hosted by Alibaba Group 阿里巴巴集团 APACHE HBASE

# HBaseConAsia2018

## JanusGraph —
## Distributed graph database with HBase

XueMin Zhang @ TalkingData

# Content

# Content

# About us

## About me

- Seven years of practical experience in technical research and development(R&D),focusing on distributed storage, distributed computing, real-time computing, etc.
- Successively worked in Sina Weibo and TalkingData, and served as the big data Team Leader of Sina r&d center.
- Technical speechers on the platforms of China Hadoop, Strata Hadoop/Data Conference and DTCC.

## About TalkingData

- Founded in 2011, TalkingData is China’ s leading third-party big data platform. With SmartDP as the core of its data intelligence application ecosystem, TalkingData empowers enterprises and helps them achieve a data-driven digital transformation.
- From the beginning, TalkingData’ s vision of using “big data for smarter business decisions and a better world” has allowed it to gradually become China’ s leading data intelligence solution provider. TalkingData creates value for clients and serves as their “performance partner,” helping modern enterprises achieve data-driven transformation and accelerating the digitization of clients from various industries. Using data-generated insights to change how people see the world and themselves, TalkingData hopes to ultimately improve people’ s lives.

# Content

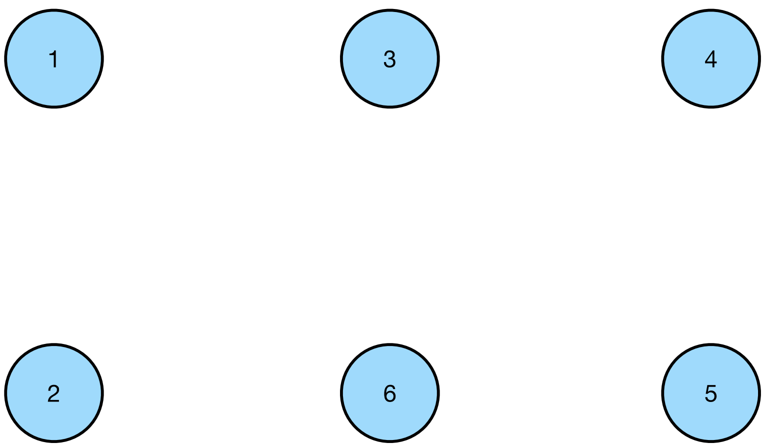
# Something about Graph

What is a Graph Database

- As name suggests, it is a database.
- Uses **graph structures** for semantic queries with **nodes, edges and properties** to represent and store data.
- Allow data in the store to be linked together directly.
- compare with traditional relational databases
  - Hybrid relations.
  - Handy in finding connections between entities.

# Something about Graph

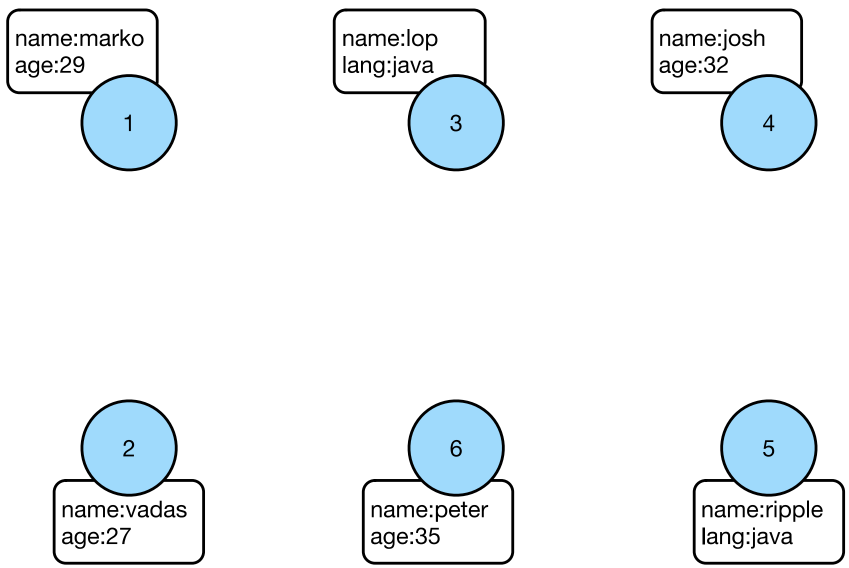
Graph Structures - Vertices




- Vertices are the ***nodes*** or ***points*** in a graph structure
- Every vertex may contain a unique ID.

# Something about Graph

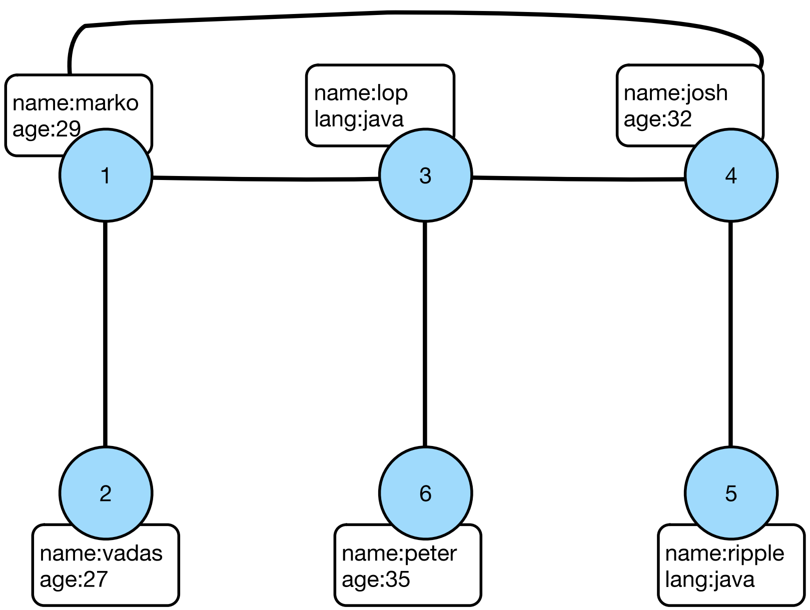
Graph Structures - Vertices




- Vertices are the ***nodes*** or ***points*** in a graph structure
- Every vertex may contain a unique ID.
- Vertices can be associated with a set of ***properties*** (key-value pairs)

# Something about Graph

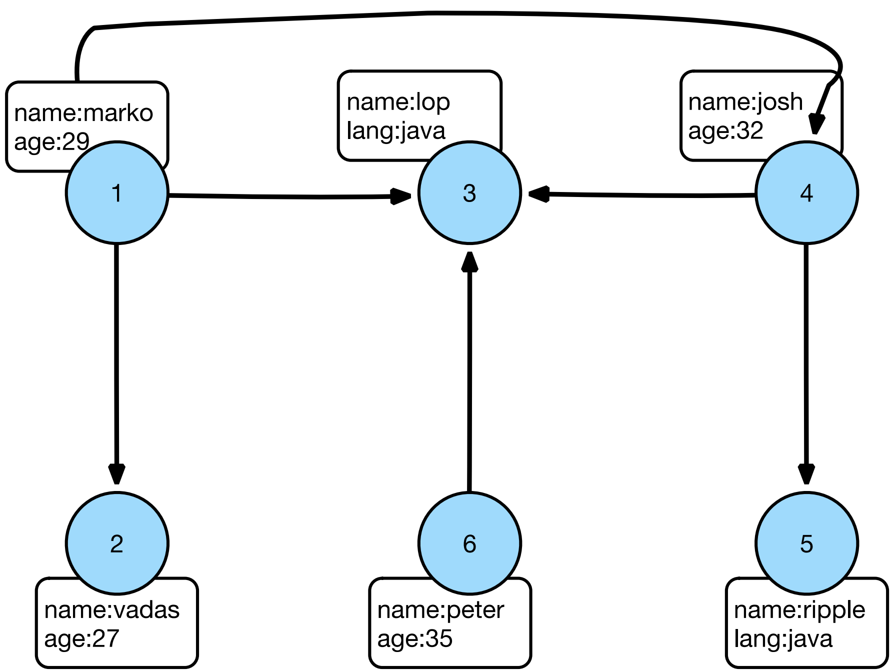
Graph Structures - Edges




- Edges are the ***connections*** between the vertices in a graph

# Something about Graph

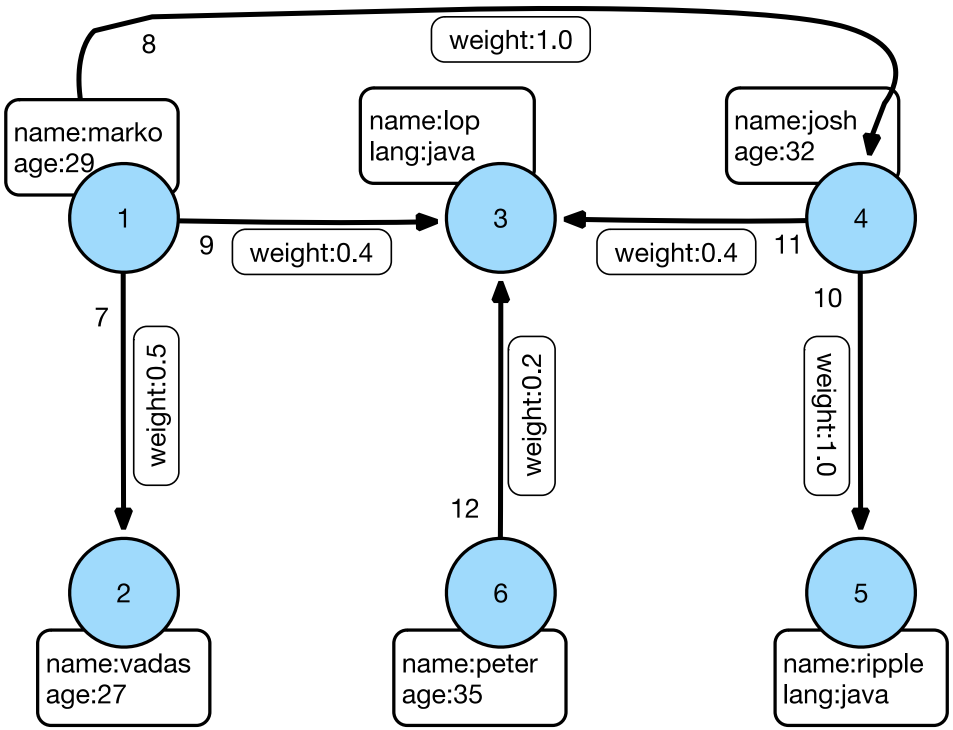
Graph Structures - Edges




- Edges are the ***connections*** between the vertices in a graph
- Edges can be ***nondirectional***, ***directional***, or ***bidirectional***

# Something about Graph

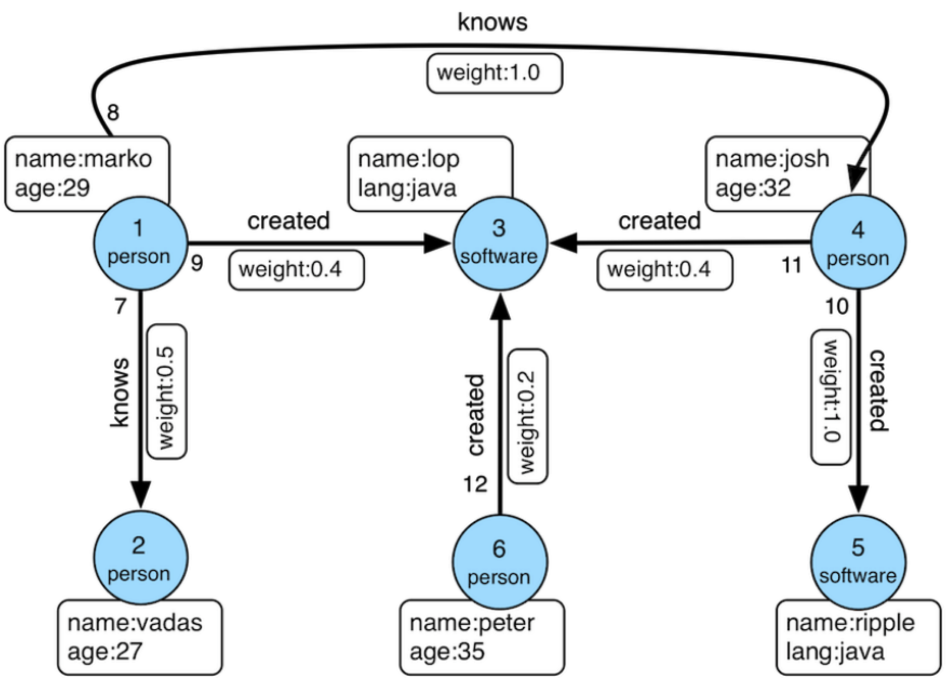
Graph Structures - Edges




- Edges are the ***connections*** between the vertices in a graph
- Edges can be ***nondirectional***, ***directional***, or ***bidirectional***
- Edges like vertices can have ***properties and id***

# Something about Graph

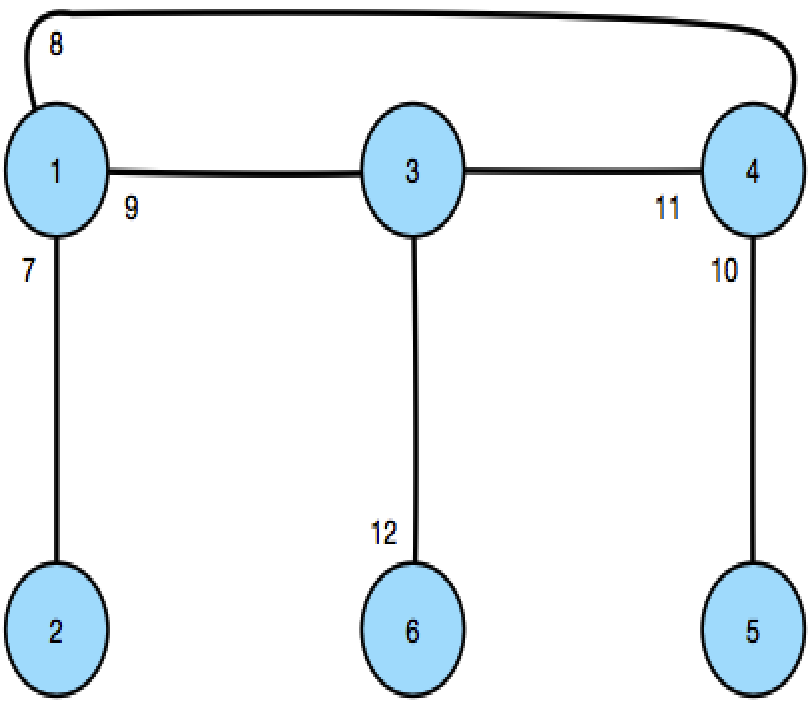
Graph Structures - Graph




- ✧ G = (V, E)
- ✧ The ***graph*** is the collection of ***vertices***, ***edges***, and associated ***properties***
- ✧ Vertices and edges can use label classification

# Something about Graph

Graph Storage Model - Adjacency Matrix




G.vertices =

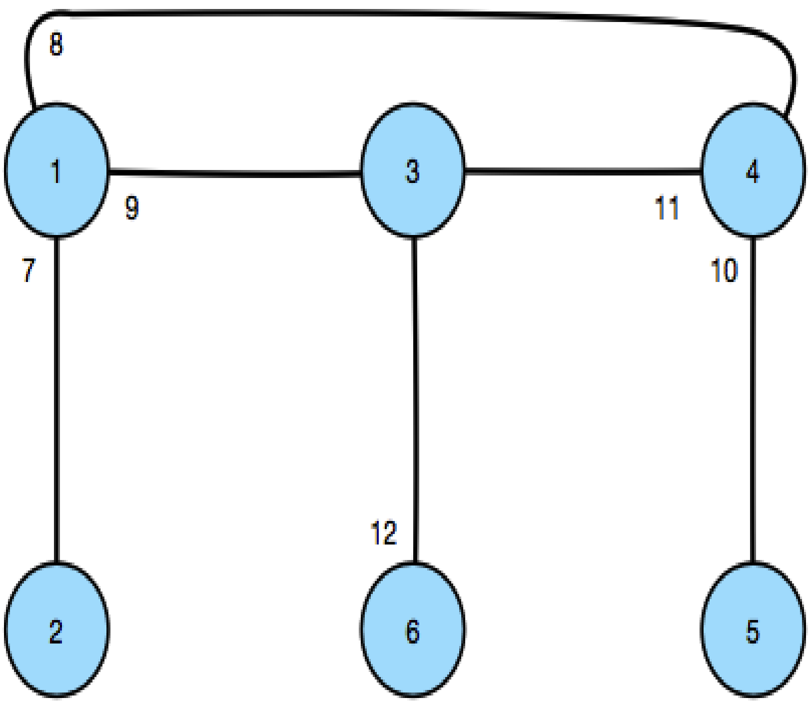
| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |

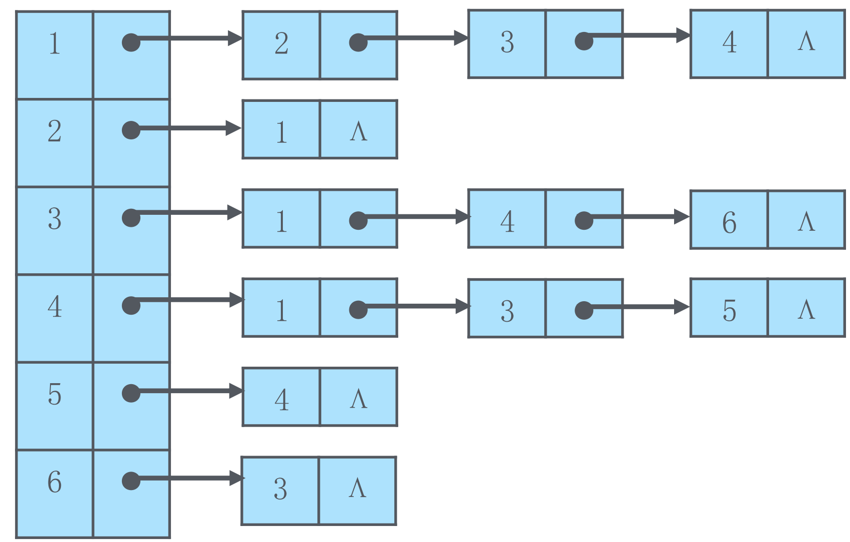
G.edges =

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 0 | 1 | 0 | 0 |
| 4 | 1 | 0 | 1 | 0 | 1 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 0 | 1 | 0 | 0 | 0 |

# Something about Graph

Graph Storage Model - Adjacency Lists



# Content

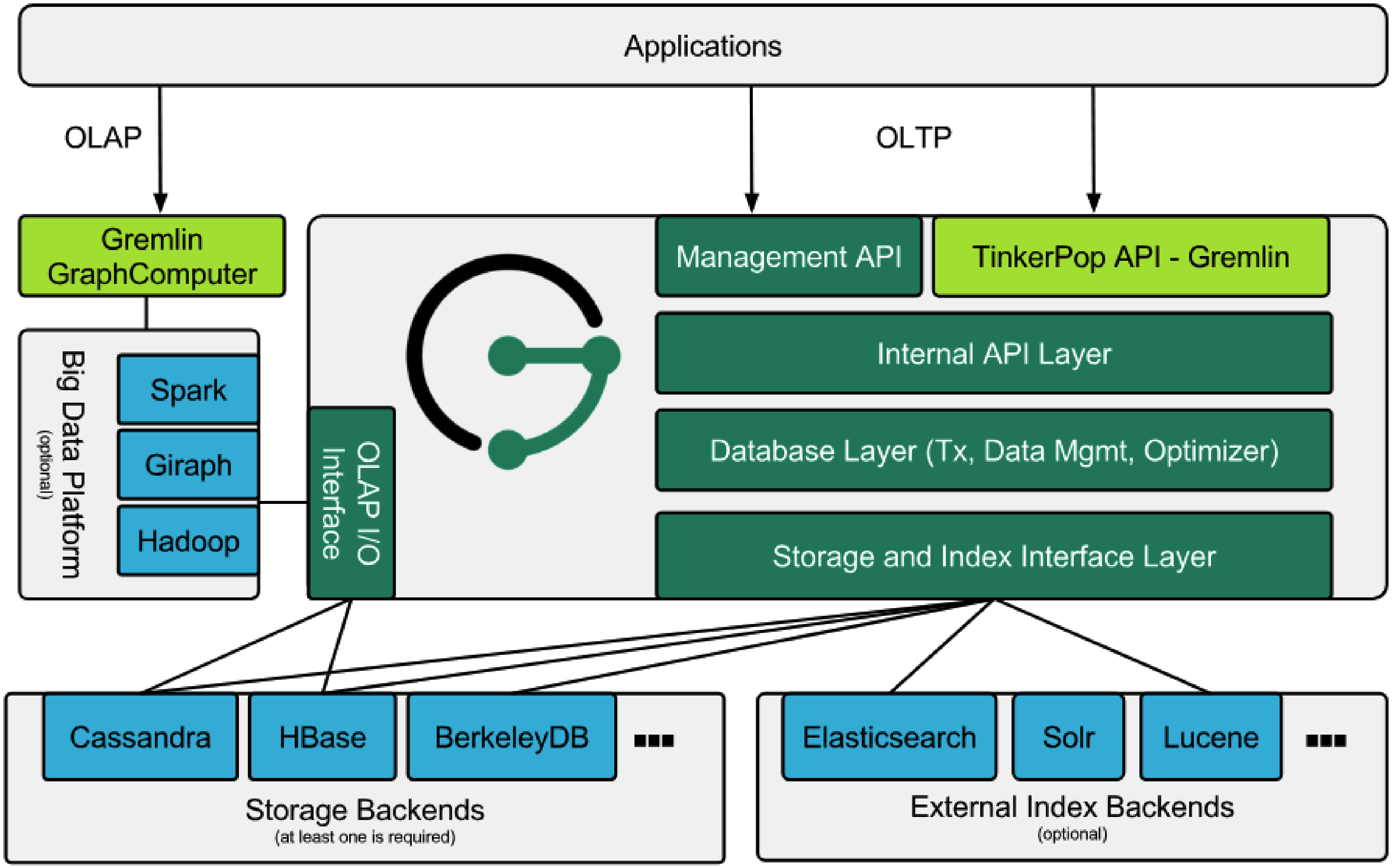
# Introduction to JanusGraph



- ✧ Scalable graph database distribute on multi-maching clusters with pluggable storage and indexing.
- ✧ Fully compliant with Apache TinkerPop graph computing framework.
- ✧ Optimized for storing/querying billions of vertices and edges.
- ✧ Supports thousands of concurrent users.
- ✧ Can execute local queries (OLTP) or cross-cluster distributed queries (OLAP).
- ✧ Sponsored by the Linux Foundation.
- ✧ Apache License 2.0

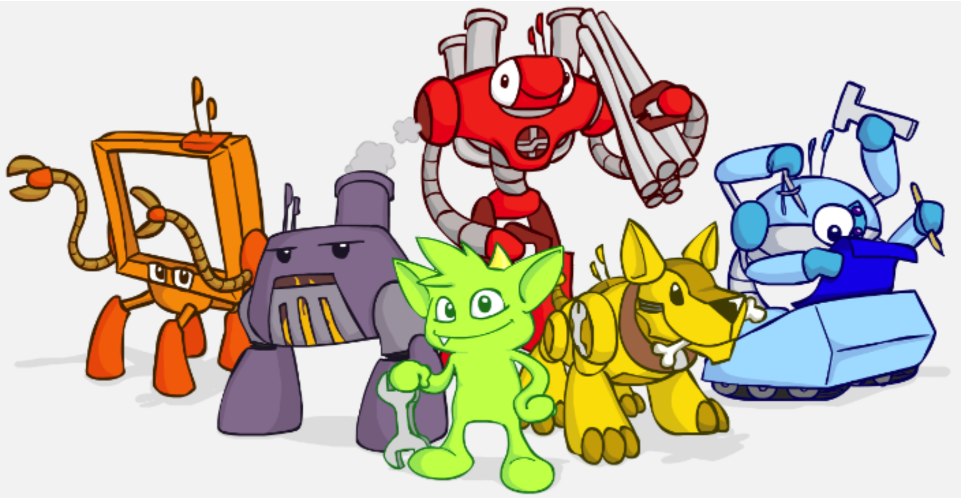
# Introduction to JanusGraph

Architecture



# Introduction to JanusGraph

Apache Tinkerpop & Gremlin



- A graph computing framework for both graph databases (OLTP) and graph analytic systems (OLAP)
- Gremlin graph traversal language

| Operation | Query |
|---|---|
| Single vertex | g.V(4160) |
| Matching a property | g.V().has("name", "Jupiter") |
| Range filtering | g.V().has("age", between(2000, 5000)) |
| To other vertices | g.V().has("name", "Jupiter").out() |
| To edges | g.V().has("name", "Jupiter").outE() |
| Filtering with traversals | g.V().has("name", "Jupiter").out().<br>has("age", between(2000, 5000)) |

# Introduction to JanusGraph



Schema and Data Modeling

- ✧ Consist of edge labels, property keys, vertex labels , index
- ✧ Explicit or Implicit
- ✧ Can evolve over time without database downtime

```
final PropertyKey name = management.makePropertyKey("name").dataType(String.class).make();

management.buildIndex("name", Vertex.class).addKey(name).buildCompositeIndex();

management.makeVertexLabel("location").make();

management.makeEdgeLabel("mother").multiplicity(Multiplicity.MANY2ONE).make();
```

# Introduction to JanusGraph

Schema - Edge Label Multiplicity

- **MULTI**: Multiple edges of the same label between vertices
- **SIMPLE**: One edge with that label (unique per label)
- **MANY2ONE**: One *outgoing* edge with that label
- **ONE2MANY**: One *incoming* edge with that label
- **ONE2ONE**: One incoming, one outgoing edge with that label

# Introduction to JanusGraph



Schema - Property Key Data Types

**Table 5.1. Native JanusGraph Data Types**

| Name | Description |
|---|---|
| String | Character sequence |
| Character | Individual character |
| Boolean | true or false |
| Byte | byte value |
| Short | short value |
| Integer | integer value |
| Long | long value |
| Float | 4 byte floating point number |
| Double | 8 byte floating point number |
| Date | Specific instant in time (`java.util.Date`) |
| Geoshape | Geographic shape like point, circle or box |
| UUID | Universally unique identifier (`java.util.UUID`) |

# Introduction to JanusGraph



Schema - Property Key Cardinality

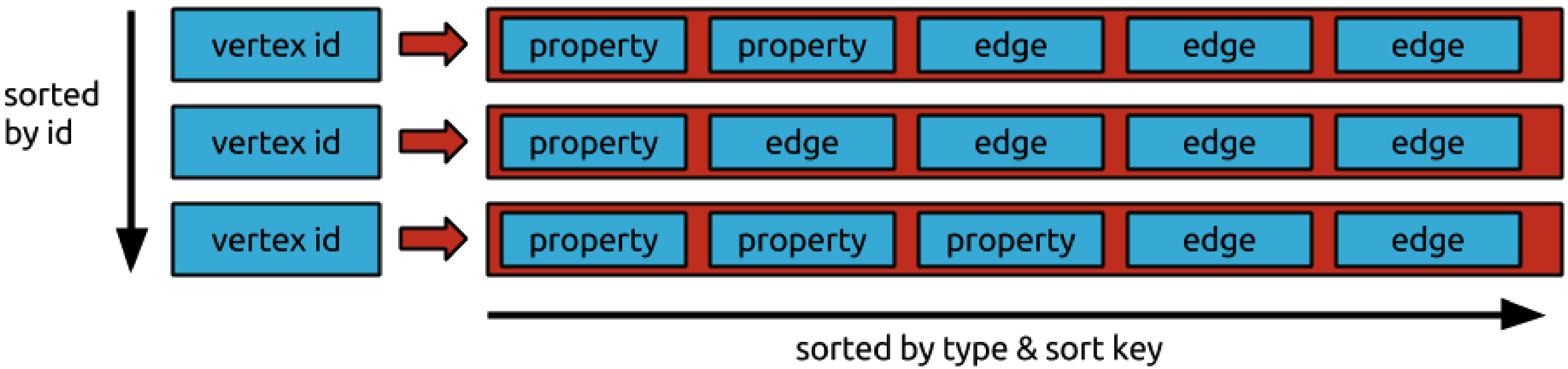
- **SINGLE:** At most one value per element.
- **LIST:** Arbitrary number of values per element. Allows duplicates.
- **SET:** Multiple values, but no duplicates.

# Introduction to JanusGraph

Storage Model



## 38.2. JanusGraph Data Layout



# Introduction to JanusGraph

What is Graph Partitioning?

- When the JanusGraph  cluster consists of multiple storage backend instances, the graph must be partitioned across those machines.
- Stores graph in an adjacency list , ssignment of vertices to machines determines the partitioning.
- Different ways to partition a graph
  - Random Graph Partitioning
  - Explicit Graph Partitioning

# Introduction to JanusGraph

Random Graph Partitioning

- Pros
  - Very efficient
  - Requires no configuration
  - Results in balanced partitions
- Cons
  - Less efficient query processing as the cluster grows
  - Requires more cross-instance communication to retrieve the desired

# Introduction to JanusGraph

Explicit Graph Partitioning

hosted by Alibaba Group 阿里巴巴集团 APACHE HBASE

- Pros
  - Ensures strongly connected subgraphs are stored on the same instance
  - Reduces the communication overhead significantly
  - Easy to setup
- Cons
  - Only enabled against storage backends that support ordered key
  - Hotspot issue

TalkingData

# Introduction to JanusGraph

Edge Cut & Vertex Cut

- Edge Cut
  - Vertices are hosted on separate machines.
  - Optimization aims to reduce the cross communication and thereby improve query execution.
- Vertex Cut (by label)
  - A vertex label can be defined as *partitioned* which means that all vertices of that label will be partitiond across the cluster.
  - In other words, Storing a subset of that vertex’ s adjacency list on each partition .
  - Address the hotspot issue caused by vertices with a large number of incident edges.

# Introduction to JanusGraph



What is Graph Index?

- graph indexes : efficient retrieval of vertices or edges by their properties
  - Composite Index (supported through the primary storage backend)
  - Mixed Index (supported through external indexing backend)
- vertex-centric indexes : effectively address query performance for large degree vertices

hosted by Alibaba Group 阿里巴巴集团 APACHE HBASE

# Content



TalkingData

# JanusGraph with HBase

HBase – Perfect Storage Backend for JanusGraph

- Tight integration with the Apache Hadoop ecosystem.
- Native support for strong consistency.
- Linear scalability with the addition of more machines.
  - Scalability and partitioning
  - Read and write speed
  - Big enough for your biggest graph
- Support for exporting metrics via JMX.
- Great open community

# JanusGraph with HBase

HBase – Perfect Storage Backend for JanusGraph

hosted by Alibaba Group 阿里巴巴集团 APACHE HBASE

- Simple configuration
  - storage.backend=hbase
  - storage.hostname=zk-host1,zk-host2,zk-host3
  - storage.hbase.table=janusgraph
  - storage.port=2181
  - storage.hbase.ext.zookeeper.znode.parent=/hbase

TalkingData

# JanusGraph with HBase

HBase – Perfect Storage Backend for JanusGraph



- A variety of reading and writing way
  - Batch to mutate
  - Get or Multi Get
  - Key range scan
  - ColumnRangeFilter
  - ColumnPaginationFilter

# JanusGraph with HBase

HBase Storage Model - Column Families

- CF attributes can be set. E.g. compression, TTL.
  - Edge store -> e
  - Index store -> g
  - Id store -> i
  - Transaction log store -> l
  - System property store -> s

# JanusGraph with HBase

HBase Storage Model - Edge store -> e



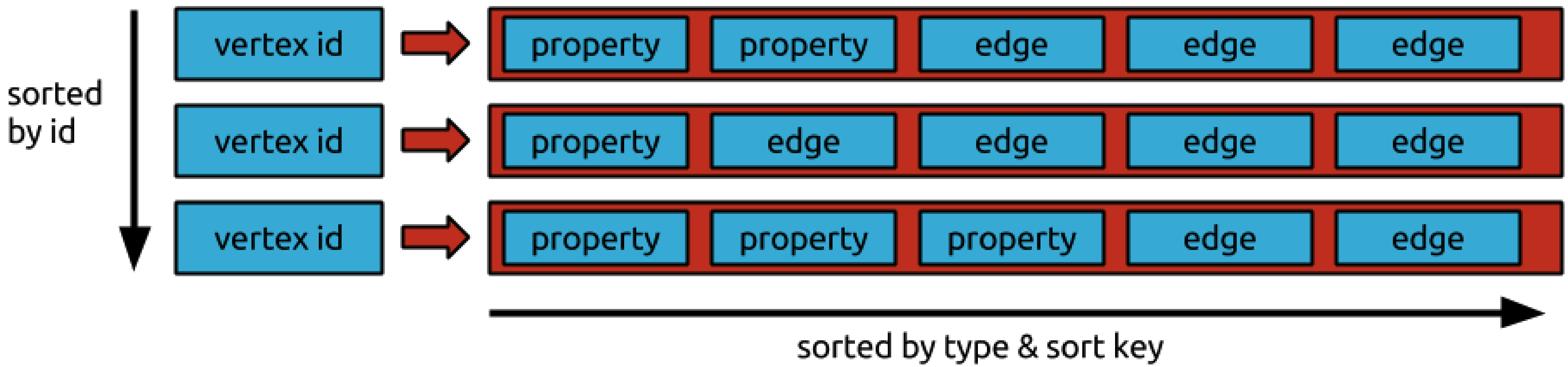
- Storage vertex label, edge, property data
  - RowKey -> Vertex ID
    - Count
    - ID padding
    - Partition ID
  - Vertex label save as edge
  - Vertex property and edge save as relation
    - Relation ID ( Property key id / Edge label id + direction )

# JanusGraph with HBase

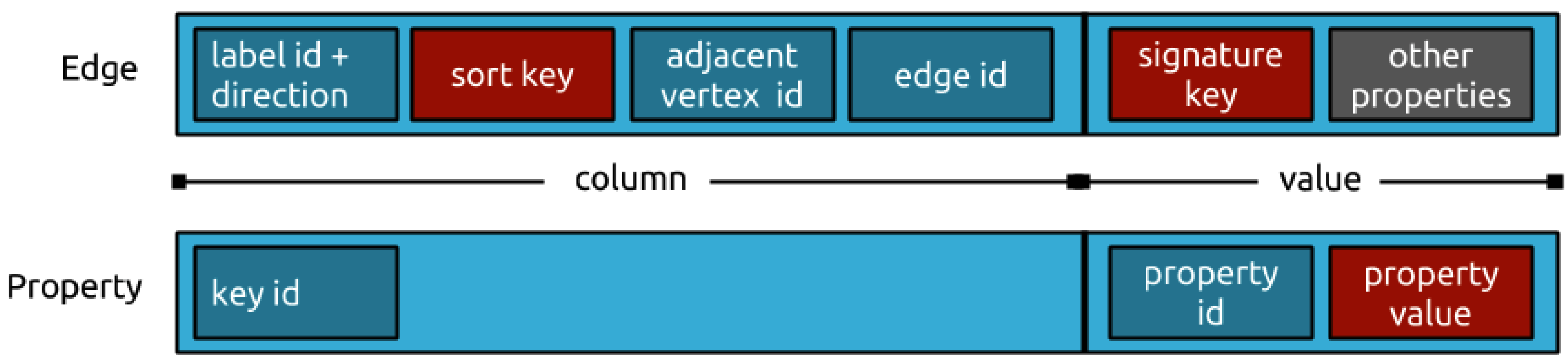
HBase Storage Model -  Edge store -> e



**38.2. JanusGraph Data Layout**




**38.3. Individual Edge Layout**



# JanusGraph with HBase

HBase Storage Model - Index store -> g

- Storage graph indexes (Composite Index) data
  - Rowkey -> property values
  - Cell value->
    - relationId
    - outVertexId
    - typeId
    - inVertexId

# JanusGraph with HBase

Optimization Suggestions

- hbase.regionserver.thread.compaction.large/small
- hbase.hstore.flusher.count
- hbase.hregion.memstore.flush.sizeh
- base.hregion.memstore.block.multiplier
- hbase.hregion.percolumnfamilyflush.size.lower.bound
- hbase.regionserver.global.memstore.size
- hfile.block.cache.size
- hbase.regionserver.global.memstore.size.lower.limit (hbase.regionserver.global.memstore.lowerLimit)
- Random vs. Explicit Partitioning

hosted by Alibaba Group 阿里巴巴集团 APACHE HBASE

# Thanks