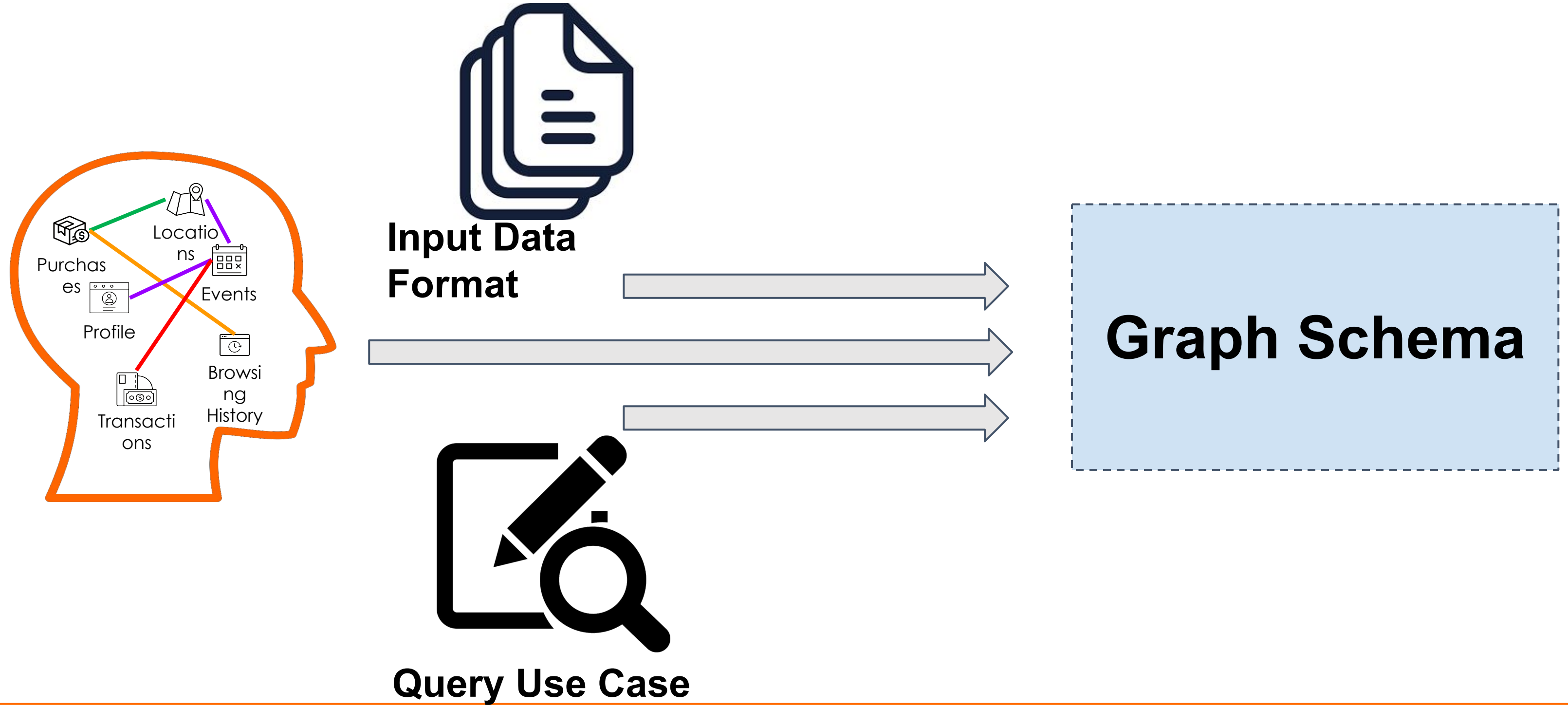
Best Practices for Modeling Your Data With Graph

Schema Design Best Practices

A good graph schema design represents important relationships in a natural way, while also minimizing system resource consumption and enabling the best query performance. A **schema** should consider **input data format** and **query use cases**.



Schema Design Topics

1. Rule of thumb: What can be vertices? What can be edges?
2. Choosing an edge type: Undirected? Directed? Reversed?
3. Granularity of edge type
4. Attribute or vertex or User-Defined Index?
5. ID As Attribute?
6. How to model time in a graph
7. Multiple events/transactions between two entities
8. Store query result and support Data Science Library
9. Design schema-based on use case

TigerGraph

Rule of Thumb: What can be Vertices? What can be Edges?

Entities or abstract concepts can be defined as **vertices**

Example: person, user, company, account, product, address, phone number, device, type, status, role etc.

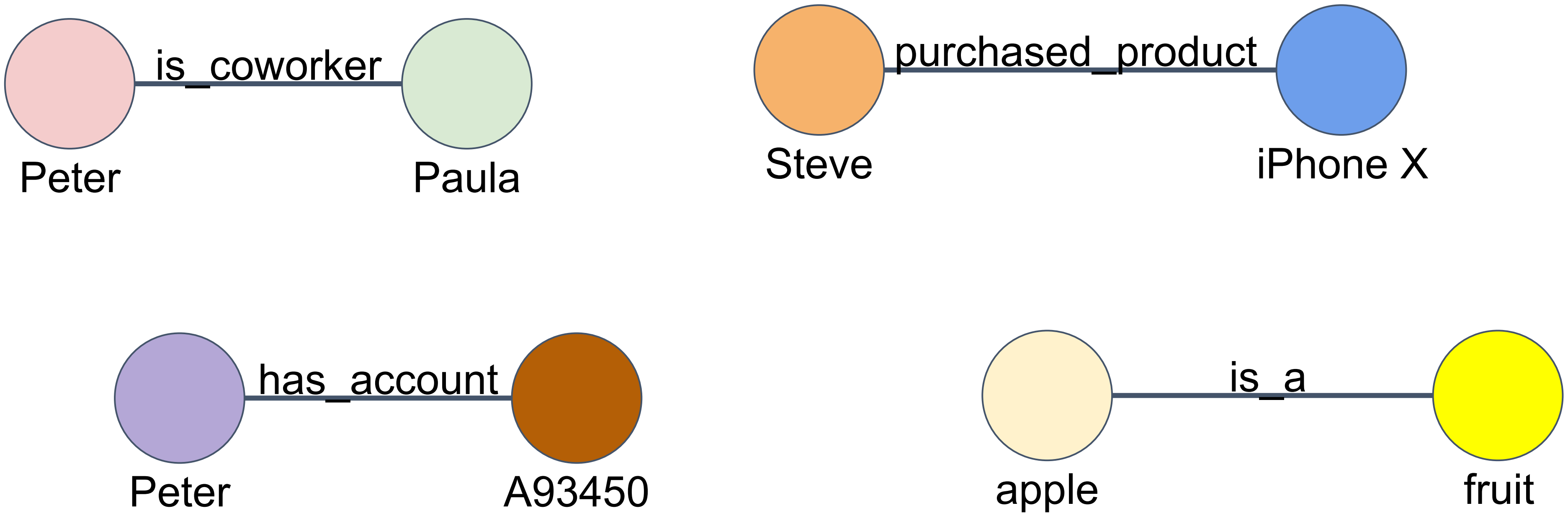
Relationships can be defined as **edges**

Such as: is_coworker, works_for, is_controlled_by, has_account, purchased_product, has_home_address, belongs_to_type, etc.

TigerGraph

Rule of Thumb: What can be Vertices? What can be Edges?

Vertices are shown as circles and edges are shown as lines.

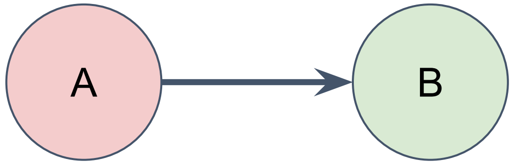


Choosing an Edge Type: Undirected? Directed? Reversed?

Undirected Edge

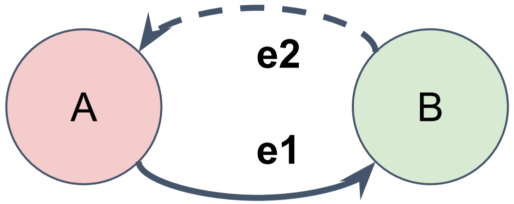
A can traverse to B, and B can traverse to A

Directed Edge



A can traverse to B, but B **cannot** traverse to A

Directed Edge + Reverse Edge



A can traverse to B via e1, B can traverse to A via e2
(e2 is automatically created upon creation of e1.
e2's attributes have the same values as e1's)

- What is the difference between undirected edge and directed edge + reverse edge?
- What to know when making edge type choices?

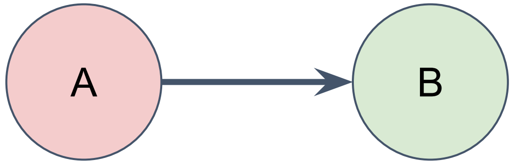
TigerGraph

Choosing an Edge Type: Undirected? Directed? Reversed?

Undirected Edge

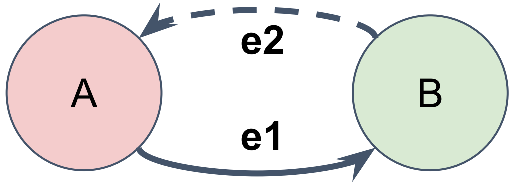
A can traverse to B, and B can traverse to A
Pros: Simple when working with undirected (symmetric) or bidirectional relationships.
  Example: "A friend_of B" $\Leftrightarrow$ "B friend_of A"
Cons: Does not carry directional info.

Directed Edge



A can traverse to B, but B **cannot** traverse to A
Pros: Saves memory and correctly describes a direction-restricted relationship
  Example: "A parent_of B" $\nRightarrow$ "B parent_of A"
Cons: Can not traverse back from target to source.

Directed Edge + Reverse Edge

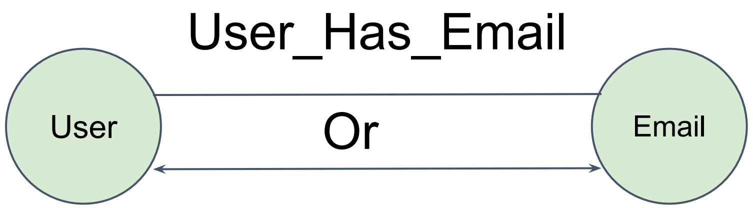


A can traverse to B via e1, B can traverse to A via e2
Pros: Flexibility to traverse in either designated direction.
  Example: e1 type is "parent_of" and e2 type is "child_of"
Cons: Need to remember two edge types.

TigerGraph

Choosing an Edge Type: Undirected? Directed? Reversed?

Given Schema:



Find users share the same email

with undirected edge:

user_share_email = SELECT t FROM start-(User_Has_Email*2)-:t;

with directed edge + reverse edge

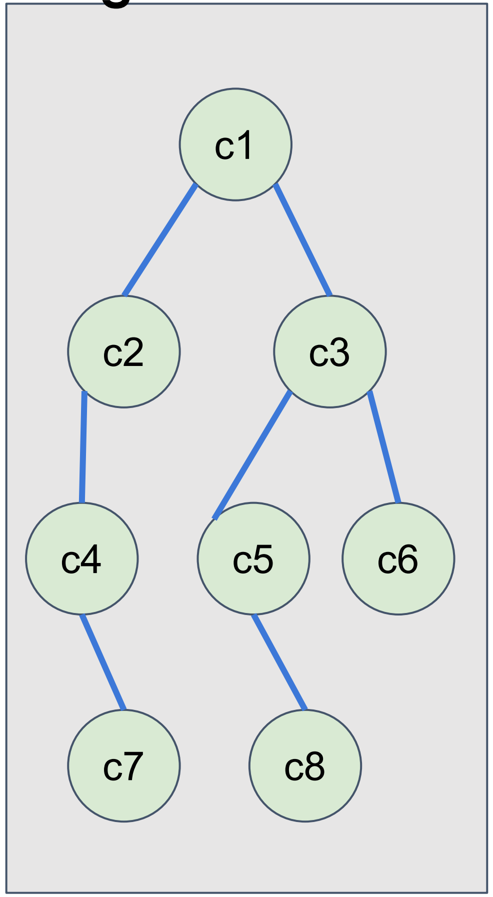
user_share_email = SELECT t FROM start-(User_Has_Email>)-email-(reverse_User_Has_Email>)-:t;

In this case, it is more concise to use undirected edge

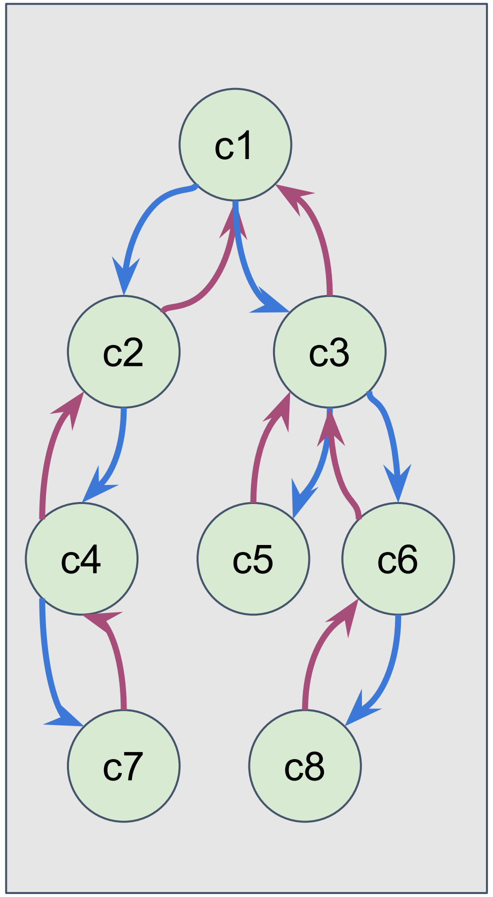
Choosing an Edge Type: Undirected? Directed? Reversed?

Use Case:
Given an enterprise graph and an input company, find its ultimate parent company and the ultimate child branches

Undirected Edge



Directed Edge + Reverse Edge



In this use case, the question is hard to answer by using undirected edges, since they do not provide any directional info (parent vs. child)

However it can be easily solved by using directed edge + reversed edge.

When querying for parent companies it can use the red edge, and when querying for child companies it can use the reverse edge (blue).

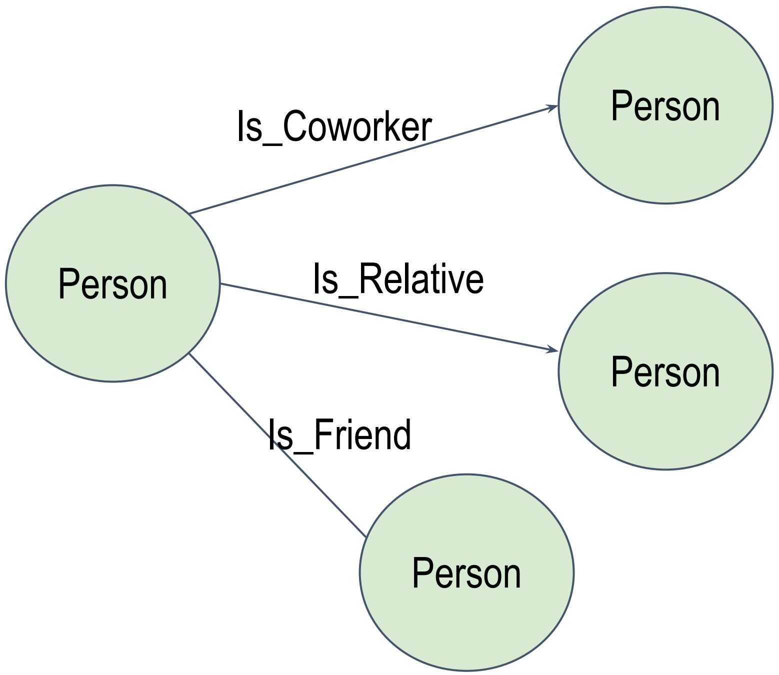
TigerGraph

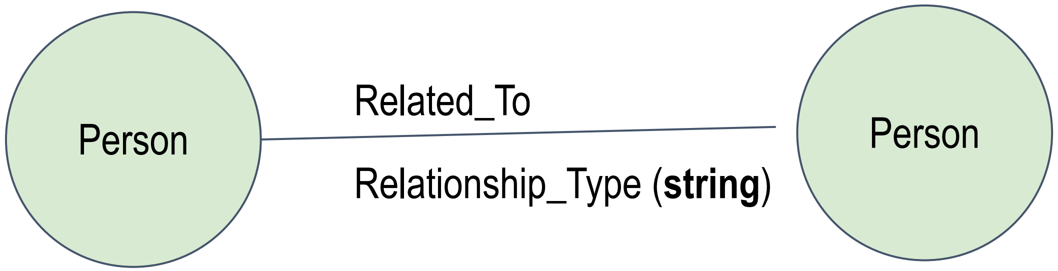
Choosing an Edge Type: Undirected? Directed? Reversed?

Quiz: What type of edge(s) would you pick?

1. A Product **is part of** a PurchaseOrder.
2. An Account **is owned by** one or more Persons.
3. Donors **donate anonymously** to a Charity.
4. A Product **is compatible with** another Product.

Granularity of Edge Type





Option 1: Person vertexes are interconnected via different edge types
Pros: Efficient when traversing specific edge types, uses less memory
Cons: Less concise when traversing all types, makes schema very large when having many relationship types.

Option 2: Person vertexes are interconnected via one edge type
Pros: Easy to traverse all relationships
Cons: Attribute for relationship type consumes extra memory, relationship type checking is slower

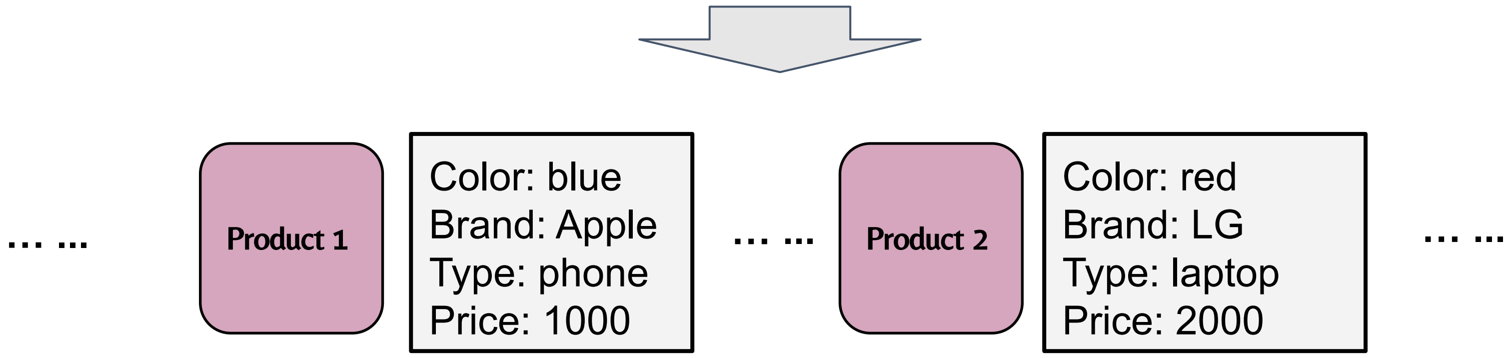
TigerGraph

Attribute or Vertex or User-Defined Index?

Given a column, should it be defined as an attribute or a vertex?

Product Table

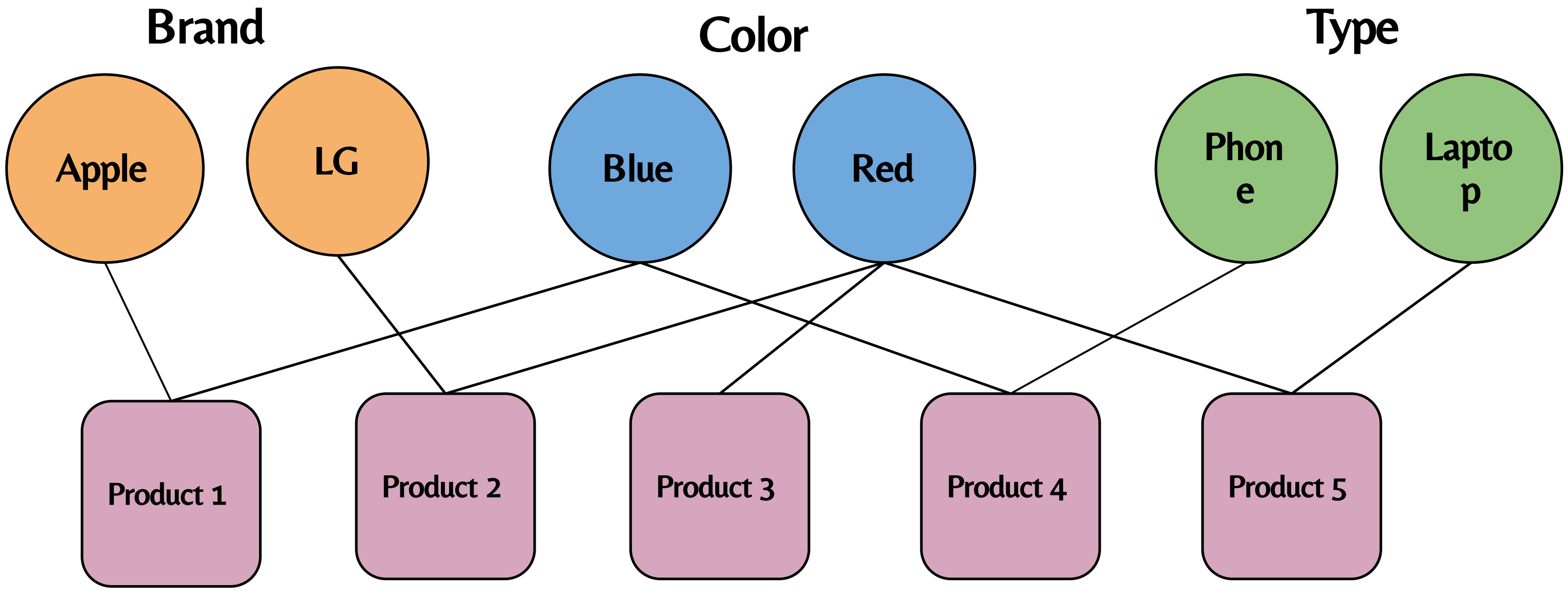
Product Name	Color	Brand	Type	Price
product 1	blue	Apple	phone	1000
product 2	red	LG	laptop	2000
...	...	...	...	...



Searching for a *red* product scans over **ALL** products

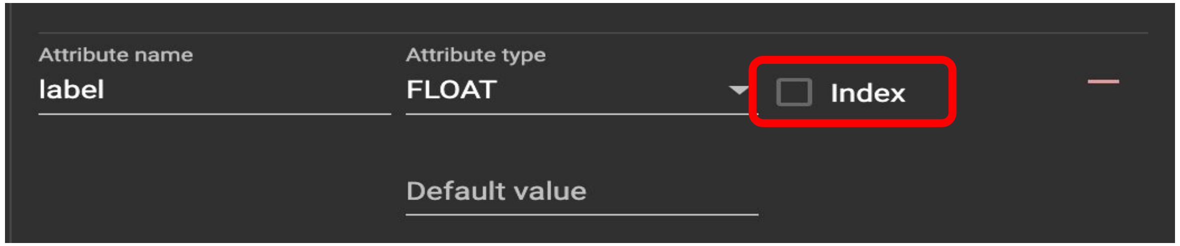
Attribute or Vertex or User-Defined Index?

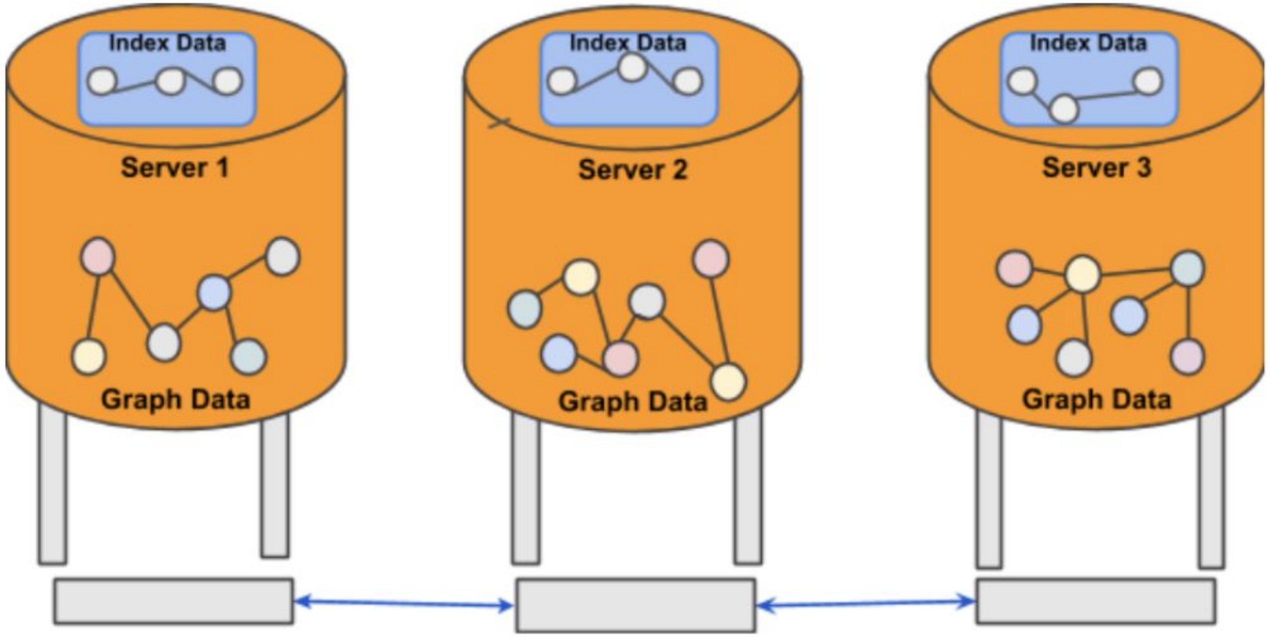
It can be beneficial to represent a column (attribute) as a vertex type if you will frequently need to query for particular values of the property. This way, the vertices act like an search index. E.g., all the red products are connected to the **Red** vertex under **Color** type.



Attribute or Vertex or User-Defined Index?

User-defined Indexes or Secondary Indexes (as they are commonly called in Database Industry) are a critical feature that enhance the performance of a database system. Indexes allow users to perform fast lookups on non-key columns or attributes without a full-fledged scan.





Attribute or Vertex or User-Defined Index?

In conclusion, the best fit for the User-define Index are queries with the following characteristics:

- Low distinct value of the attribute
- Low selectivity query on the attribute

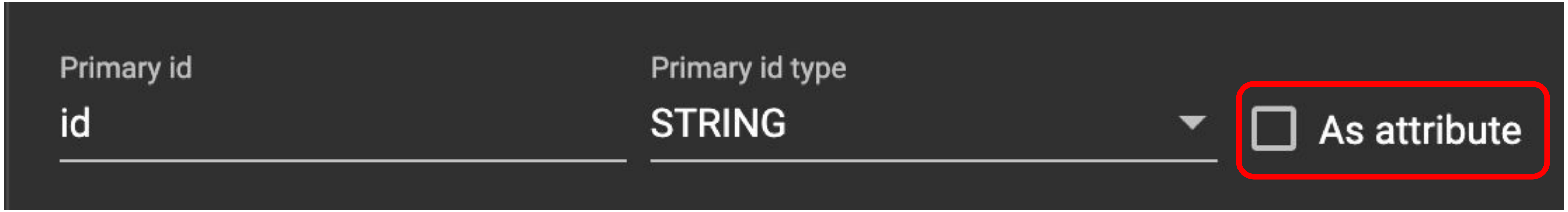
Vertex or Index?

- Secondary index supports range predicates, while index vertex does not
- Vertex index could easily create hub node, while secondary index is more scalable
- More operational overhead for user to create index vertex, and maintain them
- Index vertex would have great performance over non continuous values

As Attribute?

By default, `**As attribute**` is disabled, the ID value is not accessible from the query. Therefore `**As attribute**` needs to be checked when vertex ID values are used in the query logic for non-printing purposes such as:

1. String concatenation
2. Value comparison
3. …



As Attribute?

Why disable as default?

“USER123” <---> 1234321

IDS: Bidirectional external ID to Internal ID mapping

1234321, John, 33, john@abc.com
1234322, Tom, 27, tom@abc.com
…

Vertex Partitions: Vertex internal ID and attributes

1234321, 1234322, 2020-04-23, 3.3
1234321, 1234324, 2020-02-13, 2.3
…

Edge Partitions: Source vertex internal ID target vertex internal ID, edge attributes

As Attribute?

As a conclusion:

1. IDS can handle the requirements of using vertex ID as query parameters and printing vertex IDs in the query result

2. As a unique identifier, we can use internal ID from the query

3. '**As attribute**' is needed only when ID value is needed for purpose other than above

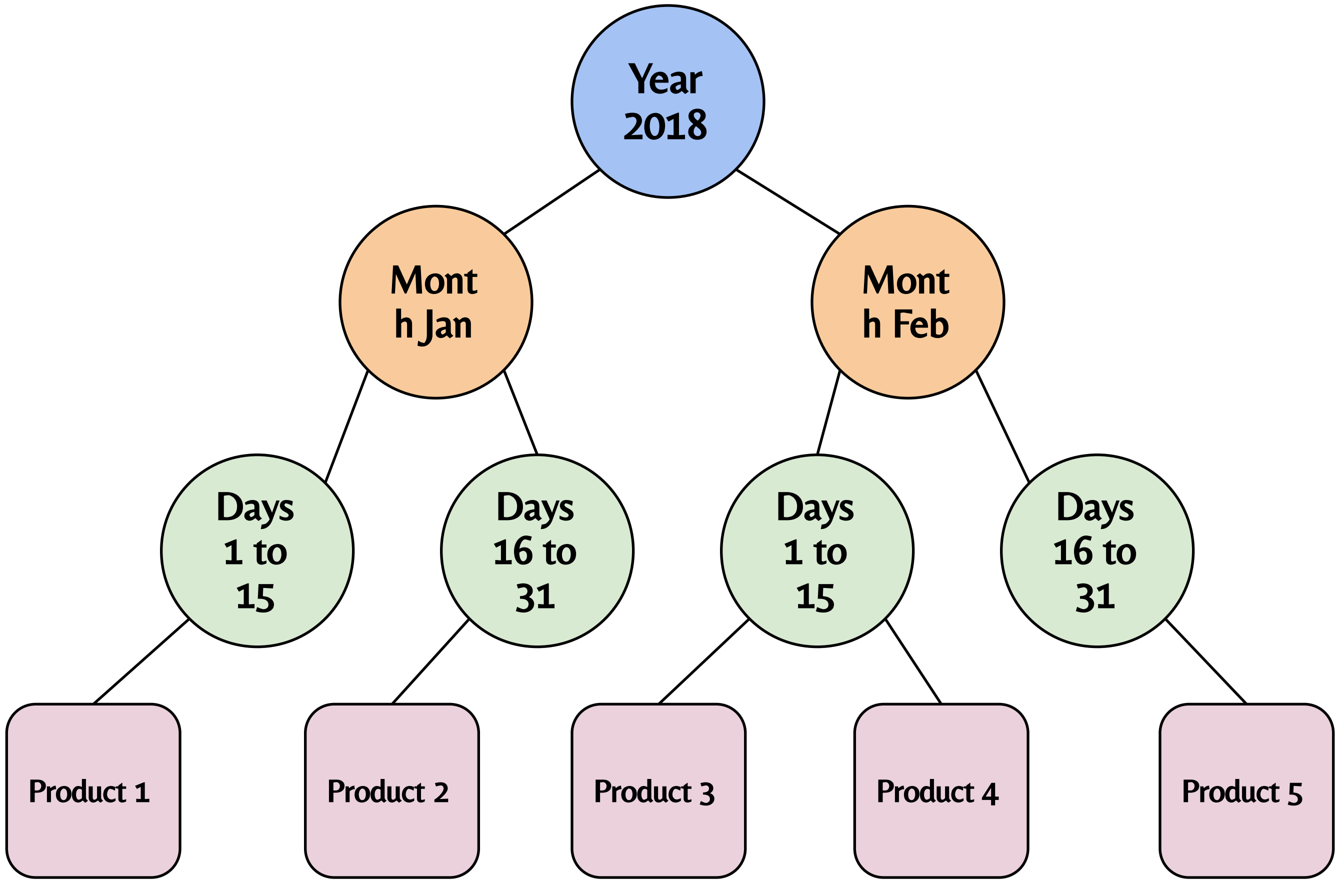
4. Disabling '**As attribute**' is more memory saving

How to Model Time in a Graph

Similar to creating vertices for attributes.

Hierarchical datetime structures can be created for faster time series querying speed.

The levels and partitioning of each level can be customized to best suit your use case.



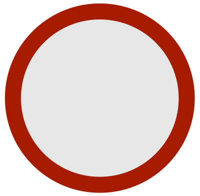
TigerGraph

How to Model Time in a Graph

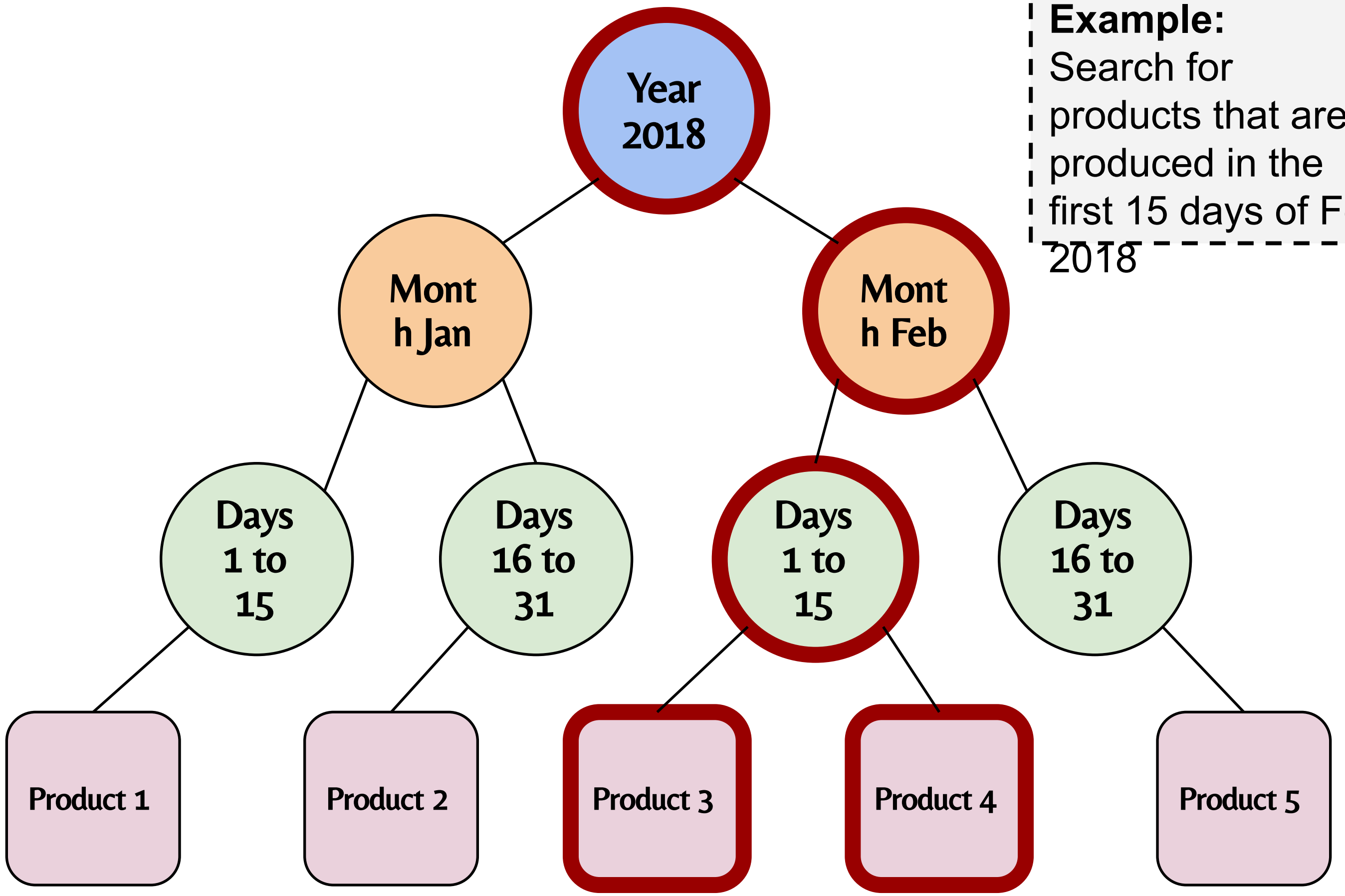
Similar to creating vertices for attributes.

Hierarchical datetime structures can be created for faster time series querying speed.

The levels and partitioning of each level can be customized to best suit your use case.



Traversed vertexes

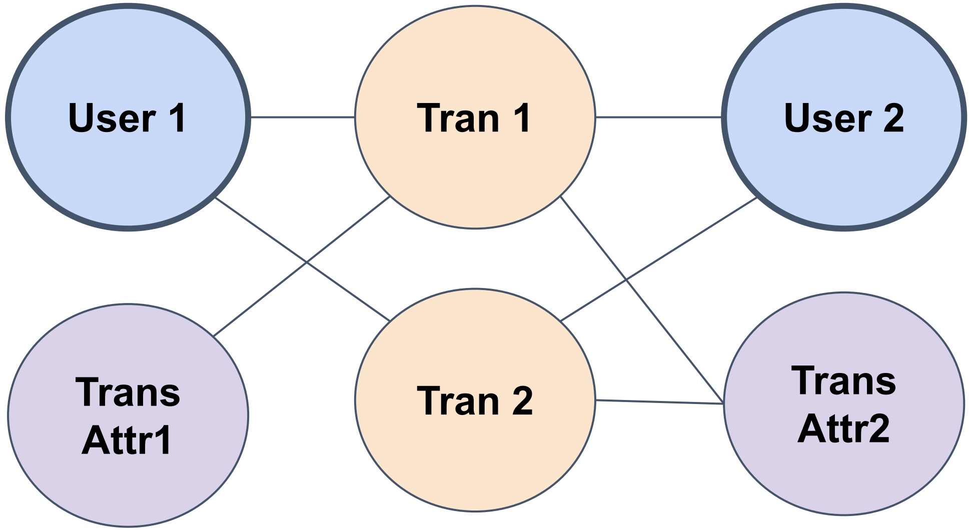


Example:
Search for products that are produced in the first 15 days of Feb 2018

Multiple Events/Transactions Between Two Entities

Method 1: Each Event as a Vertex

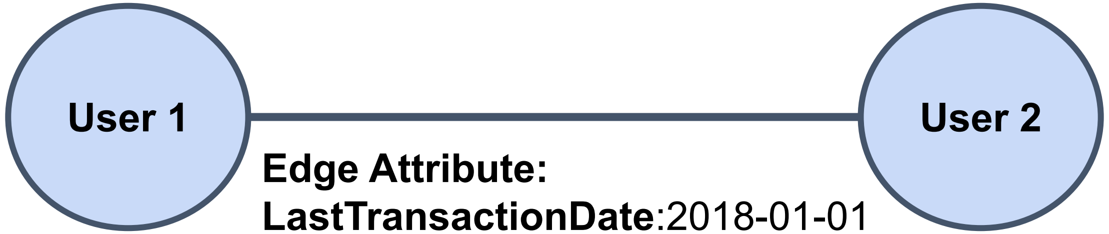
- Create a vertex for each transaction event.
- Connect transactions with the same attributes via attribute vertices.



OR

Method 2: Events aggregated into one Edge

- Connect users who have transactions with a single edge
- Aggregate historical info or use a container to hold a set of values



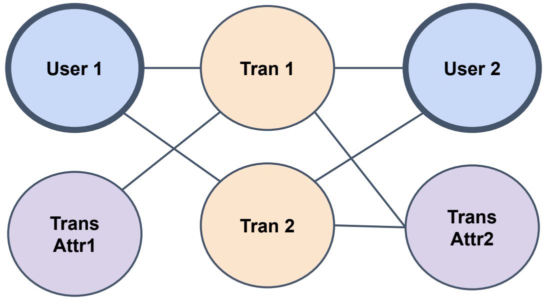
Edge Attribute:
LastTransactionDate:2018-01-01
FirstTransactionDate:2015-08-21
TotalTransactionAmount:12233.23
event_dates:[2018-01-01, 2018-02-01, 2015-08-21]

Multiple Events/Transactions Between Two Entities

- Create vertex for each transaction event.
- Connect transactions with same attribute via attribute vertices.

Pros: Easy to do transaction analytics, such as finding transaction community and similar transactions. Able to do filtering on the transaction vertex attributes.

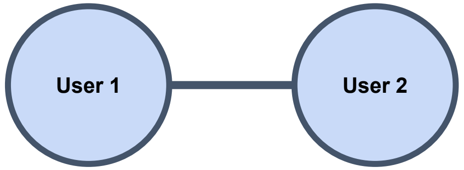
Cons: Uses more memory, takes more steps to traverse between users.



- Connect users who had transactions with a single edge
- Aggregate historical info to edge attributes

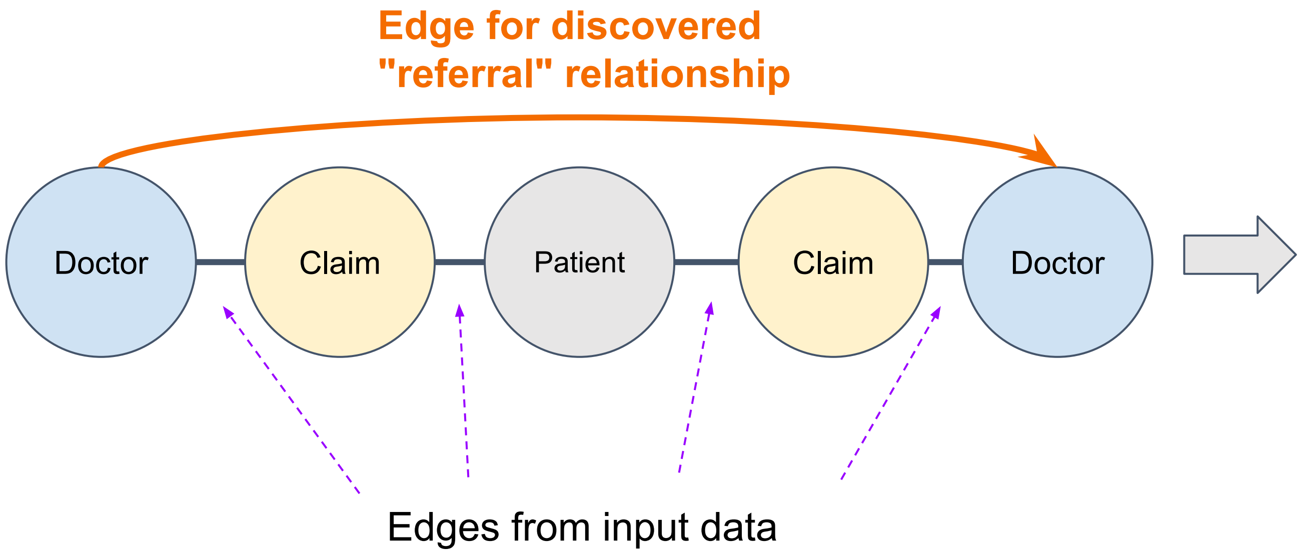
Pros: Significantly less memory usage (if without container). Takes fewer steps to traverse between users.

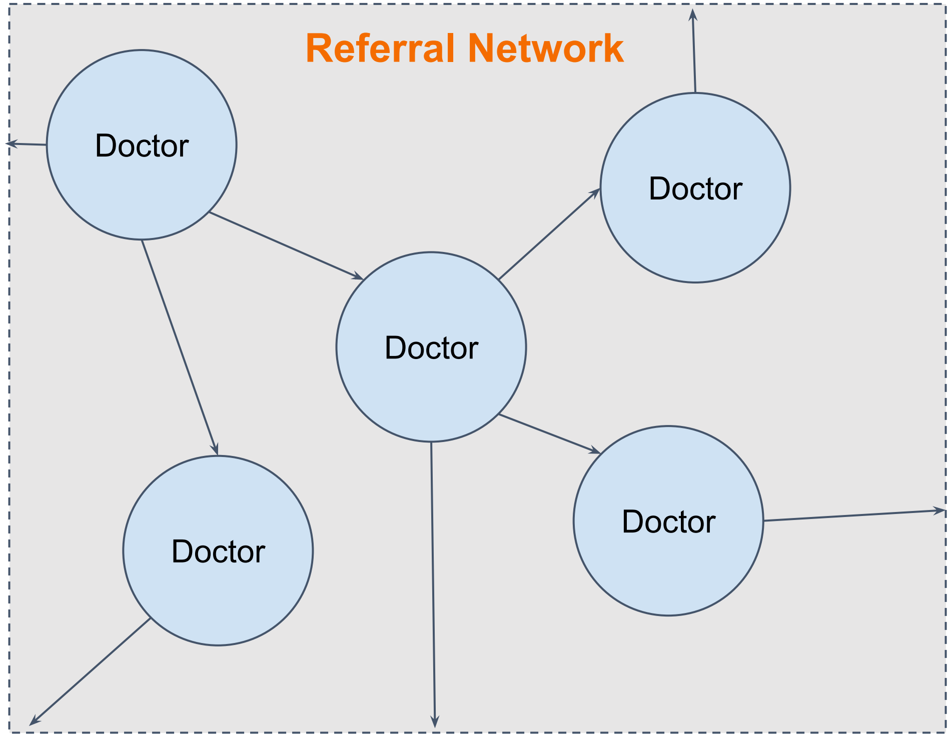
Cons: Searching on transactions is less efficient. Slower update/insert when using a container.



Store query result and support Data Science Library

Relationships that do not exist in the input data can be discovered by running graph analytic algorithms and then adding them to become part of the graph. Update the graph schema to include new relationship types.





TigerGraph

Store query result and support Data Science Library

Schema for utilizing the Data Science Library

1. **Attributes**
   - weight on the edge for weighted PageRank
   - attribute to store the PageRank

2. **Vertex Types**
   - vertex to store the community statistics

3. **Edge Types**
   - edge storing the top k similarity result

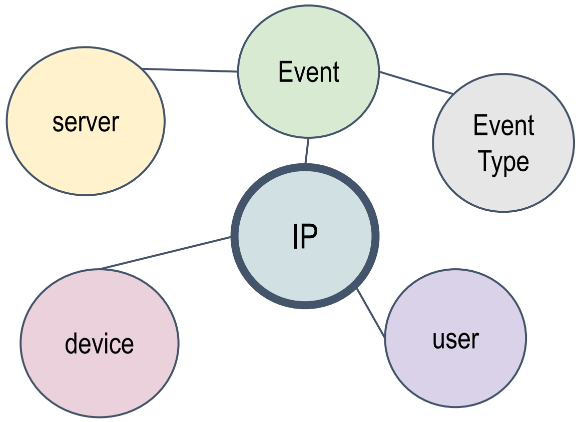
Design Schema Based on Use Case

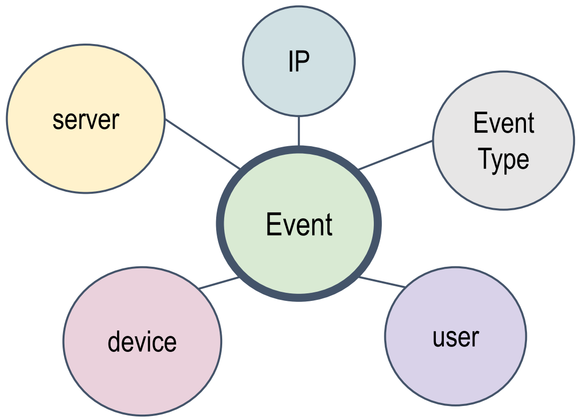
For any given data set, there can be multiple choices for creating a graph schema.

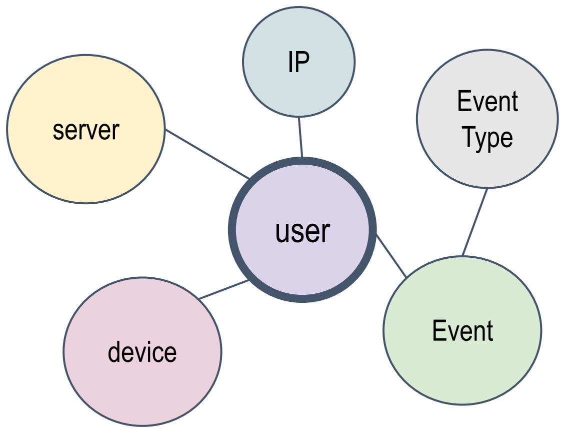
Design the schema that can solve your business problem and provide the best performance.

Event ID	IP	Server	Device	UserId	EventType	Message
001	50.124.11.1	s001	dev001	u001	et1	mmmmmmm
002	50.124.11.2	s002	dev002	u002	et2	mmmmmmm
…	…	…	…	…	…	…

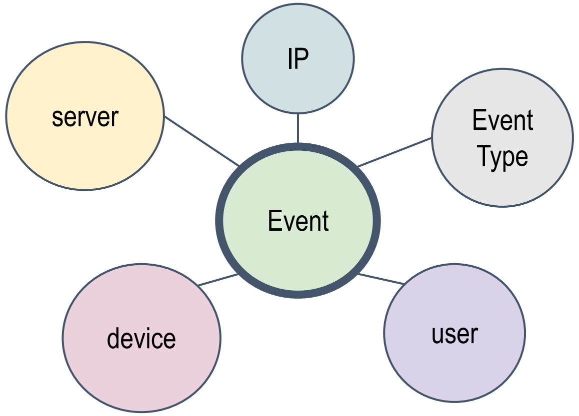
But which one serves **your use case** the best?







Design Schema Based on Use Case: Two Common Styles



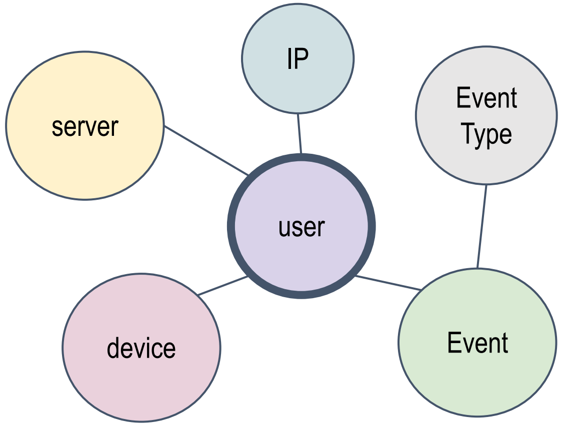
Event-centered schema

Pros: All info of an event is in its 1-hop neighborhood.

Cons: Users are 2 hops away from the device or IP she used

Suitable use cases:

1. Finding communities of events
2. Finding the servers that processed the most events of a given event type
3. Finding the servers visited by a given IP



User-centered schema

Pros: Easy to analyze the connectivities between the users.

Cons: Events are 2 hops away from their related server and IP. It is hard to tell which IP is used for which event.

Suitable use cases:

1. Starting from an input user, detect blacklisted users in k hops.
2. Given a set of blacklisted users, identify the whitelisted users similar to them.
3. Given two input users, are they connected with paths?