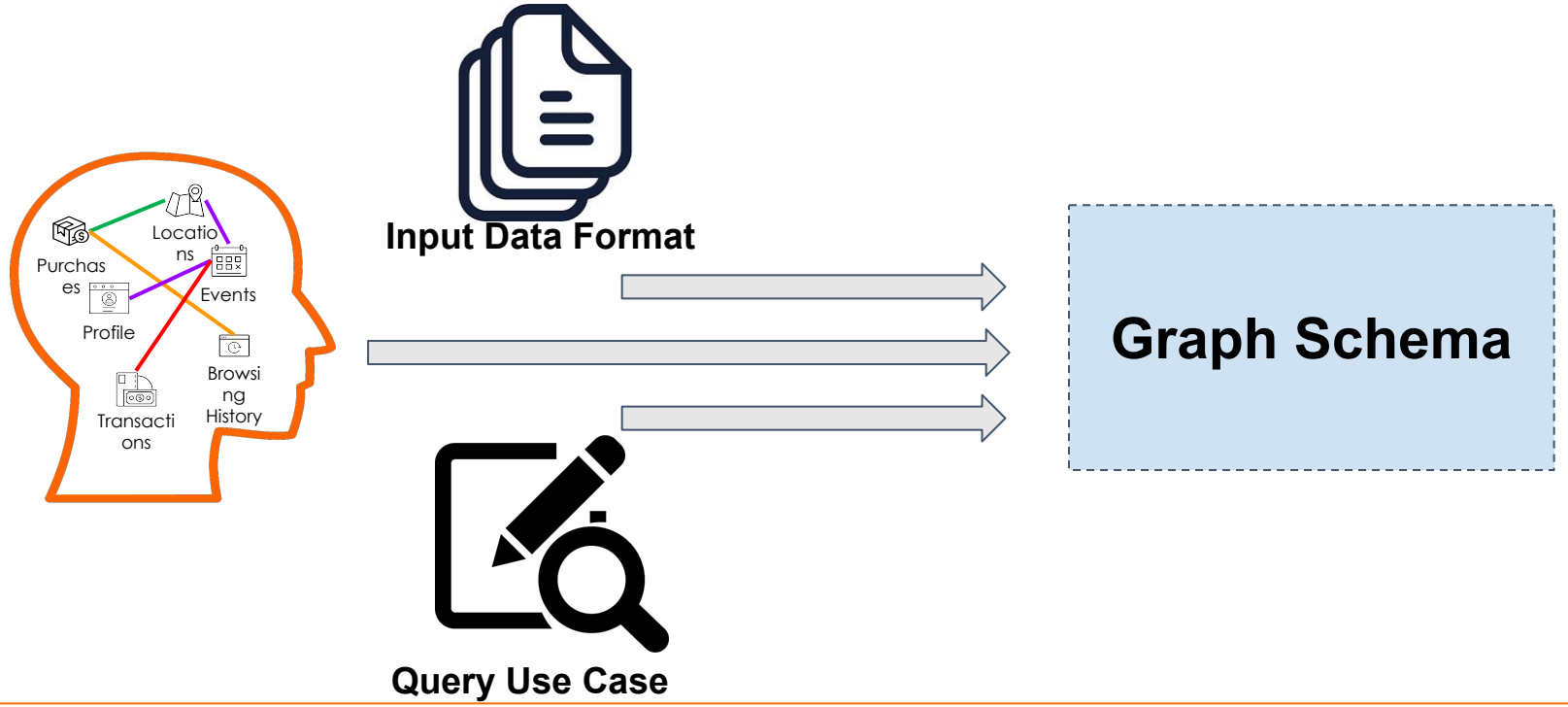

# Schema Design Best Practices

# 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**.








TigerGraph

# Schema Design Topics

1. Rule of Thumb: What can be vertexes? What can be edges?
2. Choosing an Edge Type: Undirected? Directed? Reversed?
3. Attribute or Vertex?
4. How to Model Time in a Graph
5. Multiple Events/Transactions between Two Entities
6. Derived Edges
7. Design Schema Based on Use Case

# 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**

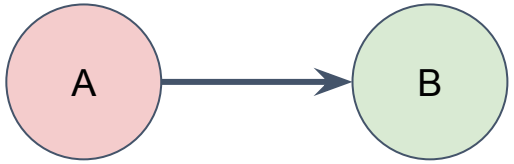
Such as: is_coworker, works_for, is_controlled_by, has_account, purchased_product, has_home_address, belongs_to_type, etc...

# Choosing an Edge Type: Undirected? Directed? Reversed?

## Undirected Edge

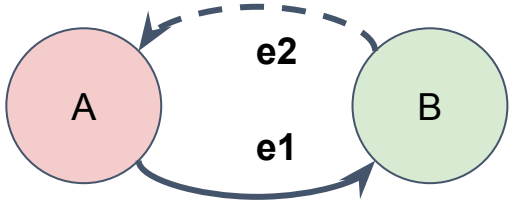

A can traverse to B, and B can traverse to A

## 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?





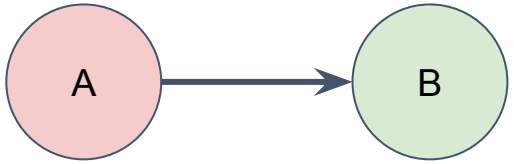
TigerGraph

# Choosing an Edge Type: Undirected? Directed? Reversed?

## Undirected Edge

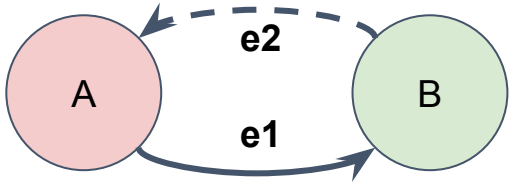

A can traverse to B, and B can traverse to A
**Pros:** Simple when working with undirected or bidirectional relationships.
Example: "A friend_of B" ⇔ "B friend_of A"
**Cons:** Does not carry directional info.

## 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" ⇏ "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.

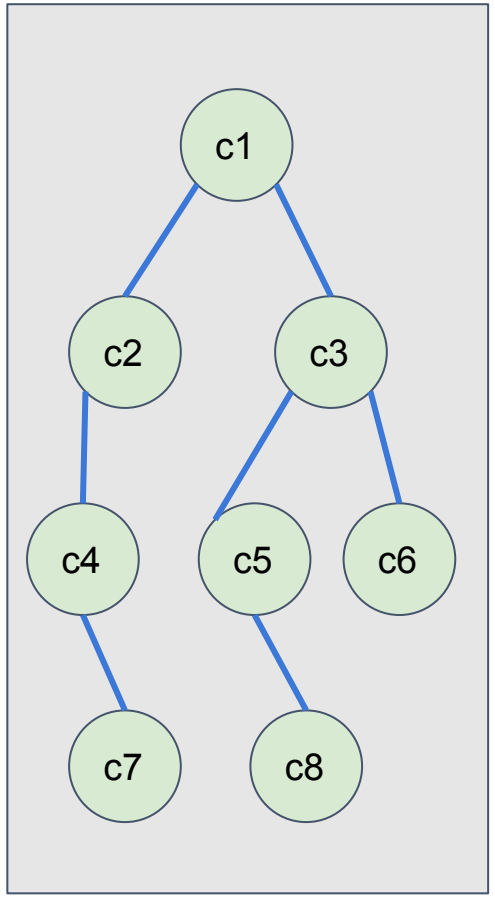


TigerGraph

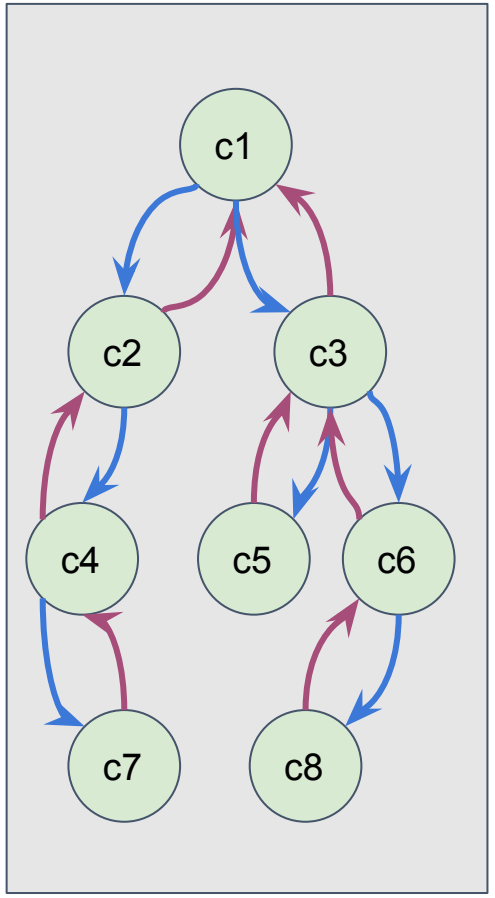
# Choosing an Edge Type: Undirected? Directed? Reversed?

**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?

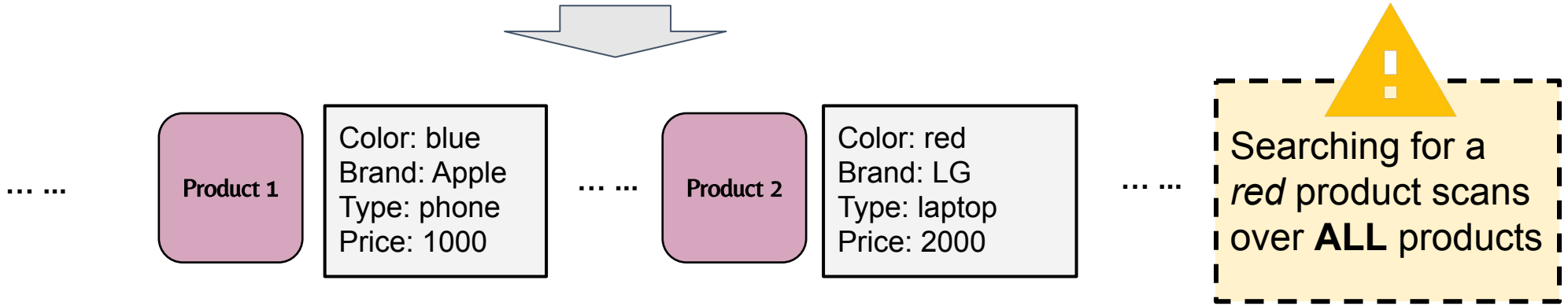
1. A Product **is part of** a PurchaseOrder.
2. An Account **is owned by** one or more Persons.
3. Donors **donate anonymously** to a Charity.
4. A Product **is compatible with** another Product.

# Attribute or Vertex?

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 |
| ... | ... | ... | ... | ... |



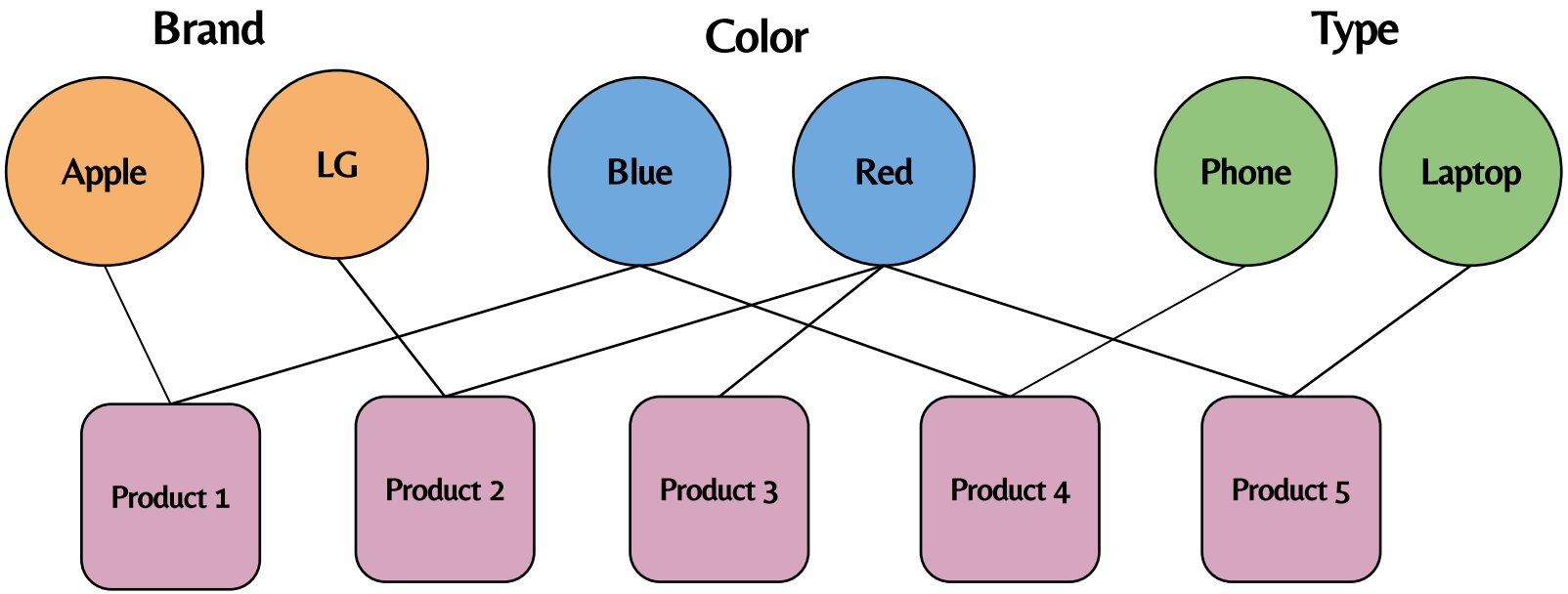





TigerGraph

# Attribute or Vertex?

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.





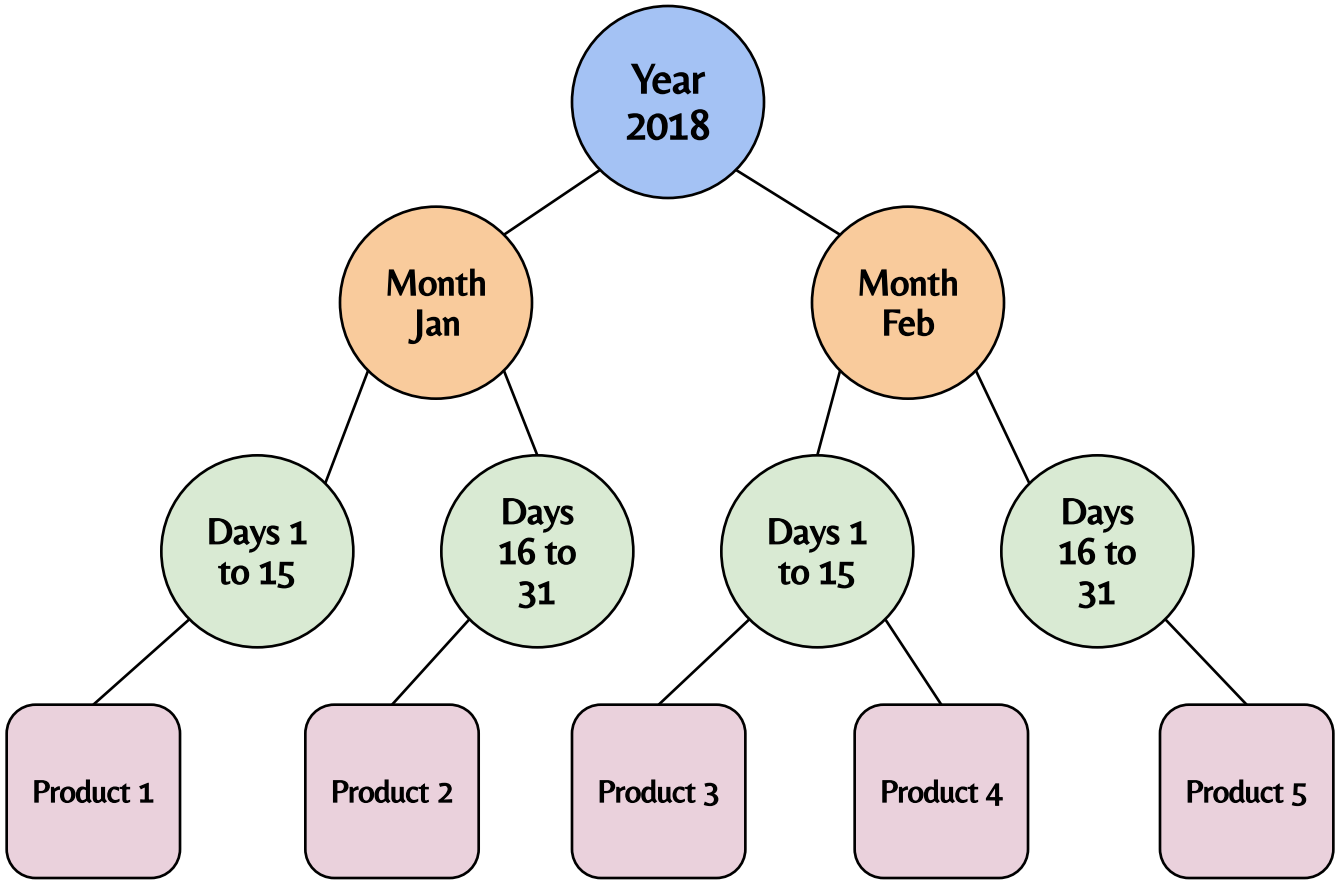

TigerGraph

# 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.








TigerGraph

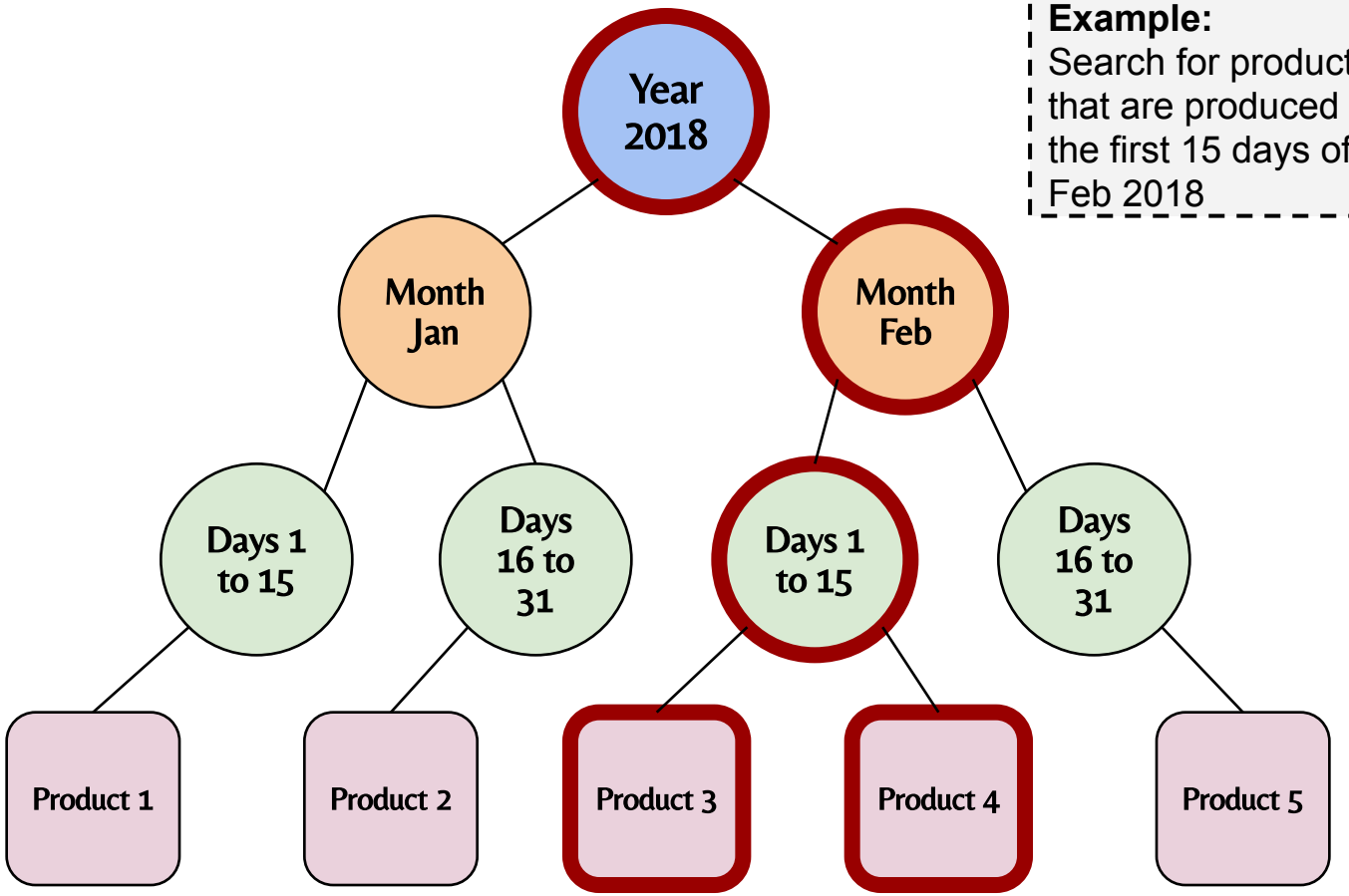
# How to Model Time in a Graph

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



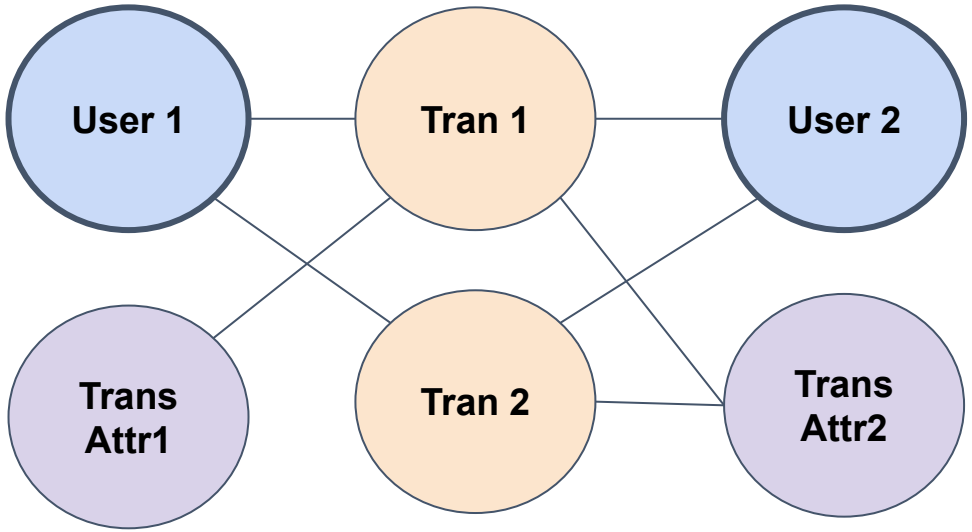


TigerGraph

# Multiple Events/Transactions Between Two Entities

## Method 1: Each Event as a Vertex

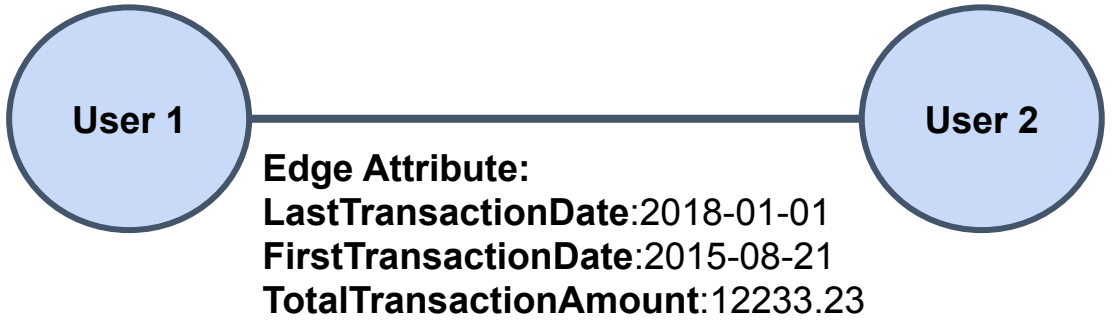
- Create a vertex for each transaction event.
- Connect transactions with the same attributes via attribute vertices.




OR

## Method 2: Events aggregated into one Edge

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







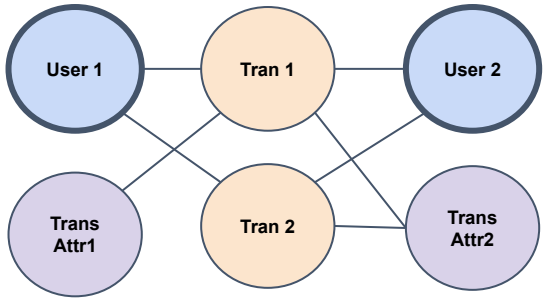

TigerGraph

# 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.

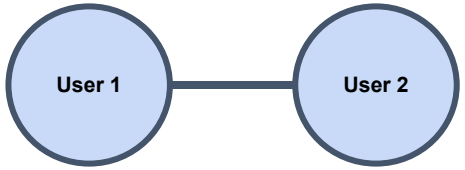
**Cons:** Uses more memory, takes more steps to traverse between users.




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

**Pros**: Significantly less memory usage. Takes fewer steps to traverse between users.

**Cons:** Searching on transactions is less efficient.

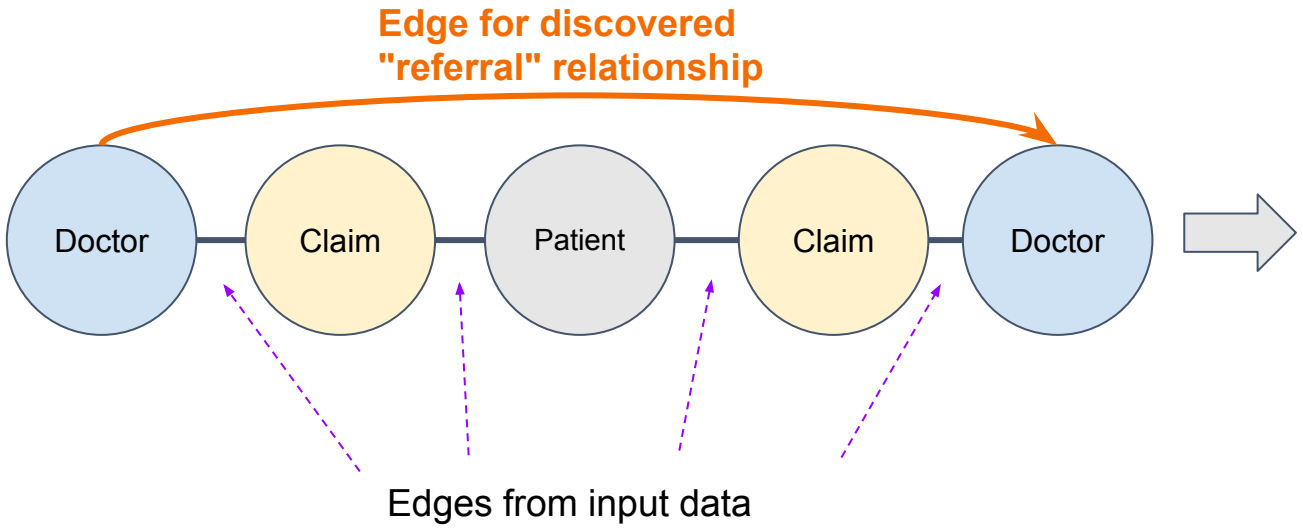


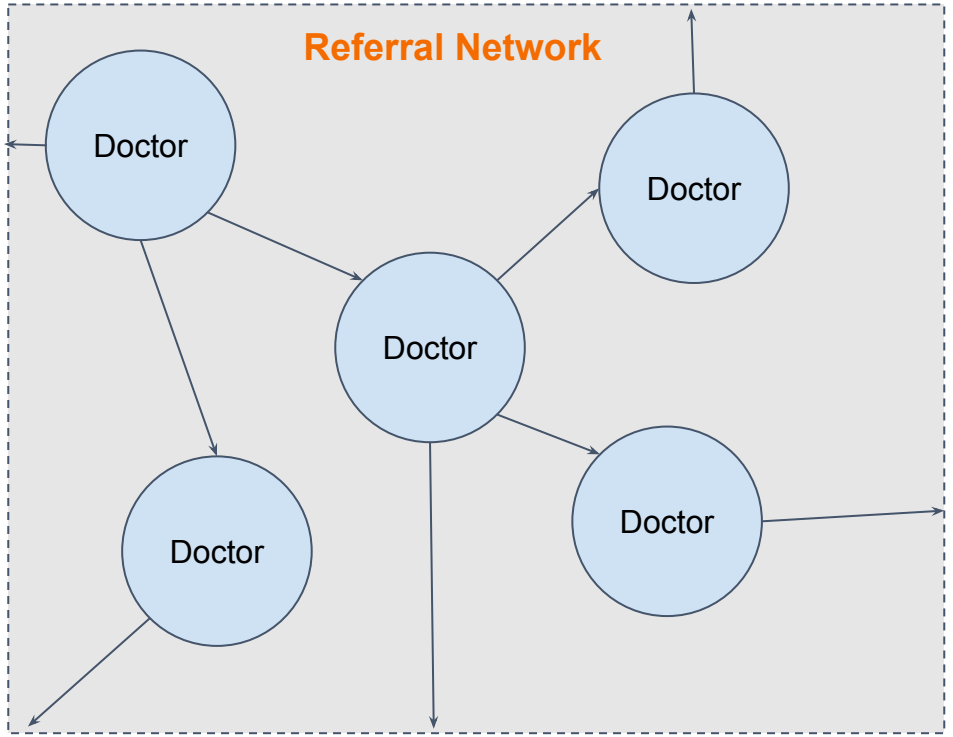



TigerGraph

# Derived Edges

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

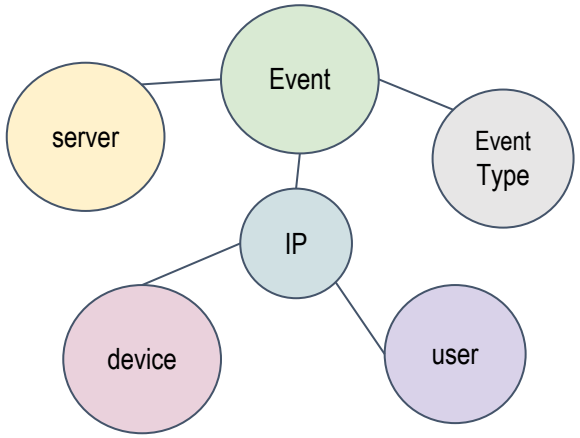
# Design Schema Based on Use Case

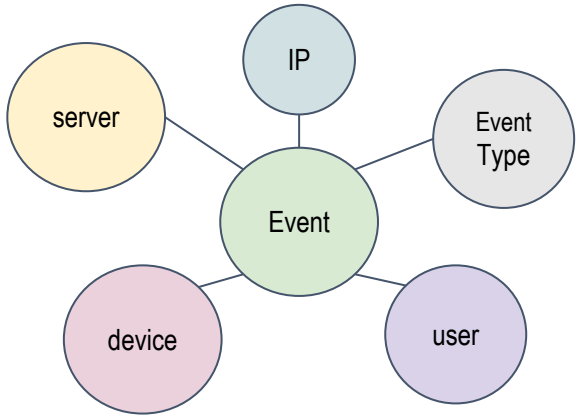
For any given data set, there can be multiple choices for creating a graph schema.

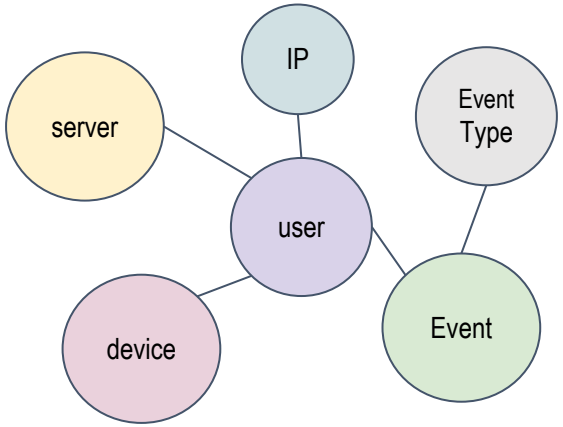
Design the schema that can solve your business problem and provide the best performance.

| 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?



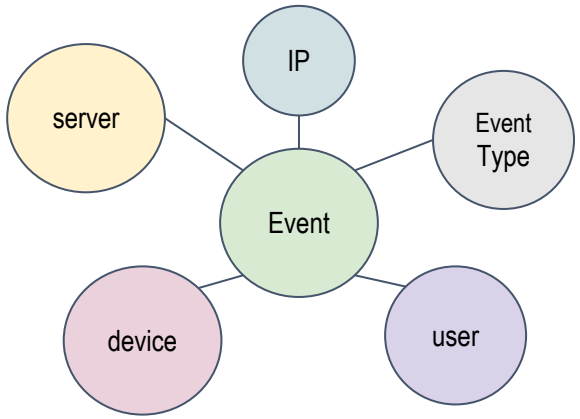








TigerGraph

# Design Schema Based on Use Case: Two Common Styles



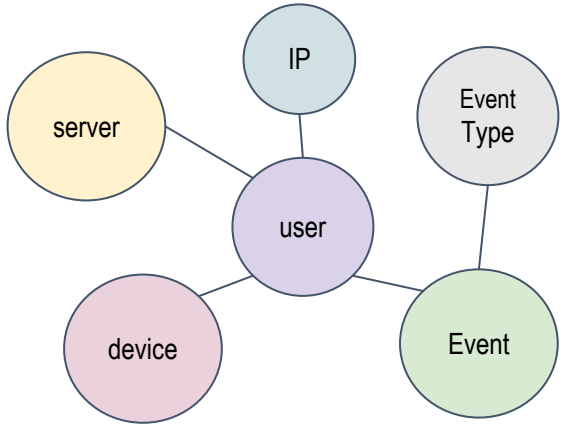

**Event-centered schema**

**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?





TigerGraph