

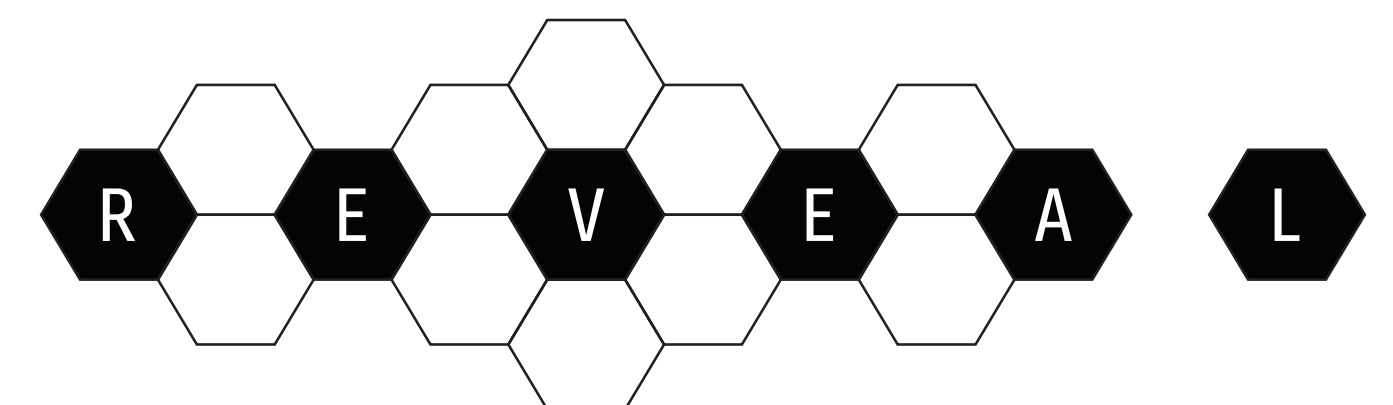
Visualizing Project Evolution

A Deterministic Poisson Disk Sampling Approach

Ian Abuaf Pelo, Michele Lanza, Marco Raglianti

07/07/2026

REVEAL @ Software Institute — USI, Lugano

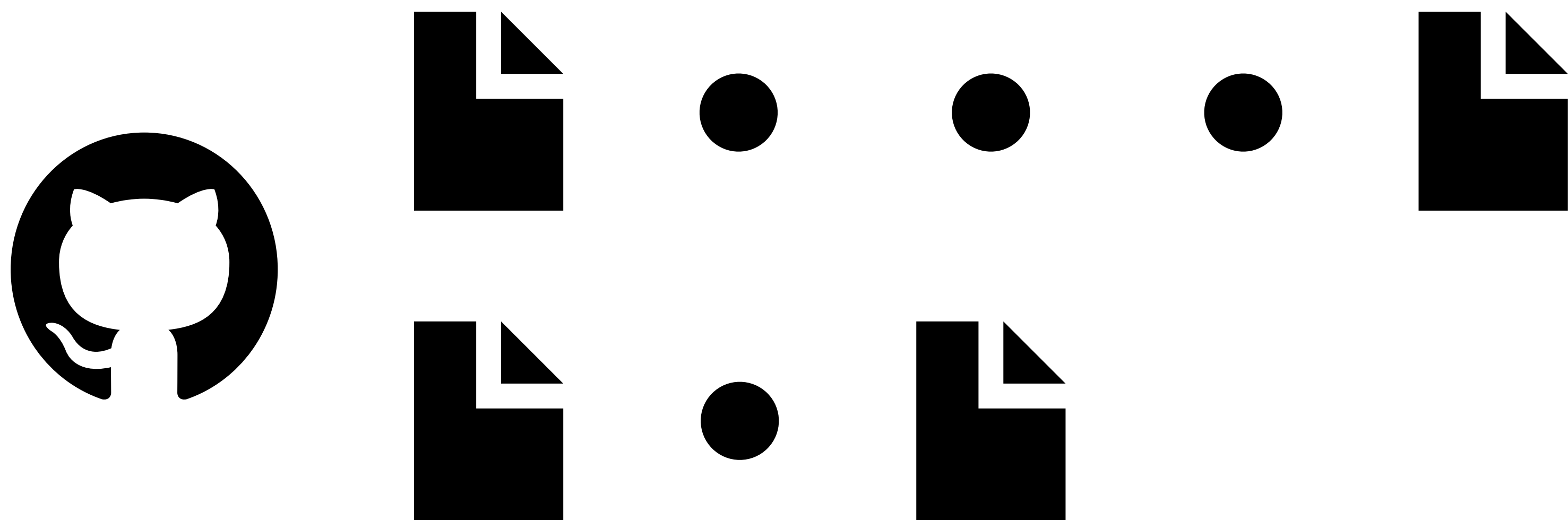




SOFTWARE

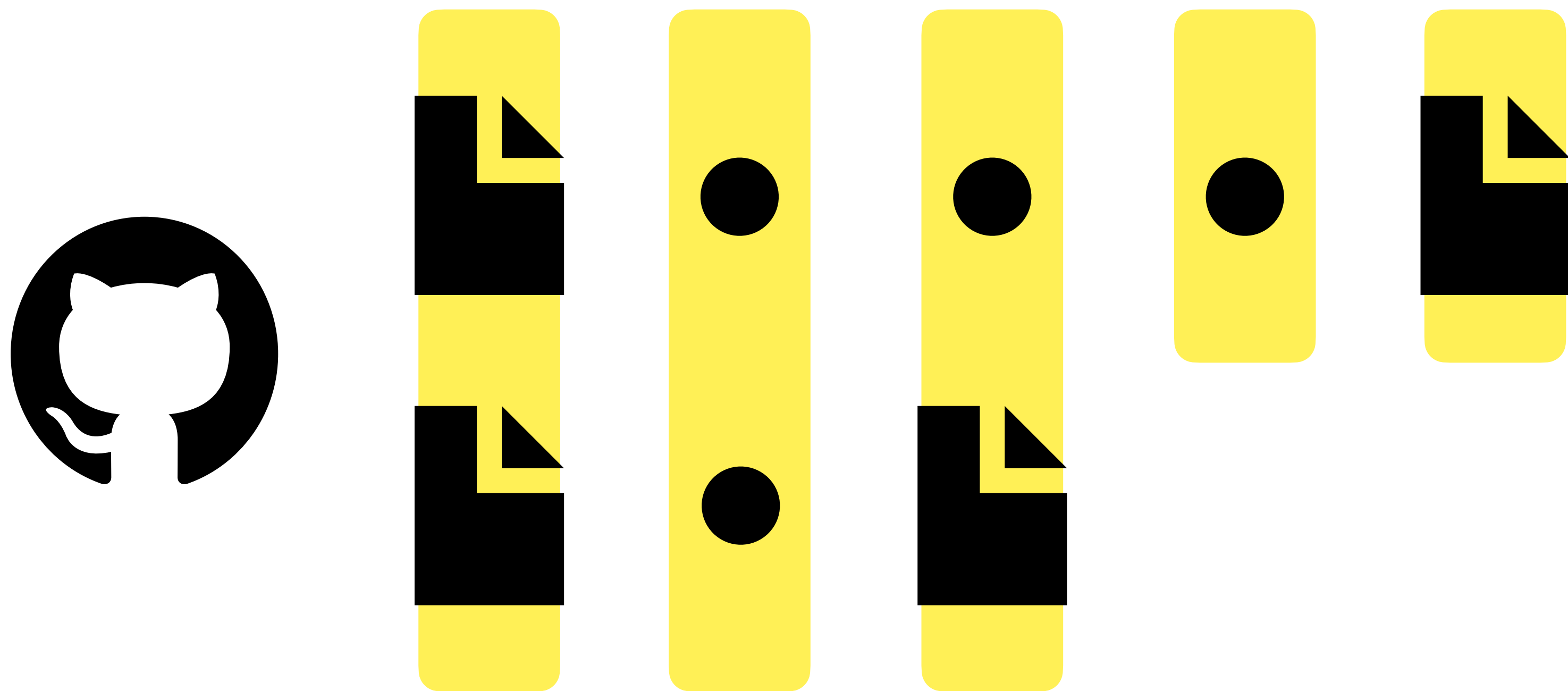
CHANGES

How to visualize Project Evolution



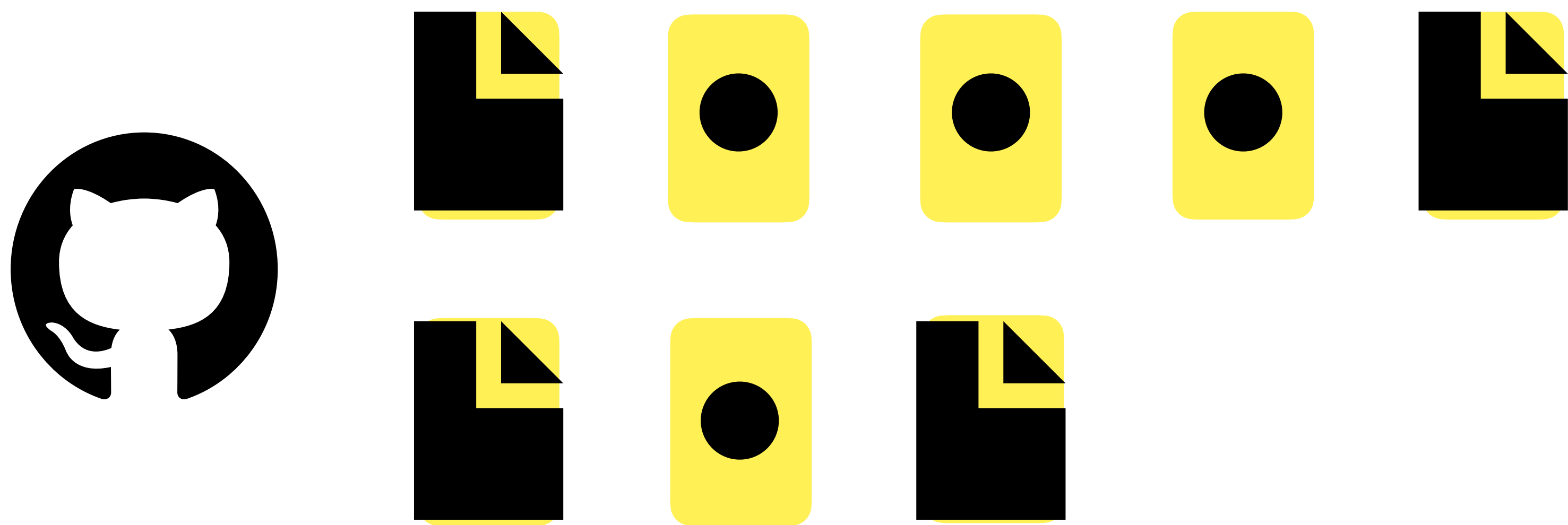
How to visualize Project Evolution

Commits



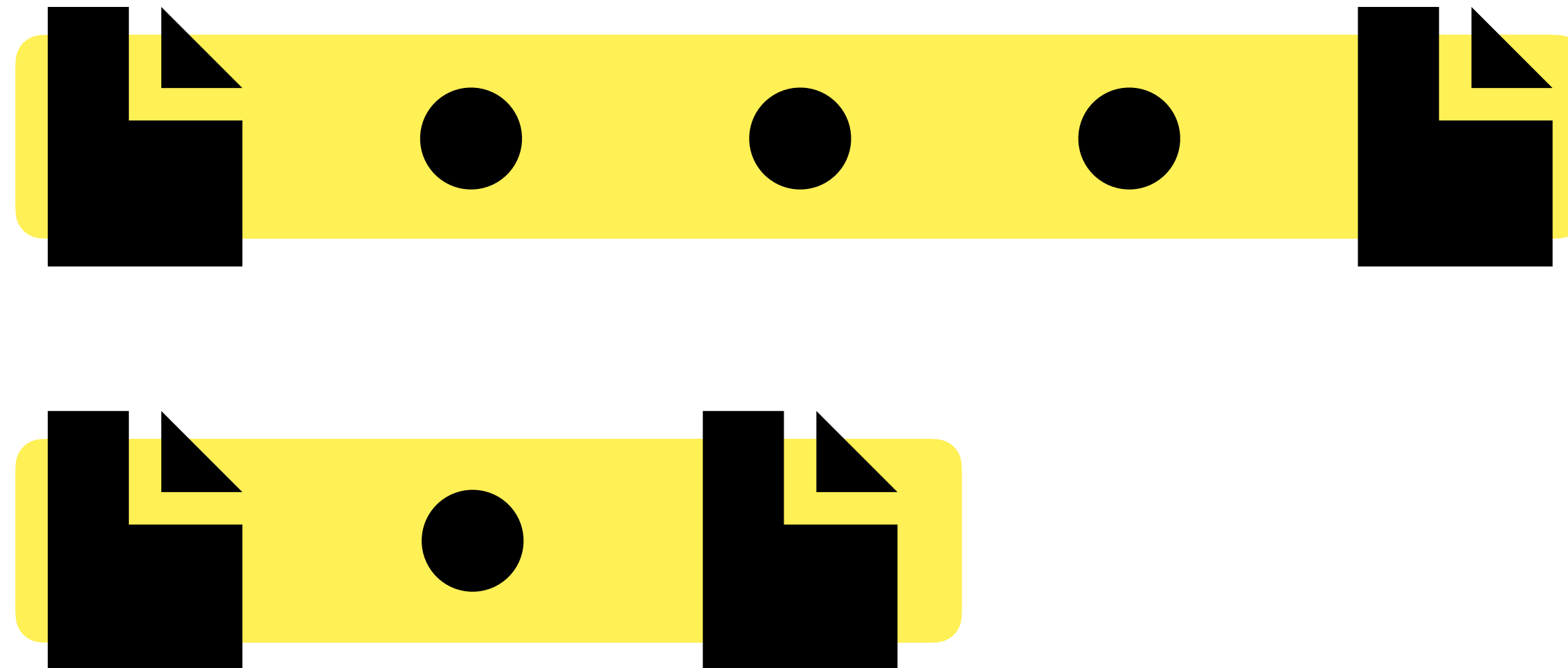
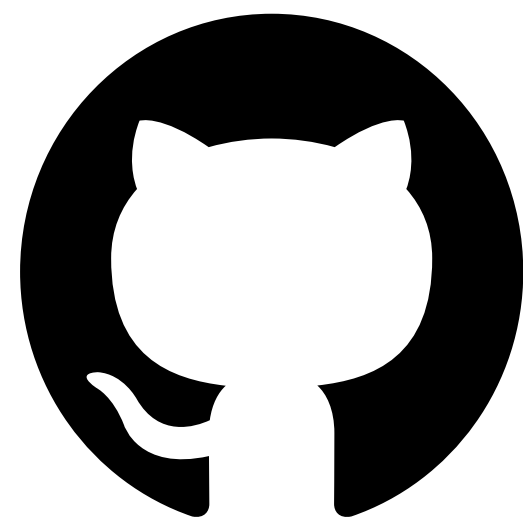
How to visualize Project Evolution

File Versions



How to visualize Project Evolution

File Histories

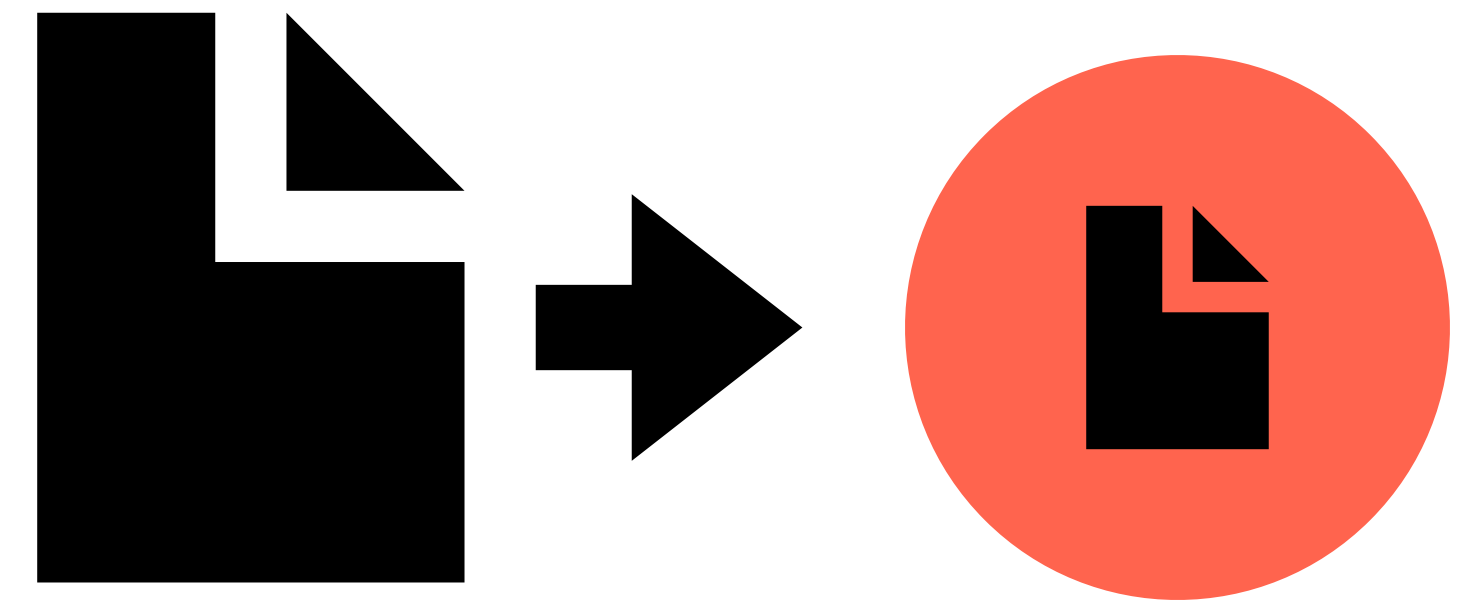


The steps to Software Visualization

- Map from a *domain object* to a *visual object*
 - What do we even want to visualize?
 - Do we care about non-code artifacts?
 - How should objects vary from each other?
 - What properties should be mapped and to what?

The steps to Software Visualization

- Map from a *domain object* to a *visual object*
 - What do we even want to visualize?
 - Do we care about non-code artifacts?
 - How should objects vary from each other?
 - What properties should be mapped and to what?

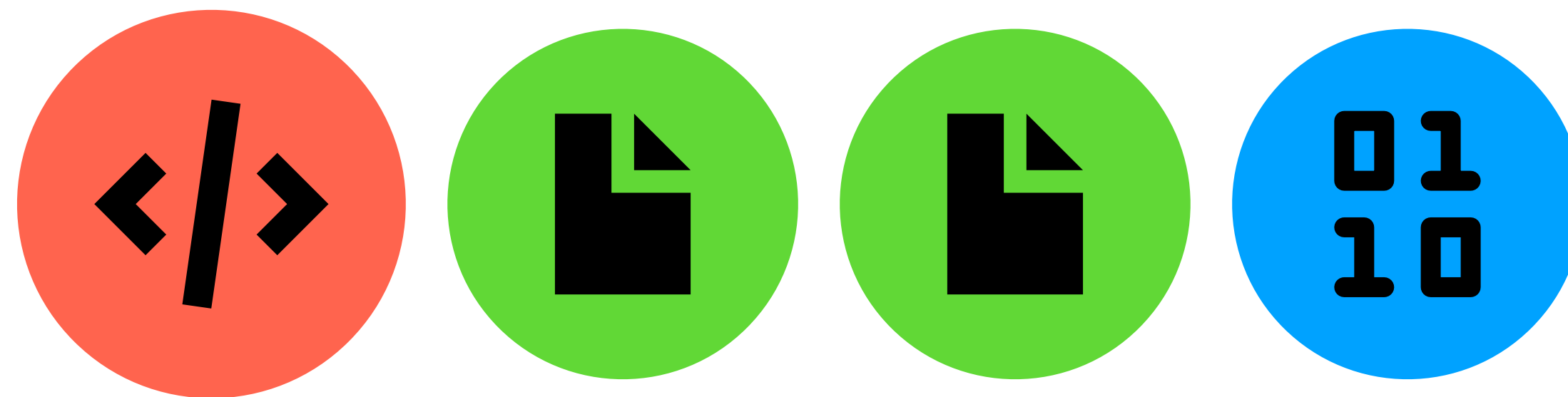


The steps to Software Visualization

- Map from a *domain object* to a *visual object*
- Use a *layout algorithm* to place the visual objects in the visualization
 - Determine the placement of every visual element

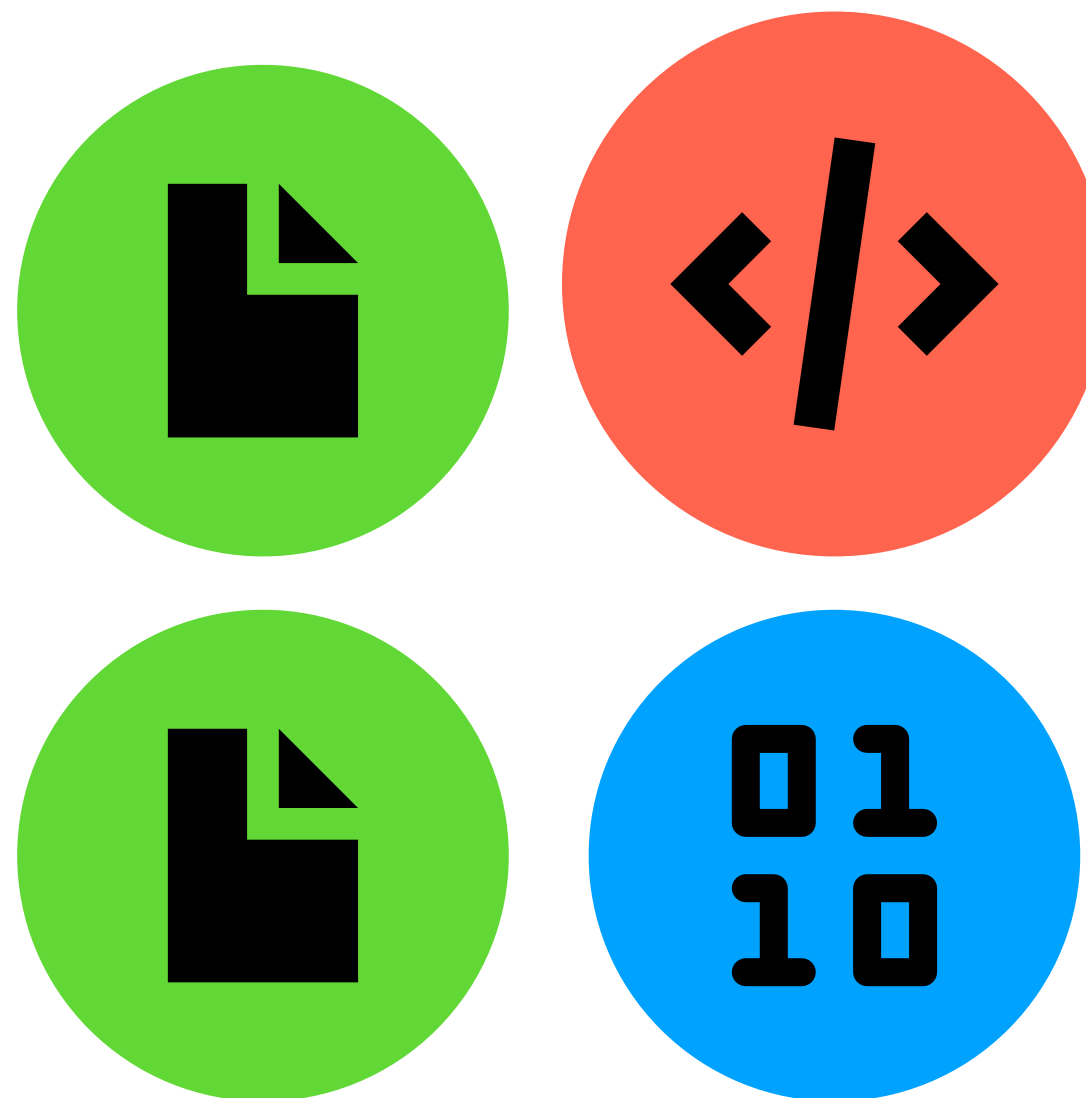
The steps to Software Visualization

- Map from a *domain object* to a *visual object*
- Use a *layout algorithm* to place the visual objects in the visualization
 - Determine the placement of every visual element



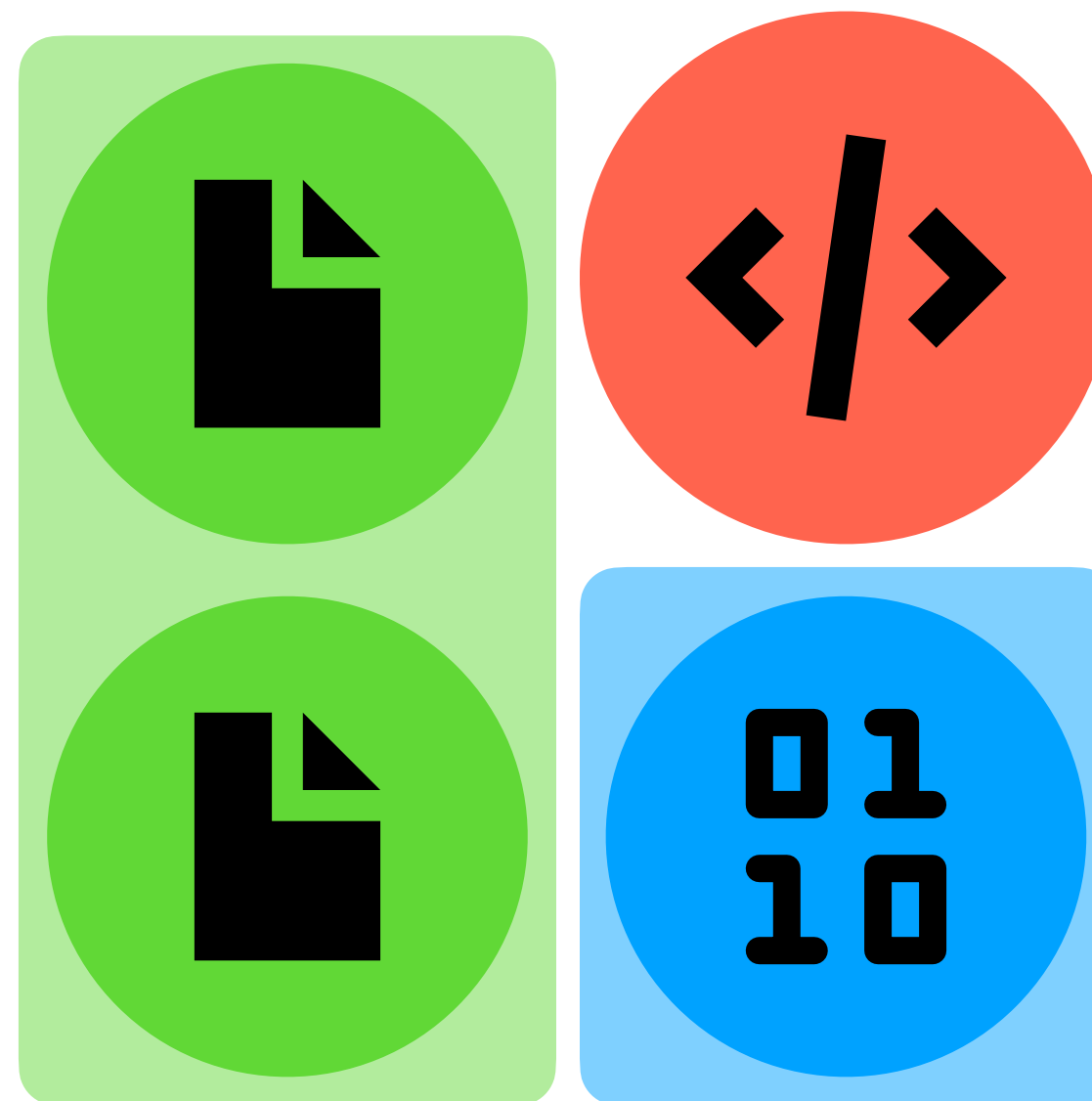
The steps to Software Visualization

- Map from a *domain object* to a *visual object*
- Use a *layout algorithm* to place the visual objects in the visualization
 - Determine the placement of every visual element



The steps to Software Visualization

- Map from a *domain object* to a *visual object*
- Use a *layout algorithm* to place the visual objects in the visualization
 - Determine the placement of every visual element

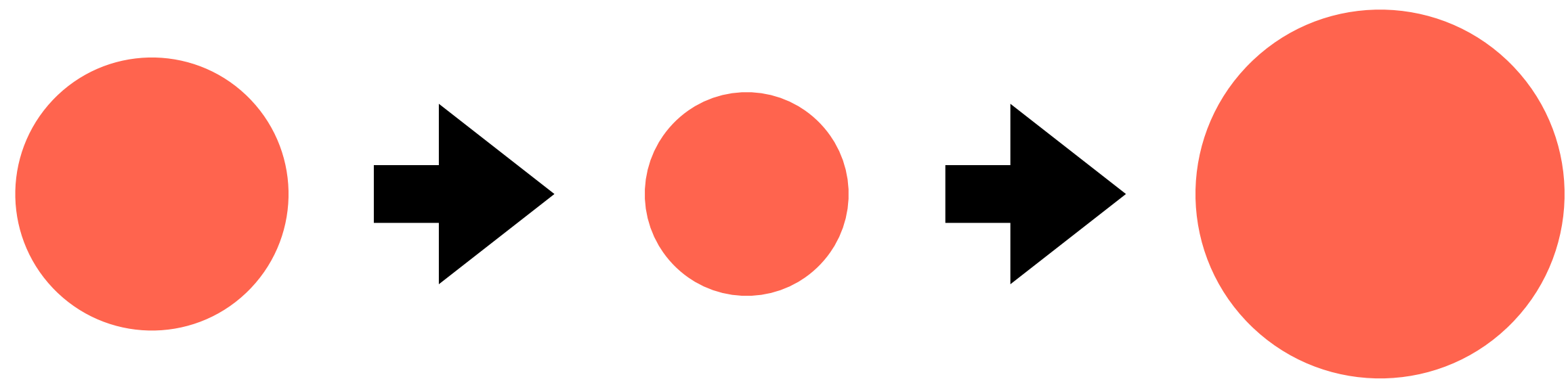


The steps to Software Visualization

- Map from a *domain object* to a *visual object*
- Use a *layout algorithm* to place the visual objects in the visualization
- Animate the visual objects to represent the evolution of the domain object
 - Changes to the domain object should reflect onto the visual object

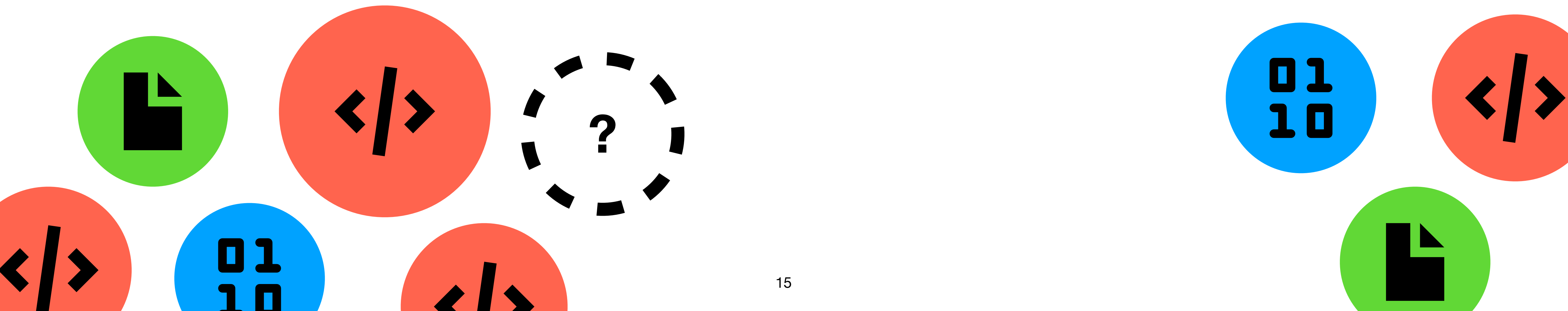
The steps to Software Visualization

- Map from a *domain object* to a *visual object*
- Use a *layout algorithm* to place the visual objects in the visualization
- Animate the visual objects to represent the evolution of the domain object
 - Changes to the domain object should reflect onto the visual object



Placing Visual Elements is hard

- The layout algorithm should convey information of the represented domain element
- Must resist evolution, in the form of changes to the visual element



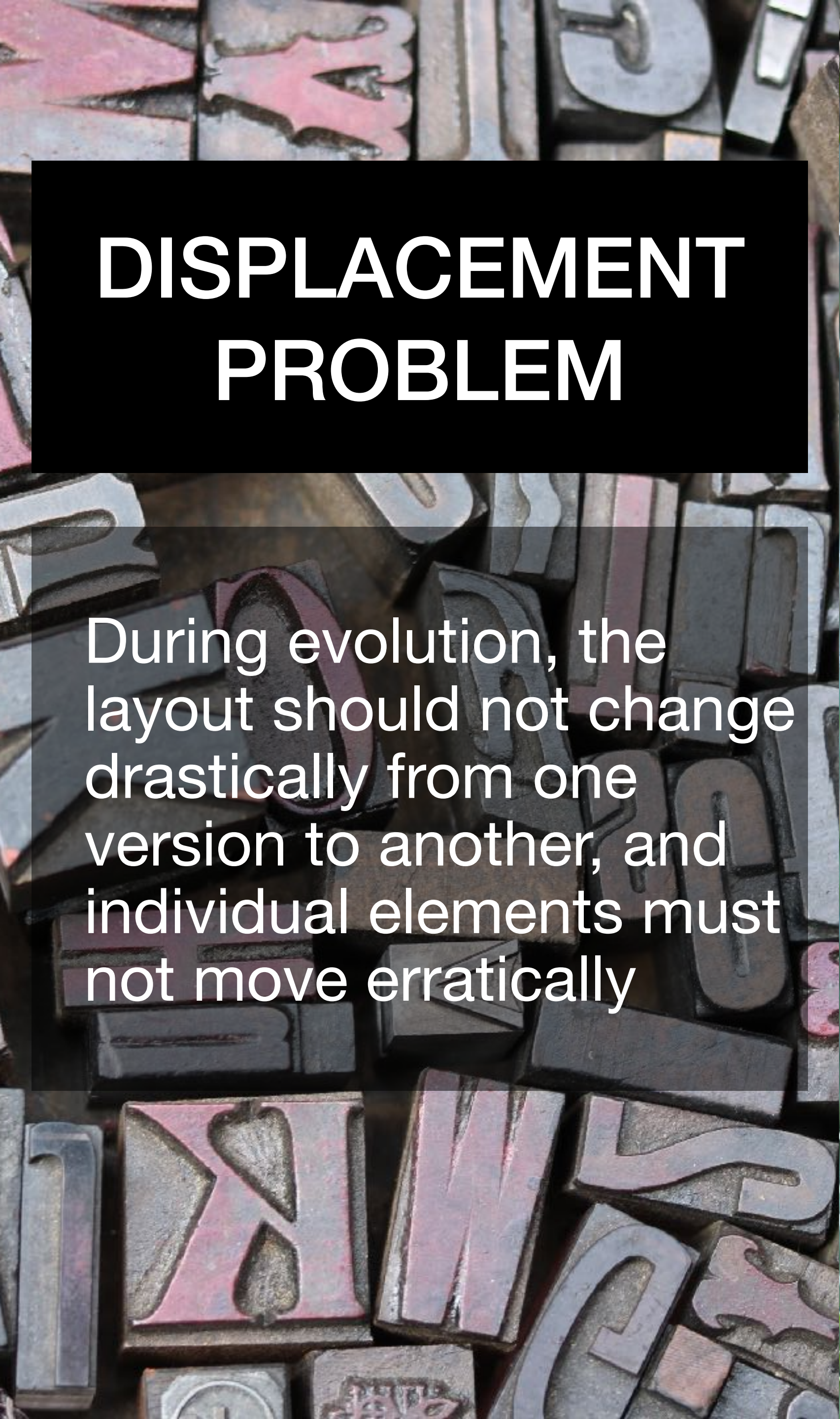


**DISPLACEMENT
PROBLEM**

**SCALABILITY
PROBLEM**

**COLLISION
PROBLEM**



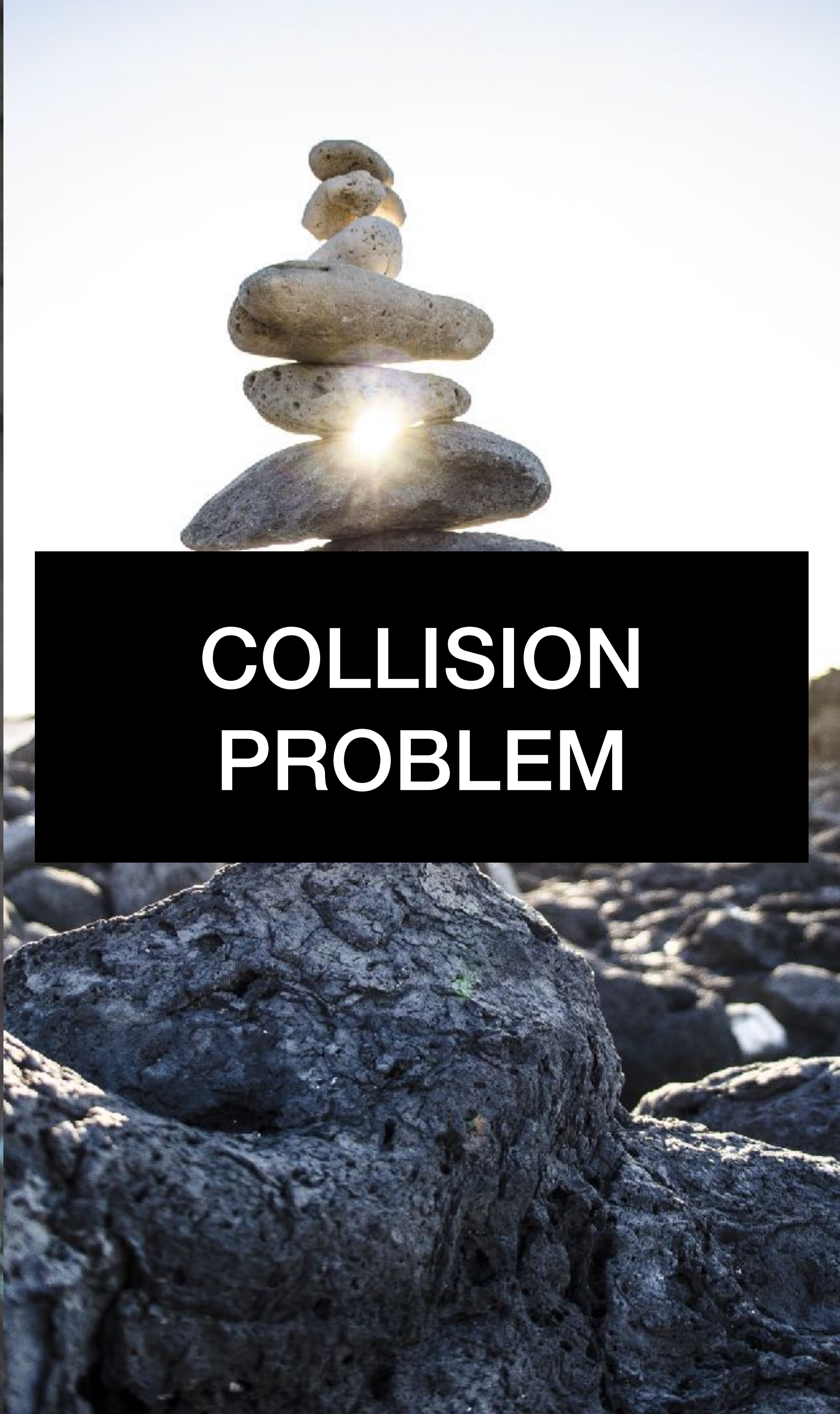


DISPLACEMENT PROBLEM

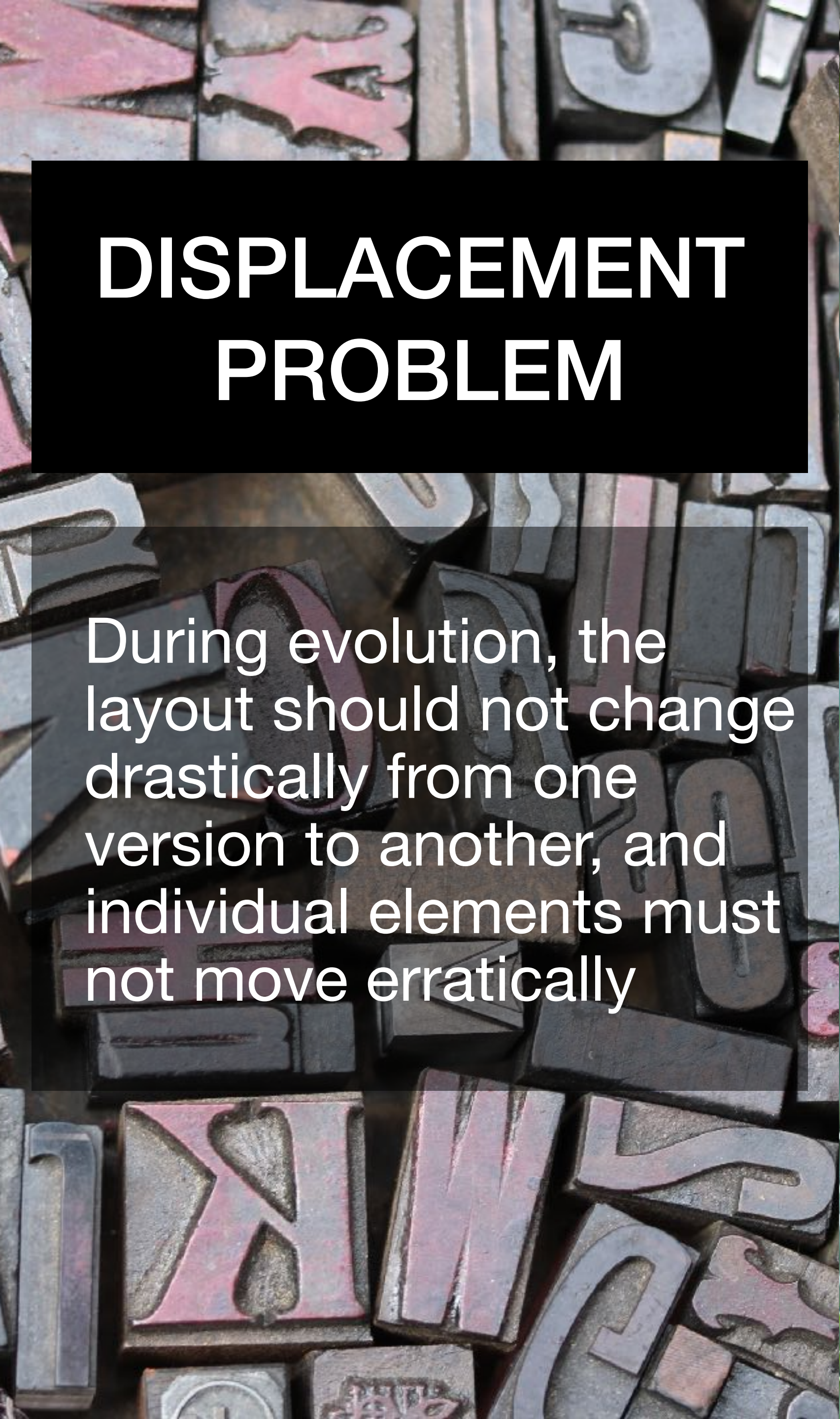
During evolution, the layout should not change drastically from one version to another, and individual elements must not move erratically



SCALABILITY PROBLEM

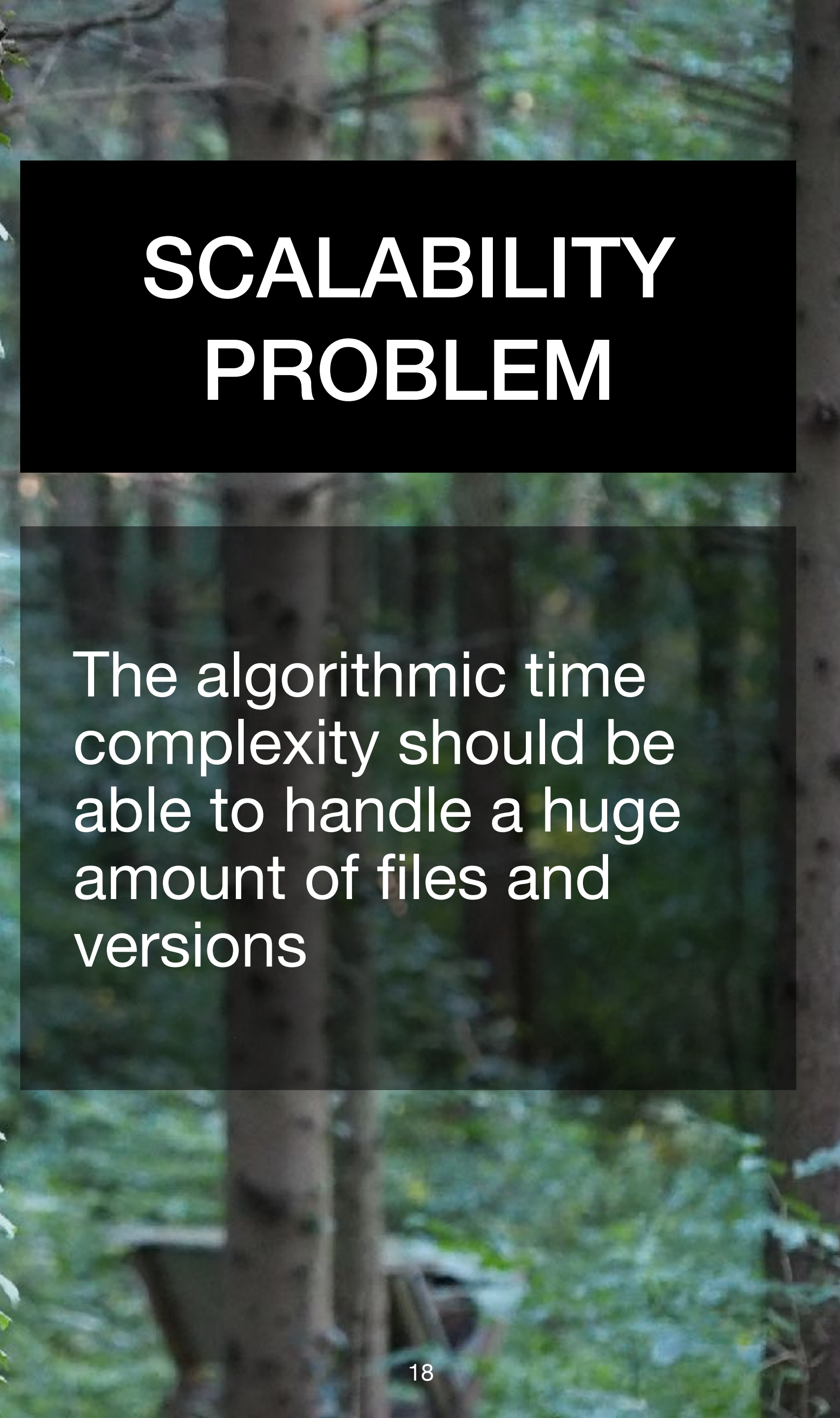


COLLISION PROBLEM



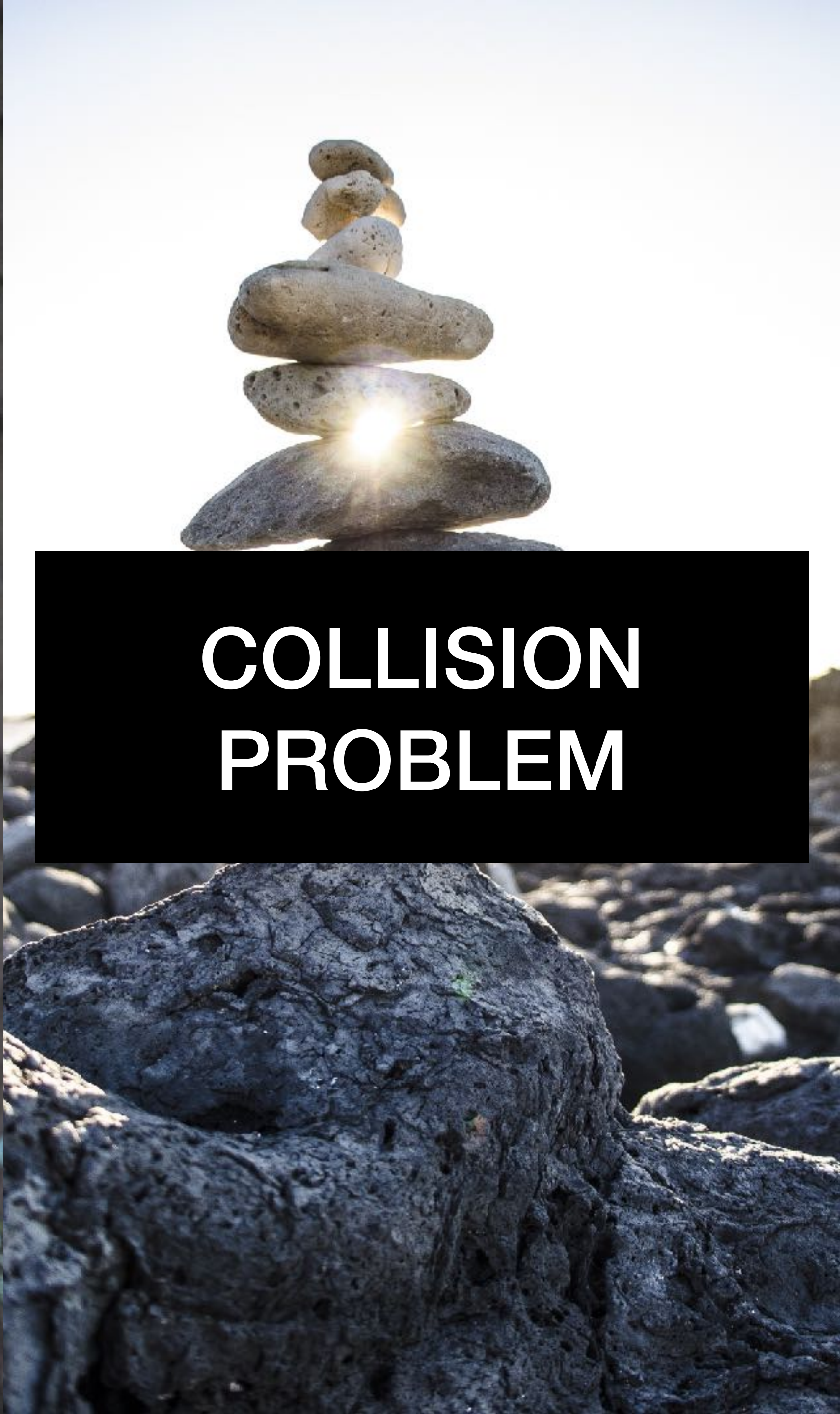
DISPLACEMENT PROBLEM

During evolution, the layout should not change drastically from one version to another, and individual elements must not move erratically

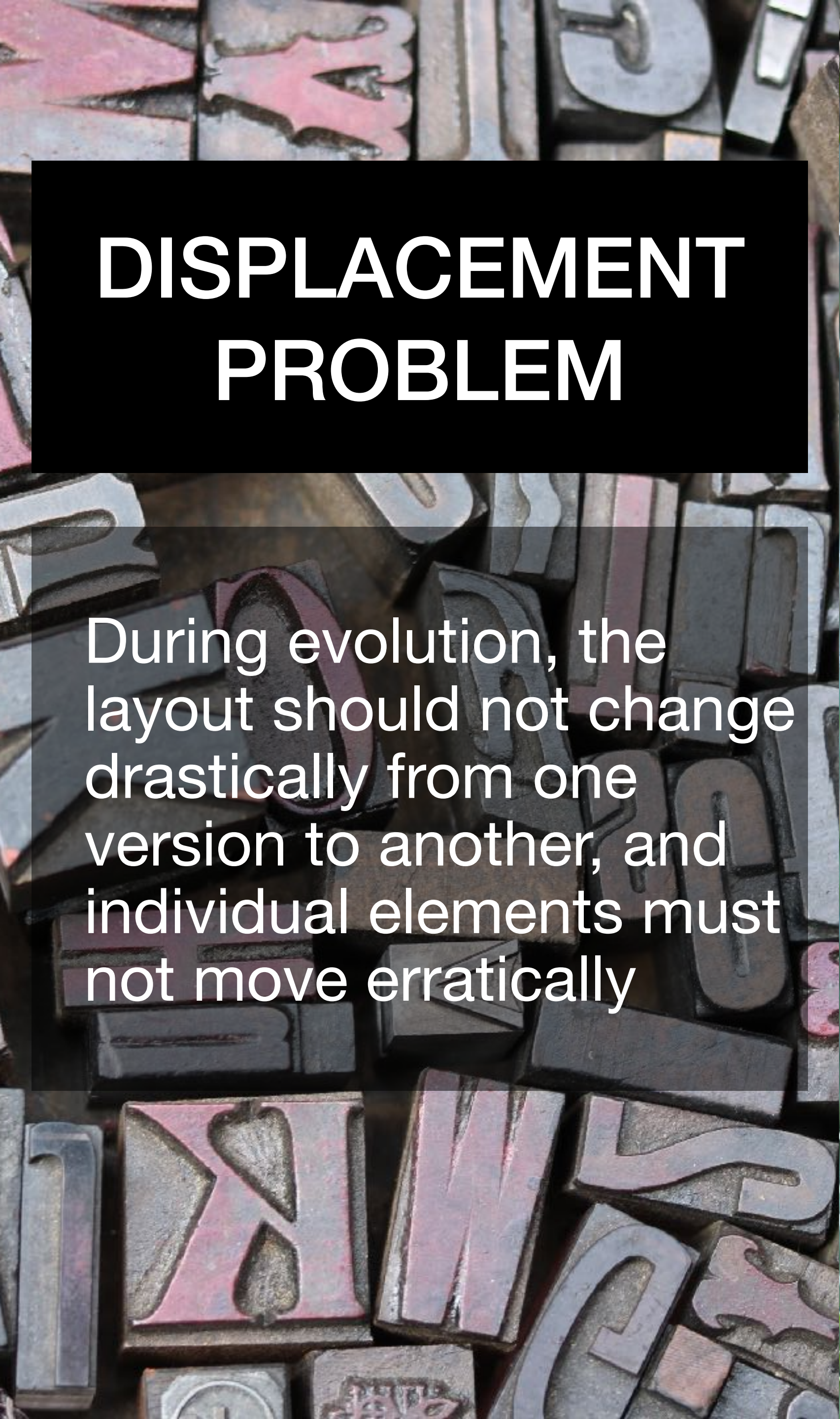


SCALABILITY PROBLEM

The algorithmic time complexity should be able to handle a huge amount of files and versions

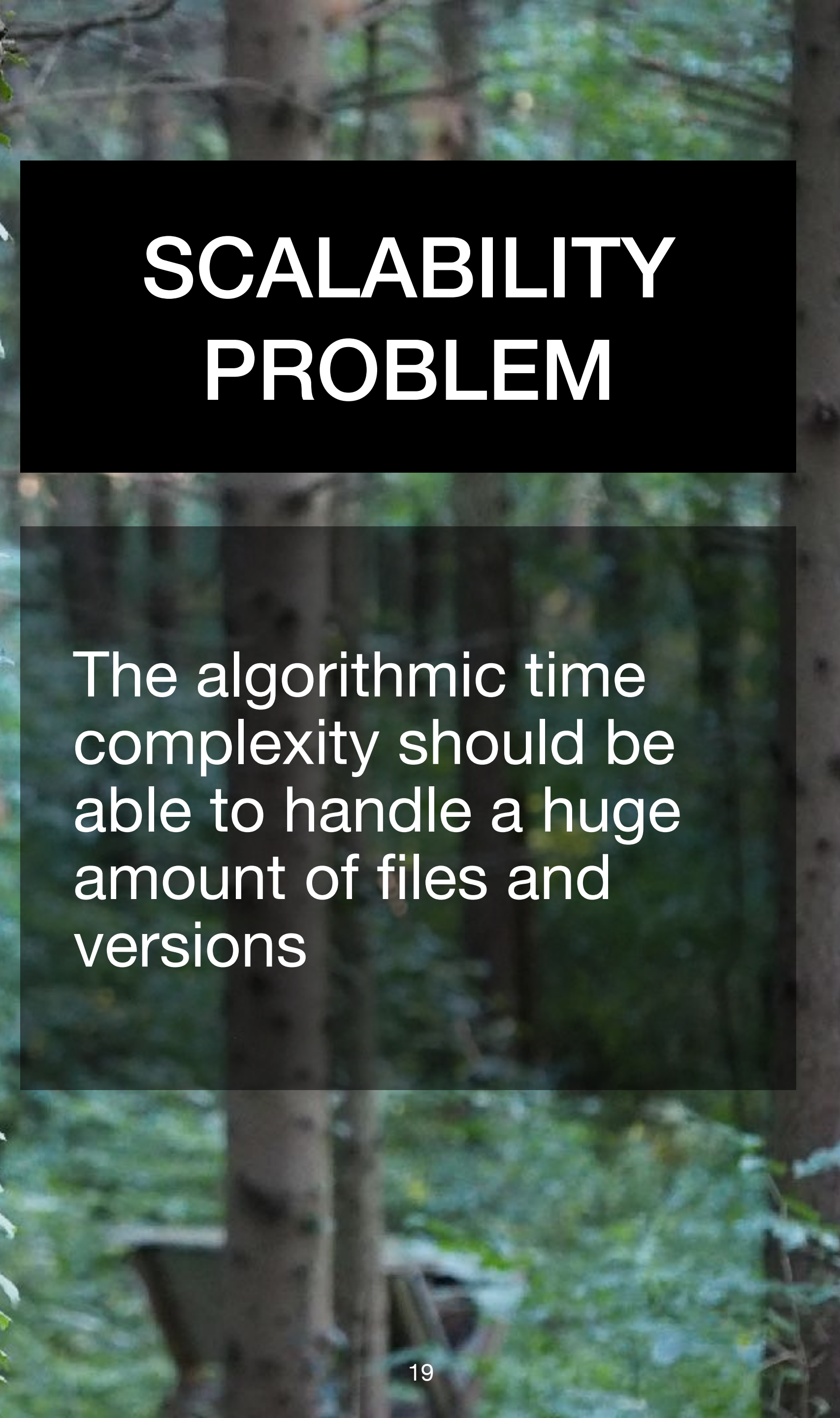


COLLISION PROBLEM



DISPLACEMENT PROBLEM

During evolution, the layout should not change drastically from one version to another, and individual elements must not move erratically



SCALABILITY PROBLEM

The algorithmic time complexity should be able to handle a huge amount of files and versions



COLLISION PROBLEM

Must avoid making collisions between visual elements, to maintain cohesiveness of the individual elements

POISSON DISK SAMPLING

TO THE RESCUE!

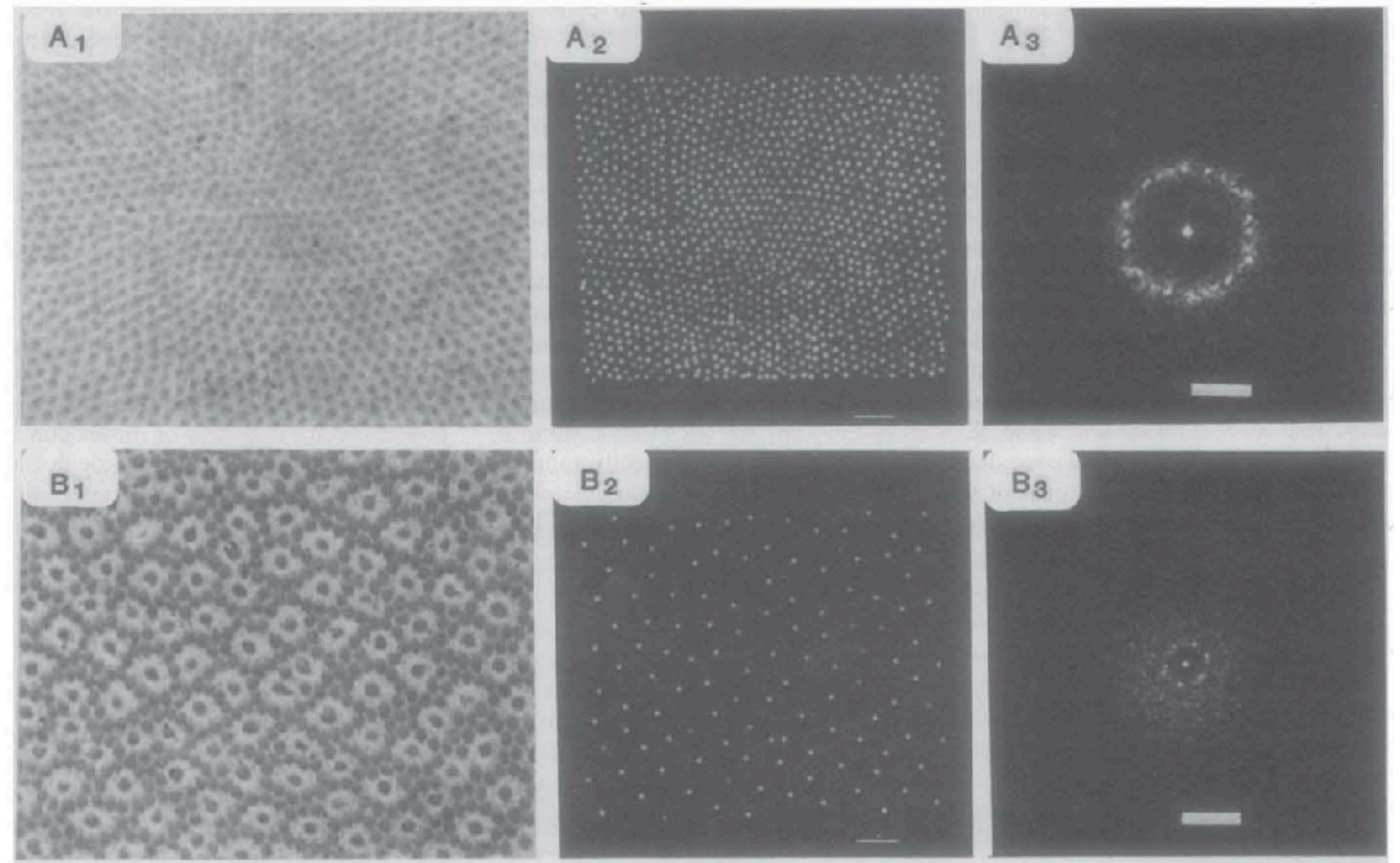


What is Poisson Disk Sampling

- “Sampling” at its core is about selecting some individuals from a large population
- We want to sample a position for each visual element so that they fill some space
- Space must be reserved for one element and cannot be assigned to other elements

What is Poisson Disk Sampling

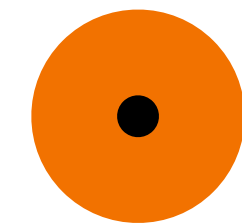
- A technique for sampling disks of fixed radius from a plane
- Inspired by the placement of photoreceptors in the retina
- It is frequently used in computer graphics for anti-aliasing



J. I. Yellott Jr, "Spectral consequences of photoreceptor sampling in the rhesus retina," Science, 1983.

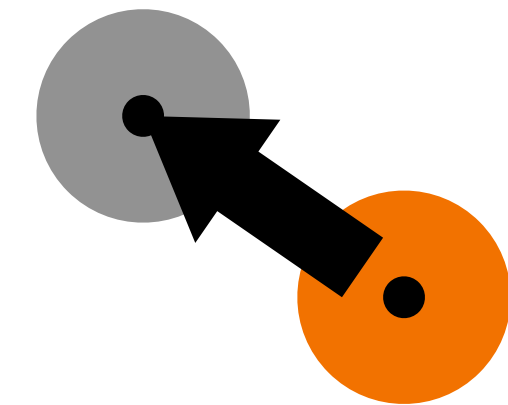
How Poisson Disk Sampling works

- Begin at a known starting position (**active point**)



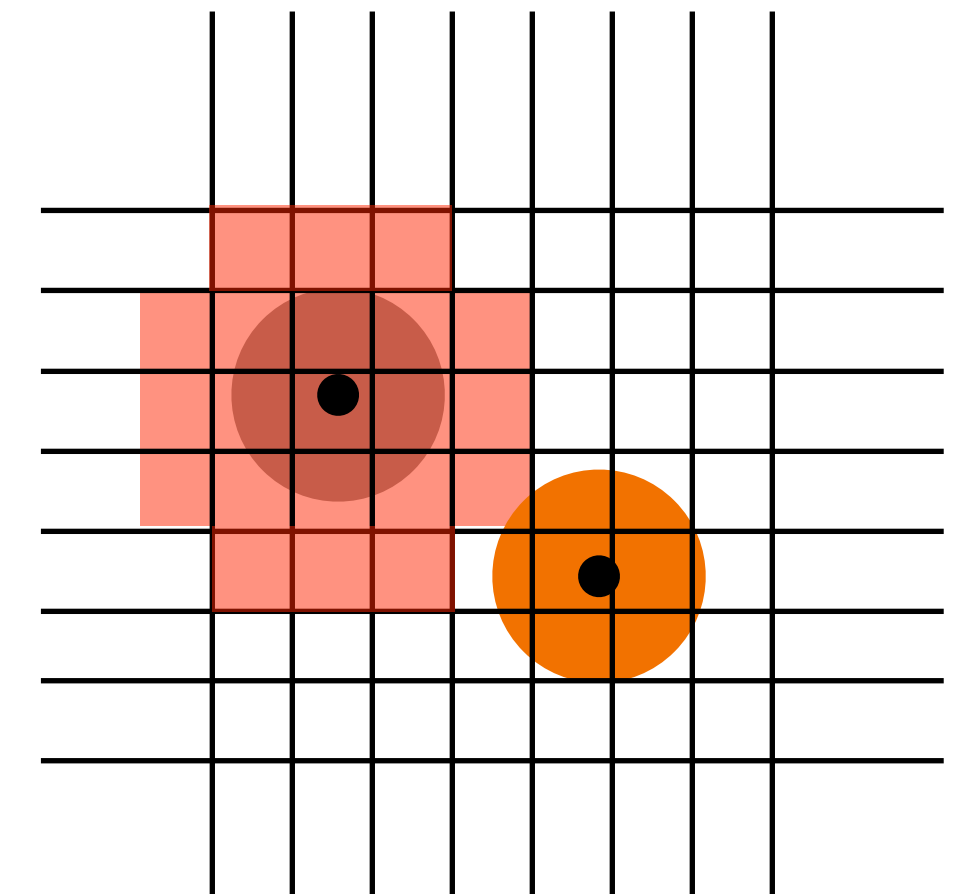
How Poisson Disk Sampling works

- Begin at a known starting position (active point)
- If there is no collision with existing points, sample a point offset from the active point by a random direction and distance



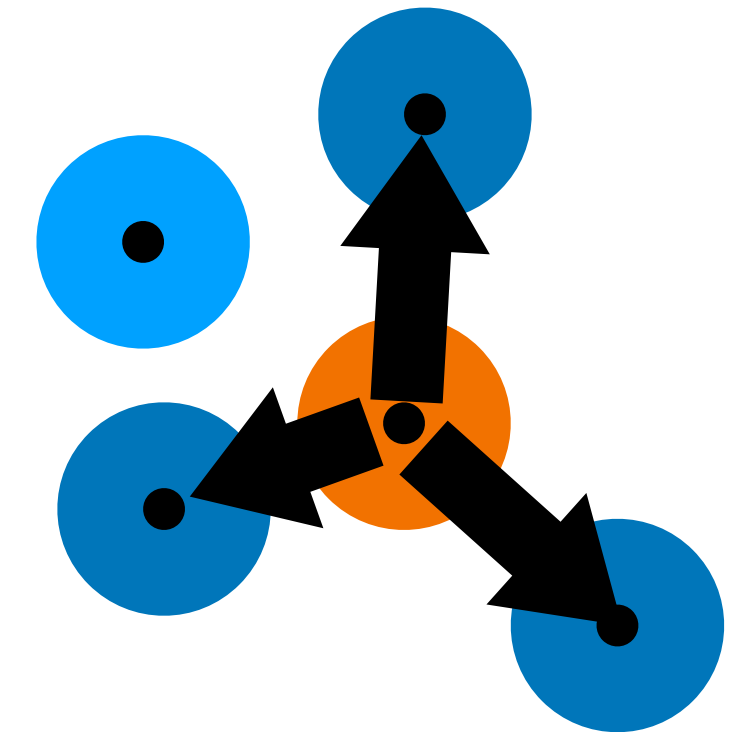
How Poisson Disk Sampling works

- Begin at a known starting position (active point)
- If there is no collision with existing points, sample a point offset from the active point by a random direction and distance
 - Use an underlying grid data structure to store existing disks and speedup collision detection



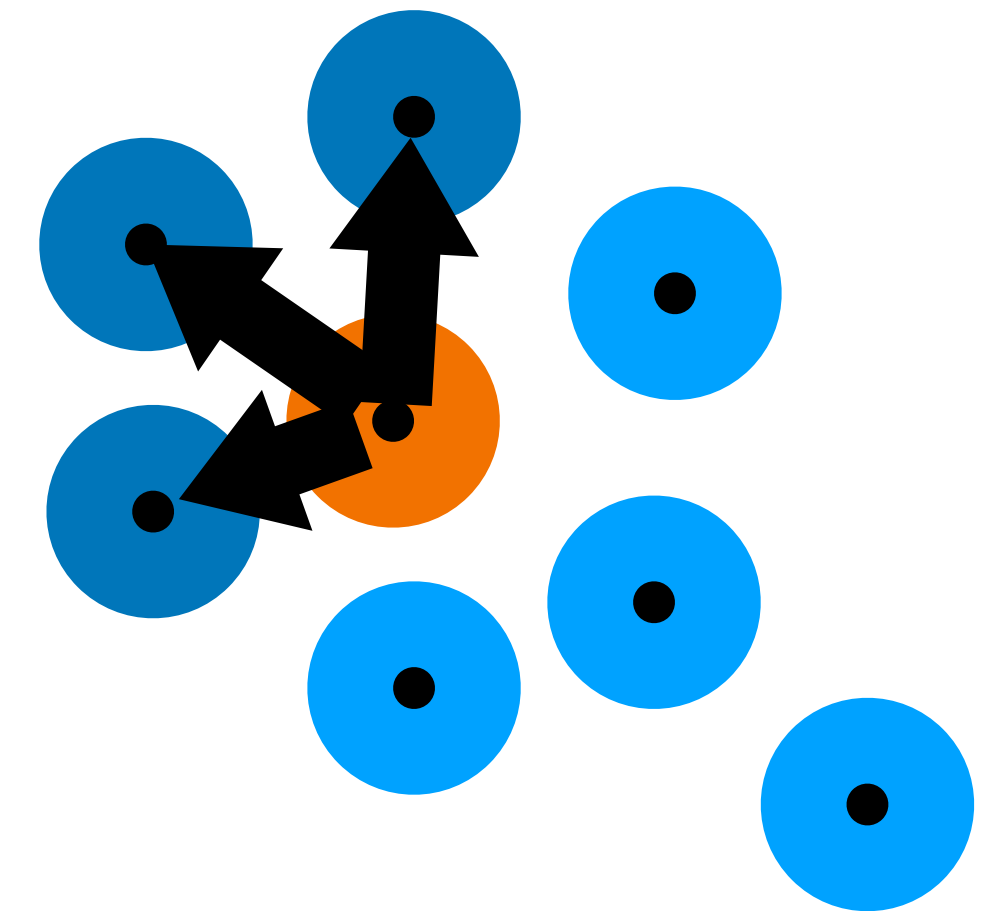
How Poisson Disk Sampling works

- Begin at a known starting position (active point)
- If there is no collision with existing points, sample a point offset from the active point by a random direction and distance
- Repeat k times and add all sampled points to **active queue**



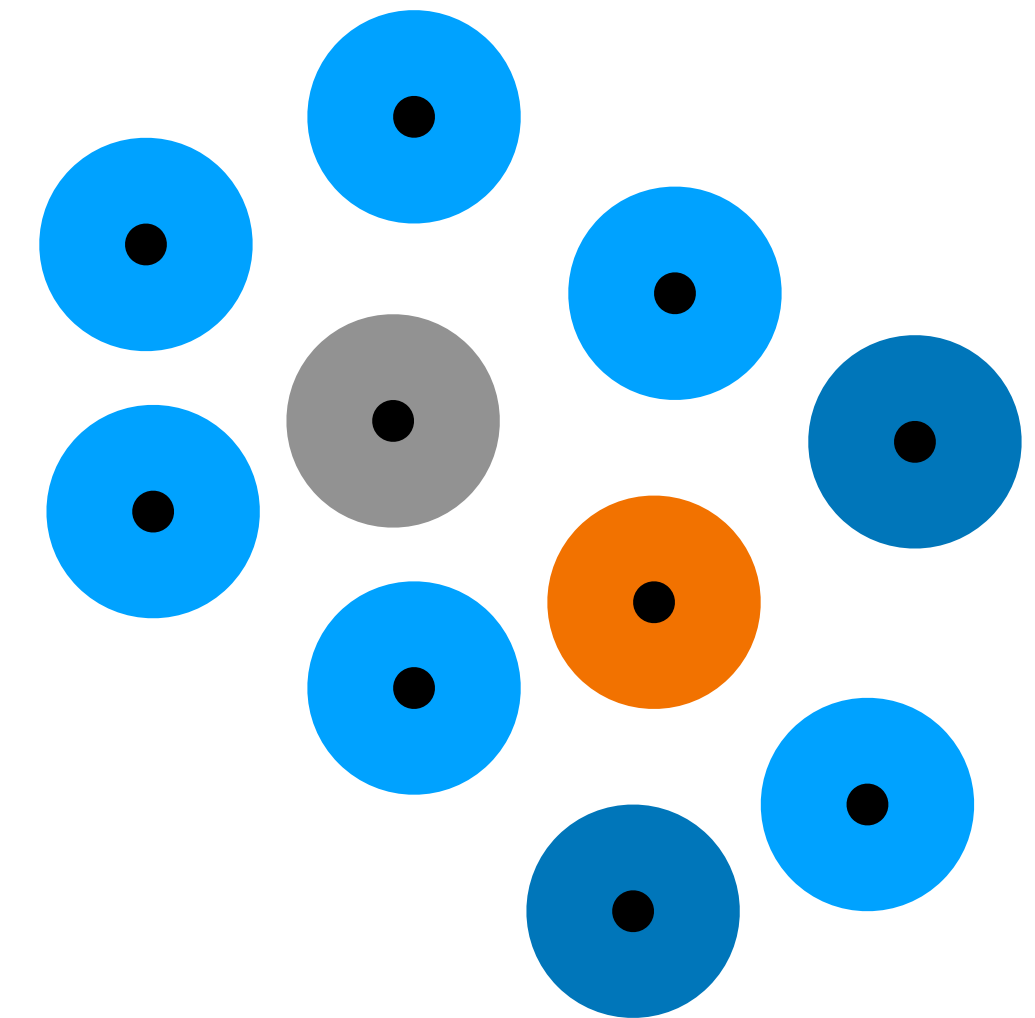
How Poisson Disk Sampling works

- Begin at a known starting position (active point)
- If there is no collision with existing points, sample a point offset from the active point by a random direction and distance
- Repeat k times and add all sampled points to active queue
- Put the current active point in the queue, choose a new **active point** from the queue and repeat



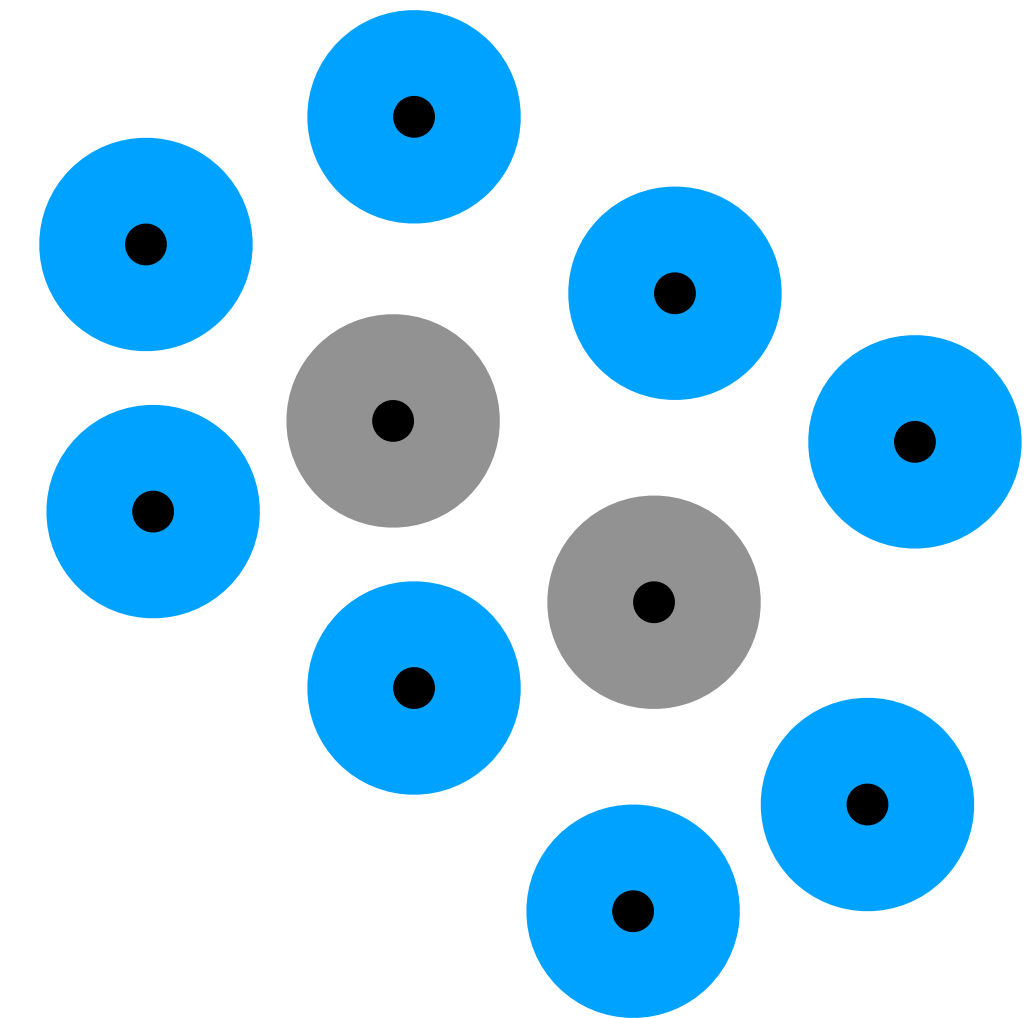
How Poisson Disk Sampling works

- If all k attempts at sampling fail, remove the active point from the active queue



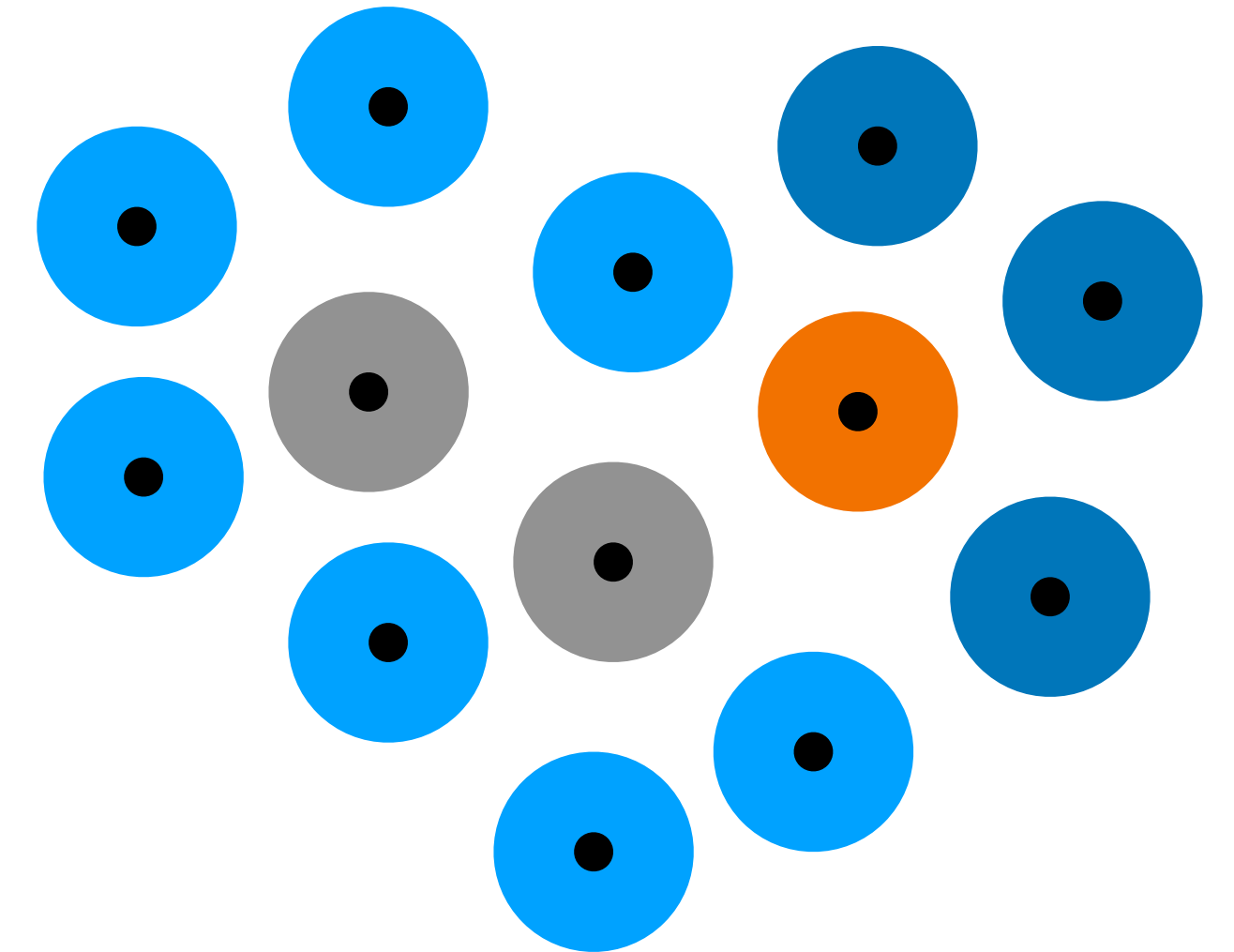
How Poisson Disk Sampling works

- If all k attempts at sampling fail, remove the active point from the active queue
- Stop when enough points have been sampled, or there are no more points in the active queue



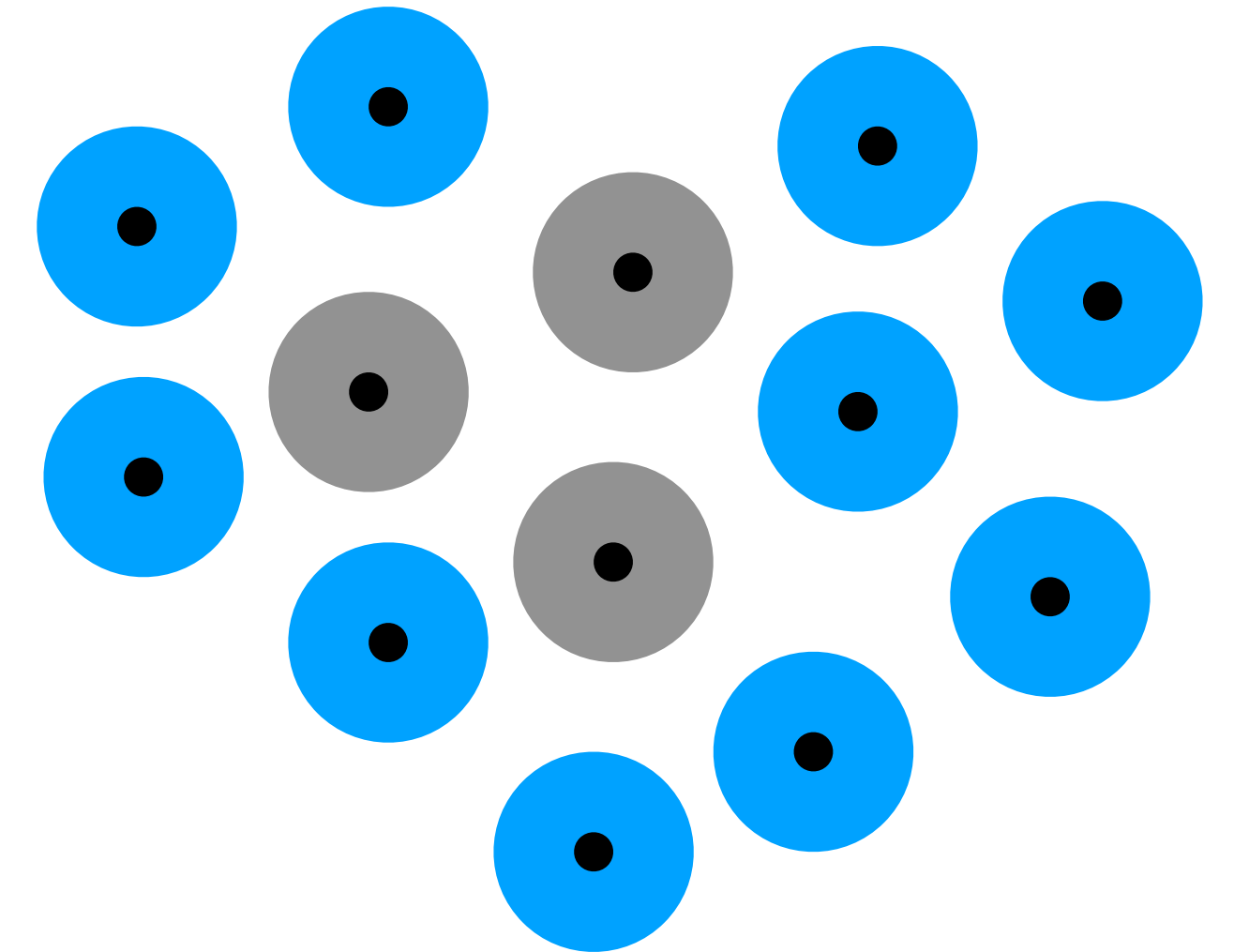
How Poisson Disk Sampling works

- If all k attempts at sampling fail, remove the active point from the active queue
- Stop when enough points have been sampled, or there are no more points in the active queue



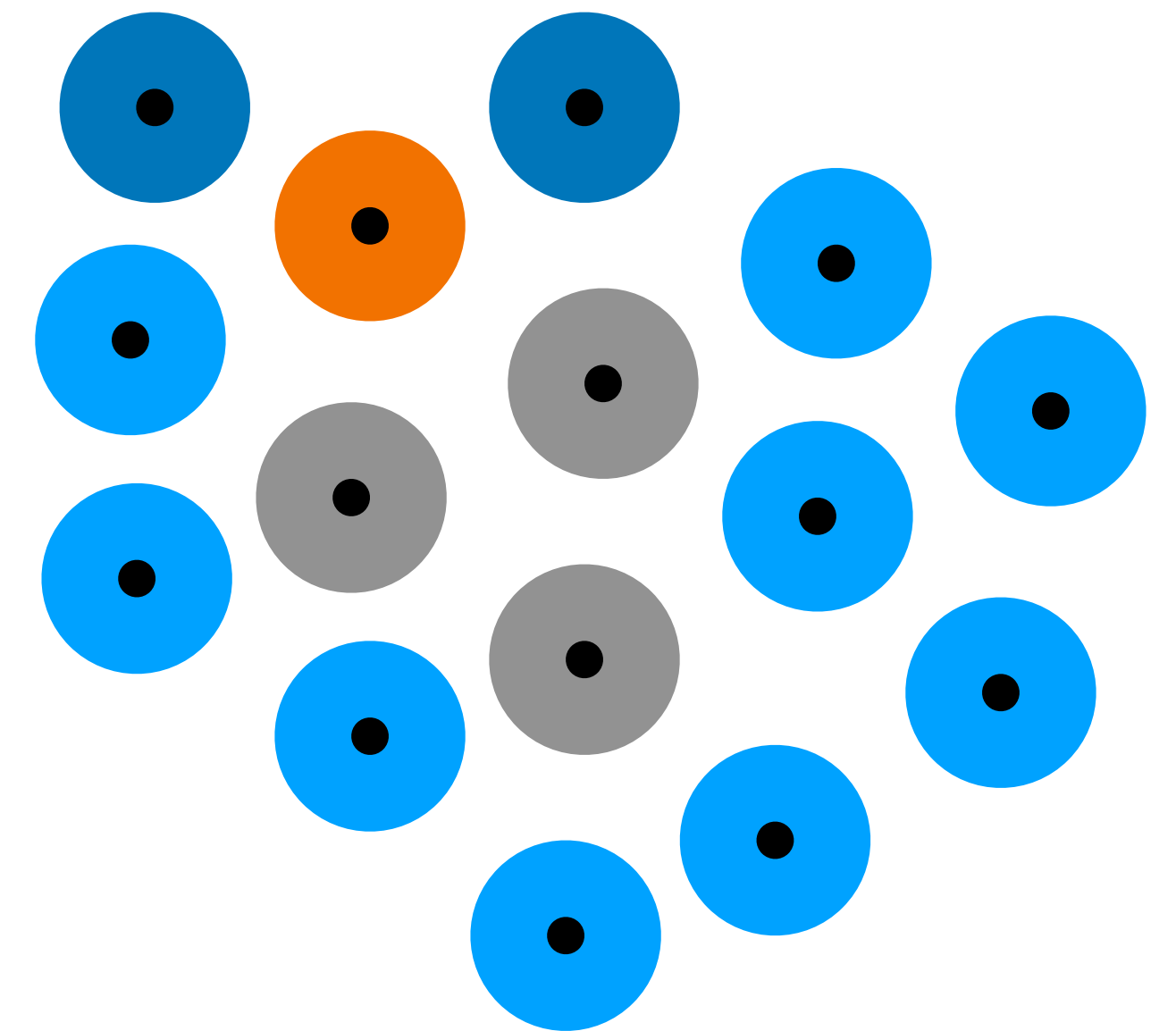
How Poisson Disk Sampling works

- If all k attempts at sampling fail, remove the active point from the active queue
- Stop when enough points have been sampled, or there are no more points in the active queue



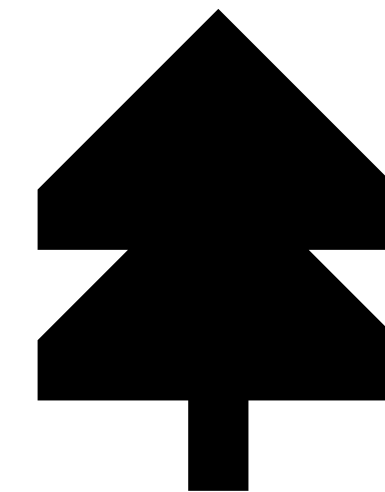
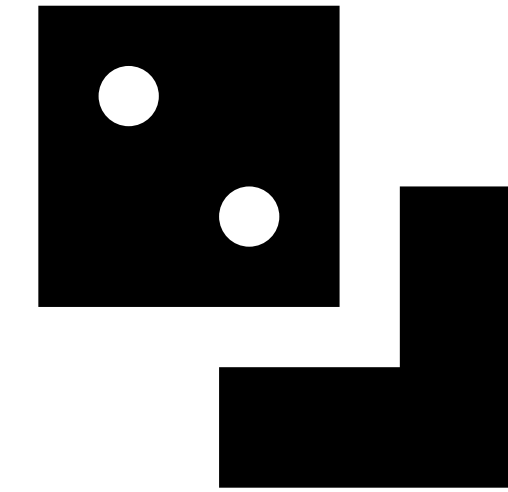
How Poisson Disk Sampling works

- If all k attempts at sampling fail, remove the active point from the active queue
- Stop when enough points have been sampled, or there are no more points in the active queue



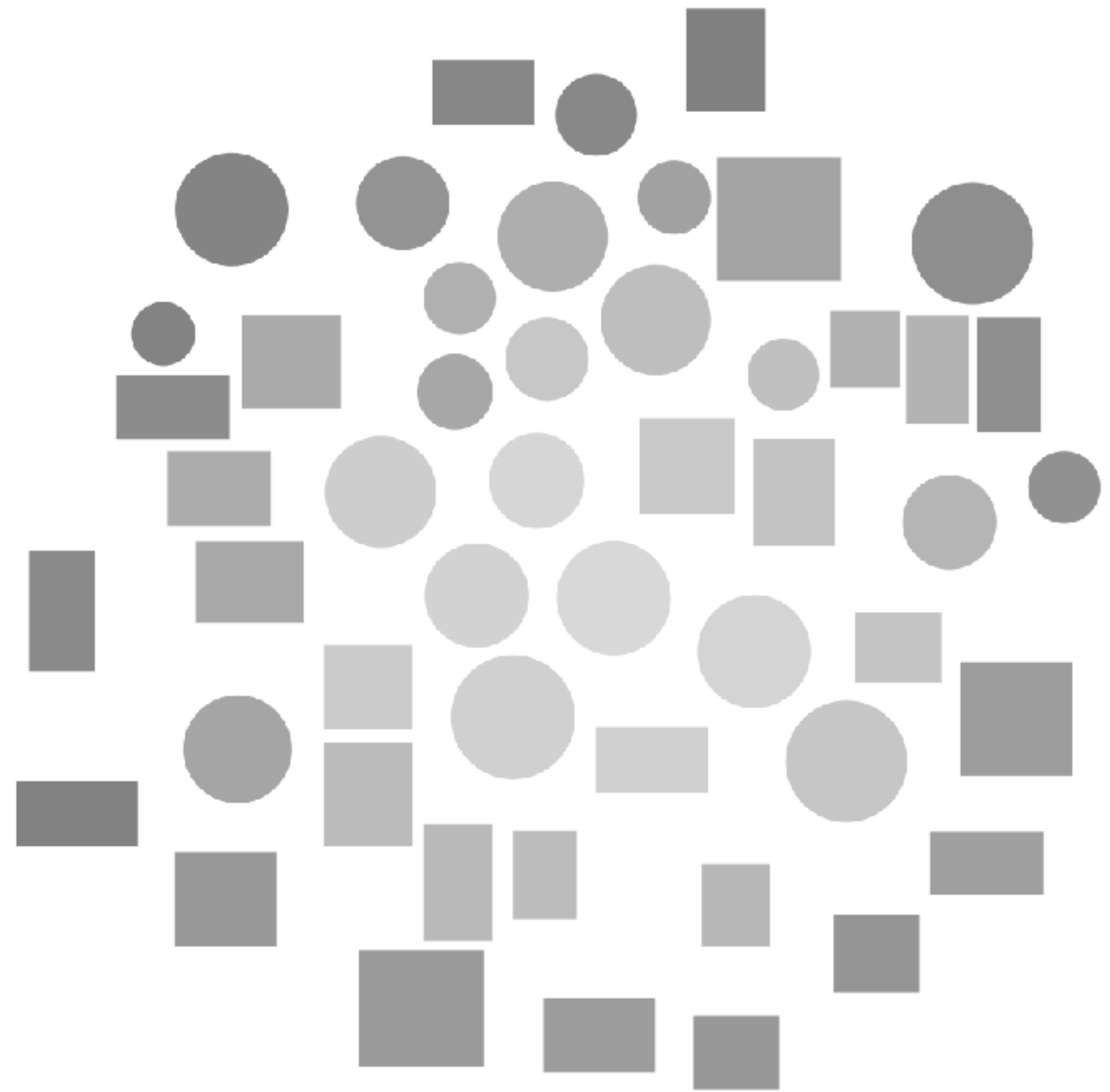
Improving Poisson Disk Sampling

- Remove the random component by fixing a seed, to achieve deterministic outputs
- Support for varying disk radius and shapes by implementing a more advanced data structure for collision checking



Improving Poisson Disk Sampling

- Remove the random component by fixing a seed, to achieve deterministic outputs
- Support for varying disk radius and shapes by implementing a more advanced data structure for collision checking

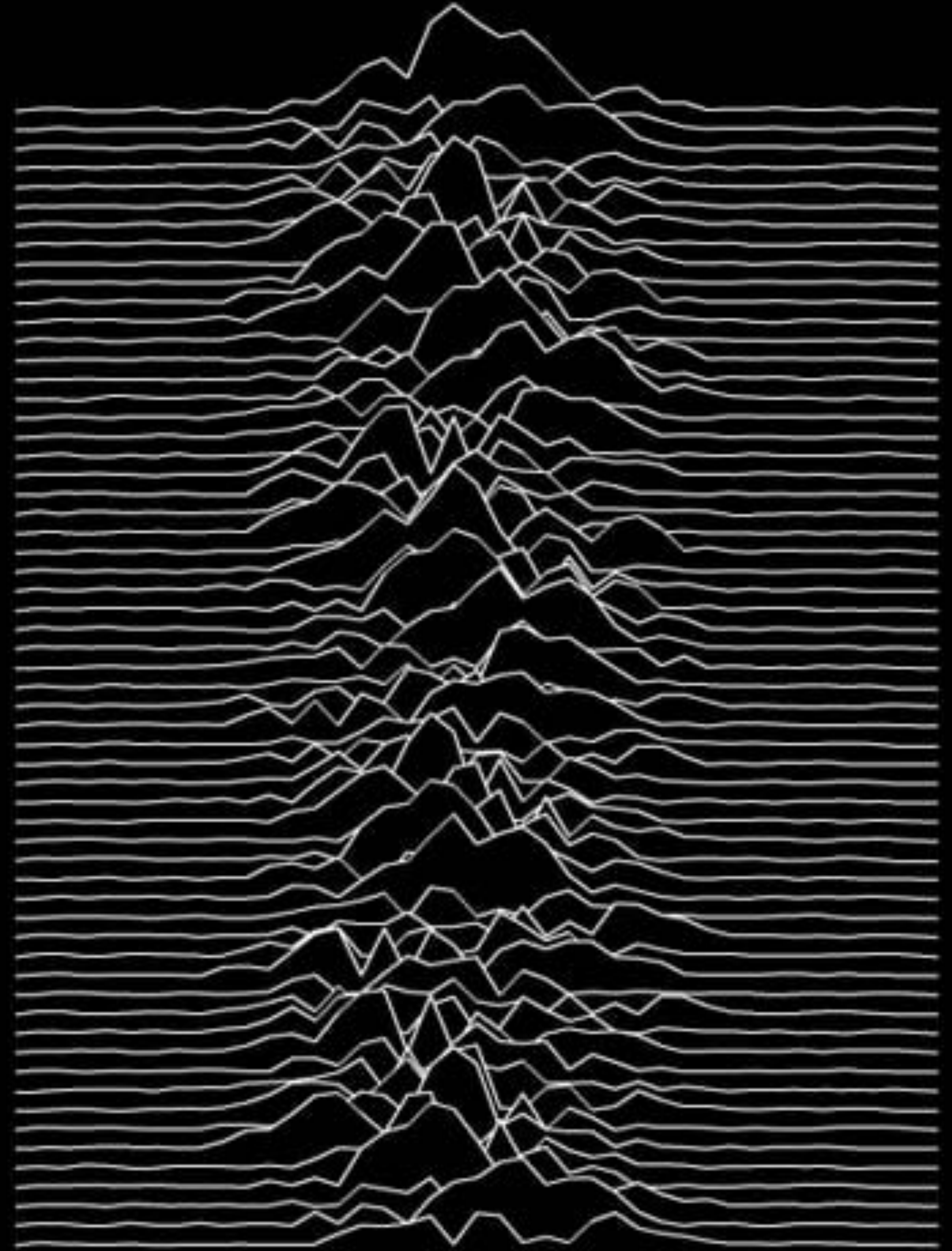


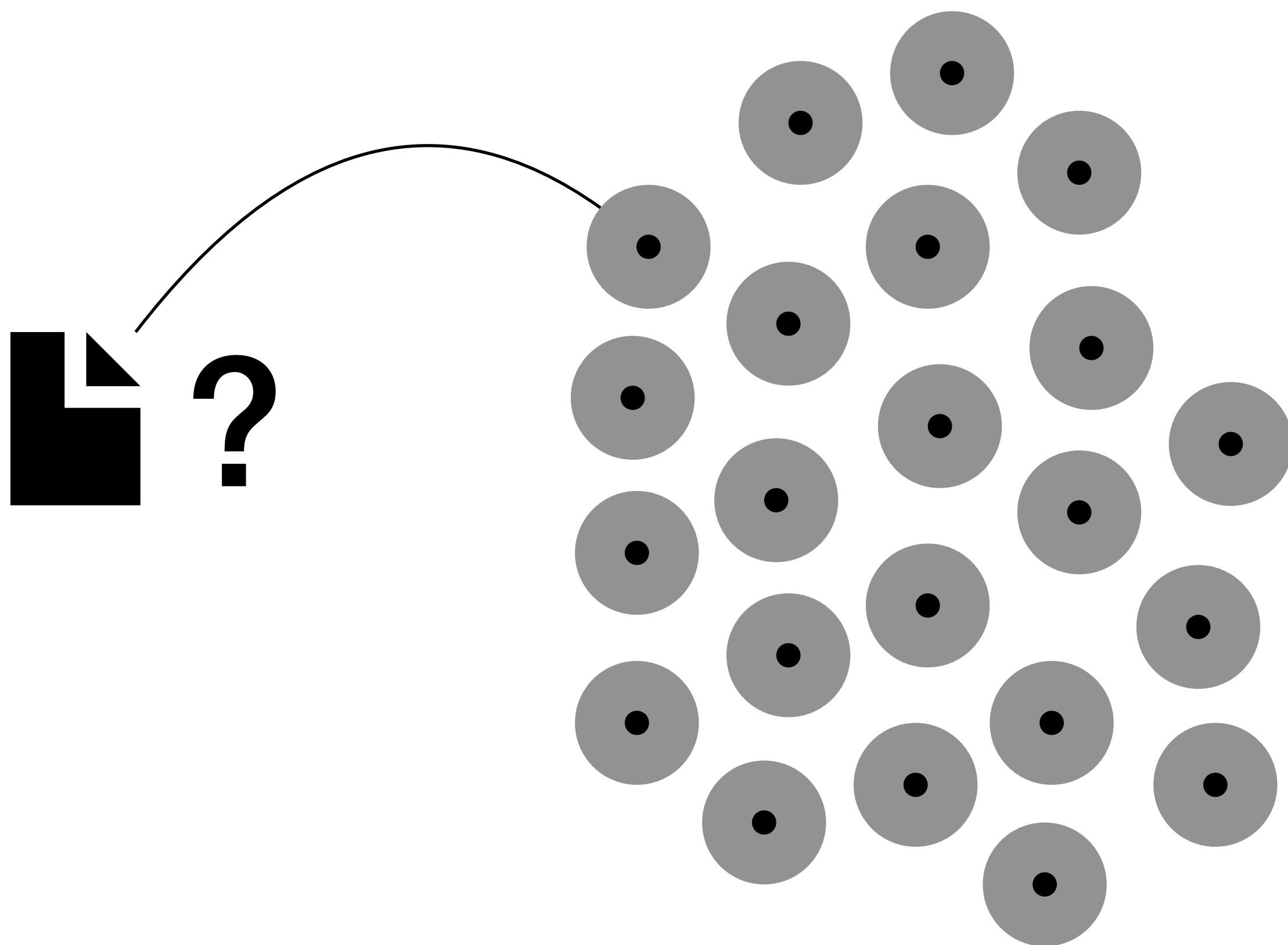
Implementing the Layout Algorithm in Pharo

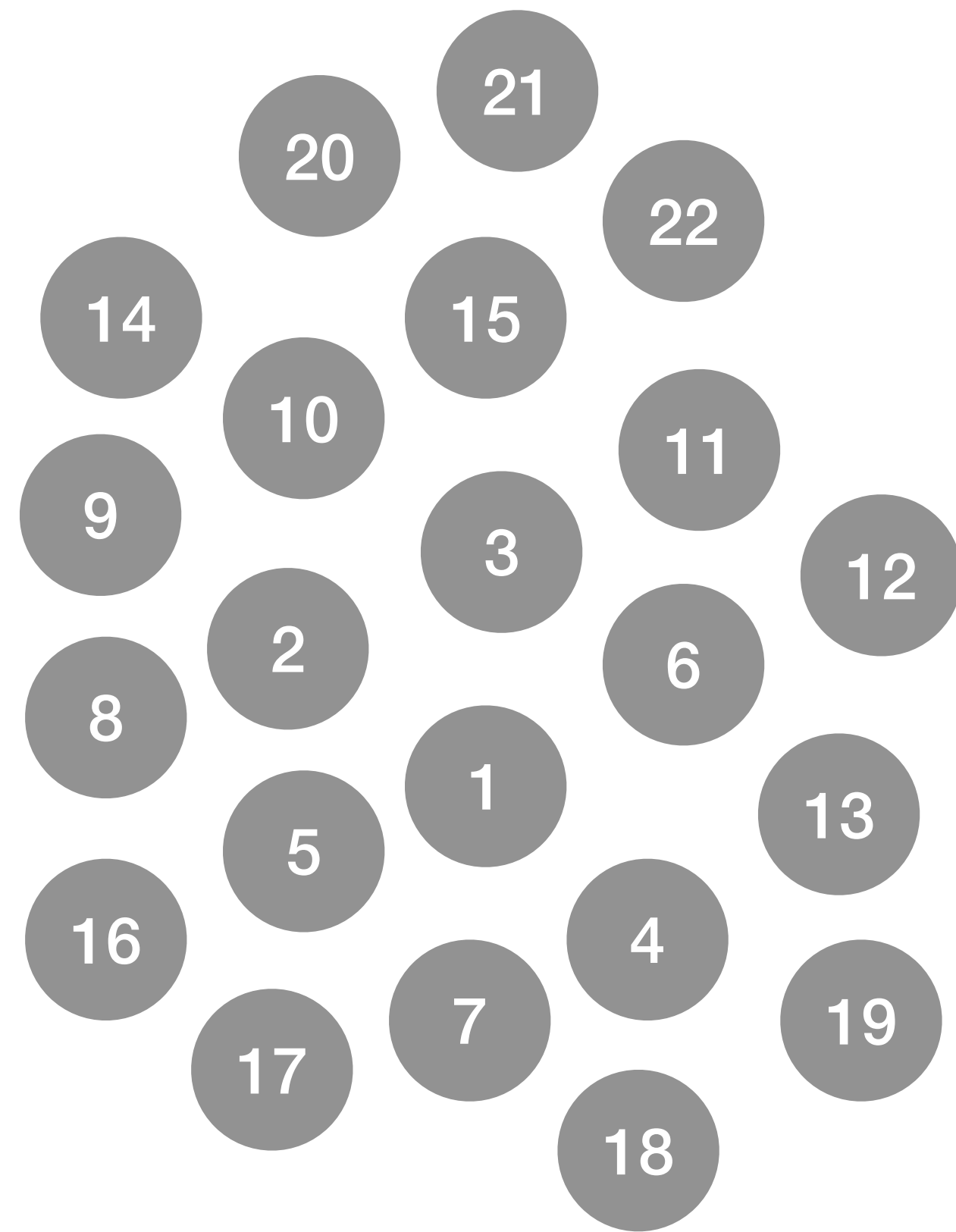
- Implemented in Pharo as a Roassal layout
- Works with standard shapes: rectangles and circles
- For non-standard shapes a bounding rectangle is used instead

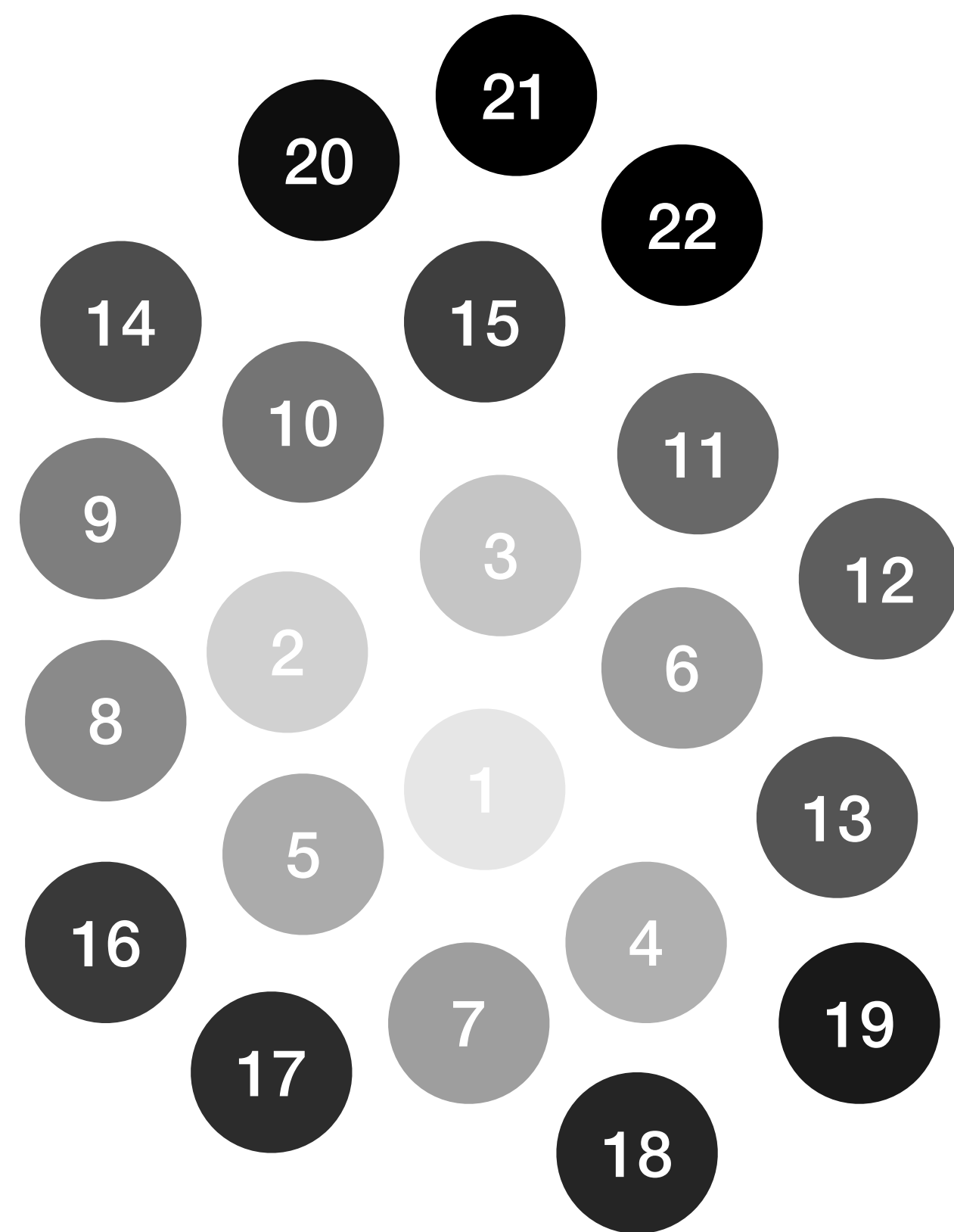
```
1 shapes := OrderedCollection new.  
2 50 timesRepeat: [ shapes add: RSBox new ].  
3 50 timesRepeat: [ shapes add: RSCircle new ].  
4  
5 layout := DVPoissonDiscLayout new  
6     margin: 2;  
7     distanceVariance: 2;  
8     seed: 1;  
9     collision: DVTreeCollision new;  
10    on: shapes.  
11  
12 canvas := RSCanvas new addAll: shapes.  
13 canvas zoomToFit open.
```

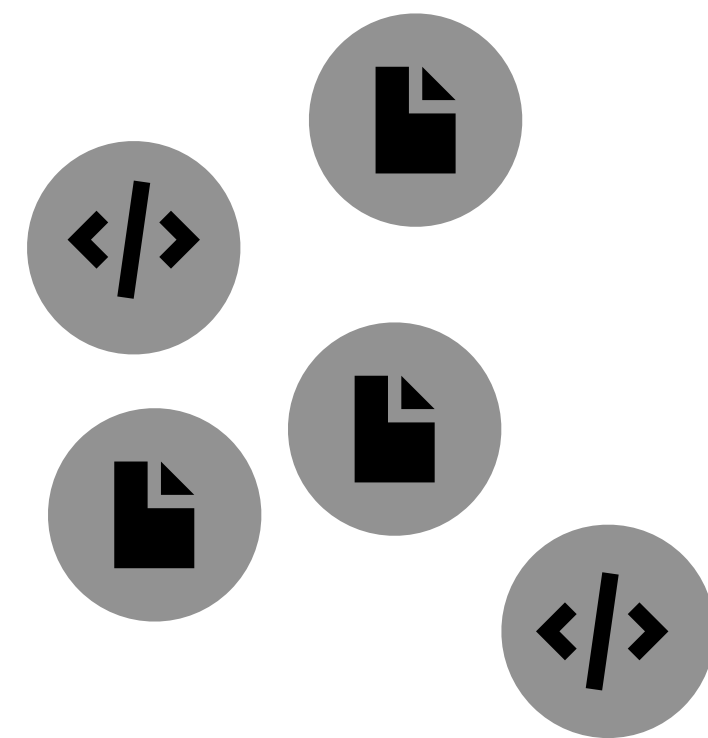
INTRODUCING DIVISION

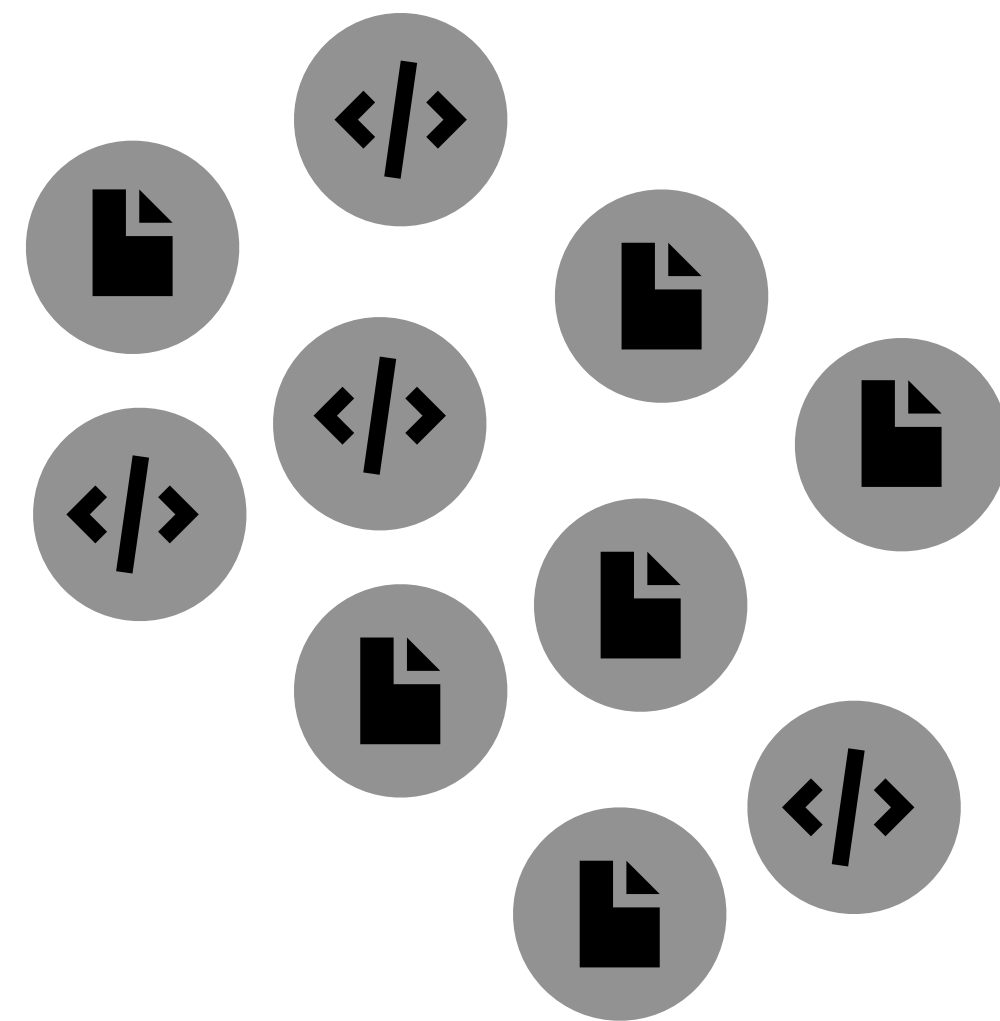


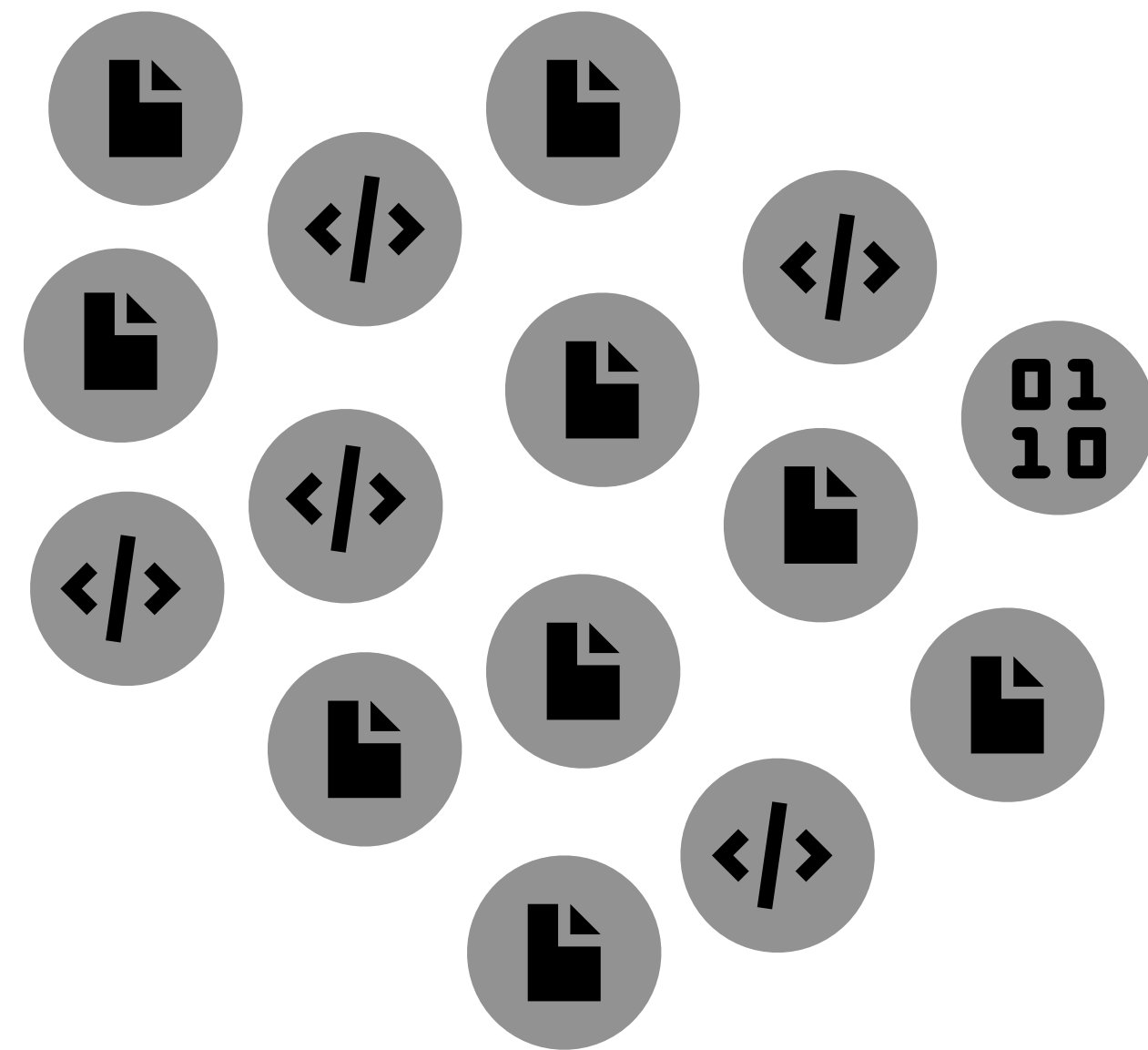


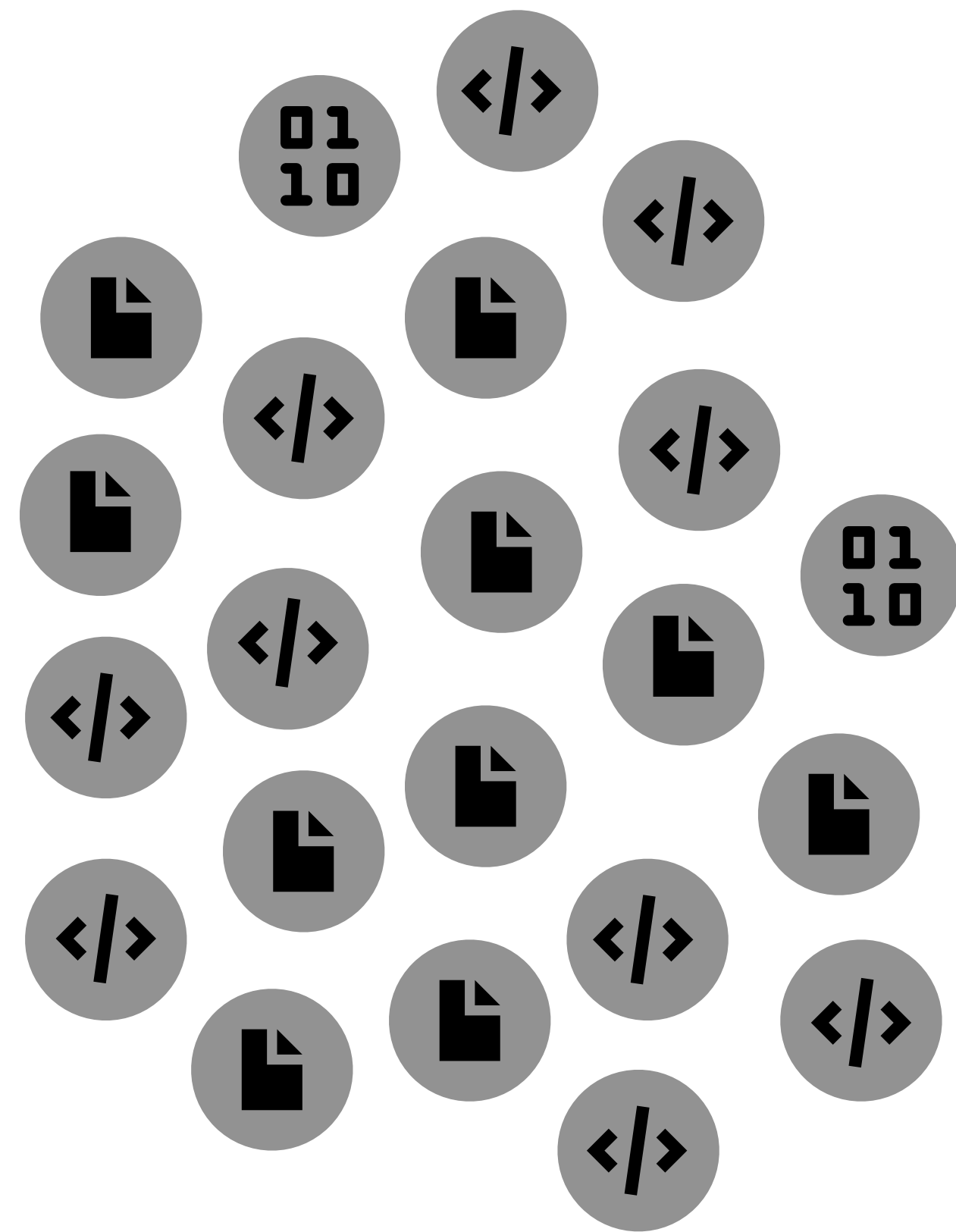


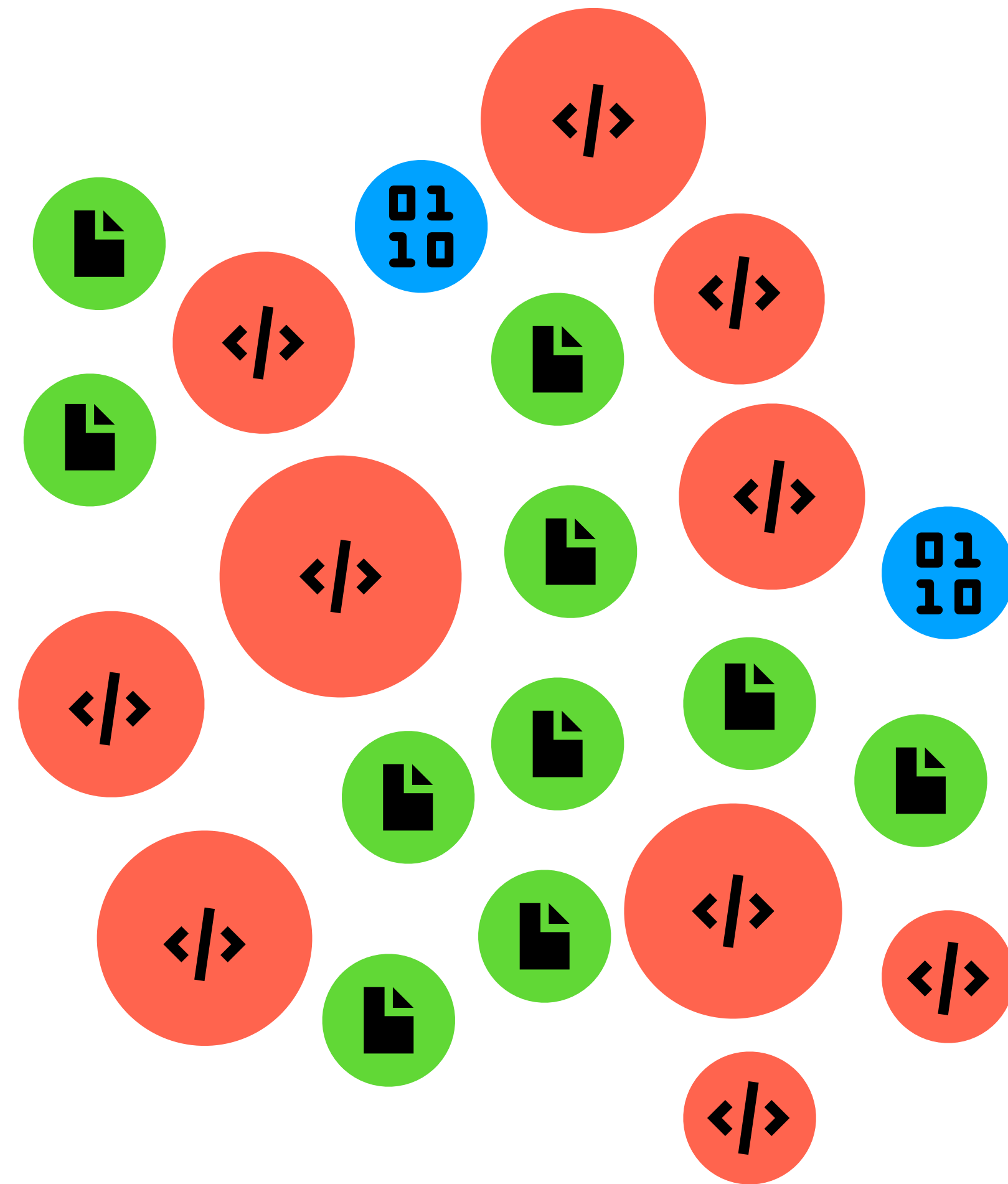


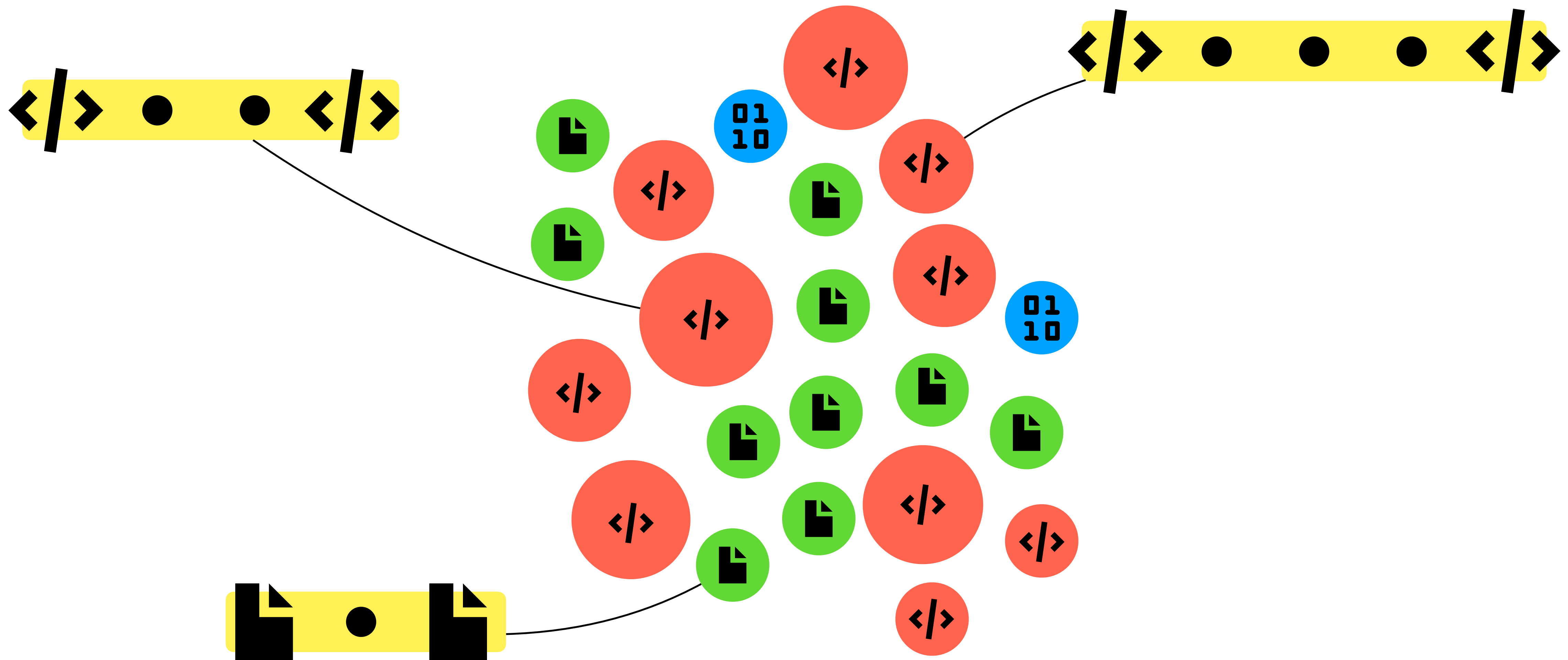


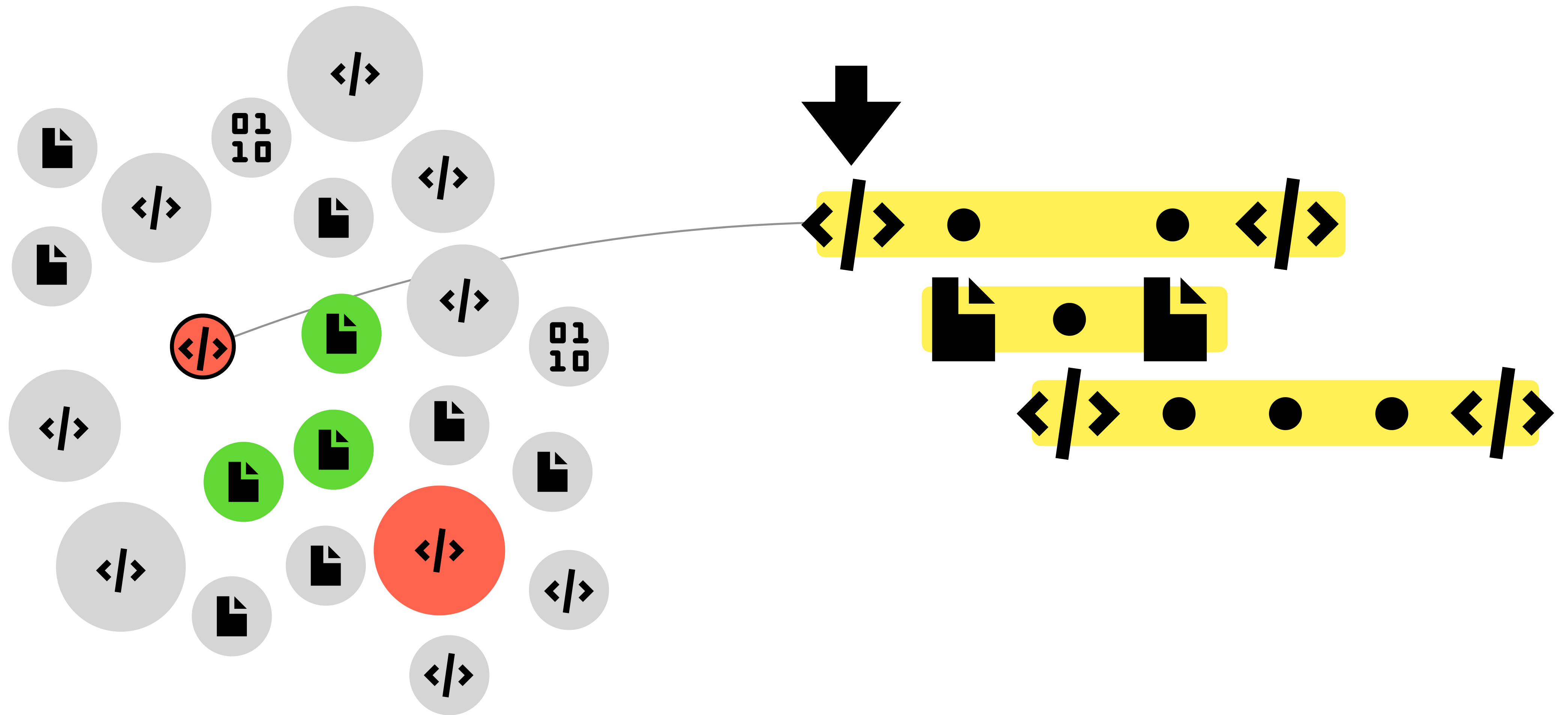


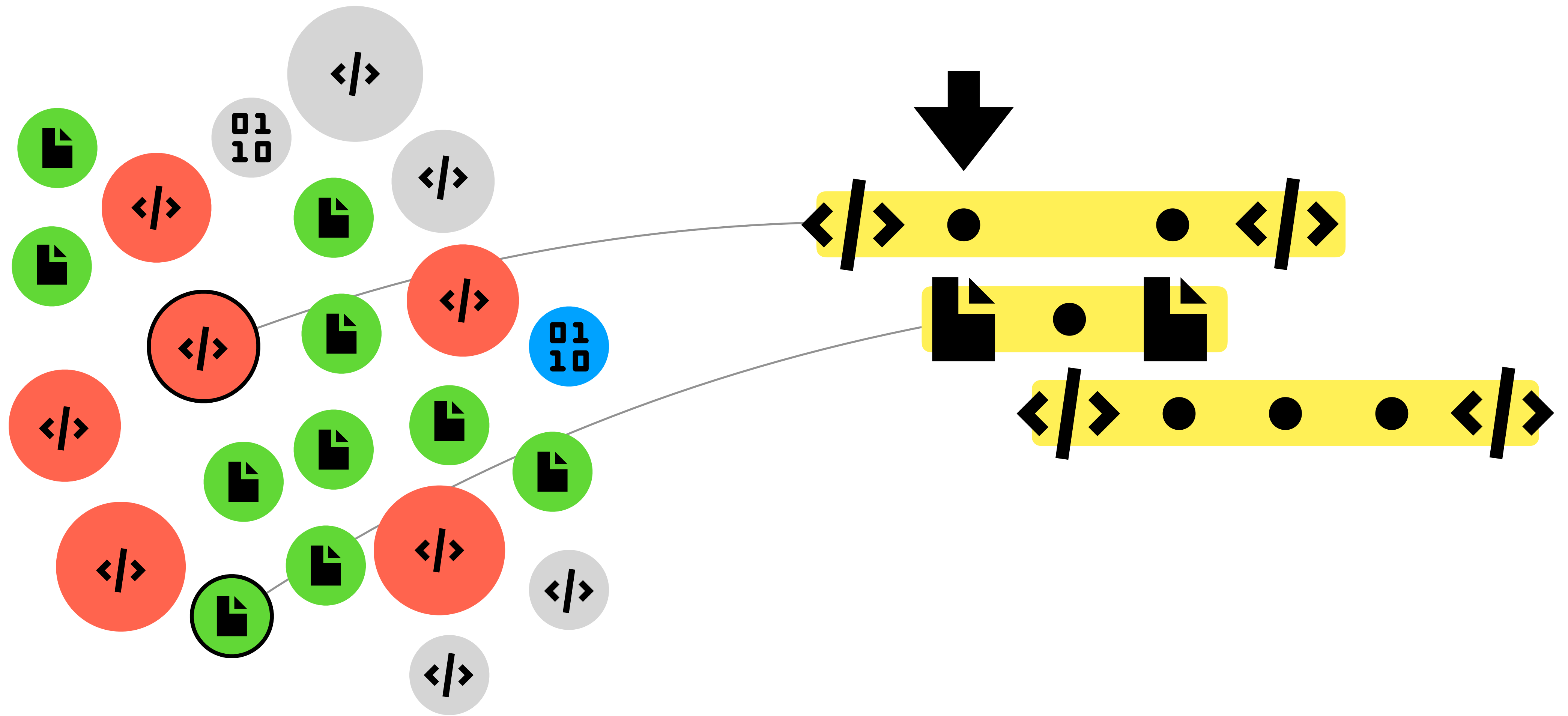


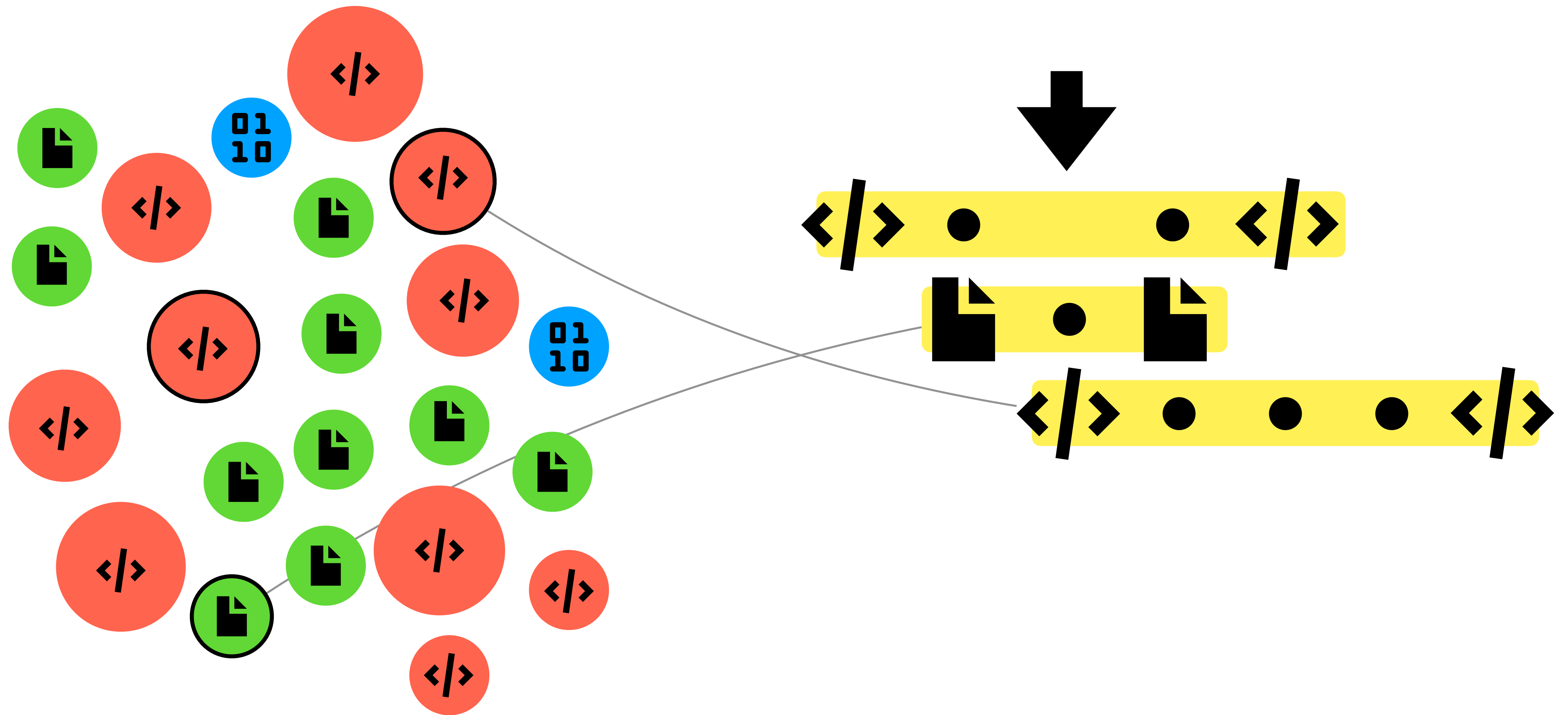


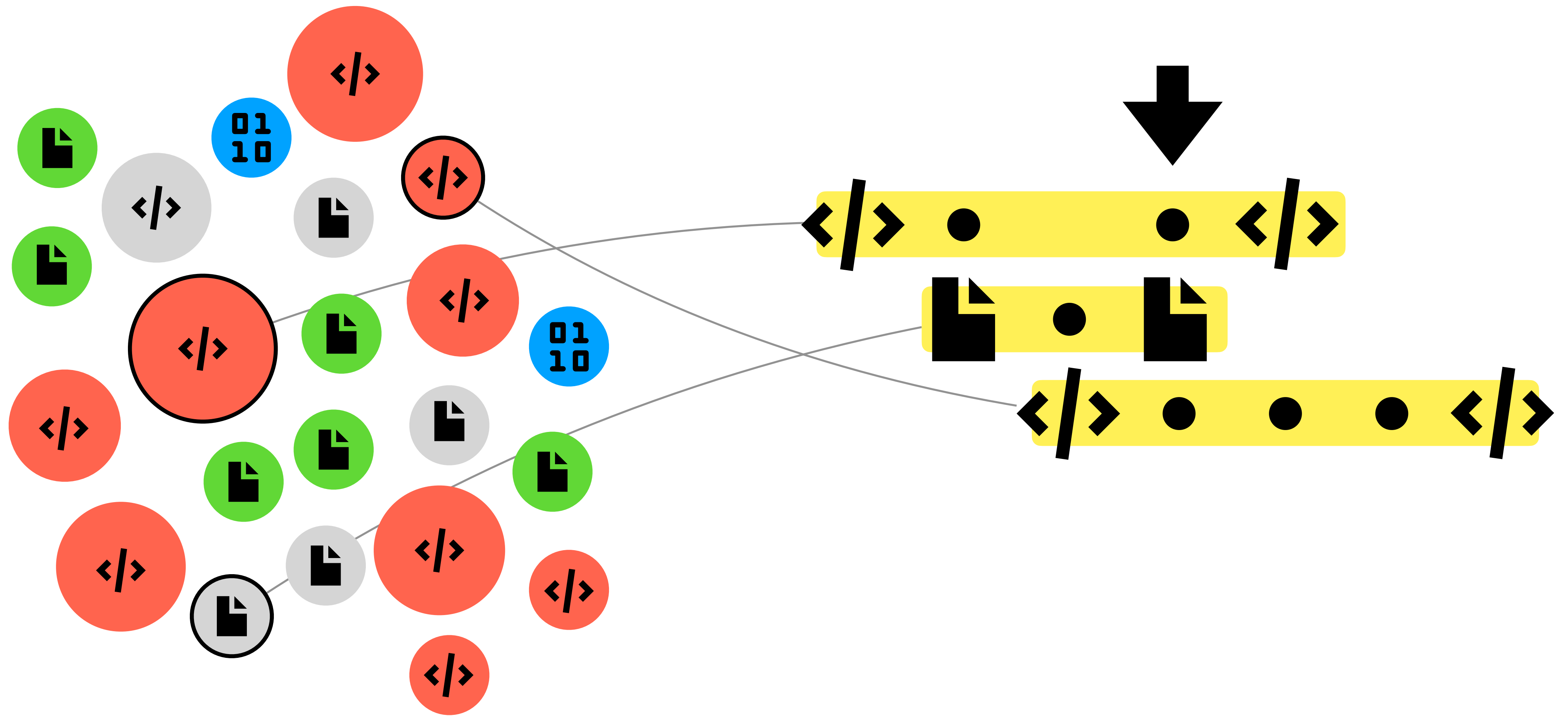


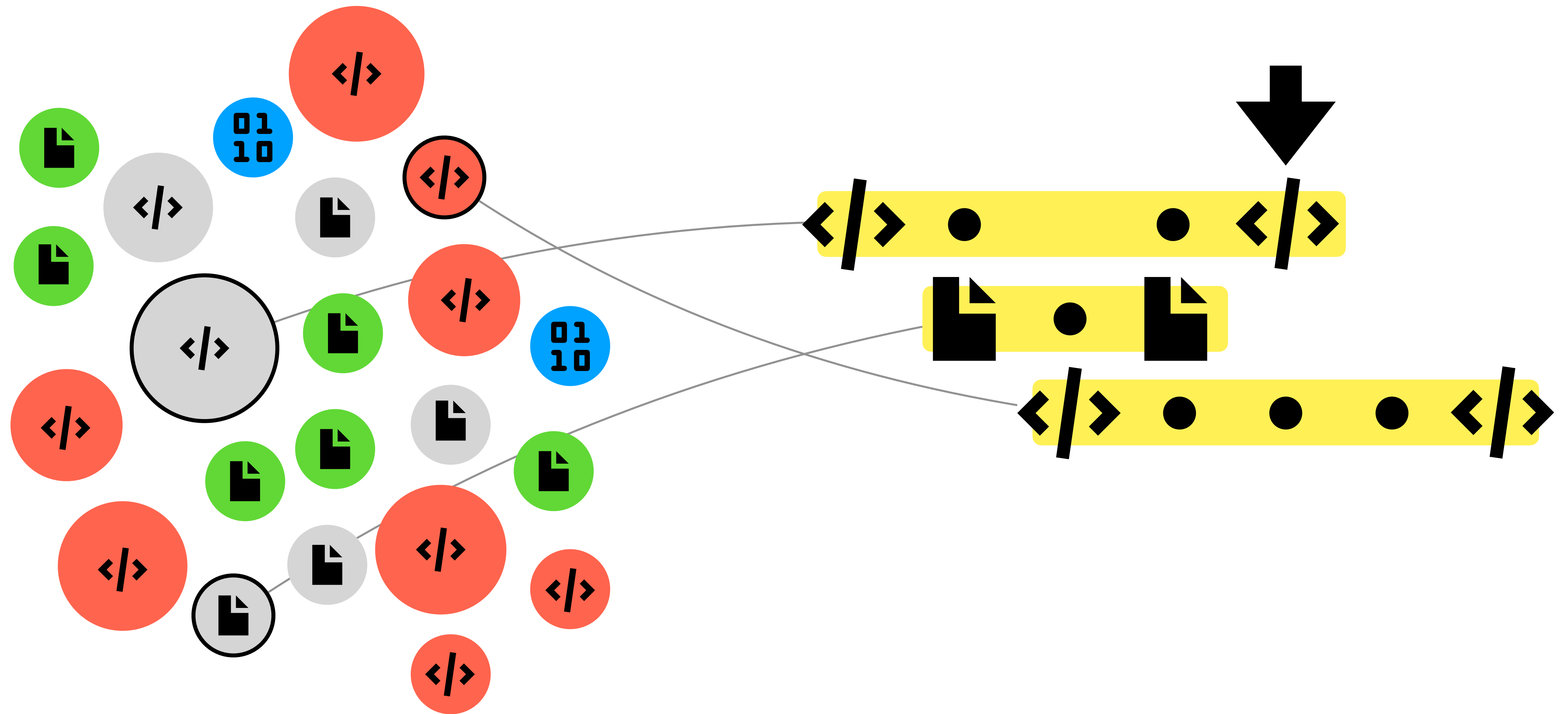


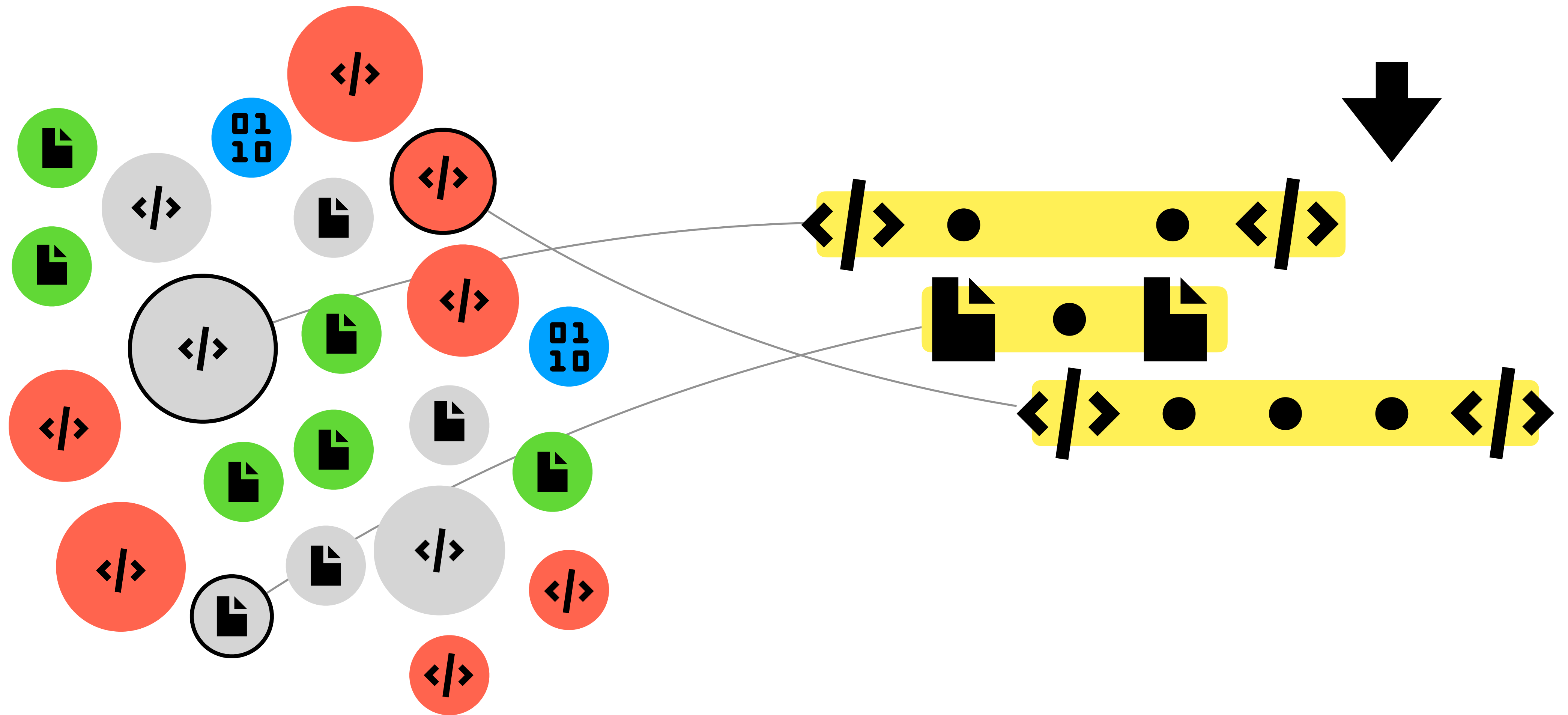




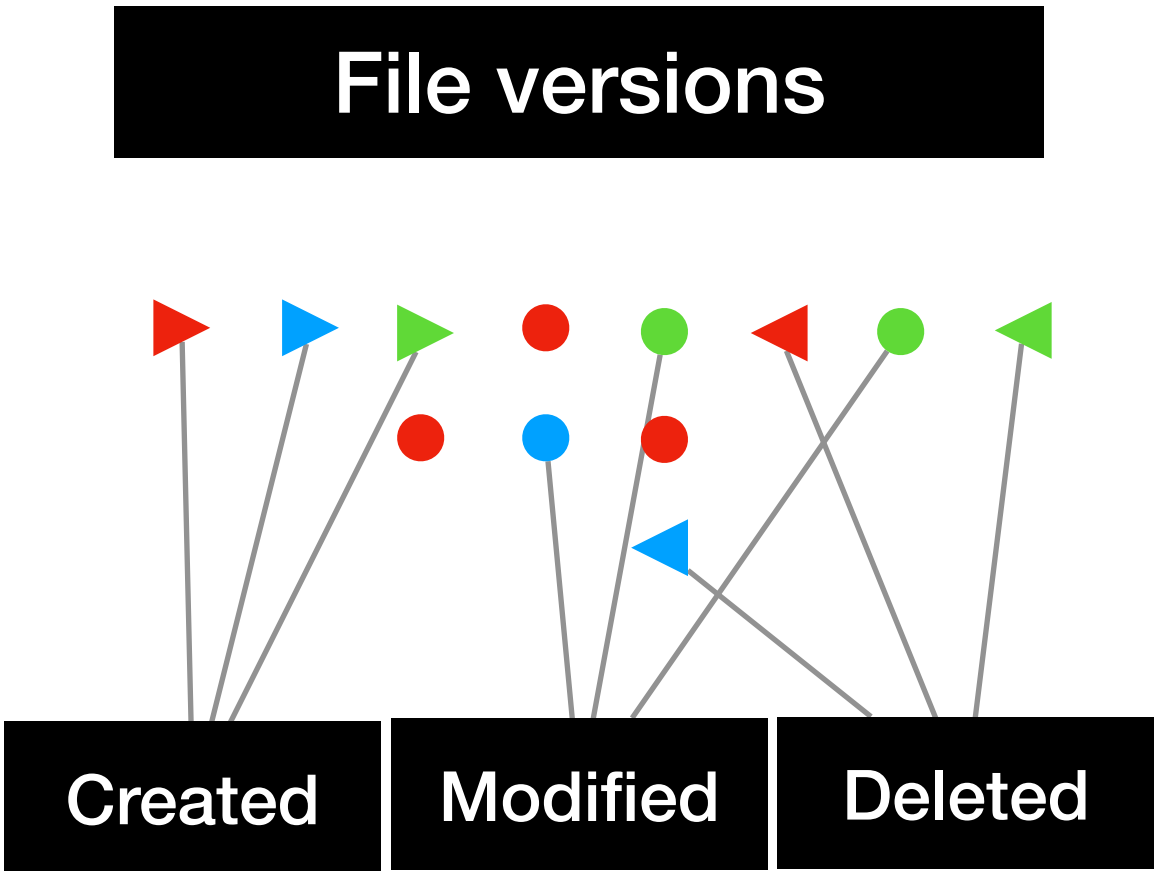
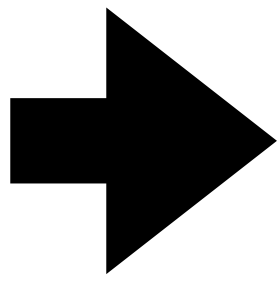
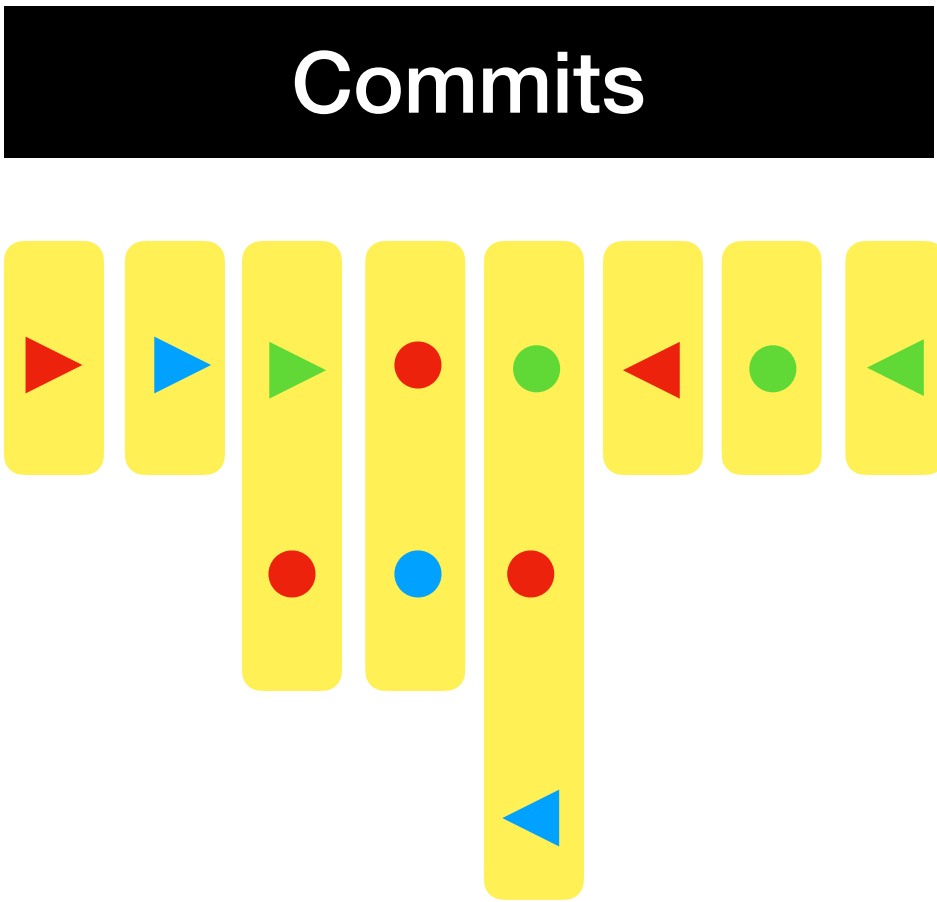




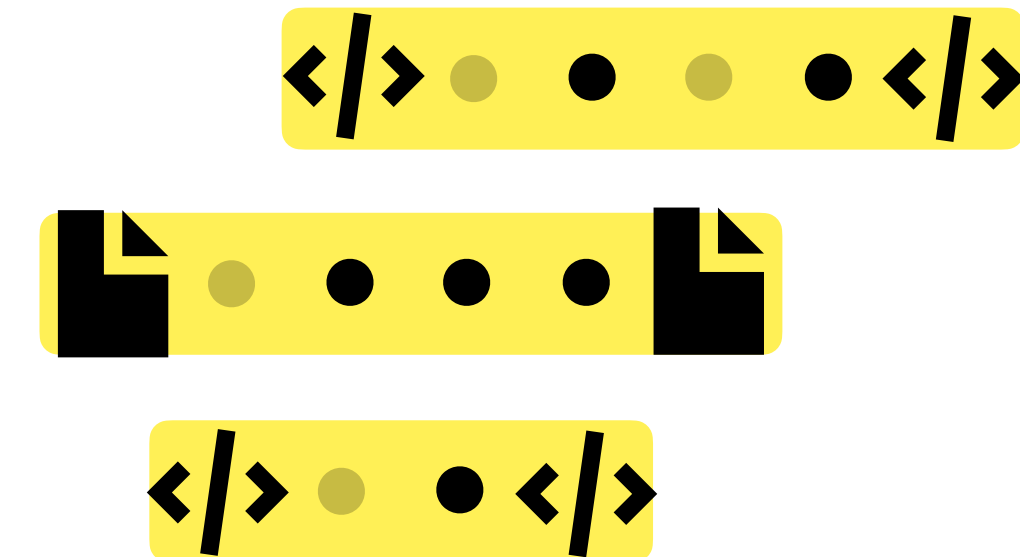
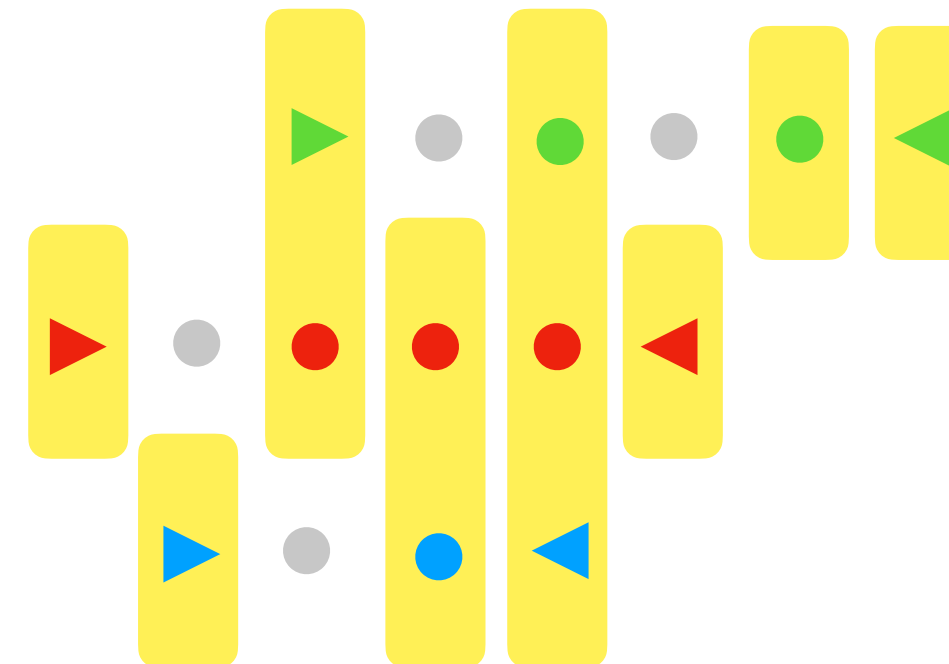
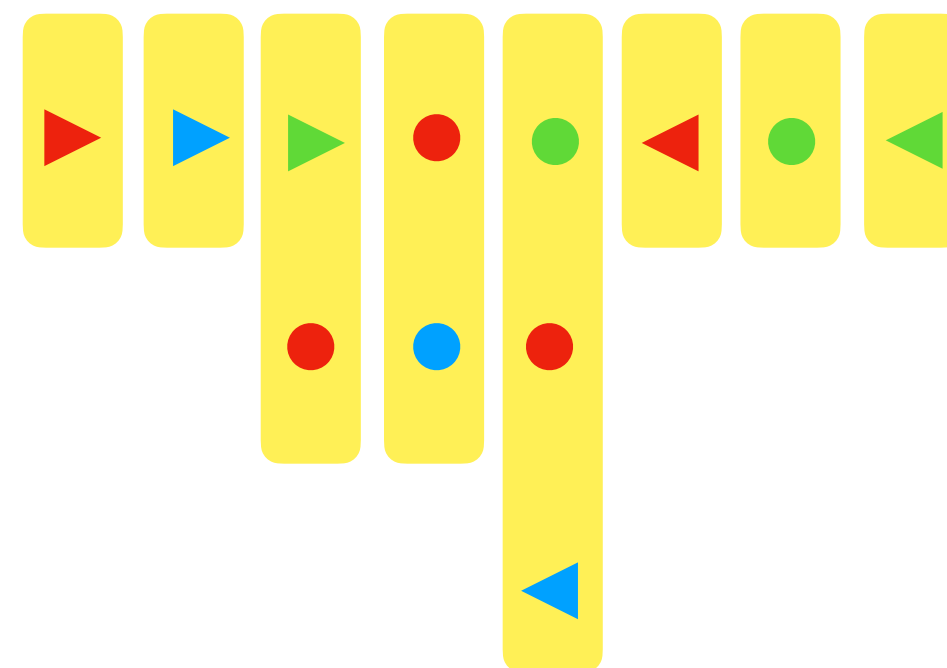




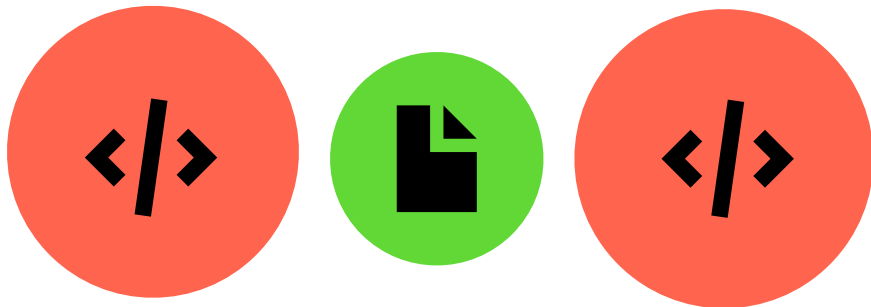
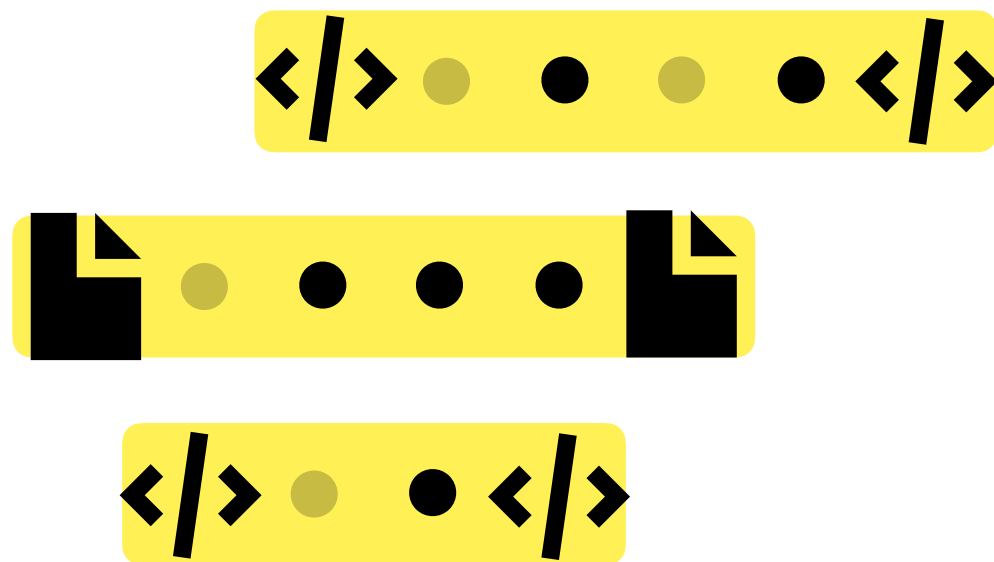
Parsing With GitMiner



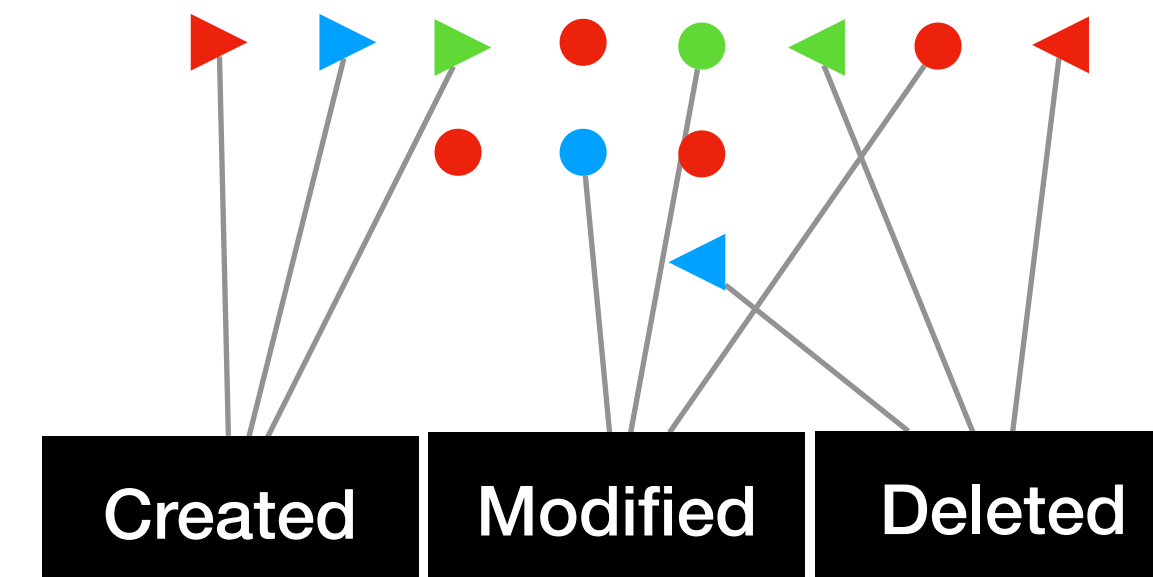
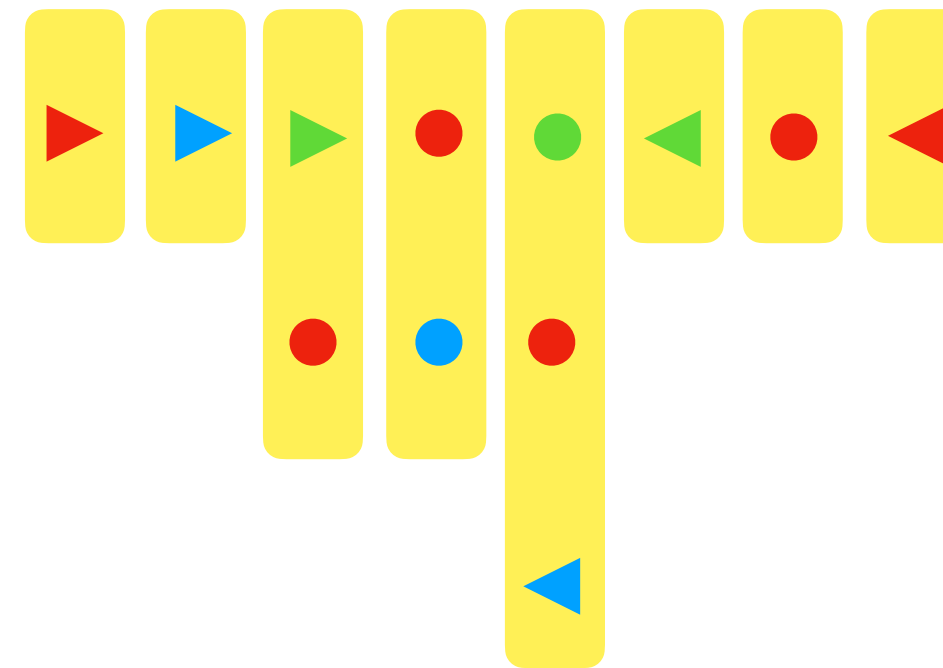
Creating File Histories from GitMiner Data



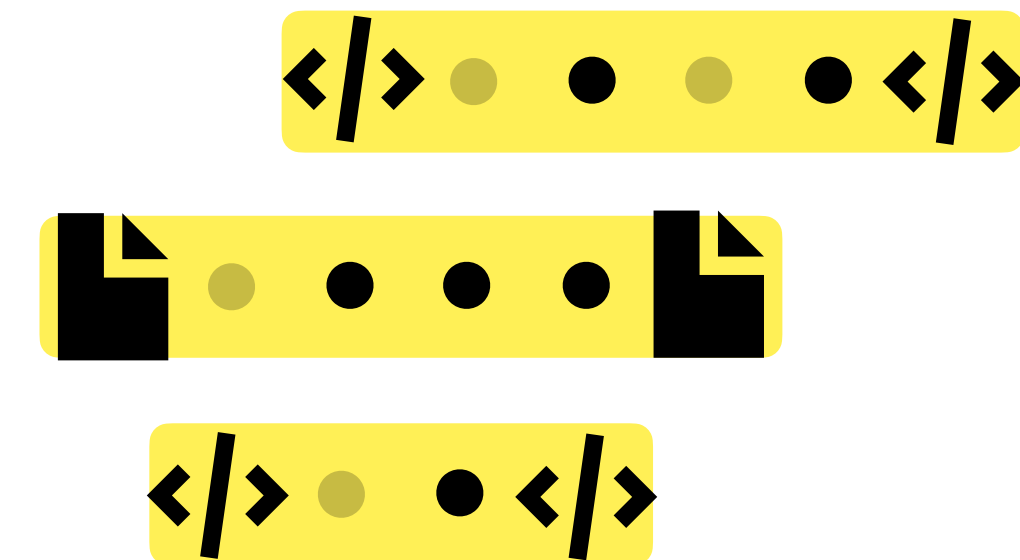
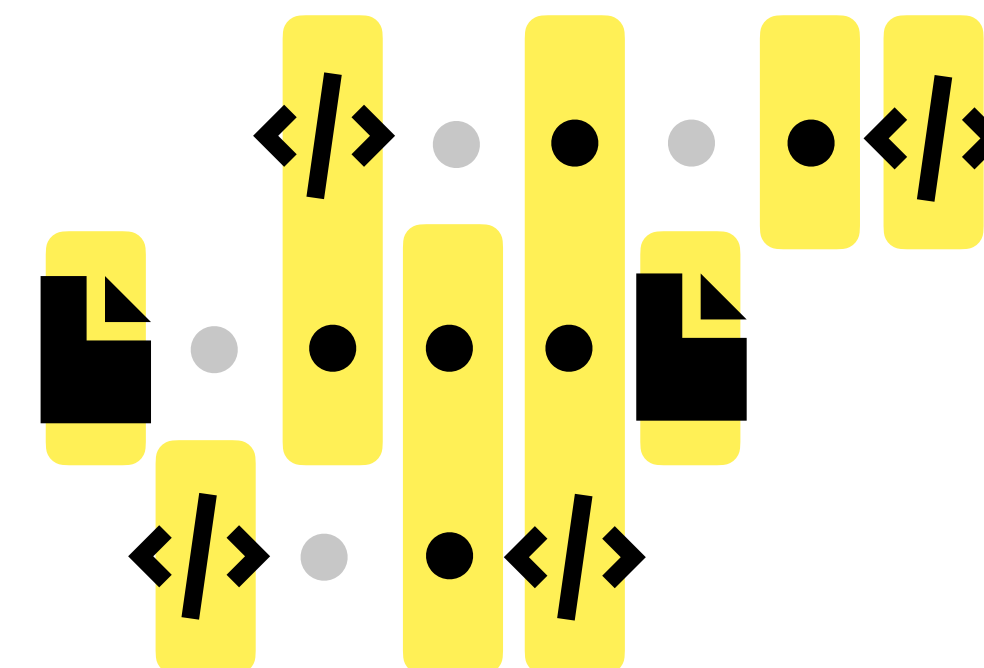
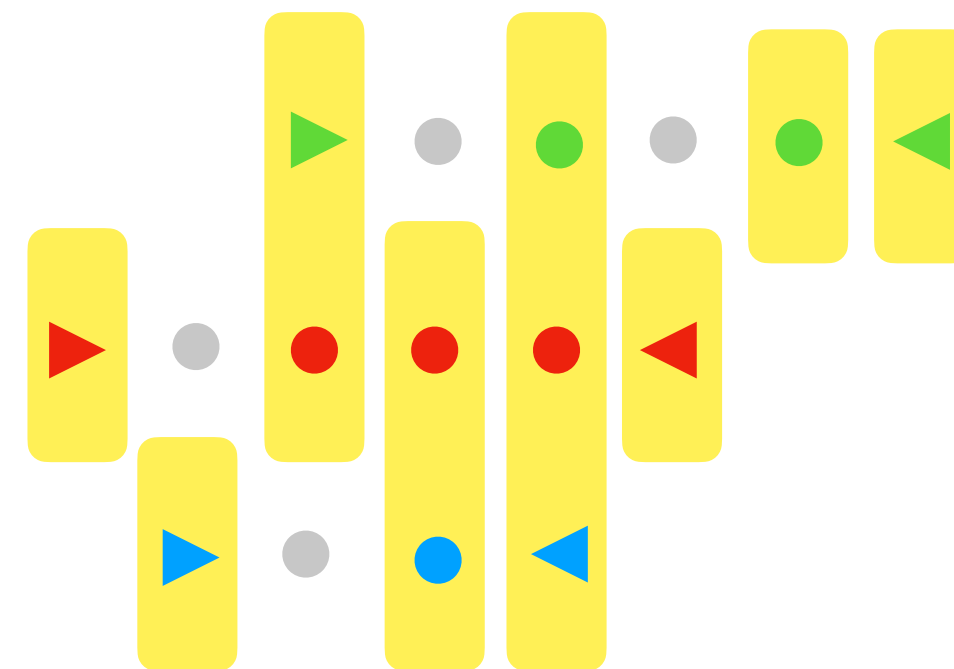
Creating Disks from
File Histories



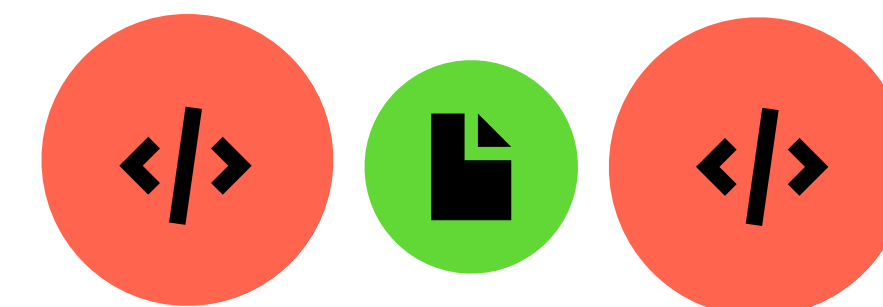
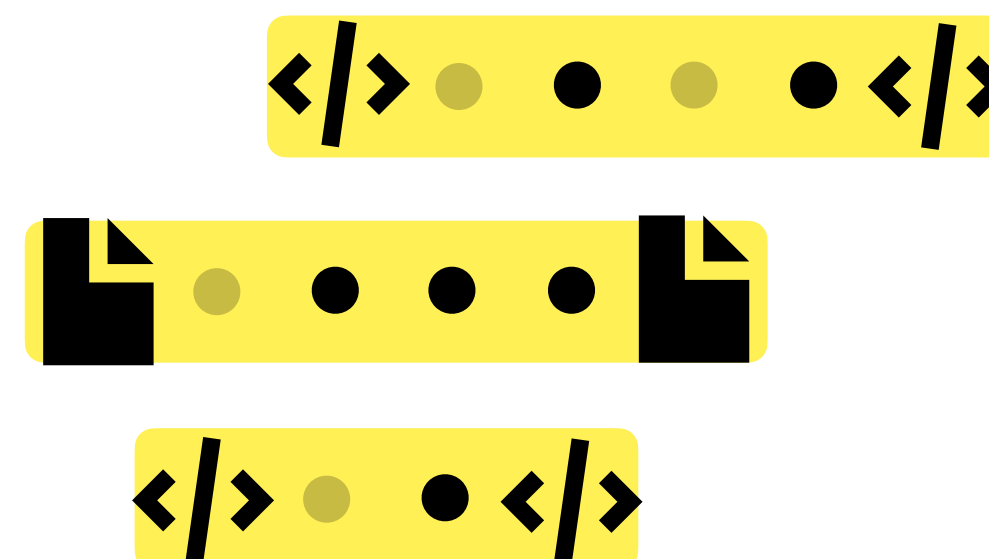
Parsing With GitMiner



Creating File Histories from GitMiner Data

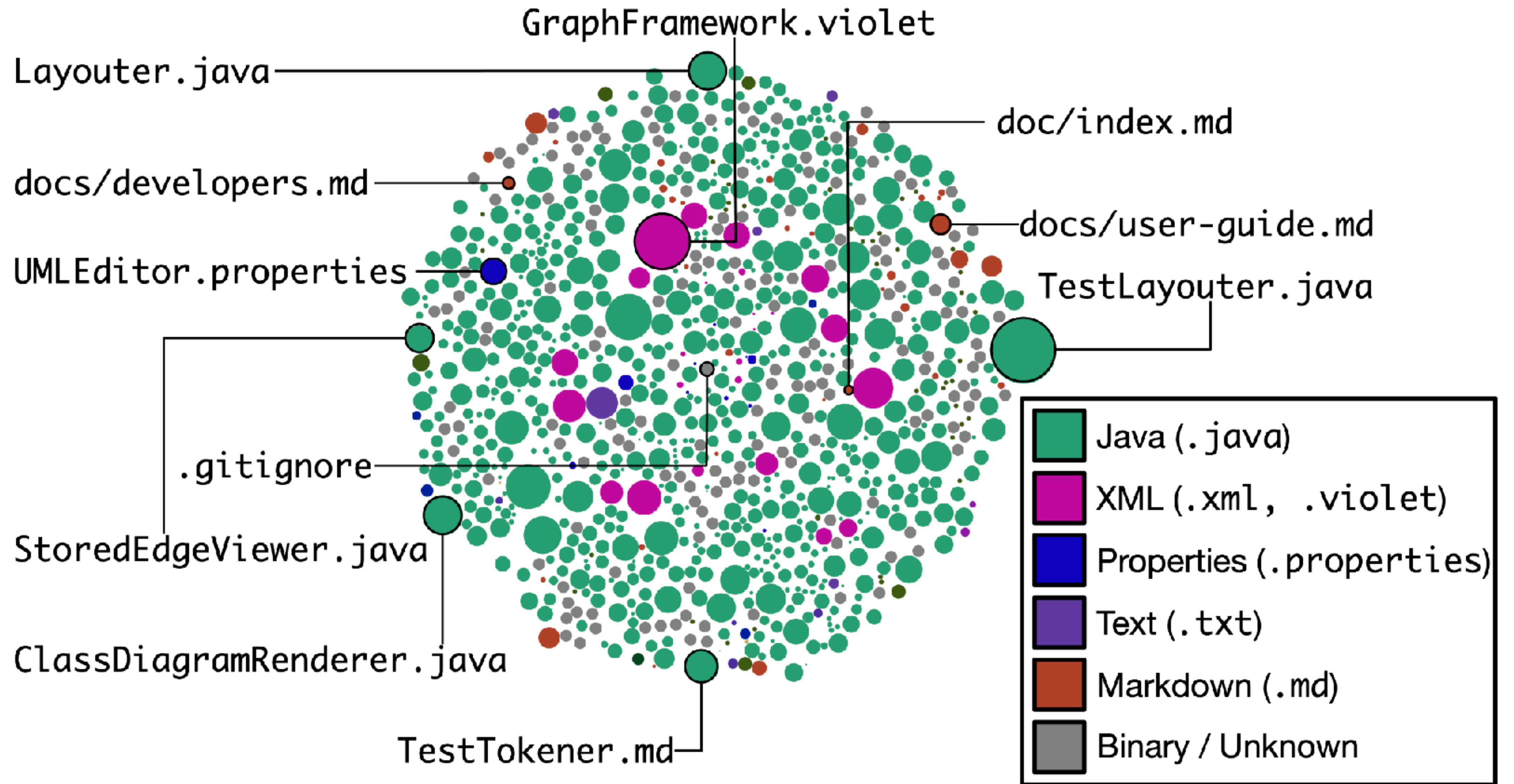


Creating Disks from File Histories





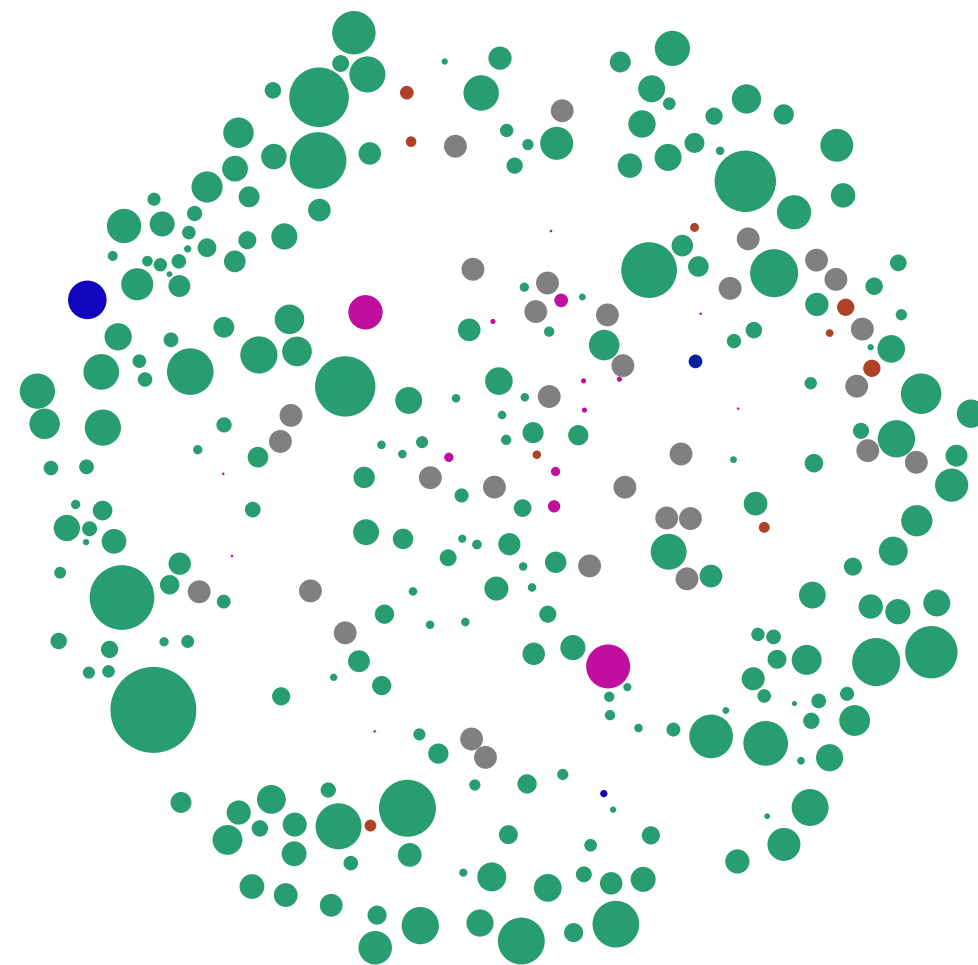
DIVISION ON REAL REPOSITORIES



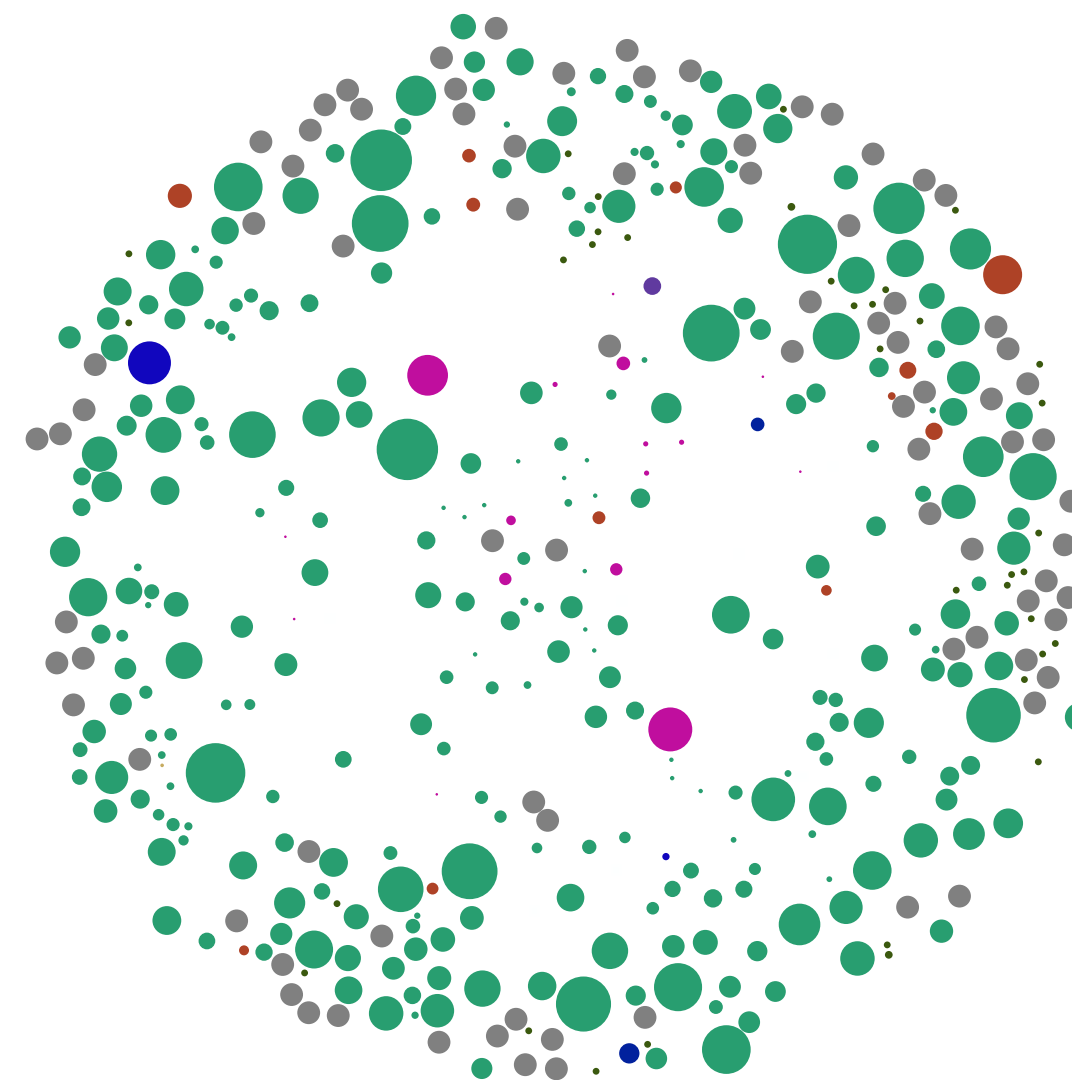
May 07 2016



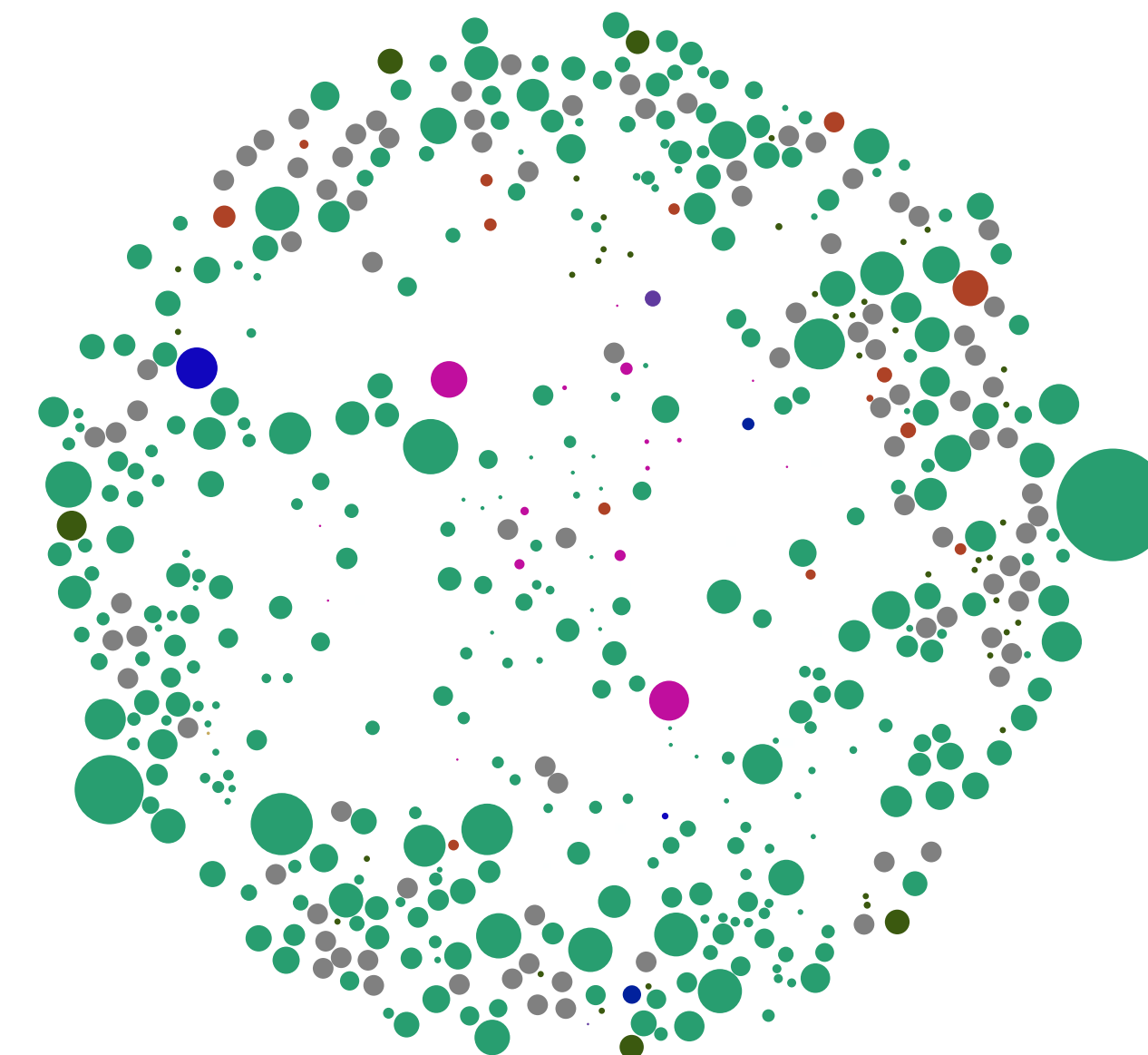
Jan 04 2019



Sep 01 2021



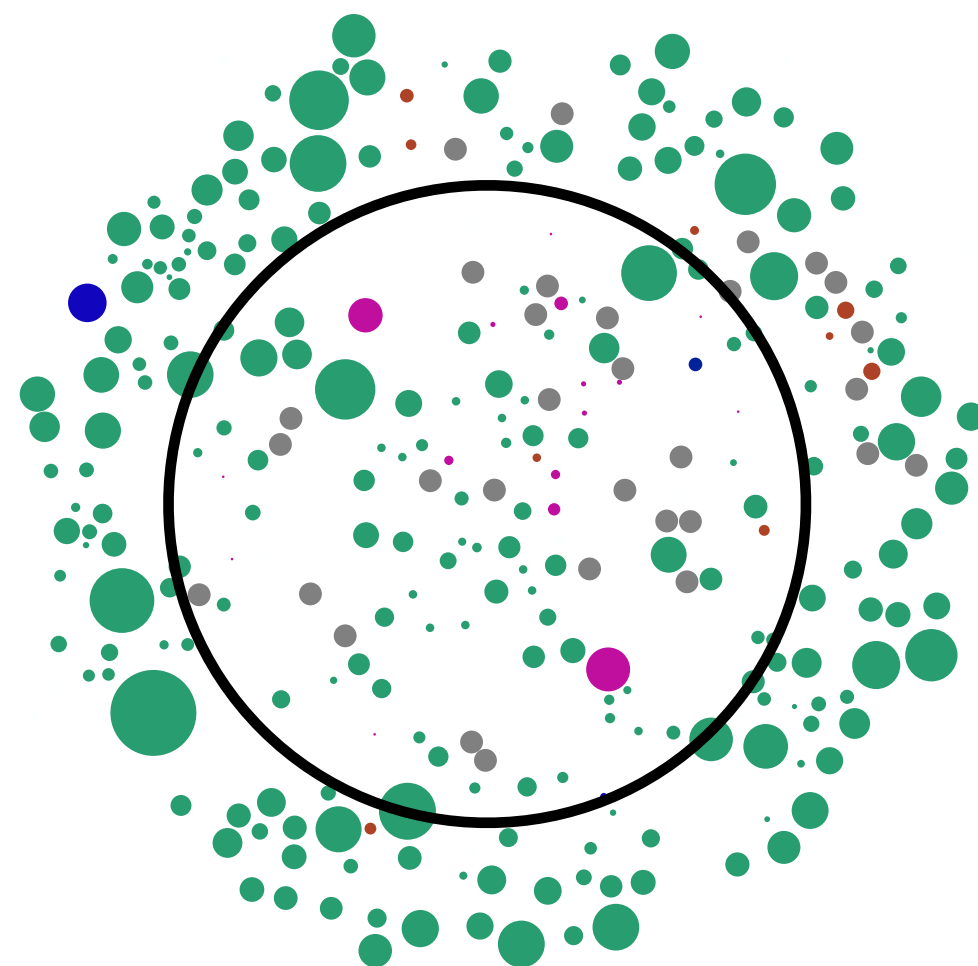
Apr 30 2024



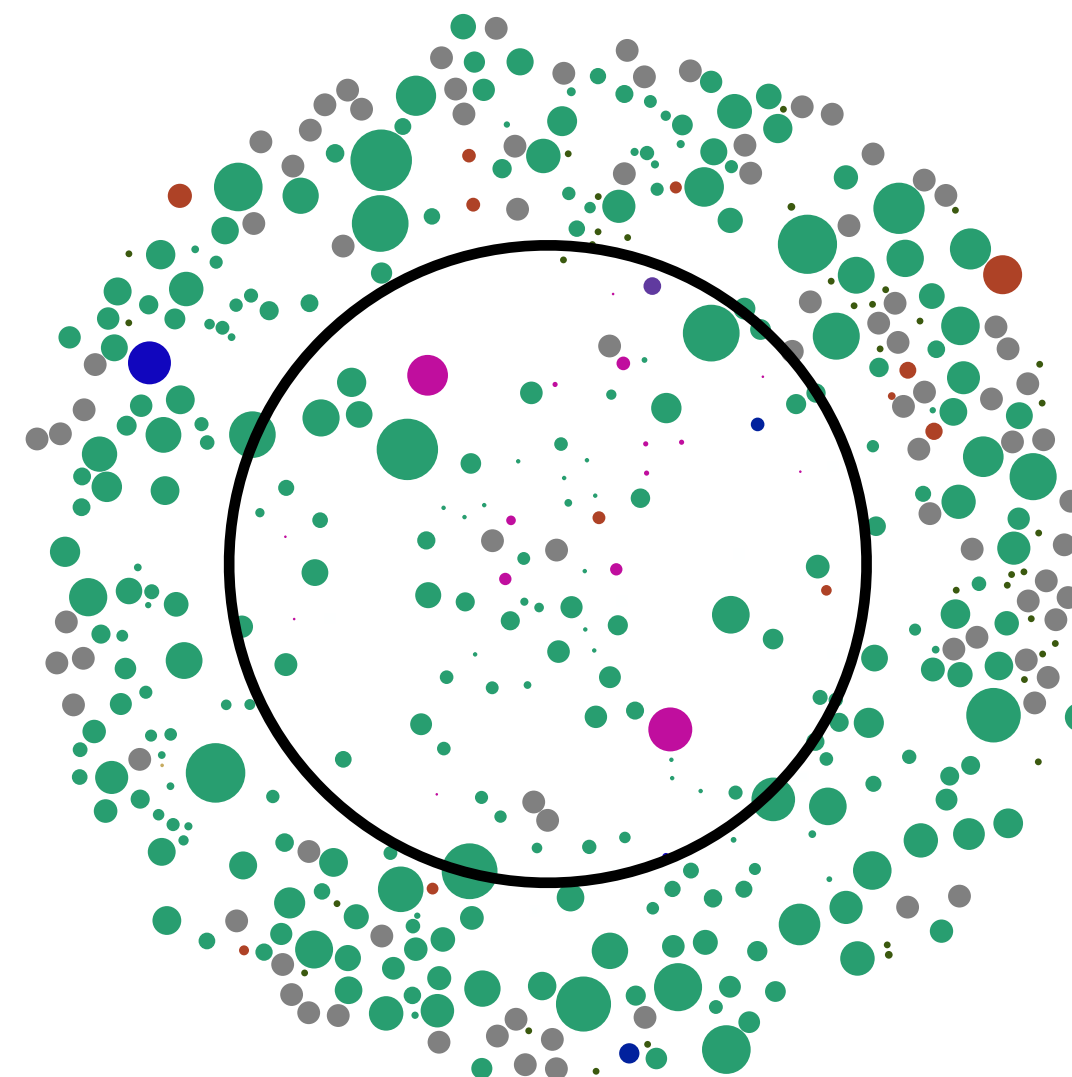
May 07 2016



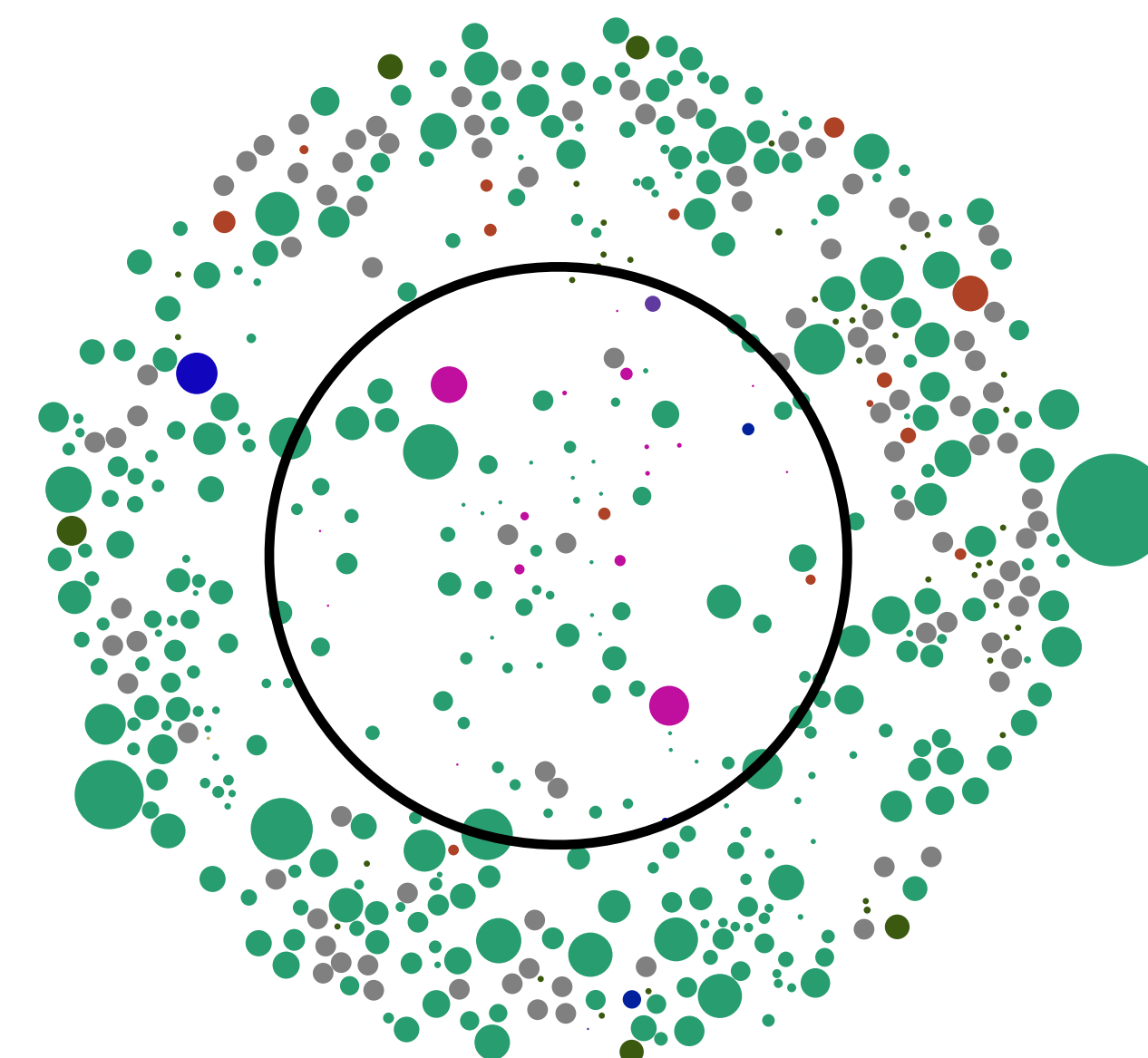
Jan 04 2019

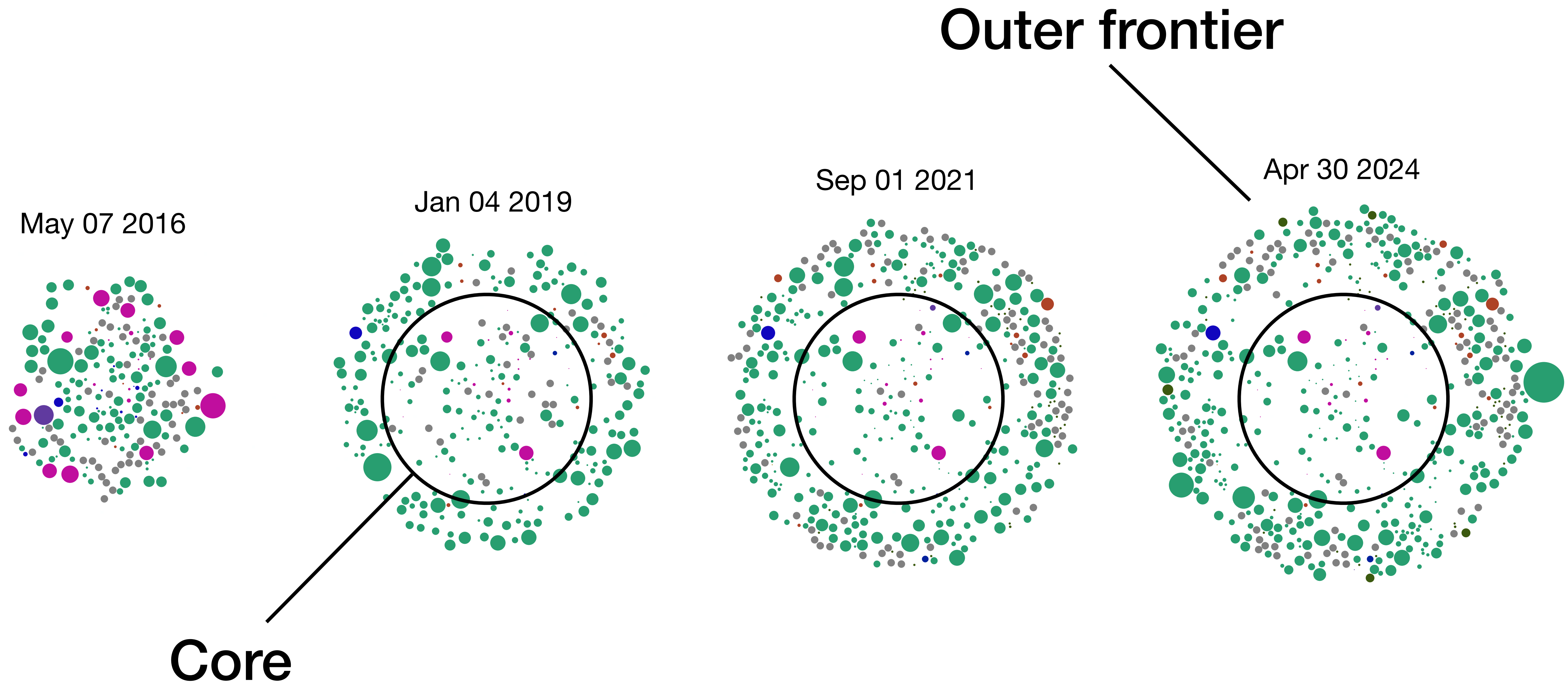


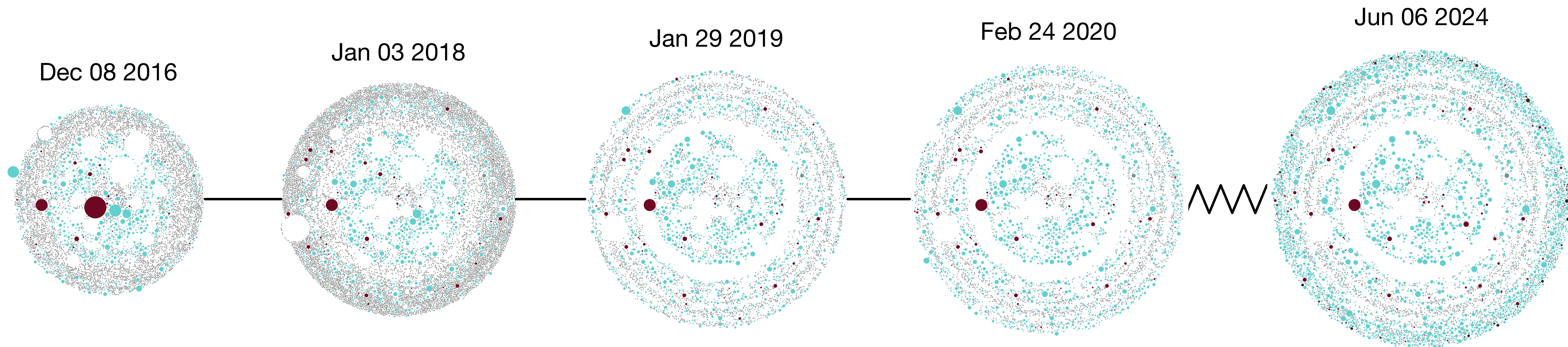
Sep 01 2021



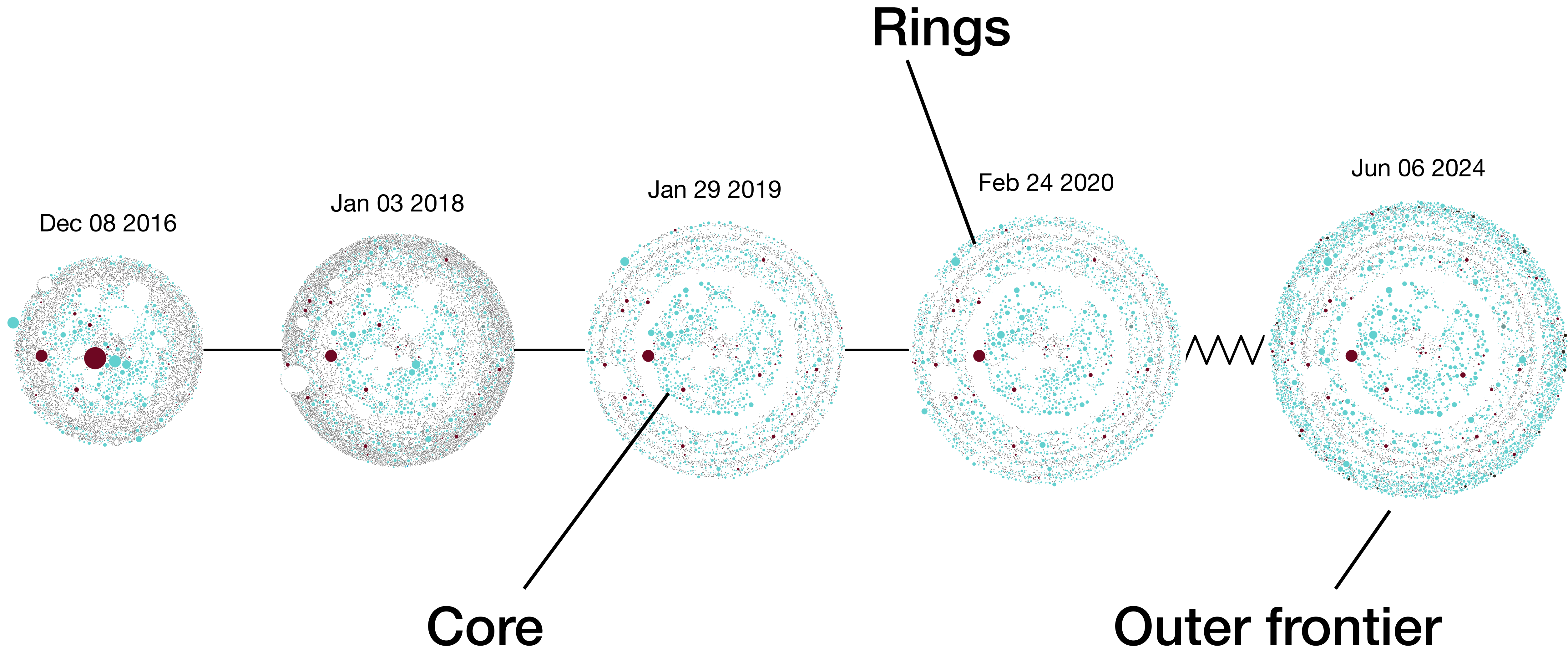
Apr 30 2024

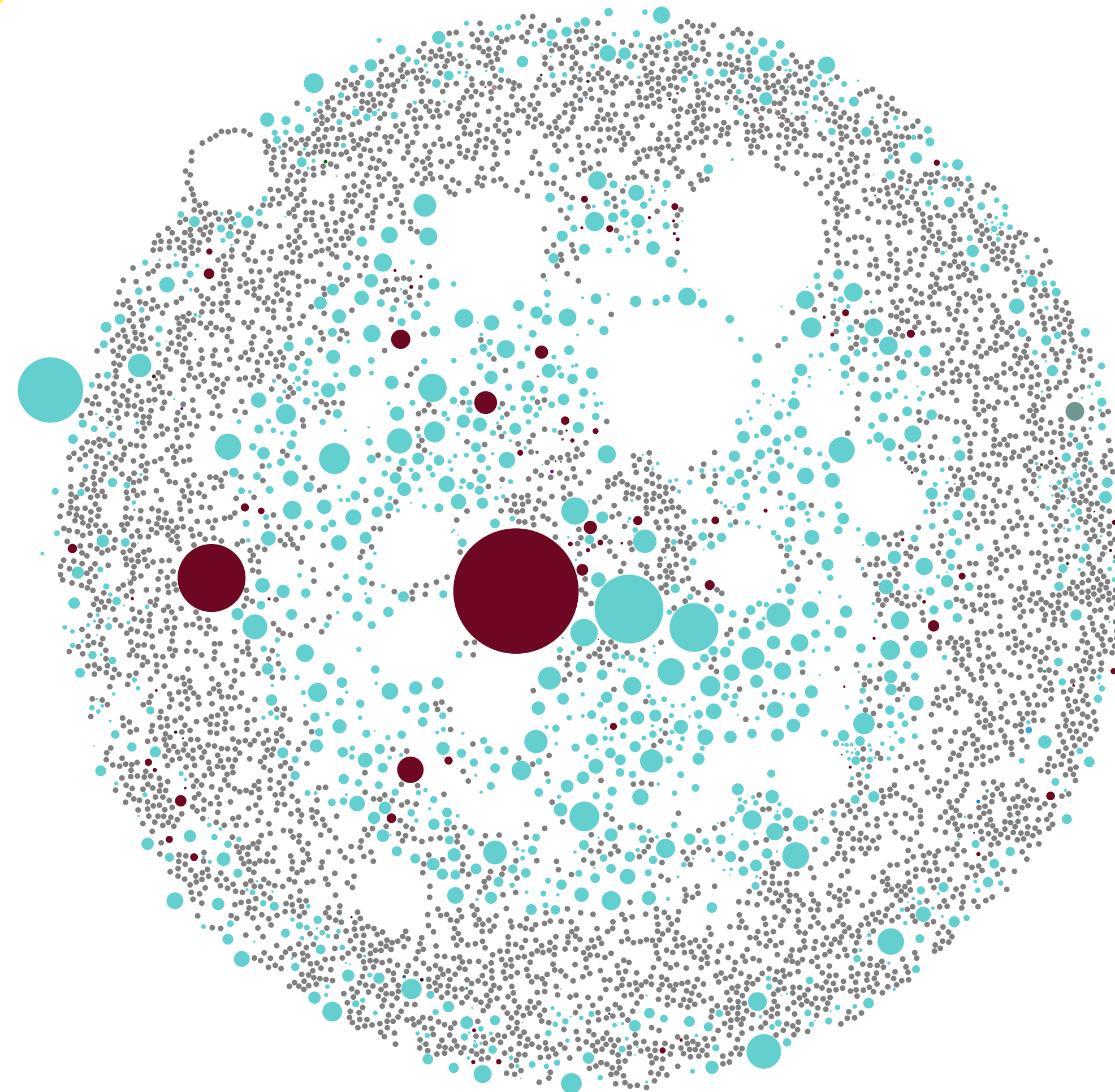
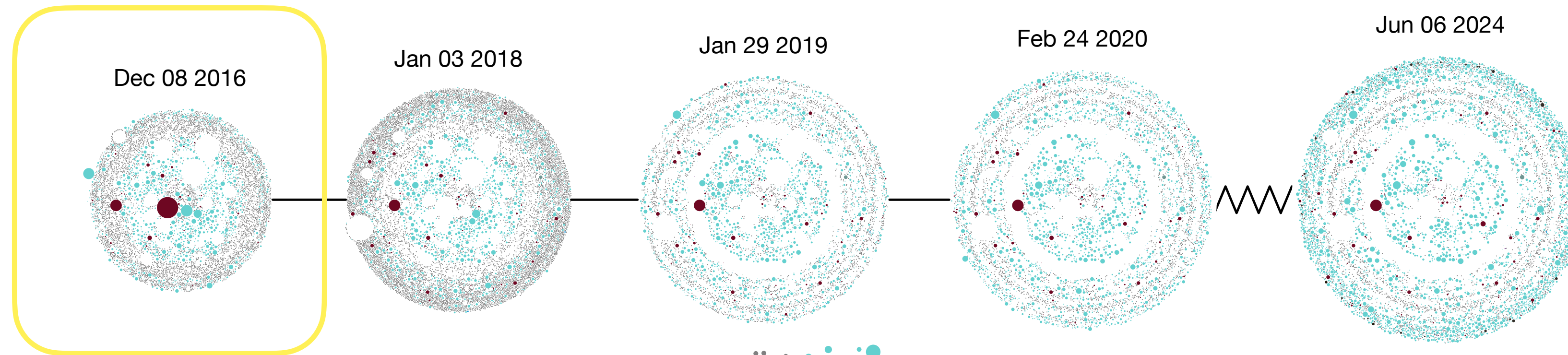


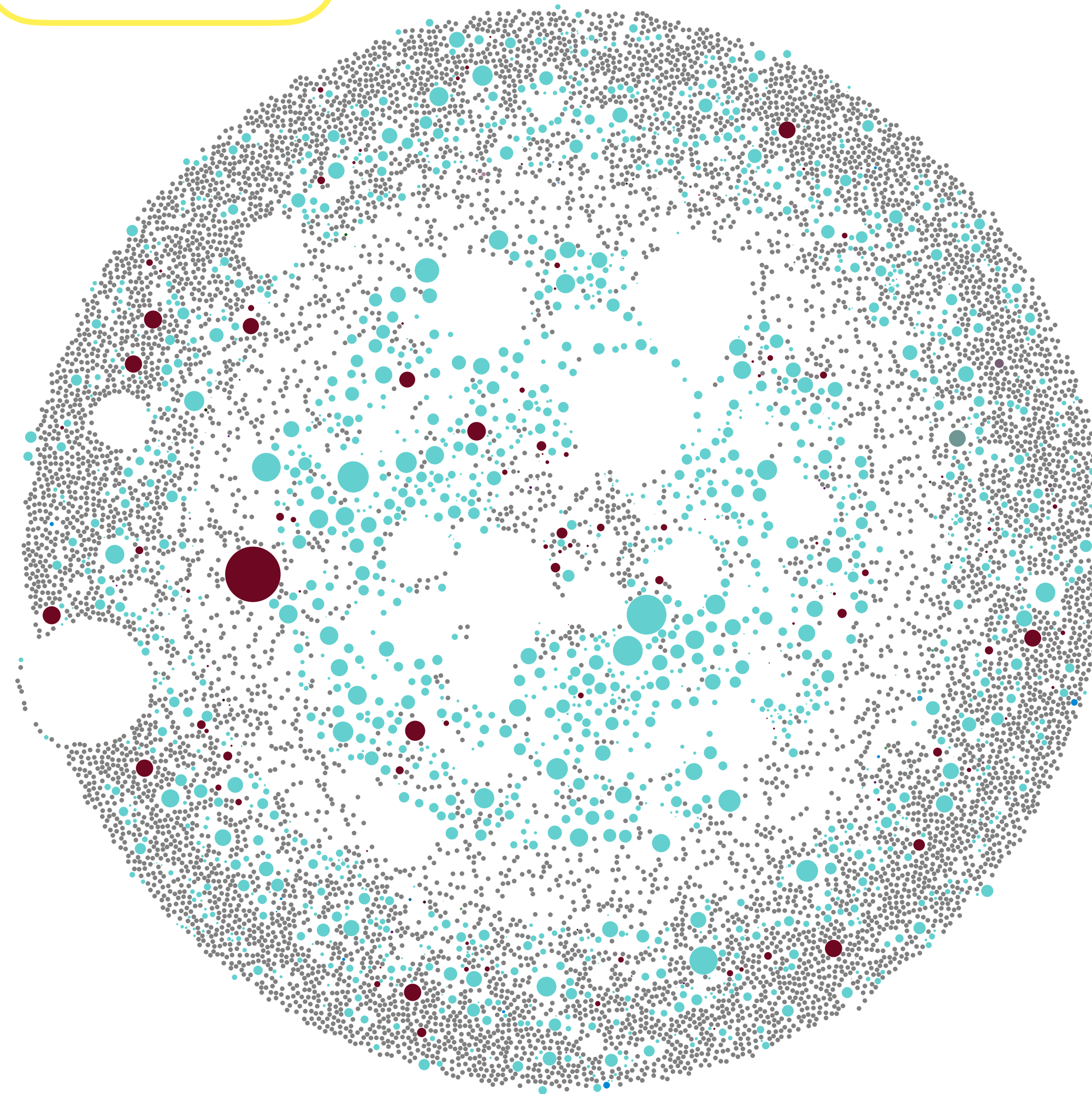
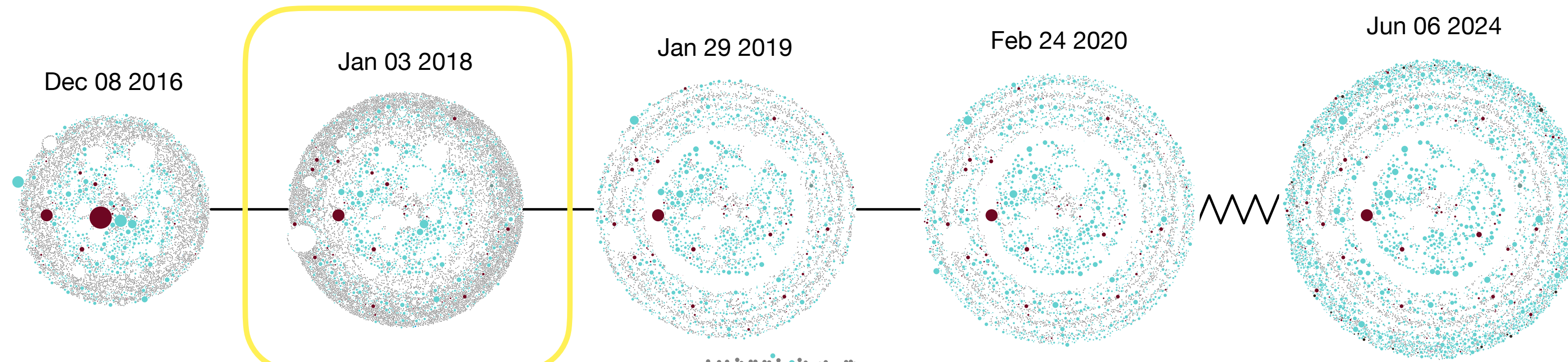


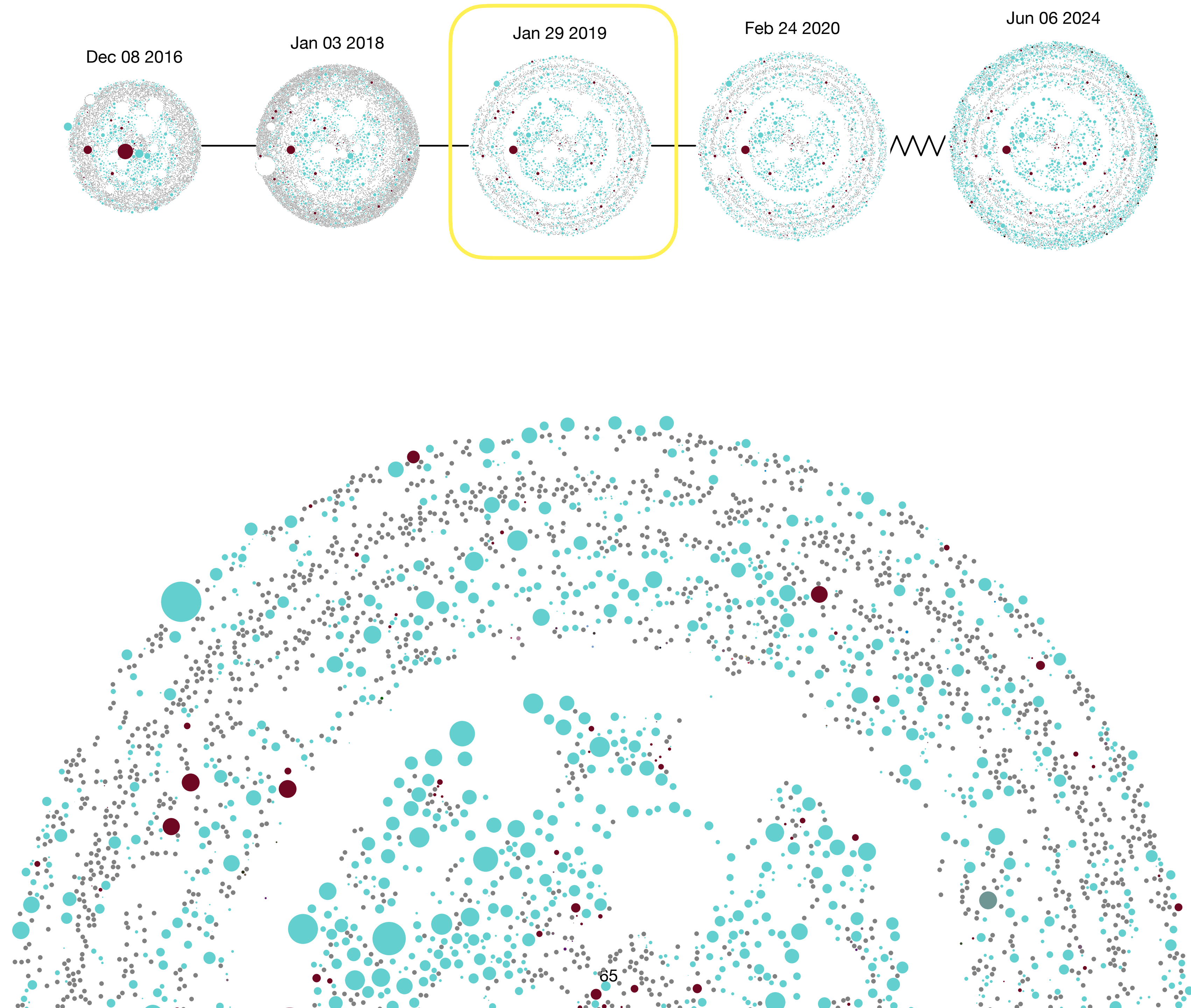


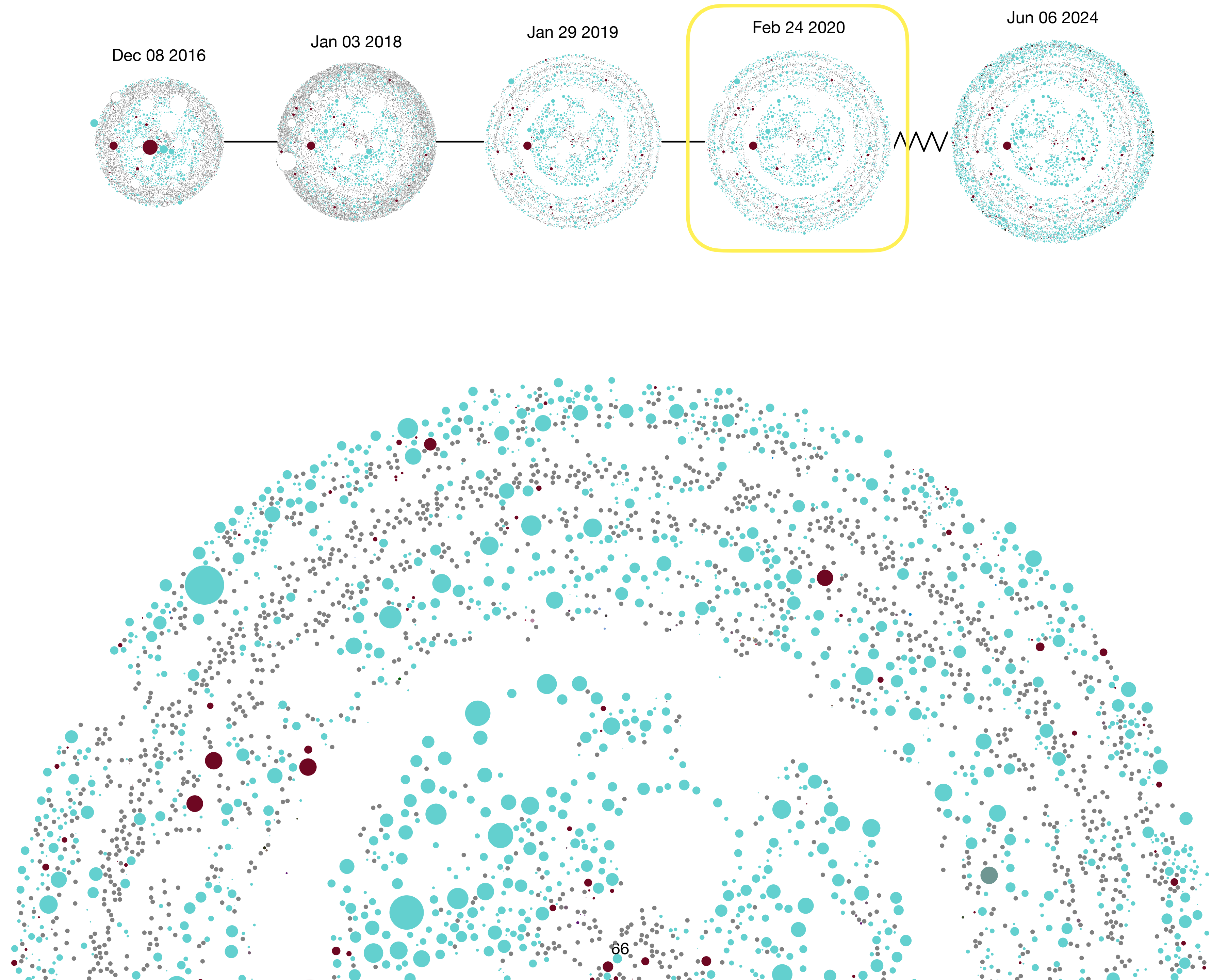
150k commits
122k file histories

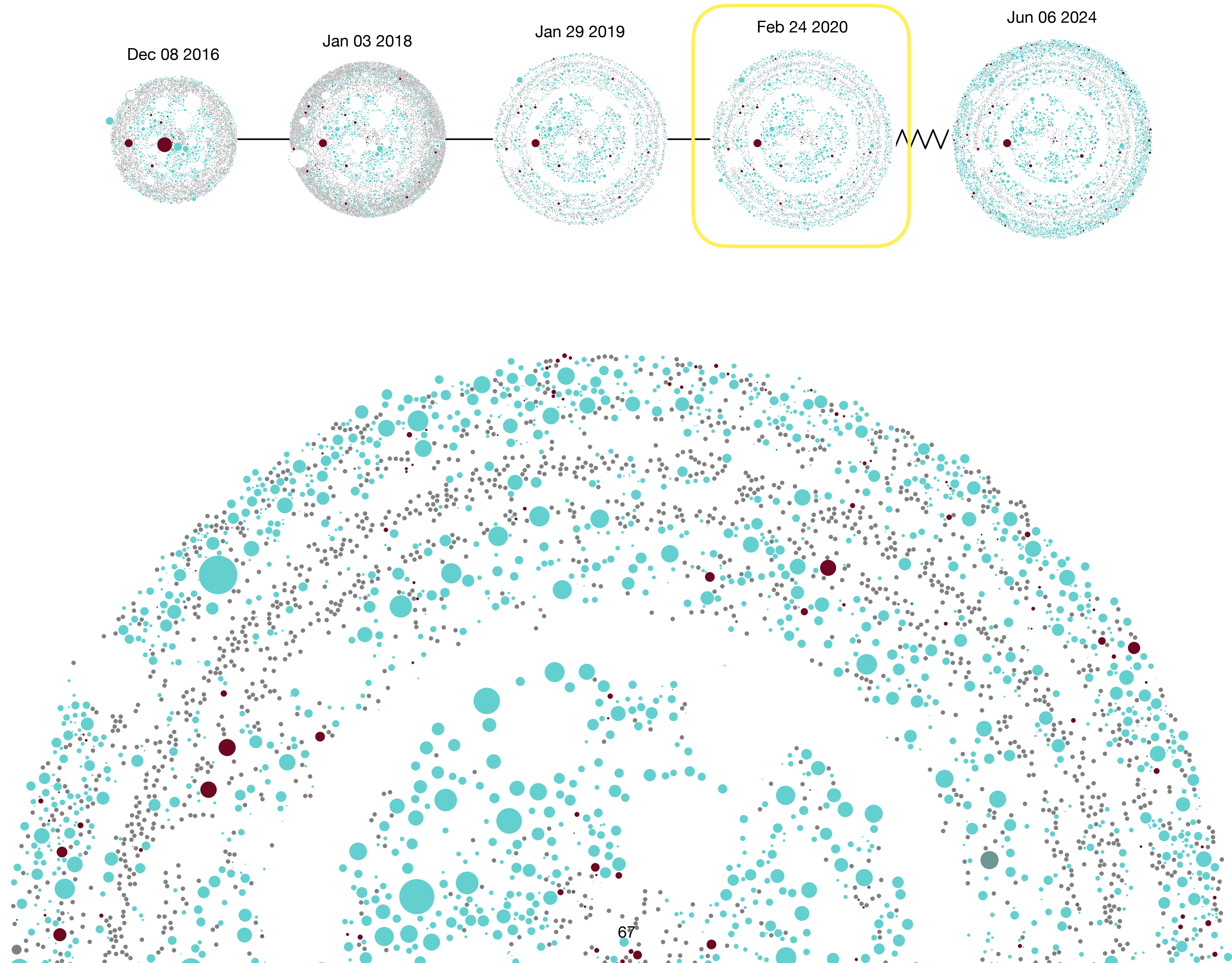


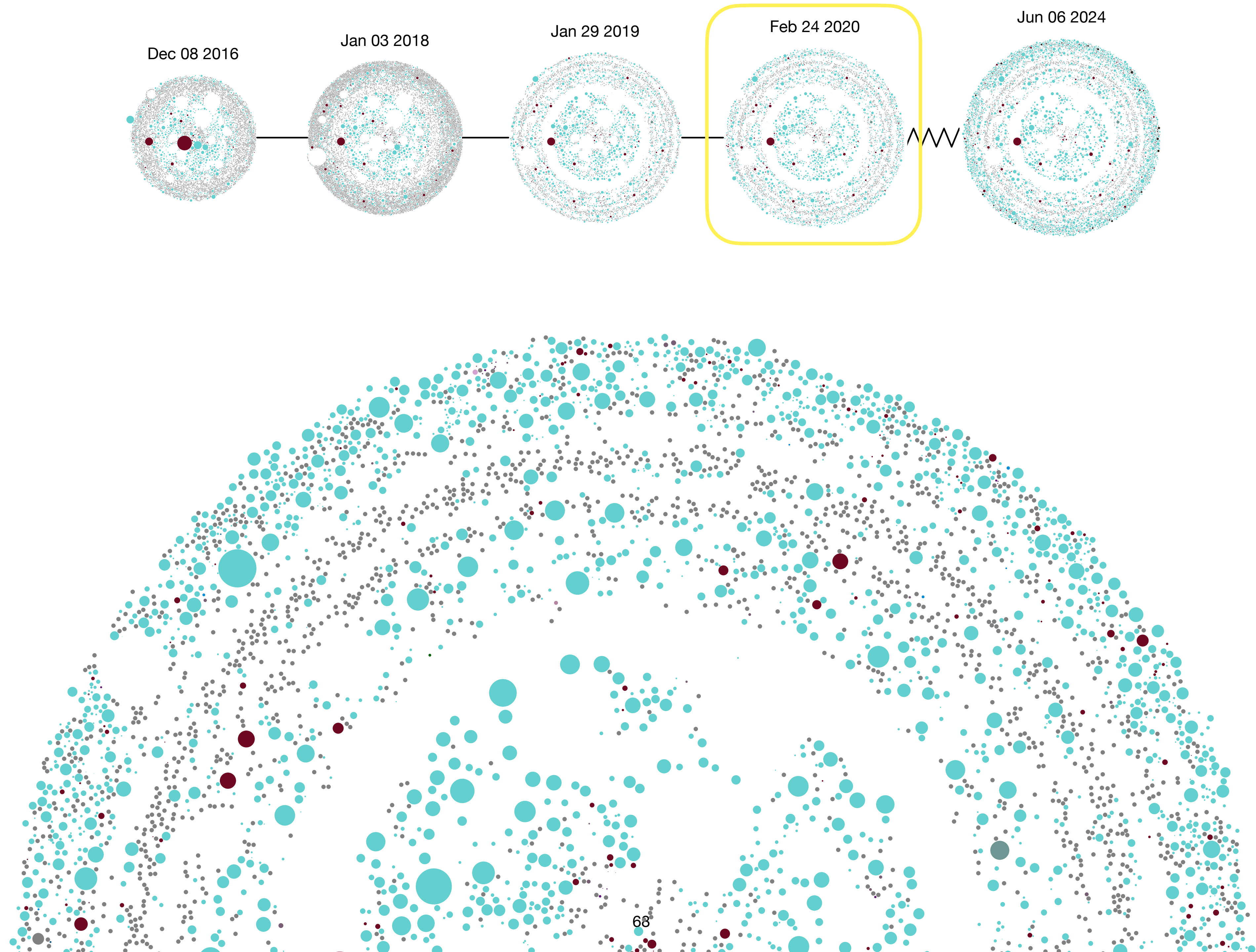


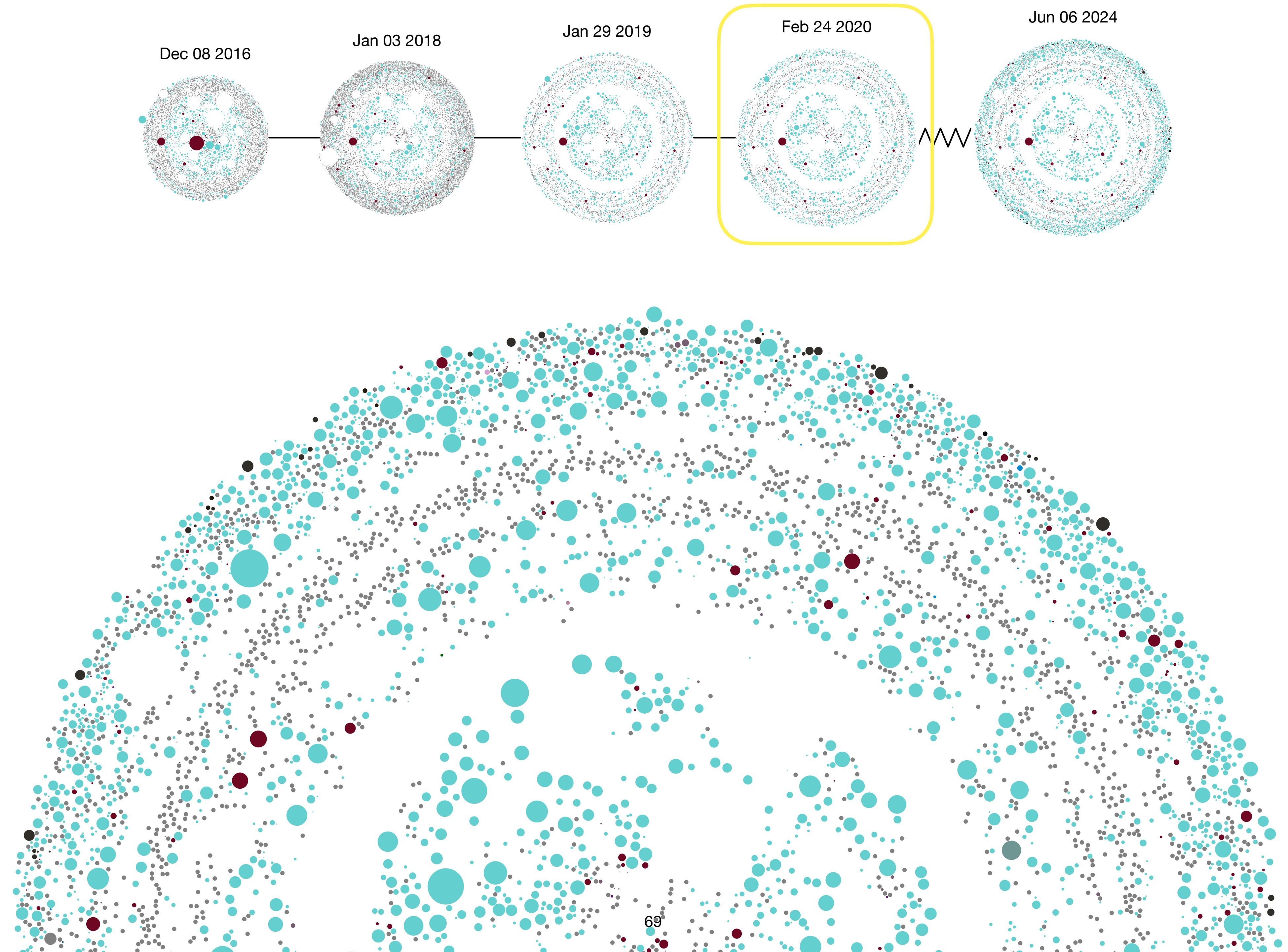


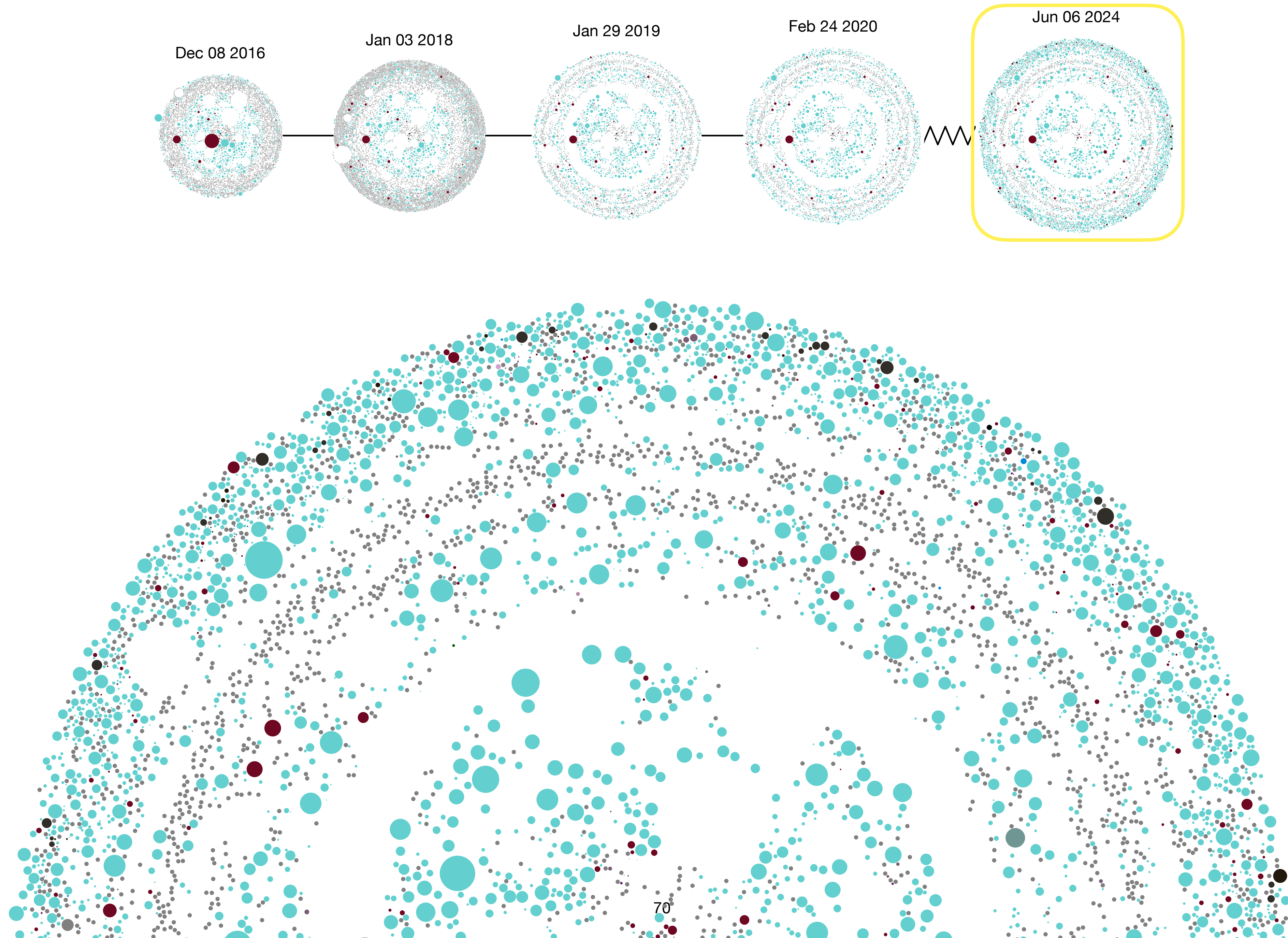












What we've learned so far

- DVision allows for visualizing large-scale repositories at the file level without *scalability* issues



What we've learned so far

- DVision allows for visualizing large-scale repositories at the file level without *scalability* issues
- Visual elements do not move during animation, so there is no *displacement*, but properties changing still shows evolution



What we've learned so far

- DVision allows for visualizing large-scale repositories at the file level without *scalability* issues
- Visual elements do not move during animation, so there is no *displacement*, but properties changing still shows evolution
- Poisson disk sampling naturally solves the *collision* problem and gives meaningful ordering to the elements





THE ZEPHYR

OPERATING SYSTEM

What is Zephyr?

- A popular *real time operating system* for embedded devices, maintained by the Linux Foundation
- Over 10 years of history, it has reached 130K commits and 80K file histories
- We were approached for a collaboration to visualize the contents of the Zephyr repository in November 2025

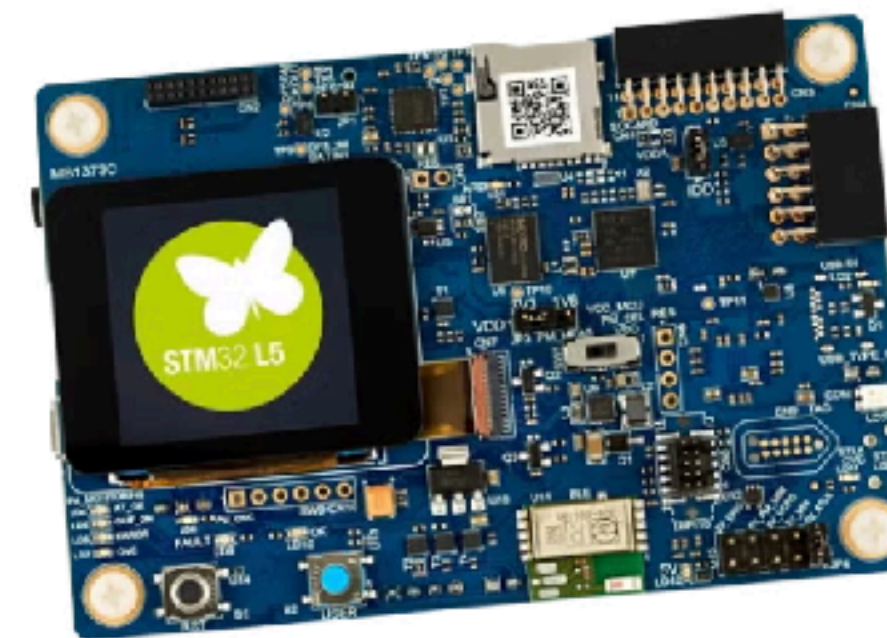
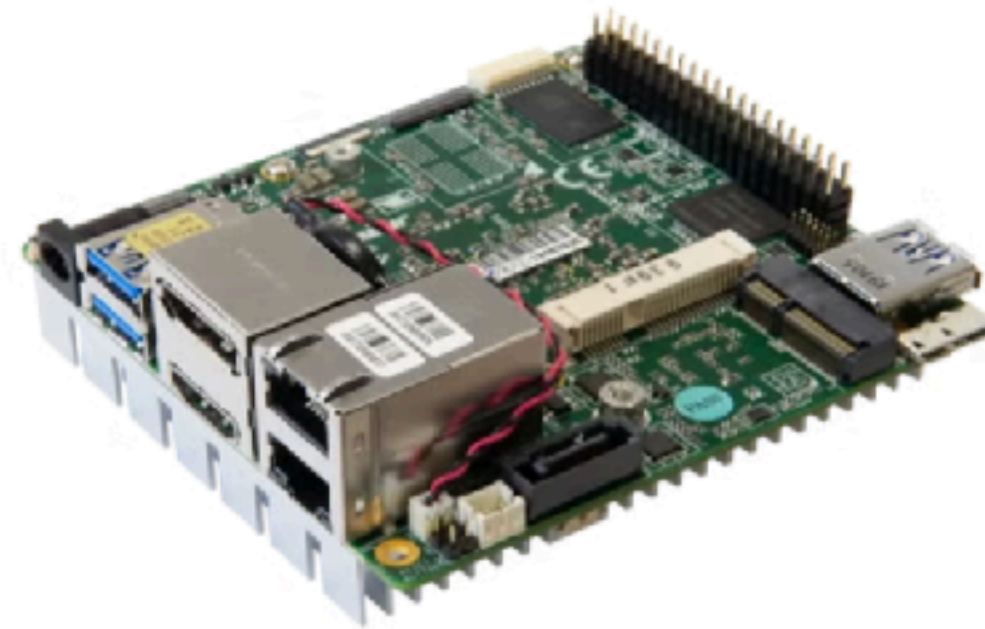
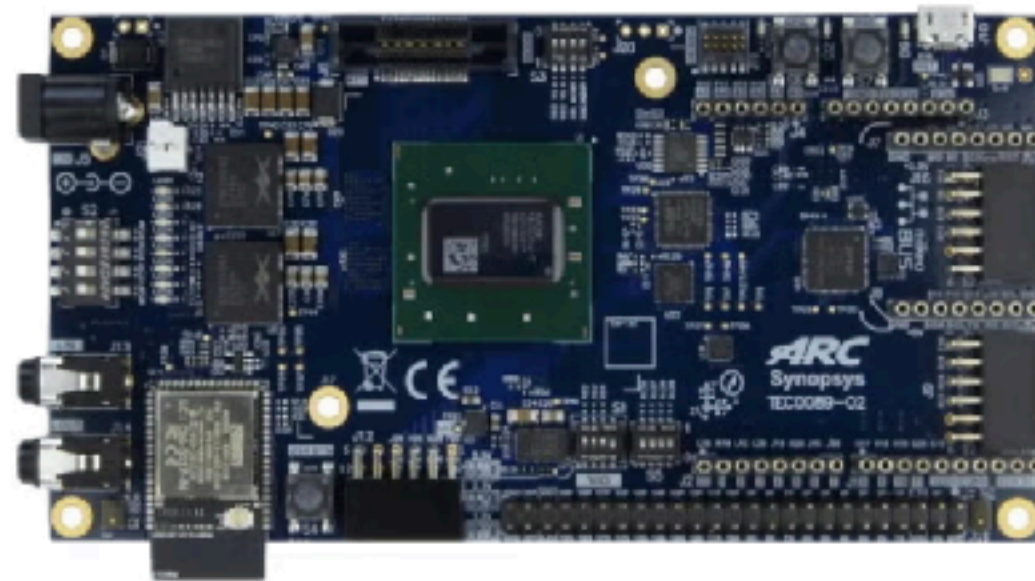


Feedback from Zephyr maintainers

- The only immediate statement of the visualization is that software grows. It does not specify which parts of the repositories are growing
- Stakeholders want to focus on their specific areas of interest, and the visualization was not designed with Zephyr in mind.
- The solution: isolate an area of the repository with massive growth and of interest to stakeholders: *boards*

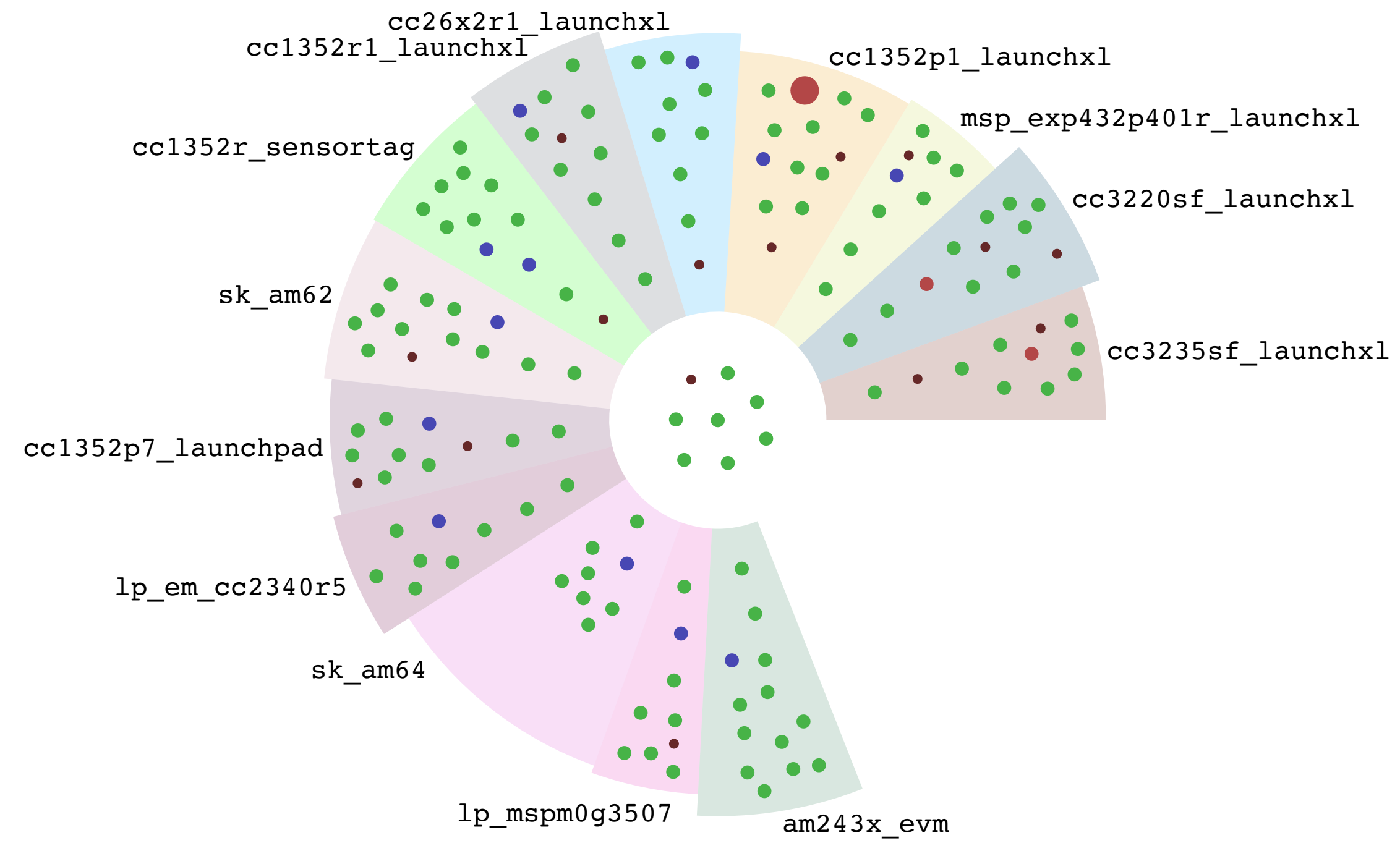
1000+ Boards Supported

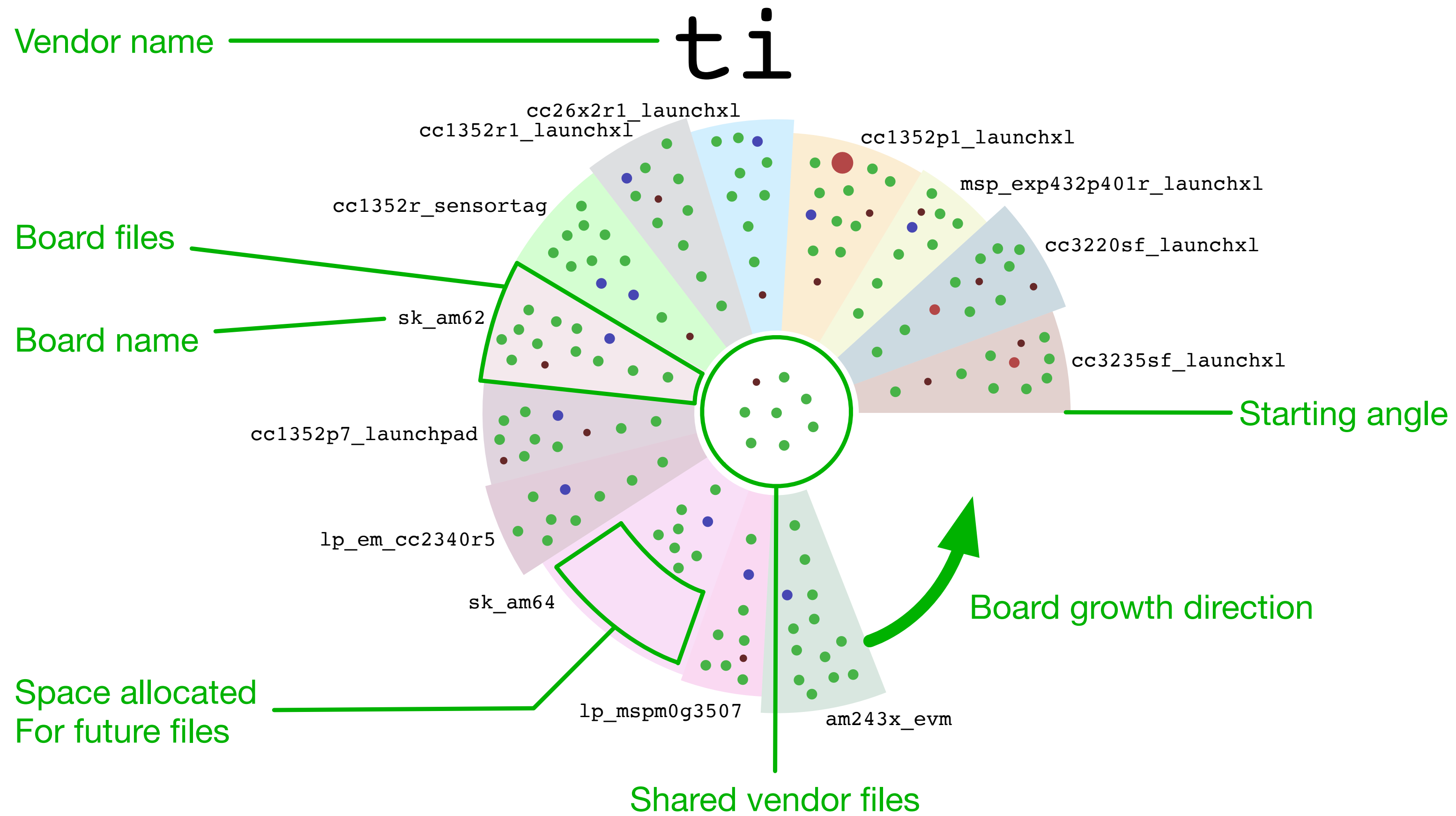
Zephyr supports more than 1000 boards. Search [our list](#) for the hardware used in your application. This diversity of supported boards gives developers and product manufacturers multiple options to solve their embedded RTOS challenges with Zephyr. If your board is not supported out of the box, adding support for a new board is simple.



<https://www.zephyrproject.org>

ti





Parsing
With GitMiner

Creating File Histories from
GitMiner Data

Creating Disks from
File Histories

Placing Disks with PDS

**Parsing
With GitMiner**

**Parsing file tree for
Boards and Subsystems**

**Creating File Histories from
GitMiner Data**

**Creating Board histories and
Subsystem histories**

**Creating Disks from
File Histories**

Placing Disks with PDS

Parsing
With GitMiner

Parsing file tree for
Boards and Subsystems

Creating File Histories from
GitMiner Data

Creating Board histories and
Subsystem histories

Display visualization and
animation on a Web App

Creating Disks from
File Histories

Placing Disks with PDS

Creating animations from Disks

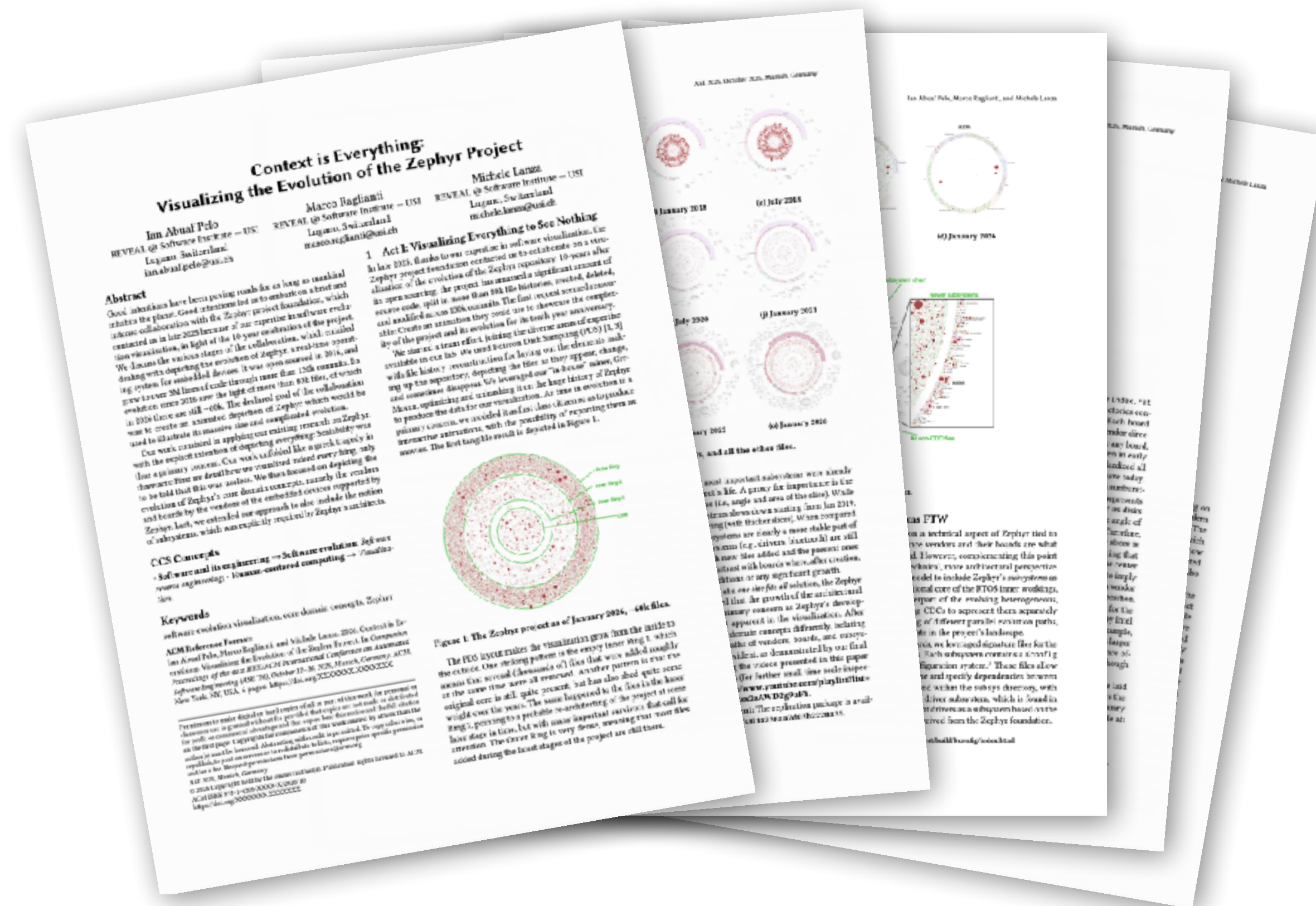
What did we learn?

- There is no *one size fits all* solution to evolution visualization



What did we learn?

- There is no *one size fits all* solution to evolution visualization
- Different maintainer roles require different points of view



What did we learn?

- There is no *one size fits all* solution to evolution visualization
- Different maintainer roles require different points of view
- A simple visualization can only show basic information when used on a large-scale project, more domain aware approaches are required

