weaviate

Build AI-supercharged RAG apps with a Vector Database

JP Hwang
Technical Curriculum Developer

Material:

https://github.com/weaviate-tutorials/workshop-oss-2024

Start with the README instructions

BONUS MATERIAL

For self-guided users

Go to:

- **https://github.com/weaviate-tutorials/workshop-oss-2024**

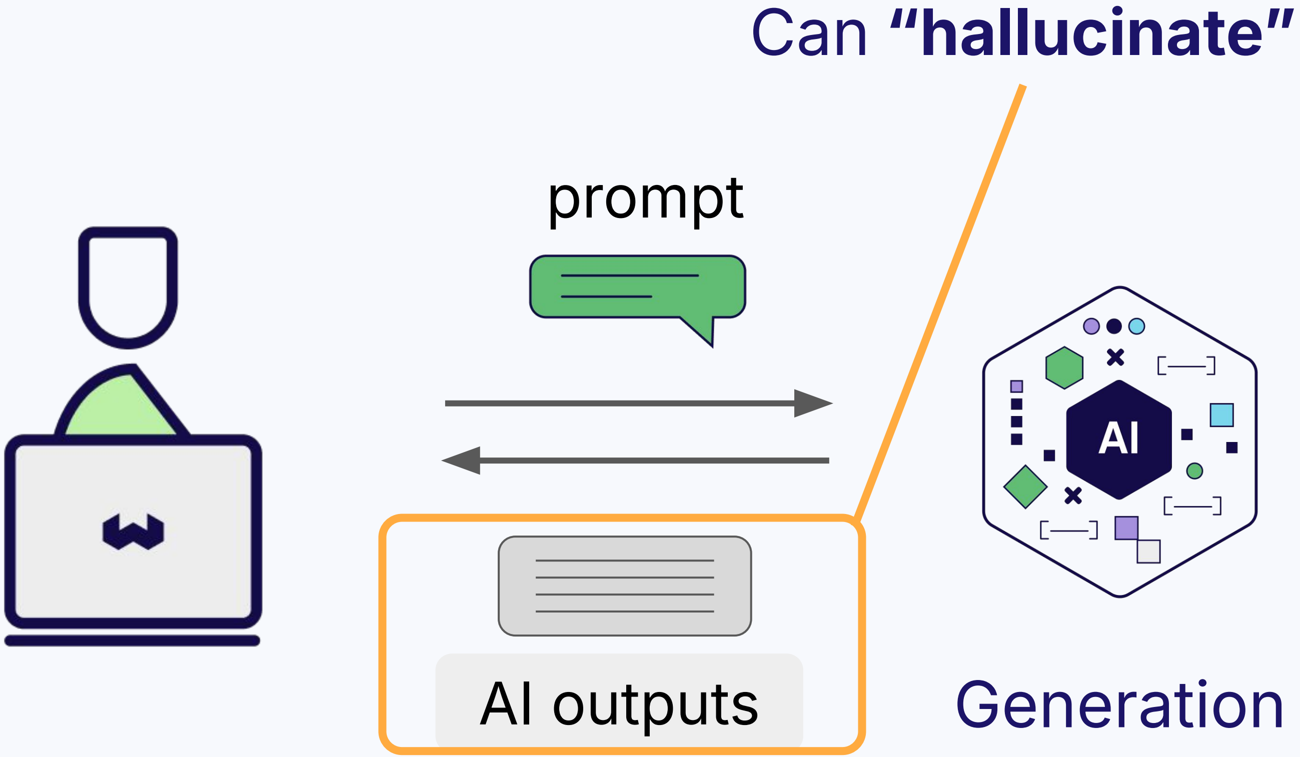
Start with the README instructions

See:

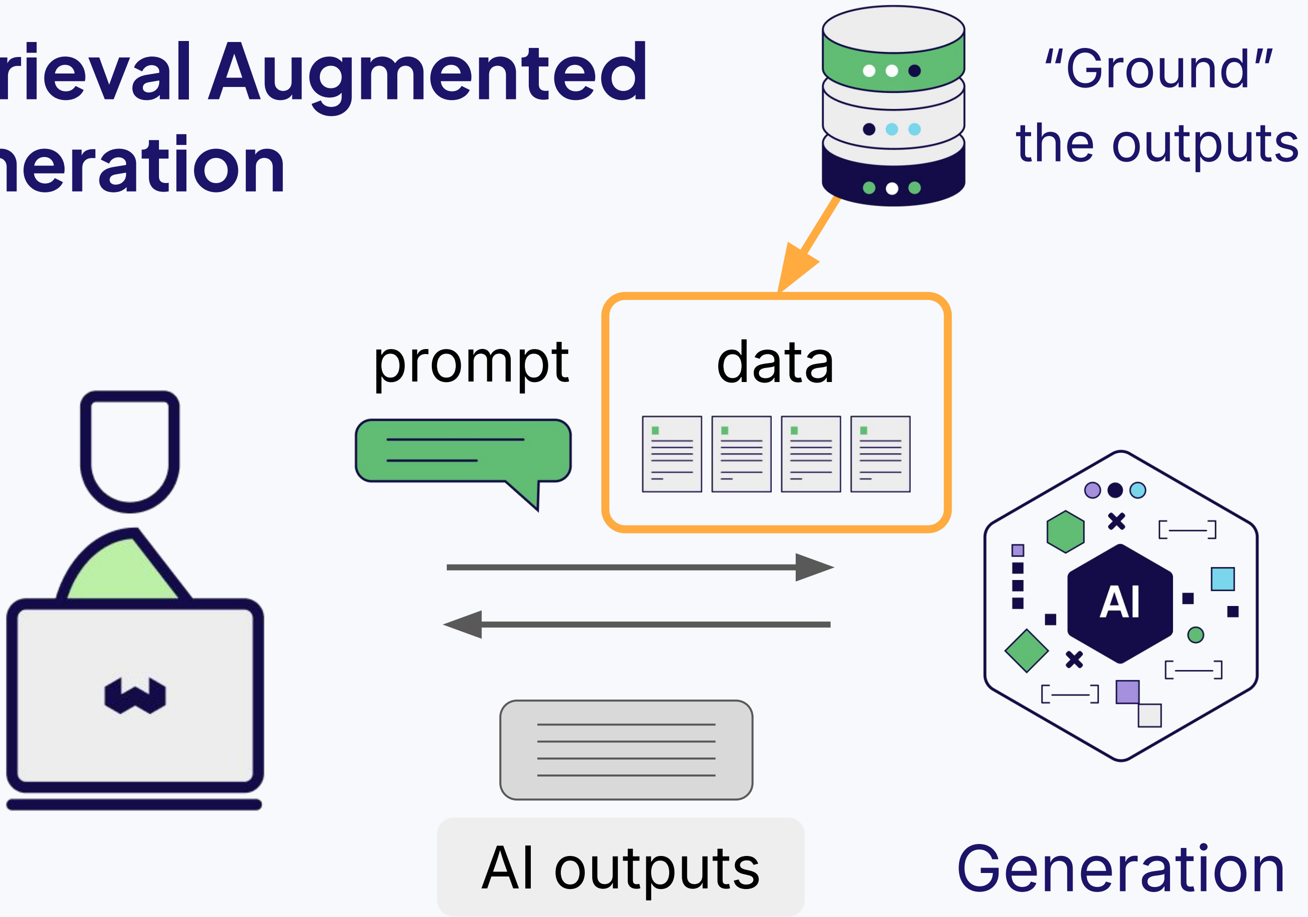
- **The “completed” workshop notebook**
- **The “hints” directory**

Recap: Vector DBs & RAG

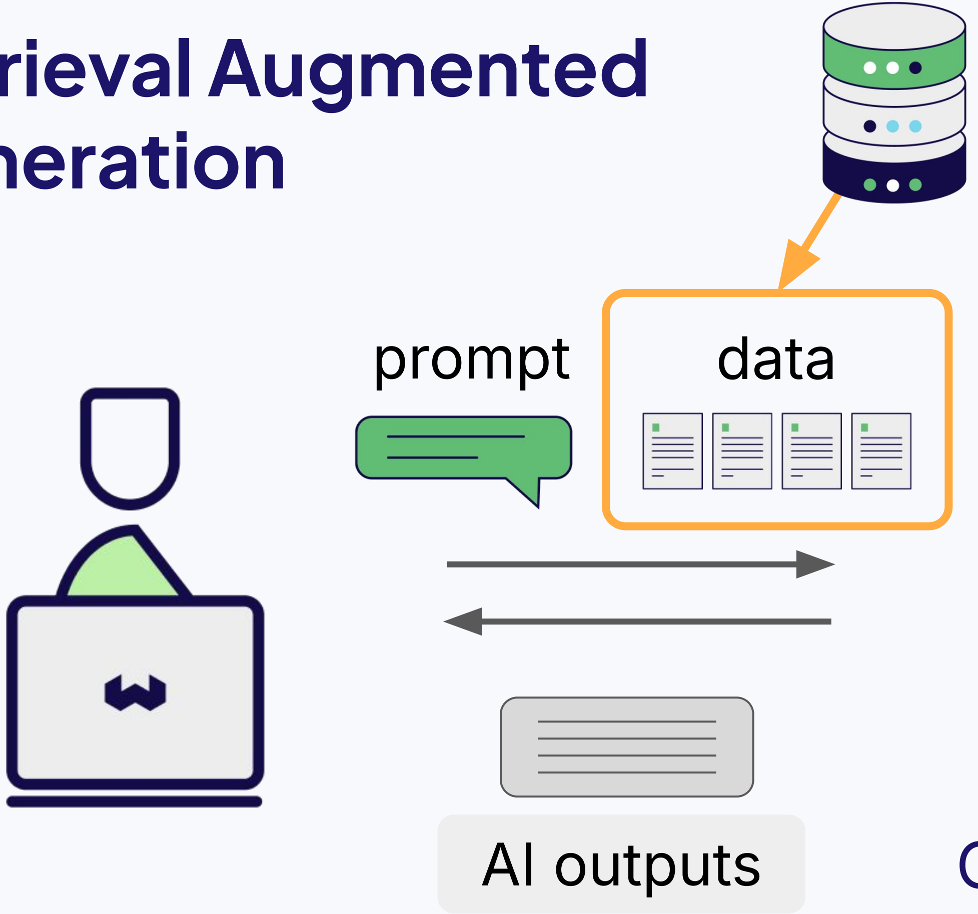
Large Language Model

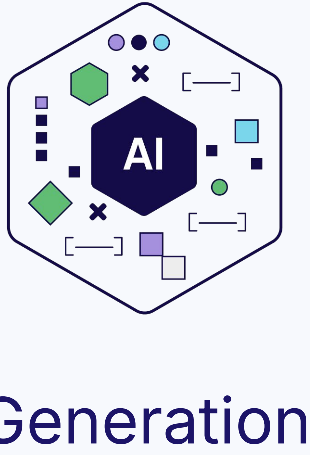


Retrieval Augmented Generation



Retrieval Augmented Generation





But... DBs are big!

Scaling up / scaling out

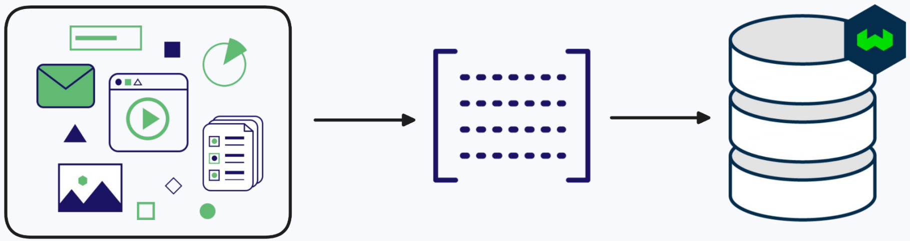
Scale: Considerations

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

Managing resource requirements

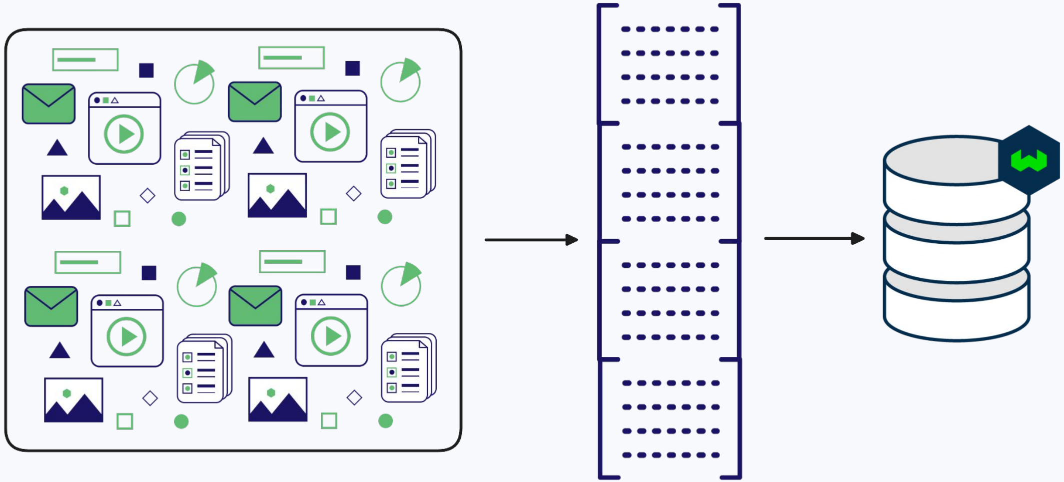
Scale: Solutions

Growth



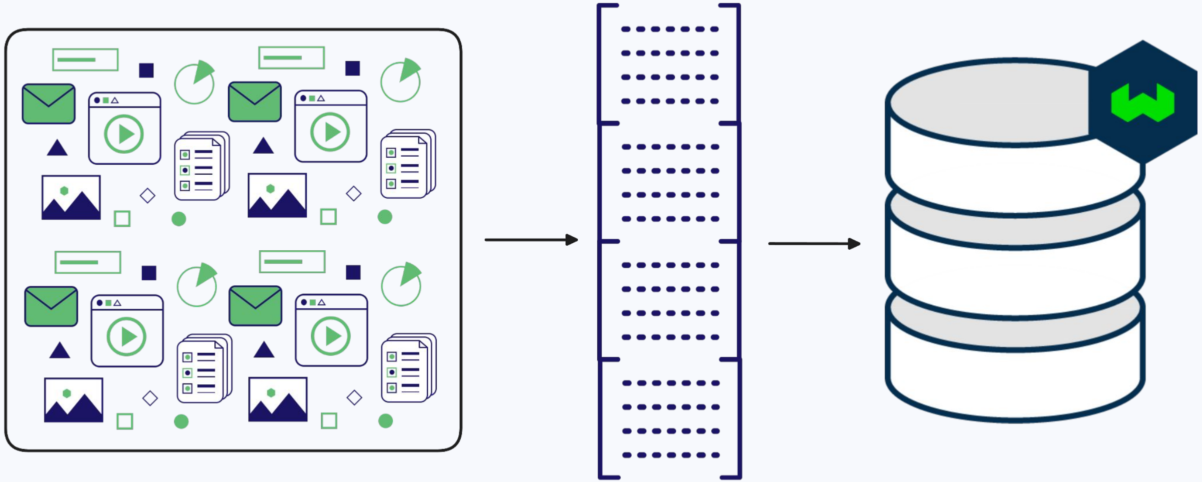
Scale: Solutions

Growth



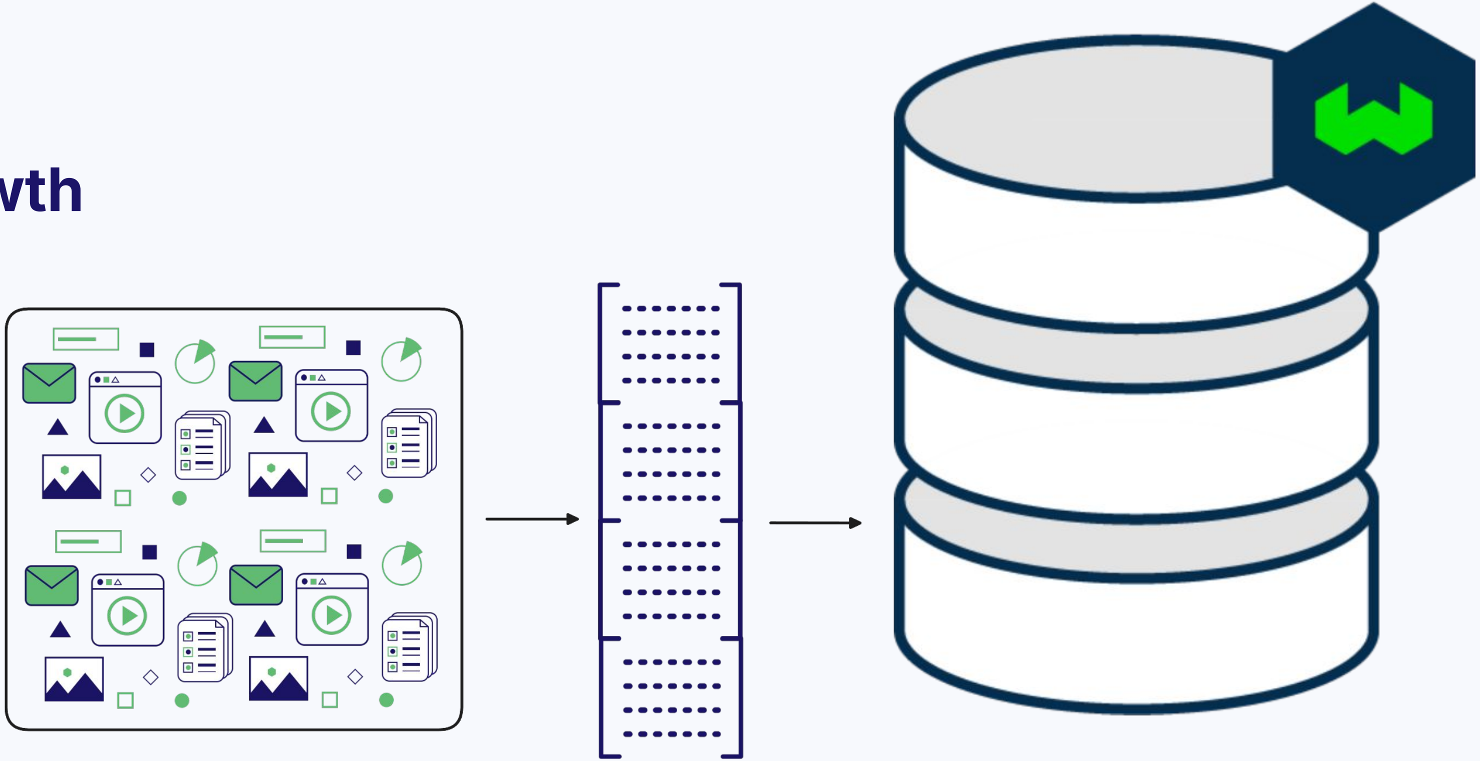
Scale: Solutions

Growth



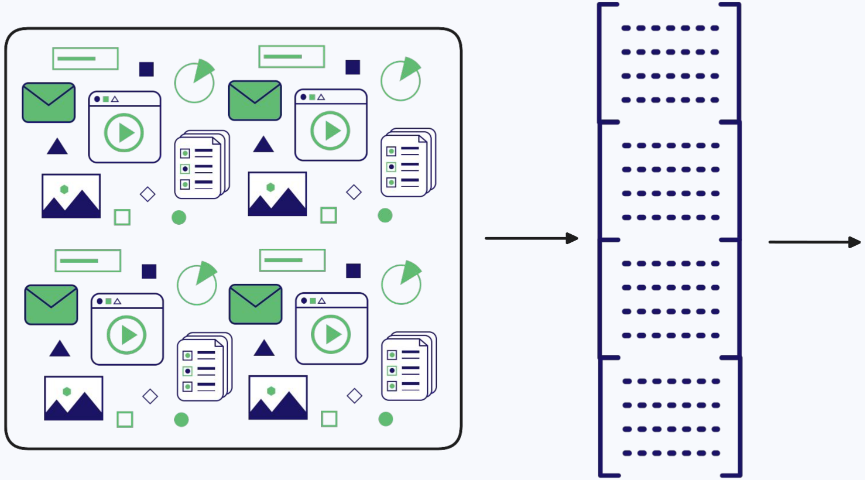
Scale: Solutions

Growth



Scale: Solutions

Growth

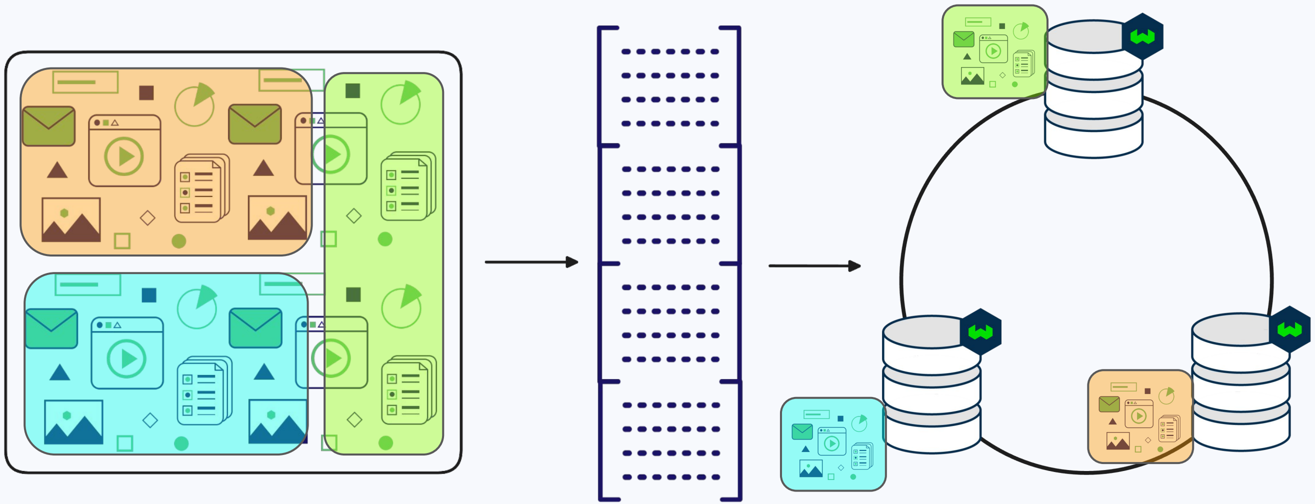


- Single point of failure
- Costs
- Efficiency
- Upgrades

Scale: Solutions

Deal with growth

Horizontal scaling
(sharding)



Scale: Solutions

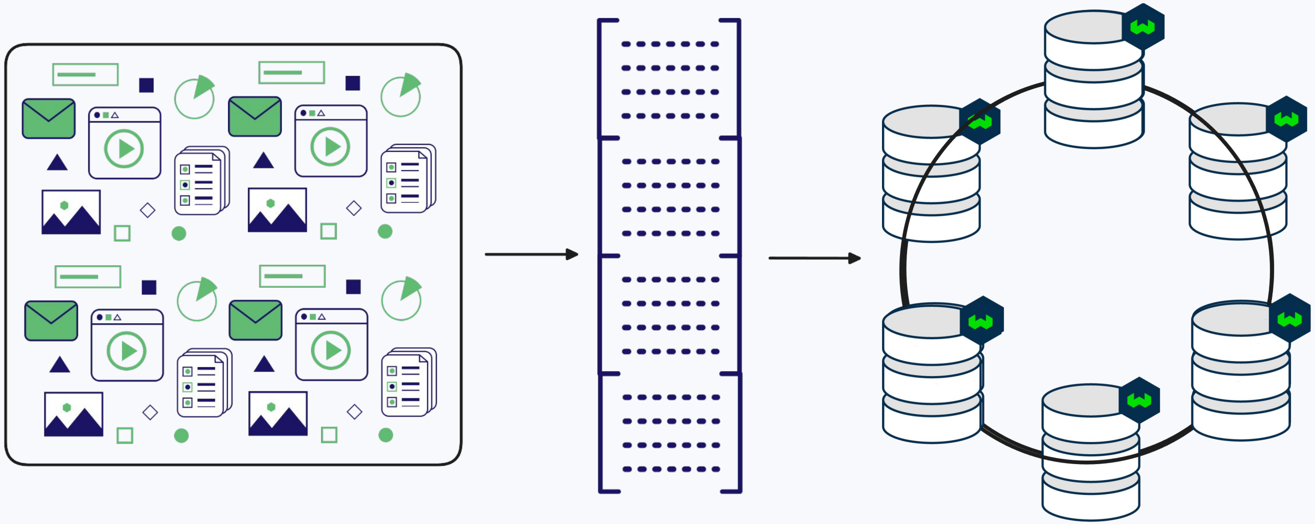
Billion scale

since 2022!

Deal with growth

To scale:

Add more nodes



Scale: Solutions

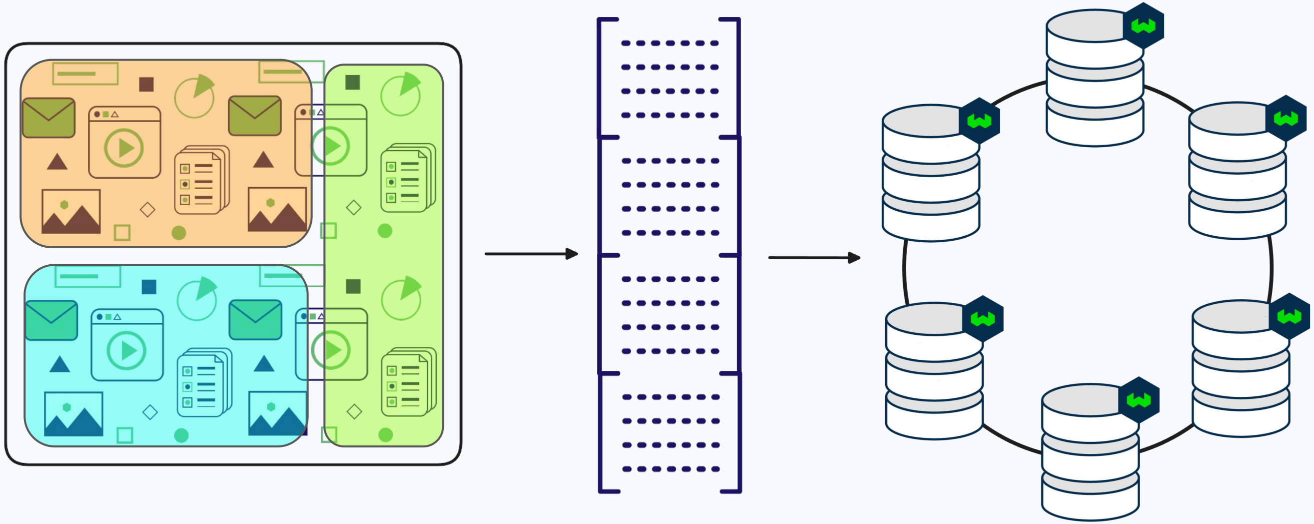
Billion scale

since 2022!

Deal with growth

To scale:

Add more nodes

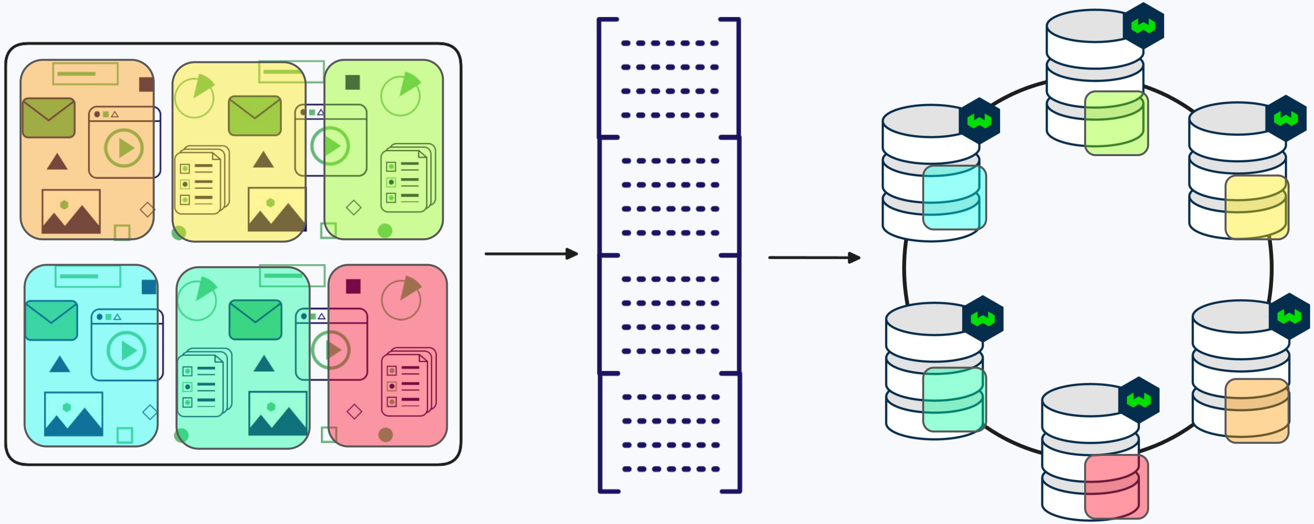


Scale: Solutions

Billion scale since 2022!

Deal with growth

To scale:
Add more nodes



Vector indexing options

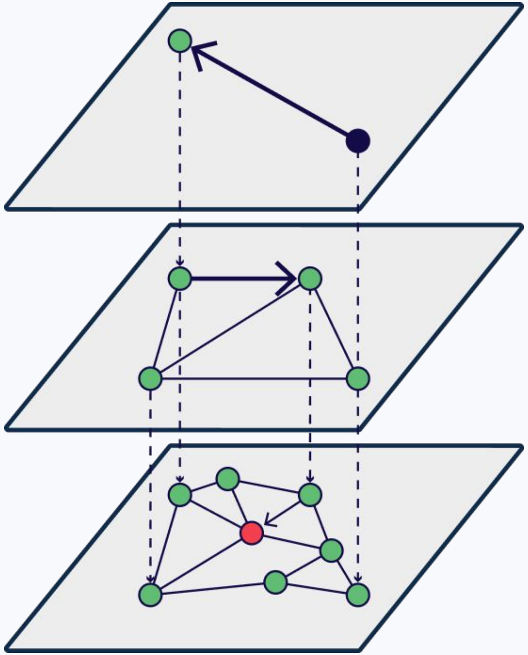
Scale: Solutions

Improve efficiency - indexing

Scale: Solutions

Improve efficiency - indexing

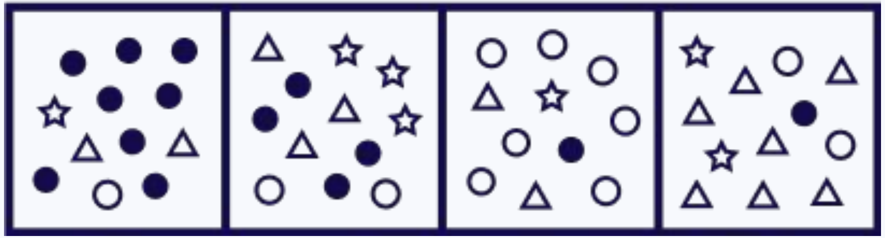
- **HNSW** index (default)



Scale: Solutions

Improve efficiency - indexing

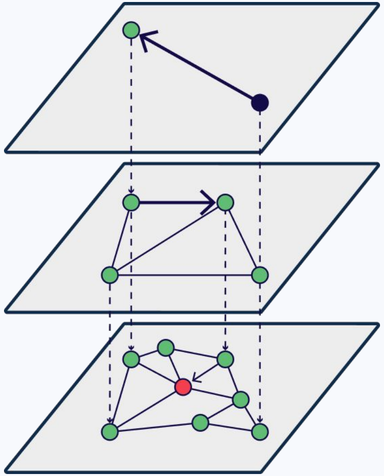
- **HNSW** index (default)
- **Flat** index

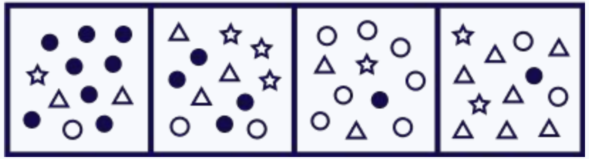


Scale: Solutions

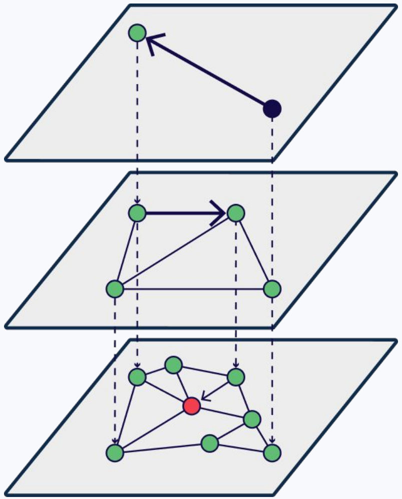
Indexes - comparison

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





Scale: Solutions

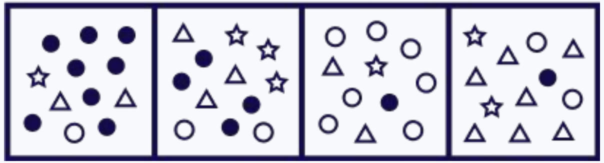


How to choose?

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

  (Tune speed / size / accuracy)

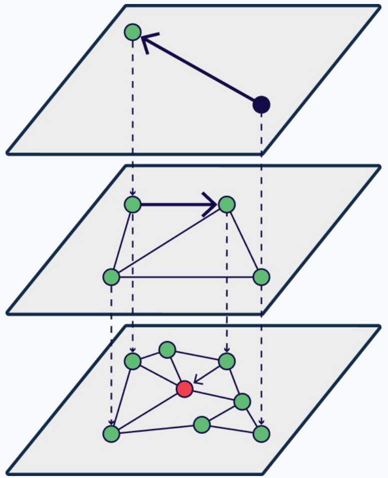
- Multi-tenancy?
  - Try **dynamic**

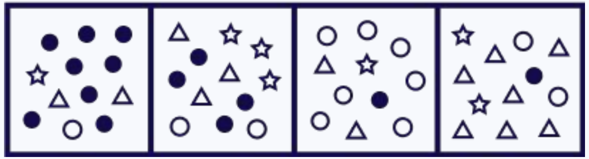


Scale: Solutions

Improve efficiency - indexing

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

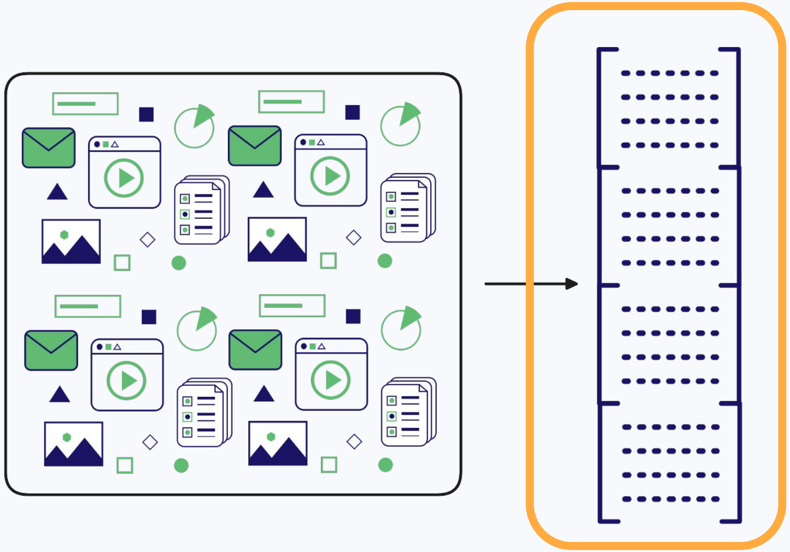




Vector quantization

Scale: Solutions

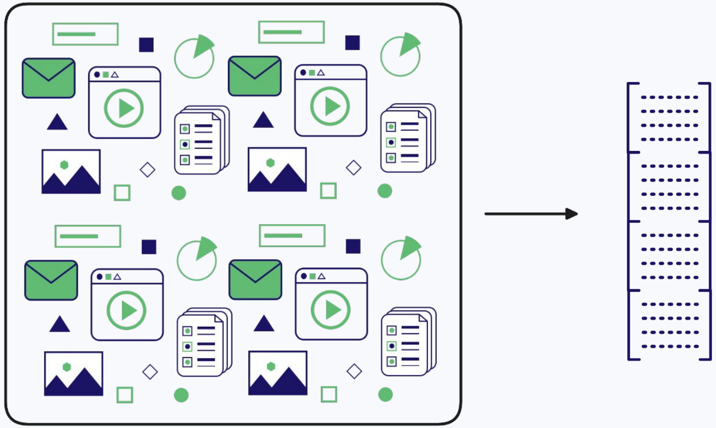
Improve efficiency



Scale: Solutions

Improve efficiency

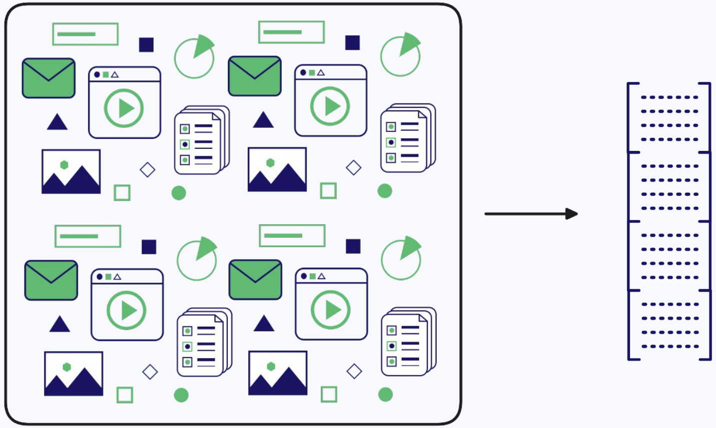
Compression



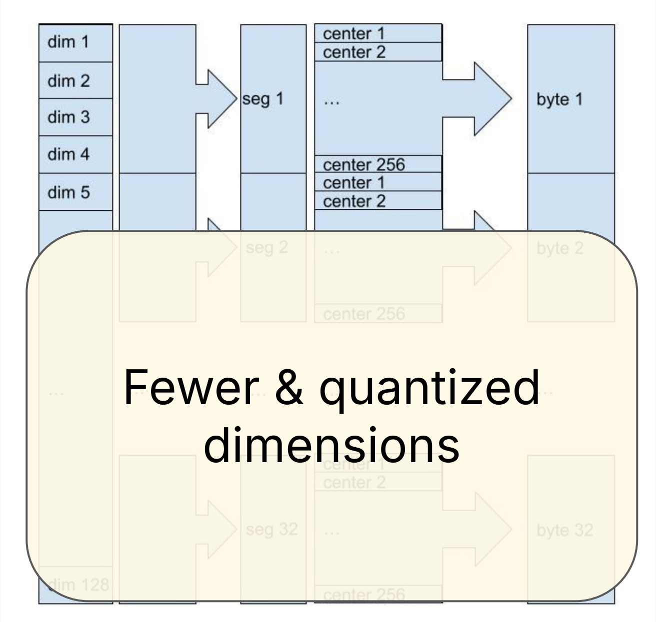
Scale: Solutions

Improve efficiency

Compression

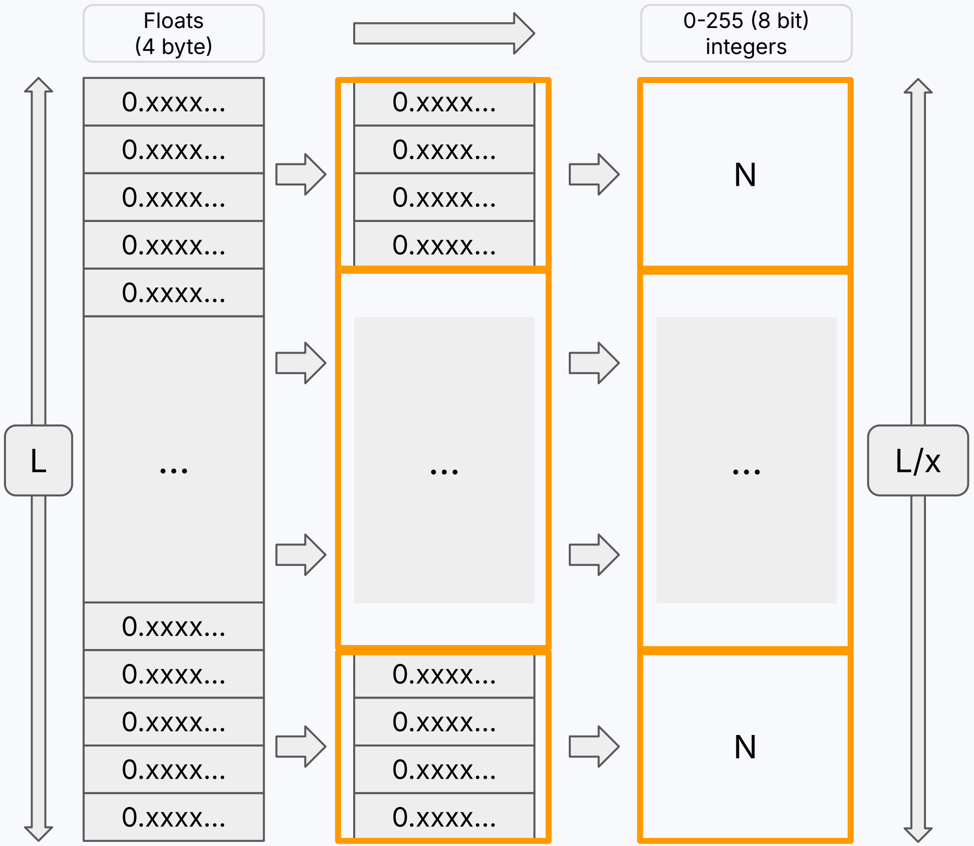


Product quantization



Customisable compression

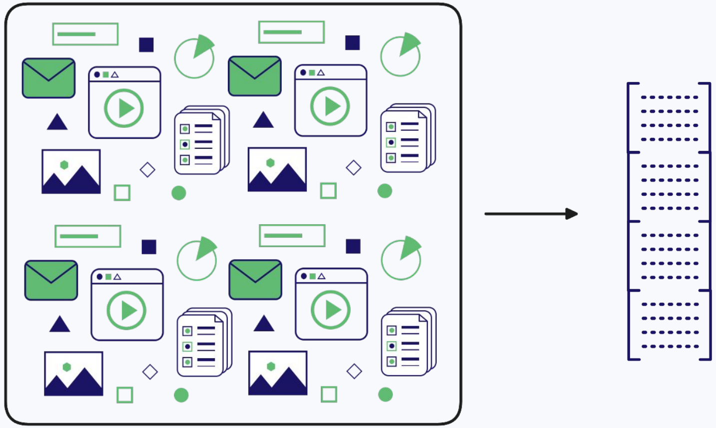
(e.g. 128 floats → 32 bytes: 16x)


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

Scale: Solutions

Improve efficiency

Compression



Binary quantization

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

↓

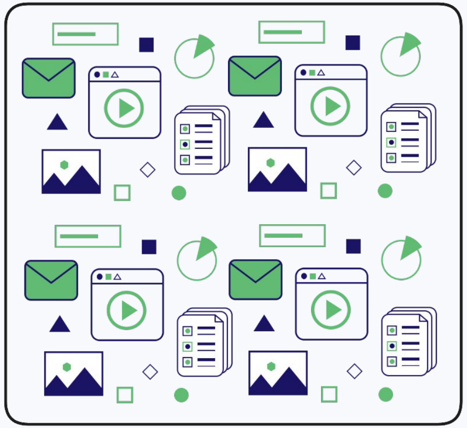
[0, 0, 1, 0, 1, 1,
...
0, 1, 0, 1, 1, 0]

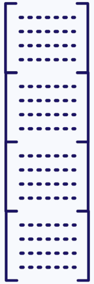
n floats → n bits
(32x reduction)

Scale: Solutions

Improve efficiency

Compression





Scalar quantization

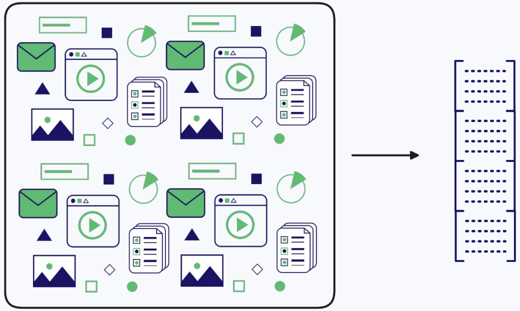
[-0.1324..., -0.9253...,
0.2389...,
...
0.3249..., 0.2390..., -0.4823...]

↓

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

n floats → n ints
(4x reduction)

Scale: Solutions



BQ / PQ / SQ compression

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

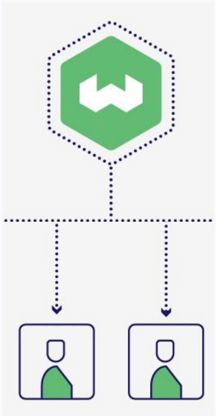
When to use which?

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

Multi-tenancy

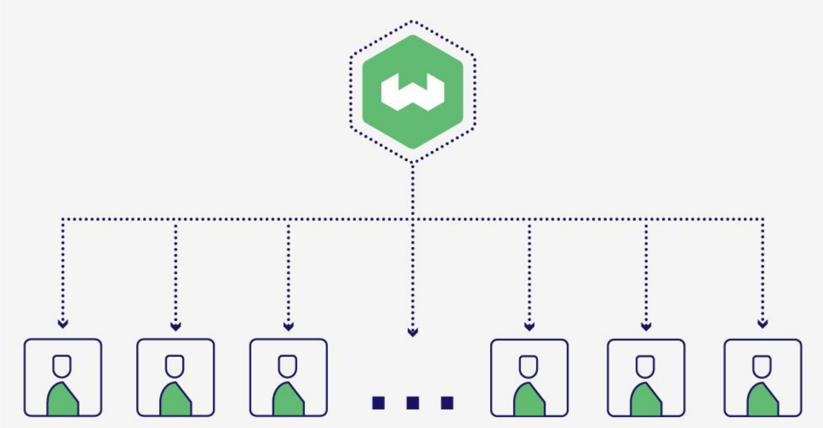
Scale: Solutions

End user growth



Scale: Solutions

End user growth



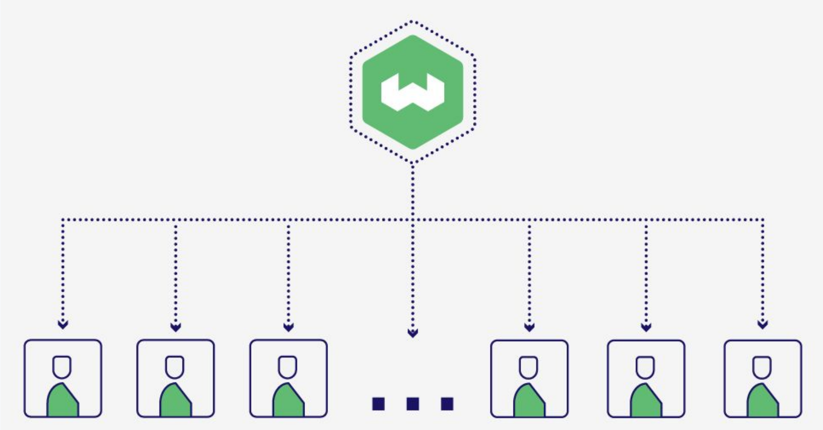
Challenges faced:

- Performance
- Data isolation
- Compliance

Developed: **Multi-tenancy**

Scale: Solutions

End user growth

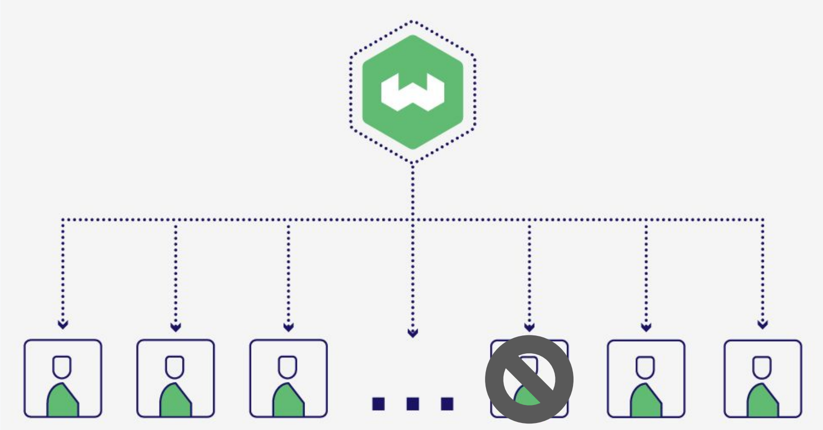


Multi-tenancy implementation

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

Scale: Solutions

End user growth

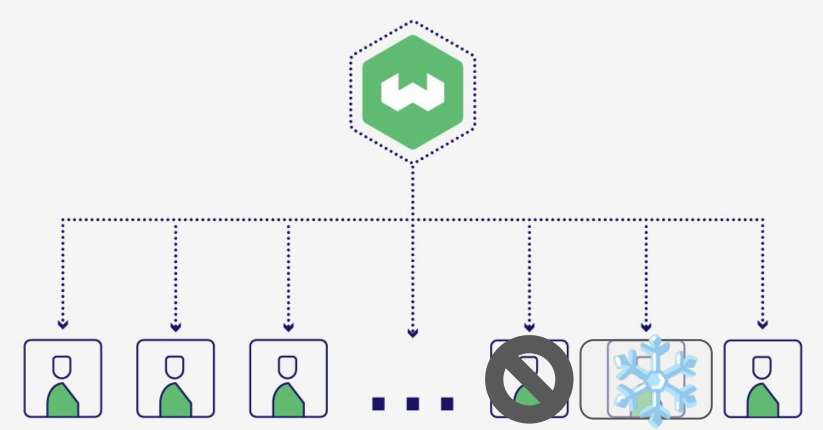


Multi-tenancy implementation

- 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)

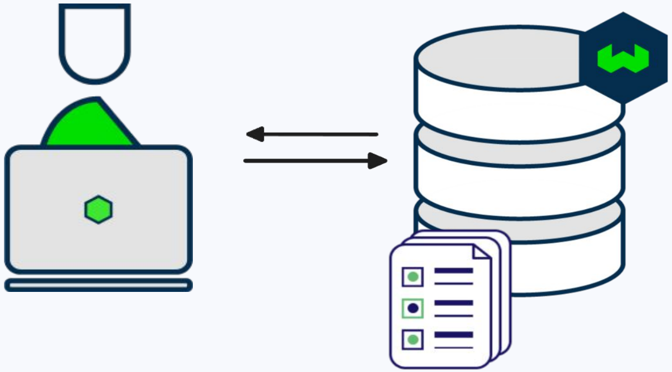
Replication

Reliability: Considerations

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

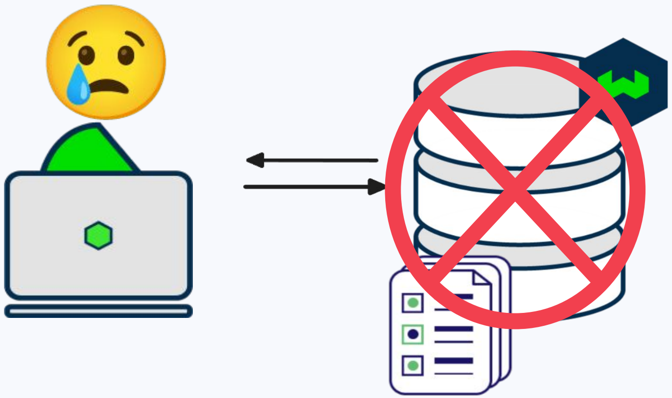
Reliability: Solutions

Happy days



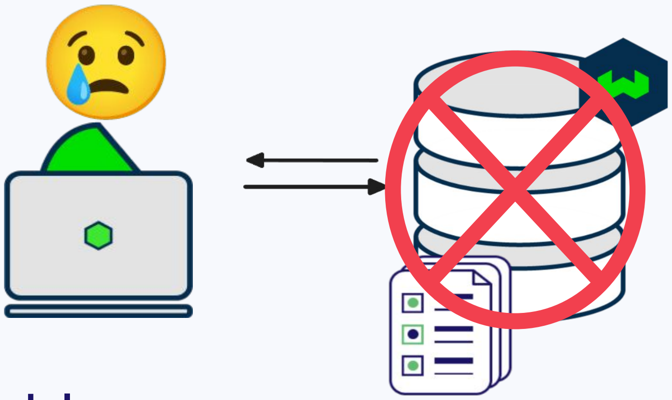
Reliability: Solutions

(Less) Happy days



Reliability: Solutions

(Less) Happy days

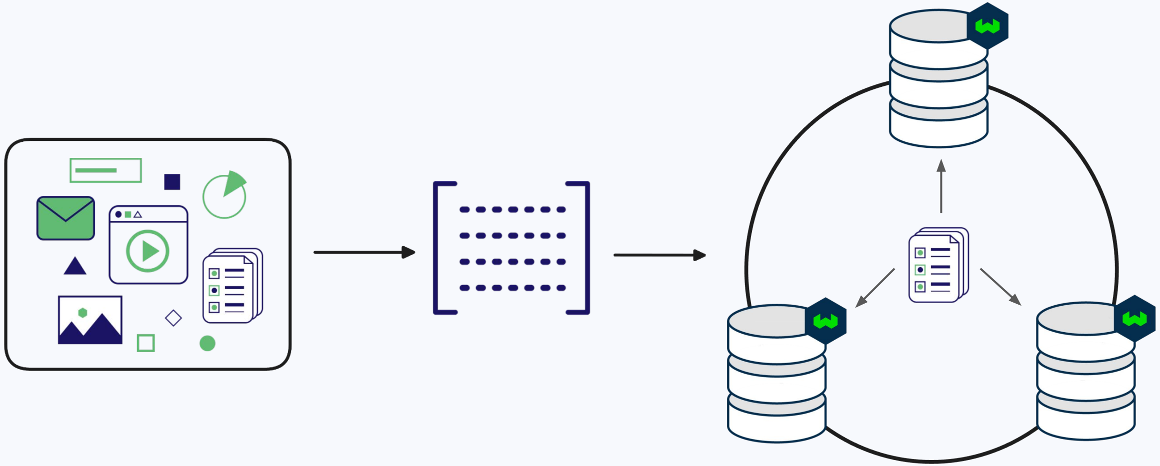


Downtime = inevitable

Reliability: Solutions

Provide redundancy

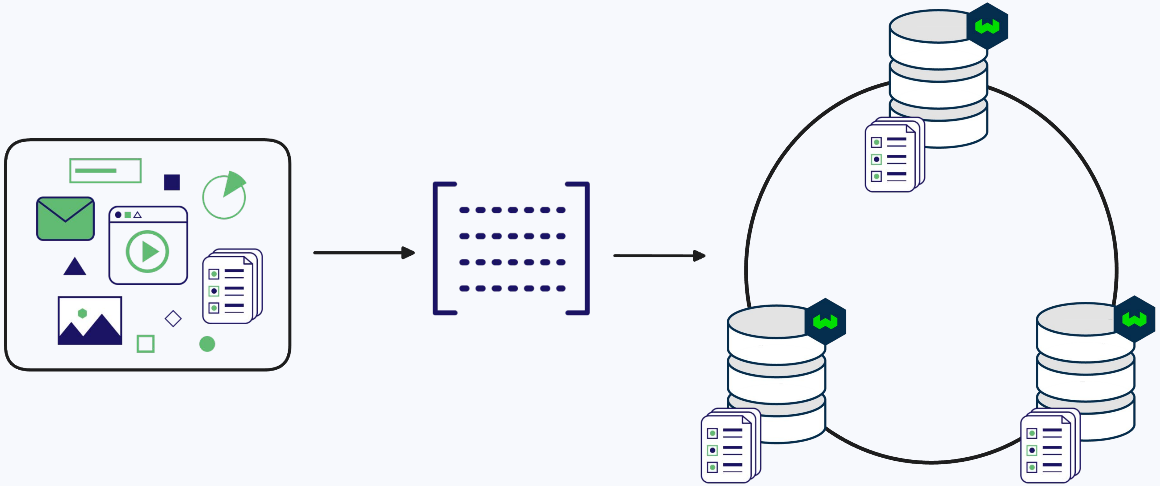
Replication



Reliability: Solutions

Provide redundancy

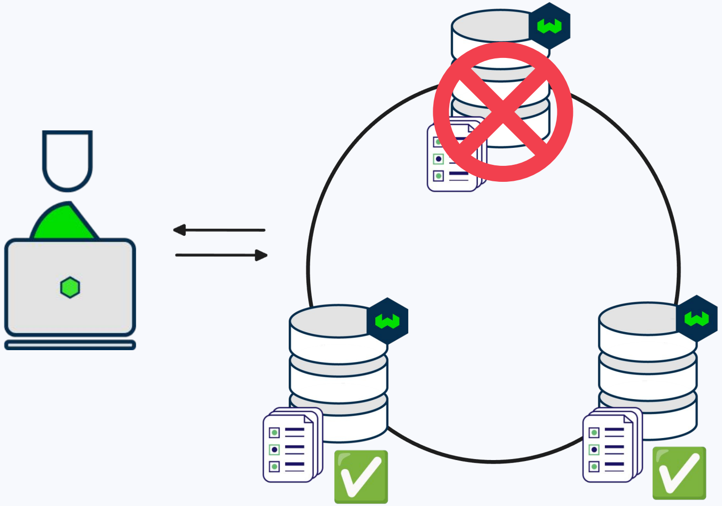
Replication



Reliability: Solutions

Provide redundancy

Replication



Reliability: Solutions

Provide redundancy

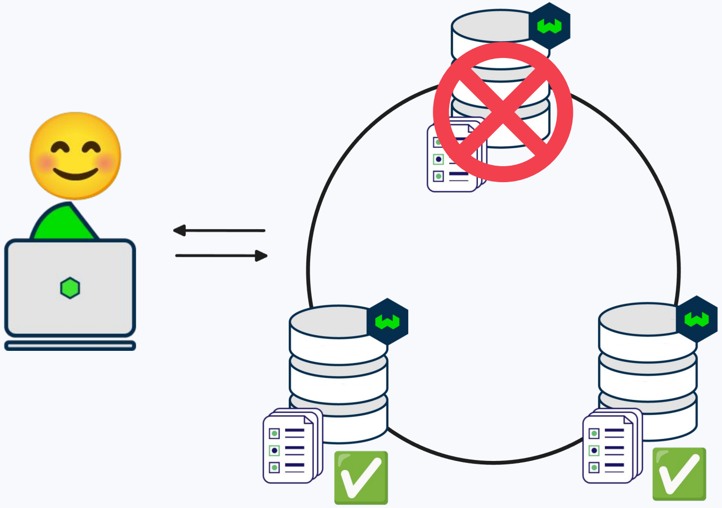
Node-level downtime:

- inevitable

System-level downtime:

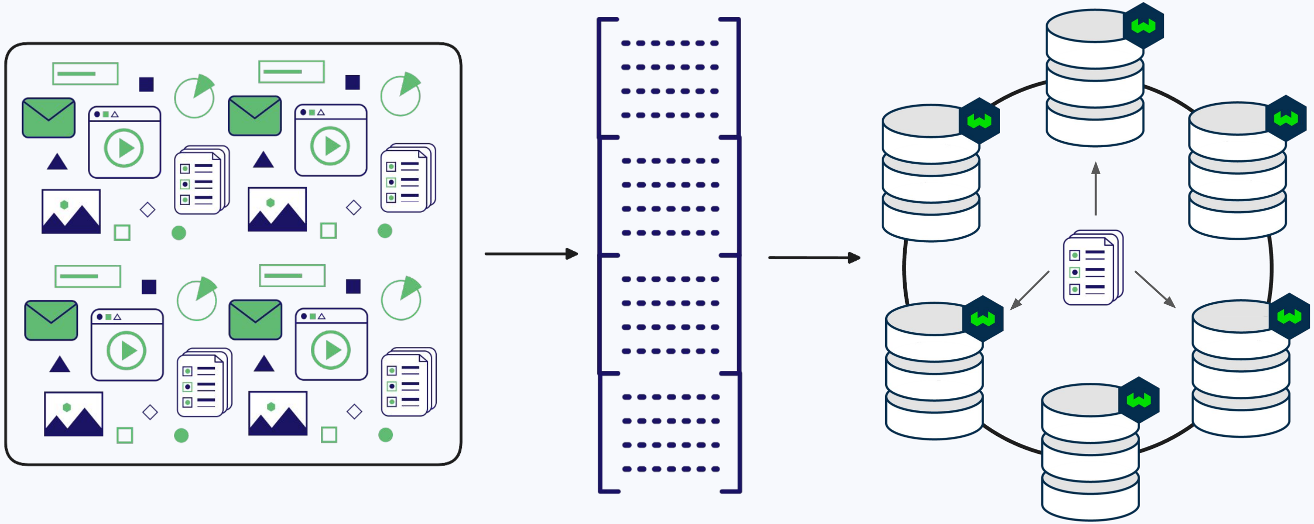
- avoidable!

Replication



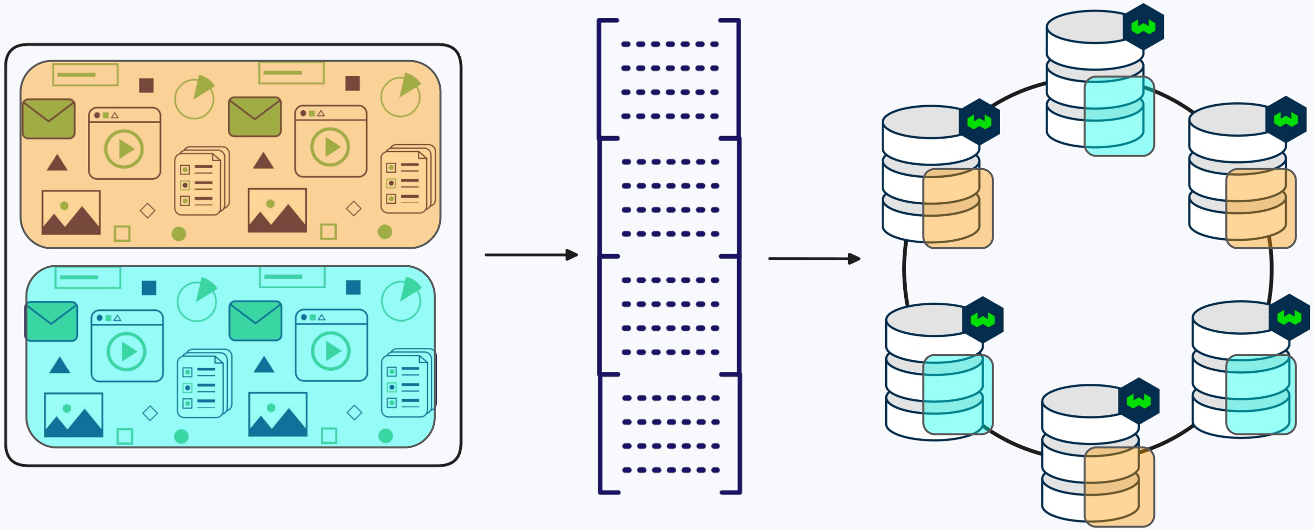
Reliability: Solutions

Sharding + replication



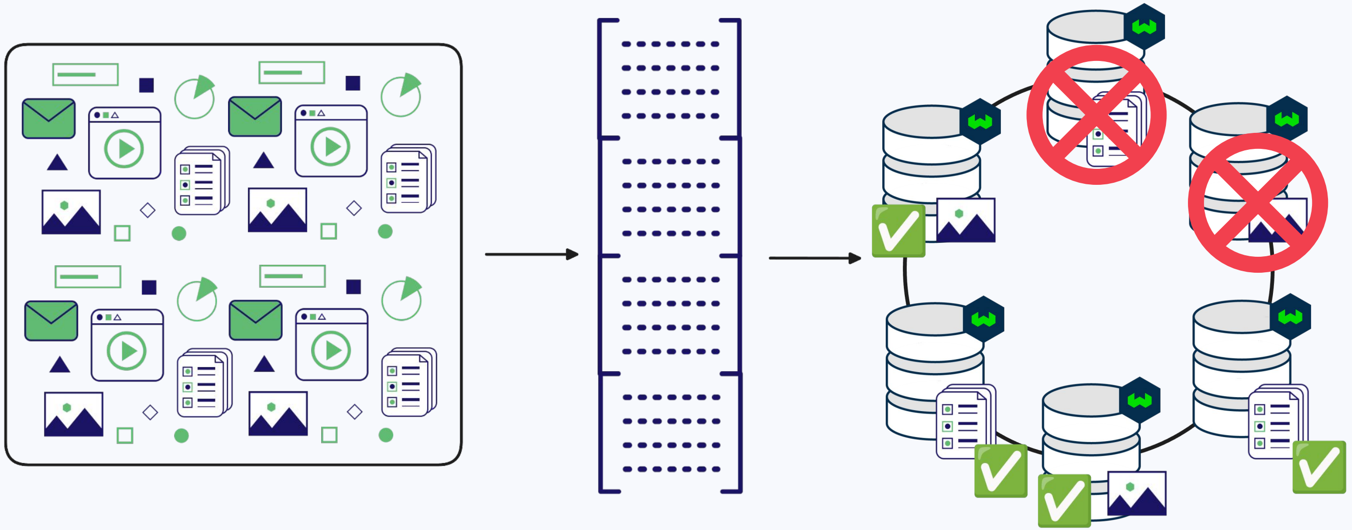
Reliability: Solutions

Sharding + replication



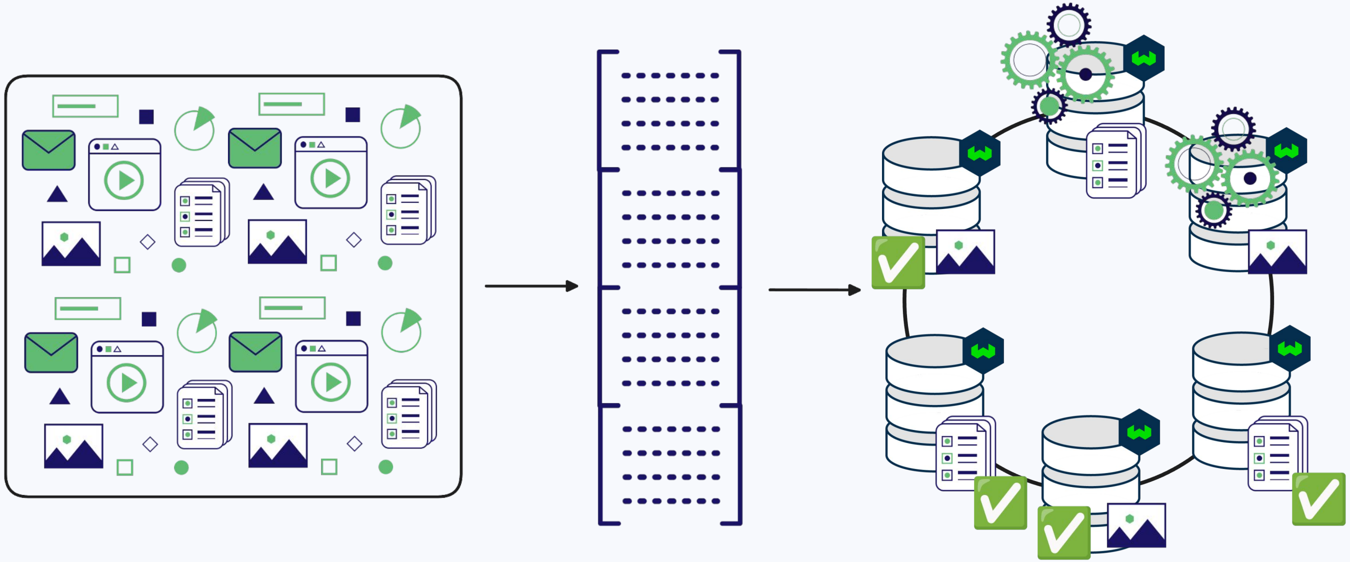
Upgrade: Solutions

Sharding + replication



Upgrade: Solutions

Sharding + replication



Upgrade: Solutions

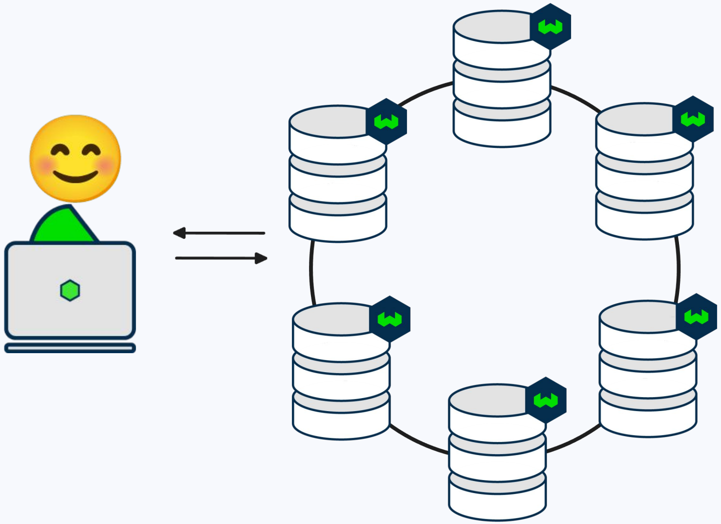
Provide redundancy

“Just works”

for the end user -

with the latest

versions

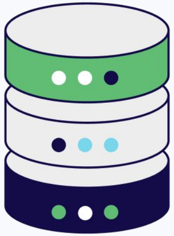


Available Solutions

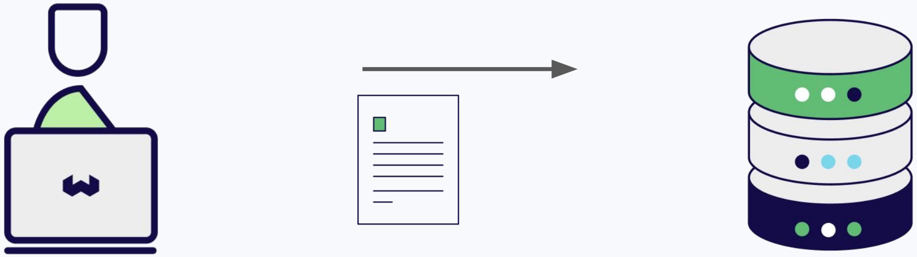
Scaling up	Scaling out
Index options	Quantization
Multi-tenancy (+ tenant states)	Replication

Developer workflow

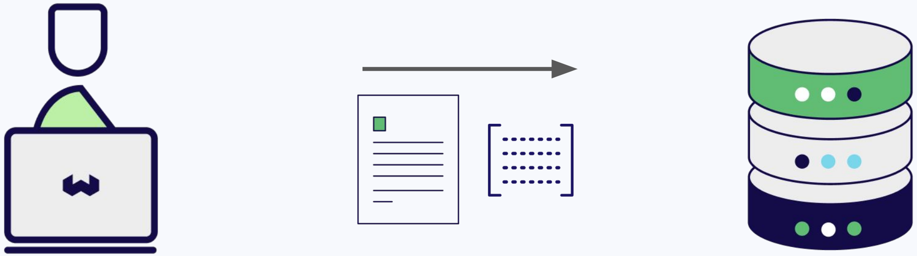
Ingest data



Ingest data



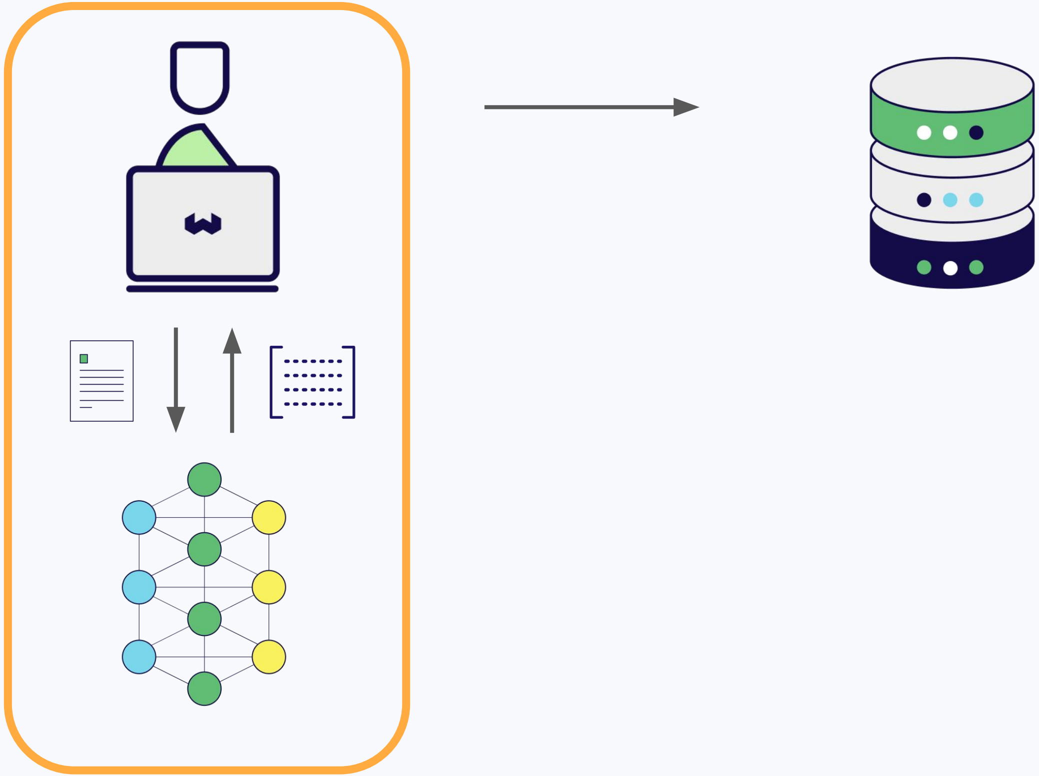
Ingest data



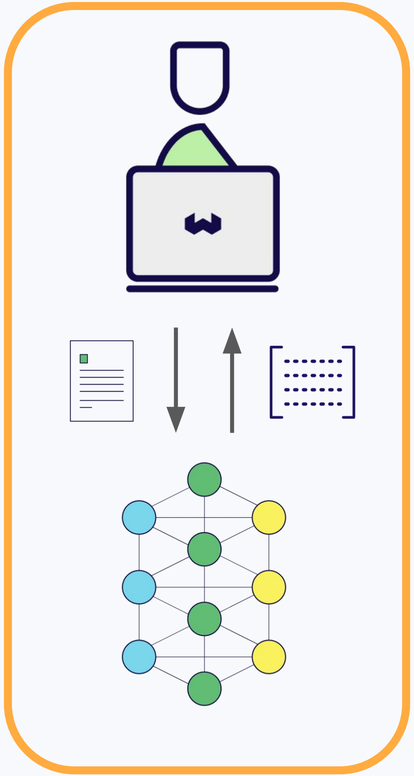
Ingest data

Where do they come from??

Obtain Embeddings

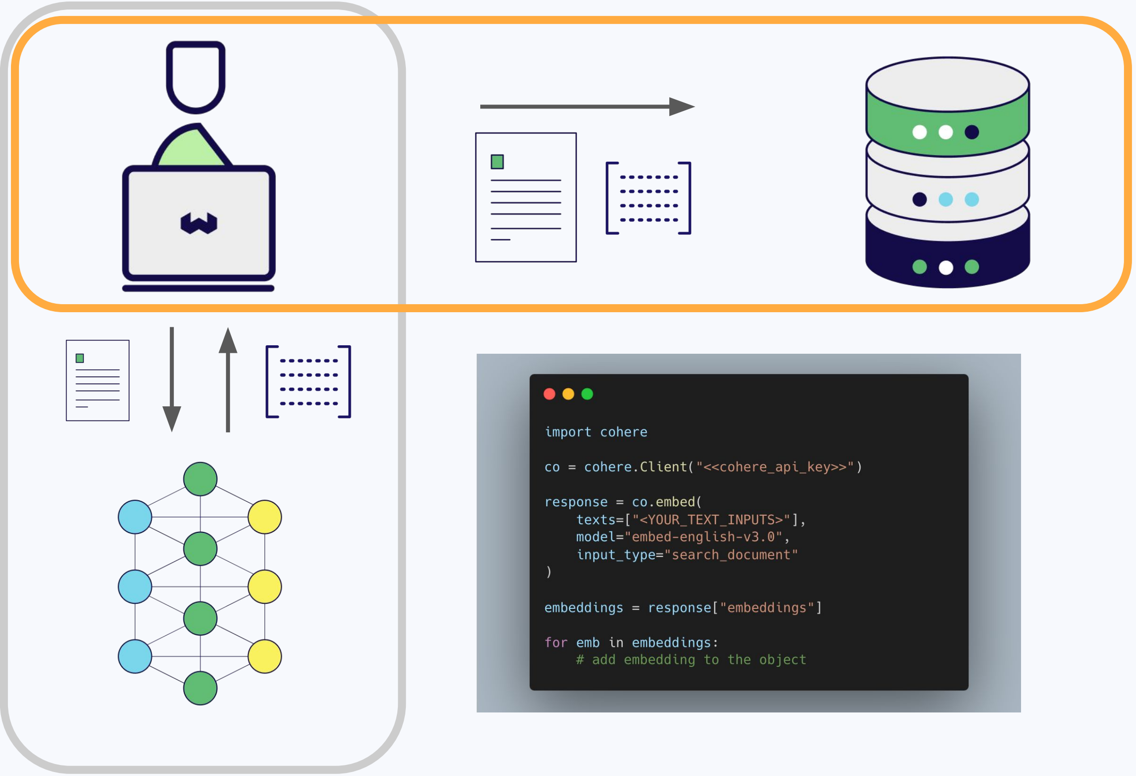


Obtain Embeddings



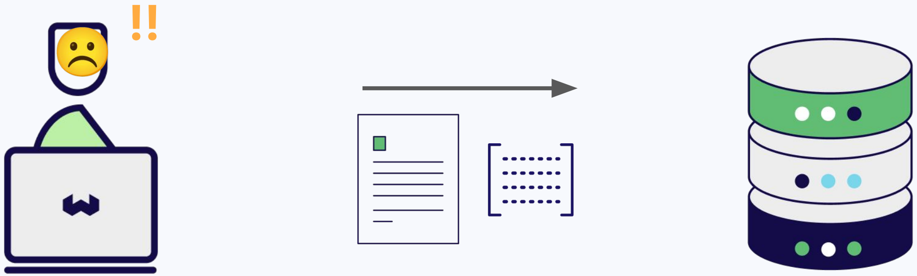
```python
import cohere

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
```

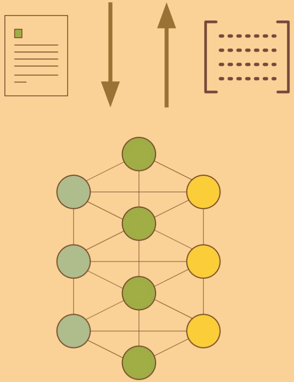
Obtain Embeddings



```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
```

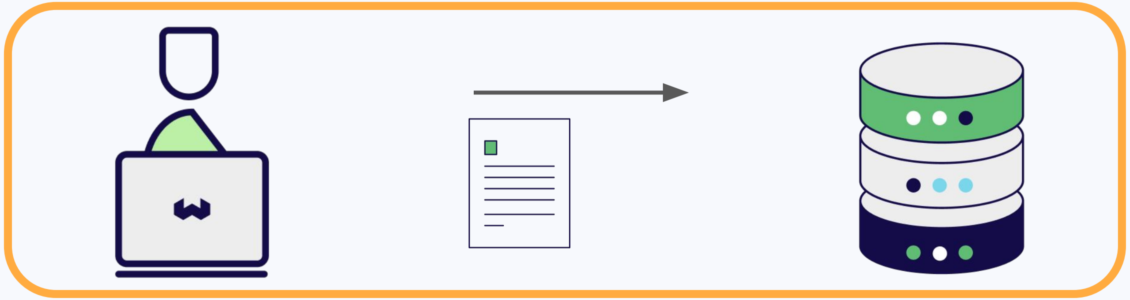
Overhead



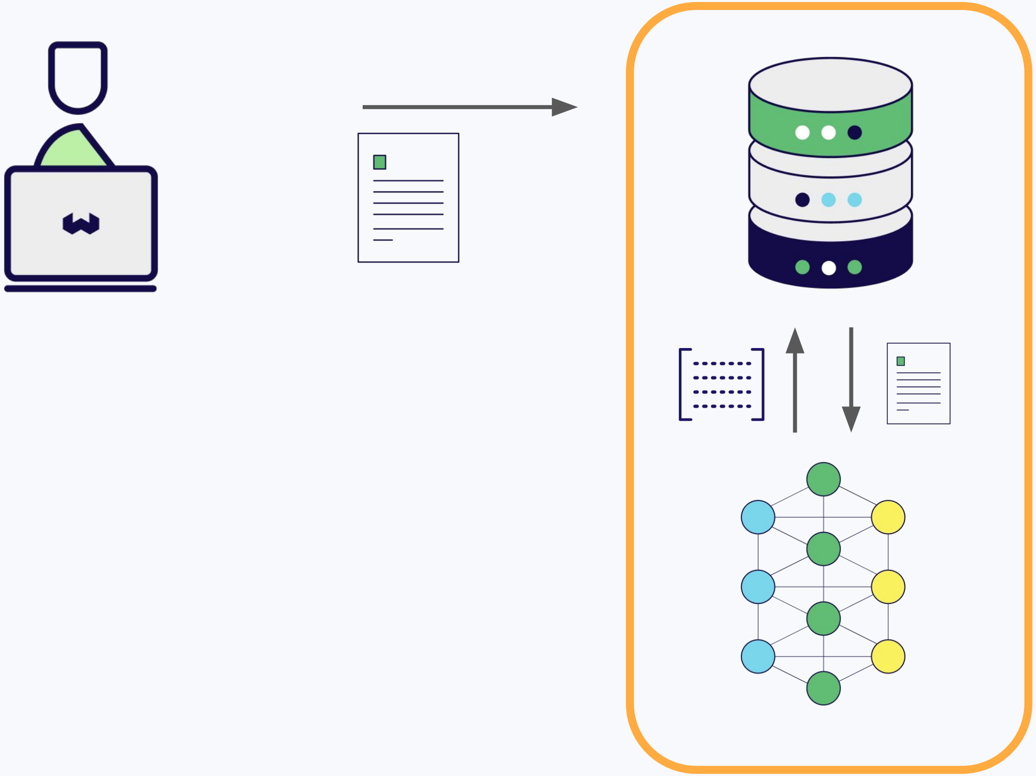


```python
import cohere

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
```

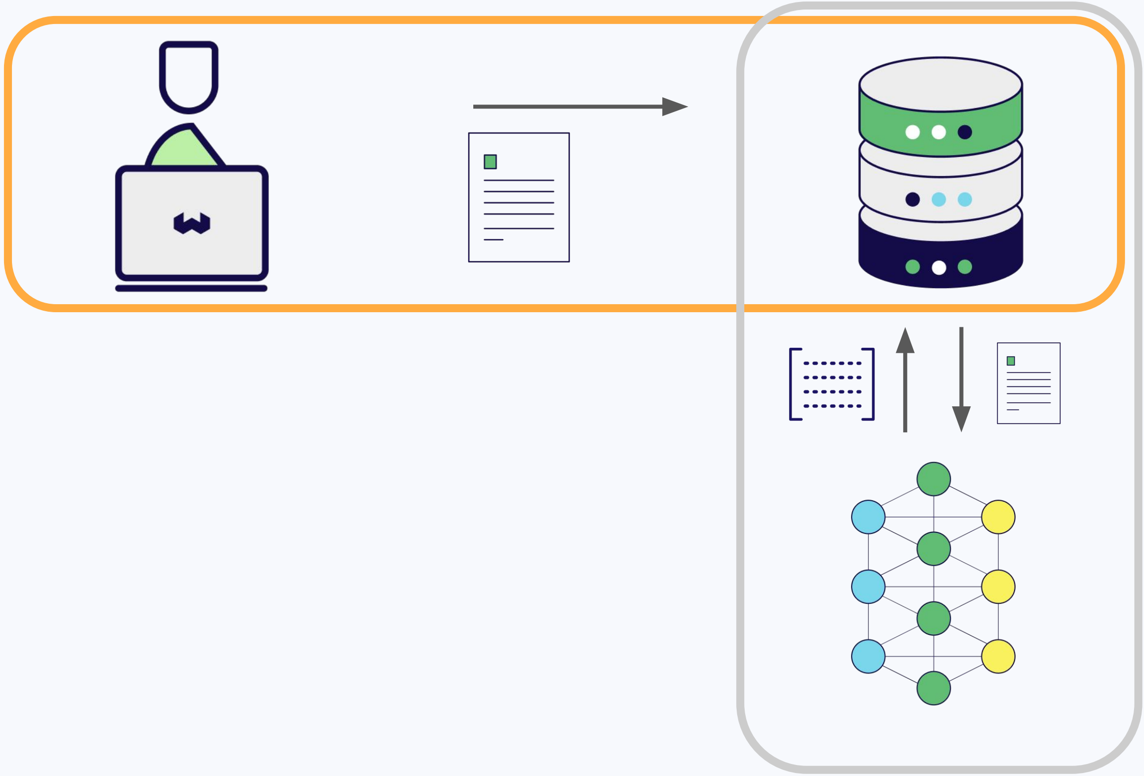
Ingest data



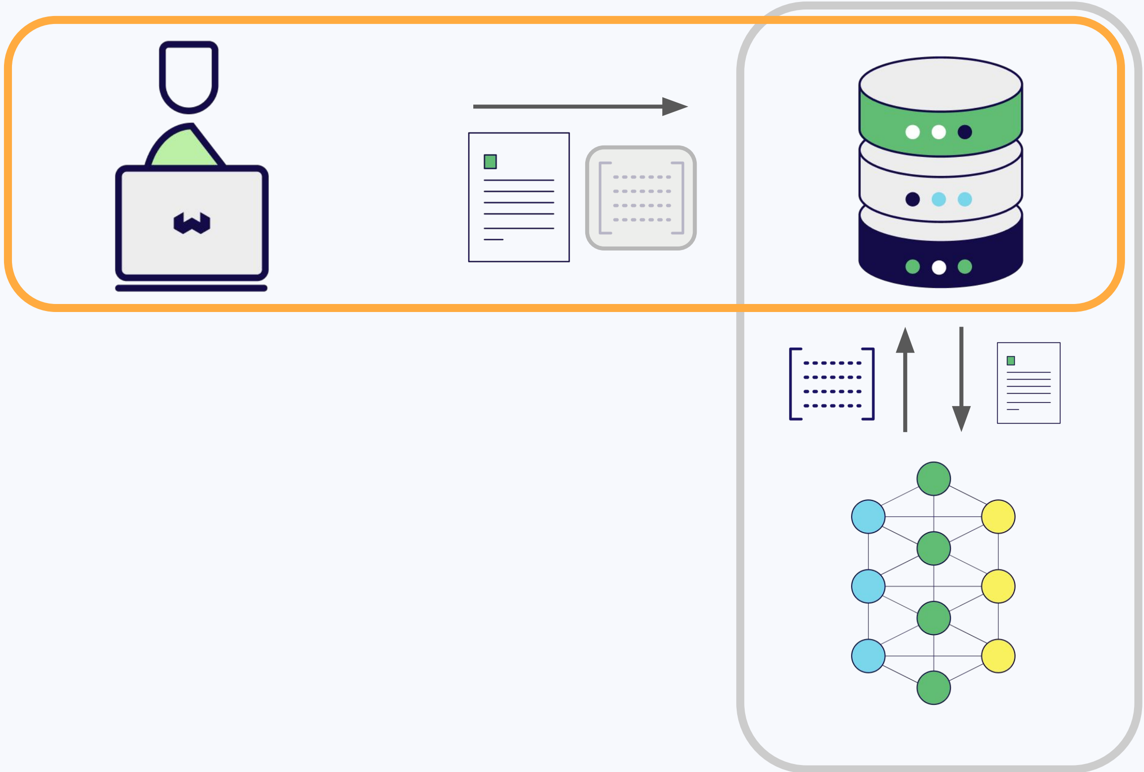
Have Weaviate obtain embeddings



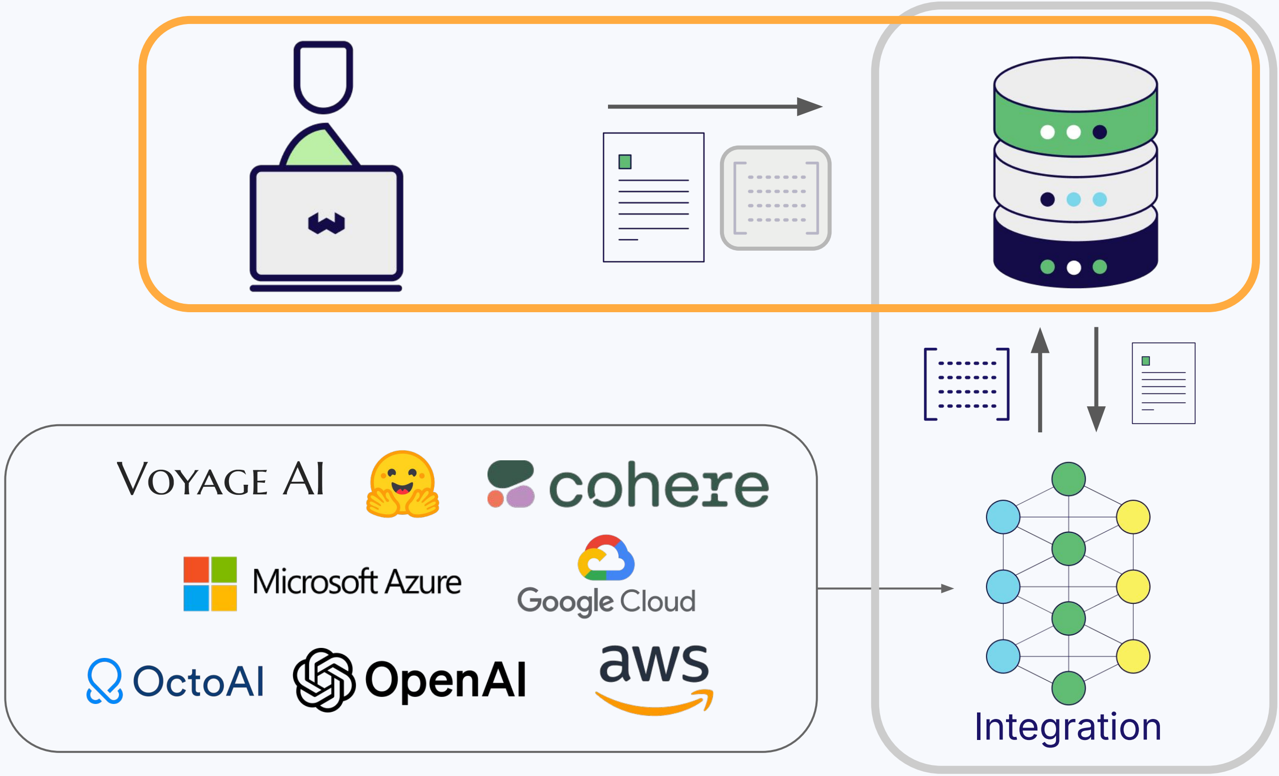
Ingest data (embeddings in background)



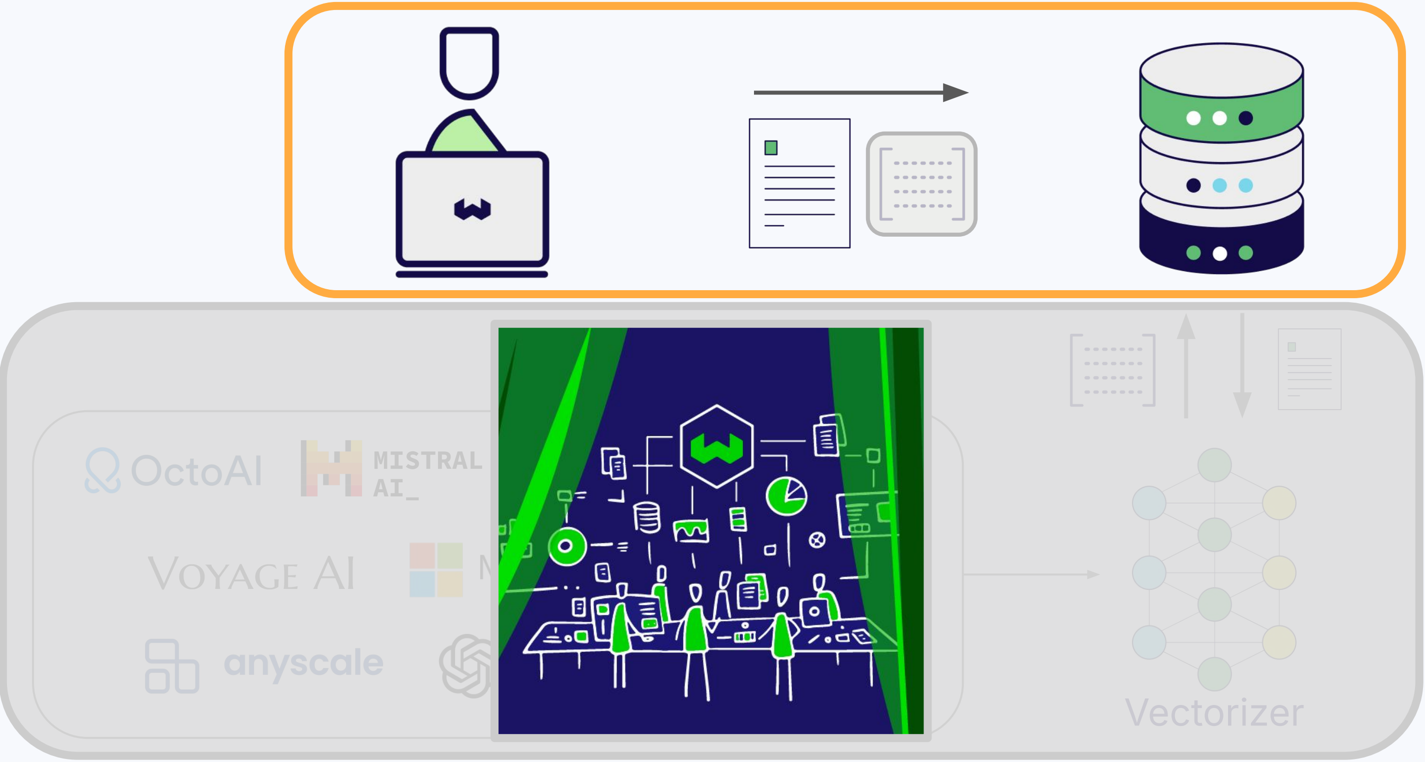
Ingest data (embeddings optional)



Ingest data (with provider integrations)



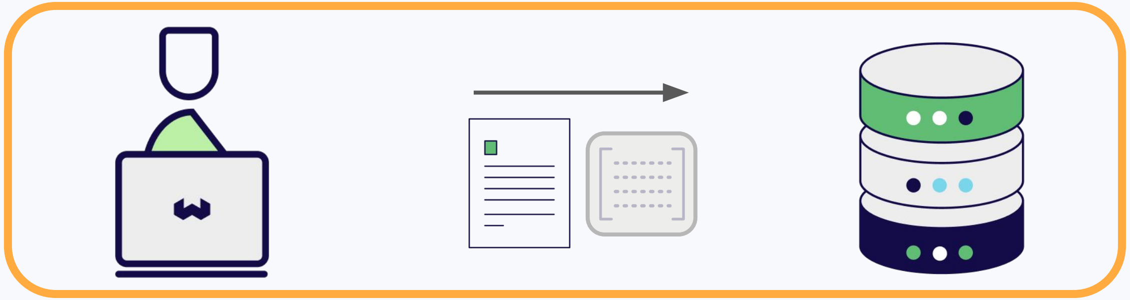
Simplified Experience



User experience

Under the hood

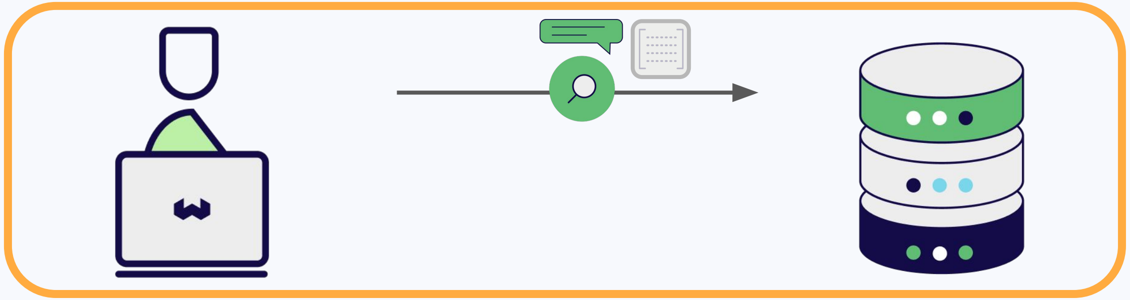
Simplified Experience



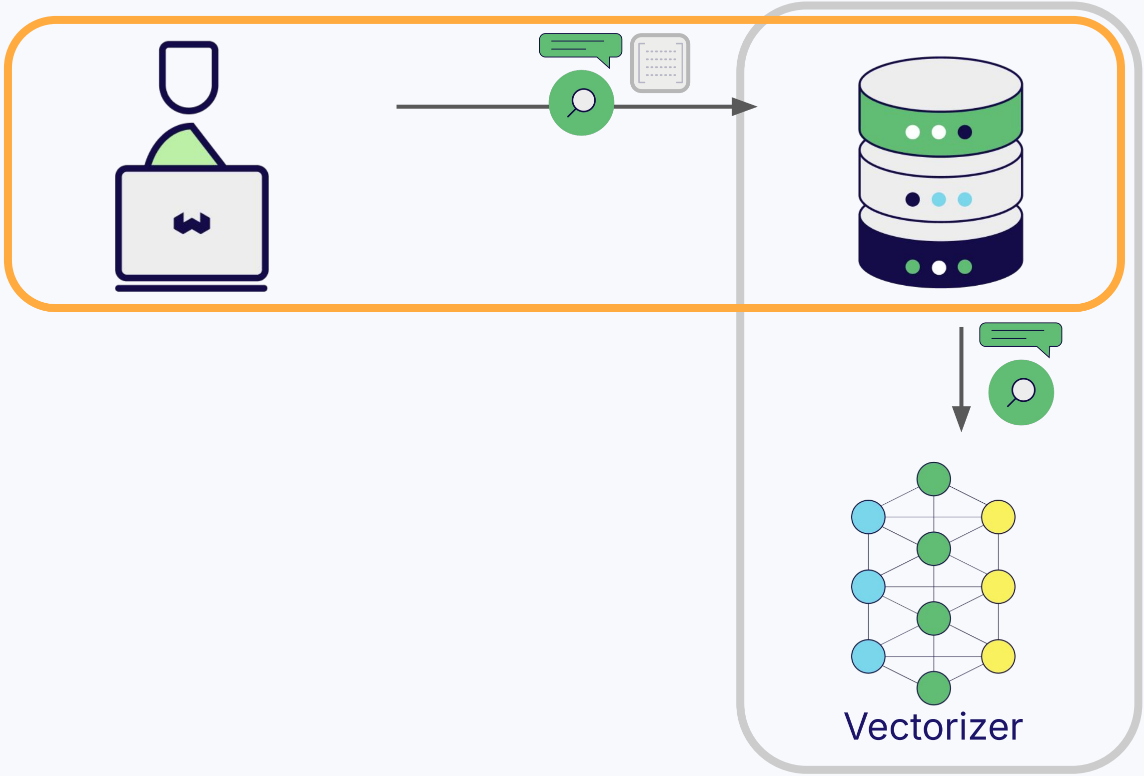
User experience

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

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

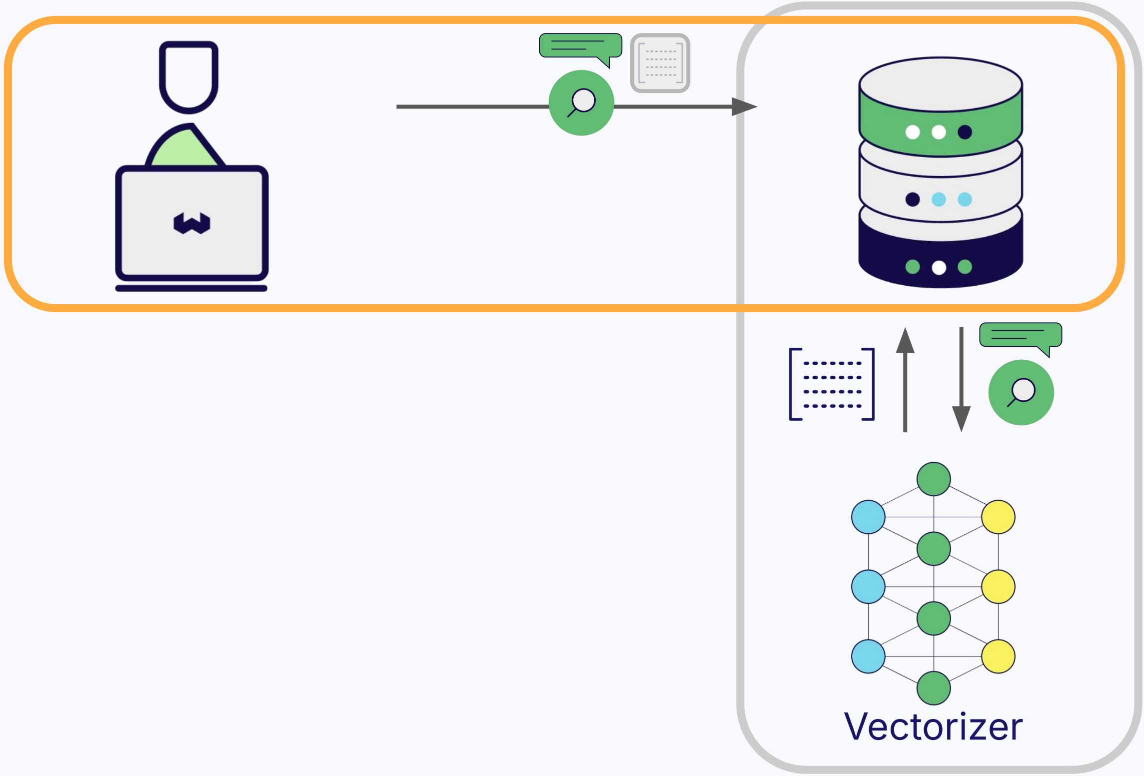
Query



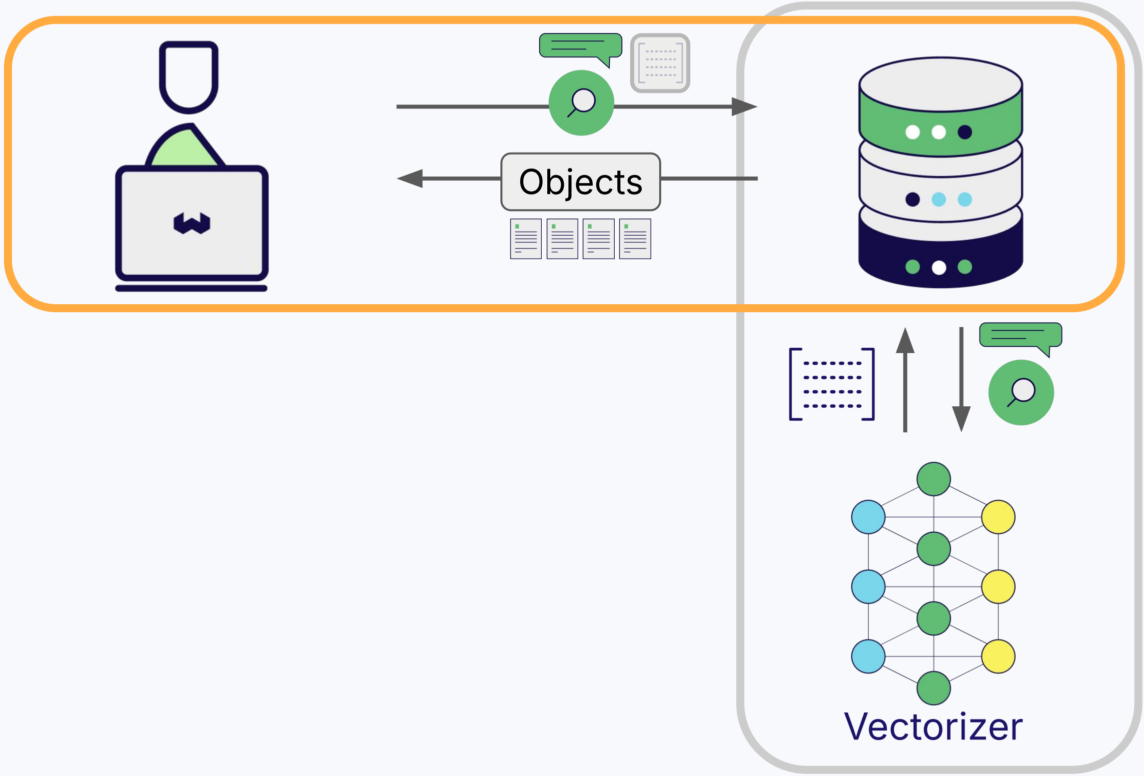
Query



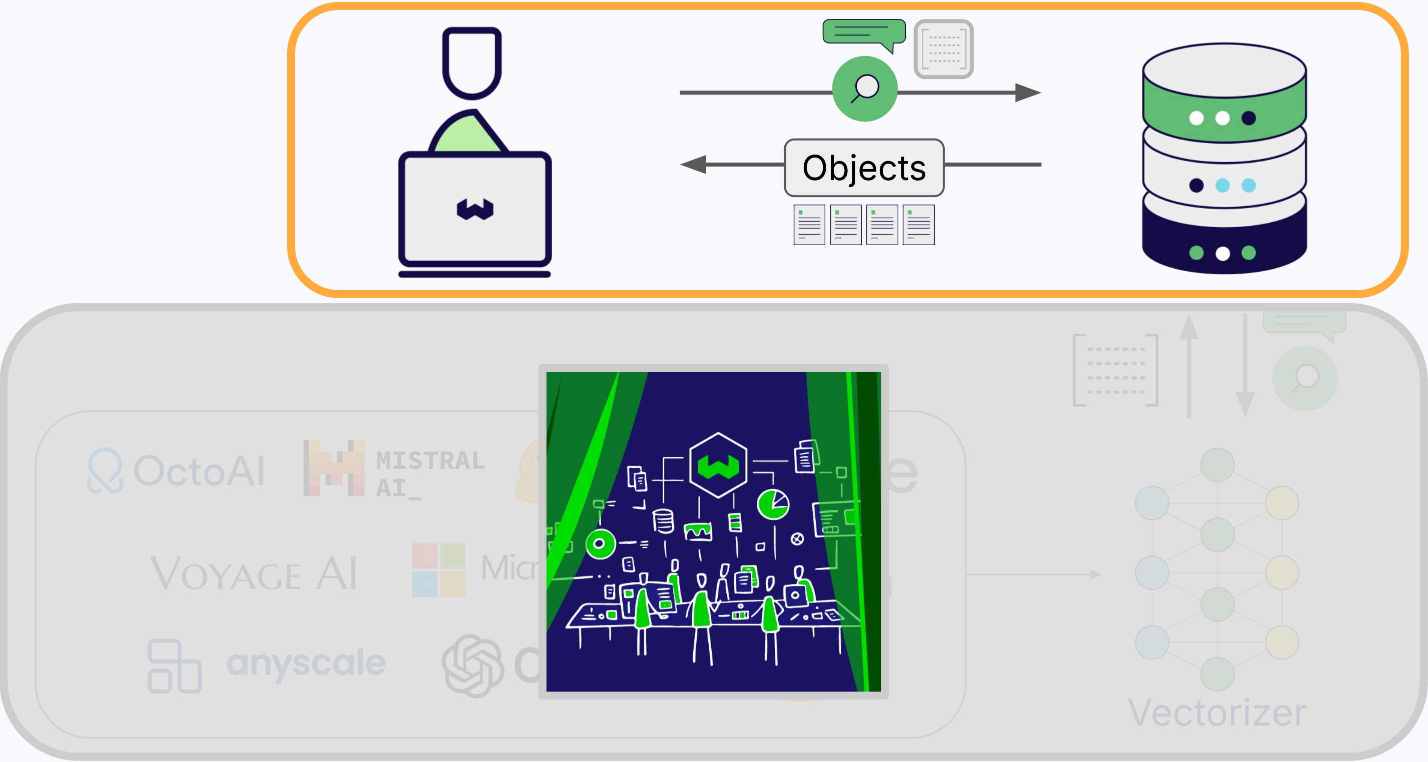
Query



Query



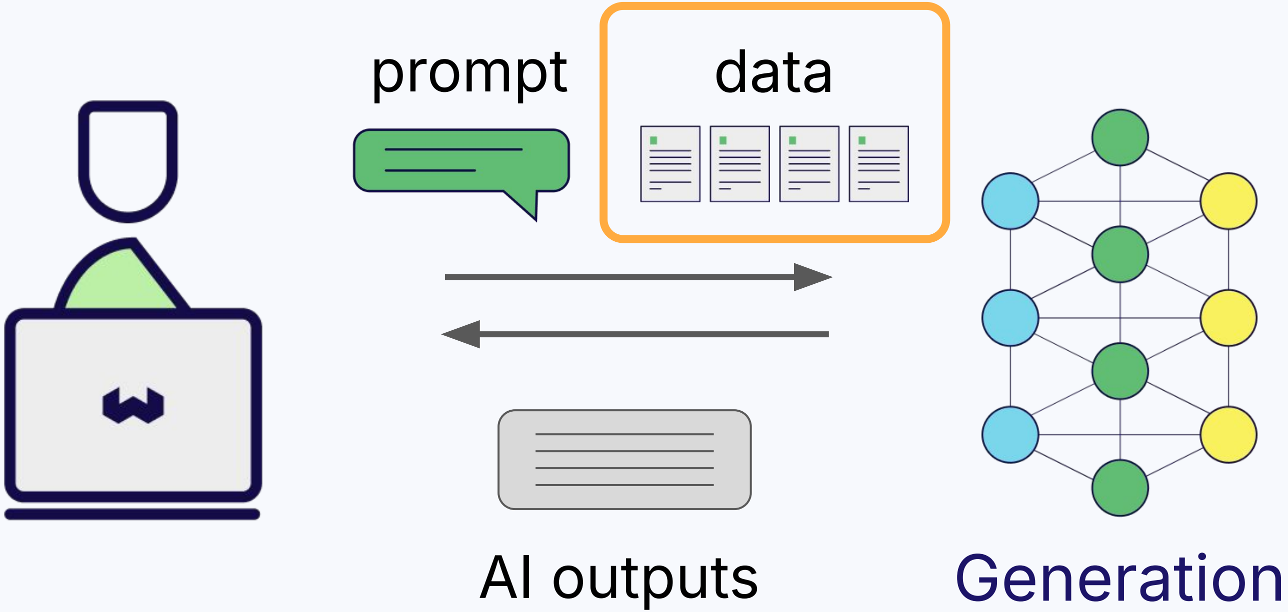
Query



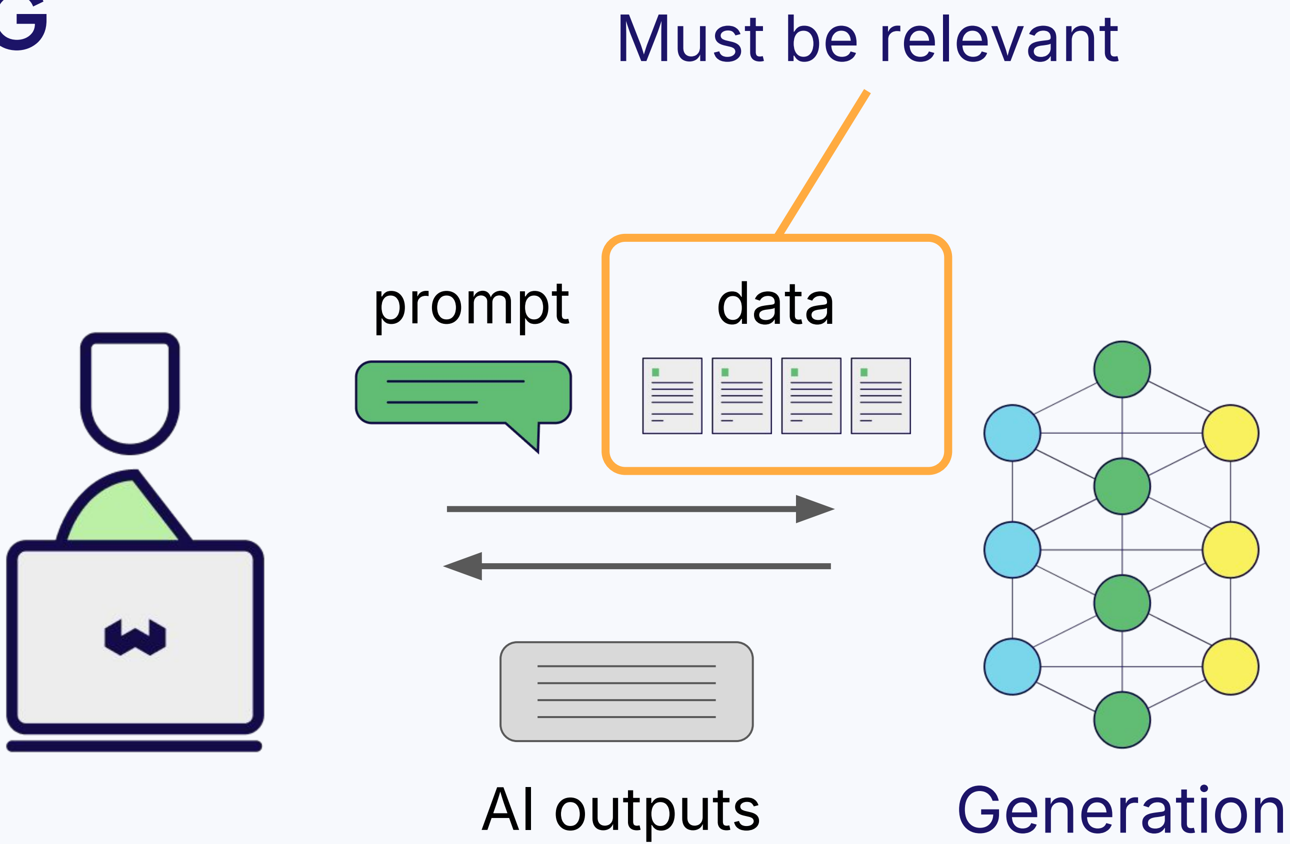
User experience

Under the hood

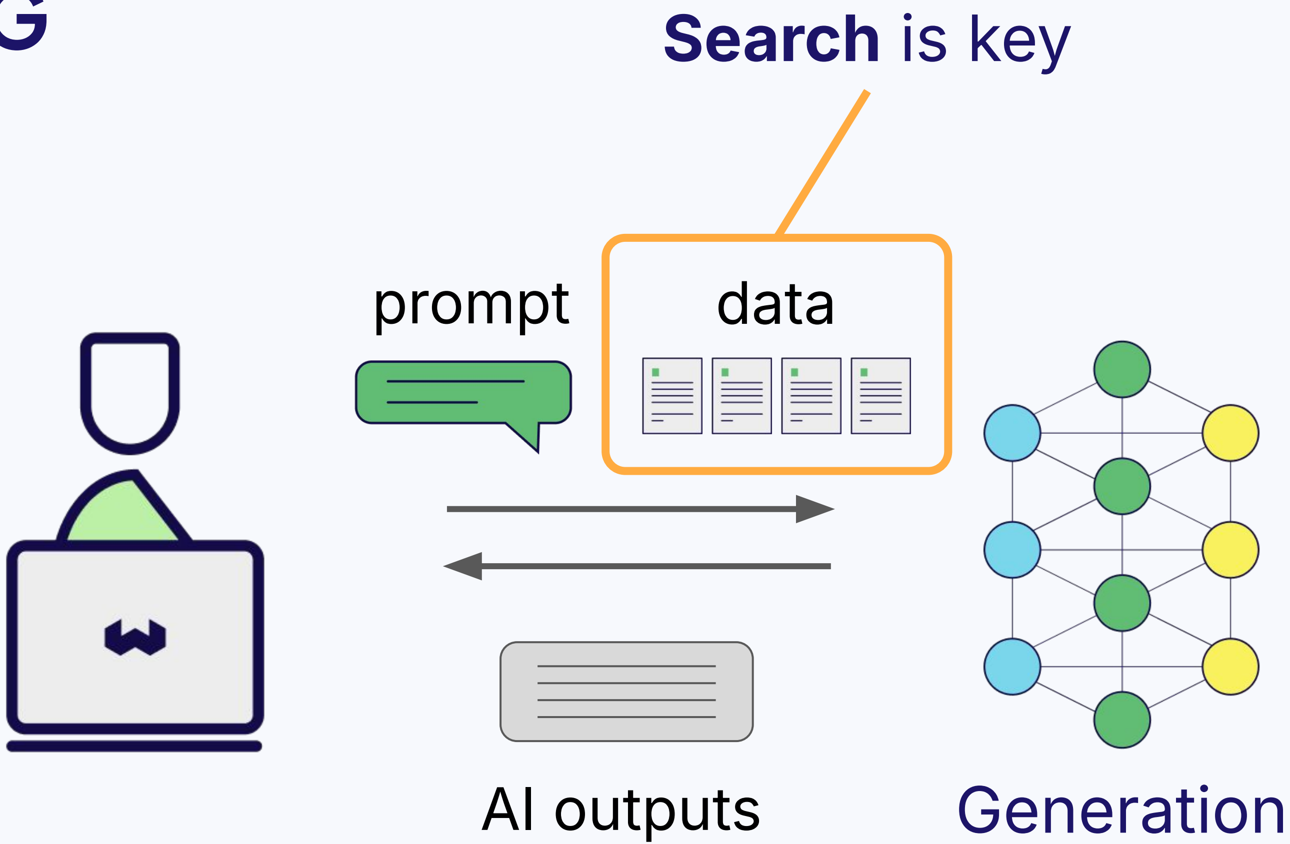
RAG



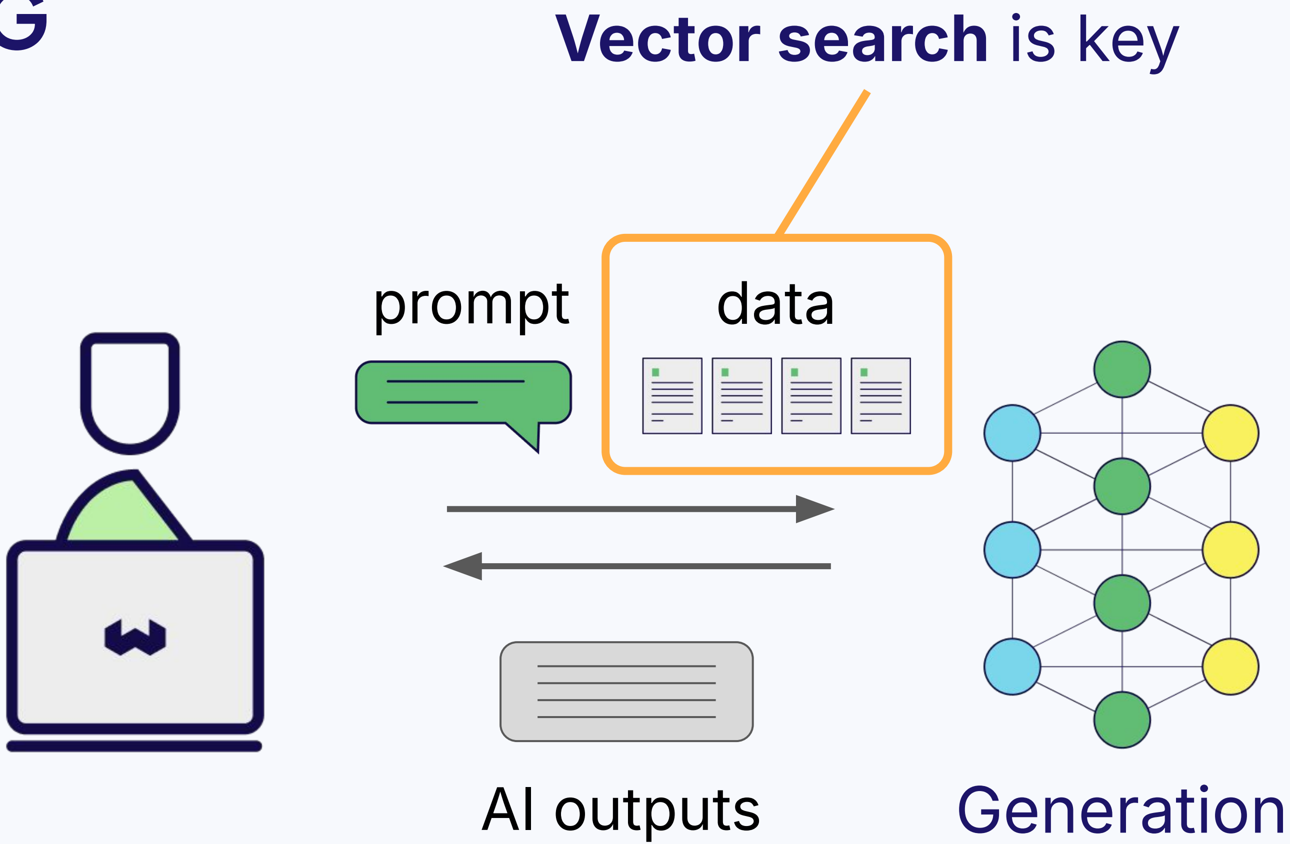
RAG



RAG

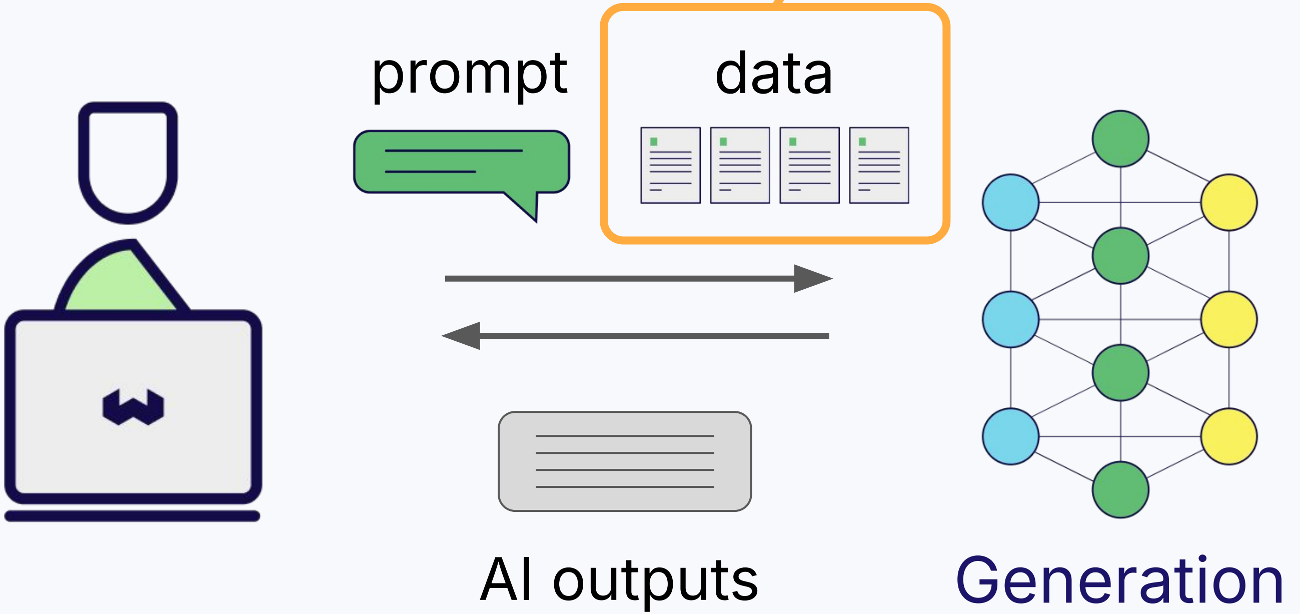


RAG



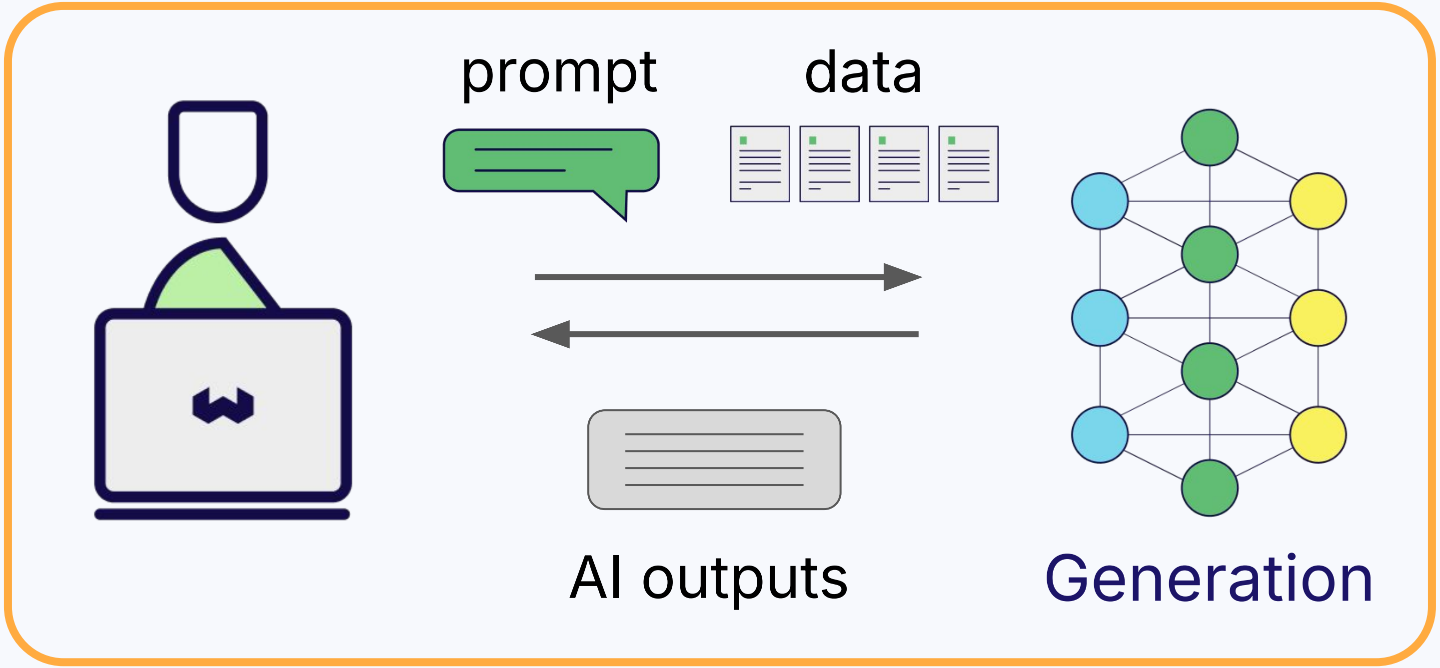
RAG

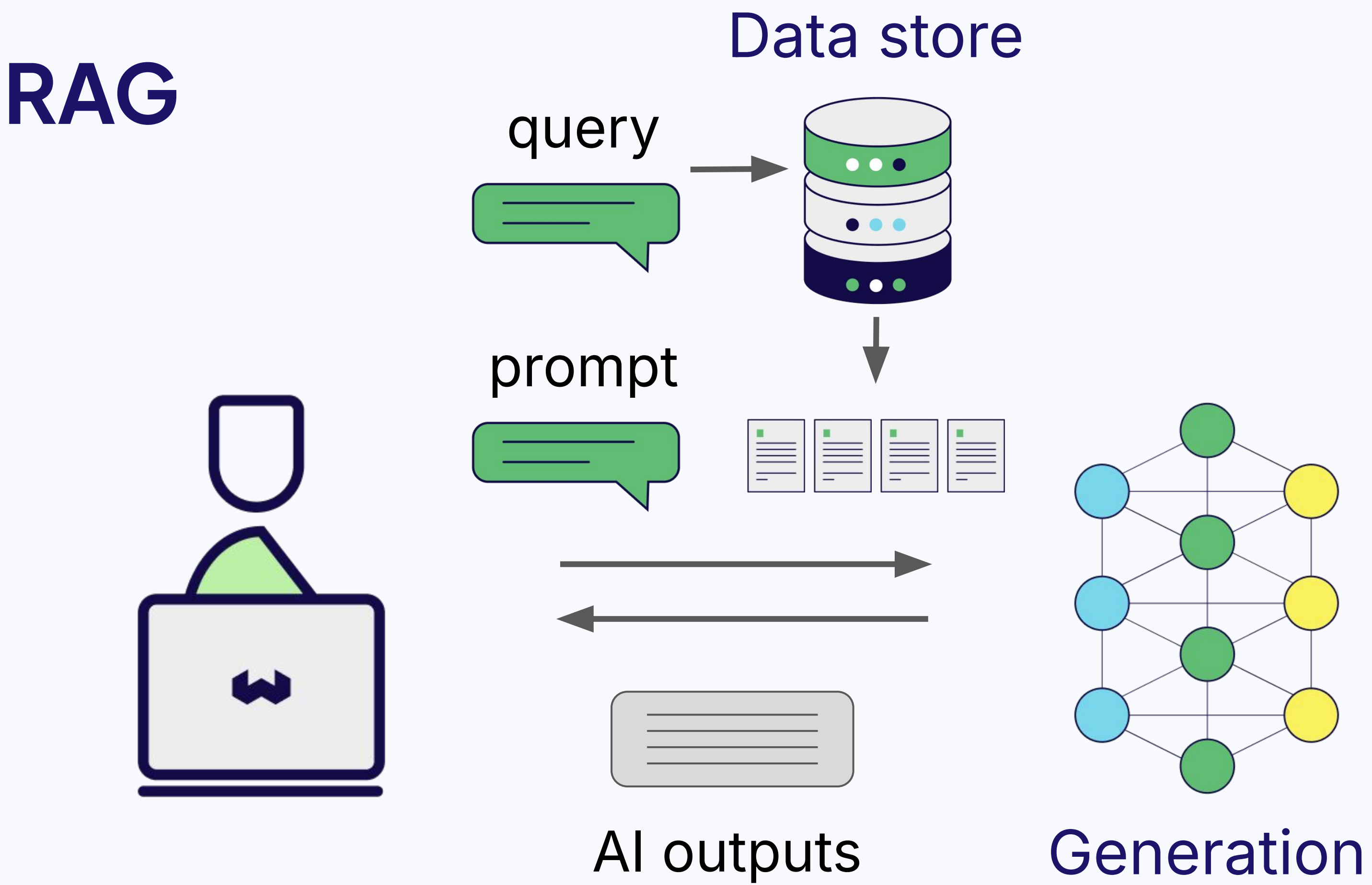
Vector search: relevance search



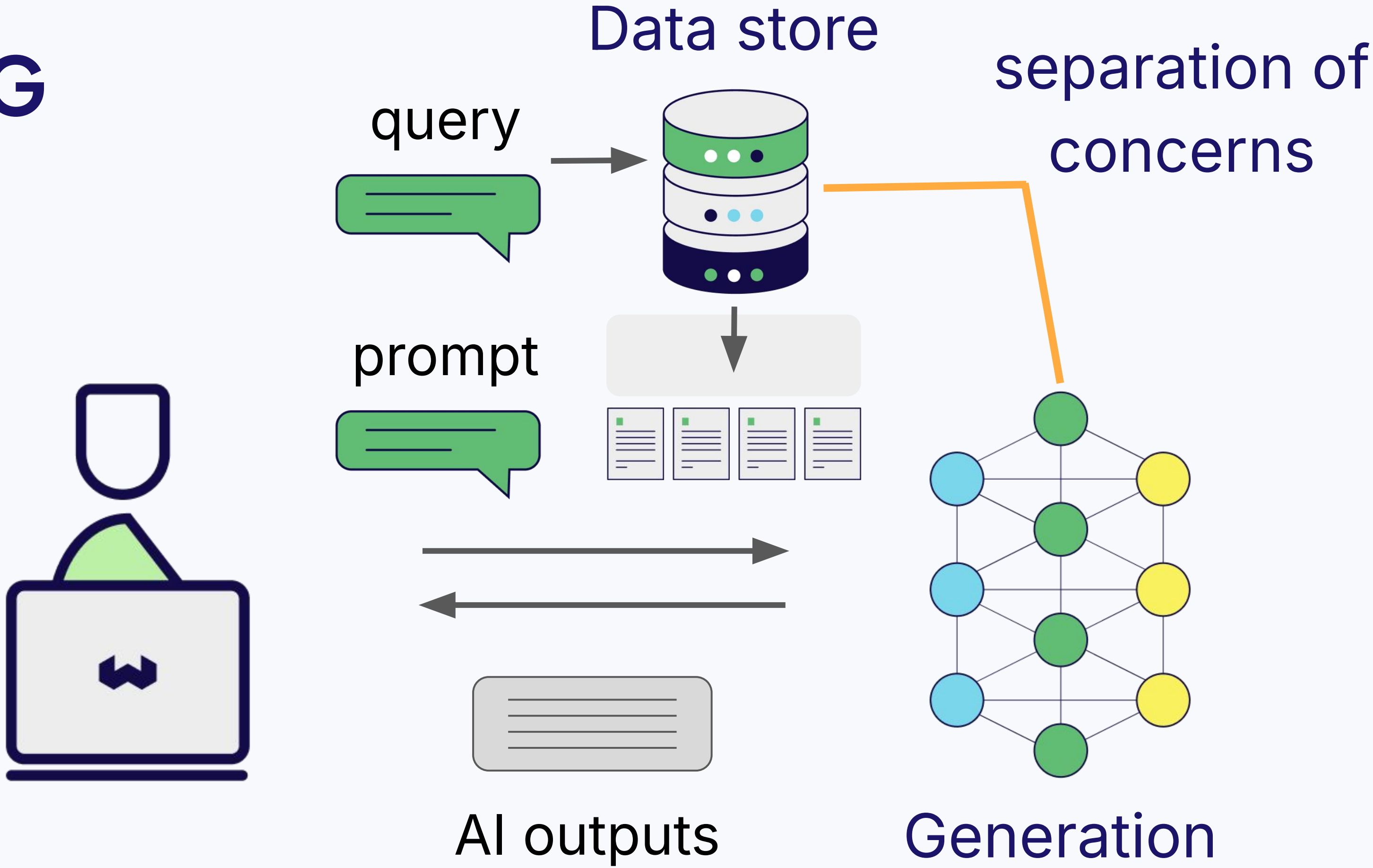
RAG

Why **vector search** is hot



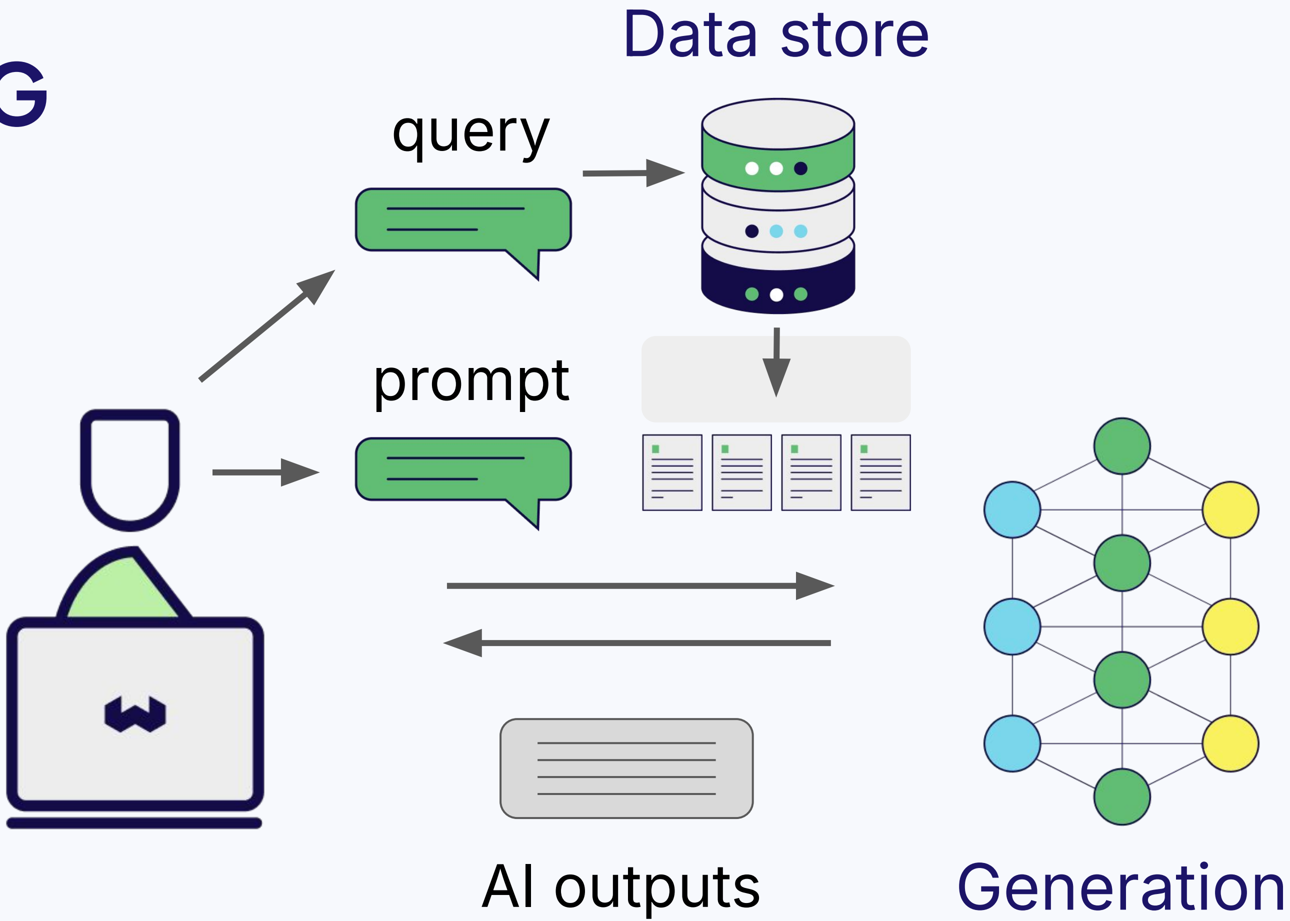

RAG
Data store
query
prompt
AI outputs
Generation

RAG

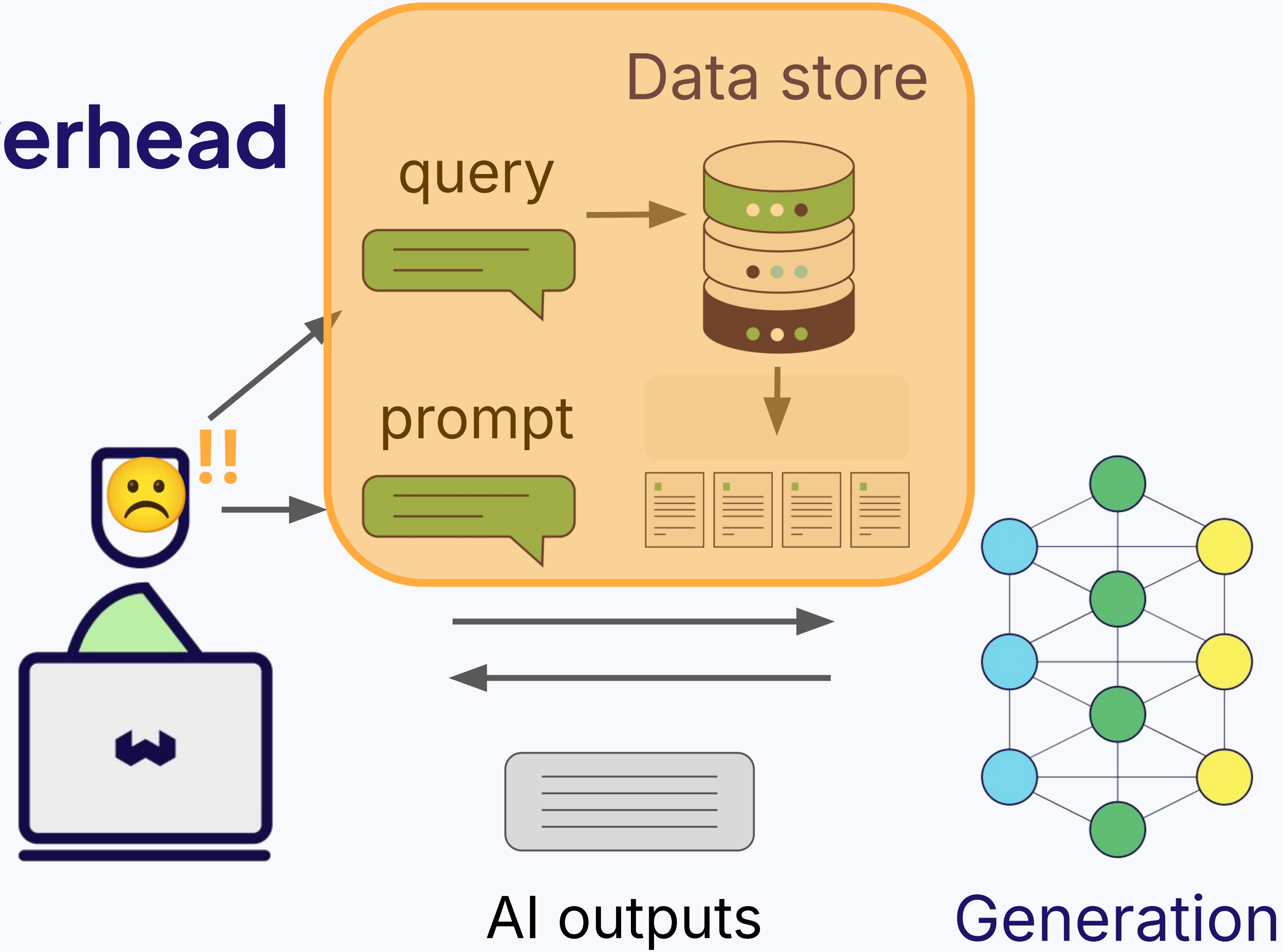


Data store
separation of concerns
query
prompt
AI outputs
Generation

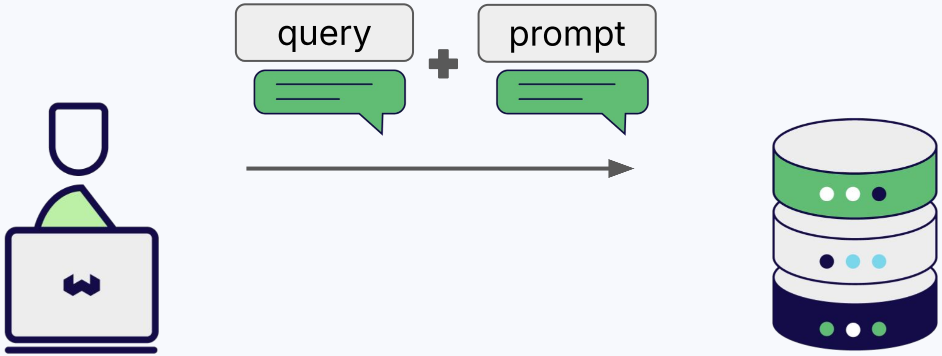
RAG



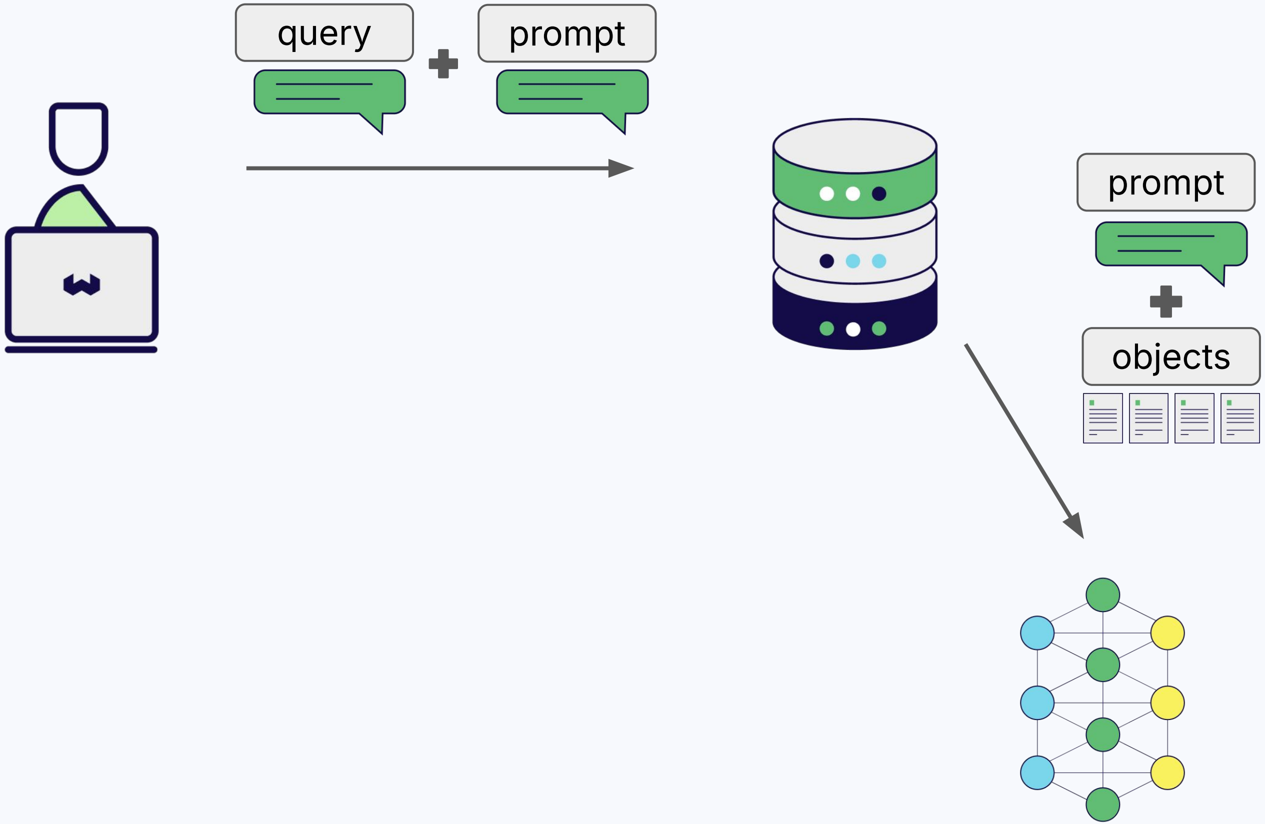
Overhead



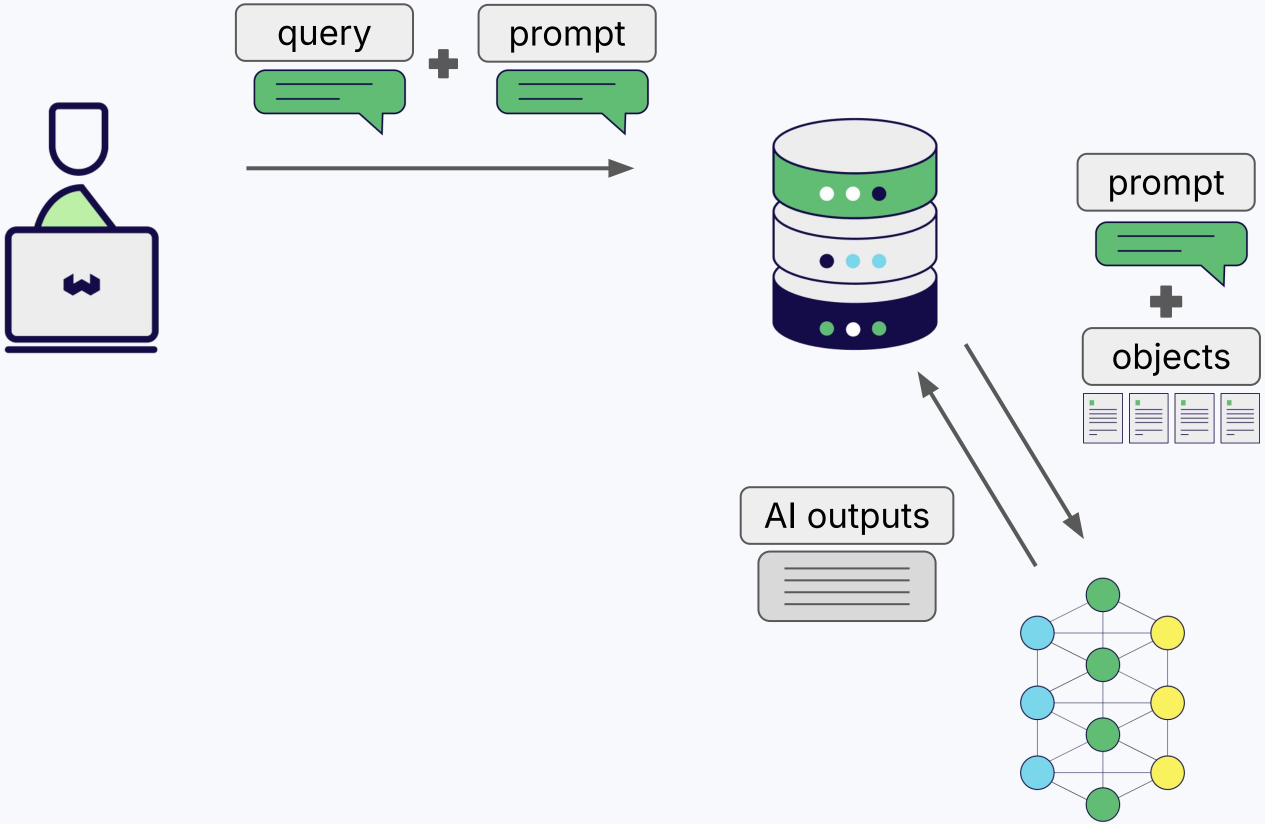
RAG



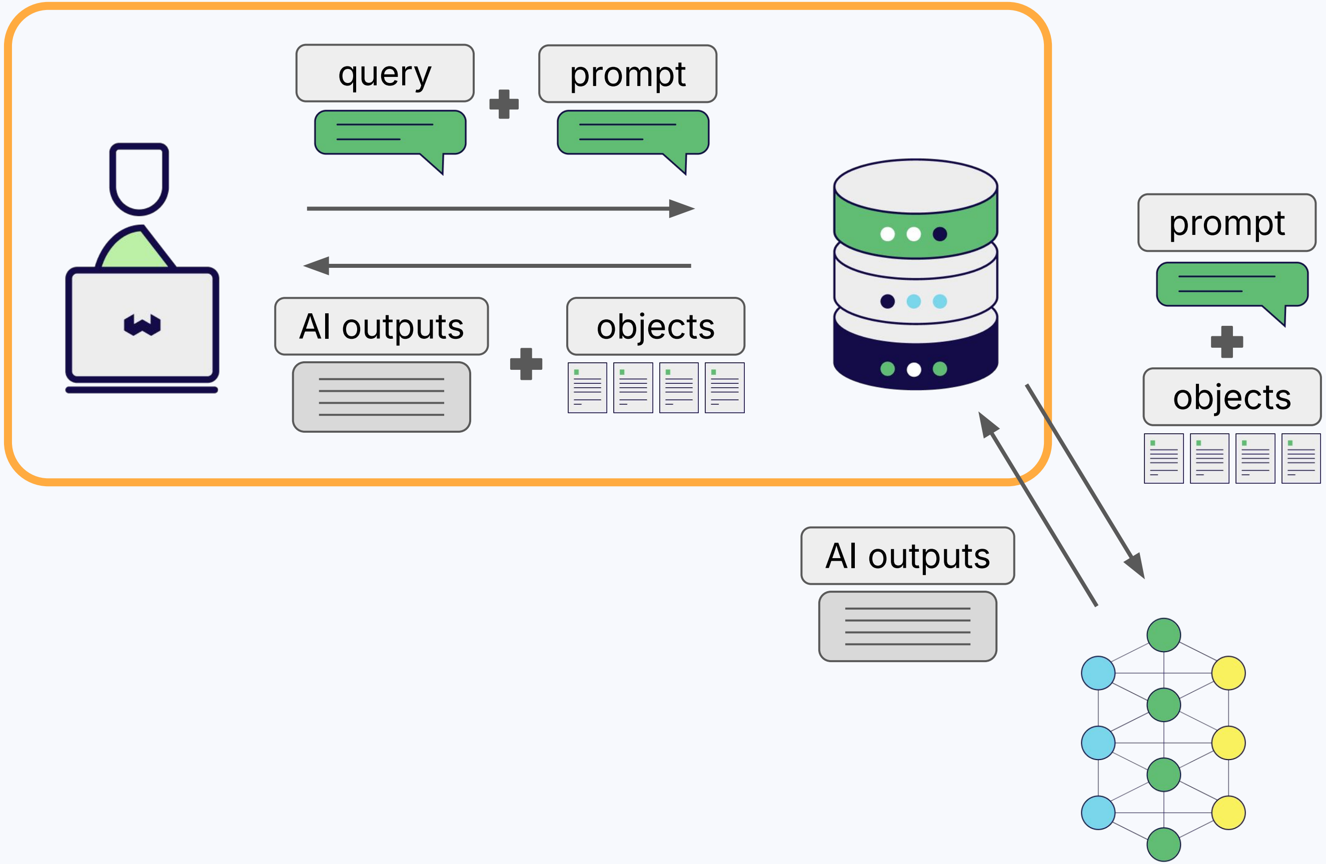
RAG



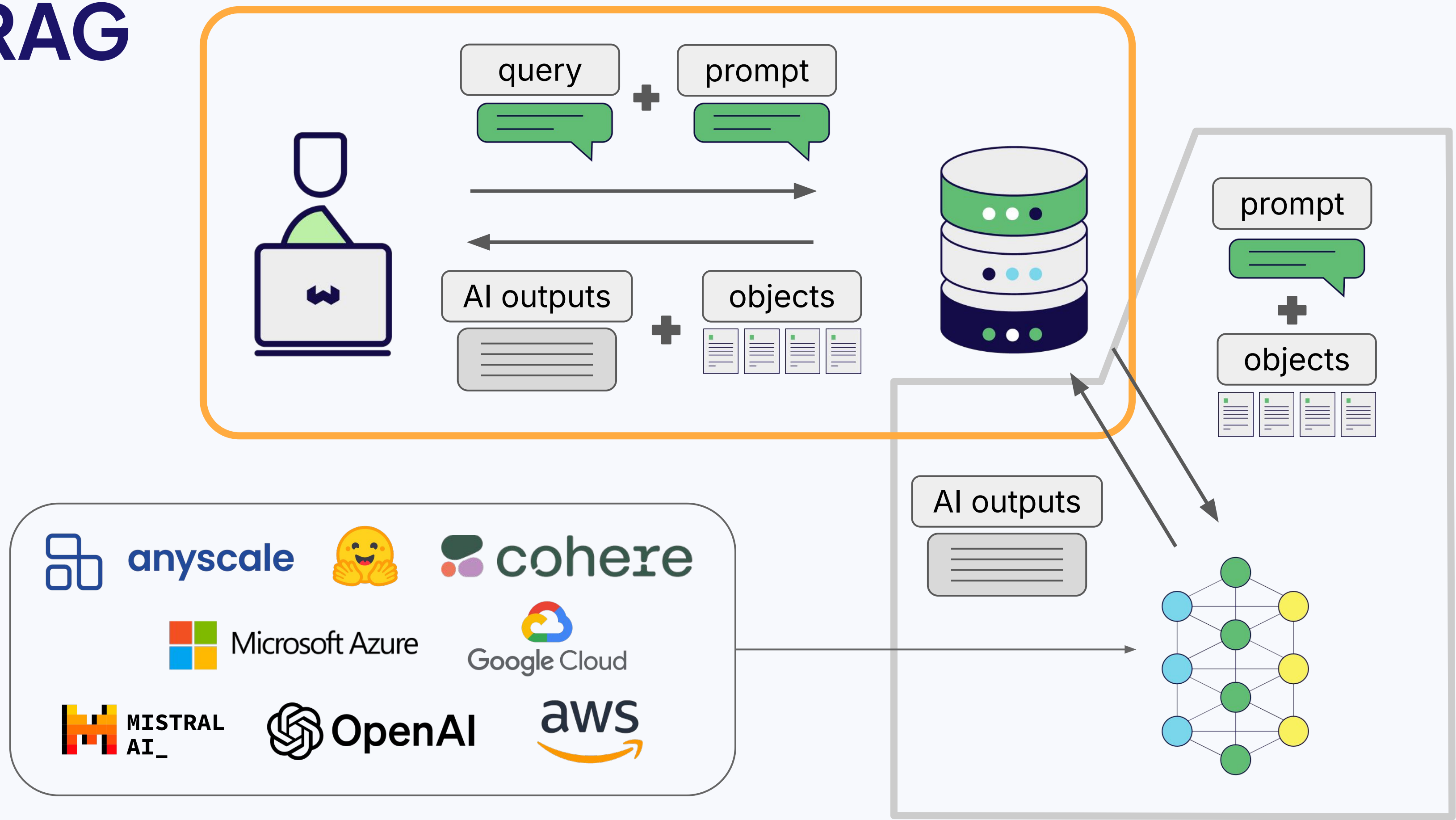
RAG



RAG



RAG


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

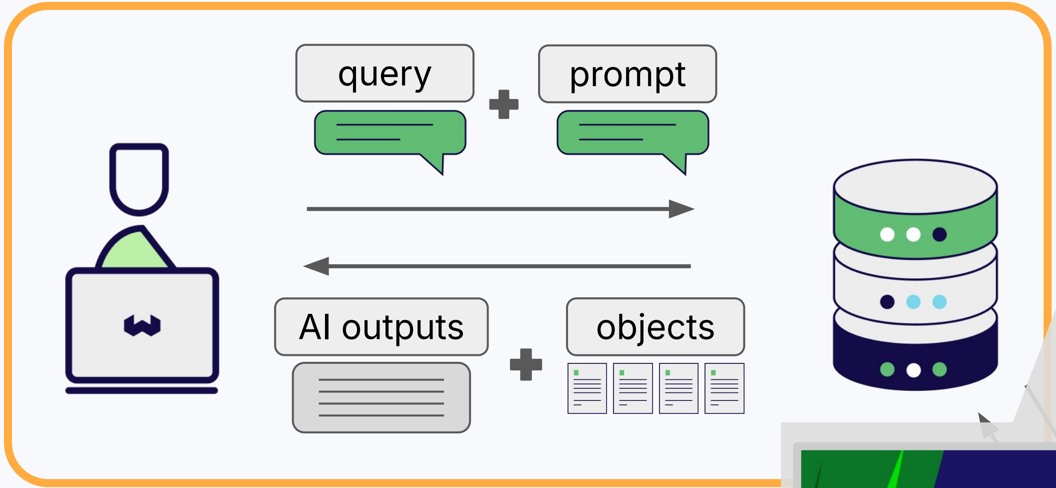
Simplified Experience

```
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

Thank you

Connect with us!

weaviate.io

weaviate/weaviate

weaviate_io