Welcome! Before we get started ...

Tell us a little about yourself!

Short survey:

https://www.menti.com/al31ja5gtijw

weaviate

How to build scalable AI applications with a vector database

JP Hwang
Developer Educator

Agenda

- **Recap: Vector DBs & RAG**
- **Challenges of scalability**
  - **Work with a local cluster**
  - **Apply practical solutions**
- **Challenges of reliability**

Let’s get started

While we wait, go to:

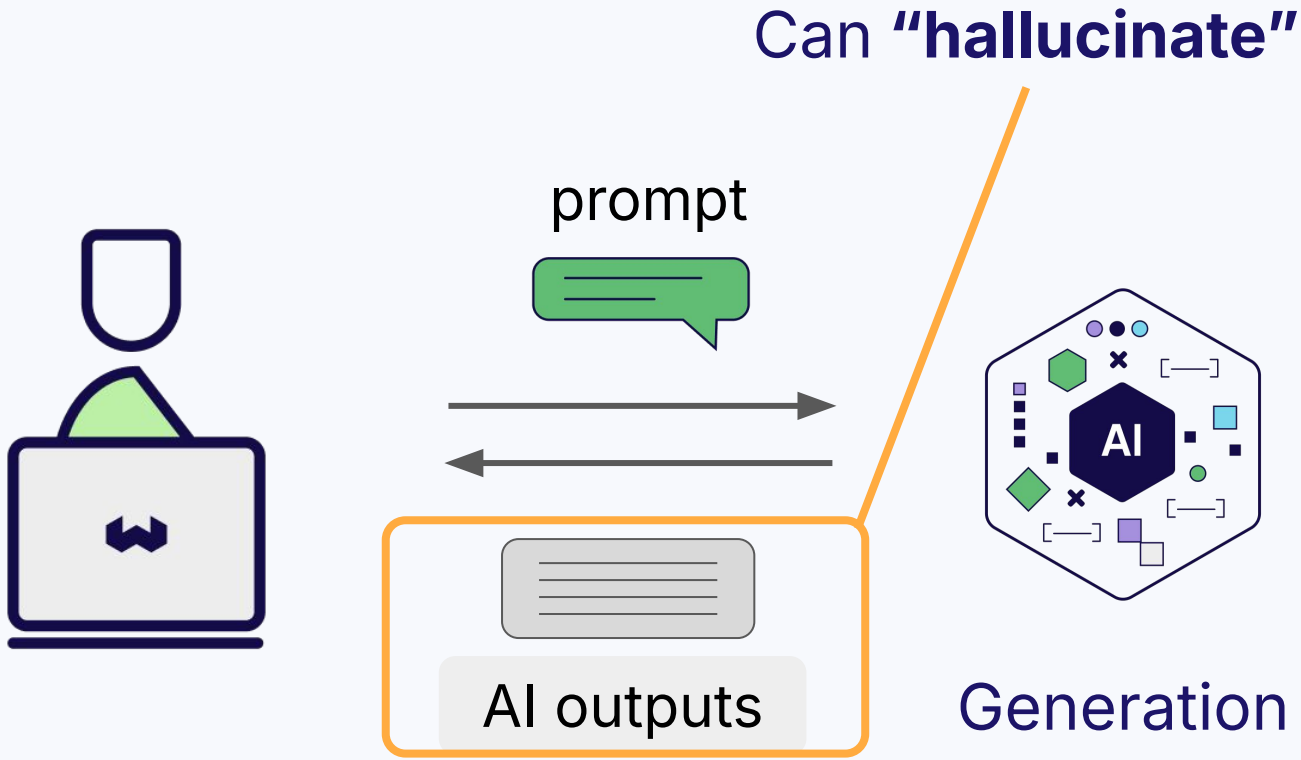
- **https://github.com/weaviate-tutorials/scalable-ai-workshop**

Start on the README instructions

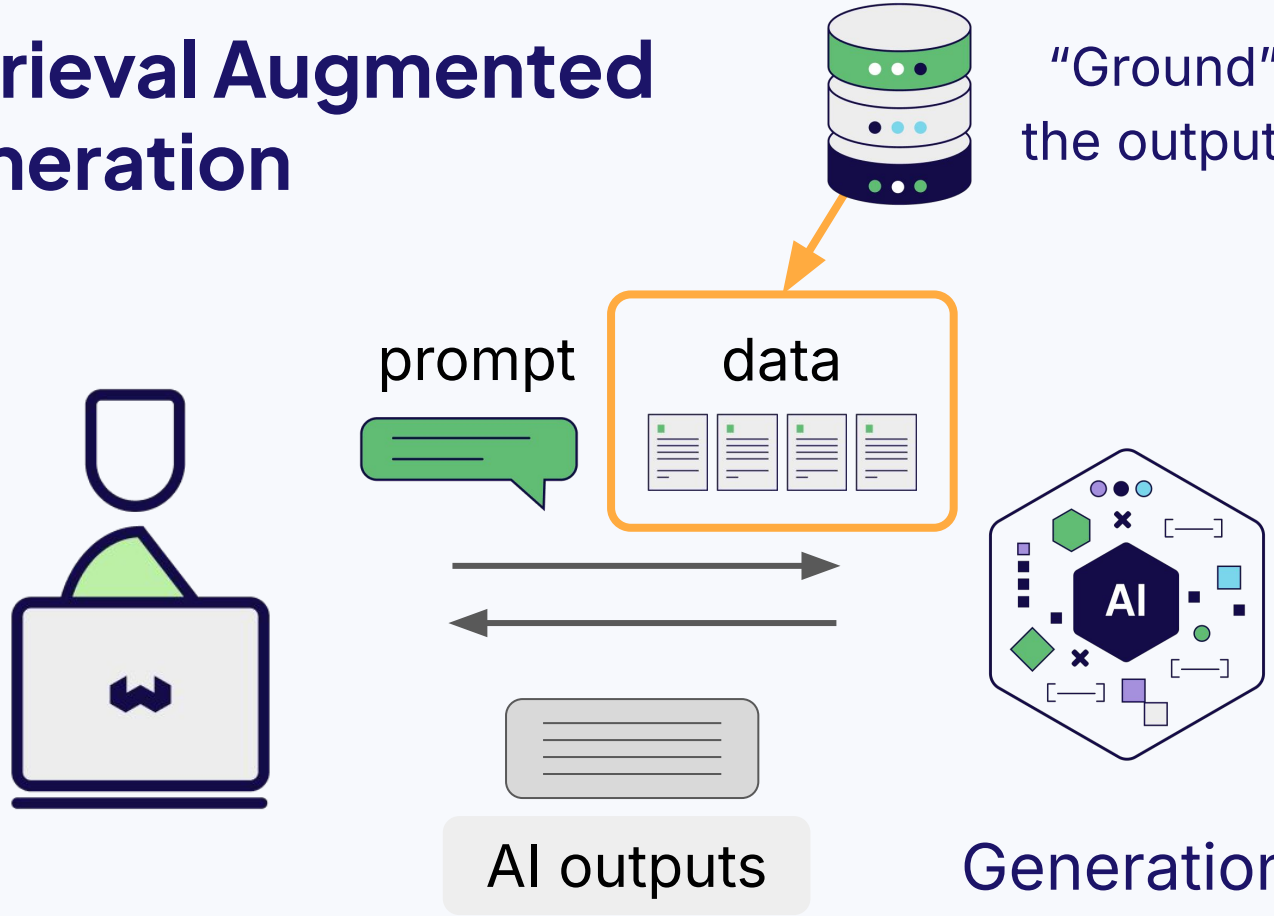
- **Step 1**
- **Step 2**

Recap: Vector DBs & RAG

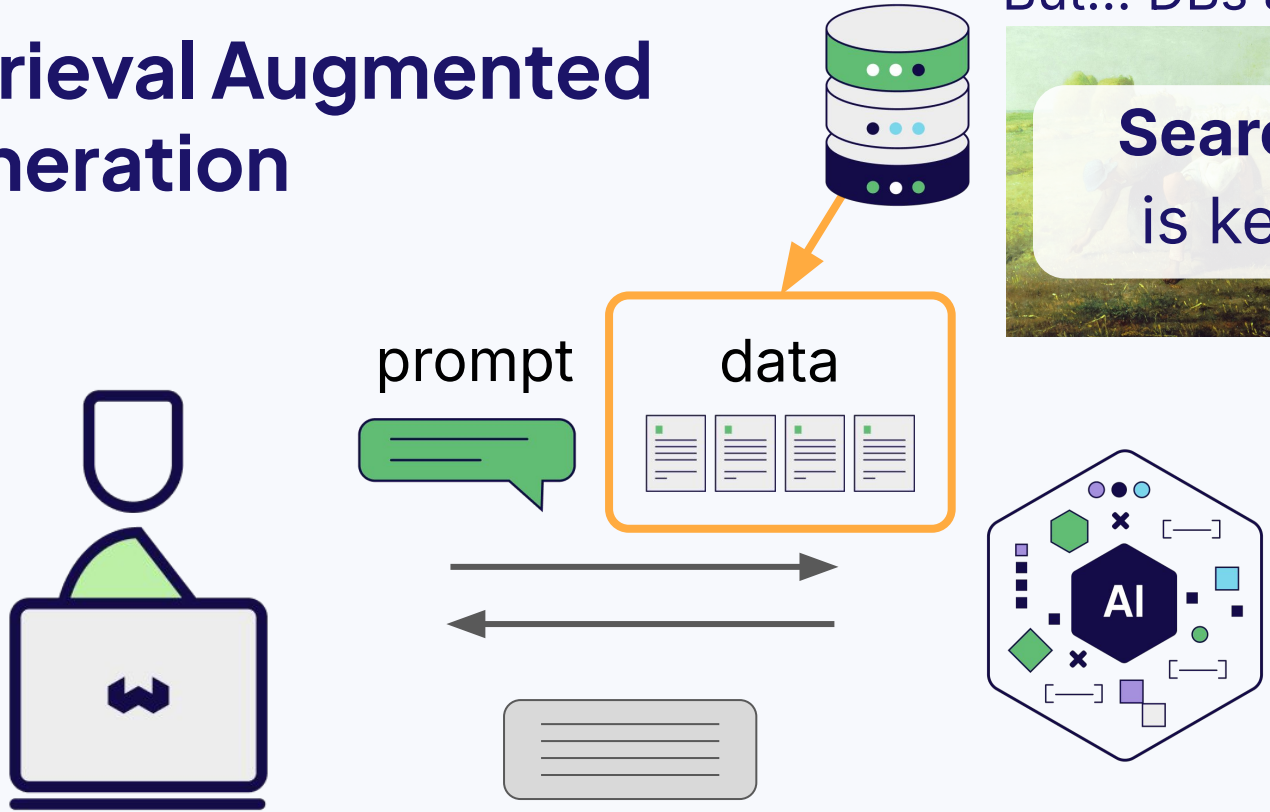
Large Language Model



Retrieval Augmented Generation



Retrieval Augmented Generation



Challenges of scale

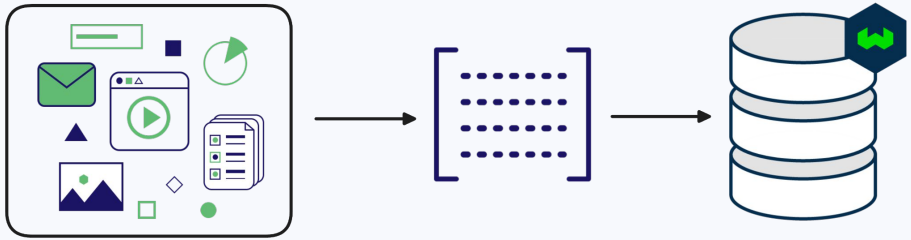
Scale: Considerations

- **Object count**: Memory & storage
- **User count**: Data management & compliance
- **Server load**: Distribute load

Managing resource requirements

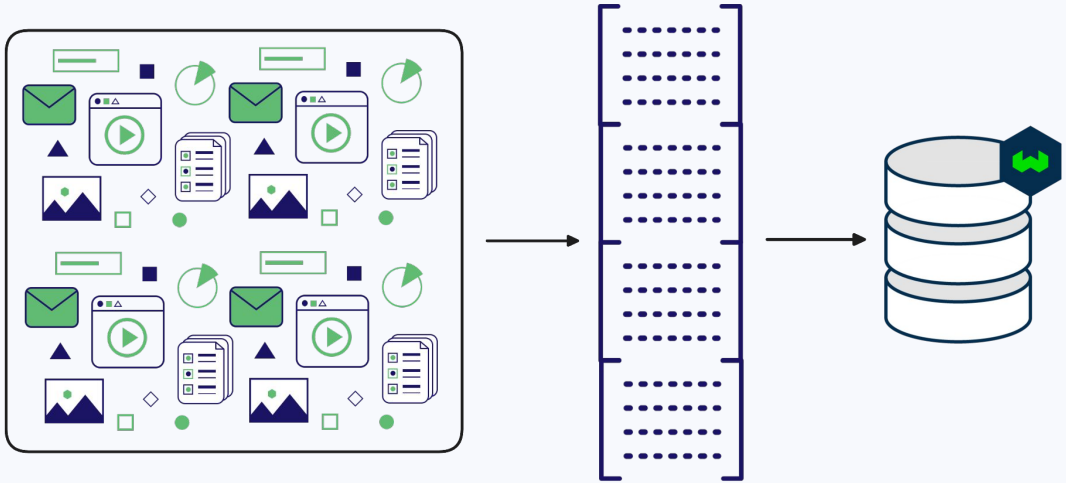
Scale: Solutions

Growth



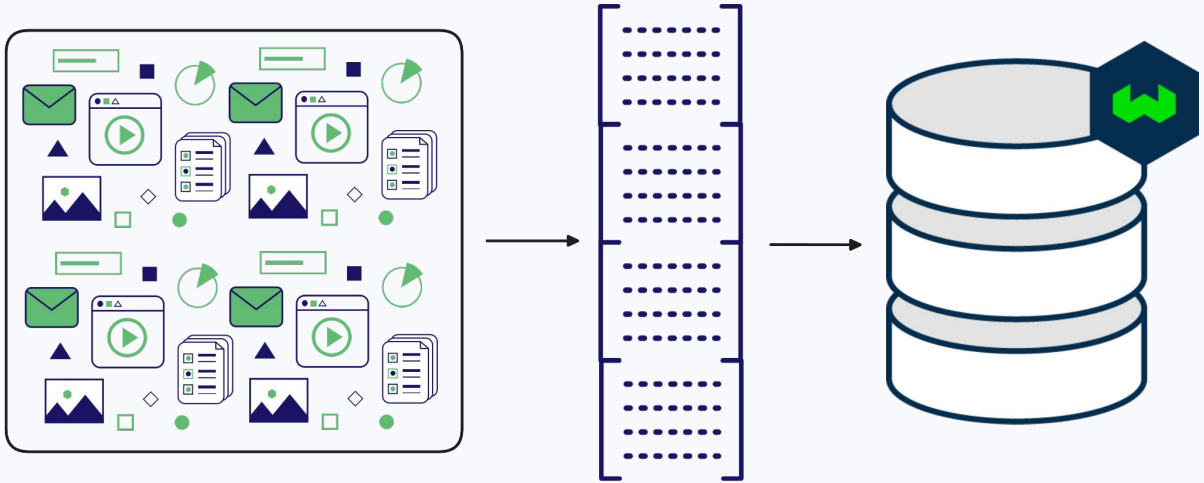
Scale: Solutions

Growth



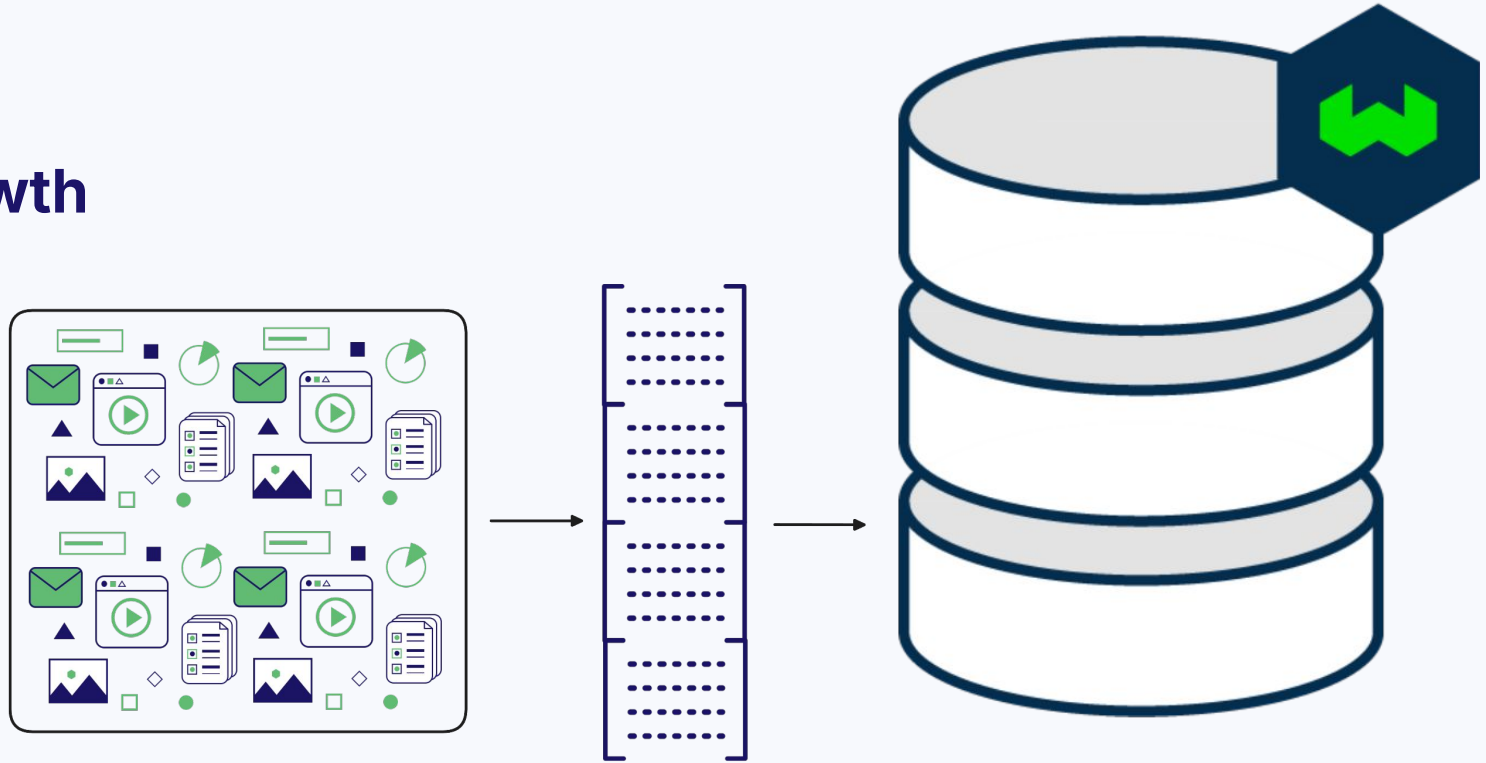
Scale: Solutions

Growth



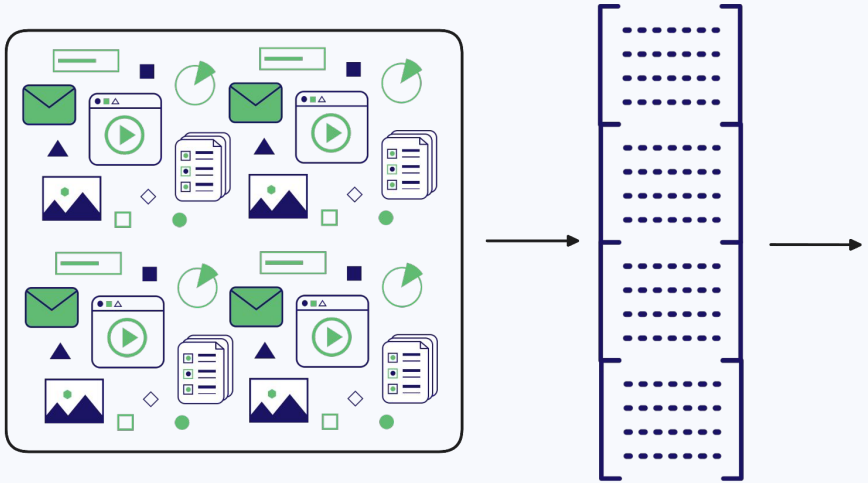
Scale: Solutions

Growth



Scale: Solutions

Growth



- Single point of failure
- Efficiency
- Upgrades
- Costs

1024 gib

19 matches

Instance name	On-Demand hourly rate	vCPU	Memory	Storage
x2gd.metal	$5.344	64	1024 GiB	2 x 1900 SSD
x2gd.16xlarge	$5.344	64	1024 GiB	2 x 1900 SSD
hpc6id.32xlarge	$5.70	64	1024 GiB	4 x 3800 NVMe SSD
x2iedn.8xlarge	$6.669	32	1024 GiB	1 x 950 NVMe SSD
x2idn.16xlarge	$6.669	64	1024 GiB	1 x 1900 NVMe SSD

AWS Spot pricing, Nov 2024

1024 gib

19 matches

Instance name	On-Demand hourly rate	orage
x2gd.metal	$5.344	1900 SSD
x2gd.16xlarge		
hpc6id.32xlarge		
x2iedn.8xlarge		
x2idn.16xlarge		

120 * 365 = **$43,800 / year!**

AWS Spot pricing, Nov 2024

Vector indexing options

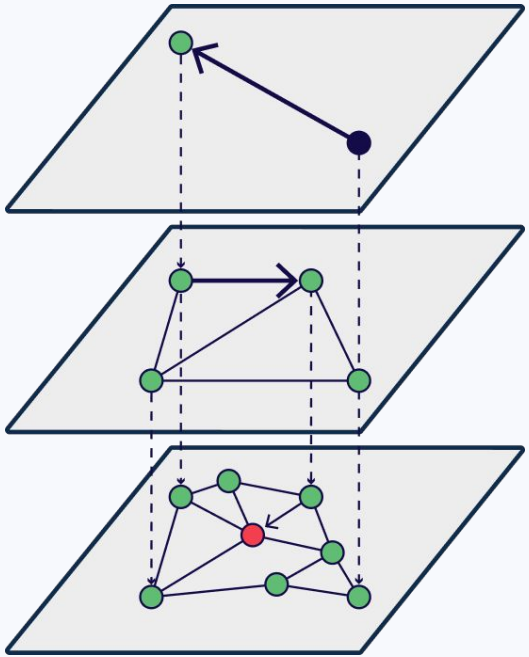
Scale: Solutions

Improve efficiency - indexing

Scale: Solutions

Improve efficiency - indexing

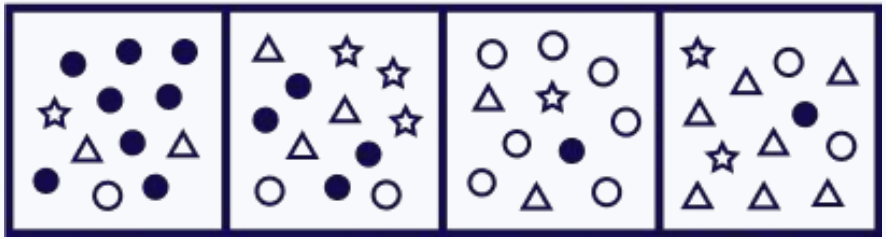
- **HNSW** index (default)



Scale: Solutions

Improve efficiency - indexing

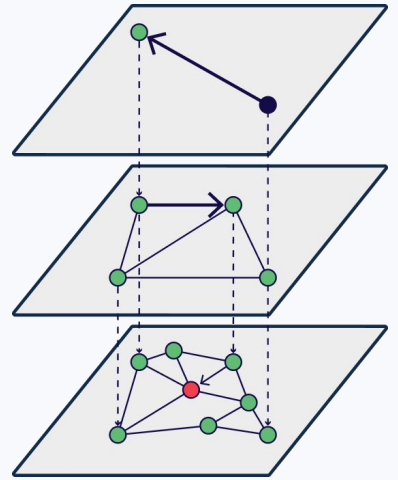
- **HNSW** index (default)
- **Flat** index

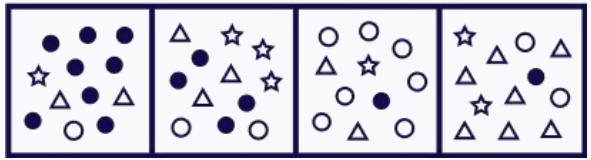


Scale: Solutions

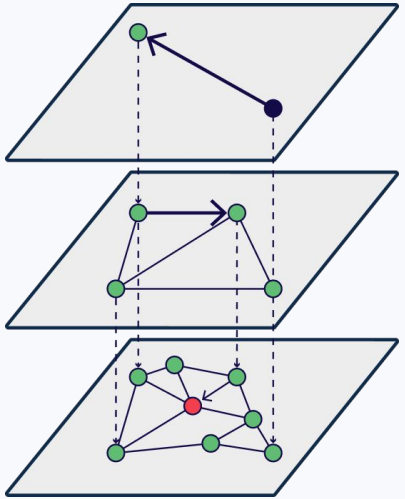
Indexes - comparison

- **HNSW**: fast + scalable
- **Flat**: tiny footprint; ~100k objs





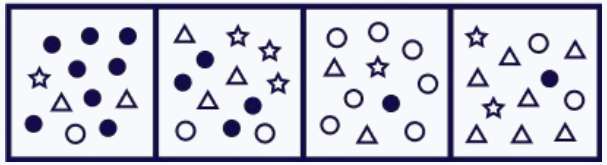
Scale: Solutions



How to choose?

- **Start** with **HNSW**

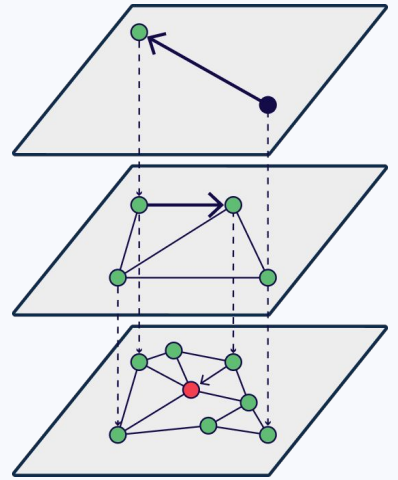
  (Tune speed / size / accuracy)
- Multi-tenancy?
  - Try **dynamic**

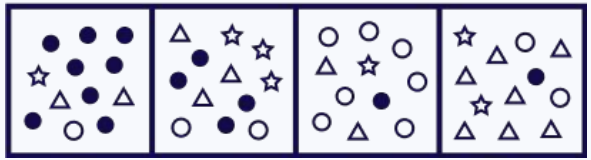


Scale: Solutions

Improve efficiency - indexing

- **HNSW** index (default)
- **Flat** index
- **Dynamic** index
  - Flat → HNSW @ threshold

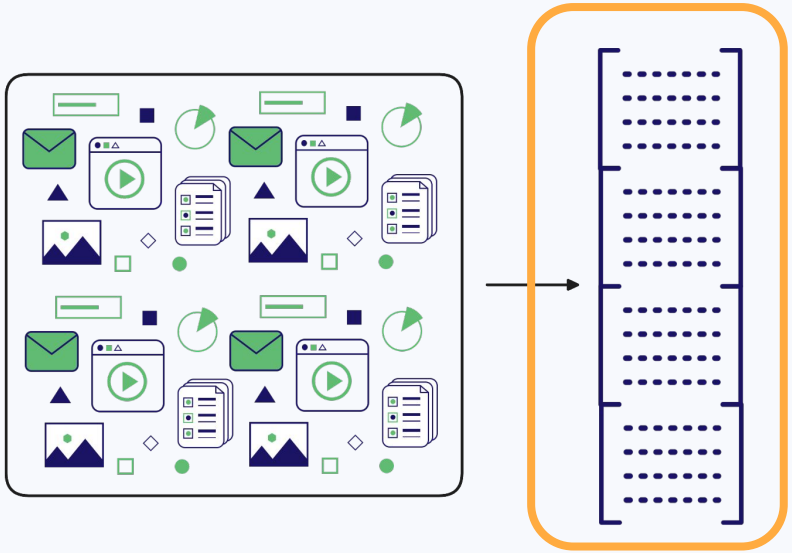




Vector quantization

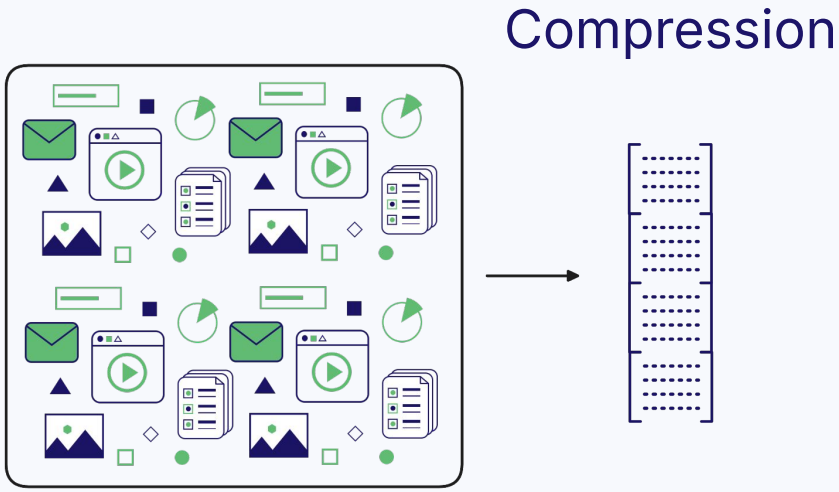
Scale: Solutions

Improve efficiency



Scale: Solutions

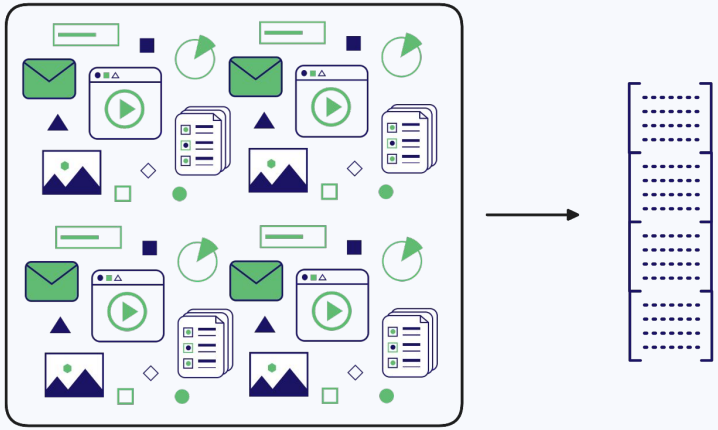
Improve efficiency



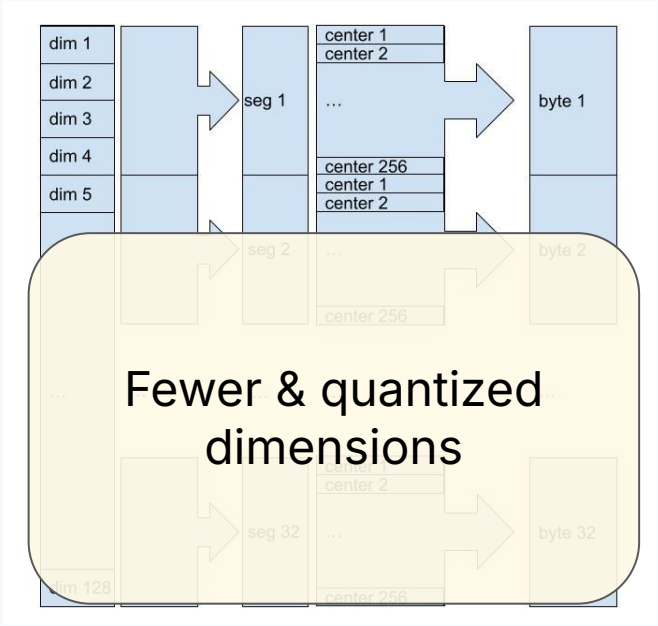
Scale: Solutions

Improve efficiency

Compression

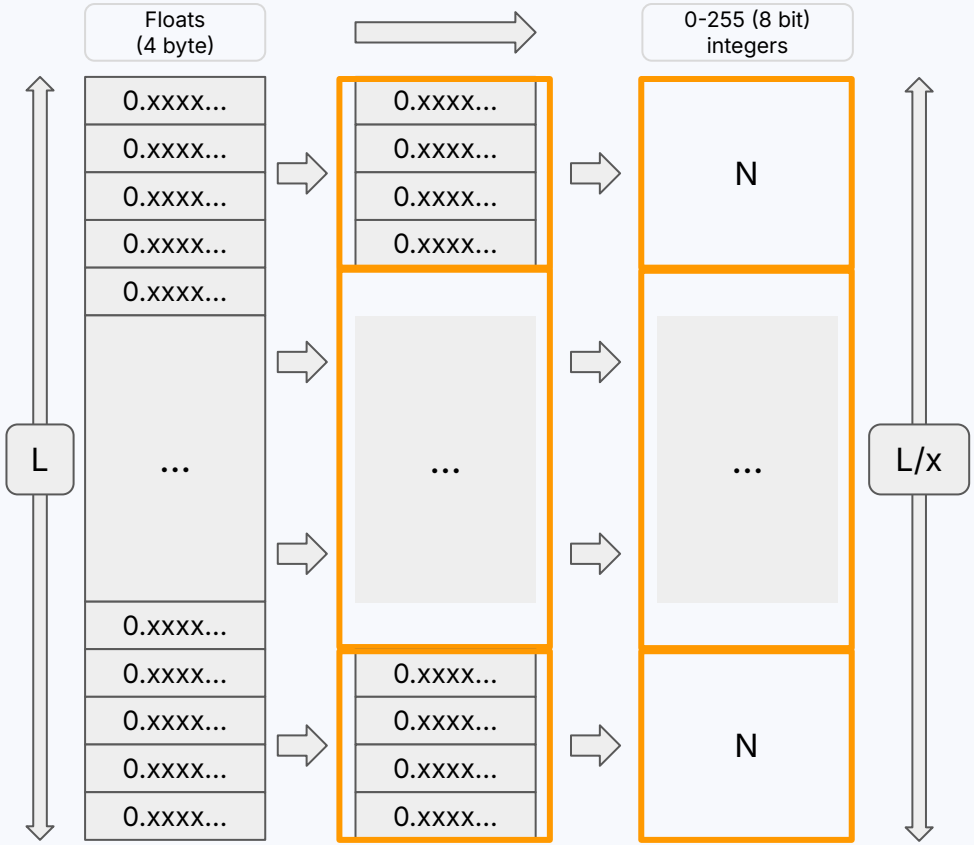


Product quantization



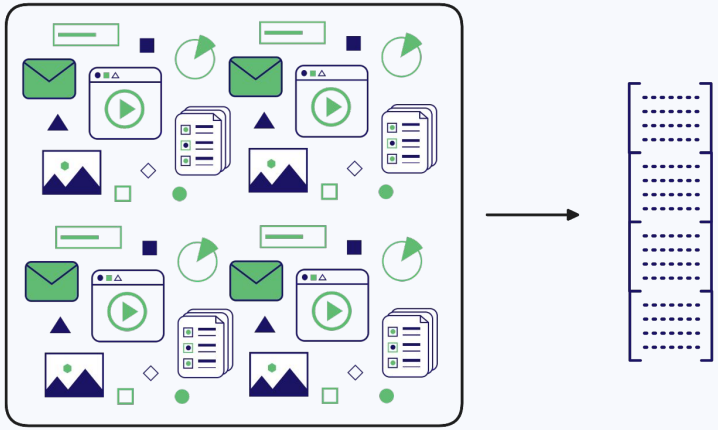
Customisable compression

(e.g. 128 floats → 32 bytes: 16x)


Floats
(4 byte)
0-255 (8 bit)
integers
L
L/x
0.xxxx...
0.xxxx...
0.xxxx...
0.xxxx...
0.xxxx...
...
0.xxxx...
0.xxxx...
0.xxxx...
0.xxxx...
0.xxxx...
0.xxxx...
0.xxxx...
0.xxxx...
0.xxxx...
0.xxxx...
N
...
N

Scale: Solutions

Improve efficiency



Compression

Binary quantization

[-0.1324..., -0.9253...,
0.2389...,
...
0.3249..., 0.2390..., -0.4823...]

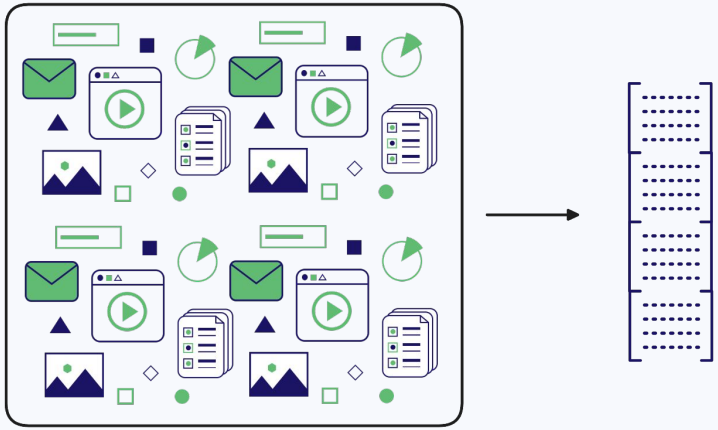
↓

[0, 0, 1, 0, 1, 1,
...
0, 1, 0, 1, 1, 0]

n floats → n bits
(32x reduction)

Scale: Solutions

Improve efficiency



Scalar quantization

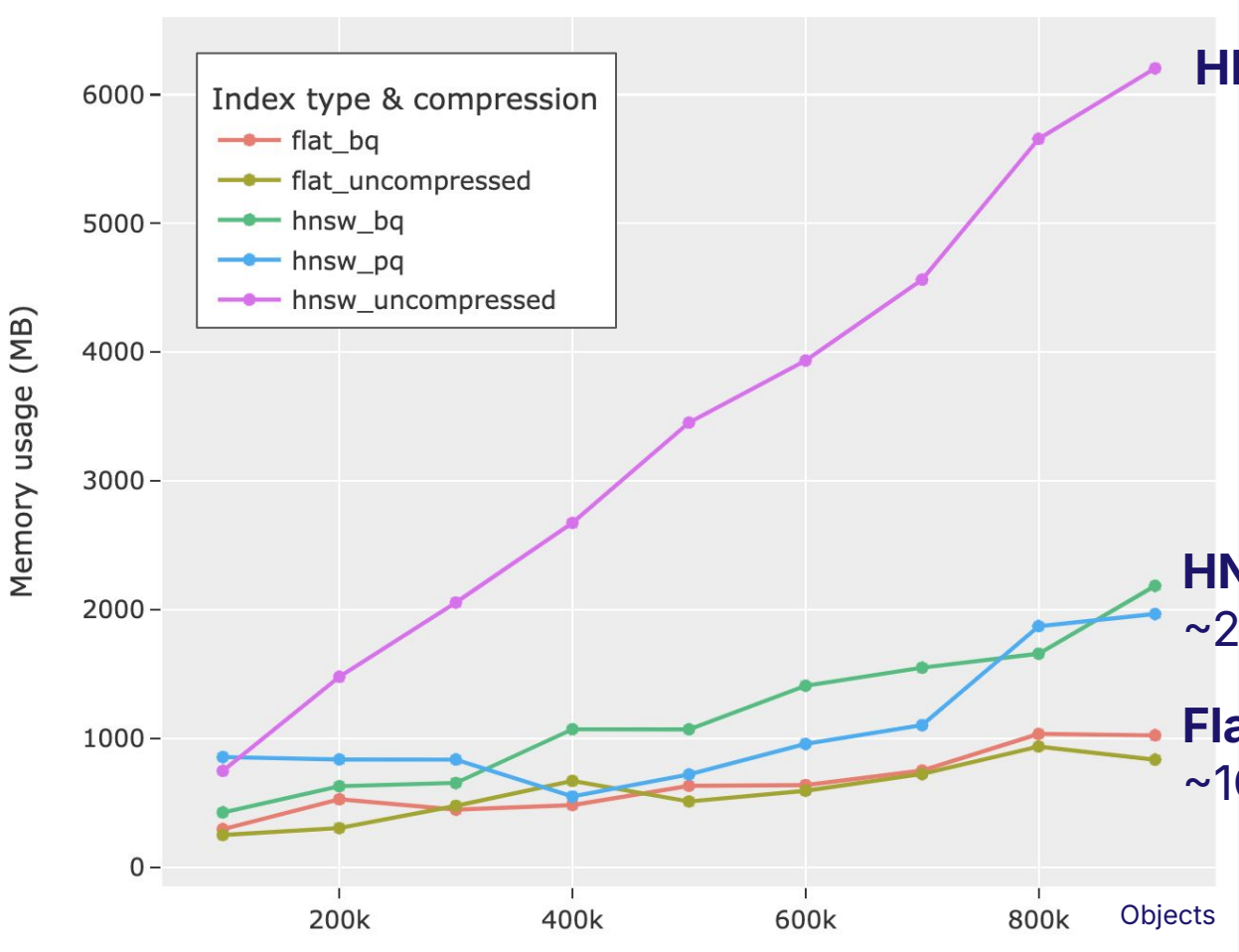
[-0.1324..., -0.9253...,
0.2389...,
...
0.3249..., 0.2390..., -0.4823...]

↓

[104, 12, 138,
...
152, 138 47]

n floats → n ints
(4x reduction)

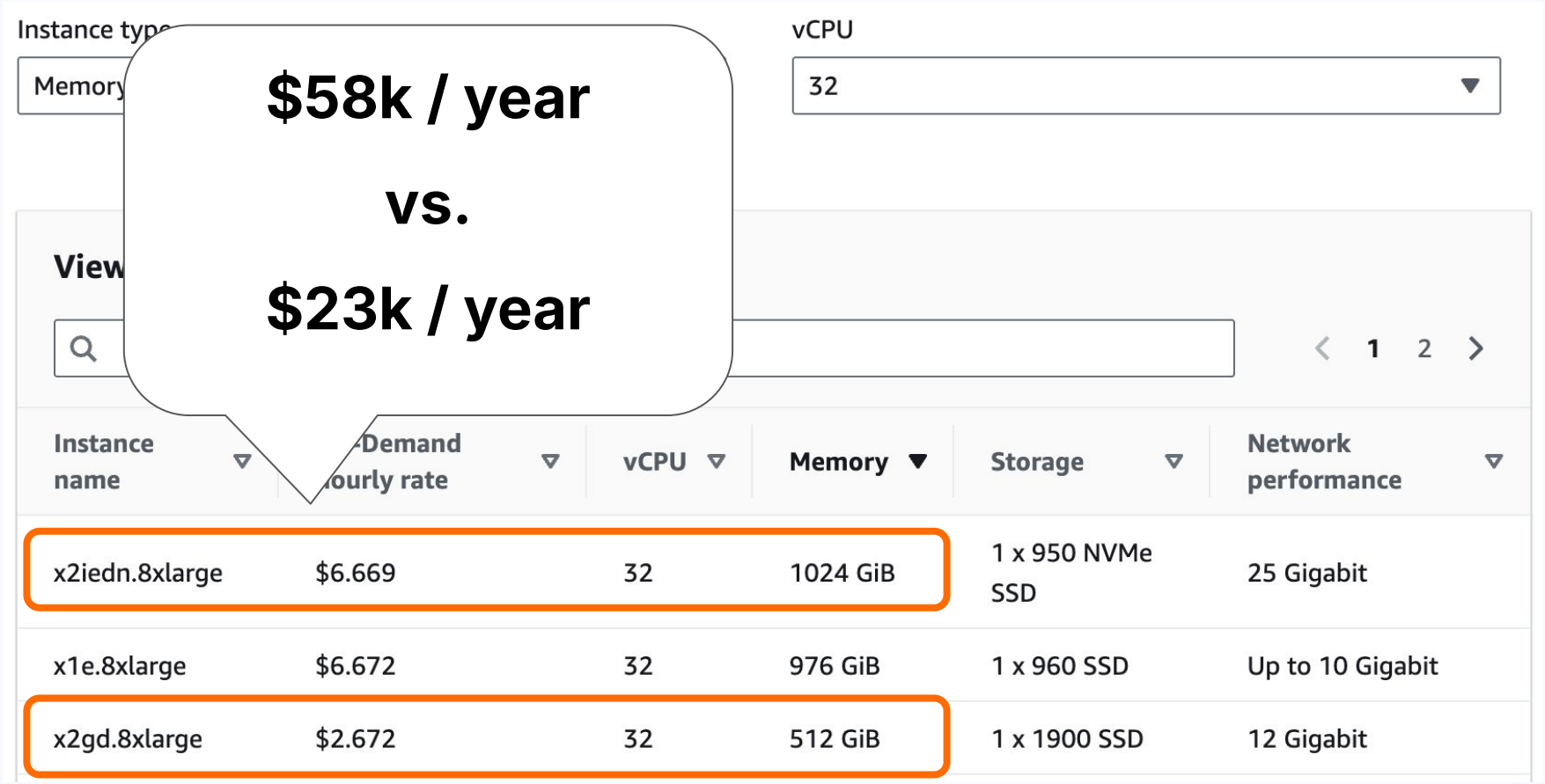
Example: Index type & quantization vs memory footprint



HNSW: ~6GB

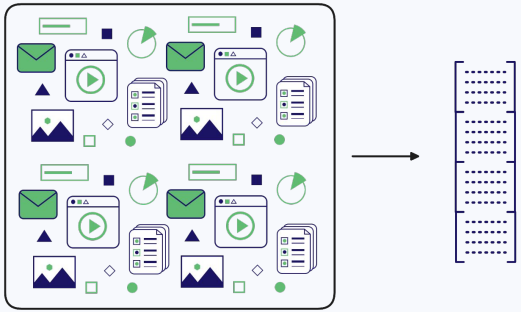
HNSW + BQ/PQ: ~2GB

Flat (+BQ): ~1GB



AWS Spot pricing, Nov 2024

Scale: Solutions



BQ / PQ / SQ compression

Search **quality** mitigated by over-fetching & rescoring

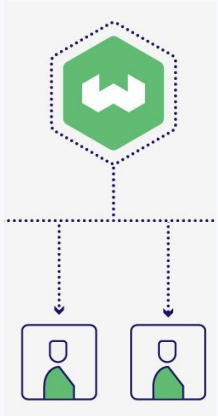
When to use which?

- Generally, try PQ first
- BQ: model-specific

Multi-tenancy

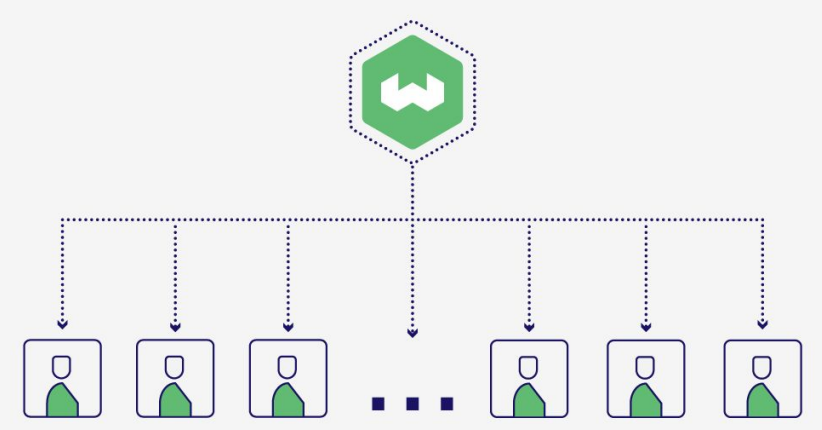
Scale: Solutions

End user growth



Scale: Solutions

End user growth



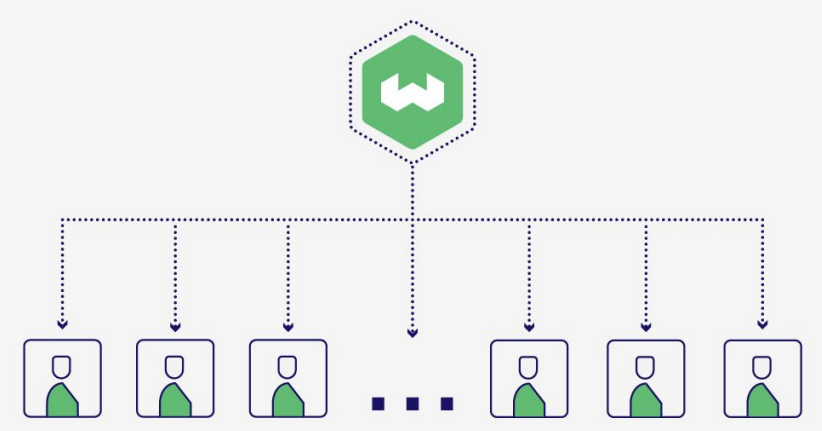
Challenges faced:

- Performance
- Data isolation
- Compliance

Developed: **Multi-tenancy**

Scale: Solutions

End user growth

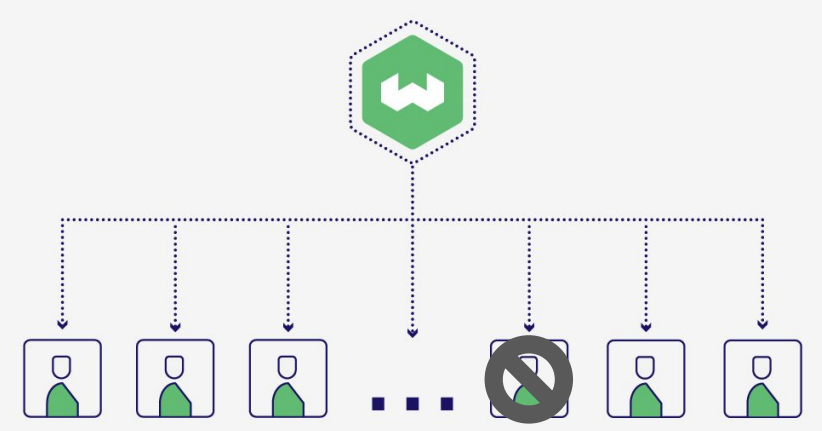


Multi-tenancy implementation

- 1000s per node
- Isolated
- Active/inactive/offloaded tenants

Scale: Solutions

End user growth

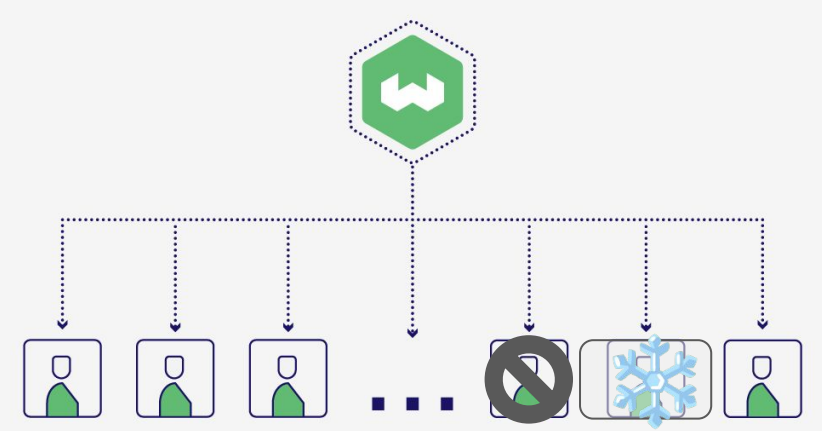


Multi-tenancy implementation

- 1000s per node
- **Isolated** (easy deletion & compliance)
- Active/inactive/offloaded tenants

Scale: Solutions

End user growth



Multi-tenancy implementation

- 1000s per node
- Isolated
- **Active/inactive/offloaded tenants** (efficient)

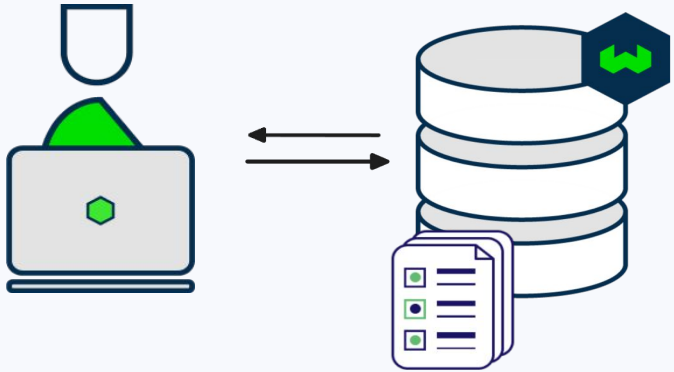
Replication

Reliability: Considerations

- **Robust to errors**: Ensure consistency
- **Downtime**: Reduce disruption
- **Backups**: In case of emergency

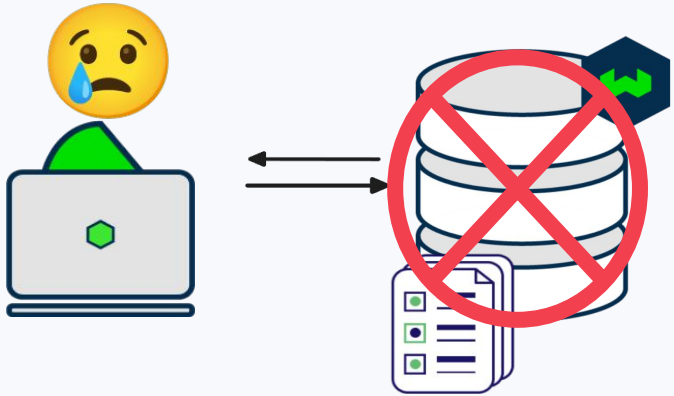
Reliability: Solutions

Happy days



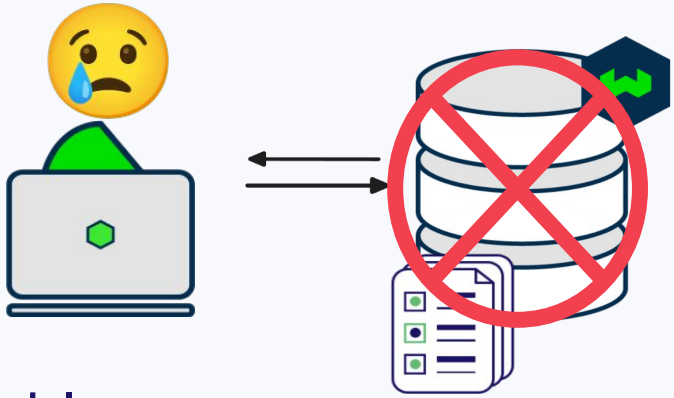
Reliability: Solutions

(Less) Happy days



Reliability: Solutions

(Less) Happy days

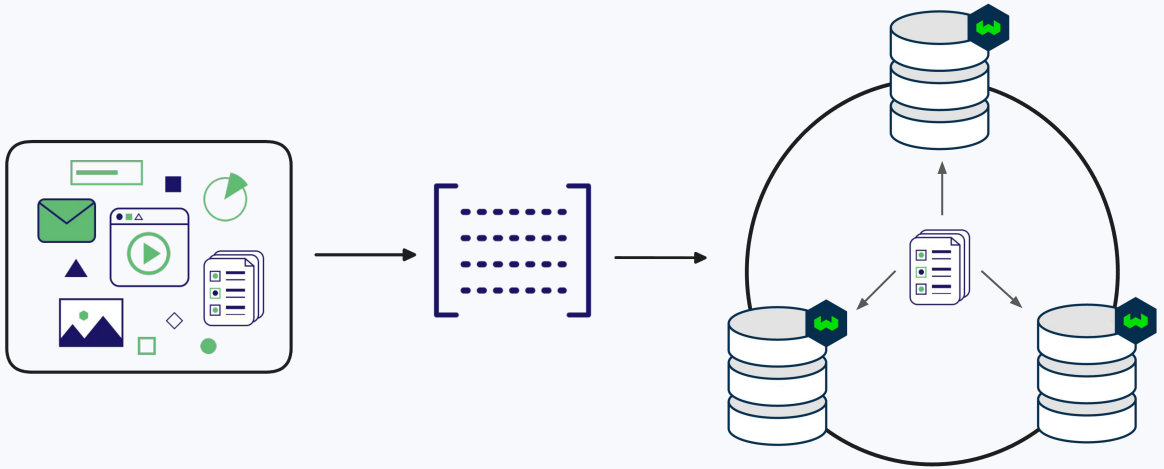


Downtime = inevitable

Reliability: Solutions

Provide redundancy

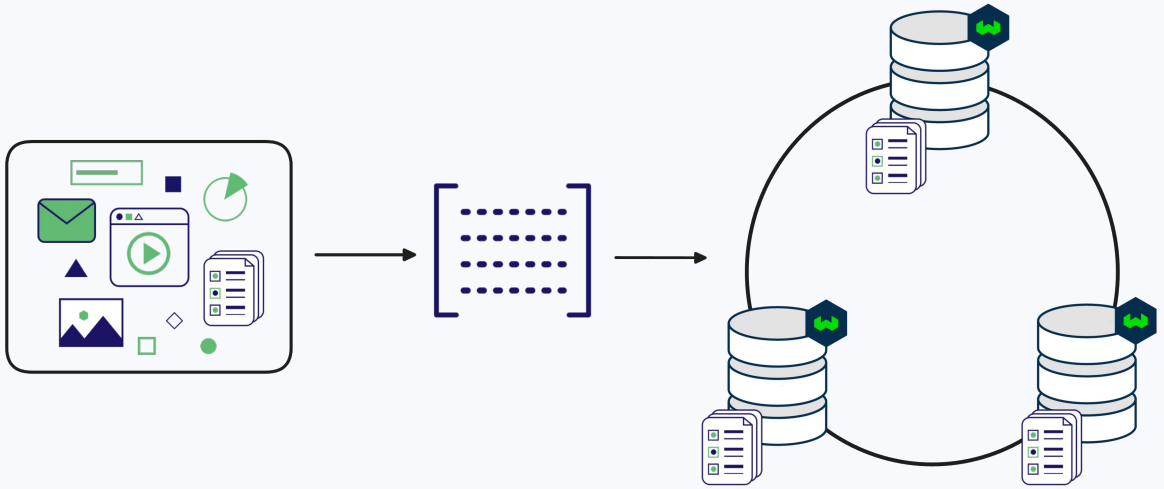
Replication



Reliability: Solutions

Provide redundancy

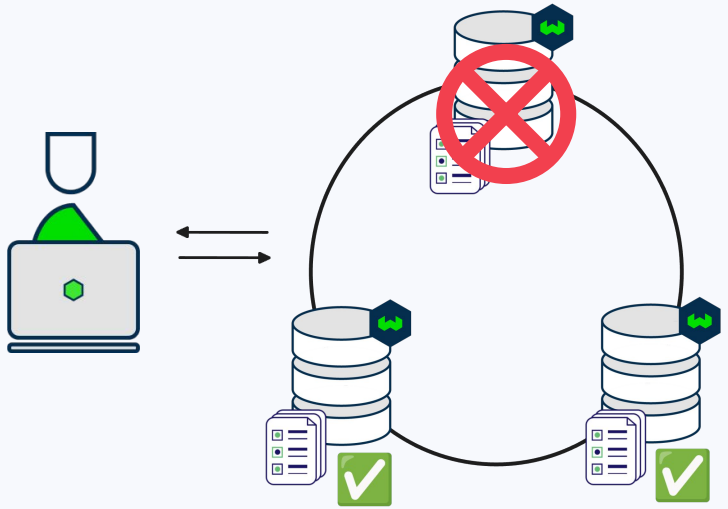
Replication



Reliability: Solutions

Provide redundancy

Replication



Reliability: Solutions

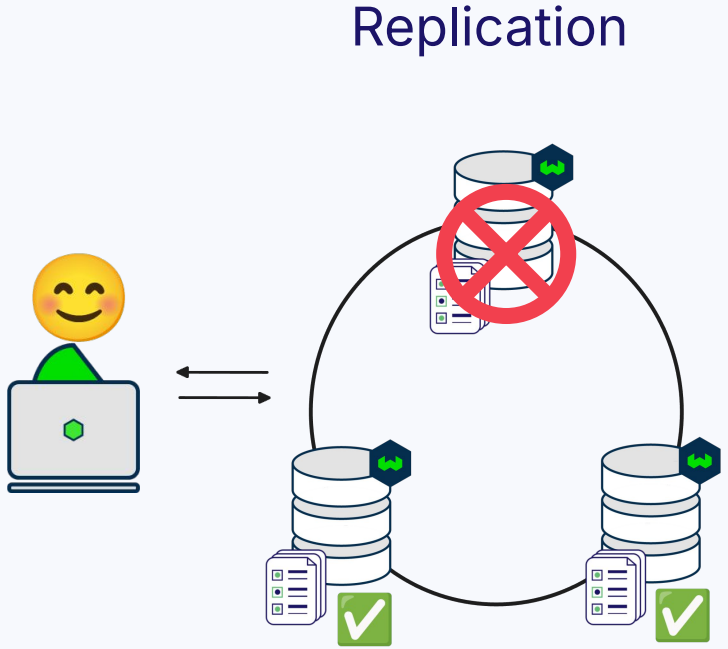
Provide redundancy

Node-level downtime:

- inevitable

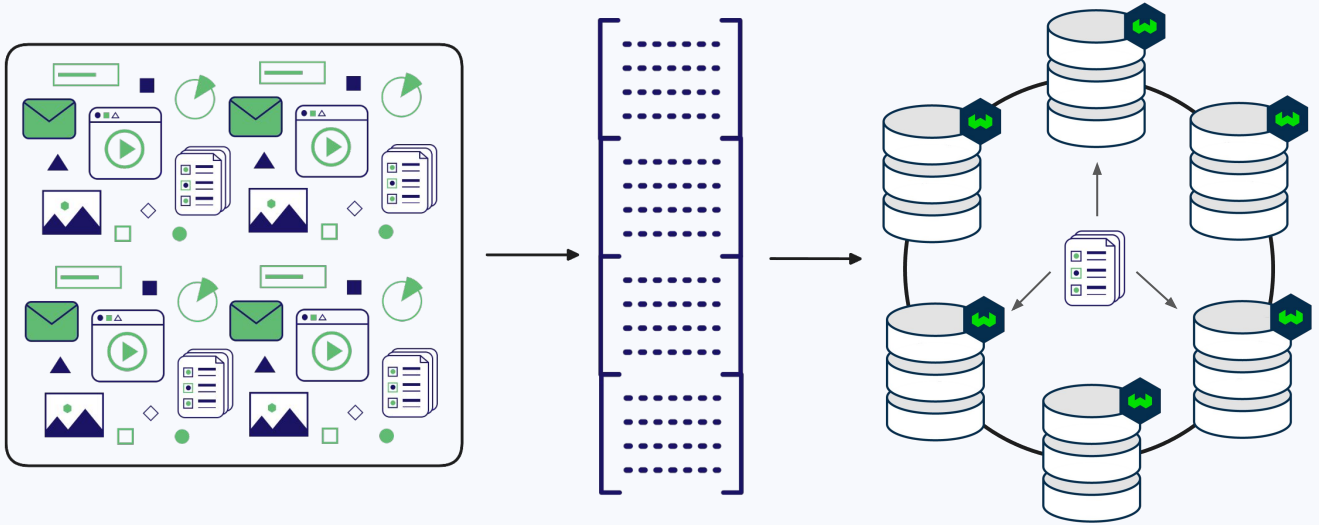
System-level downtime:

- avoidable!



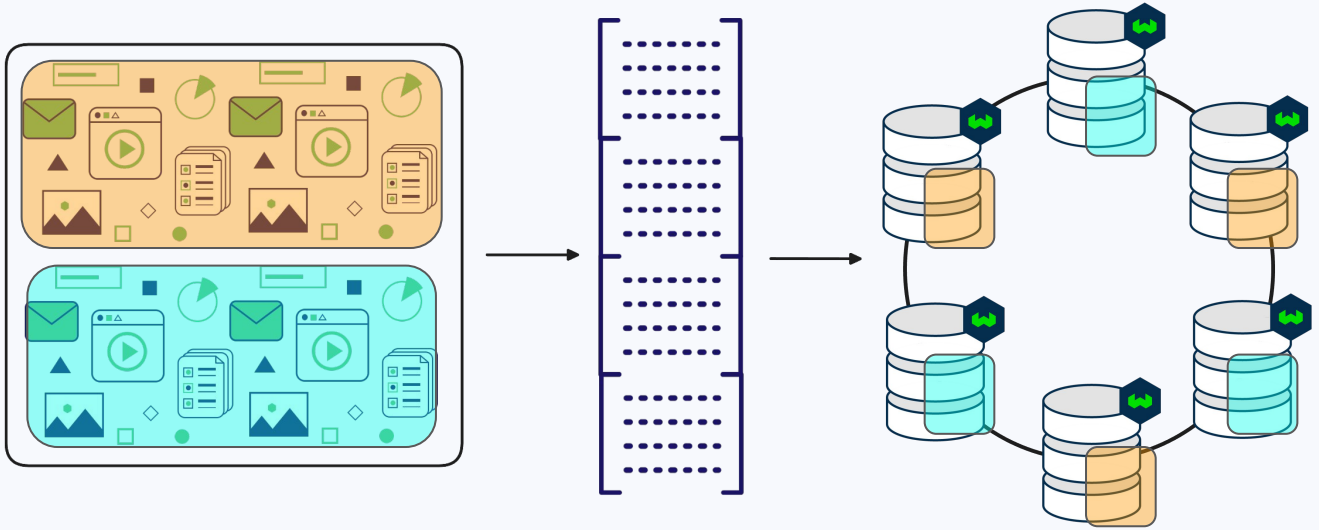
Reliability: Solutions

Sharding + replication



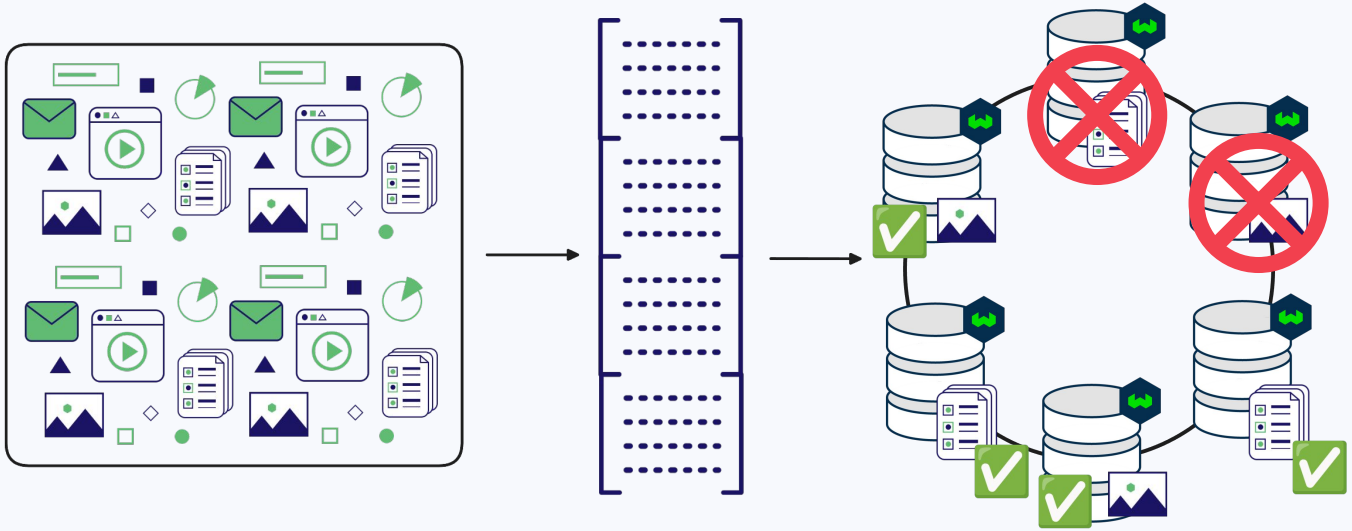
Reliability: Solutions

Sharding + replication



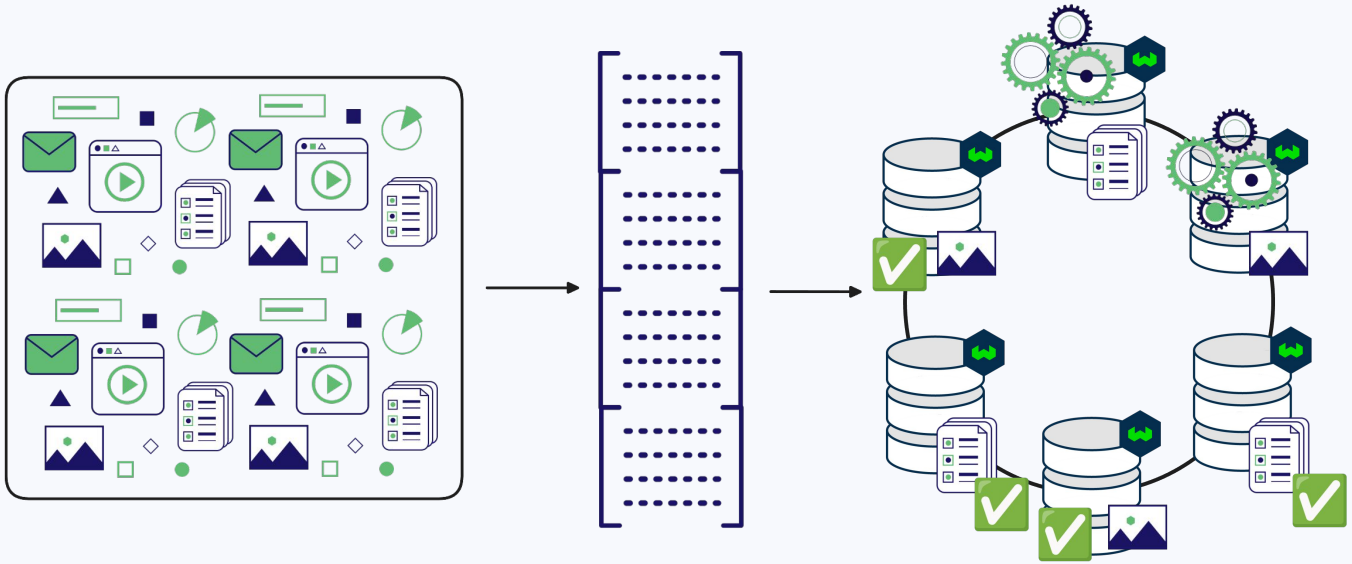
Upgrade: Solutions

Sharding + replication



Upgrade: Solutions

Sharding + replication



Upgrade: Solutions

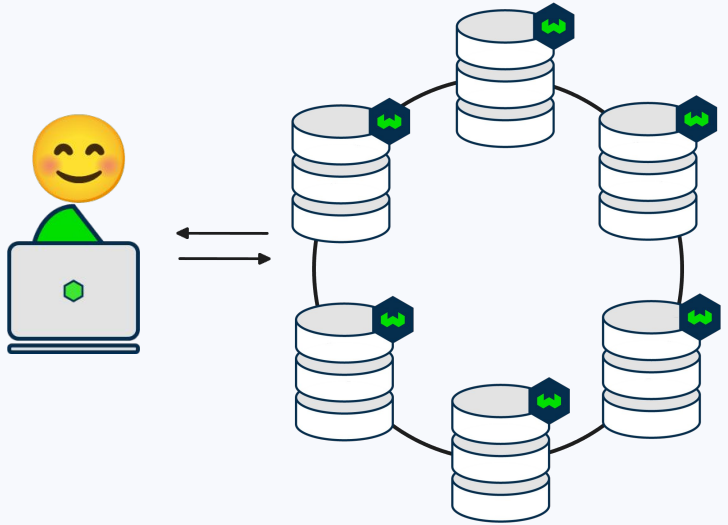
Provide redundancy

“Just works”

for the end user -

with the latest

versions

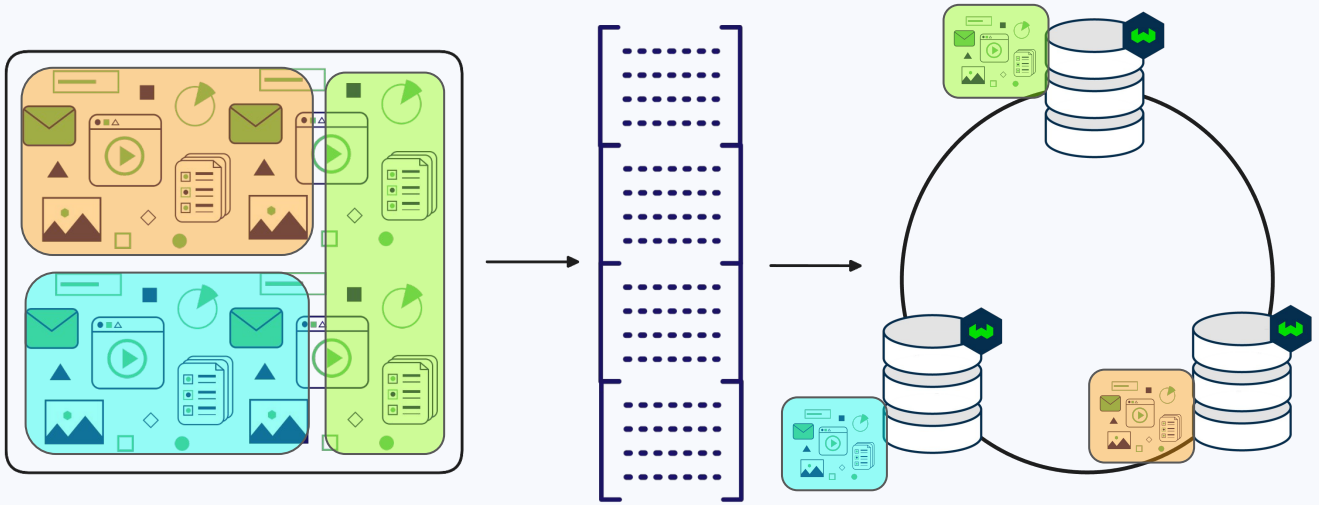


Scaling out

Scale: Solutions

Deal with growth

Horizontal scaling
(sharding)

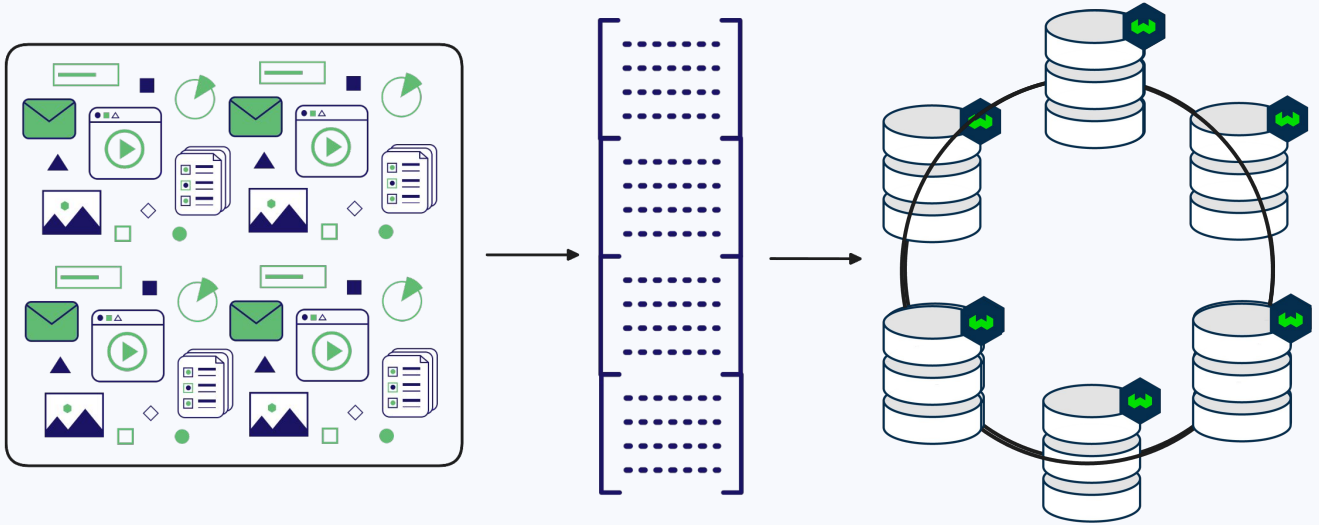


Scale: Solutions

Billion scale since 2022!

Deal with growth

To scale:
Add more nodes

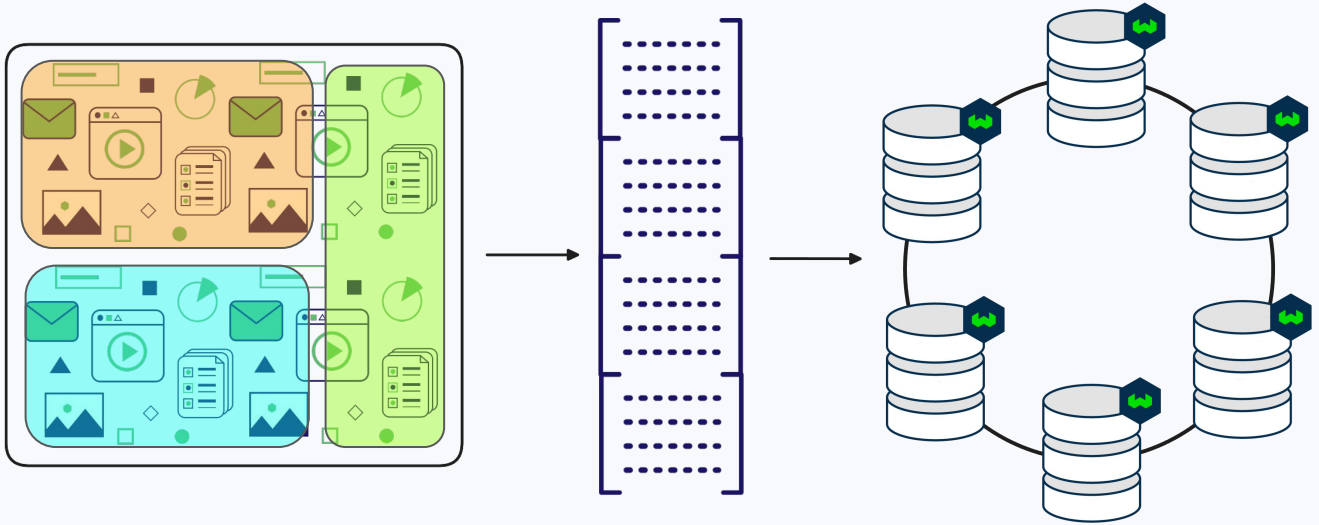


Scale: Solutions

Billion scale since 2022!

Deal with growth

To scale:
Add more nodes

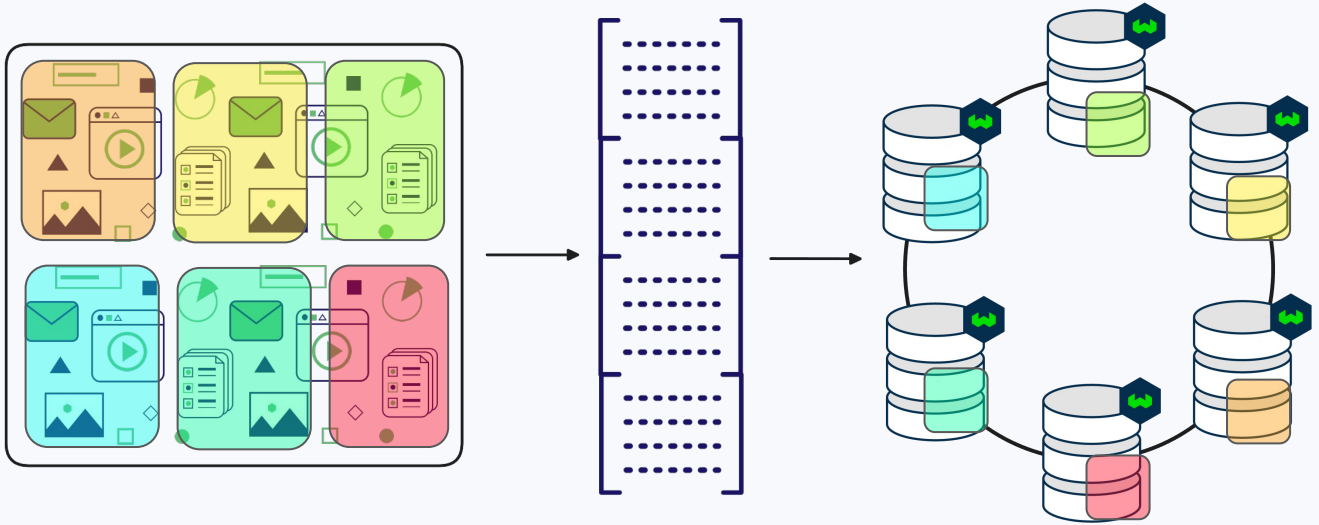


Scale: Solutions

Billion scale since 2022!

Deal with growth

To scale:
Add more nodes



Available Solutions

- Scaling up
- Scaling out
- Index options
- Quantization
- Multi-tenancy (+ tenant states)
- Replication

Thank you

Connect with us!

 weaviate.io

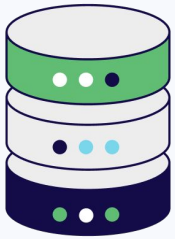
 weaviate/weaviate

 weaviate_io

BONUS SLIDES!

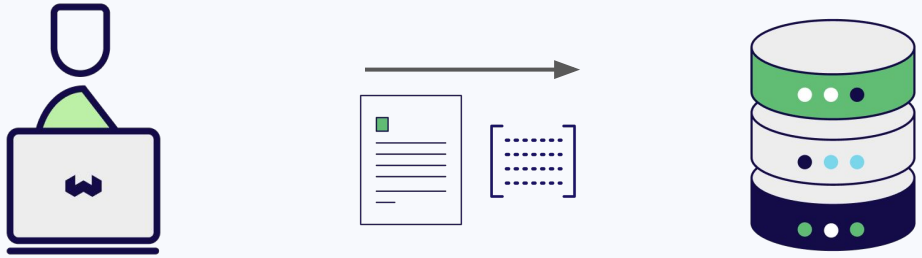
Developer workflow

Ingest data



Ingest data

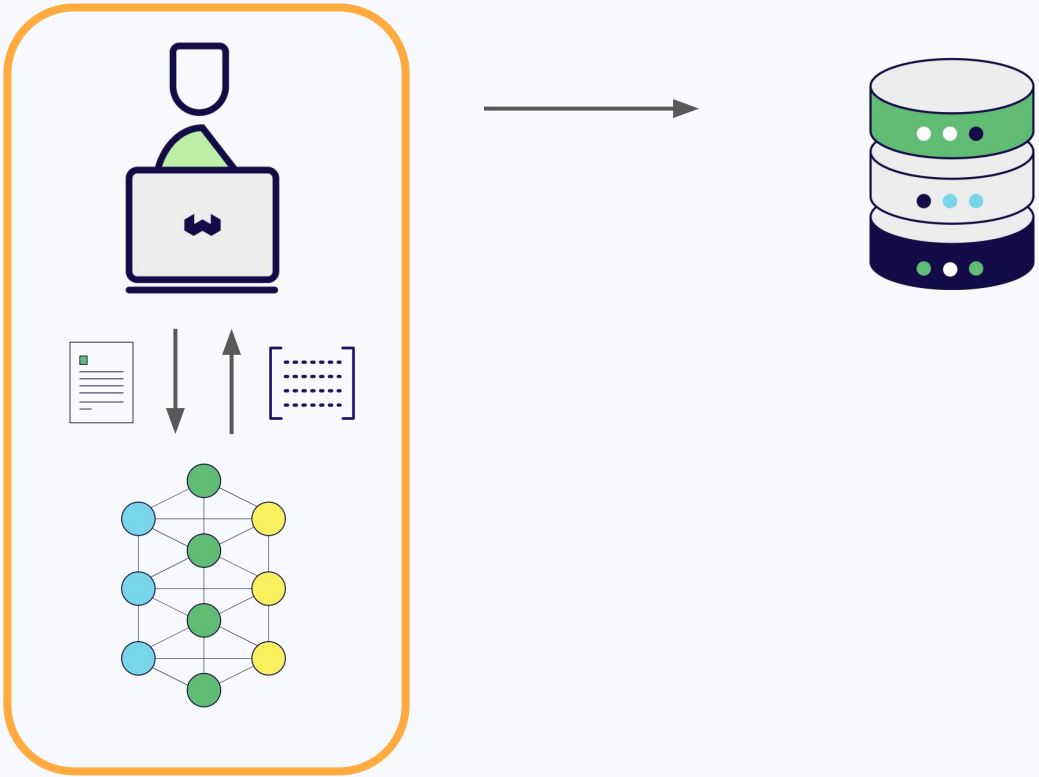
Ingest data



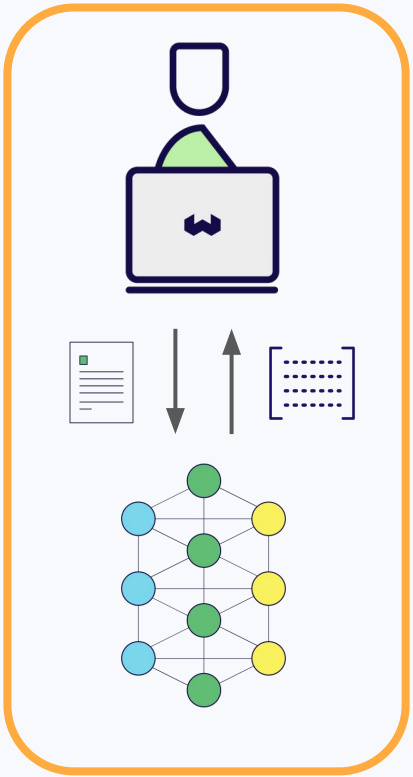
Ingest data

Where do they come from??

Obtain Embeddings

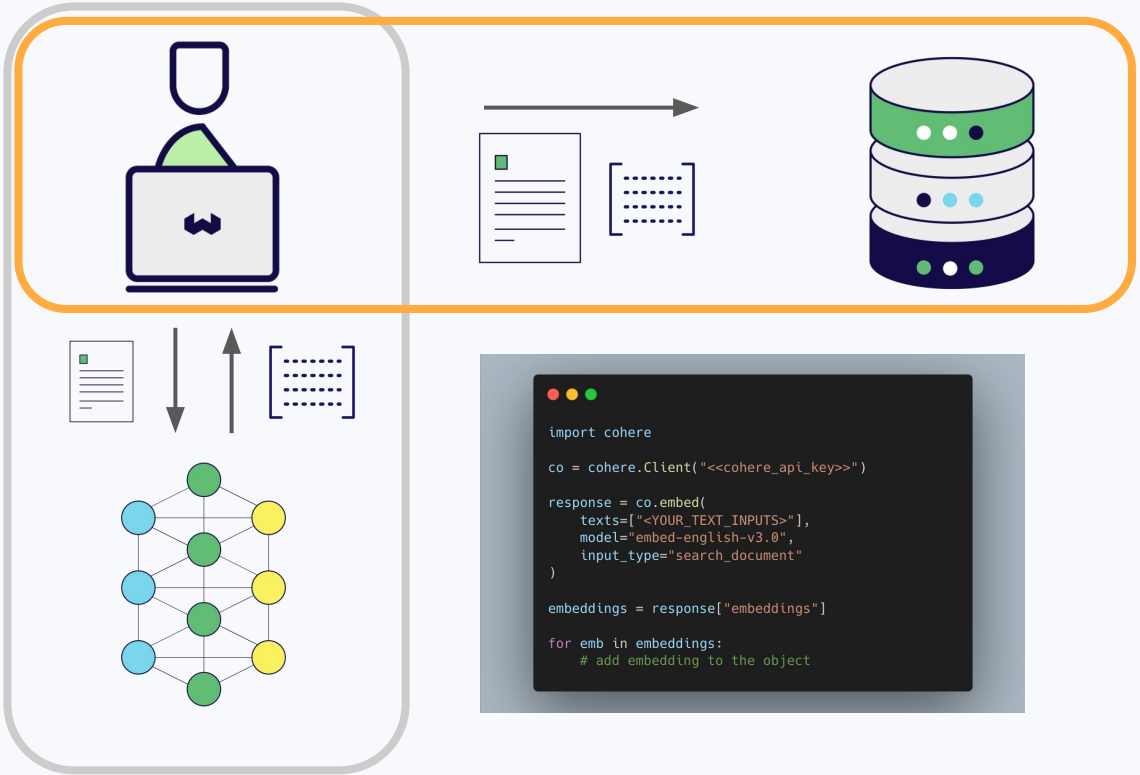


Obtain Embeddings



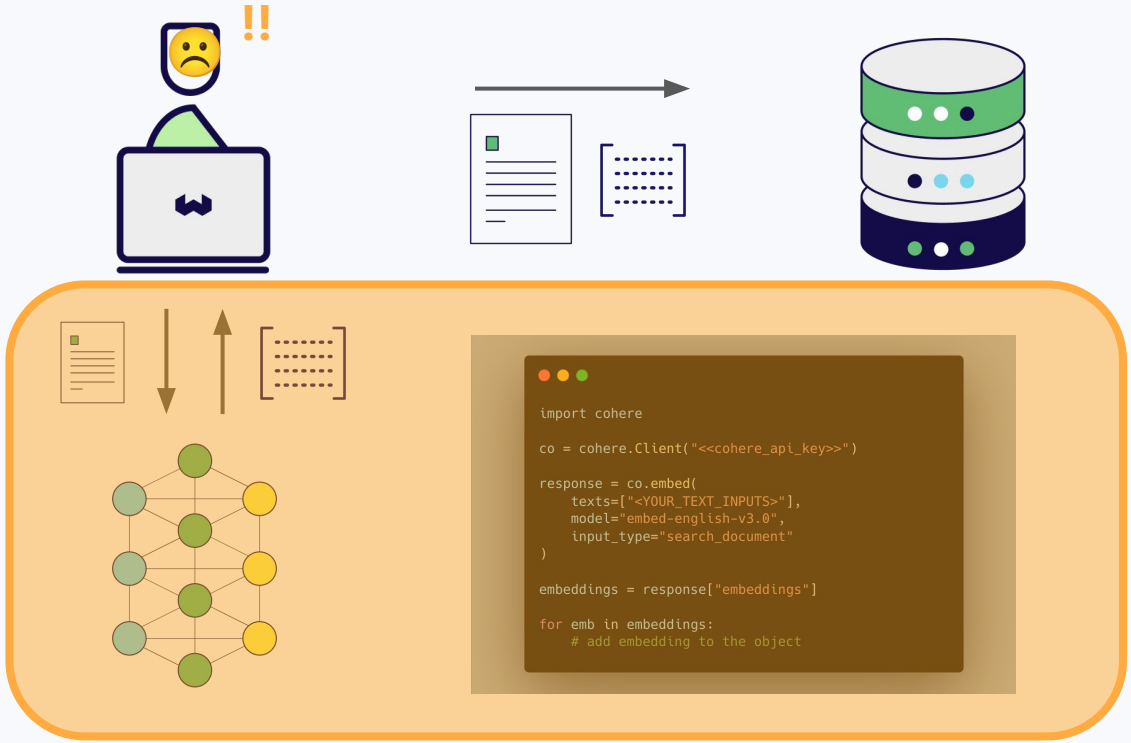
```python
import cohere

co = cohere.Client("<<cohere_api_key>>")

response = co.embed(
    texts=["<YOUR_TEXT_INPUTS>"],
    model="embed-english-v3.0",
    input_type="search_document"
)

embeddings = response["embeddings"]

for emb in embeddings:
    # add embedding to the object
```

Obtain Embeddings

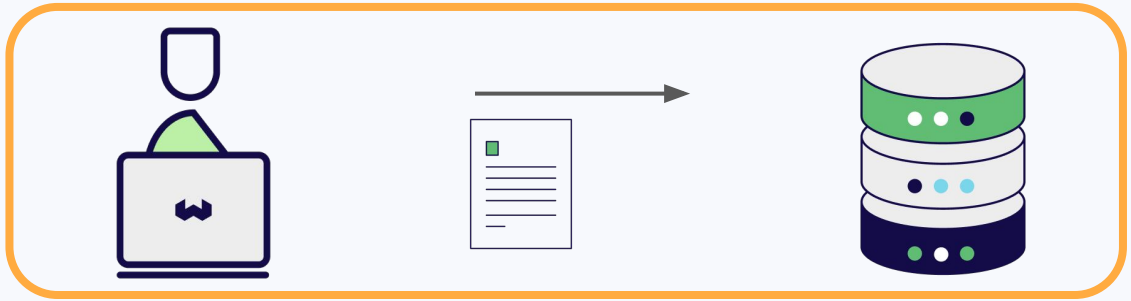


```
import cohere

co = cohere.Client("<<cohere_api_key>>")

response = co.embed(
    texts=["<YOUR_TEXT_INPUTS>"],
    model="embed-english-v3.0",
    input_type="search_document"
)

embeddings = response["embeddings"]

for emb in embeddings:
    # add embedding to the object
```

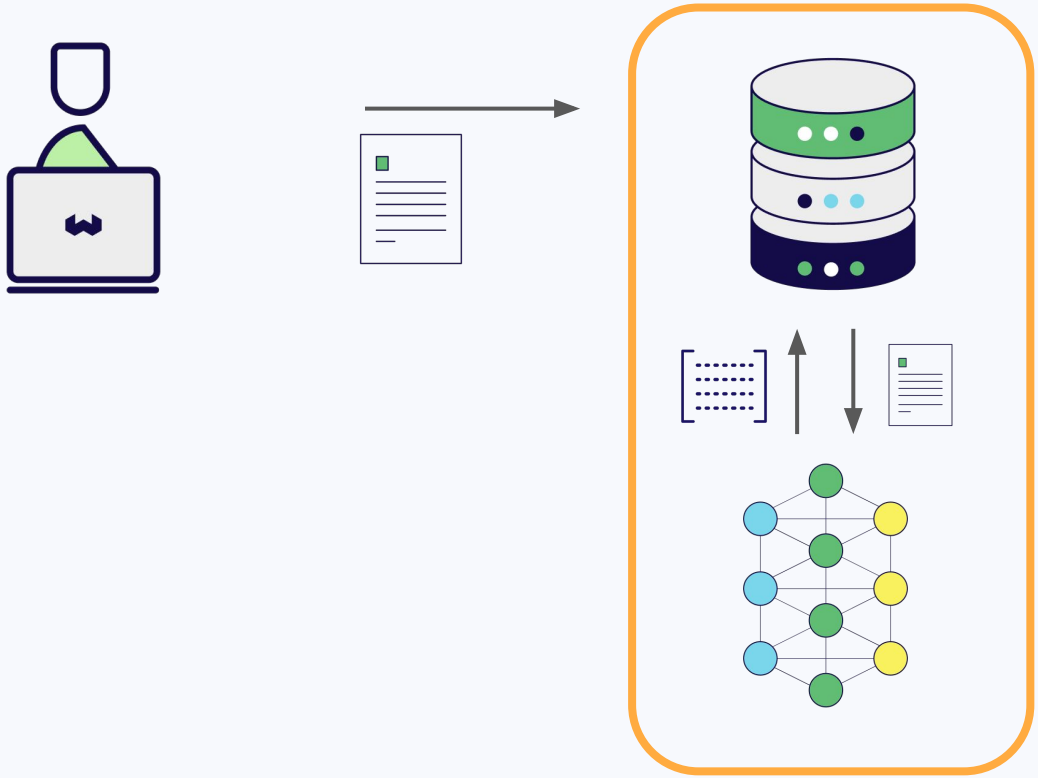
Overhead


!!
import cohere
co = cohere.Client("<<cohere_api_key>>")
response = co.embed(
texts=["<YOUR_TEXT_INPUTS>"],
model="embed-english-v3.0",
input_type="search_document"
)
embeddings = response["embeddings"]
for emb in embeddings:
# add embedding to the object

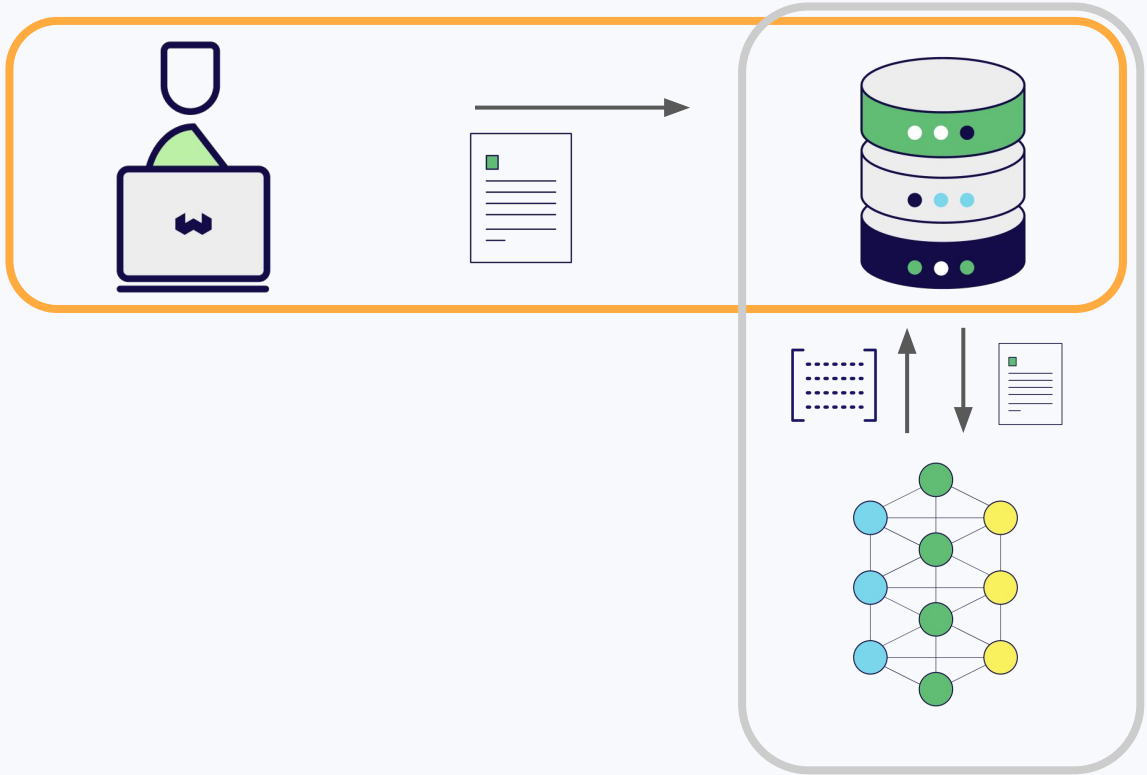
Ingest data



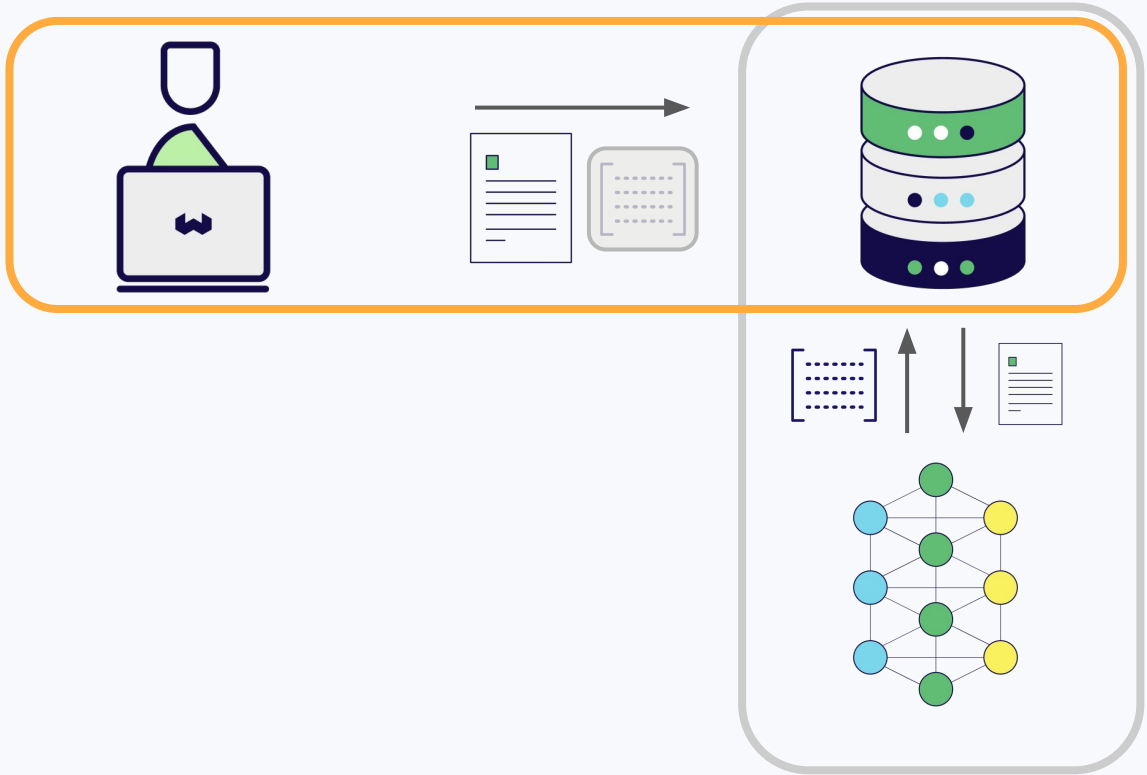
Have Weaviate obtain embeddings



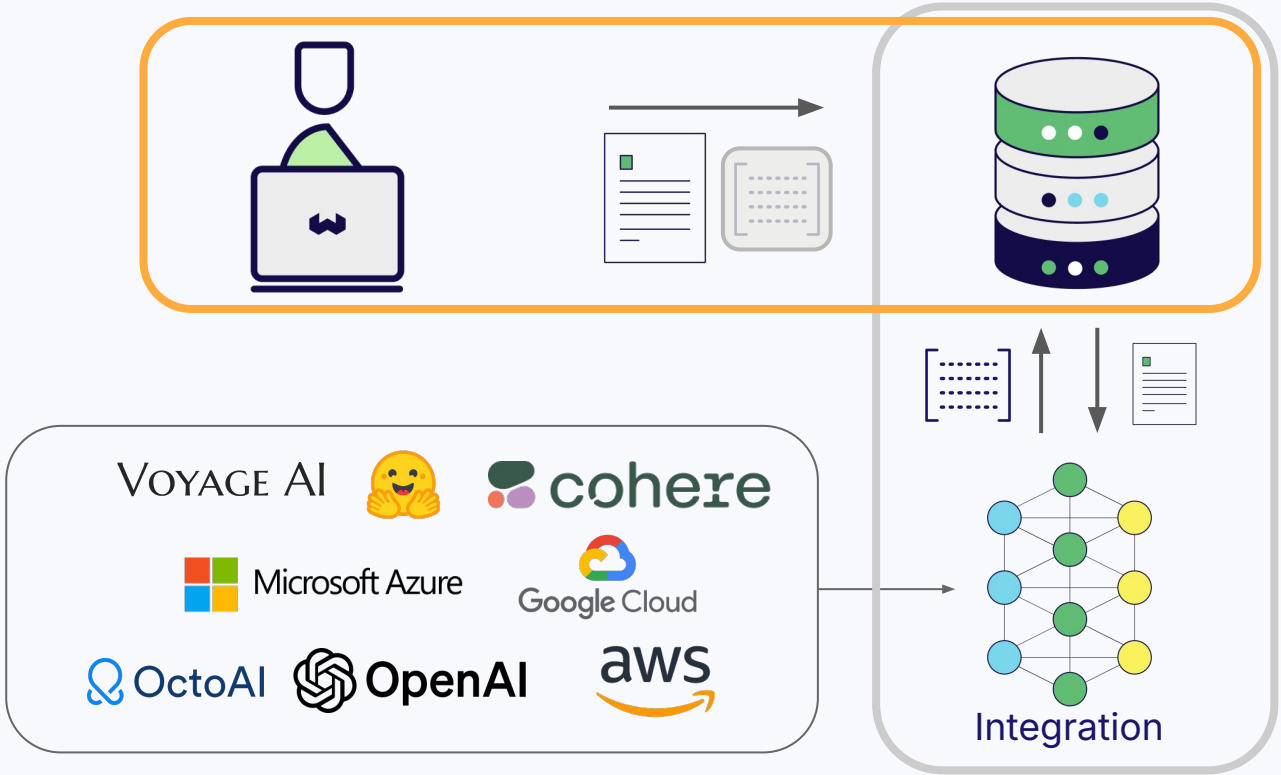
Ingest data (embeddings in background)



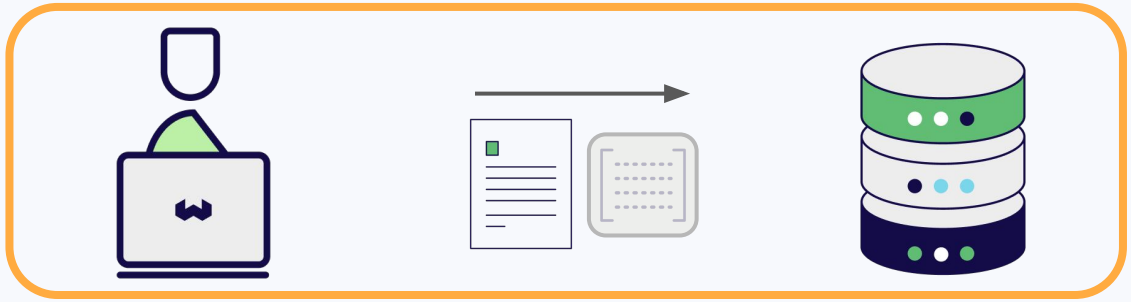
Ingest data (embeddings optional)



Ingest data (with provider integrations)



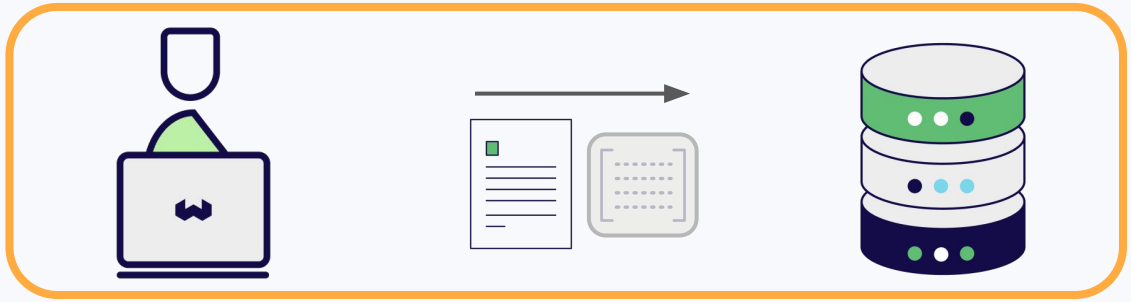
Simplified Experience



User experience

Under the hood

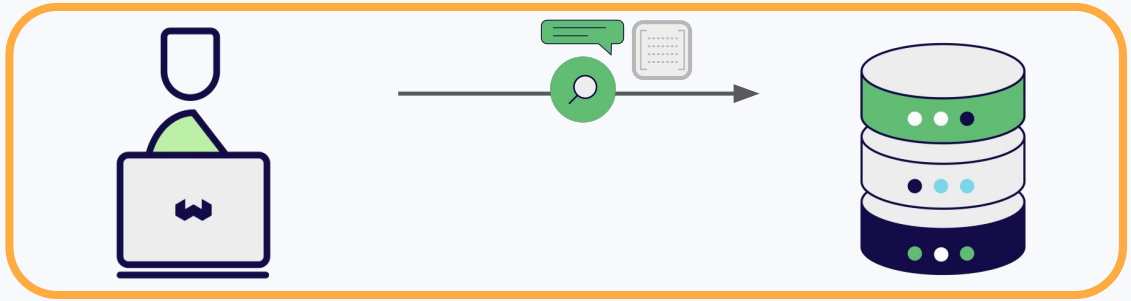
Simplified Experience



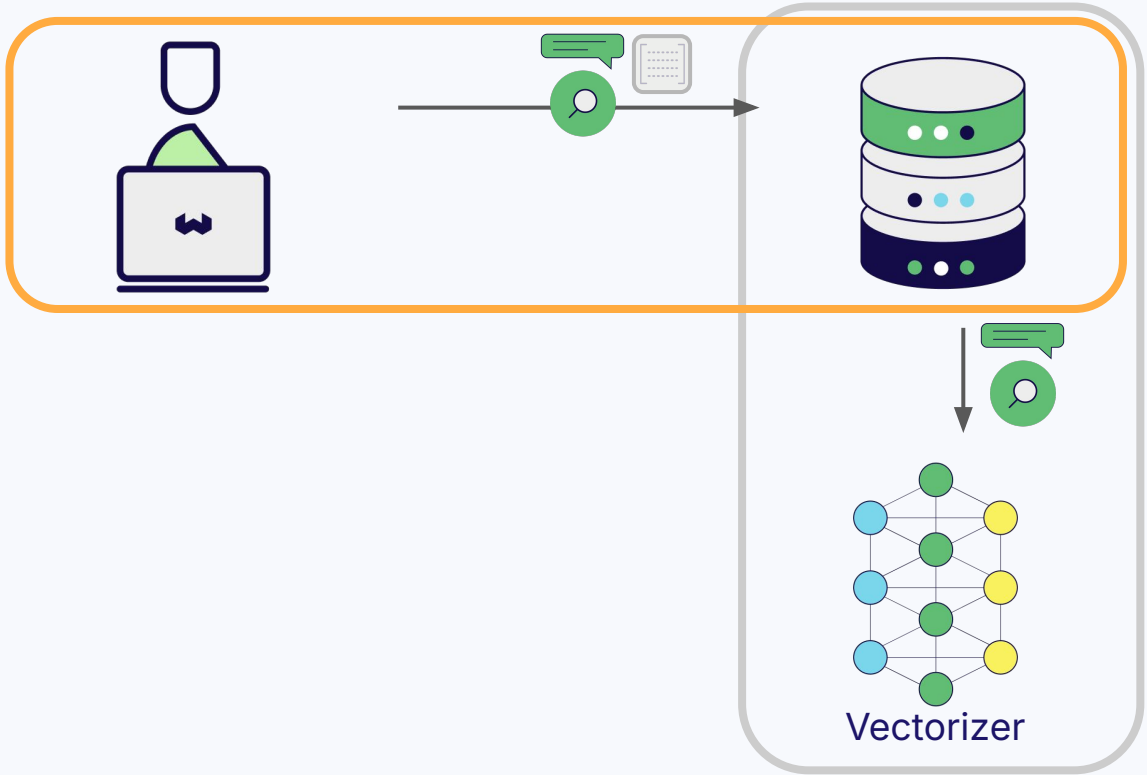
User experience

```
collection = client.collections.get("YourCollection")

with collection.batch.dynamic() as batch:
    for data_row in data_rows:
        batch.add_object(
            properties=data_row,
        )
```

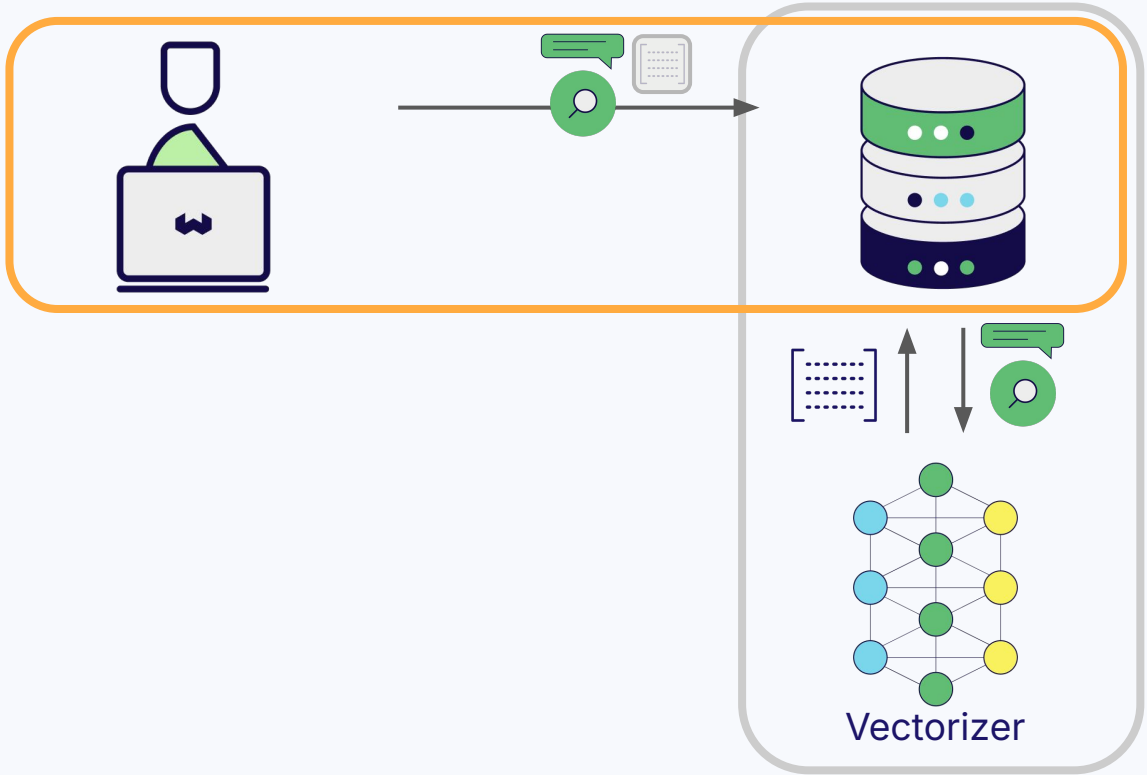
Query



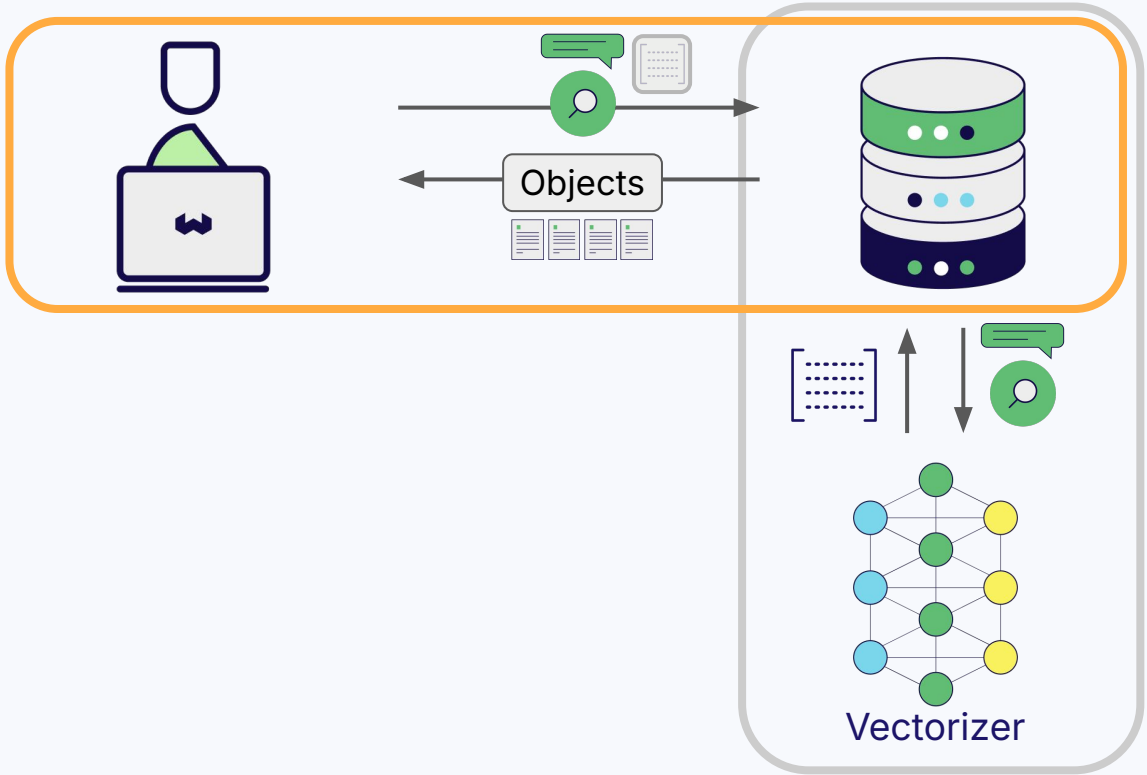
Query



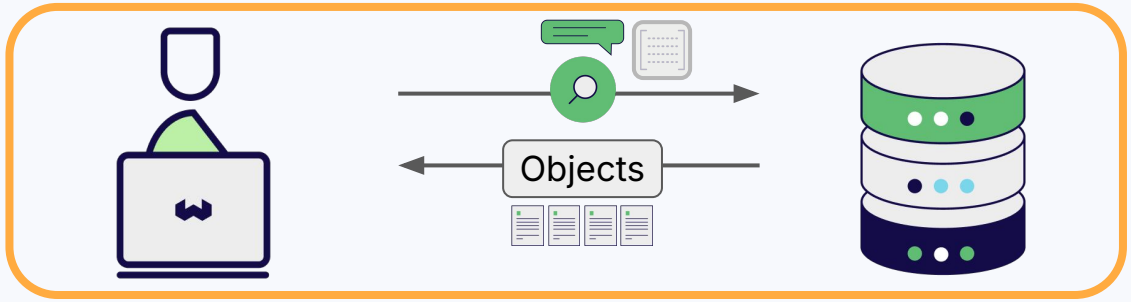
Query



Query



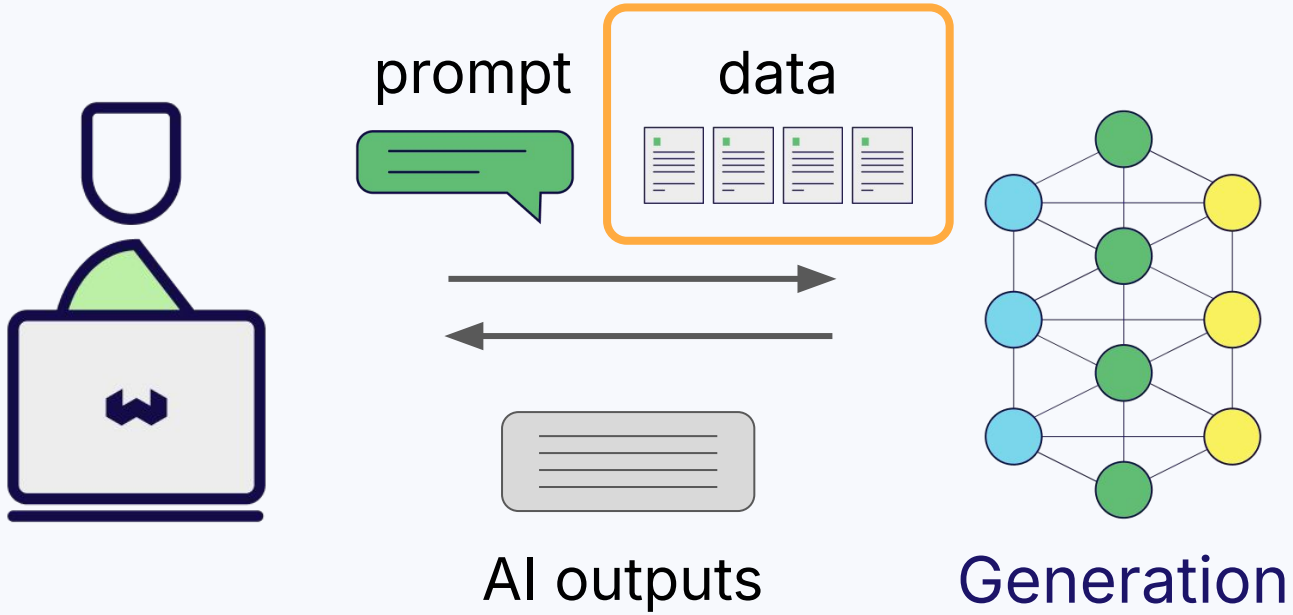
Query



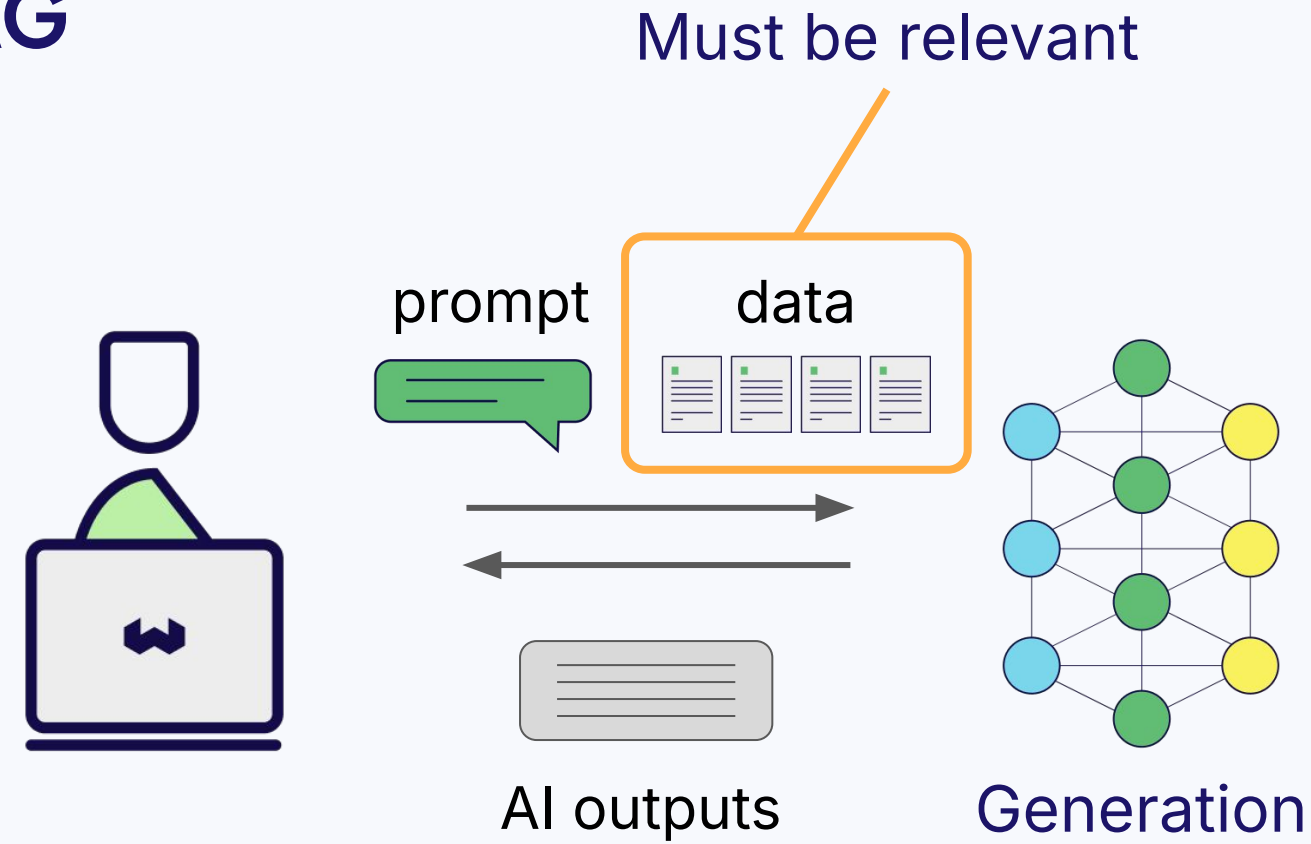
User experience

Under the hood

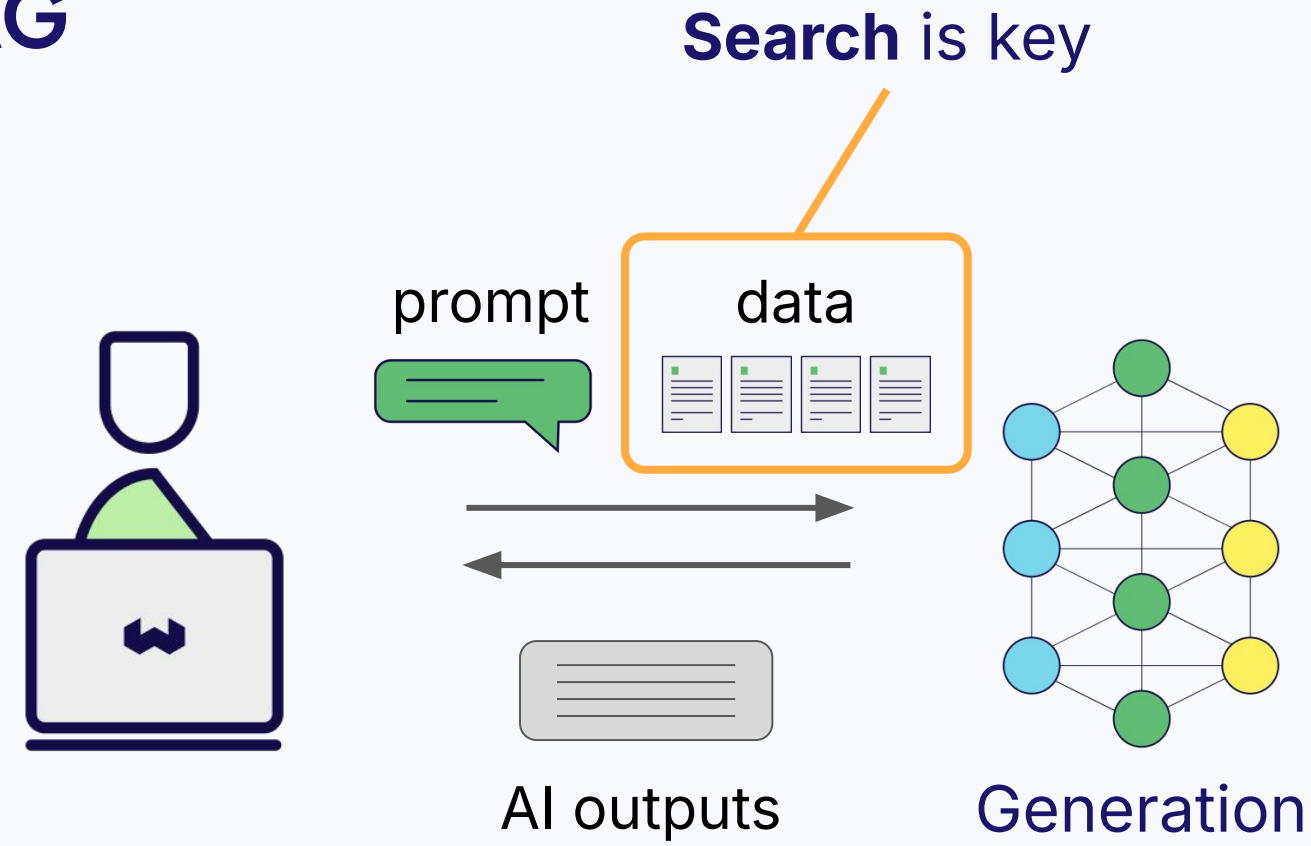
RAG



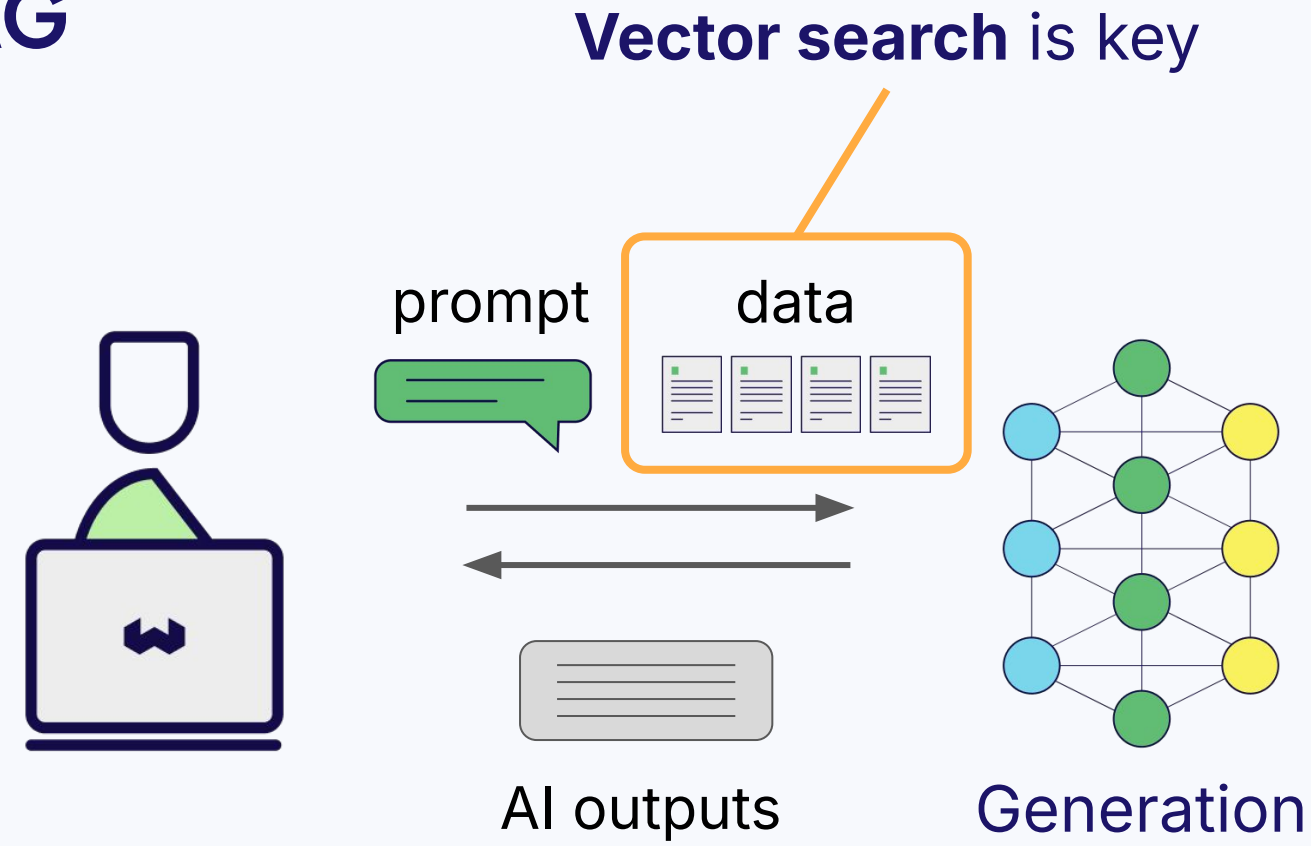
RAG



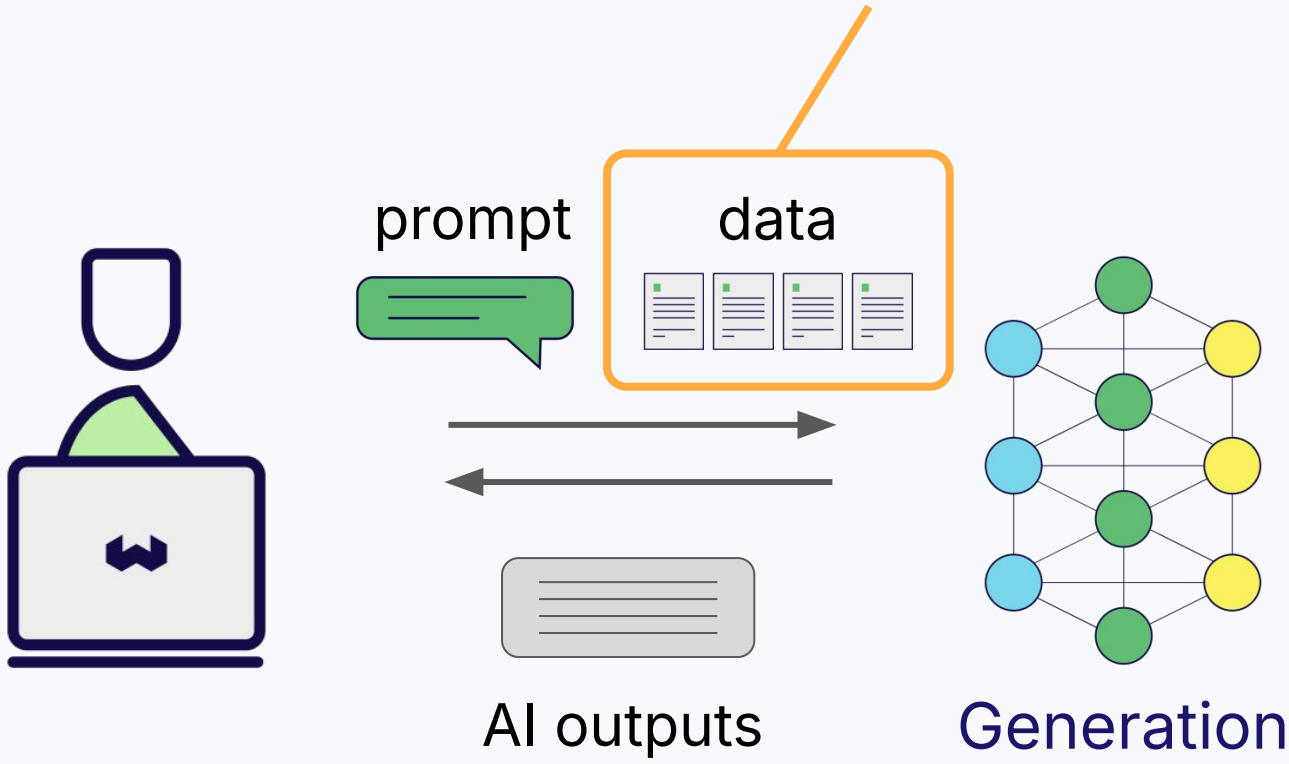
RAG



RAG

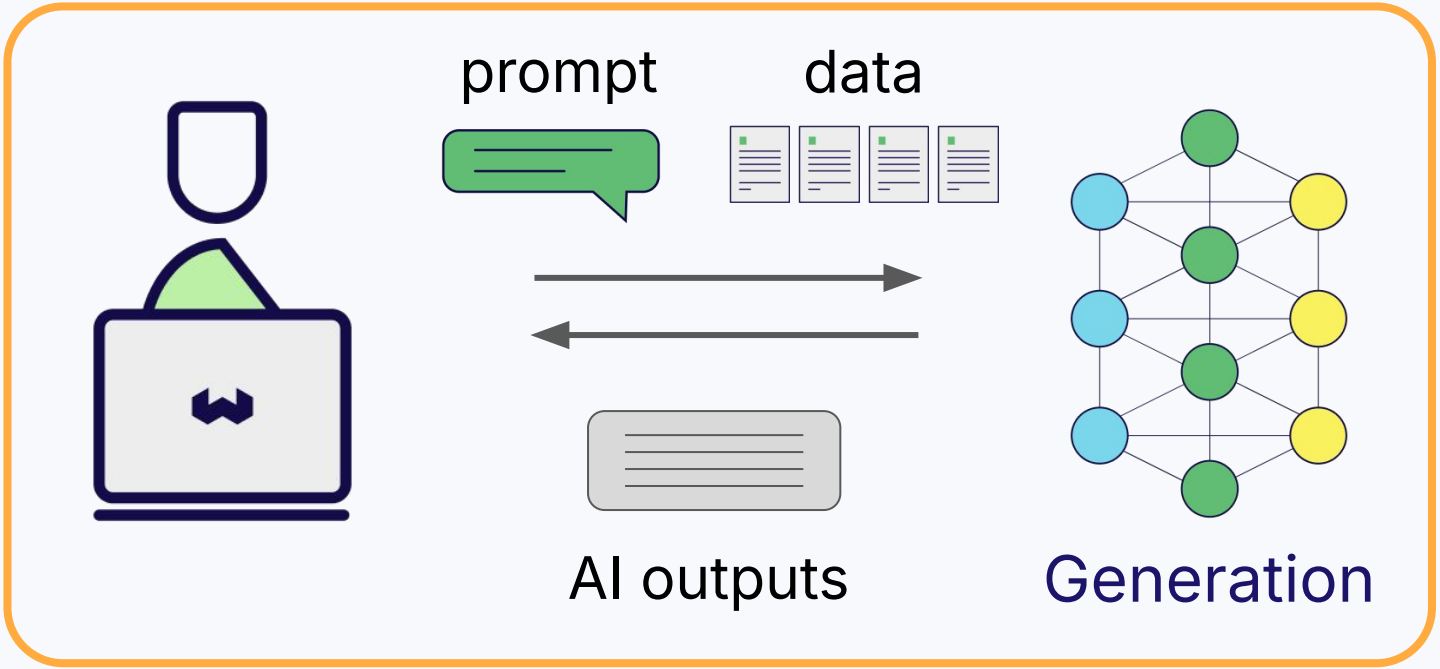


RAG

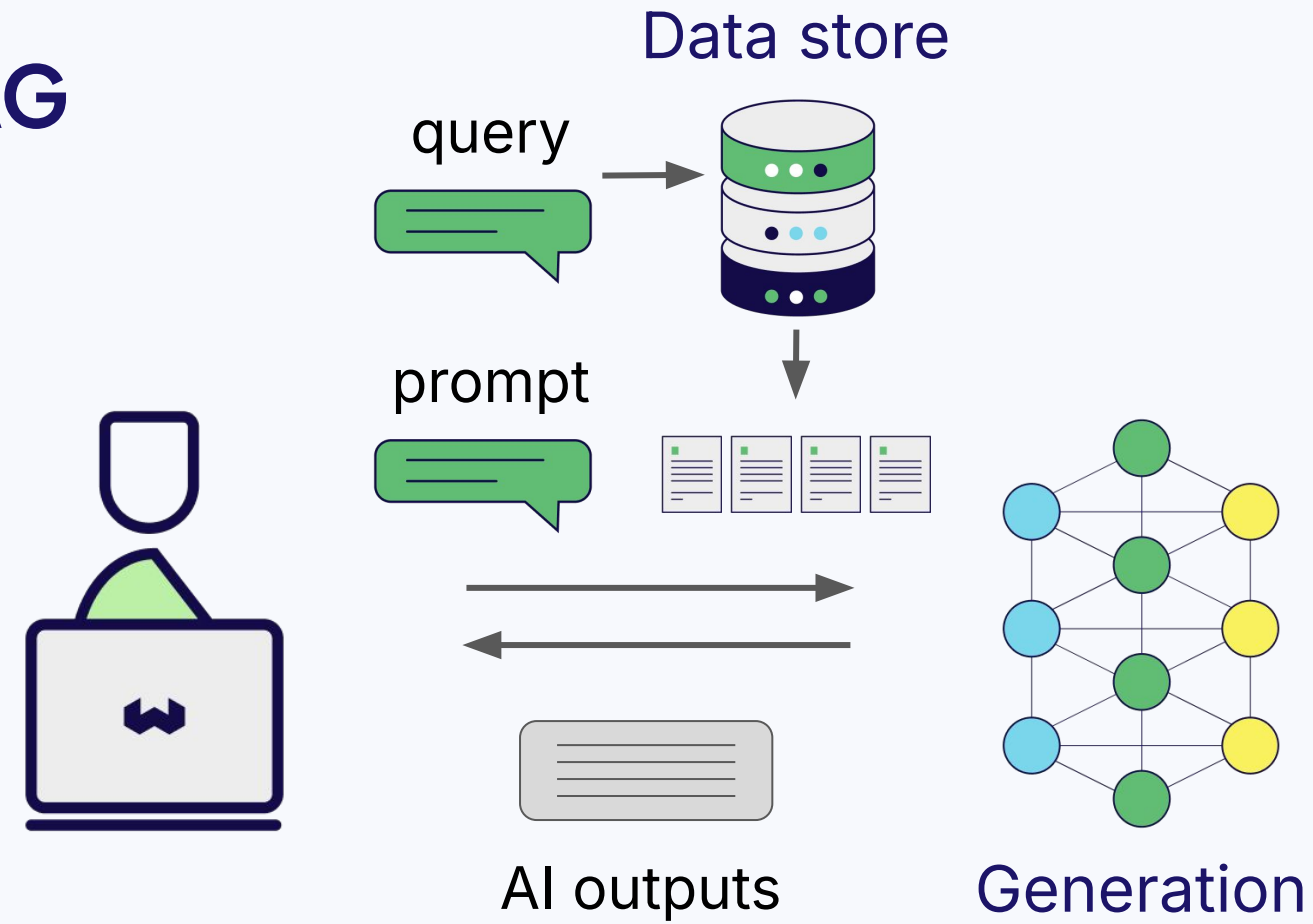

Vector search: relevance search
prompt
data
AI outputs
Generation

RAG

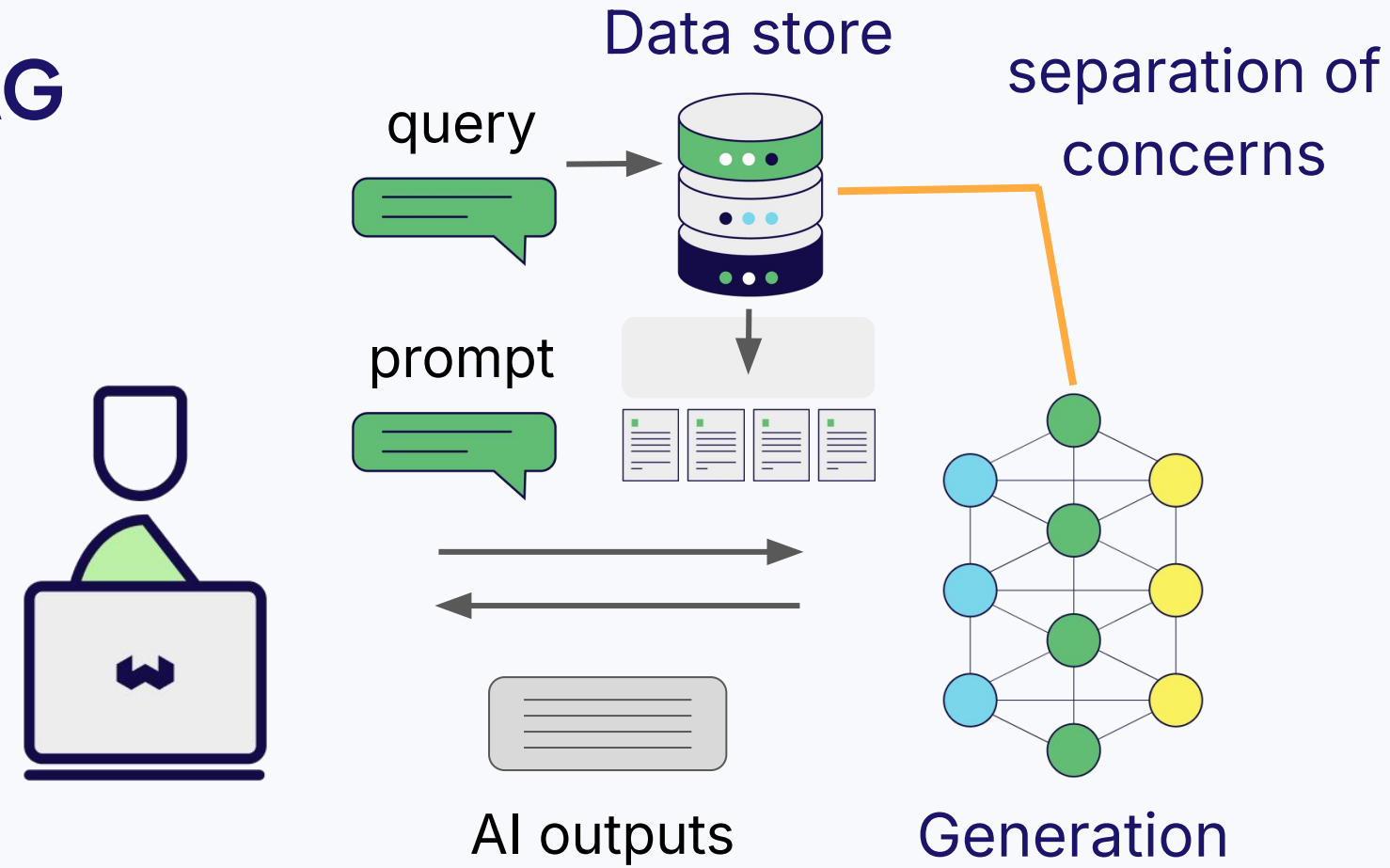
Why **vector search** is hot



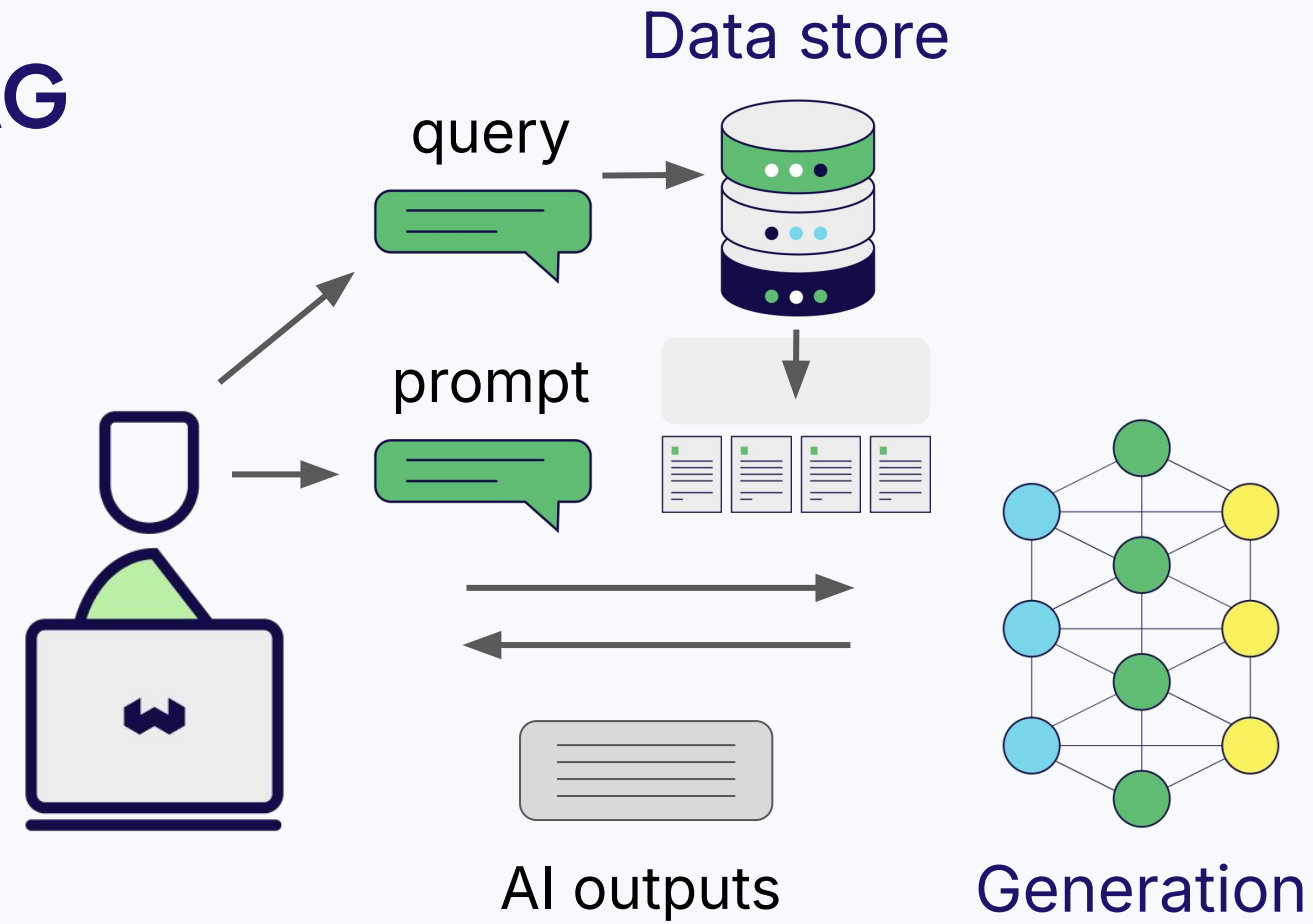
RAG



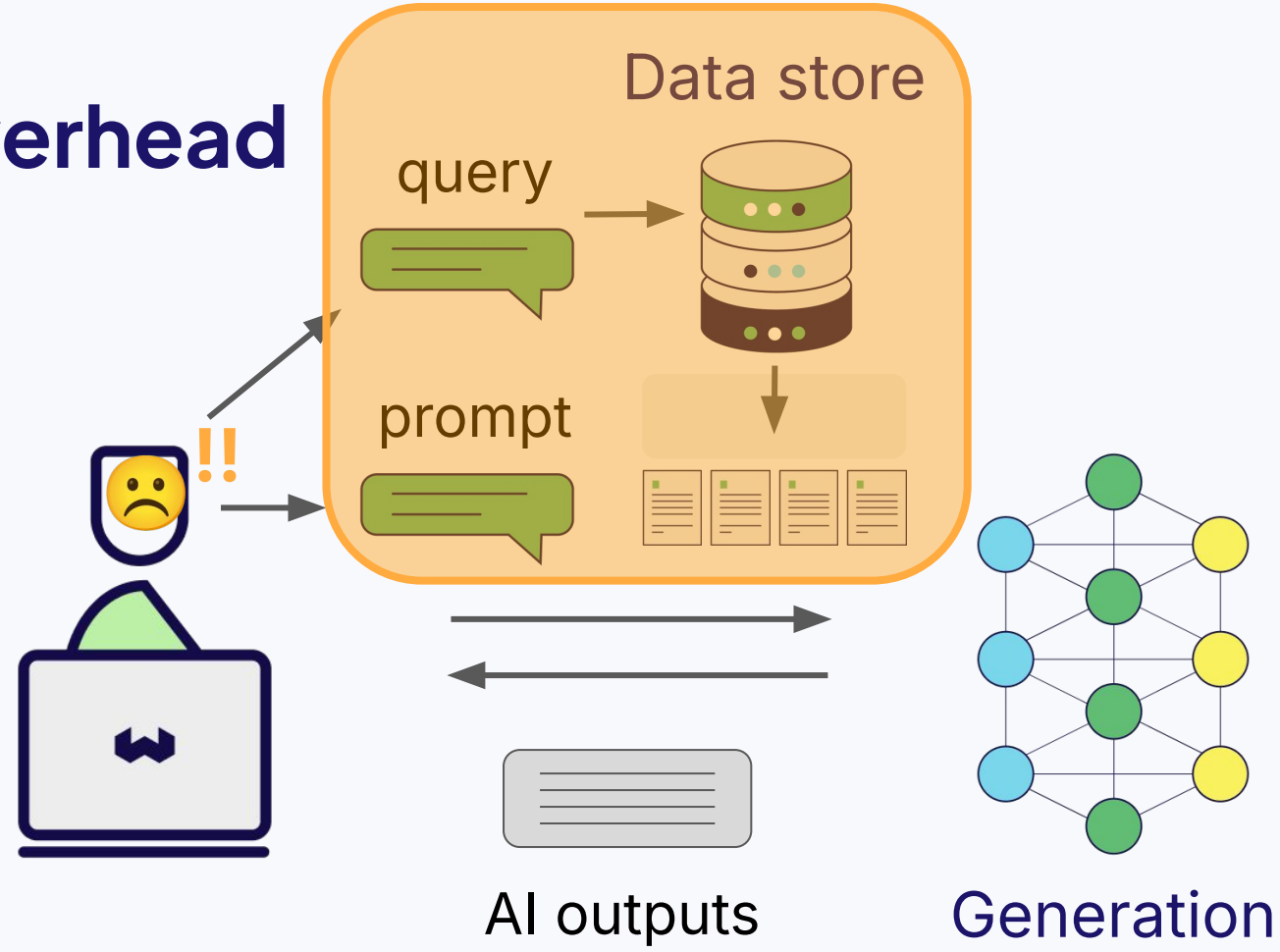
RAG


Data store
separation of concerns
query
prompt
AI outputs
Generation

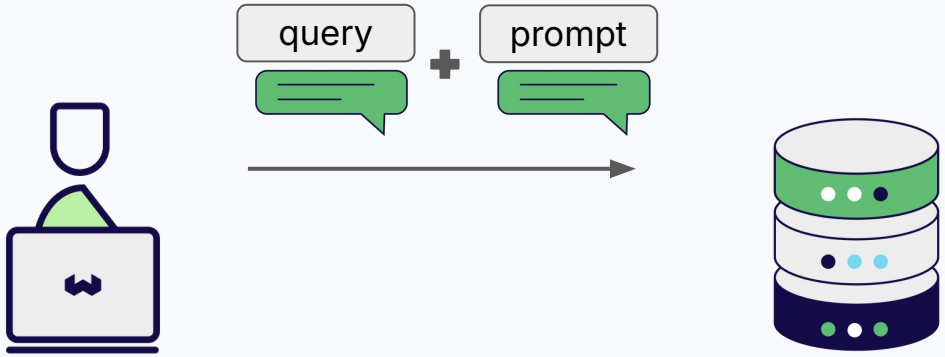
RAG



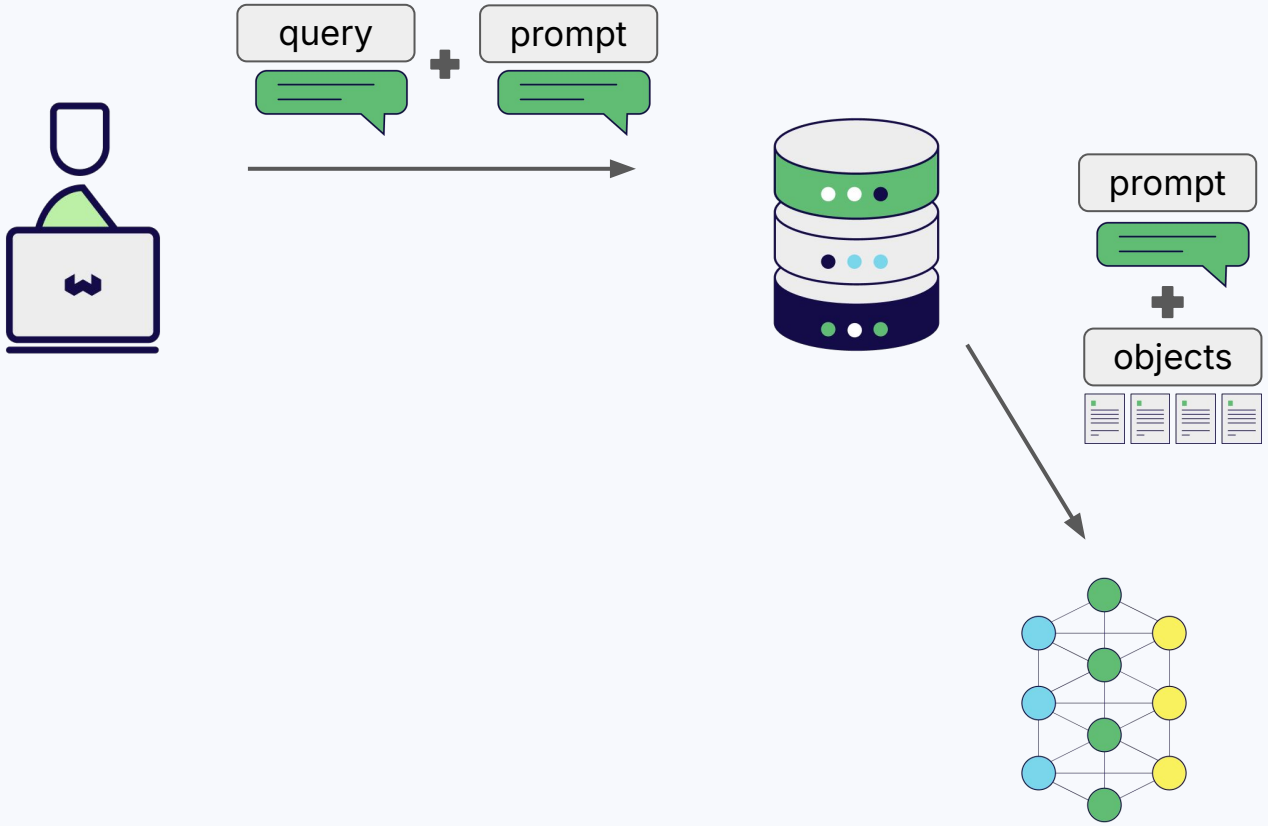
Overhead



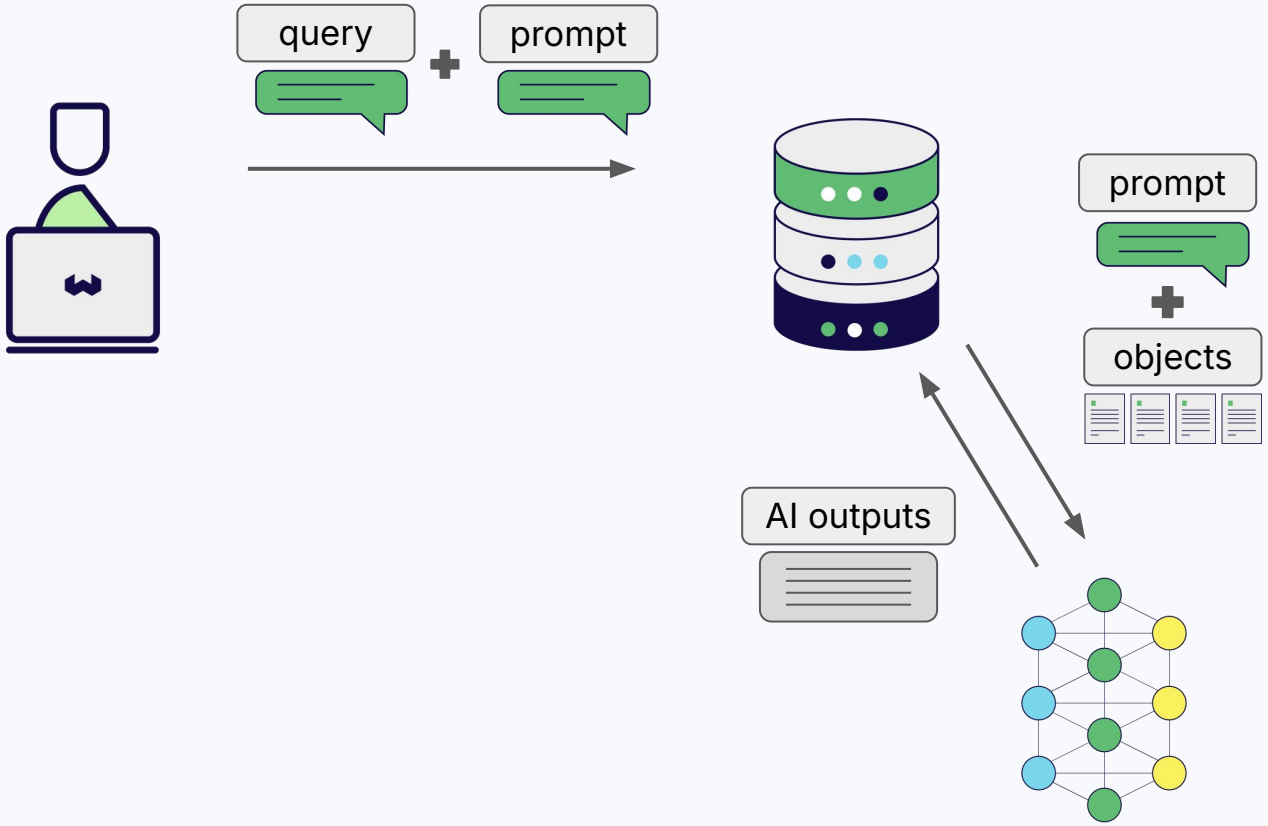
RAG



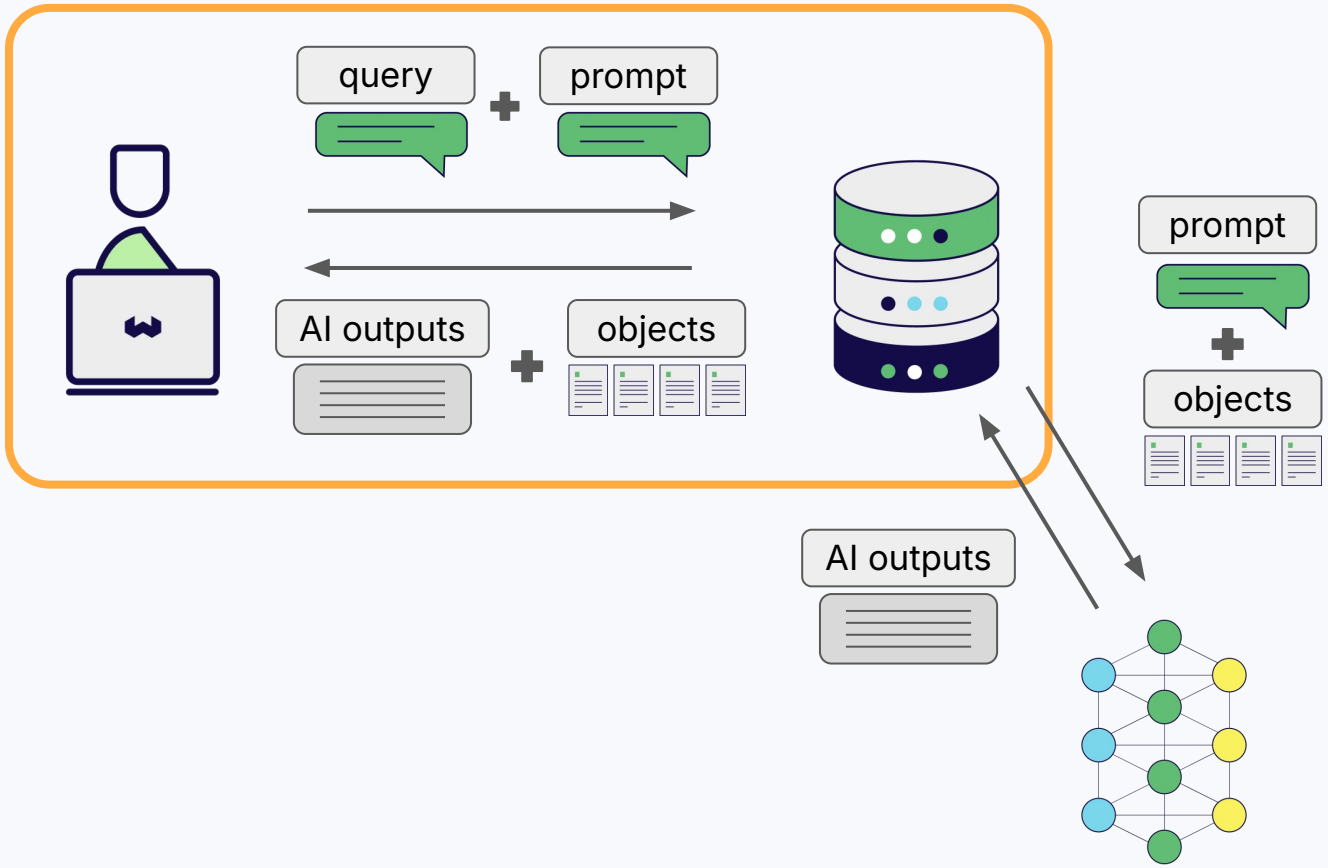
RAG



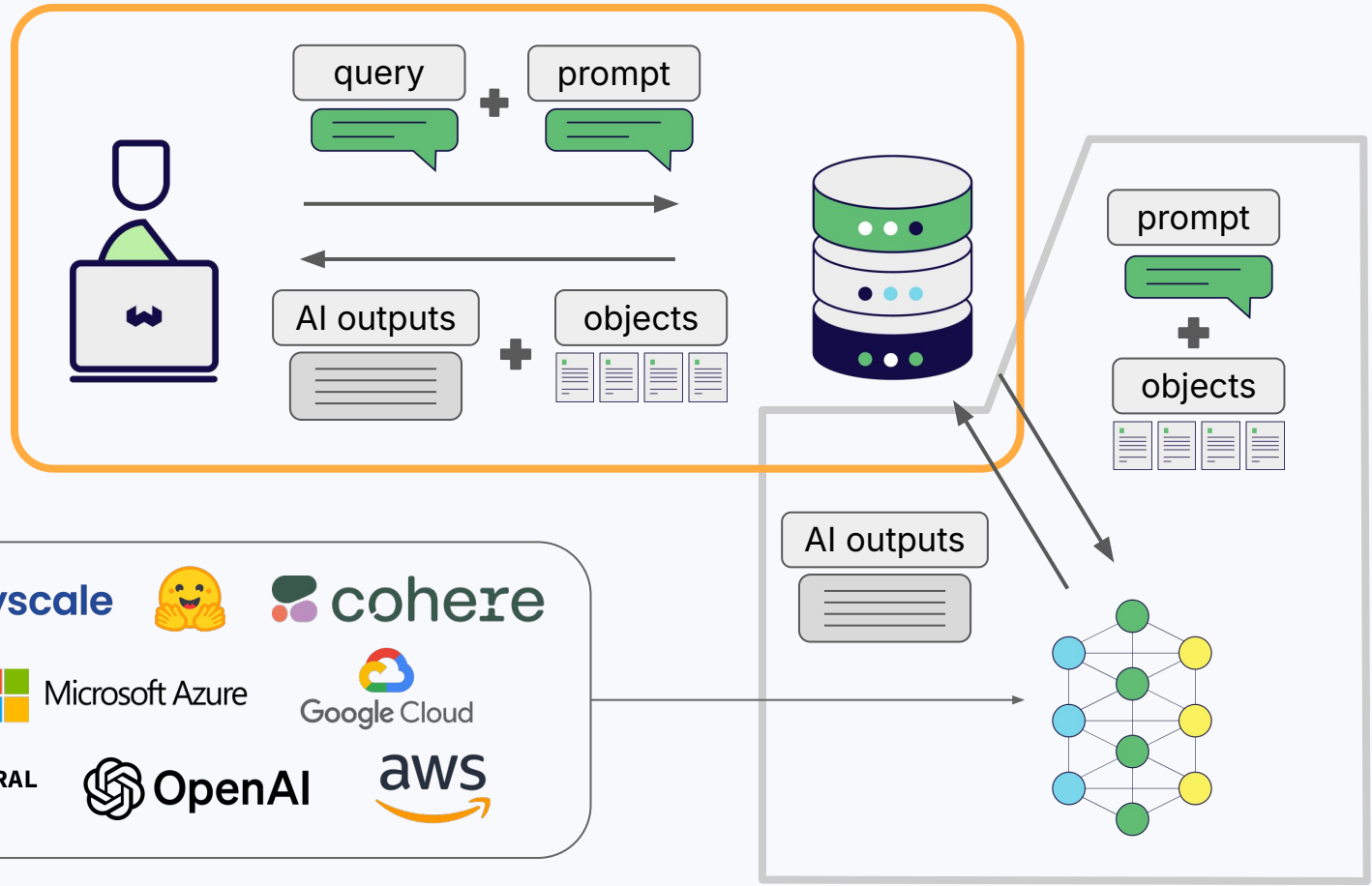
RAG



RAG



RAG


query
prompt
AI outputs
objects
prompt
objects
AI outputs
anyscale
cohere
Microsoft Azure
Google Cloud
MISTRAL AI_
OpenAI
aws

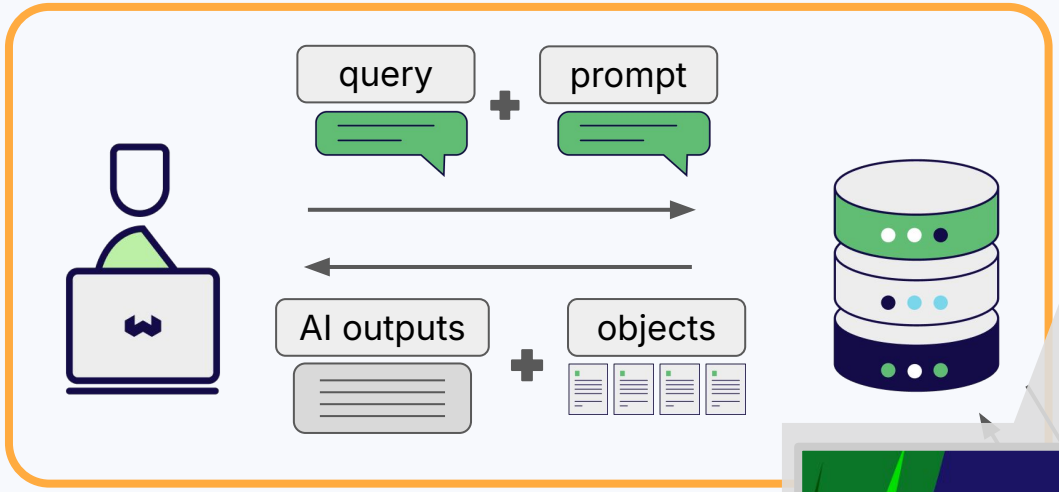
Simplified Experience

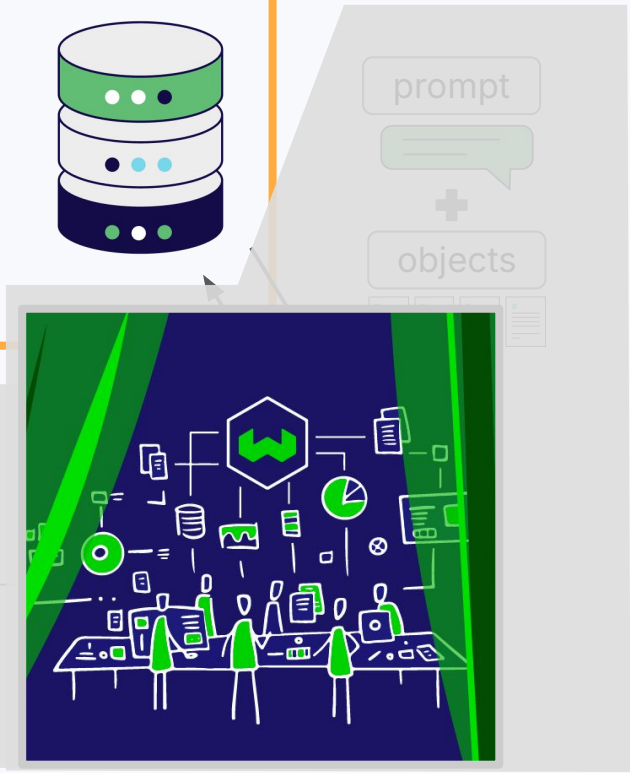
```python
collection = client.collections.get("YourCollection")
response = collection.generate.hybrid(
    query="Multi-tenancy in Weaviate"
    limit=3,
    grouped_task="Explain how multi-tenancy works"
)

print(response.generated)
```

User experience (RAG)

RAG

User experience





anyscale
cohere
Microsoft Azure
Google Cloud
MISTRAL AI_
OpenAI
aws