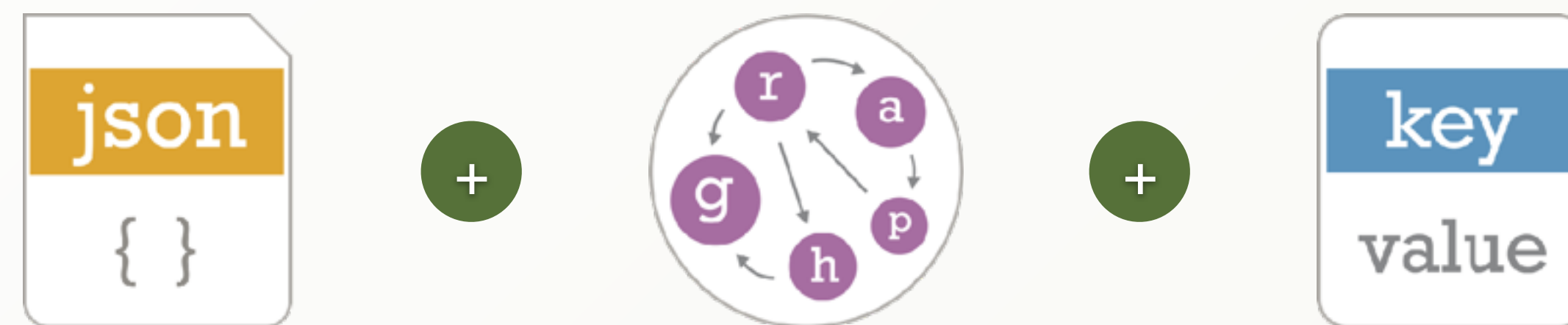




*Handling Billions Of Edges in  
a Graph Database*



# About me

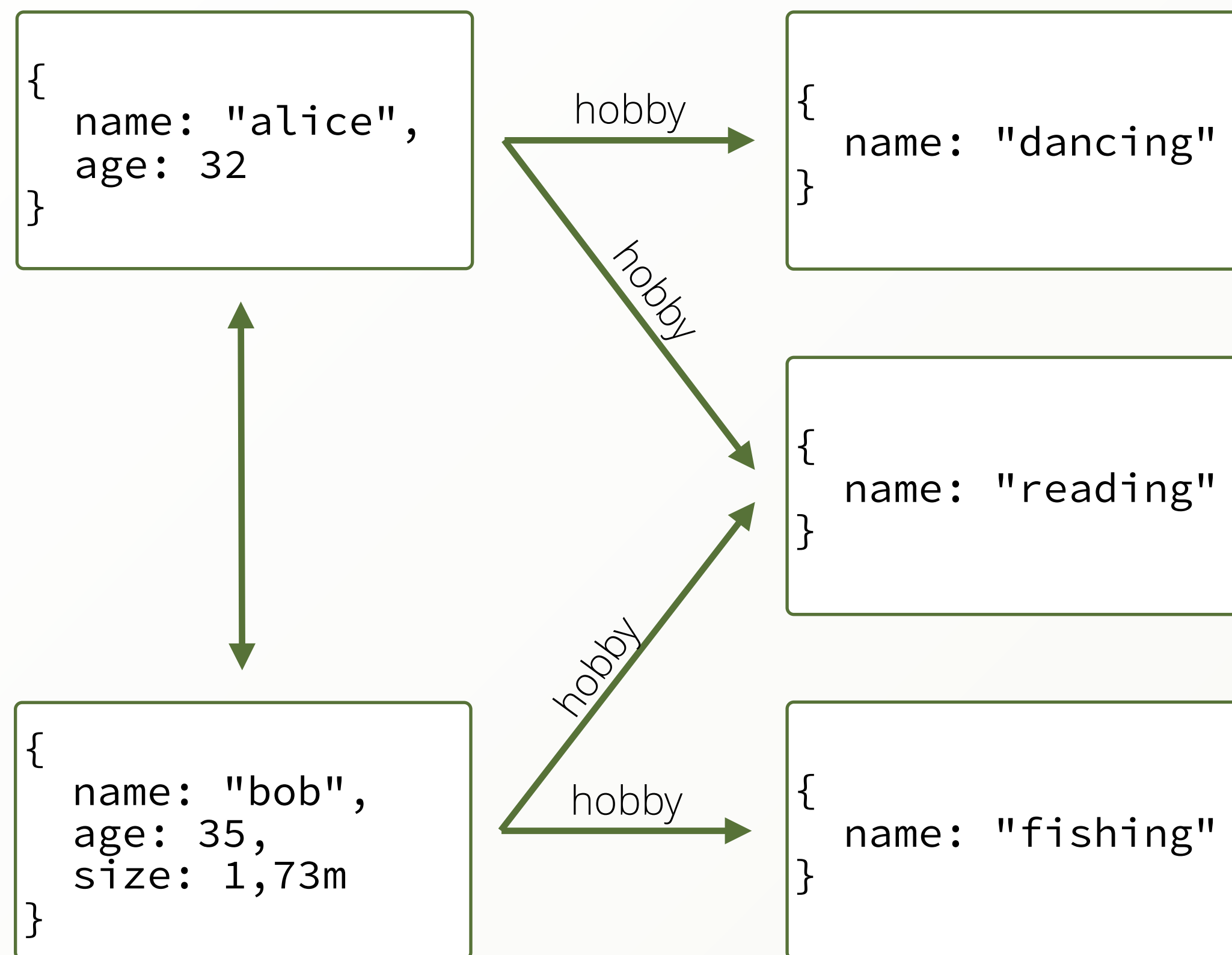
---

- ▶ Michael Hackstein
- ▶ ArangoDB Core Team
  - ▶ Graph visualisation
  - ▶ Graph features
  - ▶ SmartGraphs
- ▶ Host of cologne.js
- ▶ Master's Degree (spec. Databases and Information Systems)



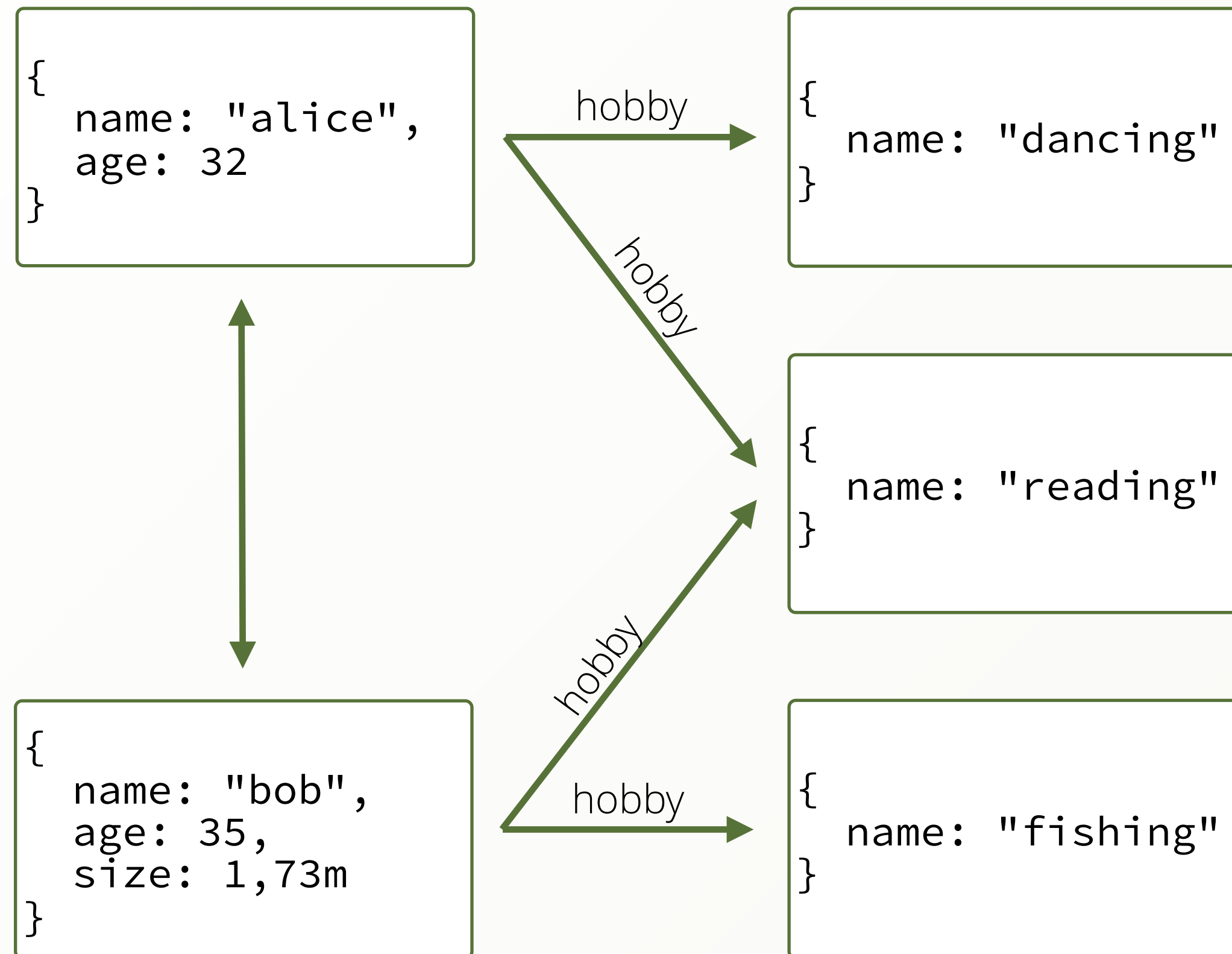
# What are Graph Databases

---



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

# What are Graph Databases

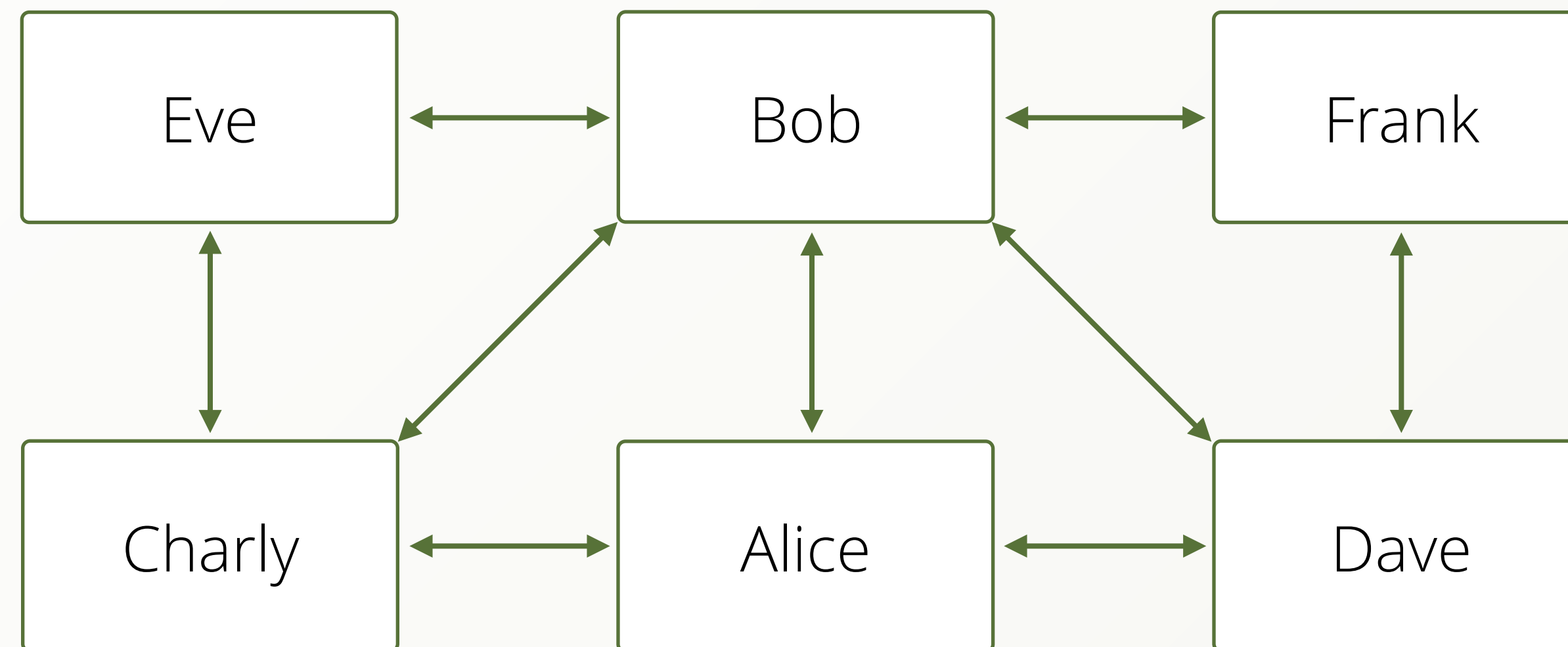


- ▶ Schema-free Objects (Vertices)
- ▶ Relations between them (Edges)
- ▶ Edges have a direction
- ▶ Edges can be queried in both directions
- ▶ Easily query a range of edges (2 to 5)
- ▶ Undefined number of edges (1 to \*)
- ▶ Shortest Path between two vertices

# Typical Graph Queries

---

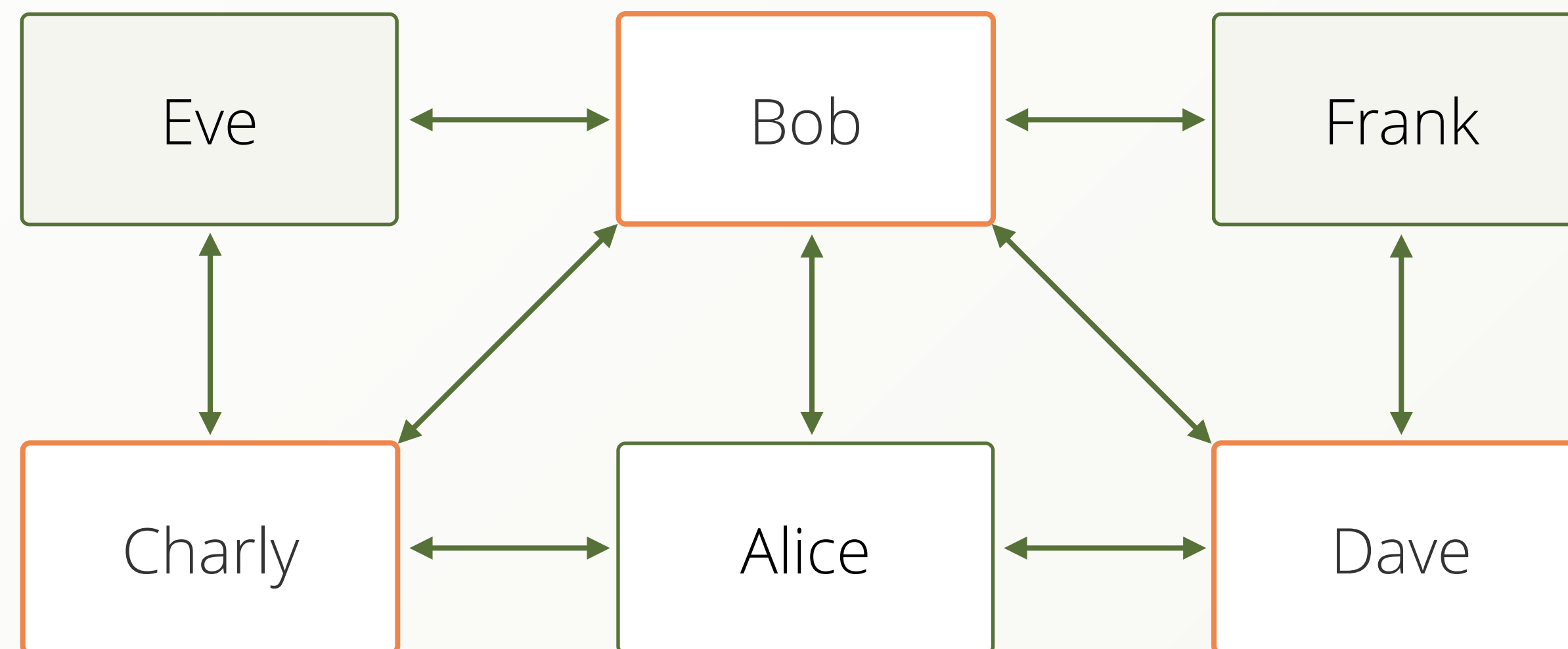
- ▶ Give me all friends of Alice



# Typical Graph Queries

---

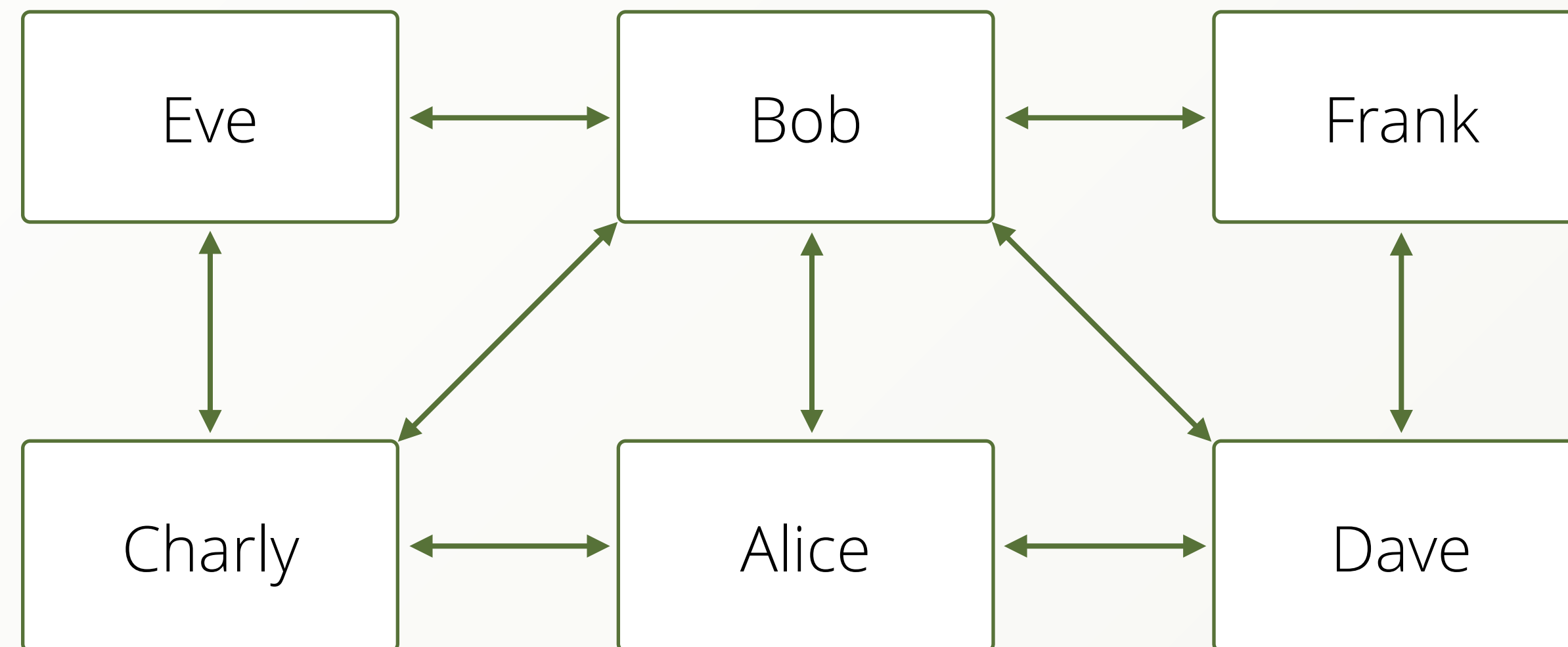
- ▶ Give me all friends of Alice



# Typical Graph Queries

---

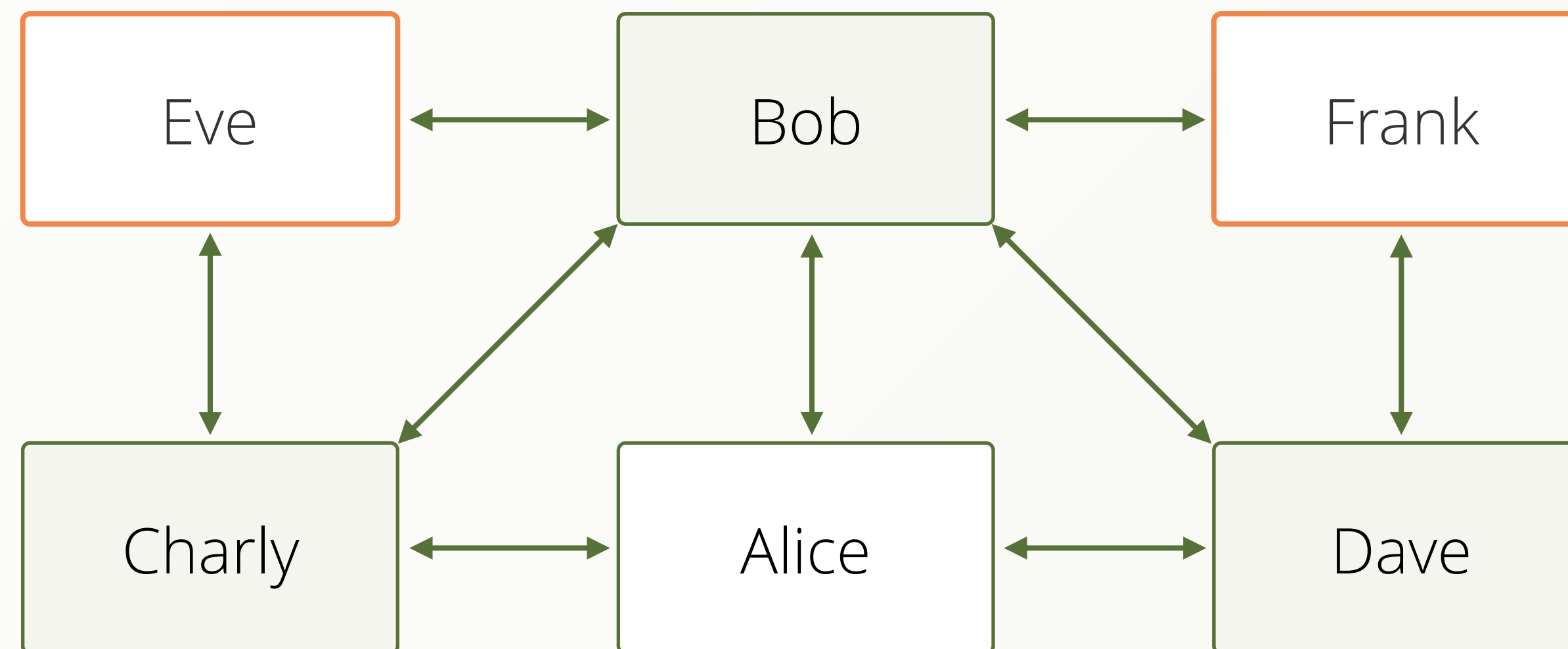
- ▶ Give me all friends-of-friends of *Alice*



# Typical Graph Queries

---

- ▶ Give me all friends-of-friends of *Alice*

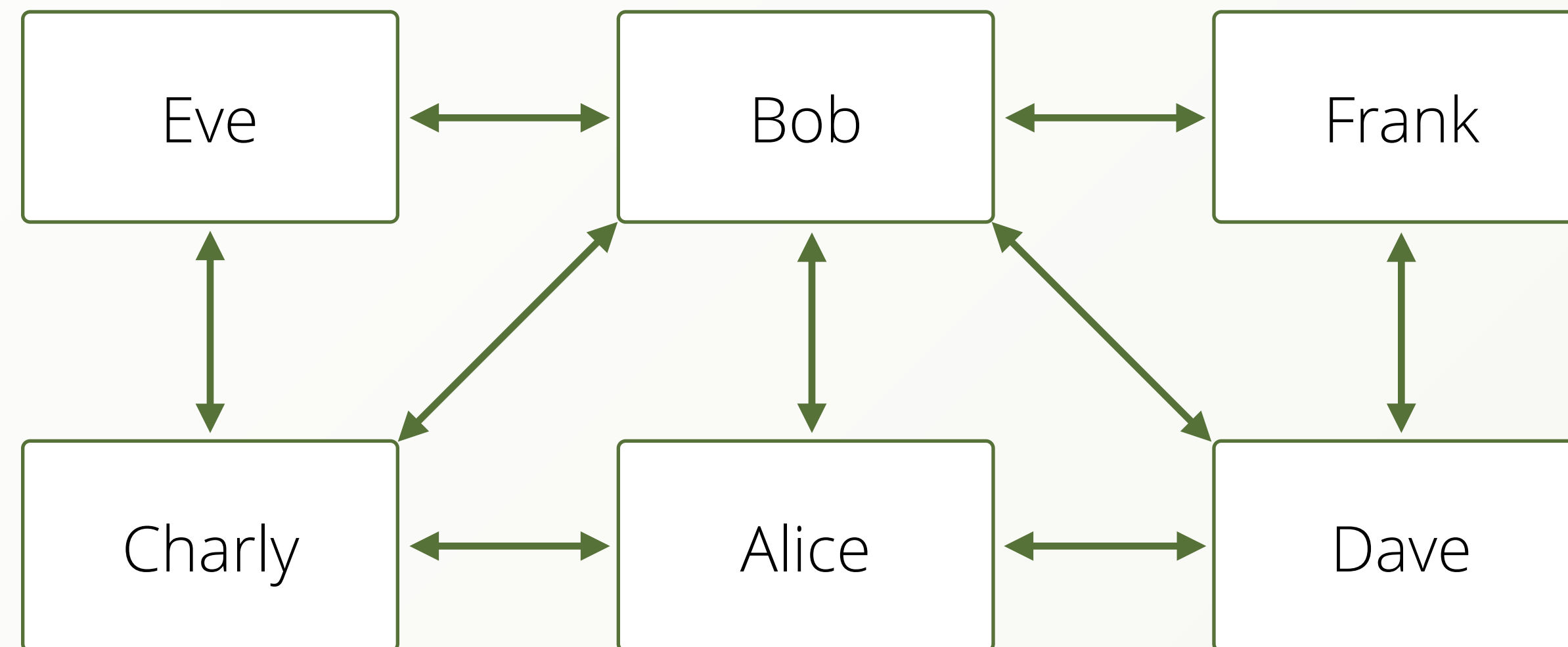




# Typical Graph Queries

---

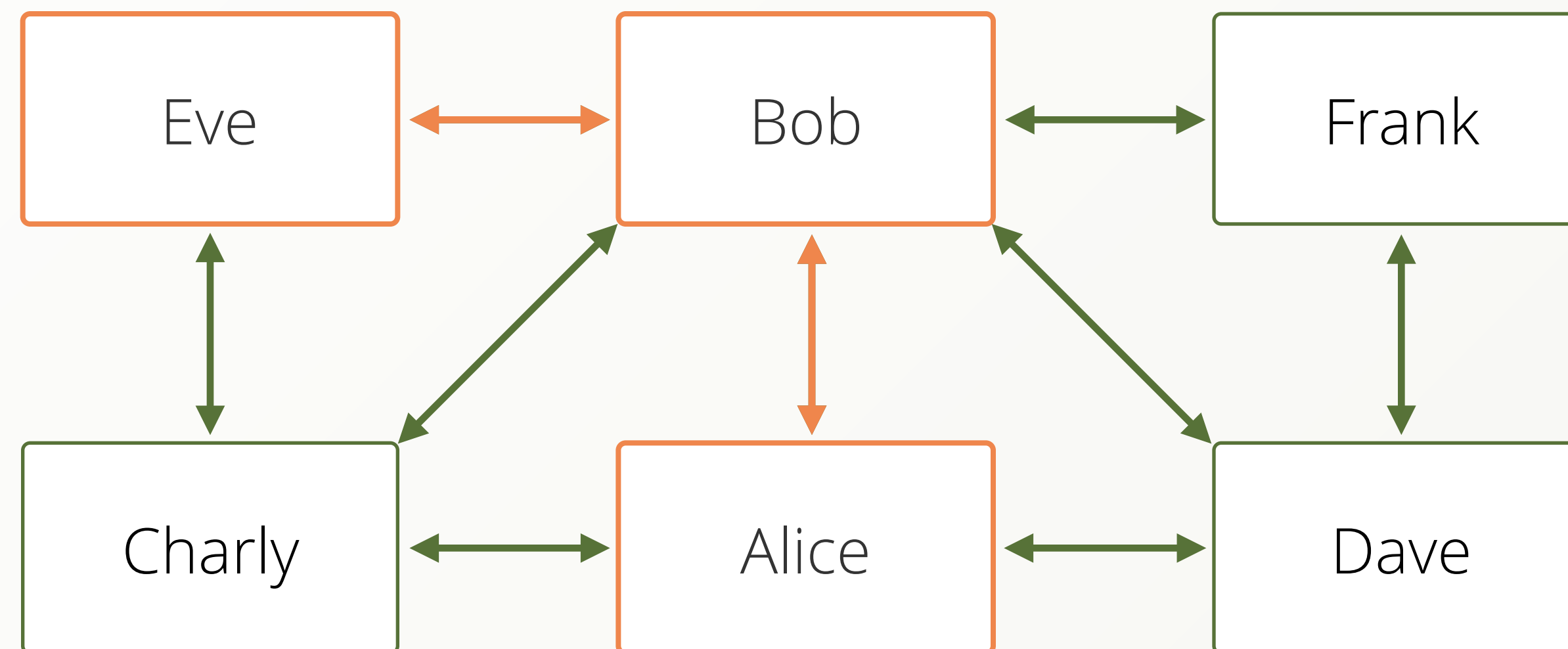
- ▶ What is the linking path between Alice and Eve



# Typical Graph Queries

---

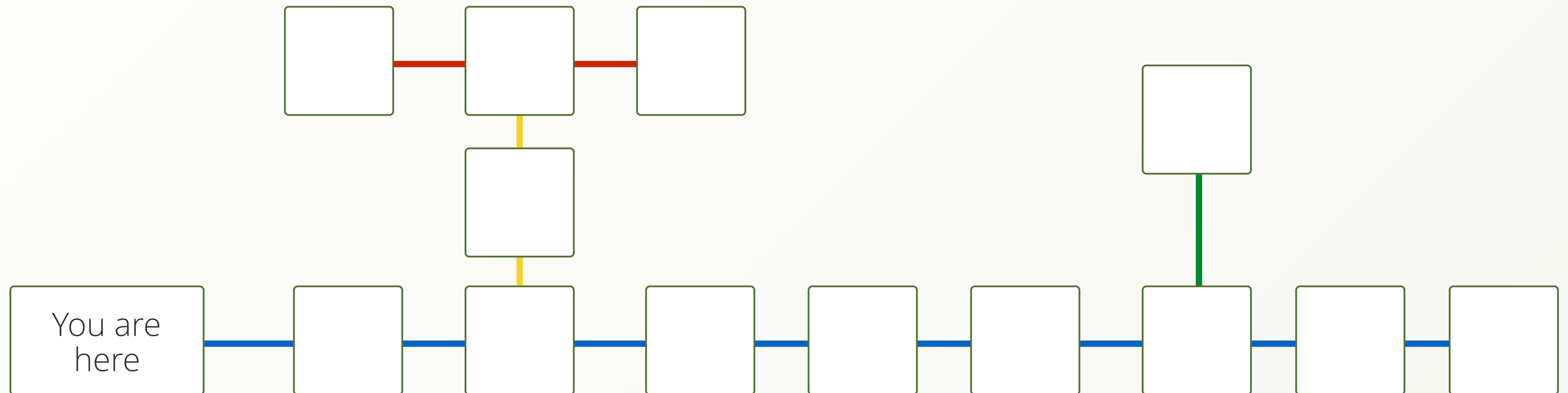
- ▶ What is the linking path between Alice and Eve



# Typical Graph Queries

---

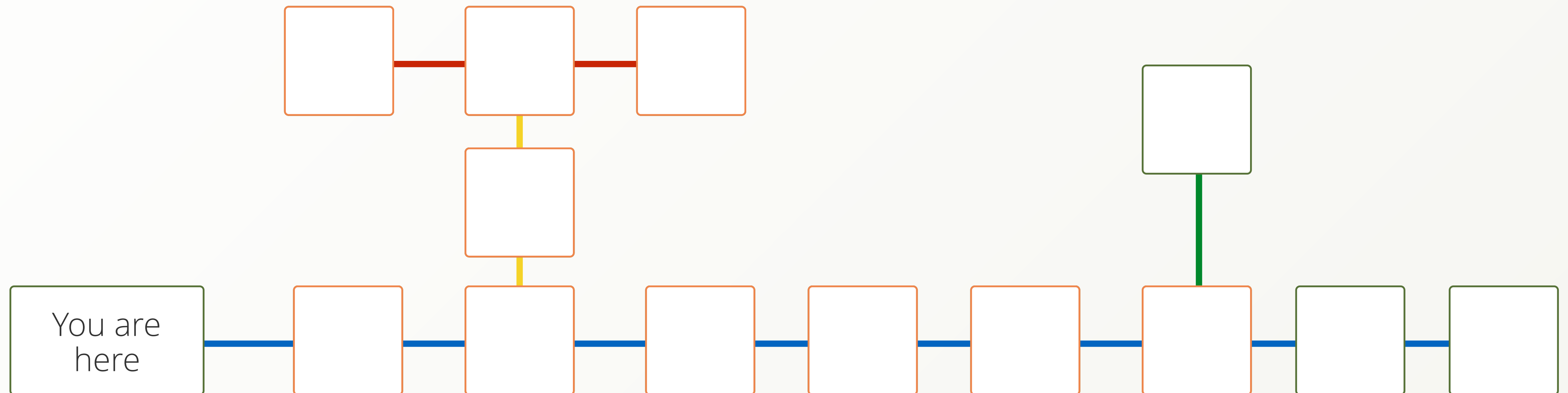
- ▶ Which **Train Stations** can I reach if I am allowed to drive **a distance of at most 6 stations** on my ticket



# Typical Graph Queries

---

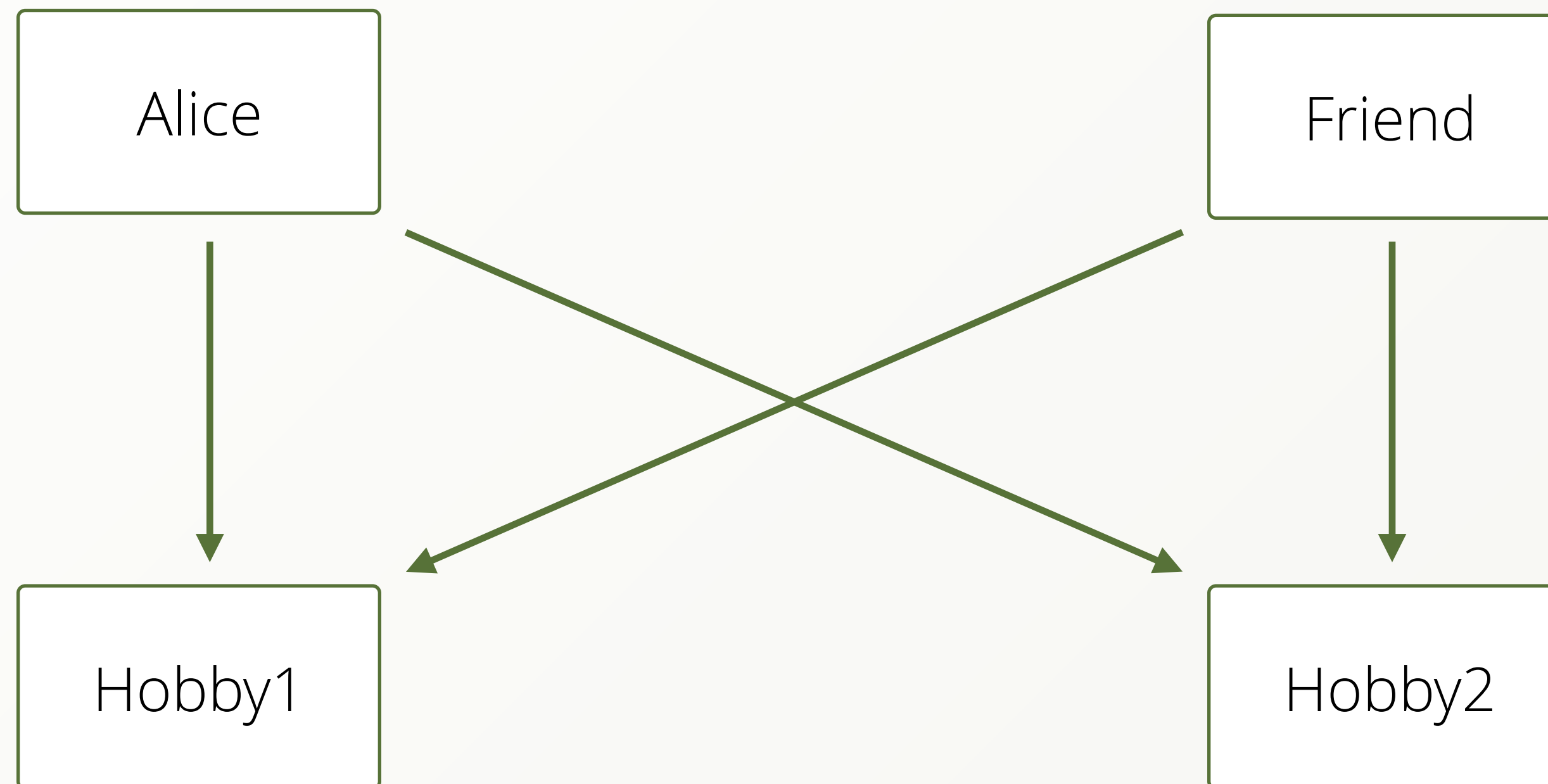
- ▶ Which **Train Stations** can I reach if I am allowed to drive **a distance of at most 6 stations** on my ticket



# Typical Graph Queries: Pattern Matching

---

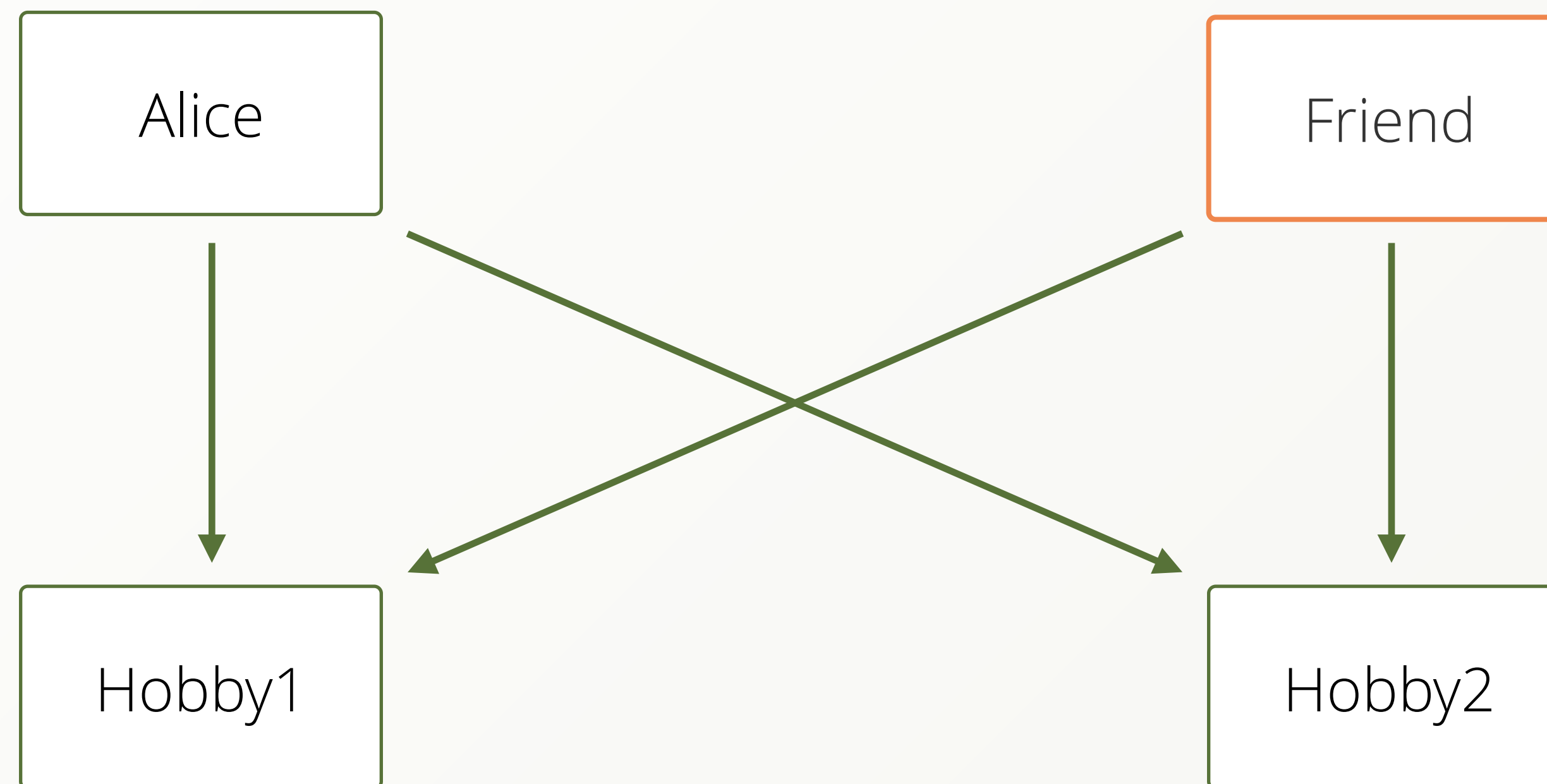
- ▶ Give me all **users** that share two **hobbies** with **Alice**



# Typical Graph Queries: Pattern Matching

---

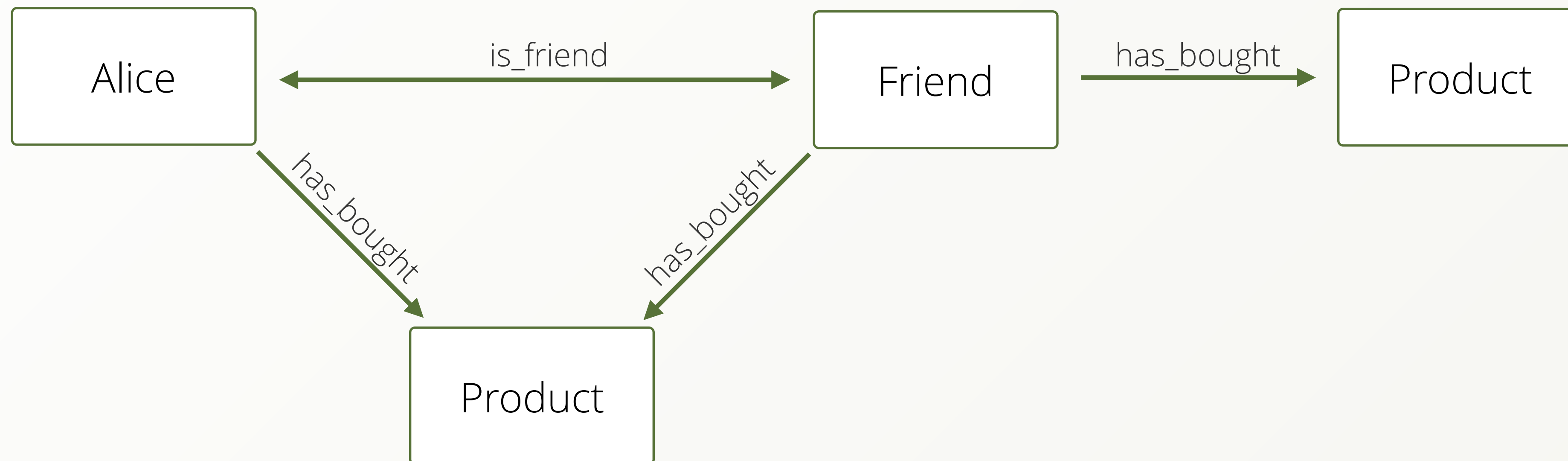
- ▶ Give me all **users** that share two **hobbies** with **Alice**



# Typical Graph Queries: Pattern Matching

---

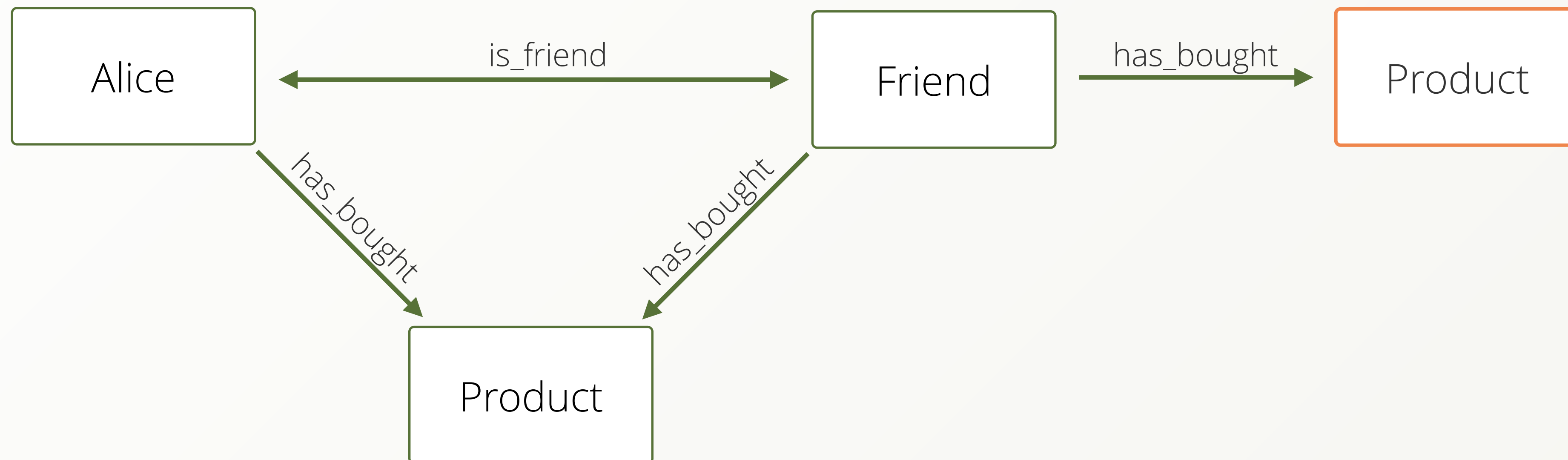
- ▶ Give me all products that at least one of my friends has bought together with the products I already own, ordered by how many friends have bought it and the products rating, but only 20 of them.



# Typical Graph Queries: Pattern Matching

---

- ▶ 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

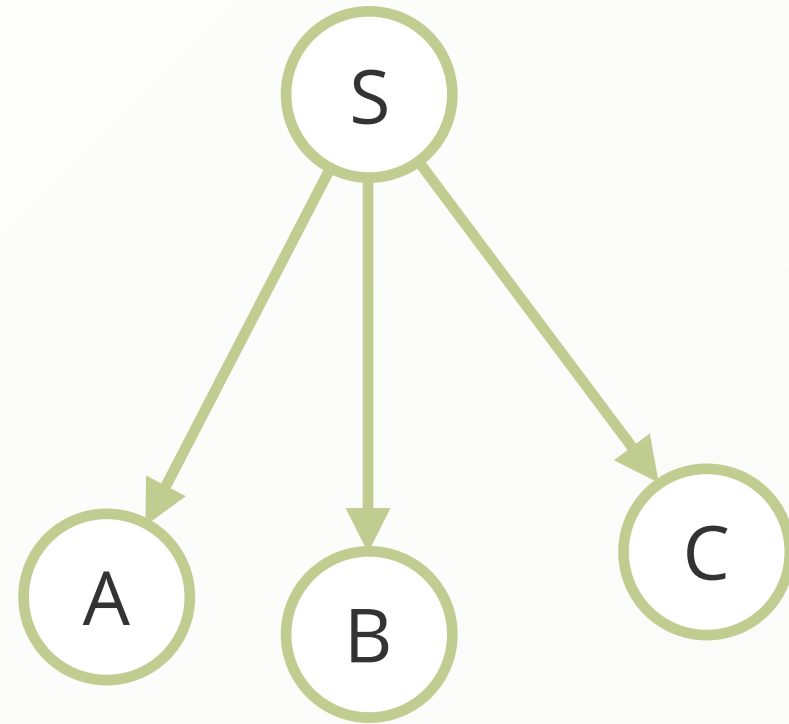
---

- ▶ We first pick a start vertex (S)



# Traversal - Iterate down two edges with some filters

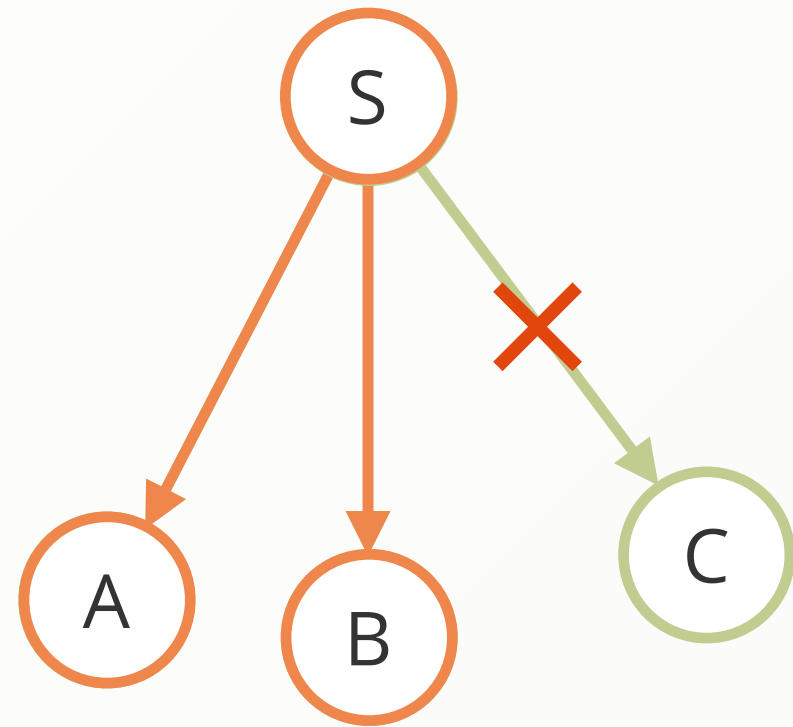
---



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

# Traversal - Iterate down two edges with some filters

---

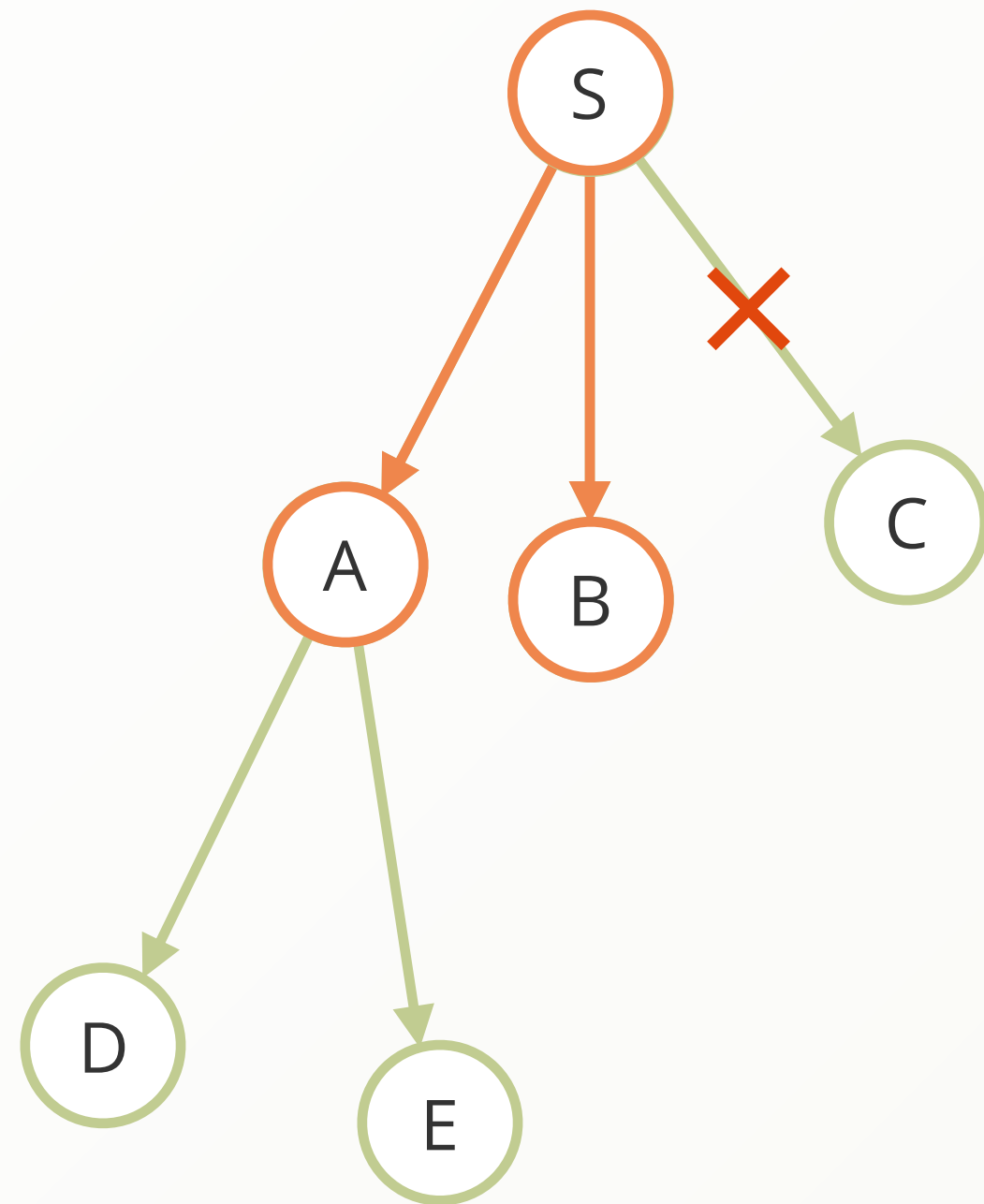


- ▶ We first pick a start vertex (S)
- ▶ We collect all edges on S
- ▶ We apply filters on edges



# Traversal - Iterate down two edges with some filters

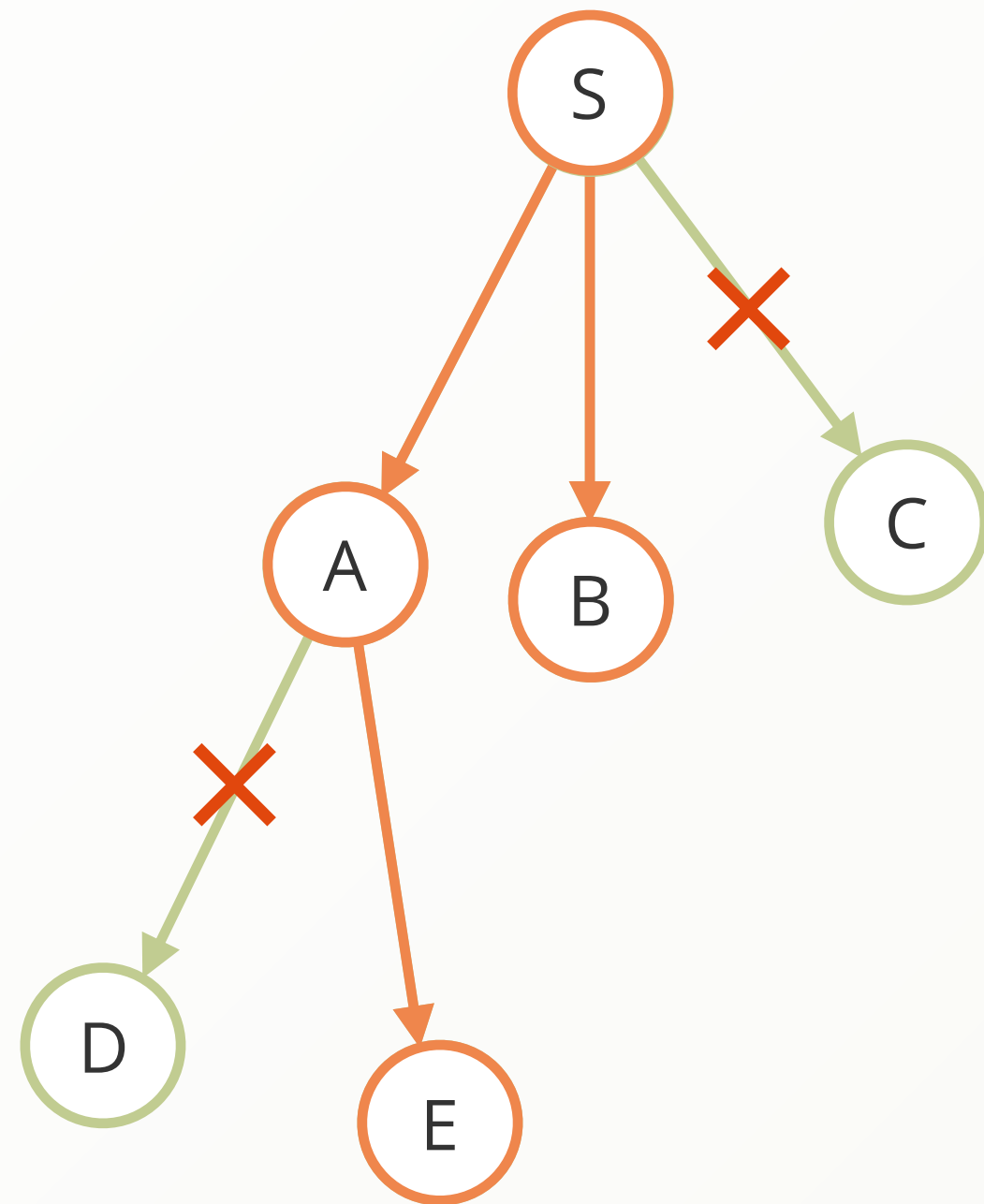
---



- ▶ 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

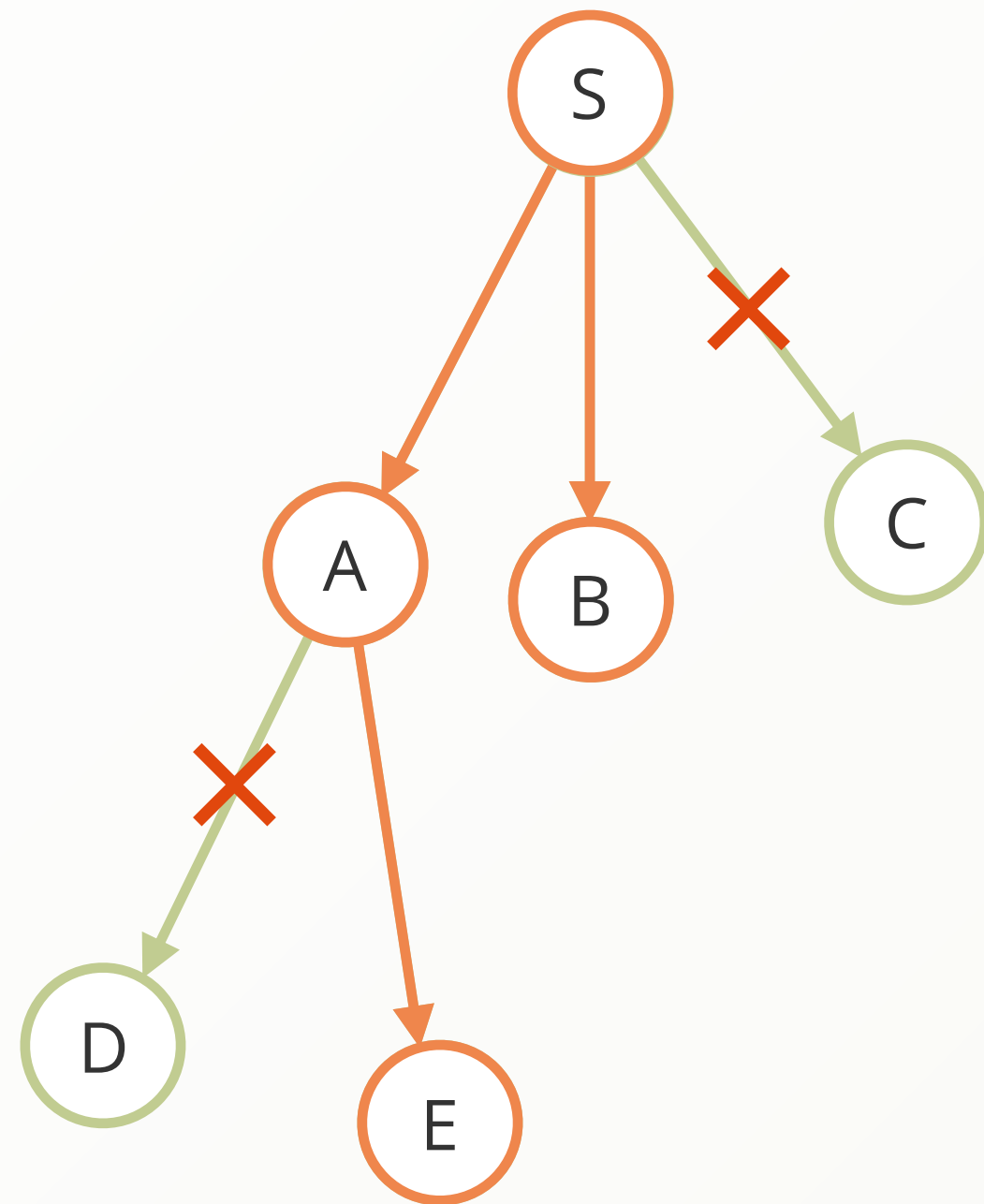
---



- ▶ 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

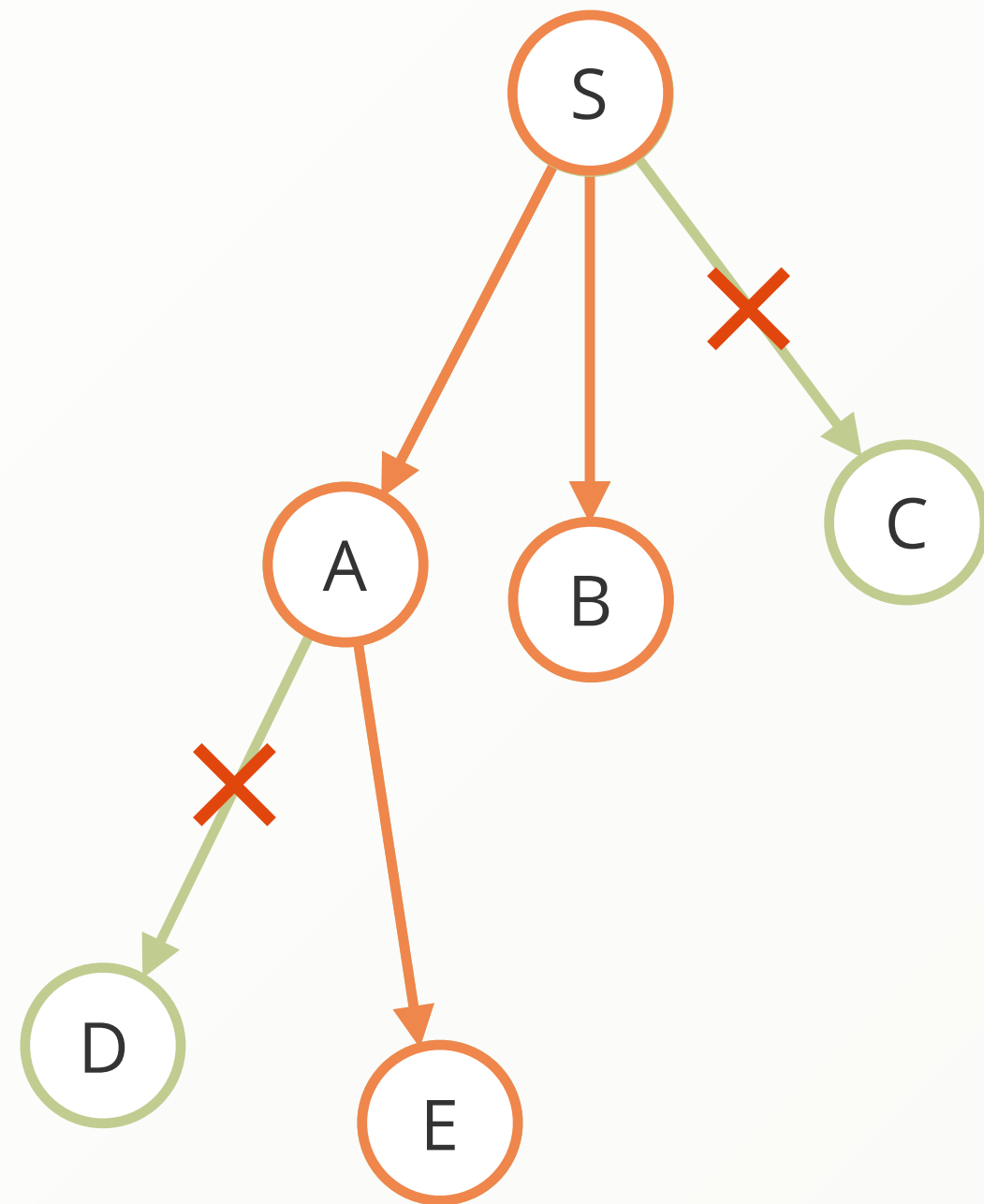
---



- ▶ 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

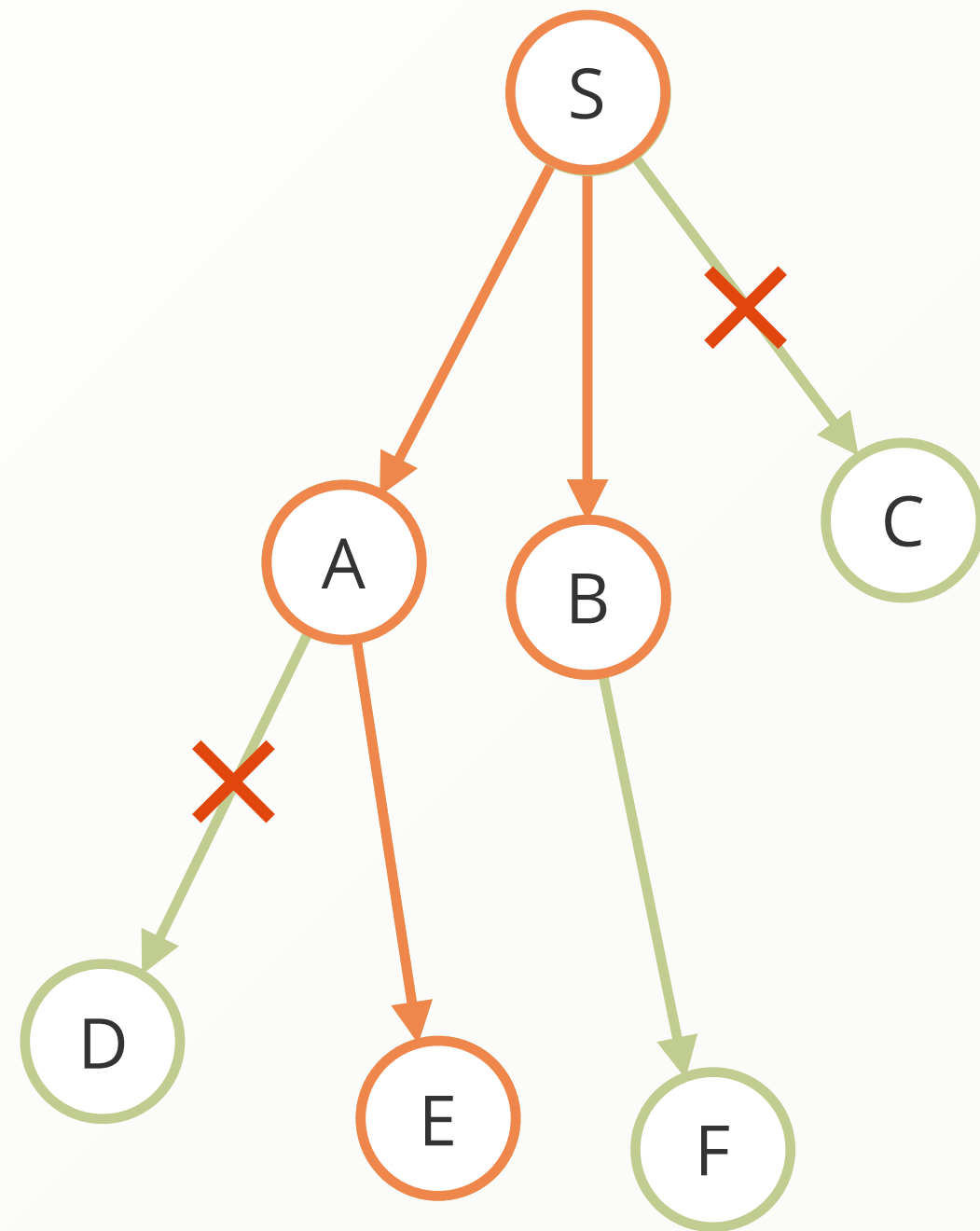
---



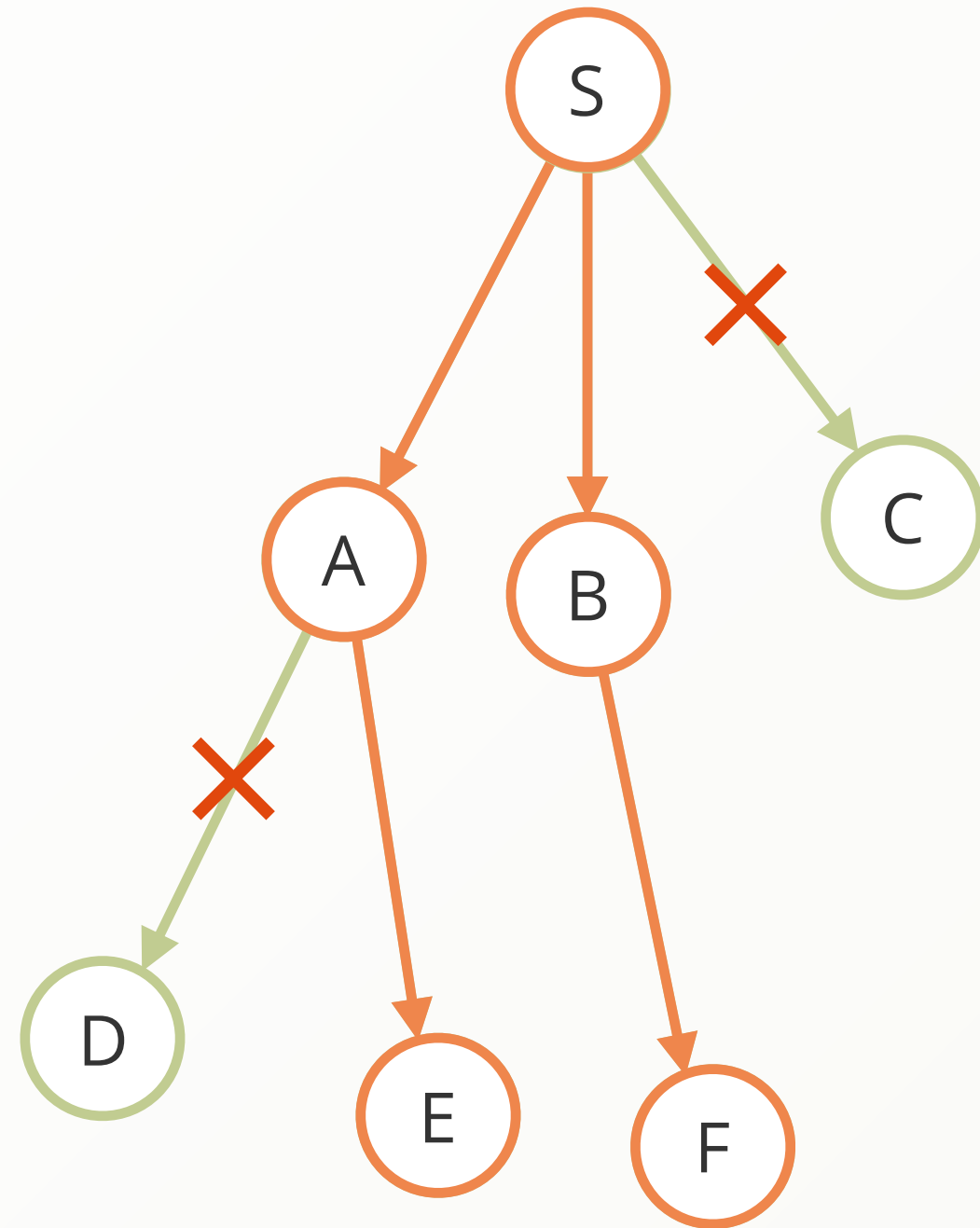
- ▶ We first pick a start vertex (S)
- ▶ We collect all edges on S
- ▶ We apply filters on edges
- ▶ We iterate down one of the new vertices (A)
- ▶ We apply filters on edges
- ▶ The next vertex (E) is in desired depth.  
Return the path  $S \rightarrow A \rightarrow E$
- ▶ Go back to the next unfinished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

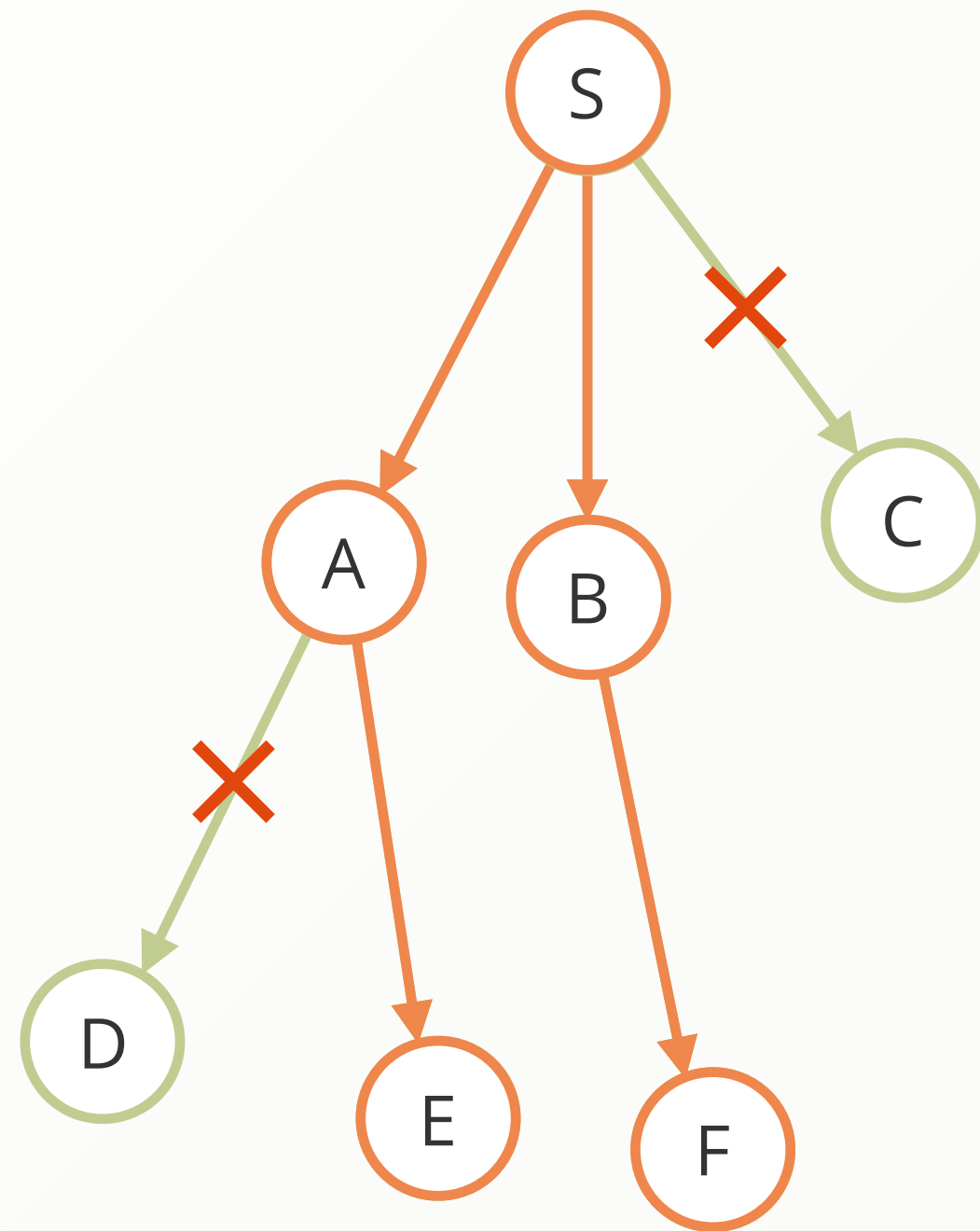


- ▶ We first pick a start vertex (S)
- ▶ We collect all edges on S
- ▶ We apply filters on edges
- ▶ We iterate down one of the new vertices (A)
- ▶ We apply filters on edges
- ▶ The next vertex (E) is in desired depth.  
Return the path  $S \rightarrow A \rightarrow E$
- ▶ Go back to the next unfinished 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:	$3n$

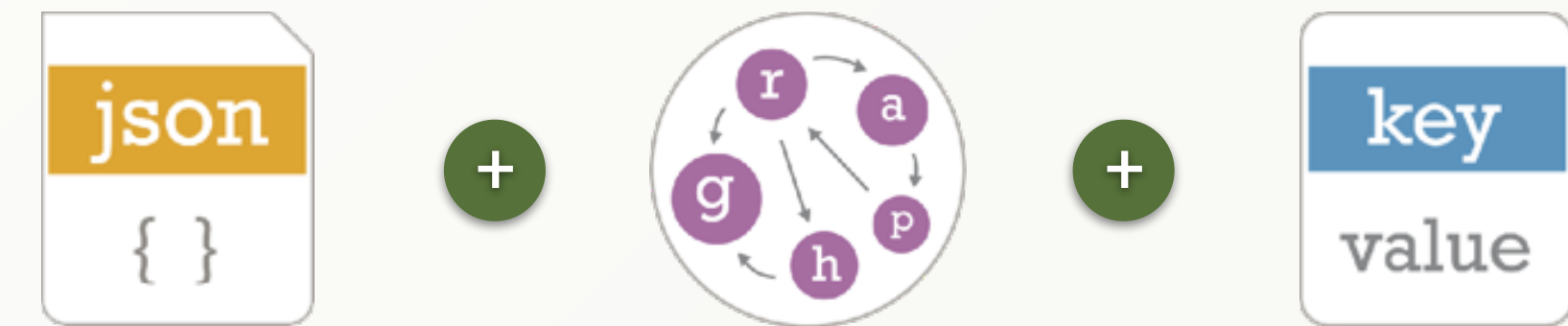


# Traversal - Complexity

---

- ▶ 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:  $O(3^n)$
- ▶ "7 degrees of separation":  $3^n < E < 3^{n+1}$

- ▶ MULTI-MODEL database
  - ▶ Stores Key Value, Documents, and Graphs
  - ▶ All in one core
- ▶ 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
```

Alice

# AQL

---

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



Alice

# AQL

---

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



# AQL

---

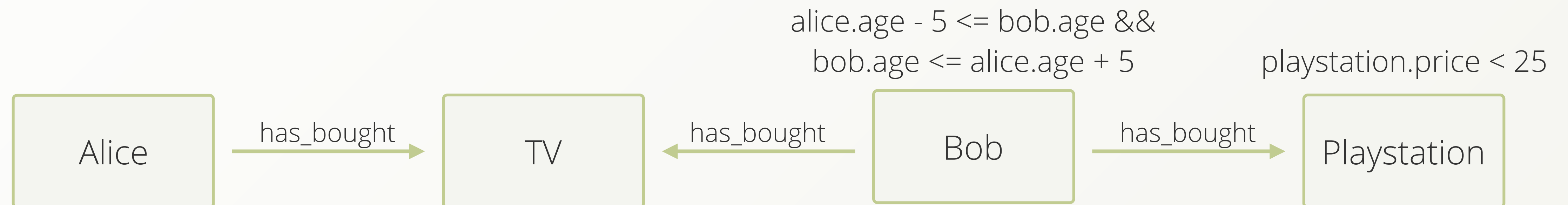
```
FOR user IN users
  FILTER user.name == "alice"
  FOR recommendation, action, path IN 3 ANY user has_bought
    FILTER path.vertices[2].age <= user.age + 5
    AND path.vertices[2].age >= user.age - 5
    FILTER recommendation.price < 25
  LIMIT 10
RETURN recommendation
```



# 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
```

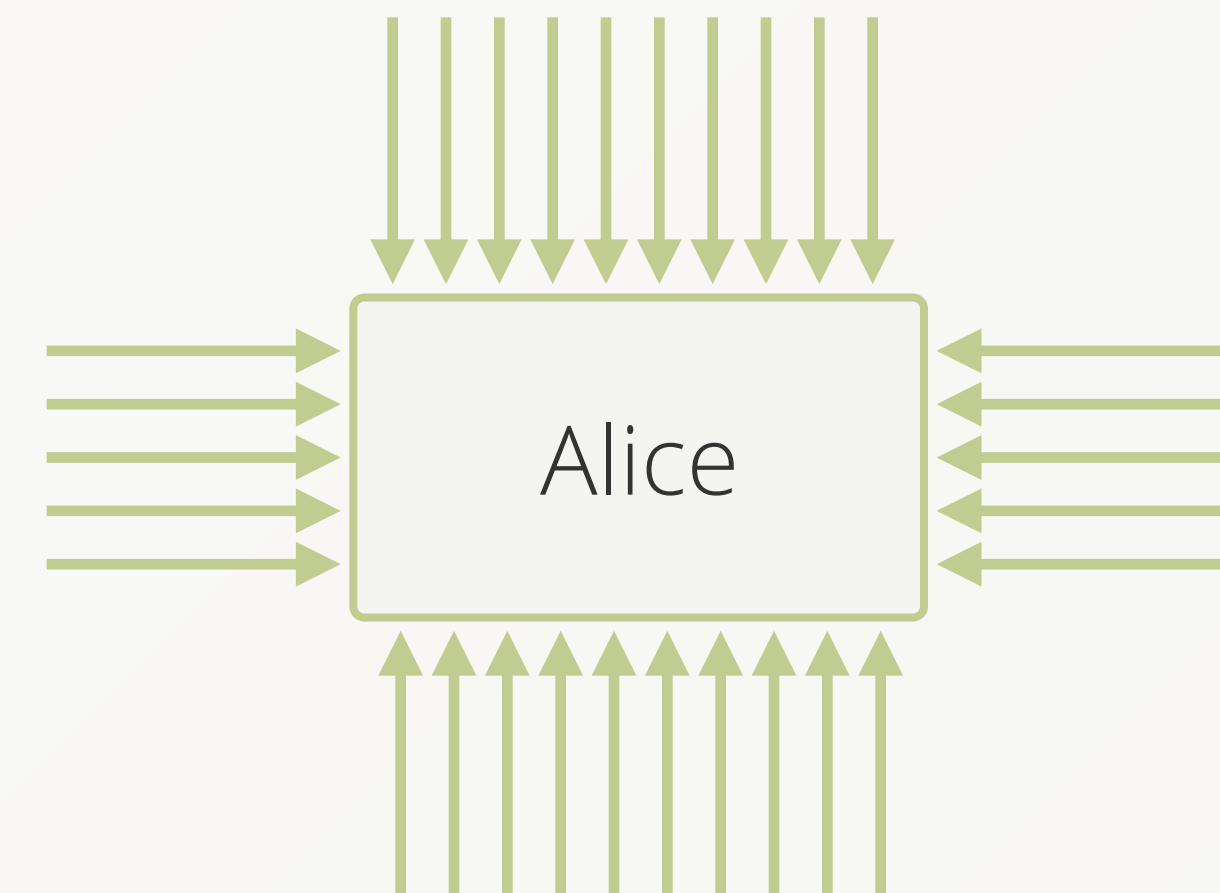
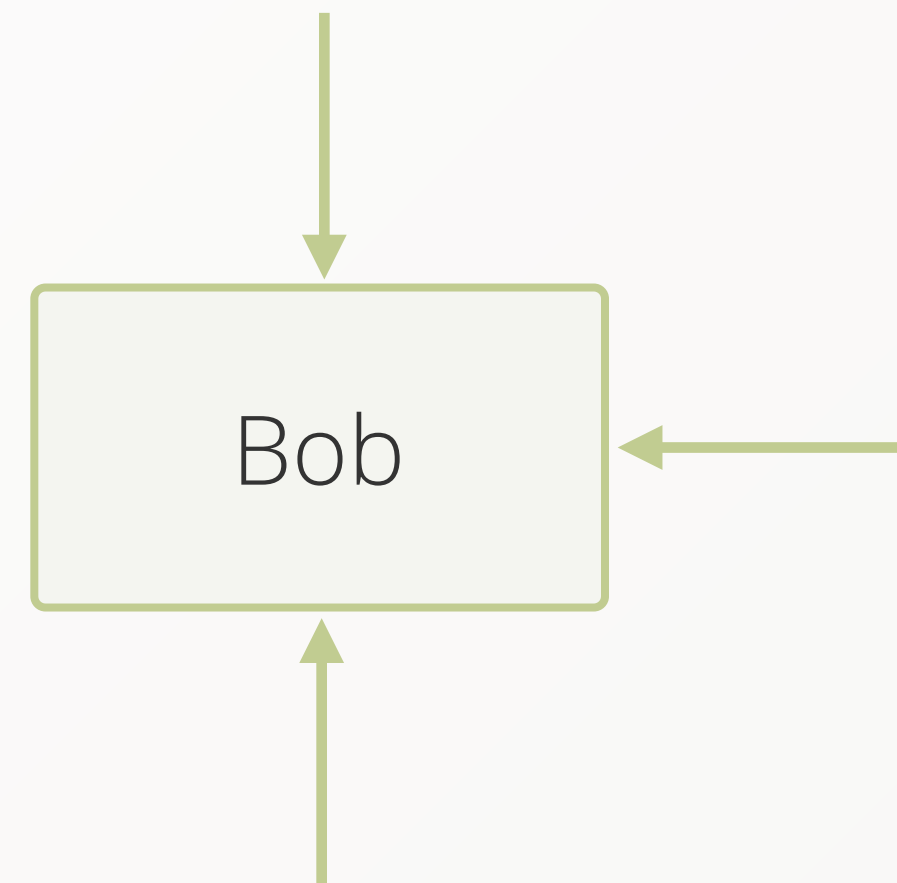




# Challenge 1: Supernodes

---

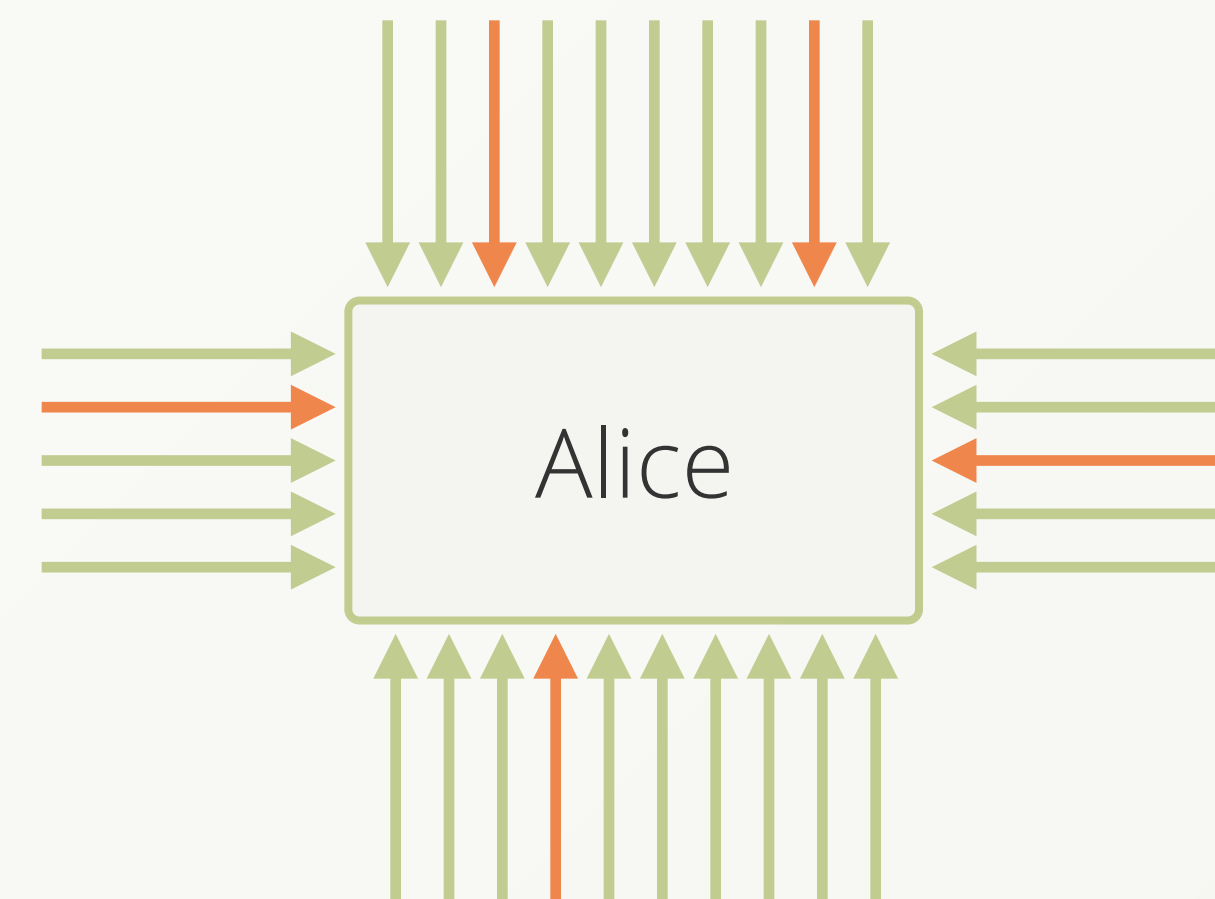
- ▶ Many graphs have "celebrities"
  - ▶ Vertices with many inbound and/or outbound edges
- ▶ Traversing over them is expensive (linear in number of Edges)
- ▶ Often you only need a subset of edges



# 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  $n$  significantly



# Challenge 2: Big Data

---

- ▶ We have the rise of big data
  - ▶ Store everything you can
- ▶ Dataset easily grows beyond one machine
- ▶ This includes graph data!

# Scaling

---

- ▶ 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?
- ▶ In a sharded environment network most of the time is the bottleneck
  - ▶ Reduce network hops
- ▶ Vertex-Centric Indexes again help with super-nodes
  - ▶ But: Only on a local machine

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

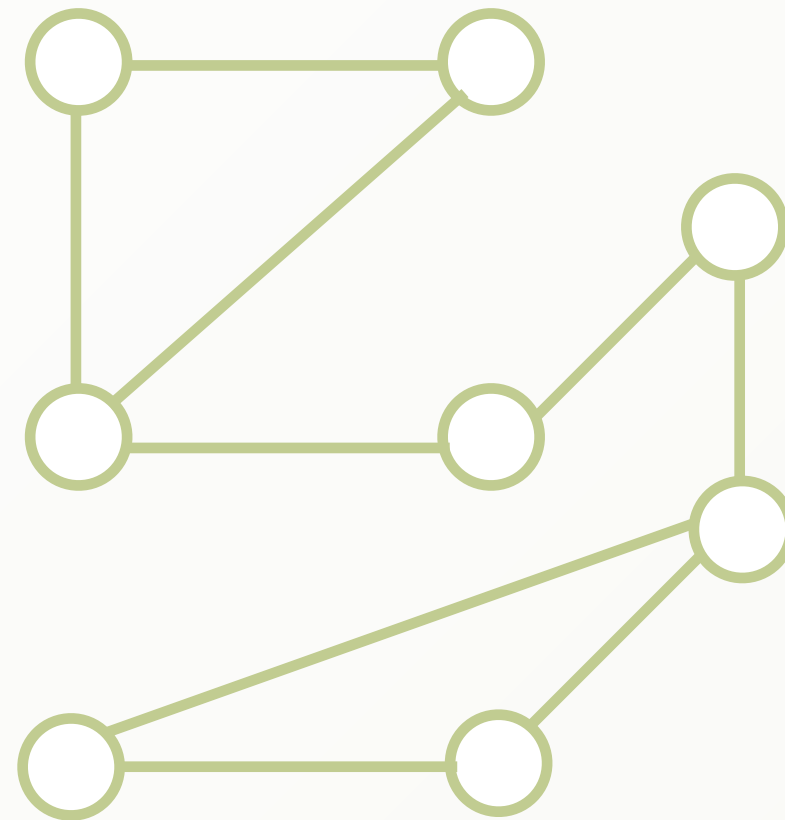
---

- ▶ 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

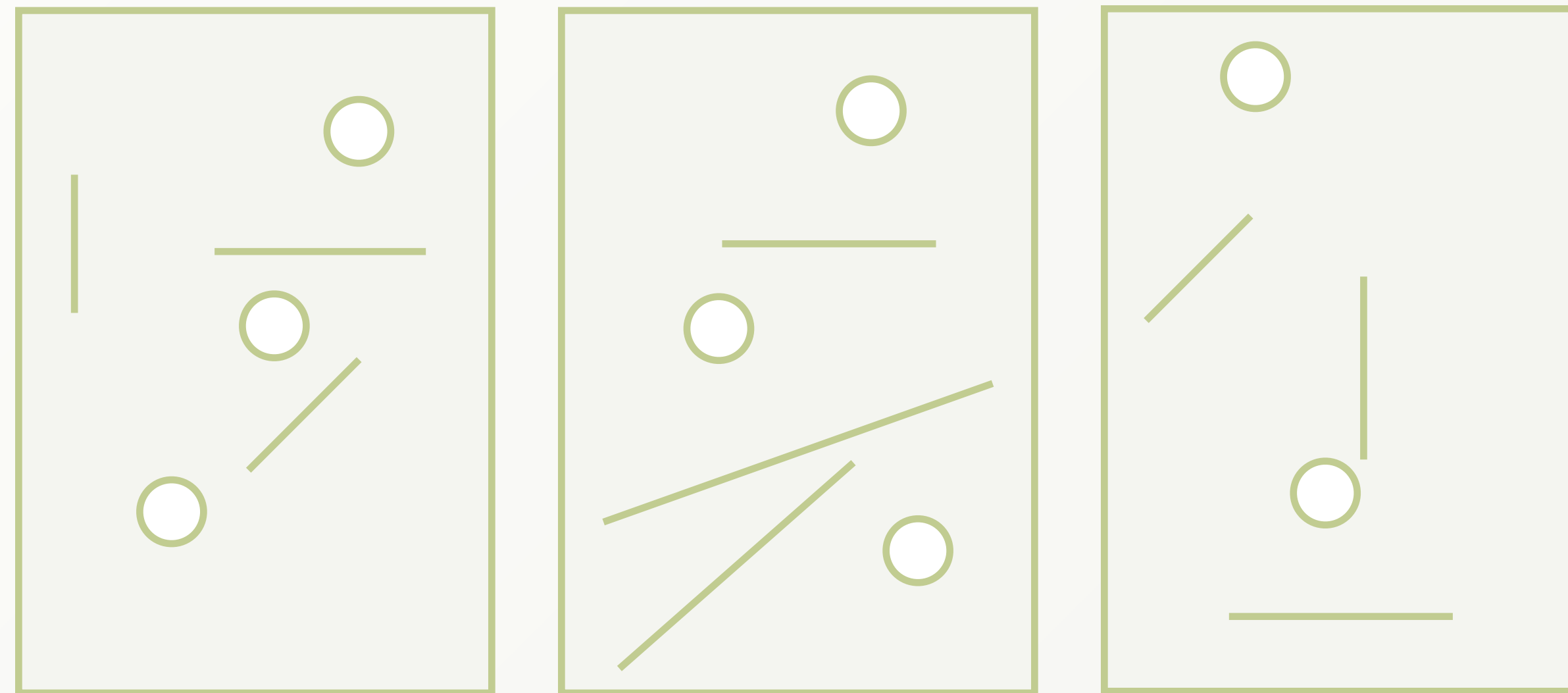
---

- ▶ 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

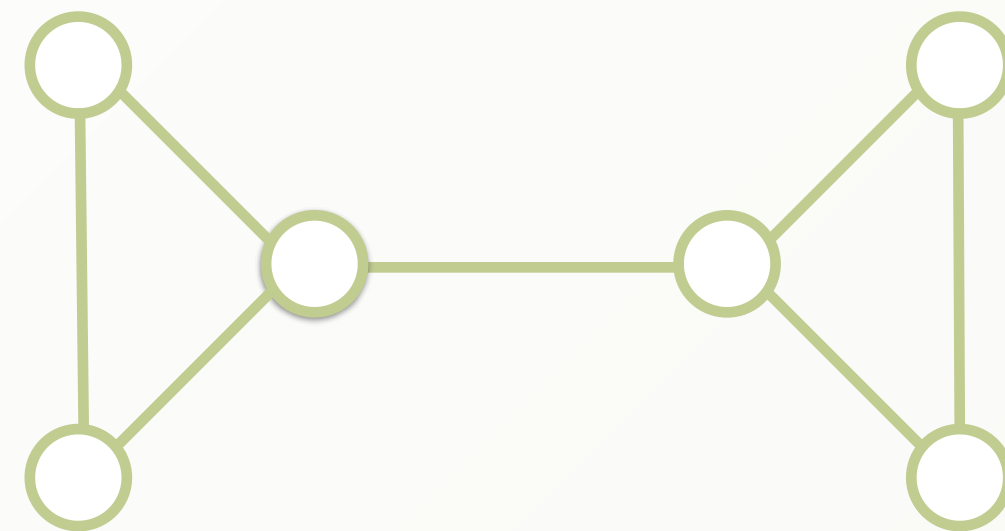




# Index-Free Adjacency

---

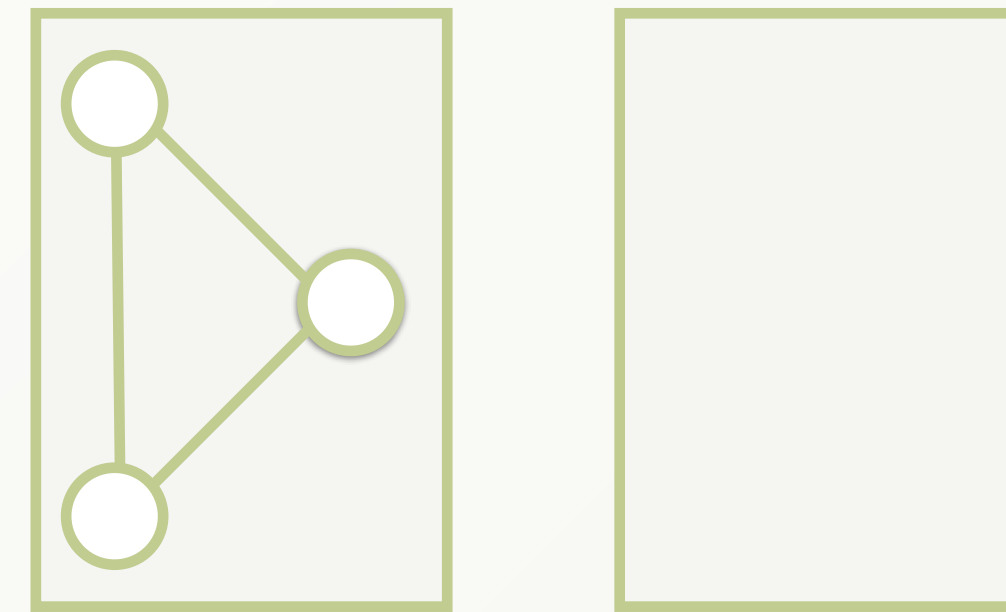
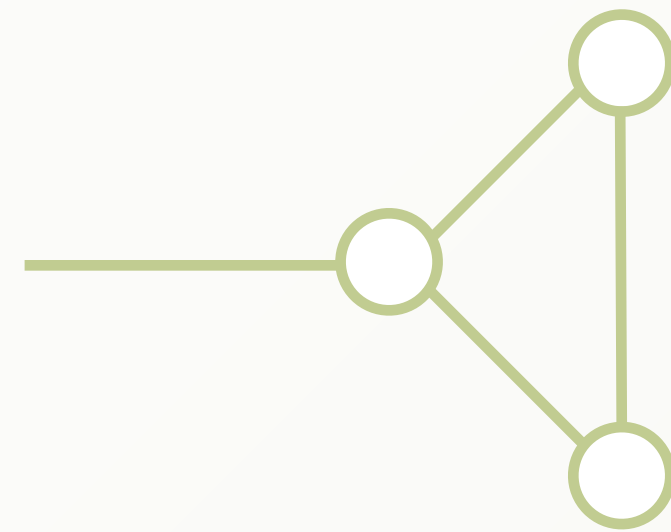
- ▶ Used by most other graph databases
- ▶ Every vertex maintains two lists of its edges (IN and OUT)
  - ▶ Do not use an index to find edges
  - ▶ How to shard this?



# Index-Free Adjacency

---

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

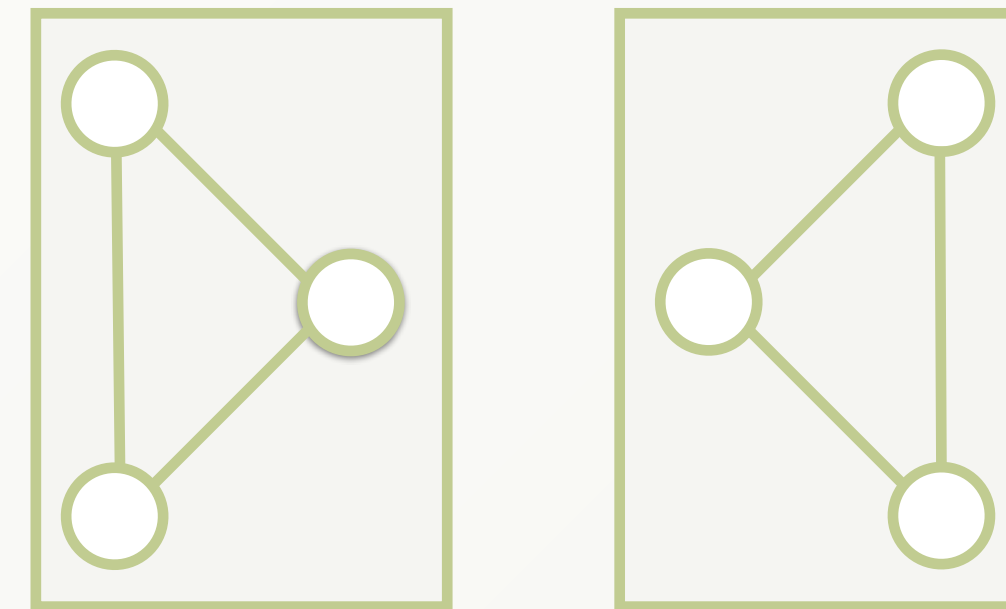


# Index-Free Adjacency

---

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

—

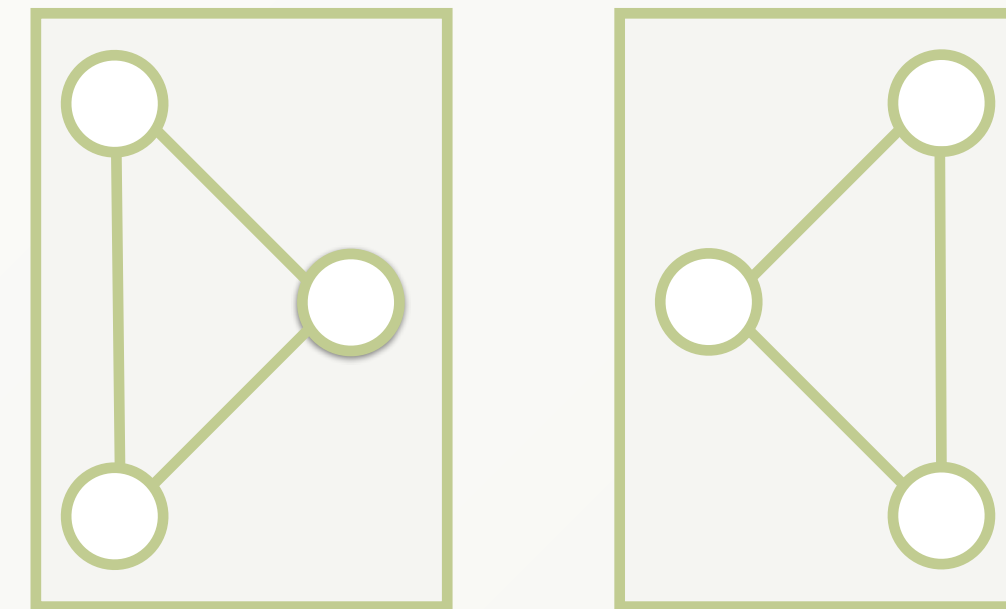


# Index-Free Adjacency

---

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

????

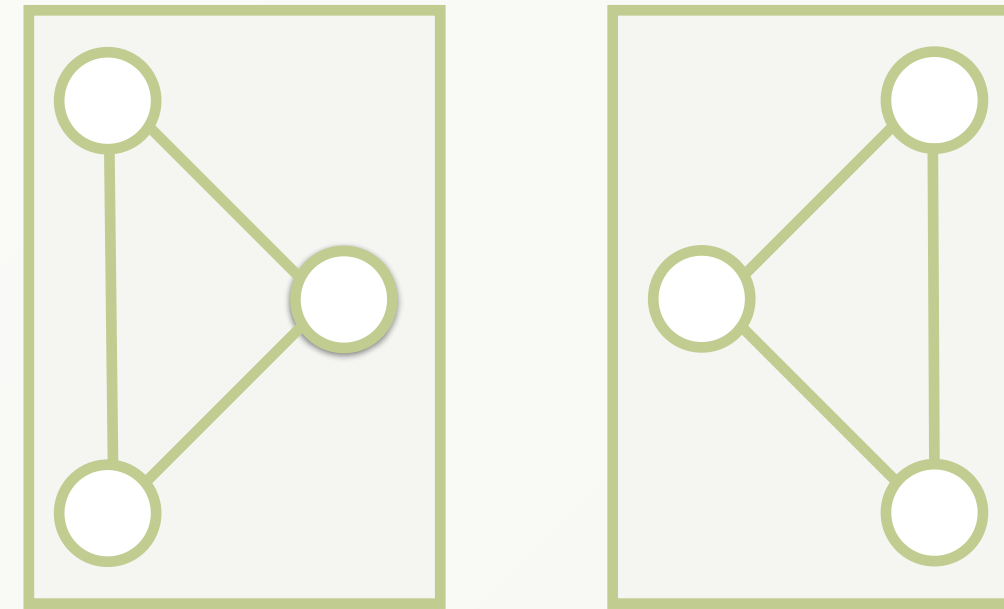


# Index-Free Adjacency

---

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

????

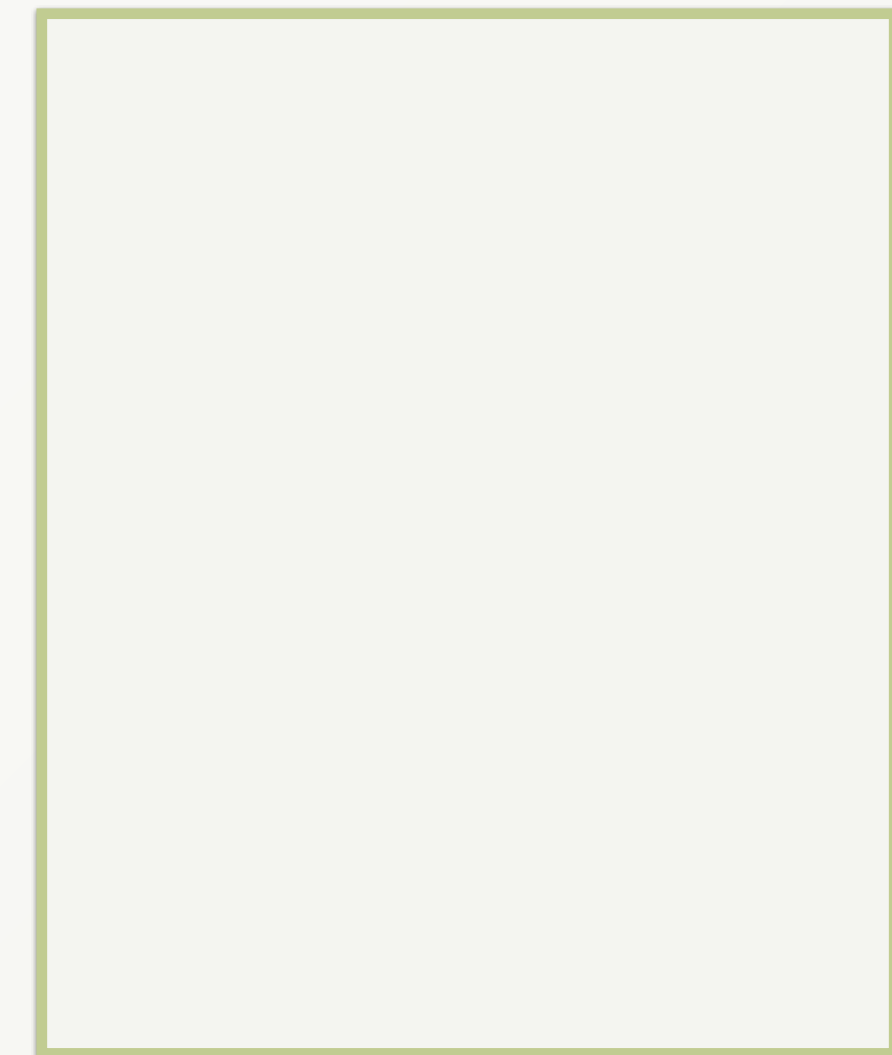
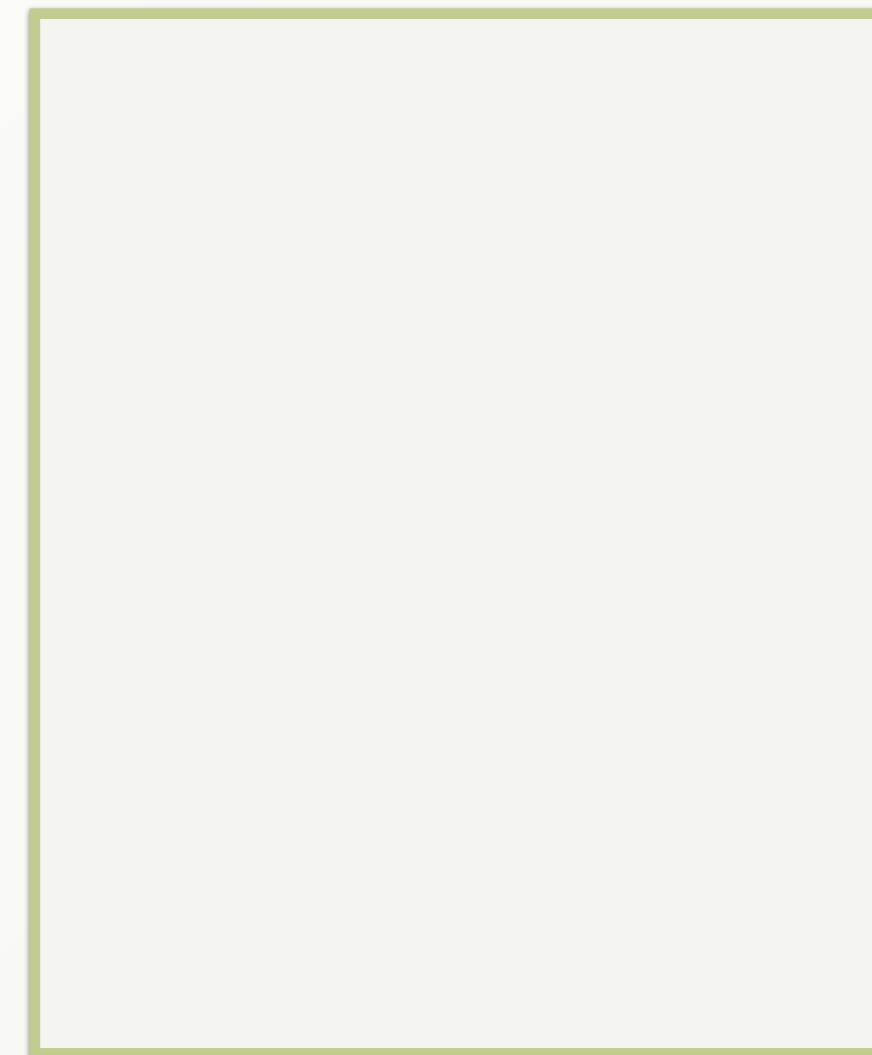
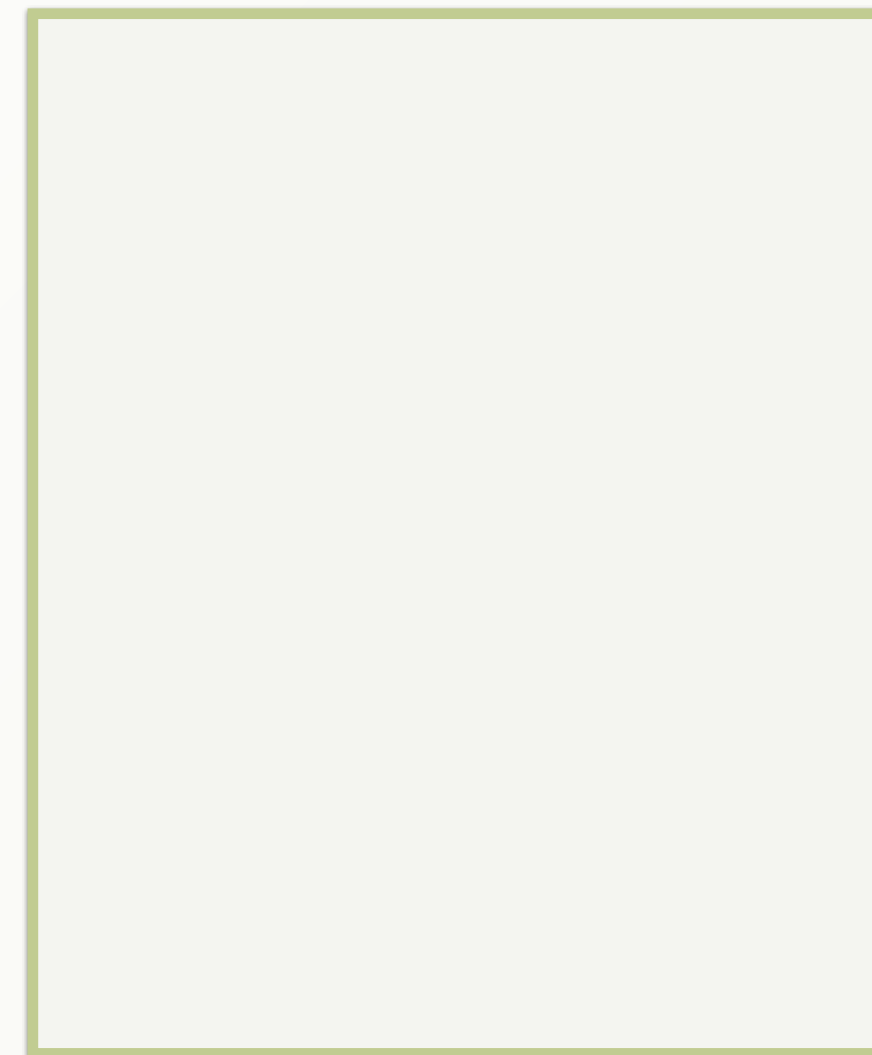


- ▶ ArangoDB uses a hash-based EdgeIndex ( $O(1)$  - lookup)
  - ▶ The vertex is independent of its edges
  - ▶ It can be stored on a different machine

# 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

---

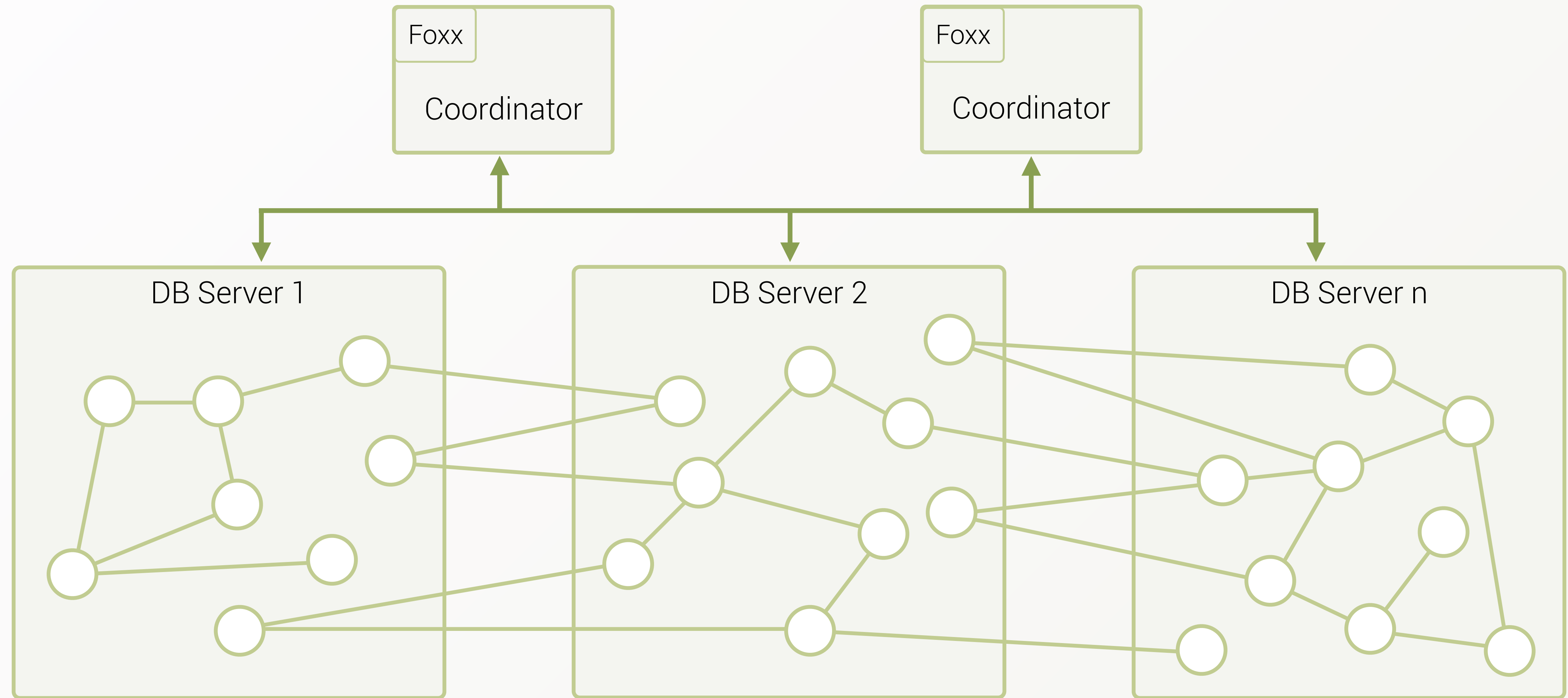
- ▶ Many Graphs have a natural distribution
  - ▶ By country/region for People
  - ▶ By tags for Blogs
  - ▶ By category for Products
- ▶ Most edges in same group
- ▶ Rare edges between groups

ArangoDB Enterprise Edition  
uses Domain Knowledge  
for short-cuts



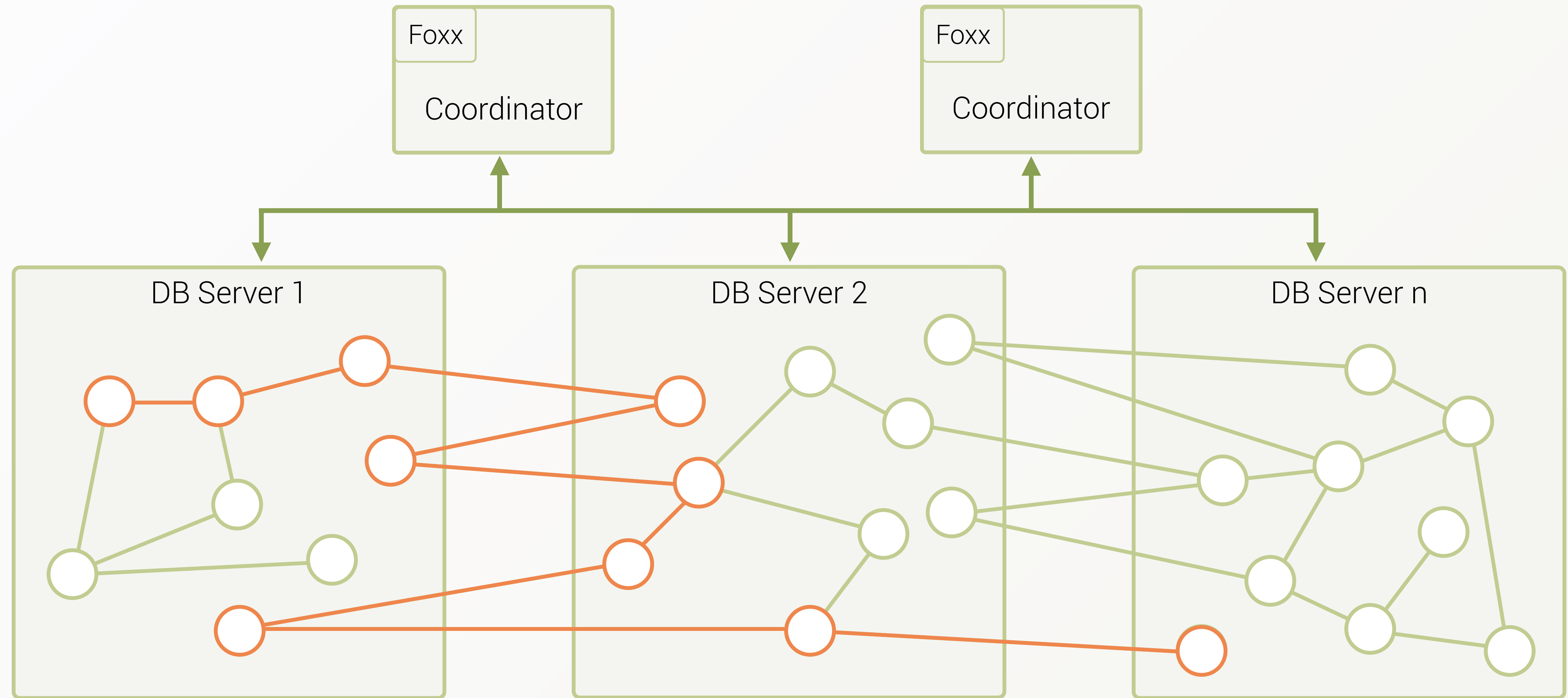
# SmartGraphs - How it works

---



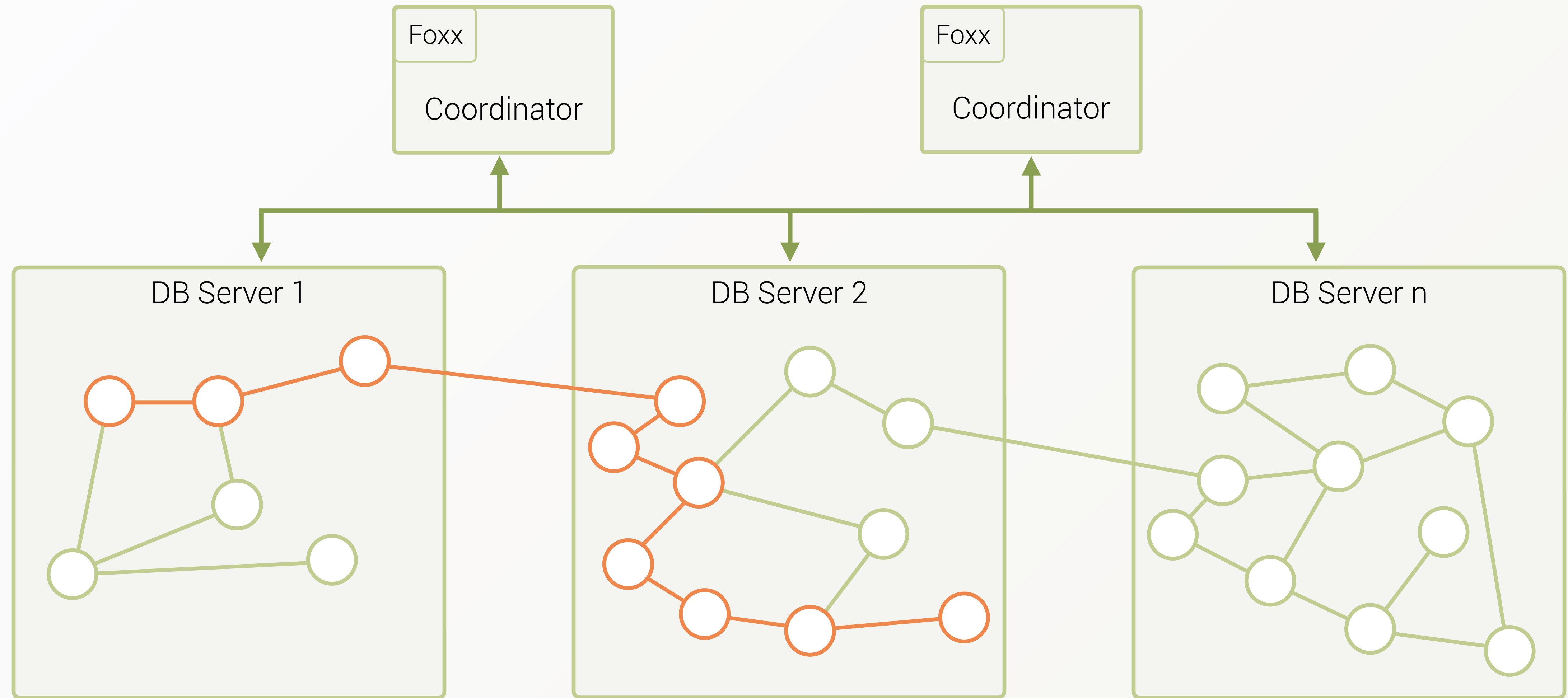
# SmartGraphs - How it works

---



# SmartGraphs - How it works

---



# Thank You

- ▶ Further questions?
  - ▶ Follow us on twitter: @arangodb
  - ▶ Join our slack: [slack.arangodb.com](https://slack.arangodb.com)
- ▶ Follow me on twitter/github: @mchacki