



To RAG or not to RAG

Piotr Przybył

GOTO 2026

\$ whoami

Piotr Przybył

Senior Developer Advocate



Java Champion
Docker Captain
Testcontainers Champion
@piotrprz
@piotrprz.bsky.social
@piotrprz@mstdn.social



Piotr Przybyl

@piotrprz.bsky.social

Followers

887

Following

260

Posts

105

Packs

0

Piotr Przybyl: The Renaissance Developer of Bluesky

An edutainment powerhouse blending programming wisdom with genuine humor.

" If Java were a garden, Piotr would be both the gardener and the talking scarecrow. "

85%

Authenticity

80%

Thought Leader

Key Topics

95%

Java

90%

Community Engagement

\$ who are you

**CAVEAT
AUDITORES**

Stream Gatherers in Java refer to techniques and tools for collecting and combining elements from a stream into various data structures or formats. In Java, the Stream API provides powerful capabilities for processing sequences of elements, and one of the key operations is collecting the results of these streams.

The most common and flexible way to gather elements from a stream is by using the `Collectors` utility class, which provides a wide variety of predefined collector implementations.

Key Collectors and Gathering Techniques

1. To List:

Collecting elements into a `List`.

java

Copy code

```
List<String> list = stream.collect(Collectors.toList());
```

2. To Set:

Collecting elements into a `Set`.

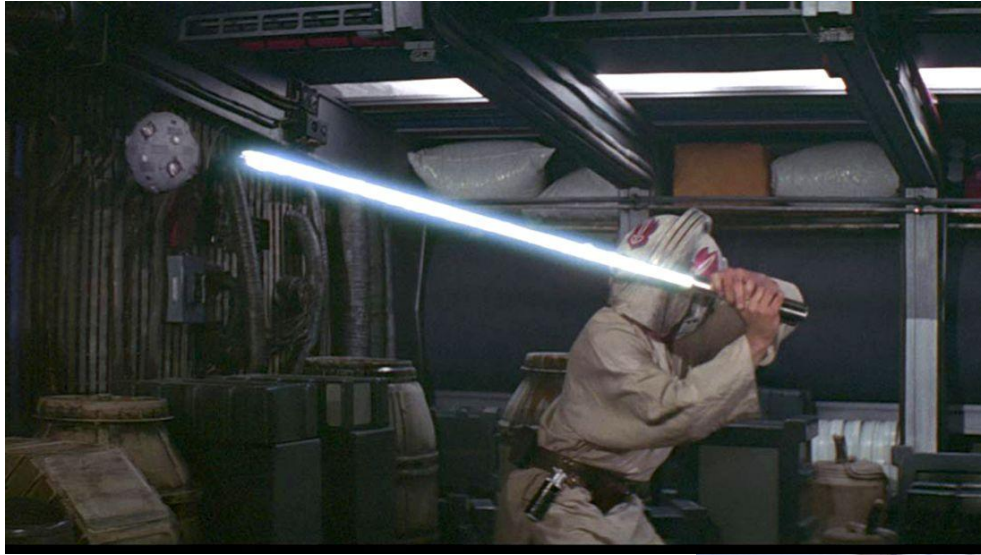
java

Copy code

```
Set<String> set = stream.collect(Collectors.toSet());
```

Not really
[stream.gather\(...\)](#)

Problem: training vs actual use-case



Reasoned about Stream Gatherers in Java for 18 seconds >

"Stream Gatherers" isn't an official term in the Java language, but it's often used informally to refer to the mechanism for collecting or accumulating elements from a stream—namely, the use of **Collectors** in the Java Streams API.

How It Works

When you work with streams (introduced in Java 8), you typically perform intermediate operations (like filtering or mapping) and then finish with a terminal operation. The most common terminal operation for gathering results is the `collect()` method. Under the hood, this method uses a **Collector** to:

- **Accumulate:** Start with an empty mutable container.
- **Accumulate Elements:** Add or "gather" each stream element into that container.
- **Combine:** Merge multiple containers (especially useful in parallel processing).
- **Finish:** Optionally perform a final transformation on the container to produce the desired result.

Built-In Collectors

Java provides a variety of built-in collectors via the `java.util.stream.Collectors` utility class, such as:

- **toList() / toSet():** Gather stream elements into a `List` or `Set`.
- **toMap():** Gather elements into a `Map` by specifying key and value mapping functions.
- **groupingBy():** Gather elements into groups based on a classification function.
- **partitioningBy():** Partition elements into two groups based on a predicate.
- **joining():** Gather elements into a single `String`, typically  by concatenating them.

Ask anything



Search



Deep research



Thought for a few seconds >

Stream Gatherers are a new, preview extension to the Java Stream API (introduced in Java 22 via JEP 461) that let you define custom intermediate operations. In essence, they let you transform a stream of elements in more flexible ways than the built-in operations (like map or filter) allow.

Key Concepts

- **Custom Intermediate Operations:**

Unlike collectors (which are used as terminal operations), gatherers work during the stream's intermediate phase. They enable transformations that can be one-to-one, one-to-many, many-to-one, or many-to-many.

- **Stateful Processing:**

A gatherer is parameterized as `Gatherer<T, A, R>`, where:

- `T` is the input element type,
- `A` is a mutable state (which can help track previously seen elements or accumulate data),
- `R` is the output element type.

This design lets you maintain context between elements (e.g., for grouping or deduplication) and even short-circuit processing when needed.

- **Four Functional Components:**

A gatherer can be built by defining:

- **Initializer:** Provides the initial state.
- **Integrator:** Processes each element, updating the state and optionally pushing output downstream.
- **Combiner:** (Optional) Merges states when processing streams in parallel.

Ask anything



Search



Deep research



Yo! How shall I say "Retrieval Augmented Generation" in Danish, Swedish and Polish? Gimme a bullet point list, be concise!

-  Generering understøttet af informationssøgning
-  Generering förstärkt med informationssökning
-  Generowanie wspomagane wyszukiwaniem

Retrieval Augmented Generation

**The Right
Answer**



```
SELECT * FROM my_search_table
```

```
WHERE
```

```
content LIKE '%milk%'
```

```
AND
```

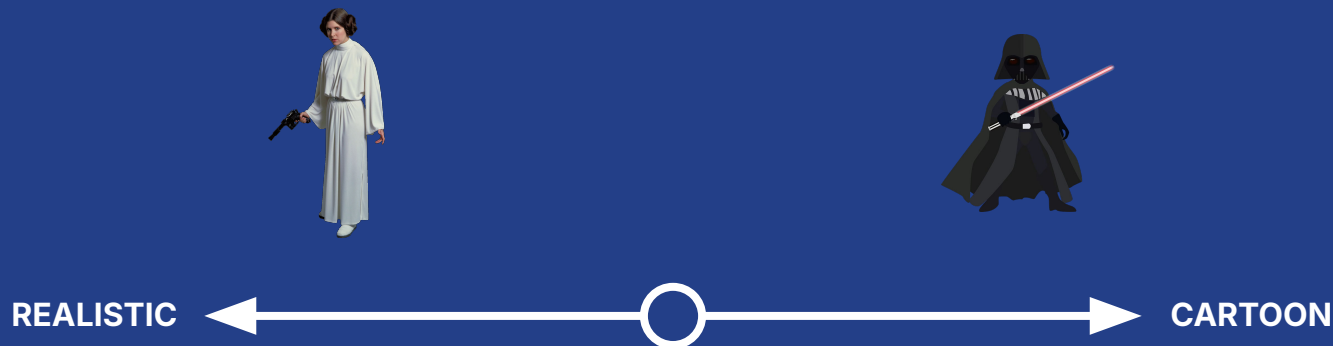
```
content LIKE '%chocolate%'
```

```
milk chocolate
```

```
chocolate milk
```

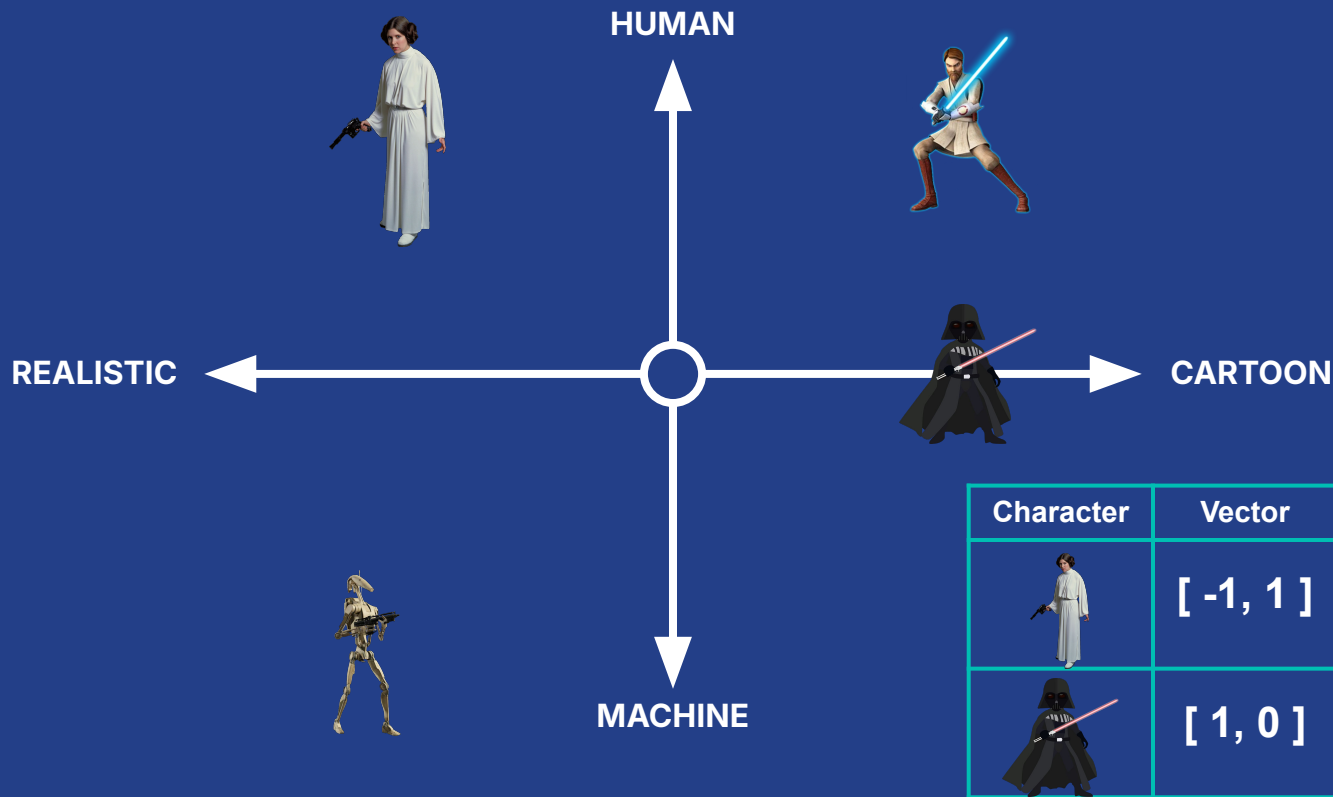
Embeddings represent your data

Example: 1-dimensional vector

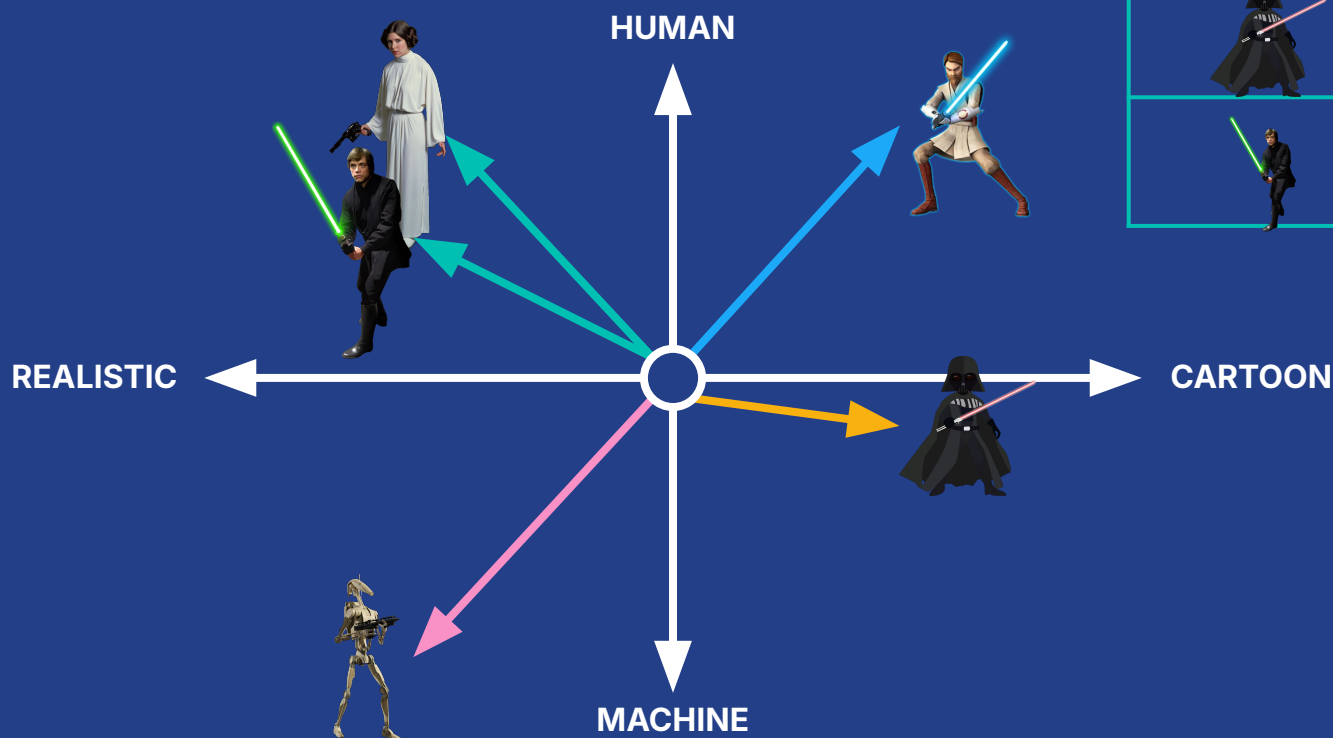


Character	Vector
	$[-1]$
	$[1]$

Multiple dimensions represent different data aspects



Similar data is grouped together



Character	Vector
	$[-1.0, 1.0]$
	$[1.0, -0.1]$
	$[-1.0, 0.8]$

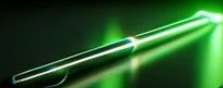
```
curl -N -f -X POST "${DOCKER_MODEL_RUNNER}/embeddings" -d '{  
  "model": "'${EMBEDDING_MODEL}'",  
  "input": "Quick brown fox jumped over a lazy dog"  
}
```

```
{  
  "embedding": [  
    0.04753968119621277,  
    -0.004536565393209457,  
    -0.027034679427742958,  
    -0.029689135029911995,  
    0.0623319037258625,  
    -0.04296858608722687,  
    0.047584425657987595,  
    ...  
  ]  
}
```

```
curl -N -f -X POST "${DOCKER_MODEL_RUNNER}/embeddings" -d '{  
  "model": "'${EMBEDDING_MODEL}'",  
  "input": "Quick brown fox jumped over a lazy programmer"  
}'
```

```
{"embedding": [  
-0.0022155670449137688,  
-0.07280080020427704,  
0.042364563792943954,  
0.0734516903758049,  
0.0579221136868,  
0.013278582133352757,  
0.06784238666296005,  
...
```



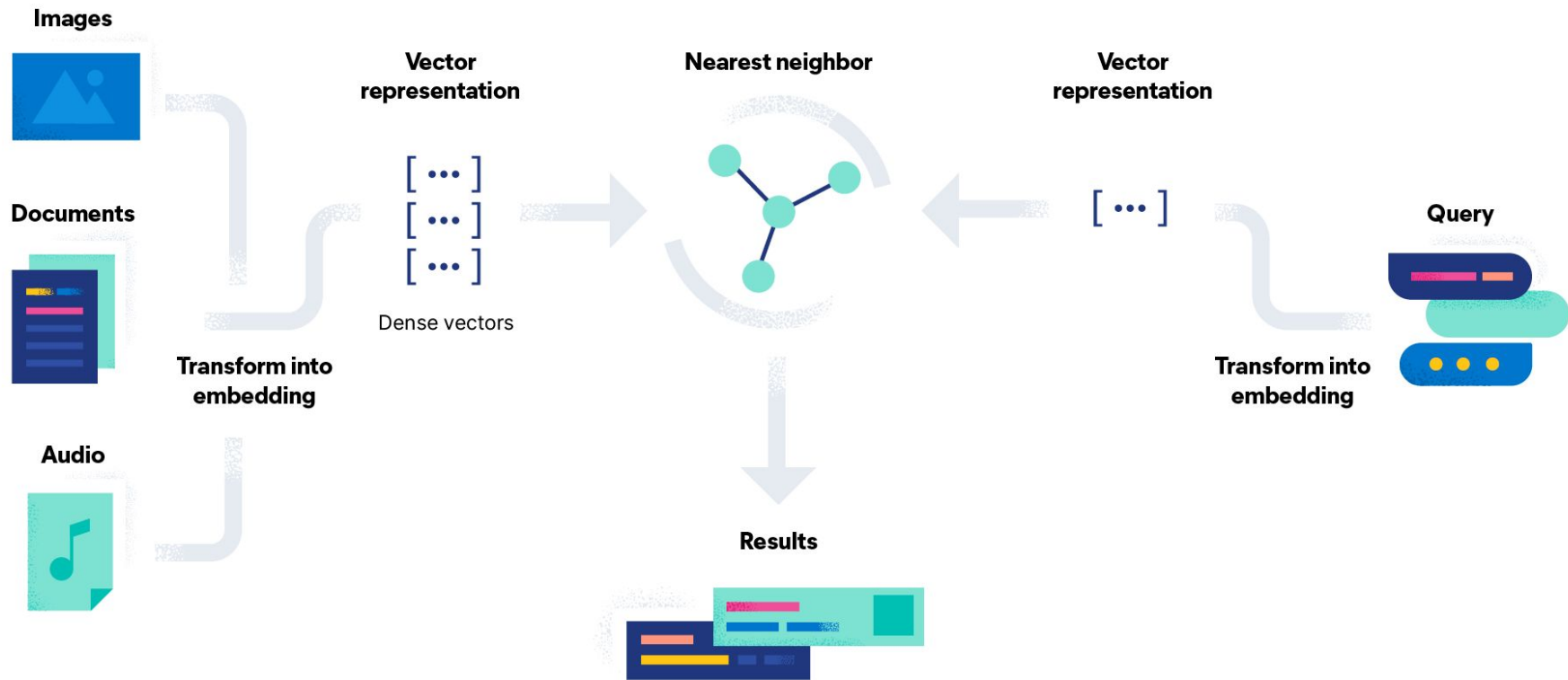


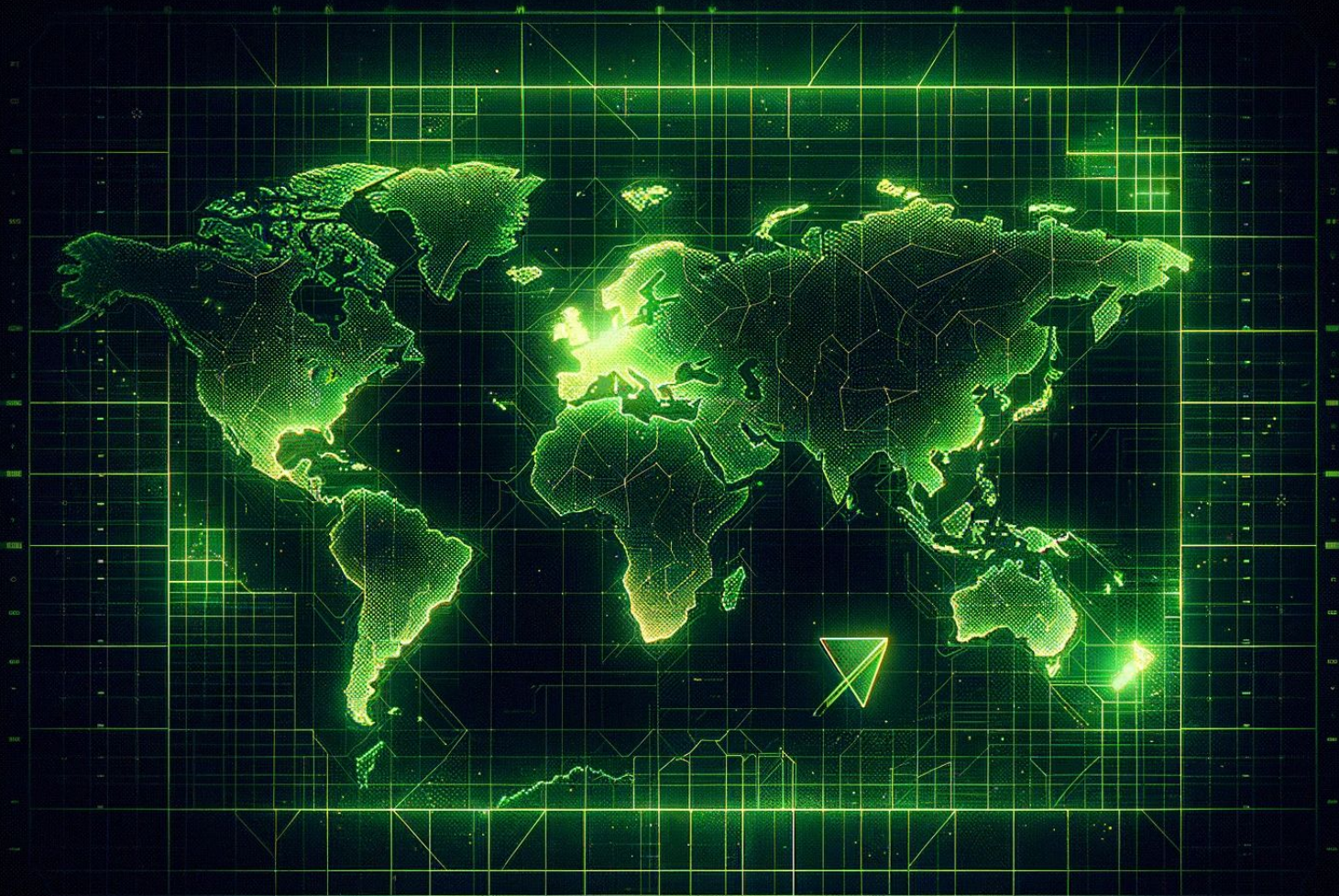


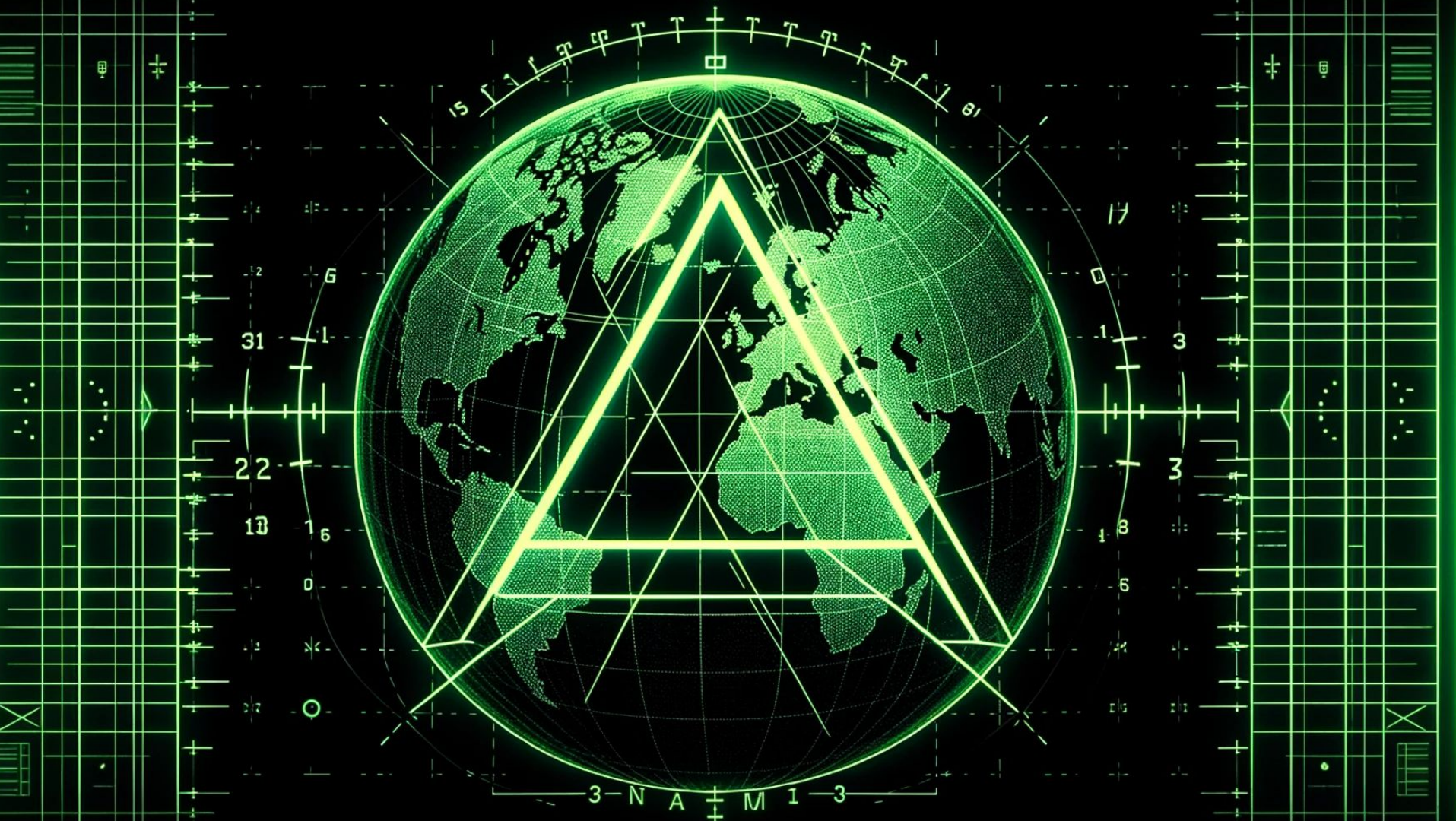
elasticsearch

Lucene

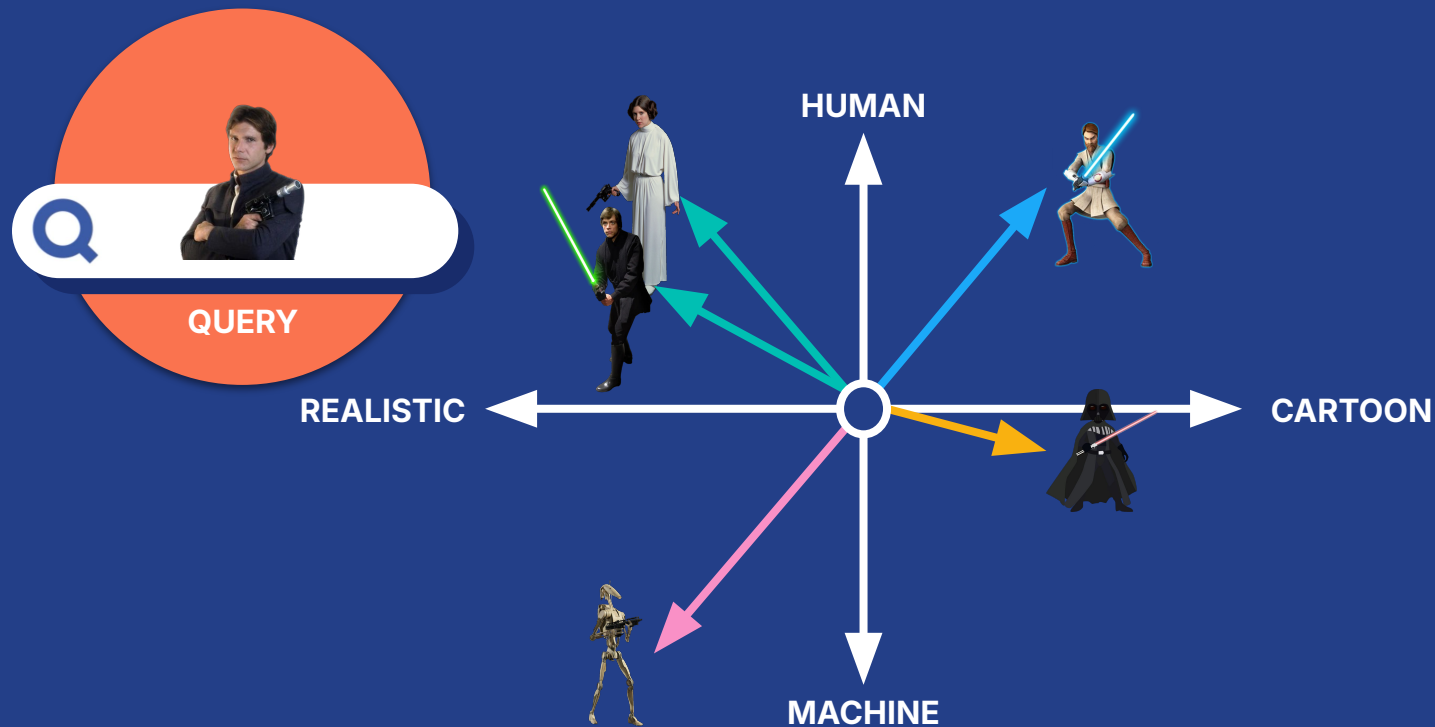
Vector Search





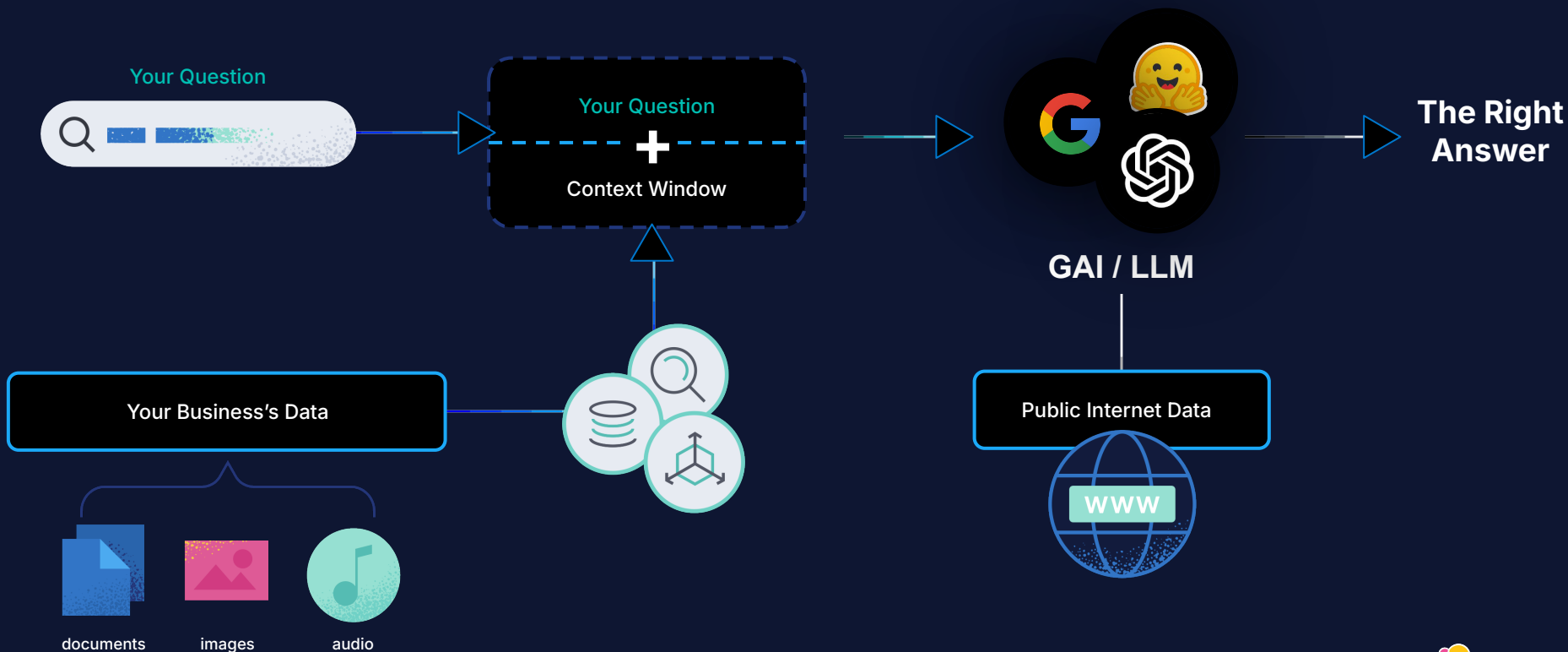


Vector search ranks objects by similarity (relevance) to the query



Relevance	Result
Query	
1	
2	
3	
4	
5	

Retrieval Augmented Generation



Should I always RAG?

Probably not ;-)

What about MCP?

Retrieval-Augmented Generation (RAG) and **Model Context Protocol (MCP)** both help large language models (LLMs) access external information, but they do so in *fundamentally different* ways.

RAG improves an LLM's answers by **retrieving relevant text or data** from external sources (like Elasticsearch, vector databases, or document stores) and **injecting it into the model's prompt**. The model never executes code or calls APIs - it simply *reads and reasons* based on retrieved content.

"Fetch the right info, then explain it."

MCP defines a **secure protocol** for LLMs to **interact with real tools, APIs, and systems** - not just to read data, but to **take action**.

Through MCP, the model can make structured requests (e.g., "query this DB," "send this email," "fetch live metrics") using connectors that control what's exposed.

"Connect safely and act through trusted tools."

Which one to choose?

Use RAG when:

- You only need the model to **read information** and generate answers.
- Your data is **unstructured** (documents, FAQs, notes).
- You want minimal security risk - no code execution, just retrieval.

Use MCP when:

- You need the model to **perform actions** or access **live structured data**.
- You require **real-time integration** with tools, databases, or APIs.
- You can enforce **strict security boundaries** (auth, scopes, auditing).

Why not both?

MCP can serve as the “retrieval layer” inside a RAG system

RAG = the *pattern* (retrieve, augment, generate).

MCP = the *protocol* enabling that retrieval safely.

Together they form a powerful combo:

MCP-powered RAG — secure, modular, and production-grade.





Topics

Drafts

More

Categories

Announcements

Elastic Stack

Elastic Search

Elastic Observability

Elastic Security

All categories

Tags

filebeat

docker

elastic-stack-security

metricbeat

elastic-stack-alerting

categories

tags

Categories

Latest

Top

Category



elasticstack.slack.com/signup#/domain-signup



Announcements



Release and security bits about all that we think will be

■ S
■ C

Elastic Stack



Elastic Stack
- all
in
vis
you
cat
E

■ Elasticsearch
■ Elastic Agent



Elastic

Members

109,798

Groups

137

Countries

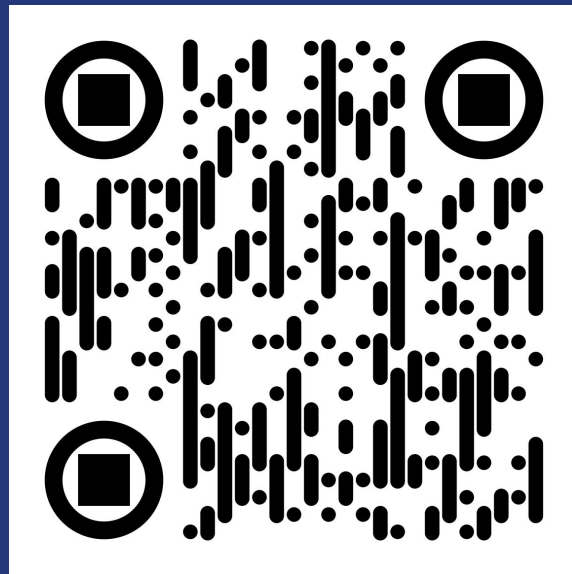
54



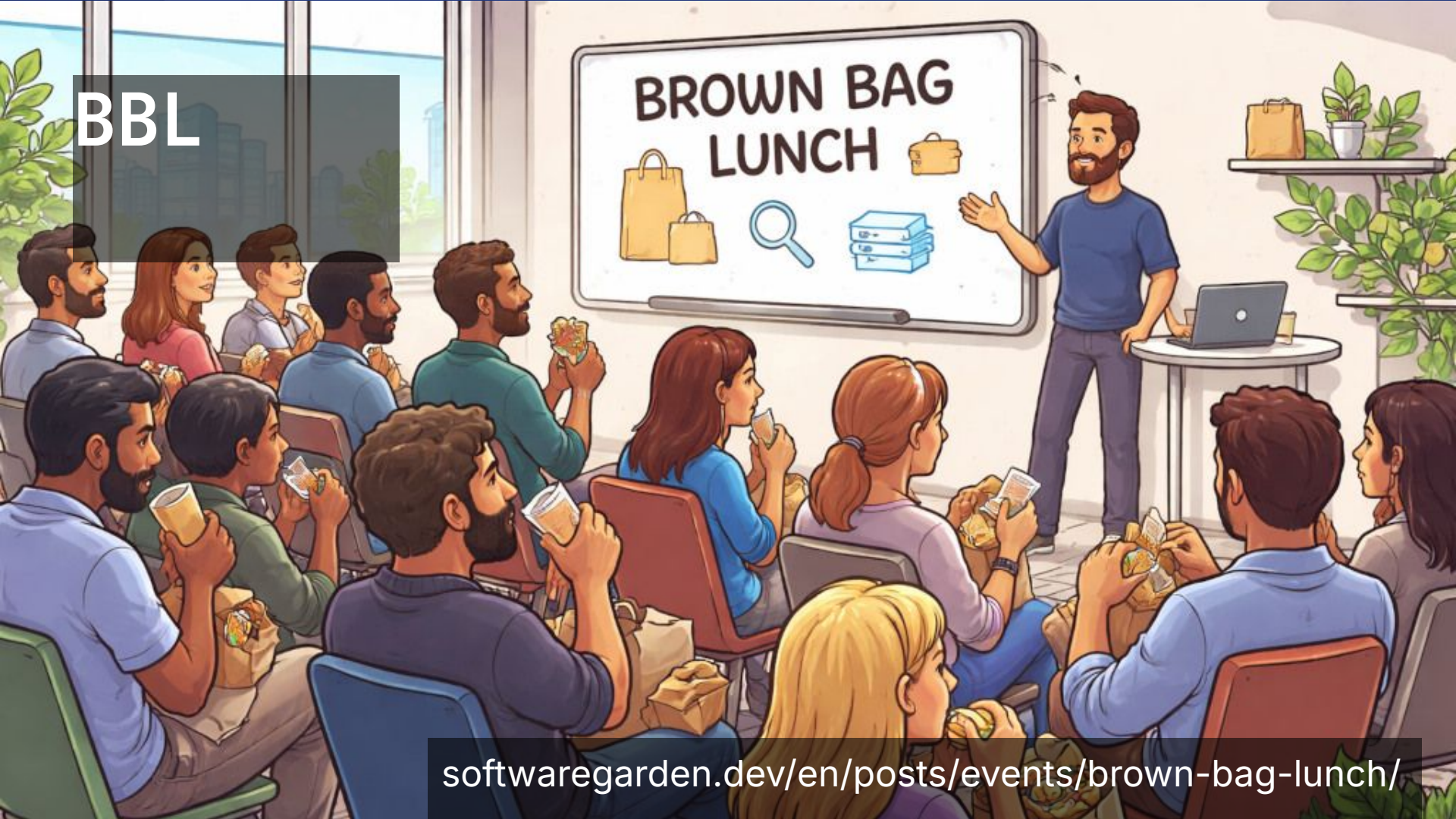


Join our community!
We're here for you

elastic.co/community

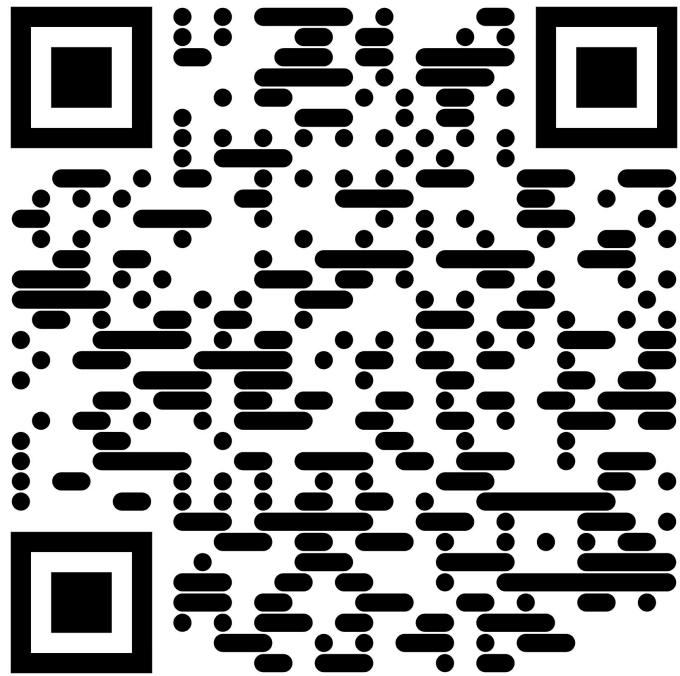


BBL

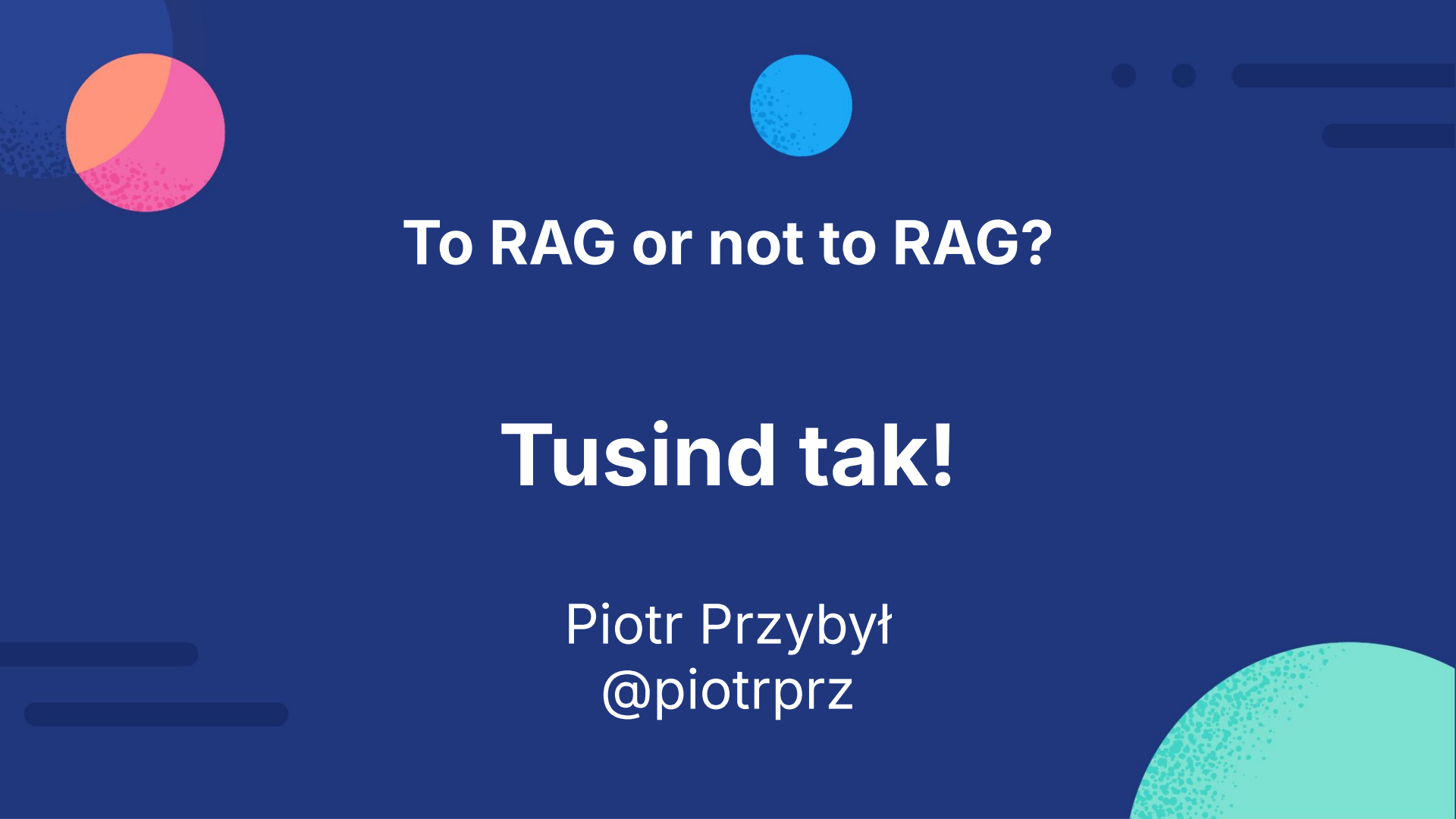


softwaregarden.dev/en/posts/events/brown-bag-lunch/





bit.ly/TRNTR-GOTO



To RAG or not to RAG?

Tusind tak!

Piotr Przybył
@piotrprz