weaviate

GenAI Beyond Prototyping: The Path to Production with AI-Native Databases

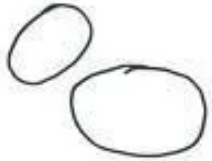
JP Hwang
Developer Educator

Let’s talk about…

Going to production

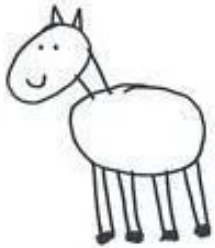
HOW TO:
DRAW A HORSE

BY VAN OKTOP

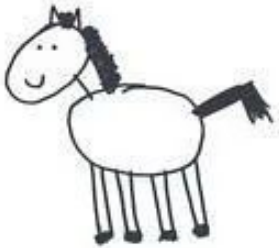


(1) DRAW 2 CIRCLES

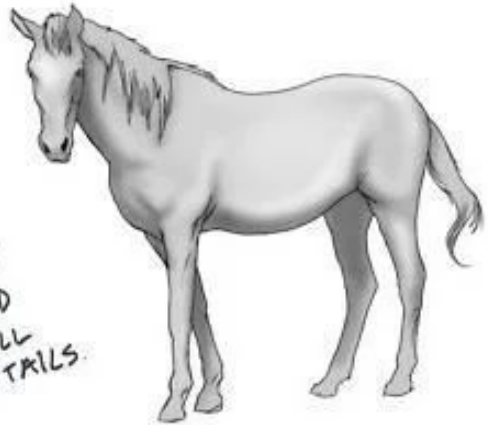
(2) DRAW THE LEGS



(3) DRAW THE FACE



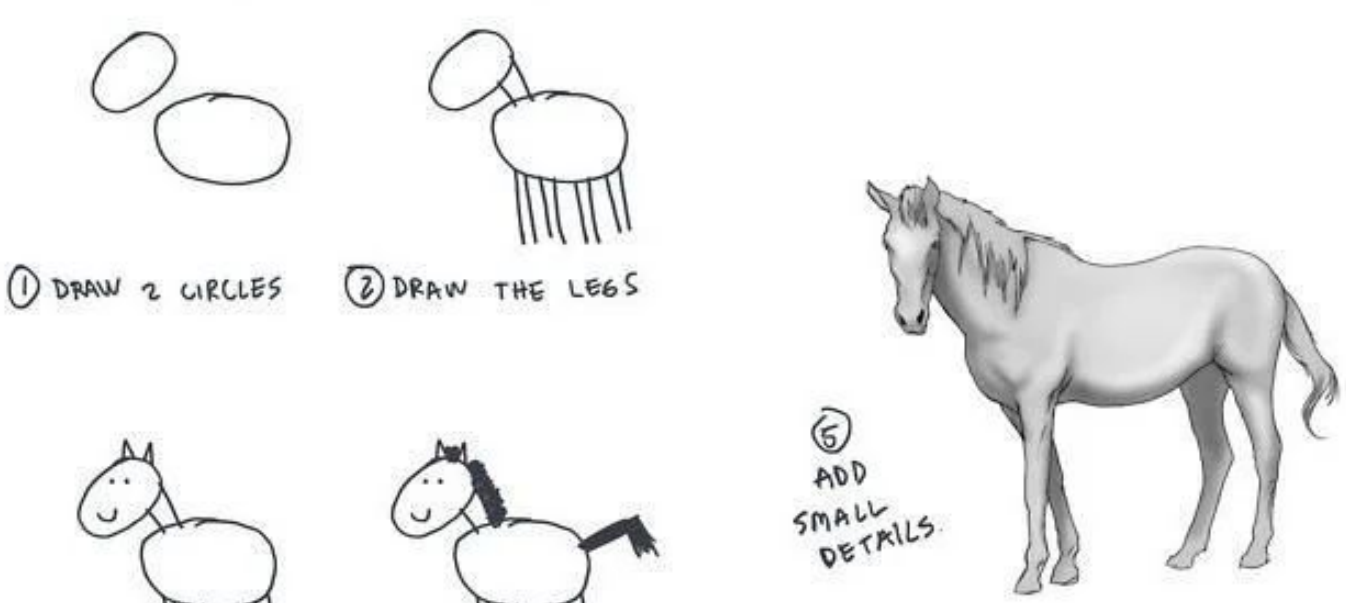
(4) DRAW THE HAIR



(5) ADD SMALL DETAILS

HOW TO:

Take your prototype app to production

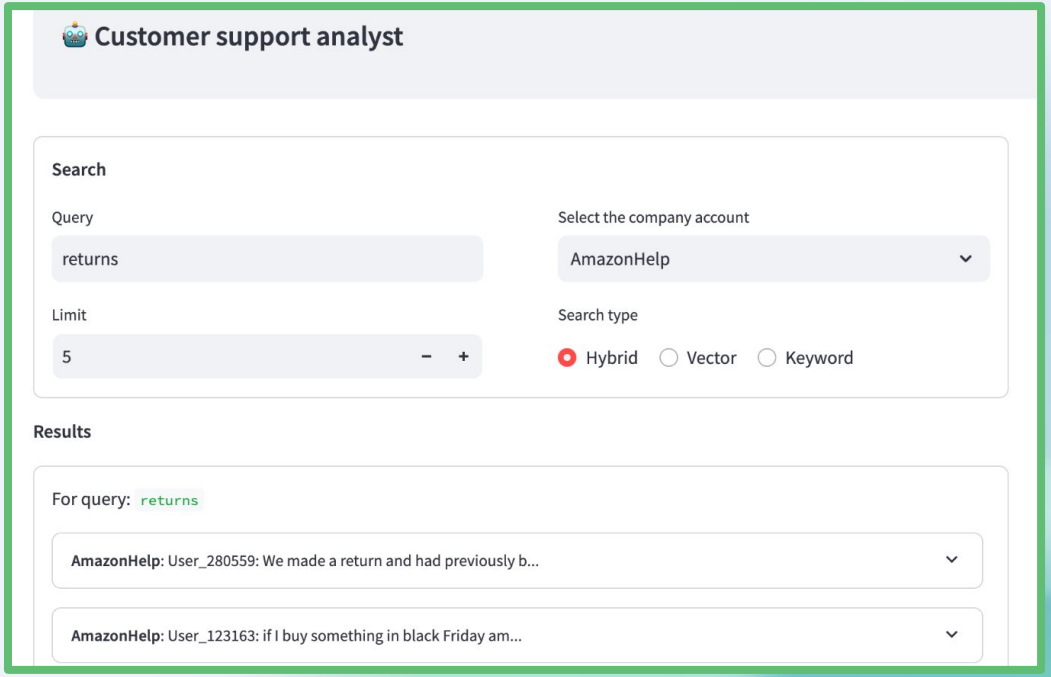


Prototype

Production

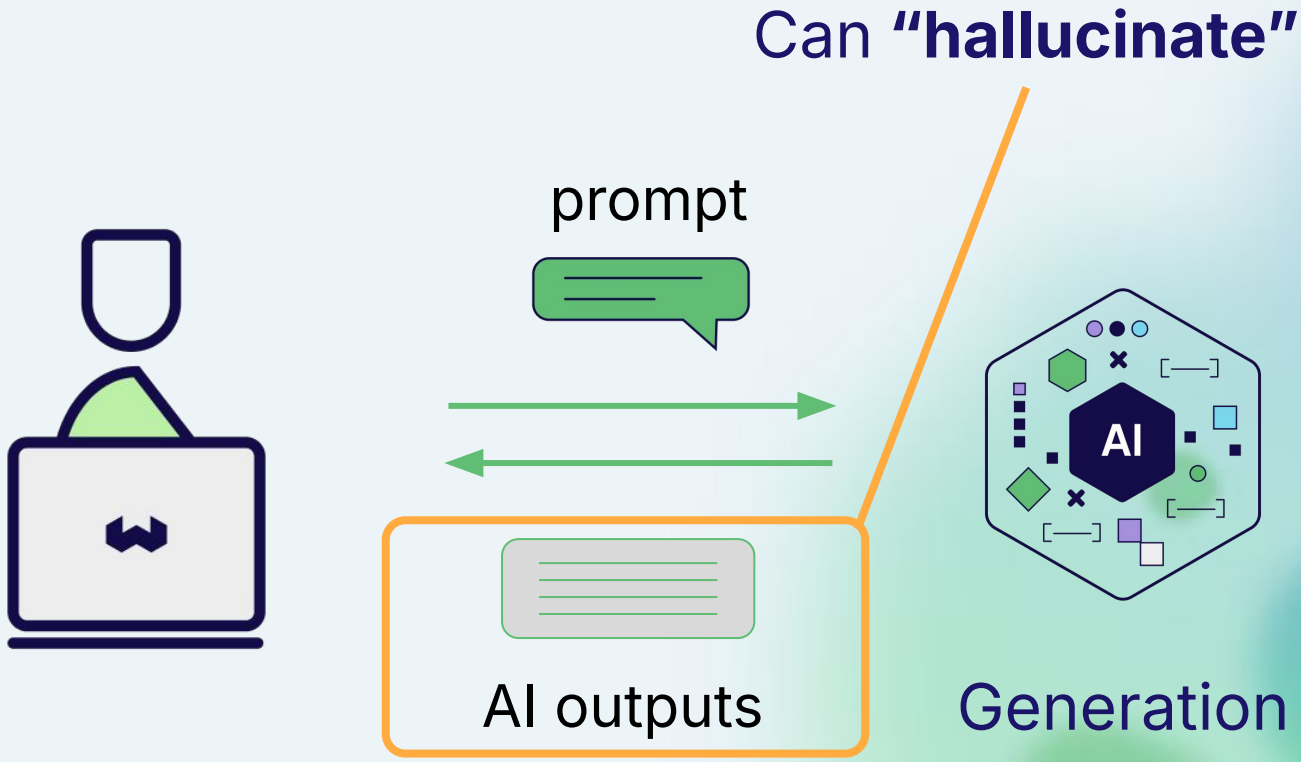
Demo!

- Large language model
- Vectorizer model
- Web app framework
- Vector search algorithm

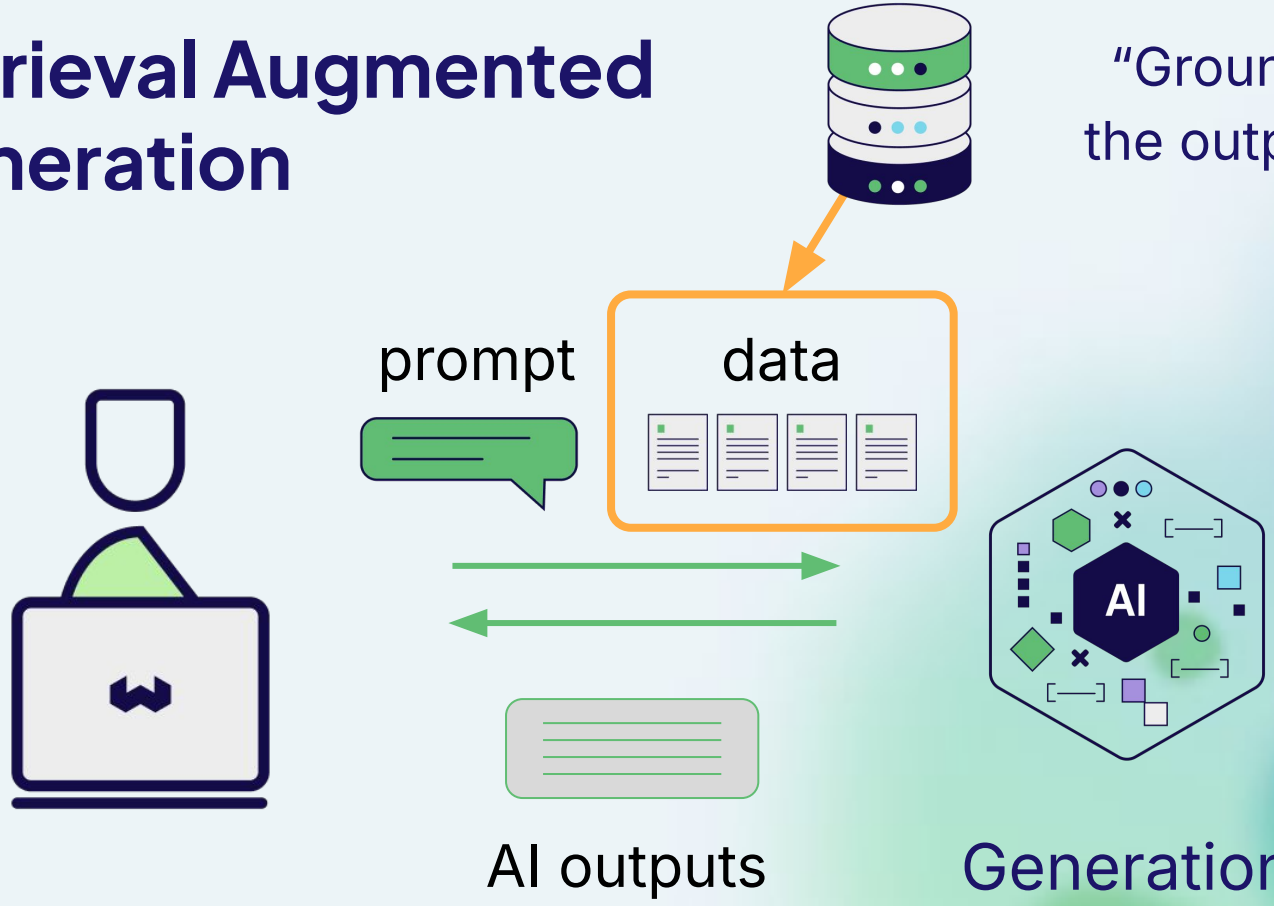


Databases & RAG
(Retrieval Augmented Generation)

Large Language Model



Retrieval Augmented Generation



Hey, this prototype is popular!

Let’s turn it into a spin-off!

Great idea! But...

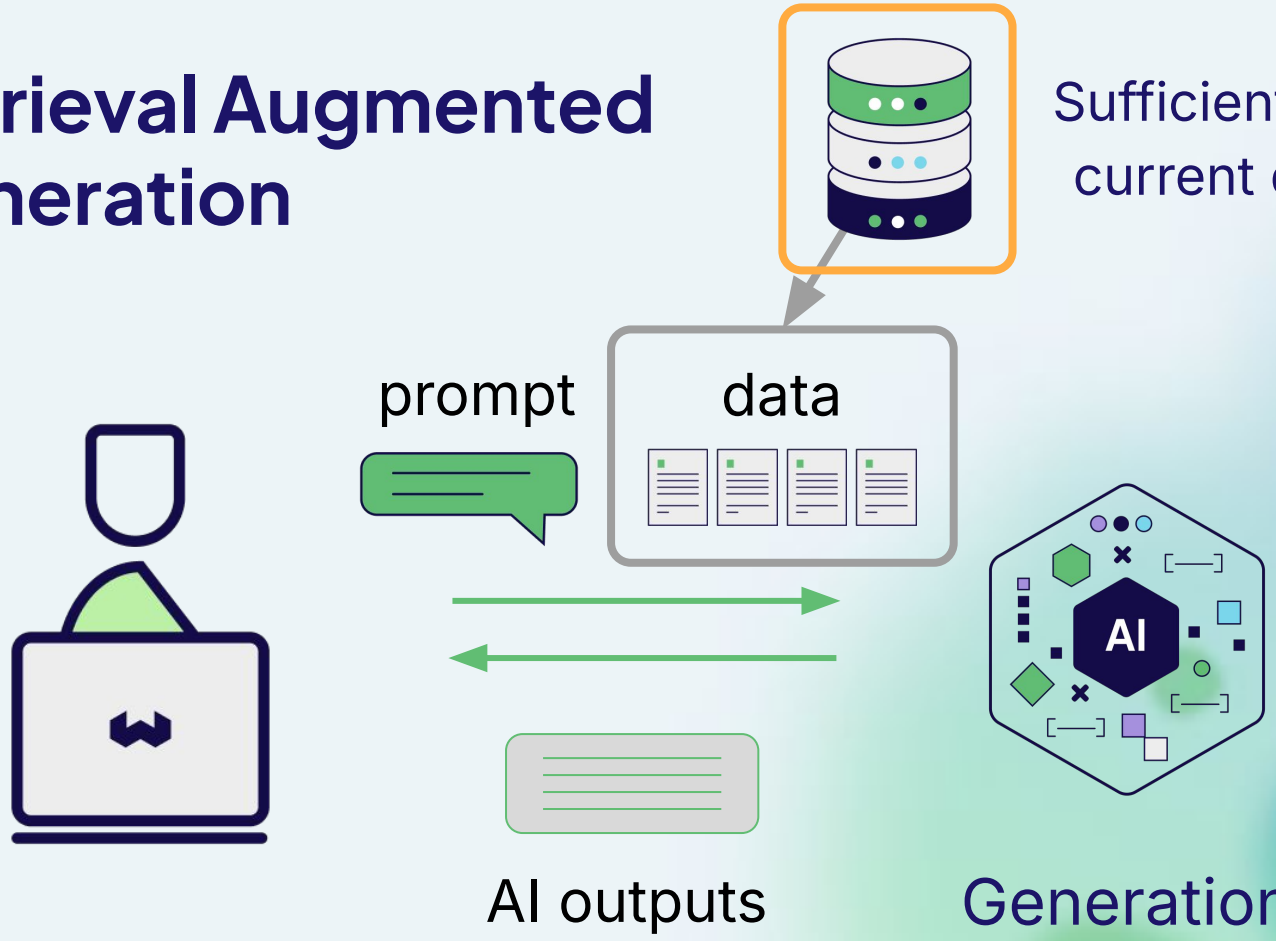
Hey, this prototype is popular!

Let’s turn it into a spin-off!

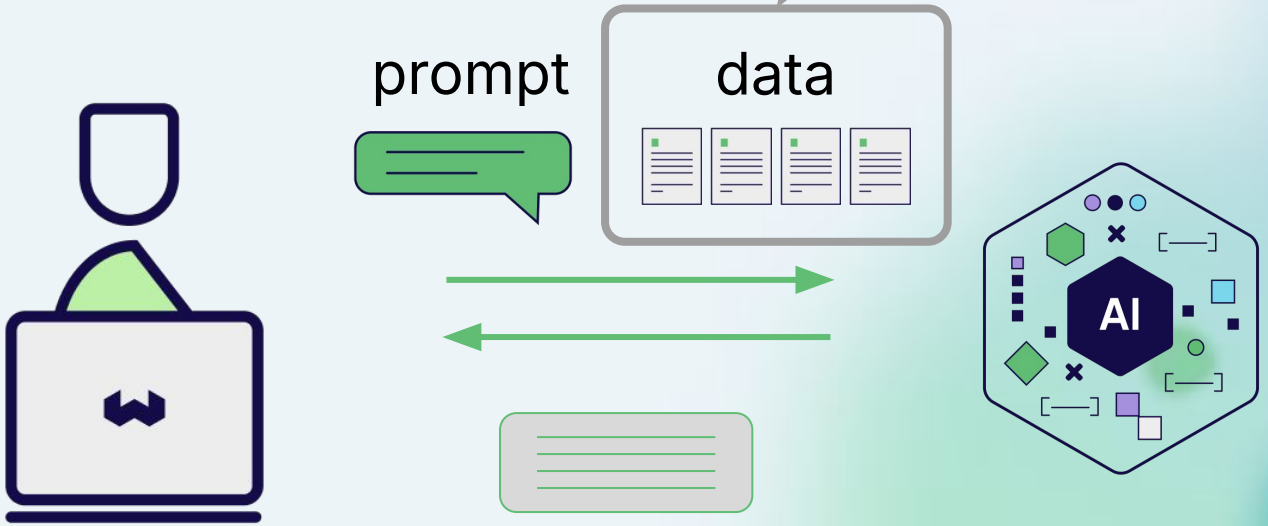
How do we:

- Deal with 1000x more data?
- Meet regulatory requirements?
- Reduce downtime?
- AND make money?

Retrieval Augmented Generation



Retrieval Augmented Generation



Dealing with scale

Search speed

Gen AI: Prototyping to Production

🤖 Customer support analyst

Search

Query

returns

Select the company account

AmazonHelp

Limit

5 − +

Search type

Hybrid Vector Keyword

Results

For query: `returns`

AmazonHelp: User_280559: We made a return and had previously b...

AmazonHelp: User_123163: if I buy something in black Friday am...

ℹ️ Example source accounts: **AmazonHelp** (26351), **AppleSupport** (15397), **Uber_Support** (10876), **Delta** (6686), **AmericanAir** (6477), **comcastcares** (5569), **hulu_support** (4249), **SpotifyCares** (6055), **TMobileHelp** (4699), **SouthwestAir** (4931)

Cluster statistics

Object count

200000

Nodes

1

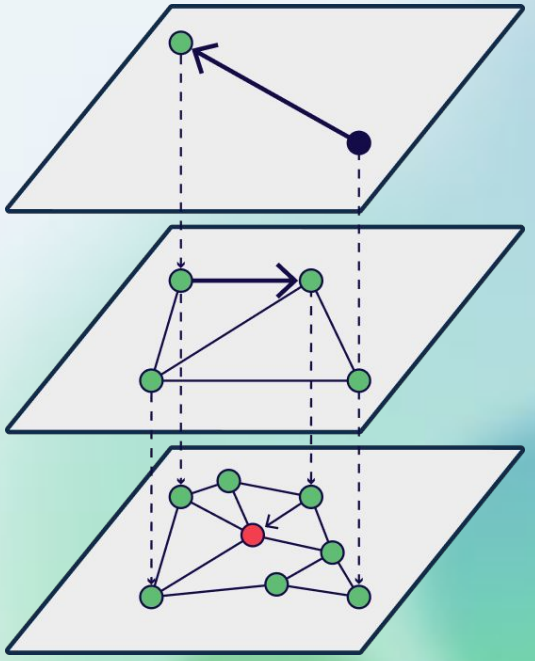
Scale: Solutions

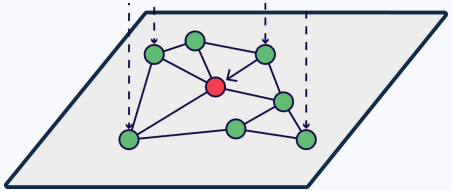
Improve efficiency - indexing

Scale: Solutions

Improve efficiency - indexing

- Default: **HNSW** index

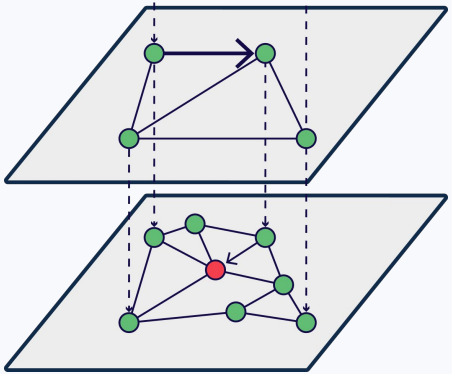




HNSW

HNSW in focus

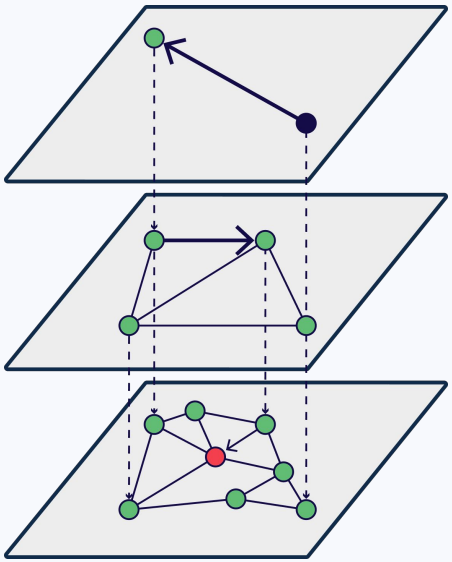
- Layered graphs
- Lowest layer: all vectors (nodes)
  - Nodes connected to neighbors



HNSW

HNSW in focus

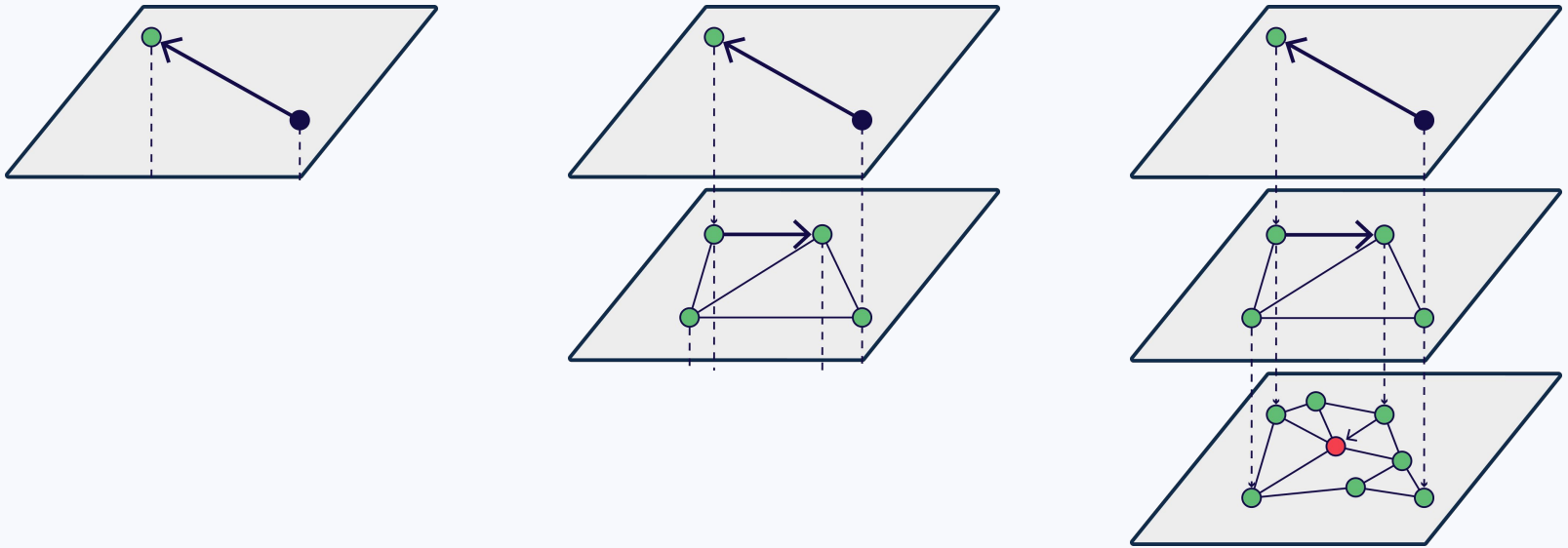
- Layered graphs
- Lowest layer: all vectors (nodes)
  - Nodes connected to neighbors
- Upper layers: subsets of vectors



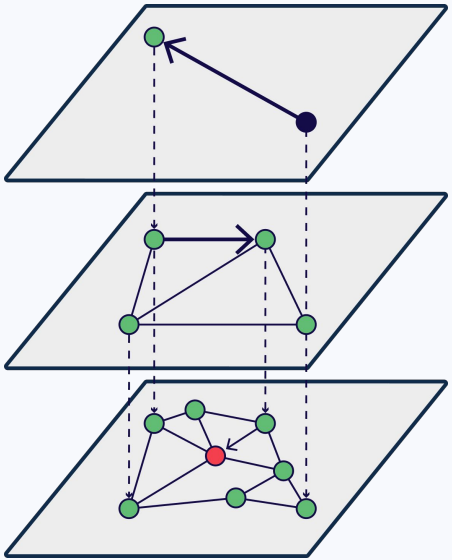
HNSW

HNSW in focus

- Layered graphs
- Lowest layer: all vectors (nodes)
  - Nodes connected to neighbors
- Upper layers: subsets of nodes
  - Higher ↔ fewer nodes
  - Higher ↔ faster traversal
- Multi-dimensional skip lists

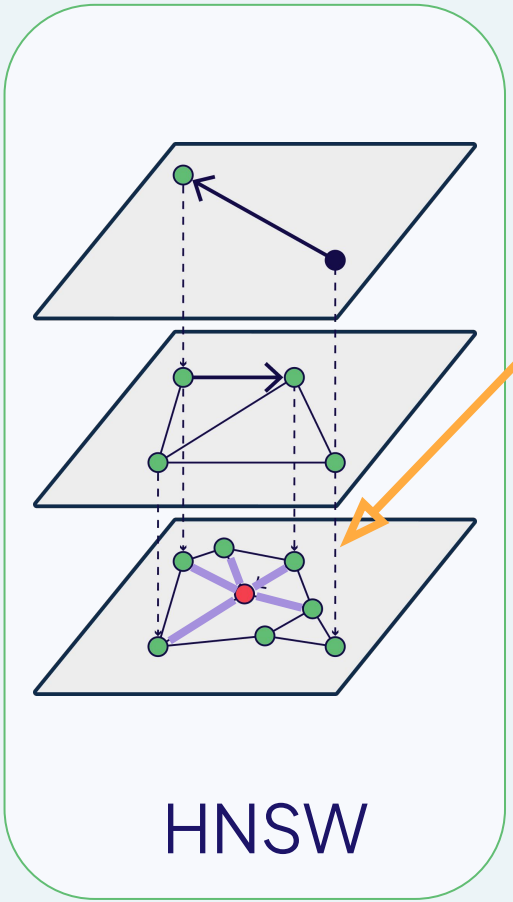


HNSW



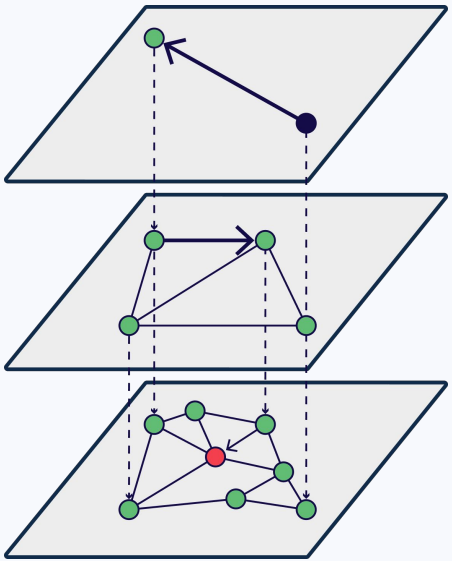
HNSW

HNSW parameters



HNSW parameters

- **maxConnections** (m)
  - N (node → node) connections
    - e.g. 32
  - Note: bottom layer → m * 2

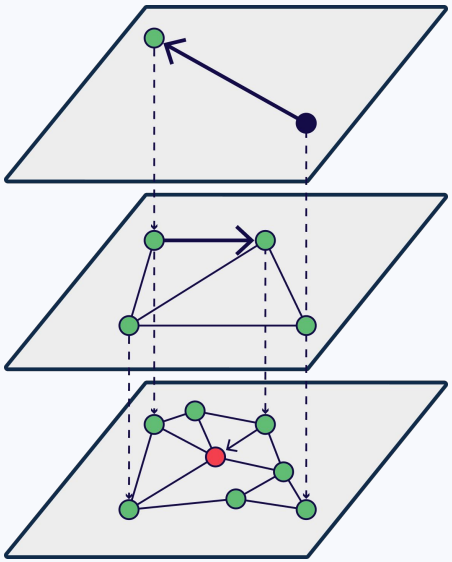


HNSW

HNSW parameters

- maxConnections (m)
- **ef** → size of list during search
  - Higher → better
  - Cost: slower

rank	object	distance
1	<uuid>	0.0514
2	<uuid>	0.0642
...	...	...
...	...	...
...	...	...
...	...	...
max(ef, limit)	<uuid>	0.3532



HNSW

HNSW parameters

- maxConnections (m)
- ef
- **efConnections**
  - For graph construction
  - Higher → better

    Slower insertion

Solutions

Search speed

- Vector index
  - HNSW
  - Logarithmic scaling
  - Scalable, fast search
  - Configurable parameters

Dealing with scale

Resource management

Search

Query

returns

Select the company account

AmazonHelp

Limit

5

Search type

Hybrid | Vector | Keyword

Results

For query: `returns`

AmazonHelp: User_280559: We made a return and had previously b...

AmazonHelp: User_123163: if I buy something in black Friday am...

AmazonHelp: User_247957: I didn't know a product didn't have f...

RAG

What should we do with the search results?

What patterns do you recognize in these comments?

Generate response

Example source accounts: **AmazonHelp** (26351), **AppleSupport** (15397), **Uber_Support** (10876), **Delta** (6686), **AmericanAir** (6477), **comcastcares** (5569), **hulu_support** (4249), **SpotifyCares** (6055), **TMobileHelp** (4699), **SouthwestAir** (4931)

Cluster statistics

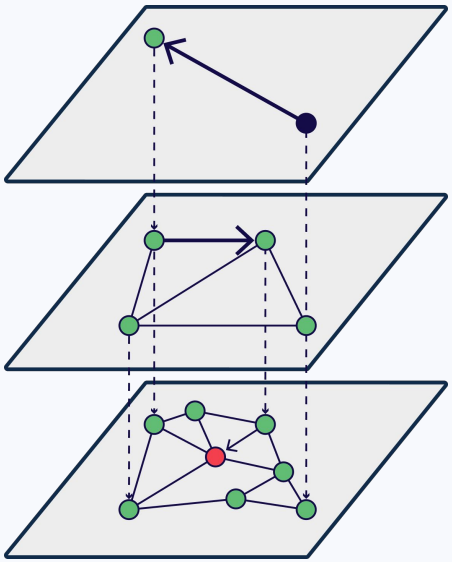
Object count

200000

Nodes

1

Memory Usage Over Time



HNSW

HNSW in focus

- In-memory data structure
- Typical limitation: memory footprint
  - Connections (~10B/ea)
  - Vectors (~4B/dim; 6kB/vector)
  - @10M objects
    - ~65GB!
  - @100M → 650GB, etc...

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

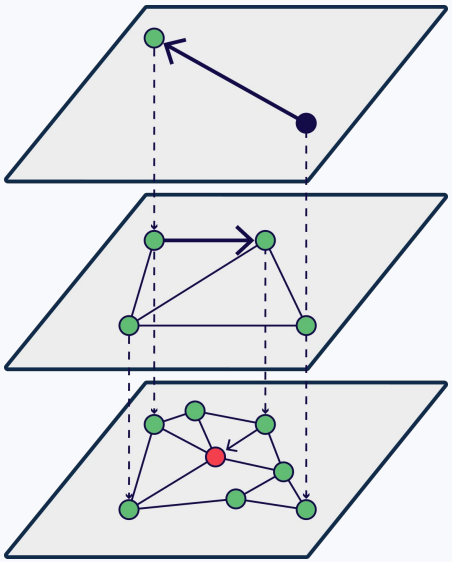
1024 gib

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!**

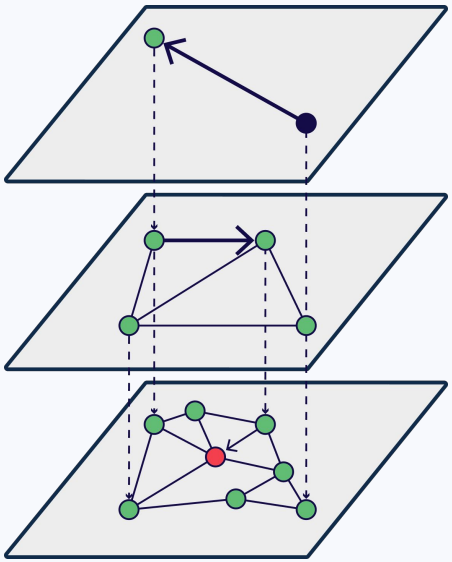
AWS Spot pricing, Nov 2024



HNSW

HNSW in focus

- In-memory data structure
- Typical limitation: memory footprint
  - Connections (~10B/ea)
  - **Vectors (~4B/dim; 6kB/vector)**
  - @10M objects
    - ~65GB!
  - @100M → 650GB, etc...



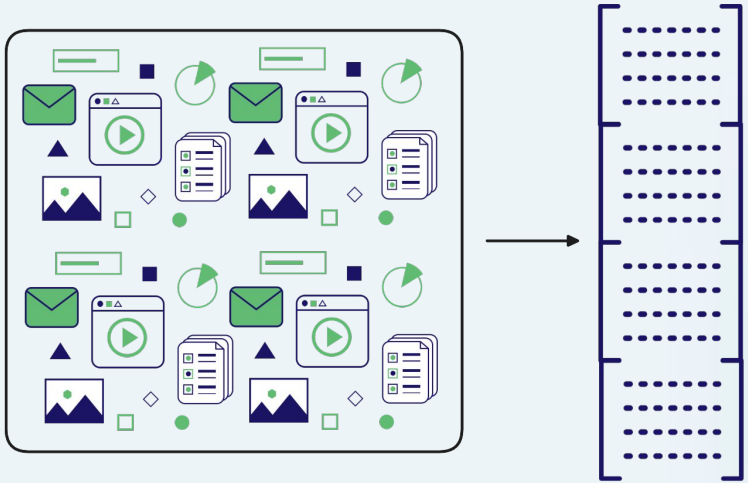
HNSW

Quantization

- Compression
- Reduce precision from floats
  - Binary / scalar / locally adaptive
- Reduce number of dimensions
  - Product quantization

Scale: Solutions

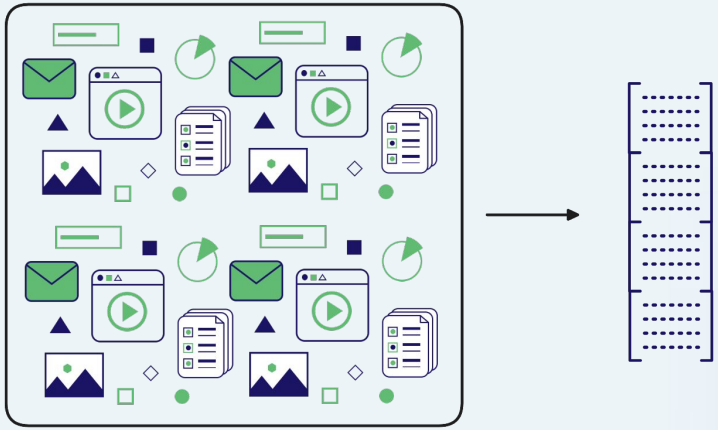
Improve efficiency



Scale: Solutions

Improve efficiency

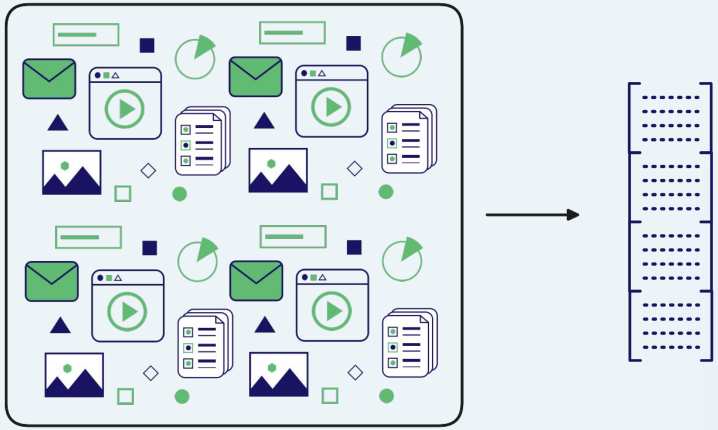
Compression



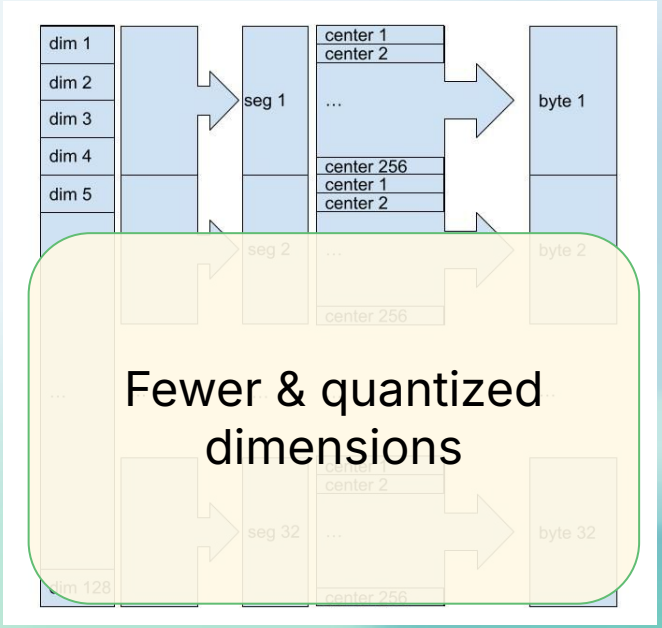
Scale: Solutions

Improve efficiency

Compression

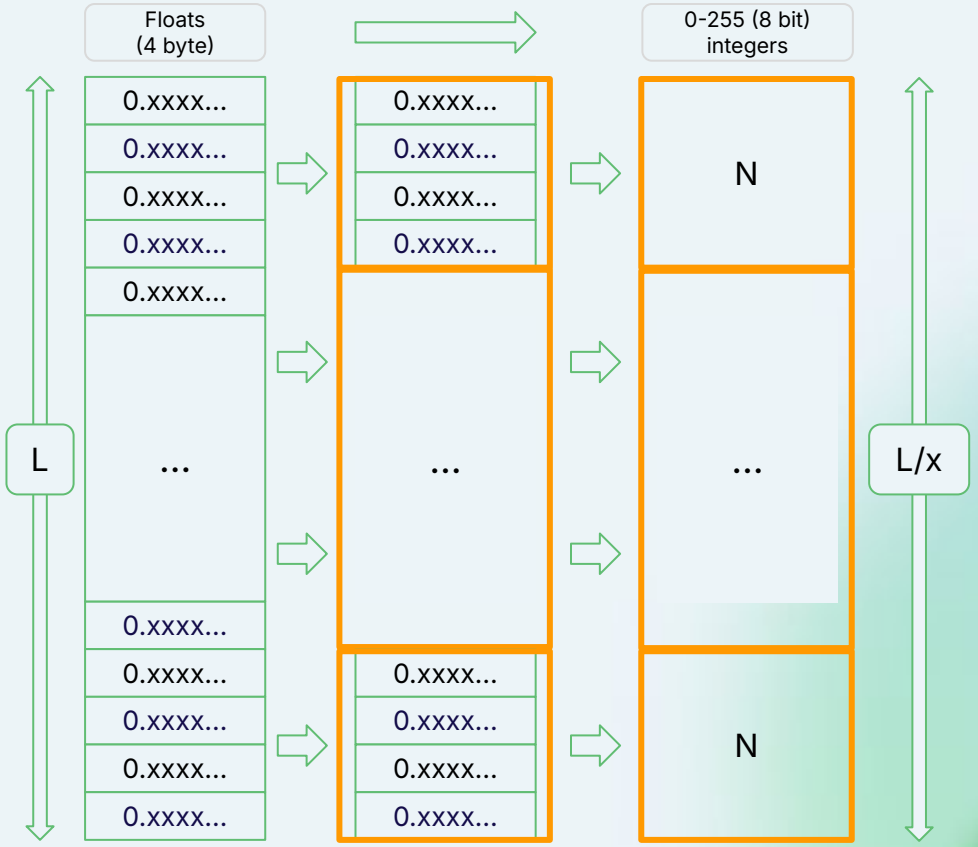


Product quantization



Customisable compression

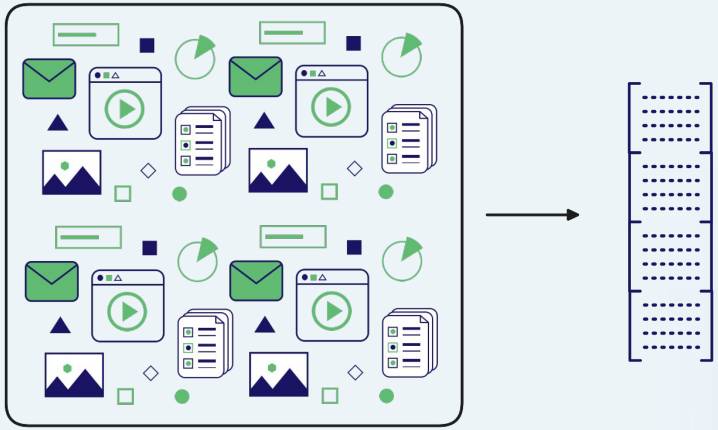
(e.g. 128 floats → 32 bytes: 16x)


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

Scale: Solutions

Improve efficiency

Compression



Binary quantization

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

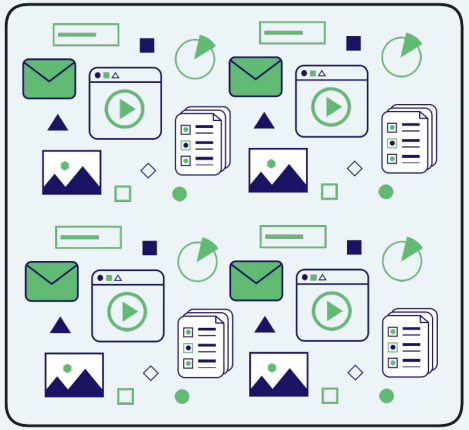
↓

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

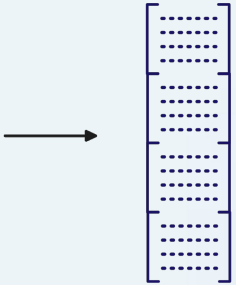
n floats → n bits
(32x reduction)

Scale: Solutions

Improve efficiency



Compression



Scalar quantization

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

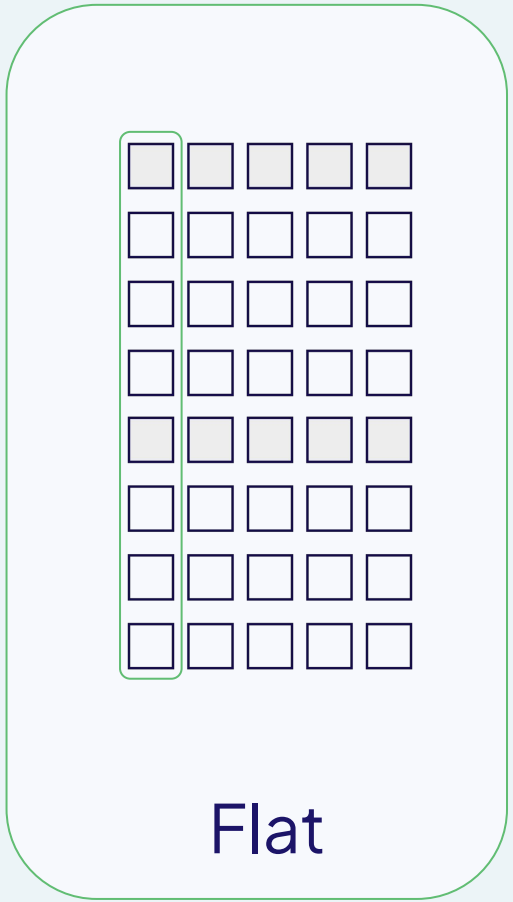
[104, 12, 138,
...
152, 138 47]

n floats → n ints
(4x reduction)

Scale: Solutions

Binary quantization

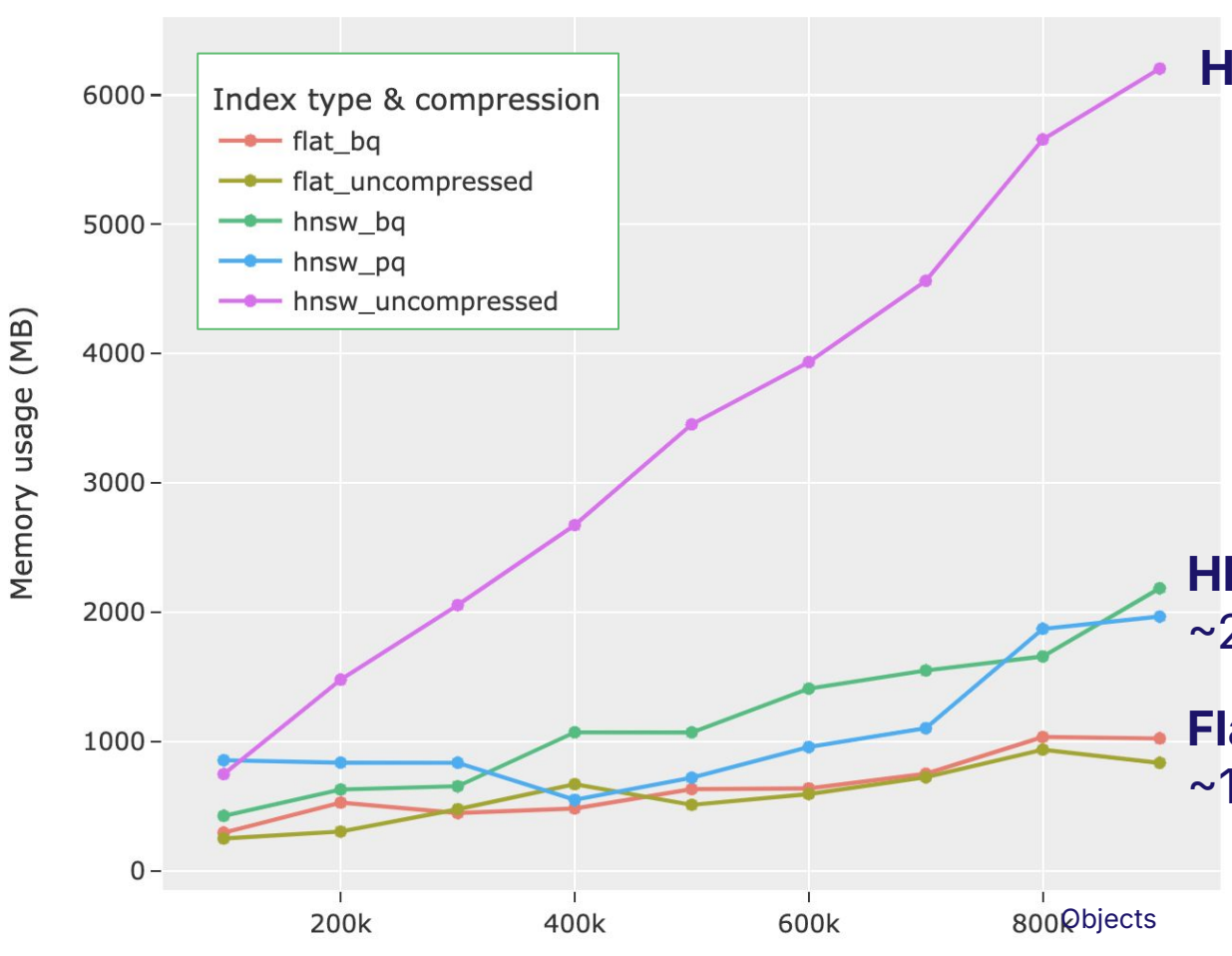
```
articles = client.collections.create(
    name="Article",
    properties=[
        Property(name="title", data_type=DataType.TEXT)
    ],
    vector_index_config=wc.Configure.VectorIndex.flat(
        quantizer=wc.Configure.VectorIndex.Quantizer.bq()
    ),
)
```



Flat index

- Simple data structure (e.g. map)
- Low overhead
- Does not scale well
  - Linear search
  - Good for small sets

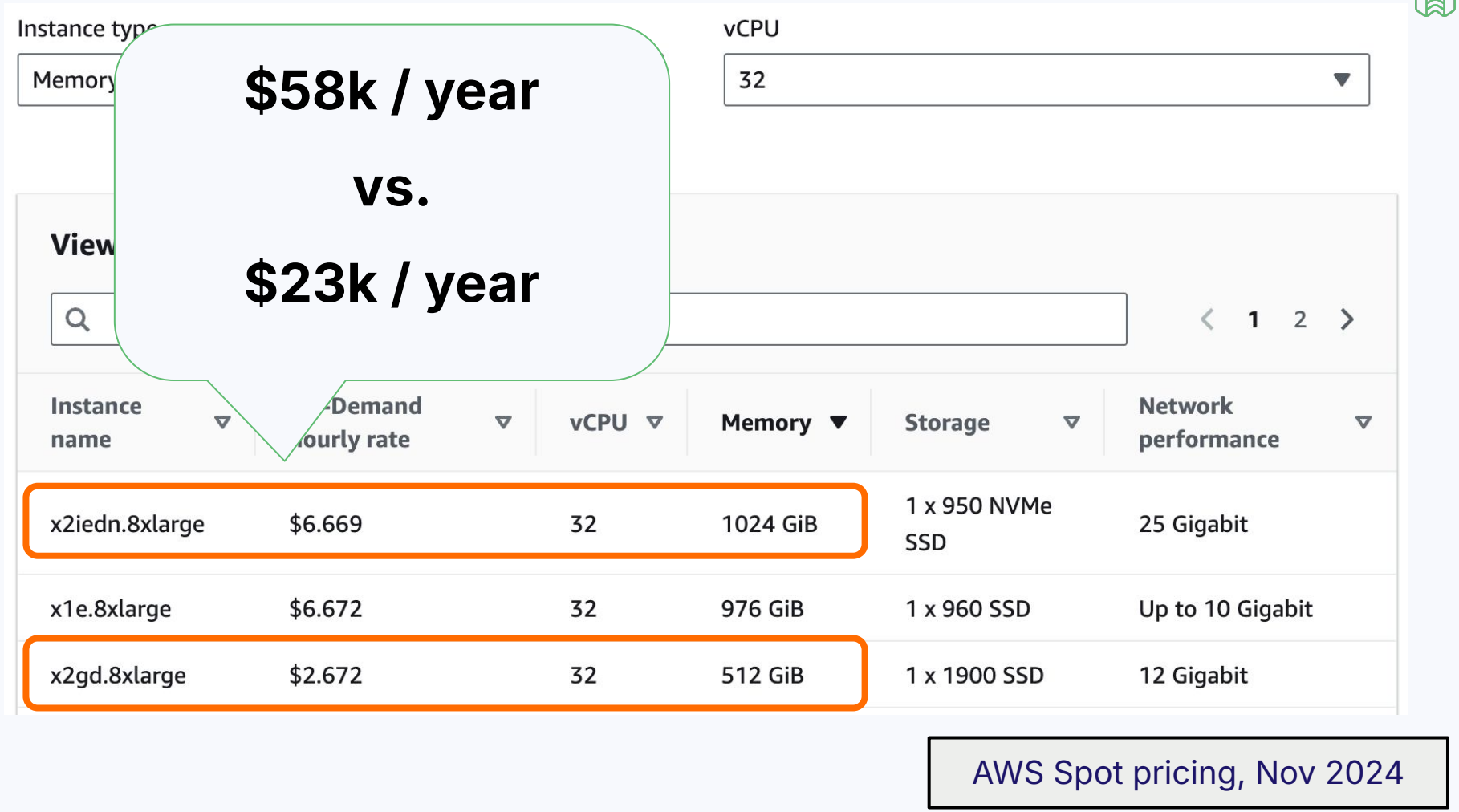
Example: Index type & quantization vs memory footprint



HNSW: ~6GB

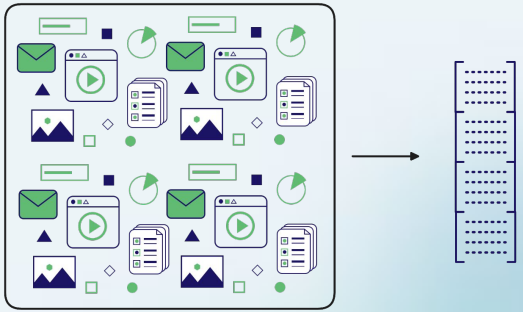
HNSW + BQ/PQ: ~2GB

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

Solutions

Memory usage

- Quantization
  - Product, binary, scalar...
  - Trade off memory vs quality
  - Can also speed up search
- Flat index for small subsets

Dealing with scale

Scaling (horizontal)

Search

Query

returns

Select the company account

AmazonHelp

Limit

5 − +

Search type

Hybrid Vector Keyword

Results

For query: `returns`

AmazonHelp: User_280559: We made a return and had previously b...

AmazonHelp: User_123163: if I buy something in black Friday am...

AmazonHelp: User_247957: I didn't know a product didn't have f...

RAG

What should we do with the search results?

What patterns do you recognize in these comments?

Generate response

Example source accounts: **AmazonHelp** (26351), **AppleSupport** (15397), **Uber_Support** (10876), **Delta** (6686), **AmericanAir** (6477), **comcastcares** (5569), **hulu_support** (4249), **SpotifyCares** (6055), **TMobileHelp** (4699), **SouthwestAir** (4931)

Cluster statistics

Object count

200000

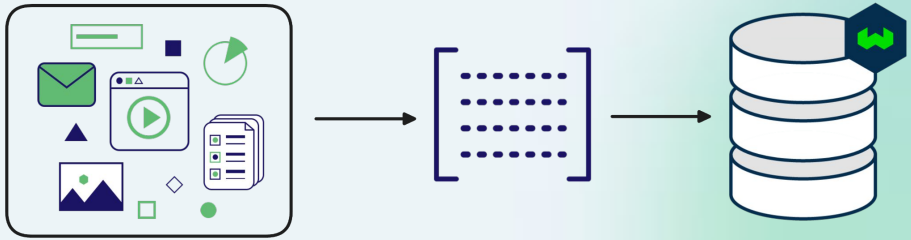
Nodes

1

Memory Usage Over Time

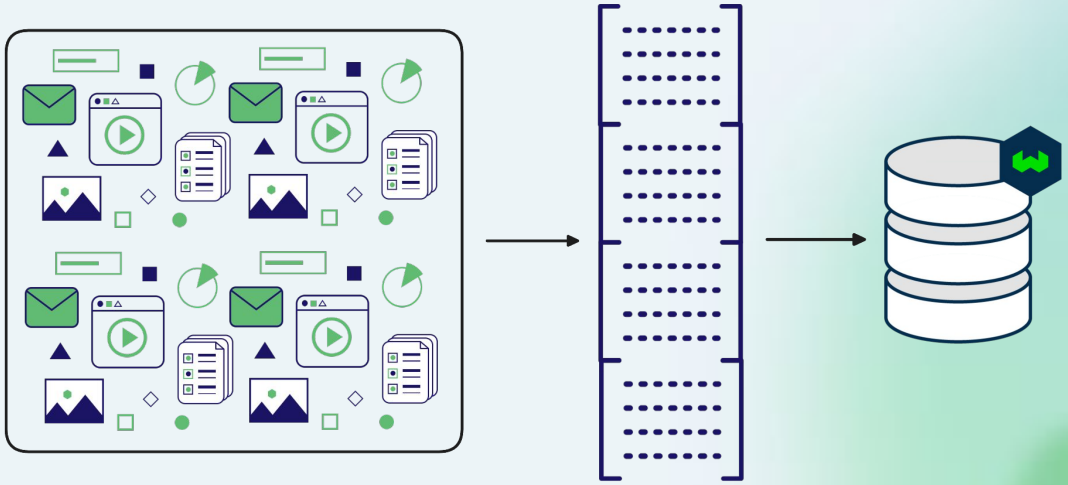
Scale: Solutions

Growth



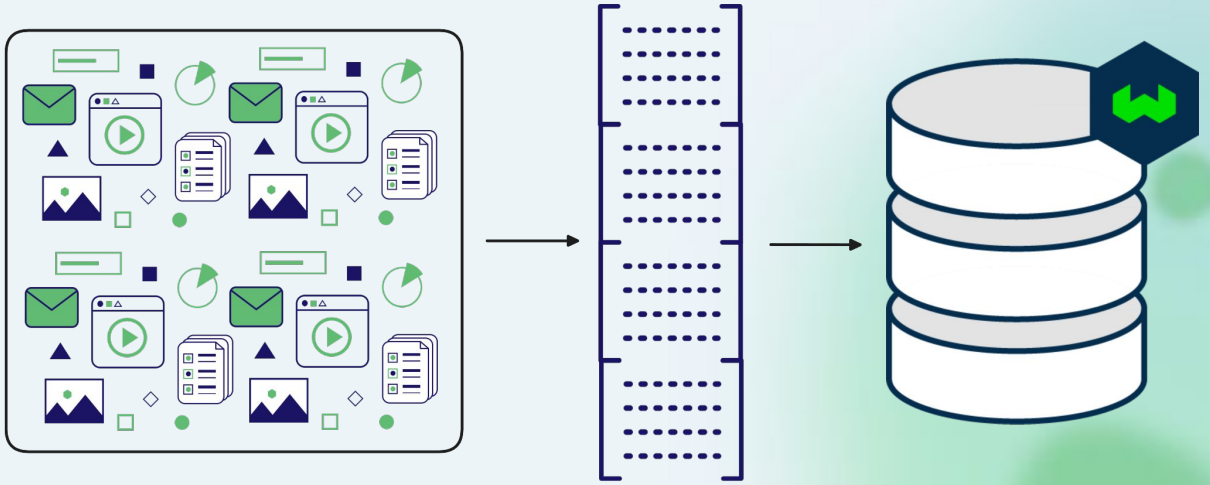
Scale: Solutions

Growth



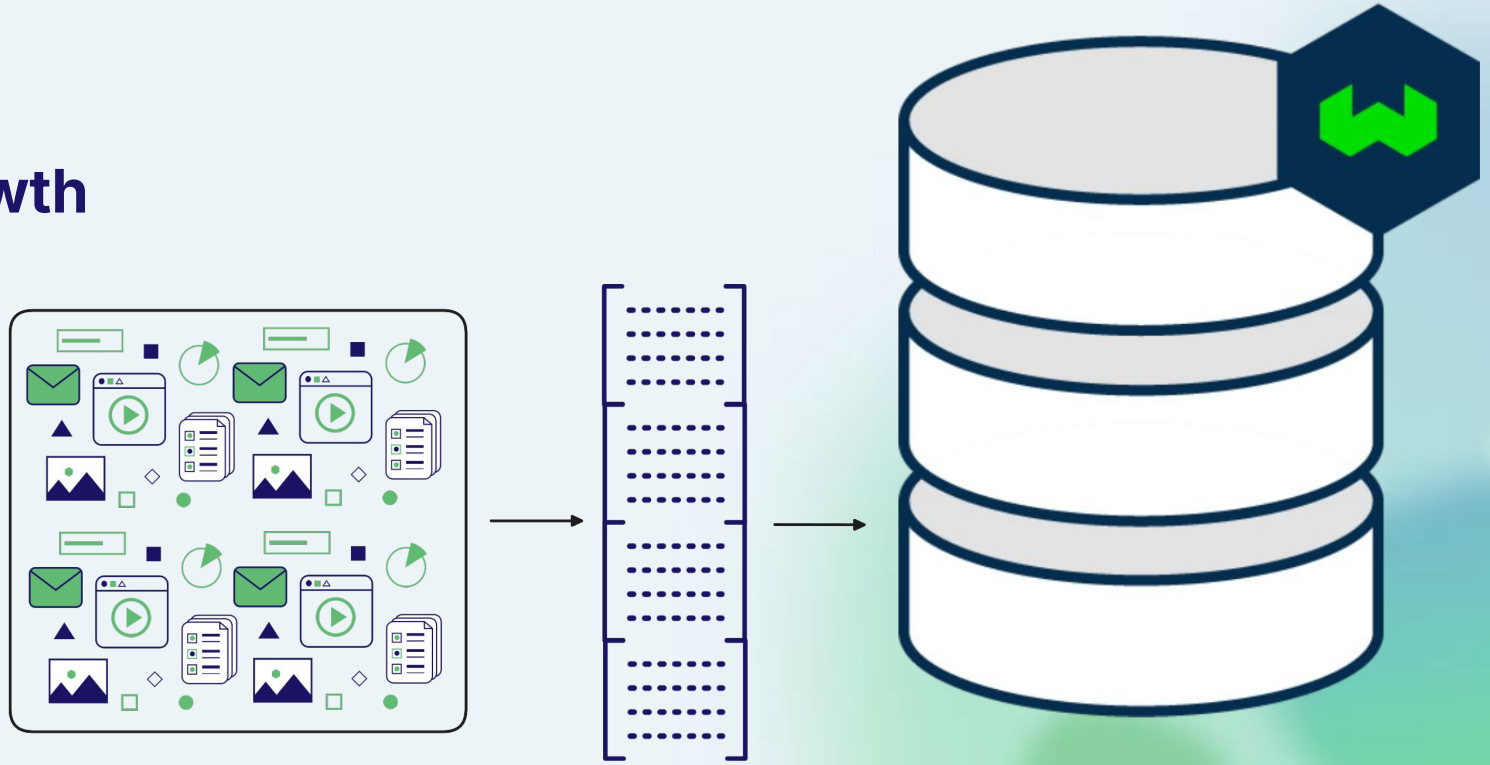
Scale: Solutions

Growth



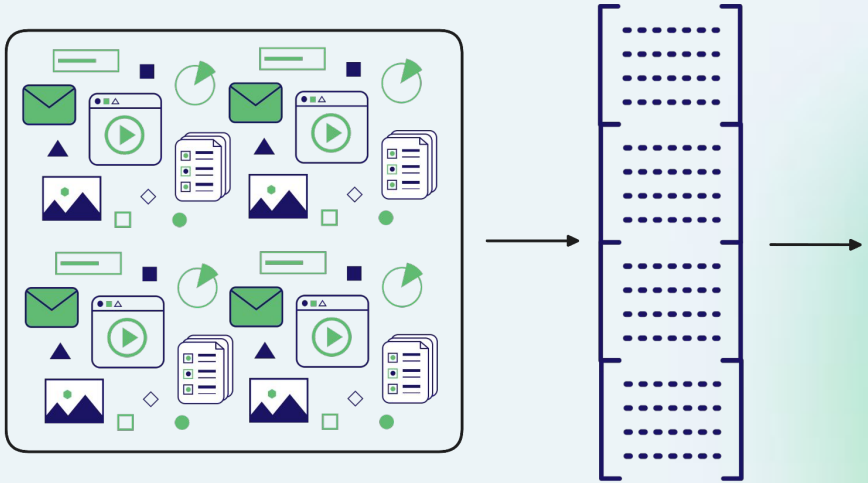
Scale: Solutions

Growth



Scale: Solutions

Growth

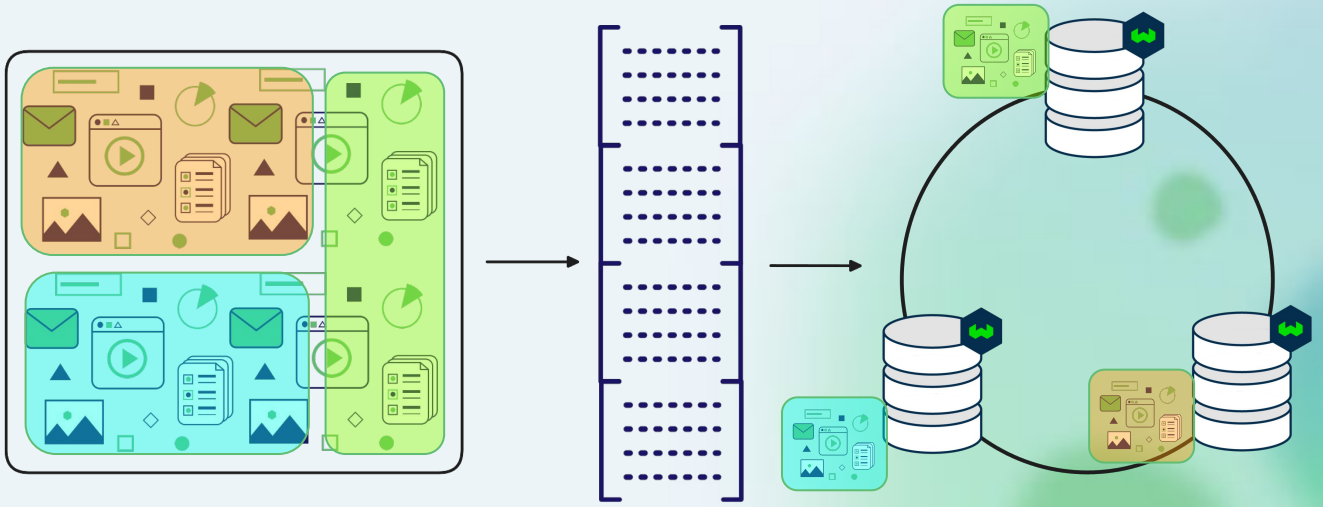


- Single point of failure
- Costs
- Efficiency
- Upgrades

Scale: Solutions

Deal with growth

Horizontal scaling
(sharding)

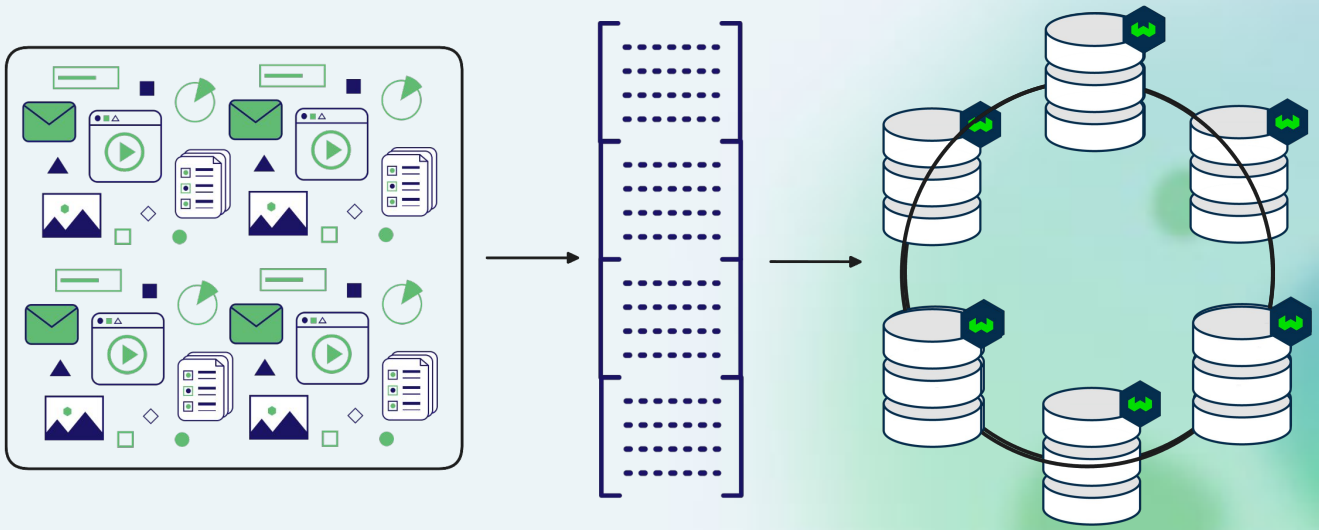


Scale: Solutions

Billion scale since 2022!

Deal with growth

To scale:
Add more nodes

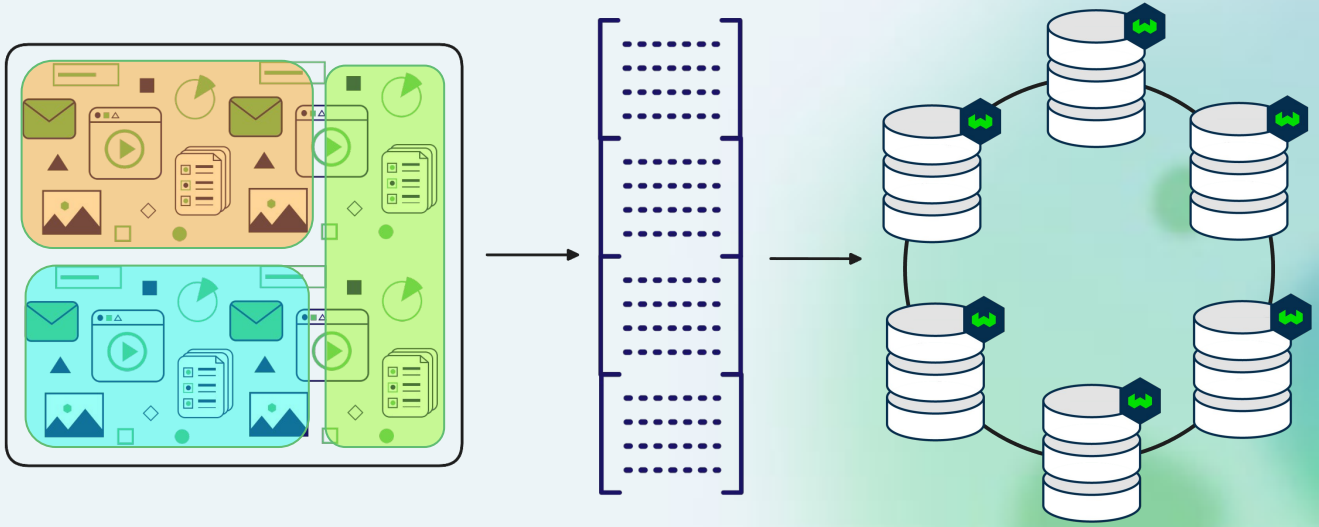


Scale: Solutions

Billion scale since 2022!

Deal with growth

To scale:
Add more nodes

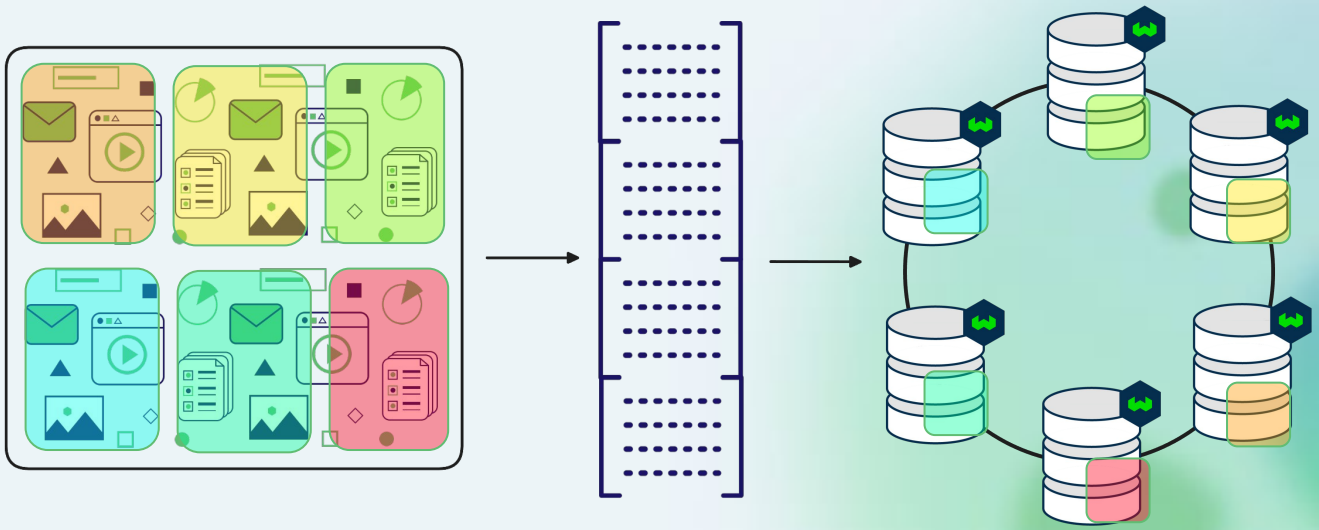


Scale: Solutions

Billion scale since 2022!

Deal with growth

To scale:
Add more nodes



Solutions

Scalability

- Horizontal scaling (sharding)
  - Shard vector + inverted indexes
  - Scale out as needed

Dealing with multiple users

Isolation

Search

Query

returns

Limit

5 − +

Select the company account

AmazonHelp

Search type

Hybrid Vector Keyword

Example source accounts: **AmazonHelp** (26351), **AppleSupport** (15397), **Uber_Support** (10876), **Delta** (6686), **AmericanAir** (6477), **comcastcares** (5569), **hulu_support** (4249), **SpotifyCares** (6055), **TMobileHelp** (4699), **SouthwestAir** (4931)

Results

For query: `returns`

AmazonHelp: User_280559: We made a return and had previously b...

AmazonHelp: User_123163: if I buy something in black Friday am...

AmazonHelp: User_247957: I didn't know a product didn't have f...

RAG

What should we do with the search results?

What patterns do you recognize in these comments?

Generate response

Cluster statistics

Object count

200000

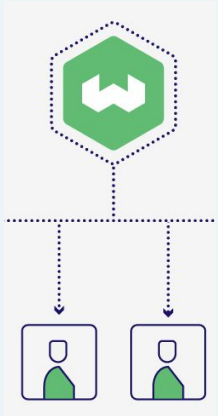
Nodes

1

Memory Usage Over Time

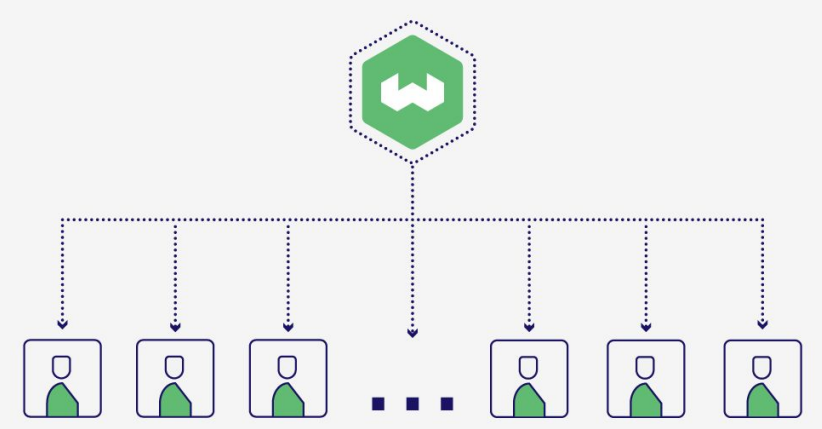
Scale: Solutions

End user growth



Scale: Solutions

End user growth



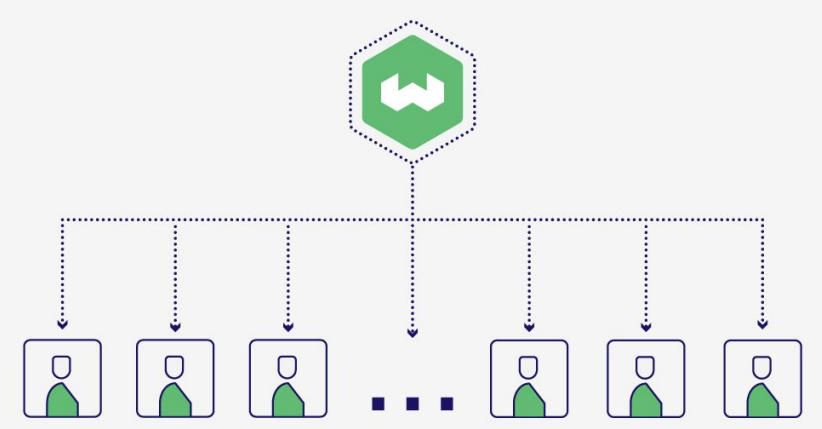
Challenges faced:

- Performance
- Data isolation
- Compliance

Developed: **Multi-tenancy**

Scale: Solutions

End user growth

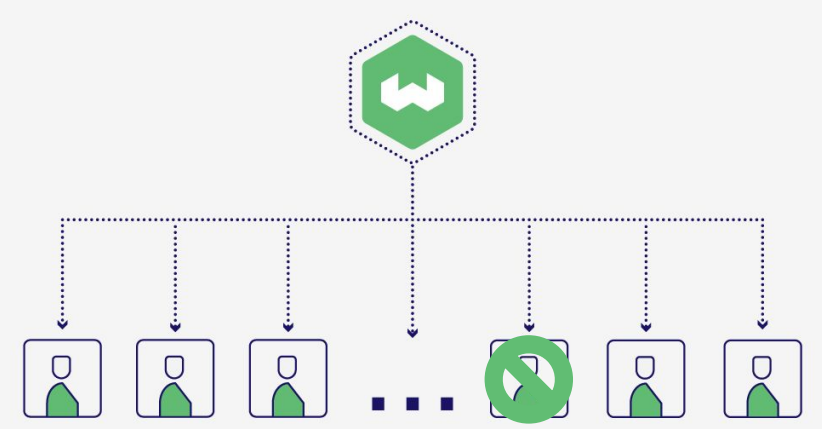


Multi-tenancy implementation

- 1000s per node
- Isolated
- Tenant state management

Scale: Solutions

End user growth

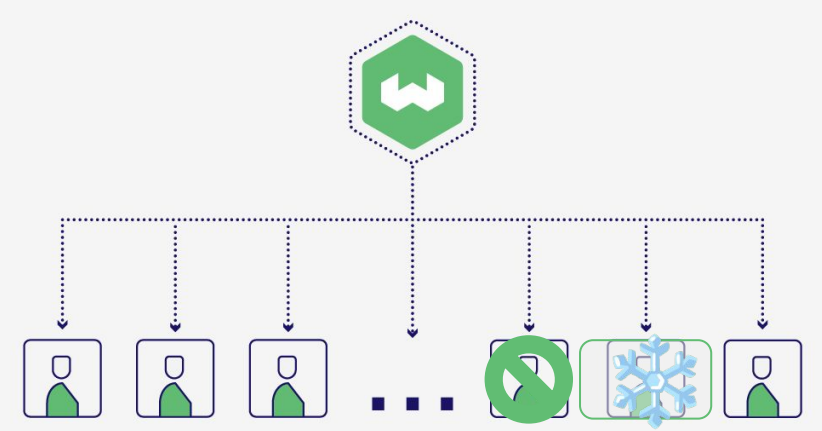


Multi-tenancy implementation

- 1000s per node
- **Isolated** (easy deletion & compliance)
- Tenant state management

Scale: Solutions

End user growth

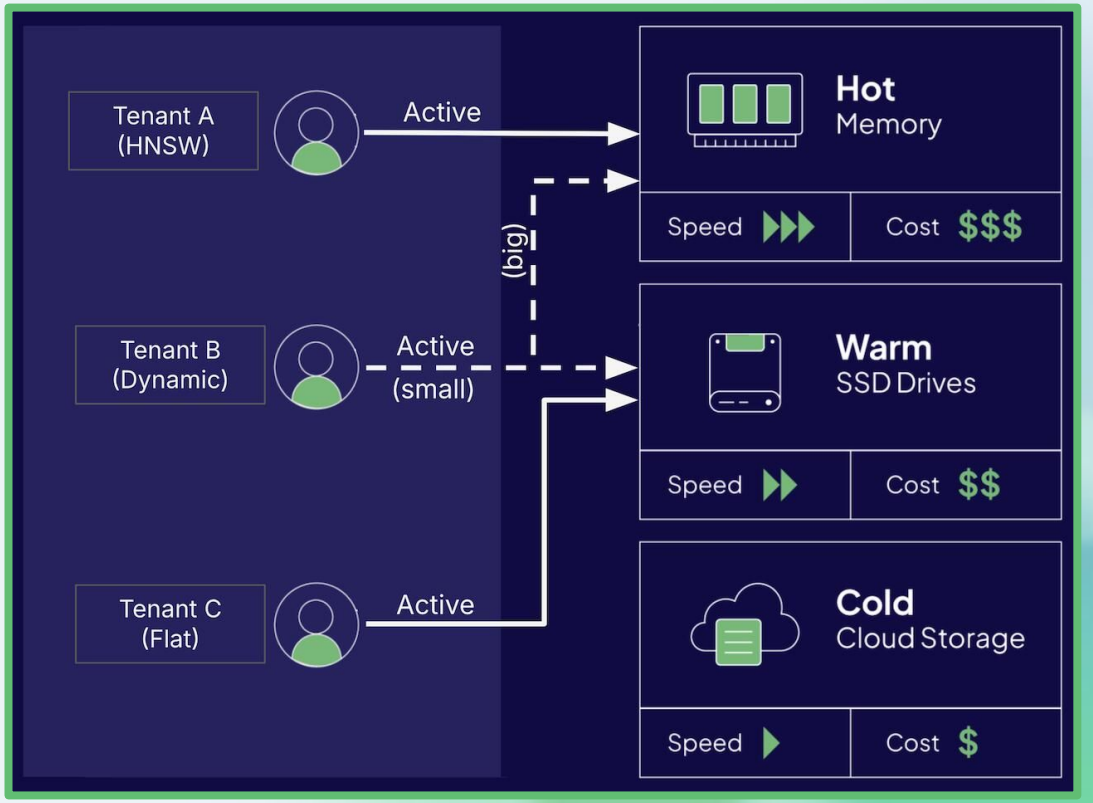


Multi-tenancy implementation

- 1000s per node
- Isolated
- **Tenant state management**
  - Active/inactive/offloaded

Scale: Solutions

Storage tiers



Solutions

Isolation

- Multi-tenancy
  - Isolated shards
    - Shard per end user
  - Easy on/offboarding
  - Manage resource usage

Dealing with reliability

Fault tolerance

Search

Query

returns

Select the company account

AmazonHelp

Limit

5

Search type

Hybrid | Vector | Keyword

Results

For query: `returns`

AmazonHelp: User_280559: We made a return and had previously b...

AmazonHelp: User_123163: if I buy something in black Friday am...

AmazonHelp: User_247957: I didn't know a product didn't have f...

RAG

What should we do with the search results?

What patterns do you recognize in these comments?

Generate response

Example source accounts: **AmazonHelp** (26351), **AppleSupport** (15397), **Uber_Support** (10876), **Delta** (6686), **AmericanAir** (6477), **comcastcares** (5569), **hulu_support** (4249), **SpotifyCares** (6055), **TMobileHelp** (4699), **SouthwestAir** (4931)

Cluster statistics

Object count

200000

Nodes

1

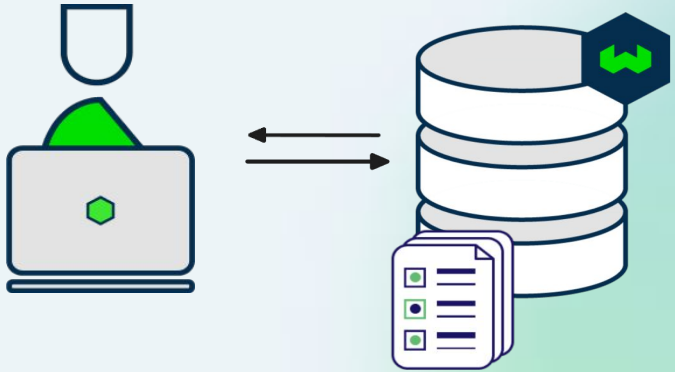
Memory Usage Over Time

Reliability: Considerations

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

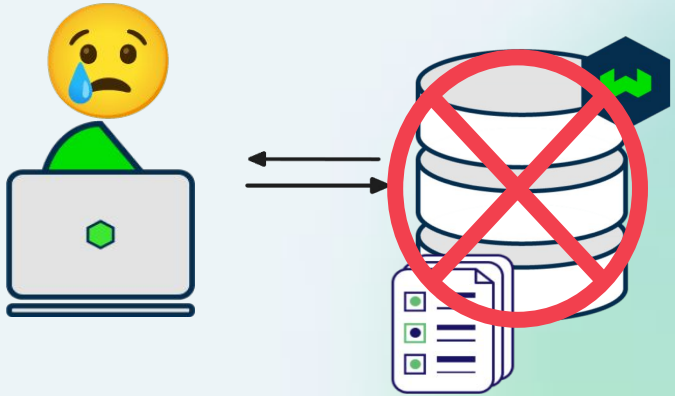
Reliability: Solutions

Happy days



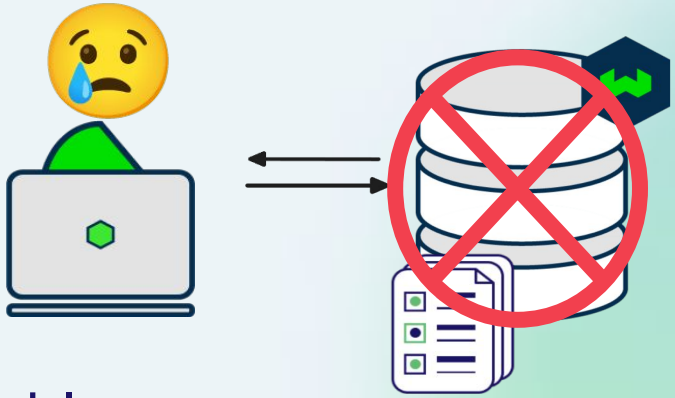
Reliability: Solutions

(Less) Happy days



Reliability: Solutions

(Less) Happy days

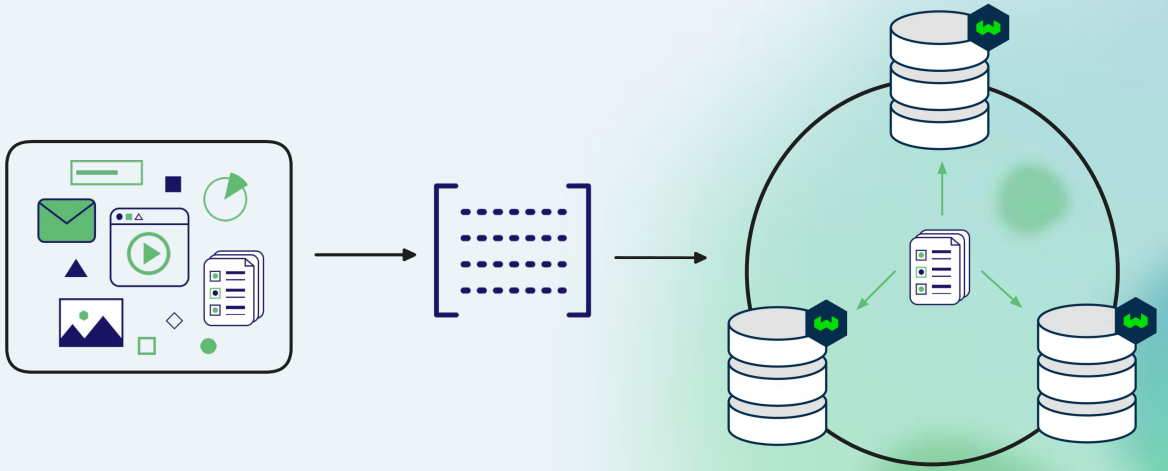


Downtime = inevitable

Reliability: Solutions

Provide redundancy

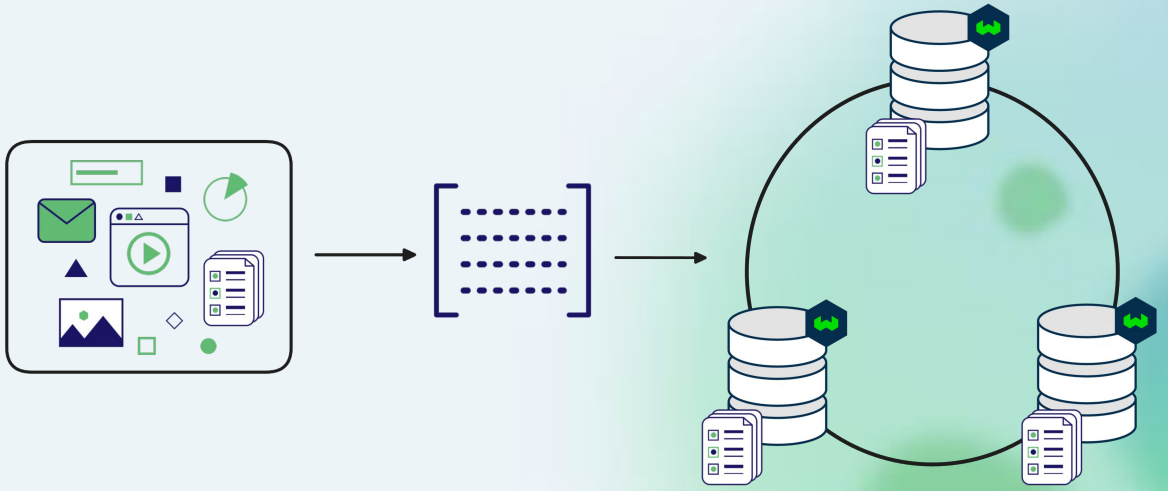
Replication



Reliability: Solutions

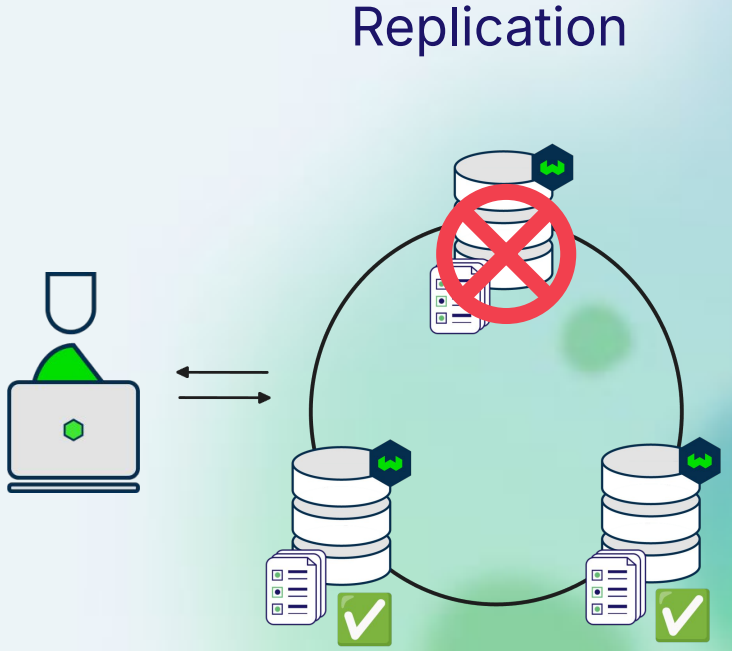
Provide redundancy

Replication



Reliability: Solutions

Provide redundancy



Reliability: Solutions

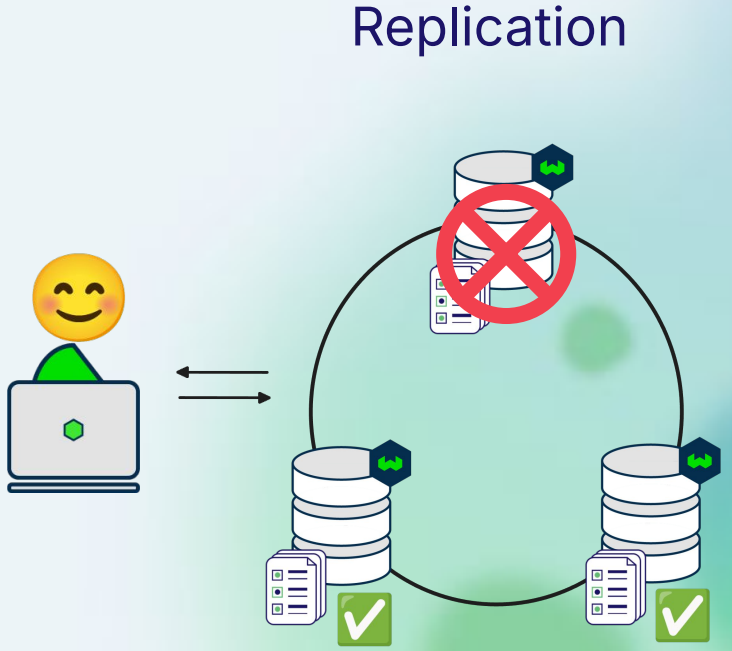
Provide redundancy

Node-level downtime:

- inevitable

System-level downtime:

- avoidable!

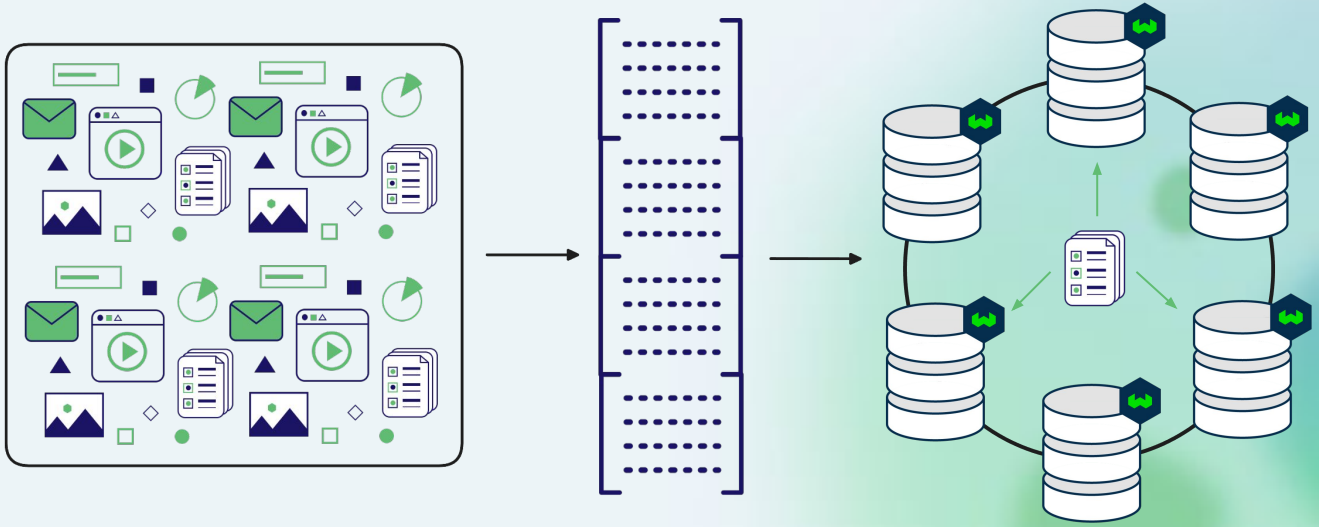


Reliability: Solutions

```
articles = client.collections.create(
    name="Article",
    properties=[
        Property(name="title", data_type=DataType.TEXT)
    ],
    replication_config=Configure.replication(factor=3)
)
```

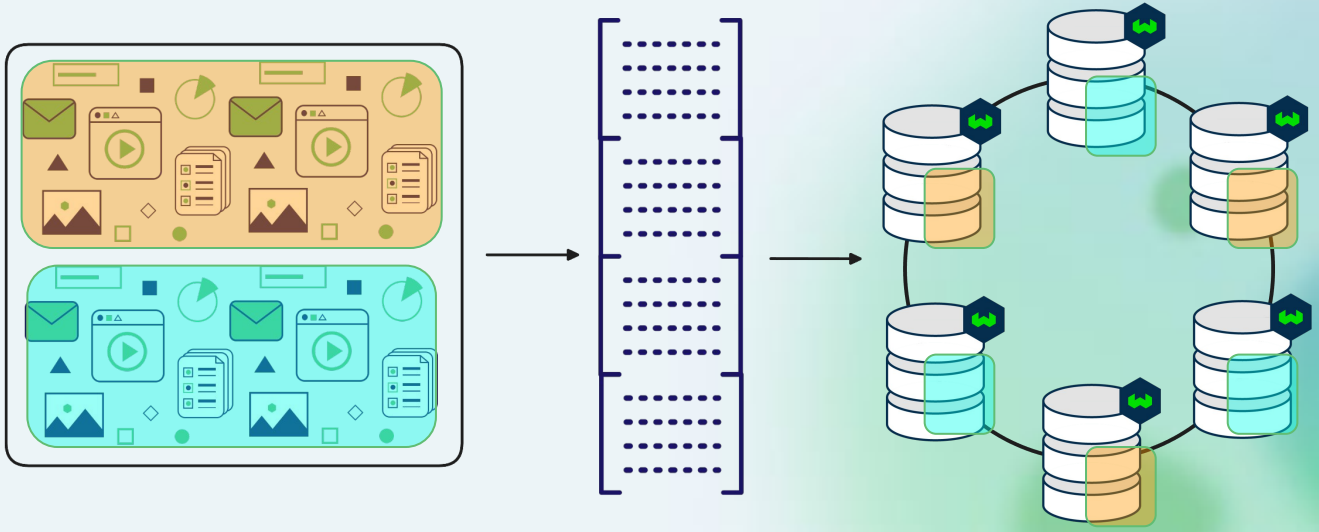
Reliability: Solutions

Sharding + replication



Reliability: Solutions

Sharding + replication



Solutions

Fault tolerance

- Replication
  - Data replication
  - Tunable consistency
  - Combine with multi-tenancy / horizontal scaling

Reliability: Solutions

Backups

Can your DB:

- Perform local & cloud backups?
- Export data easily?

Commitment to open-source

Reliability: Solutions

Backup example

```
result = client.backup.create(
    backup_id="my-very-first-backup",
    backend="filesystem",  # or s3, gcs, azure
    include_collections=["Article", "Publication"],
)
```

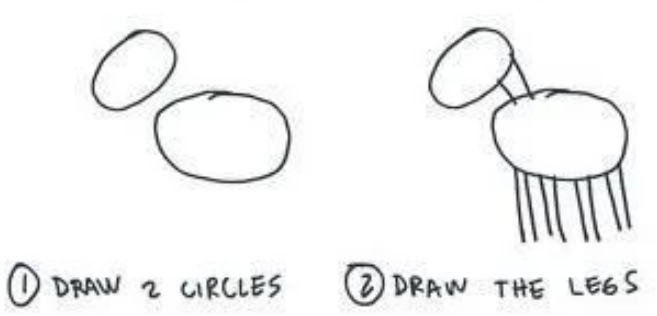
Reliability: Solutions

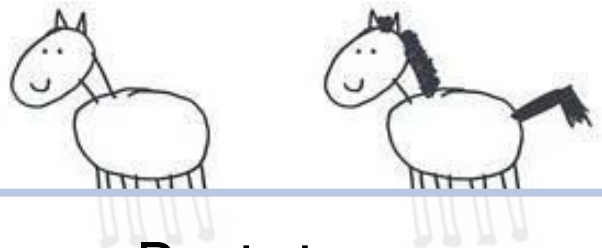
Data migration (export + import) example

```
with tgt.batch.dynamic() as batch:
    for o in src.iterator(include_vector=True):
        batch.add_object(
            properties=o.properties,
            vector=o.vector["default"],
            uuid=o.uuid
        )
```

HOW TO:

Take your prototype app to production





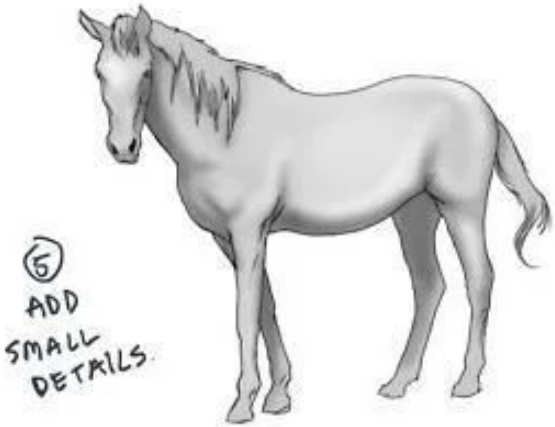
Vector index types

Vector quantization

Sharding

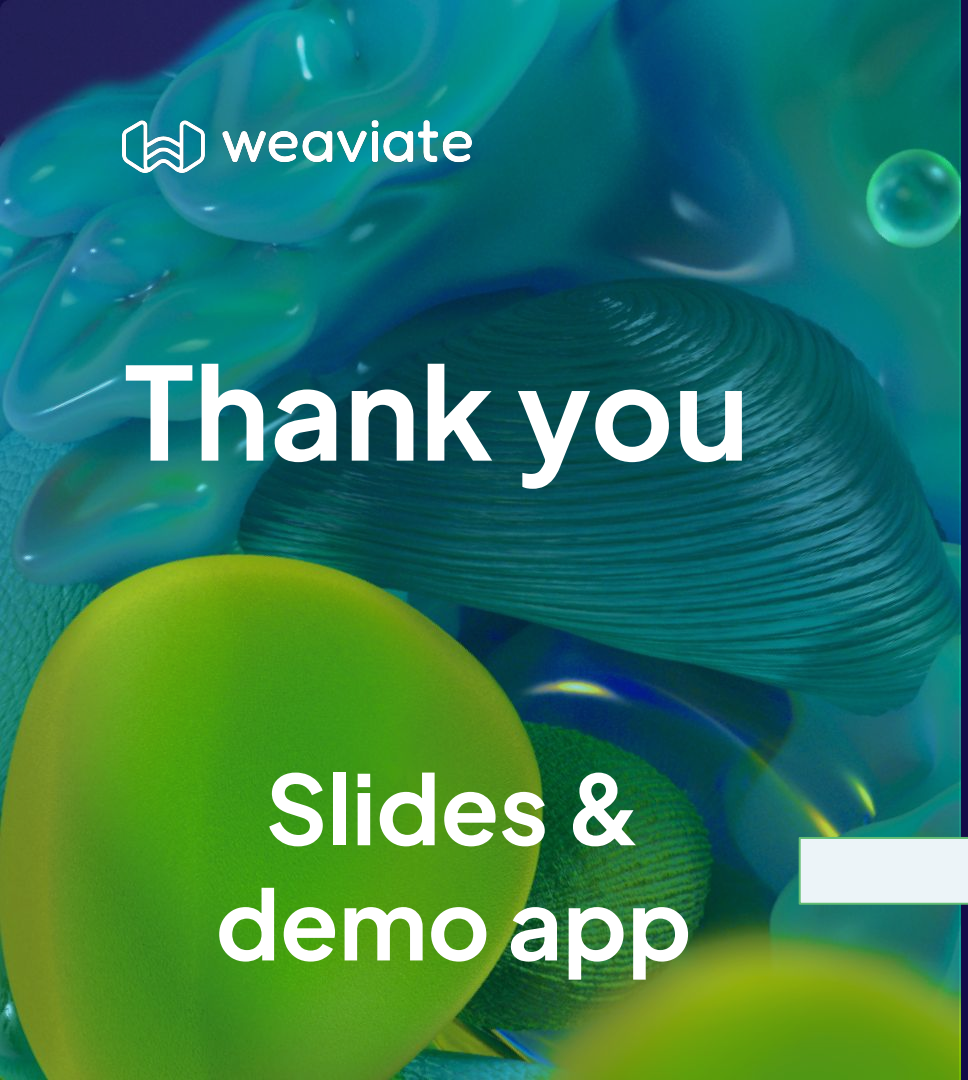
Multi-tenancy

Replication



Prototype

Production



weaviate.io

weaviate/weaviate

jphwang