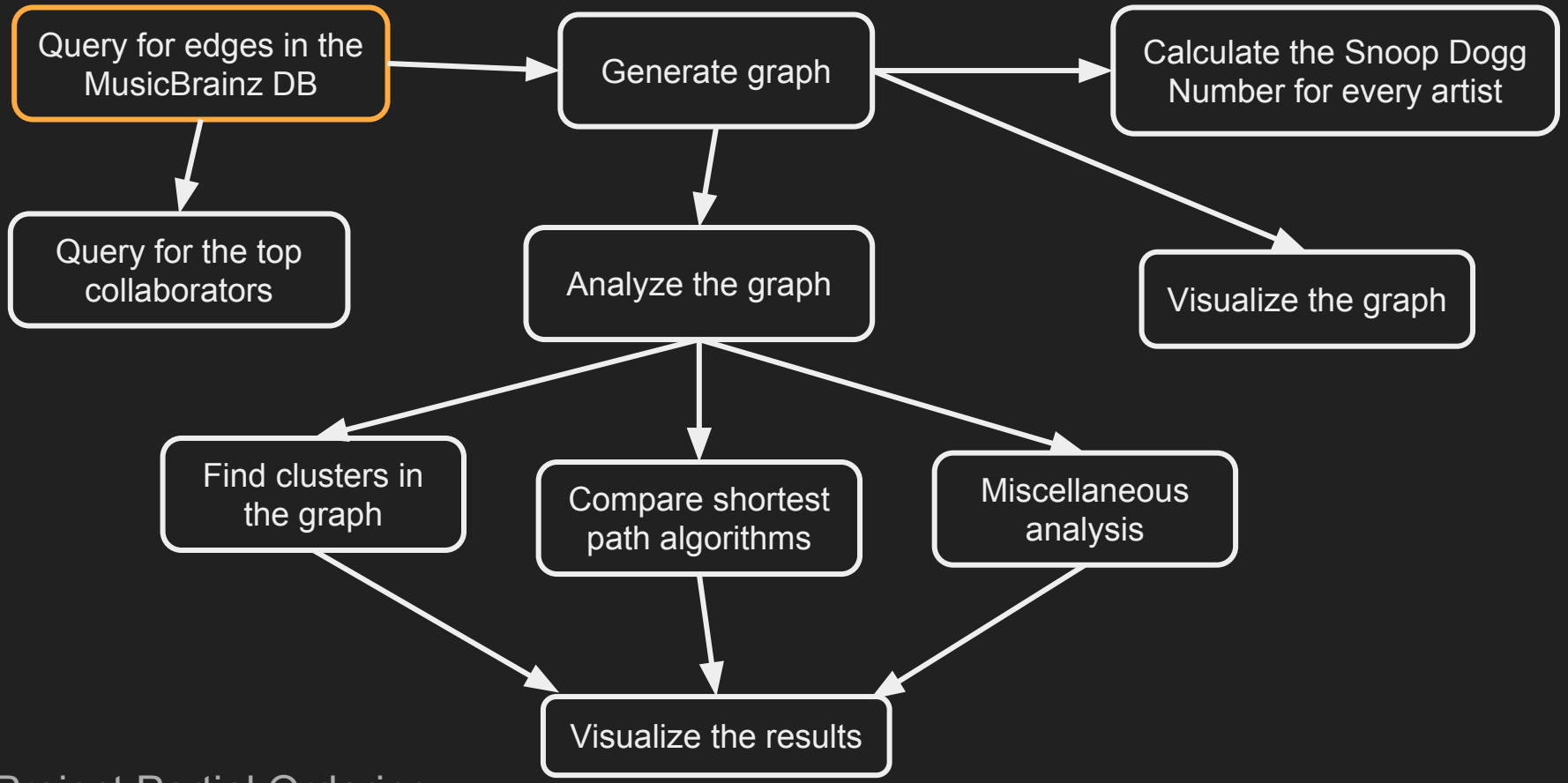
Music Collaboration Graphs

B.R.

Recap



Project Partial Ordering

Kanye West feat. RZA, Jay-Z, Pusha T, Swizz Beatz & CyHi da Prynce

Artist Credit

Kanye West
RZA
Jay-Z
Pusha T
Swizz Beatz
CyHi da Prynce

Vertices

{Kanye West, RZA}
{Kanye West, Jay-Z}
{Kanye West, Pusha T}
{Kanye West, Swizz Beatz}
{Kanye West, CyHi da Prynce}
{RZA, Jay-Z}
{RZA, Pusha T}
{RZA, Swizz Beatz}
{RZA, CyHi da Prynce}
{Jay-Z, Pusha T}
{Jay-Z, Swizz Beatz}
{Jay-Z, CyHi da Prynce}
{Pusha T, Swizz Beatz}
{Pusha T, CyHi da Prynce}
{Swizz Beatz, CyHi da Prynce}

Edges

Finding Edges via Artist Credit

acoustic guitar and guitar:	André Olbrich
bass guitar:	Hansi Kürsch
drums and percussion:	Thomen Stauch
guitar:	Marcus Siepen
additional mixer:	Henrik Vindeby
composer:	Hansi Kürsch André Olbrich
design/illustration:	Andreas Marschall
engineer:	Flemming Rasmussen Piet Sielck
lead vocals:	Hansi Kürsch
lyricist:	Hansi Kürsch
mixer:	Flemming Rasmussen
photography:	Ulf Thürmann
producer:	Flemming Rasmussen
remastered versions:	Imaginations From the Other Side
ASIN:	US: B00000IP3S [info]
Discogs:	https://www.discogs.com/release/1736518 [info]

Better Ways of Finding Edges

The Snoop Dogg Number
~270,000 vertices and ~650,000 edges (via artist credit)

Mean	Median	Mode	Max	Stddev	Variance
4.16	5	4	17	1.28	1.63

The Erdős Number
~401,000 vertices and ??? edges

Mean	Median	Mode	Max	Stddev	Variance
4.69	5	?	15	1.27	?

Erdős Number vs. Snoop Dogg Number

Comparison of Statistics

Shortest Paths in Unweighted Graphs

15.2 hours†

~300 shortest paths / minute

SDN Computation Time - A* Algorithm

~270,000 vertices and ~650,000 edges

† Multi-threaded CPython script running on an i7-6700k

```python
artists = a collection of artists
for artist in artists:
    if(pathExists)
        artist.sdn = astar_path(graph, 'Snoop Dogg', artist.name)
    else:
        artist.sdn = +inifinity
```

Finding Shortest Paths - A* Algorithm

Runtime → $O(|V| \times |E|)$

Worst case → $O(n^3)$

Dijkstra's Algorithm `NetworkX.dijkstra_path`

Total	Mean	Median	Stddev	Variance
1100	1.1	1.3	0.80	0.64

A* Algorithm `NetworkX.astar_path`

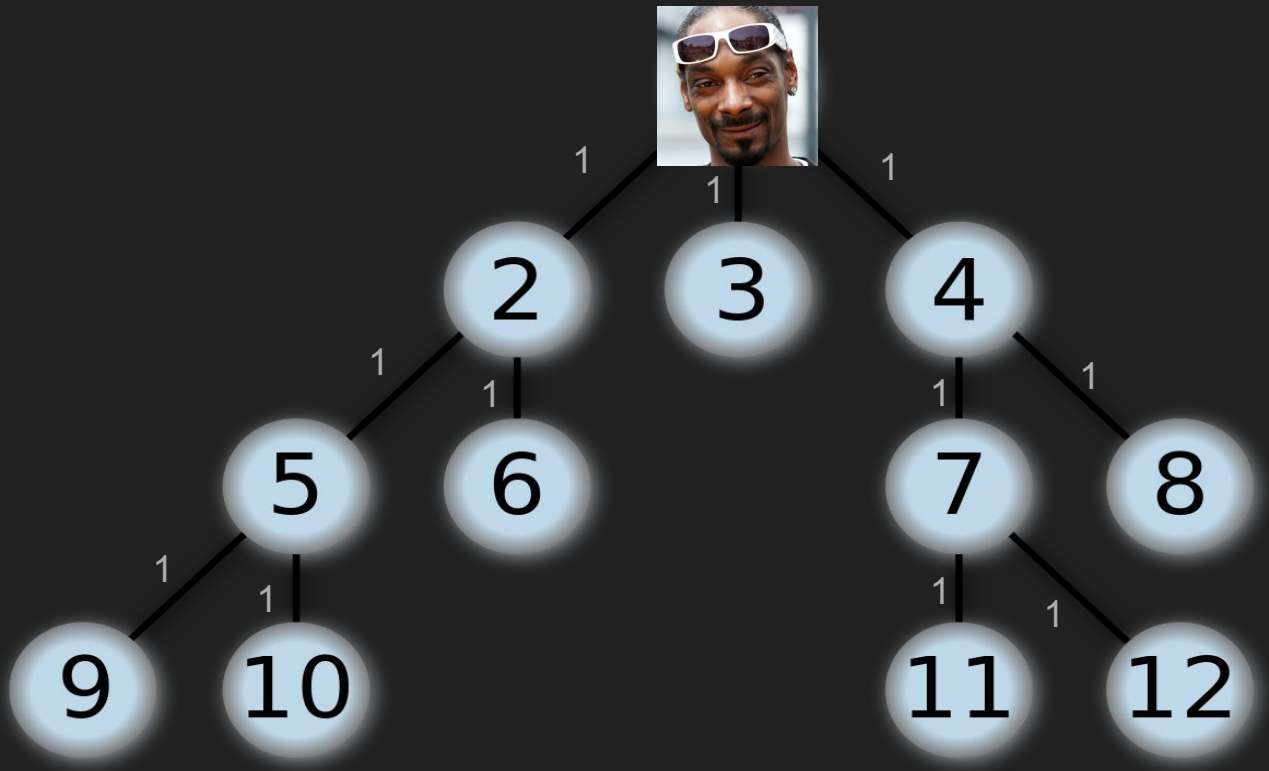
Total	Mean	Median	Stddev	Variance
540	0.54	0.60	0.41	0.17

Comparison of Shortest Path Algorithms

Finding 10,000 paths in a graph of ~270,000 vertices and ~650,000 edges
Implementations in NetworkX 1.10

All measurements in seconds.

Benchmarks performed on an Intel i7-6700K.



Using Breadth-first Search to Compute SDN

```python
start = graph.node("Snoop Dogg")
start.sdn  = 0
start.path = ["Snoop Dogg"]

for edge in edges_in_bfs_order_from( start ):
    source = edge.source
    target = edge.target

    target.sdn  = source.sdn  + 1
    target.path = source.path + target
```

Finding Shortest Paths -- Breadth-first Search

Runtime → $O(|E|)$

Worst case → $O(n^2)$

Dijkstra's Algorithm `NetworkX.dijkstra_path`

Total	Mean	Median	Stddev	Variance
1100	1.1	1.3	0.80	0.64

A* Algorithm `NetworkX.astar_path`

Total	Mean	Median	Stddev	Variance
540	0.54	0.60	0.41	0.17

Bidirectional Search Algorithm `NetworkX.shortest_path`

Total	Mean	Median	Stddev	Variance
0.30	2.9e-04	7.3e-05	6.4e-04	4.1e-07

Comparison of Shortest Path Algorithms

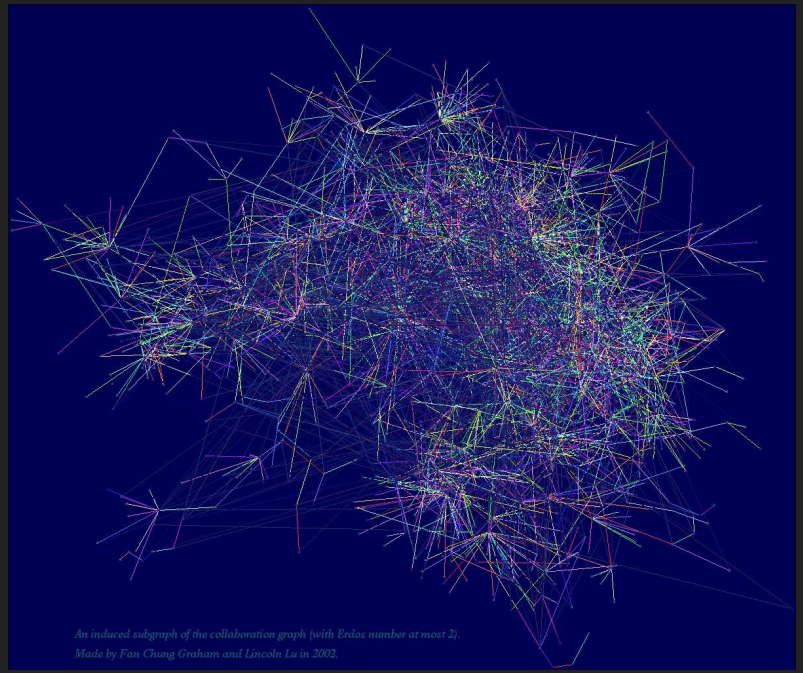
Finding 1,000 paths in an unweighted graph of ~270,000 vertices and ~650,000 edges
Implementations in NetworkX 1.10

All measurements in seconds.
Benchmarks performed on an Intel i7-6700K.

```
BIDIRECTIONAL_SEARCH
Q_I.Insert(x_I) and mark x_I as visited
Q_G.Insert(x_G) and mark x_G as visited
while Q_I not empty and Q_G not empty do
    if Q_I not empty
        x ← Q_I.GetFirst()
        if x = x_G or x ∈ Q_G
            return SUCCESS
        forall u ∈ U(x)
            x' ← f(x, u)
            if x' not visited
                Mark x' as visited
                Q_I.Insert(x')
            else
                Resolve duplicate x'
    if Q_G not empty
        x' ← Q_G.GetFirst()
        if x' = x_I or x' ∈ Q_I
            return SUCCESS
        forall u^-1 ∈ U^-1(x')
            x ← f^-1(x', u^-1)
            if x not visited
                Mark x as visited
                Q_G.Insert(x)
            else
                Resolve duplicate x
return FAILURE
```

Figure 2.7: A general template for bidirectional search.

Visualizing the Graph



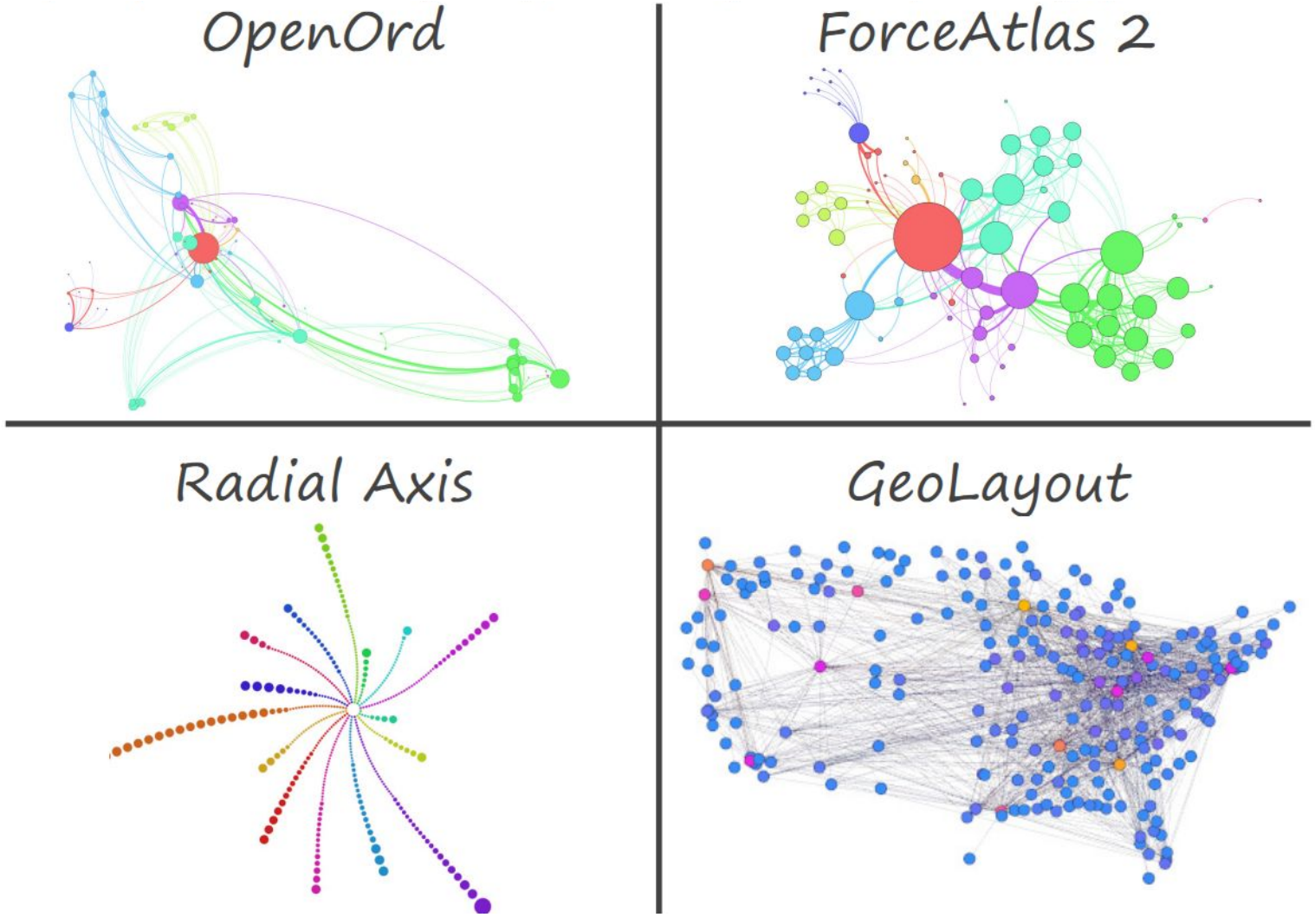
Visualization of Erdős Collaboration Graph

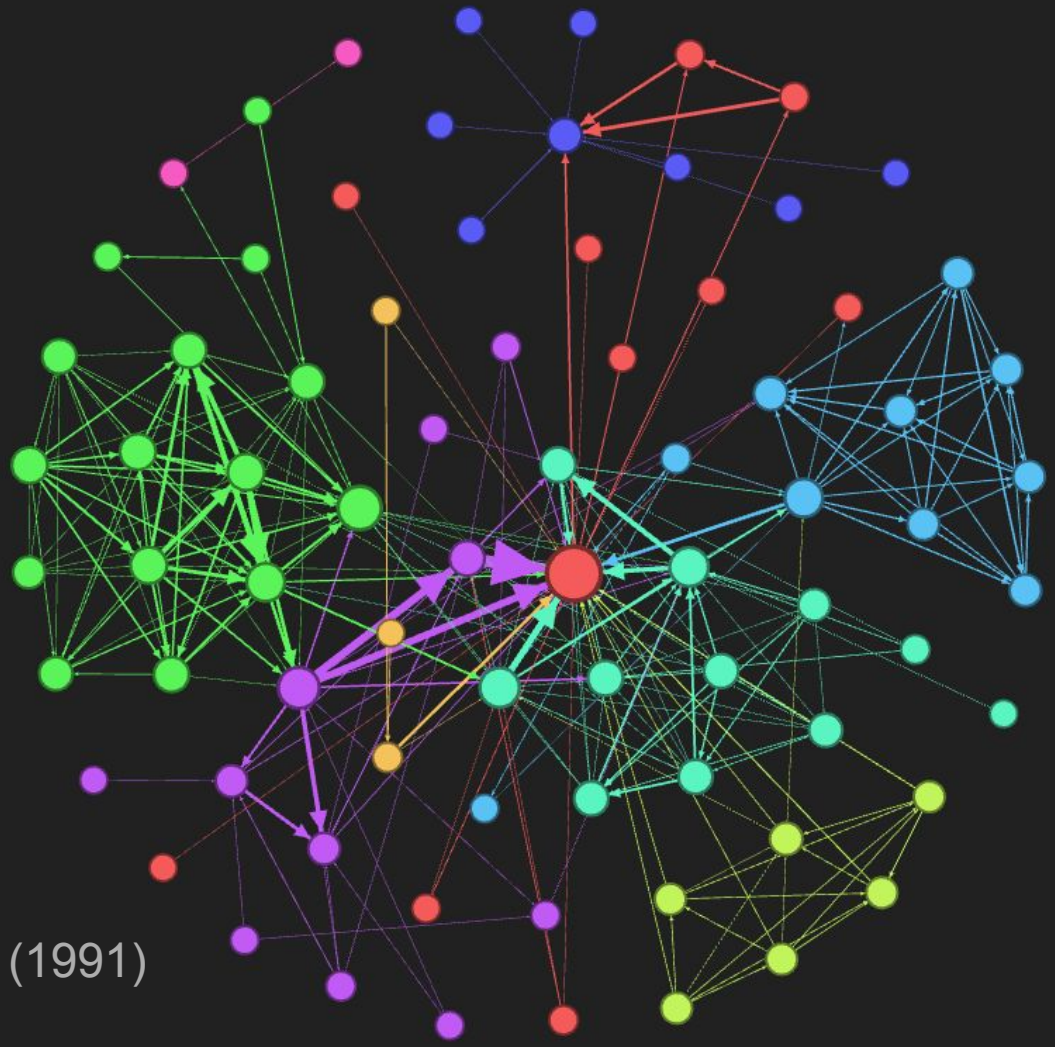
We Can Do Better Than This!

Gephi

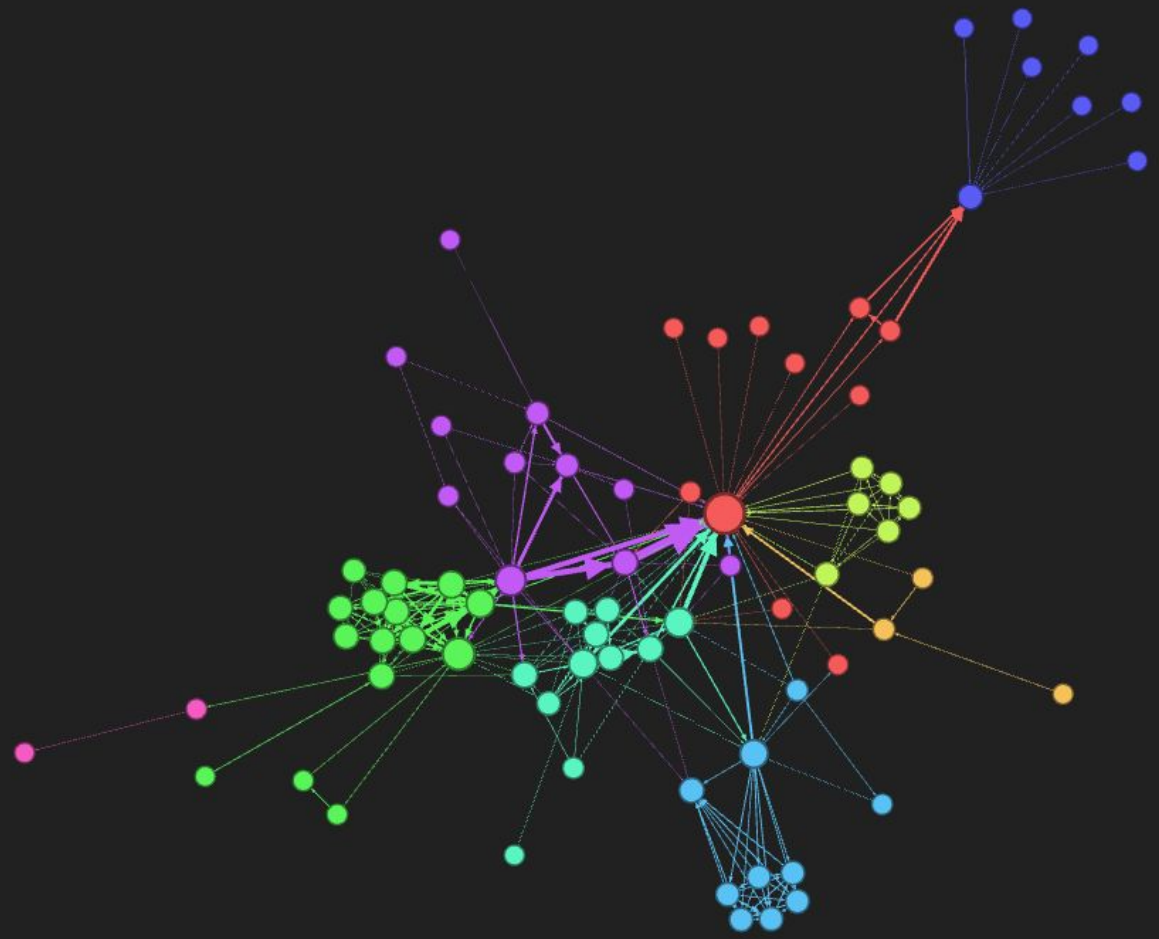
Graph Layout Algorithms

You don’t get a nice graph layout for free.

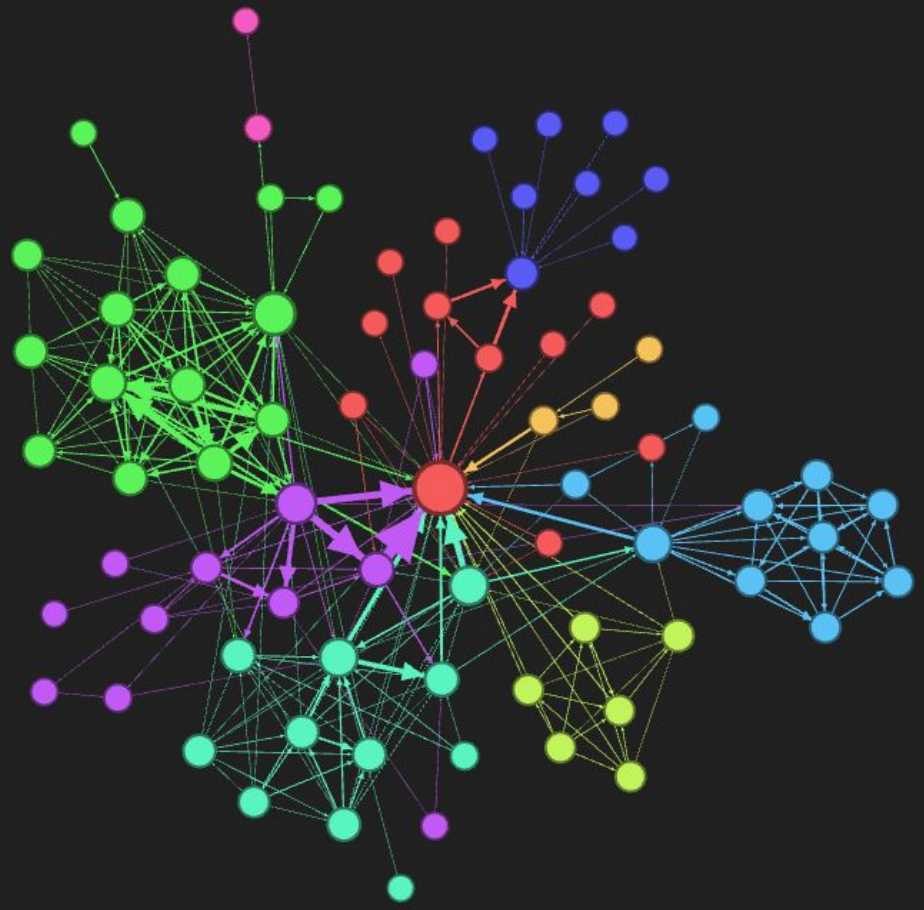

OpenOrd
ForceAtlas 2
Radial Axis
GeoLayout



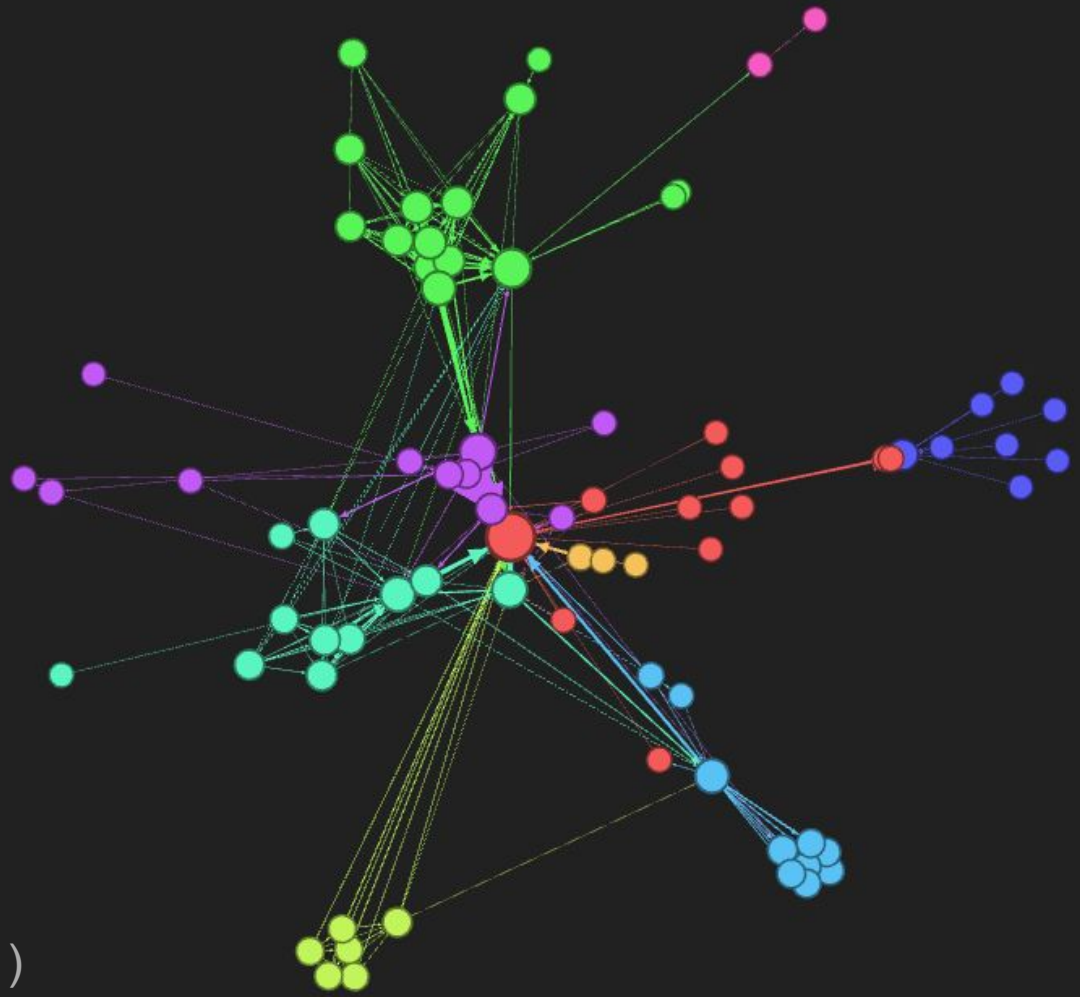
Fruchterman-Reingold (1991)



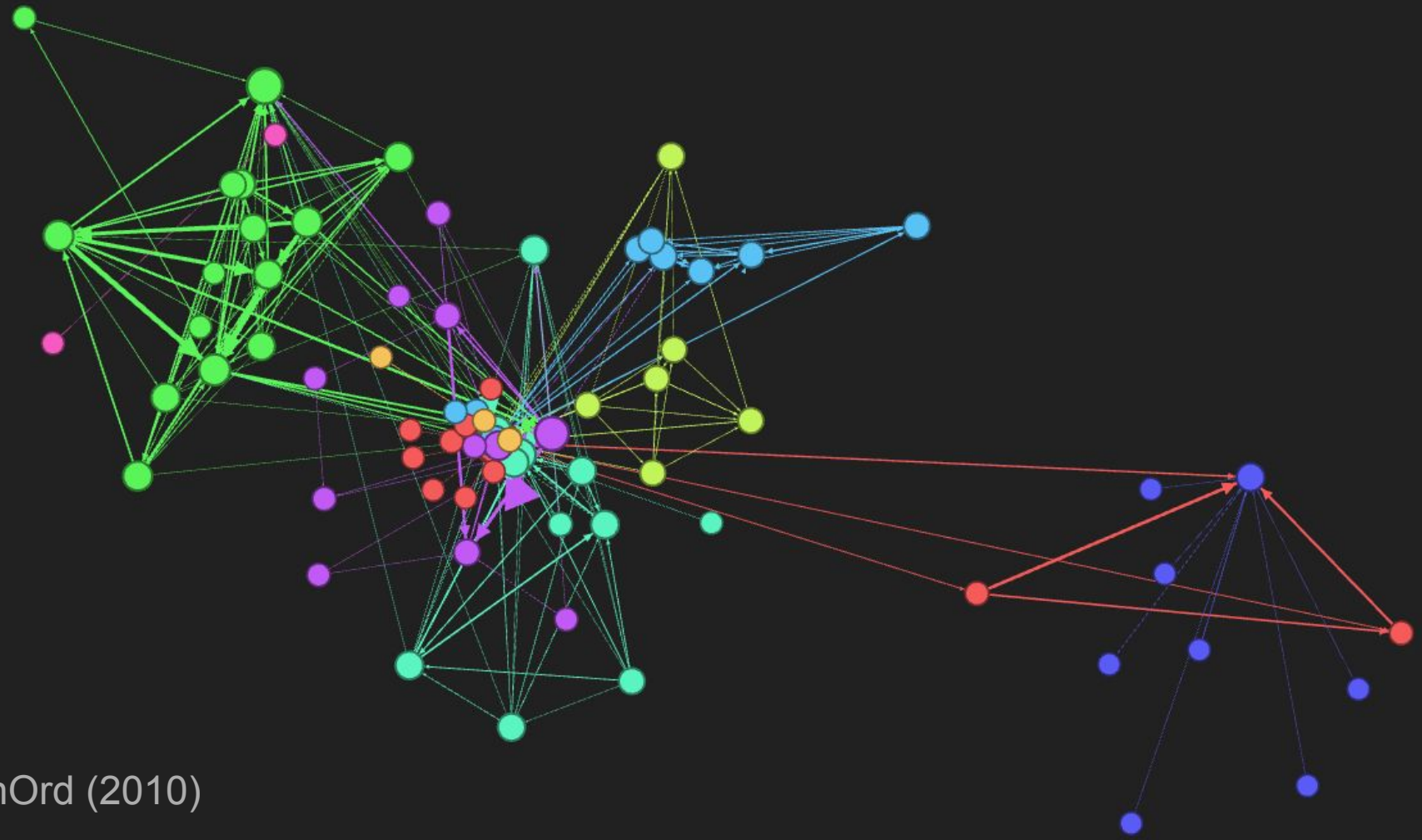
Yifan Hu (2005)



Force Atlas (2007)



Force Atlas 2 (2011)



OpenOrd (2010)

Algorithm	Date	Kind	Emphasis	Complexity	Graph Size	Edge Weight	Parallelizable	Fixed End
Fruchterman-Reingold	1991	Force-directed	Complementarities	$O(N^2)$	< 1,000			
Force Atlas	2007	Force-directed	Complementarities	$O(N^2)$	< 10,000	✓		
Force Atlas 2	2011	Force-directed	Complementarities	$O(N \log N)$	< 1,000,000	✓	✓	
Yifan Hu	2005	Force-directed	Complementarities	$O(N \log N)$	< 100,000			✓
OpenOrd	2010	Force-directed	Clusters	$O(N \log N)$	< 1,000,000	✓	✓	✓
Circular	2010	Circular	Ranking	$O(N)$	< 1,000,000			✓
Radial Axis	2011	Circular	Ranking	$O(N)$	< 1,000,000			✓
GeoLayout	2010	Geographic	Geolocation	$O(N)$	< 1,000,000			✓

Comparison of Graph Layout Algorithms

Implemented in Gephi 0.9.1

```
area  := W * L; { W and L are the width and length of the frame }
G := (V, E); { the vertices are assigned random initial positions }
k  := √(area/|V|);
function fa(z) := begin return x²/k end;
function fr(z) := begin return k²/z end;

for i := 1 to iterations do begin
    { calculate repulsive forces}
    for v in V do begin
        { each vertex has two vectors: .pos and .disp }
        v.disp := 0;
        for u in V do
            if (u # v) then begin
                {    is short hand for the difference}
                { vector between the positions of the two vertices )
                  := v.pos  - u.pos;
                v.disp := v.disp + (    /|    |) * fr (|    |)
            end
    end

    { calculate attractive forces }
    for e in E do begin
        { each edge is an ordered pair of vertices .v and .u }
          := e.v.pos  −  e.u.pos
        e.v.disp  := e.v.disp − (   /|    |) * fa(|    |);
        e.u. disp  := e.u.disp + (    /|    |) * fa(|    |)
    end

    { limit the maximum displacement to the temperature t }
    { and then prevent from being displaced outside frame}
    for v in V do begin
        v.pos := v.pos + ( v. disp/ |v.disp|) * min ( v.disp, t );
        v.pos.x  :=  min(W/2,  max(-W/2,  v.pos.x));
        v.pos.y  :=  min(L/2,  max(–L/2,  v.pos.y))
    end
    { reduce the temperature as the layout approaches a better configuration }
    t := cool(t)
end
```

Fruchterman-Reingold Layout Algorithm (1991)

Runtime → $O(n^2)$

Let’s apply these algorithms to visualize the graph.

We’re going to need weighted edges to get better results.

collaborator1	collaborator2
Nate Dogg	Snoop Dogg
Kurupt	Snoop Dogg
Dr. Dre	Snoop Dogg
Daz Dillinger	Snoop Dogg
Daz	Snoop Dogg
Snoop Dogg	Warren G
Kokane	Snoop Dogg
Snoop Dogg	Soopafly
Snoop Dogg	Tha Dogg Pound
Snoop Dogg	Wiz Khalifa
2Pac	Snoop Dogg
Goldie Loc	Snoop Dogg
Snoop Dogg	The Game
RBX	Snoop Dogg

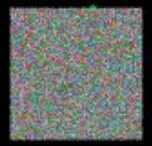
Unweighted Edges

collaborator1	collaborator2	num_collabs
Nate Dogg	Snoop Dogg	80
Kurupt	Snoop Dogg	63
Dr. Dre	Snoop Dogg	37
Daz Dillinger	Snoop Dogg	31
Daz	Snoop Dogg	31
Snoop Dogg	Warren G	27
Kokane	Snoop Dogg	25
Snoop Dogg	Soopafly	24
Snoop Dogg	Tha Dogg Pound	21
Snoop Dogg	Wiz Khalifa	20
2Pac	Snoop Dogg	20
Goldie Loc	Snoop Dogg	20
Snoop Dogg	The Game	19
RBX	Snoop Dogg	19

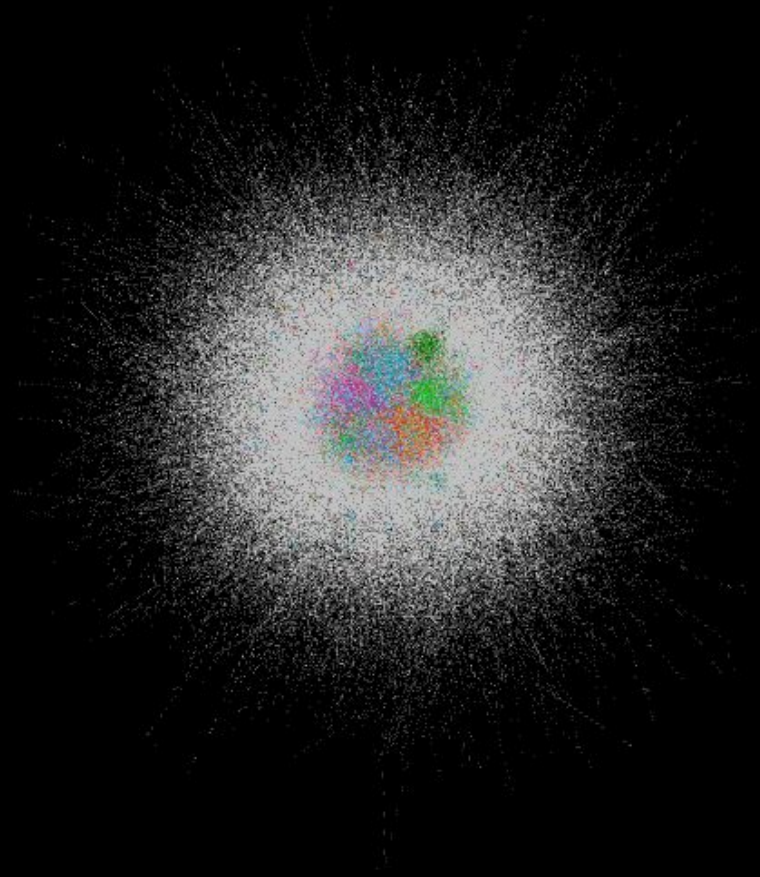
Weighted Edges

collaborator1	collaborator2	num_collabs
Berliner Philharmoniker	Herbert von Karajan	381
Academy of St Martin in the Fields	Sir Neville Marriner	214
Herbert von Karajan	Wiener Philharmoniker	212
Leonard Bernstein	New York Philharmonic	182
Eugene Ormandy	The Philadelphia Orchestra	158
Ferenc Fricsay	Radio-Symphonie-Orchester Berlin	148
London Symphony Chorus	London Symphony Orchestra	124
Academy of Ancient Music	Christopher Hogwood	120
Herbert von Karajan	Philharmonia Orchestra	118
London Symphony Orchestra	Sir Colin Davis	117
Ghostface Killah	Raekwon	116
Göteborgs Symfoniker	Neeme Järvi	111
Akademie für Alte Musik Berlin	René Jacobs	107
Lil Wayne	Rick Ross	105
Robert King	The King's Consort	104
Ferenc Fricsay	RIAS-Symphonie-Orchester Berlin	103
Berliner Philharmoniker	Claudio Abbado	101

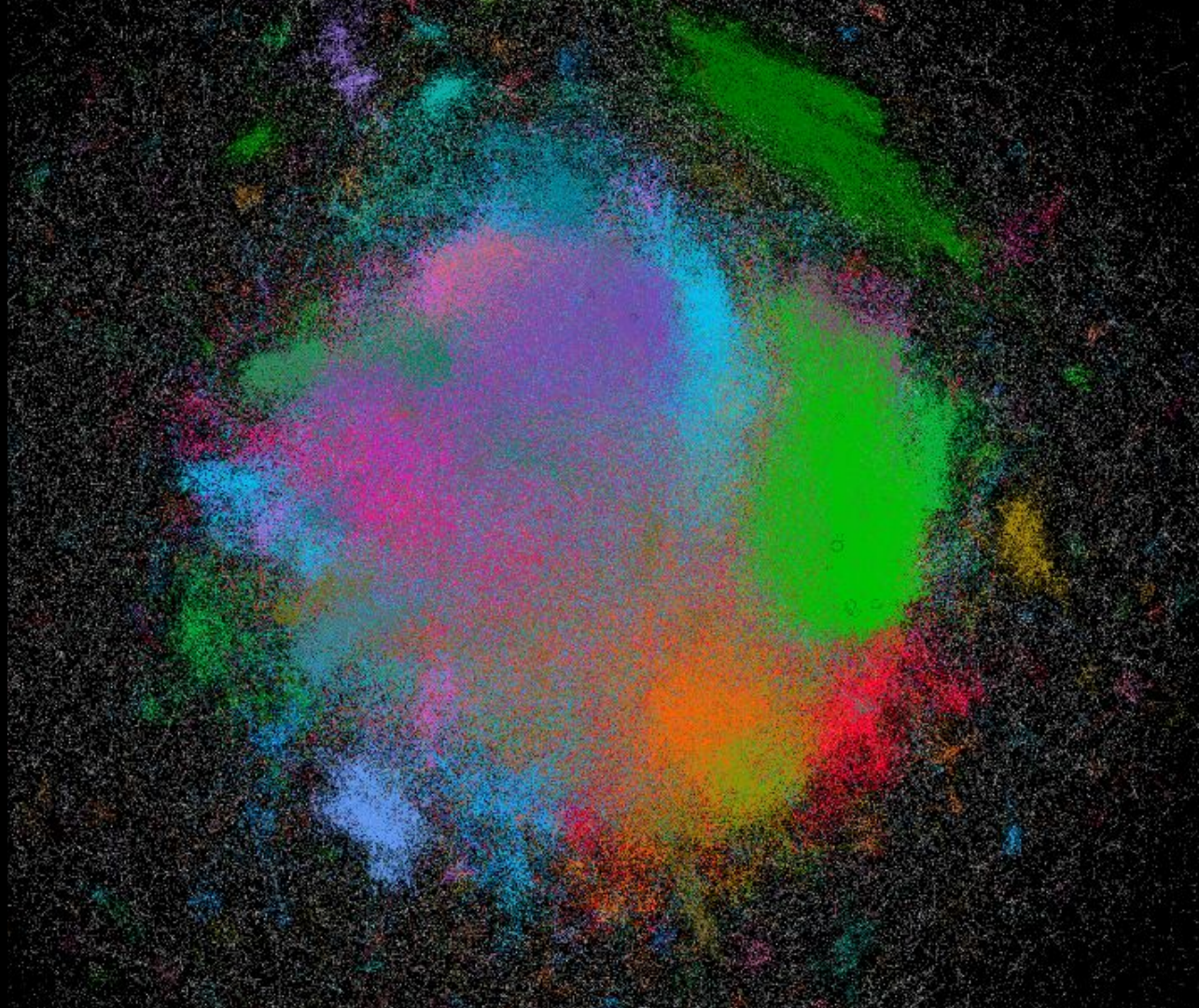
Most Significant Weighted Edges



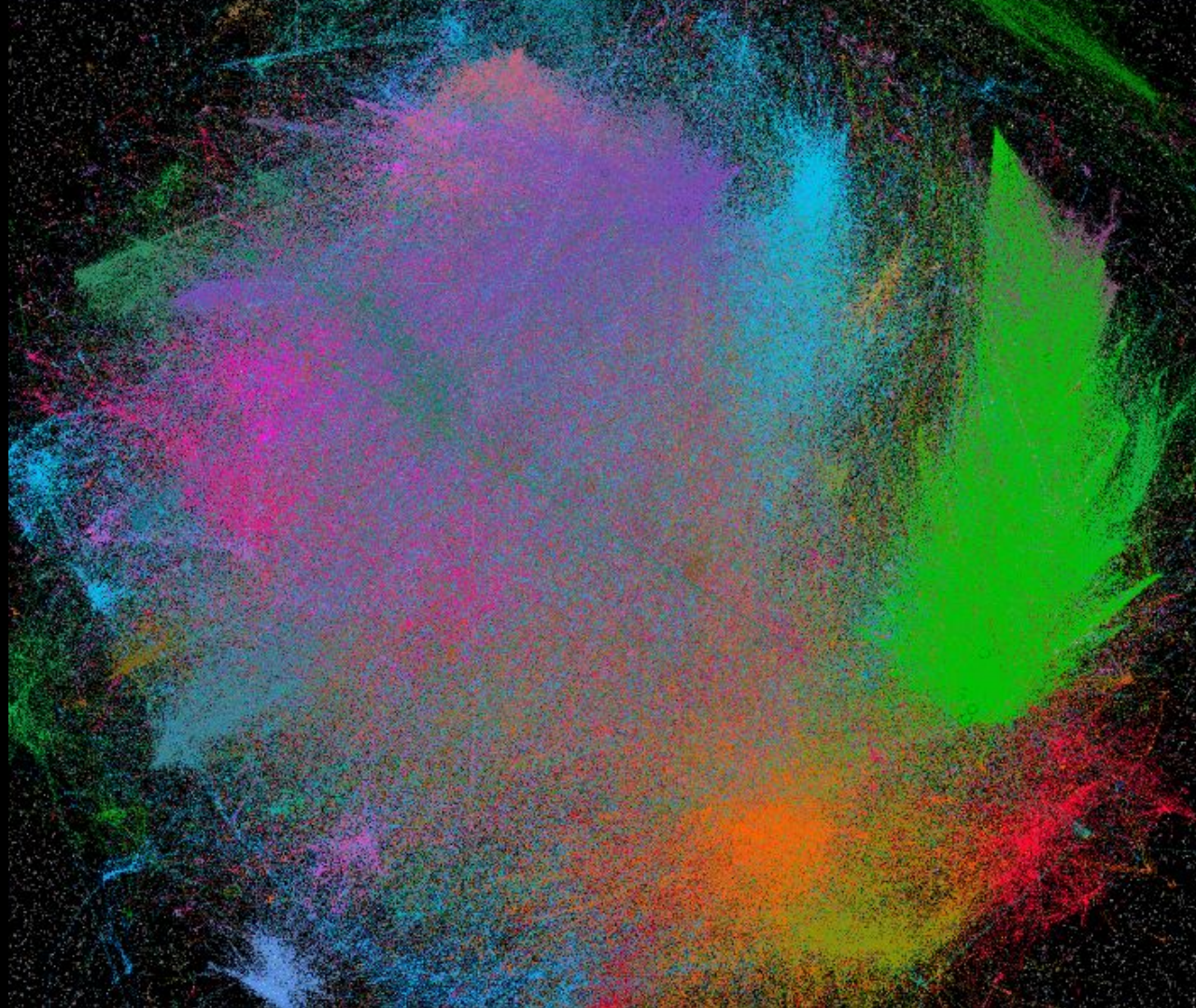
OpenOrd Layout Algorithm
Stage I - Liquid
Weighted Graph



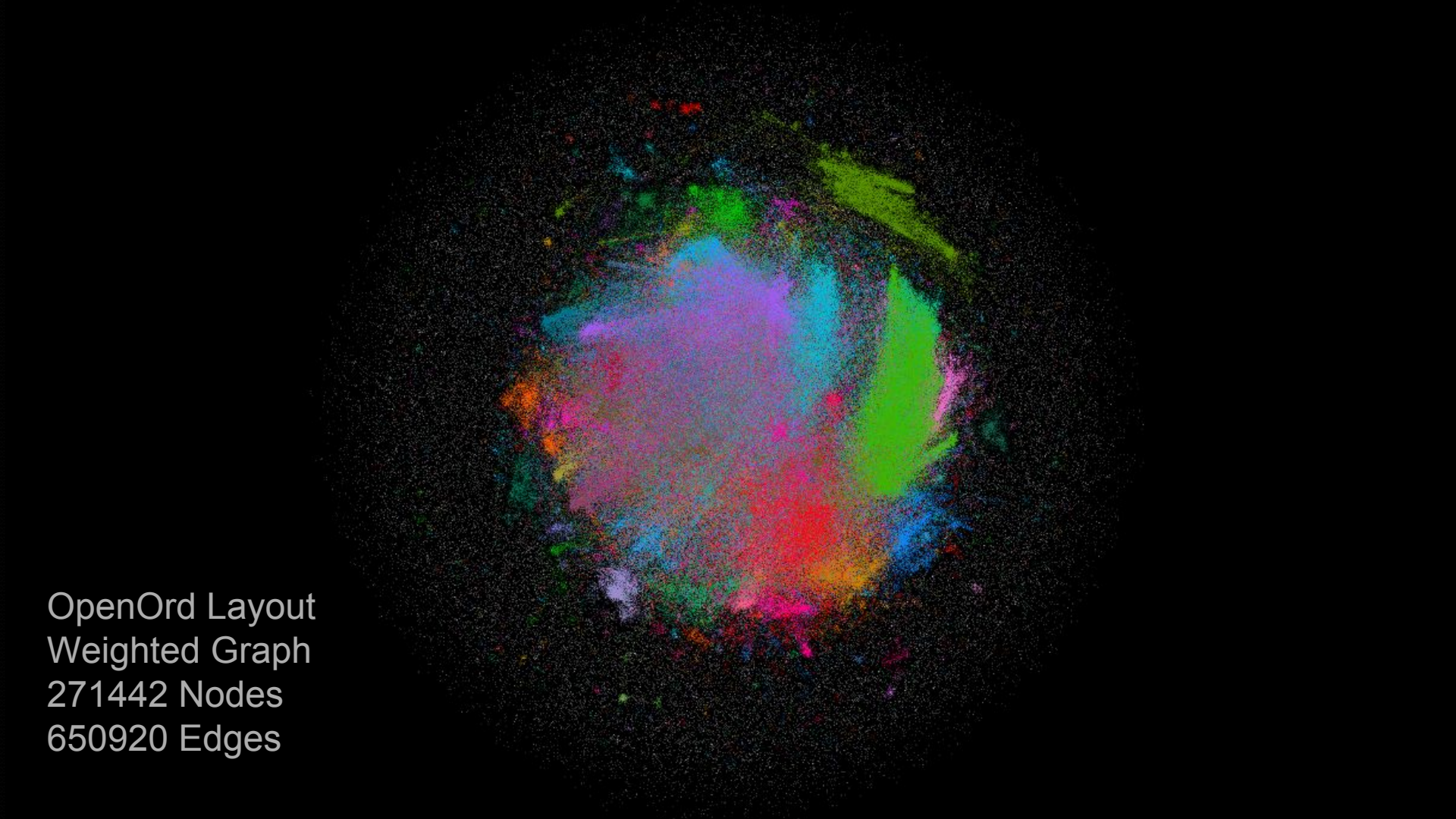
OpenOrd Layout Algorithm
Stage II - Expansion
Weighted Graph

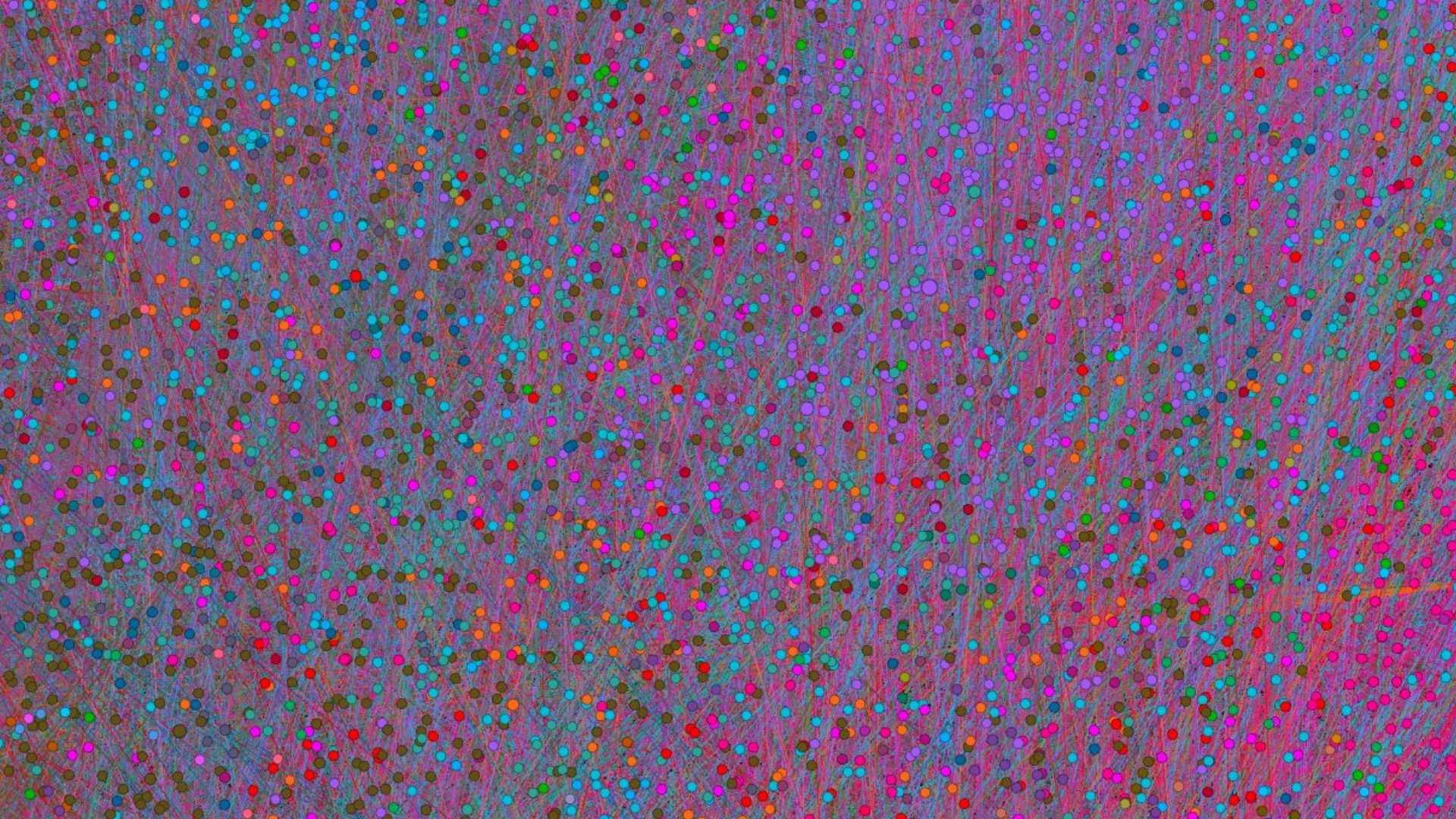


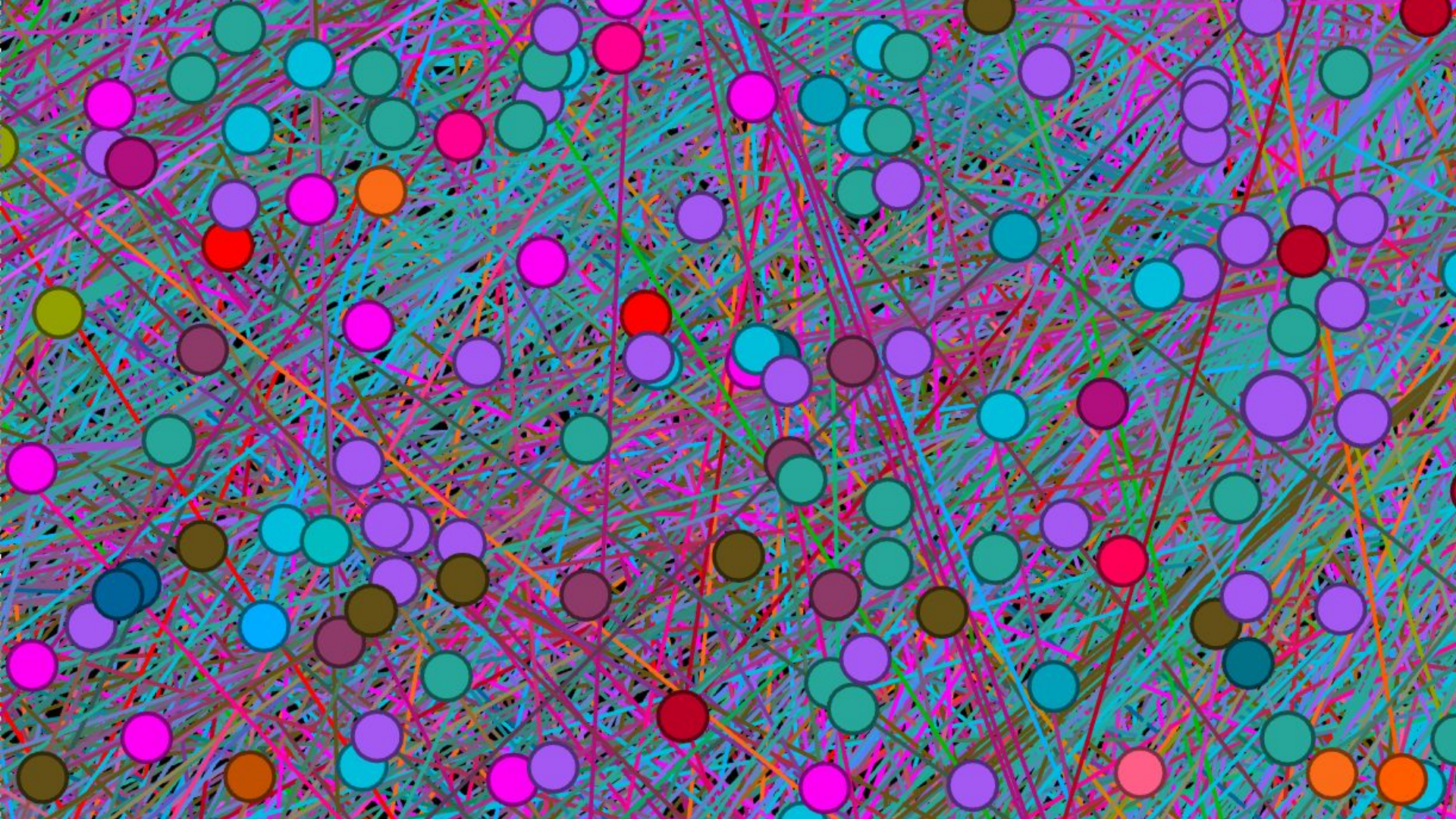
OpenOrd Layout Algorithm
Stage III - Cooldown
Weighted Graph



OpenOrd Layout Algorithm
Stage IV - Crunch
Weighted Graph


OpenOrd Layout
Weighted Graph
271442 Nodes
650920 Edges





How was the graph partitioned into these communities?

Fast unfolding of communities in large networks (2008)

Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre

1. Initialization: each node is a community.
2. For all nodes: change to a neighbour's community if it will increase Q.
3. Repeat #2 until we can't increase Q.
4. Merge all nodes belonging to the same community into one node.
5. End algorithm or do another pass (goto step #2).

Runtime → *"...the algorithm is extremely fast, i.e., computer simulations on large ad-hoc modular networks suggest that its complexity is* ***linear*** *on* ***typical and sparse*** *data"*

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

Formula to Compute Q (Modularity) of Entire Graph

$$\Delta Q = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

Formula to Compute Delta-Q of a Community Swap

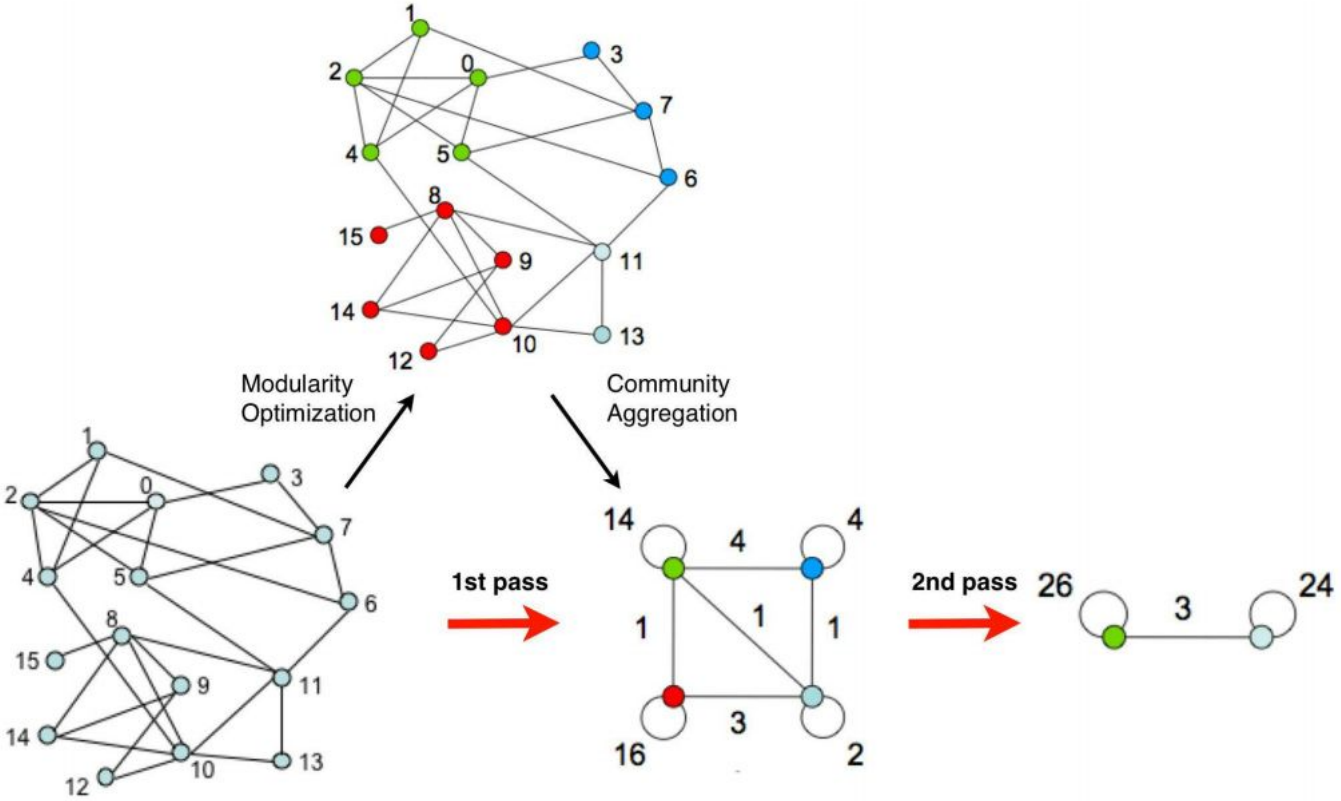
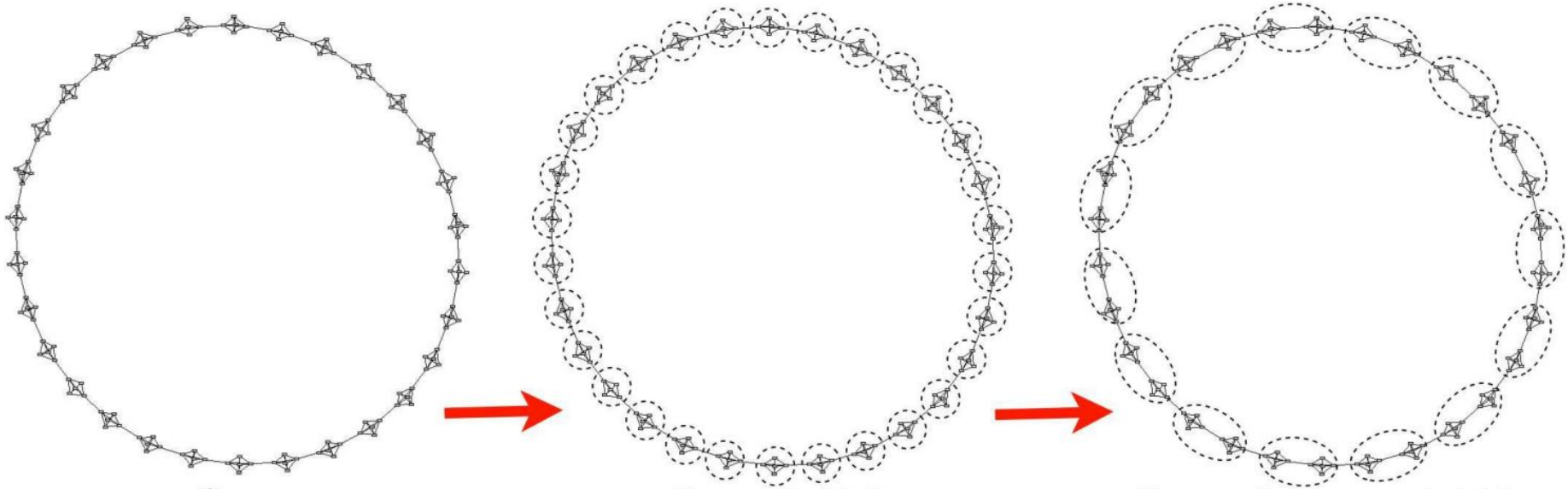
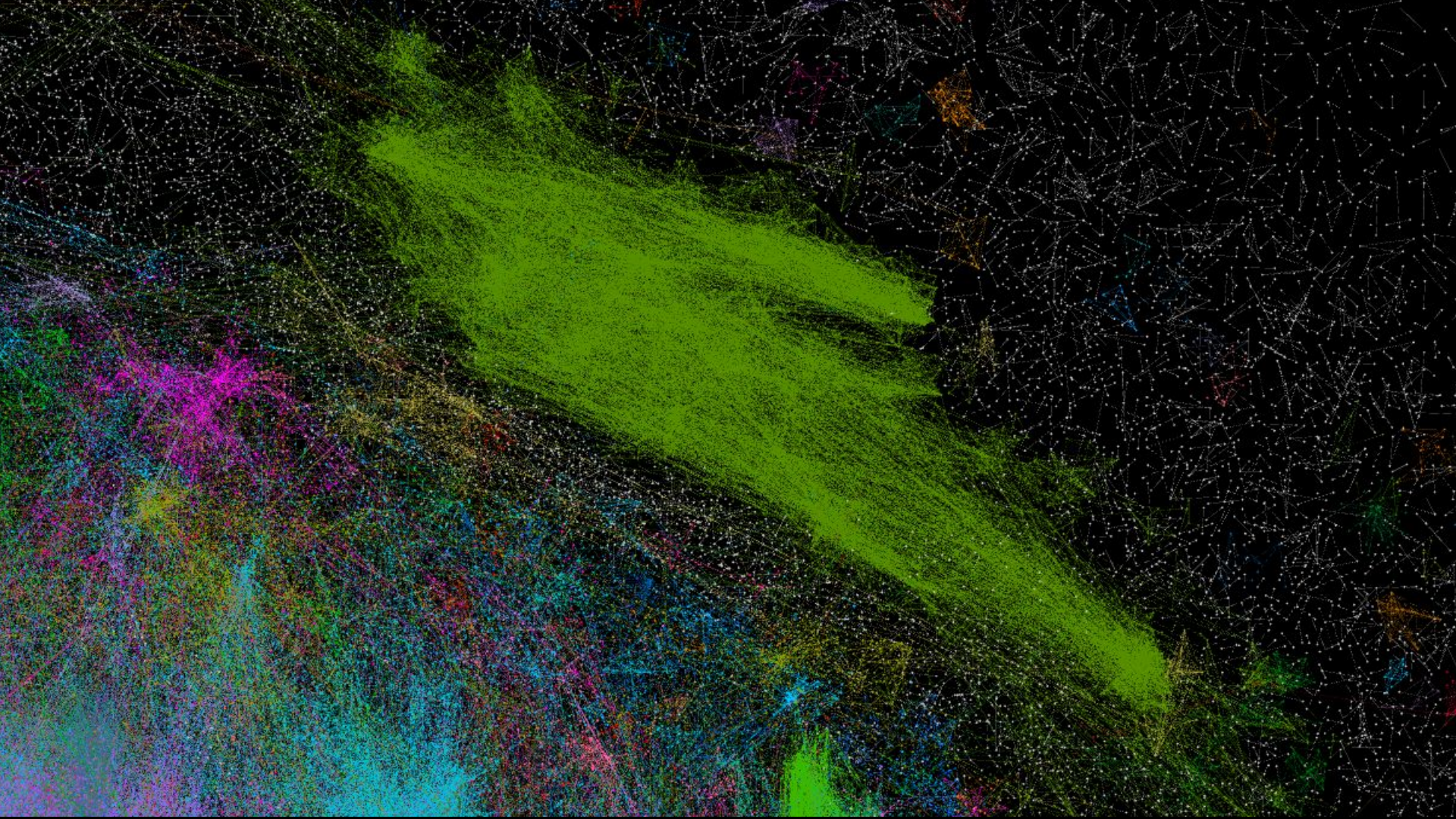
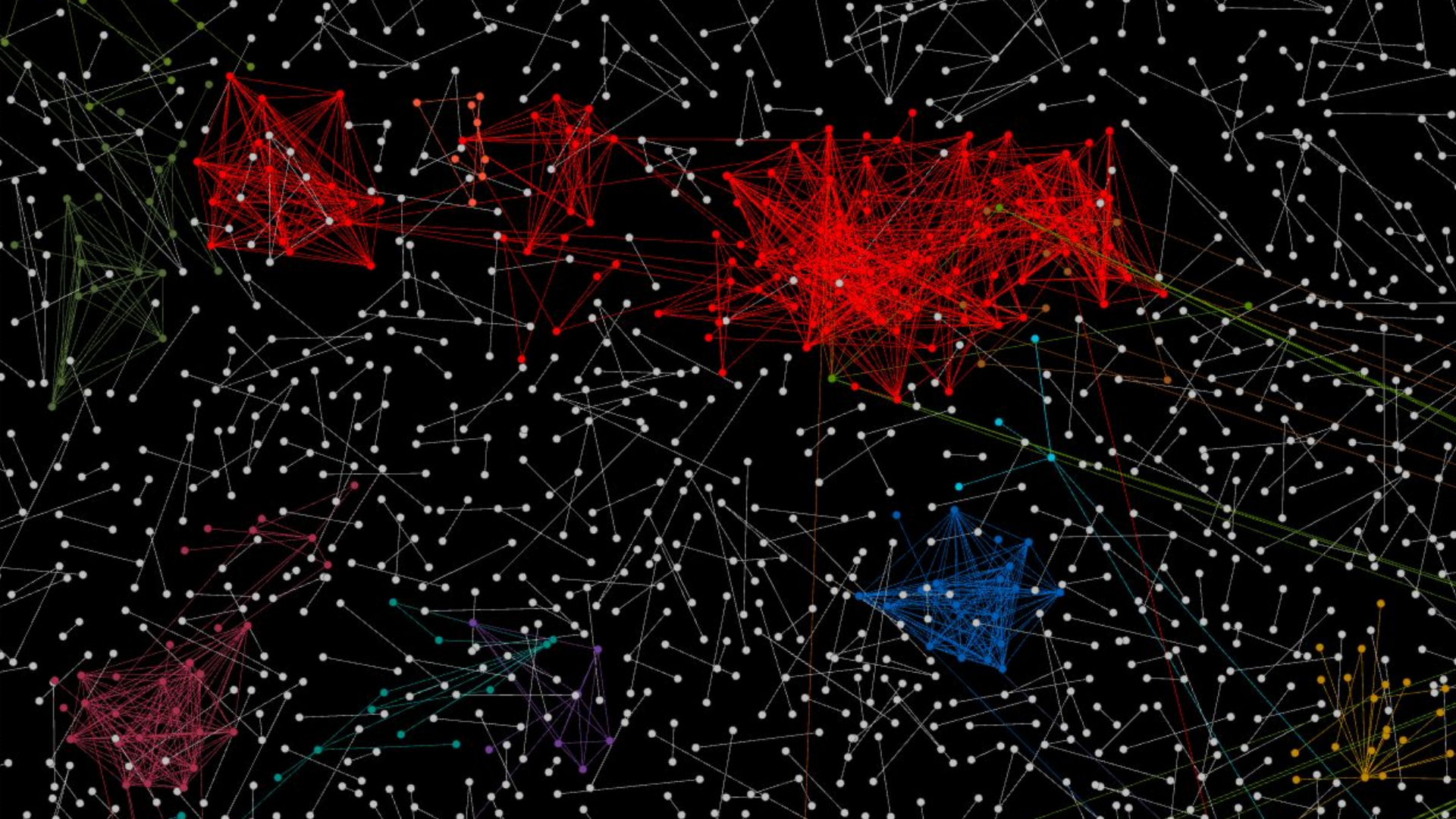


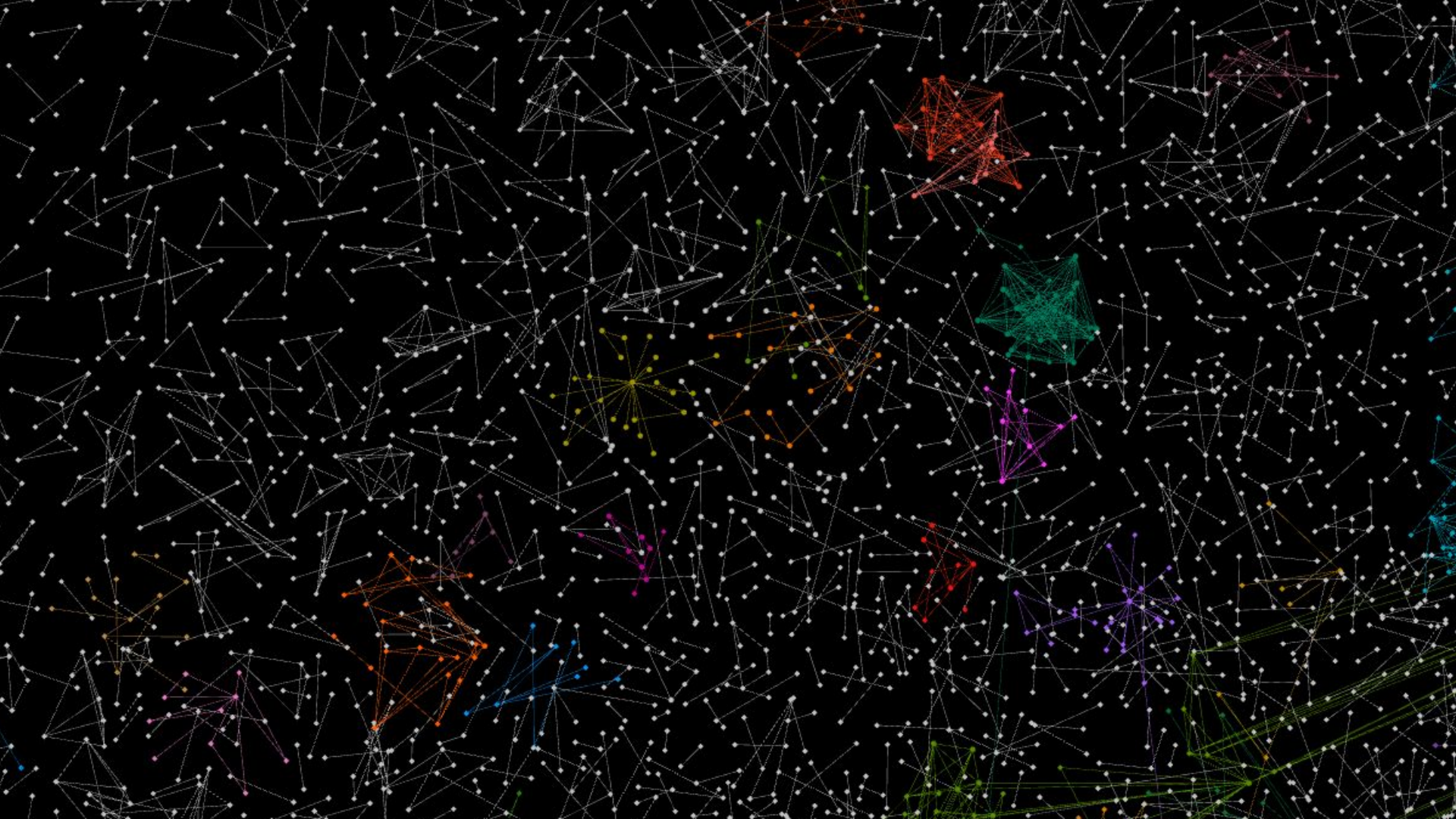
Figure 1. Visualization of the steps of our algorithm. Each pass is made of two phases: one where modularity is optimized by allowing only local changes of communities; one where the found communities are aggregated in order to build a new network of communities. The passes are repeated iteratively until no increase of modularity is possible.

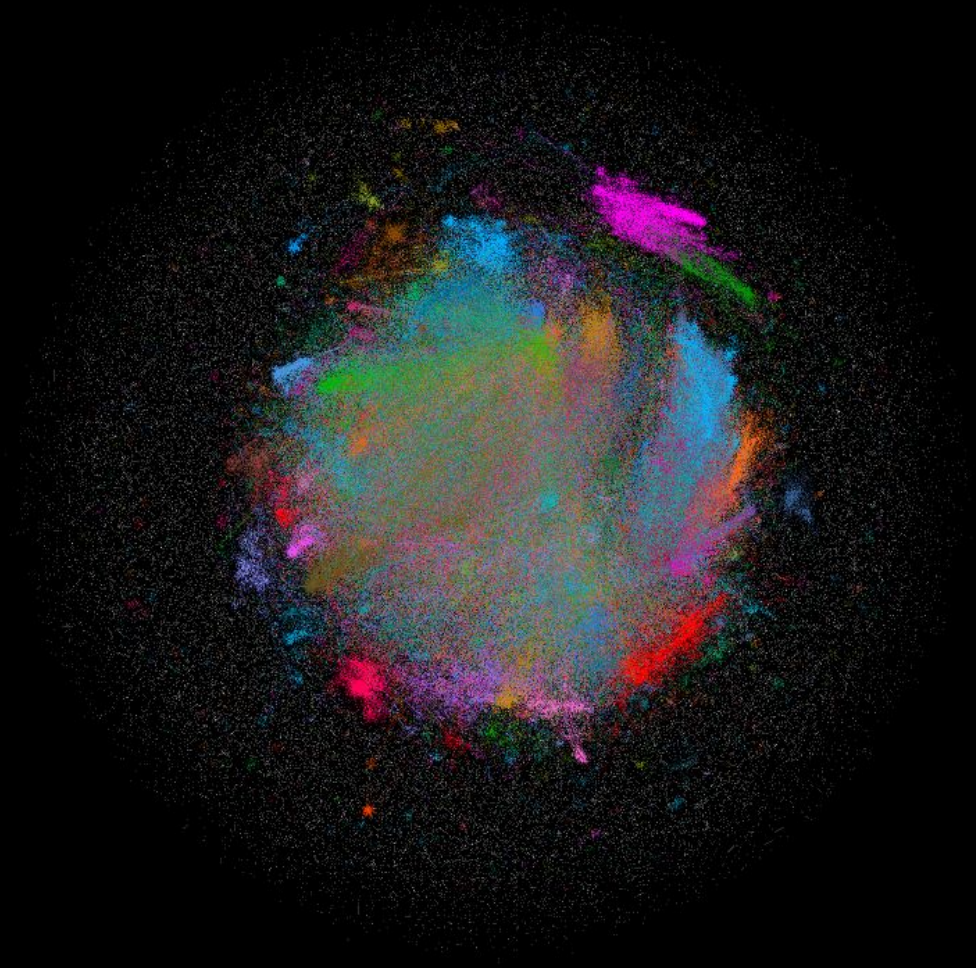


Q_{init} $Q_1 = 0.876$ $Q_2 = Q_{max} = 0.888$



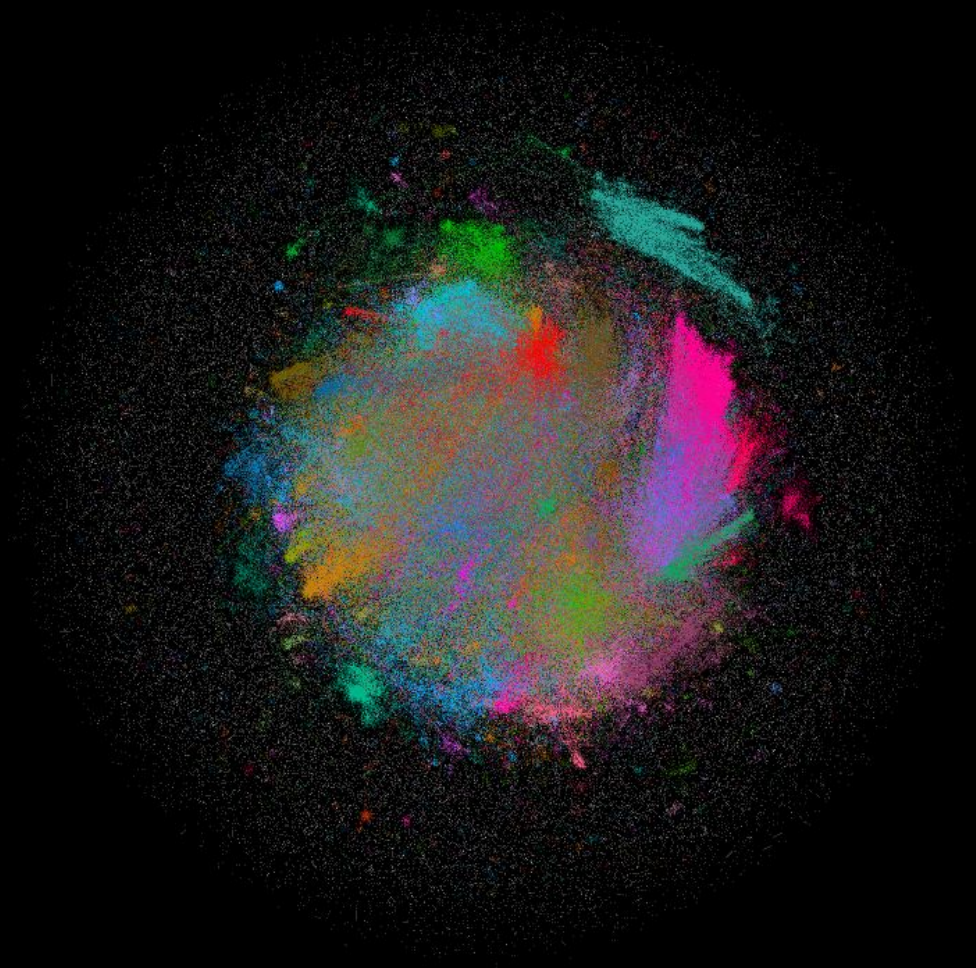






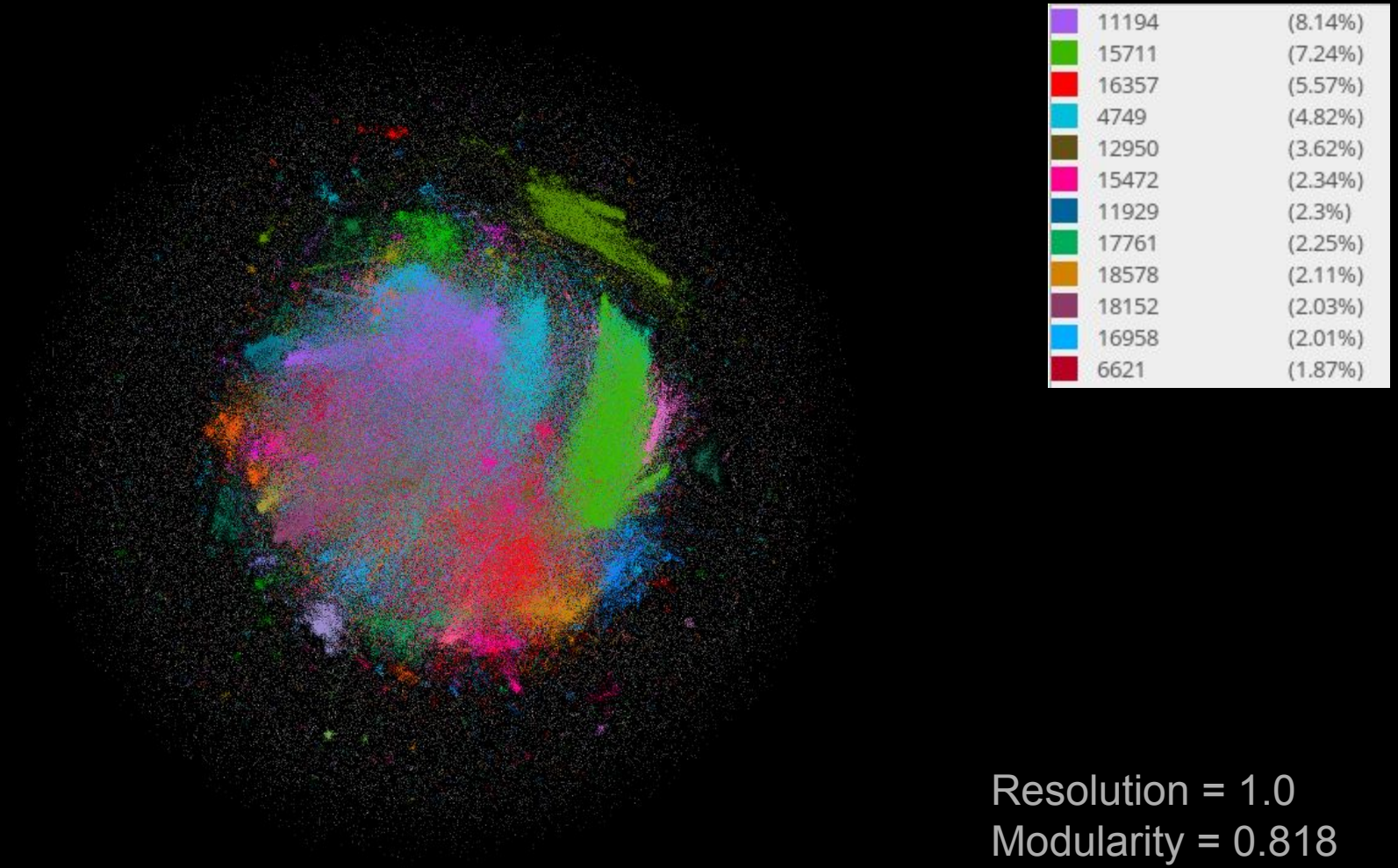
25484	(1.62%)
24438	(1.58%)
16237	(1.57%)
14832	(1.56%)
2875	(1.55%)
328	(1.54%)
10592	(1.52%)
69	(1.51%)
7357	(1.35%)
19070	(1.34%)
6158	(1.26%)
17600	(1.23%)

Resolution = 0.25
Modularity = 0.763

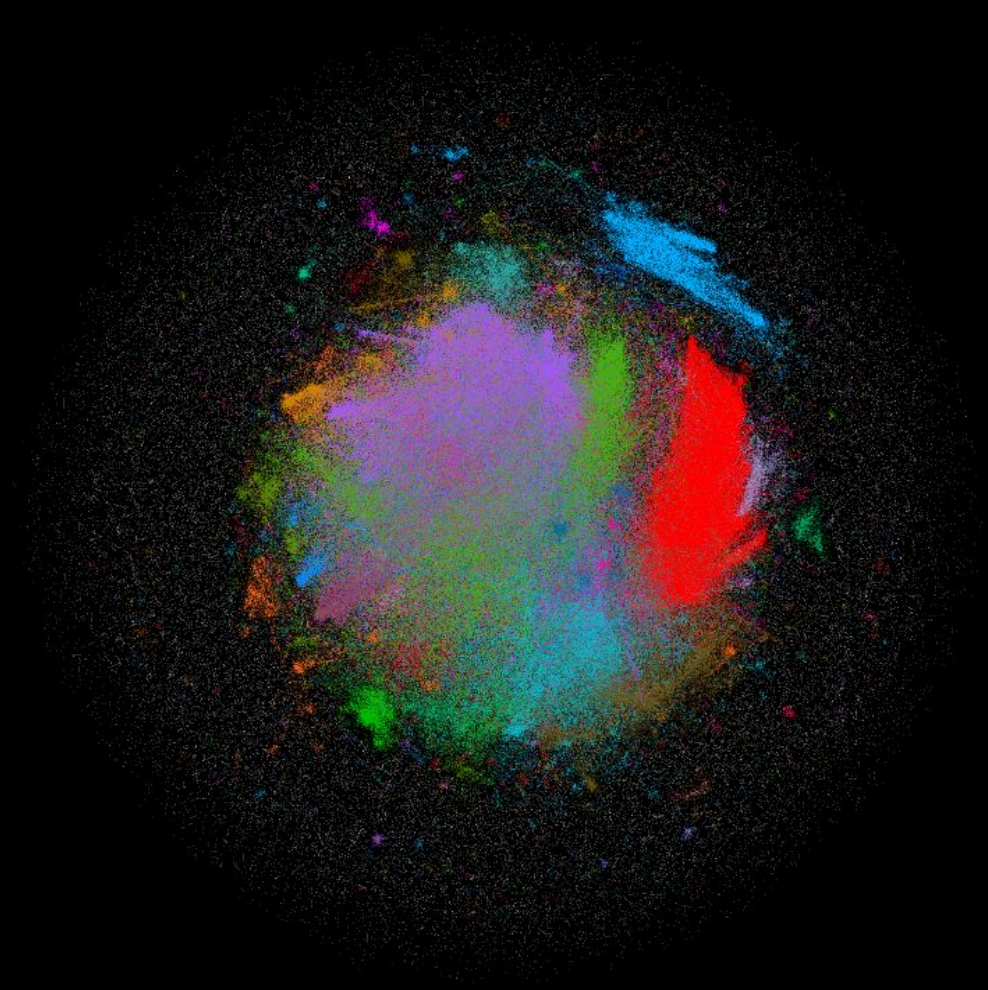


18923	(4.14%)
13868	(3.81%)
8991	(3.61%)
9080	(3.48%)
18223	(3.17%)
19110	(2.25%)
16548	(2.07%)
13678	(2.01%)
19189	(1.99%)
13425	(1.98%)
19495	(1.92%)
20161	(1.88%)

Resolution = 0.5
Modularity = 0.801

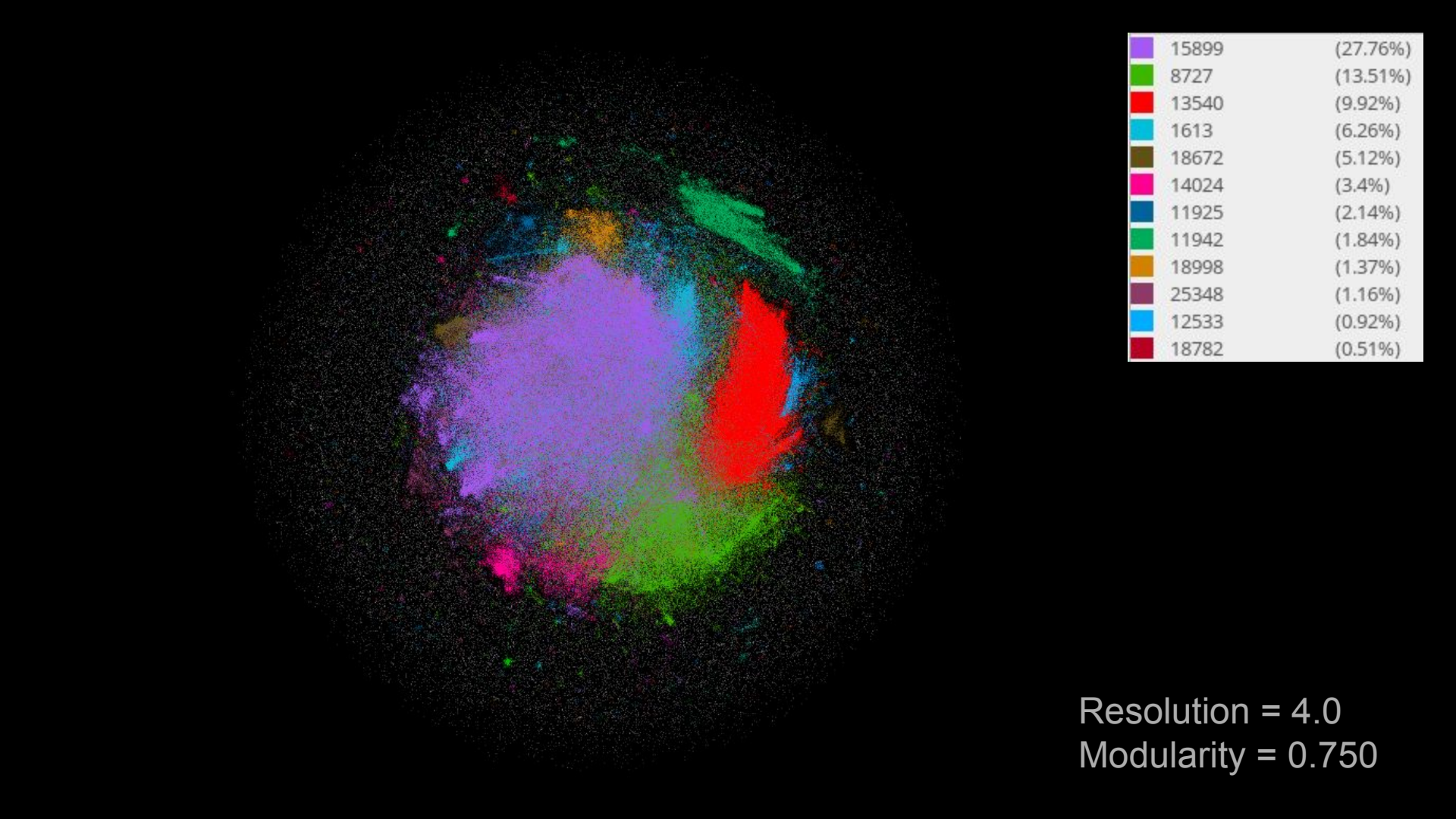


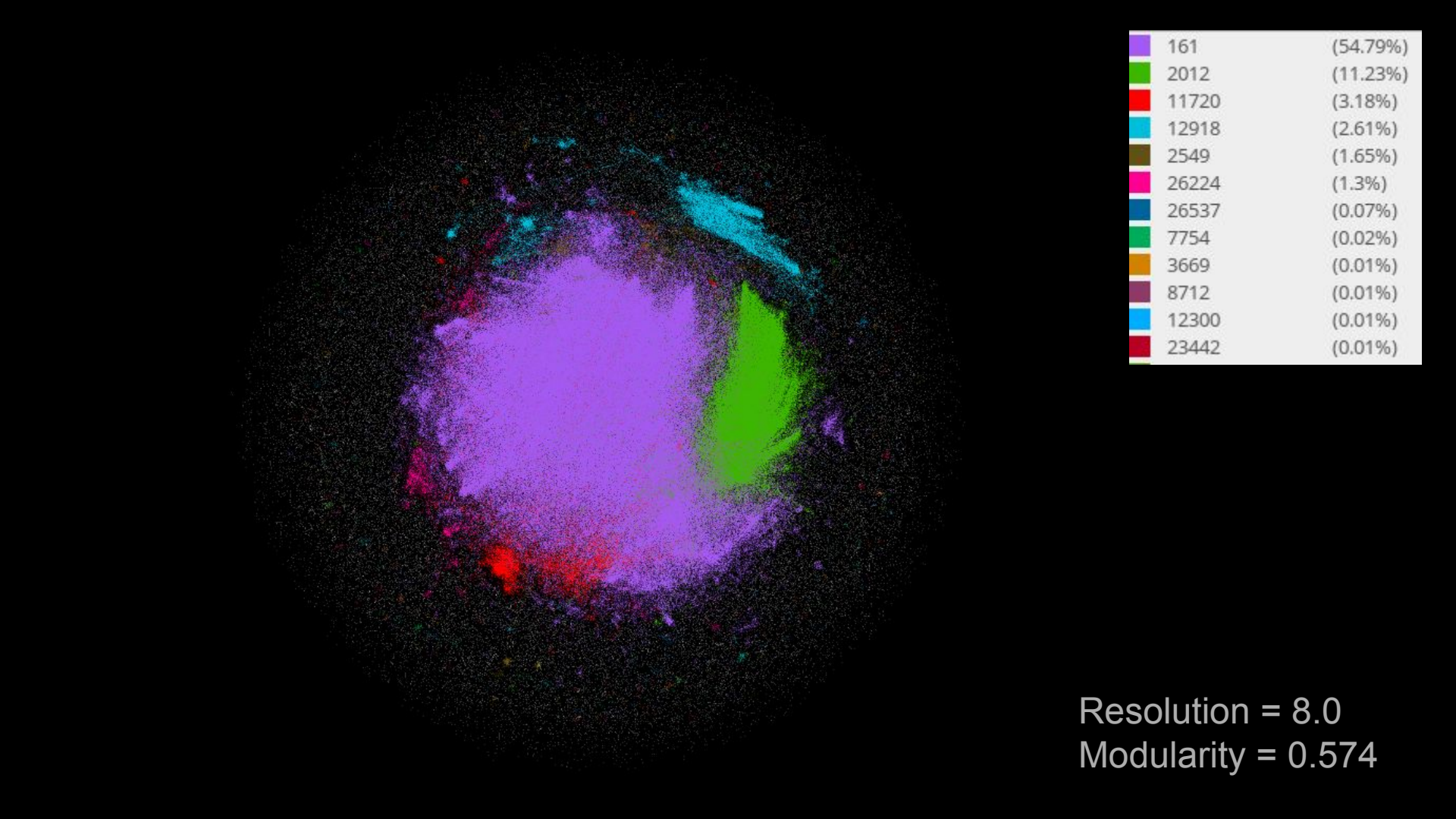
Resolution = 1.0
Modularity = 0.818

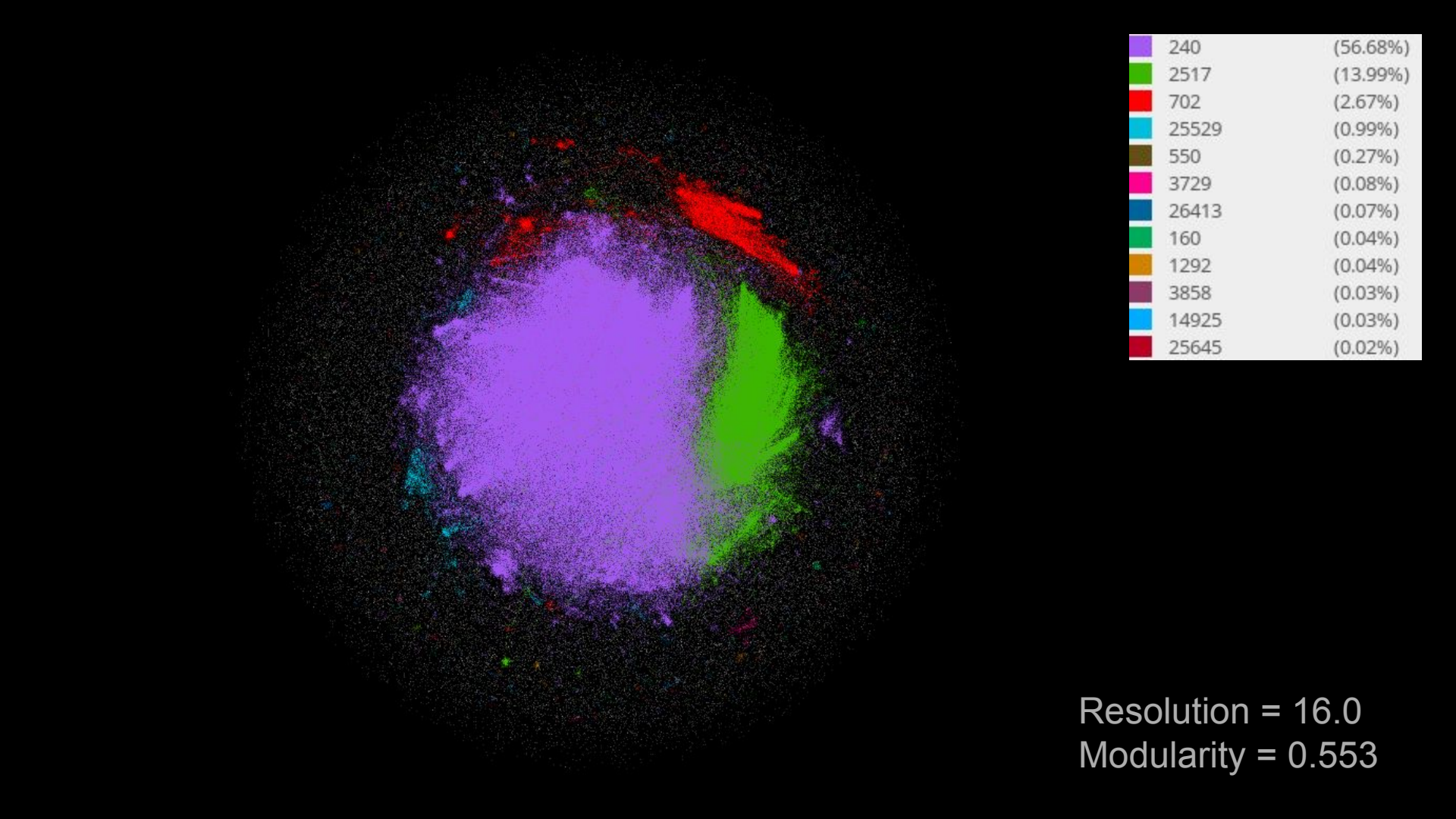


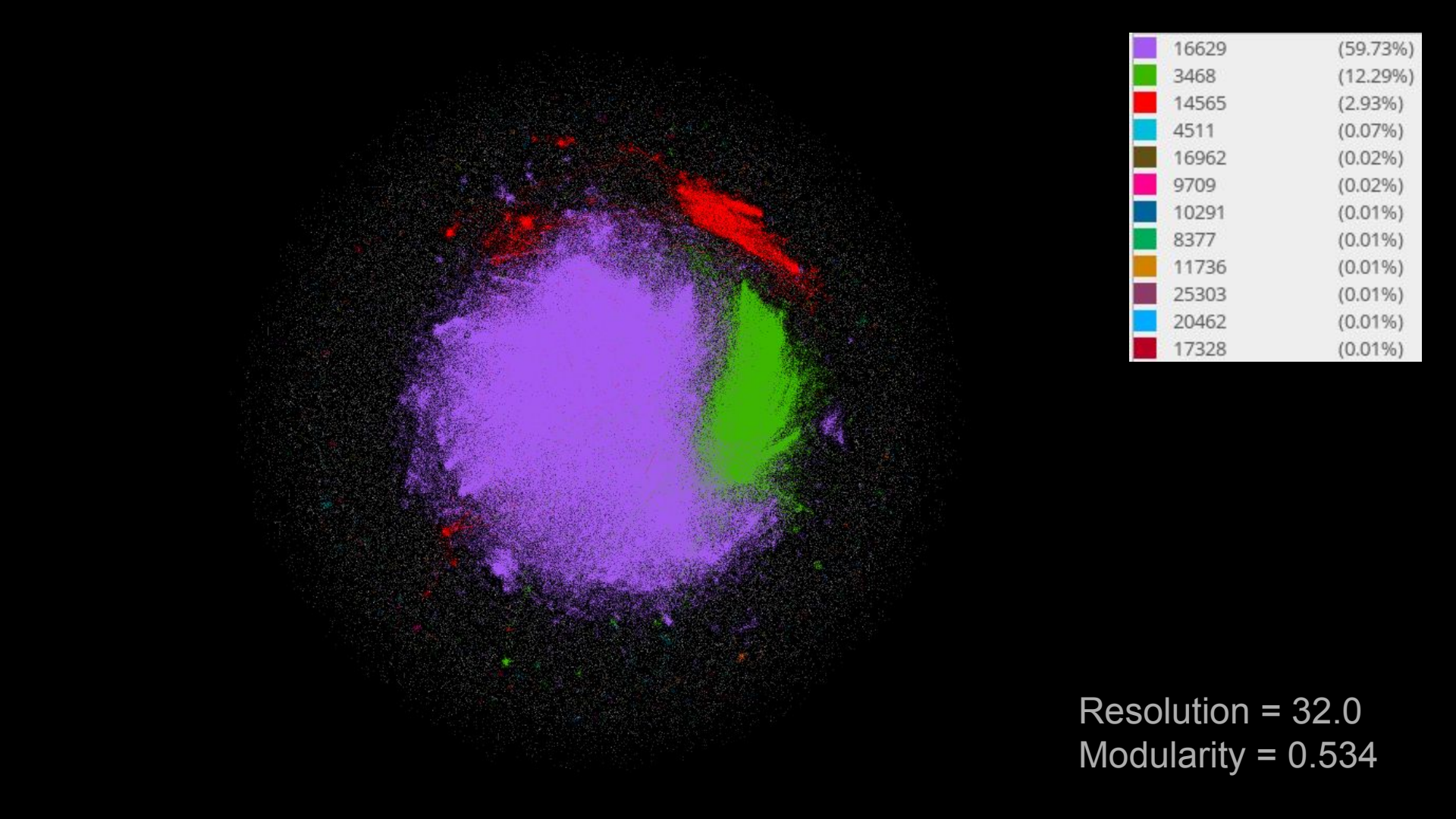
14707	(13.98%)
14074	(10.59%)
12490	(9%)
22630	(6.22%)
12107	(4.76%)
102	(3.28%)
13121	(2.71%)
20209	(2.63%)
12345	(2.61%)
13761	(2.54%)
15666	(1.61%)
14102	(1.56%)

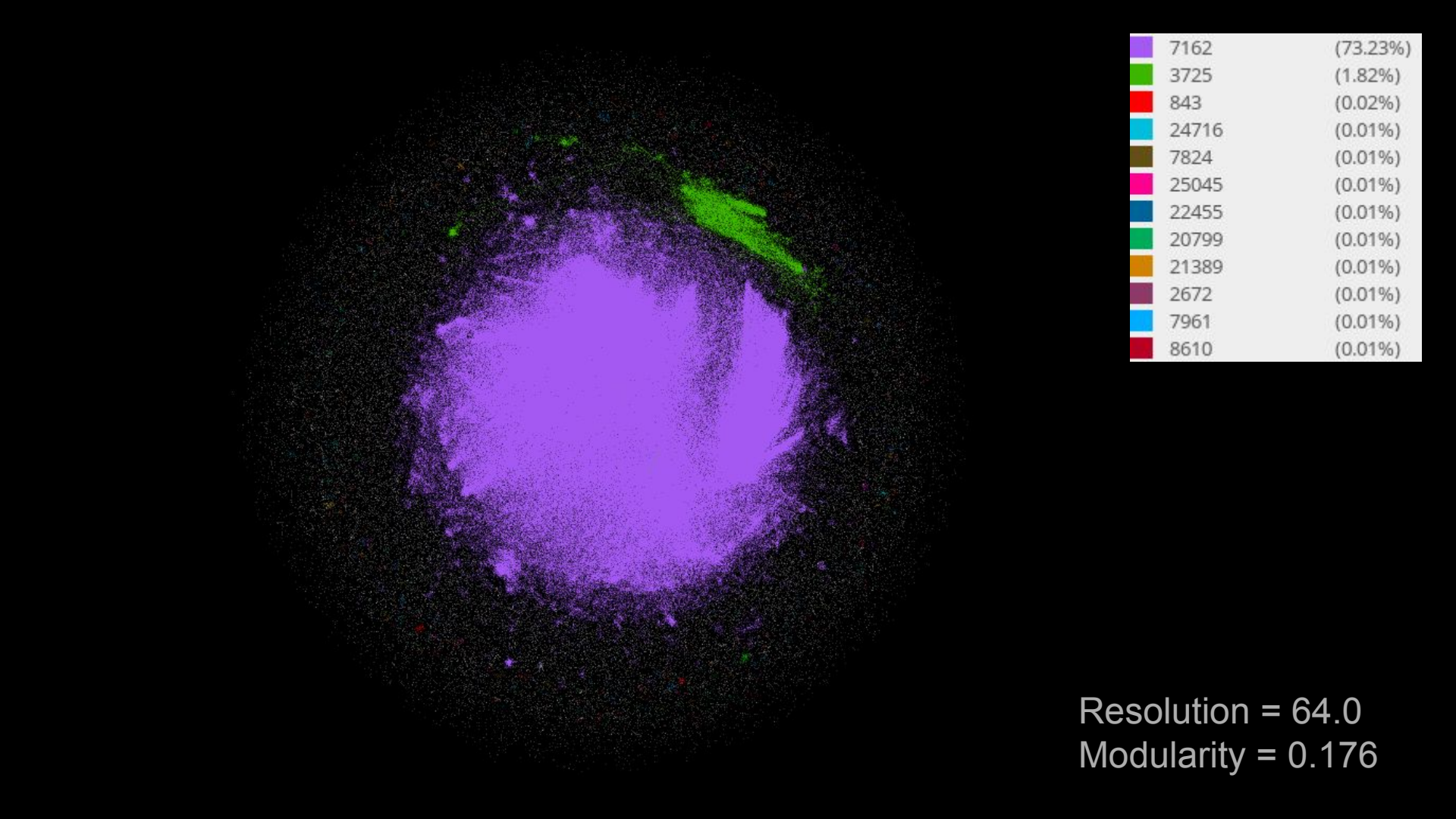
Resolution = 2.0
Modularity = 0.806


15899 (27.76%)
8727 (13.51%)
13540 (9.92%)
1613 (6.26%)
18672 (5.12%)
14024 (3.4%)
11925 (2.14%)
11942 (1.84%)
18998 (1.37%)
25348 (1.16%)
12533 (0.92%)
18782 (0.51%)
Resolution = 4.0
Modularity = 0.750

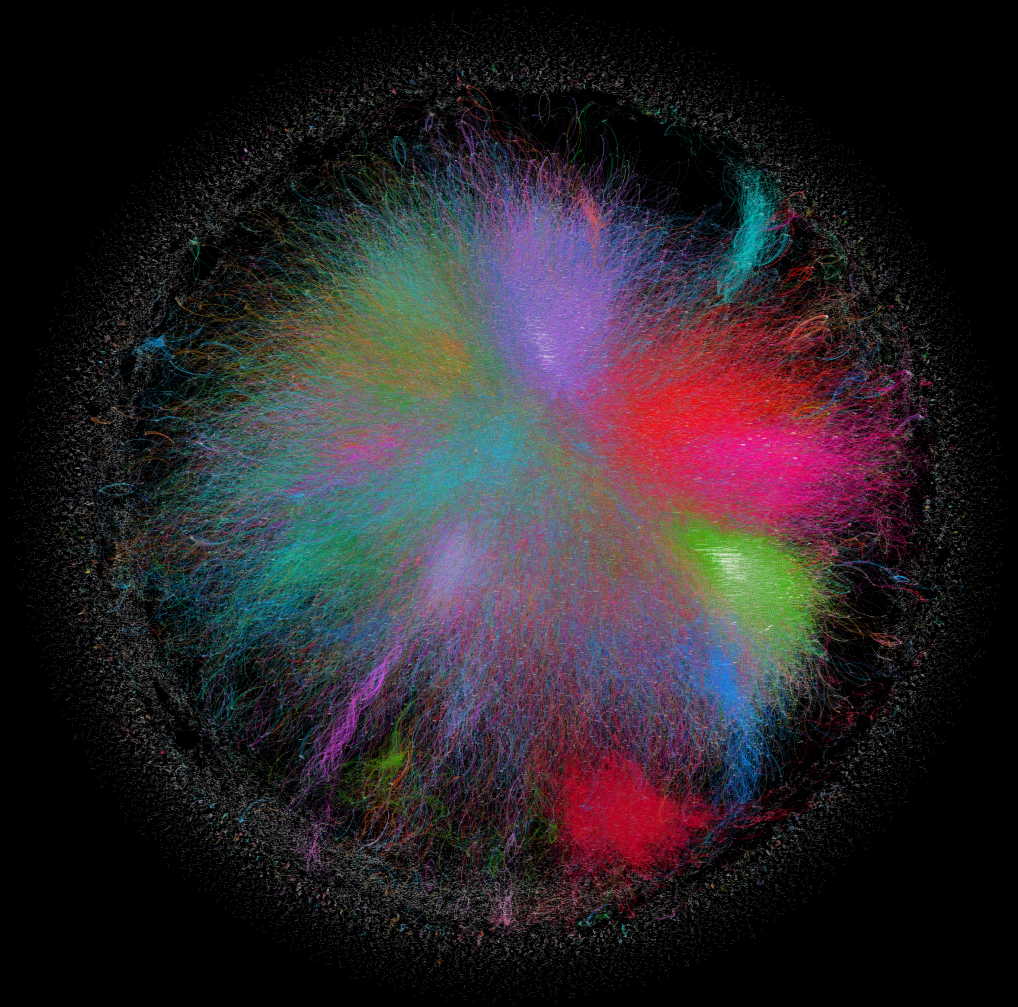

161 (54.79%)
2012 (11.23%)
11720 (3.18%)
12918 (2.61%)
2549 (1.65%)
26224 (1.3%)
26537 (0.07%)
7754 (0.02%)
3669 (0.01%)
8712 (0.01%)
12300 (0.01%)
23442 (0.01%)
Resolution = 8.0
Modularity = 0.574


240 (56.68%)
2517 (13.99%)
702 (2.67%)
25529 (0.99%)
550 (0.27%)
3729 (0.08%)
26413 (0.07%)
160 (0.04%)
1292 (0.04%)
3858 (0.03%)
14925 (0.03%)
25645 (0.02%)
Resolution = 16.0
Modularity = 0.553


16629 (59.73%)
3468 (12.29%)
14565 (2.93%)
4511 (0.07%)
16962 (0.02%)
9709 (0.02%)
10291 (0.01%)
8377 (0.01%)
11736 (0.01%)
25303 (0.01%)
20462 (0.01%)
17328 (0.01%)
Resolution = 32.0
Modularity = 0.534

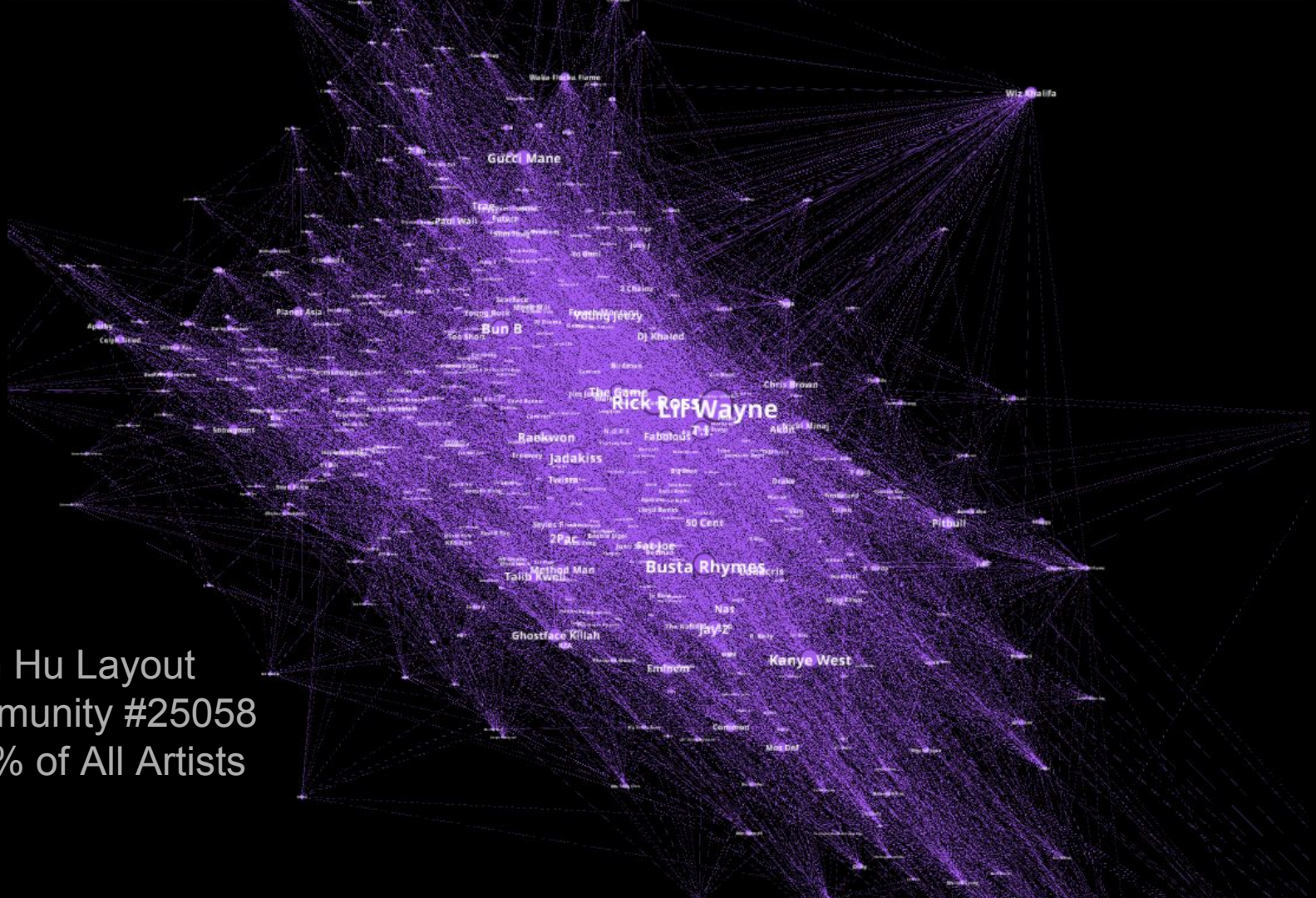

7162 (73.23%)
3725 (1.82%)
843 (0.02%)
24716 (0.01%)
7824 (0.01%)
25045 (0.01%)
22455 (0.01%)
20799 (0.01%)
21389 (0.01%)
2672 (0.01%)
7961 (0.01%)
8610 (0.01%)
Resolution = 64.0
Modularity = 0.176

Let’s try one more layout algorithm.



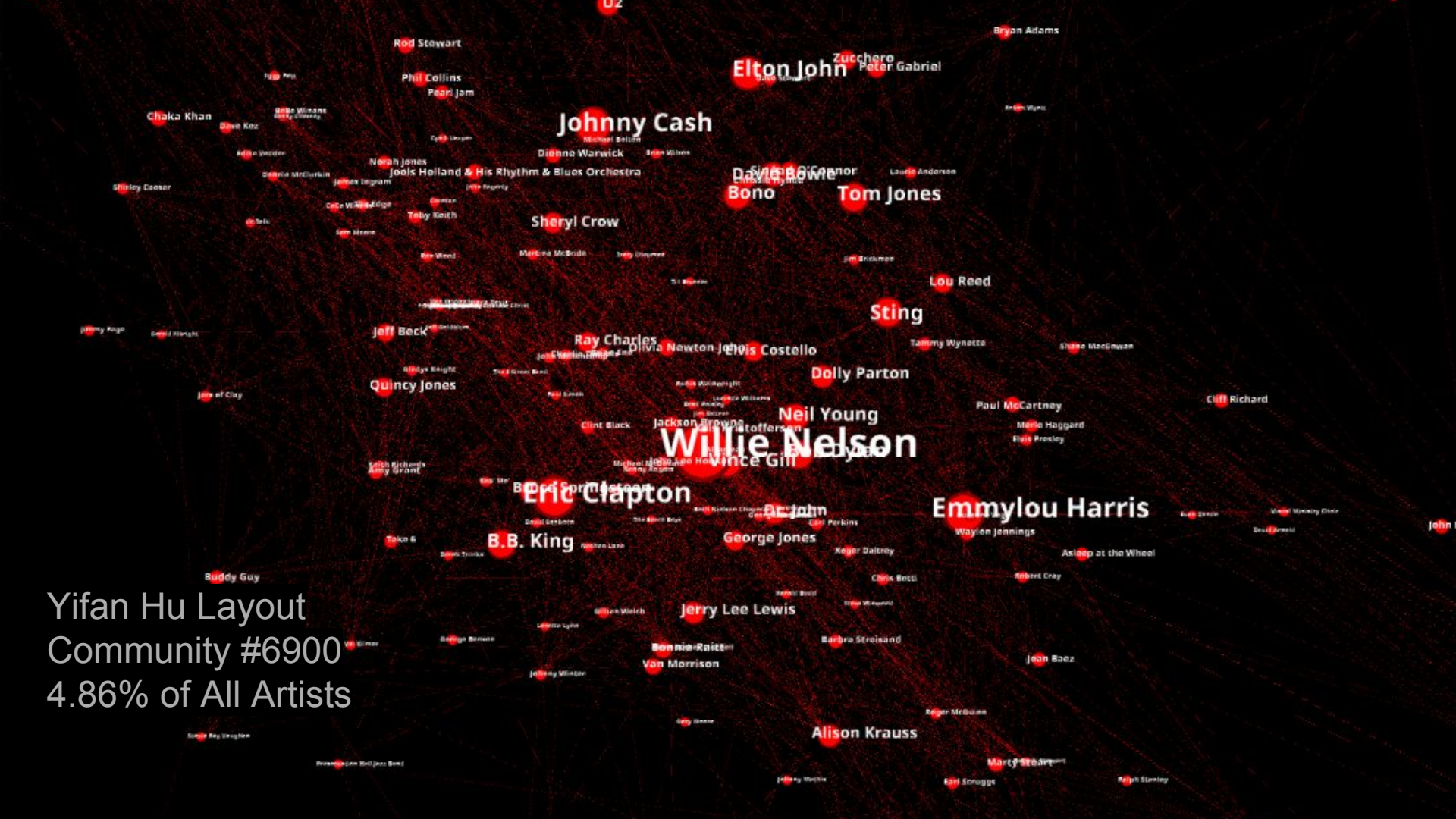
Yifan Hu Layout
Weighted Graph
271442 Nodes
650920 Edges

We can filter the nodes by community.

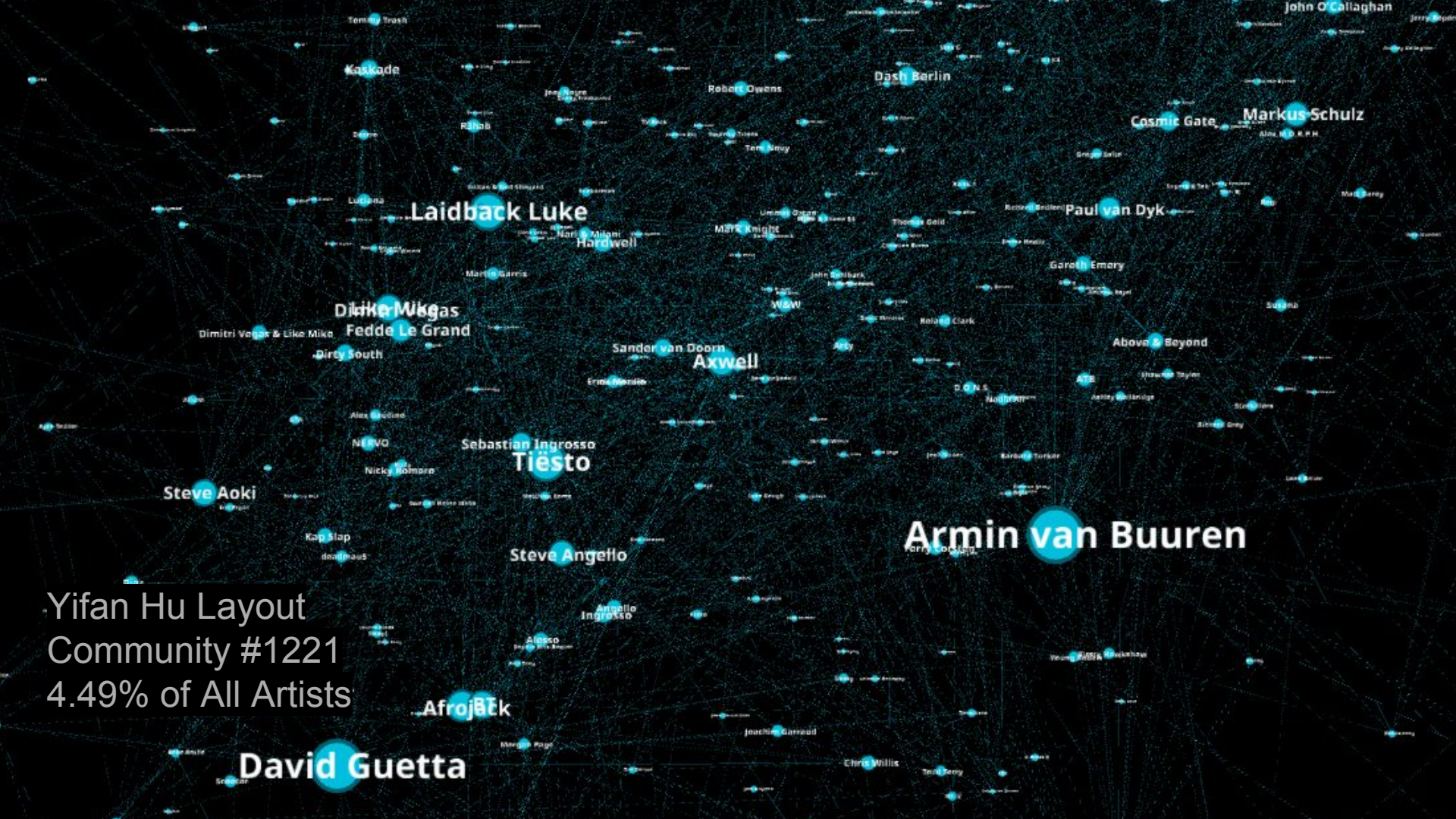


Yifan Hu Layout
Community #25058
7.54% of All Artists

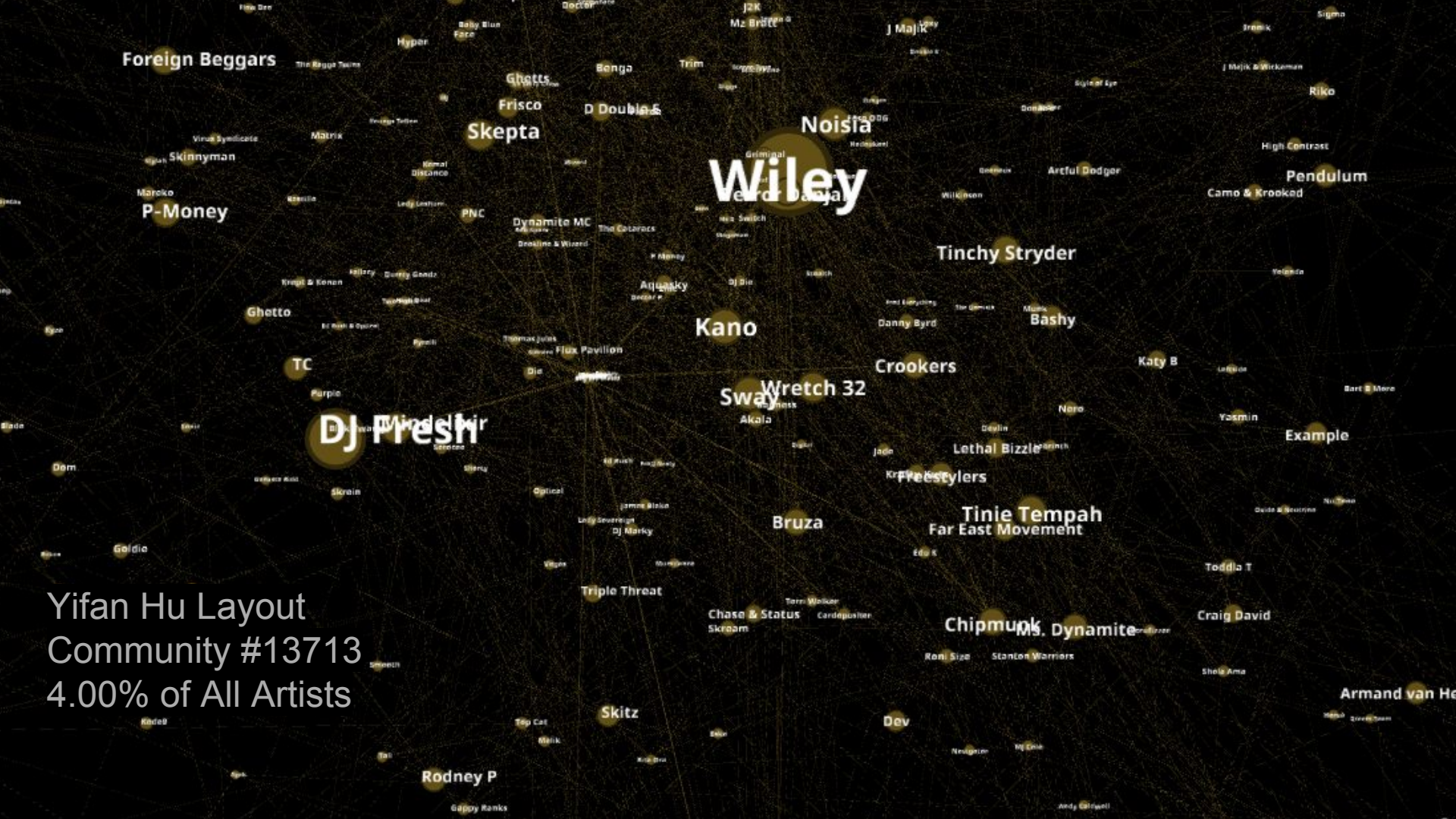
Yifan Hu Layout
Community #12163
6.93% of All Artists



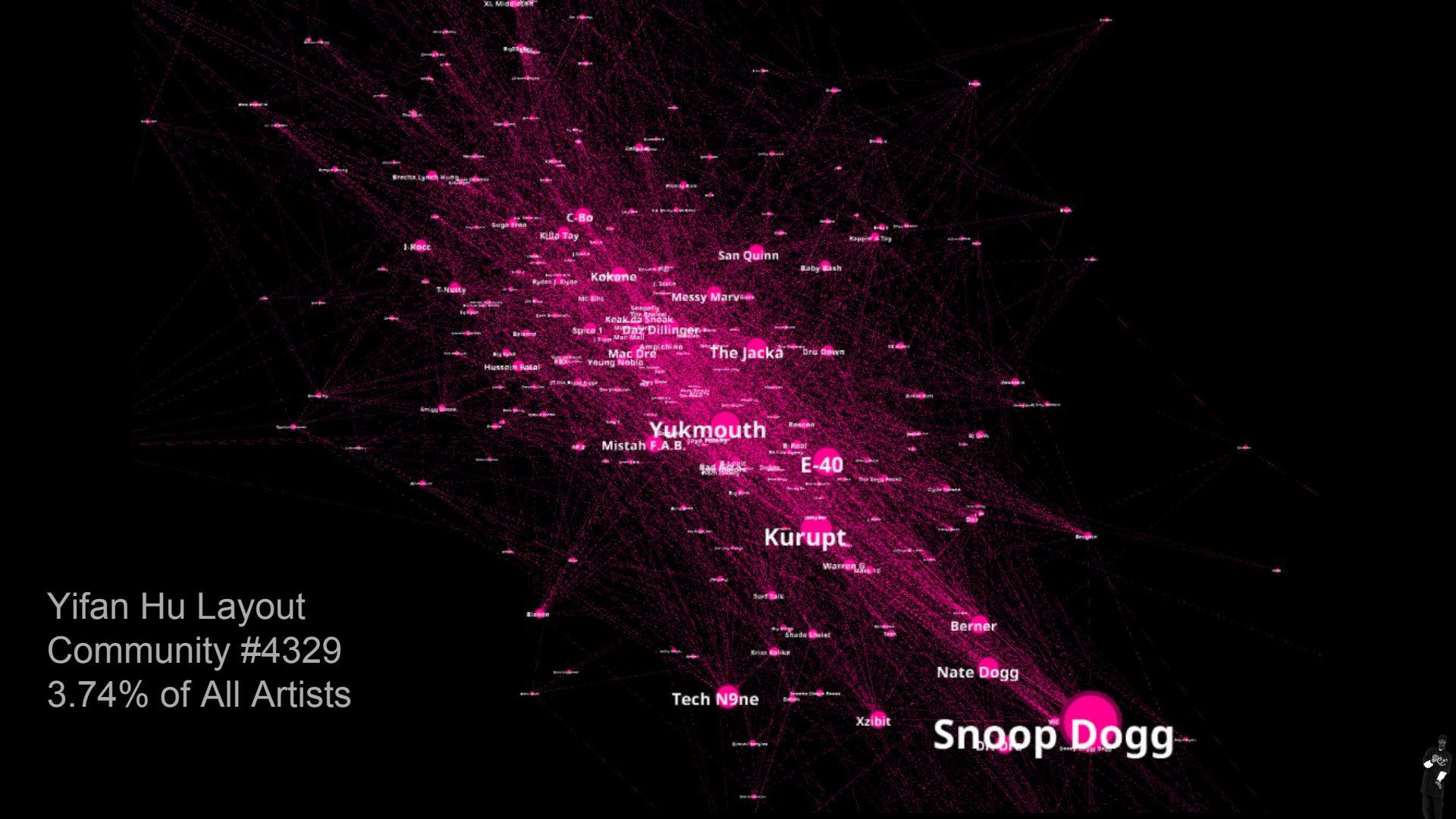
Yifan Hu Layout
Community #6900
4.86% of All Artists



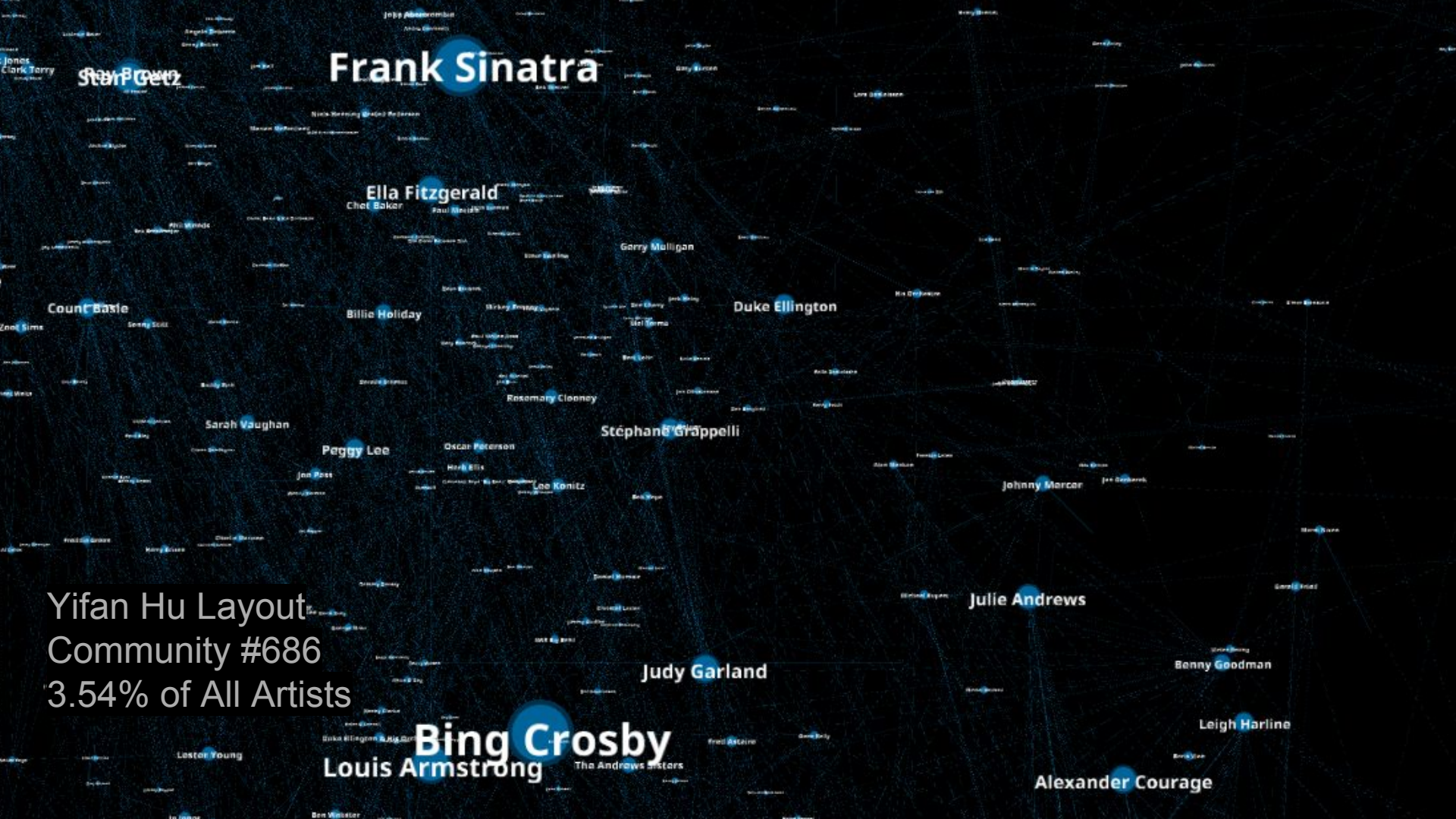
Yifan Hu Layout
Community #1221
4.49% of All Artists



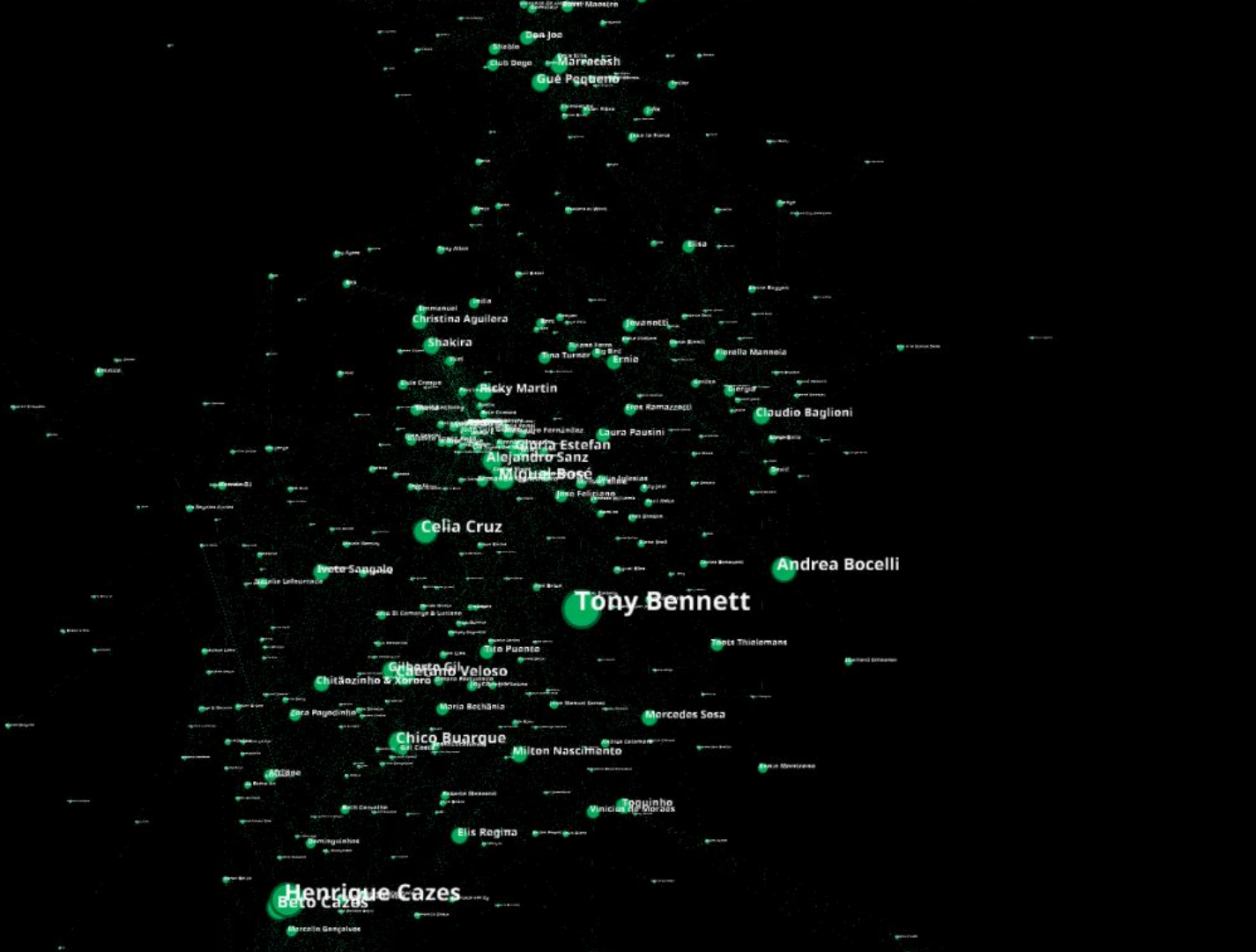
Yifan Hu Layout
Community #13713
4.00% of All Artists



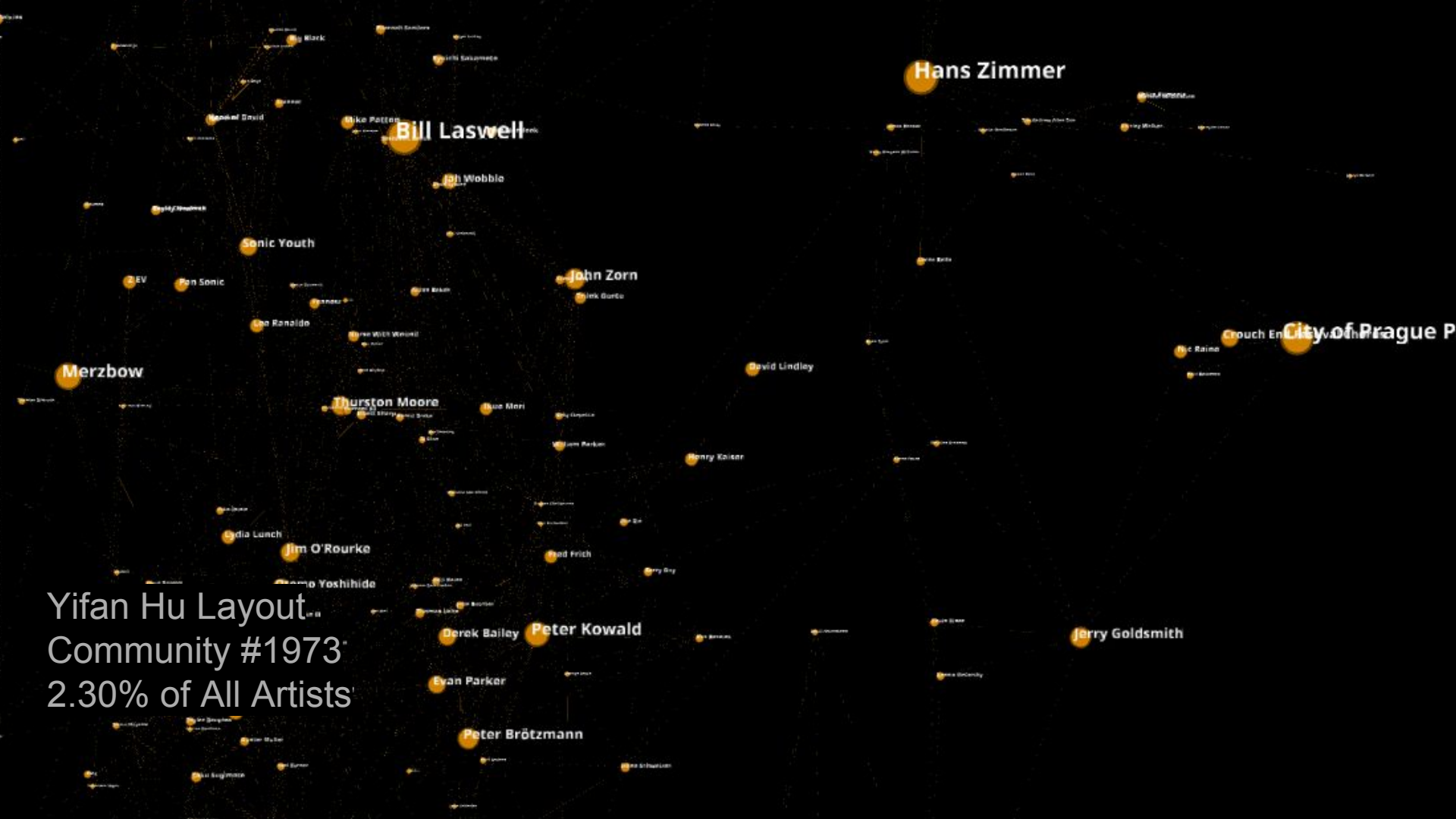
Yifan Hu Layout
Community #4329
3.74% of All Artists

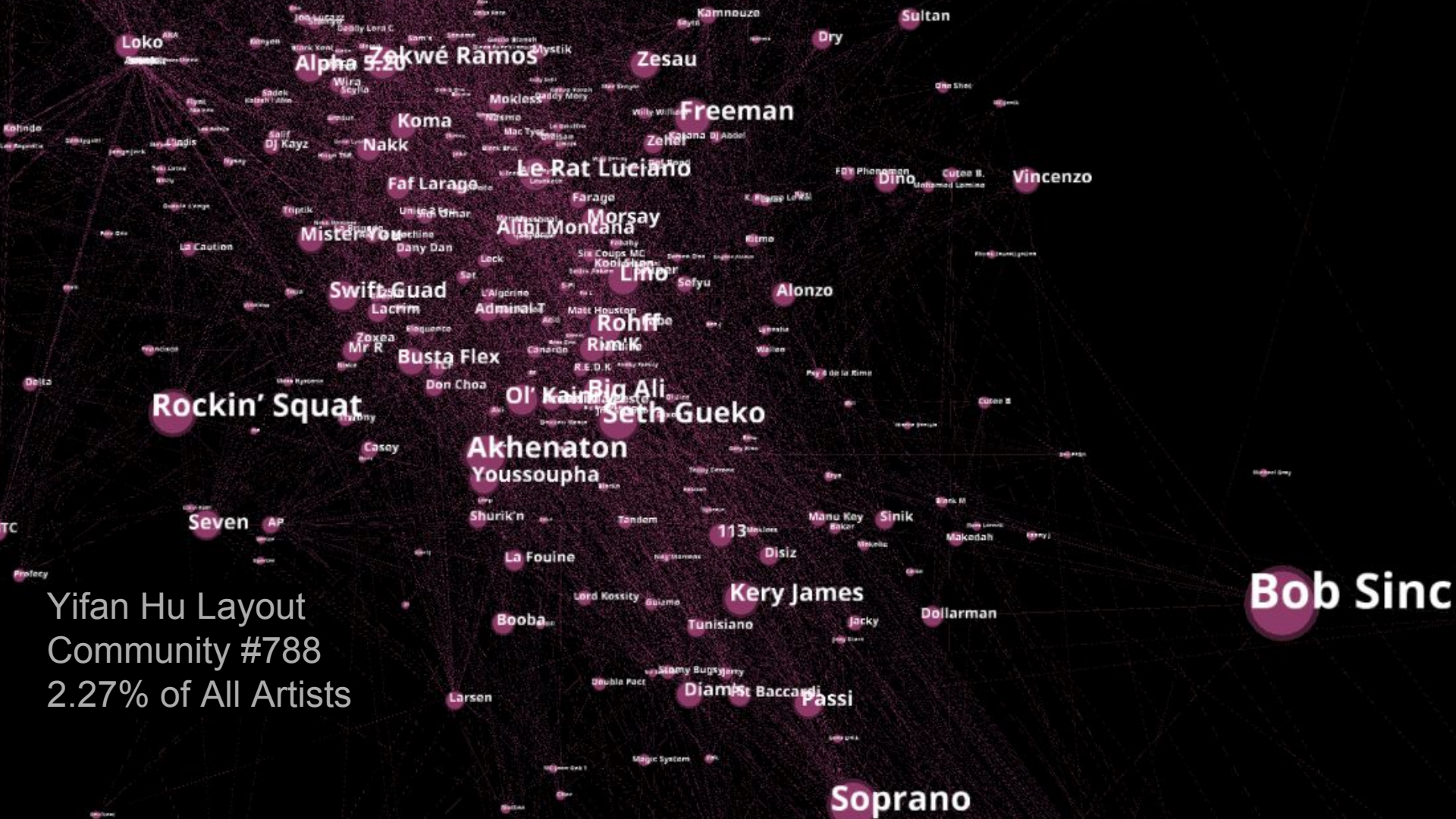


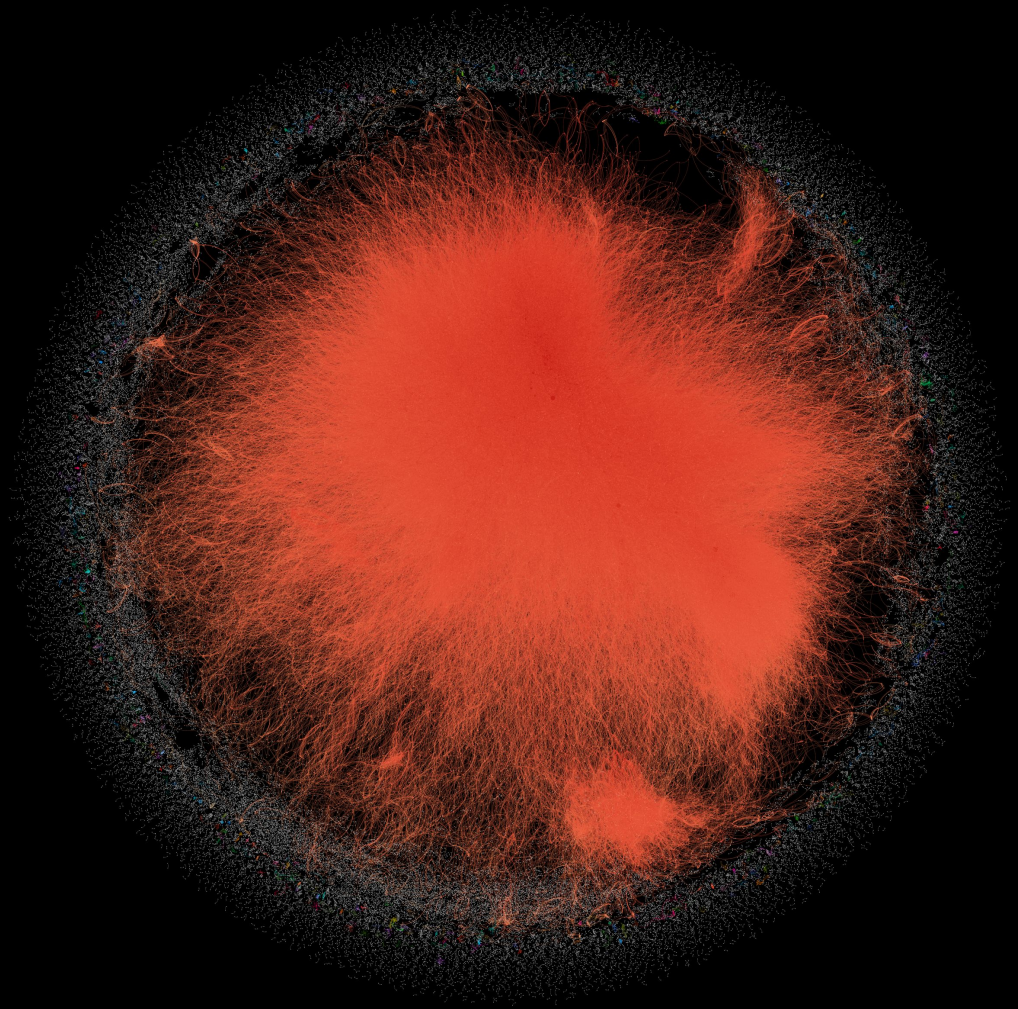
Yifan Hu Layout
Community #686
3.54% of All Artists



Yifan Hu Layout
Community #303
2.70% of All Artists


Hans Zimmer
Bill Laswell
Mike Patton
Jah Wobble
Sonic Youth
John Zorn
Merzbow
Thurston Moore
David Lindley
Henry Kaiser
Lydia Lunch
Jim O'Rourke
Derek Bailey
Peter Kowald
Jerry Goldsmith
Evan Parker
Peter Brötzmann
Yifan Hu Layout
Community #1973
2.30% of All Artists


Yifan Hu Layout
Community #788
2.27% of All Artists
Loko
Alpha 5.20
Zakwé Ramos
Zesau
Sultan
Dry
Kamnouze
Freeman
Koma
Nakk
Le Rat Luciano
Faf Larage
Vincenzo
Dino
Morsay
Alibi Montana
Mister You
Lino
Alonzo
Swift Guad
Lacrim
Rohff
Busta Flex
Rockin' Squat
Seth Gueko
Akhenaton
Youssoupha
Seven
Sinik
Kery James
Dollarman
Tunisiano
Booba
Diam's
Passi
Soprano
Bob Sinc



Yifan Hu Layout
SDN Heat Map

Quiz

1. When can you use breadth-first search to compute shortest paths in a graph?

2. List two factors you should consider when deciding on an appropriate layout type (e.g, force-directed, circular, etc.) for your graph.

3. What does the modularity (Q value) of a graph measure?

4. In the weighted music collaboration graph, edges had a weight which represented the number of distinct collaborations two artists had. How do you think these weights helped improve the accuracy of the community-finding algorithm over unweighted edges?