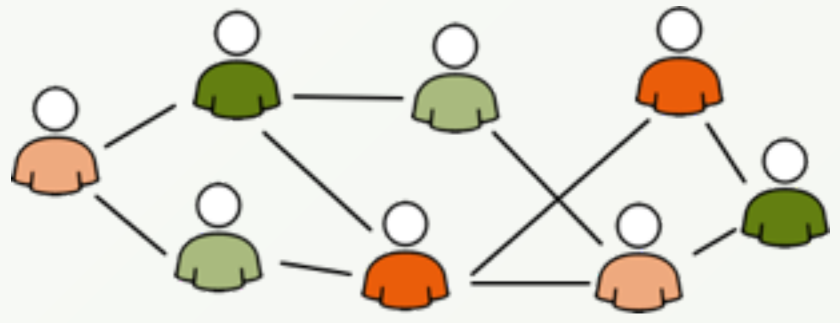
ArangoDB

Handling Billions Of Edges in a Graph Database

Michael Hackstein
@mchacki

Michael Hackstein

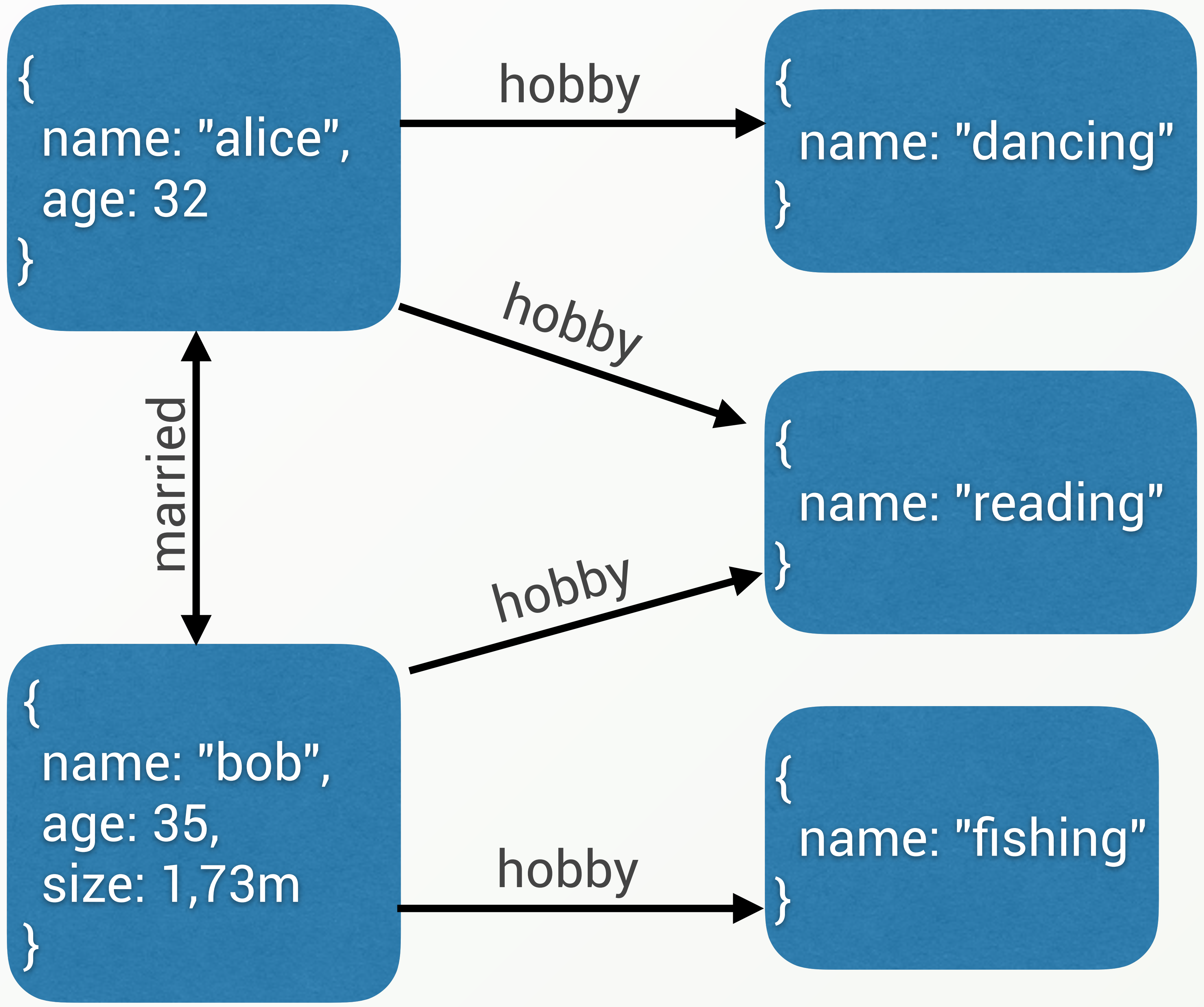
- ArangoDB Core Team
  - Web Frontend
  - Graph visualisation
  - Graph features
  - SmartGraphs



- Host of cologne.js

- Master's Degree (spec. Databases and Information Systems)

What are Graph Databases



```
{
  name: "alice",
  age: 32
}
```

hobby

```
{
  name: "dancing"
}
```

hobby

```
{
  name: "reading"
}
```

married

hobby

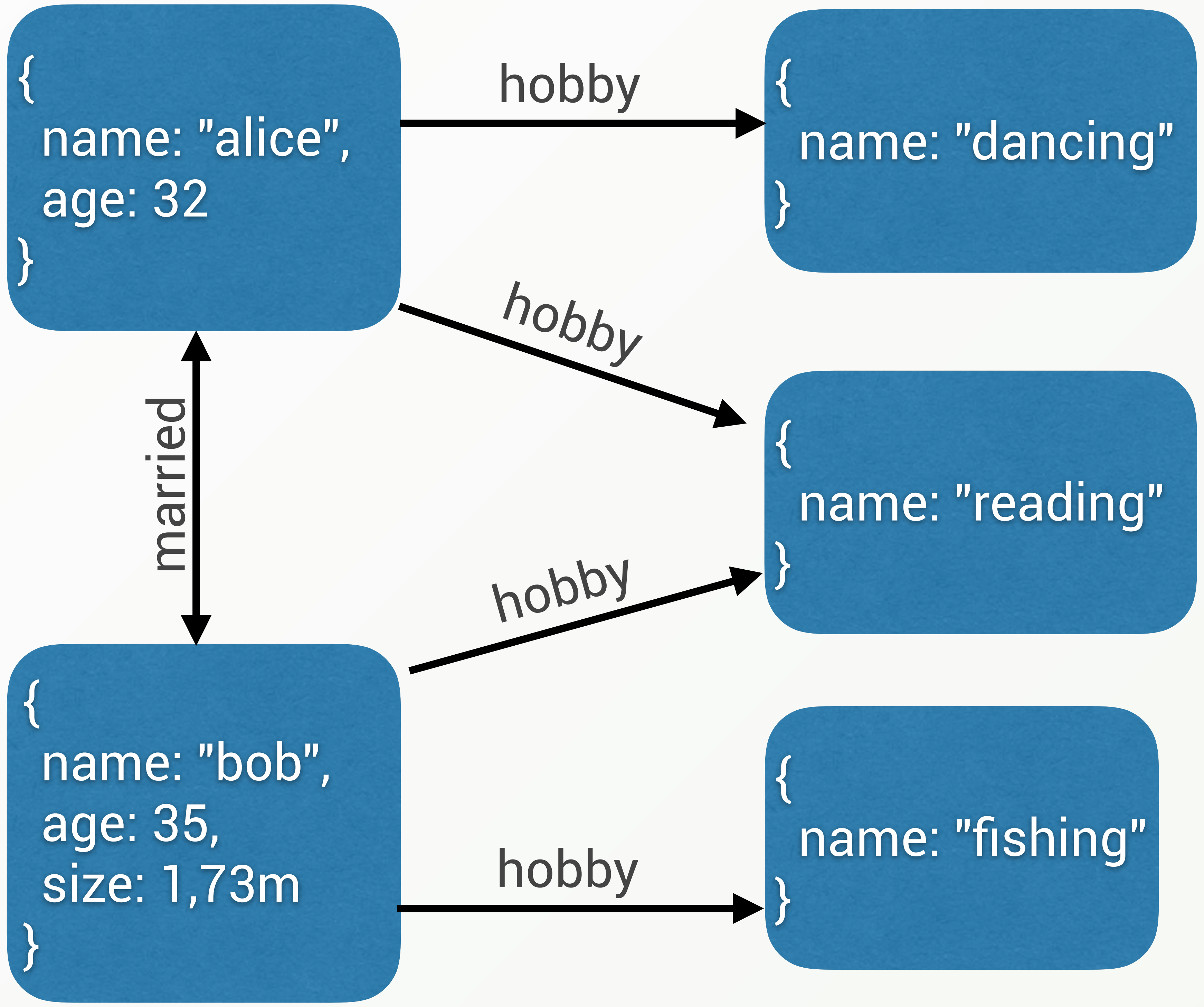
```
{
  name: "bob",
  age: 35,
  size: 1,73m
}
```

hobby

```
{
  name: "fishing"
}
```

- Schema-free Objects (Vertices)
- Relations between them (Edges)
- Edges have a direction

What are Graph Databases



- Schema-free Objects (Vertices)
- Relations between them (Edges)
- Edges have a direction

- Edges can be queried in both directions
- Easily query a range of edges (2 to 5)
- Undefined number of edges (1 to *)
- Shortest Path between two vertices

Typical Graph Queries

Typical Graph Queries

- Give me all friends of Alice

Typical Graph Queries

- Give me all friends of Alice
- Give me all friends-of-friends of Alice

Typical Graph Queries

- Give me all friends of Alice
- Give me all friends-of-friends of Alice
- What is the linking path between Alice and Bob

Typical Graph Queries

- Give me all friends of Alice
- Give me all friends-of-friends of Alice
- What is the linking path between Alice and Bob
- Which Trainstations can I reach if I am allowed to drive a distance of 6 stations on my ticket

Typical Graph Queries

- Give me all friends of Alice
- Give me all friends-of-friends of Alice
- What is the linking path between Alice and Bob
- Which Trainstations can I reach if I am allowed to drive a distance of 6 stations on my ticket
- Pattern Matching:

Typical Graph Queries

- Give me all friends of Alice
- Give me all friends-of-friends of Alice
- What is the linking path between Alice and Bob
- Which Trainstations can I reach if I am allowed to drive a distance of 6 stations on my ticket
- Pattern Matching:
  - Give me all users that share two hobbies with Alice

Typical Graph Queries

- Give me all friends of Alice
- Give me all friends-of-friends of Alice
- What is the linking path between Alice and Bob
- Which Trainstations can I reach if I am allowed to drive a distance of 6 stations on my ticket
- Pattern Matching:
  - Give me all users that share two hobbies with Alice
  - Give me all products that at least one of my friends has bought together with the products I already own, ordered by how many friends have bought it and the products rating, but only 20 of them.

Non-Typical Graph Queries

Non-Typical Graph Queries

- Give me all users which have an age attribute between 21 and 35.

Non-Typical Graph Queries

- Give me all users which have an age attribute between 21 and 35.
- Give me the age distribution of all users

Non-Typical Graph Queries

- Give me all users which have an age attribute between 21 and 35.
- Give me the age distribution of all users
- Group all users by their name

Traversal

Iterate down two edges with some filters

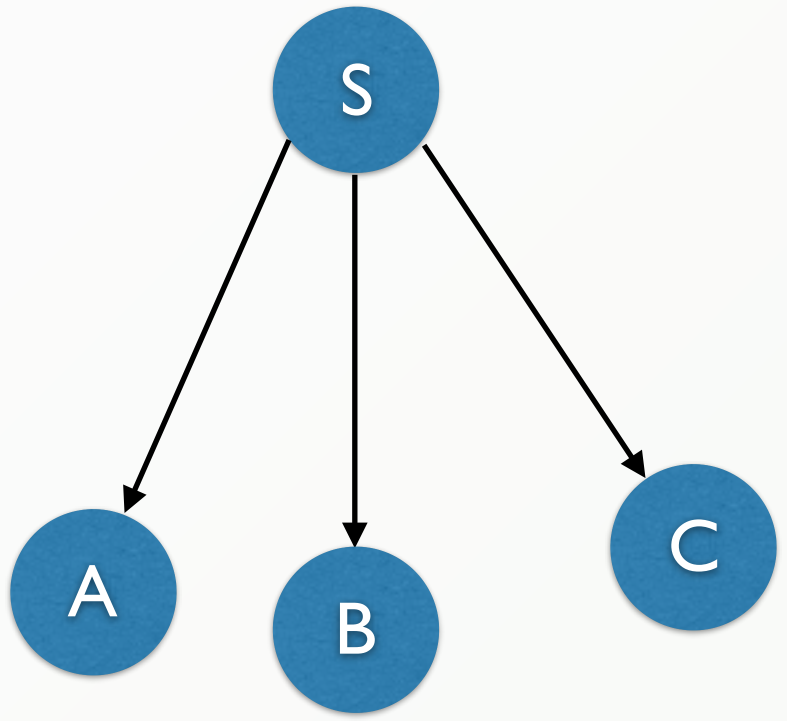
Traversal

Iterate down two edges with some filters

- We first pick a start vertex (S)

Traversal

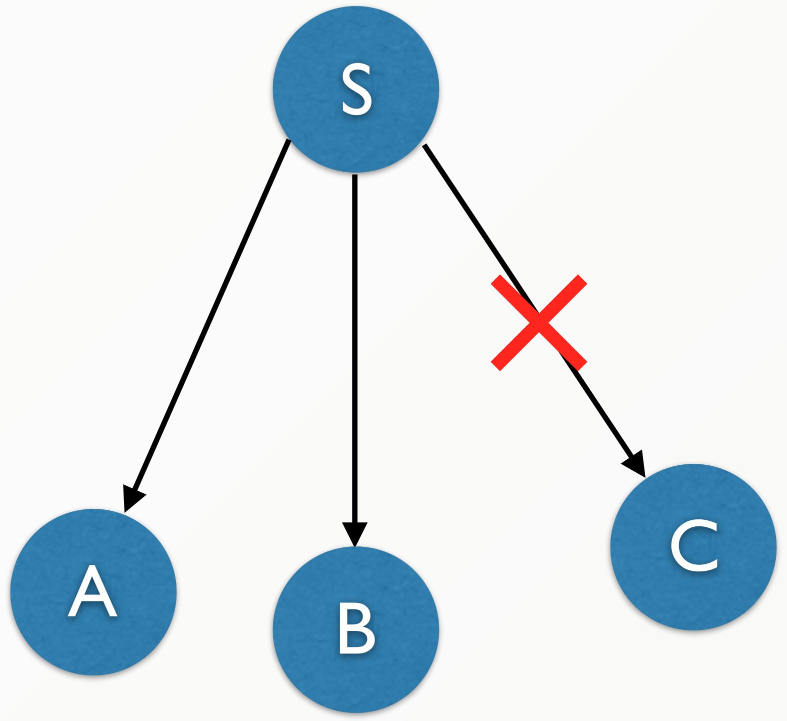
Iterate down two edges with some filters



- We first pick a start vertex (S)
- We collect all edges on S

Traversal

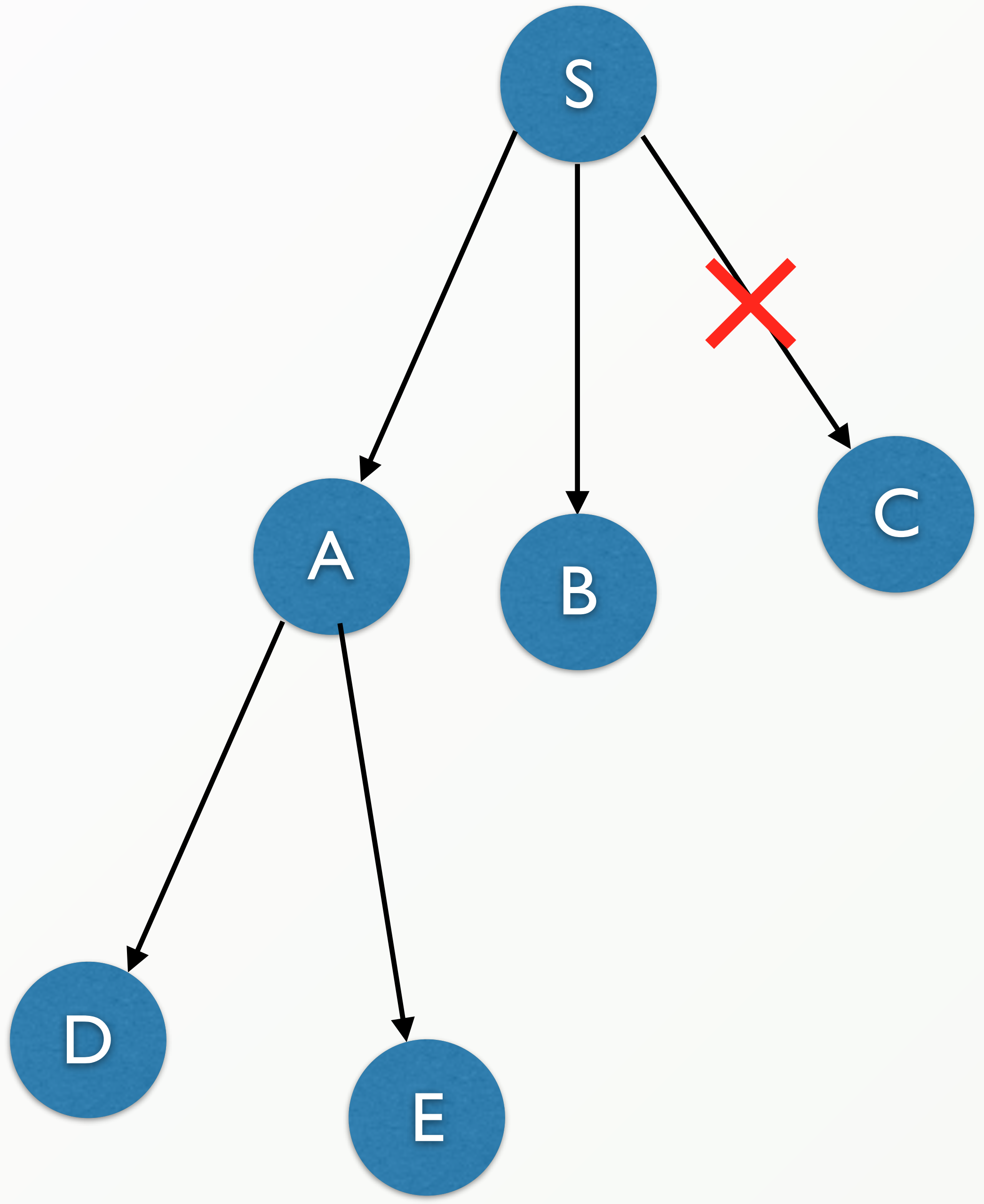
Iterate down two edges with some filters



- We first pick a start vertex (S)
- We collect all edges on S
- We apply filters on edges

Traversal

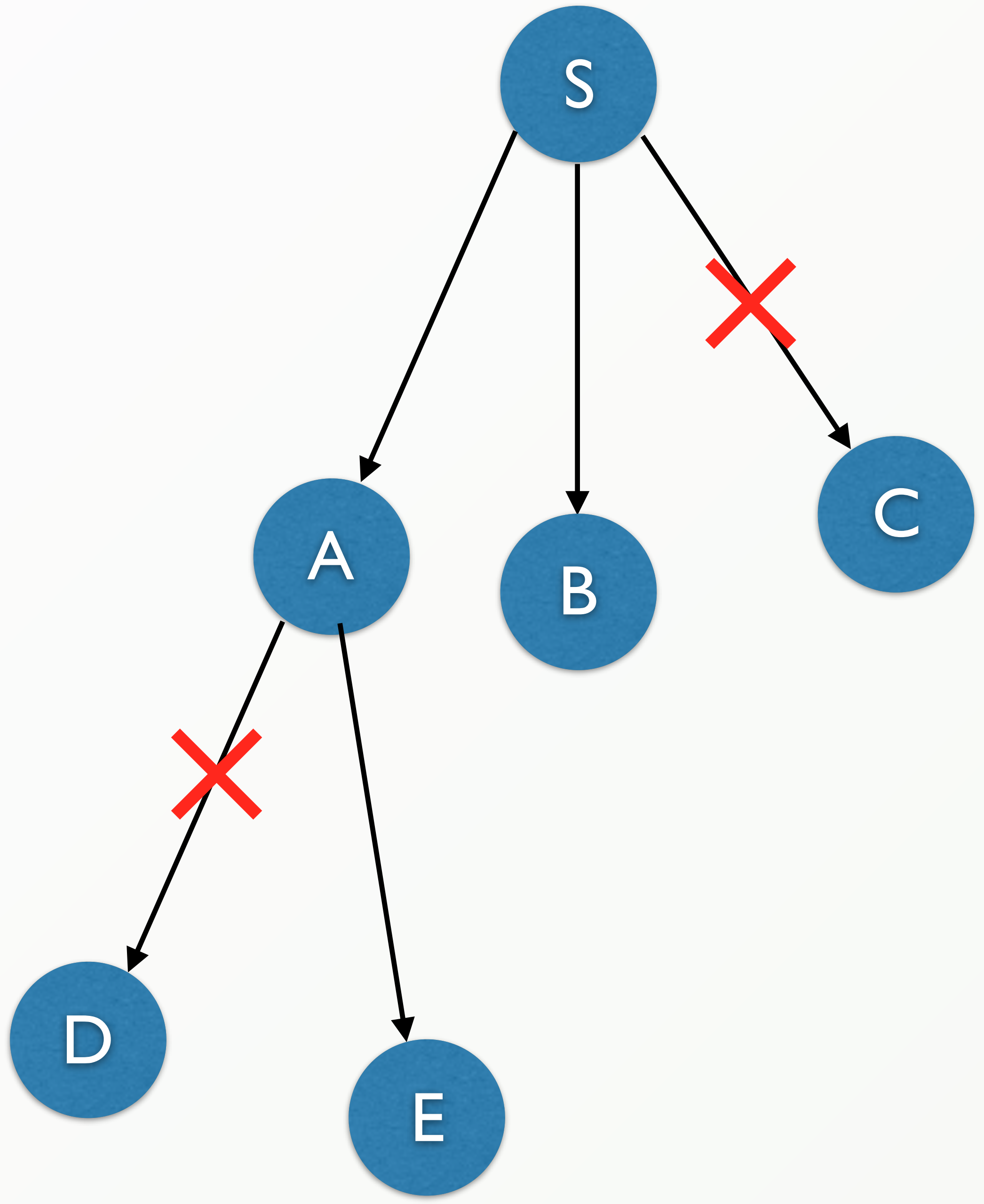
Iterate down two edges with some filters



- We first pick a start vertex (S)
- We collect all edges on S
- We apply filters on edges
- We iterate down one of the new vertices (A)

Traversal

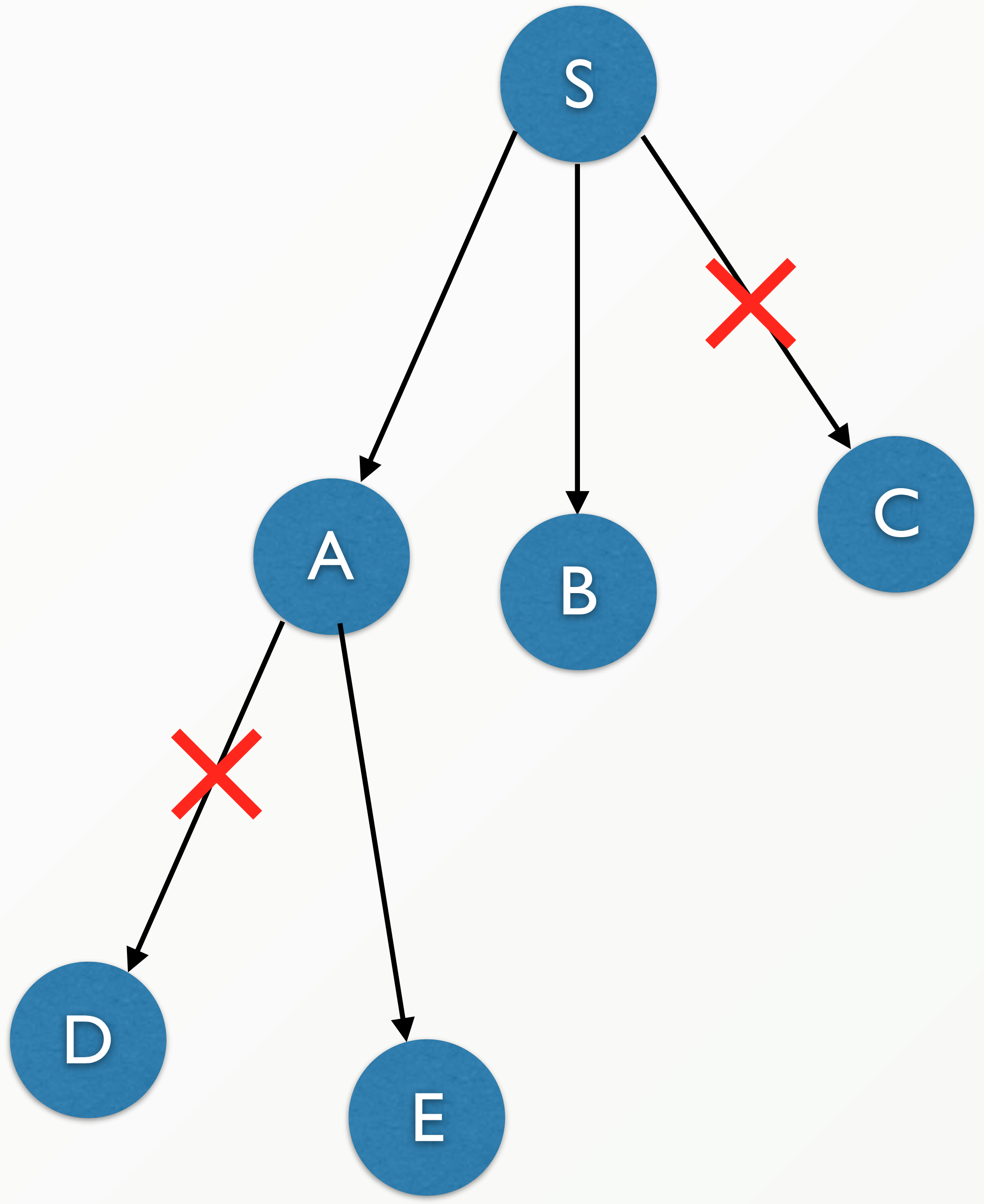
Iterate down two edges with some filters



- We first pick a start vertex (S)
- We collect all edges on S
- We apply filters on edges
- We iterate down one of the new vertices (A)
- We apply filters on edges

Traversal

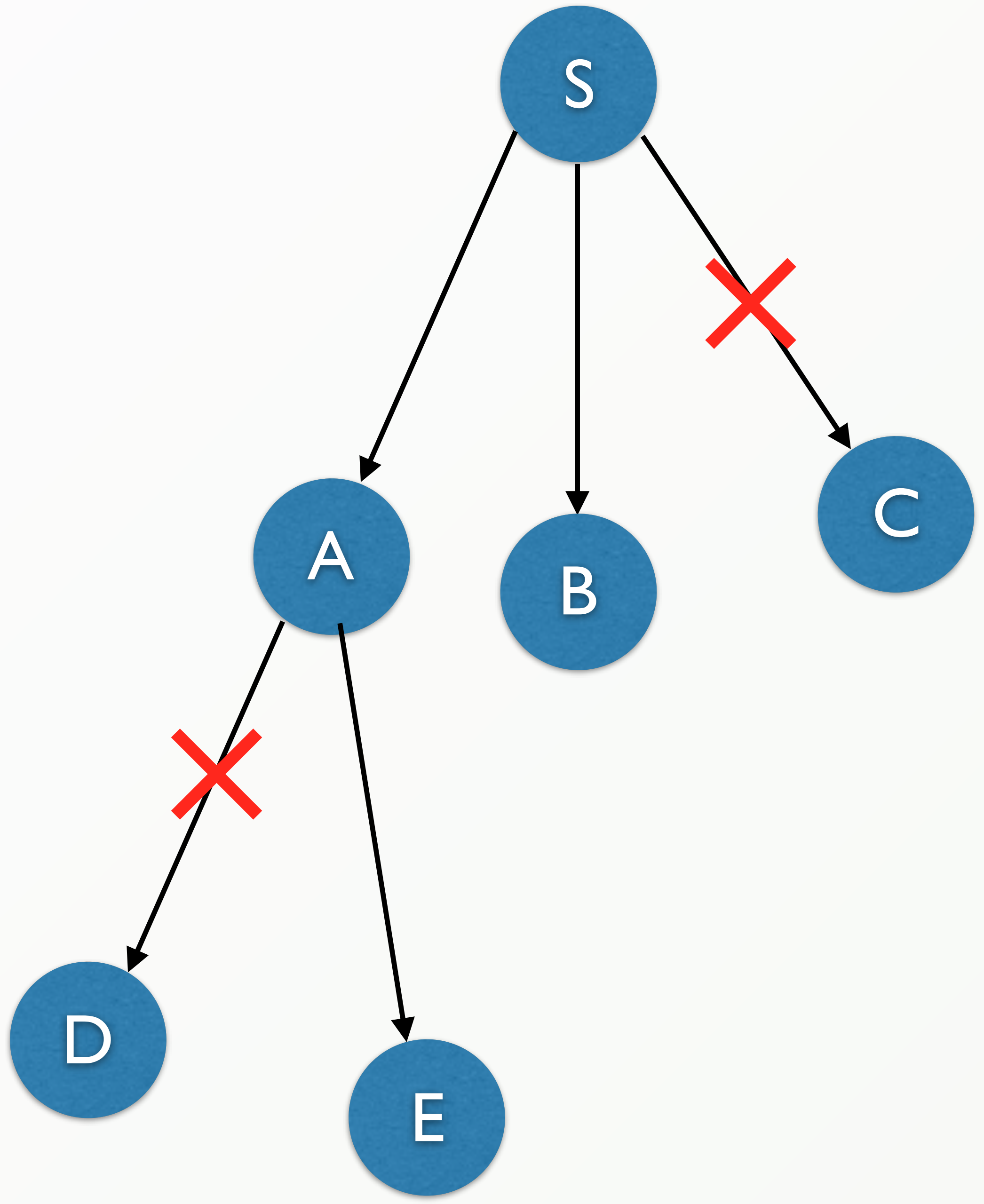
Iterate down two edges with some filters



- We first pick a start vertex (S)
- We collect all edges on S
- We apply filters on edges
- We iterate down one of the new vertices (A)
- We apply filters on edges
- The next vertex (E) is in desired depth. Return the path S -> A -> E

Traversal

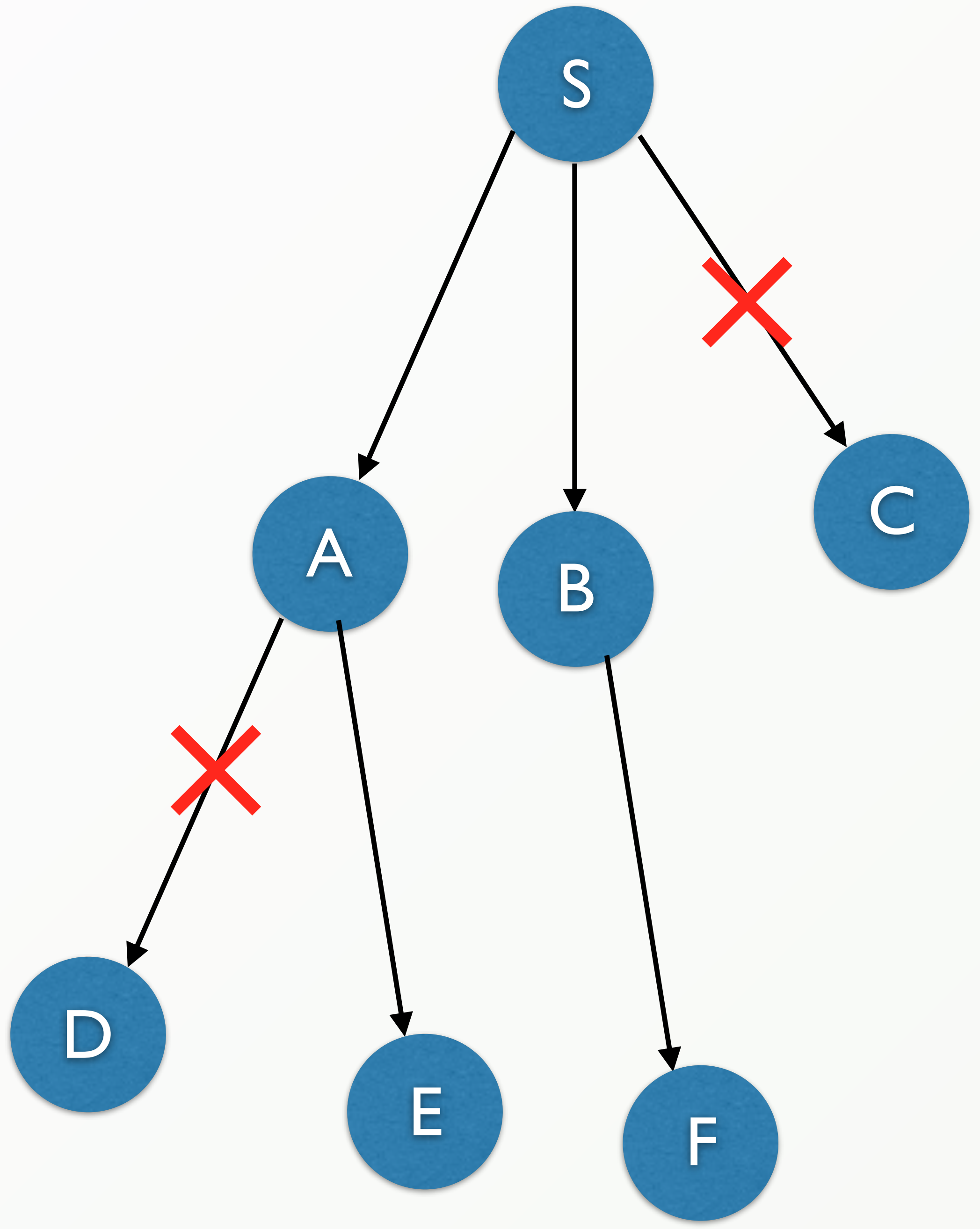
Iterate down two edges with some filters



- We first pick a start vertex (S)
- We collect all edges on S
- We apply filters on edges
- We iterate down one of the new vertices (A)
- We apply filters on edges
- The next vertex (E) is in desired depth. Return the path S -> A -> E
- Go back to the next unfinished vertex (B)

Traversal

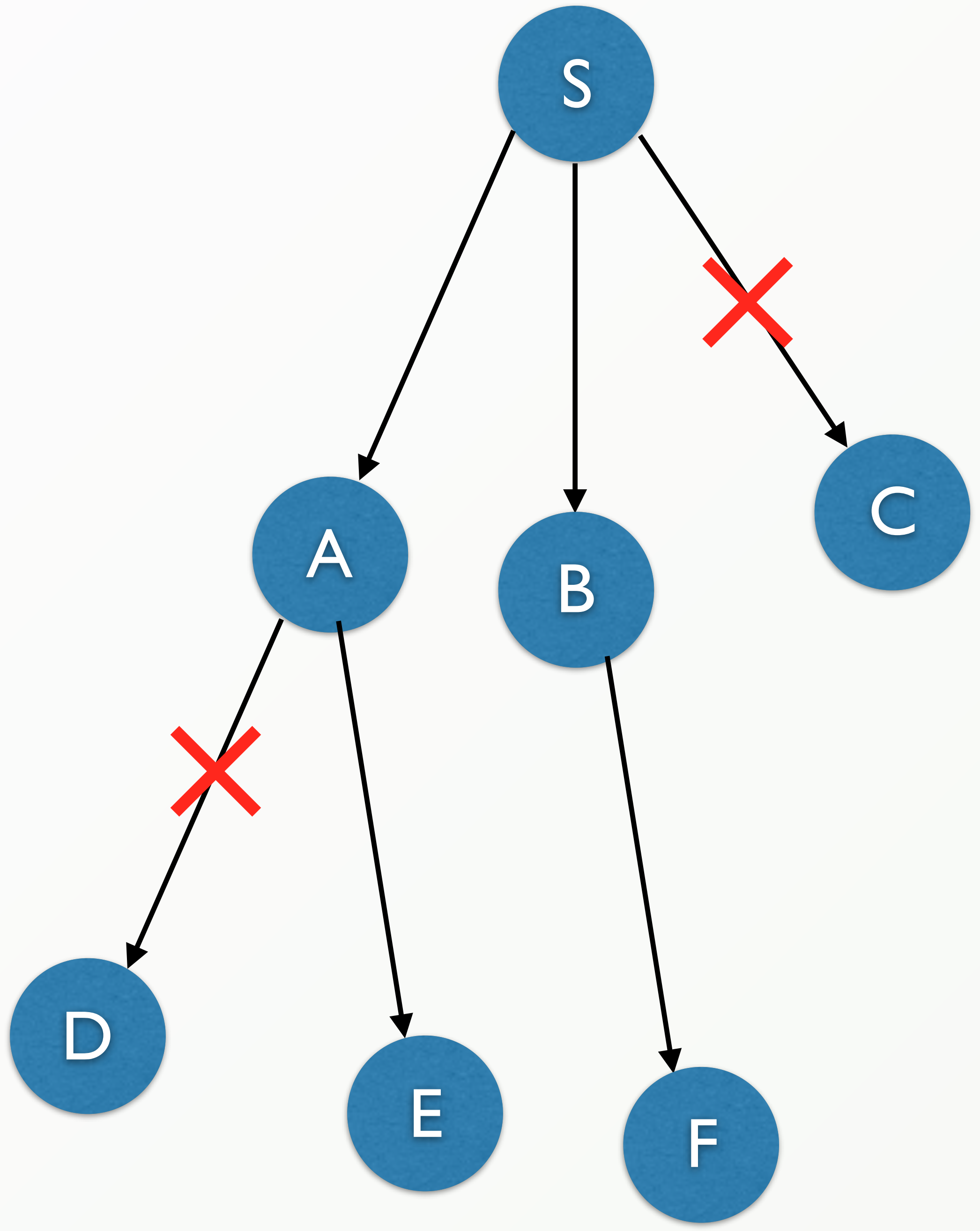
Iterate down two edges with some filters



- We first pick a start vertex (S)
- We collect all edges on S
- We apply filters on edges
- We iterate down one of the new vertices (A)
- We apply filters on edges
- The next vertex (E) is in desired depth. Return the path S -> A -> E
- Go back to the next unfinished vertex (B)
- We iterate down on (B)

Traversal

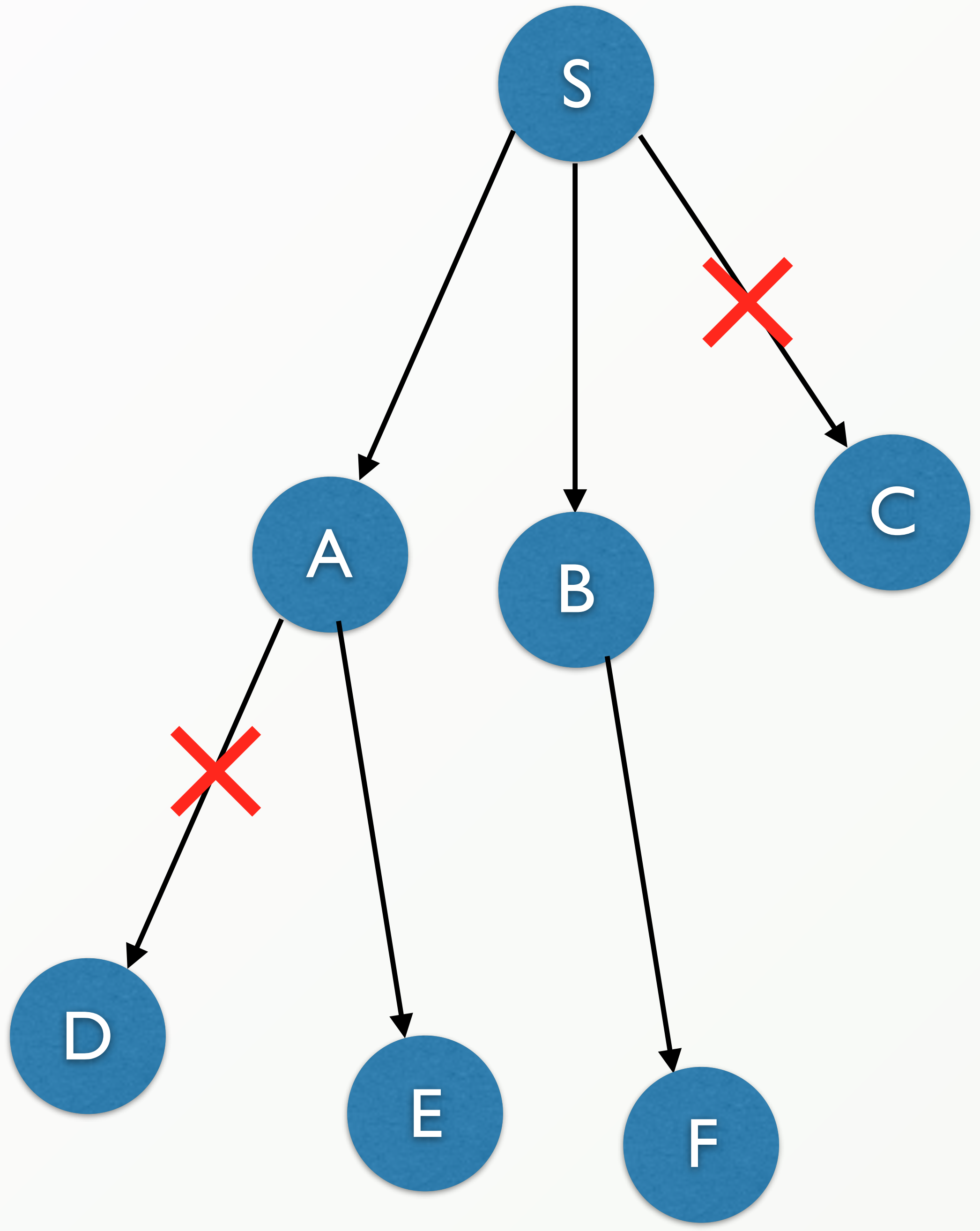
Iterate down two edges with some filters



- We first pick a start vertex (S)
- We collect all edges on S
- We apply filters on edges
- We iterate down one of the new vertices (A)
- We apply filters on edges
- The next vertex (E) is in desired depth. Return the path S -> A -> E
- Go back to the next unfinished vertex (B)
- We iterate down on (B)
- We apply filters on edges

Traversal

Iterate down two edges with some filters



- We first pick a start vertex (S)
- We collect all edges on S
- We apply filters on edges
- We iterate down one of the new vertices (A)
- We apply filters on edges
- The next vertex (E) is in desired depth. Return the path S -> A -> E
- Go back to the next unfinished vertex (B)
- We iterate down on (B)
- We apply filters on edges
- The next vertex (F) is in desired depth. Return the path S -> B -> F

Traversal - Complexity

Once:		o
Find the start vertex	Depends on indexes: Hash:	1
For every depth:		
Find all connected edges	Edge-Index or Index-Free:	1
Filter non-matching edges	Linear in edges:	n
Find connected vertices	Depends on indexes: Hash:	n * 1
Filter non-matching vertices	Linear in vertices:	n
	Only one pass:	3n

Traversal - Complexity

- Linear sounds evil?
  - NOT linear in All Edges O(E)
  - Only Linear in relevant Edges n < E
- Traversals solely scale with their result size.
- They are not effected at all by total amount of data
- BUT: Every depth increases the exponent: $O(3 * n^d)$
- "7 degrees of separation": $3*n^6 < E < 3*n^7$

ArangoDB

- MULTI-MODEL database
  - Stores Documents and Graphs
- Query language AQL
  - Document Queries
  - Graph Queries
  - Joins
  - All can be combined in the same statement
- ACID support including Multi Collection Transactions

AQL

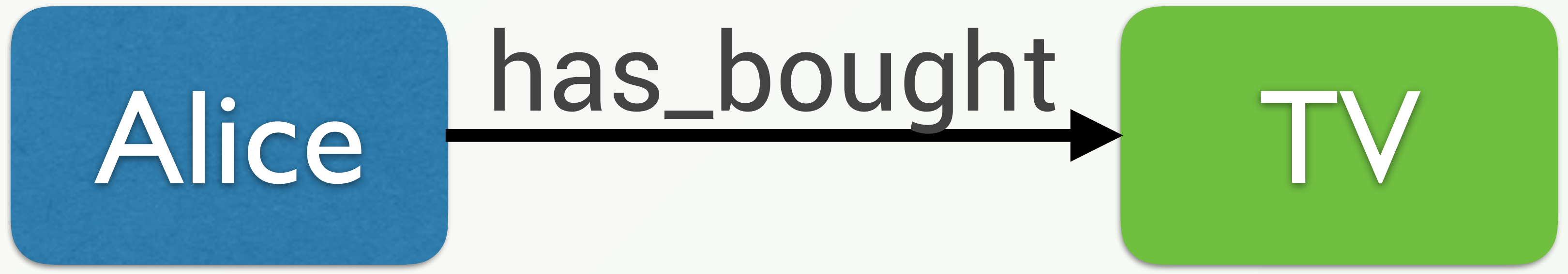
```
FOR user IN users
  RETURN user
```

AQL

```
FOR user IN users
   FILTER user.name == "alice"
   RETURN user
```

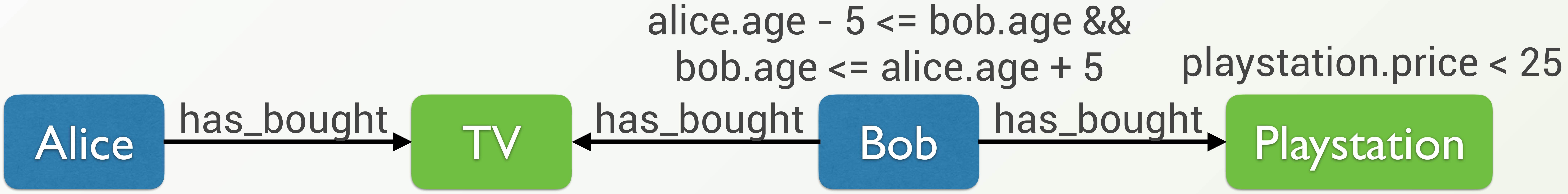
AQL

```
FOR user IN users
   FILTER user.name == "alice"
   FOR product IN OUTBOUND user has_bought
   RETURN product
```



AQL

```
FOR user IN users
  FILTER user.name == "alice"
  FOR recommendation, action, path IN 3 ANY user has_bought
    FILTER path.vertices[2].age <= user.age + 5
    AND path.vertices[2].age >= user.age - 5
    FILTER recommendation.price < 25
    LIMIT 10
    RETURN recommendation
```



Demo Time
Querying basics

First Boost - Vertex Centric Indices

- Remember Complexity? $O(3 * n^d)$
- Filtering of non-matching edges is linear for every depth

- Index all edges based on their vertices and arbitrary other attributes
  - Find initial set of edges in identical time
  - Less / No post-filtering required
  - This decreases the n

Demo Time
Vertex-Centric Indices

Scaling

- Vertex-Centric Indexes help with super-nodes
- But what if the graph is too large for one machine?

- Distribute graph on several machines (sharding)
- How to query it now?
  - No global view of the graph possible any more
  - What about edges between servers?

First let's do the cluster thingy

ArangoDB

❤

MESOSPHERE

MESOSPHERE

Apache MESOS™

Marathon

DC/OS

Demo Time
DC / OS

Is Mesosphere required?

- ArangoDB can run clusters without it
  - Setup Requires manual effort (can be scripted):
    - Configure IP addresses
    - Correct startup ordering

- This works:
  - Automatic Failover (Follower takes over if leader dies)
  - Rebalancing of shards
  - Everything inside of ArangoDB
- This is based on Mesos:
  - Complete self healing
  - Automatic restart of ArangoDBs (on new machines)

➡ We suggest you have someone on call

Now distribute the graph

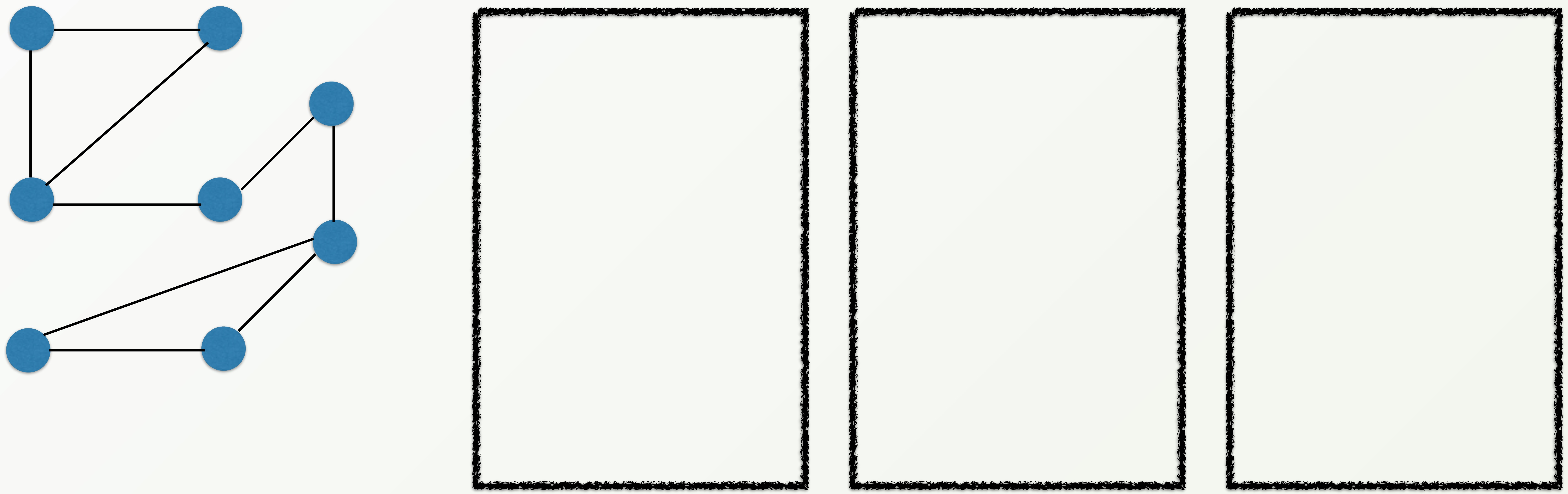
Dangers of Sharding

- Only parts of the graph on every machine
- Neighboring vertices may be on different machines
- Even edges could be on other machines than their vertices

- Queries need to be executed in a distributed way
- Result needs to be merged locally

Random Distribution

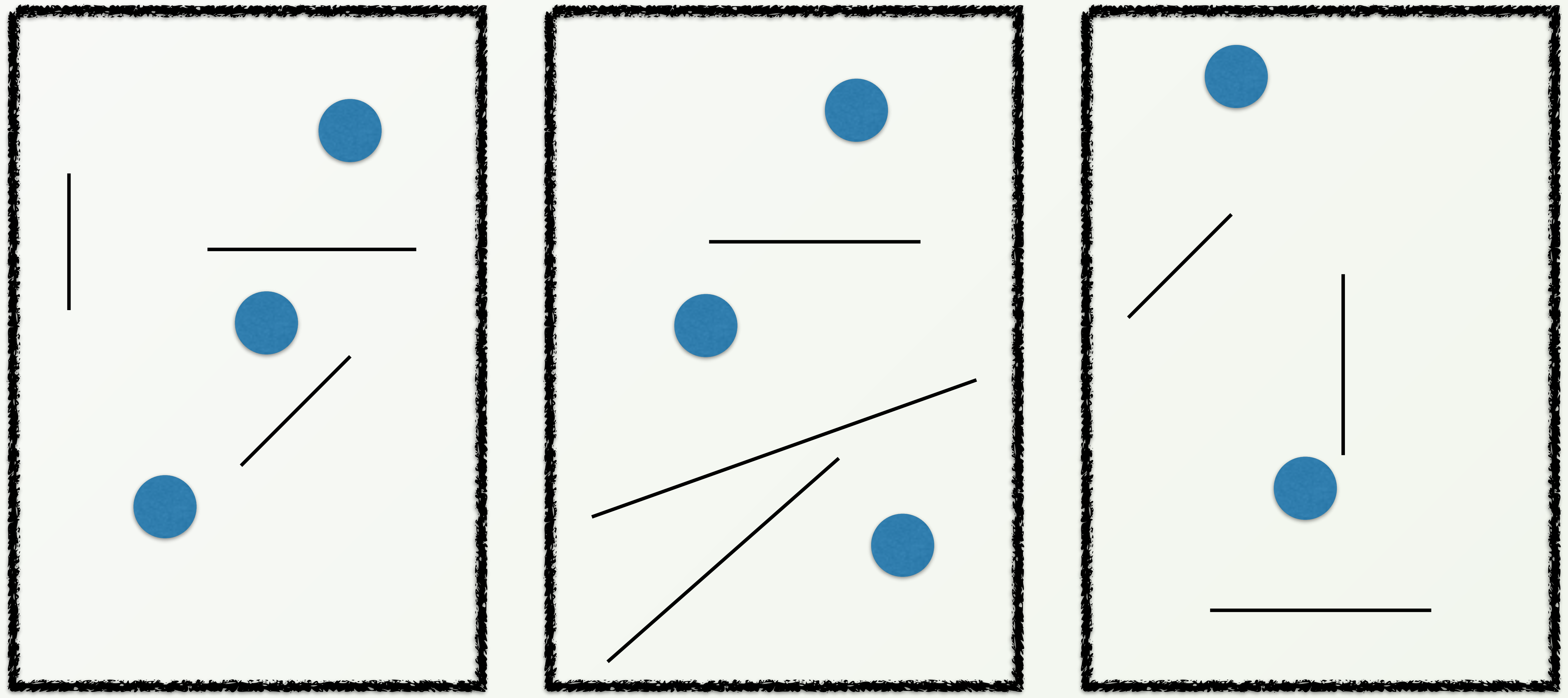
- Advantages:
  - every server takes an equal portion of data
  - easy to realize
  - no knowledge about data required
  - always works
- Disadvantages:
  - Neighbors on different machines
  - Probably edges on other machines than their vertices
  - A lot of network overhead is required for querying



Random Distribution

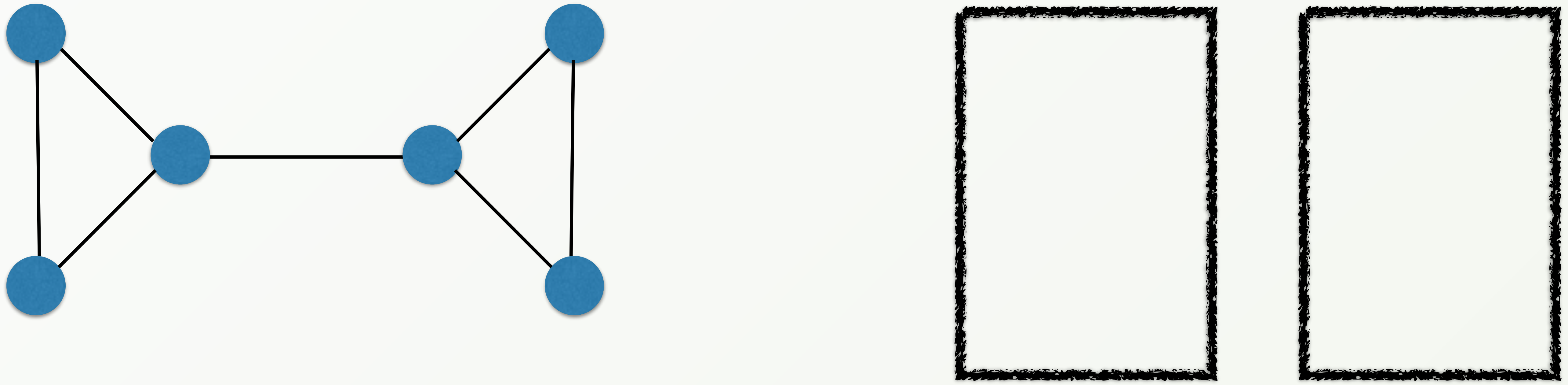
- Advantages:
  - every server takes an equal portion of data
  - easy to realize
  - no knowledge about data required
  - always works

- Disadvantages:
  - Neighbors on different machines
  - Probably edges on other machines than their vertices
  - A lot of network overhead is required for querying



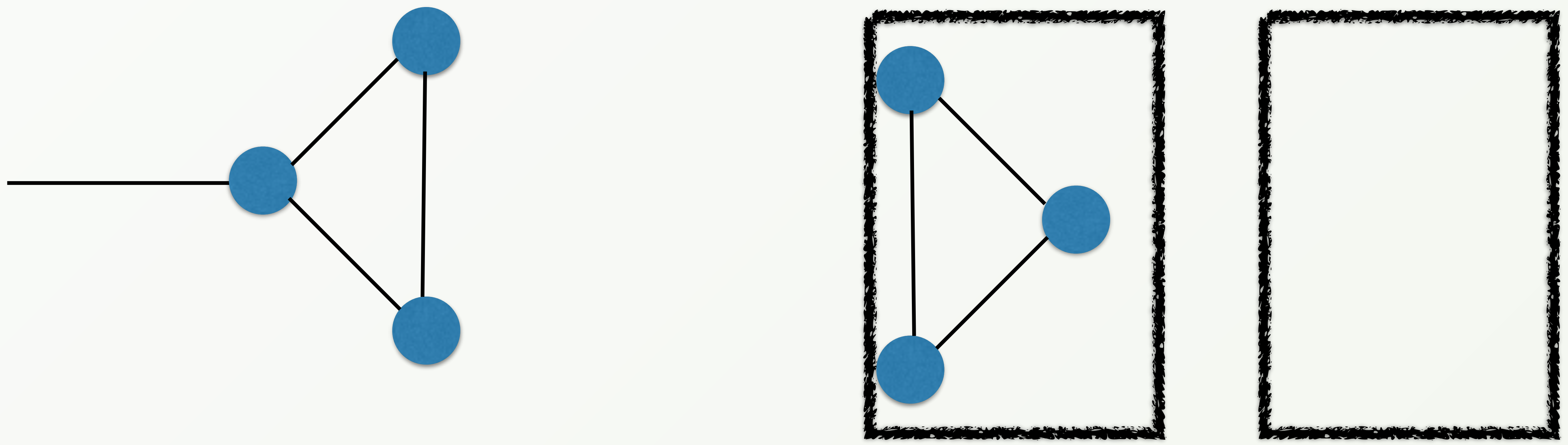
Index-Free Adjacency

- Used by most other graph databases
- Every vertex maintains two lists of it's edges (IN and OUT)
  - Do not use an index to find edges
  - How to shard this?



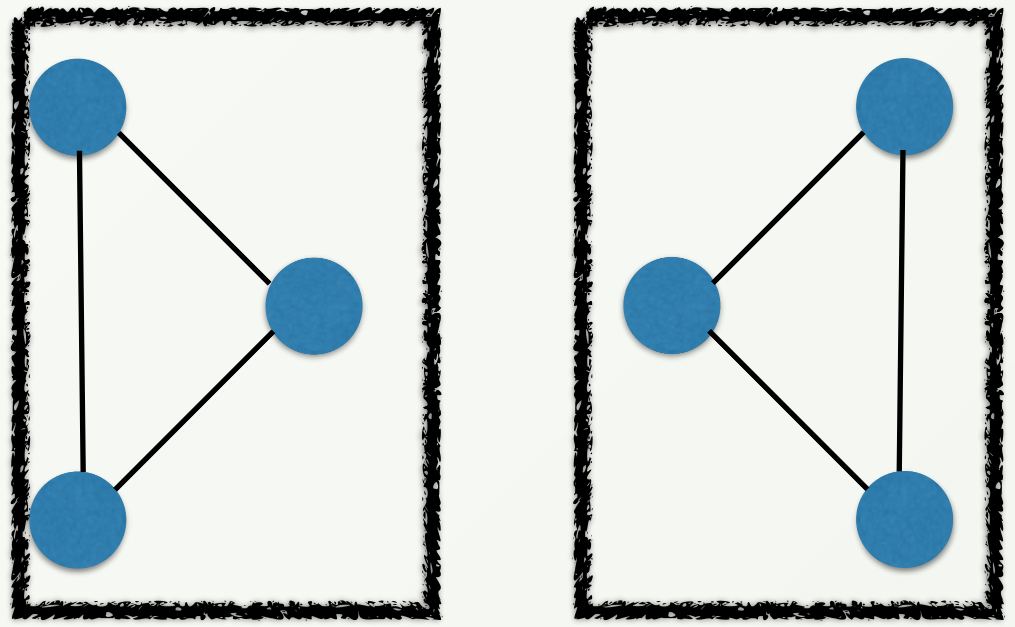
Index-Free Adjacency

- Used by most other graph databases
- Every vertex maintains two lists of it's edges (IN and OUT)
  - Do not use an index to find edges
  - How to shard this?



Index-Free Adjacency

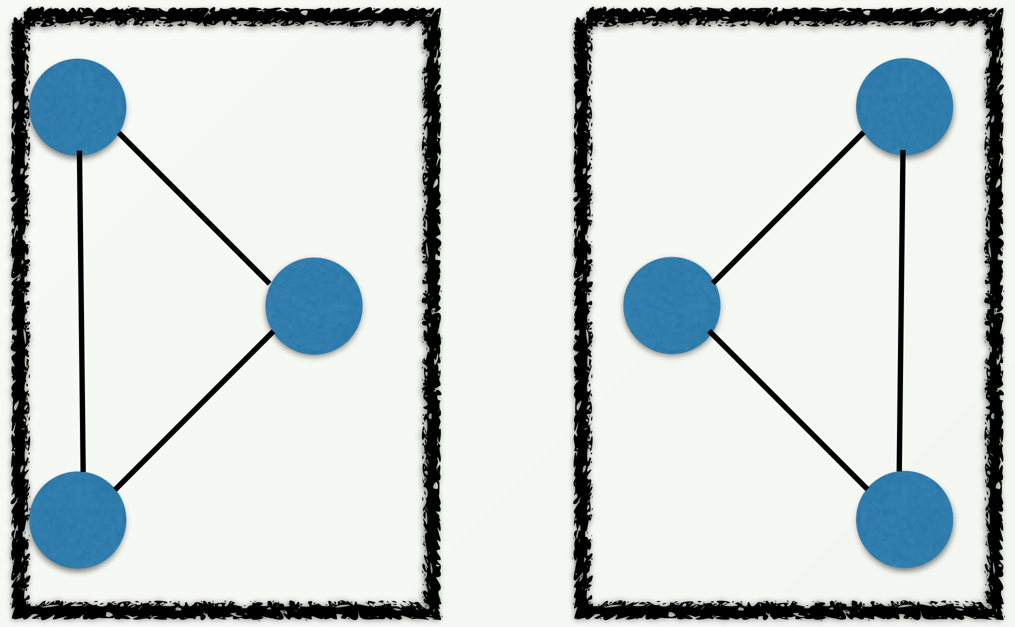
- Used by most other graph databases
- Every vertex maintains two lists of it's edges (IN and OUT)
  - Do not use an index to find edges
  - How to shard this?



Index-Free Adjacency

- Used by most other graph databases
- Every vertex maintains two lists of it's edges (IN and OUT)
  - Do not use an index to find edges
  - How to shard this?

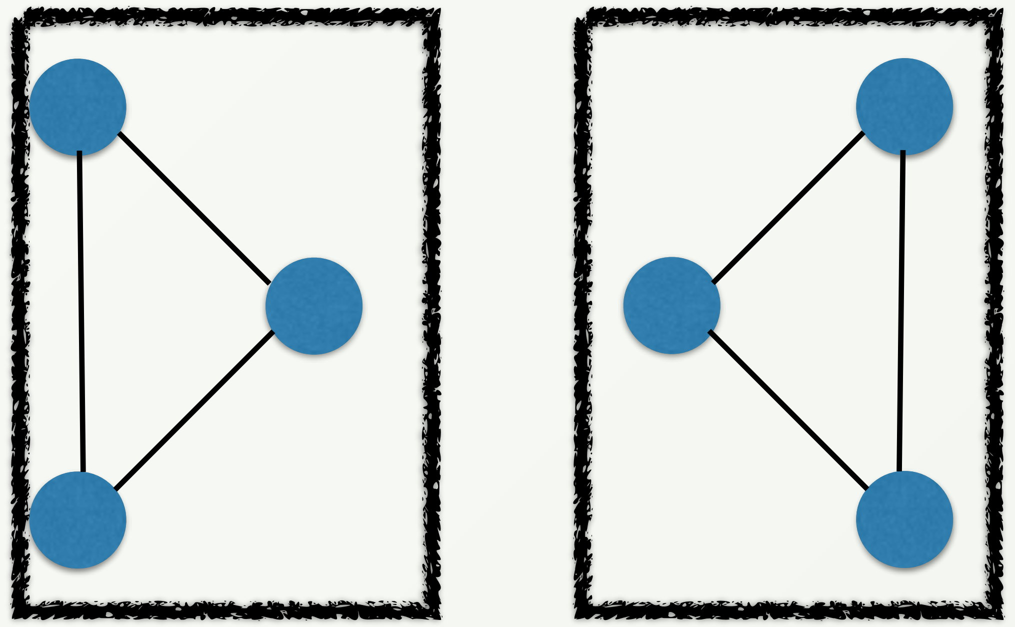
????



Index-Free Adjacency

- Used by most other graph databases
- Every vertex maintains two lists of it's edges (IN and OUT)
  - Do not use an index to find edges
  - How to shard this?

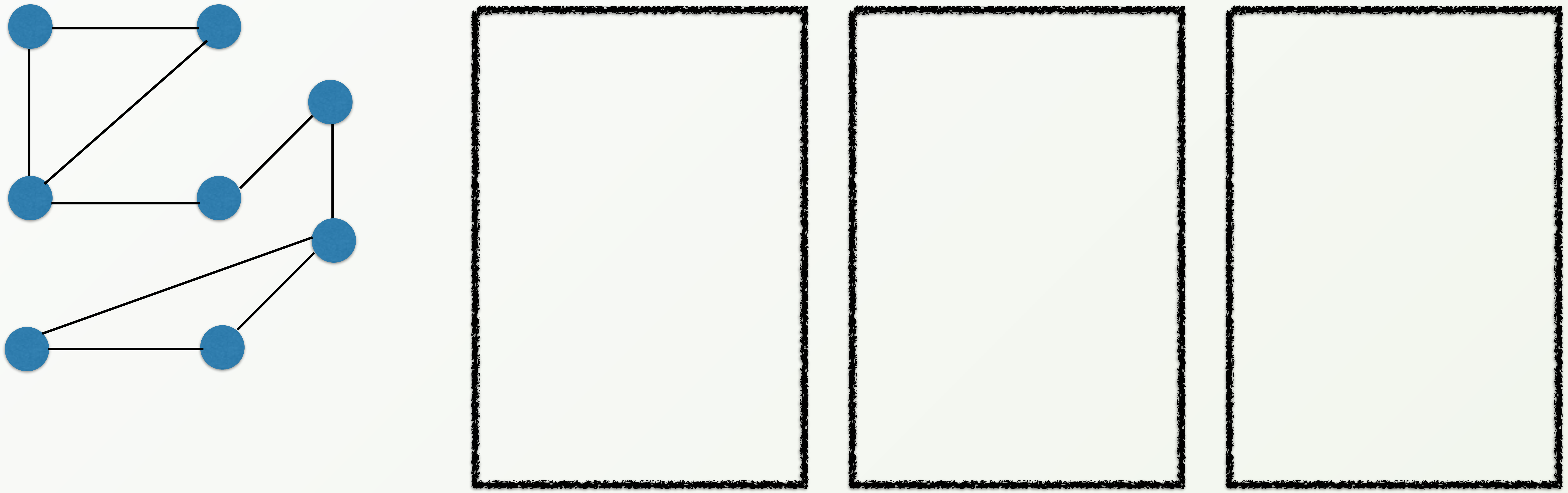
????



- ArangoDB uses an hash-based EdgeIndex (O(1) - lookup)
  - The vertex is independent of it's edges
  - It can be stored on a different machine

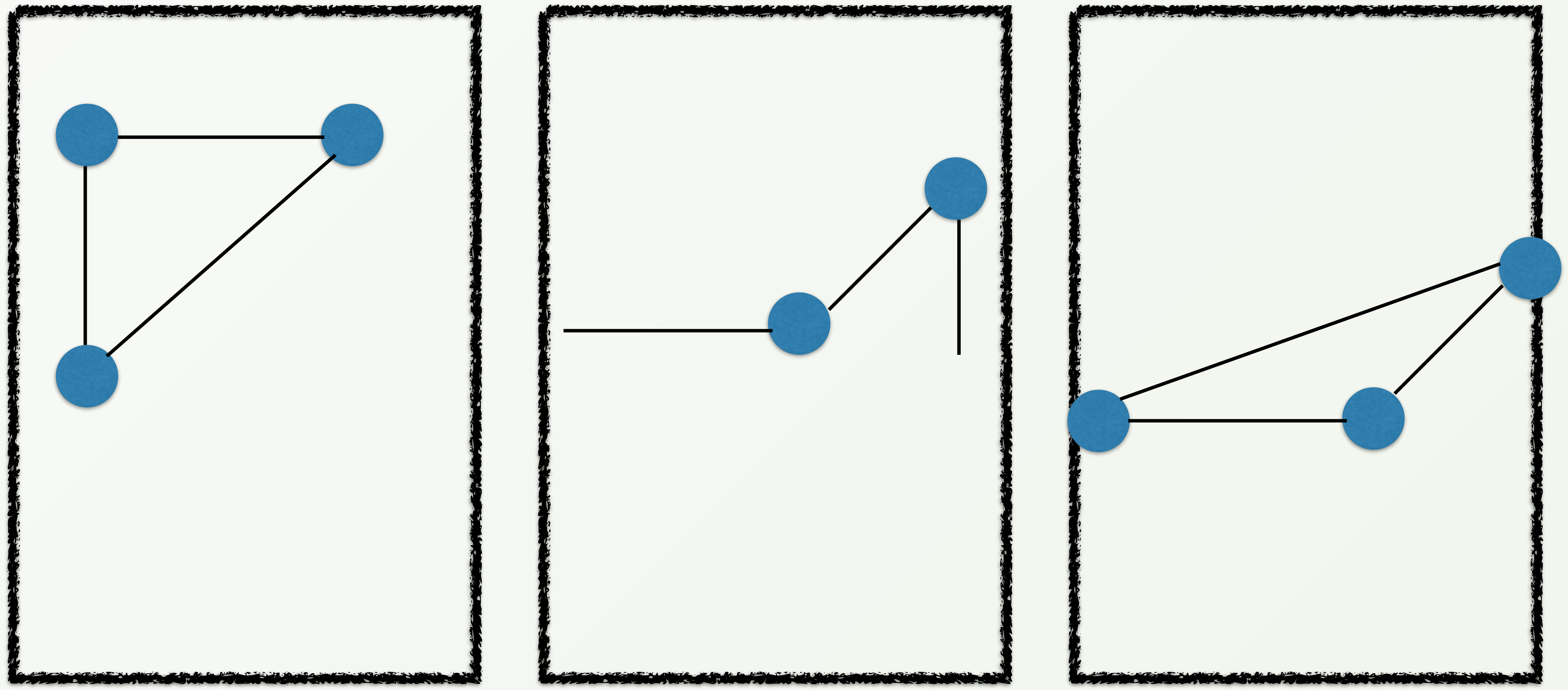
Domain Based Distribution

- Many Graphs have a natural distribution
  - By country/region for People
  - By tags for Blogs
  - By category for Products
- Most edges in same group
- Rare edges between groups



Domain Based Distribution

- Many Graphs have a natural distribution
  - By country/region for People
  - By tags for Blogs
  - By category for Products
- Most edges in same group
- Rare edges between groups



Domain Based Distribution

- Many Graphs have a natural distribution
  - By country/region for People
  - By tags for Blogs
  - By category for Products
- Most edges in same group
- Rare edges between groups

ArangoDB
Enterprise

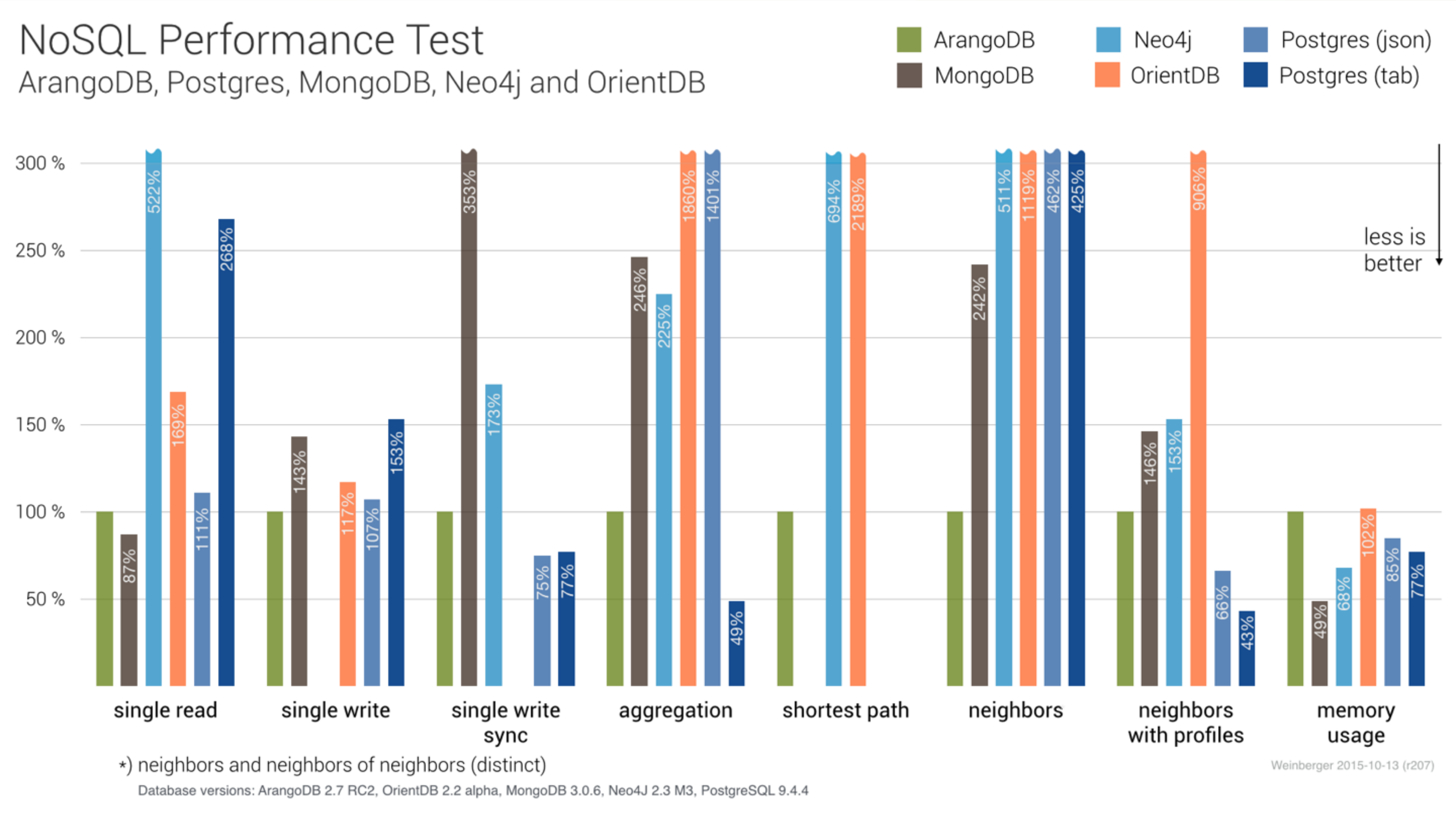
uses Domain Knowledge
for short-cuts

Sneak Preview

ArangoDB
Enterprise

SmartGraphs

Benchmark Comparison



*) neighbors and neighbors of neighbors (distinct)

Database versions: ArangoDB 2.7 RC2, OrientDB 2.2 alpha, MongoDB 3.0.6, Neo4J 2.3 M3, PostgreSQL 9.4.4

Weinberger 2015-10-13 (r207)

Source: https://www.arangodb.com/2015/10/benchmark-postgresql-mongodb-arangodb/

Thank you

- Further questions?
  - Follow us on twitter: @arangodb
  - Join our slack: slack.arangodb.com

  - Follow me on twitter/github: @mchacki
  - Write me a mail: michael@arangodb.com