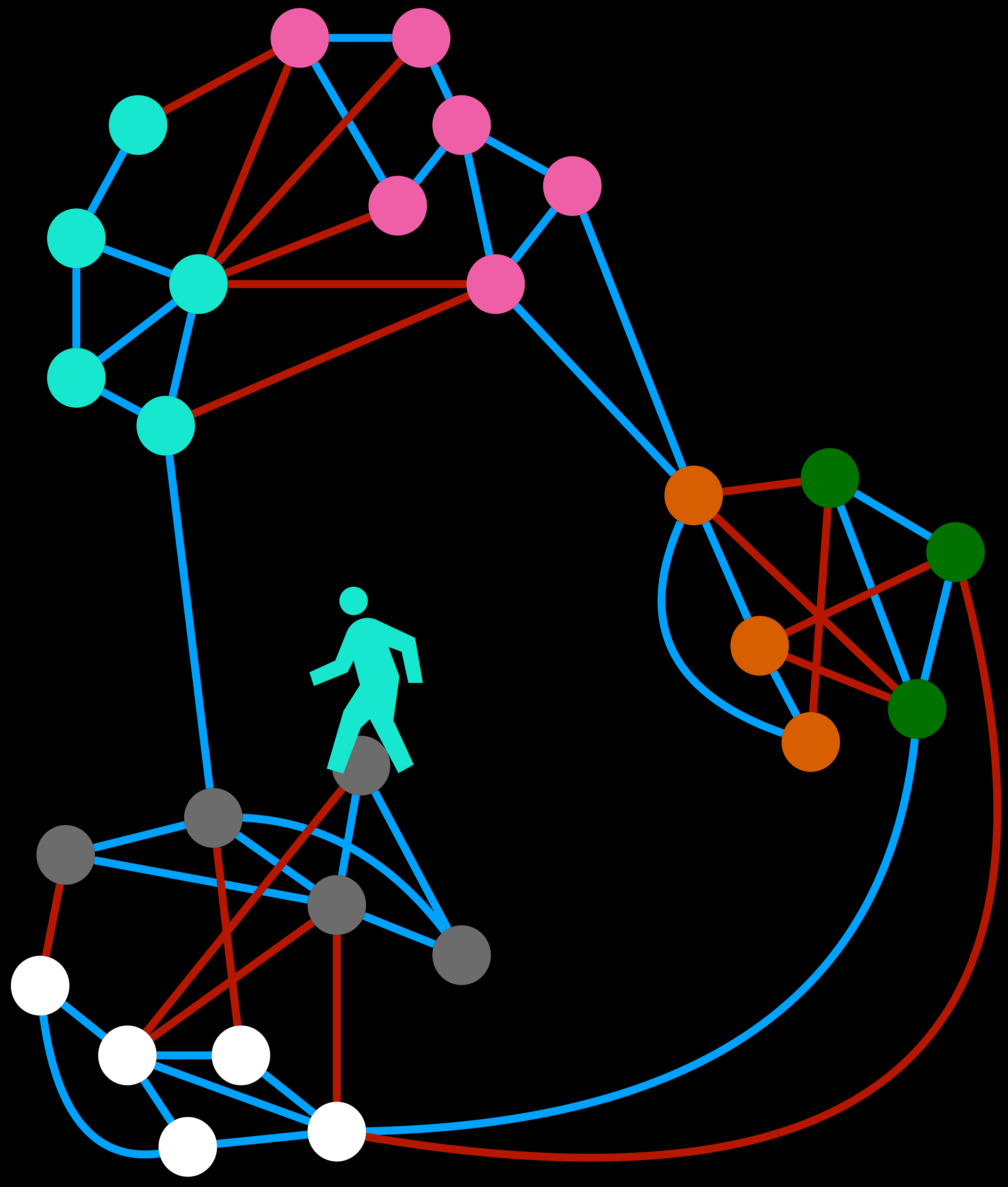
- SignedCommunityVector(u):
  - Initialize an empty vector $m \in \mathbb{R}^n$
  - Perform $\sqrt{n}$ random walks of length $\log n$ starting at u
  - For each $w \in V$:
    - $m_+(w) \leftarrow$ fraction of random walks that end at w with sign $+$
    - $m_-(w) \leftarrow$ fraction of random walks that end at w with sign $-$
    - $m(w) \leftarrow m_+(w) - m_-(w)$



Community Vectors

- SignedCommunityVector(u):
  - Initialize an empty vector $m \in \mathbb{R}^n$
  - Perform $\sqrt{n}$ random walks of length $\log n$ starting at u
  - For each $w \in V$:
    - $m_+(w) \leftarrow$ fraction of random walks that end at w with sign $+$
    - $m\_(w) \leftarrow$ fraction of random walks that end at w with sign $-$
    - $m(w) \leftarrow m_+(w) - m\_(w)$

➡ we expect that if w is on the same side as u then $m(w) \gg 0$,
if w is on the other side of u then $m(w) \ll 0$, and
if w is from a different cluster then $m(w) \approx 0$



Community Vectors

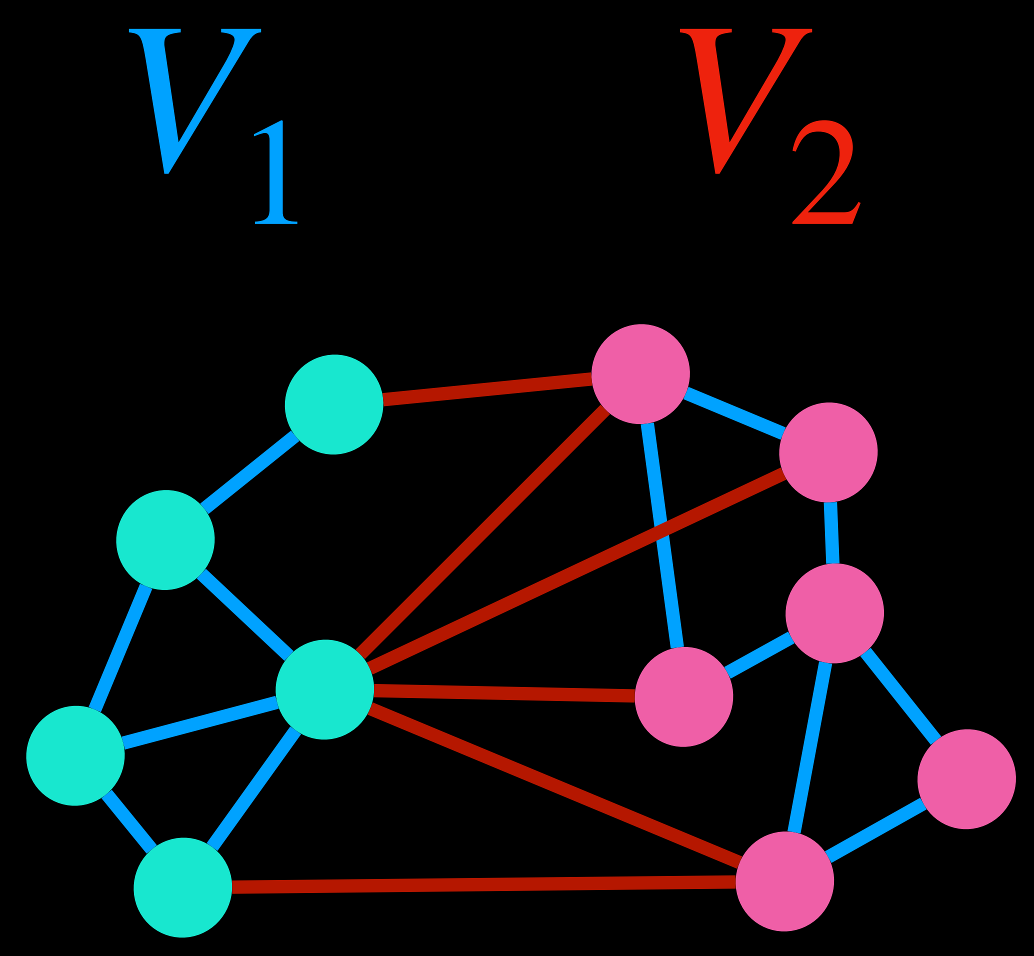
- SignedCommunityVector(u):
  - Initialize an empty vector $m \in \mathbb{R}^n$
  - Perform $\sqrt{n}$ random walks of length $\log n$ starting at u
  - For each $w \in V$:
    - $m_+(w) \leftarrow$ fraction of random walks that end at w with sign $+$
    - $m\_(w) \leftarrow$ fraction of random walks that end at w with sign $-$
    - $m(w) \leftarrow m_+(w) - m\_(w)$

➡ we expect that if w is on the same side as u then $m(w) \gg 0$,
if w is on the other side of u then $m(w) \ll 0$, and
if w is from a different cluster then $m(w) \approx 0$

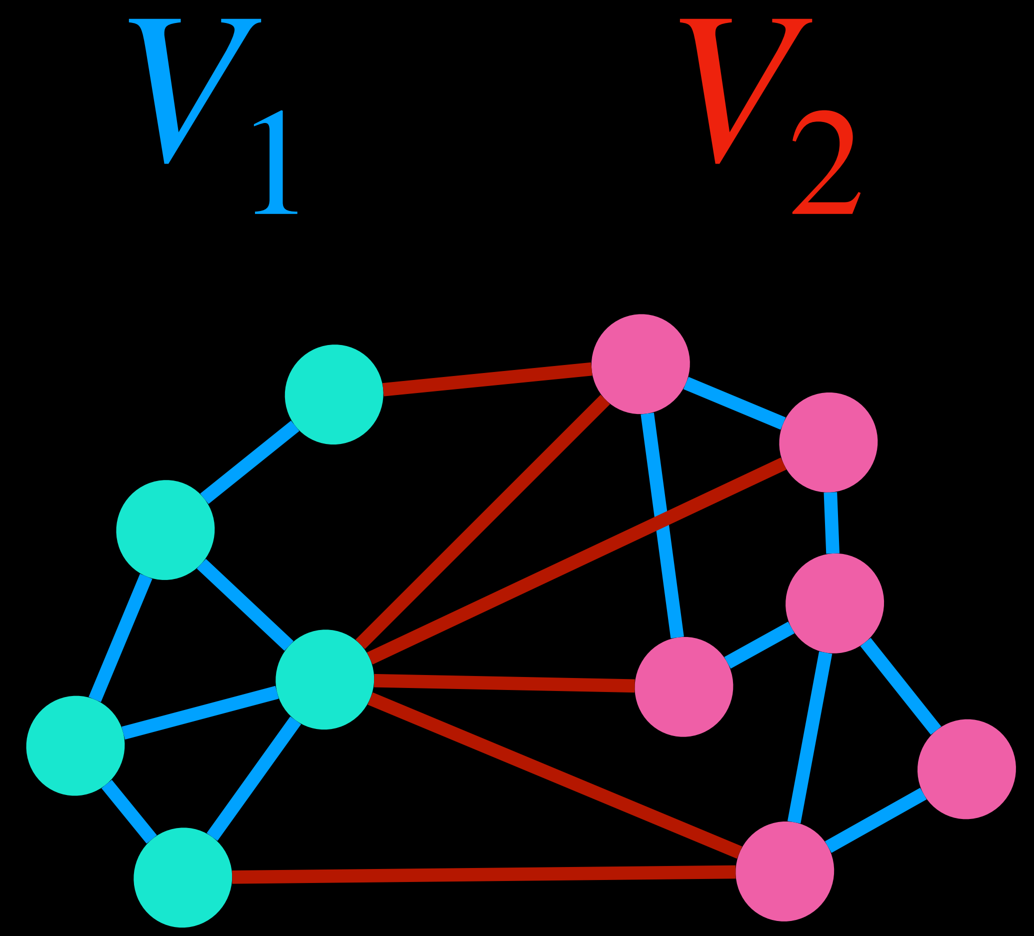
- Suppose that for each cluster $C_1, \ldots, C_k$ we know a seed-vertex $v_i \in C_i$
- **WhichCluster(u):**
  - $m_u \leftarrow$ SignedCommunityVector(u)
  - $m_{v_i} \leftarrow$ SignedCommunityVector(v_i) for all i
  - $i^* = \arg\min_i \min\{\|m_u - m_{v_i}\|_2, \|m_u + m_{v_i}\|_2\}$
  - Return that u is from cluster i^*

➡ **Intuition:** if u is from cluster i^*,
then the community vectors of m_u and $m_{v_{i^*}}$
should be similar

Main Technical Result

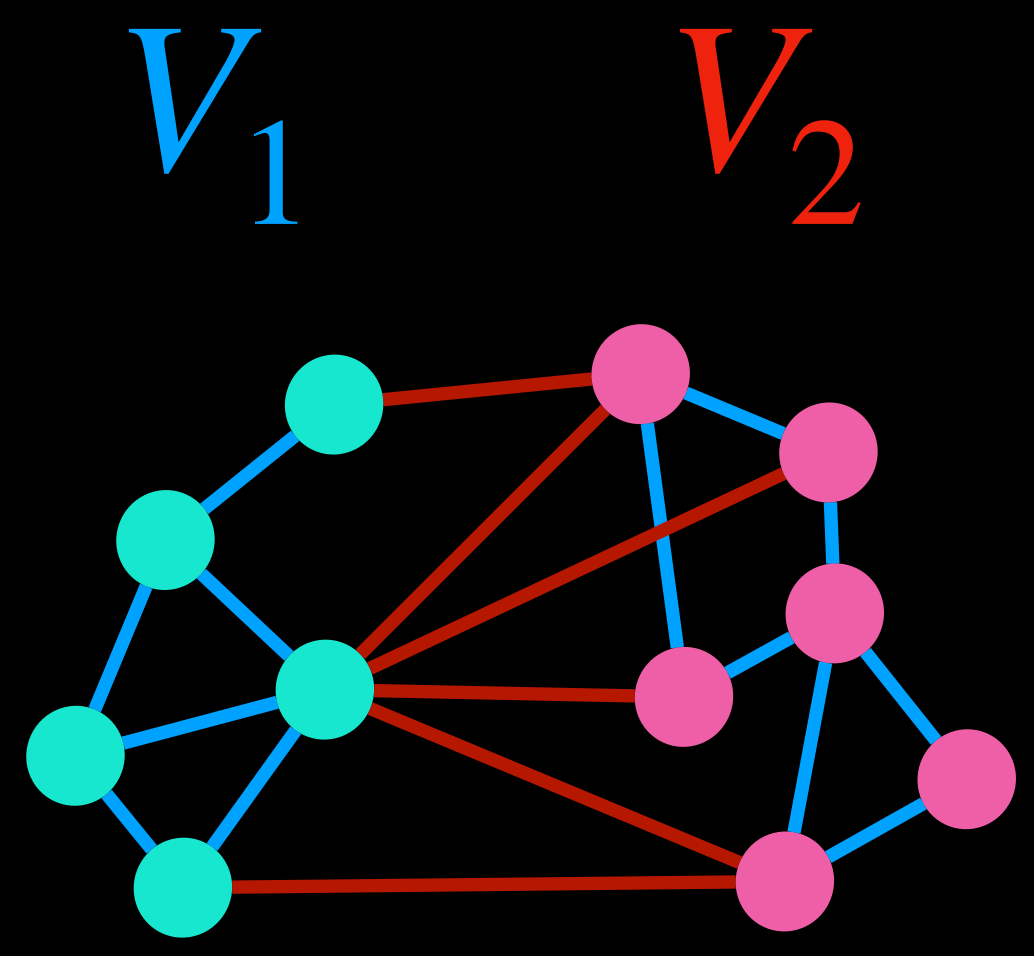


Main Technical Result



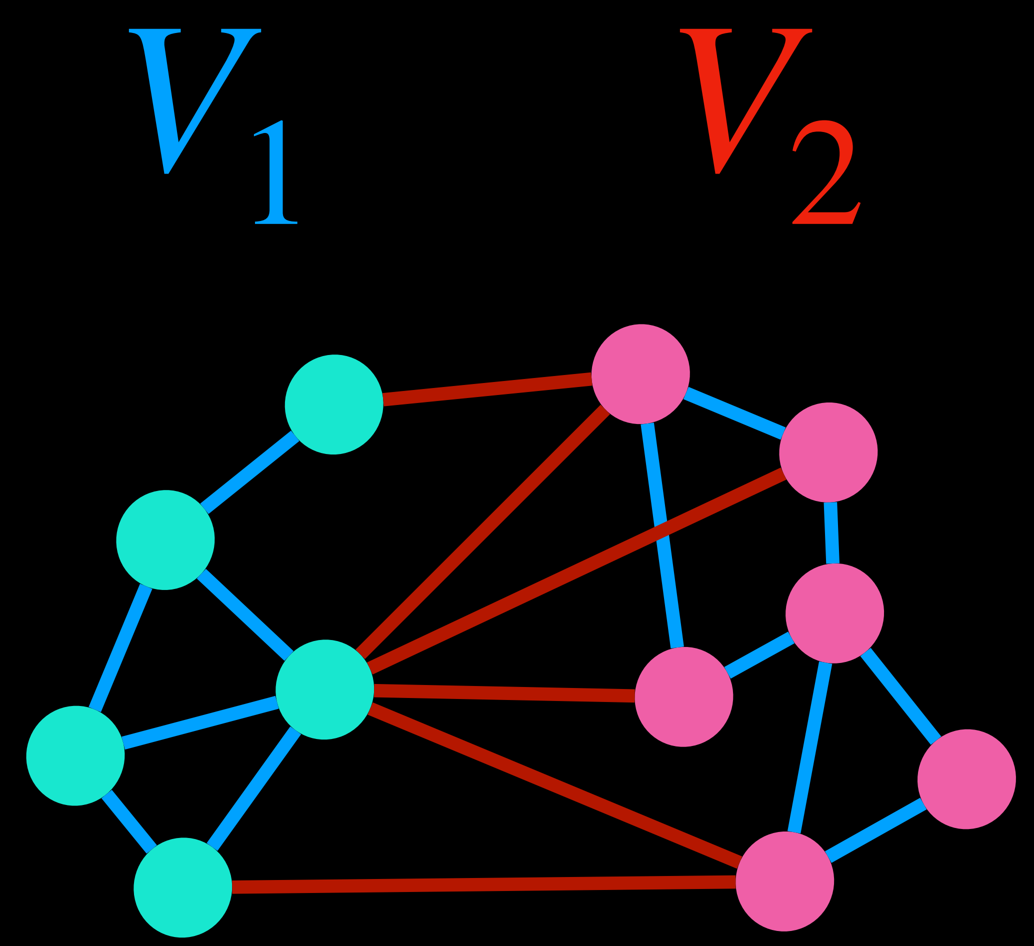
- Let $\mathscr{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ be the *signed normalized Laplacian*

Main Technical Result



- Let $\mathscr{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ be the *signed normalized Laplacian*
- Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be the orthonormal row eigenvectors of $\mathscr{L}$, i.e., $\lambda_i \mathbf{v}_i = \mathbf{v}_i \mathscr{L}$, and assume that $0 \leq \lambda_1 \leq \cdots \leq \lambda_n$

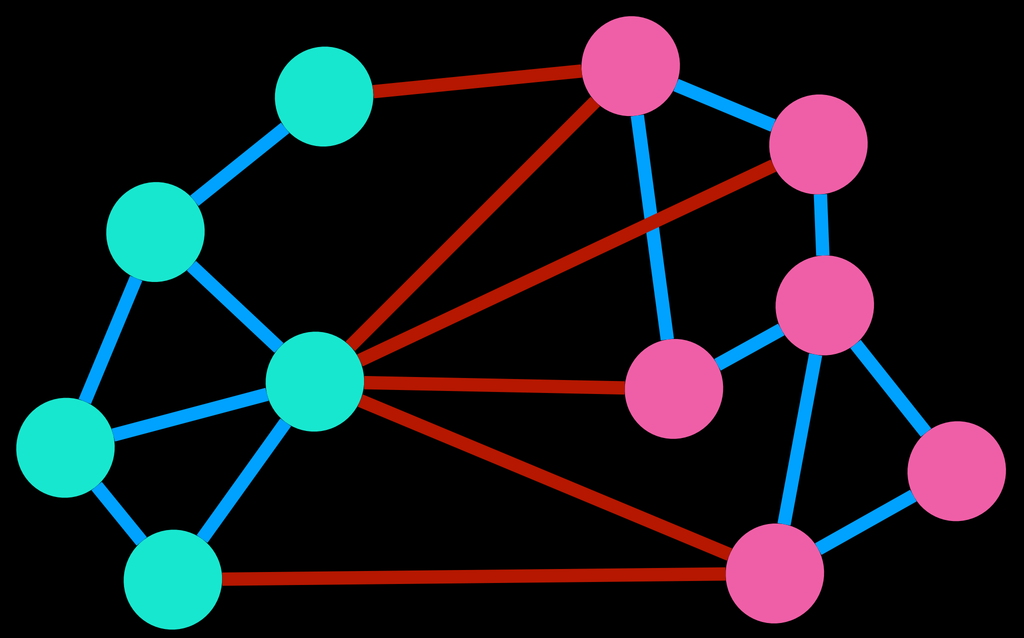
Main Technical Result



- Let $\mathscr{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ be the *signed normalized Laplacian*
- Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be the orthonormal row eigenvectors of $\mathscr{L}$, i.e., $\lambda_i \mathbf{v}_i = \mathbf{v}_i \mathscr{L}$, and assume that $0 \leq \lambda_1 \leq \cdots \leq \lambda_n$

➡ *The vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ essentially reveal the polarized communities*

Main Technical Result



- Let $\mathscr{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ be the *signed normalized Laplacian*
- Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be the orthonormal row eigenvectors of $\mathscr{L}$, i.e., $\lambda_i \mathbf{v}_i = \mathbf{v}_i \mathscr{L}$, and assume that $0 \leq \lambda_1 \leq \cdots \leq \lambda_n$

➡ *The vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ essentially reveal the polarized communities*

- More formally, consider a polarized cluster $U = (V_1, V_2)$.
  Then (if G is degree-bounded, clusterable, and still simplifying a lot) for all $i \leq 1, \ldots, k$,

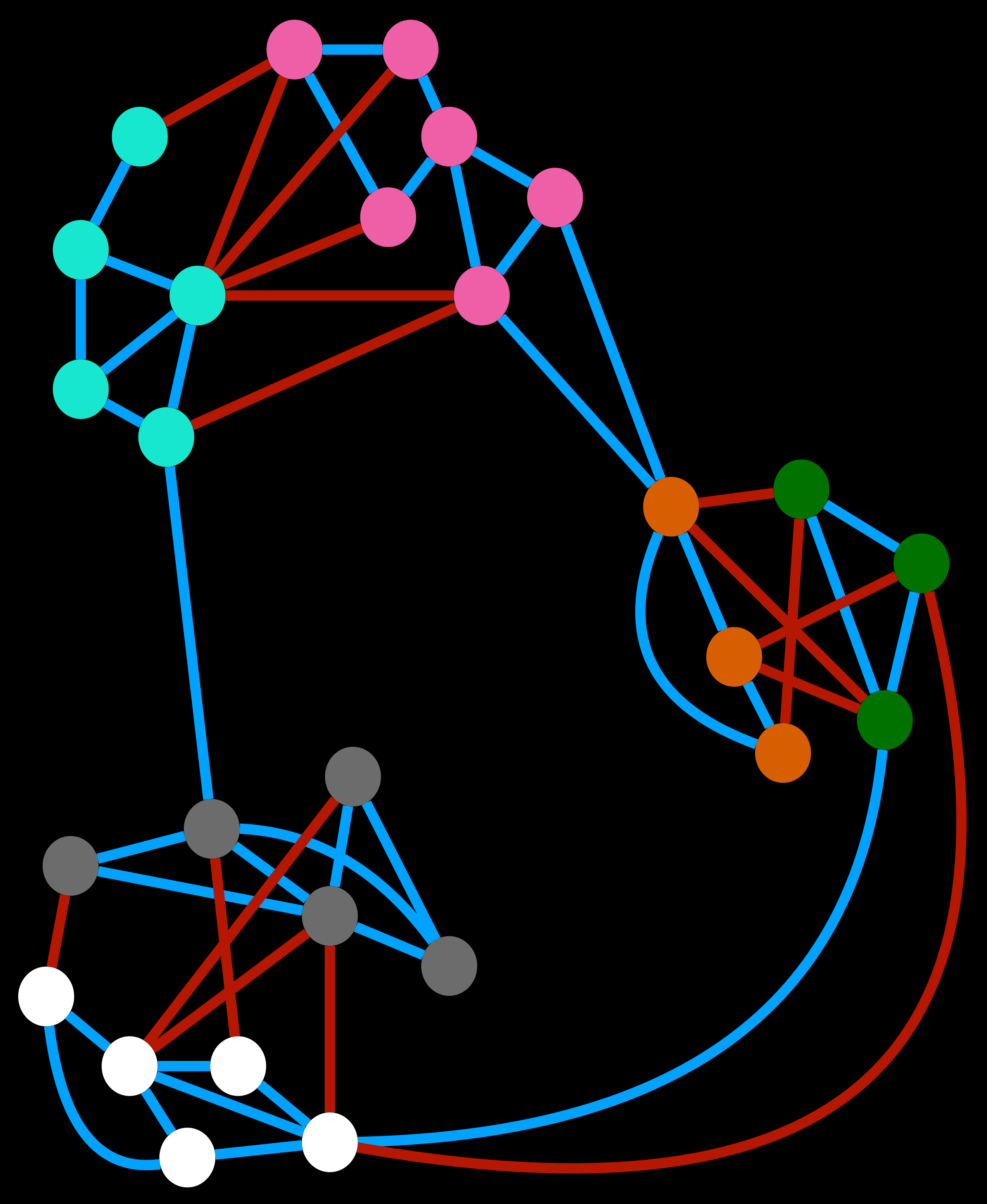
$$\mathbf{v}_i(u) \approx +\frac{1}{|U|} \text{ if } u \in V_1 \quad \text{and} \quad \mathbf{v}_i(u) \approx -\frac{1}{|U|} \text{ if } u \in V_2$$

Sublinear-Time Clustering Oracle for Signed Graphs

***Stefan Neumann* (KTH) and Pan Peng (USTC)**
@StefanResearch

- We provide a sublinear-time clustering oracle that returns the communities of vertices in signed graphs
- We prove that it works and give new insights into spectral graph theory for signed random walks
- Highly scalable and works well in practice for large clusters
- We provide a new signed graph dataset with large ground-truth communities

- Open questions:
    - Give guarantees for our heuristic
    - Use our theoretical insights for better practical algorithms
    - What else can we do with signed random walks?



Sublinear-Time Clustering Oracle for Signed Graphs

***Stefan Neumann* (KTH) and Pan Peng (USTC)**
@StefanResearch

Thank you!

- We provide a sublinear-time clustering oracle that returns the communities of vertices in signed graphs
- We prove that it works and give new insights into spectral graph theory for signed random walks
- Highly scalable and works well in practice for large clusters
- We provide a new signed graph dataset with large ground-truth communities

- Open questions:
    - Give guarantees for our heuristic
    - Use our theoretical insights for better practical algorithms
    - What else can we do with signed random walks?