Graph Databases after 15 Years – Where Are They Headed?

Gábor Szárnyas

Data Analytics devroom | FOSDEM | 2025-02-01

About me

2012–2019
MSc & PhD

graphs

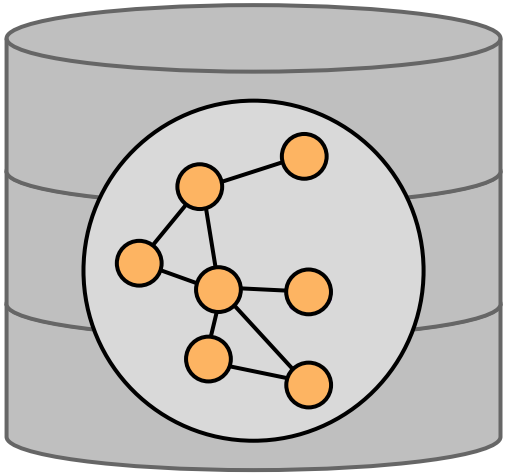
2020–2023
postdoc

graphs

2023–
devrel

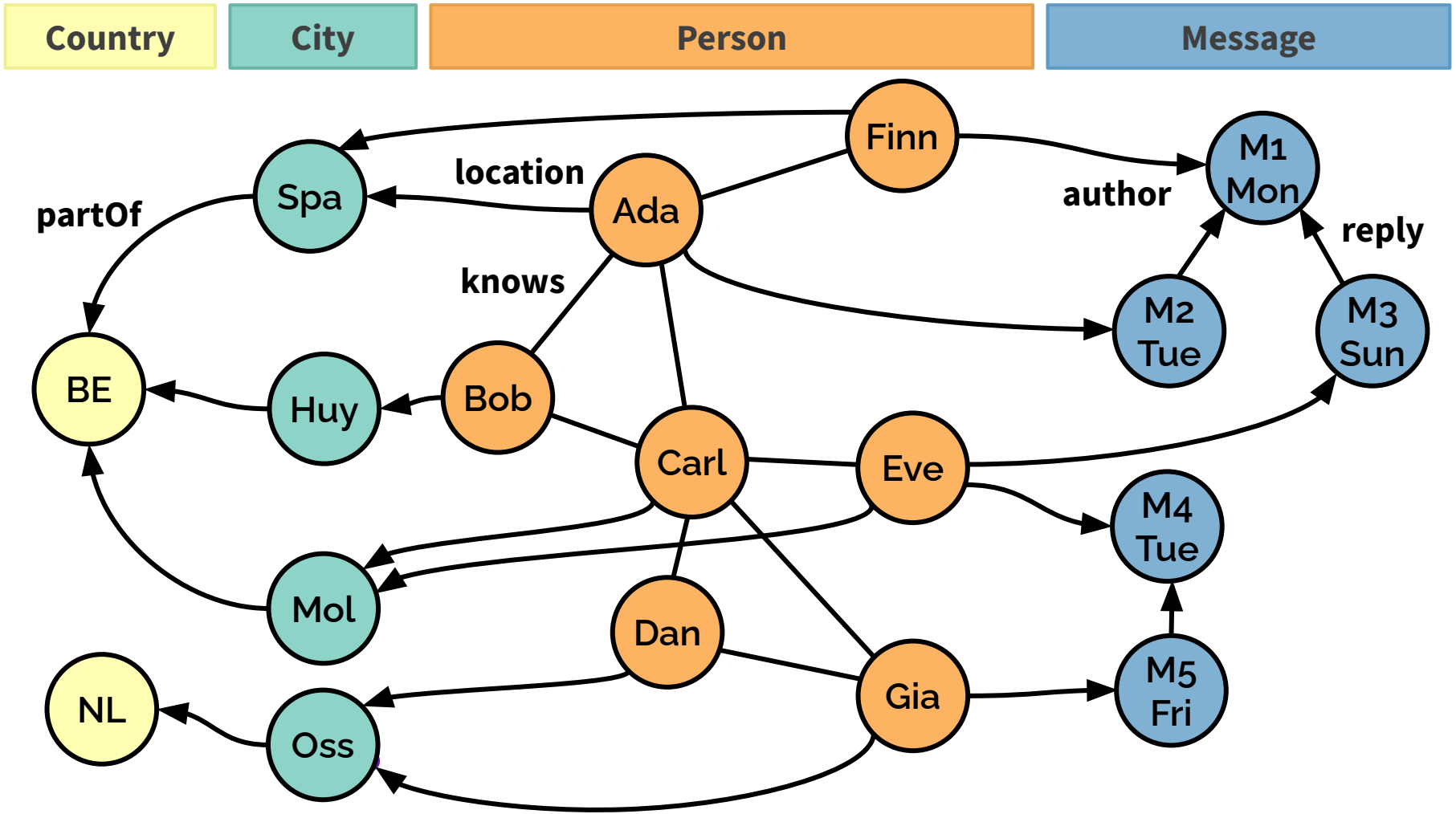
DuckDB Labs

not graphs



1. graph databases in 15 minutes
2. new research – not just for graphs!
3. pointers for benchmarks

Running example


Country
City
Person
Message
Finn
Spa
location
Ada
author
M1
Mon
partOf
reply
knows
M2
Tue
M3
Sun
BE
Huy
Bob
Carl
Eve
M4
Tue
Mol
Dan
M5
Fri
NL
Oss
Gia

Graph databases

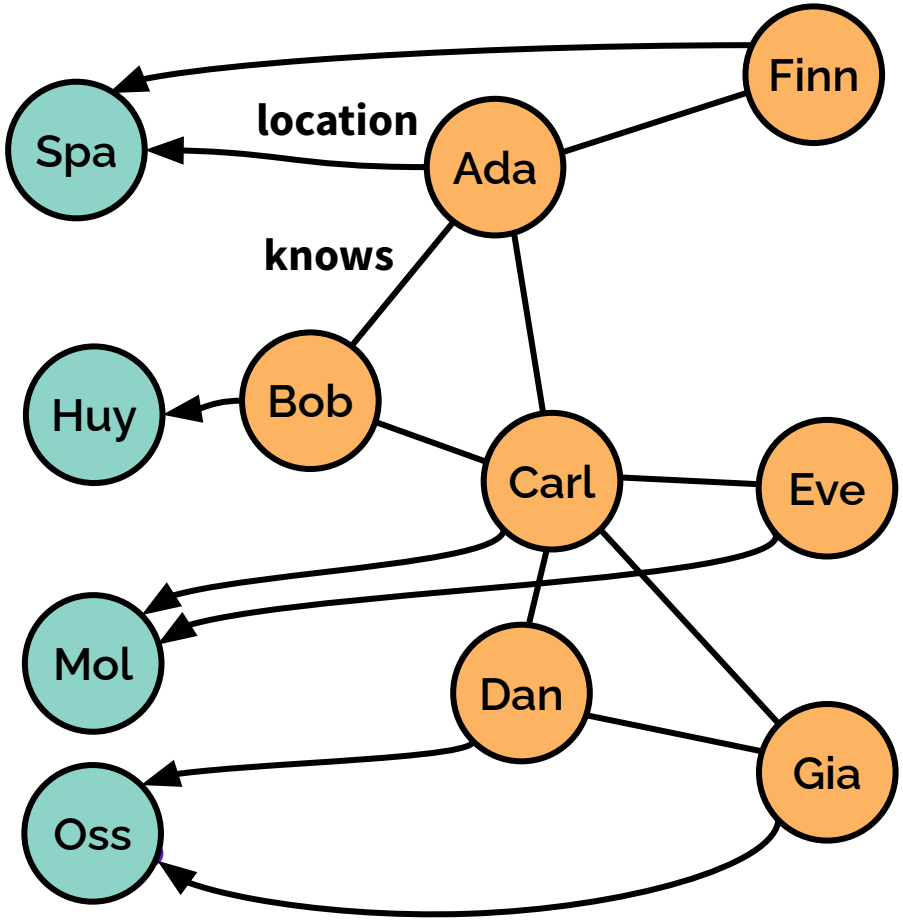
Fast data processing

No expensive run-time JOINs.

Join those who left JOINs behind

Relational databases can't join?

Where do Dan's friends live?



knows

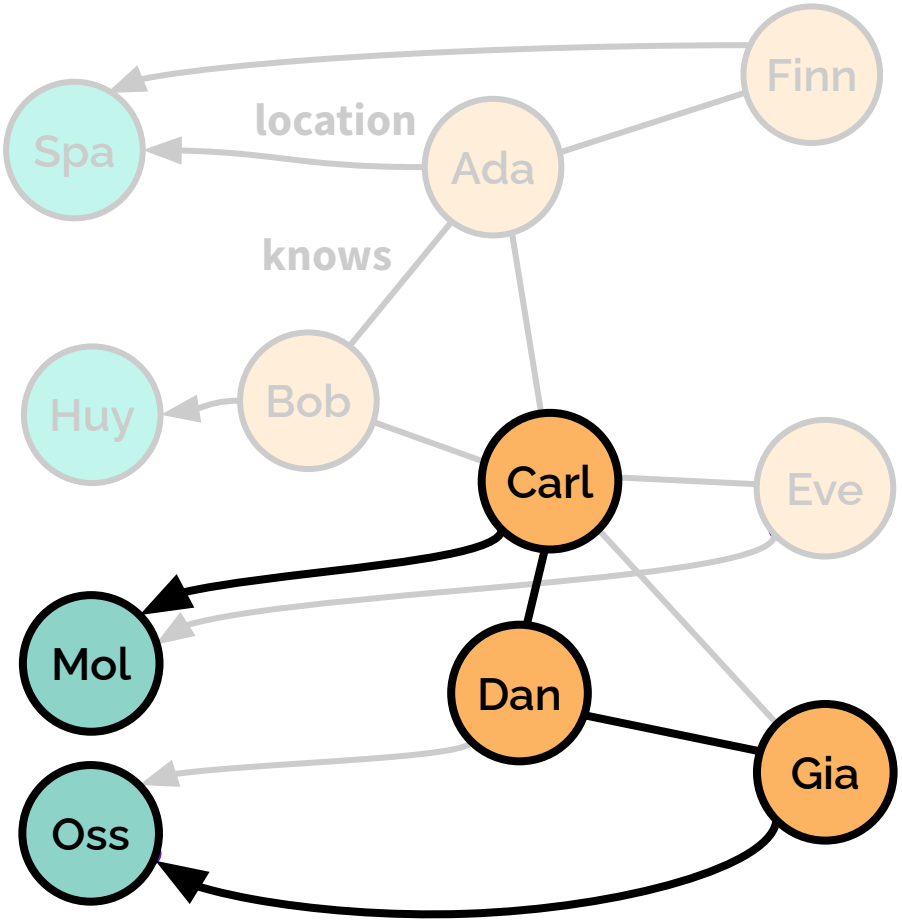
p1	p2
Ada	Bob
Ada	Carl
Ada	Finn
Bob	Carl
Carl	Dan
Carl	Eve
Carl	Gia
Dan	Gia

⋈

location

person	city
Ada	Spa
Bob	Huy
Carl	Mol
Dan	Oss
Eve	Mol
Finn	Spa
Gia	Oss

Where do Dan's friends live?



knows

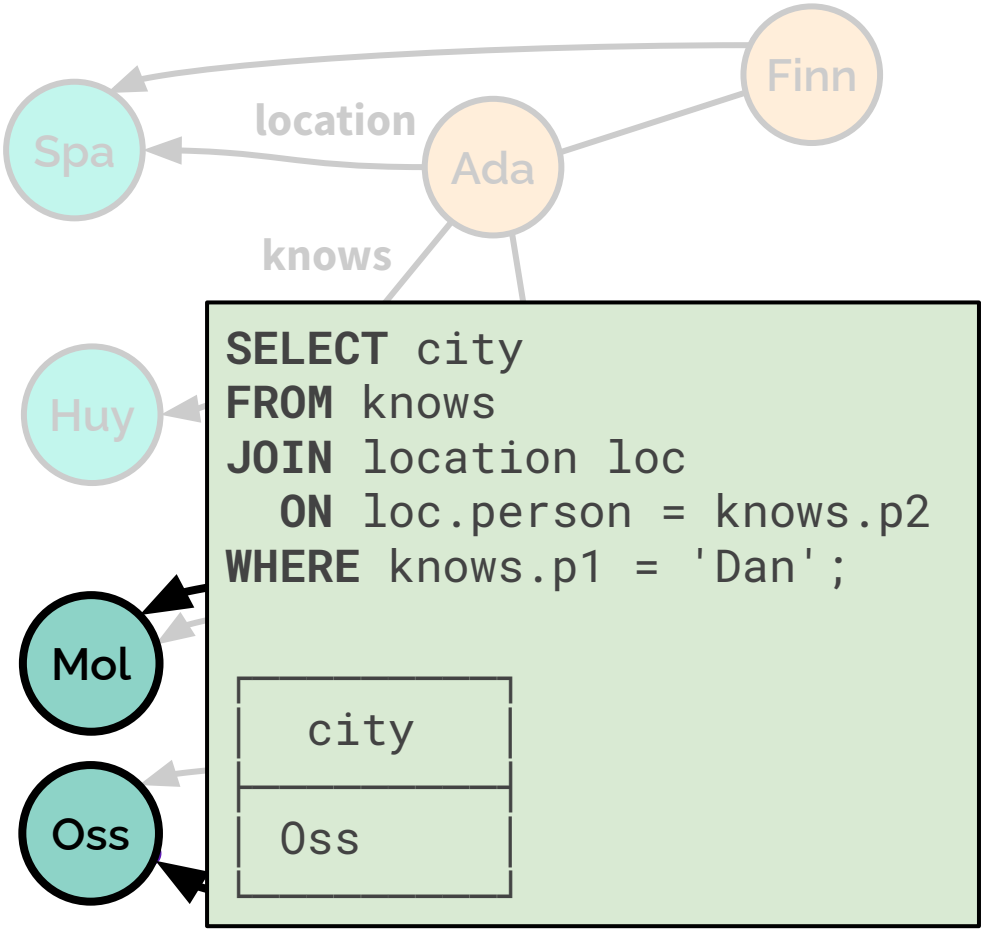
p1	p2
Ada	Bob
Ada	Carl
Ada	Finn
Bob	Carl
Carl	Dan
Carl	Eve
Carl	Gia
Dan	Gia

⋈

location

person	city
Ada	Spa
Bob	Huy
Carl	Mol
Dan	Oss
Eve	Mol
Finn	Spa
Gia	Oss

Where do Dan's friends live?




```
SELECT city
FROM knows
JOIN location loc
  ON loc.person = knows.p2
WHERE knows.p1 = 'Dan';
```

city
Oss

knows

p1	p2
Ada	Bob
Ada	Carl
Ada	Finn
Bob	Carl
Carl	Dan
Carl	Eve
Carl	Gia
Dan	Gia

⋈

location

person	city
Ada	Spa
Bob	Huy
Carl	Mol
Dan	Oss
Eve	Mol
Finn	Spa
Gia	Oss

Where do Dan's friends liv


```
SELECT city
FROM knows
JOIN location loc
  ON loc.person = knows.p2
WHERE knows.p1 = 'Dan';
```

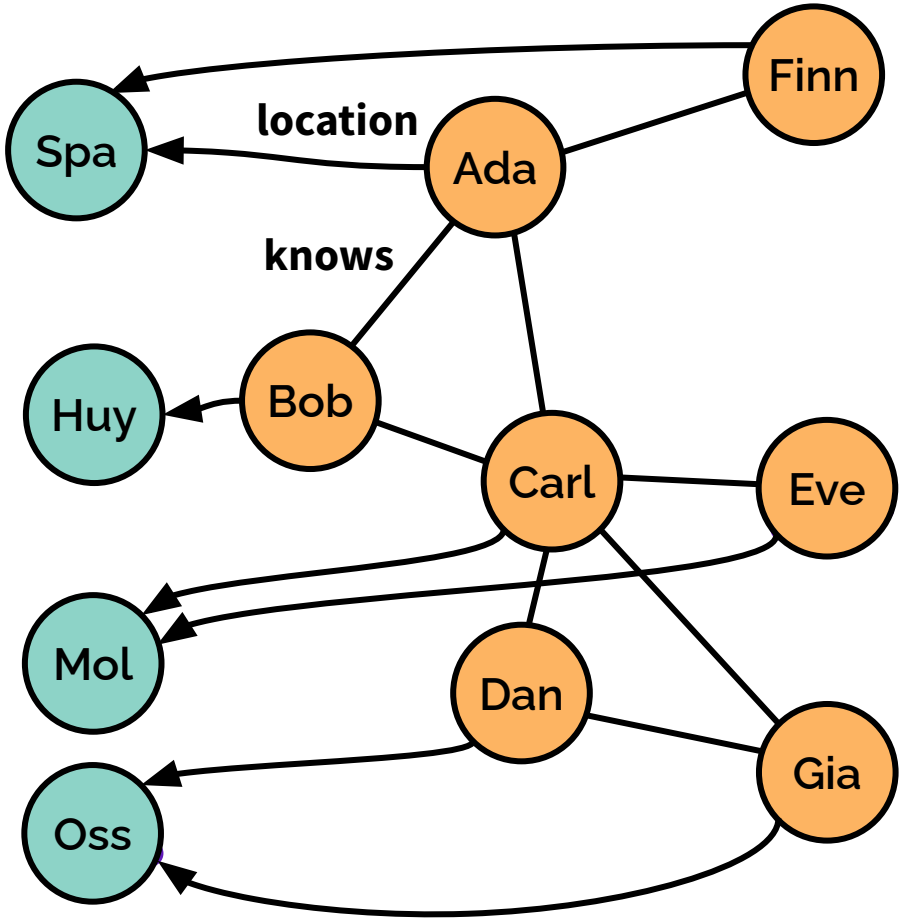
city
Oss

```
CREATE VIEW knows_undir AS
  SELECT p1, p2 FROM knows
  UNION ALL
  SELECT p2, p1 FROM knows;

SELECT city
FROM knows_undir ku
JOIN location loc
  ON loc.person = ku.p2
WHERE ku.p1 = 'Dan';
```

city varchar
Mol
Oss

Where do Dan's friends live?



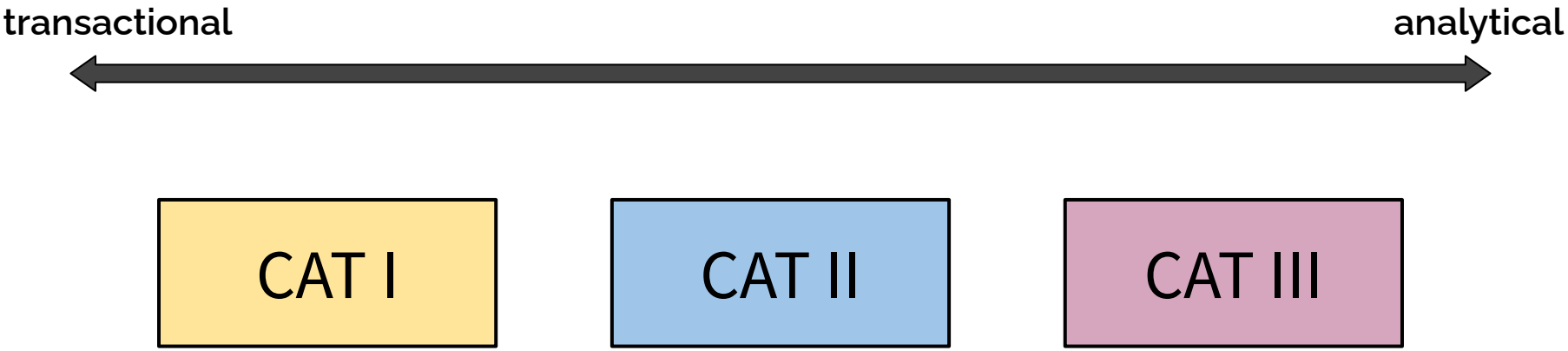
Cypher query language

```
MATCH (p1 {name: 'Dan'})
        -[:knows]-(p2)
        -[:location]->(c)
RETURN c.name;
```

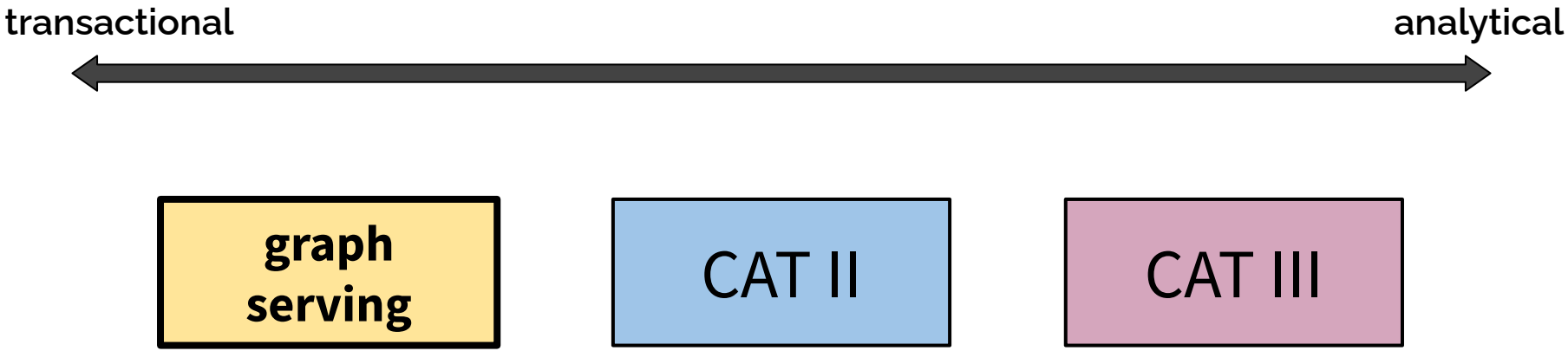
Two advantages:

- concise and readable joins
- elegant handling of bidirectional edges

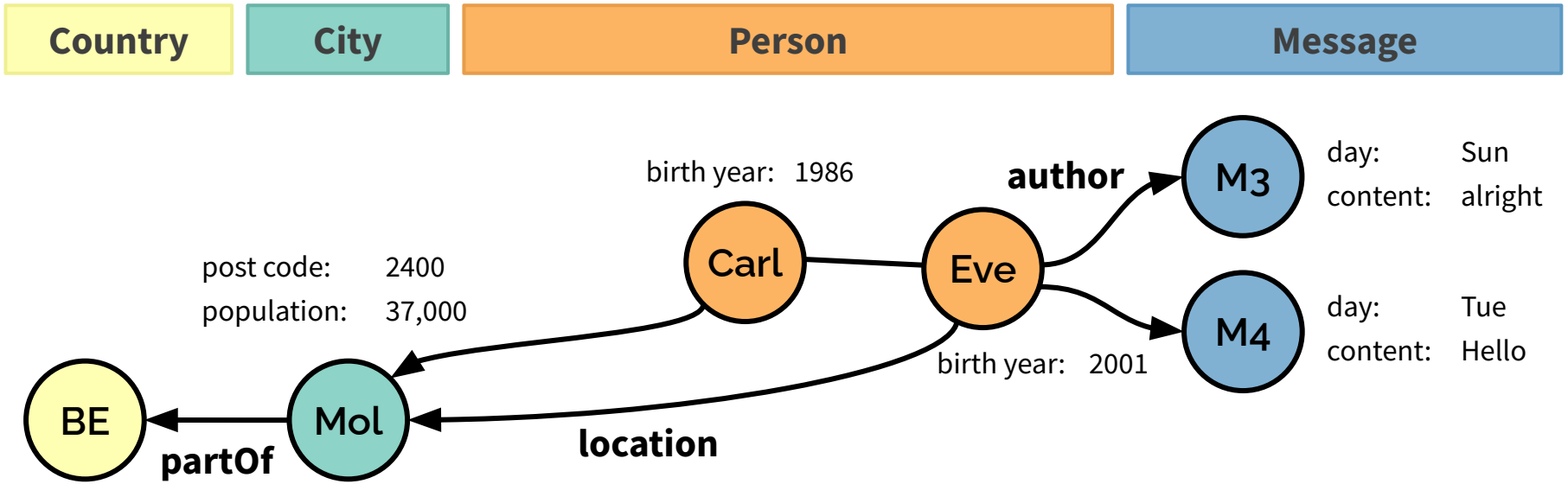
Most graph databases have some special syntax sugar for joins.



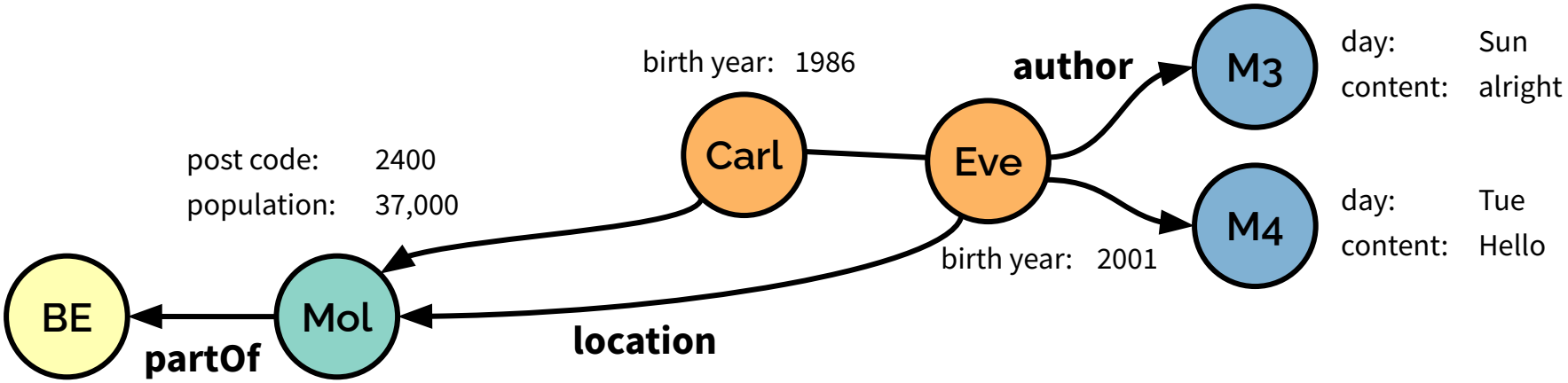
all three categories address join problems



repeated joins =
n + 1 query problem


Country
City
Person
Message
birth year: 1986
author
day: Sun
content: alright
post code: 2400
population: 37,000
Carl
Eve
M3
M4
day: Tue
content: Hello
birth year: 2001
BE
Mol
partOf
location

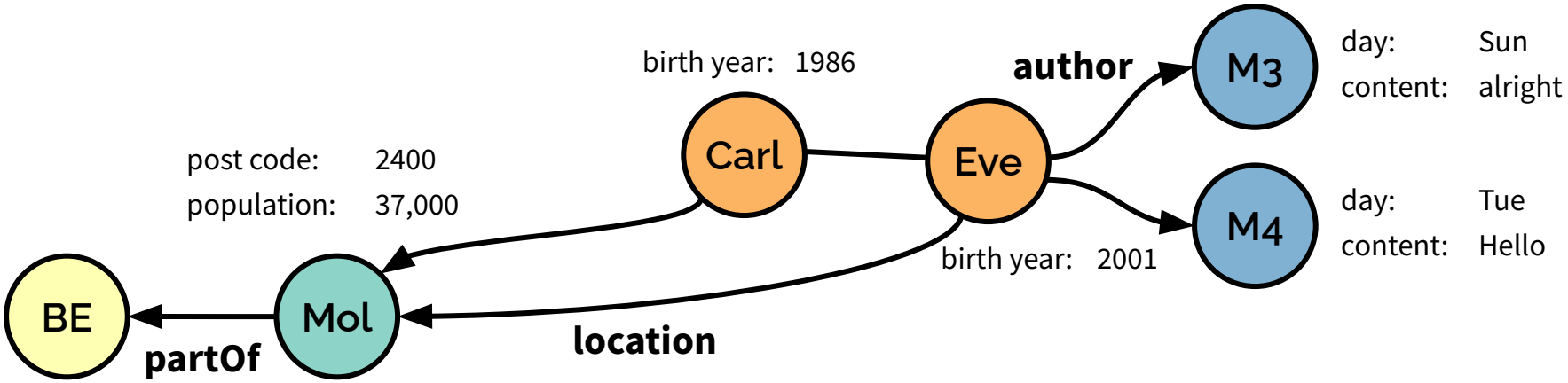
Country | City | Person | Message



partOf ⋈ City ⋈ location ⋈ Person ⋈ author ⋈ Message (left joins + filtering)

city name	post code	population	person name	birth year	message id	day	comment
Mol	2400	37,000	Carl	1986	NULL	NULL	NULL
Mol	2400	37,000	Eve	2001	M3	Sun	alright
Mol	2400	37,000	Eve	2001	M4	Tue	Hello

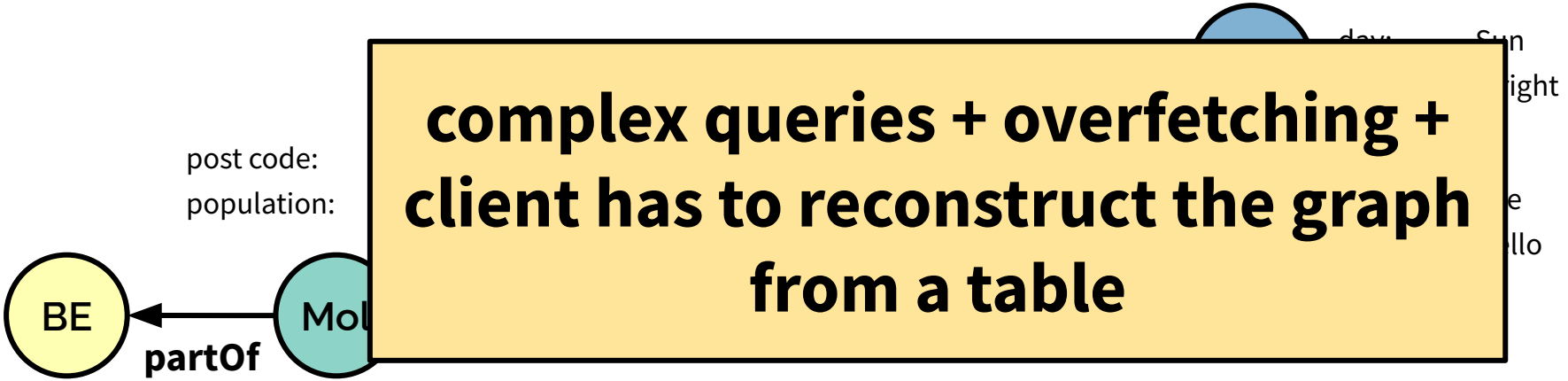
Country | City | Person | Message



partOf ⋈ City ⋈ location ⋈ Person ⋈ author ⋈ Message (left joins + filtering)

city name	post code	population	person name	birth year	message id	day	comment
Mol	2400	37,000	Carl	1986	NULL	NULL	NULL
Mol	2400	37,000	Eve	2001	M3	Sun	alright
Mol	2400	37,000	Eve	2001	M4	Tue	Hello

Country | City | Person | Message

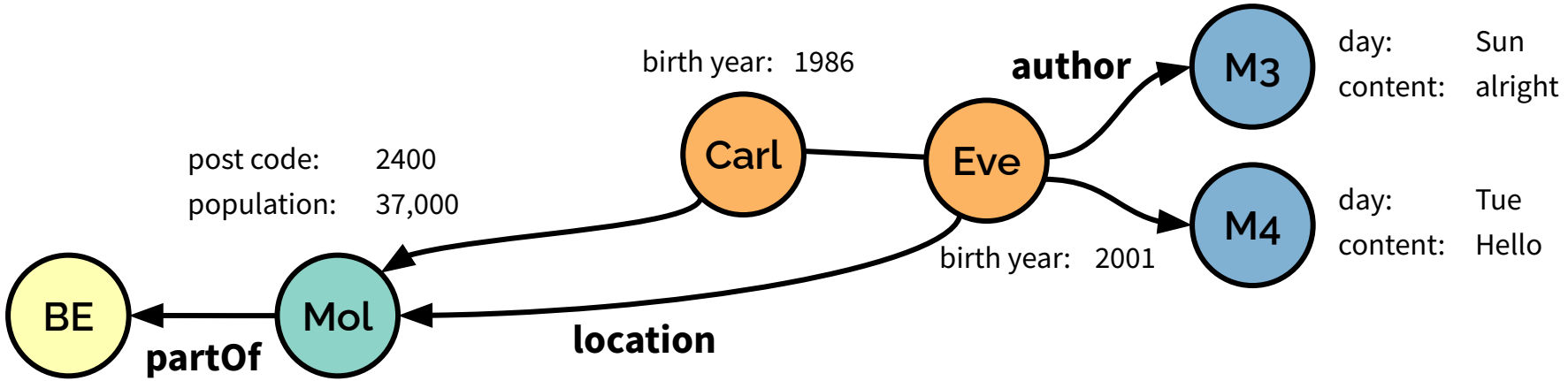


complex queries + overfetching + client has to reconstruct the graph from a table

partOf ⋈ City ⋈ location ⋈ Person ⋈ author ⋈ Message (left joins + filtering)

city name	post code	population	person name	birth year	message id	day	comment
Mol	2400	37,000	Carl	1986	NULL	NULL	NULL
Mol	2400	37,000	Eve	2001	M3	Sun	alright
Mol	2400	37,000	Eve	2001	M4	Tue	Hello

Country | City | Person | Message

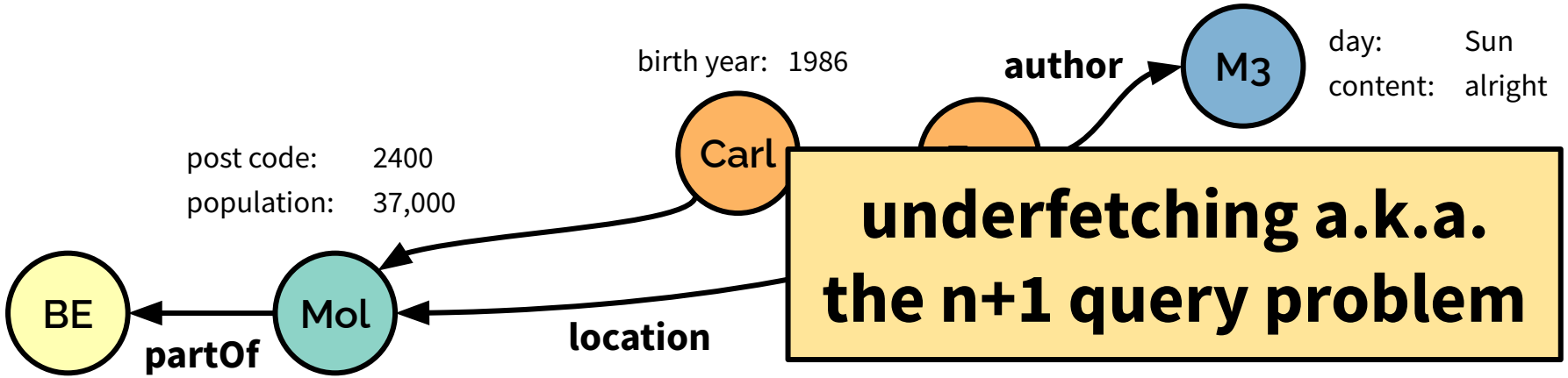


city name	post code	population
Mol	2400	37,000

city name	person name	birth year
Mol	Carl	1986
Mol	Eve	2001

person name	message id	day	content
Eve	M3	Sun	alright
Eve	M4	Sun	Hello

Country | City | Person | Message



city name	post code	population
Mol	2400	37,000

city name	person name	birth year
Mol	Carl	1986
Mol	Eve	2001

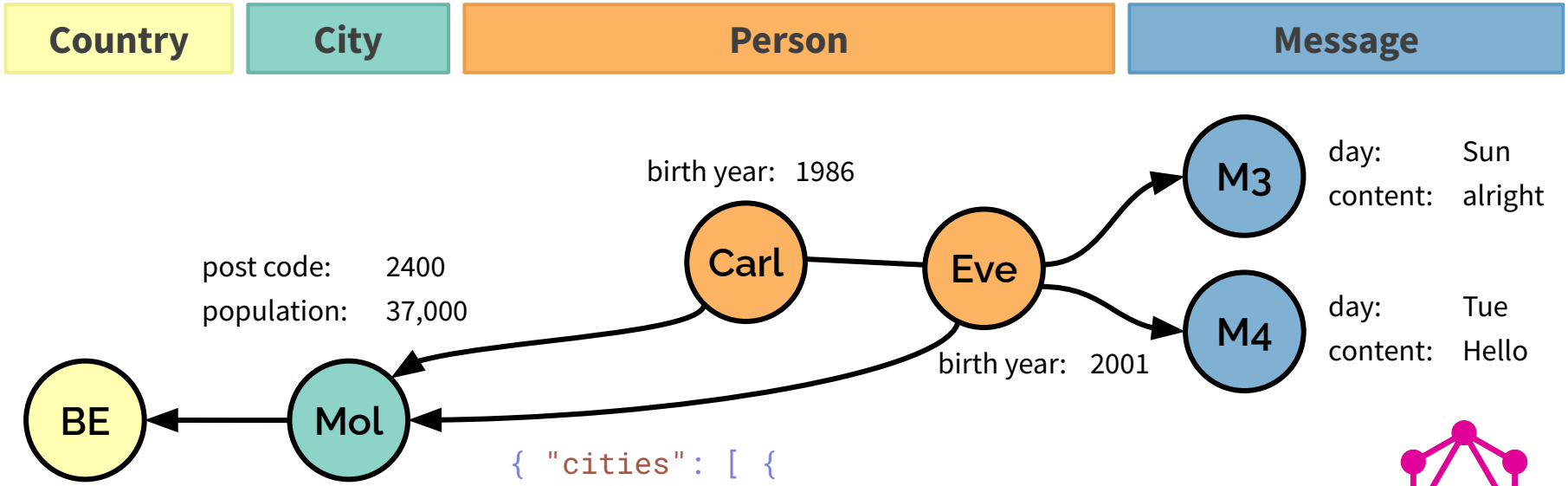
person name	message id	day	content
Eve	M3	Sun	alright
Eve	M4	Sun	Hello

Why is querying a graph difficult?

Problems:

- *overfetching* is expensive, introduces redundancy
- *underfetching* is slow and leads to the n+1 query problem

These are due to the **impedance mismatch from ORM** (object-relational mapping)




```
query social_network {
  city {
    name
    resident {
      message {
        id, day, content
      }
    }
  }
}
```

```
{ "cities": [ {
    "name": "Mol",
    "residents": [
      { "name": "Carl" },
      { "name": "Eve",
        "messages": [
          { "id": "M3", "day": "Sun", "content": "alright" },
          { "id": "M4", "day": "Tue", "content": "Hello" }
        ]
      }
    ]
} ] }
```

GraphQL

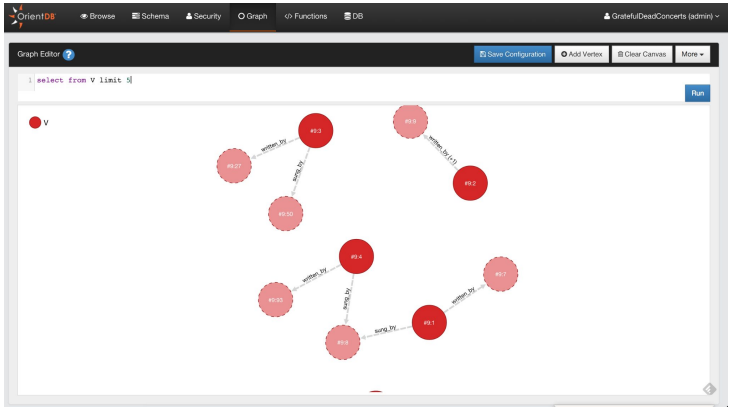
OrientDB & co.

2010

SQL dialect, Gremlin

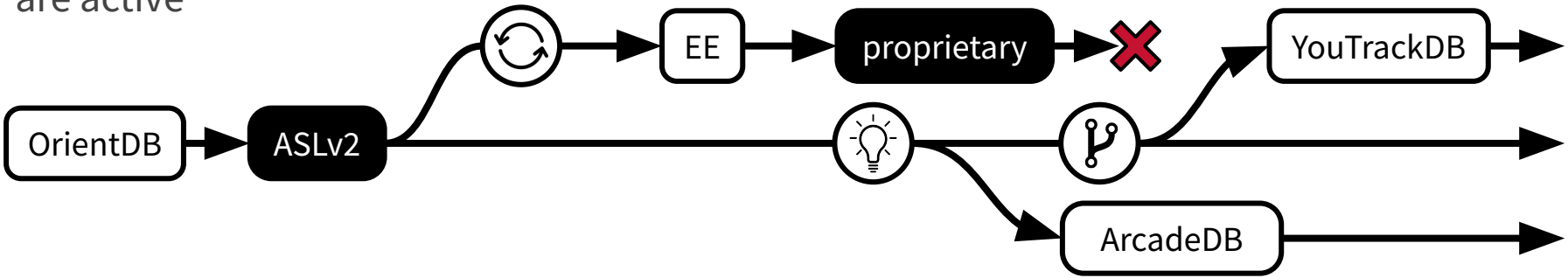
User-friendly SQL dialect:

```
SELECT * FROM message m
WHERE m.author.city.country = 'BE';
```



SAP acquired OrientDB, then abandoned it

Its open-source repository and two of its forks are active



Microsoft Cosmos DB	2017	SQL dialect, Gremlin, etc.

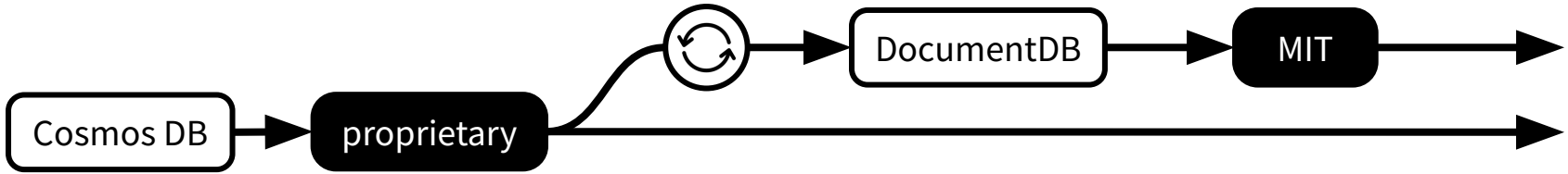
Distributed, schemaless NoSQL graph database

Powers ChatGPT in Azure

Also offers a SQL-like language, Gremlin, and a MongoDB-compatible API

NoSQL

```
SELECT VALUE {
    "name": CONCAT(e.name.last, " ", e.name.first),
    "department": e.department.name,
    "emailAddresses": [
        e.email
    ]
}
FROM
    employees e
WHERE
    STRINGEQUALS(e.department.name, "logistics", true)
```



Gremlin

AQL

DQL (≈ GraphQL)

LinkedIn LIquid

Datalog

SQL, GraphQL

Datalog, WOQL
(≈ GraphQL)

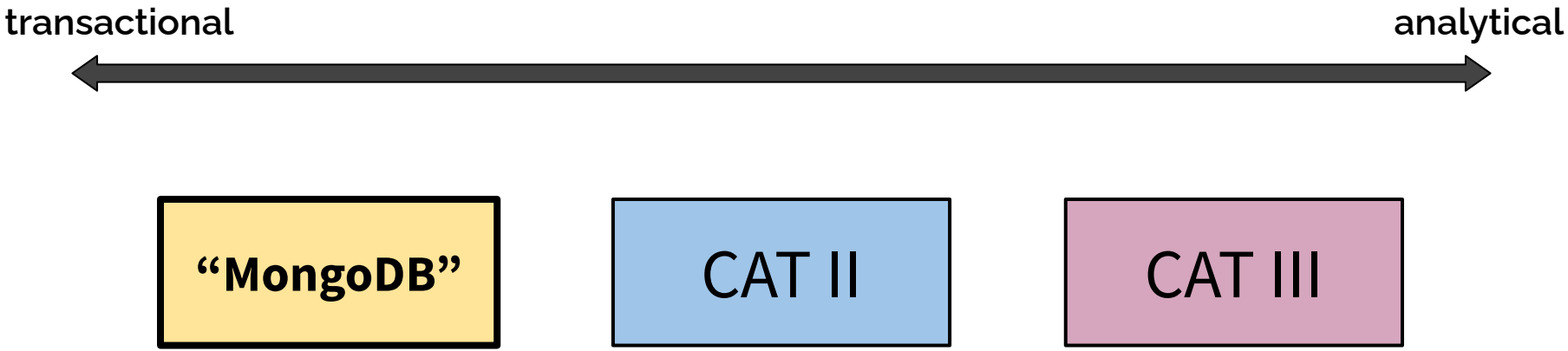
Hints for spotting graph serving systems

A graph database is likely a **graph serving** system if it is:

- backed by a key-value / document store
- called a “real-time graph database”
- categorized under “multi-model” graph database

Rank			DBMS	Database Model	Score	
Jan 2025	Dec 2024	Jan 2024			Jan 2025	Dec 2024
1.	1.	1.	Neo4j	Graph	43.69	+0.62
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model		
3.	3.	3.	Aerospike	Multi-model		
4.	4.	4.	Virtuoso	Multi-model		
5.	5.	5.	ArangoDB	Multi-model		
6.	6.	6.	OrientDB	Multi-model		
7.	7.	↑ 10.	GraphDB	Multi-model	2.72	-0.06
8.	8.	↓ 7.	Memgraph	Graph	2.62	-0.08

Document store
Graph DBMS
Key-value store
Wide column store
Spatial DBMS



repeated joins =
n + 1 query problem

“MongoDB”

repeated joins =
n + 1 query problem

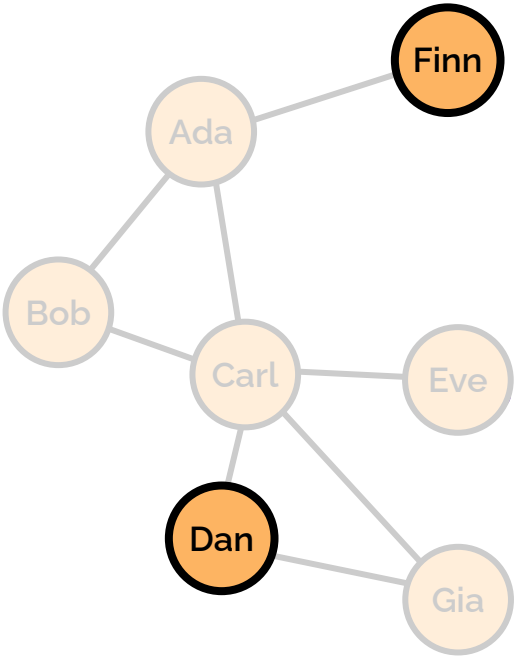
classic graph database

recursive joins =
path queries

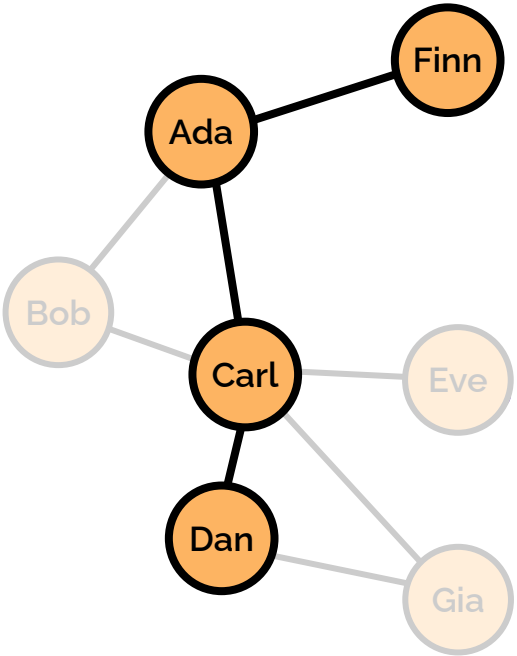
CAT III

Shortest path finding

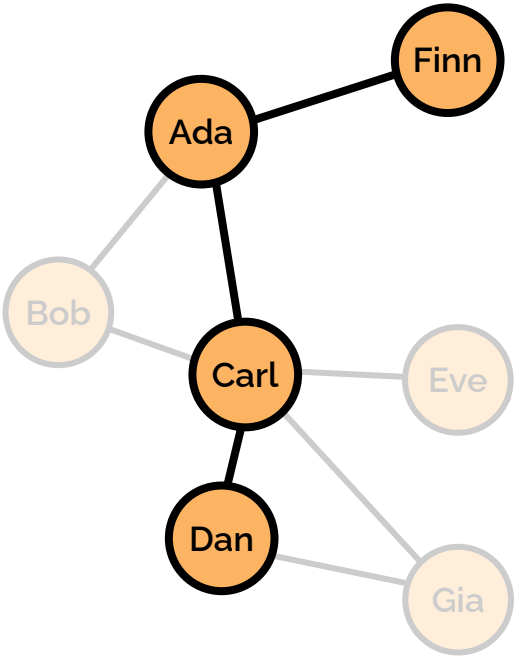
Shortest path finding



Shortest path finding



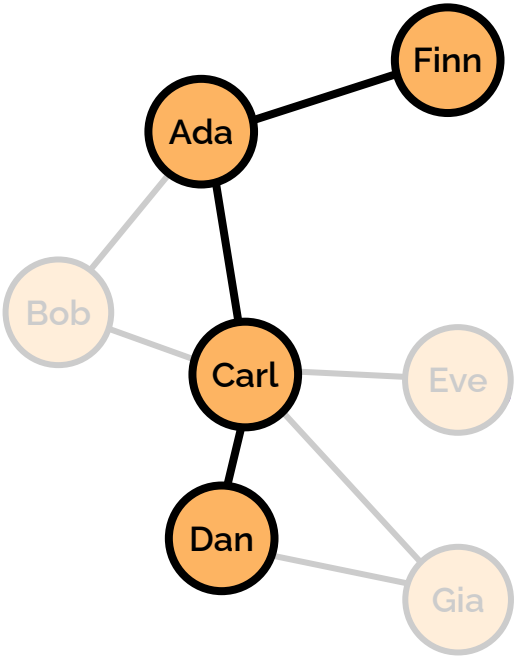
Shortest path finding in SQL:1999




```
WITH RECURSIVE paths(endNode, path, ended) AS (
  SELECT p2 AS endNode, [p1, p2] AS path,
    (p2 = 'Dan') AS ended FROM knows WHERE p1 = 'Finn'
  UNION ALL
  SELECT p2 AS endNode, array_append(path, p2) AS path,
    max(p2 = 'Dan') OVER (ROWS BETWEEN UNBOUNDED PRECEDING
      AND UNBOUNDED FOLLOWING) AS ended
    FROM paths JOIN knows ON paths.endNode = p1
   WHERE NOT EXISTS (SELECT true FROM paths previous_paths
     WHERE list_contains(previous_paths.path, p2))
       AND NOT paths.ended)
SELECT path FROM paths WHERE endNode = 'Dan';

-- Result: [Finn, Ada, Carl, Dan]
```

Shortest path finding in Cypher

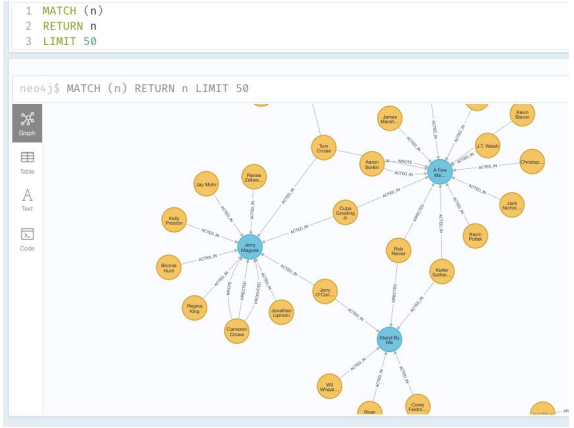



```
MATCH
  p = shortestPath(
    (:Person { name: 'Finn' })-[:knows*]-
    (:Person { name: 'Dan' })
  )
RETURN p;
```

Neo4j | 2007 | Cypher

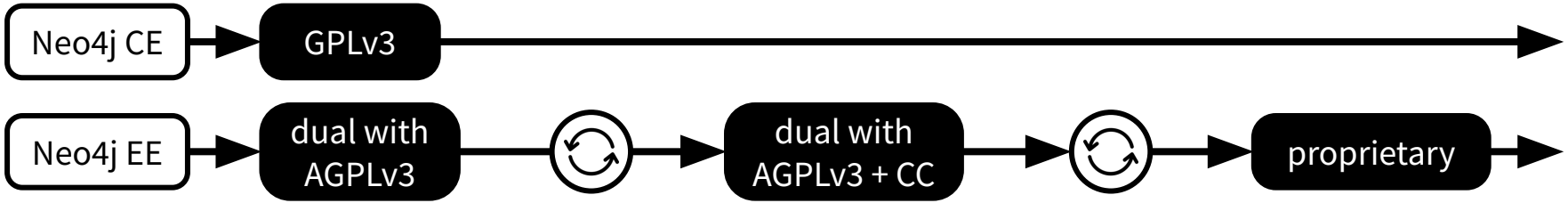
The first modern graph database

- Cypher query language → openCypher → GQL
- Advocacy work: books, meetups, FOSDEM devrooms



Leans into analytics

- Analytics suite
- Parallel runtime in EE

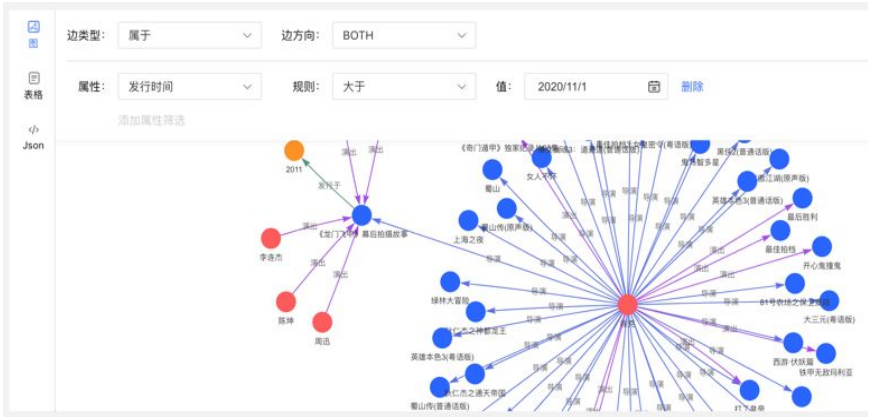


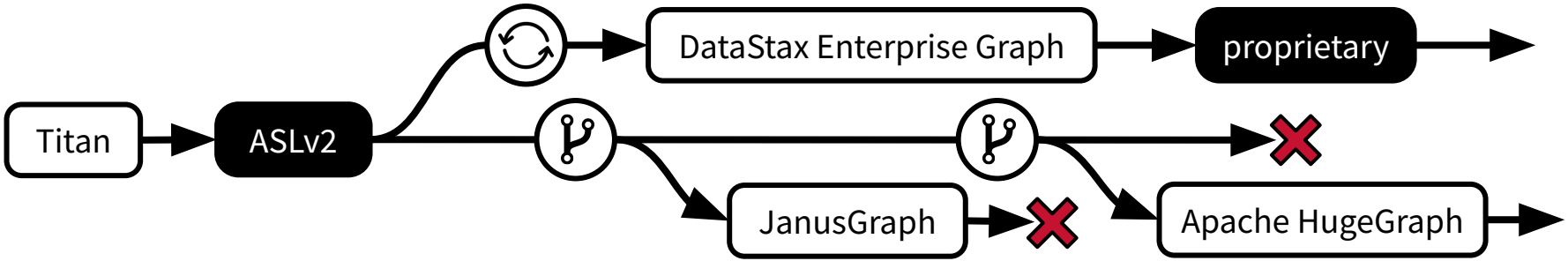
Titan / JanusGraph / HugeGraph | 2012 | Gremlin

Distributed graph database

Storage layer: RocksDB, Cassandra, MySQL / PostgreSQL, etc.

Workload: highly transactional setups





Amazon Neptune

Cypher, Gremlin, SPARQL

GALAXYBASE

Cypher, Gremlin

GraphScope

Gremlin

MEMGRAPH

Cypher

NebulaGraph

Cypher, nGQL, GQL

Google Cloud Spanner Graph

SQL/PGQ, GQL

ULTIPA

UQL, GQL

Cypher, Gremlin, SPARQL

Cypher, Gremlin

Gremlin

Cypher

Cypher, nGQL, GQL

SQL/PGQ, GQL

UQL, GQL

Extensions for RDBMS

- IBM Db2 Graph: Gremlin transpiled to SQL
- PostgreSQL AGE extension: openCypher transpiled to SQL
- PostgreSQL: SQL/PGQ (WIP)
- Oracle Database 23ai: SQL/PGQ
- Microsoft SQL Server Graph: SQL/PGQ-like custom syntax
- SAP HANA Graph: openCypher for queries, GraphScript for algorithms

transactional

analytical

“MongoDB”

repeated joins =
n + 1 query problem

“Postgres”

recursive joins =
path queries

CAT III

“MongoDB”

repeated joins =
n + 1 query problem

“Postgres”

recursive joins =
path queries

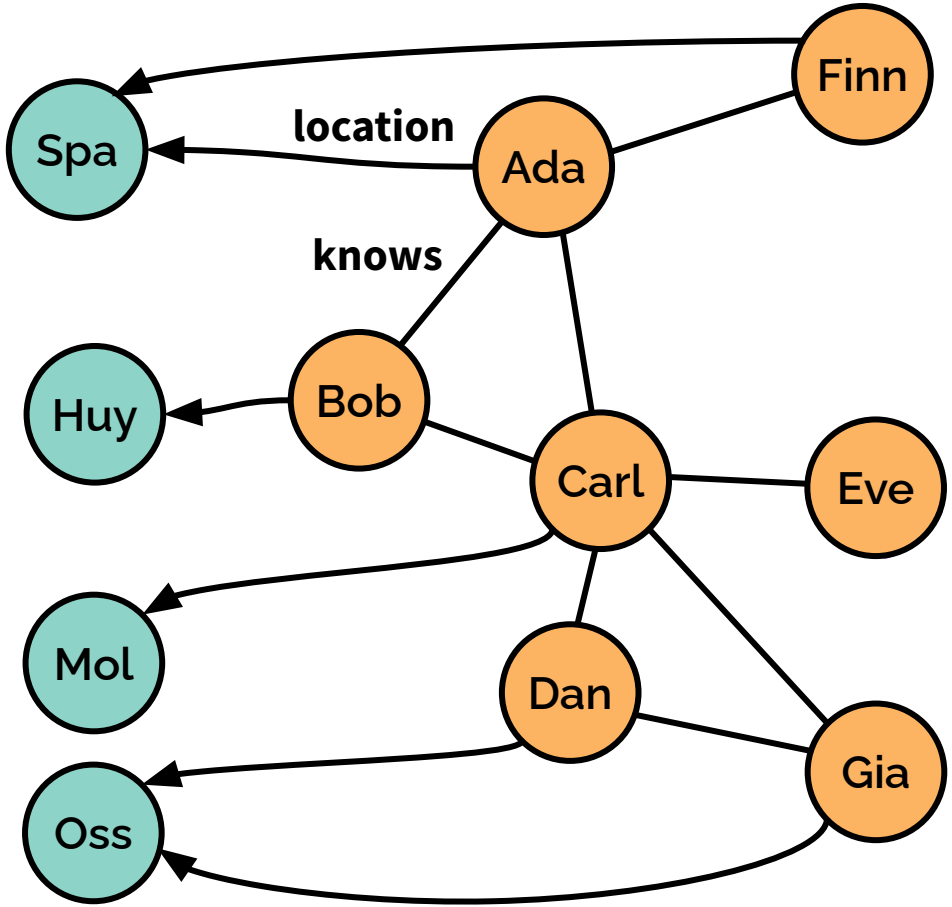
graph BI

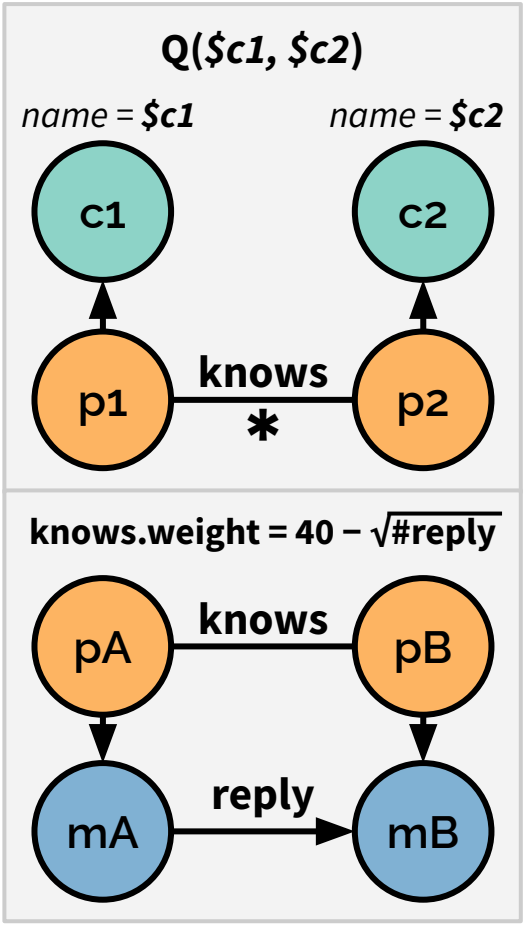
many-to-many joins =

complex recursive joins
cyclic graph patterns
long acyclic patterns

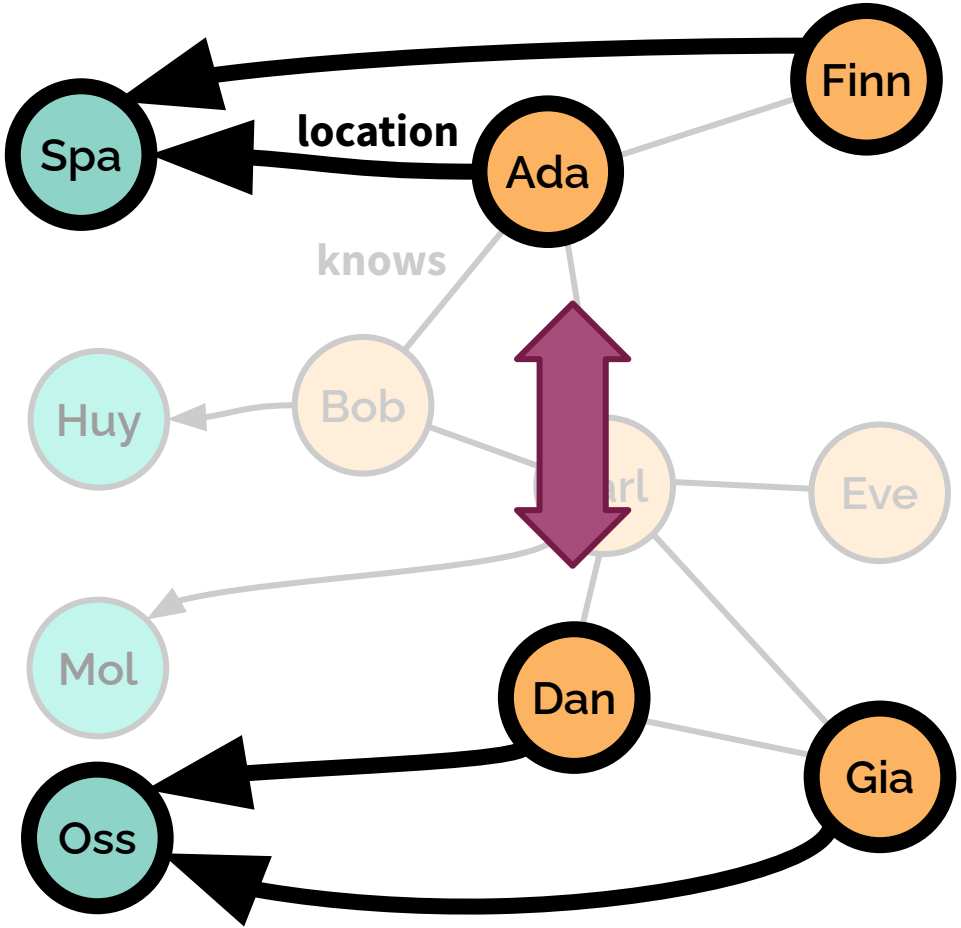
Graph BI 1: Cheapest path queries

Find cheapest paths (weighted shortest)

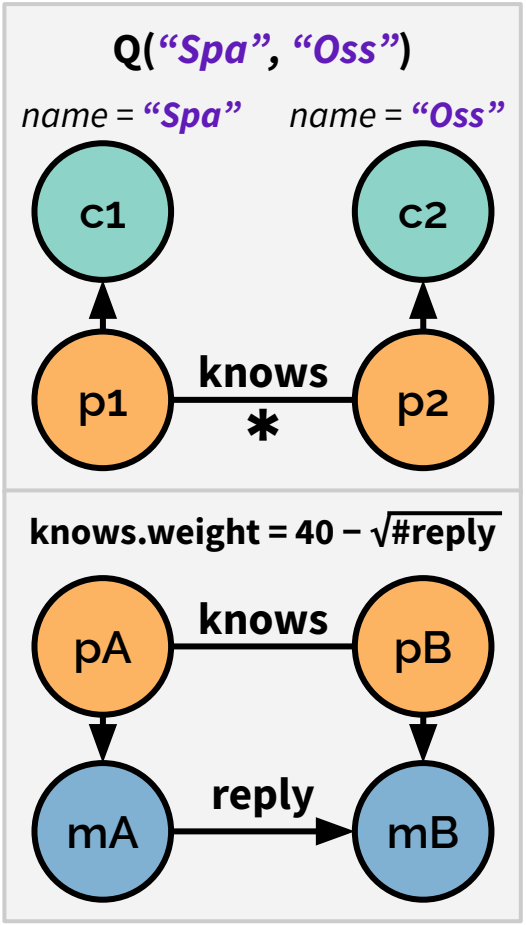




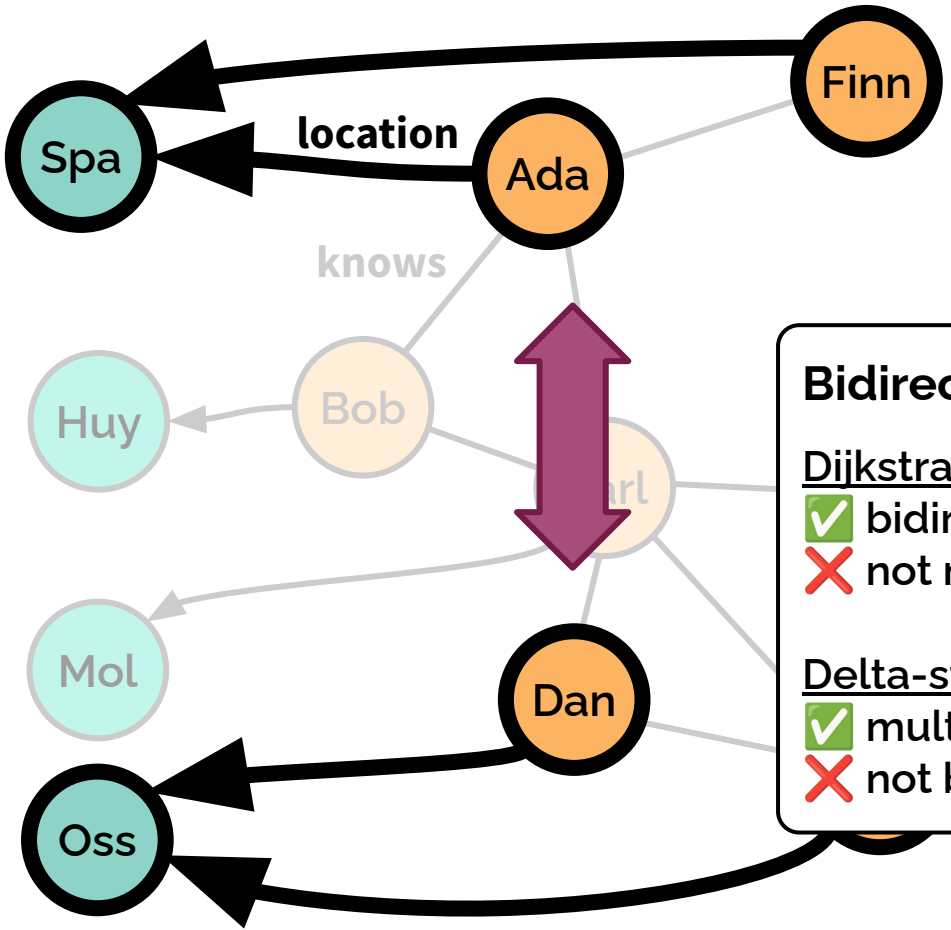
Find cheapest paths (weighted shortest)



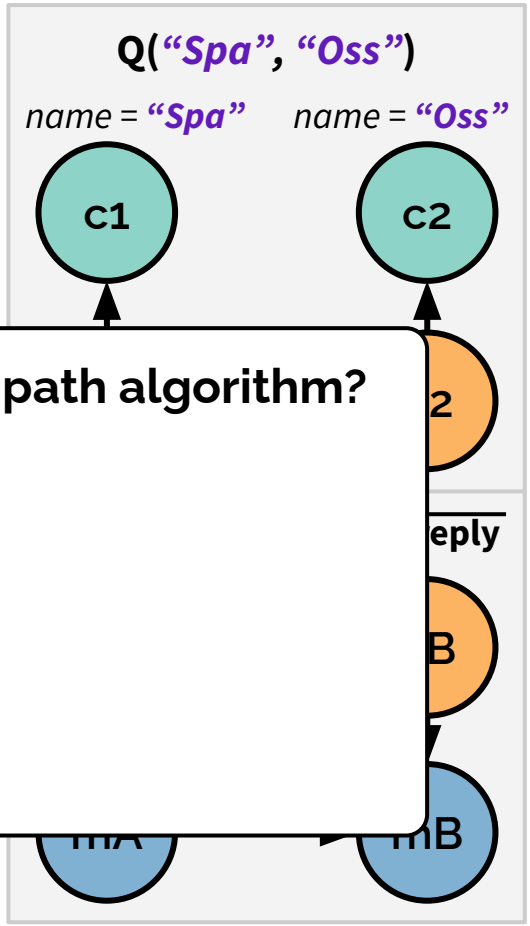
src	trg	w
Ada	Dan	…
Ada	Gia	…
Finn	Dan	…
Finn	Gia	…



Find cheapest paths (weighted shortest)



src	trg	w
Ada	Dan	…
Ada	Gia	…
Finn	Dan	…
Finn	Gia	…



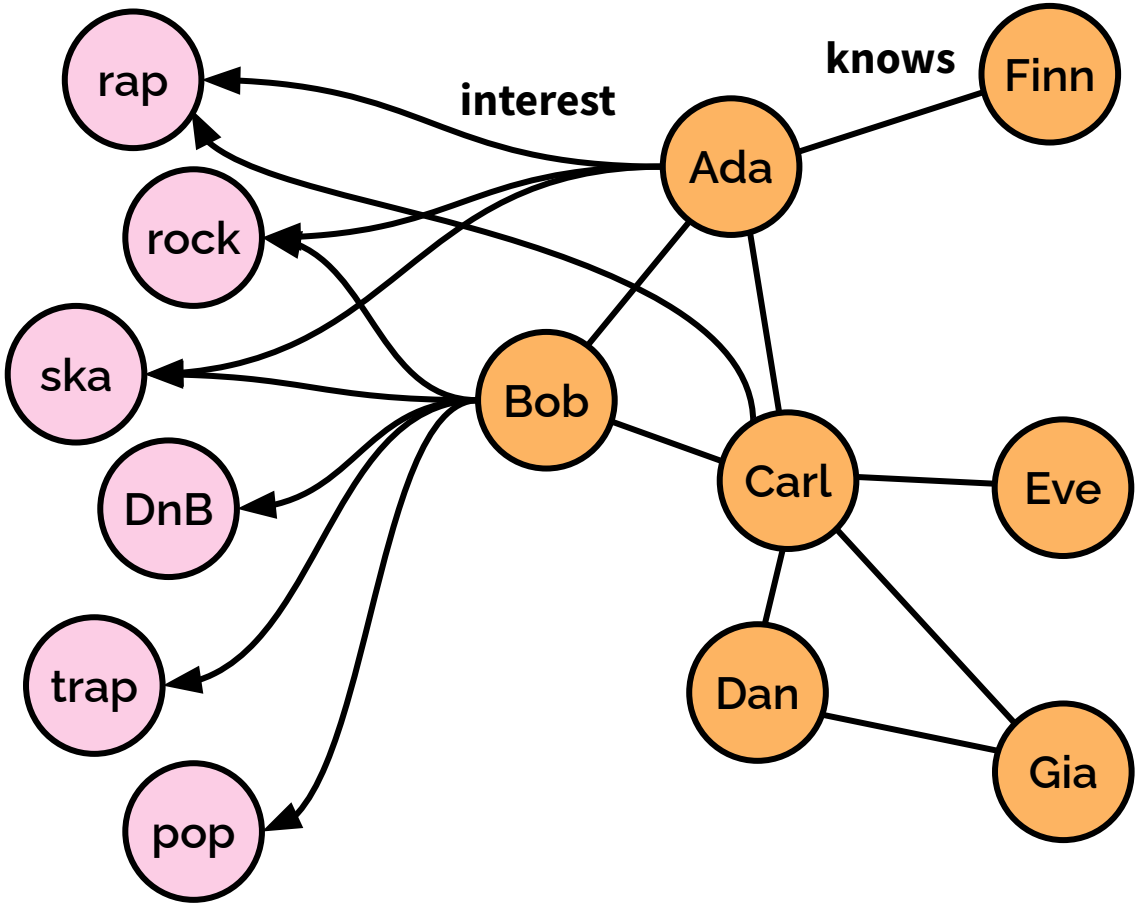
Bidirectional cheapest path algorithm?

<u>Dijkstra's algorithm</u>
✅ bidirectional
❌ not multi-threaded

<u>Delta-stepping</u>
✅ multi-threaded
❌ not bidirectional

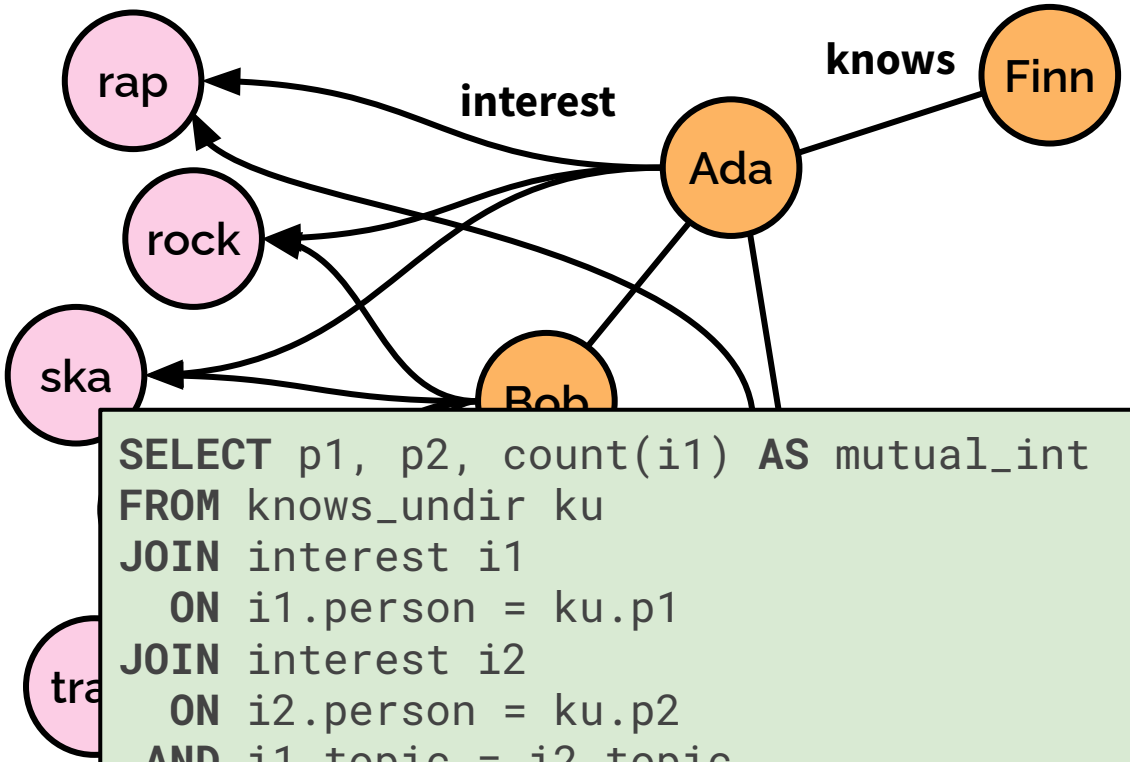
Graph BI 2: Cyclic queries

Cyclic query: Return shared interests



t1	p1	p2	t2
rap	Ada	Bob	rock
rock	Ada	Bob	rock
ska	Ada	Bob	rock
rap	Ada	Bob	ska
rock	Ada	Bob	ska
ska	Ada	Bob	ska
rap	Ada	Bob	DnB
rock	Ada	Bob	DnB
ska	Ada	Bob	DnB
rap	Ada	Bob	trap
rock	Ada	Bob	trap
ska	Ada	Bob	trap
rap	Ada	Carl	rap
rock	Ada	Carl	rap
ska	Ada	Carl	rap

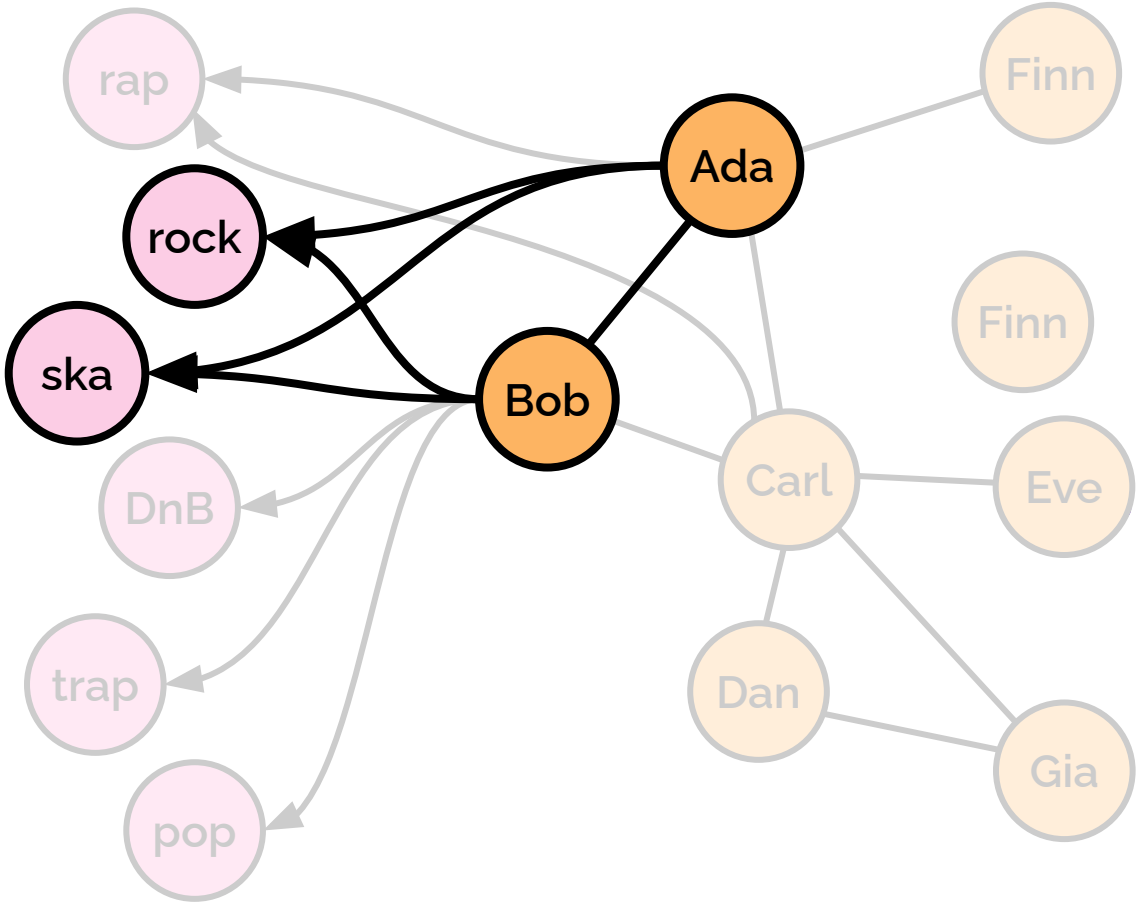
Cyclic query: Return shared interests




```
SELECT p1, p2, count(i1) AS mutual_int
FROM knows_undir ku
JOIN interest i1
  ON i1.person = ku.p1
JOIN interest i2
  ON i2.person = ku.p2
 AND i1.topic = i2.topic
WHERE p1 < p2
GROUP BY p1, p2;
```

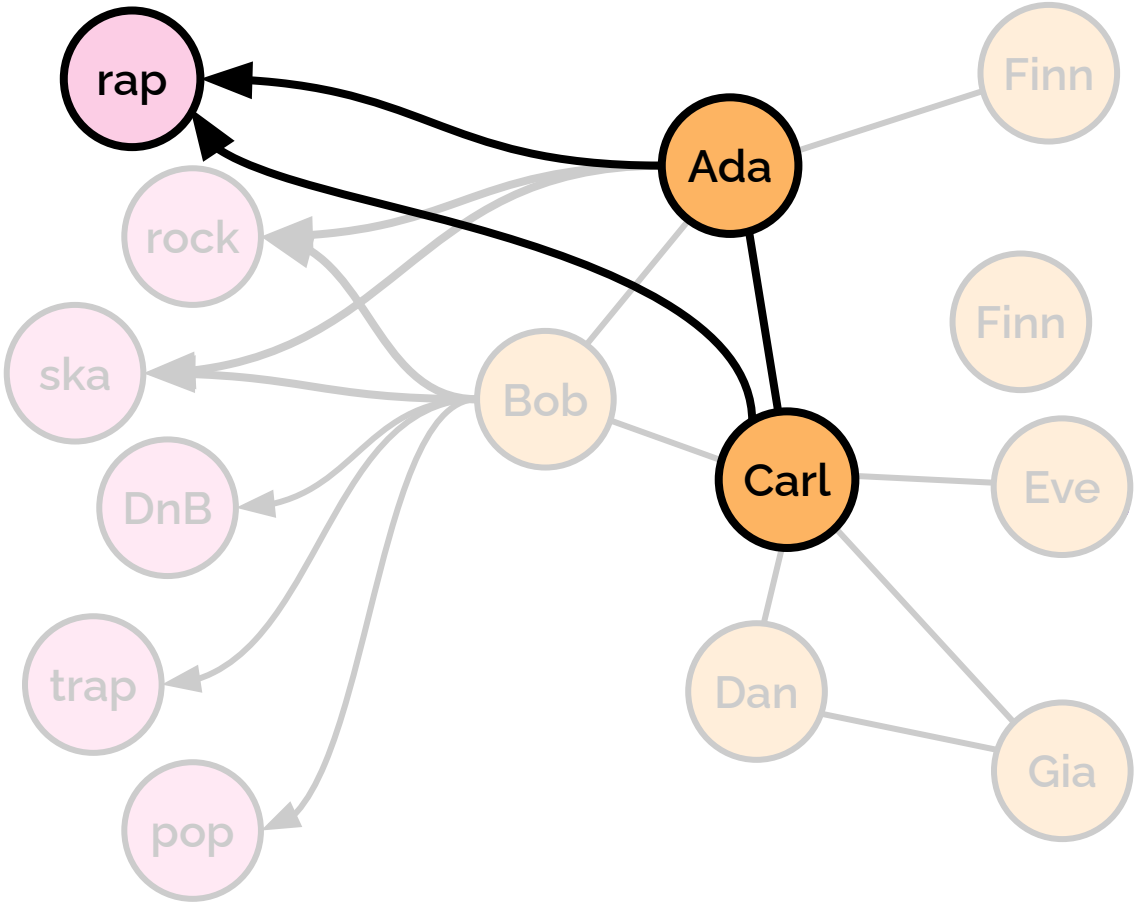
t1	p1	p2	t2
rap	Ada	Bob	rock
rock	Ada	Bob	rock
ska	Ada	Bob	rock
rap	Ada	Bob	ska
rock	Ada	Bob	ska
ska	Ada	Bob	ska
rap	Ada	Bob	DnB
rock	Ada	Bob	DnB
ska	Ada	Bob	DnB
rap	Ada	Bob	trap
rock	Ada	Bob	trap
ska	Ada	Bob	trap
rap	Ada	Carl	rap
rock	Ada	Carl	rap
ska	Ada	Carl	rap

Cyclic query: Return shared interests



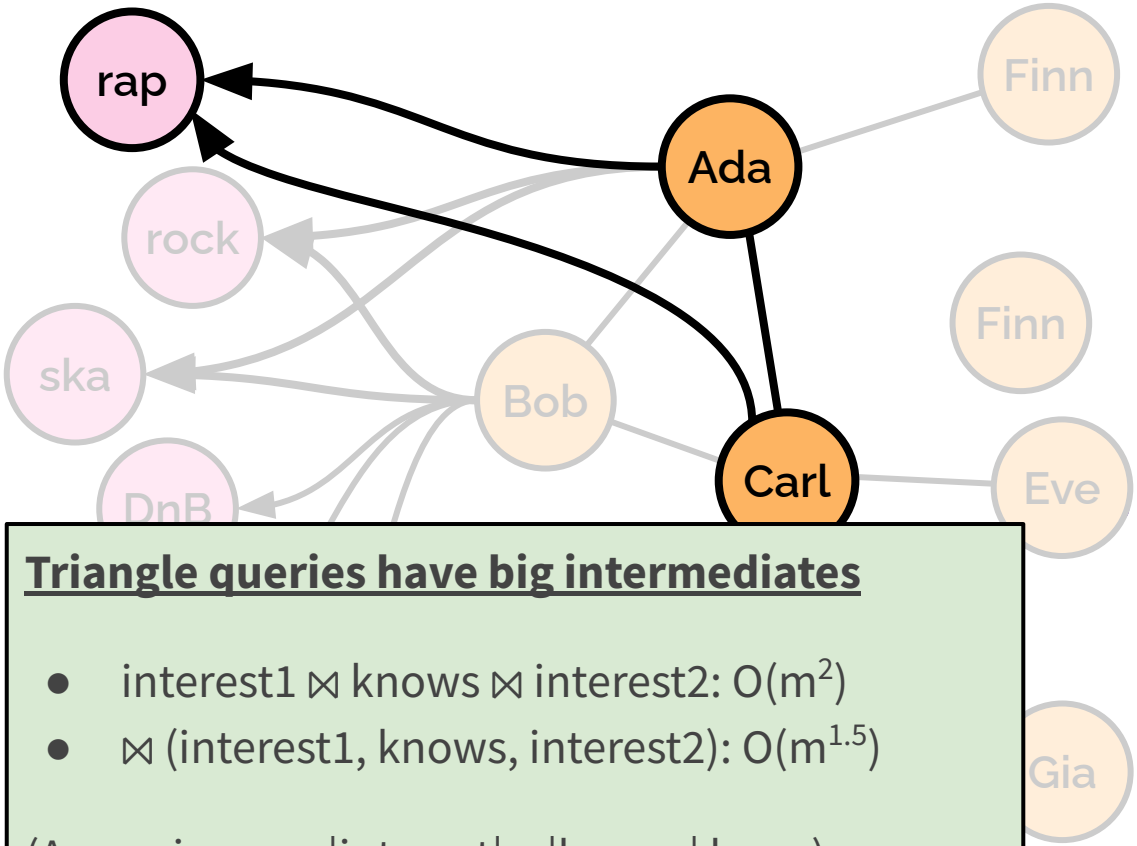
t1	p1	p2	t2
rap	Ada	Bob	rock
rock	**Ada**	**Bob**	**rock**
ska	Ada	Bob	rock
rap	Ada	Bob	ska
rock	Ada	Bob	ska
ska	**Ada**	**Bob**	**ska**
rap	Ada	Bob	DnB
rock	Ada	Bob	DnB
ska	Ada	Bob	DnB
rap	Ada	Bob	trap
rock	Ada	Bob	trap
ska	Ada	Bob	trap
rap	Ada	Carl	rap
rock	Ada	Carl	rap
ska	Ada	Carl	rap

Cyclic query: Return shared interests



t1	p1	p2	t2
rap	Ada	Bob	rock
rock	Ada	Bob	rock
ska	Ada	Bob	rock
rap	Ada	Bob	ska
rock	Ada	Bob	ska
ska	Ada	Bob	ska
rap	Ada	Bob	DnB
rock	Ada	Bob	DnB
ska	Ada	Bob	DnB
rap	Ada	Bob	trap
rock	Ada	Bob	trap
ska	Ada	Bob	trap
rap	**Ada**	**Carl**	**rap**
rock	Ada	Carl	rap
ska	Ada	Carl	rap

Cyclic query: Return shared interests



Triangle queries have big intermediates

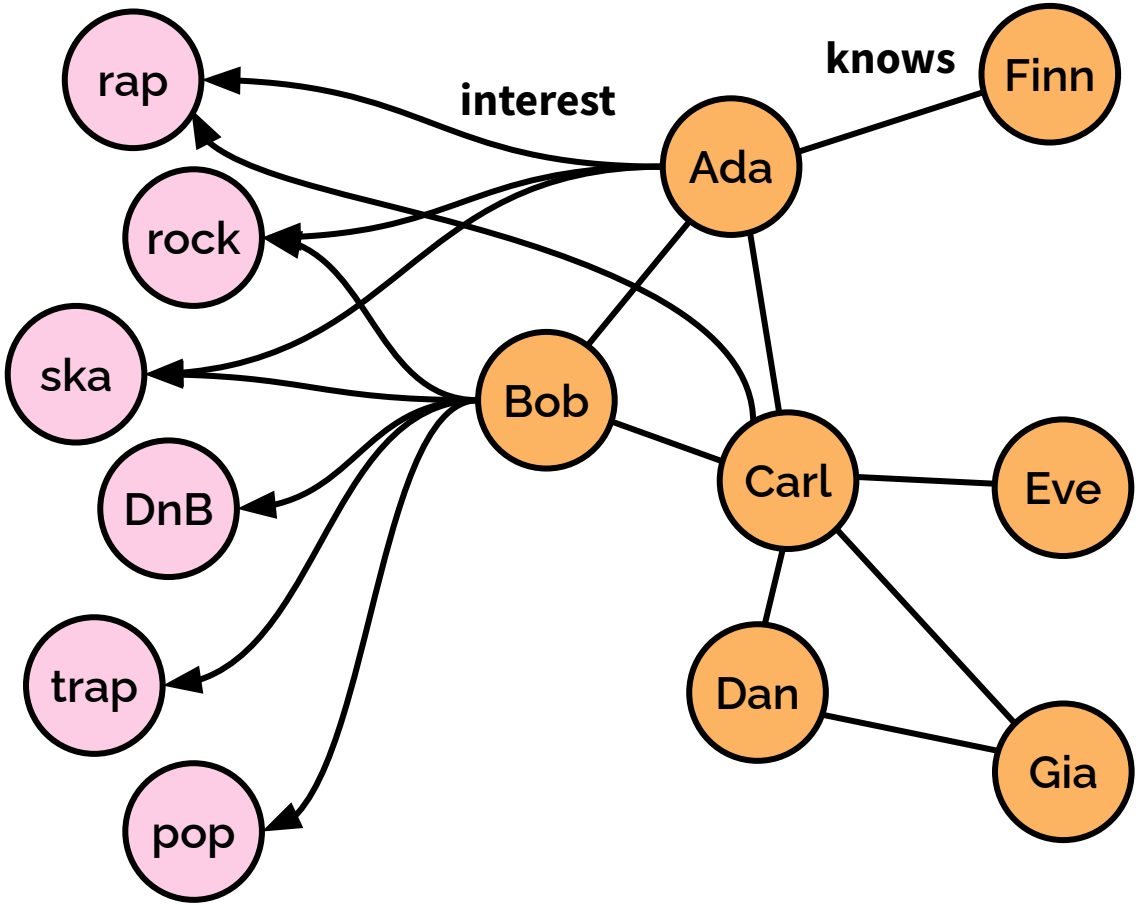
- interest1 ⋈ knows ⋈ interest2: $O(m^2)$
- ⋈ (interest1, knows, interest2): $O(m^{1.5})$

(Assuming m = |interest| = |knows| here.)

t1	p1	p2	t2
rap	Ada	Bob	rock
rock	Ada	Bob	rock
ska	Ada	Bob	rock
rap	Ada	Bob	ska
rock	Ada	Bob	ska
ska	Ada	Bob	ska
rap	Ada	Bob	DnB
rock	Ada	Bob	DnB
ska	Ada	Bob	DnB
rap	Ada	Bob	trap
rock	Ada	Bob	trap
ska	Ada	Bob	trap
rap	**Ada**	**Carl**	**rap**
rock	Ada	Carl	rap
ska	Ada	Carl	rap

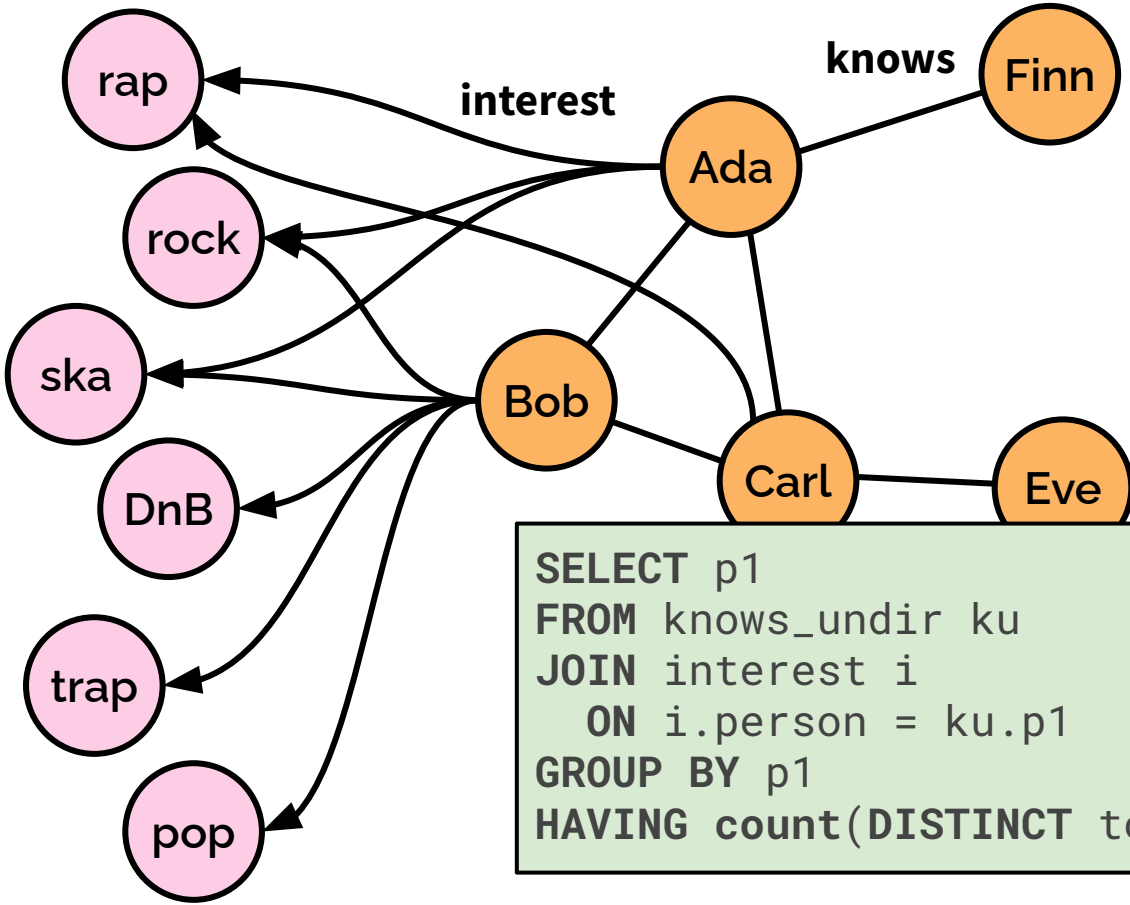
Graph BI 3: Acyclic queries

More interests than friends



Which person(s) have more topics of interests than friends?

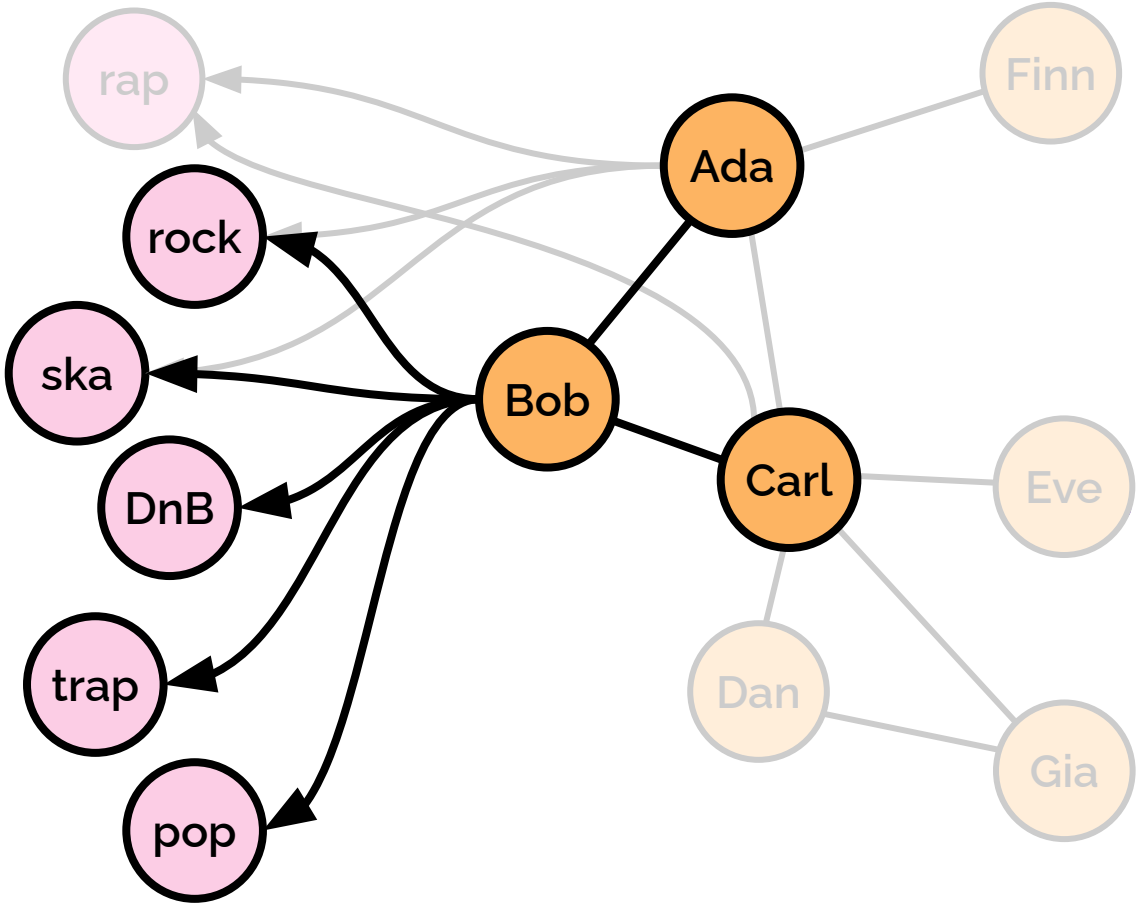
More interests than friends



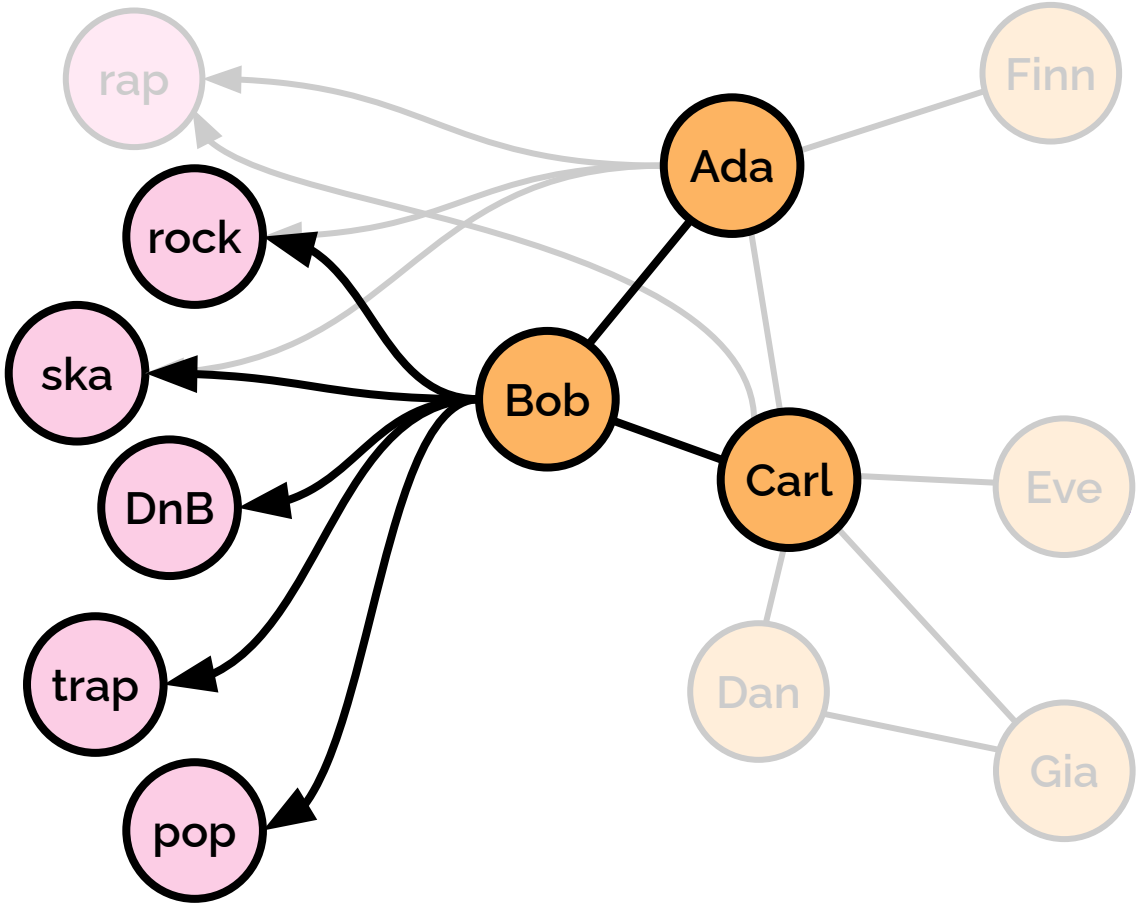
Which person(s) have more topics of interests than friends?

```
SELECT p1
FROM knows_undir ku
JOIN interest i
  ON i.person = ku.p1
GROUP BY p1
HAVING count(DISTINCT topic) > count(DISTINCT p2);
```

More interests than friends



More interests than friends



topic	p1	p2
rock	Bob	Ada
ska	Bob	Ada
DnB	Bob	Ada
trap	Bob	Ada
pop	Bob	Ada
rock	Bob	Carl
ska	Bob	Carl
DnB	Bob	Carl
trap	Bob	Carl
pop	Bob	Carl

Multi-valued dependency:

p1 ↠ **topic** and **p1** ↠ **p2**

Many-to-many joins introduce a predictable type of redundancy which we can compress away!

topic	p1	p2
rap	Ada	Bob
rock	Ada	Bob
ska	Ada	Bob
rap	Ada	Carl
rock	Ada	Carl
ska	Ada	Carl
rap	Ada	Finn
rock	Ada	Finn
ska	Ada	Finn
rock	Bob	Ada
ska	Bob	Ada
DnB	Bob	Ada
trap	Bob	Ada
pop	Bob	Ada
rock	Bob	Carl
ska	Bob	Carl
DnB	Bob	Carl
trap	Bob	Carl
pop	Bob	Carl
rap	Carl	Ada
rap	Carl	Bob

Factorization

Factorization is a lossless compression method.

flat to factorized

topic		p1		p2
{rap, rock, ska}	×	Ada	×	{Bob, Carl, Finn}
{rock, ska, DnB, trap, pop}	×	Bob	×	{Ada, Carl}
{rap}	×	Carl	×	{Ada, Bob}

```
SELECT p1
FROM knows_undir ku
JOIN interest i
  ON i.person = ku.p1
GROUP BY p1
HAVING count(DISTINCT topic) > count(DISTINCT p2);
```

topic	p1	p2
rap	Ada	Bob
rock	Ada	Bob
ska	Ada	Bob
rap	Ada	Carl
rock	Ada	Carl
ska	Ada	Carl
rap	Ada	Finn
rock	Ada	Finn
ska	Ada	Finn
rock	Bob	Ada
ska	Bob	Ada
DnB	Bob	Ada
trap	Bob	Ada
pop	Bob	Ada
rock	Bob	Carl
ska	Bob	Carl
DnB	Bob	Carl
trap	Bob	Carl
pop	Bob	Carl
rap	Carl	Ada
rap	Carl	Bob

Factorization

Factorization is a lossless compression method.

flat to factorized

topic		p1		p2
{rap, rock, ska}	×	Ada	×	{Bob, Carl, Finn}
{rock, ska, DnB, trap, pop}	×	Bob	×	{Ada, Carl}
{rap}	×	Carl	×	{Ada, Bob}

Workloads heavy on many-to-many joins could benefit from factorization but there are many open questions:

How to factorize long chains? Which queries benefit from factorization? How to implement it efficiently? How to return a factorized data structure to the client?

Where did we start from?

Advanced factorization methods (“d-representation”) also *returning compact graphs*

partOf ⋈ City ⋈ location ⋈ Person ⋈ author ⋈ Message

city name	post code	population	person name	birth year	message id	day	comment
Mol	2400	37,000	Carl	1986	NULL	NULL	NULL
Mol	2400	37,000	Eve	2001	M3	Sun	alright
Mol	2400	37,000	Eve	2001	M4	Tue	Hello

city name	post code	population		person name	birth year		message
Mol	2400	37,000	×	Carl	1986	×	NULL
				Eve	2001	×	{ ⟨M3, Sun, alright⟩, ⟨M4, Tue, Hello⟩ }

Research papers

Worst-case optimal joins:

- 2013 **AGM bound**. SIAM J. Comput.
- 2014 **Worst-case optimal joins**. PODS
- 2019 **Vertex ordering in worst-case optimal joins.** VLDB
- 2020 **Hash-based worst-case optimal join implementations.** VLDB

Factorization:

- 2012 **FDB: A query engine for factorised relational databases.** VLDB
- 2015 **Size bounds for factorized representations of query results.** TODS
- 2024 **Optimizing queries with many-to-many joins.**
- 2025 **Adaptive factorization using linear-chained hash tables.** CIDR

Kùzu

2023

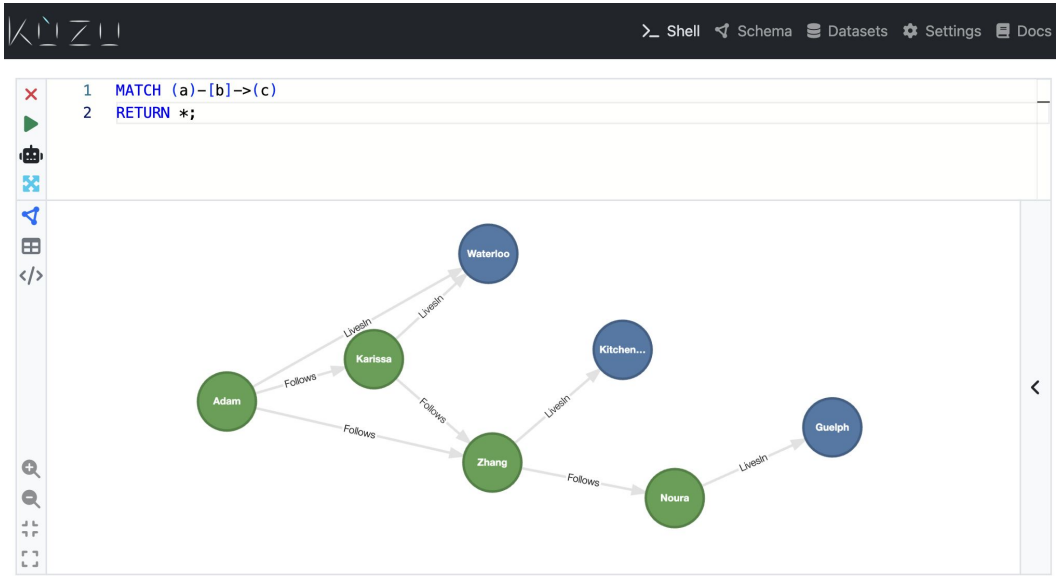
Cypher

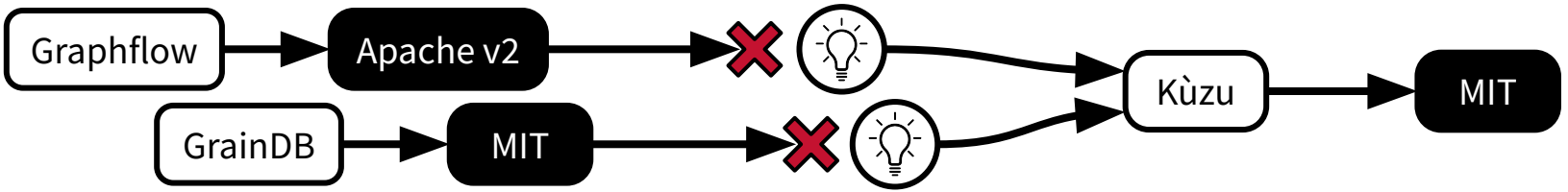
Single-node, in-process system

Uses a relational backend

Supports Cypher

Strong focus on path queries, worst-case optimal joins, and factorization





TigerGraph

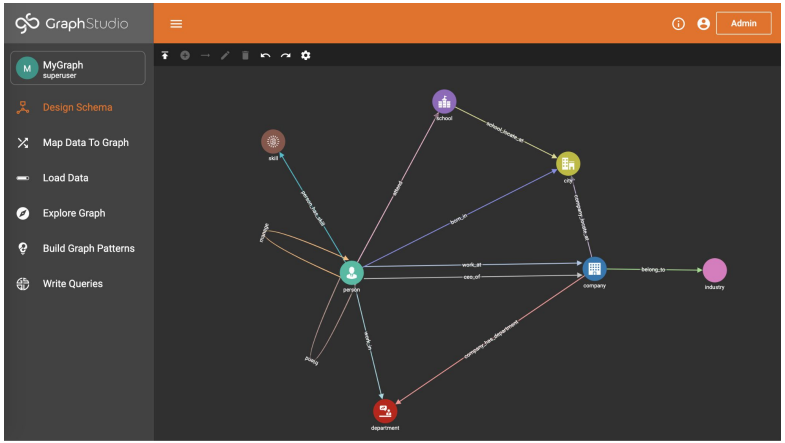
2012

GSQL

Focus on path queries

GSQL language:

```
CREATE QUERY hello(VERTEX<Person> p) {
  Start = {p};
  Result = SELECT dst
    FROM Start:src -(knows:e)- Person:dst;
    PRINT Result;
}
```



TigerGraph → proprietary

transactional

analytical

“MongoDB”

repeated joins =
n + 1 query problem

“Postgres”

recursive joins =
path queries

“Teradata”

many-to-many joins =

complex recursive joins
cyclic graph patterns
long acyclic patterns

What about schema and distribution?

Schema? Single-node vs. distributed?

	single-node	distributed
no / optional schema	Neo4j CE OrientDB / ArcadeDB / YouTrackDB	Neo4j EE Cosmos DB
strict schema	Kùzu DuckPGQ	Titan / JanusGraph / HugeGraph TigerGraph

Benchmarks

LDBC: Linked Data Benchmark Council

Mission:
Accelerate progress
in graph data management

Membership:
~25 organizations
~100 individuals


ANT
GROUP


火山引擎
Beijing Volcano (ByteDance)


ORACLE
Labs


Alibaba Cloud


aws


GRAPHWISE


neo4j


TigerGraph


pometry


fabarta


thatDot


RelationalAI


ULTIPA


MEM
GRAPH


NebulaGraph

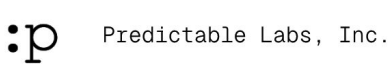

HUAWEI
HUAWEI CLOUD


Birkbeck
UNIVERSITY OF LONDON


*Sparsity


创邻科技
CREATE LINK


JCC Consulting, Inc.

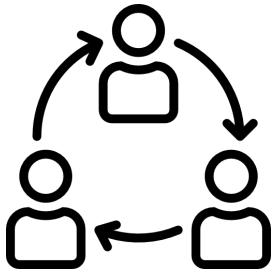

:p
Predictable Labs, Inc.


海致星图
StarGraph

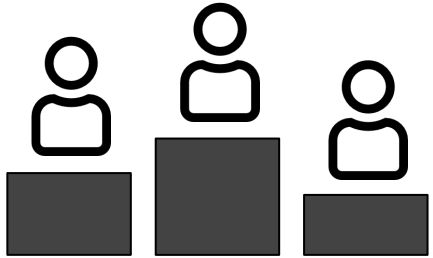
Members

- database companies
- hardware vendors
- cloud providers
- researchers

LDBC encourages members to…



collaborate
on standards

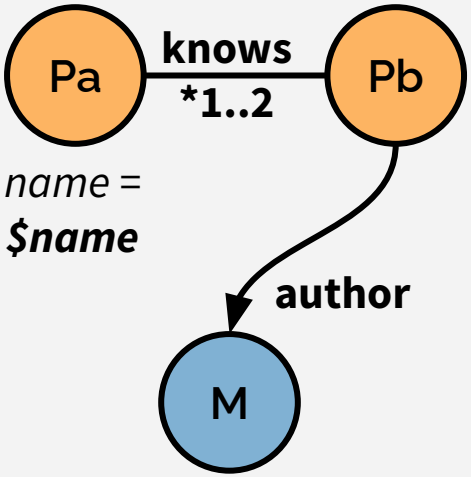


compete
on performance

Benchmarks and query languages

“Interactive” workload

Q9(*$name*, *$day*)



SQL:1992

```
SELECT DISTINCT m.id
FROM (
 SELECT k.p2id AS id
 FROM person Pa,
      knows k
 WHERE Pa.name = $name
   AND Pa.id = k.p1id
 UNION
 SELECT k2.p2id AS id
 FROM person Pa,
      knows k1,
      knows k2
 WHERE Pa.name = $name
   AND Pa.id = k1.p1id
   AND k1.p2id = k2.p1id
   AND k1.p1id <> k2.p2id
 ) Pb,
 Message m
WHERE Pb.id = m.authorId
  AND m.creationDate < $day
```

SQL/PGQ (SQL:2023)

```
SELECT id
FROM GRAPH_TABLE (socialNetwork
 MATCH ANY ACYCLIC
  (Pa:Person WHERE Pa.name = $name)
  -[:knows]-{1,2} (Pb:Person)
  -[:author]-> (m:Message)
 WHERE m.creationDate < $day
 COLUMNS (m.id))
```

GQL

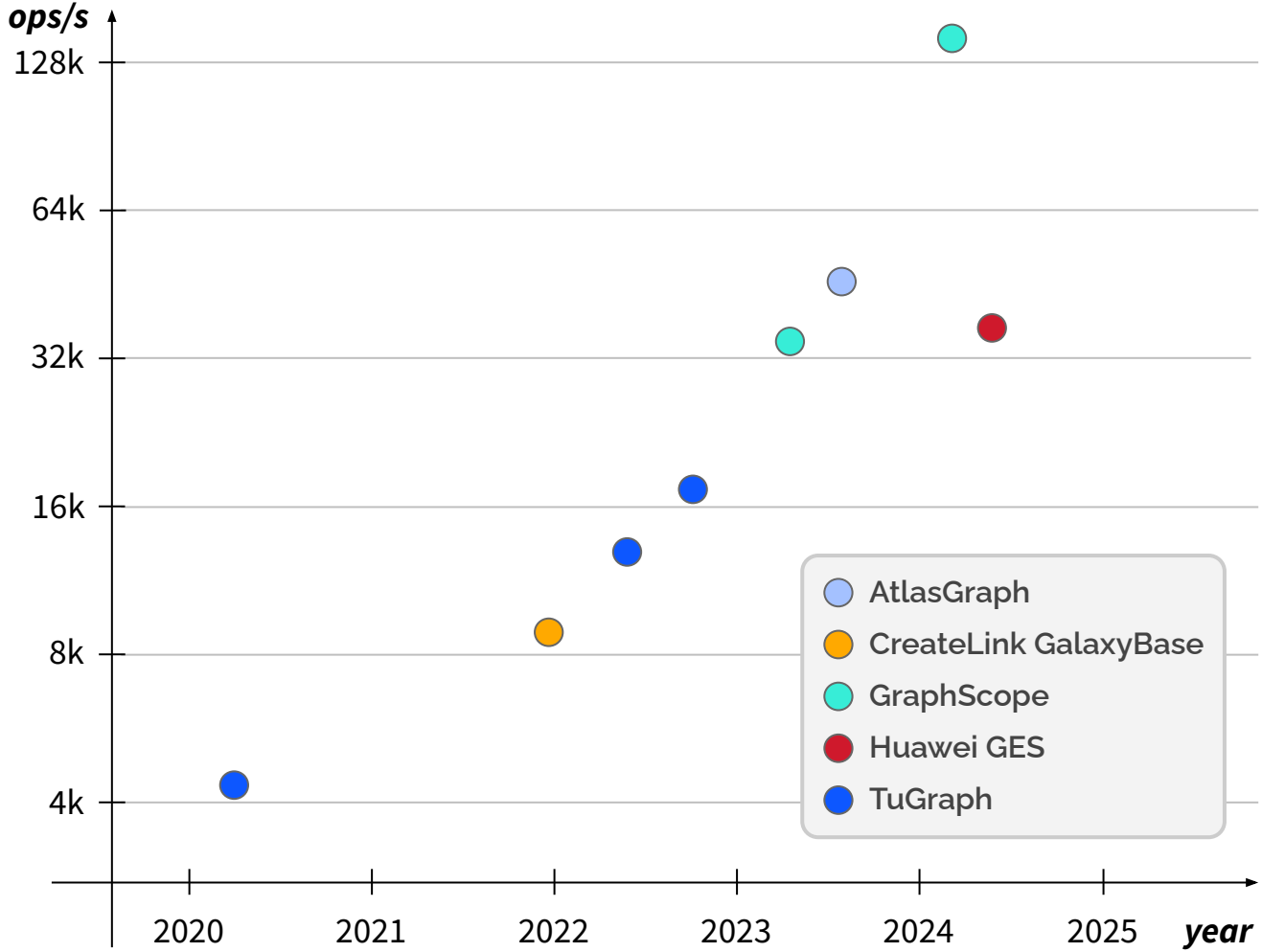
```
MATCH ANY ACYCLIC
  (Pa:Person WHERE Pa.name = $name)
  -[:knows]-{1,2} (Pb:Person)
  -[:author]-> (m:Message)
WHERE m.creationDate < $day
RETURN DISTINCT m.id
```

“Interactive” workload

SF100: 25× speedup in 4y

71× price-performance

All of these systems are developed by vendors based in China with a strong VC ecosystem

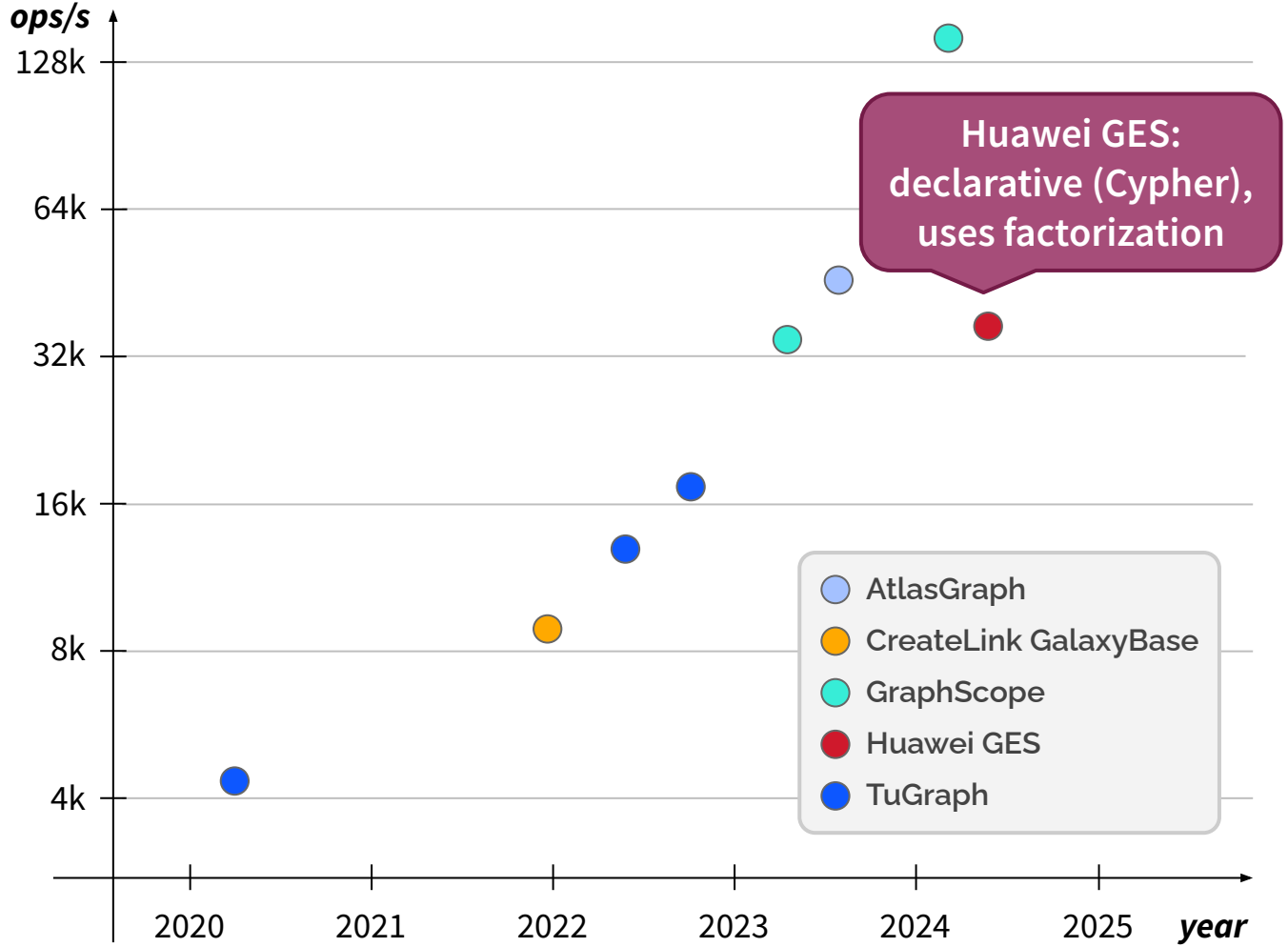


“Interactive” workload

SF100: 25× speedup in 4y

71× price-performance

All of these systems are developed by vendors based in China with a strong VC ecosystem

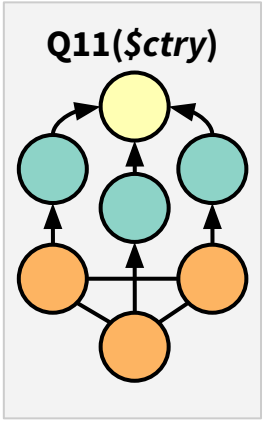


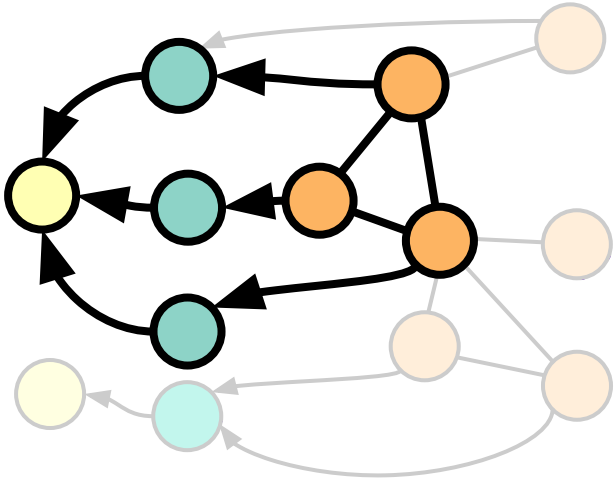
“Business Intelligence” workload

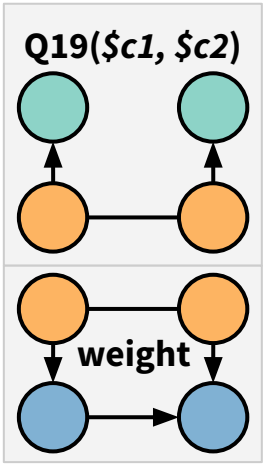
Analytical workload

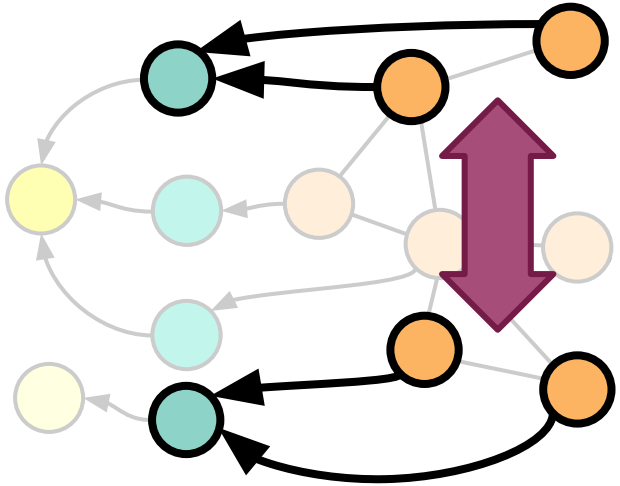
Metric 1: Power

Metric 2: Throughput









“Business Intelligence” workload

- Analytical workload
- Metric 1: Power
- Metric 2: Throughput

Audited results

	Scale factors
TigerGraph	100
	1,000 (×3)
	10,000
TuGraph™	30,000

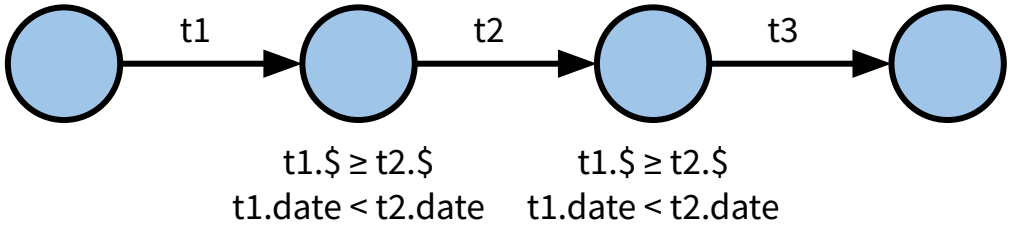
More audits coming this year!

Financial Benchmark

- Transactional workload
- Metric: Throughput
- Target: Distributed systems
- Relaxed consistency requirements

Developed by the Ant Group, Create Link, Ultipa, etc.

- Strict latency requirements (P99 < 100 ms)
- Relaxed consistency guarantees
- Truncation (sampling) on most recent edges
- Interesting queries, e.g. REM path queries (Regular Expression with Memory)



Using the benchmarks

Benchmark kit

Specification

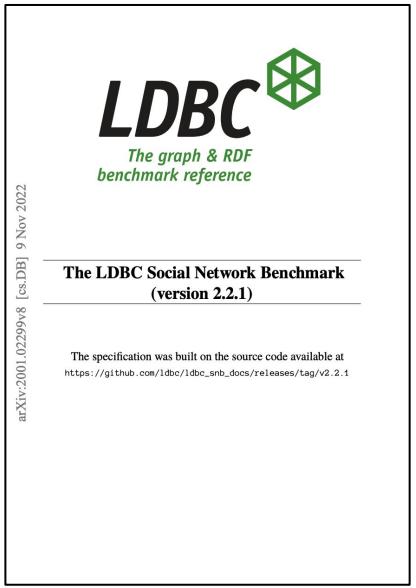
Academic paper

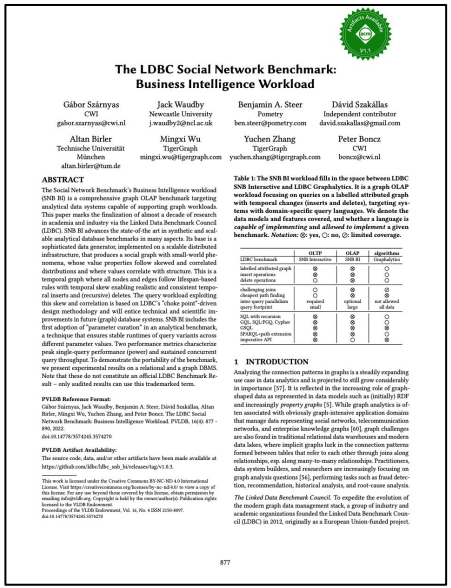
Data generator

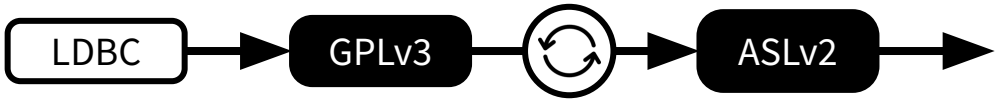
Pre-generated data sets

Driver

2+ implementations





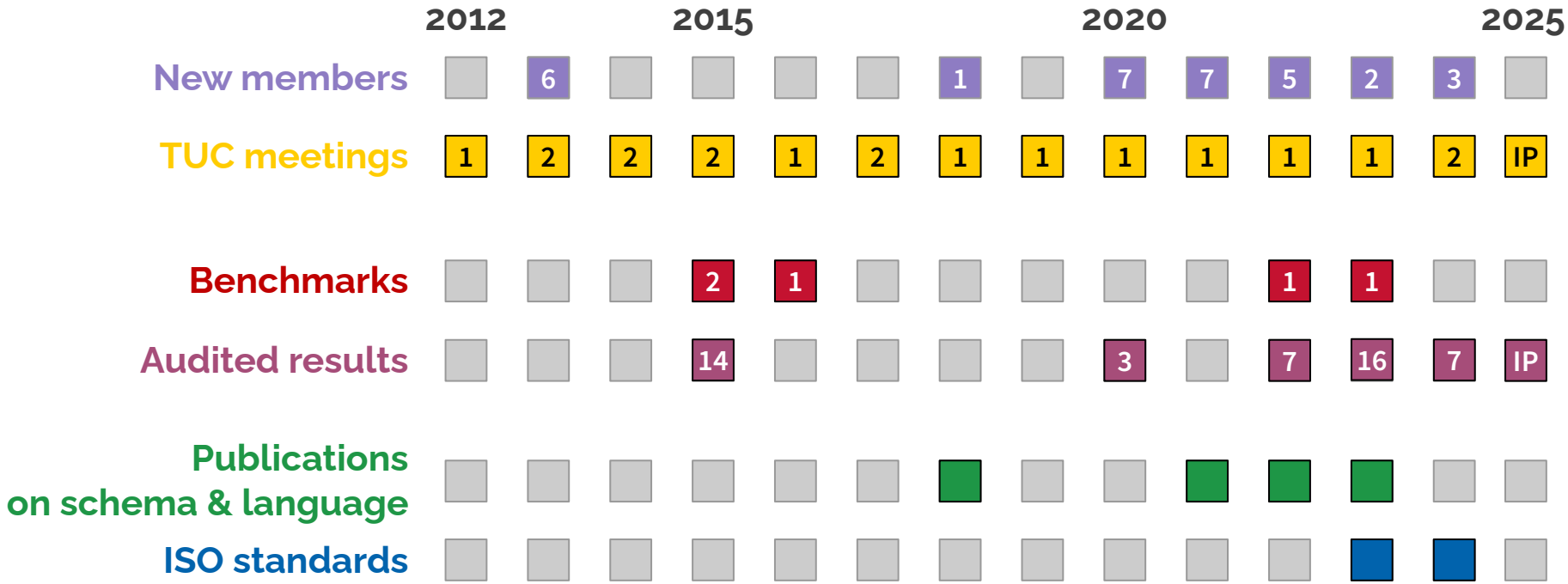


LDBC has “TPC-grade” benchmarks

- Auditing: an old model but it's still relevant
- Certified auditor, full disclosure report
- 3-year total cost of operations (licenses, support)
- Multi-week auditing process
- 5 years of auditing: published ≈50 results and had to retract 0

○ System: **GraphScope Flex 0.26.1** ○ Test sponsor: **Alibaba Cloud** ○ Date: **2024-05-14** ○ Queries implemented in: **C++** ○ System cost: **738,724 RMB** (102,128.22 USD)	100	Alibaba Cloud ecs.r8a.16xlarge 64×AMD EPYC 9T24 @ 3.7GHz vCPUs, 512GiB RAM	130,098.36 ops/s
	300	Alibaba Cloud ecs.r8a.16xlarge 64×AMD EPYC 9T24 @ 3.7GHz vCPUs, 512GiB RAM	131,263.87 ops/s

LDBC overview



Challenges
in the graph database space

Decline

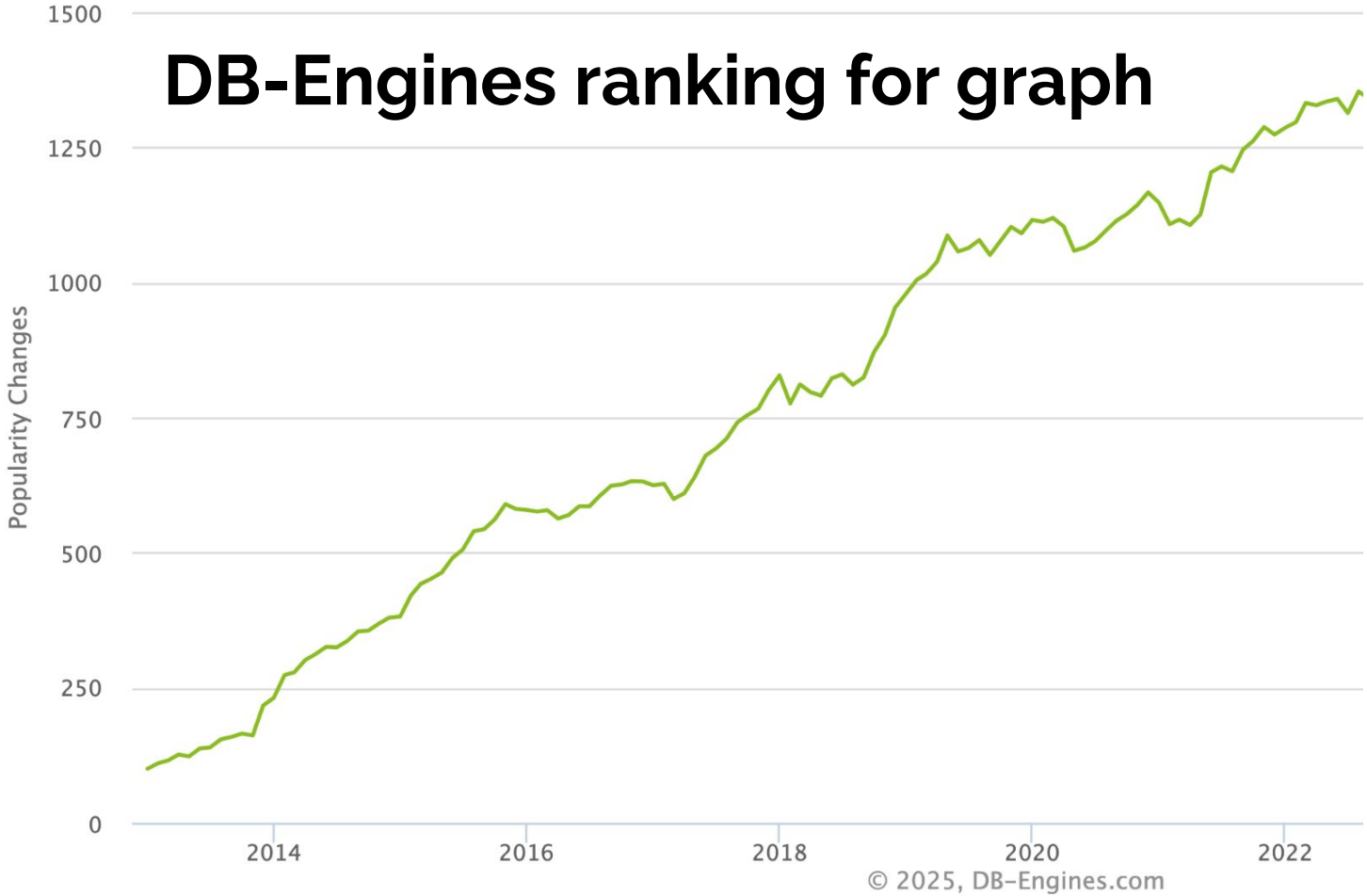
The hype cycle moved over to AI

The confusion doesn't help anyone!

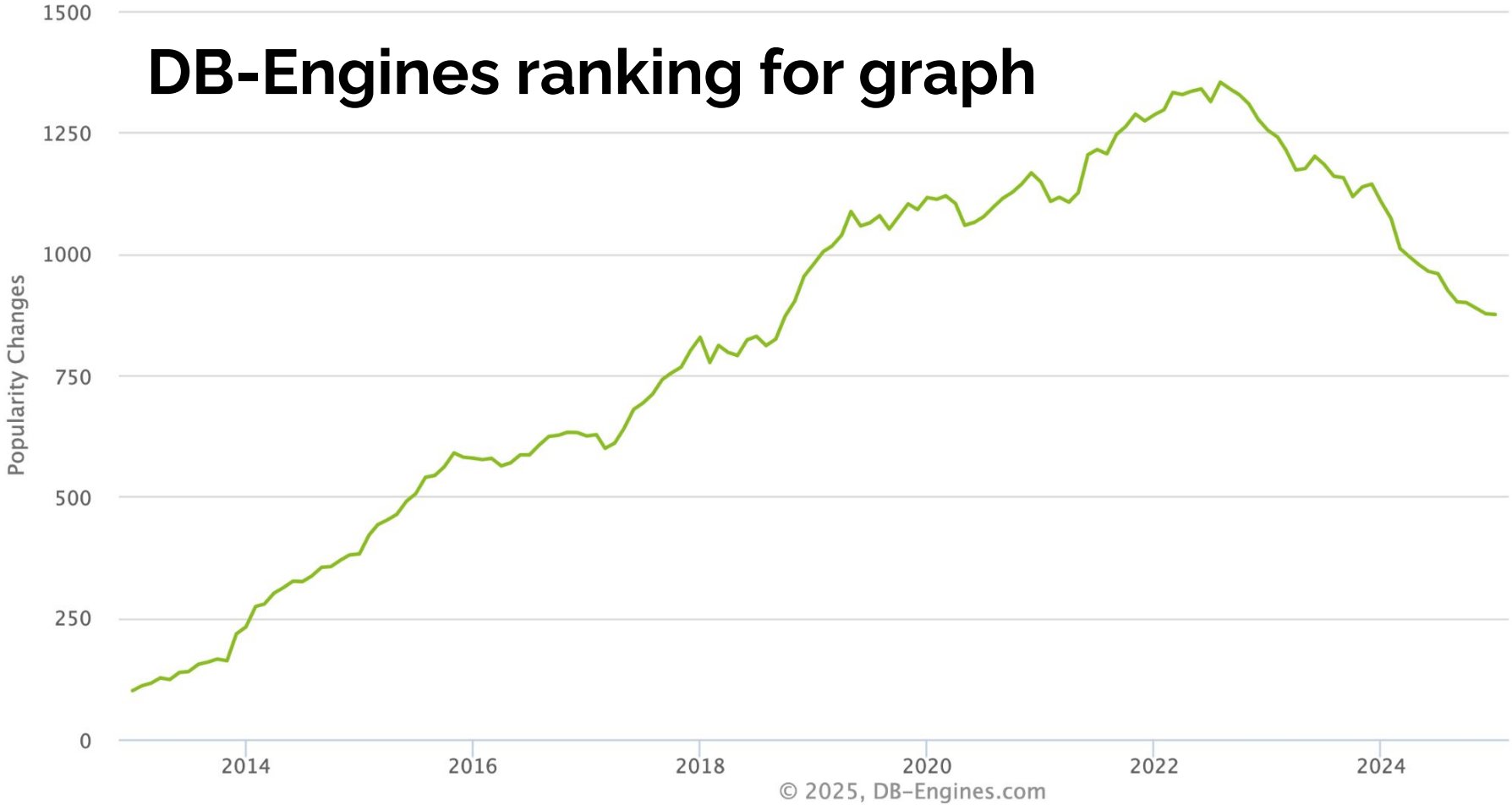
- “X is graph database system” → which category?
- “X doesn't need joins” → unnecessary and confusing conceptual shift
- “graph databases will replace RDBMSs” → this is very unlikely

There are niche use cases, which systems over-optimize for, causing fragmentation

DB-Engines ranking for graph



DB-Engines ranking for graph




GRAPH
RAG

Summing up

Graph databases <> joins

Graph databases have syntax sugar and optimizations for joins

If you have 10 joins in a query and it doesn't work well, try a graph database:

Fragmented landscape: graph databases are very specialized

Check licenses and performance results

Sources

Information sheets (all of them contain inaccuracies!):

- Database of databases: https://dbdb.io/
- DB-Engines Ranking of Graph DBMS: https://db-engines.com/en/ranking/graph+dbms
- Wikipedia page of vendors

Recommended readings / presentations:

- Amine Mhedhbi: Taming Large Intermediate Results for Joins over Graph-Structured Relations: A System Perspective
- Kùzu blog: https://blog.kuzudb.com/
- A eulogy for RedisGraph: https://www.bloorresearch.com/2023/08/a-eulogy-for-redisgraph/

Big thanks for discussions to Akon Dey, Amine Mhedhbi and Daniel ten Wolde.


LDBC
The graph & RDF
benchmark reference