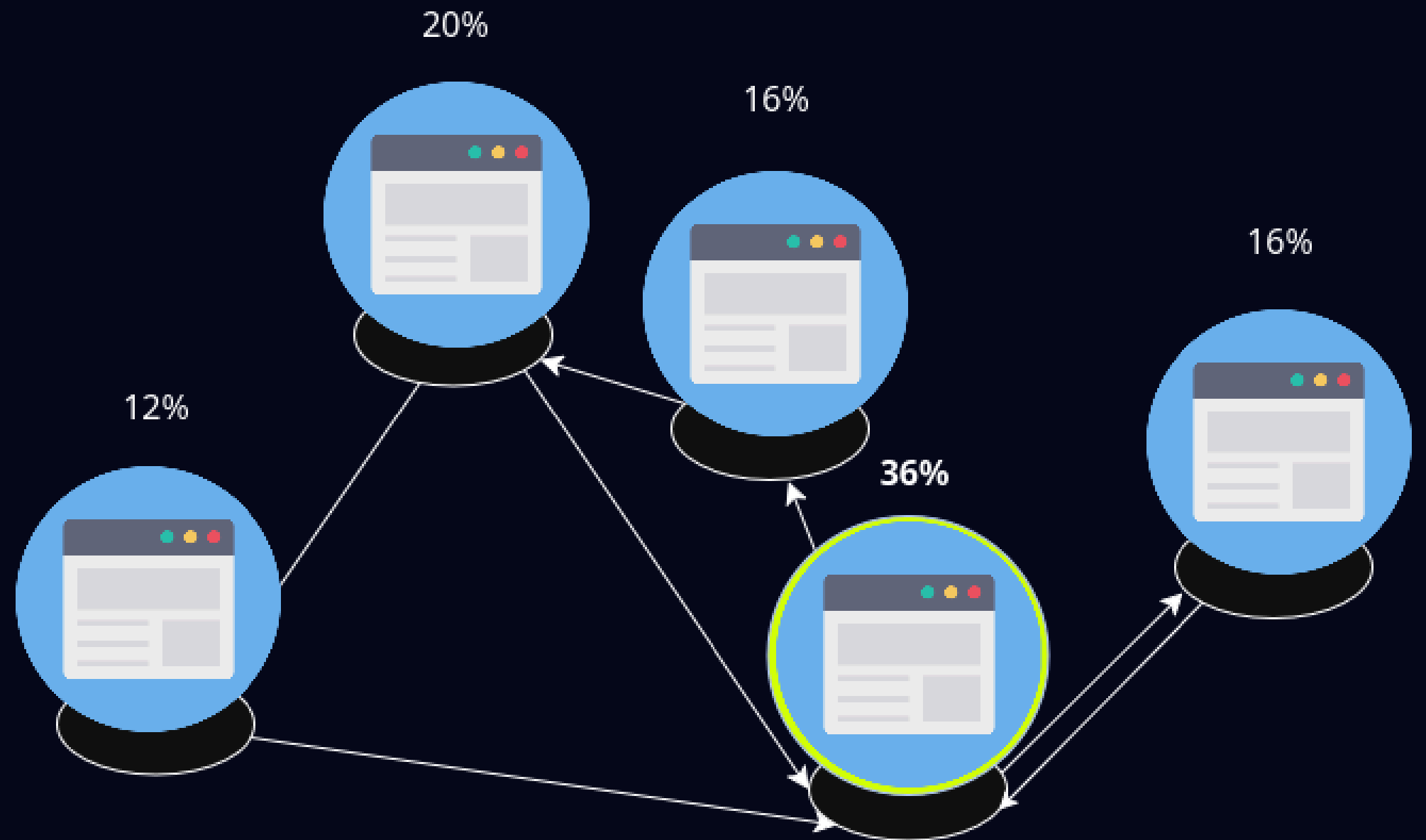


# PageRank

BEAUTIFUL SOUP

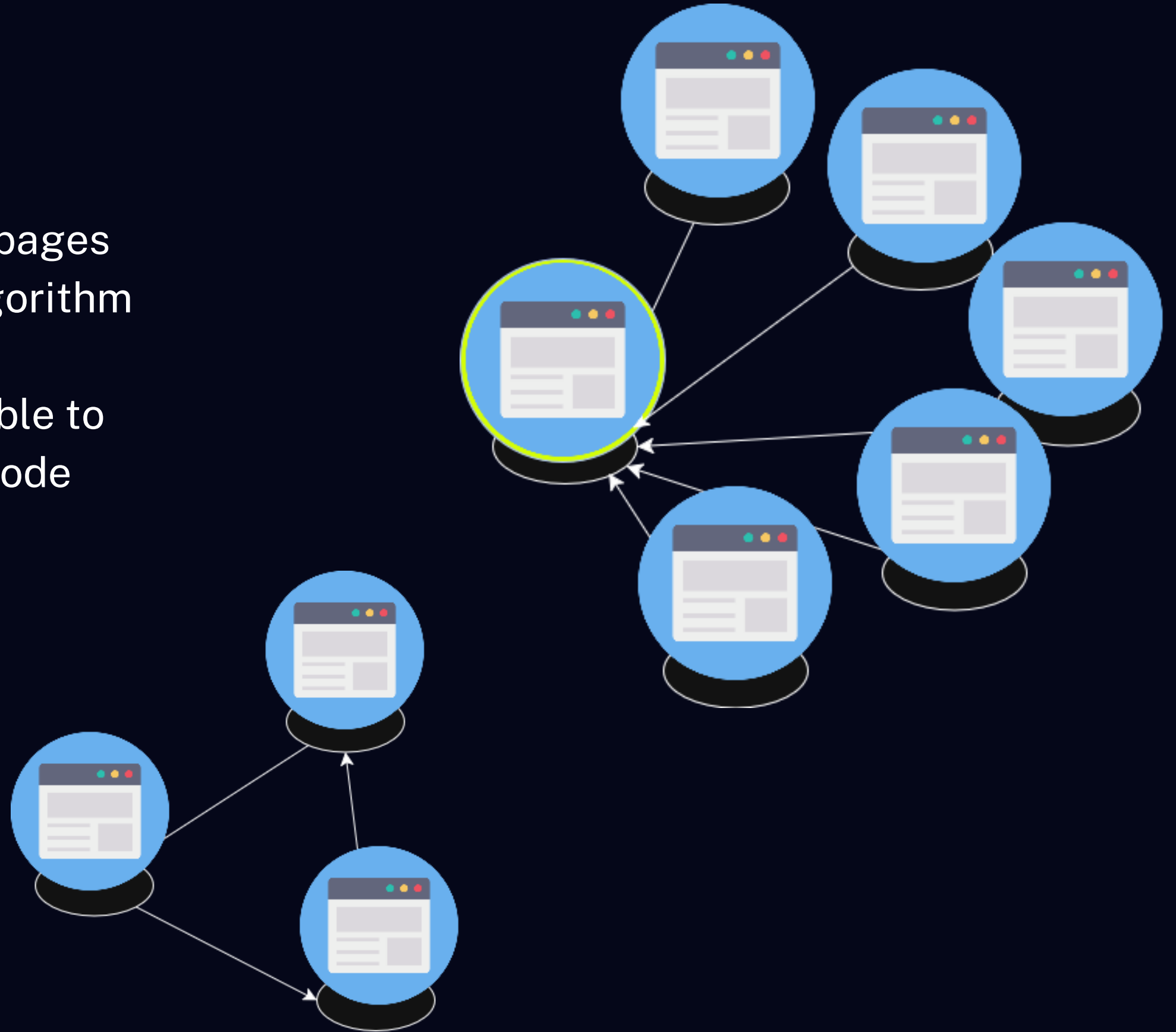
# PageRank

- Algorithm that ranks nodes based on their relevance
- A node is relevant if many (and/or relevant) pages link to it



# Random Surfer

- The Dead End Problem: "Dangling nodes" (pages that don't link to other pages) break the algorithm
- Solution: we assume that it is equally possible to jump to any other page from the dangling node



# Page rank formula

$$r^{(k+1)} = \alpha \times \left( P \times r^{(k)} + \frac{\text{dangling\_mass}(k)}{n} \right) + \frac{1 - \alpha}{n}$$

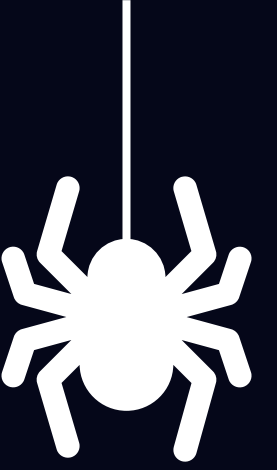
$r^{(k)}$  - page rank at iteration k

$P$  - transition matrix

$\alpha$  - damping factor (probability of following the link instead of randomly jumping to a different node – helps with convergence)

$\text{dangling\_mass}(k)$  - sum of PageRank values of all nodes that have no outgoing edges

$n$  - number of nodes



**What did we do  
and HOW?**

# Web API: Search & Live PageRank

## the WHAT

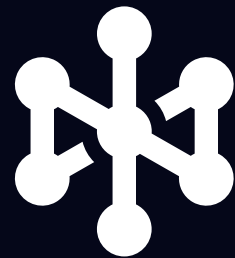
- **Query Search:**
  - Execute query search using TF-IDF and PageRank
- **Upload Graph of edges:**
  - Run PageRank on any uploaded graph
- **Search URL:**
  - Crawl a URL, build a graph, run PageRank live

# Architecture

## the HOW



**Concurrent web crawler  
(multi-threaded)**



**Graph construction  
(URL normalization +  
language filtering)**



**GPU-accelerated PageRank  
(CUDA cluster)**

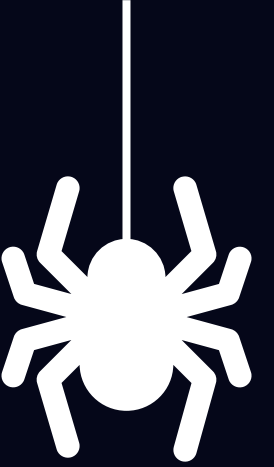


**TF-IDF  
word search engine**

stay tuned for more details on implementation..

# Crawler and Data Collection

Fast, Smart & Language-Aware



## Concurrent Fetching

Multiple pages fetched in parallel  
using a thread pool  
Faster graph discovery

## URL Normalization & Clean Text Extraction

Remove fragments, unify, slashes,  
lowercase hosts

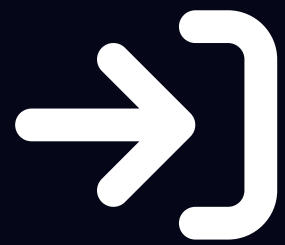
## Language Filter

Remove fragments, unify, slashes,  
lowercase hosts



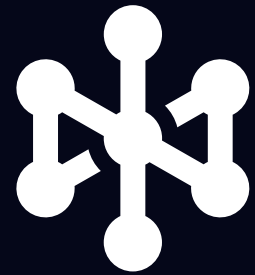
# GPU PageRank Pipeline

## CUDA cluster Integration



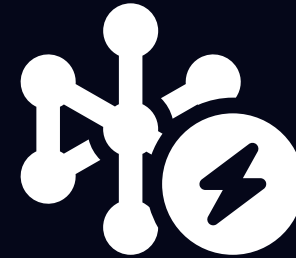
**Input**

Crawled graph  
(edges.txt)



**Algorithm**

CSR-based sparse matrix  
PageRank



**Efficiency**

GPU acceleration with CPU  
fallback



**Output?**

Sorted PageRank  
scores

# Smart Ranking TF-IDF × PageRank

## TF-IDF algorithm for word search

- computes the similarity between the query and the web page  
based on how often the words from the query appear in the text →  
takes into consideration how rare is the word
- We even have a GPU-optimized version using CSR + CuPy for large-scale search.

$$Score = 0.8 \cdot TFIDF + 0.2 \cdot PageRank$$

# Why is our solution cool?

**It's not just requests + BeautifulSoup**

TF-IDF Engine x PageRank  
using Cosine Similarity

Crawler using ThreadPoolExecutor

Robust GPU/CPU Strategy

the REAL Brin-Page

computing TOP-K nodes

CUDA kernels for sparse matrix-vector multiplication  
and damping +teleportation step

# Management Goal & Scalability

## Dynamic Architecture

works for tiny graphs, medium Web domains, or millions of nodes

## GPU Acceleration

CUDA kernels enable significant speedups for large datasets

## Real-World Extension

supports Wikipedia-scale crawling with configuration changes

**University Websites**

**AI Assistants Search**

# Our Demo

## Try the Demo

Experience PageRank in action. Search for a URL, upload a file or explore by keywords.

↔ URL

⬆ Upload

🔍 Search

🔗 Custom Graph

Enter Website URL

https://tum.de

▶ Analyzing...

We'll crawl a small neighborhood around this URL and calculate PageRank scores for the discovered pages.

- PageRank results will appear here

Enter a URL and click Analyze to see the most important pages in its local link graph.

N

Crawler start: [https://en.wikipedia.org/wiki/Graph\\_\(discrete\\_mathematics-max-pages-200-ENG-language\)](https://en.wikipedia.org/wiki/Graph_(discrete_mathematics-max-pages-200-ENG-language))

# Thank You!

**“ThreadPoolExecutors:  
they don’t complain, they don’t get tired — they just take tasks and finish them.  
Be like that.  
Queue the work, keep it simple, no drama.  
One task at a time until the job’s done.”  
– Timo Robrecht**