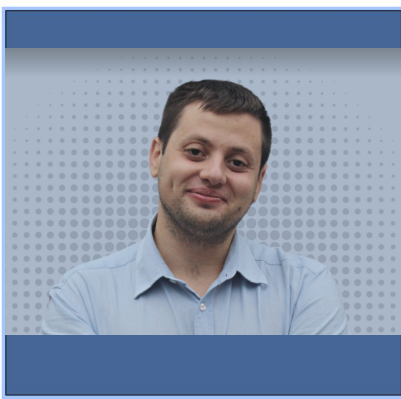


# Visualization Paradigm that will (potentially) replace Force Layouts



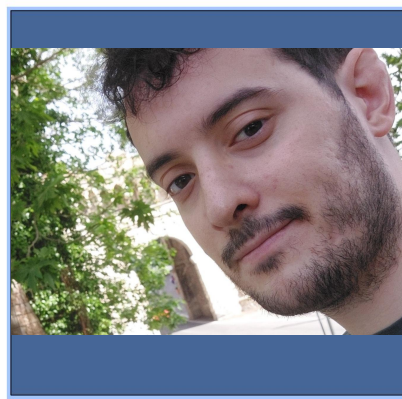


## Simone Ceccarelli

**Chapter Lead of Data Viz @LARUS**

Software engineer and passionate about  
**Algorithms, Data Visualization, Front-End  
& Parallel Programming**

**simone.ceccarelli@larus-ba.it**



## Tommaso Zazzaretti

**Front-end & Data Visualization Developer  
@LARUS**

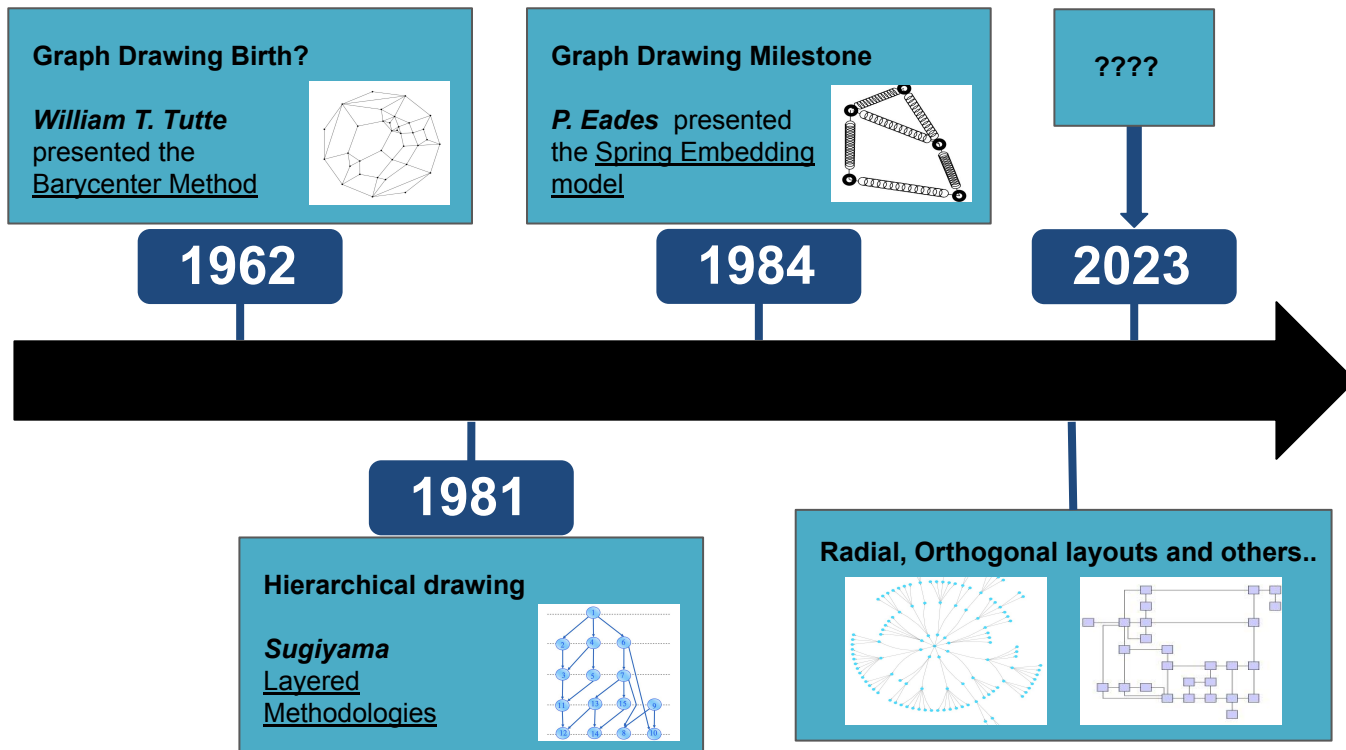
Software engineer and passionate about  
**Algorithms, Graph Visualization, Front-End**

**tommaso.zazzaretti@larus-ba.it**

# Drawing Graphs using AI



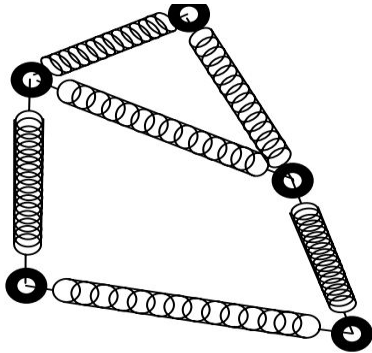
## A bit of history..



# Traditional Approaches

Several graphics visualization algorithms work by applying heuristics, models, geometric intuitions formulated by their creators, so as to achieve good visual results

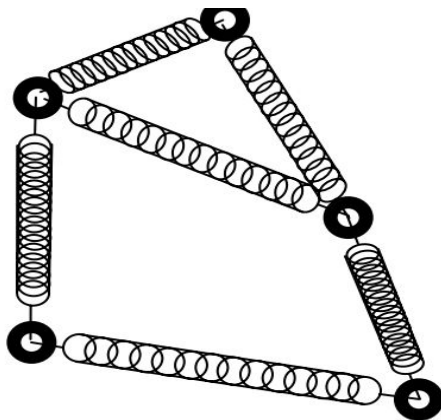
- Force directed approaches place their robustness in a metaphorical **application of a physical model** (Eades's Spring Embedding model)



# Qualities and limitations

## Benefits

- Easy to use
- Easy to implement
- Good Layout Quality
- Scalable



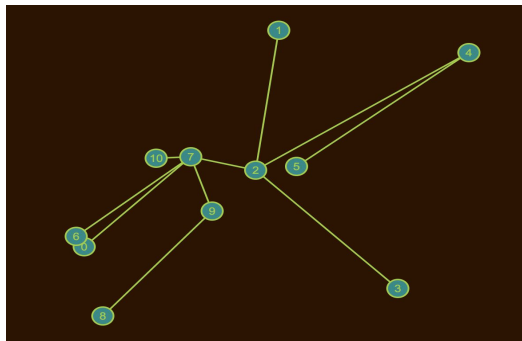
## Drawbacks

- There is no direct control over what graphic and design aspects the algorithm will improve.  
We only know that it will arrange the graph so as to find a balance between the forces

# Gradient Descent Layouts

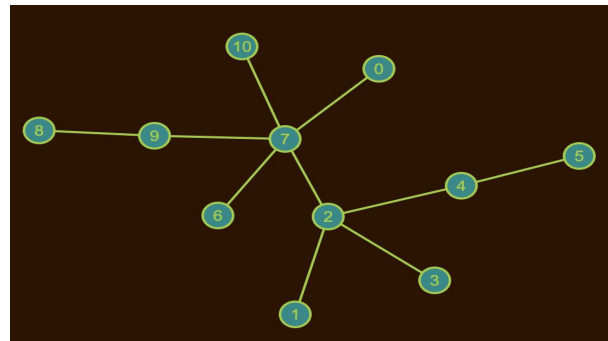
In recent years, AI techniques are increasingly used to solve complex problems.

In Graph Visualization, the use of AI optimization algorithms has led to the development of a new family of layouts based on **cost functions**



$f(G) = 4912.999$   
(Bad)

$$f : \mathbb{R}^{2|V|} \mapsto \mathbb{R}$$



$f(G) = 0.04$   
(Good)

optimization

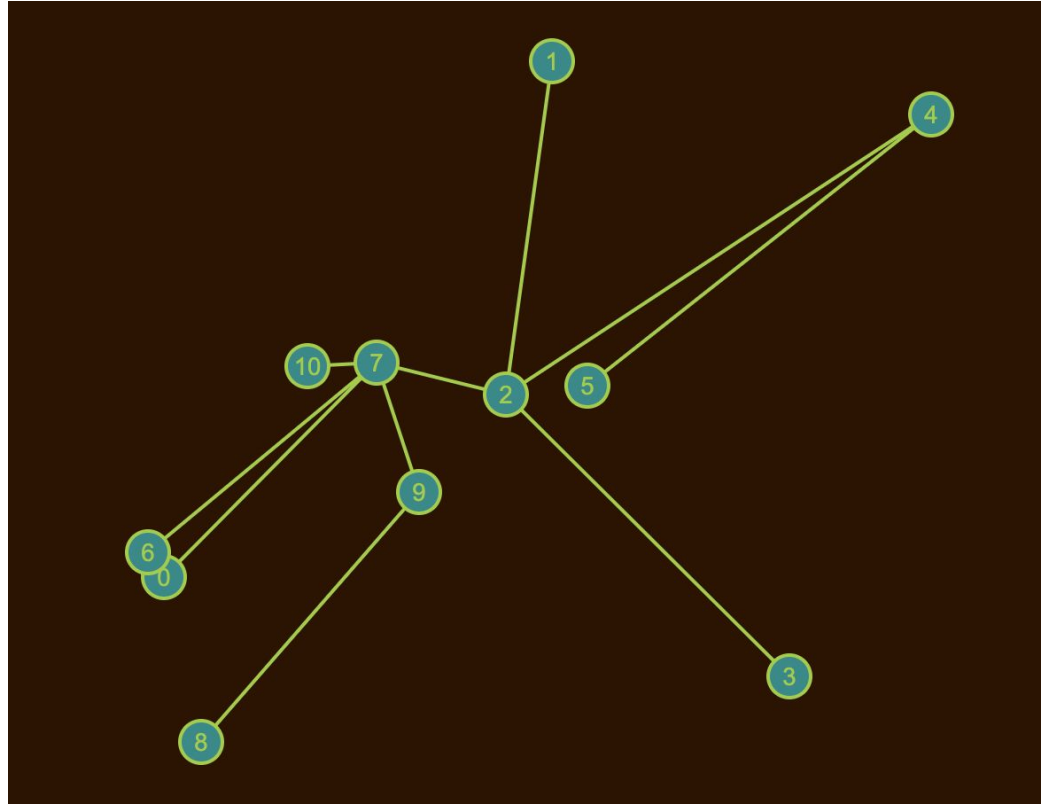


***How can we formulate cost functions related to graphs?***

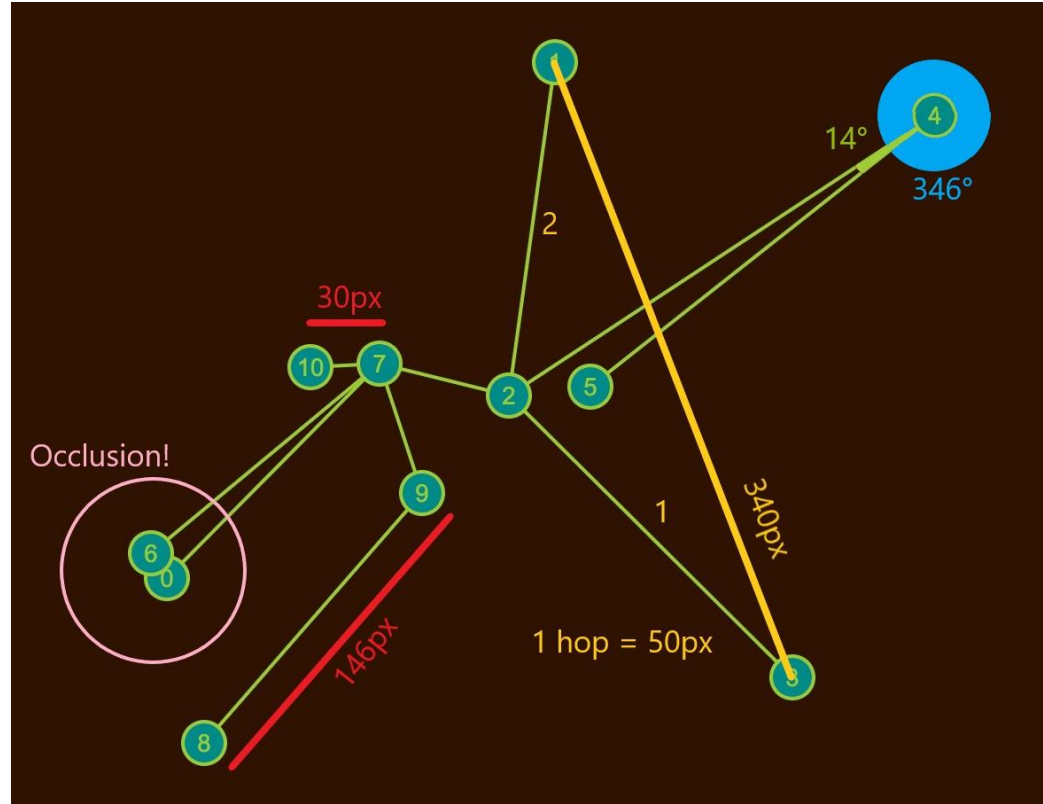


- Suppose you are observing a picture representing this graph

*What can you notice?*

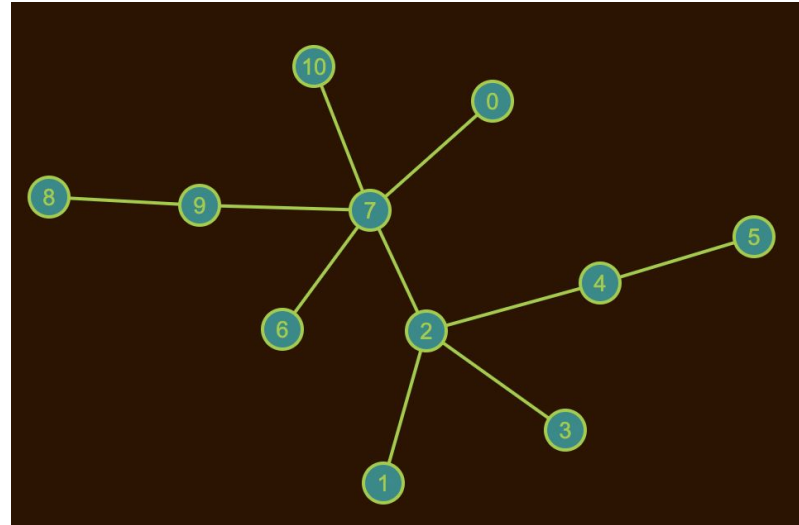


1. Topological distances not respected  
(1 hop = 50px,  
2 hop = 340px ?!)
2. High radial disparity
3. High euclidean disparity
4. There are overlapped knots



# Example: Stress Evaluation

- Given a graph  $G(V, E)$ , the Geodetic distance (or Topological distance) between 2 nodes **I** and **J** is defined as the number of hops that make up the shortest path between **I** and **J** within the graph
- Theoretically, it is not wrong to suppose that in an ideal layout, **the Euclidean distances between pairs of nodes should be similar to the topological distance**



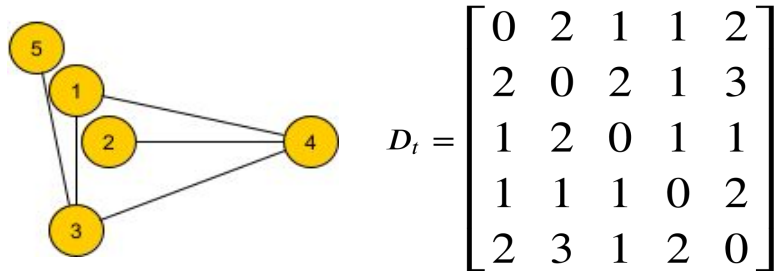
# Formulation: Cost Function

- **Empirical data source:**

*Shortest Path lengths*

- **What is our Goal?**

Make the 'Euclidean distances'  
similar to Topological distances

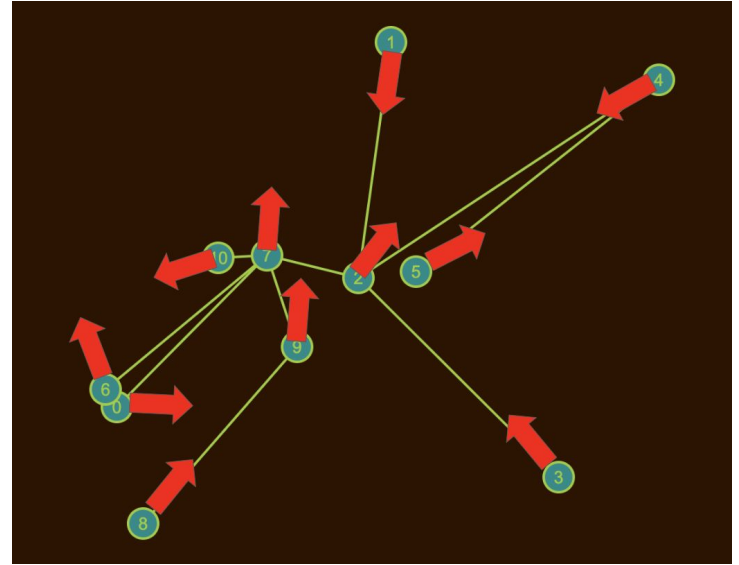
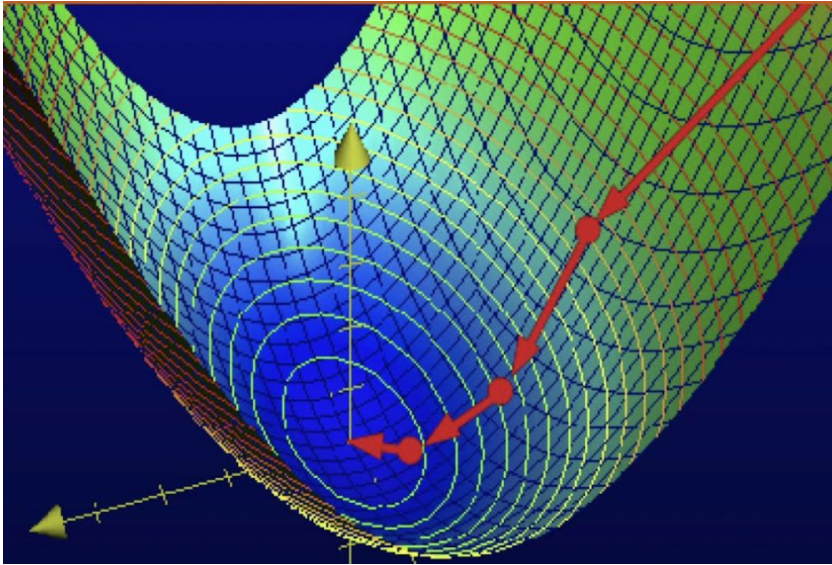


- Summing over all the **ij\_pairs** we can define a function  $f : \mathbb{R}^{2|V|} \mapsto \mathbb{R}$  in order to have a quality measure of the **Stress** of a current Layout

$$Stress = \sum_{(i,j) \in V, i \neq j} Q_{ij} = \sum_{(i,j) \in V, i \neq j} \left( \frac{\|V_i - V_j\|_2 - d_{ij}}{d_{ij}} \right)^2$$

# Optimization

- We defined a function  $f : \mathbb{R}^{2|V|} \mapsto \mathbb{R}$  whose variables are the x,y coordinates of the graph nodes.
- This means that *each point of the domain is a possible arrangement of the graph nodes on the screen*
- **Optimizing function means move the nodes!**



# References

- <https://www2.cs.arizona.edu/~kobourov/GD20-gdgd.pdf>
- <http://hdc.cs.arizona.edu/~mwli/graph-drawing/>

# Scalability & Portability



## Layout Composition

- A layout must be analyzed in terms of
  - Effectiveness
  - Efficiency
- Nowadays everything is characterized by **big data**
- What happens by increasing the number of nodes?



# Scalability

Layout execution times with a single cost function

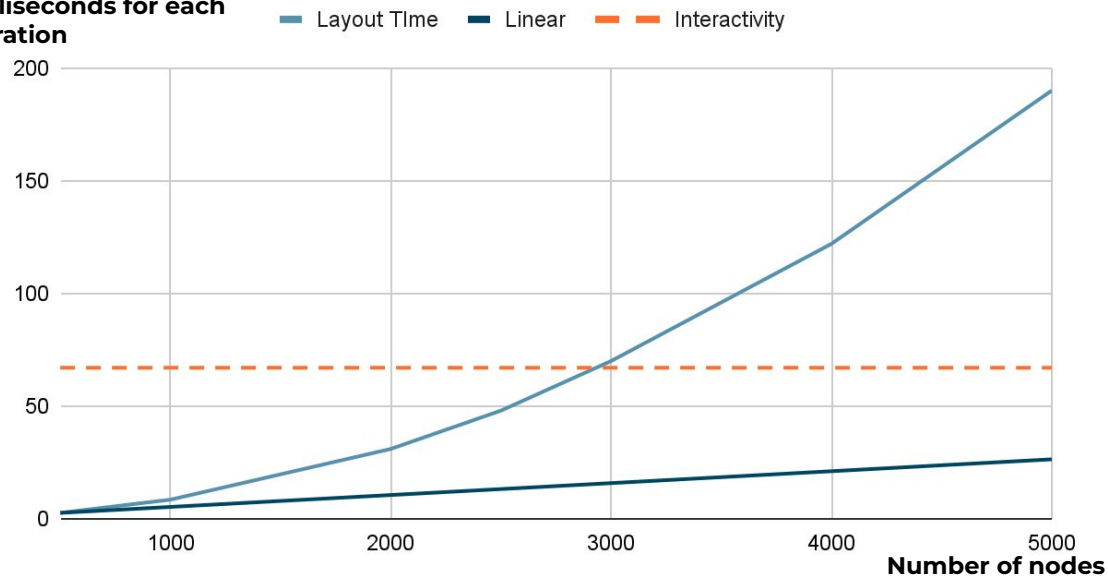
Considerations need to be made

- The iterations needed to arrive at convergence
- The CPU will be at 100% all the time blocking interactions
- Negligible rendering costs with the GPU

The edges number is about  $2N$

## Points scored

Milliseconds for each iteration



## Target

- Being able to display as many nodes and edges as possible with CPU usage & parallel programming
- Offer a portable WEB solution

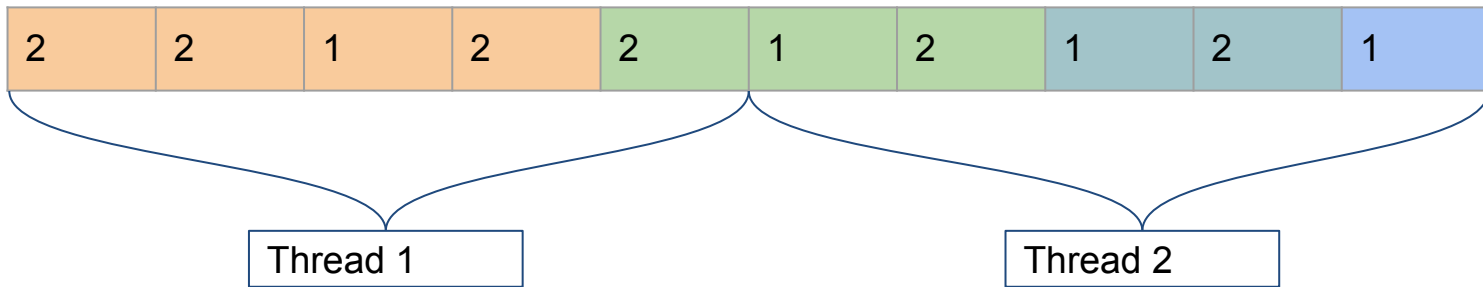
## Gradient Descent Layout

- Only **stress** is considered
  - Make the **Euclidean** distances as similar as possible to the **topological** distances

### Algorithm

1. Creation of the **topological matrix**
2. Until I achieved a good result
  - a. For each node **U**
    - i. Visit every other **V** node
      1. Calculation of the ratio between **Euclidean** and **topological**
      2. Calculation of the partial derivatives of the pair **U** and **V**
    - ii. Calculate the current inertia
    - iii. Update the positions via **gradient** and **inertia** (**ADAM** Optimization Algorithm)
  - b. For each node **U**

## Thread Load Balancing



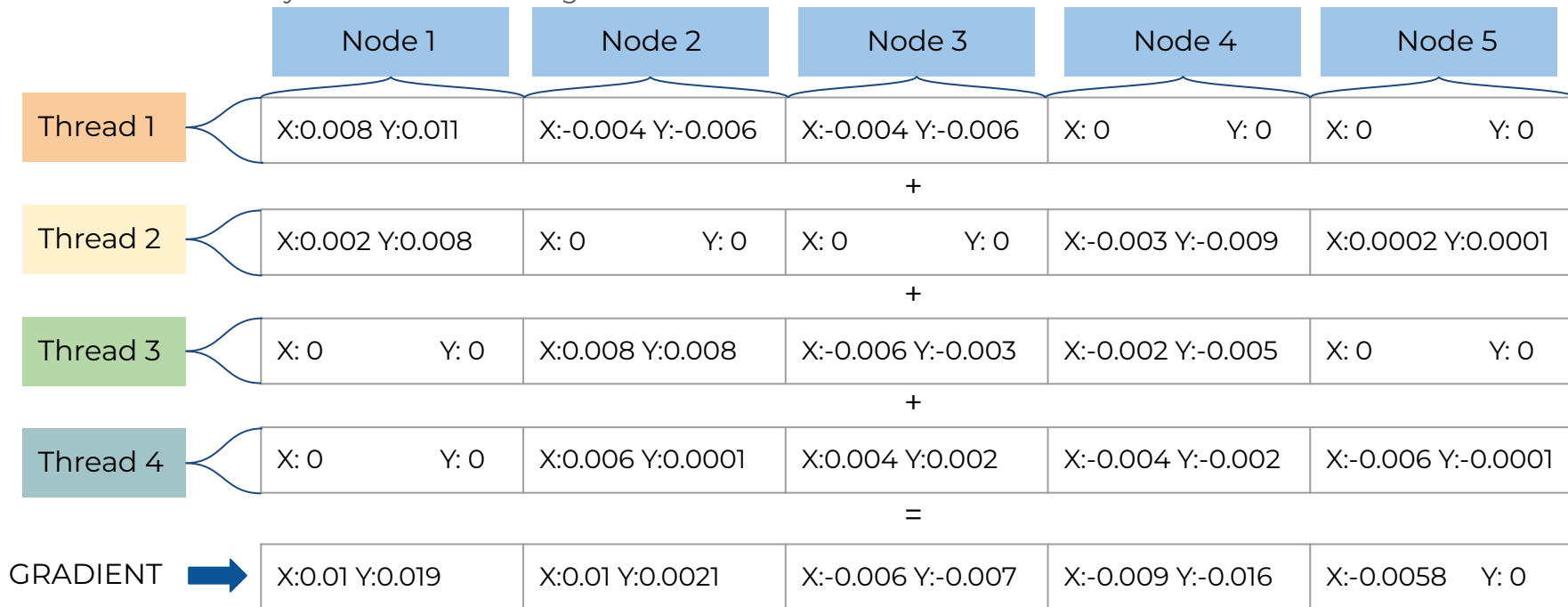
	N0	N1	N2	N3	N4
N0	0	2	2	1	2
N1	2	0	2	1	2
N2	2	1	0	1	2
N3	1	1	1	0	1
N4	2	2	2	1	0

Topological Matrix

- Work on a triangle of the matrix
  - Computational savings
  - Minimize the data to be sent to each thread
- Create an array of the computations and split it into threads

# Parallelization

- Each thread will calculate part of the gradient
- When everyone is done the real gradient is created



# Scalability

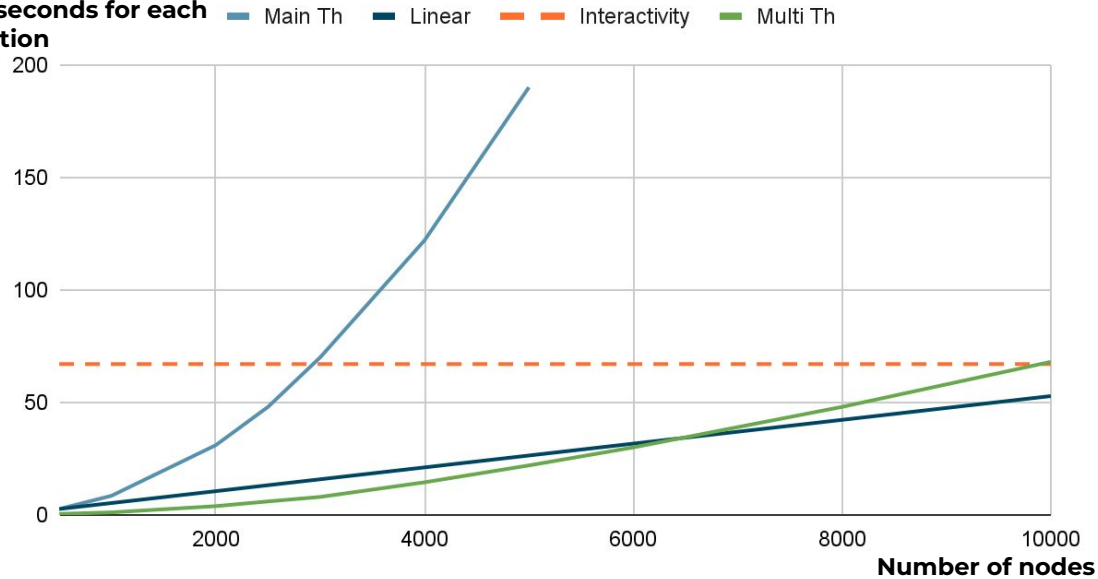
## Main thread VS multi thread

- Implemented in TypeScript
- 5 threads were used for the calculation
- Edges are about  $2N$

NB: in reality the times are better, part of the overhead is due to the calculation of the times

## Points scored

Milliseconds for each iteration



# Results in TypeScript

**Only the main Thread:** It reaches its limit with less than **3k nodes**

**Multi Thread:** With 5 threads it can visualize a graph with **10k nodes** and **20k edges**

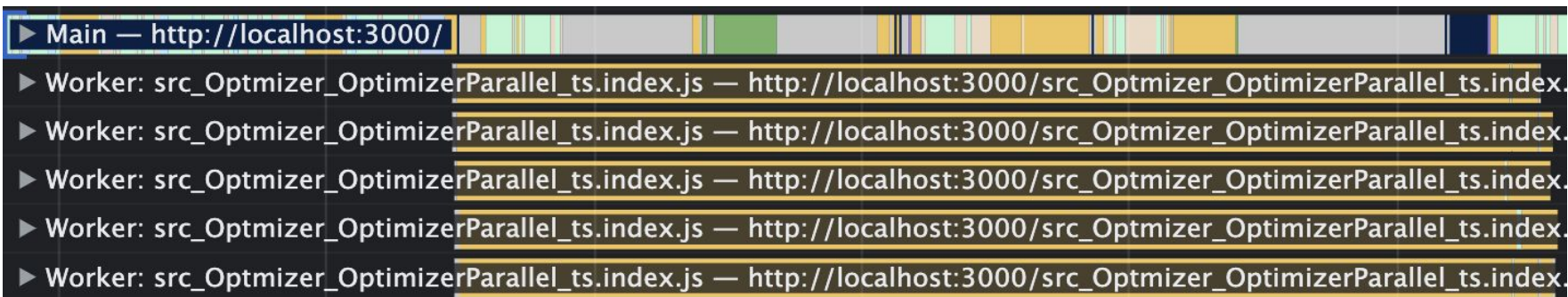
## Speed-up

- Only the main thread with **5k nodes** and **~10k edges** take **~190 milliseconds**
- Multi th with **1 thread** su **5k nodes** and **~10k edges** take **~125 milliseconds**
- Multi th with **5 thread** su **5k nodes** and **~10k edges** take **~22 milliseconds**

**SPEED-UP main vs 5 =  $190 / 22 \approx 8.6$**  How is it possible?

**SPEED-UP 1 vs 5 =  $125 / 22 \approx 5.6$**

In web applications the main thread has to do many things and we leave it free for other interactions.



# Results in TypeScript

- Layout over a random generated graph with 8k nodes:  
[https://drive.google.com/file/d/1LjSsUGAPewqspZ3a1kvu1xVc1l4La04a/view?usp=share\\_link](https://drive.google.com/file/d/1LjSsUGAPewqspZ3a1kvu1xVc1l4La04a/view?usp=share_link)



## Future Works

- Parallel Layout over GPUs
  - To increase at least an order of magnitude
  - Analyze if they continue to be effective on large scales
  - Every PC has a limited number of CPUs and these layouts need more features to optimize



Thanks for your curiosity!

Simone Ceccarelli

Chapter Lead of data Viz @LARUS

**`simone.ceccarelli@larus-ba.it`**

Tommaso Zazzaretti

Frontend Developer @LARUS

**`tommaso.zazzaretti@larus-ba.it`**

